Пятое издание
Рассматривает Asterisk 16

O'REILLY®

Asterisk

Полное руководство

Open Source телефония для предприятия

Джим Ван Меггелен,
Рассел Брайант
и Лейф Мэдсен

ПЯТОЕ ИЗДАНИЕ

Asterisk: Полное руководство

Jim Van Meggelen, Russell Bryant and Leif Madsen

Beijing · Boston · Farnham · Sebastopol · Tokyo O'REILLY®

Asterisk: The Definitive Guide

Jim Van Meggelen, Russell Bryant и Leif Madsen

Отпечатано в Соединенных Штатах Америки.
Переведено в Российской Федерации.

Опубликовано O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Книги O'Рейли можно приобрести для образовательных, деловых или рекламных целей. Онлайн-издания также доступны для большинства названий (*http://oreilly.com*). Для получения дополнительной информации свяжитесь с нашим корпоративным/организационным отделом продаж: 800-998-9938 или *corporate@oreilly.com*.

Редактор: Майк Лукидес и Рэйчел Румелиотис

Редактор разработки: Джефф Блейель

Редактор производства: Кристен Браун

Редактор текста: Дуайт Рэмси

Корректор: Рэйчел Монаган

Индексатор: Джудит МакКонвилль

Дизайнер интерьера: Дэвид Футато

Художник обложки: Карен Монтгомери

Иллюстратор: Ребекка Демарест

Перевод: GitHub: krote5k, airosa-id, zrtpsrtp, OilK

Верстка и индексация перевода: Дмитрий Кузнецов

TRANSLASTER

Июль 2019: Пятое издание

История изменений для пятого издания

2019-06-21: Первый релиз

2020-10-16: Первый релиз русского перевода

Смотри https://www.oreilly.com/catalog/errata.csp?isbn=0636920140610 для обзора подробностей.

978-1-492-03160-4

[LSI]

Оглавление

Когда мы рассматривали предисловие к каждому изданию этой книги - у нас всегда было больше людей, от которых мы хотели бы получить вклад, чем страниц, которые мы могли бы выделить. В этом пятом издании мы снова попросили избранную группу людей из сообщества Asterisk написать несколько слов об Asterisk с их точки зрения.

Джошуа Кольп (Старший разработчик программного обеспечения, Sangoma/Digium)

Более 15 лет назад я загрузил Asterisk на свой ноутбук и сделал свой первый VoIP-звонок с помощью IAX2 на АТС Digium. Я затаил дыхание в предвкушении, ожидая услышать голос, пока, наконец, звук Эллисон не раздался из моего ноутбука. В этот момент я понял, что в Asterisk есть что-то особенное. Это зажгло во мне искру интереса и воображения: мой ноутбук действительно сделал звонок! Осознание того, что всего лишь с небольшим усилием я могу принимать звонки и делать с ними то, что захочу, вызывало привыкание и возбуждение — чувство, разделяемое многими и по сей день.

Сегодня Asterisk сильно отличается от того, что было в то время. В прошлом она была в первую очередь ориентирован на то, чтобы быть АТС. Она обладала всеми этими особенностями и продолжала приобретать новые, чтобы продвинуться дальше в эту область. Однако со временем проект эволюционировал до такой степени, что Asterisk - это инструментарий, который можно использовать отдельно или в сочетании с другими проектами для создания вещей. Она существует для того, чтобы вызвать вопрос "Могу ли я это сделать?" - в твоем сознании и позволить тебе увидеть все до конца.

Этот простой вопрос - то, что движет многими решениями, принятыми в отношении Asterisk и ее направления. “Правильно ли это для пользователей?", "Неужели это то, что действительно нужно людям?", “Разве это ломает вещи?" "и “Могут ли они построить то, что они хотят с этим?" Вместе эти вопросы помогают гарантировать, что люди смогут реализовать свои идеи. Что меня сегодня волнует в Asterisk — видеть, как люди используют инструменты для создания чего-то нового без помех.

Я думаю, что в дальнейшем это будет продолжаться и для Asterisk. Она будет продолжать добавлять новые инструменты и функциональные возможности, для обеспечения большей гибкости и возможности для этих строительных вещей, уважая при этом свое наследие и то, как пользователи уже используют его. Она будет продолжать оставаться частью более крупных и лучших решений, некоторые из которых, возможно, даже не придут в голову сейчас. Мы сделали всего несколько шагов вперед и нам еще многое предстоит сделать.

Я призываю новых и старых пользователей Asterisk пересмотреть то, что может сделать Asterisk, изучить новые функции, которые были добавлены, и создать что-то новое и захватывающее из вашего обычного набора навыков. Если вы попали в тупик, где Asterisk не может сделать то, что вам нужно - тогда участвуйте в проекте и вносите свой вклад. Помогите другим, кто, возможно, пытается сделать то же самое. Станьте не просто тем, кто использует Asterisk, но и тем, кто помогает другим реализовать свою мечту.

Дэн Дженкинс (Основатель компании Nimble Ape Ltd)

Asterisk был моей первой вылазкой в мир open source телефонии, и как веб-разработчик я обнаружил, что она сильно отличается от того, к чему я привык, исходя из веб-индустрии. С тех пор проект Asterisk продвинулся вперед, и теперь он включает в себя множество API и технологий, которые типичный веб-разработчик привык ожидать. Включение WebRTC и Asterisk Rest Interface имеет жизненно важное значение для интеграции с разработчиками, используемыми для построения веб-платформы. Asterisk — это то, вокруг чего я в конечном итоге построил свой бизнес - это действительно замечательная часть программного обеспечения, и у нее есть блестящее сообщество людей, которые её используют и улучшают. Мне было очень приятно быть частью этого сообщества и корректировать эту книгу для будущего сообщества.

Джойс Уилмот (Старший веб-разработчик)

Я познакомился с Asterisk в 2012 году, когда работал в Voicenation - компании, предоставляющей прямой автоответчик 24/7/365 для тысяч клиентов. В то время колл-центр быстро перерастал стороннее программное обеспечение, которое они использовали. Не найдя гибкого и экономичного решения для своего быстрорасширяющегося колл-центра, Voicenation решила что им необходимо создать собственное программное обеспечение для колл-центра. Мне была поставлена задача создания такого программного обеспечения, с которой началось мое путешествие с Asterisk. То, что начиналось как монументальная задача (поскольку у меня не было предыдущего опыта IP-телефонии), быстро стало очарованием Asterisk, когда я обнаружил, как она упростила нашу установку, не жертвуя мощностью и гибкостью.

Перенесемся на девять лет вперед и десятки миллионов звонков спустя, а Asterisk по-прежнему верно и надежно управляет нашим колл-центром. Это было мое первое знакомство с программным обеспечением с открытым исходным кодом. Очевидно, что Asterisk - это история успеха open source, которая иллюстрирует, как программное обеспечение с открытым исходным кодом подпитывает предпринимательство — и как предпринимательство, в свою очередь, подпитывает развитие и совершенствование программного обеспечения с открытым исходным кодом. Я очень рад быть частью этого цикла и с нетерпением жду возможности стать частью сообщества, поскольку Asterisk постоянно развивается, чтобы идти в ногу с постоянно меняющимся миром телекоммуникаций.

Мэтт Florell (Основатель VICIdial)

Мое первое знакомство с open source телефонией еще в 2001 году было на самом деле не с Asterisk. Оно состоялось с другим программным пакетом и заняло у меня пару месяцев, чтобы начать работать, используя простой IVR для регистрации запросов на контакт с моим работодателем, в то время. Это была не простая система для работы или модификация, поэтому я не делал с ней ничего другого, кроме того первого проекта IVR. Два года спустя я получил от одного клиента просьбу построить гораздо более сложную телефонную систему, которая требовала бы взаимодействия пользователей через компьютер. Я знал, что платформа, которую я использовал, не будет работать для такого проекта, поэтому я посмотрел на коммерческие и открытые варианты. Именно тогда я узнал об Asterisk, которая выглядела как идеальная платформа для этого проекта. Я купил карту T1, с помощью которой можно было провести некоторые тесты и в течение двух часов после ее прибытия я настроил ее и смог воспроизвести старый проект, на создание которого у меня ушло два месяца. После этого я попался на крючок. Проект VICIdial Open-Source Contact Center вырос из этого проекта; на сегодняшний день более 100 000 систем Asterisk были установлены в составе кластеров VICIdial, и это только те, о которых мы знаем.

Asterisk сильно отличался от большинства веб-пакетов с открытым исходным кодом, с которыми я работал в прошлом, и у него было довольно много причуд и ошибок в более ранние дни, которые нам приходилось обходить (иногда довольно творчески). Но более поздний опыт работы с веткой Asterisk 13 показал значительные улучшения как в производительности, так и в стабильности по сравнению с более ранними ветками. Кроме того, было добавлено много новых функций, позволивших нам добавить новые функциональные возможности в наш пакет VICIdial. Две из них - это возможность

приостанавливать записи вызовов и добавление нескольких уровней регистрации нового носителя SIP.

Еще в 2003 году, когда я начал использовать Asterisk, настоящих “релизов" не было. Вы должны были найти стабильную сборку из одной из последних версий CVS и протестировать ее. С течением времени развитие и обслуживание различных отраслей стало намного более стабильным, а использование Asterisk в производственных системах по всему миру резко возросло. Сегодня Asterisk - это телефонное ядро тысяч различных предложений услуг, через которое ежедневно проходят миллиарды телефонных звонков. Оно устанавливается на самые разнообразные аппаратные средства, от крошечных встроенных систем до серверных ферм с сотнями мощных машин. В настоящее время миллионы людей, ежедневно использующих Asterisk, понятия не имеют, что они взаимодействуют с частью программного обеспечения с открытым исходным кодом.

Только среди нашей клиентской базы есть несколько компаний из списка Fortune 500, а также школьные округа, общественные клубы, политические организации, муниципальные службы экстренной помощи и, конечно же, тысячи различных видов коммерческих организаций. В то время как низкая стоимость приобретения является распространенной причиной для использования решения на основе Asterisk, мы часто слышим, что тот факт, что это открытый исходный код - является большим плюсом, а также отсутствие возможности блокировки источника. Один из наших крупных клиентов даже назвал использование программного обеспечения для телефонии с открытым исходным кодом “явным стратегическим преимуществом” по сравнению с конкурентами из-за гибкости систем и их способности самостоятельно управлять ими без необходимости полагаться на внешних источников. Судя по тому, что я видел до сих пор, будущее Asterisk - это постоянно растущая установленная база и постоянные усовершенствования. Я с нетерпением жду возможности поработать с ним еще как минимум 16 лет.

Мэтт Фредриксон (Директор Asterisk Engineering, Sangoma/Digium)

У меня была возможность работать с Asterisk в течение последних 18 лет и я видел, как он вырос из небольшого проекта с одним или двумя людьми в нечто, что имеет свою собственную жизнь с сотнями участников. Удивительно видеть, сколько разных мест он нарушил традиционные телекоммуникации - дома, в офисе и на предприятии. По мере того как традиционные модели коммуникации меняются, проект Asterisk продолжает оставаться там, где он делает это лучше всего — наводя мосты между старыми формами коммуникации и новыми и раздвигая границы того, что можно сделать с новыми. Эта книга поможет вам увидеть самое современное лицо Asterisk и узнать как лучше использовать его в своей телекоммуникационной инфраструктуре. Огромное спасибо Джиму Ван Меггелену за всю тяжелую работу по составлению этого самого последнего издания.

Это книга для всех, кто использует Asterisk.

Asterisk - это платформа конвергентной телефонии с открытым исходным кодом, предназначенная в первую очередь для работы на Linux. Asterisk объединяет более чем 100-летние знания в области телефонии в надежный набор тесно интегрированных телекоммуникационных приложений. Сила Asterisk заключается в её настраиваемой природе, дополненной непревзойденным соответствием стандартам. Ни одна другая управленческая автоматизированная телефонная станция (УАТС) не может быть развернута таким множеством творческих способов.

Такие приложения, как голосовая почта, конференции, очереди вызовов и агенты, музыка на удержании и парковка вызовов - все это стандартные функции, встроенные прямо в программное обеспечение. Кроме того, Asterisk может интегрироваться с другими бизнес-технологиями таким образом, о котором закрытые, проприетарные УАТС вряд ли могут мечтать.

Asterisk может показаться довольно пугающей и сложной для нового пользователя, поэтому документация так важна для её роста. Документация снижает барьер для входа и помогает людям созерцать возможности.

Выпущенный при щедрой поддержке O'Reilly Media, Asterisk: Полное руководство - это пятое издание того, что ранее называлось Asterisk: Будущее телефонии.

Эта книга была написана для участников сообщества Asterisk.

Аудитория

Эта книга предназначена для того, чтобы быть доброжелательной к тем, кто новичок в Asterisk, но мы предполагаем, что вы знакомы с базовым администрированием Linux, сетью и другими ИТ-дисциплинами. Если нет - мы рекомендуем вам изучить обширную и замечательную библиотеку книг, которые O'Reilly публикует по этим темам. Мы также предполагаем, что вы пока новичок в телекоммуникациях (как традиционная коммутируемая Телефония, так и новый мир Voice over IP).

Однако эта книга будет полезна и более опытному администратору Asterisk. Мы сами используем книгу в качестве ссылки на функции, которые не использовали в течение некоторого времени.

Программное обеспечение

Эта книга сосредоточена на документировании версии 16 Asterisk; однако многие соглашения и большая часть информации в этой книге являются агностическими версиями. Linux - это операционная система, в которой мы запускали и тестировали Asterisk, и мы задокументировали инструкции по установке для CentOS (Red Hat Enterprise Linux или RHEL).

Условные обозначения используемые в книге

В этой книге используются следующие типографские условные обозначения:

Курсив

Обозначает новые термины, URL-адреса, адреса электронной почты, имена файлов, расширения файлов, пути, каталоги и имена пакетов, а также утилиты Unix, команды, модули, параметры и аргументы.

`Моноширинный`

Используется для отображения примеров кода, содержимого файлов, взаимодействий командной строки, команд базы данных, имен библиотек и параметров.

`Моноширинный полужирный`

Обозначает команды или другой текст, который должен быть буквально набран пользователем. Также используется для акцентирования в коде.

`Моноширинный курсив`

Показывает текст, который должен быть заменен пользовательскими значениями.

`[ Ключевые слова и прочие вещи ]`

Указывает необязательные ключевые слова и аргументы.

`{ вариант-1 | вариант-2 }`

Означает выбор между `вариант-1` или `вариант-2`.

Этот элемент означает подсказку или предложение.

Этот элемент обозначает общую заметку.

Этот элемент указывает на предупреждение или предостережение.

Онлайн обучение O’Reilly

На протяжении почти 40 лет O'Reilly Media предоставляет технологии и бизнес-тренинги, знания и понимание, чтобы помочь компаниям добиться успеха.

Наша уникальная сеть экспертов и новаторов делится своими знаниями и опытом с помощью книг, статей, конференций и нашей онлайн-обучающей платформы. Платформа онлайн-обучения o'Reilly предоставляет вам по запросу доступ к живым учебным курсам, углубленным учебным путям, интерактивным средам кодирования и обширной коллекции текста и видео от O'Reilly и более чем 200 других издателей. Для получения дополнительной информации, пожалуйста, посетите *http://oreilly.com*.

Как с нами связаться

Пожалуйста, направляйте комментарии и вопросы, касающиеся этой книги, издателю:

- O'Reilly Media, Inc.
- 1005 Gravenstein Highway North
- Sebastopol, CA 95472

- 800-998-9938 (в США или Канаде)
- 707-829-0515 (международный или местный)
- 707-829-0104 (факс)

У нас есть веб-страница для этой книги, где мы перечисляем ошибки, примеры и любую дополнительную информацию. Вы можете получить доступ к этой странице по адресу *https://oreil.ly/asterisk_tdg_5E*.

Чтобы прокомментировать или задать технические вопросы об этой книге, отправьте электронное письмо по адресу *bookquestions@oreilly.com*.

Для получения дополнительной информации о наших книгах, курсах, конференциях и новостях смотрите наш веб-сайт по адресу *http://www. oreilly.com*.

Facebook: *http://facebook.com/oreilly*

Следуйте за нами в Twitter: *http://twitter.com/oreillymedia*

Смотрите нас на YouTube: *http://www.youtube.com/oreillymedia*

Благодарности от Джима Ван Меггелена

Дэвиду Даффетту спасибо за главу об интернационализации, в которой он правильно рассматривает эту технологию с более глобальной точки зрения.

Спасибо Лейфу Мэдсену, Джареду Смиту и Расселу Брайанту за Ваш вклад в предыдущие издания этой книги. Это было весело летать в одиночку, но я не могу отрицать, что скучал по вам, ребята!

Особая благодарность Мэтту Фредриксону и Мэтту Джордану из Digium, которые щедро делились со мной своим временем и знаниями, и без которых я был бы потерян. Спасибо, ребята!

Спасибо моему редактору Джеффу Блейлу за то, что он держит меня в курсе событий и помогает принимать важные решения о содержании и темпе работы над книгой.

А также благодаря остальным невоспетым героям из производственного отдела О'Рейли. Это те люди, которые берут книгу и делают ее книгой О'Рейли.

Особенно я благодарен Джойс Уилмот и Дэну Дженкинсу, моей команде технического рецензирования за то, что они нашли время для работы над книгой и предоставили необходимую обратную связь.

Томас Камерон из RedHat щедро поделился со мной своими знаниями о Selinux и помог демистифицировать компонент Linux, который слишком часто остается отключенным.

Все участники сообщества Asterisk также должны поблагодарить покойного Джима Диксона за создание первых аппаратных интерфейсов телефонии с открытым исходным кодом, начало революции и предоставление его творений сообществу в целом.

Наконец, и это самое главное, спасибо Марку Спенсеру, оригинальному автору Asterisk и основателю Digium, за Asterisk, за Pidgin, а также за вклад его творений в сообщество с открытым исходным кодом. Asterisk - это ваше наследие!

Глава 1. Революция в телефонии

Мы - то, за чем они растут. Это истинное бремя всех мастеров.
– Джедай Магистр Йода

Когда мы впервые решили в 2004 году написать книгу об Asterisk (15 лет назад, считая от этого издания!), мы уверенно предсказывали, что Asterisk кардинально изменит телекоммуникационную отрасль. Сегодня революция, которую мы предсказали, является частью истории. Asterisk уже несколько лет является самой успешной управленческой автоматической телефонной станцией (УАТС) в мире и является признанной технологией в телекоммуникационной отрасли.

Революция, столь же необходимая, как и для телекоммуникационной индустрии того времени-значительно затихла просто потому, что методы, с помощью которых люди любят общаться, изменились. В то время как 25 лет назад телефонные звонки были предпочтительным способом общения на расстоянии, нынешняя тенденция заключается в отправке сообщений или проведении видеоконференций. Телефонный звонок воспринимается как нечто мертвое, особенно будущими поколениями. Мы еще не совсем готовы к похоронам.

Asterisk остается мощной технологией, и мы считаем, что она по-прежнему является одной из лучших надежд на какую-либо разумную интеграцию между телекоммуникационными и всеми другими технологиями, с которыми могут захотеть соединиться компании. Ему необходимо будет найти свое место в коммуникационной экосистеме, которая больше не ставит телефонные звонки в важное место. Мы ожидаем, что WebRTC, который обещает коммерциализировать веб-коммуникации[1], появится в качестве замены для всех подражателей, закрытых и проприетарных продуктов “сотрудничества”, которые в настоящее время наводняют (и запутывают) рынок. Asterisk может сыграть свою роль в этом новом будущем, и сообщество Asterisk охотно и с энтузиазмом взяло на себя эту новую концепцию. Итак, может быть, вам говорят, что голос мертв, но любой, кто обратил внимание на любую научную фантастику любого рода, знает, что возможность разговаривать друг с другом на больших расстояниях не будет единственной областью тех, кто печатает на клавиатурах. Люди любят поговорить, и мы будем продолжать искать способы сделать это.

Следует отметить, что существует также огромное поколение людей, чьи воспоминания предшествуют интернету, и для этих людей телефон по-прежнему является очень полезной технологией. Если кто-то хочет иметь с ними дело - ему лучше хорошо справляться с телефонными звонками. Эти люди уходят с работы, но их кошельки все еще имеют много влияния. Возможно, АТС - это умирающая вещь, но ее хвост очень длинный.

В этой книге мы собираемся исследовать гайки и болты Asterisk. Это гибкий, открытый, отвечающий стандартам инструментарий, который, по нашему мнению, все еще очень актуален для бизнеса сегодня и будет оставаться полезным в течение многих лет. Сила Asterisk заключается в его гибкости. Он оказался очень полезным при связывании различных типов коммуникационных технологий вместе, и если он хочет иметь какое-либо будущее - ему нужно будет продолжать это делать. Новые технологии, такие как WebRTC, предлагают всяческие возможности для будущего общения, и сообщество Asterisk очень сосредоточено на этом сдвиге парадигмы.

Замечательная гибкость Asterisk имеет свою цену: она не так проста для изучения или настройки. Это не потому, что она нелогична, запутана или загадочна; напротив, это очень разумно и практично. Глаза людей загораются, когда они впервые видят диалплан Asterisk и начинают осознавать возможности. Но когда есть буквально тысячи способов достижения результата, процесс, естественно, требует дополнительных усилий. Это можно сравнить со строительством дома:

1 И, возможно, еще, учитывая, что WebRTC также революционизирует нативные приложения!

компоненты относительно легко понять, но человек, рассматривающий такую задачу, должен либо а) заручиться компетентной помощью, либо б) развить необходимые навыки посредством обучения, практики и хорошей книги по этому вопросу.

Asterisk и VoIP: преодоление разрыва между традиционной и сетевой телефонией

Иногда кажется, что мы забыли, что цель телефона - позволить людям общаться. На самом деле это простая цель, и мы должны иметь возможность сделать это гораздо более гибкими и творческими способами, чем доступны нам в настоящее время. Новые технологии всегда стремятся доминировать на рынке с помощью собственного предложения. Мало кому это удается. Коммуникационные технологии должны взаимодействовать, и такие технологии как Asterisk, снижают барьеры для входа для тех, кто хочет внедрять инновации.

Именно по этой причине — коммуникация — мы считаем, что будущее все еще существует для проектов телефонии с открытым исходным кодом, таких как Asterisk. Да, люди могут больше не хотеть делать “телефонные звонки”, но мы считаем, что в разговорах все еще будет ценность. Технологии, которые могут облегчить эти разговоры, иногда развиваются, казалось бы, радикальными способами, но основное желание общаться остается тем же самым.

Asterisk подключен к будущему, и у него есть длинный послужной список успешной интеграции коммуникационных технологий.

Проект телефонии Zapata

Когда проект Asterisk был запущен (в 1999 году), существовали и другие проекты телефонии с открытым исходным кодом. Тем не менее, Asterisk в сочетании с проектом Zapata Telephony смогла обеспечить интерфейсы телефонной сети общего пользования (ТфОП), что стало важной вехой в переходе программного обеспечения от чего-то чисто сетевого к чему-то более практичному в мире телекоммуникаций в то время, когда он был ориентирован на ТфОП.

Проект телефонии Zapata был задуман Джимом Диксоном, инженером-консультантом по телекоммуникациям, который был вдохновлен невероятными достижениями в скорости процессора, которые компьютерная индустрия теперь воспринимает как должное. Диксон считал, что гораздо более экономичные системы телефонии можно было бы создать, если бы существовала карта, на которой не было бы ничего, кроме основных электронных компонентов, необходимых для взаимодействия с телефонной линией. Вместо того, чтобы иметь дорогие компоненты на карте, цифровая обработка сигналов (DSP)[2] будет обрабатываться в процессоре с помощью программного обеспечения. Хотя это наложило бы огромную нагрузку на процессор, Диксон был уверен, что низкая стоимость процессоров по сравнению с их производительностью делает их гораздо более привлекательными, чем дорогие DSP, и, что более важно, что соотношение цены и производительности будет продолжать улучшаться по мере увеличения мощности процессоров.

Как и многие провидцы, Диксон верил, что многие другие увидят эту возможность и ему просто нужно подождать, пока кто-то другой создаст то, что для него было очевидным улучшением. Через несколько лет он заметил, что не только никто не создавал эти карты, но и казалось маловероятным, что кто-то когда-нибудь собирался. В тот момент было ясно, что если он хочет революции, то должен начать ее сам. Так родился проект телефонии Zapata:

> Поскольку эта концепция была настолько революционной, что наверняка вызвала бы много волн в индустрии, я выбрал мексиканский революционный мотив и назвал технологию и организацию в честь знаменитого мексиканского революционера Эмилиано Сапаты. Я решил назвать карту "tormenta“, что по-

2 Термин DSP также означает цифровой сигнальный процессор, который представляет собой устройство (обычно чип), способное интерпретировать и изменять сигналы различных видов. В голосовой сети DSP в первую очередь отвечают за кодирование, декодирование и перекодирование аудиоинформации. Это может потребовать много вычислительных усилий.

испански означает ”шторм”, но в контексте оно обычно используется для обозначения большого шторма, такого как ураган или что-то подобное.

Возможно, нам следует называть себя Asteristas. Несмотря на это, мы должны поблагодарить Джима Диксона, частично за то, что он придумал это и частично за то, что довел до конца, но в основном за то, что дал результаты своих усилий сообществу с открытым исходным кодом. В результате вклада Джима появился двигатель ТфОП Asterisk. И благодаря этому сочетанию VoIP и ТфОП родилась телекоммуникационная революция с открытым исходным кодом!

За прошедшие годы интерфейс телефонии Zapata в Asterisk был изменен и улучшен. Телефония Digium Asterisk Hardware Device Interface (DAHDI), используемая сегодня, является детищем вклада Джима Диксона.

Массовые изменения требуют гибкости технологий

Каждая существующая АТС страдает от недостатков. Независимо от того, насколько полнофункциональной она является, что-то всегда будет упущено, потому что даже самая многофункциональная АТС не сможет предвидеть творчество клиента. Небольшая группа пользователей захочет иметь странную маленькую функцию, о которой проектная группа либо не думала, либо не могла оправдать стоимость её создания, и, поскольку система закрыта - пользователи не смогут создать ее сами.

Если бы интернет был таким образом затруднен регулированием и коммерческими интересами - сомнительно, что он получил бы широкое признание, которым пользуется в настоящее время. Открытость интернета означала, что любой желающий мог позволить себе принять в нем участие. Так что все так и сделали. Десятки тысяч умов, которые сотрудничали в создании интернета, принесли то, что ни одна корпорация в одиночку никогда не смогла бы получить[3].

Как и во многих других проектах с открытым исходным кодом (как Linux, так и множество важного программного обеспечения, работающего в интернете), разработка Asterisk была подпитана мечтами людей, которые знали, что должно быть что-то большее, чем то, что производят традиционные отрасли. Эти люди знали, что если бы можно было взять лучшие части различных АТС и разделить их на взаимосвязанные компоненты - подобно коробке с кирпичиками LEGO, можно было бы начать понимать вещи, которые не переживут традиционный корпоративный процесс анализа рисков.

Сама Asterisk стала основой многих массово производимых творений. И все же, под капотом, душа этого проекта остается с открытым исходным кодом.

Asterisk: хакерская УАТС

Asterisk является основной АТС хакера. Термин *хакер*, был искажен средствами массовой информации в значении “злостный взломщик” для необразованных. Это прискорбно, потому что термин фактически существовал задолго до того, как СМИ испортили его значение. Хакеры создали сетевой движок, который стал интернетом. Хакеры создали Apple Macintosh и операционную систему Unix. Хакеры также строят свою следующую телекоммуникационную систему. Да, некоторые из этих людей являются злостными, но умы, которые управляют разработкой Asterisk, хорошо знают об этом, и вы обнаружите, что Asterisk позволяет создавать систему, которая способна довольно быстро реагировать на угрозы безопасности. Программное обеспечение с открытым исходным кодом не скрывает свои недостатки за корпоративными отделами. Грязь вытаскивают на открытое место, где с ней можно справиться. Вместо того чтобы ограничиваться сомнительной и часто плохой безопасностью закрытых систем - сообщество Asterisk быстро реагирует на меняющиеся тенденции в области безопасности, и вы сможете точно настроить свою телефонную систему в соответствии с корпоративной политикой и лучшими отраслевыми практиками.

3 Мы понимаем, что технология интернета сформировалась из государственных и академических институтов, но то, о чем мы здесь говорим - это не столько технология интернета, сколько культурный феномен его, который взорвался в начале 90-х годов.

Как и другие системы с открытым исходным кодом, Asterisk сможет превратиться в гораздо более безопасную платформу, чем любая проприетарная система, несмотря на свои хакерские корни, а скорее даже благодаря им.

Asterisk: профессиональная УАТС

Asterisk - это технология поддержки, и, как и в случае с Linux, все реже можно найти предприятие, которое не запускает какую-либо версию Asterisk, в каком-то качестве в своей сети, решая проблему так, как может только Asterisk. Вы уже используете Asterisk, даже если об этом не знаете.

Сообщество Asterisk

Нет смысла ходить вокруг да около: сообщество Asterisk - это тень его прежнего "я". Десять лет назад Asterisk была самой крутой вещью с открытым исходным кодом. Сегодня большинство энтузиастов двинулись дальше. Однако остается опытное и проверенное в боях сообщество профессионалов, которые были там и сделали это.

Не ждите от команды людей, готовых бесплатно работать над вашими проектами. Цена вступления в это сообщество - личная заинтересованность в развитии навыков. Если вы обратитесь к сообществу с наглостью - вам не понравятся ответы. Однако если вы проявите любопытство, энтузиазм и желание погрузиться в работу, запачкать руки и заняться ею, то найдете сообщество, более чем готовое поделиться с вами своими трудами и добытыми знаниями.

Ниже приведены некоторые из мест, где обитает сообщество Asterisk.

Дискусионный сайт сообщества Asterisk

В 2015 году Asterisk переместила свои официальные форумы на https://community.asterisk.org/. Похоже, что это самое активное сообщество сейчас, и отношение сигнал/шум отличное. Сотрудники Digium хорошо справляются с этой задачей, и некоторые из их старших и опытных людей принимают активное участие.

Имейте в виду, что это не похоже на платную поддержку. От вас ожидают, что вы будете выполнять работу самостоятельно, но вы можете рассчитывать на получение хороших качественных советов, которые направят вас в правильном направлении.

Списки рассылки Asterisk

Активность в этих списках сократилась до минимума (с сотен сообщений в день до, возможно, дюжины потоков в месяц). Они, вероятно, наиболее полезны в качестве исторического архива, но, возможно, стоит туда обратиться когда вы сталкиваетесь с неразрешимой проблемой. Списки рассылки вы найдете по адресу lists.digium.com, эти два, вероятно, будут наиболее полезными:

Asterisk-Users

Этот список-тень его прежнего себя. В то время как раньше он генерировал несколько сотен сообщений в день, большая часть этого трафика переместилась на сайт сообщества Digium Asterisk (выше).

Asterisk-Dev

Разработчики Asterisk тусуются здесь. Целью и направленностью этого списка является обсуждение разработки программного обеспечения Asterisk, и участники активно защищают эту цель. Ожидайте много гнева, если опубликуете что-либо в этом списке, не относящееся конкретно к программированию или разработке базы кода Asterisk. Общие вопросы кодирования (такие как запросы о взаимодействии с AGI или AMI) должны быть направлены в список *Asterisk-Users*.

Список Asterisk-Dev не является поддержкой второго уровня! Если вы прокрутите архивы списков рассылки, то увидите, что это строгое правило. Список рассылки Asterisk-Dev посвящен обсуждению основных разработок Asterisk, а вопросы по взаимодействию с внешними программами через AGI или AMI должны быть размещены в списке Asterisk-Users.

Сайт Asterisk Wiki

Это не совсем тусовка сообщества, но она заслуживает упоминания. Digium поддерживает Вики для Asterisk по адресу wiki.asterisk.org, этот сайт постоянно обновляется командой Digium, и автоматизированные скрипты экспортируют документацию на основе XML из источников Asterisk в саму Вики, что помогает гарантировать актуальность данных.

Более старая Вики существует по адресу www.voip-info.org, что в наши дни является своего рода историческим курьезом и источником большого просветления и путаницы. Хотя здесь содержится огромное количество информации, большая ее часть устарела. Мы включаем ссылку на неё здесь просто потому, что вы, вероятно, попадёте на неё в один прекрасный день и подумаете что попали в материнскую жилу, но то, что вы на самом деле нашли, больше похоже на музей исторических странностей: увлекательно, но необязательно актуально.

IRC-каналы

Сообщество Asterisk поддерживает каналы Internet Relay Chat (IRC) на *irc.freenode.net*. Двумя наиболее активными каналами являются *#asterisk* и *#asterisk-dev*[4]. Чтобы сократить количество вторжений спам-ботов, оба этих канала требуют регистрации для присоединения. Чтобы зарегистрироваться, запустите `/msg nickserv help` при подключении к службе через ваш любимый IRC-клиент.

Вывод

Так с чего начать? Что ж, когда речь заходит об Asterisk, можно говорить гораздо больше, чем мы можем вписать в одну книгу. Эта книга может только заложить основы, но из этого фундамента вы сможете прийти к пониманию концепции Asterisk - и из этого, кто знает, что вы будете строить?

4 Канал *#asterisk-dev* предназначен для обсуждения изменений в базовой кодовой базе Asterisk и также не является поддержкой второго уровня. Обсуждения, связанные с программированием внешних приложений, которые взаимодействуют с Asterisk через AGI или AMI, должны быть в *#asterisk*.

Прежде всего, но не обязательно в таком порядке.
– Доктор Кто

Asterisk очень отличается от других, более традиционных УАТС тем, что диалплан в Asterisk обрабатывает все входящие каналы по существу одинаково, а не разделяя их на станции, транки, периферийные модули и т.д.

В традиционной АТС существует логическое различие между станциями (телефонными аппаратами) и транками (магистралями - ресурсами, которые подключаются к внешнему миру). Это ограничение делает творческую маршрутизацию в традиционных УАТС очень сложной или невозможной.

Asterisk, с другой стороны, не имеет внутреннего понятия транков или станций. В Asterisk все, что входит или выходит из системы, проходит через какой-то канал. Существует множество различных типов каналов; однако диалплан Asterisk обрабатывает все каналы аналогичным образом, что означает, например, внутренний пользователь может существовать на конце внешнего транка (например, сотовый телефон) и обрабатываться диалпланом точно так же, как если бы пользователь был на внутреннем номере. Если вы не работали с традиционной АТС[1], то может быть не сразу очевидно насколько это является мощным и освобождающим. Рисунок 2-1 иллюстрирует различия между этими двумя архитектурами.

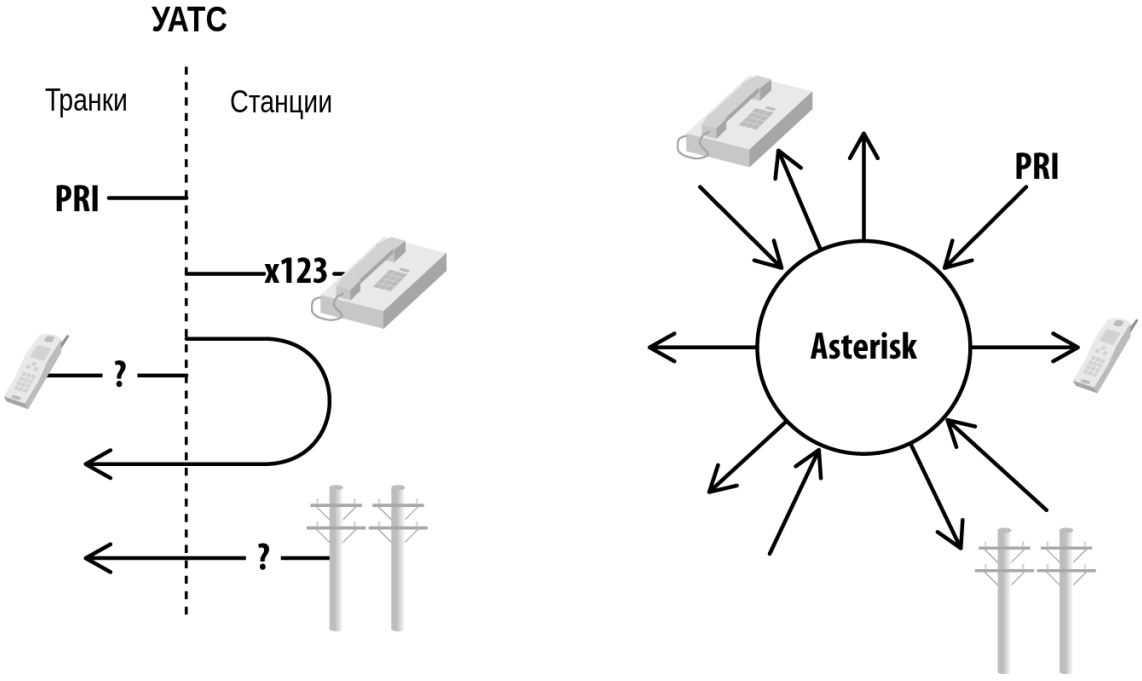

Рисунок 2-1. Архитектура Asterisk против УАТС

Модули

Asterisk построена на *модулях*. Модуль — это загружаемый компонент, обеспечивающий определенную функциональность, такую как драйвер канала (например, `chan_pjsip.so`) или ресурс, который позволяет подключиться к внешней технологии (например `func_odbc.so`). Модули Asterisk загружаются на основе параметров, определенных в файле */etc/asterisk/modules.conf*. Мы обсудим

1 Хорошим показателем того, что вы работали с традиционными УАТС, является наличие большой мозоли на лбу, полученной в результате слишком большого количества ударов головой о кирпичную стену, чтобы сосчитать.

использование многих модулей в этой книге, но на этом этапе мы просто хотим представить концепцию модулей и дать вам представление о типах модулей, которые доступны.

На самом деле можно запустить Asterisk вообще без каких-либо модулей, хотя в этом состоянии он ничего не сможет сделать. Это бывает полезно для понимания модульной природы Asterisk для оценки архитектуры.

Вы можете запустить Asterisk без модулей, загружаемых по умолчанию и загружать каждый нужный модуль вручную из консоли, но это не то, что вы хотели бы запустить в продакшн; это было бы полезно только в том случае, если бы вы настраивали производительность системы, в которой хотели убрать все, что не требуется вашим конкретным приложением Asterisk.

Типы модулей в Asterisk включают в себя следующие:

- Приложения - рабочие лошадки диалплана, такие как `Dial()`, `Voicemail()`, `Playback()`, `Queue()` и т.д.
- Модули соединения — механизмы, которые соединяют каналы (вызовы) друг с другом
- Модули записи деталей вызовов (CDR)
- Модули регистрации событий канала (CEL)
- Драйверы каналов — различные соединения с системой и из системы; SIP (Session Initiation Protocol) использует канальный драйвер PJSIP
- Трансляторы кодеков — преобразовывают различные кодеки как G729, G711, G722, Speex и так далее
- Интерпретаторы форматов — как указано выше, но относящиеся к файлам, хранящимся в файловой системе
- Функции диалплана — расширенные возможности диалплана
- Модули УАТС
- Модули ресурсов
- Дополнительные модули
- Тестовые модули

Существует официальный список типов статуса поддержки, включенных в `menuselect`[2].

Приложения

Приложения диалплана используются в *extensions.conf* для определения различных действий, применимых к вызову. Например, приложение `Dial()` отвечает за создание исходящих соединений с внешними ресурсами и, возможно, является самым важным приложением диалплана. Доступные приложения перечислены в Таблице 2-1.

Таблица 2-1. Популярные приложения диалплана

Имя	Назначение
`app_authenticate`	Сравнивает сигналы набора DTMF с указанной строкой (паролем)
`app_cdr`	Записывает данные в CDR
`app_chanspy`	Позволяет каналу прослушивать аудио на другом канале
`app_confbridge`	Обеспечивает конференц-связь
`app_dial`	Используется для соединения каналов вместе (т. е. для совершения телефонных звонков)

2 Эта команда доступна как часть процесса установки. Мы обсудим использование `menuselect` в главе об установке.

Имя	Назначение
app_directed_pickup	Отвечает на вызов, который вызывает другой добавочный номер
app_directory	Представляет список имен из *voicemail.conf*
app_dumpchan	Дамп переменных канала в интерфейс командной строки Asterisk (CLI)
app_echo	Эхо-сигналы, полученные обратно в исходный канал (может быть полезно для демонстрации задержки)
app_exec	Содержит Exec(), TryExec() и ExecIf(): выполняет приложение диалплана по условию
app_mixmonitor	Записывает обе стороны вызова (передача и прием) и смешивает их вместе в один файл
app_originate	Позволяет логике диалплана инициировать вызов (в отличие от вызова, поступающего на канал)
app_page	Создает несколько аудио соединений с указанными устройствами для публичного вещания (пейджинг)
app_parkandannounce	Включает автоматическое оповещение о припаркованных вызовах
app_playback	Воспроизведение файла в канал (не принимает ввод)
app_playtones	Воспроизведение пар тонов указанных частот (в основном DTMF)
app_queue	Обеспечивает автоматическое распределение вызовов (ACD)
app_read	Запрашивает ввод цифр от абонентов и назначает ввод переменной
app_readexten	Запрашивает ввод цифр у вызывающих абонентов и передает вызов на указанный добавочный номер и контекст
app_record	Запись полученного аудио в файл
app_senddtmf	Передает DTMF вызывающей стороне
app_stack	Обеспечивает Gosub(), GoSubIf(), Return(), SteckPop(), LOCAL() и Local_PEEK()
app_stasis	Передает управление вызовами в приложение ARI - многие разработчики Asterisk используют это приложение, а оттуда обрабатывают всю остальную часть своей разработки за пределами диалплана Asterisk
app_system	Выполняет команды в Linux shell
app_transfer	Выполняет трансфер на текущем канале
app_voicemail	Предоставляет голосовую почту
app_while	Включает While(), EndWhile(), ExitWhile() и ContinueWhile(); обеспечивает функциональность цикла while в диалплане

Модули соединений

Модули соединений выполняют фактическое соединение каналов. Эти модули, перечисленные в Таблице 2-2, в настоящее время используются только для (и необходимы) *app_confbridge*.

Таблица 2-2. Модули соединений

Имя	Назначение
bridge_builtin_features	Выполняет соединение при использовании встроенных пользовательских функций (например, найденных в *features.conf*).
bridge_multiplexed	Выполняет сложное мультиплексирование, как требуется в большом конференц-зале (несколько участников). В настоящее время используется только app_confbridge.
bridge_simple	Выполняет простое соединение канал-канал.
bridge_softmix	Выполняет простое мультиплексирование, как требуется в большом конференц-зале (несколько участников). В настоящее время используется только app_confbridge.

В следующих разделах мы рассмотрели список модулей, которые, по нашему мнению, достаточно важны для обсуждения в этой книге. Вы найдете много других модулей в загрузке Asterisk, но многие старые модули либо устарели, либо имеют слабую поддержку или вовсе не поддерживаются, и поэтому не рекомендуются для производства, если у вас нет доступа к разработчикам, которые могут поддерживать их для вас.

Модули записи деталей вызова (CDR)

Модули CDR, перечисленные в Таблице 2-3, предназначены для обеспечения как можно большего числа методов хранения записей сведений о вызовах. CDR можно хранить в файле (по умолчанию), базе данных, RADIUS или *syslog*.

Записи деталей вызовов не предназначены для использования в сложных приложениях биллинга. Если вам требуется больше контроля над биллингом и отчетностью о вызовах - обратите внимание на журнал событий канала (CEL), обсуждаемый далее. Преимущество CDR заключается в том, что он просто работает.

Таблица 2-3. Общие модули записи деталей вызова

Имя	Назначение
`cdr_adaptive_odbc`	Позволяет записывать CDR через платформу ODBC с возможностью добавления пользовательских полей
`cdr_csv`	Записывает CDR на диск в файл с разделителями-запятыми (CSV)
`cdr_custom`	Записывает CDR в файл CSV, но допускает добавление пользовательских полей
`cdr_odbc`	Пишет CDR через ODBC фреймворк
`cdr_syslog`	Записывает CDR в *syslog*

Модули журналирования событий канала

Регистрация событий канала (CEL) обеспечивает гораздо более мощный контроль над отчетами об активности вызовов. Кроме того, он требует более тщательного планирования вашего плана набора и ни в коем случае не будет работать автоматически. Модули Asterisk CEL перечислены в Таблице 2-4.

Таблица 2-4. Модули логирования событий канала

Имя	Назначение
`cel_custom`	CEL на диск/файл
`cel_manager`	CEL в AMI
`cel_odbc`	CEL в ODBC

Драйверы каналов

Без драйверов каналов у Asterisk не было бы возможности совершать или принимать вызовы. Каждый драйвер канала специфичен для протокола или типа канала, который он поддерживает (SIP, ISDN и т.д.). Модуль канала действует как шлюз к ядру Asterisk. Некоторые из наиболее популярных драйверов каналов Asterisk перечислены в Таблице 2-5.

Таблица 2-5. Популярные драйверы каналов

Имя	Назначение
`chan_bridge`	Используется внутри приложения `ConfBridge()`; не должен использоваться напрямую
`chan_dahdi`	Обеспечивает подключение к картам ТфОП, использующим драйверы каналов DAHDI
`chan_local`	Предоставляет механизм для обработки части диалплана как канала

Имя	Назначение
chan_motif	Реализует протокол Jingle, включая возможность подключения к Google Talk и Google Voice; представлен в Asterisk 11
chan_multicast_rtp	Обеспечивает подключение к потокам многоадресного Realtime Transport Protocol (RTP)
chan_pjsip	Драйвер канала Session Initiation Protocol (SIP)

Трансляторы кодеков

Трансляторы кодеков[3] (часто называемые *транскодерами*) позволяют Asterisk конвертировать форматы аудиопотоков между вызовами. Поэтому, если вызов поступает по каналу PRI (используя G.711) и должен быть передан в сжатый канал SIP (например, используя G.729 - один из многих кодеков, которые может обрабатывать SIP), соответствующий транслятор кодека выполнит преобразование.

> Кодеки - это сложные алгоритмы, обрабатывающие преобразование аналоговой информации (в данном случае звука, но также может быть и видео) в цифровой формат. Многие кодеки также обеспечивают сжатие и исправление ошибок, но это не является обязательным требованием.

Если кодек (например G.729) использует сложный алгоритм кодирования, интенсивное использование транскодинга может создать огромную нагрузку на процессор. Специализированное оборудование для декодирования/кодирования G.729 доступно от производителей оборудования, таких как Sangoma и Digium (и, вероятно, других).

Asterisk делает довольно хорошую работу по поддержке кодеков, но в основном сосредоточен на кодеках, обычно используемых телефонными приложениями (в отличие от кодеков, используемых, скажем, для музыки или видео, таких как MP3 или MP4). Они перечислены в Таблице 2-6.

Таблица 2-6. Общие трансляторы кодеков

Название	Назначение
codec_alaw	Кодек PCM A-law используется во всем мире для ТфОП (кроме Канады/США). Этот кодек (вместе с ulaw) должен быть включен на всех ваших каналах.
codec_g729	До недавнего времени это был запатентованный кодек, но теперь он является бесплатным. На момент написания этой статьи он по-прежнему продается Digium в качестве дополнения, но его также можно найти в виде бесплатного пакета. Это очень популярный кодек, если требуется сжатие (и использование процессора не является проблемой), но он накладывает нагрузку на процессор, добавляет задержку к вызовам, немного снижает качество и никоим образом не уменьшает накладные расходы.
codec_a_mu	Прямой конвертер A-law в mu-law.
codec_g722	Широкополосный аудиокодек.
codec_gsm	Кодек Global System for Mobile Communications (GSM). Очень низкое качество звука.
codec_ilbc	Интернет-кодек с низким битрейтом (iLBC).
codec_lpc10	Линейный предсказательный кодирующий вокодер (чрезвычайно низкая пропускная способность).
codec_opus	Предназначен для замены speex (и vorbis).
codec_resample	Пересемлирование между 8-ми и 16-тибитными линейными сигналами.
codec_speex	Кодек Speex.
codec_ulaw	Кодек PCM Mu-law, используемый на ТфОП в Канаде/США. Более точно описывается как μ-закон,

3 Термин кодек - это сокращение от «кодер-декодер».

Название	Назначение
	но не у многих людей есть греческая буква μ на клавиатуре, поэтому обычно пишется как ulaw[a]. Часто является кодеком по умолчанию, и должен быть включен на всех ваших каналах.

[a] Произносится как “мью-лоу”, но так же вы часто будете слышать как "ю-лоу".

Digium предоставляет некоторые дополнительные полезные модули кодеков: `codec_g729`, `codec_silk`, `codec_siren7` и `codec_siren14`. Эти модули кодеков не являются open source по различным причинам. Вы должны приобрести лицензию на использование `codec_g729`, но остальные являются бесплатными. Вы можете найти их на сайте Digium.

Интерпретаторы формата

Интерпретаторы форматов в Таблице 2-7 выполняют ту же функцию, что и трансляторы кодеков, но они работают с файлами, а не с каналами, и обрабатывают не только аудио. Если у вас есть запись в меню, которая была сохранена как GSM - вам нужно будет использовать интерпретатор формата для воспроизведения этой записи на любые каналы, не использующие кодек GSM[4].

Если вы храните запись в нескольких форматах одновременно (например, WAV, GSM и т. д.), Asterisk определит наименее затратный формат[5] для использования, когда каналу необходимо воспроизвести эту запись.

Таблица 2-7. Интерпретаторы форматов

Название	Воспроизведение файлов, хранящихся в
`format_g729`	G.729: *.g729*
`format_gsm`	RPE-LTP (оригинальный кодек GSM): *.gsm*
`format_h264`	H.264 video: *.h264*
`format_ilbc`	Интернет кодек с низким битрейтом: *.ilbc*
`format_jpeg`	Графический файл: *.jpeg*, *.jpg*
`format_ogg_vorbis`	Ogg контейнер: *.ogg*
`format_pcm`	Различные форматы импульсно-кодовой модуляции: *.alaw*, *.al*, *.alw*, *.pcm*, *.ulaw*, *.ul*, *.mu*, *.ulw*, *.g722*, *.au*
`format_siren14`	G.722.1 Annex C (14 kHz): *.siren14*
`format_siren7`	G.722.1 (7 kHz): *.siren7*
`format_sln`	8-bit signed linear: *.sln*, *.raw*
`format_vox`	*.vox*
`format_wav`	*.wav*
`format_wav_gsm`	GSM аудио в контейнере WAV: *.wav*, *.wav49*

Функции диалплана

Функции диалплана, перечисленные в Таблице 2-8, дополняют приложения диалплана (смотри “Приложения”. Они предоставляют множество полезных улучшений для таких вещей, как обработка строк, смещение времени и даты и подключение ODBC.

4 Отчасти по этой причине мы не рекомендуем формат GSM по умолчанию для системных записей. WAV-записи будут звучать лучше и использовать меньше тактов процессора.

5 Некоторые кодеки создают значительную нагрузку на ЦП, настолько, что система, которая может поддерживать несколько сотен каналов без транскодирования, будет обрабатывать только несколько десятков при его использовании.

Таблица 2-8. Список полезных функций диалплана

Название	Назначение
func_audiohookinherit	Позволяет записывать звонки после трансфера
func_blacklist	Пишет/читает черный список в *astdb*
func_callcompletion	Получает/устанавливает параметры конфигурации завершения вызова для канала
func_callerid	Получает/устанавливает идентификатор звонящего (Caller ID)
func_cdr	Получает/устанавливает переменную CDR
func_channel	Получает/устанавливает информацию канала
func_config	Включает AST_CONFIG(); считывает переменные из файла конфигурации
func_curl	Использует cURL для получения данных из URI
func_cut	Делит и нарезает строки
func_db	Предоставляет функции *astdb*
func_devstate	Получает состояние устройства
func_dialgroup	Создает группу для одновременного набора
func_dialplan	Проверяет, что назначенная цель существует в диалплане
func_env	Включает FILE(), STAT() и ENV(); выполняет действия операционной системы
func_global	Получает/устанавливает глобальные переменные
func_groupcount	Получает/устанавливает количество каналов для участников группы
func_hangupcause	Получает/устанавливает информацию о зависании из канала
func_logic	Включает ISNULL(), SET(), EXISTS(), IF(), IFTIME()и IMPORT(); выполняет различные логические функции
func_math	Включает MATH(), INC() и DEC(); выполняет математические функции
func_odbc	Позволяет интегрировать диалплан с ресурсами ODBC
func_rand	Возвращает случайное число в заданном диапазоне
func_realtime	Выполняет поиск в Asterisk Realtime Architecture (ARA)
func_redirecting	Предоставляет доступ к информации о том, откуда был перенаправлен этот вызов
func_shell	Выполняет операции оболочки Linux и возвращает результаты
func_sprintf	Выполняет функции строкового формата, аналогичные функции C с тем же именем
func_srv	Выполняет поиски SRV в диалплане
func_strings	Включает в себя более десятка функций обработки строк
func_timeout	Получает / устанавливает таймауты на канале
func_uri	Преобразует строки в URI-безопасную кодировку
func_vmcount	Возвращает количество сообщений в папке голосовой почты для определенного пользователя.

PBX Модули

Модули PBX - это периферийные модули, обеспечивающие улучшенные механизмы управления и настройки. Например, pbx_config - это модуль, загружающий традиционный диалплан Asterisk. Доступные в настоящее время модули PBX перечислены в Таблице 2-9.

Таблица 2-9. PBX модули

Название	Назначение
pbx_config	Этот модуль предоставляет традиционный, популярный язык диалплана для Asterisk. Без этого

Название	Назначение
	модуля Asterisk не может читать *extensions.conf*.
pbx_dundi	Выполняет поиск данных в удаленных системах Asterisk.
pbx_realtime	Предоставляет функциональные возможности, связанные с архитектурой Asterisk Realtime.
pbx_spool	Обеспечивает поддержку исходящих сообщений, относящихся к файлам вызовов Asterisk.

Модули ресурсов

Модули ресурсов интегрируют Asterisk с внешними ресурсами. Эта группа модулей фактически превратилась в универсальное средство для вещей, которые не вписываются в другие категории. Разобьем их на несколько подгрупп модулей, которые связаны между собой.

Бэкэнды конфигурации

Asterisk по умолчанию настроен с использованием текстовых файлов в */etc/asterisk*. Модули, перечисленные в Таблице 2-10, предоставлют альтернативные методы настройки. Смотри Главу 15 для получения подробной документации по настройке конфигурации на основе базы данных.

Таблица 2-10. Конфигурационные бэкэнд-модули

Название	Назначение
res_config_curl	Получает информацию о конфигурации, используя cURL
res_config_ldap	Получает информацию о конфигурации, используя LDAP
res_config_odbc	Получает информацию о конфигурации, используя ODBC

Итеграция календаря

Asterisk включает некоторую интеграцию с календарными системами. Вы можете читать и записывать информацию календаря из диалплана. Вы также можете иметь звонки на основе записей календаря. Интеграция основного календаря обеспечивается модулем `res_calendar`. Остальные модули предоставляют возможность подключения к определенным типам календарных серверов. Таблица 2-11 перечисляет модули интеграции календаря.

Таблица 2-11. Модули интеграции календаря

Название	Назначение
res_calendar	Обеспечивает базовую интеграцию с календарными системами
res_calendar_caldav	Позволяет функциям, предоставляемым `res_calendar` , подключаться к календарям через CalDAV
res_calendar_exchange	Позволяет функциям `res_calendar` подключаться к MS Exchange
res_calendar_icalendar	Позволяет функциям `res_calendar` подключаться к Apple/Google iCalendar

Другие модули ресурсов

Таблица 2-12 включает остальные модули ресурсов, которые не вписываются в одну из подгрупп, которые мы определили ранее в этом разделе.

Таблица 2-12. Модули ресурсов

Название	Назначение
res_adsi	Предоставляет ADSI[a]
res_agi	Предоставляет Asterisk Gateway Interface (смотри Главу 18)
res_corosync	Предоставляет распределенные сообщения индикации ожидания (MWI) и состояние

Название	Назначение
	устройства уведомления через Corosync Cluster Engine
res_crypto	Предоставляет криптографические возможности
res_curl	Предоставляет общие подпрограммы для других модулей cURL
res_fax	Предоставляет общие подпрограммы для других факсимильных модулей
res_fax_spandsp	Плагин для факса с использованием пакета spandsp
res_http_post	Обеспечивает поддержку POST upload для HTTP-сервера Asterisk
res_http_websocket	Обеспечивает поддержку WebSocket для внутреннего HTTP-сервера Asterisk (требуется WebRTC)
res_monitor	Предоставляет ресурсы записи разговоров
res_musiconhold	Предоставляет ресурсы музыки на удержании (MOH)
res_mutestream	Позволяет заглушить/включить звук аудиопотоков
res_odbc	Предоставляет общие подпрограммы для других модулей ODBC
res_phoneprov	Автонастройка телефонов с HTTP-сервера Asterisk
res_pktccops	Предоставляет ресурсы PacketCable COPS
res_security_log	Включает ведение журнала событий безопасности, генерируемых другими частями Asterisk
res_snmp	Предоставляет информацию о состоянии системы в SNMP-managed network
res_speech	Общий API распознавания речи[b]
res_stasis	Связывает различные компоненты инфраструктуры приложений Stasis
res_xmpp	Предоставляет ресурсы XMPP (FKA Jabber)

[a] Хотя большинство функций ADSI в Asterisk никогда не используется - приложение голосовой почты использует этот ресурс.
[b] Для использования требуется отдельно лицензируемый продукт.

Дополнительные модули

Дополнительные модули - это разработанные сообществом модули с правами на использование или распространение, отличными от основного кода (Таблица 2-13). Они хранятся в отдельном каталоге и не компилируются и не устанавливаются по умолчанию. Чтобы включить эти модули, используйте утилиту настройки сборки menuselect.

Таблица 2-13. Дополнительные модули

Название	Назначение	Популярность/статус
chan_ooh323	Позволяет совершать и принимать VoIP звонки по протоколу H323	Usable
format_mp3	Позволяет Asterisk воспроизводить MP3 файлы	Usable
res_config_mysql	Использует базу данных MySQL как сервер конфигурации в режиме реального времени	Useful

Тестовые модули

Тестовые модули используются командой разработчиков Asterisk для проверки нового кода. Они постоянно меняются и добавляются, и не являются полезными, если вы не разрабатываете программное обеспечение Asterisk.

Однако если вы являетесь разработчиком Asterisk, то набор тестов Asterisk может представлять для вас интерес, поскольку вы можете создавать автоматизированные тесты для Asterisk и отправлять их обратно в проект, работающий на нескольких различных операционных системах и типах машин. Постоянно расширяя число тестов, проект Asterisk позволяет избежать создания регрессий в коде.

Отправляя свои собственные тесты в проект - вы можете чувствовать себя более уверенно в будущих обновлениях.

Более подробная информация о тестах доступна в разделе “Asterisk Test Suite”, или вы можете присоединиться к каналу *#asterisk-testing* в IRC-сети Freenode.

Файловая структура

Asterisk - сложная система, состоящая из множества ресурсов. Эти ресурсы используют файловую систему несколькими способами. Поскольку Linux настолько гибок в этом отношении, полезно понять, какие данные хранятся, чтобы вы могли понять, где вы, вероятно, найдете определенный бит хранимых данных (например, сообщения голосовой почты или файлы журналов).

Конфигурационные файлы

Конфигурационные файлы Asterisk включают в себя *extensions.conf*, *pjsip.conf*, *modules.conf* и десятки других файлов, определяющих параметры для различных каналов, ресурсов, модулей и функций, которые могут использоваться.

Эти файлы обычно находятся в */etc/asterisk*. Вы будете много работать в этой папке, когда будете настраивать и администрировать свою систему Asterisk.

Модули

Модули Asterisk обычно устанавливаются в папку */usr/lib/asterisk/modules*. Обычно вам не придется взаимодействовать с этой папкой, однако будет полезно узнать где находятся модули. Например, если вы обновите Asterisk и выберете разные модули на этапе установки `menuselect`, старые (несовместимые) модули из предыдущей версии Asterisk не будут удалены, и вы получите предупреждение от сценария установки. Эти старые файлы необходимо будет удалить из папки *modules*. Это можно сделать либо вручную, либо с помощью `make uninstall`.

Библиотека ресурсов

Существует несколько ресурсов, которым требуются внешние источники данных. Например, музыка на удержании (MOH) не может воспроизводиться если у вас нет какой-то музыки для воспроизведения. Системные подсказки также должны храниться где-то на жестком диске. В папке */var/lib/asterisk* хранятся системные приглашения, сценарии AGI, музыка на удержание и другие файлы ресурсов.

Spool

Spool - это место, где приложения хранят файлы в системе Linux, которые будут часто меняться или которые будут обрабатываться другими процессами позднее. Например, задания на печать в Linux и ожидающие сообщения электронной почты обычно записываются в спул, пока они не будут обработаны.

В Asterisk спул используется для хранения временных элементов, таких как голосовые сообщения, записи вызовов[6], файлы вызовов и так далее.

Спул Asterisk находится в каталоге */var/spool/asterisk*.

6 Не записи деталей вызовов (CDR), а скорее аудиозаписи вызовов, генерируемых `MixMonitor()` и связанными приложениями.

Журналирование

Asterisk может генерировать несколько разных типов лог-файлов. Папка */var/log/asterisk* - это место, куда записываются подробные записи вызовов (CDR), события канала из CEL, журналы отладки, журналы очереди, сообщения, ошибки и другие выходные данные.

Эта папка будет чрезвычайно важна для любых предпринимаемых вами действий по устранению неполадок. Мы поговорим подробнее о том, как использовать логи Asterisk в Главе 21.

Диалплан

Диалплан - это сердце Asterisk. Все каналы, поступающие в систему, будут проходить через диалплан, содержащий сценарии потока вызовов, определяющие порядок обработки входящих вызовов.

Диалплан обычно пишется с использованием собственного синтаксиса Asterisk, который хранится в файле с именем */etc/asterisk/extensions.conf* . Существуют и другие способы управления потоком вызовов, и мы рассмотрим их позже, но независимо от того, какой метод вы в конечном итоге используете, вы обнаружите, что базовое понимание традиционной схемы диалплана будет чрезвычайно полезным. Именно на этом мы сосредоточимся большую часть первых двух третей этой книги.

Позже мы рассмотрим обработку потока вызовов вне диалплана, используя такие технологии, как AMI, AGI и ARI.

Аппаратные средства

Asterisk способен общаться с огромным количеством различных технологий. Как правило, эти соединения выполняются через сетевое соединение TCP/IP (обычно с использованием SIP). Однако подключения к более традиционным телекоммуникационным каналам, таким как PRI (T1, E1 и т.д.), BRI (EuroISDN) SS7 (в основном T1 и E1), и аналоговым (все, от нескольких портов FXO и FXS до крупных банков каналов, питаются через соединения T1/E1 CAS/RBS), также может быть достигнуто с помощью физических карт, установленных на сервере.

Многие компании производят это оборудование, такие как Digium (спонсор, владелец и основной разработчик Asterisk), Sangoma (который недавно приобрел Digium), Dialogic (также компания Sangoma), OpenVox, Pika, Voicetronix, beroNet и многие другие. Все эти компании были связаны с Asterisk на протяжении многих лет.

Наиболее популярное оборудование для Asterisk, как правило, предназначено для работы через интерфейс аппаратного устройства Digium Asterisk (известный как DAHDI). Это сложная архитектура и она выходит за рамки этой книги. Все серверные телефонные карты будут иметь требования к установке, уникальные для производителя, и вам потребуются навыки в установке оборудования Linux, а также в устранении неисправностей и подготовке традиционных сетей ТфОП.

Если вам необходимо взаимодействовать с традиционными линиями ТфОП с использованием Asterisk, мы рекомендуем вам сохранить Asterisk в качестве платформы только для SIP и взаимодействовать с помощью стороннего шлюза какого-либо рода. Имейте в виду: это не материал начального уровня, и если вы только начинаете использовать Asterisk - вам настоятельно рекомендуется оставить свои первоначальные решения только для SIP.

Версии Asterisk

Методология выпуска Asterisk прошла через несколько стилей. В прошлом это приводило к некоторой путанице, но в наши дни управление версиями довольно простое и относительно легкое

для понимания. Digium сохранил отличную ссылку на Asterisk wiki, и мы призываем вас перейти на последние подробности о версиях Asterisk.

Эта книга была написана и протестирована с использованием версии 16, но вы обнаружите, что фундаментальные концепции, которые мы исследуем, будут актуальны для большинства версий Asterisk. Концептуальная структура Asterisk не менялась в течение достаточно долгого времени, и на момент написания этой статьи не было никаких известных планов изменить это в будущем. Будущие версии будут предоставлять более мощные мультимедийные и конференц-возможности, конечно, но они, вероятно, будут реализованы в рамках существующей структуры.

Вывод

Asterisk состоит из множества различных технологий, большинство из которых сложны сами по себе. В результате понимание архитектуры Asterisk может быть ошеломляющим. Тем не менее, реальность такова, что Asterisk хорошо спроектирован для того, что он делает, и, по нашему мнению, достиг замечательного баланса между гибкостью и сложностью.

Глава 3. Установка Asterisk

> *Я долго шла к великим и благородным целям, но мой главный долг выполнять простые задачи так, как будто они великие и благородные. Мир движется не только мощными импульсами, которые дают ему герои, но также и крошечными толчками каждого трудящегося человека.*
>
> – Хелен Келлер

В этой главе мы рассмотрим установку Asterisk из исходного кода. Многие люди уклоняются от этого метода, утверждая, что он слишком сложен и отнимает много времени. Наша цель здесь - продемонстрировать, что установка Asterisk из исходного кода на самом деле не так уж сложна. Что еще более важно - мы хотим предоставить вам лучшую платформу Asterisk, на которой можно учиться.

В этой книге мы поможем вам построить функционирующую систему Asterisk с нуля. Для достижения этой цели в этой главе мы построим базовую платформу для вашей системы Asterisk. Поскольку мы устанавливаем из исходного кода, потенциально существует множество вариантов того, как вы можете это сделать. Наша цель - поставить стандартный вид платформы, подходящий для исследований во множестве областей. Можно упростить Asterisk до самых основ и запустить на очень слабой машине; однако это упражнение остается на усмотрение читателя. Процесс, который мы обсуждаем здесь, предназначен для быстрого и простого запуска, без кратковременного изменения доступа к интересным функциям.

Большинство команд, которые вы видите, будут лучше всего обрабатываться с помощью ряда операций копирования-вставки (на самом деле, мы настоятельно рекомендуем Вам иметь электронную версию этой книги под рукой именно для этой цели)[1]. Хотя это выглядит как большой набор команд, которые мы даем вам - получить конфигурацию можно от начала до конца менее чем за 30 минут, так что это действительно не так сложно, как может показаться. Мы запускаем некоторые предварительные условия, некоторую компиляцию и некоторую конфигурацию после установки, и Asterisk готов к работе.

Для краткости эти шаги будут выполнены на системе CentOS 7. Это функционально эквивалентно RHEL и достаточно похоже на Fedora, так что шаги должны быть практически идентичны. Для других платформ, таких как Debian/Ubuntu и так далее, инструкции также будут аналогичны, но вам нужно будет адаптироваться по мере необходимости для вашей платформы[2].

Первая часть инструкции по установке не будет касаться Asterisk как таковой, а скорее некоторых зависимостей, которые требуются Asterisk или необходимы для некоторых более полезных функций (таких как интеграция с базой данных). Мы постараемся сохранить инструкции достаточно общими чертами, чтобы они были полезны при любом распределении по вашему выбору.

В этих инструкциях предполагается, что вы являетесь опытным администратором Linux[3]. Полностью работающая система Asterisk будет состоять из достаточно обособленных частей, так что вам будет сложно справиться со всем этим, если у вас мало или нет основ Linux. Мы по-прежнему рекомендуем вам погрузиться, но пожалуйста, учтите тот факт, что будет крутая кривая обучения, если у вас еще нет твердого опыта командной строки Linux.

---

1 Она была выпущена под лицензией Creative Commons - поэтому, если вы приобрели печатную копию (и мы благодарим вас!), вы также можете скачать программную копию для поиска и копирования/вставки.

2 Asterisk должен работать практически на любой платформе Linux, и если вы знакомы с основным процессом установки программного обеспечения на Linux, вы должны найти установку Asterisk довольно простой.

3 Под этим мы в основном подразумеваем, что вам удобно управлять системой исключительно из оболочки.

Если вы хотите изучить командную строку Linux, одна из лучших книг, которые мы нашли - это *Командная строка Linux* Уильяма Шоттса, которая была выпущена под лицензией Creative Commons и погружается во все знания, необходимые для эффективного использования оболочки Linux. Её можно найти по адресу linuxcommand.org вы можете изучить книгу от начала до конца, и почти все, что вы узнаете, будет тем, что стоит знать любому опытному администратору Linux.

Еще одна фантастическая книга - это, конечно же, легендарное *Руководство по системному администрированию UNIX и Linux* Дэна Макина, Бена Уэйли, Трента Р. Хейна, Гарта Снайдера и Эви Немет (Prentice Hall). Настоятельно рекомендуем.

Пакеты Asterisk

Существуют пакеты Asterisk, которые можно установить с помощью систем управления пакетами, таких как *yum* или *apt-get*. Вы можете использовать их как только ознакомитесь с Asterisk.

Если вы используете RHEL, Asterisk доступен из репозитория EPEL из проекта Fedora. Пакеты Asterisk доступны в универсальном репозитории для Ubuntu.

Вы также должны отметить, что из-за истории Asterisk он может интегрироваться со множеством технологий телефонии; однако новичку в Asterisk лучше сперва изучить интеграцию SIP, прежде чем беспокоиться о более сложных, устаревших или периферийных типах каналов. После того, как вы освоитесь с Asterisk в чистой среде SIP, вам будет намного проще смотреть на интеграцию других типов каналов.

Проекты основанные на Asterisk

Многие проекты используют Asterisk в качестве базовой платформы. Некоторые из них, такие как графический интерфейс FreePBX, стали настолько популярными, что многие люди ошибочно принимают его за сам продукт Asterisk. На самом деле, графический интерфейс FreePBX настолько распространен, что его можно найти в большинстве известных проектов на основе Asterisk. Эти проекты берут базовый продукт Asterisk и добавляют веб-интерфейс администрирования, сложную базу данных и внешние функции, которые полезны в типичной УАТС (например преднастройка (provisioning) аппаратов, сервер времени и т.д.).

Мы решили не освещать эти проекты в этой книге, по нескольким причинам:

- Эта книга пытается, насколько это возможно, сосредоточиться на Asterisk и только Asterisk.
- О многих из этих проектов, основанных на Asterisk, уже написаны книги.
- Мы считаем, что если вы изучите Asterisk так, как мы будем учить вас, знания будут хорошо служить вам независимо от того, решите ли вы в конечном итоге использовать одну из этих предварительно упакованных версий Asterisk.
- Если вы хотите понять, что происходит под капотом системы на базе FreePBX, эта книга познакомит вас с некоторыми навыками, которые вам понадобятся.
- Для нас сила Asterisk заключается в том, что он не пытается решить ваши проблемы за вас. Эти проекты являются поистине удивительными примерами того, что можно построить с помощью Asterisk. Однако, если вы хотите создать свое собственное приложение Asterisk (чем на самом деле является Asterisk), эти проекты могут создавать ненужные препятствия, просто потому, что они направлены на упрощение процесса создания бизнес-АТС, а не на то, чтобы сделать возможным доступ к полному потенциалу платформы Asterisk.

Некоторые из самых популярных проектов на основе Asterisk включают (в определенном порядке):

AsteriskNOW

Управляется Digium. Использует графический интерфейс FreePBX.

Issabel

Ответвление оригинальных релизов open source продукта Elastix[4]. Использует графический интерфейс FreePBX.

The Official FreePBX Distro

Официальный дистрибутив проекта FreePBX. Управляется Sangoma.

Asterisk for Raspberry Pi

Полная установка Asterisk и FreePBX для Raspberry Pi.

AstLinux

Проект AstLinux обслуживает сообщество, которое хочет запускать Asterisk на небольших, маломощных твердотельных устройствах. Размер установки всего решения измеряется в мегабайтах (AstLinux был первоначально разработан, чтобы поместиться на картах CompactFlash). Если вы очарованы маленькими компьютерами и хотите играть с АТС в коробке, которая помещается в вашем кармане, AstLinux может быть вам интересен.

Мы рекомендуем вам проверить их[5].

Установка Linux

Asterisk разрабатывается с использованием Linux, и если вам не очень удобно переносить программное обеспечение между различными платформами, то вам необходимо использовать Linux.

В этой книге мы будем использовать CentOS в качестве платформы. Если вы предпочитаете другой дистрибутив Linux - ожидается, что у вас есть достаточные навыки его администрирования для понимания некоторых различий. В наши дни так легко и дешево запустить экземпляр любого общего дистрибутива, что нет никакого реального вреда в использовании CentOS для обучения, а затем перейти на то, что предпочтете вы, когда будете готовы.

Мы рекомендуем установить минимальную версию CentOS, так как в процессе установки мы будем проходить через обработку всех необходимых условий. Это также гарантирует, что вы не устанавливаете ничего лишнего.

Выбор вашей платформы

Итак, строго говоря, мы уже выбрали вашу платформу, но есть несколько различных способов запустить сервер CentOS (см. Таблицу 3-1).

4 Elastix больше не является продуктом на основе Asterisk или open source.
5 После того, как вы прочтете нашу книгу, конечно же.

Таблица 3-1. Сравнение платформ Linux, подходящих для Asterisk

Платформа	Перспективы	Учтите
OpenStack (DigitalOcean, Linode, VULTR, etc.)	Через несколько минут все будет готово. Недорогая в эксплуатации. Не требует никаких ресурсов в вашей локальной системе. Доступна из любой точки мира. Может использоваться в продакшене. Фантастическая для быстрого прототипирования проектов.	Вы платите, пока она работает. IP-адрес является вашим только до тех пор, пока система работает. Требуются некоторые навыки DevOps, если вы хотите развернуть в продакшене. По умолчанию брандмауэр отсутствует.
VirtualBox (или другая ПК-образная платформа)	Бесплатна в использовании. Нет внешнего воздействия. Отлично подходит для небольших лабораторных проектов.	Требует больше лошадиных сил в вашей системе. Требуется место для хранения в локальной системе. Не так просто развернуть в рабочей среде.
AWS and/or Lightsail	Недорогая в эксплуатации. Не требует никаких ресурсов в вашей локальной системе. Доступна из любой точки мира. Может использоваться в производственной среде. Вес до огромных размеров.	Вы платите, пока она работает. Несколько больше навыков требуется, чтобы собрать все необходимые ресурсы.
Физическое оборудование	Выделенная платформа. Может быть загружена и установлена везде. Полный контроль над всеми аспектами окружающей среды, оборудования, сети и так далее.	Риск отказа компонентов. Потребляемая мощность. Шум. Потенциальные затраты на хостинг. Не присуща избыточность.
Другое (на самом деле все, что будет работать с CentOS 7, должно быть в порядке)	Вы можете использовать среду, с которой вы знакомы.	Вы сам себе хозяин.
Другие Linux (на самом деле вам не нужно запускать CentOS)	Вы можете запустить такую среду, которую захотите.	Вы должны иметь хорошие навыки администрирования Linux.

Для целей обучения мы рекомендуем один из двух простых способов начать работу:

- *Если вы используете Windows в качестве рабочего стола:* загрузите VirtualBox, затем загрузите CentOS 7 Minimal ISO и установите на локальном компьютере.
- *Если вам удобно работать с подключениями на основе SSH с ключами к удаленным системам:* создайте размещенную систему (например, DigitalOcean CentOS droplet).

Эта книга была разработана и протестирована с использованием как VirtualBox, так и DigitalOcean.

Шаги для VirtualBox

Загрузите Minimal ISO с веб-сайта Centos.

Возьмите копию VirtualBox с веб-сайта платформы и установите ее.

Загрузите PuTTY если используете Windows.

- Тип: Linux
- Версия: Red Hat (64-bit)
- Объем памяти: 2048 MB
- Жёсткий диск: Создать новый виртуальный жесткий диск
- Расположение файла: выберите подходящее место для хранения образов виртуальных машин
- Размер файла: 16 ГБ подходит для наших целей, но для продакшена потребуется куда больше

Создайте новую виртуальную машину со следующими характеристиками:

- Носители: под пунктом **Носители, Контроллер: IDE** ...
  1. Вы должны увидеть крошечный значок диска CD/DVD с надписью **Пусто**.
  2. Нажмите на него и справа под **Атрибуты** появится еще один значок крошечного диска.
  3. Нажмите на него, и он попросит вас **выбрать образ оптического диска**.
  4. Найдите на жестком диске Minimal ISO, загруженный с CentOS и выберите его.
  5. Теперь трей носителей должен отображать CentOS ISO.
- Сеть: Адаптер 1

Тип подключения: **Сетевой мост**

Как только базовая машина будет определена, вам нужно будет настроить ее следующим образом:

- Дата и время: при желании можно настроить часовой пояс.
- Сеть и имя хоста: **Ethernet** - переключите с off на **on** (он должен немедленно захватить IP-адрес из вашей сети; если нет, установите вручную). Нажмите кнопку **Готово**.
- Место установки: это может потребовать от вас подтверждения, но вам не нужно ничего менять. Нажмите кнопку **Готово**.
- Вот и все. **Начать установку**.

Во время установки установите пароль root, а также создайте пользователя с именем `astmin`. Сделайте `astmin` администратором.

Установка займет несколько минут. Выпейте кофе!

После завершения установки программа установки попросит вас перезагрузиться. Перезагрузка должна занять всего 15 секунд или около того.

Поздравляем, ваша система готова. Войдите в систему как `root`.

Linux (OpenStack) Host

Очевидно, вам понадобится учетная запись на хостинге Linux-провайдера, если вы собираетесь использовать этот метод (мы обнаружили, что предложения на основе OpenStack являются самыми дешевыми с предлагаемыми качеством/производительностью/простотой). Мы использовали DigitalOcean в течение многих лет, но также обнаружили, что Linode и VULTR являются отличными поставщиками услуг в этом пространстве[6]. После сортировки вы можете войти в систему и создать новую систему примерно следующим образом:

- CentOS 7 (последняя версия, 64-бит)
- 4 Гб 2vCPU (нам действительно не нужно 4 ГБ оперативной памяти, но хорошо иметь 2xCPU; вам, вероятно, может хватить 2 ГБ 1xCPU, если вы действительно сознательны в расходах)
- Ближайший к Вам датацентр

Как только это будет запущено, войдите в систему как пользователь по умолчанию (на момент написания этой статьи - это `centos`).

Обратите внимание, что экземпляры DigitalOcean по умолчанию не имеют брандмауера. Вместо этого они предоставляют его как часть своей среды. Таким образом, система, которую вы создадите, не будет иметь собственного файрволла и будет подвергаться внешним атакам сразу после завершения настройки (вы увидите это в консоли Asterisk). У разных провайдеров будут разные политики брандмауэра.

6 Новый сервис Lightsail от Amazon также обещает упростить создание размещенных Linux-машин.

Вы несете ответственность за то, чтобы ваш брандмауер работал правильно. Мы обсудим вопросы безопасности и борьбы с мошенничеством более подробно позже в этой книге.

Зависимости

Система, которую вы только что построили, на самом деле не намного больше, чем базовая загрузочная система. Для того, чтобы подготовить её к установке Asterisk, есть несколько вещей, которые нам нужно будет установить в первую очередь.

Следующие команды необходимо ввести из командной строки или добавить в простой скрипт оболочки и таким образом запустить.

```
sudo yum -y update &&
sudo yum -y install epel-release &&
sudo yum -y install python-pip &&
sudo yum -y install vim wget dnf&&
sudo pip install alembic ansible &&
sudo pip install --upgrade pip &&
sudo mkdir /etc/ansible &&
sudo chown astmin:astmin /etc/ansible &&
sudo echo "[starfish]" >> /etc/ansible/hosts &&
sudo echo "localhost ansible_connection=local" >> /etc/ansible/hosts &&
mkdir -p ~/ansible/playbooks
```

Мы установили Ansible просто потому, что это быстрый и простой способ получить все зависимости. Мы написали playbook для выполнения следующих операций:

1. Установка `dnf`, `vim`, `wget`, и `MySQL-python`.
2. Установка репозитория MySQL community-edition.
3. Установка `mysql-server`.
4. Настройка некоторых переменных при установке `mysql-server`.
5. Запуск демона mysql-server.
6. Изменение некоторых данных MySQL (создание пользователей, установка паролей).
7. Создание базы данных/схемы MySQL для использования в Asterisk.
8. Применение некоторых рекомендаций по безопасности (удаление анонимного пользователя, тестовая база данных и т.д.).
9. Создание пользователя `asterisk`.
10. Создание пользователя `astmin`.
11. Установка зависимостей для ODBC.
12. Установка некоторых диагностических инструментов.
13. Настройка брандмауэра для разрешения трафика SIP и RTP.
14. Редактирование некоторых файлов конфигурации ODBC.

Все это можно сделать вручную, но это просто много для набора и Ansible действительно хорош в оптимизации этого процесса.

Создайте плейбук Ansible в файле *~/ansible/playbooks/starfish.yml*.

Файл *libmyodbc8a.so* является версионным, поэтому, если у вас нет версии 8 UnixODBC:

1. Запустите playbook в первый раз (чтобы установить библиотеку UnixODBC).
2. Узнайте, какой файл был установлен в */usr/lib64/libmyodbc<version>a.so.*

3. Отредактируйте playbook соответствующим образом (укажите правильное имя файла).
4. Сохраните и повторно запустите playbook (который затем обновит файлы конфигурации, чтобы указать на правильную библиотеку).

Вот тебе playbook:

```
---
- hosts: starfish
  become: yes
  vars:
# Используйте это при первом запуске этого скрипта
    current_mysql_root_password: ""
    updated_mysql_root_password: "YouNeedAReallyGoodPassword"
    current_mysql_asterisk_password: ""
    updated_mysql_asterisk_password: "YouNeedAReallyGoodPasswordHereToo"
# Закомментируйте вышеизложенное после первого запуска

# Раскомментируйте строки ниже для последующих запусков этого скрипта
#    current_mysql_root_password: "YouNeedAReallyGoodPassword"
#    updated_mysql_root_password: "{{ current_mysql_root_password }}"
#    current_mysql_asterisk_password: "YouNeedAReallyGoodPasswordHereToo"
#    updated_mysql_asterisk_password: "{{ current_mysql_asterisk_password }}"

  tasks:

  - name: Install epel-release
    dnf:
      name: epel-release
      state: present

  - name: Install dependencies
    dnf:
      name: ['vim', 'wget', 'MySQL-python']
      state: present

  - name: Install the MySQL repo.
    dnf:
      name: http://repo.mysql.com/mysql-community-release-el7-5.noarch.rpm
      state: present

  - name: Install mysql-server
    dnf:
      name: mysql-server
      state: present

  - name: Override variables for MySQL (RedHat).
    set_fact:
      mysql_daemon: mysqld
      mysql_packages: ['mysql-server']
      mysql_log_error: /var/log/mysqld.err
      mysql_syslog_tag: mysqld
      mysql_pid_file: /var/run/mysqld/mysqld.pid
      mysql_socket: /var/lib/mysql/mysql.sock
    when: ansible_os_family == "RedHat"

  - name: Ensure MySQL server is running
    service:
      name: mysqld
      state: started
      enabled: yes

  - name: update mysql root pass for localhost root account from local servers
    mysql_user:
      login_user: root
      login_password: "{{ current_mysql_root_password }}"
      name: root
      host: "{{ item }}"
      password: "{{ updated_mysql_root_password }}"
```

```
  with_items:
      - localhost

- name: update mysql root password for all other local root accounts
  mysql_user:
    login_user: root
    login_password: "{{ updated_mysql_root_password }}"
    name: root
    host: "{{ item }}"
    password: "{{ updated_mysql_root_password }}"
  with_items:
      - "{{ inventory_hostname }}"
      - 127.0.0.1
      - ::1
      - localhost.localdomain

- name: create asterisk database
  mysql_db:
    login_user: root
    login_password: "{{ updated_mysql_root_password }}"
    name: asterisk
    state: present

- name: asterisk mysql user
  mysql_user:
    login_user: root
    login_password: "{{ updated_mysql_root_password }}"
    name: asterisk
    host: "{{ item }}"
    password: "{{ updated_mysql_asterisk_password }}"
    priv: "asterisk.*:ALL"
  with_items:
      - "{{ inventory_hostname }}"
      - 127.0.0.1
      - ::1
      - localhost
      - localhost.localdomain

- name: remove anonymous user
  mysql_user:
    login_user: root
    login_password: "{{ updated_mysql_root_password }}"
    name: ""
    state: absent
    host: "{{ item }}"
  with_items:
      - localhost
      - "{{ inventory_hostname }}"
      - 127.0.0.1
      - ::1
      - localhost.localdomain

- name: remove test database
  mysql_db:
    login_user: root
    login_password: "{{ updated_mysql_root_password }}"
    name: test
    state: absent

- user:
    name: asterisk
    state: present
    createhome: yes

- group:
    name: asterisk
    state: present

- user:
```

```
      name: astmin
      groups: asterisk,wheel
      state: present

  - name: Install other dependencies
    dnf:
      name:
        - unixODBC
        - unixODBC-devel
        - mysql-connector-odbc
        - MySQL-python
        - tcpdump
        - ntp
        - ntpdate
        - jansson
        - bind-utils
      state: present

#   Tweak the firewall for UDP/SIP
  - firewalld:
      port: 5060/udp
      permanent: true
      state: enabled

#   Tweak firewall for UDP/RTP
  - firewalld:
      port: 10000-20000/udp
      permanent: true
      state: enabled

  - name: Ensure NTP is running
    service:
      name: ntpd
      state: started
      enabled: yes

# The libmyodbc8a.so file is versioned, so if you don't have version 8, see what the
# /usr/lib64/libmyodbc<version>a.so file is, and refer to that instead
# on your 'Driver64' line, and then run the playbook again
  - name: update odbcinst.ini
    lineinfile:
      dest: /etc/odbcinst.ini
      regexp: "{{ item.regexp }}"
      line: "{{ item.line }}"
      state: present
    with_items:
      - regexp: "^Driver64"
        line: "Driver64 = /usr/lib64/libmyodbc8a.so"
      - regexp: "^Setup64"
        line: "Setup64 = /usr/lib64/libodbcmyS.so"

  - name: create odbc.ini
    blockinfile:
      path: /etc/odbc.ini
      create: yes
      block: |
        [asterisk]
        Driver = MySQL
        Description = MySQL connection to 'asterisk' database
        Server = localhost
        Port = 3306
        Database = asterisk
        UserName = asterisk
        Password = {{ updated_mysql_asterisk_password }}
        #Socket = /var/run/mysqld/mysqld.sock
        Socket = /var/lib/mysql/mysql.sock
...
```

Запустите playbook с помощью следующей команды:

```
$ ansible-playbook ~/ansible/playbooks/starfish.yml
```

Сядьте поудобнее и наблюдайте, как происходит волшебство.

Как только Ansible выполнит назначенные задачи, убедитесь что ODBC может подключиться к базе данных с использованием учетных данных пользователя `asterisk`.

```
$ echo "select 1" | isql -v asterisk asterisk password
```

Вы должны увидеть результат вроде этого:

```
+---------------------------------------+
| Connected!                            |
| sql-statement                         |
| help [tablename]                      |
| quit                                  |
+---------------------------------------+
SQL> select 1
+---------------------+
| 1                   |
+---------------------+
| 1                   |
+---------------------+
SQLRowCount returns 1
1 rows fetched
```

Если вы не видите сообщение `Connected!` - значит вам необходимо устранить неполадки в вашей базе данных и установке ODBC. Первое, что вы должны сделать - это убедиться, что можете войти в базу данных из командной строки с помощью пользователя `asterisk` (`mysql -u asterisk -p`). Большинство проблем ODBC, как правило, заканчиваются проблемами с учетными данными (т.е. неправильным паролем или именем пользователя), поэтому вернитесь назад, чтобы убедиться, что все учетные данные работают так, как должны, и дважды проверьте, что вы не получили никаких сообщений об ошибках от Ansible.

На момент написания этой статьи версия *jansson*, установленная из репозитория EPEL, является более старой версией, чем требуется для Asterisk, поэтому придется установить ее вручную.

Теперь система готова, и мы готовы загрузить и установить Asterisk.

Установка Asterisk

Asterisk официально поставляется в виде архива (как исходный код) и его необходимо загрузить, извлечь и скомпилировать[7]. Это не трудно сделать когда у вас удовлетворены все зависимости. Между написанием этой книги и вашим чтением ее, возможно, были некоторые изменения в различных зависимостях - поэтому вам процесс установки, возможно, придется запускать немного иначе. Часто бывает трудно понять разницу между сообщением об ошибке, которое можно безопасно проигнорировать, и сообщением, указывающим на критическую проблему; однако, как правило, вы должны были выявить и устранить все ошибки в предыдущих процессах, прежде чем приступать к этому шагу. Если ваши зависимости отсортированы, установка Asterisk будет проходить гладко.

Загрузка и необходимые компоненты

Выйдите из системы и снова войдите как пользователь `astmin`[8]. Введите следующие команды в командной строке чтобы загрузить исходный код Asterisk:

7 Обратите внимание, что участники сообщества также будут создавать пакетные версии Asterisk. Например, репозиторий EPEL поддерживает версию, которая может быть установлена с помощью `dnf (yum)`. На момент написания этой статьи официально поддерживается только версия tarball и мы рекомендуем этот метод в настоящее время, в основном из-за множества различных модулей, поставляемых с Asterisk и полезности в возможности создавать то, что вам нужно из исходников.

8 На экземпляре DigitalOcean вам нужно будет убедиться, что ваш SSH-ключ находится в файле */home/astmin/.ssh/authorized_keys*.

Когда вы видите, что мы указываем <TAB> в имени файла, то мы имеем в виду, что вы должны нажать клавишу Tab на клавиатуре и разрешить автозаполнение возможными вариантами. Затем следует остальная часть набора текста.

```
$ mkdir ~/src
$ cd ~/src
$ wget https://downloads.asterisk.org/pub/telephony/asterisk/asterisk-16-current.tar.gz
$ tar zxvf asterisk-16-current.tar.gz
$ cd asterisk-16.<TAB> # вкладка должна заполниться автоматически (если она не имеет более
одного совпадения)
```

Теперь мы можем выполнить несколько предварительных условий, определенных командой Asterisk, а также проверить среду:

```
$ cd contrib/scripts (или cd ~/src/asterisk-16.<TAB>/contrib/scripts)
$ sudo ./install_prereq install # у asterisk есть несколько предварительных условий, которые
это разрешает
$ cd ../..
$ ./configure --with-jansson-bundled
```

Asterisk теперь готов к компиляции и установке, но есть несколько настроек, которые стоит внести в конфигурацию перед компиляцией.

Компиляция и установка

```
$ make menuselect
```

Вы увидите меню, в котором представлены различные параметры, которые можно выбрать для компилятора. Для перемещения используйте клавиши со стрелками и Tab, а для выбора/отмены выбора - клавишу Enter. По большей части, значения по умолчанию должны быть в порядке, но мы хотим сделать несколько настроек для звуковых файлов, чтобы убедиться, что у нас есть все звуки, которые мы хотим и в лучшем формате.

На этом этапе вы также можете выбрать другие языки, которые захотите иметь в своей системе. Мы рекомендуем вам выбрать форматы WAV и G722 (а также G729, если вам необходимо его поддерживать).

Под Codec Translators (`--- External ---`):

- Выбрать `[*] codec_opus`
- Выбрать `[*] codec_silk`
- Выбрать `[*] codec_siren7`
- Выбрать `[*] codec_siren14`
- Выбрать `[*] codec_g729a`

Под Core Sound Packages:

- Убрать `[*] CORE-SOUNDS-EN-GSM`
- Выбрать `[*] CORE-SOUNDS-EN-WAV`
- Выбрать `[*] CORE-SOUNDS-EN-G722`

Под Extras Sound Packages:

- Выбрать `[*] EXTRA-SOUNDS-EN-WAV`
- Выбрать `[*] EXTRA-SOUNDS-EN-G722`

Save and Exit (Сохранить и выйти).

Еще три команды и Asterisk установлена:

```
$ make              # это займет несколько минут
                    # (в зависимости от скорости вашей системы)
$ sudo make install # вы должны запустить это с повышенными привилегиями
$ sudo make config  # и это
```

По завершению выполнения команды `make config` будут предложены некоторые команды для установки примеров файлов конфигурации. Для целей этой книги, *вы **не** должны этого делать*. Мы будем создавать необходимые файлы вручную - поэтому примеры файлов будут служить только тому, чтобы нарушить и запутать этот процесс. Сказав это, отметим что примеры файлов полезны, и мы будем упоминать их на протяжении всей этой книги, так как они являются отличным справочным материалом.

Перезагрузите систему.

Как только загрузка будет завершена - войдите в систему как пользователь `astmin` и временно установите SELinux на `Permissive` (после каждой загрузки он будет возвращаться к `Enforcing`, поэтому до тех пор, пока мы не разобрались с частью установки SELinux, это должно происходить на каждой загрузке):

```
$ sudo setenforce Permissive

$ sudo sestatus
```

Последняя команда должна показать `Current mode: permissive`

Убедитесь, что Asterisk работает со следующей командой:

```
$ ps -ef | grep asterisk
```

Вы можете увидеть, что демон `/user/sbin/asterisk` запущен (в настоящее время как пользователь `root`, но мы исправим это в ближайшее время). Asterisk теперь установлен и работает; однако, есть несколько параметров конфигурации, которые нам нужно сделать, прежде чем система будет полезна.

Начальная конфигурация

По умолчанию Asterisk сохраняет свои конфигурационные файлы в директории */etc/asterisk*. Сам процесс Asterisk для запуска не нуждается в каких-либо файлах конфигурации; однако он пока не будет использоваться, поскольку ни одна из функций, которые он предоставляет, не была указана. Сейчас мы займемся некоторыми задачами начальной настройки.

Файлы конфигурации Asterisk используют символ точки с запятой (`;`) для комментариев, главным образом потому, что хэш-символ (#) является допустимым символом на телефонной клавиатуре.

Файл *modules.conf* дает вам полный контроль над тем, какие модули Asterisk будут (и не будут) загружаться. Обычно нет необходимости явно определять каждый модуль в этом файле, но вы можете это сделать если захотите. Мы собираемся создать очень простой файл, как например:

```
$ sudo chown asterisk:asterisk /etc/asterisk ; sudo chmod 664 /etc/asterisk

$ sudo -u asterisk vim /etc/asterisk/modules.conf

[modules]
autoload=yes
preload=res_odbc.so
preload=res_config_odbc.so
```

Мы используем ODBC для загрузки многих конфигураций других модулей, и нам нужно, чтобы этот коннектор был доступен до того, как Asterisk попытается загрузить что-либо еще, поэтому мы предварительно загружаем его.

Далее, мы собираемся подкорректировать файл *logger.conf* предоставляемый по умолчанию.

```
$ sudo -u asterisk vim /etc/asterisk/logger.conf

[general]
exec_after_rotate=gzip -9 ${filename}.2;
[logfiles]
;debug => debug
;security => security
console => notice,warning,error,verbose
;console => notice,warning,error,debug
messages => notice,warning,error
full => notice,warning,error,debug,verbose,dtmf,fax
;full-json => [json]debug,verbose,notice,warning,error,dtmf,fax
;syslog keyword : This special keyword logs to syslog facility
;syslog.local0 => notice,warning,error
```

Вы заметите, что многие строки закомментированы. Они здесь для справки - потому что как выяснится при отладке системы вы можете часто настраивать этот файл. Мы выяснили, что лучше подготовить и закомментировать несколько строк, чем искать синтаксис каждый раз.

Следующий файл - *asterisk.conf*, определяющий различные директории, необходимые для нормальной работы, а также параметры, необходимые для запуска от имени пользователя `asterisk`:

```
$ sudo -u asterisk vim /etc/asterisk/asterisk.conf

[directories](!)
astetcdir => /etc/asterisk
astmoddir => /usr/lib/asterisk/modules
astvarlibdir => /var/lib/asterisk
astdbdir => /var/lib/asterisk
astkeydir => /var/lib/asterisk
astdatadir => /var/lib/asterisk
astagidir => /var/lib/asterisk/agi-bin
astspooldir => /var/spool/asterisk
[options]
astrundir => /var/run/asterisk
astlogdir => /var/log/asterisk
astsbindir => /usr/sbin
runuser = asterisk ; Пользователь для запуска от его имени. По умолчанию root.
rungroup = asterisk ; Группа для запуска от её имени. По умолчанию root
```

Мы настроим остальные файлы позже, а пока это все, что нам нужно на данный момент.

Давайте обновим права владельца на файлы, чтобы пользователь `asterisk` имел к ним надлежащий доступ.

```
$ sudo chown -R asterisk:asterisk {/etc,/var/lib,/var/spool,/var/log,/var/run}/asterisk
```

Также может понадобится добавить правило в директорию */etc/tmpfiles.d* для разрешения Asterisk создавать сокет во время выполнения.

```
$ sudo vim /etc/tmpfiles.d/asterisk.conf

d /var/run/asterisk 0775 asterisk asterisk
```

(Смотри `man tmpfiles.d` для большей информации)

Далее мы инициализируем базу данных таблицами, необходимыми Asterisk для настройки на основе ODBC.

Исходные файлы Asterisk включают в себя вклад, который люди Digium поддерживают как часть Asterisk для управления версиями необходимых таблиц базы данных. Это значительно упрощает поддержание корректности базы данных в процессе обновления.

Перейдите в соответствующий каталог и сделайте копию файла конфигурации.

```
$ cd ~/src/asterisk-16.<TAB>/contrib/ast-db-manage

$ cp config.ini.sample config.ini
```

Теперь мы собираемся открыть файл и предоставить ему учетные данные для нашей базы данных (которые определены в Ansible playbook с именем *starfish.yml*, под переменной `current_mysql_asterisk_password`, которую мы использовали в начале этой главы):

```
$ vim config.ini
```

Найдите строки, которые выглядят примерно так:

```
#sqlalchemy.url = postgresql://user:pass@localhost/asterisk
sqlalchemy.url = mysql://user:pass@localhost/asterisk

# Logging configuration
[loggers]
keys = root,sqlalchemy,alembic
```

Сделайте её копию, раскомментируйте и отредактируйте с правильными учетными данными:

```
#sqlalchemy.url = postgresql://user:pass@localhost/asterisk
#sqlalchemy.url = mysql://user:pass@localhost/asterisk
sqlalchemy.url = mysql://asterisk:YouNeedAReallyGoodPasswordHereToo@localhost/asterisk

# Logging configuration
[loggers]
keys = root,sqlalchemy,alembic
```

Теперь, с этой очень простой конфигурацией мы можем использовать Alembic для автоматической настройки базы данных идеальной для Asterisk (это было несколько болезненно делать в прошлых версиях Asterisk):

```
$ alembic -c ./config.ini upgrade head
```

Alembic не используется Asterisk - поэтому конфигурация, которую вы только что выполнили, не позволяет Asterisk использовать базу данных. Все, что он делает, это запускает скрипт, который создает схему и таблицы, которые будет использовать Asterisk (вы также можете сделать это вручную, но поверьте нам - лучше чтобы Alembic справился с этим). Это часть процесса установки/обновления. *Это особенно полезно, если у вас есть живые таблицы с реальными данными в них, и вы хотите обновить и сохранить соответствующую конфигурацию.*

Войдите в базу данных и просмотрите все созданные таблицы:

```
$ mysql -u asterisk -p

mysql> use asterisk;

mysql> show tables;
```

Вы должны увидеть список, похожий на этот:

```
| alembic_version_config      |
| extensions                  |
| iaxfriends                  |
| meetme                      |
| musiconhold                 |
| ps_aors                     |
| ps_asterisk_publications    |
| ps_auths                    |
| ps_contacts                 |
| ps_domain_aliases           |
| ps_endpoint_id_ips          |
| ps_endpoints                |
| ps_globals                  |
| ps_inbound_publications     |
| ps_outbound_publishes       |
| ps_registrations            |
| ps_resource_list            |
```

```
| ps_subscription_persistence |
| ps_systems                  |
| ps_transports               |
| queue_members               |
| queue_rules                 |
| queues                      |
| sippeers                    |
| voicemail                   |
```

Мы пока не собираемся ничего настраивать в базе данных. Сначала нам нужно сделать еще несколько базовых настроек.

Выйдите из CLI базы данных.

Теперь, когда у нас есть структура базы данных для обработки конфигурации Asterisk - мы расскажем Asterisk, как подключиться к базе данных.

```
$ sudo -u asterisk vim /etc/asterisk/res_odbc.conf
```

Еще раз - вам понадобятся учетные данные, которые вы определили в своем Ansible playbook.

```
[asterisk]
enabled => yes
dsn => asterisk
username => asterisk
password => YouNeedAReallyGoodPasswordHereToo
pre-connect => yes
```

Настройки SELinux

Мы собираемся установить некоторые инструменты SELinux и внести изменения в конфигурацию SELinux для правильной загрузки системы.

Общий подход состоит в том, чтобы просто отредактировать */etc/selinux/config* и установить `enforcing=disabled`. Мы не рекомендуем этого делать, т.к. это полностью отключает SELinux и делает следующие шаги ненужными.

```
$ sudo dnf -y install setools setroubleshoot setroubleshoot-server
```

```
$ sudo vim /etc/selinux/config
```

Вы собираетесь изменить строку `SELINUX=enforcing` на `SELINUX=permissive`. Это гарантирует, что файлы журналов будут показывать потенциальные ошибки SELinux, не блокируя соответствующие процессы.

Далее мы передадим Asterisk право собственности на файл */etc/odbc.ini*.

```
$ sudo chown asterisk:asterisk /etc/odbc.ini
```

```
$ sudo semanage fcontext -a -t asterisk_etc_t /etc/odbc.ini
```

```
$ sudo restorecon -v /etc/odbc.ini
```

```
$ sudo ls -Z /etc/odbc.ini
```

Если все хорошо, то вы должны увидеть, что контекст для этого файла был установлен в `asterisk_etc_t`:

```
-rw-r--r--. asterisk asterisk system_u:object_r:asterisk_etc_t:s0 /etc/odbc.ini
```

Есть еще несколько ошибок SELinux, которые мы видели здесь во время написания книги. Они могут быть исправлены к тому времени, когда вы читаете это, но не должно быть никакого вреда в их появлении:

```
$ sudo /sbin/restorecon -vr /var/lib/asterisk/*
```

```
$ sudo /sbin/restorecon -vr /etc/asterisk*
```

Перезагрузите систему и проверьте журнал на наличие каких-либо ошибок SELinux, прежде чем мы установим его в `enforcing`.

```
$ sudo grep -i sealert /var/log/messages
```

Там может быть несколько сообщений, жалующихся на то, что Asterisk не нужно (например, скрытый файл с именем *.odbc.ini*), но до тех пор, пока он не полон ошибок о всевозможных важных компонентах Asterisk, всё должно быть хорошо. Последнее, что вам нужно изменить - это SELinux Boolean, позволяющее Asterisk создавать TTY.

```
$ sudo setsebool -P daemons_use_tty 1
```

Отредактируйте файл */etc/selinux/config* еще раз, на этот раз установив `SELINUX=enforcing`. Сохраните и перезагрузитесь еще раз.

Убедитесь что Asterisk запущен (от пользователя `asterisk`).

```
$ ps -ef | grep asterisk

asterisk 3992 3985 0 16:38 ? 00:00:01 /usr/sbin/asterisk -f -vvvg -c
```

Хорошо, мы почти закончили с установкой.

Настройки брандмауера

Мы сделаем пару настроек брандмауера чтобы подготовить нашу систему для SIP (и безопасности SIP).

```
$ sudo firewall-cmd --zone=public --add-service=sip –permanent

$ sudo firewall-cmd --zone=public --add-service=sips --permanent
```

Финальные настройки

Ваша система Asterisk готова к запуску.

Давайте поместим некоторые исходные данные в конфигурационные файлы, чтобы в следующей главе мы могли начать работать с нашей новой системой Asterisk.

Поскольку мы собираемся использовать канал PJSIP для всех вызовов, то укажем Asterisk искать конфигурацию PJSIP в базе данных:

```
$ sudo -u asterisk vim /etc/asterisk/sorcery.conf

[res_pjsip] ; Мастер настройки Realtime PJSIP
; в конечном итоге больше модулей будут использовать волшебство
; для подключения к базе данных, но в настоящее время только PJSIP использует это
endpoint=realtime,ps_endpoints
auth=realtime,ps_auths
aor=realtime,ps_aors
domain_alias=realtime,ps_domain_aliases
contact=realtime,ps_contacts

[res_pjsip_endpoint_identifier_ip]
identify=realtime,ps_endpoint_id_ips

$ sudo -u asterisk vim /etc/asterisk/extconfig.conf

[settings] ; старый механизм для подключения всех других модулей к базе данных
ps_endpoints => odbc,asterisk
ps_auths => odbc,asterisk
ps_aors => odbc,asterisk
ps_domain_aliases => odbc,asterisk
ps_endpoint_id_ips => odbc,asterisk
ps_contacts => odbc,asterisk

$ sudo -u asterisk vim /etc/asterisk/modules.conf
```

```
[modules]
autoload=yes
preload=res_odbc.so
preload=res_config_odbc.so
noload=chan_sip.so ; устаревший SIP-модуль с прошлых дней
```

Теперь мы должны поместить один бит конфигурации в файл *pjsip.conf*, определяющий механизм транспорта.

```
$ sudo -u asterisk vim /etc/asterisk/pjsip.conf

[transport-udp]
type=transport
protocol=udp
bind=0.0.0.0
```

Наконец, давайте войдем в базу данных и определим некоторые примеры конфигураций для PJSIP:

```
$ mysql -D asterisk -u asterisk -p

mysql>

insert into asterisk.ps_aors (id, max_contacts) values ('0000f30A0A01', 1);
insert into asterisk.ps_aors (id, max_contacts) values ('0000f30B0B02', 1);
insert
  into asterisk.ps_auths
  (id, auth_type, password, username)
  values
  ('0000f30A0A01', 'userpass', 'not very secure', '0000f30A0A01');
insert
  into asterisk.ps_auths
  (id, auth_type, password, username)
  values
  ('0000f30B0B02', 'userpass', 'hardly to be trusted', '0000f30B0B02');
insert
  into asterisk.ps_endpoints
  (id, transport, aors, auth, context, disallow, allow, direct_media)
  values
  ('0000f30A0A01', 'transport-udp', '0000f30A0A01', '0000f30A0A01',
  'sets', 'all', 'ulaw', 'no');
insert
  into asterisk.ps_endpoints
  (id, transport, aors, auth, context, disallow, allow, direct_media)
  values
  ('0000f30B0B02', 'transport-udp', '0000f30B0B02', '0000f30B0B02',
  'sets', 'all', 'ulaw', 'no');
exit
```

Давайте перезагрузимся, а затем войдем в нашу новую систему Asterisk и посмотрим, что мы создали.

Проверка вашей новой системы Asterisk

На этом этапе нам не нужно слишком глубоко погружаться в систему, поскольку все последующие главы будут делать именно это.

Поэтому все, что нам нужно сделать сейчас - это проверить что мы можем войти в систему и наличие созданных конечных точек PJSIP.

```
$ sudo asterisk -rvvvv+

*CLI> pjsip show endpoints
```

Вы должны увидеть две конечные точки, которые мы создали, представленные ниже:

```
Endpoint: <Endpoint/CID.....................................> <State.....> <Channels.>
   I/OAuth: <AuthId/UserName.....................................................>
       Aor: <Aor.........................................> <MaxContact>
     Contact: <Aor/ContactUri.........................> <Hash....> <Status> <RTT(ms)..>
 Transport: <TransportId........> <Type> <cos> <tos> <BindAddress..................>
```

```
   Identify: <Identify/Endpoint.....................................................>
        Match: <criteria.......................>
    Channel: <ChannelId.....................................> <State.....> <Time.....>
        Exten: <DialedExten...........> CLCID: <ConnectedLineCID.......>
==========================================================================================
 Endpoint: 0000f30A0A01 Not in use 0 of inf
     InAuth: 1/0000f30A0A01
  Transport: transport-udp udp 0 0 0.0.0.0:5060

 Endpoint: 0000f30B0B02 Unavailable 0 of inf
     InAuth: 2/0000f30B0B02
  Transport: transport-udp udp 0 0 0.0.0.0:5060
Objects found: 2
```

Если вы не видите две конечные точки в списке - значит имеется проблема с конфигурацией. Вам придется работать в обратном направлении, чтобы убедиться в отсутствии ошибок, мешающих Asterisk подключаться к базе данных и создавать экземпляры этих двух конечных точек.

Распространенные ошибки установки

Следующие условия (в произвольном порядке) вызывают большинство ошибок установки:

Синтаксическая ошибка

В некоторых случаях замены табуляции на пробел может быть достаточно, чтобы что-то сломать. UnixODBC, например, оказался чувствительным к отсутствию пробелов между определениями `key = value`. Лучший совет, который мы можем здесь дать - это использовать копирование/вставку, когда это возможно, в отличие от ручного ввода.

Проблемы с разрешениями

Они могут раздражать, но сообщения об ошибках, как правило, содержат необходимы подсказки. Файл */var/log/messages* часто является золотой жилой для полезных подсказок.

Недостающие шаги

Пропущенный шаг может не иметь заметных эффектов до тех пор, пока не пройдет много шагов. Дважды проверяйте все и проверяйте функциональность, прежде чем двигаться дальше.

Проблемы с учетными данными

Всегда проверяйте вручную, что созданные пользователи и пароли работают, прежде чем использовать их в файле конфигурации.

Невозможно и не нужно копаться в каждом предупреждении и сообщении об ошибке, которое вы можете увидеть, но если мы предусмотрели тестовый запуск и он не производит ничего, как мы указали, вы, вероятно, должны снова пройти этот шаг, пока не выясните что происходит.

Некоторые финальные заметки по конфигурации

После установки Asterisk создаст среду для себя на вашей Linux-машине. В следующих разделах приведены некоторые полезные сведения о том, как можно взаимодействовать с новой установкой Asterisk.

Примеры файлов конфигурации для дальнейшего использования

Несмотря на то, что мы предупреждали вас не запускать команду `sudo make samples` во время установки (потому что это заполнит ваш каталог */etc/asterisk* кучей вещей, которые вам не нужны), файлы примеров тем не менее являются фантастической ссылкой. В директории с исходниками Asterisk вы найдете их в двух следующих каталогах:

```
~/src/asterisk-16.<TAB>/configs/basic-pbx
~/src/asterisk-16.<TAB>/configs/samples
```

Файлы в этих папках стоит прочитать (особенно для любого модуля, с которым вы работаете и хотите исследовать, как что-то сделать).

Прочтите их когда у вас появится возможность.

Запуск `make samples` в системе, в которой уже есть файлы конфигурации, приведет к перезаписи существующих файлов.

Командная оболочка Asterisk

Asterisk можно запустить как демон или как приложение. В общем, вы можете запустить его как приложение, когда создаете, тестируете и устраняете неполадки, или как демон при запуске его в продакшн.

Команда запуска Asterisk одинакова независимо от того, выполняется ли он как демон или как приложение:

```
asterisk
```

Однако без каких-либо аргументов эта команда будет принимать определенные значения по умолчанию и запускать Asterisk в качестве фонового приложения. Другими словами, не стоит запускать команду `asterisk` самостоятельно, а лучше передавать ей некоторые параметры, чтобы лучше определить поведение, которое вы ожидаете. В следующем списке приведены некоторые примеры общего использования:

`-h`

Эта команда отображает список полезных возможных параметров для использования. Для получения полного списка всех параметров и их описаний выполните команду **`man asterisk`**.

`-c`

Эта опция запускает Asterisk как приложение (на переднем плане). Это означает, что Asterisk привязан к сеансу пользователя. Другими словами, если вы закроете сеанс пользователя, выйдя из системы или потеряв соединение - Asterisk умрет. Этот параметр обычно используется при создании, тестировании и отладке, но его не следует использовать в рабочей среде. Если вы запустили Asterisk таким образом, введите **`core stop now`** в командной строке CLI, чтобы остановить Asterisk и выйти.

`-v, -vv, -vvv, -vvvv`, *и т.д.*

Эта опция может использоваться с другими опциями (например `-cvvv`) для увеличения детализации вывода консоли. Она делает точно то же самое, что и команда CLI `core set verbose` *n*, где *n*-любое целое число между 0 и 5 (любое целое число больше 5 будет работать, но не будет предоставлять больше подробностей). Иногда полезно вообще не задавать уровень детализации. Например, если вы хотите видеть только ошибки запуска, уведомления и предупреждения, отключение детализации предотвратит отображение всех других сообщений запуска.

`-d, -dd, -ddd, -dddd`, и т.д.

Этот параметр можно использовать так же, как и `-v`, но вместо обычного вывода он будет указывать уровень вывода отладки (что в первую очередь полезно для разработчиков, устраняющих проблемы с кодом). Также необходимо включить вывод отладочной информации в файл *logger.conf* (который мы рассмотрим более подробно в Главе 21).

`-r`

Эта команда необходима если вы хотите подключиться к CLI процесса Asterisk, работающего как демон. Вы, вероятно, будете использовать этот параметр больше чем любой другой для систем Asterisk, находящихся в продакшене. Этот параметр будет работать только в том случае, если у вас

уже запущен демонизированный экземпляр Asterisk. Чтобы выйти из интерфейса командной строки при использовании этого параметра, введите **`exit`**.

`-T`

Эта опция добавит метку времени к выводу CLI.

`-x`

Эта команда позволяет передать строку в Asterisk, которая будет выполнена так, как если бы она была введена в CLI. Например, чтобы получить быстрый список всех используемых каналов без необходимости запуска консоли Asterisk, просто введите **`asterisk -rx 'core show channels'`** из оболочки, и получите результат в котором нуждаетесь.

`-g`

Этот параметр указывает Asterisk сделать дамп ядра, если оно аварийно завершает работу.

Мы рекомендуем вам попробовать несколько комбинаций этих команд чтобы увидеть их в действии.

safe_asterisk

При установке Asterisk с помощью директивы `make config` создается скрипт *safe_asterisk*, который запускается в процессе `init` Linux при каждой загрузке.

Скрипт *safe_asterisk* предоставляет следующие преимущества:

- Автоматическая перезагрузка Asterisk после аварии
- Можно настроить отправку электронной почты администратору в случае сбоя
- Определяет, где хранятся файлы сбоев (по умолчанию */tmp*)
- При сбое выполняет скрипт

Вам не нужно знать слишком много об этом скрипте, кроме понимания того, что он должен работать. В большинстве сред этот скрипт отлично работает в формате по умолчанию.

Вывод

В этой главе мы представили кураторский пример того, как должна проходить установка Asterisk. Мы выбрали дистрибутив Linux и сервер MySQL для вас ради краткости, но отметили, что Asterisk на самом деле довольно гибок в таких вопросах. Теперь у нас есть прочный фундамент, на котором можно построить систему Asterisk. В следующих главах мы рассмотрим, как подключить устройства к нашей системе Asterisk для того, чтобы начать совершать вызовы внутри и работать над все более сложными концепциями в последующих главах (таких как видеоконференции и WebRTC).

Глава 4. Сертификаты для безопасности конечных точек

Нам должно повезти только один раз. Тебе должно везти каждый раз.
– IRA к Маргарет Тэтчер после неудачной попытки убийства
Если вы действительно хотите что-то сделать, вы найдете способ. Если вы этого не сделаете - вы найдете оправдание.
– Джим Рон

Неудобство безопасности

Безопасность VoIP можно рассматривать как две отдельные (но взаимосвязанные) проблемы:

- Защита система от мошенничества с платными услугами (что, как правило, является целью попыток атак на основе SIP)
- Защита системы от перехвата вызовов (что касается конфиденциальности, а также улучшения защиты от мошенничества)

Конечно, есть много других аспектов безопасности вашей системы, но большинство из них являются общими понятиями безопасности, не специфичными для VoIP.

В этой главе мы сосредоточимся на области безопасности, которая слишком часто упускается из виду, а именно на создании и применении сертификатов и ключей для обеспечения безопасности связи между конечными точками и вашей системой. В SIP-связи шифрование является необязательным (и, к сожалению, не используется большую часть времени). В WebRTC же это необходимо.

Эта глава ни в коем случае не должна рассматриваться как последнее слово в защите вашей системы Asterisk; в Главе 22 будет рассмотрено больше. Однако мы надеемся, что это обеспечит вам прочную основу для построения безопасного решения.

Безопасность SIP

Если вы создадите какой-либо сервер, доступный из интернета, и подождёте несколько часов после его включения, то заметите, что система уже привлекла зонды, пытающиеся определить, есть ли у нее какие-либо уязвимые SIP-службы. Вскоре вы заметите, что все большее количество атак на сервер пытается поставить под угрозу безопасность учетной записи. Поздравляем — ваш сервер был автоматически добавлен в списки SIP-серверов, используемых преступниками с целью мошенничества. Если одно из этих вторжений удастся - скомпрометированная платформа, скорее всего, станет частью организованной преступной сети и вы оплатите счет за неотслеживаемые звонки в различные дорогостоящие пункты назначения.

Мы не шутим здесь; не пренебрегайте вашей безопасностью SIP или вы, вероятно, окажетесь жертвой очень дорогой атаки мошенничества.

Имена подписчиков

Абонентская часть учетных данных SIP (`userinfo` часть URI) слишком часто устанавливается как добавочный номер. Эта практика хороша для целей адресации вызовов, но совсем не рекомендуется для целей аутентификации конечной точки. Имя подписчика конечных точек не должно иметь

значения за пределами организации. MAC-адрес идеально подходит для этого. Без фактического адреса для исследования работа злоумышленника станет намного сложнее.

Вы увидите множество УАТС типа SIP (включая на основе Asterisk, к сожалению), которые назначают добавочный номер учетным данным телефона. В этой книге вы увидите, что добавочный номер не является частью учетных данных телефона. В этом есть несколько преимуществ, но с точки зрения безопасности преимущество заключается в том, что если злоумышленник знает внутренний номер - он не имеет никаких представлений о том, как аутентифицировать вызов через систему.

Таким образом, у вас может быть пользователь с SIP-адресом `100@shifteight.org`, которого система Asterisk свяжет с устройством по адресу `0000f3101010@yourpbx.com`. Когда вызов направлен на 100 - будет звонить `0000f3101010`, но вызывающий абонент никогда ничего не знает об этой конечной точке.

На протяжении всей этой книги вы увидите, что мы установим связь между внутренним номером (расширением) (который, по нашему мнению, должен быть связан с пользователем или услугой) с идентификатором устройства (который может быть устройством SIP или номером телефона) и что для их связи может использоваться простая таблица (и впоследствии повысит как безопасность, так и гибкость).

Безопасность SIP-сигнализации

По умолчанию сообщения SIP передаются в открытом виде без эффективной защиты. При атаке человек посередине (MITM) злоумышленник способен получить всевозможную информацию о Ваших звонках. Протокол защиты транспортного уровня (TLS) используется для минимизации этого риска.

О том, как настроить устройства на использование TLS, мы поговорим в следующей главе. Все, что нам нужно сделать сейчас — это создать сертификаты.

Существует три распространенных способа создания сертификатов. Мы приведем примеры для двух из них (самозаверенные - self-signed и LetsEncrypt), но оставим осуществление получения официально выданных сертификатов читателю.

Самозаверенные сертификаты

Основное преимущество самозаверенного сертификата заключается в том, что его не нужно проверять с помощью внешнего объекта. Недостатком же является недоверие внешних объектов из-за этого.

Если вы защищаете устройства SIP только для использования в контролируемой сетевой среде - самозаверенный сертификат может быть тем, что вам нужно. Он не считается лучшим подходом, но в некоторых случаях его может быть достаточно, и это, как правило, лучше чем ничего.

В этом мире, полном автоматизированной преступности, многое можно узнать о конфиденциальности и безопасности, а также криптографии, необходимой для обоих сторон. Тем не менее, это книга об Asterisk, так что мы собираемся предоставить шаблон для создания необходимых компонентов, и вам необходимо продолжить исследование самостоятельно.

Инструментарий `openssl` предоставляет инструмент, который сделает работу за нас.

Мы запустим его следующим образом:

```
$ sudo su asterisk -
$ mkdir /home/asterisk/certs
$ openssl req -x509 -nodes -newkey rsa:2048 -days 3650 \
-keyout /home/asterisk/certs/self-signed.key \
-out /home/asterisk/certs/self-signed.crt
```

Вам будет предложено предоставить некоторую информацию, а затем ваш ключ и сертификат будут записаны в папку */home/asterisk/certs*.

Вы можете добавить следующее в команду, чтобы обойти вопросы (измените информацию в соответствии с вашей ситуацией):

```
-subj "/C=CA/ST=Ontario/L=Toronto/O=ShiftEight/CN=shifteight.org"
```

Полная команда будет выглядеть примерно так:

```
$ openssl req -x509 -nodes -newkey rsa:2048 -days 3650 \
> -keyout /home/asterisk/certs/self-signed.key \
> -out /home/asterisk/certs/self-signed.crt \
> -subj "/C=CA/ST=Ontario/L=Toronto/O=ShiftEight/CN=shifteight.org"
```

Это создаст самозаверенный сертификат и закрытый ключ и сохранит их в */home/asterisk/certs/*. Мы сможем использовать их позже - когда будем настраивать наши конечные точки SIP.

Вероятно, это хорошая идея, чтобы применить `chmod` к вашим сертификатам так, чтобы только соответствующий пользователь/группа могли получить к ним доступ:

```
$ chmod 640 /home/asterisk/certs/*
```

Выйдите из аккаунта пользователя asterisk.

```
$ exit
$ who am i # You should be astmin again.
```

Существует альтернатива использованию самозаверенных сертификатов: если у вас есть доменное имя, назначенное вашему серверу, доступному из общедоступного интернета, вы можете создать проверенный сертификат с помощью LetsEncrypt. Читайте дальше.

Сертификаты LetsEncrypt

Если вы заинтересованы в безопасной связи через общедоступный Интернет (которому вы будете доверять), то наличие сертификатов домена, предоставляемых Центром сертификации (ЦС) полезно.

Проект LetsEncrypt предоставляет цифровые сертификаты бесплатной проверки домена (Domen Validation - DV). Бесплатный инструмент, предоставляемый Let's Encrypt Foundation под названием certbot, позволяет автоматизировать получение и обслуживание доверенных сертификатов.

Как минимум, вашему серверу потребуется полное доменное имя (FQDN), которое сопоставляется с внешним IP-адресом, поступающим на компьютер. Любой брандмауэр между ними должен беспрепятственно передавать трафик для этого имени хоста в систему, для которой вы получаете сертификат. Если вы не можете сделать это по какой-либо причине - получение доверенного сертификата становится более сложным (и выходит за рамки этой книги).

`certbot` может быть установлен посредством `yum` следующим образом:

```
$ sudo yum -y install certbot
```

После его установки - Вам просто нужно выполнить следующее:

```
$ sudo certbot certonly
How would you like to authenticate with the ACME CA?
-------------------------------------------------------------------------------
1: Spin up a temporary webserver (standalone)
2: Place files in webroot directory (webroot)
-------------------------------------------------------------------------------
Select the appropriate number [1-2] then [enter] (press 'c' to cancel): 1
```

Если у вас работает веб-сервер или вы уверены в Варианте 2 - это нормально, но эти шаги предполагают, что у вас нет веб-сервера и, следовательно, вам необходимо/требуется использовать встроенный временный веб-сервер, который `certbot` будет использовать для аутентификации. Этот сервер используется для подтверждения управления доменом, для которого запрашивается сертификат.

При необходимости ответьте на следующие вопросы, а затем вставьте имя хоста, назначенное IP-адресу сервера:

```
Please enter in your domain name(s) (comma and/or space separated)  (Enter 'c'
to cancel): asteriskbook.shifteight.org
```

(замените *asteriskbook.shifteight.org* на назначенное доменное имя).

`certbot` будет выполнять свою магию, и если все прошло хорошо - вы должны получить сообщение похожее на следующее:

```
IMPORTANT NOTES:
 - Congratulations! Your certificate and chain have been saved at:
   /etc/letsencrypt/live/asteriskbook.shifteight.org/fullchain.pem
   Your key file has been saved at:
   /etc/letsencrypt/live/asteriskbook.shifteight.org/privkey.pem
   Your cert will expire on 2018-07-23. To obtain a new or tweaked
   version of this certificate in the future, simply run certbot
   again. To non-interactively renew *all* of your certificates, run
   "certbot renew"
 - Your account credentials have been saved in your Certbot
   configuration directory at /etc/letsencrypt. You should make a
   secure backup of this folder now. This configuration directory will
   also contain certificates and private keys obtained by Certbot so
   making regular backups of this folder is ideal.
 - If you like Certbot, please consider supporting our work by:
   Donating to ISRG / Let's Encrypt:   https://letsencrypt.org/donate
   Donating to EFF:                    https://eff.org/donate-le
```

> Не забудьте сделать пожертвование Internet Security Research Group (ISRG) или Electronic Frontier Foundation (EFF); они выполняют важную работу и заслуживают нашей поддержки.

Теперь у вас есть сертификаты, необходимые для включения различных служб TLS в вашей системе. Мы применим их в следующей главе.

Не слишком сложно, а?

Имейте в виду - большинство сертификатов, получаемых из внешнего источника, будут иметь срок действия. В случае LetsEncrypt текущий срок действия составляет три месяца. Если вы собираетесь запустить сертификаты в продакшен - Вам нужно понять как ими управлять (например, автоматизировать обновление, где люди из LetsEncrypt проделали хорошую работу по упрощению).

Приобретение сертификатов в официальном центре сертификации

Если сертификаты LetsEncrypt не обеспечивают требуемый уровень проверки (например, если вам нужна проверка организации (OV) или расширенная проверка (EV)), вам нужно будет получить услуги центра сертификации, предоставляющего такие вещи. Эти вопросы выходят за рамки данной книги.

Если вы проработали примеры для разделов самозаверенных и LetsEncrypt сертификатов - у Вас будет хотя бы базовое представление о некоторых процессах получения сертификатов из центра сертификации, так как многие шаги будут аналогичны.

Защита медиапотока

Сертификаты, которые мы получили, могут быть использованы для защиты как нашей сигнализации, так и самой полезной нагрузки (т.е. того, что произносится или передаваемого видео). Обратите внимание, что механизмы для безопасности сигналов - это вещь протокола SIP, а механизмы

обеспечения безопасности медиапотока - являются вещью протокола RTP. Имейте в виду, что шифрование сигнализации SIP не означает, что вы автоматически также шифруете медиатрафик (RTP).

Шифрование RTP

Шифрование протокола реального времени позволит достичь эффекта защиты наших медиа-потоков.

Для обеспечения шифрования медиа обычно используются два механизма: SDES и DTLS-SRTP. SDES — это механизм шифрования мультимедиа, доверяющего безопасности сигнализации. Другими словами, если вы используете TLS для защиты сигнализации SIP, то SDES, вероятно, также обрабатывает шифрование мультимедиа.

DTLS-SRTP, с другой стороны, не доверяет сигнализации. Это важно, потому что стандарт WebRTC требует чтобы медиапоток был зашифрован таким образом.

Сертификаты, созданные нами здесь, должны работать в обоих сценариях. В следующих главах, когда мы будем настраивать конечные точки SIP или WebRTC, мы рассмотрим более подробно как использовать сертификаты. На данный момент достаточно того, что мы сгенерировали сертификаты и сделали их доступными для использования.

Вывод

Не ошибитесь: безопасность все усложняет. В старые добрые времена вы могли запустить SIP-соединение с полудюжиной строк конфигурации и назвать это днем. Это больше не летает, и хотя этот тип конфигурации по-прежнему будет работать (просто используйте UDP вместо TLS, и все, что вам нужно — это пароль), мы решили что начиная с этого выпуска все примеры конфигурации будут выбирать более безопасные параметры где это возможно. Мы не претендуем на то, чтобы представить последнее слово о безопасности VoIP, но мы собираемся привести примеры, которые заплатят больше, чем эта концепция на словах.

Далее мы обсудим, как настроить конечные точки в системе Asterisk (используя ключи и сертификаты, мы только что созданные нами).

Глава 5. Конфигурация пользовательских устройств

Я не всегда знаю о чем говорю, но я знаю, что я прав.
— Мухаммед Али

Пришло время подключить некоторые пользовательские устройства SIP к Asterisk. Хотя мы собираемся сосредоточиться на Asterisk как конце вещей, имейте в виду, что определение канала устройства в Asterisk для непосредственного подключения через него – это только половина конфигурации; вам также нужно настроить другой конец — само устройство (обычно телефон) так, чтобы оно знало куда отправлять свои вызовы.

Важное замечание относительно точки SIP

Настройка другого конца связи SIP конечно необходима, но не является частью конфигурации Asterisk и, в конечном счете, выходит за рамки этой книги. На рынке должно быть тысячи различных типов конечных точек SIP, включая настольные телефоны, программные телефоны, УАТС, прокси-серверы, серверы конференций и всевозможные другие продукты. У каждого производителя есть свои инструменты, позволяющие настраивать его продукцию (а некоторые из них требуют обширных знаний). SIP - это сложный протокол. Сказав это, большинство настольных телефонов SIP имеют какой-то веб-интерфейс, и большинство софтфонов имеют меню конфигурации, встроенное в их графический интерфейс.

В самом простом случае настройка SIP-устройства включает в себя предоставление трех параметров:

- Адрес сервера, через который оно будет подключаться (ваш сервер Asterisk)
- Имя пользователя (которое также можно назвать именем подписчика, расширением или чем-то подобным)
- Пароль

Хотя каждый тип конечной точки будет отличаться, все они будут следовать аналогичному соглашению, и хотя потенциально существуют сотни параметров конфигурации, довольно часто можно настроить только эти три вещи.

Другими словами: есть две отдельные задачи, необходимые для настройки устройства для работы с Asterisk:

- Сообщение Asterisk об устройстве (настройка учетных данных канала в Asterisk)
- Сообщение устройству об Asterisk (доступ к инструментам настройки для устройства и указание ему нахождение сервера и как к нему подключиться)

Некоторые мысли о протоколе SIP

SIP - является одноранговым протоколом, и хотя обычно используется настройка, в которой конечные точки (телефоны) ведут себя как клиенты, а какой-то шлюз (например, Asterisk) обеспечивает маршрутизацию и функции, сам протокол работает с точки зрения одноранговых отношений (Рисунок 5-1). Две конечные точки SIP могут напрямую разговаривать друг с другом (т.е. пара SIP-телефонов теоретически должна иметь возможность создавать своего рода “интерком” непосредственно между собой без АТС посередине).

При этом, безусловно, большинство SIP-транзакций происходят через какой-либо сервер,

который обычно остается в пути вызова и обеспечивает все соединения (также не требуется протоколом). Когда SIP-вызов выполняется с телефона на другой телефон через Asterisk - на самом деле происходит два вызова: вызов от исходного аппарата в Asterisk и другой отдельный вызов из Asterisk в целевой аппарат (этот второй этап вызова может даже не использовать SIP). Asterisk соединяет их вместе. Аналогично, если вы делаете „внешний“ вызов - Asterisk примет вызов от вашего аппарата, а затем отправит вызов по другому каналу, который будет считаться магистральным (транковым) и снова соединит эти каналы вместе. На уровне протокола вызов set-to-set и set-to-trunk выглядят очень похожими.

Использование SIP-телефона с Asterisk означает что вы хотите настроить SIP-телефон для совершения всех своих вызовов через Asterisk, даже если устройство вполне способно напрямую подключаться к другой конечной точке SIP без сервера Asterisk. Телефон будет рассматривать Asterisk в качестве своего регистратора и прокси-сервера (хотя Asterisk на самом деле является Back to Back User Agent или B2BUA) и будет искать Asterisk для принятия решений о маршрутизации для всех вызовов.

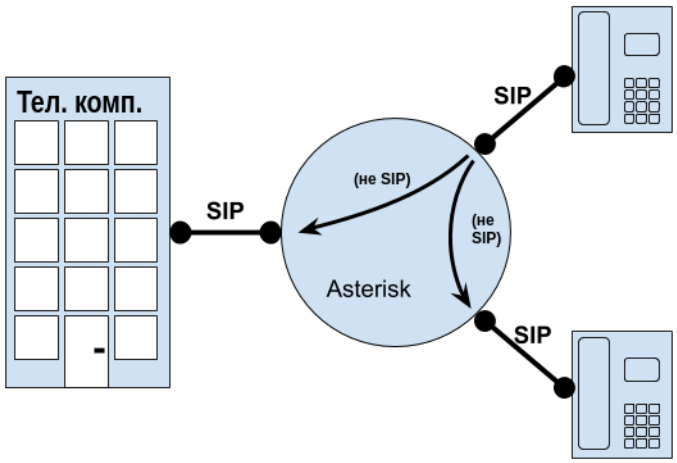

Рисунок 5-1. Asterisk в роли шлюза

Некоторые мысли о провижининге аппаратов

Хотя большинство устройств будут иметь веб-интерфейс для определения параметров, если вы используете более одного или двух телефонов в продакшене, то должны изучить использование процесса провижининга (подготовки) на основе сервера. Таким образом, аппараты будут подключаться к серверу, идентифицироваться и аутентифицироваться, а также загружать файлы конфигурации, содержащие их параметры (очень часто используется MAC-адрес телефона в качестве идентификатора для именования каждого уникального файла конфигурации). Файлы конфигурации для различных продуктов обычно обслуживаются сервером HTTPS или SFTP и будут отформатированы как XML или некоторой формой пар ключ/значение.

К сожалению, точный процесс загрузки, протокол и синтаксис этих файлов будут отличаться от производителя к производителю. Это нетрудно узнать, если вы знакомы с такими понятиями, но попытаться охватить все их (и держать постоянно меняющиеся процессы в актуальном состоянии) было бы невозможно. Производители обычно предлагают свободно загружаемые и подробные руководства по настройке для своих телефонов, поэтому с небольшим количеством исследований и знакомством с настройкой файловых сервисов на Linux вы найдете множество информации, доступной в интернете. По нашему опыту, документация, предоставленная производителями, как правило, превосходна. Она будет предоставлять самую последнюю информацию о провижининге своих устройств.

При устранении неполадок провижининга аппарата всегда сначала проверяйте загрузку с помощью компьютера. Если вы не можете загрузить файлы на свой компьютер - ваши устройства, вероятно, также не смогут их загрузить.

Мы скажем одно последнее слово об этом: убедитесь, что любой процесс, используемый вами, включает шифрование файлов конфигурации так, что если файлы украдены - только предполагаемый получатель способен их расшифровать. Большинство производителей проделали хорошую работу, чтобы сделать это довольно простой задачей. Не отправляйте незашифрованные файлы конфигурации через общедоступный Интернет.

В этой главе мы сосредоточимся на конфигурации аппаратов с точки зрения Asterisk - поэтому мы не будем слишком много говорить о самонастраиваемых телефонах; вам нужно будет провести собственное исследование в этом отношении. Мы будем использовать пару софтфонов в нашей лаборатории и Вы так же можете. В наших примерах будет предпринята попытка предоставить достаточно информации, для настройки любых SIP-устройств, используемых Вами. Если мы сможем помочь вам зарегистрировать пару софтфонов в вашей лабораторной системе, то значит направим вас по пути к более сложным сценариям (которые, как правило, требуют некоторых исследований и прототипирования с вашей стороны).

Концепции именования телефонов

Прежде чем приступить к настройке Asterisk для наших телефонов - мы предложим некоторые рекомендации по именованию телефонов.

Во-первых, вы не должны назначать своим телефонам добавочный номер; вместо этого создайте систему так, чтобы добавочный номер был назначен пользователю, а затем назначьте телефон или другие ресурсы этому пользователю. Сами телефоны должны быть названы в соответствии с чем-то уникальным для них, таким как MAC-адрес или имя компьютера. В гибкой АТС следующего поколения необходимо абстрагировать понятия пользователей, добавочных номеров и телефонов чтобы обеспечить максимальную гибкость и простоту управления.

В Asterisk действительно нет понятия пользователя вообще. Добавочные номера - это триггеры, которые инициируют последовательность инструкций. Да, вы можете написать небольшой диалплан, указав, что при наборе добавочного номера 100 Asterisk будет звонить на телефон на вашем столе. Однако добавочный 100 может также легко получить доступ к ящику голосовой почты компании или, возможно, воспроизвести приглашение, присоединиться к конференц-залу или сделать любое количество других вещей. Мы даже можем написать диалплан, который указывает, что расширение 100 должно звонить на устройство на вашем столе с понедельника по пятницу между 9 утра и 5 вечера, но в остальное время звонить на устройство на другом столе. И наоборот, когда вызов выполняется с устройства в рабочее время, идентификатор вызывающего абонента может показывать дневной номер, а в остальное время может показывать номер после рабочего дня (многие стойки регистрации становятся столами дежурных в ночное время).

В Asterisk добавочные номера (расширения) не являются телефонами. Поэтому не давайте идентификаторы телефонов, которые являются добавочными номерами.

Внутренние номера Asterisk

Концепция внутренних номеров в Asterisk имеет решающее значение. В большинстве УАТС добавочный номер - это номер, который вы набираете для вызова телефона или службы. В Asterisk внутренний номер является именем группы инструкций в диалплане. Подумайте о внутреннем номере Asterisk как об имени скрипта и вы на правильном пути. Да, внутренний номер Asterisk может быть номером, который звонит на телефон, но он может быть также словом (например, `voicemail`), которое предоставляет простую услугу какого-либо рода, никогда не отправляя вызов по какому-либо каналу.

Мы рассмотрим внутренние номера Asterisk более подробно на протяжении всей этой книги, но прежде чем мы это сделаем, мы должны настроить некоторые телефоны.

Абстракция между именем добавочного номера и телефоном, который может звонить - это мощная концепция. Отличным примером этого является функция УАТС, широко известная как *hot-desking*, которая позволяет пользователям совместно использовать один стол и/или перемещаться по разным столам. Допустим, у нас есть три агента по продажам, которые обычно работают вне офиса, но проводят пару дней каждый месяц в офисе для оформления документов. Поскольку они вряд ли будут находиться на месте одновременно - вместо того, чтобы у каждого агента был отдельный телефон, они могли бы использовать один офисный телефон (или в большем масштабе, дюжина людей могла бы разделить пул, скажем, из трех телефонов). Этот сценарий иллюстрирует удобство (и необходимость) предоставления системе возможности отделить концепцию пользователя и внутреннего номера от физического телефона.

Лучший способ назвать физический настольный телефон (или любую физическую конечную точку SIP) - использовать MAC-адрес устройства, являющийся уникальным для телефона, следующим за ним, и не требующий изменений конфигурации телефона, если меняется пользователь. У некоторых корпораций есть наклейки, которые они размещают на своем оборудовании со штрих-кодом и другой информацией, которая позволяет им хранить запас подготовленного оборудования; эти уникальные коды также будут приемлемым выбором для использования в качестве имен телефонов, поскольку они не обеспечивают никакой логической связи с конкретным человеком, но предоставляют конкретную информацию о самих устройствах.

Софтфоны (программные телефоны) на ноутбуках также могут использовать MAC-адрес или серийный номер, но убедитесь, что имя аппарата содержит ссылку на то, что это софтфон. `[JIMMY_VANM_SOFT]` - приемлимое имя, но `[JIMS_PHONE]` - нет. Если софтфон работает на настольном компьютере (т.е. он не будет перемещаться с пользователем), то назовите его, используя соглашение, которое вы используете для своих компьютеров (`[DESK-5F23-SOFT]` или `[CUST_SRVC_001_SOFT]` являются потенциально хорошими именами).

Выбор за вами относительно того, как вы хотите называть свои телефоны. Наша цель - помочь вам понять почему лучшей практикой является абстрагирование любой концепции телефона, принадлежащего человеку. Телефон - это просто способ получить и звук от человека и сигнал туда и обратно, поэтому гораздо лучше сделать возможным смешивание и сопоставление их, когда пользователи перемещаются, а люди приходят и уходят.

В этой книге, вы увидите имена телефонов, которые выглядят как MAC-адреса (например, `[0000F3000001]` и `[0000F3000002]`), или имя общего рабочего стола (`[DESK-001-SOFT]`, `[DESK-002-SOFT]`), для разделения устройств. Как правило, вы будете использовать имена телефонов, соответствующие используемому оборудованию (или какой-либо другой строке, уникальной для регистрируемого устройства).

В качестве окончательного рассмотрения мы должны четко указать что то, что мы предлагаем в отношении имен устройств, не является техническим требованием. Вы можете называть свои устройства как угодно, если они соответствуют требованиям соглашений об именах Asterisk для устройств (придерживайтесь буквенно-цифровых символов без пробелов и все будет в порядке).

Вы увидите множество систем Asterisk, связывающих имя устройства с внутренним номером пользователя, но мы не поклонники этого метода.

Телефоны, софтфоны и телефонные адаптеры

Существует три типа конечных точек, обычно предоставляемых пользователям в качестве телефонного аппарата. Они обычно называются телефонами (или настольными телефонами), софтфонами и аналоговыми терминальными адаптерами (ATA).

Аппаратный телефон - это физическое устройство - офисный телефон. У него есть трубка, нумерованные кнопки, какой-то экран и так далее. Он подключается непосредственно к сети и, вероятно, он является тем, что люди имеют в виду когда говорят о VoIP-телефоне (или SIP-телефоне). Обычно он стоит на столе, но его можно установить на стене, в лифте, на боковом столе или в коробке на обочине дороги.

Софтфон - это программное приложение, которое работает на ноутбуке, настольном компьютере, смартфоне или другом вычислительном устройстве. Звук должен проходить через звуковую систему устройства, поэтому как правило нужна гарнитура, которая будет хорошо работать с приложениями телефонии. Софтфоны стали популярными приложениями на смартфонах, потому что они позволяют подключаться к телефонным сетям, отличным от сотовой сети (например, вы можете зарегистрироваться в качестве внутреннего номера на своей корпоративной АТС). Интерфейс софтфона часто стилизован под физический телефон, но в этом нет необходимости. WebRTC позволит использовать все виды дополнительных возможностей в этой области, так как он по существу позволяет программному телефону просто быть частью сеанса в браузере. Для УАТС софтфон выглядит и ведет себя точно так же, как и аппаратный телефон.

ATA разработан, чтобы позволить традиционным аналоговым телефонам (и другим аналоговым устройствам, таким как факсы, беспроводные телефоны, пейджинговые усилители и т.д.) подключаться к сети SIP[1], и будет представлять собой коробку размером с сэндвич, которая содержит разъем RJ-11 для телефона (обычно называемый портом FXS), разъем RJ-45 для сети и разъем питания. Некоторые ATA могут поддерживать более одного телефона. Другие ATA могут иметь расширенные функции, такие как брандмауэр или порт FXO (аналоговый порт, который может подключаться к линии ТфОП). С точки зрения АТС, ATA выглядит точно так же, как SIP-телефон.

Телефоны имеют преимущество в том, что содержат хорошие акустические свойства для голосовой связи. Любой телефон приличного качества спроектирован так, чтобы улавливать частоты человеческого голоса, отфильтровывать нежелательные фоновые шумы и нормализовывать результирующую форму волны. Люди используют телефоны с тех пор, как существует телефонная сеть, и нам, как правило, нравится то, что знакомо, поэтому наличие устройства, которое взаимодействует с Asterisk с помощью знакомого интерфейса, будет привлекательным для многих пользователей. Кроме того, аппаратный телефон не требует, чтобы ваш компьютер работал все время.

Недостатком же телефонов является тот факт, что они трудно переносятся и стоят дорого по сравнению со многими качественными софтфонами на рынке сегодня, доступными бесплатно. Кроме того, дополнительный беспорядок на вашем столе может быть нежелателен, если у вас ограниченное рабочее пространство. Если вы много перемещаетесь и, как правило, не находитесь в одном и том же месте, аппаратный телефон вряд ли удовлетворит ваши потребности (хотя, по одному в каждом месте, которое вы регулярно посещаете, может быть правильным решением).

Софтфоны решают проблему переносимости, будучи установленными на устройстве, которое, вероятно, уже движется вместе с вами, например, ваш ноутбук или смартфон. Кроме того, их минимальная стоимость (как правило бесплатны, или около $30 за один полнофункциональный) является привлекательной. Поскольку многие софтфоны бесплатны, вполне вероятно, что первым телефонным аппаратом, который вы подключите к Asterisk, будет софтфон. Кроме того, поскольку софтфоны - это просто программное обеспечение, их легко устанавливать и обновлять, и как правило они имеют другие функции, используемые другими периферийными устройствами, такие как веб-

1 Или любой другой сети, если на то пошло. Более формально можно было бы назвать области аналого-цифровых шлюзов, где характер цифрового протокола может варьироваться (например, проприетарные АТС на традиционных УАТС). Дело в том, что ATA не обязательно является SIP-устройством.

камера для видеозвонков или возможность загружки файлов с вашего рабочего стола для отправки факсов. Еще одним потенциально огромным преимуществом софтфона является то, что его часто можно интегрировать с другими приложениями, работающими на устройстве.

Некоторые из недостатков софтфонов - это не всегда включенное устройство, необходимость надевать гарнитуру каждый раз при приёме вызова, и тот факт, что многие ПК будут в произвольное время в течение дня выбирать другое приложение для использования пользователем, что может привести к тому, что софтфон перестанет работать, пока какая-то фоновая задача захватила процессор. В мобильном устройстве софтфон может потреблять ресурсы, что влияет на время работы батареи, производительность и эксплуатационные расходы.

У ATA есть преимущество, позволяющее подключать аналоговые устройства к сети SIP[2], такие как беспроводные телефоны (которые все еще превосходят во многих случаях более продвинутые типы беспроводных Wi-Fi телефонов и гораздо менее дорогостоящие[3]), пейджинговые усилители, звонки и старинные телефоны[4]. ATA также иногда можно использовать для подключения к старой проводке, где сетевое соединение может работать неправильно, или к надворным постройкам (например, сторожка), где стандартное подключение ethernet невозможно.

Основным недостатком ATA является невозможность получения тех же функций через аналоговую линию, что и с SIP-телефона. Это технология, которой уже более ста лет.

С Asterisk нам не обязательно выбирать между софтфоном, аппаратным телефоном или ATA; вполне возможно и довольно часто иметь один добавочный номер, который одновременно звонит на несколько устройств, таких как настольный телефон, софтфон на ноутбуке, сотовый телефон и, возможно, стробоскоп в задней части завода (где слишком много шума для звонка).

Больше, чем любая другая конечная точка, программный телефон будет развиваться в нечто гораздо более всеобъемлющее, чем простое телефонное приложение. Появление WebRTC может наконец сделать то, что было предсказано в течении многих долгих лет: интеграция голоса в реальном времени в вычислительные (в частности, веб-приложения). Конечно, есть много способов достичь этого уже, но преимущество WebRTC заключается в том, что это открытый стандарт, встроенный прямо во все браузеры без каких-либо плагинов. Софтфон мертв; да здравствует софтфон.

Нам все еще нравится настольный телефон для регулярных телефонных звонков.

Настройка Asterisk

В этом разделе мы рассмотрим, как настроить PJSIP для обработки различных конечных точек SIP. Традиционно это делалось путем редактирования файлов в каталоге */etc/asterisk/*; однако мы решили продемонстрировать как это делается через базу данных, поскольку в целом это превосходный метод, особенно по мере роста системы. Если вам более удобны в использовании *.conf* файлы - вы должны найти, что это довольно легко сделать как только вы уяснили основы[5].

2 ATA - это не единственный способ подключения аналоговых телефонов. Поставщики оборудования, такие как Digium и Sangoma, предлагают карты, которые подключаются к серверу Asterisk и предоставляющие аналоговые телефонные порты. Более крупные установки могут также использовать банки каналов или MSANы; однако этот метод подключения устаревших телекоммуникационных схем является более продвинутым предметом, а не предметом этой книги.

3 Для действительно удивительного беспроводного аналогового телефона вы должны проверить устройства EnGenius DuraFon, которые дороги, но впечатляют.

4 Наш друг Брайан Капуч собрал вместе множество занимательных демонстраций того, как можно использовать старинное телефонное оборудование для работы с Asterisk.

5 Проще говоря, вам нужно будет найти файл *pjsip.conf.sample*, и использовать его в качестве шаблона для создания файла *pjsip.conf* в папке */etc/asterisk*, а затем отредактировать этот файл таким же образом, как мы собираемся делать в базе данных.

Asterisk позволяет устройствам, использующим множество различных протоколов, общаться с ним (и, следовательно, друг с другом), но `chan_pjsip` - единственный модуль VoIP, который все еще активно поддерживается; остальные - устаревший код. Вы вряд ли будете использовать другие протоколы VoIP (такие как IAX2, Skinny/SCCP, Unistim, H323 и MGCP). Эти протоколы имеют историческое значение, так как это было во многом связано с тем, что Asterisk будет общаться с чем угодно и обо всём, что может оказать влияние на телекоммуникационную отрасль. Однако в настоящее время SIP почти вытеснил их все, поэтому эти драйверы каналов теперь являются историческими курьезами и не более того. Если вы все еще заинтересованы в одном из этих протоколов, сосредоточьтесь на том, чтобы сначала комфортно работать с SIP и признать, что он в значительной степени самодостаточен.

Таблицы конфигурации каналов в базе данных[6] содержат сведения о конфигурации, относящиеся к этому драйверу канала, а также параметры и учетные данные, относящиеся к устройствам и провайдерам SIP, которые вы подключите к Asterisk (входящие и исходящие). Проще говоря: все звонки в Asterisk и из него должны проходить через канал.

Большинство параметров имеют значения по умолчанию, которые вы найдете в файлах примеров. Начните с чтения файла *pjsip.conf.sample* находящегося в вашем каталоге *~/src/asterisk-16<TAB>/configs/samples/*. Он предоставит множество информации о значениях по умолчанию, а также о других ресурсах, о которых стоит прочитать. Мы не будем использовать файл для фактической конфигурации (вместо этого мы используем базу данных); однако файл является отличной ссылкой и вы должны держать его под рукой, так как он будет иметь ответы на многие вопросы о параметрах, которые у Вас могут возникнуть.

Мы собираемся сосредоточиться на получении базового устройства для вас как можно проще. Мы обнаружили, что настройка каналов является одной из самых неприятных вещей, которые испытывают новые пользователи Asterisk, и хотим продемонстрировать, что на самом базовом уровне это не должно быть болезненным вообще. После того, как вы преуспеете здесь - у вас всегда будет известная хорошая конфигурация чтобы вернуться, в то время как вы продвинетесь вперед в более сложные сценарии.

Как конфигурация канала работает с диалпланом

Каналы - это то, что Asterisk соединяет вызовом со всем, что находится за его пределами, но именно диалплан определяет что происходит с вызовами по мере их прохождения через систему. Таким образом, каналы и диалплан неразрывно связаны. Диалплан - это сердце системы Asterisk: он управляет тем, как логика вызова применяется к любому соединению, к любому каналу, например что происходит, когда устройство набирает добавочный номер 101 или входящий вызов от внешнего провайдера направляется в DID. Таблицы конфигурации канала PJSIP в базе данных, а также файл */etc/asterisk/extensions.conf* будут играть важную роль в большинстве, если не во всех, вызовах, маршрутизируемых через систему. После того, как настроили свои каналы, вы обнаружите, что большая часть вашей работы будет в диалплане. Мы глубоко погрузимся в него в следующих главах.

Когда вызов поступает в Asterisk - идентификатор входящего вызова сопоставляется с конфигурацией канала используемого протокола (SIP-соединения обрабатываются драйвером канала PJSIP). Драйвер канала будет обрабатывать аутентификацию входящего соединения. Конфигурация канала также *определяет где этот канал будет входить в диалплан*.

Как только Asterisk определит как будет обрабатывать канал (т.е. он аутентифицирован и установлены различные параметры вызова), она может передать управление вызовом в правильный контекст в диалплане. Это параметр контекста в таблице `ps_endpoints`, который сообщает каналу, где вызов будет входить в диалплан (который содержит всю информацию о том, как обрабатывать и маршрутизировать вызов).

6 Или в самом файле *.conf* если вы заходите пойти по этому пути.

На Рисунке 5-2 мы видим, что поток вызовов через конфигурацию для внутреннего вызова (set-to-set) будет выглядеть примерно так:

1. Пользователь устройства `0000f30A0A0101` набирает `102`.
2. Asterisk сопоставляет входящий SIP-запрос с конечной точкой (и проверяет его подлинность).
3. Набранный номер соответствует контексту `[sets]` в диалплане.
4. Приложение `Dial()` используется для отправки вызова по каналу PJSIP на контакт, связанный с `0000f30B0B02`.
5. Определяется контактный адрес (обычно на основе регистрации, если это устройство, но также может быть жестко задан если это транк).
6. Сообщение SIP INVITE отправляется в пункт назначения.

Ключевым моментом является то, что файлы конфигурации каналов управляют не только тем, как вызовы входят в систему, но и тем, как они покидают систему. Так, например, если одно устройство вызывает другое, файл конфигурации канала используется не только для передачи вызова в диалплан, но и для направления вызова из диалплана к месту назначения.

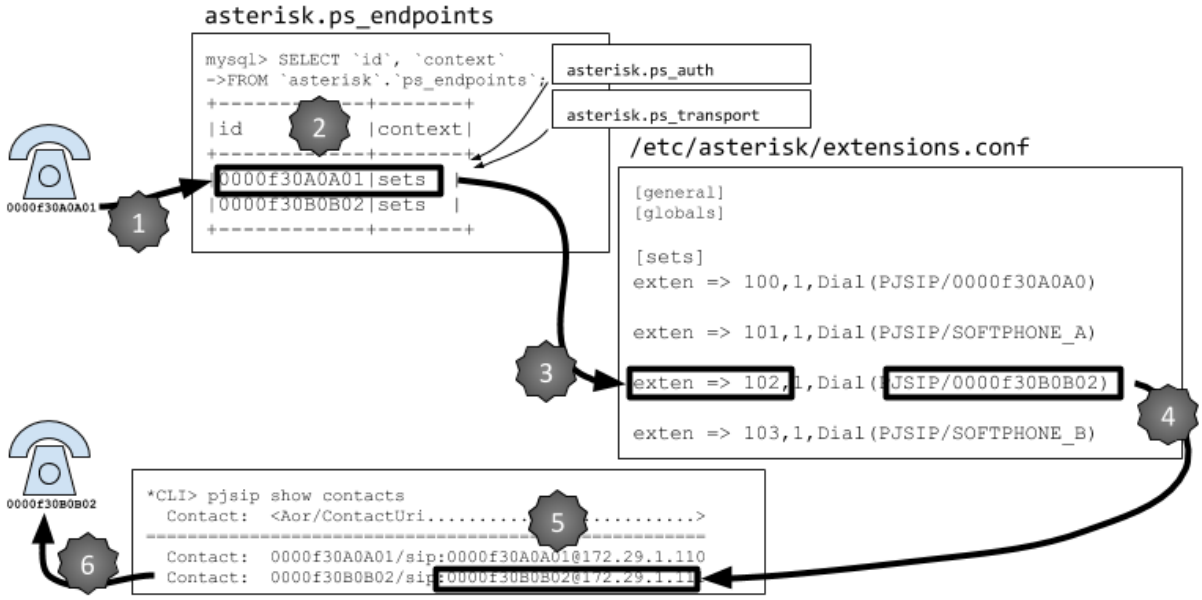

Рисунок 5-2. Отношение pjsip.conf к extensions.conf

chan_pjsip

Модуль канала PJSIP является одним из новых модулей в Asterisk. Он заменил оригинальный модуль `chan_sip`.

Старый модуль SIP - `chan_sip` был устаревшим, поэтому мы не будем касаться его в этой книге. Если вы новичок в Asterisk, то должны придерживаться PJSIP, но может быть полезно знать что `chan_sip` использовался в течение многих лет и до сих пор широко используется на старых системах.

Платформа PJSIP, реализованная в Asterisk, состоит из многих компонентов. Если вы проверите свою базу данных, то обнаружите что существует более десятка таблиц, относящихся к PJSIP (с префиксом `ps_`). Однако, не все из них относятся к заданной конфигурации.

> PJSIP - это универсальная библиотека с открытым исходным кодом для мультимедийной связи, обеспечивающая не только сигнализацию SIP, но также функции медиапотока и обхода NAT,

являющиеся важными компонентами приложений на основе SIP. Она предоставляется и поддерживается компанией Telugu Ltd. и библиотека используется в гораздо большем числе приложений, чем Asterisk. Софтфоны, проприетарные продукты и другие проекты с открытым исходным кодом также используют фреймворк. Сообщество Asterisk нуждалось/хотело новый драйвер канала SIP и вместо того, чтобы строить его с нуля, разработчики решили использовать библиотеку PJSIP.

Компоненты, перечисленные в Таблице 5-1 будут испольоваться при настройке конечных точек.

Таблица 5-1. Компоненты PJSIP в Asterisk

Компонент	Назначение
ps_aors	Address Of Record (AOR) - адрес записи - таблица использумая для определения того, как Asterisk может связаться с конечной точкой. Когда устройство пытается зарегистрироваться, Asterisk будет обращаться к AOR, чтобы идентифицировать его.
ps_auths	Раздел Authentication (проверка подлинности) содержит учетные данные, которые конечные точки должны будут предоставить системе Asterisk для авторизации.
ps_contacts	Обычно создается автоматически как часть процесса регистрации, именно здесь Asterisk будет хранить сведения о конечной точке, определенной во время регистрации.
ps_endpoints	Сердце конфигурации SIP - именно здесь определяется каждая конечная точка. Здесь также определяются связи с другими записями PJSIP.

Добавление конечной точки

Во время установки для вас было создано несколько примеров конечных точек для упрощения процесса представления рабочей системы. Если вы хотите добавить дополнительные конечные точки, то нужно просто определить дополнительные записи в каждой из таблиц `ps_aors`, `ps_auths` и `ps_endpoints`.

Допустим, мы хотим добавить пару софтфонов с именами `SOFTPHONE_A` и `SOFTPHONE_B` в нашу систему.

Во-первых, в таблицу `ps_endpoints` мы должны добавить следующее:

```
$ mysql -D asterisk -u asterisk -p
```

Давайте рассмотрим, что у нас там уже есть (из предыдущих глав):

```
mysql> select id,transport,aors,auth,context,disallow,allow from asterisk.ps_endpoints;
+--------------+---------------+--------------+--------------+----------+----------+-------+
| id           | transport     | aors         | auth         | context  | disallow | allow |
+--------------+---------------+--------------+--------------+----------+----------+-------+
| 0000f30A0A01 | transport-udp | 0000f30A0A01 | 0000f30A0A01 | starfish | all      | ulaw  |
| 0000f30B0B02 | transport-udp | 0000f30B0B02 | 0000f30B0B02 | starfish | all      | ulaw  |
+--------------+---------------+--------------+--------------+----------+----------+-------+
2 rows in set (0.00 sec)
```

Мы собираемся вставить пару дополнительных записей.

```
mysql> insert into asterisk.ps_endpoints
 (id,transport,aors,auth,context,disallow,allow)
values
 ('SOFTPHONE_A','transport-tls','SOFTPHONE_A','SOFTPHONE_A','sets','all','ulaw'),
 ('SOFTPHONE_B','transport-tls','SOFTPHONE_B','SOFTPHONE_B','sets','all','ulaw');
Query OK, 2 rows affected (0.02 sec)
```

Теперь таблица `ps_endpoints` должна выглядеть примерно так:

```
mysql> select id,transport,aors,auth,context,disallow,allow from ps_endpoints;
+--------------+---------------+--------------+--------------+---------+----------+-------+
| id           | transport     | aors         | auth         | context | disallow | allow |
+--------------+---------------+--------------+--------------+---------+----------+-------+
| 0000f30A0A01 | transport-udp | 0000f30A0A01 | 0000f30A0A01 | sets    | all      | ulaw  |
```

```
| 0000f30B0B02 | transport-udp | 0000f30B0B02 | 0000f30B0B02 | sets    | all      | ulaw  |
| SOFTPHONE_A  | transport-tls | SOFTPHONE_A | SOFTPHONE_A   | sets    | all      | ulaw  |
| SOFTPHONE_B  | transport-tls | SOFTPHONE_B | SOFTPHONE_B   | sets    | all      | ulaw  |
+--------------+---------------+-------------+---------------+---------+----------+-------+
4 rows in set (0.00 sec)
```

Затем нам понадобятся две связанные записи в таблице `ps_aors`:

```
mysql> insert into asterisk.ps_aors (id,max_contacts)
values ('SOFTPHONE_A',2),('SOFTPHONE_B',2);
Query OK, 2 rows affected (0.01 sec)
```

Затем таблица `ps_aors` должна возвращать следующее:

```
mysql> select id from asterisk.ps_aors;
+--------------+
| id           |
+--------------+
| 0000f30A0A01 |
| 0000f30B0B02 |
| SOFTPHONE_A  |
| SOFTPHONE_B  |
+--------------+
4 rows in set (0.00 sec)
```

Наконец, таблица `ps_auths` будет нуждаться в записях для каждого нового устройства..

```
insert into asterisk.ps_auths (id,auth_type,password,username)
values ('SOFTPHONE_A','userpass','iwouldnotifiwereyou','SOFTPHONE_A'),
('SOFTPHONE_B','userpass','areyoueventrying','SOFTPHONE_B');

Query OK, 2 rows affected (0.00 sec)
```

И, если все прошло хорошо, у вас будут следующие записи авторизации:[7]

```
mysql> select id,auth_type,password,username
 -> from asterisk.ps_auths;
+--------------+-----------+---------------------+--------------+
| id           | auth_type | password            | username     |
+--------------+-----------+---------------------+--------------+
| 0000f30A0A01 | userpass  | not very secure     | 0000f30A0A01 |
| 0000f30B0B02 | userpass  | hardly to be trusted | 0000f30B0B02 |
| SOFTPHONE_A  | userpass  | iwouldnotifiwereyou  | SOFTPHONE_A  |
| SOFTPHONE_B  | userpass  | areyoueventrying     | SOFTPHONE_B  |
+--------------+-----------+---------------------+--------------+
4 rows in set (0.00 sec)
```

Новые конечные точки теперь готовы к регистрации устройств на них. Вы можете убедиться, что они существуют с помощью следующей команды Asterisk CLI:

```
mysql> exit
$ sudo asterisk -rvvvvv
*CLI> pjsip show endpoints
```

Выходные данные должны содержать список новых конечных точек:

```
Endpoint: 0000f30A0A01                  Unavailable   0 of inf
    InAuth: 0000f30A0A01/0000f30A0A01
       Aor: 0000f30A0A01                2
 Transport: transport-udp     tls    0    0    0.0.0.0:5061

Endpoint: 0000f30B0B02                  Not in use    0 of inf
    InAuth: 0000f30B0B02/0000f30B0B02
       Aor: 0000f30B0B02                2
     Contact: 0000f30B0B02/sip:0000f30B0B02@172.29.1.110 7542ca7ce1 Unknown    nan
 Transport: transport-udp     udp    0    0    0.0.0.0:5060

Endpoint: SOFTPHONE_A                   Unavailable    0 of inf
    InAuth: SOFTPHONE_A/SOFTPHONE_A
       Aor: SOFTPHONE_A                 2

Endpoint: SOFTPHONE_B                   Unavailable 0 of inf
    InAuth: SOFTPHONE_B/SOFTPHONE_B
```

7 За исключением, конечно того, что ваши пароли будут намного серьезнее, чем те, которые мы использовали здесь.

```
        Aor: SOFTPHONE_B                2
```

Подождите минутку…

Вы заметили, что для новых конечных точек не определен транспорт? Это потому, что мы еще ничего не определили для TLS; мы только что определили для стандартного UDP-стиля SIP.

Поскольку у нас есть те причудливые ключи, которые мы создали в предыдущей главе, давайте реализуем их сейчас и посмотрим, сможем ли мы это исправить.

```
$ sudo vim /etc/asterisk/pjsip.conf

[transport-udp]
type=transport
protocol=udp
bind=0.0.0.0

[transport-tls]
type=transport
protocol=tls
bind=0.0.0.0
cert_file=/home/asterisk/certs/self-signed.crt     #если вы использовали certbot, то
priv_key_file=/home/asterisk/certs/self-signed.key #расположение будет таким
```

Теперь, поскольку мы помещаем некоторые файлы в папку, которая не была частью нашей конфигурации SELinux, теперь мы должны это исправить. Мы хотим, чтобы SELinux генерировал ошибку, поэтому собираемся перезагрузить модуль `res_pjsip.so`, даже если он не сможет правильно загрузить транспорт:

```
*CLI> module reload res_pjsip.so
```

Теперь у нас должны быть ошибки, которые мы ожидали, поэтому будем искать их в системном журнале.

```
$ sudo grep -i sealert /var/log/messages |egrep "cert|crt"
```

Вы увидите некоторые сообщения, которые выглядят примерно так (мы урезали их для краткости):

```
SELinux is preventing ... on the file /home/asterisk/certs/self-signed.crt.
 For complete SELinux messages run: sealert -l 1dbe51e2-7321-41d3-a5bb-f8f1b4a6f787

SELinux is preventing ... on the directory certs.
 For complete SELinux messages run: sealert -l 879db542-e0a9-43e8-8763-62fcf068bfee

SELinux is preventing ... on the file self-signed.crt.
 For complete SELinux messages run: sealert -l 8fb85940-ee82-44bd-adcb-e30d31ee516a
```

Полезно, что SELinux говорит вам именно то, что нужно сделать для решения этой проблемы!

Для каждого из трех сообщений, относящихся к доступу к сертификатам, которые мы только что настроили, выполните соответствующую команду. А мы просто выполним одну, чтобы показать, что имеем в виду, но вам может потребоваться запустить более одной, пока загрузка не пройдет без ошибок.

```
$ sealert -l 8fb85940-ee82-44bd-adcb-e30d31ee516a
```

Вы должны получить что-то вроде этого:

```
SELinux is preventing asterisk from read access on the file self-signed.crt.
***** Plugin catchall (100. confidence) suggests **************************
You can generate a local policy module to allow this access.
allow this access for now by executing:
# ausearch -c 'asterisk' --raw | audit2allow -M my-asterisk
# semodule -i my-asterisk.pp
```

Вы не `root`, но вы должны запустить обе команды, которые он определяет:

```
$ sudo ausearch -c 'asterisk' --raw | audit2allow -M my-asterisk

$ sudo semodule -i my-asterisk.pp
```

Помните, как все говорят что нужно просто отключить SELinux? Ну, больше не нужно этого делать.

Хорошо, перезапустите Asterisk (`$sudo service asterisk restart`) и убедитесь, что ваш файл журнала не генерирует ошибки SELinux (игнорируйте ошибки *ODBC.ini*, поскольку они не относятся к */etc/odbc.ini* и не должны ни на что влиять).

Вы должны увидеть, что `transport-tls` теперь готов к использованию:

```
*CLI> pjsip show transports
Transport: <TransportId........> <Type> <cos> <tos> <BindAddress..................>
===============================================================================
Transport: transport-tls tls 0 0 0.0.0.0:5061
Transport: transport-udp udp 0 0 0.0.0.0:5060
```

Если вы обнаружите, что он все еще не загружается, вернитесь и отработайте ошибки SELinux в файле */var/log/messages*. Иногда приходится иметь дело не только с одной.

Тестирование чтобы убедиться что ваши устройства зарегистрированы

После того, как ваши устройства зарегистрировались в Asterisk - вы сможете запросить их местоположение и состояние из Asterisk CLI.

Это распространенное заблуждение, что регистрация - это то, как устройство аутентифицирует себя с целью получения разрешения на выполнение вызовов. Это неверно. Единственная цель регистрации - позволить устройству определить свое местоположение в сети, чтобы Asterisk[8] знала, куда отправлять вызовы, предназначенные для этого устройства.

Аутентификация для исходящих вызовов является полностью отдельным процессом и всегда происходит на основе каждого вызова, независимо от того, зарегистрировано ли устройство. Это означает, что ваше устройство может совершать звонки, но не принимать их. Наиболее распространенной причиной этого является брандмауэр, закрывший входящий порт (и решение состоит в том, чтобы установить таймер регистрации на более короткое время, чтобы оно перерегистрировалось каждые несколько минут, так что брандмауэр будет держать соответствующий SIP-порт открытым).

Можно иметь успешно установленную регистрацию и все же совершать звонки будет запрещено. Дело в том, что только потому, что устройство зарегистрировано не означает, что оно может совершать звонки (хотя это почти всегда будет так).

Проверка регистрации устройства - это самый простой способ проверить правильность его настройки.

Помните, что настройка устройства *не* происходит в Asterisk. Вы должны настроить устройство с помощью любых инструментов, предоставленных производителем.

Для проверки состояния устройства, просто вызовите CLI Asterisk:

```
$ sudo asterisk -rvvvv
```

При вводе следующей команды возвращается список всех пиров, о которых знает Asterisk (зарегистрированные устройства будут иметь соответствующий `Contact`):

```
*CLI> pjsip show aors
 Aor: <Aor.............................................> <MaxContact>
 Contact: <Aor/ContactUri.........................> <Hash....> <Status> <RTT(ms)..>
===============================================================================
```

8 Или любой другой сервер регистратора SIP, если на то пошло.

```
Aor: 0000f30A0A01                                          2
Aor: 0000f30B0B02                                          2
Contact: 0000f30B0B02/sip:0000f30B0B02@172.29.1.110:5 7542ca7ce1 Unknown     nan

Aor: SOFTPHONE_A                                           2

Aor: SOFTPHONE_B                                           2
```

Базовый диалплан для тестирования ваших устройств

В следующей главе мы погрузимся в диалплан Asterisk. Здесь мы собираемся заложить очень простой диалплан, так что если вы регистрируете устройства на различные конечные точки SIP уже в конфигурации PJSIP - вы должны быть в состоянии сделать тестовые вызовы между ними.

```
$ sudo -u asterisk vim /etc/asterisk/extensions.conf

[general]
[globals]

[sets]
exten => 100,1,Dial(PJSIP/0000f30A0A01)

exten => 101,1,Dial(PJSIP/0000f30B0B02)

exten => 102,1,Dial(PJSIP/SOFTPHONE_A)

exten => 103,1,Dial(PJSIP/SOFTPHONE_B)

exten => 200,1,Answer()
     same => n,Playback(hello-world)
     same => n,Hangup()
```

В консоли Asterisk введите следующую команду:

```
*CLI> dialplan reload

*CLI> dialplan show
```

Вы увидите, что в контексте `sets` есть некоторые добавочные номера, которые вы можете вызвать.

Этот базовый диалплан позволит вам набирать ваши устройства SIP с помощью добавочных номеров 100, 101, 102 и 103. Вы также можете прослушать приглашение `hello-world`, созданное для этой книги, набрав добавочный номер 200.

Зарегистрируйте пару SIP-телефонов (вы можете скачать один софтфон на свой компьютер, а другой - на планшет или смартфон). Вы должны быть в состоянии набирать между вашими номерами. Откройте CLI для того, чтобы увидеть прохождение звонка. Вы должны увидеть что-то вроде этого (и устройство, которое вы вызываете, должно звонить):

```
-- Executing [102@sets:1] Dial("PJSIP/SOFTPHONE_A-00000001", "PJSIP/0000f30B0B02")
 -- Called PJSIP/0000f30B0B02
 -- PJSIP/0000f30B0B02-00000002 is ringing
```

Если этого не происходит - проверьте конфигурацию и регистрацию и убедитесь, что не сделали никаких опечаток.

Регистрация устройств в Asterisk

Существует так много различных видов SIP-устройств, которые вы можете зарегистрировать в Asterisk, невозможно привести пример, который будет полезен всем. У вас может быть ПК или Mac, рабочая станция Linux или iPhone, или Android, или настольный SIP-телефон, или какое-то иное устройство SIP; у каждого типа устройств есть много разных типов доступных клиентов SIP, и все они в основном одинаковы, но достаточно разные, чтобы раздражать новичка.

Мы не можем показать вам конкретный метод регистрации, но мы обнаружили, что из десятков или, возможно, сотен вариантов в каждом устройстве основной процесс аналогичен для всех из

них. Вам нужно будет предоставить:

- Адрес вашего сервера Asterisk (`hostname, domen, proxy и server` - это все поля, которые мы видели для него)
- Идентификатор устройства (`id, user, subscriber, username, extension, name` и тд)
- Пароль

Если это становится сложным, вы, вероятно, зашли слишком далеко. Пусть все будет просто. Если продукт не имеет хорошей документации, то это может быть неправильный продукт для вас. Для большинства SIP-телефонов вы можете найти инструкции в интернете о том, как зарегистрировать их на Asterisk.

Если вы зарегистрируете второе устройство (теперь у вас их четыре!), вы можете сделать тестовые вызовы между ними.

Потратьте немного времени на это и убедитесь, что вы все это поняли. Это имеет решающее значение для всего, что последует далее.

Под капотом: Ваш первый звонок

Чтобы вы задумались о том, что происходит под капотом, мы кратко рассмотрим некоторые вещи из того, что на самом деле происходит с протоколом SIP, когда два устройства в одной системе Asterisk вызывают друг друга.

Asterisk как B2BUA

Имейте в виду, что на самом деле здесь происходит два вызова: один от исходного устройства в Asterisk, а другой из Asterisk в целевое устройство. SIP - это одноранговый протокол, и с точки зрения протокола происходит два вызова. Протокол SIP не знает, что Asterisk соединяет вызовы; каждое устройство понимает свое соединение как соединение с Asterisk, без реального знания устройства на другой стороне. Именно по этой причине Asterisk часто называют B2BUA (Back to Back User Agent). Именно поэтому так легко соединять различные протоколы вместе с помощью Asterisk.

Для только что сделанного вызова будут созданы диалоги, показанные на Рисунке 5-3.

Для получения более подробной информации о том, как работает обмен SIP-сообщениями, обратитесь к SIP RFC.

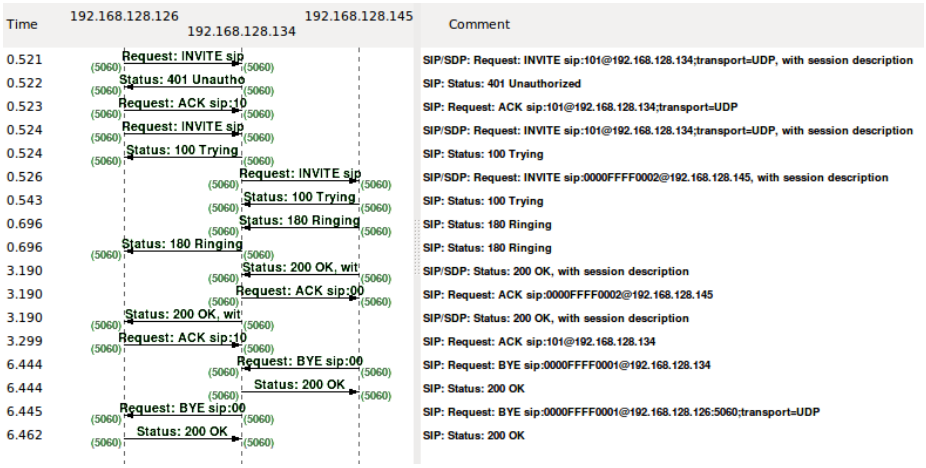

Рисунок 5-3. SIP диалоги

Вывод

В этой главе вы изучили рекомендации по именованию устройств путем абстрагирования понятий пользователь, добавочный номер и устройство, а также определения параметров конфигурации устройства и аутентификации в файлах конфигурации канала. Затем мы углубимся в магию Asterisk, которой является диалплан и посмотрим, как простые вещи могут давать отличные результаты.

Глава 6. Основы диалплана

Всё должно быть изложено так просто, как только возможно, но не более того.
— Альберт Эйнштейн

Диалплан - это сердце вашей системы Asterisk. Он определяет, как звонки поступают в систему и выходят из нее. Диалплан является скриптовым языком, который определяет инструкции, которым Asterisk следует в ответ на вызовы, поступающие от каналов. В отличие от традиционных телефонных систем, диалплан Asterisk полностью настраивается.

Опытные разработчики программного обеспечения считают код диалплана Asterisk архаичным и часто предпочитают управлять потоком вызовов с помощью API Asterisk, таких как AMI и ARI (которые мы обсудим в последующих главах). Независимо от ваших планов в этом отношении - изучение поведения Asterisk намного проще, если вы сначала поймете диалплан. Возможно, также стоит отметить, что диалплан Asterisk настроен на производительность и поэтому является самым быстрым способом выполнения потока вызовов с точки зрения быстродействия и минимальной нагрузки на систему. Диалплан работает быстро.

В этой главе представлены основные понятия, которые лягут в базу любого диалплана, написанного вами. Не пропускайте слишком много из этой главы, так как примеры строятся друг на друге и это принципиально важно для Asterisk. Обратите также внимание, что эта глава ни в коем случае не является исчерпывающим обзором всех возможных вещей, которые может сделать диалплан; наша цель - охватить только самое необходимое. В последующих главах мы рассмотрим более сложные темы диалплана. Вам рекомендуется экспериментировать.

Синтаксис диалплана

Диалплан Asterisk задается в конфигурационном файле с именем *extensions.conf*, расположенном в каталоге */etc/asterisk*.

Структура диалплана состоит из четырех иерархических компонентов: контекстов (Context), расширений (Extension), приоритетов (Priority) и приложений (Application) (смотри Рисунок 6-1).

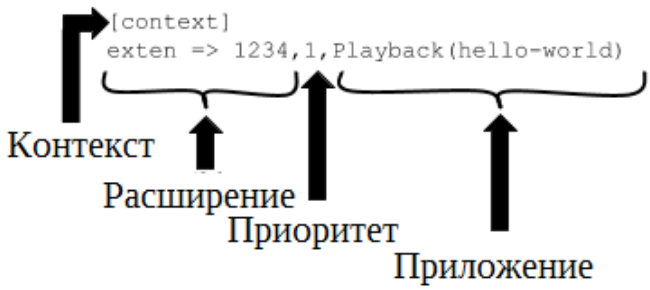

Рисунок 6-1. Иерархия диалплана

Давайте нырнем прямо туда.

> **Примеры файлов конфигурации**
>
> Основной файл *extensions.conf* был создан как часть процесса установки ранее в этой книге. Мы будем опираться на этот файл в этой главе.

Asterisk также поставляется с подробным файлом *extensions.conf*, который может быть установлен с образцами файлов конфигурации (за это отвечает команда установки `make samples`), и если вы запустили эту команду (мы не рекомендуем запускать её во время установки, но это предлагается установщиком), у вас, скорее всего, будет файл */etc/asterisk/extensions.conf*, который переполнен информацией. Вместо того, чтобы начинать исправлять под себя файл примера, мы предлагаем вам построить свой *extensions.conf* с нуля (исходный файл вы можете переименовать или переместить куда-нибудь, если хотите сохранить в качестве источника примеров).

Как и говорилось, файл примера *extensions.conf* - это фантастический ресурс, полный примеров и идей, которые вы можете использовать после того, как изучили основные понятия. Если вы выполнили наши инструкции по установке, то найдете файл *extensions.conf.sample* в каталоге *~/src/asterisk-16./configs/samples* (наряду со многими другими образцами файлов конфигурации).

Контексты

Диалплан делится на разделы, называемые *контекстами*, служащие для разделения различных частей диалплана. Расширение, определенное в одном контексте, полностью изолировано от расширений в любом другом контексте, если взаимодействие специально не разрешено.

В качестве простого примера представим, что у нас есть две компании, совместно использующие сервер Asterisk. Если мы поместим каждого автосекретаря компании (IVR) в свой собственный контекст - две компании будут полностью отделены друг от друга. Это позволяет нам самостоятельно определить, что происходит, когда, скажем, набирается номер 0:

- Абоненты, набирающие 0 из голосового меню компании A, должны быть переданы администратору компании A.
- Абоненты, набравшие 0 в голосовом меню компании B, будут отправлены в отдел обслуживания клиентов компании B.

Оба абонента находятся в одной и той же системе, взаимодействуя с одним и тем же диалпланом, но поскольку они прибыли в разные контексты, то испытывают совершенно разные потоки вызовов. То, что происходит с каждым входящим вызовом, определяется кодом диалплана в каждом контексте.

Это очень важное соображение. В традиционных УАТС, для таких вещей как прием вызова, как правило, существует набор значений по умолчанию, что означает, в случае, если вы забудете их определить, они, вероятно, будут работать в любом случае. В Asterisk все наоборот. Если вы не скажете Asterisk, как обрабатывать каждую ситуацию, и он столкнется с чем-то, что не может обработать - вызов, как правило, будет отклонен.

Контексты определяются в файле *extensions.conf*. Имя контекста помещается в квадратные скобки ([ ]). Имя может состоять из букв A - Z (верхний и нижний регистр), чисел от 0 до 9, а также дефиса и подчеркивания.[1] Контекст для входящих вызовов от оператора связи может быть назван так:

```
[incoming]
```

Имена контекстов имеют максимальную длину 79 символов (80 символов минус 1 завершающий null).

Или возможно:

```
[incoming_company_A]
```

1 Обратите внимание, что пробел явно отсутствует в списке разрешенных символов. Не используйте пробелы в именах контекста — результат вам не понравится!

Что тогда, конечно, может потребовать что-то вроде:

```
[incoming_company_B]
```

Все инструкции, помещенные после определения контекста, являются частью этого контекста, пока не будет определен следующий контекст.

В начале диалплана есть два специальных раздела с именами `[general]` и `[globals]`. Раздел `[general]` содержит список общих настроек диалплана (о которых вам, вероятно, никогда не придется беспокоиться), а контекст `[globals]` мы вскоре обсудим. На данный момент важно только знать, что эти две метки не являются контекстами, несмотря на использование синтаксиса контекста. Не используйте `[general]`, `[globals]` и `[default]`[2] в качестве имен контекста, но в противном случае называйте свои контексты как угодно.

Контексты в типичном файле *extensions.conf* могут быть структурированы примерно так:

```
[general] ; он всегда должен быть здесь
[globals] ; глобальные переменные (мы обсудим их позже)

[incoming] ; звонки от поставщиков услуг могут поступать сюда

[sets] ; в простой системе мы можем использовать это

[sets1] ; многопользовательская же нуждается в этом (устройства от одной компании входят в
        ; диалплан здесь)

[sets2] ; ... и здесь (другая группа устройств может входить в диалплан здесь)
[services] ; специальные услуги, такие как конференц-связь, могут быть определены здесь
```

Когда вы определяете канал (что не делается в *extensions.conf*), одним из обязательных параметров в каждом определении канала является `context`. *Контекст - это точка в диалплане, куда будут поступать соединения из этого канала.* Таким образом, способ подключения канала к диалплану осуществляется через контекст (Рисунок 6-2).

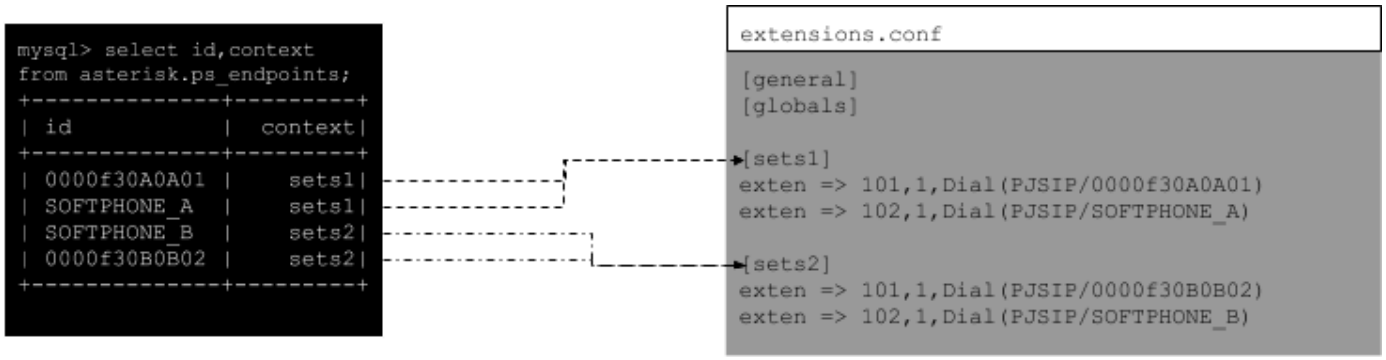

Рисунок. 6-2. Связь между конфигурацией канала (слева) и контекстами в диалплане (справа)

Это одна из наиболее важных концепций для понимания при работе с каналами и диалпланом. Устранение неполадок потока вызовов намного проще, если вы понимаете связь между контекстом канала (который сообщает каналу, где подключаться к диалплану) и контекстом диалплана (где мы создаем поток вызовов, который происходит при поступлении вызова).

Важным (возможно, самым важным) использованием контекстов является обеспечение конфиденциальности и безопасности. При правильном использовании контекстов можно предоставить некоторым каналам доступ к функциям (например, междугородним вызовам), которые недоступны другим. Если вы не проработаете свой диалплан тщательно, то можете непреднамеренно позволить другим использовать вашу систему в корыстных целях. Пожалуйста, имейте это в виду, когда строите свою систему Asterisk; в интернете есть много ботов, специально написанных для идентификации и использования плохо защищенных систем Asterisk.

2 Контекст по умолчанию был популярным способом создания простых конфигураций, но это оказалось несколько проблематичным для безопасности. Лучшая практика в эти дни - это избежать любого его использования.

Asterisk wiki описывает несколько шагов, которые вы должны предпринять, чтобы сохранить вашу систему Asterisk в безопасности. Жизненно важно чтобы вы прочитали и поняли эту страницу. Если вы игнорируете меры безопасности, изложенные там, то можете в конечном итоге позволить всем и каждому совершать междугородние или платные звонки за ваш счет!

Если вы не относитесь к безопасности вашей системы Asterisk серьезно, то можете в конечном итоге поплатиться буквально. *Пожалуйста*, потратьте время и усилия, чтобы защитить вашу систему от мошенничества.

Extensions (расширения)

В телекоммуникационной отрасли слово *extension (расширение)* обычно относится к числовому идентификатору, который при наборе будет звонить на телефон (или вызывать системный ресурс, такой как голосовая почта или очередь). В Asterisk расширение представляет нечто гораздо более мощное, поскольку оно определяет уникальную серию шагов (каждый шаг, содержащий приложение), через которые Asterisk будет принимать этот вызов.

В каждом контексте мы можем определить столько (или несколько) расширений, сколько потребуется. Когда определенное расширение запускается (входящим каналом) - Asterisk будет следовать шагам, определенным для него. Поэтому именно расширения определяют что происходит с вызовами, когда они проходят через диалплан. Хотя расширения могут использоваться для указания телефонных добавочных номеров в традиционном смысле (т.е. расширение 153 вызовет звонок SIP-телефона на столе Джона), в диалплане Asterisk они могут использоваться для гораздо большего.

Синтаксис расширения - это слово `exten`, за которым следует стрелка, образованная знаком равенства и знаком больше, как это:

```
exten =>
```

После этого следует имя (или номер) расширения.

При работе с традиционными телефонными системами мы склонны думать о расширениях как о номерах, которые вы набираете, чтобы сделать еще один телефонный звонок. В Asterisk имена расширений могут быть любыми комбинациями цифр и букв. В этой и следующей главах мы будем использовать как цифровые, так и буквенно-цифровые расширения.

Назначение имен для расширений может показаться необычной концепцией, но когда вы понимаете, что SIP поддерживает набор всех видов комбинаций символов (все, что является допустимым URI, строго говоря) - это имеет смысл. Это одна из особенностей, которая делает Asterisk настолько гибким и мощным.

Каждый шаг расширения состоит из трех компонентов:

- Имя (или номер) расширения
- Приоритет (каждое расширение может включать в себя несколько шагов; номер шага называется "приоритет”)
- Приложение (или команда), которое будет выполняться на этом шаге

Эти три компонента разделены запятыми, как здесь:

```
exten => name,priority,application()
```

Вот простой пример:

```
exten => 123,1,Answer()
```

Имя расширения - `123`, приоритет - `1`, а приложение - `Answer()`.

Приоритеты

Каждое расширение может иметь несколько шагов, называемых *приоритетами*. Приоритеты нумеруются последовательно, начиная с 1 и каждый выполняет одно конкретное приложение. Например, следующий добавочный номер ответит на звонок с приоритетом номер 1, а затем повесит трубку с приоритетом номер 2. Шаги в расширении происходят один за другим.

```
exten => 123,1,Answer()
exten => 123,2,Hangup()
```

Совершенно очевидно, что этот код на самом деле не делает ничего полезного. Ключевым моментом здесь является то, что для конкретного расширения Asterisk следует за приоритетами по порядку.

```
exten => 123,1,Answer()
exten => 123,2,делаем что-то
exten => 123,3,делаем что-то ещё
exten => 123,4,сделаем ещё одну вещь
exten => 123,5,Hangup()
```

Этот стиль синтаксиса диалплана все еще встречается время от времени, но он устарел. Новый синтаксис похож, но упрощен.

Ненумерованные приоритеты

В старых версиях Asterisk нумерация приоритетов вызывала много проблем. Представьте себе расширение, которое имеет 15 приоритетов, а затем нужно что-то добавить на Шаге 2: все последующие приоритеты должны быть перенумерованы вручную. Asterisk не обрабатывает пропущенные шаги или неправильно пронумерованные приоритеты и отладка этих типов ошибок была сложной.

Начиная с версии 1.2 Asterisk решила эту проблему: она ввела использование приоритета `n`, который означает “next." Каждый раз, когда Asterisk встречает приоритет с именем `n` - она принимает номер предыдущего приоритета и добавляет 1. Это упрощает внесение изменений в ваш диалплан, так как вам не нужно постоянно перенумеровывать все ваши шаги. Например, ваш диалплан может выглядеть примерно так:

```
exten => 123,1,Answer()
exten => 123,n,делаем что-то
exten => 123,n,делаем что-то ещё
exten => 123,n,сделаем ещё одну вещь
exten => 123,n,Hangup()
```

Внутри Asterisk будет вычислять следующий номер приоритета каждый раз, когда он сталкивается с `n`.[3] Теперь, если мы хотим добавить новый элемент в приоритет 3 - мы просто вводим новую строку, где она нам нужна, и не требуется перенумерация.

```
exten => 123,1,Answer()
exten => 123,n,делаем что-то
exten => 123,n,КАКАЯ-ТО НОВАЯ ВЕЩЬ
exten => 123,n,делаем что-то ещё
exten => 123,n,сделаем ещё одну вещь
exten => 123,n,Hangup()
```

Имейте в виду, что вы всегда должны указывать приоритет номер 1. Если вы случайно поставили `n` вместо `1` для первого приоритета (распространенная ошибка даже среди опытных кодеров диалплана), после перезагрузки диалплана вы обнаружите, что расширения не существует.

Оператор same =>

Для дальнейшего упрощения написания диалплана был создан новый синтаксис. Пока расширение остается неизменным, вы можете просто ввести `same =>` с последующим приоритетом и приложением, а не вводить полное расширение в каждой строке:

3 Asterisk допускает простую арифметику в пределах приоритета, такого как `n+200` и приоритет `s` (для same), но их использование несколько устарело из-за существования меток приоритета. Обратите внимание, что расширения и приоритет `s` - это два разных понятия.

```
exten => 123,1,Answer()
 same =>     n,делаем что-то
 same =>     n,делаем что-то
 same =>     n,сделаем ещё одну вещь
 same =>     n,Hangup()
```

Этот стиль диалплана также облегчит копирование кода из одного расширения в другое. Это предпочтительный и рекомендуемый стиль. Единственная причина обсуждения предыдущих стилей - помочь понять, как мы сюда попали.

> Не ошибитесь: диалплан Asterisk весьма своеобразен. Многие люди избегают его вообще, и используют AGI и ARI для написания своего диалплана.
>
> Хотя, конечно, есть что сказать для написания диалплана на внешнем языке (и мы рассмотрим это в последующих главах), диалплан Asterisk является родным для него, и вы не получите лучшей производительности чем с ним. Код диалплана выполняется быстро.
>
> Кроме того, если вы хотите понять, как Asterisk думает - вам нужно понять его диалплан.

Метки приоритетов

Метки приоритетов позволяют назначить имя приоритету в пределах расширения. Это дает возможность ссылаться на приоритет иначе чем на его номер (который, вероятно, неизвестен, учитывая, что диалпланы теперь, как правило, используют ненумерованные приоритеты). Позже вы узнаете, что часто необходимо отправлять вызовы из других частей диалплана на определенный приоритет в определенном расширении. Чтобы назначить текстовую метку приоритету, просто добавьте метку в скобках после приоритета, например:

```
exten => 123,n(label),application()
```

Позже мы рассмотрим, как переключаться между различными приоритетами на основе логики диалплана. Вы увидите гораздо больше меток приоритетов и будете чаще использовать их в своих диалпланах.

Очень распространенной ошибкой при написании меток является вставка запятой между номером приоритета и открывающейся скобкой, например:

```
exten => 555,n,(label),application() ;<-- ЭТО НЕ БУДЕТ РАБОТАТЬ
exten => 556,n(label),application() ;<-- Это, что надо
```

Эта ошибка нарушит часть вашего диалплана и вы получите соответствующее сообщение, указывающее, что приложение не может быть найдено.

Приложения

Приложения — это рабочие лошадки диалплана. Каждое приложение выполняет определённое действие в текущем канале, такое как — воспроизведение звука, приём набора сигналов DTMF, поиск чего-то в базе данных, выполнение вызова в канал, завершение вызова, кормление кошки или что-то иное.[4] В предыдущем примере мы показали два простых приложения: `Answer()` и `Hangup()`. Очевидно что они делают, но также очевидно что сами по себе они не очень полезны.

Некоторые приложения, включая `Answer()` и `Hangup()` не требуют дополнительных инструкций для выполнения своей задачи. Но большинству приложений требуется дополнительная информация. Эти дополнительные элементы или *аргументы* передаются в приложения чтобы повлиять на выполнение действий. Чтобы передать аргументы приложению, поместите их в круглых скобках, следующих за именем приложения, разделяя запятыми.

4 Хорошо, кормление кошки не является обычным использованием для телефонной системы, но через Asterisk такие вещи не невозможны. Доку Брауну бы это понравилось.

Приложения Answer(), Playback() и Hangup()

Приложение `Answer()` используется для ответа на канал, который звонит. Это кажется простой вещью, но много вещей происходит на канале с этой одной командой. `Answer()` сообщает каналу отправить обратно на дальний конец сообщение, что вызов был отвечен, а также включить медиа-пути (сетевые потоки, которые будут нести звук между вызывающим абонентом и системой). Как мы уже упоминали ранее, `Answer()` не принимает аргументов. `Answer()` не всегда требуется (на самом деле, в некоторых случаях он может быть вообще нежелательным), но это эффективный способ обеспечить подключение канала перед выполнением дальнейших действий.

Приложение Progress()

Иногда полезно иметь возможность передавать информацию обратно в сеть перед ответом на вызов. Приложение `Progress()` пытается предоставить информацию о ходе выполнения вызова исходному каналу. Некоторые операторы связи ожидают этого, и таким образом, вы можете решить странные проблемы с сигнализацией, вставив `Progress()` в диалплан, куда поступают ваши входящие вызовы. С точки зрения биллинга, использование `Progress()` позволяет поставщику услуг знать, что вы обрабатываете вызов, не запуская счетчик биллинга.

Приложение `Playback()` используется для воспроизведения ранее записанного звукового файла в канале. Ввод от пользователя игнорируется, что означает невозможность использования `Playback()` в автосекретаре, например если не хотите принимать ввод в этот момент.[5]

Asterisk поставляется со многими профессионально записанными звуковыми файлами, которые могут быть найдены в каталоге звуков по умолчанию (обычно */var/lib/asterisk/sounds*). При компиляции Asterisk можно установить различные наборы образцов звуков, записанных на различных языках и в различных форматах файлов. Мы будем использовать эти файлы во многих наших примерах. Некоторые из файлов в наших примерах взяты из дополнительного звукового пакета, который мы установили в Главе 3. Вы также можете иметь свои собственные звуковые подсказки, записанные в тех же голосах, что и стоковые подсказки, посетив www.theivrvoice.com. Далее в книге мы поговорим о том, как можно использовать телефон и диалплан для создания и управления собственными системными записями (или импорта *.wav* файлов).

Чтобы использовать функцию `Playback()`: укажите имя файла в качестве аргумента. Например, воспроизведение `Playback(filename)` воспроизведёт звуковой файл с именем *filename.wav*, предполагая, что он находится в каталоге звуков по умолчанию. Обратите внимание, что вы можете включить полный путь к файлу, если захотите, например:

```
Playback(/home/john/sounds/filename)
```

В предыдущем примере будет воспроизводиться *filename.wav* из каталога */home/john/sounds*. Это может быть проблематично из-за потенциальных проблем с правами доступа к файлам. Если вы планируете иметь много пользовательских звуков в своей системе, то вам, вероятно, понадобится создать для них выделенный каталог, а также проверить, чтобы Asterisk имела туда доступ.

Вы также можете использовать относительные пути из каталога звуков Asterisk, как показано ниже:

```
Playback(custom/filename)
```

В этом примере будет воспроизводиться *filename.wav* из подкаталога *custom* каталога звуков по умолчанию (возможно */var/lib/asterisk/sounds/en/custom/filename.wav*). Если указанный каталог

5 Существует еще одно приложение под названием `Background()`, которое очень похоже на `Playback()` за исключением того, что оно позволяет получать ввод данных от вызывающего абонента. Вы можете прочитать больше об этом приложении в Главах 14 и 16.

содержит более одного файла с этим именем, но с разными расширениями, Asterisk автоматически воспроизведёт лучший.[6]

Приложение `Hangup()` делает именно то, что следует из его названия: оно завершает активный канал. Вы должны использовать это приложение в конце контекста когда хотите завершить текущий вызов, чтобы убедиться, что абоненты не продолжают выполнение диалплана таким образом, который вы, возможно, не ожидали. Приложение `Hangup()` не требует никаких аргументов, но вы можете передать код причины ISDN если захотите, например `Hangup(16)`. Код будет переведен в сопоставимое сообщение SIP и отправлен на дальний конец.

По мере работы над книгой мы будем знакомить вас со многими другими приложениями Asterisk, но пока достаточно теории; давайте напишем диалплан!

Базовый прототип диалплана

Итак, повторю, что форма всех диалпланов строится на основе четырех понятий: контекст, расширение, приоритет и приложение (Рисунок 6-3).

Рисунок 6-3. Прототип диалплана

Простой диалплан

Ладно, хватит теории. Откройте файл */etc/asterisk/extensions.conf* в вашем любимом редакторе, и давайте посмотрим на ваш первый диалплан (который был создан в Главе 5). Мы собираемся кое-что добавить к нему.

Hello World

Как это обычно бывает во многих технологических книгах (особенно в книгах по компьютерному программированию), наш первый пример называется “Hello World.”

В первом приоритете нашего расширения мы отвечаем на вызов. Во втором мы проигрываем звуковой файл с именем *hello-world*, а в третьем вешаем трубку. Код, который нас интересует для этого примера выглядит так:

```
exten => 200,1,Answer()
    same => n,Playback(hello-world)
    same => n,Hangup()
```

Если вы следовали Главе 5 - у вас уже будет настроен канал или два, а также пример диалплана, содержащего этот код. Если нет, то вам нужен файл *extensions.conf* в каталоге */etc/asterisk*, содержащий следующий код:

6 Asterisk выбирает лучший файл на основе затрат на транскодинг — то есть она выбирает файл, являющийся наименее трудоемким для преобразования в свой собственный аудиоформат. Когда вы запускаете Asterisk - она вычисляет затраты на перевод между различными аудиоформатами (они часто варьируются от системы к системе). Вы можете увидеть эти затраты на перевод, набрав **`core show translation`** в Asterisk CLI. Приведенные цифры показывают, сколько микросекунд требуется Asterisk для перекодирования одной секунды звука.

```
[general]
[globals]

[sets]
exten => 100,1,Dial(PJSIP/0000f30A0A01) ; Замените 0000f30A0A01 на имя вашего устройства

exten => 101,1,Dial(PJSIP/SOFTPHONE_A)

exten => 102,1,Dial(PJSIP/0000f30B0B02)

exten => 103,1,Dial(PJSIP/SOFTPHONE_B)

exten => 200,1,Answer()
    same => n,Playback(hello-world)
    same => n,Hangup()
```

Если у вас нет настроенных каналов - сейчас самое время создать их. Существует реальное удовлетворение, приходящее от совершения вашего первого вызова в диалплан Asterisk в системе, которую вы построили с нуля. Люди получают эту ухмылку на лицах когда понимают, что они только что создали телефонную систему. Это удовольствие может быть и вашим, поэтому, пожалуйста, не идите дальше, пока не сделаете эту маленькую работу диалплана. Если у вас есть какие-либо проблемы, вернитесь к Главе 5 и проработайте примеры оттуда.

Если у вас еще нет этого кода диалплана, то нужно будет добавить его и перезагрузить диалплан с помощью этой команды CLI:

```
$ sudo asterisk -rvvvvv # ('r' подключение к демону Asterisk; 'v' означает verbosity)
*CLI> dialplan reload
```

или вы можете выполнить команду непосредственно из оболочки с помощью:

```
$ sudo asterisk -rx "dialplan reload" # ('rx' выполнение команды Asterisk и возврат)
```

Вызов расширения 200 с любого из ваших настроенных телефонов[7] должен вознаградить вас дружелюбным голосом Эллисон Смит, говорящим: "Hello, World.”

Если это не работает - проверьте консоль Asterisk на наличие сообщений об ошибках и убедитесь, что ваши каналы назначены контексту `sets`.

Мы не рекомендуем Вам продвигаться вперед в этой книге, пока вы не проверите следующее:

1. Вызовы между добавочными номерами 100 и 101 работают.
2. Вызов расширения 200 воспроизводит “Hello World.”

Хотя этот пример очень короткий и простой - он подчеркивает основные концепции диалплана: контексты, расширения, приоритеты и приложения. Теперь у вас есть фундаментальные знания, на которых строятся все диалпланы.

Когда вы создаете диалплан, будет полезно открыть CLI Asterisk в новом окне. Вы будете часто перезагружать диалплан, и во время тестирования потока вызовов захотите увидеть что происходит. CLI Asterisk полезен для обеих этих вещей.

```
$ sudo asterisk -rvvvvv
*CLI> dialplan reload # из командной строки Asterisk перезагружает диалплан
```

Поэтому лучше всего было бы редактировать файлы конфигураций в одном окне, а перезагружать и отлаживать в другом.

7 Если вы еще не настроили два телефона, пожалуйста, вернитесь к Главе 5 и установите несколько телефонов, чтобы вы могли поиграть с ними. Вы можете уйти только с одним телефоном для тестирования, но на самом деле необходимы два. Есть много бесплатных софтфонов и некоторые из них довольно хороши.

Создание интерактивного диалплана

Диалплан, который мы только что построили, был статическим; он всегда будет выполнять одни и те же действия при каждом вызове. Многие диалпланы также нуждаются в логике для выполнения различных действий на основе ввода пользователя, поэтому давайте посмотрим на это сейчас.

Приложения Goto(), Background() и WaitExten()

Как следует из названия, приложение `Goto()` используется для отправки вызова в другую часть диалплана. `Goto()` требует, чтобы мы передали контекст назначения, расширение и приоритет в качестве аргументов, например:

```
same => n,Goto(context,extension,priority)
```

Мы создадим новый контекст под названием `TestMenu` и создадим расширение в нашем контексте `sets`, которое будет передавать вызовы в этот контекст с помощью `Goto()`:

```
exten => 200,1,Answer()
 same => n,Playback(hello-world)
 same => n,Hangup()

exten => 201,1,Goto(TestMenu,start,1) ; добавьте это в конец
                                      ; контекста [sets]

[TestMenu]
exten => start,1,Answer()
```

Теперь, когда устройство входит в контекст `[sets]` и набирает 201 - вызов будет передан в расширение `start` в контексте `TestMenu` (который в настоящее время не будет делать ничего интересного, потому что у нас есть ещё код для записи).

Мы использовали расширение `start` в этом примере, но могли бы использовать все что угодно в качестве имени расширения: либо числовое, либо буквенное. Мы предпочитаем использовать буквенные-символы для расширений, которые недоступны напрямую, так как это упрощает чтение диалплана. Суть в том, что можно было бы назвать нашей целью расширения `123` или `xyz321`, или `99luftballons`, или всё что угодно чтобы начать. Слово `start` не означает ничего особенного для диалплана - это просто имя расширения.

Одним из наиболее полезных приложений в интерактивном диалплане Asterisk является приложение `Background()`[8]. Как и `Playback()`, оно воспроизводит записанный звуковой файл. Однако в отличие от функции `Playback()`, когда вызывающий абонент нажимает клавишу (или серию клавиш) на клавиатуре своего телефона - он прерывает воспроизведение и передает вызов на добавочный номер, соответствующий нажатой цифре (цифрам). Если вызывающий абонент нажимает 5, например, Asterisk прекратит воспроизведение звуковой подсказки и отправит управление вызовом на первый приоритет расширения 5 (при условии, что расширение 5 существует для отправки вызова).

Наиболее распространенным использованием приложения `Background()` является создание основных голосовых меню (часто называемых автосекретарями, IVR,[9] или телефонными деревьями). Многие компании используют голосовые меню для направления абонентов на соответствующие добавочные номера, тем самым освобождая своих администраторов от необходимости отвечать на каждый вызов.

`Background()` имеет тот же синтаксис что и `Playback()`:

```
[TestMenu]
exten => start,1,Answer()
  same => n,Background(enter-ext-of-person)
```

8 Следует отметить, что некоторые люди ожидают, что `Background()` из-за его названия, будет продолжаться дальше через следующие шаги в диалплане во время воспроизведения звука. На самом деле, его название относится к тому, что он воспроизводит звук в фоновом режиме, ожидая DTMF на переднем плане.

9 Дополнительную информацию об автосекретарях и IVR можно найти в Главе 14.

Если вы хотите, чтобы Asterisk ждала ввода от вызывающего абонента после завершения воспроизведения звуковой подсказки - вы можете использовать `WaitExten()`. Приложение `WaitExten()` ожидает пока вызывающий абонент введет цифры DTMF и используется непосредственно после приложения `Background()`, например:

```
[TestMenu]
exten => start,1,Answer()
  same => n,Background(enter-ext-of-person)
  same => n,WaitExten()
```

Если вы хотите, чтобы приложение `WaitExten()` ждало ввода для ответа определенное количество секунд (вместо использования таймаута по умолчанию)[10] просто передайте количество секунд в качестве первого аргумента `WaitExten()`, например:

```
    same => n,WaitExten(5) ; Передаем аргумент времени для WaitExten()
```

И `Background()` и `WaitExten()` позволяют абоненту вводить цифры DTMF. Затем Asterisk пытается найти расширение, соответствующее введенным абонентом цифрам, в текущем контексте. Если Asterisk найдет совпадение, то отправит вызов на этот добавочный номер. Давайте продемонстрируем, добавив несколько строк в наш пример диалплана:

```
[TestMenu]
exten => start,1,Answer()
    same => n,Background(enter-ext-of-person)
    same => n,WaitExten(5)

exten => 1,1,Playback(digits/1)

exten => 2,1,Playback(digits/2)
```

После внесения этих изменений сохраните и перезагрузите диалплан:

```
*CLI> dialplan reload
```

Если вы звоните на добавочный номер 201, то должны услышать звуковое приглашение, которое говорит: “Enter the extension of the person you are trying to reach". Система будет ждать 5 секунд, пока вы введете цифру. Если вы нажмете 1 или 2: Asterisk будет действовать соответственно расширению и произносить эту цифру вам. Поскольку мы не предоставили никаких дальнейших инструкций, ваш звонок будет закончен. Вы также обнаружите, что если введете другую цифру (например, 3), диалплан не сможет продолжиться.

Давайте немного приукрасим. Мы собираемся использовать приложение `Goto()` чтобы заставить диалплан повторить приветствие после воспроизведения номера:

```
[TestMenu]
exten => start,1,Answer()
   same => n,Background(enter-ext-of-person)
   same => n,WaitExten(5)

exten => 1,1,Playback(digits/1)
   same => n,Goto(TestMenu,start,1)

exten => 2,1,Playback(digits/2)
   same => n,Goto(TestMenu,start,1)
```

Эти новые строки отправят управление вызовом обратно в расширение `start` после воспроизведения набранного номера.

Если посмотрите детали приложения `Goto()`, то обнаружите что вы действительно можете передать в приложение один, два или три аргумента. Если передадите один аргумент - Asterisk предположит что это приоритет назначения в текущем расширении. Если передадите два аргумента - Asterisk будет рассматривать их как расширение и приоритет для перехода в текущем контексте.

10 Смотри функцию диалплана `TIMEOUT()` для получения информации о том, как изменить тайм-ауты по умолчанию. См. Главу 10 для получения информации о том, что такое функции диалплана.

В этом примере мы передали все три аргумента для ясности, но передача только расширения и приоритета имела бы тот же эффект, поскольку контекст назначения совпадает с исходным контекстом.

Обработка неверных значений и тайм-аутов

Нам нужно расширение для недопустимых значений. В Asterisk, когда контекст получает запрос на расширение, которое не является допустимым в этом контексте (например, нажатие 9 в предыдущем примере), вызов отправляется на расширение `i`. Нам также нужно расширение для обработки ситуаций, когда вызывающий абонент не делает ввода (тайм-аут по умолчанию составляет 10 секунд). Вызовы будут отправлены на расширение `t`, если вызывающий слишком долго не нажимает цифру после вызова `WaitExten()`. Вот как будет выглядеть наш диалплан после добавления этих двух расширений:

```
[TestMenu]
exten => start,1,Answer()
    same => n,Background(enter-ext-of-person)
    same => n,WaitExten(5)

exten => 1,1,Playback(digits/1)
    same => n,Goto(TestMenu,start,1)

exten => 2,1,Playback(digits/2)
    same => n,Goto(TestMenu,start,1)

exten => i,1,Playback(pbx-invalid)
    same => n,Goto(TestMenu,start,1)

exten => t,1,Playback(please-try-again)
    same => n,Goto(TestMenu,start,1)
```

Использование расширений `i`[11] и `t` делает наше меню более надежным и удобным для пользователя. Но оно по прежнему все еще довольно ограничено, потому что внешние абоненты все еще не имеют возможности соединиться с живым человеком. Для этого нам нужно будет узнать о приложении `Dial()`.

Использование приложения Dial()

Одной из наиболее ценных особенностей Asterisk является возможность подключения различных абонентов друг к другу. Хотя Asterisk в настоящее время используется в основном для SIP-соединений, она поддерживает широкий спектр типов каналов (от аналоговых до SS7 и различных старых протоколов VoIP, таких как MGCP и SCCP). Asterisk берет на себя большую часть тяжелой работы по подключению и переводу между разрозненными сетями. Все, что вам нужно сделать, это научиться использовать приложение `Dial()`.

Синтаксис приложения `Dial()` является более сложным, чем у других приложений, которые мы использовали до этого, но оно является тем, где происходит большая часть магии Asterisk. `Dial()` принимает до четырех аргументов, которые мы рассмотрим далее.

Синтаксис `Dial()` выглядит следующим образом:

```
Dial(Technology/Resource[&Technology2/Resource2[&...]][,timeout[,options[,URL]]])
```

Проще говоря, вы сообщаете `Dial()`, на какой канал[12] хотите отправить вызов и устанавливаете несколько параметров для настройки поведения. Использование `Dial()` может быть сложным, но в самом основном оно очень простое.

11 Расширение `i` предназначено для перехвата недопустимых значений, предоставленных приложением диалплана, например `Background()`. Оно не используется для сопоставления на неверно набранные номера или несовпадения шаблонов.

12 Или каналы, если вы хотите звонить более чем по одному за раз.

Аргумент 1: назначение

Первый аргумент - это назначение, которое вы пытаетесь вызвать, которое (в самой простой форме) состоит из технологии (или транспорта), через которую выполняется вызов, косой черты и адреса удаленной конечной точки или ресурса.

В эти дни вы, скорее всего, будете использовать PJSIP в качестве типа канала, но в не слишком далеком прошлом общие типы технологий также включали DAHDI (для аналоговых и T1/E1/J1 каналов), старый канал SIP (до PJSIP) и IAX2.[13] Если вы посмотрите на более старый диалплан, то можете увидеть некоторые из этих представленных протоколов. В дальнейшем рекомендуется и поддерживается только PJSIP и DAHDI.

Предположим, что мы хотим вызвать один из наших каналов PJSIP с именем `SOFTPHONE_B`. технология - `PJSIP`, а идентификатор ресурса (или канала) - `SOFTPHONE_B`. Аналогично, вызов устройства DAHDI (определенного в *chan_dahdi.conf)* может иметь пункт назначения `DAHDI/14169671111`. Если бы мы хотели чтобы Asterisk вызывал канал `PJSIP/SOFTPHONE_B` при достижении расширения `103` в диалплане то добавили бы следующее расширение:

```
exten => 101,1,Dial(PJSIP/SOFTPHONE_A)

exten => 103,1,Dial(PJSIP/SOFTPHONE_B)

exten => 200,1,Answer()
```

Мы также можем одновременно набирать несколько каналов, объединяя назначения амперсандом (`&`), например:

```
exten => 101,1,Dial(PJSIP/SOFTPHONE_A)

exten => 103,1,Dial(PJSIP/SOFTPHONE_B)

exten => 110,1,Dial(PJSIP/0000f30A0A01&PJSIP/SOFTPHONE_A&PJSIP/SOFTPHONE_B)

exten => 200,1,Answer()
```

Приложение `Dial()` вызовет все указанные назначения одновременно и соединит входящий вызов с тем каналом назначения, который ответит первым (другие каналы немедленно прекратят звонить). Если приложение `Dial()` не может связаться ни с одним из назначений, Asterisk установит переменную с именем `DIALSTATUS` соответственно причине, по которой не может набрать назначение, и продолжит со следующего приоритета в расширении.[14]

Приложение `Dial()` также позволяет подключаться к удаленной конечной точке VoIP, ранее не определенной в одном из файлов конфигурации канала. Полный синтаксис:

`Dial(technology/`*`user[:password]@remote_host[:port][/remote_extension]`*`)`

Полный синтаксис приложения `Dial()` немного отличается для каналов DAHDI:

`Dial(DAHDI/`*`[gGrR]channel_or_group[/remote_extension]`*`)`

Например, вот как бы вы набрали `1-800-555-1212` на канале DAHDI номер 4:[15]

```
exten => 501,1,Dial(DAHDI/4/18005551212)
```

13 IAX2 (произносится как “EEKS”), является протоколом обмена между Asterisk (v2). В первые дни Asterisk он был популярен для транкинга, поскольку значительно уменьшал накладные расходы сигнализации на нагруженных линиях. Пропускная способность стала намного дешевле, а протокол SIP стал почти повсеместным. Протокол IAX2 больше не поддерживается активно, но он по-прежнему сохраняет некоторую популярность за свою способность обходить брандмауэры и поддержку нескольких медиапотоков. Тем не менее, его использование является устаревшим, и на самом деле не рекомендуется.

14 Мы рассмотрим переменные в разделе “Использование переменных”. В следующих главах мы обсудим, как заставить ваш диалплан принимать решения, основанные на значении `DIALSTATUS`.

15 Имейте в виду: это предполагает, что этот канал подключается к чему-то, что знает, как достичь внешних номеров.

Аргумент 2: таймаут

Вторым аргументом приложения `Dial()` является тайм-аут, заданный в секундах. Если задан тайм-аут, `Dial()` попытается вызвать указанное назначение(я) в течение этого количества секунд, прежде чем сдаться и перейти к следующему приоритету в расширении. Если тайм-аут не указан: `Dial()` будет продолжать набирать вызываемый канал(ы) пока кто-то не ответит или вызывающий абонент не повесит трубку. Давайте добавим тайм-аут в 10 секунд к нашему расширению:

```
exten => 101,1,Dial(PJSIP/SOFTPHONE_A)

exten => 102,1,Dial(PJSIP/0000f30B0B02,10)

exten => 103,1,Dial(PJSIP/SOFTPHONE_B)
```

Если на вызов отвечают до истечения тайм-аута - каналы соединяются и диалплан выполняется. Если адресат просто не отвечает, занят или недоступен иным образом, Asterisk установит переменную с именем `DIALSTATUS`, а затем продолжит работу со следующим приоритетом в расширении.

Давайте поместим то, что мы узнали ранее, в другой пример:

```
exten => 102,1,Dial(PJSIP/0000f30B0B02,10)
   same => n,Playback(vm-nobodyavail)
   same => n,Hangup()
```

Как вы можете видеть: этот пример будет воспроизводить звуковой файл *vm-nobodyavail.gsm*, если вызов остается без ответа (а затем повесит трубку). Обратите внимание: то что мы сделали не является голосовой почтой; мы просто проиграли файл подсказку, которая могла бы быть любым другим аудио файлом. Отправку звонков на голосовую почту мы рассмотрим позже.

Аргумент 3: опции

Третий аргумент для `Dial()` - это строка параметров. Он может содержать один или несколько символов, изменяющих поведение приложения `Dial()`. Список возможных вариантов слишком длинный чтобы охватить его здесь, поэтому для примера рассмотрим один из самых популярных вариантов - опцию `m`. Если вы поместите букву `m` в качестве третьего аргумента - вызывающая сторона услышит музыку удержания вместо звонка во время вызова канала назначения (при условии, конечно, что музыка на удержании была настроена правильно). Чтобы добавить опцию `m` к нашему последнему примеру - мы просто изменим первую строку:

```
exten => 102,1,Dial(PJSIP/0000f30B0B02,10,m)
    same => n,Playback(vm-nobodyavail)
    same => n,Hangup()
```

Аргумент 4: URI

Четвертым и последним аргументом приложения `Dial()` является URI. Если канал назначения поддерживает получение URI во время вызова, указанный URI будет отправлен (например, если у вас есть IP-телефон, поддерживающий получение URI, он появится на дисплее телефона; аналогично, если вы используете софтфон, URI может появиться на экране вашего компьютера). Этот аргумент используется очень редко.

Обновление диалплана

Давайте изменим расширения 1 и 2 в нашем меню, чтобы использовать приложение `Dial()`, и добавим расширения 3 и 4 просто для количества:

```
[TestMenu]
exten => start,1,Answer()
   same => n,Background(enter-ext-of-person)
   same => n,WaitExten(5)

exten => 1,1,Dial(PJSIP/0000f30A0A01,10)
   same => n,Playback(vm-nobodyavail)
   same => n,Hangup()
```

```
exten => 2,1,Dial(PJSIP/0000f30B0B02,10)
   same => n,Playback(vm-nobodyavail)
   same => n,Hangup()

exten => 3,1,Dial(PJSIP/SOFTPHONE_A,10)
   same => n,Playback(vm-nobodyavail)
   same => n,Hangup()

exten => 4,1,Dial(PJSIP/SOFTPHONE_B,10)
   same => n,Playback(vm-nobodyavail)
   same => n,Hangup()

exten => i,1,Playback(pbx-invalid)
   same => n,Goto(TestMenu,start,1)

exten => t,1,Playback(vm-goodbye)
   same => n,Hangup()
```

Пустые аргументы

Обратите внимание, что второй, третий и четвертый аргументы могут быть оставлены пустыми; требуется только первый аргумент. Например, если вы хотите указать параметр, но не тайм-аут, просто оставьте аргумент тайм-аут пустым, например:

```
exten => 4,1,Dial(SIP/SOFTPHONE_B,,m)
```

Использование переменных

Если у вас есть опыт программирования - вы уже понимаете что такое переменная. Если нет, то мы кратко объясним, что это такое и как её использовать. Любая работа диалплана за пределами только что приведенных очень простых примеров значительно выиграет от использования переменных. Они являются одной из полезных функций настраиваемого диалплана, который вы не найдете в типичной проприетарной АТС.

Переменная - это именованный контейнер, который может содержать значение. Думайте о ней как о почтовом ящике. Преимущество переменной заключается в том, что ее содержимое может изменяться, но ее имя остается постоянным. Это означает что вы можете написать код, который ссылается на имя переменной, и не беспокоиться о том, каким будет её значение. Практически невозможно запрограммировать что-либо без переменных.

Существует два способа ссылки на переменную. Чтобы ссылаться на имя переменной, просто введите имя переменной. Если же вы хотите сослаться на значение переменной, необходимо ввести знак доллара, открывающую фигурную скобку, имя переменной и закрывающую фигурную скобку. Итак, используя аналогию с почтовым ящиком, если вы ссылаетесь на сам ящик, просто используйте его имя, если ссылаетесь на его содержимое, используйте обертку `${}`. Переменная с именем MyVar называется `MyVar` и доступ к ее содержимому осуществляется с помощью `${MyVar}`. Вот как мы могли бы использовать переменную внутри приложения `Dial()`:[16]

```
exten => 203,1,Noop(say some digits)
   same => n,Answer()
   same => n,Set(SomeDigits=123)
   same => n,SayDigits(${SomeDigits})
   same => n,Wait(.25)
   same => n,Set(SomeDigits=543)
   same => n,SayDigits(${SomeDigits})
```

В нашем диалплане всякий раз, когда мы ссылаемся на `${SomeDigits}` - Asterisk автоматически заменит его любым значением, присвоенным переменной с именем `SomeDigits`.

16 В частности то, что мы устанавливаем здесь, является переменной канала.

Обратите внимание, что имена переменных чувствительны к регистру. Переменная по имени `SOMEDIGITS` отличается от переменной `SomeDigits`. Вы также должны знать, что любые переменные, заданные Asterisk, будут прописными. Некоторые переменные, такие как `CHANNEL` и `EXTEN`, зарезервированы Asterisk. Вы не должны пытаться установить их. Распространенным является запись глобальных переменных в верхнем регистре и переменных канала в виде Pascal/Camel, но это не является строго обязательным.

Существует три типа переменных, которые мы можем использовать в нашем диалплане: глобальные переменные, переменные канала и переменные среды. Давайте воспользуемся моментом, чтобы посмотреть на каждый тип.

Глобальные переменные

Как следует из их названия, *глобальные* переменные видны всем каналам в любое время. Глобальные переменные полезны тем, что их можно использовать в любом месте диалплана для повышения читаемости и управляемости. Предположим на мгновение, что у вас есть большой диалплан и несколько сотен ссылок на канал `PJSIP/0000f30A0A01`. Теперь представьте, что вы заменили телефон другим устройством (возможно, другим MAC-адресом) и должны теперь пройти через свой диалплан и изменить все эти ссылки на `PJSIP/0000f30A0A01`. Не очень удобно.

С другой стороны, если бы вы определили глобальную переменную, содержащую значение `PJSIP/0000f30A0A01` в начале вашего диалплана, а затем ссылались на нее, то пришлось бы изменить только одну строку кода, чтобы повлиять на все места в диалплане, где использовался этот канал.

Глобальные переменные должны быть объявлены в контексте `[globals]` в начале *extensions.conf*. В качестве примера мы создадим несколько глобальных переменных, которые хранят идентификаторы каналов наших устройств. Эти переменные задаются во время анализа диалплана Asterisk:

```
[globals]
UserA_DeskPhone=PJSIP/0000f30A0A01
UserA_SoftPhone=PJSIP/SOFTPHONE_A
UserB_DeskPhone=PJSIP/0000f30B0B02
UserB_SoftPhone=PJSIP/SOFTPHONE_B
```

Мы вернемся к ним позже.

Канальные переменные

Переменная *канала* - это переменная, связанная только с определенным вызовом. В отличие от глобальных переменных, переменные канала определяются только на время текущего вызова и доступны только для каналов, участвующих в этом вызове.

Существует множество предопределенных переменных канала, доступных для использования в диалплане, которые описаны в Asterisk wiki. Вы определяете переменную канала с расширением 203 и приложением `Set()`:

```
exten => 203,1,Noop(say some digits)
  same => n,Set(SomeDigits=123)
  same => n,SayDigits(${SomeDigits})
  same => n,Wait(.25)
  same => n,Set(SomeDigits=543)
  same => n,SayDigits(${SomeDigits})
```

Вы увидите гораздо больше переменных канала. Читайте дальше.

Переменные среды

Переменные *среды* - это способ доступа к переменным среды Unix из Asterisk. На них ссылаются с помощью функции диалплана `ENV()`[17]. Синтаксис выглядит как `${ENV(var)}`, где *var* - переменная среды Unix, на которую вы хотите ссылаться. Переменные среды обычно не используются в диалплане Asterisk, но они доступны в случае необходимости.

Добавление переменных в ваш диалплан

Теперь, когда мы узнали о переменных, давайте включим их в наш диалплан. Мы добавим три глобальные переменные, которые свяжут имя переменной с именем канала:

```
[general]
[globals]
UserA_DeskPhone=PJSIP/0000f30A0A01
UserA_SoftPhone=PJSIP/SOFTPHONE_A
UserB_DeskPhone=PJSIP/0000f30B0B02
UserB_SoftPhone=PJSIP/SOFTPHONE_B

[sets]
exten => 100,1,Dial(${UserA_DeskPhone})

exten => 101,1,Dial(${UserA_SoftPhone})

exten => 102,1,Dial(${UserB_DeskPhone},10)
   same => n,Playback(vm-nobodyavail)
   same => n,Hangup()

exten => 103,1,Dial(${UserB_SoftPhone})

exten => 110,1,Dial(${UserA_DeskPhone}&${UserA_SoftPhone}&${UserB_SoftPhone})

exten => 200,1,Answer()
```

Теперь давайте обновим наше тестовое меню:

```
[TestMenu]
exten => start,1,Answer()
   same => n,Background(enter-ext-of-person)
   same => n,WaitExten(5)

exten => 1,1,Dial(${UserA_DeskPhone},10)
   same => n,Playback(vm-nobodyavail)
   same => n,Hangup()

exten => 2,1,Dial(${UserA_SoftPhone},10)
   same => n,Playback(vm-nobodyavail)
   same => n,Hangup()

exten => 3,1,Dial(${UserB_DeskPhone},10)
   same => n,Playback(vm-nobodyavail)
   same => n,Hangup()

exten => 4,1,Dial(${UserB_SoftPhone},10)
   same => n,Playback(vm-nobodyavail)
   same => n,Hangup()

exten => i,1,Playback(pbx-invalid)
```

Редко имеет смысл жестко кодировать данные в диалплане. Почти всегда лучше использовать переменную.

Проверьте диалплан на предмет опечаток и протестируйте его. Также вы можете посмотреть как это выглядит в Asterisk CLI при выполнении:

```
# asterisk -rvvvvvv
```

17 Позже мы перейдем к функциям диалплана. Не беспокойтесь слишком о переменных окружения прямо сейчас. Они не важны для понимания диалплана.

```
*CLI> dialplan reload
    -- Executing [201@sets:1] Goto("PJSIP/0000f30A0A01", "TestMenu,start,1")
    -- Goto (TestMenu,start,1)
    -- Exec [start@TestMenu:1] Answer("PJSIP/0000f30A0A01", "")
    -- Exec [start@TestMenu:2] BackGround("PJSIP/0000f30A0A01", "enter-ext-of-person")
    -- <PJSIP/0000f30A0A01> Playing 'enter-ext-of-person.slin' (language 'en')
    -- Exec [1@TestMenu:1] Dial("PJSIP/0000f30A0A01", "PJSIP/0000f30A0A01,10")
    -- Called PJSIP/0000f30A0A01
    -- PJSIP/0000f30A0A01-00000011 is ringing
 == Spawn extension (TestMenu, 1, 1) exited non-zero on 'PJSIP/0000f30A0A01'
```

Объединение переменных

Чтобы объединить переменные, просто поместите их вместе, например:

```
exten => 204,1,Answer()
   same => n,Set(ONETWO=12)
   same => n,Set(THREEFOUR=34)
   same => n,SayDigits(${ONETWO}${THREEFOUR}) ; проще простого
   same => n,Wait(0.2)
   same => n,Set(NOTFIVE=${THREEFOUR}${ONETWO}) ; легче не бывает
   same => n,SayNumber(${NOTFIVE}) ; видите, что мы здесь сделали?
   same => n,Wait(0.2)
   same => n,SayDigits(2${ONETWO}3) ; Вы можете объединять константы и переменные
```

Наследование переменных канала

Переменные канала всегда связаны с исходным каналом, который их задает, и больше недоступны после передачи канала.

Чтобы разрешить переменным канала следовать за каналом при его передаче по системе, необходимо использовать наследование переменных канала. Существует два модификатора, которые позволяют переменной канала следовать за каналом: одиночное подчеркивание и двойное подчеркивание.

Одиночное подчеркивание (_) приводит к тому, что переменная канала наследуется каналом для одной передачи, после чего она больше недоступна для дополнительных передач. Если вы используете двойное подчеркивание (__) - переменная канала будет наследоваться на протяжении всего срока жизни этого канала.

Установка переменных канала для наследования просто требует префикса имени канала с одним или двойным подчеркиванием. Затем на переменные канала ссылаются точно так же, как и обычно.

Вот пример установки переменной канала для наследования одной передачи:

```
exten => example,1,Set(_MyVariable=thisValue)
```

Вот пример установки переменной канала для бесконечного наследования передачи:

```
exten => example,1,Set(__MyVariable=thisValue)
```

Если вы хотите прочитать значение переменной канала - не используете подчеркивание(я):

```
exten => example,1,Verbose(1,Value of MyVariable is: ${MyVariable})
```

Совпадения по шаблонам

Если мы хотим чтобы люди могли набирать номер через Asterisk и подключаться к внешним ресурсам, нам нужен способ сопоставить любой возможный номер телефона, который может набрать вызывающий абонент. Для таких ситуаций Asterisk предлагает создание шаблонов. Шаблоны позволяют создать в диалплане одно расширение, которое соответствует множеству различных номеров. Это чрезвычайно полезно.

Синтаксис сравнения по шаблонам

Когда мы используем шаблоны, определенные буквы и символы в нём представляют то, что мы пытаемся сопоставить. Шаблоны всегда начинаются с подчеркивания (_). Это говорит Asterisk, что мы ищем совпадение по шаблону, а не по явному имени расширения.

Если вы забудете подчеркивание в начале вашего шаблона - Asterisk подумает, что это просто именованное расширение и не будет выполнять сопоставление шаблонов. Это одна из самых распространенных ошибок, которые совершают люди, начинающие изучать Asterisk.

После подчеркивания можно использовать один или несколько следующих символов:

`X`

Соответствует любой одиночной цифре от 0 до 9.

`Z`

Соответствует любой одиночной цифре от 1 до 9.

`N`

Соответствует любой отдельной цифре от 2 до 9.

Другой распространенной ошибкой является попытка использовать буквы `X`, `Z` и `N` буквально в соответствии с шаблоном; для этого оберните их в квадратные скобки (без учета регистра), как пример: `_ale[X][Z]A[N]der`.

`[15-7]`

Соответствует одному символу из указанного диапазона цифр. В этом случае шаблону соответствует один `1`, а также любое число в диапазоне `5`, `6`, `7`.

`.` (период)

Совпадение с подстановочным знаком; соответствует *одному или нескольким* символам, независимо от того, что они из себя представляют.

Если вы не будете осторожны, подстановочные совпадения могут заставить ваши диалпланы делать то, что вы не ожидаете (например, сопоставление встроенных расширений, таких как `i` или `h`). Вы должны использовать подстановочное соответствие в шаблоне только после того, как сопоставили как можно больше других цифр. Например, следующий шаблон никогда не должен использоваться:

```
_.
```

На самом деле, Asterisk предупредит вас, если вы попытаетесь его использовать. Вместо этого, если вам действительно нужно всеохватывающее совпадение шаблона, используйте чтобы соответствовать всем строкам, которые начинаются с цифры, за которой следует один или несколько символов (см. `!` если хотите иметь возможность соответствовать нулю или более символов):

```
_X.
```

Или этот, чтобы соответствовать любой буквенно-цифровой строке:

```
_[0-9a-zA-Z].
```

`!` (bang)

Подстановочный знак соответствия; соответствует *нулю или более символов*, независимо от того, что они из себя представляют.

Чтобы использовать сопоставление шаблонов в вашем диалплане, просто поместите шаблон вместо имени расширения (или номера):

```
exten => _4XX,1,Noop(User Dialed ${EXTEN})
    same => n,Answer()
    same => n,SayDigits(${EXTEN})
    same => n,Hangup()
```

В этом примере шаблон соответствует любому трехзначному расширению от 400 до 499.[18]

Еще одна важная вещь, которую нужно знать о сопоставлении шаблонов, заключается в том, что если Asterisk найдет более одного шаблона, соответствущих набранному расширению - она будет использовать *наиболее точный* (слева направо). Предположим, вы определили следующие два шаблона, и вызывающий абонент набрал 555-1212:

```
exten => _555XXXX,1,Answer()
    same => n,SayDigits(${EXTEN})
exten => _55512XX,1,Answer()
    same => n,Playback(tt-monkeys)
```

В этом случае будет выбрано второе расширение, поскольку оно более конкретно. Загрузите это и сделайте звонки на 5550000, 5550123, 5551212, 5551200, 5551300, 5551299 и так далее чтобы почувствовать, как это работает. Поиграйте с различными совпадениями шаблонов. Например, что будет соответствовать шаблону `_555NNNN`? Что будет соответствовать шаблону `_[0-9]`?

Североамериканский план нумерации - примеры совпадений шаблонов

Этот шаблон соответствует любому семизначному числу, если первая цифра равна 2 или более:

```
_NXXXXXX
```

Предыдущий шаблон будет совместим с любым североамериканским планом нумерации местного семизначного номера.

В областях с 10-значным набором этот шаблон будет выглядеть следующим образом:

```
_NXXNXXXXXX
```

Обратите внимание, что ни один из этих двух шаблонов не будет обрабатывать междугородние звонки. Мы рассмотрим их в ближайшее время.

NANP и мошенничество

Североамериканский план нумерации (NANP) - это общая схема нумерации телефонов, используемая 19 странами Северной Америки и Карибского бассейна. Все эти страны имеют общий код страны 1.

В США и Канаде существует достаточная конкуренция, что вы можете сделать междугородний звонок на большинство номеров в коде страны 1 и ожидать разумной платы. Однако многие люди не понимают, что 17 других стран, многие из которых имеют очень разные правила телекоммуникаций, разделяют NANP. В некоторые из этих мест довольно дорого звонить.

Одна популярная афера с использованием NANP - попытка обмануть наивных североамериканцев в вызове дорогих поминутных платных номеров в карибской стране; абоненты считают, что, поскольку они набрали 1-NPA-NXX-XXXX чтобы дозвониться до номера - они будут платить по своему стандартному национальному междугороднему тарифу. Поскольку в рассматриваемой стране могут быть правила, допускающие такую форму вымогательства, абонент в конечном итоге несет ответственность за оплату вызова.

Возможно, будет разумно блокировать звонки на коды регионов в страны NANP за пределами США и Канады, пока у вас не будет возможности пересмотреть свои тарифы на вызовы в эти страны. Википедия имеет хорошую ссылку на основы того, что вам нужно знать о NANP, в том

18 Мы использовали переменную канала `EXTEN`, без введения в неё. Читайте дальше - она будет рассмотрена ниже в этой главе.

числе какие NPA (коды регионов) какой стране принадлежат.

Давайте попробуем другой:

```
_1NXXNXXXXXX
```

Этот номер будет соответствовать номеру с 1, за которым следует код города между 200 и 999, а затем любое семизначное число, которое не начинается с 0 или 1. В области вызова NANP этот шаблон будет использоваться для сопоставления любого междугороднего номера.[19]

И, наконец, этот:

```
_011.
```

Обратите внимание на период в конце. Этот шаблон соответствует любому числу, которое начинается с 011 и имеет по крайней мере еще одну цифру. В NANP это указывает на международный номер телефона. (Мы будем использовать такие шаблоны в следующем разделе для добавления возможности исходящего набора в нашем диалплане.)

Общие глобальные совпадения шаблонов

За пределами Северной Америки существует большое различие в том, как обрабатывается нумерация; однако некоторые шаблоны являются общими. Вот несколько простых примеров:

```
; UK, Germany, Italy, China, etc.
exten => _00X.,1,noop() ; международный телефонный код
exten => _0X.,1,noop() ; национальный префикс набора номера
exten => 112,1,Noop(--==[ Экстренный вызов ]==--)

; Австралия
exten => _0011X.,1,noop() ; международный телефонный код
exten => _0X.,1,noop() ; национальный префикс набора номера

; Голландский Карибский Бассейн (Саба)
exten => _00X.,1,noop() ; международный
exten => _416XXXX,1,noop() ; локальный (островной)
exten => _0[37]XXXXXX,1,noop() ; звонок на код страны 599 неостровной (не Кюрасао)
exten => _09XXXXXXX,1,Noop() ; звонок на код страны 599 неостровной (Кюрасао)
```

Вам нужно будет понять план набора номера вашего региона, чтобы произвести необходимое совпадение шаблона.

Использование канальной переменной ${EXTEN}

Итак, что произойдет, если вы хотите использовать сопоставление шаблонов, но должны знать, какие цифры были фактически набраны? Введите переменную канала `${EXTEN}`. Всякий раз, когда вы набираете расширение - Asterisk записывает полученные цифры в переменную канала `${EXTEN}`. Мы использовали приложение `SayDigits()`, чтобы продемонстрировать это.

```
exten => _4XX,1,Noop(User Dialed ${EXTEN})
    same => n,Answer()
    same => n,SayDigits(${EXTEN})
    same => n,Hangup()

exten => _555XXXX,1,Answer()
    same => n,SayDigits(${EXTEN})
```

В этих примерах, приложение `SayDigits()` читает номер расширения, которое вы набрали.

19 Если вы выросли в Северной Америке, то можете полагать, что 1, которую вы набираете перед междугородним звонком - это “междугородний код”. Это не совсем правильно. Число 1 также является международным кодом страны для NANP. Имейте это в виду, если отправляете свой номер телефона кому-то в другой стране. Получатель может не знать код вашей страны и поэтому не сможет позвонить вам только с вашим кодом города и номером телефона. Ваш полный номер телефона с кодом страны +1 NPA NXX XXXX (где NPA - ваш код города) - например, +1 416 555 1212. Это также известно как формат E.164 (Wikipedia может рассказать вам все о E.164).

Часто бывает полезно манипулировать ${EXTEN}, удаляя определенное количество цифр в передней части расширения. Это достигается с помощью синтаксиса ${EXTEN:*x*}, где *x* - это позиция начала возвращаемой строки слева направо. Например, если значение ${EXTEN} равно 95551212, ${EXTEN:1} равно 5551212. Давайте попробуем другой пример:

```
exten => _XXX,1,Answer()
    same => n,SayDigits(${EXTEN:1})
```

В этом примере приложение SayDigits() будет начинать со второй цифры и, таким образом, считывать только последние две цифры набранного добавочного номера.

Продвинутые возможности манипуляций с цифрами

Переменная ${EXTEN} в общем случае имеет синтаксис ${EXTEN:*x*:*y*}, где *x* - начальная позиция, а *y* - количество возвращаемых цифр. Учитывая следующую строку набора:

```
94169671111
```

мы можем извлечь следующие строки цифр, используя конструкцию ${EXTENT:*x*:*y*}:

- ${EXTEN:1:3} будет содержать 416
- ${EXTEN:4:7} будет содержать 9671111
- ${EXTEN:-4:4} начнется с 4 цифры с конца и вернет 4 цифры, давая нам 1111
- ${EXTEN:2:-4} начнет со второй цифры и исключит последние четыре цифры, давая нам 16967
- ${EXTEN:-6:-4} начнет с шестой цифры с конца и исключит последние четыре цифры, давая 67
- ${EXTEN:1} даст нам все цифры после первой или 4169671111 (если количество цифр для возврата оставлено пустым, то вернет всю оставшуюся строку)

Это очень мощная конструкция, но большинство из этих вариаций не очень распространены в обычном использовании. По большей части вы будете использовать ${EXTEN} (или, возможно ${EXTEN:1} если вам нужно удалить внешний код доступа, например приставку 9).

Включения (Includes)

Asterisk имеет важную функцию, позволяющую расширениям одного контекста быть доступными из другого контекста. Это достигается за счет использования директивы include, позволяющей нам контролировать доступ к различным разделам диалплана.

Оператор include принимает следующую форму, где *context* - это имя удаленного контекста, который мы хотим включить в текущий:

```
include => context
```

Включение одного контекста в другой позволяет набирать расширения в пределах включенного контекста.

Когда мы включаем другие контексты в наш текущий - мы должны помнить о порядке, в котором мы их включаем. Asterisk сначала попытается сопоставить набранное расширение в текущем контексте. В случае неудачи она затем попытается использовать первый включенный контекст (включая любые контексты, включенные уже в этот контекст), а затем продолжит работу с другими включенными контекстами в том порядке, в котором они были включены.

Мы обсудим директиву include подробнее в Главе 7.

Вывод

Описанное в этой главе - базовый, но функциональный диалплан. Есть еще многое, что мы не рассмотрели, но это - основа. В следующих главах мы продолжим строить свой диалплан на этом фундаменте.

Если части этого диалплана для вас непонятны - вы можете вернуться и перечитать один или два раздела, прежде чем перейти к следующей главе. Крайне важно, чтобы вы поняли эти принципы и то как их применять, поскольку следующие главы основаны на этой информации.

Глава 7. Внешние подключения

Вы не всегда можете контролировать то, что происходит снаружи. Но вы всегда можете контролировать то, что происходит внутри.
– Уэйн Дайер

В предыдущих главах мы рассмотрели много важной информации, необходимой для работы системы Asterisk. Однако нам еще предстоит обсудить то, что жизненно важно для любой АТС: а именно, подключение ее к внешнему миру. В этой главе мы обсудим внешние подключения.

Новаторская архитектура Asterisk была знаменательной в значительной степени из-за того, что она рассматривала все типы каналов как равные. Это отличается от традиционной АТС, где транки (линии соединяющие АТС с внешним миром) и расширения (линии соединяющие пользователей и ресурсы) логически разделены. Тот факт, что диалплан Asterisk обрабатывает все каналы аналогичным образом означает, что в системе Asterisk вы можете очень легко выполнить то, что гораздо сложнее (или невозможно) достичь на традиционной АТС.

Однако, эта гибкость имеет цену. Поскольку система по своей сути не знает разницы между внутренним ресурсом (например, телефонным аппаратом) и внешним (например, каналом телефонной связи), вы должны убедиться, что ваш диалплан обрабатывает каждый тип ресурса соответствующим образом.

Основы транкинга

Целью *транкинга* является обеспечение общего соединения между двумя объектами. У деревьев есть стволы (англ. trunks), и все, что проходит между корнями и листьями - проходит через ствол. Железные дороги используют термин "магистраль (trunks)" для обозначения основной линии, соединяющей фидерные линии вместе.

В телекоммуникациях транк соединяет две системы вместе. Операторы связи используют транки (магистрали) для подключения своих сетей друг к другу. В АТС линии, соединяющие АТС с внешним миром (с точки зрения АТС), обычно называются транками (хотя сами операторы связи обычно не считают их транками). С технической точки зрения определение транка не так ясно, как было раньше (транки АТС использовали совершенно другую технологию от станционных линий, но теперь обе, как правило, являются SIP), но как концепция, транки по-прежнему важны. С SIP все технически одноранговое, поэтому с точки зрения технологии больше нет такой вещи как транк (или, возможно, вернее сказать что все является транком). С функциональной точки зрения все еще полезно иметь возможность различать ресурсы VoIP, подключающиеся к внешнему миру (транки (магистрали), линии и т.д.) и средства IP-телефонии подключенных конечных пользователей (радиостанции, устройства, расширения, телефонные трубки, телефоны и т.д.).

В УАТС Asterisk у вас могут быть транки, которые идут к вашему VoIP-провайдеру для междугородних звонков внутри страны, транки для звонков за границу и транки, соединяющие ваши офисы вместе. Эти транки могут фактически работать через одно и то же сетевое соединение, но в диалплане вы можете относиться к ним совершенно по-разному. У вас даже может быть транк в Asterisk, который просто зацикливается на себе (обычно это какой-то хитрый хак, решающий некоторую проблему с пространством имен или CDR, которую не удалось решить другим способом).

Фундаментальный диалплан для исходящих соединений

В традиционной АТС доступ к внешним линиям обычно осуществляется с помощью кода доступа, который необходимо набрать перед номером.[1] Для этой цели обычно используется цифра 9.

В Asterisk аналогично можно назначить 9 для маршрутизации внешних вызовов, но поскольку диалплан Asterisk намного более интеллектуальный, на самом деле нет необходимости заставлять пользователей набирать 9 перед вызовом. Как правило, у вас будет диапазон номеров для вашей системы (скажем, 100-199) и диапазон кодов функций (от *00 до *99). Все, что находится за пределами этих диапазонов и соответствует шаблону набора номера для вашей страны или региона и может рассматриваться как внешний вызов.

Если у вас есть один оператор, обеспечивающий всю внешнюю маршрутизацию, то вы можете обрабатывать свой внешний набор через несколько простых совпадений шаблонов. Пример в этом разделе действителен для североамериканского плана нумерации (NANP). Если ваша страна не входит в NANP (обслуживающий Канаду, США и многие страны Карибского бассейна), вам понадобится другой шаблон соответствия.

Раздел `[globals]` содержит две переменные, названные `LOCAL` и `TOLL`.[2] Целью этих переменных является упрощение управления вашим диалпланом, если вам когда-либо понадобится изменить провайдера. Они позволяют внести одно изменение в диалплан, которое повлияет на все места, где указан данный канал:

```
[globals]
; Эти каналы одинаковы для asterisk, как и любая конечная точка PJSIP,
; поэтому они будут настроены аналогично телефонным аппаратам.
; Каждый поставщик услуг будет иметь свои собственные требования к конфигурации
; (хотя все они будут похожи)
LOCAL=PJSIP/my-itsp
TOLL=PJSIP/my-other-itsp
```

Раздел `[external]` содержит фактический код диалплана, распознающий набранные номера и передающий их в приложение `Dial()`: [3]

```
[external]
exten => _NXXNXXXXXX,1,Dial(${LOCAL}/${EXTEN}) ; 10-значный шаблон для NANP
exten => _NXXXXXX,1,Dial(${LOCAL}/${EXTEN}) ; 7-значный шаблон NANP
exten => _1NXXNXXXXXX,1,Dial(${TOLL}/${EXTEN}) ; шаблон международного направления
; для NANP
exten => _011.,1,Dial(${TOLL}/${EXTEN}) ; Шаблон международного вызова,
; сделанного из NANP
; Этот раздел функционально совпадает с приведенным выше разделом.
; Это для людей, которым нравится набирать '9' для их звонков.
exten => _9NXXNXXXXXX,1,Dial(${LOCAL}/${EXTEN:1})
exten => _9NXXXXXX,1,Dial(${LOCAL}/${EXTEN:1})
exten => _91NXXNXXXXXX,1,Dial(${TOLL}/${EXTEN:1})
exten => _9011.,1,Dial(${TOLL}/${EXTEN:1})
```

В любом контексте, используемом для комплектов или пользовательских устройств, вы будете использовать директиву `include=>`, чтобы разрешить доступ к контексту `external`:

```
[sets]
include => external
```

Крайне важно, чтобы вы не включали доступ к внешним линиям в любом контексте, который может обрабатывать входящий вызов. Риск здесь заключается в том, что фишинговый бот может в итоге получить доступ к вашим исходящим транкам (вы будете удивлены тем, насколько эти фишинговые боты распространены).

1 В системе клавиш каждая линия имеет соответствующую кнопку на каждом телефоне, и доступ к линиям осуществляется нажатием нужной клавиши линии.

2 Вы можете назвать их как угодно.

3 Дополнительную информацию о совпадениях шаблонов см. в Главе 6.

Мы не можем не подчеркнуть, насколько важно, чтобы вы гарантировали, что никакой внешний ресурс не может получить доступ к вашим платным линиям.

ТфОП

Телефонная сеть общего пользования (ТфОП, на англ. PSTN) существует уже более ста лет. Это предшественник многих технологий, формирующих наш мир сегодня, от интернета до MP3-плееров.

Использование линий ТфОП старой школы в системах Asterisk больше не является распространенным явлением. Технические сложности, затраты и ограничения устаревшей технологии оправданы только в ситуациях, когда надежное подключение к интернету недоступно (и даже тогда традиционные линии будут проблематичным выбором). Даже сами операторы связи в значительной степени перешли на VoIP для своих внутренних транков.

ТфОП пора на пенсию

Больше, чем любой технический фактор, пожалуй, самым значительным гвоздем в гробу ТфОП является тот факт, что большинство технических экспертов в области традиционной телефонии близки или достигли пенсионного возраста и современные дети не интересуются такими вещами. Суть в том, что вы все чаще будете обнаруживать, что поставщики услуг больше не имеют квалифицированного персонала, необходимого для развертывания традиционных услуг ТфОП. Все крутые ребята изучают VoIP (который в конечном счете является просто сетевой технологией) и все операторы связи делают все возможное и самое яркое в сфере бизнеса VoIP/SIP.

Таким образом, если раньше вы не могли не подключиться через PRI, в настоящее время это уже не так. На самом деле многие компании поставляют PRI через SIP соединения, что для Asterisk является избыточным и ненужным.

Там, где ТфОП все еще может оставаться в силе в течение нескольких лет - это телефонные номера. Если бы VoIP был изобретен без предшествующего ему ТфОП маловероятно, что когда-либо было бы изобретено что-то вроде телефонного номера. Тем не менее, они у нас есть, и мы их используем, и причина, по которой мы делаем это, возможно, не столько из-за какой-либо полезности, которую они обеспечивают, а скорее из-за того, что они управляются сложным, многонациональным консорциумом органов по стандартизации и кураторами, обеспечивающими целостность глобального плана маршрутизации вызовов.

Можно представить, что если бы интернет определял телефонную сеть (и телефонные звонки были такими же бесплатными как электронная почта), все наши SIP-телефоны, вероятно, звонили бы весь день с одним спам-предложением за другим. Это так и происходит, но в значительно меньших масштабах из-за того, что телефонный звонок стоит денег. И даже если он стоит всего лишь копейки - этого достаточно, чтобы делать невыгодным большую часть бессмысленного спама.

Еще одной особенностью ТфОП является соответствие стандартам и совместимость. Если вы посмотрите на любой интернет-голосовой продукт - он либо является проприетарным, огороженным садами, либо управляется сообществом и не сможет получить какую-либо полезную тягу. Мы считаем, что это не изменится до тех пор, пока не будет создан какой-то механизм доверия, гарантирующий проверку личности входящих абонентов каким-то широко признанным органом.

Традиционные транки ТфОП

Этот раздел был написан как дань уважения телекоммуникационной отрасли и истории самой Asterisk. Отчасти это связано с тем, что раз Asterisk могла взаимодействовать со столькими различными типами линий старой школы, потому

она и добилась раннего успеха. В наши дни использование этих старых линий по большей части исчезло в истории.

Существует два типа фундаментальных технологий, используемых поставщиками услуг ТфОП для предоставления телефонных линий: аналоговые и цифровые.

Аналоговая телефония

Первые телефонные сети были полностью аналоговыми. Звуковой сигнал, который вы сгенерировали своим голосом, использовался для генерации электрического сигнала, который передавался на другой конец провода. Электрический сигнал имел те же характеристики, что и производимый звук.

Аналоговые линии имеют несколько характеристик, отличающих их от других линий, которые вы можете подключить к Asterisk:

- Отсутствует сигнальный канал — сигнализация состояния линии является электромеханической, а адресация осуществляется с использованием внутриполосных звуковых сигналов.
- Механизм наблюдения за отключением обычно задерживается на несколько секунд и не является полностью надежным.
- Контроль дальнего конца минимален (например, контроль ответа отсутствует).
- Различия в линиях означают, что звуковые характеристики будут варьироваться от линии к линии и потребуют настройки.

Входящие аналоговые линии, которые вы захотите подключить к системе Asterisk, должны быть подключены к порту Foreign eXchange Office (FXO). Поскольку в любом стандартном компьютере не существует такого понятия, как порт FXO, то перед подключением традиционных аналоговых линий системе должен быть предоставлен какой-либо порт FXO. Такие компании, как Digium и Sangoma предлагают карты с таким функционалом, но вы также можете приобрести устройство SIP, которое предоставляет эти порты.

FXO и FXS

Для любой аналоговой линии есть два конца: офис (как правило, центральный офис ТфОП) и станция (как правило, телефон, но также может быть карта, такая как модем или линейная карта в АТС).

Центральный офис отвечает за:

- Питание на линии (номинально 48 Вольт постоянного тока)
- Напряжение звонка (номинально 90 Вольт переменного тока)
- Предоставление гудка (сигнала ответа станции)
- Обнаружение состояния трубки (положена или поднята)
- Отправка дополнительной сигнализации, такой как идентификатор вызывающего абонента (Caller ID)

Станция отвечает за:

- Обеспечение звонка (или, по крайней мере, возможности каким-то образом обрабатывать напряжение звонка)
- Предоставление номеронабирателя (или какой-либо способ отправки DTMF)
- Предоставление рычажного переключателя для указания состояния линии

Порт Foreign eXchange (FX) *называется тем, к чему он подключается*, а не тем, что он делает. Так, например, порт Foreign Exchange Office (FXO) на самом деле является станцией: он соединяется с центральным офисом. Порт Foreign eXchange Station (FXS) - это фактически порт, предоставляющий услуги центрального офиса (другими словами, вы бы подключили аналоговый аппарат к порту FXS).

Обратите внимание, что мы не можем перейти с FXO на FXS просто изменив настройки. Порты FXO и FXS требуют совершенно разную электронику.

Это старая школа, ребята. Вы можете запускать старые телефоны возрастом более 100 лет с порта FXS!

Мы не рекомендуем использовать аналоговые транки в системе Asterisk. Их конфигурация и использование выходят за рамки данной книги.[4]

Цифровая телефония

Цифровая телефония была разработана для преодоления многих ограничений аналоговой. Некоторые из преимуществ цифровых линий включают:

- Отсутствие потери амплитуды на больших расстояниях
- Снижение уровня шума на линиях (особенно на междугородних дистанциях)
- Возможность выполнять более одного вызова через физический канал
- Более быстрая установка соединения и разъединение
- Более богатая сигнальная информация (особенно при использовании ISDN)
- Более низкая стоимость для поставщиков услуг
- Более низкая цена для клиентов (на более высоких плотностях)

Существует несколько типов цифровых линий, получивших широкое применение в телекоммуникационной отрасли:

T1 (24 канала)

Используется в Канаде и США(в основном для ISDN-PRI)[5]

E1 (32 канала)

Используется в остальном мире (ISDN-PRI или MFC/R2)

BRI (2 канала)

Используется для линий ISDN-BRI (Евро-ISDN)

Обратите внимание, что физическая линия может быть дополнительно определена протоколом, работающим по схеме. Например, T1 может использоваться для ISDN-PRI или CAS, а E1 может использоваться для ISDN-PRI, CAS или MFC/R2.

Трудно обосновать использование этих типов линий. По сравнению с протоколами VoIP, они стали дорогими, сложными и несколько негибкими. Если вам нужно подключить такие линии к системе Asterisk - мы рекомендуем какое-то шлюзовое устройство для преобразования линии в SIP, а затем подключиться через SIP к вашей системе Asterisk. Если вам нужна система с одним шасси, такие компании как Digium и Sangoma предлагают цифровые карты ТфОП, которые могут быть

4 Мы должны отметить, что в прошлом мы много писали на эту тему и эта работа была выпущена под лицензией Creative Commons и находится в свободном доступе.

5 В Японии также используется линия, называемая J-1, которую проще всего назвать 24-канальной E1.

установлены непосредственно на ваш сервер Asterisk; они подключаются к Asterisk через драйвер канала DAHDI. Использование этой технологии выходит за рамки данной книги.[6]

VoIP

По сравнению с длительной историей телекоммуникационной отрасли,[7] VoIP по-прежнему является относительно новой концепцией. За столетие, или около того, до появления VoIP единственным способом подключить вас к ТфОП было использование каналов, предоставленных для этой цели вашей местной телефонной компанией. Теперь же VoIP позволяет устанавливать соединения между конечными точками без участия ТфОП (хотя в большинстве сценариев VoIP в какой-то момент все равно будет присутствовать компонент ТфОП, особенно если используется традиционный телефонный номер E.164). ТфОП по-прежнему контролирует телефонные номера и мы будем использовать их до тех пор, пока кто-нибудь не придумает механизм адресации на основе интернета, который не будет подвергаться злоупотреблениям, как электронная почта.[8]

Преобразование сетевых адресов (NAT)

Если вы собираетесь использовать VoIP через любой вид глобальной сети (например, интернет), вы будете иметь дело с брандмауэрами и довольно часто с преобразованием сетевых адресов (NAT).[9] Базовое понимание того, как протоколы SIP и RTP работают вместе для создания VoIP-вызова может быть полезно в понимании и в отладке функциональных проблем (таких как проблема “односторонней слышимости” часто возникающая при ошибках конфигурации NAT). NAT позволяет использовать один внешний IP-адрес совместно несколькими устройствами за маршрутизатором. Поскольку NAT обычно обрабатывается в брандмауэре - он также является частью уровня безопасности между частной сетью и интернетом.

VoIP-вызов с использованием SIP не состоит только из сигнальных сообщений для настройки вызова (часть SIP-соединения). Он также требует потоков RTP (медиапотоки), которые несут фактическое аудио соединение,[10] как показано на Рисунке 7-1.

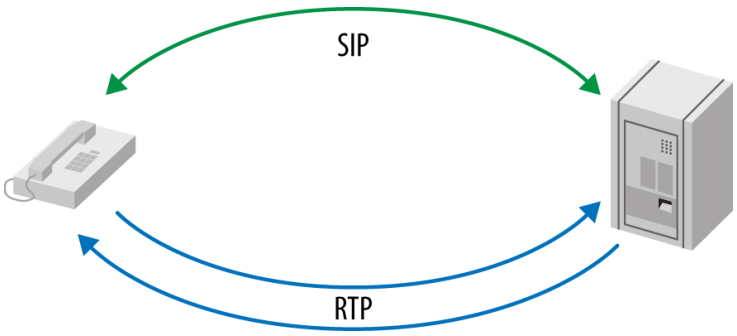

Рисунок 7-1. SIP и RTP

Использование отдельного протокола для передачи звука - это то, что может усложнить обход NAT для VoIP-соединений, особенно если удаленные телефоны находятся за одним NAT, а УАТС - за

6 Мы хотели бы еще раз отметить, что мы много писали о цифровых микросхемах и DAHDI в предыдущих выпусках, и этот объем работ был выпущен по лицензии Creative Commons и находится в свободном доступе. Также Sangoma/Digium предоставляют подробные инструкции по установке и настройке своих карт PSTN. Если вы хотите использовать эту технологию - воспользуйтесь услугами профессионального технического ресурса. Это сложный материал полный нюансов, с которым вам не понравится играть, если у вас не было какого-то подобного опыта. Данный материал также необязателен для изучения или понимания Asterisk.

7 «Длительной» по отношению к другим электронным технологиям.

8 На мгновение просто представьте, что на ваш телефон приходит такое же количество спам-звонков, как спам-писем на адрес электронной почты. Фактически, то что ТфОП регулируется, стоит денег и использование телефонных номеров служит для ограничения чумы спама, от которого пострадала электронная почта.

9 Страница Wikipedia по трансляции сетевых адресов является всеобъемлющей и полезной. Эта страница является неплохим началом для получения дополнительной информации о различных типах NAT и о том, как он работает в целом.

10 SIP-это не единственный протокол, использующий RTP для передачи медиапотоков.

другим. Проблема вызвана тем фактом, что в то время, как SIP-сигнализация обычно разрешена в брандмауэрах на обоих концах, потоки RTP не могут быть распознаны как часть сеанса SIP, и, таким образом, будут проигнорированы или заблокированы, как показано на Рисунке 7-2. Эффект блокировки одного или обоих потоков RTP заключается в том, что пользователи будут видеть что их вызовы проходят, и смогут отвечать на них, но не смогут слышать другого абонента (или не смогут быть услышаны).

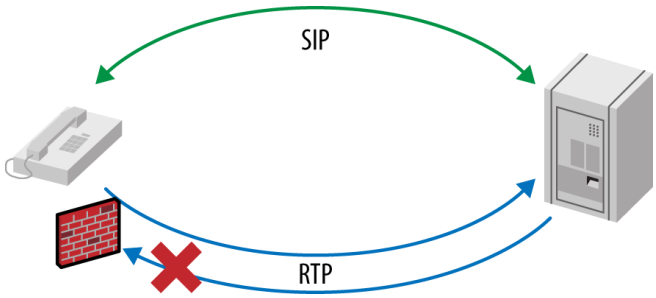

Рисунок 7-2. RTP заблокирован брандмауэром

В этом разделе мы обсудим некоторые методы, которые вы можете использовать для устранения проблем, вызванных NAT. Существует два различных сценария, которые необходимо рассмотреть, каждый из которых требует определения параметров в файле *pjsip.conf*. Проблемы NAT могут быть раздражающими при устранении неполадок, поскольку в продакшене существует множество различных типов брандмауэров и множество различных способов их настройки.

В общем, вам нужно добавить следующие параметры в секцию транспорта вашего файла */etc/asterisk/pjsip.conf*:

```
[transport-udp]
type=transport
protocol=udp
bind=0.0.0.0
local_net=x.x.x.x/xx                ; IP/CIDR вашей внутренней сети
external_media_address=x.x.x.x      ; Внешний IP-адрес для использования в обработке RTP
external_signaling_address=x.x.x.x  ; Внешний адрес для SIP-сигнализации
```

Если вы хотите узнать внешний адрес вашей УАТС, выполните следующие действия из командной строки:

```
$ dig +short myip.opendns.com @resolver1.opendns.com
```

Вероятно, установка этих параметров безопасна для всех случаев, но будьте готовы экспериментировать, комментируя настройки и перезагружая PJSIP для тестирования различных сценариев.

Конечные точки за NAT

Если ваши телефонные аппараты находятся за удаленным NAT, в таблице `ps_endpoints` вашей базы данных могут быть параметры, которые следует настроить из настроек по умолчанию. Вам нужно будет поэкспериментировать с изменением следующих значений выбрав между `yes` и `no`.

```
MySQL> update ps_endpoints set rtp_symmetric='yes',force_rport='yes',rewrite_contact='yes'
```

Другие параметры, которые вы можете посмотреть, включают `media_address` и `direct_media`.

При внесении изменений учитывайте значения по умолчанию. Если вы сомневаетесь - установите значение поля, которое вы изменили на `NULL` - это фактически вернет его к значению по умолчанию.

Сохранение удаленного брандмауэра открытым

Иногда возникает проблема с SIP-телефоном, когда телефон регистрируется и функционирует при первой загрузке, но затем внезапно становится недоступным. Это часто происходит если

удаленный брандмауэр не видя никакой активности, поступающей от устройства, закрывает внешнее соединение с телефоном, и таким образом, АТС теряет возможность подавать сигнал на его вызов. Эффект заключается в том, что если АТС попытается отправить вызов на телефон - она не сможет подключиться (удаленный брандмауэр отклонит соединение). Если, с другой стороны, пользователь делает вызов, то в течение нескольких минут устройство снова сможет принимать входящие вызовы. Естественно это может путать пользователей.

Относительно простое решение этой проблемы[11] включает в себя установку таймера регистрации на удаленном телефоне на достаточно низкое значение, которое будет стимулировать соединение каждую минуту или около того, и, таким образом, убедит брандмауэр, что этому соединению может быть позволено существовать на некоторое время дольше. Это небольшой хак, но он оказался успешным. Проблема с предложением универсального решения заключается в том, что существует множество различных моделей брандмауэров: от недорогих устройств потребительского класса до сложных контроллеров границ сеанса, и это одно из немногих решений, которое, по-видимому, надежно решает проблему почти во всех случаях.

Этот подход лучше всего подходит для небольших систем (менее 100 телефонов). Большая система с сотнями или тысячами телефонов не будет хорошо обслуживаться этим решением, так как будет увеличена нагрузка на систему из-за почти постоянного потока регистраций с удаленных телефонов. В таком случае необходимо будет более тщательно продумать общую конструкцию (например, вместо Asterisk можно было бы использовать выделенный сервер регистратора для обработки регистрационного трафика).

В идеальном мире вы могли бы указать конкретную модель брандмауэра и разработать конфигурацию для этих брандмауэров, обеспечивающую правильную обработку вашего SIP-трафика. На самом деле, вы столкнетесь не только с разными моделями брандмауэров, но даже с разными версиями прошивки для одной и той же модели брандмауэра.

Asterisk за NAT

Во-первых, мы должны сказать вам, что размещение вашей АТС за NAT не рекомендуется. Гораздо лучше обеспечить брандмауэр без уровня NAT (особенно если у вас есть конечные точки, не находящиеся в той же сети, что и УАТС).

Если у вас возникли сложности с работой АТС за NAT - вам нужно будет плотно взаимодействовать с вашей командой сетевых инженеров, чтобы убедиться в корректной работе вашего устройства NAT (как правило брандмауэра). Если же их навыки окажутся недостаточными для решения этой задачи - Вам могут потребоваться услуги внешнего консультанта, обладающего достаточной квалификацией для настройки прохождения трафика SIP/RTP через NAT. Как мы уже говорили - нахождение вашей АТС за NAT не рекомендуется.

Как правило, конечные точки также будут находиться за NAT, и, таким образом, у вас будет сценарий с двойным NAT, который, вероятно, потребует нескольких часов экспериментов с различными настройками не только в Asterisk, но и в брандмауэре чтобы добиться успеха. Помните - очень важно чтобы вы проверяли звук в обоих направлениях; недостаточно просто проверить, что звонки можно совершать и отвечать на .

В сценарии, где нет выбора, кроме как использовать двойной NAT, мы рекомендуем выяснить, можно ли использовать VPN между УАТС и удаленными конечными точками. Во многих случаях его будет легче настроить. Мы хотели бы сказать, что есть простой и надежный способ, который будет работать во всех случаях, но, к сожалению, такого метода нет.

Для работы с NAT Вы также можете изучить такие технологии как STUN, TURN и ICE. Подробности использования этих технологий выходят за рамки данной книги, так как они требуют внешних

11 Вопрос о том, является ли это наилучшим решением, все еще обсуждается.

серверов, но многие люди успешно применили эти протоколы там, где другие методы потерпели неудачу.

Терминация и инициирование ТфОП

Передача вызовов между средой VoIP и ТфОП требует некоего шлюза для преобразования сигнализации VoIP (обычно SIP) в нечто совместимое с протоколами ТфОП. Эти процессы называются *инициированием* и *терминацией* (Рисунок 7-3).

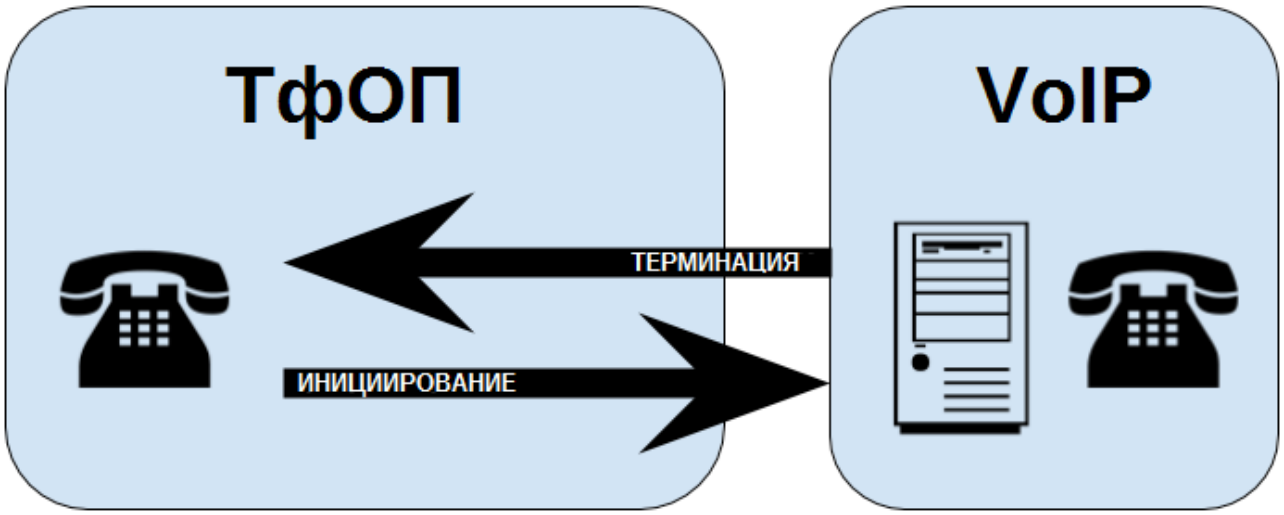

Рисунок 7-3. Терминация и инициирование ТфОП

Люди часто путают термины *инициирование* и *терминация*. Для нас полезно помнить, что, поскольку ТфОП уже существовал когда появился VoIP - термины развивались по отношению к нему. В идеале процессы, вероятно, должны называться *инициирование* ТфОП и *терминация* ТфОП, и мы рекомендуем вам запомнить их таким образом.[12]

Терминация ТфОП

До тех пор, пока VoIP полностью не заменит ТфОП будет необходимость подключения вызовов из VoIP-сетей к телефонной сети общего пользования. Этот процесс называется *терминацией* (Рисунок 7-4).

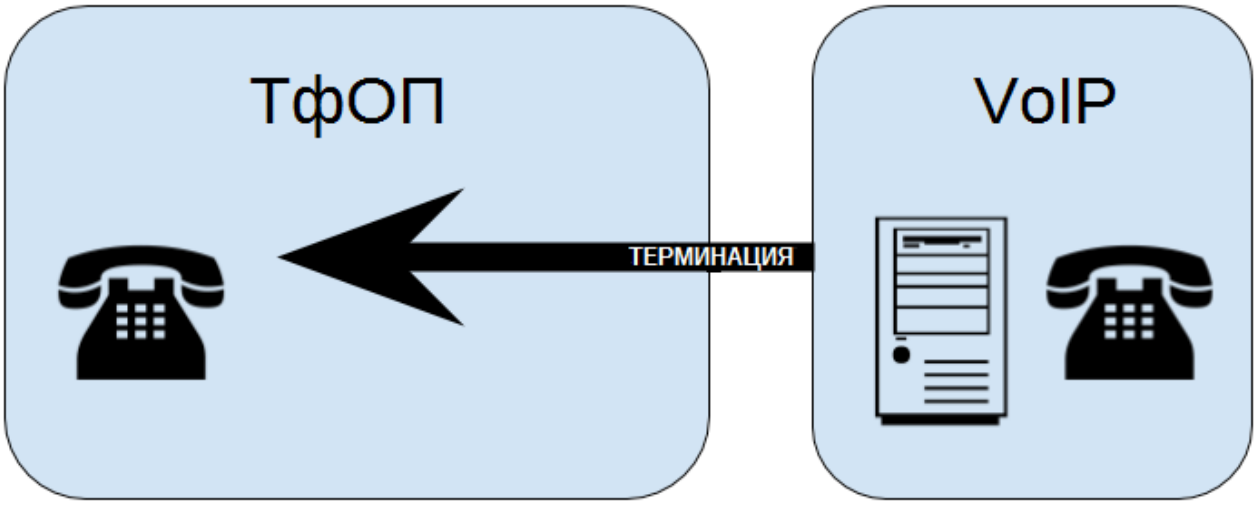

Рисунок 7-4. Терминация ТфОП

12 И, возможно, даже использовать их таким образом в разговоре, так как многие люди смущены этими терминами, и мало кто признает это, когда вы говорите с ними.

Хотя вы можете сконструировать систему Asterisk для работы в качестве шлюза терминации (используя некоторые интерфейсы ТфОП), на практике вы с большей вероятностью будете использовать *поставщика услуг интернет-телефонии* (ITSP, также иногда называемый *оператором VoIP*) для терминации ваших телефонных звонков. Интернет-провайдеры обычно вкладывают огромные инвестиции в инфраструктуру и вам будет трудно сделать что-то лучше не тратя кучу денег.

Если вам действительно нужно подключить систему Asterisk непосредственно к ТфОП, вам потребуется следующее:

- Соответствующая линия(и) от телеф.комп. ТфОП (аналоговая, BRI, PRI, SS7, MFC/R2 и т.д.)
- Подходящее оборудование для подключения к этой линии (FXO, BRI, T1, E1 и т.д.)
- Эхоподавление (аппаратное или программное)
- Навыки, необходимые для правильной настройки вашего оборудования для оператора, с которым вы имеете дело (есть множество вариантов каждой из этих типов линий, и это может быть трудно даже для тех, кто хорошо знает технологию)[13]

Кроме того, вам часто придется обрабатывать гораздо более сложную логику маршрутизации, которая учитывает такие вещи, как география, корпоративная политика, стоимость, доступные ресурсы и т.д.

Для того, чтобы отправить ваши вызовы на ITSP - ваш диалплан должен выглядеть примерно так:

```
; Для систем NANP
[to-pstn] ; Да, мы проходим через ITSP, но ТфОП - это наш пункт назначения
exten => _1NXXNXXXXXX.,1,Dial(${TOLL}/${EXTEN}) ;  код страны плюс номер телефона

; Добавляем '1' и отправляем
exten => _NXXNXXXXXX.,1,Dial(${LOCAL}/1${EXTEN}) ; код страны плюс номер телефона

; Отбрасываем '011' и отправляем
exten => _011X.,1,Dial(${TOLL}/${EXTEN:3}) ; код страны плюс номер телефона

; Вызов экстренных служб
exten => 911,1,Dial(${LOCAL}/911) ; определение этого потребует информации от вашего
оператора

; Большая часть остального мира
[to-pstn]
; Убрать префикс NDD, добавить код страны и отправить
exten => _0X.,1,Dial(${TOLL}/<добавить сюда код страны>${EXTEN:1})

; Убрать префикс IDD и отправить
exten => _00X,1,Dial(${LOCAL}/${EXTEN:2}) ; код страны плюс номер телефона

; Вызов экстренных служб (и других служб)
exten => 11X,1,Dial(${LOCAL}/${EXTEN}) ; для определения этого потребуется информация от
вашего оператора
```

Учитывая, что большинство каналов ТфОП позволит вам набрать любой номер в любой точке мира, и учитывая, что вы будете платить за все понесенные расходы - мы ещё раз подчеркиваем важность обеспечения безопасности на машине шлюза, обеспечивающего терминацию ТфОП. Преступники прикладывают много усилий для взлома телефонных систем (особенно плохо защищенных систем Asterisk), и если вы не уделите пристального внимания всем аспектам безопасности, то станете жертвой мошенничества. Это лишь вопрос времени.

Не допускайте никаких незащищенных VoIP-соединений в любом контексте, содержащем терминацию ТфОП.

13 Поверьте нам, Джим Ван Меггелен работал с этим материалом в течение многих лет, прежде чем попасть в VoIP.

Терминация, как правило, будет более сложной, чем мы описали здесь — даже если вы используете ITSP в качестве своего оператора, но основная концепция довольно проста: сопоставьте шаблон номера, который ваши пользователи могут набрать, подготовьте его для оператора, удалив или добавив необходимые цифры, и отправьте вызов соответствующей конечной точке PJSIP (транку). Мы здесь только обсудили диалплан; в более позднем разделе мы обсудим, как настроить SIP-транки для передачи этого трафика.

Инициирование ТфОП

Вы также можете принимать звонки от ТфОП в свою сеть VoIP. Процесс выполнения этого обычно называют *инициированием*. Это просто означает что вызов инициирован в ТфОП (Рисунок7-5).

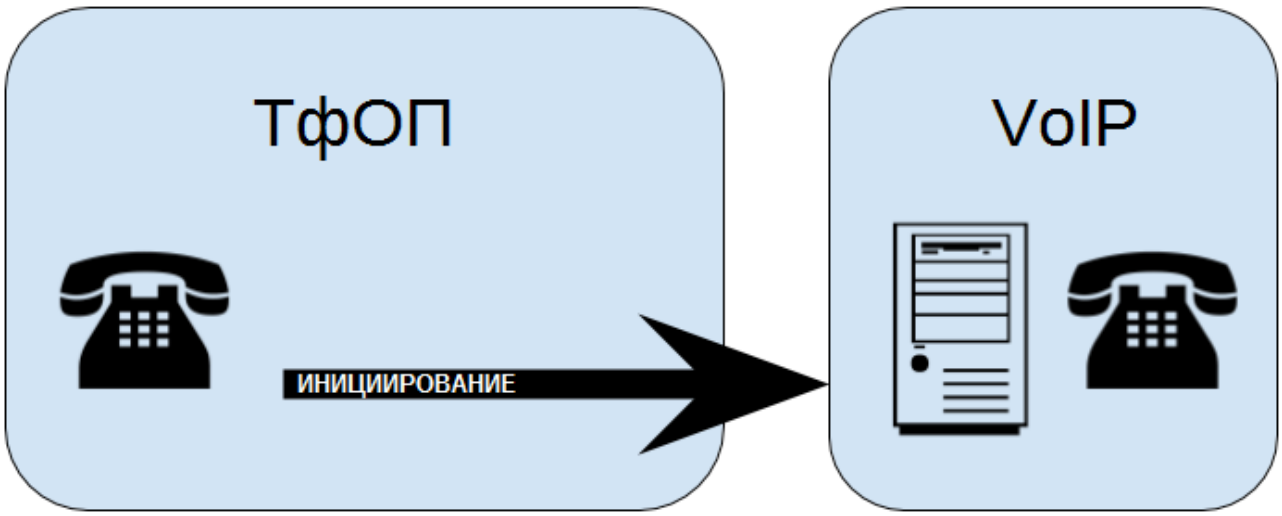

Рисунок 7-5. Инициирование ТфОП

Для того, чтобы обеспечить инициирование - требуется номер телефона.

В старые добрые времена, когда VoIP и Asterisk были молоды, для людей было довольно распространено обрабатывать подключение линии к ТфОП самостоятельно, используя аналоговые или цифровые транки, предоставляемые местной телефонной компанией. По большей части этот тип соединения теперь обрабатывается ITSP, и вам просто нужно подключить вашу систему к VoIP-оператору через SIP-транк.

Телефонные номера — при использовании в целях инициирования обычно называются DID'ами (номера Direct Inward Dialing). Ваш оператор связи отправит вызов вниз по каналу в вашу систему и передаст DID (или специальные полученные цифры в некоторых случаях[14]), которые будет интерпретировать диалплан Asterisk. Другими словами - вам понадобится контекст диалплана, принимающий входящие вызовы от вашего оператора, с расширениями или шаблонами, которые будут коррелировать с вашими DID.

Чтобы принять вызов по линии VoIP - вам нужно будет обработать цифры, посылаемые Вам поставщиком услуг (DID или номер телефона). Номера DNIS и DID не должны совпадать, но, как правило, будут. Ранее провайдер обычно спрашивал, в каком формате вы хотите получать цифры. В настоящее время оператор VoIP, как правило, говорит вам в каком формате будет отправлять, и вы

14 В традиционных УАТС назначение DID состояло в том, чтобы разрешить подключение непосредственно к дополнительному номеру в офисе. Многие УАТС не могут поддерживать такие понятия, как перевод номера или гибкие длины цифр, и поэтому оператор должен передавать добавочный номер, а не набранный номер (который также назывался номером DNIS, от Directory Number Information Service). Например, телефонный номер 416-555-1234 мог бы быть сопоставлен с добавочным номером 100, и таким образом оператор отправил бы цифры 100 в УАТС вместо DNIS 4165551234. Если вы когда-нибудь замените старую АТС системой Asterisk, то можете обнаружить данные переводы при миграции диалпланов, и вам нужно будет получить список сопоставлений между номерами, набираемыми абонентом, и номерами, отправляемыми на УАТС. Было также распространено видеть, что оператор передает только последние четыре цифры номера DNIS, которые УАТС затем переводит во внутренний номер. С VoIP-транками это будет редко иметь место, но имейте в виду, что это возможно.

должны принять его. Два распространенных формата: DNIS (который по сути является цифрами вызываемого DID) или E.164, что означает, что они будут включать код страны с номером.

В диалплане входящий канал связывается с контекстом, который будет знать как обрабатывать входящие номера. Он может выглядеть примерно так:

```
[from-pstn]
exten => _X.,1,Verbose(2,Incoming call to ${EXTEN})
same => n,Goto(number-mapping,${EXTEN},1)

[number-mapping]
exten => 4165550100,1,Goto(sets,100,1)
exten => 4165550101,1,Goto(sets,101,1)
exten => 4165550102,1,Goto(sets,102,1)
exten => 4165550103,1,Goto(sets,103,1)
exten => 4165554321,1,Goto(main-menu,${EXTEN},1)
exten => 4165559876,1,VoiceMailMain() ; удобный ход для прослушивания голосовых сообщений
exten => i,1,Verbose(2,Incoming call to invalid number)
```

В контексте `number-mapping` вы явно перечисляете все идентификаторы DID, которые ожидаете принимать, а также обработчик ошибок для всех неперечисленных DID (вы можете отправлять недопустимые номера на приёмную или в автосекретарь или даже в некоторый контекст, воспроизводящий оповещение об ошибке).

Теперь мы готовы обсудить, как настроить транки для передачи вашего внешнего трафика.

Настройка SIP-транков

SIP является самым популярным из VoIP-протоколов. Настолько, что термины *VoIP* и *SIP* стали означать почти одно и то же. В предыдущих изданиях этой книги мы рассмотрели некоторые из других протоколов, которые были популярны в то время (в первую очередь IAX2 и H.323), но для этого издания больше нет реальной причины обсуждать что-либо кроме SIP. Драйверы каналов для этих старых протоколов по-прежнему доступны в Asterisk, но они больше не поддерживаются.

Протокол SIP является одноранговым и на самом деле не имеет формальной спецификации транка. Это означает, что независимо от того, подключаете ли вы один телефон к серверу или соединяете два сервера вместе - SIP-соединения будут одинаковыми. По правде говоря, есть некоторые различия в том, как эти ресурсы могут быть настроены, и определенно будет разница в том, как ваш диалплан будет обрабатывать маршрутизацию по транкам.

Подключение системы Asterisk к SIP-провайдеру

Довольно часто используется один и тот же поставщик услуг ITSP для терминации и инициирования, но имейте в виду, что эти два процесса не связаны друг с другом. Если звонки, идущие в одном направлении, проходят ваше тестирование - это не означает, что звонки в другом направлении в порядке. При изменении конфигурации каждый раз проверяйте маршрутизацию как внутри, так и снаружи.

Многие компании предоставляют примеры конфигураций для Asterisk. К сожалению, эти документы обычно относятся к устаревшему драйверу `chan_sip`. Digium разработал мастер настройки PJSIP, призванный значительно упростить конфигурацию провайдера. Вы все еще можете настроить транки ITSP, используя те же методы, которые мы показывали ранее для настройки других конечных точек (создание записей в `ps_endpoint`, `ps_aors`, `ps_auths` и т.д.), но вместо того, чтобы снова повторять все это, мы рассмотрим мастер настройки, поскольку он объединяет несколько компонентов в один файл конфигурации. Мы обнаружили, что в отличии от часто меняющихся конечных точек пользователей, конечные точки провайдеров меняются редко, поэтому бывает полезно настроить провайдеров в файле конфигурации, а не в базе данных.

Перед созданием любой конфигурации важно определить, как оператор будет взаимодействовать с вашей системой. Есть две фундаментальные модели, которые мы видели:

Аутентификация на основе пароля, включая регистрацию[15]

Это характерно для небольших провайдеров, ориентированных на рынок малого бизнеса. Представляет такой же тип услуги, которую вы получите, если просто регистрируете SIP-телефон непосредственно в службе.

IP-аутентификация

Без пароля и без регистрации. Это чаще встречается у провайдеров, предоставляющих услуги оптового транкинга более крупным предприятиям и реселлерам. (Как правило, они также сопровождаются каким-то минимальным обязательством с точки зрения объема.) От вас ожидают, что у вас будут хорошие навыки работы с SIP и сетью.

Это не строгие ограничения, но они наиболее часто встречаются на практике.

Таким образом, существует два способа настройки ITSP в файле */etc/asterisk/pjsip_wizard.conf*.

Во–первых, если провайдер использует IP-аутентификацию - он будет ожидать, что вы будете отправлять трафик со статического IP-адреса (и если ваш адрес изменится, то нужно будет сообщить его, чтобы он мог перенастроить свое оборудование). Ваш файл *pjsip_wizard.conf* может выглядеть примерно так:

```
; ITSP используя IP-аутентификацию
[itsp-no-auth]
type=wizard
remote_hosts=itsp.example.com
endpoint/context=pstn-in
endpoint/allow = !all,ulaw,g722
sends_registrations=no
accepts_registrations=no
sends_auth=no
accepts_auth=no
```

Кроме того, если ваш IP-адрес часто меняется (или ваш оператор требует этот метод) - вы можете зарегистрировать свою систему у провайдера (что потребует от вас отправки учетных данных для аутентификации, чтобы доказать что это действительно Вы). Ваши звонки, как правило, также потребуется аутентифицировать:

```
[itsp-with-auth]
type=wizard
remote_hosts=itsp.example.com
endpoint/context=pstn-in
endpoint/allow = !all,ulaw,g722
sends_registration=yes
accepts_registrations=no
sends_auth=yes
accepts_auth=no
outbound_auth/username=itsp_provided_username
outbound_auth/password= itsp_provided_password
```

Обратите внимание, что имена `[itsp-no-auth]` и `[itsp-with-auth]` не имеют смысла для Asterisk. Они становятся именами каналов PJSIP, на которые вы отправляете свои звонки.

Настройка транков для терминирования. Мастер PJSIP создал определения каналов, необходимые для нашего оператора связи. Чтобы отправить вызов - нам нужно только внести незначительные изменения в раздел `[globals]` нашего файла *extensions.conf*, как показано ниже:

```
[globals]
UserA_DeskPhone=PJSIP/0000f30A0A01
UserA_SoftPhone=PJSIP/SOFTPHONE_A
UserB_DeskPhone=PJSIP/0000f30B0B02
UserB_SoftPhone=PJSIP/SOFTPHONE_B

TOLL=PJSIP/itsp-no-auth
```

15 Помните, что регистрация - это просто механизм, посредством которого конечная точка SIP сообщает серверу регистратора где она расположена. Это полезно если ваш IP-адрес изменяется, как это может быть в случае потребителей или малого бизнеса где часто используются динамические адреса.

```
LOCAL=${TOLL}
;OR
;TOLL=PJSIP/itsp-with-auth
;LOCAL=${TOLL}
```

Настройка транков для инициирования. Для входящих звонков в файле */etc/asterisk/extensions.conf*, вам понадобится контекст соответствующий контексту, указанному для канала ITSP. Предположим, что у нас есть два DID'a NANP: 4169671111 и 4167363636. Данный код необходимо поместить над контекстом `[sets]`:

```
TOLL=PJSIP/itsp-no-auth
LOCAL=${TOLL}
;OR
;TOLL=PJSIP/itsp-with-auth
;LOCAL=${TOLL}

[pstn-in]
exten => 4169671111,1,Dial(sets,100,1)
exten => 4167363636,1,Dial(sets,101,1)
[sets]
exten => 100,1,Dial(${UserA_DeskPhone})
```

В небольшой системе это довольно легко администрировать. В более крупной - целесообразно поместить DID'ы в таблицу своей базы данных и заставить диалплан искать нужную цель. Мы будем работать с базами данных далее в этой книге.[16]

В этом и заключается суть настройки связи с оператором. Она может показаться достаточно сложной, потому что есть много вариантов, но на самом деле всё довольно просто. Обычно обнаруживаются проблемы с незначительными несоответствиями конфигурации. Будьте методичны, и, пожалуйста, пожалуйста, пожалуйста, будьте параноиком по поводу безопасности!

Набор экстренных служб

В Северной Америке люди привыкли иметь возможность набрать 911 чтобы позвонить в экстренные службы. За пределами Северной Америки хорошо известны номера экстренных служб 112 и 999. Если вы сделаете свою систему Asterisk доступной для людей, то обязаны (во многих случаях регулируется) гарантировать совершение звонков в экстренные службы с любого телефона, подключенного к системе (даже с телефонов, ограниченных от совершения звонков).

Одна из важных частей информации, которую экстренная организация должна знать - это то, где произошла чрезвычайная ситуация (например, куда направить пожарные машины). В традиционном транке ТфОП эта информация уже известна оператору и впоследствии передается любому местному органу, выполняющему эти задачи (в Канаде и США они называются пунктами ответов на вопросы общественной безопасности или PSAP). С VoIP-линиями все может стать немного сложнее в силу того, что они физически не привязаны к какому-либо географическому местоположению.

Вы должны убедиться, что ваша система будет правильно обрабатывать экстренные вызовы с любого телефона, подключенного к нему, и вам необходимо сообщить что доступно вашим пользователям. Например, если вы разрешаете пользователям регистрироваться в системе с софтфонов на своих ноутбуках - что произойдет, если они находясь в гостиничном номере в другой стране наберут 911?[17]

Диалплан для обработки экстренных вызовов не должен быть сложным. На самом деле гораздо лучше держать его простым. Администраторы часто испытывают соблазн реализовать все виды причудливых функций в контекстах вызова аварийных служб своих диалпланов, но ошибка в одной

16 Для этой обработки в таблице просто потребуется поле для DID и еще три для целевого контекста, расширения и приоритета.

17 Не думайте, что это не сможет произойти. Звонки в службу спасения происходят в чрезвычайной ситуации, и небезопасно предполагать, что человек будет в рациональном, адекватном состоянии. Запись, сообщающая пользователям софтфонов, какой адрес система будет отправлять в PSAP - может указывать на то, что пожарные машины будут отправляться не туда, где они нужны. («Этот телефон зарегистрирован в нашей системе Торонто. Экстренные службы будут отправлены в наш офис по адресу 301 Front St W. Нажмите 1, чтобы продолжить этот вызов».).

из этих функций может вызвать сбой экстренного вызова и стоить кому-то жизни. *Здесь не место для игр*. Раздел `[emergency-services]` вашего диалплана может выглядеть примерно так:

```
[emergency-services]
exten => 911,1,Goto(dialpsap,1)
exten => 9911,1,Goto(dialpsap,1) ; некоторые люди будут набирать '9'
                                 ; потому что они привыкли делать это с АТС
exten => 999,1,Goto(dialpsap,1)
exten => 112,1,Goto(dialpsap,1)

exten => dialpsap,1,Verbose(1,Call initiated to PSAP!)
    same => n,Dial(${LOCAL}/911) ; ЗАМЕНИТЕ 911 ЗДЕСЬ НА ЧТО УГОДНО
                                 ; ПОДХОДЯЩЕЕ ДЛЯ ВАШЕГО РАЙОНА

[internal]
include => emergency-services ; эта запись должна быть в любом контексте,
                              ; который используется людьми
```

В контекстах с пользователями, работающими удаленно, лучше использовать следующий код:

```
[no-emergency-services]
exten => 911,1,Goto(nopsap,1)
exten => 9911,1,Goto(nopsap,1) ; для людей, которые набирают '9' перед внешними вызовами
exten => 999,1,Goto(nopsap,1)
exten => 112,1,Goto(nopsap,1)

exten => nopsap,1,Verbose(1,Call initiated to PSAP!)
    same => n,Playback(no-emerg-service) ; вам нужно будет записать эту подсказку

[remote-users]
include => no-emergency-services
```

В Северной Америке правила обязали многих VoIP-операторов предлагать то, что в народе известно как *E911*.[18] Когда вы регистрируетесь в их сервисах - им потребуется адресная информация для каждого из DID, которые вы хотите связать с исходящими вызовами. Эта адресная информация будет затем отправлена в PSAP, соответствующий этому адресу, и ваши экстренные вызовы должны обрабатываться так же, как если бы они были набраны по традиционной схеме ТфОП.

Суть в том, что вы должны убедиться, что телефонная система, создаваемая Вами, обрабатывает экстренные вызовы в соответствии с местными правилами и ожиданиями пользователей.

Вывод

Обычно прогнозируется, что ТфОП в конечном итоге полностью исчезнет. Однако, прежде чем это произойдет, потребуется широко используемый и надежный распределенный механизм, позволяющий организациям и отдельным лицам публиковать адресную информацию, чтобы их можно было найти. Любая голосовая технология, не использующая PSTN в настоящее время, является либо запатентованным продуктом с закрытой экосистемой, либо игровой площадкой спамеров и преступников. Мы подозреваем что ТфОП будет существовать еще некоторое время, и поэтому инициирование и терминация должны быть частью вашей системы Asterisk.

18 На самом деле, это не оператор предлагает эту услугу; скорее это возможность PSAP. E911 также используется и на транках ТфОП, но поскольку это происходит без какого-либо участия с вашей стороны (операторы ТфОП предоставляют эту информацию за вас), вы, как правило, даже не знаете, что на ваших местных линиях есть E911.

Глава 8. Голосовая почта

Просто оставьте сообщение, может быть, я перезвоню.
– Джо Уолш

До того как почта и мгновенные сообщения стали широко распространены - голосовая почта была очень популярна. Даже теперь, когда большинство людей предпочитает обмениваться текстовыми сообщениями, голосовая почта является важным компонентом любой телефонной станции.

В Asterisk есть достаточно гибкая система голосовой почты называемая Comedian Mail.[1] В диалплане она реализуется посредством модуля `app_voicemail.so`.

Предупреждение о модуле голосовой почты в Asterisk

Модуль `app_voicemail` - один из старейших в Asterisk, как следствие он имеет много ограничений, особенно если сравнивать его с другими - постоянно совершенствуемыми модулями. Код модуля настолько устарел, что ни у кого не возникает желания с ним разбираться, и поэтому очень маловероятно появление в нем новых функций. Вы должны понимать что `app_voicemail` не просто предоставляет элемент диалплана; для работы голосовой почты должно произойти много событий, например хранение и управление файлами, взаимодействие с почтовой системой операционной системы, распознавание часовых поясов, работа с форматами файлов, вопросы безопасности и еще куча вещей. И хотя `app_voicemail` все это делает, в итоге получается довольно неуклюжая подсистема (стоит заметить, что в традиционных АТС для голосовой почты выделяется отдельная машина).

Было предпринято множество попыток реинжинировать голосовую почту, но все они оказались неудачными. Причина проста: объем работ (а следовательно и стоимость), необходимых для перепроектирования модуля (таким образом, чтобы удовлетворить потребности сообщества), в сочетании с отсутствием интереса к технологии голосовой почты в целом, быстро убивали любую инициативу.

Тем не менее важно отметить, что голосовая почта в Asterisk работает, и работает хорошо. Возможно, она даже удовлетворит ваши потребности. В ином случае сообщество будет более чем благодарно, если вы попытаетесь перепроектировать её.

Вот некоторые функции, которые включает в себя модуль голосовой почты:

- Неограниченное количество защищенных паролем ящиков голосовой почты, каждый из которых содержит подкаталоги для сортировки голосовой почты
- Различные приветствия для различных статусов, таких как "недоступен" или "занят"
- Наличие предустановленных приветствий и возможность создания собственных
- Возможность ассоциирования телефонов с несколькими почтовыми ящиками и почтового ящика с несколькими телефонами
- Уведомление о голосовом сообщении на электронную почту, опционально с прикрепленным аудио файлом
- Широковещательная голосовая почта и перенаправление голосовой почты

1 Это название было игрой слов, отчасти вдохновленной системой голосовой почты Nortel Meridian Mail. Nortel (и Meridian Mail) ушли,но Comedian Mail продолжает работать.

- Индикатор ожидания сообщения (мигающий светодиод или специальный сигнал) поддерживаемый на многих типах телефонов
- Справочник сотрудников на основе голосовой почты

Теперь давайте познакомимся с основными частями конфигурационного файла голосовой почты, включая настройки в глобальном разделе, различными возможными региональными настройками, интеграцией голосовой почты в ваш диалплан и проведем краткий обзор того, как Asterisk хранит голосовую почту в файловой системе Linux.

Файл конфигурации voicemail.conf

Ранее в базе данных MySQL мы установили таблицу, необходимую для голосовой почты, и поэтому сейчас мы можем создавать в ней почтовые ящики без какой-либо другой конфигурации. Однако, также можно создавать почтовые ящики в конфигурационном файле */etc/asterisk/voicemail.conf* (в том числе в этом файле можно изменять различные настройки по умолчанию). Мы продолжим использовать базу данных для создания пользователей и управления ими - поскольку она гораздо лучше подходит для этой задачи, но также исследуем конфигурационный файл чтобы вы могли почувствовать гибкость настройки голосовой почты Asterisk.

Файл *voicemail.conf* содержит несколько секций, в которых могут быть переопределены различные предустановленные параметры. В большинстве случаев вам не понадобится их менять; однако вы можете посмотреть в файле примера *~/src/asterisk-16.<ваша_версия>/configs/samples/voicemail.conf.sample*. Он содержит полезную информацию о различных настройках.

Далее мы рассмотрим простейший файл *voicemail.conf*. Если у вас возникнет желание доработать базовую конфигурацию - просто добавьте или измените соответствующие опции.

Исходный файл voicemail.conf

Мы рекомендуем использовать следующий пример кофигурации как базовый. Вы можете ознакомиться с файлом *~/src/asterisk-16.<ваша_версия>/configs/samples/voicemail.conf.sample* для детализации различных настроек.

Разместите следующий код в файле */etc/asterisk/voicemail.conf*:

```
; Конфигурация Voicemail

[general]
format=wav49|wav
serveremail=voicemail@shifteight.org
attach=yes
skipms=3000
maxsilence=10
silencethreshold=128
maxlogins=3
emaildateformat=%A, %B %d, %Y at %r
pagerdateformat=%A, %B %d, %Y at %r
sendvoicemail=yes ; Разрешить пользователю создавать и отправлять голосовую почту изнутри

[zonemessages]
eastern=America/New_York|'vm-received' Q 'digits/at' IMp
central=America/Chicago|'vm-received' Q 'digits/at' IMp
central24=America/Chicago|'vm-received' q 'digits/at' H N 'hours'
military=Zulu|'vm-received' q 'digits/at' H N 'hours' 'phonetic/z_p'
european=Europe/Copenhagen|'vm-received' a d b 'digits/at' HM
```

Настройка Linux-сервера для отправки почтовых сообщений администратору выходит за рамки данной книги. Вы должны будете протестировать вашу службу голосовой почты чтобы убедиться что она корректно обрабатывается почтовым

агентом[2] и что нижеследующие спам-фильтры не отклоняют эти сообщения (одна из причин почему это может происходить — использование сервером Asterisk в теле письма имени хоста которое не может быть разрешено)

Вы можете создать массивный и сложный файл `voicemail.conf` (и даже хранить в нем почтовые ящики пользователей), но для упрощения задачи мы сосредоточимся на нескольких примерах.

Секция [general]

В первой секции файла *voicemail.conf* - `[general]` определяются глобальные настройки. Многие из этих настроек могут быть переопределены в настройках каждого конкретного ящика. В Таблице 8-1 мы перечислили некоторые опции, которые, как мы считаем, наиболее важно рассмотреть.

Таблица 8-1. Настройки секции [general] файла voicemail.conf

Опция	Значение/пример	Примечание
`format`	`wav49|gsm|wav`	Для каждого перечисленного формата Asterisk создает отдельную запись в этом формате, каждый раз, когда остается сообщение. Преимущество этого механизма в экономии ресурсов на транскодировании, которое не надо выполнять если для записи используется тот же самый кодек что и для канала. Мы любим WAV за высокое качество, и WAV49 потому что он хорошо сжимается и легок для передачи по почте. Мы не любим GSM за шумы в записи, но он пользуется некторой популярностью.[a]
`serveremail`	`user@domain`	Адрес электронной почты, с которого отправляется письмо от Asterisk.[b]
`attach`	`yes,no`	Если для ящика голосовой почты указан адрес электронной почты - эта опция определяет, будет ли прикреплено записанное сообщение к письму (если нет, то будет отправлено простое уведомление и пользователю нужно будет позвонить на голосовую почту чтобы получить свое сообщение).
`maxmsg`	`9999`	По умолчанию, Asterisk разрешает хранить максимум 100 сообщений на пользователя. Для пользователей, удаляющих прослушанные сообщения, это не является проблемой. Для пользователей, предпочитающих сохранять свои сообщения, этот лимит будет достигнут очень быстро. С размерами жестких дисков в наши дни - вы можете легко хранить тысячи сообщений для каждого пользователя. Поэтому, по нашему мнению, можно выставить эту опцию в максимальное значение и позволить пользователям самим управлять этими данными. Имейте в виду, что после нескольких лет хранения - старые сообщения голосовой почты в больших системах могут занимать много места на жестком диске.
`maxsecs`	`600`	Эта установка может быть полезной, когда большая система голосовой почты имеет хранилище размером в 40МБ[c]: в данном случае необходимо ограничение длины сообщения, иначе система легко использует весь объем хранилища. Этот параметр может раздражать вызывающих абонентов (хотя он заставляет их переходить к сути сообщения, поэтому некоторым людям это нравится). В настоящее время с терабайтными дисками - нет никаких технических причин для ограничения длины сообщения. Но есть два соображения: 1) если канал завис, то хорошо бы иметь какое-либо ограничение чтобы система не записывала бесконечное пустое сообщение; 2) если абонент использует свой почтовый ящик как голосовую записную книжку - он не обрадуется если вы его отключите через три минуты. Вероятно, будет правильным установить значение где-то между 600 секундами (10 минут) и 3600 секундами (1 час).
`emailsubject`	`[PBX]: New message ${VM_MSGNUM} in mailbox ${VM_MAILBOX}`	Этой настройкой вы можете задать вид темы письма, отсылаемое Asterisk. Подробное описание смотрите в файле примера *voicemail.conf.sample*.
`emailbod`	`Dear ${VM_NAME}:\`	Этой настройкой вы определяете как будет выглядеть тело письма. Подробное

2 Также иногда называют агентом передачи сообщений (Message Transfer Agent).

Опция	Значение/пример	Примечание
y	n\n\t you have a ${VM_DUR} long message (number ${VM_MSGNUM})\n in mailbox ${VM_MAILBOX} \n\n\t\t\t\t -- Asterisk\n	описание смотрите в файле примера *voicemail.conf.sample*.
emaildateformat	%A, %d %B %Y at %H:%M:%S	Эта опция позволяет определить формат даты в письме. Используется тот же синтаксис, что и в функции STRFTIME языка C.
pollmailboxes	no, yes	Если содержимое почтового ящика меняется чем-нибудь кроме app_voicemail (Например внешним приложением или другой системой Asterisk) - эта настройка выставленная в "yes" указывает периодически опрашивать почтовые ящики на предмет изменений и выставляет правильную индикацию ожидающих сообщений (MWI).
pollfreq	30	Используется в сочетании с параметром pollmailboxes и в секундах задает частоту опроса почтового ящика.

[a] Разделителем для каждого формата служит знак вертикальной черты - pipe (|).
[b] Рассылка писем от Asterisk требует аккуратной настройки, так как многие спам-фильтры находят сообщения от Asterisk очень подозрительными и просто блокируют их.
[c] Да, вы все верно прочитали: мегабайт.

Проверка паролей голосовой почты внешними средствами

По умолчанию, Asterisk не проверяет пароли пользователей на устойчивость к взлому. Любой кто разрабатывает системы голосовой почты скажет вам, что подавляющее число пользователей устанавливает такие пароли к своим ящикам, которые легко запомнить, например 1234 или 1111. Хотя обычно фрод-боты не предназначены для баловства - наличие слабых паролей представляет собой дыру в системе безопасности голосовой почты.

Поскольку модуль app_voicemail.so не имеет встроенной возможности проверки паролей - настройки externpass, externpassnotify и externpasscheck позволяют проверять их с помощью внешней программы. Asterisk вызовет приложение, находящееся по указанному вами пути и передаст следующие аргументы:

```
mailbox context oldpass newpass
```

Затем скрипт будет оценивать аргументы основываясь на правилах, которые вы в нем определили, и, соответственно, он должен вернуть в Asterisk значение VALID в случае успеха или INVALID в случае неудачи (на самом деле возвращаемое значение для пароля не прошедшего проверку может быть любое, кроме слов VALID и FAILURE). Это значение выводится в stdout - на стандартный вывод. Если сценарий вернул значение INVALID - Asterisk будет воспроизводить запись ошибочного пароля и пользователю будет нужно попробовать набрать что-то иное.

Возможно вам стоит реализовать следующие правила:

- Минимальная длина пароля должна составлять 6 символов
- Пароль не должен представлять собой строку повторяющихся цифр (т.к. 111111)
- Пароль не должен представлять собой последовательность цифр (т.к. 123456 или 456789)

Asterisk поставляется с простейшим сценарием, значительно усовершенствующим безопасность вашей системы голосовой почты. Он расположен в каталоге исходного кода: */contrib/scripts/voicemailpwcheck.py*.

Мы настоятельно рекомендуем вам скопировать его в ваш каталог */usr/local/bin* (или туда где вы держите подобные вещи) и затем раскомментировать опцию externpasscheck= в вашем файле

voicemail.conf.

Часть раздела `[general]` - это раздел дополнительных опций (Они упоминаются в конфигурационном файле как *расширенные опции*, хотя ничего особенного в них нет). Они (перечисленные в Таблице 8-2) определены также как и другие в секции `[general]`, но выделяет их то, что они могут быть переопределены для каждого отдельного почтового ящика, что перезапишет свойства установленные в этой области секции `[general]`. *Другими словами: нижеследующие опции могут быть установлены в базе данных когда вы создаете новый почтовый ящик.*

Таблица 8-2. Утвержденный список дополнительных опций для voicemail.conf

Опция	Значение/ пример	Примечание
`tz`	`eastern, european, etc.`	Указывает имя часового пояса также определенного в разделе `[zonemessages]` (мы будем говорить об этом в следующей части главы).
`locale`	`de_DE.utf8, es_US.utf8, etc.`	Используется для определения того, как Asterisk задает формат строки с данными даты и времени в различных локалях. Для определения локали, корректной для вашей Linux-системы, выполните в консоли операционной системы команду **`locale -a`**.
`attach`	`yes, no`	Если для вашего голосового почтового ящика определен адрес электронной почты - эта опция определяет, будут ли сообщения прикреплены к почтовым уведомлениям (В ином случае будет отправлено простое уведомление).
`attachfmt`	`wav49, wav, etc.`	Если опция `attach` включена и сообщения хранятся в различных форматах - данная опция определяет формат, в котором будут отправлены записанные сообщения в почтовых уведомлениях. Обычно `wav49` является хорошим выбором, так как использует лучший алгоритм сжатия, а следовательно, использует меньшую полосу пропускания, и в тоже время не так жутко звучит как формат `gsm`.
`exitcontext`	*`context`*	Эта опция позволяет звонящим абонентам выйти из системы голосовой почты, когда они находятся в процессе записи сообщений (для примера, нажатие на 0 переключает на оператора). По умолчанию, контекст с которого пришел вызов, будет использоваться для выхода. По желанию этот параметр может быть определен в иной контекст.
`review`	`yes, no`	Этот параметр почти всегда должен быть установлен в `yes` (Хотя его значение по умолчанию `no`). Люди расстраиваются, если ваша система голосовой почты не позволяет им прослушать свое сообщение перед отправкой.
`operator`	`yes, no`	В соответствии с лучшими практиками вы должны разрешить своим абонентам в любой момент прекратить запись, если они передумали оставлять голосовое сообщение. Обратите внимание, что для обработки этих вызовов контексте `exitcontext` требуется расширение `o` (не «ноль», а буква «о»).
`delete`	`no, yes`	После того как почтовое уведомление (которое может включать само голосовое сообщение) будет отправлено - оно будет удалено. Эта опция рискованна, так как факт отправки сообщения не гарантирует его получения (Спам фильтры любят удалять сообщения голосовой почты Asterisk). На новых системах оставьте эту опцию в значении `no` до тех пор, пока не убедитесь что сообщения не теряются из-за спам-фильтров.
`nextaftercmd`	`yes, no`	Эта удобная маленькая опция сэкономит вам немного времени, так как переключает вас на следующее сообщение сразу после завершения работы с предыдущим.
`passwordlocation`	`spooldir`	Если вы захотите, то можете хранить пароли в собственных spool каталогах каждого почтового ящика.[a] Одно из преимуществ использования опции `spooldir` в том, что она позволяет вам определять операторы `#include` в файле *voicemail.conf* (Имеется в виду что вы можете хранить ссылки на почтовые ящики в нескольких файлах, как, например, с кодом диалплана). По другому так сделать

Опция	Значение/ пример	Примечание
		невозможно, потому что в обычном случае `app_voicemail` записывает изменение пароля в файловую систему и не может обновлять пароль почтового ящика, хранящийся за пределами *voicemail.conf* или spool. Если вы не используете опцию `passwordlocation` - вы не сможете определять почтовые ящики за пределами *voicemail.conf* так как пароль не будет обновлен. Хранение паролей в файле в специальном spool каталоге решает эту проблему.

[a] Обычно каталог spool находится по пути */var/spool/asterisk* и может быть переопределен в */etc/asterisk/asterisk.conf*.

Секция [zonemessages]

Следующим в файле *voicemail.conf* идет раздел `[zonemessages]`. Его целью является разрешить обработку сообщений в соответствии с часовым поясом, таким образом, вы можете воспроизводить сообщения пользователей с правильными отметками времени. Вы можете установить имя пояса в нужное вам значение. Затем вы можете определить на какой часовой пояс будет ссылаться это имя пояса, а так же некоторые параметры, определяющие как воспроизводятся временные метки. Примеры синтаксиса вы можете посмотреть в файле *~/src/asterisk-16.*/configs/samples/voicemail.conf.sample*. Asterisk включает примеры показанные в Таблице 8-3. Можно настроить любой часовой пояс, известный Linux-системе. Просто используйте для поясов имена как в Linux и затем определите как вы хотите чтобы они обрабатывались.

Таблица 8-3. Настройки секции [zonemessages] для voicemail.conf

Имя пояса	Значение/пример	Примечание
`eastern`	`America/New_York|'vm-received' Q 'digits/at' IMp`	Это значение подойдет для восточных часовых поясов (EST/EDT).
`central`	`America/Chicago|'vm-received' Q 'digits/at' IMp`	Это значение подойдет для центрального часового пояса (CST/CDT).
`central24`	`America/Chicago|'vm-received' q 'digits/at' H N 'hours'`	Это значение так же подойдет для CST/CDT, но будет отображать время в 24 часовом формате.
`military`	`Zulu|'vm-received' q 'digits/at' H N 'hours' 'phonetic/z_p'`	Это значение подойдет для UTC - Всемирное координированное время (время Zulu, formerly GMT).
`european`	`Europe/Copenhagen|'vm-received' a d b 'digits/at' HM`	Это значение подойдет для Центральной Европы (CEST).

Почтовые ящики

Вы можете настраивать почтовые ящики в конфигурационном файле *voicemail.conf*, но это не рекомендуемый путь. Для определения ваших ящиков мы будем использовать базу данных. Первая вещь которую нам надо сделать - это сообщить Asterisk, что голосовая почта пользователей доступна в базе данных. Это можно сделать отредактировав файл */etc/asterisk/extconfig.conf*:

```
$ sudo vim /etc/asterisk/extconfig.conf

[settings] ; older mechanism for connecting all other modules to the database
ps_endpoints => odbc,asterisk
ps_auths => odbc,asterisk
ps_aors => odbc,asterisk
ps_domain_aliases => odbc,asterisk
ps_endpoint_id_ips => odbc,asterisk
ps_contacts => odbc,asterisk
voicemail => odbc,asterisk,voicemail
```

Вы должны перезапустить Asterisk для применения этих изменений.

```
$ sudo service asterisk restart
```

В системе голосовой почты почтовый ящик должен быть определен в контексте. Это не имеет отношения к какому-нибудь контексту диалплана; Этот контекст является специфичной меткой голосовой почты, определяющей какие почтовые ящики будут сгруппированы вместе, а также используется для именования директории в спуле, содержащей различные файлы, связанные с этим почтовым ящиком (приветствия, сообщения, конверты и т.д.). Обычно вам не стоит волноваться об этом, так как все почтовые ящики окажутся в контексте `default`. На самом деле вам нужно определить только контексты, настройки которых отличаются от остальных - если вы используете сложную, многопользовательскую систему, в которой возможно перекрытие внутренних номеров, или если не хотите, чтобы определенные группы пользователей были доступны другим группам пользователей.

Таблица `asterisk.voicemail` поддерживает много опций; однако, для создания почтового ящика необходимо заполнить только три поля, плюс еще два рекомендовано. Требуются поля `context`, `mailbox` и `password`, а `fullname` и `email` являются строго рекомендованными. Вот простой MySQL запрос позволяющий вам создать несколько почтовых ящиков:

```
INSERT INTO `asterisk`.`voicemail` (context,mailbox,password,fullname,email)
VALUES
('default','100','486541','Russell Bryant', 'russell@shifteight.org'),
('default','101','957642','Leif Madsen', 'leif@shifteight.org'),
('default','102','656844','Jared Smith', 'jared@shifteight.org'),
('default','103','375416','Jim VanMeggelen', 'jim@shifteight.org')
;
```

Ниже части, определяющие почтовый ящик:

`mailbox`

Это номер почтового ящика. Обычно он соответствует добавочному номеру абонента.

`Password`

Это числовой пароль с помощью которого владелец почтового ящика сможет получить доступ к своей голосовой почте. Если пользователь меняет пароль, система обновит это поле в базе данных.

Если перед паролем стоит знак дефиса (-), пользователь не сможет изменить свой пароль почтового ящика.

`fullname (FirstName LastName)`

Это имя владельца почтового ящика. Каталог компании использует текст в этом поле для проверки имен пользователей. Вы можете использовать только один пробел - чтобы отделить имя от фамилии, поэтому если ваша фамилия Ван Меггелен, вы должны записать ее как ВанМеггелен. Другие знаки пунктуации также могут вызвать проблемы. (Мы смотрим на тебя, О'Рейли.)

`email address`

Это почтовый адрес владельца ящика. Asterisk может выслать запись голосовой почты на определенный адрес электронной почты.

Asterisk не может обрабатывать концепцию фамилии, отличающуюся от простого слова. Это значит, что перед добавлением в *voicemail.conf*, во всех таких фамилиях как О'Рейли, Брайан-Мэдсен-Смитт, и, да, даже Ван Меггелен, должны быть удалены все пунктуационные символы и пробелы.

Есть довольно много других опций, которые вы можете определить для каждого пользователя. Маловероятно что вы будете использовать их все, но в Таблице 8-4 содержится список тех из них которые могут быть вам полезны:

Таблица 8-4. Параметры почтового ящика

Опция	Описание
delete	Asterisk удалит с сервера отправленное по электронной почте голосовое сообщение. Эта опция полезна для пользователей, желающих получать голосовую почту только по почте. Допустимые значения yes или no. *Параметр устанавливается на каждый отдельный ящик.*
envelope	Включает или выключает проигрывание даты и времени звонка перед воспроизведением голосового сообщения. Допустимые значения yes или no. По умолчанию стоит значение yes.
exitcontext	Контекст диалплана, в который будет произведен выход из приложения Voicemail(), когда нажата клавиша * или 0. Хорошо работает в сочетании с опцией operator. Для выхода по *, в контексте необходимо иметь расширение a, или расширение o для выхода по 0. Вам нужно будет поработать над своим диалпланом чтобы включить эти функции - поэтому лучше оставить эту опцию пока пустой, до того момента когда будете уверены что сделали все, что хотели.
forcegreeting	Принудительно записывает приветствие для новых почтовых ящиков. Новый почтовый ящик определяется его номером и паролем. Допустимые значения yes или no. По умолчанию стоит значение no.
forcename	Принудительно записывает имя человека для новых почтовых ящиков. Новый почтовый ящик определяется его номером и паролем. Допустимые значения yes или no. По умолчанию стоит значение no.
hidefromdir	Если установлено значение yes - этот почтовый ящик будет скрыт от приложения Directory(). По умолчанию no.
locale	Позволяет установить языковой стандарт для почтового ящика, чтобы контролировать форматирование строк даты/времени. Смотри *voicemail.sample.conf* для получения дополнительной информации.
messagewrap	Позволяет зацикливать очередь сообщений (то есть позволяет переходить к первому сообщению после последнего или к последнему, если мы хотим прослушать предыдущее первому сообщению). Допустимые значения yes или no. По умолчанию стоит значение no.
minpassword	Устанавливает минимальную длину пароля. Аргумент должен быть задан числом.
nextaftercmd	Переходит к следующему сообщению после нажатия пользователем клавиши 7 (delete) или 9 клавиши (save). Допустимые значения yes или no. По умолчанию установлено yes.
operator	Позволяет отправителю голосовой почты набрать 0 до, во время, или после записи голосовой почты. Завершит работу с расширением o в том же контексте или в контексте, заданном параметром exitcontext. Допустимые значения yes или no. По умолчанию стоит значение no. С этим связаны риски безопасности - поэтому лучше не использовать его до тех пор, пока вы не убедитесь, что exitcontext не разрешает вызовам выходить из системы (т. е. в итоге вы совершаете дорогой зарубежный вызов).
passwordlocation	По умолчанию, пароль для голосовой почты хранится в конфигурационном файле *voicemail.conf* и изменяется Asterisk всякий раз, когда изменяется пароль. Это может быть нежелательно, особено если вы хотите распарсить пароль из внешнего местоположения (или сценария). Альтернативной опцией для установки пароля является spooldir, который помещает пароль для пользователя голосовой почты в файл с именем *secret.conf* в каталоге спулинга голосовой почты пользователя. Допустимые параметры: voicemail.conf и spooldir. Опция по умолчанию - voicemail.conf.
review	Если этот параметр включен - пользователь, записывающий сообщение голосовой почты, сможет перезаписать свое сообщение. После нажатия клавиши # для сохранения голосовой почты ему будет предложено перезаписать или сохранить сообщение. Допустимые значения yes или no. По умолчанию стоит значение no.
saycid	Если этот параметр включен и в */var/spool/asterisk/voicemail/recordings/callerids* имеется приглашение - оно будет воспроизводиться перед сообщением, вместо произнесения цифр CallerID. Допустимые значения yes или no. По умолчанию стоит значение no.
sayduration	Озвучивает длительность записи перед сообщением. Допустимые значения yes или no. По умолчанию стоит значение yes.
saydurationm	Позволяет установить минимальную продолжительность воспроизведения (в минутах).

Опция	Описание
	Например, если вы установите значение 2 - вы не будете проинформированы о длине сообщения длиной менее 2 минут. Допустимые значения - целые числа. По умолчанию 2.
searchcontexts	Для таких приложений, как Voicemail(), VoicemailMain() и Directory() - контекст голосовой почты является необязательным аргументом. Если контекст голосовой почты не указан, то по умолчанию выполняется поиск в контексте default. Если эта опция включена, то поиск будет происходить во всех контекстах. Предупреждаем, что, если параметр включен - номер почтового ящика должен быть уникальным во всех контекстах, иначе произойдет коллизия, и система не поймет, какой почтовый ящик использовать. Допустимые значения yes или no. По умолчанию стоит значение no.
sendvoicemail	Позволяет пользователю создавать и отправлять сообщения голосовой почты из приложения VoicemailMain(). Доступно как опция 5 в расширенном меню. Если эта опция отключена, то опция 5 в расширенном меню не будет предложена. Допустимые значения yes или no. По умолчанию стоит значение no.
tempgreetwarn	Включает уведомление для пользователя, когда его временное приветствие включено. Допустимые значения yes или no. По умолчанию стоит значение no.
tz	Устанавливает часовой пояс для пользователя голосовой почты (или глобально). Смотрите */usr/share/timezone* для различных доступных часовых поясов. Не применимо, если envelope=no.
volgain	Опция volgain позволяет вам установить уровень громкости для сообщений голосовой почты. Значение указывается в децибелах (дБ). Для этого должно быть установлено приложение *sox*.

Интеграция диалплана голосовой почты

Существует два основных приложения диалплана, предоставляемых модулем app_voicemail.so в Asterisk. Первое, называемое просто VoiceMail(), делает именно то, что вы ожидаете - то есть записывает сообщение в почтовый ящик. Второе - VoiceMailMain() позволяет пользователю входить в почтовый ящик для получения сообщений.

Приложение диалплана VoiceMail()

Если вы хотите передать вызов на голосовую почту - вам нужно предоставить два аргумента: почтовый ящик (или почтовые ящики), в котором должно быть оставлено сообщение и любые относящиеся к этому параметры, например, какое приветствие следует воспроизвести или отметка срочности сообщения. Структура команды VoiceMail() такова:

```
VoiceMail(mailbox[@context][&mailbox[@context][&...]][,options])
```

Параметры, которые вы можете передать VoiceMail(), обеспечивающие более высокий уровень контроля, подробно описаны в Таблице 8-5.

Таблица 8-5. Необязательные аргументы VoiceMail()

Аргумент	Цель
b	Предписывает Asterisk воспроизводить приветствие "занято" для почтового ящика (если приветствие "занято" отсутствует - будет воспроизводиться приветствие "недоступно").
d([c])	Принимает цифры, подлежащие обработке в контексте c. Если контекст не указан - по умолчанию будет использоваться текущий контекст.
g(#)	Применяет заданную величину усиления (в децибелах) к записи. Работает только на каналах DAHDI.
s	Подавляет воспроизведение инструкций вызывающим абонентам после воспроизведения приветствия.
u	Поручает Asterisk воспроизводить уведомление о недоступном почтовом ящике (это поведение по умолчанию).
U	Указывает что сообщение должно быть отмечено как срочное. Этот эффект наиболее заметен, когда голосовая почта хранится на IMAP сервере, в этом случае электронное письмо будет также отмечено как

Аргумент	Цель
	срочное. Когда владелец почтового ящика звонит в систему голосовой почты Asterisk - он также будет проинформирован, что запись является срочной.
P	Указывает, что сообщение должно быть помечено как приоритетное.

Приложение `VoiceMail()` перенаправляет абонентов в указанный почтовый ящик, чтобы они могли оставить сообщение. Почтовый ящик должен быть указан как *`mailbox@context`*, где *`context`* - это имя контекста голосовой почты (не контекст диалплана). Буквы опций `b` или `u` могут быть добавлены для запроса типа приветствия. Если используется буква `b` - вызывающий абонент услышит сообщение о *занятости* владельца почтового ящика (если оно существует). Если используется буква `u` - вызывающий абонент услышит сообщение о *недоступности* владельца почтового ящика (также при условии что оно существует). *Если приветствия не существует - система сгенерирует общее сообщение: Абонент с почтовым ящиком <mailbox> недоступен. Пожалуйста, оставьте сообщение после сигнала.*

В диалплане, который мы создали в Главе 6, было создано несколько расширений. Рассмотрим пример добавочного номера 102, который позволяет вызывать абонентов `UserB_DeskPhone`:

```
exten => 102,1,Dial(${UserB_DeskPhone},10)
  same => n,Playback(vm-nobodyavail)
  same => n,Hangup()
```

В примере выше, вместо почтового ящика, мы просто проигрываем приветствие и больше ничего. Давайте поменяем диалплан чтобы звонок приходил на действительный почтовый ящик. Для начала мы просто дадим голосовой почте воспроизвести общее приветствие, которое услышит звонящий. Помните, что вторым аргументом приложения `Dial()` является время ожидания. Если в течении времени ожидания на вызов не ответили - он отправляется на следующий приоритет. У нас есть 10-секундный интервал времени ожидания и новый приоритет для отправки звонящего на голосовую почту после ожидания ответа:

```
exten => 102,1,Dial(${UserB_DeskPhone},10)
  same => n,Voicemail(${EXTEN}@default,u))
  same => n,Hangup()
```

Если захотим - мы можем сделать больше и изменить код так, что если пользователь занят (разговаривает по другой линии), вызывающий абонент услышит сообщение "занято". Для этого мы будем использовать переменную `${DIALSTATUS}`, содержащую одно из нескольких значений состояния (наберите **`core show application Dial`** в консоли Asterisk для получения списка всех возможных значений):

```
exten => 102,1,Dial(${UserA_SoftPhone})
  same => n,GotoIf($["${DIALSTATUS}" = "BUSY"]?busy:unavail)

  same => n(unavail),VoiceMail(101@default,u)
  same => n,Hangup()

  same => n(busy),VoiceMail(101@default,b)
  same => n,Hangup()
```

Теперь звонящие получат ответ голосовой почты (с соответствующим приветствием), если пользователь занят или недоступен. Альтернативный синтаксис для определения нужного сообщения ("недоступен" или "занят") заключается в использовании функции `IF()`:[3]

```
exten => 103,1,Dial(${UserB_SoftPhone})
  same => n,Voicemail(${EXTEN}@default,${IF($["${DIALSTATUS}" = "BUSY"]?b:u)})
  same => n,Hangup()
```

Небольшая проблема заключается в том, что наши пользователи не могут ни получать свои сообщения, ни настраивать свои приветствия, ни какие-либо другие параметры голосовой почты. Мы исправим это в следующей секции.

3 Мы углубимся в такие функции как `IF()` в Главе 10.

Приложение диалплана VoiceMailMain()

Пользователи могут получать сообщения голосовой почты, изменять параметры и записывать приветствия с помощью приложения `VoiceMailMain()`. `VoiceMailMain()` принимает два аргумента: номер почтового ящика (и, если необходимо, контекст), а также несколько параметров. Оба аргумента являются необязательными.

Структура приложения `VoiceMailMain()` выглядит следующим образом:

```
VoiceMailMain([mailbox][@context][,options])
```

При использовании приложения `VoiceMailMain()` без аргументов - будет воспроизведено сообщение предлагающее вызывающему абоненту предоставить номер своего почтового ящика. Варианты, которые могут быть предоставлены, перечислены в Таблице 8-6

Таблица 8-6. Необязательные аргументы VoiceMailMain().

Аргумент	Цель
`p`	Позволяет обрабатывать параметр *mialbox* как префикс к номеру почтового ящика
`g(#)`	Увеличивает громкость на # децибел при воспроизведении сообщений
`s`	Пропускает проверку пароля
`a(folder)`	Запускает сессию в одной из следующих папок голосовой почты (по умолчанию `0`): 0 — `INBOX` 1 — `Old` 2 — `Work` 3 — `Family` 4 — `Friends` 5 — `Cust1` 6 — `Cust2` 7 — `Cust3` 8 — `Cust4` 9 — `Cust5`

Чтобы позволить пользователям набирать добавочный номер для проверки своей голосовой почты - вы можете добавить этот номер в диалплан следующим образом:

```
exten => *98,1,NoOp(Access voicemail retrieval.)
   same => n,VoiceMailMain()
```

Любой пользователь, чье устройство обозначено в контексте `[sets]`, теперь может набрать *98 и войти в свой почтовый ящик для прослушивания сообщений, записи своего имени, установки приветствия и т.д.

Стандартные комбинации кнопок голосовой почты

Рисунок 8-1 Показывает стандартную конфигурацию комбинаций клавиш для почты Asterisk. Некоторые параметры могут быть включены или отключены в зависимости от конфигурации *voicemail.conf* (например, `envelope=no`). Рисунок можно дать пользователям в качестве памятки.

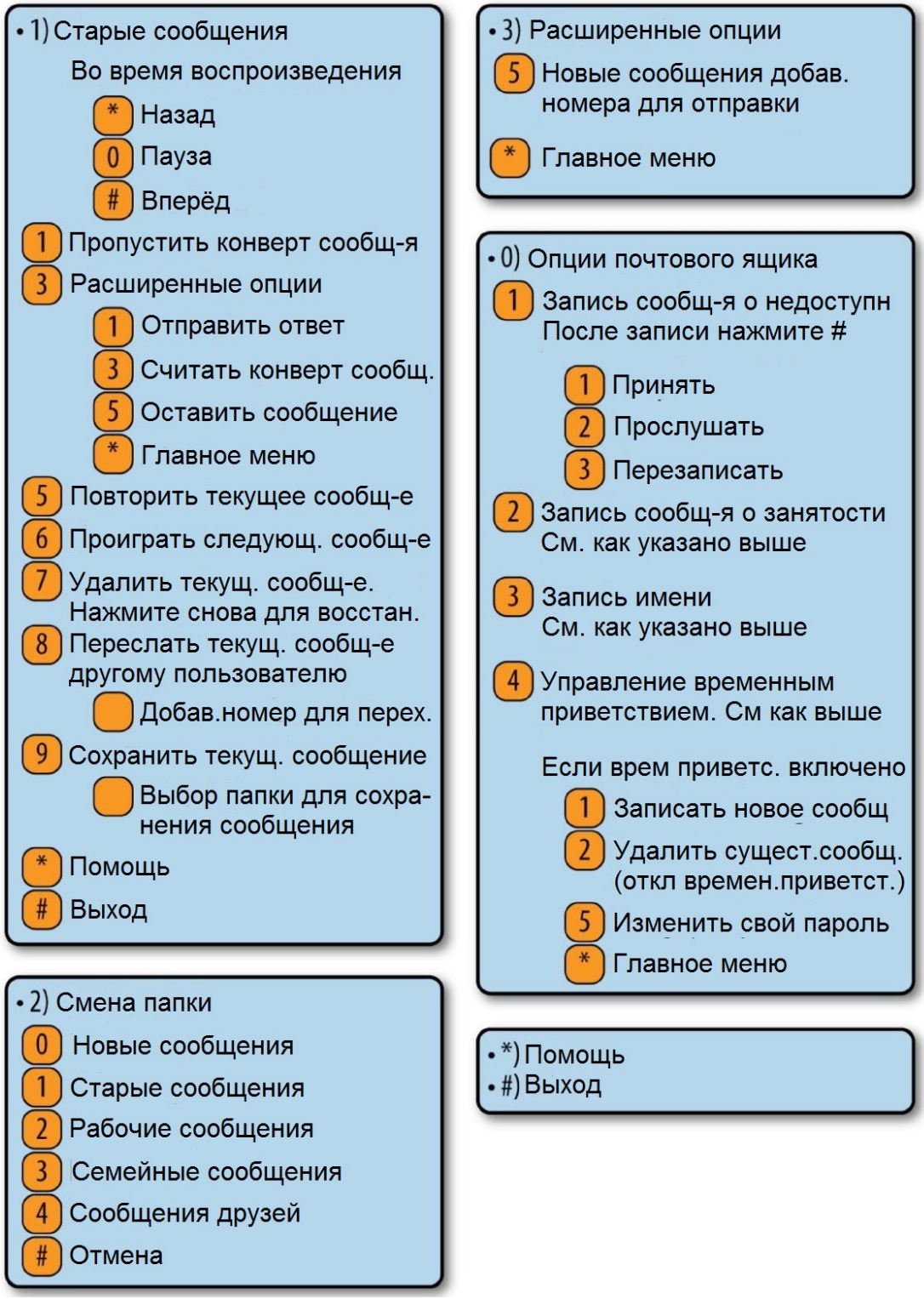

Рисунок 8-1. Конфигурация клавиш для Comedian Mail

Создание каталога набор-по-имени

Еще одной особенностью голосовой почты Asterisk, которую мы должны рассмотреть является каталог набор-по-имени. Он создается приложением `Directory()`. Это приложение использует имена, определенные в почтовых ящиках в *voicemail.conf*, чтобы предоставить вызывающему абоненту каталог для набора пользователей по имени.

`Directory()` принимает до трех аргументов: контекст голосовой почты из которого будут считываться имена, необязательный контекст диалплана, в котором будет вызываться пользователь и строка параметров (так же являющаяся необязательной). По умолчанию `Directory()` ищет

пользователя по фамилии, но установка параметра `f` заставляет вместо этого искать по имени. Давайте добавим два каталога набора-по-имени в контекст `TestMenu` нашего диалплана так, чтобы абоненты могли осуществлять поиск по имени или фамилии:

```
exten => 4,1,Dial(${UserB_SoftPhone},10)
   same => n,Playback(vm-nobodyavail)
   same => n,Hangup()

exten => 8,1,Directory(default,sets,f)
exten => 9,1,Directory(default,sets)

exten => i,1,Playback(pbx-invalid)
   same => n,Goto(TestMenu,start,1)
```

Если вы набираете 201 и затем нажимаете 8 - вы получаете каталог с поиском по имени, если вы нажали 9, то каталог с поиском по фамилии.

Голосовая почта по электронной

Когда Asterisk впервые вышла - она делала нечто очень простое, но тем не менее революционное на рынке АТС того времени. Ни один из крупных брендов АТС не смог предоставить технологию эффективной отправки голосовых сообщений на электронную почту (проще говоря просто отправку на электронную почту сообщения как вложение WAV файла). Конечно некоторые производители предлагали такую функциональность, но она была излишне сложной, ненадежной и дорогой. Asterisk прекратил это недоразумение и просто разрешил ящику голосовой почты иметь прикрепленный адрес электронной почты - таким образом сообщения отправлялись обычными механизмами отправки электронной почты Linux. Это оказалось и просто и эффективно, и действительно показало насколько устарели традиционные производители АТС.

К сожалению, в каждой хорошей истории есть плохой парень, а в этом случае из них случилась эпидемия: спамеры почти поставили интернет на колени. Больше нельзя было доверять простой передаче SMTP, так как любая машина, открытая для пересылки писем, быстро становилась бы целью для спамеров.

Таким образом электронная почта стала более сложной. Если вы хотите послать письмо из системы Asterisk, то как показано в Таблице 8-7, у вас есть три основных способа это сделать.

Таблица 8-7. Обзор методов передачи голосовой почты по e-mail

Метод	Недостатки	Достоинства
1. Отправить письмо непосредственно на SMTP порт адреса, указанного в записи MX, возвращаемой от запроса к домену. **Почти всегда обречены на провал.**	Спам-фильтры будут стремится отбрасывать подозрительный трафик. А этот трафик будет выглядеть очень подозрительным	На сервере Asterisk не требуется дополнительной настройки.
2. Ретрансляция вашей почты через хост, который знает и доверяет вашей системе. **Команде, занимающейся сервером ретрансляции, необходимы твердые знания технологий DNS и почтовых серверов.**	Необходимо настроить нижеследующий сервер передачи (это необходимо для доверия сообщениям, ретранслируемым с вашего сервера Asterisk).	Относительно простая настройка на сервере Asterisk.

Метод	Недостатки	Достоинства
3. Создать нормальный аккаунт пользователя на почтовом сервере (с действительным почтовым адресом), и посылать сообщения как аутентифицированный пользователь через эту платформу. **Мы рекомендуем этот метод**, так как он обычно хорошо работает и требования могут быть легко сообщены команде, поддерживающей вашу почту.	Чуть сложнее в настройке на сервере Asterisk.	Легко настроить учетную запись электронной почты для Asterisk: вам просто нужно создать в своей почтовой системе пользователя с именем «АТС компании» или именем, которое её идентифицирует, а затем использовать учетные данные этого пользователя для отправки всей электронной почты.

По существу: вам нужно убедиться, что почтовый агент (MTA)[4] вашего Asterisk сервера может посылать почту из оболочки/пользовательского аккаунта Asterisk. Движок голосовой почты Asterisk будет использовать теже механизмы для отправки сообщений по электронной почте.

Для получения подробной информации по почтовым агентам вы можете ознакомиться с книгами по администрированию линукса такими как *Unix и Linux. Руководство системного администратора*, 5-е издание или специфичное для MTA издание *Postfix: The Definitive Guide* издательства O'Reilly.

Бэкэнды хранения голосовой почты

Хранение сообщений в традиционных системах голосовой почты всегда было чрезмерно сложным.[5] Asterisk, не только предоставляет вам простой, основанный на файловой системе механизм хранения данных, но также предлагает несколько дополнительных параметров хранения сообщений.

Файловая система Linux

По умолчанию, Asterisk хранит голосовые сообщения в spool-каталоге */var/spool/asterisk/voicemail/<voicemailcontext>/<mailbox>*. Сообщения могут хранится в нескольких форматах (таких как `wav` и `wav49`), в зависимости от того, какой формат вы указали в разделе `[general]` вашего файла *voicemail.conf*. Ваши приветствия также хранятся в этом каталоге.

Asterisk не создает каталог для почтового ящика в котором нет записей (как в случае с новыми почтовыми ящиками), поэтому наличие этого каталога не может использоваться как надежный метод определения существования почтового ящика голосовой почты.

Рисунок 8-2 показывает на примере, что может находиться в каталоге почтового ящика. Этот почтовый ящик не имеет новых сообщений во входящих (каталог *INBOX*), имеет два сохраненных сообщения в каталоге *Old*, записи для приветствия (*greet*) и статусов *занято (busy)* и *недоступен (unavail)*.

4 Популярные MTA в наши дни - Postfix и Exim. Вездесущий sendmail все еще существует, хотя его популярность за последние несколько лет снизилась. По умолчанию вы найдете Postfix на ваших компьютерах с RHEL/CentOS и, скорее всего, Exim на платформах Debian/Ubuntu (хотя Postfix там же часто рекомендуется в качестве MTA).

5 Nortel обычно хранила свои сообщения в некоем специальном разделе в собственном формате, что делало практически невозможным извлечение сообщений из системы, отправку их по электронной почте, их архивирование, или действительно полезные действия с ними. Ах, старые добрые времена закрытых, проприетарных систем. Мы скучаем ... нет ... подождите ... мы не скучаем по ним!

```
/var/spool/asterisk/voicemail/default/301
      ./INBOX
      ./Old
            ./Old/msg0000.WAV
            ./Old/msg0000.txt
            ./Old/msg0001.WAV
            ./Old/msg0001.txt
      ./Urgent
      ./busy.WAV
      ./unavail.WAV
      ./greet.WAV
```

Рисунок 8-2. Пример каталога ящика голосовой почты

Для каждого сообщения есть соответствующий файл *msg####.txt*, содержащий информацию о конверте для сообщения. Файл *msg####.txt* также имеет критически важное значение для индикации ожидающего сообщения (MWI), так как это файл, который Asterisk ищет в *INBOX* чтобы определить необходимость включения индикатора сообщения для пользователя.

IMAP

Некоторые организации предпочитают управлять голосовой почтой как частью их почтовой системы. Это называется *унифицированным обменом сообщениями* в телекоммуникационной отрасли и его реализация традиционно является дорогостоящей и сложной. Asterisk позволяет довольно просто интегрировать голосовую и электронную почту либо через встроенный обработчик голосовой почты, либо через связь с сервером IMAP. Мы не рекомендуем IMAP интеграцию из-за большого объема работы при малой отдаче и выносим это за рамки данной книги.

Хранение сообщений в базе данных

Возможно настроить голосовую почту Asterisk для хранения сообщений в базе данных как Binary Large OBject (BLOB). Это выглядит как простой путь чтобы позволить синхронизировать сообщения между системами. Мы никогда не были поклонниками этой идеи, так как базы данных не создавались для хранения больших объемов бинарных данных. Кроме того есть много других путей синхронизации файлов между системами.

Заключение

Система голосовой почты Asterisk является зрелым и функциональным модулем и неотъемлемой частью любой АТС. Она вряд ли будет улучшена сверх того что уже доступно, но это тоже не является проблемой.

Глава 9. Интернационализация

Дэвид Дюффетт

Английский? Кто должен тратить время на его изучение? Я никогда не поеду в Англию!
– Дэн Касталланета

Телефония - одна из тех сфер жизни, где люди не любят сюрпризов: будь они дома или на работе. Когда люди пользуются телефонами, все, что выходит за рамки нормы, не оправдывается, и, как человек, который, вероятно, занимается поставками телефонных систем, вы будете знать, что неудовлетворенные ожидания могут привести к неописуемым страданиям с точки зрения дополнительной работы, потерянных денег и других проблем, связанных с недовольством клиентов.

Помимо того, что пользовательский интерфейс соответствует тому, что ожидают пользователи, также необходимо, чтобы ваша Asterisk чувствовала себя «как дома». Например, если исходящий вызов сделан по аналоговой линии (FXO), Asterisk будет нужно интерпретировать тоны, которые она «слышит» на линии (занято, вызов и т.д.).

По умолчанию (а возможно, как и следовало бы ожидать, поскольку она была “рождена в США”), Asterisk настроена на работу в Северной Америке. Однако, поскольку Asterisk развертывается во многих местах и (к счастью) люди со всего мира вносят свой вклад в неё, вполне возможно настроить Asterisk для правильной работы практически в любом месте, где вы решите её развернуть.

Если вы читали эту книгу с самого начала - глава за главой, вы уже сделали некоторый выбор во время установки и начальной конфигурации, которая настроила ваш Asterisk для работы в вашем регионе (и оправдает ожидания ваших клиентов).

Довольно много глав в этой книге содержат информацию, которая поможет вам интернационализировать[1] или (возможно, более правильно) локализовать вашу реализацию Asterisk. Цель этой главы - обеспечить единое место, куда можно будет ссылаться, обсуждать и объяснять все аспекты изменений, необходимые для внесения в телефонную систему на базе Asterisk в этом контексте. Причина использования фразы "телефонная система на основе Asterisk", а не просто "Asterisk", заключается в том, что некоторые изменения необходимо будет внести в другие части системы (IP-телефоны, ATAs и т.д.), в то время как другие изменения будут реализованы в конфигурационных файлах Asterisk и DAHDI.

Давайте начнем с составления списка (не в определенном порядке) вещей, которые возможно потребуется изменить для оптимизации вашей телефонной системы на основе Asterisk для данного местоположения за пределами Северной Америки. Вы можете выкрикнуть что-нибудь ещё, если хотите.…

- Язык/акцент подсказок
- Физическое подключение для интерфейсов ТфОП (FXO, BRI, PRI)
- Тоны, слышимые пользователями IP-телефонов и/или ATA
- Формат идентификатора вызывающего абонента (CallerID), передаваемый и/или принимаемый аналоговыми интерфейсами

1 *i18n* - это термин, используемый для сокращения слова *internationalization*, из-за его длины. Формат:<*первый_символ*><*число*><*последний_символ*>, где <*число*> - это количество букв между первой и последней буквами. Другие слова, такие как localization (L10n) и modularization (m12n), также нашли применение с этой схемой, которую Лейф находит немного смешной. Дополнительную информацию можно найти в W3C глоссарий онлайн.

- Сигналы для аналоговых интерфейсов, подаваемые или определяемые Asterisk
- Формат меток времени/даты для голосовой почты
- Способ, которым вышеуказанные метки времени/даты объявляются системой Asterisk
- Шаблоны в диалплане (IP-телефонов, ATA и самой Asterisk, если вы используете образец диалплана)
- Способ указания аналоговому устройству об ожидании голосовой почты (MWI)
- Тоны, подаваемые абонентам Asterisk (они вступают в игру, когда пользователь находится "внутри" системы; например, тоны, услышанные во время трансфера вызова)

Мы рассмотрим все в этом списке, приняв стратегию работы от внешнего края системы к самому ядру (самой Asterisk). Мы закончим с удобным контрольным списком того, что вам может понадобится изменить и где это сделать.

Хотя принципы, описанные в этой главе, позволят вам адаптировать установку Asterisk специально для вашего региона (или для вашего клиента), для обеспечения согласованности все наши примеры будут сосредоточены на том, как адаптировать Asterisk для одного региона: Соединенного Королевства.

Внешние устройства по отношению к серверу Asterisk

Существуют огромные различия между хорошим старомодным аналоговым телефоном и любым из большого количества IP-телефонов, и нам нужно подобрать одно из действительно фундаментальных различий, чтобы пролить свет на следующее объяснение, охватывающее настройки, которые нам, возможно, придется изменить на внешних по отношению к Asterisk устройствах, таких как IP-телефоны.

Вы когда-нибудь задумывались о том, что аналоговый телефон - это совершенно немое устройство (мы знаем, что базовая модель очень, очень дешевая), которое должно подключаться к интеллектуальной сети (ТфОП), тогда как IP-телефон (например, SIP или IAX2) - это очень умное устройство, которое подключается к немой сети (Интернет или любая обычная IP-сеть)? Рисунки 9-1 и 9-2 проиллюстрируют разницу.

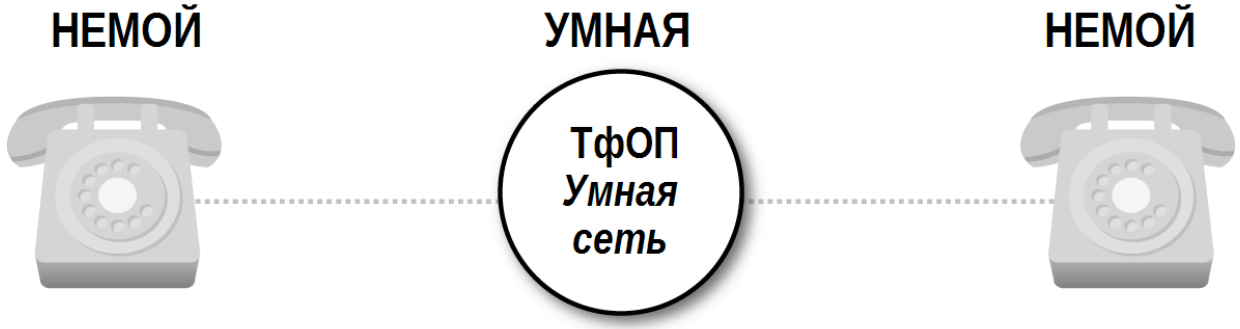

Рисунок 9-1. Старые времена: немые устройства подключаются к умной сети

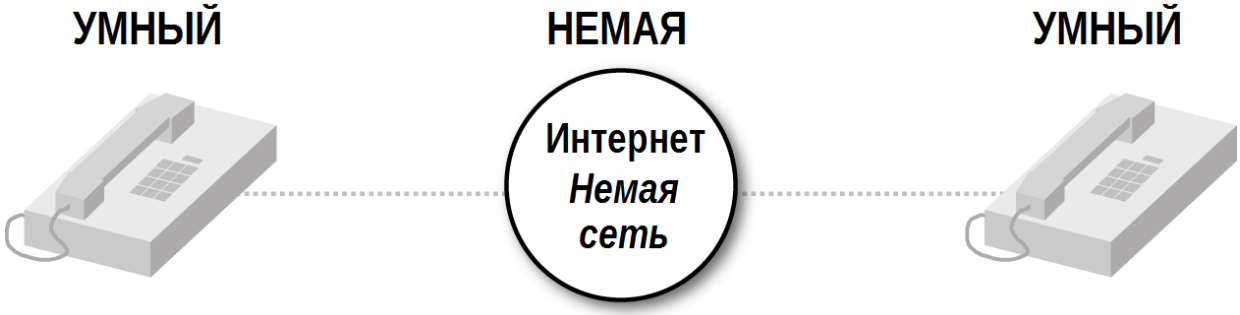

Рисунок 9-2. Ситуация на сегодняшний день: умные устройства подключаются через немую сеть

Можем ли мы взять два аналоговых телефона, подключить их непосредственно друг к другу и иметь функциональность, которую мы обычно связываем с обычным телефоном? Нет, конечно нет, потому что сеть предоставляет все: фактическое питание телефона, сигнал вызова (от местной станции или центрального офиса), информацию об идентификаторе вызывающего абонента (CallerID), сигнал вызова (от удаленной [ближайшего к телефону назначения] станции или ЦО), всю необходимую сигнализацию и так далее.

И наоборот, можем ли мы взять два IP-телефона, подключить их непосредственно друг к другу и получить некоторую разумную функциональность? Конечно можем, потому что весь интеллект находится внутри самих IP-телефонов - они обеспечивают тоны, которые мы слышим (сигнал вызова, звонок, занято) и запускают протокол, выполняющий всю необходимую сигнализацию (как правило SIP). Фактически, вы можете попробовать это для себя - большинство средних IP-телефонов имеют встроенный коммутатор Ethernet, поэтому вы можете подключить два IP-телефона непосредственно друг к другу с помощью обычного (прямого) кабеля Ethernet или просто подключить их через обычный коммутатор. Они должны иметь фиксированные IP-адреса в отсутствие DHCP-сервера и вы сможете набрать IP-адрес другого телефона, просто используя клавишу * для точек в адресе.

Рисунок 9-2 указывает на тот факт, что на IP-телефоне мы несем ответственность за настройку всех тонов, которые предоставила бы сеть в былые времена. Это можно сделать одним (по крайней мере) из двух способов. Первый заключается в настройке тонов, предоставляемых IP-телефоном на собственном веб-интерфейсе устройства. Вы делаете это, просматривая IP-адрес телефона (IP-адрес обычно можно получить с помощью опции меню на телефоне), а затем выбрав соответствующие параметры. Например, на IP-телефоне Yealink тоны устанавливаются на странице веб-графического интерфейса *Настройки* под вкладкой *Тоны* (где вы найдете список различных типов тонов, которые можно изменить — в случае Yealink это набор, второй вызов, КПВ, занято, переполнение, режим ожидания, перезвонить, инфо, заикание, сообщение, автоответ и быстрый набор).

Другой способ, которым эта конфигурация может быть применена - это автоматическое предоставление телефону этих настроек. Полное объяснение механизма автопровижининга выходит за рамки этой книги, но, как правило, вы можете настроить тоны в соответствующих атрибутах необходимых элементов в XML-файле.

В то время, как мы меняем настройки на IP-телефонах, есть еще две вещи, которые может потребоваться изменить, чтобы телефоны выглядели правильно и функционировали как часть системы.

Большинство телефонов отображают время в режиме ожидания, и, поскольку, многие люди находят особенно раздражающим, когда их телефоны показывают неправильное время; мы должны убедиться что отображается правильное местное время. Должно быть довольно легко найти соответствующую страницу веб-интерфейса (или атрибутов XML) для указания сервера синхронизации времени. Вы также обнаружите что есть настройки для перехода на летнее время и другие важные вещи.

Последнее, что нужно изменить - это потенциальный showstopper, когда речь идет о телефонном звонке - диалплан. Мы говорим не о диалплане, который находится в */etc/asterisk/extensions.conf*, а о диалплане телефона. Не все понимают что IP-телефоны также имеют схемы набора номеров, хотя эти диалпланы больше связаны с тем, какие строки набора разрешены, чем с тем, что делать с данным набором.

Общее правило, по-видимому, заключается в том, что если вы набираете при положенной трубке - встроенная схема набора номера проигнорируется, но если вы поднимаете трубку - в игру вступает диалплан телефона и может случиться так, что диалплан не позволит набрать необходимую строку. Хотя эта проблема может проявляться в отказе телефона передавать определенные типы номеров в Asterisk, она также может повлиять на любые коды функций, которые вы планируете использовать. Это может быть легко исправлено путем поиска номера модели телефона вместе с «UK dialplan» (или конкретным нужным вам регионом) или вы можете перейти на соответствующую страницу в веб-

интерфейсе пользователя и там либо вручную настроить диалплан, либо выбрать страну из выпадающего списка (в зависимости от типа телефона, с которым вы работаете).

Предварительное обсуждение конфигурации IP-телефона также относится к любым аналоговым телефонным адаптерам (ATA), которые вы планируете использовать, в частности к тем, которые поддерживают интерфейс FXS. Кроме того, может потребоваться указать некоторые электрические характеристики телефонного интерфейса, такие как линейное напряжение и импеданс, а также формат идентификатора вызывающего абонента, который будет работать с локальными телефонами. Все, что отличается - это способ получения IP-адреса для веб-интерфейса - обычно это делается набором определенного кода на подключенном аналоговом телефоне, что приводит к озвучиванию IP-адреса вызывающему абоненту.

Конечно, ATA также может иметь интерфейс FXO, который также должен быть настроен для правильного взаимодействия с аналоговой линией, предоставляемой в вашем регионе. Типы настроек, которые необходимо изменить, аналогичны интерфейсу FXS.

Что делать, если вы подключаете аналоговый телефон или линию к карте Digium? Мы рассмотрим это в следующий раз.

Подключение ТфОП, DAHDI, карт Digium и аналоговых телефонов

Прежде чем мы перейдем к конфигурации DAHDI и Asterisk, нам нужно физически подключиться к ТфОП. К сожалению, общемировых стандартов для подобных соединений не существует; на самом деле, даже в разных частях одной страны часто существуют различия.

Primary Rate Interfaces (PRI) обычно оканчиваются соединением RJ45 в настоящее время, хотя импеданс соединений может варьироваться. В некоторых странах (в частности, в Южной Америке) все еще можно найти PRI, обжатый двумя разъемами BNC: один для передачи и один для приема.

Проще говоря PRI, оконеченный RJ45, будет соединением ISDN, а если вы обнаружите, что соединение выполнено парой разъемов BNC (push-and-twist coaxial connectors), велика вероятность что вы имеете дело с более старым протоколом на основе CAS (например, MFCR2).

На Рисунке 9-3 показан адаптер, необходимый в том случае, если ваша телефонная компания поставила разъемы BNC (карты Sangoma/Digium требуют подключения RJ45). Он называется *balun*, поскольку преобразует из сбалансированного соединения (RJ45) в несбалансированное соединение (BNC) в дополнение к изменению импеданса соединения.

Basic Rate Interfaces (BRI) распроастранены в континентальной Европе и почти всегда поставляются через соединение RJ45.

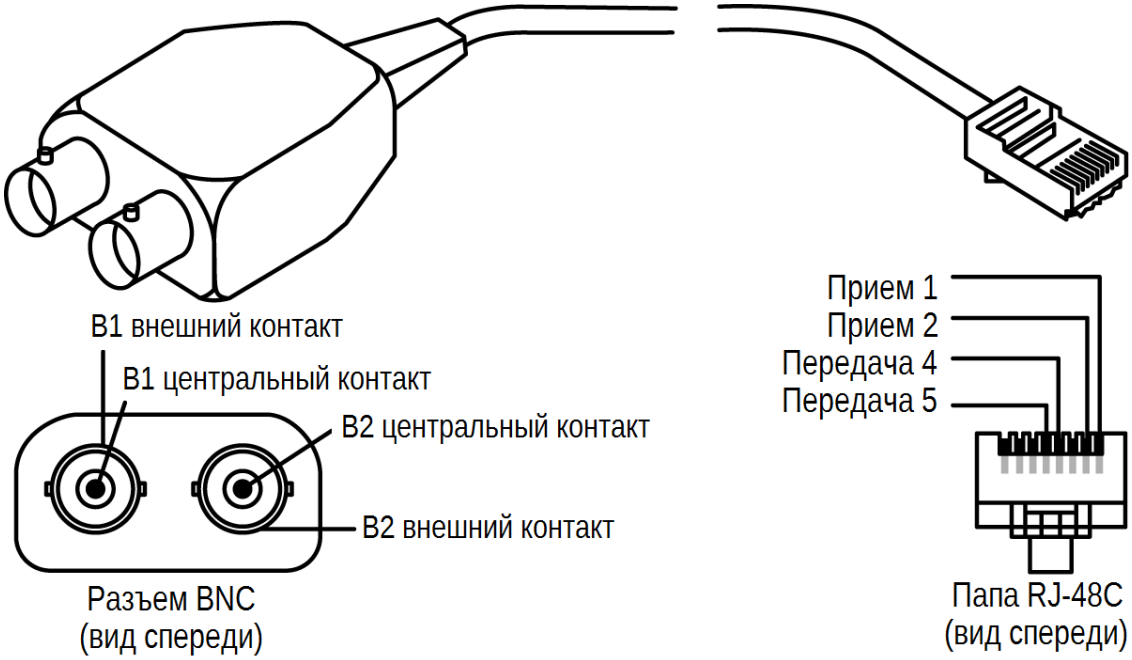

Рисунок 9-3. Balun

Аналоговые соединения сильно различаются в зависимости от местоположения - вы должны знать, какой тип разъема используется в вашей местности. Важно помнить, что аналоговая линия - это только два провода, и они должны подключаться к двум средним контактам разъема RJ11, подключаемым в плату Digium, другой конец является локальным. На Рисунке 9-4 показан коннектор, используемый в Великобритании, где два провода подключены к контактам 2 и 5.

Рисунок 9-4. Штекер BT, используемый для аналоговых соединений ТфОП в Великобритании (обратите внимание, присутствуют только контакты 2–5)

Интерфейс аппаратного устройства Digium Asterisk (Digium Asterisk Hardware Device Interface) или DAHDI на самом деле охватывает несколько вещей. Он содержит драйверы ядра для плат адаптеров телефонии, работающих в рамках DAHDI, а также утилиты автоматической настройки и инструменты тестирования. Эти части содержатся в двух отдельных пакетах (*dahdi-linux* и *dahdi-tools*), но мы также можем использовать один полный пакет, который называется *dahdi-linux-complete*. Все три пакета доступны на сайте Digium.

После того, как вы установили тип соединения PRI, предоставленного вам телекоммуникационным оператором, вам понадобятся некоторые дополнительные сведения для правильной настройки DAHDI и Asterisk (например, является ли соединение ISDN или протоколом на основе CAS). Опять же, вы найдете их в Главе 7.

Драйверы DAHDI

Соединения, в которых потребуется реальная локализация - это аналоговые интерфейсы. Для того, чтобы настроить телефонную систему на базе Asterisk так, чтобы она лучше всего работала в данной местности, сначала необходимо настроить некоторые низкоуровневые аспекты взаимодействия карты Digium с подключенным устройством или линией. Это делается через драйвер(ы) ядра DAHDI, в файле с именем */etc/dahdi/system.conf*.

В следующих строках (взятых из примера конфигурации, который вы получаете с новой установкой DAHDI), вы найдете настройки `loadzone` и `defaultzone`. Настройка `loadzone` позволяет выбрать

набор(ы) тонов, которые карта будет генерировать (подавать на аналоговые телефоны) и распознавать (на подключенных аналоговых телефонных линиях):

```
# Tone Zone Data
# ^^^^^^^^^^^^^^
# Наконец, вы можете предварительно загрузить некоторые тональные
# зоны, чтобы предотвратить их перезапись другими пользователями (если
# вы разрешаете пользователям (не root), открывать интерфейсы /dev/dahdi/*).
# Кроме того, это означает, что они не должны быть загружены во время выполнения.
# Формат - "loadzone=<zone>", где zone представляет собой двухбуквенный код страны.
# Вы также можете указать зону по умолчанию в "defaultzone=<zone>",
# где zone - это двухбуквенный код страны.
# Актуальный список зон можно найти в файле zonedata.c
#
loadzone = us
#loadzone = us-old
#loadzone=gr
#loadzone=it
#loadzone=fr
#loadzone=de
#loadzone=uk
#loadzone=fi
#loadzone=jp
#loadzone=sp
#loadzone=no
#loadzone=hu
#loadzone=lt
#loadzone=pl
defaultzone=us
#
```

В файле */etc/dahdi/system.conf* используется символ решётка (#) для обозначения комментария вместо точки с запятой (;), как в файлах */etc/asterisk*.

Хотя можно загрузить несколько различных наборов тонов (вы можете увидеть все наборы в *zonedata.c*) и переключаться между ними, в большинстве случаев вам понадобится только:

```
loadzone=uk    # загружаемый набор тонов
defaultzone=uk # по умолчанию DAHDI использует этот набор
```

…или тоны, необходимые для Вашего региона.

Если вы выполняете *dahdi_genconf* для автоматической (или она должна быть автомагическая?) настройки своих адаптеров DAHDI, то заметите, что сгенерированный */etc/dahdi/system.conf* объявит как `loadzone`, так и `defaultzone` как `us`. Несмотря на предупреждения не редактировать файл вручную, нормальная практика - изменить эти параметры на то, что Вам нужно.

Если вам интересно как проверить почтовый ящик, связанный с каналом, к которому подключен аналоговый телефон, на наличие голосовых сообщений, то это делается заикающимся тоном. Формат этой заикнутой мелодии определяется выбранной комбинацией `loadzone/defaultzone`.

В качестве быстрого отступления, аналоговые телефоны, у которых есть индикатор ожидания сообщения (например, светодиод или лампочка, мигающая для указания на новое голосовое сообщение), достигают этого, автоматически периодически отключаясь и слушая заикающийся тон. Вы можете увидеть это, посмотрев вывод командной строки Asterisk, чтобы увидеть как канал DAHDI становится активен (если у вас нет ничего лучше!).

Вот и все на уровне DAHDI. Мы выбрали протокол(ы) для соединений PRI или BRI, тип сигнализации для аналоговых каналов (все рассмотрено в Главе 7) и тональные сигналы для аналоговых соединений, которые только что были обсуждены.

Связь между Linux, DAHDI и Asterisk (и, следовательно, */etc/dahdi/system.conf* и */etc/asterisk/chan_dahdi.conf*) показана на Рисунке 9-5.

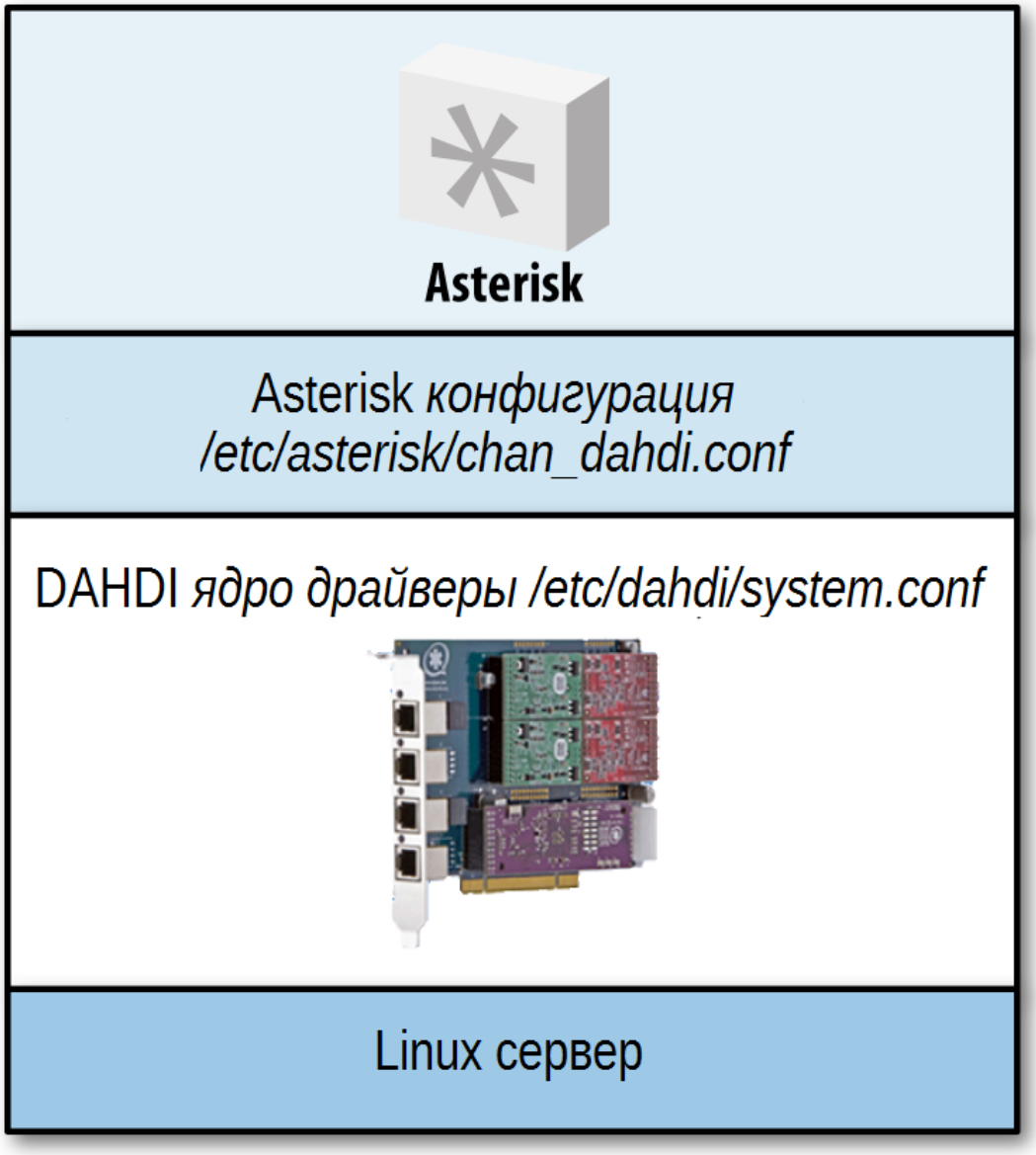

Рисунок 9-5. Связь между Linux, DAHDI и Asterisk

После завершения настройки на уровне DAHDI (в файле */etc/dahdi/system.conf*), вам нужно выполнить *dahdi_cfg -vvv* чтобы DAHDI перечитал конфигурацию. Это также хороший шанс, чтобы использовать *dahdi_tool* для проверки что все в порядке на уровне Linux.

Таким образом, если что-то не работает должным образом после настройки Asterisk для работы с адаптерами DAHDI, вы можете быть уверены, что проблема ограничена *chan_dahdi.conf* (или *#include dahdi-channels.conf* если вы используете эту часть вывода *dahdi_genconf*).

Интернационализация в Asterisk

Теперь, когда все настроено на уровне Linux, нам нужно только настроить Asterisk, чтобы использовать каналы, которые мы только что включили на уровне Linux, и настроить способ, которым Asterisk интерпретирует и генерирует информацию, поступающую из этих каналов или исходящую через них. Эта работа выполняется в файле */etc/asterisk/chan_dahdi.conf*.

В этом файле мы не только расскажем Asterisk какие каналы у нас есть (эти настройки будут соответствовать тому, что мы уже сделали в DAHDI), но и настроим ряд вещей, гарантирующих что Asterisk хорошо подходит для своего нового дома.

Caller ID

Ключевым компонентом этого изменения является CallerID (идентификатор вызывающего абонента). Хотя методы доставки CallerID являются довольно стандартными в мире BRI и PRI, в аналоговом мире они сильно различаются; таким образом, если вы подключите американский аналоговый телефон к телефонной сети Великобритании - он, фактически, будет работать как телефон, но CallerID отображаться не будет. Это потому, что эта информация передается разными способами в разных местах по всему миру, и американский телефон будет искать сигнализацию CallerID в формате США, в то время как телефонная сеть Великобритании будет поставлять ее в формате Великобритании (если она включена - CallerID не является стандартным в Великобритании, вы должны попросить, а иногда даже заплатить за это!).

Мало того, что формат отличается, но и способ сообщить телефону (или Asterisk) искать CallerID также может варьироваться от места к месту. Это важно, так как мы не хотим, чтобы Asterisk тратил время на поиск информации о CallerID, если он не отображается на линии.

Опять же, Asterisk по умолчанию использует североамериканский формат идентификатора (записи в */etc/asterisk/chan_dahdi.conf* не описывают этого, это просто значение по умолчанию) и для его изменения нам потребуется сделать несколько записей, которые опишут технические детали системы идентификации вызывающего абонента. В случае Великобритании о доставке информации CallerID сигнализирует смена полярности на телефонной линии (другими словами, жилы A и B пары телефонных проводов временно переключаются) и фактическая информация о CallerID доставляется в формате, известном как V.23 (частотная манипуляция или FSK). Таким образом, записи в файле *chan_dahdi.conf* для получения идентификатора вызывающего абонента в британском стиле на любых интерфейсах FXO будут выглядеть следующим образом:

```
cidstart=polarity ; доставка caller ID будет осуществляться
                  ; методом изменения полярности
cidsignalling=v23 ; доставка информации о caller ID
                  ; будет в формате V23
```

Конечно, вам также может понадобиться отправить CallerID, используя ту же информацию локальной сигнализации, на любые аналоговые телефоны, подключенные к интерфейсам FXS, и может потребоваться еще одна запись, поскольку в некоторых местах информация CallerID отправляется после указанного количества звонков. Если это так - вы должны использовать эту запись:

```
sendcalleridafter=2
```

Прежде, чем вы сможете сделать эти записи, вам нужно будет установить детали вашей локальной системы CallerID (кто-то из вашей местной телефонной компании или Google может помочь в этом, но также есть некоторая полезная информация в файле примера */etc/asterisk/chan_dahdi.conf*).

Язык и/или акцент подсказок

Как вы знаете, подсказки (или записи), используемые Asterisk, хранятся в */var/lib/asterisk/sounds/*. В более старых версиях Asterisk все звуки были в этом фактическом каталоге, но в наши дни вы найдете ряд подкаталогов, позволяющих использовать различные языки или акценты. Имена этих подкаталогов произвольны; вы можете называть их как угодно.

Обратите внимание, что имена файлов в этих каталогах должны соответствовать ожиданиям Asterisk — например файл */var/lib/asterisk/sound/en-hello.gsm* будет содержать слово "Hello" (произнесенное прекрасной Элисон), тогда как *hello.gsm* в */var/lib/asterisk/sounds/es* (для испанского в данном случае) будет содержать слово "Hola" (произнесенное испанским эквивалентом прекрасной Эллисон[2]).

По умолчанию используется каталог */var/lib/asterisk/sounds/en/*, так как же его изменить?

2 Который, по сути, та же самая Эллисон, которая делает английские подсказки; Джун Уоллак делает французские подсказки. Мужские подсказки с австралийским акцентом сделаны Камероном Туми. Все таланты озвучки также доступны для записи дополнительных подсказок. Смотрите страницу Digium IVR для получения дополнительной информации.

Есть два способа. Один из них - установить язык в файле конфигурации канала, на который поступают вызовы, используя директиву `language`. Например, строка:

```
language=en_UK
```

помещенная в *chan_dahdi.conf*, *sip.conf* и т.д. (Для применения в целом или только для данного канала или профиля) укажет Asterisk использовать звуковые файлы, найденные в */var/lib/asterisk/sounds/en_UK/* (которые могут содержать подсказки с британским акцентом) для всех вызовов, поступающих через эти каналы.

Другой способ - изменить язык во время телефонного звонка через диалплан. Этот параметр (наряду со многими атрибутами отдельного вызова) можно задать с помощью функции диалплана `CHANNEL()`. Полное описание функций диалплана см. в Главе 10.

В следующем примере вызывающий абонент может выбрать один из трех языков для продолжения вызова:

```
; дает выбор (1) Французского, (2) Испанского или (3) Немецкого
exten => s,1,Background(choose-language)
    same => n,WaitExten(5)

exten => 1,1,Set(CHANNEL(language)=fr)

exten => 2,1,Set(CHANNEL(language)=es)

exten => 3,1,Set(CHANNEL(language)=de)

; следующий приоритет для  расширений 1, 2 или 3 будет обрабатываться здесь
exten => _[123],n,Goto(menu,s,1)
```

Если вызывающий абонент нажмет 1 - звуки будут воспроизводиться из */var/lib/asterisk/sounds/fr/*; если он нажмет 2 - звуки будут браться из */var/lib/asterisk/sounds/es/* и так далее.

Как уже упоминалось, имена этих каталогов произвольны и необязательно должны быть длиной всего два символа - главное, чтобы они соответствовали имени подкаталога, созданного в директиве `language` в конфигурации канала или при установке в качестве аргумента в `CHANNEL(language)` через диалплан.

Штампы времени/даты и произношение

Asterisk использует системное время Linux с хост-сервера, как и следовало ожидать, но у нас могут быть пользователи системы, находящиеся в разных часовых поясах или даже в разных странах. Голосовая почта - это место, где резина попадает на дорогу, так как именно здесь пользователи вступают в контакт с информацией о времени/дате.

Рассмотрим сценарий, в котором одни пользователи системы находятся в США, а другие - в Великобритании.

Помимо разницы во времени, еще одна вещь, которую следует учитывать — это то, какой формат даты и времени используют люди в разных местах - в США даты обычно упорядочиваются по месяцам, дням, годам, а время указывается в 12-часовом формате (например, 2:54 P.M.).

В отличие от этого, в Великобритании даты упорядочены как день, месяц, год и время указывается в 24-часовом формате (14:54 hrs), хотя некоторые люди в Великобритании предпочитают 12-часовой формат часов, поэтому мы рассмотрим его тоже.

Поскольку все эти вещи связаны с голосовой почтой, Вы были бы правы, предположив, что мы настраиваем это в */etc/asterisk/voicemail.conf* - в частности, в разделе файла `[zonemessages]`.

Вот часть `[zonemessages]` образца файла *voicemail.conf*, поставляемого с Asterisk, добавлены зоны `UK24` (для британцев, которые любят 24-часовой формат) и `UK12` (для британцев, предпочитающих 12-часовой формат):

```
[zonemessages]
; Пользователи могут находиться в разных часовых поясах или иметь разные
; объявления для сообщения при входе в систему голосовой почты. Здесь можно
; установить сообщение и часовой пояс для каждого пользователя. Определите
; пользователя в одну из этих зон с помощью tz=атрибут в параметрах почтового
; ящика. Конечно, замена языка здесь всё еще применяется, поэтому у вас
; может быть несколько деревьев каталогов, имеющих альтернативные
; варианты языка.
;
; Найдите в /usr/share/zoneinfo/ названия часовых поясов.
; Посмотрите страницу руководства для strftime для быстрой справки о том,
; как выполняется подстановка переменных в значениях ниже.
;
; Поддерживаемые значения:
; 'filename' имя звукового файла (требует одиночные кавычки вокруг имени
; файла)
; ${VAR} подстановка переменных
; A или a - день недели (Saturday, Sunday, ...)
; B или b или h - название месяца (January, February, ...)
; d или e - число в месяце (first, second, ... thirty-first)
; Y - год
; I или l - час, 12-часовой формат
; H - час, 24-часовой формат (одноразрядные часы перед "oh")
; k - час, 24-часовой формат (одноразрядные часы НЕ перед "oh")
; M - минута, с 00 произносится как "o'clock"
; N - минута, с 00 произносится как "hundred" (военное время США)
; P или p - AM или PM
; Q "today", "yesterday" или ABdY
; (*примечание: нестандартное значение strftime)
; q " (для today), "yesterday", weekday или ABdY
; (*примечание: нестандартное значение strftime)
; R - 24-часовой формат, включая минуты
;
eastern=America/New_York|'vm-received' Q 'digits/at' IMp
central=America/Chicago|'vm-received' Q 'digits/at' IMp
central24=America/Chicago|'vm-received' q 'digits/at' H N 'hours'
military=Zulu|'vm-received' q 'digits/at' H N 'hours' 'phonetic/z_p'
european=Europe/Copenhagen|'vm-received' a d b 'digits/at' HM
UK24=Europe/London|'vm-received' q 'digits/at' H N 'hours'
UK12=Europe/London|'vm-received' Q 'digits/at' IMp
```

Эти зоны не только определяют время, но и указывают порядок следования и считывания времени и дат.

Создав эти зоны, мы можем перейти к контекстной части *voicemail.conf*, чтобы связать соответствующие почтовые ящики с правильными зонами:

```
[default]
4001 => 1234,Russell Bryant,rb@shifteight.org,,|tz=central
4002 => 4444,David Duffett,dd@shifteight.org,,|tz=UK24
4003 => 4450,Mary Poppins,mp@shifteight.org,,|tz=UK12|attach=yes
```

Как вы могли заметить, когда мы объявляем почтовый ящик, мы также (необязательно) связываем его с определенной зоной. Полную информацию о голосовой почте можно найти в Главе 8.

Последнее, что нужно локализовать в нашей конфигурации Asterisk - это тоны, воспроизводимые для абонентов Asterisk, находящихся внутри системы (например, тоны, которые абонент слышит во время трансфера).

Как было указано ранее в этой главе, исходные сигналы, которые люди слышат при звонке в систему, будут поступать с IP-телефона или с DAHDI для аналоговых каналов.

Эти тоны задаются в */etc/asterisk/indication.conf*. Вот часть файла примера, где вы можете увидеть данный регион, указанный в директиве `country`. Нам просто нужно изменить код страны соответствующим образом:

```
;
; indications.conf
```

```
;
; Конфигурационный файл для индикаций конкретного местоположения
;
; ПРИМЕЧАНИЕ:
; При добавлении стран в этот файл, пожалуйста, держите их в алфавитном
; порядке в соответствии с 2-символьными кодами стран!
;
; Категория [general] предназначена для некоторых глобальных переменных.
; Все остальные категории интерпретируются как индикации для
; конкретного местоположение
;
[general]
country=uk ; по умолчанию - US, поэтому мы изменили его на uk
```

Ваш диалплан должен будет отражать схему нумерации для вашего региона. Если вы еще не знаете схему для своего региона, местный регулятор электросвязи обычно предоставляет подробную информацию об этом. Кроме того, пример диалплана в */etc/asterisk/extensions.conf* конечно же содержит номера и шаблоны для Северной Америки.

Вывод - простая шпаргалка

Как вы теперь можете видеть - есть довольно много вещей, которые необходимо изменить, чтобы полностью локализовать вашу телефонную систему на основе Asterisk и не все из них находятся в конфигурации Asterisk или даже DAHDI — некоторые параметры должны быть изменены на подключенных IP-телефонах или ATA.

Прежде, чем мы покинем главу, взгляните на Таблицу 9-1: шпаргалка, подсказывающая что и где изменить для дальнейшего использования.

Таблица 9-1. Шпаргалка интернационализации

Что изменить	Где это изменить
Тоны выполнения вызова	• IP-телефоны—в самом телефоне • ATAs—в самом ATA • Аналогвоые телефоны—DAHDI (*/etc/dahdi/system.conf*)
Тип PRI/BRI и протокол	DAHDI—*/etc/dahdi/system.conf* and */etc/asterisk/chan_dahdi.conf*
Физические подключения ТфОП	• Balun если требуется для PRI • Подключить аналоговую пару к средним двум контактам RJ11, подключенному к плате Digium
Caller ID на аналоговой линии	Asterisk—*/etc/asterisk/chan_dahdi.conf*
Языковые подсказки и/или акцент	• Канал—*/etc/asterisk/sip.conf*, */etc/asterisk/iax.conf*, */etc/asterisk/chan_dahdi.conf*, и тд. • Диалплан—функция `CHANNEL(language)`
Штампы время/дата и произношение для голосовой почты	Asterisk—*/etc/asterisk/voicemail.conf*
Тоны, предоставляемые Asterisk	Asterisk—*/etc/asterisk/indications.conf*

Пусть все ваши развертывания Asterisk чувствуют себя как дома…

Для получения списка всех способов, которыми технология не смогла улучшить качество жизни, нажмите три.

– Элис Кан

Хорошо. Основы диалплана позади, но вы знаете что это еще не все. Если вы еще не разобрались с Главой 6, пожалуйста, вернитесь и прочтите ее еще раз. Мы собираемся перейти к более сложным темам.

Выражения и манипуляции с переменными

Мы начинаем наше погружение в более глубокие аспекты диалпланов: пришло время познакомить вас с несколькими инструментами, которые значительно увеличат мощь, которую вы можете использовать в своем диалплане. Эти конструкции добавляют невероятный интеллект к вашему диалплану, позволяя ему принимать решения на основе различных критериев, которые вы определяете. Наденьте свой мыслительный колпачок и давайте начнем.

В этой главе мы используем лучшие практики, которые были разработаны на протяжении многих лет при создании диалплана. Основное, что вы заметите, это то, что все первые приоритеты начинаются с приложения `NoOp()` (что просто означает No Operation - отсутствие операции; ничего функционального не произойдет). Другое заключается в том, что все следующие строки будут начинаться с `same => n` что является ярлыком, который говорит: "используется тоже (same) расширение, что и ранее". Кроме того, отступ - это два пробела.

Базовые выражения

Выражения - это комбинации переменных, операторов и значений, которые соединяются вместе для получения результата. Выражение может проверять значения, изменять строки или выполнять математические вычисления. Допустим, у нас есть переменная под названием `COUNT`. На простом английском языке два выражения, использующие эту переменную, могут быть [`COUNT` плюс 1] или [`COUNT` делить на 2]. Каждое из этих выражений имеет определенный результат или значение, зависящее от значения данной переменной.

В Asterisk выражения всегда начинаются со знака доллара и открывающей квадратной скобки и заканчиваются закрывающей квадратной скобкой, как показано здесь:

```
$[expression]
```

Таким образом, мы запишем наши два примера следующим образом:

```
$[${COUNT} + 1]
$[${COUNT} / 2]
```

Когда Asterisk встречает выражение в диалплане, он заменяет все выражение результирующим значением. Важно отметить, что это происходит *после* подстановки переменных. Для демонстрации рассмотрим следующий код:[1]

```
exten => 321,1,NoOp()
 same => n,Answer()
 same => n,Set(COUNT=3)
```

1 Помните, что когда вы *ссылаетесь* на переменную - вы можете вызывать ее по ее имени, но когда вы ссылаетесь на *значение* переменной, вы должны использовать знак доллара и скобки вокруг ее имени.

```
same => n,Set(NEWCOUNT=$[${COUNT} + 1])
same => n,SayNumber(${NEWCOUNT})
```

Во втором приоритете мы присваиваем значение `3` переменной с именем `COUNT`.

В третьем приоритете задействовано только одно приложение - `Set()`, но на самом деле происходят три вещи:

1. Asterisk заменяет `${COUNT}` на число `3` в выражении. Выражение фактически становится таким: `same => n,Set(NEWCOUNT=$[3 + 1])`
2. Asterisk вычисляет выражение, прибавляя `1` к `3`, и заменяет его вычисленным значением `4`: `same => n,Set(NEWCOUNT=4)`
3. Приложение `Set()` присваивает значение `4` новой переменной `COUNT`.

Третий приоритет просто вызывает приложение `SayNumber()`, которое проговаривает текущее значение переменной `${NEWCOUNT}` (устанавливается в значение `4` во втором приоритете).

Попробуйте это в своём диалплане.

Операторы

Когда вы создаете диалплан Asterisk: вы действительно пишете код на специализированном языке сценариев. Это означает, что диалплан Asterisk, как и любой язык программирования, распознает символы, называемые операторами, позволяющие управлять переменными. Давайте рассмотрим типы операторов, доступных в Asterisk:

Логические операторы

Эти операторы оценивают "истинность" утверждения. В вычислительных терминах это по существу относится к тому, является ли утверждение чем-то или ничем (ненулевым или нулевым, истинным или ложным, включенным или выключенным и т.д.). Логическими операторами являются:

`expr1 | expr2`

Этот оператор (называемый оператором “или” или “пайп”) возвращает оценку *`expr1`* если она истинна (ни одна строка не равна нулю). В противном случае он возвращает оценку *`expr2`*.

`expr1 & expr2`

Этот оператор (называемый “и”) возвращает вычисление *`expr1`*, если оба выражения истинны (т.е. ни одно из выражений не является в пустой строкой или нулем). В противном случае возвращает ноль.

`expr1` `{=, >, >=, <, <=, !=}` *`expr2`*

Эти операторы возвращают результаты сравнения целых чисел, если оба аргумента являются целыми числами; в противном случае возвращают результаты сравнения строк. Результат каждого сравнения равен `1`, если указанное отношение истинно, или `0` если отношение ложно. (Если вы выполняете сравнение строк - они будут выполняться в соответствии с текущими локальными настройками вашей операционной системы.)

Математические операторы

Хотите выполнить расчет? Вам понадобится один из них:

`expr1` `{+, -}` *`expr2`*

Эти операторы возвращают результат сложения или вычитания целочисленных аргументов.

`expr1` `{*, /, %}` *`expr2`*

Возвращают, соответственно, результат умножения, целочисленного деления или остатка деления целочисленных аргументов.

Операторы регулярных выражений

Вы также можете использовать операторы регулярных выражений в Asterisk:

Дополнительную информацию об особенностях работы оператора регулярного выражения в Asterisk можно найти на сайте Уолтера Докса.

expr1 `:` *expr2*

Этот оператор сопоставляет *expr1* с *expr2*, где *expr2* должно быть регулярным выражением.[2] Регулярное выражение привязывается к началу строки с неявным `^`.[3]

Если шаблон не содержит подвыражения - возвращается количество совпадающих символов. Он вернет `0` если совпадений не найдено. Если шаблон содержит подвыражение -- \(... \) -- возвращается строка, соответствующая `\1`. Если совпадение не найдено - возвращается пустая строка.

expr1 `=~` *expr2*

Этот оператор работает так же, как и оператор `:`, за исключением того, что он не привязан к началу.

Функции диалплана

Функции диалплана позволяют добавить больше мощи к вашим выражениям; вы можете думать о них как об интеллектуальных переменных. Функции диалплана позволяют вычислять длины строк, даты и время, контрольные суммы MD5 и т.д. в пределах выражений диалплана.

Вы увидите использование функции `Playback(silence/1)` во всех примерах в этой главе. Мы делаем так поскольку она ответит на линию, если еще не ответили и воспроизведёт некоторую тишину на линии. Это позволяет другим приложениям, таким как `SayNumber()`, воспроизводить звук без пропусков.

Синтаксис

Функции диалплана имеют следующий базовый синтаксис:

```
FUNCTION_NAME(argument)
```

Вы ссылаетесь на имя функции так же, как и на имя переменной, но на значение функции ссылаются с добавлением знака доллара, открывающейся и закрывающейся фигурной скобки:

```
${FUNCTION_NAME(argument)}
```

Функции также могут инкапсулировать другие функции, например:

```
${FUNCTION_NAME(${FUNCTION_NAME(argument)})}
 ^              ^ ^              ^        ^^^^
 1              2 3              4        4321
```

Как вы, вероятно, уже поняли необходимо быть очень осторожными, чтобы убедиться в наличии соответствующих круглых и фигурных скобок. В предыдущем примере мы обозначили открывающие круглые и фигурные скобки цифрами, а их соответствующие закрывающие аналоги - теми же цифрами.

Примеры функций диалплана

Функции часто используются совместно с приложением `Set()` для получения или установки значения переменной. В качестве простого примера рассмотрим функцию `LEN()`. Эта функция вычисляет длину строки своего аргумента:

```
exten => 205,1,Answer()
  same => n,SayDigits(123)
```

2 Для получения дополнительной информации о регулярных выражениях возьмите копию справочника Jeffrey E. F. Friedl's *Mastering Regular Expressions* (O'Reilly, 2006), или посетите *http://www.regular-expressions.info*.

3 Если вы не знаете, что ^ имеет отношение к регулярным выражениям, то просто обязаны прочитать Mastering Regular Expressions (Освоение регулярный выражений). Это изменит вашу жизнь!

```
  same => n,SayNumber(123)
  same => n,SayNumber(${LEN(123)})
```

Давайте рассмотрим еще один простой пример. Если бы мы хотели установить один из различных таймаутов канала - мы могли бы использовать функцию `TIMEOUT()`. Функция `TIMEOUT()` принимает три аргумента: `absolute`, `digit` и `response`. Чтобы установить тайм-аут с помощью функции `TIMEOUT()`, мы могли бы использовать приложение `Set()`, например:

```
exten => 206,1,Answer()
  same => n,Set(TIMEOUT(response)=1)
  same => n,Background(enter-ext-of-person)
  same => n,WaitExten()  ; TIMEOUT() установлен в значение 1
  same => n,Playback(like_to_tell_valid_ext)
  same => n,Set(TIMEOUT(response)=5)
  same => n,Background(enter-ext-of-person)
  same => n,WaitExten()  ; Теперь должно быть 5 секунд
  same => n,Playback(укажите_действительный_файл)
  same => n,Hangup()
```

Обратите внимание на отсутствие `${ }` вокруг назначения с помощью функции. Так же, как если бы мы присваивали значение переменной, мы присваиваем значение функции без использования инкапсуляции `${}`; однако, если мы хотим использовать значение, возвращаемое функцией, то нам нужна инкапсуляция.

```
exten => 207,1,Answer()
  same => n,Set(TIMEOUT(response)=1)
  same => n,SayNumber(${TIMEOUT(response)})
  same => n,Set(TIMEOUT(response)=5)
  same => n,SayNumber(${TIMEOUT(response)})
  same => n,Hangup()
```

Вы можете получить список всех активных функций с помощью следующей команды CLI:

```
*CLI> core show functions
```

Или по определенной функции, например `CALLERID()`, командой:

```
*CLI> core show function CALLERID
```

Ближе к концу этой главы мы рассмотрим несколько функций, с которыми вы захотите поэкспериментировать. Далее в книге мы покажем вам как создавать функции на основе баз данных с помощью `func_odbc`.

Условное ветвление

Расширенная логика, предоставляемая через выражения и функции, позволит вашему диалплану принимать более сложные решения, что часто приводит к *условному ветвлению*.

Приложение GotoIf()

Ключом к условному ветвлению является приложение `GotoIf()`. `GotoIf()` вычисляет выражение и отправляет вызывающий объект в определенное место назначения в зависимости от того, имеет ли выражение значение истинности или лжи.

`GotoIf()` использует специальный синтаксис, часто называемый *условным синтаксисом*:

```
GotoIf(expression?destination1:destination2)
```

Если выражение принимает значение "истина" - вызывающий объект отправляется в *`destination1`*. Если выражение принимает значение "ложь" - вызывающий объект отправляется в *`destination2`*. Итак, что же такое истина и что такое ложь? Пустая строка и число `0` оцениваются как ложь. *Все остальное оценивается как истина.*

Каждый из пунктов назначения может быть одним из следующих:

- Метка приоритета в пределах одного расширения, например `weasels`
- Расширение и метка приоритета в одном контексте, например `123,weasels`

- Контекст, расширение и метка приоритета, такие как `incoming,123,weasels`

Давайте используем `GotoIf()` в качестве примера. Вот небольшое приложение для подбрасывания монет. Вызовите его несколько раз, чтобы проверить правильность.

```
exten => 209,1,Noop(Test use of conditional branching to labels)
  same => n,GotoIf($[ ${RAND(0,1)} = 1 ]?weasels:iguanas)
; same => n,GotoIf(${RAND(0,1)}?weasels:iguanas) ;тоже работает, но не в каждой ситуации

  same => n(weasels),Playback(weasels-eaten-phonesys) ; ПРИМЕЧАНИЕ: ТО ЖЕ РАСШИРЕНИЕ
  same => n,Hangup()

  same => n(iguanas),Playback(office-iguanas) ; ВСЕ ТО ЖЕ РАСШИРЕНИЕ
  same => n,Hangup()
```

Вы заметите, что мы использовали приложение `Hangup()` после каждого использования приложения `Playback()`. Это делается для того, чтобы при переходе к метке `weasels` вызов останавливался до того, как он попадет на звуковой файл *office-iguanas*. Становится все более распространенным видеть расширения, разбитые на несколько компонентов (защищенных друг от друга командой `Hangup()`), каждый из которых представляет собой отдельную последовательность шагов, выполняемых после `GotoIf()`.

Предоставление только ложного условного пути

Любой из пунктов назначения может быть опущен (но не оба). Если выражение оценивается как пустое назначение - Asterisk просто переходит к следующему приоритету в текущем расширении.

Мы могли бы выполнить предыдущий пример следующим образом:

```
exten => 209,1,Noop(Test use of conditional branching)
  same => n,GotoIf($[ ${RAND(0,1)} = 1 ]?:iguanas)
  same => n,Playback(weasels-eaten-phonesys) ; больше нет ярлыка weasels
  same => n,Hangup()
  same => n(iguanas),Playback(office-iguanas) ; ОБРАТИТЕ ВНИМАНИЕ ЧТО ЭТО ТО ЖЕ РАСШИРЕНИЕ
  same => n,Hangup()
```

Между `?` и `:` ничего нет, так что если оператор оценивается как истина, выполнение будет продолжено на следующем шаге. Поскольку это то, что мы хотим, ярлык не нужен.

Мы действительно не рекомендуем делать так, потому что это трудно читать. Тем не менее, вы увидите такие диалпланы, поэтому хорошо знать, что этот синтаксис технически корректен.

Вместо того, чтобы использовать метки (лейблы), мы могли бы также отправить вызов на различные расширения. Поскольку они недоступны - мы можем использовать буквы, а не цифры для "номера" расширения. В этом примере условная ветвь отправляет вызов на совершенно разные расширения в одном и том же контексте. В остальном результат тот же.

```
exten => 210,1,Noop(Test use of conditional branching to extensions)
  same => n,GotoIf($[ ${RAND(0,1)} = 1 ]?weasels,1:iguanas,1)

exten => weasels,1,Playback(weasels-eaten-phonesys) ; РАЗЛИЧНЫЕ РАСШИРЕНИЯ
  same => n,Hangup()

exten => iguanas,1,Playback(office-iguanas) ; ТАКЖЕ РАЗЛИЧНЫЕ РАСШИРЕНИЯ
  same => n,Hangup()
```

Рассмотрим еще один пример условного ветвления. На этот раз мы будем использовать оба `Goto()` и `GotoIf()` для обратного отсчета от 5, а затем повесим трубку:

```
exten => 211,1,NoOp()
  same => n,Answer()
  same => n,Set(COUNT=5)

  same => n(start),GotoIf($[ ${COUNT} > 0 ]?:goodbye)
  same => n,SayNumber(${COUNT})
```

```
same => n,Set(COUNT=$[ ${COUNT} - 1 ])
same => n,Goto(start)

same => n(goodbye),Playback(vm-goodbye)
same => n,Hangup()
```

Давайте проанализируем этот пример. Во втором приоритете мы задаем переменную `COUNT` равную `5`. Далее, проверяем, чтобы увидеть если `COUNT` больше `0`. Если это так - мы переходим к следующему приоритету. (Не забывайте, что если мы опустим назначение в приложении `GotoIf()`, управление перейдет к следующему приоритету.) Там мы произносим число, вычитаем `1` из него и возвращаемся к метке приоритета `start`. Опять же, если `COUNT` меньше или равен `0`, управление переходит к метке приоритета `goodbye`; в противном случае мы запускаем цикл еще раз.

Кавычки и префиксы переменных в условных ветвлениях

Сейчас самое время воспользоваться моментом и посмотреть на некоторые небрежные вещи с условными ветвлениями. В Asterisk недопустимо иметь нулевое значение по обе стороны от оператора сравнения. Давайте рассмотрим примеры, которые могли бы привести к ошибке:

```
$[ = 0 ]
$[ foo = ]
$[ > 0 ]
$[ 1 + ]
```

Любой из наших примеров вызовет такое предупреждение:

```
WARNING[28400][C-000000eb]: ast_expr2.fl:470 ast_yyerror: ast_yyerror():
syntax error: syntax error, unexpected '=', expecting $end; Input:
 = 0
 ^
```

Это маловероятно (если у вас нет опечатки), что вы целенаправленно реализуете что-то из наших примеров. Однако, когда вы выполняете математическое действие или сравнение с неназначенной переменной канала, это фактически то, что делаете Вы.

Примеры, используемые нами чтобы показать вам как работает условное ветвление, являются недопустимыми. Мы сначала инициализировали переменную и можем ясно видеть, что переменная канала, которую мы используем в нашем сравнении, была установлена, поэтому мы в безопасности. Но что, если вы не всегда так уверены?

В Asterisk строки необязательно должны быть заключены в двойные или одинарные кавычки, как во многих языках программирования. Фактически, если вы используете двойные или одинарные кавычки, это будет буквенной конструкцией в строке. Если мы посмотрим на следующие фрагменты расширения...

```
same => n,Set(TEST_1=foo)
same => n,Set(TEST_2='foo')
same => n,NoOp(Are TEST_1 and TEST_2 equiv? $[${TEST_1} = ${TEST_2}])
```

...мы должны отметить, что значение, возвращаемое нашим сравнением в `NoOp()`, не будет значением `1` (значения совпадают или *истина*), возвращаемое значение будет `0` (значения не совпадают или *ложь*).

Мы можем использовать это в своих интересах при выполнении сравнений, обертывая наши переменные канала в одинарные или двойные кавычки. Делая это, мы удостоверяемся, что даже когда переменная канала не может быть установлена, наше сравнение будет допустимым

синтаксисом.

В следующем примере мы получим ошибку:

```
exten => 212,1,NoOp()
  same => n,GotoIf($[ ${TEST} != valid ]?error_handling)
  same => n,Hangup() ; We're getting an error and ending up here

  same => n(error_handling),Playback(goodbye)
  same => n,Hangup()
```

Однако, мы можем обойти это, обернув то, что мы сравниваем, в дополнительные символы (в данном случае кавычки). Тот же пример, но сделан допустимым:

```
exten => 213,1,NoOp()
  same => n,GotoIf($[ "${TEST}" != "valid" ]?error_handling)
  same => n,Hangup()

  same => n(error_handling),Playback(goodbye)
  same => n,Hangup()
```

Даже если `${TEST}` не была установлена (другими словами, она не существует и поэтому не имеет значения), мы все равно делаем сравнение чего-то:

```
$["" != "valid"]
```

Если вы привыкнете распознавать эти ситуации и использовать методы обертки и префикса, описанные нами, вы напишете гораздо более безопасные диалпланы.

Обратите внимание еще раз, что символ кавычки не имеет никакого особого значения здесь. Мы использовали его только потому, что это логический символ для этой цели. Следующее тоже работает:

```
  same => n,GotoIf($[_${TEST}_ != _valid_]?error_handling)
;ИЛИ
  same => n,GotoIf($[AAAAA${TEST}AAAAA != AAAAAvalidAAAAA]?error_handling)
```

Не все символы будут работать, так как некоторые могут иметь другие значения для Asterisk и вызвать проблемы. Придерживайтесь кавычек и всё должно быть в порядке.

Классический пример условного ветвления ласково называют логикой "психо-экс". Если CallerID входящего вызова совпадает с номером телефона человека, с которым вы больше никогда не захотите разговаривать, Asterisk выдает другое сообщение, чем для любого другого абонента. Хотя он несколько прост и примитивен в данном случае это хороший пример для изучения условного ветвления в диалплане Asterisk.

В этом примере используется функция `CALLERID()`, позволяющая получить информацию об CallerID при входящем вызове. Предположим, ради этого примера, что номер телефона жертвы 888-555-1212:[4]

```
exten => 214,1,NoOp(CALLERID(num): ${CALLERID(num)} CALLERID(name): ${CALLERID(name)})
  same => n,GotoIf($[ ${CALLERID(num)} = 8885551212 ]?reject:allow)

  same => n(allow),Dial(${UserA_DeskPhone})
  same => n,Hangup()

  same => n(reject),Playback(abandon-all-hope)
  same => n,Hangup()
```

4 Если вы хотите проверить это (то, что делаете), то можете выбрать одно из ваших рабочих лабораторных устройств, и в базе данных Asterisk под таблицей `ps_endpoints` установить поле `callerid` в `'8885551212'`. Затем вы можете позвонить с него на номер 214, чтобы увидеть блок в действии.
`UPDATE asterisk.ps_endpoints SET callerid='8885551212' WHERE id='<endpoint выбранный в качестве жертвы>'`

В приоритете 1 мы вызываем приложение GotoIf(). Оно сообщает Asterisk перейти к приоритету с меткой reject, если номер CallerID соответствует 8885551212, а в противном случае перейти к метке приоритета allow (мы могли бы просто опустить имя метки в результате чего GotoIf() просто провалился).[5] Если CallerID абонента совпадает - управление вызовом переходит к метке приоритета reject, которая воспроизводит тонкий намёк нежелательному абоненту. В противном случае вызов пытается набрать получателя по каналу, на который ссылается глобальная переменная UserA_DeskPhone.

Условное ветвление по времени с GotoIfTime()

Другой способ использования условного ветвления в диалплане - это использование приложения GotoIfTime(). В то время, как GotoIf() оценивает выражение для дальнейших действий, GotoIfTime() смотрит на текущее системное время и использует его, чтобы решить, следует ли следовать другой ветви в диалплане.

Наиболее очевидное использование этого приложения - это озвучивание вашим абонентам другое приветствие до и после рабочих часов.

Синтаксис приложения GotoIfTime() выглядит следующим образом:

```
GotoIfTime(times,days_of_week,days_of_month,months?label)
```

Короче говоря, GotoIfTime() отправляет вызов на указанный label, если текущая дата и время соответствуют критериям, указанным *times*, *days_of_week*, *days_of_month* и *months*. Давайте рассмотрим каждый аргумент более подробно:

times

Это список одного или нескольких временных диапазонов в 24-часовом формате. Например, с 9:00 утра до 5:00 вечера будет указано как 09:00-17:00. День начинается в 0:00 и заканчивается в 23:59.

Стоит отметить, что время будет правильно оборачиваться. Таким образом, если вы хотите указать время закрытия вашего офиса, то можете указать 18:00-9:00 в параметре *times*, и оно будет работать как и ожидалось. Обратите внимание, что этот метод работает также и для других компонентов GotoIfTime(). Например, вы можете написать sat-sun, чтобы указать выходные дни.

days_of_week

Это список из одного или нескольких дней недели. Дни должны быть указаны как mon, tue, wed, thu, fri, sat и/или sun. С понедельника по пятницу будет выражаться как mon-fri. Вторник и четверг будут обозначены как tue&thu.

Обратите внимание, что можно указать совокупность диапазонов и одного дня, как: sun-mon&wed&fri-sat или более просто: wed&fri-mon.

days_of_month

Это список чисел дней месяца. Дни указываются цифрами от 1 до 31. С 7-го по 12-е число будет выражено как 7-12, а 15-е и 30-е числа месяца будут записаны как 15&30. Это может быть полезно для праздников, которые обычно приходятся на один и тот же день месяца, но не на один и тот же день недели.[6]

months

5 Но мы делаем это так, потому что так легче читать.

6 Мы понятия не имеем, как реализовать Пасху, но открыты для предложений.

Это список из одного или нескольких месяцев в году. Месяцы должны быть записаны как `jan-apr` для диапазона и разделены амперсандами когда требуется включить месяцы не последовательно, как например `jan&mar&jun`. Вы также можете комбинировать их так: `jan-apr&jun&oct-dec`.

Если вы хотите сопоставить все возможные значения для любого из этих аргументов - просто поставьте * в этом аргументе.

Аргумент *`label`* может быть любым из следующих:

- Метка приоритета в пределах одного расширения, например `time_has_passed`
- Расширение и приоритет в одном контексте, например `123,time_has_passed`
- Контекст, расширение и также приоритет, например `incoming,123,time_has_passed`

Теперь, когда мы рассмотрели синтаксис, давайте рассмотрим несколько примеров. Следующий пример будет соответствовать с *9:00 утра до 5:59 вечера*, с *понедельника по пятницу*, в *любой день месяца*, в *любом месяце года*:

```
exten => s,1,NoOp()
  same => n,GotoIfTime(09:00-17:59,mon-fri,*,*?open,s,1)
```

Если абонент звонит в течение этих часов - вызов будет направлен на первый приоритет расширения `start` в контексте с именем `open`. Если вызов выполняется вне указанного времени - он просто продолжит работу со следующего приоритета текущего расширения. Мы собираемся добавить новый контекст с именем `[closed]` сразу после примера соответствия шаблону `55512XX` и изменить контекст `[Test Menu]`, который мы создали в Главе 6, чтобы обработать наше новое правило по времени.

```
exten => _55512XX,1,Answer()
  same => n,Playback(tt-monkeys)
  same => n,Hangup()

exten => *98,1,NoOp(Access voicemail retrieval.)
  same => n,VoiceMailMain()

[closed]
exten => start,1,Noop(after hours handler)
  same => n,Playback(go-away2)
  same => n,Hangup()

[TestMenu]
exten => start,1,Noop(main autoattendant)
  same => n,GotoIfTime(16:59-08:00,mon-fri,*,*?closed,start,1)
  same => n,GotoIfTime(11:59-09:00,sat,*,*?closed,start,1)
  same => n,GotoIfTime(00:00-23:59,sun,*,*?closed,start,1)
  same => n,Background(enter-ext-of-person)
  same => n,WaitExten(5)

exten => 1,1,Dial(${UserA_DeskPhone},10)
  same => n,Playback(vm-nobodyavail)
  same => n,Hangup()
```

GoSub

Приложение диалплана `GoSub()` позволяет отправить вызов в отдельный раздел диалплана, сделать что-то полезное, а затем вернуть вызов в точку в диалплане, откуда он пришел. Вы можете передать аргументы в `GoSub()`, а также получить от него код возврата. Оно значительно увеличивает функциональность вашего диалплана.

Подпрограммы являются важнейшей способностью в любом языке программирования, и в не меньшей степени в диалплане Asterisk. Для тех, кто новичок в программировании: подпрограмма позволяет создать блок универсального кода, который может быть повторно использован различными частями диалплана для избежания повторения. Подумайте о них как о шаблоне в текстовом документе или

пустой форме, и у вас появится представление. Как только вы увидите их в действии - должно стать ясно, насколько полезными они могут быть.

Определение подпрограмм

При использовании `GoSub()` в диалплане нет особых требований к именованию. Фактически, вы можете использовать `GoSub()` в том же контексте и расширении если пожелаете. В большинстве случаев, однако, ваши подпрограммы должны быть написаны в отдельных контекстах: один контекст для каждой подпрограммы. При создании контекста мы рекомендуем добавить к имени `sub`, чтобы знать что контекст вызывается из приложения `GoSub()`.

Давайте рассмотрим очевидный пример того, где подпрограмма была бы полезна.

Как вы могли заметить, при создании нашего примера диалплана для пользователей, которых мы добавили, логика диалплана для каждого пользователя может потребовать несколько строк кода.

```
[sets]
exten => 100,1,Dial(${UserA_DeskPhone},12)
  same => n,Voicemail(100@default)
  same => n,GotoIf($["${DIALSTATUS}" = "BUSY"]?busy:unavail)

  same => n(unavail),VoiceMail(100@default,u)
  same => n,Hangup()

  same => n(busy),VoiceMail(100@default,b)
  same => n,Hangup()

exten => 101,1,Dial(${UserA_SoftPhone})
  same => n,GotoIf($["${DIALSTATUS}" = "BUSY"]?busy:unavail)

  same => n(unavail),VoiceMail(101@default,u)
  same => n,Hangup()

  same => n(busy),VoiceMail(101@default,b)
  same => n,Hangup()

exten => 102,1,Dial(${UserB_DeskPhone},10)
  same => n,Playback(vm-nobodyavail)
  same => n,Hangup()

exten => 103,1,Dial(${UserB_SoftPhone})
  same => n,Hangup()
```

Мы предоставили только двум пользователям реальную, рабочую голосовую почту, и определили только четыре телефона как внутренние номера, и все же у нас уже есть беспорядок в виде повторяющегося кода, который будет все труднее поддерживать и расширять. Это быстро станет неуправляемым, если мы не найдем способа получше.

Давайте напишем подпрограмму для обработки набора номера наших пользователей. Добавьте следующее в самый конец вашего диалплана:

```
; SUBROUTINES
[subDialUser]
exten => _[0-9].,1,Noop(Dial extension ${EXTEN},channel: ${ARG1}, mailbox: ${ARG2})
  same => n,Noop(mboxcontext: ${ARG3}, timeout ${ARG4})
  same => n,Dial(${ARG1},${ARG4})
  same => n,GotoIf($["${DIALSTATUS}" = "BUSY"]?busy:unavail)

  same => n(unavail),VoiceMail(${ARG2}@${ARG3},u)
  same => n,Hangup()

  same => n(busy),VoiceMail(${ARG2}@${ARG3},b)
  same => n,Hangup()
```

Теперь измените верхнюю часть своего диалплана следующим образом:

```
[OLD_sets] ; что было [sets] теперь [OLD_sets] (называйте как угодно, имя изменить недолго)
```

```
exten => 100,1,Dial(${UserA_DeskPhone},12)
  same => n,Voicemail(100@default)
  same => n,GotoIf($["${DIALSTATUS}" = "BUSY"]?busy:unavail)
;(и тд)
```

Мы переименовали наш контекст `[sets]`, который, конечно, сломает наш диалплан, так как наши телефоны входят в диалплан в нем. Итак, мы собираемся снова добавить его немного ниже:

```
exten => 103,1,Dial(${UserB_SoftPhone})
  same => n,Hangup()

[sets]

exten => 110,1,Dial(${UserA_DeskPhone}&${UserA_SoftPhone}&${UserB_SoftPhone})
  same => n,Hangup()
;(etc)
```

Итак, теперь у нас снова есть наш контекст `[sets]`, а также `[OLD_sets]`, в котором есть наш старый, осиротевший код. Как мы набираем наши телефоны? Как эта подпрограмма, которую мы только что написали, поможет нам?

```
exten => 103,1,Dial(${UserB_SoftPhone})
 same => n,Hangup()

[sets]

;subDialUser args:
; - ARG1 канал(ы) для вызова
; - ARG2 почтовый ящик
; - ARG3 контекст почтового ящика
; - ARG4 Тайм-аут
exten => 100,1,Gosub(subDialUser,${EXTEN},1(${UserA_DeskPhone},${EXTEN},default,12))
exten => 101,1,Gosub(subDialUser,${EXTEN},1(${UserA_SoftPhone},${EXTEN},default,3))
exten => 102,1,Gosub(subDialUser,${EXTEN},1(${UserB_DeskPhone},${EXTEN},default,6))
exten => 103,1,Gosub(subDialUser,${EXTEN},1(${UserB_SoftPhone},${EXTEN},default,24))

exten => 110,1,Dial(${UserA_DeskPhone}&${UserA_SoftPhone}&${UserB_SoftPhone})
 same => n,Hangup()
```

Сохраните его, перезагрузите диалплан и выполните несколько тестовых вызовов. Поиграйте с параметрами и посмотрите что изменится. Добавьте несколько почтовых ящиков в свою базу данных и посмотрите что произойдет. Если вы вдохновлены - напишите новую подпрограмму `subDialUserNEW` и посмотрите что сможете придумать. На этом этапе вы также можете удалить весь код в контексте `[OLD_sets]`, поскольку он теперь заброшен, но вы также можете оставить его, поскольку он не причиняет вреда.

Теперь вы можете добавить сотни внутренних номеров и каждый из них будет использовать только одну строку диалплана.

Всякий раз, когда вы обнаружите, что где-то пишете дубликат кода диалплана, остановитесь. Вполне вероятно, что пришло время написать подпрограмму.

Возврат из подпрограммы

Приложение диалплана `GoSub()` не возвращается автоматически после выполнения подпрограммы. Если вы закончили с вызовом, то можете, конечно, использовать `Hangup()`, однако, если вы не хотите отключаться, а скорее вернуть вызов туда, откуда он пришел, вы можете использовать приложение `Return()`.

Поскольку вы можете вложить подпрограмму в подпрограмму, а также выполнять их одну за другой, когда попадаете в более сложные подпрограммы, то вскоре обнаружите, что это весьма полезная возможность.

Локальные (Local) каналы

Каналы Local - это метод выполнения других областей диалплана из приложения `Dial()` (в отличие от отправки вызова из канала). Думайте о них как о подпрограммах, которые вы можете вызвать из `Dial()`.

Они могут показаться немного странной концепцией когда вы впервые начинаете их использовать, но поверьте нам - когда мы говорим вам, что они могут быть ответом на проблему, которую вы не можете решить никаким другим способом. Вы почти наверняка захотите использовать их, когда начнете писать расширенные диалпланы. Лучший способ проиллюстрировать использование локальных каналов - на примере. Предположим, у нас есть ситуация, когда нам нужно позвонить нескольким людям, но нам нужно обеспечить задержки разной длины перед набором каждого из участников. Использование локальных каналов является решением проблемы.

С помощью приложения `Dial()` вы, конечно, можете звонить на несколько конечных точек (см. расширение 110 в вашем диалплане для иллюстрации этого), но все три канала будут звонить одновременно и в течение одного и того же периода времени.

```
exten => 110,1,Dial(${UserA_DeskPhone}&${UserA_SoftPhone}&${UserB_SoftPhone})
  same => n,Hangup()
```

Однако, допустим, мы хотим ввести некоторые задержки до звонка пользователю, а также прекратить звонить в разные места в разное время. Использование локальных каналов дает нам независимое управление над каждым из каналов, которые мы хотим набрать, поэтому мы можем вводить задержки и контролировать период времени, в течение которого каждый канал звонит независимо.

Допустим, у нас есть небольшая компания, где в первую очередь на входящие звонки отвечает администратор, но есть также два участника команды, которым поручено отвечать на вызовы, и, наконец, может помочь владелец, если это необходимо.

Требования таковы:

- Телефон на стойке регистрации должен звонить сразу и продолжать звонить и не останавливаться, пока не ответят.
- Телефоны участников команды не должны звонить в течение первых 9 секунд, после чего они могут звонить, пока не ответят.
- Телефон владельца должен звонить только в том случае, если вызов оставался без ответа в течение 12 секунд. Кроме того, мы притворяемся, что это сотовый телефон, и поэтому должны прекратить звонить через 18 секунд, чтобы на вызов не ответила голосовая почта сотового телефона.

Мы будем использовать наши существующие настроенные каналы чтобы использовать различные функции. Если у вас есть какой-либо способ сделать это, пожалуйста, постарайтесь, чтобы все они были зарегистрированы где-то, чтобы они могли звонить при вызове. Это даст вам гораздо лучшее представление о том, что происходит при тестировании.[7]

Это прекрасное время для подпрограммы:

```
[subDialDelay]
exten => _[a-zA-Z0-9].,1,Noop(channel ${ARG1}, pre-delay ${ARG2}, timeout ${ARG3})
;  same => n,Progress() ; Optional; Signals back that the call is proceeding
  same => n,Wait(${ARG2}) ; how long to wait before dialing
  same => n,Dial(${ARG1},${ARG3}) ; timeout can be blank (infinite)
  same => n,Hangup()
```

У вас уже есть подпрограмма в нижней части файла. Добавьте новую туда же, чтобы все ваши подпрограммы были сгруппированы вместе.

7 Устаревшие телефоны и планшеты на базе Android могут отлично подойти для этого.

Теперь нам нужен контекст, в котором мы будем создавать расширения, которые будут использоваться локальным каналом:

```
;LOCAL CHANNELS
[localDialDelay]
exten => receptionist,1,Gosub(subDialDelay,${EXTEN},1(${UserA_DeskPhone},0,600))
exten => team_one,1,Gosub(subDialDelay,${EXTEN},1(${UserA_SoftPhone},9,600))
exten => team_two,1,Gosub(subDialDelay,${EXTEN},1(${UserB_DeskPhone},9,600))
exten => owner,1,Gosub(subDialDelay,${EXTEN},1(${UserB_SoftPhone},12,18))
```

Несмотря на то, что назначение для локального канала на самом деле является просто диалпланом — так же, как вы могли бы перейти с помощью `Goto()`, эти конструкции, как правило, очень специализированы и вписываются в диалплан лучше в своей собственной области - внизу с подпрограммами. Вот почему мы назвали контекст с префиксом `local`. Это необязательно, но делает вещи легче для понимания.

Теперь мы сшиваем все это вместе в нашем контексте `[sets]`.

Во-первых, давайте предоставим возможность набирать каждый локальный канал индивидуально, чтобы мы могли проверить каждый канал и убедиться что он делает то, что должен.

```
exten => 103,1,Gosub(subDialUser,${EXTEN},1(${UserB_SoftPhone},${EXTEN},default,24))

; Они предназначены для тестирования по отдельности, прежде чем мы соберем их вместе
exten => 104,1,Dial(Local/receptionist@localDialDelay)
exten => 105,1,Dial(Local/team_one@localDialDelay)
exten => 106,1,Dial(Local/team_two@localDialDelay)
exten => 107,1,Dial(Local/owner@localDialDelay)
```

Наконец, давайте доставим готовый продукт.

```
exten => 107,1,Dial(Local/owner@localDialDelay)

; Мы собираемся назначить некоторые переменные,
; чтобы сохранить простоту чтения строки набора
exten => 108,1,Noop(DialDelay)
  same => n,Set(Recpn=Local/receptionist@localDialDelay)
  same => n,Set(Team1=Local/team_one@localDialDelay)
  same => n,Set(Team2=Local/team_two@localDialDelay)
  same => n,Set(Boss=Local/owner@localDialDelay)
  same => n,Dial(${Recpn}&${Team1}&${Team2}&${Boss},600)
```

Вам действительно нужно зарегистрировать несколько телефонов и попробовать, чтобы увидеть все это в работе.

Решение, которое мы создали, идеально подходит для изучения локальных каналов, но у него есть несколько проблем, которые нужно понять, если вы когда-нибудь захотите запустить его в продакшен:

- Несмотря на то, что мы установили тайм-аут набора номера, вы обнаружите, что конечные точки SIP имеют собственное мнение об этом. Это не редкость для конечной точки SIP, чтобы иметь свои собственные идеи о таймауте. Таким образом, вы можете установить его на звонок в течение 600 секунд и задаться вопросом: почему он сбрасывает вызов через минуту или около того. Вы можете потратить часы на устранение неполадок вашего диалплана, только чтобы обнаружить, что проблема была настройкой на другом конце. *Проверьте каждый кусок, прежде чем склеить их все вместе.*
- Сотовые телефоны имеют свою собственную голосовую почту, и если она отвечает на вызов - Asterisk подключит вызов к этому “отвеченному” каналу. Один из способов обойти это - повесить трубку до того, как это произойдет, а затем немедленно перезвонить. Это некрасиво и не рекомендуется.
- Сотовые телефоны часто сразу переходят на голосовую почту, если находятся вне зоны действия сети или выключены. *Это считается ответом для Asterisk.* Данное решение не

справляется с подобной проблемой.

- Настройка вызова на сотовый телефон (т.е. время между моментом набора и началом вызова) обычно занимает около дюжины секунд или около того.
- Помните, что софтфон на мобильном телефоне совсем не похож на телефонный звонок на этот мобильный телефон. Первый - это SIP-соединение, а другой - это вызов ТфОП. (Вы можете звонить одновременно если хотите, но это не всегда хорошая идея.)
- Некоторые типы смартфонов будут отдавать приоритет входящим GSM-вызовам. Если вы отвечаете на вызов по софтфону, и кто-то звонит на ваш номер мобильного телефона, софтфон может быть поставлен на удержание. Разные телефоны справляются с этим по-разному.
- Мы действительно не справились с переполнением здесь. Что будет если *никто* не ответит? Это не имеет значения в лаборатории, но можете быть уверены, что это будет иметь значение в продакшене.
- `Dial()` ожидает ответный звонок из пункта назначения. Если все ваши локальные каналы имеют задержку `Wait()`, вызывающий абонент будет слышать тишину, пока что-то не укажет на звонок. Вы можете исправить это, используя `Dial()` и имитируя сигнал вызова опцией `'r'` или добавив фиктивный локальный канал, который просто возвращает сигнал вызова.

Если вы проверите образец диалплана, мы добавили решение проблемы тишины на задержанных локальных каналах.

Вот и все. Локальные каналы: создавайте их по частям, и вы в кратчайшие сроки предоставите мощный диалплан.

Они невероятно полезны при создании сложных приложений очередей.

Использование базы данных Asterisk

Asterisk предоставляет простой механизм для хранения данных, называемый *Asterisk database* (AstDB). Это не внешняя реляционная база данных, а просто серверная часть на основе SQLite для хранения простых пар ключ/значение.

База данных Asterisk хранит свои данные в группах, называемых *семействами (families)*, со значениями, определяемыми *ключами (keys)*. В семействе ключ может быть использован только один раз. Например, если бы у нас было семейство `test`, мы могли бы хранить только одно значение с ключом `count`. Каждое сохраненное значение должно быть связано с семейством.

Хранение данных в AstDB

Чтобы сохранить новое значение в базе данных Asterisk - мы используем приложение `Set()` с функцией `DB()`. Например, чтобы присвоить ключу `count` в семействе `test` значение `1`, мы напишем следующее:

```
exten => 216,1,NoOp()
  same => n,Set(DB(testkey/count)=1)
```

Сделайте тестовый вызов на 216 чтобы установить значение. Обратите внимание, что если ключ с именем `count` уже существует в семействе `test`, его значение будет перезаписано новым (в этом случае значение жестко закодировано, поэтому оно будет перезаписано с тем же значением, но позже мы увидим, как можем изменить значение и сохранить его).

Вы также можете сохранять значения из командной строки Asterisk, запустив команду `database put` *`family key value`*. Для нашего примера, вы должны ввести `database put test count 1`.

Итак, пока мы это делаем, давайте также добавим значение в базу данных из консоли:

```
*CLI> database put somekey somevalue 42
```

А теперь запросим базу данных из консоли, чтобы увидеть, какие значения там находятся:

```
*CLI> database show
```

Если все хорошо, вы должны увидеть результат, подобный следующему:

```
/pbx/UUID                         : d562019a-d2c4-4b88-bcd9-602b3b46fe07
/somekey/count                    : 1
/somekey/somevalue                : 42
/testkey/count                    : 1
4 results found.
localhost*CLI>
```

Получение данных из AstDB

Чтобы извлечь значение из базы данных Asterisk и присвоить его переменной, мы снова будем использовать приложение `Set()` и функцию `DB()`. Давайте получим значение `somevalue` (из семейства `somekey`), назначим его переменной `THE_ANSWER`, а затем передадим значение вызывающему объекту:

```
exten => 217,1,NoOp()
  same => n,Set(THE_ANSWER=${DB(somekey/somevalue)})
  same => n,Answer()
  same => n,SayNumber(${THE_ANSWER})
```

Вы также можете проверить значение данного ключа из командной строки Asterisk, запустив команду `database get` *`family key`*. Чтобы просмотреть все содержимое базы данных AstDB, используйте команду `database show`.

Удаление данных из AstDB

Существует два способа удаления данных из базы данных Asterisk. Для удаления ключа можно использовать приложение `DB_DELETE()`. Оно принимает путь к ключу в качестве аргументов, например:

```
; удаляет ключ и возвращает его значение за один шаг
exten => 218,1,Verbose(0, We just blew away ${DB_DELETE(somekey/somevalue)})
```

Вы также можете удалить все семейство ключей с помощью приложения `DBdeltree()`. Приложение `DBdeltree()` принимает один аргумент: имя семейства ключей для удаления. Чтобы удалить все семейство `test`, выполните следующие действия:

```
exten => 219,1,DBdeltree(somekey)
```

Чтобы удалить ключи и семейства ключей из базы данных AstDB через интерфейс командной строки, используйте команды `database del` *`key`* и `database deltree` *`family`* соответственно.

Если вы сейчас позвоните по номеру 217, то увидите, что ничего не сказано, потому что база данных ничего не возвращает. Вы также можете запустить `database show` из CLI и отметить, что это семейство и ключ были удалены.

Использование AstDB в диалплане

Существует бесконечное количество способов использования базы данных Asterisk в диалплане. Чтобы представить AstDB - мы рассмотрим два простых примера. Первый - простой пример подсчета показывает, что база данных Asterisk является постоянной (она даже переживает перезагрузку системы). Во втором примере мы будем использовать функцию `BLACKLIST()`, чтобы оценить находится ли номер в черном списке и должен ли он быть заблокирован.

Чтобы начать пример с подсчетом, давайте сначала извлечем число (значение ключа `count`) из базы данных и назначим его переменной с именем `COUNT`. Если ключ не существует - `DB()` вернет значение `NULL` (нет значения). Поэтому мы можем использовать функцию `ISNULL()`, чтобы проверить, было ли

возвращено значение. Если нет - мы инициализируем AstDB с помощью приложения `Set()`, где установим значение в базе данных равным 1. Это произойдет только в том случае, если этой записи базы данных не существует:

```
exten => 220,1,NoOp()
  same => n,Set(COUNT=${DB(test/count)})           ; получаем текущее значение базы данных
  same => n,GotoIf($[${ISNULL(${COUNT})}]?firstcount:saycount) ; есть ли значение?

  same => n(firstcount),Set(DB(test/count)=1)      ; устанавливаем значение 1
  same => n,Goto(saycount)

  same => n(saycount),NoOp()
  same => n,Answer
  same => n,SayNumber(${COUNT})
  same => n,Goto(increment)                        ; не требуется, но хорошая привычка

  same => n(increment),Set(COUNT=$[${COUNT} + 1])  ; увеличение на единицу
  same => n,Set(DB(test/count)=${COUNT})           ; и присвоение нового значения в базе
                                                   ; данных
  same => n,Goto(saycount)                         ; вернемся и повторим снова
```

Проверьте это. Послушайте, как он считает какое-то время, а затем повесьте трубку. Когда вы снова наберете этот номер - отсчет продолжится с того места, где остановился. Значение, сохраненное в базе данных, будет сохраняться даже при перезапуске Asterisk.

В первое время встроенная база данных Asterisk была необходима. Сегодня, однако, она не так часто используется. Она, вероятно, хороша для установки нескольких семафоров здесь и там, но по большей части, если вы хотите хранить данные - используйте один из бэкэндов реляционной базы данных (мы обсудим интеграцию реляционных баз данных в последующих главах).

Полезные функции Asterisk

Теперь, когда мы рассмотрели некоторые из основ, давайте рассмотрим несколько популярных функций, которые были включены в Asterisk.

Концеренц-связь с ConfBridge()

Приложение `ConfBridge()` позволяет нескольким абонентам общаться друг с другом как если бы они все физически находились в одном месте. Некоторые из основных функций включают в себя:

- Возможность создания защищенных паролем конференций
- Администрирование конференции (отключение звука, блокировка или выброс участников)
- Возможность отключение всех, кроме одного участника (полезно для объявлений компании, радиопередач и др.)
- Статическое или динамическое создание конференции
- Звук высокой четкости, который может быть микширован при частоте дискретизации от 8 кГц до 96 кГц
- Видео-возможности, включая добавление динамического переключения видео-каналов на самого громкого докладчика
- Динамически управляемая система меню для администраторов конференций и пользователей
- Дополнительные опции доступны в *confbridge.conf*

В этой главе мы сосредоточены на диалплане - поэтому собираемся продемонстрировать только базовый мост аудиоконференции:

```
$ sudo -u asterisk vim /etc/asterisk/confbridge.conf
[general]

[default_user]
```

```
type=user

[default_bridge]
type=bridge
```

После создания файла *confbridge.conf*, нам нужно загрузить модуль `app_confbridge.so`. Это можно сделать в консоли Asterisk:

```
*CLI> module load app_confbridge.so
```

С загруженным модулем мы можем создать простой диалплан для доступа к нашему конференц-мосту:

```
exten => 221,1,NoOp()
  same => n,ConfBridge(${EXTEN})
```

Это только верхушка айсберга для проведения конференций. Мы сделали базовую конфигурацию, но есть гораздо больше возможностей для настройки. Мы рассмотрим их более подробно в Главе 11.

Полезные функции диалплана

Мы обсуждали функции ранее в этой главе, но у нас есть что сказать ещё. В настоящее время существует около 150 функций, предоставляемых диалпланом Asterisk. Вот небольшой, кураторский список из тех, с которыми стоит поэкспериментировать.

CALLERID()

`CALLERID()` поддерживает множество различных типов данных, но вы обнаружите, что обычно используете одно из `name` или `num`.

```
exten => 222,1,Noop(CALLERID function)
  same => n,Noop(CALLERID currently ${CALLERID(all)})
  same => n,Set(CALLERID(num)=4169671111)
  same => n,Noop(CALLERID now ${CALLERID(all)})
  same => n,Set(CALLERID(name)="Somename")
  same => n,Noop(CALLERID now ${CALLERID(all)})
  same => n,Hangup()
```

Об остальных не беспокойтесь. Если они вам понадобятся - вы будете знать, что они обозначают и почему вы хотите их использовать.

CHANNEL()

`CHANNEL()` позволяет взаимодействовать с загрузкой абсолютных данных, относящихся к каналу. Некоторые элементы позволяют изменять их, в то время как другие будут полезны только для справки (например, `peerip` позволит вам прочитать, но не изменить, IP-адрес узла). Существуют также переменные канала, работающие только с определенными типами каналов (например, элементы `pjsip`, конечно же могут использоваться только на каналах PJSIP).

```
exten => 223,1,Noop(CHANNEL function)
  same => n,Answer()
  same => n,Noop(CHANNEL(name) is ${CHANNEL(name)})
  same => n,Noop(CHANNEL(musicclass) is ${CHANNEL(musicclass)})
  same => n,Noop(CHANNEL(rtcp,all_jitter) is ${CHANNEL(rtcp,all_jitter)})
  same => n,Noop(CHANNEL(rtcp,all_loss) is ${CHANNEL(rtcp,all_loss)})
  same => n,Noop(CHANNEL(rtcp,all_rtt) is ${CHANNEL(rtcp,all_rtt)})
  same => n,Noop(CHANNEL(rtcp,txcount) is ${CHANNEL(rtcp,txcount)})
  same => n,Noop(CHANNEL(rtcp,rxcount) is ${CHANNEL(rtcp,rxcount)})
  same => n,Noop(CHANNEL(pjsip,local_uri) is ${CHANNEL(pjsip,local_uri)})
  same => n,Noop(CHANNEL(pjsip,remote_uri) is ${CHANNEL(pjsip,remote_uri)})
  same => n,Noop(CHANNEL(pjsip,request_uri) is ${CHANNEL(pjsip,request_uri)})
  same => n,Noop(CHANNEL(pjsip,local_tag) is ${CHANNEL(pjsip,local_tag)})
```

CURL()

`CURL()` - это простая, но мощная функция, предоставляющая однострочный метод разрешения URL-адресов, который во многих случаях является всем необходимым для базового взаимодействия с внешним веб-сервисом.

```
exten => 224,1,Noop(CURL function)
  same => n,Set(ExternalIP=${CURL(http://whatismyip.akamai.com)})
  same => n,Noop(The external IP address is ${ExternalIP})
```

Если вам нужно более сложное взаимодействие с внешним сервисом - возможно вам понадобится какая-то программа AGI. Тем не менее, вы можете встроить тонну данных в URL и по простоте `CURL()` трудно превзойти.

CUT()

Если вам нужно нарезать ваши переменные - вы найдете функцию `CUT()` весьма полезной. Форма проста:

```
CUT(varname,char-delim,range-spec)
```

Это может быть визуально сложно, так как символ разделителя может быть трудно увидеть вложенным между двумя запятыми (например, если разделитель был точкой/десятичной дробью). Давайте развернем предыдущий пример, чтобы увидеть, для чего он хорош (и дать вам визуальный пример того, как разделитель может потеряться в синтаксисе).

```
exten => 225,1,Noop(CUT function)
  same => n,Set(ExternalIP=${CURL(http://whatismyip.akamai.com)})
  same => n,Noop(The external IP address is ${ExternalIP})
  same => n,Answer()
  same => n,SayDigits(=${CUT(ExternalIP,.,1)})
  same => n,Playback(letters/dot)
  same => n,SayDigits(=${CUT(ExternalIP,.,2)})
  same => n,Playback(letters/dot)
  same => n,SayDigits(=${CUT(ExternalIP,.,3)})
  same => n,Playback(letters/dot)
  same => n,SayDigits(=${CUT(ExternalIP,.,4)})
```

Обратите внимание, что вы вызываете функцию `CUT()` с фигурными скобками `${CUT()}`, но переменная, на которую ссылаются внутри `CUT()`, определяется без фигурных скобок. Это связано с тем, что мы вызываем переменную, а не запрашиваем ее содержимое (`CUT()` будет иметь дело с содержимым, поэтому нам просто нужно назвать переменную, которую она будет резать на ломтики и кубики и погрузится в то, что там хранится).

IF() и STRFTIME()

Комбинация `IF()` и `STRFTIME()` является мощной конструкцией и вы найдете ее неотъемлемой частью логики своего диалплана:

```
exten => 226,1,Noop(IF)
  same => n,Answer()
  same => n,Playback(${IF($[$[${STRFTIME(,,%S)} % 2] = 1]?hear-odd-noise:good-evening)})
```

Подождите...что?[8]

Давайте разберем это (мы сделаем отступы в коде таким образом, чтобы показать прогрессию вложенных функций и операторов):

8 Существует функция языка C с именем `STRFTIME()`, возвращающая текущее время в виде отформатированной строки. Здесь она работает аналогично. Фактически, часть `format` функции принимает тот же синтаксис, что и функция в C.

```
exten => 227,1,Noop(IF)
  same => n,Answer()
  same => n,Wait(.5)
                same => n,Wait(.5)
              same => n,Noop(${STRFTIME(,,%S)}) ; текущее время - только секунды
            same => n,Noop($[ ${STRFTIME(,,%S)} % 2 ]) ; разделить на 2 – вернуть остаток
    same => n,Noop(${IF($[ $[ ${STRFTIME(,,%S)} % 2 ] = 1 ]?odd:even)})
same => n,Playback(${IF($[ $[ ${STRFTIME(,,%S)} % 2 ] = 1 ]?hear-odd-noise:good-evening)})
```

Функция `IF()` позволяет передавать логику в приложение `Playback()`. Мы фактически говорим "Если это правда, что время в секундах нечетное, проиграть подсказку `hear-odd-noise`, в противном случае - проиграть `good-evening`".

Если мы выстроим код более типичным образом - он будет выглядеть так (обратите внимание, что некоторые необязательные пробелы также были удалены):

```
exten => 228,1,Noop(IF)
  same => n,Answer()
  same => n,Wait(.5)
  same => n,Noop(${STRFTIME(,,%S)})
  same => n,Noop($[${STRFTIME(,,%S)} % 2])
  same => n,Noop(${IF($[$[${STRFTIME(,,%S)} % 2 ] = 1]?odd:even)})
  same => n,Playback(${IF($[$[${STRFTIME(,,%S)} % 2 ] = 1]?hear-odd-noise:good-evening)})
```

Последнюю строку очень трудно понять, если вы не знаете как мы туда попали, но она демонстрирует силу вложенности.

Сначала эти конструкции могут показаться трудными для записи - поэтому разбейте их и выполните построчно, и в конечном итоге они станут проще для пониманпия (и ваш диалплан впоследствии станет более мощным). Играйте с ними.

LEN()

Возможность возвращать длину чего-либо с помощью функции `LEN()` может быть очень удобной.

```
exten => 229,1,Noop(LEN)
 same => n,Set(LengthyString=${RAND(1,2000)})
 same => n,Noop(${LEN(${LengthyString})})
 same => n,Noop(${IF( $[ ${LEN(${LengthyString})} <= 3 ]?tooshort:youcanride)})
```

REGEX()

Да, вы можете использовать регулярные выражения в Asterisk. Это несколько продвинутая тема, не потому, что `REGEX()` является сложной функцией сама по себе, а потому что регулярные выражения являются выражениями сами по себе.

Посмотрите *http://www.regular-expressions.info/* для получения дополнительной информации или возьмите копию книги O'Reilly *Регулярные выражения* от Джеффри Фридла.

Привыкните к использованию других функций в Asterisk, получите некоторый опыт работы с регулярными выражениями, а затем попробуйте `REGEX()`.

STRFTIME()

Мы только что видели функцию `STRFTIME()` в нашем примере `IF()`. Она позволяет возвращать время в различных форматах. В общем, ввод должен быть пустым (что по умолчанию соответствует текущему времени). Вы также можете дать этой функции определенную строку времени Unix и она будет работать с ней.

```
exten => 230,1,Noop(STRFTIME)
  same => n,Noop(${STRFTIME(,,%S)}) ; мы уже видели это раньше
  same => n,Noop(${STRFTIME(,,%B)}) ; месяц
  same => n,Noop(${STRFTIME(,,%H)}) ; часы в 24-часовом формате
  same => n,Noop(${STRFTIME(,,%m)}) ; месяц в десятичном виде
```

```
same => n,Noop(${STRFTIME(,,%M)}) ; минуты
same => n,Noop(${STRFTIME(,,%Y)}) ; год - 4 цифры
same => n,Noop(${STRFTIME(,,%Y-%m-%d %H:%m:%S)}) ; всё в одной строке
```

Вывод

В этой главе мы рассмотрели еще несколько приложений диалплана Asterisk и, надеюсь, мы дали вам еще несколько инструментов, которые вы можете использовать для дальнейших экспериментов при создании собственных диалпланов. Как и в других главах - мы приглашаем вас вернуться и перечитать любые разделы, которые требуют уточнения.

Глава 11. Функции АТС, включая парковку, пейджинг и конференц-связь

Я не верю в ангелов, нет. Но у меня есть крошечный парковочный ангел. Он у меня на приборной панели, и ты его заводишь. Крылья хлопают, и это должно дать вам место для парковки. Это работало до сих пор.
– Билли Коннолли

В этой главе рассматриваются некоторые периферийные функции, общие для бизнес-телефонных сред. Мы кратко рассмотрим файл *features.conf*, а затем посвятим несколько разделов пейджингу и парковке и, наконец, немного поработаем с механизмом конференц-связи Asterisk - `confbridge`.

Во-первых, давайте скопируем файл *features.conf* из каталога установки и рассмотрим его:

```
$ sudo cp ~/src/asterisk-16.<TAB>/configs/samples/features.conf.sample \
/etc/asterisk/features.conf
$ sudo chown asterisk:asterisk /etc/asterisk/features.conf
```

features.conf

Asterisk предоставляет несколько функций, общих для большинства УАТС, многие из которых имеют необязательные параметры. Файл *features.conf* - это место, где вы можете настроить или определить различные параметры объектов в Asterisk.

Функции на основе DTMF

Многие параметры в *features.conf* применяются только при вызовах, совершенных с помощью приложений диалплана `Dial()` или `Queue()`, используя один или несколько параметров `k`, `K`, `H`, `h`, `T`, `t`, `W`, `w`, `X` или `x`. Функции доступны путем передачи DTMF-сигналов (т.е. они не могут быть доступны через SIP-сообщения, а только в аудиоканале через тональные сигналы, запускаемые пользователями, набирающими необходимые цифры на своих клавиатурах).[1]

Трансферы SIP-каналов (например через SIP-телефон) могут быть обработаны с использованием возможностей самого телефона и не будут затронуты файлом `features.conf`.

Раздел [general]

В разделе `[general]` *features.conf*, вы можете определить параметры, которые точно настраивают поведение функции трансфера в Asterisk. Они не имеют ничего общего с тем, как SIP-телефоны обрабатывают трансфер вызовов. Вместо этого вы получаете доступ к этим функциям с помощью DTMF во время вызова (вызов должен быть установлен, поэтому звонящие или выполняющиеся вызовы, не будут иметь доступа к этим функциям).

Пример файла *features.conf.sample* в каталоге *~/asterisk/* содержит подробные сведения о различных параметрах и примеры их установки.

Эти функции не так часто используются как в прошлом, главным образом потому, что многие из этих вещей могут быть обработаны более продвинутыми способами, чем набор DTMF с телефонного аппарата (например, через какую-то внешнюю интеграцию или, если на то пошло, с самого телефона, используя свои собственные внутренние функции трансфера).

1 Да, мы понимаем, что сообщение SIP INFO на самом деле является SIP-сообщением и технически не является частью аудиоканала, но дело в том, что вы не можете использовать кнопку «передача» или «парковка» на своем SIP-телефоне для доступа к этим функциям во время разговора. Вам нужно будет отправить DTMF.

Раздел [featuremap]

Раздел `[featuremap]`, приведенный в Таблице 11-1, позволяет определить определенные последовательности DTMF, запускающие функции на каналах, соединенных с помощью опций в приложениях `Dial()` или `Queue()`. Два варианта, которые вы, скорее всего, будете использовать, это `parkcall` и `automixmon`.

Таблица 11-1. features.conf раздел [featuremap]

Параметр	Значение/ пример	Примечание	Флаги Dial()/Queue()
`blindxfer`	`#1`	Вызывает слепой (неконтролируемый) трансфер	`T, t`
`disconnect`	`*0`	Завершает вызов	`H, h`
`automon`	`*1`	Запускает запись текущего вызова с помощью приложения `Monitor()` (повторное нажатие этой последовательности клавиш останавливает запись)	`W, w`
`atxfer`	`*2`	Выполняет автоматический трансфер	`T, t`
`parkcall`	`#72`	Паркует вызов	`K, k`
`automixmon`	`*3`	Запускает запись текущего вызова с помощью приложения `MixMonitor()` (повторное нажатие этой последовательности клавиш останавливает запись)	`X, x`

Раздел [applicationmap]

Раздел `[applicationmap]` в *features.conf* возможно, является самым изящным, поскольку он позволяет сопоставлять коды DTMF с приложениями диалплана. Вызывающий абонент будет поставлен на удержание, пока приложение не завершит выполнение.

Синтаксис для определения сопоставления приложения выглядит следующим образом (оно должно отображаться в одной строке; разрывы строк недопустимы):[2]

```
Name => DTMF_sequence,ActivateOn[/ActivatedBy],App([Args])[,MOH_Class]
```

То, что вы делаете, заключается в следующем:

1. Присвоение сопоставлению имени, позволяющему включить его в диалплан с помощью переменной канала `DYNAMIC_FEATURES` (подробнее об этом чуть позже).
2. Определение последовательности DTMF, активирующей эту функцию (мы рекомендуем использовать для этого по крайней мере две цифры).
3. Определение того, на каком канале будет активирована функция и (необязательно) какому участнику разрешено активировать её (по умолчанию обоим каналам разрешено использовать/активировать).
4. Задает имя приложения, которое вызовет это сопоставление, и его аргументы.
5. Предоставление дополнительного класса музыки на удержание (MOH) для назначения этой функции (который будет слышать противоположный канал при выполнении приложения). Если вы не определяете какой-либо класс - вызывающий абонент будет слышать просто тишину.

Вот пример сопоставления приложения, которое вызовет скрипт AGI:[3]

```
agi_test => *6,self/callee,AGI(agi-test.agi),default
```

2 В синтаксисе есть некоторая гибкость (подробности можно посмотреть в файле примера), но в нашем примере используется стиль, который мы рекомендуем, поскольку он наиболее соответствует типичному синтаксису диалплана.

3 Мы рассмотрим AGI в Главе 18, но вкратце скрипты AGI - это внешние программы, которые вы можете запускать из диалплана. Удобно? Очень!

Вы можете добавить это в свой файл */etc/asterisk/features.conf* если пожелаете.

Поскольку приложения, вызванные сопоставлением приложений, выполняются вне ядра АТС - вы не можете выполнять приложения, запускающие диалплан (например, `Goto()`, `Macro()`, `Background()` и т.д.). Если вы хотите использовать сопоставление приложений для создания внешних процессов (включая выполнение кода диалплана), то вам нужно будет вызвать внешнее приложение через вызов `AGI()` или приложение `System()`. Дело в том, что если вы хотите выполнить что-то сложное с помощью сопоставления приложений - вам нужно будет это очень тщательно протестировать, так как не все будет работать так, как вам хотелось бы.

Чтобы использовать сопоставление приложений - вы должны объявить его в диалплане, установив переменную `DYNAMIC_FEATURES` где-то перед командой `Dial()`, оединяющей каналы. Используйте модификатор двойного подчеркивания в имени переменной для гарантии того, что сопоставление приложения будет доступно обоим каналам в течение всего жизненного цикла вызова. Давайте добавим его в нашу подпрограмму `subDialUser`, чтобы он был доступен всякий раз, когда любой из наших внутренних номеров вызывает другой:

```
[subDialUser]
exten => _[0-9].,1,Noop(Dial extension ${EXTEN},channel: ${ARG1}, mailbox: ${ARG2})
    same => n,Noop(mboxcontext: ${ARG3}, timeout ${ARG4})
    same => n,Set(__DYNAMIC_FEATURES=agi_test)
    same => n,Dial(${ARG1},${ARG4})
    same => n,GotoIf($["${DIALSTATUS}" = "BUSY"]?busy:unavail)
```

Если вы хотите, чтобы при вызове было доступно более одного сопоставления приложения - вам нужно использовать символ `#` в качестве разделителя между несколькими именами сопоставлений:

```
Set(__DYNAMIC_FEATURES=agi_test#my_other_map)
```

Причина, по которой символ `#` был выбран вместо простой запятой, заключается в том, что более старые версии приложения `Set()` интерпретировали запятую иначе, чем более поздние и синтаксис для сопоставления приложений никогда не обновлялся.

Не забудьте перезагрузить модуль `features` после внесения изменений в файл *features.conf*:

```
*CLI> module reload features
```

Вы можете проверить что ваши изменения произошли через команду CLI `features show`.

Кроме того, поскольку мы представляем здесь скрипт AGI, есть некоторые команды, которые необходимо выполнить, чтобы сделать упомянутый скрипт AGI доступным для Asterisk.

```
$ sudo cp ~/src/asterisk-16.<TAB>/agi/agi-test.agi /var/lib/asterisk/agi-bin/

$ sudo chown asterisk:asterisk /var/lib/asterisk/agi-bin/*

$ sudo chmod 755 /var/lib/asterisk/agi-bin/*
```

Убедитесь, что вы протестировали сопоставление приложения, прежде чем передать его в пользование своим пользователям!

Динамическое создание сопоставления приложения из диалплана

Вы можете создавать сопоставления объектов непосредственно из диалплана, делая динамическое определение объекта (и его DTMF) для каждого канала. Это делается с помощью функций диалплана `FEATURE()` и `FEATUREMAP()`. Допустимые значения для `FEATUREMAP()` включают следующие, устанавливающие или извлекающие последовательность DTMF, используемую для запуска функциональности:

`atxfer`

Трансфер с уведомлением

`blindxfer`

Слепой трансфер

`automon`

Авто `Monitor()` (запись вызовов)

`disconnect`

Разъединение вызова

`parkcall`

Парковка вызова

`automixmon`

Авто `MixMonitor()` (запись вызова)

Функция `FEATUREMAP()` позволяет получить текущую последовательность DTMF для этой функции:

```
exten => 232,1,Noop(Current DTMF for parkcall: ${FEATUREMAP(parkcall)})
```

Или вы можете использовать последовательность DTMF для особой функции на текущем канале:

```
exten => 233,1,NoOp()
    same => n,Set(FEATUREMAP(parkcall)=*9)
    same => n,Noop(DTMF for parkcall now: ${FEATUREMAP(parkcall)})
```

Если вы хотите установить тайм-аут парковки для канала, то можете сделать это с помощью функции `FEATURE()`. Она будет содержать единственный аргумент - `parkingtime`, который является значением в секундах до того, как припаркованный вызов будет возвращен вызывающему абоненту (или месту назначения, в зависимости от того, как вы настроили парковку):

```
exten => 234,1,NoOp()
    same => n,Set(FEATURE(parkingtime)=60)
```

Группировка сопоставлений приложений

Если у вас есть множество объектов, которое необходимо активировать для определенного контекста или расширения, вы можете сгруппировать несколько объектов в группу сопоставлений приложений, чтобы одно назначение переменной `DYNAMIC_FEATURES` назначило все объекты этого сопоставления.

Группировки сопоставлений приложений добавляются в конце файла *features.conf*. Каждой группе присваивается имя, а затем перечисляются соответствующие объекты:

```
[shifteight]
unpauseMonitor => *1    ; пользовательское сопоставление клавиш
pauseMonitor => *2      ; пользовательское сопоставление клавиш
agi_test =>             ; непользовательское сопоставление клавиш
```

Если вы хотите задать пользовательское сопоставление клавиш для объекта в группе сопоставления приложения, просто сопоставьте `=>` с нужным сопоставлением клавиш. Если вы не зададите сопоставление клавиш, то для этого объекта будет

использоваться сопоставление клавиш по умолчанию (как указано в разделе [featuremap]). Независимо от того, хотите ли вы назначить пользовательское сопоставление клавиш или нет, требуется оператор =>.

В диалплане эту группировку сопоставлений приложений можно назначить через приложение Set():

```
same => Set(__DYNAMIC_FEATURES=shifteight) ; используйте двойное подчеркивание,
                                           ; если хотите убедиться, что обе
                                           ; ветви вызова имеют назначенную переменную.
```

Парковка и пейджинг

Хотя эти две функции полностью отделены друг от друга - они так часто используются вместе, что мы могли бы рассматривать их как одну отдельную функцию.

Парковка вызовов позволяет удерживать вызовы, а затем извлекать их из места, отличного от того, на котором им первоначально ответили. Пейджинг использует систему громкой связи, позволяющую отправлять объявления из телефонной системы (например, чтобы сообщить для кого предназначен припаркованный вызов и как его можно получить).

Некоторые компании, возможно, с большими складами, открытыми площадками или сотрудниками, передвигающимися по офису, используют функции пейджинга и парковки своих систем для прямых звонков по офису. В этой главе мы покажем вам, как использовать парковку и пейджинг в традиционных настройках (park’n’page), а также несколько более современных подходов к этим часто используемым функциям.

Парковка вызовов

Парковка позволяет удерживать вызов в системе без привязки к определенному добавочному номеру. Затем вызов может быть извлечен любым, кто знает код парковки для этого вызова. Эта функция часто используется вместе с публичным адресом (PA) или системой «пейджинга». По этой причине её часто называют «park-and-page». Следует отметить, что парковка и пейджинг на самом деле разделены. Мы кратко рассмотрим пейджинг, но сначала давайте поговорим о парковке вызовов.

Давайте возьмем копию файла примера, который будем использовать для настройки парковки вызовов:

```
$ sudo cp ~/src/asterisk-16.<TAB>/configs/samples/res_parking.conf.sample \
/etc/asterisk/res_parking.conf

$ sudo chown asterisk:asterisk /etc/asterisk/res_parking.conf

$ sudo asterisk -rx 'module load res_parking.so'
```

Для парковки вызова в Asterisk Вам необходимо перевести вызывающего абонента на код функции, назначенный для парковки в файле *res_parking.conf* с помощью директивы parkext. По умолчанию это 700:

```
parkext => 700      ; номер, набираемый для парковки (все слоты парковки)
```

Вам нужно подождать завершения перевода пока вы не получите номер слота поиска парковки из системы, или у вас не будет возможности получить вызов. По умолчанию слоты поиска, назначенные с помощью директивы parkpos в *res_parking.conf*, нумеруются от 701 до 720:

```
parkpos => 701-720  ; расширения для парковки вызовов (слоты парковки по умолчанию)
```

После того, как вызов припаркован, любой пользователь системы может получить его, набрав номер слота поиска (parkpos), назначенного этому вызову. Затем вызов будет соединен с каналом, который набрал код поиска.

Существует два распространенных способа определения назначения слотов поиска. Это делается с помощью директивы findlot в файле *res_parking.conf*. Метод по умолчанию (findlot => first) всегда использует слот с наименьшим номером, если он доступен, и назначает коды с более высоким

номером только в случае необходимости. Второй метод (`findlot => next`) будет чередовать слоты поиска при каждой последующей парковке, возвращаясь к первому слоту поиска только после того, как использовался последний. Какой метод вы выберете - будет зависеть от того, насколько загружена ваша парковка. Если вы редко пользуетесь парковкой - лучше всего подойдет `findslot` как `first` по умолчанию (люди будут привыкать к тому, что их припаркованные вызовы всегда находятся в одном и том же слоте). Если вы постоянно используете функцию парковки (например, в автосалоне), гораздо лучше для каждой последующей парковки назначать следующий слот, поскольку вы часто будете парковать более одного вызова одновременно. Ваши пользователи привыкнут внимательно слушать фактический номер парковки (вместо того, чтобы просто набирать 701), и это сведет к минимуму вероятность случайного получения неправильного вызова в занятой системе.

Обработка тайм-аутов припаркованных вызовов с опцией comebacktoorigin

Этот параметр настраивает поведение парковки вызова, когда время ожидания вызова на парковке истекает (см. параметр `parkingtime`). `comebacktoorigin` может иметь одно из двух значений:

`yes` (по умолчанию)

Когда тайм-аут припаркованного вызова превышен - Asterisk попытается отправить вызов обратно узлу, который совершил этот вызов. Если канал больше недоступен для Asterisk - вызывающий абонент будет отключен.

`no`

Эта опция будет использоваться если вы хотите выполнить пользовательские функции диалплана для припаркованных вызовов, превысивших тайм-ауты. Вызывающий абонент будет отправлен в определенную область диалплана, где логика может быть применена для изящной обработки оставшейся части вызова (это может включать в себя простой возврат вызова на другой внутренний номер или выполнение какого-либо вида поиска).

Также может потребоваться учитывать вызовы, когда исходный канал не может обработать возвращенный припаркованный вызов. Если, например, вызов был припаркован каналом, который также является транком для другой системы, не будет достаточно информации, чтобы отправить вызов обратно правильному абоненту в той системе. Действия после тайм-аута будут более сложными, чем при `comebacktoorigin=yes`, которые можно обрабатывать изящно.

"Припаркованные вызовы", с параметром `comebacktoorigin=no` всегда будут отправляться в контекст `parkedcallstimeout`.

Диалплан (и контексты) были подробно рассмотрены в Главе 6.

Внутренний номер, на который они будут отправлены, будет построен на основе имени канала, запарковавшего вызов. Например, если узел SIP-пир с именем `0004F2040808` запарковал этот вызов, добавочный номер будет `SIP_0004F2040808`.

Если это расширение не существует - вместо этого вызов будет отправлен на расширение `s` в контексте `parkedcallstimeout`. Наконец, если расширение `s` в `parkedcallstimeout` не существует - вызов будет отправлен на расширение `s` контекста `default`.

Кроме того, для любых вызовов, где `comebacktoorigin = no` будет добавлено расширение `SIP_0004F2040808`, созданное в контексте `park-dial`. Это расширение будет настроено для `Dial()` на `SIP/0004F2040808`.[4]

Если вы используете парковку - вам также понадобится способ объявить о припаркованных вызовах, чтобы пользователи знали как их получить. В то время как вы можете просто бегать по коридору, крича «Вася, прими звонок на номере 701!», Более профессиональный метод - использовать систему пейджинга (более формально известную как система громкой связи), которую мы сейчас обсудим.

Пейджинг (aka Публичное обращение)

Во многих системах УАТС полезно подключить телефонную систему к какой-либо системе громкой связи. Это включает в себя набор кода функции или добавочного номера, устанавливающего соединение с каким-то общедоступным ресурсом, а затем делающим объявление через телефонную трубку, которое транслируется на все устройства, связанные с этим ресурсом пейджинга (возможно, вы слышали продавца в каком-либо магазине запрашивающего проверку цены по телефону). Как правило, это будет внешняя пейджинговая система, состоящая из усилителя, подключенного к потолочным динамикам; однако, пейджинг через колонки офисных телефонов также популярен (в основном по соображениям стоимости). Если у вас есть бюджет (или существующая система оповещения) - система оповещения обычно лучше, но пейджинг на основе телефонных аппаратов также может хорошо работать во многих случаях. Также возможно иметь комбинацию пейджинга на основе аппаратов и потолочного, где, например, пейджинг на основе аппаратов будет использоваться для офисов, а система оповещения - для склада, коридора, парковки и зоны общего пользования (столовая, приемная и т.д.).

В Asterisk для пейджинга используется приложение `Page()`. Это приложение просто принимает список каналов в качестве аргумента, вызывает все перечисленные каналы одновременно и, когда они отвечают, помещает каждый в конференц-зал. Имея это в виду становится очевидным, что одним из требований работы пейджинга является то, что *каждый канал назначения должен иметь возможность автоматически отвечать на входящее соединение* и передавать вызов на какой-то динамик (другими словами, `Page()` не будет работать, если все телефоны будут просто звонить).

Таким образом, хотя само приложение `Page()` является безболезненным и простым в использовании, заставить все каналы назначения правильно обрабатывать входящий пейджинг будет немного сложнее. Мы вернемся к этому в ближайшее время.

Приложение `Page()` принимает три аргумента: 1) группа каналов, к которым должен быть подключен пейджинг, 2) параметры и 3) время ожидания (таймаут):

```
exten => *724,1,Noop(Page)
  same => n,Set(ChannelsToPage=${UserA_DeskPhone}&${UserA_SoftPhone}}&${UserB_DeskPhone})
  same => n,Page(${ChannelsToPage},i,120)
```

Параметры (описанные в Таблице 11-2) дают вам некоторую гибкость в отношении того, как работает `Page()`, но большая часть конфигурации будет зависеть от того, как целевые устройства обрабатывают входящее соединение. В следующем разделе мы рассмотрим различные способы настройки устройств для получения пейджинга.

Таблица 11-2. Параметры Page()

Параметр	Описание	Объяснение
`d`	Включить полнодуплексный звук	Иногда его называют "пейджингом talkback", использование этого параметра подразумевает, что оборудование, принимающее пейджинг, имеет возможность передавать звук обратно одновременно с получением. Как правило, вы не станете использовать это, если у вас нет в нем потребности.
`i`	Игнорировать попытки перенаправления вызова	Как правило эту опцию включают, потому что переадресованный вызов может направиться куда угодно и это не то место, куда нужно направиться Вашему пейджингу.

4 Мы надеемся вы понимаете, что фактический добавочный номер будет связан с названием канала, который запарковал вызов, и не будет `SIP_0004F2040808` (если Лейф не продаст вам телефон Polycom из своей лаборатории).

Параметр	Описание	Объяснение
q	Не воспроизводить звуковой сигнал для вызывающего абонента (тихий режим)	Как правило не используется, так как для пейджинга следует подавать звуковой сигнал для предупреждения людей о том, что пейджинг вот-вот установится. Однако, если у вас есть внешний усилитель, который обеспечивает свой собственный сигнал, вы можете установить эту опцию.
r	Записать пейджинг в файл	Если в будущем вы намеревались использовать один и тот же пейджинг несколько раз, то можете записать его, а затем использовать позже, вызывая с помощью `Originate()` или используя опцию `A(x)` для `Page()`.
s	Набирать канал только в том случае, если состояние устройства `NOT_INUSE`	Эта опция, вероятно, полезна (и надежна) только для SIP-каналов и даже в этом случае может не работать, если на одной линии одновременно разрешены несколько вызовов (довольно часто встречается на SIP-телефонах). Поэтому не полагайтесь на эту опцию во всех случаях.
A(x)	Воспроизвести объявления x всем участникам	Вы можете использовать ранее записанный файл для воспроизведения через систему пейджинга. Если вы объедините это с `Originate()` и `Record()`, то могли бы реализовать систему отложенного пейджинга.
n	Не воспроизводить объявление звонящему (подразумевается `A(x)`)	По умолчанию система будет воспроизводить звук пейджинга как для вызывающего, так и для вызываемого абонента. Если эта опция включена - звук пейджинга не будет транслироваться вызывающему абоненту (человеку, выполняющему пейджинг).

Из-за того, как работает `Page()` он очень ресурсоемкий. Мы не можем это не подчеркнуть. Внимательно читайте дальше, и мы расскажем, как обеспечить, чтобы пейджинг не вызывал проблем с производительностью в продакшене (что наверняка может случиться, если пейджинг не спроектирован правильно).

Места для отправки Вашего пейджинга

Как мы уже говорили, `Page()` само по себе очень простое. Хитрость в том, как собрать все это вместе. Пейджинги могут быть отправлены на различные виды каналов, и все они требуют различной конфигурации.

Внешний пейджинг

Если в здании установлена система громкой связи, общепринято подключать телефонную систему к внешнему усилителю и отправлять пейджинг через вызов на канал. Лучший способ сделать это - использовать какое-либо устройство FXS (такое как ATA), подключаемое через интерфейс пейджинга, как например Bogen UTI1,[5] который затем подключается к усилителю пейджинга.[6]

Другой популярный способ подключения к пейджинговой системе - подключить выход звуковой карты вашего сервера Asterisk к усилителю пейджинга и отправлять вызовы на канал с именем `Console/DSP`. Нам не нравится этот метод, потому что, хоть он и может показаться недорогим и простым, на практике может занимать много времени. Предполагается, что звуковые драйверы на

5 Bogen UTI1 полезен тем, что он может обрабатывать все входящие и исходящие соединения, что практически гарантирует безболезненность подключения вашей телефонной системы к любому виду внешнего пейджингового оборудования, независимо от его возраста или неясности. Стоимость устройства может быть компенсирована за счет экономии времени благодаря специальному полнофункциональному интерфейсу, напоминающему швейцарский армейский нож, с существующей системой громкой связи (или, в этом отношении, как часть новой системы оповещения).

6 В этой книге мы предполагаем, что внешнее пейджинговое оборудование уже установлено и работает со старой телефонной системой, но ничто не мешает вам установить совершенно новую систему оповещения и подключить ее к вашей системе Asterisk. Возможно, вы чувствуете, что мы подключаем Bogen здесь очень быстро, но это просто потому, что они очень долго занимались телефонной связью. Мы используем их в течение почти 30 лет, но они делают это дольше, поэтому, пока вам удобно собирать все части - вы можете выполнить работу правильно с первого раза.

вашем сервере работают правильно, уровни звука на этом канале нормализуются правильно, на вашем сервере имеется достойная встроенная звуковая карта, хорошее заземление, и ... ну, на наш взгляд, это путь не рекомендуется.[7]

В вашем диалплане пейджинг на внешний усилитель будет выглядеть как простой `Dial()` для устройства, подключенного к пейджинговому оборудованию. Вам необходимо настроить ATA так же, как любой SIP-телефон (через `ps_endpoints`, `ps_auth` и т.д. в базе данных), с именем, похожим на `PagingATA`. Затем вы подключаете ATA к Bogen UTI1 и для пейджинга у вас будет этот код диалплана:

```
exten => *725,1,Verbose(2,Paging to external amplifier)   ; '*' является частью того, что вы
                                                          ; набираете
    same => n,Set(PageDevice=PJSIP/PagingATA)       ; Это, вероятно, относится к [globals]
    same => n,Page(${PageDevice},i,120)
```

Вы можете назвать это устройство как угодно (например, мы часто используем MAC-адрес в качестве имени SIP-устройства), но для всего, что не является телефоном пользователя, может быть полезно использовать имя, которое выделяет его из других устройств.

На рынке также есть много пейджинговых устройств на основе SIP (динамики пейджинга с поддержкой SIP популярны, но как нам кажется, довольно дороги для того, что вы получаете, особенно в большом развертывании).

Аппаратный пейджинг

Пейджинг на основе аппаратов впервые стал популярным в клавишных телефонных системах, где громкоговорители офисных телефонов используются в качестве системы оповещения бедняков. Большинство SIP-телефонов имеют возможность автоматического ответа на вызов по громкой связи, которая выполняет то, что требуется для каждого телефона. В дополнение к этому - необходимо передавать звук более чем в одно устройство одновременно. Asterisk использует встроенный механизм конференц-связи для обработки деталей под капотом. Вы используете приложение `Page()`, чтобы это произошло.

Как и `Dial()`, приложение `Page()` может обрабатывать несколько каналов. Поскольку вы захотите чтобы `Page()` сигнализировало сразу на несколько устройств (возможно, даже на все устройства в вашей системе), вы можете получить длинные строки устройств, которые будут выглядеть примерно так:

```
Page(PJSIP/SET1&PJSIP/SET2&PJSIP/SET3&PJSIP/SET4&PJSIP/SET5&PJSIP/SET6&PJSIP/SET7&...
```

За пределами определенного размера система Asterisk не сможет создать несколько аппаратных пейджингов. Например, в офисе с 200 телефонами использование SIP для пейджинга каждого устройства будет невозможно; трафик и загрузка процессора на вашем сервере Asterisk были бы просто слишком велики. В таких случаях вы должны смотреть либо на многоадресный пейджинг, либо на внешний.

Возможно, самая сложная часть пейджинга для SIP-устройств заключается в том, что вам обычно приходится указывать каждому устройству, что оно должно отвечать на вызов автоматически, но разные производители SIP-телефонов используют разные SIP-сообщения для этой цели. Таким образом, в зависимости от используемой вами модели телефона команды, необходимые для выполнения пейджинговой связи на основе SIP, будут разными. Вот некоторые примеры:

- Для Mitel (ранее известны как Aastra):

```
exten => *726,1,Verbose(2,Paging to Aastra sets)
    same => n,SIPAddHeader(Alert-Info: info=alert-autoanswer)
    same => n,Set(PageDevice=SIP/00085D000000)
```

7 Если вам это интересно - мы хотим предложить попробовать это в своей лаборатории. Это может оказаться очень полезным - может сэкономить некоторые расходы на оборудовании. Мы только что обнаружили, что аппаратное обеспечение дешевле чем трудозатраты, поэтому предпочли бы потерять пару сотен долларов на заведомо исправном оборудовании, а не на том, чтобы какой-то плохой техник копался на месте в течение восьми часов, а расстроенный клиент требовал знать, когда возникшая проблема с пейджингом будет решена.

```
    same => n,Page(${PageDevice},i)
```

- Для Polycom:

```
exten => *727,1,Verbose(2,Paging to Polycom sets)
    same => n,SIPAddHeader(Alert-Info: Ring Answer)
    same => n,Set(PageDevice=SIP/0004F2000000)
    same => n,Page(${PageDevice},i)
```

- Для Snom:

```
exten => *728,1,Verbose(2,Paging to Snom sets)
    same => n,Set(VXML_URL=intercom=true)
; замените 'domain.com' на домен вашей системы
    same => n,SIPAddHeader(Call-Info: sip:domain.com\;answer-after=0)
    same => n,Set(PageDevice=SIP/000413000000)
    same => n,Page(${PageDevice},i)
```

- Для Cisco SPA (бывшие телефоные Linksys, кроме серии 79XX):

```
exten => *729,1,Verbose(2,Paging to Cisco SPA sets, but not Cisco 79XX sets)
    same => n,SIPAddHeader(Call-Info:\;answer-after=0)    ; Телефоны Cisco SPA
    same => n,Set(PageDevice=SIP/0004F2000000)
    same => n,Page(${PageDevice},i)
```

Полагаем вы поняли, что произойдет, если у вас такое сочетание телефонов в рабочей среде? Как вы контролируете, какие заголовки отправлять на определенные телефоны?[8]

В любом случае, это некрасиво.

К счастью, многие из этих устройств поддерживают многоадресную IP-рассылку, что является гораздо лучшим способом отправки пейджинга нескольким устройствам (для подробностей читайте далее). Тем не менее, если в вашей системе всего несколько телефонов одного производителя - пейджинг на основе SIP может быть самым простым способом, поэтому мы не хотим вас напугать.

Многоадресный пейджинг через канал MulticastRTP

Если вы серьезно относитесь к пейджингу через аппараты в вашей системе, и у вас есть более чем горсть телефонов, вам нужно будет посмотреть на использование IP-многоадресной рассылки. Концепция IP multicast существует уже давно,[9] но он не получил широкого распространения. Тем не менее - он идеально подходит для пейджинга в пределах одного места.

Asterisk имеет канал (`chan_multicast_rtp`), предназначенный для создания многоадресной рассылки RTP. Этот поток затем подписывается на различные телефоны, и в результате каждый раз, когда медиапоток появляется в многоадресном потоке, телефоны передают его на свои динамики.

Поскольку `MulticastRTP` является драйвером канала - он не имеет приложения, но вместо этого будет работать в любом месте диалплана, где вы могли бы использовать канал иначе. В нашем случае мы будем использовать приложение `Page()` для инициирования нашей многоадресной рассылки.

Чтобы использовать многоадресный канал, вы просто посылаете ему вызов так же, как и любому другому каналу. Синтаксис канала выглядит следующим образом:

`MulticastRTP/`*`type`*`/`*`ip-address:port`*`[/`*`linksys address:port`*`]`

Тип может быть либо `basic`, либо `linksys`. Основной синтаксис канала `MulticastRTP` выглядит следующим образом:

```
exten => *730,1,Page(MulticastRTP/basic/239.0.0.1:1234)
```

8 Подсказка: локальный канал здесь будет вашим другом.

9 Он даже имеет свое собственное пространство зарезервированных IP-адресов класса D, от 224.0.0.0 до 239.255.255.255 (но прочитайте про IP-multicast прежде чем просто взять один из них и назначить его). Часть этого адресного пространства являются частным, часть - общедоступной и некоторые предназначены для целей, отличных от тех, для которых вы желаете их использовать. Информацию о многоадресной адресации можно найти на странице Википедии.

Не все устройства поддерживают IP multicast, но мы протестировали его на Snom,[10] Linksys/Cisco, Polycom(прошивка 4.x или новее) и Aastra и он работает очень хорошо.

Многоадресный пейджинг на телефонах Cisco SPA

Функция многоадресного пейджинга на телефонах Cisco SPA немного странная, но после настройки она работает отлично. Хитрость заключается в том, что адрес, который вы вводите в телефон, не является адресом групповой рассылки, по которому передается пейджинг, а скорее своего рода сигнальным каналом.

Мы обнаружили, что вы можете сделать этот адрес таким же, как адрес многоадресной рассылки, но просто использовать другой номер порта.

Диалплан выглядит так:

```
exten => *724,1,Page(MulticastRTP/linksys/239.0.0.1:1234/239.0.0.1:6061)
```

В телефоне SPA вам нужно войти в интерфейс администрирования и перейти на вкладку *SIP*. В самом низу страницы вы найдете раздел под названием *Linksys Key System Parameters*. Вам необходимо задать следующие параметры:

- `Linksys Key System: Yes`
- `Multicast Address: 239.0.0.1:6061`

Обратите внимание, что адрес многоадресной рассылки, назначенный телефону, является вторым в определении канала (в нашем примере используется порт 6061).

Обратите внимание, что вы можете написать команду `Page()` в этом формате в среде, где есть сочетание телефонов SPA (ранее Linksys, теперь Cisco) и других типов. Другие телефоны будут использовать первый адрес и будут работать так же, как если бы вы использовали `basic` вместо `linksys`.

SIP-адаптеры для пейджинга

На рынке существует множество пейджинговых динамиков на основе SIP. Эти устройства адресуются в диалплане точно так же, как SIP ATA, подключенный к UTI1 (другими словами для системы это просто телефонный аппарат), но физически они похожи на внешние пейджинговые динамики. Поскольку они отвечают автоматически - как правило нет необходимости передавать им какую-либо дополнительную информацию, как при использовании SIP-телефона.

Для небольших установок (где требуется не более полудюжины колонок), эти устройства могут быть экономически эффективными, поскольку не требуют никакого другого оборудования. Однако, для чего-либо большего (или для установки в сложной среде, такой как склад или автостоянка), вы получите лучшую производительность при гораздо меньших затратах с традиционной аналоговой системой оповещения, подключенной к телефонной системе через аналоговый (FXS) интерфейс.

У нас не было опыта работы с этими типами устройств, но есть надежда, что они будут поддерживать многоадресную передачу в качестве стандарта. Имейте это в виду, если планируете использовать их в большом количестве. Обычно лучше заказать одно устройство, протестировать его в прототипной конфигурации, и лишь когда вы убедитесь что оно делает то, что вам нужно, расчитать количество.

Комбинации пейджинга

Во многих организациях может потребоваться как пейджинг на основе аппаратов, так и внешний пейджинг. Например, производственному объекту может потребоваться использовать аппаратный пейджинг для офиса, но потолочный для завода и склада. С точки зрения Asterisk - это довольно

10 Очень громко, и нет возможности отрегулировать усиление.

просто сделать. Когда вы вызываете приложение `Page()`, то просто указываете различные ресурсы, на которые хотите отправить пейджинг, разделенные символом `&` и все они будут включены в конференцию, которую создает приложение `Page()`.

Соберем все это вместе

На данный момент у вас должен быть список различных типов каналов, на которые вы хотите отправить пейджинг. Поскольку `Page()` почти всегда будет сигнализировать о более чем одном канале - мы рекомендуем установить глобальную переменную в разделе `[globals]` вашего файла *extensions.conf*, определяющую список включаемых каналов, а затем вызвать приложение `Page()` с этой строкой:

```
[globals]
MULTICAST=MulticastRTP/linksys/239.0.0.1:1234
;MULTICAST=MulticastRTP/linksys/239.0.0.1:1234/239.0.0.1:6061 ; если у Вас есть телефоны SPA

BOGEN=PJSIP/ATAforPaging  ; Assumes an ATA named [ATAforPaging]
PAGELIST=${MULTICAST}&${BOGEN} ; Имена переменных произвольны.
;...

[sets]
; ...
exten => *731,1,Page(${PAGELIST},i,120)
```

Этот пример предлагает несколько возможных конфигураций, в зависимости от оборудования. Хотя строго не требуется, чтобы была определена переменная `PAGELIST` - мы обнаружили, что это будет иметь тенденцию упрощать управление несколькими ресурсами пейджинга, особенно во время процесса настройки и тестирования.

Зоны пейджинга

Зонирование пейджинга популярно в таких местах, как автомобильные дилерские центры, где отдел запчастей, отдел продаж и, возможно, отдел подержанных автомобилей требуют пейджинга, но не хотят (или не должны) слышать пейджинги друг друга.

В зонировании пейджинга пользователь, отправляющий пейджинг, должен выбрать, в какую зону будет помещен пейджинг. Контроллер зоны пейджинга, такой как Bogen PCM2000, обычно используется для обеспечения сигнализации различных зон: приложение `Page()` сигнализирует контроллеру зоны, контроллер зоны отвечает, а затем отправляется дополнительная цифра для выбора зоны, в которую должен быть отправлен пейджинг. Большинство контроллеров зон позволяют создавать пейджинги для всех зон, в дополнение к объединению зон (например, пейджинги для отделов продаж новых и подержанных автомобилей).

Вы также можете иметь отдельные расширения в диалплане, которые будут разделять ATA (или группы телефонов), но это может оказаться более сложным и дорогостоящим, чем просто покупка контроллера пейджинга, предназначенного для обработки этого. Зонирование пейджинга не требует каких-либо существенно отличающихся технологий, но оно требует немного больше размышлений и планирования в отношении как диалплана, так и аппаратного обеспечения.

И это парковка и пейджинг. Это тонна информации, которую нужно переварить, но как только вы ее освоите - все покажется вполне логичным.

Продвинутая конферец-связь

Приложение `ConfBridge()` - это продвинутое приложение для конференц-связи в Asterisk, обеспечивающее передачу звука высокой четкости и базовых видеоконференций. Ранее мы ввели базовую рабочую настройку для `ConfBridge()`. Если вы создаете свой диалплан во время чтения, то найдете базовый мост конференции в файле *extensions.conf*, выглядящий примерно так:

```
exten => 221,1,NoOp()
```

```
same => n,ConfBridge(${EXTEN})
```

В традиционной конфигурации Asterisk будет файл *confbridge.conf*, в котором мы можем настроить параметры для применения в различных сценариях. Это все еще возможно, но больше не имеет смысла делать это таким образом. Итак, мы собираемся пропустить сразу весь файл конфигурации, за исключением того, чтобы сказать, что файл примера (находится по адресу *~/src/asterisk-15./configs/samples/confbridge.conf.sample*) теперь становится отличным справочным документом, но не более того. Продолжайте читать и поймёте.

Прежде всего, нам нужно объяснить, что для конференции необходимо настроить три типа элементов, а именно `bridge`, `menu` и `user`.

Тип `bridge` определяет сами конференц-залы, тип `menu` определяет меню, к которому можно получить доступ из конференций, а тип `user` позволяет различным участникам конференции применять к ним определенную конфигурацию. Например, большая телеконференция может иметь докладчика (который будет вести большую часть разговора), администратора (для помощи докладчику) и десятки участников (которым может быть запрещено выступать).

Давайте создадим подпрограмму для начала:

```
[subConference]
exten => _[0-9].,1,Noop(Creating conference room for ${EXTEN})
  same => n,Goto(${ARG1})
  same => n,Noop(INVALID ARGUMENT ARG1: ${ARG1})

  same => n(admin),Noop()
  same => n,Authenticate(${ARG2}) ; Можно также использовать
                                  ; Set(CONFBRIDGE(user,pin)=${ARG2})
  same => n,Set(ConfNum=$[${EXTEN} - 1]) ; Hack: вычтите 1, чтобы получить номер конференции
  same => n,Set(CONFBRIDGE(bridge,record_conference)=yes) ; Запись, когда прибыл админ
  same => n,Set(RecordingFileName=${ConfNum}-${STRFTIME(,,%Y-%m-%d %H:%m:%S)})
  same => n,Set(CONFBRIDGE(bridge,record_file)=${RecordingFileName}) ; уникальное имя
  same => n,Set(CONFBRIDGE(user,admin)=yes) ; Админ
  same => n,Set(CONFBRIDGE(user,marked)=yes) ; Маркировать этого пользователя
  same => n,Set(CONFBRIDGE(menu,7)=decrease_talking_volume) ; Уменьшить громкость
  same => n,Set(CONFBRIDGE(menu,9)=increase_talking_volume) ; Увеличить громкость
  same => n,Set(CONFBRIDGE(menu,4)=set_as_single_video_src) ; Блокировать видео на меня
  same => n,Set(CONFBRIDGE(menu,5)=release_as_single_video_src) ; Вернуться к докладчику
  same => n,Set(CONFBRIDGE(menu,6)=admin_toggle_mute_participants); Отключить звук у всех,
                                                                  ; кроме администраторов
  same => n,Set(CONFBRIDGE(menu,2)=participant_count) ; Сколько участников?
  same => n,ConfBridge(${ConfNum})
  same => n,Return()

  same => n(participant),Noop()
  same => n,Set(ConfNum=${EXTEN})
  same => n,Set(CONFBRIDGE(user,wait_marked)=yes) ; Ждите маркированного пользователя
  same => n,Set(CONFBRIDGE(user,announce_only_user)=no) ; Ждите маркированного пользователя
  same => n,Set(CONFBRIDGE(user,music_on_hold_when_empty)=yes) ; Ждите маркированного
                                                               ; пользователя
  same => n,Set(CONFBRIDGE(menu,7)=decrease_talking_volume) ; Уменьшить громкость
  same => n,Set(CONFBRIDGE(menu,9)=increase_talking_volume) ; Увеличить громкость
  same => n,ConfBridge(${ConfNum})
  same => n,Return()
```

Мы можем установить параметры `bridge`, `user` и `menu` как в предыдущем примере. Все параметры, которые вы можете использовать, описаны в файле *~/src/asterisk-16.*/configs/samples/confbridge.conf.sample*.

Когда мы вызываем подпрограмму, то можем передать пользователя в качестве аргумента. Поместите следующий новый код в ваш контекст `[sets]` после `_55512XX` и до `*724`:

```
exten => _55512XX,1,Answer()
 same => n,Playback(tt-monkeys)

; ConfBridge
exten => *600,1,GoSub(subConference,${EXTEN:1},1(participant)) ;
```

```
exten => *601,1,GoSub(subConference,${EXTEN:1},1(admin,4242)) ;

exten => *724,1,Noop(Page)
 same => n,Set(ChannelsToPage=${UserA_DeskPhone}&${UserA_SoftPhone}&${UserB_DeskPhone})
 same => n,Page(${ChannelsToPage},i,120)
```

Если вы наберете *600, то станете участником. Если наберете *601 - вас попросят ввести PIN-код (4242) и вы присоединитесь как администратор. Мы использовали метки диалплана для управления потоком вызовов в подпрограмме. Его легко читать и легко администрировать.

В Главе 15 мы узнаем, как использовать внешнюю базу данных для хранения и получения этих параметров, а не жестко кодировать их в диалплане.

Видео-конференцсвязь

Механизм конференций в Asterisk может обрабатывать видео, но это очень упрощенное предложение, и вы должны тщательно оценить его, чтобы убедиться что оно соответствует вашим потребностям. Некоторые из более серьезных ограничений включают в себя:

- Все участники видеоконференции должны использовать один и тот же видеокодек; в Asterisk нет возможности перекодирования видео.
- В Asterisk отсутствует мультиплексирование видео; одновременно участнику может быть показан только один источник видео.

Чтобы пользователь мог использовать видео (будь то в конференции или просто для обычных вызовов) - ему необходимо включить видеокодеки. Это можно сделать, изменив поле `allow` в таблице `asterisk.endpoints` и добавив `'h264, vp8'` в поле `allow`. Убедитесь, что вы не удаляете кодеки, которые уже есть (например, аудиокодек `ulaw`). Функциональная запись в этом поле может выглядеть следующим образом:

```
ulaw,h264,vp8
```

Прежде чем пытаться использовать видео с вашими конференциями убедитесь, что ваши устройства могут использовать его с прямыми вызовами. Если вы можете использовать видеоконференции между вашими устройствами, вполне вероятно, что это также будет работать в ваших конференц-залах.

В Главе 20 мы погрузимся в WebRTC, где исследуем более мощные концепции в предоставлении мультимедийной коммуникации, включая конференц-связь.

Вывод

В этой главе мы рассмотрели файл *features.conf*, содержащий функциональные возможности для включения трансфера через набор DTMF, записи звонков во время разговора и настройки парковок для одной или нескольких компаний. Мы также рассмотрели различные способы объявления вызовов и информации людям в офисе с использованием множества методов пейджинга, в том числе традиционных систем служебного оповещения и многоадресной пейджинговой связи на телефонные аппараты на рабочих столах сотрудников. После этого мы углубились в приложение `ConfBridge()`, которое чрезвычайно гибко в настройке и имеет множество доступных функций. Это исследование различных методов реализации традиционных функций парковки, пейджинга и конференций современным способом, надеюсь покажет вам гибкость, которую может предложить Asterisk.

Глава 12. Очереди автоматического распределения вызовов

Англичанин, даже если он один, формирует упорядоченную очередь из одного человека.
– Джордж Майкс

Автоматическое распределение вызовов (ACD) или организация очереди вызовов позволяет УАТС ставить в очередь входящие вызовы от нескольких пользователей. Оно объединяет несколько вызовов в шаблон удержания, присваивает каждому вызову рейтинг и определяет порядок, в котором этот вызов должен быть доставлен доступному оператору (как правило, в порядке очереди). Когда агент становится доступным, вызывающий абонент с самым высоким рейтингом в очереди доставляется этому агенту, а все остальные повышаются в рейтинге.

Если вы когда-либо звонили в организацию и слышали, что «все наши операторы заняты», вы испытали ACD. Преимущество ACD для вызывающих абонентов в том, что им не нужно продолжать набирать номер в попытке связаться с кем-то, а для организаций преимущества заключаются в том, что они могут лучше обслуживать своих клиентов и решать проблемы, когда звонящих больше, чем агентов.[1]

Существует два типа колл-центров: входящие и исходящие. ACD относится к технологии, которая обрабатывает входящие вызовы, тогда как термин *Dialer* (или *Predictive Dialer*) относится к технологии, обрабатывающей центры обработки исходящих вызовов. В этой книге мы прежде всего сосредоточимся на входящих звонках.

Мы все были разочарованы плохо спроектированными и управляемыми очередями: длительное удержание, радио вместо мелодии, ошеломляющее время ожидания и бессмысленные сообщения, которые каждые 20 секунд сообщают вам, насколько важен ваш звонок, несмотря на то, что вы ждали 30 минут и прослушали это сообщение так много раз, что можете процитировать его по памяти. С точки зрения обслуживания клиентов - проектирование очереди может быть одним из наиболее важных аспектов вашей телефонной системы. Как и в случае с автосекретарем - прежде всего следует помнить, что *ваши абоненты не заинтересованы в том, чтобы ожидать в очереди*. Они позвонили потому что *хотят поговорить с Вами*. Все ваши проектные решения должны помнить об этом важном факте: люди хотят общаться с другими людьми, а не с Вашей телефонной системой.[2]

Цель этой главы - научить вас создавать и проектировать очереди, доставляющие абонентов по назначению максимально быстро и безболезненно.

В этой главе мы можем переключаться между использованием терминов *участники очереди* и *агенты*. Так как мы не собираемся тратить много времени на модуль Asterisk с именем `chan_agent` (используя `AgentLogin()`), нам нужно прояснить, что в этой книге, когда мы используем термин *agent* - имеется в виду конечный пользователь - человек, а не канальная технология в Asterisk с именем `chan_agent`. Читайте дальше, и это обретёт больше смысла.

1 Это распространенное заблуждение, что очередь может позволить вам обрабатывать больше вызовов. Это не совсем верно: ваши абоненты все равно захотят поговорить с живым человеком, и они будут готовы ждать настолько долго. Другими словами, если у вас мало сотрудников - ваша очередь может оказаться не более чем препятствием для ваших абонентов. Это то же самое, говорите ли вы по телефону или на кассе Walmart. Никто не любит ждать в очереди. Идеальная очередь невидима для звонящих, так как на их звонки отвечают сразу, без ожидания.

2 Существует несколько книг, в которых обсуждаются метрики колл-центра и доступные стратегии организации очередей, например, «Руководство по метрикам колл-центра» Джеймса Эббота (Роберт Хьюстон Смит).

Создание простой очереди ACD

Для начала мы собираемся создать простую очередь ACD. Она будет принимать звонящих и пытаться доставить их участнику очереди.

В Asterisk термин *участник* относится к каналу (обычно одноранговому узлу SIP), назначенному очереди, который можно набрать, например, `SIP/0000FFFF0001`. *Агент* технически относится к каналу агента, также используемому для набора конечных точек. К сожалению, канал агента является устаревшей технологией в Asterisk, так как он ограничен в гибкости и может вызвать непредвиденные проблемы, которые трудно диагностировать и разрешать. Мы не будем охватывать использование `chan_agent` - поэтому имейте в виду, что мы будем использовать термин *member (участник)* для обозначения телефонного устройства и *agent (агент)* для обозначения лица, обрабатывающего вызов. Поскольку один из них, как правило, не эффективен без другого - любой термин может относиться к обоим.

Мы создадим очередь(и) в файле *queues.conf* и добавим в нее участников через консоль Asterisk. В разделе “Участники очереди” мы рассмотрим, как создать диалплан, позволяющий нам динамически добавлять и удалять участников очереди (а также приостанавливать и возобновлять их).

Первым шагом является создание пустого файла *agents.conf* в вашем каталоге конфигурации - */etc/asterisk*. Мы не будем использовать или редактировать этот файл, но модуль `app_queue` ожидает его нахождения и не будет загружаться, если файл не существует:

```
$ cd /etc/asterisk

$ sudo -u asterisk touch agents.conf
```

Поскольку мы еще не сделали этого - мы также собираемся настроить базовую музыку для режима ожидания (MOH), используя файл примера:

```
$ sudo cp ~/src/asterisk-16.*/configs/samples/musiconhold.conf.sample \
/etc/asterisk/musiconhold.conf

$ sudo chown asterisk:asterisk /etc/asterisk/musiconhold.conf
```

Затем вам нужно создать файл *queues.conf*, но мы не будем его редактировать, потому что мы будем создавать наши очереди в базе данных (файл просто должен быть там):

```
$ sudo touch -u asterisk queues.conf
```

Далее мы создадим несколько очередей в нашей базе данных:

```
MySQL>; INSERT INTO `asterisk`.`queues`
(name,strategy,joinempty,leavewhenempty,ringinuse,autofill,musiconhold, \
monitor_format,monitor_type)

VALUES
'sales','rrmemory','unavailable,invalid,unknown','unavailable,invalid,unknown','no','yes',\
'default','wav','MixMonitor'),
('support','rrmemory','unavailable,invalid,unknown','unavailable,invalid,unknown','no',\
'yes','default','wav','MixMonitor') ;
```

Это даст нам две очереди с названиями `sales` и `support`. Вы можете называть их как угодно, но мы будем использовать эти имена далее в этой книге, поэтому - если вы используете другие имена очередей - запомните или запишите ваши названия для дальнейшего использования.

Мы также определили параметры очередей, перечисленные в Таблице 12-1.

Таблица 12-1. Примерные параметры очереди

Параметр	Назначение
`strategy=rrmemory`	Использование стратегии кругового перебора с памятью
`joinempty=unavailable,invalid,unknown`	Не присоединяться к очереди если нет доступных

Параметр	Назначение
	участников
`leavewhenempty=unavailable,invalid,unknown`	Покинуть очередь когда нет доступных участников
`ringinuse=no`	Не звонить участникам, когда они уже используются (предотвращает многократные звонки участникам)
`autofill=yes`	Распределить всех ожидающих абонентов среди доступных участников
`musiconhold=default`	Воспроизведение музыки из класса `[default]` (см. *musiconhold.conf*)

`strategy`, которую мы будем использовать - это `rrmemory`, что означает круговой перебор с памятью. Стратегия `rrmemory` работает путем чередования агентов в очереди в последовательном порядке, отслеживая, какой агент получил последний вызов и предоставляя следующий вызов следующему агенту. Когда он попадает к последнему агенту - очередь возвращается к началу (при входе агентов они добавляются в конец списка).

Несколько примечаний по стратегиям

`ringall`

Звонят все доступные участники (по умолчанию). Эта стратегия распределения на самом деле не считается ACD. В традиционных терминах телефонии это называется "групповой вызов" (`ring group`).

`leastrecent`

Каждый следующий звонок будет получать участник, который в последний раз положил трубку раньше всех остальных. В очереди, где есть много вызовов примерно одинаковой продолжительности, она справедлива. Но не будет справеливым, если агент был на вызове в течение часа, а все его коллеги получили последний звонок 30 минут назад, потому что агент, который закончил последним свой 60-минутный вызов получит следующий звонок.

`fewestcalls`

Вызывается первый свободный участник, который обработал наименьшее количество вызовов из данной очереди. Это может быть несправедливо, если звонки не всегда имеют одинаковую продолжительность. Агент мог обрабатывать три звонка по 15 минут каждый, а его коллега имел четыре 5-секундных звонка; агент, который обработал три звонка, получит следующий звонок.

`random`

Звонит случайный интерфейс. Эта стратегия на самом деле может быть хороша и в конечном итоге будет очень справедлива с точки зрения равномерного распределения вызовов между агентами.

`rrmemory`

Обзванивает участников по кругу, запоминается последний участник, ответивший на вызов. Это также может быть справедливым, но не так как `random`.

`linear`

Участники вызваются в указанном порядке, всегда начиная с начала списка. Это работает, если у вас есть команда, в которой есть некоторые агенты, которые должны обрабатывать большинство вызовов, и другие агенты, которые должны получать вызовы только если основные агенты заняты.

`wrandom`

Вызывается случайный участник, но используется пенальти `penalty` (ударение на первую букву "e") участников в качестве веса `weight`. Стоит рассмотреть в очередях с большой нагрузкой среди агентов.

Мы установили `joinempty` на `no`, так как ставить абонентов в очередь, где нет доступных агентов чтобы принимать их звонки, это плохо.

Вы можете установить его в значение `yes` для удобства тестирования, но мы не рекомендуем запускать его в производство, если вы не используете очередь для какой-либо функции, не предназначенной для передачи абонентов вашим агентам. Никто не хочет ждать в очереди, которая никуда не ведет.

Опция `leavewhenempty` используется для управления тем, должны ли абоненты выпадать из приложения `Queue()` и продолжать работу в диалплане, если ни один из участников не может принимать их вызовы. Мы установили это значение на `yes`, потому что обычно мы не хотим чтобы абоненты ждали в очереди без зарегистрированных агентов.

С точки зрения бизнеса - вы должны сказать своим агентам, чтобы они завершали все звонки в очереди, прежде чем выходить из системы в течение дня. Если вы обнаружите, что в конце дня в очереди много вызовов - возможно, вы решите продлить чью-то смену, чтобы обслужить их. В противном случае - они просто добавят вам стресса, когда перезвонят на следующий день в худшем настроении. Вы можете использовать `GotoIfTime()` ближе к концу дня, чтобы перенаправить абонентов на голосовую почту или другое подходящее место в вашем диалплане, пока ваши агенты завершают все оставшиеся вызовы в очереди.

Мы выставим `ringinuse` на `no`, что говорит Asterisk не звонить участникам, когда их устройства уже используются. Целью установки `ringinuse` в `no` является предотвращение многократных вызовов одного и того же участника из одной или нескольких очередей.

Следует отметить, что упомянутые `joinempty` и `leftwhenempty` ищут либо участников, не вошедших в очередь, либо недоступных. Агенты в состоянии `Ringing` или `InUse` не считаются недоступными и поэтому не будут блокировать абонентов от присоединения к очереди и заставлять их отключаться при `joinempty=no` и/или `leftwhenempty=yes`.

Опция `autofill` указывает очереди немедленно распределять всех ожидающих абонентов между всеми доступными участниками. Предыдущие версии Asterisk распределяли только одного абонента за один раз - это означало, что в то время как Asterisk подавал сигнал агенту, все остальные вызовы удерживались (даже если другие агенты были доступны) до тех пор, пока первый абонент в очереди не был подключен к агенту (что, очевидно, приводило к узким местам в тех версиях Asterisk, где использовались сильно загруженные очереди). Если у вас нет особой потребности в обратной совместимости - *этот параметр всегда* должен быть установлен в `yes`.

Убедитесь, что ваш файл */etc/asterisk/extconfig.conf* содержит следующие строки:

```
queues => odbc,asterisk,queues
queue_members => odbc,asterisk,queue_members
```

Сохраните и перезагрузите конфигурацию очереди из интерфейса командной строки Asterisk CLI:

```
*CLI> queues reload
```

Убедитесь, что ваши очереди были загружены в память (не забудьте убедиться, что файл *agents.conf* существует:

```
localhost*CLI> queue show
support      has 0 calls (max unlimited) in 'rrmemory' strategy
```

```
(0s holdtime, 0s talktime), W:0, C:0, A:0, SL:0.0% within 0s
   No Members
   No Callers

sales        has 0 calls (max unlimited) in 'rrmemory' strategy
(0s holdtime, 0s talktime), W:0, C:0, A:0, SL:0.0% within 0s
   No Members
   No Callers
```

Выходные данные `queue show` предоставляют различную информацию, в том числе детали, подробно описанные в Таблице 12-2.

Таблица 12-2. Описание вывода queue show

Поле	Описание
`W:`	Вес очереди
`C:`	Количество вызовов в очереди
`A:`	Количество звонков, на которые ответил участник
`SL:`	Уровень обслуживания

Теперь, когда вы создали очереди - вам нужно настроить диалплан так, чтобы звонки могли попадать в очередь.

Добавьте следующую логику диалплана в файл *extensions.conf* (где-нибудь в контексте `[sets]`):

```
exten => 610,1,Noop()
  same => n,Progress()
  same => n,Queue(sales)
  same => n,Hangup()

exten => 611,1,Noop()
  same => n,Progress()
  same => n,Queue(support)
  same => n,Hangup()
```

Сохраните изменения в *extensions.conf* и перезагрузите диалплан с помощью команды CLI `dialplan reload`.

Если вы наберете добавочный номер 610 или 611, то получите следующий вывод:

```
== Setting global variable 'SIPDOMAIN' to '172.29.1.178'
-- Executing [610@sets:1] NoOp("PJSIP/SOFTPHONE_A-00000004", "") in new stack
-- Executing [610@sets:2] Progress("PJSIP/SOFTPHONE_A-00000004", "") in new stack
-- Executing [610@sets:3] Queue("PJSIP/SOFTPHONE_A-00000004", "test") in new stack
   > 0x7facc801ed60 -- Strict RTP learning after remote set to: 172.29.1.166:4022
-- Started music on hold, class 'testmoh', on channel 'PJSIP/SOFTPHONE_A-00000004'
   > 0x7facc801ed60 -- Strict RTP switching to RTP target 172.29.1.166:4022 as source
   > 0x7facc801ed60 -- Strict RTP learning complete - Locking on 172.29.1.166:4022
-- Stopped music on hold on PJSIP/SOFTPHONE_A-00000004
== Spawn extension (sets, 610, 3) exited non-zero on 'PJSIP/SOFTPHONE_A-00000004'
```

Обратите внимание, что *в этот момент вы не присоединитесь к очереди*, потому что в очереди нет агентов для ответа на вызов. У нас настроены `joinempty=no` и `leftwhenempty=yes` - поэтому вызывающие не будут помещаться в очередь. (Это была бы хорошая возможность поэкспериментировать с опциями `joinempty` и `leftwhenempty` в *queues.conf*, чтобы лучше понять их влияние на очереди).

В следующем разделе мы покажем, как добавлять участников в очередь (а также другие взаимодействия участников с очередью, такие как пауза/отмена паузы).

Участники очереди

Очереди не очень полезны, если кто-то не отвечает на входящие вызовы - поэтому нам нужен метод, позволяющий агентам входить в очереди для ответа на вызовы. Существуют различные способы

решения этой задачи - поэтому мы покажем вам, как добавлять участников в очередь как вручную (как администратору через CLI или жестко прописывая в таблице `queue_members`), так и динамически (в качестве агента через расширение, определенное в диалплане). Мы начнем с метода Asterisk CLI, который позволяет легко добавлять участников в очередь для тестирования с минимальными изменениями диалплана. Далее мы покажем, как вы можете определить участников в таблице `queue_members`. Наконец, мы покажем вам, как добавить логику диалплана, позволяющую агентам входить в очереди и выходить из них, а также приостанавливать и возобновлять себя в очередях, в которые они вошли (это, вероятно, лучший метод для продакшена).

Управление участниками очереди через CLI

Мы можем добавить участников очереди в любую доступную очередь через команду Asterisk CLI `queue add`. Формат команды добавления очереди `queue add` (все в одной строке):

```
*CLI> queue add member channel to queue [[[penalty penalty] as
membername] state_interface interface]
```

`channel` - это канал, который мы хотим добавить в очередь, например `SIP/0000FFFF0003`, а имя *`queue`* будет что-то вроде `support` или `sales` - любое имя очереди, которое существует в */etc/asterisk/queues.conf*. Пока мы будем игнорировать вариант с *`penalty`*, но обсудим его в разделе «Расширенные очереди» (`penalty` используется для контроля ранга участника в очереди, что может быть важно для операторов, вошедших в несколько очередей или имеющих разные навыки). Мы можем определить *`membername`* чтобы предоставить подробные сведения для механизма регистрации очередей.

Опция `state_interface` информирует очередь о состоянии устройства, отслеживаемого для этого агента. Детали работы с состояниями устройств обсуждаются в Главе 13. Сходите и проработайте эту главу, а затем вернитесь сюда и продолжайте. Не волнуйтесь - мы подождем.

Теперь, когда вы добавили `callcounter=yes` в *sip.conf* (мы будем использовать SIP-каналы во всех остальных наших примерах), давайте посмотрим как добавлять участников в наши очереди из Asterisk CLI.

Добавление участника очереди в очередь `support` можно выполнить с помощью команды `queue add member`:

```
*CLI> queue add member PJSIP/SOFTPHONE_B to support

Added interface 'PJSIP/SOFTPHON_B' to queue 'support'
```

Запрос очереди подтвердит, что наш новый участник был добавлен:

```
*CLI> queue show support

support has 0 calls (max unlimited) in 'rrmemory' strategy (0s holdtime, 0s talktime),
W:0, C:0, A:0, SL:0.0%, SL2:0.0% within 0s
    Members:
        PJSIP/SOFTPHONE_B (ringinuse disabled) (dynamic)(Not in use) has taken no calls yet
    No Callers
```

Чтобы удалить участника очереди - вы должны использовать команду `queue remove member`:

```
*CLI> queue remove member PJSIP/SOFTPHONE_B from support

Removed interface PJSIP/SOFTPHONE_B from queue 'support'
```

Конечно же вы можете снова использовать команду `queue show`, чтобы убедиться, что ваш участник был удален из очереди:

```
*CLI> queue show support

support has 0 calls (max unlimited) in 'rrmemory' strategy (0s holdtime, 0s talktime),
W:0, C:0, A:0, SL:0.0%, SL2:0.0% within 0s
   Members:
      PJSIP/SOFTPHONE_B (ringinuse disabled) (dynamic) (Not in use) has taken no calls yet
   No Callers
```

Мы также можем приостанавливать и возобновлять участников в очереди из консоли Asterisk, используя команды `queue pause member` и `queue unpause member`. Они используют формат, аналогичный предыдущим командам, которые мы использовали:

```
*CLI> queue pause member PJSIP/SOFTPHONE_B queue support reason Callbacks

paused interface 'PJSIP/SOFTPHONE_B' in queue 'support' for reason 'Callbacks'

*CLI> queue show support
support has 0 calls (max unlimited) in 'rrmemory' strategy
(0s holdtime, 0s talktime), W:0, C:0, A:0, SL:0.0% within 0s
   Members:
      SIP/0000FFFF0001 (dynamic) (paused) (Not in use) has taken no calls yet
   No Callers

*CLI> queue show support

support has 0 calls (max unlimited) in 'rrmemory' strategy (0s holdtime, 0s talktime),
 W:0, C:0, A:0, SL:0.0%, SL2:0.0% within 0s
   Members:
      PJSIP/SOFTPHONE_B (ringinuse disabled) (dynamic) (paused:Callbacks) (Not in use)
has taken no calls yet
   No Callers
```

Добавляя причину (reason) приостановки работы участника очереди, например время обеда (lunchtime), вы гарантируете, что ваши журналы очереди будут содержать дополнительную информацию, которая может оказаться полезной. Вот как можно возобновить работу участника:

```
*CLI> queue unpause member PJSIP/SOFTPHONE_B queue support reason FinishedCallBacks

unpaused interface 'PJSIP/SOFTPHONE_B' in queue 'support' for reason 'FinishedCallbacks'
```

В производственной среде CLI (интерфейс командной строки) не является лучшим способом управления состоянием агентов в очереди. Вместо этого существуют приложения диалплана, позволяющие агентам информировать очередь об их доступности.

Определение участников очереди в таблице queue_members

Если вы определите участника очереди в таблице базы данных `asterisk.queue_members`, то он всегда будет зарегистрирован в очереди. Это как правило не очень хорошо, если ваши участники люди, так как люди во время работы делают перерывы и могут отлучаться от рабочего места.

В каждом определении очереди вы просто определяете участников следующим образом:

```
MySQL> insert into `asterisk`.`queue_members`
(queue_name,interface,penalty)

VALUES
'hotline','PJSIP/SOME_NON_HUMAN','0');
```

В типичной очереди (в которой есть группа людей, отвечающих на вызовы), вы обнаружите, что определение участников в таблицу `queue_members` может навредить. Агенты должны иметь возможность входить и выходить из системы (а не автоматически регистрироваться всякий раз, когда очередь перезагружается). Мы не рекомендуем определять участников в таблице `queue_members`, если только нет других целей (таких как банк устройств, отвечающих на вызовы, где вы хотите использовать очередь для балансировки нагрузки вызовов в пул устройств или групповой вызов, где все телефоны звонят одновременно, независимо от того, сидит ли кто-нибудь рядом с телефоном).

Управление участниками очереди с помощью логики диалплана

В колл-центре, в котором работают живые агенты, чаще всего агенты сами входят в систему и выходят из нее в начале и конце своей смены (или когда они идут на обед, в ванную или иным образом недоступны для очереди).

Для этого мы будем использовать следующие приложения диалплана:

- AddQueueMember()
- RemoveQueueMember()

При входе в очередь может случиться так, что агенту необходимо перевести себя в состояние, когда он временно недоступен для приема вызовов. Следующие приложения позволят сделать это:

- PauseQueueMember()
- UnpauseQueueMember()

Приложения Add/Remove используются для входа и выхода из системы, а Pause/Unpause - для коротких периодов отсутствия агента. Разница лишь в том, что Pause и Unpause устанавливают элемент как недоступный/доступный (unavailable/available), фактически не удаляя их из очереди. Это бывает полезно для отчетности (если участник приостановлен - администратор очереди может видеть, что он вошел в очередь, но просто недоступен для приема вызовов в данный момент). Если вы не уверены, какой из них использовать, мы рекомендуем агентам использовать Add/Remove, когда они физически не находятся у своего телефона и Pause/Unpause, когда они находятся на своем рабочем месте, но временно недоступны.

Если есть сомнения - будет лучше чтобы ваши агенты выходили из системы.

Использование паузы и снятия с паузы

В некоторых средах Pause и Unpause используются для всех действий в течение дня, которые делают агента недоступным (например, во время обеденного перерыва и при выполнении работы не связанной с очередью). Однако в большинстве call-центров - если агент не находится рядом с телефоном и не готов принять вызов в данный момент - он вообще не должен входить в систему, даже если будет отсутствовать на рабочем месте в течение нескольких минут. (например, для перерыва в ванной).

Некоторым руководителям нравится использовать настройки Pause/Unpause как своего рода часы для отслеживания когда их сотрудники приходят на работу и уходят в конце дня, а также сколько времени они проводят за своими столами и на перерывах. Это может быть неразумной практикой, так как цель этих приложений - информировать очередь о доступности агента, а отслеживание активности - вторичная функция.

Здесь важно отметить, что параметр joinempty в таблице asterisk.queues был рассмотрен ранее. Если агент приостановлен - он все еще находится в очереди. Предположим, что рабочая смена подходит к концу, а один агент несколько часов назад поставил себя на паузу для работы над проектом. Все остальные агенты вышли из системы и ушли домой. Поступает вызов. Очередь заметит, что агент вошел в очередь, и, следовательно, поставит вызов в очередь, несмотря на то, что в действительности в данное время в этой очереди нет людей, способных ответить на вызов. Этот абонент может в конечном итоге задержаться в очереди без персонала на неопределенный срок.

Короче говоря, агенты, которые не сидят за столами и не планируют принимать звонки в течение следующих нескольких минут, должны выйти из системы. Pause/Unpause следует использовать только для кратковременных моментов недоступности (если это вообще возможно). Если вы хотите использовать свою телефонную систему для учета рабочего времени - есть много отличных способов сделать это с помощью Asterisk, но приложения queue member - это не тот способ, который мы посоветуем.

Давайте создадим простую логику диалплана, позволяющую нашим агентам указывать свою доступность для очереди. Мы собираемся использовать функцию диалплана CUT(), чтобы извлечь имя нашего канала из вызова в систему, чтобы очередь знала какой канал входит в очередь.

Мы создали этот диалплан чтобы показать простой процесс входа и выхода из очереди, а также изменения приостановленного статуса участника в очереди. Мы делаем это только для одной очереди, которую ранее определили в файле *queues.conf*. Переменные состояния канала,

установленные приложениями AddQueueMember(), RemoveQueueMember(), PauseQueueMember() и UnpauseQueueMember() могут использоваться для воспроизведения Playback() объявлений участникам очереди после выполнения ими определенных функций для сообщения им, успешно ли они совершили вход/выход или паузу/возобновление:

```
exten => *731,1,Page(${PAGELIST},i,120)

exten => *732,1,Verbose(2,Logging In Queue Member)
   same => n,Set(MemberChannel=${CHANNEL(channeltype)}/${CHANNEL(endpoint)})
   same => n,AddQueueMember(support,${MemberChannel})
   same => n,Verbose(1,${AQMSTATUS}) ; ADDED, MEMBERALREADY, NOSUCHQUEUE
   same => n,Playback(agent-loginok)
   same => n,Hangup()

exten => *733,1,Verbose(2,Logging Out Queue Member)
   same => n,Set(MemberChannel=${CHANNEL(channeltype)}/${CHANNEL(endpoint)})
   same => n,RemoveQueueMember(support,${MemberChannel})
   same => n,Verbose(1,${RQMSTATUS}) ; REMOVED, NOTINQUEUE, NOSUCHQUEUE
   same => n,Playback(agent-loggedoff)
   same => n,Hangup()

exten => *734,1,Verbose(2,Pause Queue Member)
   same => n,Set(MemberChannel=${CHANNEL(channeltype)}/${CHANNEL(endpoint)})
   same => n,PauseQueueMember(support,${MemberChannel})
   same => n,Verbose(1,${PQMSTATUS}) ; PAUSED, NOTFOUND
   same => n,Playback(dictate/paused)
   same => n,Hangup()

exten => *735,1,Verbose(2,Unpause Queue Member)
   same => n,Set(MemberChannel=${CHANNEL(channeltype)}/${CHANNEL(endpoint)})
   same => n,UnpauseQueueMember(support,${MemberChannel})
   same => n,Verbose(1,${UPQMSTATUS}) ; UNPAUSED, NOTFOUND
   same => n,Playback(agent-loginok)
   same => n,Hangup()

exten => *98,1,NoOp(Access voicemail retrieval.)
```

Автоматический вход и выход из нескольких очередей

Довольно часто агент является участником более чем одной очереди. Вместо того, чтобы иметь отдельное расширение для входа в каждую очередь (или требовать от агентов информацию о том, в какие очереди они хотят войти), этот код использует базу данных Asterisk (astdb) для хранения информации об участии в очереди для каждого агента, а затем циклически проходит через каждую очередь, в которую входят агенты, поочередно регистрируя их в каждой очереди.

Для того, чтобы этот код работал, необходимо добавить запись, аналогичную следующей, в AstDB через CLI Asterisk. Например, будет сохранён элемент SOFTPHONE_A как находящийся в очередях support и sales:[3]

```
CLI> database put queue_agent SOFTPHONE_A/available_queues support^sales
```

Вам нужно будет сделать это один раз для каждого агента, независимо от того, в скольких очередях они являются участниками.

Если Вы запросите базу данных Asterisk, то должны получить результат, подобный следующему:

```
pbx*CLI> database show queue_agent
/queue_agent/SOFTPHONE_A/available_queues : support^sales
```

Следующий код диалплана является примером того, как разрешить автоматическое добавление этого участника в очереди support и sales. Мы определили подпрограмму, используемую для настройки трех канальных переменных (MemberChannel, MemberChanType, AvailableQueues). Эти переменные канала затем используют расширения на вход (*736), выход (*737), паузу (*738) и возобновление

3 Мы собираемся использовать символ ^ в качестве разделителя. Возможно, вы могли бы использовать вместо него другой символ, только кроме такого, который анализатор синтаксиса Asterisk будет рассматривать как обычный разделитель (и, таким образом, будет сбит с толку). Поэтому избегайте запятых, точек с запятой и так далее.

(*739). Каждое из расширений использует подпрограмму subSetupAvailableQueues чтобы установить эти переменные канала и убедиться, что AstDB содержит список одной или нескольких очередей для устройства, с которого вызывается участник очереди.

В конце вашего файла *extensions.conf*, куда вы поместили свои подпрограммы, добавьте следующее:

```
[subSetupAvailableQueues]
; Эта функция используется для различных процедур входа/выхода/паузы/возобновления
; в нашем примере вход в несколько очередей.
;
exten => start,1,Verbose(2,Checking for available queues)
; Получаем имя текущих каналов пира
   same => n,Set(MemberChannel=${CHANNEL(endpoint)})
; Получаем тип технологии текущих каналов
   same => n,Set(MemberChanType=${CHANNEL(channeltype)})
; Полуаем список доступных очередей для этого агента
   same => n,Set(AvailableQueues=${DB(queue_agent/${MemberChannel}/available_queues)})
; если этому агенту не назначены очереди, мы будем обрабатывать их
; в расширении no_queues_available
   same => n,GotoIf($[${ISNULL(${AvailableQueues})}]?no_queues_available,1)
   same => n,Return()

exten => no_queues_available,1,Verbose(2,No queues available for agent ${MemberChannel})
; воспроизведение сообщения о том, что канал еще не назначен
   same => n,Playback(silence/1&channel&not-yet-assigned)
   same => n,Hangup()
```

Далее, в контекст [sets] добавьте следующее:

```
; Вход в несколько очередей через систему AstDB
exten => *736,1,Verbose(2,Logging into multiple queues per the database values)
; получить доступные очереди для этого канала
   same => n,GoSub(subSetupAvailableQueues,start,1())
   same => n,Set(QueueCounter=1)   ; установка переменной счетчика
; используя CUT() получить первую очередь из списка, возвращенную из AstDB.
; Обратите внимание, что мы использовали '^' в качестве разделителя.
   same => n,Set(WorkingQueue=${CUT(AvailableQueues,^,${QueueCounter})})
; Пока переменная канала WorkingQueue содержит значение, циклично выполняем
   same => n,While($[${EXISTS(${WorkingQueue})}])
; AddQueueMember(queuename[,interface[,penalty[,options[,membername
;  [,stateinterface]]]]])
; Добавить канал в очередь, настроить интерфейс для вызова
;  и интерфейс для мониторинга состояния устройства
; *** Это все должно быть в одной строке
   same => n,AddQueueMember(
    ${WorkingQueue},${MemberChanType}/${MemberChannel},,,${MemberChanType}/${MemberChannel})
    same => n,Set(QueueCounter=$[${QueueCounter} + 1])    ; увеличивает наш счетчик
; получить следующую доступную очередь; если она равна нулю - завершить цикл
    same => n,Set(WorkingQueue=${CUT(AvailableQueues,^,${QueueCounter})})
    same => n,EndWhile()
; пусть агент знает, что он вошёл в систему
   same => n,Playback(silence/1&agent-loginok)
   same => n,Hangup()

exten => no_queues_available,1,Verbose(2,No queues available for ${MemberChannel})
   same => n,Playback(silence/1&channel&not-yet-assigned)
   same => n,Hangup()

; Используется для регистрации агентов во всех настроенных очередях в базе данных AstDB
exten => *737,1,Verbose(2,Logging out of multiple queues)
; Поскольку мы повторно использовали некоторый код, то поместили дубликат кода в
подпрограмму
   same => n,GoSub(subSetupAvailableQueues,start,1())
   same => n,Set(QueueCounter=1)
   same => n,Set(WorkingQueue=${CUT(AvailableQueues,^,${QueueCounter})})
   same => n,While($[${EXISTS(${WorkingQueue})}])
   same => n,RemoveQueueMember(${WorkingQueue},${MemberChanType}/${MemberChannel})
   same => n,Set(QueueCounter=$[${QueueCounter} + 1])
   same => n,Set(WorkingQueue=${CUT(AvailableQueues,^,${QueueCounter})})
   same => n,EndWhile()
```

```
   same => n,Playback(silence/1&agent-loggedoff)
   same => n,Hangup()

; Используется для приостановки агентов во всех доступных очередях
exten => *738,1,Verbose(2,Pausing member in all queues)
   same => n,GoSub(subSetupAvailableQueues,start,1())
   ; если мы не определяем очередь, то участник приостанавливается во всех очередях
   same => n,PauseQueueMember(,${MemberChanType}/${MemberChannel})
   same => n,GotoIf($[${PQMSTATUS} = PAUSED]?agent_paused,1:agent_not_found,1)

exten => agent_paused,1,Verbose(2,Agent paused successfully)
   same => n,Playback(dictate/paused)
   same => n,Hangup()

; Используется для отмены паузы агентов во всех доступных очередях
exten => *739,1,Verbose(2,UnPausing member in all queues)
   same => n,GoSub(subSetupAvailableQueues,start,1())
 ; если мы не определяем очередь, то элемент не будет снят с паузы во всех очередях
   same => n,UnPauseQueueMember(,${MemberChanType}/${MemberChannel})
   same => n,GotoIf($[${UPQMSTATUS} = UNPAUSED]?agent_unpaused,1:agent_not_found,1)

exten => agent_unpaused,1,Verbose(2,Agent paused successfully)
   same => n,Playback(silence/1&available)

; Используется как для приостановки, так и для продолжения функциональности диалплана
exten => agent_not_found,1,Verbose(2,Agent was not found)
   same => n,Playback(silence/1&cannot-complete-as-dialed)
```

Вы можете дополнительно усовершенствовать эти процедуры входа и выхода, чтобы учесть, что переменные канала `AQMSTATUS` и `RQMSTATUS` устанавливаются каждый раз при использовании `AddQueueMember()` и `RemoveQueueMember()`. Например, можно установить флаг, позволяющий участнику очереди знать, что он не был добавлен в очередь, или даже добавить записи или использовать системы преобразования текста в речь для воспроизведения конкретной очереди, создающей проблему. Или, если вы отслеживаете очереди через AMI, то можете получить всплывающее окно экрана, или использовать `JabberSend()` для отправки участнику очереди мгновенного сообщения, или...(Разве Asterisk это не весело?).

Расширенные очереди

В этом разделе мы рассмотрим некоторые более тонкие элементы управления очередью, такие как параметры управления объявлениями и когда абоненты (callers) должны быть помещены в очередь (или удалены из нее). Мы также рассмотрим пенальти (ударение на первую букву "е") и приоритеты, исследуя возможности контроля агентов в нашей очереди, отдавая предпочтение пулу агентов, а затем увеличивая этот пул динамически на основе времени ожидания в очереди. Наконец, мы рассмотрим использование локальных каналов в качестве участников очереди, что даст нам возможность выполнять трюки диалплана до подключения абонента к агенту.

Очередь с приоритетом (Queue Weighting)

Иногда вам нужно добавить людей в очередь с более высоким приоритетом, чем у других абонентов. Возможно абонент уже провел некоторое время в очереди и агент принял некоторую информацию, но понял, что абонент должен быть переведен в другую очередь. В этом случае, чтобы свести к минимуму общее время ожидания абонента, возможно, было бы желательно перенести вызов в приоритетную очередь, которая имеет более высокий вес (`weight`) (и, следовательно, более высокое предпочтение), где ему ответят быстрее.

Установка более высокого приоритета для очереди выполняется с помощью параметра `weight`. Если у вас есть две очереди с разными весом (например, `support` и `support-priority`), агентам, назначенным в обе очереди, будут переданы вызовы из очереди с более высоким приоритетом, а не вызовы из очереди с более низким приоритетом. Эти агенты не будут принимать никаких вызовов из очереди с более низким приоритетом, пока очередь с более высоким приоритетом не будет очищена.

(Обычно есть некоторые агенты, которые назначаются только в очередь с более низким приоритетом, чтобы гарантировать своевременную обработку этих вызовов.) Например, если мы поместим участника очереди Джеймса Шоу в обе очереди `support` и `support-priority`, абоненты в очереди `support-priority` будут иметь предпочтительное положение вместе с Джеймсом, по сравнению с абонентами в очереди `support`.

Давайте посмотрим как бы мы реализовали это. Во-первых, нам нужно создать новую очередь, аналогичную очереди `support`, за исключением опции `weight`.

```
MySQL> INSERT INTO `asterisk`.`queues`
(name,strategy,joinempty,leavewhenempty,ringinuse,autofill,musiconhold,monitor_format,
monitor_type,weight)

VALUES
('support-priority','rrmemory','unavailable,invalid,unknown','unavailable,invalid,unknown',
'no','yes','default','wav','MixMonitor','10');
```

С нашей новой настроенной очередью мы можем теперь создать два расширения для трансфера абонентов. Это можно сделать везде, где вы обычно размещаете логику диалплана для выполнения трансферов. Мы будем использовать контекст `LocalSets`, который ранее включили в качестве начального контекста для наших устройств:

```
exten => 611,1,Noop()
   same => n,Progress()
   same => n,Queue(support)
   same => n,Hangup()

exten => 612,1,Noop()
   same => n,Progress()
   same => n,Queue(support-priority)
   same => n,Hangup()

exten => *724,1,Noop(Page)
```

Осталось убедиться что все ваши участники очереди помещены в обе очереди.

Приоритет участника очереди

Внутри очереди мы можем применить пенальти к участникам, чтобы уменьшить их предпочтение быть вызванными, когда есть люди, ожидающие в определенной очереди. Например, мы можем применять пенальти когда хотим чтобы они были участниками очереди, но принимали вызовы только тогда, когда очередь заполнится до тех пор, когда все наши предпочтительные агенты будут недоступны. Выставляя величину пенальти для каждого участника очереди[4] - мы можем контролировать предпочтения для прихождения звонков, но при этом гарантировать что другие участники очереди будут доступны для ответа абонентов, если предпочтительный участник недоступен.

Пенальти также могут быть определены с помощью `AddQueueMember()`. Мы изменим наш вход в несколько очередей, чтобы обеспечить необходимые пенальти.

Во-первых, давайте обновим нашу AstDB, чтобы включить пенальти для участника:

```
*CLI> database put queue_agent SOFTPHONE_A/penalty 0^2

*CLI> database show queue agent

/queue_agent/SOFTPHONE_A/available_queues    : support^sales
/queue_agent/SOFTPHONE_A/penalty             : 0^2
```

Далее, несколько изменений в нашем диалплане.

Подпрограмме нужна новая строка (некоторый код был удален для краткости, заменен на `; ...`):

4 Похоже на добавление балласта к жокею или гоночному автомобилю.

```
[subSetupAvailableQueues]
; ...
; Получить список очередей, доступных для этого агента
   same => n,Set(AvailableQueues=${DB(queue_agent/${MemberChannel}/available_queues)})
   same => n,Set(MemberPenalties=${DB(queue_agent/${MemberChannel}/penalty)})
; если нет назначенных очередей ...
```

Контекст `[sets]` также требует нескольких новых строк (некоторый код был удален для краткости, заменен на `; ...`). Вставляйте/изменяйте только код, выделенный жирным шрифтом.

```
exten => \*736,1,Verbose(2,Logging into multiple queues per the database values)
; ...
   same => n,Set(WorkingQueue=${CUT(AvailableQueues,^,${QueueCounter})})
   same => n,Set(WorkingPenalty=${CUT(MemberPenalties,^,${QueueCounter})})
; While the WorkingQueue ...
; …
   same => n,Set(WorkingQueue=${CUT(AvailableQueues,^,${QueueCounter})})
   same => n,Set(WorkingPenalty=${CUT(MemberPenalties,^,${QueueCounter})})
   same => n,EndWhile()
; ...
```

Эти примеры, вероятно, не подходят для продакшена (мы бы использовали специально построенные таблицы MySQL для такого рода вещей, а не AstDB), но он дает вам представление о том, как диалплан может быть использован для применения динамической логики к более сложным сценариям конфигурации.

Динамическое изменение пенальти (queuerules)

Используя таблицу `asterisk.queuerules` можно определить правила, изменяющие значения переменных канала `QUEUE_MIN_PENALTY` и `QUEUE_MAX_PENALTY`. Переменные канала `QUEUE_MIN_PENALTY` и `QUEUE_MAX_PENALTY` используются для управления тем, какие участники очереди предпочтительнее для обслуживания абонентов. Допустим, у нас есть очередь с названием `support` и имеется пять участников очереди с различными пенальти в диапазоне от `1` до `5`. Если до того, как абонент войдет в очередь, для переменной канала `QUEUE_MIN_PENALTY` задано значение `2`, а для `QUEUE_MAX_PENALTY - 4`, то для ответа на этот вызов будут считаться доступными только участники очереди, пенальти которых находятся в диапазоне от `2` до `4`.:

```
same => n,Set(QUEUE_MIN_PENALTY=2) ; установить минимальный пенальти участника
same => n,Set(QUEUE_MAX_PENALTY=4) ; установить максимальное пенальти участника
same => n,Queue(support)           ; вход в очередь с минимальными и максимальными пенальти
                                   ; для участников, которые будут использоваться
```

Более того, во время пребывания абонента в очереди, мы можем динамически изменять значения `QUEUE_MIN_PENALTY` и `QUEUE_MAX_PENALTY` для этого абонента. Это позволяет использовать либо больше, либо другой набор участников очереди, в зависимости от того, как долго вызывающий абонент ожидает в очереди. Например, в предыдущем примере мы могли бы изменить минимальное значение пенальти на `1`, а максимальное - на `5`, если абонент находится более 60 секунд в очереди.

Файл примера *~/src/asterisk-15.*/configs/samples/queuerules.conf.sample* содержит отличную справку о том, как работают правила очереди.

Правила определяются с использованием таблицы `asterisk.queuerules`. Несколько правил могут быть созданы для того, чтобы облегчить различные изменения пенальти на протяжении всего вызова. Давайте посмотрим, как мы можем определить правило.:

```
MySQL> insert into `asterisk`.`queue_rules`
(rule_name,time,min_penalty,max_penalty)

VALUES
('more_members',60,5,1);
```

Новые правила будут касаться только новых абонентов, входящих в очередь, а не существующих абонентов, уже находящихся в ней.

Мы назвали правило `more_members` и определили следующие значения:

`60`

Количество секунд ожидания перед изменением значений пенальти.

`5`

Новое `QUEUE_MAX_PENALTY`.

`1`

Новое `QUEUE_MIN_PENALTY`.

Теперь мы можем указать нашим очередям использовать его.

```
MySQL> update `asterisk`.`queues`

set defaultrule='more_members' where `name` in ('sales','support')
```

Файл *queuerules.conf.sample* показывает, что эти правила достаточно гибкие. Если вы хотите детально контролировать приоритеты вызовов - вам может потребоваться дополнительная лабораторная работа.

Управление объявлениями

Asterisk имеет возможность проигрывать несколько объявлений абонентам, ожидающим в очереди. Например, вы можете объявить позицию вызывающего абонента в очереди, объявить среднее время ожидания или периодически благодарить вызывающих абонентов за ожидание (или все, что скажут ваши аудиофайлы). Важно тщательно настроить значения, контролирующие когда эти объявления воспроизводятся для абонентов, потому что объявление их позиции, благодарность за ожидание и информирование о среднем времени ожидания слишком часто - будет раздражать их, что не является нашей целью.

Воспроизведение объявлений между музыкальными файлами на удержании

Вместо того, чтобы разбираться со сложностями объявлений для каждой из ваших очередей - вы можете альтернативно (или совместно) использовать функциональность объявлений, определенную в *musiconhold.conf*. Перед воспроизведением файла музыки на удержании - будет воспроизведен файл объявления, а затем воспроизведен снова между аудиофайлами. Допустим, у вас есть 5-минутный цикл аудио, но вы хотите воспроизводить сообщение “Спасибо за ожидание” каждые 30 секунд. Вы можете разбить аудиофайл на 30-секундные сегменты, задать их имена, начиная с `00-`, `01-`, `02-` и так далее (чтобы они воспроизводились по порядку), а затем определить объявления в *musiconhold.conf* чтобы выглядело примерно так:

```
[moh_jazz_queue]
mode=files
sort=alpha
announcement=queue-thankyou
directory=moh_jazz_queue
```

В таблице очередей есть несколько параметров, которые можно использовать для точной настройки того, какие объявления и когда воспроизводятся для ваших абонентов. Полный список опций очереди доступен в разделе *~/src/asterisk-16.*/configs/samples/queues.conf.sample*. Таблица 12-3 рассматривает несколько наиболее полезных из них.

Таблица 12-3. Параметры, связанные с контролем времени запроса в очереди

Параметр	Доступные значения	Описание
`announce-frequency`	Значение в секундах	Определяет, как часто объявлять позицию вызывающего абонента и/или предполагаемое время удержания в очереди. Установите это значение на ноль чтобы отключить.
`min-announce-frequency`	Значение в секундах	Указывает минимальное количество времени, которое должно пройти, прежде чем мы снова объявим позицию абонента в очереди. Используется когда позиция абонента может часто меняться для предотвращения прослушивания нескольких объявлений за короткий промежуток времени.
`periodic-announce-frequency`	Значение в секундах	Указывает, как часто делать периодические объявления абоненту.
`random-periodic-announce`	`yes, no`	Если установлено значение `yes` - будут воспроизводиться определенные периодические объявления в случайном порядке. Смотрите `periodic-announce`.
`relative-periodic-announce`	`yes, no`	Если установлено значение `yes` - для `periodic-announce-frequency` таймер запустится когда будет достигнут конец воспроизводимого файла, а не с самого начала. По умолчанию `no`.
`announce-holdtime`	`yes`, `no`, `once`	Определяет, следует ли воспроизводить расчетное время ожидания вместе с периодическими объявлениями. Может быть установлено значение `yes`, `no` или только один раз - `once`.
`announce-position`	`yes`, `no`, `limit`, `more`	Определяет, следует ли объявлять позицию вызывающего абонента в очереди. Если установлено значение `no` - позиция никогда не будет объявлена. Если установлено значение `yes` - позиция абонента всегда будет объявлена. При значении `limit` - абонент услышит свою позицию в очереди только в том случае, если она находится в пределе, определенном параметром `announce-position-limit`. Если задано значение `more` - вызывающий услышит свою позицию только в том случае, если она выходит за пределы числа, определенного параметром `announce-position-limit`.
`announce-position-limit`	Число от нуля и больше	Используется, если вы определили объявленную позицию как `limit` или `more`.
`announce-round-seconds`	Значение в секундах	Если это значение отлично от нуля — объявляется также количество секунд ожидания и округляется до определенного значения.

Таблица 12-4 определяет файлы, которые будут использоваться при воспроизведении объявлений вызывающему абоненту.

Таблица 12-4. Параметры управления воспроизведением подсказок в очереди

Параметр	Доступные значения	Описание
`musicclass`	Класс музыки определенный в *musiconhold.conf*	Устанавливает класс музыки, который будет использоваться определенной очередью. Вы также можете переопределить это значение с помощью канальной переменной `CHANNEL(musicclass)`.
`queue-thankyou`	Имя файла для воспроизведения	Если не определено - воспроизводится значение по умолчанию («Благодарим за терпение»). Если установлено пустое значение - подсказка не будет воспроизводиться вообще.
`queue-youarenext`	Имя файла для воспроизведения	Если не определено - воспроизводится значение по умолчанию («Ваш звонок является первым в очереди и будет отвечен первым доступным оператором»). Если установлено пустое значение - подсказка не будет воспроизводиться вообще.
`queue-thereare`	Имя файла для воспроизведения	Если не определено - воспроизводится значение по умолчанию («Ваша позиция в очереди»). Если установлено пустое значение -

Параметр	Доступные значения	Описание
		подсказка не будет воспроизводиться вообще.
`queue-callswaiting`	Имя файла для воспроизведения	Если не определено - воспроизводится значение по умолчанию («Ожидайте, пожалуйста, ответа оператора»). Если установлено пустое значение - подсказка не будет воспроизводиться вообще.
`queue-holdtime`	Имя файла для воспроизведения	Если не определено - воспроизводится значение по умолчанию («Прогнозируемое время ожидания составляет»). Если установлено пустое значение - подсказка не будет воспроизводиться вообще.
`queue-minutes`	Имя файла для воспроизведения	Если не определено - воспроизводится значение по умолчанию («минут»). Если установлено пустое значение - подсказка не будет воспроизводиться вообще.
`queue-seconds`	Имя файла для воспроизведения	Если не определено - воспроизводится значение по умолчанию («секунд»). Если установлено пустое значение - подсказка не будет воспроизводиться вообще.
`queue-reporthold`	Имя файла для воспроизведения	Если не определено - воспроизводится значение по умолчанию («Время ожидания»). Если установлено пустое значение - подсказка не будет воспроизводиться вообще.
`periodic-announce`	Набор периодических объявлений для воспроизведения, разделенных запятыми	Подсказки воспроизводятся в том порядке, в котором они определены. По умолчанию используется параметр `queue-periodic-announce` ("Все наши операторы заняты, пожалуйста, оставайтесь на линии и Ваш звонок будет обслужен первым доступным оператором").

Существует масса возможностей для гибкости при проектировании взаимодействия абонента во время ожидания, но, пожалуйста, не забывайте, что ваши абоненты никогда не будут счастливы ожидая в очереди. Кроме того, если вы нашли какую-то более-менее приличную музыку для МОН и ваши абоненты наслаждаются ею - прерывание воспроизведения еще одним сообщением несёт риск по-настоящему вскипятить их кровь. Когда абоненту наконец ответит оператор - он получит всю вспышку гнева, даже если это на самом деле ваша вина.[5]

Так что не нужно усложнять настройку удержания. Абоненты знают что они ждут, и они не рады этому. Доставьте их оператору как можно быстрее, с минимальным количеством глупостей пока они ожидают в очереди, и не поддавайтесь искушению сделать очередь более важной для абонентов, чем она есть на самом деле.

Переполнение

К сожалению, ваша очередь не всегда будет своевременно доставлять ваших абонентов к агенту. Когда различные условия заставляют очередь отклонять входящих абонентов - мы имеем ситуацию переполнения. Переполнение очереди выполняется либо со значением таймаута, либо при отсутствии доступных участников очереди (как определено `joinempty` или `leavewhenempty`). В этом разделе мы обсудим, как контролировать возникновение переполнения.

Контроль времени ожидания

Приложение `Queue()` поддерживает два вида тайм-аута: один определяет максимальный период времени, в течение которого вызывающий абонент находится в очереди, а другой - как долго следует звонить устройству при попытке подключить вызывающего абонента к участнику очереди. Эти парметры не связаны, но могут влиять друг на друга. В этом разделе мы будем говорить о максимальном периоде времени, в течение которого вызывающий абонент остается в приложении `Queue()` до того, как вызов переполнится, до следующего шага в диалплане, который может быть чем-то вроде `VoiceMail()` или даже другой очередью. После того, как вызов выпал из очереди - он может отправиться куда угодно, куда обычно может идти вызов, если он контролируется диалпланом.

5 Просто предупреждаю.

Тайм-ауты указываются в двух местах. Тайм-аут, указывающий, в течение какого времени звонить участникам очереди - указывается в таблице `queues`. Абсолютный тайм-аут (время пребывания абонента в очереди) контролируется с помощью приложения `Queue()`. Чтобы задать максимальное время пребывания абонентов в очереди - просто укажите его после имени очереди в приложении `Queue()`:

```
; Очередь
exten => 610,1,Noop()
   same => n,Progress()
   same => n,Queue(sales,120)
   same => n,Voicemail(${EXTEN}@queues,u)
   same => n,Hangup()

exten => 611,1,Noop()
   same => n,Progress()
   same => n,Queue(support,120)
   same => n,Voicemail(${EXTEN}@queues,u)
   same => n,Hangup()

exten => 612,1,Noop()
   same => n,Progress()
   same => n,Queue(support-priority,120)
   same => n,Voicemail(${EXTEN}@queues,u)
   same => n,Hangup()
```

Поскольку мы отправляем звонки на голосовую почту - нам понадобятся почтовые ящики:

```
MySQL> INSERT INTO `asterisk`.`voicemail`
(context,mailbox,password,fullname,email)

VALUES
('queues','610','192837','Queue sales','name@shifteight.org'),
('queues','611','192837','Queue support','name@shifteight.org'),
('queues','612','192837','Queue support-priority','name@shifteight.org');
```

Конечно, мы могли бы определить другое назначение, но приложение `VoiceMail()` является общим местом назначения переполнения для очереди. Очевидно, что отправка звонков на голосовую почту не идеальна (они надеялись поговорить с кем-то вживую), поэтому убедитесь, что кто-то регулярно проверяет эту почту и перезванивает вашим клиентам.

Предположим, мы установили абсолютное время ожидания равным 10 секундам - значение времени ожидания для звонков участникам очереди равным 5 секундам, а значение тайм-аута для повторной попытки - 4 секунды. В этом случае мы будем звонить участнику очереди в течение 5 секунд, а затем ждать 4 секунды, прежде чем звонить другому участнику очереди. Это дает нам до 9 секунд нашего абсолютного тайм-аута в 10 секунд. Получается, мы должны позвонить второму участнику очереди в течение 1 секунды и затем выйти из очереди, или мы должны позвонить этому участнику в течение полных 5 секунд перед выходом?

Мы контролируем, какое значение тайм-аута имеет приоритет с помощью опции `timeoutpriority` в таблице `queues`. Доступные значения: `app` (по умолчанию) и `conf`. Если мы хотим, чтобы тайм-аут приложения (абсолютный тайм-аут) имел приоритет, что привело бы к тому, что наш абонент был исключен ровно через 10 секунд (даже если он только начинал звонить агенту), мы должны установить значение `timeoutpriority` в `app`. Если мы хотим, чтобы таймаут файла конфигурации имел приоритет и закончил звонить участнику очереди, что заставит абонента оставаться в очереди немного дольше - мы должны установить для `timeoutpriority` значение `conf`. Значением по умолчанию является `app` (по умолчанию в предыдущих версиях Asterisk). Вероятно, в большинстве случаев вы будете использовать `conf` (особенно если хотите, чтобы опыт работы с абонентами был как можно менее странным).

```
MySQL> update `asterisk`.`queues` set timeoutpriority='conf'
  where name in ('sales','support','support-priority');
```

Цель состоит в том, чтобы доставить абонентов к агентам, так?

Управление временем присоединения и выхода из очереди

Asterisk предоставляет две опции, контролирующие когда вызывающие абоненты могут присоединиться и вынуждены покинуть очереди - обе на основе статусов участников очереди. Первая опция - `joinempty`, используется для контроля возможности абонентов изначально входить в очередь. Вторая - `leftwhenempty`, используется для управления событиями, приводящими к тому, что вызывающие абоненты, уже находящиеся в очереди, будут удалены из этой очереди (т.е. если все участники очереди станут недоступными). Оба параметра допускают разделенный запятыми список значений для управления этим поведением, как показано в Таблице 12-5.

Таблица 12-5. Параметры, которые можно установить для joinempty или leftwhenempty

Значение	Описание
`paused`	Участники считаются недоступными если они приостановлены.
`penalty`	Участники считаются недоступными если их пенальти меньше, чем `QUEUE_MAX_PENALTY`.
`inuse`	Участники считаются недоступными если состояния их устройств `InUse`.
`ringing`	Участники считаются недоступными если состояния их устройств `Ringing`.
`unavailable`	Применяется главным образом к каналам агента; если агент не вошел в систему, но является участником очереди - канал считается недоступным.
`invalid`	Участники считаются недоступными если их статус устройств является `Invalid`. Это типичное условие ошибки.
`unknown`	Участники считаются недоступными если состояние устройства `unknown`.
`wrapup`	Участники считаются недоступными если они в настоящее время находятся в состоянии перерыва после завершения вызова.

Для `joinempty`, перед помещением вызывающего абонента в очередь, все участники проверяются на доступность, используя факторы, перечисленные в качестве критериев. Если все участники считаются недоступными - вызывающему абоненту не будет разрешено войти в очередь, и выполнение диалплана будет продолжено со следующим приоритетом.[6] Для опции `leavewhenempty` статусы участников периодически проверяются на соответствие перечисленным условиям; если выясняется, что ни один участник не доступен для приема вызовов - абонент удаляется из очереди, а выполнение диалплана продолжается со следующего приоритета.

Примером использования `joinempty` может быть:

```
joinempty=unavailable,invalid,unknown
```

В этой конфигурации до того, как вызывающий абонент войдет в очередь, будут проверены состояния всех участников очереди и вызывающему не будет разрешено войти в очередь, если по крайней мере один участник очереди не будет найден со статусом, который не является `unavailable`, `invalid` или `unknown`.

Примером `leavewhenempty` может быть что-то вроде:

```
leavewhenempty=unavailable,invalid,unknown
```

В этом случае статусы участников очереди будут периодически проверяться, а вызывающие абоненты будут удаляться из очереди, если не будут найдены участники очереди, которые не имеют статуса недоступных, недействительных или неизвестных (`navailable`, `invalid`, `unknown`).

Предыдущие версии Asterisk использовали значения `yes`, `no`, `strict и loose` в качестве доступных значений. Сравнение этих значений показано в Таблице 12-6.

6 Если приоритет `n+1` (откуда было вызвано приложение `Queue()`) не определен - вызов будет прерван. Другими словами: не используйте эту функцию, если ваш диалплан не делает что-то полезное на шаге, следующем сразу за `Queue()`.

Таблица 12-6. Сравнение между старыми и новыми значениями для контроля присоединения и отключения абонентов от очереди

Значение	Сопоставление (joinempty)	Сопоставление (leavewhenempty)
yes	(пусто)	penalty,paused,invalid
no	penalty,paused,invalid	(пусто)
strict	penalty,paused,invalid,unavailable	penalty,paused,invalid,unavailable
loose	penalty,invalid	penalty,invalid

Использование локальных каналов

Использование локальных каналов в качестве участников очереди является мощным инструментом диалплана для вызова устройств агентов. Когда `Queue()` решает передать вызов агенту - использование локальных каналов позволяет нам определить пользовательские переменные канала, записать в файл журнала, установить некоторое ограничение на длину вызова (например, если это платная услуга), отправлять сообщения всех видов по всем устройствам, выполнять транзакции с базой данных и многие другие действия, которые мы могли бы захотеть сделать именно в этот момент. Обычно мы не контролируем, когда приложение `Queue()` решило представить вызывающего абонента конкретному участнику, но с локальными каналами мы получаем возможность последнего удара и даже можем вернуть `Congestion()`, который будет иметь эффект возврата вызывающего абонента в очередь, поскольку очередь не будет считать, что этот вызов был успешно доставлен агенту (это может быть очень удобно, поскольку некоторые внешние условия могут быть оценены до того, как вызов будет просто отправлен на конечную точку).

При использовании локальных каналов для очередей они добавляются так же, как и любые другие каналы, обычно динамически через приложение диалплана `AddQueueMember()`.

Нам нужно будет определить локальный канал, где происходит все волшебство и, поскольку локальные каналы обычно используются аналогично подпрограммам - нам нравится называть и находить их в диалплане с подпрограммами, с именем контекста, начинающимся с `local` (сродни тому, как подпрограммы начинаются с `sub`). Если вы создавали свой диалплан вместе с книгой, то заметите, что у вас уже есть локальный канал `[localDialDelay]`. Добавьте этот код где-нибудь в этой части диалплана.

```
[localMemberConnector]
exten => _[A-Za-z0-9].,1,Verbose(2,Connect ${CALLERID(all)} to Agent at ${EXTEN})
   ; отфильтровать все плохие символы, разрешить буквенно-цифровые символы и дефис
   same => n,Set(QueueMember=${FILTER(A-Za-z0-9-,${EXTEN})})
   ; назначить первое поле QueueMember технологии; дефис в качестве разделителя
   same => n,Set(Technology=${CUT(QueueMember,-,1)})
   ; назначить второе поле QueueMember устройству, дефис в качестве разделителя
   same => n,Set(Device=${CUT(QueueMember,-,2)})
   ; вызов агента
   same => n,Dial(${Technology}/${Device})
   same => n,Hangup()
```

Этот код, возможно, еще не имеет полного смысла, но то, что он делает - это берет `${EXTEN}` (который на данный момент является сложной буквенно-цифровой строкой) и нарезает его для извлечения фактически вызываемого канала (т.е. мы передаем как часть локального канала всю информацию, необходимую для набора фактического канала).

Давайте посмотрим на код `AddQueueMember` и посмотрим, сможем ли мы придать ему больше смысла:

```
exten => *740,1,Noop(Logging in device ${CHANNEL(endpoint)} into the support queue)
   same => n,Set(MemberTech=${CHANNEL(channeltype)})
   same => n,Set(MemberIdent=${CHANNEL(endpoint)})
   same => n,Set(Interface=${MemberTech}/${MemberIdent})
   ;;; СЛЕДУЮЩЕЕ ДОЛЖНО БЫТЬ ВСЕ В ОДНУ СТРОКУ
   same => n,AddQueueMember(support,Local/${MemberTech}-${MemberIdent}@localMemberConnector
,,,${IF($[${MemberTech} = PJSIP]?${Interface})})
   same => n,Playback(silence/1)
```

```
    same => n,Playback(${IF($[${AQMSTATUS} = ADDED]?agent-loginok:agent-incorrect)})
    same => n,Hangup()
```

Как только вы все это введете и перезагрузите свой диалплан - войдите в очередь, набрав *740, и давайте посмотрим, что у нас есть.

```
*CLI> queue show support

support has 0 calls (max unlimited) in 'rrmemory' strategy (1s holdtime, 0s talktime),
W:0, C:1, A:1, SL:0.0%, SL2:0.0% within 0s
   Members:
      PJSIP/SOFTPHONE_A (Local/PJSIP-SOFTPHONE_A@localMemberConnector)
(ringinuse disabled) (dynamic) (Not in use)
   No Callers
```

Теперь участник очереди идентифицируется как локальный канал с именем PJSIP-SOFTPHONE_A в контексте [localMemberConnector]. (Канал PJSIP/SOFTPHONE_A будет отслеживаться на предмет фактического состояния конечной точки.) Когда Queue() решает послать вызов участнику - вызов будет в конечном итоге в контексте [localMemberConnector], где EXTEN (PJSIP-SOFTPHONE_A) будет порезан вдоль и поперёк, чтобы получить наш тип канала и конечную точку[7], которая фактически будет вызвана.

На данном этапе цель всей этой дополнительной сложности ясна не сразу. Пока что мы не получаем ничего полезного от всего этого дополнительного кода.

Итак, теперь, когда мы можем добавлять устройства в очередь, используя локальные каналы, давайте посмотрим, как это может быть полезно.

Допустим, у нас есть клиент, который просто не выносит нашего лучшего агента. Он хороший клиент - поэтому мы не хотим его потерять, но это наш лучший агент и мы не собираемся его увольнять.

Чтобы настроить это - мы собираемся назначить CallerID для SOFTPHONE_B, чтобы у нас было с чем сравнить.

```
MySQL> UPDATE `asterisk`.`ps_endpoints` SET callerid='SOFTPHONE_B <103>' \
WHERE id='SOFTPHONE_B';
```

Мы собираемся встроить небольшую хитрость в наш диалплан, которая будет отклонять вызов агенту, если CallerID соответствует нашему чувствительному клиенту.

```
[localMemberConnector]
exten => _[A-Za-z0-9].,1,Verbose(2,Connect ${CALLERID(all)} to Agent at ${EXTEN})
 same => n,Wait(0.1) ; Prevent loop from completely hogging CPU
 same => n,Set(QueueMember=${FILTER(A-Za-z0-9-_,${EXTEN})}) ; allow alphanum, - , _
 same => n,Set(Technology=${CUT(QueueMember,-,1)}) ; first field, hyphen is separator
 same => n,Set(Device=${CUT(QueueMember,-,2)}) ; second field, hypen separator
 ; is this our mismatched pair?
 same => n,DumpChan()
 same => n,Noop(${CALLERID(all)} : ${Device})
 same=>n,GotoIf($["${CALLERID(num)}"="103"&"${Device}"="SOFTPHONE_A"]?rejectcall:ringagent)
 ; dial the agent
 same => n(ringagent),Dial(${Technology}/${Device})
 same => n,Hangup()
 ; send it back!
 same => n(rejectcall),Congestion()
 same => n,Hangup()
```

Передача Congestion() приведет к тому, что вызывающий будет возвращен в очередь (пока это происходит - вызывающий не получает никаких признаков того, что что-то не так и продолжает слушать музыку пока на его вызов не ответит какой-то канал).[8] В идеале - ваша очередь запрограммирована на попытку вызова другого агента; однако, вы должны иметь в виду, что если app_queue определяет, что этот участник все еще является его приоритетным выбором при вызове -

7 Возможно, мы могли бы использовать / вместо - в качестве разделителя, что дало бы нам Local/PJSIP/SOFTPHONE_A@localMemberConnector, но мы чувствовали, что это может привести к странным синтаксическим ошибкам и окажется неудобным для фильтрации и анализа, поэтому мы использовали -.

8 Очевидно, что не стоит использовать какой-либо код диалплана в локальном канале, отвечающий на канал, например, Answer(), Playback() и т.д.

вызов будет просто повторно подключен к тому же агенту (и снова получит перегрузку, и таким образом потенциально создаст логический цикл захвата процессора). Чтобы избежать этого - вам нужно будет убедиться, что очередь использует стратегию распределения, такую как `round_robin`, `random` или любую другую, которая гарантирует что один и тот же участник не будет вызван снова и снова. Именно поэтому мы добавляем крошечную небольшую задержку в наш контекст `[LocalMemberConnector]` так, что если цикл, подобный этому, действительно произойдет - в нем есть по крайней мере небольшой регулятор.

Давайте просто рассмотрим наш код. Установите для CallerID значение, отличное от `103` и вызов должен пройти.

```
MySQL> UPDATE `asterisk`.`ps_endpoints` SET callerid='SOFTPHONE_B <123>' \
WHERE id='SOFTPHONE_B';
```

Использование локальных каналов для ваших каналов участников не облегчит проектирование и отладку очереди, но даст вам гораздо больше власти над вашими очередями, чем простое использование `app_queue`, поэтому, если у вас большие требования к очереди - использование локальных каналов даст вам уровень контроля, который вы не имели бы в противном случае.

Статистика очереди: файл queue_log

Файл *queue_log* (обычно расположенный в */var/log/asterisk*) содержит совокупную информацию о событиях для очередей, определенных в вашей системе (например, когда очередь перезагружается, когда участники очереди добавляются или удаляются, события паузы/возобновления и т.д.), а также некоторые сведения о вызовах (например, их статус и каналы, к которым были подключены абоненты). Журнал очередей включен по умолчанию, но им можно управлять с помощью файла */etc/asterisk/logger.conf*. Есть три параметра, связанных с файлом *queue_log*, в частности:

`queue_log`

Определяет, включен ли журнал очередей или нет. Допустимые значения `yes` или `no` (по умолчанию - `yes`).

`queue_log_to_file`

Определяет, следует ли записывать журнал очереди в файл, даже если имеется серверная часть в realtime. Допустимые значения - `yes` или `no` (по умолчанию - `no`).

`queue_log_name`

Управляет именем журнала очереди. По умолчанию это `queue_log`.

Журнал очереди представляет собой разделенный на каналы список событий. Поля в файле *queue_log*:

- Метка времени UNIX Epoch
- Уникальный идентификатор звонка
- Имя очереди
- Имя соединительного канала (bridged chanel)
- Тип события
- Пусто или дополнительные параметры события

Информация, содержащаяся в параметрах события, зависит от типа события. Типичный файл *queue_log* будет выглядеть примерно так:

```
1530389309|NONE|NONE|NONE|QUEUESTART|
1530409313|CLI|support|PJSIP/SOFTPHONE_B|ADDMEMBER|
1530409467|CLI|support|PJSIP/SOFTPHONE_B|REMOVEMEMBER|
1530409666|NONE|support|PJSIP/SOFTPHONE_B|PAUSE|Callbacks
1530411108|NONE|support|PJSIP/SOFTPHONE_B|UNPAUSE|FinishedCallbacks
1530440239|1530440239.10|support|PJSIP/SOFTPHONE_A|ADDMEMBER|
1530440303|1530440303.16|support|PJSIP/SOFTPHONE_A|REMOVEMEMBER|
1530497165|1530497165.54|support|Local/PJSIP-SOFTPHONE_A@MemberConnector|ADDMEMBER|
```

```
1530497388|CLI|support|Local/PJSIP-SOFTPHONE_A@MemberConnector|REMOVEMEMBER|
1530497408|1530497408.60|support|Local/PJSIP-SOFTPHONE_A@localMemberConnector|ADDMEMBER|
1530497506|1530497506.71|support|NONE|ENTERQUEUE||SOFTPHONE_B|1
1530497511|1530497506.71|support|PJSIP/SOFTPHONE_A|CONNECT|5|1530497506.72|4
1530497517|1530497506.71|support|PJSIP/SOFTPHONE_A|COMPLETEAGENT|5|6|1
1530509861|1530509861.134|support|NONE|ENTERQUEUE||SOFTPHONE_B|1
1530509864|1530509861.134|support|PJSIP/SOFTPHONE_A|RINGCANCELED|2224
1530509864|1530509861.134|support|NONE|ABANDON|1|1|3
1530510503|1530510503.156|support|NONE|ENTERQUEUE||103|1
1530510503|1530510503.156|support|PJSIP/SOFTPHONE_A|RINGNOANSWER|0
1530510511|1530510503.156|support|NONE|ABANDON|1|1|8
1530510738|1530510738.163|support|NONE|ENTERQUEUE||123|1
1530510742|1530510738.163|support|PJSIP/SOFTPHONE_A|CONNECT|4|1530510738.164|4
1530510752|1530510738.163|support|PJSIP/SOFTPHONE_A|COMPLETECALLER|4|10|1
```

Как видно из этого примера - уникальный идентификатор события может существовать не всегда. Внешние службы, такие как Asterisk CLI, могут выполнять действия в очереди, и в этих случаях вы увидите что-то вроде CLI в поле уникального идентификатора.

Доступные события и информация, которую они предоставляют, описаны в Таблице 12-7.

Таблица 12-7. События в журнале очереди Asterisk

Событие	Предоставленная информация
ABANDON	Пишется когда абонент в очереди вешает трубку до того, как на его звонок ответит агент. Для ABANDON предусмотрены три параметра: положение вызывающего абонента при отбое, исходное положение вызывающего абонента при входе в очередь и время ожидания абонента до отбоя.
ADDMEMBER	Пишется при добавлении участника в очередь. Имя соеденительного канала будет заполнено названием канала, добавленного в очередь.
AGENTDUMP	Указывает что агент повесил трубку на абоненте во время воспроизведения объявления очереди, прежде, чем они были соединены вместе.
AGENTLOGIN	Пишется при входе агента в систему. Поле `bridged channel` будет содержать что-то вроде `Agent/9994`, если войти в систему с помощью `chan_agent`, а поле первого параметра будет содержать вошедший канал (например, `SIP/0000FFFF0001`).
AGENTLOGOFF	Пишется когда агент выходит из системы, вместе с параметром, указывающим, как долго агент входил в систему. Обратите внимание, что поскольку вы часто будете использовать `RemoveQueueMember()` для выхода из системы - этот параметр может не записываться. Вместо него смотрите событие REMOVEMEMBER.
COMPLETEAGENT	Пишется когда вызов соединяется с оператором и агент вешает трубку, наряду с параметрами, указывающими время, в течение которого вызывающий абонент удерживался в очереди, продолжительность вызова с оператором и исходное положение, в котором вызывающий абонент вошел в очередь.
COMPLETECALLER	То же, что COMPLETEAGENT, за исключением того, что вызывающий абонент повесил трубку, а не агент.
CONFIGURELOAD	Указывает, что конфигурация очереди была перезагружена (например, через *module reload app_queue.so*).
CONNECT	Пишется при соединении абонента и агента. Записываются также три параметра: время ожидания абонента в очереди, уникальный идентификатор канала участника очереди, к которому был подключен абонент, и время, в течение которого телефон участника очереди звонил до получения ответа.
ENTERQUEUE	Записывается при входе абонента в очередь. Также записываются два параметра: URL (если указан) и идентификатор вызывающего абонента.
EXITEMPTY	Пишется когда вызывающий объект удаляется из очереди из-за отсутствия агентов, доступных для ответа на вызов (как указано параметром `leavewhenempty`). Записываются также три параметра: положение вызывающего абонента в очереди, исходное положение, с которым абонент вошел в очередь и время, в течение которого вызывающий абонент находился в очереди.

Событие	Предоставленная информация
EXITWITHKEY	Пишется когда вызывающий абонент выходит из очереди, нажав одну клавишу DTMF на своем телефоне чтобы выйти из очереди и продолжить в диалплане (как разрешено параметром `context` в *queues.conf*). Записываются четыре параметра: ключ, используемый для выхода из очереди, позиция вызывающего абонента в очереди при выходе, исходная позиция, с которой абонент вошел в очередь и количество времени, в течение которого вызывающий абонент ожидал в очереди.
EXITWITHTIMEOUT	Пишется когда вызывающий удален из очереди из-за `timeout`, указаного параметром `timeout` для `Queue()`. Также записываются три параметра: положение, в котором находился вызывающий абонент при выходе из очереди, исходное положение вызывающего абонента при входе в очередь и количество времени, в течение которого вызывающий абонент ожидал в очереди.
PAUSE	Пишется когда участник очереди приостановлен.
PAUSEALL	Пишется когда все участники очереди приостановлены.
UNPAUSE	Пишется когда участник очереди возобновлён.
UNPAUSEALL	Пишется когда все участники очереди возобновлены.
PENALTY	Пишется при изменении пенальти участника. Пенальти может быть изменен несколькими способами, такими как функция `QUEUE_MEMBER_PENALTY()`, Asterisk Manager Interface или командой Asterisk CLI.
REMOVEMEMBER	Пишется когда участник очереди удаляется из очереди. Поле bridge channel будет содержать имя участника, удаленного из очереди.
RINGNOANSWER	Пишется когда участник очереди звонит в течение определенного периода времени и превышается значение времени ожидания для вызова участника очереди. Также будет записан один параметр, указывающий время, в течение которого звонил добавочный номер участника.
TRANSFER	Записывается при переходе абонента на другой добавочный номер. Также записываются дополнительные параметры, включающие расширение и контекст, в который был передан абонент, время удержания абонента в очереди, количество времени, в течение которого вызывающий абонент разговаривал с участником очереди и исходное положение вызывающего абонента, когда он вошел в очередь.[a]
SYSCOMPAT	Записывается если агент пытается ответить на вызов, но не удается установить вызов из-за несовместимости в настройке медиапотока.

[a] Обратите внимание, что при передаче вызывающего абонента с использованием SIP-трансфера (а не встроенных трансферов, запускаемых DTMF и настраиваемых в *features.conf*), событие `TRANSFER` может не записываться.

Вывод

Мы начали эту главу с рассмотрения основных очередей вызовов, обсуждения того, что они из себя представляют, как работают и когда вы, возможно, захотите их использовать. После создания простой очереди мы изучили как управлять участниками очереди с помощью различных средств (включая использование локальных каналов, обеспечивающих возможность выполнения некоторой логики диалплана непосредственно перед подключением к участнику очереди). Конечно, нам нужна возможность отслеживать что делают наши очереди - поэтому мы быстро просмотрели файл *queue_log* и различные поля, записанные в результате событий, происходящих в наших очередях.

Благодаря информации, представленной в этой главе, вы обладаете большинством базовых знаний, необходимых для реализации очередей в Asterisk.

Глава 13. Состояния устройств

Среди беспорядка найдите простоту.
– Альберт Эйнштейн

Часто бывает полезно иметь возможность определить состояние устройств, подключенных к телефонной системе. Например, оператору в приемной может потребоваться видеть статусы всех сотрудников офиса, чтобы определить, может ли кто-то принять телефонный звонок. Сама Asterisk нуждается в такой же информации. В качестве другого примера, если вы создали очередь вызовов, как описано в Главе 12, Asterisk должна знать, когда агент становится доступен, чтобы можно было отправить другой вызов. В этой главе рассматриваются понятия состояний устройств в Asterisk, а также способы использования и доступа устройств и приложений к этой информации.

Состояния устройств

Существует две категории устройств, для которых Asterisk предоставляет информацию о состоянии: канальные устройства (такие как конечные точки PJSIP) и виртуальные (являющиеся встроенными службами, которые можно отслеживать, например конференц-залы).

Чтобы ссылаться на состояние *канала* - Вы делаете это точно так же, как с `Dial()`, например `DEVICE_STATE(PJSIP/000f300B0B02)`, тогда как для ссылки на состояние *виртуального устройства* используется формат *`тип виртуального устройства:идентификатор`*, например `DEVICE_STATE(ConfBridge:1234)`.

Виртуальные устройства включают вещи, находящиеся внутри Asterisk, но предоставляющие полезную информацию о состоянии.

Таблица 13-1. Устройства, для которых Asterisk может предоставлять информацию о состоянии

Устройство	Описание
`PJSIP/`*`channel name`*	Многие каналы позволяют контролировать их состояние, но канал PJSIP предлагает на сегодняшний день наибольшее количество полезных данных; таким образом, мониторинг SIP-устройств является наиболее распространенным использованием `DEVICE_STATE`.
`ConfBridge:`*`conference bridge`*	Состояние конференц-моста MeetMe. Состояние будет отражать, имеются ли в настоящее время участники в конференц-зале. Более подробную информацию об использовании `MeetMe()` для конференц-связи можно найти в Главе 11.
`Custom:`*`custom name`*	Пользовательские состояния устройств. Эти состояния имеют пользовательские имена и изменяются с помощью функции `DEVICE_STATE()`. Пример использования можно найти в разделе "Использование пользовательских состояний устройств".
`Park:`*`exten@context`*	Состояние слота парковки вызова. Информация о состоянии будет отражать, припаркован ли вызывающий абонент в данный момент на этом добавочном номере. Дополнительную информацию о парковке вызовов в Asterisk можно найти в "Парковке вызовов".
`Calendar:`*`calendar name`*	Состояние календаря. Asterisk будет использовать содержимое названного календаря, чтобы установить состояние `available` или `busy`.

Проверка состояний устройств

Функция диалплана `DEVICE_STATE()` считывает текущее состояние устройства.

```
exten => 7012,1,Answer()
   same => n,Set(DeviceIdent=PJSIP/000f300B0B02)
   same => n,Verbose(3,${DeviceIdent} is ${DEVICE_STATE($DeviceIdent)})
   same => n,Hangup()
```

Если мы вызываем добавочный номер 7012 с того же устройства, на котором проверяем состояние, в консоли Asterisk появляется следующее подробное сообщение:

```
-- PJSIP/000f300B0B02 is INUSE
```

В Главе 17 обсуждается Asterisk Manager Interface (AMI). Действие диспетчера `GetVar` можно использовать для получения значения состояния устройства из внешней программы. Вы можете использовать его для получения значения обычной переменной или для возврата значения из функции диалплана, например `DEVICE_STATE()`.

Вот список значений, которые вернет функция `DEVICE_STATE()` (в зависимости, конечно, от того, что было найдено):

- `UNKNOWN`
- `NOT_INUSE`
- `INUSE`
- `BUSY`
- `INVALID`
- `UNAVAILABLE`
- `RINGING`
- `RINGINUSE`
- `ONHOLD`

Эта информация может затем использоваться в диалплане для принятия решений о потоке вызовов (например, локальный канал вызова агента может использовать эту информацию для определения того, что телефон агента находится на вызове по другой линии и, таким образом, отклонить вызов, чтобы тот вернулся в очередь).

Состояния номеров с использованием директивы hint

Состояние внутреннего номера - это механизм диалплана, который Asterisk использует, чтобы разрешить SIP-устройствам подписываться на информацию о присутствии. Например, телефон в приемной может иметь модуль Busy Lamp Field (BLF), содержащий кнопки, которые будут использоваться для отображения состояний различных телефонов в офисе. Телефон с BLF будет отправлять запросы на подписку, чтобы сообщить Asterisk, с каких устройств он хочет получать информацию о присутствии. В диалплане мы используем директиву `hint` для определения сопоставления между расширением и одним или несколькими устройствами.

Хинты

Чтобы определить хинт в диалплане, вместо приоритета используется ключевое слово `hint`.

```
[hints]
;exten = <extension>,hint,<device state id>[& <more dev state id],<presence state id>

exten => 100,hint,${UserA_DeskPhone}

exten => 221,hint,ConfBridge:221
```

Часто вы можете увидеть хинты, определенные в том же разделе диалплана, что и обычные расширения. Это может сделать диалплан немного визуально загроможденным и это также предполагает, что хинт так или иначе связан с набираемым добавочным номером, что на самом деле не так.

```
[sets]

exten => 100,hint,${UserA_DeskPhone}
exten => 100,1,Gosub(subDialUser,${EXTEN},1(${UserA_DeskPhone},${EXTEN},default,22))
exten => 101,hint,${UserA_SoftPhone}
exten => 101,1,Gosub(subDialUser,${EXTEN},1(${UserA_SoftPhone},${EXTEN},default,23))
exten => 102,hint,${UserB_DeskPhone}
exten => 102,1,Gosub(subDialUser,${EXTEN},1(${UserB_DeskPhone},${EXTEN},default,26))
exten => 103,hint,${UserB_SoftPhone}
exten => 103,1,Gosub(subDialUser,${EXTEN},1(${UserB_SoftPhone},${EXTEN},default,24))

exten => 110,1,Dial(${UserA_DeskPhone}&${UserA_SoftPhone}&${UserB_SoftPhone})
```

В нашем примере мы сделали прямую взаимосвязь между добавочным номером hint`а и набираемым добавочным номером, хотя этого не требуется.

Проверка состояний внутреннего номера

Самый простой способ проверить текущее состояние хинтов расширений - через CLI Asterisk. Команда `core show hints` отобразит все настроенные в данный момент хинты:

```
*CLI> core show hints
    -= Registered Asterisk Dial Plan Hints =-
100@hints : PJSIP/0000f30A0A01   State:Unavailable Presence:not_set Watchers 0
101@hints : PJSIP/SOFTPHONE_A    State:Unavailable Presence:not_set Watchers 0
102@hints : PJSIP/0000f30B0B02   State:Unavailable Presence:not_set Watchers 0
103@hints : PJSIP/SOFTPHONE_B    State:Unavailable Presence:not_set Watchers 0
110@hints : PJSIP/0000f30A0A01&P State:Unavailable Presence:not_set Watchers 0
221@hints : ConfBridge:221       State:Unavailable Presence:not_set Watchers 0
----------------
- 6 hints registered
```

В дополнение к отображению состояния каждого хинта, вывод `core show hints` также выводит количество наблюдателей. *Наблюдатель (watcher)* - это объект, который подписался на получение обновлений о состоянии этого расширения. Если конечная точка SIP подписывается на состояние добавочного номера - количество наблюдателей будет увеличено.

Состояние внутреннего номера также может быть получено с помощью функции диалплана `EXTENSION_STATE()`. Эта функция работает так же, как `DEVICE_STATE()`, описанная в предыдущем разделе. Добавьте следующий пример в файл */etc/asterisk/extensions.conf* как новый номер сразу после 235:

```
exten => 234,1,NoOp()
  same => n,Set(FEATURE(parkingtime)=60)

exten => 235,1,Noop(The state of 100@hints is ${EXTENSION_STATE(100@hints)} )
  same => n,Hangup()

exten => 321,1,NoOp()
```

Когда этот номер вызывается из конечной точки с номером 100 - в консоли Asterisk отобразится следующее сообщение:

```
 -- The state of 100@hints is INUSE
```

В следующем списке перечислены возможные значения, которые могут быть возвращены функцией `EXTENSION_STATE()`:

- `UNKNOWN`
- `NOT_INUSE`
- `INUSE`
- `BUSY`
- `UNAVAILABLE`
- `RINGING`
- `RINGINUSE`

- HOLDINUSE
- ONHOLD

SIP-присутствие

Asterisk дает устройствам возможность подписаться на состояние расширения с использованием протокола SIP. Эту функцию часто называют BLF (Busy Lamp Field), см. Рисунок 13-1.

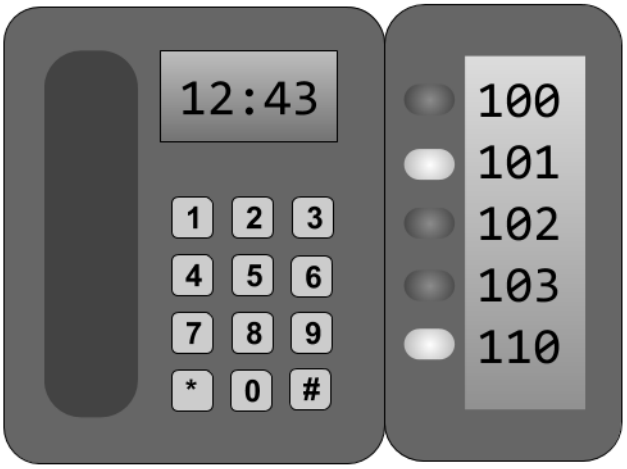

Рисунок 13-1. Busy Lamp Field или боковая панель

Конфигурация модуля будет немного (или очень) отличаться для каждого производителя; однако информация о подписке, так или иначе, должна включать следующее:

- Адрес сервера Asterisk (это может быть определено для каждой кнопки или может применяться ко всему телефону).
- Контекст для подписки (в нашем примере диалплана он называется `[hints]`). Этот параметр определяется в поле `subscribe_context` таблицы `asterisk.ps_endpoints`.
- Соответствующий добавочный номер (100, 101, 102 и т.д.)[1]

Один из самых простых и недорогих способов тестирования присутствия - использование opensource софтфона для Windows - MicroSIP.[2] Сначала вам нужно скачать MicroSIP и зарегистрировать его в вашей системе Asterisk. Затем на вкладке контактов софтфона вы можете щелкнуть правой кнопкой мыши в открытой области чтобы добавить контакт. В разделе «Имя» вы можете указать все что пожелаете, но в разделе «Номер» вы должны ввести *`extension`*`@`*`hints_context`*, который в нашем случае будет одним из `100@hints`, `101@hints`, `102@hints` или `103@hints`. Если вы настроили все в Asterisk согласно предыдущим примерам, то должны увидеть, как состояние ваших подписок меняется в зависимости от того, что делает устройство. Вы также можете отслеживать это с точки зрения Asterisk с помощью такой команды:

```
$ watch -n 0.5 "sudo asterisk -rx 'core show hints'"
```

Конфигурация присутствия на физических настольных телефонах по существу одинакова, но может быть сложнее понять конкретный синтаксис, который требуется каждому производителю. Наш совет - заставить его работать с MicroSIP (который вы сможете запустить на WINE под Linux или macOS). Это простая установка и оттуда у вас будет отличная конфигурация, которой вы можете доверять, когда выбираете аналогичную конфигурацию для одного из ваших настольных телефонов.

Использование пользовательских состояний устройств

В дополнение к устройствам, которые Asterisk знает как внутренне контролировать (`PJSIP`, `ConfBridge`, `Park`, `Calendar`), Asterisk также предоставляет возможность создавать собственные

1 Элементы 2 и 3 могут быть сформированы в виде одной строки, похожей на `100@hints` или что-то подобное.
2 Который написан с использованием той же библиотеки PJSIP, что использует Asterisk.

состояния устройств, которые могут быть очень полезны при разработке некоторых интересных приложений.

Пользовательские состояния устройств определяются с помощью префикса `Custom:`. Текст, который идет после префикса, может быть чем угодно. Чтобы установить или прочитать значение пользовательского состояния устройства - используйте функцию диалплана `DEVICE_STATE()`. Поместите это в свой файл *extensions.conf* сразу после расширения 235:

```
exten => 235,1,Noop(The state of 100@hints is ${EXTENSION_STATE(100@hints)} )
  same => n,Hangup()

exten => 236,1,Noop(Set a custom status)
  same => n(blink),Set(DEVICE_STATE(Custom:rudolph)=UNAVAILABLE)
  same => n,Set(DEVICE_STATE(Custom:santa)=NOT_INUSE)
  same => n,Wait(0.75)
  same => n,Set(DEVICE_STATE(Custom:rudolph)=NOT_INUSE)
  same => n,Set(DEVICE_STATE(Custom:santa)=UNAVAILABLE)
  same => n,Wait(0.75)
  same => n,Goto(blink)
```

Затем добавьте это себе в контекст `[hints]`:

```
exten => 221,hint,ConfBridge:221
exten => santa,hint,Custom:santa
exten => rudolph,hint,Custom:rudolph
```

Весело, да?

Вы заметите, что когда повесите трубку - одно из пользовательских состояний устройств останется "Unavailable". Это важный момент: в системе нет ничего, что обновляло бы ваши пользовательские состояния устройства, если вы сами не внедрили что-то для этого.

Вывод

Функциональность состояний устройств в Asterisk может использоваться для отслеживания состояния различных ресурсов и доставки информации об этих состояниях различным подписчикам. Обычно (и традиционно) используемое для BLF, пользовательское состояние устройства позволяет этому ресурсу быть гораздо более гибким, чем в традиционной УАТС.

Глава 14. Автосекретарь

Я не отвечаю на звонки. У меня всегда такое чувство, что на другом конце провода кто-то есть.
– Фред Коуплз

Во многих УАТС, как правило, есть система меню для автоматического ответа на входящие вызовы, позволяющая абонентам направлять себя на различные расширения и ресурсы в системе с помощью выбора меню. Эта система известна в телекоммуникационной отрасли как *автосекретарь* (АС). АС, как правило, предоставляет следующие возможности:

- Перевод на добавочный номер
- Перевод на голосовую почту
- Переход в очередь
- Воспроизведение сообщения (например “наш адрес…”)
- Подключение к подменю (например “для получения списка наших отделов...”)
- Соединение с приемной
- Повтор выбора

Для всего остального, особенно если требуется внешняя интеграция, например поиск по базе данных, обычно требуется интерактивное голосовое меню (сокращенно на английском - IVR).

АС - это не IVR

В open source телеком-сообществе вы часто услышите термин IVR, используемый для описания автосекретаря.[1] Однако в телекоммуникационной отрасли в течение многих десятилетий до появления VoIP или УАТС с открытым исходным кодом IVR отличался от АС. По этой причине, когда вы говорите с кем-то, имеющим многолетний опыт работы в области телекоммуникаций, о любом виде телекоммуникационного меню, вы должны убедиться, что говорите об одном и том же. Для специалиста по телекоммуникациям термин *IVR* подразумевает относительно комплексную и сложную в разработке (и последующе затратную), в то время как АС - это простая и недорогая вещь, которая является общей для большинства АТС.

В этой главе мы поговорим о создании автосекретаря. IVR мы обсудим в Главе 16.[2]

Проектирование вашего АС

Самая распространенная ошибка новичков при проектировании АС - излишняя сложность. В то время как может быть много радости и чувства выполненного долга в создании многоуровневого АС с десятками отличных вариантов и кучами действительно интересных подсказок - ваши абоненты имеют другую повестку дня. Люди звонят по телефону в первую очередь потому, что хотят с кем-то поговорить. В то время как люди привыкли к реальности автосекретарей (и в некоторых случаях они могут ускорить процесс), по большей части люди предпочитают говорить с кем-то живым. Это означает, что есть два основных правила, которых должен придерживаться каждый АС:

- Быть проще.

1 Это, скорее всего, потому, что "IVR" гораздо легче сказать, чем "автосекретарь".
2 Следует отметить, что Asterisk - это отличный инструмент для создания IVR. Но также отлично подходит и для создания автосекретаря.

- Убедитесь, что вы всегда делаете обработчик для людей, которые собираются нажимать 0, когда слышат меню. Если вы не хотите иметь опцию 0 - имейте в виду, что многие люди будут оскорблены этим, они повесят трубку и не перезвонят. В бизнесе это вообще плохо.

Прежде чем вы начнете кодировать свой АС - разумно его спроектировать. Вам нужно будет определить поток вызовов и необходимо будет указать подсказки, которые будут воспроизводиться на каждом шаге. Для этого могут быть полезны программные инструменты для построения диаграмм, но нет необходимости фантазировать. Таблица 14-1 предоставляет хороший шаблон для базового АС, который будет делать то, что вам нужно.

Таблица 14-1. Базовый автосекретарь

Шаг или выбор	Образец приглашения	Примечание	Имя файла[a]
Приветствие - рабочее время	Спасибо,что позвонили в компанию ABC.	Дневное приветствие. Воспроизводится сразу после того, как система отвечает на вызов.	*daygreeting.wav*
Приветствие - нерабочеее время	Спасибо,что позвонили в компанию ABC. Наш офис сейчас закрыт.	Ночное приветствие. Как и выше, но играет в нерабочее время.	*nightgreeting.wav*
Главное меню	Если вы знаете внутренний номер оператора, с которым хотите связаться, пожалуйста, введите его сейчас. Для свзи с отделом продаж, пожалуйста, нажмите 1; с отделом обслуживания - нажмите 2; для нашего каталога компании, нажмите #. Для получения информации о нашем адресе и факсе - нажмите кнопку 3. Чтобы повторить это сообщение - нажмите 9, или оставайтесь на линии, или нажмите 0, чтобы связаться с оператором.	Подсказка главного меню. Играет сразу после приветствия. Для вызывающего абонента приветствие и главное меню звучат как единое приглашение; однако в системе полезно держать эти подсказки раздельно.	*mainmenu.wav*
1	Пожалуйста, подождите пока мы переключим ваш звонок.	Перевод на очередь sales.	*holdwhileweconnect.wav*
2	Пожалуйста, подождите пока мы переключим ваш звонок.	Перевод на очередь support.	*holdwhileweconnect.wav*
#	n/a	Запуск приложения `Directory()`	n/a
3	Наш адрес [address]. Наш номер факса [fax number], и тд.	Воспроизведение записи, содержащей информацию об адресе и факсе. Возврат вызывающего абонента к подсказке меню после завершения.	*faxandaddress.wav*
0	Соединяем со специалистом. Пожалуйста, подождите.	Перевод на приемную/оператора.	*transfertoreception.wav*
9	n/a	Повтор. Воспроизведение подсказки меню (но не приветствия).	n/a
t	n/a	Тайм-аут. Если абонент не сделал выбора - считайте, что он набрал 0 (или в некоторых случаях повторите подсказку).	n/a

Шаг или выбор	Образец приглашения	Примечание	Имя файла[a]
i	Вы сделали неверный выбор. Пожалуйста, попробуйте еще раз.	Абонент нажал неверную цифру: воспроизведение подсказки меню (но не приветствия).	*invalid.wav*
_XXX[b]	n/a	Перевод вызова на набранный номер.	*holdwhileweconnect.wav*

[a] Эти файлы пока нигде не существуют. Мы используем их в качестве примеров.
[b] Это совпадение шаблонов должно соответствовать вашему диапазону внутренних номеров.

Давайте рассмотрим различные компоненты этого шаблона. Затем мы покажем вам код диалплана, необходимый для его реализации, а также как создавать подсказки.

Приветствие

Первое, что слышит абонент - это на самом деле две подсказки.

Первая подсказка - это приветствие. Единственное, что приветствие должно сделать - это поприветствовать абонента. Примером приветствия может быть “Спасибо, что позвонили Брайанту, Ван Меггелену и партнерам”, “Добро пожаловать в школу мудрости и дизайна футболок Лейфа” или “Вы позвонили в офисы адвокатов Дьюи, Читама и Хоуи.” Вот именно - выбор для звонящего будет позже. Это позволяет записывать различные приветствия без необходимости записывать все новое меню. Например, в течение нескольких недель каждый год вы можете делать, чтобы ваше приветствие говорило "приветствие сезона" или что-то еще, но ваше меню не нужно будет менять. Кроме того, если вы хотите воспроизвести другую запись в нерабочее время (“Спасибо за звонок. Наш офис сейчас закрыт.”), то можете использовать разные приветствия, но сердце меню останется прежним. Наконец, если вы хотите иметь возможность вернуть абонентов в меню из другой части системы, то не захотите, чтобы они снова услышали приветствие.

Главное меню

Приглашение главного меню - это место, где вы сообщаете своим абонентам о доступных им вариантах выбора. Вы должны проговаривать его как можно быстрее (без спешки или глупостей).[3] Когда вы записываете выбор - *всегда сообщайте пользователям о действии, которое будет предпринято, прежде чем дать им возможность выполнить это действие с помощью набора цифр.* Поэтому не говорите "нажмите 1 для продаж", а лучше скажите: "для продаж нажмите 1". Причина этого заключается в том, что большинство людей не обратят должного внимания на подсказку, пока не услышат интересующий их выбор. Как только они услышат свой выбор, то будут в полном внимании и им можно будет сообщить какую кнопку нажать, чтобы направить их туда - куда требуется.

Еще один момент, который следует рассмотреть - это порядок, в котором следует разместить выбор. Типичный бизнес, например, захочет, чтобы продажи были первым пунктом меню и большинство абонентов также будут ожидать этого. Самое главное - думать о своих клиентах. Например, большинство людей не будут заинтересованы в адресной и факсимильной информации - поэтому не делайте их первыми пунктами.[4] Подумайте о том, чтобы как можно быстрее соединить абонентов с местом их назначения когда вы проектируете свой выбор. Безжалостно режьте все, что не является абсолютно необходимым.

3 При необходимости вы можете использовать программу для редактирования звука, такую как Audacity, чтобы удалить тишину и даже немного ускорить запись.

4 На самом деле, мы обычно не рекомендуем это в АС, потому что это добавляется к тому, что абонент должен слушать, и большинство людей все равно пойдет на сайт для получения такого рода информации.

Выбор 1

Вариант 1 в нашем примере будет простым переводом. Обычно это ресурс, расположенный в другом контексте, и он, как правило, имеет внутренний добавочный номер, так что внутренние пользователи могут также передавать вызовы на него. В этом примере мы будем использовать эту опцию для отправки абонентов в очередь `sales`, созданную в Главе 12.

Выбор 2

Вариант 2 будет технически идентичен варианту 1. Только место назначения будет другим. Этот выбор переместит абонентов в очередь `support`.

Выбор

Хорошо иметь возможность выбора справочника как можно ближе к началу записи. Многие люди будут использовать каталог, если знают что тот существует, но им не требуется прослушивать все меню, чтобы узнать о нем. Нетерпеливые люди будут нажимать 0 - поэтому чем раньше вы расскажете им о справочнике, тем больше шансов, что они им воспользуются, и тем самым уменьшится нагрузка на вашего секретаря.

Выбор 3

Если у вас есть опция, которая ничего не делает, кроме воспроизведения записи абоненту (например, адрес и факс), вы можете оставить весь код для неё в том же контексте, что и меню, и просто вернуть абонента в главное меню в конце записи. В общем, такие варианты не очень полезны, как нам хотелось бы думать - поэтому в большинстве случаев вы, вероятно, захотите оставить её.

Выбор 9

Очень важно дать звонящему возможность услышать подсказку еще раз. Многие люди не будут обращать внимания на все меню, и если вы не дадите им возможность услышать меню снова - они, скорее всего, нажмут 0.

Обратите внимание, что вам не нужно воспроизводить приветствие снова - только подсказку главного меню.

Выбор 0

Как было сказано ранее - нравится вам это или нет, это выбор, который предпочтут многие (возможно, большинство) ваши абоненты. Если вы действительно не хотите чтобы кто-то отвечал на эти звонки, то можете отправлять вызовы в почтовый ящик, но мы не рекомендуем это делать. Если вы занимаетесь бизнесом - многие из ваших абонентов будут вашими клиентами. Вы же хотите, чтобы им было легко связаться с вами. Доверьтесь нам.

Тайм-аут

Многие люди будут звонить по номеру и не обращать слишком много внимания на то, что происходит. Они знают, что если просто будут ждать на линии, то в конце концов их переведут на оператора. Или, возможно, они находятся в автомобиле, и просто не могут нажимать кнопки своих телефонов. В любом случае - сделайте им одолжение. Если они не делают никакого выбора, не преследуйте их и не заставляйте их делать этого. Соедините их с оператором.

Invalid (Неверно)

Люди совершают ошибки. Это нормально. Обработчик неверных направлений сообщит им о том, что они выбрали - не является допустимым вариантом, и вернет их в приглашение меню, чтобы они

могли повторить попытку выбора. Обратите внимание, что вы не должны воспроизводить приветствие снова, только приглашение главного меню.

Вызов добавочного номера

Если кто-то звонит в вашу систему и знает добавочный номер, который хочет набрать, ваш автосекретарь должен иметь код для обработки этого.

Хотя Asterisk может обрабатывать перекрытие между вариантами меню и добавочными номерами (например, у вас может быть выбор меню 1 и расширения от 100 до 199) - лучше избегать этого перекрытия. В противном случае диалплану всегда придется ожидать межразрядный тайм-аут всякий раз, когда кто-то нажимает 1, потому что не будет знать - планируют ли набрать добавочный номер 123. Межразрядный тайм-аут - это задержка, которую система допускает между цифрами, прежде чем предполагает что было введено все число. Этот таймер гарантирует, что абоненты имеют достаточно времени для набора многоразрядного номера, но он также вносит задержку в обработку одноразрядных вводов.

Создание вашего АС

После того, как вы разработали свой АС, есть три вещи, которые вам нужно сделать чтобы заставить его работать должным образом:

- Запись подсказок.
- Создание диалплана для меню.
- Направление входящих каналов в контекст АС.

Начнем с того, что поговорим о записях.

Запись подсказок

Запись подсказок для телефонной системы является критически важной задачей. Это то, что ваши абоненты будут слышать при взаимодействии с вашей системой, и качество и профессионализм этих записей отразится на вашей организации.

Asterisk очень гибок в этом отношении и может работать со многими различными аудиоформатами. Мы обнаружили, что в целом наиболее лучшим форматом для использования является WAV. Файлы, сохраненные в этом формате, могут быть самыми разными, но только один тип WAV-файла будет работать с Asterisk: файлы должны быть закодированы в 16-битном, 8000 Гц, моно-формате.

Рекомендуемый формат файла подсказки

Формат WAV, который мы рекомендуем, удобен для системных подсказок, потому что он может быть легко преобразован в любой другой формат, который могут использовать ваши телефоны без потери или искажения, и почти любой компьютер может воспроизводить его без специального программного обеспечения. Таким образом, Asterisk может не только легко обрабатывать файл, но и легко будет работать с ним на ПК (что может быть полезно). Asterisk может обрабатывать и другие форматы файлов, и в некоторых случаях они могут быть более подходящими для ваших нужд, но в целом мы считаем, что 16-битные 8 кГц WAV-файлы являются самыми простыми в работе и, в большинстве случаев, наилучшего качества.

Есть два основных способа добавить подсказки в систему. Один из них заключается в записи звуковых файлов в студии или на ПК, а затем в перемещении этих файлов в систему. Второй способ

заключается в записи подсказок непосредственно в системе с помощью телефонного аппарата. Мы предпочитаем второй способ.

Наш совет таков: не зацикливайтесь на сложностях записи звука через ПК или в студии.[5] Это вообще не нужно. Телефон будет делать записи отличного качества, и причины этого просты: микрофон и электроника в телефоне тщательно проработаны для захвата человеческого голоса в формате, который идеально подходит для передачи по телефонным сетям, и поэтому телефон также идеально подходит для выполнения записи подсказок. Аппарат захватит звук в правильном формате, отфильтрует фоновый шум и нормализует уровень децибел.

Да, правильно подготовленная подсказка в студии будет лучше подсказки, записанной по телефону, но если у вас нет оборудования или опыта - воспользуйтесь нашим советом и используйте телефон для записи, потому что плохо подготовленная подсказка в студии будет намного хуже, чем подсказка, записанная по телефону.

Использование диалплана для создания записей

Самый простой способ записи подсказок - это использование приложения `Record()`.

Добавьте эту новую подпрограмму в нижнюю часть вашего файла *extensions.conf*:

```
[subRecordPrompt]
exten => 500,1,Playback(vm-intro)
   same => n,Record(daygreeting.wav)
   same => n,Wait(2)
   same => n,Playback(daygreeting)
   same => n,Hangup

exten => 501,1,Playback(vm-intro)
   same => n,Record(mainmenu.wav)
   same => ... etc ... (создайте код диалплана для каждой подсказки, которую необходимо
                        ; записать)
```

Чтобы использовать этот контекст - вам нужно будет включить его в контекст, в котором ваши устройства входят в диалплан. Поэтому в контекст `[LocalSets]` вы должны добавить строку `include=>UserServices`. В рабочей среде вам, вероятно, понадобится пароль, чтобы кто попало не записывали подсказки.

Эта подпрограмма воспроизводит запрос, издает звуковой сигнал, делает запись и после воспроизводит её.[6] Примечательно, что приложение `Record()` принимает в качестве аргумента все имя файла, в то время как приложение `Playback()` исключает расширение типа файла (*.wav*, *.gsm* и др.). Это происходит потому, что приложение `Record()` должно знать, в каком формате должна быть сделана запись, а `Playback()` - нет. Вместо этого `Playback()` автоматически выбирает наилучший доступный аудиоформат, основываясь на кодеке, используемом вашим телефоном и форматах, доступных в папке *sounds* (например, если у вас есть *daygreeting.wav* и *daygreeting.gsm* в папке *sounds* - `Playback(daygreeting)` выберет тот, который требует наименьших усилий процессора для воспроизведения вызывающему абоненту).

Вам, вероятно, потребуется отдельное расширение для записи каждого из приглашений, возможно, скрытое от обычного набора номеров, чтобы предотвратить затирание любой из ваших текущих подсказок меню с помощью неправильно набранного расширения. Если количество подсказок велико - повторение этого номера с небольшими изменениями для каждого будет утомительным, но есть способы обойти это. Мы покажем вам, как сделать вашу быструю запись более разумной в Главе 16, но сейчас только что описанный метод будет служить нашим непосредственным потребностям.

---

5 Если вы не являетесь экспертом в этих областях - в таком случае пойдите на это!

6 Подсказка *vm-intro* не идеальна (она просит вас оставить сообщение), но она достаточно близка для наших целей. Инструкции по использованию по крайней мере верны: нажмите #, чтобы завершить запись. После того, как вы научились записывать подсказки - можете вернуться назад, записать пользовательское приглашение и изменить приоритет 1, чтобы отразить более подходящие инструкции для записи собственных приглашений.

Вот диалплан (выделен жирным шрифтом), который создаст все наши подсказки. Поместите его туда, куда захотите в контексте [sets]:

```
exten => _4XX,1,Noop(User Dialed ${EXTEN})
  same => n,Answer()
  same => n,SayDigits(${EXTEN})
  same => n,Hangup()

exten => 500,1,GoSub(subRecordPrompt,${EXTEN},1(daygreeting)
exten => 501,1,GoSub(subRecordPrompt,${EXTEN},1(nightgreeting)
exten => 502,1,GoSub(subRecordPrompt,${EXTEN},1(mainmenu)
exten => 503,1,GoSub(subRecordPrompt,${EXTEN},1(holdwhileweconnect)
exten => 504,1,GoSub(subRecordPrompt,${EXTEN},1(faxandaddress)
exten => 505,1,GoSub(subRecordPrompt,${EXTEN},1(transfertoreception)
exten => 506,1,GoSub(subRecordPrompt,${EXTEN},1(invalid)
exten => 507,1,GoSub(subRecordPrompt,${EXTEN},1(holdwhileweconnect)

exten => _555XXXX,1,Answer()
  same => n,SayDigits(${EXTEN})
exten => _55512XX,1,Answer()
  same => n,Playback(tt-monkeys)
```

Записи (они же подсказки) будут помещены в директорию */var/lib/asterisk/sounds*. Вы можете поместить их в другое место, если вы укажете полный путь при записи и воспроизведении (и убедитесь, что каталог, в который вы их поместили, доступен пользователю asterisk). В продакшене их следует размещать в другом месте, чтобы отделить пользовательские приглашения от общих. На данный момент мы не будем усложнять и поместим их в ту же директорию, что и системные подсказки.

Диалплан

Вот код, необходимый для создания АС, который мы разработали ранее. Мы часто используем пустые строки перед метками внутри расширения, чтобы облегчить чтение диалплана, но обратите внимание, что только потому, что есть пустая строка, не означает, что есть другое расширение.

Вы можете поместить этот код в конце контекста [TestMenu], прямо перед вашими подпрограммами:

```
[MainMenu]

exten => s,1,Verbose(1, Caller ${CALLERID(all)} has entered the auto attendant)
    same => n,Answer()

; установка межразрядного таймера
    same => n,Set(TIMEOUT(digit)=2)

; ожидание в одну секунду для установки звука
    same => n,Wait(1)

; Если с Пн-Пт с 9-17 переход на метку daygreeting
    same => n,GotoIfTime(9:00-17:00,mon-fri,*,*?daygreeting:afterhoursgreeting)

    same => n(afterhoursgreeting),Background(nightgreeting) ; ПРИВЕТСТВИЕ НЕРАБОЧЕГО ВРЕМЕНИ
    same => n,Goto(menuprompt)

    same => n(daygreeting),Background(daygreeting)   ; ДНЕВНОЕ ПРИВЕТСТВИЕ
    same => n,Goto(menuprompt)

    same => n(menuprompt),Background(mainmenu)  ; ПОДСКАЗКА ГЛАВНОГО МЕНЮ
    same => n,WaitExten(4)                      ; более 4 секунд скорее всего
                                                ; слишком много
    same => n,Goto(0,1)                         ; Считайте, что вызывающий нажал '0'

exten => 1,1,Verbose(1, Caller ${CALLERID(all)} has entered the sales queue)
    same => n,Goto(sets,610,1)     ; Очередь sales - смотри Главу 12 для подробностей

exten => 2,1,Verbose(1, Caller ${CALLERID(all)} has entered the service queue)
    same => n,Goto(sets,611,1)     ; Очередь support - смотри Главу 12 для подробностей
```

```
exten => 3,1,Verbose(1, Caller ${CALLERID(all)} has requested address and fax info)
    same => n,Background(faxandaddress)            ; Информация об адресе и факсе
    same => n,Goto(s,menuprompt)      ; Вернуть абонента к подсказке главного меню

exten => #,1,Verbose(1, Caller ${CALLERID(all)} is entering the directory)
    same => n,Directory(default)   ; Отправить абонента в каталог.
                                   ; Использовать InternalSets в качестве контекста набора

exten => 0,1,Verbose(1, Caller ${CALLERID(all)} is calling the operator)
    same => n,Goto(sets,611,1)     ; Очередь support - смотри Главу 12 для подробностей

exten => i,1,Verbose(1, Caller ${CALLERID(all)} has entered an invalid selection)
    same => n,Playback(invalid)
    same => n,Goto(s,menuprompt)

exten => t,1,Verbose(1, Caller ${CALLERID(all)} has timed out)
    same => n,Goto(0,1)

; Вы должны иметь шаблон соответствия для внутренних номеров,
; которые вы позволите набирать внешним абонентам
; НО НЕ ПРОСТО ВКЛЮЧИТЬ КОНТЕКСТ LocalSets
; ИНАЧЕ АБОНЕНТЫ СМОГУТ СОВЕРШАТЬ ЗВОНКИ ИЗ ВАШЕЙ СИСТЕМЫ.

; ЧТО БЫ ВЫ НИ ДЕЛАЛИ ЗДЕСЬ - ТЩАТЕЛЬНО ПРОВЕРЬТЕ, ЧТОБЫ УБЕДИТЬСЯ, ЧТО ВНЕШНИЕ АБОНЕНТЫ
; НЕ СМОГУТ СДЕЛАТЬ НИЧЕГО, КРОМЕ НАБОРА ВНУТРЕННИХ НОМЕРОВ

exten => _1XX,1,Verbose(1,Call to an extension starting with '1')
    same => n,Goto(sets,${EXTEN},1)
```

Доставка входящих звонков в АС

Любой вызов, поступающий в систему, будет входить в диалплан в контексте, определенном для канала, на который поступает вызов. Во многих случаях это будет контекст с именем `[incoming]` или `[from-pstn]`, или что-то подобное. Вызовы будут поступать либо с добавочным номером (как в случае с DID), либо без него (как в случае с традиционной аналоговой линией).

Как бы ни назывался контекст и как бы ни назывался добавочный номер - вы будете отправлять каждый входящий вызов в меню.

```
[incoming] ; DID, приходящий в канал с
           ; context=incoming
exten => 4169671111,1,Goto(MainMenu,s,1)
```

В зависимости от того, как вы настраиваете входящие каналы — как правило будете использовать приложение `Goto()` для отправки вызова АС. Это гораздо аккуратнее, чем просто кодирование всего АС во входящем контексте.

Поскольку у нас в лаборатории[7] нет больше входящих линий - мы создадим простое расширение, которое доставит нас к нашему новому модному АС:

```
exten => 613,1,Noop()
  same => n,Goto(MainMenu,s,1)
  same => n,Hangup()
```

И это все! Простой автосекретарь, которым легко управлять, и который справится с ожиданиями большинства абонентов.

IVR

Мы рассмотрим интерактивное голосовое меню (IVR) более подробно в Главе 16, но прежде чем мы это сделаем поговорим о том, что важно для любого IVR. Интеграция баз данных является предметом следующей главы.

7 А если Вы это делаете, то и в Вашей; поздравляю и, пожалуйста, будьте осторожны, плавая с акулами.

Вывод

Автосекретарь может предоставить очень полезную услугу для абонентов. Однако, если он не разработан и плохо реализован, то также может стать барьером для ваших абонентов, который может отпугнуть их. Потратьте время, чтобы тщательно спланировать свой АС и держать его простым.

Глава 15. Интеграция реляционной базы данных

Нет ничего более раздражающего, чем хороший пример
– Марк Твен

В этой главе мы рассмотрим интеграцию некоторых особенностей и функций Asterisk в базу данных. Существует несколько баз данных, доступных для Linux, и Asterisk поддерживает наиболее популярные из них через свой ODBC-коннектор. Хотя в этой главе будут продемонстрированы примеры использования ODBC-коннектора с базой данных MySQL - вы также обнаружите, что большинство концепций будет применяться к любой базе данных, поддерживаемой unixODBC.

Интеграция Asterisk с базами данных является одним из фундаментальных аспектов построения большой кластерной или распределенной системы. Мощь базы данных позволит вам использовать динамически изменяющиеся данные в ваших диалпланах для таких задач, как обмен информацией в массиве систем Asterisk или интеграция с веб-службами. Наша любимая функция диалплана, которую мы рассмотрим позже в этой главе - это `func_odbc`. Мы также рассмотрим архитектуру Asterisk Realtime Architecture (ARA), записи деталей вызовов (CDR) и сведения о регистрации из любых очередей ACD, которые у вас могут быть.

Хотя не все развертывания Asterisk требуют реляционных баз данных, понимание того, как их использовать, открывает сокровищницу, полную новых способов разработки вашего телекоммуникационного решения.

Ваш выбор базы данных

В Главе 3 мы установили и настроили MySQL плюс ODBC-коннектор к нему и использовали таблицы, предоставляемые Asterisk, чтобы позволить различным параметрам конфигурации храниться в базе данных.

Мы выбрали MySQL в первую очередь потому, что это самый популярный движок баз данных с открытым исходным кодом, и вместо того, чтобы прыгать вокруг, дублируя тривиальные команды на разных движках, мы оставили реализацию других типов баз данных для набора навыков читателю. Если вы хотите использовать другую базу данных - такую как MariaDB, PostGreSQL, Microsoft SQL и фактически десяток (возможно, сотни) других баз данных, поддерживаемых unixODBC, вполне вероятно, что Asterisk будет работать и с ней.

Asterisk также предлагает собственные коннекторы для нескольких баз данных; однако ODBC работает так хорошо, что мы никогда не находили очевидной причины делать что-то иначе. Мы собираемся как рекомендовать использовать ODBC, так и сосредоточиться исключительно на нем. Если вы предпочитаете что-то другое - эта глава все равно должна предоставить вам основные принципы, а также некоторые рабочие примеры, и оттуда вы, конечно, можете перейти к другим методологиям.

Обратите внимание, что независимо от выбранной базы данных, эта книга не может научить вас базам данных. Мы постарались, насколько это возможно, привести примеры, не требующие слишком большого опыта в администрировании баз данных (DBA), но простой факт заключается в том, что базовые навыки DBA являются необходимым условием для полного использования возможностей любой базы данных, в том числе любой, которую вы можете интегрировать с вашей системой Asterisk. В наши дни навыки работы с базами данных необходимы практически для всех дисциплин системного администрирования, поэтому мы сочли целесообразным предположить, по крайней мере, базовый уровень знакомства с концепциями баз данных.

Управление базами данных

Хотя в эту книгу не входит обучение управлению базами данных, по крайней мере стоит кратко отметить некоторые приложения, которые можно использовать для управления базами данных. Есть много вариантов, некоторые из которых являются локальными клиентскими приложениями, запущенными с Вашего компьютера и подключающимися к базе данных, а другие - веб-приложениями, которые могут обслуживаться с того же компьютера, на котором запущена сама база данных, что позволяет вам подключаться удаленно.

Некоторые из тех, которые мы использовали, включают:

- phpMyAdmin
- MySQL Workbench
- Navicat (коммерческая)

В наших примерах мы будем использовать командную строку MySQL не потому, что она превосходит, а просто потому, что она присутствует в любой системе с MySQL, так что вы уже получили ее и использовали в этой книге.

Для разработки более производительной базы данных командная строка, вероятно, окажется не такой мощной, как хорошо продуманный графический интерфейс. Возьмите хотя бы копию MySQL Workbench и дайте ему закрутиться.

Устранение неисправностей базы данных

При работе с подключениями к базе данных ODBC и Asterisk важно помнить, что подключение ODBC абстрагирует часть информации, передаваемой между Asterisk и базой данных. В тех случаях, когда все работает не так, как ожидалось, вам может потребоваться включить ведение лога для базы данных, чтобы увидеть, что Asterisk отправляет в базу данных (например, какие инструкции `SELECT`, `INSERT` или `UPDATE` запускаются из Asterisk), что видит база данных и почему она может отклонять инструкции.

Например, одной из наиболее распространенных проблем, обнаруживаемых при интеграции базы данных ODBC, является неверно определенная таблица или отсутствующий столбец, который ожидает Asterisk. В то время, как большие успехи были сделаны в виде адаптивных модулей, не все части Asterisk являются адаптивными. В случае хранения голосовой почты ODBC, возможно, вы пропустили столбец, например `flag`, являющийся новым, отсутствующим в версиях Asterisk до 11.[1] как уже отмечалось - чтобы понять, почему ваши данные не записываются в базу данных как положено, вы должны включить логирование на стороне базы данных, а затем определить, какой оператор выполняется и почему база данных отклоняет его.

SQL-инъекция

Безопасность всегда учитывается при создании сетевых приложений, и безопасность базы данных не является исключением.

В случае Asterisk вам нужно подумать о том, какие входные данные вы принимаете от пользователей (обычно то, что они могут отправить в диалплан) и работать над очисткой этого ввода, чтобы убедиться, что вы разрешаете только символы, которые действительны для вашего приложения. Например, типичный телефонный звонок допускает только цифры в качестве входных данных (и, возможно, символы * и #), поэтому нет никаких причин принимать любые другие символы. Имейте в виду, что протокол SIP позволяет больше, чем просто цифры в качестве части адреса - поэтому не думайте, что те, кто пытается скомпрометировать вашу систему, ограничатся только цифрами.

1 На самом деле это было проблемой, с которой столкнулся один из авторов во время работы над этой книгой, и он нашел столбец `flag`, посмотрев на запись лога во время тестирования.

Немного дополнительного времени, потраченного на очистку разрешенного ввода, повысит безопасность вашего приложения.

Мощь вашего диалплана с функцией func_odbc

Функциональный модуль диалплана `func_odbc` позволяет определять и использовать относительно простые функции в вашем диалплане, которые будут извлекать информацию из баз данных при обработке вызовов. Существует множество способов, которыми это может быть использовано, например, управление пользователями или разрешение совместного использования динамической информации в кластеризованном наборе машин Asterisk. Мы не будем утверждать, что это облегчит разработку и написание кода диалплана, но обещаем что это позволит вам добавить совершенно новый уровень мощности к вашим диалпланам, особенно если вам удобно работать с базами данных. Мы не знаем никого в сообществе Asterisk, кто бы не любил `func_odbc`.

Способ работы `func_odbc` заключается в том, что вы можете определять SQL-запросы, которым присваиваете имена функций. Файл *func_odbc.conf* - это место, где вы указываете отношения между создаваемыми функциями и исполняемыми инструкциями SQL. Именованные функции, созданные в диалплане, используются для извлечения и обновления значений в базе данных.

Чтобы у Вас было хорошее настроение для того, что последует далее - мы хотим чтобы вы представили себе Дагвуд сэндвич.[2]

Можете ли вы передать общий опыт такой вещи, показав кому-то фотографию помидора или размахивая ломтиком сыра? Едва. Именно с этой загадкой мы столкнулись, пытаясь привести полезные примеры того, почему `func_odbc` настолько мощен. Итак, мы решили собрать целый сэндвич для вас. Это довольно полный рот, но после нескольких укусов этого, арахисовое масло и желе никогда не будет тем же самым.

Отношения файлов конфигурации ODBC

Чтобы Asterisk мог использовать ODBC из диалплана - все файлы должны быть выстроены в линию. Рисунок 15-1 пытается передать это визуально. Вы, вероятно, найдете эту диаграмму более полезной, как только проработаете примеры в следующих разделах.

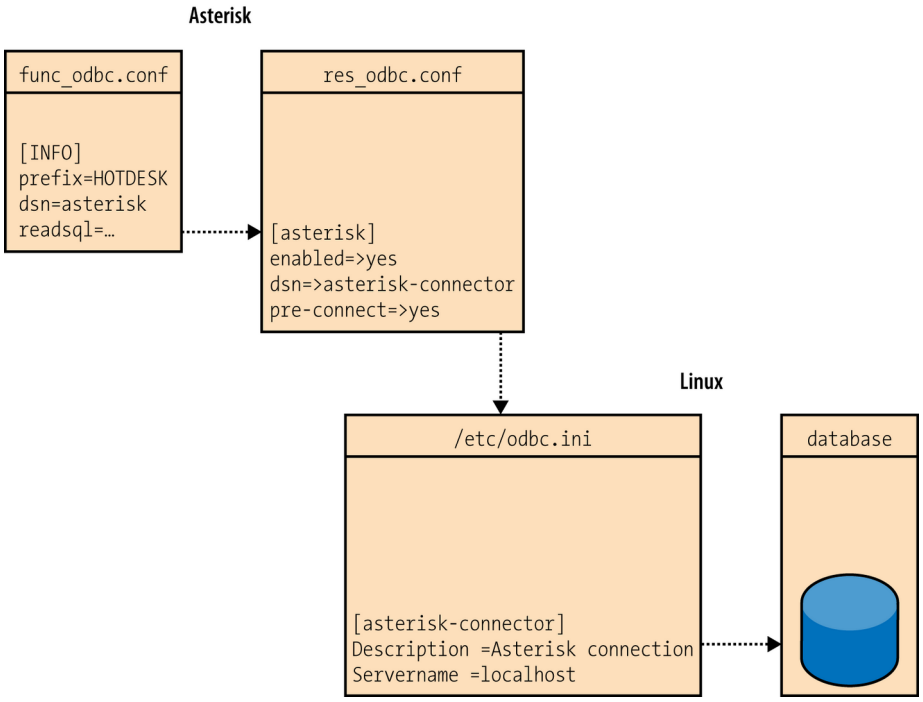

Рисунок 15-1. Отношения между func_odbc.conf, res_odbc.conf, /etc/odbc.ini (unixODBC) и подключением к базе данных

2 А если вы не знаете что такое Дагвуд - для этого существует Википедия. Я не настолько стар.

Мягкое введение в func_odbc

Прежде чем мы погрузимся в func_odbc, чувствуем что немного истории не помешает.

Самое первое использование func_odbc, произошедшее когда его автор все еще находился в процессе его написания, также является хорошим введением в его использование. Клиент одного из авторов модуля отметил, что некоторые люди, звонившие на его коммутатор, придумали способ совершать бесплатные звонки через его систему. В то время как его конечным намерением было изменить свой диалплан для избежания этих проблем - ему нужно было внести в черный список определенные CallerID абонентов, и база данных, которую он хотел использовать для этого, была базой данных Microsoft SQL Server.

За некоторыми исключениями, это фактический диалплан:

```
[span3pri]
exten => _50054XX,1,NoOp()
   same => n,Set(CDR(accountcode)=pricall)
   ; Имеется ли этот Caller ID в базе данных?
   same => n,GotoIf($[${ODBC_ANIBLOCK(${CALLERID(number)})}]?busy)
   same => n(dial),Dial(DAHDI/G1/${EXTEN})
   same => n(busy),Busy(10) ; Да, вы в черном списке.
   same => n,Hangup
```

Этот диалплан, в двух словах, передает все вызовы в другую систему для целей маршрутизации, за исключением тех вызовов, чьи CallerID находятся в черном списке. Звонки, поступающие в эту систему, использовали блок из 100 семизначных DID. Вы заметите, что используется функция диалплана, которую вы не найдете ни в одной из функций, поставляемых с Asterisk: ODBC_ANIBLOCK(). Вместо этого эта функция была определена в другом файле конфигурации *func_odbc.conf*:

```
[ANIBLOCK]
dsn=telesys
readsql=SELECT IF(COUNT(1)>0, 1, 0) FROM Aniblock WHERE NUMBER='${ARG1}'
```

Итак, ваша функция ODBC_ANIBLOCK()[3] подключается к источнику данных в *res_odbc.conf* называемому telesys и выбирает количество записей, которые имеют номер, указанный аргументом, который является (ссылаясь на предыдущий диалплан) идентификатором вызывающего абонента. Номинально эта функция должна возвращать либо 1 (указывая, что идентификатор вызывающего абонента существует в таблице Aniblock), либо 0 (если это не так). Это значение также вычисляется непосредственно как true или false, что означает, что нам не нужно использовать выражение в нашем диалплане для усложнения логики.

Вот в двух словах, что такое func_odbc: написание пользовательских функций диалплана, которые возвращают результат из базы данных. Далее, более подробный пример того, как можно использовать func_odbc.

Веселимся с func_odbc: горячий стол

Ладно, вернемся к Дагвуд-сэндвичу, который мы обещали.

Мы считаем, что значение func_odbc станет для вас более ясным если вы поработаете со следующим примером, который создаст новую функцию в вашей системе Asterisk, которая сильно зависит от func_odbc.

Представьте себе небольшую компанию с отделом продаж из пяти человек, которым приходится делить два стола. Это не так жестоко, как кажется, потому что эти люди большую часть своего времени проводят в дороге, и каждый из них находится в офисе не более одного дня в неделю.

3 Мы используем функцию SQL IF(), чтобы убедиться, что возвращаемое значение 0 или 1. Это работает на MySQL 5.1 или более поздней версии. Если оно не работает в вашей установке SQL - вы также можете проверить возвращаемый результат в диалплане, используя функцию IF() там.

Тем не менее, когда они попадают в офис, они хотели бы, чтобы система знала, за каким столом они сидят и их звонки могли быть направлены туда. Кроме того, босс хочет иметь возможность отслеживать когда они находятся в офисе и контролировать привилегии вызова с этих телефонов, когда там никого нет.

Эта потребность, как правило, решается с помощью так называемой функции *горячего стола*. Мы построили её для вас чтобы показать вам силу `func_odbc`.

Давайте начнем с простых вещей и создадим две новых учетных записи телефонов в нашей базе данных.

Во-первых, таблица конечных точек:

```
MySQL> INSERT INTO asterisk.ps_endpoints (id,transport,aors,auth,context,disallow,allow, \
direct_media,callerid)

VALUES
('HOTDESK_1','transport-tls','HOTDESK_1','HOTDESK_1','hotdesk','all','ulaw','no', \
'HOTDESK_1'),
('HOTDESK_2','transport-tls','HOTDESK_2','HOTDESK_2','hotdesk','all','ulaw','no', \
'HOTDESK_2');
```

И `auths`:

```
MySQL> INSERT INTO asterisk.ps_auths (id,auth_type,password,username)

VALUES
('HOTDESK_1','userpass','notsohot1','HOTDESK_1'),
('HOTDESK_2','userpass','notsohot2','HOTDESK_2');
```

Наконец `aors`:

```
MySQL> INSERT INTO asterisk.ps_aors
(id,max_contacts)

VALUES
('HOTDESK_1',1),
('HOTDESK_2',1);
```

Обратите внимание, что мы сказали этим двум конечным точкам войти в диалплан в контексте с именем `[hotdesk]`. Мы определим его в ближайшее время.

Это все для нашей конфигурации конечной точки. У нас есть несколько ломтиков хлеба, которые еще не стали бутербродом.

Теперь давайте создадим пользовательскую базу данных, которую будем использовать для этого.

Подключитесь к консоли MySQL как `root`:

```
$ mysql -u root -p
```

Сначала нам нужна новая схема чтобы разместить все это. Технически, это возможно поместить в схему `asterisk`, но мы предпочитаем оставить её в покое, зарезервированную только для всех скриптов Alembik Asterisk, которые выполняются с ней во время обновлений.

```
MySQL> CREATE SCHEMA pbx;

MySQL> GRANT SELECT,INSERT,UPDATE,DELETE,EXECUTE,SHOW VIEW ON pbx.* TO 'asterisk'@'::1';

MySQL> GRANT SELECT,INSERT,UPDATE,DELETE,EXECUTE,SHOW VIEW ON pbx.* TO \
'asterisk'@'127.0.0.1';

MySQL> GRANT SELECT,INSERT,UPDATE,DELETE,EXECUTE,SHOW VIEW ON pbx.* TO \
'asterisk'@'localhost';

MySQL> GRANT SELECT,INSERT,UPDATE,DELETE,EXECUTE,SHOW VIEW ON pbx.* TO \
'asterisk'@'localhost.localdomain';

MySQL> FLUSH PRIVILEGES;
```

Затем создайте таблицу со следующим битом SQL:

```
CREATE TABLE pbx.ast_hotdesk
(
  id serial NOT NULL,
  extension text,
  first_name text,
  last_name text,
  cid_name text,
  cid_number varchar(10),
  pin int,
  status bool DEFAULT false,
  endpoint text,
  CONSTRAINT ast_hotdesk_id_pk PRIMARY KEY (id)
);
```

После этого заполните базу данных следующей информацией (некоторые значения, которые вы видите на самом деле, изменятся только после завершения работы диалплана, но мы включим их здесь в качестве примера).

В консоли MySQL выполните следующую команду:

```
MySQL> INSERT INTO pbx.ast_hotdesk
(extension, first_name, last_name, cid_name, cid_number, pin, status)

VALUES
('1101','Herb','Tarlek','WKRP','1101','110111',0)
('1102','Al','Bundy','Garys','1102','110222',0),
('1103','Willy','Loman','','1103','110333',0),
('1104','Jerry','Lundegaard','Gustafson','1104','110444',0),
('1105','Moira','Brown','Craterside','1105','110555',0);
```

Повторите эти команды, изменяя `VALUES` по мере необходимости, для всех записей, которые вы хотите иметь в базе данных.[4] После ввода данных примера можно просмотреть данные в таблице `ast_hotdesk`, выполнив простую инструкцию `SELECT` из консоли базы данных:

```
MySQL> SELECT * FROM pbx.ast_hotdesk;
```

Что может дать вам что-то вроде следующего вывода:

```
+--+---------+----------+----------+----------+----------+------+------+--------+
|id|extension|first_name|last_name |cid_name  |cid_number|pin   |status|endpoint|
+--+---------+----------+----------+----------+----------+------+------+--------+
| 1|1101     |Herb      |Tarlek    |WKRP      |1101      |110111|     0|NULL    |
| 2|1102     |Al        |Bundy     |Garys     |1102      |110222|     0|NULL    |
| 3|1103     |Willy     |Loman     |          |1103      |110333|     0|NULL    |
| 4|1104     |Jerry     |Lundegaard|Gustafson |1104      |110444|     0|NULL    |
| 5|1105     |Moira     |Brown     |Craterside|1105      |110555|     0|NULL    |
+--+---------+----------+----------+----------+----------+------+------+--------+
```

Теперь у нас есть приправы, так что давайте перейдем к нашему диалплану. Именно здесь произойдет волшебство.

Где-то в *extensions.conf* мы собираемся создать контекст `[hotdesk]`. Для начала давайте определим расширение сопоставления с шаблоном, позволяющее пользователям входить в систему:

```
[hotdesk]
include => sets

exten => _*99110[1-5],1,Noop(Hotdesk login)
  same => n,Set(HotExten=${EXTEN:3}) ; отрезаем начальные *99
  same => n,Noop(Hotdesk Extension ${HotExten} is changing status) ; для лога
  same => n,Set(${HotExten}_STATUS=${HOTDESK_INFO(status,${HotExten})})
  same => n,Set(${HotExten}_PIN=${HOTDESK_INFO(pin,${HotExten})})
  same => n,Noop(${HotExten}_PIN is now ${${HotExten}_PIN})
  same => n,Noop(${HotExten}_STATUS is ${${HotExten}_STATUS})})
```

Мы еще не закончили написание этого расширения, но нам нужно отвлечься на несколько страниц, чтобы обсудить, где мы находимся на данном этапе.

4 Обратите внимание, что в первом примере пользователю присваивается статус `1` и местоположение, в то время как для второго примера мы не определяем значения для этих полей.

Когда торговый агент садится за стол, он входит в систему, набирая *99 плюс свой добавочный номер. В этом случае мы разрешили входить в систему расширениям с 1101 по 1105 посредством нашего шаблона соответствия `_99110[1-5]`. Вы могли бы так же легко сделать его менее ограниченным, используя `_9911XX` (разрешая с 1100 до 1199). Это расширение использует функцию `func_odbc` для выполнения поиска с помощью функции диалплана `HOTDESK_INFO()`. Эта пользовательская функция (которую мы определим в файле *func_odbc.conf*) выполняет инструкцию SQL и возвращает все, что извлекается из базы данных.

Итак, давайте создадим */etc/asterisk/func_odbc.conf*, и в нем определим новую функцию `HOTDESK_INFO()`:

```
$ sudo -u asterisk vim /etc/asterisk/func_odbc.conf

[INFO]
prefix=HOTDESK
dsn=asterisk
synopsis=Select value of field in ARG1, where 'extension' matches ARG2
description=Allow dialplan to extract data from any field in pbx.ast_hotdesk table.
readsql=SELECT ${ARG1} FROM pbx.ast_hotdesk WHERE extension = '${ARG2}'
```

Это очень много всего в нескольких строчках. Давайте быстро прикроем их, прежде чем двинемся дальше.

Вы должны быть в состоянии перезагрузить свой диалплан (`dialplan reload`) и `func_odbc` (`module reload func_odbc.so`) и протестировать диалплан до этого момента (наберите 991101 с одного из устройств, которые вы назначили этому контексту). Убедитесь, что ваша детальность консоли установлена по крайней мере на 3 (`*CLI> core set verbose 3`), так вы сможете увидеть в консоли что этот диалплан работает (вызов этого диалплана быстро вернет "занято", даже если он работает успешно). Для остальной части этого раздела мы настоятельно рекомендуем вам тестировать все после каждого изменения. Если вы этого не сделаете - у вас уйдёт много времени для поиска ошибок. Очень важно, чтобы вы кодировали с зарегистрированным телефоном и открытой консолью Asterisk, чтобы перезагрузить и протестировать изменения в течение нескольких секунд после их написания.

Прежде всего, `prefix` является необязательным (префикс по умолчанию - `ODBC`). Это означает, что если вы не определяете `prefix` - Asterisk добавляет `ODBC` к имени функции (в данном случае `INFO`), означающее, что эта функция станет `ODBC_INFO()`. Это не очень удачно описывает то, что делает функция - поэтому может быть полезно назначить префикс, который помогает связать ваши функции ODBC с задачами, которые они выполняют. Мы выбрали '`HOTDESK`' - означающее, что эта пользовательская функция в диалплане будет называться `HOTDESK_INFO()`.

Причина, по которой `prefix` является отдельным, заключается в том, что автор модуля хотел уменьшить возможные коллизии с существующими функциями диалплана. Цель `prefix` состояла в том, чтобы разрешить несколько копий одной и той же функции, подключенной к разным базам данных, для систем Asterisk с несколькими арендаторами. Мы, как авторы, были более либеральны в нашем использовании `prefix`, чем первоначально предполагал разработчик.

Атрибут `dsn` сообщает Asterisk, какое соединение использовать из *res_odbc.conf*. Поскольку в *res_odbc.conf* можно настроить несколько подключений к базам данных мы указываем какое из них использовать здесь. На Рисунке 15-1 показана взаимосвязь между различными конфигурациями файлов и тем, как они ссылаются на цепочку для подключения к базе данных.

Файл *func_odbc.conf.sample* в каталоге исходников Asterisk содержит дополнительную информацию о том, как обрабатывать несколько баз данных и управлять чтением и записью информации в различные соединения DSN. В

частности, аргументы `readhandle`, `writehandle`, `readsql` и `writesql` обеспечивают большую гибкость для интеграции и управления базами данных.

Наконец, мы определяем наш оператор SQL с атрибутом `readsql`. Функции диалплана могут вызываться в двух различных форматах: один для получения информации, а другой для записи. Атрибут `readsql` используется, когда мы вызываем функцию `HOTDESK_INFO()` с форматом поиска (мы могли бы выполнить отдельный оператор SQL с атрибутом `writesql`; мы обсудим формат для этого атрибута немного позже в этой главе).

Чтение значений из этой функции будет принимать этот формат в диалплане:

```
exten => s,n,Set(RETURNED_VALUE=${HOTDESK_INFO(status,1101)})
```

Это вернет значение, расположенное в базе данных в столбце `status`, где значение столбца `extension` будет равно `1101`. `status` и `1101`, которые мы передаем функции `HOTDESK_INFO()`, затем помещаются в инструкцию SQL, которую мы назначили атрибуту `readsql`, доступным как `${ARG1}` и `${ARG2}` соответственно. Если бы мы передавали третий параметр, то он был бы доступен как `${ARG3}`.

После выполнения инструкции SQL возвращаемое значение (если оно есть) присваивается переменной канала `RETURNED_VALUE`.

Использование функции ARRAY()

В нашем примере мы используем два отдельных вызова базы данных и присваиваем эти значения паре переменных канала: `${HotdeskExtension}_STATUS` и `${HotdeskExtension}_PIN`. Это было сделано для упрощения примера. Мы собираемся сократить имена переменных здесь, потому что печатный формат не может обрабатывать такие длинные строки, поэтому в следующих примерах вы увидите "HE" вместо "HotdeskExtension". Если вы собираетесь использовать этот пример, пожалуйста, замените `HE` на `HotdeskExtension`:

```
same => n,Set(${HE}_STATUS=${HOTDESK_INFO(status,${HE})})
same => n,Set(${HE}_PIN=${HOTDESK_INFO(pin,${HE})})
```

В качестве альтернативы мы могли бы вернуть несколько столбцов и сохранить их в отдельные переменные, используя функцию диалплана `ARRAY()`. Если бы мы определили наш оператор SQL в функции *func_odbc.conf* следующим образом:

```
readsql=SELECT pin,status FROM ast_hotdesk WHERE extension = '${HE}'
```

мы могли бы использовать функцию `ARRAY()` для сохранения каждого столбца информации для строки в отдельной переменной с помощью одного вызова базы данных (обратите внимание, что мы используем пример функции с именем `HOTDESK_INFO()`, которую мы не создали):

```
same => n,Set(ARRAY(${HE}_PIN,${HE}_STATUS)=${HOTDESK_INFO(${HE})})
```

Использование `ARRAY()` удобно в любое время, когда вы можете получить значения, разделенные запятыми, и хотите назначить значения отдельным переменным, например, с помощью `CURL()`. Однако это также может усложнить чтение, отладку и обслуживание кода.

Итак, в первых двух строках следующего блока кода мы передаем значение `status` и значение, содержащееся в переменной `${HotdeskExtension}` (например, `1101`) в функцию `HOTDESK_INFO()`. Затем эти два значения заменяются в инструкции SQL на `${ARG1}` и `${ARG2}` соответственно, и выполняется инструкция SQL. Наконец, возвращаемое значение присваивается переменной канала `${HotdeskExtension}_STATUS`.

Давайте закончим писать расширение для сопоставления прямо сейчас:

```
exten => _*99110[1-5],1,Noop(Hotdesk login)
  same => n,Set(HotdeskExtension=${EXTEN:3}) ; отрезаем начальные *99
```

```
same => n,Noop(Hotdesk Extension ${HotdeskExtension} is changing status) ; для лога
same => n,Set(${HotdeskExtension}_STATUS=${HOTDESK_INFO(status,${HotdeskExtension})})
same => n,Set(${HotdeskExtension}_PIN=${HOTDESK_INFO(pin,${HotdeskExtension})})
same => n,Noop(${HotdeskExtension}_PIN is now ${${HotdeskExtension}_PIN})
same => n,Noop(${HotdeskExtension}_STATUS is ${${HotdeskExtension}_STATUS})})
same => n,GotoIf($["${${HotdeskExtension}_PIN}" = ""]?invalid_user)
same => n,GotoIf($[${ODBCROWS} < 0]?invalid_user)
same => n,GotoIf($[${${HotdeskExtension}_STATUS} = 1]?logout:login,1)
```

Мы допишем некоторые метки для обработки `invalid_user` и `logout` немного позже - поэтому не волнуйтесь, если вам кажется что чего-то не хватает.

Возможно, вы заметили, что в некоторых примерах `Goto/GotoIf` в директиве может быть `1`. Это может показаться запутанным, если только вы не вспомните, что для цели нужна только разница между текущим *`context,extension,priority/label`*. Таким образом, если вы отправляете что-то на метку, например `logout`, которая находится в том же расширении, вам не нужно указывать контекст и расширение, тогда как если вы отправляете вызов на расширение с именем `login` (все еще в том же контексте), вам нужно указать что вы хотите перевести вызов на метку/приоритет `1`. В предыдущем примере мы могли бы записать нашу директиву следующим образом:

```
... = 1] ? hotdesk,${EXTEN},logout : hotdesk,login,1
           ^same    ^same   ^diff    ^same   ^diff ^diff
```

Другими словами, **`true`** переводит к контексту `[hotdesk]`, расширению `99110[1-5]`, метке `logout`; а **`false`** - к контексту `[hotdesk]`, расширению `login` и метке/приоритету `1`.

Мы написали только то, что отличается.

Если хотите, для ясности вы всегда можете указывать *`context,extension,priority`* для всех ваших директив. Это Ваш выбор.

После присвоения значения столбца `status` переменной `${HotdeskExtension}_STATUS` (если пользователь идентифицирует себя как расширение `1101`, имя переменной будет `1101_STATUS`), мы проверяем, получили ли значение обратно из базы данных, используя переменную канала `${ODBCROWS}`.

Последняя строка блока проверяет состояние телефона и, если агент в данный момент вошел в систему - выводит его оттуда. Если агент еще не вошел в систему - он перейдет к расширению `login`.

Расширение диалплана `login` выполняет некоторые начальные проверки для подтверждения PIN-кода, введенного агентом. (Кроме того, мы использовали функцию `FILTER()` чтобы убедиться, что были введены только числа для избежания некоторых проблем с SQL-инъекцией.) Мы разрешаем три попытки введения правильного PIN-кода и если все попытки недействительны - разрываем связь:

```
exten => login,1,NoOp()             ; установим начальные значения счетчика
   same => n,Set(PIN_TRIES=1)       ; счетчик попыток ввода pin
   same => n,Set(MAX_PIN_TRIES=3) ; установим максимальное количество попыток входа в
                                    ; систему
   same => n,Playback(silence/1)  ; воспроизведем некоторую тишину, чтобы первая подсказка
                                    ; не была обрезана
   same => n(get_pin),NoOp()
   same => n,Set(PIN_TRIES=$[${PIN_TRIES} + 1])   ; увеличить счетчик попыток pin-кода
   same => n,Read(PIN_ENTERED,enter-password,${LEN(${${HotdeskExtension}_PIN})})
   same => n,Set(PIN_ENTERED=${FILTER(0-9,${PIN_ENTERED})})
   same => n,GotoIf($["${PIN_ENTERED}" = "${${HotdeskExtension}_PIN}"]?valid:invalid)
   same => n,Hangup()

   same => n(invalid),Playback(vm-invalidpassword)
   same => n,GotoIf($[${PIN_TRIES} <= ${MAX_PIN_TRIES}]?get_pin)
   same => n,Playback(goodbye)
   same => n,Hangup()
```

```
    same => n(valid),Noop(Valid PIN)
```

Если введенный PIN-код совпадает - мы продолжаем процесс входа в систему через метку (`valid`). Сначала используем переменную `CHANNEL`, чтобы выяснить, с какого телефонного устройства звонит агент. Переменная `CHANNEL` обычно заполняется чем-то похожим на `PJSIP/HOTDESK_1-ab4034c`, поэтому мы используем функцию `CUT()` сперва для удаления части строки строки `PJSIP/`. Затем удаляем часть строки `-ab4034c`, и остается то, что мы хотим (`HOTDESK_1`):[5]

```
same => n(valid),Noop(Valid PIN)
; CUT технологию канала и назначаем переменной LOCATION
same => n,Set(LOCATION=${CUT(CHANNEL,/,2)})
; CUT уникальный идентификатор и сохраняем остаток в переменную LOCATION
same => n,Set(LOCATION=${CUT(LOCATION,-,1)})
; мы вернемся к этому в ближайшее время
```

Мы собираемся создать и использовать еще несколько функций в файле *func_odbc.conf:* `HOTDESK_CHECK_SET()`, которая определит, назначены ли уже другие пользователи этому телефону; `HOTDESK_STATUS()`, которая назначит телефон этому агенту; и `HOTDESK_CLEAR_SET()`, удаляющая всех других пользователей, назначенных в данный момент этому телефону (которые, возможно, забыли выйти из системы).

В файле *func_odbc.conf* нам нужно будет создать следующие функции:

```
; func_odbc.conf
[CHECK_SET]
prefix=HOTDESK
dsn=asterisk
synopsis=Check if this set is already assigned to somebody.
readsql=SELECT COUNT(status) FROM pbx.ast_hotdesk WHERE status = '1'
readsql+= AND endpoint = '${ARG1}'

[STATUS]
prefix=HOTDESK
dsn=asterisk
synopsis=Assign hotdesk extension to this endpoint/set.
writesql=UPDATE pbx.ast_hotdesk SET status = '${SQL_ESC(${VAL1})}',
writesql+= endpoint = '${SQL_ESC(${VAL2})}'
writesql+= WHERE extension = '${SQL_ESC(${ARG1})}'

[CLEAR_SET]
prefix=HOTDESK
dsn=asterisk
synopsis=Clear all instances of this endpoint
writesql=  UPDATE pbx.ast_hotdesk SET status=0,endpoint=NULL
writesql+= WHERE endpoint='${SQL_ESC(${VAL1})}'
```

Из-за ограничений длины строк в книге мы разбили команды `readsql` и `writesql` на несколько строк, используя синтаксис `+=`, который указывает Asterisk добавлять содержимое после `readsql+=` к самому последнему определенному значению `readsql=` (или `writesql` и `writesql+`). Использование `+=` применимо не только к опции `readsql`, но и может использоваться в других местах в других файлах *.conf* внутри Asterisk.

В нашем диалплане нам нужно будет вызвать функцию, которую мы только что создали, и передать поток вызовов метке `forcelogout`, если кто-то уже вошел в это устройство:

```
    same => n(valid),Noop(Valid PIN)
    same => n,Set(LOCATION=${CUT(CHANNEL,/,2)})
    same => n,Set(LOCATION=${CUT(LOCATION,-,1)})
  ; Мы вернемся к этому в ближайшее время ; вы можете удалить этот комментарий/строку
    same => n(checkset),Set(SET_USED=${HOTDESK_CHECK_SET(${LOCATION})})
    same => n,GotoIf($[${SET_USED} > 0]?forcelogout)

    ; Установить статус агента на '1' и обновить местоположение/конечную точку
```

5 Да, вы можете вложить функцию в функцию и поэтому сделайте это все в одной строке. Мы не сделали этого, так как это сложнее отлаживать и не влияет на производительность.

```
  same => n(set_login_status),Set(HOTDESK_STATUS(${HotdeskExtension})=1,${LOCATION})
  same => n,Noop(ODBCROWS is ${ODBCROWS})
  same => n,GotoIf($[${ODBCROWS} < 1]?error,1)
  same => n,Playback(agent-loginok)
  same => n,Hangup()

  same => n(forcelogout),NoOp()
; установить для всех текущих пользователей, вошедших в систему на этом устройстве, статус
; вышедших из системы
  same => n,Set(HOTDESK_CLEAR_SET()=${LOCATION})
  same => n,Goto(checkset)      ; return to logging in
```

Есть некоторые потенциально новые концепции, которые мы только что представили в примерах. В частности, синтаксис функции HOTDESK_STATUS() содержит несколько новых трюков, которые вы могли заметить. Теперь у нас есть переменные ${Val*x*} и ${ARG*x*} в нашем операторе SQL.

Мы также завернули значения ${Val*x*} и ${ARG*x*} в функцию SQL_ESC(), которая будет экранировать символы, такие как обратные кавычки, которые могут быть использованы в атаке SQL-инъекцией.

Они содержат информацию, которую мы передаем функции из диалплана. В этом случае у нас есть две переменные VAL и одна переменная ARG, которые были установлены из диалплана с помощью этого оператора:

```
same => n(set_login_status),Set(HOTDESK_STATUS(${HotdeskExtension})=1,${LOCATION})
```

Обратите внимание, что синтаксис немного отличается от синтаксиса функции чтения. Это сигнализирует Asterisk о том, что вы хотите выполнить запись (это тот же структурный синтаксис, что и для других функций диалплана).

Мы включаем значение переменной ${HotdeskExtension} в наш вызов функции HOTDESK_STATUS() (которая затем становится переменной ${ARG1} для этой функции в *func_odbc.conf*). Однако мы также передаем два значения, '1' и ${LOCATION}. Они будут связаны с переменными функции ${VAL1} и ${VAL2} соответственно.

Использование SQL непосредственно в диалплане

Некоторые предпочитают писать свои SQL-операторы непосредственно в диалплане, а не создавать пользовательскую функцию для каждого типа транзакции базы данных, которую они могут захотеть выполнить.

Теоретически, вы можете создать только одну функцию в *func_odbc.conf* как эта:

```
[SQL]
prefix=GENERIC
dsn=asterisk
readsql=${SQL_ESC(${ARG1})}
writesql=${SQL_ESC(${VALUE})} ; Целое значение, необработанное
```

Затем в диалплане вы можете написать практически любой тип SQL, который захотите (при условии, что ODBC-коннектор может обрабатывать его, что не имеет никакого отношения к Asterisk). Эта функция выше затем отправит любую строку, которую вы указали непосредственно в соединение ODBC с вашей базой данных.[6]

```
Некоторые утверждают, что это приводит к большей путанице в вашем диалплане; другие будут
настаивать на том, что преимущество наличия гораздо более простого func_odbc.conf стоит
того:
  same => n,Set(result=${GENERIC_SQL(SELECT col FROM table WHERE …)})
  same => n,Verbose(1,${result})
  same => n,Set(GENERIC_SQL()=UPDATE table SET field="VAL" WHERE …)
  same => n,Verbose(1,ODBC_RESULT is ${OBDBC_RESULT})
```

6 Это также может представлять ненужную угрозу безопасности.

Мы считаем, что в целом лучше создавать конкретные функции в *func_odbc.conf* для обработки запросов из вашего диалплана; тем не менее, нет никакого соблазна использовать одну функцию для обработки всех запросов SQL.

Многорядная функциональность с func_odbc

Asterisk имеет многорядный режим, позволяющий ей обрабатывать несколько строк данных, возвращаемых из базы данных. Например, если бы мы создали функцию диалплана в *func_odbc.conf*, возвращающую все доступные расширения - нам нужно было бы включить режим мультистрочности для функции. Это заставило бы функцию работать немного по-другому, возвращая идентификационный номер, который затем можно было бы передать функции `ODBC_FETCH()` для возврата каждой строки по очереди.

Далее следует простой пример. Предположим, что у нас есть следующий *func_odbc.conf*:

```
[AVAILABLE_EXTENS]
prefix=HOTDESK
dsn=asterisk
mode=multirow
readsql=SELECT extension FROM ast_hotdesk WHERE status = '${ARG1}'
```

и диалплан в *extensions.conf*, выглядящий примерно так:

```
exten => *9997,1,Noop(multirow)
 same => n,Set(ODBC_ID=${HOTDESK_AVAILABLE_EXTENS()})
 same => n,GotoIf($[${ODBCROWS} < 1]?no_rows)
 same => n,Answer()
 same => n,Set(COUNTER=1)
 same => n,While($[${COUNTER} <= ${ODBCROWS}])
   same => n,Set(AVAIL_EXTEN_${COUNTER}=${ODBC_FETCH(${ODBC_ID})})
   same => n,SayDigits(${AVAIL_EXTEN_${COUNTER}})
   same => n,Wait(0.2) ; Pause between speaking
   same => n,Set(COUNTER=$[${COUNTER} + 1])
 same => n,EndWhile()
 same => n(norows),ODBCFinish()
 same => n,Hangup()
```

Обратите внимание, что если у вас нет нескольких конечных точек для входа в систему - это никогда не вернет более одного расширения в вашей лаборатории, потому что только одно устройство будет входить в систему в любое время. Вы можете добавить некоторые фиктивные данные в таблицу чтобы просто посмотреть, как это работает:

```
MySQL> UPDATE pbx.ast_hotdesk
       SET status='1',endpoint='HOTDESK_2'
       WHERE id='3'
       ;
MySQL> UPDATE pbx.ast_hotdesk
       SET status='1',endpoint='HOTDESK_3'
       WHERE id='5'
       ;
```

Функция `ODBC_FETCH()` по существу будет обрабатывать информацию как стек, и каждый вызов к ней с переданным `ODBC_ID` будет выводить следующую строку информации из стека. У нас также есть возможность использовать переменную канала `ODBC_FETCH_STATUS`, которая устанавливается после вызова функции `ODBC_FETCH()` (возвращающая `SUCCESS` - если доступны дополнительные строки, или `FAILURE` - если нет дополнительных строк). Это позволяет нам написать диалплан, подобный приведенному ниже, использующий счетчик, но все же циклически перебирающий данные. Это может быть полезно, если мы ищем что-то конкретное и не нужно просматривать

все данные. Как только мы закончим, приложение диалплана ODBCFinish() должно быть вызвано для очистки всех оставшихся данных.

Вот еще один пример *extensions.conf*:

```
[multirow_example_2]
exten => start,1,Verbose(1,Looping example with break)
   same => n,Set(ODBC_ID=${GET_ALL_AVAIL_EXTENS(1)})
   same => n(loop_start),NoOp()
   same => n,Set(ROW_RESULT=${ODBC_FETCH(${ODBC_ID})})
   same => n,GotoIf($["${ODBC_FETCH_STATUS}" = "FAILURE"]?cleanup,1)
   same => n,GotoIf($["${ROW_RESULT}" = "1104"]?good_exten,1)
   same => n,Goto(loop_start)

exten => cleanup,1,Verbose(1,Cleaning up after all iterations)
   same => n,Verbose(1,We did not find the extension we wanted)
   same => n,ODBCFinish(${ODBC_ID})
   same => n,Hangup()

exten => good_exten,1,Verbose(1,Extension we want is available)
   same => n,ODBCFinish(${ODBC_ID})
   same => n,Verbose(1,Perform some action we wanted)
   same => n,Hangup()
```

Ладно, мы немного отклонились от темы. Давайте завершим несколько частей компонентов агента, которые еще не обработали.

В расширении _*99110[1-5] нам нужны следующие метки:

```
same => n,GotoIf($[${${HotdeskExtension}_STATUS} = 1]?logout:login,1)

same => n(invalid_user),Noop(Hot Desk extension ${HotdeskExtension} does not exist)
same => n,Playback(silence/2&login-fail)
same => n,Hangup()

same => n(logout),Noop()
same => n,Set(HOTDESK_STATUS(${HotdeskExtension})=0,) ; Note VAL2 is empty
same => n,GotoIf($[${ODBCROWS} < 1]?error,1)
same => n,Playback(silence/1&agent-loggedoff)
same => n,Hangup()
```

Мы также включаем контекст hotdesk_outbound, который будет обрабатывать исходящие вызовы после того, как мы зарегистрируем агента в системе:

```
include => hotdesk_outbound ; эта строка может быть в любом месте контекста [hotdesk]
```

Контекст [hot desk_outbound] использует многие из тех же принципов, которые уже обсуждались. Он использует совпадение шаблонов для перехвата любых номеров, набранных с телефонов горячего стола. Сначала мы устанавливаем нашу переменную LOCATION с помощью переменной CHANNEL, затем определяем какое расширение (агент) зарегистрировано в системе и присваиваем это значение переменной WHO. Если эта переменная имеет значение NULL - мы отклоняем исходящий вызов. Если оно не равно NULL, то мы получаем информацию об агенте с помощью функции HOTDECK_INFO() и присваиваем ее нескольким переменным CHANNEL.

```
include => hotdesk_outbound

; вставьте этот код прямо под контекстом [hotdesk]
[hotdesk_outbound]
exten => _NXXXXXX.,1,NoOp()
 same => n,Set(LOCATION=${CUT(CHANNEL,/,2)})
 same => n,Set(LOCATION=${CUT(LOCATION,-,1)})
 same => n(checkset),Set(VALID_AGENT=${HOTDESK_CHECK_SET(${LOCATION})})
 same => n,Noop(VALID_AGENT is ${VALID_AGENT})
 same => n,Set(${CALLERID(name)}=${HOTDESK_INFO(cid_name,${VALID_AGENT})})
 same => n,Set(${CALLERID(num)}=${HOTDESK_INFO(cid_number,${VALID_AGENT})})
 same => n,GotoIf($[${VALID_AGENT} = 0]?notallowed) ; Nobody logged in--calls not allowed
 same => n,Dial(${LOCAL}/${EXTEN}) ; Смотрите главу Внешние подключения
 same => n,Hangup()
```

```
same => n(notallowed),Playback(sorry-cant-let-you-do-that2)
same => n,Hangup()
```

Если вы не вошли в систему - вызов завершится ошибкой с сообщением. Если вы вошли в систему - вызов будет передан приложению `Dial()` (которое также может завершиться неудачей, если у вас нет настроенного оператора связи, но это описывается в предыдущих главах - поэтому мы оставим это в том разделе).

Нам требуется еще один последний бит диалплана. Мы создали эту сложную среду, которая позволяет нашим агентам входить и выходить, но на самом деле нет никакого способа вызвать их!

Мы собираемся исправить это сейчас, сделав четыре вещи:

1. Мы собираемся включить контекст `[sets]` в контекст `[hotdesk]`, чтобы наши агенты могли использовать другие части нашего диалплана.
2. Мы собираемся дать нашим агентам почтовые ящики.
3. Мы создадим новую подпрограмму, проверяющую службу поддержки на наличие агента и a) звонящая им - если они там есть или b) отправляющая вызов на голосовую почту - если их нет.
4. Мы собираемся построить диалплан в контексте `[sets]`, чтобы каждый мог позвонить нашим агентам.

Давайте сначала уберем почтовые ящики:

```
MySQL> insert into `asterisk`.`voicemail`
(mailbox,fullname,context,password)
VALUES
('1101','Herb Tarlek','default','110111'),
('1102','Al Bundy','default','110222'),
('1103','Willy Loman','default','110333'),
('1104','Jerry Lundegaard','default','110444'),
('1105','Moira Brown','default','110555');
```

Вся остальная работа заключается в *extensions.conf*:

Далеко внизу, в самом низу, давайте создадим подпрограмму, которая будет обрабатывать все для нас:

```
[subDialHotdeskUser]
exten => _[a-zA-Z0-9].,1,Noop(Call Hotdesk)
 same => n,Set(HOTDESK_ENDPOINT=${HOTDESK_INFO(endpoint,${EXTEN})}) ; Get assigned device
 same => n,GotoIf($["${HOTDESK_ENDPOINT}" = ""]?voicemail) ; if blank, send to voicemail
 same => n(ringhotdesk),Dial(PJSIP/${HOTDESK_ENDPOINT},${ARG1})
 same => n(voicemail),Voicemail(${EXTEN})
 same => n,Hangup()
```

И где-то гораздо ближе к началу мы добавим наших пользователей горячего стола в раздел диалплана, где живут наши другие пользователи:

```
exten => 110,1,Dial(${UserA_DeskPhone}&${UserA_SoftPhone}&${UserB_SoftPhone})

exten => 1101,1,GoSub(subDialHotdeskUser,${EXTEN},1(12))
exten => 1102,1,GoSub(subDialHotdeskUser,${EXTEN},1(12))
exten => 1103,1,GoSub(subDialHotdeskUser,${EXTEN},1(12))
exten => 1104,1,GoSub(subDialHotdeskUser,${EXTEN},1(12))
exten => 1105,1,GoSub(subDialHotdeskUser,${EXTEN},1(12))

exten => 200,1,Answer()
     same => n,Playback(hello-world)
     same => n,Hangup()
```

И наконец, вернувшись в наш контекст `[hotdesk]`, мы позволим нашим агентам использовать остальную часть телефонной системы:

```
[hotdesk]

include => sets
```

```
exten => _*99110[1-5],1,Noop(Hotdesk login)
```

Попробуйте несколько сценариев:

1. Вызов от внутреннего агента.
2. Вызов от обычного пользователя к зарегистрированному агенту.
3. Вызов от обычного пользователя к недоступному агенту.

Поразитесь этому технологическому террору, который вы создали.

Теперь, когда мы реализовали довольно сложную функцию в диалплане, используя `func_odbc` для извлечения и хранения данных в удаленной реляционной базе данных, вы можете увидеть как с помощью нескольких довольно простых функций в файле *func_odbc.conf* и нескольких таблиц в базе данных можно создать несколько мощных приложений телефонии.

Хорошо, давайте перейдем к архитектуре Asterisk Realtime, которая во многих случаях была устаревшей из-за ODBC, но все еще может быть полезной.

Использование Realtime

Архитектура Asterisk Realtime (ARA) позволяет хранить все параметры, обычно хранящиеся в конфигурационных файлах Asterisk (обычно располагающихся в */etc/asterisk*), в базе данных. Существует два типа realtime: *статический* и *динамический*.

Статическая версия аналогична традиционному способу чтения конфигурационного файла (информация загружается только при запуске из CLI), за исключением того, что вместо этого данные считываются из базы данных.[7]

Динамический realtime загружает и обновляет информацию по мере ее использования живой системой, обычно используется для таких вещей, как SIP (или IAX2 и т.д.) пользователи и пиры, а также ящики голосовой почты.

Внесение изменений в статическую информацию требует перезагрузки, как если бы вы изменили текстовый файл в системе, но динамическая информация опрашивается Asterisk по мере необходимости, поэтому при внесении изменений в эти данные перезагрузка не требуется. Realtime настраивается в файле *extconfig.conf* находящемся в каталоге */etc/asterisk*. Этот файл сообщает Asterisk что и откуда нужно грузить из базы данных, позволяя загружать определенные данные из базы данных, а другие - из стандартных файлов конфигурации.

Другим (возможно, более старым) способом хранения конфигурации Asterisk был внешний скрипт, который взаимодействовал бы с базой данных и генерировал соответствующие плоские файлы (или *.conf* файлы), а затем перезагружал соответствующий модуль после того, как новый файл был записан. В этом есть преимущество (если база данных выйдет из строя - ваша система будет продолжать функционировать; скрипт просто не будет обновлять файлы до тех пор, пока не будет восстановлено подключение к базе), но у него также есть недостатки. Одним из основных недостатков является то, что любые изменения, внесенные пользователем, будут недоступны до запуска скрипта обновления. Это, вероятно, не является большой проблемой для небольших систем, но на нагруженных системах ожидание применения изменений может вызвать проблемы, такие как приостановка вызова во время загрузки и анализа большого файла.

Вы можете частично избавиться от этого, используя реплицированную систему баз данных. Asterisk предоставляет возможность аварийного переключения на другую систему баз данных. Таким образом, вы можете кластеризировать бэкэнд базы

7 Да, вызов этого "realtime" несколько вводит в заблуждение, поскольку обновления данных не повлияют ни на что происходящее в реальном времени (пока не будет выполнена перезагрузка соответствующего модуля).

данных, используя отношения master-master (для PostgreSQL, pgcluster или Postgre-R;[8] для MySQL это встроено[9]) или master-slave (для PostgreSQL или Slony-I; для MySQL это встроено) системы репликации.

Наш неофициальный обзор таких вещей показывает, что использование скриптов для записи плоских файлов из баз данных не так популярно как запрос базы данных в реальном времени (и обеспечение базы данных с достаточной степенью отказоустойчивости для обработки того факта, что живая телекоммуникационная система зависит от нее).

Статический Realtime

Статический realtime был одним из самых ранних способов хранения конфигурации Asterisk в базе данных. Он все еще несколько полезен для хранения простых конфигурационных файлов в базе данных (которые обычно можно поместить в */etc/asterisk*). Мы больше не склонны использовать его, потому что динамический realtime намного лучше для больших наборов данных, а конфигурации на основе файлов более чем адекватны для небольших параметров конфигурации.

Те же правила, которые применяются к плоским файлам в вашей системе, по-прежнему применяются при использовании статического realtime. Например, после внесения изменений в конфигурацию вам все равно придется выполнить команду `module reload` для соответствующей технологии (например, `*CLI> module reload res_musiconhold.so`).

При использовании статического realtime мы сообщаем Asterisk, какие файлы хотим загрузить из базы данных, используя следующий синтаксис в файле *extconfig.conf*:

```
; /etc/asterisk/extconfig.conf
[settings]
filename.conf => driver,database[,table]
```

Нет никакого конфигурационного файла с именем *filename.conf*. Вместо этого используйте фактическое имя файла конфигурации, который вы храните в базе данных. Если имя таблицы не указано - Asterisk будет использовать имя файла в качестве имени таблицы вместо этого (за вычетом части *.conf*). Кроме того, все настройки внутри *extconfig.conf* должны находиться под заголовком `[settings]`. Имейте в виду - вы не можете загружать определенные файлы из realtime в принципе, как например *asterisk.conf*, *extconfig.conf* и *logger.conf*.

Модуль статического realtime использует очень специфично отформатированную таблицу, позволяющую Asterisk считывать различные статические файлы из базы данных. Таблица 15-1 иллюстрирует столбцы, как они должны быть определены в вашей базе данных.

Таблица 15-1. Макет таблицы и описание ast_config

Имя столбца	Тип столбца	Описание
`id`	Serial, автоувеличивающийся	Автоувеличивающееся уникальное значение для каждой строки таблицы.
`cat_metric`	Integer	Вес категории внутри файла. Более низкая метрика означает что она отображается выше в файле (см. врезку).
`var_metric`	Integer	Вес в пределах категории. Более низкая метрика означает, что она отображается выше в списке (см. врезку). Это полезно для таких вещей, как порядок кодеков в *sip.conf*, или *iax.conf*, в котором `disallow=all` должно быть первым (показатель `0`), затем `allow=ulaw` (метрика 1), а затем

8 *pgcluster* по-видимому, является мертвым проектом, а Postgres-R, скорей всего, находится в зачаточном состоянии, поэтому в настоящее время не может быть хорошего решения для репликации master-master с помощью PostgreSQL.

9 Есть несколько учебных пособий в интернете, описывающих, как настроить репликацию с MySQL.

Имя столбца	Тип столбца	Описание
		`allow=gsm` (метрика 2).
`filename`	Varchar 128	Имя файла, которое модуль обычно считывает с жесткого диска вашей системы (например *musiconhold.conf*, *sip.conf*, *iax.conf*).
`category`	Varchar 128	Имя раздела в файле, например `[general]`. Не заключайте имя в квадратные скобки при сохранении в базу данных.
`var_name`	Varchar 128	Параметр слева от знака равенства (например, `disallow` - это `var_name` в `disallow=all`).
`var_val`	Varchar 128	Параметр справа от знака равенства(например `all` - это `var_val` в `disallow=all`).
`commented`	Integer	Любое значение, отличное от 0, будет читаться так, как если бы оно было префиксировано точкой с запятой в файле конфигурации (закомментировано).

Несколько слов о метриках

Метрики в статическом realtime используются для управления порядком считывания объектов в память. Представьте себе `cat_metric` и `var_metric` как исходные номера строк в файле конфигурации. Сначала обрабатывается более высокое значение `cat_metric` - поскольку Asterisk сопоставляет категории снизу вверх. Однако в пределах одной категории сначала обрабатывается более низкая `var_metric` - потому что Asterisk обрабатывает параметры "сверху-вниз" (например для `disallow=all` должно быть установлено значение ниже, чем значение `allow` в категории, чтобы убедиться, что оно обрабатывается в первую очередь).

О статическом realtime сказать больше нечего. Он был очень полезен в прошлом, но теперь в основном вытеснен динамическим realtime. Если вы хотите прочитать о нём подробнее, то он рассматривается в более старых версиях этой книги.

Динамический Realtime

Система динамического realtime используется для загрузки объектов, которые могут часто изменяться, как например объекты PJSIP, очереди и их участники, а также голосовая почта. По мере того, как будут добавляться новые записи на регулярной основе, мы можем использовать возможности базы данных чтобы позволить нам загружать эту информацию по мере необходимости.

Вы уже много работали с динамическим realtime - поскольку именно так мы работали над всей этой книгой, как во время установки, так и в большинстве примеров, которые проработали.

Весь realtime настраивается в файле */etc/asterisk/extconfig.conf*; однако динамический режим realtime имеет четко определенные имена конфигурации. Все предопределенные имена должны быть настроены под заголовком `[settings]`. Например, определение одноранговых узлов (пиров) SIP выполняется с использованием следующего формата:

```
; extconfig.conf
[settings]
sippeers => driver,database[,table]
```

Имя таблицы является необязательным. Если оно опущено, Asterisk будет использовать предопределенное имя (т.е. `sippeers`) для определения таблицы, в которой будут искаться данные.

Пример файла *~/src/asterisk-16.<TAB>/configs/samples/extconfig.conf.sample* содержит отличную информацию о динамическом realtime.

Хранение записей деталей вызовов (CDR)

Записи деталей вызовов (CDR) содержат информацию о вызовах, прошедших через вашу систему Asterisk. Они рассматриваются далее в Главе 21. Хранение CDR - это популярное использование баз данных в Asterisk, потому что оно облегчает работу с ними. Кроме того, помещая записи в базу данных - вы открываете множество возможностей, включая создание собственного веб-интерфейса для отслеживания статистики, такой как использование вызовов и наиболее часто вызываемых назначений, биллинга счетов или проверка счетов телефонной компании.

Вы всегда должны реализовывать хранение CDR в базе данных на любой производственной системе (вы всегда можете хранить CDR в файле, так что ничего не потеряно).

Настройка системного имени для глобальных Unique ID

CDR состоит из уникального идентификатора и нескольких полей информации о вызове (включая исходный и целевой каналы, длину вызова, последнее выполненное приложение и т.д.). В кластеризованном наборе блоков Asterisk теоретически возможно дублирование уникальных идентификаторов, поскольку каждая система Asterisk учитывает только себя. Чтобы решить эту проблему мы можем автоматически добавлять системный идентификатор к передней части идентификаторов, добавив опцию в */etc/asterisk/asterisk.conf*. Для каждого из ваших блоков задайте идентификатор, добавив что-то вроде:

```
[options]
systemname=toronto
```

Лучший способ хранения записей деталей вызовов - это модуль `cdr_adaptive_odbc`. Он позволяет вам выбрать какие столбцы данных, встроенных в Asterisk, хранятся в вашей таблице и позволяет добавлять дополнительные столбцы, которые могут быть заполнены функцией диалплана `CDR()`. Вы даже можете хранить разные части данных CDR в разных таблицах и базах данных, если это необходимо.

Чтобы создать таблицу, у нас есть Alembic. Этот процесс практически идентичен тому, который вы выполняли во время установки системы, за исключением, конечно самого процесса, отличается так же и *.ini*-файл.

```
$ cd ~/src/asterisk-15.<TAB>/contrib/ast-db-manage

$ cp cdr.ini.sample cdr.ini

$ egrep ^sqlalchemy config.ini

sqlalchemy.url = mysql://asterisk:YouNeedAReallyGoodPasswordHereToo@localhost/asterisk
```

Учетные данные, которые мы использовали ранее, также будут работать для CDR.

```
$ sudo vim cdr.ini
```

Добавьте строку, которую вы только что получили от grep, в этот файл и сохраните.

```
$ alembic -c ./cdr.ini upgrade head

INFO  [alembic.runtime.setup] Creating new alembic_version_cdr table.
INFO  [alembic.runtime.migration] Running upgrade  -> 210693f3123d, Create CDR table.
INFO  [alembic.runtime.migration] Running upgrade 210693f3123d -> 54cde9847798
```

Alembic не слишком многословен - поэтому результат будет скупой, но, похоже, он успешно завершен. Давайте проверим.

```
$ mysql -u asterisk -p

MySQL> describe asterisk.cdr
```

Вы должны получить список всех полей в таблице (что означает, что Alembic выполнился успешно). Если вы получите сообщение типа `Table 'asterisk.cdr' doesn't exist` - это указывает что

Alembic не завершил настройку, и вам нужно просмотреть сообщения с вывода Alembic'a, чтобы увидеть, что пошло не так (обычно ошибка в учетных данных).

Ну, это было не так уж и трудно, правда? Следующий шаг - указать Asterisk, чтобы он использовал эту новую таблицу для CDR в будущем.

```
$ sudo -u asterisk touch /etc/asterisk/cdr_adaptive_odbc.conf

$ sudo -u asterisk vim /etc/asterisk/cdr_adaptive_odbc.conf
```

В этот новый файл вставьте следующее:

```
[adaptive_connection]
connection=asterisk
table=cdr
```

Это довольно просто, неправда ли? Отлично, теперь нам просто нужно перезагрузить модуль `cdr_adaptive_odbc.so` в Asterisk:

```
$ sudo asterisk -rvvvvvvv

*CLI> module reload cdr_adaptive_odbc.so
```

Вы можете проверить, что бэкэнд Adaptive ODBC был загружен, выполнив следующие действия:[10]

```
*CLI> cdr show status

Call Detail Record (CDR) settings
----------------------------------
  Logging:                    Enabled
  Mode:                       Simple
  Log unanswered calls:       No
  Log congestion:             No

* Registered Backends
  -------------------
    cdr-syslog
    Adaptive ODBC
    cdr-custom
    csv
    cdr_manager
```

Теперь сделайте вызов, на который будет получен ответ (например используя функции `Playback()` или `Dial()` в другом канале и ответив на него). Вы должны получить некоторые CDR, сохранившиеся в вашей базе данных. Вы можете проверить это, запустив команду `SELECT * FROM CDR;` из консоли базы данных.

При наличии основной информации CDR, хранящейся в базе данных, вам может потребоваться добавить в таблицу `cdr` дополнительную информацию, такую как стоимость маршрута. Вы можете использовать директиву `ALTER TABLE` для добавления столбца, называемого `route_rate,` к таблице:

```
sql> ALTER TABLE cdr ADD COLUMN route_rate varchar(10);
```

Теперь перезагрузите модуль `cdr_adaptive_odbc.so` из консоли Asterisk:

```
*CLI> module reload cdr_adaptive_odbc.so
```

и заполните новый столбец из диалплана Asterisk с помощью функции `CDR()`, например:

```
exten => _NXXNXXXXXX,1,Verbose(1,Example of adaptive ODBC usage)
   same => n,Set(CDR(route_rate)=0.01)
   same => n,Dial(SIP/my_itsp/${EXTEN})
   same => n,Hangup()
```

После внесения изменений в базу данных и диалплан вы можете сделать звонок, а затем посмотреть свои CDR. Вы должны увидеть что-то вроде следующего:

```
+--------------+----------+---------+------------+
| src          | duration | billsec | route_rate |
+--------------+----------+---------+------------+
| 0000FFFF0008 | 37       | 30      | 0.01       |
```

10 Вы можете увидеть различные зарегистрированные бэкенды, в зависимости от того, какую конфигурацию вы сделали с другими компонентами различных модулей CDR.

```
+--------------+----------+---------+------------+
```

На самом деле сохранение стоимости в записи вызова может быть неидеальным (CDR обычно используется в качестве исходного ресурса, а такие вещи, как тарифы, добавляются ниже по потоку с помощью программного обеспечения для биллинга). Возможность добавления настраиваемых полей в CDR очень полезна, но будьте осторожны, чтобы не использовать записи вызовов для замены надлежащей платформы биллинга. Лучше всего сохранить ваш CDR чистым и сделать дальнейшую обработку ниже.

Дополнительные параметры конфигурации для cdr_adaptive_odbc.conf

Некоторые дополнительные параметры конфигурации существуют в файле *cdr_adaptive_odbc.conf*, которые можгут быть полезны. Во-первых, вы можете определить несколько баз данных или таблиц для хранения информации - поэтому, если у вас есть несколько баз данных, которым нужна одна и та же информация, вы можете просто определить их в *res_odbc.conf*, создать таблицы в базах данных, а затем обращаться к ним в отдельных разделах конфигурации:

```
[mysql_connection]
connection=asterisk_mysql
table=cdr

[mssql_connection]
connection=production_mssql
table=call_records
```

Если вы зададите несколько разделов, используя одно и то же соединение и таблицу, то получите дублирующиеся записи.

Помимо простой настройки нескольких соединений и таблиц (которые, конечно, могут содержать или не содержать одну и ту же информацию; модуль CDR, который мы используем, адаптирован к подобным ситуациям), мы можем определить альясы для встроенных переменных, таких как `accountcode`, `src`, `dst` и `billsec`.

Если бы мы добавили альясы для имен столбцов для нашего соединения MS SQL - мы могли бы изменить наше определение соединения следующим образом:

```
[mssql_connection]
connection=production_mssql
table=call_records
alias src => Source
alias dst => Destination
alias accountcode => AccountCode
alias billsec => BillableTime
```

В некоторых ситуациях можно указать соединение, в котором требуется регистрировать вызовы только из определенного источника или в определенное место назначения. Мы можем сделать это с помощью фильтров:

```
[logging_for_device_0000FFFF0008]
connection=asterisk_mysql
table=cdr_for_0000FFFF0008
filter src => 0000FFFF0008
```

Если вам нужно заполнить определенный столбец информацией, основанной на имени раздела, вы можете установить его статически с помощью параметра `static`, который вы можете использовать с параметром filter:

```
[mysql_connection]
connection=asterisk_mysql
```

```
table=cdr

[filtered_mysql_connection]
connection=asterisk_mysql
table=cdr
filter src => 0000FFFF0008
static "DoNotCharge" => accountcode
```

В предыдущем примере вы получите повторяющиеся записи в той же таблице, но вся информация будет одинаковой, за исключением заполненного столбца accountcode, поэтому вы должны иметь возможность отфильтровать его с помощью SQL.

Интеграция базы данных с очередями

С call-центром (часто называемым очередями) может быть очень полезно иметь возможность разрешить настройку параметров очереди без необходимости редактирования и перезагрузки конфигурационных файлов. Управление call-центром может быть сложной задачей, а возможность более простой настройки параметров может сделать жизнь каждого человека намного проще.

Сами очереди мы уже разместили в базе данных в Главе 12. Однако если вы также хотите сохранить параметры диалплана, относящиеся к вашим очередям, база данных также может это сделать.

Хранение параметров диалплана для очереди в базе данных

Приложение диалплана `Queue()` позволяет передавать в него несколько параметров. Команда CLI `core show Application Queue` определяет следующий синтаксис:

```
[Syntax]
Queue(queuename[,options[,URL[,announceoverride[,timeout[,AGI[,macro[,gosub[,
  rule[,position]]]]]]]])
```

Поскольку мы храним нашу очередь в базе данных - почему бы также не сохранить параметры, которые вы хотите передать в очередь аналогичным образом?[11]

```
MySQL> CREATE TABLE `pbx`.`QueueDialplanParameters` (
  `QueueDialplanParametersID` mediumint(8) NOT NULL auto_increment,
  `Description` varchar(128) NOT NULL,
  `QueueID` mediumint(8) unsigned NOT NULL COMMENT 'Pointer to asterisk.queues table',
  `options` varchar(45) default 'n',
  `URL` varchar(256) default NULL,
  `announceoverride` bit(1) default NULL,
  `timeout` varchar(8) default NULL,
  `AGI` varchar(128) default NULL,
  `macro` varchar(128) default NULL,
  `gosub` varchar(128) default NULL,
  `rule` varchar(128) default NULL,
  `position` tinyint(4) default NULL,
  `queue_tableName` varchar(128) NOT NULL,
  PRIMARY KEY  (`QueueDialplanParametersID`)
);
```

Используя `func_odbc` вы можете написать функцию, которая будет возвращать параметры диалплана, относящиеся к этой очереди:

```
[QUEUE_DETAILS]
prefix=GET
dsn=asterisk
readsql=SELECT * FROM pbx.QueueDialplanParameters
readsql+= WHERE QueueDialplanParametersID='${ARG1}'
```

11 Обратите внимание, что мы создаем эту таблицу в нашей схеме `pbx`, а не в схеме `asterisk`, и это потому, что это не таблица, которая поставляется вместе с Asterisk, а та, которую мы создаем сами. Мы рекомендуем разрешить Asterisk и Alembic иметь исключительный контроль над схемой `asterisk` и использовать пользовательскую схему (например, `pbx`) для всего, что мы можем создать.

Затем передайте эти параметры приложению `Queue()` по мере поступления вызовов:

```
exten => s,1,Verbose(1,Call entering queue named ${SomeValidID)
  same => n,Set(QueueParameters=${GET_QUEUE_DETAILS(SomeValidID)})
  same => n,Queue(${QueueParameters})
```

Хотя это несколько сложнее в разработке чем просто написание соответствующего диалплана, преимущество состоит в том, что вы сможете управлять большим числом очередей с более широким набором параметров, используя диалплан, который достаточно гибок для обработки любых параметров, которые принимает приложение очередей в Asterisk. Для чего-то большего чем очень простая очередь, мы думаем, что вы найдете использование базы данных более удобным на которую и возможно возложить все эти усилия.

Запись queue_log в базу данных

Наконец, мы можем хранить наш журнал `queue_log` в базе данных, что может упростить внешними приложениями извлечение сведений о производительности очереди из системы:

```
CREATE TABLE queue_log (
  id int(10) UNSIGNED NOT NULL AUTO_INCREMENT,
  time char(26) default NULL,
  callid varchar(32) NOT NULL default '',
  queuename varchar(32) NOT NULL default '',
  agent varchar(32) NOT NULL default '',
  event varchar(32) NOT NULL default '',
  data1 varchar(100) NOT NULL default '',
  data2 varchar(100) NOT NULL default '',
  data3 varchar(100) NOT NULL default '',
  data4 varchar(100) NOT NULL default '',
  data5 varchar(100) NOT NULL default '',
  PRIMARY KEY (`id`)
);
```

Отредактируйте свой файл *extconfig.conf* для ссылки на таблицу `queue_log`:

```
[settings]
queue_log => odbc,asterisk,queue_log
```

Перезагрузите Asterisk, и ваша очередь теперь будет записывать информацию в базу данных. Например, вход агента в очередь `sales` должна производить что-то вроде этого:

```
mysql> select * from queue_log;
+----+----------------------------+----------------------+-----------+
| id | time                       | callid               | queuename |
+----+----------------------------+----------------------+-----------+
|  1 | 2013-01-22 15:07:49.772263 | NONE                 | NONE      |
|  2 | 2013-01-22 15:07:49.809028 | toronto-1358885269.1 | support   |
+----+----------------------------+----------------------+-----------+

+------------------+------------+-------+-------+-------+-------+-------+
| agent            | event      | data1 | data2 | data3 | data4 | data5 |
+------------------+------------+-------+-------+-------+-------+-------+
| NONE             | QUEUESTART |       |       |       |       |       |
| SIP/0000FFFF0001 | ADDMEMBER  |       |       |       |       |       |
+------------------+------------+-------+-------+-------+-------+-------+
```

Если вы разрабатываете какое-либо внешнее приложение, нуждающееся в доступе к статистике очередей - хранение данных таким образом будет намного лучше, чем использование файла */var/log/asterisk/queue_log*.

Вывод

В этой главе вы узнали о нескольких областях, где Asterisk может интегрироваться с реляционной базой данных. Это полезно для систем, где необходимо начать масштабирование путем кластеризации нескольких блоков Asterisk, работающих с одной и той же централизованной информацией, или когда вы хотите начать создавать внешние приложения для изменения информации, не требующие перезагрузки системы (т.е. без изменения плоских файлов).

Глава 16. Введение в интерактивное голосовое меню

Однажды Алиса подошла к развилке и увидела на дереве Чеширского кота. «По какой дороге мне пойти?» - спросила она.
«Куда ты хочешь пойти?» - был его ответ.
«Не знаю», - ответила Алиса.
«Тогда», - сказал кот, «это не имеет значения.»

– Льюис Кэролл

Термин *интерактивое голосовое меню* (на самом деле ответ) (IVR) часто неправильно используется для обозначения автосекретаря, но это совершенно разные вещи. Цель системы IVR состоит в том, чтобы принять входные данные от абонента, выполнить действие, основанное на этих входных данных (обычно, поиск данных во внешней системе, такой как база данных), и сообщить результат абоненту. Назначение автосекретаря (о котором мы говорили в Главе 14) - маршрутизация вызовов. Первоначально, IVR даже не должен был быть телефонной системой. Все, что принимало информацию от человека и выдавало на запрос результат - попадало в сферу IVR. Традиционно системы IVR были сложными, дорогими и раздражающими в реализации. Asterisk все это изменила.

Компоненты IVR

Самые основные элементы IVR очень похожи на элементы автосекретаря, хотя цель и отличается. Нам нужно по крайней мере одно приветствие чтобы сообщить вызывающему, что ожидает IVR, метод получения входных данных от вызывающего, логику для проверки что ответ вызывающего является допустимым вводом, логику определения следующего шага IVR, и, наконец, механизм хранения ответов, если это применимо. Мы могли бы думать об IVR как о дереве решений, хотя оно не должно иметь никаких ветвей. Например, опрос может представлять точно такой же набор подсказок для каждого вызывающего абонента, независимо от того, какой выбор делают абоненты и единственная логика маршрутизации, включенная в опрос, заключается в том, являются ли полученные ответы допустимыми для вопросов.

С точки зрения вызывающего абонента, каждый IVR должен начинаться с подсказки. Этот первоначальный запрос будет сообщать абоненту что он попал на IVR и попросит собеседника ввести первые данные. Мы обсуждали подсказки в автосекретаре в Главе 14. Позже мы создадим диалплан, который позволит вам лучше управлять несколькими голосовыми подсказками.

Второй компонент IVR - это метод получения входных данных от вызывающего абонента. Напомним, что в Главе 14 мы обсуждали `Background()` и `WaitExten()` как метод получения нового расширения. Хотя вы можете создать IVR с помощью `Background()` и `WaitExten()` обычно проще и практичнее использовать приложение `Read()`, которое обрабатывает как приглашение, так и захват ответа. Приложение `Read()` было разработано специально для использования с системами IVR. Его синтаксис выглядит следующим образом:

```
Read(variable[,filename[&filename2...]][,maxdigits][,option][,attempts][,timeout])
```

Аргументы описаны в Таблице 16-1.

Таблица 16-1. Приложение Read()

Аргумент	Цель
`variable`	Переменная, в которой хранится ответ абонента. Рекомендуется присвоить каждой переменной в IVR имя, аналогичное приглашению, связанному с этой переменной. Это поможет позже, если по деловым соображениям или простоте использования вам потребуется изменить порядок шагов IVR. Присвоение

имен переменным `var1`, `var2` и т.д. может показаться более простым в краткосрочной перспективе, но позже в вашем жизненном цикле это сделает исправление ошибок более трудным.

`prompt` — Файл (или список файлов, соединенных вместе с символом `&`) для воспроизведения вызывающему абоненту, запрашивающий ввод. Не забудьте опустить расширение в конце каждого имени файла.

`maxdigits` — Максимальное количество символов, которые можно использовать в качестве входных данных. В случае вопросов "Да/нет" и "множественный выбор" рекомендуется ограничить это значение `1`. В случае более длинных значений вызывающий абонент всегда может прервать ввод, нажав клавишу #.

`options`

`s` *(skip)*

Немедленно выйти, если канал не отвечает.

`i` *(indication)*

Вместо того чтобы воспроизводить подсказку, воспроизвести какой-либо сигнал индикации (например, сигнал набора номера).

`n` *(no answer)*

Считывание цифр от абонента, даже если на линию еще не ответили.

`attempts`

Количество попыток для воспроизведения подсказки. Если вызывающий не вводит ничего - приложение `Read()` может автоматически запросить пользователя. По умолчанию используется одна попытка.

`timeout`

Количество секунд, в течение которых вызывающий должен совершить свой ввод. Значение по умолчанию в Asterisk равно 10 секундам, хотя его можно изменить для одного приглашения с помощью этой опции или для всего сеанса, назначив значение с помощью функции диалплана `TIMEOUT(response)`.

Как только входные данные получены - они должны быть проверены. Если вы не проверите входные данные, то с большей вероятностью обнаружите, что ваши абоненты жалуются на нестабильное приложение. Недостаточно обрабатывать ожидаемые входные данные; вам также нужно обрабатывать входные данные, которых вы не ожидаете. Например, абоненты могут быть разочарованы и набрать 0 в вашем IVR; если вы сделали хорошую работу - вы это предусмотрите и соедините их с кем-то, кто может помочь им или предоставить полезную альтернативу. Хорошо спроектированный IVR (как и любая программа) будет пытаться предвидеть все возможные входные данные и предоставлять механизмы для их изящной обработки.

После проверки входных данных вы можете отправить их на внешний ресурс для обработки. Это может быть сделано с помощью запроса в базу данных, отправки в URI, программы AGI или многих других вещей. Это внешнее приложение должно выдать результат, который вы сможете передать обратно абоненту. Это может быть подробный результат такой как "Баланс вашего счета..." или простое подтверждение такое как "ваш счет был обновлен". Мы не можем придумать ни одного реального случая, когда не будет требоваться какой-то результат, возвращаемый звонящему.

Иногда IVR может иметь несколько шагов, и поэтому результат может включать запрос дополнительной информации от вызывающего абонента для перехода к следующему шагу приложения IVR.

Можно проектировать очень сложные системы IVR с десятками или даже сотнями возможных путей. Мы уже говорили об этом раньше и повторим еще раз: люди не любят разговаривать с вашей телефонной системой независимо от того насколько она умна. Держите ваше IVR простым для ваших абонентов и они гораздо более вероятно получат выгоду от него.

Безупречно вкусное IVR

Отличный пример IVR, который любят использовать люди - это тот, который используют многие компании по доставке пиццы: когда вы звоните, чтобы сделать заказ, IVR смотрит на Ваш CallerID и говорит: "Если вы хотите точно такой же заказ, как в прошлый раз, нажмите 1."

Это все, что оно делает, и это прекрасно.

Очевидно, что эти компании могли бы разработать массивно сложные IVR, которые позволили бы Вам выбрать каждую деталь вашего пирога ("для семизерновой корочки, нажмите 7"), но сколько нетрезвых, голодных клиентов могли бы успешно перемещаться по чему-то подобному в 3 часа ночи?

Лучшее IVR - это то, которое требует наименьшего количества входных данных от вызывающего абонента. Жми эту кнопку 1 и Ваша'ца уже в пути! Ура!

Конструктивные соображения IVR

При разработке вашего собственного IVR, есть некоторые важные вещи, которые следует иметь в виду. Мы составили этот список вещей, которые нужно и не нужно делать в вашем IVR.

Делать

- Держать его простым.
- Должна быть возможность набрать 0 чтобы связаться с живым человеком.
- Корректно обрабатывать ошибки.

Не делать

- Подумать что IVR может полностью заменить людей.
- Использовать ваше IVR чтобы показать людям насколько вы умны.
- Попробовать воспроизвести ваш сайт с помощью IVR.
- Постараться построить IVR когда не можете принять числовой или устный ввод. Никто не хочет писать свое имя на клавиатуре телефона.[1]
- Заставлять своих абонентов слушать рекламу. Помните, что они могут повесить трубку в любой момент, когда пожелают.

Модули Asterisk для создания IVR

"Фронтенд" IVR (части, которые взаимодействуют с абонентами) может обрабатываться в диалплане. Можно построить систему IVR, используя только диалплан (возможно, с использованием *astdb* для хранения и извлечения данных); однако, как правило, вам нужно будет взаимодействовать с чем-то внешним по отношению к Asterisk (“бэкенд” IVR).

CURL()

Функция диалплана `CURL()` в Asterisk позволяет охватить все веб-приложения одной строкой кода диалплана. Мы будем использовать её в нашем примере IVR в этой главе позже.

Возможно вы найдете `CURL()` довольно простой в использовании, создание веб-приложения потребует опыта работы с веб-разработкой.

1 Особенно если это что-то вроде Ван Меггелена.

func_odbc

Используя `func_odbc` можно разрабатывать чрезвычайно сложные приложения в Asterisk, используя только код диалплана и поиск по базе данных. Если вы не являетесь сильным программистом, но очень хорошо разбираетесь в диалпланах Asterisk и базах данных, вы полюбите `func_odbc` так же, как и мы. Проверьте это в Главе 15.

AGI

Интерфейс Asterisk Gateway является настолько важной частью интеграции внешних приложений с Asterisk, что мы посвятили ему отдельную главу. Дополнительную информацию вы найдете в Главе 18.

AMI

Интерфейс Asterisk Manager - это интерфейс сокета, который можно использовать для получения информации о конфигурации и состоянии, запроса выполняемых действий и уведомления о событиях, происходящих с вызовами. Мы также написали целую главу об AMI. Дополнительную информацию вы найдете в Главе 17.

ARI

Интерфейс Asterisk REST основан на знаниях, полученных в течение многих лет о том, как интегрировать Asterisk с веб-приложениями текущего поколения. Это настолько важно, что да - еще раз, есть целая глава, посвященная ему. Если вы хотите построить сложное IVR с помощью Asterisk - более подробно рассмотрите ARI в Главе 19.

Простое IVR с использованием CURL()

Прежде чем приступить к написанию внешней программы для обработки чего-либо - мы всегда тщательно обдумываем, есть ли способ выполнить работу в диалплане. Один из мощных способов, которым Asterisk может взаимодействовать с внешними данными - это URL-адрес, что очень хорошо делает программа GNU/Linux cURL. В Asterisk функция `CURL()` является функцией диалплана.

Мы собираемся использовать `CURL()` в качестве примера того, как может выглядеть чрезвычайно простое IVR. Мы запросим наш внешний IP-адрес у https://ipinfo.io/ip.[2]

На самом деле, большинство приложений IVR будут намного сложнее. Даже большинство применений `CURL()` будет сложным, так как URI может возвращать массивный и сильно изменяющийся объем данных, подавляющее большинство из которых будет непонятно Asterisk. Дело в том, что IVR - это не только диалплан; это также очень много о внешних приложениях, которые запускаются диалпланом, выполняющих реальную работу IVR.

Модуль `CURL()` был установлен во время нашего процесса установки несколько глав назад.

Диалплан

Диалплан для нашего примера IVR очень прост. Функция `CURL()` извлекает ваш IP-адрес из https://ipinfo.io/ip, а затем `SayAlpha()` озвучит результат вызывающему абоненту:

2 Эти бесплатные сайты поиска IP-адресов, похоже, все время покупаются и превращаются в рекламные шлюзы, поэтому то, что работало при написании этой книги, может больше не работать. Вам нужен сайт который вернет ваш IP-адрес и ничего больше. Сегодня, например https://ipinfo.io/ip. К тому времени, когда вы прочтете это, может быть что-то другое.

```
exten => *764,1,Verbose(2, Run CURL to get IP address from whatismyip.org)
    same => n,Answer()
    same => n,Set(MyIPAddressIs=${CURL(https://ipinfo.io/ip)})
    same => n,SayAlpha(${MyIPAddressIs})
    same => n,Hangup()
```

Простота этого до невозможности крута. В традиционной системе IVR на программирование такого рода может уйти несколько дней, если предположить, что это вообще возможно.

Функция записи подсказок IVR

В Главе 14 мы создали простой диалплан для записи подсказок. Он был довольно ограничен в том, что записывал только одно имя файла, и поэтому для каждого запроса требовалось отдельное расширение. Здесь мы расширим его чтобы создать полноценное меню для записи подсказок. Поскольку это сложная часть диалплана, а не подпрограмма или локальный канал, мы создадим новый раздел диалплана для различных функций и поместим туда такие вещи:

```
;FEATURES
[prompts]
exten => s,1,Answer
exten => s,n,Set(step1count=0) ; Инициализация счетчиков

; Если мы не получаем ответа после 3-х раз - мы перестаем спрашивать
   same => n(beginning),GotoIf($[${step1count} > 2]?end)
   same => n,Read(which,prompt-instructions,3)
   same => n,Set(step1count=$[${step1count} + 1])

; Все подсказки должны быть длиной 3 цифры
   same => n,GotoIf($[${LEN(${which})} != 3]?beginning)
   same => n,Set(step1count=0) ; Запрос успешен; сброс счетчиков
   same => n,Set(step2count=0)

   same => n(step2),Set(step2count=$[${step2count} + 1])
   same => n,GotoIf($[${step2count} > 2]?beginning) ; Нет ответа после 3 попыток

; Если файл не существует, то не спрашивайть нужно ли его воспроизводить
   same => n,GotoIf($[${STAT(f,/var/lib/asterisk/sounds/${which}.wav)} = 0]?recordonly)
   same => n,Background(prompt-tolisten)

   same => n(recordonly),Background(prompt-torecord)
   same => n,WaitExten(10) ; Ожидаем 10 секунд ответа
   same => n,Goto(step2)

   same => n(end),Playback(goodbye)
   same => n,Hangup()

exten => 1,1,Set(step2count=0)
   same => n,Background(/var/lib/asterisk/sounds/${which})
   same => n,Goto(s,step2)

exten => 2,1,Set(step2count=0)
   same => n,Playback(prompt-waitforbeep)
   same => n,Record(${CHANNEL(uniqueid)}.wav)

   same => n(listen),Playback(${CHANNEL(uniqueid)})
   same => n,Set(step3count=0)
   same => n,Read(saveornot,prompt-1tolisten-2tosave-3todiscard,1,,2,3)
   same => n,GotoIf($["${saveornot}" = "1"]?listen)
   same => n,GotoIf($["${saveornot}" = "2"]?saveit)
   same => n,GotoIf($["${saveornot}" = "3"]?tossit)
   same => n,Goto(listen)

   same => n(tossit),System(rm -f /var/lib/asterisk/sounds/${CHANNEL(uniqueid)}.wav)
   same => n,Goto(s,beginning)

   same => n(saveit),Noop('Set' app used to shorten example)
```

```
    same => n,Set(PromptToSave=/var/lib/asterisk/sounds/${CHANNEL(uniqueid)}.wav
    same => n,Set(WhereToSave=/var/lib/asterisk/sounds/${which}.wav
    same => n,System(mv -f ${PromptToSave} ${WhereToSave})
    same => n,Playback(prompt-saved)
    same => n,Goto(s,beginning)
```

В этой системе имя запроса больше не является описательным - вместо этого оно является числом. Это означает - что вы можете записывать гораздо большее разнообразие подсказок, используя один и тот же механизм, но компромисс заключается в том, что ваши подсказки больше не будут иметь описательных имен.

Если вы хотите проверить его - вам нужно будет записать подсказки, которые использует эта функция IVR (это своего рода мета, но да, нашему создателю подсказок нужны подсказки).

Поместите это в свой диалплан:

```
exten => 510,1,GoSub(subRecordPrompt,${EXTEN},1(prompt-tolisten))     ; нажмите 1
exten => 511,1,GoSub(subRecordPrompt,${EXTEN},1(prompt-torecord))     ; нажмите 2
exten => 512,1,GoSub(subRecordPrompt,${EXTEN},1(prompt-instructions)) ;3-цифры (от 000 до
999)
exten => 513,1,GoSub(subRecordPrompt,${EXTEN},1(prompt-waitforbeep))  ; ждите сигнала
exten => 514,1,GoSub(subRecordPrompt,${EXTEN},1(prompt-1tolisten-2tosave-3todiscard))
exten => 515,1,GoSub(subRecordPrompt,${EXTEN},1(prompt-saved))
```

Затем позвоните им по одному и запишите по мере необходимости.

После того, как вы записали подсказки, необходимые вашему создателю подсказок, вы должны быть в состоянии проверить их.

```
exten => *742,1,Noop(Prompts)
    same => n,Goto(prompts,s,1)
    same => n,Hangup()
```

С этого момента вы можете записывать подсказки, используя только числовой идентификатор. Вам понадобится способ отслеживать что говорит подсказка, но с точки зрения записи - вам не нужно больше писать диалплан каждый раз, когда нужна подсказка.

Распознавание речи и преобразование текста-в-речь

Хотя традиционно и по-прежнему в большинстве случаев сегодня система IVR представляет предварительно записанные подсказки вызывающему абоненту и принимает ввод через панель набора номера, также возможно: а) искусственно генерировать подсказки, широко известные как преобразование текста-в-речь; и б) принимать устный ввод через механизм распознавания речи.

В то время как концепция возможности вести интеллектуальный разговор с машиной - это то, что авторы научной фантастики обещают нам в течение многих лет, реальная наука об этом остается сложной и подверженной ошибкам. Несмотря на свои удивительные возможности - компьютеры плохо приспособлены к задаче оценки тонких нюансов человеческой речи.

Тем не менее, следует отметить, что такие компании как Google, достигли удивительных успехов как в преобразовании текста-в-речь, так и в распознавании речи. Уже доступны API, которые могут проделать замечательную работу по осмыслению того, что им говорят. Google, конечно, выигрывает от наличия массивного бэкенда, который может выполнять почти чудесные трюки обработки; то, что ваш IVR не сможет полностью использовать.

Преобразование текста-в-речь

Преобразование текста-в-речь (также известное как синтез речи) требует чтобы система была способна искусственно создавать фразы из сохраненных данных. Хотя было бы неплохо, если бы мы могли просто назначить звук букве и заставить компьютер воспроизводить каждый звук, когда он читает буквы, письменный язык часто не фонетичен и редко отражает нюансы речи (английский, возможно, один из худших языков в этом отношении).

Существуют отличные API, доступные от Google (и других), которые сделают очень хорошую работу по чтению того, что было написано. На момент написания этой книги все еще очень очевидно, что речь идет о компьютере, но тем не менее можно генерировать системные подсказки на лету из текста, а не записывать их заранее. Полезность этого трудно оценить, так как люди все еще не заинтересованы в разговоре с вашими машинами - они позвонили потому, что хотят поговорить с вами.

Распознавание речи

Поскольку нам удалось убедить наши компьютеры говорить с нами - мы, естественно, хотим иметь возможность говорить и с ними.[3]

Распознавание речи раньше было сложным и дорогостоящим, но недавно Google выпустила API, который позволяет огромной мощности их возможностей распознавания речи быть доступной для внешних приложений.

Вывод

Asterisk является отличной платформой IVR. Вся эта книга, во многом, учит вас навыкам, которые могут быть применены для развития IVR. В то время, как основные СМИ действительно уделяют внимание Asterisk только как “свободной УАТС”, реальность такова, что Asterisk является наиболее мощной при использовании в качестве IVR. В любой солидной организации, очень вероятно, что системные администраторы Linux используют Asterisk для решения телекоммуникационных проблем, неразрешимых ранее, либо невероятно дорогими для решения. Это скрытая революция, но не менее значимая из-за своей относительной неизвестности.

Если вы занимаетесь IVR-бизнесом - вам обязательно нужно познакомиться с Asterisk.

3 Вообще-то, большинство из нас разговаривает с компьютерами, но это редко бывает вежливо.

Глава 17. AMI и файлы вызовов

Джон Малкович: я видел мир, который не должен видеть ни один человек!
Крейг Шварц: Правда? Потому что для большинства людей это довольно приятный опыт.

– Быть Джоном Малковичем

Интерфейс Asterisk Manager (Asterisk Manager Interface - AMI) - это интерфейс мониторинга и управления системой, предоставляемый Asterisk. Он позволяет в реальном времени отслеживать события, происходящие в системе, а также позволяет запрашивать к Asterisk выполнение некоторых действий. Доступные действия имеют широкий диапазон и включают такие вещи, как возврат информации о состоянии или инициирование новых вызовов. Для Asterisk было разработано много интересных приложений, использующих AMI в качестве основного интерфейса для Asterisk.

Эта глава также включает документацию по использованию файлов вызовов. Файлы вызовов Asterisk - это простой способ инициировать несколько вызовов. Как только объем исходящих вызовов увеличивается, или ваши потребности становятся более сложными, вы можете перейти к использованию AMI. На самом деле, мы находим файлы вызовов достаточно полезными, так что сначала поговорим о них.

Файлы вызовов

Обычно для инициализации вызовов используется AMI, но во многих ситуациях проще использовать файлы вызовов. Файл вызова - это простой текстовый файл, описывающий вызов, который вы хотите совершить через Asterisk. Когда файл вызова помещается в каталог */var/spool/asterisk/outgoing*, Asterisk немедленно обнаружит, что файл был помещен туда, и обработает вызов.

Asterisk поставляется с образцом файла вызова, который вы найдете в *~/src/asterisk-16.<TAB>/sample.call* (или там, где находится корневой каталог исходников Asterisk).

Ваш первый файл вызова

Для вашего первого файла вызова давайте создадим вызов между двумя вашими телефонами. Убедитесь, что хотя бы два ваших телефона зарегистрированы и работают. Для этого примера мы будем использовать `SOFTPHONE_A` и `SOFTPHONE_B`.

Создайте в домашнем каталоге следующий файл:

```
$ vim ~/call-file

Channel: PJSIP/SOFTPHONE_A
Extension: 103
Context: sets
```

Сделайте копию этого файла (так что вам не придется заново создавать его каждый раз, когда захотите запустить его):

```
$ cp ~/call-file docall
```

Измените владельца файла docall на `asterisk`:

```
$ chown asterisk:asterisk docall
```

Переместите файл *docall* в каталог *outgoing* Asterisk.

```
$ sudo mv docall /var/spool/asterisk/outgoing
```

Иногда самый простой способ - лучший.

Вы, вероятно, обнаружите, что делаете несколько правок в исходном файле вызова. Вы можете просто переместить созданный файл, а не делать его копию, но тогда вам придется заново создавать его каждый раз, когда вы его редактируете, и это раздражает. Весь этот набор можно сохранить как однострочный и запустить следующим образом:

```
$ cp ~/call-file docall \
 sudo chown asterisk:asterisk docall \
 sudo mv docall /var/spool/asterisk/outgoing/
```

Попробуйте, и вы увидите, насколько это проще, чем каждый раз создавать и перемещать новый файл вызова.

Использование `mv` вместо `cp` здесь важно. Asterisk следит за тем, чтобы содержимое отображалось в каталоге *spool*. Если вы используете копирование - Asterisk может попытаться прочитать новый файл до того, как содержимое будет скопировано в него. Создание файла, а затем его перемещение позволяет избежать этой проблемы.

Освойтесь с использованием файлов вызовов и вы обнаружите что они решают проблемы, которые в противном случае вам пришлось бы решать гораздо большим объемом работ.

Заметки о файлах вызова

Компонент `Channel` файла вызова является обязательным. Обычно вызов, поступающий в Asterisk, инициируется конечной точкой (например, вы делаете вызов со своего телефона). В файле вызова - это соединение должно происходить наоборот - Asterisk обращается к конечной точке, и только когда она отвечает, вызов может начаться. Планируйте соответственно.

Вы также должны указать `Context`, в котором начнется вызов как только первоначальный канал ответит. Это может быть полезно, так как это означает, что вы можете подключить вызов через контекст, который обычно недоступен для этого канала, но на практике мы бы предложили вам просто использовать тот же контекст, через который канал вошел бы в диалплан, если бы он инициировал вызов как обычно.

Расширение, конечно, также должно быть указано. Обычно - это номер телефона, по которому нужно позвонить, но, конечно, это может быть любой допустимый добавочный номер в `Context`.

Остальные параметры файла вызова являются необязательными и подробно описаны в файле *~/src/asterisk-16.<TAB>/sample.call* и на веб-сайте Asterisk wiki.

Быстрый запуск AMI

Этот раздел предназначен для того, чтобы как можно быстрее испачкать руки с помощью AMI. Во-первых, поместите следующую конфигурацию в */etc/asterisk/manager.conf*:

```
; Включить AMI и указать ему принимать соединения только от localhost.
[general]
enabled = yes
webenabled = yes
bindaddr = 127.0.0.1

; Создайть аккаунт с именем "hello" и паролем "world"
[hello]
secret=world
read=all     ; Получать все типы событий
write=all    ; Разрешить этому пользователю выполнять все действия
```

Этот пример конфигурации настроен так, чтобы разрешить только локальные подключения к AMI. Если вы собираетесь сделать этот интерфейс доступным по сети - настоятельно рекомендуется использовать только протокол TLS. Использование TLS более подробно рассматривается далее в этой главе.

Как только конфигурация AMI готова - включите встроенный HTTP-сервер, поместив следующее содержимое в */etc/asterisk/http.conf*:

```
; Включить встроенный HTTP-сервер и слушайть только соединения на localhost.
[general]
enabled = yes
bindaddr = 127.0.0.1
```

Перезагрузите диспетчер и http-серверы из Asterisk CLI:

```
*CLI> manager reload

*CLI> module reload http
```

AMI через TCP

Существует несколько способов подключения к AMI, но наиболее распространенным является TCP-сокет. Мы будем использовать `telnet` для демонстрации подключения AMI. Для этого нам нужно будет установить `telnet`:

```
$ sudo yum -y install telnet
```

В этом примере показаны следующие шаги:

- Подключение к AMI через TCP-сокет на порту 5038.
- Вход в систему, используя действие `Login`.
- Выполнение действия `Ping`.
- Выход из системы с помощью действия `Logoff`.

Вот как это сделать с помощью `telnet`:

```
$ telnet localhost 5038

Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Asterisk Call Manager/4.0.3
```

Вы подключились, но он завершит сеанс, если вы не подтвердите свою подлинность. Вставьте в окно `telnet` следующее:

```
Action: Login
Username: hello
Secret: world
```

Обратите внимание, что после команд должна быть пустая строка (нажмите Enter после вставки всего, если ничего не происходит).

```
Response: Success
Message: Authentication accepted
```

Ладно - мы ему нравимся. Давайте выполним простую команду, чтобы убедиться, что он действительно говорит с нами:

```
Action: Ping
Response: Success
Ping: Pong
```

Все идет нормально. Мы просто уберемся и выйдем сейчас.

```
Action: Logoff
Response: Goodbye
Message: Thanks for all the fish.
Connection closed by foreign host.
```

Вы убедились что AMI принимает соединения через TCP-соединение.

AMI через HTTP

Также можно использовать AMI через HTTP. Мы будем выполнять те же действия что и раньше, но через HTTP вместо собственного TCP-интерфейса к AMI. AMI через HTTP подробно описаны в "AMI через HTTP".

Учетные записи, используемые для подключения к AMI через HTTP, являются теми же учетными записями, настроенными в файле */etc/asterisk/manager.conf*.

В этом примере показано как получить доступ к AMI по протоколу HTTP, войти в систему, выполнить действие `Ping` и выйти из системы:

```
$ curl "http://localhost:8088/rawman?action=login&username=hello&secret=world" \
-c /tmp/tempcookie

Response: Success
Message: Authentication accepted
$ curl "http://localhost:8088/rawman?action=ping" -b /tmp/tempcookie

Response: Success
Ping: Pong
Timestamp: 1538871944.474131
$ curl "http://localhost:8088/rawman?action=logoff" -b /tmp/tempcookie

Response: Goodbye
Message: Thanks for all the fish.
```

Интерфейс HTTP для AMI позволяет интегрировать управление вызовами Asterisk в веб-службу.

Конфигурация

Раздел "AMI быстрый старт" показал очень простой набор конфигурационных файлов для начала работы. Существует много способов тонкой настройки конфигурации AMI.

manager.conf

Основной конфигурационный файл для AMI - это */etc/asterisk/manager.conf*. Раздел `[general]` содержит параметры, управляющие общей работой AMI. Любые другие разделы в *manager.conf* определяют учетные записи для входа в систему и использования AMI. Пример файла содержит подробные объяснения различных параметров и может быть найден в *~/src/asterisk-16<TAB>/configs/samples/manager.conf.sample*.

Если вы собираетесь выставить свой AMI за пределы машины, на которой он работает, вам потребуется настроить подключение TLS.

Конфигурационный файл *manager.conf* также содержит конфигурацию учетных записей пользователей AMI. Вы создаете учетную запись, добавляя раздел с именем пользователя в квадратных скобках. В каждом разделе `[username]` есть параметры, которые могут быть установлены применительно только к этой учетной записи. Файл *~/src/asterisk-16<TAB>/configs/samples/manager.conf.sample* также содержит подробные объяснения каждого из этих параметров. Наш пользователь по имени `[hello]`, имеет простейшую конфигурацию, которая позволяет все операции чтения и записи. Обычно следует создавать пользователей AMI, ограниченных только действиями, необходимыми для их функционирования.

В разделе `[username]` параметры `read` и `write` определяют к каким действиям и событиям диспетчера имеет доступ конкретный пользователь. На данный момент есть 20 из них: `all`, `system`, `call`, `log`, `verbose`, `agent`, `user`, `config`, `command`, `dtmf`, `reporting`, `cdr`, `dialplan`, `originate`, `agi`, `cc`, `aoc`, `test`, `security` и `message`. Вы увидите, что файл *manager.conf.sample* содержит ссылку на каждый из них, относящийся к вашему выпуску (и, если какие-либо из них добавлены, не перечисленные здесь - они будут в файле примера).

Обратите особое внимание на разрешения `system`, `command` и `originate`. Эти разрешения предоставляют значительные полномочия всем приложениям, которые имеют право их использовать. Предоставляйте эти разрешения только приложениям, над которыми у вас есть полный контроль (и в идеале они работают в одном окне).

http.conf

Как мы уже видели - интерфейс Asterisk Manager может быть доступен как по протоколу HTTP, так и по протоколу TCP. Для этого в Asterisk встроен очень простой HTTP-сервер. Все параметры, относящиеся к AMI, находятся в разделе `[general]` файла */etc/asterisk/http.conf*.

Включение доступа к AMI по протоколу HTTP требует наличия */etc/asterisk/manager.conf* и */etc/asterisk/http.conf*. AMI должен быть включен в *manager.conf* с параметром `enabled`, установленным в `yes` и `webenabled` должен быть установлен в значение `yes` чтобы разрешить доступ по протоколу HTTP. Наконец, опция `enabled` в *http.conf* должна быть установлена в `yes` чтобы включить сам HTTP-сервер.

Доступные опции будут найдены в вашем файле *~/src/asterisk-16<TAB>/configs/samples/http.conf.sample*.

Обзор протокола

В AMI есть два основных типа сообщений: события диспетчера и действия диспетчера.

События диспетчера - это односторонние сообщения, посылаемые Asterisk клиентам AMI для сообщения о том, что произошло в системе (Рисунок 17-1).

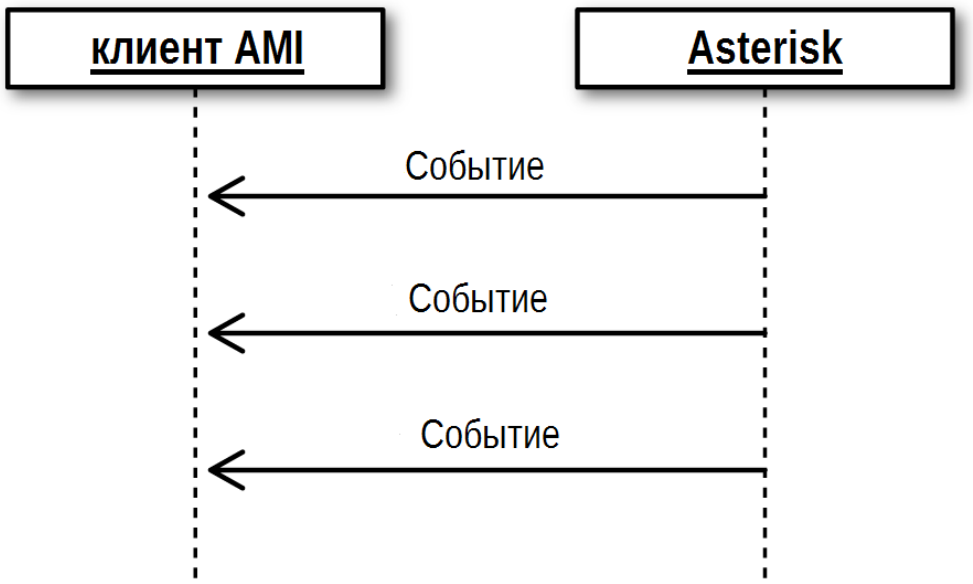

Рисунок 17-1. События диспетчера

Действия диспетчера - это запросы от клиента к Asterisk для выполнения некоторого действия и возврата результата (Рисунок 17-2). Например, действие AMI инициирует запросы, чтобы Asterisk

создал новый вызов, и, естественно, клиентскому приложению потребуются ответы от Asterisk, чтобы указать ход выполнения этого действия.

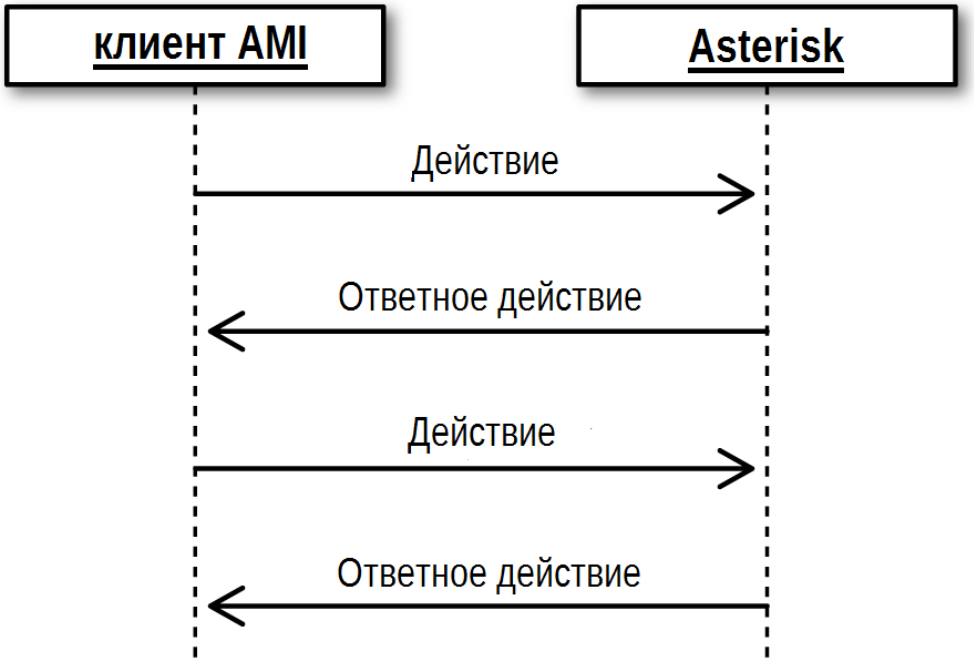

Рисунок 17-2. Действия диспетчера

Другие действия менеджера - это запросы данных. Например, есть действие - получить список всех активных каналов в системе: сведения о каждом канале доставляются как событие. Когда список результатов будет завершен - будет отправлено сообщение о том, что цель достигнута. См. Рисунок 17-3 для графического представления клиента, отправляющего этот тип управляющего действия и получающего список ответов.

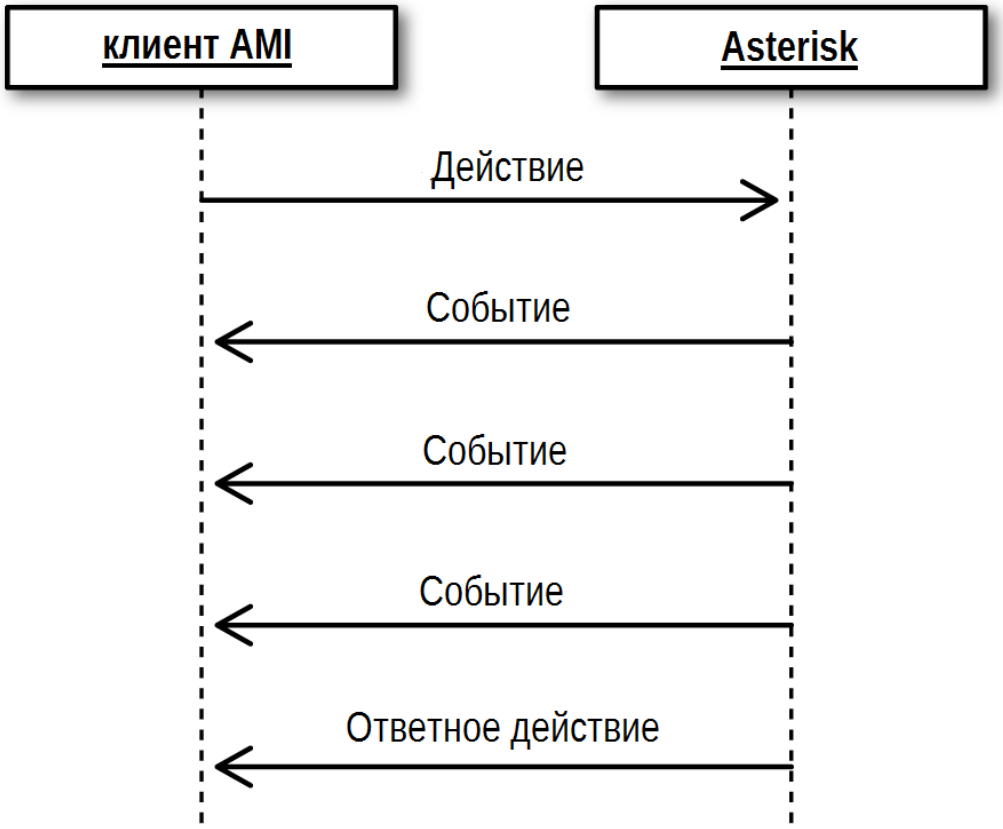

Рисунок 17-3. Действия диспетчера, возвращающие список данных

Кодировка сообщений

Все сообщения AMI, включая события, действия и ответы на действия, кодируются одинаково. Сообщения являются текстовыми, со строками, заканчивающимися возвратом каретки и символом перевода строки. Сообщение завершается пустой строкой:

```
Header1: This is the first header<CR><LF>
Header2: This is the second header<CR><LF>
Header3: This is the last header of this message<CR><LF>
<CR><LF>
```

Если вы запускаете тесты из telnet-клиента - это означает, что после последней строки инструкций вам нужно будет дважды нажать клавишу Enter.

События

События всегда имеют заголовок `Event` и заголовок `Privilege`. В заголовке `Event` указывается имя события, а в заголовке `Privilege` - уровни разрешений, связанные с данным событием. Любые другие заголовки, включенные в событие, являются специфичными для данного типа события. Вот вам пример:

```
Event: Hangup
Privilege: call,all
Channel: SIP/0004F2060EB4-00000000
Uniqueid: 1283174108.0
CallerIDNum: 2565551212
CallerIDName: Russell Bryant
Cause: 16
Cause-txt: Normal Clearing
```

CLI Asterisk включает в себя `manager show events` и `manager show event <event>`. Выполните эти команды в CLI Asterisk, чтобы получить список событий или узнать подробности конкретного события.

Не забывайте, что отличным справочником для всех вещей Asterisk, включая AMI, является официальная Asterisk wiki.

Действия

При выполнении действия *необходимо* включить заголовок `Action`. Заголовок `Action` определяет, какое действие выполняется. Остальные заголовки являются аргументами для действия и могут потребоваться или не потребоваться в зависимости от действия.

Чтобы получить список заголовков, связанных с определенным действием, введите в CLI Asterisk команду `manager show command <Action>`. Чтобы получить полный список действий, поддерживаемых используемой версией Asterisk, введите `manager show commands`.

Окончательный ответ на действие обычно представляет собой сообщение, содержащее заголовок `Response`. Значение заголовка `Response` будет `Success`, если действие было выполнено успешно. Если действие не было успешно выполнено, то значение заголовка ответа будет `Error`. Например:

```
Action: Login
Username: hello
Secret: world

Response: Success
Message: Authentication accepted
```

AMI через HTTP

Помимо собственного TCP-интерфейса, можно также получить доступ к AMI по протоколу HTTP. Программисты с имеющимся опытом написания приложений, использующие веб-API, скорее всего предпочтут его, по сравнению с подключением TCP. В то время как интерфейс TCP предлагает только

один тип структуры сообщений - AMI через HTTP предлагает несколько вариантов кодирования. Вы можете получать ответы в том же формате, что и в TCP: в формате XML или в виде базовой HTML-страницы. Тип кодировки выбирается на основе поля в URL запросе. Варианты кодирования рассматриваются более подробно далее в этом разделе.

Аутентификация и обработка сессии

Существует два метода выполнения аутентификации против AMI через HTTP. Первый - это использование действия `Login`, аналогичного аутентификации с помощью собственного интерфейса TCP. Это метод, который использовался в примере быстрого запуска, как показано в AMI через HTTP.

После успешной аутентификации Asterisk предоставит файл cookie, который идентифицирует аутентифицированный сеанс. Вот пример ответа на действие `Login`, которое включает в себя файл cookie сеанса от Asterisk:

```
$ curl -v "http://localhost:8088/rawman?action=login&username=hello&secret=world"
```

Второй вариант аутентификации - это HTTP-дайджест аутентификации. В этом примере запрошенный тип кодировки, основанный на URL-запросе, является `rawman`. Чтобы указать, что следует использовать дайджест аутентификации HTTP, префикс типа кодировки в URL-адресе запроса должен содержать `a`:

```
$ curl -v --digest -u hello:world http://127.0.0.1:8088/arawman?action=ping
```

Кодирование /rawman (/arawman)

Тип кодирования `rawman` - это то, что до сих пор использовалось во всех примерах AMI через HTTP в этой главе. Ответы, полученные от запросов, использующих `rawman`, форматируются точно так же, как они были бы, если бы запросы были отправлены по прямому TCP-соединению к AMI.

```
curl -v "http://localhost:8088/rawman?action=login&username=hello&secret=world"

curl -v --digest -u hello:world http://127.0.0.1:8088/arawman?action=ping
```

Кодирование /manager (/amanager)

Тип кодировки `manager` предоставляет ответ в простой HTML-форме. Этот интерфейс в первую очередь полезен для экспериментов с AMI:

```
$ curl -v "http://localhost:8088/manager?action=login&username=hello&secret=world"

$ curl -v --digest -u hello:world http://localhost:8088/amanager?action=ping
```

Кодирование /mxml (/amxml)

Тип кодировки `mxml` предоставляет ответы на действия, закодированные в XML:

```
$ curl -v "http://localhost:8088/mxml?action=login&username=hello&secret=world"

$ curl -v --digest -u hello:world http://localhost:8088/amxml?action=ping
```

События диспетчера

При подключении к собственному интерфейсу TCP для AMI события доставляются асинхронно. При использовании AMI через HTTP необходимо получать события путем опроса для них. Вы получаете события по протоколу HTTP, выполняя действие `WaitEvent`. В следующем примере показано, как события могут быть извлечены с помощью действия `WaitEvent`. Шаги такие:

1. Запустите сеанс HTTP AMI с помощью действия `Login`.
2. Зарегистрируйте SIP-телефон на Asterisk, чтобы создать событие.
3. Извлеките событие с помощью действия `WaitEvent`.

Взаимодействие выглядит следующим образом:

```
$ wget --save-cookies cookies.txt \
> "http://localhost:8088/mxml?action=login&username=hello&secret=world" -O -

<ajax-response>
<response type='object' id='unknown'>
    <generic response='Success' message='Authentication accepted' />
</response>
</ajax-response>


$ wget --load-cookies cookies.txt \
< "http://localhost:8088/mxml?action=waitevent" -O -

<ajax-response>
<response type='object' id='unknown'>
    <generic response='Success' message='Waiting for Event completed.' />
</response>
<response type='object' id='unknown'>
    <generic event='PeerStatus' privilege='system,all'
             channeltype='SIP' peer='SIP/0000FFFF0004'
             peerstatus='Registered' address='172.16.0.160:5060' />
</response>
<response type='object' id='unknown'>
    <generic event='WaitEventComplete' />
</response>
</ajax-response>
```

Вам потребуется разработать механизмы в вашем приложении чтобы гарантировать что буферизованные события часто опрашиваются.

Пример использования

Большая часть этой главы до сих пор обсуждала концепции и конфигурацию, связанные с AMI. В этом разделе приведены некоторые примеры использования.

Инициирование вызова

AMI имеет действие `Originate`, которое можно использовать для инициирования вызова. Многие из принятых заголовков совпадают с параметрами, размещенными в файлах вызовов. В Таблице 17-1 перечислены заголовки, принятые действием `Originate`.

Таблица 17-1. Заголовки для действия Originate

Параметр	Пример значения	Описание
`ActionID`	`a3a58876-f7c9-4c28-aa97-50d8166f658d`	Этот заголовок принимается большинством действий AMI. Он используется для предоставления уникального идентификатора, который также будет включен во все ответы на действие. Он дает вам возможность определить с каким запросом связан ответ. Он важен, так как все действия, их ответы и события передаются по одному и тому же соединению (если только не используется AMI через HTTP).
`Channel`	`SIP/myphone`	Этот заголовок является критическим и обязательно должен быть указан. Он описывает исходящий вызов, который будет инициирован. Значение имеет тот же синтаксис, что и аргумент канала для приложения `Dial()` в диалплане.
`Context`	`default`	Этот заголовок используется для указания положения в диалплане, которое будет запущено после ответа на исходящий вызов. Заголовки `Context`, `Exten` и `Priority` должны быть использованы вместе. При использовании этих заголовков не следует использовать заголовки `Application` и `Data`.
`Exten`	`s`	Смотри документацию по заголовку `Context`.

Параметр	Пример значения	Описание
Priority	1	Смотри документацию по заголовку Context.
Application	ConfBridge	Заголовки Application и Data можно использовать вместо заголовков Context, Exten и Priority. В этом случае исходящий вызов напрямую соединяется с одним приложением после ответа на вызов.
Data	500	Смотри документацию по заголовку Application.
Timeout	30000	Этот заголовок определяет, как долго (в миллисекундах) ждать ответа, прежде чем отказаться от исходящего вызова. Значение по умолчанию - 30000 миллисекунд (30 секунд).
CallerID	Matthew Jordan <(555) 867-5309>	Этот заголовок можно использовать для указания идентификатора вызывающего абонента, используемого для исходящего вызова.
Account	someaccount	Этот заголовок задает код учетной записи CDR для исходящего вызова.
Variable	VARIABLE=VALUE или FUNCTION(arguments)=VALUE	Заголовок Variable может использоваться для задания как переменных канала, так и функций канала на исходящем канале. Его можно задать несколько раз.
Codecs	ulaw,alaw	Этот параметр можно использовать для ограничения количества кодеков, разрешенных для исходящего вызова. Если этот параметр не указан, то набор кодеков, настроенных в файле конфигурации драйвера канала, будет по-прежнему учитываться.
EarlyMedia	true	Если этот заголовок указан и установлен в true - исходящий вызов будет подключен к указанному добавочному номеру или приложению, как только появится какой-либо медиапоток.
Async	true	Если этот заголовок задан и имеет значение true, то этот вызов будет инициирован асинхронно. Это позволит вам продолжить выполнение других действий на AMI-соединении во время обработки вызова.

Самый простой пример использования действия Originate через telnet:

```
$ telnet localhost 5038

Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Asterisk Call Manager/4.0.3
```

Как только соединение установлено Вам необходимо войти в систему.

```
Action: Login
Username: hello
Secret: world

Response: Success
Message: Authentication accepted
```

Теперь вы готовы инициировать свой звонок. Мы делаем практически то же самое, что и с файлом вызова, только на этот раз с помощью AMI:

```
Action: Originate
Channel: PJSIP/SOFTPHONE_A
Context: sets
Exten: 103
Priority: 1
```

Вы должны услышать звонок SOFTPHONE_A. Как только вы ответите на него - вызов будет сделан на SOFTPHONE_B.

AMI больше не участвует в том, что происходит. Вы можете отключиться, и вызов будет продолжен (оставьте его в данный момент, так как мы собираемся работать с текущим вызовом далее).

```
Action: Logoff

Response: Goodbye
```

```
Message: Thanks for all the fish.
Connection closed by foreign host.
```

Если вы уже повесили трубку - это не проблема. Вам просто нужно будет восстановить вызов, что, конечно же, вы можете сделать, просто позвонив по одному номеру с другого (101-103 или как пожелаете).

Перенаправление вызова

Перенаправление (или transferring — трансфер, передача) вызова из AMI - еще одна функция, заслуживающая упоминания. Действие AMI `Redirect` можно использовать для отправки одного или двух каналов на любой другой модуль в диалплане Asterisk. Если вам нужно перенаправить два канала, которые соединены вместе, сделайте это с обоими одновременно. В противном случае - как только один канал будет перенаправлен, другой будет отключен.

Важно понимать, что каналы Asterisk не существуют до тех пор, пока не будет выполнен вызов. Имя, которое мы все считаем именем канала (например, `SOFTPHONE_A`), на самом деле не является именем канала, а просто ссылкой на данные, которые используются для создания канала. Присвоение имени каналу происходит при возникновении вызова (то есть, когда канал фактически создан). Все это означает, что вы должны определить полное название канала, прежде, чем сможете действовать на нем. Инициируйте вызов, а затем просмотрите `Event: Newchannel`, и увидите имя канала под заголовком `Channel:`.

```
Action: Originate
Channel: PJSIP/SOFTPHONE_A
Context: sets
Exten: 103
Priority: 1

Response: Success
Message: Originate successfully queued

Event: Newchannel
Privilege: call,all
Channel: PJSIP/SOFTPHONE_A-00000013
ChannelState: 0
ChannelStateDesc: Down
CallerIDNum: <unknown>
CallerIDName: <unknown>
ConnectedLineNum: <unknown>
ConnectedLineName: <unknown>
Language: en
AccountCode:
Context: sets
Exten: s
Priority: 1
Uniqueid: 1538939479.29
Linkedid: 1538939479.29
```

Событие `Newchannel` предоставит имя созданного канала, которым в данном примере является `PJSIP/SOFTPHONE_A-00000013`.

Вам нужно будет отслеживать эти названия каналов, если вы хотите правильно выполнять действия по текущим вызовам. Как только вызов заканчивается - канал разрушается. Новому вызову, использующему ту же конечную точку, будет присвоено другое имя канала. Одно определение канала может поддерживать несколько вызовов (например, возможны несколько вызовов на телефон), и именно поэтому имя канала отличается от определения канала.

Вы можете перенаправить один канал (другой будет отключен):

```
Action: Redirect
Channel: PJSIP/SOFTPHONE_A-00000013
```

```
Exten: 209
Context: sets
Priority: 1
```

Или можете перенаправить два канала:

```
Action: Redirect
Channel: PJSIP/SOFTPHONE_A-00000015
Context: sets
Exten: 209
Priority: 1
ExtraChannel: PJSIP/SOFTPHONE_B-00000016
ExtraContext: sets
ExtraExten: 209
ExtraPriority: 1
```

Функция перенаправления позволяет создавать мощные внешние приложения, которые могут управлять текущими вызовами.

Фреймворки разработки

Многие разработчики приложений пишут код, который напрямую взаимодействует с AMI. Однако существует ряд фреймворков, которые были созданы с целью облегчить разработку приложений AMI. Если вы ищете фреймворки Asterisk на популярном языке программирования - по вашему выбору, вы, скорее всего, найдете один. На вас лежит ответственность за определение пригодности структуры, в которой вы заинтересованы. Некоторые вещи, которые вы должны искать в рамках включают в себя:

Зрелость

Этот проект существует уже несколько лет? Зрелый проект гораздо менее вероятно будет иметь серьезные ошибки в нем.

Поддержка

Проверьте возраст последнего обновления. Если проект не обновлялся в течение пяти лет - есть большая вероятность что он был заброшен. Возможно, он еще пригодится, но вы будете предоставлены сами себе. Аналогично, как выглядит баг-трекер? Есть ли много важных ошибок, которые игнорируются? (Будьте проницательны здесь, так как часто реалии поддержки свободного проекта требуют дисциплинированной сортировки - не все функции будут добавлены.)

Качество кода

Это хорошо написанная структура? Если он не был хорошо спроектирован - вы должны знать об этом, когда решаете стоит ли доверять ему свой проект.

Сообщество

Есть ли активное сообщество разработчиков, использующих этот проект? Вероятно, вам понадобится помощь - будет ли она доступна, когда вы в ней будете нуждаться?

Документация

Код должен быть хорошо прокомментирован, но в идеале необходима вики или другая официальная документация для поддержки библиотеки.

В Таблице 17-2 перечислены некоторые структуры, которые, как мы обнаружили на момент написания данной статьи, соответствовали предыдущим критериям. Там могут быть и другие.

Таблица 17-2. Фреймворки разработки AMI

Фреймворк	Язык
Adhearsion	Ruby
StarPy	Python
Asterisk-Java	Java

Фреймворк	Язык
AsterNET	.NET
ami-io	Node.js
panoramisk	Python

Вывод

AMI предоставляет API для мониторинга событий из системы Asterisk, а также запрашивает Asterisk выполнять широкий спектр действий. Был предоставлен интерфейс HTTP и был разработан ряд фреймворков, облегчающих разработку приложений.

Кофеин. Шлюз к наркотикам.
– Эдди Веддер

Диалплан Asterisk превратился в простой, но мощный программный интерфейс для обработки вызовов. Однако многие люди, особенно с опытом программирования, предпочитают реализовывать обработку вызовов на традиционном языке программирования. Asterisk Gateway Interface (AGI) позволяет разрабатывать управление вызовами от первого лица на выбранном вами языке программирования.

Быстрый запуск

В этом разделе приведен краткий пример использования AGI.

Во-первых, давайте создадим скрипт, который мы собираемся запустить. Скрипты AGI как правило помещаются в */var/lib/asterisk/agi-bin*.

```
$ cd /var/lib/asterisk/agi-bin
$ vim hello-world.sh
#!/bin/bash
# Получим все переменные, отправленные Asterisk
while read VAR && [ -n ${VAR} ] ; do : ; done
# Отвечаем на вызов.
echo "ANSWER"
read RESPONSE
# Произнести буквы из "Hello World"
echo 'SAY ALPHA "Hello World" ""'
read RESPONSE
exit 0
$ chown asterisk:asterisk hello-world.sh
$ chmod 700 hello-world.sh
```

Теперь добавьте следующую строку в */etc/asterisk/extensions.conf* в контекст `[sets]`:

```
exten => 237,1,AGI(hello-world.sh)
```

Сохраните и перезагрузите свой диалплан и теперь, когда вы звоните на номер 237, то должны услышать, как Эллисон произносит “Hello World.”

Варианты AGI

Существует несколько вариантов AGI, которые отличаются в первую очередь методом, используемым для связи с Asterisk. Полезно быть в курсе всех вариантов для совершения лучшего выбора, основанного на потребностях вашего приложения.

Process-Based AGI

Process-based AGI (AGI на основе процесса) является простейшим вариантом AGI. Пример быстрого запуска в начале этой главы является примером сценария Process-based AGI. Скрипт вызывается с помощью приложения `AGI()` из диалплана Asterisk. Запускаемое приложение указывается в качестве первого аргумента функции `AGI()`. Если не указан полный путь - приложение должно находиться в каталоге */var/lib/asterisk/agi-bin*. Аргументы, передаваемые приложению AGI, могут быть указаны в качестве дополнительных аргументов приложения `AGI()` в диалплане Asterisk. Синтаксис такой:

```
AGI(command[,arg1[,arg2[,...]]])
```

Убедитесь что приложение имеет соответствующие разрешения на исполнение пользователем Asterisk. В противном случае функция `AGI()` завершится ошибкой.

Как только Asterisk выполнит ваше приложение AGI - связь между Asterisk и вашим приложением будет осуществляться через `stdin` и `stdout`. Более подробно об этом сообщении будет рассказано в разделе “Обзор коммуникаций AGI”. Для получения более подробной информации о вызове `AGI()` из диалплана проверьте документацию, встроенную в Asterisk:

```
*CLI> core show application AGI
```

Плюсы Process-Based AGI (на основе процессов)

Это самая простая форма реализации AGI.

Минусы Process-Based AGI

Это наименее эффективная форма AGI с точки зрения потребления ресурсов. Вместо неё системы с высокой нагрузкой должны рассматривать FastAGI, описанный в разделе "FastAGI - AGI через TCP".

EAGI

EAGI (Enhanced AGI - расширенный AGI) является легким вариантом `AGI()`. Он вызывается в диалплане Asterisk как `EAGI()`. Разница в том, что в дополнение к связи через `stdin` и `stdout` Asterisk также обеспечивает однонаправленный аудиопоток, поступающий из канала на файловый дескриптор 3. Для получения более подробной информации о том, как вызвать `EAGI()` из диалплана Asterisk, проверьте документацию, встроенную в Asterisk:

```
*CLI> core show application EAGI
```

Плюсы расширенного AGI

Он проще Process-based AGI, включая канал аудиопотока только для чтения. Это единственный вариант, предлагающий эту функцию.

Минусы расширенного AGI

Поскольку для запуска приложения для каждого вызова необходимо создать новый процесс - он имеет те же проблемы эффективности, что и обычный Process-Based AGI.

Для альтернативного способа получения доступа к аудио вне Asterisk - рассмотрите возможность использования JACK. Asterisk имеет модуль для интеграции JACK, называемый `app_jack`. Он предоставляет приложение `Jack()` и функцию диалплана `JACK_HOOK()`.

FastAGI - AGI через TCP

FastAGI - это термин, используемый для управления вызовами AGI через TCP-соединение. При использовании Process-Based AGI экземпляр приложения AGI выполняется в системе для каждого вызова и связь с этим приложением осуществляется через `stdin` и `stdout`. С помощью FastAGI осуществляется TCP-соединение с сервером FastAGI. Управление вызовами осуществляется с использованием того же протокола AGI, но связь осуществляется через TCP-соединение и не требует запуска нового процесса для каждого вызова. Протокол AGI более подробно рассматривается в разделе "Обзор коммуникаций AGI". Использование FastAGI гораздо более масштабируемо, чем Process-Based AGI, хотя и более сложно в реализации.

Чтобы использовать FastAGI - вы вызываете приложение `AGI()` в диалплане Asterisk, но вместо имени приложения, которое нужно выполнить, вы предоставляете URL-адрес `agi://`. Например:

```
exten => 238,1,AGI(agi://127.0.0.1)
```

Номер порта по умолчанию для соединения FastAGI - 4573. После двоеточия к URL-адресу можно добавить другой номер порта. Например:

```
exten => 238,1,AGI(agi://127.0.0.1:4574)
```

Так же, как и в случае AGI на основе процессов, в приложение FastAGI могут передаваться аргументы. Для этого добавьте их в качестве дополнительных аргументов в приложение AGI(), разделенных запятыми:

```
exten => 238,1,AGI(agi://192.168.1.199,arg1,arg2,arg3)
```

FastAGI также поддерживает использование записей DNS SRV, если вы предоставляете URL в виде hagi://. Используя записи SRV, DNS-серверы могут возвращать несколько узлов, к которым Asterisk может попытаться подключиться. Это может быть использовано для обеспечения высокой доступности и балансировки нагрузки. В следующем примере, для нахождения доступного для подключения сервера FastAGI, Asterisk выполнит поиск DNS для _agi._tcp.shifteight.org:

```
exten => 238,1,AGI(hagi://shifteight.org)
```

В этом примере DNS-сервера для домена shifteight.org потребуется хотя бы одна SRV-запись, настроенная для _agi._tcp.shifteight.org.

Плюсы FastAGI

Более эффективен чем process-based AGI. Вместо того, чтобы порождать новый процесс на вызов, можно построить сервер FastAGI для обработки нескольких вызовов.

DNS может использоваться для достижения высокой доступности и балансировки нагрузки между серверами FastAGI для дальнейшего повышения масштабируемости.

Минусы FastAGI

Он является более сложным при реализации сервера FastAGI, чем реализация приложения AGI на основе процессов.

Async AGI - AMI-контролируемый AGI

Async AGI позволяет приложению, использующему интерфейс AMI, асинхронно ставить команды AGI в очередь для выполнения на канале. Это может быть особенно полезно, если вы уже широко используете AMI и хотите улучшить свое приложение для обработки управления вызовами, а не писать подробный диалплан Asterisk или разрабатывать отдельный сервер FastAGI.

Дополнительную информацию об Asterisk Manager Interface можно найти в Главе 17.

Async AGI вызывается приложением AGI() в диалплане Asterisk. Аргумент для AGI() должен быть agi:async, как показано в следующем примере:

```
exten => 239,AGI(agi:async)
```

Дополнительную информацию о том, как использовать Async AGI через AMI, можно найти в следующем разделе.

Плюсы Async AGI

Существующее приложение AMI можно использовать для управления вызовами с помощью команд AGI.

Минусы Async AGI

Это самый сложный способ реализации AGI.

Настройка /etc/asterisk/manager.conf для Async AGI

Чтобы использовать Async AGI, учетная запись AMI должна иметь разрешение `agi` как на `read`, так и на `write`. Например, следующий пользователь определенный в *manager.conf* будет иметь возможность как а) выполнять действия менеджера AGI, так и б) получать события AGI:

```
; Определить пользователя с именем "hello" и паролем "world".
; Предоставить этому пользователю разрешения на чтение/запись для AGI.
;
[hello]
secret = world
read = agi
write = agi
```

Обзор коммуникаций AGI

В предыдущем разделе были рассмотрены возможные варианты использования AGI. Этот раздел содержит более подробные сведения о том, как ваше пользовательское приложение AGI взаимодействует с Asterisk после вызова функции `AGI()`.

Установка сеанса AGI

После вызова `AGI()` или `EAGI()` из диалплана Asterisk - в приложение AGI передается некоторая информация для установки сеанса. В этом разделе рассматривается какие шаги предпринимаются в начале сеанса AGI для различных вариантов.

Process-based AGI/FastAGI

Для приложения AGI на основе процессов или подключения к серверу FastAGI переменные, перечисленные в Таблице 18-1, будут первыми фрагментами информации, отправленными из Asterisk в ваше приложение. Каждая переменная будет находиться на своей собственной строке, в виде:

`agi_`*`переменная`*`:` *`значение`*

Таблица 18-1. Переменные среды AGI

Переменная	Значение/пример	Описание
`agi_request`	`hello-world.sh`	Первый аргумент, который был передан в приложение `AGI()` или `EAGI()`. Для process-based AGI - это имя выполненного приложения AGI. Для FastAGI это будет URL-адрес, использованный для подключения к серверу FastAGI.
`agi_channel`	`SIP/0004F2060EB4-00000009`	Имя канала, выполнившего команду приложения `AGI()` или `EAGI()`.
`agi_language`	`en`	Язык, установленный на `agi_channel`.
`agi_type`	`SIP`	Тип канала для `agi_channel`.
`agi_uniqueid`	`1284382003.9`	`uniqueid` для `agi_channel`.
`agi_version`	`1.8.0-beta4`	Используемая версия Asterisk.
`agi_callerid`	`12565551212`	Полная строка callerID, установленная на `agi_channel`.
`agi_calleridname`	`Russell Bryant`	Имя caller ID, установленное на `agi_channel`.
`agi_callingpres`	`0`	Представление вызывающего абонента, связанное с caller ID, установленное на `agi_channel`. Дополнительные сведения см. в выводе `core show function CALLERPRES` в CLI Asterisk.
`agi_callingani2`	`0`	ANI2 абонента, связанный с `agi_channel`.
`agi_callington`	`0`	TH (тип номера) ID абонента, связанный с `agi_channel`.

Переменная	Значение/пример	Описание
agi_callingtns	0	Набранный номер TNS (выбор транзитной сети), связанный с agi_channel.
agi_dnid	7010	Набранный номер, связанный с agi_channel.
agi_rdnis	unknown	Номер перенаправления, связанный с agi_channel.
agi_context	phones	Контекст диалплана, в котором находился agi_channel при выполнении приложения AGI() или EAGI().
agi_extension	500	Расширение в диалплане, которое выполнялось agi_channel при запуске приложения AGI() или EAGI().
agi_priority	1	Приоритет agi_extension в agi_context, в котором выполнилось AGI() или EAGI().
agi_enhanced	0.0	Указание на то, был ли использован AGI() или EAGI() из диалплана. 0.0 указывает на то, что был использован AGI(). 1.0 указывает на то, что был использован EAGI().
agi_accountcode	myaccount	Код учетной записи, связанный с agi_channel.
agi_threadid	140071216785168	threadid потока в Asterisk, на котором выполняется приложение AGI() или EAGI(). Это может быть полезно для связывания журналов, созданных приложением AGI, с журналами, созданными Asterisk, поскольку журналы Asterisk содержат идентификаторы потоков.
agi_arg_*<argument number>*	my argument	Эти переменные предоставляют содержимое дополнительных аргументов, предоставленных приложению AGI() или EAGI().

Пример переменных, которые могут быть отправлены в приложение AGI, см. в разделе выходные данные отладки связи AGI в разделе Быстрый старт. Конец списка переменных будет обозначен пустой строкой. Код обрабатывает эти переменные путем считывания строк ввода в цикле, пока не будет получена пустая строка. В этот момент приложение продолжается и начинает выполнять команды AGI.

Async AGI

При использовании Async AGI Asterisk будет отправлять событие диспетчера называемое AsyncAGI чтобы инициировать сеанс Async AGI. Это событие позволит приложениям, прослушивающим события диспетчера, взять на себя управление вызовом с помощью события менеджера AGI. Вот пример события менеджера, отправленного Asterisk:

```
Event: AsyncAGI
Privilege: agi,all
SubEvent: Start
Channel: SIP/0000FFFF0001-00000000
Env: agi_request%3A%20async%0Aagi_channel%3A%20SIP%2F0000FFFF0001-00000000%0A \
     agi_language%3A%20en%0Aagi_type%3A%20SIP%0A \
     agi_uniqueid%3A%201285219743.0%0A \
     agi_version%3A%201.8.0-beta5%0Aagi_callerid%3A%2012565551111%0A \
     agi_calleridname%3A%20Julie%20Bryant%0Aagi_callingpres%3A%200%0A \
     agi_callingani2%3A%200%0Aagi_callington%3A%200%0Aagi_callingtns%3A%200%0A \
     agi_dnid%3A%20111%0Aagi_rdnis%3A%20unknown%0Aagi_context%3A%20LocalSets%0A \
     agi_extension%3A%20111%0Aagi_priority%3A%201%0Aagi_enhanced%3A%200.0%0A \
     agi_accountcode%3A%20%0Aagi_threadid%3A%20-1339524208%0A%0A
```

Значение самого заголовка Env в этом событии менеджера AsyncAGI находится все на одной линии. Длинное значение заголовка Env было закодировано URL-адресом.

Команды и ответы

После настройки сеанса AGI Asterisk начинает выполнять обработку вызовов в ответ на команды, отправленные из приложения AGI. Как только команда AGI будет выдана Asterisk - никакие другие команды не будут обработаны на этом канале, пока текущая не будет завершена. Когда он закончит обработку команды - Asterisk ответит с результатом.

AGI обрабатывает команды последовательно. После выполнения команды никакие другие команды не могут быть выполнены до тех пор, пока Asterisk не вернет ответ. Некоторые команды могут выполняться очень долго. Например, в команде `EXEC` AGI выполняет приложение Asterisk. Если есть команда `EXEC Dial` - связь с AGI блокируется до тех пор, пока вызов не будет выполнен. Если на данном этапе вашему приложению AGI необходимо продолжить взаимодействие с Asterisk - оно может сделать это с помощью AMI, который рассматривается в Главе 17.

Вы можете получить полный список доступных команд AGI из консоли Asterisk, выполнив команду `agi show commands`. Эти команды описаны в Таблице 18-2. Для получения более подробной информации о конкретной команде AGI, включая сведения о синтаксисе для всех ожидаемых аргументов, используйте `agi show commands topic` *`COMMAND`*. Например, чтобы просмотреть встроенную документацию для команды AGI `ANSWER` - вы бы использовали `agi show commands topic ANSWER`.

Таблица 18-2. Команды AGI

Команда AGI	Описание
`ANSWER`	Ответ на входящий вызов.
`ASYNCAGI BREAK`	Завершение сеанса Async AGI и возврат канала в диалплан Asterisk.
`CHANNEL STATUS`	Получение статуса канала. Используется для получения текущего состояния канала, такого как up (ответ), down (трубка положена) или вызов.
`DATABASE DEL`	Удаляет пару ключ/значение из встроенной базы данных AstDB.
`DATABASE DELTREE`	Удаляет дерево пар ключ/значение из встроенной базы данных AstDB.
`DATABASE GET`	Извлекает значение для ключа в базе данных AstDB.
`DATABASE PUT`	Устанавливает значение для ключа в базе данных AstDB.
`EXEC`	Выполняет приложение диалплана Asterisk на канале. Эта команда очень полезна в том, что между `EXEC` и `GET FULL VARIABLE`, вы можете сделать все что угодно с вызовом, который можете сделать из диалплана Asterisk.
`GET DATA`	Считывание цифр от вызывающего абонента.
`GET FULL VARIABLE`	Оценивает выражение диалплана Asterisk. Вы можете отправить строку, содержащую переменные и/или функции диалплана и Asterisk вернет результат после выполнения соответствующих подстановок. Эта команда очень полезна в том, что между `EXEC` и `FET FULL VARIABLE`, вы можете сделать все, что угодно с вызовом, который вы можете сделать из диалплана Asterisk.
`GET OPTION`	Потоковая передача звукового файла во время ожидания цифры от вызывающего абонента. Это похоже на приложение диалплана `Background()`.
`GET VARIABLE`	Извлечение значения переменной канала.
`HANGUP`	Завершение вызова.[a]
`NOOP`	Ничего не делать. Вы получите результата запроса от этой команды, как и любой другой. Он может быть использован в качестве простого теста связи с Asterisk.
`RECEIVE CHAR`	Получить один символ. Это работает только для типов каналов, которые его поддерживают, таких как iax2, использующий фреймы `TEXT`, или SIP, использующий метод `MESSAGE`.

Команда AGI	Описание
RECEIVE TEXT	Получение текстового сообщения. Это работает только в тех же случаях, что и RECEIVE CHAR.
RECORD FILE	Запись аудиопотока от вызывающего абонента в файл. Это блокирующая операция, аналогичная приложению диалплана Record(). Чтобы записать вызов в фоновом режиме, во время выполнения других операций, используйте EXEC Monitor или EXEC MixMonitor.
SAY ALFA	Озвучить строку символов. Вы можете найти пример этого в разделе "Быстрый старт". Чтобы получить локализованную обработку этой и другой команды SAY, установите язык канала либо в файле конфигурации устройств (например, *sip.conf*), либо в диалплане установкой функцией диалплана CHANNEL(language).
SAY DIGIT	Озвучить строку цифр. Например, 100 будет озвучено "один ноль ноль", если язык канала установлен на русский.
SAY NUMBER	Озвучить номер телефона. Например, 100 будет сказано как "сто", если язык канала установлен на русский.
SAY PHONETIC	Произнести строку символов, но используя общее слово для каждой буквы (Альфа, Браво, Чарли...).
SAY DATE	Произнести заданную дату
SAY TIME	Произнести заданное ВРЕМЯ
SAY DATETIME	Произнести заданную дату и время, используя указанный формат.
SEND IMAGE	Отправить изображение в канал. IAX2 поддерживает это, но нет никаких активно разработанных клиентов IAX2, о которых мы знаем, поддерживающих это.
SEND TEXT	Отправить текст в канал, который его поддерживает. Это может быть использовано по крайней мере с каналами SIP и IAX2.
SET AUTOHANGUP	Запланировать отключение канала в указанный момент времени в будущем.
SET CALLERID	Установить имя и номер CallerID на канале.
SET CONTEXT	Установить текущий контекст диалплана на канале.
SET EXTENSION	Установить текущее расширение диалплана на канале.
SET MUSIC	Запуск или остановка музыки на удержание на канале.
SET PRIORITY	Установить текущий приоритет диалплана на канале.
SET VARIABLE	Задать для переменной канала указанное значение.
STREAM FILE	Потоковая передача содержимого файла в канал.
CONTROL STREAM FILE	Передать содержимое файла в канал, но также позволить каналу управлять потоком. Например, канал может приостановить, перемотать поток назад или вперед.
TDD MODE	Переключить режим TDD (Telecommunications Device for the Deaf - телекоммуникационное устройство для глухих) на канале.
VERBOSE	Отправить на канал сообщение verbose logger. Подробные сообщения отображаются в консоли Asterisk, если значение параметра verbose достаточно высокое. Подробные сообщения также будут отправляться в любой файл журнала, настроенный для verbose регистратора канала в */etc/asterisk/logger.conf*.
WAIT FOR DIGIT	Дождитесь, пока вызывающий абонент нажмет цифру.
SPEECH CREATE	Инициализировать распознавание речи. Это необходимо сделать перед использованием других речевых команд AGI. [b]
SPEECH SET	Установить настройку речевого движка. Доступные настройки относятся только к используемому механизму распознавания речи.
SPEECH DESTROY	Уничтожить ресурсы, выделенные для выполнения распознавания речи. Эта команда должна быть последней выполненной речевой командой.
SPEECH LOAD GRAMMAR	Загрузить грамматику

Команда AGI	Описание
`SPEECH UNLOAD GRAMMAR`	Выгрузить грамматику
`SPEECH ACTIVATE GRAMMAR`	Активировать грамматику, которая была загружена.
`SPEECH DEACTIVATE GRAMMAR`	Деактивировать грамматику
`SPEECH RECOGNIZE`	Воспроизвести запрос и выполнить распознавание речи, а также ждать ввода цифр.
`GOSUB`	Выполнить функцию диалплана. Она будет выполняться так же, как и приложение диалплана `GoSub()`.

[a] Когда используется команда AGI `HANGUP` канал отключается не сразу. Вместо этого канал помечается как нуждающийся в отключении. Ваше приложение AGI должно завершиться раньше, чем Asterisk продолжит и выполнит фактический процесс отключения.

[b] Хотя Asterisk включает в себя основной API для обработки распознавания речи, он не поставляется с модулем, обеспечивающим механизм распознавания речи. В настоящее время Digium предоставляет два коммерческих варианта распознавания речи: Lumenvox и Vestec.

Process-based AGI/FastAGI

Команды AGI отправляются в Asterisk в одну строку. Строка должна заканчиваться символом новой строки. После отправки команды в Asterisk дальнейшие команды не будут обрабатываться до тех пор, пока последняя не будет завершена и ответ не будет отправлен обратно в приложение AGI. Вот пример ответа на команду AGI:

```
200 result=0
```

Консоль Asterisk позволяет отлаживать взаимодействие с приложением AGI. Чтобы включить отладку связи AGI - выполните команду `agi set debug on`. Чтобы отключить - `agi set debug off`. Пока режим отладки включен - вся связь с приложением AGI и из него будет выводиться в консоли Asterisk. Пример такого вывода можно найти в разделе Быстрый старт.

Async AGI

При использовании Async AGI можно выполнять команды с помощью действий менеджера AGI. Чтобы просмотреть встроенную документацию для действий менеджера AGI - выполните `manager show command AGI` в CLI Asterisk. Демонстрация поможет выяснить как выполняются команды AGI с помощью метода Async AGI. Во-первых, расширение создается в диалплане, который выполняет сеанс Async AGI в канале:

```
exten = > 240,AGI(agi:async)
```

При выполнении приложения AGI вызываемое событие менеджера `AsyncAGI` будет отправлено вместе со всеми переменными среды AGI. Подробная информация об этом событии находится в разделе "Async AGI" . После этого действия менеджера AGI могут начать выполняться через AMI.

Ниже приведен пример выполнения действия диспетчера и его события, генерируемые во время обработки Async AGI. После первоначального выполнения действия менеджера AGI немедленно появляется ответ, указывающий на то, что команда была поставлена в очередь на выполнение. Позже появляется событие менеджера, указывающее что команда в очереди была выполнена. Заголовок идентификатора команды можно использовать для связывания начального запроса с событием, указывающим на то, что команда была выполнена:

```
Action: AGI
Channel: SIP/0004F2060EB4-00000013
ActionID: my-action-id
CommandID: my-command-id
Command: VERBOSE "Puppies like cotton candy." 1

Response: Success
```

```
ActionID: my-action-id
Message: Added AGI command to queue

Event: AsyncAGI
Privilege: agi,all
SubEvent: Exec
Channel: SIP/0004F2060EB4-00000013
CommandID: my-command-id
Result: 200%20result%3D1%0A
```

Следующие выходные данные - это то, что было замечено в консоли Asterisk во время этого сеанса Async AGI:

```
      -- Executing [7011@phones:1] AGI("SIP/0004F2060EB4-00000013",
         "agi:async") in new stack
 agi:async: Puppies like cotton candy.
  == Spawn extension (phones, 7011, 1)
exited non-zero on 'SIP/0004F2060EB4-00000013'
```

Завершение сеанса AGI

Сеанс AGI завершается когда приложение AGI готово к его завершению. Подробности того как это происходит, зависят от того, использует ли ваше приложение process-based AGI, FastAGI или async AGI.

Process-based AGI/FastAGI

Ваше приложение AGI может выйти или закрыть свое соединение в любое время. Если канал не был отключен до завершения работы приложения - выполнение диалплана будет продолжено.

Если отключение канала происходит пока сеанс AGI все еще активен - Asterisk предоставит уведомление о том, что это произошло, чтобы ваше приложение могло соответствующим образом настроить свою работу.

Если канал завершается пока приложение AGI все еще выполняется - произойдет несколько вещей. Если команда AGI находится в середине выполнения - вы можете получить код результата `-1`. Однако вы не должны зависеть от этого, поскольку не все команды AGI требуют взаимодействия с каналом. Если выполняемая команда не требует взаимодействия с каналом - результат не будет отражать завершение.

Следующее что происходит после того, как канал завершается - это отправка уведомления о завершении в ваше приложение. Для process-based AGI, сигнал `SIGHUP` будет отправлен в процесс для уведомления о его завершении. Для подключения FastAGI - Asterisk отправит строку, содержащую слово `HANGUP`.

> Если вы хотите отключить функцию отправки Asterisk сигнала `SIGHUP` для вашего приложения process-based AGI или строки `HANGUP` для вашего сервера FastAGI - вы можете сделать это, установив переменную канала `AGISIGHUP` как показано в этом коротком примере:
>
> ```
> ; нет SIGHUP (AGI) или HANGUP (FastAGI)
> exten => 237,1,Set(AGISIGHUP=no)
>     same => n,AGI(hello-world.sh)
> ```

После того, как завершение канала произошло - единственные команды AGI, которые могут использоваться, являются теми, которые не требуют взаимодействия с каналом. Документация для команд AGI, встроенных в Asterisk, включает указание на то, можно ли использовать каждую команду после того, как канал был отключен.

Async AGI

При использовании Async AGI интерфейс менеджера предоставляет механизмы для уведомления о завершении канала. Если вы хотите завершить сеанс Async AGI для канала - необходимо выполнить команду `ASYNCAGI BREAK`. Когда сеанс Async AGI закончится - Asterisk отправит событие менеджера `AsyncAGI` с `SubEvent` об `End`. Ниже приведен пример завершения сеанса Async AGI:

```
Action: AGI
Channel: SIP/0004F2060EB4-0000001b
ActionID: my-action-id
CommandID: my-command-id
Command: ASYNCAGI BREAK

Response: Success
ActionID: my-action-id
Message: Added AGI command to queue

Event: AsyncAGI
Privilege: agi,all
SubEvent: End
Channel: SIP/0004F2060EB4-0000001b
```

На этом этапе канал возвращается к следующему шагу в диалплане Asterisk (если он еще не был отключен).

Пример: Доступ к базе данных учетной записи

Пример 18-1 - это пример сценария AGI. Чтобы запустить этот скрипт - сначала поместите его в каталог */var/lib/asterisk/agi-bin*. Тогда вы бы выполнили его из диалплана Asterisk вот так:

```
exten = > 241,1, AGI(account-lookup.py)
    same => n,Hangup()
```

Этот пример написан на Python и очень скудно документирован для краткости. Он демонстрирует как сценарий AGI взаимодействует с Asterisk с помощью `stdin` и `stdout`.

Сценарий предлагает пользователю ввести номер учетной записи, а затем воспроизводит значение, связанное с этим номером. В интересах краткости мы жестко закодировали несколько поддельных учетных записей в скрипт — это, очевидно, будет что-то нормально обрабатываемое подключением к базе данных.

Сценарий намеренно лаконичен, так как мы заинтересованы в кратком показе некоторых функций AGI, не заполняя эту книгу страницами кода.

Пример 18-1. account-lookup.py

```
#!/usr/bin/env python
# Пример для AGI (Asterisk Gateway Interface).

import sys

def agi_command(cmd):
    '''Вписать команду и вернуть ответ'''
    print cmd
    sys.stdout.flush() #очистка буфера
    return sys.stdin.readline().strip() # строка пробелов

asterisk_env = {} # чтение переменной среды AGI из Asterisk
while True:
    line = sys.stdin.readline().strip()
    if not len(line):
        break
    var_name, var_value = line.split(':', 1)
    asterisk_env[var_name] = var_value

# Поддельная "база данных" учетных записей.
```

```
ACCOUNTS = {
    '12345678': {'balance': '50'},
    '11223344': {'balance': '10'},
    '87654321': {'balance': '100'},
}

response = agi_command('ANSWER')

# три аргумента: приглашение, тайм-аут, максимальная длина
response = agi_command('GET DATA enter_account 3000 8')

if 'timeout' in response:
    response = agi_command('STREAM FILE goodbye ""')
    sys.exit(0)

# Ответ будет выглядеть как: 200 result=<digits>
# С разделителем '=' мы получаем индекс 1
account = response.split('=', 1)[1]

if account == '-1': # ответ при ошибке
    response = agi_command('STREAM FILE astcc-account-number-invalid ""')
    response = agi_command('HANGUP')
    sys.exit(0)

if account not in ACCOUNTS: # неверный
    response = agi_command('STREAM FILE astcc-account-number-invalid ""')
    sys.exit(0)

balance = ACCOUNTS[account]['balance']

response = agi_command('STREAM FILE account-balance-is ""')
response = agi_command('SAY NUMBER %s ""' % (balance))
sys.exit(0)
```

Фреймворки разработки

Был предпринят ряд усилий по созданию фреймворков или библиотек, облегчающих программирование AGI. Вы заметите, что некоторые из них уже упоминались в Главе 17. Так же, как и в случае с AMI, при оценке фреймворка мы рекомендуем вам найти тот, который соответствует следующим критериям:

Зрелость

Этот проект существует уже несколько лет? Зрелый проект гораздо менее вероятно будет иметь серьезные ошибки в нем.

Поддержка

Проверьте дату последнего обновления. Если проект не обновлялся в течении нескольких лет - есть большая вероятность что он был заброшен. Он все еще может быть полезен, но вы будете предоставлены сами себе. Аналогично, как выглядит трекер ошибок? Много ли важных ошибок, которые игнорируются? (Будьте проницательны здесь, так как часто реалии поддержки свободного проекта требуют тщательного отбора — не все функции будут добавлены.)

Качество кода

Это хорошо написанная структура? Если он не был спроектирован хорошо - вы должны знать об этом, решая стоит ли доверять ему свой проект.

Сообщество

Есть ли активное сообщество разработчиков, поддерживающих этот проект? В случае если вам понадобится помощь - будет ли она доступна?

Документация

Код должен быть хорошо прокомментирован, но в идеале, вики или другая официальная документация для поддержки библиотеки имеет важное значение.

На момент подготовки настоящего документа фреймворки, перечисленные в Таблице 18-3, удовлетворяли всем или большинству из приведенных выше критериев. Если вы не видите здесь библиотеку для вашего предпочтительного языка программирования, она может быть где-то там, но просто не вошедшей в наш список.

Таблица 18-3. Фреймворки разработки AGI

Фреймворк	Язык
Adhearsion	Ruby
Asterisk-Java	Java
AsterNET	.NET
ding-dong	Node.js
PAGI	PHP
Panoramisk	Python
StarPy	Python + Twisted

Вывод

AGI предоставляет мощный интерфейс для Asterisk, позволяющий реализовать управление вызовами от первого лица на выбранном вами языке программирования. Вы можете использовать несколько подходов к реализации приложения AGI. Некоторые подходы могут обеспечить лучшую производительность, но ценой большей сложности. AGI предоставляет среду программирования, которая может облегчить интеграцию Asterisk с другими системами или просто обеспечить более удобную среду программирования управления вызовами для опытного программиста. Во многих случаях наилучшим подходом будет использование предварительно созданной структуры, особенно при оценке или прототипировании сложного проекта. Для максимальной производительности мы по-прежнему рекомендуем вам рассмотреть возможность написания как можно большего объема вашего приложения с помощью диалплана Asterisk.

Люди, которые думаю, будто все знают, раздражают нас, людей, которые действительно все знают.
– Айзек Азимов

Интерфейс Asterisk REST (ARI) был создан для устранения ограничений, присущих разработке внешних или расширенных функций вне Asterisk. В то время как AGI позволяет запускать внешние приложения, а AMI позволяет осуществлять наблюдение и контроль выполняемых вызовов, любая попытка интегрировать их в полное внешнее приложение быстро становится сложной и запутанной. ARI позволяет разработчикам создавать автономное и полное приложение, используя Asterisk в качестве базового движка.

На момент написания этой статьи ARI требует очень простого диалплана для запуска приложения `Stasis()`, которое затем передает канал в ARI. К тому времени, когда вы читаете это, очень вероятно, что это требование изменилось, поскольку сообщество разработчиков Asterisk активно работает над тем, чтобы позволить ARI появляться без какого-либо диалплана в середине.

Использование внешнего интерфейса - такого как ARI, для управления Asterisk, не обязательно облегчит вашу жизнь. Навыки, необходимые для реализации и устранения неполадок приложений этого типа, требуют комплексного набора навыков не только на выбранном языке, но и в области системного администрирования Linux, администрирования Asterisk, устранения неполадок в сети и основных концепций телефонии. Для опытного разработчика ARI может дать необходимую мощь в приложениях, но для тех, кто учится, мы рекомендуем изучить диалплан, прежде чем погружаться во внешние среды разработки. Диалплан является своеобразным, но он также полностью интегрирован, высокопроизводителен и относительно прост в освоении.

Сказав это - давайте разберемся с ARI.

Быстрый запуск ARI

В этом разделе приведен простой рабочий пример ARI. Позже в этой главе мы рассмотрим все более подробно.[1]

В этом разделе быстрого запуска мы будем использовать очень простой уровень доступа HTTP. Вы должны быть очень осторожны при вводе такого рода конфигурации в продакшен. Если, например, вы собираетесь запустить приложение на отдельном компьютере и подключить его к Asterisk через сокет, то потребуется более безопасное соединение. То, что мы делаем в этом разделе, сродни парусному клубу, использующему шлюпки для обучения; полезно в качестве начинания, но глупо и опасно выходить в море на таком судне.

Базовая конфигурация Asterisk

У вас уже должен быть запущен веб-сервер Asterisk - поэтому вам просто нужно проверить, что ваш файл */etc/asterisk/http.conf* выглядит следующим образом:

```
[general]
enabled = yes
bindaddr = 127.0.0.1
```

1 Хотя, честно говоря, в конфигурации нет особой сложности, если вы решите внедрить одну из платформ, что настоятельно рекомендуется для продакшена и которую мы рассмотрим позже.

Далее нужен простой файл */etc/asterisk/ari.conf*:

```
[general]
enabled = yes
pretty = yes
[asterisk]
type = user
read_only = no
password = чтобывыниделалинеиспользуйтеэтотпароль
```

Хорошо, давайте загрузим модуль `ari` сейчас:

```
$ sudo asterisk -rx 'module load res_ari.so'
Loaded res_ari.so => (Asterisk RESTful Interface)
```

Затем в файл */etc/asterisk/extensions.conf* необходимо добавить расширение для запуска приложения диалплана `Stasis()`:[2]

```
exten => 242,1,Noop()
  same => n,Stasis(zarniwoop)
  same => n,Hangup()
```

Перезагрузите ваш диалплан с помощью:

```
$ sudo asterisk -rx 'dialplan reload'
Dialplan reloaded.
```

На этом этапе может быть стоит просто перезагрузить Asterisk:

```
$ sudo service asterisk restart
```

Осталось всего несколько шагов, и вы готовы протестировать свою среду ARI.

Тестирование вашей среды ARI

Поскольку ARI зависит от WebSockets - нам понадобится инструмент, позволяющий тестировать из командной строки. Node.JS package manager (npm) позволит нам найти и установить `wscat`, который мы будем использовать для наших тестов.

```
$ sudo yum -y install npm

$ sudo npm install -g wscat

/usr/bin/wscat -> /usr/lib/node_modules/wscat/bin/wscat
/usr/lib
+-- wscat@2.2.1
  +-- commander@2.15.1
  +-- read@1.0.7
  ¦ +-- mute-stream@0.0.8
  +-- ws@5.2.2
    +-- async-limiter@1.0.0
```

Теперь давайте зажжем его и посмотрим, что получим!

```
$ wscat -c "ws://localhost:8088/ari/events?api_key= \
 asterisk:чтобывыниделалинеиспользуйтеэтотпароль&app=zarniwoop"

connected (press CTRL+C to quit)
>
```

Пока все идет хорошо. Давайте сделаем звонок в наше приложение `Stasis()` и посмотрим что произойдет.

Откройте новое окно SSH (оставьте предыдущее как есть чтобы видеть что происходит в сеансе `wscat`). Подключитесь к CLI Asterisk в этом новом сеансе оболочки:

```
$ sudo asterisk -rvvvv
```

Используя один из ваших лабораторных телефонов, позвоните на номер 242.

В Asterisk CLI, вы должны увидеть это:

2 Мы назвали приложение "zarniwoop", потому что “hello-world” использовалось в Digium wiki для ARI, и нам показалось что лучше избегать перекрытия. Вы, конечно, можете назвать его как угодно.

```
*CLI>
 == Setting global variable 'SIPDOMAIN' to '172.29.1.57'
 -- Executing [242@sets:1] NoOp("PJSIP/SOFTPHONE_A-00000001", "") in new stack
 -- Executing [242@sets:2] Stasis("PJSIP/SOFTPHONE_A-00000001", "zarniwoop") in new stack
```

И в сеансе `wscat` вы должны увидеть это:

```
>
< {
  "type": "StasisStart",
  "timestamp": "2019-01-27T21:43:43.720-0500",
  "args": [],
  "channel": {
    "id": "1548643423.2",
    "name": "PJSIP/SOFTPHONE_A-00000002",
    "state": "Ring",
    "caller": {
      "name": "101",
      "number": "SOFTPHONE_A"
    },
    "connected": {
      "name": "",
      "number": ""
    },
    "accountcode": "",
    "dialplan": {
      "context": "sets",
      "exten": "242",
      "priority": 2
    },
    "creationtime": "2019-01-27T21:43:43.709-0500",
    "language": "en"
    },
  "asterisk_id": "08:00:27:27:bf:0e",
  "application": "zarniwoop"
}
>
```

Хорошо, теперь мы откроем еще один сеанс оболочки[3], чтобы взаимодействовать с этим соединением, которое мы создали. Из этой новой оболочки выполните следующую команду:

```
$ curl -v -u asterisk:чтобывыниделалинеиспользуйтеэтотпароль -X POST \
 "http://localhost:8088/ari/channels/1548643423.2/play?media=sound:believe-its-free" sd
```

Обратите внимание, что "`id` "из JSON, возвращаемый в сеансе `wscat`, должен использоваться после части `'channels/'` команды `curl`. Другими словами, вы должны сопоставить идентификатор канала в вашей команде с идентификатором канала, связанным с вашим вызовом. Таким образом, вы можете одновременно обрабатывать множество звонков.

Работа со средой ARI с использованием Swagger

Asterisk ARI был разработан чтобы быть совместимым со спецификацией OpenAPI (aka Swagger) и это означает, что многие инструменты, совместимые с этой спецификацией, будут работать с ARI. Например, вы можете взаимодействовать с установкой AIR с помощью Swagger-UI, который будет полезен как для отладки, так и в качестве источника документации.

Во-первых, нам нужно будет открыть наш HTTP-сервер Asterisk в локальной сети (в настоящее время он разрешает только соединения с 127.0.0.1). В вашем файле */etc/asterisk/http.conf* мы привяжем HTTP-сервер к локальному IP-адресу машины "Asterisk":

```
$ sudo vim /etc/asterisk/http.conf
; Включить встроенный HTTP-сервер и прослушивать только соединения на локальном хосте.
[general]
enabled = yes
;bindaddr = 127.0.0.1 ; закомментируйте это
bindaddr = 172.29.1.57 ; LAN IP ВАШЕГО СЕРВЕРА ASTERISK
```

3 Если ваш компьютер имеет только один экран, то, вероятно, это то место, где вы задумаетесь, что неплохо было бы иметь их больше.

Далее нам нужно добавить строку в ваш файл */etc/asterisk/ari.conf*:

```
$ sudo vim /etc/asterisk/ari.conf
[general]
enabled = yes
pretty = yes
allowed_origins=http://ari.asterisk.org
...
```

Сохраните и перезагрузите модули `http` и `ari` в Asterisk:

```
$ sudo asterisk -rx 'module reload http' ; sudo asterisk -rx 'module reload ari'
```

Теперь на рабочем столе разработчика откройте браузер и перейдите в раздел http://ari.asterisk.org.

Вы увидите страницу, похожую на Рисунок 19-1.

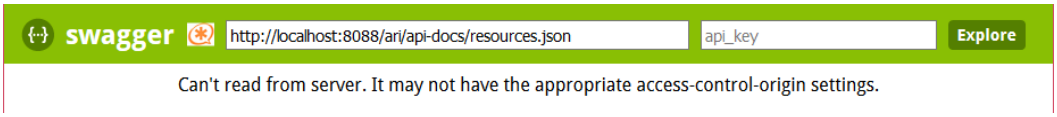

Рисунок 19-1. Swagger UI для ARI

Замените `localhost` на LAN IP-адрес вашего сервера Asterisk, а в поле api_key введите Ваш ARI *`user:password`* из */etc/asterisk/ari.conf* (например, `asterisk:чтобывыниделалинеиспользуйтеэтотпароль`). Если у вас все настройки верны, то вы будете вознаграждены с результатами как на Рисунке 19-2.

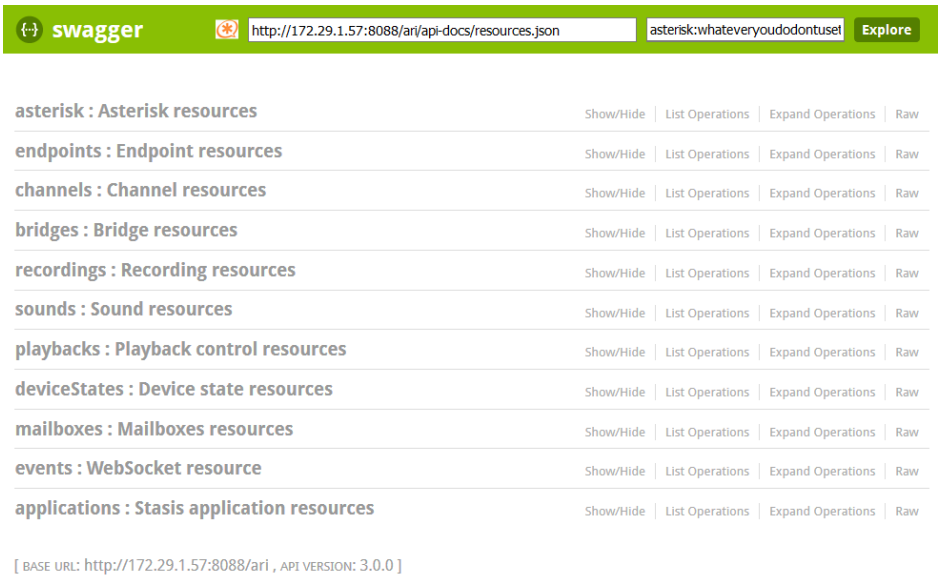

Рисунок 19-2. ARI Swagger

Вы видите полную документацию для вашего модуля ARI, и можете на самом деле так же передавать к нему запросы. Это очень полезно при отладке, и хвала людям Digium за это.

В качестве примера того, для чего это нужно - выберите пункт `endpoints:Endpoint resources`, нажмите кнопку `GET` рядом с `/endpoints`, и вы увидите экран, показанный на Рисунке 19-3.

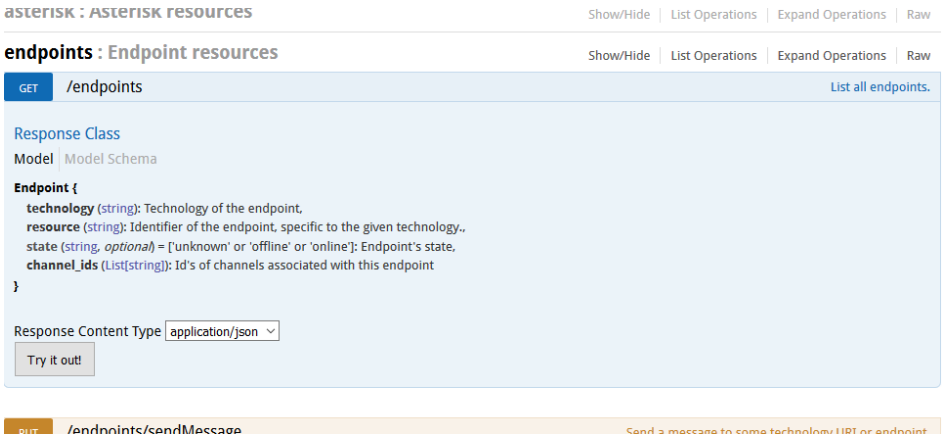

Рисунок 19-3. Получение конечных точек

Ну, давай - жми на кнопку "Try it out!".

Обратите внимание на "`id`" канала в сеансе `wscat`, который вы хотите скопировать для использования в Swagger UI (вы увидите несколько строк вывода JSON, связанных с вызовом).

Выполните следующие действия по каналу через интерфейс Swagger UI: `POST: Answer` (ответ на канал), `POST: hold` (поставить вызов на удержание), `DELETE: hold` (принять вызов из режима ожидания). Обратите внимание на то, что происходит с каналом в каждом случае.

Использование этого Swagger UI также документировано в Asterisk wiki.

Это значительно упростит процесс разработки и тестирования.

Хорошо, это быстрый старт. Давайте нырнем поглубже в ARI.

Строительные блоки ARI

Есть три компонента, которые работают вместе для обеспечения ARI:

- RESTful интерфейс, через который внешнее приложение взаимодействует с Asterisk.
- WebSocket, который передает информацию обратно во внешнее приложение из Asterisk (в формате JSON).
- Приложение диалплана `Stasis()`, которое соединяет управление каналом с внешним приложением.

REST

Термин *RESTful* происходит от Representational State Transfer (REST), который является архитектурной моделью для веб-служб (в отличие, скажем, от протокола). Термин RESTful обычно относится к любому API, который обеспечивает взаимодействие через URL-адреса с данными, представленными в формате JSON.[4] Итак, все, что является "RESTful", должно соответствовать ограничениям REST, но на практике может быть реализовано как более свободная интерпретация (которая, если выполнит свою работу, действительно может быть достаточно хорошей).

WebSocket

Соединение WebSocket - это механизм, осуществляющий связь между внутренними компонентами Asterisk и интерфейсом RESTful. В Asterisk могут происходить события, которые клиент не инициировал, и WebSocket позволяет Asterisk сигнализировать об этих изменениях клиенту.

4 Строго говоря, REST - это гораздо больше, но на практике в наши дни не редкость предположить, что REST API будет основан на URL и JSON просто потому, что много сервисов представлены именно в этих форматах.

Встроенный HTTP-сервер Asterisk потенциально предоставляет другие сервисы через веб-интерфейс. Например, WebRTC также подключается через веб-сервер. Если вы вносите изменения или добавляете новые службы - убедитесь что проверили не только элемент, над которым работаете, но и другие службы, работающие через тот же сервер, чтобы убедиться что случайно не расстроили что-то другое.

Stasis

Шина сообщений Stasis позволяет ядру Asterisk связывать события с другими модулями и компонентами. Она как правило является внутренней для Asterisk; тем не менее, в случае ARI приложение диалплана с именем `Stasis()` позволяет диалплану передать управление вызовами внешнему приложению ARI.

Само приложение `Stasis()` требуется для того, чтобы сигнализировать диалплану, что управление вызовами должно быть передано внешней программе через ARI.

> Начиная с Asterisk 16, больше нет необходимости писать код диалплана для определения соединения от входящего канала к клиентскому приложению ARI. Многие разработчики в сообществе Asterisk пишут всю свою логику управления вызовами во внешних приложениях, и необходимость писать несколько строк диалплана только для передачи каналов в свое приложение считалась сложной и запутанной. Они запросили (и разработали) механизм, с помощью которого Asterisk автоматически создает диалплан для обработки этой функции.
>
> При создании экземпляра API, ссылка на приложение в URL-адресе — например, наше приложение `zarniwoop` - инициирует автоматическое создание контекста диалплана, названного в соответствии с именем приложения (в данном случае `[stasis-zarniwoop]`), включая расширение, шаблон которого соответствует всему. Затем это расширение будет передавать все вызовы, поступающие в этот контекст в `Stasis(zarniwoop)`. Вам нужно будет связать ваши каналы с правильным контекстом (`context=stasis-zarniwoop`) в ваших таблицах конфигурации PJSIP (или другого канала), и в момент вызова этих каналов они будут автоматически подключены через `Stasis()` к клиентскому приложению.
>
> Если все это кажется запутанным - нет причин, по которым вам нужно прекратить использовать фактический диалплан для обработки этого, как мы делали ранее в нашем примере быстрого запуска.

Понимание работы `Stasis()` обычно не требуется, если вы не собираетесь разрабатывать сам продукт Asterisk (т.е. присоединиться к команде разработчиков Asterisk и создавать новые возможности в Asterisk).

Как правило, после первых экспериментов с ARI вы захотите реализовать фреймворк для облегчения работ по разработке внешнего приложения.

Фреймворки

Производственное приложение, использующее ARI, выиграет от внедрения платформы для упрощения разработки, добавления уровня безопасности и обеспечения среды управления.

Существует несколько таких библиотек. Какую из них вы выберете - отчасти будет продиктовано тем, какой язык вы предпочитаете использовать, а также должны учитывать, имеет ли структура, в которой вы заинтересованы, активное сообщество и наличие активной поддержки.

Те, которые описаны ниже, перечислены в Asterisk wiki. Мы рассмотрели репозиторий кода для каждого, и хотя некоторые проекты все еще активно поддерживаются - другие не обновлялись в течение довольно продолжительного времени. Если вы планируете реализовать один из этих

фреймворков - вам нужно будет сделать собственную экспертизу, чтобы убедиться что вы можете получить поддержку для него. Во многих случаях, возможно, стоит обратиться к разработчикам и определить их ставки за консультации, чтобы вы могли обеспечить приоритетный доступ к их времени, если вам это понадобится.

ari-py (и aioari) для Python

Фреймворк ari-py был написан Digium в 2013-2014 годах и с тех пор не обновлялся. Эта структура основана на клиенте Asterisk Swagger.py.

Вскоре после выпуска ari-py он был разветвлен в проект aioari, который предоставляет асинхронную версию ari-py. С тех пор этот код постоянно обновляется (хотя на момент написания этой статьи он не обновлялся с начала 2018 года). Эта структура должна быть включена в вашу оценку структуры Python для ARI.

Если вы хотите разрабатывать приложения ARI на Python - одна из этих двух платформ может быть тем, что вы ищете. Если хотите создать большое приложение ARI - вам необходимо убедиться, что вы тщательно проверили влияние производительности использования Python на то, что делаете.

Digium предоставил образцы для этой структуры (и других) на *https://github.com/asterisk/ari-examples*.

node-ari-client

Для любителей JavaScript есть основанный на Node.js ARI-фреймворк, который был впервые выпущен в начале 2014 года и на момент написания этой статьи все еще обновляется. Он основан на автоматически генерируемом API, который приходит из swagger-js.

Для разработчиков JavaScript/Node вы можете начать: *https://github.com/asterisk/node-ari-client*.

Digium предоставил образцы для этой структуры (и других) на *https://github.com/asterisk/ari-examples*.

AsterNET.ARI

Сторонники Windows так же не остались в стороне. Проект AsterNET.ARI предоставляет платформу для .NET, которая дополняет проект Masternet (который также включает интеграцию с интерфейсами Asterisk FastAGI и AMI).

Вы можете найти репозиторий для AsterNET.ARI здесь: *https://github.com/skrusty/AsterNET.ARI*.

Digium предоставил образцы для этой структуры (и других) на *https://github.com/asterisk/ari-examples*.

ari4java

Проект ari4java является одним из наиболее активно разрабатываемых фреймворков ARI, которые мы нашли. Он разрабатывается с 2013 года и репозиторий получал коммиты одновременно с этой записью.

Если Java - ваш язык, вы точно захотите проверить репозиторий ari4java по адресу *https://github.com/l3nz/ari4java*.

phpari

Проект phpari предоставляет платформу ARI для сообщества PHP. Он разрабатывается с 2014 года и на момент написания этой статьи репозиторий все еще обновлялся.

Для поклонников PHP вы найдете репозиторий по адресу *https://github.com/greenfieldtech-nirs/phpari*.

aricpp

Если вы привыкли писать на C++ - есть даже проект ARI для вас. Платформа aricpp состоит только из файлов заголовков, так что вы можете встроить свои функции прямо в то, что разрабатываете. Эта библиотека также была протестирована на производительность с помощью SIPp, и хотя у нас нет цифр по этому поводу, нам кажется что скомпилированная среда, протестированная на производительность, очень даже стоит того, чтобы её попробовать, если у вас есть соответствующие навыки.

Один из самых новых фреймворков ARI, этот проект получает регулярные обновления. Проверьте его *https://github.com/daniele77/aricpp*.

asterisk-ari-client

Да, Ruby тоже имеет фреймворк ARI.

Вы можете найти его по адресу *https://github.com/svoboda-jan/asterisk-ari*.

Вывод

ARI предоставляет RESTful API текущего поколения, который может использоваться для разработки коммуникационных приложений с использованием популярных языков разработки. С его помощью опытный разработчик может использовать мощь самой успешной платформы АТС в истории. Это позволяет коммуникационным приложениям следующего поколения взаимодействовать с устаревшими телекоммуникационными протоколами и приложениями, что может оказаться очень полезным, поскольку мы все чаще призваны преодолевать разрыв между прошлым, настоящим и будущим коммуникационных технологий.

Глава 20. WebRTC

Web, каким я его себе представлял, мы еще не видели. В будущем все еще намного больше, чем прошлом.
– Тим Бернерс-Ли

Браузер как телефон

В интернет-коммуникации назревает новая революция и, хотя она вряд ли наделает столько шума как революция телекоммуникаций с открытым исходным кодом, она определенно имеет потенциал чтобы тихо заменить сердце каждого текущего коммуникационного приложения.

Сегодня интернет предлагает множество приложений для конференций с закрытым исходным кодом. Все они делают примерно одно и то же, и все же большинство из них требуют установки проприетарного программного обеспечения прежде, чем вы сможете их использовать (что, конечно, поможет сохранить загрузку в памяти вашего компьютера). Каждое из них не отличается от последнего приложения конференц-связи, которое вы были вынуждены установить (для некоторых других встреч, на которых вы присутствовали). Каждая из этих компаний надеется, что она станет выше других чтобы доминировать в пространстве. Между тем, WebRTC спокойно создает стандарт, который принудительно устраняет все концепции проприетарных мультимеа-коммуникаций, которые, как мы надеемся, устранят некоторых из этих узколобых, огороженных стеной мышления, и откроют коммуникации для некоторых фактических инноваций.

С тех самых пор, как существуют веб-браузеры, предпринимались попытки интегрировать мультимедиа в интернет. Это оказалось сложнее, чем ожидалось, так что сегодня телефон по-прежнему является отдельным приложением (или, конечно, отдельным устройством в целом).

WebRTC обещает изменить все это.

В этой главе мы познакомим вас с интерпретацией WebRTC, предложенной Asterisk. Ни в коем случае не следует считать это всеобъемлющим введением; все, на что у нас будет время — это провести вас через создание стандартного приложения для видеоконференций, которое, по сути, является приложением “Hello World”, которое все используют для начала работы с WebRTC. Это отличный способ пнуть шины, но важно понимать, что WebRTC будет намного больше.

Предварительное знание

Прежде чем погрузиться в WebRTC - есть некоторые базовые технологии, которые должны объединиться.

Прежде всего: если вы серьезно относитесь к WebRTC — вам понадобится доступ к веб-разработчику, а в идеале к кому-то, кто имеет глубокие знания различных языков, протоколов и технологий, которые делают интернет работающим. WebRTC — это веб-разработка и это технология bleeding-edge, и вы столкнетесь с несовместимостью, проблемами браузера, нераскрытыми ошибками, неполной документацией и другими проблемами, присущими новой технологии. Если вы не являетесь full-stack разработчиком с уверенными сетевыми и Linux навыками - у вас будет очень крутая кривая обучения с WebRTC!

Вероятно, Цахи Левент-Леви сказал по этому поводу лучше всего:

WebRTC - это технология, которая является частью VoIP и частью Web. ... Для того, чтобы действительно быть профессиональным разработчиком WebRTC, вы должны быть в состоянии понять две очень разные технические области:

1. *Вы должны знать, как работает VoIP. Как движется медиапоток по сети в режиме реального времени (такие вещи, как RTP, RTCP, Jitter Buffer и множество других сокращений).*
2. *Вам нужно знать и понимать, как разрабатывать для web — frontend и backend (любой full-stack разработчик?). JavaScript - это данность. Бонусные очки за node.js.*

Итак, да, вы должны быть full-stack разработчиком плюс гуру VoIP если хотите комфортно погрузиться в WebRTC. Мы говорим это не для того чтобы отпугнуть вас, а чтобы заверить, что если вы находите это сложным, то это не связано с каким-либо недостатком с вашей стороны, а просто потому, что это сложный, многослойный материал.

Сказав все это - можно получить вкус WebRTC без всего этого и в этой главе мы собираемся настроить Asterisk для поддержки WebRTC и запустить предварительно построенное веб-приложение, которое продемонстрирует основные аудио/видео возможности реализации Asterisk WebRTC. У вас все еще будет эта крутая кривая обучения, но, надеюсь, мы создали фундамент, на котором можно строить.

Настройка Asterisk для WebRTC

Для передачи вызовов через Asterisk с помощью WebRTC необходимо использовать драйвер канала PJSIP. Конфигурация будет аналогична конфигурации стандартных SIP-телефонов, но не идентична.

Для этого нам понадобится тип транспорта, который мы добавим в файл */etc/asterisk/pjsip.conf*:

```
[transport-udp]
type=transport
protocol=udp
bind=0.0.0.0

[transport-tls]
type=transport
protocol=tls
bind=0.0.0.0
cert_file=/home/asterisk/certs/self-signed.crt
priv_key_file=/home/asterisk/certs/self-signed.key
```

Это все для редактирования конфигурационного файла. Для остальных изменений PJSIP мы будем использовать базу данных.[1]

Мы создадим двух новых подписчиков с именами `WS_PHONE_A` и `WS_PHONE_B`. Клиент WebRTC будет использовать учетные данные для этих конечных точек для связи с драйвером канала PJSIP в Asterisk (т.е. для совершения телефонных звонков).

В таблицу `ps_aors` необходимо добавить две записи:

```
INSERT into asterisk.ps_aors
(id, max_contacts)
values ('WS_PHONE_A', 5),
       ('WS_PHONE_B', 5)
;
```

Необходимы соответствующие записи `ps_auth`:

1 Обратите внимание, что вы можете настроить драйвер канала PJSIP полностью с помощью файла конфигурации, но в этой книге мы делаем это только там, где это необходимо и, в противном случае, используем базу данных для конфигурации канала PJSIP

```
INSERT into asterisk.ps_auths
(id, auth_type, password, username)
values ('WS_PHONE_A','userpass','spiderwrench','WS_PHONE_A'),
       ('WS_PHONE_B','userpass','arachnoratchet','WS_PHONE_B')
;
```

Затем мы создадим сами конечные точки:

```
INSERT INTO asterisk.ps_endpoints
 (id,aors,auth,context,
 transport,dtls_auto_generate_cert,webrtc,disallow,allow)
VALUES
 ('WS_PHONE_A','WS_PHONE_A','WS_PHONE_A','sets',
  'transporttls','yes','yes','all','vp8,opus,ulaw'),
 ('WS_PHONE_B','WS_PHONE_B','WS_PHONE_B','sets',
  'transport-tls','yes','yes','all','vp8,opus,ulaw');
```

В Главе 4 мы уже создали наши сертификаты - поэтому должны иметь возможность использовать их здесь.

```
$ ls -l /home/asterisk/certs/
```

Мы должны позаботиться о конфигурации канала для нашего примера WebRTC.

Теперь нам нужно настроить веб-сервер Asterisk для обработки HTTPS.

```
$ sudo vim /etc/asterisk/http.conf

[general]
enabled=yes
bindaddr=0.0.0.0
bindport=8088
tlsenable=yes
tlsbindaddr=0.0.0.0:8089
tlscertfile=/home/asterisk/certs/self-signed.crt
tlsprivatekey=/home/asterisk/certs/self-signed.key
```

Сохранимся и перезапустим Asterisk.

```
$ sudo service asterisk restart
```

Убедитесь, что Asterisk теперь работает не только на HTTP-сервере, но и на HTTPS:

```
*CLI> http show status
HTTP Server Status:
Server Enabled and Bound to 0.0.0.0:8088
HTTPS Server Enabled and Bound to 0.0.0.0:8089

Enabled URI's:
/ws => Asterisk HTTP WebSocket
```

Ищите в выходных данных `HTTPS`, чтобы проверить работоспособность сертификатов и также должны увидеть `/ws` - поскольку это указывает на загрузку компонентов WebSockets.

Подсказка: если он не работает - всегда проверяйте */var/log/messages* для любых сообщений SELinux.

```
$ sudo grep sealert /var/log/messages
```

Брандмауэр в настоящее время не настроен для этих портов - поэтому нам нужно добавить несколько правил для обработки:

```
$ sudo firewall-cmd --zone=public --add-port=8088/tcp
$ sudo firewall-cmd --zone=public --add-port=8088/tcp --permanent
$ sudo firewall-cmd --zone=public --add-port=8089/tcp
$ sudo firewall-cmd --zone=public --add-port=8089/tcp --permanent
$ sudo firewall-cmd --zone=public --add-port=5061/udp
$ sudo firewall-cmd --zone=public --add-port=5061/udp --permanent
```

На этом этапе вам нужно запустить веб-браузер и установить соединение. Ваш браузер будет жаловаться на соединение, если вы используете самоподписанный сертификат, но он позволит вам осуществить соединение. Это критический шаг, так как вы должны указать вашему браузеру хранить

сертификат постоянно, так что WebRTC может использовать соединение WebSocket. Следующий URL-адрес соединит вас:

```
https://ip-of-asterisk-server:8089/ws
```

Если вы получите сообщение `Upgrade Required` (о необходимости обновления) - *это хорошо*. Это означает, что соединение хорошее и это просто протокол, говорящий вам, что для этого недостаточно технологии, чтобы это было фактическое соединение WebSocket. Мы там, где должны быть.

Конечно, следующая вещь - фактически испытать сеанс WebRTC через ту среду, которую мы настроили; и для того, чтобы проверить все это - вам нужно будет запустить браузер и загрузить в нем какой-нибудь клиент WebRTC. Следующий раздел будет делать именно это.

Cyber Mega Phone

Для того, чтобы увидеть WebRTC в действии на вашей системе Asterisk - вам нужно что-то, что работает в вашем браузере. Самый простой способ увидеть это в действии — взять *Cyber Mega Phone* от Digium. Он позволит вам быстро настроить рабочий сеанс WebRTC с помощью Asterisk.

Во-первых, так как WebRTC требует использования TLS (что не является обязательным, как в случае с SIP), мы собираемся заставить вас еще раз проверить установку ваших сертификатов. Если вы еще не сделали этого - сейчас самое время поработать над Главой 4, или использовать скрипт, предоставляемый как часть исходного кода Asterisk, генерирующий ключи и сертификаты (вы найдете его в исходниках Asterisk в каталоге */home/astmin/src/asterisk-16.<TAB>/contrib/scripts/*. Скрипт называется *ast_tls_cert* и он задокументирован в wiki Asterisk.

Хорошо, теперь нам нужен диалаплан для приема наших звонков WebRTC:

```
$ vim /etc/asterisk/extensions.conf

exten => 246,1,Noop()
  same => n,Answer()
  same => n,Wait(0.5)
  same => n,StreamEcho(4)
  same => n,Hangup()
```

Сам Cyber Mega Phone находится на GitHub под аккаунтом Asterisk.

Вы можете загрузить код и запустить его с локального компьютера или же можете загрузить его на веб-сервер и пользоваться им оттуда.

Давайте воспользуемся им с нашего сервера Asterisk:

```
$ cd /var/lib/asterisk/static-http

$ sudo git clone https://github.com/asterisk/cyber_mega_phone_2k.git

$ sudo chown -R asterisk:asterisk cyber_mega_phone_2k ; sudo chmod 755 cyber_mega_phone_2k
```

Нам понадобится небольшое изменение в конфигурации HTTP-сервера Asterisk, чтобы он мог обслуживать статический контент.

```
$ sudo vim /etc/asterisk/http.conf

[general]
enabled=yes
bindaddr=0.0.0.0
bindport=8088
tlsenable=yes
tlsbindaddr=0.0.0.0:8089
tlscertfile=/home/asterisk/certs/asterisk.crt
tlsprivatekey=/home/asterisk/certs/asterisk.key
enablestatic=yes
redirect=/cmp2k /static/cyber_mega_phone_2k/index.html
```

Сохраните и перезагрузите http-модуль из консоли Asterisk:

```
*CLI> module reload http
```

Теперь с помощью браузера можно перейти к новому клиентскому приложению WebRTC:

```
https://your asterisk server:8089/cmp2k
```

Если все пойдет по плану, вы увидите что-то вроде Рисунка 20-1.

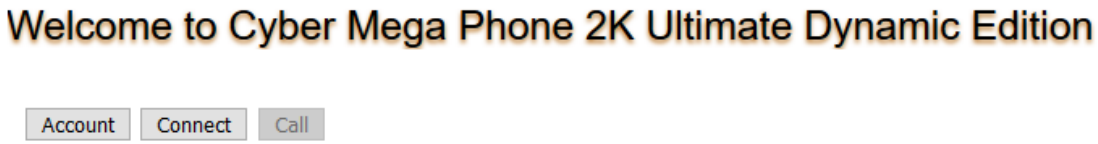

Рисунок 20-1. Cyber Mega Phone 2K

Нажмите кнопку Account и введите учетные данные пользователя WebRTC (см. Рисунок 20-2).

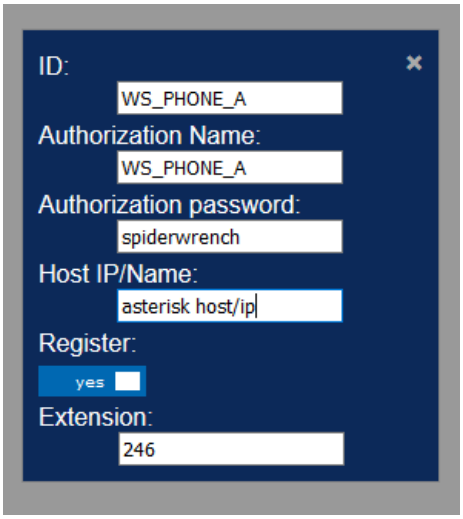

Рисунок 20-2. Данные учетной записи WebRTC

После ввода сведений, относящихся к системе, нажмите X для сохранения и закрытия.

Теперь вы можете нажать кнопку Connect и, если все прошло хорошо, ваш клиент WebRTC должен зарегистрироваться в Asterisk (это хорошее время для мониторинга консоли Asterisk, чтобы увидеть, что происходит и есть ли какие-либо ошибки).

Если вы нажмете кнопку Call сейчас, то должны подключиться через WebRTC и увидеть два окна (Рисунок 20-3). Одно из них — ваше локальное видео, а другое — отображает дальний конец (т.е. оно имитирует другого пользователя, повторяя то, что отправили Вы). Если Ваше аудио также работает, то вы даже можете получить некоторый шум обратной связи!

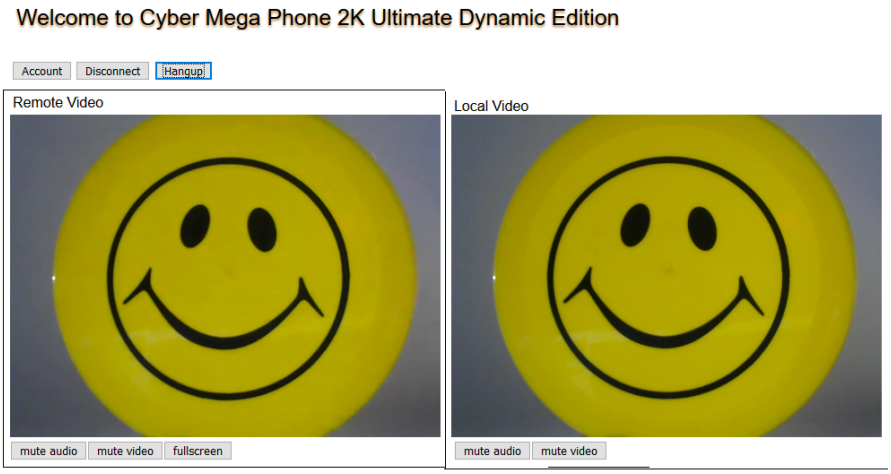

Рисунок 20-3. Приложение Echo с видео

Вы видите что есть окно удаленного видео рядом с окном локального видео. Возможно, мы не достигли многого, чтобы хвастаться, но ваша система Asterisk обрабатывает WebRTC - поэтому улыбнитесь и сделайте перерыв. Вы это заслужили.

Подробнее об WebRTC

Экосистема WebRTC быстро развивается и то, что верно на момент написания этой статьи - может быть неверным в ближайшем будущем. Мы нашли следующие ресурсы, которые могут быть очень полезными:

- Цахи Левант-Леви участвует во многих различных инициативах WebRTC, и он щедро делится знаниями о том, как познать WebRTC. Проверьте его сайт bloggeek.me. Подпишитесь на него.
- Группа людей под руководством Kranky Geek выпустила несколько конференций об WebRTC и поделилась многими полезными видео на YouTube. На YouTube-канале Kranky Geek вы найдете их.
- Ознакомьтесь с различными протоколами сигнализации, которые популярны в WebRTC: SIP, VIRTO (из проекта FreeSwitch), XMPP и даже JSON.
- Посмотрите различные библиотеки сигнализации WebRTC. В настоящее время к популярным относятся: sipML5 (возможно, самая первая библиотека WebRTC) и JsSIP (плюс форк JsSIP с именем *SIP.js*).
- webrtc.org является официальным домом WebRTC и, безусловно, заслуживает некоторого внимания. Проверьте домашнюю страницу перед началом работы.
- Онлайн-платформа обучения o'Рейли имеет несколько видео, которые стоит посмотреть. Для любых книг и видео следите за датой публикации, так как все, что старше года или двух, скорее всего, устарело — WebRTC все еще находится в стадии быстрого развития.

Нам еще так много предстоит узнать, но у нас закончились страницы.

Вывод

WebRTC является захватывающим и важным и, вполне вероятно, что разработчики и интеграторы VoIP должны быть знакомы с этой технологией если хотят сохранить свои навыки актуальными. На момент написания этой статьи WebRTC все еще находится в стадии разработки. Как и в любом исследовании новых границ, те, кто прокладывает путь, должны быть творческими, настойчивыми, оптимистичными и жесткими.

Asterisk может быть полезным компонентом в будущей VoIP-среде, служа, по крайней мере, мостом между продуктами WebRTC следующего поколения и телекоммуникациями старой школы.

Глава 21. Системный мониторинг и журналирование

Хаос присущ всем сложным вещам. Стремитесь дальше с усердием.
– Будда

Asterisk поставляется с несколькими подсистемами, позволяющими получить подробную информацию о работе вашей системы. Как для устранения неполадок, так и для использования в целях биллинга или задач персонала, различные модули мониторинга Asterisk могут помочь вам следить за внутренней работой вашей системы.

logger.conf

При устранении неполадок в системе Asterisk вам будет очень полезно обратиться к некоторым историческим записям того, что происходило в системе в то время, когда произошла указанная проблема. Параметры для хранения этой информации определены в файле */etc/asterisk/logger.conf*.

В идеале, вы можете захотеть чтобы система хранила запись каждой вещи что она делает. Однако у этого есть свои издержки. На занятой системе с включенным полным журналированием отладки будет создаваться большой объем данных. Хотя сегодня хранение данных намного дешевле, чем было в молодости Asterisk, возможно, все еще необходимо достичь баланса между детализацией и требованиями к хранению.

Файл */etc/asterisk/logger.conf* позволяет определить все виды различных уровней ведения журнала, а также несколько файлов журналов, если это необходимо. Эта гибкость превосходна, но она также может сбивать с толку.

Формат записи в файле *logger.conf* выглядит следующим образом:

```
filename => type[,type[,type[,...]]]
```

Мы уже работали с *logger.conf*, так что у вас уже будут записи в нем, похожие на следующие:

```
[general]
exec_after_rotate=gzip -9 ${filename}.2;

[logfiles]
;debug => debug
;console => notice,warning,error,verbose
console => notice,warning,error,debug
messages => notice,warning,error
full => notice,warning,error,debug,verbose,dtmf,fax
;full-json => [json]debug,verbose,notice,warning,error,dtmf,fax
;syslog keyword : This special keyword logs to syslog facility
;syslog.local0 => notice,warning,error
```

Если вы внесете какие-либо изменения в этот файл - вам нужно будет перезагрузить регистратор, выполнив следующую команду из командной оболочки:

```
$ sudo touch full messages
$ chown asterisk:asterisk /var/log/asterisk/*
$ asterisk -rx 'logger reload'
```

или из интерфейса командной строки Asterisk:

```
*CLI> logger reload
```

Детальность журналирования: полезна, но опасна

Мы боролись с тем, чтобы рекомендовать добавление следующей строки в ваш файл *logger.conf*:

```
verbose => notice,warning,error,verbose
```

Это, вполне возможно, один из самых полезных инструментов отладки, которые вы имеете при настройке и устранении неисправностей диалплана, и поэтому настоятельно рекомендуется. Опасность заключается в том, что если вы забудете отключить его когда закончите отладку - вы оставите бомбу замедленного действия в своей системе, которая медленно заполнит жесткий диск и однажды убьет вашу систему - через несколько месяцев или лет, когда вы меньше всего этого ожидаете.

Используйте его. Это просто фантастика. Просто помните, что вам нужно будет управлять своим хранилищем, чтобы гарантировать, что ваши файлы журналов не заполнят весь диск!

Вы можете указать любое имя файла - какое захотите, но специальное имя файла *console* фактически выведет выходные данные в интерфейс командной строки Asterisk, а не в любой файл на жестком диске. Все остальные имена файлов будут сохранены в файловой системе в каталоге */var/log/asterisk*. Типы *logger.conf* описаны в Таблице 21-1.

Таблица 21-1. Типы logger.conf

Тип	Описание
`notice`	Вы увидите много таких во время перезагрузки, но они также будут появляться во время обычного потока вызовов. Уведомление - это просто любое событие, о котором Asterisk хочет Вам сообщить.
`warning`	Предупреждение представляет проблему, которая может быть достаточно серьезной чтобы повлиять на вызов (включая обрыв вызова, поскольку поток вызовов не может продолжаться). Предупреждения должны быть устранены.
`error`	Ошибки представляют собой значительные проблемы в системе, которые должны быть решены немедленно.
`debug`	Отладка полезна только в том случае, если вы устраняете неполадки с самим кодом Asterisk. Вы не должны использовать отладку для устранения неполадок вашего диалплана, но можете использовать её, если разработчики Asterisk попросят вас предоставить журналы для проблемы, о которой вы сообщали. Не используйте отладку в продакшене, так как количество предоставляемых деталей может заполнить жесткий диск в течение нескольких дней.[a]
`verbose`	Это один из самых полезных типов ведения журнала, но он также является одним из наиболее рискованных, чтобы оставлять его без присмотра, из-за возможности заполнения выводом вашего жесткого диска.[b]
`dtmf`	Регистрация DTMF может быть полезна если вы получаете жалобы на то, что вызовы не маршрутизируются от автосекретаря правильно.
`fax`	Этот тип ведения журнала вызывает сообщения, связанные с факсом из серверной части технологии факса (`res_fax_spandsp` или `res_fax_digium`) для регистрации в системе факсов.
`*`	Будет регистрировать все (и мы имеем в виду все). Не используйте его, если не понимаете последствий хранения такого количества данных. Это не закончится хорошо.

[a] Это не теория. Это случилось и с нами и было это совсем невесело.

[b] Это не так рискованно, как `debug`, так как займет месяцы чтобы заполнить жесткий диск, но опасность заключается в том, что это произойдет, скажем, через год, когда вы находитесь на летних каникулах, и сразу не будет очевидно, в чем проблема. Совсем невесело.

В системе ведения журнала Asterisk существует особенность, которая вызовет у вас некоторое замешательство, если вы о ней не знаете. Уровень ведения журнала для типов `verbose` и `debug` привязаны к детализации, заданной в консоли. Это означает, что если вы ведете журнал типа `verbose` или `debug` и кто-то входит в CLI и выдает

команду `core set verbose 0`, или `core set debug 0` - регистрация этих данных в вашем лог-файле будет остановлена.

Просмотр журналов Asterisk

Поиск по журнальным файлам может быть проблемой. Хитрость заключается в том, чтобы иметь возможность фильтровать вывод, чтобы выводилась только та информация, которая имеет отношение к тому, что вы ищете.

Для начала вам нужно иметь приблизительное представление о том, когда произошла неприятность, которую вы ищете. Как только вы сориентируетесь на приблизительное время - вам нужно будет найти подсказки, которые помогут идентифицировать данный звонок. Очевидно, что чем больше информации у вас есть о вызове - тем быстрее вы сможете его зафиксировать.

В Asterisk 11 появилась функция ведения журнала, которая помогает отлаживать определенный вызов. Записи журнала, связанные с вызовом, теперь включают идентификатор вызова. Этот идентификатор вызова можно использовать с командой `grep` чтобы найти все записи журнала, связанные с этим вызовом. В следующем примере записи журнала идентификатором вызова является `C-00000004`:

```
[Dec  4 08:22:32] WARNING[14199][C-00000004]: app_voicemail.c:6286
leave_voicemail: No entry in voicemail config file for '234123452'
```

В более ранних версиях Asterisk есть еще один трюк, который вы можете использовать. Если, например, вы выполняете подробное логирование - следует отметить, что каждый отдельный вызов имеет идентификатор потока, который при использовании с командой `grep`, часто может помочь вам отфильтровать все, что не относится к вызову, который вы пытаетесь отладить. Например, в следующем подробном журнале у нас есть несколько вызовов и поскольку вызовы происходят одновременно, это может оказаться очень запутанным для отслеживания одного вызова:

```
$ tail -1000 verbose
[Mar 11 …] VERBOSE[31362] logger.c:     -- IAX2/shifteight-4 answered Zap/1-1
[Mar 11 …] VERBOSE[2973] logger.c:     -- Starting simple switch on 'Zap/1-1'
[Mar 11 …] VERBOSE[31362] logger.c:   == Spawn extension (shifteight, s, 1)
exited non-zero on 'Zap/1-1'
[Mar 11 …] VERBOSE[2973] logger.c:     -- Hungup 'Zap/1-1'
[Mar 11 …] VERBOSE[3680] logger.c:     -- Starting simple switch on 'Zap/1-1'
[Mar 11 …] VERBOSE[31362] logger.c:     -- Hungup 'Zap/1-1'
```

Чтобы отфильтровать один конкретный вызов - мы могли бы использовать команду `grep` на ID потока. Например:

```
$ grep 31362 verbose
```

дало бы нам:

```
[Mar 11 …] VERBOSE[31362] logger.c:     -- IAX2/shifteight-4 answered Zap/1-1
[Mar 11 …] VERBOSE[31362] logger.c:   == Spawn extension (shifteight, s, 1)
exited non-zero on 'Zap/1-1'
[Mar 11 …] VERBOSE[31362] logger.c:     -- Hungup 'Zap/1-1'
```

Этот метод не гарантирует что вы увидите все относящееся к одному вызову, так как вызов может породить дополнительные потоки, но для основной отладки диалплана мы находим этот подход весьма полезным, когда ID вызовов из Asterisk 11 недоступны.

Журналирование демоном Linux syslog

Linux содержит очень мощный механизм ведения журнала, которым Asterisk может воспользоваться. Хотя обсуждение всех разновидностей `syslog` и всех возможных способов ведения журнала Asterisk выходит за рамки этой книги - достаточно сказать, что если вы хотите, чтобы Asterisk отправлял журналы демону `syslog` - вам просто нужно указать следующее в вашем файле */etc/asterisk/logger.conf*:

```
syslog.local0 => notice,warning,error ; или любой тип(ы), который вы хотите логировать
```

Вам понадобится обозначение в вашем файле конфигурации *syslog*[1] с именем `local0`, которое должно выглядеть примерно так:

```
local0.*        /var/log/asterisk/syslog
```

Вы можете использовать `local0` через `local7` для этого, но проверьте свой файл *syslog.conf* чтобы убедиться, что ничто другое не использует один из этих каналов `syslog`.

Использование `syslog`[2] позволяет гораздо более мощное ведение журнала, но также требует больше знаний, чем простое логирование Asterisk в файлы. Это в основном будет полезно, если вы уже собираете другие журналы в системе на какой-то централизованный сервер `syslog`.

Проверка ведения журнала

Вы можете просмотреть состояние всех ваших параметров *logger.conf* через CLI Asterisk, выполнив команду:

```
*CLI> logger show channels
```

Вы должны увидеть выходные данные, аналогичные:

```
Channel                     Type     Status  Configuration
-------                     ----     ------  -------------
syslog.local0               Syslog   Enabled  - NOTICE WARNING ERROR VERBOSE
/var/log/asterisk/verbose   File     Enabled  - NOTICE WARNING ERROR VERBOSE
/var/log/asterisk/messages  File     Enabled  - NOTICE WARNING ERROR
                            Console  Enabled  - NOTICE WARNING ERROR DTMF=
```

Ротация лога

Существует некоторая поддержка ротации журналов, встроенная в Asterisk. Ротация логов будет производиться в следующих случаях:

- Если вы запустите команду `logger rotate` в CLI Asterisk:

  ```
  *CLI> logger route
  ```

- Во время перезагрузки конфигурации, если размер любого существующего файла журнала превышает 1 ГБ
- Если Asterisk получает сигнал `SIGXFSZ`, указывающий на то, что файл, в который он записывался, слишком велик

Call Detail Records - Записи деталей вызовов

Система CDR в Asterisk используется для регистрации истории вызовов в системе. В некоторых развертываниях эти записи используются для нужд биллинга. В других случаях - записи вызовов используются для анализа объемов вызовов с течением времени. Они также могут использоваться в качестве средства отладки администраторами Asterisk.

Содержимое CDR

CDR имеет ряд полей, которые включены по умолчанию. Таблица 21-2 перечисляет их.

Таблица 21-2. Поля CDR по умолчанию

Вариант	Значение/пример	Примечание
`accountcode`	`12345`	Идентификатор учетной записи. Это поле определяется пользователем и по умолчанию является пустым.

1 Который обычно находится в */etc/syslog.conf*

2 И `rsyslog`, `syslog-ng` и всего остального тоже.

Вариант	Значение/пример	Примечание
src	12565551212	Идентификационный номер вызывающего абонента. Он устанавливается автоматически и доступен только для чтения.
dst	102	Целевой добавочный номер для вызова. Это поле устанавливается автоматически и доступно только для чтения.
dcontext	PublicExtensions	Контекст назначения для вызова. Это поле устанавливается автоматически и доступно только для чтения.
clid	"Big Bird"	Полный идентификатор вызывающего абонента, включая имя вызывающей стороны. Это поле устанавливается автоматически и доступно только для чтения.
channel	SIP/0004F2040808-a1bc23ef	Канал вызывающей стороны. Это поле устанавливается автоматически и доступно только для чтения.
dstchannel	SIP/0004F2046969-9786b0b0	Канал вызываемой стороны. Это поле устанавливается автоматически и доступно только для чтения.
lastapp	Dial	Последнее выполненное приложение диалплана. Это поле устанавливается автоматически и доступно только для чтения.
lastdata	SIP/0004F2046969,30,tT	Аргументы, переданные на рассмотрение lastapp. Это поле устанавливается автоматически и доступно только для чтения.
start	2010-10-26 12:00:00	Время начала вызова. Это поле устанавливается автоматически и доступно только для чтения.
answer	2010-10-26 12:00:15	Время ответа на вызов. Это поле устанавливается автоматически и доступно только для чтения.
end	2010-10-26 12:03:15	Время окончания вызова. Это поле устанавливается автоматически и доступно только для чтения.
duration	195	Количество секунд, прошедших между началом и концом вызова. Это поле устанавливается автоматически и доступно только для чтения.
billsec	180	Количество секунд между ответом и концом вызова. Это поле устанавливается автоматически и доступно только для чтения.
disposition	ANSWER	Указание на то, что случилось с вызовом. Это может быть так NO ANSWER, FAILED, BUSY, ANSWER или UNKNOWN.
amaflags	DOCUMENTATION	Флаг автоматического учета сообщений (AMA), связанный с этим вызовом. Может быть одно из следующих действий: OMIT, BILLING, DOCUMENTATION или Unknown.
userfield	PerMinuteCharge:0.02	Поле пользователя общего назначения. Это поле пусто по умолчанию и может быть установлено в пользовательскую строку.[a]
uniqueid	1288112400.1	Уникальный идентификатор для объекта канала src. Это поле устанавливается автоматически и доступно только для чтения.

[a] Поле userfield сейчас это не так актуально, как раньше. Пользовательские переменные CDR - более гибкий способ получения пользовательских данных в CDRs.

Вы можете получить доступ ко всем полям записей CDR в диалплане Asterisk с помощью функции CDR(). Функция CDR() также используется для установки полей CDR, которые определены пользователем:

```
exten => 115,1,Verbose(Call start time: ${CDR(start)})
   same => n,Set(CDR(userfield)=zombie pancakes)
```

В дополнение к полям, которые всегда включены в CDR, можно добавить пользовательские поля. Это можно сделать в диалплане используя прилоежние Set() совместно с функцией CDR():

```
exten => 115,1,NoOp()
   same => n,Set(CDR(mycustomfield)=coffee)
   same => n,Verbose(I need some more ${CDR(mycustomfield)})
```

Если вы решите использовать пользовательские переменные CDR - убедитесь, что выбранный сервер CDR способен регистрировать их.

Чтобы Просмотреть встроенную документацию для функции `CDR()` - выполните следующую команду в консоли Asterisk:

```
*CLI> core show function CDR
```

В дополнение к функции `CDR()` некоторые приложения диалплана могут использоваться для влияния на записи CDR. Мы еще рассмотрим их.

Приложения диалплана

Несколько приложений диалплана можно использовать для влияния на CDR текущего вызова. Чтобы получить список приложений CDR, загруженных в текущей версии Asterisk, можно использовать следующую команду CLI:

```
*CLI> core show applications like CDR
    -= Matching Asterisk Applications =-
               ForkCDR: Forks the Call Data Record.
                 NoCDR: Tell Asterisk to not maintain a CDR for the current call
              ResetCDR: Resets the Call Data Record.
    -= 3 Applications Matching =-
```

Каждое приложение имеет документацию, встроенную в Asterisk, которую можно просмотреть с помощью следующей команды:

```
*CLI> core show application <application name>
```

cdr.conf

Файл *cdr.conf* имеет раздел `[general]`, содержащий параметры, применяемые ко всей системе CDR. Дополнительные необязательные разделы могут существовать в этом файле - они применяются к определенным модулям бэкэнда логирования CDR. Таблица 21-3 перечисляет параметры, доступные в разделе `[general]`.

Таблица 21-3. cdr.conf раздел [general]

Параметр	Значение/пример	Примечание
`enable`	`yes`	Включение ведения журнала CDR. Значение по умолчанию: `yes`.
`unanswered`	`no`	Регистрировать неотвеченные звонки. Обычно, только отвеченные вызовы пишутся в CDR. Регистрация всех попыток вызова может привести к большому количеству дополнительных записей вызовов, о которых большинство людей не заботятся. Значение по умолчанию: `no`.
`end before hexten`	`no`	Закройте CDR перед запуском расширения `h` в диалплане Asterisk. Обычно CDR не закрывают до тех пор, пока диалплан не будет полностью завершен. Значение по умолчанию: `no`.
`initiated seconds`	`no`	При расчете поля `billsec` всегда округляется. Например, если разница между временем ответа на вызов и временем его окончания составляет 1 секунду и 1 микросекунду, то `billsec` будет установлен равным 2 секундам. Это помогает гарантировать, что CDR Asterisk совпадает с поведением, используемым телекоммуникационными компаниями. Значение по умолчанию - `no`.
`batch`	`no`	Очередь записей CDR будет регистрироваться пакетами, а не синхронно в конце каждого вызова. Это предотвращает ведение журнала CDR от блокирования в конце вызова. Использование пакетного режима может быть невероятно полезно при работе с базой данных, которая может медленно обрабатывать запросы. Значение по умолчанию - `no`, но мы

Параметр	Значение/пример	Примечание
		рекомендуем включить его. [a]
size	100	Количество записей CDR в очереди до их регистрации в пакетном режиме. Значение по умолчанию: 100.
time	300	Установите максимальное количество секунд, в течение которых CDR будет ожидать очереди пакетной обработки перед регистрацией. Процесс пакетного ведения журнала CDR будет запущен в конце этого периода времени, даже если size ещё не накопился. Значение по умолчанию: 300 секунд.
sheduler only	no	Установите, должна ли пакетная обработка CDR выполняться путем порождения нового потока или в контексте планировщика пакетной обработки CDR. Значение по умолчанию: no и мы рекомендуем не менять его.
safe shutdown	yes	Заблокируйте выключение Asterisk, чтобы убедиться, что все записи CDR из очереди зарегистрированы. Значение по умолчанию: yes и мы рекомендуем оставить его таким, так как этот параметр предотвращает важную потерю данных.

[a] Недостатком включения этой опции является то, что если Asterisk по какой-либо причине упадет или умрет - записи CDR будут потеряны, так как они хранятся только в памяти существующего процесса Asterisk. Смотрите safeshutdown для дополнительной информации.

Бэкэнды

Модули бэкэнда Asterisk CDR предоставляют способ регистрации CDR. Большинство бэкендов CDR требуют определенной конфигурации, чтобы заставить их работать.

cdr_adaptive_odbc

Как следует из названия, модуль cdr_adaptive_odbc позволяет хранить CDR в базе данных через ODBC. "Адаптивная" часть названия относится к тому, что она работает для адаптации к структуре таблицы: нет статической структуры таблицы, которая должна использоваться с этим модулем. Когда модуль загружен (или перезагружен) - он считывает структуру таблицы. При регистрации CDR он ищет переменную CDR, соответствующую имени каждого столбца. Это относится как ко встроенным переменным CDR, так и к пользовательским. Если вы хотите писать встроенную переменную channel - просто создайте столбец с именем channel.

Добавление пользовательского содержимого CDR так же просто, как и его настройка в диалплане. Например, если мы хотим записывать User-Agent, предоставляемый SIP-устройством - мы могли бы добавить его в качестве пользовательской переменной CDR:

```
exten => 105,n,Set(CDR(useragent)=${CHANNEL(useragent)})
```

Чтобы эта пользовательская переменная CDR была вставлена в базу данных с помощью cdr_adaptive_odbc, все, что нам нужно сделать - это создать столбец под названием useragent.

Несколько таблиц могут быть указаны в конфигурационном файле cdr_adaptive_odbc. Каждая должна быть в своем собственном разделе конфигурации. Название раздела может быть любым - модуль не использует его. Вот пример простой конфигурации таблицы:

```
[mytable]

connection = asterisk
table = asterisk_cdr
```

Более подробный пример настройки базы данных для ведения журнала CDR можно найти в разделе "Хранение записей деталей вызовов".

Таблица 21-4 перечисляет параметры, которые могут быть указаны в разделе конфигурации таблицы в файле *cdr_adaptive_odbc.conf*.

Таблица 21-4. Таблица параметров конфигурации cdr_adaptive_odbc.conf

Параметр	Значение/пример	Примечание
connection	pgsql1	Подключение к используемой базе данных. Это ссылка на настроенное соединение в *res_odbc.conf*. Это поле является обязательным.
table	asterisk_cdr	Имя таблицы. Это поле является обязательным.
usegmtime	no	Указывает, следует ли регистрировать метки времени с помощью GMT вместо местного времени. Значением по умолчанию для этого параметра является no.

В дополнение к полям пары ключ/значение, которые показаны в предыдущей таблице - *cdr_adaptive_odbc.conf* позволяет использовать несколько других элементов конфигурации. Первый - это псевдоним (альяс) столбца. Обычно переменные CDR регистрируются в столбцах с тем же именем. alias позволяет сопоставить имя переменной со столбцом с другим именем. Синтаксис таков:

```
alias CDR variable => column name
```

Вот пример сопоставления столбцов с помощью параметра alias:

```
alias src => source
```

Также можно задать фильтр содержимого. Это позволяет задать критерии, которые должны совпадать, для записей, вставляемых в таблицу. Синтаксис таков:

```
filter CDR variable => content
```

Вот пример фильтра содержимого:

```
filter accountcode => 123
```

Наконец, *cdr_adaptive_odbc.conf* позволяет определять статическое содержимое для столбца. Это может быть полезно в сочетании с набором filters. Это статическое содержимое может помочь дифференцировать записи, вставленные в одну и ту же таблицу, по различным разделам конфигурации. Синтаксисом для статического содержимого является:

```
static "Здесь статическое содержимое" => column name
```

Ниже приведен пример указания статического содержимого, вставляемого с помощью CDR:

```
static "My Content" => my_identifier
```

cdr_csv

Модуль cdr_csv - это очень простой бэкэнд CDR, записывающий CDR в файл CSV (значения разделенные запятыми). Этот файл называется */var/log/asterisk/cdr-csv/Master.csv*. Пока ведение журнала CDR включено в *cdr.conf* и этот модуль загружен - CDR будут зарегистрированы в файле *Master.csv*. Мы рекомендуем, чтобы независимо от любого другого бэкенда CDR, выбранного Вами для настройки, Вы также оставили настроенным и его, поскольку он будет служить отличной резервной копией, если вы потеряете другие данные CDR из-за сети или связанных с ней проблем.

Хотя для обеспечения работы этого модуля никакие параметры не требуются - есть некоторые параметры, настраивающие его поведение. Эти параметры перечислены в разделе Таблица 21-5 и помещаюися в раздел [csv] *cdr.conf*.

Таблица 21-5. cdr.conf параметры раздела [csv]

Параметр	Значение/пример	Примечание
usegmtime	no	Регистрация меток времени, используя GMT вместо местного времени. Значение по умолчанию: no.
loguniqueid	no	Запись переменной CDR uniqueid. Значение по умолчанию: no.
loguserfield	no	Запись переменной CDR userfield. Значение по умолчанию: no.
accountlogs	yes	Создание отдельного CSV-файла для каждого отдельного значения параметра переменной CDR accountcode. Значение по умолчанию: yes.

Порядок переменных CDR в CSV файлах, созданных с помощью модуля cdr_csv:

```
<accountcode>,<src>,<dst>,<dcontext>,<clid>,<channel>,<dstchannel>,<lastapp>, \
    <lastadata>,<start>,<answer>,<end>,<duration>,<billsec>,<disposition>, \
    <amaflags>[,<uniqueid>][,<userfield>]
```

Поместите следующие строки в файл */etc/asterisk/cdr.conf*:

```
[general]
enable=yes

[csv]
usegmtime=yes     ; писать время в формате GMT. По умолчанию 'no'
loguniqueid=yes   ; писать uniqueid. По умолчанию 'no'
loguserfield=yes  ; писать пользовательское поле. По умолчанию 'no'
accountlogs=yes   ; создавать отдельный файл журнала для каждой учетной записи. По умолчанию
'yes'
;newcdrcolumns=yes ; включить ведение журнала в формате столбцов CDR после-1.8
(peeraccount,linkedid,sequence)
                   ; По умолчанию 'no'.
```

Сохраните его, смените владельца и перезагрузите модуль CDR.

```
$ chown asterisk:asterisk /etc/asterisk/cdr.conf

$ sudo asterisk-rx 'module reload cdr'
```

cdr_custom

Этот бэкэнд CDR позволяет выполнять пользовательское форматирование записей CDR в файле журнала. Этот модуль наиболее используется для подгоняемого выхода CSV. Файл конфигурации, используемый для этого модуля - это */etc/asterisk/cdr_custom.conf*. Единственный необходимый раздел `[mappings]` обязательно должен существовать в этом файле. Раздел `[mappings]` содержит сопоставления между именем файла и пользовательским шаблоном для CDR. Шаблон задается с помощью функций диалплана Asterisk.

В следующем примере показан пример конфигурации для `cdr_custom`, позволяющий использовать один CDR-файл журнала - *Master.csv*. Этот файл будет создан как */var/log/asterisk/cdr-custom/Master.csv*. Шаблон, который был определен, использует обе функции диалплана `CDR()` и `CSV_QUOTE()`. `CDR()` извлекает значения из регистрируемого CDR-файла. `CSV_QUOTE()` гарантирует что значения правильно экранированы для формата CSV:

```
[mappings]

Master.csv => ${CSV_QUOTE(${CDR(clid)})},${CSV_QUOTE(${CDR(src)})},
${CSV_QUOTE(${CDR(dst)})},${CSV_QUOTE(${CDR(dcontext)})},
${CSV_QUOTE(${CDR(channel)})},${CSV_QUOTE(${CDR(dstchannel)})},
${CSV_QUOTE(${CDR(lastapp)})},${CSV_QUOTE(${CDR(lastdata)})},
${CSV_QUOTE(${CDR(start)})},${CSV_QUOTE(${CDR(answer)})},
${CSV_QUOTE(${CDR(end)})},${CSV_QUOTE(${CDR(duration)})},
${CSV_QUOTE(${CDR(billsec)})},${CSV_QUOTE(${CDR(disposition)})},
${CSV_QUOTE(${CDR(amaflags)})},${CSV_QUOTE(${CDR(accountcode)})},
${CSV_QUOTE(${CDR(uniqueid)})},${CSV_QUOTE(${CDR(userfield)})}
```

В файле фактической конфигурации значения для сопоставления в *Master.csv* должны быть на одной строке.

cdr_manager

Бэкэнд `cdr_manager` выдает CDR в виде событий на Asterisk Manager Interface (AMI), который мы подробно обсудили в Главе 17. Этот модуль сконфигурирован в файле */etc/asterisk/cdr_manager.conf*. Первый раздел в этом файле — это раздел `[general]`, который содержит один параметр для включения этого модуля (значение по-умолчанию - `no`):

```
[general]

enabled = yes
```

Другой раздел в *cdr_manager.conf* - это `[mappings]`. Он позволяет добавлять пользовательские переменные CDR в события менеджера. Синтаксис таков:

```
CDR variable => Header name
```

Вот пример добавления двух пользовательских переменных CDR:

```
[mappings]

rate => Rate
carrier => Carrier
```

При такой конфигурации записи CDR будут отображаться в интерфейсе менеджера как события. Чтобы создать пример события мы будем использовать следующий пример диалплана:

```
exten => 110,1,Answer()
    same => n,Set(CDR(rate)=0.02)
    same => n,Set(CDR(carrier)=BS&S)
    same => n,Hangup()
```

Эта команда используется для выполнения данного расширения и создания примера события диспетчера:

```
*CLI> console dial 110@testing
```

Наконец, вот пример события менеджера, созданного в результате этого тестового вызова:

```
Event: Cdr
Privilege: cdr,all
AccountCode:
Source:
Destination: 110
DestinationContext: testing
CallerID:
Channel: Console/dsp
DestinationChannel:
LastApplication: Hangup
LastData:
StartTime: 2010-08-23 08:27:21
AnswerTime: 2010-08-23 08:27:21
EndTime: 2010-08-23 08:27:21
Duration: 0
BillableSeconds: 0
Disposition: ANSWERED
AMAFlags: DOCUMENTATION
UniqueID: 1282570041.3
UserField:
Rate: 0.02
Carrier: BS&S
```

cdr_odbc

Этот модуль включает устаревший интерфейс ODBC для ведения журнала CDR. Новые установки должны использовать `cdr_adaptive_odbc` вместо него.

cdr_sqlite

Этот модуль позволяет отправлять CDR в базу данных SQLite с помощью SQLite версии 2. Если у вас нет особой необходимости в SQLite версии 2 в отличие от версии 3 - мы рекомендуем использовать для новых установок `cdr_sqlite3_custom`.

Этот модуль не требует никакой конфигурации для работы. Если модуль был скомпилирован и загружен в Asterisk - он вставит CDR в таблицу под названием `cdr` в базе данных, расположенной по адресу */var/log/asterisk/cdr.db*.

cdr_sqlite3_custom

Этот бэкэнд CDR вставляет CDR в базу данных SQLite с помощью SQLite версии 3. База данных, созданная этим модулем, находится в каталоге */var/log/asterisk/master.db*. Для этого модуля требуется

файл конфигурации */etc/asterisk/cdr_sqlite3_custom.conf*. Файл конфигурации определяет имя таблицы, а также настраивает какие переменные CDR будут вставлены в базу данных.

cdr_syslog

Этот модуль позволяет вести журнал CDR используя `syslog`. Чтобы включить эту функцию, сначала добавьте запись в файл конфигурации *syslog - /etc/syslog.conf*. Например:

```
local4.*  /var/log/asterisk/asterisk-cdr.log
```

Модуль Asterisk также имеет файл конфигурации. Добавьте следующий раздел в файл */etc/asterisk/cdr_syslog.conf*:

```
[cdr]

facility = local4
priority = info
template = "We received a call from ${CDR(src)}"
```

Вот пример записи системного журнала, использующего эту конфигурацию:

```
$ cat /var/log/asterisk/asterisk-cdr.log

Aug 12 19:17:36 pbx cdr: "We received a call from 2565551212"
```

Пример записей деталей вызова

Мы будем использовать модуль `cdr_custom` для иллюстрации некоторых примеров записей CDR для различных сценариев вызовов. Конфигурация, используемая для файла */etc/asterisk/cdr_custom.conf* представлена в "cdr_custom".

Односторонний вызов

В этом примере мы покажем, как выглядит CDR для простого одностороннего вызова:

```
exten => 227,1,VoiceMailMain(@${GLOBAL(VOICEMAIL_CONTEXT)})
```

Вот запись CDR из */var/log/asterisk/cdr-custom/Master.csv*, которая была создана в результате вызова этого расширения:

```
"","SOFTPHONE_A","227","sets","""101"" <SOFTPHONE_A>","PJSIP/SOFTPHONE_A-00000002",
"","Playback","hear-odd-noise",
"2019-03-04 02:31:39","2019-03-04 02:31:39","2019-03-04 02:31:42",
3,3,"ANSWERED","DOCUMENTATION","1551666699.4",""
```

Откройте его в электронной таблице, и он будет аккуратно выстроен.

Предостережения

Система CDR в Asterisk очень хорошо работает для довольно простых сценариев вызовов. Однако, поскольку сценарии вызовов становятся более сложными - включая звонки нескольким сторонам, трансферы, парковку и другие подобные функции — система CDR начинает отставать. Многие пользователи сообщают, что записи не отображают всю информацию, которую они ожидают. Многие исправления ошибок были сделаны для решения некоторых проблем, но стоимость регрессий или изменений в поведении при внесении изменений в этой области очень высока, так как эти записи используются для биллинга.

В результате команда разработчиков Asterisk стала все более сопротивляться к внесению дополнительных изменений в систему CDR. Вместо этого была разработана новая система, Channel event logging (CEL), которая предназначена для решения проблемы ведения журнала более сложных сценариев вызовов. Имейте в виду, что записи деталей вызовов являются более простыми и легкими в использовании - именно поэтому мы все еще рекомендуем использовать CDR если они удовлетворяют вашим потребностям.

Регистрация событий канала

Регистрация событий канала (CEL) предоставляет более гибкие средства регистрации деталей сложных сценариев вызова. Вместо сворачивания вызова до одной записи журнала - регистрируется ряд событий для вызова. Это дает более точную картину того, что произошло с вызовом, за счет более сложного лога.

Для получения более подробной информации о CEL - ознакомьтесь с wiki Asterisk.

Вывод

Asterisk очень хорошо позволяет отслеживать множество различных аспектов её работы - от простых записей детализации вызовов до полной отладки выполняемого кода. Загляните в каталоги исходного кода и вы найдете гораздо больше компонентов, чем у нас было места для освещения здесь. Эти различные механизмы помогут вам в усилиях по управлению вашей АТС Asterisk и они представляют собой один из способов, которым Asterisk значительно превосходит большинство (если не все) традиционных АТС.

Глава 22. Безопасность

Мы тратим время на поиски безопасности и ненавидим, когда получаем ее.
– Джон Стейнбек

Безопасность вашей системы Asterisk имеет решающее значение, особенно если система подключена к Интернету. Злоумышленники могут заработать много денег, используя системы для бесплатных телефонных звонков. В этой главе даются советы о том, как обеспечить более надежную защиту для вашего развертывания VoIP.

Сканирование действительных учетных записей

Если вы выставляете свою систему Asterisk в общедоступный Интернет, одна из вещей, которую вы почти наверняка увидите - это сканирование действительных учетных записей. Пример 22-1 содержит записи лога с одной из производственных систем Asterisk авторов.[1] Это сканирование началось с проверки различных общих имен пользователей, а затем перешло к сканированию числовых аккаунтов. Обычно пользователи называют учетные записи SIP так же, как и внутренние номера на АТС. Подобное сканирование использует этот факт.

Используйте нечисловые имена пользователей для своих учетных записей VoIP, чтобы их было сложнее угадать. Например, в этой книге мы используем MAC-адрес SIP-телефона в качестве имени учетной записи в Asterisk.

Пример 22-1. Отрывок лога сканирования учетной записи

```
[Aug 22 15:17:15] NOTICE[25690] chan_sip.c: Registration from
'"123"<sip:123@127.0.0.1>' failed for '203.86.167.220:5061' - No matching peer
found
[Aug 22 15:17:15] NOTICE[25690] chan_sip.c: Registration from
'"1234"<sip:1234@127.0.0.1>' failed for '203.86.167.220:5061' - No matching peer
found
[Aug 22 15:17:15] NOTICE[25690] chan_sip.c: Registration from
'"12345"<sip:12345@127.0.0.1>' failed for '203.86.167.220:5061' - No matching peer
found

...

[Aug 22 15:17:17] NOTICE[25690] chan_sip.c: Registration from
'"100"<sip:100@127.0.0.1>' failed for '203.86.167.220:5061' - No matching peer found
[Aug 22 15:17:17] NOTICE[25690] chan_sip.c: Registration from
'"101"<sip:101@127.0.0.1>' failed for '203.86.167.220:5061' - No matching peer found
```

В любой системе логи будут полны попыток вторжения. Это просто характер подключения систем к интернету. В этой главе мы обсудим некоторые способы настройки вашей системы таким образом, чтобы она имела надежные механизмы для решения этих проблем.

Уязвимости аутентификации

В первом разделе этой главы обсуждалось сканирование имен пользователей. Даже если у вас есть имена пользователей, которые трудно угадать, очень важно, чтобы у вас были надежные пароли. Если злоумышленник может получить действительное имя пользователя - он, скорее всего, попытается подобрать пароль. Надежные пароли усложняют эту задачу.

1 Реальный IP-адрес был заменен на 127.0.0.1 в записях лога.

Схема аутентификации SIP-протокола по умолчанию является слабой. Аутентификация выполняется с помощью механизма вызова и ответа MD5. Если злоумышленник сможет перехватить любой трафик, например SIP-вызов, выполненный с ноутбука в открытой беспроводной сети, ему будет намного проще работать с брутфорсом паролей, поскольку это не потребует запросов аутентификации у сервера.

Используйте надежные пароли. В Интернете доступно множество ресурсов, которые помогают определить, что представляет собой надежный пароль. Есть также много надежных генераторов паролей. Используй их!

Fail2ban

В предыдущих двух разделах рассматривались атаки, связанные со сканированием действительных имен пользователей и подбором паролей брутфорсом. Fail2ban - это приложение, которое может просматривать журналы Asterisk и обновлять правила брандмауэра чтобы блокировать источник атаки в ответ на слишком большое количество неудачных попыток аутентификации.

Используйте Fail2ban при предоставлении услуг Voice over IP в ненадежных сетях. Оно автоматически обновит правила брандмауэра, чтобы заблокировать источники атак.

Установка

Fail2ban доступен в виде пакета во многих дистрибутивах. Кроме того, вы можете установить его из исходников, загрузив с веб-сайта Fail2ban. Чтобы установить Fail2ban на RHEL, необходимо включить репозиторий EPEL (который был рассмотрен в Главе 3). Вы можете установить Fail2ban, выполнив следующую команду:

```
$ sudo yum install fail2ban
```

Установка Fail2ban из пакета будет включать скрипт запуска, чтобы гарантировать запуск при загрузке компьютера. Если вы устанавливаете из исходных кодов - убедитесь, что вы предпринимаете необходимые шаги для гарантированной постоянной работы Fail2ban.

Конфигурация

Во-первых, мы хотим настроить журнал безопасности в Asterisk, который сможет использовать Fail2ban.

```
$ sudo vim /etc/asterisk/logger.conf
```

Раскомментируйте (или добавьте) строку, которая разрешает чтение `security => security` и измените `dateformat` для понимания её в журнале Fail2ban.

```
[general]
exec_after_rotate=gzip -9 ${filename}.2;
dateformat = %F %T
[logfiles]
;debug => debug
security => security
;console => notice,warning,error,verbose
console => notice,warning,error,debug
messages => notice,warning,error
full => notice,warning,error,debug,verbose,dtmf,fax
```

Затем перезагрузите logger Asterisk:

```
$ sudo asterisk -rx 'logger reload'
```

Поскольку текущие версии Fail2ban уже поставляются с определением изолятора Asterisk - все, что нам нужно сделать, это включить его:

Для этого рекомендуется создать файл */etc/fail2ban/jail.local* (технически вы можете поместить его в */etc/fail2ban/jail.conf*, но он скорее всего будет перезаписан):

```
$ sudo vim /etc/fail2ban/jail.local

[asterisk]
enabled  = true
filter   = asterisk
action   = iptables-allports[name=ASTERISK, protocol=all]
           sendmail[name=ASTERISK, dest=me@shifteight.org, sender=fail2ban@shifteight.org]
logpath  = /var/log/asterisk/messages
           /var/log/asterisk/security
maxretry = 5
findtime = 21600
bantime  = 86400
```

Мы установили запрет на 24 часа, но вы можете сделать время больше или меньше, как пожелаете (время запрета определяется в секундах - так что его необходимо рассчитать). Поскольку большинство атакующих хостов меняются через несколько часов - нет никакого вреда в разблокировании IP-адреса через 24 часа. Если хост атакует снова - он снова будет заблокирован.

О, вы также можете указать ему игнорировать ваш IP (или любые другие IP-адреса, с которых можно получать попытки подключения). Если вы случайно заблокировали себя, когда делали какую-то лабораторную работу и неправильно регистрировались , не волнуйтесь - вы в конечном итоге сделаете это с собой (если, конечно, не создадите список игнорирования для соответствующих IP-адресов).

```
[DEFAULT]
ignoreip = <ip-адрес(а), разделенные запятыми>

[asterisk]
enabled  = true
filter   = asterisk
action   = iptables-allports[name=ASTERISK, protocol=all]
           sendmail[name=ASTERISK, dest=me@shifteight.org, sender=fail2ban@shifteight.org]
logpath  = /var/log/asterisk/messages
           /var/log/asterisk/security
maxretry = 5
findtime = 21600
bantime  = 86400
```

Перезапустите Fail2ban и все будет хорошо.

```
$ sudo systemctl reload fail2ban
```

Проверьте это, если можете, с IP-адреса, который вы не против заблокировать (например, дополнительный компьютер в вашей лаборатории, который может стать объектом тестирования в данном случае). Попытайтесь зарегистрироваться с использованием неверных учетных данных, и после пяти попыток (или любого другого значения, для которого вы установили `maxretry`) этот IP-адрес должен быть заблокирован.

Вы можете увидеть какие адреса блокирует Asterisk jail, с помощью команды:

```
$ sudo fail2ban-client status asterisk
```

И если вы хотите разблокировать IP[2] - следующая команда должна сделать это.

```
$ sudo fail2ban-client set asterisk unbanip <ip для разбанивания>
```

Дополнительную информацию о Fail2ban можно найти на странице Fail2ban wiki.

2 Например, себя, потому что вы забыли определить `ignoreip`...

Шифрование медиапотока

В то время как мы приводили в этой книге примеры, использующие шифрование, имейте в виду, что вы можете настроить SIP так, что медиапоток будет отправляться в незашифрованном виде. В этом случае любой, кто перехватит RTP-трафик между двумя SIP-узлами, сможет использовать довольно простые инструменты для извлечения звука из этих вызовов.

Уязвимости диалплана

Диалплан Asterisk - это еще одна область где важна безопасность. Диалплан можно разбить на несколько контекстов для обеспечения управления доступом к расширениям. Например, вы можете разрешить своим офисным телефонам совершать звонки через провайдера. Тем не менее, вы не захотите, чтобы анонимные абоненты, попадающие в главное меню вашей компании, могли затем набрать номер через вашего провайдера. Используйте контексты чтобы гарантировать доступ к услугам, которые стоят вам денег, только доверенным абонентам.

Создавайте контексты диалплана с большой осторожностью. Кроме того, избегайте размещения любых расширений, которые могут стоить вам денег в контексте `[default]`.

Одна из последних обнаруженных и опубликованных уязвимостей диалплана Asterisk - это идея инъекций диалплана. Уязвимость инъекций диалплана начинается с расширения, имеющего шаблон, который заканчивается символом соответствия всему - точкой. Возьмите это расширение в качестве примера:

```
exten => _X.,1,Dial(PJSIP/otherserver/${EXTEN},30)
```

Шаблон для этого расширения соответствует всем расширениями (любой длины), которые начинаются с цифры. Такие шаблоны довольно распространены и удобны. Затем расширение отправляет этот вызов на другой сервер, используя протокол IAX2, с таймаутом набора 30 секунд. Обратите внимание на использование здесь переменной `${EXTEN}`. Вот где кроется уязвимость.

В мире Voice over IP нет причин, по которым набираемый номер должен быть числовым. На самом деле, это довольно распространенное использование SIP, чтобы иметь возможность набрать кого-то по имени. Поскольку нечисловые символы могут быть частью набранного добавочного номера, что произойдет, если кто-то отправит вызов на такой добавочный номер?

```
1234&DAHDI/g1/12565551212
```

Подобный вызов является попыткой использования уязвимости инъекции диалплана. В предыдущем определении расширения, как только `${EXTEN}` был вычислен - фактический оператор `Dial()`, который будет исполняться, будет иметь вид:

```
exten => _X.,1,Dial(PJSIP/otherserver/1234&DAHDI/g1/12565551212,30)
```

Если в системе настроена PRI - этот вызов совершит вызов через PRI на номер, выбранный злоумышленником, даже если вы явно не предоставили доступ к PRI этому абоненту. Эта проблема может стоить вам больших затрат.

Существует несколько подходов к решению этой проблемы. Первый и самый простой подход - всегда использовать строгое сопоставление шаблонов. Если вы знаете длину ожидаемых расширений и ожидаете только числовые расширения - используйте строгое соответствие числовому шаблону. Например, такой шаблон будет работать, если вы ожидаете только четырехзначные числовые номера:

```
exten => _XXXX,1,Dial(PJSIP/otherserver/${EXTEN},30)
```

Другой подход к смягчению уязвимостей инъекций диалплана заключается в использовании функции диалплана `FILTER()` . Возможно, вы хотели бы разрешить числовые расширения любой длины. `FILTER()` позволяет сделать это легко и безопасно:

```
exten => _X.,1,Set(SAFE_EXTEN=${FILTER(0-9A-F,${EXTEN})})
    same => n,Dial(PJSIP/otherserver/${SAFE_EXTEN},30)
```

Дополнительные сведения о синтаксисе функции диалплана `FILTER()` см. в выводе данных команды `core show function FILTER` в Asterisk CLI.

Более комплексный (но и сложный) подход может заключаться в проверке всех набранных цифр функциями, не входящими в диалплан (например, запросы к базе данных, которые проверяют набранную строку на соответствие разрешениям пользователя, шаблонам маршрутизации, таблицам ограничений и т.д.). Это мощная концепция, но она выходит за рамки данной книги.

Будьте осторожны с уязвимостями инъекциями диалплана. Используйте строгое сопоставление шаблонов или используйте функцию диалплана `FILTER()` во избежание подобных проблем.

Безопасность сетевых API Asterisk

Чтобы защитить AGI, AMI и ARI - вам нужно будет тщательно рассмотреть следующие рекомендуемые методы:

- Разрешить подключения непосредственно к API только из `localhost/127.0.0.1`.
- Использовать соответствующую платформу между API Asterisk и вашим клиентским приложением, а также обрабатывать безопасность соединения через фреймворк.
- Контролировать доступ к фреймворку и системе с помощью строгих правил брандмауэра.

Кроме того, применяются те же правила безопасности и рекомендации, что и в любом критически важном веб-приложении.

Другие меры по снижению риска

В Asterisk есть и другие полезные функции, которые можно использовать для снижения риска атак. Первый заключается в использовании параметров `permit` и `deny` для создания списков управления доступом (ACL) для привилегированных учетных записей. Рассмотрим АТС, которая имеет SIP-телефоны в локальной сети, но также принимает SIP-вызовы через интернет. Такие вызовы получают доступ только к главному меню компании, в то время как локальные SIP-телефоны имеют возможность совершать исходящие вызовы, которые уже стоят денег. В этом случае рекомендуется настроить списки управления доступом, чтобы только устройства в локальной сети могли использовать учетные записи для телефонов.

В таблице `ps_endpoints` параметры `permit` и `deny` позволяют указать IP-адреса, но также можно указать метку в файле `/etc/asterisk/acl.conf`. Фактически ACL принимаются почти везде, где настроены подключения к IP-службам. Например, еще одно полезное место для ACL находится в файле */etc/asterisk/manager.conf* для ограничения учетных записей AMI до одного хоста, который может использовать интерфейс менеджера.

ACL можно определить в */etc/asterisk/acl.conf*.

```
[named_acl_1]
deny=0.0.0.0/0.0.0.0
permit=10.1.1.50
permit=10.1.1.55

[named_acl_2] ; Именованные ACLs также поддерживают IPv6.
deny=::
permit=::1/128

[local_phones]
```

```
deny=0.0.0.0/0.0.0.0
permit=192.168.0.0/255.255.0.0
```

Когда именованные ACL были определены в *acl.conf* - попросите Asterisk загрузить их с помощью команды `reload acl`. После загрузки они должны быть доступны через интерфейс командной строки Asterisk:

```
*CLI> module reload acl

*CLI> acl show

acl
---
named_acl_1
named_acl_2
local_phones

*CLI> acl show named_acl_1

ACL: named_acl_1
-----------------------------------------
  0: deny - 0.0.0.0/0.0.0.0
  1: allow - 10.1.1.50/255.255.255.255
  2: allow - 10.1.1.55/255.255.255.255
```

Теперь, вместо того, чтобы потенциально повторять одни и те же записи `permit` и `deny` в нескольких местах, вы можете применить ACL по его имени. Вы найдете поле `acl` в таблице `ps_endpoints`, которое можно использовать для указания на именованный ACL в файле *acl.conf*.

```
mysql> select id,transport,aors,context,disallow,allow,acl from ps_endpoints;

|id          |transport    |aors        |context|disallow|allow        |acl |
|0000f30A0A01|transport-udp|0000f30A0A01|sets   |all     |ulaw         |NULL|
|0000f30B0B02|transport-udp|0000f30B0B02|sets   |all     |ulaw         |NULL|
|SOFTPHONE_A |transport-udp|SOFTPHONE_A |sets   |all     |ulaw,h264,vp8|NULL|
|SOFTPHONE_B |transport-udp|SOFTPHONE_B |sets   |all     |ulaw,h264,vp8|NULL|

mysql> update ps_endpoints
        set acl='local_phones'
        where id in ('0000f30A0A01','0000f30B0B02','SOFTPHONE_A','SOFTPHONE_B')
       ;

mysql> select id,transport,aors,context,disallow,allow,acl from ps_endpoints;

|id          |transport    |aors        |context|disallow|allow        |acl         |
|0000f30A0A01|transport-udp|0000f30A0A01|sets   |all     |ulaw         |local_phones|
|0000f30B0B02|transport-udp|0000f30B0B02|sets   |all     |ulaw         |local_phones|
|SOFTPHONE_A |transport-udp|SOFTPHONE_A |sets   |all     |ulaw,h264,vp8|local_phones|
|SOFTPHONE_B |transport-udp|SOFTPHONE_B |sets   |all     |ulaw,h264,vp8|local_phones|
```

Используйте ACL, когда это возможно, на всех привилегированных аккаунтах для сетевых служб.

Ещё одним способом снижения риска безопасности является настройка лимитов вызовов. Рекомендуемый метод реализации ограничений вызовов - использование функций диалплана `GROUP()` и `GROUP_COUNT()`. Вот пример, ограничивающий количество вызовов от каждого узла SIP не более чем двумя одновременно:

```
exten => _X.,1,Set(GROUP(users)=${CHANNEL(endpoint)})
    same => n,NoOp(${CHANNEL(endpoint)} : ${GROUP_COUNT(${CHANNEL(endpoint)})} calls)
    same => n,GotoIf($[${GROUP_COUNT(${CHANNEL(endpoint)})} > 2]?denied:continue)
    same => n(denied),NoOp(There are too many calls up already. Hang up.)
    same => n,HangUp()
    same => n(continue),NoOp(continue processing call as normal here ...)
```

Используйте ограничения вызовов, для гарантии что если учетная запись скомпрометирована - ее нельзя использовать для совершения сотен телефонных звонков одновременно.

Ресурсы

Для устранения некоторых уязвимостей системы безопасности необходимо внести изменения в исходный код Asterisk. Когда эти проблемы обнаруживаются - команда разработчиков Asterisk выпускает новые релизы, содержащие только исправления для проблем безопасности, чтобы обеспечить быстрое и легкое обновление. В этом случае команда разработчиков Asterisk также публикует рекомендательный документ по безопасности, в котором обсуждаются сведения об уязвимости. Мы рекомендуем Вам подписаться на список рассылки *asterisk-announce*, чтобы убедиться, что вы узнаете об этих проблемах, когда они возникают.

Подпишитесь на список asterisk-announce, чтобы быть в курсе уязвимостей системы безопасности Asterisk.

Одним из самых популярных инструментов для сканирования учетных записей SIP и взлома паролей является SIPVicious. Мы настоятельно рекомендуем вам взглянуть на него и использовать для аудита ваших собственных систем. Если ваша система доступна из интернета - другие, скорее всего, запустят SIPVicious против неё - поэтому убедитесь, что вы сделали это в первую очередь.

Вывод—Лучший идиот

В технологической индустрии есть принцип, который гласит: "Как только что-то станет идиотским - природа изобретет лучшего идиота." Суть этого заявления заключается в том, что никакие усилия в области развития не могут считаться завершенными. Всегда есть место для улучшения.

Когда дело доходит до безопасности - вы всегда должны иметь в виду, что люди, которые хотят воспользоваться вашей системой, очень мотивированы. Независимо от того, насколько безопасна ваша система, кто-то всегда будет искать пути для её взлома.

Мы не защищаем паранойю, но предполагаем, что то, что мы описали здесь, ни в коем случае не является последним словом о безопасности VoIP. Несмотря на то, что в этой книге мы постарались быть максимально всеобъемлющими - вы должны взять на себя ответственность за безопасность своей системы.

Преступники упорно работают, чтобы найти слабые места и использовать их.

Глава 23. Asterisk: Будущее телефонии

Эй, я только что познакомился с тобой,
И это безумие!
Но вот мой номер телефона,
Так что позвони мне, может быть?
– Карли Рей Джепсен

Мы подошли к последней главе этой книги. Мы рассмотрели много (и эта книга была значительно доработана за эти годы), но надеемся что мы ясно дали понять, что лишь просто поцарапали поверхность Asterisk. Чтобы завершить работу мы хотим потратить некоторое время на изучение того, что могли бы увидеть от Asterisk и open source телефонии в ближайшем будущем.

Когда ещё писали первое издание *Asterisk: Будущее телефонии* мы уверенно утверждали, что коммуникационные механизмы с открытым кодом, такие как Asterisk, вызовут сдвиг в мышлении, который трансформирует телекоммуникационную отрасль. Во многих отношениях наша вера оказалась правильной; однако некоторые могут утверждать, что это была пустая победа, потому что то, что также произошло за это время - это сдвиг от телекоммуникаций как основного средства связи в реальном времени. Более молодые поколения почти не пользуются телефонными звонками и считают их разрушительными, раздражающими, а в некоторых случаях даже грубыми.

Таким образом, несмотря на то, что Asterisk открыла новый век для телекоммуникационной индустрии - теперь она стала эталоном для технологий, которые, по мнению многих, так же хороши, как и мертвы.

Хотя не может быть никаких сомнений в том, что телефон больше не является основной коммуникационной технологией в мире (далеко не случайно!), если мы проанализируем коммуникации до их сути - мы обнаружим, что у этого материала еще есть будущее.

Телефон мертв (за исключением тех случаев, когда это не так)

Хотя очевидно, что молодые поколения уже не так часто пользуются телефоном, верно также и то, что старшие поколения очень недовольны и разочарованы современными коммуникационными технологиями. Для них телефон представляет собой надежный, предсказуемый и простой в понимании способ связи, и они, вероятно, будут продолжать использовать его до конца своей жизни. Поскольку в этом мире ужасно много пожилых людей, и многие из них являются высшими руководителями, лицами, принимающими решения, и акционерами — не говоря уже о состоятельных клиентах, похоже, что сегодня хорошей стратегией для бизнеса является продолжение обеспечения того, с помощью чего их клиенты могли связаться с ними по телефону.

Когда человек попробует все другие способы коммуникации, такие как электронная почта, веб-формы и, возможно, даже текстовые сообщения - он, наконец, возьмет телефон и позвонит. Похоже, что во многих случаях проблема, которая не могла быть решена никаким другим способом, наконец-то решится по телефону.

Было бы также правильно сказать, что все более бедные рабочие компании, занимающиеся обработкой коммуникаций со своими клиентами, являются источником большого разочарования и путаницы. Однако, как всегда, там, где есть проблема, есть и возможность. Компании, которые сохраняют приверженность к превосходной телекоммуникационной инфраструктуре, могут оказаться с явным конкурентным преимуществом, не используя ничего более сложного, чем хорошее

старомодное обслуживание клиентов. Если вы хотите обслуживать клиентов старше 50 лет - вы должны сделать все возможное, чтобы ваша телефонная система работала хорошо.

Еще один интересный компонент традиционных телекоммуникационных сетей заключается в том, что, хотя мы никогда не можем быть уверены в том, что используем одно и то же программное обеспечение для конференц-связи (никогда в истории не было так много почти идентичных приложений, которые должны были быть установлены только для того, чтобы люди могли говорить друг с другом), мы можем быть разумно уверены, что если один из нас возьмет телефон и наберет номер телефона другого - успешный разговор будет возможен без каких-либо неисправностей или установки программного обеспечения. В эпоху, когда кажется, что никакая конференция не может начаться без того, чтобы кому-то не пришлось устранять неполадки в своем приложении, этот вид универсальной согласованности и надежности, вероятно, все еще имеет некоторое значение. Сегодняшнее горячее новое программное обеспечение для совместной работы в офисе - это завтрашняя забытая игрушка (куда ты, Skype?). Храбрые старые телефонные солдаты включились.

Мы пока не уверены, что телефон мертв.

Перегрузка связи

Во многих отношениях способность к общению определяет наш вид. Да, другие существа способны сигнализировать друг другу базовыми способами, но наше увлечение созданием постоянно меняющихся и инновационных способов коммуникаций друг с другом - это не то, с чем бы сталкивалось любое другое существо.

От почтового голубя до телеграфа, телефона и телевидения, каждая новая технология служила одной и той же цели: улучшению нашей способности общаться. Сегодня мы достигли самой замечательной вещи: теперь разумно ожидать мгновенного общения практически с любым человеком на планете.

Проблема, которую мы никогда не предсказывали, заключается в том, что слишком много хорошего начало подавлять нас. Будет интересно посмотреть, как это происходит в культурном плане.

Проблемы с разработкой открытого исходного кода

Хотя Александр Грэм Белл больше всего известен как отец телефона,[1] реальность такова, что во второй половине 1800-х годов десятки умов работали над достижением цели передачи голоса по телеграфным линиям. Эти люди были в основном деловыми людьми, стремящимися создать продукт, с помощью которого они могли бы сделать свои состояния.

Мы привыкли думать о традиционных телефонных компаниях как о монополиях, но это было не так в их первые дни. Ранняя история телефонных услуг проходила в очень конкурентной среде, с появлением новых компаний по всему миру, часто почти без уважения к патентам, которые они могли бы нарушить. Многие известные монополии получили свое начало через ведение (и победу) патентных войн.

Интересно сравнить историю телефона с историей GNU Linux и Интернета. В то время как телефон был создан как коммерческое мероприятие, а телекоммуникационная индустрия была создана с помощью судебных процессов и корпоративных поглощений, Linux и Интернет возникли из академического сообщества, которое имело тенденцию ценить обмен знаниями над прибылью.

К сожалению, в очередной раз слишком много хорошего стало захлестывать. То, что мы видели в последнее время - это потеря видения для разработки с открытым исходным кодом. Слишком мало разработчиков устали от требований слишком большого количества пользователей, не желающих вносить свой вклад. Большинство проектов с открытым исходным кодом должны были по необходимости оградить команду разработчиков от эгоистичных требований тех, кто намерен только

1 Вы когда-нибудь слышали об Элише Грее или Антонио Меуччи?

брать и никогда не давать. Это злоупотребление разработчиками, к сожалению, даже распространилось на компании, которые построили высокодоходные бизнесы на проектах с открытым исходным кодом, в которые они никогда не вносили ни копейки. Многомиллиардный бизнес, получающий прибыль от усилий команды, едва способной оплачивать свои счета, не является моделью устойчивого развития. Остается посмотреть как эта история будет развиваться, но программное обеспечение с открытым исходным кодом уже не то, что было 10 лет назад.

Asterisk повезло в том, что он финансируется усилиями Sangoma/Digium - родителями проекта. Их задачей было и всегда будет выяснить, как воспитать продукт таким образом, чтобы требования бизнеса были совместимы с потребностями проекта. Это была непростая задача. Мы будем болеть за них. До этих пор они проделали замечательную работу.

Будущее Asterisk

Итак, есть ли у Asterisk будущее? Мы не видим, почему его не должно быть. Он продолжает делать то, что он всегда делал и также упорно работает, чтобы быть совместимым с подходящими технологиями, выходящими на поверхность. По крайней мере, Asterisk будет продолжать очень хорошо интегрироваться с телефонными технологиями и мы пока не готовы назвать эту историю полностью рассказанной.

WebRTC

Следите за WebRTC. Мы подозреваем, что если у open source и open-standards коммуникаций есть какое-то будущее, WebRTC выступает в качестве наиболее перспективного кандидата для достижения этого.

Asterisk вряд ли будет в центре этой революции, но он будет играть определенную роль.

Будущее телефонии

Телефония может выглядеть мертвой, но мы все еще видим движение в хвосте, и это действительно длинный хвост.

Алфавитный указатель

Об авторах

Джим Ван Меггелен является партнером-основателем и техническим директором компании Clearly Core Inc. - канадского поставщика решений для телефонии с открытым исходным кодом. Он имеет почти 30-летний опыт работы в сфере корпоративных телекоммуникаций, обладая обширными знаниями как в области устаревших телекоммуникаций, так и в области VoIP.

Рассел Брайант - выдающийся инженер компании Red Hat, где он работает над проектами облачной инфраструктуры. До работы в Red Hat Рассел провел семь лет работая в Digium над проектом Asterisk. Роль Рассела в Digium началась как разработчика программного обеспечения и завершилась тем, что он стал руководителем проекта Asterisk и инженером-менеджером команды, занимающейся разработкой Asterisk.

Лейф Медсен является архитектором обеспечения облачных услуг в команде CloudOps в Red Hat, где он руководит инженерными разработками по обеспечению гарантий обслуживания как телекоммуникационных, так и корпоративных компаний. Он впервые связался с сообществом Asterisk, когда искал решение для голосовой конференции. Как только он узнал, что официальной документации Asterisk не существует - он стал соучредителем проекта Asterisk Documentation Project.

Послесловие

Животное на обложке *Asterisk: Окончательное руководство* является морской звездой (*Asteroidea*), группа иглокожих (иглокожие беспозвоночные, встречающиеся только в море). Большинство морских звезд имеют пятикратную радиальную симметрию (плечи или лучи, ответвляющиеся от центрального диска тела кратны пяти), хотя некоторые виды имеют четыре или девять плеч. Существует более 1500 видов морских звезд.

Морские звезды живут на морском дне и в приливных бассейнах, цепляясь за скалы и передвигаясь (медленно), используя сосудистую систему на водной основе, чтобы манипулировать сотнями крошечных трубчатых ног, называемых *подиями*. Маленькая луковица или *ампула* в верхней части трубки сжимается, выталкивая воду и расширяя ногу морской звезды. Ампула расслабляется и нога втягивается. На кончике каждой ножки находится присоска, которая позволяет морской звезде вскрывать раковины моллюсков, устриц или мидий. Морские звезды - плотоядные животные, они едят кораллы, рыбу, двустворчатых моллюсков и улиток.

Морские звезды могут сгибать и манипулировать своими руками, чтобы поместиться в небольших местах. На конце каждой руки находится глазок, примитивный датчик, который обнаруживает свет и помогает морской звезде определить направление. Морские звезды также обладают способностью регенерировать отсутствующую конечность. Некоторые виды могут даже вырастить полную, новую морскую звезду из отрубленной руки.

Многие животные на обложках О’Рейли находятся под угрозой исчезновения; все они важны для мира. Иллюстрация обложки - Карен Монтгомери, основанная на черно-белой гравюре из *Дуврского живописного архива*. ~~Шрифты обложки - Gilroy Semibold и Guardian Sans. Шрифт текста в Adobe Minion Pro; шрифт заголовка является Adobe Myriad Condensed; и шрифт кода Далтона Маага - Ubuntu Mono.~~