EXPERT INSIGHT

Mastering Python Networking

Your one-stop solution to using Python for network automation, programmability, and DevOps

Foreword by:
Michael Kennedy
Founder, Talk Python

Third Edition

Eric Chou

Packt>

Освоение сетевых технологий Python
Третье издание

Предисловие

Эта книга, которую вы держите в руках или читаете на экране, обладает силой, которая может стать вашей, если вы уделите время ее изучению. Подобно молоту Тора или костюму Железного человека, программирование - это суперсила, которая усиливает ваши существующие знания и навыки. Многие люди считают, или им говорят, что они должны изучать программирование и Python ради них самих. Навыки программирования востребованы, поэтому вы должны стать программистом. Возможно, это хороший совет. Но лучшим советом было бы ответить на вопрос: "Как вы можете воспользоваться имеющимся опытом и опередить своих коллег, автоматизировав и расширив этот опыт с помощью навыков программирования?". Цель этой книги - сделать именно это для сетевых специалистов. Вы изучите Python в контексте конфигурации сети, администрирования, мониторинга и многого другого.

Если вам надоело входить в систему и набирать кучу команд для настройки сети, Python для вас. Если вам нужно быть уверенным, что конфигурация сети надежна и повторяема, Python для вас. Если вам нужно отслеживать в реальном времени, что происходит в сети, то, как вы догадались, Python для вас. Вы, вероятно, согласны с тем, что изучение навыков работы с программным обеспечением может быть применено в сетевой инженерии. В конце концов, такие термины, как программно-определяемая сеть (SDN), были на слуху в последние несколько лет. Но почему именно Python? Может быть, вам стоит изучать JavaScript, Go или какой-нибудь другой язык? А может быть, вам стоит уделить больше внимания Bash и shell-сценариям.

Python хорошо подходит для сетевой инженерии по двум причинам.

Во-первых, как Эрик продемонстрирует в этой книге, существует множество библиотек Python (иногда называемых пакетами), разработанных специально для сетевой инженерии. Целенаправленный поиск на сайте https://pypi.org по теме сети позволяет найти более 500 различных библиотек для автоматизации и мониторинга сети. С помощью таких библиотек, как Ansible, вы можете создавать сложные конфигурации сети и серверов декларативно, используя простые конфигурационные файлы.

Используя Pexpect или Paramiko, вы сможете программировать удаленные устаревшие системы так, как если бы у них был собственный API сценариев. Если у устройства, которое вы настраиваете, есть API, есть вероятность, что для работы с ним вы сможете использовать специально созданную библиотеку Python. Таким образом, очевидно, что Python хорошо подходит для этой работы.

Во-вторых, Python занимает особое место среди языков программирования. Python - это то, что я называю языком полного спектра. Я определяю его так: это одновременно и язык, который невероятно прост для начала работы (print("hello world") anyone?), и очень мощный, являющийся технологией, стоящей за таким невероятным программным обеспечением, как youtube.com. Это не нормально. У нас есть надежные языки для начинающих, позволяющие быстро создавать программное обеспечение. На ум приходит Visual Basic. Также как и MATLAB и другие коммерческие языки. Тем не менее, когда они заходят слишком далеко, они сильно падают. Можете ли вы представить себе Linux, Firefox или интенсивную видеоигру, созданную с помощью любого из этих языков? Ни в коем случае.

На другом конце спектра мы также имеем очень мощные языки, такие как C++, .NET, Java и многие другие. C++, по сути, является языком, используемым для создания некоторых модулей ядра Linux и крупных программ с открытым исходным кодом, таких как Firefox. Тем не менее, эти языки не являются дружественными для новичков. Вам придется изучить указатели, компиляторы, компоновщики, заголовки, классы, доступность (публичная/частная) и т.д., чтобы просто начать.

Python живет в двух сферах. С одной стороны, на нем невероятно легко работать, используя всего несколько строк кода и простые концепции программирования. С другой стороны, он все чаще становится языком выбора для некоторых наиболее значимых программ в мире, таких как YouTube, Instagram, Reddit и других. Microsoft выбрала Python в качестве языка для реализации интерфейса командной строки (CLI) для Azure (хотя, конечно, для использования CLI не обязательно знать или использовать Python).

Итак, дело вот в чем. Программирование - это суперсила. Оно может поднять ваши знания в области сетевого проектирования в стратосферу. Python - один из самых быстрорастущих и популярных языков программирования в мире. Кроме того, в Python есть множество отшлифованных библиотек для работы с сетями во многих аспектах. Эта книга, Mastering Python Networking, Third Edition, сочетает в себе все это и изменит ваше представление о сетевых технологиях. Наслаждайтесь путешествием.

Michael Kennedy Portland, OR Founder, Talk Python

Введение

В 2014 году я провел первый семинар Coding 101 по Python и REST API для сетевых инженеров в зоне DevNet на Cisco Live. Зал был полон выдающихся сетевых инженеров и архитекторов, и многие из них сделали свой первый вызов API на этом семинаре. С тех пор я имею честь работать с сетевыми инженерами со всего мира, которые решили добавить кодирование в свой набор навыков. ИТ- и операционные команды меняются. Я считаю, что новым нормальным явлением станет работа сетевых инженеров и разработчиков программного обеспечения бок о бок в одной команде. Масштабируемость, сложность и безопасность, которые современные приложения требуют от сети, требуют автоматизации, чтобы сделать управление сетью повторяемым, надежным и гибким в масштабе.

Сетевые инженеры - это специалисты по решению проблем с глубокими знаниями. Добавление Python, навыков автоматизации сети и API к набору инструментов сетевого инженера создает мощную комбинацию. Благодаря этим дополнительным слоям инженеры могут подходить к проблемам по-новому и решать новые типы задач. Mastering Python Networking, Third Edition - это ценный ресурс как для сетевых инженеров, которые хотят освоить навыки кодирования, так и для инженеров-программистов, которые хотят воспользоваться новыми возможностями программируемой инфраструктуры.

Один из вопросов, который я часто слышу от инженеров: "С чего мне начать?". Мой совет: начните с простого. Ищите проблемы, с которыми сталкивается ваша команда и которые "доступны только для чтения", и сосредоточьтесь на использовании автоматизации для устранения неполадок и сбора информации. Затем вы можете использовать автоматизацию для передачи собранной информации в системы тикетинга или чат-приложения, и вскоре вы начнете строить рабочий процесс. Такой безопасный старт, эволюция только для чтения, помогает командам обрести уверенность в автоматизации и познакомиться с инструментами.

В самом начале сосредоточьтесь на изучении основных навыков кодирования, которые помогут вам в каждом проекте, включая кодирование на Python и RESTful API, а также на использовании таких инструментов, как Git и GitHub, для управления исходным кодом и совместной работы с другими. Потратьте время на настройку среды разработки. Попробуйте различные редакторы кода и инструменты, такие как Postman и curl, для изучения API. Сформируйте твердое понимание того, как работать с JSON и XML. Начните изучать методологии разработки программного обеспечения, такие как разработка на основе тестирования (TDD), и основные принципы DevOps.

Mastering Python Networking, Third Edition - отличный ресурс для отработки этих навыков, поскольку он помогает изучить эти темы в контексте сетевых технологий. Книга начинается с использования Python

для базового взаимодействия с сетевыми устройствами с помощью CLI и API, затем поднимается вверх по стеку с универсальной основой автоматизации - все с точки зрения сетевых инженеров. Попутно Эрик приводит примеры на Python по сетевой безопасности, мониторингу и созданию собственных API с помощью фреймворка Flask. Эрик также знакомит с работой с сетями в облаке AWS и Azure, а также с распространенными инструментами DevOps, такими как Git, Jenkins и TDD. На протяжении всей книги Эрик объясняет темы, используя прагматичный, основанный на практике подход, который поможет вам применить эти концепции в своей работе.

DevOps и облачные технологии преобразуют нашу отрасль. Команды разработчиков, операторов, специалистов по безопасности и сетевых инженеров объединяются новыми способами с общими целями и обязанностями для достижения бизнес-результатов. Сетевые инженеры, овладевшие навыками работы с программным обеспечением, помогут определить эту трансформацию.

Итак, найдите свой первый проект и работайте над освоением этих навыков в рамках этого проекта. Выберите что-то, что вы делаете каждый день, что вы хотите повторять надежно, и попытайтесь автоматизировать это. Возьмитесь за дело и пишите код рано и часто. Создайте или найдите лабораторию разработки, где вы сможете работать. Чем больше вы сможете экспериментировать, тем быстрее вы научитесь.

Инновации, которые будут созданы в течение следующих пяти лет совместной работой сетевых инженеров и разработчиков программного обеспечения, станут революционными. Сделайте свой первый шаг и не забудьте отпраздновать, когда получите свой первый успешный ответ API 200 OK.

Счастливого кодирования!

Mandy Whaley Senior Director of Developer Experience, Cisco DevNet

Авторы

Об авторе

Эрик Чоу - опытный технолог с более чем 20-летним стажем. Он работал в одних из крупнейших сетей в отрасли, работая в Amazon, Azure и других компаниях из списка Fortune 500. Эрик увлекается автоматизацией сетей, Python и помогает компаниям создавать более эффективные системы безопасности.

Помимо того, что он является автором книги Mastering Python Networking (Packt), он также является соавтором книги Distributed Denial of Service (DDoS): Практическое обнаружение и защита (O'Reilly Media).

Эрик также является основным автором двух патентов США в области IP-телефонии. Он делится своим глубоким интересом к технологиям через свои книги, занятия и блог, а также вносит вклад в некоторые популярные проекты с открытым исходным кодом Python.

Я хотел бы поблагодарить членов и разработчиков сообщества разработчиков открытого кода, сетевых технологий и Python за то, что они щедро делились своим сочувствием, знаниями и кодом. Без них многие проекты, упомянутые в этой книге, были бы невозможны. Надеюсь, что и я по-своему внес небольшой вклад в эти замечательные сообщества.

Я хотел бы поблагодарить команду Packt, Тушара, Тома, Яна, Алекса, Джона и многих других, за возможность сотрудничать в работе над третьим изданием книги. Особая благодарность техническому

рецензенту Рикарду Кёркко за великодушное согласие рецензировать книгу.

Спасибо вам, Мэнди и Майкл, за написание предисловий к этой книге. Я не могу выразить вам свою признательность. Ребята, вы просто супер! Моим родителям и семье, ваша постоянная поддержка и ободрение сделали меня тем, кто я есть, я люблю вас.

О рецензенте

Рикард Кёркко - консультант по NetDevOps в компании SDNit (https://sdnit.se), где он является частью группы опытных технических специалистов, проявляющих большой интерес к развивающимся сетевым технологиям и уделяющих им особое внимание.

Он программист-самоучка с основным упором на Python. Его повседневная работа включает в себя работу с инструментами оркестровки, такими как Ansible, для управления сетевыми устройствами.

Он также выступал в качестве технического рецензента книги "Практическое руководство по командам, редакторам и программированию оболочки Linux", третье издание, автор Марк Г. Собелл.

Предисловие

Как писал Чарльз Диккенс в "Повести о двух городах", "Это были лучшие времена, это были худшие времена, это был век мудрости, это был век глупости". Эти, казалось бы, противоречивые слова прекрасно описывают хаос и настроение, ощущаемые в период перемен и перехода. Мы, несомненно, переживаем похожее время в связи с быстрыми изменениями в области сетевой инженерии. Поскольку разработка программного обеспечения становится все более интегрированной во все аспекты сетевых технологий, традиционный интерфейс командной строки и вертикально интегрированные методы сетевого стека больше не являются лучшими способами управления современными сетями. Для сетевых инженеров изменения, которые мы наблюдаем, полны воодушевления и возможностей, но в то же время являются сложными, особенно для тех, кто должен быстро адаптироваться и идти в ногу со временем. Эта книга написана для того, чтобы облегчить переход для специалистов по сетевым технологиям, предоставляя практическое руководство, в котором рассматриваются способы перехода от традиционной платформы к платформе, построенной на программно-ориентированных методах.

В этой книге мы используем Python в качестве языка программирования для решения задач сетевого проектирования. Python - это простой в изучении язык программирования высокого уровня, который может эффективно дополнить творческий потенциал сетевых инженеров и их навыки решения проблем для оптимизации повседневной работы. Python становится неотъемлемой частью многих крупномасштабных сетей, и в этой книге я надеюсь поделиться с вами полученными уроками.

После публикации первого и второго изданий этой книги мне удалось провести интересные и содержательные беседы со многими из читателей книги.

Я смирился с успехом первых двух изданий и принял близко к сердцу отзывы, которые мне дали. В этом третьем издании я постарался включить многие из новых библиотек, обновить существующие примеры с учетом новейшего программного обеспечения и новых аппаратных платформ, а также добавил еще две главы, которые, по моему мнению, важны для современных сетевых инженеров.

Время перемен открывает большие возможности для технологического прогресса. Концепции и инструменты, приведенные в этой книге, очень помогли мне в моей карьере, и я надеюсь, что они могут сделать то же самое для вас.

Для кого эта книга

Эта книга идеально подходит для ИТ-специалистов и инженеров по эксплуатации, которые уже управляют группами сетевых устройств и хотели бы расширить свои знания по использованию Python и других инструментов для решения сетевых проблем. Рекомендуется базовое знание сетевых технологий и языка Python.

О чем рассказывает эта книга

В главе 1 "Обзор набора протоколов TCP/IP и языка Python" рассматриваются фундаментальные технологии, составляющие современные интернет-коммуникации, начиная с модели OSI и модели клиент-сервер и заканчивая наборами протоколов TCP, UDP и IP. В главе рассматриваются основы языка Python, такие как типы, операторы, циклы, функции и пакеты.

В главе 2 "Низкоуровневое взаимодействие с сетевыми устройствами" на практических примерах показано, как использовать Python для выполнения команд на сетевом устройстве. В ней также обсуждаются проблемы, связанные с наличием в автоматизации только интерфейса CLI. В главе для примеров будут использованы библиотеки Pexpect, Paramiko, Netmiko и Nornir.

В главе 3 "API и сети, управляемые намерениями" рассматриваются новые сетевые устройства, поддерживающие интерфейсы прикладного программирования (API) и другие высокоуровневые методы взаимодействия. В ней также показаны инструменты, которые позволяют абстрагироваться от низкоуровневых задач, фокусируясь на намерениях сетевых инженеров. В главе будут приведены обсуждение и примеры Cisco NX-API, Meraki, Juniper PyEZ, Arista Pyeapi и Vyatta VyOS.

В главе 4, "Система автоматизации на Python - основы Ansible", рассматриваются основы Ansible, системы автоматизации с открытым исходным кодом на базе Python. Ansible отходит на шаг от API и фокусируется на декларативном определении задач. В этой главе мы рассмотрим преимущества использования Ansible и его высокоуровневой архитектуры, а также увидим несколько практических примеров использования Ansible с устройствами Cisco, Juniper и Arista.

Глава 5 "Python Automation Framework - Beyond Basics" опирается на знания, полученные в предыдущей главе, и охватывает более сложные темы Ansible. Мы рассмотрим условия, циклы, шаблоны, переменные, Ansible Vault и роли. Также будут рассмотрены основы написания пользовательских модулей.

Глава 6 "Безопасность сети с помощью Python" знакомит с несколькими инструментами Python, которые помогут вам защитить вашу сеть. В ней будет обсуждаться использование Scapy для тестирования безопасности, использование Ansible для быстрого внедрения списков доступа, а также использование Python для анализа сетевой криминалистики.

В главе 7 "Мониторинг сети с помощью Python - часть 1" рассматривается мониторинг сети с помощью различных инструментов. Глава содержит несколько примеров использования SNMP и PySNMP для запросов с целью получения информации об устройствах. Будут показаны примеры Matplotlib и Pygal для построения графиков результатов. В конце главы приводится пример Cacti с использованием сценария Python в качестве источника входных данных.

В главе 8 "Мониторинг сети с помощью Python - часть 2" рассматриваются дополнительные инструменты мониторинга сети. Глава начнется с использования Graphviz для построения графика сети на основе информации LLDP. Мы перейдем к примерам с мониторингом сети на основе push, используя Netflow и другие технологии. Мы будем использовать Python для декодирования пакетов потока и ntop для визуализации результатов. Также будет сделан обзор Elasticsearch и того, как его можно

использовать для мониторинга сети.

В главе 9 "Создание сетевых веб-сервисов с помощью Python" показано, как использовать веб-фреймворк Python Flask для создания собственного API для автоматизации сети. Сетевой API предлагает такие преимущества, как абстрагирование запрашивающего от деталей сети, консолидация и настройка операций, а также обеспечение лучшей безопасности за счет ограничения доступных операций.

В главе 10, Облачная сеть AWS, показано, как с помощью AWS создать функциональную и отказоустойчивую виртуальную сеть. Мы рассмотрим технологии виртуального частного облака, такие как CloudFormation, таблицы маршрутизации VPC, списки доступа, Elastic IP, шлюзы NAT, Direct Connect и другие связанные темы.

В главе 11, Облачные сетевые службы Azure, рассказывается о сетевых службах Azure и о том, как создавать сетевые службы с помощью этой службы. Мы обсудим Azure VNet, Express Route и VPN, сетевые балансировщики нагрузки Azure и другие связанные с ними сетевые службы.

Глава 12 "Анализ сетевых данных с помощью Elastic Stack" показывает, как мы можем использовать Elastic Stack в качестве набора тесно интегрированных инструментов для анализа и мониторинга нашей сети. Мы рассмотрим различные области, начиная с установки, настройки, импорта данных с помощью Logstash и Beats, поиска данных с помощью Elasticsearch и заканчивая визуализацией с помощью Kibana.

В главе 13, Работа с Git, мы проиллюстрируем, как можно использовать Git для совместной работы и контроля версий кода. В этой главе будут использованы практические примеры использования Git для работы в сети.

В главе 14 "Непрерывная интеграция с Jenkins" используется Jenkins для автоматического создания конвейеров операций, которые могут сэкономить нам время и повысить надежность.

В главе 15 "Тестово-ориентированная разработка для сетей" объясняется, как использовать Python's unittest и pytest для создания простых тестов для проверки нашего кода. Мы также увидим примеры написания тестов для нашей сети для проверки достижимости, задержки сети, безопасности и сетевых транзакций. Мы также увидим, как можно интегрировать тесты в инструменты непрерывной интеграции, такие как Jenkins.

Чтобы получить максимальную пользу от этой книги

Чтобы извлечь максимальную пользу из этой книги, рекомендуется иметь некоторые базовые практические знания по работе с сетью и знание языка Python. Большинство глав можно читать в любом порядке, за исключением глав 4 и 5, которые следует читать по порядку. Помимо основных программных и аппаратных инструментов, представленных в начале книги, в соответствующих главах будут представлены новые инструменты, относящиеся к каждой из глав.

Настоятельно рекомендуется следовать и практиковать показанные примеры в собственной сетевой лаборатории.

Глава I

Обзор набора протоколов TCP/IP и Python

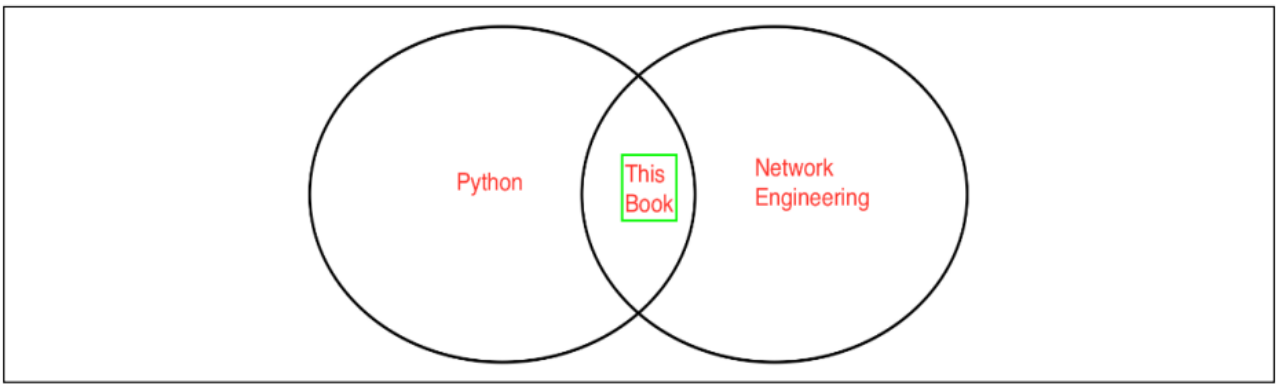
Добро пожаловать в новую захватывающую эпоху сетевого инжиниринга! Когда я начал работать сетевым инженером на рубеже тысячелетий, эта роль заметно отличалась от сегодняшней роли

сетевого инженера. В то время сетевые инженеры в основном обладали специфическими знаниями для управления и эксплуатации локальных и глобальных сетей с помощью интерфейса командной строки. Хотя они могли иногда пересекать стены дисциплины, чтобы решать задачи, обычно связанные с системным администрированием и разработчиками, не было явных ожиданий, что сетевой инженер будет писать код или понимать концепции программирования. Сегодня это уже не так.

С течением времени движение DevOps и Software-Defined Networking (SDN), среди прочих факторов, значительно размыли границы между сетевыми инженерами, системными инженерами и разработчиками.

Тот факт, что вы взяли в руки эту книгу, говорит о том, что вы, возможно, уже являетесь приверженцем сетевого DevOps, или, возможно, вы рассматриваете возможность пойти по этому пути и проверить сетевую программируемость. Возможно, вы, как и я, много лет проработали сетевым инженером и хотели узнать, чем вызвана шумиха вокруг языка программирования Python. Возможно, вы даже уже свободно владеете языком программирования Python, но вам интересно, какое применение он находит в области сетевой инженерии.

Если вы относитесь к одному из этих лагерей или просто интересуетесь Python в области сетевой инженерии, я считаю, что эта книга для вас:



Уже написано много книг, в которых отдельно рассматриваются темы сетевой инженерии и Python. Я не намерен повторять их усилия в этой книге. Вместо этого, эта книга предполагает, что у вас есть некоторый практический опыт управления сетями, а также базовое понимание сетевых протоколов. Будет полезно, если вы уже знакомы с Python как языком, но некоторые основы мы рассмотрим позже в этой главе. Вам не нужно быть экспертом в Python или сетевой инженерии, чтобы получить максимальную пользу от этой книги. Цель этой книги - опираясь на базовые основы сетевой инженерии и Python, помочь читателям изучить и применить на практике различные приложения, которые могут облегчить их жизнь.

В этой главе мы проведем общий обзор некоторых концепций сетевых технологий и Python. Остальная часть главы должна определить уровень ожидаемых предварительных знаний, необходимых для получения максимальной пользы от этой книги. Если вы хотите подтянуть содержание этой главы, существует множество бесплатных или недорогих ресурсов, которые помогут вам войти в курс дела. Я бы рекомендовал бесплатную Академию Хана (https://www. khanacademy.org/) и учебники по Python на сайте https://www.python.org/.

В этой главе мы очень быстро рассмотрим соответствующие сетевые темы на высоком уровне, не слишком углубляясь в детали. Судя по моему опыту работы в этой области, типичный сетевой инженер или разработчик может не помнить точную машину состояния протокола Transmission Control Protocol (TCP) для выполнения своих ежедневных задач (я знаю, что не помню), но они будут знакомы с основами модели Open Systems Interconnection (OSI), операциями TCP и User Datagram Protocol (UDP),

различными полями заголовков IP и другими фундаментальными концепциями.

Мы также рассмотрим высокоуровневый обзор языка Python; этого достаточно для тех читателей, которые не занимаются кодом на Python ежедневно, чтобы иметь почву для дальнейшего изучения книги.

В частности, мы рассмотрим следующие темы:

- Обзор интернета
- Модель OSI и клиент-сервер
- Наборы протоколов TCP, UDP и IP
- Синтаксис Python, типы, операторы и циклы
- Расширение Python с помощью функций, классов и пакетов.

Конечно, информация, представленная в этой главе, не является исчерпывающей; при необходимости ознакомьтесь с дополнительной информацией в справочной литературе.

Как сетевые инженеры, мы обычно сталкиваемся с проблемой масштаба и сложности сетей, которыми нам приходится управлять. Они варьируются от небольших домашних сетей, сетей среднего размера, обеспечивающих работу малого бизнеса, до крупных сетей многонациональных предприятий, охватывающих весь земной шар. Самая большая сеть из всех - это, конечно же, Интернет. Без Интернета не было бы электронной почты, веб-сайтов, API, потокового мультимедиа и облачных вычислений в том виде, в котором мы их знаем. Поэтому, прежде чем мы углубимся в специфику протоколов и Python, давайте начнем с обзора Интернета.

Обзор интернета

Что такое интернет? На этот, казалось бы, простой вопрос можно получить разные ответы в зависимости от вашего образования. Интернет означает разные вещи для разных людей; молодые, пожилые, студенты, учителя, бизнесмены, поэты - все они могут дать разные ответы на один и тот же вопрос.

Для сетевого инженера Интернет - это глобальная компьютерная сеть, состоящая из паутины межсетевых соединений, объединяющих большие и малые сети. Другими словами, это сеть сетей без централизованного владельца. Возьмем для примера вашу домашнюю сеть. Она может состоять из устройства, которое объединяет функции маршрутизации, коммутации Ethernet и беспроводных точек доступа, соединяющих вместе ваши смартфоны, планшеты, компьютеры и телевизоры с выходом в Интернет, чтобы устройства могли общаться друг с другом. Это и есть ваша локальная вычислительная сеть (LAN).

Когда вашей домашней сети требуется связь с внешним миром, она передает информацию из вашей локальной сети в более крупную сеть, которую часто называют интернет-провайдером (ISP). Провайдер обычно рассматривается как предприятие, которому вы платите за доступ в Интернет. Для этого они объединяют небольшие сети в более крупные сети, которые они обслуживают. Сеть провайдера часто состоит из граничных узлов, которые объединяют трафик в основную сеть. Функция основной сети заключается в соединении этих граничных сетей через более высокоскоростную сеть.

На специальных пограничных узлах ваш провайдер подключается к другим провайдерам, чтобы передать ваш трафик соответствующим образом к месту назначения. Обратный путь от места назначения до вашего домашнего компьютера, планшета или смартфона может проходить по одному и тому же маршруту через все эти сети обратно к вашему устройству, хотя источник и место назначения остаются неизменными.

Давайте посмотрим на компоненты, составляющие эту паутину сетей.

Серверы, узлы и сетевые компоненты

Хосты - это конечные узлы сети, которые взаимодействуют с другими узлами. В современном мире узлом может быть традиционный компьютер, а может быть смартфон, планшет или телевизор. С развитием Интернета вещей (IoT) широкое определение узла может быть расширено до камеры Интернет-протокола (IP), телевизионных приставок и постоянно растущего числа типов датчиков, которые мы используем в сельском хозяйстве, фермерстве, автомобилях и т.д. С ростом числа узлов, подключенных к Интернету, все они нуждаются в адресации, маршрутизации и управлении. Потребность в надлежащем сетевом взаимодействии никогда не была столь велика.

Чаще всего, когда мы находимся в Интернете, мы делаем запросы на услуги.

Это может быть просмотр веб-страницы, отправка или получение электронной почты, передача файлов и так далее. Эти услуги предоставляются серверами. Как следует из названия, серверы предоставляют услуги нескольким узлам и обычно имеют более высокий уровень аппаратной спецификации. В некотором смысле, серверы - это специальные суперузлы сети, которые предоставляют дополнительные возможности своим коллегам. Мы рассмотрим серверы позже в разделе модели клиент-сервер. Если представить серверы и узлы как города и поселки, то сетевые компоненты - это дороги и шоссе, которые соединяют их вместе. Фактически, термин "информационная супермагистраль" приходит на ум при описании сетевых компонентов, которые передают постоянно увеличивающиеся биты и байты по всему миру. В модели OSI, которую мы рассмотрим чуть позже, эти сетевые компоненты являются устройствами первого-третьего уровней, которые иногда проникают и на четвертый уровень. Это маршрутизаторы и коммутаторы второго и третьего уровней, которые направляют трафик, а также транспортные средства первого уровня, такие как оптоволоконные кабели, коаксиальные кабели, витые медные пары и некоторые виды оборудования для мультиплексирования с плотным разделением по длине волны (DWDM).

В совокупности хосты, серверы, хранилища и сетевые компоненты составляют интернет, каким мы его знаем сегодня.

Возникновение центров обработки данных

В предыдущем разделе мы рассмотрели различные роли, которые играют серверы, хосты и сетевые компоненты в межсетевом взаимодействии. Из-за более высокой аппаратной мощности, которая требуется серверам, их часто собирают в центральном месте для более эффективного управления. Мы часто называем эти места центрами обработки данных.

Корпоративные центры обработки данных

На типичном предприятии компания обычно имеет деловые потребности во внутренних инструментах, таких как электронная почта, хранение документов, отслеживание продаж, оформление заказов, инструменты управления персоналом и интранет для обмена знаниями. Эти услуги воплощаются в файловых и почтовых серверах, серверах баз данных и веб-серверах. В отличие от пользовательских компьютеров, это, как правило, компьютеры высокого класса, требующие большой мощности, охлаждения и сетевых соединений.

Побочным продуктом аппаратного обеспечения также является производимый ими шум, который не подходит для обычного рабочего пространства. Серверы обычно размещаются в центральном месте, называемом главной распределительной рамой (MDF), в здании предприятия для обеспечения необходимого питания, резервирования питания, охлаждения и подключения к сети.

Для подключения к MDF трафик пользователя обычно агрегируется в более близком к пользователю месте, которое иногда называют промежуточной распределительной рамкой (IDF), прежде чем он объединяется в пучок и подключается к MDF. Нередко распределение IDF-MDF соответствует физическому расположению здания предприятия или кампуса. Например, каждый этаж здания может состоять из IDF, который агрегируется с централизованным MDF на другом этаже того же здания. Если предприятие состоит из нескольких зданий, дальнейшая агрегация может быть выполнена путем объединения трафика зданий перед их подключением к центру обработки данных предприятия.

Корпоративные центры обработки данных, как правило, придерживаются трехуровневой схемы построения сети. Этими уровнями являются уровни доступа, распределения и основной уровень. Конечно, как и в любой другой схеме, здесь нет жестких правил или универсальной модели; трехслойные схемы являются лишь общим руководством. В качестве примера, чтобы наложить трехслойную конструкцию на наш предыдущий пример User-IDF-MDF, уровень доступа аналогичен портам, к которым подключается каждый пользователь, IDF можно представить как уровень распределения, а основной уровень состоит из соединения с MDF и центрами обработки данных предприятия. Конечно, это обобщение корпоративных сетей, поскольку некоторые из них не будут следовать той же модели.

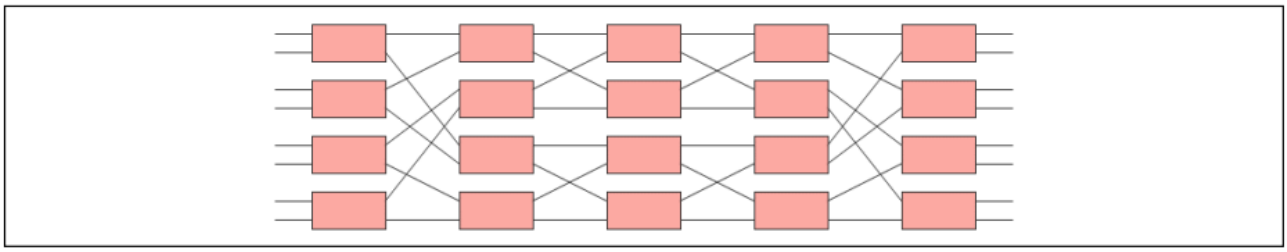
Облачные центры обработки данных

С развитием облачных вычислений и программного обеспечения, или инфраструктуры как услуги (IaaS), центры обработки данных, построенные провайдерами облачных вычислений, стали очень большими по масштабу, иногда их называют гипермасштабными центрами обработки данных. То, что мы называем облачными вычислениями, - это доступность вычислительных ресурсов по требованию, предлагаемых такими компаниями, как Amazon, Microsoft и Google, без непосредственного управления ресурсами со стороны пользователя.

Из-за количества серверов, которые им необходимо разместить, облачные центры обработки данных обычно требуют гораздо, гораздо больше мощности электропитания, охлаждения и сетевых мощностей, чем любой корпоративный центр обработки данных. Даже после многолетней работы над центрами обработки данных облачных провайдеров, каждый раз, когда я посещаю центр обработки данных облачного провайдера, я все еще поражаюсь их масштабам. В качестве примера можно привести облачные центры обработки данных, которые настолько велики и требовательны к электроэнергии, что их обычно строят рядом с электростанциями, где можно получить самую дешевую электроэнергию, не теряя при этом эффективности при ее транспортировке. Их потребности в охлаждении настолько высоки, что некоторые компании вынуждены творчески подходить к выбору места строительства дата-центра. Например, компания Facebook построила свой центр обработки данных Lulea на севере Швеции (всего в 70 милях к югу от полярного круга), чтобы использовать холодную температуру для охлаждения. Любая поисковая система может предоставить вам некоторые поразительные цифры, когда речь идет о науке строительства и управления облачными центрами обработки данных для таких компаний, как Amazon, Microsoft, Google и Facebook. Например, дата-центр Microsoft в Западном Де-Мойне, штат Айова, состоит из 1,2 миллиона квадратных футов на 200 акрах земли и потребовал от города примерно 65 миллионов долларов на модернизацию общественной инфраструктуры.

В масштабах облачных провайдеров услуги, которые им необходимо предоставлять, как правило, нерентабельны или нецелесообразны для размещения на одном сервере. Услуги распределяются между целым парком серверов, иногда в разных стойках, чтобы обеспечить избыточность и гибкость для владельцев услуг.

Требования к задержкам и избыточности, а также физическое распределение серверов оказывают огромную нагрузку на сеть. Количество соединений, необходимых для подключения серверного парка, приводит к взрывному росту сетевого оборудования, такого как кабели, коммутаторы и маршрутизаторы. Эти требования выражаются в том, что сетевое оборудование необходимо устанавливать в стойки, обеспечивать его работоспособность и управлять им. Типичный сетевой дизайн представляет собой многоступенчатую сеть Clos:



В некотором смысле, облачные центры обработки данных - это место, где автоматизация сети становится необходимостью для обеспечения скорости, гибкости и надежности. Если следовать традиционному способу управления сетевыми устройствами через терминал и интерфейс командной строки, то количество необходимых инженерных часов не позволит обеспечить доступность услуги в разумные сроки.

И это не говоря уже о том, что повторение действий человека чревато ошибками, неэффективно и является ужасной тратой инженерного таланта. Дополнительной сложностью является то, что часто возникает необходимость быстро изменить некоторые конфигурации сети для удовлетворения быстро меняющихся потребностей бизнеса.

Лично я начал свой путь автоматизации сети с помощью Python именно с облачных центров обработки данных несколько лет назад, и с тех пор я никогда не оглядывался назад.

Пограничные центры обработки данных

Если у нас достаточно вычислительных мощностей на уровне центров обработки данных, зачем хранить что-то еще где-то, кроме этих центров обработки данных? Все соединения от клиентов по всему миру можно направить обратно на серверы центра обработки данных, предоставляющие услугу, и на этом можно закончить, верно? Ответ, конечно, зависит от конкретного случая использования. Самым

большим ограничением при маршрутизации запроса и сессии от клиента до крупного центра обработки данных является задержка, возникающая при транспортировке. Другими словами, большая задержка - это то место, где сеть становится узким местом.

Конечно, любой учебник по элементарной физике скажет вам, что число задержек в сети никогда не будет равно нулю: даже если свет движется в вакууме так быстро, как это возможно, ему все равно требуется время для физической транспортировки. В реальном мире задержка будет гораздо выше, чем скорость света в вакууме. Почему? Потому что сетевой пакет должен пройти через несколько сетей, иногда через подводный кабель, медленные спутниковые каналы, сотовую связь 3G или 4G или Wi-Fi.

Что если нам нужно уменьшить сетевую задержку? Одним из решений может быть уменьшение количества сетей, через которые проходят запросы конечных пользователей. Будьте как можно ближе к конечному пользователю на границе, где пользователь входит в вашу сеть, и разместите на границе достаточно ресурсов для обслуживания запроса. Это особенно характерно для обслуживания медиаконтента, такого как музыка и видео.

Давайте на минуту представим, что вы создаете следующее поколение услуг потокового видео. Чтобы повысить удовлетворенность клиентов бесперебойной потоковой передачей, вы захотите разместить видеосервер как можно ближе к клиенту, либо внутри, либо очень близко к интернет-провайдеру клиента. Кроме того, для обеспечения избыточности и скорости соединения, восходящий поток фермы видеосерверов будет подключен не просто к одному или двум провайдерам, а ко всем провайдерам, к которым я могу подключиться, чтобы уменьшить количество хопов. Все соединения будут иметь такую пропускную способность, которая необходима для снижения задержек в часы пик. Эта потребность привела к появлению пограничных центров обработки данных пиринговых бирж крупных провайдеров интернет-услуг и поставщиков контента. Даже если количество сетевых устройств не так велико, как в облачных центрах обработки данных, они тоже могут извлечь выгоду из автоматизации сети с точки зрения повышения надежности, гибкости, безопасности и видимости, которые обеспечивает автоматизация сети.

Мы рассмотрим вопросы безопасности (глава 6, Безопасность сети с Python) и видимости (глава 7, Мониторинг сети с Python - часть 1, и глава 8, Мониторинг сети с Python - часть 2) в последующих главах этой книги. Как и многие сложные предметы, сложность которых разбивается путем разделения предмета на более мелкие, легко усваиваемые части, сетевые технологии основаны на концепции уровней. За многие годы были разработаны различные сетевые модели. В этой книге мы рассмотрим две наиболее важные модели, начиная с модели OSI.

Модель OSI

Ни одна книга по сетевым технологиям не будет полной без рассмотрения модели OSI. Эта модель представляет собой концептуальную модель, которая компонует телекоммуникационные функции в различные уровни. Модель определяет семь уровней, и каждый из них располагается независимо друг от друга с определенными структурами и характеристиками.

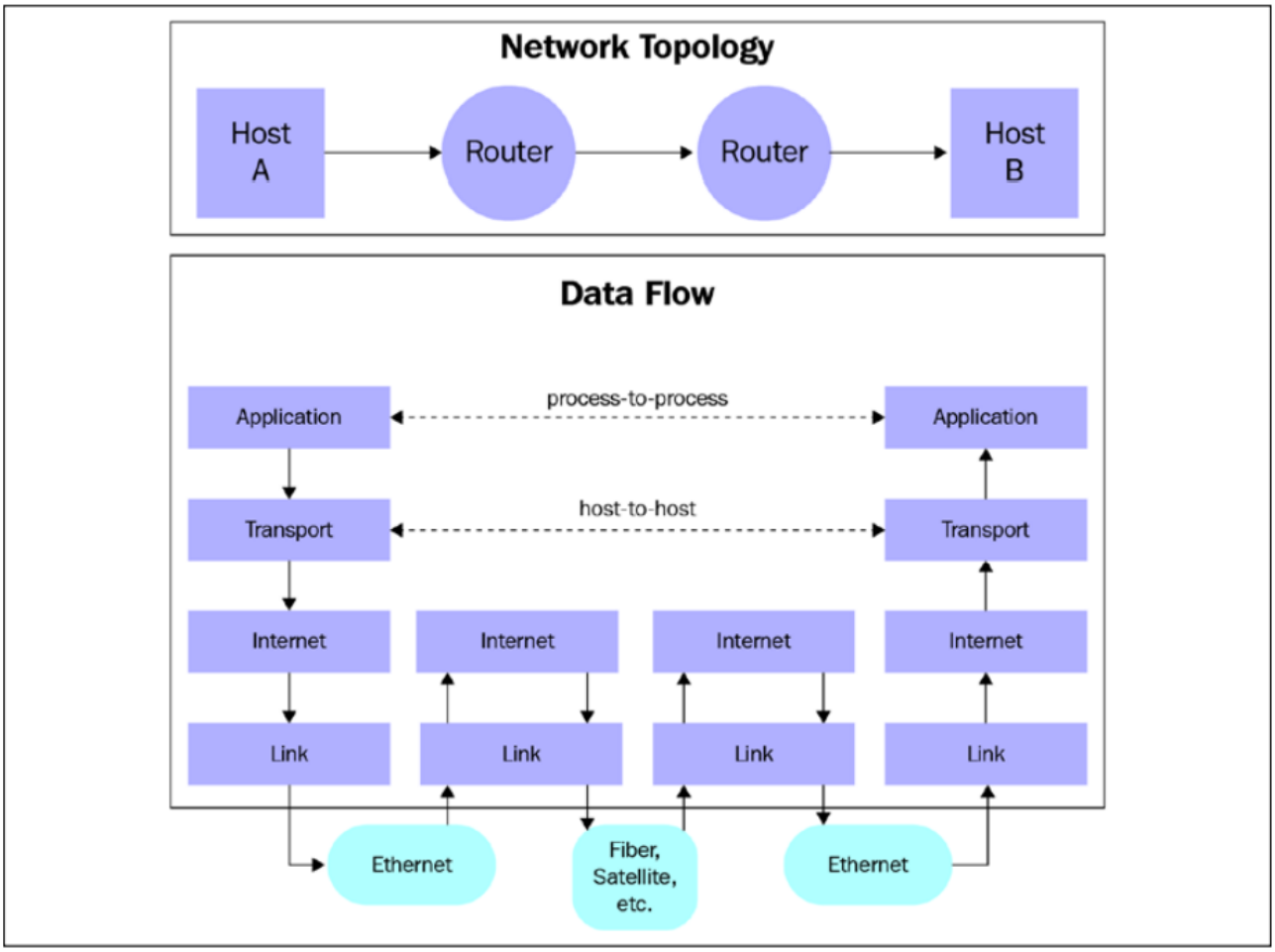
Например, на сетевом уровне IP располагается поверх различных типов канальных уровней, таких как Ethernet или frame relay. Эталонная модель OSI - это хороший способ нормализовать различные и разнообразные технологии в набор общих языков, которые люди могут согласовать. Это значительно сокращает возможности сторон, работающих над отдельными уровнями, и позволяет им углубленно рассматривать конкретные задачи, не слишком беспокоясь о совместимости:

OSI Model			
Layer		Protocol data unit (PDU)	Function
Host Layers	7. Application	Data	High-level APIs, including resource sharing, remote file access
	6. Presentation		Translation of data between a networking service and an application; including character encoding, data compression, and encryption / decryption
	5. Session		Managing communication sessions, i.e. continuous exchange of information in the form of mutiple back-and-forth transmissions between two nodes
	4. Transport	Segment (TCP) /Datagram (UDP)	Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing
Media Layers	3. Network	Packet	Structuring and managing a multi-node network, including addressing, routing and traffic control
	2. Data link	Frame	Reliable transmission of data frames between two nodes connected by a physical layer
	1. Physical	Bit	Transmission and reception of raw bit streams over a physical medium

Модель OSI была первоначально разработана в конце 1970-х годов и позже опубликована совместно Международной организацией по стандартизации (ISO), которая сегодня известна как Сектор стандартизации электросвязи Международного союза электросвязи (ITU-T). Она широко признана и на нее часто ссылаются при введении новой темы в области телекоммуникаций.

Примерно в то же время, когда разрабатывалась модель OSI, формировался интернет. Эталонную модель, которую использовал первоначальный разработчик, часто называют моделью TCP/IP. TCP и IP были первоначальными наборами протоколов, содержащимися в этой модели. Это в некоторой степени похоже на модель OSI в том смысле, что они разделяют сквозную передачу данных на уровни абстракции

Отличием модели является то, что она объединяет уровни с 5 по 7 в модели OSI в прикладной уровень, а физический и канальный уровни объединяются в канальный уровень:



Обе модели - OSI и TCP/IP - полезны для обеспечения стандартов сквозной передачи данных. Однако в основном в этой книге мы будем чаще обращаться к модели TCP/IP, поскольку именно на ней изначально был построен интернет. Мы будем обращаться к модели OSI по мере необходимости, например, при обсуждении веб-фреймворка в последующих главах. Подобно моделям на транспортном уровне, существуют также эталонные модели, регулирующие коммуникацию на уровне приложений. В современной сети модель клиент-сервер - это то, на чем основано большинство приложений. Мы рассмотрим модель клиент-сервер в следующем разделе.

Модель клиент-сервер

Эталонные модели клиент-сервер демонстрировали стандартный способ передачи данных между двумя узлами. Конечно, сейчас мы все знаем, что не все узлы созданы одинаковыми. Даже в самые первые дни существования сети Агентства передовых исследовательских проектов (ARPANET) существовали узлы рабочих станций и узлы серверов, целью которых было предоставление контента другим узлам. Эти серверные узлы обычно имеют более высокие аппаратные характеристики и более тщательно управляются инженерами. Поскольку эти узлы предоставляют ресурсы и услуги другим, их правильно называть серверами. Серверы обычно простаивают, ожидая, пока клиенты инициируют запросы на их ресурсы. Эта модель распределенных ресурсов, которые запрашиваются по запросу клиента, называется моделью клиент-сервер.

Почему это важно? Если задуматься на минуту, то важность сетевого взаимодействия в значительной степени подчеркивается этой моделью клиент-сервер. Без необходимости передачи услуг между клиентами и серверами нет особой нужды в сетевых соединениях. Именно необходимость передачи битов и байтов от клиента к серверу проливает свет на важность сетевой инженерии. Конечно, мы все знаем о том, как самая большая из всех сетей - Интернет - изменила жизнь каждого из нас и продолжает это делать.

Вы можете спросить, как каждый узел может определить время, скорость, источник и пункт назначения каждый раз, когда ему нужно поговорить друг с другом? Это приводит нас к сетевым протоколам.

Пакеты сетевых протоколов

На заре развития компьютерных сетей протоколы были проприетарными и строго контролировались компанией, разработавшей метод соединения. Если вы использовали протокол IPX/SPX компании Novell на своих узлах, то эти же узлы не могли взаимодействовать с узлами Apple AppleTalk, и наоборот. Эти наборы проприетарных протоколов обычно имеют аналогичные уровни эталонной модели OSI и следуют методу связи клиент-сервер, но не совместимы друг с другом. Проприетарные протоколы обычно работают только в закрытых локальных сетях, не требующих связи с внешним миром. Когда трафик все же должен выходить за пределы локальной сети, обычно используется устройство интернет-трансляции, например маршрутизатор, для перевода с одного протокола на другой. Например, чтобы подключить сеть на базе AppleTalk к Интернету, используется маршрутизатор для подключения и преобразования протокола AppleTalk в сеть на базе IP. Дополнительный перевод обычно не идеален, но поскольку в первые годы большая часть коммуникаций происходила внутри локальной сети, администраторы сетей с этим смирились.

Однако по мере того, как потребность в межсетевом взаимодействии выходит за пределы локальной сети, возрастает необходимость в стандартизации наборов сетевых протоколов. Собственные протоколы в конечном итоге уступили место стандартизированным наборам протоколов TCP, UDP и IP, которые значительно расширили возможности одной сети общаться с другой. Интернет, величайшая из всех сетей, полагается на эти протоколы для правильного функционирования. В следующих нескольких разделах мы рассмотрим каждый из этих протоколов.

Протокол управления передачей

TCP - это один из основных протоколов, используемых сегодня в Интернете. Если вы открывали веб-страницу или отправляли электронное письмо, вы сталкивались с протоколом TCP. Этот протокол находится на четвертом уровне модели OSI и отвечает за доставку сегмента данных между двумя узлами надежным и проверенным на ошибки способом. TCP состоит из 160-битного заголовка, который содержит, среди прочего, порты источника и назначения, порядковый номер, номер подтверждения, управляющие флаги и контрольную сумму:

TCP Header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved				N S	C W R	E C E	U R G	A C K	P S H	R S T	S Y N	F I N	Window Size														
						000																											
16	128	Checksum																Urgent pointer (if URG set)															
20	160	Options (if data offset > 5. Padded at the end with "0" bytes if necessary.)																															
...	...	...																															

Figure 6: TCP header

Функции и характеристики TCP

TCP использует сокеты дейтаграмм или порты для установления связи между хостами. Стандартный орган, называемый Internet Assigned Numbers Authority (IANA), назначает известные порты для обозначения определенных служб, например, порт 80 для HTTP (веб) и порт 25 для SMTP (почта). Сервер в модели клиент-сервер обычно прослушивает один из этих хорошо известных портов, чтобы получать запросы на связь от клиента. TCP-соединение управляется операционной системой с помощью

сокета, который представляет собой локальную конечную точку для соединения.

Работа протокола состоит из машины состояний, где машина должна отслеживать, когда она слушает входящее соединение, во время сеанса связи, а также освобождать ресурсы после закрытия соединения. Каждое TCP-соединение проходит через ряд состояний, таких как Listen, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT и CLOSED.

TCP-сообщения и передача данных

Самое большое отличие TCP от UDP, который является его близким родственником на том же уровне, заключается в том, что он передает данные упорядоченным и надежным образом. Тот факт, что работа TCP гарантирует доставку, часто называют TCP протоколом, ориентированным на соединение. Для этого он сначала устанавливает трехстороннее рукопожатие для синхронизации порядкового номера между передатчиком и приемником, SYN, SYN-ACK и ACK. Подтверждение используется для отслеживания последующих сегментов в разговоре. Наконец, в конце разговора одна сторона посылает сообщение FIN, а другая сторона подтверждает сообщение FIN, а также посылает собственное сообщение FIN. Инициатор FIN затем подтвердит полученное сообщение FIN. Многие из нас, кто сталкивался с проблемами TCP-соединения, могут сказать вам, что эта операция может быть довольно сложной. Конечно, можно оценить, что большую часть времени эта операция просто тихо выполняется в фоновом режиме.

О протоколе TCP можно написать целую книгу; на самом деле, об этом протоколе написано много отличных книг.

Поскольку этот раздел является кратким обзором, если вас заинтересует, The TCP/IP Guide (http://www.tcpipguide.com/) - это отличный бесплатный ресурс, который вы можете использовать для более глубокого изучения темы.

Протокол пользовательских дейтаграмм

UDP также является одним из основных членов набора интернет-протоколов. Как и TCP, он работает на четвертом уровне модели OSI, который отвечает за доставку сегментов данных между приложением и уровнем IP. В отличие от TCP, заголовок составляет всего 64 бита и состоит только из порта источника и назначения, длины и контрольной суммы. Легкий заголовок делает его идеальным для приложений, которые предпочитают более быструю доставку данных, не устанавливая сеанс между двумя хостами и не нуждаясь в надежной доставке данных. Возможно, это трудно представить при современных скоростных интернет-соединениях, но дополнительный заголовок имел большое значение для скорости передачи данных в ранние времена X.21 и каналов frame relay.

Помимо разницы в скорости, отсутствие необходимости поддерживать различные состояния, такие как TCP, также экономит ресурсы компьютера на двух конечных точках:

UDP Header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Length																Checksum															

Figure 7: UDP header

Теперь вы можете задаться вопросом, почему UDP вообще используется в современную эпоху; учитывая отсутствие надежной передачи данных, разве мы не хотим, чтобы все соединения были надежными и безошибочными? Если вы подумаете о потоковой передаче мультимедийного видео или звонках по Skype, то эти приложения выигрывают от облегченного заголовка, когда приложение просто

хочет доставить дейтаграмму как можно быстрее. Можно также рассмотреть быстрый процесс поиска в системе доменных имен (DNS) на основе протокола UDP, где компромисс между точностью и задержкой обычно склоняется в сторону малой задержки.

Когда адрес, который вы набираете в браузере, переводится в понятный компьютеру адрес, пользователь выигрывает от облегчения процесса, поскольку это должно произойти еще до того, как первый бит информации будет доставлен вам с вашего любимого сайта.

Опять же, этот раздел не полностью раскрывает тему UDP, и читателю предлагается изучить эту тему с помощью различных ресурсов, если вы хотите узнать больше о UDP.

Статья Википедии о UDP, https://en.wikipedia.org/ wiki/User_Datagram_Protocol, является хорошей отправной точкой, чтобы узнать больше о UDP.

Интернет-протокол

Как вам скажут сетевые инженеры, мы живем на уровне IP, который является уровнем 3 в модели OSI. IP выполняет, помимо прочего, функции адресации и маршрутизации между конечными узлами. Адресация IP, вероятно, является его самой важной задачей. Адресное пространство делится на две части: сетевую и хостовую. Маска подсети используется для указания того, какая часть адреса сети состоит из сети, а какая - из хоста, путем сопоставления части сети с 1, а части хоста - с 0. IPv4 выражает адрес в точечной нотации, например, 192.168.0.1.

Маска подсети может быть либо в точечной нотации (255.255.255.0), либо использовать прямую косую черту для выражения количества битов, которые должны учитываться в сетевом бите (/24):

IPv4 Header Format

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags			Fragment Offset											
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															
24	192																																
28	224																																
32	256																																

Figure 8: IPv4 header

Заголовок IPv6, следующее поколение IP-заголовка IPv4, имеет фиксированную часть и различные заголовки расширения:

Fixed Header format

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	*Version*				*Traffic Class*								*Flow Label*																			
4	32	*Payload Length*																*Next Header*								*Hop Limit*							
8	64	*Source Address*																															
12	96																																
16	128																																
20	160																																
24	192	*Destination Address*																															
28	224																																
32	256																																
36	288																																

Figure 9: IPv6 header

Поле IPv6 Next Header в разделе фиксированного заголовка может указывать на последующий заголовок расширения, который несет дополнительную информацию. Оно также может идентифицировать протокол верхнего уровня, такой как TCP и UDP. Заголовки расширения могут

включать информацию о маршрутизации и фрагментах. Как бы ни хотелось разработчикам протоколов перейти от IPv4 к IPv6, сегодня Интернет по-прежнему адресуется в основном по протоколу IPv4, а некоторые сети поставщиков услуг адресуются по IPv6 внутри сети.

Трансляция сетевых адресов IP (NAT) и сетевая безопасность

NAT обычно используется для трансляции ряда частных адресов IPv4 в общедоступные маршрутизируемые адреса IPv4. Но он также может означать трансляцию между IPv4 и IPv6, например, на границе оператора, когда внутри сети используется IPv6, который необходимо трансформировать в IPv4, когда пакет покидает сеть. Иногда NAT6 на 6 также используется по соображениям безопасности.

Безопасность - это непрерывный процесс, объединяющий все аспекты работы в сети, включая автоматизацию и Python. Цель этой книги - использовать Python, чтобы помочь вам управлять сетью; вопросы безопасности будут рассмотрены в следующих главах книги, например, использование Python для реализации списков доступа, поиска нарушений в журнале и так далее. Мы также рассмотрим, как можно использовать Python и другие инструменты для получения наглядности в сети, например, графической топологии сети, динамически основанной на информации о сетевых устройствах.

Концепции IP-маршрутизации

IP-маршрутизация - это передача пакетов промежуточными устройствами между двумя конечными точками на основе IP-заголовка. При любой коммуникации через Интернет пакет проходит через различные промежуточные устройства. Как уже упоминалось, промежуточные устройства состоят из маршрутизаторов, коммутаторов, оптических передач и различных других устройств, которые не рассматриваются за пределами сетевого и транспортного уровня. В аналогии с дорожным путешествием вы можете проехать по Соединенным Штатам от города Сан-Диего в Калифорнии до города Сиэтл в Вашингтоне. IP-адрес источника аналогичен Сан-Диего, а IP-адрес назначения можно представить как Сиэтл. Во время путешествия вы будете останавливаться во многих промежуточных пунктах, таких как Лос-Анджелес, Сан-Франциско и Портленд; их можно рассматривать как промежуточные маршрутизаторы и коммутаторы между источником и пунктом назначения.

Почему это было важно? В некотором смысле, эта книга посвящена управлению и оптимизации этих промежуточных устройств. В эпоху мегацентров обработки данных, которые занимают площадь нескольких полей для американского футбола, необходимость в эффективных, гибких, надежных и экономичных способах управления сетью становится основным конкурентным преимуществом для компаний. В следующих главах мы подробно рассмотрим, как можно использовать программирование на Python для эффективного управления сетью.

Теперь, когда мы рассмотрели эталонные модели сетей и пакеты протоколов, мы готовы погрузиться в сам язык Python. В этой главе мы начнем с общего обзора Python.

Обзор языка Python

В двух словах, эта книга о том, как облегчить нашу жизнь сетевого инженера с помощью Python. Но что такое Python и почему этот язык выбирают многие инженеры DevOps? По словам исполнительного резюме Python Foundation (https:// www.python.org/doc/essays/blurb/):

"Python - это интерпретируемый, объектно-ориентированный, высокоуровневый язык программирования с динамической семантикой. Его высокоуровневая встроенная структура данных в сочетании с динамической типизацией и динамическим связыванием делает его очень привлекательным для быстрой разработки приложений, а также для использования в качестве языка сценариев или "клея" для соединения существующих компонентов. Простой, легко изучаемый синтаксис Python подчеркивает читабельность и, следовательно, снижает стоимость

сопровождения программ."

Если вы новичок в программировании, то объектно-ориентированная и динамическая семантика, о которой говорилось ранее, вероятно, не имеет для вас большого значения. Но я думаю, мы все можем согласиться, что быстрая разработка приложений и простой, легко изучаемый синтаксис - это хорошо. Python, будучи интерпретируемым языком, практически не требует компиляции перед выполнением, поэтому время на написание, тестирование и редактирование программ на Python значительно сокращается. Для простых сценариев, если ваш сценарий не работает, оператор print обычно является всем необходимым для отладки происходящего.

Использование интерпретатора также означает, что Python легко переносится на различные типы операционных систем, такие как Windows и Linux, и программа на Python, написанная на одной операционной системе, может быть использована на другой практически без изменений.

Функции, модули и пакеты способствуют повторному использованию кода, разбивая большую программу на простые многократно используемые части. Объектно-ориентированная природа Python делает еще один шаг вперед, группируя компоненты в объекты. Фактически, все файлы Python являются модулями, которые можно повторно использовать или импортировать в другую программу Python. Это облегчает обмен программами между инженерами и поощряет повторное использование кода. Python также имеет мантру "батареи включены", которая означает, что для выполнения обычных задач вам не нужно загружать никаких дополнительных пакетов, кроме самого языка Python. Чтобы достичь этой цели без чрезмерного разрастания кода, при установке интерпретатора Python устанавливается набор модулей Python, так называемых стандартных библиотек. Для таких распространенных задач, как регулярные выражения, математические функции и декодирование JSON, все, что вам нужно, - это оператор import, и интерпретатор перенесет эти функции в вашу программу. Эта мантра "батареи включены" - то, что я считаю одной из самых "убийственных" особенностей языка Python.

Наконец, тот факт, что код Python может начинаться в относительно небольшом скрипте с нескольких строк кода и вырасти в полноценную производственную систему, очень удобен для сетевых инженеров. Как многие из нас знают, сеть обычно растет органически, без генерального плана. Язык, который может расти вместе с размером вашей сети, бесценен. Вы можете удивиться, увидев, что язык, который многие считали языком сценариев, используется для создания полноценных производственных систем многими передовыми компаниями (организации, использующие Python; https://wiki.python.org/moin/OrganizationsUsingPython).

Если вы когда-нибудь работали в среде, где вам приходилось переключаться между работой на платформах разных производителей, таких как Cisco IOS и Juniper Junos, вы знаете, насколько болезненно переключаться между синтаксисом и использованием при попытке решить одну и ту же задачу. Поскольку Python достаточно гибок как для маленьких, так и для больших программ, в нем нет такого резкого переключения контекста. Это просто один и тот же код Python от мала до велика!

В оставшейся части главы мы проведем высокоуровневый экскурс по языку Python, чтобы немного подтянуть знания. Если вы уже знакомы с основами, не стесняйтесь быстро просмотреть ее или пропустить остальную часть главы.

Версии Python

Как многие читатели уже знают, в последние несколько лет Python переживает переход от Python 2 к Python 3. Python 3 был выпущен еще в 2008 году, более 10 лет назад, и активно развивается с последним выпуском 3.7. К сожалению, Python 3 не имеет обратной совместимости с Python 2.

На момент написания третьего издания этой книги, в конце 2019 года, сообщество Python в основном перешло на Python 3. Фактически, Python 2 официально прекратит свое существование с 1 января 2020 года (https://pythonclock.org/). Последняя версия Python 2.x, 2.7, была выпущена более шести лет назад, в середине 2010 года. К счастью, обе версии могут сосуществовать на одной машине. Учитывая, что к тому времени, когда вы будете читать этот отрывок, Python 2 будет устаревшим и не будет поддерживаться, нам всем следует перейти на Python 3. Более подробная информация о вызове интерпретатора Python приведена в следующем разделе, но здесь приведен пример вызова Python 2 и Python 3 на машине Ubuntu Linux:

```
$ python2
Python 2.7.15+ (default, Jul 9 2019, 16:51:35)
[GCC 7.4.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
$ python3.7
Python 3.7.4 (default, Sep 2 2019, 20:47:34)
[GCC 7.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
```

После выхода версии 2.7 большинство фреймворков Python теперь поддерживают Python 3. Python 3 также имеет много хороших возможностей, таких как асинхронный ввод/вывод, которые можно использовать, когда нам нужно оптимизировать наш код. В этой книге для примеров кода будет использоваться Python 3, если не указано иное. Мы также постараемся указать на различия между Python 2 и Python 3, когда это будет необходимо.

Если определенная библиотека или фреймворк лучше подходит для Python 2, например, Ansible (см. следующую информацию), это будет указано, и мы будем использовать Python 2 вместо него. Мы должны стремиться использовать Python 3 по умолчанию и использовать Python 2 только в случае крайней необходимости.

На момент написания статьи Ansible 2.8 и выше поддерживают Python 3. До версии 2.5 поддержка Python 3 считалась техническим предпросмотром. Учитывая относительно новую возможность поддержки, многие модули сообщества все еще находятся в процессе перехода на Python 3. Дополнительную информацию об Ansible и Python 3 можно найти на сайте https://docs.ansible.com/ansible/2.5/dev_guide/ developing_python_3.html.

Операционная система

Как уже упоминалось, Python является кроссплатформенным. Программы на Python можно запускать на Windows, Mac и Linux. В действительности, когда необходимо обеспечить кросс-платформенную совместимость, необходимо соблюдать определенную осторожность, например, обращать внимание на тонкие различия между обратными слешами в именах файлов Windows и активировать виртуальную среду на разных платформах. Поскольку эта книга предназначена для DevOps, системных и сетевых инженеров, Linux является предпочтительной платформой для целевой аудитории, особенно в производстве. Код в этой книге будет протестирован на машине Linux Ubuntu 18.04 LTS. Я также постараюсь сделать все возможное, чтобы код работал одинаково на платформах Windows и macOS. Если вас интересуют подробности об ОС, то они следующие:

```
$ uname -a
Linux network-dev-2 4.18.0-25-generic #26~18.04.1-Ubuntu SMP Thu Jun 27
07:28:31 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
```

Запуск программы Python

Программы Python выполняются интерпретатором, что означает, что код подается через этот интерпретатор для выполнения базовой операционной системой и отображения результатов. В сообществе разработчиков Python существует несколько различных реализаций интерпретатора, таких как IronPython и Jython. В этой книге мы будем использовать наиболее распространенный на сегодняшний день интерпретатор Python - CPython. Всякий раз, когда мы упоминаем Python в этой книге, мы имеем в виду CPython, если не указано иное.

Одним из способов использования Python является интерактивная подсказка. Это полезно, когда вы хотите быстро проверить часть кода или концепции Python без написания целой программы.

Обычно это делается простым вводом ключевого слова Python:

```
$ python3.7
Python 3.7.4 (default, Sep 2 2019, 20:47:34)
[GCC 7.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hello world")
hello world
```

В Python 3 оператор print является функцией, поэтому для него требуются круглые скобки. В Python 2 круглые скобки можно опустить.

Интерактивный режим - одна из самых полезных возможностей Python. В интерактивной оболочке вы можете ввести любое допустимое утверждение или последовательность утверждений и немедленно получить результат. Обычно я использую этот режим для изучения функций или библиотек, с которыми я не знаком. Интерактивный режим также можно использовать для более сложных задач, таких как эксперименты с поведением структур данных, например, мутабельных и неизменяемых типов данных. Поговорим о мгновенном удовлетворении!

В Windows, если вы не получаете приглашение оболочки Python, возможно, эта программа отсутствует в системном пути поиска. Последняя версия программы установки Python для Windows содержит флажок для добавления Python в системный путь; убедитесь, что он установлен во время установки. Или вы можете добавить программу в путь вручную, перейдя к настройкам среды.

Однако более распространенным способом запуска программы Python является сохранение файла Python и последующий его запуск через интерпретатор. Это избавит вас от необходимости набирать одни и те же утверждения снова и снова, как это приходится делать в интерактивной оболочке. Файлы Python - это обычные текстовые файлы, которые обычно сохраняются с расширением .py. В мире *Nix вы также можете добавить строку shebang (#!) сверху, чтобы указать интерпретатор, который будет использоваться для запуска файла. Символ # можно использовать для указания комментариев, которые не будут выполняться интерпретатором. Следующий файл helloworld.py содержит следующие утверждения:

```
# This is a comment print("hello world")
```

Это может быть выполнено следующим образом:

```
$ python helloworld.py
hello world
$
```

Встроенные типы Python

Python реализует динамическую типизацию, или утиную типизацию, и пытается автоматически определить тип объекта по мере того, как вы их объявляете. В Python есть несколько стандартных типов, встроенных в интерпретатор:

- Числа: int, float, complex и bool (подкласс int со значением True или False)
- Последовательности: str, list, tuple и range
- Отображения: dict
- Наборы: set и frozenset
- None: Нулевой объект

Тип None

Тип None обозначает объект, не имеющий значения. Тип None возвращается в функциях, которые явно ничего не возвращают. Тип None также используется в аргументах функций, чтобы выдать ошибку, если вызывающая сторона не передала фактическое значение.

Цифры

Числовые объекты Python - это, по сути, числа. За исключением булевых чисел, числовые типы int, long, float и complex являются знаковыми, то есть они могут быть положительными или отрицательными. Булево число - это подкласс целого числа, которое может иметь одно из двух значений: 1 - истина и 0 - ложь. На практике мы почти всегда проверяем булевы значения True или False вместо чисел 1 и 0. Остальные числовые типы различаются тем, насколько точно они могут представлять число; в Python 3 int не имеет максимального размера, в то время как в Python 2 int - это целые числа с ограниченным диапазоном. Floats - это числа, использующие представление двойной точности (64-битное) на машине.

Последовательности

Последовательности - это упорядоченные наборы объектов с индексом из неотрицательных целых чисел. В этом и нескольких следующих разделах мы будем использовать интерактивный интерпретатор для иллюстрации различных типов.

Пожалуйста, не стесняйтесь набирать текст на своем компьютере. Иногда людей удивляет, что строка - это тип последовательности. Но если присмотреться, то строки - это серия символов, собранных вместе. Строки заключаются в одинарные, двойные или тройные кавычки.

Обратите внимание, что в следующих примерах кавычки должны совпадать, а тройные кавычки допускаются чтобы строка охватывала разные строки:

```
>>> a = "networking is fun"
>>> b = 'DevOps is fun too'
>>> c = """what about coding?
... super fun!"""
>>>
```

Два других часто используемых типа последовательностей - это списки и кортежи. Списки - это последовательности произвольных объектов. Списки можно создавать, заключая объекты в квадратные скобки. Как и строки, списки индексируются ненулевыми целыми числами, которые начинаются с нуля.

Значения списка извлекаются путем обращения к номеру индекса:

```
>>> vendors = ["Cisco", "Arista", "Juniper"]
>>> vendors[0]
'Cisco'
>>> vendors[1]
'Arista'
>>> vendors[2]
'Juniper'
```

Кортежи похожи на списки, они создаются путем заключения значений в круглые скобки. Как и в списках, значения в кортеже извлекаются путем ссылки на его индексный номер. В отличие от списков, значения не могут быть изменены после создания:

```
>>> datacenters = ("SJC1", "LAX1", "SFO1")
>>> datacenters[0]
'SJC1'
>>> datacenters[1]
'LAX1'
>>> datacenters[2]
'SFO1'
```

Некоторые операции являются общими для всех типов последовательностей, например, возврат элемента по индексу, а также нарезка:

```
>>> a
'networking is fun'
>>> a[1]
'e'
>>> vendors
['Cisco', 'Arista', 'Juniper']
>>> vendors[1]
'Arista'
>>> datacenters
('SJC1', 'LAX1', 'SFO1')
>>> datacenters[1]
'LAX1'
>>>
>>> a[0:2]
'ne'
>>> vendors[0:2]
['Cisco', 'Arista']
>>> datacenters[0:2]
('SJC1', 'LAX1')
>>>
```

Помните, что индекс начинается с 0. Поэтому индекс 1 на самом деле является вторым элементом в последовательности.

Существуют также общие функции, которые можно применять к типам последовательностей, например, проверка количества элементов, минимального и максимального значений:

```
>>> len(a)
17
```

```
>>> len(vendors)
3
>>> len(datacenters)
3
>>>
>>> b = [1, 2, 3, 4, 5]
>>> min(b)
1
>>> max(b)
5
```

Неудивительно, что существуют различные методы, которые применяются только к строкам. Стоит отметить, что эти методы не изменяют сами базовые строковые данные и всегда возвращают новую строку. Короче говоря, мутабельные объекты, например, списки и словари, могут быть изменены после их создания, а неизменяемые объекты, например, строки, - нет. Если вы хотите использовать новое значение, вам нужно перехватить возвращаемое значение и присвоить его другой переменной:

```
>>> a
'networking is fun'
>>> a.capitalize()
'Networking is fun'
>>> a.upper()
'NETWORKING IS FUN'
>>> a
'networking is fun'
>>> b = a.upper()
>>> b
'NETWORKING IS FUN'
>>> a.split()
['networking', 'is', 'fun']
>>> a
'networking is fun'
>>> b = a.split()
>>> b
['networking', 'is', 'fun']
>>>
```

Вот некоторые из распространенных методов для списка. Тип данных Python list - это очень полезная структура для объединения нескольких элементов и итерации по ним по одному. Например, мы можем составить список коммутаторов позвоночника центра обработки данных и применить один и тот же список доступа ко всем из них, перебирая их по очереди. Поскольку значение списка может быть изменено после создания (в отличие от кортежей), мы также можем расширять и сужать существующий список по мере выполнения программы:

```
>>> routers = ['r1', 'r2', 'r3', 'r4', 'r5']
>>> routers.append('r6')
>>> routers
['r1', 'r2', 'r3', 'r4', 'r5', 'r6']
>>> routers.insert(2, 'r100')
>>> routers
['r1', 'r2', 'r100', 'r3', 'r4', 'r5', 'r6']
>>> routers.pop(1)
'r2'
>>> routers
```

```
['r1', 'r100', 'r3', 'r4', 'r5', 'r6']
```

Списки данных Python отлично подходят для хранения данных, но иногда бывает немного сложно отслеживать данные, если нам нужно ссылаться на них по местоположению. Далее мы рассмотрим типы отображения Python.

Составление карты

Python предоставляет один тип отображения, называемый словарем. Словарь - это то, что я считаю базой данных бедняка, поскольку он содержит объекты, которые можно индексировать по ключам. В других языках программирования это часто называют ассоциированным массивом или таблицей хеширования. Если вы использовали какие-либо словареподобные объекты в других языках, вы будете знать, что это мощный тип, потому что вы можете ссылаться на объект с помощью человекочитаемого ключа. Этот ключ, вместо простого списка элементов, будет иметь больше смысла для бедного парня, который пытается поддерживать и устранять неполадки в коде.

Этот парень может оказаться вами через несколько месяцев после того, как вы написали код и устраняли неполадки в два часа ночи. Объект в значении словаря также может быть другим типом данных, например, списком. Поскольку мы использовали квадратные скобки для списков и круглые скобки для кортежей, мы можем использовать фигурные скобки для создания словаря:

```
>>> datacenter1 = {'spines': ['r1', 'r2', 'r3', 'r4']}
>>> datacenter1['leafs'] = ['l1', 'l2', 'l3', 'l4']
>>> datacenter1
{'leafs': ['l1', 'l2', 'l3', 'l4'], 'spines': ['r1',
'r2', 'r3', 'r4']}
>>> datacenter1['spines']
['r1', 'r2', 'r3', 'r4']
>>> datacenter1['leafs']
['l1', 'l2', 'l3', 'l4']
```

Словарь Python - один из моих любимых контейнеров данных, которые я использую в своих сетевых сценариях. Есть и другие контейнеры данных, которые могут пригодиться, set - один из них.

Наборы

Набор используется для хранения неупорядоченной коллекции объектов. В отличие от списков и кортежей, множества неупорядочены и не могут быть проиндексированы числами. Но есть одна особенность, которая делает множества полезными: элементы множества никогда не дублируются. Представьте, что у вас есть список IP-адресов, которые вам нужно внести в список доступа. Единственная проблема в этом списке IP заключается в том, что он полон дубликатов.

Теперь подумайте о том, сколько строк кода вам пришлось бы использовать для циклического просмотра списка IP-адресов, чтобы отсортировать уникальные элементы по одному. Однако встроенный тип set позволит вам удалить дублирующиеся записи всего одной строкой кода. Честно говоря, я не так часто использую тип данных Python set, но когда он мне нужен, я всегда очень благодарен, что он существует. После создания набора или наборов их можно сравнить друг с другом с помощью объединений, пересечений и различий:

```
>>> a = "hello"
# Use the built-in function set() to convert the string to a set
>>> set(a)
{'h', 'l', 'o', 'e'}
```

```
>>> b = set([1, 1, 2, 2, 3, 3, 4, 4])
>>> b
{1, 2, 3, 4}
>>> b.add(5)
>>> b
{1, 2, 3, 4, 5}
>>> b.update(['a', 'a', 'b', 'b'])
>>> b
{1, 2, 3, 4, 5, 'b', 'a'}
>>> a = set([1, 2, 3, 4, 5])
>>> b = set([4, 5, 6, 7, 8])
>>> a.intersection(b)
{4, 5}
>>> a.union(b)
{1, 2, 3, 4, 5, 6, 7, 8}
>>> 1 *
{1, 2, 3}
>>>
```

Теперь, когда мы рассмотрели различные типы данных, мы рассмотрим операторы Python.

Операторы Python

В Python есть некоторые числовые операторы, такие как +, - и так далее; обратите внимание, что усекающее деление (//, также известное как деление на пол) усекает результат до целого числа и плавающей точки и возвращает целое значение. Оператор modulo (%) возвращает значение остатка при делении:

```
>>> 1 + 2
3
>>> 2 - 1
1
>>> 1 * 5
5
>>> 5 / 1 #returns float
5.0
>>> 5 // 2 # // floor division
2
>>> 5 % 2 # modular operator
1
```

Существуют также операторы сравнения. Обратите внимание на двойной знак равенства для сравнения и одинарный знак равенства для присвоения переменной:

```
>>> a = 1
>>> b = 2
>>> a == b
False
>>> a > b
False
>>> a < b
True
>>> a <= b
True
```

Мы также можем использовать два общих оператора членства, чтобы узнать, относится ли объект к типу последовательности:

```
>>> a = 'hello world'
>>> 'h' in a
True
>>> 'z' in a
False
>>> 'h' not in a
False
>>> 'z' not in a
True
```

Операторы Python позволяют нам эффективно выполнять простые операции. В следующем разделе мы рассмотрим, как можно использовать потоки управления для повторения этих операций.

Инструменты для управления потоком управления Python

Операторы if, else и elif управляют условным выполнением кода. В отличие от некоторых других языков программирования, Python использует отступы для структурирования блоков. Как и следовало ожидать, формат условного оператора выглядит следующим образом:

```
if expression:
 do something
elif expression:
 do something if the expression meets
elif expression:
 do something if the expression meets
...
else:
 statement
```

Вот простой пример:

```
>>> a = 10
>>> if a > 1:
... print("a is larger than 1")
... elif a < 1:
... print("a is smaller than 1")
... else:
... print("a is equal to 1")
...
a is larger than 1
>>>
```

Цикл while будет выполняться до тех пор, пока условие не станет False, поэтому будьте осторожны с ним, если не хотите продолжения выполнения (и краха вашего процесса):

```
while expression:
 do something
```

```
>>> a = 10
```

```
>>> b = 1
>>> while b < a:
... print(b)
... b += 1
...
1
2
3
4
5
6
7
8
9
>>>
```

Цикл for работает с любым объектом, который поддерживает итерацию; это означает, что все встроенные типы последовательностей, такие как списки, кортежи и строки, могут быть использованы в цикле for.

Буква i в следующем цикле for является итерирующей переменной, поэтому вы можете выбрать то, что имеет смысл в контексте вашего кода:

```
for i in sequence:
 do something
>>> a = [100, 200, 300, 400]
>>> for number in a:
... print(number)
...
100
200
300
400
```

Теперь, когда мы рассмотрели типы данных Python, операторы и потоки управления, мы готовы объединить их в многократно используемые части кода, называемые функциями.

Функции Python

Чаще всего, когда вы копируете и вставляете некоторые фрагменты кода, вам следует разбить его на самостоятельные куски функций. Такая практика позволяет повысить модульность, облегчает сопровождение и позволяет повторно использовать код. Функции Python определяются с помощью ключевого слова def с именем функции, за которым следуют параметры функции. Тело функции состоит из операторов Python, которые должны быть выполнены. В конце функции вы можете выбрать, возвращать ли значение вызывающей функции, или, по умолчанию, она возвращает объект None, если вы не указали возвращаемое значение:

```
def name(parameter1, parameter2):
 statements
 return value
```

В последующих главах мы увидим еще много примеров функций, поэтому здесь приведен краткий пример. В следующих примерах мы используем позиционные параметры, поэтому первый элемент

всегда называется первой переменной в функции. Другой способ обращения к параметрам - это ключевые слова со значениями по умолчанию, например def subtract(a=10, b=5):

```
>>> def subtract(a, b):
... c = a - b
... return c
...
>>> result = subtract(10, 5)
>>> result
5
>>>
```

Функции Python отлично подходят для группировки задач. Можем ли мы сгруппировать различные функции в более крупный кусок многократно используемого кода? Да, мы можем сделать это с помощью классов Python.

Классы Python

Python - это язык объектно-ориентированного программирования (ООП). В Python объекты создаются с помощью ключевого слова class. Объект Python чаще всего представляет собой набор функций (методов), переменных и атрибутов (свойств). После определения класса можно создавать его экземпляры. Класс служит образцом для последующих экземпляров.

Тема ООП выходит за рамки этой главы, поэтому здесь приведен простой пример определения объекта маршрутизатора:

```
>>> class router(object):
... def __init__(self, name, interface_number, vendor):
... self.name = name
... self.interface_number = interface_number
... self.vendor = vendor
...
>>>
```

После определения вы можете создать столько экземпляров этого класса, сколько захотите:

```
>>> r1 = router("SF01-R1", 64, "Cisco")
>>> r1.name
'SF01-R1'
>>> r1.interface_number
64
>>> r1.vendor
'Cisco'
>>>
>>> r2 = router("LAX-R2", 32, "Juniper")
>>> r2.name
'LAX-R2'
>>> r2.interface_number
32
>>> r2.vendor
'Juniper'
>>>
```

Конечно, в Python объектов и ООП гораздо больше. Мы рассмотрим больше примеров в следующих главах.

Модули и пакеты Python

Любой исходный файл Python может быть использован в качестве модуля, фактически, файл Python является модулем, и любые функции и классы, которые вы определили в этом исходном файле, могут быть использованы повторно. Чтобы загрузить код, файл, ссылающийся на модуль, должен использовать ключевое слово import. При импорте файла происходят три вещи:

1. Файл создает новое пространство имен для объектов, определенных в исходном файле 2. Вызывающая программа выполняет весь код, содержащийся в модуле 3. Файл создает в вызывающей программе имя, которое ссылается на импортируемый модуль. Имя совпадает с именем модуля

Помните функцию subtract(), которую вы определили с помощью интерактивной оболочки? Чтобы повторно использовать эту функцию, мы можем поместить ее в файл с именем subtract.py:

```
def subtract(a, b):
 c = a - b
 return c
```

В файле в том же каталоге, что и subtract.py, вы можете запустить интерпретатор Python и импортировать эту функцию:

```
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import subtract
>>> result = subtract.subtract(10, 5)
>>> result
5
```

Это работает потому, что по умолчанию Python сначала ищет в текущем каталоге доступные модули. Помните стандартную библиотеку, о которой мы упоминали некоторое время назад? Вы угадали, это просто файлы Python, используемые в качестве модулей.

Если вы находитесь в другом каталоге, вы можете вручную добавить местоположение пути поиска, используя модуль sys с помощью sys.path.

Пакеты позволяют сгруппировать коллекцию модулей. Это позволяет организовать модули Python для большей защиты пространства имен и дальнейшего повторного использования. Пакет определяется путем создания каталога с именем, которое вы хотите использовать в качестве пространства имен, затем вы можете разместить исходный файл модуля в этом каталоге.

Для того чтобы Python распознал его как пакет Python, просто создайте файл __init__.py в этом каталоге. Файл __init__.py часто может быть пустым. В том же примере, что и с файлом subtract.py, если бы вы создали каталог math_stuff и создали файл __init__.py:

```
echou@pythonicNeteng:~/Master_Python_Networking/Chapter1$ mkdir math_
stuff
echou@pythonicNeteng:~/Master_Python_Networking/Chapter1$ touch math_
```

```
stuff/__init__.py
echou@pythonicNeteng:~/Master_Python_Networking/Chapter1$ tree
.
├── helloworld.py
└── math_stuff
    ├── __init__.py
    └── subtract.py
1 directory, 3 files
echou@pythonicNeteng:~/Master_Python_Networking/Chapter1$
```

То, как вы теперь будете ссылаться на модуль, должно включать имя пакета с использованием точечной нотации, например, math_stuff.subtract:

```
>>> from math_stuff.subtract import subtract
>>> result = subtract(10, 5)
>>> result
5
>>>
```

Как видите, модули и пакеты - это отличный способ организовать большие файлы кода и значительно упростить совместное использование кода Python.

Резюме

В этой главе мы рассмотрели модель OSI и познакомились с наборами сетевых протоколов, такими как TCP, UDP и IP. Они работают как уровни, которые обрабатывают адресацию и переговоры о связи между любыми двумя хостами. Эти протоколы были разработаны с учетом их расширяемости и в значительной степени остались неизменными с момента их первоначального создания. Учитывая взрывной рост Интернета, это довольно большое достижение.

Мы также быстро рассмотрели язык Python, включая встроенные типы, операторы, потоки управления, функции, классы, модули и пакеты. Python - это мощный, готовый к производству язык, который также легко читается. Это делает язык идеальным выбором, когда речь идет об автоматизации сети. Сетевые инженеры могут использовать Python, чтобы начать с простых сценариев и постепенно переходить к другим расширенным функциям.

В главе 2 "Низкоуровневое взаимодействие с сетевыми устройствами" мы начнем рассматривать использование Python для программного взаимодействия с сетевым оборудованием.