МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА» (СПбГУТ)

Факультет ИКСС

Кафедра ЗСС

Допустить к защите

Заведующий кафедрой

_____ _____

(подпись)

«27» мая 2022 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Исследование возможности применения технологий автоматизации в целях повышения информационной безопасности в компьютерных сетях

(тема ВКР)

Направление/специальность подготовки

10.03.01 Информационная безопасность

(код и наименование направления/специальности)

Направленность (профиль)

Безопасность компьютерных систем

(наименование)

Квалификация *Бакалавр*

(наименование квалификации в соответствии с ФГОС ВО)

Студент:

Захаров Денис Артурович; ИКБ-82 _____

(Ф.И.О., № группы) *(подпись)*

Научный руководитель:

к. т. н., доц. каф. ЗСС; Ушаков И. А. _____

(учёная степень, учёное звание, Ф.И.О.)(подпись)

Санкт-Петербург, 2022

Оборотная сторона титульного листа

_____

работа написана мною самостоятельно

_____

работа не содержит неправомерных заимствований

_____

_____

работа может быть размещена в электронно-библиотечной системе университета

_____ _____ _____

(дата) *(подпись)* *(ФИО студента)*

Текст ВКР размещен в электронно-библиотечной системе университета.

Руководитель отдела комплектования библиотеки _____

(Ф.И.О.)

_____ _____

(дата) *(подпись)*

Коэффициент оригинальности ВКР _____ % .

Проверил: _____

(Должность, Ф.И.О.)

_____ _____

(дата) *(подпись)*

МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА» (СПбГУТ)

Факультет ИКСС Кафедра ЗСС
Направление (специальность) 10.03.01 Информационная безопасность
(код и наименование)

Утверждаю:
Зав. кафедрой Красов А. В.
_____
(Ф.И.О., подпись)
«27» мая 2022 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы (ВКР)

1. Студент _____Захаров Денис Артурович_____ № группы ИКБ-82_
(фамилия, имя, отчество)

2. Руководитель_____Ушаков Игорь Александрович к.т.н., доцент кафедры ЗСС_____
(фамилия, имя, отчество, должность, уч. степень и звание)

3. Квалификация _____Бакалавр_____
(наименование в соответствии с ФГОС ВО)

4. Тема ВКР Исследование возможности применения технологий автоматизации в целях повышения информационной безопасности в компьютерных сетях
_____
_____
утверждена приказом ректора университета от «15» апреля 2022 г. № 418/кс

5. Исходные данные (технические требования): техническое задание
_____
_____
_____
_____

6. Содержание работы (анализ состояния проблемы, проведение исследований, разработка, расчеты параметров, экономическое обоснование и др.)
РАЗРАБОТКА И ДИЗАЙН ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ, ДИЗАЙН ПАТТЕРНЫ ПРОГРАММ, ВНИКАЕМ И НАЧИНАЕМ ИСПОЛЬЗЛВАТЬ API, РАЗВЕРТЫВАНИЕ ПРИЛОЖЕНИЙ И БЕЗОПАСНОСТЬ, ИНФРАСТРУКТУРА И АВТОМАТИЗАЦИЯ, ОПИСАНИЕ ЛАБОРАТОРНЫХ И ПРОЕКТНЫХ РАБОТ, ПРИЛОЖЕНИЕ А (ЛАБОРАТОРНЫЕ РАБОТЫ), ПРИЛОЖЕНИЕ Б (ПРОЕКТНЫЕ РАБОТЫ), ПРИЛОЖЕНИЕ В (ДОПОЛНЯЮЩИЙ ТЕОРИТИЧЕСКИЙ МАТЕРИАЛ)
7. Вид отчетных материалов, представляемых в ГЭК (пояснительная записка, перечень, графического материала, отчет о НИР, технический проект, образцы и др.):
презентация, пояснительная записка_____

_____
_____
_____
_____
_____

8. Консультанты по ВКР с указанием относящихся к ним разделов

Раздел	Руководитель	Подпись дата	
		Задание выдал	Задание принял
1. Ознакомление с теоритической состовляющей работы	Ушаков И.А.		
2. Составление требований и реализация теоритической части ВКР	Ушаков И.А.		
3. Составление требований и реализация практической чати ВКР	Ушаков И.А.		
4. Составление презентации и речи для презентации ВКР	Ушаков И.А.		

Дата выдачи задания «21» ноября 2021 г.

Дата представления ВКР к защите «10» июня 2022 г.

Руководитель ВКР _____

(подпись)

Студент _____
(подпись)

КАЛЕНДАРНЫЙ ПЛАН

Наименование этапа выпускной квалификационной работы	Срок выполнения	Примечание
1. Постановка цели выполнения ВКР и задач	21.11.2021	
2. Работа с теоретическим материалом	22.11.2021 — 22.05.2022	
3. Сбор информации, необходимой для написания работы		
4. Систематизация и обработка материалов ВКР		
5. Анализ полученных в работе результатов, обобщение		
6. Подготовка отчетных материалов, представляемых в государственную экзаменационную комиссию, доклада к защите и презентации	16.05.2022 — 30.06.2022	
7. Консультации с руководителем ВКР		
8. Представление выполненной ВКР руководителю для подготовки отзыва		
9. Подготовка к защите ВКР, включая подготовку к процедуре защиты и процедуру защиты		

Научный руководитель,
доц. кафедры ЗСС _____ Ушаков И.А.
подпись, дата

Студент,
группа ИКБ-82 _____ Захаров Д.А.

РЕФЕРАТ

Тема дипломной работы: «Исследование возможности применения технологий автоматизации в целях повышения информационной безопасности в компьютерных сетях».

Пояснительная записка содержит:

Листов: 618

Рисунков: 110

Таблиц: 15

Используемых источников: 20

Перечень ключевых слов: Information Security, Linux, Python, Docker, Jenkins, REST API, Ansible, DevOps, SRE, Automation

Цель работы: Исследовать возможность применения технологий автоматизации с целью повышения ИБ для помощи разработчикам, при создании новых и эксплуатации существующих приложений, их интеграции с другими продуктами, платформами и API-интерфейсами. Сопроводить материал практической составляющей в качестве дополнительных технологий, не вошедших в основную часть дипломной работы, а также комплексом лабораторных и проектных работ.

Методы проведения работы: Рассмотрены проблемы и угрозы предшествующего года в индустрии; подходы к разработке ПО; инструменты автоматизации; процессы, применяемые в организациях.

Область применения: Дипломная работа, может быть использована в целях подготовки инженеров по специальностям, связанным с защитой информации, посредством обучения современным best practice разработки программного обеспечения, сетевых технологий и автоматизации.

Экономическая значимость работы: Исключение репутационных и финансовых убытков из-за выгорания сотрудников и кабератак, посредством применения автоматизации в целях изоляции процессов для повышения их информационной безопасности.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 7
1 РАЗРАБОТКА И ДИЗАЙН ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 9
1.1 Жизненный цикл разработки программного обеспечения (SDLC) 9
1.2 Методологии Разработки Программного Обеспечения 10
2 ДИЗАЙН ПАТТЕРНЫ ПРОГРАММ 15
2.1 Введение 15
2.2 Базовые паттерны проектирования 16
2.3 Пример — Паттерн Observer 16
3 ВНИКАЕМ И НАЧИНАЕМ ИСПОЛЬЗЛВАТЬ API 20
3.1 Введение 20
3.2 Знакомство с API 20
3.3 Введение в REST API 22
4 РАЗВЕРТЫВАНИЕ ПРИЛОЖЕНИЙ И БЕЗОПАСНОСТЬ 29
4.1 Введение в развертывание и безопасность приложений 29
4.2 Варианты развертывания с помощью различных моделей 29
4.3 Разработка и развертывание экспериментального приложения 34
4.4 Continuous Integration/Continuous Deployment (CI/CD) 38
4.5 Сети для разработки приложений и безопасности 41
4.6 OWASP Top Ten на 2021 год 55
5 ИНФРАСТРУКТУРА И АВТОМАТИЗАЦИЯ 57
5.1 Автоматизация инфраструктуры 57
5.2 Инструменты автоматизации 61
6 ОПИСАНИЕ ЛАБОРАТОРНЫХ И ПРОЕКТНЫХ РАБОТ 67
6.1.1 Установка виртуальной лабораторной среды 1.1.2 67
6.1.2 Знакомство с Linux 1.2.2 67
6.1.3 Знакомство с Языком Программирования Python 1.3.3 67
6.1.4 Исследование инструментов разработки на языке Python 3.1.12 68
6.1.5 Контроль версий программного обеспечения с помощью Git 3.3.11 . 68
6.1.6 Знакомство с классами в Python 3.4.6 69
6.1.7 Unit-тестирование на Python 3.5.7 69
6.1.8 Парсинг различных типов данных с помощью Python 3.6.6 70
6.1.9 Изучение Rest API с помощью API Simulator и Postman 4.5.5 71

6.1.10 Интеграция REST API и Python приложения 4.9.2 71
6.1.11 Инструменты для устранения неполадок в сети 5.6.7 72
6.1.12 Сборка веб-приложения в Docker контейнере 6.2.7 72
6.1.13 Построение CI/CD с помощью Jenkins 6.3.6 72
6.1.14 Исследование эволюции парольных систем 6.5.10 73
6.1.15 Установка виртуальной машины CSR1000v 7.0.3 73
6.1.16 Ansible для резервного копирования и настройки устройства 7.4.7. 74
6.1.17 Ansible для автоматизации установки веб-сервера 7.4.8 74
6.1.18 Автоматизированное тестирование с помощью pyATS и Genie 7.6.3 75
6.1.19 Исследование YANG моделей 8.3.5 75
6.1.20 NETCONF для доступа к устройству IOS XE 8.3.6 76
6.1.21 RESTCONF для доступа к устройству IOS XE 8.3.7 76
6.1.22 Создание Python программы для управления Webex Teams 8.6.7 77
6.1.23 Исследование простой сети 5.4.6 77
6.1.24 Исследование сетевых протоколов 5.5.7 78
6.1.25 Сравнение CLI и SDN контроллера 8.8.2 78
6.1.26 Сравнение CLI и SDN контроллера 8.8.3 79
6.2.1 Формирование команды 79
6.2.2 Формирование Agile команды 80
6.2.3 Социальное программирование 80
6.2.4 Автоматизированное тестирование и развертывание программного обеспечения 80
6.2.5 Программирование и автоматизация сети 80
ЗАКЛЮЧЕНИЕ 81
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 82
ПРИЛОЖЕНИЕ А 84
ПРИЛОЖЕНИЕ Б 395
ПРИЛОЖЕНИЕ В 417

ВВЕДЕНИЕ

По мере роста темпов развития IT требуется всё быстрее предоставлять приложения и службы пользователям. Появляется больше типов устройств, их количество растет, при этом угрозы безопасности становятся изощрёнными. IT сектору необходимо соответствовать новым условиям, однако физическая и сетевая инфраструктура сдерживает их. Никто не любит тратить деньги и совершать лишние действия, в результате этого происходят нарушения информационной безопасности, как из-за выгорания сотрудников работы в таком темпе, так из-за кибератак, в результате предприятие несёт репутационный, а как следствие финансовый ущерб (Рисунок 1) [12, 13]. Вследствие чего, происходит медленная поставка программного обеспечения.

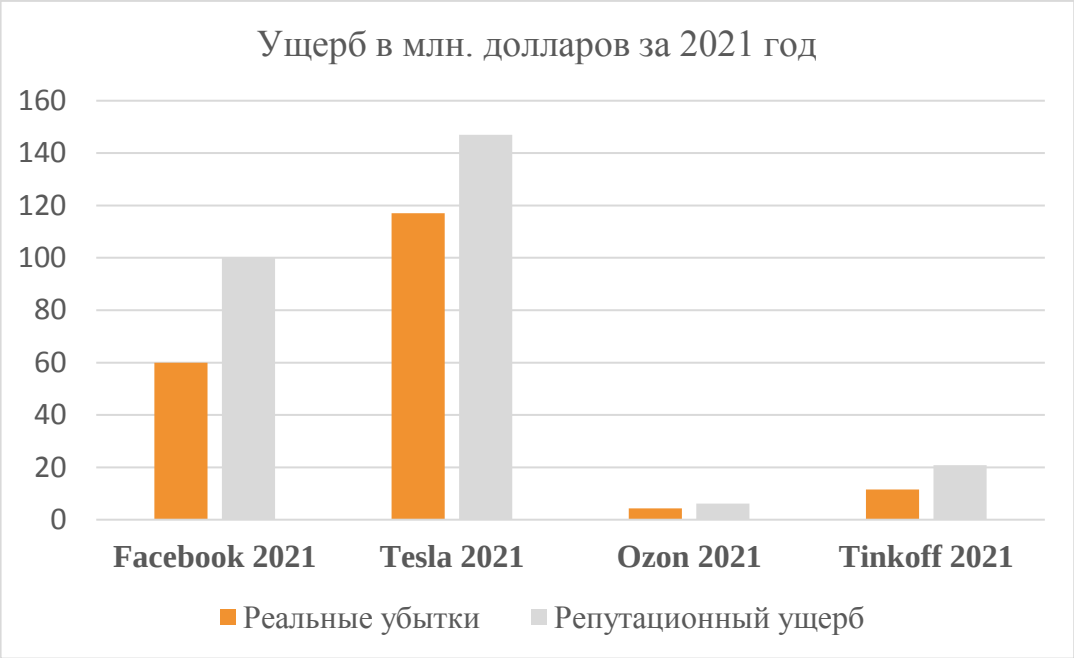


Рисунок 1 — Финансовые потери в IT секторе за 2021г.

Одним из решений проблемы, напрашивается автоматизация рутинных процессов, в компаниях, которые можно избежать, но в связи с этим необходимо понять эти процессы и инструменты, а индустрия постепенно приходит к автоматизации, во всех её отраслях от приложений до их интеграции. И очень важно обладать необходимыми знаниями чтобы оставаться на волне с этим стремительным темпом развития технологий.

Причиной такого сдвига в отрасли являются Application Programming Interfaces (API). Нам необходимо повысить эффективность развёртывания приложений, сетей и инфраструктур. Один из способов сделать это — устранить повторяющиеся ручные задачи с помощью автоматизации с использованием API. API позволяют создавать автоматизированные сценарии, а также интеграции, которые создают сеть между приложениями. Это позволяет приложениям обмениваться данными, запрашивать выполнение действий или даже уведомлять друг друга о наступлении определенного

события. Об API, их преимуществах и способах устранения неполадок мы подробно поговорим в Главе 3 "Вникаем и начинаем использлвать API".

Чтобы уметь использовать API для написания сценариев и создания приложений, вы должны знать, как писать код. Вам необходимо изучить основные концепции разработки программного обеспечения и вооружиться необходимыми инструментами для написания качественного кода.

Автоматизация и интеграция через API не могут существовать без сети. Сеть обеспечивает связь между приложениями и клиентами. Использование API помогло сделать сеть более эффективной благодаря программируемости сети. Программируемость сети — это возможность конфигурировать, контролировать и реагировать на события в сети в режиме реального времени. Важно изучить основы работы сети.

С ростом числа API в отрасли и совместной интеграции приложений важно понимать, как защитить сеть и приложения, чтобы избежать таких инцидентов, как утечка данных. Для каждого типа развертывания существуют свои проблемы безопасности, поэтому вам необходимо знать о различных способах развертывания, чтобы сделать лучший выбор для вашего приложения. Более подробная информация о защите сети и приложений рассматривается в Главе 4 "Развертывание приложений и безопасность".

API уже открыли двери для автоматизации сети. Автоматизация сети может быть продвинута на шаг вперед за счет автоматизированного развертывания приложений. Это можно сделать с помощью подхода “Инфраструктура как кода”. По сути, это возможность управлять инфраструктурой с помощью автоматизации, вместо того чтобы настраивать ее вручную, об этом поговорим в Главе 5 "Инфраструктура и Автоматизация".

1 РАЗРАБОТКА И ДИЗАЙН ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

1.1 Жизненный цикл разработки программного обеспечения (SDLC)

Процесс разработки программного обеспечения также известен как жизненный цикл разработки программного обеспечения (SDLC). Этот процесс — нечто большее, чем просто кодинг. Он также включает в себя сбор требований, создание прототипа, тестирование и исправление ошибок.

В этой главе мы обсудим фазы жизненного цикла разработки программного обеспечения, а затем методологии управления реальными требованиями программных проектов. Обсуждаемые здесь методологии начинаются с водопадного метода, который делает акцент на предварительном планировании. Другие методологии в этой теме, такие как Agile и Lean, являются более динамичными и адаптивными, чем метод водопада.

Фазы жизненного цикла разработки программного обеспечения

Жизненный цикл разработки программного обеспечения (SDLC) — это процесс разработки программного обеспечения, начинающийся с идеи и заканчивающийся “доставкой”. Этот процесс, обычно, состоит из шести фаз. Каждая фаза опирается на результаты предыдущей фазы. Стандартного SDLC не существует, поэтому точные фазы могут варьироваться, но наиболее распространенными являются следующие (Рисунок 2):

Рисунок 2 — Фазы SDLC

Исторически сложилось так, что команды разработчиков обычно следовали этим фазам прибегая к методологии водопада. Целью водопада было завершить каждую фазу SDLC до последней детали, прежде чем перейти к следующей, никогда не возвращаясь к предыдущей фазе и записывая все по пути.

Хотя “водопадный” метод всё еще широко используется сегодня, он постепенно вытесняется более адаптивными, гибкими методологиями, которые позволяют создавать более качественное программное обеспечение,

быстрее и с меньшей болью. Эти методы известны под общим названием "Agile разработка".

Важно понимать, что SDLC можно применять по-разному. Его этапы могут повторяться, а порядок может быть изменен. Отдельные фазы могут выполняться параллельно на многих уровнях (например, требования могут быть собраны отдельно для деталей пользовательского интерфейса, интеграции back-end, параметров работы и производительности и т.д.).

Мы рассмотрим фазы SDLC более подробно и в их классическом порядке (только помните: это описание, а не предписание)

1.2 Методологии Разработки Программного Обеспечения

Методология разработки программного обеспечения также известна как модель жизненного цикла разработки программного обеспечения. Эти методологии представляют собой не что иное, как набор правил, шагов, ролей и принципов. Существует множество различных методологий, но мы остановимся на трех наиболее популярных:

- Waterfall;
- Agile;
- Lean.

Каждая методология имеет свои достоинства и недостатки. Решение о том, какую из них использовать, зависит от многих факторов, таких как тип проекта, продолжительность проекта и размер команды.

Разработка ПО с применением Waterfall

Waterfall или Водопад — это традиционная модель разработки программного обеспечения, которая практикуется и по сей день. Водопадная модель практически идентична жизненному циклу разработки программного обеспечения, поскольку каждая фаза зависит от результатов предыдущей фазы проиллюстрировано на рисунке 3.

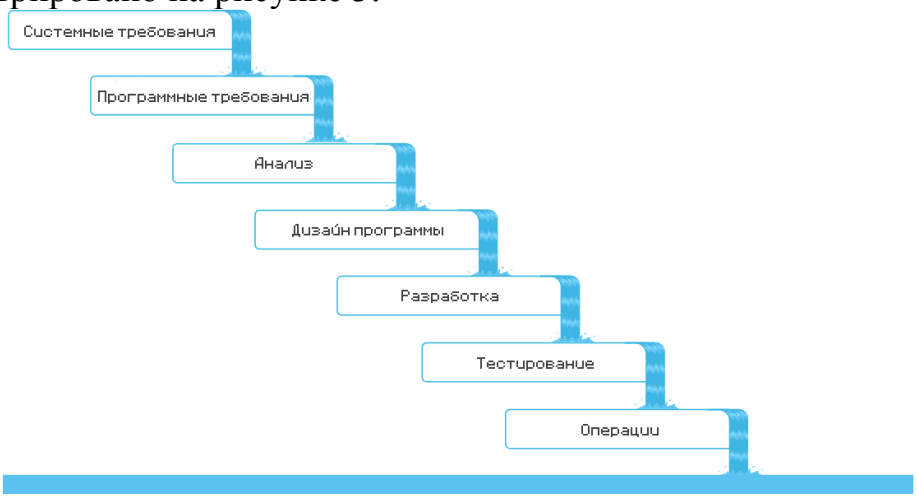


Рисунок 3 — Модель Waterfall

В водопаде вода течет только в одном направлении. При водопадном методе процесс идет в одном направлении и никогда не может пойти вспять.

Подумайте об этом как об эстафете, где один бегун должен закончить свою дистанцию, прежде чем передать эстафету следующему, который ждет его. Эстафета всегда движется вперед.

Говорят, что оригинальная модель водопада была создана Уинстоном В. Ройсом. Его оригинальная модель состояла из семи фаз:

- Требования к системе;
- Программные требования;
- Анализ;
- Дизайн продукта;
- Его разработка;
- Тестирование;
- Дальнейшие операции.

Как видите, водопадная модель — это всего лишь одна из итераций жизненного цикла разработки программного обеспечения. В настоящее время существует множество вариаций фаз водопадной модели, но идея о том, что каждая фаза не может пересекаться и должна быть завершена, прежде чем двигаться дальше, остается неизменной.

Поскольку результат каждой фазы является критическим для следующей, одно неверное решение может свести на нет всю итерацию; поэтому в большинстве реализаций водопадной модели требуется документация, обобщающая результаты каждой фазы в качестве исходных данных для следующей фазы. Если требования меняются в течение текущей итерации, то эти новые требования не могут быть учтены до следующей итерации водопада, что может оказаться дорогостоящим для крупных программных проектов и привести к значительным задержкам, прежде чем требуемые функции станут доступны пользователям.

Разработка ПО с применением Agile

Методология Agile является гибким и ориентированным на клиента. Хотя методологии, схожие с Agile, уже применялись на практике, она не являлась официальной до 2001 года, когда семнадцать разработчиков программного обеспечения объединились, чтобы найти решение своих проблем, связанных с существующими вариантами, и выработали Манифест о гибкой разработке программного обеспечения, также известный как Манифест Agile.

Манифест Agile

Согласно манифесту, ценность Agile следующие:

- Личности и взаимодействие вместо процессов и инструментов;
- Рабочее программное обеспечение вместо исчерпывающей документации;
- Сотрудничество с клиентами вместо переговоров по контракту;
- Реагирование на изменения вместо следования плану.

Также, в манифесте перечислены двенадцать различных принципов:

- **Клиентоориентированность** — Наш главный приоритет — удовлетворение потребностей клиента путем своевременной и непрерывной поставки ценного программного обеспечения.
- **Принятие изменений и адаптация** — Приветствовать изменение требований, даже на поздних этапах разработки. Agile-процессы используют изменения для конкурентного преимущества клиента.
- **Частая поставка рабочего программного обеспечения** — Поставляйте рабочее программное обеспечение часто, от пары недель до пары месяцев, отдавая предпочтение более коротким срокам.
- **Сотрудничество** — бизнесмены и разработчики должны ежедневно работать вместе на протяжении всего проекта.
- **Мотивированные команды** — Стройте проекты вокруг мотивированных людей. Обеспечьте им необходимые условия и поддержку и доверьте им выполнение работы.
- **Беседы лицом к лицу** — Наиболее эффективным и действенным методом передачи информации команде разработчиков и внутри нее является беседа лицом к лицу.
- **Работающее программное обеспечение** — Работающее программное обеспечение является основным показателем прогресса.
- **Работа в устойчивом темпе** — Agile-процессы способствуют устойчивому развитию. Спонсоры, разработчики и пользователи должны иметь возможность поддерживать постоянный темп работы неограниченное время.
- **Agile-среда** — Постоянное внимание к техническому совершенству и хорошему дизайну повышает гибкость.
- **Простота** — искусство максимально сократить объем невыполненной работы — имеет важное значение.
- **Самоорганизующиеся команды** — Лучшие архитектуры, требования и проекты появляются в самоорганизующихся командах.
- **Непрерывное совершенствование** — через регулярные промежутки времени команда размышляет о том, как стать более эффективной, а затем настраивает и корректирует свое поведение соответствующим образом.

Методы Agile

Манифест Agile по своему замыслу не был точным в отношении того, как он *(Agile)* должен работать. После создания Манифеста его авторы (и многие другие) продолжали развивать эти идеи, впитывая новые из многих источников и проверяя их на практике. В результате за последние несколько десятилетий появилось множество вариантов Agile, и некоторые из них стали широко популярными. К ним относятся:

– **Agile Scrum** — В регби термин "схватка" описывает момент игры, когда игроки собираются вместе и пытаются завладеть мячом. Scrum фокусируется на небольших, самоорганизующихся командах, которые ежедневно проводят короткие встречи и работают в итеративных спринтах, постоянно адаптируя результаты для удовлетворения меняющихся требований.
– **Lean** — Основанный на бережливом производстве, метод Lean делает акцент на устранении напрасных усилий при планировании и выполнении, а также на снижении когнитивной нагрузки на программиста.
– **Экстремальное программирование (XP)** — По сравнению со Scrum, XP более предписывает лучшие практики программной инженерии и более целенаправленно решает специфические вопросы качества жизни, с которыми сталкиваются команды разработчиков программного обеспечения.
– **Feature-Driven Development (FDD)** — FDD предписывает, что разработка программного обеспечения должна вестись в терминах общей модели, разбитой, спланированной, спроектированной и построенной по функциям. Он определяет процесс моделирования, используемый для оценки и планирования поставки функций, а также детальный набор ролей для основной команды разработчиков и сотрудников службы поддержки.

Из описанных выше методологий Agile Scrum (Рисунок 4), вероятно, является самой популярной.

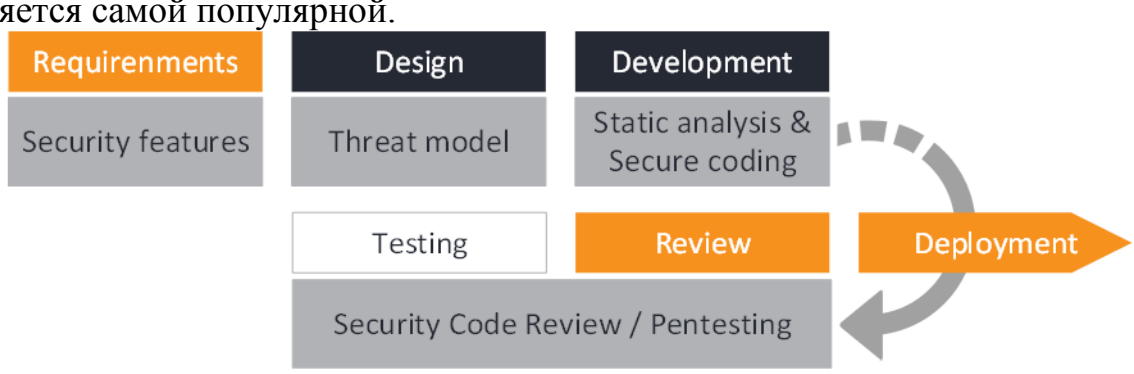


Рисунок 4 — Agile Scrum с подходом Security SDLS рассмотренная на СтудВесне2021

Ниже мы обсудим некоторые термины и концепции Scrum, которые были более или менее универсально приняты сообществом Agile во всех методологиях.

Спринты

В модели Agile по-прежнему применяется жизненный цикл разработки программного обеспечения. В отличие от водопадного метода, где есть одна длинная итерация SDLC, Agile — это множество быстрых итераций SDLC.

Эти быстрые итерации называются спринтами, и цель спринтов — обеспечить частое предоставление работающего программного обеспечения,

что является принципом манифеста Agile. Спринт — это определенный период времени (тайм-бокс), который обычно составляет от двух до четырех недель, но желательно как можно короче. Продолжительность спринта должна быть определена до начала процесса и должна редко меняться.
Во время спринта каждая команда берет на себя столько задач, также известных как пользовательские истории, сколько, по их мнению, они могут выполнить в течение установленного времени спринта. По окончании спринта программное обеспечение должно быть работоспособным и готовым к сдаче, но это не обязательно означает, что оно будет сдано; спринт не всегда приводит к релизу, но Agile требует, чтобы программное обеспечение оставалось готовым к сдаче.

Scrum-команды

Scrum-команды — это кросс-функциональные, совместные, самоуправляемые и самодостаточные команды. В идеале в таких Scrum-группах не должно быть больше 10 человек, но они должны быть достаточно большими, чтобы закончить историю пользователя в течение спринта.

Каждый день каждая scrum-команда должна проводить ежедневный Стендап — собрание, которое длиться не более 15 минут и проходить в одно и то же время каждый день. Цель ежедневного стендапа заключается в том, чтобы все члены команды были в курсе того, что каждый человек сделал с момента последнего стендапа, над чем они собираются работать до следующего стендапа и какие препятствия мешают им завершить свою задачу. Scrum-мастер проводит такие стендапы, и его задача — сообщать о препятствиях и/или помогать их устранять.

Lean разработка программного обеспечения

Lean или Бережливая разработка программного обеспечения основана на принципах бережливого производства, которые направлены на минимизацию отходов и максимизацию ценности для клиента.

В своей простейшей форме бережливая разработка программного обеспечения обеспечивает только то, что нужно клиентам. В книге Lean Software Development: An Agile Toolkit, есть семь принципов бережливого производства:

- Устранить потери;
- Усилить обучение;
- Принимайте решения как можно позже;
- Выполняйте работу как можно быстрее;
- Расширяйте возможности команды;
- Обеспечьте целостность;
- Оптимизируйте максимально.

2 ДИЗАЙН ПАТТЕРНЫ ПРОГРАММ

2.1 Введение

Паттерны проектирования программного обеспечения — это лучшие практические решения для решения общих проблем при разработке программного обеспечения. Модели проектирования не зависят от языка. Это означает, что они могут быть реализованы на любом современном вычислительном языке общего назначения или на любом языке, поддерживающем объектно-ориентированное программирование. Часто популярные паттерны проектирования стимулируют создание дополнительных фреймворков, которые упрощают реализацию в широко используемых языках и парадигмах.

Этот способ мышления о паттернах был быстро подхвачен пионерами объектно-ориентированного кодирования и Agile-разработки программного обеспечения. В 1994 году Эрих Гамма, Ричард Хелм, Ральф Джонсон и Джон Влиссидес (известные под общим названием "Банда четырех" (GoF)) опубликовали книгу под названием "Паттерны проектирования — элементы многократно используемого объектно-ориентированного программного обеспечения" [1]. Я приведу широкий взгляд на паттерны, которые они определили и задокументировали.

Разрабатывать *(программировать)* под интерфейс, а не под реализацию. Жесткая связь логики основной программы с реализацией конкретной функциональности, как правило, делает код трудным для понимания и сопровождения. Опыт показывает, что лучше всего работает свободное соединение логических слоев с помощью абстрактных интерфейсов. Например, основной код вызывает функции и методы в общем виде. Функции нижнего уровня реализуют соответствующие интерфейсы для функциональности, которую они обеспечивают, гарантируя, например, что все функции сериализации, используемые в программе, вызываются одинаково.

Объектно-ориентированные языки, такие как Golang, Java, формализуют эти идеи. Они позволяют явно объявлять интерфейсы, которые могут реализовывать классы. Определение интерфейса — это, по сути, набор прототипов функций, определяющих имена и типы функций и параметров, которые логика более высокого уровня может использовать для вызова ряда классов. Например, интерфейс для ряда классов "транспортных средств" (например, классов "автомобиль", "мотоцикл", "трактор") может включать прототипы `start_engine()`, `stop_engine()`, `accelerate()` и `brake()`.

Предпочитайте композицию объектов наследованию классов. Объектно-ориентированные языки позволяют наследовать: более обобщенные базовые классы могут быть унаследованы производными классами. Таким образом, класс "утка" может унаследовать значительную функциональность от класса "птица". Однако это требует, чтобы класс "птица" реализовывал очень широкий спектр методов, большинство из которых не могут быть использованы конкретным производным классом.

Принцип предпочтения композиции над наследованием предполагает, что лучшей идеей может быть реализация конкретного класса (класс — утка) путем создания только необходимых уникальных подклассов (класс — кряква) вместе с абстрактными интерфейсами (интерфейс — 'как_утка') для классов (класс — муха, класс — плавание), которые могут быть широко использованы аналогичным образом (класс — пингвин реализует интерфейс 'как_пингвин', позволяющий совместно использовать класс 'плавание', но не 'полёт'). Организация программного обеспечения таким образом оказалась наиболее гибкой, в конечном счете, более легкой в обслуживании и поощряет повторное использование кода.

Паттерны проектирования программного обеспечения уже доказали свою успешность, поэтому их использование может ускорить разработку, поскольку разработчикам не нужно придумывать новые решения и проходить через проверку концепции, чтобы убедиться, что они работают.

2.2 Базовые паттерны проектирования

В своей книге "Паттерны проектирования" "Банда четырех" разделила паттерны на три основные категории:

- **Порождающие** — паттерны, используемые для руководства, упрощения и абстрагирования создания программных объектов в масштабе. *Простыми словами, они беспокоятся о гибком создании объектов без внесения в программу лишних зависимостей;*
- **Структурные** — паттерны, описывающие надежные способы использования объектов и классов для различных видов программных проектов. *То есть показывают различные способы построения связей между объектами;*
- **Поведенческие** — паттерны, описывающие, как объекты могут взаимодействовать и работать вместе для решения привычных задач в программной инженерии. Или по-человечески заботятся об эффективной коммуникации между объектами.

Они перечислили в общей сложности 23 паттерна проектирования, которые сегодня считаются основой более новых паттернов проектирования. Большинство этих паттернов на том или ином уровне выражают основные принципы хорошего объектно-ориентированного проектирования программного обеспечения.

Давайте более подробно рассмотрим два наиболее часто используемый паттерн проектирования: паттерн проектирования "Observer" (поведенческий паттерн проектирования.

2.3 Пример — Паттерн Observer

Паттерн проектирования Обсервер или Наблюдатель — это схема уведомления по подписке, которая позволяет объектам (наблюдателям или

подписчикам) получать уведомления о событиях, когда происходят изменения в объекте (субъекте или публикующем), за которым они наблюдают. Примеры паттерна проектирования "наблюдатель" можно найти в современных приложениях. Подумайте о социальных сетях. Пользователи (наблюдатели) следят за другими пользователями (субъектами). Когда субъект публикует что-то в своей социальной сети, он уведомляет всех наблюдателей о том, что появилось сообщение, и наблюдатели просматривают это обновление. Так, например, в игре Minecraft для построения сложных механизмов и детектирования изменений, используется упомянутый выше, не шаблон, а блок Observer представлен на рисунке 5.

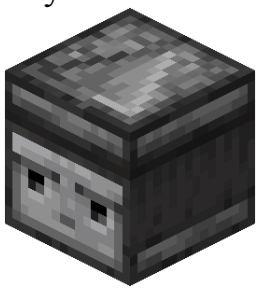


Рисунок 5 — Observer в Minecraft

Чтобы реализовать этот механизм подписки (Рисунок 6):

1. Субъект должен иметь возможность хранить список всех своих наблюдателей;
2. Субъект должен иметь методы для добавления и удаления наблюдателей;
3. Все наблюдатели должны реализовать обратный вызов, который будет вызываться, когда издатель посылает уведомление, предпочтительно с использованием стандартного интерфейса, чтобы упростить задачу для издателя. В этом интерфейсе должен быть объявлен механизм уведомления, который должен иметь параметры, позволяющие издателю отправлять необходимые данные наблюдателю.

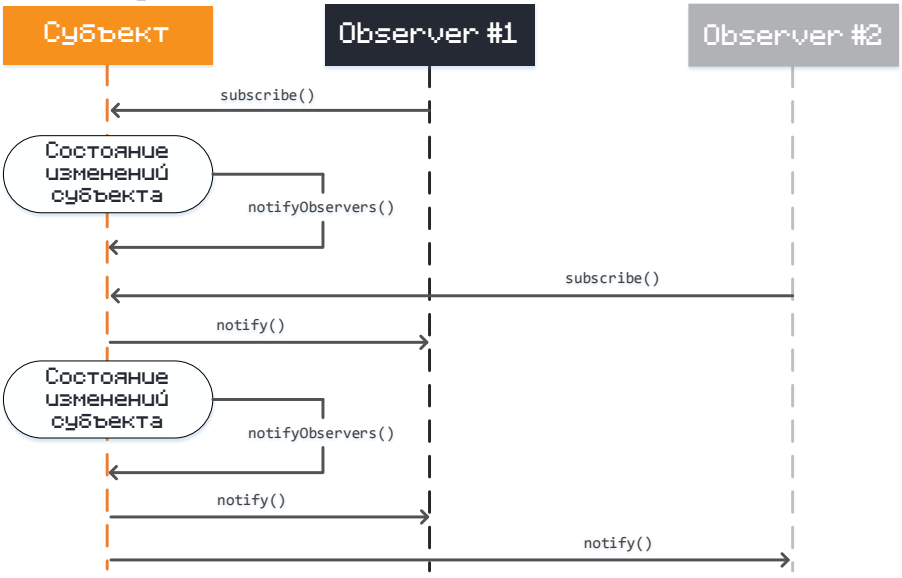


Рисунок 6 — Механизм подписки

Использование этого паттерна проектирования выглядит следующим образом:

1. Observer добавляет себя в список наблюдателей субъекта, вызывая метод субъекта для добавления observer’a.
2. Когда в субъекте происходят изменения, он уведомляет всех наблюдателей в списке, используя обратный вызов (call back) каждого наблюдателя и передавая необходимые данные.
3. Call back наблюдателя запускается и выполняется для запуска “процесса уведомление”.
4. Пункты 2 и 3 повторяются каждый раз, когда происходит изменение в субъекте.
5. Когда наблюдатель закончит получать уведомления, он удаляется из списка наблюдателей субъекта, вызывая метод субъекта для удаления наблюдателя.

Преимущество паттерна проектирования Observer заключается в том, что наблюдатели могут получать данные от субъекта в режиме реального времени, как только происходит изменение. Механизмы подписки всегда обеспечивают лучшую производительность, чем другие варианты, такие как опрос (polling).

Системы Контроля Версий

Контроль версий — это способ управления изменениями в наборе файлов с целью сохранения истории этих изменений. Существует три типа систем контроля версий: Локальная, Централизованная и Распределенная (Рисунок 7).

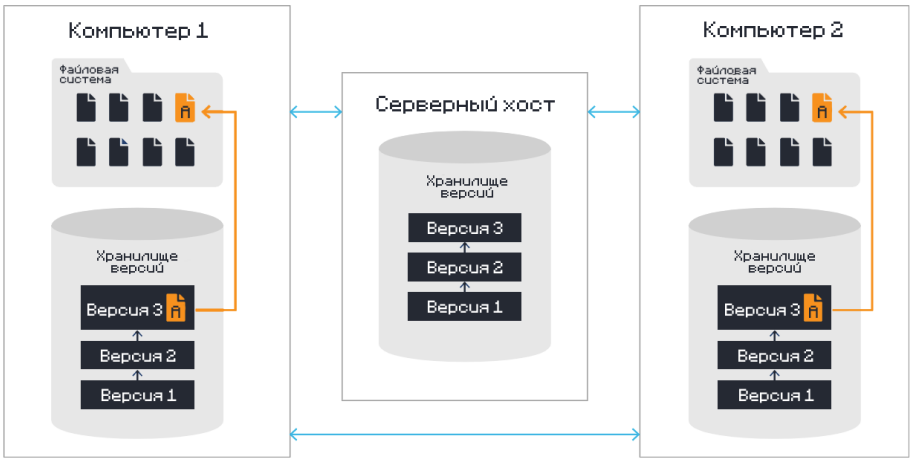


Рисунок 7 — Распределенная система контроля версий

Git — это реализация распределенной системы контроля версий с открытым исходным кодом. Git имеет два типа репозиториев — локальный и удаленный. Ветвление позволяет пользователям работать над кодом независимо, не затрагивая основной код в репозитории. В дополнение к функциональности распределенного контроля версий и управления исходным кодом Git, GitHub также предоставляет дополнительные возможности, такие как: review кода, документация, управление проектами, отслеживание ошибок и запросы функций.

Code Review и тестирование

Code review — процесс просмотра разработчиками просматривают кодовой *баз*ы, подмножество кода или конкретных изменения кода и предоставления по ним обратной связи (Рисунок 8). И уже на этом этапе можно мы столкнуться с рутинными проверками Code Style, Наличие Инъекций, плохих генераторов ПСЧ, автоматизировать которые, можно с помощью любых подходящие для вас инструментов, например, позднее мы рассмотрим Jenkins Pipline [10], в который не составляет труда внедрить соответствующие проверки для любого любимого вами языка программирования.

Рисунок 8 — Процесс Code review

Тестирование программного обеспечения подразделяется на две общие категории: функциональное и нефункциональное. Детальное функциональное тестирование небольших фрагментов кода (строк, блоков, функций, классов и других компонентов в изоляции) — называется модульным тестированием. После модульного тестирования идет интеграционное тестирование, которое позволяет убедиться в том, что все эти отдельные части правильно сочетаются друг с другом, образуя полноценное приложение. Разработка, ориентированная на тестирование (иногда называемая Test-First Development) — это тестирование для подтверждения замысла проекта в свете требований. Это означает написание кода тестирования до написания кода приложения. Выразив требования в тестовом коде, разработчики могут писать код приложения до тех пор, пока он не пройдет тесты.

Вникаем в форматы данных

Для тестирование программного обеспечения требуется стандарт «контракт» связи между его составляющими посредством обмена request/response между API ручками, которые мы рассмотрим в следующей главе. Таковыми выступают форматы:

- XML — использующийся в SOAP архитектурах;
- JSON — использующийся в REST и SOAP архитектурах;
- YAML — использующийся в сценарии автоматизации DockerCompose/Jenkins Pipline/Ansible inventory и др.

3 ВНИКАЕМ И НАЧИНАЕМ ИСПОЛЬЗЛВАТЬ API

3.1 Введение

Если вы планируете автоматизировать свою сеть (или даже только её часть), вам понадобится умение создавать и устранять неполадки в интерфейсах прикладного программирования (API). API определяют способ взаимодействия пользователей, разработчиков и других приложений с компонентами приложения. Поскольку API-интерфейсов так много, вы должны быть уверены, что ж/б понимаете их основы. В этом модуле основное внимание мы уделим REST API.

3.2 Знакомство с API

Что такое API?

API (Application programming interface) — это контракт, который предоставляет программа. «Ко мне можно обращаться так и так, я обязуюсь делать то и это».

Если переводить на человеческий, это было бы слово «договор». Договор между двумя сторонами, как договор на покупку машины:

- мои обязанности — внести такую-то сумму;
- обязанность продавца — дать машину.

Перевести можно, да. Но никто так не делает ¯_(ツ)_/¯

Все используют слово «контракт». *Так принято*. К тому же это слово входит в название стиля разработки:

- Code first — сначала пишем код, потом по нему генерируем контракт;
- Contract first — сначала создаем контракт, потом по нему пишем или генерируем код.

API позволяет одной части программного обеспечения общаться с другой. Его можно сравнить с розеткой. Если бы не было розетки, что бы вам пришлось делать, чтобы включить ноутбук?

- Открыть стену;
- Отсоединить провода;
- Соединить провода;
- Понять все провода в стене.

API определяет, как разработчик может написать одну часть программного обеспечения для взаимодействия с функциями существующего приложения или даже для создания совершенно новых приложений.

API может использовать общие веб-интерфейсы или протоколы связи, а также собственные стандарты. Хорошим примером возможностей использования

API может являться приложение для рекомендаций ресторанов, которое выдает список подходящих ресторанов в данном районе. Вместо того чтобы

создавать картографическую функцию с нуля, приложение интегрирует API стороннего производителя для обеспечения функциональности карты. Создатель API определяет, как и при каких обстоятельствах разработчики могут получить доступ к интерфейсу.

В рамках этого процесса API также определяет, какой тип данных, сервисов и функциональности приложение раскрывает третьим лицам; если API не раскрывает их, то они не раскрываются, и точка. *(То есть, при условии, что безопасность настроена должным образом!)* Предоставляя API, приложения могут контролировать то, что они раскрывают безопасным способом. Схема работы API представлен на рисунке 9.

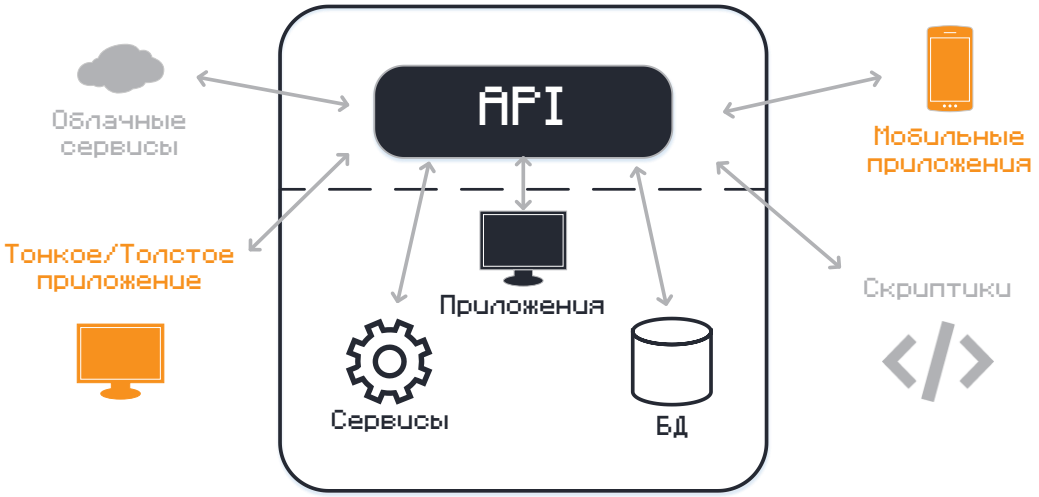


Рисунок 9 — Пример различных типов API-интеграций.

Подумайте об этом как о кнопках на приборной панели вашего автомобиля. Когда вы поворачиваете ключ или нажимаете кнопку ВКЛЮЧИТЬ ДВИГАТЕЛЬ, (Tesla Model X), чтобы завести машину, вы видите, что машина заводится. Вы не видите и даже не заботитесь о том, что двигатель запускает электронную систему зажигания, которая заставляет газ поступать в двигатель, и поршни начинают двигаться. Всё, что вы знаете, это то, что, если вы хотите, чтобы автомобиль завелся, вы поворачиваете ключ или нажимаете кнопку. Если вы нажмете другую кнопку, например, кнопку включения радио, автомобиль не заведется, потому что это не то определение, которое автомобиль (приложение) определил для запуска двигателя. Сам автомобиль (приложение) может делать гораздо больше вещей, чем запуск двигателя, но эти вещи не доступны водителю (стороннему пользователю).

Для чего использовать API?

API обычно создаются для использования другими приложениями, *поэтому их и называют интерфейсами прикладного программирования*. Но они не обязательно должны использоваться исключительно программно; их также могут использоваться людьми, которые хотят взаимодействовать с приложением вручную.

Вот лишь некоторые из многочисленных случаев использования API:

- **Автоматизация** — создание скрипта, который выполняет ваши ручные задачи автоматически. *Пример: Вы — менеджер, и в последний день каждого платежного периода вам нужно вручную войти на портал и загрузить табель учета рабочего времени для каждого сотрудника, который вам подчиняется. Затем вам нужно вручную сложить количество отработанных каждым человеком часов, чтобы определить, сколько ему нужно заплатить. Если вы создадите автоматизированный скрипт, который вызовет API для получения данных из каждого табеля учёта рабочего времени и вычислит общую сумму на основе этих данных, ваш сценарий сможет распечатать список общего количества отработанных часов для каждого человека в удобном для пользователя формате. Подумайте, сколько времени это сэкономит!*
- **Интеграция данных** — приложение может потреблять данные, предоставляемые другим приложением, или реагировать на них. *Пример: Многие сайты электронной коммерции используют платежные сервисы, доступ к которым осуществляется через REST API. Платежный интерфейс позволяет сайту продавца передавать относительно малозначимые данные, такие как описание купленного объекта и цена. Затем пользователь самостоятельно проходит аутентификацию в платежном сервисе для подтверждения платежа. Это позволяет продавцу получать оплату без необходимости прямого доступа к данным кредитной карты клиента. Продавец получает обратно код подтверждения для использования в бухгалтерии.*
- **Функциональность** — приложение может интегрировать функциональность другого приложения в свой продукт. *Пример: Многие онлайн-сервисы, такие как Ozon и Uber, обмениваются данными с Яндекс.Карты и Google Maps соответственно, для создания и оптимизации маршрутов поездок. Они также встраивают функциональность Google Maps в свои собственные приложения и веб-сайты для представления карт маршрутов в реальном времени.*

3.3 Введение в REST API

REST API веб-служб

REST API — это интерфейс программирования, который взаимодействует по протоколу HTTP, придерживаясь принципов архитектурного стиля REST [20]. Пример взаимодействия представлен на рисунке 10. Шесть принципов архитектурного стиля REST таковы:

- Модель клиент-сервер (Client-Server);
- Отсутствие состояния (Stateless);
- Кэширование (Cache);
- Единообразие интерфейса (Uniform Interface);
- Слои (Layered System);

– Код по требованию (Code-On-Demand).

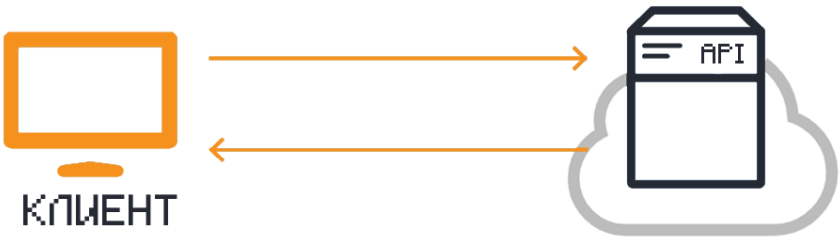


Рис. 10 — Модель запросов/ответов REST API

Поскольку REST API взаимодействуют по протоколу HTTP, они используют те же понятия, что и протокол HTTP:

– Запросы/ответы HTTP;
– HTTP verbs;
– HTTP статус коды;
– HTTP заголовки/тело;

Запросы REST API

Запросы REST API —способ для приложения (клиента) попросить сервер выполнить определенную функцию. Поскольку это API, эти функции предопределены сервером и должны соответствовать предоставленной спецификации.

Запросы REST API состоят из четырех основных компонентов:

– Uniform Resource Identifier (URI);
– HTTP метод;
– Header;
– Body;

Унифицированный идентификатор ресурса (URI)

Унифицированный идентификатор ресурса (Uniform Resource Identifier URI) иногда называют унифицированным локатором ресурса (URL). URI определяет, каким ресурсом хочет манипулировать клиент. REST запрос должен идентифицировать запрашиваемый ресурс; идентификация ресурсов для REST API обычно является частью URI.

URI по сути имеет тот же формат, что и URL, который вы используете в браузере для перехода на веб-страницу. Синтаксис состоит из следующих компонентов в расположенных в определенном порядке:

– Схема;
– Авторитет (Источник);
– Путь;
– Запрос.

Если собрать все компоненты вместе, URI будет выглядеть следующим образом (Рисунок 11): `scheme:[//authority][/path][?query].`

Рисунок 11 — Uniform Resource Identifier

Различные компоненты унифицированный идентификатор ресурса (URI)

Протокол

Схема определяет, какой протокол HTTP должен быть использован. Для REST API возможны два варианта:

- `http` — соединение открыто;
- `https` — соединение безопасно.

Источник

Источник, или место назначения, состоит из двух частей, которым предшествуют две прямые косые черты (//):

- Хост;
- Порт.

Хост — это имя хоста или IP-адрес сервера, предоставляющего REST API (веб-сервис). Порт — это конечная точка связи или номер порта, связанный с хостом. Перед портом всегда ставится двоеточие (:). Обратите внимание, что, если сервер использует порт по умолчанию — 80 для HTTP и 443 для HTTPS — порт может быть опущен в URI.

Путь

Для REST API путь обычно известен как путь к ресурсу и представляет собой местоположение ресурса, данных или объекта, которым нужно управлять на сервере. Путь предваряется косой чертой (/) и может состоять из нескольких сегментов, разделенных косой чертой (/).

Запрос

Запрос, включающий параметры запроса, является необязательным. Запрос предоставляет дополнительные сведения для определения границ, фильтрации или уточнения запроса. Если запрос присутствует, перед ним ставится знак вопроса (?). Для параметров запроса не существует определенного синтаксиса, но обычно они определяются как набор пар ключ-значение, разделенных амперсандом (&). Например:

```
http://example.com/update/person?id=42&email=person%40example.com
```

Метод HTTP

REST API используют стандартные методы HTTP, также известные как HTTP-verbs, как способ сообщить веб-службе, какое действие запрашивается для данного ресурса. Не существует стандарта, определяющего, какой метод

HTTP сопоставлен с каким действием, но предлагаемое сопоставление представлено в таблице 1:

Таблица 1 — Методы HTTP

HTTP Метод	Действие	Описание
POST	Создание	Создайте новый объект или ресурс.
GET	Чтение	Получение сведений о ресурсе из системы.
PUT	Обновление	Замените или обновите существующий ресурс.
PATCH	Частичное обновление	Обновление некоторых сведений из существующего ресурса.
DELETE	Удаление	Удалить ресурс из системы.

Заголовок

REST API используют стандартный формат заголовков HTTP для передачи дополнительной информации между клиентом и сервером, но эта дополнительная информация является необязательной. HTTP-заголовки представляются в виде пар имя-значение, разделенных двоеточием (:), [имя]:[значение].

Существует два типа заголовков: заголовки запросов и заголовки сущностей.

Заголовки запросов

Заголовки запросов включают дополнительную информацию, не относящуюся к содержанию сообщения.

Например, вот типичный заголовок запроса, который вы можете найти для запроса REST API:

Заголовки сущностей

Заголовки сущностей — это дополнительная информация, которая описывает содержимое тела сообщения.

Вот типичный заголовок сущности, который вы можете найти для запроса REST API (Таблица 2).

Таблица 2 — Entity Header

Ключ	Пример значения	Описание
Content-Type	application/json	Укажите формат данных в теле сообщения

Тело

Тело запроса REST API содержит данные, относящиеся к ресурсу, которым клиент хочет манипулировать. Запросы, использующие метод HTTP POST, PUT и PATCH, обычно включают тело. В зависимости от метода, тело является необязательным, но, если данные предоставляются в теле, их тип должен быть указан в заголовке с помощью ключа Content-Type. Некоторые API построены таким образом, чтобы принимать несколько типов данных в запросе.

Ответы REST API

Ответы REST API по сути являются ответами HTTP. Эти ответы передают результаты HTTP-запроса клиента. Ответ может содержать

запрошенные данные или даже информировать клиента о том, что с его запросом возникли проблемы (Таблица 3).

Таблица 3 — Параметры запроса

Ключ	Пример значения	Описание
Authorization	Basic dmFncmFudDp2YWdyYW50	Предоставьте учетные данные для авторизации запроса

В основном, они похожи на запросы, однако состоят из трёх основных компонентов:

- HTTP Статус;
- Заголовок;
- Тело.

HTTP Статус

REST API используют стандартные коды HTTP в ответе, чтобы сообщить клиенту, был ли запрос успешным или неуспешным. Сам код HTTP может помочь клиенту определить причину ошибки и иногда может предоставить предложения по устранению проблемы.

Коды HTTP всегда состоят из трех цифр. Первая цифра — это категория ответа. Две другие цифры не имеют значения, но обычно присваиваются в числовом порядке. Существует пять различных категорий (Таблица 4):

- 1xx — Информационный;
- 2xx — Успешная операция;
- 3xx — Перенаправление;
- 4xx — Ошибка на стороне клиента;
- 5xx — Ошибка на стороне сервера.

Таблица 4 — Распространённые коды HTTP

HTTP Код	Сообщение	Описание
200	OK	Запрос был успешным и обычно содержит полезную нагрузку (тело)
201	Created	Запрос был выполнен, и запрошенный ресурс был создан
202	Accepted	Запрос принят к обработке и находится в процессе выполнения
400	Bad Request	Запрос не будет обработан из-за ошибки в запросе
401	Unauthorized	Запрос не имеет допустимых аутентификационных данных для выполнения запроса
403	Forbidden	Запрос был понят, но отклонен сервером
404	Not Found	Запрос не может быть выполнен, потому что путь к ресурсу запроса не был найден на сервере
500	Internal Server Error	Запрос не может быть выполнен из-за ошибки сервера
503	Service Unavailable	Запрос не может быть выполнен, так как в настоящее время сервер не может обработать запрос

Подробную информацию о каждом статус коде HTTP можно получить из официального реестра статус кодов HTTP, который ведется Управлением по присвоению номеров в Интернете (IANA).

Заголовок

Как и запрос, заголовок ответа также использует стандартный формат заголовков HTTP и также является необязательным. Заголовок в ответе служит для предоставления дополнительной информации между сервером и

клиентом в формате пары имя-значение, разделенной двоеточием (:), [имя]:[значение].
Существует два типа заголовков: заголовки ответа и заголовки сущностей.

Заголовки ответов

Заголовки ответа содержат дополнительную информацию, не относящуюся к содержанию сообщения.
Некоторые типичные заголовки ответа, которые вы можете найти для запроса REST API, включают (Таблица 5):

Таблица 5 — Response Headers

Ключ	Пример значения	Описание
Set-Cookie	JSESSIONID=30A9DN810FQ428P; Path=/	Используется для отправки файлов cookie с сервера
Cache-Control	Cache-Control: max-age=3600, public	Укажите директивы, которым ДОЛЖНЫ подчиняться все механизмы кэширования

Заголовки сущностей

Заголовки сущностей — это дополнительная информация, которая описывает содержимое тела сообщения. Один общий заголовок сущности определяет тип возвращаемых данных (Таблица 6):

Таблица 6 — Entitiy Headers

Ключ	Пример значения	Описание
Content-Type	application/json	Укажите формат данных в теле сообщения

Стили проектирования API

API могут предоставляться синхронно или асинхронно. Синхронные API отвечают на запрос напрямую, обычно предоставляя данные (или другой подходящий ответ) немедленно. Асинхронные API предоставляют ответ, свидетельствующий о том, что запрос был получен, но этот ответ не содержит никаких фактических данных. Сервер обрабатывает запрос, что может занять некоторое время, и посылает уведомление (или запускает обратный вызов) с данными после обработки запроса. После этого клиент может действовать на основе полученных данных.

Аутентификация в REST API

В целях безопасности большинство REST API требуют аутентификации, чтобы случайные пользователи не могли ошибочно или злонамеренно создавать, обновлять или удалять информацию, а также получать доступ к информации, которая не должна быть общедоступной. Без аутентификации REST API позволяет любому получить доступ к функциям и системным сервисам, которые были открыты через интерфейс. Аутентификация — это акт проверки личности пользователя. Пользователь доказывает, что он тот, за кого себя выдает. Авторизация — это подтверждение пользователем того, что у него есть разрешения на выполнение запрошенного действия на данном ресурсе. К распространенным типам механизмов аутентификации относятся Basic, Bearer и API Key. Открытая авторизация, также известная как OAuth, объединяет аутентификацию и авторизацию.

Ограничение скорости API

Использование ограничения скорости API — это способ для веб-службы контролировать количество запросов, которые пользователь или приложение может сделать за определенную единицу времени. К распространенным алгоритмам ограничения скорости относятся: Leaky bucket, Token bucket, счетчик с фиксированным окном и счетчик со скользящим окном (Рисунок 12).

Рисунок 12 — Принцип работы алгоритмов Leaky bucket и Token bucket

Многие API с ограничением скорости добавляют информацию об ограничении скорости в заголовок ответа. Поскольку стандарта не существует, пара ключ-значение, используемая в заголовке, может отличаться в разных API. Некоторые часто используемые ключи включают:

- X-RateLimit-Limit: максимальное количество запросов, которое может быть сделано за определенную единицу времени;
- X-RateLimit-Remaining: Количество оставшихся запросов, которые запросчик может сделать в текущем окне ограничения скорости;
- X-RateLimit-Reset: Время, когда окно ограничения скорости будет сброшено;
- Если лимит скорости превышен, сервер автоматически отклоняет запрос и отправляет HTTP-ответ, информирующий пользователя. Кроме того, обычно ответ, содержащий ошибку "Превышен лимит скорости", также включает в себя значимый код состояния HTTP.

Работа с Webhooks

Webhooks — это обратный вызов HTTP или HTTP POST на указанный URL, который уведомляет ваше приложение, когда определенное действие или "событие" произошло в одном из ваших ресурсов на платформе. Webhooks позволяют приложениям получать данные в режиме реального времени, поскольку они запускаются при определенных действиях или событиях. Благодаря веб-хукам приложения становятся более эффективными, поскольку им больше не требуется механизм опроса.

4 РАЗВЕРТЫВАНИЕ ПРИЛОЖЕНИЙ И БЕЗОПАСНОСТЬ

4.1 Введение в развертывание и безопасность приложений

Вы узнаете, как разработка программного обеспечения, приложений и сетей объединяются, чтобы сделать возможной автоматизацию в огромных масштабах.

Эта глава познакомит вас с вариантами развертывания приложений. Вы узнаете о компонентах конвейера непрерывной интеграции и непрерывного развертывания, включая контейнеры и микросервисы. Вы узнаете о примерах развертывания в тестовых средах и в производственных средах. Вы также узнаете о мерах безопасности развертывания и поймете известные уязвимости.

4.2 Варианты развертывания с помощью различных моделей

Выбор вариантов развертывания

Мы уже познакомились с основными концепциями разработки программного обеспечения и API. Теперь пришло время рассмотреть развертывание приложений.

Даже если вы единственный разработчик, создающий приложение для себя, при развертывании приложения вы должны учитывать множество различных факторов, начиная от создания соответствующего окружения, правильного определения инфраструктуры и заканчивая базовыми концепциями безопасности. Это означает, что разработчики должны не только создавать код приложения, но и заботиться о том, как приложения развертываются, защищаются, эксплуатируются, контролируются, масштабируются и обслуживаются.

Тем временем физическая и виртуальная инфраструктура и платформы, на которых разрабатываются и развертываются приложения, быстро развиваются. Часть этого развития направлена на то, чтобы облегчить жизнь разработчикам и операторам. Например, такие парадигмы платформ, как Containers as a Service и Serverless Computing, призваны позволить разработчикам сосредоточиться на создании основной функциональности приложений, не заботясь о конфигурации, механике, масштабировании и других операциях с базовой платформой [4].

В этой части мы обсудим, где "живет" современное программное обеспечение. Далее рассмотрим основные методы развертывания и тестирования приложений, а также методы и инструменты рабочего процесса для быстрой и эффективной доставки программного обеспечения. Наконец, перейдем к некоторым основам сетевого взаимодействия и безопасности. Все разработчики должны быть знакомы с этими понятиями поскольку это база, которая действительно кормит.

Среды развертывания

Некоторый хаос на ранних стадиях разработки — это нормально, но код должен быть хорошо протестирован к тому времени, когда он попадает к

пользователям. Чтобы это произошло, код должен пройти ряд этапов или сред, повышающих его надежность. Эти среды являются автономными и предназначены для имитации конечного окружения, в которой код будет "жить".

Как правило, в крупных организациях используется четырехуровневая структура: development, testing, staging, и production.

Dvelopment окружение

Dvelopment окружение — это место, где вы занимаетесь разработкой, которого обычно достаточно для управления фундаментальными аспектами инфраструктуры. Может также включать "макетные" ресурсы, которые обеспечивают форму реальных ресурсов, но не их содержание. *Например, у вас может быть база данных с минимальным количеством тестовых записей или приложение, имитирующее вывод удаленной службы. Каждый разработчик обычно имеет свою собственную среду разработки.*

Тестовое окружение

Когда вам кажется, что код готов, вы можете перейти ко второй среде, выделенной для тестирования кода, которая должна быть структурно похожа на конечную производственную среду, даже если она гораздо меньшего масштаба.

Она часто включает инструменты автоматизированного тестирования, такие как Jenkins, CircleCI или Travis CI, а также интеграцию с системой контроля версий, которая должна быть общей для всей команды и может включать инструменты для проверки кода, такие как Gerrit их логотипы представлены на рисунке 13.

Рисунок 13 — Jenkins, CircleCI и Gerrit

Staging окружение

После того как код протестирован, он перемещается в стэйджовую среду. Она должна быть максимально приближена к реальной производственной среде, чтобы код мог пройти окончательное приёмочное тестирование в реалистичной среде.

Production окружение

Наконец, код попадает во всем известный прод, где с ним взаимодействуют конечные пользователи. К этому моменту, код был многократно протестирован и должен быть свободен от ошибок. Сама production среда должна быть рассчитана на ожидаемый трафик, с учётом

всплесков, *резкого повышения активности*, которые могут возникать сезонно или в связи с определенными событиями. Пуш в прод представлен на рисунке 14. Справиться с этими скачками можно при проектировании инфраструктуры. Однако прежде чем рассматривать инфраструктуру, необходимо узнать о различных моделях, которые можно использовать для развертывания программного обеспечения.

Рисунок 14 — Production ready код

Модели развертывания

На заре развития компьютеров не было выбора, как устанавливать программное обеспечение; вы просто устанавливали его на сам компьютер. Сегодня эта модель известная как "bare metal", является одним из множества доступных вариантов, включающих виртуальные машины, контейнеры и новые варианты, такие как serverless вычисления [7].

Bare metal

Самый знакомый и самый простой способ развертывания программного обеспечения — это установка непосредственно на целевой компьютер, или "bare metal". Помимо того, что это самый простой метод, он имеет другие преимущества, например, тот факт, что программное обеспечение имеет прямой доступ к операционной системе и оборудованию. Это особенно полезно в ситуациях, когда вам нужен доступ к специализированному оборудованию, или для приложений высокопроизводительных вычислений (HPC). Структура Bare metal представлена на рисунке 15.

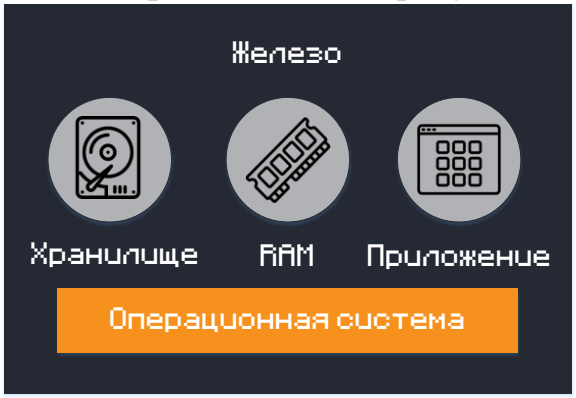


Рисунок 15 — Bare metal

А развертывание на голом железе — это, по сути, развертывание на реальном компьютере. Однако одно место, где "bare metal" может быть недостатком, — это изоляция различных рабочих нагрузок друг от друга. В среде "голого металла" каждое приложение на машине использует одно и то

же ядро, операционную систему, хранилище и т.д. Есть вещи, которые можно сделать, чтобы изолировать некоторые ресурсы, но, если это проблема, другие модели, вероятно, будут лучшим выбором. Кроме того, он не очень гибок в плане ресурсов; машина с 64 ГБ оперативной памяти не станет больше или меньше, если только кто-то физически не разберет её на части, чтобы добавить или удалить оборудование.

Чаще всего используется в качестве инфраструктуры для размещения средств виртуализации и облачных систем.

Виртуальные машины

Одним из способов решения проблем гибкости и изоляции является использование виртуальных машин, или ВМ. Виртуальная машина — это как компьютер внутри компьютера; она обладает собственной вычислительной мощностью, сетевыми интерфейсами и хранилищем.

Использование виртуальных машин преодолевает ряд ограничений. *Например, если у вас есть три рабочие нагрузки, которые вы хотите изолировать друг от друга, вы можете создать три отдельные виртуальные машины на одном "голом" сервере.* Структура ВМ представлена на рисунке 16.

Рисунок 16 — Виртуальные машины совместно используют ресурсы хоста.

Инфраструктура на основе контейнеров

Продвигаясь по лестнице абстракции от виртуальных машин, вы обнаружите контейнеры. Программное обеспечение для создания и управления или оркестрацией контейнеров доступно от Docker, AWS (Elasticized Container Service), Microsoft (Azure Container Service) и других компаний.

Контейнеры были разработаны для обеспечения многих преимуществ виртуальных машин, но с несколько другой архитектурой.

Во-первых, контейнеры предназначены для быстрого запуска, и поэтому они не включают столько базовой программной инфраструктуры. ВМ содержит целую гостевую операционную систему, а контейнер использует

операционную систему хост-машины и специфические для контейнера двоичные файлы, и библиотеки.

Рисунок 17 — Инфраструктура на основе контейнеров

Во-вторых, если виртуальные машины эмулируют целый компьютер, то контейнер обычно представляет собой только приложение или группу приложений. Ценность использования контейнеров заключается в том, что все библиотеки и двоичные файлы, необходимые для запуска приложения, включены, поэтому пользователю не нужно выполнять дополнительные шаги по установке.

Важное различие между контейнером Docker и виртуальной машиной заключается в том, что каждая виртуальная машина имеет свою собственную полную операционную систему. Контейнеры содержат только часть операционной системы. Например, на хост-компьютере Ubuntu Linux может быть запущена ВМ CentOS Linux, ВМ Ubuntu Linux и ВМ Windows 10. Каждая из этих виртуальных машин имеет свою собственную полную ОС. Это может быть очень ресурсоемким для хост-компьютера.

В Docker контейнеры используют одно и то же ядро главного компьютера. Например, на главном компьютере с Ubuntu Linux может быть контейнер Ubuntu Linux и контейнер Centos Linux. Оба этих контейнера используют одно и то же ядро Linux. Однако на этом же хост-компьютере Ubuntu Linux не может быть контейнера под управлением Windows 10, потому что Windows использует другое ядро. Совместное использование одного и того же ядра требует гораздо меньше ресурсов, чем использование отдельных виртуальных машин, каждая из которых имеет собственное ядро.

Контейнеры также решают проблему, возникающую, когда нескольким приложениям для работы требуются разные версии одной и той же библиотеки. Поскольку каждое приложение находится в собственном контейнере, оно изолировано от любых конфликтующих библиотек и двоичных файлов, и не имеют доступа к основной операционной системе. Это повышает безопасность приложений, потому что они не смогут случайно или

умышленно навредить основной системе. Если приложение в контейнере завершится с ошибкой или зависнет, это никак не затронет основную ОС.

Контейнеры также полезны благодаря экосистеме инструментов вокруг них. Такие инструменты, как Kubernetes [6], делают возможной довольно сложную оркестровку контейнеров, а тот факт, что контейнеры часто разрабатываются без статических данных и быстро запускаются, означает, что вы можете экономить ресурсы, не запуская их без необходимости.
Контейнеры также являются основой "облачных" вычислений, в которых приложения, как правило, не имеют статических данных. Отсутствие статичности позволяет любому экземпляру конкретного контейнера обрабатывать запрос. Если добавить это к другому аспекту облачных вычислений, в котором особое внимание уделяется сервисам, становятся возможными serverless вычисления.

4.3 Разработка и развертывание экспериментального приложения

Что такое Docker?

Самый популярный способ контейнеризации приложения — развертывание его в виде контейнера Docker [9]. Контейнер — это способ инкапсуляции всего необходимого для работы приложения, чтобы его можно было легко развернуть в различных средах. Docker — это способ создания и запуска контейнера, а точнее — это формат, включающий в себя несколько различных технологий для создания того, что мы сегодня знаем как контейнеры. К этим технологиям относятся:

- Namespaces — они изолируют различные части запущенного контейнера. Например, сам процесс изолируется в пространстве имен pid (идентификатор процесса), файловая система изолируется в пространстве имен mnt (монтирование), а сетевое взаимодействие изолируется в пространстве имен net.
- Control groups — cgroups являются стандартной концепцией linux, которая позволяет системе ограничивать ресурсы, такие как оперативная память или хранилище, используемые приложением.
- Union File Systems — UnionFS представляют собой файловые системы, которые строятся слой за слоем, объединяя ресурсы.

Docker образ — это набор файлов, доступных только для чтения и не имеющих состояния. Он содержит исходный код, библиотеки и другие зависимости, необходимые для запуска приложения. Docker Контейнер — это исполняемый экземпляр образа Docker. У вас может быть много запущенных контейнеров одного и того же образа Docker. Простым языком, Docker образ — это как рецепт торта, вы можете приготовить столько тортов (контейнеров), сколько захотите.

Образы, в свою очередь, могут храниться в реестрах, таких как Docker Hub. В целом, система выглядит следующим образом (Рисунок 18):

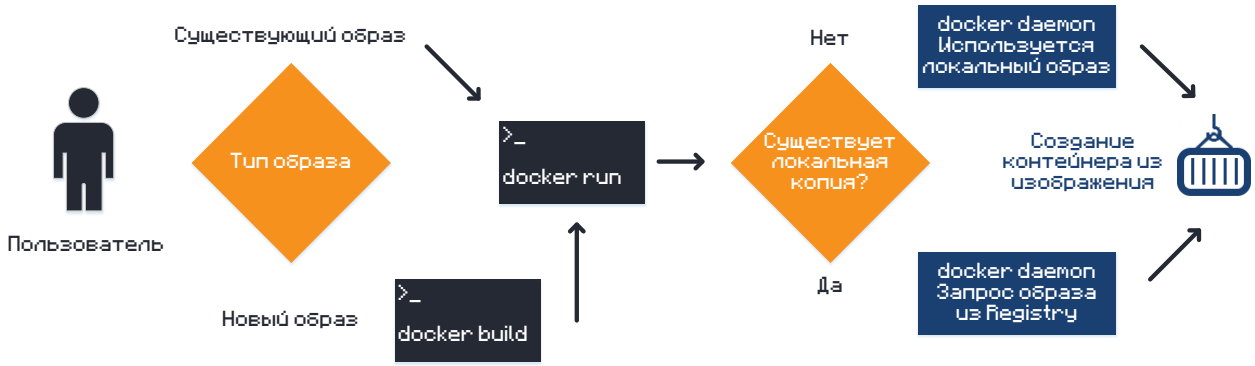


Рисунок 18 — Процесс создания контейнера

Создание контейнера подразумевает извлечение образа или шаблона из репозитория, а затем его использование для создания контейнера.

Таким образом, упрощенная версия рабочего процесса создания контейнера выглядит следующим образом:

Шаг 1. Либо создайте новый образ с помощью docker build, либо возьмите копию существующего образа из реестра с помощью docker pull.

Шаг 2. Запустите контейнер на основе образа с помощью docker run или docker container create.

Шаг 3. Демон Docker проверяет, есть ли у него локальная копия образа. Если нет, он пулит его из registry.

Шаг 4. Демон Docker создает контейнер на основе образа и, если использовался docker run, входит в него и выполняет запрошенную команду.

Как видите, если вы собираетесь создать развертывание примера приложения на основе контейнера, вам придется создать образ. Для этого вам понадобится Dockerfile.

Что такое Dockerfile?

Если вы пользовались таким языком программирования, как C, вы знаете, что он требует компиляции вашего кода. Если да, то вы можете быть знакомы с понятием "makefile". Это файл, который утилита make использует для компиляции и сборки всех частей приложения.

Именно это и делает Dockerfile для Docker. Это просто текстовый файл с именем Dockerfile. Он определяет шаги, которые команда сборки docker должна выполнить для создания образа, который затем может быть использован для создания целевого контейнера.

Анатомия Dockerfile

Конечно, если бы все, что вы могли сделать с помощью Dockerfile, это запустить чистую операционную систему, это было бы полезно, но вам нужен способ начать с шаблона и строить на его основе.

Рассмотрим следующий Dockerfile, который контейнеризирует приложение Python:

```
FROM python
WORKDIR /home/ubuntu
COPY ./sample-app.py /home/ubuntu/.
RUN pip install flask
```

```
CMD python /home/ubuntu/sample-app.py
EXPOSE 8080
```

В приведенном выше Dockerfile объяснение команд выглядит следующим образом:

- Команда FROM устанавливает Python в образ Docker. Она вызывает образ по умолчанию на базе Debian Linux из Docker Hub с установленной последней версией Python;
- Команда WORKDIR указывает Docker использовать /home/ubuntu в качестве рабочего каталога;
- Команда COPY указывает Docker скопировать файл sample-app.py из текущего каталога Dockerfile в /home/ubuntu;
- Команда RUN позволяет напрямую запускать команды на контейнере. В этом примере установлен Flask. Flask - это платформа для поддержки вашего приложения в качестве веб-приложения;
- Команда CMD запустит сервер, когда вы запустите настоящий контейнер. Здесь вы используете команду python для запуска sample-app.py внутри контейнера;
- Команда EXPOSE сообщает Docker, что вы хотите открыть порт 8080. Обратите внимание, что это порт, на котором прослушивается Flask. Если вы настроили свой веб-сервер на прослушивание в другом месте (например, https-запросы на порту 443), здесь нужно это отметить;

Используйте команду docker build для сборки образа.

```
$ docker build -t sample-app-image
```

Локальный запуск Docker контейнера Docker

Теперь, когда образ создан, используйте его для создания нового контейнера и выполнения некоторых действий, введя команду docker run. В этом случае указывается несколько параметров. Параметр -d является сокращением от --detach и говорит о том, что вы хотите запустить его в фоновом режиме. Параметр -P указывает Docker опубликовать его на портах, которые вы указали (в данном случае 8080).

```
$ docker run -d -P sample-app-image
```

Вы можете увидеть контейнер, просмотрев список процессов:

```
$ docker ps
```

Docker также позволяет указать конкретный порт для перенаправления, что позволяет создать более предсказуемую систему:

```
$ docker run -d -p 8080:8080 --name pythontest sample-app-image
$ docker ps
```

Когда ваш контейнер запущен, вы можете войти в него так же, как вы вошли бы в любой физический или виртуальный хост, используя команду exec с хоста, на котором запущен контейнер:

```
$ docker exec -it pythontest /bin/sh
# whoami
root
# pwd
```

```
/var/www/html
# exit
$
```

Чтобы остановить и удалить запущенный контейнер, можно вызвать его по имени:

```
$ docker stop pythontest
pythontest
$ docker rm pythontest
pythontest
$
```

Теперь, если вы снова посмотрите на запущенные процессы, вы увидите, что он исчез.

```
$ docker ps
```

Сохранение образа Docker в Registry

Теперь, когда вы знаете, как создать и использовать свой образ, пришло время сделать его доступным для использования другими людьми. Один из способов сделать это — сохранить его в реестре образов.

По умолчанию Docker использует реестр Docker Hub, но вы можете создать и использовать свой собственный реестр. Для начала вам нужно войти в реестр:

```
$ docker login
Login with your Docker ID to push and pull images from Docker Hub.
If you don't have a Docker ID, head over to https://hub.docker.com
to create one
Username: bonchstudent # Тут должно быть ваше имя пользователя
Password:              # Тут должен быть ваш пароль
WARNING! Your password will be stored unencrypted in
/home/ubuntu/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#creden
tials-store
Login Succeeded
```

Затем вы коммитите запущенный экземпляр контейнера вашего образа. Например, в этом примере запущен контейнер pythontest. Зафиксируйте контейнер с помощью команды docker commit.

```
$ docker commit pythontest sample-app
```

Затем используйте команду docker tag, чтобы присвоить изображению, которое вы зафиксировали, метку или тег. Метка имеет следующий вид:

```
<репозиторий>/<имя_образа>:<метка>
```

Первая часть, хранилище, обычно представляет собой имя пользователя учетной записи, хранящей изображение. В данном примере это bonchstudent. Далее следует имя изображения, и, наконец, необязательный тег. (Помните, если вы не укажете его, изображение будет отображаться как последнее).
В этом примере тег v1:

```
$ docker tag sample-app bonchstudent/sample-app:v1
```

Теперь образ готов к отправке в хранилище:

```
$ docker push bonchstudent/sample-app:v1
```

Отсюда видно, что новое изображение хранится локально:

```
$ docker images
```

4.4 Continuous Integration/Continuous Deployment (CI/CD)

Знакомство с CI/CD

Continuous Integration/Continuous Deployment или как её ужасно переводят Непрерывная интеграция/непрерывное развертывание (CI/CD) — это философия развертывания программного обеспечения, которая занимает видное место в области DevOps. Сам DevOps — это коммуникация и уверенность в том, что все члены команды работают вместе для обеспечения бесперебойной работы.

Continuous Integration

Случалось, ли вам когда-нибудь проделать большую работу над приложением, а когда вы пытались объединить его с основным приложением, возникало множество конфликтов слияния, любой из которых потенциально мог внести серьезные ошибки? Непрерывная интеграция призвана устранить эту проблему.

Идея Continuous Integration заключается в том, что вы и все остальные разработчики проекта постоянно объединяете свои изменения с основной ветвью существующего приложения. Это означает, что любой набор изменений невелик, а вероятность возникновения проблем низка. Если все используют основную ветвь, то каждый, кто проверяет код, будет иметь последнюю версию того, что разрабатывают остальные, а также в ходе автоматизированного тестирования кода перед его слиянием с основной веткой большинство проблем будет отловлено до того, как они станут более серьезной проблемой.

Процесс непрерывной интеграции обеспечивает ряд дополнительных преимуществ, поскольку каждый коммит дает возможность системе выполнять дополнительные задачи. Например, конвейер может быть настроен на выполнение таких задач:

1. Компиляция кода;
2. Выполнение модульных тестов;
3. Статический анализ кода;
4. Интеграционное тестирование;
5. Создание пакетов и версионирование;
6. Публикация пакета версий в Docker Hub или другие репозитории.

Обратите внимание, что бывают ситуации, связанные с большими и сложными изменениями, такими как добавление новых функций, в которых изменения должны быть сгруппированы. В этом случае каждый коммит может запускать только часть конвейера CI, а этапы упаковки и версионирования

будут выполняться только тогда, когда вся функция будет объединена с мастером.

В некоторых случаях адаптация к такому способу работы требует изменения мышления со стороны организации или отдельных разработчиков, которые привыкли работать в своей ветке или в ветках фич. Однако эти изменения необходимы, если вы хотите достичь следующего шага: Continuous Delivery, что не совсем то же самое, что Continuous Deployment [18, 19].

Continuous Delivery

Continuous Delivery — это процесс разработки в спринтах, которые достаточно коротки, чтобы код всегда находился в состоянии, пригодном для развертывания. При непрерывной интеграции небольшие наборы изменений постоянно интегрируются в основную ветвь кода. Непрерывная поставка означает, что эти изменения разрабатываются таким образом, чтобы они были самодостаточными и в любой момент времени можно было развернуть работающее приложение.

Обратите внимание, что Continuous Delivery не означает, что вы постоянно развертываете, этот процесс называется Continuous Deployment. Непрерывная доставка гарантирует, что у вас всегда есть версия, которую вы можете развернуть.

CI/CD — это процесс автоматизированного тестирования и развертывания, который может включать доставку.
Этот процесс говорит нам о двух вещах:

- Вы должны думать о тестировании заранее руководствуясь идеей о "Test Driven Development": вы пишете автоматизированные тестовые процедуры, которые могут быть запущены инфраструктурой CI/CD;
- Если что-то ломается, всё останавливается. Идея этой концепции заключается в том, что при обнаружении ошибки вся разработка останавливается до тех пор, пока она не будет исправлена, и система возвращается в состояние, пригодное для развертывания. Это может быть достигнуто путем поиска и исправления ошибки, либо путем отката изменений до исчезновения ошибки, но важно то, что система должна оставаться развертываемой. *На практике большинство организаций не следуют этой процедуре, но именно эта идея лежит в основе CI/CD.*

Continuous Deployment

Continuous Deployment — это высшее проявление CI/CD. Когда изменения сделаны, протестированы, интегрированы в основную ветку и снова протестированы, они развертываются в продакшн с помощью автоматизации. Это означает, что код развертывается в производство постоянно, а это значит, что ваши пользователи будут вашими последними тестировщиками. Другими словами, Continuous Deployment — это особый тип

Continuous Delivery, в котором каждая сборка, помеченная как готовая к производству, развертывается.

Некоторые организации предпочитают такой тип развертывания, поскольку это означает, что пользователи всегда имеют самый актуальный код. Большинство организаций придерживаются более осторожного подхода, при котором для развертывания кода в производство требуется человек.

Предотвращение воздействия на пользователей

Хотя мы стараемся проводить обширное тестирование в рамках процесса CI/CD, всегда есть вероятность, что плохая сборка пройдет через ворота. Чтобы избежать воздействия на пользователей или, по крайней мере, ограничить его, можно использовать такие стратегии развертывания, как:

- Rolling upgrade — наиболее простой вариант Continuous Delivery, при котором изменения периодически распространяются таким образом, что они не влияют на текущих пользователей, и никому не приходится "переустанавливать" программное обеспечение;
- Canary pipeline — в этом случае новая версия распространяется на подмножество пользователей (или серверов, в зависимости от архитектуры). Если у этих пользователей возникнут проблемы, изменения можно легко откатить. Если эти пользователи не испытывают проблем, изменения распространяются на остальную часть производства [15];
- Blue-green deployment — в этом случае создается совершенно новая среда (синяя) с новым кодом, а старая среда (зеленая) остается в резерве. Если пользователи новой среды испытывают проблемы, трафик может быть перенаправлен обратно в исходную среду. Если в течение определенного времени проблем не возникнет, новая среда становится производственной, а старая среда выводится из эксплуатации для следующего изменения [14].

Преимущества CI/CD

Компании готовы пойти на такие, казалось бы, "радикальные" изменения в своих процессах из-за преимуществ, которые дает использование CI/CD для разработки. К этим преимуществам относятся [2, 3, 18, 19]:

- Интеграция с agile-методологиями — Agile-разработка построена на идее коротких спринтов, после которых команда разработчиков предоставляет функциональное приложение с некоторым подмножеством необходимых функций. CI/CD работает в тех же рамках коротких спринтов. Каждый коммит — это версия концепции "предоставить рабочую версию программного обеспечения";
- Сокращение среднего времени до разрешения проблем (MTTR) — поскольку наборы изменений малы, становится намного проще изолировать ошибки, когда они возникают, и либо исправить их, либо откатить назад и устранить все проблемы;

- Автоматизированное развертывание — с автоматизированным тестированием и предсказуемым развертыванием появляется возможность автоматизированного развертывания. Это означает, что можно использовать такие стратегии развертывания, как канареечное развертывание конвейера релизов, при котором один набор пользователей получает новый набор функций, а остальные - старый. Этот процесс позволяет провести живое тестирование новой функции, чтобы убедиться, что она работает так, как ожидается, прежде чем распространять ее на всю базу пользователей;
- Менее разрушительные релизы функций — поскольку разработка ведется небольшими частями, которые всегда приводят к развертываемому артефакту, можно представлять пользователям постепенные изменения, а не масштабные, которые могут дезориентировать пользователей;
- Улучшенное качество — все эти преимущества ведут к повышению качества программного обеспечения, поскольку оно было тщательно протестировано перед широким внедрением. А поскольку устранять ошибки проще, они, скорее всего, будут устранены своевременно, а не накоплен технический долг;
- Улучшенное время выхода на рынок — поскольку функции можно внедрять по отдельности, они могут быть предложены пользователям гораздо быстрее, чем если бы их пришлось внедрять все одновременно.

4.5 Сети для разработки приложений и безопасности

В наши дни сетевое взаимодействие необходимо учитывать во всех, кроме самых простых, случаях использования. Это особенно верно, когда речь идет об облачных и контейнерных развертываниях. Вот некоторые из приложений, которые необходимо учитывать при развертывании облака [14]:

- Межсетевые экраны;
- Балансировщики нагрузки;
- DNS;
- Reverse прокси.

Межсетевые экраны

Межсетевые экраны или коротко Брандмауэры — это самая основная защита компьютера от несанкционированного доступа со стороны отдельных лиц или приложений. Они могут принимать любые формы, от специального аппаратного устройства до настроек в операционной системе отдельного компьютера.

На самом базовом уровне брандмауэр принимает или отклоняет пакеты на основе IP-адресов и портов, к которым они адресованы. Например, рассмотрим веб-сервер. На этом сервере находится программное обеспечение веб-сервера, а также приложение, представляющее сайт, и база данных,

содержащая содержимое сайта. Без брандмауэра доступ к серверу может быть получен несколькими способами (Рисунок 19):

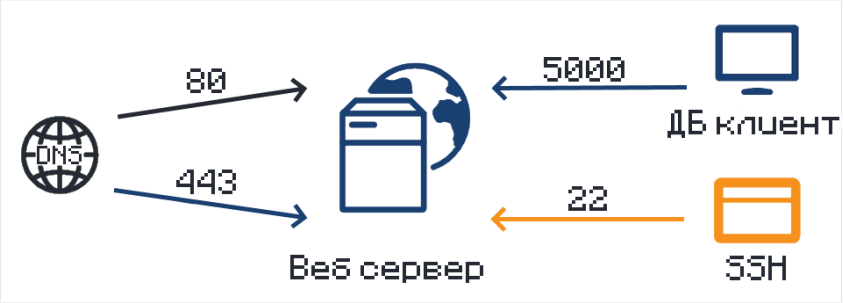


Рисунок 19 — Обращение к серверу

Различные приложения обращаются к серверу разными способами:

- Веб-браузер может получить доступ к веб-приложению с помощью HTTP-запроса на порт 80 или HTTPS-запроса на порт 443;
- Клиент базы данных может получить доступ к базе данных с помощью UDP-запроса на порт 5000;
- SSH-клиент может войти на сам сервер с помощью TCP-запроса на порт 22.

Но действительно ли это то, что вам нужно? Вы определенно хотите получить доступ к веб-приложению, хотя, возможно, вам нужны только HTTPS-запросы. Вы определенно НЕ хотите, чтобы кто-то имел прямой доступ к базе данных; в этом случае такой доступ действительно нужен только веб-приложению. Возможно, вы захотите иметь возможность входить на сервер с помощью SSH-клиента, а не иметь физический доступ к машине.

Чтобы добиться этого, настройте брандмауэр с определенными "правилами", которые наслаиваются друг на друга. Например, у вас может быть следующее правило:

- Запретите доступ всем желающим, используя следующие правила (Рисунок 20)...
    1. Разрешить TCP-запросы к порту 443 от кого угодно
    2. Блокировать все TCP-запросы к порту 22
    3. Блокировать все TCP-запросы к порту 80

(HTTPS, и SSH используют TCP-запросы).

Рисунок 20 — Доступ только по HTTPS

Брандмауэр может разрешать одни соединения и отклонять другие.

Поэтому в данном случае единственным доступом к машине будут HTTPS-запросы. Все остальное будет блокироваться брандмауэром.

В некоторых случаях вам требуется разрешить доступ, но не всем подряд. Например, вы можете настроить свои системы таким образом, чтобы вход в важные системы осуществлялся только с одной машины. Это называется "jump box", и каждый должен сначала войти на этот сервер, а затем с него войти на целевую машину. Jump box можно использовать для предоставления дополнительного доступа, обеспечивая при этом дополнительный уровень безопасности (Рисунок 21).

Рисунок 21 — Jump box

Комбинация Jump бокса и брандмауэра.

Например, если ваш jump box имеет внутренний IP-адрес 172.0.30.42, правила брандмауэра могут выглядеть следующим образом:

– Запретите доступ всем, кроме...
    1. Разрешить TCP-запросы к порту 443 от кого угодно
    2. Разрешить TCP-запросы на порт 22 только с 172.0.30.42
    3. Разрешить UDP запросы на порт 5000 только с 172.0.30.42

При развертывании программного обеспечения необходимо учитывать некоторые моменты, когда речь идет о брандмауэрах:

– Брандмауэры должны предотвращать любой внешний доступ к непроверенному приложению.
– Брандмауэры должны быть настроены так, чтобы приложение могло быть соответствующим образом протестировано. Например, если приложению необходим доступ к версии базы данных для разработчиков, правила брандмауэра должны разрешать это.
– Среда должна быть как можно более близкой копией производственной, чтобы быстро выявить любые проблемы, связанные с конфигурацией брандмауэра.

Обратите внимание, что брандмауэры не только блокируют входящий трафик; они также могут быть настроены на блокировку исходящего трафика. Например, в школах часто устанавливают брандмауэры, которые не позволяют ученикам получать доступ ко всем образовательным сайтам, кроме небольшого количества, используя школьную сеть.

Балансировщик нагрузки

Балансировщик нагрузки делает именно то, о чем говорит: он принимает запросы и "балансирует" их, распределяя между несколькими серверами (Рисунок 22). Например, если у вас 10 серверов, на которых размещено ваше веб-приложение, запросы сначала приходят на балансировщик нагрузки, который затем распределяет их между этими 10 серверами.

Рисунок 22 — Распределение запросов по разным серверам

Балансировщики нагрузки могут принимать решения о том, какие серверы должны получить конкретный запрос, несколькими различными способами. Постоянные сессии — если приложению требуется постоянная сессия, например, пользователю необходимо войти в систему, балансировщик нагрузки будет отправлять запросы на сервер, обрабатывающий эту сессию (Рисунок 23).

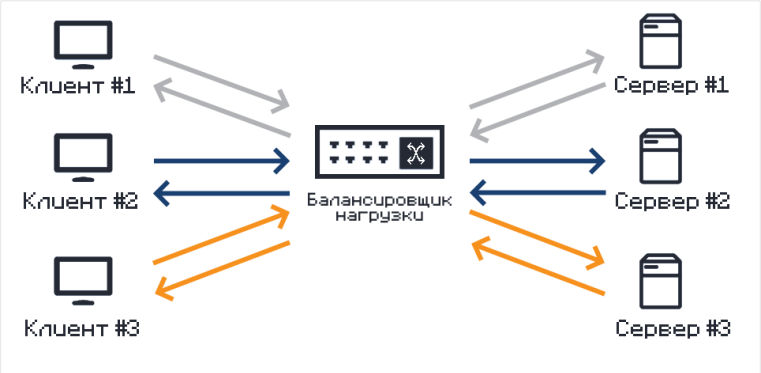


Рисунок 23 — Постоянные сессии

Круговая балансировка — При круговой балансировке нагрузки сервер просто отправляет каждый запрос на "следующий" сервер в списке (Рисунок 24).

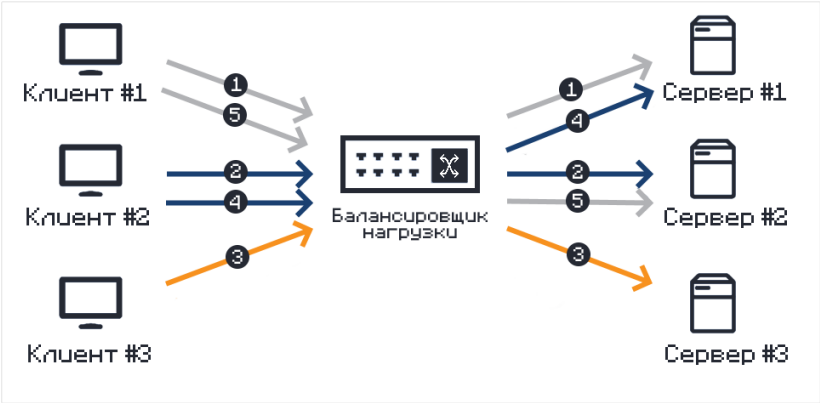


Рисунок 24 — Round robin балансировка

Наименьшее количество соединений — Часто имеет смысл посылать запросы на сервер, который наименее "занят" — имеет наименьшее количество активных соединений. На рисунке 25 сервер 3 получает первый запрос, потому что в настоящее время он не обрабатывает никаких транзакций. Сервер 3 получает второй запрос, потому что у него наименьшее количество активных транзакций. Сервер 1 и сервер 3 теперь имеют по две активные транзакции, поэтому балансировщик нагрузки будет использовать метод round robin. Сервер 1 получает третий запрос, Сервер 3 - четвертый, а Сервер 1 - пятый.

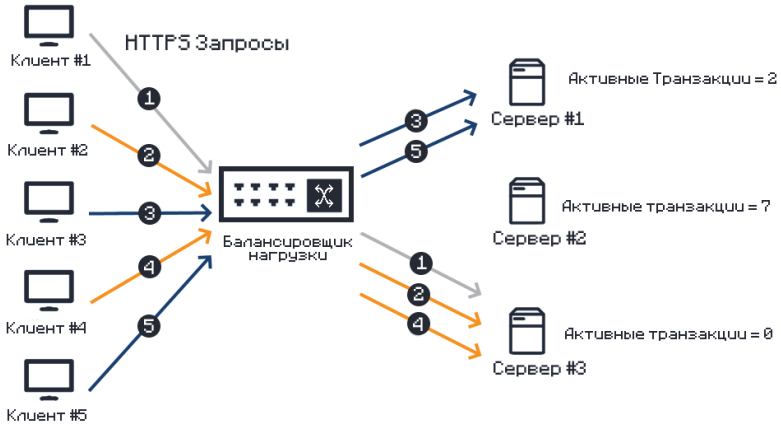


Рисунок 25 — Наименьшее количество соединений

IP Hash — При использовании этого алгоритма балансировщик нагрузки принимает решение на основе хэша (закодированного значения, основанного на IP-адресе запроса). Это можно сравнить с тем, как если бы вы посетили какое-либо мероприятие, и очереди на разные станции формировались бы на основе первой буквы вашей фамилии. Это также простой способ поддерживать последовательность сессий (Рисунок 26).

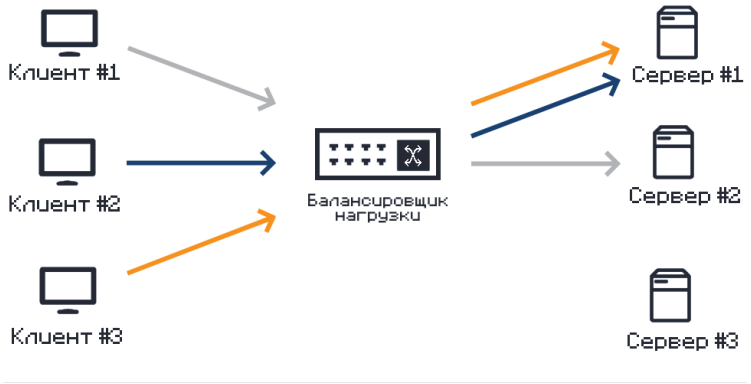


Рисунок 26 — IP-хэш

Для целей развертывания могут использоваться и другие, более сложные алгоритмы. Некоторые из этих примеров включают:

1. **Blue-green deployment** — напомню, что при таком развертывании изменения применяются к новой производственной среде (синей), а не вносятся в существующую производственную среду (зеленую). Балансировщик нагрузки отправляет трафик в синюю среду, когда она готова, а если возникают проблемы, балансировщик нагрузки может отправить трафик обратно в зеленую среду, и изменения могут быть откачены [14].
2. **Canary deployment** — это развертывание начинается с перенаправления небольшой части трафика в синюю среду. Затем балансировщик нагрузки может увеличить количество трафика, перенаправляемого в синюю среду, пока не будут обнаружены проблемы и трафик не вернется в старую среду, либо все серверы и пользователи перейдут в новую среду, а старая будет выведена из эксплуатации или использована для следующего толчка [15].

DNS

DNS, или система доменных имен, — это то, как серверы в Интернете переводят человекочитаемые имена (например, zss.spb.ru или www.example.com) в IP-адреса, доступные для машинной маршрутизации, такие как 74.125.157.99 (для Google) или 208.80.152.201 (для Wikipedia). Эти IP-адреса необходимы для реальной навигации в Интернете (Рисунок 27).

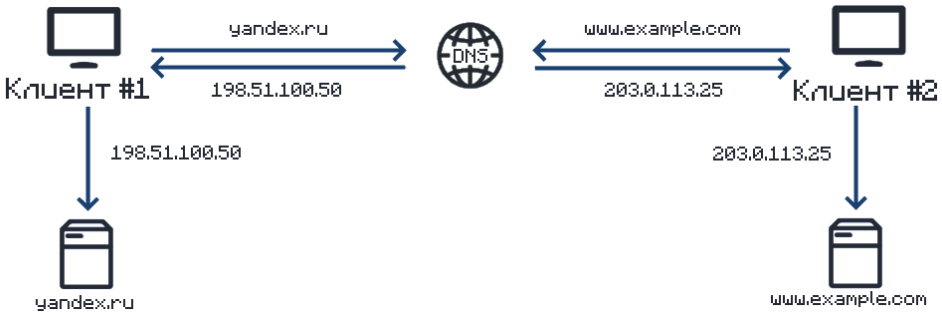


Рисунок 27 — Иллюстрация работы DNS

При развертывании программного обеспечения эта система выгодна тем, что вы можете изменить значение этих адресов. В данном примере приложение ищет базу данных по адресу database.example.com:5000, который находится по IP-адресу 203.0.113.25. В другом примере, вы можете создать версию приложения для разработки, и вы хотите, чтобы она обращалась к версии базы данных, которая находится по адресу 172.24.18.36. (Рисунок 28)

Вы можете настроить машину для разработки на использование DNS-сервера, который указывает database.example.com как 172.24.18.36, протестировать приложение на тестовой базе данных, не внося никаких изменений в приложение.

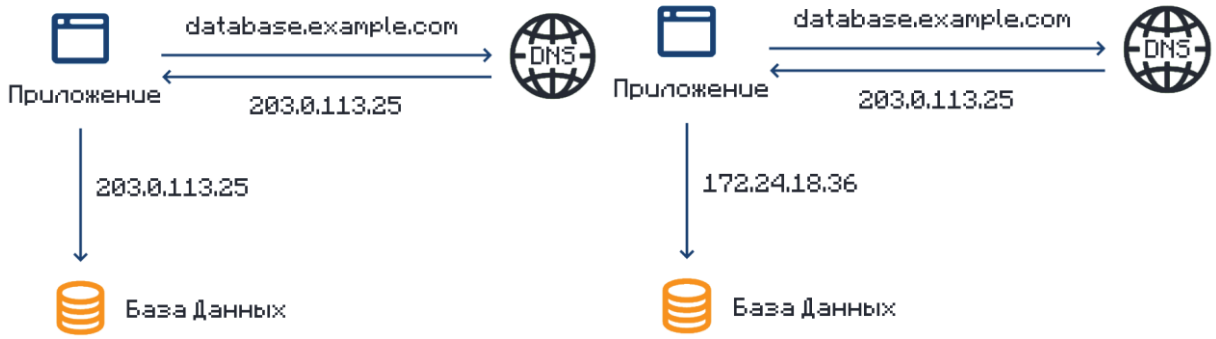


Рисунок 28 — Приложение может указывать на DNS-сервер, который использует различные IP-адреса для ресурсов разработки

Ещё один способ использования DNS в процессе развертывания программного обеспечения — эмуляция некоторых функций, которые может выполнять балансировщик нагрузки. Для этого нужно изменить IP-адрес целевого сервера, когда вы будете готовы к "запуску". (Это не всегда хороший вариант, поскольку изменения DNS могут распространяться через Интернет в целом в течение дня или более).

Reverse прокси

Обратный прокси похож на обычный прокси: однако, если обычный прокси (Рисунок 29) работает для того, чтобы запросы с нескольких компьютеров выглядели так, как будто все они исходят от одного клиента, то обратный прокси работает для того, чтобы ответы выглядели так, как будто все они исходят от одного сервера.

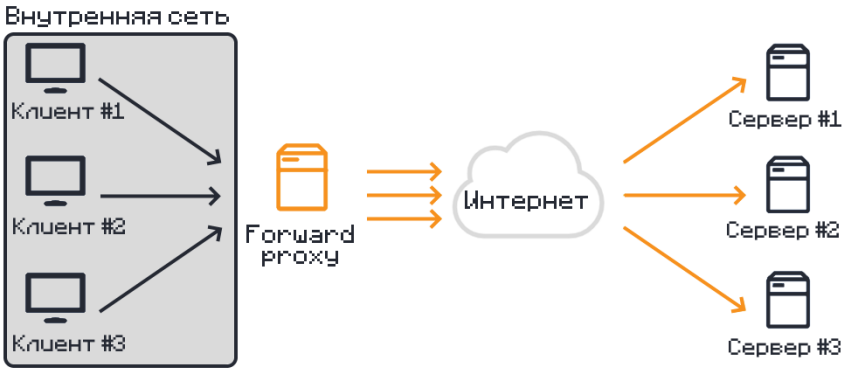


Рисунок 29 — Forward proxy

Прямой прокси-сервер создает впечатление, что все запросы внутри сети приходят с одного и того же IP.

Все запросы, поступающие из внутренней сети, проходят через прокси-сервер, поэтому невозможно узнать, что находится за прокси. Часто этот метод используется для отслеживания или ограничения доступа, как, например, в школьной среде.

В случае с обратным прокси ситуация аналогична (Рисунок 30):

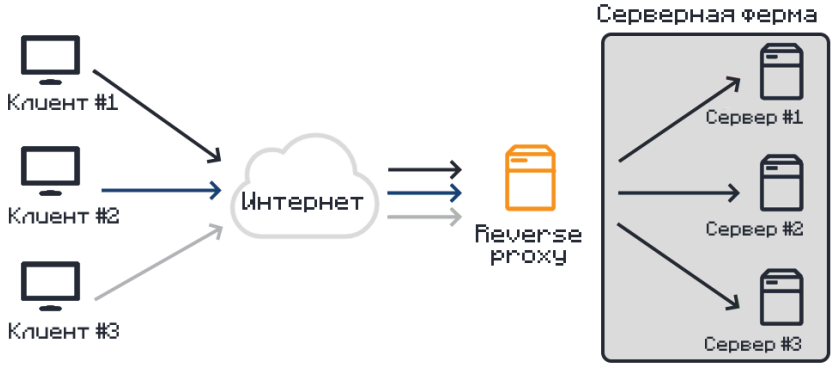


Рисунок 30 — Reverse proxy

Обратный прокси позволяет распределять запросы к одному IP-адресу по разным направлениям внутри сети.

Все запросы в сети поступают к прокси, где они оцениваются и отправляются на соответствующий внутренний сервер для обработки. Как и прямой прокси, обратный прокси может оценивать трафик и действовать соответствующим образом. Таким образом, он похож на брандмауэр или балансировщик нагрузки и может использоваться как брандмауэр.

Поскольку он очень похож на эти функции, обратный прокси может также использоваться для развертывания программного обеспечения аналогичным образом.

Защита приложения

Обеспечение безопасности вашего приложения — это тема, заслуживающая отдельной книги, но в следующих абзацах мы рассмотрим некоторые из наиболее распространенных проблем, на которые следует обратить внимание в процессе развертывания.

Что такое OWASP?

Проект Open Web Application Security Project (OWASP) направлен на предоставление образования, инструментов и других ресурсов, с целью помочь разработчикам избежать некоторых наиболее распространенных проблем безопасности веб-приложений [11, 17]. Ресурсы, предоставляемые OWASP, включают:

– **Инструменты** — OWASP производит такие инструменты, как OWASP Zed Attack Proxy (ZAP), который ищет уязвимости во время разработки, OWASP Dependency Check, который ищет известные уязвимости в

вашем коде, и OWASP DefectDojo, который упрощает процесс тестирования.

- **Проекты** — OWASP выпускает OWASP ModSecurity Core Rule Set (CRS), общие правила обнаружения атак, которые можно использовать с брандмауэрами веб-приложений, и OWASP CSRFGuard, который помогает предотвратить атаки подделки межсайтовых запросов (CSRF).
- **Документацию** — OWASP, пожалуй, наиболее известна своей документацией, которая включают Стандарт проверки безопасности приложений OWASP, OWASP Top Ten, описывающий 10 наиболее распространенных проблем безопасности веб-приложений, и серию шпаргалок OWASP Cheat Sheet, объясняющую, как уменьшить эти проблемы.

Давайте рассмотрим некоторые из наиболее распространенных проблем из этой десятки.

SQL-инъекция

Вы узнали об использовании данных в приложении и о том, как их защитить. Одна из проблем, связанных с использованием данных в приложении, заключается в том, что при взаимодействии с пользователем можно создать потенциально опасную ситуацию.

Например, захотите ли вы когда-нибудь выполнить такое утверждение?

```
<p>
    <code>
        select * from users where username = ‘bob’ and password =
‘pass’; drop table products;
    </code>
</p>
```

Конечно, нет. Потому что в этом случае вы бы удалили таблицу продуктов. Но если вы не будете осторожны, вы можете сделать именно это. Как? Рассмотрим этот пример. Допустим, у вас есть форма (Рисунок 31):

Рисунок 31 — Пустая форма

Пользователь вводит следующие данные (Рисунок 32):

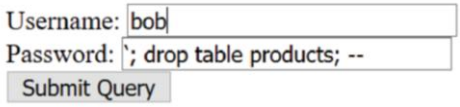


Рисунок 32 — Форма с инъекцией

Это странная вещь для ввода, но проследите пример до конца. Если у вас есть код, который просто интегрирует то, что набрал пользователь, вы получите эквивалент этого:

```
username = "bob"
userpass = "'; drop table products; --"
sqlstatement = "select * from users where username='"+username+"'
and password="'+userpass+"'"
```

Если вы сделаете эту замену, то получите:

```
sqlstatement = "select * from users where username='bob' and
password='; drop table products; --'"
```

В этом случае хакеру даже не нужно вводить правильный пароль для bob (или любое другое имя пользователя, если на то пошло); важно то, что опасный оператор drop table products выполняется независимо от того, что происходит, а двойной тире (--) является комментарием, который не позволяет всему, что находится после него, вызвать ошибку и предотвратить выполнение оператора.

Как предотвратить это? Хотя соблазнительно думать, что можно просто "обеззаразить" вводимые данные, удалив одинарные кавычки ('), это проигрышная битва. Вместо этого предотвратите это с помощью параметризованных операторов. Как этого добиться, зависит от языка и базы данных, но вот как это сделать в Python:

```
with connection.cursor() as cursor:
    cursor.execute("SELECT * FROM users WHERE username =
%(username)s and password = %(userpass)s", {'username':
request.args.get('username'), ‘userpass’:
scrambled(request.args.get('userpass'))})
    result = cursor.fetchone()
```

Делая это таким образом, вы создаете эти строковые переменные, username и userpass, которые динамически вставляются в строку таким образом, что пользователь не может использовать их для создания нескольких утверждений.

Часто такая ситуация возникает при поиске, где пользователь по определению вводит то, что станет частью утверждения базы данных. Рассмотрим этот код:

```
from flask import request
from flask import render_template
sample = Flask(__name__)
DBHOST = 'NOT SET'
def getdb(targethost = ''):
    import MySQLdb
    if targethost == '':
        global DBHOST
```

```
        targethost = DBHOST
    return MySQLdb.connect(host=targethost,
                           user="bonchstudent",
                           passwd="pass",
                           db="products")
@sample.route("/")
def main():
    return render_template("index.html")
@sample.route("/test")
def test():
    return "You are calling me from "+request.remote_addr
@sample.route("/config")
def config():
    return render_template("config.html")
@sample.route("/get_config")
def get_config():
    global DBHOST
    return DBHOST
@sample.route("/config_action", methods=['GET', 'POST'])
def config_action():
    global DBHOST
    DBHOST = request.args.get('dbhost')
    return "Saved database host as "+DBHOST
@sample.route("/search")
def search():
    db = getdb()
    cur = db.cursor()
    search_term = request.args.get('search_term')
    cur.execute("select * from products where title like '%s'" %
search_term)
    output = ""
    for row in cur.fetchall():
        output = output +  str(row[0]) + " -- " + str(row[1]) + "<br
/>"
    db.close()
    return output
if __name__ == "__main__":
    sample.run(host="0.0.0.0", port=80)
```

Как бы вы исправили этот код?

```
...
@sample.route("/search")
def search():
    db = getdb()
    cur = db.cursor()
    search_term = request.args.get('search_term')
    # Тут вы уверены, что запрос рассматривается как строковое
значение
    cur.execute("select * from products where title like
%(search_term)s", {'search_term': search_term})
```

```
    output = ""
    for row in cur.fetchall():
        output = output + str(row[0]) + " -- " + str(row[1]) + "<br
/>"
    db.close()
    return output
if __name__ == "__main__":
    sample.run(host="0.0.0.0", port=80)
```

Межсайтовый скриптинг (XSS)

Атаки межсайтового скриптинга происходят, когда содержимое, отправленное пользователем и не прошедшее санитарную обработку, отображается другим пользователям. Наиболее очевидный вариант такого использования — когда один пользователь отправляет комментарий, содержащий скрипт, выполняющий вредоносное действие, и у всех, кто просматривает страницу комментариев, этот скрипт выполняется на компьютере.

Например, рассмотрим страницу, отображаемую этим кодом:

```
...
@sample.route("/product_comments")
def search():
    db = getdb()
    cur = db.cursor()
    prod_id = request.args.get('prod_id')
    cur.execute("select * from products where id = %(prod_id)s",
{'prod_id': prod_id})
    output = ""
    for row in cur.fetchall():
        output = output +  str(row[0]) + ": " + str(row[1]) + "<br />"
    db.close()
    return output
...
```

Этот код просто извлекает данные о комментариях из базы данных и отображает их на странице. Если бы пользователь по имени Робин отправлял такие материалы, как:

```
<script type="text/javascript">
    alert("Попался!")
</script>
```

Тогда пользователь, пришедший на страницу, получит содержимое, которое выглядит следующим образом:

```
Робин: <script type="text/javascript">alert("Попался!")</script>
```

Когда пользователь загрузит страницу, он увидит окно предупреждения, вызванное вставленным JavaScript.

В данном случае мы просто отображаем предупреждение, что безвредно. Но этот скрипт мог бы с тем же успехом сделать что-то вредоносное, например, украсть куки или еще что-нибудь похуже.

Более серьезная проблема заключается в том, что вы имеете дело не только с данными, хранящимися в вашей базе данных, или "хранимыми XSS-атаками". Например, рассмотрим эту страницу, на которой отображается содержимое параметра запроса:

```
...
<h1>Результаты поиска для {{ request.args['search_term'] }}</h1>
{ for item in cursor }
...
```

Хакер может обманом заставить человека зайти на вашу страницу по ссылке в электронном письме, в параметре которой содержится вредоносный код:

```
http://www.example.com?search_term=%3Cscript%3Ealert%28%27Попался%21%27%29%3C%2Fscript%3E
```

Эта ссылка, включающая "закодированную в url" версию скрипта, приведет к тому, что ничего не подозревающий пользователь увидит страницу:

```
...
<h1>Результаты поиска для <script>alert("Попался!")</script></h1>
...
```

И это, конечно же, приведет к выполнению скрипта. Это называется отраженной XSS-атакой. Как же ее предотвратить?

Основная стратегия заключается в том, чтобы по возможности обеззараживать контент, а если его невозможно обеззаражить, не отображать его вовсе.

Опытные веб-разработчики обычно знают, что нужно проверять вредоносное содержимое в комментариях, но есть и другие места, которые необходимо проверять на наличие "недоверенного" содержимого. OWASP рекомендует никогда не отображать недоверенное содержимое в следующих местах:

- Внутри скриптовых тегов;
- Внутри комментариев;
- Как часть имен атрибутов;
- Как часть имен тегов;

– В CSS (в тегах стилей).

Вы можете отображать содержимое в некоторых местах, если оно предварительно подверглось санитарной обработке. К таким местам относятся:

– В качестве содержимого HTML-тега
– В качестве значения атрибута
– Как переменная в вашем JavaScript

Санирование содержимого может быть сложным процессом, как видно из широкого спектра возможностей, которые есть у злоумышленника. Стоит использовать инструмент, созданный специально для санитарной обработки содержимого, например, OWASP Java HTML Sanitizer, HtmlSanitizer или Python Bleach.

Подделка межсайтовых запросов (CSRF)

Другой тип атаки, имеющий некоторые общие черты с XSS-атаками, — это подделка межсайтовых запросов (CSRF), иногда произносимая как "Sea Surf". В обоих случаях злоумышленник намерен заставить пользователя выполнить код злоумышленника, обычно даже не подозревая об этом. Разница в том, что CSRF-атаки обычно направлены не на целевой сайт, а на другой сайт, на котором пользователь уже прошел аутентификацию.

Вот пример. Допустим, пользователь заходит на сайт своего банка

```
http://sperbank.example.com
```

В другом окне они находятся на странице обсуждения, на которой есть интересная ссылка, и они нажимают на нее.
К сожалению, ссылка была для

```
http://sperbank.example.com/changeemail?new_email=attacker@example.com
.
```

Браузер думает, что это обычная ссылка, поэтому он вызывает URL, отправляя cookie для sperbank.example.com, которые, как вы помните, включают учетные данные пользователя для аутентификации. Банк считает, что этот запрос поступил от пользователя, и выполняет изменение. Теперь злоумышленник может пойти дальше и изменить пароль пользователя, а затем войти в банк и нанести ему любой ущерб.

Обратите внимание, что даже если пользователь достаточно умен, чтобы не нажимать на подобную странную ссылку, если сайт уязвим для XSS-атак, эта атака может быть осуществлена без необходимости пользователя что-либо делать. Тщательно созданный тег `<img>` может достичь того же результата.

Интересным аспектом CSRF является то, что атакующий никогда не получает результатов атаки; он может судить о результатах только постфактум, и он должен быть в состоянии предсказать, каковы будут последствия, чтобы воспользоваться преимуществами успешной атаки.

Атаки CSRF, как известно, трудно предотвратить, но не невозможно. Одним из методов является включение скрытого маркера, который должен сопровождать любые запросы пользователя. Например, форма входа в банк может выглядеть следующим образом:

```
...
<form action="https://sperbank.example.com" method="POST">
    Username: <input type="text" name="username" style="width:
200px" />
    Password: <input type="text" name="password" style="width:
200px" />
    <br >
        <input type="hidden" name="CSRFToken" value="d063937d-
c418-46e6-8354-6f5d8faff095" />
        <input type="submit" value="Log in">
</form>
...
```

Этот CSRFToken должен сопровождать каждый запрос пользователя, чтобы он считался легитимным. Поскольку злоумышленник не может предугадать этот токен, запрос типа:

```
https://sperbank.example.com/changeemail?new\_eamail=attacker@example.
com
```

будет автоматически отклонен.

4.6 OWASP Top Ten на 2021 год

Теперь, когда вы знаете о трёх самых известных атаках, вот весь список OWASP Top 10 за 2021, не за 2022 т.к он всё еще идет на момент написания дипломной работы [11].

- **A01:2021-Broken Access Control** — Контроль доступа обеспечивает соблюдение политики таким образом, чтобы пользователи не могли действовать за пределами своих полномочий. Неисправности обычно приводят к несанкционированному раскрытию информации, модификации или уничтожению всех данных или выполнению бизнес-функции за пределами полномочий пользователя.
- **A02:2021-Cryptographic Failures**. — необходимо определить потребности в защите данных, если эти данные подпадают под действие законов о конфиденциальности, например, Общего регламента ЕС о защите данных (GDPR), или нормативных актов, например, о защите финансовых данных, таких как стандарт PCI Data Security Standard (PCI DSS).

- **A03:2021-Инъекции** — Этот пункт состоит из всевозможных инъекционных атак. Все базы данных, потенциально уязвимы. Фактически, любое действие, которое зависит от ввода данных пользователем, уязвимо, включая прямые команды. Вы можете “смягчить” эти типы атак, используя параметризованные API, экранируя пользовательский ввод, а также используя предложения LIMIT для ограничения уязвимости в случае нарушения.
- **A04:2021-Insecure Design** — это **новая категория** для **2021** года, в которой основное внимание уделяется рискам, связанным с недостатками дизайна. Если мы действительно хотим “двигаться влево” как индустрия, то это требует более широкого использования моделирования угроз, безопасных моделей и принципов проектирования, а также эталонных архитектур.
- **A05:2021-Security Misconfiguration** — 8 пунктов с помощью которых приложению будет считаться с “Нарушением конфигурации системы безопасности”. Без согласованного, повторяющегося процесса конфигурирования безопасности приложений системы подвергаются повышенному риску.
- **A06:2021-Vulnerable and Outdated Components** — использование компонентов с известными уязвимостями. Это единственная категория, в которой нет общих уязвимостей и экспозиций (CVE), сопоставленных с включенными в нее CWE.
- **A07:2021-Identification and Authentication Failures** — теперь она включает CWE, которые больше связаны с ошибками идентификации.
- **A08:2021-Software and Data Integrity Failures** — ещё одна **новая категория** для **2021** года, сфокусированная на принятии предположений, связанных с обновлениями программного обеспечения, критическими данными и CI/CD конвейерами без проверки целостности.
- **A09:2021-Security Logging and Monitoring Failures** — сбои в этой категории могут напрямую влиять на видимость, оповещение об инцидентах и криминалистику; призвана помочь в обнаружении, эскалации и реагировании на активные нарушения. Без регистрации и мониторинга невозможно обнаружить нарушение.
- **A10:2021-Server-Side Request Forgery** — SSRF возникает, когда веб-приложение получает удаленный ресурс без проверки URL, предоставленного пользователем.

  Поскольку современные веб-приложения предоставляют конечным пользователям удобные функции, получение URL-адреса становится обычной задачей. В результате частота возникновения SSRF растет. Кроме того, серьезность SSRF становится выше из-за облачных сервисов и сложности архитектур.

5 ИНФРАСТРУКТУРА И АВТОМАТИЗАЦИЯ

5.1 Автоматизация инфраструктуры

Мы узнали об автоматизации масштабирования. Теперь пришло время научиться автоматизировать инфраструктуру сетей. Это серьезное изменение в том, как специалисты думают о сетях, от проектирования до создания и управления.

Конечно, существует множество инструментов для автоматизации различных задач, а также специализированные инструменты для моделирования сети для тестирования. Но ещё более ценным для сетевых инженеров является скорость с точностью и безопасными возможностями самообслуживания. Puppet, Ansible и Chef отлично подходят для этих целей.

Почему нам нужна автоматизация?

Предприятия конкурируют и контролируют затраты, работая быстро и имея возможность масштабировать свою деятельность. Скорость и маневренность позволяют предприятию изучать, экспериментировать и использовать возможности, опережая конкурентов. Масштабирование операций позволяет предприятию эффективно захватывать долю рынка и приводить мощности в соответствие со спросом.

Разработчикам необходимо ускорить все этапы создания программного обеспечения: кодирование и итерации, тестирование и постановку. Практика DevOps требует от разработчиков развертывания и управления приложениями в производстве, поэтому разработчикам также следует автоматизировать эти действия. Практика DevSecOps помимо этого охватывает аспекты безопасности, рассмотренные нами в предыдущей главе.

Ниже перечислены некоторые риски, которые могут возникнуть в средах, развертываемых и управляемых вручную.

Ограничения, связанные с ручными операциями

Создание простого монолитного сервера веб-приложений может занять у практикующего ИТ-оператора 30 минут и более, особенно при подготовке к работе в производственных средах, что же говорить о микросерверной архитектуре. Когда этот процесс умножается на десятки или сотни корпоративных приложений, множество физических объектов, центров обработки данных и/или облаков, ручные процессы в какой-то момент приведут к сбою или даже отказу сети. Это увеличивает расходы и замедляет бизнес.

Ручные процессы, такие как:

- ожидание доступности инфраструктуры,
- ручная настройка и развертывание приложений,
- обслуживание production систем,

являются медленными и очень трудно масштабируемыми. Они могут помешать вашей команде предоставлять новые возможности коллегам и клиентам. Ручные процессы всегда подвержены человеческим ошибкам, а документация, предназначенная для людей, часто бывает неполной и

неоднозначной, её трудно тестировать, и она быстро устаревает. Это затрудняет разработку и использование накопленных знаний об известных хороших конфигурациях и лучших практиках в больших организациях и их разрозненных инфраструктурах.

Перебои в работе сервера в Facebook представлены на рисунках 35 [12, 13].

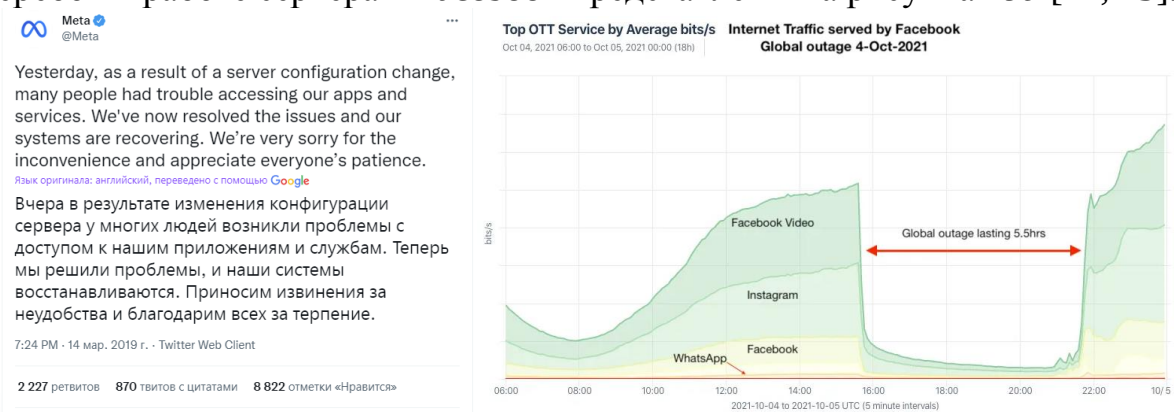


Рисунок 35 — Сбой в работе Facebook 2019 и 2021

Риски использования зависимости

Современная экосистема программного обеспечения децентрализована. Разработчикам больше не нужно создавать и управлять монолитными решениями полного стека. Вместо этого они специализируются на создании отдельных компонентов в соответствии со своими потребностями и интересами. Разработчики могут смешивать и подбирать другие компоненты, инфраструктуру и сервисы, необходимые для создания комплексных решений и их эффективной эксплуатации в масштабе.

Эта современная экосистема программного обеспечения объединяет работу сотен тысяч независимых авторов, каждый из которых получает выгоду от участия в этом обширном сотрудничестве. Участники могут свободно обновлять свои работы в зависимости от потребностей и возможностей, что позволяет им быстро выводить на рынок новые функции, исправлять ошибки и повышать безопасность.

Ответственные разработчики пытаются предвидеть и минимизировать влияние обновлений и новых релизов на пользователей, строго придерживаясь стандартов, намеренно разрабатывая обратную совместимость, обязуясь обеспечить долгосрочную поддержку ключевых версий продукта (например, версии "LTS" дистрибутива Ubuntu Linux) и другие лучшие практики.

Эта экосистема вводит новые требования и новые риски:

1. Компоненты должны быть способны работать вместе со многими другими компонентами во многих различных ситуациях, не отдавая предпочтения конкретным компонентам-компаньонам или архитектурам больше, чем это абсолютно необходимо;
2. Разработчики компонентов могут отказаться от поддержки устаревших функций и редко встречающихся интеграций. Это нарушает процессы, которые зависят от этих функций. Также сложно или невозможно

провести исчерпывающее тестирование релиза, учитывая каждую конфигурацию;

3. Настройки приложений с зависимостями имеют тенденцию замыкаться на хрупких и все более небезопасных стеках развертывания. Они фактически становятся монолитом, которым нелегко управлять, улучшать, масштабировать или переносить на новые, возможно, более экономичные инфраструктуры. Обновления и исправления могут быть отложены, потому что изменения рискованно применять и трудно отменить.

DevOps и SRE

Чтобы full-stack автоматизация была действительно эффективной, она требует изменений в организационной культуре, включая разрушение исторически сложившихся границ между отделами разработки (Dev) и эксплуатации (Ops). DevOps развивался и продолжает развиваться во многих местах параллельно. Некоторые ключевые события сформировали эту дисциплину в том виде, в котором мы знаем ее сегодня [2, 3, 18, 19].

Ключевой момент 1: Site Reliability Engineering (SRE). Роль SRE призвана объединить дисциплины и навыки Dev и Ops.

Ключевой момент 2: Дебуа и "Agile Infrastructure" Дебуа был сторонником автоматизации виртуальной инфраструктуры, использования контроля версий для хранения кода развертывания инфраструктуры и применения методов Agile для разработки и обслуживания решений на уровне инфраструктуры.

Ключевой момент 3: Оллспоу и Хаммонд. Джон Оллспоу и Пол Хаммонд выступили с презентацией на VelocityConf в 2009 году. В ней описывались автоматизация, командная работа, разделение ответственности, прозрачность, доверие, взаимная подотчетность и практика коммуникаций.
DevOps/SRE имеют много основных принципов и лучших практик:

1. Фокус на автоматизации;
2. Идея о том, что "Сбои — это нормально";
3. Переосмысление понятия "доступность" с точки зрения того, что может выдержать бизнес.

Зачем нужна Full-Stack автоматизация?

Автоматизация инфраструктуры может обеспечить множество преимуществ. Они сводятся к скорости, воспроизводимости, повышнию информационной безопасности посредством, снижения риска ошибок, вызванных человеческим фактором, а также сохранение доступности к информационным ресурсам, модернизации ИТ в целом за счет ускорения разработки новых приложений и возможности работать в больших масштабах при снижении рисков. Автоматизация является ключевым компонентом функциональной программно-определяемой инфраструктуры и распределенных и динамичных приложений. Ниже перечислены дополнительные преимущества Full-Stack автоматизации.

Самообслуживание

Автоматизированные механизмы самообслуживания позволяют пользователям запрашивать инфраструктуру по требованию, включая:

- Стандартные компоненты инфраструктуры, такие как экземпляры баз данных и конечные точки VPN.
- Платформы для разработки и тестирования
- Усиленные веб-серверы и другие экземпляры приложений, а также изолированные сети и защищенный доступ в интернет, которые делают их полезными, безопасными и устойчивыми к ошибкам
- Аналитические платформы, такие как Apache Hadoop, Elastic Stack, InfluxData и Splunk.

Масштабирование по требованию

Приложения и платформы должны иметь возможность масштабироваться вверх и вниз в зависимости от требований к трафику и рабочей нагрузке и использовать разнородные мощности. В качестве примера можно привести серийное масштабирование из частного облака в общедоступное и соответствующее формирование трафика. Облачные платформы могут предоставлять возможность автоматического масштабирования (autoscale) виртуальных машин, контейнеров или рабочих нагрузок на бессерверной основе.

Наблюдаемость (Observability)

Наблюдаемая система позволяет пользователям сделать вывод о внутреннем состоянии сложной системы по ее выходным данным. Достигается путем мониторинга платформы и приложений за счет проактивного производственного тестирования на предмет выявления отказов и проблем с производительностью. Но в динамичной работе, включающей автомасштабирование и другие модели поведения приложений, сложность возрастает, и объекты становятся эфемерными. В недавнем отчете поставщика платформы наблюдаемости DataDog говорится, что среднее время жизни контейнера при оркестровке составляет всего 12 часов; микросервисы и функции приложения могут жить всего несколько секунд. Сделать эфемерные сущности наблюдаемыми и протестировать их в производстве можно только с помощью автоматизации. Стандартный набор инструметов представлен ниже рисунок 36:

Рисунок 36 — Grafana, Thanos и Prometheus

Автоматизированное устранение проблем

Некоторые разработчики программного обеспечения и Observability-эксперты рекомендуют так называемую хаос-инженерию. Эта философия основана на утверждении, что неудачи — это нормально: при масштабировании приложений некоторые части всегда выходят из строя. В связи с этим приложения и платформы должны быть спроектированы таким образом, чтобы:

- Минимизировать последствия проблем: Быстро распознавать проблемы и перенаправлять трафик на альтернативные мощности, гарантируя, что конечные пользователи не будут серьезно затронуты, а оперативный персонал не будет вызываться без необходимости.
- Самовосстановление: Распределяйте ресурсы в соответствии с политикой и автоматически перераспределяйте вышедшие из строя компоненты по мере необходимости, чтобы вернуть приложение в здоровое состояние в текущих условиях.
- Мониторинг событий: Запоминайте все, что привело к инциденту, чтобы можно было запланировать исправления и провести вскрытие.
- Некоторые сторонники хаос-инженерии даже рекомендуют использовать средства автоматизации, чтобы вызвать контролируемые (или случайные) сбои в производственных системах. Это постоянно заставляет разработчиков и операторов предвидеть проблемы и повышать устойчивость и способность к самовосстановлению. Проекты с открытым исходным кодом, такие как Chaos Monkey, и платформы "Failure-as-a-Service", такие как Gremlin, специально созданы для того, чтобы ломать вещи, как случайным, так и более контролируемым и эмпирическим образом. Развивающаяся дисциплина называется "тестирование с помощью инъекций сбоя".

5.2 Инструменты автоматизации

Введение в инструменты автоматизации

В этой теме мы рассмотрим три наиболее популярных инструмента автоматизации: Ansible, Puppet и Chef.

Что делают для нас инструменты автоматизации?

Инструменты автоматизации, такие как Ansible, Puppet или Chef, предлагают мощные возможности по сравнению с ситуативными стратегиями автоматизации с использованием BASH, Python или других языков программирования.

Упрощение и стандартизация

Инструменты автоматизации "оборачивают" утилиты операционной системы и функции API для упрощения и стандартизации доступа. Часто они также устанавливают интеллектуальные значения по умолчанию, которые ускоряют составление и тестирование кода. Они делают код,

ориентированный на инструменты, менее многословным и более понятным, чем сценарии.

Модули инструментов автоматизации позволяют использовать лучшие практики, которые делают код более безопасным, а идемпотентность — более простой. Например, многие функции Ansible могут создавать резервные копии конфигурационных файлов на целевых системах или извлекать копии и хранить их локально на машине развертывателя перед внесением изменений. Это помогает обеспечить восстановление в случае сбоя или прерывания развертывания.

Ускорение разработки с помощью готовых функций

Инструменты автоматизации обычно предоставляют очень мощную функциональность для ускорения разработки. Например, по умолчанию Ansible 2.4+ предоставляет функциональность, позволяющую легко получать снапшоты конфигурации из сетевых компонентов Cisco ACI. Он также имеет дополнительные функции, позволяющие откатывать конфигурации ACI к предыдущему состоянию снапшотов.

Облегчение повторного использования, разделение проблем, повышение безопасности

Современные средства автоматизации стремятся быть "управляемыми данными" и обеспечивают следующее:

- Компиляция определений переменных
- Инвентаризация сервера в виде структурированных данных и других деталей отдельно от общего кода
- Упорядоченные средства для внедрения значений переменных в код, шаблоны конфигурационных файлов и другие места назначения во время выполнения.

Например, Ansible Vault поддерживает сложные функции для шифрования конфиденциальных файлов и переменных, безопасного редактирования зашифрованного содержимого файлов и многое другое.

Обнаружение и управление устройствами

Средства автоматизации обычно собирают информацию с целевых устройств по умолчанию в ходе обычной работы. Эта информация включает конфигурацию оборудования, настройки BIOS, операционную систему, конфигурацию сетевых и других периферийных карт, и подсистем, установленные приложения и другие сведения.

Ручное масштабирование

Большинство средств автоматизации могут работать как в локальном режиме, так и в режиме клиент-сервер или распределенного агента. Это позволяет инструменту управлять тысячами или десятками тысяч узлов.

Вовлечение сообщества

Большинство популярных инструментов доступны в основных версиях с открытым исходным кодом, что помогает сообществу ускорить разработку, найти и исправить ошибки.

Важнейшие концепции

Идемпотентность: обзор

Идемпотентное программное обеспечение дает один и тот же желаемый результат при каждом запуске. В программном обеспечении для развертывания идемпотентность обеспечивает конвергенцию и композитивность. Идемпотентные компоненты развертывания позволяют:

- Легче собирать компоненты в коллекции, которые создают новые виды инфраструктуры и выполняют новые операционные задачи;
- Выполнять целые коллекции сборки/развертывания для безопасного устранения небольших проблем с инфраструктурой, выполнения инкрементных обновлений, изменения конфигурации или управления масштабированием.

Обеспечение vs Конфигурация vs Деплой vs Оркестрация

Специалисты по эксплуатации склонны мыслить в терминах иерархии уровней инфраструктуры и связанных с ними задач-доменов:

- **Обеспечение** — получение вычислительной, складской и сетевой инфраструктуры (реальной или виртуальной), обеспечение связи, введение её в эксплуатацию и обеспечение готовности к использованию операторами и разработчиками.
- **Конфигурирование** — означает установку базовых приложений и сервисов, а также выполнение операций, задач и тестов, необходимых для подготовки платформы низкого уровня к развертыванию приложений или платформы более высокого уровня.
- **Деплой** — включает создание, организацию, интеграцию и подготовку многокомпонентных приложений или платформ более высокого уровня, часто на нескольких узлах.
- **Оркестрация** — может означать несколько вещей. В конкретном смысле она обычно относится к встроенной пользователем или платформой автоматизации, направленной на управление жизненным циклом рабочей нагрузки и динамическое реагирование на изменяющиеся условия, особенно в контейнерных средах. В абстрактном смысле это понятие может относиться к процессам или рабочим процессам, которые связывают задачи автоматизации для получения бизнес-преимуществ, например, самообслуживания.

Популярные инструменты автоматизации

В настоящее время наиболее популярными инструментами являются Ansible, Puppet и Chef. Они обладают следующими характеристиками:

1. Относительно просты в освоении;

2. Доступны open-source версии;
3. Плагины и адаптеры позволяют прямо или косвенно управлять многими типами ресурсов: программно-определяемой пустой инфраструктурой (Cisco UCS, устройства Cisco), частным облаком (VMware, OpenStack) и публичным облаком (Amazon Web Services, Microsoft Azure, Google Cloud Platform).

Ansible

Ansible, вероятно, является наиболее популярным из современных решений для автоматизации. Он доступен с открытым исходным кодом, а также в версии с дополнительными функциями от IBM/Red Hat, которая называется Ansible Tower. Его название происходит от романов писательницы Урсулы К. ЛеГуин, в которых "ансибл" — это технология будущего, позволяющая мгновенно общаться на космических расстояниях.

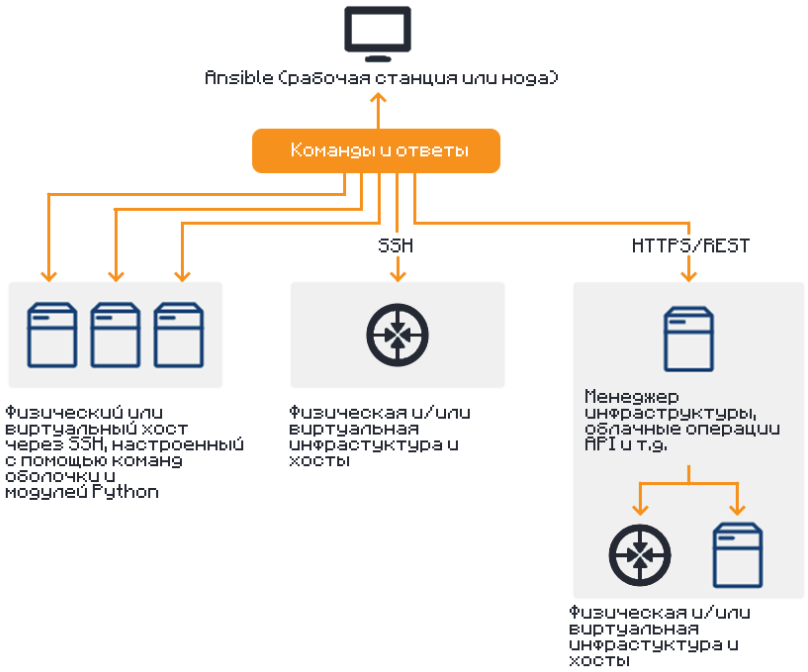


Рисунок 37 — Архитектура Ansible

Puppet

Puppet был основан как open source в 2005 году и коммерциализирован как Puppet Enterprise компанией Puppet Labs в 2011 году.

Основная архитектура Puppet имеет следующие характеристики:

– Назначенный сервер для размещения основных компонентов приложения:
    1. Puppet Server (исторически называемый "Puppet Master")
    2. Facter, служба сбора фактов
    3. PuppetDB, в которой могут храниться факты, каталоги узлов и история последних событий конфигурации.
– Безопасный клиент, также известный как Puppet Agent, установленный и настроенный на целевых машинах. Клиенты и сервер проходят взаимную аутентификацию с помощью самоподписанных

сертификатов, а для передачи данных используется протокол SSL. Агенты собирают факты (под контролем службы Facter) и вносят изменения в конфигурацию по указанию Puppet Server.

– Для облачных API и оборудования, на котором не может работать агент, в Puppet есть модули, позволяющие устанавливать такие соединения.
– В масштабных реализациях, где под управлением находится множество устройств, не поддерживающих агентов, Puppet позволяет использовать прокси-агент, чтобы разгрузить работу по прямому подключению к CLI устройств и обмену информацией.

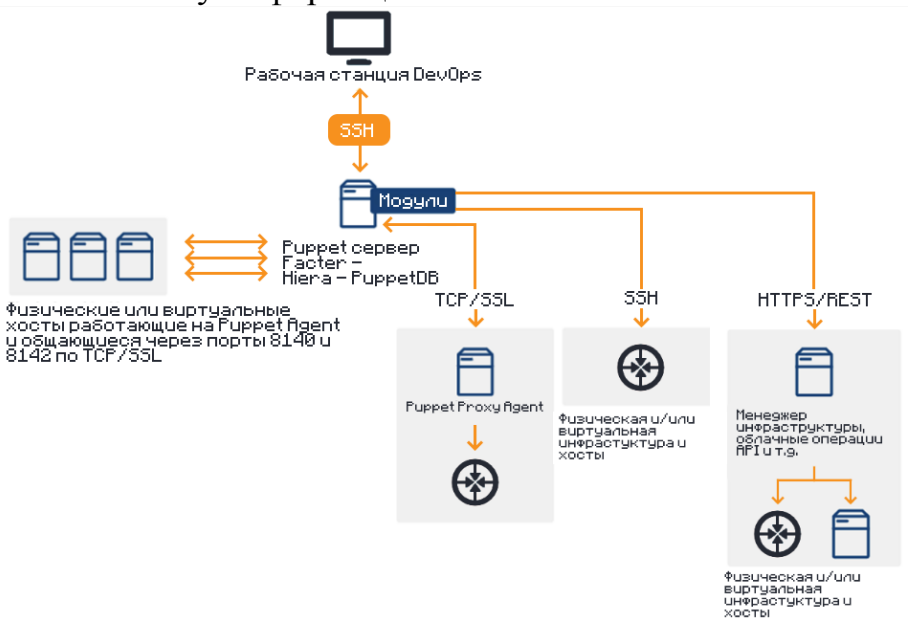


Рисунок 38 — Архитектура Puppet

Chef

Chef предоставляет полную систему для работы с инфраструктурой как с кодом. Продукты Chef частично лицензируются, но бесплатны для личного использования.

Продукты и решения Chef позволяют создавать, тестировать, организовывать, хранить в репозитории и выполнять на удаленных объектах инфраструктуру как код либо с отдельной рабочей станции Chef, либо опосредованно с центрального сервера Chef Infra Server. Вы должны знать основные компоненты Chef:

– Chef Workstation — автономная рабочая станция оператора.
– Chef Infra Client — агент хоста.
– Cookbooks (и прокси-клиенты) — позволяют управлять оборудованием и ресурсами, которые не могут запустить Chef Infra Client локально.
– Сервер Chef Infra — отвечает на запросы от агентов Chef Infra на проверенных хостах и предоставляет обновления конфигурации, на основе которых агенты затем сходятся в конфигурации хоста.

Большинство задач конфигурирования также можно выполнять непосредственно между Chef Workstation и управляемыми узлами, и устройствами.

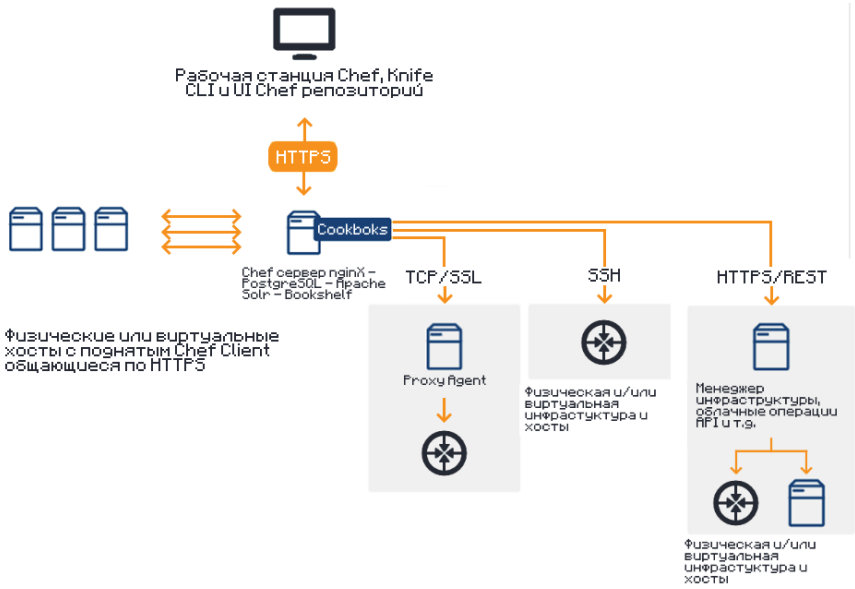


Рисунок 39 — Архитектура Chef

Infrastructure as Code

Неизменяемость буквально означает "состояние неизменности", но в языке DevOps это означает поддержание систем полностью в виде кода, без выполнения каких-либо ручных операций над ними [5, 8, 16]. Принятие обязательств по неизменяемости позволяет вам обращаться с кодовой базой автоматизации так же, как с любым кодом приложения:

- Доверие: Кодовая база описывает то, что на самом деле работает на железе или облачных серверах.
- Управление кодовой базой с помощью Agile-процедур и использования контроля версий, чтобы все было ясно и просто.

Автоматизация тестирования

DevOps обычно нуждается в более тонких способах определения и внедрения инфраструктур, подтверждения того, что развернутые инфраструктуры работают так, как нужно, проактивного обеспечения их бесперебойной работы, упреждающего принятия мер при неизбежных сбоях, а также поиска и устранения проблем при возникновении ошибок [2, 3, 18, 19].

Благодаря использованию инструментов модульного тестирования в тандеме с автоматизацией более высокого порядка и в сочетании с CI/CD, можно создать среду, в которой код будет автоматически тестироваться при внесении изменений. Такие фреймворки делают тесты частью вашей кодовой базы, следуя за кодом через коммиты, merge|pull запросы и code-review до QA/test и Production. Это особенно полезно в средах разработки, управляемой TDD, где написание тестов — это непрерывный процесс, который фактически ведет разработку, автоматически поощряя очень высокий уровень покрытия тестами, повышая в разы безопасность итогового продукта.

6 ОПИСАНИЕ ЛАБОРАТОРНЫХ И ПРОЕКТНЫХ РАБОТ

6.1.1 Установка виртуальной лабораторной среды 1.1.2

Цель лабораторной работы:

Часть 1: Подготовка компьютера к Виртуализации
Часть 2: Знакомство с DEVASC VM GUI
Часть 3: Создание Учётной записи для лабораторной среды
Часть 4: Установка Webex Teams на ваше устройство

Общие сведения / Сценарий

В этой лабораторной работе вы установите виртуальную машину DEVASC (DEVASC VM) в Oracle VirtualBox. После завершения установки вы изучите интерфейс GUI. Затем вы создадите необходимые учетные записи для служб, которые вы будете использовать в ходе дальнейших лабораторных работ. Наконец, вы установите Webex Teams в лабораторной среде для общения с другими студентами вашего класса и для использования в последующих лабораторных работах.

Необходимые ресурсы:

- Хост-компьютер как минимум с 4 ГБ ОЗУ и 15 ГБ свободного дискового пространства.
- Высокоскоростной доступ в Интернет для загрузки Oracle VirtualBox и DEVASC VM

6.1.2 Знакомство с Linux 1.2.2

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC
Часть 2: Обзор навигации по синтаксису команд
Часть 3: Обзор управления файлами
Часть 4: Обзор регулярных выражений
Часть 5: Обзор системного администрирования

Общие сведения / Сценарий

В этой лабораторной работе вы познакомитесь с основными навыками работы с Linux, включая навигацию по командам, управление файлами, регулярные выражения и системное администрирование. Тем не менее, выполнение лабораторной работы должно послужить хорошим показателем ваших навыков работы с Linux и помочь вам определить, где вам может потребоваться дополнительное изучение.

Необходимые ресурсы

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

6.1.3 Знакомство с Языком Программирования Python 1.3.3

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC
Часть 2: Запуск Python и VS Code
Часть 3: Обзор типов данных и переменных
Часть 4: Обзор списков и словарей
Часть 5: Обзор функции ввода
Часть 6: Обзор функций If, For и While
Часть 7: Обзор методов доступа к файлам

Общие сведения / Сценарий

В этой лаборатории вы познакомитесь с основным набором программирования на языке Python, включая типы данных, переменные, списки, словари, ввод данных пользователем, операторы if, циклы for и while, а также доступ к файлам. Тем не менее, выполнение лабораторной работы должно послужить хорошим показателем ваших навыков программирования на Python и помочь вам определить, где вам может понадобиться дополнительное изучение.

Необходимые ресурсы

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

6.1.4 Исследование инструментов разработки на языке Python 3.1.12

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC
Часть 2: Обзор установки Python
Часть 3: Виртуальные среды PIP и Python
Часть 4: Совместное использование виртуальной среды

Общие сведения / сценарий

Лабораторная практика с Python дает вашим пальцам "мышечную память" для более эффективной работы в Python. Однако на начальном этапе вам важно знать, как поддерживать среду разработки Python. В этой лабораторной работе вы рассмотрите установку Python, PIP и виртуальные среды Python.

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

6.1.5 Контроль версий программного обеспечения с помощью Git 3.3.11

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC
Часть 2: Инициализация Git
Часть 3: Постановка и фиксация файла в репозитории Git
Часть 4: Управление файлом и отслеживание изменений

Часть 5: Ветви и слияние
Часть 6: Обработка конфликтов при слиянии
Часть 7: Интеграция Git с GitHub

Общие сведения / Сценарий

В этой лабораторной работе вы изучите основы распределенной системы контроля версий Git, включая большинство функций, которые необходимо знать для совместной работы над программным проектом. Вы также интегрируете свой локальный репозиторий Git с облачным репозиторием GitHub.

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

6.1.6 Знакомство с классами в Python 3.4.6

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC
Часть 2: Обзор функций, методов и классов
Часть 3: Определение функции
Часть 4: Определение класса с методами
Часть 5: Обзор сценария circleClass.py

Общие сведения / сценарий

В этой лабораторной работе вы рассмотрите методы, функции и классы Python. Затем вы создадите класс и создадите его несколько раз с различными значениями. Наконец, вы рассмотрите пример реализации класса Circle.

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

6.1.7 Unit-тестирование на Python 3.5.7

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC
Часть 2: Изучение опций в unittest Framework
Часть 3: Тестирование функции Python с помощью unittest

Общие сведения / сценарий

Юнит-тесты исследуют независимые единицы кода, такие как функции, классы, модули и библиотеки. Есть много причин для написания сценария с использованием библиотеки unittest в Python. Одна из очевидных причин заключается в том, что, если вы обнаружили проблему в изолированном коде путем целенаправленного тестирования, вы знаете, что проблема находится в тестируемой функции или другом модуле. Проблема не кроется в более крупном приложении, которое может вызывать эту функцию. Вы также будете точно знать, что вызвало ошибку, потому что вы написали модульный тест, который выявил проблему. Ошибки, найденные таким образом, обычно

быстро и легко устраняются, а исправления, сделанные на таком детальном уровне, с меньшей вероятностью вызовут непредвиденные побочные эффекты в дальнейшем в другом коде, который полагается на протестированный код.
Если код небольшой, можно выполнять модульные тесты вручную, но обычно модульные тесты должны быть автоматизированы. При написании модульного теста подумайте о следующем:

- Юнит-тест должен быть простым и легко реализуемым;
- Юнит-тест должен быть хорошо документирован, чтобы можно было легко понять, как запустить тест, даже спустя несколько лет;
- Рассмотрите методы тестирования и входные данные со всех сторон;
- Результаты тестирования должны быть последовательными. Это важно для автоматизации тестирования;
- Тестовый код должен работать независимо от тестируемого кода. Если вы пишете тесты, которые должны изменять состояние программы, фиксируйте состояние до его изменения и возвращайте его обратно после выполнения теста;
- Когда тест не работает, результаты должны быть легко читаемы и четко указывать на то, что ожидалось и где возникли проблемы.

В этой лабораторной работе вы изучите фреймворк unittest и используете unittest для тестирования функции.

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

6.1.8 Парсинг различных типов данных с помощью Python 3.6.6

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC
Часть 2: Разбор XML в Python
Часть 3: Разбор JSON в Python
Часть 4: Разбор YAML в Python

Общие сведения / Сценарий

Разбор означает анализ сообщения, разбиение его на составные части и понимание назначения каждой части в контексте. При передаче сообщений между компьютерами они передаются в виде потока символов. Эти символы представляют собой строку. Это сообщение должно быть разобрано на семантически эквивалентную структуру данных, содержащую данные распознанных типов (например, целые числа, плавающие числа, строки и булевы), прежде чем эти данные можно будет интерпретировать и действовать с ними.

В этой лабораторной работе вы будете использовать Python для разбора каждого формата данных по очереди: XML, JSON и YAML. Мы рассмотрим примеры кода и поговорим о том, как работает каждый парсер.

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

6.1.9 Изучение Rest API с помощью API Simulator и Postman 4.5.5

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC
Часть 2: Исследование документации API с помощью API Simulator
Часть 3: Использование Postman для выполнения вызовов API в API Simulator
Часть 4: Использование Python для добавления 100 книг в API Simulator

Общие сведения / Сценарий

ВМ DEVASC включает симулятор API школьной библиотеки с документацией по API и соответствующей базой данных. Вы можете использовать симулятор в автономном режиме для изучения API и тестирования их функциональности.

В этой лабораторной работе вы узнаете, как использовать симулятор API школьной библиотеки для выполнения вызовов API для составления списка, добавления и удаления книг. Позже вы будете использовать Postman для выполнения таких же вызовов API.

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

6.1.10 Интеграция REST API и Python приложения 4.9.2

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC
Часть 2: Демонстрация приложения MapQuest Directions
Часть 3: Получение API-ключа MapQuest
Часть 4: Создание базового приложения MapQuest Direction
Часть 5: Обновление приложения MapQuest Directions с помощью дополнительных функций
Часть 6: Тестирование полной функциональности приложения

Общие сведения / сценарий

В этой лабораторной работе вы создадите приложение в Visual Studio Code (VS Code), которое будет получать JSON данные из MapQuest Directions API, анализировать их и форматировать для вывода пользователю. Вы будете использовать запрос GET Route из MapQuest Directions API. Ознакомьтесь с документацией API GET Route Directions здесь: https://developer.mapquest.com/documentation/directions-api/route/get/

Необходимые ресурсы:

- 1 ПК

- Virtual Box или VMWare
- DEVASC виртуальная машина

6.1.11 Инструменты для устранения неполадок в сети 5.6.7

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC
Часть 2: Знакомство с инструментом ifconfig
Часть 3: Знакомство с инструментом ping
Часть 4: Знакомство с инструментом traceroute
Часть 5: Знакомство с инструментом nslookup

Общие сведения / Сценарий

При устранении проблем с сетевым подключением разработчику важно понимать, как использовать основные инструменты для устранения неполадок в сети. Эти инструменты используются для определения того, в чем может заключаться проблема подключения.

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

6.1.12 Сборка веб-приложения в Docker контейнере 6.2.7

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC
Часть 2: Создаём простой Bash скрипт
Часть 3: Создаём демо веб-приложение
Часть 4: Настройка веб-приложения для использования файлов веб-сайта
Часть 5: Создаём Bash скрипт для создания и запуска Docker контейнера
Часть 6: Сборка, запуск и проверка Docker контейнера

Общие сведения / Сценарий

В этой лабораторной работе вы рассмотрите основные методы написания сценариев на языке bash, поскольку работа с bash является необходимым условием для выполнения остальных лабораторных работ. Затем вы создадите и измените сценарий Python для простого веб-приложения. Затем вы создадите сценарий bash для автоматизации процесса создания Dockerfile, сборки Docker-контейнера и запуска Docker-контейнера. Наконец, вы будете использовать команды docker для изучения тонкостей работы контейнера Docker.

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

6.1.13 Построение CI/CD с помощью Jenkins 6.3.6

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC
Часть 2: Закоммитьте Sample App на Git

Часть 3: Изменение Sample App и внесение изменений в Git
Часть 4: Загрузка и запуск Jenkins Docker Image
Часть 5: Конфигурация Jenkins
Часть 6: Использование Jenkins для запуска сборки вашего приложения
Часть 7: Использование Jenkins для тестирования сборки
Часть 8: Создание Пайплайна в Jenkins

Общие сведения / сценарий

В этой лабораторной вы закоммитете код Sample App в репозитории GitHub, измените код локально, а затем зафиксируете свои изменения. Затем вы установите контейнер Docker, включающий последнюю версию Jenkins. Вы настроите Jenkins, а затем будете использовать Jenkins для загрузки и запуска программы Sample App. Затем вы создадите джобу тестирования в Jenkins, которая будет проверять успешный запуск программы Sample App каждый раз, когда вы её собираете. Наконец, вы интегрируете Sample App и джобу тестирования в Continuous Integration/Continuous Development пайплайн, который будет проверять готовность Sample App к развертыванию при каждом изменении кода.

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

6.1.14 Исследование эволюции парольных систем 6.5.10

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC
Часть 2: Python: пароли в обычном тексте
Часть 3: Python: парольный хэша

Общие сведения / сценарий

В этой лабораторной работе вы создадите приложение, которое хранит имя пользователя и пароль в открытом виде в базе данных с помощью кода python. Затем вы протестируете сервер, чтобы убедиться, что учетные данные не только сохранены правильно, но и что пользователь может использовать их для входа в систему. Затем вы выполните те же действия, но с хэшированным паролем, чтобы учетные данные нельзя было прочитать. Важно надежно хранить учетные данные и другие сведения, чтобы предотвратить взлом различных серверов и систем.

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

6.1.15 Установка виртуальной машины CSR1000v 7.0.3

Цель лабораторной работы:

Часть 1: Установка виртуальной машины CSR1000v на VirtualBox
Часть 2: Проверка подключения с CSR1000v VM

Общие сведения / сценарий

В этой лабораторной работе вы установите маршрутизатор облачных сервисов CSR1000v. Это программный виртуальный маршрутизатор Cisco, развернутый на виртуальной машине (ВМ) на аппаратном обеспечении сервера x86.

Необходимые ресурсы:

- ПК с не менее чем 4 ГБ оперативной памяти и 2 ГБ свободного дискового пространства
- Virtual Box или VMWare
- DEVASC виртуальная машина

6.1.16 Ansible для резервного копирования и настройки устройства 7.4.7

Цель лабораторной работы:

Часть 1: Запуск виртуальных машин DEVASC и CSR1000v

Часть 2: Настройка Ansible

Часть 3: Ansible для резервного копирования конфигурации

Часть 4: Ansible для настройки устройства

Общие сведения / Сценарий

В этой лабораторной работе вы изучите основы использования Ansible для автоматизации некоторых базовых задач управления устройствами. Сначала вы настроите Ansible в вашей виртуальной машине DEVASC. Затем, вы будете использовать Ansible для подключения к CSR1000v и резервного копирования его конфигурации. Наконец, вы настроите CSR1000v с адресацией IPv6.

Необходимые ресурсы:

- ПК с не менее чем 4 ГБ оперативной памяти
- Virtual Box или VMWare
- DEVASC виртуальная машина
- CSR1000v виртуальная машина

6.1.17 Ansible для автоматизации установки веб-сервера 7.4.8

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC

Часть 2: Настройка Ansible

Часть 3: Проверка связи с локальным веб-сервером

Часть 4: Ansible Playbooks для автоматизации установки веб-сервера

Часть 5: Добавление опций в Ansible Playbook для Apache Web Servers

Общие сведения / сценарий

В этой лабораторной работе вы сначала настроите Ansible так, чтобы он мог взаимодействовать с веб-сервером приложения. Затем вы создадите сборник plays, который автоматизирует процесс установки Apache на веб-сервер. Вы также создадите настраиваемый playbook, который установит Apache с помощью специальных инструкций.

Необходимые ресурсы:

- ПК с не менее чем 4 ГБ оперативной памяти
- Virtual Box или VMWare
- DEVASC виртуальная машина

6.1.18 Автоматизированное тестирование с помощью pyATS и Genie 7.6.3

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC
Часть 2: Создание виртуальной среды Python
Часть 3: pyATS для тестирования
Часть 4: Genie для анализа командного вывода IOS
Часть 5: Genie для сравнения конфигураций
Часть 6: Завершение лабораторной и дальнейшие исследования

Общие сведения / Сценарий

В этой лабораторной работе вы изучите основы pyATS и Genie. Инструмент pyATS представляет собой экосистему сквозного тестирования, специализирующуюся на тестировании на основе данных и многократном использовании, и разработанную таким образом, чтобы подходить для быстрых итераций Agile-разработки. Расширяемый по своей конструкции, pyATS позволяет разработчикам начинать с небольших, простых и линейных тестовых случаев и масштабироваться до больших, сложных и асинхронных тестовых наборов.

Genie расширяет и развивает pyATS для использования в сетевой среде. Примеры возможностей, которые предоставляет Genie, включают:

- возможность подключения устройств, парсеры и API-интерфейсы
- платформо-агностические объектные модели Python для таких функций, как OSPF и BGP
- пул многократно используемых тестовых примеров
- механизм запуска тестов на основе YAML

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина
- CSR1kv виртуальная машина

6.1.19 Исследование YANG моделей 8.3.5

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC
Часть 2: Изучение YANG модели на GitHub
Часть 3: Изучение YANG модели с помощью pyang

Общие сведения / сценарий

Модели YANG определяют точную структуру, типы данных, синтаксис и правила проверки содержимого сообщений, которыми обмениваются управляемое устройство и другая система, взаимодействующая с устройством.

Работа с файлами, использующими язык YANG, может быть немного утомительной из-за уровня деталей в этих файлах.

В этой лабораторной работе вы узнаете, как использовать инструмент pyang с открытым исходным кодом для преобразования моделей данных YANG из файлов, использующих язык YANG, в более простой для чтения формат. Используя преобразование вида "дерево", вы определите, каковы ключевые элементы YANG-модели ietf-interfaces.

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

6.1.20 NETCONF для доступа к устройству IOS XE 8.3.6

Цель лабораторной работы:

Часть 1: Создание сети и проверка подключения
Часть 2: Использование сеанса NETCONF для сбора информации
Часть 3: Использование ncclient для подключения к NETCONF
Часть 4: Использование ncclient для получения конфигурации
Часть 5: Использование ncclient для настройки устройства
Часть 6: Задача: Модификация программы, использованной в этой лабораторной работе

Общие сведения / сценарий

Протокол конфигурации сети (NETCONF), определенный в RFC 4741 и 6241, использует модели данных YANG для взаимодействия с различными устройствами в сети. YANG (yet another next generation) — это язык моделирования данных. Этот язык определяет данные, которые передаются по протоколам управления сетью, таким как NETCONF. При использовании NETCONF для доступа к устройству IOS XE данные возвращаются в формате XML.

В этой лабораторной работе вы будете использовать клиент NETCONF, ncclient, который представляет собой модуль Python для создания сценариев на стороне клиента. Вы будете использовать ncclient для проверки конфигурации NETCONF, получения конфигурации устройства и изменения конфигурации устройства.

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина
- CSR1kv виртуальная машина

6.1.21 RESTCONF для доступа к устройству IOS XE 8.3.7

Цель лабораторной работы:

Часть 1: Создание сети и проверка подключения
Часть 2: Настройка устройства IOS XE для доступа к RESTCONF
Часть 3: Запуск и настройка Postman

Часть 4: Postman для отправки GET-запросов
Часть 5: Postman для отправки PUT запроса
Часть 6: Python скрипт для отправки GET-запросов
Часть 7: Python скрипт для отправки PUT запроса

Общие сведения / сценарий

Протокол RESTCONF предоставляет упрощенное подмножество функций NETCONF через RESTful API. RESTCONF позволяет выполнять вызовы RESTful API к устройству IOS XE. Данные, возвращаемые API, могут быть отформатированы в XML или JSON. В первой половине этой лабораторной работы вы будете использовать программу Postman для создания и отправки запросов API службе RESTCONF, запущенной на CSR1kv. Во второй половине лабораторной работы вы создадите сценарии Python для выполнения тех же задач, что и программа Postman.

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина
- CSR1kv виртуальная машина

6.1.22 Создание Python программы для управления Webex Teams 8.6.7

Цель лабораторной работы:

Часть 1: Создание сети и проверка подключения
Часть 2: Получение Access Token для Webex Teams
Часть 3: Проверка Access Token
Часть 4: Управление людьми в Webex Teams
Часть 5: Управление комнатами в Webex Teams
Часть 6: Управление членством в Webex Teams
Часть 7: Управление сообщениями в Webex Teams

Общие сведения / Сценарий

В этой лабораторной работе вы будете использовать API Webex Teams для аутентификации, управления людьми, управления комнатами, управления членством в комнатах и отправки сообщения.

Необходимые ресурсы:

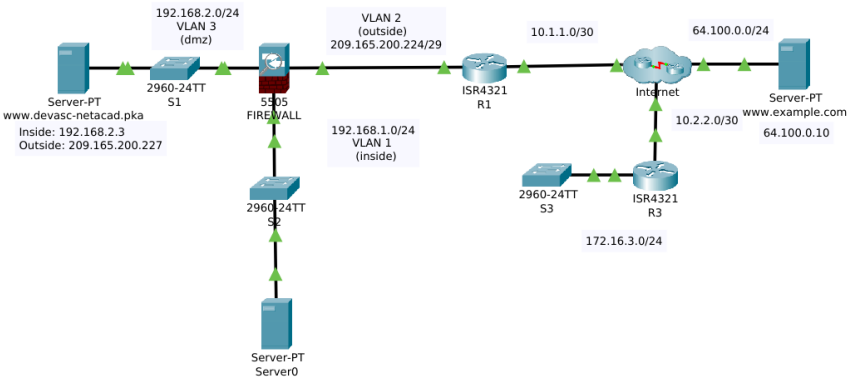
- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

6.1.23 Исследование простой сети 5.4.6

Цель лабораторной работы:

Часть 1: Добавление ПК в топологию
Часть 2: Тестирование подключения по сети
Часть 3: Создание веб-страницы и ее просмотр
Часть 4: Изучение списков доступа FIREWALL

Схема сети



Необходимые ресурсы:

- 1 ПК
- Packet Tracer 7.4.0

6.1.24 Исследование сетевых протоколов 5.5.7

Цель лабораторной работы:

Часть 1: Настройка DNS

Часть 2: Настройка DHCP

Часть 3: Настройка NTP

Часть 4: Использование SSH для безопасного доступа к коммутатору

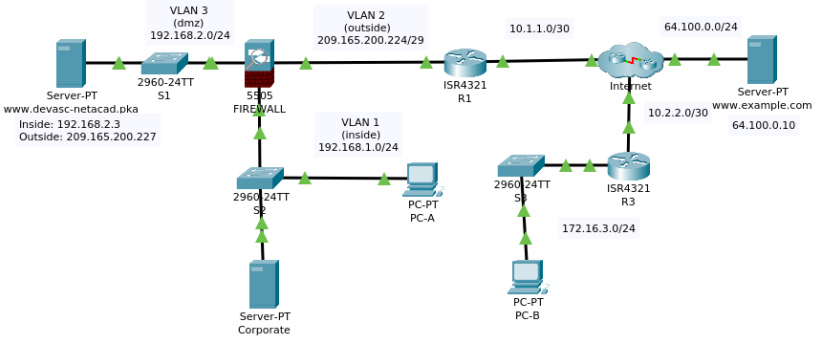
Часть 5: Изучение идентификаторов объектов SNMP MIB

Часть 6: Настройка HTTPS

Часть 7: Настройка EMAIL

Часть 8: Настройка FTP

Схема сети



Необходимые ресурсы:

- 1 ПК
- Packet Tracer 7.4.0

6.1.25 Сравнение CLI и SDN контроллера 8.8.2

Цель лабораторной работы:

Часть 1: Изучение топологии сети

Часть 2: Использование CLI для сбора информации

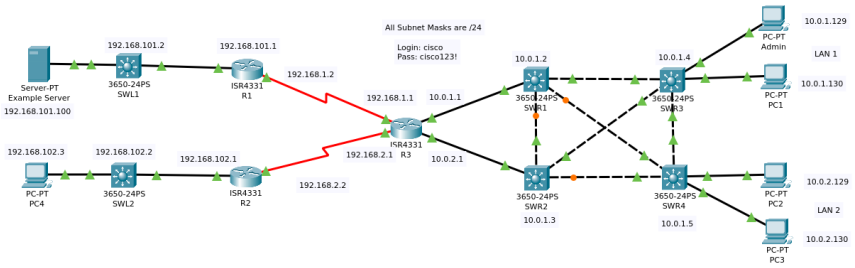
Часть 3: Настройка контроллера SDN

Часть 4: SDN контроллер для обнаружения топологии

Часть 5: SDN контроллер для сбора информации

Часть 6: SDN контроллер для настройки параметров сети

Схема сети



Необходимые ресурсы:

- 1 ПК
- Packet Tracer 7.4.0

6.1.26 Сравнение CLI и SDN контроллера 8.8.3

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC

Часть 2: Проверка внешнего подключения к Packet Tracer

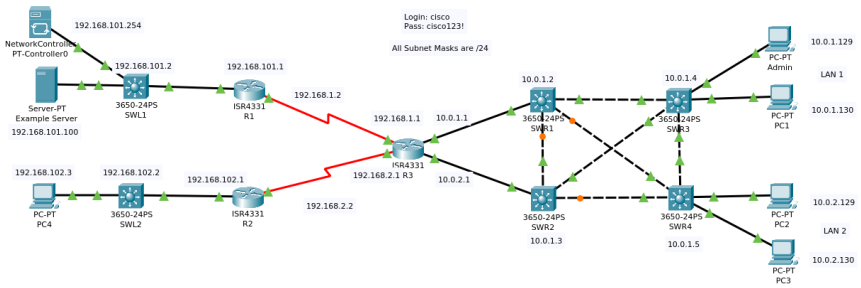
Часть 3: Запрос маркера аутентификации с помощью Postman

Часть 4: Отправка REST-запросов с помощью Postman

Часть 5: Отправка REST-запросов с помощью VS Code

Часть 6: Отправка REST-запросов внутри Packet Tracer

Схема сети



Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

6.2.1 Формирование команды

Задачи проектной работы:

- Выбор ведущего и записывающего.
- Проведите упражнение "растопить лед".
- Обсудите эффективную работу в команде.
- Выберите роль для каждого члена команды.

6.2.2 Формирование Agile команды

Задачи проектной работы:

- Выбор проекта.
- Создание Agile команды.
- Распределение проектных ролей.
- Выберите каналы связи, которые будут способствовать работе команды.

6.2.3 Социальное программирование

В этом задании вы выполните следующие задачи:

- Выберите свой проект.
- Определите функции, которые вы обязуетесь реализовать в своем приложении.

6.2.4 Автоматизированное тестирование и развертывание программного обеспечения

Задачи проектной работы:

- Разработать процесс автоматизированного тестирования.
- Реализовать процесс автоматизированного тестирования.

6.2.5 Программирование и автоматизация сети

Задачи проектной работы:

- Разработка нового приложение для автоматизации сети.
- Использование моделирования данных с помощью YANG.
- Реализация автоматизации с помощью NETCONF.
- Автоматизация процесса с помощью Python.

ЗАКЛЮЧЕНИЕ

Рост технологий, требует беспрерывной и постоянной работы компьютерных сетей при поставке обновлений, в связи с чем происходит сдвиг в сторону усложнения как сетей, так и самого обновляемого ПО. Несоответствие данным требованиям, влечёт убытки для бизнеса как репутационные так и финансовые из-за невозможности сотрудников охватывать огромный спектр процессов и инструментов применяемых в организациях. Исследовав проблему возможности применения технологий автоматизации с целью повышения ИБ, и рассмотрев проблемы и угрозы предшествующего года в IT, существующие подходы к разработке ПО, сами инструменты автоматизации; мною был предложен подход использования этих инструментов автоматизации для помощи разработчикам, при создании новых и эксплуатации как существующих приложений, так и уже существующих продуктов.

Основное внимание было уделено развертыванию приложений, их безопасности, инфраструктуре и автоматизация, с помощью которых создаются, не только полезные, но и хорошо защищённые и протестированные приложения и продукты, которые уже стали частью современного человека, а также идеи того, что автоматизация будет действительно эффективной, если произойдет ряд изменений в организационной культуре(фокус на автоматизации, сбои случаются это нормально и переосмысление в связи с этим понятия доступность с точки зрения, что может выдержать бизнес).

Теоретическая составляющая помимо дополнительной теории, подкреплена комплексом лабораторных работ, состоящий из исследования сетевых технологий, полного цикла разработки по, обеспечения его безопасности и интеграцией его инструментами автоматизации.

Выполнение курса лабораторных и проектных работ позволит студенту получить опыт разработки в условиях реального продакшена, что даст более ясное представление о принципах разработки надежного программного обеспечения и его поставки в компьютерных сетях.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Design Patterns: Elements of Reusable Object-Oriented Software / E. Gamma, R. Helm, R. Johnson, J. Vlissides. – United States : Addison-Wesley, 1994. – 395 с.
2. Site Reliability Engineering: How Google Runs Production Systems / N. Murphy, C. Jones, B. Beyer, J. Petoff. – United States : O'Reilly Media, 2016. – 550 с.
3. Davis J. Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling at Scale / J. Davis, R. Daniels. – United States : O'Reilly Media, 2016. – 410 с.
4. Cisco Certified DevNet Associate DEVASC 200-901 Official Cert Guide / C. Jackson, J. Gooley, A. Iliesiu, A. Malegaonkar. – United States : Cisco Press, 2020. – 600 с.
5. Gurbatov G. COMPARATIVE ANALYSIS OF INFRASTRUCTURE AS CODE TOOLS / G. Gurbatov, A. Panichev, I. Ushakov // ПОДГОТОВКА ПРОФЕССИОНАЛЬНЫХ КАДРОВ В МАГИСТРАТУРЕ ДЛЯ ЦИФРОВОЙ ЭКОНОМИКИ (ПКМ-2021). – Saint Petersburg : , 2022. – С. 208-213.
6. Gurbatov G. KUBERNETES BASIC SECURITY / G. Gurbatov, A. Panichev, I. Ushakov // АКТУАЛЬНЫЕ ПРОБЛЕМЫ ИНФОТЕЛЕКОММУНИКАЦИЙ В НАУКЕ И ОБРАЗОВАНИИ. – Saint Petersburg : , 2021. – С. 282-286.
7. Avdeeva M. COMPARATIVE ANALYSIS OF THE OPENSTACK CLOUD CONCEPT AND TRADITIONAL VIRTUALIZATION ARCHITECTURE / M. Avdeeva, I. Ushakov, A. Filippov // АКТУАЛЬНЫЕ ПРОБЛЕМЫ ИНФОТЕЛЕКОММУНИКАЦИЙ В НАУКЕ И ОБРАЗОВАНИИ (АПИНО 2020). – Saint Petersburg : , 2020. – С. 33-38.
8. Gurbatov G. COMPARATIVE ANALYSIS OF IAAS (OPENSTACK) AND PAAS (OPENSHIFT) / G. Gurbatov, A. Panichev, I. Ushakov // АКТУАЛЬНЫЕ ПРОБЛЕМЫ ИНФОТЕЛЕКОММУНИКАЦИЙ В НАУКЕ И ОБРАЗОВАНИИ (АПИНО 2020). – Saint Petersburg : , 2020. – С. 272-276.
9. Docker Documentation : сайт. – URL: https://docs.docker.com/ (дата обращения: 15.05.2022)
10. Jenkins User Documentation : сайт. – URL: https://www.jenkins.io/doc/ (дата обращения: 15.05.2022)
11. OWASP Top Ten : сайт. – URL: https://owasp.org/www-project-top-ten/ (дата обращения: 09.05.2022)
12. Глобальный сбой в работе Facebook, Instagram и WhatsApp продолжался более 5 часов, DNS Facebook заработал // habr : сайт. – URL: https://habr.com/ru/news/t/581538/ (дата обращения: 03.05.2022)
13. 2021 Facebook outage // Wikipedia : сайт. – URL: https://en.wikipedia.org/wiki/2021_Facebook_outage (дата обращения: 13.03.2022)
14. Сине-зеленый деплой // habr : сайт. – URL: https://habr.com/ru/post/309832/ (дата обращения: 13.03.2022)

15. Тестируем на проде: Canary Deployment // habr : сайт. – URL: https://habr.com/ru/company/oleg-bunin/blog/493026/ (дата обращения: 13.03.2022)
16. Облачные бизнес-модели: Простыми словами об IaaS, PaaS и SaaS // habr : сайт. – URL: https://habr.com/ru/company/1cloud/blog/320538/ (дата обращения: 13.03.2022)
17. Алексей Картынник. (2019) Защита информации / Белые хакеры и политики безопасности / Всё о кибербезопасности / 19 Декабря // YouTube : сайт. – URL: https://www.youtube.com/watch?v=4oLzvhDRZT4&ab_channel=АйТиБорода
18. Алексей Картынник. (2019) Docker, непрерывная интеграция и плоская земля / Интервью с DevOps Engineer / 8 Апреля // YouTube : сайт. – URL: https://www.youtube.com/watch?v=nrr4IwOVhZk&ab_channel=АйТиБорода
19. Алексей Картынник. (2020) DevOps ЗДОРОВОГО ЧЕЛОВЕКА / Все пути ведут в КУБЕРНЕТЕС / Интервью с Дмитрием Столяровым / 5 Ноября // YouTube : сайт. – URL: https://www.youtube.com/watch?v=htm12lYKDUs&ab_channel=АйТиБорода
20. Алексей Картынник. (2020) АРХИТЕКТУРА ПРОГРАММ / GRASP, KISS, Дядя Боб и Война / Интервью с Сергеем Немчинским 27 Апреля – // YouTube : сайт. – URL: https://www.youtube.com/watch?v=1yc1G31NaT0&t=299s&ab_channel=АйТиБорода

ПРИЛОЖЕНИЕ А

Лабораторные работы.

В данном приложении рассматриваются примеры лабораторных работ. Для каждой работы представлены описание, задачи, процедура возможного выполнения и порядок предоставления отчета. Весь процесс выполнения сопровождается четкими указаниями, это позволяет успешнее воспринимать материалы курса.

Комплекс лабораторных работ предназначен для освоения реализации основных механизмов защиты информации, применяемых в современных сетях связи. В работах подробно описаны этапы настройки функций защиты на активном сетевом оборудовании. Работы выполняются на реальном оборудовании, что поможет студентам в дальнейшем применить полученные навыки в практической реализации средств защиты.

Выполнение описанных лабораторных работ также позволит студенту получить более ясное представление о принципах и алгоритмах действия механизмов защиты сетевого оборудования.

Лабораторная работа №1.1.2

Установка виртуальной лабораторной среды

Цель лабораторной работы:

Часть 1: Подготовка компьютера к Виртуализации
Часть 2: Знакомство с DEVASC VM GUI
Часть 3: Создание Учётной записи для лабораторной среды
Часть 4: Установка Webex Teams на ваше устройство

Необходимые ресурсы:

- Хост-компьютер как минимум с 4 ГБ ОЗУ и 15 ГБ свободного дискового пространства.
- Высокоскоростной доступ в Интернет для загрузки Oracle VirtualBox и DEVASC VM

Порядок выполнения работы:

Часть 1: Подготовка компьютера к Виртуализации

В этой части вы установите на программное обеспечение для виртуализации и DEVASC VM. Ваш инструктор предоставит вам файл DEVASC VM.

Примечание: Описанные ниже действия исполнялись на Windows 10 используя VirtualBox v6.1.4. Ваши шаги могут отличаться.

Независимо от операционной системы или версии VirtualBox, убедитесь, что вы нашли и выбрали опции, указанные в следующих шагах.
Примечание: Если у вас установлен VirtualBox, вы можете перейти к Шагу 2.

Шаг 1: Загрузка и установка VirtualBox.

VMware Player и Oracle VirtualBox — это две программы виртуализации, которые вы можете загрузить и установить для поддержки образов виртуальных машин. В этой лабораторной работе вы будете использовать программу VirtualBox.

- Перейдите на сайт https://www.virtualbox.org/. На открывшейся странице, перейдите в раздел Downloads.
- Выберите и загрузите соответствующий установочный файл VirtualBox в зависимости от вашей операционной системы.
- Запустите установщик VirtualBox, выбрав в параметры установки по умолчанию.
- Запустите VirtualBox, и переходите к следующему шагу.

Шаг 2: Импорт DEVASC VM.

a. Перейдите на страницу DevNet Associate Virtual Machines (VMs) netacad.com.

DevNet Associate Virtual Machines (VMs)

The DevNet Associate course uses several virtual machines (VMs) to create the lab experiences. Use the links below to download the course specific VMs.
Two labs are included in the course that detail the installation and setup steps for the VMs.

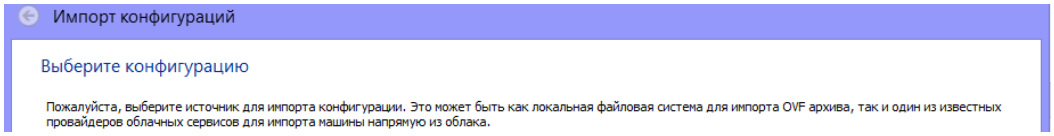
For China Only – If you are accessing this page from within greater China, please click here for a faster download experience.
For all others, use the links below:

Description	File Name
DevNet Associate VM	DEVASC_VM.OVA
CSR1000v	DEVASC_CSR1000v.zip

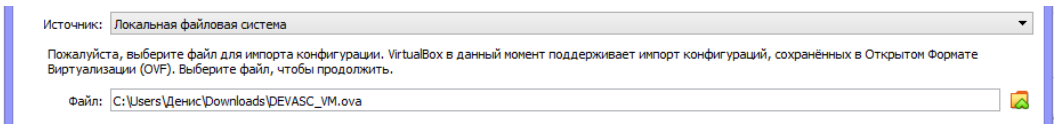
b. Скачайте DEVASC_VM.OVA и запомните расположение загруженной виртуальной машины.

Примечание*: На странице также расположен файл DEVASC_CSR1000v.zip. Сейчас нет необходимости его загружать. Для его установки требуется дополнительный файл, предоставляемый инструктором*

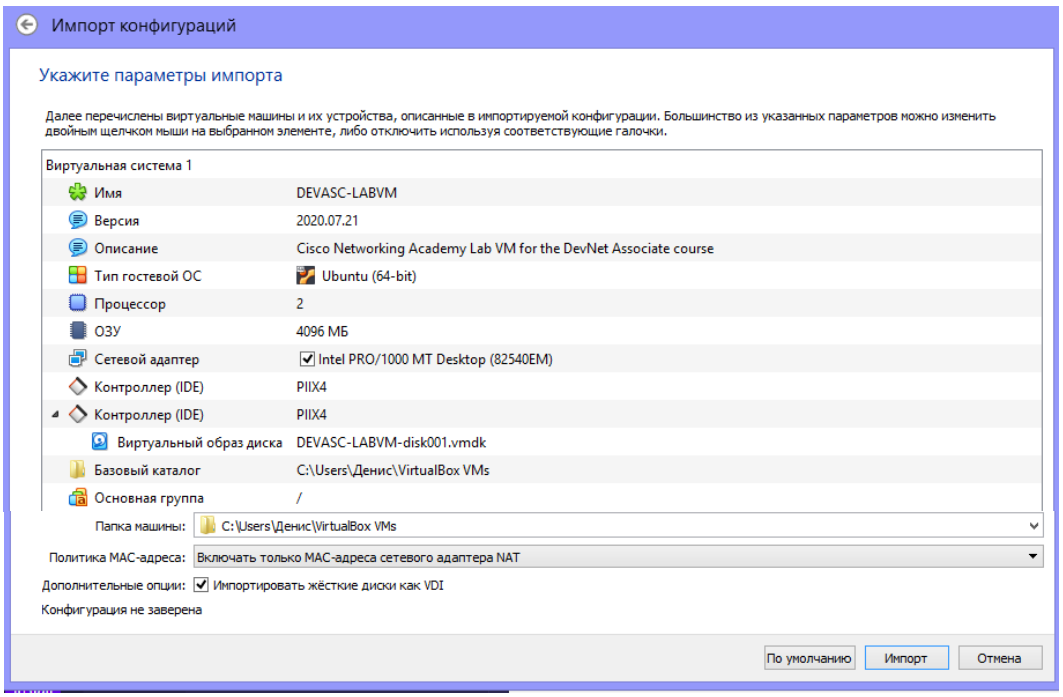
c. В VirtualBox, выберете Файл > Импорт Конфигураций.



d. В поле Источник выберите Локальная файловая система. В поле Файл укажите путь до образа загруженной виртуальной машины DEVASC VM. И нажмите далее



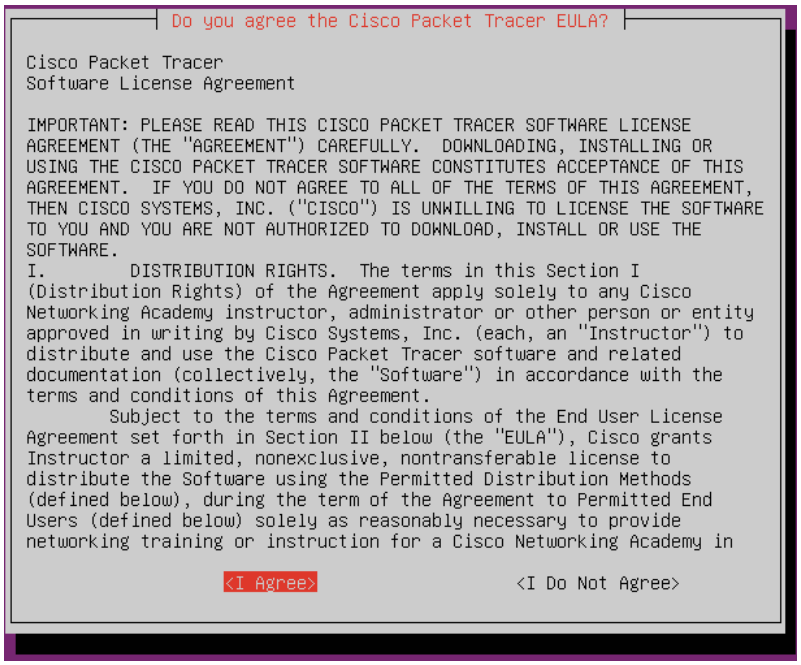
e. Нажмите Импорт что бы продолжить. Сам процесс импорта займет какое-то время.



Часть 2: Знакомство с DEVASC VM GUI

a. Запустите DEVASC VM. Это займет несколько минут, поскольку образ Ubuntu начнет загрузку.

b. DEVASC VM содержит Packet Tracer. Вы должны согласиться с Cisco Packet Tracer EULA чтобы продолжить запуск VM. Когда вы увидите лицензионное соглашение используйте клавиши со стрелками для прокрутки текста. Нажмите клавишу со стрелкой вправо, чтобы выбрать, когда закончите. Нажмите клавишу пробела, чтобы перейти к экрану Соглашение. Нажмите клавишу со стрелкой влево, чтобы выбрать нужный пункт.



c. Образ Ubuntu продолжит загрузку. Закрывайте любые всплывающие сообщения.
d. В следующих лабораторных, вы будете использовать терминал, VS Code, Packet Tracer, Браузер Chromium и Postman. Откройте эти приложения и познакомьтесь с ними.

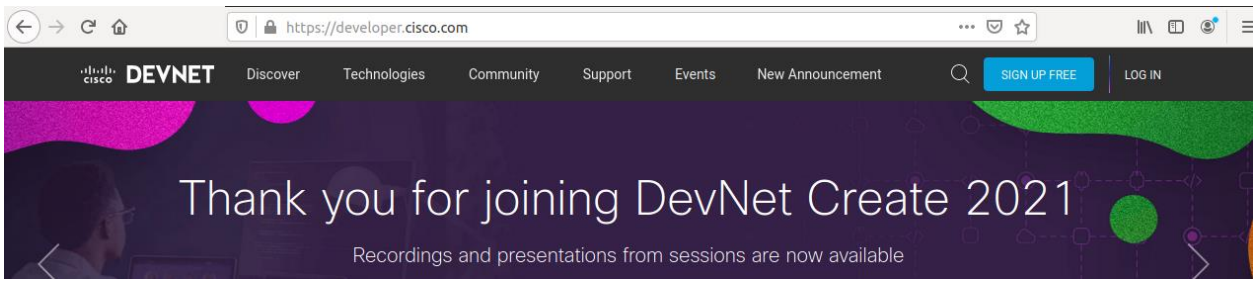
Помните, что это виртуальная машина и она полностью отделена от компьютера, на котором она установлена. Вы можете вносить любые изменения. Если вы допустите ошибку или что-то сломаете, вы можете просто удалить виртуальную машину из VirtualBox и импортировать свежую копию из загруженного файла. Это отличный способ экспериментировать с программами, не затрагивая реальный компьютер. Не бойтесь исследовать и получать удовольствие!

Часть 3: Создание Учётной записи для лабораторной среды

Существует множество инструментов, которые вам понадобится использовать для выполнения лабораторных работ в этом курсе. Некоторые из них требуют наличия собственной учетной записи. Их легко настроить и можно сделать это бесплатно. В этой части вы создадите учетные записи, которые понадобятся вам в течение всего курса.

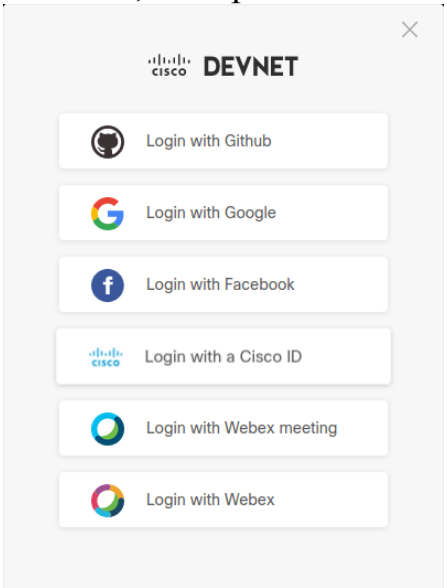
Шаг 1: Создание учетной записи DevNet.

a. Откройте Браузер Chromium/FireFox и прейдите по адресу developer.cisco.com.



b. Нажмите SIGN UP FREE.

c. Вы можете выбрать любой из типов входа, для которого у вас уже есть учетная запись. Это позволяет легко ассоциировать учетную запись с другими учетными записями. Если вы не хотите связывать свою учетную запись DevNet с другими приложениями, выберите Вход с помощью Cisco ID.

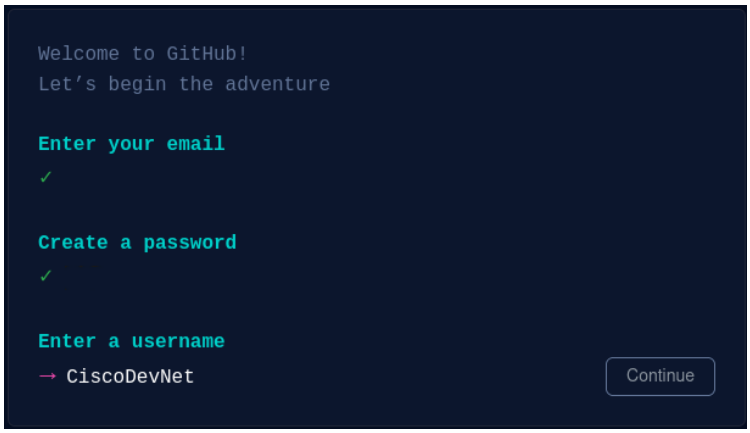


d. Следуйте инструкциям чтобы продолжить создание учетной записи.

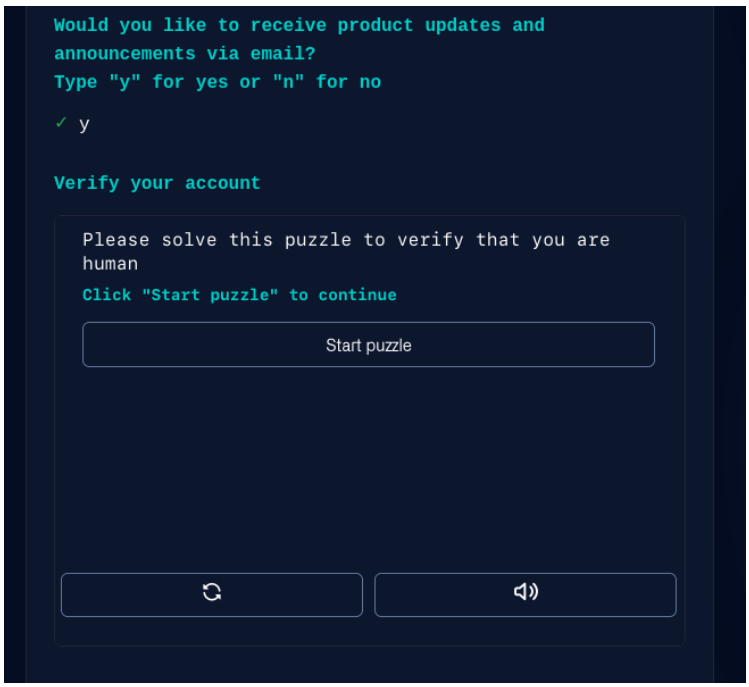
Шаг 2: Создание учетной записи GitHub.

a. Прейдите по адресу github.com и нажмите кнопку Sign Up.

b. Заполните поля Email, Password и Username и нажмите кнопку Continue.



d. Пройдите верификацию, того что вы человек. После чего нажмите кнопку Create Account



e. На открывшейся странице введите код, высланный на ваш e-mail, чтобы подтвердить свою учетную запись.

e. Проведите начальные настройки персонализации, или нажмите Skip personalization внизу страницы.

Шаг 3: Создание учетной записи Webex.

a. Прейдите по адресу webex.com.

b. Нажмите Start for Free или Sign up now, it's free.

c. Введите Email.

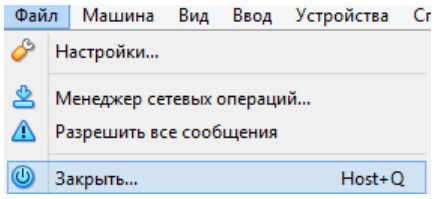
d. Нажмите Sign Up.

e. Следуйте инструкциям чтобы продолжить создание учетной записи

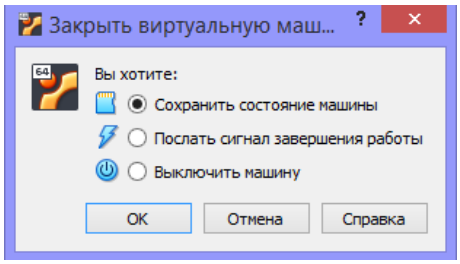
Шаг 4: Выключите VM.

Когда вы закончите работу с ВМ, вы можете сохранить состояние машины ВМ для дальнейшего использования или закрыть ВМ. Закройте ВМ с помощью графического интерфейса пользователя:

a. Из Virtual Box меню Файл, выберите Закрыть...



b. Нажмите радиокнопку Сохранить состояние машины и нажмите OK. При следующем запуске виртуальной машины вы сможете возобновить работу в операционной системе в ее текущем состоянии. Другие два варианта:
   a. Отправьте сигнал выключения: Это имитирует нажатие кнопки питания на физическом компьютере.
   b. Выключите питание машины: Это имитирует выдергивание вилки из розетки на физическом компьютере. Закрытие ВМ с помощью CLI: Чтобы выключить ВМ с помощью командной строки, можно воспользоваться опциями меню внутри ВМ или ввести команду sudo shutdown -h now в окне терминала. Перезагрузка ВМ: Если вы хотите перезагрузить ВМ, вы можете воспользоваться опциями меню внутри ВМ или ввести команду sudo reboot в терминале.



Часть 4: Установка Webex Teams на ваше устройство

В этой части, вы установите Webex Teams

Шаг 1: Загрузка и установка.

a. Прейдите по адресу www.webex.com/downloads.html
b. В разделе Webex Teams нажмите Загрузить для Windows, или вы можете выбрать установку Teams на мобильное устройство.

   Примечание: Для MacOS вы загрузите версию для Mac. На момент написания этой лабораторной работы настольного приложения для Linux не существует. Однако вы можете запустить Webex Teams в браузере. Нажмите Войти в верхней части и выберите Webex Teams. Затем войдите в систему, используя учетные данные новой учетной записи Webex.

   c. Откройте установочный файл и следуйте инструкциям для завершения установки.

Шаг 2: Запуск Webex Teams.

Откройте Webex Teams. Если ваш преподаватель уже создал команду, вы должны увидеть, что теперь являетесь ее членом. Если нет, то вы можете сделать одно из следующих действий:

- Сообщите преподавателю свой адрес электронной почты, чтобы вас можно было добавить в группу класса.

- Создайте свою собственную команду. На левой панели нажмите кнопку Команды. Затем нажмите кнопку с плюсом, чтобы Создать команду. Команда необходима для следующего шага.

Шаг 3: Добавление пользователя в Webex Teams.
a. В Webex Teams найдите пользователя в поле "Поиск". Вы можете искать по имени пользователя или электронной почте. Пользователь должен быть членом вашей команды. b. Щелкните пользователя в списке результатов поиска, чтобы открыть место.
c. Введите сообщение для пользователя. Пользователь получит уведомление о вашем сообщении и сможет ответить на него.
Содержание отчета:

1. Титульный лист
2. Скриншот VirtualBox с ипортированной рабочей машиной, а также профилей учетных записей, создаваемых в лабораторной

Лабораторная работа №1.2.2

Знакомство с Linux

Цель лабораторной работы:
Часть 1: Запуск виртуальной машины DEVASC
Часть 2: Обзор навигации по синтаксису команд
Часть 3: Обзор управления файлами
Часть 4: Обзор регулярных выражений
Часть 5: Обзор системного администрирования
Сценарий

В этой лабораторной работе вы получите первый опыт работы с Linux, включая навигацию по командам, управление файлами, работу с регулярными выражениями и навыки системного администрирования. Эта лабораторная работа не предназначена для замены предыдущего опыта работы с Linux и не обязательно охватывает все навыки Linux, необходимые для данного курса. Тем не менее, она должна послужить хорошим показателем ваших навыков работы с Linux и помочь вам определить, где вам может потребоваться дополнительное изучение определенных моментов.

Необходимые ресурсы

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

Порядок выполнения работы
Часть 1: Запуск DEVASC VM

Если вы еще не завершили **лабораторную работу - Установка лабораторной среды виртуальной машины**, сделайте это сейчас. Если вы уже завершили эту лабораторную работу, запустите виртуальную машину DEVASC.

Часть 2: Обзор навигации по синтаксису команд
В этой части вы будете использовать команды **ls**, **pwd**, **cd** и **sudo** для изучения базового синтаксиса команд.

Шаг 1: Откройте терминал в DEVASC-LABVM
Дважды щелкните значок эмулятора терминала на рабочем столе, чтобы открыть окно терминала.

Шаг 2: Навигация по директориям
Используйте команду **ls** для отображения файлов в текущем каталоге. Помните, что команды чувствительны к регистру.

```
devasc@labvm:~$ ls
Desktop    Downloads  Music     Public  Templates
Documents  labs       Pictures  snap    Videos
devasc@labvm:~$
```

Используйте команду **ls** с аргументом labs, чтобы отобразить содержимое папки labs.

```
devasc@labvm:~$ ls labs
devnet-src
devasc@labvm:~$
```

Используйте команду **ls** с опцией **-l** для отображения "длинном формате" содержимого текущего каталога.

```
devasc@labvm:~$ ls -l
total 40
drwxr-xr-x 2 devasc devasc 4096 Mar 30 21:25 Desktop
drwxr-xr-x 2 devasc devasc 4096 Apr 15 19:09 Documents
drwxr-xr-x 2 devasc devasc 4096 Apr 15 19:09 Downloads
drwxr-xr-x 5 devasc devasc 4096 Mar 30 21:21 labs
drwxr-xr-x 2 devasc devasc 4096 Apr 15 19:09 Music
drwxr-xr-x 2 devasc devasc 4096 Apr 15 19:09 Pictures
drwxr-xr-x 2 devasc devasc 4096 Apr 15 19:09 Public
drwxr-xr-x 5 devasc devasc 4096 Mar 30 21:24 snap
drwxr-xr-x 2 devasc devasc 4096 Apr 15 19:09 Templates
drwxr-xr-x 2 devasc devasc 4096 Apr 15 19:09 Videos
devasc@labvm:~$
```

Используйте команду **ls** с опцией **-r** для отображения содержимого текущего каталога в обратном алфавитном порядке.

```
devasc@labvm:~$ ls -r
Videos     snap    Pictures  labs       Documents
Templates  Public  Music     Downloads  Desktop
devasc@labvm:~$
```

Одновременно можно использовать несколько опций. Используйте команду **ls** с опциями **-l** и **-r** для отображения содержимого текущего каталога как в прямом, так и в обратном порядке.

```
devasc@labvm:~$ ls -lr
total 40
drwxr-xr-x 2 devasc devasc 4096 Apr 15 19:09 Videos
drwxr-xr-x 2 devasc devasc 4096 Apr 15 19:09 Templates
drwxr-xr-x 5 devasc devasc 4096 Mar 30 21:24 snap
drwxr-xr-x 2 devasc devasc 4096 Apr 15 19:09 Public
drwxr-xr-x 2 devasc devasc 4096 Apr 15 19:09 Pictures
drwxr-xr-x 2 devasc devasc 4096 Apr 15 19:09 Music
drwxr-xr-x 5 devasc devasc 4096 Mar 30 21:21 labs
drwxr-xr-x 2 devasc devasc 4096 Apr 15 19:09 Downloads
drwxr-xr-x 2 devasc devasc 4096 Apr 15 19:09 Documents
drwxr-xr-x 2 devasc devasc 4096 Mar 30 21:25 Desktop
devasc@labvm:~$
```

Существует множество других опций, которые можно использовать с командой **ls**. Используйте команду **man** с аргументом **ls**, чтобы посмотреть на все возможности в руководстве. Команду **man** можно использовать для поиска любой команды в системе. Для перехода к последующим экранам используйте пробел. Нажмите **q**, чтобы выйти из системы.

```
devasc@labvm:~$ man ls
(The command line disappears and the manual page for ls opens.)
LS(1)                           User Commands                          LS(1)

NAME
       ls - list directory contents
SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the FILEs (the current directory by default).
       Sort entries alphabetically if none of -cftuvSUX nor --sort  is  speci-
       fied.

       Mandatory  arguments  to  long  options are mandatory for short options
       too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

       --author
 Manual page ls(1) line 1 (press h for help or q to quit)
```

Вы также можете использовать аргумент **--help** после большинства команд, чтобы увидеть краткое описание всех доступных опций команды.

```
devasc@labvm:~$ ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.
  -a, --all                  do not ignore entries starting with .
  -A, --almost-all           do not list implied . and ..
 (полный вывод опущен)
devasc@labvm:~$
```

Используйте команду **pwd** для отображения текущей рабочей директории

```
devasc@labvm:~$ pwd
/home/devasc
devasc@labvm:~$
```

С помощью команды **cd** измените каталог на /home/devasc/Documents.

```
devasc@labvm:~$ cd Documents
devasc@labvm:~/Documents$
```

Используйте команду **cd** с символом / для смены каталога на корневой каталог. Снова используйте команду pwd, чтобы убедиться, что вы теперь находитесь в корневом каталоге.

```
devasc@labvm:~/Documents$ cd /
devasc@labvm:/$ pwd
/
devasc@labvm:/$
```

Вернитесь в каталог **/home/devasc/Documents**.
*Совет: Вы можете перемещаться по одному каталогу за раз или по всему пути до пункта назначения. Чтобы быстро ввести команду, наберите первые несколько букв имени каталога и нажмите **Tab**, чтобы система автоматически ввела остальную часть имени. Помните, что имена чувствительны к регистру.*

```
devasc@labvm:/$ cd /home/devasc/Documents/
devasc@labvm:~/Documents$
```

Используйте символы **..** для перемещения вверх на один каталог. Снова используйте **pwd**, чтобы убедиться, что вы вернулись в домашний каталог пользователя.

```
devasc@labvm:~/Documents$ cd ..
devasc@labvm:~$ pwd
/home/devasc
devasc@labvm:~$
```

Шаг 3: Используйте команды суперпользователя для административного доступа.

Используйте команду **sudo**, чтобы выполнить одну команду от имени пользователя root. При этом новый терминал не будет создан. Используйте команду **sudo apt-get update** для обновления списка доступных пакетов, установленных на ВМ. Эта команда не будет работать без использования команды sudo.

Примечание: Ваш output, скорее всего, будет отличаться.

```
devasc@labvm:~$ sudo apt-get update
Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [97.9 kB]
Get:2 http://us.archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:3 http://us.archive.ubuntu.com/ubuntu focal-updates InRelease [89.1 kB]
Get:4 http://us.archive.ubuntu.com/ubuntu focal-backports InRelease [89.2 kB]
Get:5 http://us.archive.ubuntu.com/ubuntu focal/main i386 Packages [723 kB]
Get:6 http://us.archive.ubuntu.com/ubuntu focal/main amd64 Packages [981 kB]
(полный вывод опущен)
Fetched 677 kB in 2s (346 kB/s)
Reading package lists... Done
devasc@labvm:~$
```

Часть 3: Обзор навигации по синтаксису команд

В этой части вы рассмотрите права доступа файлов, изменение прав и прав владельца, перемещение файлов, копирование файлов, удаление файлов и просмотр файлов.

Шаг 1: Проверьте права файлов.

Используйте команду **ls Desktop -l** для отображения содержимого папки "Рабочий стол".

```
devasc@labvm:~$ ls Desktop -l
total 28
-rwxr-xr-x 1 devasc devasc 1095 Mar 30 21:24 chromium_chromium.desktop
-rwxr-xr-x 1 devasc devasc  401 Mar 30 21:25 cisco-packet-tracer_cisco-pacet-
tracer.desktop
-rwxr-xr-x 1 devasc devasc  776 Mar 30 21:23 code.desktop
-rwxr-xr-x 1 devasc devasc  373 Mar 30 21:25 drawio_drawio.desktop
-rwxr-xr-x 1 devasc devasc  250 Mar 30 21:21 exo-terminal-emulator.desktop
-rwxr-xr-x 1 devasc devasc   99 Mar 30 21:21 labs.desktop
-rwxr-xr-x 1 devasc devasc  334 Mar 30 21:24 postman_postman.desktop
devasc@labvm:~$
```

Ответьте на следующие вопросы относительно приведенного выше вывода. Если необходимо, найдите в Интернете информацию о разрешении файлов Linux, показанном в выводе команды **ls**.

Что означает начальное тире в информации о правах файлов?

- Предполагаемый ответ: Это поле типа файла. Прочерк/Тире обозначает обычный файл.

Что будет стоять на месте тире, если речь идет о каталоге?

- Предполагаемый ответ: На месте тире, окажется буква d — "directory"

Что означают следующие три буквы или тире в информации о правах файлов?

- Предполагаемый ответ: Они представляют собой разрешения владельца файла на его использование.

Что означают три средние буквы или тире в информации о разрешении?

- Предполагаемый ответ: Они представляют собой права доступа группы к файлу.

Что означают последние три буквы или тире в информации о разрешении?

- Предполагаемый ответ: Они представляют собой разрешения других лиц на этот файл.

На что указывает первое слово "**devasc**" в информации о правах доступа?

- Предполагаемый ответ: Указывает на поле владельца пользователя, который является владельцем файла.

На что указывает второй экземпляр "**devasc**" в информации о разрешении?

- Предполагаемый ответ: Указывает на поле владельца группы и является группой файла.

Что означает тип разрешения "r"?

- Предполагаемый ответ: Буква "r" означает разрешение на "чтение" файла; позволяет читать или копировать содержимое файла.

Что означает тип разрешения "w"?

- Предполагаемый ответ: Буква "w" означает разрешение на "запись" файла; позволяет изменять или перезаписывать содержимое файла. А также добавлять или удалять файлы из каталога.

Что означает тип разрешения "x"?

- Предполагаемый ответ: Буква "x" означает разрешение на "исполнение" файла; позволяет запускать файл как процесс, хотя файлы сценариев также требуют разрешения на чтение.

Шаг 2: Изменение прав доступа и владельца файла.

С помощью команды **cd** перейдите в каталог Документы.

```
devasc@labvm:~$ cd Documents/
devasc@labvm:~/Documents$
```

С помощью команды **echo** создайте файл сценария оболочки, внутри которого будет команда **ls ../Desktop**. Помните, что символ больше чем (>) перенаправляет вывод команды в файл.

```
devasc@labvm:~/Documents$ echo "ls ../Desktop" > myfile.sh
devasc@labvm:~/Documents$
```

Сценарий **myfile.sh** хранится в каталоге **/Documents**. Используйте команду **cat** для просмотра единственной команды в сценарии. Этот файл будет использоваться в качестве примера для изменения прав доступа и собственности.

```
devasc@labvm:~/Documents$ cat myfile.sh
ls ../Desktop
devasc@labvm:~/Documents$
```

Используйте команду **./myfile.sh** для запуска сценария.

```
devasc@labvm:~/Documents$ ./myfile.sh
bash: ./myfile.sh: Permission denied
```

Доступ запрещен, так как для файла необходимо установить разрешение executable. Используйте команду **ls -l myfile.sh** для просмотра текущих разрешений файлов.

```
devasc@labvm:~/Documents$ ls -l myfile.sh
-rw-rw-r-- 1 devasc devasc 14 Apr 16 12:46 myfile.sh
```

Используйте команду **chmod +x myfile.sh**, чтобы разрешить выполнение файла.

```
devasc@labvm:~/Documents$ chmod +x myfile.sh
devasc@labvm:~/Documents$
```

Снова попробуйте запустить скрипт **./myfile.sh**

```
devasc@labvm:~/Documents$ ./myfile.sh
chromium_chromium.desktop              exo-terminal-emulator.desktop
cisco-packet-tracer_cisco-pacet-tracer.desktop      labs.desktop
code.desktop                           postman_postman.desktop
drawio_drawio.desktop
devasc@labvm:~/Documents$
```

Отобразите разрешения файла myfile.sh.

```
devasc@labvm:~/Documents$ ls -l
```

```
total 4
-rwxrwxr-x 1 root devasc 14 Apr 16 21:28 myfile.sh
devasc@labvm:~/Documents$
```

Шаг 3: Использование команды перемещения файлов.

С помощью команды **mv** переместите файл **myfile.sh** на рабочий стол.

```
devasc@labvm:~/Documents$ mv myfile.sh /home/devasc/Desktop/
devasc@labvm:~/Documents$
```

Отобразите содержимое папки "Рабочий стол"

```
devasc@labvm:~/Documents$ ls ../Desktop/
chromium_chromium.desktop                          exo-terminal-emulator.desktop
cisco-packet-tracer_cisco-pacet-tracer.desktop     labs.desktop
code.desktop                                       myfile.sh
drawio_drawio.desktop                              postman_postman.desktop
devasc@labvm:~/Documents$
```

Верните файл в папку Документы.

```
devasc@labvm:~/Documents$ mv ../Desktop/myfile.sh myfile.sh
devasc@labvm:~/Documents$
```

С помощью команды **mv** переименуйте myfile.sh в myfile_renamed.sh.

```
devasc@labvm:~/Documents$ mv myfile.sh myfile_renamed.sh
devasc@labvm:~/Documents$ ls
myfile_renamed.sh
devasc@labvm:~/Documents$
```

Шаг 4: Использование команды копирования файлов.

С помощью команды **cp** создайте копию файла **myfile_renamed.sh**.

```
devasc@labvm:~/Documents$ cp myfile_renamed.sh
myfile_renamed_and_copied.sh
devasc@labvm:~/Documents$ ls
myfile_renamed_and_copied.sh  myfile_renamed.sh
devasc@labvm:~/Documents$
```

Шаг 5: Использование команду удаления файлов.

Используйте команду **rm** для удаления файла **myfile_renamed_and_copied.sh**.

```
devasc@labvm:~/Documents$ rm myfile_renamed_and_copied.sh
devasc@labvm:~/Documents$ ls
mbr.img  myfile_renamed.sh
devasc@labvm:~/Documents$
```

Шаг 6: Перенаправление стандартного потока вывода.

С помощью оператора перенаправления (**>**) поместите текст в новый файл с именем **linux.txt**.

```
devasc@labvm:~$ echo "Я съел деда!" > linux.txt
devasc@labvm:~$
```

С помощью команды **cat** перенаправьте содержимое файла **linux.txt** в другой файл.

```
devasc@labvm:~$ cat linux.txt > linux2.txt
devasc@labvm:~$
```

С помощью команды **cat** просмотрите содержимое файла **linux2.txt**.

```
devasc@labvm:~$ cat linux2.txt
Я съел деда!
devasc@labvm:~$
```

Используйте команду **echo** для добавления текста в файл **linux2.txt**.

```
devasc@labvm:~$ echo "I LOVE Linux!" >> linux2.txt
devasc@labvm:~$
```

Используйте команду **cat** для просмотра содержимого файла **linux2.txt**.

```
devasc@labvm:~$ cat linux2.txt
Я съел деда!
I LOVE Linux!
devasc@labvm:~$
```

Используйте команду **echo** для перезаписи содержимого файла с помощью одинарной угловой скобки.

```
devasc@labvm:~$ echo "Linux is POWERFUL!" > linux.txt
devasc@labvm:~$
```

Используйте команду **cat** для просмотра содержимого файла **linux.txt**. *Обратите внимание, что предыдущее состояние "Я съел деда!" было перезаписано.*

```
devasc@labvm:~$ cat linux.txt
Linux is POWERFUL!
devasc@labvm:~$
```

Шаг 7: Используйте текстовый редактор vi.

Используйте следующую команду для запуска текстового редактора **vi** и открытия текстового файла.

```
devasc@labvm:~$ vi linux2.txt
```

В окне редактора отображается следующее содержимое:

```
Linux is AWESOME!
I LOVE Linux!
```

С помощью текстового редактора измените содержимое на следующее:

```
Linux is Linux
I am AWESOME!
```

*Клавиша **a** позволит вам войти в режим редактирования, добавляя текст после позиции курсора, а клавиша **i** позволит вам войти в режим редактирования, вставляя текст в позицию курсора. Чтобы перейти в командный режим, вам понадобится клавиша **Esc**. Помните, что **d** удалит (вырежет), **y** выдернет (скопирует), а **p** вставит (вставит) текущую строку с курсором.*

Сохраните текст в новом файле с именем "**linux3.txt**". Помните, что вам нужно будет находиться в командном режиме и набрать двоеточие **(:)** для входа в режим **ex**, чтобы вы могли записать (сохранить) документ **(:w linux3.txt)**. Затем вы можете использовать команду **quit** (выход) **(:q)** для выхода из редактора **vi**. Также можно сэкономить на спичках используя команду **:wq**

Используйте команду cat для просмотра содержимого файла linux3.txt.

```
devasc@labvm:~$ cat linux3.txt
Linux is Linux
I am AWESOME!
devasc@labvm:~$
```

Часть 4: Обзор регулярных выражений

В этой части вы используете команду **grep**, чтобы рассмотреть, как можно использовать регулярные выражения для фильтрации.

*Примечание: Ваши результаты могут отличаться от показанных ниже, поскольку состояние виртуальной машины основано на последней загруженной вами итерации, а также на любых изменениях, которые вы могли внести. Тем не менее, вы должны получить некоторые результаты из файла **passwd**, но ваш выделенный результат будет отличаться.*

Используйте команду **grep** для фильтрации содержимого файла **passwd**, чтобы отобразить строку из файла **passwd**, содержащую **devasc**. Обратите внимание, что два экземпляра devasc выделены. Также обратите внимание, что команда **grep** чувствительна к регистру. Экземпляр DEVASC не выделяется.

```
devasc@labvm:~$ grep devasc /etc/passwd
devasc:x:900:900:DEVASC,,,:/home/devasc:/bin/bash
devasc@labvm:~$
```

Используйте команду **grep**, чтобы показать, сколько раз **root** встречается в файле **passwd**. Обратите внимание, что все три экземпляра **root** выделены.

```
devasc@labvm:~$ grep root /etc/passwd
root:x:0:0:root:/root:/bin/bash
devasc@labvm:~$
```

Используйте команду **grep** с символом якоря (циркумфлекс) ^, чтобы найти слово, но только в начале строки. Обратите внимание, что выделено только слово в начале строки.

```
devasc@labvm:~$ grep '^root' /etc/passwd
root:x:0:0:root:/root:/bin/bash
devasc@labvm:~$
```

Используйте команду **grep** с символом якоря (доллара) **$**, чтобы найти слово в конце строки.

```
devasc@labvm:~$ grep 'false$' /etc/passwd
tss:x:106:114:TPM software stack,,,:/var/lib/tpm:/bin/false
lightdm:x:107:117:Light Display Manager:/var/lib/lightdm:/bin/false
hplip:x:115:7:HPLIP system user,,,:/run/hplip:/bin/false
devasc@labvm:~$
```

Используйте команду **grep** с символом якоря (точка) **.** для поиска слов определенной длины с разными буквами в них. Обратите внимание, что выделен не только **daem**, но и **dnsm**.

```
devasc@labvm:~$ grep 'd..m' /etc/passwd
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
dnsmasq:x:109:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
avahi-autoipd:x:110:121:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/usr/sbin/nologin
usbmux:x:111:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
avahi:x:113:122:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/usr/sbin/nologin
colord:x:116:125:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin
pulse:x:117:126:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
devasc@labvm:~$
```

Используйте команду **grep** для поиска строк, в которых присутствуют только числа 8 или 9. Обратите внимание, что возвращаются только строки, содержащие 8, 9 или оба числа.

```
devasc@labvm:~$ grep '[8-9]' /etc/passwd
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
uuidd:x:103:109::/run/uuidd:/usr/sbin/nologin
devasc:x:900:900:DEVASC,,,:/home/devasc:/bin/bash
systemd-network:x:999:999:systemd Network Management:/:/usr/sbin/nologin
systemd-resolve:x:998:998:systemd Resolver:/:/usr/sbin/nologin
systemd-timesync:x:997:997:systemd Time Synchronization:/:/usr/sbin/nologin
systemd-coredump:x:996:996:systemd Core Dumper:/:/usr/sbin/nologin
rtkit:x:108:119:RealtimeKit,,,:/proc:/usr/sbin/nologin
dnsmasq:x:109:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
devasc@labvm:~$
```

Используйте команду **grep** для поиска литеральных символов. Обратите внимание, что возвращаются только строки, содержащие запятую.

```
devasc@labvm:~$ grep '[,]' /etc/passwd
devasc:x:900:900:DEVASC,,,:/home/devasc:/bin/bash
tss:x:106:114:TPM software stack,,,:/var/lib/tpm:/bin/false
rtkit:x:108:119:RealtimeKit,,,:/proc:/usr/sbin/nologin
dnsmasq:x:109:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
avahi-autoipd:x:110:121:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/usr/sbin/nologin
usbmux:x:111:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
```

```
kernoops:x:112:65534:Kernel Oops Tracking Daemon,,,:/:/usr/sbin/nologin
avahi:x:113:122:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/usr/sbin/nologin
hplip:x:115:7:HPLIP system user,,,:/run/hplip:/bin/false
colord:x:116:125:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin
pulse:x:117:126:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
devasc@labvm:~$
```

Используйте команду **grep** для поиска вхождений нулевого или более шаблона, предшествующего ему. Обратите внимание, что возвращаются только строки с new и ne.

```
devasc@labvm:~$ grep 'new*' /etc/passwd
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
messagebus:x:100:103::/nonexistent:/usr/sbin/nologin
_apt:x:102:65534::/nonexistent:/usr/sbin/nologin
tcpdump:x:104:110::/nonexistent:/usr/sbin/nologin
systemd-network:x:999:999:systemd Network Management:/:/usr/sbin/nologin
kernoops:x:112:65534:Kernel Oops Tracking Daemon,,,:/:/usr/sbin/nologin
saned:x:114:124::/var/lib/saned:/usr/sbin/nologin
devasc@labvm:~$
```

Часть 5: Обзор Системного Администрирования

В этой части вы рассмотрите основные задачи системного администрирования Linux, включая выключение компьютера, просмотр и тестирование конфигурации сети, просмотр процессов, управление установочными пакетами, обновление паролей пользователей, добавление содержимого в файлы и использование текстовых редакторов.

Шаг 1: Выключение компьютера.

Используйте команду **shutdown now**, чтобы инициировать немедленное завершение работы ОС (и ВМ). Вам не нужно выполнять это действие, так как ВМ выключится, и вам нужно будет перезапустить ее вручную. Форматом аргумента time может быть слово **now**, время суток в формате **hh:mm** или количество минут задержки в формате **+minutes**.

```
devasc@labvm:~$ shutdown now
```

Используйте команду **date** для проверки даты установки ОС.

```
devasc@labvm:~$ date
Fri 4 Feb 2022 08:53:20 PM UTC
devasc@labvm:~$
```

Используйте команду **shutdown +1 "Come back soon!"**, чтобы выключить ОС через 1 минуту и вывести сообщение "Come back soon!". Не забудьте отменить команду, иначе ваша ВМ выключится.

```
devasc@labvm:~$ shutdown +1 "Come back soon!"
Shutdown scheduled for Fri 2022-02-04 20:57:13 UTC, use 'shutdown -c' to cancel.
devasc@labvm:~$ shutdown -c
devasc@labvm:~$
```

Шаг 2: Просмотр и тестирование конфигурации сети.

Используйте команду **ip address** для отображения конфигурации сети. Вывод будет немного более подробным. Например, обратите внимание, что для интерфейса dummy0 показаны пять адресов IPv4.

```
devasc@labvm:~$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:ce:2b:8b brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic enp0s3
       valid_lft 75746sec preferred_lft 75746sec
    inet6 fe80::a00:27ff:fece:2b8b/64 scope link
       valid_lft forever preferred_lft forever
3: dummy0: <BROADCAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ether 46:8b:41:b5:de:aa brd ff:ff:ff:ff:ff:ff
    inet 192.0.2.1/32 scope global dummy0
       valid_lft forever preferred_lft forever
    inet 192.0.2.2/32 scope global dummy0
       valid_lft forever preferred_lft forever
    inet 192.0.2.3/32 scope global dummy0
       valid_lft forever preferred_lft forever
    inet 192.0.2.4/32 scope global dummy0
       valid_lft forever preferred_lft forever
    inet 192.0.2.5/32 scope global dummy0
       valid_lft forever preferred_lft forever
    inet6 fe80::448b:41ff:feb5:deaa/64 scope link
       valid_lft forever preferred_lft forever
devasc@labvm:~$
```

Используйте команду **ping** с параметрами -c 4, чтобы четыре раза пропинговать компьютер в локальной сети. Вы должны использовать действительный IP-адрес устройства в вашей локальной сети. В следующем примере используется 192.168.1.1, но в вашей сети, скорее всего, будут другие IPv4-адреса.

```
devasc@labvm:~$ ping -c 4 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=63 time=1.13 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=63 time=2.30 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=63 time=1.31 ms
64 bytes from 192.168.1.1: icmp_seq=4 ttl=63 time=2.49 ms

--- 192.168.1.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 1.130/1.809/2.492/0.594 ms
devasc@labvm:~$
```

Вы также можете пинговать имя, и система доменных имен (DNS) преобразует имя в IP-адрес. Например, выполнить ping веб-сайта Cisco. Ваша виртуальная машина сначала отправит DNS-запрос, чтобы получить IP-адрес, а затем отправит пакеты ping. Процесс DNS не отображается в выводе ping.

```
devasc@labvm:~$ ping -c 4 www.cisco.com
PING e2867.dsca.akamaiedge.net (23.204.11.200) 56(84) bytes of data.
64      bytes      from      a23-204-11-200.deploy.static.akamaitechnologies.com
(23.204.11.200): icmp_seq=1 ttl=58 time=185 ms
64      bytes      from      a23-204-11-200.deploy.static.akamaitechnologies.com
(23.204.11.200): icmp_seq=2 ttl=58 time=28.8 ms
64      bytes      from      a23-204-11-200.deploy.static.akamaitechnologies.com
(23.204.11.200): icmp_seq=3 ttl=58 time=28.8 ms
64      bytes      from      a23-204-11-200.deploy.static.akamaitechnologies.com
(23.204.11.200): icmp_seq=4 ttl=58 time=26.4 ms

--- e2867.dsca.akamaiedge.net ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3007ms
rtt min/avg/max/mdev = 26.443/67.339/185.363/68.147 ms
devasc@labvm:~$
```

Шаг 3: Просмотр текущих процессов

Используйте команду **ps** для отображения процессов, запущенных в текущем терминале.

```
devasc@labvm:~$ ps
    PID TTY          TIME CMD
   1416 pts/0    00:00:00 bash
   1453 pts/0    00:00:00 ps
devasc@labvm:~$
```

Используйте команду **ps** с параметром **-e** для отображения всех процессов, запущенных на компьютере.

```
devasc@labvm:~$ ps -e
    PID TTY          TIME CMD
      1 ?        00:00:01 systemd
      2 ?        00:00:00 kthreadd
      3 ?        00:00:00 rcu_gp
      4 ?        00:00:00 rcu_par_gp
      6 ?        00:00:00 kworker/0:0H-kblockd
      7 ?        00:00:00 kworker/0:1-events
      9 ?        00:00:00 mm_percpu_wq
 (полный вывод опущен)
```

Вы можете направить вывод любой команды на один экран за раз, добавив **| more**. Отобразится один экран вывода с надписью - more-, показанной внизу. Теперь вы можете использовать клавишу Enter для вывода одной строки за раз, пробел для вывода одного экрана за раз или Ctrl+C для выхода и возврата в командную строку.

```
devasc@labvm:~$ ps -e | more
    PID TTY          TIME CMD
      1 ?        00:00:01 systemd
```

```
      2 ?        00:00:00 kthreadd
      3 ?        00:00:00 rcu_gp
      4 ?        00:00:00 rcu_par_gp
      6 ?        00:00:00 kworker/0:0H-kblockd
      9 ?        00:00:00 mm_percpu_wq
     10 ?        00:00:00 ksoftirqd/0
--More--
```

Используйте ps с параметром **-ef** для более детального отображения всех процессов, запущенных на компьютере.

```
devasc@labvm:~$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 20:57 ?        00:00:01 /sbin/init
root           2       0  0 20:57 ?        00:00:00 [kthreadd]
root           3       2  0 20:57 ?        00:00:00 [rcu_gp]
root           4       2  0 20:57 ?        00:00:00 [rcu_par_gp]
root           6       2  0 20:57 ?        00:00:00 [kworker/0:0H-kblockd]
root           9       2  0 20:57 ?        00:00:00 [mm_percpu_wq]
root          10       2  0 20:57 ?        00:00:00 [ksoftirqd/0]
root          11       2  0 20:57 ?        00:00:01 [rcu_sched]
(полный вывод опущен)
```

Шаг 4: Управление пакетами

Используйте команду **apt-get update**, чтобы обновить список доступных пакетов в ОС, как было показано ранее в части 1 этой лабораторной работы. Вы должны использовать разрешения административного уровня для использования этой команды.

```
devasc@labvm:~$ sudo apt-get update
Hit:1 http://security.ubuntu.com/ubuntu focal-security InRelease
Get:2 http://us.archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Hit:3 http://us.archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:4 http://us.archive.ubuntu.com/ubuntu focal-backports InRelease
Get:5 http://us.archive.ubuntu.com/ubuntu focal/main i386 Packages [721 kB]
Get:6 http://us.archive.ubuntu.com/ubuntu focal/main amd64 Packages [974 kB]
Get:7 http://us.archive.ubuntu.com/ubuntu focal/main Translation-en [506 kB]
(полный вывод опущен)
```

Используйте команду **apt-cache search** для поиска определенного пакета.

```
devasc@labvm:~$ apt-cache search speed test
(полный вывод опущен)
smalt-examples - Sequence Mapping and Alignment Tool (examples)
speedtest-cli - Command line interface for testing internet bandwidth using
speedtest.net
sup - Software Upgrade Protocol implementation
sysbench - multi-threaded benchmark tool for database systems
tcpreplay - Tool to replay saved tcpdump files at arbitrary speeds (полный вывод
опущен)
```

Используйте команду **apt-get install** для установки пакета.

```
devasc@labvm:~$ sudo apt-get install speedtest-cli
Reading package lists... Done
```

```
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  speedtest-cli
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 23.9 kB of archives.
After this operation, 106 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu focal/universe amd64 speedtest-cli all
2.1.2-2 [23.9 kB]
Fetched 23.9 kB in 1s (43.9 kB/s)
Selecting previously unselected package speedtest-cli.
(полный вывод опущен)
```

Теперь вы можете использовать команду **speedtest-cli** для проверки текущей скорости подключения к Интернету.

```
devasc@labvm:~$ speedtest-cli
Retrieving speedtest.net configuration...
Testing from Cable Company (192.168.100.21)...
Retrieving speedtest.net server list...
Selecting best server based on ping...
Hosted by Comcast (Albuquerque, NM) [494.76 km]: 48.636 ms
Testing                                                                  download
speed.......................................................................
......
Download: 90.87 Mbit/s
Testing                                                                    upload
speed.......................................................................
...........................
Upload: 17.87 Mbit/s
devasc@labvm:~$
```

Используйте команду **apt-get upgrade** для обновления всех пакетов и зависимостей на компьютере.

```
devasc@labvm:~$ sudo apt-get upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following packages have been kept back:
  libnss-systemd libpam-systemd libsystemd0 libyelp0 linux-generic linux-headers-
generic
(полный вывод опущен)
```

Используйте команду **apt-get purge**, чтобы полностью удалить пакет с компьютера.

```
devasc@labvm:~$ sudo apt-get purge speedtest-cli
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be REMOVED:
  speedtest-cli*
0 upgraded, 0 newly installed, 1 to remove and 0 not upgraded.
After this operation, 106 kB disk space will be freed.
```

```
Do you want to continue? [Y/n]
(Reading database ... 211937 files and directories currently installed.)
Removing speedtest-cli (2.1.2-2) ...
(полный вывод опущен)
```

Шаг 5: Обновление паролей

Используйте команду **passwd** для обновления пароля.

Примечание: Если вы действительно меняете пароль для пользователя devasc, убедитесь, что вы его запомнили.

```
devasc@labvm:~$ passwd
Changing password for devasc.
Current password:
New password:
Retype new password:
passwd: password updated successfully
devasc@labvm:~$
```

Используйте команду **passwd** с параметром **-S** для просмотра состояния пароля.

```
devasc@labvm:~$ passwd -S
devasc P 02/04/2022 0 99999 7 -1
devasc@labvm:~$
```

Используйте страницы руководства по команде **passwd (man passwd)** для изучения опции **-S** и поиска ответов на следующие вопросы.

Каков текущий статус пароля?

- **Предполагаемый ответ:** P означает, что пароль пригоден для использования.

Какое минимальное количество дней должно пройти, чтобы пароль можно было изменить?

- **Предполагаемый ответ:** 0

Какое количество дней после истечения срока действия пароля учетная запись остается активной?

- **Предполагаемый ответ:**-1 означает, что срок действия пароля никогда не истекает из-за неактивности.

Содержание отчета:

1. Титульный лист
2. Скриншоты промежуточных действий по каждой из Частей.
3. Ответы на вопросы из Частей 3 и 5.

Лабораторная работа №1.3.3
Знакомство с Языком Программирования Python

Часть 1: Запуск виртуальной машины DEVASC
Часть 2: Запуск Python и VS Code
Часть 3: Обзор типов данных и переменных
Часть 4: Обзор списков и словарей
Часть 5: Обзор функции ввода
Часть 6: Обзор функций If, For и While
Часть 7: Обзор методов доступа к файлам

Сценарий

В этой лабораторной работе вы обретете базовые навыки и знания программирования на языке Python, такие как типы данных, переменные, списки, словари, ввод данных пользователем, операторы if, циклы for и while, а также доступ к файлам. Эта лабораторная работа не предназначена для замены предыдущего опыта программирования и не обязательно охватывает все навыки работы с Python. Тем не менее, она может проверить ваши навыки программирования на Python и помочь определить, где вам может потребоваться дополнительное изучение материала.

Примечание: *Напоминание. При написании скриптов обязательно соблюдайте правильные отступы в Python. Если вам нужен учебник, найдите в Интернете " Python indention rules".*

Необходимые ресурсы

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

Порядок выполнения работы

Часть 1: Запуск DEVASC VM

Если вы еще не завершили **лабораторную работу - Установка лабораторной среды виртуальной машины**, сделайте это сейчас. Если вы уже завершили эту лабораторную работу, запустите виртуальную машину DEVASC.

Часть 2: Запуск Python и VS Code

В этой части вы рассмотрите запуск интерактивного интерпретатора Python и использование Visual Studio Code для написания и запуска сценария "Hello World".

Шаг 1: Запустите Python.

Чтобы проверить версию Python, запущенную в виртуальной машине, откройте окно терминала и введите команду **python3 -V**. Эту команду удобно выполнять, если вы устанавливаете Python на другой компьютер или хотите проверить, какая версия установлена.

```
devasc@labvm:~$ python3 -V
Python 3.8.2
```

Примечание: Вам нужно будет изменить его на **python2 -V**, если на другом используемом вами устройстве используется версия 2. Однако, начиная с 1 января 2020 года, Python 2 больше не поддерживается.

Примечание: На момент написания этой лабораторной работы Python 3.10.2 был последней версией. Хотя вы можете обновить свою установку Python с помощью команды sudo apt-get install python3, эта лабораторная работа и остальные лабораторные работы в этом учебном треке будут основаны на Python 3.8.2.

Чтобы запустить Python, введите python3. Три угловые скобки (>>>) указывают на то, что вы находитесь в интерактивном интерпретаторе Python.

Шаг 2: Используйте Интерпретатор в качестве калькулятора.
Отсюда вы можете выполнять различные базовые задачи программирования, включая математические операции. В таблице показан синтаксис языка Python, который следует использовать для наиболее распространенных математических операций.

Операция	Математическая запись	Синтаксис
Сложение	a+b	a+b
Вычитание	a-b	a-b
Умножение	axb	a*b
Деление	a÷b	a/b
Возведение в степень	a^b	a**b

Введите несколько математических операций, используя синтаксис Python, как показано в примерах.

```
>>> 2+3
5
>>> 10-4
6
>>> 2*4
8
>>> 20/5
4.0
>>> 3**2
9
```

Напомним, что в Python используется стандартный порядок операций, известный как PEMDAS. Математические выражения оцениваются в следующем порядке.

Parentheses — Скобки

Exponents — Возведение в степень

Multiplication и **D**ivision — Умножение и Деление

Addition и **S**ubtraction — Сложение и Вычитание

Попробуйте ввести выражение со сложным порядком операций в интерактивном интерпретаторе.

Шаг 3: Используйте интерактивный интерпретатор для печати строки.
Строка — это любая последовательность символов, таких как буквы, цифры, символы или знаки препинания. Интерактивный интерпретатор будет напрямую выводить текст, который вы вводите как строку, если вы заключаете строку в одинарные кавычки (**'**) или двойные кавычки (**"**).

Введите **"Привет Мир!"** или **Привет Мир!'** в интерактивном интерпретаторе.

```
>>> "Привет Мир"
'Привет Мир!'
>>> 'Привет Мир'
'Привет Мир!'
```

Команду **print** можно также использовать непосредственно в интерактивном интерпретаторе.

```
>>> print("Привет Мир!")
Привет Мир!
```

Чтобы выйти из интерактивного интерпретатора, введите **quit().**

```
>>> quit()
devasc@labvm:~$
```

Шаг 4: Откройте VS Code и создайте скрипт Привет Мир.
Существует множество сред разработки, доступных разработчикам для управления своими проектами. В этом курсе лабораторных работах будет использоваться установленная на ВМ Visual Studio Code (VS Code) от Microsoft.

Откройте VS Code. Если вы работаете впервые, скорее всего, перед вами откроется окно приветствия.

Не стесняйтесь изучать меню и опции VS Code в свободное время. А пока нажмите **File > New File**, чтобы открыть новый файл.

В новом файле введите команду **print** из предыдущего шага.

Сохраните сценарий под именем **hello-world.py** в папке **labs/devnet-src/python**. Обязательно добавьте расширение **.py** для файла Python.

Чтобы запустить ваш скрипт, нажмите **Run > Run Without Debugging**. Откроется окно терминала внутри VS Code, запустите код для запуска экземпляра Python, запустите ваш скрипт, затем выйдите из Python обратно в командную строку

```
devasc@labvm:~/labs/devnet-src/python$  env DEBUGPY_LAUNCHER_PORT=36095
/usr/bin/python3 /home/devasc/.vscode/extensions/ms-python.python-
2020.4.76186/pythonFiles/lib/python/debugpy/no_wheels/debugpy/launcher
/home/devasc/labs/devnet-src/python/hello-world.py
Привет мир!
devasc@labvm:~/labs/devnet-src/python$
```

Теперь, когда у вас открыта командная строка внутри VS Code, вы можете вручную запустить Python и быстро запустить свой сценарий с помощью следующей команды.

```
devasc@labvm:~/labs/devnet-src/python$ python3 hello-world.py
Привет мир!
devasc@labvm:~/labs/devnet-src/python$
```

Вы также можете открыть окно терминала вне VS Code и ввести ту же команду, удостоверившись, что указали информацию о пути.

Далее, вы, как правило, будете запускать свои программы непосредственно в VS Code.

Часть 3: Обзор типов данных и переменных

В этой части вы будете использовать интерактивный интерпретатор для обзора типов данных, создания переменных, конкатенации строк и преобразования между типами данных.

Шаг 1: Используйте интерактивный интерпретатор для изучения основных типов данных.

В программировании типы данных — это классификация, которая сообщает интерпретатору, как разработчик намерен использовать данные. Например, интерпретатору необходимо знать, являются ли данные, введенные пользователем, числом или строкой. Хотя существует несколько различных типов данных, мы остановимся только на следующих:

- **Integer** — используется для целых чисел (без десятичных знаков), таких как 1, 2, 3 и так далее. Если целое число вводится с десятичной

дробью, интерпретатор игнорирует дробную часть. Т.е. 3,75 интерпретируется как 3.

- **Float** — используется для указания чисел, с плавающей точкой, например, 3.14159.
- **String** — любая последовательность символов, таких как буквы, цифры, символы или знаки препинания.
- **Boolean** — любой тип данных, который имеет значение True или False.

Используйте команду **type()** для определения основных типов данных: int, float, string, Boolean.

```
devasc@labvm:~/labs/devnet-src$ python3
Python 3.8.2 (default, Mar 13 2022, 10:14:16)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> type(98)
<class 'int'>
>>> type(98.6)
<class 'float'>
>>> type("Дароу!")
<class 'str'>
>>> type(True)
<class 'bool'>
```

Шаг 2: Изучите различные логические операторы.

Тип данных Boolean использует операторы, показанные в таблице.

Оператор	Значение
>	Больше чем
<	Меньше чем
==	Равен
!=	Не равен
>=	Больше чем или равен
<=	Меньше чем или равен

В интерпретаторе попробуйте использовать различные логические операторы.

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

Шаг 3: Используйте интерпретатор для создания и использования переменной.

Логическим оператором для определения равенства двух значений является двойной знак равенства (==). Одинарный знак равенства (=) используется для присвоения значения переменной. Затем переменная может быть использована в других командах для получения значения. Например, создайте и используйте следующую переменную в интерактивном интерпретаторе.

```
>>> x=3
>>> x*5
15
>>> "ZSS"*x
'ZSSZSSZSS'
```

Шаг 4: Используйте интерпретатор для конкатенации нескольких строковых переменных.

Конкатенация — это процесс объединения нескольких строк в одну строку. Например, конкатенация **"Нефте"** и **"Юганск"** — это **"НефтеЮганск"**.

Введите следующие четыре переменные, а затем объедините их вместе в операторе **print()** со знаком плюс (+). Обратите внимание, что переменная **space** была определена для использования в качестве пробела между словами.

```
>>> str1="Кафедра Защищенных"
>>> str2="систем"
>>> str3="связи"
>>> space=" "
>>> print(str1+space+str2+space+str3)
Кафедра Защищенных систем связи
```

Чтобы вывести переменные без использования переменной для создания пробела, разделите переменные запятой.

```
>>> print(str1,str2,str3)
Кафедра Защищенных систем связи
```

Шаг 5: Просмотр отливки и печать различных типов данных.

Преобразование между типами данных называется приведением типов. Приведение типов часто необходимо при работе с различными типами данных. Например, конкатенация не работает при объединении данных разных типов.

```
>>> x=3
>>> print("Значение переменной x — " + x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
>>>
```

Используйте функцию **str()** для преобразования целочисленного типа данных в строковый тип данных.

```
>>> print("Значение переменной x – " + str(x))
Значение переменной x – 3
>>> type(x)
<class 'int'>
```

Обратите внимание, что тип данных для переменной x все еще целое число. Чтобы преобразовать тип данных, переназначьте переменную на новый тип данных.

```
>>> x=str(x)
>>> type(x)
<class 'str'>
```

Вам может понадобится отобразить плавающее число с определённой точностью. Для этого можно использовать f-строки и функцию **"{:.2f}".format**.

Примечание: поищите в Интернете, чтобы узнать больше о f-строках и функции format.

```
>>> num = 22/7
>>> f"Значение переменной num – {num}"
'Значение переменной num – 3.142857142857143'
>>> pi = "{:.2f}".format(num)
>>> f"Значение переменной pi – {pi}."
'Значение переменной pi – 3.14.'
>>>
```

Часть 4: Обзор списков и словарей

В этой части вы рассмотрите методы создания и работы со списками и словарями.

Шаг 1: Создайте список и работайте с ним.

При разработке ПО, зачастую переменная списка используется для хранения упорядоченной информации. В некоторых средах программирования списки также называются массивами.

- Создайте список, используя квадратные скобки **[ ]** и заключая каждый элемент списка в кавычки.
- Разделите элементы запятой.
- Используйте команду **type()** для проверки типа данных.
- Используйте команду **len()**, чтобы вернуть количество элементов в списке.
- Вызовите имя переменной списка, чтобы отобразить его содержимое.

В следующем примере показано, как создать переменную списка под названием **hostnames**.

```
>>> hostnames=["R1","R2","R3","S1","S2"]
```

```
>>> type(hostnames)
<class 'list'>
>>> len(hostnames)
5
>>> hostnames
['R1', 'R2', 'R3', 'S1', 'S2']
```

На элемент списка можно ссылаться и манипулировать им, используя его индекс.

- Первый элемент в списке индексируется как ноль, второй - как единица и так далее.
- На последний элемент можно ссылаться с индексом [-1].
- Замените элемент, присвоив индексу новое значение.
- Используйте команду **del**, чтобы удалить элемент из списка.

```
>>> hostnames[0]
'R1'
>>> hostnames[-1]
'S2'
>>> hostnames[0]="RTR1"
>>> hostnames
['RTR1', 'R2', 'R3', 'S1', 'S2']
>>> del hostnames[3]
>>> hostnames
['RTR1', 'R2', 'R3', 'S2']
>>>
```

Шаг 2: Создайте словарь и работайте с ним.

Словари - это неупорядоченные списки объектов. Каждый объект содержит пару ключ/значение.

- Создайте словарь, используя фигурные скобки **{ }**.
- Каждый элемент словаря включает пару ключ и значение.
- Разделяйте ключ и его значение двоеточием.
- Используйте кавычки для ключей и значений, которые являются строками.

Создайте следующий словарь под названием **ipAddress** с тремя парами ключ/значение, чтобы указать значения IP-адресов для трех маршрутизаторов.

```
>>> ipAddress={"R1":"10.1.1.1","R2":"10.2.2.1","R3":"10.3.3.1"}
>>> type(ipAddress)
<class 'dict'>
```

В отличие от списков, на объекты внутри словаря нельзя ссылаться по их индексу. Вместо этого вы ссылаетесь на объект словаря, используя его ключ.

- Ключ заключен в квадратные скобки [ ].
- На ключи, которые являются строками, можно ссылаться, используя одинарные или двойные кавычки.

- Используйте **ключ** в **словаре**, чтобы проверить, существует ли ключ в словаре.
- Добавьте пару ключ/значение, установив новый ключ равным значению.

```
>>> ipAddress
{'R1': '10.1.1.1', 'R2': '10.2.2.1', 'R3': '10.3.3.1'}
>>> ipAddress['R1']
'10.1.1.1'
>>> ipAddress["S1"]="10.1.1.10"
>>> ipAddress
{'R1': '10.1.1.1', 'R2': '10.2.2.1', 'R3': '10.3.3.1', 'S1': '10.1.1.10'}
>>>
```

Значения в паре ключ/значение могут быть любыми другими типами данных, включая списки и словари. Например, если R3 имеет более одного IP-адреса, как вы представите это в словаре **ipAddress**? Создайте список для значения ключа R3.

```
>>> ipAddress["R3"]=["10.3.3.1","10.3.3.2","10.3.3.3"]
>>> ipAddress
{'S1': '10.1.1.10', 'R2': '10.2.2.1', 'R1': '10.1.1.1', 'R3':
['10.3.3.1', '10.3.3.2', '10.3.3.3']}
>>>
```

Часть 5: Обзор функции ввода

В этой части вы рассмотрите, как использовать функцию **input** для хранения и отображения данных, предоставленных пользователем.

Шаг 1: Создайте переменную для хранения пользовательского ввода и последующего отображения значения.

Большинство программ требуют ввода данных либо из базы данных, либо с другого компьютера, либо щелчками мыши, либо с клавиатуры. Для ввода с клавиатуры используйте функцию **input()**, которая включает необязательный параметр для ввода строки запроса. Если вызвать функцию **input**, программа остановится, пока пользователь не введет данные и не нажмет клавишу Enter. Назначьте функцию **input()** переменной, которая запрашивает ввод у пользователя, а затем выведите значение введенной пользователем информации.

```
>>> firstName = input("Как вас зовут? ")
Как вас зовут? Имя_Пользователя
>>> print("Здравствуй, " + firstName +"!")
Здравствуй, Имя_Пользователя!
>>>
```

Шаг 2: Создайте сценарий для сбора личной информации.

Создайте и запустите скрипт для сбора личной информации.

Откройте пустой файл сценария и сохраните его под именем **personal-info.py** в папке **~/labs/devnet-src/python**.

Создайте сценарий, который запрашивает четыре части информации, такие как: имя, фамилия, местоположение и возраст.

Добавьте оператор print, который объединяет всю информацию в одно предложение.

Ваш сценарий должен выполняться без ошибок, как показано в следующем выводе.

```
devasc@labvm:~/labs/devnet-src$ python3 person-info.py
Как вас зовут? Владимир
Какая у вас фамилия? Кан
Где вы находитесь? Нефтеюганск
Сколько вам лет? 5
Здравствуйте, Владимир Кан! Нефтеюганск - место где вы находитесь и вам 5 лет.
devasc@labvm:~/labs/devnet-src$ ^C
```

personal-info.py

```
firstName = input("Как вас зовут? ")
lastName = input("Какая у вас фамилия? ")
location = input("Где вы находитесь? ")
age = input("Сколько вам лет? ")
print("Здравствуйте, " + firstName, lastName + "! location + " - место
где вы находитесь и вам" + age + лет.")
```

Часть 6: Обзор конструкций If, For и While

В этой части мы рассмотрим, как создавать операторы if, а также циклы for и while.

Шаг 1: Используйте конструкцию if/else.

В программировании условные операторы проверяют, является ли что-то истинным, а затем выполняют инструкции на основе полученной оценки. Если оценка ложная, выполняются другие инструкции.

Откройте пустой скрипт и сохраните его под именем **if-vlan.py**. Введите в файл следующий сценарий.

```
nativeVLAN = 1
dataVLAN = 100
if nativeVLAN == dataVLAN:
    print("Native VLAN и data VLAN одинаковые.")
else:
    print("Native VLAN и data VLAN разные.")
```

Примечание: В Python для отступа используйте четыре пробела. Если вы сохраните свой файл в VS Code, отступ в четыре пробела будет автоматическим. Когда вы начинаете оператор else, убедитесь, что вы отступили на левое поле.

Сохраните сценарий и запустите его. Ваш результат должен выглядеть, как показано в следующем примере.

Native VLAN и data VLAN разные.

Измените переменные так, чтобы nativeVLAN и dataVLAN имели одинаковое значение. Сохраните и запустите сценарий снова. Ваш результат должен выглядеть, как показано в следующем примере.
Native VLAN и data VLAN одинаковые.

Шаг 2: Создайте функцию if/elif/else.
Что если нам нужно рассмотреть более двух условных операторов? В этом случае мы можем использовать утверждения **elif** в середине функции **if/elif/else**. Утверждение **elif** оценивается, если утверждение **if** ложно, и перед утверждением **else**. Вы можете иметь столько операторов **elif**, сколько захотите. Однако первый из них будет выполнен, а все остальные не будут проверены. Не будет проверяться и оператор **else**.
Сценарий в следующем примере просит пользователя ввести номер IPv4 ACL, а затем проверяет, является ли этот номер стандартным IPv4 ACL, расширенным IPv4 ACL или ни стандартным, ни расширенным IPv4 ACL.

Создайте этот сценарий для своих файлов. Откройте пустой сценарий и сохраните его под именем **if-acl.py**. Скопируйте сценарий в этот файл.

```
aclNum = int(input("Какой номер ACL IPv4? "))
if aclNum >= 1 and aclNum <= 99:
    print("Это стандартный ACL IPv4.")
elif aclNum >=100 and aclNum <= 199:
    print("Это расширенный ACL IPv4.")
else:
    print("Это ни стандартный и ни расширенный IPv4 ACL.")
```

Примечание: Тип данных для входной функции изменяется со строки по умолчанию на целое число, чтобы оценки **if** и **elif** работали.

Выполните несколько раз, чтобы проверить каждое утверждение.

```
devasc@labvm:~/labs/devnet-src/python$ python3 if-acl.py
Какой номер ACL IPv4? 10
Это стандартный ACL IPv4.
devasc@labvm:~/labs/devnet-src/python$ python3 if-acl.py
Какой номер ACL IPv4? 110
Это расширенный ACL IPv4.
devasc@labvm:~/labs/devnet-src/python$ python3 if-acl.py
Какой номер ACL IPv4? 200
Это ни стандартный и ни расширенный IPv4 ACL.
devasc@labvm:~/labs/devnet-src/python$
```

Шаг 3: Создайте цикл for.
Функция Python **for** используется для цикла или итерации по элементам списка или для выполнения операции над серией значений.

Введите следующее в интерактивном интерпретаторе, чтобы увидеть, как работает цикл for. Имя переменной является произвольным и может быть любым по выбору программиста. Часто программисты сокращают его до буквы **i**.

Примечание: Убедитесь, что вы ввели четыре пробела для отступа функции print() и дважды нажали клавишу Enter для выхода из цикла for.

```
devasc@labvm:~/labs/devnet-src/python$ python3
Python 3.8.2 (default, Mar 13 2020, 10:14:16)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.>>>
devices=["R1","R2","R3","S1","S2"]
>>> for item in devices:
...     print(item)
...
R1
R2
R3
S1
S2
>>>
```

А если вам необходимо перечислить только те элементы, которые начинаются с буквы R? Для этого в цикл **for** можно встроить оператор **if**. Продолжая использовать тот же экземпляр Python, введите в интерактивном интерпретаторе следующее.

Примечание: Убедитесь, что вы ввели четыре пробела для отступа функции **if** и восемь пробелов для отступа функции **print()**. Дважды нажмите клавишу Enter, чтобы выйти и выполнить цикл **for**.

```
>>> for item in devices:
...     if "R" in item:
...         print(item)
...
R1
R2
R3
>>>
```

Вы также можете использовать комбинацию цикла **for** и оператора **if** для создания нового списка. Реализуйте следующий пример, чтобы увидеть, как использовать метод **append()** для создания нового списка **switches**. Обязательно соблюдайте требования по отступам.

```
>>> switches=[]
>>> for item in devices:
...     if "S" in item:
...         switches.append(item)
...
>>> switches
['S1', 'S2']
>>>
```

Шаг 4: Цикл While

Вместо того чтобы выполнять блок кода один раз, как в операторе **if**, вы можете использовать цикл **while**. Цикл **while** продолжает выполнять блок кода до тех пор, пока логическое выражение остается истинным. Это может привести к бесконечному выполнению программы, если вы не убедитесь, что в программе предусмотрено условие для остановки цикла **while**. Циклы **while** не остановятся, пока логическое выражение не будет оценено как false.

Откройте пустой скрипт и сохраните его под именем **while-loop.py**. Используйте цикл **while** для создания следующей программы, которая считает от запрограммированного значения до заданного пользователем числа. Программа делает следующее:

- Запрашивает у пользователя число.
- Приводит введенное строковое значение к целому числу: **x = int(x).**
- Устанавливает переменную для начала счета: **y = 1.**
- Пока **y <= x**, печатает значение y и увеличивает y на 1.

```
x=input("Введите число, до которого следует вести подсчёт: ")
x=int(x)
y=1
while y<=x:
    print(y)
    y=y+1
```

Сохраните и запустите свой скрипт. Вы должны получить результат, подобный следующему.

```
devasc@labvm:~/labs/devnet-src/python$ python3 while-loop.py
Введите число, до которого следует вести подсчёт: 10
1
2
3
4
5
6
7
8
9
10
devasc@labvm:~/labs/devnet-src/python$
```

Вместо использования **while y <= x** мы можем модифицировать цикл **while**, чтобы использовать логическую проверку и ключевое слово **break** для остановки цикла, когда проверка оценивается как false. Измените сценарий **while-loop.py**, как показано ниже:

```
x=input("Введите число, до которого следует вести подсчёт: ")
x=int(x)
y=1
```

```
while True:
    print(y)
    y=y+1
    if y>x:
        break
```

Шаг 5: Используйте цикл while для проверки команды выхода пользователя из программы.

Что если нам необходимо, чтобы программа выполнялась столько раз, сколько захочет пользователь, пока он сам не выйдет из программы? Для этого мы можем встроить программу в цикл **while**, который будет проверять, вводит ли пользователь команду выхода из программы, например **q** или **quit**.

Модифицируйте **while-loop.py**, внеся в него следующие изменения:

- Добавьте еще один цикл **while** в начало, который будет проверять наличие команды **quit**.
- Добавьте функцию **if** в цикл **while** для проверки наличия **q** или **quit**.

Примечание: Убедитесь, что вы включили все необходимые уровни отступов.

```
while True:
    x=input("Введите число, до которого следует вести подсчёт: ")
    if x == 'q' or x == 'quit':
        break

    x=int(x)
    y=1
    while True:
        print(y)
        y=y+1
        if y>x:
            break
```

Сохраните и запустите свой скрипт. Ваш вывод должен быть похож на следующий, в котором пользователь ввел два разных значения перед выходом из программы.

```
devasc@labvm:~/labs/devnet-src/python$ python3 while-loop.py
Введите число, до которого следует вести подсчёт: 3
1
2
3
Введите число, до которого следует вести подсчёт: 5
1
2
3
4
5
Введите число, до которого следует вести подсчёт: quit
devasc@labvm:~/labs/devnet-src/python$
```

Часть 7: Обзор методов доступа к файлам

В этой части вы рассмотрите методы доступа к файлу, чтения и манипулирования им.

Шаг 1: Создайте программу, которая читает внешний файл.

Помимо ввода данных пользователем, вы можете получить доступ к базе данных, другой компьютерной программе или файлу, чтобы обеспечить ввод данных в вашу программу. Для доступа к файлу можно использовать функцию **open()**, используя следующий синтаксис:

```
open(имя_файла, [режим])
```

Параметр *имя_файла* — это название открываемого файла. Если файл находится в каталоге, отличном от каталога вашего скрипта, вам также потребуется указать информацию о пути. Для наших целей нас интересуют только три параметра режима:

- **r** — чтение файла (режим по умолчанию, если параметр *режим* опущен).
- **w** — записать файл, заменив его содержимое.
- **a** — добавить в файл.

Откройте пустой сценарий и сохраните его под именем **file-access.py**.

Создайте следующую программу для чтения и печати содержимого файла devices.txt, который находится в том же каталоге, что и ваша программа. После печати содержимого файла используйте функцию **close()**, чтобы удалить его из памяти компьютера.

```
file=open("devices.txt","r")
for item in file:
    print(item)
file.close()
```

Примечание: Содержимое файла устанавливается в переменную с именем **file**. Однако эта переменная может называться как угодно по выбору программиста.

Сохраните и запустите свою программу. Вы должны получить результат, похожий на следующий.

```
devasc@labvm:~/labs/devnet-src/python$ python3 file-access.py
Cisco 819 Router

Cisco 881 Router

Cisco 888 Router

Cisco 1100 Router

Cisco 4321 Router
```

```
Cisco 4331 Router

Cisco 4351 Router

Cisco 2960 Catalyst Switch

Cisco 3850 Catalyst Switch

Cisco 7700 Nexus Switch

Cisco Meraki MS220-8 Cloud Managed Switch

Cisco Meraki MX64W Security Appliance

Cisco Meraki MX84 Security Appliance

Cisco Meraki MC74 VoIP Phone

Cisco 3860 Catalyst Switch
devasc@labvm:~/labs/devnet-src/python$
```

Шаг 2: Удалите пустые строки из вывода.

Вы могли заметить, что Python добавил пустую строку после каждой записи. Мы можем удалить эту пустую строку с помощью метода **strip()**.

Отредактируйте программу **file-access.py**, включив в нее метод **strip()**.

```
file=open("devices.txt","r")
for item in file:
   item=item.strip()
   print(item)
file.close()
```

Сохраните и запустите свою программу. Вы должны получить результат, подобный следующему.

```
devasc@labvm:~/labs/devnet-src/python$ python3 file-access.py
Cisco 819 Router
Cisco 881 Router
Cisco 888 Router
Cisco 1100 Router
Cisco 4321 Router
Cisco 4331 Router
Cisco 4351 Router
Cisco 2960 Catalyst Switch
Cisco 3850 Catalyst Switch
Cisco 7700 Nexus Switch
Cisco Meraki MS220-8 Cloud Managed Switch
Cisco Meraki MX64W Security Appliance
Cisco Meraki MX84 Security Appliance
Cisco Meraki MC74 VoIP Phone
Cisco 3860 Catalyst Switch
```

```
devasc@labvm:~/labs/devnet-src/python$
```

Шаг 3: Скопируйте содержимое файла в переменную списка.

В большинстве случаев, когда программисты обращаются к внешнему ресурсу, такому как база данных или файл, они хотят скопировать его содержимое в локальную переменную, на которую можно ссылаться и манипулировать ею, не затрагивая исходный ресурс.

Файл devices.txt представляет собой список устройств Cisco, который можно легко скопировать в список Python, выполнив следующие действия:

- Создайте пустой список.
- Используйте параметр **append** для копирования содержимого файла в новый список.
- Распечатайте список.

Измените файл **file-access.py**, как показано ниже

```
devices=[]
file=open("devices.txt","r")
for item in file:
   item=item.strip()
   devices.append(item)
file.close()
print(devices)
```

Сохраните и запустите свою программу. Вы должны получить результат, подобный следующему.

```
devasc@labvm:~/labs/devnet-src/python$ python3 file-access.py
['Cisco 819 Router', 'Cisco 881 Router', 'Cisco 888 Router', 'Cisco 1100 Router', 'Cisco 4321 Router', 'Cisco 4331 Router', 'Cisco 4351 Router', 'Cisco 2960 Catalyst Switch', 'Cisco 3850 Catalyst Switch', 'Cisco 7700 Nexus Switch', 'Cisco Meraki MS220-8 Cloud Managed Switch', 'Cisco Meraki MX64W Security Appliance', 'Cisco Meraki MX84 Security Appliance', 'Cisco Meraki MC74 VoIP Phone', 'Cisco 3860 Catalyst Switch']
devasc@labvm:~/labs/devnet-src/python$
```

Шаг 4: Задача: Создайте сценарий, позволяющий пользователю добавлять устройства.

Что если вам нужно добавить несколько устройств в файл **devices.txt**? Вы можете создать программу, которая будет открывать файл в режиме добавления, а затем попросит пользователя указать имя новых устройств. Для создания сценария выполните следующие шаги:

1) Откройте новый файл и сохраните его под именем **file-access-input.py**.

2) Для функции **open()** используйте режим a, который позволит вам добавить элемент в файл devices.txt.

3) Внутри цикла **while True**: встройте команду функции **input()**, которая запрашивает у пользователя новое устройство.

4) Установите значение пользовательского ввода в переменную с именем **newItem**.

5) Используйте оператор if, который прерывает цикл, если пользователь набирает exit и печатает сообщение "All done!".

6) Используйте команду **file.write(newItem + "\n")** для добавления нового устройства, введенного пользователем.

7) Закройте файл, чтобы освободить его из памяти компьютера.

Запускайте и устраняйте неисправности вашего сценария до тех пор, пока не получите результат, подобный следующему.

```
devasc@labvm:~/labs/devnet-src/python$ python3 file-access-input.py
Введите имя устройства: Cisco 1941 Router
Введите имя устройства: Cisco 2950 Catalyst Switch
Введите имя устройства: exit
Всё готово!
devasc@labvm:~/labs/devnet-src/python$
```

```
file = open("devices.txt", "a")
while True:
    newItem = input("Введите имя устройства: ")
    if newItem == "exit":
        print("Всё готово!")
        break
    file.write(newItem + "\n")
file.close()
```

Содержание отчета:

1. Титульный лист
2. Скриншоты промежуточных действий по каждой из Частей.

Лабораторная работа №3.1.12

Исследование инструментов разработки на языке Python

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC

Часть 2: Обзор установки Python

Часть 3: Виртуальные среды PIP и Python

Часть 4: Совместное использование виртуальной среды

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

Порядок выполнения работы

Часть 1: Запуск виртуальной машины DEVASC

Если вы еще не завершили **лабораторную работу - Установка лабораторной среды виртуальной машины**, сделайте это сейчас. Если вы уже завершили эту лабораторную работу, запустите виртуальную машину **DEVASC**.

Часть 2: Обзор установки Python

Эти команды предоставляют вам базовые методы для получения дополнительной информации о локальной среде Python.

В виртуальной машине DEVASC можно проверить версию уже установленного Python с помощью команды python3 -V.

```
devasc@labvm:~$ python3 -V
Python 3.8.2
devasc@labvm:~$
```

Чтобы увидеть каталог для локальной среды Python, используйте команду which python3.

```
devasc@labvm:~$ which python3
/usr/bin/python3
devasc@labvm:~$
```

Часть 3: PIP и виртуальные среды Python

PIP расшифровывается как Pip Installs Packages. Многие люди впервые узнают о PIP и начинают использовать команды pip3 install на общесистемной установке Python. Когда вы выполняете команду pip3 install в своей системе, вы можете внести конкурирующие зависимости в свою системную установку, которые могут понадобиться или не понадобиться для всех проектов Python. Поэтому лучшей практикой является включение виртуальной среды Python. Затем установите только те пакеты, которые необходимы для проекта в этой виртуальной среде. Таким образом, вы будете точно знать, какие пакеты установлены в тех или иных условиях. Вы сможете легко переключать зависимости пакетов при переходе на новую виртуальную среду, не допуская сбоев и проблем из-за конкурирующих версий программного обеспечения.

Чтобы установить виртуальную среду Python, используйте инструмент venv в Python 3, а затем активируйте виртуальную среду, как показано в следующих шагах.

Шаг 1: Создайте виртуальную среду Python 3.

Внутри виртуальной машины DEVASC измените каталог labs/devnet-src/python.

```
devasc@labvm:~$ cd labs/devnet-src/python/
devasc@labvm:~/labs/devnet-src/python$
```

Введите следующую команду, чтобы использовать инструмент venv для создания виртуальной среды Python 3 с именем devfun. Переключатель -m указывает Python запустить модуль venv. Имя выбирается программистом.

```
devasc@labvm:~/labs/devnet-src/python$ python3 -m venv devfun
devasc@labvm:~/labs/devnet-src/python$
```

Шаг 2: Активируйте и протестируйте виртуальную среду Python 3.

Активируйте виртуальную среду. Подсказка изменится, чтобы указать имя среды, в которой вы сейчас работаете, в данном примере это devfun. Теперь, когда вы используете форму команды pip3 install здесь, система будет устанавливать пакеты только для активной виртуальной среды.

```
devasc@labvm:~/labs/devnet-src/python$ source devfun/bin/activate
(devfun) devasc@labvm:~/labs/devnet-src/python$
```

Выполните команду pip3 freeze, чтобы убедиться в отсутствии дополнительных пакетов Python, установленных в настоящее время в среде devfun.

```
(devfun) devasc@labvm:~/labs/devnet-src/python$ pip3 freeze
(devfun) devasc@labvm:~/labs/devnet-src/python$
```

Теперь установите пакет Python requests в среду devfun.

```
(devfun) devasc@labvm:~/labs/devnet-src/python$ pip3 install requests
Collecting requests
  Downloading requests-2.23.0-py2.py3-none-any.whl (58 kB)
     |████████████████████████████████| 58 kB 290 kB/s
Collecting urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1
  Downloading urllib3-1.25.9-py2.py3-none-any.whl (126 kB)
     |████████████████████████████████| 126 kB 1.7 MB/s
Collecting idna<3,>=2.5
  Downloading idna-2.9-py2.py3-none-any.whl (58 kB)
     |████████████████████████████████| 58 kB 18.3 MB/s
Collecting certifi>=2017.4.17
  Downloading certifi-2020.4.5.1-py2.py3-none-any.whl (157 kB)
     |████████████████████████████████| 157 kB 19.8 MB/s
Collecting chardet<4,>=3.0.2
  Downloading chardet-3.0.4-py2.py3-none-any.whl (133 kB)
     |████████████████████████████████| 133 kB 59.2 MB/s
Installing collected packages: urllib3, idna, certifi, chardet, requests
Successfully installed certifi-2020.4.5.1 chardet-3.0.4 idna-2.9 requests-2.23.0
urllib3-1.25.9
(devfun) devasc@labvm:~/labs/devnet-src/python$
```

Повторно введите команду pip3 freeze, чтобы увидеть пакеты, установленные теперь в среде devfun.

```
(devfun) devasc@labvm:~/labs/devnet-src/python$ pip3 freeze
certifi==2020.4.5.1
chardet==3.0.4
idna==2.10
requests==2.24.0
urllib3==1.25.9
(devfun) devasc@labvm:~/labs/devnet-src/python$
```

Чтобы деактивировать виртуальную среду и вернуться к своей системе, введите команду deactivate.

```
(devfun) devasc@labvm:~/labs/devnet-src/python$ deactivate
devasc@labvm:~/labs/devnet-src/python$
```

Шаг 3: Проверьте текущие пакеты, установленные в системной среде.

Введите команду system wide python3 -m pip freeze, чтобы посмотреть, какие пакеты установлены в системной среде.

Примечание: Поскольку Python 3 вызывается с помощью следующей команды, вы используете pip вместо pip3.

```
devasc@labvm:~/labs/devnet-src/python$ python3 -m pip freeze
ansible==2.9.4
apache-libcloud==2.8.0
appdirs==1.4.3
argcomplete==1.8.1
astroid==2.3.3
bcrypt==3.1.7
blinker==1.4
certifi==2019.11.28
<вывод опущен>
xmltodict==0.12.0
zipp==1.0.0
devasc@labvm:~/labs/devnet-src/python$
```

Если вы хотите быстро найти версию установленного пакета, передайте вывод в команду grep. Введите следующее, чтобы узнать версию установленного пакета requests.

```
devasc@labvm:~/labs/devnet-src/python$ python3 -m pip freeze | grep
requests
requests==2.22.0
requests-kerberos==0.12.0
requests-ntlm==1.1.0
requests-toolbelt==0.9.1
requests-unixsocket==0.2.0
devasc@labvm:~/labs/devnet-src/python$
```

Часть 4: Совместное использование виртуальной среды

Вывод команды pip3 freeze имеет определенный формат не просто так. Вы можете использовать все перечисленные зависимости, чтобы другие люди, которые хотят работать над тем же проектом, что и вы, могли получить такое же окружение, как у вас.

Разработчик может создать файл требований, например, requirements.txt, с помощью команды pip3 freeze > requirements.txt. Затем другой разработчик может из другой активированной виртуальной среды использовать команду pip3 install -r requirements.txt для установки пакетов, необходимых проекту.

Повторно активируйте виртуальную среду devfun.

```
devasc@labvm:~/labs/devnet-src/python$ source devfun/bin/activate
```

```
(devfun) devasc@labvm:
```

Отправьте вывод команды pip3 freeze в текстовый файл requirements.txt.

```
(devfun) devasc@labvm:~/labs/devnet-src/python$ pip3 freeze >
requirements.txt
```

Деактивируйте виртуальную среду devfun. С помощью команды ls можно увидеть, что файл requirements.txt находится в каталоге /python.

```
(devfun) devasc@labvm:~/labs/devnet-src/python$ deactivate
devasc@labvm:~/labs/devnet-src/python$ ls
devfun        file-access-input.py  if-acl.py       requirements.txt
devices.txt   file-access.py        if-vlan.py      while-loop.py
devnew        hello-world.py        person-info.py
```

Создайте и активируйте новую виртуальную среду Python под названием devnew.

```
devasc@labvm:~/labs/devnet-src/python$ python3 -m venv devnew
devasc@labvm:~/labs/devnet-src/python$ source devnew/bin/activate
(devnew) devasc@labvm:~/labs/devnet-src/python$
```

Используйте команду pip3 install -r requirements.txt для установки тех же пакетов, которые установлены в виртуальной среде devfun.

```
(devnew) devasc@labvm:~/labs/devnet-src/python$ pip3 install -r
requirements.txt
Collecting certifi==2020.4.5.1
  Using cached certifi-2020.4.5.1-py2.py3-none-any.whl (157 kB)
Collecting chardet==3.0.4
  Using cached chardet-3.0.4-py2.py3-none-any.whl (133 kB)
Collecting idna==2.9
  Using cached idna-2.9-py2.py3-none-any.whl (58 kB)
Requirement already satisfied: pkg-resources==0.0.0 in
./devnew/lib/python3.8/site-packages (from -r requirements.txt (line 4)) (0.0.0)
Collecting requests==2.23.0
  Using cached requests-2.23.0-py2.py3-none-any.whl (58 kB)
Collecting urllib3==1.25.9
  Using cached urllib3-1.25.9-py2.py3-none-any.whl (126 kB)
Installing collected packages: certifi, chardet, idna, urllib3, requests
Successfully installed certifi-2020.4.5.1 chardet-3.0.4 idna-2.9 requests-2.23.0
urllib3-1.25.9
(devnew) devasc@labvm:~/labs/devnet-src/python$
```

При вводе pip3 freeze в среде devnew вы должны увидеть следующий вывод.

```
(devnew) devasc@labvm:~/labs/devnet-src/python$ pip3 freeze
certifi==2020.4.5.1
chardet==3.0.4
idna==2.9
pkg-resources==0.0.0
requests==2.23.0
urllib3==1.25.9
```

Деактивируйте виртуальную среду devnew.

```
(devnew) devasc@labvm:~/labs/devnet-src/python$ deactivate
devasc@labvm:~/labs/devnet-src/python$
```

Содержание отчета:

1. Титульный лист

2. Скриншоты промежуточных действий по каждой из Частей.

Лабораторная работа №3.3.11
Контроль версий программного обеспечения с помощью Git

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC

Часть 2: Инициализация Git

Часть 3: Постановка и фиксация файла в репозитории Git

Часть 4: Управление файлом и отслеживание изменений

Часть 5: Ветви и слияние

Часть 6: Обработка конфликтов при слиянии

Часть 7: Интеграция Git с GitHub

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

Порядок выполнения работы

Часть 1: Запуск виртуальной машины DEVASC

Если вы еще не завершили **лабораторную работу - Установка лабораторной среды виртуальной машины**, сделайте это сейчас. Если вы уже завершили эту лабораторную работу, запустите виртуальную машину **DEVASC**.

Часть 2: Инициализация Git

В этой части вы инициализируете репозиторий Git.

Шаг 1: Откройте терминал в DEVASC-LABVM.

Дважды щелкните значок эмулятора терминала на рабочем столе.
Примечание: Автор будет выполнять все операции в терминале Visual Studio Code.

Шаг 2: Инициализируем репозиторий Git.

Используйте команду ls для отображения списка текущего каталога. Помните, что команды чувствительны к регистру.

```
devasc@labvm:~$ ls
Desktop    Downloads  Music     Public  Templates
Documents  labs       Pictures  snap    Videos
devasc@labvm:~$
```

```
devasc@labvm:~$ ls
Desktop  devfun  Documents  Downloads  labs  snap
devasc@labvm:~$
```

Далее настройте информацию о пользователе, которая будет использоваться для этого локального хранилища. Это позволит связать вашу информацию с работой, которую вы вносите в локальное хранилище. Используйте свое имя вместо "Имя Пользователяг" для имени в кавычках " ". Используйте @example.com для адреса электронной почты.

Примечание: На данном этапе эти параметры могут быть любыми. Однако, когда вы сбросите эти глобальные значения в Части 7, вы будете использовать имя пользователя собственной учетной записи GitHub. Если хотите, вы можете использовать свое имя пользователя GitHub сейчас.

```
devasc@labvm:~$ git config --global user.name "SampleUser"
devasc@labvm:~$ git config --global user.email sample@example.com
```

В любой момент вы можете просмотреть эти настройки с помощью команды git config --list.

```
devasc@labvm:~$ git config --list
user.name=SampleUser
user.email=sample@example.com
devasc@labvm:~$
```

С помощью команды cd перейдите в папку devnet-src:

```
devasc@labvm:~$ cd labs/devnet-src/
devasc@labvm:~/labs/devnet-src$
```

Создайте каталог git-intro и перейдите в него:

```
devasc@labvm:~/labs/devnet-src$ mkdir git-intro
devasc@labvm:~/labs/devnet-src$ cd git-intro
devasc@labvm:~/labs/devnet-src/git-intro$
```

```
devasc@labvm:~/labs/devnet-src$ mkdir git-intro
devasc@labvm:~/labs/devnet-src$ cd git-intro/
devasc@labvm:~/labs/devnet-src/git-intro$
```

Используйте команду git init для инициализации текущего каталога (git-intro) в качестве репозитория Git. Появившееся сообщение указывает на то, что вы создали локальный репозиторий внутри вашего проекта, содержащийся в

скрытом каталоге .git. Именно здесь находится вся история изменений. Вы можете увидеть ее с помощью команды ls -a.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git init
Initialized empty Git repository in /home/devasc/labs/devnet-src/git-intro/.git/
devasc@labvm:~/labs/devnet-src/git-intro$ ls -a
.  ..  .git
devasc@labvm:~/labs/devnet-src/git-intro$
```

```
devasc@labvm:~/labs/devnet-src/git-intro$ git init
Initialized empty Git repository in /home/devasc/labs/devnet-src/git-intro/.git/
devasc@labvm:~/labs/devnet-src/git-intro$
```

В процессе работы над проектом вы захотите проверить, какие файлы изменились. Это полезно, когда вы фиксируете файлы в репозитории, но не хотите коммитить их все. Команда git status отображает измененные файлы в рабочем каталоге, которые помещены в очередь для следующих коммитов.

Это сообщение сообщает вам:

- Что вы находитесь на ветке master. (Ветви обсуждаются позже в этой лабораторной работе).
- Сообщение о коммите - Initial commit.
- Для коммита ничего не изменилось.

Вы увидите, что статус вашего репозитория изменится, как только вы добавите файлы и начнете вносить изменения.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
devasc@labvm:~/labs/devnet-src/git-intro$
```

```
devasc@labvm:~/labs/devnet-src/git-intro$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
devasc@labvm:~/labs/devnet-src/git-intro$
```

Часть 3: Staging и Commiting файла в хранилище

В этой части вы создадите файл, застэджите этот файл и закоммитите его в репозитории Git.

Шаг 1: Создайте файл.

Репозиторий git-intro создан, но пуст. Используя команду echo, создайте файл DEVASC.txt с информацией, заключенной в кавычки.

```
devasc@labvm:~/labs/devnet-src/git-intro$ echo "Я на пути к успешной сдаче экзамена Cisco DEVASC" > DEVASC.txt
devasc@labvm:~/labs/devnet-src/git-intro$
```

С помощью команды ls -la убедитесь, что файл, а также каталог .git находятся в каталоге git intro. Затем с помощью команды cat отобразите содержимое файла DEVASC.txt.

```
devasc@labvm:~/labs/devnet-src/git-intro$ ls -la
total 16
drwxrwxr-x 3 devasc devasc 4096 Apr 17 20:38 .
drwxrwxr-x 5 devasc devasc 4096 Apr 17 19:50 ..
-rw-rw-r-- 1 devasc devasc   48 Apr 17 20:38 DEVASC.txt
drwxrwxr-x 7 devasc devasc 4096 Apr 17 19:57 .git
evasc@labvm:~/src/git-intro$ cat DEVASC.txt
Я на пути к успешной сдаче экзамена Cisco DEVASC
devasc@labvm:~/labs/devnet-src/git-intro$
```

```
devasc@labvm:~/labs/devnet-src/git-intro$ ls -la
total 16
drwxrwxr-x  3 devasc devasc 4096 апр 28 22:01 .
drwxr-xr-x 15 devasc devasc 4096 апр 28 21:09 ..
-rw-rw-r--  1 devasc devasc   78 апр 28 22:02 DEVASC.txt
drwxrwxr-x  7 devasc devasc 4096 апр 28 21:23 .git
devasc@labvm:~/labs/devnet-src/git-intro$
```

Шаг 2: Изучите статус репозитория.

Изучите состояние репозитория с помощью git status. Обратите внимание, что Git нашел новый файл в каталоге и знает, что он не отслеживается.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
      DEVASC.txt

nothing added to commit but untracked files present (use "git add" to track)
devasc@labvm:~/labs/devnet-src/git-intro$
```

```
devasc@labvm:~/labs/devnet-src/git-intro$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        DEVASC.txt

nothing added to commit but untracked files present (use "git add" to track)
devasc@labvm:~/labs/devnet-src/git-intro$
```

Шаг 3: Постановка файла.

Далее используйте команду git add для "постановки" файла DEVASC.txt. Постановка - это промежуточный этап перед фиксацией файла в хранилище с помощью команды git commit. Эта команда создает снимок содержимого файла на момент ввода этой команды. Любые изменения в файле требуют еще одной команды git add перед фиксацией файла.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git add DEVASC.txt
```

Снова используя команду git status, обратите внимание на поэтапные изменения, отображаемые как "новый файл: DEVASC.txt".

```
devasc@labvm:~/labs/devnet-src/git-intro$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
      new file:   DEVASC.txt

devasc@labvm:~/labs/devnet-src/git-intro$
```

```
devasc@labvm:~/labs/devnet-src/git-intro$ git add DEVASC.txt
devasc@labvm:~/labs/devnet-src/git-intro$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   DEVASC.txt

devasc@labvm:~/labs/devnet-src/git-intro$
```

Шаг 4: Коммит файла.

Теперь, когда вы поставили свои изменения, вам нужно зафиксировать их, чтобы сообщить Git'у, что вы хотите начать отслеживать эти изменения. Зафиксируйте ваш этапный контент как новый снимок фиксации с помощью команды git commit. Переключатель -m message позволяет добавить

сообщение, объясняющее сделанные изменения. Обратите внимание на комбинацию цифр и букв, выделенную в выводе. Это и есть идентификатор фиксации. Каждый коммит идентифицируется уникальным хэшем SHA1. ID фиксации - это первые 7 символов полного хэша фиксации. Ваш ID фиксации будет отличаться от отображаемого.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git commit -m "Коммичу DEVASC.txt для начала отслеживания изменений"
[master (root-commit) 2008010] Коммичу DEVASC.txt для начала отслеживания изменений 1 file changed, 1 insertion(+)
 create mode 100644 DEVASC.txt
devasc@labvm:~/labs/devnet-src/git-intro$
```

```
devasc@labvm:~/labs/devnet-src/git-intro$ git commit -m "Коммичу DEVASC.txt для начала отслежив
ания изменений"
[master (root-commit) 2008010] Коммичу DEVASC.txt для начала отслеживания изменений
 1 file changed, 1 insertion(+)
 create mode 100644 DEVASC.txt
devasc@labvm:~/labs/devnet-src/git-intro$
```

Шаг 5: Просмотр истории фиксации.

Используйте команду git log, чтобы показать все коммиты в истории текущей ветви. По умолчанию все коммиты делаются в ветке master. (Ветви будут обсуждаться позже.) Первая строка - это хэш фиксации с идентификатором фиксации в виде первых 7 символов. Файл зафиксирован в ветке master. Далее следует ваше имя и адрес электронной почты, дата фиксации и сообщение, которое вы приложили к фиксации.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git log
commit 2008010d883515b55a401ac1820e4335f848400a (HEAD -> master)
Author: Sample User <sample@example.com>
Date:   Sat Apr 18 18:03:28 2020 +0000

    Коммичу DEVASC.txt для начала отслеживания изменений
devasc@labvm:~/labs/devnet-src/git-intro$
```

Часть 4: Изменение файла и отслеживание изменений

В этой части вы измените файл, застэйжите файл, закоммитите его и проверите изменения в хранилище.

Шаг 1: Измените файл.

Внесите изменения в файл DEVASC.txt с помощью команды echo. Обязательно используйте ">>" для добавления существующего файла. Символ ">" перезапишет существующий файл. Используйте команду cat для просмотра измененного файла.

```
devasc@labvm:~/labs/devnet-src/git-intro$ echo "Я начинаю понимать Git!" >> DEVASC.txt
```

Используйте команду cat для просмотра измененного файла.

```
devasc@labvm:~/labs/devnet-src/git-intro$ cat DEVASC.txt
Я на пути к успешной сдаче экзамена Cisco DEVASC
Я начинаю понимать Git!
devasc@labvm:~/labs/devnet-src/git-intro$
```

Шаг 2: Проверьте изменения в хранилище.

Проверьте изменения в хранилище с помощью команды git status.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
      modified:   DEVASC.txt

no changes added to commit (use "git add" and/or "git commit -a")
devasc@labvm:~/labs/devnet-src/git-intro$
```

```
devasc@labvm:~/labs/devnet-src/git-intro$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   DEVASC.txt

no changes added to commit (use "git add" and/or "git commit -a")
devasc@labvm:~/labs/devnet-src/git-intro$
```

Шаг 3: Застэйджите измененный файл.

Измененный файл нужно будет снова застэйджить, прежде чем его можно будет закоммитить с помощью команды git add.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git add DEVASC.txt
```

Шаг 4: Закоммитье застэйджите файл.

Зафиксируйте поэтапный файл с помощью команды git commit. Обратите внимание на новый идентификатор фиксации.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git commit -m "Добавлена
дополнительная строка в файл"
[master 018a3f3] Добавлена дополнительная строка в файл
 1 file changed, 1 insertion(+)
devasc@labvm:~/labs/devnet-src/git-intro$
```

```
devasc@labvm:~/labs/devnet-src/git-intro$ git add DEVASC.txt
devasc@labvm:~/labs/devnet-src/git-intro$ git commit -m "Добавлена дополнительная строка в файл
"
[master 018a3f3] Добавлена дополнительная строка в файл
 1 file changed, 1 insertion(+)
devasc@labvm:~/labs/devnet-src/git-intro$
```

Шаг 5: Проверьте изменения в хранилище.

Снова используйте команду git log, чтобы показать все фиксации. Обратите внимание, что журнал содержит исходную запись о фиксации вместе с записью о фиксации, которую вы только что выполнили. Последняя фиксация показана первой. В выводе выделены ID фиксации (первые 7 символов SHA1-хэша), дата/время фиксации и сообщение о фиксации для каждой записи.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git log
commit 018a3f3b6afa53f7b465d4c7009b2e2975212558 (HEAD -> master)
Author: Sample User <sample@example.com>
Date:   Sat Apr 18 19:17:50 2020 +0000
```

```
    Добавлена дополнительная строка в файл

commit 2008010d883515b55a401ac1820e4335f848400a
Author: Sample User <sample@example.com>
Date:   Sat Apr 18 18:03:28 2020 +0000

    Коммичу DEVASC.txt для начала отслеживания изменений
devasc@labvm:~/labs/devnet-src/git-intro$
```

Если у вас есть несколько записей в журнале, вы можете сравнить два коммита с помощью команды git diff, добавив ID оригинального коммита первым, а ID последнего коммита - вторым: git diff <коммит-ID оригинальный> <коммит-ID последний>. Вам нужно будет использовать свои идентификаторы коммитов. Знак "+" в конце, за которым следует текст, указывает на содержимое, которое было добавлено в файл.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git diff 018a3f3 2008010
```

```
devasc@labvm:~/labs/devnet-src/git-intro$ git diff 018a3f3 2008010
diff --git a/DEVASC.txt b/DEVASC.txt
index 18a86c6..28095c2 100644
--- a/DEVASC.txt
+++ b/DEVASC.txt
@@ -1,2 +1 @@
 Я на пути к успешной сдаче экзамена Cisco DEVASC
-Я начинаю понимать Git!
devasc@labvm:~/labs/devnet-src/git-intro$
```

Часть 5: Ветви и слияние

При создании репозитория файлы автоматически помещаются в ветвь, называемую master. По возможности рекомендуется использовать ветви, а не обновлять главную ветвь напрямую. Ветвление используется для того, чтобы вы могли вносить изменения в другую область, не затрагивая основную ветвь. Это делается для того, чтобы предотвратить случайные обновления, которые могут переписать существующий код.

В этой части вы создадите новую ветвь, проверите ветвь, внесете изменения в ветвь, выполните этап и фиксацию ветви, объедините изменения ветви с основной ветвью, а затем удалите ветвь.

Шаг 1: Создайте новую ветвь

Создайте новую ветку под названием feature с помощью команды git branch <имя ветки>.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git branch feature
```

Шаг 2: Проверьте текущую ветку

Используйте команду git branch без указания имени ветви, чтобы отобразить все ветви для этого хранилища. Символ "*" рядом с веткой master указывает на то, что это текущая ветка - ветка, которая в данный момент "проверяется".

```
devasc@labvm:~/labs/devnet-src/git-intro$ git branch
  feature
* master
devasc@labvm:~/labs/devnet-src/git-intro$
```

Шаг 3: Проверьте новую ветку

Используйте команду git checkout <имя ветки> для перехода к ветке с функциями.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git checkout feature
```

Шаг 4: Проверка текущей ветки

Убедитесь, что вы переключились на функциональную ветвь с помощью команды git branch. Обратите внимание на "*" рядом с веткой feature. Теперь это рабочая ветвь.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git branch
* feature
  master
devasc@labvm:~/labs/devnet-src/git-intro$
```

Добавьте новую строку текста в файл DEVASC.txt, снова используя команду echo со знаками ">>".

```
devasc@labvm:~/labs/devnet-src/git-intro$ echo "Этот текст был добавлен
первоначально, во время работы над функцией" >> DEVASC.txt
```

Проверьте, что строка была добавлена в файл с помощью команды cat.

```
devasc@labvm:~/labs/devnet-src/git-intro$ cat DEVASC.txt
```

```
devasc@labvm:~/labs/devnet-src/git-intro$ cat DEVASC.txt
Я на пути к успешной сдаче экзамена Cisco DEVASC
Я начинаю понимать Git!
Этот текст был добавлен первоначально, во время работы над функцией
devasc@labvm:~/labs/devnet-src/git-intro$
```

Шаг 5: Поместите измененный файл в функциональную ветвь.

Поместите обновленный файл в текущую функциональную ветвь.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git add DEVASC.txt
```

Используйте команду git status и заметите, что измененный файл DEVASC.txt помещен в ветку feature.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git status
On branch feature
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
      modified:   DEVASC.txt

devasc@labvm:~/labs/devnet-src/git-intro$
```

Шаг 6: Закоммитьте стэйджовый файл в фиче ветке.

Зафиксируйте поэтапный файл с помощью команды git commit. Обратите внимание на новый идентификатор фиксации и ваше сообщение.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git commit -m "Добавлена третья строка в feature ветке"
[feature bcbc441] Добавлена третья строка в feature ветке
 1 file changed, 1 insertion(+)
devasc@labvm:~/labs/devnet-src/git-intro$
```

```
devasc@labvm:~/labs/devnet-src/git-intro$ git commit -m "Добавлена третья строка в feature ветке"
[feature bcbc441] Добавлена третья строка в feature ветке
 1 file changed, 1 insertion(+)
devasc@labvm:~/labs/devnet-src/git-intro$
```

Используйте команду git log, чтобы показать все коммиты, включая коммит, который вы только что сделали в ветви feature. Предыдущая фиксация была сделана в ветке master.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git log
```

```
devasc@labvm:~/labs/devnet-src/git-intro$ git log
commit bcbc441c433d53d8a95f6b3350fa5f1f53e48460 (HEAD -> feature)
Author: Denis Zakharov <misterzurg@gmail.com>
Date:   Thu Apr 28 23:10:18 2022 +0000

    Добавлена третья строка в feature ветке

commit 018a3f3b6afa53f7b465d4c7009b2e2975212558 (master)
Author: Denis Zakharov <misterzurg@gmail.com>
Date:   Thu Apr 28 22:32:30 2022 +0000

    Добавлена дополнительная строка в файл

commit 2008010d883515b55a401ac1820e4335f848400a
Author: Denis Zakharov <misterzurg@gmail.com>
Date:   Thu Apr 28 22:17:24 2022 +0000

    Коммичу DEVASC.txt для начала отслеживания изменений
devasc@labvm:~/labs/devnet-src/git-intro$
```

Шаг 7: Проверьте master-ветку.

Переключитесь на master-ветку с помощью команды git checkout master и проверьте текущую рабочую ветку с помощью команды git branch.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git checkout master
Switched to branch 'master'
devasc@labvm:~/labs/devnet-src/git-intro$ git branch
  feature
* master
```

```
devasc@labvm:~/labs/devnet-src/git-intro$
```

Шаг 8: Объедините содержимое файлов из функциональной ветви в основную ветвь.

Ветви часто используются при внедрении новых функций или исправлений. Они могут быть представлены на рассмотрение членам команды, а затем, после проверки, могут быть перенесены в основную кодовую базу - главную ветвь.

Слейте содержимое (известное как история) из ветки функций в главную ветку с помощью команды git merge <имя ветки>. Имя ветви - это ветвь, из которой истории перетягиваются в текущую ветвь. Вывод показывает, что один файл был изменен и вставлена одна строка.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git merge feature
Updating 018a3f3..bcbc441
Fast-forward
 DEVASC.txt | 1 +
 1 file changed, 1 insertion(+)
devasc@labvm:~/labs/devnet-src/git-intro$
```

Проверьте добавление содержимого в файл DEVASC.txt в основной ветви с помощью команды cat.

```
devasc@labvm:~/labs/devnet-src/git-intro$ cat DEVASC.txt
```

```
devasc@labvm:~/labs/devnet-src/git-intro$ cat DEVASC.txt
Я на пути к успешной сдаче экзамена Cisco DEVASC
Я начинаю понимать Git!
Этот текст был добавлен первоначально, во время работы над функцией
devasc@labvm:~/labs/devnet-src/git-intro$
```

Шаг 9: Удаление ветки.

Убедитесь, что фича ветка всё ещё доступна, используя команду git branch.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git branch
  feature
* master
devasc@labvm:~/labs/devnet-src/git-intro$
```

Удалите функциональную ветвь с помощью команды git branch -d <имя-ветки>. .

```
devasc@labvm:~/labs/devnet-src/git-intro$ git branch -d feature
Deleted branch feature (was bcbc441).
devasc@labvm:~/labs/devnet-src/git-intro$
```

Убедитесь, что фича ветка больше не доступна, используя команду git branch.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git branch
* master
devasc@labvm:~/labs/devnet-src/git-intro$
```

Часть 6: Обработка конфликтов слияния

Иногда может возникнуть конфликт слияния. Это происходит, когда вы внесли перекрывающиеся изменения в файл, а Git не может автоматически объединить изменения.

В этой части вы создадите тестовую ветвь, измените ее содержимое, выполните постановку и фиксацию тестовой ветви, переключитесь на главную ветвь, снова измените содержимое, выполните постановку и фиксацию главной ветви, попытаетесь объединить ветви, обнаружите и разрешите конфликт, снова выполните постановку и фиксацию главной ветви и проверите свою фиксацию.

Шаг 1: Создайте тестовую ветку.

Создайте новую ветку test.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git branch test
```

Шаг 2: Переключитесь на ветку тест.

Проверьте (переключитесь на) ветку test.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git checkout test
Switched to branch 'test'
devasc@labvm:~/labs/devnet-src/git-intro$
```

Убедитесь, что рабочая ветвь является тестовой.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git branch
  master
* test
devasc@labvm:~/labs/devnet-src/git-intro$
```

Шаг 3: Проверьте текущее содержимое файла DEVASC.txt.

Проверьте текущее содержимое файла DEVASC.txt. Обратите внимание, что первая строка включает слово "Cisco".

```
devasc@labvm:~/labs/devnet-src/git-intro$ cat DEVASC.txt
Я на пути к успешной сдаче экзамена Cisco DEVASC
Я начинаю понимать Git!
Этот текст был добавлен первоначально, во время работы над функцией
devasc@labvm:~/labs/devnet-src/git-intro$
```

Шаг 4: Измените содержимое файла DEVASC.txt в тестовой ветке.

Используйте команду sed, чтобы изменить слово "Cisco" на "NetAcad" в файле DEVASC.txt.

```
devasc@labvm:~/labs/devnet-src/git-intro$ sed -i 's/Cisco/NetAcad/'
DEVASC.txt
```

Шаг 5: Проверьте содержимое измененного файла DEVASC.txt в тестовой ветке.

Проверьте изменения в файле DEVASC.txt.

```
devasc@labvm:~/labs/devnet-src/git-intro$ sed -i 's/Cisco/NetAcad/' DEVASC.txt
devasc@labvm:~/labs/devnet-src/git-intro$ cat DEVASC.txt
Я на пути к успешной сдаче экзамена NetAcad DEVASC
Я начинаю понимать Git!
Этот текст был добавлен первоначально, во время работы над функцией
devasc@labvm:~/labs/devnet-src/git-intro$
```

Шаг 6: Поставьте и зафиксируйте тестовую ветку.

Поставьте и зафиксируйте файл одной командой git commit -a. Опция -a влияет только на файлы, которые были изменены и удалены. Она не влияет на новые файлы.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git commit -a -m "Замена Cisco на NetAcad"
[test 830eab9] Замена Cisco на NetAcad
 1 file changed, 1 insertion(+), 1 deletion(-)
devasc@labvm:~/labs/devnet-src/git-intro$
```

Шаг 7: Проверьте мастер-ветку.

Проверьте (переключитесь на) master-ветку.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git checkout master
Switched to branch 'master'
devasc@labvm:~/labs/devnet-src/git-intro$
```

Убедитесь, что ветвь master является вашей текущей рабочей ветвью.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git branch
* master
  test
devasc@labvm:~/labs/devnet-src/git-intro$
```

Шаг 8: Измените содержимое файла DEVASC.txt в основной ветви.

Используйте команду sed, чтобы изменить слово "Cisco" на "DevNet" в файле DEVASC.txt.

```
devasc@labvm:~/labs/devnet-src/git-intro$ sed -i 's/Cisco/DevNet/' DEVASC.txt
```

Шаг 9: Проверьте содержимое измененного файла DEVASC.txt в основной ветви.

Проверьте изменения в файле.

```
devasc@labvm:~/labs/devnet-src/git-intro$ cat DEVASC.txt
```

```
devasc@labvm:~/labs/devnet-src/git-intro$ git checkout master
Switched to branch 'master'
devasc@labvm:~/labs/devnet-src/git-intro$ sed -i 's/Cisco/DevNet/' DEVASC.txt
devasc@labvm:~/labs/devnet-src/git-intro$ cat DEVASC.txt
Я на пути к успешной сдаче экзамена DevNet DEVASC
Я начинаю понимать Git!
Этот текст был добавлен первоначально, во время работы над функцией
devasc@labvm:~/labs/devnet-src/git-intro$
```

Шаг 10: Поставьте и зафиксируйте мастер-ветвь.

Застэйжите и зафиксируйте файл с помощью команды git commit -a.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git commit -a -m "Замена Cisco на DevNet"
```

```
devasc@labvm:~/labs/devnet-src/git-intro$ git commit -a -m "Замена Cisco на DevNet"
[master db6807b] Замена Cisco на DevNet
 1 file changed, 1 insertion(+), 1 deletion(-)
devasc@labvm:~/labs/devnet-src/git-intro$
```

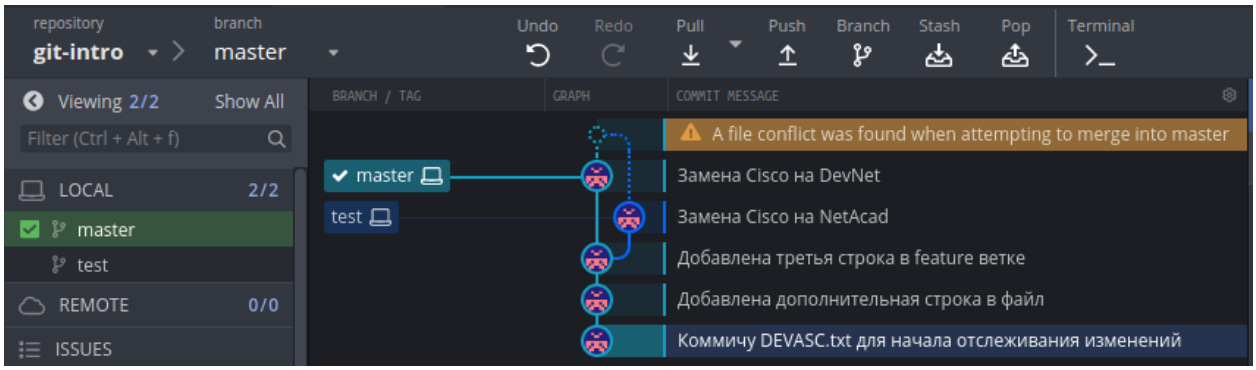
Шаг 11: Попытайтесь объединить тестовую ветвь с основной ветвью.

Попытка объединить историю тестовой ветки с основной веткой.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git merge test
Auto-merging DEVASC.txt
CONFLICT (content): Merge conflict in DEVASC.txt
Automatic merge failed; fix conflicts and then commit the result.
devasc@labvm:~/labs/devnet-src/git-intro$
```

Шаг 12: Найдите конфликт.

Например удобно смотреть историю ветвления в приложении GitKraken или Fork, которое вы можете дополнительно установить:



А пока вернемся в терминал. Используйте команду git log для просмотра коммитов. Обратите внимание, что версия HEAD является основной ветвью. Это будет полезно на следующем этапе.

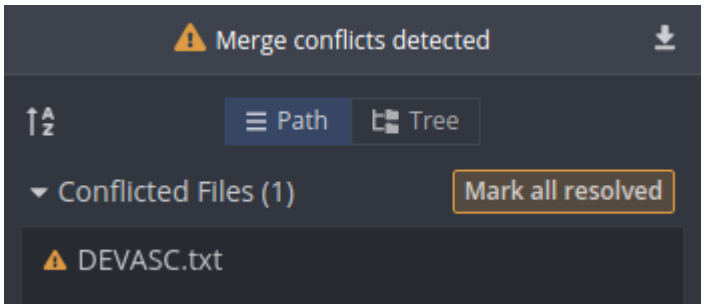
```
devasc@labvm:~/labs/devnet-src/git-intro$ git log
```

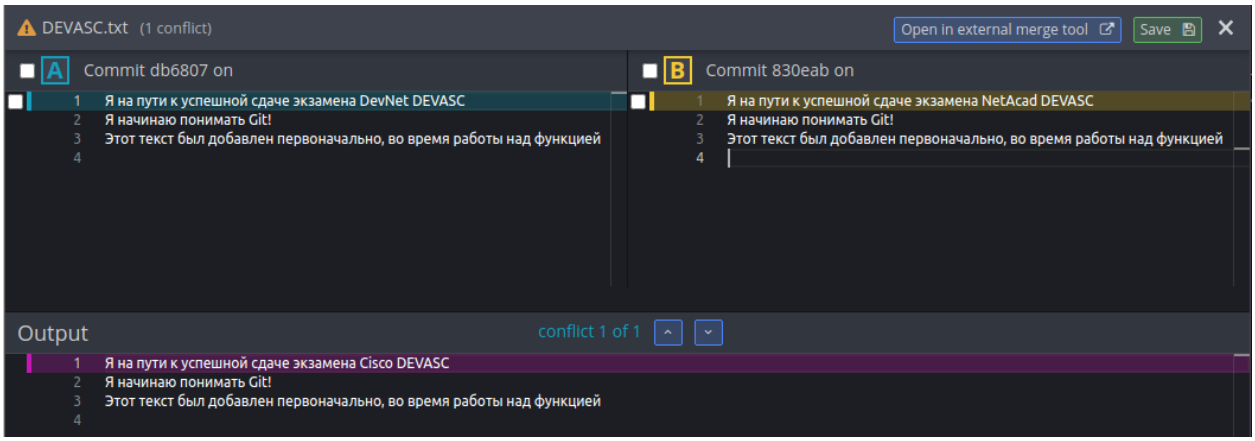
С помощью команды cat просмотрите содержимое файла DEVASC.txt. Теперь файл содержит информацию, которая поможет вам найти конфликт. Версия HEAD (ведущая ветвь), содержащая слово "DevNet", конфликтует с версией тестовой ветви и словом "NetAcad".

```
devasc@labvm:~/labs/devnet-src/git-intro$ cat DEVASC.txt
<<<<<<< HEAD
Я на пути к успешной сдаче экзамена DevNet DEVASC
=======
Я на пути к успешной сдаче экзамена NetAcad DEVASC
>>>>>>> test
Я начинаю понимать Git!
Этот текст был добавлен первоначально, во время работы над функцией
devasc@labvm:~/labs/devnet-src/git-intro$
```

```
devasc@labvm:~/labs/devnet-src/git-intro$ cat DEVASC.txt
<<<<<<< HEAD
Я на пути к успешной сдаче экзамена DevNet DEVASC
=======
Я на пути к успешной сдаче экзамена NetAcad DEVASC
>>>>>>> test
Я начинаю понимать Git!
Этот текст был добавлен первоначально, во время работы над функцией
devasc@labvm:~/labs/devnet-src/git-intro$
```

Тоже самое мы можем наблюдать в GitKraken:





Шаг 13: Вручную отредактируйте файл DEVASC.txt, чтобы удалить противоречивый текст.

Используйте команду vim для редактирования файла.

```
devasc@labvm:~/labs/devnet-src/git-intro$ vim DEVASC.txt
```

С помощью стрелок вверх и вниз выберите нужную строку текста. Нажмите dd (удалить) на следующих выделенных строках. dd удалит строку, на которой находится курсор.

```
<<<<<<< HEAD
Я на пути к успешной сдаче экзамена DevNet DEVASC
=======
Я на пути к успешной сдаче экзамена NetAcad DEVASC exam
>>>>>>> test
Я начинаю понимать Git!
Этот текст был добавлен первоначально, во время работы над функцией
```

```
Я на пути к успешной сдаче экзамена DevNet DEVASC
Я начинаю понимать Git!
Этот текст был добавлен первоначально, во время работы над функцие
```

Сохраните изменения в vim, нажав ESC, затем наберите : (двоеточие), затем wq и нажмите enter.

```
ESC
:
wq
<Enter or Return>
```

```
Я на пути к успешной сдаче экзамена DevNet DEVASC
Я начинаю понимать Git!
Этот текст был добавлен первоначально, во время работы над функцией
:wq
```

Шаг 14: Проверьте свои правки в файле DEVASC.txt в основной ветви.

Проверьте изменения с помощью команды cat

```
devasc@labvm:~/labs/devnet-src/git-intro$ cat DEVASC.txt
```

Шаг 15: Застэйджите и зафиксируйте мастер-ветку.

Застэйджите и зафиксируйте DEVASC.txt в ветке master с помощью команды git commit -a.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git add DEVASC.txt
devasc@labvm:~/labs/devnet-src/git-intro$ git commit -a -m "Ручное
слияние из тестовой ветки"
[master 22d3da4] Ручное слияние из тестовой ветки
devasc@labvm:~/labs/devnet-src/git-intro$
```

Шаг 16: Проверьте коммит.

Используйте команду git log для проверки фиксации. При необходимости вы можете использовать q, чтобы выйти из отображения журнала git.

```
devasc@labvm:~/labs/devnet-src/git-intro$ git log
```

Часть 7: Интеграция Git с GitHub

До сих пор все изменения, которые вы внесли в свой файл, хранились на вашей локальной машине. Git работает локально и не требует центрального файлового сервера или облачного хостинга. Git позволяет пользователю локально хранить файлы и управлять ими.

Хотя Git полезен для одного пользователя, интеграция локального репозитория Git с облачным сервером, таким как GitHub, полезна при работе в команде. Каждый член команды хранит копию репозитория на своей локальной машине и обновляет центральный облачный репозиторий для обмена любыми изменениями.

Существует довольно много популярных сервисов Git, включая GitHub, Stash от Atlassian и GitLab. Поскольку он легко доступен, в этих примерах вы будете использовать GitHub.

Шаг 1: Создайте учетную запись GitHub.

Если вы не сделали этого ранее, перейдите на сайт github.com и создайте учетную запись GitHub. Если у вас есть учетная запись GitHub, перейдите к шагу 2.

Шаг 2: Войдите в свою учетную запись GitHub.

Войдите в свою учетную запись GitHub.

Шаг 3: Создайте репозиторий.

Выберите кнопку "**New repository**" или нажмите на значок "+" в правом верхнем углу и выберите "Новое хранилище".

Создайте репозиторий, используя следующую информацию:

Имя репозитория: **devasc-study-team**

Описание: Совместная работа по материалам DEVASC

Публичный/частный: Частный

Выбрать: **Create repository**

Шаг 4: Создайте новый каталог devasc-study-team

Если вы еще не находитесь в каталоге git-intro, перейдите в него сейчас.

```
devasc@labvm:~$ cd ~/labs/devnet-src/git-intro
```

Создайте новый каталог с именем **devasc-study-team**. Каталог не обязательно должен совпадать с именем хранилища.

```
devasc@labvm:~/labs/devnet-src/git-intro$ mkdir devasc-study-team
```

Шаг 5: Измените каталог на devasc-study-team.

Используйте команду cd, чтобы изменить каталоги на devasc-study-team.

```
devasc@labvm:~/labs/devnet-src/git-intro$ cd devasc-study-team
devasc@labvm:~/labs/devnet-src/git-intro/devasc-study-team$
```

Шаг 6: Скопируйте файл DEVASC.

С помощью команды cp скопируйте файл DEVASC.txt из родительского каталога git-intro в подкаталог devasc-study-team. Две точки и косая черта перед именем файла указывают на родительский каталог. Пробел и точка после имени файла указывают на копирование файла в текущий каталог с тем же именем.

```
devasc@labvm:~/labs/devnet-src/git-intro/devasc-study-team$ cp
../DEVASC.txt .
```

Проверьте, что файл был скопирован с помощью команды ls, а содержимое файла - с помощью команды cat.

```
devasc@labvm:~/labs/devnet-src/git-intro/devasc-study-team$ ls
DEVASC.txt
devasc@labvm:~/labs/devnet-src/git-intro/devasc-study-team$ cat
DEVASC.txt
```

Шаг 7: Инициализируйте новый Git-репозиторий.

Используйте команду git init для инициализации текущего каталога (devasc-study-team) в качестве репозитория Git. Появившееся сообщение указывает на то, что вы создали локальный репозиторий внутри вашего проекта, содержащийся в скрытом каталоге .git. Именно здесь находится вся история изменений.

```
devasc@labvm:~/labs/devnet-src/git-intro/devasc-study-team$ git init
Initialized empty Git repository in /home/devasc/src/git-intro/devasc-study-team/.git/
devasc@labvm:~/labs/devnet-src/git-intro/devasc-study-team$
```

Затем проверьте свои глобальные переменные git с помощью команды git config --list.

```
devasc@labvm:~/labs/devnet-src/git-intro/devasc-study-team$ git config --list
user.name=SampleUser
user.email=sample@example.com
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
devasc@labvm:~/labs/devnet-src/git-intro/devasc-study-team$
```

Если переменные user.name и user.email не соответствуют вашим учетным данным GitHub, измените их сейчас.

```
devasc@labvm:~$ git config --global user.name "Имя пользователя GitHub"
devasc@labvm:~$ git config --global user.email Адрес электронной почты GitHub
```

Шаг 8: Соедините Git-репозиторий с репозиторием GitHub.

Используйте команду git remote add, чтобы добавить URL-адрес Git в качестве удаленного псевдонима. Значение "origin" указывает на вновь созданный репозиторий на GitHub. Используйте ваше имя пользователя GitHub в пути URL для github-username.

Примечание: Ваше имя пользователя чувствительно к регистру.

```
devasc@labvm:~/labs/devnet-src/git-intro/devasc-study-team$ git remote add origin https://github.com/имя-пользователя-github-/devasc-study-team.git
```

Убедитесь, что удаленный репозиторий на github.com, связан с локальным.

```
devasc@labvm:~/labs/devnet-src/git-intro/devasc-study-team$ git remote --verbose
origin https://github.com/username/devasc-study-team.git (fetch)
origin https://github.com/username/devasc-study-team.git (push)
devasc@labvm:~/labs/devnet-src/git-intro/devasc-study-team$
```

Просмотрите журнал git. Ошибка указывает на отсутствие фиксаций.

```
devasc@labvm:~/labs/devnet-src/git-intro/devasc-study-team$ git log
fatal: your current branch 'master' does not have any commits yet
```

Шаг 9: Застэйджите и закоммитьте файл DEVASC.txt.

Используйте команду git add для стэджинга файла DEVASC.txt.

```
devasc@labvm:~/labs/devnet-src/git-intro/devasc-study-team$ git add DEVASC.txt
```

Используйте команду git commit для коммита файла DEVASC.txt.

```
devasc@labvm:~/labs/devnet-src/git-intro/devasc-study-team$ git commit -m "Добавлен файл DEVASC.txt в devasc-study-team"
[master (root-commit) c60635f] Добавлен файл DEVASC.txt в devasc-study-team
 1 file changed, 3 insertions(+)
 create mode 100644 DEVASC.txt
devasc@labvm:~/labs/devnet-src/git-intro/devasc-study-team$
```

Шаг 10: Проверьте коммит.

Используйте команду git log для проверки фиксации.

```
devasc@labvm:~/labs/devnet-src/git-intro/devasc-study-team$ git log
commit c60635fe4a1f85667641afb9373e7f49a287bdd6 (HEAD -> master)
Author: username <user@example.com>
Date:   Thu Apr 28 23:41:08 2022 +0000

    Добавлен файл DEVASC.txt в devasc-study-team
devasc@labvm:~/labs/devnet-src/git-intro/devasc-study-team$
```

Используйте команду git status для просмотра информации о состоянии. Фраза "working tree clean" означает, что Git сравнил ваш список файлов с тем, что вы сообщили Git'у, и это чистый лист, на котором нет ничего нового.

```
devasc@labvm:~/labs/devnet-src/git-intro/devasc-study-team$ git status
On branch master
nothing to commit, working tree clean
devasc@labvm:~/labs/devnet-src/git-intro/devasc-study-team$
```

Шаг 11: Запушьте файл из Git в GitHub.

Используйте команду git push origin master для отправки (push'а) файла в ваш репозиторий GitHub. Вам будет предложено ввести имя пользователя и пароль, которые вы использовали при создании учетной записи GitHub.

```
devasc@labvm:~/labs/devnet-src/git-intro/devasc-study-team$ git push origin master
Username for 'https://github.com': имя_пользователя
Password for 'https://username@github.com': пароль
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 347 bytes | 347.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/username/devasc-study-team.git
 * [new branch]      master -> master
devasc@labvm:~/labs/devnet-src/git-intro/devasc-study-team$
```

Примечание: Если после ввода имени пользователя и пароля вы получаете фатальную ошибку о том, что репозиторий не найден, скорее всего, вы ввели неверный URL. Вам необходимо отменить команду git add командой git remote rm origin.

Шаг 12: Проверьте файл на GitHub.

Зайдите в свою учетную запись GitHub и в разделе "Репозитории" выберите имя пользователя/devasc-study-team.

Вы должны увидеть, что файл DEVASC.txt был добавлен в этот репозиторий GitHub. Нажмите на файл, чтобы просмотреть его содержимое.

Содержание отчета:

1. Титульный лист
2. Скриншоты промежуточных действий по каждой из Частей.
3. Ссылка на репозиторий GitHub

Лабораторная работа №3.4.6

Знакомство с классами в Python

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC

Часть 2: Обзор функций, методов и классов

Часть 3: Определение функции

Часть 4: Определение класса с методами

Часть 5: Обзор сценария circleClass.py

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

Порядок выполнения работы

Часть 1: Запуск виртуальной машины DEVASC

Если вы еще не завершили **лабораторную работу - Установка лабораторной среды виртуальной машины**, сделайте это сейчас. Если вы уже завершили эту лабораторную работу, запустите виртуальную машину **DEVASC**.

Часть 2: Рассмотрим функции, методы и классы

В этой части вы рассмотрите разницу между функциями и методами. Вы также рассмотрите основную структуру класса.

Шаг 1: Что такое функция?

В качестве краткого обзора напомним, что функция - это независимо определенный блок кода, который вызывается по имени. В следующем примере функция с именем functionName определяется, а затем вызывается. Обратите внимание, что это независимый блок кода. Он не инкапсулирован в какой-либо другой код.

```
# Определение функции
```

```
def functionName():
    ...блоки кода...

# Вызов функции
functionName()
```

Шаг 2: Что такое метод?

Метод, однако, не может быть вызван сам по себе. Он зависит от объекта, в котором он определен. В следующем примере объявлен класс className и определены три метода. Класс инстанцируется, а затем вызывается каждый метод класса.

Примечание: В этом псевдокоде не показан явно метод конструктора класса __init__ с переменной self. Этот специальный метод рассматривается ниже.

```
# Определение коасса
class className

    # Определение метода
    def method1Name
        ...блоки кода

    # Определение ещё одного метода
    def method2Name
        ...блоки кода

    # Определение и ещё одного метода
    def method3Name
        ...блоки кода

# Создаем экземпляр класса
myClass = className()

# Вызовем методы класса у его экземпляра.
myClass.method1Name()
myClass.method2Name()
myClass.method3Name()
```

Часть 3: Определение функции

В этой части вы определите функцию с аргументами, а затем вызовете функцию.

Откройте новый текстовый файл и сохраните его под именем **myCity.py** в каталоге ~/labs/devnet-src/python.

```
def myCity(city):
     print("Мой город - " + city + ".")
```

Вызовите функцию myCity, передавая ей различные значения для city, как показано в следующих примерах.

```
myCity("Санкт-Петербург")
```

```
myCity("Москва")
myCity("Нефтеюганск")
```

Сохраните и запустите файл myCity.py. Вы должны получить следующий результат.

```
devasc@labvm:~/labs/devnet-src/python$ python3 myCity.py
Мой город - Санкт-Петербург.
Мой город - Москва.
Мой город - Нефтеюганск.
devasc@labvm:~/labs/devnet-src/python$
```

Часть 4: Определение класса с методами

В этой части вы определите класс, с помощью функции __init__() определите метод для класса, а затем создадите экземпляры класса.

Шаг 1: Определите, а затем инстанцируйте класс с помощью метода __init__().

Класс в Python используется для создания объектов, имеющих свойства и методы. Все классы Python обычно включают явно определенную функцию __init__(), хотя вы можете создать класс и без ее определения. Функция __init__() всегда запускается при инстанцировании класса. При инстанцировании класса создается его копия, которая наследует все переменные и методы класса.

Примечание: Хотя функцию иногда называют __init__(), она зависит от класса. Поэтому технически она является методом.

Откройте новый текстовый файл и сохраните его под именем **myLocation.py**.

Определите класс с именем Location и нажмите Enter. Если вы работаете в VS Code, то текстовый редактор должен автоматически сделать отступ в четыре пробела.

```
class Location:
    |<-- Ваш курсор должен быть здесь
```

Далее определите функцию __init__(). По традиции, первый параметр называется self. Параметр self является ссылкой на текущий экземпляр класса и используется для доступа к переменным, принадлежащим всему классу. Затем функции __init__() присваиваются все переменные, необходимые всему классу. В следующем примере определите переменные name и country. Нажмите дважды Enter, а затем дважды backspace до левого поля.

```
    def __init__(self, name, country):
        self.name = name
        self.country = country

|<-- Ваш курсор должен быть здесь
```

Вы можете проверить, что теперь этот класс готов к использованию. Создайте экземпляр класса, присвоив ему имя по своему выбору. Затем укажите два параметра класса name и country. В следующем примере класс

Location используется для инстанцирования класса loc с указанными вами именем и страной. Используйте свое имя и страну.

```
loc = Location("Ваше_Имя", "Ваша_Страна")
```

Чтобы проверить, что в инстанцированном классе loc теперь есть назначенные вами имя и страна, добавьте в сценарий операторы печати.

```
print(loc.name)
print(loc.country)
```

Чтобы убедиться, что loc действительно является классом, добавьте следующий оператор print, который выведет тип данных для loc.

```
print(type(loc))
```

Сохраните и запустите свой скрипт. Вы должны получить следующий результат, но с указанными вами именем и страной.

```
devasc@labvm:~/labs/devnet-src/python$ python3 myLocation.py
Ваше_Имя
Ваша_Страна
<class '__main__.Location'>
devasc@labvm:~/labs/devnet-src/python$
```

Шаг 2: Добавьте метод в класс Location.

Теперь добавьте в класс Location метод, который может быть вызван программистом при инстанцировании класса. В этом простом примере создайте метод для печати утверждения: "Меня зовут [имя], и я живу в [страна]".

Удалите код, начинающийся с инстанцирования класса loc. Теперь ваш сценарий myLocation.py должен содержать только следующий код.

```
class Location:
    def __init__(self, name, country):
        self.name = name
        self.country = country
```

Установив курсор в конец строки self.country = country, дважды нажмите клавишу Enter и один раз backspace.

```
        self.country = country

    |<-- Ваш курсор должен быть здесь
```

Определите новый метод под названием myLocation и присвойте ему параметр self, чтобы новый метод мог получить доступ к переменным, определенным в функции __init__(). Затем определите оператор print для вывода строки, указанной выше.

Примечание: Оператор print() должен располагаться на одной строке.

```
def myLocation(self):
  print("Привет, меня зовут " + self.name + ", и я живу в стране " +
self.country + ".")
```

Дважды нажмите клавишу Enter и дважды backspace.

Сохраните и запустите свой сценарий, чтобы убедиться в отсутствии ошибок. Вы пока не получите никаких результатов.

Шаг 3: Создайте несколько экземпляров класса Location и вызовите метод myLocation.

Теперь, когда у вас есть класс, вы можете инстанцировать его столько раз, сколько захотите, каждый раз предоставляя разные значения для переменных класса.

Добавьте следующий код в сценарий myLocation.py для инстанцирования класса Location и вызова метода. Комментарии добавлять не нужно.

```
# Первый экземпляр класса Location
loc1 = Location("Денис", "Россия")
# Вызов метода
loc1.myLocation()
```

Сохраните и запустите свой сценарий. Вы должны получить следующий результат.

```
devasc@labvm:~/labs/devnet-src/python$ python3 myLocation.py
Привет, меня зовут Денис, и я живу в стране Россия.
devasc@labvm:~/labs/devnet-src/python$
```

Добавьте еще два инстанса, а затем четвертый, в котором укажите имя и значения для your_loc.

```
loc2 = Location("김현아", "한국")
loc3 = Location("Шерхан", "Казахстан")
loc2.myLocation()
loc3.myLocation()
your_loc = Location("Ваше_Имя", "Ваша_Страна")
your_loc.myLocation()
```

Сохраните и запустите свой сценарий. Вы должны получить следующий результат.

```
devasc@labvm:~/labs/devnet-src/python$ python3 myLocation.py
Привет, меня зовут Денис, и я живу в стране Россия.
Привет, меня зовут 김현아, и я живу в стране 한국.
Привет, меня зовут Шерхан, и я живу в стране Казахстан.
Привет, меня зовут Ваше_Имя, и я живу в стране Ваша_Страна.
devasc@labvm:~/labs/devnet-src/python$
```

Шаг 4: Просмотрите на весь код myLocation.py.

Если у вас возникли ошибки при работе с вашим скриптом, просмотрите следующий пример, который содержит весь код, использованный в этой части.

```
# Определяем клас с полями **name** и **country**.
# Затем определим метод, принадлежащий классу. Его цель
# вывести предложение, в котором используются переменные.
```

```
class Location:
    def __init__(self, name, country):
        self.name = name
        self.country = country

    def myLocation(self):
        print("Привет, меня зовут " + self.name + ", и я живу в стране
" + self.country + ".")

# Первый экземпляр класса Location
loc1 = Location("Денис", "Россия")
# Вызов метода
loc1.myLocation()

# Ещё три экземпляра и вызова методов для класса Location
loc2 = Location("김현아", "한국")
loc3 = Location("Шерхан", "Казахстан")
loc2.myLocation()
loc3.myLocation()
your_loc = Location("Ваше_Имя", "Ваша_Страна")
your_loc.myLocation()
```

Часть 5: Изучим скрипт circleClass.py

Пример ниже показывает, как создать класс, который вычисляет длинну окружности, а затем выводит вычисленное значение. В этом сценарии есть несколько моментов, на которые следует обратить внимание.

Класс включает три метода, в том числе функцию __init__(). Функция __init__() предоставляет метод для ввода значения радиуса.

Метод circumference вычисляет длинну окружности и возвращает значение, сохраняя его в переменной circumferenceValue.

Метод printCircumference печатает строку. Обратите внимание, что переменные приведены к строкам с помощью функции str(). В противном случае оператор print выдал бы ошибку, поскольку self.radius и myCircumference не являются строками.

Класс Circle инстанцировался три раза.

```
# Учитывая значение радиуса, выведите длинну окружности.
# Длинна окружности вычисляется по формуле: c = pi * 2 * radius

class Circle:

    def __init__(self, radius):
        self.radius = radius
```

```
    def circumference(self):
      pi = 3.14
      circumferenceValue = pi * self.radius * 2
      return circumferenceValue

    def printCircumference(self):
      myCircumference = self.circumference()
      print ("Длина окружности с hflbecjv" + str(self.radius) + " равна
" + str(myCircumference))

# Первый экземпляр класса Circle.
circle1 = Circle(2)
# Вызовите функцию printCircumference для экземпляра circle1.
circle1.printCircumference()

# Ещё два экземпляра класса Circle.
circle2 = Circle(5)
circle2.printCircumference()

circle3 = Circle(7)
circle3.printCircumference()
```

Содержание отчета:

1. Титульный лист
2. Скриншоты выводов к **myLocation.py и circleClass.py**.
3. GitHub дириктория Lab-3-4-6 со скриптами **myLocation.py и circleClass.py**

Лабораторная работа №3.5.7

Unit-тестирование на Python

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC

Часть 2: Изучение опций в unittest Framework

Часть 3: Тестирование функции Python с помощью unittest

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

Порядок выполнения работы

Часть 1: Запуск виртуальной машины DEVASC

Если вы еще не завершили **лабораторную работу - Установка лабораторной среды виртуальной машины**, сделайте это сейчас. Если вы

уже завершили эту лабораторную работу, запустите виртуальную машину **DEVASC**.

Часть 2: Изучение возможностей в unittest Framework

Python предоставляет фреймворк модульного тестирования (называемый unittest) как часть стандартной библиотеки Python. Если вы не знакомы с этим фреймворком, изучите "Python unittest Framework" для ознакомления. Найдите его в Интернете, чтобы найти документацию на сайте python.org. Вам понадобятся эти знания или документация, чтобы ответить на вопросы этой части.

Ответьте на следующие вопросы

Какой класс unittest вы используете для создания отдельного блока тестирования?

- Предполагаемый ответ: Модуль unittest предоставляет базовый класс TestCase, который можно использовать для создания новых тестовых случаев.

Программа запуска тестов отвечает за выполнение тестов и предоставление вам результатов. Программа запуска тестов может иметь графический интерфейс, но пока мы ограничимся командной строкой для запуска тестов.

Как программа запуска тестов узнает, какие методы являются тестом?

- Предполагаемый ответ: Методы, имена которых начинаются с test_, информируют программу запуска тестов о том, какие методы являются тестами.

Какая команда выведет все опции для unittest, показанные в следующем выводе?

```
devasc@labvm:~/labs/devnet-src$ команда, дающая вывод ниже
<вывод опущен>
optional arguments:
  -h, --help           show this help message and exit
  -v, --verbose        Verbose output
  -q, --quiet          Quiet output
  --locals             Show local variables in tracebacks
  -f, --failfast       Stop on first fail or error
  -c, --catch          Catch Ctrl-C and display results so far
  -b, --buffer         Buffer stdout and stderr during tests
  -k TESTNAMEPATTERNS  Only run tests which match the given substring

Examples:
  python3 -m unittest test_module               - run tests from test_module
  python3 -m unittest module.TestClass         - run tests from module.TestClass
  python3 -m unittest module.Class.test_method  - run specified test method
  python3 -m unittest path/to/test_file.py      - run tests from test_file.py
```

```
<Вывод опущен>
For test discovery all test modules must be importable from the top level
directory of the project.
devasc@labvm:~/labs/devnet-src$
```

- Предполагаемый ответ: python3 -m unittest -h

Часть 3: Тестирование функции Python с помощью unittest

В этой части вы будете использовать unittest для тестирования функции, выполняющей рекурсивный поиск в объекте JSON. Функция возвращает значения, помеченные заданным ключом. Программистам часто приходится выполнять подобные операции с объектами JSON, возвращаемыми вызовами API.

В этом тесте будут использоваться три файла, представленные в следующей таблице:

Файл	Описание
recursive_json_search.py	Этот скрипт будет включать функцию **json_search()**, которую мы хотим протестировать
test_data.py	Это данные, которые ищет функция **json_search()**
test_json_search.py	Это файл, который вы создадите для тестирования функции **json_search()** в сценарии **recursive_json_search.py**

Шаг 1: Просмотрите файл test_data.py.

Откройте файл **~/labs/devnet-src/unittest/test_data.py** и изучите его содержимое. Эти данные в формате JSON типичны для данных, возвращаемых при вызове Cisco DNA Center API. Образец данных достаточно сложен, чтобы быть хорошим тестом. Например, в нем чередуются типы dict и list.

```
devasc@labvm:~/labs/devnet-src$ more unittest/test_data.py
key1 = "issueSummary"
key2 = "XY&^$#*@!1234%^&"

data = {
  "id": "AWcvsjx864kVeDHDi2gB",
  "instanceId": "E-NETWORK-EVENT-AWcvsjx864kVeDHDi2gB-1542693469197",
  "category": "Warn",
  "status": "NEW",
  "timestamp": 1542693469197,
  "severity": "P1",
  "domain": "Availability",
  "source": "DNAC",
  "priority": "P1",
  "type": "Network",
  "title": "Device unreachable",
  "description": "This network device leaf2.abc.inc is unreachable from controll
er. The device role is ACCESS.",
  "actualServiceId": "10.10.20.82",
  "assignedTo": "",
```

```
        "enrichmentInfo": {
          "issueDetails": {
            "issue": [
              {
--More--(12%)
```

Шаг 2: Создайте функцию json_search(), которую вы будете тестировать.

Наша функция должна принимать ключ и объект JSON в качестве входных параметров и возвращать список совпавших пар ключ/значение. Вот текущая версия функции, которую необходимо протестировать, чтобы убедиться, что она работает так, как задумано. Цель этой функции - сначала импортировать тестовые данные. Затем она ищет данные, которые соответствуют ключевым переменным в файле **test_data.py**. Если она находит совпадение, то добавляет совпавшие данные в список. Функция **print()** в конце печатает содержимое списка для первой переменной **key1 = "issueSummary"**.

```
from test_data import *
def json_search(key,input_object):
    ret_val=[]
if isinstance(input_object, dict): # Словарь для обхода
        for k, v in input_object.items(): # поиск ключа в словаре
            if k == key:
                temp={k:v}
                ret_val.append(temp)
            if isinstance(v, dict): # значение является другим словарём, поэтому
повторим поиск в нём
                json_search(key,v)
            elif isinstance(v, list): # это список
                for item in v:
                    if not isinstance(item, (str,int)): # если словарь или список
также повторим поиск
                        json_search(key,item)
    else: # Обход списка, поскольку некоторые API возвращают объект JSON в виде
списка
        for val in input_object:
            if not isinstance(val, (str,int)):
                json_search(key,val)
    return ret_val
print(json_search("issueSummary",data))
```

Откройте файл ~/labs/devnet-src/unittest/recursive_json_search.py.

Скопируйте приведенный выше код в файл и сохраните.

Запустите код. Вы должны получить отсутствие ошибок и вывод [ ], указывающий на пустой список. Если функция json_search() была закодирована правильно (а это не так), то это говорит о том, что нет данных с ключом "issueSummary", о которых сообщают данные JSON, возвращаемые Cisco DNA Center API. Другими словами, нет никаких проблем, о которых можно было бы сообщить.

```
devasc@labvm:~/labs/devnet-src/unittest$ python3
recursive_json_search.py
```

```
[]
devasc@labvm:~/labs/devnet-src/unittest$
```

```
devasc@labvm:~/labs/devnet-src/unittest$ python3 recursive_json_search.py
[]
devasc@labvm:~/labs/devnet-src/unittest$
```

Но как узнать, что функция **json_search()** работает так, как задумано? Вы можете открыть файл **test_data.py** и выполнить поиск по ключу **"issueSummary"**, как показано ниже. Если вы это сделаете, то действительно обнаружите, что проблема существует. Это небольшой набор данных и относительно простой рекурсивный поиск. Однако производственные данные и код редко бывают настолько простыми. Поэтому тестирование кода жизненно важно для быстрого поиска и исправления ошибок в коде.

```
<Вывод опущен>
      "issue": [
        {
          "issueId": "AWcvsjx864kVeDHDi2gB",
          "issueSource": "Cisco DNA",
          "issueCategory": "Availability",
          "issueName": "snmp_device_down",
          "issueDescription": "This network device leaf2.abc.inc is unreachable
from controller. The device role is ACCESS.",
          "issueEntity": "network_device",
          "issueEntityValue": "10.10.20.82",
          "issueSeverity": "HIGH",
          "issuePriority": "",
          "issueSummary": "Network Device 10.10.20.82 Is Unreachable From
Controller",
          "issueTimestamp": 1542693469197,
          "suggestedActions": [
            {
<Вывод опущен>
```

Шаг 3: Создайте несколько модульных тестов, которые будут проверять, работает ли функция так, как задумано.

Откройте файл ~ labs/devnet-src/unittest/test_json_search.py.

В первой строке сценария после комментария импортируйте библиотеку unittest.

```
import unittest
```

Добавьте строки для импорта тестируемой функции, а также тестовых данных JSON, которые использует функция.

```
from recursive_json_search import *
from test_data import *
```

Теперь добавьте следующий код класса **json_search_test** в файл **test_json_search.py**. Код создает подкласс TestCase фреймворка unittest. Класс определяет некоторые тестовые методы, которые будут использоваться для функции **json_search()** в сценарии **recursive_json_search.py**. Обратите внимание, что каждый тестовый метод начинается с

test_, что позволяет фреймворку unittest обнаруживать их автоматически. Добавьте следующие строки в нижнюю часть файла ~labs/devnet-src/unittest/test_json_search.py:

```
class json_search_test(unittest.TestCase):
    '''тестовый модуль для проверки поиска в `recursive_json_search.py`'''
    def test_search_found(self):
        '''ключ должен быть найден, возвразаемый список не должен быть
пустым'''
        self.assertTrue([]!=json_search(key1,data))
    def test_search_not_found(self):
        '''ключ не должен быть найден, возвразаемый список должен быть пустым
'''
        self.assertTrue([]==json_search(key2,data))
    def test_is_a_list(self):
        '''должен вернуть список'''
        self.assertIsInstance(json_search(key1,data),list)
```

В коде unittest вы используете три метода для тестирования функции поиска:

1) Учитывая существующий ключ в объекте JSON, проверьте, может ли тестирующий код найти такой ключ.
2) Учитывая несуществующий ключ в объекте JSON, проверьте, подтверждает ли тестирующий код, что ключ не может быть найден.
3) Проверьте, возвращает ли наша функция список, как она всегда должна делать.

Для создания этих тестов сценарий использует некоторые встроенные методы **assert** класса **unittest TestCase** для проверки условий. Метод **assertTrue(x)** проверяет, истинно ли условие, а **assertIsInstance(a, b)** проверяет, является ли **a** экземпляром типа **b**. Здесь используется тип **list**.

Также обратите внимание, что комментарии для каждого метода задаются с помощью тройной одинарной кавычки (''). Это необходимо, если вы хотите, чтобы при выполнении теста выводилось описание метода теста. Использование одинарного символа хэша (#) для комментария не позволит вывести описание неудачного теста.

В последней части сценария добавьте метод **unittest.main()**. Это позволит запускать unittest из командной строки. Цель **if __name__ == '__main__'** — убедиться, что метод **unittest.main()** запускается только при непосредственном выполнении скрипта. Если сценарий импортирован в другую программу, **unittest.main()** не будет запущен. *Например, для выполнения этого теста вы можете использовать не* ***unittest****, а другую программу запуска тестов.*

```
if __name__ == '__main__':
    unittest.main()
```

Шаг 4: Запустите тест, чтобы увидеть первые результаты.

Запустите тестовый сценарий в его текущем состоянии, чтобы посмотреть, какие результаты он возвращает в данный момент. Во-первых, вы видите пустой список. Во-вторых, вы видите выделенный символ **.F**. в выводе. Точка (**.**) означает, что тест прошел, а **F** означает, что тест не прошел. Таким образом, первый тест прошел, второй тест не прошел, а третий тест прошел.

```
devasc@labvm:~/labs/devnet-src/unittest$ python3 test_json_search.py
 []
.F.
======================================================================
FAIL: test_search_found (__main__.json_search_test)
ключ должен быть найден, возвразаемый список не должен быть пустым
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test_json_search.py", line 11, in test_search_found
    self.assertTrue([]!=json_search(key1,data))
AssertionError: False is not true

----------------------------------------------------------------------
Ran 3 tests in 0.001s

FAILED (failures=1)
devasc@labvm:~/labs/devnet-src/unittest$
```

```
devasc@labvm:~/labs/devnet-src/unittest$ python3 test_json_search.py
[]
.F.
======================================================================
FAIL: test_search_found (__main__.json_search_test)
ключ должен быть найден, возвразаемый список не должен быть пустым
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test_json_search.py", line 10, in test_search_found
    self.assertTrue([]!=json_search(key1,data))
AssertionError: False is not true

----------------------------------------------------------------------
Ran 3 tests in 0.001s

FAILED (failures=1)
devasc@labvm:~/labs/devnet-src/unittest$
```

Чтобы вывести каждый тест и его результаты, запустите сценарий снова под unittest с опцией **verbose** (**-v**). Обратите внимание, что для сценария **test_json_search.py** не требуется расширение **.py**. Вы можете видеть, что ваш метод тестирования **test_search_found** не работает.

Примечание: Python не обязательно запускает тесты по порядку. Тесты запускаются в алфавитном порядке на основе имен методов тестирования.

```
devasc@labvm:~/labs/devnet-src/unittest$ python3 -m unittest -v
test_json_search
```

```
devasc@labvm:~/labs/devnet-src/unittest$ python3 -m unittest -v test_json_search
[]
test_is_a_list (test_json_search.json_search_test)
должен вернуть список ... ok
test_search_found (test_json_search.json_search_test)
ключ должен быть найден, возвразаемый список не должен быть пустым ... FAIL
test_search_not_found (test_json_search.json_search_test)
ключ не должен быть найден, возвразаемый список должен быть пустым ... ok

======================================================================
FAIL: test_search_found (test_json_search.json_search_test)
ключ должен быть найден, возвразаемый список не должен быть пустым
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/devasc/labs/devnet-src/unittest/test_json_search.py", line 10, in test_search_foun
d
    self.assertTrue([]!=json_search(key1,data))
AssertionError: False is not true

----------------------------------------------------------------------
Ran 3 tests in 0.004s

FAILED (failures=1)
devasc@labvm:~/labs/devnet-src/unittest$
```

Шаг 5: Исследуйте и исправьте первую ошибку в сценарии recursive_json_search.py.

Утверждение, ключ должен быть найден, возвращаемый список не должен быть пустым .... FAIL, указывает на то, что ключ не найден. Почему? Если мы посмотрим на текст нашей рекурсивной функции, то увидим, что выражение **ret_val=[ ]** выполняется многократно, каждый раз, когда вызывается функция. Это приводит к тому, что функция опустошает список и теряет накопленные результаты от выражения **ret_val.append(temp)**, который добавляет в список, созданный **ret_val=[ ]**.

```
def json_search(key,input_object):
    ret_val=[]
    if isinstance(input_object, dict):
        for k, v in input_object.items():
            if k == key:
                temp={k:v}
                ret_val.append(temp)
```

Переместите **ret_val=[ ]** из нашей функции в **recursive_json_search.py**, чтобы итерация не перезаписывала каждый раз накопленный список.

```
ret_val=[]
def json_search(key,input_object):
```

Сохраните и запустите сценарий. Вы должны получить следующий результат, который подтверждает, что вы решили проблему. После выполнения сценария список больше не пуст.

```
devasc@labvm:~/labs/devnet-src/unittest$ python3
recursive_json_search.py
[{'issueSummary': 'Network Device 10.10.20.82 Is Unreachable From Controller'}]
devasc@labvm:~/labs/devnet-src/unittest$
```

Шаг 6: Запустите тест снова, чтобы проверить, все ли ошибки в сценарии теперь исправлены.

Вы получили некоторый вывод при последнем запуске recursive_json_search.py, однако вы ещё не можете быть уверены, что устранили все ошибки в сценарии? Запустите unittest снова без опции -v и посмотрите, возвращает ли test_json_search какие-либо ошибки. Как правило, вы не испольуете опцию **-v**, чтобы минимизировать вывод на консоль и ускорить выполнение тестов. В начале журнала вы можете увидеть ..F, что означает третий тест не прошел. Также обратите внимание, что список все еще распечатывается. Вы можете остановить это поведение, удалив функцию **print()** в сценарии **resursive_json_search.py**. Но это не обязательно.

```
devasc@labvm:~/labs/devnet-src/unittest$ python3 -m unittest
test_json_search
[{'issueSummary': 'Network Device 10.10.20.82 Is Unreachable From Controller'}]
..F
======================================================================
FAIL: test_search_not_found (test_json_search.json_search_test)
ключ не должен быть найден, возвразаемый список должен быть пустым
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/devasc/labs/devnet-src/unittest/test_json_search.py", line 14, in
test_search_not_found
    self.assertTrue([]==json_search(key2,data))
AssertionError: False is not true

----------------------------------------------------------------------
Ran 3 tests in 0.001s

FAILED (failures=1)
devasc@labvm:~/labs/devnet-src/unittest$
```

Откройте файл **test_data.py** и найдите **issueSummary**, который является значением для **key1**. Вы должны найти его дважды, но только один раз в объекте данных JSON. Но если вы будете искать значение **key2**, который является **XY&^$#*@!1234%^&**, вы обнаружите его только вверху, где он определен, потому что его нет в объекте JSON данных. Третий тест проверяет, что его там нет. В комментарии к третьему тесту говорится, что ключ не должен быть найден, должен возвращаться пустой список. Однако функция вернула непустой список.

Шаг 7: Исследуйте и исправьте вторую ошибку в сценарии recursive_json_search.py.

Просмотрите код **recursive_json_search.py** ещё раз. Обратите внимание, что **ret_val** теперь является глобальной переменной после того, как вы исправили её в предыдущем шаге. Это означает, что её значение сохраняется при нескольких вызовах функции **json_search()**. Это хороший пример того, почему использование глобальных переменных в функциях - плохая практика.

Чтобы решить эту проблему, оберните функцию **json_search()** внешней функцией. Удалите существующую функцию **json_search()** и замените её рефакторингом, приведенным ниже: (Вызов функции дважды не повредит, но повторение функции - не лучшая практика :/).

```
from test_data import *
def json_search(key,input_object):
    """
    Поиск ключа в объекте JSON, ничего не возвращается, если ключ не найден
    key : "keyword" для поиска, с учетом регистра
    input_object : JSON объект, который нужно разобрать, в данном случае
test_data.py
    inner_function() фактически выполняет рекурсивный поиск
    возвращает список пар key:value
    """
    ret_val=[]
    def inner_function(key,input_object):
        if isinstance(input_object, dict): # Словарь для обхода
            for k, v in input_object.items(): # поиск ключа в словаре
                if k == key:
                    temp={k:v}
                    ret_val.append(temp)
                if isinstance(v, dict): # значение является другим словарём,
поэтому повторим поиск в нём
                    inner_function(key,v)
                elif isinstance(v, list):
                    for item in v:
                        if not isinstance(item, (str,int)): # если это словарь
или список также повторим поиск
                            inner_function(key,item)
        else: # Обход списка, поскольку некоторые API возвращают объект JSON в
виде списка
            for val in input_object:
                if not isinstance(val, (str,int)):
                    inner_function(key,val)
    inner_function(key,input_object)
    return ret_val
print(json_search("issueSummary",data))
```

Сохраните файл и запустите **unittest** в каталоге. Вам не нужно имя файла. Это потому, что функция **unittest** Test Discovery запустит любой найденный локальный файл, имя которого начинается с test. Вы должны получить следующий результат. Обратите внимание, что все тесты теперь пройдены и список для ключа "issueSummary" заполнен. Вы можете смело удалить функцию **print()**, поскольку она обычно не используется, когда этот тест объединяется с другими тестами для более масштабного тестирования.

```
devasc@labvm:~/labs/devnet-src/unittest$ python3 -m unittest
[{'issueSummary': 'Network Device 10.10.20.82 Is Unreachable From Controller'}]
...
----------------------------------------------------------------------
```

```
Ran 3 tests in 0.001s

OK
devasc@labvm:~/labs/devnet-src/unittest$
```

```
devasc@labvm:~/labs/devnet-src/unittest$ python3 -m unittest
[{'issueSummary': 'Network Device 10.10.20.82 Is Unreachable From Controller'}]
...
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
devasc@labvm:~/labs/devnet-src/unittest$
```

Содержание отчета:

1. Титульный лист
2. Ответы на контрольные вопросы.
3. GitHub дириктория Lab-3-5-6 со поломанными так и исправленными скриптами.

Лабораторная работа №3.6.6

Парсинг различных типов данных с помощью Python

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC

Часть 2: Разбор XML в Python

Часть 3: Разбор JSON в Python

Часть 4: Разбор YAML в Python

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

Порядок выполнения работы

Часть 1: Запуск виртуальной машины DEVASC

Если вы еще не завершили **лабораторную работу - Установка лабораторной среды виртуальной машины**, сделайте это сейчас. Если вы уже завершили эту лабораторную работу, запустите виртуальную машину **DEVASC**.

Часть 2: Разбор XML в Python

Из-за гибкости, которую обеспечивает расширяемый язык разметки (XML), он может быть сложным для анализа. Поля данных XML с текстовыми метками не однозначно отображаются на типы данных по умолчанию в Python или других популярных языках. Кроме того, не всегда очевидно, как значения атрибутов должны быть представлены в данных.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<rpc message-id="1"
 xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <edit-config>
  <target>
   <candidate/>
  </target>
  <default-operation>merge</default-operation>
  <test-option>set</test-option>
  <config>
   <int8.1
    xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
    nc:operation="create"
    xmlns="http://netconfcentral.org/ns/test">9</int8.1>
  </config>
 </edit-config>
</rpc>
```

Шаг 1: Создайте сценарий для разбора данных XML.

Откройте файл **parsexml.py**, находящийся в каталоге ~/labs/devnet-src/parsing.

Импортируйте модуль ElementTree библиотеки xml и механизм регулярных выражений. Модуль ElementTree будет использоваться для выполнения синтаксического анализа. Механизм регулярных выражений будет использоваться для поиска определенных данных.

Примечание: Если у вас нет опыта использования регулярных выражений в Linux, Python или других объектно-ориентированных языках программирования, поищите учебники в интернете.

```
import xml.etree.ElementTree as ET
import re
```

Затем с помощью функции parse из ET (ElementTree) разберите файл myfile.xml и присвойте его переменной (xml). Затем получите корневой элемент с помощью функции getroot и присвойте его переменной (root).

```
xml = ET.parse("myfile.xml")
root = xml.getroot()
```

Теперь на верхнем уровне дерева можно искать содержащий тег **<edit-config>**, а когда он найден, этот блок тегов можно искать по двум именованным значениям, которые он содержит: **<default-operation>** и **<test-option>**. Создайте регулярное выражение для получения содержимого корня XML в теге **<rpc>**, а затем добавьте дополнительные регулярные выражения для углубления в содержимое, чтобы найти значение элементов **<edit-config>**, **<default-operation>** и **<test-option>**.

```
ns = re.match('{.*}', root.tag).group(0)
editconf = root.find("{}edit-config".format(ns))
defop = editconf.find("{}default-operation".format(ns))
testop = editconf.find("{}test-option".format(ns))
```

Добавьте **print()** для вывода значения элементов **<default-operation>** и **<test-option>**

```
print("Операция по умолчанию содержит: {}".format(defop.text))
```

```
print("Тестовая опция содержит: {}".format(testop.text))
```

Шаг 2: Запустите скрипт.

Сохраните и запустите файл parsexml.py. Вы должны получить следующий результат.

```
devasc@labvm:~/labs/devnet-src/parsing$ python3 parsexml.py
Операция по умолчанию содержит: merge
Тестовая опция содержит: set
devasc@labvm:~/labs/devnet-src/parsing$
```

```
devasc@labvm:~/labs/devnet-src/parsing$ python3 parsexml.py
Операция по умолчанию содержит: merge
Тестовая опция содержит: set
devasc@labvm:~/labs/devnet-src/parsing$
```

Часть 3: Разбор JSON в Python

Разбор JavaScript Object Notation (JSON) является частым требованием при взаимодействии с REST API. Алгоритм разбора выполняется следующим образом:

1) Пройдите аутентификацию, используя комбинацию пользователь/пароль, чтобы получить токен, срок действия которого истекает через определенное время. Этот маркер используется для аутентификации последующих запросов.
2) Выполните GET-запрос к REST API, аутентифицировавшись при необходимости, чтобы получить состояние ресурса, запросив JSON в качестве выходного формата.
3) Измените полученный JSON, если это необходимо.
4) Выполните POST (или PUT) к тому же REST API (опять же, аутентификация по необходимости), чтобы изменить состояние ресурса, снова запрашивая JSON в качестве выходного формата и интерпретируя его по мере необходимости, чтобы определить, была ли операция успешной.

Пример JSON для разбора — это ответ на запрос токена:


```
{

"access_token":"ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDE
wN2ItYTU3",
 "expires_in":1209600,

"refresh_token":"MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OT
EyMzQ1Njc4",
 "refreshtokenexpires_in":7776000
}
```


В Python скриптах библиотека **json** может быть использована, как нистранно, для разбора JSON в структуры данных Python и сериализации структур данных обратно в JSON. Библиотека **yaml** может быть использована для преобразования данных в YAML.

Следующая программа использует оба модуля для разбора приведенных выше данных JSON, извлечения и печати значений данных, а также вывода YAML-версии файла. Она использует метод **json** библиотеки **loads()** для разбора строки, в которую был считан файл. Затем он использует обычные ссылки на данные Python для извлечения значений из полученной структуры данных Python. Наконец, он использует функцию **yaml** библиотеки **dump()** для сериализации данных Python обратно в виде YAML на терминал.

Шаг 1: Создайте сценарий для разбора данных JSON.

Откройте файл **parsejson.py**, находящийся в каталоге ~/labs/devnet-src/parsing.

Импортируйте библиотеки **json** и **yaml**.

```
import json
import yaml
```

С помощью оператора Python **with** откройте файл **myfile.json** и установите его в переменную с именем **json_file**. Затем используйте метод **json.load** для загрузки файла JSON в строку, заданную переменной **ourjson**.

Примечание: Нет необходимости явно закрывать файл, так как оператор **with** обеспечивает правильное открытие и закрытие файла.

```
with open('myfile.json','r') as json_file:
    ourjson = json.load(json_file)
```

Добавьте оператор print для ourjson, чтобы увидеть, что теперь это словарь Python.

```
print(ourjson)
```

Шаг 2: Запустите сценарий для печати данных JSON, а затем измените его для печати интересующих вас данных.

Сохраните и запустите свой сценарий. Вы должны увидеть следующий результат.

```
devasc@labvm:~/labs/devnet-src/parsing$ python3 parsejson.py
{'access_token':
'ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN2ItYTU3',
'expires_in':                    1209600,                    'refresh_token':
'MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTEyMzQ1Njc4',
'refreshtokenexpires_in': 7776000}
devasc@labvm:~/labs/devnet-src/parsing$
```

```
devasc@labvm:~/labs/devnet-src/parsing$ python3 parsejson.py
{'access_token': 'ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN
2ItYTU3', 'expires_in': 1209600, 'refresh_token': 'MDEyMzQ1Njc4OTAxMjM0NTY3
ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTEyMzQ1Njc4', 'refreshtokenexpires_in': 777600
0}
devasc@labvm:~/labs/devnet-src/parsing$
```

Шаг 3: Вывод разобранных данных JSON в формате данных YAML.

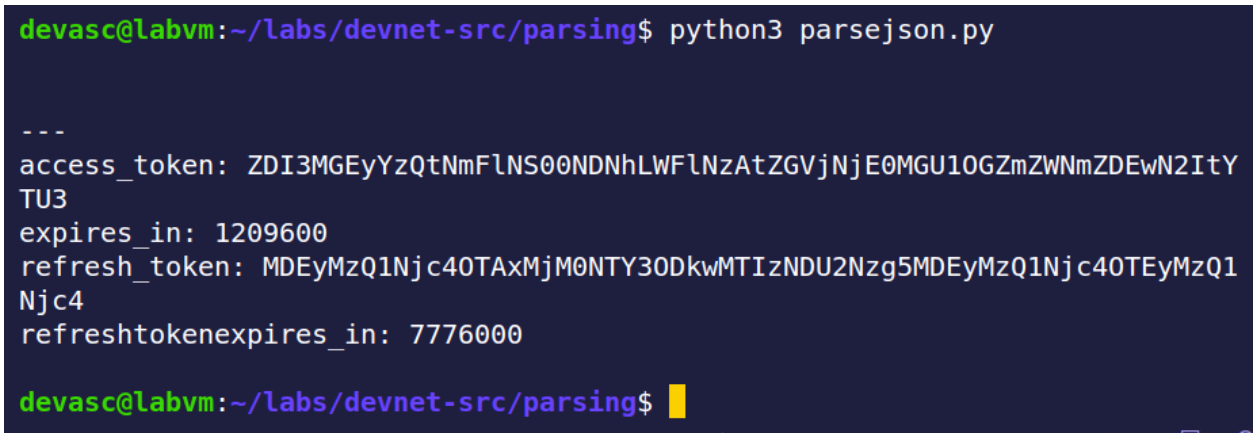
Добавьте оператор **print**, который выведет три тире, необходимые для файла YAML. Два **\n** добавят две строки после предыдущего вывода. Затем добавьте оператор для печати **ourjson** в виде данных YAML с помощью метода **dump()** библиотеки **yaml**.

```
print("\n\n---")
```

```
print(yaml.dump(ourjson))
```

Сохраните и запустите свой скрипт. Вы должны увидеть следующий результат.

```
devasc@labvm:~/labs/devnet-src/parsing$ python3 parsejson.py
<результаты предыдущих шагов опущены>
---
access_token: ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN2ItYTU3
expires_in: 1209600
refresh_token: MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTEyMzQ1Njc4
refreshtokenexpires_in: 7776000

devasc@labvm:~/labs/devnet-src/parsing$
```



Примечание: Для компактности скриншота, автор закомментировал строчку print(ourjson)

Часть 4: Разбор YAML в Python

Следующая программа импортирует библиотеки **json** и **yaml**, использует PyYAML для разбора файла YAML, извлечения и печати значений данных и вывода JSON-версии файла. Он использует метод библиотеки yaml **safe_load()** для разбора потока файлов и обычные ссылки на данные Python для извлечения значений из полученной структуры данных Python. Затем он использует функцию json библиотеки **dumps()** для сериализации данных Python обратно в виде JSON.

Пример YAML для разбора — это тот же файл YAML, который вы вывели в Части 3:

```
---
access_token: ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN2ItYTU3
expires_in: 1209600
refresh_token: MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTEyMzQ1Njc4
refreshtokenexpires_in: 7776000
```

Шаг 1: Создайте скрипт для разбора данных YAML.

Откройте файл **parseyaml.py**, находящийся в каталоге ~/labs/devnet-src/parsing.

Импортируйте библиотеки **json** и **yaml**.

```
import json
```

```
import yaml
```

Используйте оператор **with**, чтобы открыть **myfile.yaml** и установить его в переменную с именем **yaml_file**. Затем используйте метод **yaml.safe_load** для загрузки файла YAML в строку, заданную переменной **ouryaml**.

```
with open('myfile.yaml','r') as yaml_file:
    ouryaml = yaml.safe_load(yaml_file)
```

Добавьте **print()** для **ouryaml**, чтобы увидеть, что теперь это словарь Python.

```
print(ouryaml)
```

Шаг 2: Запустите сценарий для печати данных YAML, а затем измените его для печати интересующих вас данных.

Сохраните и запустите свой сценарий. Вы должны увидеть следующий результат.

```
devasc@labvm:~/labs/devnet-src/parsing$ python3 parseyaml.py
{'access_token':
'ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN2ItYTU3',
'expires_in':                    1209600,                    'refresh_token':
'MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTEyMzQ1Njc4',
'refreshtokenexpires_in': 7776000}
devasc@labvm:~/labs/devnet-src/parsing$
```

```
devasc@labvm:~/labs/devnet-src/parsing$ python3 parseyaml.py
{'access_token': 'ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN
2ItYTU3', 'expires_in': 1209600, 'refresh_token': 'MDEyMzQ1Njc4OTAxMjM0NTY3
ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTEyMzQ1Njc4', 'refreshtokenexpires_in': 777600
0}
devasc@labvm:~/labs/devnet-src/parsing$
```

Добавьте операторы печати, которые отображают значение токенов и количество секунд до истечения его срока действия.

```
print("Токен доступа {}".format(ouryaml['access_token']))
print("Он истечёт через {} секунд.".format(ouryaml['expires_in']))
```

Сохраните и запустите свой сценарий. Вы должны увидеть следующий результат.

```
devasc@labvm:~/labs/devnet-src/parsing$ python3 parseyaml.py
{'access_token': 'ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN
2ItYTU3', 'expires_in': 1209600, 'refresh_token': 'MDEyMzQ1Njc4OTAxMjM0NTY3
ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTEyMzQ1Njc4', 'refreshtokenexpires_in': 777600
0}
Токен доступа ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN2ItY
TU3
Он истечёт через 1209600 секунд.
devasc@labvm:~/labs/devnet-src/parsing$
```

Шаг 3: Выведите разобранные данные YAML в формате JSON.

Добавьте **print()**, чтобы добавить две пустые строки после предыдущего вывода. Затем добавьте оператор для печати **ouryaml** в виде данных JSON с помощью метода **dumps()** библиотеки **json**. Добавьте параметр отступа, чтобы приукрасить данные JSON.

```
print("\n\n")
```

```
print(json.dumps(ouryaml, indent=4))
```

Сохраните и запустите свой скрипт. Вы должны увидеть следующий вывод. Обратите внимание, что вывод выглядит так же, как myfile.json.

```
devasc@labvm:~/labs/devnet-src/parsing$ python3 parseyaml.py
<результаты предыдущих шагов опущены>
{
  "access_token": "ZDI3MGEyYzQtNmFlNS00NDNhLWFlNzAtZGVjNjE0MGU1OGZmZWNmZDEwN2ItYTU3",
  "expires_in": 1209600,
  "refresh_token": "MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OTEyMzQ1Njc4",
  "refreshtokenexpires_in": 7776000
}
devasc@labvm:~/labs/devnet-src/parsing$
```

Содержание отчета:

1. Титульный лист
2. Скриншоты промежуточных запусков скриптов.
3. GitHub дириктория Lab-3-6-6 с рабочими скриптами **parsexml.py**, **parsejson.py**, **parseyaml.py**

Лабораторная работа №4.5.5

Изучение Rest API с помощью API Simulator и Postman

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC

Часть 2: Исследование документации API с помощью API Simulator

Часть 3: Использование Postman для выполнения вызовов API в API Simulator

Часть 4: Использование Python для добавления 100 книг в API Simulator

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

Порядок выполнения работы

Часть 1: Запуск виртуальной машины DEVASC

Если вы еще не завершили **лабораторную работу - Установка лабораторной среды виртуальной машины**, сделайте это сейчас. Если вы уже завершили эту лабораторную работу, запустите виртуальную машину **DEVASC**.

Часть 2: Исследование документации API с помощью API Simulator

Чтобы понять, как обращаться к REST API, разработчики обычно начинают с изучения документации по API. Формат запросов, ответов, заголовков и

параметров для REST API обычно документируется с помощью спецификации OpenAPI (ранее Swagger Specification).

Шаг 1: Откройте веб-браузер Chromium.

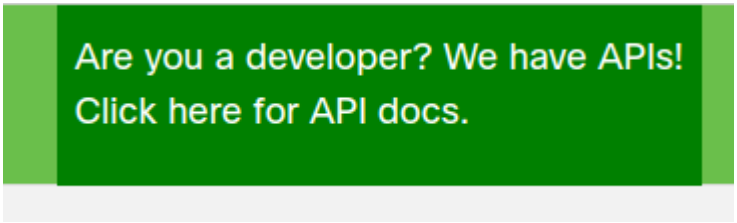
Дважды щелкните значок веб-браузера Chromium на рабочем столе.

Шаг 2: Подключитесь к веб-сайту студенческой библиотеки.

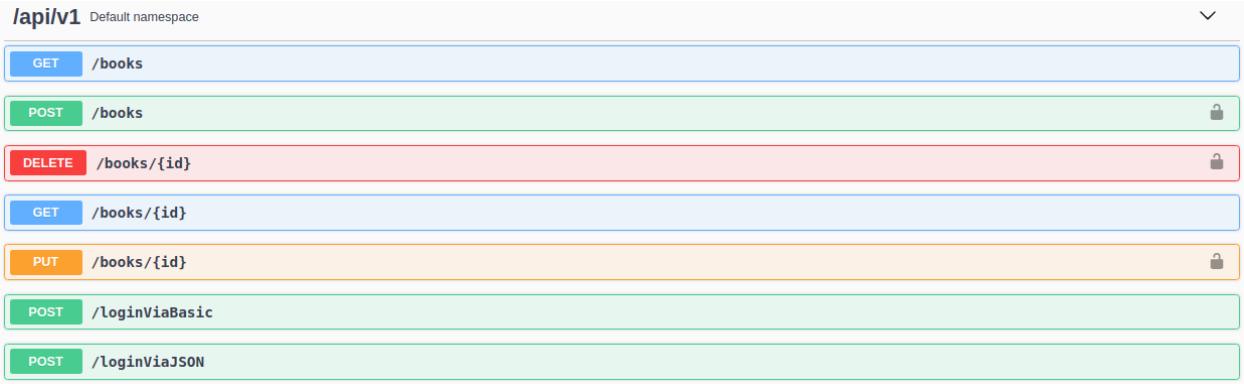
Если браузер автоматически не открывает сайт библиотеки, в адресной строке наберите: **library.demo.local** и нажмите кнопку return, чтобы перейти туда.

Шаг 3: Перейдите на страницу документации API.

На веб-сайте по умолчанию открывается вкладка **Our Books** и отображается список книг. В правом верхнем углу, где написано: **Click here for API docs**, нажмите здесь, чтобы перейти на веб-страницу документации API.



Теперь вы увидите список API в пространстве имен /api/v1 Default.



Обратите внимание на стрелку, направленную вниз в крайнем правом углу. Щелчок в любом месте на полосе /api/v1 свернет список API и повернет стрелку вправо. Щелкните еще раз на той же полосе, чтобы снова отобразить список API.

Обратите внимание на **замок** справа от нескольких API. Замок указывает на то, что для использования этих API требуется токен.

Шаг 4: Получите список книг с помощью API GET /books.

Нажмите в любом месте панели для API **GET /books**. Этот API возвращает список книг в библиотеке.

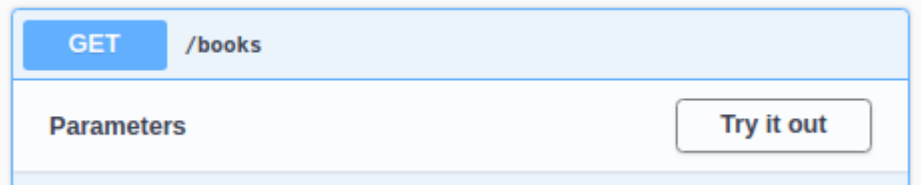
- **Parameters** — Существует несколько необязательных параметров API. Их можно использовать для фильтрации, сортировки или постраничного вывода.

- **Response content type** — Выберите application/json, чтобы увидеть список различных форматов данных, в которых можно просматривать информацию. Выберете как application/json.
- **Code** — По умолчанию в коде отображается **200**, что означает, что запрос API от сервера был успешным, как показано в описании. (Вы еще не отправляли API-запрос).

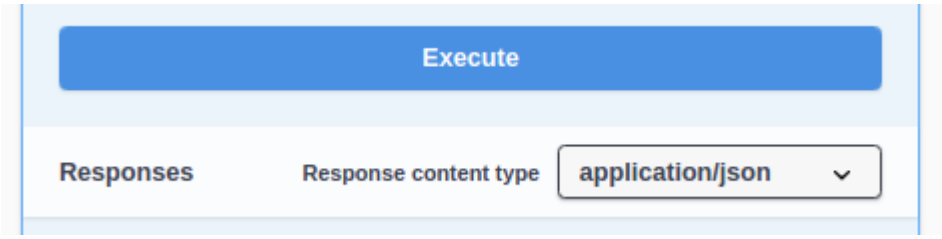
Шаг 5: Используйте функцию Try it out в документации API.

Одной из наиболее мощных возможностей спецификации OpenAPI является возможность протестировать вызов API, чтобы проверить, правильно ли вы его создали. Вы также можете просмотреть ответ, чтобы убедиться, что он соответствует вашим ожиданиям. Подобную возможность тестирования вы можете увидеть в документации по API для других организаций, которые используют эту возможность спецификации OpenAPI.

В документации API **GET /books** нажмите кнопку **Try it out**.



Обратите внимание, что теперь у вас есть возможность ввести информацию для необязательных параметров. Оставьте параметры пустыми и нажмите кнопку **Execute**.



В разделе "Ответы" вы увидите:

- **Curl**: Команда **curl**, которую вы можете использовать для получения доступа к той же информации для API /books.
- **URL запроса**: Этот URL используется в запросе API, который можно использовать для запроса той же информации с помощью curl, Postman и Python.
- **Code**: Это код ответа HTTP. 200 означает успешный вызов.
- **Response body**: Список книг в формате JSON.
- **Response header**: Информация об API, возвращенная с сервера.

В **Response body** вы увидите список книг в формате JSON:

```
[
    {
        "id": 0,
        "title": "IP Routing Fundamentals",
        "author": "Mark A. Sportack"
```

```
        },
        {
            "id": 1,
            "title": "Python for Dummies",
            "author": "Stef Maruch Aahz Maruch"
        },
        {
            "id": 2,
            "title": "Linux for Networkers",
            "author": "Cisco Systems Inc."
        },
        {
            "id": 3,
            "title": "NetAcad: 20 Years Of Online-Learning",
            "author": "Cisco Systems Inc."
        }
    ]
```


Шаг 6: Используйте команду curl в окне терминала.

API GET /books предоставляет информацию для доступа к содержимому, отображаемому в теле ответа, с помощью curl. curl — это инструмент командной строки для передачи данных на сервер или с сервера, используя любой из поддерживаемых протоколов, включая HTTP и HTTPS.

Выберите команду curl, щелкните правой кнопкой мыши и скопируйте ее в буфер обмена:

```
curl -X GET "http://library.demo.local/api/v1/books" -H "accept: application/json"
```

Откройте окно терминала. Щелкните правой кнопкой мыши и вставьте содержимое из буфера обмена в терминал, затем нажмите Enter. Обратите внимание, что вывод предоставляет ту же информацию, что и интерфейс OpenAPI библиотеки.

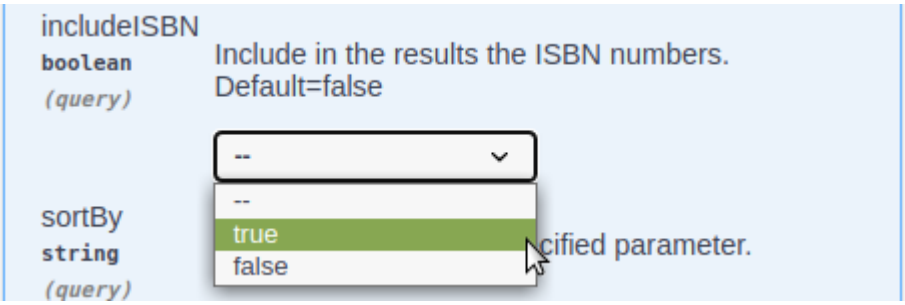
```
devasc@labvm:~$ curl -X GET "http://library.demo.local/api/v1/books" -H "accept: application/json"
```

```
devasc@labvm:~/labs/devnet-src/parsing$ curl -X GET "http://l
ibrary.demo.local/api/v1/books" -H "accept: application/json"
[
    {
        "id": 0,
        "title": "IP Routing Fundamentals",
        "author": "Mark A. Sportack"
    },
    {
        "id": 1,
        "title": "Python for Dummies",
        "author": "Stef Maruch Aahz Maruch"
    },
    {
        "id": 2,
        "title": "Linux for Networkers",
        "author": "Cisco Systems Inc."
    },
    {
        "id": 3,
        "title": "NetAcad: 20 Years Of Online-Learning",
        "author": "Cisco Systems Inc."
    }
]
devasc@labvm:~/labs/devnet-src/parsing$
```

Шаг 7: Получите список книг с их ISBN, используя API GET /books.

Вернитесь к API GET /books API веб-сайта API библиотеки.

В разделе Параметры выберите стрелку вниз рядом с параметром **includeISBN** и выберите **true**.



Нажмите **Execute**.

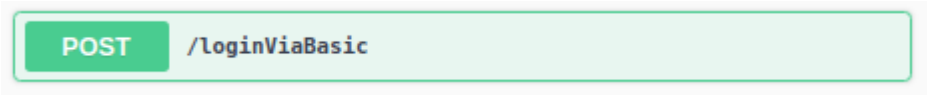
Обратите внимание на следующие изменения в разделе Responses:

- **Curl** теперь включает параметр для ISBN.
  `curl -X GET "http://library.demo.local/api/v1/books?includeISBN=true" -H "accept: application/json"`
- **URL запроса** теперь включает параметр для ISBN.
  `http://library.demo.local/api/v1/books?includeISBN=true`
- Тело ответа содержит тот же список книг, что и ранее, но теперь включает ISBN книги.

Чтобы минимизировать прокрутку, когда вы закончите работу с API, вы можете закрыть это конкретное окно API, щелкнув в любом месте строки заголовка. Теперь вы можете легко просмотреть все API.

Шаг 8: Получите токен с помощью API POST /loginViaBasic.

Щелкните API **POST /loginViaBasic**.



Обратите внимание, что параметров нет. Нажмите **Try it out**, а затем нажмите Выполнить.

В окне Вход в систему вам будет предложено ввести имя пользователя и пароль. Введите следующую информацию и нажмите **Sign in**:

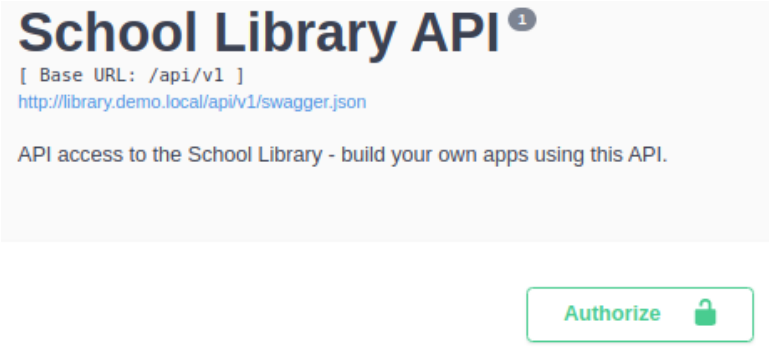
Username: cisco

Password: Cisco123!

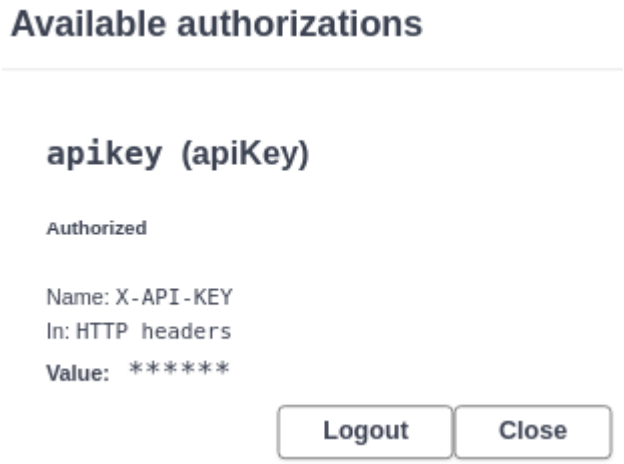
Токен будет отображен в теле ответа. Выберите информацию между кавычками, щелкните правой кнопкой мыши и скопируйте ее в буфер обмена. Ваш токен будет отличаться от показанного ниже.

```
{
  "token": "cisco|KZZzteQbC5iV3HKEzB7hCJ6qHQXen4rLGh72YJKeVfs"
}
```

Прокрутите вверх до верхней части страницы API школьной библиотеки и нажмите зеленую кнопку Authorize. Появится диалоговое окно Доступные авторизации.



Щелкните правой кнопкой мыши, вставьте маркер после Value и нажмите Authorize. Обратите внимание, что Имя — X-API-KEY. Эта информация вместе с маркером будет использоваться позже в Postman.



Закройте диалоговое окно Available authorizations и вернитесь к списку API. Обратите внимание, что блокировки некоторых API изменились. Теперь эти API доступны для использования.

Щелкните по строке для API POST /loginViaBasic, чтобы закрыть окно.

Шаг 9: Добавьте книги с помощью API POST /books.

Нажмите API POST /books.

В разделе Parameters обратите внимание на то, что полезная нагрузка является обязательной. Это означает, что данный API требует информацию для этого параметра в формате, указанном типом содержимого Parameter, который является JSON.

Нажмите кнопку Try it out.

Измените **id**, **title** и **author** с помощью информации, показанной ниже.

```
{
  "id": 4,
  "title": "Industrial Society and Its Future",
  "author": "Theodore John Kaczynski"
}
```

Нажмите **Execute**.

Проверьте успешность отправки сообщения в ответе сервера. Код 200 означает, что отправка сообщения прошла успешно. В теле ответа вы должны увидеть книгу, которую вы добавили, а также новый идентификатор. Вы также увидите обновленную информацию для curl и URL запроса.

Чтобы добавить ещё одну книгу, измените id, title и author с помощью информации, показанной ниже.

```
{
  "id": 5,
  "title": "Первая научная история войны 1812 года",
  "author": "Евгений Николаевич Понасенков"
}
```

Нажмите **Execute**.

Проверьте успешность отправки сообщения в ответе сервера. Код 200 означает, что отправка была успешной. Вы должны увидеть книгу, которую вы добавили, в теле ответа вместе с новым идентификатором. Вы также увидите обновленную информацию для curl и URL запроса.

Примечание: Если вы получили код **401**, проверьте текст тела ответа. Скорее всего, вы получили ответ **"error": "Invalid API key"**. Это происходит потому, что вы ввели не все символы для вашего API-ключа. Или, возможно, вы

добавили ненужный пробел. Вернитесь к предыдущему шагу и повторите процесс авторизации.

Щелкните по строке для API POST /books, чтобы закрыть окно.

Вы можете проверить, что книги были добавлены на страницу "Наши книги". Вернитесь на вкладку **School Library** в браузере (http://library.demo.local) и обновите страницу. Будьте осторожны, чтобы не закрыть вкладку API школьной библиотеки. Если вы это сделаете, то вам придется повторно пройти аутентификацию.

Шаг 10: Получите список книг с помощью API GET /books.

Вернитесь на вкладку API школьной библиотеки в браузере. Щелкните на API GET /books.

Нажмите кнопку **Try it out**. Если вы видите кнопку **Cancel** красного цвета, значит, вы уже находитесь в режиме тестирования.

Нажмите **Execute**.

В разделе **Server response** в **Response body** вы увидите две добавленные вами книги. Обратите внимание, что каждая из них имеет уникальный идентификатор.


```
[
  {
    "id": 0,
    "title": "IP Routing Fundamentals",
    "author": "Mark A. Sportack"
  },
  {
    "id": 1,
    "title": "Python for Dummies",
    "author": "Stef Maruch Aahz Maruch"
  },
  {
    "id": 2,
    "title": "Linux for Networkers",
    "author": "Cisco Systems Inc."
  },
  {
    "id": 3,
    "title": "NetAcad: 20 Years Of Online-Learning",
    "author": "Cisco Systems Inc."
  },
  {
    "id": 4,
    "title": "Industrial Society and Its Future",
    "author": "Theodore John Kaczynski"
  },
  {
```

```
        "id": 5,
        "title": "Первая научная история войны 1812 года",
        "author": "Евгений Николаевич Понасенков"
      }
    ]
```


Щелкните по строке API **GET /books**, чтобы закрыть окно.

Шаг 11: Внесите конкретную книгу в список, используя GET /books{id} API.

Нажмите кнопку **GET /books{id}** API. Обратите внимание, что этот API требует **id** в качестве параметра.

Справа от **Parameters** нажмите кнопку **Try it out**.

В разделе **Parameters** введите **5** для требуемого идентификатора.

Нажмите кнопку **Execute**. Обратите внимание на информацию, предоставленную Curl и Request URL.

- **Curl** — Это команда curl для выполнения той же функции с помощью curl.
- **URL запроса** — Это URL, который можно использовать для получения той же информации с помощью Postman и Python.

Убедитесь, что получение было успешным в ответе сервера. Код 200 означает, что отправка прошла успешно. В теле ответа вы увидите запрошенную книгу с идентификатором 5.


```
{
    "id": 5,
    "title": "Первая научная история войны 1812 года",
    "author": "Евгений Николаевич Понасенков"
}
```


Щелкните на строке для GET /books{id} API, чтобы закрыть окно.

Шаг 12: Удалите конкретную книгу с помощью API DELETE /books{id} API.

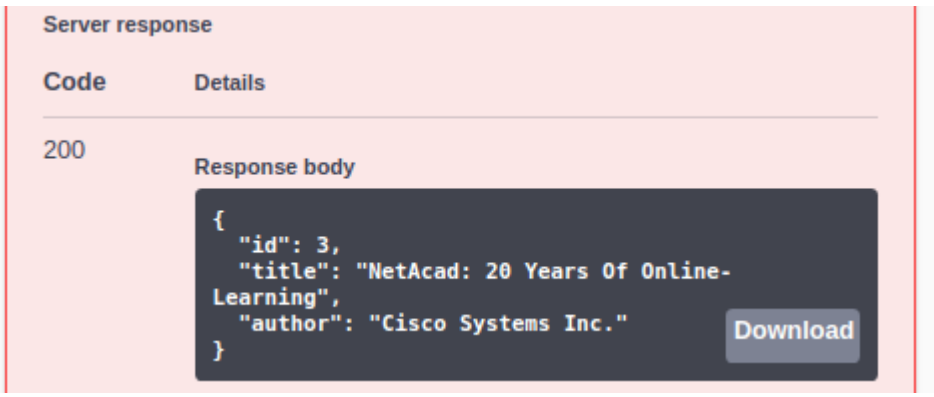
Нажмите кнопку **DELETE /books{id}** API. Обратите внимание, что этот API требует **id** в качестве параметра.

Нажмите **Try it out**.

В разделе **Parameters** введите 3.

Нажмите **Execute**

Убедитесь, что удаление прошло успешно, в ответе сервера. Код 200 означает, что удаление прошло успешно. В теле ответа вы увидите удаленную книгу с идентификатором 3.



Щелкните на строке для **DELETE /books{id}** API, чтобы закрыть окно.

Шаг 13: Получите список книг с помощью API GET /books.

Нажмите API **GET /books.**

Нажмите кнопку **Try it out**. Если вы видите кнопку **Cancel** красного цвета, значит, вы уже находитесь в режиме тестирования.

Нажмите **Execute**

В разделе Server response в теле Response больше не будет отображаться книга с **id 3**.

```
[
  {
    "id": 0,
    "title": "IP Routing Fundamentals",
    "author": "Mark A. Sportack"
  },
  {
    "id": 1,
    "title": "Python for Dummies",
    "author": "Stef Maruch Aahz Maruch"
  },
  {
    "id": 2,
    "title": "Linux for Networkers",
    "author": "Cisco Systems Inc."
  },
  {
    "id": 4,
    "title": "Industrial Society and Its Future",
    "author": "Theodore John Kaczynski"
  },
  {
    "id": 5,
    "title": "Первая научная история войны 1812 года",
    "author": "Евгений Николаевич Понасенков"
  }
]
```

Примечание: Не закрывайте вкладку API библиотеки в браузере Chromium. Вы будете использовать документацию API в следующей части.

Часть 3: Использование Postman для выполнения вызовов API в API Simulator

В этой части вы будете использовать Postman для выполнения тех же вызовов API, которые вы делали в документации по API студенческой библиотеки. Postman является полезным инструментом, когда веб-сайт разработчика API недоступен, и при этом предоставляет возможность легко сохранять, организовывать и повторно использовать API.

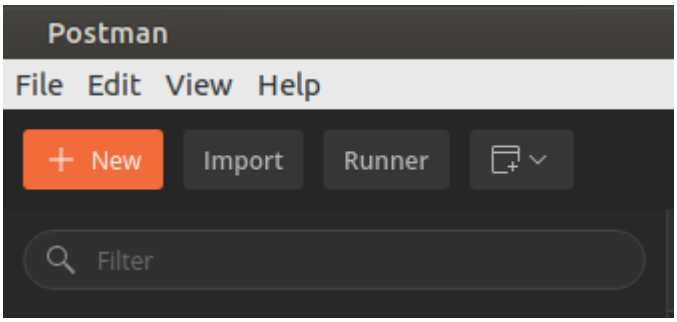
Шаг 1: Откройте Postman.

Дважды щелкните значок Postman на рабочем столе. Обычно вы входите в Postman.

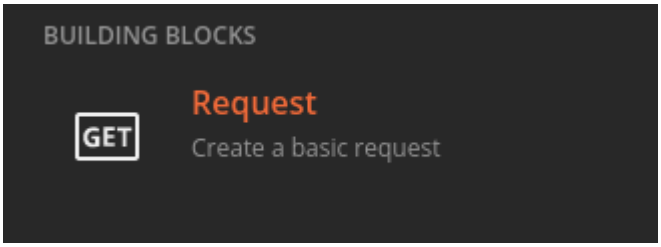
Шаг 2: Получите список книг, используя API GET /books.

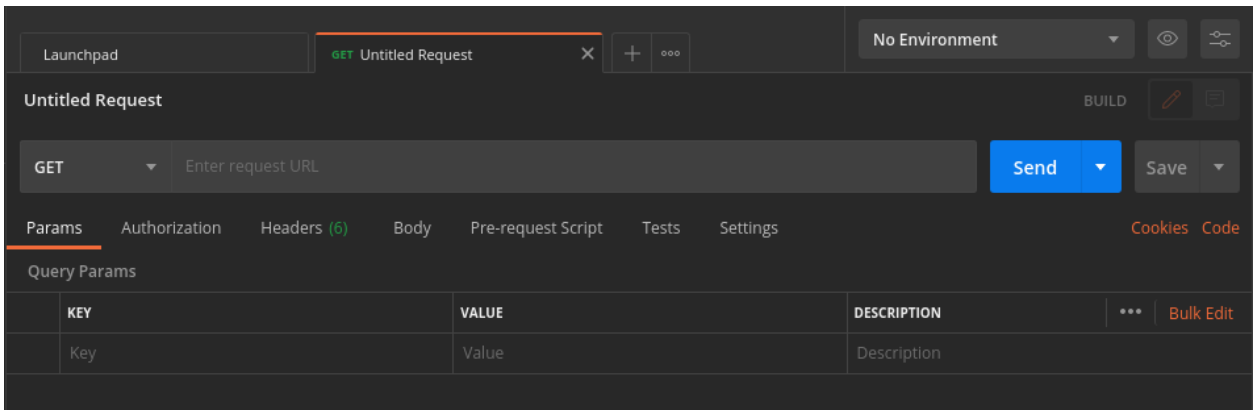
Примечание: Не обязательно заводить учетную запись и входить в Postman.

В главном окне рядом с вкладкой Launchpad нажмите на значок плюса "+", чтобы создать запрос без названия. По умолчанию это будет GET-запрос.



Нажмите стрелку вниз рядом с GET, чтобы просмотреть различные операции API, включая GET, POST и DELETE. Оставьте выбор на GET. Нажмите стрелку вверх рядом с GET, чтобы закрыть список.





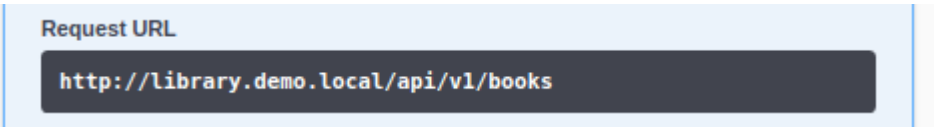
Нажмите стрелку вниз рядом с **GET**, чтобы просмотреть различные операции API, включая **GET**, **POST** и **DELETE**. Оставьте выбор на **GET**. Нажмите стрелку вверх рядом с **GET**, чтобы закрыть список.

Введите URL-адрес запроса.

Вернитесь на вкладку **School Library API** в Chromium и, если необходимо, разверните API **GET /books**.

В разделе Request URL выберите, щелкните правой кнопкой мыши и скопируйте URL в буфер обмена:

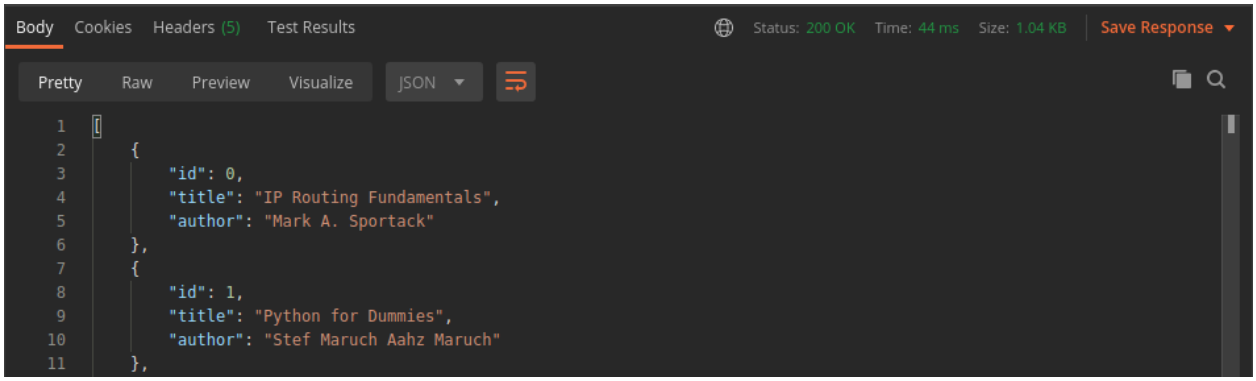
```
http://library.demo.local/api/v1/books
```



Вернитесь в **Postman** и вставьте URL рядом с GET, где написано Enter request URL.

Примечание: Если при вставке добавляется строка под URL-адресом, удалите лишнюю строку.

Нажмите кнопку **Send**. Чтобы убедиться, что запрос API прошел успешно, вы увидите ответ, в котором зеленым цветом будет указан код состояния 200 OK. Внизу раскройте раздела Body, чтобы увидеть ответ. Обратите внимание, что по умолчанию используется **Pretty** и **json**.



Шаг 3: Получите токен с помощью API POST /loginViaBasic.

В главном окне нажмите значок плюса "+", чтобы создать новый Untitled Request.

Вернитесь на вкладку **School Library API** в Chromium и, если необходимо, разверните API **POST /loginViaBasic**.

В разделе Request URL выберите, щелкните правой кнопкой мыши и скопируйте URL в буфер обмена:

```
http://library.demo.local/api/v1/loginViaBasic
```

Примечание: Если URL-адрес запроса больше не отображается, то, вероятно, вы закрыли и снова открыли страницу документации API школьной библиотеки и больше не авторизованы. Нажмите **Try it out**, затем **Execute**, а затем повторно авторизуйтесь с именем пользователя **cisco** и паролем **Cisco123!**

Вернитесь в Postman и вставьте URL рядом с POST, где написано " Enter request URL".

Примечание: Если при вставке добавляется строка под URL, удалите лишнюю строку.

Нажмите кнопку Authorization. В этой области заполните следующее:

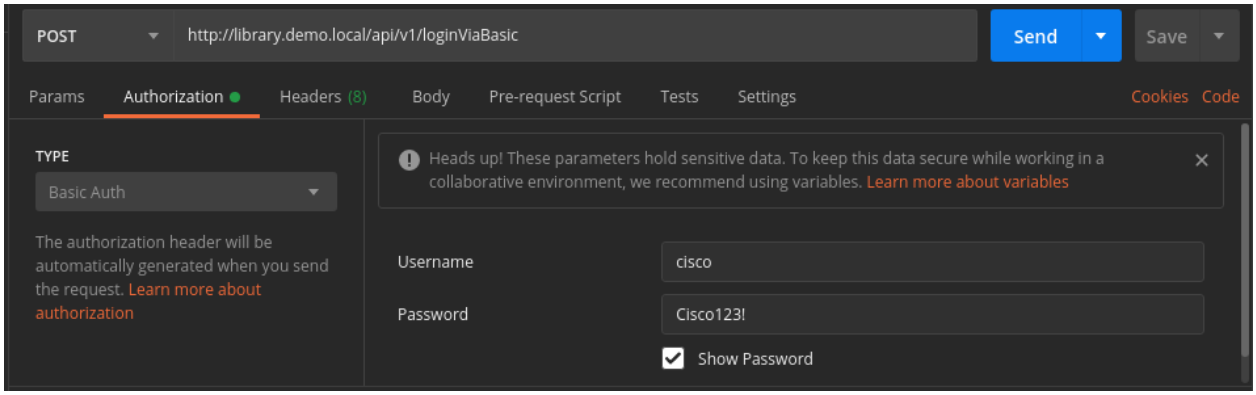
В выпадающем списке **Type** выберите Basic Auth.

В полях Username и Password заполните следующие поля:

Username: cisco

Password: Cisco123!

Нажмите **Send**



При необходимости прокрутите вниз до раздела "Body", чтобы увидеть свой новый токен. Он может отличаться от того, что показан здесь.

```
{
    "token": "cisco|9DXC08dbBSkHIzIY5s4NJBRLn97xtBGmXEnl8GP2e_4"
}
```

Шаг 4: Добавьте книгу с помощью API POST /books.

Теперь вы добавите книгу NetAcad: 20 Years Of Online-Learning, которую вы удалили в части 2, при использовании функции Try it out в документации API школьной библиотеки.

В главном окне нажмите значок плюса "+", чтобы создать Untitled Request.

Щелкните стрелку вниз рядом с GET и выберите POST

Введите URL-адрес запроса.

Вернитесь на вкладку School Library API в Chromium и разверните API **POST /books**.

В разделе **Request URL** выберите, щелкните правой кнопкой мыши и скопируйте URL в буфер обмена:

```
http://library.demo.local/api/v1/books
```

Вернитесь в **Postman** и вставьте URL рядом с **POST**, где написано "Enter request URL".

Примечание: Если при вставке добавляется строка под URL-адресом, удалите лишнюю строку.

Нажмите кнопку Authorization. В этой области заполните следующее:

В раскрывающемся списке Тип выберите **API Key**.

В поле Ключ введите **X-API-KEY**.

Вернитесь на вкладку Post в Postman и скопируйте Token, который вы получили в Шаге 3. Не забудьте включить все, что находится в кавычках. Ваш токен будет отличаться от того, что показан здесь.

```
{
    "token": "cisco|9DXC08dbBSkHIzIY5s4NJBRLn97xtBGmXEnl8GP2e_4"
}
```

Вернитесь на вторую вкладку Post в Postman. Вставьте токен в поле **Value**

В той же строке с вкладкой Authorization нажмите кнопку Body. В этом разделе вы сможете выбрать формат вводимых данных.

- Нажмите кнопку raw
- Нажмите Текст и измените этот параметр на JSON.

В области ввода вы увидите цифру 1, означающую "строка 1". Введите следующий объект JSON.

```
{
  "id": 3,
  "title": "NetAcad: 20 Years Of Online-Learning",
  "author": "Cisco Systems Inc",
  "isbn": "000-0000001123"
}
```

Нажмите **Send**.

Чтобы убедиться, что запрос API прошел успешно, теперь вы увидите ответ, в котором зеленым цветом будет указан код состояния 200 OK.

Шаг 5: Проверьте книгу с помощью API Get /books.

Вернитесь на первую вкладку GET. Как видите, Postman позволяет легко переключаться между различными вызовами API.

Нажмите кнопку **Send**.

Чтобы убедиться, что запрос API прошел успешно, теперь вы увидите ответ, в котором зеленым цветом будет указан код состояния 200 OK.

Нажмите кнопку Body, чтобы просмотреть ответ. Обратите внимание, что по умолчанию используется Pretty и json.


```
[
    {
        "id": 0,
        "title": "IP Routing Fundamentals",
        "author": "Mark A. Sportack"
    },
    {
        "id": 1,
        "title": "Python for Dummies",
        "author": "Stef Maruch Aahz Maruch"
    },
    {
        "id": 2,
        "title": "Linux for Networkers",
        "author": "Cisco Systems Inc."
    },
    {
        "id": 3,
        "title": "NetAcad: 20 Years Of Online-Learning",
        "author": "Cisco Systems Inc."
    },
    {
        "id": 4,
        "title": "Industrial Society and Its Future",
        "author": "Theodore John Kaczynski"
    }
    {
        "id": 5,
        "title": "Первая научная история войны 1812 года",
        "author": "Евгений Николаевич Понасенков"
    },
]
```


Шаг 6: Используйте дополнительные параметры с API Get /books.

Перейдите на веб-сайт School Library API. Прокрутите страницу вверх до API GET /books и при необходимости разверните ее.

Обратите внимание на доступные параметры:

- **includeISBN**: Включает в вывод номера ISBN. По умолчанию=false

- **sortBy**: Сортировка результатов с помощью указанного параметра. По умолчанию=id
- **author**: Возвращает только книги указанного автора.
- **page**: Используется для указания номера страницы.

Нажмите кнопку **Try it out**.

Под параметрами:

- Нажмите **includeISBN** и выберите **true**
- Нажмите кнопку **sortBy** и выберите **author**

Нажмите **Execute**.

В теле ответа вы увидите список книг, теперь отсортированный по авторам и включающий ISBN.


```
[
  {
    "id": 3,
    "title": "NetAcad: 20 Years Of Online-Learning",
    "author": "Cisco Systems Inc",
    "isbn": "000-0000001123"
  },
  {
    "id": 2,
    "title": "Linux for Networkers",
    "author": "Cisco Systems Inc.",
    "isbn": "000-0000000123"
  },
  {
    "id": 0,
    "title": "IP Routing Fundamentals",
    "author": "Mark A. Sportack",
    "isbn": "978-1578700714"
  },
  {
    "id": 1,
    "title": "Python for Dummies",
    "author": "Stef Maruch Aahz Maruch",
    "isbn": "978-0471778646"
  },
  {
    "id": 4,
    "title": "Industrial Society and Its Future",
    "author": "Theodore John Kaczynski"
  },
  {
    "id": 5,
    "title": "Первая научная история войны 1812 года",
    "author": "Евгений Николаевич Понасенков"
  }
]
```


Обратите внимание, что URL запроса теперь включает параметры. Вы увидите это снова в Postman.

```
http://library.demo.local/api/v1/books?includeISBN=true&sortBy=author
```

Вернитесь в Postman и перейдите на первую вкладку API, GET http://library.demo.local/api/v1/books. Теперь вы включите некоторые параметры с сайта API школьной библиотеки.

Нажмите Params. Вы увидите под Query Params поля ввода для KEY и VALUE. Введите следующую информацию:

- В поле KEY введите **includeISBN**, а в поле **Value** введите **true**.

  Обратите внимание, что слева от значения будет автоматически включена галочка и добавлена новая строка.
- В поле KEY введите **sortBy** и в поле **Value** введите **author**

Обратите внимание, что при вводе этих параметров запроса, он обновил исходный URL рядом с GET. Это тот же URL запроса, который вы видели на веб-сайте API школьной библиотеки для этого же вызова API. Это URL, который Postman будет использовать с этими параметрами запроса при выполнении вызова API.

```
http://library.demo.local/api/v1/books?includeISBN=true&sortBy=author
```

Нажмите **Send**.

Обратите внимание, что теперь в теле отображается тот же список книг, отсортированный по авторам и включающий ISBN, который вы видели на сайте School Library API.


```
[
  {
    "id": 3,
    "title": "NetAcad: 20 Years Of Online-Learning",
    "author": "Cisco Systems Inc",
    "isbn": "000-0000001123"
  },
  {
    "id": 2,
    "title": "Linux for Networkers",
    "author": "Cisco Systems Inc.",
    "isbn": "000-0000000123"
  },
  {
    "id": 0,
    "title": "IP Routing Fundamentals",
    "author": "Mark A. Sportack",
    "isbn": "978-1578700714"
  },
  {
    "id": 1,
    "title": "Python for Dummies",
    "author": "Stef Maruch Aahz Maruch",
    "isbn": "978-0471778646"
  },
  {
    "id": 4,
```

```
    "title": "Industrial Society and Its Future",
    "author": "Theodore John Kaczynski"
  },
  {
    "id": 5,
    "title": "Первая научная история войны 1812 года",
    "author": "Евгений Николаевич Понасенков"
  }
]
```


Часть 4: Использование Python для добавления 100 книг в API Simulator

Вы можете использовать инструмент OpenAPI Specification Try It или Postman, чтобы добавить столько книг, сколько хотите. Однако вам придется добавлять их по одной за раз. Лучшим решением было бы написать программу для добавления книг. В этой части вы смоделируете процесс добавления 100 книг с помощью библиотеки Python **faker**.

Шаг 1: Откройте Visual Studio (VS) Code и перейдите в каталог school-library.

Откройте VS Code из кнопки меню или двойным щелчком по значку на рабочем столе.
Click File > Open Folder..., перейдите в каталог labs/devnet-src/school-library folder, и нажмите OK.

Шаг 2: Исследуйте библиотеки, используемые программой add100RandomBooks.py.

В панели VS Code EXPLORER слева щелкните файл add100RandomBooks.py, чтобы открыть его, если это необходимо.

В верхней части обратите внимание на "shebang", который устанавливает интерпретатор на Python 3, а затем на три импортируемые библиотеки.

```
#!/usr/bin/env python3

import requests
import json
from faker import Faker
```

Здесь и далее мы будем использовать библиотеку запросов Python — **requests**, которая необходима для выполнения запросов API с помощью GET, POST, DELETE и других методов HTTP.

Faker — это библиотека Python, которая генерирует для вас "фальшивые" данные. Эта программа использует библиотеку Python faker для генерации случайных названий книг, авторов и ISBN.

Откройте окно терминала и запустите Python 3.
Из **faker** импортируйте модуль **Faker()**.

Назначьте модуль **Faker()** на **fake**.
Чтобы увидеть все методы, введите **fake**. а затем дважды нажмите клавишу табуляции. Обратите внимание на метод **fake.name()**, который вы будете использовать в следующем шаге. В следующем шаге и далее в этой лабораторной работе вы также будете использовать три выделенных метода (перед которыми стоит fake.): catch_phrase(), isbn13() и name().

```
devasc@labvm:~/labs/devnet-src/school-library$ python3
Python 3.8.2 (default, Apr 27 2020, 15:53:34)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from faker import Faker
>>> fake = Faker()
>>> fake.<press Tab key twice - be sure you include the period>
Display all 252 possibilities? (y or n)
fake.add_provider(              fake.future_datetime(           fake.pyfloat(
fake.address(                   fake.generator_attrs            fake.pyint(
fake.am_pm(                     fake.get_formatter(             fake.pyiterable(
<вывод опущен>
fake.catch_phrase(              fake.ipv4_public(               fake.random_element(
fake.chrome(                    fake.isbn10(                    fake.random_int(
fake.city(                      fake.isbn13(                    fake.random_letter(
<вывод опущен>
fake.date_time_ad(              fake.msisdn(                    fake.texts(
fake.date_time_between(         fake.name(                      fake.time(
<вывод опущен>
fake.future_date(               fake.pydict(                    fake.zipcode_plus4(
>>>
```

Шаг 3: Потренируйтесь генерировать случайные данные с помощью библиотеки faker.

Введите следующую команду для создания вымышленного имени. При каждом выполнении команды будет создаваться разное вымышленное имя.

```
>>> print('Меня зовут {}.'.format(fake.name()))
Меня зовут Katherine Ross.
>>>
```

Используя три выделенных сообщения из Шага 2, введите команду, которая выведет следующий фальшивый вывод.

```
Меня зовут Gary Castaneda и я автор "Organic incremental neural-net" (ISBN 978-0-669-01935-3).
>>>
```

Пердполагаемая команда:

```
>>> print('Меня зовут {} и я автор "{}" (ISBN {}).'.format(fake.name(),fake.catch_phrase(),fake.isbn13()))
```

Позже в программе используется цикл для итерации этих трех методов для создания записей для школьной библиотеки. Введите следующее, чтобы сгенерировать 10 случайных имен. После "..." вам нужно будет второй раз нажать клавишу return.

```
>>> for i in range(10):
...     print(fake.name())
...
Kevin Moyer
Mr. Christopher Green MD
Spencer Jensen
Whitney Guzman
Nicole Scott
Tammy Lewis
Craig Edwards
Michael Diaz
Ryan Mccoy
Terry Rocha
>>>
```

Выйдите из интерпретатора Python после завершения работы с библиотекой faker.

```
>>> quit()
devasc@labvm:~/labs/devnet-src/school-library$
```

Шаг 4: Рассмотрите переменные функции.

Две функции в программе используют три переменные для получения токена авторизации от службы School Library API.

```
APIHOST = "http://library.demo.local"
LOGIN = "cisco"
PASSWORD = "Cisco123!"
```

Шаг 5: Рассмотрите функцию getAuthToken.

Функция getAuthToken использует три переменные для построения запроса. Переменная r вызывает метод POST для API loginViaBasic и сохраняет значение токена в случае успешного вызова. Обратите внимание на использование f-строки для построения URL запроса.

```
def getAuthToken():
    authCreds = (LOGIN, PASSWORD)
    r = requests.post(
        f"{APIHOST}/api/v1/loginViaBasic",
        auth = authCreds
    )
```

Если вызов не удался (Статус код HTTP не равен 200), возникает исключение и выводится в окно терминала. Снова обратите внимание на использование f-строки для построения сообщения об исключении. Вы можете проверить код исключения, изменив одну из переменных в Шаге 4.

```
if r.status_code == 200:
        return r.json()["token"]
    else:
        raise Exception(f"Статус код {r.status_code} и сообщение {r.text}, при
попытке авторизации.")
```

Шаг 6: Просмотрите функцию addBook.

Подобно функции addAuthToken, функция addBook использует три переменные, показанные в шаге 4, для построения запроса. Переменная r

вызывает метод POST для API books. Данные поступают из переменной book, которая указывается в заключительной части программы.

```
def addBook(book, apiKey):
    r = requests.post(
        f"{APIHOST}/api/v1/books",
        headers = {
            "Content-type": "application/json",
            "X-API-Key": apiKey
            },
        data = json.dumps(book)
    )
```

Если вызов не удался, возникает исключение и выводится в окно терминала. Вы можете проверить это, изменив одну из переменных в Шаге 4.

```
if r.status_code == 200:
        print(f"Книга {book} добавлена.")
    else:
        raise Exception(f"Код ошибки {r.status_code} и сообщение {r.text}, при
попытке добавления книги {book}.")
```

Шаг 7: Просмотрите код, вызывающий эти две функции.

Вызывается функция addAuthToken, и результаты сохраняются в переменной apiKey.

```
apiKey = getAuthToken()
```

Модуль Faker() устанавливается на переменную с именем fake. Затем цикл for выполняет итерацию 100 раз. Переменная i используется позже в цикле для установки значения ключа id для каждой новой книги от 4 до 104 включительно.

Примечание: Если вы хотите сохранить две предыдущие книги, добавленные ранее в этой лаборатории, измените диапазон на (6, 106).

```
fake = Faker()
for i in range(4, 104):
```

Далее, три переменные хранят значения методов, вызванных из модуля **Faker(): catch_phrase(), name()** и **isbn13()**.

```
    fakeTitle = fake.catch_phrase()
    fakeAuthor = fake.name()
    fakeISBN = fake.isbn13()
```

Напомним, что для параметра полезной нагрузки API books требуется JSON в следующем формате:

```
{
  "id": 0,
  "title": "string",
  "author": "string"
}
```

Переменная book строится с использованием трех необходимых ключей для параметра полезной нагрузки и значений из трех поддельных переменных.

```
    book = {"id":i, "title": fakeTitle, "author": fakeAuthor, "isbn": fakeISBN}
```

Наконец, вызывается функция addBook, передающая переменные book и apiKey. Поскольку функция addBook является частью цикла, она будет вызвана 101 раз, по одному разу для идентификаторов книг с 4 по 105.

Шаг 8: Запустите и проверьте программу add100RandomBooks.py.
Введите команду Python 3 для запуска программы add100RandomBooks.py. Вы должны получить результат, похожий на следующий, хотя названия ваших книг и номера ISBN будут иметь другие поддельные значения.

```
devasc@labvm:~/labs/devnet-src/school-library$ python3
add100RandomBooks.py
Book {'id': 4, 'title': 'Assimilated client-server frame', 'author': 'Chelsea
Mitchell', 'isbn': '978-0-411-83123-3'} added.
Book {'id': 5, 'title': 'Adaptive tangible conglomeration', 'author': 'Edward
Ryan', 'isbn': '978-1-64406-014-8'} added.
<вывод опущен>
Book {'id': 103, 'title': 'Fundamental uniform data-warehouse', 'author': 'Dennis
David', 'isbn': '978-1-68465-896-1'} added.
Book {'id': 104, 'title': 'Organic 4thgeneration functionalities', 'author':
'Nicole Gilbert', 'isbn': '978-0-13-176202-2'} added.
devasc@labvm:~/labs/devnet-src/school-library$
```

Вернитесь в браузер Chromium и обновите веб-страницу http://library.demo.local/. Теперь вы должны увидеть добавленные 100 новых книг.

Примечание: Если вы попали на страницу документации API вместо главной страницы (http://library.demo.local/api/v1/docs) и используете функцию Try It out, вы получите список только первых 10 книг. Вы можете ввести значение от 2 до 10 в параметр страницы, чтобы увидеть остальные книги.

Как бы вы добавили еще 100 книг?

- Предполагаемый ответ: Изменил диапазон for от 104 до 204.

Лабораторная работа №4.9.2

Интеграция REST API и Python приложения

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC

Часть 2: Демонстрация приложения MapQuest Directions

Часть 3: Получение API-ключа MapQuest

Часть 4: Создание базового приложения MapQuest Direction

Часть 5: Обновление приложения MapQuest Directions с помощью дополнительных функций

Часть 6: Тестирование полной функциональности приложения

Необходимые ресурсы:

– 1 ПК

– Virtual Box или VMWare
– DEVASC виртуальная машина

Порядок выполнения работы

Часть 1: Запуск виртуальной машины DEVASC

Если вы еще не завершили **лабораторную работу - Установка лабораторной среды виртуальной машины**, сделайте это сейчас. Если вы уже завершили эту лабораторную работу, запустите виртуальную машину **DEVASC**.

Часть 2: Демонстрация приложения MapQuest Directions

В этой лабораторной работе мы создадим сценарий для MapQuest Directions Application шаг за шагом.

Приложение запрашивает начальное местоположение и пункт назначения. Затем оно запрашивает JSON-данные из MapQuest Directions API, анализирует их и отображает полезную информацию.

```
>>>
Начальное местоположение: Washington
Место назначения: Baltimore
URL:
https://www.mapquestapi.com/directions/v2/route?key=ваш_api_ключ&from=Washingto
n&to=Baltimore
API Статус: 0 = Успешный вызов маршрута.

Маршрут из Washington в Baltimore
Продолжительность поездки:   00:49:19
Километры:      61.32
Использованное топливо (Литр.): 6.24
=============================================
Start out going north on 6th St/US-50 E/US-1 N toward Pennsylvania Ave/US-1 Alt
N. (1.28 км)
Turn right onto New York Ave/US-50 E. Continue to follow US-50 E (Crossing into
Maryland). (7.51 км)
Take the Balt-Wash Parkway exit on the left toward Baltimore. (0.88 км)
Merge onto MD-295 N. (50.38 км)
Turn right onto W Pratt St. (0.86 км)
Turn left onto S Calvert St/MD-2. (0.43 км)
Welcome to BALTIMORE, MD. (0.00 км)
=============================================

Начальное местоположение: quit
>>>
```

Часть 3: Получение API-ключа MapQuest

Перед созданием приложения необходимо выполнить следующие шаги, чтобы получить ключ MapQuest API.

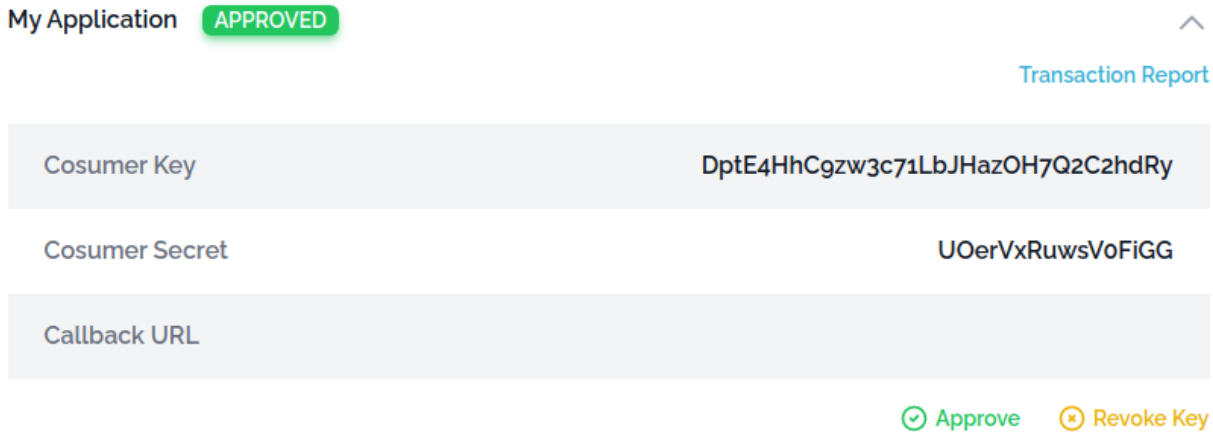
Перейдите по адресу: https://developer.mapquest.com/.

Нажмите кнопку **Sign Up** в верхней части страницы.

Заполните форму для создания новой учетной записи. Для компании введите **SPbSUT**

После нажатия кнопки **Sign Me Up** вы будете перенаправлены на страницу Manage Profile, нажмите **Manage Keys** в списке опций слева.

Разверните **My Application** и нажмите **Approve**.



Скопируйте **Consumer Key** в текстовый файл для дальнейшего использования. Этот ключ, будет использовать до конца этой лабораторной работы.

Часть 4: Создание базового приложения MapQuest Direction

В этой части вы создадите сценарий Python для отправки URL запроса к MapQuest directions API. Затем вы протестируете свой вызов API. На протяжении всей этой лабораторной работы вы будете создавать свой скрипт по частям, каждый раз сохраняя файл с новым именем. Это поможет в изучении частей приложения, а также даст вам серию сценариев, к которым вы сможете вернуться, если столкнетесь с какими-либо проблемами в текущей версии вашего приложения.

Шаг 1: Создайте новый файл в VS Code.

Откройте VS Code.

Select File > Open Folder...

Перейдите в каталог ~/labs/devnet-src/mapquest и нажмите OK. Этот каталог в настоящее время пуст, и в нем вы будете хранить каждую итерацию вашего приложения.

Select File > New File.

Select File > Save as..., назовите файл **mapquest_parse-json_1.py** и нажмите Save.

Шаг 2: Импорт модулей для приложения.

Чтобы начать сценарий разбора данных JSON, вам нужно импортировать два модуля из библиотеки Python: **requests** и **urllib.parse**. Модуль **requests** предоставляет функции для получения данных JSON из URL. Модуль **urllib.parse** предоставляет множество функций, которые позволят вам

разбирать и манипулировать данными JSON, полученными в результате запроса к URL.

Добавьте следующие импорты в верхней части вашего сценария.

```
import urllib.parse
import requests
```

Выберите **Terminal** > **New Terminal**, чтобы открыть терминал внутри VS Code.

Сохраните и запустите свой сценарий. Вы не должны получить никаких ошибок. Для проверки функциональности кода, вам придётся часто сохранять и запускать свои скрипты.

```
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-
json_1.py
devasc@labvm:~/labs/devnet-src/mapquest$
```

Шаг 3: Создайте URL для запроса к API направлений MapQuest.

Первым шагом в создании запроса API является создание URL, который ваше приложение будет использовать для вызова. Первоначально URL будет представлять собой комбинацию следующих переменных:

- **main_api** — основной URL, к которому вы обращаетесь
- **orig** — параметр для точки отправления
- **dest** — параметр для пункта назначения
- **key** — ключ API MapQuest, который вы получили с сайта разработчика.

Создайте переменные для построения URL, который будет отправлен в запросе. В следующем коде замените **ваш_api_ключ** на **Consumer Key**, который вы скопировали из своего аккаунта разработчика MapQuest.

```
main_api = "https://www.mapquestapi.com/directions/v2/route?"
orig = "Washington, D.C."
dest = "Baltimore, Md"
key = "ваш_api_ключ"
```

Объедините четыре переменные main_api, orig, dest и key, чтобы отформатировать запрошенный URL. Используйте метод urlencode для правильного форматирования значения адреса. Эта функция формирует часть параметров URL и преобразует возможные специальные символы в значении адреса в допустимые (например, пробел в "+", а запятую в "%2C").

```
url = main_api + urllib.parse.urlencode({"key":key, "from":orig, "to":dest})
```

Создайте переменную для хранения ответа запрашиваемого URL и выведите возвращаемые данные в формате JSON. В переменной **json_data** хранится представление словаря Python's Dictionary ответа в формате json, полученного методом get модуля requests. Метод requests.get выполнит вызов API MapQuest API. Оператор print будет временно использоваться для проверки возвращаемых данных. Вы замените этот оператор print на более сложные варианты отображения данных позже в лабораторной работе.

```
json_data = requests.get(url).json()
print(json_data)
```

Ваш окончательный код должен выглядеть следующим образом, но с другим значением для ключа.

```
import urllib.parse
import requests

main_api = "https://www.mapquestapi.com/directions/v2/route?"
orig = "Washington, D.C."
dest = "Baltimore, Md"
key = "DptE4HhC9zw3c71LbJHazOH7Q2C2hdRy" #Замените на свой ключ MapQuest

url = main_api + urllib.parse.urlencode({"key":key, "from":orig, "to":dest})

json_data = requests.get(url).json()
print(json_data)
```

Шаг 4: Протестируйте URL-запрос.

Сохраните и запустите сценарий mapquest_parse-json_1.py и убедитесь, что он работает.

При необходимости устраните неполадки в коде. Хотя ваш результат может немного отличаться, вы должны получить ответ в формате JSON, похожий на следующий. Обратите внимание, что выходной сигнал представляет собой словарь с двумя парами ключ/значение. Значением для ключевого маршрута является другой словарь, который включает дополнительные словари и списки. Ключ info включает пару ключ/значение statuscode, которую вы будете использовать позже в лабораторной работе.

```
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-
json_1.py
{'route': {'hasTollRoad': False, 'hasBridge': True, 'boundingBox': {'lr': {'lng':
-76.612137, 'lat': 38.892063}, 'ul': {'lng': -77.019913, 'lat': 39.290443}},
'distance': 38.089, 'hasTimedRestriction': False, 'hasTunnel': False,
'hasHighway': True, 'computedWaypoints': [], 'routeError': {'errorCode': -400,
'message': ''}, 'formattedTime': '00:49:29', 'sessionId': '5eadfc17-00ee-5f21-
02b4-1a24-0647e6e69816', 'hasAccessRestriction': False, 'realTime': 2915,
'hasSeasonalClosure': False, 'hasCountryCross': False, 'fuelUsed': 1.65, 'legs':
[{'hasTollRoad': False, 'hasBridge': True, 'destNarrative': 'Proceed to
BALTIMORE, MD.', 'distance': 38.089, 'hasTimedRestriction': False, 'hasTunnel':
False, 'hasHighway': True, 'index': 0, 'formattedTime': '00:49:29', 'origIndex':
-1, 'hasAccessRestriction': False, 'hasSeasonalClosure': False,
'hasCountryCross': False, 'roadGradeStrategy': [[]], 'destIndex': 3, 'time':
2969, 'hasUnpaved': False, 'origNarrative': '', 'maneuvers': [{'distance': 0.792,
'streets': ['6th St', 'US-50 E', 'US-1 N'], 'narrative': 'Start out going north
on 6th St/US-50 E/US-1 N toward Pennsylvania Ave/US-1 Alt N.', 'turnType': 0,
'startPoint': {'lng': -77.019913, 'lat': 38.892063}, 'index': 0, 'formattedTime':
'00:02:06', 'directionName': 'North', 'maneuverNotes': [], 'linkIds': [],
'signs': [{'extraText': '', 'text': '50', 'type': 2, 'url':
'http://icons.mqcdn.com/icons/rs2.png?n=50&d=EAST', 'direction': 8},
{'extraText': '', 'text': '1', 'type': 2, 'url':
'http://icons.mqcdn.com/icons/rs2.png?n=1&d=NORTH',
<<<<<                    >>>>>
      вывод опущен
<<<<<                    >>>>>
'geocodeQuality': 'CITY', 'adminArea1Type': 'Country', 'adminArea3Type': 'State',
'latLng': {'lng': -76.61233, 'lat': 39.29044}}], 'time': 2969, 'hasUnpaved':
False, 'locationSequence': [0, 1], 'hasFerry': False}, 'info': {'statuscode': 0,
'copyright': {'imageAltText': '© 2019 MapQuest, Inc.', 'imageUrl':
'http://api.mqcdn.com/res/mqlogo.gif', 'text': '© 2019 MapQuest, Inc.'},
'messages': []}}
```

```
devasc@labvm:~/labs/devnet-src/mapquest$
```

```
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-json_1.py
{'route': {'hasTollRoad': False, 'hasBridge': True, 'boundingBox': {'lr': {
'lng': -76.612137, 'lat': 38.892063}, 'ul': {'lng': -77.019913, 'lat': 39.2
90443}}, 'distance': 38.089, 'hasTimedRestriction': False, 'hasTunnel': Fal
se, 'hasHighway': True, 'computedWaypoints': [], 'routeError': {'errorCode'
: -400, 'message': ''}, 'formattedTime': '00:49:17', 'sessionId': '62700d1b
-0230-6750-02b4-3533-12aa46ed0e55', 'hasAccessRestriction': False, 'realTim
e': 3440, 'hasSeasonalClosure': False, 'hasCountryCross': False, 'fuelUsed'
: 1.65, 'legs': [{'hasTollRoad': False, 'hasBridge': True, 'destNarrative':
 'Proceed to BALTIMORE, MD.', 'distance': 38.089, 'hasTimedRestriction': Fa
```

Измените переменные **orig** и **dest**. Повторно запустите сценарий, чтобы получить другие результаты. Чтобы получить желаемые результаты, лучше всего указывать и город, и штат для городов в США. При указании городов в других странах обычно используется либо английское название города и страны, либо родное название. Например:

```
orig = "Saint-Petersburg, Russia"
dest = "Rome, Italy"
```

или

```
orig = "Moscow, Russia"
dest = "Roma, Italia"
```

Шаг 5: Выведите URL и проверьте статус JSON-запроса.

Теперь, когда вы знаете, что JSON-запрос работает, вы можете добавить еще немного функциональности в приложение.

Сохраните свой сценарий под именем **mapquest_parse-json_2.py**.

Удалите оператор **print(json_data)**, поскольку вам больше не нужно проверять правильность форматирования запроса.

Добавьте приведенные ниже утверждения, которые будут выполнять следующие действия:

- Выведите построенный URL, чтобы пользователь мог увидеть точный запрос, сделанный приложением.
- Разберите данные JSON, чтобы получить значение statuscode.
- Запустите цикл if, который проверяет успешность вызова, на что указывает возвращаемое значение 0. Добавьте оператор print для отображения значения statuscode и его значения. \n добавляет пустую строку под выводом.

```
print("URL: " + (url))

json_data = requests.get(url).json()
json_status = json_data["info"]["statuscode"]

if json_status == 0:
    print("API Статус: " + str(json_status) + " = Успешный вызов маршрута.\n")
```

Шаг 6: Проверка состояния и команды печати URL.

В приведенном здесь примере используются следующие параметры.

```
orig = "Washington, D.C."
dest = "Baltimore, Md"
```

Сохраните и запустите сценарий **mapquest_parse-json_2.py** и убедитесь, что он работает. При необходимости устраните неполадки в коде. Вы должны получить результат, похожий на следующий. Обратите внимание, что ваш ключ встроен в URL-запрос.

```
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-
json_2.py
URL:
https://www.mapquestapi.com/directions/v2/route?key=DptE4HhC9zw3c71LbJHazOH7Q2C
2hdRy&from=Washington%2C+D.C.&to=Baltimore%2C+Md
API Статус: 0 = Успешный вызов маршрута.

devasc@labvm:~/labs/devnet-src/mapquest$
```

```
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-json_2.py
URL: https://www.mapquestapi.com/directions/v2/route?key=DptE4HhC9zw3c71LbJ
HazOH7Q2C2hdRy&from=Washington%2C+D.C.&to=Baltimore%2C+Md
API Статус: 0 = Успешный вызов маршрута.
```

Шаг 7: Пользовательские данные для начального и конечного местоположений.

Вы использовали статические значения для переменных местоположения. Однако приложение требует, чтобы пользователь вводил их. Давайте обновим приложение:

Сохраните свой сценарий под именем **mapquest_parse-json_3.py**.

Перенесите переменные **orig** и **dest** внутрь цикла while, в котором запрашивается ввод пользователем начального местоположения и пункта назначения. Цикл **while** позволяет пользователю продолжать делать запросы для различных направлений. Добавьте следующий код, выделенный ниже, после ключевого параметра. Убедитесь, что весь оставшийся код правильно расположен с отступом внутри цикла while.

```
import urllib.parse
import requests

main_api = "https://www.mapquestapi.com/directions/v2/route?"
key = "DptE4HhC9zw3c71LbJHazOH7Q2C2hdRy"

while True:
   orig = input("Начальное местоположение: ")
   dest = input("Место назначения: ")
   url = main_api + urllib.parse.urlencode({"key": key, "from":orig, "to":dest})
   print("URL: " + (url))
   json_data = requests.get(url).json()
   json_status = json_data["info"]["statuscode"]
   if json_status == 0:
       print("API Статус: " + str(json_status) + " = Успешный вызов маршрута.\n")
```

Шаг 8: Протестируйте функциональность пользовательского ввода.

Запустите сценарий **mapquest_parse-json_3.py** и убедитесь, что он работает. При необходимости устраните неполадки в коде. Вы должны получить

результат, подобный тому, что показан ниже. Чтобы завершить работу программы, введите **Ctrl+C**. Вы получите ошибку **KeyboardInterrupt**, как показано на рисунке ниже. Чтобы прекратить работу приложения более изящно, вы добавите функцию выхода из программы в следующем шаге.

```
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-
json_3.py
Starting Location: Washington, D.C.
Destination: Baltimore, Md
URL:
https://www.mapquestapi.com/directions/v2/route?key=fZadaFOY22VIEEemZcBFfxl5vjS
XIPpZ&from=Washington%2C+D.C.&to=Baltimore%2C+Md
API Статус: 0 = Успешный вызов маршрута.

Starting Location: ^CTraceback (most recent call last):
  File "mapquest_parse-json_3.py", line 9, in <module>
    orig = input("Начальное местоположение: ")
KeyboardInterrupt

devasc@labvm:~/labs/devnet-src/mapquest$
```

```
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-json_3.py
Начальное местоположение: Washington, D.C.
Место назначения: Baltimore, Md
URL: https://www.mapquestapi.com/directions/v2/route?key=DptE4HhC9zw3c71LbJ
HazOH7Q2C2hdRy&from=Washington%2C+D.C.&to=Baltimore%2C+Md
API Статус: 0 = Успешный вызов маршрута.

Начальное местоположение: ^CTraceback (most recent call last):
  File "mapquest_parse-json_3.py", line 8, in <module>
    orig = input("Начальное местоположение: ")
KeyboardInterrupt

devasc@labvm:~/labs/devnet-src/mapquest$
```

Шаг 9: Добавьте функцию выхода из приложения.

Вместо принудительного выхода из приложения с помощью прерывания клавиатуры вы добавите возможность для пользователя ввести **q** или **quit** в качестве ключевых слов для выхода из приложения. Выполните следующие шаги для обновления приложения:

Сохраните свой сценарий под именем **mapquest_parse-json_4.py**.

Добавьте оператор **if** после каждой переменной местоположения, чтобы проверить, вводит ли пользователь **q** или **quit**, как показано ниже.

```
while True:
    orig = input("Начальное местоположение: ")
    if orig == "quit" or orig == "q":
        break
    dest = input("Место назначения: ")
    if dest == "quit" or dest == "q":
        break
```

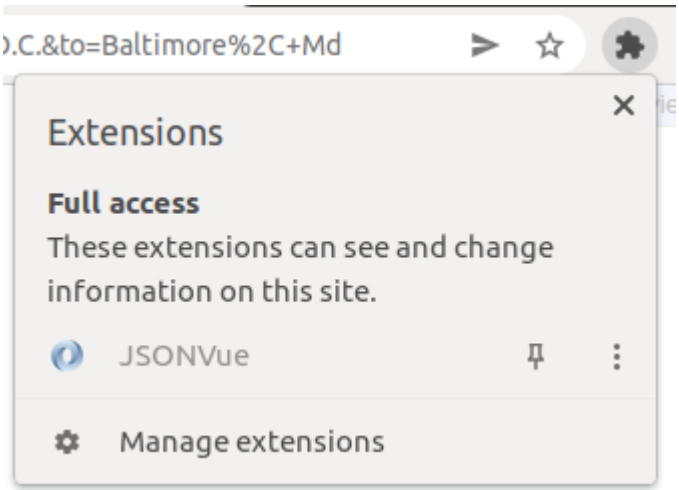
Шаг 10: Протестируйте функциональность выхода из системы.

Запустите сценарий **mapquest_parse-json_4.py** четыре раза, чтобы проверить каждую переменную местоположения. Убедитесь, что и **quit**, и **q** завершают работу приложения. При необходимости устраните неполадки в коде. Вы должны получить результат, похожий на следующий.

```
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-
json_4.py
Начальное местоположение: q
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-
json_4.py
Начальное местоположение: quit
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-
json_4.py
Начальное местоположение: Washington, D.C
Место назначения: q
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-
json_4.py
Начальное местоположение Starting Location: Washington, D.C.
Место назначения: quit
devasc@labvm:~/labs/devnet-src/mapquest$
```

Шаг 11: Отображение данных JSON в JSONView.

Браузер Chromium в виртуальной машине DEVASC содержит расширение JSONView. Вы можете использовать его для просмотра объекта JSON в удобном для чтения, цветном и сворачиваемом формате.



Запустите ваш **mapquest_parse-json_4.py** снова и скопируйте код, возвращаемый для URL. Не используйте код, приведенный ниже. Ваш результат будет включать ваш ключ API.

```
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-
json_4.py
Начальное местоположение: Washington, D.C.
Место назначения: Baltimore, Md
URL:
https://www.mapquestapi.com/directions/v2/route?key=DptE4HhC9zw3c71LbJHazOH7Q2C
2hdRy&from=Washington%2C+D.C.&to=Baltimore%2C+Md
API Статус: 0 = Успешный вызов маршрута.

Starting Location: quit
```

```
devasc@labvm:~/labs/devnet-src/mapquest$
```

Вставьте URL в адресную строку браузера Chromium.

Сверните данные JSONView, выбрав тире "-" перед route, вы увидите, что есть два корневых словаря: route и info

```
{
 - route:{
      hasTollRoad: false,
      hasBridge: true,
 <вывод опущен>
```

Вы увидите, что есть два корневых словаря: route и info. Обратите внимание, что info содержит пару ключ/значение statuscode, используемую в вашем коде.

```
{
 + route: {},
 - info: {
       statuscode: 0,
     - copyright: {
           imageAltText: "© 2022 MapQuest, Inc.",
           imageUrl: "http://api.mqcdn.com/res/mqlogo.gif",
           text: "© 2022 MapQuest, Inc."
       },
       messages: [ ]
   }
}
```

Разверните словарь маршрутов **route** (нажмите на знак плюс "+" перед **route**) и изучите богатые данные. Там есть значения, указывающие, есть ли на маршруте платные дороги, мосты, туннели, шоссе, закрытия или пересечения с другими странами. Вы также должны увидеть значения расстояния, общего времени, которое займет поездка, и расхода топлива. Чтобы разобрать и отобразить эти данные в вашем приложении, вы должны указать словарь **route**, а затем выбрать пару ключ/значение, которую вы хотите вывести. В следующей части лабораторной работы вы выполните разбор словаря маршрутов.

Часть 5: Обновление приложения MapQuest Directions с помощью дополнительных функций

В этой части мы добавим дополнительные функции в приложение MapQuest Directions, чтобы предоставить пользователю больше информации. Включим краткую информацию о поездке, а затем список направлений, разобранных из словаря ног. В качестве заключительного шага мы добавим базовую проверку ошибок для подтверждения вводимых пользователем данных.

Шаг 1: Отображение краткой информации о поездке, включающей продолжительность, расстояние и использованное топливо.

Сохраните свой сценарий под именем **mapquest_parse-json_5.py**.

Ниже команды печати статуса API добавьте несколько операторов печати, которые отображают местоположение "от" и "до", а также ключи **formattedTime**, **distance** и **fuelUsed**.

Дополнительные операторы также включают операторы print, которые будут выводить двойную строку перед следующим запросом о начальном местоположении. Убедитесь, что эти утверждения встроены в функцию while True.

```
if json_status == 0:
        print("API Статус: " + str(json_status) + " = Успешный вызов маршрута.\n")
        print("=============================================")
        print("Маршрут из " + (orig) + " в " + (dest))
        print("Продолжительность поездки: " + (json_data["route"]["formattedTime"]))
        print("Мили: " + str(json_data["route"]["distance"]))
        print("Использованное топливо (Гал.): " + str(json_data["route"]["fuelUsed"]))
        print("=============================================")
```

Сохраните и запустите файл **mapquest_parse-json_5.py**, чтобы увидеть следующий результат.

```
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-json_5.py
Начальное местоположение: Washington, D.C.
Место назначения: Baltimore, Md
URL: https://www.mapquestapi.com/directions/v2/route?key=DptE4HhC9zw3c71LbJHazOH7Q2C2hdRy&from=Washington%2C+D.C.&to=Baltimore%2C+Md
API Статус: 0 = Успешный вызов маршрута.

=============================================
Маршрут из Washington, D.C. в Baltimore, Md
Продолжительность поездки:   00:49:17
Мили:            38.089
Использованное топливо (Гал.): 1.65
=============================================
Начальное местоположение: q
devasc@labvm:~/labs/devnet-src/mapquest$
```

По умолчанию MapQuest использует **Британскую имперскую систему мер**, и к сожалению, нет переключателя для изменения данных в метрическую систему. Поэтому, вероятно, нам следует преобразовать приложение для отображения метрических значений, как показано ниже.

```
    print("Километры:      " + str((json_data["route"]["distance"])*1.61))
    print("Использованное топливо (Литр.): " + str((json_data["route"]["fuelUsed"])*3.78))
```

Запустите модифицированный сценарий **mapquest_parse-json_5.py**, чтобы увидеть следующий результат:

```
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-json_5.py
Начальное местоположение: Washington, D.C.
Место назначения: Baltimore, Md
```

```
URL:
https://www.mapquestapi.com/directions/v2/route?key=DptE4HhC9zw3c71LbJH
azOH7Q2C2hdRy&from=Washington%2C+D.C.&to=Baltimore%2C+Md
API Статус: 0 = Успешный вызов маршрута.

=============================================
Маршрут из Washington, D.C. в Baltimore, Md
Продолжительность поездки: 00:49:17
Километры:       61.32329
Использованное топливо (Литр.): 6.236999999999999
=============================================
Начальное местоположение: q
devasc@labvm:~/labs/devnet-src/mapquest$
```

Лишние десятичные знаки для Километров и Использованного топлива ненесут особо важной информации. Используйте аргумент **"{:.2f}".format** для форматирования плавающих значений до 2 знаков после запятой перед преобразованием их в строковые значения, как показано ниже. Каждый оператор должен быть на одной строке.

```
    print("Километры:                                                  "        +
str("{:.2f}".format((json_data["route"]["distance"])*1.61)))
    print("Использованное          топливо          (Литр.):          "          +
str("{:.2f}".format((json_data["route"]["fuelUsed"])*3.78)))
```

Шаг 2: Протестируйте функциональность парсинга и форматирования.

Сохраните и запустите сценарий **mapquest_parse-json_5.py**, чтобы убедиться в его работоспособности. При необходимости устраните неполадки в коде. Убедитесь, что вы правильно расставили все открывающие и закрывающие круглые скобки. Вы должны получить результат, похожий на следующий.

```
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-
json_5.py
Начальное местоположение: Washington, D.C.
Место назначения: Baltimore, Md
URL:
https://www.mapquestapi.com/directions/v2/route?key=DptE4HhC9zw3c71LbJH
azOH7Q2C2hdRy&from=Washington%2C+D.C.&to=Baltimore%2C+Md
API Статус: 0 = Успешный вызов маршрута.

=============================================
Маршрут из Washington, D.C. в Baltimore, Md
Продолжительность поездки: 00:49:17
Километры:       61.32
Использованное топливо (Литр.):  6.24
=============================================
Начальное местоположение: q
devasc@labvm:~/labs/devnet-src/mapquest$
```

Шаг 3: Просмотрите список маневров в данных JSON.

Теперь мы готовы отобразить пошаговый маршрут от начального места до пункта назначения. Вернитесь в браузер Chromium, где ранее вы

просматривали вывод в JSONView. Если вы закрыли браузер, скопируйте URL-адрес с прошлого запуска программы и вставьте его в адресную строку браузера.

Внутри словаря **route** найдите список "**legs**". **legs** включает один большой словарь с большей частью данных JSON. Найдите список маневров **maneuvers** и сверните каждый из семи словарей внутри него, как показано ниже (нажмите на знак "-" минус, чтобы переключить его на знак "+" плюс). Если вы используете разные места, у вас, вероятно, будет разное количество словарей маневров.

```
- legs: [
    - {
        hasTollRoad: false,
        hasBridge: true,
        destNarrative: "Proceed to BALTIMORE, MD.",
        distance: 38.089,
        hasTimedRestriction: false,
        hasTunnel: false,
        hasHighway: true,
        index: 0,
        formattedTime: "00:49:29",
        origIndex: -1,
        hasAccessRestriction: false,
        hasSeasonalClosure: false,
        hasCountryCross: false,
      - roadGradeStrategy: [
            [ ]
        ],
        destIndex: 3,
        time: 2969,
              hasUnpaved: false,
        origNarrative: "",
      - maneuvers: [
          + {…},
          + {…},
          + {…},
          + {…},
          + {…},
          + {…},
          + {…}
        ],
        hasFerry: false
    }
      ],
    - options: {
```

Разверните первый словарь в списке **maneuvers**. Каждый словарь содержит **narrative** ключ со значением, например, "Начать движение на север...", как показано ниже. Вам нужно разобрать данные JSON, чтобы извлечь значение для **narrative** ключа для отображения в вашем приложении.

```
- legs: [
    - {
      hasTollRoad: false,
```

```
        hasBridge: true,
        destNarrative: "Proceed to BALTIMORE, MD.",
        distance: 38.089,
        hasTimedRestriction: false,
        hasTunnel: false,
        hasHighway: true,
        index: 0,
        formattedTime: "00:49:29",
        origIndex: -1,
        hasAccessRestriction: false,
        hasSeasonalClosure: false,
        hasCountryCross: false,
      - roadGradeStrategy: [
            [ ]
        ],
        destIndex: 3,
        time: 2969,
              hasUnpaved: false,
        origNarrative: "",
     - maneuvers: [
         - {
              distance: 0.792,
            - streets: [
                  "6th St",
                  "US-50 E",
                  "US-1 N"
              ],
              narrative: "Start out going north on 6th St/US-50E/US-1 N toward
Pennsylvania Ave/US-1 Alt N.",
              turnType: 0,
            - startPoint: {
                 lng: -77.019913,
                 lat: 38.892063
              },
              index: 0,
              formattedTime: "00:02:06",
              directionName: "North",
              maneuverNotes: [ ],
              linkIds: [ ],
            - signs: [
                - {
                      extraText: "",
                      ext: "50",
                      type: 2,
<вывод опущен>
```


Примечание: Для отображения значения в описании была добавлена обводка слов.

Шаг 4: Добавьте цикл for для итерации данных JSON маневров.

Выполните следующие шаги, чтобы обновить приложение для отображения значения ключа описания. Для этого необходимо создать цикл for, который будет итерационно просматривать список маневров, отображая значение

описательной части для каждого маневра от начального места до пункта назначения.

Сохраните свой сценарий под именем **mapquest_parse-json_6.py**.

Добавьте цикл for, выделенный ниже, после второго двустрочного оператора печати. Цикл for итерирует каждый список maneuvers и выполняет следующие действия:

Выводит значение **narrative**.

Конвертирует мили в километры с помощью ***1.61**

Форматирует значение километража для вывода только двух знаков после запятой с помощью функции **"{:.2f}".format**.

Добавьте print(), который будет отображать двойную строку перед тем, как приложение запросит другое начальное местоположение, как показано ниже.

```
print("Использованное топливо (Литр.): " +
str("{:.2f}".format((json_data["route"]["fuelUsed"])*3.78)))
print("==============================================")
for each in json_data["route"]["legs"][0]["maneuvers"]:
    print((each["narrative"]) + " (" + str("{:.2f}".format((each["distance"])*1.61) + "
км)"))
print("==============================================\n")
```

Шаг 5: Активность - тестирование итерации JSON.

Сохраните и запустите сценарий **mapquest_parse-json_6.py**, чтобы убедиться в его работоспособности. При необходимости устраните неполадки в коде. Вы должны получить результат, похожий на следующий:

```
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-
json_6.py
Начальное местоположение: Washington, D.C.
Место назначения: Baltimore, Md
URL:
https://www.mapquestapi.com/directions/v2/route?key=DptE4HhC9zw3c71LbJH
azOH7Q2C2hdRy&from=Washington%2C+D.C.&to=Baltimore%2C+Md
API Статус: 0 = Успешный вызов маршрута.

==============================================
Маршрут из Washington, D.C. в Baltimore, Md
Продолжительность поездки: 00:49:17
Километры:      61.32
Использованное топливо (Литр.):  6.24
==============================================
Start out going north on 6th St/US-50 E/US-1 N toward Pennsylvania Ave/US-1 Alt
N. (1.28 км)
Turn right onto New York Ave/US-50 E. Continue to follow US-50 E (Crossing into
Maryland). (7.51 км)
Take the Balt-Wash Parkway exit on the left toward Baltimore. (0.88 км)
Merge onto MD-295 N. (50.38 км)
Turn right onto W Pratt St. (0.86 км)
Turn left onto S Calvert St/MD-2. (0.43 км)
Welcome to BALTIMORE, MD. (0.00 км)
==============================================
```

```
Начальное местоположение: q
devasc@labvm:~/labs/devnet-src/mapquest$
```

```
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-json_6.py
Начальное местоположение: Washington, D.C.
Место назначения: Baltimore, Md
URL: https://www.mapquestapi.com/directions/v2/route?key=DptE4HhC9zw3c71LbJHazOH7Q2C2hdRy&from=W
ashington%2C+D.C.&to=Baltimore%2C+Md
API Статус: 0 = Успешный вызов маршрута.

=============================================
Маршрут из Washington, D.C. в Baltimore, Md
Продолжительность поездки: 00:49:17
Километры:      61.32
Использованное топливо (Литр.): 6.24
=============================================
Start out going north on 6th St/US-50 E/US-1 N toward Pennsylvania Ave/US-1 Alt N. (1.28 км)
Turn right onto New York Ave/US-50 E. Continue to follow US-50 E (Crossing into Maryland). (7.51
 км)
Take the Balt-Wash Parkway exit on the left toward Baltimore. (0.88 км)
Merge onto MD-295 N. (50.38 км)
Turn right onto W Pratt St. (0.86 км)
Turn left onto S Calvert St/MD-2. (0.43 км)
Welcome to BALTIMORE, MD. (0.00 км)
=============================================

Начальное местоположение: q
devasc@labvm:~/labs/devnet-src/mapquest$
```

Шаг 6: Проверка на недопустимый ввод данных пользователем.

Теперь вы готовы добавить последнюю функцию в ваше приложение, чтобы сообщать об ошибке, когда пользователь вводит недопустимые данные. Вспомните, что перед разбором данных JSON вы запустили цикл if, чтобы убедиться, что возвращаемый код состояния равен 0:

```
json_status = json_data["info"]["statuscode"]

    if json_status == 0:
        print("API Статус: " + str(json_status) + " = Успешный вызов
маршрута.\n")
```

Но что произойдет, если код состояния не равен 0? Например, пользователь может ввести недопустимое местоположение или не ввести одно или несколько мест. Если это так, то приложение отображает URL и запрашивает новое начальное местоположение. Пользователь понятия не имеет, что произошло.

Чтобы вызвать отказ приложения без уведомления пользователя, попробуйте использовать в приложении следующие значения. Вы должны увидеть похожие результаты.

```
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-
json_6.py
Место назначения: Washington, D.C.
Продолжительность поездки: Beijing, China
URL:
https://www.mapquestapi.com/directions/v2/route?key=DptE4HhC9zw3c71LbJHazOH7Q
2C2hdRy%2C+D.C.&to=Beijing%2C+China
Место назначения: Washington, D.C.
```

```
Продолжительность поездки: Bal
URL:
https://www.mapquestapi.com/directions/v2/route?key=DptE4HhC9zw3c71LbJHazOH7Q
2C2hdRy&from=Washington%2C+D.C.&to=Bal
Место назначения: q
devasc@labvm:~/labs/devnet-src/mapquest$
```

Скопируйте один из URL-адресов на вкладку браузера Chromium. Обратите внимание, что единственная запись в словаре **route** - это словарь **routeError** с **errorCode** 2. В словаре info **statuscode** равен **402**. Следовательно, ваш цикл if никогда не выполнял код для случая, когда **statuscode** равен **0**.

Сохраните свой сценарий под именем **mapquest_parse-json_7.py**.

Чтобы предоставить информацию об ошибке, когда statuscode равен 402, 611 или другому значению, добавьте два оператора elif и оператор else в цикл if. Операторы elif и else должны быть согласованы с предыдущим оператором if. После последнего двустрочного оператора print в поле if json_status == 0 добавьте следующие операторы elif и else:

```
    if json_status == 0:
       <объявления опущены>
       for each in json_data["route"]["legs"][0]["maneuvers"]:
           print((each["narrative"]) + " (" +
str("{:.2f}".format((each["distance"])*1.61) + " км)"))
        print("=============================================\n")
    elif json_status == 402:
        print("***********************************************")
        print("Статус: " + str(json_status) + "; Неверные данные пользователя для одной
или обеих локаций.")
        print("***********************************************\n")
    elif json_status == 611:
        print("***********************************************")
        print("Статус: " + str(json_status) + "; Отсутствие записи для одного или обоих
мест.")
        print("***********************************************\n")
    else:

print("*********************************************************************")
        print("Для статуса: " + str(json_status) + "; Обратитесь:")
        print("https://developer.mapquest.com/documentation/directions-api/status-
codes")
        print("*************************************************************
*******\n")
```

Первое утверждение **elif** выводится, если значение **statuscode** состояния равно **402** для недопустимого местоположения. Второе утверждение **elif** выводится, если значение **statuscode** равно **611**, потому что пользователь не указал одно или оба местоположения. Оператор **else** печатается для всех других значений **statuscode**, например, когда сайт MapQuest выдает ошибку. Оператор **else** завершает цикл **if/else** и возвращает приложение к циклу while.

Часть 6: Тестирование полной функциональности приложения

Запустите сценарий **mapquest_parse-json_7.py** и убедитесь, что он работает. При необходимости устраните неполадки в коде. Протестируйте все функции приложения. Вы должны получить результат, похожий на следующий.

```
devasc@labvm:~/labs/devnet-src/mapquest$ python3 mapquest_parse-
json_7.py
Начальное местоположение: Washington, D.C.
Место назначения: Baltimore, Md
URL:
https://www.mapquestapi.com/directions/v2/route?key=DptE4HhC9zw3c71LbJHazOH7Q
2C2hdRy&from=Washington%2C+D.C.&to=Baltimore%2C+Md
API Статус: 0 = Успешный вызов маршрута.

=============================================
Маршрут из Washington, D.C. в Baltimore, Md
Продолжительность поездки:   00:49:29
Километры:      61.32
Использованное топливо (Литр.): 6.24
=============================================
Start out going north on 6th St/US-50 E/US-1 N toward Pennsylvania Ave/US-1 Alt
N. (1.28 км)
Turn right onto New York Ave/US-50 E. Continue to follow US-50 E (Crossing into
Maryland). (7.51 км)
Take the Balt-Wash Parkway exit on the left toward Baltimore. (0.88 км)
Merge onto MD-295 N. (50.38 км)
Turn right onto W Pratt St. (0.86 км)
Turn left onto S Calvert St/MD-2. (0.43 км)
Welcome to BALTIMORE, MD. (0.00 км)
=============================================

Начальное местоположение: Moscow, Russia
Место назначения: Beijing, China
URL:
https://www.mapquestapi.com/directions/v2/route?key=DptE4HhC9zw3c71LbJHazOH7Q
2C2hdRy&from=Moscow%2C+Russia&to=Beijing%2C+China
API Статус: 0 = Успешный вызов маршрута.

=============================================
Маршрут из Moscow, Russia в Beijing, China
Trip Duration:   84:31:10
Километры:      7826.83
Использованное топливо (Литр.): 793.20
=============================================
Start out going west on Кремлёвская набережная/Kremlin Embankment. (0.37 км)
Turn slight right onto ramp. (0.15 км)
Turn slight right onto Боровицкая площадь. (0.23 км)
<вывод опущен>
Turn slight left onto 前门东大街/Qianmen East Street. (0.31 км)
Turn left onto 广场东侧路/E. Guangchang Rd. (0.82 км)
广场东侧路/E. Guangchang Rd becomes 东长安街/E. Chang'an Str. (0.19 км)
Welcome to BEIJING. (0.00 км)
=============================================
```

```
Начальное местоположение: Washington, D.C.
Место назначения: Beijing, China
URL:
https://www.mapquestapi.com/directions/v2/route?key=DptE4HhC9zw3c71LbJHazOH7Q
2C2hdRy&from=Washington%2C+D.C.&to=Beijing%2C+China
*********************************************
Статус: 402; Неверные данные пользователя для одной или обеих локаций.
*********************************************

Начальное местоположение: Washington, D.C.
Место назначения: Bal
URL:
https://www.mapquestapi.com/directions/v2/route?key=DptE4HhC9zw3c71LbJHazOH7Q
2C2hdRy&from=Washington%2C+D.C.&to=Bal
*********************************************
Статус: 402; Неверные данные пользователя для одной или обеих локаций.
*********************************************

Начальное местоположение: Washington, D.C.
Место назначения:
URL:
https://www.mapquestapi.com/directions/v2/route?key=DptE4HhC9zw3c71LbJH
azOH7Q2C2hdRy&from=Washington%2C+D.C.&to=
*********************************************
Статус: 611; Отсутствие записи для одного или обоих мест.
*********************************************

Начальное местоположение: quit
devasc@labvm:~/labs/devnet-src/mapquest$
```

Лабораторная работа №5.4.6

Packet Tracer — Исследование простой сети

Цель лабораторной работы:

Часть 1: Добавление ПК в топологию

Часть 2: Тестирование подключения по сети

Часть 3: Создание веб-страницы и ее просмотр

Часть 4: Изучение списков доступа FIREWALL

Необходимые ресурсы:

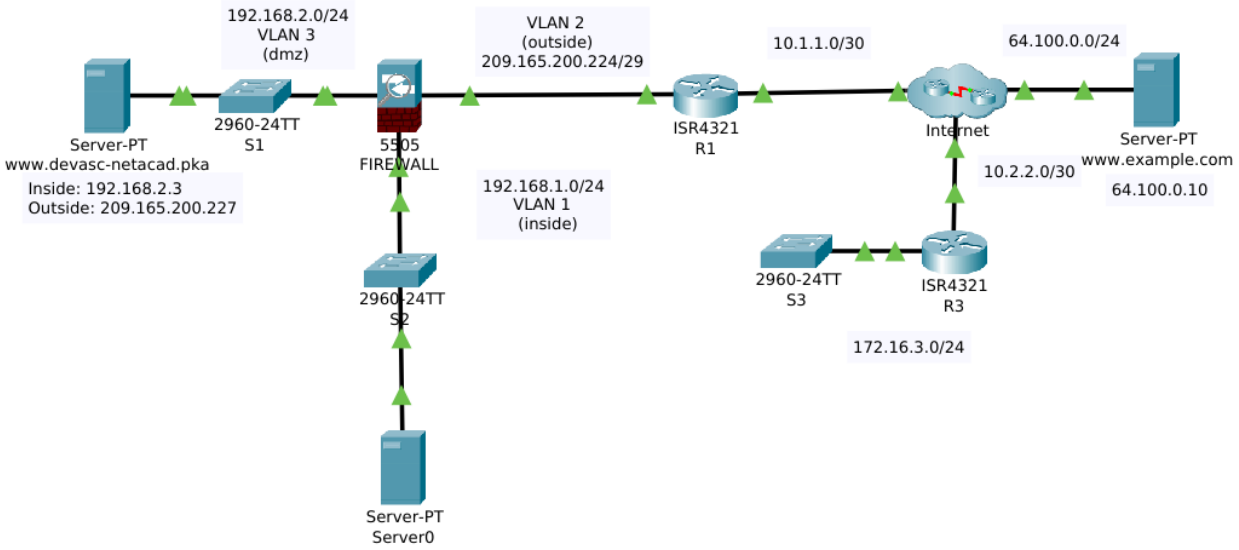
- 1 ПК
- Packet Tracer 7.4.0

Таблица адресации

Устройство	Интерфейс	IPv4 Адрес	Маска подсети	Шлюз по умолчанию
R1	G0/0/0	209.165.200.225	255.255.255.248	нет сведений
	G0/0/1	10.1.1.2	255.255.255.252	

R3	G0/0/0	10.2.2.2	255.255.255.252	нет сведений
	G0/0/1	172.16.3.1	255.255.255.0	
FIREWALL	VLAN1	192.168.1.1	255.255.255.0	нет сведений
	VLAN2	209.165.200.226	255.255.255.248	
	VLAN3	192.168.2.1	255.255.255.0	
DEVASC Server	NIC	Входящий: 192.168.2.3	255.255.255.0	192.168.1.1
	VLAN1	Исходящий: 209.165.200.227	255.255.255.248	209.165.200.225
Example Server	NIC	64.100.0.10	255.255.255.0	64.100.0.1
PC-A	NIC	192.168.1.2	255.255.255.0	192.168.1.1
PC-B	NIC	172.16.3.2	255.255.255.0	172.16.3.1

Схема сети



Порядок выполнения работы

Часть 1: Добавление ПК в топологию

В этой части вы добавите ПК в топологию и настроите их на адресацию IPv4.

Шаг 1: Разместите ПК и подключите их к сети.

Перетащите ПК в рабочую область и поместите его рядом с **S2**.

Переименуйте ПК в **PC-A**.

Перетащите ПК в рабочую область и поместите его рядом с **S3**.

Переименуйте ПК в **PC-B**.

Подключите медный кабель (**Copper Straight-Through**) от порта **FastEthernet0 PC-A** к любому доступному порту **FastEthernet** на **S2**.

Подключите медный кабель (**Copper Straight-Through**) от порта **FastEthernet0 PC-B** к любому доступному порту **FastEthernet** на **S3**.

Шаг 2: Настройте адресацию IPv4 для ПК.

Щелкните **PC-A** > **Desktop** > **IP Configuration**

Заполните следующую информацию об адресации IPv4:

```
IPv4 Адрес: 192.168.1.2
Маска подсети: 255.255.255.0
Шлюз по умолчанию: 192.168.1.1
```

Повторите это и для **PC-B**:

```
IPv4 Адрес: 172.16.3.2
Маска подсети: 255.255.255.0
Шлюз по умолчанию: 172.16.3.1
```

В окне Instructions для этого задания процент выполнения должен составлять 100%. Если это не так, нажмите кнопку

Проверить результаты, чтобы узнать, какие необходимые компоненты еще не завершены. *Остальная часть этого задания не оценивается.*

Часть 2: Тестирование подключения по сети

Щелкните **PC-B** > **Command Prompt**

Попытайтесь выполнить пинг R3. Введите

```
ping 172.16.3.1
```

Пинганите **Example Server** по адресу **64.100.0.10**

Пинганите **DEVASC Server** по адресу **209.165.200.227**

Часть 3: Создание веб-страницы и ее просмотр

В этой части вы создадите простую веб-страницу на сервере DEVASC, а затем проверите, что PC-B может получить доступ к веб-странице.

Шаг 1: Создайте веб-страницу

Щелкните **Server-PT www.devasc-netacad.pka**.

Перейдите в Службы **(Services**). В этом разделе по умолчанию выбрана первая служба - **HTTP**. Нажмите кнопку **New File**.

Назовите его **index.html**.

Packet Tracer понимает Hypertext Markup Language (HTML). Поместите следующий html-код в поле под именем файла. Если вы знаете HTML, не стесняйтесь изменять код.

```
<html>
<center><font size='+2' color='blue'>Веб-разработка</font></center>
<hr> Это моя HTML страничка! Добро пожаловать!
```

Нажмите **Save**. Нажмите **Yes** в ответ на предупреждение.

Шаг 2: Просмотрите веб-страницу.

Нажмите **PC-B**.

Откройте **Desktop**.

Перейдите в **Web Browser**.

Поместите следующий адрес в поле URL: **http://209.165.200.227**.

Ваша веб-страница должна отобразиться. Если нет, проверьте свои настройки и попробуйте снова.

Часть 4: Изучение списков доступа FIREWALL

В этой части мы познакомимся со списоком доступа устройства FIREWALL, отредактируем его и проверим, что устройство FIREWALL теперь запрещает доступ для пинга. FIREWALL теперь запрещает доступ к ping.

Шаг 1: Изучите список доступа на устройстве FIREWALL.

Нажмите **FIREWALL**

Перейдите **CLI**.

Нажмите Enter несколько раз, чтобы появилось приглашение.

Введите **en** и нажмите Enter.

Пароль отсутствует. Снова нажмите Enter.

Напишите **show run** и нажмите Enter.

Нажмите пробел, чтобы прокрутить текущую конфигурацию.

Обратите внимание на следующий список доступа:

```
access-list OUTSIDE-DMZ extended permit icmp any host 192.168.2.3
access-list OUTSIDE-DMZ extended permit tcp any host 192.168.2.3 eq www
```

Хост **192.168.2.3** — это внутренний IPv4-адрес сервера DEVASC в DMZ.

- Первое выражение access-list позволяет любому устройству получить доступ к серверу с помощью протокола Internet Control Message Protocol (ICMP), который используется командой ping.
- Второе выражение access-list позволяет любому устройству получить доступ к серверу с помощью протокола Hypertext Transfer Protocol (HTTP), который используется веб-браузерами.

Шаг 2: Измените и проверьте эффективность списка доступа.

Как правило, вы не хотите, чтобы внешний мир мог пинговать ваши внутренние серверы. Поэтому вам следует удалить утверждение списка доступа, которое явно разрешает доступ для пинга.

Войдите в режим глобальной конфигурации с помощью команды **configure terminal**.

```
FIREWALL# configure terminal
```

Удалите утверждение списка доступа, разрешающее ping, с помощью следующей команды и нажмите Enter.

```
FIREWALL(config)# no access-list OUTSIDE-DMZ extended permit icmp any host 192.168.2.3
```

Из командной строки на **PC-B** выполните **ping** внешнего IPv4-адреса сервера DEVASC. Пинг должен завершиться неудачей.

С помощью веб-браузера на **PC-B** откройте веб-страницу сервера DEVASC по адресу http://209.165.200.227. Вы должны увидеть веб-страницу, поскольку мы не удалили запись, разрешающую доступ по протоколу HTTP.

Лабораторная работа №5.5.7

Packet Tracer — Исследование сетевых протоколов

Цель лабораторной работы:

Часть 1: Настройка DNS

Часть 2: Настройка DHCP

Часть 3: Настройка NTP

Часть 4: Использование SSH для безопасного доступа к коммутатору

Часть 5: Изучение идентификаторов объектов SNMP MIB

Часть 6: Настройка HTTPS

Часть 7: Настройка EMAIL

Часть 8: Настройка FTP

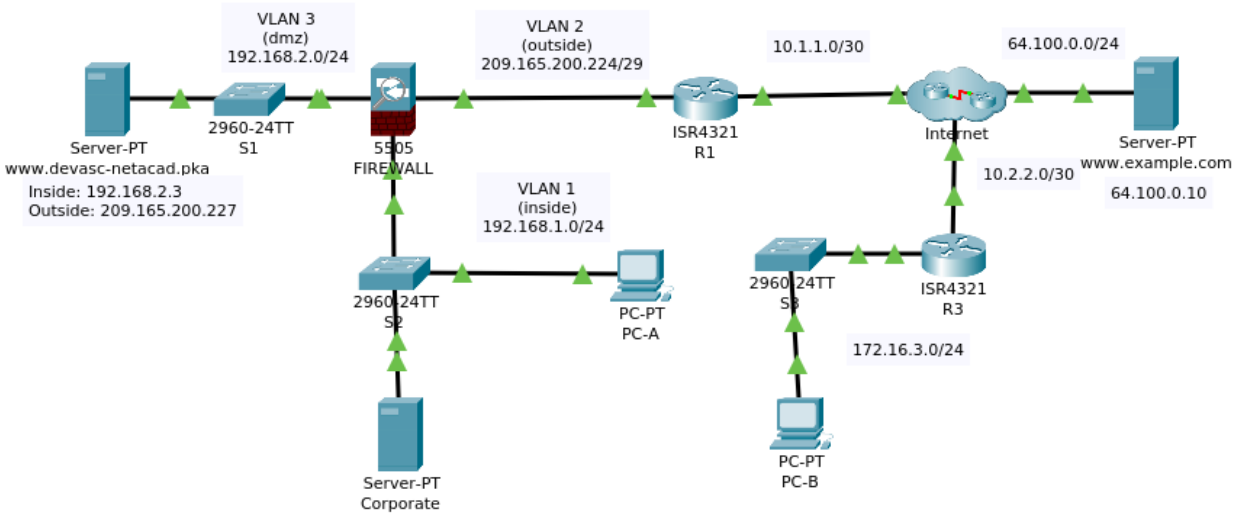
Необходимые ресурсы:

- 1 ПК
- Packet Tracer 7.4.0

Таблица адресации

Устройство	Интерфейс	IPv4 Адрес	Маска подсети	Шлюз по умолчанию
R1	G0/0/0	209.165.200.225	255.255.255.248	нет сведений
	G0/0/1	10.1.1.2	255.255.255.252	
R3	G0/0/0	10.2.2.2	255.255.255.252	нет сведений
	G0/0/1	172.16.3.1	255.255.255.0	
FIREWALL	VLAN1	192.168.1.1	255.255.255.0	нет сведений
	VLAN2	209.165.200.226	255.255.255.248	
	VLAN3	192.168.2.1	255.255.255.0	
DEVASC Server	NIC	Входящий: 192.168.2.3	255.255.255.0	192.168.1.1
	VLAN1	Исходящий: 209.165.200.227	255.255.255.248	209.165.200.225
Example Server	NIC	64.100.0.10	255.255.255.0	64.100.0.1
PC-A	NIC	Присвоен DHCP	255.255.255.0	192.168.1.1
PC-B	NIC	172.16.3.2	255.255.255.0	172.16.3.1

Схема сети



Порядок выполнения работы

Часть 1: Настройка DNS

Всем узлам в сети присваивается IP-адрес. IP-адрес может быть адресом IPv4 или IPv6. или и то, и другое. Это относится и ко всем узлам в Интернете. Но вы не используете их IP-адреса для общения с ними. Вы используете обычные имена, такие как cisco.com. Система доменных имен (DNS) - это служба, которая автоматически переводит общие, легко запоминающиеся имена в IP-адреса, чтобы связь между устройствами может осуществляться. В этом задании Packet Tracer устройства используют IPv4 адреса.

Шаг 1: Настройте локальный DNS-сервер.

Щелкните **Corporate server** > **Services**. > **DNS**.

Нажмите **On**, чтобы включить службу DNS.

Теперь, когда DNS включен, вам нужно предоставить информацию для всех хостов в сети (сетях), для которых вы хотите перевести их имена в адреса IPv4.

В поле **Name** введите **www.example.com**.

В поле **Address** введите IPv4-адрес **64.100.0.10**.

Нажмите кнопку **Add**.

Теперь вы увидите запись, которая показывает имя хоста и IPv4-адрес **Example Server**. Здесь DNS будет искать имя хоста и возвращать IPv4-адрес этого хоста любому устройству, которое его запрашивает.

Шаг 2: Настройте и протестируйте использование локального DNS-сервера.

Нажмите **PC-A** > **Config**.

В поле DNS Server введите IPv4-адрес Corporate DNS server: **192.168.1.3**.

Теперь, когда PC-A использует общие имена хостов, он будет посылать DNS-запрос на IPv4-адрес узла с таким именем хоста.

Нажмите **Desktop** > **Command Prompt**.

Пропингуйте **www.example.com**.

```
C:\> ping www.example.com
Pinging 64.100.0.10 with 32 bytes of data:
Request timed out.
<вывод опущен>

C:\> ping www.example.com
Pinging 64.100.0.10 with 32 bytes of data:
Reply from 64.100.0.10: bytes=32 time=3ms TTL=125
<вывод опущен>

C:\>
```

Шаг 3: Настройте и протестируйте использование удаленного DNS-сервера.

PC-B не имеет локального DNS-сервера. Поэтому в качестве DNS-сервера он будет использовать **Example Server**.

Нажмите **PC-B** > **Config**.

В поле DNS Server введите IPv4-адрес корпоративного DNS-сервера: 64.100.0.10.

Нажмите **Desktop** > **Command Prompt**.

Пропингуйте **www.example.com**.

Пропингуйте **www.devasc-netacad.pka**.

Закройте окно **Command Prompt** и нажмите **Web Browser**.

Введите **www.example.com** в поле URL и нажмите **Go**. Теперь вы должны увидеть веб-страницу **Example.com**.

Введите **www.devasc-netacad.pka** в поле URL и нажмите **Go**. Теперь вы должны увидеть веб-страницу сервера **DEVASC**.

Часть 2: Настройка DHCP

Ручная настройка адресов IPv4 подходит для очень маленьких сетей, но в больших сетях необходимо автоматически предоставлять IPv4-адресацию устройствам при подключении к сети. Dynamic Host Configuration Protocol (DHCP) обеспечивает эту услугу. Это также удобно при перемещении устройств, поскольку при перемещении в другую подсеть, они получат новый адрес и смогут общаться с другими узлами.

Еще одна замечательная особенность DHCP заключается в том, что он автоматически устанавливает не только IPv4-адрес для хоста, но и подсеть, шлюз по умолчанию и адрес DNS-сервера. Это позволяет легко настраивать несколько частей информации настраивать на хостах автоматически.

Шаг 1: Настройте DHCP на Corporate server.

Щелкните **Corporate server > Services > DHCP**

Нажмите кнопку **On**, чтобы включить службу DHCP.

Теперь мы определим пул адресов IPv4, которые мы желаем назначить хостам. Будете использовать IPv4 адреса в подсети 192.168.1.0. Мы не можем использовать адрес 192.168.1.1, поскольку он уже используется интерфейсом FIREWALL. Мы также не можем использовать адрес Corporate server 192.168.1.3. Кроме того, хорошей практикой является оставление некоторых адресов свободными для статического назначения серверам или другим устройствам, адрес которых должен оставаться неизменным.

В настоящее время имя пула - **serverPool**. Не изменяйте его.

Для параметра Default Gateway введите IPv4-адрес внутреннего интерфейса FIREWALL: **192.168.1.1**. Это обеспечит каждому узлу DHCP маршрут в другие сети

Для DNS Server введите IPv4-адрес Corporate server: **192.168.1.3**. Это предоставит каждому узлу DHCP адрес, который будет использоваться для отправки сообщений DNS.

Для начального IP-адреса используйте **192.168.1.10**. Это позволит в будущем создать в сети несколько статически назначенных устройств.

Для маски подсети используйте **255.255.255.0**.

Для параметра **Maximum number of users** введите **245**.

Нажмите **Save**, чтобы перезаписать serverPool по умолчанию.

Шаг 2: Проверьте конфигурацию DHCP.

Нажмите **PC-A > IP Configuration >DCHP**.

Это может занять немного времени, но вы должны получить IPv4-адрес от маршрутизатора за пределами первых 10 адресов. Вы также должны увидеть маску подсети, шлюз по умолчанию и DNS-сервер, которые будут предоставлены для вас автоматически

Часть 3: Настройка NTP

Часы на маршрутизаторе или коммутаторе важны для управления, обеспечения безопасности и устранения неполадок в сети. Даже в небольших сетях важно синхронизировать время на всех устройствах. Пытаться сделать это вручную практически невозможно, особенно в больших сетях. Протокол сетевого времени (NTP) можно использовать для синхронизации времени на каждом устройстве, получая его от NTP-сервера, что обеспечивает одинаковое время на всех устройствах.

Шаг 1: Включите службу NTP.

Щелкните **Corporate server > Services > NTP**.

Нажмите кнопку **On** рядом с пунктом **Services**.

Шаг 2: Исследуйте NTP на S2.

S2 уже настроен на использование корпоративного сервера в качестве NTP-сервера.

Нажмите S2 > CLI.

Войдите в привилегированный режим EXEC с помощью команды **enable**. Используйте **cisco** в качестве пароля.

```
S2> enable
Password:
S2#
```

Отобразите текущее время и дату с помощью команды **show clock detail**. Обратите внимание, что время устанавливается оборудованием и не является точным.

```
S2# show clock detail
*0:3:44.318 UTC Mon Mar 1 1993
Time source is hardware calendar
S2#
```

Вы можете вручную настроить время с помощью команды **clock**. Однако лучше использовать NTP-сервер. Войдите в режим глобальной конфигурации с помощью команды **configure terminal**.

```
S2# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
S2(config)#
```

Настройте **S2** на использование **Corporate server** в качестве NTP-сервера. Выйдите из режима глобальной конфигурации и убедитесь, что **S2** теперь использует NTP. Теперь время и дата должны быть точными.

```
S2(config)# ntp server 192.168.1.3
S2(config)# exit
S2# show clock detail
14:1:26.216 UTC Thu May 21 2020
Time source is NTP
S2#
```

Часть 4: Использование SSH для безопасного доступа к коммутатору

Secure Shell (SSH) — это протокол, который используется для шифрования связи между клиентом и хостом. SSH является предпочтительным типом соединения, поскольку он безопасен по сравнению с Telnet. SSH уже был настроен на S2.

Щелкните **PC-A > Desktop > Command Prompt**.

Попытайтесь установить небезопасный сеанс **Telnet** на **S2**.

```
C:\> telnet 192.168.1.4
Trying 192.168.1.4 ...Open
[Connection to 192.168.1.4 closed by foreign host]
```

S2 отклоняет ваш запрос, поскольку он настроен только для доступа по SSH.

Попытайтесь установить SSH-соединение с S2. Пароль - cisco.

```
C:\> ssh -l administrator 192.168.1.4
Password:
S2>
```

Теперь мы можем безопасно настроить **S2**

Мы получаем доступ к командной строке для S2 через защищенное соединение. Войдите в режим глобальной конфигурации с командой **enable**, чтобы убедиться, что вы можете настроить коммутатор удаленно. Используйте **cisco** в качестве пароль. Затем введите **exit**, чтобы завершить сеанс SSH.

```
S2> enable
Password:
S2# exit
[Connection to 192.168.1.4 closed by foreign host]
C:\>
```

Часть 5: Изучение идентификаторов объектов SNMP MIB

Simple Network Management Protocol (SNMP) можно использовать для получения и установки переменных, связанных с состоянием и конфигурацией сетевых узлов, таких как маршрутизаторы и коммутаторы, а также сетевых клиентских компьютеров.

Нажмите **PC-B. > MIB Browser**.

Введите адрес **R3** в поле **Address**: 172.16.3.1.

Нажмите **Advanced**.

Введите **read** в поле для **Read Community**.

Введите **write** в поле **Write Community**.

Измените **SNMP Version** на **v3**.

Нажмите **OK**.

Нажмите стрелку рядом с **MIB Tree**, чтобы развернуть дерево.

Щелкните стрелку рядом с MIBs **router_std**.

Продолжайте расширять дерево, пока не дойдете до **.mgmt**.

Разверните **.mgmt**.

Продолжайте расширять дерево, пока не достигнете **.system**.

Разверните **.system**.

Щелкните **.sysName**.

Нажмите кнопку **GO**.

Теперь вы увидите, что значение объекта **R3**.

Разверните дерево **.interfaces** > **.ifTable** > **.ifEntry** > **.ifOperStatus** и нажмите **GO**.

Вы увидите, что два из трех интерфейсов работают. Теперь вы можете легко запросить любую информацию о маршрутизаторе.

Часть 6: Настройка HTTPS

Когда вы подключаетесь к серверу с помощью HTTP, вы предполагаете, что это правильный сервер. Данные передаваемые между вами и сервером, отправляются в виде открытого текста, поэтому если кто-то перехватит эти данные, он сможет их прочесть и манипулировать ими. Обычно это не

является проблемой, если вы просто просматриваете Интернет. Но если вы создаете учетную запись, получаете доступ к ней или предоставите какую-либо личную информацию, она может быть перехвачена и использована кем-то другим. Secure HTTP (HTTPS) добавляет уровень безопасности, шифруя соединение между вами и сервером. Сайт должен иметь сертификат безопасности из надежного источника, чтобы убедиться в его легитимности. Ваш браузер проверяет, что сертификат действителен и получен из надежного источника, прежде чем соединить вас с сайтом.

Шаг 1: Откройте веб-страницу с компьютера.

Нажмите **PC-B > Desktop > Web Browser**.

Введите **www.devasc-netacad.pka** в поле URL и нажмите **Go**. Обратите внимание, что протокол - HTTP (http://).

Шаг 2: Изучите межсетевой экран.

Нажмите **FIREWALL > CLI**.

Пароль отсутствует, поэтому нажмите Enter.

Введите **show run** и нажмите Enter.

Используйте пробел для прокрутки конфигурации брандмауэра.

Обратите внимание на следующие две конфигурации в списке доступа OUTSIDE-DMZ:

```
<вывод опущен>
access-list OUTSIDE-DMZ extended permit icmp any host 192.168.2.3
access-list OUTSIDE-DMZ extended permit tcp any host 192.168.2.3 eq www
access-list OUTSIDE-DMZ extended permit tcp any host 192.168.2.3 eq 443
<вывод опущен>
```

Строка с **www** разрешает порт 80, который является незащищенным HTTP-трафиком. Строка с портом 443 разрешает порт **443**, который является защищенным HTTP (HTTPS) трафиком.

Удалите access-list, разрешающий незащищенный HTTP-трафик на порту 80

```
FIREWALL# configure terminal
FIREWALL(config)# no access-list OUTSIDE-DMZ extended permit tcp any host 192.168.2.3
eq www
FIREWALL(config)#
```

Шаг 3: Настройте HTTPS.

Щелкните **DEVASC Server > Services > HTTP.**

Обратите внимание, что для HTTP установлено значение On, а для HTTPS - Off.

Выключите HTTP и включите HTTPS. Даже если FIREWALL больше не будет разрешать доступ по HTTP, лучше всего настроить сервер так, чтобы он разрешал только HTTPS.

Нажмите кнопку On для HTTPS.

Шаг 4: Проверьте конфигурацию HTTPS.

Нажмите **PC-B > Web Browser**.

Убедитесь, что **PC-B** больше не может получить доступ к www.devasc-netacad.pka с помощью **HTTP**. Через несколько секунд вы должны получить сообщение Request Timeout. Нажмите **Fast Forward Time**, чтобы ускорить процесс.

Измените http на https и нажмите Go. Теперь вы должны увидеть веб-страницу https://www.devasc-netacad.pka

Часть 7: Настройка EMAIL

Клиенты электронной почты используют Simple Mail Transfer Protocol (SMTP), порт 25, для отправки электронной почты на сервер. SMTP также используется для отправки электронной почты между серверами. Клиент электронной почты использует протокол Post Office Protocol 3 (POP3), порт 110, для получения почты с сервера.

Шаг 1: Настройте сервер EMAIL.

Щелкните **Example Server > Services > EMAIL**.

Включите службы **SMTP** и **POP3**.

Введите **www.example.com** в поле **Domain Name box**.

Нажмите **Set**.

Шаг 2: Создание пользователей.

В поле **User** введите **Student1**.

Введите **class** в качестве пароля.

Нажмите на плюс (+), чтобы добавить пользователя.

Повторите этот шаг, чтобы добавить пользователя под именем **Student2** с тем же паролем.

Шаг 3: Конфигурирование клиентов.

Щелкните **PC-A > Desktop > Email**.

Введите следующую информацию:

```
Ваше имя: Student1
Адрес электронной почты: Student1@www.example.com
Сервер входящих сообщений: 64.100.0.10
Сервер исходящих сообщений: 64.100.0.10
Имя пользователя: Student1
Пароль: class
```

Нажмите кнопку **Save**.

Повторите настройку на **PC-B**, заменив **Student1** на **Student2**.

Шаг 4: Отправка и получение электронной почты

На **PC-B** откройте программу Email, если она не открыта.

Нажмите кнопку **Compose**.

Заполните следующую информацию:

Кому: Student1@www.example.com

Тема: Электронная почта

В поле сообщения введите сообщение для **Student1**, например, "Сколько тебе лет?".

Нажмите **Send**.

На **PC-A** откройте Email, если он не открыт.

Нажмите Receive. Это может занять некоторое время и несколько попыток.

Дважды щелкните сообщение, когда оно придет, чтобы прочитать его.

Нажмите Reply.

Введите ответ на сообщение электронной почты, например, "Мне 5, мне 5 лет!"и нажмите **Send**.

Вернитесь в **PC-B**, нажмите **Receive**, чтобы прочитать ответ.

Часть 8: Настройка FTP

File Transfer Protocol (FTP) — это широко используемое приложение для передачи файлов между клиентами и серверами в сети. Сервер настраивается для запуска службы, где клиенты подключаются, входят в систему и передают файлы. FTP использует порт 21 в качестве командного порта сервера для создания соединения. Затем FTP использует порт 20 для передачи данных.

Шаг 1: Настройка сервера.

Нажмите **Corporate server > Services > FTP.**

Нажмите кнопку **On**, чтобы включить службу FTP.

В поле Username введите Student.

В поле Password введите class.

Установите все флажки под этими полями, чтобы разрешить пользователю запись, чтение, удаление, переименование.

Нажмите кнопку Add.

Шаг 2: Используйте службу FTP

Нажмите **PC-A > Desktop > Command Prompt.**

Введите **dir**, чтобы увидеть файлы на компьютере.

```
C:\> dir
Volume in drive C has no label.
Volume Serial Number is 5E12-4AF3
Directory of C:\

2/6/2106 23:28 PM 26 sampleFile.txt
26 bytes 1 File(s)
C:\>
```

FTP на Corporate server IPv4-адрес

```
C:\> ftp 192.168.1.3
Trying to connect...192.168.1.3
Connected to 192.168.1.3
220- Welcome to PT Ftp server
Username:
```

Введите имя пользователя и пароль, которые вы настроили ранее для получения доступа.

Введите **dir**, чтобы увидеть файлы, доступные на сервере.

```
ftp> dir
```

```
Listing /ftp directory from 192.168.1.3:
0 : asa842-k8.bin 5571584
1 : asa923-k8.bin 30468096
2 : c1841-advipservicesk9-mz.124-15.T1.bin 33591768
3 : c1841-ipbase-mz.123-14.T7.bin 13832032
<вывод опущен>
```

Введите **put sampleFile.txt**, чтобы отправить файл на сервер

```
ftp> put sampleFile.txt

Writing file sampleFile.txt to 192.168.1.3:
File transfer in progress...

[Transfer complete - 26 bytes]

26 bytes copied in 0.08 secs (325 bytes/sec)
ftp>
```

Снова используйте команду **dir**, чтобы снова перечислить содержимое FTP-сервера и увидеть файл.

Введите **get asa842-k8.bin**, чтобы получить файл с сервера. Это может занять 30 секунд или более поскольку файл очень большой.

```
ftp> get asa842-k8.bin

Reading file asa842-k8.bin from 192.168.1.3:
File transfer in progress...

[Transfer complete - 5571584 bytes]

5571584 bytes copied in 46.893 secs (42706 bytes/sec)
ftp>
```

Введите delete sampleFile.txt, чтобы удалить файл с сервера.

```
ftp> delete sampleFile.txt

Deleting file sampleFile.txt from 192.168.1.3: ftp>
[Deleted file sampleFile.txt successfully ]
ftp>
```

Введите **quit**, чтобы выйти из FTP-клиента.
Снова отобразите содержимое каталога на ПК, чтобы увидеть файл изображения с FTP-сервера

Лабораторная работа №5.6.7

Инструменты для устранения неполадок в сети

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC

Часть 2: Знакомство с инструментом ifconfig

Часть 3: Знакомство с инструментом ping

Часть 4: Знакомство с инструментом traceroute

Часть 5: Знакомство с инструментом nslookup

Необходимые ресурсы:

– 1 ПК
– Virtual Box или VMWare
– DEVASC виртуальная машина

Порядок выполнения работы

Часть 1: Запуск виртуальной машины DEVASC

Если вы еще не завершили **лабораторную работу - Установка лабораторной среды виртуальной машины**, сделайте это сейчас. Если вы уже завершили эту лабораторную работу, запустите виртуальную машину **DEVASC**.

Часть 2: Знакомство с инструментом ifconfig

Утилита ifconfig — это приложение для использования в операционных системах на базе UNIX, таких как Linux. Аналогичная утилита доступна в Windows под названием ipconfig. Эти приложения используются для управления сетевыми интерфейсами из командной строки. С помощью ifconfig можно выполнить следующие действия:

- Настраивать IP-адреса и маски подсети для интерфейсов.
- Получать информацию о состоянии сетевых интерфейсов.
- Включать или отключать сетевые интерфейсы.
- Изменять MAC-адреса сетевого интерфейса.

Шаг 1: Просмотрите параметры ifconfig.

Инструмент ifconfig имеет множество различных опций, которые могут быть добавлены к команде для выполнения определенных задач.

Откройте окно терминала либо непосредственно с рабочего стола, либо в VS Code.

Введите **ifconfig --help**, чтобы увидеть все доступные опции для команды.

```
devasc@labvm:~$ ifconfig --help
Usage:
  ifconfig [-a] [-v] [-s] <interface> [[<AF>] <address>]
  [add <address>[/<prefixlen>]]
  [del <address>[/<prefixlen>]]
  [[-]broadcast [<address>]]  [[-]pointopoint [<address>]]
  [netmask <address>]  [dstaddr <address>]  [tunnel <address>]
  [outfill <NN>] [keepalive <NN>]
  [hw <HW> <address>]  [mtu <NN>]
  [[-]trailers]  [[-]arp]  [[-]allmulti]
  [multicast]  [[-]promisc]
  [mem_start <NN>]  [io_addr <NN>]  [irq <NN>]  [media <type>]
  [txqueuelen <NN>]
  [[-]dynamic]
  [up|down] ...
```

Здесь представлен обзор некоторых наиболее широко используемых вариантов:

- **add or del** – Эта опция позволяет добавлять или удалять IP-адреса и их маску подсети.
- **hw ether** – Используется для изменения физического MAC-адреса. Это может быть полезно, например, для изменения его на легко узнаваемое имя, чтобы он выделялся в журналах при поиске неисправностей.
- **up and down** – Эти опции используются для включения и отключения интерфейсов. Убедитесь, какой интерфейс вы отключаете. Если это тот интерфейс, который вы используете для удаленного подключения к устройству, вы будете отключены!

Шаг 2: Просмотрите состояние всех интерфейсов.

Отобразите состояние всех используемых сетевых интерфейсов, выдав команду **ip addr** самостоятельно.

```
devasc@labvm:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 08:00:27:e9:3d:e6 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic enp0s3
       valid_lft 85901sec preferred_lft 85901sec
    inet6 fe80::a00:27ff:fee9:3de6/64 scope link
       valid_lft forever preferred_lft forever
3: dummy0: <BROADCAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
group default qlen 1000
    link/ether e2:2b:24:96:98:b8 brd ff:ff:ff:ff:ff:ff
    inet 192.0.2.1/32 scope global dummy0
       valid_lft forever preferred_lft forever
    inet 192.0.2.2/32 scope global dummy0
       valid_lft forever preferred_lft forever
    inet 192.0.2.3/32 scope global dummy0
       valid_lft forever preferred_lft forever
    inet 192.0.2.4/32 scope global dummy0
       valid_lft forever preferred_lft forever
    inet 192.0.2.5/32 scope global dummy0
       valid_lft forever preferred_lft forever
    inet6 fe80::e02b:24ff:fe96:98b8/64 scope link
       valid_lft forever preferred_lft forever
devasc@labvm:~$
```

Из этого вывода мы можем многое узнать об интерфейсах виртуальной машины:

- Имеется 3 интерфейса, интерфейс loopback (**lo**), **enp0s3** и **dummy0**.
- **ether** показывает MAC-адрес и то, что Ethernet является инкапсуляцией соединения.

- **inet** – IP-адрес, маска подсети показана в косой черте, а **brd** – широковещательный адрес.
- **UP** указывает, что интерфейс включен.
- **mtu** – это Maximum Transmission Unit, определяющий максимальное количество байт, которое кадр может быть передан на данном носителе до фрагментации.

Часть 3: Знакомство с инструментом ping

Инструмент **ping** – это приложение, которое используется для тестирования сетевого соединения между устройствами. ping использует протокол Internet Control Message Protocol (ICMP) для отправки пакетов на устройство в сети и ожидает ответа устройства. ping сообщает о сетевых ошибках, потере пакетов и времени жизни (TTL), среди прочей статистики.

Шаг 1: Просмотрите параметры пинга.

Ping доступен только в окне терминала или командной строки.

Введите **ping -help**, чтобы увидеть все доступные параметры команды.

```
devasc@labvm:~$ ping -help

Usage
  ping [options] <destination>

Options:
  <destination>      dns name or ip address
  -a                 use audible ping
  -A                 use adaptive ping
  -B                 sticky source address
  -c <count>         stop after <count> replies
  -D                 print timestamps
  -d                 use SO_DEBUG socket option
  -f                 flood ping
  -h                 print help and exit
  -I <interface>     either interface name or address
  -i <interval>      seconds between sending each packet
  -L                 suppress loopback of multicast packets
  -l <preload>       send <preload> number of packages while waiting replies
  -m <mark>          tag the packets going out
  -M <pmtud opt>     define mtu discovery, can be one of <do|dont|want>
  -n                 no dns name resolution
  -O                 report outstanding replies
  -p <pattern>       contents of padding byte
  -q                 quiet output
  -Q <tclass>        use quality of service <tclass> bits
  -s <size>          use <size> as number of data bytes to be sent
  -S <size>          use <size> as SO_SNDBUF socket option value
  -t <ttl>           define time to live
  -U                 print user-to-user latency
  -v                 verbose output
  -V                 print version and exit
  -w <deadline>      reply wait <deadline> in seconds
  -W <timeout>       time to wait for response
```

```
IPv4 options:
  -4                 use IPv4
  -b                 allow pinging broadcast
  -R                 record route
  -T <timestamp>     define timestamp, can be one of <tsonly|tsandaddr|tsprespec>

IPv6 options:
  -6                 use IPv6
  -F <flowlabel>     define flow label, default is random
  -N <nodeinfo opt>  use icmp6 node info query, try <help> as argument

For more details see ping(8).
devasc@labvm:~$
```

Шаг 2: Выполните Ping хоста.

Инструмент ping имеет множество различных опций, которые можно выбрать для настройки того, как будет происходить взаимодействие. Некоторые из опций, которые можно указать, включают:

- Сколько эхо-запросов ICMP нужно отправить.
- IP-адрес источника, если на устройстве имеется несколько интерфейсов.
- Время ожидания ответа.
- Размер пакета, если вы хотите отправлять пакеты большего размера, чем 64 байта по умолчанию. Это поможет определить, каков максимальный блок передачи (MTU).

Пропингуем www.yandex.ru, чтобы проверить, доступен ли он.

```
devasc@labvm:~$ ping -c 5 www.yandex.ru
PING www.yandex.ru (5.255.255.80) 56(84) bytes of data.
64 bytes from yandex.ru (5.255.255.80): icmp_seq=1 ttl=53 time=49.2 ms
64 bytes from yandex.ru (5.255.255.80): icmp_seq=2 ttl=53 time=18.9 ms
64 bytes from yandex.ru (5.255.255.80): icmp_seq=3 ttl=53 time=19.8 ms
64 bytes from yandex.ru (5.255.255.80): icmp_seq=4 ttl=53 time=15.9 ms
64 bytes from yandex.ru (5.255.255.80): icmp_seq=5 ttl=53 time=33.4 ms

--- www.yandex.ru ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 15.879/27.426/49.212/12.445 ms
devasc@labvm:~$
```

```
devasc@labvm:~$ ping -c 5 www.yandex.ru
PING www.yandex.ru (5.255.255.80) 56(84) bytes of data.
64 bytes from yandex.ru (5.255.255.80): icmp_seq=1 ttl=53 time=49.2 ms
64 bytes from yandex.ru (5.255.255.80): icmp_seq=2 ttl=53 time=18.9 ms
64 bytes from yandex.ru (5.255.255.80): icmp_seq=3 ttl=53 time=19.8 ms
64 bytes from yandex.ru (5.255.255.80): icmp_seq=4 ttl=53 time=15.9 ms
64 bytes from yandex.ru (5.255.255.80): icmp_seq=5 ttl=53 time=33.4 ms

--- www.yandex.ru ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 15.879/27.426/49.212/12.445 ms
devasc@labvm:~$
```

Этот пинг задал счетчик из 5 пакетов.

Инструмент **ping** автоматически выполняет разрешение DNS, возвращая 5.255.255.80 (ваш возвращенный IP-адрес может быть другим). Также отображается время жизни (TTL) для полученных эхо-ответов и время прохождения в оба конца. Итоговая статистика подтверждает, что было передано 5 пакетов ICMP эхо-запроса и получено 5 пакетов ICMP эхо-ответа, что составляет 0% потери пакетов. Также отображаются статистические данные о минимальном, среднем, максимальном и стандартном отклонении времени, которое потребовалось пакетам, чтобы добраться до места назначения и обратно.

Если вы не получаете никаких ответов от адресата, это не обязательно означает, что хост находится в автономном режиме или недоступен. Это может означать, что ICMP-пакеты блокируются брандмауэром. Лучшей практикой является предоставление только тех услуг, которые должны быть доступны на узлах сети.

Для **IPv6** существует аналогичная утилита, которая называется ping6 и также доступна в большинстве операционных систем.

Часть 4: Знакомство с инструментом traceroute

Инструмент **traceroute** отображает маршрут, по которому пакеты идут к месту назначения. Альтернатива для Windows называется **tracert**. Наблюдение за тем, какой путь проходит сетевой трафик от источника к месту назначения, важно для устранения неполадок, поскольку можно обнаружить и устранить петли маршрутизации и неоптимальные пути.

traceroute использует ICMP-пакеты для определения пути к месту назначения. Поле Time to Live (TTL) в заголовке IP-пакета используется для предотвращения бесконечных петель в сети. Для каждого хопа или маршрутизатора, через который проходит IP-пакет, поле TTL уменьшается на единицу. Когда значение поля TTL достигает 0, пакет отбрасывается, что позволяет избежать бесконечных циклов. Обычно поле TTL устанавливается на максимальное значение, 255, в источнике трафика, поскольку хост пытается максимально увеличить шансы того, что пакет попадет в пункт назначения. traceroute изменяет эту логику и постепенно увеличивает значение TTL,

начиная с 1, и продолжает добавлять 1 к полю TTL в следующем пакете и так далее. Установка значения TTL равного 1 для первого пакета означает, что пакет будет отброшен на первом маршрутизаторе. По умолчанию большинство маршрутизаторов отправляют обратно источнику трафика пакет ICMP Time Exceeded, информирующий его о том, что пакет достиг значения TTL 0 и должен быть отброшен. traceroute использует информацию, полученную от маршрутизатора, для определения его IP-адреса и имени хоста, а также времени прохождения маршрута.

Для IPv6 существует альтернатива под названием traceroute6 для операционных систем на базе UNIX и tracert6 для операционных систем на базе Microsoft Windows.

Шаг 1: Просмотрите параметры traceroute.

Введите **traceroute --help**, чтобы увидеть все доступные параметры команды.

```
devasc@labvm:~$ traceroute --help
Usage: traceroute [OPTION...] HOST
Print the route packets trace to network host.

  -f, --first-hop=NUM        set initial hop distance, i.e., time-to-live
  -g, --gateways=GATES       list of gateways for loose source routing
  -I, --icmp                 use ICMP ECHO as probe
  -m, --max-hop=NUM          set maximal hop count (default: 64)
  -M, --type=METHOD          use METHOD (`icmp' or `udp') for traceroute
                             operations, defaulting to `udp'
  -p, --port=PORT            use destination PORT port (default: 33434)
  -q, --tries=NUM            send NUM probe packets per hop (default: 3)
      --resolve-hostnames    resolve hostnames
  -t, --tos=NUM              set type of service (TOS) to NUM
  -w, --wait=NUM             wait NUM seconds for response (default: 3)
  -?, --help                 give this help list
      --usage                give a short usage message
  -V, --version              print program version

Mandatory or optional arguments to long options are also mandatory or optional
for any corresponding short options.

Report bugs to <bug-inetutils@gnu.org>.
devasc@labvm:~$
```

В **traceroute** также доступны несколько опций, включая:

- Значение TTL первого отправленного пакета, по умолчанию 1.
- Максимальное значение TTL. По умолчанию он будет увеличивать значение TTL до 64 или пока не будет достигнут пункт назначения.
- Адрес источника в случае наличия нескольких интерфейсов на устройстве.
- Значение Quality of Service (QoS) в IP-заголовке.
- Длину пакета.

Шаг 2: Используйте traceroute, чтобы найти путь к веб-серверу.

Из-за того, как Virtual Box реализует NAT-сеть, вы не можете отслеживать за пределами своей виртуальной машины. Вам нужно будет переключить свою виртуальную машину в режим Bridged. Поэтому мы рекомендуем оставить вашу ВМ в режиме NAT.

Однако вы должны иметь возможность использовать команду **traceroute** на локальном хосте. Для компьютеров Mac и Linux используйте команду **traceroute**. Для хостов Windows используйте команду **tracert**, как показано ниже. Откройте командную строку на локальном узле и проследите маршрут до **cloud.yandex.ru**, чтобы узнать, сколько переходов и сколько времени требуется для достижения маршрута. Ваши результаты будут отличаться.

```
C:\> tracert cloud.yandex.ru

Трассировка маршрута к cloud.yandex.ru [87.250.250.108]
с максимальным числом прыжков 30:

  1     1 ms     2 ms    <1 мс  192.168.0.1
  2     4 ms     2 ms     3 ms  bras240-lo0.spb.corbina.net
[85.21.129.73]
  3     *        *        *     Превышен интервал ожидания для запроса.
  4    14 ms    12 ms    47 ms  bmor19-bb-tengige0-3-0-
3.spb.corbina.net [85.21.
225.198]
  5    20 ms   149 ms    10 ms  ko-crs-be5.corbina.net [195.14.54.184]
  6    47 ms   132 ms    52 ms  corbina-gw.dante.yandex.net
[83.102.145.178]
  7    16 ms    49 ms    17 ms  vla-32z5-ae1-1.yndx.net [93.158.172.25]
  8     *        *        *     Превышен интервал ожидания для запроса.
  9   124 ms    25 ms    13 ms  cloud.yandex.ru [87.250.250.108]

Трассировка завершена.

C:\>
```

Результат показывает, что на пути имеется 5 переходов, а также отображается время пути туда и обратно.

Часть 5: Знакомство с инструментом nslookup

Инструмент nslookup используется для запроса системы доменных имен (DNS) с целью получения сопоставления доменных имен с IP-адресами. Этот инструмент полезен для определения того, разрешает ли DNS-сервер, настроенный на определенном хосте, имена хостов в IP-адреса.

Шаг 1: Запрос домена.

Чтобы использовать **nslookup**, необходимо ввести имя хоста, которое вы пытаетесь преобразовать в IP-адрес. Для поиска IP-адреса будет использован настроенный DNS-сервер. Вы также можете указать DNS-сервер для использования.

Использование: nslookup [HOST] [SERVER]

Введите nslookup practicum.yandex.ru, чтобы определить IP-адрес домена.

```
devasc@labvm:~$ nslookup practicum.yandex.ru
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
Name:   practicum.yandex.ru
Address: 87.250.250.5
Name:   practicum.yandex.ru
Address: 2a02:6b8::313

devasc@labvm:~$
```

Команда возвращает неавторитетный ответ, а также имя и адрес IPv4 и IPv6. Неавторитетный ответ означает, что сервер не содержит оригинальных записей зоны домена, а создан на основе предыдущих поисков DNS.

Шаг 2: Запрос IP-адреса.

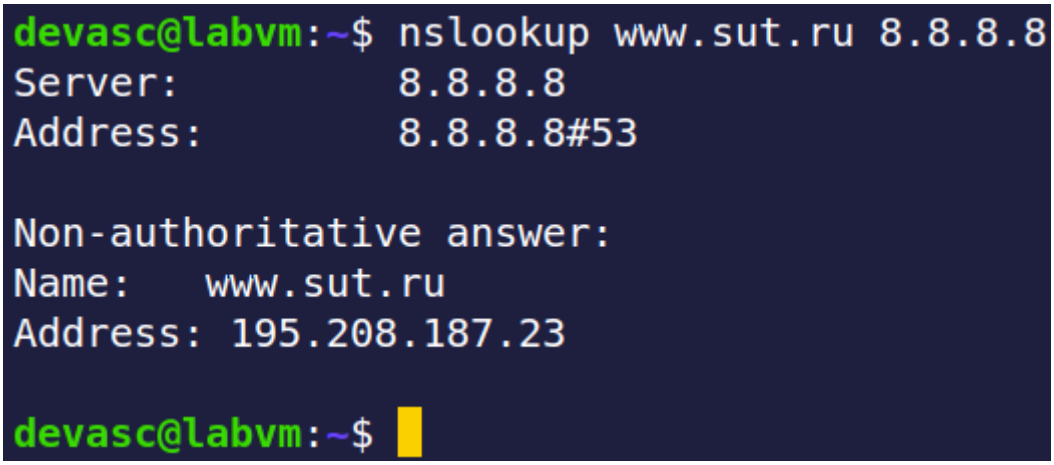
Вы также можете просмотреть IP-адреса, чтобы узнать связанный с ним домен. Запросите у DNS-сервера IP-адрес 8.8.8.8.

```
devasc@labvm:~$ nslookup 8.8.8.8
8.8.8.8.in-addr.arpa      name = dns.google.

Authoritative answers can be found from:

devasc@labvm:~$
```

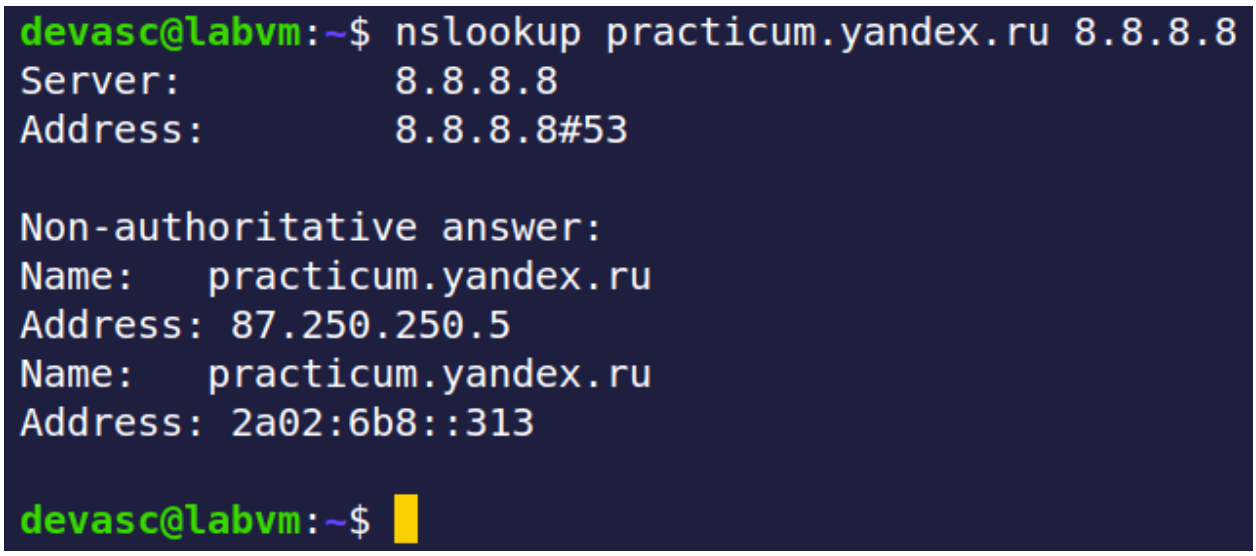
Шаг 3: Запросите домен, используя определенный DNS-сервер.

Введите nslookup www.sut.ru 8.8.8.8, чтобы определить IP-адрес домена в соответствии с DNS Google.

```
devasc@labvm:~$ nslookup www.sut.ru 8.8.8.8
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
Name:   www.sut.ru
Address: 195.208.187.23

devasc@labvm:~$
```



```
devasc@labvm:~$ nslookup practicum.yandex.ru 8.8.8.8
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
Name:   practicum.yandex.ru
Address: 87.250.250.5
Name:   practicum.yandex.ru
Address: 2a02:6b8::313

devasc@labvm:~$
```



Обратите внимание, что при использовании этого метода сервер разрешил адрес на два разных IP-адреса, причем все они отличаются от предыдущего DNS-запроса. Эти серверы имеют разный кэш DNS-запросов practicum.yandex.ru.

Лабораторная работа №6.2.7

Сборка демо веб-приложения в Docker контейнере

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC

Часть 2: Создаём простой Bash скрипт

Часть 3: Создаём демо веб-приложение

Часть 4: Настройка веб-приложения для использования файлов веб-сайта

Часть 5: Создаём Bash скрипт для создания и запуска Docker контейнера

Часть 6: Сборка, запуск и проверка Docker контейнера

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

Порядок выполнения работы

Часть 1: Запуск виртуальной машины DEVASC

Если вы еще не завершили **лабораторную работу - Установка лабораторной среды виртуальной машины**, сделайте это сейчас. Если вы уже завершили эту лабораторную работу, запустите виртуальную машину **DEVASC**.

Часть 2: Создаём простой Bash скрипт

Знание Bash крайне важно для работы с Continuous Integration, Continuous Deployment, контейнерами и со средой разработки. Bash скрипт помогают программистам автоматизировать множество задач в одном файле сценария.

Шаг 1: Создайте пустой файл bash.

Измените рабочий каталог на ~/labs/devnet-src/sample-app и добавьте новый файл **user-input.sh**.

```
devasc@labvm:~$ cd labs/devnet-src/sample-app/
devasc@labvm:~/labs/devnet-src/sample-app$ touch user-input.sh
```

Шаг 2: Откройте файл в текстовом редакторе nano.

Используйте команду **nano**, чтобы открыть текстовый редактор nano.

```
devasc@labvm:~/labs/devnet-src/sample-app$ nano user-input.sh
```

Шаг 3: Добавьте "she-bang" в начало сценария.

Отсюда вы можете вводить команды для вашего сценария bash. Используйте клавиши со стрелками для навигации в nano. Обратите внимание на команды внизу (здесь не показаны) для управления файлом. Символ карата (^) указывает на использование клавиши CTRL или Command на клавиатуре. Например, чтобы выйти из nano, введите CTRL+X.

Добавьте 'she-bang', который сообщает системе, что этот файл включает команды, которые необходимо выполнить в оболочке bash.

```
#!/bin/bash
```

Примечание: Вы можете использовать графический текстовый редактор или открыть файл с помощью VS Code. Однако будет здорово, иметь опыт работы с текстовыми редакторами командной строки, такими как nano и vim.

Шаг 4: Добавьте bash скрипт простые команды.

Введите несколько простых команд bash. Следующие команды будут запрашивать у пользователя имя, устанавливать это имя в переменную **userName** и выводить на экран строку текста с именем пользователя.

```
echo -n "Введите ваше имя: "
read userName
echo "Ваше имя — $userName."
```

Шаг 5: Выйдите из редактора nano и сохраните скрипт.

Нажмите **CTRL+X**, затем **Y**, затем **ENTER**, чтобы выйти из **nano** и сохранить скрипт.

Шаг 6: Запустите свой скрипт из командной строки.

Вы можете запустить его непосредственно из командной строки с помощью следующей команды.

```
devasc@labvm:~/labs/devnet-src/sample-app$ bash user-input.sh
Введите ваше имя: Impostor
Ваше имя — Impostor.
devasc@labvm:~/labs/devnet-src/sample-app$
```

```
devasc@labvm:~/labs/devnet-src/sample-app$ bash user-input.sh
Введите ваше имя: Amogus
Ваше имя — Amogus.
devasc@labvm:~/labs/devnet-src/sample-app$
```

Шаг 7: Измените режим скрипта на исполняемый файл для всех пользователей.

Измените режим скрипта на исполняемый с помощью команды **chmod**. Установите параметры **a+x**, чтобы сделать скрипт исполняемым (**x**) для всех пользователей (**a**). После использования команды **chmod** обратите внимание, что разрешения были изменены для пользователей, групп и других, чтобы включить "x" (исполняемый).

```
devasc@labvm:~/labs/devnet-src/sample-app$ ls -l user-input.sh
-rw-rw-r-- 1 devasc devasc 108 мая  4 09:46 user-input.sh

devasc@labvm:~/labs/devnet-src/sample-app$ chmod a+x user-input.sh

devasc@labvm:~/labs/devnet-src/sample-app$ ls -l user-input.sh
-rwxrwxr-x 1 devasc devasc 108 мая  4 09:46 user-input.sh
```

Шаг 8: Переименуйте файл, чтобы удалить расширение .sh.

Вы можете переименовать файл, удалив расширение, чтобы пользователям не приходилось добавлять .sh к команде для выполнения сценария.

```
devasc@labvm:~/labs/devnet-src/sample-app$ mv user-input.sh user-input
```

Шаг 9: Запустите скрипт из командной строки.

Теперь сценарий можно запустить из командной строки без команды source или расширения. Чтобы запустить сценарий bash без исходной команды, вы должны предварять сценарий символом "./".

```
devasc@labvm:~/labs/devnet-src/sample-app$ ./user-input
Введите ваше имя: Sus
Ваше имя —  Bob.
devasc@labvm:~/labs/devnet-src/sample-app$
```

```
devasc@labvm:~/labs/devnet-src/sample-app$ ./user-input
Введите ваше имя: Sus
Ваше имя — Sus.
devasc@labvm:~/labs/devnet-src/sample-app$
```

Часть 3: Создаём демо веб-приложение

Прежде чем запускать приложение в Docker контейнере, нам сначала нужно это приложение. В этой части мы создадим очень простой сценарий на Python, который будет отображать IP-адрес клиента при посещении им веб-страницы.

Шаг 1: Установите Flask и откройте порт на брандмауэре DEVASC VM.

Разработчики веб-приложений, использующие Python, обычно используют фреймворк. Фреймворк — это библиотека кода, облегчающая разработчикам создание надежных, масштабируемых и поддерживаемых веб-приложений. **Flask** – это фреймворк для веб-приложений, написанный на языке Python. Другие фреймворки включают Tornado и Pyramid.

Вы будете использовать этот фреймворк для создания примера веб-приложения. Flask получает запросы, а затем предоставляет ответ пользователю в веб-приложении. Это полезно для динамических веб-приложений, поскольку позволяет взаимодействовать с пользователем и создавать динамический контент. Что делает ваше демо веб-приложение динамическим, так это то, что он будет показывать IP-адрес клиента.

Откройте окно терминала и устанавите flask.

```
devasc@labvm:~/labs/devnet-src/sample-app$ pip3 install flask
devasc@labvm:~/labs/devnet-src/sample-app$ pip3 install flask
Requirement already satisfied: flask in
/home/devasc/.local/lib/python3.8/site-packages (1.1.2)
Requirement already satisfied: Werkzeug>=0.15 in
/home/devasc/.local/lib/python3.8/site-packages (from flask) (1.0.1)
Requirement already satisfied: Jinja2>=2.10.1 in
/home/devasc/.local/lib/python3.8/site-packages (from flask) (2.11.2)
Requirement already satisfied: click>=5.1 in
/home/devasc/.local/lib/python3.8/site-packages (from flask) (7.1.2)
Requirement already satisfied: itsdangerous>=0.24 in
/home/devasc/.local/lib/python3.8/site-packages (from flask) (1.1.0)
Requirement already satisfied: MarkupSafe>=0.23 in
/home/devasc/.local/lib/python3.8/site-packages (from Jinja2>=2.10.1-
>flask) (1.1.1)
devasc@labvm:~/labs/devnet-src/sample-app$
```

Шаг 2: Откройте файл sample_app.py.

Откройте **файл sample_app.py**, расположенный в каталоге /sample-app. Вы можете сделать это внутри VS Code или использовать текстовый редактор командной строки, например, nano или vim.

Шаг 3: Импортируйте flask.

Добавьте следующие команды для импорта необходимых методов из библиотеки flask.

```
from flask import Flask
from flask import request
```

Шаг 4: Создайте экземпляр класса Flask.

Создайте экземпляр класса Flask и назовите его sample. Обязательно используйте два символа подчеркивания до и после "name".

```
sample = Flask(__name__)
```

Шаг 5: Определите метод показывающий IP-адреса клиента.

Далее настройте Flask так, чтобы при посещении пользователем страницы по умолчанию (корневой каталог) выводилось сообщение с IP-адресом клиента.

```
@sample.route("/")
def main():
    return "Ты заходишь на меня с " + request.remote_addr + "\n"
```

Обратите внимание на анотацию @sample.route("/") Flask. Такие фреймворки, как Flask, используют технику маршрутизации (.route) для ссылки на URL приложения (это не следует путать с сетевой маршрутизацией). Здесь "/" (корневой каталог) привязан к функции main(). Таким образом, когда пользователь перейдет по URL http://localhost:8080/ (корневой каталог), в браузере будет отображен результат оператора возврата.

Шаг 6: Настройте приложение для локального запуска.

Наконец, настройте Flask на локальный запуск приложения по адресу http://0.0.0.0:8080, который также является http://localhost:8080. Обязательно используйте два знака подчеркивания до и после "name", а также до и после "main".

```
if __name__ == "__main__":
    sample.run(host="0.0.0.0", port=8080)
```

Шаг 7: Сохраните и запустите ваше демо веб-приложение.

Сохраните скрипт и запустите его из командной строки. Вы должны увидеть следующий результат, который указывает на то, что ваш сервер "sample-app" запущен. Если вы не видите следующего вывода или получаете сообщение об ошибке, внимательно проверьте ваш скрипт sample_app.py.

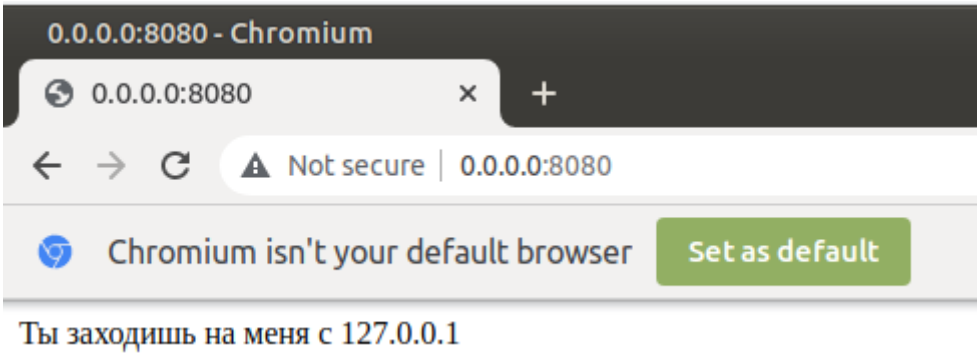
```
devasc@labvm:~/labs/devnet-src/sample-app$ python3 sample_app.py
 * Serving Flask app "sample-app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
```

```
devasc@labvm:~/labs/devnet-src/sample-app$ python3 sample_app.py
 * Serving Flask app "sample_app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production dep
loyment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
```

Шаг 8: Убедитесь, что сервер запущен.

Вы можете проверить работу сервера одним из двух способов.

- Откройте веб-браузер Chromium и введите 0.0.0.0:8080 в поле URL. Вы должны получить следующий результат:



  Если вы получаете ответ "HTTP 400 Bad Request", внимательно проверьте свой скрипт **sample_app.py**.

- Откройте другое окно терминала и используйте инструмент URL командной строки (cURL) для проверки ответа сервера

```
devasc@labvm:~/labs/devnet-src/sample-app$ curl http://0.0.0.0:8080
Ты заходишь на меня с 127.0.0.1
devasc@labvm:~/labs/devnet-src/sample-app$
```

Шаг 9: Остановите сервер.

Вернитесь в окно терминала, в котором запущен сервер, и нажмите **CTRL+C**, чтобы остановить сервер.

Часть 4: Настройка веб-приложения для использования файлов веб-сайта

В этой части мы создадим демо веб-приложение, содержащую **index.html** и **style.css**. index.html обычно является первой страницей, загружаемой в браузере клиента при посещении вашего сайта. **style.css** - стили, используемые для красоты веб-страницы.

Шаг 1: Изучите каталоги, которые будут использоваться веб-приложением.

Каталоги **templates** и **static** уже находятся в каталоге sample-app. Откройте **index.html** и **style.css**, чтобы просмотреть их содержимое. Если вы знакомы с HTML и CSS, не стесняйтесь настраивать эти каталоги и файлы как угодно. Однако убедитесь, что вы сохранили встроенный **{{request.remote_addr}}** Python-код в файле **index.html**, поскольку это динамический аспект примера веб-приложения.

```
devasc@labvm:~/labs/devnet-src/sample-app$ cat templates/index.html
<html>
<head>
    <title>Sample app</title>
    <link rel="stylesheet" href="/static/style.css" />
</head>
<body>
    <h1>Ты заходишь на меня с {{request.remote_addr}}</h1>
</body>
</html>
devasc@labvm:~/labs/devnet-src/sample-app$ cat static/style.css
```

```
body {background: lightsteelblue;}
devasc@labvm:~/labs/devnet-src/sample-app$
```

Шаг 2: Обновите демо веб-приложение.

Теперь, когда вы изучили основные файлы веб-сайта, необходимо обновить файл **sample_app.py**, чтобы он отображал файл index.html, а не просто возвращал данные. Генерирование HTML-контента с помощью кода Python может быть громоздким, особенно при использовании условных операторов или повторяющихся структур. HTML-файл может быть отрисован во Flask автоматически с помощью функции render_template. Для этого необходимо импортировать метод render_template из библиотеки flask и внести правки в возвращаемую функцию. Внесите выделенные правки в ваш скрипт.

```
from flask import Flask
from flask import request
from flask import render_template

sample = Flask(__name__)

@sample.route("/")
def main():
    return render_template("index.html")

if __name__ == "__main__":
    sample.run(host="0.0.0.0", port=8080)
```

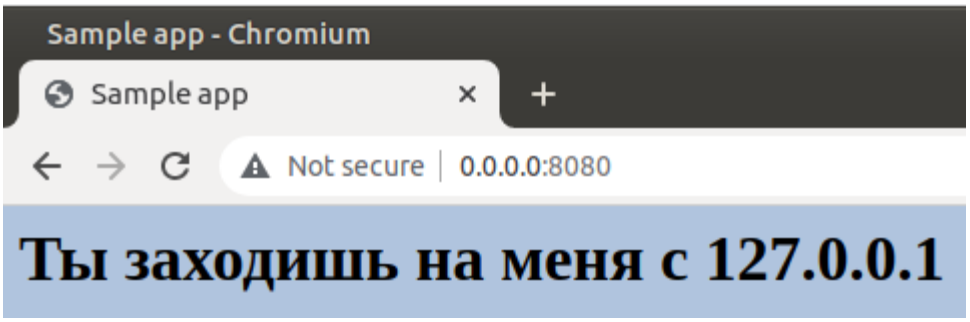
Шаг 3: Сохраните и запустите свой скрипт.

Сохраните и запустите **sample_app.py**. Вы должны получить результат, подобный следующему:

```
devasc@labvm:~/labs/devnet-src/sample-app$ python3 sample_app.py
 * Serving Flask app "sample-app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
```

Шаг 4: Убедитесь, что ваша программа запущена.

Опять же, вы можете проверить, что ваша программа запущена, одним из двух способов.



Шаг 5: Остановите сервер.

Вернитесь в окно терминала, в котором запущен сервер, и нажмите **CTRL+C**, чтобы остановить сервер.

Часть 5: Создаём Bash скрипт для создания и запуска Docker контейнера

Приложение может быть развернуто на сервере (физический сервер, предназначенный для однопользовательской среды) или на виртуальной машине, как вы только что сделали в предыдущей части. Оно также может быть развернуто в контейнерном решении, таком как Docker. В этой части мы создадим bash скрипт, который создаёт и запускает Docker контейнер.

Шаг 1: Создайте временные директории для хранения файлов сайта.

Откройте bash скрипт **sample-app.sh** в каталоге ~/labs/devnet-src/sample-app. Добавьте 'she-bang' и команды для создания структуры каталогов с tempdir в качестве родительской папки.

```
#!/bin/bash

mkdir tempdir
mkdir tempdir/templates
mkdir tempdir/static
```

Шаг 2: Скопируйте каталоги сайта и sample_app.py во временную директорию.

В файл sample-app.sh добавьте команды для копирования каталога сайта и скрипта в tempdir.

```
cp sample_app.py tempdir/.
cp -r templates/* tempdir/templates/.
cp -r static/* tempdir/static/.
```

Шаг 3: Создайте Dockerfile.

На этом шаге мы добавим необходимые команды bash echo в файл sample-app.sh для создания Dockerfile в tempdir. Этот Dockerfile будет использоваться для сборки контейнера.

Вам нужен Python, запущенный в контейнере, поэтому добавьте команду Docker FROM для установки Python в контейнер.

```
echo "FROM python" >> tempdir/Dockerfile
```

Вашему приложению sample_app.py нужен Flask, поэтому добавьте команду Docker RUN для установки Flask в контейнер.

```
echo "RUN pip install flask" >> tempdir/Dockerfile
```

Вашему контейнеру понадобятся каталоги сайта и sample_app.py для запуска приложения, поэтому добавьте команды Docker COPY, чтобы добавить их в каталог в контейнере Docker. В этом примере вы создадите /home/myapp в качестве родительского каталога внутри контейнера Docker. Помимо копирования файла sample_app.py в Dockerfile, вы также скопируете файл index.html из каталога templates и файл style.css из каталога static.

```
echo "COPY  ./static /home/myapp/static/" >> tempdir/Dockerfile
echo "COPY  ./templates /home/myapp/templates/" >> tempdir/Dockerfile
echo "COPY  sample_app.py /home/myapp/" >> tempdir/Dockerfile
```

С помощью команды Docker **EXPOSE** откройте порт 8080 для использования веб-сервером.

```
echo "EXPOSE 8080" >> tempdir/Dockerfile
```

Наконец, добавьте команду Docker **CMD** для выполнения сценария Python.

```
echo "CMD python3 /home/myapp/sample_app.py" >> tempdir/Dockerfile
```

Шаг 4: Соберите Docker контейнер.

Добавьте команды в файл sample-app.sh для перехода в каталог tempdir и сборки контейнера Docker. Опция -t команды docker build позволяет указать имя контейнера, а задняя точка (.) указывает, что вы хотите, чтобы контейнер был собран в текущем каталоге.

```
cd tempdir
docker build -t sampleapp .
```

Шаг 5: Запустите контейнер и убедитесь, что он работает.

Добавьте команду **docker run** в файл **sample-app.sh**, чтобы запустить контейнер.

```
docker run -t -d -p 8080:8080 --name samplerunning sampleapp
```

В параметрах запуска docker указано следующее:

- **-t** указывает, что для контейнера создаётся терминал, для доступа к нему через командную строку.
- **-d** указывает, что контейнер будет работать в фоновом режиме, а также печатает container ID при выполнении команды **docker ps -a**.
- **-p** указывает, что вы хотите опубликовать внутренний порт контейнера на хосте. Первое "8080" указывает на порт приложения, запущенного в контейнере docker (наше демо). Второе "8080" указывает docker использовать этот порт на хосте. Эти значения не обязательно должны быть одинаковыми. Например, внутренний порт 80 на внешний 800 (80:800).
- **--name** указывает сначала то, как вы хотите назвать экземпляр контейнера (samplerunning), а затем образ контейнера, на котором будет основан экземпляр (sampleapp). Имя экземпляра может быть любым. Однако имя образа должно соответствовать имени контейнера, которое вы указали в команде сборки docker (sampleapp).

Добавьте команду **docker ps -a** для отображения всех запущенных в данный момент контейнеров Docker. Эта команда будет последней, выполняемой bash скриптом.

```
docker ps -a
```

Шаг 6: Сохраните ваш bash скрипт.

Часть 6: Сборка, запуск и проверка Docker контейнера

В этой части мы запустим bash скрипт, который создаст каталоги, скопирует файлы, создаст Dockerfile, соберёт Docker контейнер, запустит экземпляр контейнера Docker и отобразит вывод команды docker ps -a с подробной

информацией о запущенном контейнере. Затем вы исследуете контейнер Docker, остановите его работу и удалите контейнер.

Примечание: Убедитесь, что вы остановили все другие процессы веб-сервера, которые могли быть запущены в предыдущих частях этой лабораторной работы.

Шаг 1: Запустите bash-скрипт.

Запустите bash-скрипт из командной строки. Вы должны увидеть вывод, похожий на следующий. После создания каталогов tempdir сценарий выполняет команды для создания контейнера Docker. Обратите внимание, что шаг 7/7 в выводе выполняет sample_app.py, который создает веб-сервер. Также обратите внимание на идентификатор контейнера. Вы увидите его в командной строке Docker позже в лабораторной работе.

```
devasc@labvm:~/labs/devnet-src/sample-app$ bash ./sample-app.sh
Sending build context to Docker daemon  6.144kB
Step 1/7 : FROM python
latest: Pulling from library/python
90fe46dd8199: Pulling fs layer
35a4f1977689: Pulling fs layer
bbc37f14aded: Pull complete
74e27dc593d4: Pull complete
4352dcff7819: Pull complete
deb569b08de6: Pull complete
98fd06fa8c53: Pull complete
7b9cc4fdefe6: Pull complete
512732f32795: Pull complete
Digest: sha256:ad7fb5bb4770e08bf10a895ef64a300b288696a1557a6d02c8b6fba98984b86a
Status: Downloaded newer image for python:latest
 ---> 4f7cd4269fa9
Step 2/7 : RUN pip install flask
 ---> Running in 32d28026afea
Collecting flask
  Downloading Flask-1.1.2-py2.py3-none-any.whl (94 kB)
Collecting click>=5.1
  Downloading click-7.1.2-py2.py3-none-any.whl (82 kB)
Collecting Jinja2>=2.10.1
  Downloading Jinja2-2.11.2-py2.py3-none-any.whl (125 kB)
Collecting Werkzeug>=0.15
  Downloading Werkzeug-1.0.1-py2.py3-none-any.whl (298 kB)
Collecting itsdangerous>=0.24
  Downloading itsdangerous-1.1.0-py2.py3-none-any.whl (16 kB)
Collecting MarkupSafe>=0.23
  Downloading MarkupSafe-1.1.1-cp38-cp38-manylinux1_x86_64.whl (32 kB)
Installing collected packages: click, MarkupSafe, Jinja2, Werkzeug, itsdangerous,
flask
Successfully installed Jinja2-2.11.2 MarkupSafe-1.1.1 Werkzeug-1.0.1 click-7.1.2
flask-1.1.2 itsdangerous-1.1.0
Removing intermediate container 32d28026afea
 ---> 619aee23fd2a
Step 3/7 : COPY  ./static /home/myapp/static/
 ---> 15fac1237eec
Step 4/7 : COPY  ./templates /home/myapp/templates/
 ---> dc807b5cf615
```

```
Step 5/7 : COPY  sample_app.py /home/myapp/
 ---> d4035a63ae14
Step 6/7 : EXPOSE 8080
 ---> Running in 40c2d35aa29a
Removing intermediate container 40c2d35aa29a
 ---> eb789099a678
Step 7/7 : CMD python3 /home/myapp/sample_app.py
 ---> Running in 41982e2c6209
Removing intermediate container 41982e2c6209
 ---> a2588e9b0593
Successfully built a2588e9b0593
Successfully tagged sampleapp:latest
8953a95374ff8ebc203059897774465312acc8f0ed6abd98c4c2b04448a56ba5
CONTAINER ID          IMAGE                   COMMAND                      CREATED
STATUS                  PORTS                     NAMES
f674aad75d9e          sampleapp             "/bin/sh -c 'python …"    1 second ago
Up Less than a second   0.0.0.0:8080->8080/tcp   samplerunning
devasc@labvm:~/labs/devnet-src/sample-app$
```

Шаг 2: Исследуйте запущенный Docker контейнер и веб-приложение.

Создание каталогов tempdir не отображается в выводе. Вы можете добавить команду echo для вывода сообщений об успешном создании каталогов. Вы также можете убедиться в их наличии с помощью команды ls. Помните, что в этом каталоге находятся файлы и папки, используемые для создания контейнера и запуска веб-приложения. Это не контейнер, который был создан.

```
devasc@labvm:~/labs/devnet-src/sample-app$ ls tempdir/
Dockerfile  sample_app.py  static  templates
devasc@labvm:~/labs/devnet-src/sample-app$
```

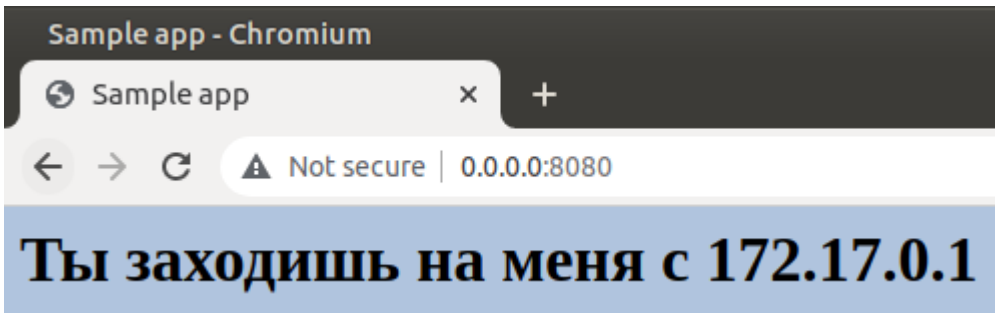
Обратите внимание на Dockerfile, созданный вашим сценарием bash. Откройте этот файл, чтобы посмотреть, как он выглядит в окончательном виде без команд **echo**.

```
devasc@labvm:~/labs/devnet-src/sample-app$ cat tempdir/Dockerfile
FROM python
RUN pip install flask
COPY  ./static /home/myapp/static/
COPY  ./templates /home/myapp/templates/
COPY  sample_app.py /home/myapp/
EXPOSE 8080
CMD python3 /home/myapp/sample_app.py
```

Вывод команды **docker ps -a** может быть трудночитаемым в зависимости от ширины экрана вашего терминала. Вы можете перенаправить его в текстовый файл, где его можно будет лучше просмотреть без обводки слов.

```
devasc@labvm:~/labs/devnet-src/sample-app$ docker ps -a >> running.txt
devasc@labvm:~/labs/devnet-src/sample-app$
```

Контейнер Docker создает свой собственный IP-адрес из адресного пространства частной сети. Убедитесь, что веб-приложение запущено и сообщает IP-адрес. В веб-браузере по адресу http://localhost:8080 вы должны увидеть сообщение Ты заходишь на меня с 172.17.0.1, отформатированное как H1 на светло-стальном голубом фоне.



При желании вы также можете использовать команду curl.

```
devasc@labvm:~/labs/devnet-src/sample-app$ curl http://172.17.0.1:8080
<html>
<head>
    <title>Sample app</title>
    <link rel="stylesheet" href="/static/style.css" />
</head>
<body>
    <h1>Ты заходишь на меня с 172.17.0.1</h1>
</body>
</html>devasc@labvm:~/labs/devnet-src/sample-app$
devasc@labvm:~/labs/devnet-src/sample-app$
```

По умолчанию Docker использует подсеть IPv4 172.17.0.0/16 для сети контейнеров. (При необходимости этот адрес можно изменить.) Введите команду ip address, чтобы отобразить все IP-адреса, используемые вашим экземпляром виртуальной машины DEVASC. Вы должны увидеть loopback-адрес 127.0.0.1, который веб-приложение использовало ранее в лабораторной работе, и новый интерфейс Docker с IP-адресом 172.17.0.1.

```
devasc@labvm:~/labs/devnet-src/sample-app$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
<вывод опущен>
4: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default
    link/ether 02:42:c2:d1:8a:2d brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:c2ff:fed1:8a2d/64 scope link
       valid_lft forever preferred_lft forever
<вывод опущен>
```

Шаг 3: Получите доступ и исследуйте работающий контейнер.

Помните, что контейнер Docker — это способ инкапсуляции всего необходимого для запуска приложения, чтобы его можно было легко развернуть в различных средах — не только в вашей виртуальной машине DEVASC.

Чтобы получить доступ к работающему контейнеру, введите команду **docker exec -it**, указав имя работающего контейнера (**samplerunning**) и то, что вам нужна оболочка bash (/bin/bash). Опция **-i** указывает, что вы хотите, чтобы он был интерактивным, а опция -t указывает, что вы хотите получить терминальный доступ. Приглашение изменится на **root@containerID**. ID вашего контейнера будет отличаться от показанного ниже. Обратите внимание, что ID контейнера совпадает с ID, показанным в выводе docker ps -a.

```
devasc@labvm:~/labs/devnet-src/sample-app$ docker exec -it
samplerunning /bin/bash
root@f674aad75d9e:/#
```

Теперь у вас есть доступ root к Docker контейнеру samplerunning. Отсюда вы можете использовать команды Linux для изучения контейнера Docker. Введите ls, чтобы просмотреть структуру каталогов на корневом уровне.

```
root@f674aad75d9e:/# ls
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
root@f674aad75d9e:/#
```

Вспомните, что в своём bash скрипте мы добавили команды в Dockerfile, которые скопировали каталоги и файлы вашего приложения в каталог home/myapp. Введите команду ls еще раз для этой папки, чтобы увидеть сценарий sample_app.py и каталоги. Чтобы лучше понять, что включено в ваш контейнер Docker, вы можете использовать команду ls для изучения других каталогов, таких как /etc и /bin.

```
root@f674aad75d9e:/# ls home/myapp/
sample_app.py  static  templates
root@f674aad75d9e:/#
```

Выйдите из контейнера Docker, чтобы вернуться в командную строку DEVASC VM.

```
root@f674aad75d9e:/# exit
exit
devasc@labvm:~/labs/devnet-src/sample-app$
```

Шаг 4: Остановите и удалите Docker контейнер.

Вы можете остановить Docker контейнер с помощью команды **docker stop**, указав имя запущенного контейнера. Потребуется несколько секунд для очистки и кэширования контейнера. Вы можете убедиться, что он все ещё существует, введя команду **docker ps -a**. Однако, если вы обновите веб-страницу http://localhost:8080, вы увидите, что веб-приложение больше не запущено.

```
devasc@labvm:~/labs/devnet-src/sample-app$ docker stop samplerunning
samplerunning
devasc@labvm:~/labs/devnet-src/sample-app$ docker ps -a
CONTAINER ID        IMAGE               COMMAND
CREATED             STATUS                          PORTS
NAMES
```

```
f674aad75d9e        sampleapp           "/bin/sh -c 'python3…"   11
minutes ago       Exited (137) 5 seconds ago
samplerunning
devasc@labvm:~/labs/devnet-src/sample-app$
```

Остановленный контейнер можно перезапустить с помощью команды **docker start**. Контейнер немедленно запустится.

```
devasc@labvm:~/labs/devnet-src/sample-app$ docker start samplerunning
samplerunning
devasc@labvm:~/labs/devnet-src/sample-app$
```

Чтобы окончательно удалить контейнер, сначала остановите его, а затем удалите с помощью команды **docker rm**. Вы всегда можете восстановить его снова, выполнив программу **sample-app**. Для проверки удаления контейнера используйте команду **docker ps -a**.

```
devasc@labvm:~/labs/devnet-src/sample-app$ docker stop samplerunning
samplerunning
devasc@labvm:~/labs/devnet-src/sample-app$ docker rm samplerunning
samplerunning
devasc@labvm:~/labs/devnet-src/sample-app$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
devasc@labvm:~/labs/devnet-src/sample-app$
```

Лабораторная работа №6.3.6

Построение CI/CD Пайплайна с помощью Jenkins

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC

Часть 2: Закоммитьте Sample App на Git

Часть 3: Изменение Sample App и внесение изменений в Git

Часть 4: Загрузка и запуск Jenkins Docker Image

Часть 5: Конфигурация Jenkins

Часть 6: Использование Jenkins для запуска сборки вашего приложения

Часть 7: Использование Jenkins для тестирования сборки

Часть 8: Создание Пайплайна в Jenkins

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

Порядок выполнения работы

Часть 1: Запуск виртуальной машины DEVASC

Если вы еще не завершили **лабораторную работу - Установка лабораторной среды виртуальной машины**, сделайте это сейчас. Если вы

уже завершили эту лабораторную работу, запустите виртуальную машину **DEVASC**.

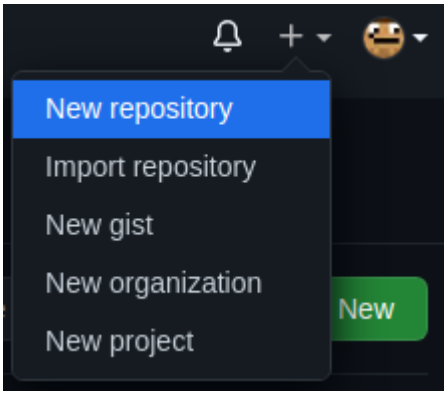
Часть 2: Закоммитьте Sample App на Git

В этой части вы создадите репозиторий GitHub для коммита файлов Sample App, созданного в 6.2.7. Вы создали учетную запись GitHub в предыдущей части. Если вы еще не сделали этого, посетите сайт github.com и создайте учетную запись.

Шаг 1: Войдите на GitHub и создайте новый репозиторий.

Войдите на сайт https://github.com/, используя свои учетные данные.

Выберите кнопку "New repository" или нажмите на значок "+" в правом верхнем углу и выберите "New repository".

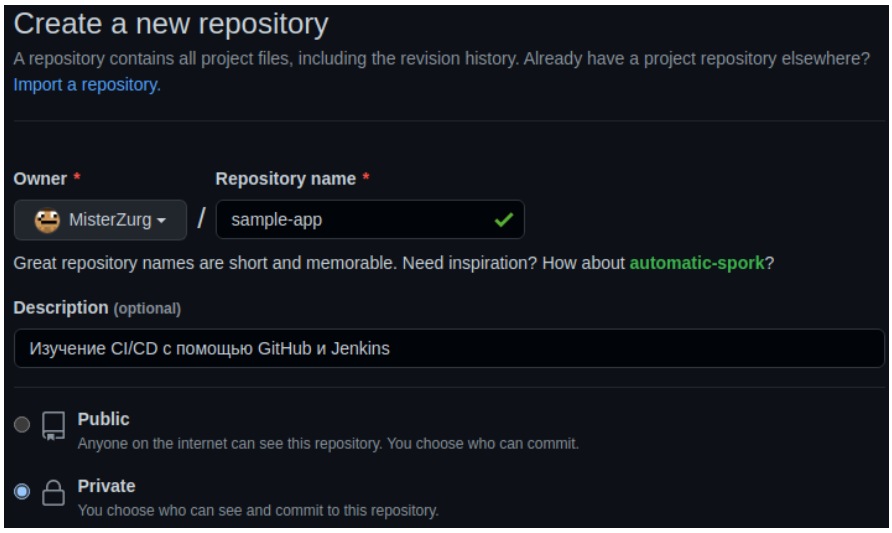


Создайте репозиторий, используя следующую информацию:

Имя репозитория: **sample-app**

Описание: **Изучение CI/CD с помощью GitHub и Jenkins**

Видимость: **Private**



Примечание: Дополнительно можете выбрать чек бокс Add a README file в меню Initialize this repository withng, для более подробной информации зачем вам таковой.

Нажмите **Create repository**

Шаг 2: Настройте учетные данные Git локально на виртуальной машине.

Откройте окно терминала с VS Code в DEVASC VM. Используйте свое имя вместо "Sample User" для имени в кавычках " ". Используйте @example.com для адреса электронной почты.

```
devasc@labvm:~$ git config --global user.name "Sample User"
devasc@labvm:~$ git config --global user.email sample@example.com
```

Шаг 3: Инициализируйте каталог в качестве репозитория Git.

Мы будем использовать файлы sample-app, которые вы создали в предыдущей лабораторной работе. Однако эти файлы также хранятся для вашего удобства в каталоге /labs/devnet-src/jenkins/sample-app. Перейдите в каталог jenkins/sample-app и инициализируйте его как Git-репозиторий.

```
devasc@labvm:~$ cd labs/devnet-src/jenkins/sample-app/
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ git init
Initialized empty Git repository in /home/devasc/labs/devnet-src/jenkins/sample-app/.git/
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$
```

Шаг 4: Свяжите локальный Git-репозиторий с репозиторием на GitHub.

С помощью команды **git remote add origin <URL>** свяжем локальный репозиторий, со вновь созданный репозиторий на GitHub. Используя URL репозитория Git, который вы создали в Шаге 1, вам нужно только заменить github-username в следующей команде на ваше имя пользователя GitHub.

```
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ git remote add origin https://github.com/github-username/sample-app.git
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$
```

Шаг 5: Застэйджите, закоммитьте и переместите файлы sample-app в репозиторий GitHub.

Используйте команду git add для стэджа файлов в каталог **jenkins/sample-app**. Используйте аргумент "звездочка" (*) для постановки всех файлов в текущем каталоге.

```
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ git add *
```

Используйте команду **git status**, чтобы увидеть файлы и каталоги, которые поставлены и готовы к коммиту в вашем репозитории GitHub.

```
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   sample-app.sh
        new file:   sample_app.py
        new file:   static/style.css
        new file:   templates/index.html

devasc@labvm:~/labs/devnet-src/jenkins/sample-app$
```

```
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ git add *
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   sample-app.sh
        new file:   sample_app.py
        new file:   static/style.css
        new file:   templates/index.html

devasc@labvm:~/labs/devnet-src/jenkins/sample-app$
```

Используйте команду **git commit** для коммита стэджовых файлов и начала отслеживания изменений. Добавьте сообщение по своему выбору или используйте предоставленное здесь.

```
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ git commit -m
"Коммит файлов sample-app."
[master 4030ab6] Коммит файлов sample-app.
 4 files changed, 45 insertions(+)
 create mode 100644 sample-app.sh
 create mode 100644 sample_app.py
 create mode 100644 static/style.css
 create mode 100644 templates/index.html
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$
```

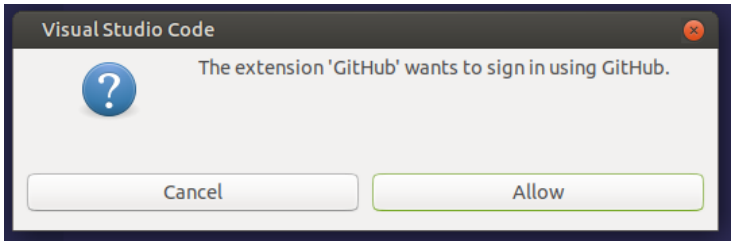
```
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ git commit -m "Коммит фа
йлов sample-app."
[master 9fa4b30] Коммит файлов sample-app.
 4 files changed, 45 insertions(+)
 create mode 100644 sample-app.sh
 create mode 100644 sample_app.py
 create mode 100644 static/style.css
 create mode 100644 templates/index.html
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$
```

Используйте команду **git push**, чтобы перенести локальные файлы sample-app в ваш репозиторий GitHub.

```
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ git push origin
master
Username for 'https://github.com': username
Password for 'https://MisterZurg@github.com': password
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 2 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (8/8), 1.05 KiB | 1.05 MiB/s, done.
Total 8 (delta 0), reused 0 (delta 0)
To https://github.com/MisterZurg/sample-app.git
   d0ee14a..4030ab6  master -> master
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$
```

После чего в браузере вам предолжат ввести ваши учётные данные.

Примечание: Если вместо запроса имени пользователя вы получаете сообщение от VS Code с текстом: "Расширение 'Git' хочет войти, используя GitHub", значит, вы неправильно настроили учетные данные GitHub в Шаге 2 и/или URL GitHub в Шаге 4. URL должен содержать правильное имя пользователя с учетом регистра и имя репозитория, который вы создали в Шаге 1. Чтобы отменить предыдущую команду **git add**, выполните команду git remote rm origin. Затем вернитесь к Шагу 2, убедившись, что ввели правильные учетные данные, а в Шаге 4 введите правильный URL.



Примечание: Если после ввода имени пользователя и пароля вы получаете фатальную ошибку о том, что репозиторий не найден, скорее всего, вы ввели неверный URL. Вам необходимо отменить команду git add командой git remote rm origin.

Часть 3: Изменение Sample App и внесение изменений в Git

В части 4 вы установите образ Jenkins Docker, который будет использовать порт 8080. Вспомните, что ваши файлы sample-app также указывают порт 8080. Сервер Flask и сервер Jenkins не могут одновременно использовать 8080.

В этой части вы измените номер порта, используемого файлами образца-приложения, запустите образец-приложение снова, чтобы убедиться, что оно работает на новом порту, а затем перенесите изменения в свой репозиторий GitHub.

Шаг 1: Откройте файлы sample-app.

Убедитесь, что вы все еще находитесь в каталоге **~/labs/devnet-src/jenkins/sample-app**, поскольку именно эти файлы связаны с вашим репозиторием GitHub. Откройте оба файла **sample_app.py** и **sample-app.sh** для редактирования.

Шаг 2: Отредактируйте файлы sample-app.

В файле sample_app.py измените один экземпляр порта 8080 на 5050, как показано ниже.

```
from flask import Flask
from flask import request
from flask import render_template

sample = Flask(__name__)

@sample.route("/")
def main():
    return render_template("index.html")

if __name__ == "__main__":
```

```
    sample.run(host="0.0.0.0", port=5050)
```

В файле sample-app.sh измените три экземпляра порта 8080 на 5050, как показано ниже.

```
#!/bin/bash

mkdir tempdir
mkdir tempdir/templates
mkdir tempdir/static

cp sample_app.py tempdir/.
cp -r templates/* tempdir/templates/.
cp -r static/* tempdir/static/.

echo "FROM python" >> tempdir/Dockerfile
echo "RUN pip install flask" >> tempdir/Dockerfile
echo "COPY  ./static /home/myapp/static/" >> tempdir/Dockerfile
echo "COPY  ./templates /home/myapp/templates/" >> tempdir/Dockerfile
echo "COPY  sample_app.py /home/myapp/" >> tempdir/Dockerfile
echo "EXPOSE 5050" >> tempdir/Dockerfile
echo "CMD python3 /home/myapp/sample_app.py" >> tempdir/Dockerfile

cd tempdir

docker build -t sampleapp .

docker run -t -d -p 5050:5050 --name samplerunning sampleapp
docker ps -a
```

Шаг 3: Соберете и проверьте sample-app.

Введите команду bash для создания вашего приложения с использованием нового порта 5050.

```
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ bash ./sample-app.sh
Sending build context to Docker daemon  6.144kB
Step 1/7 : FROM python
 ---> 2b7ca628da40
Step 2/7 : RUN pip install flask
 ---> Using cache
 ---> 4e39a3a408ea
Step 3/7 : COPY  ./static /home/myapp/static/
 ---> Using cache
 ---> 30711f334fd9
Step 4/7 : COPY  ./templates /home/myapp/templates/
 ---> Using cache
 ---> b7a429dbaaea
Step 5/7 : COPY  sample_app.py /home/myapp/
 ---> Using cache
 ---> 0eb49020f706
Step 6/7 : EXPOSE 5050
 ---> Using cache
 ---> 4dfe279482f4
Step 7/7 : CMD python /home/myapp/sample_app.py
 ---> Using cache
 ---> 90449d63de2c
```

```
Successfully built 90449d63de2c
Successfully tagged sampleapp:latest
99dcbf867e96843ff5bffdec30331659f9612898e7cc86d6953e21de88dcf8a3
CONTAINER ID          IMAGE                  COMMAND                      CREATED
STATUS                  PORTS                     NAMES
99dcbf867e96         sampleapp            "/bin/sh -c 'python …"    1 second ago
Up Less than a second   0.0.0.0:5050->5050/tcp   samplerunning
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$
```

Откройте вкладку браузера и перейдите на localhost:5050. Вы должны увидеть сообщение Ты заходишь на меня с 172.17.0.1.

Выключите сервер, когда убедитесь, что он работает на порту 5050. Вернитесь в окно терминала, в котором запущен сервер, и нажмите CTRL+C, чтобы остановить сервер.

Шаг 4: Запуште изменения на GitHub.

Теперь мы готовы перенести изменения в свой репозиторий GitHub. Введите следующие команды.

```
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ git add *
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   sample-app.sh
        modified:   sample_app.py
        new file:   tempdir/Dockerfile
        new file:   tempdir/sample_app.py
        new file:   tempdir/static/style.css
        new file:   tempdir/templates/index.html

devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ git commit -m
"Изменён порт с 8080 на 5050."
[master 98d9b2f] Изменён порт с 8080 на 5050.
 6 files changed, 33 insertions(+), 3 deletions(-)
 create mode 100644 tempdir/Dockerfile
 create mode 100644 tempdir/sample_app.py
 create mode 100644 tempdir/static/style.css
 create mode 100644 tempdir/templates/index.html
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ git push origin
master
```

Примечание: Если вы авторизовывались через расширение GitHub Visual Studio, см. выше, повторный запрос пароля ниже, будет отстутствовать. И файлы сразу запушаться на GitHub.

```
Username for 'https://github.com': username
Password for 'https://MisterZurg@github.com': password
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 2 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 748 bytes | 748.00 KiB/s, done.
Total 6 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/MisterZurg/sample-app.git
```

```
   a6b6b83..98d9b2f  master -> master
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$
```

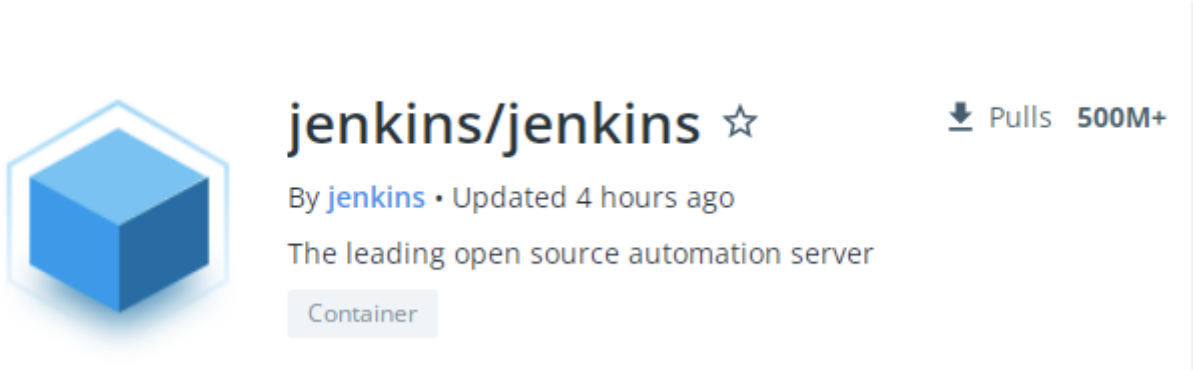
Вы можете проверить, что ваш репозиторий GitHub обновлен, посетив сайт https://github.com/github-user/sample-app. Вы должны увидеть новое сообщение (Изменён порт с 8080 на 5050.) и обновленную временную метку последнего коммита.

Часть 4: Загрузка и запуск Jenkins Docker Image

В этой части мы загрузим Jenkins Docker образ. Затем запустите его экземпляр и убедимся, что сервер Jenkins запущен.

Шаг 1: Загрузите Jenkins Docker образ.

Docker Образ Jenkins хранится здесь: https://hub.docker.com/r/jenkins/jenkins. На момент написания этого пункта лабораторной работы в Overview указано, что для загрузки последней версии контейнера Jenkins следует использовать команду **docker pull jenkins/jenkins**.



Вы должны получить результат, похожий на следующий:

```
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ docker pull jenkins/jenkins
Using default tag: latest
latest: Pulling from jenkins/jenkins
6aefca2dc61d: Already exists
53c03cc52d80: Pull complete
4d0896265ff9: Pull complete
2f62ad119e69: Pull complete
44704a7fc5b5: Pull complete
f4410ed22112: Pull complete
7c492eb9cb22: Pull complete
a3e3d06d7857: Pull complete
7f5bbec44f2d: Pull complete
be0b9dc5868f: Pull complete
99dcd0e51757: Pull complete
d5b8a7dcd362: Pull complete
a16b1d85dde6: Pull complete
9320f233b43f: Pull complete
04c4ee4c1673: Pull complete
0115957bf5a4: Pull complete
929b8088475f: Pull complete
Digest: sha256:f1058caddd535b238c80f49a2a7b0a9de71a82bb58d642472eba1f40258dc189
Status: Downloaded newer image for jenkins/jenkins:latest
docker.io/jenkins/jenkins:latest
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$
```

Шаг 2: Запустите Jenkins Docker-контейнер.

Введите следующую команду в одну строку, она запустит Jenkins Docker-контейнер, а затем позволит выполнять команды Docker внутри вашего сервера Jenkins.

```
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ docker run --rm -u
root -p 8080:8080 -v jenkins-data:/var/jenkins_home -v $(which
docker):/usr/bin/docker -v /var/run/docker.sock:/var/run/docker.sock -v
"$HOME":/home --name jenkins_server jenkins/jenkins
```

Ниже перечислены опции, используемые в этой команде docker run:

- **--rm** — автоматически удаляет контейнер Docker, когда вы прекращаете его запуск.
- **-u** — указывает пользователя; этот Docker-контейнер запускается от имени root, соответственно, все команды Docker, введенные внутри сервера Jenkins, будут разрешены.
- **-p** — указывает порт, на котором будет локально работать сервер Jenkins.
- **-v** — связывают монтируемые тома, необходимые для Jenkins и Docker. Первый -v указывает, где будут храниться данные Jenkins. Второй -v указывает, где взять Docker, чтобы можно было запустить Docker внутри контейнера Docker, в котором запущен сервер Jenkins. Третий -v указывает переменную PATH для домашнего каталога.

Шаг 3: Убедитесь, что сервер Jenkins запущен.

Теперь сервер Jenkins должен быть запущен. Скопируйте пароль администратора, который отображается в выводе, как показано ниже.

Не вводите никаких команд в этом окне сервера. Если вы случайно остановите сервер Jenkins, вам нужно будет снова ввести команду docker run из Шага 2 выше. После первоначальной установки пароль администратора отображается, как показано ниже.

```
<вывод опущен>
*************************************************************
*************************************************************
*************************************************************

Jenkins initial setup is required. An admin user has been created and a password
generated.
Please use the following password to proceed to installation:

05ac6e20f40b40749ea8323b83129c99 <-- Ваш пароль будет другим

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

*************************************************************
*************************************************************
*************************************************************
<вывод опущен>
2022-05-04 20:10:47.055+0000 [id=22]    INFO    hudson.WebAppMain$3#run: Jenkins
is fully up and running
```

Примечание: Если вы потеряли пароль, или он не отображается, как показано выше, или вам необходимо перезапустить сервер Jenkins, вы всегда можете получить пароль, обратившись к командной строке докер-контейнера Jenkins. Создайте второе окно терминала в VS Code и введите следующие команды, чтобы не останавливать сервер Jenkins:

```
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ docker exec -it
jenkins_server /bin/bash
root@f674aa47a54e:/# cat /var/jenkins_home/secrets/initialAdminPassword
05ac6e20f40b40749ea8323b83129c99
root@f674aa47a54e:/# exit
exit
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$
```

Шаг 4: Исследуем уровни абстракции, которые в настоящее время работают на вашем компьютере.

Следующая диаграмма иллюстрирует уровни абстракции в этой реализации Docker-inside-Docker (dind). Такой уровень сложности не является необычным в современных сетях и облачных инфраструктурах.

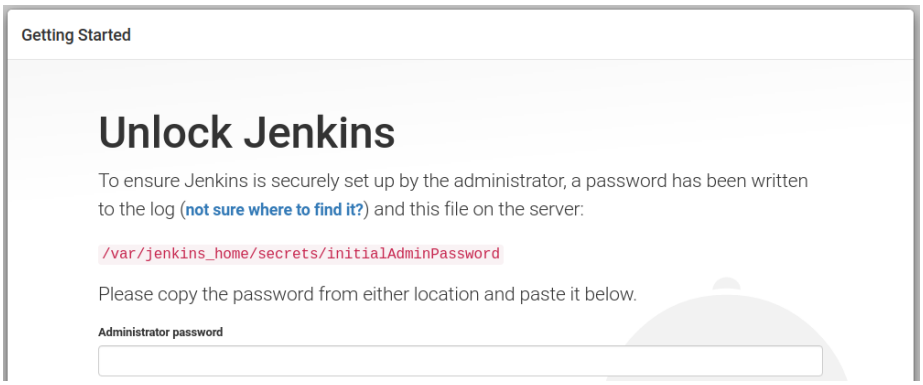
```
+----------------------------------------+
| Операционная система вашего компьютера |
|  +-----------------------------------+  |
|  |Вируальная Машина                  |  |
|  |  +----------------------------+   |  |
|  |  |Docker контейнер            |   |  |
|  |  |  +-----------------------+  |  |  |
|  |  |  | Jenkins сервер        |  |  |  |
|  |  |  |  +----------------+   |  |  |  |
|  |  |  |  |Docker контейнер|   |  |  |  |
|  |  |  |  +----------------+   |  |  |  |
|  |  |  +-----------------------+  |  |  |
|  |  +----------------------------+   |  |
|  +-----------------------------------+  |
+----------------------------------------+
```

Часть 5: Конфигурация Jenkins

В этой части мы завершим начальную настройку сервера Jenkins.

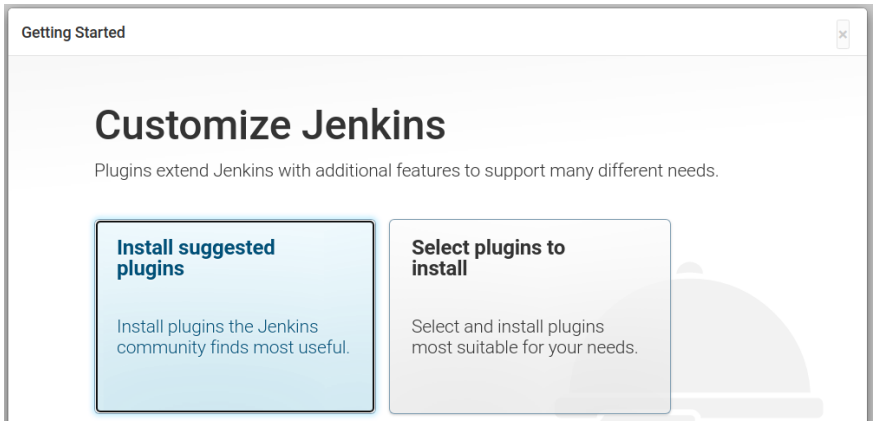
Шаг 1: Откройте вкладку веб-браузера.

Перейдите по адресу http://localhost:8080/ и войдите в систему, используя скопированный пароль администратора.



Шаг 2: Установите рекомендуемые плагины Jenkins.

Нажмите **Install suggested plugins** и подождите, пока Jenkins загрузит и установит плагины. В окне терминала вы увидите сообщения журнала по мере выполнения установки. Убедитесь, что вы не закрыли это окно терминала. Вы можете открыть другое окно терминала для доступа к командной строке.



Шаг 3: Пропустите создание нового администратора.

После завершения установки появится окно **Create First Admin User**. Пока что внизу нажмите **Skip and continue as admin**.

Шаг 4: Пропустите создание экземпляра конфигурации.

В окне **Instance Configuration** ничего не меняйте. Нажмите кнопку **Save and Finish**.

Шаг 5: Начните использовать Jenkins.

В следующем окне нажмите **Start using Jenkins**. Теперь вы должны оказаться на главном дашборде с сообщением **Welcome to** ~~the club Buddy~~ **Jenkins**!

Часть 6: Использование Jenkins для запуска сборки вашего приложения

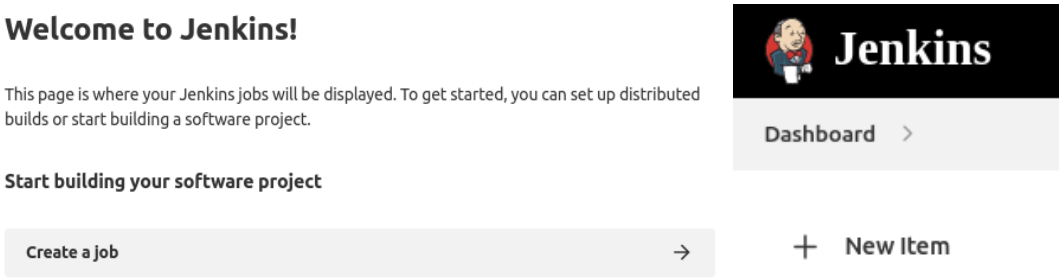
Основной единицей Jenkins является **job** *задание* (также известная как проект). Вы можете создавать задания, которые выполняют различные задачи, включая следующие:

- Получение кода из репозитория, например, GitHub.
- Сборка приложения с помощью скрипта или инструмента сборки.
- Упаковать приложение и запустить его на сервере.

В этой части мы создадим простое задание Jenkins, которое получит последнюю версию вашего sample-app с GitHub и запустит сценарий сборки. Затем в Jenkins вы сможете протестировать свое приложение и добавить его в пайплайн разработки.

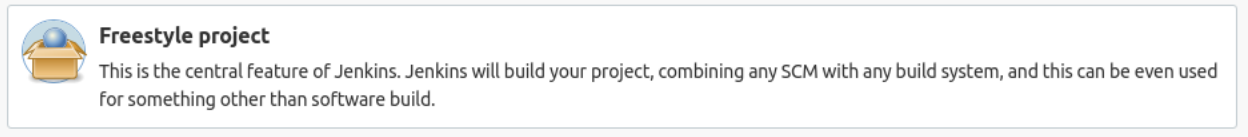
Шаг 1: Создайте новое задание.

Щелкните ссылку **Create a job** прямо под сообщением **Welcome to Jenkins!** Кроме того, вы можете щелкнуть New Item в меню слева.



В поле **Enter an item name** заполните имя **BuildAppJob**.

Выберите проект **Freestyle** в качестве типа задания. В описании аббревиатура SCM означает управление конфигурацией программного обеспечения — это классификация программного обеспечения, которая отвечает за отслеживание и контроль изменений в программном обеспечении.



Прокрутите до самого низа и нажмите OK.

Шаг 2: Настройте задание Jenkins BuildAppJob.

Теперь вы находитесь в окне конфигурации, где можно ввести подробную информацию о вашем задании. Вкладки в верхней части окна — это просто ярлыки для перехода к разделам ниже. Переходите по вкладкам, чтобы изучить параметры, которые можно настроить. Для этого простого задания вам нужно добавить только несколько деталей конфигурации.

Перейдите на вкладку General, добавьте описание для вашей работы. Например, "**Моя первая работа в Jenkins**".

Перейдите на вкладку **Source Code Management** и выберите кнопку **Git**. В поле Repository URL добавьте ссылку на репозиторий GitHub для примера приложения, указав имя пользователя с учетом регистра. Не забудьте добавить расширение .git в конце URL. Например:

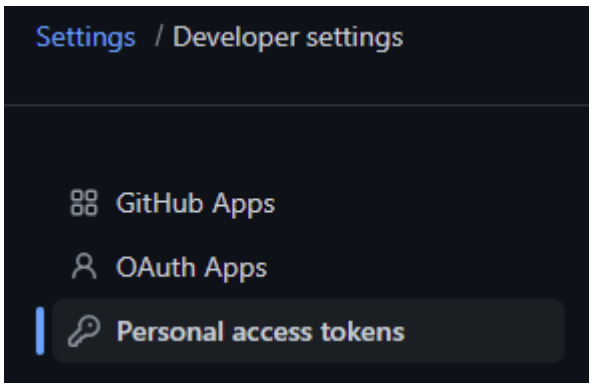
`https://github.com/`*`github-username`*`/sample-app.git`

Поскольку парольная аутентификация нынче не в моде *Support for password authentication was removed on August 13, 2021. Please use a personal access token instead.*

Нам нужно создать **personal access token** если таковой отсутствует.

Перейдите на https://github.com/settings/apps

В левой боковой панели нажмите **Personal access tokens**.

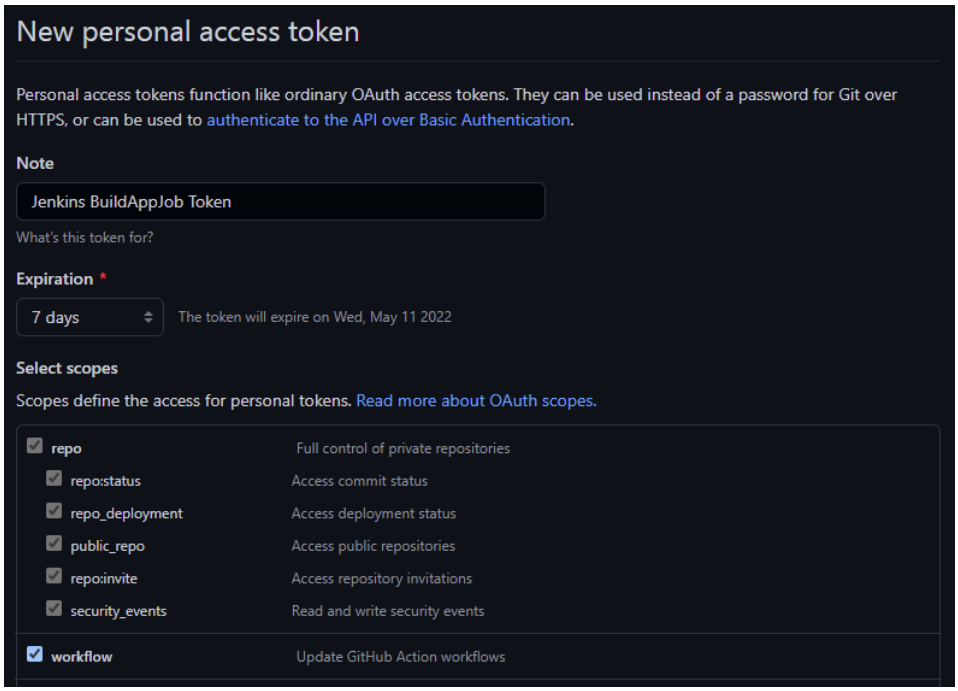


Нажмите **Generate new token**, после чего вам предложат ввести пароль.

Дайте своему Токену исчерпывающее имя, например. **Jenkins BuildAppJob Token**

Чтобы установить срок действия токена, выберите выпадающее меню Expiration, затем выберите значение по умолчанию или воспользуйтесь выбором календаря.

Выберите области действия, или разрешения, которые вы хотите предоставить этому токену. Чтобы использовать маркер для доступа к репозиториям из командной строки, выберите repo.



Обязательно скопируйте сгенерированный **Personal access tokens**. Поскольку больше не сможете его увидеть!

Воернитесь на вкладку Jenkins, в разделе **Credentials** нажмите кнопку **Add** и выберите **Jenkins**

В диалоговом окне **Add Credentials** введите имя пользователя и а на место пароля **Personal access tokens** который вы только что сгенерировали на GitHub, а затем нажмите кнопку **Add**.

Примечание: Вы получите сообщение об ошибке, что подключение не удалось. Это происходит потому, что вы еще не выбрали учетные данные.

В выпадающем списке **Credentials**, где сейчас написано **None**, выберите учетные данные, которые вы только что настроили.

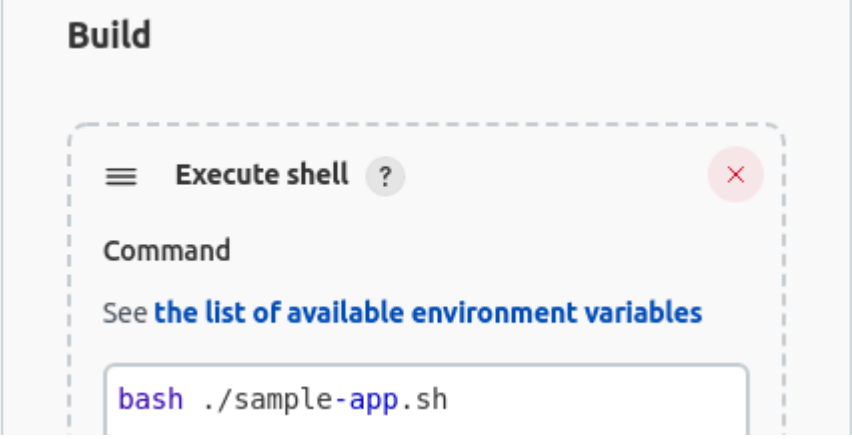
После того как вы добавили правильный URL и учетные данные, Jenkins проверяет доступ к репозиторию. У вас не должно быть сообщений об ошибках. В противном случае проверьте URL и учетные данные. Вам нужно будет добавить их снова, поскольку на данном этапе нет возможности удалить ранее введенные.

В верхней части окна конфигурации **BuildAppJob** перейдите на вкладку **Build**.

В раскрывающемся списке **Add build step** выберите **Execute shell.**

В поле Command введите команду, которую вы используете для запуска сценария **build for sample-app.sh**.

```
bash ./sample-app.sh
```



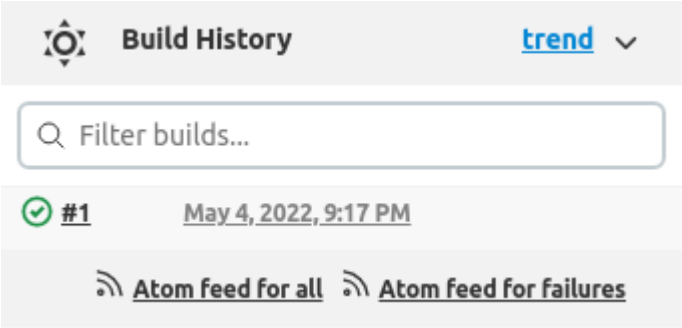
Нажмите кнопку **Save**. Вы вернетесь на dashboard Jenkins с выбранным заданием BuildAppJob.

Шаг 3: Поручите Jenkins создание приложения.

В левой части нажмите **Build Now**, чтобы запустить задание. Jenkins загрузит ваш Git-репозиторий и выполнит команду сборки bash ./sample-app.sh. Ваша сборка должна пройти успешно, потому что вы ничего не изменили в коде со времен части 3, когда вы модифицировали код.

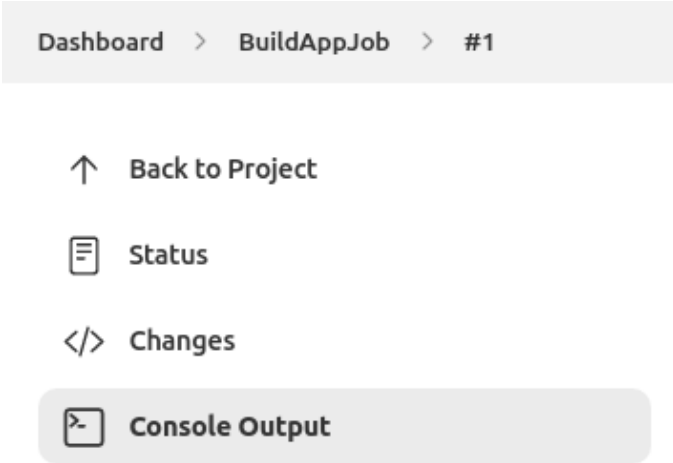
Шаг 4: Перейдите к деталям сборки.

Слева, в разделе Build History, нажмите на номер сборки, который должен быть №1, если вы не создавали приложение несколько раз.



Шаг 5: Просмотрите вывод консоли.

Слева нажмите **Console Output**. Вы должны увидеть вывод, похожий на следующий. Обратите внимание на сообщения об успехе внизу, а также на вывод команды **docker ps -a**. Запущены два контейнера docker: один для вашего примера приложения работает на локальном порту 5050, а другой для Jenkins на локальном порту 8080.

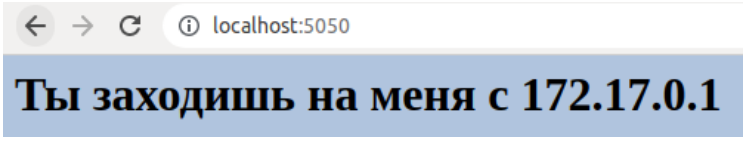



```
Started by user admin
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/BuildAppJob
The recommended git tool is: NONE
using credential d31d1631-758c-4dc9-959d-a5417c3ba0ca
Cloning the remote Git repository
Cloning repository https://github.com/MisterZurg/sample-app.git
 > git init /var/jenkins_home/workspace/BuildAppJob # timeout=10
Fetching upstream changes from https://github.com/MisterZurg/sample-app.git
 > git --version # timeout=10
 > git --version # 'git version 2.30.2'
using GIT_ASKPASS to set credentials
 > git fetch --tags --force --progress -- https://github.com/MisterZurg/sample-app.git +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git config remote.origin.url https://github.com/MisterZurg/sample-app.git # timeout=10
 > git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
Avoid second fetch
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
Checking out Revision 2c32bef4efadc265ffa4dd4aec2594be2e76e6fc (refs/remotes/origin/master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 2c32bef4efadc265ffa4dd4aec2594be2e76e6fc # timeout=10
```

```
Commit message: "Изменён порт с 8080 на 5050."
First time build. Skipping changelog.
[BuildAppJob] $ /bin/sh -xe /tmp/jenkins2049920068478646052.sh
+ bash ./sample-app.sh
mkdir: cannot create directory 'tempdir': File exists
mkdir: cannot create directory 'tempdir/templates': File exists
mkdir: cannot create directory 'tempdir/static': File exists
Sending build context to Docker daemon  6.656kB

Step 1/21 : FROM python
 ---> 2b7ca628da40
Step 2/21 : RUN pip install flask
 ---> Using cache
 ---> 4e39a3a408ea
Step 3/21 : COPY  ./static /home/myapp/static/
 ---> 85db350158f2
Step 4/21 : COPY  ./templates /home/myapp/templates/
 ---> 7a94185b6f05
Step 5/21 : COPY  sample_app.py /home/myapp/
 ---> 05dee7755ba7
Step 6/21 : EXPOSE 5050
 ---> Running in 8be5870fa2ac
Removing intermediate container 8be5870fa2ac
 ---> 444ec13b27e3
Step 7/21 : CMD python /home/myapp/sample_app.py
 ---> Running in af2a51be6b25
Successfully built af2a51be6b25
Successfully tagged sampleapp:latest
See 'docker run --help'.
CONTAINER ID          IMAGE                  COMMAND                       CREATED
STATUS                      PORTS                                NAMES
5c85551452e9         jenkins/jenkins      "/sbin/tini -- /usr/…"   About an hour
ago   Up About an hour          0.0.0.0:8080->8080/tcp, 50000/tcp   jenkins_server
99dcbf867e96          90449d63de2c          "/bin/sh -c 'python …"    2 hours ago
Exited (137) 2 hours ago                                           samplerunning
Finished: SUCCESS
```

Шаг 6: Откройте другую вкладку веб-браузера и убедитесь, что образец приложения запущен.

Введите локальный адрес, localhost:5050. Вы должны увидеть, что содержимое вашего index.html отображается в светло-стальном синем цвете фона, а "Ты заходишь на меня с 172.17.0.1" отображается как H1.



Часть 7: Использование Jenkins для тестирования сборки

В этой части вы создадите второе задание, которое протестирует сборку, чтобы убедиться, что она работает правильно.

Примечание: Вам необходимо остановить и удалить докер-контейнер samplerunning.

```
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ docker stop
samplerunning
samplerunning
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ docker rm
samplerunning
samplerunning
```

Шаг 1: Начните новое задание для тестирования вашего образца-приложения.

Вернитесь на вкладку в браузере с Jenkins и вернитнсь на главную страницу.

Щелкните ссылку **New Item**, чтобы создать новое задание.

В поле Enter an item name введите имя **TestAppJob**.

В качестве типа задания выберите **Freestyle project**.

Прокрутите страницу до самого низа и нажмите OK.

Шаг 2: Настройте задание Jenkins TestAppJob.

Добавьте описание вашего задания. Например, "Мой первый тест Jenkins".

Оставьте для параметра **Source Code Management** значение **None**.

Перейдите на вкладку **Build Triggers** и установите флажок **Build after other projects are built**. Для **Projects to watch** заполните имя **BuildAppJob**.

Шаг 3: Напишите тестовый скрипт, который должен запускаться после стабильной сборки BuildAppJob.

Перейдите на вкладку **Build.**

Нажмите **Add build step** и выберите **Execute shell**.

Введите следующий скрипт. Команда if должна быть вся на одной строке, включая ; then. Эта команда выполнит grep вывода, возвращаемого командой cURL, чтобы увидеть, возвращается ли сообщение You are calling me from 172.17.0.1. Если true, сценарий завершается с кодом 0, что означает отсутствие ошибок в сборке BuildAppJob. Если false, сценарий завершается с кодом 1, что означает, что BuildAppJob не удалось.

```
if curl http://172.17.0.1:5050/ | grep "Ты заходишь на меня с 172.17.0.1"; then
   exit 0
else
   exit 1
fi
```

Нажмите кнопку **Save**, а затем ссылку **Back to Dashboard**.

Шаг 4: Попросите Jenkins снова запустить задание BuildAppJob.

Обновите веб-страницу с помощью кнопки обновления браузера.

Теперь вы должны увидеть два задания, перечисленные в таблице. Для задания BuildAppJob нажмите кнопку сборки в крайнем правом углу (часы со стрелкой).

S	W	Name ↓	Last Success	Last Failure	Last Duration
✓		BuildAppJob	37 min #2	N/A	5 sec
...		TestAppJob	N/A	N/A	N/A

Шаг 5: Убедитесь, что оба задания выполнены.

Если все идет хорошо, вы должны увидеть, что временная метка в столбце **Last Success** обновилась как для **BuildAppJob**, так и для **TestAppJob**. Это означает, что ваш код для обоих заданий выполнился без ошибок. Но вы также можете проверить это самостоятельно.

S	W	Name ↓	Last Success	Last Failure	Last Duration
✓		BuildAppJob	12 sec #6	N/A	3.9 sec
✓		TestAppJob	2.5 sec #4	1 min 12 sec #3	1.1 sec

Примечание: Если временные метки не обновляются, убедитесь, что включена функция автоматического обновления, нажав на ссылку в правом верхнем углу.

Примечание: Если TestAppJob фейлится, возможно потому, что в index.html в заголвке располагается You are calling me from 172.17.0.1, вам потребуется либо изменить <h1> в index.html на <h1>Ты заходишь на меня с {{request.remote_addr}}</h1> либо в конфигурации TestAppJob заменить grep "You are calling me from с 172.17.0.1"

Щелкните ссылку для **TestAppJob**. В разделе **Permalinks** щелкните ссылку для последней сборки, а затем щелкните **Console Output**. Вы должны увидеть вывод, похожий на следующий:

```
Started by upstream project "BuildAppJob" build number 6
originally caused by:
 Started by user admin
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/TestAppJob
[TestAppJob] $ /bin/sh -xe /tmp/jenkins17682716252211920304.sh
+ curl http://172.17.0.1:5050/
+ grep Ты заходишь на меня с 172.17.0.1
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100   192  100   192    0     0  96000      0 --:--:-- --:--:-- --:--:-- 96000
    <h1>Ты заходишь на меня с 172.17.0.1</h1>
+ exit 0
Finished: SUCCESS
```

Нет необходимости проверять, запущен ли ваш Sample App, поскольку **TestAppJob** уже сделал это за вас. Однако вы можете открыть вкладку браузера для 172.17.0.1:5050, чтобы убедиться, что оно действительно запущено.

Часть 8: Создание Пайплайна в Jenkins

Хотя в настоящее время вы можете запустить два задания, просто нажав кнопку Build Now для BuildAppJob, проекты по разработке программного обеспечения обычно намного сложнее. Эти проекты могут значительно выиграть от автоматизации сборок для непрерывной интеграции изменений кода и постоянного создания сборок разработки, готовых к развертыванию. В этом и заключается суть CI/CD. Пайплайе может быть автоматизирован для запуска на основе различных триггеров, в том числе периодически, на основе “прослушивания” GitHub на предмет изменений или из удаленно запущенного скрипта. Однако в этой части мы напишем скрипт пайплайна в Jenkins, который будет запускать ваши два приложения каждый раз, когда вы нажмете кнопку Build Now.

Шаг 1: Создайте задание Pipeline.

Нажмите **Jenkins** в левом верхнем углу, а затем **New Item**.

В поле **Enter an item name** введите **SamplePipeline**.

Выберите **Pipeline** в качестве типа задания.

Прокрутите страницу до самого низа и нажмите **OK**.

Шаг 2: Настройте задание SamplePipeline.

Вдоль верхней части перейдите по вкладкам и изучите каждый раздел страницы конфигурации. Обратите внимание, что существует несколько различных способов инициировать сборку. Для задания **SamplePipeline** вы будете запускать его вручную.

В разделе **Pipeline** добавьте следующий сценарий.

```
node {
   stage('Preparation') {
       catchError(buildResult: 'SUCCESS') {
          sh 'docker stop samplerunning'
          sh 'docker rm samplerunning'
       }
   }
   stage('Build') {
```

```
            build 'BuildAppJob'
        }
        stage('Results') {
            build 'TestAppJob'
        }
    }
```

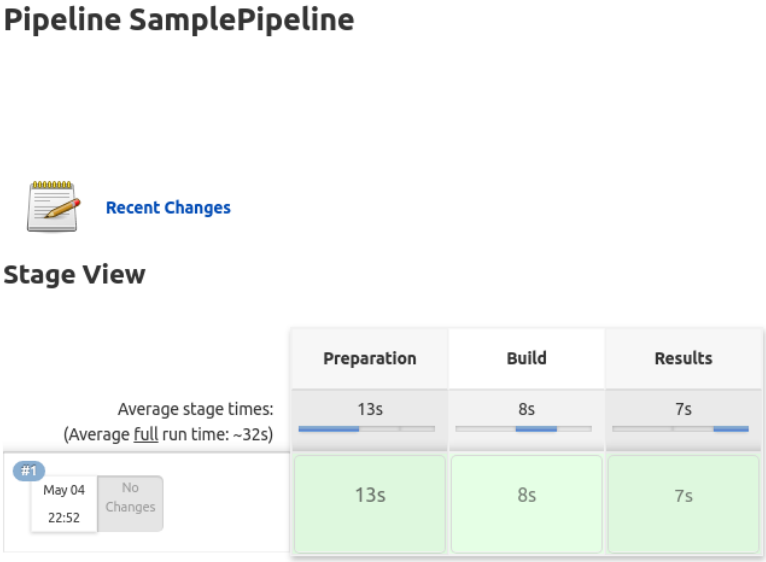
Этот скрипт выполняет следующие действия:

- Он создает сборку состоящую из одной ноды в отличие от распределенной или мульти-нодовой. Распределенные или мульти-нодовой конфигурации предназначены для более крупных пайплайнов.
- На этапе **Preparation SamplePipeline** сначала убедится, что все предыдущие экземпляры Docker -контейнера BuildAppJob остановлены и удалены. Но если еще нет запущенного контейнера, вы получите ошибку. Поэтому мы используем функцию catchError, чтобы отловить любые ошибки и вернуть значение "SUCCESS". Это обеспечит продолжение работы пайплайна на следующем этапе.
- На этапе **Build**, **SamplePipeline** создаст ваше задание **BuildAppJob**.
- На этапе результатов **SamplePipeline** создаст ваше задание **TestAppJob**.

Нажмите кнопку **Save**, и вы вернетесь на дашбоард Jenkins для задания **SamplePipeline**.

Шаг 3: Запустите SamplePipeline.

Слева нажмите **Build Now**, чтобы запустить задание **SamplePipeline**. Если вы написали скрипт **Pipeline** без ошибок, то в представлении **Stage View** должны появиться три зеленых поля с количеством секунд, затраченных на сборку каждого этапа. Если это не так, нажмите **Configure** слева, чтобы вернуться к конфигурации **SamplePipeline** и проверить свой сценарий **Pipeline**.



Шаг 4: Проверьте вывод SamplePipeline.

Щелкните на ссылке последней сборки в разделе Permalinks, а затем щелкните Console Output. Вы должны увидеть вывод, похожий на следующий:

```
Started by user admin
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/SamplePipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Preparation)
[Pipeline] catchError
[Pipeline] {
[Pipeline] sh
+ docker stop samplerunning
samplerunning
[Pipeline] sh
+ docker rm samplerunning
samplerunning
[Pipeline] }
[Pipeline] // catchError
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Build)
[Pipeline] build (Building BuildAppJob)
Scheduling project: BuildAppJob
Starting building: BuildAppJob #7
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Results)
[Pipeline] build (Building TestAppJob)
Scheduling project: TestAppJob
Starting building: TestAppJob #5
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Лабораторная работа №6.5.10

Исследование эволюции парольных систем

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC

Часть 2: Python: пароли в обычном тексте

Часть 3: Python: парольный хэша

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

Порядок выполнения работы

Часть 1: Запуск виртуальной машины DEVASC

Если вы еще не завершили **лабораторную работу - Установка лабораторной среды виртуальной машины**, сделайте это сейчас. Если вы

уже завершили эту лабораторную работу, запустите виртуальную машину **DEVASC**.

Часть 2: Демонстрация приложения

Ваш преподаватель может продемонстрировать приложение "Простой текст пароля и хэширование". Однако мы постепенно, шаг за шагом, создадим его, что называется from scratch.

Сначала приложение сохраняет имя пользователя и пароль в виде обычного текста в базе данных веб-службы. Затем оно проверяет, что учетные данные были сохранены и работают правильно. Второй метод, хэширование пароля, также сохраняет их и проверяет в базе данных веб-службы.

Часть 3: Установка пакетов и создание веб-службы

Шаг 1: Откройте каталог безопасности в VS Code и установите пакеты Python.

Откройте VS Code. Затем нажмите **File > Open Folder...** и перейдите в каталог devnet-src/security. Нажмите OK.

На панели EXPLORER откройте **password-evolution.py**, чтобы открыть его.

Откройте терминал в VS Code. Щелкните **Terminal > New Terminal**.

Используйте следующие команды для установки пакетов, необходимых в этой лабораторной работе. Эти пакеты могут быть уже установлены на вашей виртуальной машине. В этом случае вы получите сообщение **Requirement already satisfied**.

```
devasc@labvm:~/labs/devnet-src/security$ pip3 install pyotp
Requirement already satisfied: pyotp in
/home/devasc/.local/lib/python3.8/site-packages (2.3.0)
devasc@labvm:~/labs/devnet-src/security$ pip3 install flask
Requirement already satisfied: flask in
/home/devasc/.local/lib/python3.8/site-packages (1.1.2)
Requirement already satisfied: itsdangerous>=0.24 in
/home/devasc/.local/lib/python3.8/site-packages (from flask) (1.1.0)
Requirement already satisfied: click>=5.1 in
/home/devasc/.local/lib/python3.8/site-packages (from flask) (7.1.2)
Requirement already satisfied: Werkzeug>=0.15 in
/home/devasc/.local/lib/python3.8/site-packages (from flask) (1.0.1)
Requirement already satisfied: Jinja2>=2.10.1 in
/home/devasc/.local/lib/python3.8/site-packages (from flask) (2.11.2)
Requirement already satisfied: MarkupSafe>=0.23 in
/home/devasc/.local/lib/python3.8/site-packages (from Jinja2>=2.10.1-
>flask) (1.1.1)
devasc@labvm:~/labs/devnet-src/security$
```

Шаг 2: Импорт библиотек.

В файл password-evolution.py добавьте следующий код. Обратите внимание на команду, db_name = 'test.db'. Это база данных SQL (sqlite3), в которой хранятся имена пользователей и пароли, которые вы будете создавать.

```
import pyotp    # генерирует одноразовые пароли
import sqlite3  # база данных для имени пользователя/паролей
import hashlib  # безопасные хэши и дайджесты сообщений
import uuid     # создание уникальных идентификаторов
from flask import Flask, request
app = Flask(__name__) # Обязательно используйте два знака подчеркивания
до и после "name"

db_name = "test.db"
```

Шаг 3: Создайте веб-сервис.

Далее добавьте в файл следующий код Flask для отображения фразы веб-контента по корневому пути. Когда пользователь перейдет по URL (корневой каталог), в браузере будет отображен вывод оператора возврата.

```
@app.route("/")
def index():
    return "Эволюция парольных систем!"
```

Добавьте в файл следующий код для создания веб-сервиса на порту 5000 с самоподписанным сертификатом TLS. Параметр **ssl_context="adhoc"** позволяет запускать приложение через HTTPS без необходимости использования сертификатов или при использовании самоподписанного сертификата. Обязательно используйте два знака подчеркивания до и после **name** и **main**.

```
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000, ssl_context="adhoc")
```

Сохраните и запустите файл **password-evolution.py**. Команда **nohup** (no hangup) поддерживает процесс даже после выхода из оболочки или терминала. & заставляет команду работать в фоновом режиме.

```
devasc@labvm:~/labs/devnet-src/security$ nohup python3 password-
evolution.py &
[1] 3417
devasc@labvm:~/labs/devnet-src/security$ nohup: ignoring input and
appending output to 'nohup.out'

devasc@labvm:~/labs/devnet-src/security$
```

Нажмите **Enter**, чтобы получить новую командную строку.

Теперь ваш сервер Flask запущен. В VS Code в дириктории **/security** вы должны увидеть текстовый файл **nohup.out**, созданный Flask. Щелкните файл, чтобы прочитать его вывод.

Убедитесь, что веб-сервис запущен. Обязательно используйте HTTPS, а не HTTP. Параметр -k позволяет curl выполнять "небезопасные" SSL-соединения

и передачи. Без опции -k вы получите сообщение об ошибке "SSL certificate problem: self-signed certificate". Команда выведет сообщение из команды return, которую вы закодировали в своем сценарии.

```
devasc@labvm:~/labs/devnet-src/security$ curl -k https://0.0.0.0:5000/
Эволюция парольных систем!devasc@labvm:~/labs/devnet-src/security$
```

```
devasc@labvm:~/labs/devnet-src/security$ curl -k https://0.0.0.0:5000/
Эволюция парольных систем!devasc@labvm:~/labs/devnet-src/security$
```

Нажмите Enter, чтобы получить командную строку с новой строки.

Прежде чем продолжить, завершите работу программы. Используйте следующую команду для её остановки:

```
devasc@labvm:~/labs/devnet-src/security$ pkill -f password-evolution.py
devasc@labvm:~/labs/devnet-src/security$
```

Часть 4: Исследование Python программы, хранящей пароли в обычном тексте

При первом использовании паролей, они просто хранились в базе данных в виде обычного текста. Когда пользователь вводил свои учетные данные, система просматривала пароль на предмет его совпадения. Система была очень проста в реализации, но в то же время очень небезопасна. В этой части мы изменим **password-evolution.py**, чтобы позволить ему хранить идентификационные данные пользователя в базе данных test.db. Затем мы создадите пользователя и выполните аутентификацию по этим учетным данным. Наконец, мы познакомимся с базой данных test.db, чтобы убедиться, что они хранились в открытом виде.

Шаг 1: Удалите конфигурацию сервера.

Удалите следующие строки из файла **password-evolution.py**.

```
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000, ssl_context="adhoc")
```

Шаг 2: Настройте сервер для хранения учетных данных.

Добавьте следующий код Flask, чтобы настроить сервер на хранение имени пользователя и пароля пользователя в открытом виде. Используя метод HTTP POST, этот код позволяет пользователю создать (**"signup"**) новое имя пользователя и пароль, которые будут храниться в файле базы данных **test.db**. Позже, когда пользователь введет имя пользователя и пароль, этот код вернет сообщение **"signup success"**.

```
######################################## Открытый Текст
#########################################################
@app.route("/signup/v1", methods=["POST"])
def signup_v1():
    conn = sqlite3.connect(db_name)
    c = conn.cursor()
    c.execute('''CREATE TABLE IF NOT EXISTS USER_PLAIN
            (USERNAME  TEXT    PRIMARY KEY NOT NULL,
             PASSWORD  TEXT    NOT NULL);''')
    conn.commit()
    try:
```

```
        c.execute("INSERT INTO USER_PLAIN (USERNAME,PASSWORD) "
                  "VALUES ('{0}',
'{1}')".format(request.form['username'], request.form['password']))
        conn.commit()
    except sqlite3.IntegrityError:
        return "username has been registered."
    print('username: ', request.form['username'], ' password: ',
request.form['password'])
    return "signup success"
```

Примечание:Убедитесь в правильности отступов, иначе код может работать некорректно.

Добавьте следующий код Flask в файл **password-evolution.py** для проверки новых учетных данных.

```
def verify_plain(username, password):
    conn = sqlite3.connect("test.db")
    c = conn.cursor()
    query = "SELECT PASSWORD FROM USER_PLAIN WHERE USERNAME =
'{0}'".format(username)
    c.execute(query)
    records = c.fetchone()
    conn.close()
    if not records:
        return False
    return records[0] == password
```

Добавьте следующий код Flask в файл **password-evolution.py**. Этот код используется при каждой попытке входа в систему для чтения параметров из HTTP-запроса и проверки учетной записи. В случае успешного входа будет возвращено сообщение **"login success"**, в противном случае пользователь увидит сообщение **"Invalid username/password"**.

```
@app.route("/login/v1", methods=["GET", "POST"])
def login_v1():
    error = None
    if request.method == "POST":
        if verify_plain(request.form["username"],
request.form["password"]):
            error = "login success"
        else:
            error = "Invalid username/password"
    else:
        error = "Invalid Method"
    return error
```

Шаг 3: Запустите сервер и протестируйте его.

Добавьте обратно код конфигурации сервера, который вы удалили ранее.

```
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000, ssl_context="adhoc")
```

Сохраните программу как **password-evolution-plain-text.py** и запустите для запуска обновленной веб-службы.

```
devasc@labvm:~/labs/devnet-src/security$ nohup python3 password-evolution-plain-text.py &
[1] 3765
devasc@labvm:~/labs/devnet-src/security$ nohup: ignoring input and appending output to 'nohup.out'
```

Используйте следующие команды curl для создания (регистрации) двух учетных записей пользователей, **alice** и **bob**, и отправки POST в веб-сервис. Каждая команда включает имя пользователя, пароль и вызываемую функцию регистрации, которая хранит эту информацию, включая пароль, в виде открытого текста. Вы должны увидеть сообщение "signup success" из команды return, которую вы включили в предыдущий шаг.

Примечание: После каждой команды нажмите **Enter**.

```
devasc@labvm:~/labs/devnet-src/security$ curl -k -X POST -F 'username=alice' -F 'password=myalicepassword'  'https://0.0.0.0:5000/signup/v1'
signup successdevasc@labvm:~/labs/devnet-src/security$

devasc@labvm:~/labs/devnet-src/security$ curl -k -X POST -F 'username=bob' -F 'password=passwordforbob'  'https://0.0.0.0:5000/signup/v1'
signup successdevasc@labvm:~/labs/devnet-src/security$
devasc@labvm:~/labs/devnet-src/security$
```

```
devasc@labvm:~/labs/devnet-src/security$ curl -k -X POST -F 'username=alice
' -F 'password=myalicepassword'  'https://0.0.0.0:5000/signup/v1'
signup successdevasc@labvm:~/labs/devnet-src/security$ curl -k -X POST -F '
username=bob' -F 'password=passwordforbob'  'https://0.0.0.0:5000/signup/v1
'
signup successdevasc@labvm:~/labs/devnet-src/security$ curl -k -X POST -F '
```

Шаг 4: Убедитесь, что новые пользователи могут войти в систему.

Используйте следующие команды curl, чтобы проверить, что оба пользователя могут войти в систему со своими паролями, которые хранятся в открытом виде.

```
devasc@labvm:~/labs/devnet-src/security$ curl -k -X POST -F 'username=alice' -F 'password=myalicepassword' 'https://0.0.0.0:5000/login/v1'
login successdevasc@labvm:~/labs/devnet-src/security$
devasc@labvm:~/labs/devnet-src/security$

devasc@labvm:~/labs/devnet-src/security$ curl -k -X POST -F 'username=bob' -F 'password=passwordforbob' 'https://0.0.0.0:5000/login/v1'
login successdevasc@labvm:~/labs/devnet-src/security$
```

```
signup successdevasc@labvm:~/labs/devnet-src/security$ curl -k -X POST -F '
username=alice' -F 'password=myalicepassword' 'https://0.0.0.0:5000/login/v
1'
login successdevasc@labvm:~/labs/devnet-src/security$ curl -k -X POST -F 'u
sername=bob' -F 'password=passwordforbob' 'https://0.0.0.0:5000/login/v1'
login successdevasc@labvm:~/labs/devnet-src/security$
```

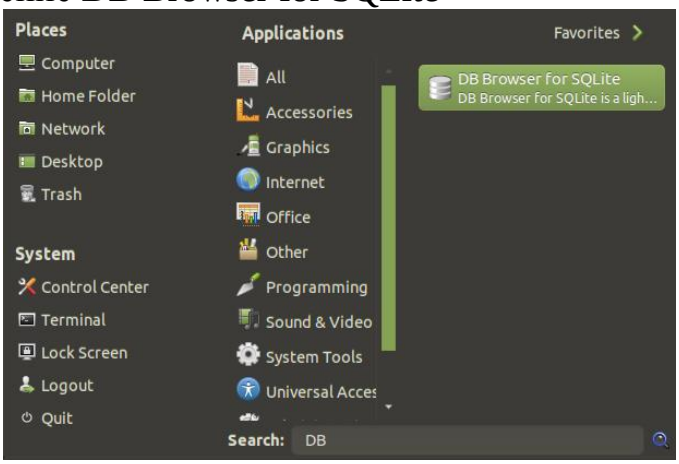
Завершите работу сервера.

```
login successdevasc@labvm:~/labs/devnet-src/security$ pkill -f
password-evolution-plain-text.py
[1]+  Terminated              nohup python3 password-evolution.py
devasc@labvm:~/labs/devnet-src/security$
```

Шаг 5: Проверьте содержимое файла test.db.

Возможно, вы заметили, что SQLite создал файл **test.db** в дириктории **/security**. Вы можете открыть этот файл и посмотреть имя пользователя и пароли для **alice** и **bob**. Однако в этом шаге вы будете использовать приложение для просмотра файлов базы данных SQLite.

Откройте приложение DB Browser for SQLite

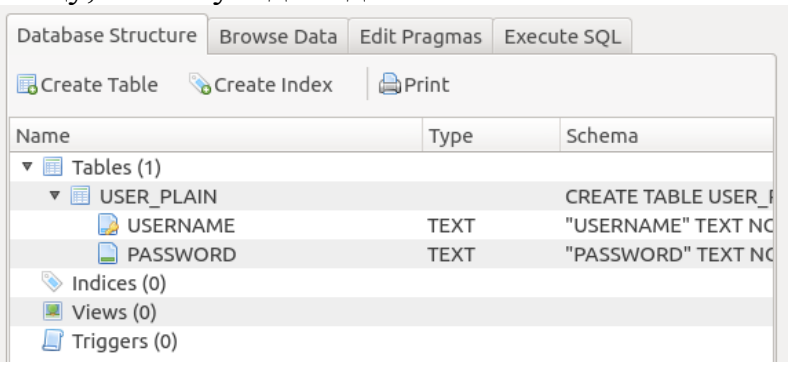


После запуска DB Browser for SQLite откройте файл test.db:

- Выберите: **File > Open database…**
- Перейдите в каталог labs/devnet-src/security и выберите test.db.
- Нажмите кнопку **Open**.

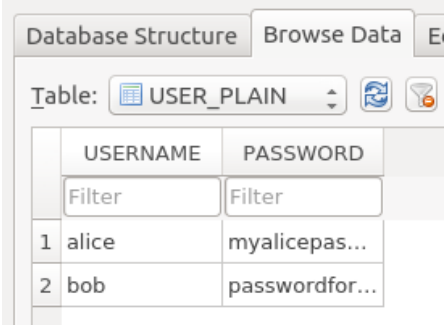
На вкладке **Database Structure** обратите внимание на таблицу USER_PLAIN, которая совпадает с кодом, созданным ранее.

Разверните таблицу, чтобы увидеть два поля: USERNAME и PASSWORD.



Выберите вкладку **Browse Data**.

Таблица: USER_PLAIN уже выбрана. Здесь вы можете увидеть имена пользователей **bob** и **alice** вместе с их паролями в открытом виде.



Закройте приложение DB Browser for SQLite

Часть 5: Хеширование паролей в Python

Вместо того чтобы хранить пароли в открытом виде, вы можете хэшировать их в момент создания. Когда пароль хэшируется, он преобразуется в нечитаемый набор символов. Это позволяет невозможность обратного преобразования. Даже если база данных будет украдена, ее нельзя будет использовать, поскольку хэш неизвестен. Сейчас мы изменим файл **password-evolution-plain-text.py**, чтобы создать веб-API, который может принимать веб-запросы и сохранять пароль нового пользователя в хэшированном формате.

Шаг 1: Удалите конфигурацию сервера.

Ещё раз удалите следующие две строки из файла **password-evolution-plain-text.py**. Эти строки будут добавлены позже.

```
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000, ssl_context="adhoc")
```

Шаг 2: Настройте сервер для хранения учетных данных.

Добавьте следующий код, чтобы сервер мог хэшировать пароль, используя алгоритм SHA256. Этот код позволяет пользователю создать ("signup") новое имя пользователя и пароль, которые будут храниться в файле базы данных test.db SQL. Разница в том, что пароли будут храниться в виде хэш-значений, а не в открытом виде. Эта процедура использует sha256, но не солит хэш. Последствия использования хэша без соли вы увидите при просмотре файла базы данных test.db.

```
######################################### Хеширование Пароля
#######################################################
@app.route("/signup/v2", methods=["GET", "POST"])
def signup_v2():
    conn = sqlite3.connect(db_name)
    c = conn.cursor()
    c.execute('''CREATE TABLE IF NOT EXISTS USER_HASH
            (USERNAME  TEXT    PRIMARY KEY NOT NULL,
             HASH      TEXT    NOT NULL);''')
    conn.commit()
    try:
        hash_value =
hashlib.sha256(request.form["password"].encode()).hexdigest()
```

```
        c.execute("INSERT INTO USER_HASH (USERNAME, HASH) "
                  "VALUES ('{0}',
'{1}')".format(request.form["username"], hash_value))
        conn.commit()
    except sqlite3.IntegrityError:
        return "username has been registered."
    print("username: ", request.form["username"], " password: ",
request.form["password"], " hash: ", hash_value)
    return "signup success"
```

Добавьте (скопируйте) следующий код в файл **password-evolution-plain-text.py** для проверки того, что пароль был сохранен только в хэш-формате. Код определяет функцию verify_hash, которая сравнивает имя пользователя и пароль в хэш-формате. Если сравнение истинно, то пароль был сохранен только в хэш-формате.

```
def verify_hash(username, password):
    conn = sqlite3.connect(db_name)
    c = conn.cursor()
    query = "SELECT HASH FROM USER_HASH WHERE USERNAME =
'{0}'".format(username)
    c.execute(query)
    records = c.fetchone()
    conn.close()
    if not records:
        return False
    return records[0] == hashlib.sha256(password.encode()).hexdigest()
```

Добавьте следующий код в файл **password-evolution-plain-text.py**. Следующий код считывает параметры из HTTP POST запроса и проверяет, что пользователь указал правильный пароль при входе в систему.

```
@app.route("/login/v2", methods=["GET", "POST"])
def login_v2():
    error = None
    if request.method == 'POST':
        if verify_hash(request.form['username'],
request.form['password']):
            error = "login success"
        else:
            error = "Invalid username/password"
    else:
        error = "Invalid Method"
    return error
```

Шаг 3: Запустите сервер и протестируйте его.

Добавьте обратно код конфигурации сервера, который вы удалили ранее.

```
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000, ssl_context="adhoc")
```

Сохраните файл как **password-evolution-hashing.py**, а затем запустите сценарий для запуска обновленной веб-службы.

```
devasc@labvm:~/labs/devnet-src/security$ nohup python3 password-evolution-hashing.py &
[1] 4282
devasc@labvm:~/labs/devnet-src/security$ nohup: ignoring input and appending output to 'nohup.out'
```

Используйте следующие команды curl для создания трех новых учетных записей пользователей с хэшированным паролем. Обратите внимание, что два пользователя, **rick** и **allan**, используют один и тот же пароль.

```
devasc@labvm:~/labs/devnet-src/security$ curl -k -X POST -F 'username=rick' -F 'password=samepassword' 'https://0.0.0.0:5000/signup/v2'
signup successdevasc@labvm:~/labs/devnet-src/security$

devasc@labvm:~/labs/devnet-src/security$ curl -k -X POST -F 'username=allan' -F 'password=samepassword' 'https://0.0.0.0:5000/signup/v2'
signup successdevasc@labvm:~/labs/devnet-src/security$

devasc@labvm:~/labs/devnet-src/security$ curl -k -X POST -F 'username=dave' -F 'password=differentpassword' 'https://0.0.0.0:5000/signup/v2'
signup successdevasc@labvm:~/labs/devnet-src/security$
```

Используйте команды curl для проверки входа всех трех пользователей с их паролями, сохраненными в хэш-памяти. Пользователь allan введен дважды, первый раз с неправильным паролем. Обратите внимание на сообщение "Invalid username/password", которое совпадает с кодом для этой функции, который вы добавили в предыдущем шаге.

```
devasc@labvm:~/labs/devnet-src/security$ curl -k -X POST -F 'username=rick' -F 'password=samepassword' 'https://0.0.0.0:5000/login/v2'
login successdevasc@labvm:~/labs/devnet-src/security$

devasc@labvm:~/labs/devnet-src/security$ curl -k -X POST -F 'username=allan' -F 'password=wrongpassword' 'https://0.0.0.0:5000/login/v2'
Invalid username/passwordevasc@labvm:~/labs/devnet-src/security$

devasc@labvm:~/labs/devnet-src/security$ curl -k -X POST -F 'username=allan' -F 'password=samepassword' 'https://0.0.0.0:5000/login/v2'
login successdevasc@labvm:~/labs/devnet-src/security$

devasc@labvm:~/labs/devnet-src/security$ curl -k -X POST -F 'username=dave' -F 'password=differentpassword' 'https://0.0.0.0:5000/login/v2'
login successdevasc@labvm:~/labs/devnet-src/security$
```

```
Invalid username/passworddevasc@labvm:~/labs/devnet-src/secur
ity$ curl -k -X POST -F 'username=allan' -F 'password=samepas
sword' 'https://0.0.0.0:5000/signup/v2'
signup successdevasc@labvm:~/labs/devnet-src/security$ curl -
k -X POST -F 'username=allan' -F 'password=samepassword' 'htt
ps:/login/v2'000/signup/v2'
login successdevasc@labvm:~/labs/devnet-src/security$ curl -k
 -X POST -F 'username=dave' -F 'password=differentpassword' '
https://0.0.0.0:5000/login/v2'
login successdevasc@labvm:~/labs/devnet-src/security$
```

Это подтверждает, что хэшированный пароль надежно хранится, а пароли пользователей защищены в случае их компрометации.

Завершить работу сервера

```
devasc@labvm:~/labs/devnet-src/security$ pkill -f password-evolution-
hashing.py
[1]+  Terminated              nohup python3 password-evolution.py
devasc@labvm:~/labs/devnet-src/security$
```

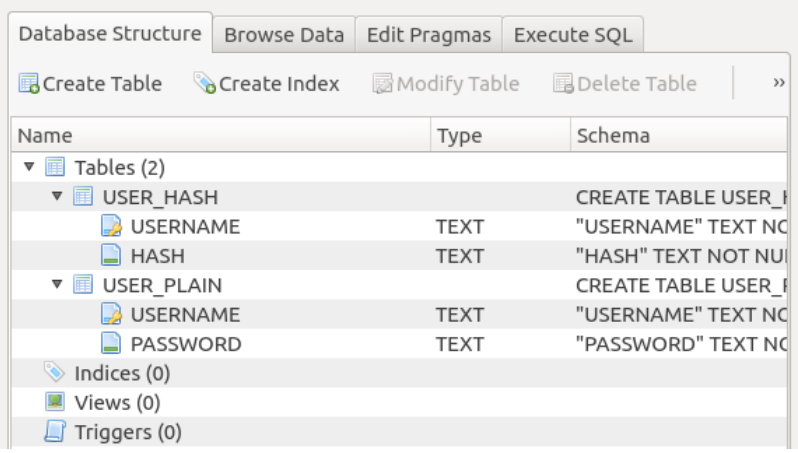
Шаг 4: Проверьте содержимое файла test.db.

Откройте приложение **DB Browser for SQLite**.

Откройте файл **test.db**.

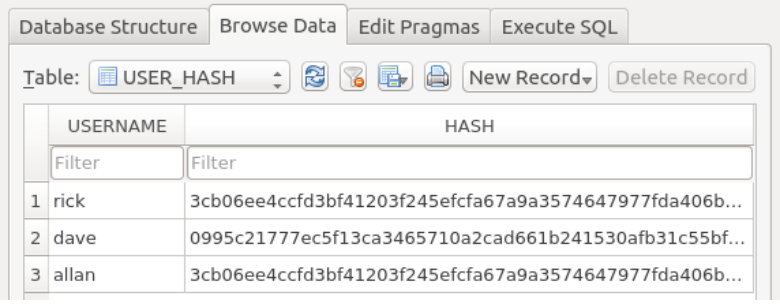
Выберите вкладку **Database Structure**.

Вы заметите две структуры, которые совпадают с кодом, который вы включили ранее: USER_PLAIN и USER HASH.



Выберите вкладку **Browse Data**.

Таблица: USER_HASH уже должна быть выбрана. Теперь вы увидите имена пользователей **rick**, **allan** и **dave** вместе с их хэшированными паролями.



Обратите внимание, что у **rick** и **allan** одинаковые хэшированные пароли. Это потому, что у них был один и тот же пароль, а хэш-функция не включала соль, чтобы сделать их хэш уникальным. “**Соление хэша**” — это процесс добавления случайных данных к хэшу. Чтобы гарантировать уникальность паролей, повысить их сложность и предотвратить атаки на пароли даже при одинаковых входных данных, на вход хэш-функции необходимо добавить соль.

Часть 6: Обзор окончательной программы

Ниже приведен полный сценарий, который вы создали в этой лабораторной работе.

```
import pyotp    # генерирует одноразовые пароли
import sqlite3  # база данных для имени пользователя/паролей
import hashlib  # безопасные хэши и дайджесты сообщений
import uuid     # создание уникальных идентификаторов
from flask import Flask, request
app = Flask(__name__) # Обязательно используйте два знака подчеркивания до и после "name"

db_name = "test.db"


@app.route("/")
def index():
    return "Эволюция парольных систем!"


########################################                Открытый                Текст
#########################################################
@app.route("/signup/v1", methods=["POST"])
def signup_v1():
    conn = sqlite3.connect(db_name)
    c = conn.cursor()
    c.execute('''CREATE TABLE IF NOT EXISTS USER_PLAIN
           (USERNAME  TEXT    PRIMARY KEY NOT NULL,
            PASSWORD  TEXT    NOT NULL);''')
    conn.commit()
    try:
        c.execute("INSERT INTO USER_PLAIN (USERNAME,PASSWORD) "
                  "VALUES       ('{0}',       '{1}')".format(request.form["username"],
request.form["password"]))
        conn.commit()
    except sqlite3.IntegrityError:
        return "username has been registered."
    print("username:     ",      request.form["username"],      "      password:      ",
request.form["password"])
    return "signup success"


def verify_plain(username, password):
    conn = sqlite3.connect("test.db")
    c = conn.cursor()
    query = "SELECT PASSWORD FROM USER_PLAIN WHERE USERNAME = '{0}'".format(username)
    c.execute(query)
    records = c.fetchone()
    conn.close()
    if not records:
        return False
    return records[0] == password


@app.route("/login/v1", methods=["GET", "POST"])
def login_v1():
    error = None
    if request.method == "POST":
        if verify_plain(request.form["username"], request.form["password"]):
```

```
            error = "login success"
        else:
            error = "Invalid username/password"
    else:
        error = "Invalid Method"
    return error



########################################              Хеширование              Пароля
#########################################################
@app.route("/signup/v2", methods=["GET", "POST"])
def signup_v2():
    conn = sqlite3.connect(db_name)
    c = conn.cursor()
    c.execute('''CREATE TABLE IF NOT EXISTS USER_HASH
           (USERNAME  TEXT    PRIMARY KEY NOT NULL,
            HASH      TEXT    NOT NULL);''')
    conn.commit()
    try:
        hash_value = hashlib.sha256(request.form["password"].encode()).hexdigest()
        c.execute("INSERT INTO USER_HASH (USERNAME, HASH) "
                  "VALUES ('{0}', '{1}')".format(request.form["username"], hash_value))
        conn.commit()
    except sqlite3.IntegrityError:
        return "username has been registered."
    print("username:      ",      request.form["username"],      "      password:      ",
request.form["password"], " hash: ", hash_value)
    return "signup success"


@app.route("/login/v2", methods=["GET", "POST"])
def login_v2():
    error = None
    if request.method == 'POST':
        if verify_hash(request.form['username'], request.form['password']):
            error = "login success"
        else:
            error = "Invalid username/password"
    else:
        error = "Invalid Method"
    return error


def verify_hash(username, password):
    conn = sqlite3.connect(db_name)
    c = conn.cursor()
    query = "SELECT HASH FROM USER_HASH WHERE USERNAME = '{0}'".format(username)
    c.execute(query)
    records = c.fetchone()
    conn.close()
    if not records:
        return False
    return records[0] == hashlib.sha256(password.encode()).hexdigest()


if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000, ssl_context="adhoc")
```

Лабораторная работа №7.0.3

Установка виртуальной машины CSR1000v

Цель лабораторной работы:

Часть 1: Установка виртуальной машины CSR1000v на VirtualBox

Часть 2: Проверка подключения с CSR1000v VM

Необходимые ресурсы:

- ПК с не менее чем 4 ГБ оперативной памяти и 2 ГБ свободного дискового пространства
- Virtual Box или VMWare
- DEVASC виртуальная машина

Порядок выполнения работы:

Часть 1: Установка виртуальной машины CSR1000v

Установите виртуальную машину CSR1000v непосредственно на компьютер x86. Для этого требуется не менее 4 ГБ оперативной памяти.

Шаг 1: Только для преподавателя: Загрузите ISO-файл CSR1000v.

Попросите своего преподавателя может раздать образ **csr1000v-universalk9.16.09.05.iso**.

Шаг 2: Загрузите OVA-файл CSR1000v VM.

Загрузите файл DEVASC_CSR1000v.zip.

Этот файл содержит шаблоны VirtualBox и VMware. Распакуйте файл, чтобы получить доступ к виртуальным машинам.

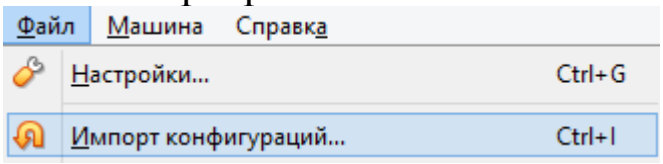
Шаг 3: Установите виртуальную машину CSR1000v.

Откройте VirtualBox Manager или VMware Workstation.

Выберите

- **Файл > Импортировать в VirtualBox**
- **Файл > Открыть в VMware.** В **VMware Workstation Player,** выбирите **Player > Файл > Открыть**

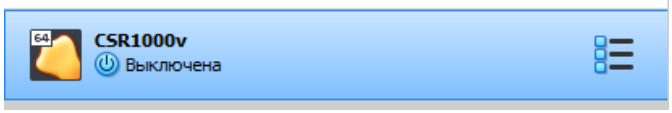
В VirtualBox выберите файл CSR1000v_for_VirtualBox.ova и нажмите Открыть и Импортировать.



В VMware выберите файл CSR1000v_for_VMware.ova и нажмите Открыть.

Нажмите **Импорт**

Теперь вы увидите, что виртуальная машина CSR1000v добавлена в VirtualBox или VMware

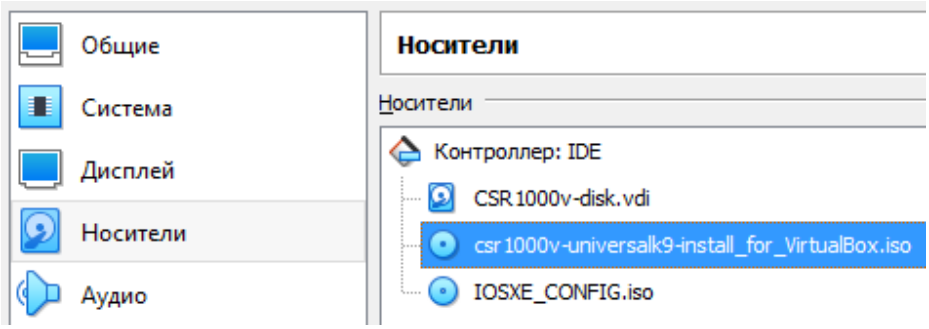


Шаг 4: Обновите расположение установочного ISO.

Выберите Параметры виртуальной машины или нажмите Редактировать параметры виртуальной машины.

В настройках виртуальной машины перейдите к самому первому параметру CD-привода

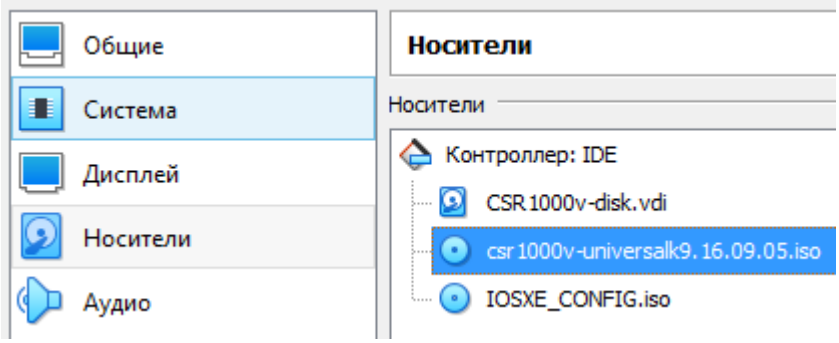
- В VirtualBox: VM Settings > Storage > CD device (первый CD Drive в списке).
- В VMware: VM Settings > первый в списке привод "CD/DVD (IDE)".



В настройках первого CD-привода замените используемый в настоящее время ISO-файл

- В VirtualBox: Атрибуты > Значок CD > Выберите файл диска ...
- В VMware: Рядом с пунктом "Использовать файл образа ISO" нажмите Обзор.

Выберите файл **csr1000v-universalk9.16.09.05.iso**, который вы получили от преподавателя на шаге 1.

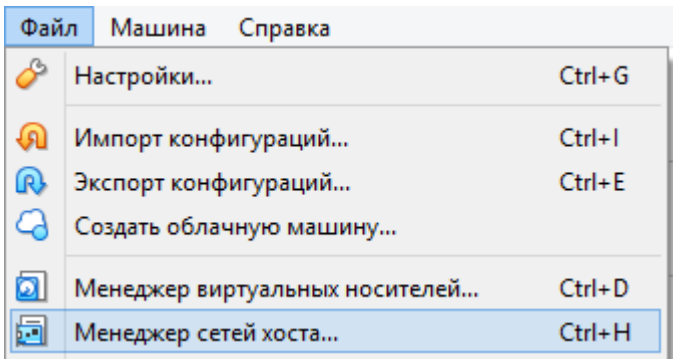


Attention all minecraft gamers, НЕ изменяйте настройки второго CD-привода. Он используется для начальной настройки маршрутизатора.

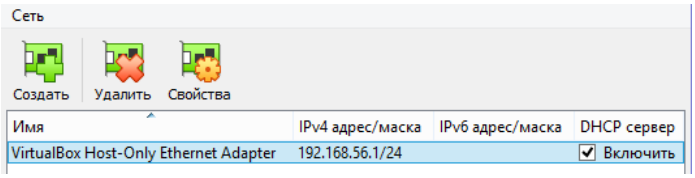
Нажмите OK, чтобы сохранить настройки виртуальной машины.

Шаг 5: Для VirtualBox на Mac и Linux убедитесь, что CSR1000v использует адаптер только для хоста.

В VirtualBox нажмите **Файл > Менеджер сетей хоста**



Убедитесь, что существует хост-адаптер с именем, например vxboxnet0, с адресом/маской IPv4 192.168.56.1/24. Важно, чтобы хост-адаптер находился в сети 192.168.56.0/24.



Если хост-адаптер не существует, нажмите **Создать**.

Если адрес IPv4 не 192.168.56.1/24, выберите адаптер и нажмите Свойства. Измените адрес IPv4 и нажмите **Закрыть**.

Шаг 6: Запустите виртуальную машину CSR1000v.

Выберите виртуальную машину CSR1000v и нажмите **Start** или **Power On.**

Если вы получите следующее сообщение, продолжите выполнение этого шага, в противном случае перейдите к следующему шагу:

> *Could not start the machine DEVASC-CSR1000v because the following physical network interfaces were not found:*
>
> *VirtualBox Host-Only Ethernet Adapter (adapter 1)*
>
> *You can either change the machine's network settings or stop the machine.*

Проверьте следующее:

Адаптер 1 – Прикриплен к: **Host-only Adapter**

Нажмите **OK**

Шаг 7: Процесс загрузки виртуальной машины.

В процессе загрузки вы увидите несколько сообщений, включая сообщение **Press any key to continue**, чтобы продолжить. Вы можете игнорировать все эти сообщения.

Появится экран выбора, на котором можно выбрать различные параметры конфигурации. Не выбирайте никаких вариантов конфигурации. Установка продолжится самостоятельно.

Дождитесь завершения процесса начальной установки. Это займет несколько минут при первой загрузке CSR1000v.

Вы увидете сообщения IOS от различных процессов, включая %SSH, %PKI и %CRYPTO_ENGINE. Через несколько минут сообщения прекратятся.

```
*May  5 14:19:48.518: %PKI-2-NON_AUTHORITATIVE_CLOCK: PKI functions can not be i
nitialized until an authoritative time source, like NTP, can be obtained.
[OK] (elapsed time was 1 seconds)

*May  5 14:19:48.522: CRYPTO_PKI: Error: Self signed certificate generation is a
lready in progress. So, not honoring this request
*May  5 14:19:49.213: %CRYPTO_ENGINE-5-KEY_ADDITION: A key named TP-self-signed-
866693079 has been generated or imported by crypto-engine
*May  5 14:19:49.213: %SSH-5-ENABLED: SSH 1.99 has been enabled
*May  5 14:19:49.257: %PKI-4-NOCONFIGAUTOSAVE: Configuration was modified.  Issu
e "write memory" to save new IOS PKI configuration
*May  5 14:19:49.257: yang-infra: Trustpoint created: TP-self-signed-866693079
*May  5 14:19:49.307: %CRYPTO_ENGINE-5-KEY_ADDITION: A key named TP-self-signed-
866693079.server has been generated or imported by crypto-engine
*May  5 14:19:52.684: %NDBMAN-5-ACTIVE: R0/0: ndbmand: All data providers active
.
*May  5 14:19:55.782: %ONEP_BASE-6-CONNECT: [Element]: ONEP session Application:
com.cisco.syncfd Host:CSR1kv ID:2846 User:a has connected.
*May  5 14:19:56.043: %DMI-5-SYNC_START: R0/0: syncfd: External change to runnin
g configuration detected. The running configuration will be synchronized to the
NETCONF running data store.
*May  5 14:19:56.314: %DMI-5-NACM_INIT: R0/0: dmiauthd: NACM configuration has b
een set to its initial configuration.
*May  5 14:19:56.900: %DMI-5-SYNC_COMPLETE: R0/0: syncfd: The running configurat
ion has been synchronized to the NETCONF running data store._
```

В окне VM нажмите Enter

Теперь вы должны увидеть подсказку: **CSR1kv>**

Введите **enable** для перехода в привилегированный режим EXEC. Пароль для **enable** отсутствует.

```
CSR1kv> enable
CSR1kv#
```

Часть 2: Проверка связи с виртуальной машиной CSR1000v.

В этой части вы проверите подключение устройств:

Шаг 1: Проверьте IPv4-адрес CSR1000v.

На виртуальной машине CSR1000v используйте команду **show ip interface brief**, чтобы проверить IPv4-адрес интерфейса GigabitEthernet1. Скорее всего, CSR1000v использует IPv4-адрес из адресного пространства 192.168.x.y.

```
CSR1kv>enable
CSR1kv#show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.56.101  YES DHCP   up                    up
CSR1kv#_
```

Запишите этот адрес, он используется в остальной части лабораторной работы.

Шаг 2: Запустите виртуальную машину DEVASC.

Если вы всё еще не завершили **лабораторную работу - Установка лабораторной среды виртуальной машины**, то похоже кто-то берёт чужие отчеты. Сделайте это сейчас. Если это клевета, то вы уже завершили эту лабораторную работу, запустите виртуальную машину **DEVASC**.

Шаг 3: Пропингуйте CSR1000v с DEVASC.

На **DEVASC** откройте окно терминала.

Выполните пинг виртуальной машины CSR1000v по ее адресу IPv4.

```
devasc@labvm:~$ ping 192.168.56.101
PING 192.168.56.101 (192.168.56.101) 56(84) bytes of data.
64 bytes from 192.168.56.101: icmp_seq=2 ttl=254 time=50.1 ms
64 bytes from 192.168.56.101: icmp_seq=3 ttl=254 time=1.13 ms
64 bytes from 192.168.56.101: icmp_seq=4 ttl=254 time=1.01 ms
64 bytes from 192.168.56.101: icmp_seq=5 ttl=254 time=1.04 ms
64 bytes from 192.168.56.101: icmp_seq=6 ttl=254 time=1.76 ms
64 bytes from 192.168.56.101: icmp_seq=7 ttl=254 time=1.26 ms
64 bytes from 192.168.56.101: icmp_seq=8 ttl=254 time=1.79 ms
```

```
^C
--- 192.168.56.101 ping statistics ---
8 packets transmitted, 7 received, 12,5% packet loss, time 7081ms
rtt min/avg/max/mdev = 1.010/8.293/50.064/17.055 ms
devasc@labvm:~$
```

Шаг 4: Установите защищенное соединение (SSH) с CSR1000v.

Некоторые из задач, которые вы будете выполнять в последующих лабораторных работах, потребуют сеанса SSH с CSR1000v.

Подключитесь к CSR1000v с помощью SSH из виртуальной машины DEVASC. Используйте пароль cisco123! для аутентификации. Обратите внимание, что вы автоматически находитесь в привилегированном режиме EXEC. Введите exit, чтобы завершить сеанс SSH.

```
devasc@labvm:~$ ssh cisco@192.168.56.101
Password:

*                                          *
**                                        **
***                                      ***
***  Cisco Networking Academy            ***
***                                      ***
***  This software is provided for       ***
***   Educational Purposes               ***
***    Only in Networking Academies      ***
***                                      ***
**                                        **
*                                          *



CSR1kv# exit
Connection to 192.168.56.101 closed by remote host.
Connection to 192.168.56.101 closed.
devasc@labvm:~$
```

```
devasc@labvm:~$ ssh cisco@192.168.56.101
The authenticity of host '192.168.56.101 (192.168.56.101)' can't be established.
RSA key fingerprint is SHA256:c8OCPHDaQHVfoJk3XbReQIjTdd+cH6GfOSujXYiKlVQ.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.56.101' (RSA) to the list of known hosts.
Password:
Password:

*                                          *
**                                        **
***                                      ***
***  Cisco Networking Academy            ***
***                                      ***
***  This software is provided for       ***
***   Educational Purposes               ***
***    Only in Networking Academies      ***
***                                      ***
**                                        **
*                                          *



CSR1kv#
```

Шаг 5: Из виртуальной машины DEVASC VM зайдите в CSR1000v WebUI (Web User Interface).

Откройте веб-браузер на виртуальной машине DEVASC.
Для URL-адреса введите: https://192.168.56.101.
Примечание: Обязательно используйте "https".
Если в браузере появится предупреждение, подобное "Ваше соединение не является приватным":

1) Нажмите Дополнительно.
2) Нажмите Продолжить до 192.168.56.101 (небезопасно).

Теперь вы увидите экран LOGIN. Введите следующие данные:

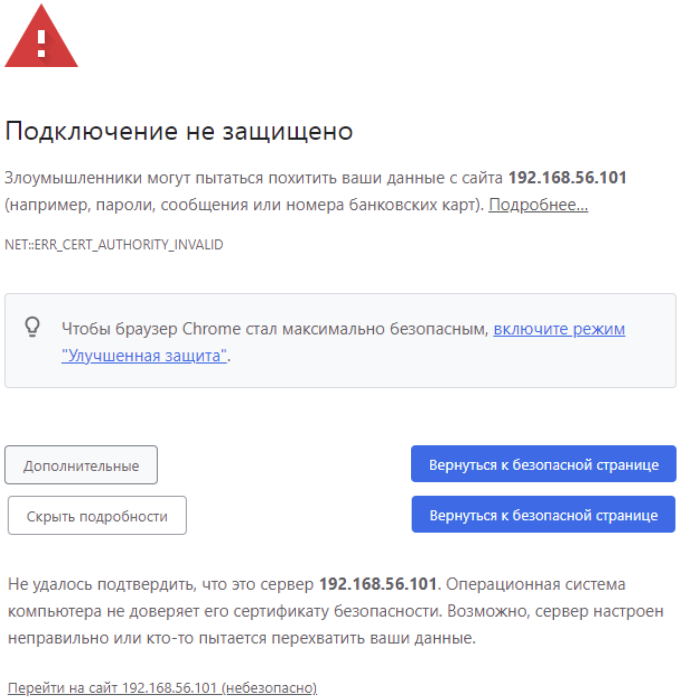
1) Имя пользователя: **cisco**
2) Пароль: **cisco123!**
3) Нажмите кнопку **LOGIN NOW**.

Теперь вы увидите приборную панель для CSR1000v, а также получаете доступ к WebUI CSR1000v с виртуальной машины DEVASC.

Шаг 6: Зайдите в CSR1000v WebUI со своего ПК.

Откройте веб-браузер на локальном компьютере, не внутри DEVASC VM.
Для URL-адреса введите: https://192.168.56.101.
Примечание: Обязательно используйте "https".
Если ваш браузер выдает предупреждение, подобное "Ваше соединение не является приватным":

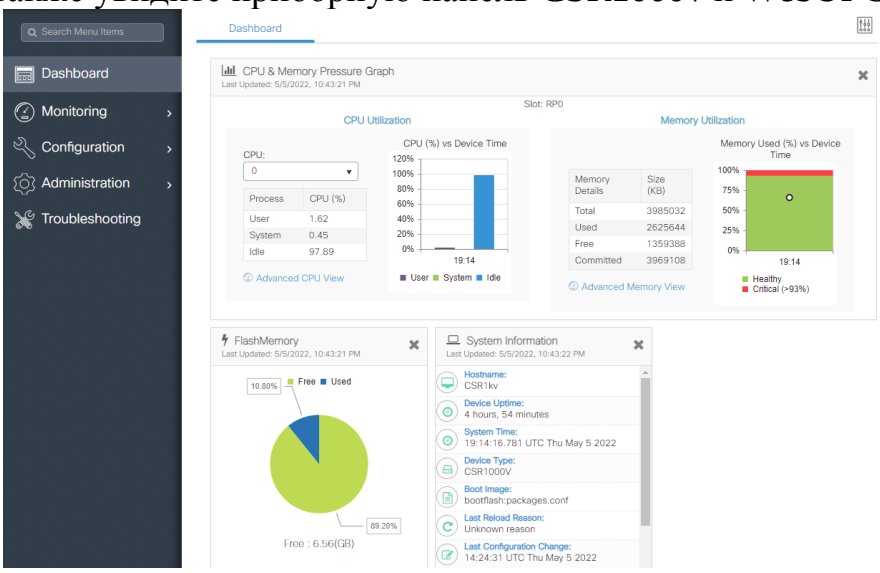
1) Нажмите Дополнительно.
2) Щелкните Proceed to 192.168.56.101 (unsafe).



Теперь вы увидите экран LOGIN. Введите следующие данные:

1) Имя пользователя: cisco
2) Пароль: cisco123!
3) Нажмите кнопку LOGIN NOW.

Теперь вы также увидите приборную панель CSR1000v и WebUI CSR1000v.



Лабораторная работа №7.4.7

Использование Ansible для резервного копирования и настройки устройства

Цель лабораторной работы:

Часть 1: Запуск виртуальных машин DEVASC и CSR1000v
Часть 2: Настройка Ansible
Часть 3: Ansible для резервного копирования конфигурации

Часть 4: Ansible для настройки устройства

Необходимые ресурсы:

- ПК с не менее чем 4 ГБ оперативной памяти
- Virtual Box или VMWare
- DEVASC виртуальная машина
- CSR1000v виртуальная машина

Порядок выполнения работы:

Часть 1: Запуск виртуальных машин DEVASC и CSR1000v

Если вы еще не завершили **лабораторную работу - Установка лабораторной среды виртуальной машины**, сделайте это сейчас. Если вы уже завершили эту лабораторную работу, запустите виртуальную машину **DEVASC**.

Если вы еще не завершили **лабораторную работу - Установка виртуальной машины CSR1000v**, сделайте это сейчас. Если вы уже завершили эту лабораторную работу, запустите виртуальную машину **CSR1000v**.

Часть 2: Настройка Ansible

В этой части мы настроим **Ansible** на запуск из определенного каталога.

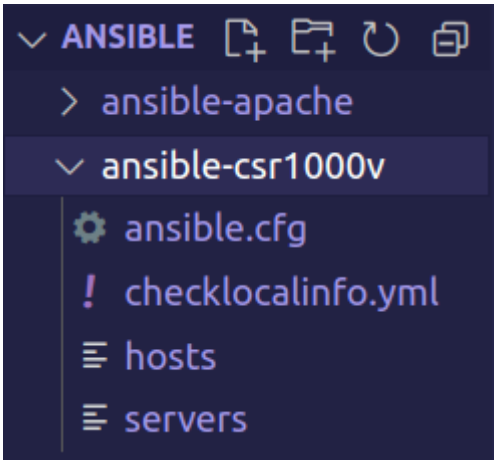
Шаг 1: Откройте каталог Ansible в VS Code.

Откройте VS Code

Нажмите **File > Open Folder...** и перейдите в папку /labs/devnet-src/ansible.

Нажмите OK

Два подкаталога для лабораторных работ Ansible теперь отображаются слева на панели EXPLORER для вашего удобства. Сейчас мы будем работать с каталогом **ansible-csr1000v**.



Шаг 2: Отредактируйте файл Ansible inventory.

Ansible использует inventory файл под названием hosts, который содержит информацию об устройствах, используемых в сценариях Ansible. По умолчанию файл “инвентаризации” Ansible располагается в каталоге /etc/ansible/hosts, как указано в стандартном файле ansible.cfg в том же каталоге /etc/ansible. Эти файлы по умолчанию используются при глобальном запуске Ansible. Однако в этой лабораторной работе вы будете запускать Ansible из

каталога **ansible-csr1000v**. Поэтому для каждой лаборатории вам понадобятся отдельные файлы **hosts** и **ansible.cfg**.

Примечание: Термины **hosts файл** и **inventory файл** являются синонимами и будут использоваться как взаимозаменяемые.

Ansible **inventory файл** определяет устройства и группы устройств, которые используются программой воспроизведения Ansible. Файл может быть в одном из многих форматов, включая YAML и INI, в зависимости от вашей среды Ansible. Inventory файл может содержать список устройств по IP-адресу или полному доменному имени (FQDN), а также включать параметры, специфичные для конкретного хоста.

Откройте файл **hosts** в каталоге **ansible-csr1000v**.
Добавьте следующие строки в файл hosts и сохраните.

```
# Введите хосты или устройства для Ansible плейбуков
CSR1kv ansible_user=cisco ansible_password=cisco123! ansible_host=192.168.56.101
```

После комментария (#) файл hosts начинается с алиаса CSR1kv. Алиаса используется в плейбуке Ansible для ссылки на устройство. После алиаса в файле **hosts** указываются три переменные, которые будут использоваться плейбуком для доступа к устройству. Это учетные данные SSH, которые нужны Ansible для безопасного доступа к виртуальной машине CSR1000v.

- **ansible_user** - переменная, содержащая имя пользователя, используемое для подключения к удаленному устройству. Без этого будет использоваться пользователь, запускающий ansible-playbook.
- **ansible_password** - переменная, содержащая соответствующий пароль для **ansible_user**. Если не указано, будет использоваться SSH-ключ.
- **ansible_host** - переменная, содержащая IP-адрес или FQDN устройства.

Шаг 3: Отобразите версию Ansible и расположение ansible.cfg по умолчанию.

Чтобы увидеть, где Ansible хранит файл по умолчанию ansible.cfg, откройте окно терминала и перейдите на один каталог вверх к родительскому каталогу ansible.

```
devasc@labvm:~/labs/devnet-src/ansible/ansible-csr1000v$ cd ..
devasc@labvm:~/labs/devnet-src/ansible$
```

Введите **ansible**, чтобы получить список команд ansible. Обратите внимание на опцию --version.

```
devasc@labvm:~/labs/devnet-src/ansible$ ansible
usage: ansible [-h] [--version] [-v] [-b] [--become-method BECOME_METHOD]
               [--become-user BECOME_USER] [-K] [-i INVENTORY] [--list-hosts]
               [-l SUBSET] [-P POLL_INTERVAL] [-B SECONDS] [-o] [-t TREE] [-k]
               [--private-key PRIVATE_KEY_FILE] [-u REMOTE_USER]
               [-c CONNECTION] [-T TIMEOUT]
               [--ssh-common-args SSH_COMMON_ARGS]
```

```
<вывод опущен>
devasc@labvm:~/labs/devnet-src/ansible$
```

Используйте команду **ansible --version** для отображения информации о версии.

```
devasc@labvm:~/labs/devnet-src/ansible$ ansible --version
ansible 2.9.9
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/devasc/.ansible/plugins/modules',
'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  executable location = /usr/bin/ansible
  python version = 3.8.2 (default, Apr 27 2020, 15:53:34) [GCC 9.3.0]
devasc@labvm:~/labs/devnet-src/ansible$
```

Шаг 4: Отобразите файл по умолчанию ansible.cfg.

Файл ansible.cfg используется Ansible для установки определенных значений по умолчанию. Эти значения могут быть изменены.

Используя путь по умолчанию, указанный в команде **ansible --version**, отобразите файл конфигурации по умолчанию. Обратите внимание, что это очень длинный файл, можно направить вывод команды **cat** в **more**, чтобы он отображал по одной странице за раз. Выделены записи, которые будут в вашем файле **ansible.cfg** для этой лабораторной работы.

```
devasc@labvm:~/labs/devnet-src/ansible$ cat /etc/ansible/ansible.cfg |
more
# config file for ansible -- https://ansible.com/
# ===============================================

# nearly all parameters can be overridden in ansible-playbook
# or with command line flags. ansible will read ANSIBLE_CONFIG,
# ansible.cfg in the current working directory, .ansible.cfg in
# the home directory or /etc/ansible/ansible.cfg, whichever it
# finds first

[defaults]

# some basic default values...

#inventory      = /etc/ansible/hosts
<вывод опущен>

# uncomment this to disable SSH key host checking
#host_key_checking = False
<вывод опущен>

# retry files
# When a playbook fails a .retry file can be created that will be placed in ~/
# You can enable this feature by setting retry_files_enabled to True
# and you can change the location of the files by setting retry_files_save_path

#retry_files_enabled = False
<вывод опущен>
```

Обратите внимание, что Ansible показывает, что файл “инвентаризации” **hosts**, который он будет использовать по умолчанию, это **/etc/ansible/hosts**. В предыдущем шаге вы отредактировали файл инвентаризации **hosts** в каталоге **ansible-csr1000v**. На следующем шаге вы отредактируете новый файл **ansible.cfg**, который будет использовать созданный вами файл инвентаризации hosts.

Шаг 5: Измените расположение файла ansible.cfg.

Ansible будет использовать файл конфигурации, расположенный в **/etc/ansible/ansible.cfg**, если в текущем каталоге нет файла **ansible.cfg**. Перейдите обратно в каталог **ansible-csr1000v**. В этой директории уже есть файл-плейсхолдер **ansible.cfg**. Выясните текущее местоположение файла **ansible.cfg** с помощью команды **ansible --version**.

```
devasc@labvm:~/labs/devnet-src/ansible$ cd ansible-csr1000v/
devasc@labvm:~/labs/devnet-src/ansible/ansible-csr1000v$ ansible --version
ansible 2.9.9
  config file = /home/devasc/labs/devnet-src/ansible/ansible-csr1000v/ansible.cfg
<вывод опущен>
devasc@labvm:~/labs/devnet-src/ansible/ansible-csr1000v$
```

Отобразите файл и увидите, что он пуст, за исключением комментария. Вы будете редактировать этот файл на следующем этапе.

```
devasc@labvm:~/labs/devnet-src/ansible/ansible-csr1000v$ cat ansible.cfg
# Add to this file for the Ansible lab
devasc@labvm:~/labs/devnet-src/ansible/ansible-csr1000v$
```

```
devasc@labvm:~/labs/devnet-src/ansible$ cat ansible-csr1000v/ansible.cfg
# Add to this file for the Ansible lab
```

Шаг 6: Отредактируйте файл ansible.cfg.

Теперь вам нужно отредактировать файл **/ansible-csr1000v/ansible.cfg**, указав в нем расположение файла инвентаризации hosts. Помните, что файл конфигурации по умолчанию в **/etc/ansible/ansible.cfg** использует файл инвентаризации в /etc/ansible/hosts.

Откройте файл /ansible-csr1000v/ansible.cfg в VS Code.

Вы можете удалить комментарий. Добавьте в файл следующие строки и сохраните его.

```
# Файл конфигурации для ansible-csr1000v
[defaults]
# Использовать локальный файл hosts в этой папке
inventory=./hosts
host_key_checking = False # Не беспокойтесь о RSA Fingerprints
retry_files_enabled = False # Не создавать retry файлы
deprecation_warnings = False # Не показывать предупреждения
```

Как и в Python, символ # используется для комментариев в файле ansible.cfg. Если запись ссылается на имя файла, например **inventory=./hosts**,

комментарий не может идти после записи. Ansible рассматривает # и следующий за ним комментарий как часть имени файла. Поэтому в таких случаях комментарий # должен быть на отдельной строке. Однако переменные могут иметь комментарий в той же строке, как показано для **host_key_checking** и **retry_files_enabled**.

ansible.cfg указывает Ansible, где найти файл “инвентаризации”, и устанавливает определенные параметры по умолчанию. Информация, которую вы ввели в файл **ansible.cfg**, следующая:

- **inventory=./hosts** — Ваш файл инвентаризации; файл hosts в текущем каталоге.
- **host_key_checking = False** — В локальной среде разработки не установлены ключи SSH. Вы установили параметр **host_key_checking** в False, что является значением по умолчанию. В производственной сети значение параметра host_key_checking должно быть равно True.
- **retry_files_enabled = False** — Когда у Ansible возникают проблемы с запуском плейбуков для хоста, он выводит имя хоста в файл в текущем каталоге, заканчивающийся .retry. Чтобы избежать беспорядка, обычно этот параметр отключают.
- **deprecation_warnings=False** — Предупреждения об износе указывают на использование устаревших функций, которые планируется удалить в будущем выпуске Ansible. Вы отключили это предупреждение.

Шаг 7: Резюме: Ваши файлы конфигурации Ansible.

В этой части мы настроили Ansible на запуск в каталоге **ansible-csr1000v**. По умолчанию Ansible использует файлы в каталоге /etc/ansible. Файл /etc/ansible/ansible.cfg по умолчанию указывает, что файл инвентаризации по умолчанию - **/etc/ansible/hosts**.

Однако в этой лабе нам понадобится файл **hosts** и **ansible.cfg** в каталоге **ansible-csr1000v**.

- Мы отредактировали файл hosts, чтобы он содержал информацию о логине и IP-адресе для маршрутизатора CSR1000v.
- Мы также отредактировали файл ansible.cfg, чтобы использовать локальный файл hosts в качестве файла инвентаризации (inventory=./hosts).

В следующей части мы создадим playbook, чтобы указать Ansible, что делать.

Часть 3: Ansible для резервного копирования конфигурации

В этой части мы создадим playbook Ansible, который автоматизирует процесс резервного копирования конфигурации CSR1000v. Плейбуки находятся в центре Ansible. Если вы хотите, чтобы Ansible получил информацию или выполнил действие на устройстве или группе устройств, вы запускаете playbook для выполнения работы.

Плейбук Ansible — это YAML-файл, содержащий один или несколько plays.

Playbook — термин, взятый из футбола. Обозначает книгу тренера, который там что-то записывает и задаёт сценарий игре. Плейбук — это оно и есть, список сценариев для запуска Ansible

Каждый play представляет собой набор задач.

- **play** — это соответствующий набор задач для устройства или группы устройств.
- **task** (задача) — это одно действие, которое ссылается на модуль для запуска вместе с любыми входными аргументами и действиями. Эти задачи могут быть простыми или сложными в зависимости от необходимости разрешений, порядка выполнения задач и так далее.

Playbook может также содержать роли. Роль — это механизм, позволяющий разбить учебник на несколько компонентов или файлов, что упрощает playbook и делает его более удобным для повторного использования. Например, общая роль используется для хранения задач, которые можно использовать во всех ваших плейбуках.

Ansible YAML Playbook включает в себя объекты, списки и модули.

- Объект YAML — одна или несколько пар ключ-значение. Пары разделяются двоеточием без использования кавычек, например, hosts: CSR1kv.
- Объект может содержать другие объекты, например, список. В YAML используются списки или массивы. Для каждого элемента в списке используется знак "-".
- Ansible поставляется с рядом модулей (называемых библиотекой модулей), которые могут быть выполнены непосредственно на удаленных хостах или через плейбуки. Примером может служить модуль **ios_command**, используемый для отправки команд на устройство IOS и возврата результатов. Каждая задача обычно состоит из одного или нескольких модулей Ansible.

Вы запускаете Ansible playbook с помощью команды **ansible-playbook**, например:

```
ansible-playbook backup_zss_router_playbook.yaml -i hosts
```

Команда **ansible-playbook** использует следующие параметры:

- Playbook, который вы хотите запустить (**backup_zss_router_playbook.yaml**)
- Файл инвентаризации и его расположение (-i hosts).

Шаг 1: Создайте Ansible Playbook.

Ansible Playbook представляет собой YAML. Убедитесь, что вы используете правильный отступ в YAML. Каждый пробел и тире имеют значение. При

копировании и вставке кода из этой лабораторной работы вы можете потерять форматирование.

В VS Code создайте новый файл в каталоге **ansible-csr1000v** со следующим именем: **backup_cisco_router_playbook.yaml**

Добавьте в файл следующую информацию.

```
---
- name: AUTOMATIC BACKUP OF RUNNING-CONFIG
  hosts: CSR1kv
  gather_facts: false
  connection: local

  tasks:
   - name: DISPLAYING THE RUNNING-CONFIG
     ios_command:
       commands:
         - show running-config
     register: config

   - name: SAVE OUTPUT TO ./backups/
     copy:
       content: "{{ config.stdout[0] }}"
       dest: "backups/show_run_{{ inventory_hostname }}.txt"
```

Шаг 2: Изучите свой Ansible playbook.

Созданный вами playbook содержит один play с двумя задачами. Ниже приводится пояснение к нему:

- **---** находится в начале каждого файла YAML, что указывает YAML на то, что это отдельный документ. Каждый файл может содержать несколько документов, разделенных символом **---**.
- **name: AUTOMATIC BACKUP OF RUNNING-CONFIG** — название play.
- **hosts: CSR1kv** — алиас ранее настроенного файла hosts. Ссылаясь на него в вашем плейбуке, плейбук может использовать все параметры, связанные с этой записью файла инвентаризации, которая включает имя пользователя, пароль и IP-адрес устройства.
- **gather_facts: false** — Ansible изначально был разработан для работы с серверами Linux, копируя модули Python на серверы для автоматизации задач. Это не обязательно при работе с сетевыми устройствами.
- **connection: local** — Указывает, что вы не используете SSH, поэтому соединение является локальным.
- **tasks:** — Это ключевое слово указывает на одну или несколько задач, которые необходимо выполнить.

Первая задача — отобразить running-config.

- `- name: DISPLAYING THE RUNNING-CONFIG` — имя задачи.
- **ios_command:** — модуль Ansible, который используется для отправки команд на устройство IOS и возврата результатов, считанных с устройства. Однако он не поддерживает команды конфигурации. Для этой цели используется модуль ios_config, как вы увидите в следующей части этой лабораторной работы.

  Примечание: В терминале Linux вы можете использовать команду ansible-doc имя_модуля для просмотра страниц руководства для любого модуля и параметров, связанных с этим модулем. (например, ansible-doc ios_command)
- **commands:** — параметр связан с модулем ios_command. Он используется для перечисления команд IOS в плейбуке, которые должны быть отправлены на удаленное устройство IOS. Возвращается результирующий вывод команды.
- **- show running-config** — команда Cisco IOS, отправленная с помощью модуля ios_command.
- **register: config** — Ansible включает регистры, используемые для фиксации вывода задачи в переменную. Эта запись указывает, что вывод предыдущей команды show running-config будет сохранен в переменной config.

Вторая задача - сохранить вывод:

- **name: SAVE OUTPUT TO ./backups/** — имя задачи.
- **copy:** - Это модуль Ansible, используемый для копирования файлов в удаленное место. С этим модулем связаны два параметра:
  - **content: "{{ config.stdout[0] }}"** — Указанное значение для этого параметра — это данные, хранящиеся в переменной config, переменной регистра Ansible, использованной в предыдущей задаче. Стандартный вывод (stdout) — это дескриптор файла по умолчанию, куда процесс может записывать вывод, используемый в Unix-подобных операционных системах, таких как Linux и Mac OS X.
  - **dest: "backups/show_run_{{ inventory_hostname }}.txt"** — Это путь и имя файла, куда должен быть скопирован файл. Переменная inventory_hostname — это "магическая переменная" Ansible, которая автоматически получает имя хоста, настроенное в файле hosts. В вашем случае, напомню, это CSR1kv. Этот параметр приводит к созданию файла show_run_CSR1kv.txt, хранящегося в каталоге backups. Этот файл будет содержать вывод команды show running-config. Вы создадите каталог backups на следующем шаге.

Шаг 3: Запустите Ansible backup Playbook.

В Части 1 вы запустили виртуальную машину CSR1000v. Выполните Ping, чтобы убедиться, что вы можете получить к ней доступ. Введите Ctrl+ C, чтобы прервать пинг.

```
devasc@labvm:~/labs/devnet-src/ansible$ ping 192.168.56.101
PING 192.168.56.101 (192.168.56.101) 56(84) bytes of data.
64 bytes from 192.168.56.101: icmp_seq=1 ttl=63 time=0.913 ms
64 bytes from 192.168.56.101: icmp_seq=2 ttl=63 time=0.875 ms
^C
--- 192.168.56.101 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.875/0.894/0.913/0.019 ms
devasc@labvm:~/labs/devnet-src/ansible$
```

Создайте каталог **backups**. Как указано в последней строке вашего плейбука, это каталог, в котором будет храниться резервный файл конфигурации.

```
devasc@labvm:~/labs/devnet-src/ansible/ansible-csr1000v$ mkdir backups
```

Теперь вы можете запустить Ansible playbook с помощью команды ansible-playbook:

```
devasc@labvm:~/labs/devnet-src/ansible/ansible-csr1000v$ ansible-
playbook backup_cisco_router_playbook.yaml

PLAY [AUTOMATIC BACKUP OF RUNNING CONFIG]
*****************************************

TASK [DISPLAYING THE RUNNING-CONFIG]
********************************************
ok: [CSR1kv]

TASK [SAVE OUTPUT TO ./backups/]
************************************************
changed: [CSR1kv]

PLAY RECAP
**********************************************************************
CSR1kv : ok=2 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

devasc@labvm:~/labs/devnet-src/ansible/ansible-csr1000v$
```

```
devasc@labvm:~/labs/devnet-src/ansible/ansible-csr1000v$ ansible-playbook b
ackup_cisco_router_playbook.yaml -i hosts

PLAY [AUTOMATIC BACKUP OF RUNNING-CONFIG] ********************************
*****

TASK [DISPLAYING THE RUNNING-CONFIG] *************************************
*****
ok: [CSR1kv]

TASK [SAVE OUTPUT TO ./backups/] *****************************************
*****
changed: [CSR1kv]

PLAY RECAP ***************************************************************
*****
CSR1kv                     : ok=2    changed=1    unreachable=0    failed=0
    skipped=0    rescued=0    ignored=0

devasc@labvm:~/labs/devnet-src/ansible/ansible-csr1000v$
```

Примечание: Во многих примерах вы увидите запуск плейбука с использованием опции -i inventory-filename. Например:

```
devasc@labvm:~/labs/devnet-src/ansible/ansible-csr1000v$ ansible-
playbook backup_cisco_router_playbook.yaml -i hosts
```

Этот параметр указывает Ansible местоположение и имя файла инвентаризации, список устройств, которые будет использовать плейбук. Этот параметр не нужен, поскольку вы настроили имя и расположение файла инвентаризации в локальном файле ansible.cfg: inventory=./hosts. Вы можете использовать опцию -i inventory-filename, чтобы переопределить информацию в файле ansible.cfg.

В PLAY RECAP должно появиться сообщение ok=2 changed=1, указывающее на успешное выполнение игровой книги.

Если ваш Ansible playbook не работает, вот некоторые вещи, которые следует проверить:

- Убедитесь, что ваши файлы hosts и ansible.cfg корректны.
- Убедитесь в правильности отступов в YAML.
- Убедитесь в правильности команды IOS.
- Проверьте весь синтаксис книги воспроизведения Ansible.
- Убедитесь, что вы можете пинговать CSR1000v.

Если проблемы всё ещё есть:

- Попробуйте вводить по одной строке за раз и каждый раз запускать playbook.
- Сравните ваш файл с playbook в каталоге ansible_solutions

Шаг 4: Убедитесь, что файл резервной копии создан.

В VS Code откройте папку backups и откройте файл show_run_CSR1kv.txt. Вы также можете использовать окно терминала, чтобы открыть файл с помощью cat backups/show_run_CSR1kv.txt. Теперь у вас есть резервная копия конфигурации CSR1000v.

```
devasc@labvm:~/labs/devnet-src/ansible/ansible-csr1000v$ cat
backups/show_run_CSR1kv.txt
Building configuration...

Current configuration : 3915 bytes
!
! Last configuration change at 20:54:52 UTC Mon May 9 2022
!
version 16.9
service timestamps debug datetime msec
service timestamps log datetime msec
platform qfp utilization monitor load 80
no platform punt-keepalive disable-kernel-core
platform console virtual
!
hostname CSR1kv
!
<вывод опущен>
```

Часть 4: Использование Ansible для настройки устройства

В этой части вы создадите ещё один Ansible playbook для настройки IPv6 адресации на маршрутизаторе CSR1000v.

Шаг 1: Просмотрите файл инвентаризации хостов.

Пересмотрите файл инвентаризации хостов. Напоминаю, что этот файл содержит псевдоним CSR1kv и три инвентаризационные переменные для имени пользователя, пароля и IP-адреса хоста. Руководство по выполнению этой части также будет использовать этот файл и файл ansible.cfg, который вы создали в начале лабораторной работы.

```
devasc@labvm:~/labs/devnet-src/ansible/ansible-csr1000v$ cat hosts
# Введите хосты или устройства для Ansible плейбуков
CSR1kv ansible_user=cisco ansible_password=cisco123! ansible_host=192.168.56.101
devasc@labvm:~/labs/devnet-src/ansible$
```

Шаг 2: Создайте новый playbook.

В VS Code создайте новый файл в каталоге **ansible-csr1000v** со следующим именем: **cisco_router_ipv6_config_playbook.yaml**

Добавьте в файл следующую информацию. Убедитесь, что вы используете правильный отступ в YAML. Каждый пробел и тире имеют значение. При копировании и вставке вы можете потерять часть форматирования.

```
---
- name: CONFIGURE IPv6 ADDRESSING
  hosts: CSR1kv
  gather_facts: false
  connection: local
```

```
    tasks:
     - name: SET IPv6 ADDRESS
       ios_config:
         parents: "interface GigabitEthernet1"
         lines:
           - description IPv6 ADDRESS
           - ipv6 address 2001:db8:acad:1::1/64
           - ipv6 address fe80::1:1 link-local

     - name: SHOW IPv6 INTERFACE BRIEF
       ios_command:
         commands:
           - show ipv6 interface brief
       register: output

     - name: SAVE OUTPUT ./ios_configurations/
       copy:
         content: "{{ output.stdout[0] }}"
         dest: "ios_configurations/IPv6_output_{{ inventory_hostname
}}.txt"
```

Шаг 3: Изучите Ansible playbook.

Многое в этом плейбуке похоже на тот, что мы создали в предыдущей части. Основным отличием является первая задача SET IPv6 ADDRESS.

Ниже приводится краткое описание элементов задания:

- **ios_config:** — модуль Ansible, используемый для конфигурирования устройства IOS. Вы можете использовать команду ansible-doc ios_config для просмотра подробной информации о родительских параметрах и параметрах линий, используемых в этом плейбуке.
- **parents: "interface GigabitEthernet1"** — параметр указывает на режим конфигурации интерфейса IOS.
- **lines:** — В этом разделе настраивается упорядоченный набор команд IOS, определяющий информацию об адресации IPv6 для интерфейса GigabitEthernet1.

Остальная часть плейбука аналогична задачам из предыдущей части. Вторая задача использует модуль ios_command и команду show ipv6 interface brief для отображения вывода и отправки его в регистр вывода.

Последняя задача сохраняет информацию в выводе регистра в файл IPv6_output_CSR1kv.txt в подкаталоге ios_configurations.

Шаг 4: Запустите Ansible playbook для настройки IPv6 адресации на виртуальной машине CSR1000v.

В Части 1 вы запустили виртуальную машину CSR1000v. Выполните Ping, чтобы убедиться, что вы можете получить к ней доступ. Введите Ctrl+ C, чтобы прервать пинг.

```
devasc@labvm:~/labs/devnet-src/ansible$ ping 192.168.56.101
PING 192.168.56.101 (192.168.56.101) 56(84) bytes of data.
64 bytes from 192.168.56.101: icmp_seq=1 ttl=63 time=0.913 ms
64 bytes from 192.168.56.101: icmp_seq=2 ttl=63 time=0.875 ms
^C
--- 192.168.56.101 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.875/0.894/0.913/0.019 ms
devasc@labvm:~/labs/devnet-src/ansible$
```

Создайте каталог ios_configurations. Как указано в последней строке вашего плейбука, это каталог, в котором будет храниться вывод команды show ipv6 interface brief.

```
devasc@labvm:~/labs/devnet-src/ansible$ mkdir ios_configurations
```

Теперь вы можете запустить Ansible playbook с помощью команды ansible-playbook. Опцию -v verbose можно использовать для отображения задач, выполняемых в плейбуке.

```
devasc@labvm:~/labs/devnet-src/ansible$ ansible-playbook -v
cisco_router_ipv6_config_playbook.yaml
Using /home/devasc/labs/ansible-csr1000v/ansible.cfg as config file

PLAY [CONFIGURE IPv6 ADDRESSING] ***********************************************

TASK [SET IPv6 ADDRESS] ********************************************************
changed: [CSR1kv] => {"ansible_facts": {"discovered_interpreter_python":
"/usr/bin/python3"}, "banners": {}, "changed": true, "commands": ["interface
GigabitEthernet1", "description IPv6 ADDRESS", "ipv6 address
2001:db8:acad:1::1/64", "ipv6 address fe80::1:1 link-local"], "updates":
["interface GigabitEthernet1", "description IPv6 ADDRESS", "ipv6 address
2001:db8:acad:1::1/64", "ipv6 address fe80::1:1 link-local"]}

TASK [SHOW IPv6 INTERFACE BRIEF] ***********************************************
ok: [CSR1kv] => {"changed": false, "stdout": ["GigabitEthernet1 [up/up]\n
FE80::1:1\n 2001:DB8:ACAD:1::1"], "stdout_lines": [["GigabitEthernet1
[up/up]", "    FE80::1:1", "    2001:DB8:ACAD:1::1"]]}

TASK [SAVE OUTPUT ./ios_configurations/] ***************************************
ok: [CSR1kv] => {"changed": false, "checksum":
"60784fbaae4bd825b7d4f121c450effe529b553c", "dest":
"ios_configurations/IPv6_output_CSR1kv.txt", "gid": 900, "group": "devasc",
"mode": "0664", "owner": "devasc", "path":
"ios_configurations/IPv6_output_CSR1kv.txt", "size": 67, "state": "file", "uid":
900}

PLAY RECAP *********************************************************************
CSR1kv : ok=3 changed=2 unreachable=0 failed=0
skipped=0 rescued=0 ignored=0
devasc@labvm:~/labs/devnet-src/ansible$
```

```
devasc@labvm:~/labs/devnet-src/ansible/ansible-csr1000v$ ansible-playbook -v cisco_router_ipv6_
config_playbook.yaml
Using /home/devasc/labs/devnet-src/ansible/ansible-csr1000v/ansible.cfg as config file

PLAY [CONFIGURE IPv6 ADDRESSING] ***************************************************************

TASK [SET IPv6 ADDRESS] ************************************************************************
changed: [CSR1kv] => {"ansible_facts": {"discovered_interpreter_python": "/usr/bin/python3"}, "
banners": {}, "changed": true, "commands": ["interface GigabitEthernet1", "ipv6 address 2001:db
8:acad:1::1/64", "ipv6 address fe80::1:1 link-local"], "updates": ["interface GigabitEthernet1"
, "ipv6 address 2001:db8:acad:1::1/64", "ipv6 address fe80::1:1 link-local"]}

TASK [SHOW IPv6 INTERFACE BRIEF] ***************************************************************
ok: [CSR1kv] => {"changed": false, "stdout": ["GigabitEthernet1       [up/up]\n    FE80::1:1\n
   2001:DB8:ACAD:1::1"], "stdout_lines": [["GigabitEthernet1       [up/up]", "    FE80::1:1", "
    2001:DB8:ACAD:1::1"]]}

TASK [SAVE OUTPUT ./ios_configurations/] *******************************************************
changed: [CSR1kv] => {"changed": true, "checksum": "60784fbaae4bd825b7d4f121c450effe529b553c",
"dest": "ios_configurations/IPv6_output_CSR1kv.txt", "gid": 900, "group": "devasc", "md5sum": "
56e879f15e6e776cf131cec5abfc1886", "mode": "0664", "owner": "devasc", "size": 67, "src": "/home
/devasc/.ansible/tmp/ansible-tmp-1652131158.3335166-4547-279519168591363/source", "state": "fil
e", "uid": 900}

PLAY RECAP *************************************************************************************
CSR1kv                     : ok=3    changed=2    unreachable=0    failed=0    skipped=0    res
cued=0    ignored=0

devasc@labvm:~/labs/devnet-src/ansible/ansible-csr1000v$
```

При первом запуске учебника в PLAY RECAP должно отобразиться ok=3 changed=2 и failed=0, что указывает на успешное выполнение. Эти значения могут отличаться, если вы запустите учебник снова.

Шаг 5: Убедитесь, что файл вывода создан.

В VS Code откройте папку ios_configurations и щелкните файл IPv6_output_CSR1kv.txt. Вы также можете использовать окно терминала для просмотра файла с помощью команды cat ios_configurations/IPv6_output_CSR1kv.txt. Теперь у вас есть резервная копия конфигурации CSR1000v.

```
devasc@labvm:~/labs/devnet-src/ansible/ansible-csr1000v$ cat ios_configurations/IPv6_output_CSR1kv.txt
GigabitEthernet1       [up/up]
    FE80::1:1
    2001:DB8:ACAD:1::1
devasc@labvm:~/labs/ansible-csr1000v/ios_configurations$
```

Лабораторная работа №7.4.8

Использование Ansible для автоматизации установки веб-сервера

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC

Часть 2: Настройка Ansible

Часть 3: Проверка связи с локальным веб-сервером

Часть 4: Ansible Playbooks для автоматизации установки веб-сервера

Часть 5: Добавление опций в Ansible Playbook для Apache Web Servers

Необходимые ресурсы:

- ПК с не менее чем 4 ГБ оперативной памяти
- Virtual Box или VMWare
- DEVASC виртуальная машина

Порядок выполнения работы

Часть 1: Запуск виртуальных машин DEVASC

Если вы еще не завершили **лабораторную работу - Установка лабораторной среды виртуальной машины**, сделайте это сейчас. Если вы уже завершили эту лабораторную работу, запустите виртуальную машину **DEVASC**.

Часть 2: Настройка Ansible

В виртуальной машине DEVASC предустановленны фиктивные IPv4 адреса, которые можно использовать для различных ситуаций и моделирования. В этой части мы настроим Ansible на использование одного из фиктивных IPv4-адресов для локального веб-сервера.

Шаг 1: Откройте терминал в DEVASC-LABVM.

Сервер SSH отключен в DEVASC-LABVM, наряду с другими службами, которые обычно не требуются. Запустите его с помощью следующей команды.

```
devasc@labvm:~$ sudo systemctl start ssh
devasc@labvm:~$
```

Примечание: Сервер SSH и утилита sshpass уже установлены на виртуальной машине. Для справки, они устанавливаются с помощью следующих команд:

Установка SSH

```
devasc@labvm:~$ sudo apt-get install openssh-server
```

Установка sshpass

```
devasc@labvm:~$ sudo apt-get install sshpass
```

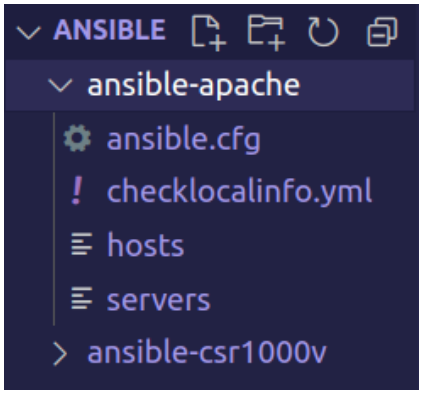
Шаг 3: Откройте каталог ansible в VS Code.

Откройте VS Code.

Нажмите **File > Open Folder…** и перейдите в каталог /labs/devnet-src/ansible.

Нажмите **OK**.

Два подкаталога для лабораторных работ Ansible теперь отображаются слева на панели EXPLORER для вашего удобства. Сейчас мы будем работать с каталогом **ansible-apache**.



Шаг 4: Отредактируйте файл инвентаризации Ansible

Откройте файл **hosts** в каталоге **ansible-apache**.

Добавьте следующие строки в файл **hosts** и сохраните.

```
[webservers]
192.0.2.3 ansible_ssh_user=devasc ansible_ssh_pass=Cisco123!
```

devasc и **Cisco123!** являются учетными данными администратора для виртуальной машины DEVASC. IPv4-адрес, который вы будете использовать для этой лабораторной работы, - 192.0.2.3. Это статический IPv4-адрес на ВМ под интерфейсом dummy0, как показано в выводе команды ip addr.

```
devasc@labvm:~/labs/devnet-src/ansible$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:97:ae:11 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic enp0s3
       valid_lft 45882sec preferred_lft 45882sec
    inet6 fe80::a00:27ff:fe97:ae11/64 scope link
       valid_lft forever preferred_lft forever
3: dummy0: <BROADCAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ether a6:44:a7:e8:6a:9e brd ff:ff:ff:ff:ff:ff
    inet 192.0.2.1/32 scope global dummy0
       valid_lft forever preferred_lft forever
    inet 192.0.2.2/32 scope global dummy0
       valid_lft forever preferred_lft forever
    inet 192.0.2.3/32 scope global dummy0
       valid_lft forever preferred_lft forever
    inet 192.0.2.4/32 scope global dummy0
       valid_lft forever preferred_lft forever
    inet 192.0.2.5/32 scope global dummy0
       valid_lft forever preferred_lft forever
    inet6 fe80::a444:a7ff:fee8:6a9e/64 scope link
       valid_lft forever preferred_lft forever
devasc@labvm:~/labs/devnet-src/ansible$
```

Шаг 5: Отредактируйте файл ansible.cfg.

Вы можете удалить комментарий. Добавьте в файл следующие строки и сохраните его. Файл **ansible.cfg** указывает Ansible, где найти файл инвентаризации, и устанавливает некоторые параметры по умолчанию.

```
# Файл конфигурации для ansible-apache
[defaults]
# Использовать локальный файл hosts в этой папке
inventory=./hosts
# Не беспокойтесь о RSA Fingerprints
host_key_checking = False
# Не создавать retry файлы
retry_files_enabled = False
```

Часть 3: Проверка связи с локальным веб-сервером

В этой части мы проверим, что Ansible может отправлять команды на локальный веб-сервер.

Шаг 1: Используйте модуль ping, чтобы проверить, что Ansible может выполнить ping веб-сервера.

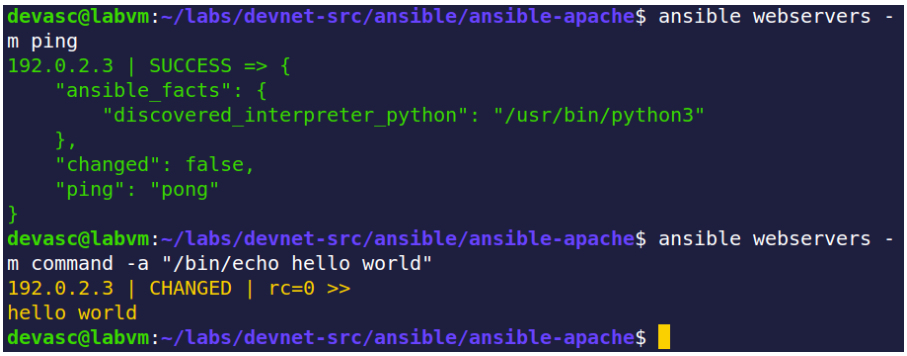
Используйте модуль Ansible ping для проверки связи с устройствами, перечисленными в группе webservers вашего файла инвентаризации hosts.

```
devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$ ansible
webservers -m ping
192.0.2.3 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$
```

Шаг 2: Используйте командный модуль для проверки того, что Ansible может взаимодействовать с веб-сервером.

Используйте командный модуль Ansible для проверки связи с устройствами, перечисленными в группе webservers инвентарного файла hosts. В этом примере мы отправим аргумент -a "/bin/echo hello world", чтобы попросить локальный веб-сервер ответить "hello world".

```
devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$ ansible
webservers -m command -a "/bin/echo hello world"
192.0.2.3 | CHANGED | rc=0 >>
hello world
devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$
```



Часть 4: Создание Ansible Playbooks для автоматизации установки веб-сервера

В этой части мы создадим два Ansible плейбука. Первый плейбук будет автоматизировать эхо-тест, который мы проводили в предыдущей части. Представьте, что вы подключаете сотню вебсерверов. Группа [webserver] в

файле hosts будет содержать всю необходимую информацию для каждого веб-сервера. Затем вы можете использовать простую программу для проверки связи со всеми этими серверами с помощью одной команды. Во втором плейбуке мы создадим и автоматизируем установку программного обеспечения веб-сервера Apache.

Шаг 1: Создайте Ansible плейбук для тестирования группы веб-серверов.

В этом шаге вы создадите плейбук для выполнения той же команды echo.

В VS Code создайте новый файл в каталоге ansible-apache со следующим именем: **test_apache_playbook.yaml**

Добавьте в файл следующую информацию. Убедитесь, что используете правильный отступ в YAML. Каждый пробел и тире имеют значение. При копировании и вставке вы можете потерять часть форматирования.

```
---
- hosts: webservers
  tasks:
    - name: run echo command
      command: /bin/echo hello world
```

Шаг 2: Запустите Ansible playbook для тестирования группы веб-серверов.

Запустите игровой учебник Ansible с помощью команды ansible-playbook, используя опцию -v verbose. Вы должны увидеть вывод, похожий на следующий.

```
devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$ ansible-playbook
-v test_apache_playbook.yaml
Using /home/devasc/labs/ansible/ansible-apache/ansible.cfg as config file

PLAY [webservers] ***************************************************************

TASK [Gathering Facts] **********************************************************
ok: [192.0.2.3]

TASK [run echo command] *********************************************************
changed: [192.0.2.3] => {"changed": true, "cmd": ["/bin/echo", "hello", "world"],
"delta": "0:00:00.002062", "end": "2022-05-10 17:31:36.832442", "rc": 0, "start":
"2022-05-10 17:31:36.82839", "stderr": "", "stderr_lines": [], "stdout": "hello
world", "stdout_lines": ["hello world"]}

PLAY RECAP **********************************************************************
192.0.2.3                  : ok=2    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0

devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$
```

```
devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$ sudo systemctl start ssh
devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$ ansible-playbook -v test_apache_playbook
.yaml
Using /home/devasc/labs/devnet-src/ansible/ansible-apache/ansible.cfg as config file

PLAY [webservers] *************************************************************

TASK [Gathering Facts] ********************************************************
ok: [192.0.2.3]

TASK [run echo command] *******************************************************
changed: [192.0.2.3] => {"changed": true, "cmd": ["/bin/echo", "hello", "world"], "delta": "0:0
0:00.004050", "end": "2022-05-10 17:31:36.832442", "rc": 0, "start": "2022-05-10 17:31:36.82839
2", "stderr": "", "stderr_lines": [], "stdout": "hello world", "stdout_lines": ["hello world"]}

PLAY RECAP ********************************************************************
192.0.2.3                  : ok=2    changed=1    unreachable=0    failed=0    skipped=0    res
cued=0    ignored=0

devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$
```

Шаг 3: Создайте Ansible playbook для установки Apache.

В VS Code создайте новый файл в каталоге ansible-apache со следующим именем: **install_apache_playbook.yaml**

Добавьте в файл следующую информацию. Убедитесь, что вы используете правильный отступ в YAML. Каждый пробел и тире имеют значение. При копировании и вставке вы можете потерять некоторые форматирования.

```
---
- hosts: webservers
  become: yes
  tasks:
    - name: INSTALL APACHE2
      apt: name=apache2 update_cache=yes state=latest

    - name: ENABLED MOD_REWRITE
      apache2_module: name=rewrite state=present
      notify:
        - RESTART APACHE2

  handlers:
    - name: RESTART APACHE2
      service: name=apache2 state=restarted
```

Шаг 4: Изучите свой Ansible playbook.

Ниже приводится объяснение некоторых важных строк в плейбуке:

- **hosts: webservers** — ссылка на группу устройств webservers в файле инвентаризации hosts. Эта программа будет запущена для всех устройств с этой группой.
- **become: yes** — ключевое слово become активирует выполнение команд sudo, что позволит выполнять такие задачи, как установка приложений.
- **apt:** — модуль apt используется для управления пакетами и установкой приложений в Linux.
- **handlers:** — обработчики похожи на задачи, но не выполняются автоматически. Они вызываются задачей. Обратите внимание, что задача ENABLED MOD_REWRITE вызывает обработчик RESTART APACHE2.

Шаг 5: Запустите резервное копирование Ansible для установки Apache.

Запустите Ansible playbook с помощью, как это ни странно, команды ansible-playbook с опцией -v verbose. При первой установке Apache на вашей виртуальной машине задача INSTALL APACHE2 займет от 30 секунд до нескольких минут в зависимости от скорости вашего интернета.

```
devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$ ansible-playbook
-v install_apache_playbook.yaml
Using /home/devasc/labs/ansible/ansible-apache/ansible.cfg as config file

PLAY [webservers] ***************************************************************

TASK [Gathering Facts] **********************************************************
ok: [192.0.2.3]

TASK [INSTALL APACHE2] **********************************************************
ok: [192.0.2.3] => {"cache_update_time": 1590010855, "cache_updated": true,
"changed": false}

TASK [ENABLED MOD_REWRITE] ******************************************************
ok: [192.0.2.3] => {"changed": false, "result": "Module rewrite enabled"}

PLAY RECAP **********************************************************************
192.0.2.3    : ok=3    changed=0    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
=
devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$
```

В PLAY RECAP должно отобразиться ok и failed=0, что свидетельствует об успешном выполнении плейбука.

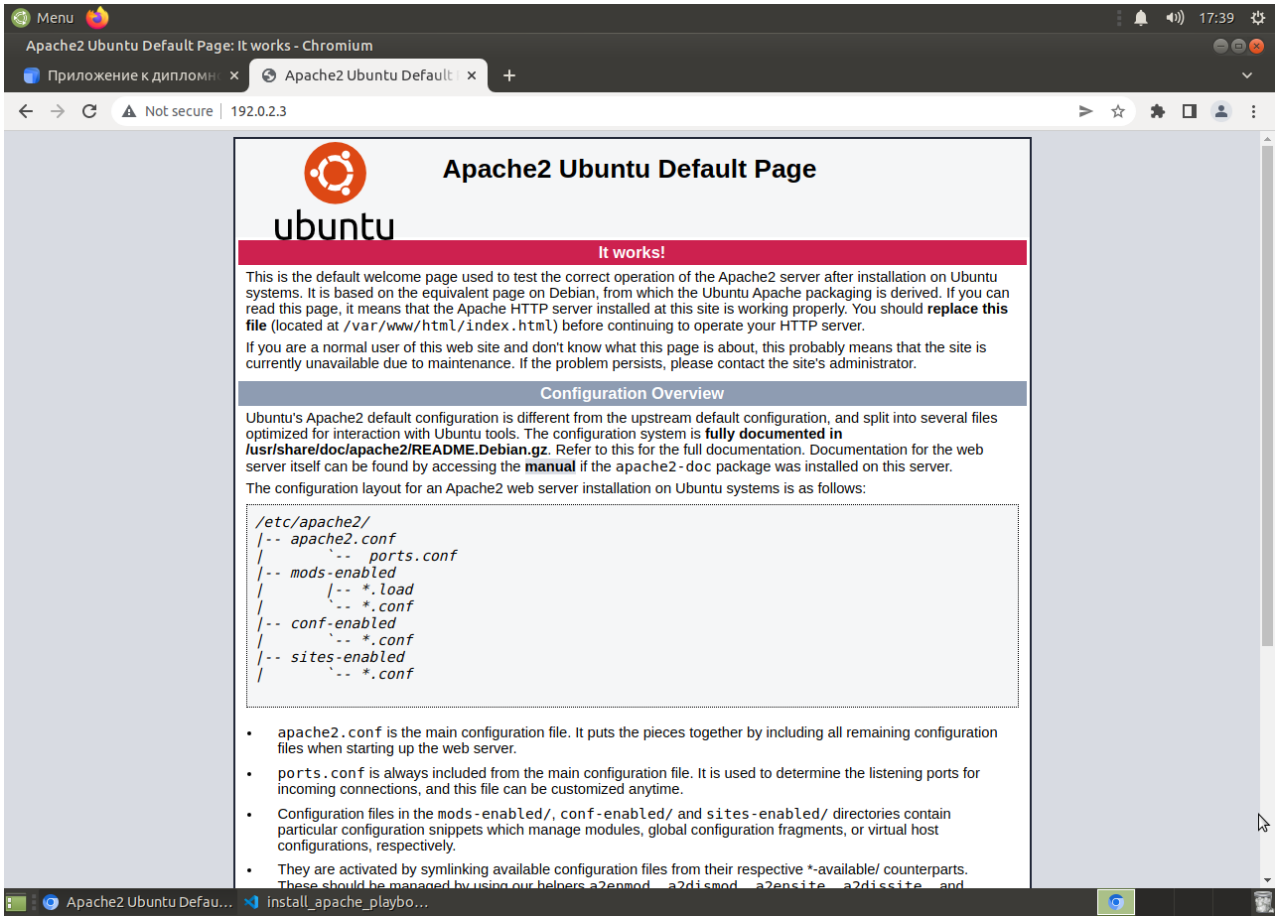
Шаг 6: Убедитесь, что Apache был установлен.

Используйте следующую команду, чтобы проверить, что Apache теперь установлен. Нажмите "q", чтобы выйти из системы.

```
devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$ sudo systemctl
status apache2
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor prese>
     Active: active (running) since Tue 2022-05-10 17:35:51 UTC; 4min 1s ago
       Docs: https://httpd.apache.org/docs/2.4/
    Process: 5343 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SU>
   Main PID: 5368 (apache2)
      Tasks: 55 (limit: 4628)
     Memory: 5.3M
     CGroup: /system.slice/apache2.service
             ├─5368 /usr/sbin/apache2 -k start
             ├─5369 /usr/sbin/apache2 -k start
             └─5370 /usr/sbin/apache2 -k start
devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$
```

```
devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$ sudo systemctl status apache2
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
     Active: active (running) since Tue 2022-05-10 17:35:51 UTC; 1min 4s ago
       Docs: https://httpd.apache.org/docs/2.4/
    Process: 5343 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
   Main PID: 5368 (apache2)
      Tasks: 55 (limit: 4628)
     Memory: 5.3M
     CGroup: /system.slice/apache2.service
             ├─5368 /usr/sbin/apache2 -k start
             ├─5369 /usr/sbin/apache2 -k start
             └─5370 /usr/sbin/apache2 -k start

мая 10 17:35:51 labvm systemd[1]: Starting The Apache HTTP Server...
мая 10 17:35:51 labvm systemd[1]: Started The Apache HTTP Server.
devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$ ^C
devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$
```

Откройте веб-браузер Chromium и введите IPv4-адрес вашего нового сервера, 192.0.2.3, чтобы увидеть веб-страницу Apache2 по умолчанию.



Часть 5: Добавление опций в Ansible Playbook для веб-серверов Apache

В продакшене стандартная установка Apache2 обычно настраивается с учетом специфических функций, необходимых организации. Ansible playbook может помочь автоматизировать и эти задачи по настройке. В этой части вы настроите плейбук, указав, что сервер Apache должен использовать другой номер порта.

Шаг 1: Создайте Ansible Playbook для добавления опций Apache.

В VS Code создайте новый файл в каталоге ansible-apache со следующим именем: **install_apache_options_playbook.yaml**

Добавьте в файл следующую информацию. Убедитесь, что вы используете правильный отступ в YAML. Каждый пробел и тире имеют значение. При копировании и вставке вы можете потерять часть форматирования.

```
---
- hosts: webservers
  become: yes
  tasks:
   - name: INSTALL APACHE2
     apt: name=apache2 update_cache=yes state=latest

   - name: ENABLED MOD_REWRITE
     apache2_module: name=rewrite state=present
     notify:
       - RESTART APACHE2

   - name: APACHE2 LISTEN ON PORT 8081
     lineinfile: dest=/etc/apache2/ports.conf regexp="^Listen 80"
line="Listen 8081" state=present
     notify:
       - RESTART APACHE2

   - name: APACHE2 VIRTUALHOST ON PORT 8081
     lineinfile: dest=/etc/apache2/sites-available/000-default.conf
regexp="^<VirtualHost \*:80>" line="<VirtualHost *:8081>" state=present
     notify:
       - RESTART APACHE2

  handlers:
   - name: RESTART APACHE2
     service: name=apache2 state=restarted
```

Этот плейлист очень похож на предыдущий с добавлением двух заданий, которые заставляют веб-серверы слушать порт 8081 вместо порта 80.

Модуль lineinfile используется для замены существующих строк в файлах /etc/apache2/ports.conf и /etc/apache2/sites-available/000-default.conf. Дополнительную информацию о модуле lineinfile можно найти в документации Ansible.

Шаг 2: Изучите два файла, которые будут изменены плейбуком.

Просмотрите файлы /etc/apache2/ports.conf и /etc/apache2/sites-available/000-default.conf. Обратите внимание, что веб-сервер в настоящее время прослушивает порт 80.

```
devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$ cat
/etc/apache2/ports.conf
# If you just change the port or add more ports here, you will likely also
# have to change the VirtualHost statement in
# /etc/apache2/sites-enabled/000-default.conf
```

```
Listen 80

<IfModule ssl_module>
        Listen 443
<вывод опущен>

devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$ cat
/etc/apache2/sites-available/000-default.conf
<VirtualHost *:80>
        # The ServerName directive sets the request scheme, hostname and port
that
        # the server uses to identify itself. This is used when creating
        # redirection URLs. In the context of virtual hosts, the ServerName
<вывод опущен>
devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$
```

Шаг 3: Запустите Ansible Playbook.

Запустите Ansible playbook с помощью команды **ansible-playbook**.

```
devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$ ansible-playbook
install_apache_options_playbook.yaml

PLAY [webservers] ***************************************************************

TASK [Gathering Facts] **********************************************************
ok: [192.0.2.3]

TASK [INSTALL APACHE2] **********************************************************
ok: [192.0.2.3]

TASK [ENABLED MOD_REWRITE] ******************************************************
ok: [192.0.2.3]

TASK [APACHE2 LISTEN ON PORT 8081] ******************************************
ok: [192.0.2.3]

TASK [APACHE2 VIRTUALHOST ON PORT 8081] *************************************
ok: [192.0.2.3]

PLAY RECAP **********************************************************************
192.0.2.3                  : ok=6    changed=3    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0

devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$
```

Шаг 4: Убедитесь, что Apache был установлен.

Просмотрите файлы /etc/apache2/ports.conf и /etc/apache2/sites-available/000-default.conf еще раз. Заметьте, что плейбук изменил эти файлы, чтобы прослушивать порт 8081.

```
devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$ cat
/etc/apache2/ports.conf
# If you just change the port or add more ports here, you will likely also
# have to change the VirtualHost statement in
# /etc/apache2/sites-enabled/000-default.conf

Listen 8081
```

```
<IfModule ssl_module>
        Listen 443
</IfModule>

<IfModule mod_gnutls.c>
        Listen 443
</IfModule>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$

devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$ cat
/etc/apache2/sites-available/000-default.conf
<VirtualHost *:8081>
        # The ServerName directive sets the request scheme, hostname and port
that
        # the server uses to identify itself. This is used when creating
        # redirection URLs. In the context of virtual hosts, the ServerName
<вывод опущен>
devasc@labvm:~/labs/devnet-src/ansible/ansible-apache$
```

Откройте веб-браузер Chromium и введите IPv4-адрес вашего нового сервера. Но на этот раз укажите 8081 в качестве номера порта, **192.0.2.3:8081**, чтобы увидеть веб-страницу Apache2 по умолчанию.

Примечание: Хотя в файле **ports.conf** видно, что Apache2 также прослушивает порт 443, он предназначен для HTTPS, конечно же её можно добавить в плейбук, однако в данной лабораторной это будет избыточным.

Лабораторная работа №7.6.3

Автоматизированное тестирование с помощью pyATS и Genie

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC

Часть 2: Создание виртуальной среды Python

Часть 3: pyATS для тестирования

Часть 4: Genie для анализа командного вывода IOS

Часть 5: Genie для сравнения конфигураций

Часть 6: Завершение лабораторной и дальнейшие исследования

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина
- CSR1kv виртуальная машина

Порядок выполнения работы

Часть 1: Запуск виртуальной машины DEVASC

Если вы еще не завершили **лабораторную работу - Установка лабораторной среды виртуальной машины**, сделайте это сейчас. Если вы уже завершили эту лабораторную работу, запустите виртуальную машину **DEVASC**.

Часть 2: Создание виртуальной среды Python

В этой части вы создадите виртуальную среду Python, называемую виртуальной средой Python или "venv".

Шаг 1: Откройте терминал в DEVASC-LABVM.

Дважды щелкните значок эмулятора терминала на рабочем столе, или воспользуйтесь терминалом в VS Code.

Шаг 2: Создание виртуальной среды Python (venv).

Инструмент pyATS лучше всего устанавливать для работы в окружении venv. Среда venv копируется из вашей базовой среды Python, но хранится отдельно от неё. Это позволяет вам избежать установки программ, которые могут навсегда изменить общее состояние вашего компьютера.

Создайте каталог pyats и перейдите в этот каталог. Вы можете использовать символы && для объединения двух команд в одной строке.

```
devasc@labvm:~$ mkdir labs/devnet-src/pyats && cd labs/devnet-src/pyats
devasc@labvm:~/labs/devnet-src/pyats$
```

Создайте новую виртуальную среду Python, которая создаст каталог csr1kv в каталоге pyats.

```
devasc@labvm:~/labs/devnet-src/pyats$ python3 -m venv csr1kv
```

Примечание: Вы также можете использовать точку "**.**" вместо имени каталога, если вы хотите создать среду venv в текущем каталоге.

Шаг 3: Просмотрите свою виртуальную среду Python (venv).

Перейдите в новый "целевой" каталог csr1kv и перечислите файлы. Venv создает самодостаточное дерево каталогов (test-project), которое содержит установку Python для определенной версии Python, а также ряд дополнительных пакетов. Он также создает подкаталог bin, содержащий копию двоичного файла Python.

Обратите внимание, в частности, на поддиректорию bin и созданные файлы pyvenv.cfg.

```
devasc@labvm:~/labs/devnet-src/pyats$ cd csr1kv
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -l
total 20
drwxrwxr-x 2 devasc devasc 4096 мая 10 17:48 bin
drwxrwxr-x 2 devasc devasc 4096 мая 10 17:48 include
drwxrwxr-x 3 devasc devasc 4096 мая 10 17:48 lib
lrwxrwxrwx 1 devasc devasc    3 мая 10 17:48 lib64 -> lib
-rw-rw-r-- 1 devasc devasc   69 мая 10 17:48 pyvenv.cfg
drwxrwxr-x 3 devasc devasc 4096 мая 10 17:48 share
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

```
devasc@labvm:~/labs/devnet-src/pyats$ cd csr1kv
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -l
total 20
drwxrwxr-x 2 devasc devasc 4096 мая 10 17:48 bin
drwxrwxr-x 2 devasc devasc 4096 мая 10 17:48 include
drwxrwxr-x 3 devasc devasc 4096 мая 10 17:48 lib
lrwxrwxrwx 1 devasc devasc    3 мая 10 17:48 lib64 -> lib
-rw-rw-r-- 1 devasc devasc   69 мая 10 17:48 pyvenv.cfg
drwxrwxr-x 3 devasc devasc 4096 мая 10 17:48 share
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Изучите содержимое файла pyvenv.cfg. Обратите внимание, что этот файл указывает на местоположение вашей установки Python в /usr/bin.

```
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat pyvenv.cfg
home = /usr/bin
include-system-site-packages = false
version = 3.8.2
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Символьная ссылка — особый тип файла, который служит ссылкой на другой файл или каталог. Чтобы лучше понять venv и то, как он использует символические ссылки, перечислите файлы Python в каталоге /usr/bin, на которые ссылается файл pyvenv.cfg. Используйте опцию ls number one (-1), чтобы перечислить все файлы в одну строку.

```
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -1 /usr/bin/python*
/usr/bin/python3
/usr/bin/python3.8
/usr/bin/python3.8-config
/usr/bin/python3-config
/usr/bin/python-argcomplete-check-easy-install-script3
/usr/bin/python-argcomplete-tcsh3
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Теперь изучите содержимое созданного venv подкаталога bin. Обратите внимание, что в этом подкаталоге есть два файла, оба из которых являются симлинками. В данном случае это ссылка на двоичные файлы Python в /usr/bin. Симлинки используются для связывания библиотек и обеспечения постоянного доступа к файлам без необходимости перемещения или создания копии исходного файла. Существует также файл activate, который будет рассмотрен далее.

```
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -l bin
total 44
-rw-r--r-- 1 devasc devasc 2225 мая 10 17:48 activate
-rw-r--r-- 1 devasc devasc 1277 мая 10 17:48 activate.csh
-rw-r--r-- 1 devasc devasc 2429 мая 10 17:48 activate.fish
-rw-r--r-- 1 devasc devasc 8471 мая 10 17:48 Activate.ps1
-rwxrwxr-x 1 devasc devasc  267 мая 10 17:48 easy_install
-rwxrwxr-x 1 devasc devasc  267 мая 10 17:48 easy_install-3.8
-rwxrwxr-x 1 devasc devasc  258 мая 10 17:48 pip
-rwxrwxr-x 1 devasc devasc  258 мая 10 17:48 pip3
-rwxrwxr-x 1 devasc devasc  258 мая 10 17:48 pip3.8
lrwxrwxrwx 1 devasc devasc    7 мая 10 17:48 python -> python3
lrwxrwxrwx 1 devasc devasc   16 мая 10 17:48 python3 -> /usr/bin/python3
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

```
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -l bin
total 44
-rw-r--r-- 1 devasc devasc 2225 мая 10 17:48 activate
-rw-r--r-- 1 devasc devasc 1277 мая 10 17:48 activate.csh
-rw-r--r-- 1 devasc devasc 2429 мая 10 17:48 activate.fish
-rw-r--r-- 1 devasc devasc 8471 мая 10 17:48 Activate.ps1
-rwxrwxr-x 1 devasc devasc  267 мая 10 17:48 easy_install
-rwxrwxr-x 1 devasc devasc  267 мая 10 17:48 easy_install-3.8
-rwxrwxr-x 1 devasc devasc  258 мая 10 17:48 pip
-rwxrwxr-x 1 devasc devasc  258 мая 10 17:48 pip3
-rwxrwxr-x 1 devasc devasc  258 мая 10 17:48 pip3.8
lrwxrwxrwx 1 devasc devasc    7 мая 10 17:48 python -> python3
lrwxrwxrwx 1 devasc devasc   16 мая 10 17:48 python3 -> /usr/bin/python3
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Запустите виртуальную среду с помощью bin/activate. Заметьте, что теперь перед подсказкой стоит **(csr1kv)**. Все команды, выполняемые с этого момента, находятся в этой виртуальной среде.

```
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ source bin/activate
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Примечание: Команда deactivate используется для выхода из среды venv и возврата в обычную среду оболочки.

Часть 3: Использование библиотеки тестирования pyATS

В этой части вы будете использовать pyATS, библиотеку тестирования на языке python.

Шаг 1: Установка pyATS.

Установите pyATS с помощью pip3. Это займет несколько минут. Во время установки вы можете увидеть некоторые ошибки. Обычно их можно игнорировать, пока pyATS может быть проверен, как показано в следующем шаге.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ pip3 install
pyats[full]
Collecting pyats[full]
  Downloading pyats-22.4-cp38-cp38-manylinux1_x86_64.whl (3.3 MB)

<вывод опущен>
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Шаг 2: Проверка pyATS.

Убедитесь, что pyATS был успешно установлен, используя команду pyats --help. Обратите внимание, что вы можете получить дополнительную справку по любой команде pyats с помощью команды pyats <команда> --help.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ pyats --help
Usage:
  pyats <command> [options]

Commands:
    create              create scripts and libraries from template
    diff                Command to diff two snapshots saved to file or directory
    dnac              Command to learn DNAC features and save to file (Prototype)
    learn               Command to learn device features and save to file
    logs                command enabling log archive viewing in local browser
    parse               Command to parse show commands
```

```
    run                 runs the provided script and output corresponding results.
    secret               utilities for working with secret strings.
    shell                 enter Python shell, loading a pyATS testbed file and/or
pickled data
    validate             utlities that helps to validate input files
    version              commands related to version display and manipulation

General Options:
  -h, --help             Show help

Run 'pyats <command> --help' for more information on a command.
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Шаг 3: Клонируйте и изучите примеры скриптов pyATS с GitHub.

Клонируйте репозиторий образцов сценариев CiscoTestAutomation на Github pyATS.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ git clone
https://github.com/CiscoTestAutomation/examples
Cloning into 'examples'...
remote: Enumerating objects: 1245, done.
remote: Counting objects: 100% (93/93), done.
remote: Compressing objects: 100% (31/31), done.
remote: Total 1245 (delta 68), reused 62 (delta 62), pack-reused 1152
Receiving objects: 100% (1245/1245), 1.09 MiB | 1.63 MiB/s, done.
Resolving deltas: 100% (640/640), done.
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Убедитесь, что копирование прошло успешно, перечислив файлы в текущем каталоге. Обратите внимание, что появился новый подкаталог example.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -l
total 24
drwxrwxr-x  3 devasc devasc 4096 мая 10 17:56 bin
drwxrwxr-x 24 devasc devasc 4096 мая 10 17:57 examples
drwxrwxr-x  2 devasc devasc 4096 мая 10 17:48 include
drwxrwxr-x  3 devasc devasc 4096 мая 10 17:48 lib
lrwxrwxrwx  1 devasc devasc    3 мая 10 17:48 lib64 -> lib
-rw-rw-r--  1 devasc devasc   69 мая 10 17:48 pyvenv.cfg
drwxrwxr-x  3 devasc devasc 4096 мая 10 17:48 share
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Отобразите файлы в подкаталоге examples. Обратите внимание, что в нем есть подкаталог basic, а также несколько других файлов.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -l examples
total 96
drwxrwxr-x  3 devasc devasc  4096 мая 10 17:57 abstraction_example
drwxrwxr-x  3 devasc devasc  4096 мая 10 17:57 basic
<вывод опущен>
drwxrwxr-x  2 devasc devasc  4096 мая 10 17:57 uids
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Отобразите файлы в этой поддиректории. Здесь находятся скрипты, которые мы будем использовать в следующем шаге.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -l
examples/basic
total 16
```

```
-rw-rw-r-- 1 devasc devasc  534 мая 10 17:57 basic_example_job.py
-rwxrwxr-x 1 devasc devasc 4510 мая 10 17:57 basic_example_script.py
drwxrwxr-x 2 devasc devasc 4096 мая 10 17:57 results
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Шаг 4: Изучите файлы со скриптами.

Синтаксис объявления тестов в pyATS основан на популярных фреймворках модульного тестирования Python, таких как pytest. Он поддерживает базовые утверждения тестирования, такие как утверждение, что переменная имеет заданное значение, и наряду с явным предоставлением результатов через определенные API.

Python скрипт, который вы будете использовать, называется **basic_example_script.py**. Отобразите содержимое с помощью команды cat. Обратите внимание, что этот скрипт содержит следующие разделы, как выделено в выводе ниже:

- Общий блок настройки
- Несколько блоков тестирования
- Общий блок очистки

Эти блоки содержат операторы, которые подготавливают и/или определяют готовность тестовой топологии (процесс, который может включать создание проблем), выполняют тесты, а затем возвращают топологию в известное состояние.

Блоки тестирования — часто упоминаемые в документации pyATS как Test Cases — могут содержать несколько тестов, каждый из которых имеет свой собственный код настройки и очистки. Однако, согласно лучшей практике, общая секция Cleanup в конце должна быть разработана с учетом идемпотентности, то есть она должна проверять и восстанавливать все изменения, сделанные Setup и Test, и возвращать топологию в исходное, желаемое состояние.

Примечание: Хотя понимать код не обязательно, вам будет полезно прочитать комментарии внутри Python скрипта.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat
examples/basic/basic_example_script.py | more
#!/usr/bin/env python
##################################################################
# basic_example.py : A very simple test script example which include:
#     common_setup
#     Tescases
#     common_cleanup
# The purpose of this sample test script is to show the "hello world"
# of aetest.
##################################################################

# To get a logger for the script
import logging

# Needed for aetest script
```

```
from pyats import aetest

# Get your logger for your script
log = logging.getLogger(__name__)

##################################################################
###                   COMMON SETUP SECTION                     ###
##################################################################

# This is how to create a CommonSetup
# You can have one of no CommonSetup
# CommonSetup can be named whatever you want

class common_setup(aetest.CommonSetup):
    """ Common Setup section """

    # CommonSetup have subsection.
    # You can have 1 to as many subsection as wanted
    # here is an example of 2 subsections

    # First subsection
    @aetest.subsection
    def sample_subsection_1(self):
        """ Common Setup subsection """
        log.info("Aetest Common Setup ")

    # If you want to get the name of current section,
    # add section to the argument of the function.

    # Second subsection
    @aetest.subsection
    def sample_subsection_2(self, section):
        """ Common Setup subsection """
        log.info("Inside %s" % (section))

        # And how to access the class itself ?

        # self refers to the instance of that class, and remains consistent
        # throughout the execution of that container.
        log.info("Inside class %s" % (self.uid))

##################################################################
###                      TESTCASES SECTION                     ###
##################################################################

# This is how to create a testcase
# You can have 0 to as many testcase as wanted

# Testcase name : tc_one
class tc_one(aetest.Testcase):
    """ This is user Testcases section """

    # Testcases are divided into 3 sections
    # Setup, Test and Cleanup.
```

```
    # This is how to create a setup section
    @aetest.setup
    def prepare_testcase(self, section):
        """ Testcase Setup section """
        log.info("Preparing the test")
        log.info(section)

    # This is how to create a test section
    # You can have 0 to as many test section as wanted

    # First test section
    @ aetest.test
    def simple_test_1(self):
        """ Sample test section. Only print """
        log.info("First test section ")

    # Second test section
    @ aetest.test
    def simple_test_2(self):
        """ Sample test section. Only print """
        log.info("Second test section ")

    # This is how to create a cleanup section
    @aetest.cleanup
    def clean_testcase(self):
        """ Testcase cleanup section """
        log.info("Pass testcase cleanup")

# Testcase name : tc_two
class tc_two(aetest.Testcase):
    """ This is user Testcases section """

    @ aetest.test
    def simple_test_1(self):
        """ Sample test section. Only print """
        log.info("First test section ")
        self.failed('This is an intentional failure')

    # Second test section
    @ aetest.test
    def simple_test_2(self):
        """ Sample test section. Only print """
        log.info("Second test section ")

    # This is how to create a cleanup section
    @aetest.cleanup
    def clean_testcase(self):
        """ Testcase cleanup section """
        log.info("Pass testcase cleanup")

#####################################################################
####                        COMMON CLEANUP SECTION                 ###
#####################################################################
```

```
# This is how to create a CommonCleanup
# You can have 0 , or 1 CommonCleanup.
# CommonCleanup can be named whatever you want :)
class common_cleanup(aetest.CommonCleanup):
    """ Common Cleanup for Sample Test """

    # CommonCleanup follow exactly the same rule as CommonSetup regarding
    # subsection
    # You can have 1 to as many subsection as wanted
    # here is an example of 1 subsections

    @aetest.subsection
    def clean_everything(self):
        """ Common Cleanup Subsection """
        log.info("Aetest Common Cleanup ")

if __name__ == '__main__': # pragma: no cover
    aetest.main()
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Скрипт pyATS — это файл Python, в котором объявлены тесты pyATS. Он может быть запущен непосредственно как отдельный файл Python, генерируя вывод только в окно терминала. В качестве альтернативы, один или несколько скриптов pyATS могут быть скомпилированы в "задание" и запущены вместе как пакетный процесс с помощью модуля pyATS EasyPy. EasyPy обеспечивает параллельное выполнение нескольких сценариев, собирает журналы в одном месте и предоставляет центральную точку для внесения изменений в тестируемую топологию.

Используйте cat для отображения файла задания pyATS, **pyats_sample_job.py**. Обратите внимание на инструкции по запуску этого файла, выделенные ниже.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat
examples/basic/basic_example_job.py
# To run the job:
# pyats run job basic_example_job.py
# Description: This example shows the basic functionality of pyats
#              with few passing tests

import os
from pyats.easypy import run

# All run() must be inside a main function
def main():
    # Find the location of the script in relation to the job file
    test_path = os.path.dirname(os.path.abspath(__file__))
    testscript = os.path.join(test_path, 'basic_example_script.py')

    # Execute the testscript
    run(testscript=testscript)
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Шаг 5: Запустите pyATS вручную, чтобы вызвать обычный тест-кейс.

Используя задание pyATS и файлы сценария, запустите pyATS вручную, чтобы вызвать основной тестовый пример. Это позволит убедиться, что задание pyATS и файлы сценария работают правильно. Информация в выходных данных выходит за рамки данной лабораторной работы, однако вы заметите, что задание и сценарий выполнили все необходимые задачи.

Примечание: приведенный ниже вывод был усечен. Репозиторий Cisco Test Automation на GitHub может быть изменен, что включает задание pyATS и файлы скриптов. Ваш вывод может быть другим, но не должен фатально повлиять на результат. Например, в файл **basic_example_script.py** был добавлен преднамеренный сбой, который не вызывает никаких проблем. Это пример того, что репозитории динамичны.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ pyats run job
examples/basic/basic_example_job.py
2022-05-10T18:03:54: %EASYPY-INFO: Starting job run: basic_example_job
2022-05-10T18:03:54:          %EASYPY-INFO:          Runinfo          directory:
/home/devasc/.pyats/runinfo/basic_example_job.2022May10_18:03:53.899937
2022-05-10T18:03:54: %EASYPY-INFO: ---------------------------------------------
-----------------------------------
2022-05-10T18:03:57: %EASYPY-INFO: Starting task execution: Task-1
2022-05-10T18:03:57: %EASYPY-INFO:     test harness = pyats.aetest
2022-05-10T18:03:57: %EASYPY-INFO:     testscript   = /home/devasc/labs/devnet-
src/pyats/csr1kv/examples/basic/basic_example_script.py
2022-05-10T18:03:57: %AETEST-INFO: +---------------------------------------------
-----------------------------------+
2022-05-10T18:03:57: %AETEST-INFO: |                              Starting common
setup                              |
<вывод опущен>
-------+
2022-05-10T18:03:57: %SCRIPT-INFO: First test section
2022-05-10T18:03:57: %AETEST-ERROR: Failed reason: This is an intentional failure
2022-05-10T18:03:57:  %AETEST-INFO: The result of section simple_test_1 is =>
FAILED
2022-05-10T18:03:57: %AETEST-INFO: +---------------------------------------------
-----------------------------------+
2022-05-10T18:03:57: %AETEST-INFO: |                              Starting section
simple_test_2                        |
<вывод опущен>
-------+
2022-05-10T18:03:59: %EASYPY-INFO: |                                 Easypy Report
|
2022-05-10T18:03:59: %EASYPY-INFO: +---------------------------------------------
-----------------------------------+
<вывод опущен>
2022-05-10T18:03:59: %EASYPY-INFO: Overall Stats
2022-05-10T18:03:59: %EASYPY-INFO:     Passed      : 3
2022-05-10T18:03:59: %EASYPY-INFO:     Passx       : 0
2022-05-10T18:03:59: %EASYPY-INFO:     Failed      : 1
2022-05-10T18:03:59: %EASYPY-INFO:     Aborted     : 0
2022-05-10T18:03:59: %EASYPY-INFO:     Blocked     : 0
2022-05-10T18:03:59: %EASYPY-INFO:     Skipped     : 0
2022-05-10T18:03:59: %EASYPY-INFO:     Errored     : 0
2022-05-10T18:03:59: %EASYPY-INFO:
2022-05-10T18:03:59: %EASYPY-INFO:     TOTAL       : 4
2022-05-10T18:03:59: %EASYPY-INFO:
```

```
2022-05-10T18:03:59: %EASYPY-INFO: Success Rate   : 75.00 %
2022-05-10T18:03:59: %EASYPY-INFO:
2022-05-10T18:03:59: %EASYPY-INFO: +-------------------------------------------
-----------------------------------+
2022-05-10T18:03:59: %EASYPY-INFO: |                                Task Result
Summary                              |
2022-05-10T18:03:59: %EASYPY-INFO: +-------------------------------------------
-----------------------------------+
2022-05-10T18:03:59:  %EASYPY-INFO:  Task-1:  basic_example_script.common_setup
PASSED
2022-05-10T18:03:59:    %EASYPY-INFO:    Task-1:    basic_example_script.tc_one
PASSED
2022-05-10T18:03:59:    %EASYPY-INFO:    Task-1:    basic_example_script.tc_two
FAILED
2022-05-10T18:03:59:  %EASYPY-INFO:  Task-1:  basic_example_script.common_cleanup
PASSED
2022-05-10T18:03:59: %EASYPY-INFO:
2022-05-10T18:03:59: %EASYPY-INFO: +-------------------------------------------
-----------------------------------+
2022-05-10T18:03:59: %EASYPY-INFO: |                                Task Result
Details                              |
2022-05-10T18:03:59: %EASYPY-INFO: +-------------------------------------------
-----------------------------------+
2022-05-10T18:03:59: %EASYPY-INFO: Task-1: basic_example_script
2022-05-10T18:03:59:          %EASYPY-INFO:          |--          common_setup
PASSED
2022-05-10T18:03:59:    %EASYPY-INFO:    |            |--    sample_subsection_1
PASSED
2022-05-10T18:03:59:    %EASYPY-INFO:    |            `--    sample_subsection_2
PASSED
2022-05-10T18:03:59:            %EASYPY-INFO:            |--            tc_one
PASSED
2022-05-10T18:03:59:    %EASYPY-INFO:    |            |--    prepare_testcase
PASSED
2022-05-10T18:03:59:    %EASYPY-INFO:    |            |--    simple_test_1
PASSED
2022-05-10T18:03:59:    %EASYPY-INFO:    |            |--    simple_test_2
PASSED
2022-05-10T18:03:59:    %EASYPY-INFO:    |            `--    clean_testcase
PASSED
2022-05-10T18:03:59:            %EASYPY-INFO:            |--            tc_two
FAILED
2022-05-10T18:03:59:    %EASYPY-INFO:    |            |--    simple_test_1
FAILED
2022-05-10T18:03:59:    %EASYPY-INFO:    |            |--    simple_test_2
PASSED
2022-05-10T18:03:59:    %EASYPY-INFO:    |            `--    clean_testcase
PASSED
2022-05-10T18:03:59:          %EASYPY-INFO:          `--          common_cleanup
PASSED
2022-05-10T18:03:59:    %EASYPY-INFO:                `--    clean_everything
PASSED
2022-05-10T18:03:59: %EASYPY-INFO: Sending report email...
2022-05-10T18:03:59: %EASYPY-INFO: Missing SMTP server configuration, or failed
to reach/authenticate/send mail. Result notification email failed to send.
2022-05-10T18:03:59: %EASYPY-INFO: Done!

Pro Tip
-------
   Use the following command to view your logs locally:
```

```
        pyats logs view

(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Часть 4: Genie для анализа командного вывода IOS

В этой части вы будете использовать Genie для получения неструктурированного вывода IOS и его разбора в JSON.

Шаг 1: Создайте testbed YAML.

Инструменты pyATS и Genie используют файл YAML, чтобы знать, к каким устройствам подключаться и каковы соответствующие учетные данные. Этот файл известен как testbed. Genie включает встроенную функциональность для создания testbed для вас.

Введите команду genie --help, чтобы увидеть все доступные команды. Для получения дополнительной справки по любой команде используйте параметр <команда>, как показано ниже для команды create. Обратите внимание, что testbed является одним из параметров команды create.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ genie --help
Usage:
  genie <command> [options]

Commands:
    create              Create Testbed, parser, triggers, ...
    diff                Command to diff two snapshots saved to file or directory
    dnac              Command to learn DNAC features and save to file (Prototype)
    learn               Command to learn device features and save to file
    parse               Command to parse show commands
    run                    Run Genie triggers & verifications in pyATS runtime
environment
    shell                enter Python shell, loading a pyATS testbed file and/or
pickled data

General Options:
  -h, --help            Show help

Run 'genie <command> --help' for more information on a command.
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ genie create --
help
Usage:
  genie create <subcommand> [options]

Subcommands:
    parser              create a new Genie parser from template
    testbed             create a testbed file automatically
    trigger             create a new Genie trigger from template

General Options:
  -h, --help            Show help
  -v, --verbose         Give more output, additive up to 3 times.
  -q, --quiet           Give less output, additive up to 3 times, corresponding
to WARNING, ERROR,
                        and CRITICAL logging levels
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Чтобы создать testbed YAML, введите приведенную ниже команду. Параметр --output создаст файл testbed.yml в каталоге с именем yaml. Каталог будет создан автоматически. Параметр --encode-password закодирует пароли в YAML-файле. Параметр interactive означает, что вам будет задан ряд вопросов. Ответьте "нет" на первые три вопроса. Затем дайте следующие ответы, чтобы создать файл testbed.yaml.

- **Device hostname** — должно соответствовать имени хоста устройства, которое для данной лабораторной работы является CSR1kv.
- **IP адрес** — адрес должен соответствовать IPv4 адресу CSR1kv, который вы обнаружили ранее в этой лаборатории. Здесь показан 192.168.56.101.
- **Username** — локальное имя пользователя, используемое для ssh.
- **Default password** — локальный пароль, используемый для ssh.
- **Enable password** — Оставьте пустым. На маршрутизаторе не настроен привилегированный пароль.
- **Protocol** — SSH вместе с группой обмена ключами, ожидаемой маршрутизатором.
- **OS** — ОС на маршрутизаторе.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ genie create
testbed interactive --output yaml/testbed.yml --encode-password
Start creating Testbed yaml file ...
Do all of the devices have the same username? [y/n] n
Do all of the devices have the same default password? [y/n] n
Do all of the devices have the same enable password? [y/n] n

Device hostname: CSR1kv
   IP (ip, or ip:port): 192.168.56.101
   Username: cisco
Default Password (leave blank if you want to enter on demand):
cisco123!
Enable Password (leave blank if you want to enter on demand):
   Protocol (ssh, telnet, ...): ssh -o KexAlgorithms=diffie-hellman-
group14-sha1
   OS (iosxr, iosxe, ios, nxos, linux, ...): iosxe
More devices to add ? [y/n] n
Testbed file generated:
yaml/testbed.yml

(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

С помощью cat просмотрите файл testbed.yml в каталоге yaml. Обратите внимание на ваши записи в файле YAML. Ваш пароль SSH зашифрован, а пароль включения "попросит" пользователя ввести пароль, если он требуется.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat
yaml/testbed.yml
devices:
```

```
  CSR1kv:
    connections:
      cli:
        ip: 192.168.56.101
        protocol: ssh -o KexAlgorithms=diffie-hellman-group14-sha1
    credentials:
      default:
        password: '%ENC{w5PDosOUw5fDosKQwpbCmMKH}'
        username: cisco
      enable:
        password: '%ASK{}'
    os: iosxe
    type: iosxe
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Шаг 2: Используйте Genie для разбора вывода команды show ip interface brief в JSON.

Если вы еще не завершили лабораторную работу — установка виртуальной машины CSR1kv, вы зачем сюда поступали? Если вы уже завершили эту лабораторную работу, запустите ВМ CSR1kv сейчас.

На ВМ CSR1kv введите команду show ip interface brief из привилегированного режима exec. Ваш адрес может быть увеличен до другого адреса, отличного от 192.168.56.101. Запишите IPv4-адрес для вашей ВМ CSR1kv. Вы будете использовать его позже в лабораторной работе.

```
CSR1kv> en
CSR1kv# show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.56.101  YES DHCP   up                    up
CSR1kv#
```

Используя testbed YAML, вызовите Genie для разбора неструктурированного вывода команды show ip interface brief в структурированный JSON. Эта команда включает команду IOS для разбора (show ip interface brief), testbed YAML (testbed.yml) и указанное устройство в файле тестовой площадки (CSR1kv).

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ genie parse "show
ip interface brief" --testbed-file yaml/testbed.yml --devices CSR1kv
Enter enable password for device CSR1kv: <Enter>
2022-05-10T18:12:20: %UNICON-WARNING: Device 'CSR1kv' connection 'cli' does not
have IP and/or port specified, ignoring
Device 'CSR1kv' connection 'cli' does not have IP and/or port specified, ignoring
  0%|                                                                          |
0/1 [00:00<?, ?it/s]{
  "interface": {
    "GigabitEthernet1": {
      "interface_is_ok": "YES",
      "ip_address": "192.168.56.101",
      "method": "DHCP",
      "protocol": "up",
      "status": "up"
    }
  }
```

```
}
100%|
| 1/1 [00:00<00:00,  2.22it/s]

(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ^C
```

Шаг 3: Используйте Genie для разбора вывода команды show version в JSON.

Для другого примера разберите неструктурированный вывод команды show version в структурированный JSON.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ genie parse "show version" --testbed-file yaml/testbed.yml --devices CSR1kv
Enter enable password for device CSR1kv: <Enter>
2022-05-10T18:13:20: %UNICON-WARNING: Device 'CSR1kv' connection 'cli' does not have IP and/or port specified, ignoring
Device 'CSR1kv' connection 'cli' does not have IP and/or port specified, ignoring
  0%|                                                                                |
0/1 [00:00<?, ?it/s]{
  "version": {
    "chassis": "CSR1000V",
    "chassis_sn": "9K8P1OFYE3D",
    "compiled_by": "mcpre",
    "compiled_date": "Thu 30-Jan-20 18:48",
    "curr_config_register": "0x2102",
    "disks": {
      "bootflash:.": {
        "disk_size": "7774207",
        "type_of_disk": "virtual hard disk"
      },
      "webui:.": {
        "disk_size": "0",
        "type_of_disk": "WebUI ODM Files"
      }
    },
    "hostname": "CSR1kv",
    "image_id": "X86_64_LINUX_IOSD-UNIVERSALK9-M",
    "image_type": "production image",
    "last_reload_reason": "reload",
    "license_level": "ax",
    "license_type": "Default. No valid license found.",
    "main_mem": "2182252",
    "mem_size": {
      "non-volatile configuration": "32768",
      "physical": "3985032"
    },
    "next_reload_license_level": "ax",
    "number_of_intfs": {
      "Gigabit Ethernet": "1"
    },
    "os": "IOS-XE",
    "platform": "Virtual XE",
    "processor_type": "VXE",
    "returned_to_rom_by": "reload",
    "rom": "IOS-XE ROMMON",
```

```
    "rtr_type": "CSR1000V",
    "system_image": "bootflash:packages.conf",
    "uptime": "2 days, 6 hours, 26 minutes",
    "uptime_this_cp": "2 days, 6 hours, 27 minutes",
    "version": "16.9.5",
    "version_short": "16.9"
  }
}
100%|
| 1/1 [00:00<00:00,  2.06it/s]

(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Часть 5: Использование Genie для сравнения конфигураций

Как вы видели, Genie можно использовать для разбора команд show в структурированный json. Genie также можно использовать, для:

- Ежегодных снапшотов конфигураций и сравнивнения их между собой
- Автоматизированного тестового деплоя в виртуальной среде для проверки перед развертыванием в проде.
- Устранения неполадок в конфигурациях путем проведения сравнений между устройствами.

В частях 5 и 6 вы увидите, как проводить сравнение между двумя различными выходами.

Шаг 1: Добавьте IPv6-адрес к CSR1kv.

На виртуальной машине CSR1kv добавьте следующий адрес IPv6:

```
CSR1kv(config)# interface gig 1
CSR1kv(config-if)# ipv6 address 2001:db8:acad:56::101/64
```

Шаг 2: Используйте Genie для проверки конфигурации и разбора вывода в JSON.

Разберите неструктурированный вывод команды show ipv6 interface в структурированный JSON. Используйте параметр --output для отправки вывода в каталог verify-ipv6-1. Обратите внимание, что в выводе Genie сообщает, что было создано два файла.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ genie parse "show
ipv6 interface gig 1" --testbed-file yaml/testbed.yml --devices CSR1kv
--output verify-ipv6-1
Enter enable password for device CSR1kv: <Enter>
2022-05-10T18:14:25: %UNICON-WARNING: Device 'CSR1kv' connection 'cli' does not
have IP and/or port specified, ignoring
Device 'CSR1kv' connection 'cli' does not have IP and/or port specified, ignoring
100%|
| 1/1 [00:00<00:00,  2.08it/s]
+==============================================================================
+
| Genie Parse Summary for CSR1kv                                             |
+==============================================================================
+
|  Connected to CSR1kv                                                       |
|  -  Log: verify-ipv6-1/connection_CSR1kv.txt                               |
```

```
|-------------------------------------------------------------------------------
|
|  Parsed command 'show ipv6 interface gig 1'                                  |
|  -  Parsed structure: verify-ipv6-1/CSR1kv_show-ipv6-interface-              |
| gig-1_parsed.txt                                                             |
|  -  Device Console:   verify-ipv6-1/CSR1kv_show-ipv6-interface-              |
| gig-1_console.txt                                                            |
|-------------------------------------------------------------------------------
|
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Отобразите файлы, созданные Genie в каталоге verify-ipv6-1. Обратите внимание, что было создано два файла с одинаковыми именами, но один заканчивается на _console.txt, а другой на _parsed.txt. Имя каждого файла включает имя устройства и команду IOS, использованную в команде разбора Genie.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -l verify-ipv6-1
total 16
-rw-rw-rw- 1 devasc devasc 9094 connection_CSR1kv.txt
-rw-rw-r-- 1 devasc devasc  745 мая 10 18:10 CSR1kv_show-ipv6-interface-gig-1_console.txt
-rw-rw-r-- 1 devasc devasc  877 мая 10 18:10 CSR1kv_show-ipv6-interface-gig-1_parsed.txt
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

С помощью cat просмотрите содержимое файла _console.txt. Обратите внимание на глобальный одноадресный адрес IPv6, который вы настроили, и автоматический локальный адрес канала EUI-64.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat verify-ipv6-1/CSR1kv_show-ipv6-interface-gig-1_console.txt
+++ CSR1kv: executing command 'show ipv6 interface gig 1' +++
show ipv6 interface gig 1
GigabitEthernet1 is up, line protocol is up
  IPv6 is enabled, link-local address is FE80::A00:27FF:FE73:D79F
  No Virtual link-local address(es):
  Description: VBox
  Global unicast address(es):
    2001:DB8:ACAD:56::101, subnet is 2001:DB8:ACAD:56::/64
  Joined group address(es):
    FF02::1
    FF02::1:FF00:101
    FF02::1:FF73:D79F
  MTU is 1500 bytes
  ICMP error messages limited to one every 100 milliseconds
  ICMP redirects are enabled
  ICMP unreachables are sent
  ND DAD is enabled, number of DAD attempts: 1
  ND reachable time is 30000 milliseconds (using 30000)
  ND NS retransmit interval is 1000 milliseconds
CSR1kv#
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Используйте cat для изучения содержимого файла _parsed.txt. Это разобранный JSON-файл команды show ipv6 interface gig 1.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat verify-ipv6-1/CSR1kv_show-ipv6-interface-gig-1_parsed.txt
{
  "GigabitEthernet1": {
    "enabled": true,
    "ipv6": {
      "2001:DB8:ACAD:56::101/64": {
        "ip": "2001:DB8:ACAD:56::101",
        "prefix_length": "64",
        "status": "valid"
      },
      "FE80::A00:27FF:FE73:D79F": {
        "ip": "FE80::A00:27FF:FE73:D79F",
        "origin": "link_layer",
        "status": "valid"
      },
      "enabled": true,
      "icmp": {
        "error_messages_limited": 100,
        "redirects": true,
        "unreachables": "sent"
      },
      "nd": {
        "dad_attempts": 1,
        "dad_enabled": true,
        "ns_retransmit_interval": 1000,
        "reachable_time": 30000,
        "suppress": false,
        "using_time": 30000
      }
    },
    "joined_group_addresses": [
      "FF02::1",
      "FF02::1:FF00:101",
      "FF02::1:FF73:D79F"
    ],
    "mtu": 1500,
    "oper_status": "up"
  },
  "_exclude": []
}
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Шаг 3: Измените IPv6 Link-Local адрес.

На виртуальной машине CSR1kv добавьте следующий адрес IPv6:

```
CSR1kv> en
CSR1kv# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
CSR1kv(config)# interface gig 1
CSR1kv(config-if)# ipv6 address fe80::56:1 link-local
```

Шаг 4: Используйте Genie для проверки конфигурации и разбора вывода в JSON.

Разберите неструктурированный вывод команды show ipv6 interface в структурированный JSON. Используйте параметр --output для отправки вывода в другой каталог verify-ipv6-2. Вы можете использовать историю команд для вызова предыдущей команды (стрелка вверх). Убедитесь, что вы изменили 1 на 2, чтобы создать новый каталог verify-ipv6-2.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ genie parse "show
ipv6 interface gig 1" --testbed-file yaml/testbed.yml --devices CSR1kv
--output verify-ipv6-2
Enter enable password for device CSR1kv: <Enter>
2022-05-10T18:16:29: %UNICON-WARNING: Device 'CSR1kv' connection 'cli' does not
have IP and/or port specified, ignoring
Device 'CSR1kv' connection 'cli' does not have IP and/or port specified, ignoring
100%|
| 1/1 [00:00<00:00,  2.24it/s]
+==============================================================================
+
| Genie Parse Summary for CSR1kv                                             |
+==============================================================================
+
|  Connected to CSR1kv                                                       |
|  -  Log: verify-ipv6-2/connection_CSR1kv.txt                               |
|----------------------------------------------------------------------------
|
|  Parsed command 'show ipv6 interface gig 1'                                |
|  -  Parsed structure: verify-ipv6-2/CSR1kv_show-ipv6-interface-            |
| gig-1_parsed.txt                                                           |
|  -  Device Console:   verify-ipv6-2/CSR1kv_show-ipv6-interface-            |
| gig-1_console.txt                                                          |
|----------------------------------------------------------------------------
|

(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Отобразите файлы, созданные Genie в каталоге verify-ipv6-2. Они аналогичны двум файлам, которые вы создали перед изменением локального адреса канала IPv6.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -l verify-
ipv6-2
total 16
-rw-rw-rw- 1 devasc devasc 4536 мая 10 18:16 connection_CSR1kv.txt
-rw-rw-r-- 1 devasc devasc  728 мая 10 18:16 CSR1kv_show-ipv6-interface-gig-
1_console.txt
-rw-rw-r-- 1 devasc devasc  846 мая 10 18:16 CSR1kv_show-ipv6-interface-gig-
1_parsed.txt
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Используйте cat для изучения содержимого каждого файла. Изменения выделены в выводе ниже.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat verify-ipv6-
2/CSR1kv_show-ipv6-interface-gig-1_console.txt
+++ CSR1kv: executing command 'show ipv6 interface gig 1' +++
show ipv6 interface gig 1
```

```
GigabitEthernet1 is up, line protocol is up
  IPv6 is enabled, link-local address is FE80::56:1
  No Virtual link-local address(es):
  Description: VBox
  Global unicast address(es):
    2001:DB8:ACAD:56::101, subnet is 2001:DB8:ACAD:56::/64
  Joined group address(es):
    FF02::1
    FF02::1:FF00:101
    FF02::1:FF56:1
  MTU is 1500 bytes
  ICMP error messages limited to one every 100 milliseconds
  ICMP redirects are enabled
  ICMP unreachables are sent
  ND DAD is enabled, number of DAD attempts: 1
  ND reachable time is 30000 milliseconds (using 30000)
  ND NS retransmit interval is 1000 milliseconds
CSR1kv#
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat verify-ipv6-
2/CSR1kv_show-ipv6-interface-gig-1_parsed.txt
{
  "GigabitEthernet1": {
    "enabled": true,
    "ipv6": {
      "2001:DB8:ACAD:56::101/64": {
        "ip": "2001:DB8:ACAD:56::101",
        "prefix_length": "64",
        "status": "valid"
      },
      "FE80::56:1": {
        "ip": "FE80::56:1",
        "origin": "link_layer",
        "status": "valid"
      },
      "enabled": true,
      "icmp": {
        "error_messages_limited": 100,
        "redirects": true,
        "unreachables": "sent"
      },
      "nd": {
        "dad_attempts": 1,
        "dad_enabled": true,
        "ns_retransmit_interval": 1000,
        "reachable_time": 30000,
        "suppress": false,
        "using_time": 30000
      }
    },
    "joined_group_addresses": [
      "FF02::1",
      "FF02::1:FF00:101",
      "FF02::1:FF56:1"
    ],
```

```
    "mtu": 1500,
    "oper_status": "up"
  },
  "_exclude": []
}
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Шаг 5: Используйте Genie для сравнения разницы между конфигурациями.

В предыдущем шаге довольно легко найти изменение локального адреса канала IPv6. Но предположим, что вы ищете проблему в сложной конфигурации. Возможно, вы пытаетесь найти различия между конфигурацией OSPF на маршрутизаторе, который получает правильные маршруты, и на другом маршрутизаторе, который их не получает, и вы хотите увидеть разницу в их конфигурациях OSPF.Возможно, вы пытаетесь найти разницу в длинном списке ACL-запросов между двумя маршрутизаторами, которые должны иметь идентичные политики безопасности. Genie может выполнить сравнение за вас и облегчить поиск различий.

Используйте следующую команду, чтобы Genie нашел различия между двумя разобранными JSON-файлами. Обратите внимание, что в выводе указано, где вы можете найти сравнения Genie. В данном случае первое имя файла — это предыдущая конфигурация, а второе имя файла — текущая конфигурация.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ genie diff
verify-ipv6-1 verify-ipv6-2
1it [00:00, 579.32it/s]
+==============================================================================
+
| Genie Diff Summary between directories verify-ipv6-1/ and verify-ipv6-2/     |
+==============================================================================
+
|  File: CSR1kv_show-ipv6-interface-gig-1_parsed.txt                           |
|   - Diff can be found at ./diff_CSR1kv_show-ipv6-interface-gig-1_parsed.txt  |
|------------------------------------------------------------------------------
|
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Используйте cat для просмотра содержимого файла с различиями. Знак плюс "+" указывает на добавления, а знак минус "-" — на то, что было удалено.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat
./diff_CSR1kv_show-ipv6-interface-gig-1_parsed.txt
--- verify-ipv6-1/CSR1kv_show-ipv6-interface-gig-1_parsed.txt
+++ verify-ipv6-2/CSR1kv_show-ipv6-interface-gig-1_parsed.txt
 GigabitEthernet1:
  ipv6:
+  FE80::56:1:
+   ip: FE80::56:1
+   origin: link_layer
+   status: valid
-  FE80::A00:27FF:FE73:D79F:
-   ip: FE80::A00:27FF:FE73:D79F
-   origin: link_layer
-   status: valid
```

```
  joined_group_addresses:
-  index[2]: FF02::1:FF73:D79F
+  index[2]: FF02::1:FF56:1
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Часть 6: Завершение лабораторной и дальнейшие исследования

В этой части вы деактивируете Python venv и изучите другие варианты использования Genie.

Шаг 1: Деактивируйте виртуальную среду Python.

После завершения этой лабораторной работы вы можете деактивировать виртуальную среду Python с помощью команды deactivate. Заметьте, что перед prompt больше не стоит "(csr1kv)".

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ deactivate
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Шаг 2: Изучите больше примеров использования pyATS и Genie.

Ранее в этой лаборатории вы клонировали папку с примерами из репозитория Cisco Test Automation with pyATS and Genie на GitHub.

В этом репозитории GitHub есть много других примеров использования. Возможно, вы захотите изучить другие папки и различные другие примеры использования. Дополнительную информацию можно найти на следующих веб-сайтах:

- Поиск: "NetDevOps validation using Cisco pyATS | Genie for network engineers: no coding necessary".
- Cisco GitHub: https://github.com/CiscoTestAutomation

Лабораторная работа №8.3.5

Исследование YANG моделей

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC

Часть 2: Изучение YANG модели на GitHub

Часть 3: Изучение YANG модели с помощью pyang

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

Порядок выполнения работы

Часть 1: Запуск виртуальной машины DEVASC

Если вы еще не завершили **лабораторную работу - Установка лабораторной среды виртуальной машины**, сделайте это сейчас. Если вы уже завершили эту лабораторную работу, запустите виртуальную машину **DEVASC**.

Часть 2: Изучение YANG модели на GitHub

В этой части вы установите модуль pyang в вашу виртуальную машину DEVASC и изучите, как он преобразует файлы YANG. Pyang упрощает работу с файлами YANG. Модуль поставляется с исполняемым файлом командной строки pyang, который преобразует файлы YANG в более человекочитаемый формат.

Шаг 1: Изучите Cisco IOS XE YANG модели в репозитории GitHub.

Откройте Chromium и перейдите на страницу https://github.com/YangModels/yang.

В ветве master перейдите к моделям YANG для Cisco IOS XE версии 17.8.1, щелкнув следующие каталоги: vendor > cisco > xe > 1781

Щелкните ietf-interfaces.yang и прокрутите все узлы контейнера, узлы листа и узлы списка. Если вы знакомы с выводом команды IOS show interfaces, то вы должны узнать некоторые или все узлы. Например, около строки 221 вы увидите включенный лист.

```
leaf enabled {
  type boolean;
  default "true";
  description
    "This leaf contains the configured, desired state of the
     interface.
     Systems that implement the IF-MIB use the value of this
     leaf in the 'running' datastore to set
     IF-MIB.ifAdminStatus to 'up' or 'down' after an ifEntry
     has been initialized, as described in RFC 2863.
     Changes in this leaf in the 'running' datastore are
     reflected in ifAdminStatus, but if ifAdminStatus is
     changed over SNMP, this leaf is not affected.";
  reference
    "RFC 2863: The Interfaces Group MIB - ifAdminStatus";
}
```

Шаг 2: Скопируйте модель ietf-interfaces.yang в папку на вашей виртуальной машине.

Откройте код VS.

Нажмите File > Open Folder... и перейдите в каталог devnet-src.

Нажмите OK.

Откройте окно терминала в VS Code: Terminal > New Terminal.

Создайте подкаталог pyang в каталоге /devnet-src.

```
devasc@labvm:~/labs/devnet-src$ mkdir pyang
devasc@labvm:~/labs/devnet-src$
```

Вернитесь на вкладку Chromium, где модель ietf-interfaces.yang все еще открыта. При необходимости прокрутите страницу назад к вершине и нажмите Raw, чтобы отобразить только данные модели YANG.

Выберите и скопируйте URL-адрес.

В терминале перейдите в папку pyang.

С помощью wget сохраните необработанный файл ietf-interfaces.yang.

```
devasc@labvm:~/labs/devnet-src/pyang$ wget
https://raw.githubusercontent.com/YangModels/yang/master/vendor/cisco/x
e/1781/ietf-interfaces.yang
--2022-05-10                                                    19:28:07--
https://raw.githubusercontent.com/YangModels/yang/master/vendor/cisco/xe/1781/i
etf-interfaces.yang
Resolving        raw.githubusercontent.com        (raw.githubusercontent.com)...
185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting                    to                    raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 24248 (24K) [text/plain]
Saving to: 'ietf-interfaces.yang'

ietf-interfaces.ya 100%[===============>]  23,68K  --.-KB/s    in 0s

2022-05-10 19:28:07 (55,8 MB/s) - 'ietf-interfaces.yang' saved [24248/24248]

devasc@labvm:~/labs/devnet-src/pyang$
```

Теперь у вас есть локальная версия модели ietf-interfaces.yang, которой вы можете манипулировать с помощью pyang.

Часть 3: Изучение YANG модели с помощью pyang

В этой части вы установите модуль pyang в вашу виртуальную машину DEVASC и изучите, как он преобразует модель YANG, которую вы скопировали с GitHub. Pyang упрощает работу с файлами YANG. Модуль поставляется с исполняемым файлом командной строки pyang, который преобразует файлы YANG в более удобочитаемый формат.

Шаг 1: Убедитесь, что pyang установлен и обновлен.

В VS Code откройте окно терминала.

Проверьте, что pyang уже установлен, с помощью команды pyang -v. Номер вашей версии может отличаться от указанного здесь.

```
devasc@labvm:~/labs/devnet-src$ pyang -v
pyang 2.2.1
devasc@labvm:~/labs/devnet-src$
```

(Опционально) Вы можете проверить наличие последних обновлений pyang с помощью следующей команды pip3. Любые обновления после написания этой лабораторной работы будут загружены и установлены.

```
devasc@labvm:~/labs/devnet-src$ pip3 install pyang --upgrade
Collecting pyang
  Downloading pyang-2.5.3-py2.py3-none-any.whl (592 kB)
     |████████████████████████████████| 592 kB 1.1 MB/s
Requirement already satisfied, skipping upgrade: lxml in
/home/devasc/.local/lib/python3.8/site-packages (from pyang) (4.5.1)
Installing collected packages: pyang
  Attempting uninstall: pyang
    Found existing installation: pyang 2.2.1
    Uninstalling pyang-2.2.1:
      Successfully uninstalled pyang-2.2.1
Successfully installed pyang-2.5.3
```

```
devasc@labvm:~/labs/devnet-src/pyang$
```

Шаг 2: Преобразование модели ietf-interfaces.yang.

Перейдите в каталог pyang.

```
devasc@labvm:~/labs/devnet-src$ cd pyang
devasc@labvm:~/labs/devnet-src/pyang$
```

Введите pyang -h | more, чтобы изучить варианты преобразования модели YANG. Найдите опцию -f, как показано ниже. Вы будете использовать опцию форматирования дерева.

```
devasc@labvm:~/labs/devnet-src/pyang$ pyang -h | more
Usage: pyang [options] [<filename>...]

Validates the YANG module in <filename> (or stdin), and all its dependencies.

Options:
  -h, --help            Show this help message and exit
  -v, --version         Show version number and exit
<вывод опущен>
  -f FORMAT, --format=FORMAT
                        Convert to FORMAT.  Supported formats are: yang, yin,
                        dsdl, jstree, jsonxsl, capability, identifiers, jtox,
                        uml, name, omni, tree, depend, sample-xml-skeleton
<вывод опущен>
devasc@labvm:~/labs/devnet-src/pyang$
```

Преобразуйте модель ietf-interfaces.yang в формат дерева с помощью следующей команды. Обратите внимание, что включенный лист гораздо легче найти и прочитать в этом формате.

```
devasc@labvm:~/labs/devnet-src/pyang$ pyang -f tree ietf-
interfaces.yang
ietf-interfaces.yang:6: error: module "ietf-yang-types" not found in search path
module: ietf-interfaces
  +--rw interfaces
  |  +--rw interface* [name]
  |     +--rw name                        string
  |     +--rw description?                string
  |     +--rw type                        identityref
  |     +--rw enabled?                    boolean
  |     +--rw link-up-down-trap-enable?   enumeration {if-mib}?
  +--ro interfaces-state
     +--ro interface* [name]
        +--ro name               string
        +--ro type               identityref
        +--ro admin-status       enumeration {if-mib}?
        +--ro oper-status        enumeration
        +--ro last-change?       yang:date-and-time
        +--ro if-index           int32 {if-mib}?
        +--ro phys-address?      yang:phys-address
        +--ro higher-layer-if*   interface-state-ref
        +--ro lower-layer-if*    interface-state-ref
        +--ro speed?             yang:gauge64
        +--ro statistics
           +--ro discontinuity-time    yang:date-and-time
```

```
            +--ro in-octets?            yang:counter64
            +--ro in-unicast-pkts?      yang:counter64
            +--ro in-broadcast-pkts?    yang:counter64
            +--ro in-multicast-pkts?    yang:counter64
            +--ro in-discards?          yang:counter32
            +--ro in-errors?            yang:counter32
            +--ro in-unknown-protos?    yang:counter32
            +--ro out-octets?           yang:counter64
            +--ro out-unicast-pkts?     yang:counter64
            +--ro out-broadcast-pkts?   yang:counter64
            +--ro out-multicast-pkts?   yang:counter64
            +--ro out-discards?         yang:counter32
            +--ro out-errors?           yang:counter32
devasc@labvm:~/labs/devnet-src/pyang$
```

Лабораторная работа №8.3.6

Использование NETCONF для доступа к устройству IOS XE

Цель лабораторной работы:

Часть 1: Создание сети и проверка подключения

Часть 2: Использование сеанса NETCONF для сбора информации

Часть 3: Использование ncclient для подключения к NETCONF

Часть 4: Использование ncclient для получения конфигурации

Часть 5: Использование ncclient для настройки устройства

Часть 6: Задача: Модификация программы, использованной в этой лабораторной работе

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина
- CSR1kv виртуальная машина

Порядок выполнения работы

Часть 1: Запуск виртуальных машин и проверка подключения

В этой части вы запустите две виртуальные машины и проверите подключение. Затем вы установите соединение безопасной оболочки (SSH).

Шаг 1: Запуск виртуальных машин

Если вы еще не выполнили **лабораторную работу - Установка лабораторной среды виртуальной машины** и лабораторную работу **Установка виртуальной машины CSR1kv**, сделайте это сейчас. Запустите DEVASC и CSR1000v.

Шаг 2: Проверьте соединение между виртуальными машинами.

На CSR1kv нажмите Enter, чтобы получить командную строку, а затем используйте show ip interface brief, чтобы проверить, что адрес IPv4 равен 192.168.56.101. Если адрес отличается, запишите его.

Откройте терминал в VS Code в DEVASC.

Выполните Ping CSR1kv для проверки соединения. Вы уже должны были сделать это ранее в лабораторных работах по установке. Если вы не можете выполнить пинг, вернитесь к лабораторным работам, перечисленным выше в части 1a.

```
devasc@labvm:~$ ping -c 5 192.168.56.101
PING 192.168.56.101 (192.168.56.101) 56(84) bytes of data.
64 bytes from 192.168.56.101: icmp_seq=1 ttl=254 time=1.37 ms
64 bytes from 192.168.56.101: icmp_seq=2 ttl=254 time=1.15 ms
64 bytes from 192.168.56.101: icmp_seq=3 ttl=254 time=0.981 ms
64 bytes from 192.168.56.101: icmp_seq=4 ttl=254 time=1.01 ms
64 bytes from 192.168.56.101: icmp_seq=5 ttl=254 time=1.14 ms

--- 192.168.56.101 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 0.981/1.130/1.365/0.135 ms
devasc@labvm:~$
```

Шаг 3: Проверьте подключение SSH к CSR1kv.

В терминале DEVASC выполните SSH к виртуальной машине CSR1kv с помощью следующей команды:

```
devasc@labvm:~$ ssh  cisco@192.168.56.101
```

Примечание: При первом SSH к CSR1kv ваша виртуальная машина DEVASC предупредит вас о подлинности CSR1kv. Поскольку вы доверяете CSR1kv, ответьте "да" на запрос.

```
The authenticity of host '192.168.56.101 (192.168.56.101)' can't be
established.
RSA key fingerprint is
SHA256:HYv9K5Biw7PFiXeoCDO/LTqs3EfZKBuJdiPo34VXDUY.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
yes
Warning: Permanently added '192.168.56.101' (RSA) to the list of known
hosts.
```

Введите cisco123! в качестве пароля, и теперь вы должны находиться в привилегированной командной строке EXEC для CSR1kv.

```
Password: <cisco123!>

CSR1kv#
```

Оставьте сеанс SSH открытым для следующей части.

Часть 2: Использование сеанса NETCONF для сбора информации

В этой части вы убедитесь, что NETCONF запущен, включите NETCONF, если он не запущен, и убедитесь, что NETCONF готов к SSH соединению. Затем вы подключитесь к процессу NETCONF, запустите сеанс NETCONF, соберете информацию об интерфейсе и закроете сеанс.

Шаг 1: Проверьте, запущен ли NETCONF на CSR1kv.

В сеансе SSH с CSR1kv используйте команду **show platform software yang-management process**, чтобы проверить, запущен ли демон NETCONF SSH (ncsshd).

```
CSR1kv# show platform software yang-management process
confd            : Running
nesd             : Running
syncfd           : Running
ncsshd           : Running
dmiauthd         : Running
nginx            : Running
ndbmand          : Running
pubd             : Running

CSR1kv#
```

Если NETCONF не запущен, как показано в выводе выше, введите команду глобальной конфигурации netconf-yang.

```
CSR1kv# config t
CSR1kv (config)# netconf-yang
```

Введите **exit**, чтобы закрыть сеанс SSH.

Шаг 2: Получите доступ к процессу NETCONF через терминал SSH.

На этом шаге вы восстановите SSH-сессию с CSR1kv. Но на этот раз вы укажете порт NETCONF 830 и отправите netconf как команду подсистемы.

Примечание: Для получения дополнительной информации об этих опциях изучите страницы руководства по SSH (man ssh).

Введите следующую команду в окне терминала. Вы можете использовать стрелку вверх, чтобы вызвать последнюю команду SSH и просто добавить параметры -p и -s, как показано на рисунке. Затем введите cisco123! в качестве пароля.

```
devasc@labvm:~$ ssh cisco@192.168.56.101 -p 830 -s netconf
cisco@192.168.56.101's password:
```

CSR1kv ответит сообщением hello, содержащим более 400 строк с перечислением всех своих возможностей NETCONF. Конец сообщений NETCONF обозначается]]>]]>.

```
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<capabilities>
<capability>urn:ietf:params:netconf:base:1.0</capability>
<capability>urn:ietf:params:netconf:base:1.1</capability>
<capability>urn:ietf:params:netconf:capability:writable-
running:1.0</capability>
<capability>urn:ietf:params:netconf:capability:xpath:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
(вывод опущен)
      </capability>
</capabilities>
<session-id>20</session-id></hello>]]>]]>
```

Шаг 3: Запустите сеанс NETCONF, отправив сообщение hello с клиента.

Чтобы начать сеанс NETCONF, клиент должен отправить свое собственное сообщение hello. Сообщение hello должно включать версию базовых возможностей NETCONF, которую клиент хочет использовать.

Скопируйте и вставьте следующий XML-код в сеанс SSH. Обратите внимание, что конец сообщения приветствия клиента обозначен символом]]>]]>.

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <capabilities>
   <capability>urn:ietf:params:netconf:base:1.0</capability>
 </capabilities>
</hello>
]]>]]>
```

Переключитесь на ВМ CSR1kv и используйте команду show netconf-yang sessions, чтобы убедиться, что сеанс NETCONF был запущен. Если экран ВМ CSR1kv темный, нажмите Enter, чтобы разбудить его.

```
CSR1kv> en
CSRk1v# show netconf-yang sessions
R: Global-lock on running datastore
C: Global-lock on candidate datastore
S: Global-lock on startup datastore

Number of sessions : 1

session-id  transport    username             source-host           global-lock
--------------------------------------------------------------------------------
20          netconf-ssh  cisco                192.168.56.1          None

CSR1kv#
```

Шаг 4: Отправьте сообщения RPC на устройство IOS XE.

Во время сеанса SSH клиент NETCONF может использовать сообщения Remote Procedure Call (RPC) для отправки операций NETCONF на устройство IOS XE. В таблице перечислены некоторые из наиболее распространенных операций NETCONF.

Операция	Описание
<get>	Получение информации о текущей конфигурации и состоянии устройства
<get-config>	Извлечение всего или части указанного хранилища данных конфигурации
<edit-config>	Загружает всю или часть конфигурации в указанное хранилище данных конфигурации
<copy-config>	Замена всего хранилища данных конфигурации на другое
<delete-config>	Удаление хранилища данных конфигурации
<commit>	Копирование хранилища данных кандидата в действующее хранилище данных

<lock> / <unlock>	Блокировка или разблокировка всей системы хранения данных конфигурации
<close-session>	Graceful завершение сессии NETCONF
<kill-session>	Принудительное завершение сеанса NETCONF

Скопируйте и вставьте следующий XML-код RPC get message в терминальный сеанс SSH для получения информации об интерфейсах на R1.

```
<rpc message-id="103" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <get>
  <filter>
    <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"/>
   </filter>
 </get>
</rpc>
]]>]]>
```

Напомним, что XML не требует отступов или пробелов. Поэтому CSR1kv вернет длинную строку данных XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="103"><data><interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"><interface><name>GigabitEthernet1</name><description>VBox</description><type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">ianaift:ethernetCsmacd</type><enabled>true</enabled><ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"></ipv4><ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"></ipv6></interface></interfaces></data></rpc-reply>]]>]]>
```

Скопируйте XML, который был возвращен, но не включайте заключительные символы "]]>]]>". Эти символы не являются частью XML, возвращаемого маршрутизатором.

Найдите в Интернете запрос "prettify XML". Найдите подходящий сайт и используйте его инструмент для преобразования вашего XML в более читабельный формат, например, следующий:

```
<?xml version="1.0"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="103">
  <data>
    <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
      <interface>
        <name>GigabitEthernet1</name>
        <description>VBox</description>
        <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">ianaift:ethernetCsmacd</type>
        <enabled>true</enabled>
        <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"/>
        <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"/>
      </interface>
    </interfaces>
  </data>
</rpc-reply>
```

Шаг 5: Закройте сеанс NETCONF.

Чтобы закрыть сессию NETCONF, клиент должен отправить следующее сообщение RPC:

```
<rpc message-id="9999999"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <close-session />
</rpc>
```

Через несколько секунд вы вернетесь к приглашению терминала. Вернитесь к приглашению CSR1kv и покажите открытые сессии netconf. Вы увидите, что сессия была закрыта.

```
CSR1kv# show netconf-yang sessions
There are no active sessions

CSR1kv#
```

Часть 3: Использование ncclient для подключения к NETCONF

Работа с NETCONF не требует работы с необработанными сообщениями NETCONF RPC и XML. В этой части вы узнаете, как использовать модуль ncclient Python для простого взаимодействия с сетевыми устройствами с помощью NETCONF. Кроме того, вы узнаете, как определить, какие модели YANG поддерживаются устройством. Эта информация будет полезна при создании производственной системы автоматизации сети, которая требует, чтобы конкретные модели YANG поддерживались данным сетевым устройством.

Шаг 1: Убедитесь, что ncclient установлен и готов к работе.

В терминале DEVASC-VM введите команду pip3 list --format=columns, чтобы увидеть все установленные в настоящее время модули Python. Отправьте вывод в more. Ваш вывод может отличаться от приведенного ниже. Но вы должны увидеть ncclient в списке, как показано на рисунке. Если это не так, используйте команду pip3 install ncclient для его установки.

```
devasc@labvm:~$ pip3 list --format=columns | more
Package                Version
---------------------- -----------
ansible                2.9.6
apache-libcloud        2.8.0
appdirs                1.4.3
argcomplete            1.8.1
astroid                2.3.3
(вывод опущен)
ncclient               0.6.7
netaddr                0.7.19
netifaces              0.10.4
netmiko                3.1.0
ntlm-auth              1.1.0
oauthlib               3.1.0
(вывод опущен)
xmltodict              0.12.0
zipp                   1.0.0
devasc@labvm:~$
```

Шаг 2: Создайте сценарий для использования ncclient для подключения к службе NETCONF.

В VS Code, нажмите **File > Open Folder...** и перейдите в каталог devnet-src. Нажмите OK.

Откройте окно терминала в VS Code: **Terminal > New Terminal**.

Создайте подкаталог netconf в каталоге /devnet-src

```
devasc@labvm:~/labs/devnet-src$ mkdir netconf
devasc@labvm:~/labs/devnet-src$
```

На панели EXPLORER в разделе DEVNET-SRC щелкните правой кнопкой мыши каталог netconf и выберите New File.

Назовите файл ncclient-netconf.py.

В файле сценария импортируйте класс manager из модуля ncclient. Затем создайте переменную m для представления метода connect(). Метод connect() включает всю информацию, необходимую для подключения к службе NETCONF, запущенной на CSR1kv. Обратите внимание, что для NETCONF используется порт 830.

```
from ncclient import manager

m = manager.connect(
    host="192.168.56.101",
    port=830,
    username="cisco",
    password="cisco123!",
    hostkey_verify=False
    )
```

Если параметр hostkey_verify установлен в True, CSR1kv попросит вас проверить отпечаток SSH. В лабораторных условиях безопасно установить это значение в False, как мы и сделали здесь.

Сохраните и запустите программу, чтобы убедиться в отсутствии ошибок. Пока вы не увидите никакого вывода.

```
devasc@labvm:~/labs/devnet-src$ cd netconf/
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
devasc@labvm:~/labs/devnet-src/netconf$
```

Вы можете проверить, что CSR1kv принял запрос на сеанс NETCONF. Должно появиться сообщение syslog %DMI-5-AUTH_PASSED в ВМ CSR1kv. Если экран черный, нажмите Enter, чтобы разбудить маршрутизатор. Сообщение syslog можно увидеть над баннером.

Шаг 3: Добавьте функцию печати в сценарий, чтобы перечислить возможности NETCONF для CSR1kv.

Объект m, возвращаемый функцией manager.connect(), представляет собой удаленную сессию NETCONF. Как вы видели ранее, в каждой сессии NETCONF сервер сначала отправляет свои возможности, которые представляют собой список поддерживаемых моделей YANG в формате XML. В модуле ncclient полученный список возможностей хранится в списке m.server_capabilities.

Используйте цикл for и функцию print для отображения возможностей устройства:

```
print("#Поддерживаемые возможности (YANG модели):")
for capability in m.server_capabilities:
    print(capability)
```

Сохраните и запустите программу. На выходе вы получите тот же результат, что и при отправке сложного сообщения hello, но без открывающего и закрывающего XML-тега <capability> в каждой строке.

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
#Поддерживаемые возможности (YANG модели):
urn:ietf:params:netconf:base:1.0
urn:ietf:params:netconf:base:1.1
urn:ietf:params:netconf:capability:writable-running:1.0
urn:ietf:params:netconf:capability:xpath:1.0
<вывод опущен>
urn:ietf:params:xml:ns:netconf:base:1.0?module=ietf-netconf&revision=2011-06-01
urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults?module=ietf-netconf-
with-defaults&revision=2011-06-01

        urn:ietf:params:netconf:capability:notification:1.1

devasc@labvm:~/labs/devnet-src/netconf$
```

Часть 4: Использование ncclient для получения конфигурации

В этой части вы будете использовать NETCONF ncclient для получения конфигурации для CSR1kv, использовать модуль xml.dom.minidom для форматирования конфигурации и использовать фильтр с get_config() для получения части работающей конфигурации.

Шаг 1: Используйте функцию get_config() для получения текущей конфигурации для R1.

Если вы хотите пропустить вывод возможностей (400+ строк), закомментируйте блок операторов, которые выводят возможности, как показано ниже:

```
'''
print("#Поддерживаемые возможности (YANG модели):")
for capability in m.server_capabilities:
    print(capability)
'''
```

Вы можете использовать метод get_config() объекта сессии m NETCONF для получения конфигурации для CSR1kv. Метод get_config() ожидает строкового параметра source, который указывает исходное хранилище данных NETCONF. Для отображения результатов используйте функцию print. Единственным хранилищем данных NETCONF на CSR1kv в настоящее время является работающее хранилище данных. Вы можете проверить это с помощью команды show netconf-yang datastores.

```
netconf_reply = m.get_config(source="running")
print(netconf_reply)
```

Сохраните и запустите свою программу. Вывод будет содержать более 100 строк, поэтому IDLE может сжать их. Дважды щелкните на текстовом сообщении Squeezed в окне оболочки IDLE, чтобы развернуть вывод.

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"       message-
id="urn:uuid:3f31bedc-5671-47ca-9781-4d3d7aadae24"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"><data><native
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-
native"><version>16.9</version><boot-start-marker/><boot-end-
marker/><banner><motd><banner>
(вывод опущен)
devasc@labvm:~/labs/devnet-src/netconf$
```

Обратите внимание, что возвращаемый XML не отформатирован. Вы можете скопировать его на тот же сайт, который вы нашли ранее, чтобы отформатировать XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply                        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"                      message-
id="urn:uuid:3f31bedc-5671-47ca-9781-4d3d7aadae24">
  <data>
    <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
      <version>16.9</version>
      <boot-start-marker/>
      <boot-end-marker/>
      <banner>
        <motd>
          <banner>^C</banner>
        </motd>
      </banner>
      <service>
        <timestamps>
          <debug>
            <datetime>
              <msec/>
            </datetime>
          </debug>
          <log>
            <datetime>
              <msec/>
            </datetime>
          </log>
        </timestamps>
      </service>
      <platform>
        <console xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-platform">
          <output>virtual</output>
        </console>
      </platform>
      <hostname>CSR1kv</hostname>
(вывод опущен)
```

Шаг 2: Используйте Python для улучшения XML.

Python имеет встроенную поддержку для работы с XML-файлами. Модуль xml.dom.minidom можно использовать для украшения вывода с помощью функции toprettyxml().

В начале сценария добавьте утверждение для импорта модуля xml.dom.minidom.

```
import xml.dom.minidom
```

Замените простую функцию печати print(netconf_reply) версией, которая печатает оптимизированный вывод XML.

```
print(xml.dom.minidom.parseString(netconf_reply.xml).toprettyxml())
```

Сохраните и запустите свою программу. XML отображается в более читабельном формате.

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
<?xml version="1.0" ?>
<rpc-reply                      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"                    message-
id="urn:uuid:3a5f6abc-76b4-436d-9e9a-7758091c28b7">
        <data>
                <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
                        <version>16.9</version>
                        <boot-start-marker/>
                        <boot-end-marker/>
                        <banner>
                                <motd>
                                        <banner>^C</banner>
                                </motd>
                        </banner>
(вывод опущен)
        </data>
</rpc-reply>

devasc@labvm:~/labs/devnet-src/netconf$
```

Шаг 3: Используйте фильтр с get_config(), чтобы получить только определенную модель YANG.

Администратор сети может захотеть получить только часть текущей конфигурации устройства. NETCONF поддерживает возврат только тех данных, которые определены в параметре фильтра функции get_conf().

Создайте переменную под названием netconf_filter, которая извлекает только данные, определенные моделью Cisco IOS XE Native YANG.

```
netconf_filter = """
<filter>
    <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native" />
</filter>
"""
netconf_reply = m.get_config(source="running", filter=netconf_filter)
print(xml.dom.minidom.parseString(netconf_reply.xml).toprettyxml())
```

Сохраните и запустите свою программу. Начало вывода будет таким же, как показано ниже. Однако на этот раз отображается только элемент <native> XML. Ранее отображались все модели YANG, доступные на CSR1kv.

Фильтрация полученных данных для отображения только родного модуля YANG значительно уменьшает результат. Это происходит потому, что родной модуль YANG включает только подмножество всех моделей Cisco IOX XE YANG.

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
<?xml version="1.0" ?>
<rpc-reply                      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"                      message-
id="urn:uuid:4da5b736-1d33-47c3-8e3c-349414be0958">
        <data>
                <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
                        <version>16.9</version>
                        <boot-start-marker/>
                        <boot-end-marker/>
                        <banner>
                                <motd>
                                        <banner>^C</banner>
                                </motd>
                        </banner>
                        <service>
                                <timestamps>
                                        <debug>
                                                <datetime>
                                                        <msec/>
                                                </datetime>
                                        </debug>
                                        <log>
                                                <datetime>
                                                        <msec/>
                                                </datetime>
                                        </log>
                                </timestamps>
                        </service>
                        <platform>
                                <console  xmlns="http://cisco.com/ns/yang/Cisco-
IOS-XE-platform">
                                        <output>virtual</output>
                                </console>
                        </platform>
                        <hostname>CSR1kv</hostname>
(вывод опущен)
                </native>
        </data>
</rpc-reply>

devasc@labvm:~/labs/devnet-src/netconf$
```

Часть 5: Использование ncclient для настройки устройства

В этой части вы будете использовать ncclient для настройки CSR1kv с помощью метода edit_config() модуля manager.

Шаг 1: Используйте ncclient для редактирования имени хоста для CSR1kv.

Чтобы обновить существующий параметр в конфигурации для CSR1kv, вы можете извлечь местоположение параметра из полученной ранее конфигурации. Для этого шага вы установите переменную для изменения значения <hostname>.

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
<?xml version="1.0" ?>
<rpc-reply                      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"                    message-
id="urn:uuid:4da5b736-1d33-47c3-8e3c-349414be0958">
        <data>
                <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
(вывод опущен)
                        <hostname>CSR1kv</hostname>
(вывод опущен)
```

Ранее вы определили переменную <filter>. Чтобы изменить конфигурацию устройства, вы определите переменную <config>. Добавьте следующую переменную в сценарий ncclient_netconf.py. Вы можете использовать NEWHOSTNAME или любое другое имя хоста по вашему желанию.

```
netconf_hostname = """
<config>
  <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
     <hostname>NEWHOSTNAME</hostname>
  </native>
</config>
"""
```

Используйте функцию edit_config() объекта сессии m NETCONF для отправки конфигурации и сохранения результатов в переменной netconf_reply, чтобы их можно было распечатать. Параметры для функции edit_config() следующие:

- **target** — целевое хранилище данных NETCONF, которое должно быть обновлено
- **config** — модификация конфигурации, которая должна быть отправлена

```
netconf_reply = m.edit_config(target="running",
config=netconf_hostname)
```

Функция edit_config() возвращает ответное сообщение XML RPC с <ok/>, указывающее на то, что изменение было успешно применено. Повторите предыдущий оператор печати для отображения результатов.

```
print(xml.dom.minidom.parseString(netconf_reply.xml).toprettyxml())
```

Сохраните и запустите свою программу. Вы должны получить результат, аналогичный показанному ниже. Вы также можете проверить, что имя хоста изменилось, переключившись на виртуальную машину CSR1kv.

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
(вывод опущен)
<?xml version="1.0" ?>
```

```
<rpc-reply                      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"                    message-
id="urn:uuid:e304b225-7951-4029-afd5-59e8e7edbaa0">
        <ok/>
</rpc-reply>

devasc@labvm:~/labs/devnet-src/netconf$
```

Отредактируйте свой сценарий, чтобы изменить имя хоста на CSR1kv. Сохраните и запустите свою программу. Вы также можете просто закомментировать код из предыдущего шага, если хотите избежать повторного изменения имени хоста.

Шаг 2: Используйте ncclient для создания нового интерфейса loopback на R1.

Создайте новую переменную <config> для хранения конфигурации нового интерфейса loopback. Добавьте следующее в сценарий ncclient_netconf.py.

Примечание: Вы можете использовать любое описание по своему усмотрению. Однако используйте только буквенно-цифровые символы, иначе их придется экранировать с помощью обратной косой черты (\ ).

```
netconf_loopback = """
<config>
 <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <interface>
   <Loopback>
    <name>1</name>
    <description>My first NETCONF loopback</description>
    <ip>
     <address>
      <primary>
       <address>10.1.1.1</address>
       <mask>255.255.255.0</mask>
      </primary>
     </address>
    </ip>
   </Loopback>
  </interface>
 </native>
</config>
"""
```

Добавьте следующую функцию edit_config() в файл ncclient_netconf.py, чтобы отправить новую конфигурацию loopback на R1 и затем распечатать результаты.

```
netconf_reply = m.edit_config(target="running",
config=netconf_loopback)
print(xml.dom.minidom.parseString(netconf_reply.xml).toprettyxml())
```

Сохраните и запустите свою программу. Вы должны получить результат, похожий на следующий:

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
(вывод опущен)
```

```
<?xml version="1.0" ?>
<rpc-reply                          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"                          message-
id="urn:uuid:98437f47-7a93-4cac-9b9e-9bc8afc9dfa1">
        <ok/>
</rpc-reply>

devasc@labvm:~/labs/devnet-src/netconf$
```

На CSR1kv проверьте, что новый интерфейс loopback был создан.

```
CSR1kv>en
CSR1kv# show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.56.101  YES DHCP   up                    up
Loopback1              10.1.1.1        YES other  up                    up
CSR1kv# show run | section interface Loopback1
interface Loopback1
 description My first NETCONF loopback
 ip address 10.1.1.1 255.255.255.0
CSR1kv#
```

Шаг 3: Попытайтесь создать новый интерфейс loopback с тем же адресом IPv4.

Создайте новую переменную netconf_newloop. Она будет содержать конфигурацию, создающую новый интерфейс loopback 2, но с тем же IPv4-адресом, что и на loopback 1: 10.1.1.1 /24. В CLI маршрутизатора это приведет к ошибке из-за попытки присвоить дублирующий IP-адрес интерфейсу.

```
netconf_newloop = """
<config>
 <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <interface>
   <Loopback>
    <name>2</name>
    <description>My second NETCONF loopback</description>
    <ip>
     <address>
      <primary>
       <address>10.1.1.1</address>
       <mask>255.255.255.0</mask>
      </primary>
     </address>
    </ip>
   </Loopback>
  </interface>
 </native>
</config>
"""
```

Добавьте следующую функцию edit_config() в файл ncclient_netconf.py для отправки новой конфигурации loopback на CSR1kv. Оператор print для этого шага не нужен.

```
netconf_reply = m.edit_config(target="running", config=netconf_newloop)
```

Сохраните и запустите программу. Вы должны получить вывод об ошибке, подобный следующему, с сообщением RPCError Device refused one or more commands.

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
Traceback (most recent call last):
  File "ncclient-netconf.py", line 80, in <module>
    netconf_reply = m.edit_config(target="running", config=netconf_newloop)
  File    "/home/devasc/.local/lib/python3.8/site-packages/ncclient/manager.py",
line 231, in execute
    return cls(self._session,
  File                                  "/home/devasc/.local/lib/python3.8/site-
packages/ncclient/operations/edit.py", line 69, in request
    return self._request(node)
  File                                  "/home/devasc/.local/lib/python3.8/site-
packages/ncclient/operations/rpc.py", line 348, in _request
    raise self._reply.error
ncclient.operations.rpc.RPCError: inconsistent value: Device refused one or more
commands
devasc@labvm:~/labs/devnet-src/netconf$
```

NETCONF не применит ни одну из отправленных конфигураций, если одна или несколько команд будут отклонены. Чтобы проверить это, введите команду show ip interface brief на R1. Обратите внимание, что ваш новый интерфейс не был создан.

```
CSR1kv# show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.56.101  YES DHCP   up                    up
Loopback1              10.1.1.1        YES other  up                    up
```

Часть 6: Челлендж: Измените программу, использованную в этой лаборатории

Ниже приведена полная программа, созданная в этой лаборатории, без единого комментария кода, чтобы вы могли запустить сценарий без ошибок. Ваш сценарий может выглядеть иначе. Практикуйте свои навыки работы с Python, изменяя программу для отправки различных команд проверки и конфигурации.

```
from ncclient import manager
import xml.dom.minidom

m = manager.connect(
    host="192.168.56.101",
    port=830,
    username="cisco",
    password="cisco123!",
    hostkey_verify=False
    )

print("#Supported Capabilities (YANG models):")
for capability in m.server_capabilities:
    print(capability)
```

```
netconf_reply = m.get_config(source="running")
print(xml.dom.minidom.parseString(netconf_reply.xml).toprettyxml())

netconf_filter = """
<filter>
    <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native" />
</filter>
"""
netconf_reply = m.get_config(source="running", filter=netconf_filter)
print(xml.dom.minidom.parseString(netconf_reply.xml).toprettyxml())

netconf_hostname = """
<config>
  <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
     <hostname>CSR1kv</hostname>
  </native>
</config>
"""
netconf_reply = m.edit_config(target="running",
config=netconf_hostname)
print(xml.dom.minidom.parseString(netconf_reply.xml).toprettyxml())

netconf_loopback = """
<config>
 <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <interface>
   <Loopback>
    <name>1</name>
    <description>My NETCONF loopback</description>
    <ip>
     <address>
      <primary>
       <address>10.1.1.1</address>
       <mask>255.255.255.0</mask>
      </primary>
     </address>
    </ip>
   </Loopback>
  </interface>
 </native>
</config>
"""
netconf_reply = m.edit_config(target="running",
config=netconf_loopback)
print(xml.dom.minidom.parseString(netconf_reply.xml).toprettyxml())

netconf_newloop = """
<config>
 <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <interface>
```

```
    <Loopback>
     <name>2</name>
     <description>My second NETCONF loopback</description>
     <ip>
      <address>
       <primary>
        <address>10.1.1.1</address>
        <mask>255.255.255.0</mask>
       </primary>
      </address>
     </ip>
    </Loopback>
   </interface>
  </native>
</config>
"""
netconf_reply = m.edit_config(target="running", config=netconf_newloop)
```

Лабораторная работа №8.3.7

Использование RESTCONF для доступа к устройству IOS XE

Цель лабораторной работы:

Часть 1: Создание сети и проверка подключения

Часть 2: Настройка устройства IOS XE для доступа к RESTCONF

Часть 3: Запуск и настройка Postman

Часть 4: Postman для отправки GET-запросов

Часть 5: Postman для отправки PUT запроса

Часть 6: Python скрипт для отправки GET-запросов

Часть 7: Python скрипт для отправки PUT запроса

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина
- CSR1kv виртуальная машина

Порядок выполнения работы

Часть 1: Запуск виртуальных машин и проверка подключения

В этой части вы запустите две виртуальные машины и проверите подключение. Затем вы установите соединение безопасной оболочки (SSH).

Шаг 1: Запуск виртуальных машин

Если вы еще не выполнили **лабораторную работу - Установка лабораторной среды виртуальной машины** и лабораторную работу **Установка**

виртуальной машины CSR1kv, сделайте это сейчас. Запустите DEVASC и CSR1000v.

Шаг 2: Проверьте соединение между виртуальными машинами.

На CSR1kv нажмите Enter, чтобы получить командную строку, а затем используйте show ip interface brief, чтобы проверить, что адрес IPv4 равен 192.168.56.101. Если адрес отличается, запишите его.

Откройте терминал в VS Code в DEVASC.

Выполните Ping CSR1kv для проверки соединения. Вы уже должны были сделать это ранее в лабораторных работах по установке. Если вы не можете выполнить пинг, вернитесь к лабораторным работам, перечисленным выше в части 1a.

```
devasc@labvm:~$ ping -c 5 192.168.56.101
PING 192.168.56.101 (192.168.56.101) 56(84) bytes of data.
64 bytes from 192.168.56.101: icmp_seq=1 ttl=254 time=1.37 ms
64 bytes from 192.168.56.101: icmp_seq=2 ttl=254 time=1.15 ms
64 bytes from 192.168.56.101: icmp_seq=3 ttl=254 time=0.981 ms
64 bytes from 192.168.56.101: icmp_seq=4 ttl=254 time=1.01 ms
64 bytes from 192.168.56.101: icmp_seq=5 ttl=254 time=1.14 ms

--- 192.168.56.101 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 0.981/1.130/1.365/0.135 ms
devasc@labvm:~$
```

Шаг 3: Проверьте подключение SSH к CSR1kv.

В терминале DEVASC выполните SSH к виртуальной машине CSR1kv с помощью следующей команды:

```
devasc@labvm:~$ ssh  cisco@192.168.56.101
```

Примечание: При первом SSH к CSR1kv ваша виртуальная машина DEVASC предупредит вас о подлинности CSR1kv. Поскольку вы доверяете CSR1kv, ответьте "да" на запрос.

```
The authenticity of host '192.168.56.101 (192.168.56.101)' can't be
established.
RSA key fingerprint is
SHA256:HYv9K5Biw7PFiXeoCDO/LTqs3EfZKBuJdiPo34VXDUY.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
yes
Warning: Permanently added '192.168.56.101' (RSA) to the list of known
hosts.
```

Введите cisco123! в качестве пароля, и теперь вы должны находиться в привилегированной командной строке EXEC для CSR1kv.

```
Password: <cisco123!>

CSR1kv#
```

Оставьте сеанс SSH открытым для следующей части.

Часть 2: Настройка устройства IOS XE для доступа к RESTCONF

В этой части вы настроите виртуальную машину CSR1kv на прием сообщений RESTCONF. Вы также запустите службу HTTPS.

Примечание: Службы, обсуждаемые в этой части, могут быть уже запущены на вашей виртуальной машине. Однако убедитесь, что вы знаете команды для просмотра запущенных служб и их включения.

Шаг 1: Убедитесь, что демоны RESTCONF запущены.

RESTCONF уже должен быть запущен. В терминале вы можете использовать команду show platform software yang-management process, чтобы увидеть, запущены ли все демоны, связанные со службой RESTCONF. Демон NETCONF также может быть запущен, но он не будет использоваться в этой лабораторной работе. Если один или несколько необходимых демонов не запущены, перейдите к шагу 2.

```
CSR1kv# show platform software yang-management process
confd            : Running
nesd             : Running
syncfd           : Running
ncsshd           : Running
dmiauthd         : Running
nginx            : Running
ndbmand          : Running
pubd             : Running

CSR1kv#
```

Шаг 2: Включите и проверьте службу RESTCONF.

Введите команду глобальной конфигурации restconf, чтобы включить службу RESTCONF на CSR1kv.

```
CSR1kv#configure terminal
CSR1kv(config)# restconf
```

Убедитесь, что необходимые демоны RESTCONF запущены. Напомним, что ncsshd — это служба NETCONF, которая может быть запущена на вашем устройстве. Для этой лабораторной работы она нам не нужна. Однако вам нужен nginx, который является HTTPS-сервером. Это позволит выполнять вызовы REST API к службе RESTCONF.

```
CSR1kv(config)# exit
CSR1kv# show platform software yang-management process
confd            : Running
nesd             : Running
syncfd           : Running
ncsshd           : Not Running
dmiauthd         : Running
nginx            : Not Running
ndbmand          : Running
pubd             : Running
```

Шаг 3: Включите и проверьте службу HTTPS.

Введите следующие команды глобальной конфигурации, чтобы включить сервер HTTPS и указать, что аутентификация сервера должна использовать локальную базу данных.

```
CSR1kv# configure terminal
CSR1kv(config)# ip http secure-server
CSR1kv(config)# ip http authentication local
```

Убедитесь, что сервер HTTPS (nginx) запущен.

```
CSR1kv(config)# exit
CSR1kv# show platform software yang-management process
confd            : Running
nesd             : Running
syncfd           : Running
ncsshd           : Not Running
dmiauthd         : Running
nginx            : Running
ndbmand          : Running
pubd             : Running
```

Часть 3: Запуск и настройка Postman

В этой части вы откроете Postman, отключите SSL-сертификаты и изучите пользовательский интерфейс.

Шаг 1: Откройте Postman.

В виртуальной машине DEVASC откройте приложение Postman.

Шаг 2: Отключите проверку сертификации SSL.

По умолчанию в Postman включена проверка SSL-сертификатов. Вы не будете использовать SSL сертификаты с CSR1kv, поэтому вам нужно отключить эту функцию.

Нажмите **File > Settings.**

На вкладке **General** в поле **SSL certificate verification** установите значение **OFF**.

Закройте диалоговое окно **Settings**.

Часть 4: Postman для отправки GET-запросов

В этой части вы будете использовать Postman для отправки GET запроса на CSR1kv, чтобы проверить, что вы можете подключиться к службе RESTCONF.

Шаг 1: Изучите пользовательский интерфейс Postman.

В центре вы увидите Launchpad. При желании вы можете исследовать эту область.

Щелкните знак плюс (+) рядом с вкладкой Launchpad, чтобы открыть запрос GET Untitled Request. В этом интерфейсе вы будете выполнять всю свою работу в этой лаборатории.

Шаг 2: Введите URL-адрес для CSR1kv.

Тип запроса уже установлен на GET. Оставьте тип запроса установленным на GET.

В поле "Enter request URL" введите URL, который будет использоваться для доступа к службе RESTCONF, запущенной на CSR1kv:

```
https://192.168.56.101/restconf/
```

Шаг 3: Введите учетные данные для аутентификации.

Под полем URL находятся вкладки Params, Authorization, Headers, Body, Pre-request Script, Test и Settings. В этой лабораторной работе вы будете использовать авторизацию, заголовки и тело.

Перейдите на вкладку Authorization.

В разделе Type нажмите стрелку вниз рядом с "Inherit auth from parent" и выберите Basic Auth.

В поле Username и Password введите локальные учетные данные аутентификации для CSR1kv:

Имя пользователя: cisco

Пароль: cisco123!

Щелкните Headers. Затем щелкните 7 hidden. Вы можете проверить, что ключ авторизации имеет значение Basic, которое будет использоваться для аутентификации запроса, когда он будет отправлен на CSR1kv.

Шаг 4: Установите JSON в качестве типа данных для отправки и получения от CSR1kv.

Вы можете отправлять и получать данные от CSR1kv в формате XML или JSON. Для этой лабораторной работы вы будете использовать JSON.

В области Headers щелкните в первом пустом поле Key и введите Content-Type для типа ключа. В поле Value введите application/yang-data+json. Это указывает Postman на отправку данных JSON на CSR1kv.

Ниже ключа Content-Type добавьте ещё одну пару ключ/значение. Поле Key будет Accept, а поле Value - application/yang-data+json.

Примечание: При необходимости вы можете изменить application/yang-data+json на application/yang-data+xml для отправки и получения XML-данных вместо JSON-данных.

Шаг 5: Отправьте запрос API на CSR1kv.

Теперь у Postman есть вся информация, необходимая для отправки GET-запроса. Нажмите кнопку **Send**. Ниже **Temporary Headers** вы должны увидеть следующий JSON-ответ от CSR1kv. Если нет, проверьте, что вы выполнили предыдущие шаги в этой части лабораторной работы и правильно настроили RESTCONF и службу HTTPS в части 2.

```
{
    "ietf-restconf:restconf": {
        "data": {},
        "operations": {},
        "yang-library-version": "2016-06-21"
    }
}
```

Этот ответ JSON подтверждает, что Postman теперь может отправлять другие запросы REST API на CSR1kv.

Шаг 6: Используйте запрос GET для сбора информации для всех интерфейсов на CSR1kv.

Теперь, когда у вас есть успешный GET-запрос, вы можете использовать его в качестве шаблона для дополнительных запросов. В верхней части Postman, рядом с вкладкой Launchpad, щелкните правой кнопкой мыши вкладку GET, которую вы только что использовали, и выберите Duplicate Tab.

Используйте YANG-модель ietf-interfaces для сбора информации об интерфейсах. Для URL добавьте data/ietf-interfaces:interfaces:

```
https://192.168.56.101/restconf/data/ietf-interfaces:interfaces
```

Нажмите кнопку Send. Вы должны увидеть ответ JSON от CSR1kv, похожий на показанный ниже. Вывод может отличаться в зависимости от конкретного маршрутизатора.


```
{
  "ietf-interfaces:interfaces": {
    "interface": [
      {
        "name": "GigabitEthernet1",
        "description": "VBox",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": true,
        "ietf-ip:ipv4": {},
        "ietf-ip:ipv6": {}
      }
    ]
  }
}
```


Шаг 7: Используйте запрос GET для сбора информации для определенного интерфейса на CSR1kv.

В этой лабораторной настроен только интерфейс GigabitEthernet1. Чтобы указать только этот интерфейс, расширьте URL, чтобы запрашивать информацию только для этого интерфейса.

Продублируйте последний запрос GET.

Добавьте параметр interface= для указания интерфейса и введите имя интерфейса.

```
https://192.168.56.101/restconf/data/ietf-interfaces:interfaces/interface=GigabitEthernet1
```

Примечание: Если вы запрашиваете информацию об интерфейсе с другого устройства Cisco с именами, использующими прямые косые черты, например GigabitEthernet0/0/1, используйте HTML-код %2F для прямых косых черт в имени интерфейса. Таким образом, 0/0/1 становится 0%2F0%2F1.

Нажмите кнопку Send. Вы должны увидеть ответ в формате JSON от CSR1kv, похожий на приведенный ниже. Вывод может отличаться в зависимости от вашего конкретного маршрутизатора. При настройке CSR1kv по умолчанию вы не увидите информацию об IP-адресации.

```
{
  "ietf-interfaces:interface": {
    "name": "GigabitEthernet1",
    "description": "VBox",
    "type": "iana-if-type:ethernetCsmacd",
    "enabled": true,
    "ietf-ip:ipv4": {},
    "ietf-ip:ipv6": {}
  }
}
```

Этот интерфейс получает адресацию от шаблона Virtual Box. Поэтому IPv4-адрес не отображается в show running-config. Вместо этого вы увидите команду ip address dhcp. Вы можете увидеть это также в кратком выводе show ip interface.

```
CSR1kv# show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.56.101  YES DHCP   up                    up
CSR1kv#
```

В следующей части вам нужно будет использовать ответ JSON от вручную настроенного интерфейса. Откройте командный терминал с CSR1kv и вручную настройте интерфейс GigabitEthernet1 с тем же адресом IPv4, который в настоящее время назначен ему Virtual Box.

```
CSR1kv# conf t
CSR1kv(config)# interface g1
CSR1kv(config-if)# ip address 192.168.56.101 255.255.255.0
CSR1kv(config-if)# end
CSR1kv# show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.56.101  YES manual up                    up
CSR1kv#
```

Вернитесь в Postman и снова отправьте GET-запрос. Теперь вы должны увидеть информацию об адресации IPv4 в ответе JSON, как показано ниже. В следующей части вы скопируете этот формат JSON для создания нового интерфейса.

```
{
  "ietf-interfaces:interface": {
    "name": "GigabitEthernet1",
    "description": "VBox",
    "type": "iana-if-type:ethernetCsmacd",
    "enabled": true,
    "ietf-ip:ipv4": {
      "address": [
        {
          "ip": "192.168.56.101",
          "netmask": "255.255.255.0"
        }
      ]
    },
    "ietf-ip:ipv6": {}
  }
```

```
}
```

Часть 5: Postman для отправки запроса PUT

В этой части вы настроите Postman на отправку PUT запроса на CSR1kv для создания нового loopback интерфейса.

Шаг 1: Повторите и измените последний GET-запрос.

Продублируйте последний GET-запрос.

Для параметра Type запроса нажмите стрелку вниз рядом с GET и выберите PUT.

Для параметра interface= измените его на =Loopback1, чтобы указать новый интерфейс.

```
https://192.168.56.101/restconf/data/ietf-interfaces:interfaces/interface=Loopback1
```

Шаг 2: Настройте тело запроса, указав информацию для новой петли.

Чтобы отправить запрос PUT, необходимо предоставить информацию для тела запроса. Рядом с вкладкой Heders нажмите кнопку Body. Затем нажмите радиокнопку Raw. В настоящее время поле пустое. Если вы нажмете Send now, вы получите код ошибки 400 Bad Request, поскольку Loopback1 еще не существует, и вы не предоставили достаточно информации для создания интерфейса.

Заполните раздел Body необходимыми данными JSON для создания нового интерфейса Loopback1. Вы можете скопировать раздел Body предыдущего запроса GET и изменить его. Или вы можете скопировать следующее в раздел Body запроса PUT. Обратите внимание, что тип интерфейса должен быть установлен на softwareLoopback.

```
{
  "ietf-interfaces:interface": {
    "name": "Loopback1",
    "description": "My first RESTCONF loopback",
    "type": "iana-if-type:softwareLoopback",
    "enabled": true,
    "ietf-ip:ipv4": {
      "address": [
        {
          "ip": "10.1.1.1",
          "netmask": "255.255.255.0"
        }
      ]
    },
    "ietf-ip:ipv6": {}
  }
}
```

Нажмите кнопку Send, чтобы отправить запрос PUT на CSR1kv. Ниже раздела Body вы должны увидеть код ответа HTTP Status: 201 Created. Это означает, что ресурс был создан успешно.

Вы можете убедиться, что интерфейс был создан. Вернитесь в сеанс SSH с CSR1kv и введите show ip interface brief. Вы также можете запустить вкладку

Postman, содержащую запрос на получение информации об интерфейсах на CSR1kv, который был создан в предыдущей части этой лабораторной работы.

```
CSR1kv# show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.56.101  YES manual up                    up
Loopback1              10.1.1.1        YES other  up                    up
CSR1kv#
```

Часть 6: Python скрипт для отправки GET-запросов

В этой части вы создадите сценарий Python для отправки GET-запросов на CSR1kv.

Шаг 1: Создайте каталог RESTCONF и запустите сценарий.

Откройте VS Code. Затем нажмитеFile > Open Folder... и перейдите в каталог devnet-src. Нажмите OK.

Откройте окно терминала в VS Code: Terminal > New Terminal.

Создайте подкаталог restconf в каталоге /devnet-src.

```
devasc@labvm:~/labs/devnet-src$ mkdir restconf
devasc@labvm:~/labs/devnet-src$
```

В панели EXPLORER в разделе DEVNET-SRC щелкните правой кнопкой мыши каталог restconf и выберите New File.

Назовите файл restconf-get.py.

Введите следующие команды, чтобы импортировать необходимые модули и отключить предупреждения о сертификатах SSL:

```
import json
import requests
requests.packages.urllib3.disable_warnings()
```

Модуль json включает методы для преобразования данных JSON в объекты Python и наоборот. Модуль requests содержит методы, позволяющие отправлять REST-запросы на URL.

Шаг 2: Создайте переменные, которые будут компонентами запроса.

Создайте переменную api_url и присвойте ей URL, по которому будет осуществляться доступ к информации об интерфейсе на CSR1kv.

```
api_url = "https://192.168.56.101/restconf/data/ietf-interfaces:interfaces"
```

Создайте словарь headers, содержащий ключи Accept и Content-type, и присвойте ключам значение application/yang-data+json.

```
headers = { "Accept": "application/yang-data+json",
            "Content-type":"application/yang-data+json"
           }
```

Создайте кортежную переменную Python с именем basicauth, которая содержит два ключа, необходимых для аутентификации, имя пользователя и пароль.

Шаг 3: Создайте переменную для отправки запроса и хранения ответа в формате JSON.

Используйте переменные, созданные на предыдущем шаге, в качестве параметров для метода requests.get(). Этот метод отправляет запрос HTTP GET к API RESTCONF на CSR1kv. Присвойте результат запроса переменной с именем resp. Эта переменная будет содержать JSON-ответ от API. Если запрос прошел успешно, JSON будет содержать возвращенную модель данных YANG.

Введите следующее утверждение:

```
resp = requests.get(api_url, auth=basicauth, headers=headers, verify=False)
```

В таблице ниже перечислены различные элементы этого заявления:

Элемент	Описание
`resp`	Переменная для хранения ответа от API
`requests.get()`	Метод, который фактически выполняет GET-запрос
`api_url`	Переменная, в которой хранится строка адреса URL
`auth`	Кортежная переменная, созданная для хранения информации об аутентификации
`headers=headers`	Параметр, которому присваивается переменная headers
`verify=False`	Отключает проверку SSL-сертификата при выполнении запроса

Чтобы увидеть код ответа HTTP, добавьте print().

```
print(resp)
```

Сохраните и запустите свой скрипт. Вы должны получить результат, показанный ниже. Если нет, проверьте все предыдущие шаги в этой части, а также конфигурацию SSH и RESTCONF для CSR1kv.

```
devasc@labvm:~/labs/devnet-src$ cd restconf/
devasc@labvm:~/labs/devnet-src/restconf$ python3 restconf-get.py
<Response [200]>
devasc@labvm:~/labs/devnet-src/restconf$
```

Шаг 4: Форматирование и отображение данных JSON, полученных от CSR1kv.

Теперь вы можете извлечь значения ответа модели YANG из JSON ответа.

JSON ответа не совместим с объектами словаря и списка Python, поэтому его необходимо преобразовать в формат Python. Создайте новую переменную с именем response_json и присвойте ей переменную resp. Добавьте метод json() для преобразования JSON. Утверждение выглядит следующим образом:

```
response_json = resp.json()
```

Добавьте print() для отображения данных JSON.

```
print(response_json)
```

Сохраните и запустите свой сценарий. Вы должны получить результат, подобный следующему:

```
devasc@labvm:~/labs/devnet-src/restconf$ python3 restconf-get.py
<Response [200]>
{'ietf-interfaces:interfaces': {'interface': [{'name': 'GigabitEthernet1',
'description': 'VBox', 'type': 'iana-if-type:ethernetCsmacd', 'enabled': True,
```

```
'ietf-ip:ipv4':     {'address':     [{'ip':     '192.168.56.101',     'netmask':
'255.255.255.0'}]}, 'ietf-ip:ipv6': {}}, {'name': 'Loopback1', 'description': 'My
first RESTCONF loopback', 'type': 'iana-if-type:softwareLoopback', 'enabled':
True,    'ietf-ip:ipv4':    {'address':    [{'ip':    '10.1.1.1',    'netmask':
'255.255.255.0'}]}, 'ietf-ip:ipv6': {}}]}}
devasc@labvm:~/labs/devnet-src/restconf$
```

Чтобы сделать вывод более красивым, измените оператор print, чтобы использовать функцию json.dumps() с параметром "indent":

```
print(json.dumps(response_json, indent=4))
```

Сохраните и запустите свой скрипт. Вы должны получить результат, показанный ниже. Этот результат практически идентичен результату вашего первого GET-запроса Postman.

```
devasc@labvm:~/labs/devnet-src/restconf$ python3 restconf-get.py
<Response [200]>
{
    "ietf-interfaces:interfaces": {
        "interface": [
            {
                "name": "GigabitEthernet1",
                "description": "VBox",
                "type": "iana-if-type:ethernetCsmacd",
                "enabled": true,
                "ietf-ip:ipv4": {
                    "address": [
                        {
                            "ip": "192.168.56.101",
                            "netmask": "255.255.255.0"
                        }
                    ]
                },
                "ietf-ip:ipv6": {}
            },
            {
                "name": "Loopback1",
                "description": "My first RESTCONF loopback",
                "type": "iana-if-type:softwareLoopback",
                "enabled": true,
                "ietf-ip:ipv4": {
                    "address": [
                        {
                            "ip": "10.1.1.1",
                            "netmask": "255.255.255.0"
                        }
                    ]
                },
                "ietf-ip:ipv6": {}
            }
        ]
    }
}
devasc@labvm:~/labs/devnet-src/restconf$
```

Часть 7: Python скрипт для отправки запроса PUT

В этой части вы создадите Python скрипт для отправки запроса PUT на CSR1kv. Как и в Postman, вы создадите новый loopback интерфейс.

Шаг 1: Импортируйте модули и отключите предупреждения SSL.

На панели EXPLORER в разделе DEVNET-SRC щелкните правой кнопкой мыши каталог restconf и выберите New File.

Назовите файл restconf-put.py.

Введите следующие команды, чтобы импортировать необходимые модули и отключить предупреждения о сертификатах SSL:

```
import json
import requests
requests.packages.urllib3.disable_warnings()
```

Шаг 2: Создайте переменные, которые будут компонентами запроса.

Создайте переменную api_url и присвойте ей URL, нацеленный на новый интерфейс Loopback2.

Примечание: Эта спецификация переменной должна находиться на одной строке в вашем сценарии.

```
api_url = "https://192.168.56.101/restconf/data/ietf-interfaces:interfaces/interface=Loopback2"
```

Создайте словарную переменную headers, содержащую ключи Accept и Content-type, и присвойте ключам значение application/yang-data+json.

```
headers = { "Accept": "application/yang-data+json",
            "Content-type":"application/yang-data+json"
           }
```

Создайте кортежную переменную Python с именем basicauth, которая содержит два значения, необходимые для аутентификации, - имя пользователя и пароль.

```
basicauth = ("cisco", "cisco123!")
```

Создайте в Python словарную переменную yangConfig, которая будет содержать данные YANG, необходимые для создания нового интерфейса Loopback2. Вы можете использовать тот же словарь, который ранее использовали в Postman. Однако измените номер и адрес интерфейса. Также помните, что в Python булевы значения должны быть написаны с заглавной буквы. Поэтому убедитесь, что в паре ключ/значение для "enabled" буква T написана с заглавной буквы: True.

```
yangConfig = {
    "ietf-interfaces:interface": {
        "name": "Loopback2",
        "description": "My second RESTCONF loopback",
        "type": "iana-if-type:softwareLoopback",
        "enabled": True,
        "ietf-ip:ipv4": {
            "address": [
                {
                    "ip": "10.2.1.1",
                    "netmask": "255.255.255.0"
                }
```

```
                ]
            },
            "ietf-ip:ipv6": {}
        }
    }
```

Шаг 3: Создайте переменную для отправки запроса и хранения ответа в формате JSON.

Используйте переменные, созданные на предыдущем шаге, в качестве параметров метода requests.put(). Этот метод отправляет запрос HTTP PUT в API RESTCONF. Присвойте результат запроса переменной с именем resp. Эта переменная будет содержать JSON-ответ от API. Если запрос прошел успешно, JSON будет содержать возвращенную модель данных YANG.

Прежде чем вводить утверждения, обратите внимание, что эта спецификация переменной должна находиться только в одной строке вашего сценария. Введите следующие утверждения:

Примечание: Эта спецификация переменной должна находиться на одной строке в вашем сценарии.

```
resp = requests.put(api_url, data=json.dumps(yangConfig), auth=basicauth, headers=headers, verify=False)
```

Введите приведенный ниже код для обработки ответа. Если ответ является одним из сообщений об успехе HTTP, будет выведено первое сообщение. Любое другое значение кода считается ошибкой. В случае обнаружения ошибки будет выведен код ответа и сообщение об ошибке.

```
if(resp.status_code >= 200 and resp.status_code <= 299):
    print("STATUS OK: {}".format(resp.status_code))
else:
    print('Error. Status Code: {} \nError message: {}'.format(resp.status_code,resp.json()))
```

В таблице ниже перечислены различные элементы этих заявлений:

Элемент	Описание
resp	Переменная для хранения ответа от API
requests.put()	Метод, который фактически выполняет PUT-запрос
api_url	Переменная, в которой хранится строка адреса URL
data	Данные для отправки в конечную точку API, которые форматируются как JSON.
auth	Переменная кортежа, созданная для хранения информации об аутентификации.
headers=headers	Параметр, которому присваивается переменная headers.
verify=False	Параметр, отключающий проверку SSL-сертификата при выполнении запроса.
resp.status_code	Код состояния HTTP в ответе на запрос API PUT.

Сохраните и запустите сценарий для отправки запроса PUT на CSR1kv. Вы должны получить сообщение 201 Status Created. Если нет, проверьте свой код и конфигурацию CSR1kv.

Вы можете проверить, что интерфейс был создан, введя show ip interface brief на CSR1kv.

```
CSR1kv# show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.56.101  YES manual up                    up
Loopback1              10.1.1.1        YES other  up                    up
Loopback2              10.2.1.1        YES other  up                    up
CSR1kv#
```

Программы, используемые в данной лабораторной

В этой лабое использовались следующие программы Python:

```
#=====================================================================
#resconf-get.py
import json
import requests
requests.packages.urllib3.disable_warnings()

api_url = "https://192.168.56.101/restconf/data/ietf-interfaces:interfaces"

headers = { "Accept": "application/yang-data+json",
            "Content-type":"application/yang-data+json"
           }

basicauth = ("cisco", "cisco123!")

resp = requests.get(api_url, auth=basicauth, headers=headers, verify=False)

print(resp)

response_json = resp.json()
print(json.dumps(response_json, indent=4))

#end of file

#=====================================================================
#resconf-put.py
import json
import requests
requests.packages.urllib3.disable_warnings()

api_url              =              "https://192.168.56.101/restconf/data/ietf-
interfaces:interfaces/interface=Loopback2"

headers = { "Accept": "application/yang-data+json",
            "Content-type":"application/yang-data+json"
           }

basicauth = ("cisco", "cisco123!")

yangConfig = {
    "ietf-interfaces:interface": {
        "name": "Loopback2",
        "description": "My second RESTCONF loopback",
        "type": "iana-if-type:softwareLoopback",
```

```
            "enabled": True,
            "ietf-ip:ipv4": {
                "address": [
                    {
                        "ip": "10.2.1.1",
                        "netmask": "255.255.255.0"
                    }
                ]
            },
            "ietf-ip:ipv6": {}
        }
    }

resp = requests.put(api_url, data=json.dumps(yangConfig), auth=basicauth, headers=headers, verify=False)

if(resp.status_code >= 200 and resp.status_code <= 299):
    print("STATUS OK: {}".format(resp.status_code))
else:
    print('Error. Status Code: {} \nError message: {}'.format(resp.status_code,resp.json()))
```

Лабораторная работа №8.6.7

Создание Python программы для управления Webex Teams

Цель лабораторной работы:

Часть 1: Создание сети и проверка подключения

Часть 2: Получение Access Token для Webex Teams

Часть 3: Проверка Access Token

Часть 4: Управление людьми в Webex Teams

Часть 5: Управление комнатами в Webex Teams

Часть 6: Управление членством в Webex Teams

Часть 7: Управление сообщениями в Webex Teams

Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

Порядок выполнения работы

Часть 1: Запуск виртуальной машины DEVASC

Если вы еще не завершили **лабораторную работу - Установка лабораторной среды виртуальной машины**, сделайте это сейчас. Если вы уже завершили эту лабораторную работу, запустите виртуальную машину **DEVASC**.

Примечание: В рамках лабораторной работы "**Установка лабораторной среды виртуальной машины**" необходимо установить приложение **Webex**

Teams и добавить в список контактов как минимум еще одного человека. Это необходимо сделать до начала работы над лабораторной работой.

Часть 2: Получение Access Token для Webex Teams

В этой части вы зарегистрируетесь или войдете в свою учетную запись Webex, изучите документацию по API, получите свой маркер доступа, а затем протестируете свой маркер доступа, который вы будете использовать в своих вызовах API.

Шаг 1: Войдите в Webex

Откройте веб-браузер Chromium.

Перейдите на веб-сайт разработчика Webex: https://developer.webex.com/.

Войдите в систему, если у вас уже есть учетная запись. Если нет, пройдите процесс регистрации.

Шаг 2: Изучите документацию по API.

Нажмите **Documentation**.

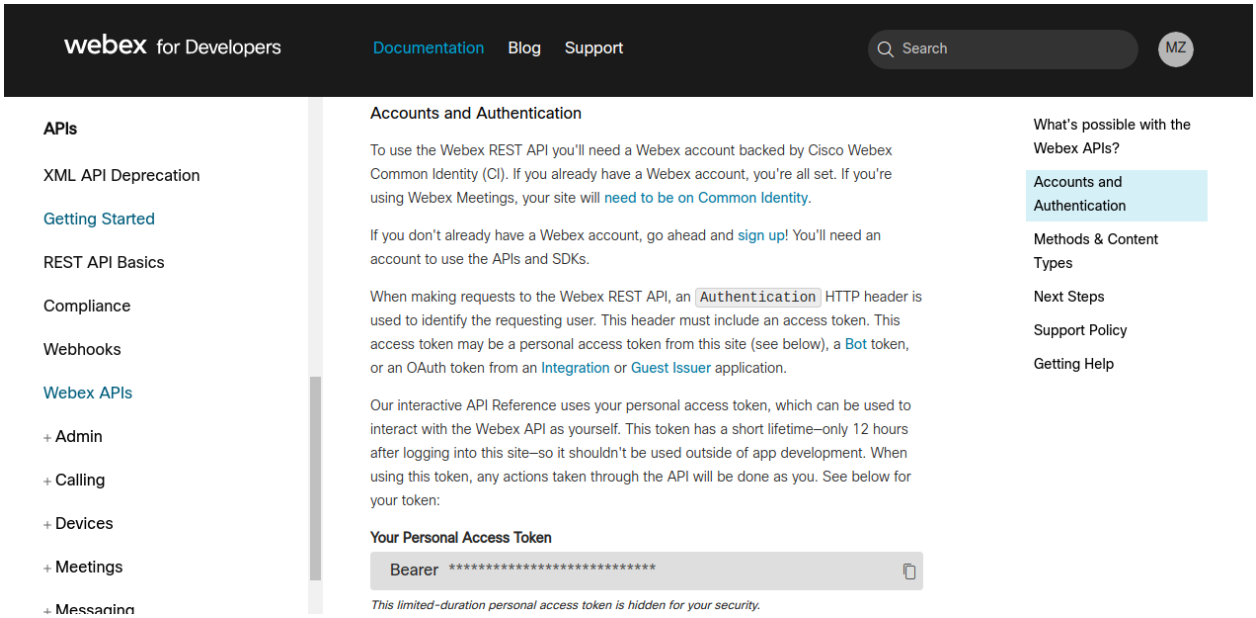
Webex имеет множество API, которые вы можете использовать в своих приложениях. Щелкните API, чтобы раскрыть его подменю.

Изучите все разновидности вызовов API. В этой лабораторной работе мы будем использовать документацию API для People, Rooms, Membership и Message.

Шаг 3: Получите токен доступа.

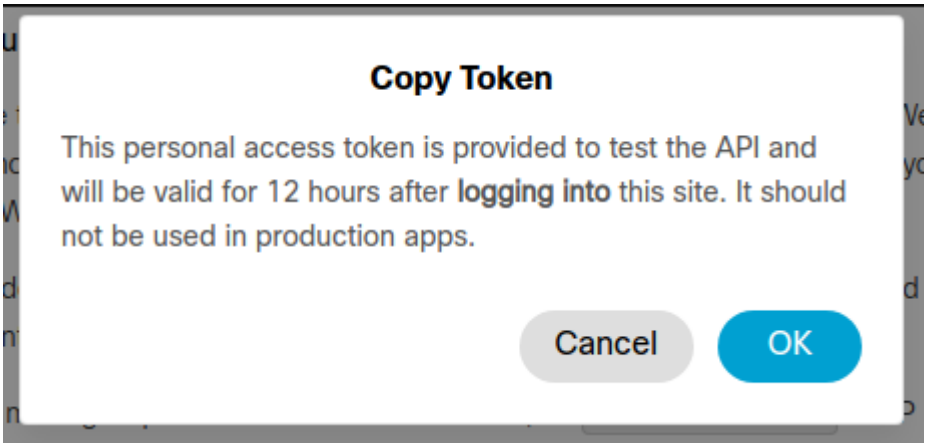
Находясь в разделе **Documentation**, при необходимости прокрутите страницу назад к верху и нажмите **Getting Started** в разделе **REST API**.

В разделе **Accounts and Authentication** обратите внимание, что Webex поддерживает персональный токен доступа. Токен аутентификации требуется для всех вызовов REST API. Нажмите **Copy** в разделе **Your Personal Access Token**.



Примечание: Персональный токен доступа предоставляет доступ к вашей учетной записи всем, кто его знает. Обязательно храните его в тайне.

Примечание: Вы получите сообщение о том, что токен действителен в течение определенного периода времени, который на момент написания данной работы составлял 12 часов. Вам нужно будет получить новый токен и обновить Python скрипт, если вы вернетесь к этой работе после истечения срока действия токена.



Скопируйте свой личный токен доступа в текстовый файл, чтобы использовать его в дальнейшем в этой лабораторной работе.

Часть 3: Проверка Access Token

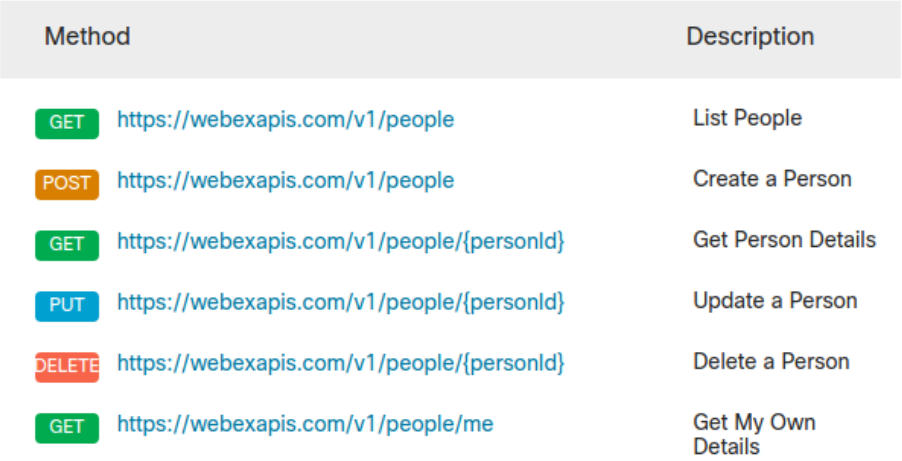
Вы можете проверить свой токен доступа в документации OpenAPI на сайте разработчика. Однако вы будете использовать свой токен в сценариях Python. Поэтому вам следует проверить, что он работает и в сценарии.

Шаг 1: Проверьте свой токен доступа на сайте разработчика.

Вы можете проверить свой токен доступа в документации OpenAPI по адресу https://developer.webex.com.

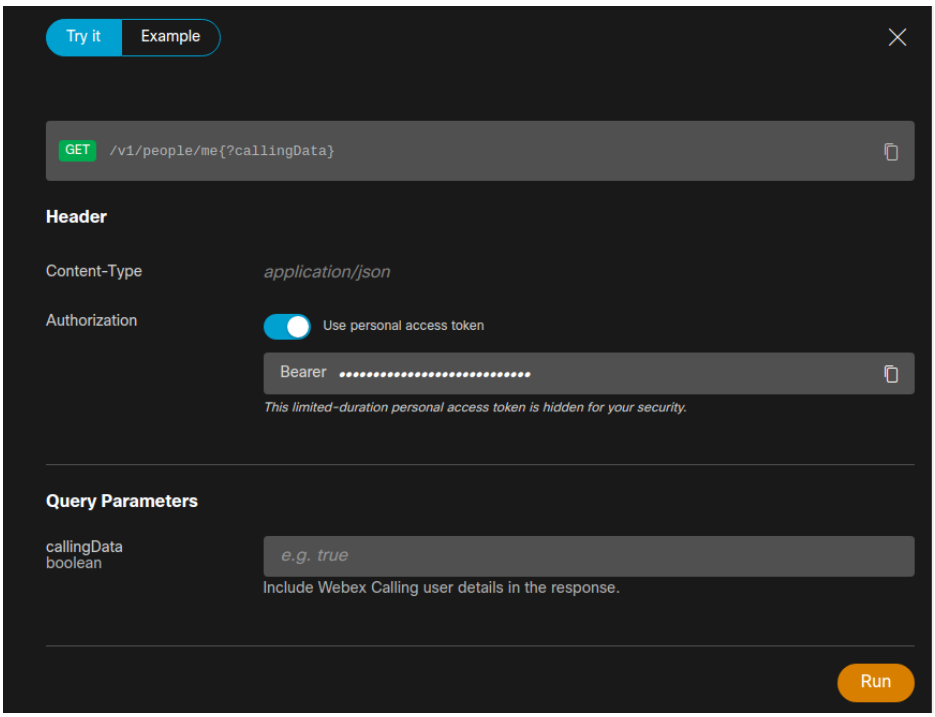
Вернитесь в браузер и нажмите **Documentation**, если необходимо.

В разделе **Full API Reference** нажмите **People**, а затем нажмите **Get My Own Details**.

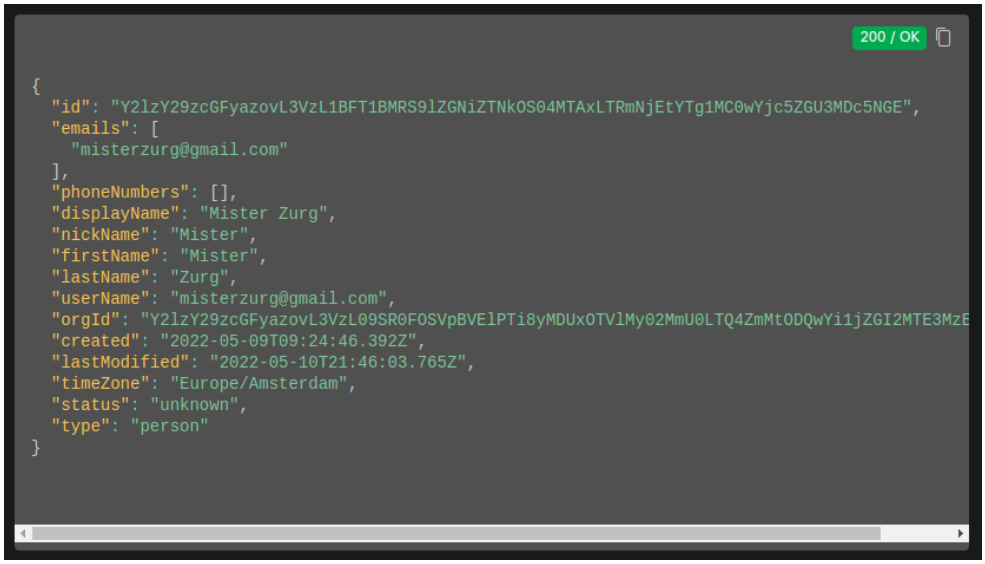


Method		Description
GET	https://webexapis.com/v1/people	List People
POST	https://webexapis.com/v1/people	Create a Person
GET	https://webexapis.com/v1/people/{personId}	Get Person Details
PUT	https://webexapis.com/v1/people/{personId}	Update a Person
DELETE	https://webexapis.com/v1/people/{personId}	Delete a Person
GET	https://webexapis.com/v1/people/me	Get My Own Details

На панели **Try it** справа обратите внимание, что ваш токен уже введён.



Вы можете нажать кнопку **Try it** или **Run**, чтобы проверить свой доступ. Вы увидите **Response** с вашей личной информацией.



Нажмите **Request**, чтобы увидеть полный URL-адрес, использованный для отправки GET-запроса. Вы будете использовать этот URL на следующем шаге в своем сценарии Python.

В средней секции вы можете просмотреть всю документацию по **Response Properties**.

Шаг 2: Используйте Python скрипт для проверки токена доступа.

Откройте VS code. Затем нажмите **File > Open Folder...** и перейдите в каталог devnet-src/webex-teams. Нажмите **OK**.

На панели EXPLORER теперь должны быть видны все файлы placeholder.py, которые вы будете использовать в этой лабораторной работе. Щелкните файл **authentciation.py**.

Поместите в файл следующий код. Не забудьте заменить **your_token_here** на ваш персональный токен доступа, который вы скопировали в предыдущем шаге.

```
import requests
import json

access_token = 'your_token_here'
url = 'https://webexapis.com/v1/people/me'
headers = {
    'Authorization': 'Bearer {}'.format(access_token)
}
res = requests.get(url, headers=headers)
print(json.dumps(res.json(), indent=4))
```

Сохраните и запустите файл. Вы должны получить тот же результат, который вы видели в документации OpenAPI.

Примечание: Значения некоторых ключей были сокращены в приведенном ниже выводе.

```
devasc@labvm:~/labs/devnet-src/webex-teams$ python3 authentication.py
{
    "id": "Y2lz...c5NGE",
    "emails": [
        "ваша-почта@example.com"
    ],
    "phoneNumbers": [],
    "displayName": "Ваше-Имя Ваша-Фамилия",
    "nickName": "Ваше-Имя-Пользователя",
    "firstName": "Ваше-Имя",
    "lastName": "Ваша-Фамилия",
    "avatar": "https://9643-417f-9974...6baa4~1600",
    "orgId": "Y2lzY2...E1YjQ",
    "created": "2022-05-09T09:24:46.392Z",
    "lastActivity": "2022-05-10T21:46:03.765Z",
    "status": "active",
    "type": "person"
}
devasc@labvm:~/labs/devnet-src/webex-teams$
```

```
devasc@labvm:~/labs/devnet-src/webex-teams$ python3 authentication.py
{
    "id": "Y2lzY29zcGFyazovL3VzL1BFT1BMRS9lZGNiZTNkOS04MTAxLTRmNjEtYTg1MC0wYjc5ZGU3MDc5NGE",
    "emails": [
        "misterzurg@gmail.com"
    ],
    "phoneNumbers": [],
    "displayName": "Mister Zurg",
    "nickName": "Mister",
    "firstName": "Mister",
    "lastName": "Zurg",
    "userName": "misterzurg@gmail.com",
    "orgId": "Y2lzY29zcGFyazovL3VzL09SR0FOSVpBVElPTi8yMDUxOTVlMy02MmU0LTQ4ZmMtODQwYi1jZGI2MTE
3MzE1YjQ",
    "created": "2022-05-09T09:24:46.392Z",
    "lastModified": "2022-05-10T21:46:03.765Z",
    "timeZone": "Europe/Amsterdam",
    "status": "unknown",
    "type": "person"
}
devasc@labvm:~/labs/devnet-src/webex-teams$
```

Часть 4: Управление людьми в Webex Teams

В Webex Teams люди – это зарегистрированные пользователи. С помощью API можно получить список людей, создать человека, получить данные об отдельном человеке, обновить человека и удалить его.

Шаг 1: Найдите в документации API сведения о зарегистрированном пользователе Webex Teams.

Вернитесь на веб-сайт developer.webex.com. В разделе **API Reference > People** щелкните **Method for List People**.

В разделе **Query Parameters**, найдите параметр электронной почты. Это параметр, который вы будете использовать для поиска конкретного пользователя в вашей организации. В качестве альтернативы можно использовать параметр **displayName**, если вы знаете точное имя. Вы можете использовать функцию **Try it**.

Шаг 2: Используйте Python скрипт для получения списка сведений о зарегистрированном пользователе Webex Teams.

В VS Code щелкните файл **list-people.py**

Поместите в файл следующий код. Обязательно замените your_token_here на ваш личный токен, а **user@example.com** — на реального пользователя Webex Team в вашей организации.

```
import requests
import json

access_token = 'your_token_here'
url = 'https://webexapis.com/v1/people'
headers = {
    'Authorization': 'Bearer {}'.format(access_token),
    'Content-Type': 'application/json'
}
params = {
    'email': 'user@example.com'
}
```

```
res = requests.get(url, headers=headers, params=params)
print(json.dumps(res.json(), indent=4))
```

Сохраните и запустите скрипт. Вы должны получить вывод, аналогичный приведенному ниже. Если вы получите сообщение вида **{'message': 'Invalid email address.'...**, это означает, что вы не заменили пустой параметр email на легитимный адрес электронной почты зарегистрированного пользователя Webex Teams. Значение для ключа id будет использовано в следующем вызове API

Примечание: Значения некоторых ключей были усечены в приведенном ниже выводе.

```
devasc@labvm:~/labs/devnet-src/webex-teams$ python3 list-people.py
{
    "notFoundIds": null,
    "items": [
        {
            "id": "Y2l...2I", # Вы будете использовать это значение в следующем
шаге
            "emails": [
                "user@example.com"
            ],
            "phoneNumbers": [
                {
                    "type": "mobile",
                    "value": "+1234567690"
                }
            ],
            "displayName": "displayName",
            "nickName": "nickName",
            "firstName": "firstName",
            "lastName": "lastName",
            "avatar": "https://9643-417f-9974...6baa4~1600",
            "orgId": "Y2lzY...UxMGY",
            "created": "2012-06-15T20:39:19.726Z",
            "lastActivity": "2022-05-10T21:46:03.765Z",
            "status": "active",
            "type": "person"
        }
    ]
}
devasc@labvm:~/labs/devnet-src/webex-teams$
```

Шаг 3: Перечислите дополнительные административные данные человека.

Если вы являетесь администратором Webex Teams, вы можете получить дополнительные сведения о человеке, используя значение ключа person id в вызове API. Добавьте следующий код в сценарий list-people.py. Замените **previous_id_here** на значение id из предыдущего вызова API.

```
person_id = 'previous_id_here'
url = 'https://webexapis.com/v1/people/{}'.format(person_id)
headers = {
    'Authorization': 'Bearer {}'.format(access_token),
```

```
    'Content-Type': 'application/json'
}
res = requests.get(url, headers=headers)
print(json.dumps(res.json(), indent=4))
```

Сохраните файл и запустите его. Как человек, не являющийся администратором, вы получите информацию, очень похожую на предыдущий шаг.

```
devasc@labvm:~/labs/devnet-src/webex-teams$ python3 list-people.py
{
    <вывод первого вызова API опущен>
}
{
    "id": "Y2l...2I",
    "emails": [
        "user@example.com"
    ],
    "phoneNumbers": [
        {
            "type": "mobile",
            "value": "+1234567890"
        }
    ],
    "displayName": "displayName",
    "nickName": "nickName",
    "firstName": "firstName",
    "lastName": "lastName",
    "avatar": "https://9643-417f-9974...6baa4~1600",
    "orgId": "Y2l...MGY",
    "created": "2012-06-15T20:39:19.726Z",
    "lastActivity": "2022-05-10T21:50:03.765Z",
    "status": "active",
    "type": "person"
}
devasc@labvm:~/labs/devnet-src/webex-teams$
```

Часть 5: Управление комнатами в Webex Teams

Комнаты, также называемые пространствами в пользовательском интерфейсе, позволяют людям отправлять сообщения и файлы для виртуального сотрудничества в местах коллективных встреч. В этой части вы увидите список комнат, создадите комнату и получите сведения о комнате.

Шаг 1: Найдите и изучите документацию API для комнат.

Вернитесь на developer.webex.com. В разделе API Reference нажмите Rooms.

Изучите различные вызовы API, которые можно выполнить с помощью API Rooms.

Щелкните GET-запрос для List Rooms и изучите параметры запроса.

Шаг 2: Используйте Python скрипт для составления списка всех комнат для аутентифицированного пользователя.

Для этого шага вам нужно быть участником хотя бы одной комнаты. Разговор с одним собеседником считается комнатой в Webex Teams.

В VS Code щелкните файл **list-rooms.py**.

Поместите в файл следующий код. Обязательно замените **your_token_here** на ваш личный маркер доступа.

```
import requests

access_token = 'your_token_here'
url = 'https://webexapis.com/v1/rooms'
headers = {
    'Authorization': 'Bearer {}'.format(access_token),
    'Content-Type': 'application/json'
}
params={'max': '100'}
res = requests.get(url, headers=headers, params=params)
print(res.json())
```

Сохраните и запустите файл. Ваш результат будет отличаться от приведенного ниже. Здесь указана только одна комната. Значения идентификаторов были усечены.

```
devasc@labvm:~/labs/devnet-src/webex-teams$ python3 list-rooms.py
{'items': [{'id': 'Y2l...ZTE0', 'title': 'User Name', 'type': 'direct',
'isLocked': False, 'lastActivity': '2022-05-10T22:14:27.361Z', 'creatorId':
'Y2lz...yM2U', 'created': '2022-05-10T22:14:27.361Z', 'ownerId': 'Y2lz...xMGY'}
# additional rooms displayed up to 'max' value.
]}
devasc@labvm:~/labs/devnet-src/webex-teams$
```

Шаг 3: Найдите и изучите документацию по API для отправки сообщений в API Rooms.

Вернитесь на веб-сайт developer.webex.com. В разделе API Reference нажмите Rooms, если необходимо.

API Rooms имеет один метод POST для создания комнаты. Нажмите на ссылку посмотреть доступные **Query Parameters**. В своей программе вы будете использовать необходимый параметр **title**.

Шаг 4: Используйте Python скрипт для создания комнаты.

В VS Code щелкните файл **create-rooms.py**.

Поместите в файл следующий код. Обязательно замените **your_token_here** на ваш личный маркер доступа. Обратите внимание, что это POST-запрос и в нем используется параметр title.

```
import requests

access_token = 'your_token_here'
url = 'https://webexapis.com/v1/rooms'
headers = {
    'Authorization': 'Bearer {}'.format(access_token),
    'Content-Type': 'application/json'
}
params={'title': 'Подготовка к собесу ASMR!'}
res = requests.post(url, headers=headers, json=params)
print(res.json())
```

Сохраните и запустите файл. Вы должны получить ответ, подобный следующему. Значения идентификаторов были усечены. Идентификатор и название комнаты выделены. Скопируйте идентификатор комнаты и сохраните его в текстовом файле для использования в дальнейшей работе.

```
devasc@labvm:~/labs/devnet-src/webex-teams$ python3 create-rooms.py
{'id': 'Y2l...jEy', 'title': 'Подготовка к собесу ASMR!', 'type': 'group', 'isLocked': False, 'lastActivity': '2022-05-10T22:14:27.361Z', 'creatorId': 'Y2l...NGE', 'created': '2022-05-10T22:14:27.361Z', 'ownerId': 'Y2l...YjQY'}
devasc@labvm:~/labs/devnet-src/webex-teams$
```

```
devasc@labvm:~/labs/devnet-src/webex-teams$ python3 create-rooms.py
{'id': 'Y2lzY29zcGFyazovL3VybjpURUFNOmV1LWNlbnRyYWwtMV9rL1JPT00vOGU2Njc5M
TAtZDBhZS0xMWVjLWEwNmUtNDFhN2JmMTliZjEy', 'title': 'Подготовка к собесу A
SMR!', 'type': 'group', 'isLocked': False, 'lastActivity': '2022-05-10T22
:14:27.361Z', 'creatorId': 'Y2lzY29zcGFyazovL3VzL1BFT1BMRS9lZGNiZTNkOS04M
TAxLTRmNjEtYTg1MC0wYjc5ZGU3MDc5NGE', 'created': '2022-05-10T22:14:27.361Z
', 'ownerId': 'Y2lzY29zcGFyazovL3VzL09SR0FOSVpBVElPTi8yMDUxOTVlMy02MmU0LT
Q4ZmMtODQwYi1jZGI2MTE3MzE1YjQ'}
devasc@labvm:~/labs/devnet-src/webex-teams$
```

В приложении Webex Teams проверьте, что теперь вы видите комнату DevNet Associate Training! В настоящее время вы являетесь единственным участником.

Шаг 5: Используйте Python скрипт для получения информации о номере.

Что если вы хотите начать собрание в новой комнате? Вы можете сделать еще один вызов GET, чтобы получить адрес протокола инициирования сеанса (SIP), URL собрания и номера телефонов для набора.

В VS Code щелкните файл **get-room-details.py**.

Поместите в файл следующий код. Замените **your_token_here** на ваш персональный токен доступа. Замените **your_room_id** на значение, полученное в предыдущем шаге.

```
import requests

access_token = 'your_token_here'
room_id = 'your_room_id'
url = 'https://webexapis.com/v1/rooms/{}/meetingInfo'.format(room_id)
headers = {
    'Authorization': 'Bearer {}'.format(access_token),
    'Content-Type': 'application/json'
}
res = requests.get(url, headers=headers)
print(res.json())
```

Сохраните и запустите файл. Вы должны получить ответ, подобный следующему. Значения были усечены.

```
devasc@labvm:~/labs/devnet-src/webex-teams$ python3 get-room-details.py
{'roomId': 'Y2l...jEy', 'meetingLink': 'https://cisco.webex.com/m/804...4fa2', 'sipAddress': '274...198@cisco.webex.com', 'meetingNumber': '274...7979', 'callInTollFreeNumber': '+8-800-...-3535', 'callInTollNumber': '+8-800-...-3535'}
devasc@labvm:~/labs/devnet-src/webex-teams$
```

```
devasc@labvm:~/labs/devnet-src/webex-teams$ python3 get-room-details.py
{'roomId': 'Y2lzY29zcGFyazovL3VybjpURUFNOmV1LWNlbnRyYWwtMV9rL1JPT00vOGU2N
jc5MTAtZDBhZS0xMWVjLWEwNmUtNDFhN2JmMTliZjEy', 'meetingLink': 'https://mee
t198.webex.com/m/80487355-cceb-4944-b131-ae96b7b24fa2', 'sipAddress': '27
439217979@meet198.webex.com', 'meetingNumber': '27439217979', 'meetingId'
: 'd41800e8ef75456d8761cb45cd243565'}
devasc@labvm:~/labs/devnet-src/webex-teams$
```

Часть 6: Управление членством в Webex Teams

В этой части вы будете использовать API Membership, чтобы добавить кого-либо в свою комнату.

Шаг 1: Найдите и изучите документацию API для членства.

Вернитесь на веб-сайт developer.webex.com. В разделе API Reference нажмите Memberships.

Изучите различные вызовы API, которые можно выполнить с помощью API Memberships.

Выберите GET-запрос для List Memberships и изучите параметры запроса.

Шаг 2: Используйте Python скрипт для составления списка членов комнаты.

В VS Code щелкните файл **list-memberships.py**.

Поместите в файл следующий код. Замените **your_token_here** на ваш персональный токен доступа. Замените **your_room_id** на значение, полученное в предыдущей части.

```
import requests

access_token = 'your_token_here'
room_id = 'your_room_id'
url = 'https://webexapis.com/v1/memberships'
headers = {
    'Authorization': 'Bearer {}'.format(access_token),
    'Content-Type': 'application/json'
}
params = {'roomId': room_id}
res = requests.get(url, headers=headers, params=params)
print(res.json())
```

Сохраните и запустите файл. Вы должны получить ответ, похожий на следующий. Вы должны быть единственным участником, если только вы не добавили кого-либо в приложение Webex Teams. Значения идентификаторов были усечены.

```
devasc@labvm:~/labs/devnet-src/webex-teams$ python3 list-memberships.py
{'items': [{'id': 'Y2l...RjZg', 'roomId': 'Y2l...GNm', 'personId': 'Y2l...M2U',
'personEmail': 'user@example.com', 'personDisplayName': 'personDisplayName',
'personOrgId': 'Y2l...MGY', 'isModerator': False, 'isMonitor': False, 'created':
'2022-05-10T22:16:27.361Z'}]}
devasc@labvm:~/labs/devnet-src/webex-teams$
```

Шаг 3: Найдите и изучите документацию по API для отправки сообщений в Memberships API.

Вернитесь на веб-сайт developer.webex.com. В разделе API Reference нажмите Memberships, если необходимо.

API Memberships имеет один метод POST для Create a Membership. Нажмите на ссылку посмотреть доступные Query Parameters. В своем сценарии вы будете использовать необходимые параметры roomID и personEmail.

Шаг 4: С помощью Python скрипта создайте членство, добавив кого-либо в комнату.

Для этого шага вам понадобится электронная почта другого человека, который является зарегистрированным пользователем Webex Teams в вашей организации. Вы можете использовать ту же электронную почту, которую вы использовали ранее для перечисления сведений о человеке.

В VS Code щелкните файл **create-membership.py.**

Поместите в файл следующий код. Замените **your_token_here** на ваш персональный токен доступа. Замените **your_room_id на** значение, полученное в предыдущей части. Замените **new-user@example.com** на email человека, которого вы хотите добавить в комнату.

```
import requests

access_token = 'your_token_here'
room_id = 'your_room_id'
person_email = 'new-user@example.com'
url = 'https://webexapis.com/v1/memberships'
headers = {
    'Authorization': 'Bearer {}'.format(access_token),
    'Content-Type': 'application/json'
}
params = {'roomId': room_id, 'personEmail': person_email}
res = requests.post(url, headers=headers, json=params)
print(res.json())
```

Сохраните и запустите файл. Вы должны получить ответ, похожий на следующий. Вы должны быть единственным участником, если только вы не добавили кого-либо в приложение Webex Teams. Значения идентификаторов были усечены.

```
devasc@labvm:~/labs/devnet-src/webex-teams$ python3 list-memberships.py
{'items': [{'id': 'Y2l...RjZg', 'roomId': 'Y2l...GNm', 'personId': 'Y2l...M2U',
'personEmail':  'user@example.com',  'personDisplayName':  'personDisplayName',
'personOrgId': 'Y2l...MGY', 'isModerator': False, 'isMonitor': False, 'created':
'2022-05-10T22:16:27.361Z'}]}
devasc@labvm:~/labs/devnet-src/webex-teams$
```

Часть 7: Управление сообщениями в Webex Teams

В Webex Teams сообщение может содержать обычный текст, Markdown или вложение файла. Каждое сообщение отображается в отдельной строке вместе с меткой времени и информацией об отправителе. Вы можете использовать этот API Messages для списка, создания и удаления сообщений. В этой части вы отправите сообщение в комнату, которую вы создали в этой лабораторной работе.

Шаг 1: Найдите и изучите документацию API для сообщений.

Вернитесь на веб-сайт developer.webex.com. В разделе Справочник API нажмите Сообщения.

Изучите различные вызовы API, которые можно выполнить с помощью API Messages.

Щелкните POST-запрос для **Create a Message** и изучите **Query Parameters**. Обратите внимание, что для простого текстового сообщения можно использовать параметр text или markdown. В этом шаге вы зададите сообщение с форматированием в формате Markdown. Поищите в Интернете, чтобы узнать больше о Markdown.

Шаг 2: Используйте Python скрипт для отправки сообщения в комнату Webex.

В VS Code щелкните файл **create-markdown-message.py.**

Поместите в файл следующий код. Замените **your_token_here** на ваш персональный токен доступа. Замените **your_room_id** на значение, полученное в предыдущей части.

```
import requests

access_token = 'your_token_here'
room_id = 'your_room_id'
message = 'Here in my garage, just bought this new Lamborghini here.'
url = 'https://webexapis.com/v1/messages'
headers = {
    'Authorization': 'Bearer {}'.format(access_token),
    'Content-Type': 'application/json'
}
params = {'roomId': room_id, 'markdown': message}
res = requests.post(url, headers=headers, json=params)
print(res.json())
```

Сохраните и запустите файл. Вы должны получить ответ, похожий на следующий. Обратите внимание, что Markdown был преобразован в HTML. Значения идентификаторов были усечены.

```
devasc@labvm:~/labs/devnet-src/webex-teams$ python3 create-markdown-message.py
{'id': 'Y2l...RjZg', 'roomId': 'Y2l...GNm', 'roomType': 'group', 'text': 'Here in my garage, just bought this new Lamborghini here.', 'personId': 'Y2l...NGE', 'personEmail': 'user@example.com', 'markdown': 'Here in my garage, just bought this new Lamborghini here.', 'html': '<p>Here in my garage, just bought this new Lamborghini here.</p>', 'created': '2022-05-10T22:27:41.505Z'}
devasc@labvm:~/labs/devnet-src/webex-teams$
```

```
devasc@labvm:~/labs/devnet-src/webex-teams$ python3 create-markdown-message.py
{'id': 'Y2lzY29zcGFyazovL3VybjpURUFNOmV1LWNlbnRyYWwtMV9rL01FU1NBR0UvNjdiZjNiMTAtZDBiMC0xMWVjLWI2YWUtYmI3ZWNmOTk2NGNm', 'roomId': 'Y2lzY29zcGFyazovL3VybjpURUFNOmV1LWNlbnRyYWwtMV9rL1JPT00vOGU2Njc5MTAtZDBhZS0xMWVjLWEwNmUtNDFhN2JmMTliZjEy', 'roomType': 'group', 'text': 'Here in my garage, just bought this new Lamborghini here.', 'personId': 'Y2lzY29zcGFyazovL3VzL1BFT1BMRS9lZGNiZTNkOS04MTAxLTRmNjEtYTg1MC0wYjc5ZGU3MDc5NGE', 'personEmail': 'misterzurg@gmail.com', 'markdown': 'Here in my garage, just bought this new Lamborghini here.', 'html': '<p>Here in my garage, just bought this new Lamborghini here.</p>', 'created': '2022-05-10T22:27:41.505Z'}
devasc@labvm:~/labs/devnet-src/webex-teams$
```

Packet Tracer — Сравнение CLI и SDN контроллера управления сетью

Цель лабораторной работы:

Часть 1: Изучение топологии сети

Часть 2: Использование CLI для сбора информации

Часть 3: Настройка контроллера SDN

Часть 4: SDN контроллер для обнаружения топологии

Часть 5: SDN контроллер для сбора информации

Часть 6: SDN контроллер для настройки параметров сети

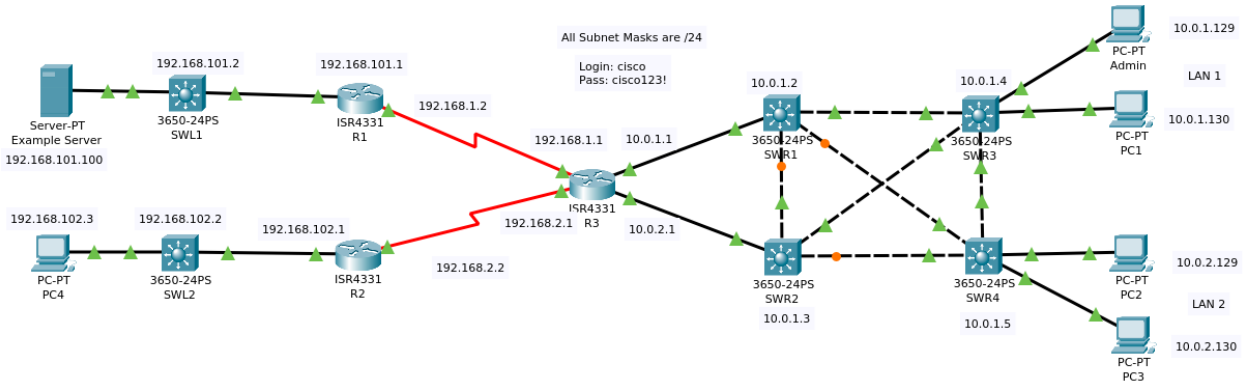
Необходимые ресурсы:

- 1 ПК
- Packet Tracer 7.4.0

Таблица адресации

Устройство	Интерфейс	IP Адрес
R1	G0/0/0	192.168.101.1
	S0/1/0	192.168.1.2
R2	G0/0/0	192.168.101.2
	S0/1/1	192.168.2.2
R3	G0/0/0	10.0.1.1
	G0/0/1	10.0.2.1
	S0/1/0	192.168.1.1
	S0/1/1	192.168.2.1
SWL1	VLAN 1	192.168.101.2
SWL2	VLAN 1	192.168.102.2
SWR1	VLAN 1	10.0.1.2
SWR2	VLAN 1	10.0.1.3
SWR3	VLAN 1	10.0.1.4
SWR4	VLAN 1	10.0.1.5
Admin	NIC	10.0.1.129
PC1	NIC	10.0.1.130
PC2	NIC	10.0.2.129
PC3	NIC	10.0.2.130
PC4	NIC	192.168.102.3
Example Server	NIC	192.168.101.100
PT-Controlle	NIC	192.168.101.254

Схема сети



Порядок выполнения работы

Часть 1: Изучение топологии сети

В этой части вы познакомитесь с топологией, которую будете использовать для работы по программированию сети.

Шаг 1: Изучите документацию по конфигурации сети

Сеть настроена следующим образом:

- Маршрутизаторы работают под управлением OSPFv2.
- SSH включен на всех устройствах с пользователем **cisco** и паролем **cisco123!**
- На R1 нет хостов.
- R2 LAN IPv4 настроен статически.
- R3 является сервером DHCPv4 для LAN1 и LAN2.
- Коммутаторы относятся к уровню 2 (без VLAN).
- Все коммутаторы **SWR#** принадлежат к LAN1.

Шаг 2: Убедитесь, что все устройства могут пинговать друг друга.

Либо используйте командную строку на каждом устройстве, либо используйте инструмент Add Simple PDU (P), чтобы проверить, что все устройства могут пинговать друг друга

Часть 2: Использование CLI для сбора информации

В этой части вы вручную обращаетесь к каждому устройству, чтобы собрать информацию о версии программного обеспечения.

Шаг 1: С Admin PC получите безопасный доступ к коммутатору SWR3.

Щелкните **Admin** > **Desktop** > **Command Prompt**.

Введите команду **ssh -l cisco 10.0.1.4**. Параметр -l - это буква "L", а не цифра один.

Когда появится запрос, введите **cisco123!** в качестве пароля. Теперь вы вошли в систему **SWR3**.

Шаг 2: Соберите информацию о программном обеспечении на SWR3.

Введите следующую команду, чтобы отфильтровать вывод команды show version для просмотра только установленной версии. RELEASE SOFTWARE,

установленного на устройстве. Обратите внимание, что SWR3 работает под управлением IOS 16.3.2 и Boot Loader 4.2.6

```
SWR3# show version | include RELEASE
Cisco IOS Software [Denali], Catalyst L3 Switch Software (CAT3K_CAA-UNIVERSALK9-M), Version 16.3.2, RELEASE SOFTWARE (fc4) BOOTLDR: CAT3K_CAA Boot Loader (CAT3K_CAA-HBOOT-M) Version 4.26, RELEASE SOFTWARE (P)
SWR3#
```

Скопируйте информацию в буфер обмена

Откройте пустой файл в текстовом редакторе и вставьте в него скопированную информацию.

Сохраните файл под именем **software-versions.txt**

Шаг 3: Соберите информацию о программном обеспечении для остальных сетевых устройств.

Из командной строки на SWR3 получите безопасный доступ к другому сетевому устройству и повторите Шаг 2 выше.

Продолжайте документировать версии программного обеспечения, пока не закончите работу со всеми девятью сетевыми устройствами: SWL1, SWL2, SWR1, SWR2, SWR3, SWR4, R1, R2 и R3.

Выйдите из всех сеансов SSH.

Часть 3: Настройка контроллера SDN

В течение многих лет сетевые администраторы использовали ранние средства автоматизации, такие как сценарии bash или программное обеспечение с поддержкой SNMP, чтобы выполнить процесс, аналогичный тому, что вы сделали в предыдущем шаге. Однако с появлением SDN, этот процесс был значительно усовершенствован. Packet Tracer предоставляет простой PT-Controller для имитации контроллера SDN. В этой части вы подключите и настроите PT-Controller.

Примечание: Чтобы узнать больше о реализации сетевого контроллера в Packet Tracer, щелкните меню Справка, затем Содержание. В указателе слева, примерно в середине, вы найдете заголовок Configuring Devices (Настройка устройств). Под этим заголовком найдите пункт Сетевые контроллеры.

Шаг 1: Добавьте сетевой контроллер в топологию.

В левом нижнем углу интерфейса Packet Tracer нажмите **End Devices** > **Network Controller**.

Добавьте сетевой контроллер в пустое место слева от коммутатора **SWL1**. Его имя уже должно быть **PT-Controller0**. Если это не так, щелкните имя и измените его.

Внизу снова нажмите на кнопку с изображением **Connections**. Щелкните на **Copper StraightThrough** кабель.

Щелкните **PT-Controller0** и выберите **GigabitEthernet0**. Затем щелкните **SWL1** и выберите первый доступный интерфейс Gigabit Ethernet**.**

Шаг 2: Настройте подключение для PT-Controller0.

Нажмите **PT-Controller0 > Config**

Для параметра **Gateway/DNS IPv4** введите **192.168.101.1** в качестве адреса шлюза.

Слева в разделе **INTERFACE** нажмите **GigabitEthernet0**.

Для конфигурации IP введите IP-адрес **192.168.101.254** и маску подсети **255.255.255.0**.

Слева в разделе **REAL WORLD** нажмите **Controller**. Если **Server Status - Stopped**, перейдите к следующему подэтапу. Если в **Preferences**, **Server Status - Disabled**, то вам необходимо включить внешний доступ следуя следующим инструкциям:

1) Выберите **Options > Preferences** в меню Packet Tracer;
2) Нажмите **Miscellaneous**;
3) В разделе **External Network Access** нажмите **Enable External Access** для REST-интерфейса сетевого контроллера API;
4) При необходимости закройте **Preferences** и нажмите **PT-Controller0 > Config**;
5) Слева в разделе **REAL WORLD** нажмите **Controller.**

Статус сервера теперь должен быть **Stopped**. Нажмите **Access Enabled**, чтобы включить его. Состояние сервера изменится на **Listening** на порту 58000. Если порт имеет другое значение, измените его на 58000. Это номер порта в скриптах Python

Шаг 3: Проверьте подключение к PT-Controller0 от имени администратора.

Убедитесь, что администратор может выполнить ping PT-Controller0. Если вы не можете выполнить ping, убедитесь, что ваша конфигурация соответствует спецификациям, указанным в предыдущем шаге.

Шаг 4: Зарегистрируйте нового пользователя и войдите в систему PT-Controller0.

Щелкните **Admin > Desktop > Web Browser**.

Введите IPv4-адрес **192.168.101.254** для доступа к настройке пользователя для **PT-Controller0**.

Введите **cisco** в поле Имя пользователя и **cisco123!** в полях Пароль и Подтверждение пароля, а затем нажмите **SETUP**.

Примечание: Вы можете использовать здесь любое имя пользователя и пароль. Для простоты мы рекомендуем использовать общие учетные данные, используемые в остальной части работы.

На экране входа пользователя в систему введите свои учетные данные и нажмите кнопку **LOG IN**.

Теперь вы вошли в панель управления **PT-Controller0**. На данном этапе может быть полезно развернуть окно, чтобы вы могли видеть весь интерфейс.

Часть 4: SDN контроллер для обнаружения топологии

В этой части вы настроите PT-Controller0 на использование протокола Cisco Discover Protocol (CDP) для автоматического обнаружения девяти сетевых устройств в вашей топологии. PT-Controller0 также обнаружит все пять хост-устройств, подключенных к сети

Шаг 1: Добавьте учетные данные для доступа ко всем сетевым устройствам в топологии.

В графическом интерфейсе **Network Controller** нажмите кнопку меню.

Выберите **Provisioning**. Отсюда вы можете вручную добавить сетевые устройства. Однако вы будете использовать CDP для автоматического обнаружения устройств.

Нажмите **CREDENTIALS** и затем нажмите **+ CREDENTIAL**, чтобы добавить новую учетную запись.

Для имени пользователя введите **cisco**, а для пароля введите **cisco123!** Оставьте **Enable Password** пустым. Для **Description** введите **admin credentials**, а затем нажмите **OKAY**.

Новые учетные данные CLI теперь хранятся на PT-Controller0 для использования в задачах автоматизации.

Шаг 2: Используйте CDP для обнаружения всех устройств в сети.

Нажмите **DISCOVERY**, а затем нажмите **+ DISCOVERY**, чтобы добавить новое открытие.

Для имени введите **SWL1**. Для IP-адреса введите **192.168.101.2**. Для CLI **Credential List** опустите список вниз и выберите **cisco - admin credentials**.

Нажмите **ADD**.

Теперь вы должны увидеть статус **In Progress**. Вы можете подождать, пока Packet Tracer завершит моделирование этого процесс или нажать кнопку **Fast Forward Time** в главном окне Topology, чтобы ускорить процесс.

Часть 5: SDN контроллер для сбора информации

В этой части вы будете использовать графический интерфейс PT-Controller0 для просмотра информации о сетевых устройствах и хост-устройствах в топологии. Вы просмотрите топологию, созданную контроллером, а затем проведете трассировку пути по сети.

Шаг 1: Просмотрите список обнаруженных сетевых устройств.

Нажмите NETWORK DEVICE. Теперь вы должны увидеть все девять сетевых устройств в списке.

Нажмите на значок шестеренки рядом с именем хоста любого устройства, чтобы просмотреть информацию, собранную в процессе обнаружения процессом обнаружения. Обратите внимание, что в списке указана версия программного обеспечения, а также множество другой подробной информации об устройстве.

Шаг 2: Просмотрите список всех обнаруженных хост-устройств.

Вернитесь на **Dashboard**. Щелкните меню рядом, затем выберите **Dashboard**. (Вы также можете просто щелкнуть по баннеру сетевого контроллера, чтобы вернуться на Dashboard из любого места).

На Dashboard'е вы увидите графики с количеством узлов, с которыми можно связаться по протоколу ping, и количеством сетевых устройств, которые находятся под управлением. Оба показателя должны быть равны 100%.

Вы также должны увидеть плитки для QoS, сетевого устройства и хоста. Нажмите на значок шестеренки для Host. Это приведет вас на вкладку **HOSTS** для **ASSURANCE**.

На этой странице вы можете просмотреть всю информацию о подключении уровней 2 и 3 для каждого хоста, а также сетевое устройство, к которому он подключен.

Нажмите на значок шестеренки рядом с любым узлом, чтобы просмотреть более подробную информацию.

Шаг 3: Просмотрите топологию, созданную PT-Controller0.

Перейдите на вкладку **TOPOLOGY**. Обратите внимание, что PT-Controller динамически создал ту же топологию, которую вы видите в главном окне Packet Tracer.

В этом окне вы можете щелкнуть любое сетевое устройство, чтобы просмотреть его подробную информацию.

Вы также можете щелкнуть и перетащить значки устройств, чтобы изменить топологию. Однако ваши изменения не будут сохранены, когда вы покинете рабочую область **TOPOLOGY**.

Шаг 4: Проследите путь от одного устройства к другому.

Перейдите на вкладку **PATH TRACE**.

Нажмите **+ PATH**, чтобы добавить новый путь.

Проследите путь от одного конца сети до другого. Например, можно ввести IP-адреса от PC1 до PC4. Затем нажмите **OKAY**.

Нажмите на новый путь, который был добавлен, чтобы начать трассировку пути.

Вы получите отчет Маршрут, в котором показаны все переходы от источника к пункту назначения. Обратите внимание, что в списке указана информация только об устройствах уровня 3. Коммутаторы показаны как устройства **UNKNOWN**. Это потому, что все они работают только на уровне 2.

Часть 6: SDN контроллер для настройки параметров сети

Основным преимуществом автоматизации сети с помощью контроллера является возможность конфигурировать глобальные сетевые настройки и политики для всех устройств, а затем передавать эту конфигурацию нажатием одной кнопки. В этой части вы настроим PT-Controller0 с сетевыми параметрами для DNS, NTP и Syslog. Затем вы передадите эту конфигурацию

на поддерживаемые сетевые устройства. Наконец, вы проверите и протестируете политику.

Шаг 1: Изучите конфигурацию сервера Example.

Нажмите **Example Server > Services**.

В разделе **SERVICES** щелкните **DNS**. Обратите внимание, что служба DNS включена и что есть одна запись для www.example.com.

В разделе **SERVICES** выберите **SYSLOG**. Обратите внимание, что служба Syslog включена.

В разделе **SERVICES** выберите **NTP**. Обратите внимание, что служба **NTP** включена.

Шаг 2: Настройте глобальную политику для DNS, SYSLOG и NTP.

Нажмите кнопку **Admin**. Если вы закрыли **Admin**, вам нужно будет открыть приложение **Web Browser** и повторно пройти аутентификацию на **PT-Controller0**.

Щелкните **меню > Policy**.

На вкладке **QOS** обратите внимание на наличие опций для настройки области действия и политики. В этом задании вы будете настраивать е **NETWORK SETTINGS**.

Нажмите **NETWORK SETTINGS > DNS**. Введите **example.com** в качестве доменного имени и **192.168.101.100** в качестве IP-адреса.

Нажмите **Save**.

Щелкните **NTP**. Введите **192.168.101.100** в качестве IP-адреса.

Нажмите **Save**.

Нажмите **SYSLOG**. Введите **192.168.101.100** в качестве IP-адреса.

Нажмите **Save**.

Снова нажмите **DNS, NTP** и **SYSLOG**, чтобы проверить правильность информации. Если нет, исправьте информацию сохраняя каждый раз.

Нажмите **PUSH CONFIG**.

Откроется диалоговое окно **Push All Network Settings**. Проверьте настройки и нажмите OKAY. На экране появится сообщение **“Saved Successfully”** появится короткое сообщение.

Шаг 3: Проверьте и протестируйте сетевые настройки, которые были переданы на устройства.

В нижней части окна **NETWORK SETTINGS** находится следующее:

Примечание: Эта функциональность поддерживается только на устройствах под управлением ОС IOS-XE и Switch 2960-24TT. Это означает, что для данной версии Packet Tracer ваши глобальные настройки применяются только к маршрутизаторам.

Щелкните любой из трех маршрутизаторов. R1 показан в следующем выводе.

Щелкните CLI.

Щелкните внутри окна и нажмите Enter, чтобы получить командную строку.

Войдите в привилегированный режим EXEC и проверьте настройки DNS.

```
R1> enable
R1# show run | begin ip domain
ip domain-name example.com
ip name-server 192.168.101.100
!
<вывод опущен>
R1#
```

Введите следующие команды для проверки настроек NTP. Время на R1 должно совпадать с вашим текущим временем. Packet Tracer может потребоваться некоторое время для распространения сообщений NTP. Вы можете нажать кнопку Fast Forward Time чтобы ускорить процесс.

```
R1# show ntp associations

address ref clock st when poll reach delay offset disp
*~192.168.101.100127.127.1.1 1 12 16 377 0.00 0.00 0.12
* sys.peer, # selected, + candidate, - outlyer, x falseticker, ~ configured
R1# show clock
15:30:54.268 UTC Thu Jun 11 2020
R1#
```

Введите следующую команду, чтобы убедиться в том, что ведение журнала настроено.

```
R1# show run | include logging
logging 192.168.101.100
R1#
```

Чтобы проверить ведение журнала, выключите интерфейс Serial0/1/0, а затем снова включите его.

```
R1# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
R1(config)# interface s0/1/0
R1(config-if)# shutdown
%LINK-5-CHANGED: Interface Serial0/1/0, changed state to administratively down
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/1/0, changed state to
down
15:36:37: %OSPF-5-ADJCHG: Process 1, Nbr 192.168.2.1 on Serial0/1/0 from FULL to
DOWN, Neighbor Down: Interface down or detached
R1(config-if)# no shutdown
%LINK-5-CHANGED: Interface Serial0/1/0, changed state to up
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/1/0, changed state to up
15:36:53: %OSPF-5-ADJCHG: Process 1, Nbr 192.168.2.1 on Serial0/1/0 from LOADING
to FULL, Loading Done
R1(config-if)# end
R1#
```

Нажмите **Example Server > Services > SYSLOG**. Вы должны увидеть, что те же сообщения syslog, которые вы видели на CLI, также записываются на сервер. Дважды щелкните любую из записей, чтобы просмотреть сообщения.

Лабораторная работа №8.8.3

Packet Tracer — Сравнение CLI и SDN контроллера управления сетью

Цель лабораторной работы:

Часть 1: Запуск виртуальной машины DEVASC

Часть 2: Проверка внешнего подключения к Packet Tracer

Часть 3: Запрос маркера аутентификации с помощью Postman

Часть 4: Отправка REST-запросов с помощью Postman

Часть 5: Отправка REST-запросов с помощью VS Code

Часть 6: Отправка REST-запросов внутри Packet Tracer

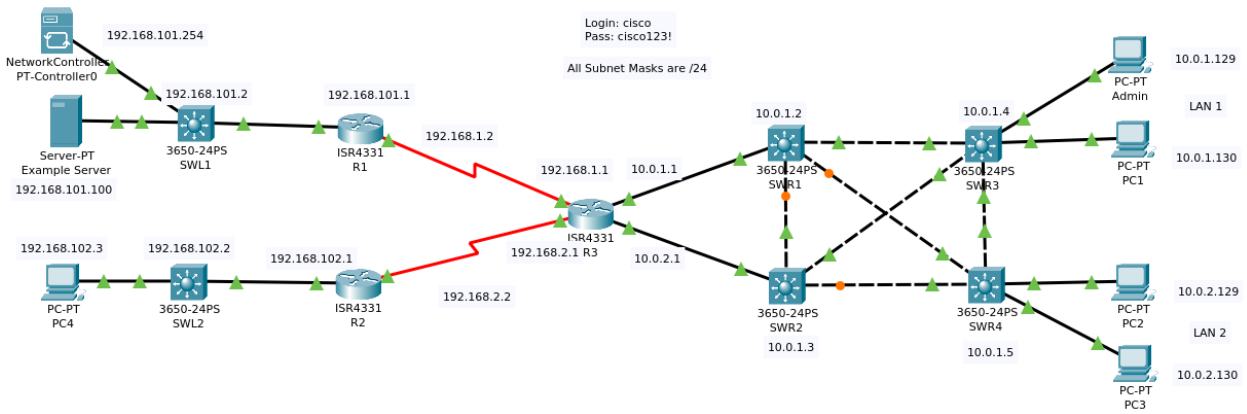
Необходимые ресурсы:

- 1 ПК
- Virtual Box или VMWare
- DEVASC виртуальная машина

Таблица адресации

Устройство	Интерфейс	IP Адрес
R1	G0/0/0	192.168.101.1
	S0/1/0	192.168.1.2
R2	G0/0/0	192.168.101.2
	S0/1/1	192.168.2.2
R3	G0/0/0	10.0.1.1
	G0/0/1	10.0.2.1
	S0/1/0	192.168.1.1
	S0/1/1	192.168.2.1
SWL1	VLAN 1	192.168.101.2
SWL2	VLAN 1	192.168.102.2
SWR1	VLAN 1	10.0.1.2
SWR2	VLAN 1	10.0.1.3
SWR3	VLAN 1	10.0.1.4
SWR4	VLAN 1	10.0.1.5
Admin	NIC	10.0.1.129
PC1	NIC	10.0.1.130
PC2	NIC	10.0.2.129
PC3	NIC	10.0.2.130
PC4	NIC	192.168.102.3
Example Server	NIC	192.168.101.100
PT-Controlle	NIC	192.168.101.254

Схема сети



Порядок выполнения работы

Часть 1: Запуск виртуальной машины DEVASC

Если вы еще не завершили **лабораторную работу - Установка лабораторной среды виртуальной машины**, серьёзно… на последней лабе. Если вы уже завершили эту лабораторную работу, запустите виртуальную машину **DEVASC**.

Часть 2: Проверка внешнего подключения к Packet Tracer

В этой части вы проверите, что Packet Tracer может быть доступен другим приложениям на виртуальной машине DEVASC. Это задание должно быть выполнено полностью в среде виртуальной машины DEVASC.

Шаг 1: Если вы ещё не сделали этого, откройте программу Packet Tracer.

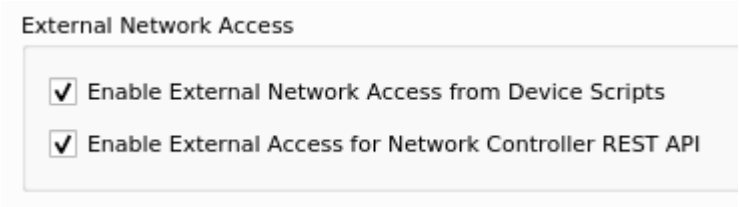
В виртуальной машине DEVASC VM откройте учебную программу своего курса в браузере Chromium.

Перейдите на страницу этого задания.

Загрузите и запустите файл Packet Tracer - Implement REST APIs with an SDN Controller.pka, связанный с этими инструкциями.

Шаг 2: Проверьте настройки Packet Tracer для внешнего доступа.

Нажмите **Options** > **Preferences** > **Miscellaneous**. Убедитесь, что в разделе **External Access,** установлен флажок **Enable External Access for Network Controller REST API**.



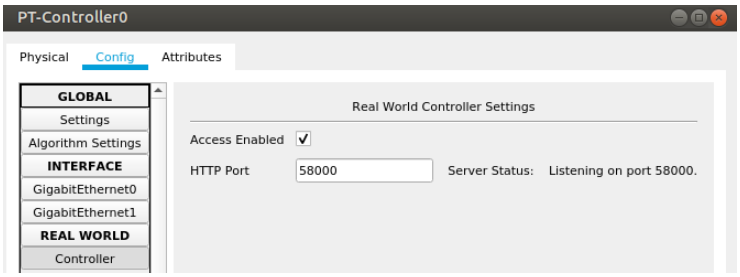
Закройте окно **Preferences**.

Нажмите **PT-Controller0** > **Config**.

Слева в разделе **REAL WORLD** нажмите **Controller**.

Установите флажок **Access Enabled** и обратите внимание на номер порта, который, скорее всего, равен 58000. Это номер порта, который вам

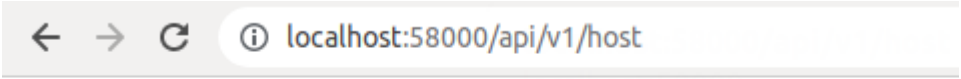
понадобится при внешнем доступе к трассировщику пакетов из Chromium, VS Code, и Postman позже в этой работе.



Шаг 3: Убедитесь, что вы можете получить доступ к Packet Tracer из другой программы на виртуальной машине DEVASC.

Откройте Chromium и перейдите на страницу http://localhost:58000/api/v1/host.

Вы получите следующий ответ. Этот шаг проверяет, что вы можете получить внешний доступ к Packet Tracer и PTController0. Обратите внимание, что для авторизации требуется токен. Вы получите его в следующей части.




```
{
  - response: {
        detail: "Security Authentication Failure",
        errorCode: "REST_API_EXTERNAL_ACCESS",
        message: "Ticket-based authorization: empty ticket."
    },
    version: "1.0"
}
```

```
{
    "response": {
        "detail": "Security Authentication Failure",
        "errorCode": "REST_API_EXTERNAL_ACCESS",
        "message": "Ticket-based authorization: empty ticket."
    },
    "version": "1.0"
}
```

Часть 3: Запрос маркера аутентификации с помощью Postman

В этой части вы изучите документацию REST API в Packet Tracer и воспользуетесь Postman для запроса токена аутентификации от PT-Controller0. Вы также можете сделать это в VS Code с помощью Python скрипта.

Шаг 1: Изучите документацию REST API для сетевого контроллера.

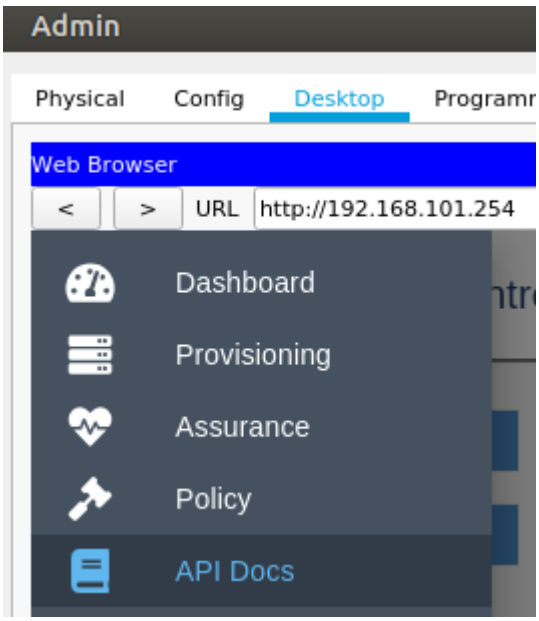
Чтобы просмотреть документацию REST API для PT-Controller0, выполните следующие действия:

Нажмите **Admin > Desktop > Web Browser**.

Введите **192.168.101.254**.

Войдите на PT-Controller0 под пользователем **cisco** и паролем **cisco123!**

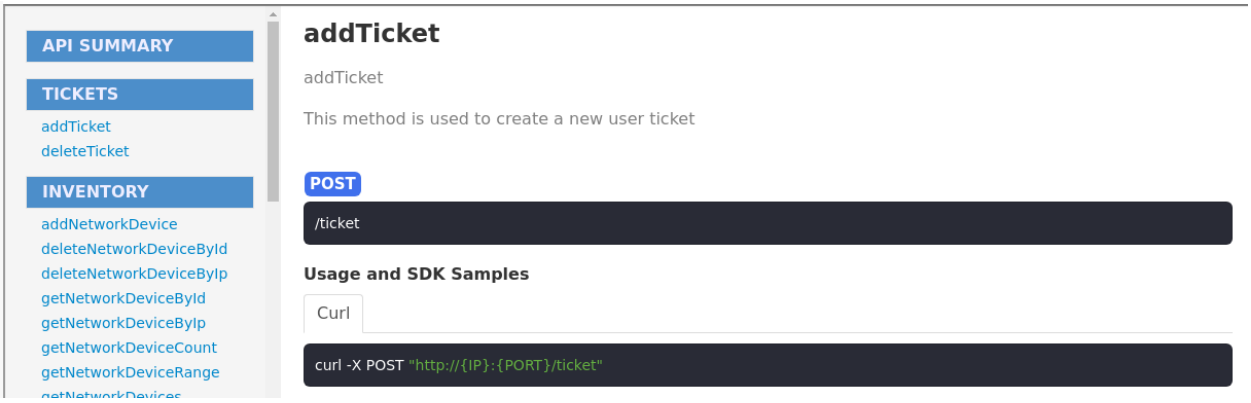
Нажмите на меню и выберите API Docs.



Вы также можете получить доступ к этой же документации из меню Help. (**Help > Contents**).

В навигационной панели слева прокрутите вниз примерно две трети и выберите **Network Controller API**. Здесь представлена та же документация, что и для PT-Controller0.

В документации API щелкните **addTicket**. Вы будете использовать эту документацию в следующем шаге.



Примечание: Некоторые функции REST API могут быть недоступны в текущей версии Packet Tracer.

Шаг 2: Создайте новый POST-запрос.

Изучив документацию по методу addTicket REST API Method, откройте Postman. В области запуска нажмите на знак плюс, чтобы создать новый **Untitled Request**.

Нажмите стрелку вниз и измените тип с GET на POST.

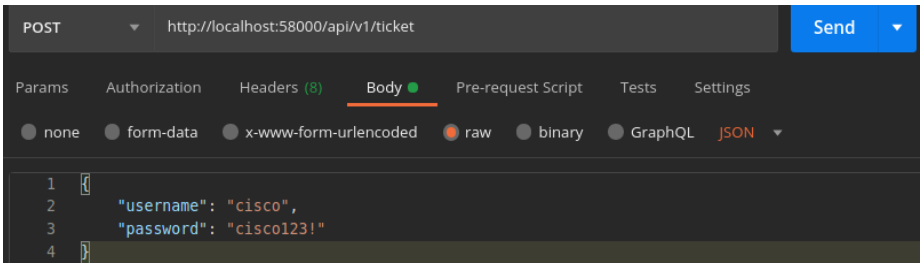
Введите URL **http://localhost:58000/api/v1/ticket**.

Ниже поля URL нажмите кнопку Тело. Измените тип на raw.

Нажмите стрелку вниз рядом с Text и измените его на JSON. Это изменение также установит HTTP-заголовок "Content-type" на "application/json", который требуется для этого вызова API.

Вставьте следующий объект JSON в поле Body. Убедитесь, что ваш код правильно отформатирован

```
{
    "username": "cisco",
    "password": "cisco123!"
}
```

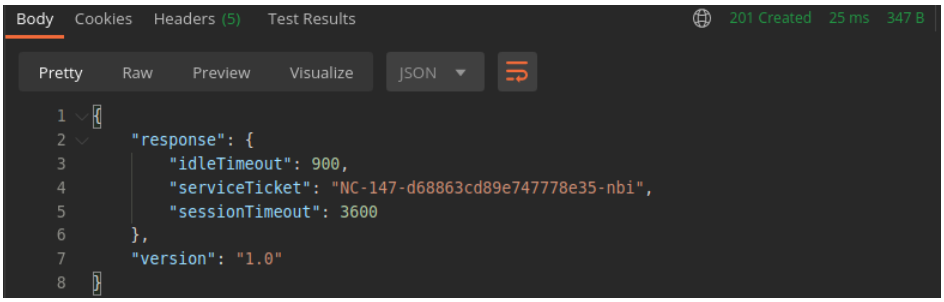


Шаг 3: Отправьте POST-запрос

Нажмите кнопку **Send**, чтобы отправить POST-запрос на PT-Controller0

Вы должны получить ответ, похожий на следующий. Однако **ваш_serviceTicket** будет фактическим значением.

```
{
    "response": {
        "idleTimeout": 900,
        "serviceTicket": "ваш_serviceTicket",
        "sessionTimeout": 3600
    },
    "version": "1.0"
}
```



Скопируйте значение serviceTicket без кавычек в текстовый файл для последующего использования.

Часть 4: Отправка REST-запросов с помощью Postman

В этой части вы будете использовать свой служебный билет для отправки трех REST-запросов на PT-Controller0.

Шаг 1: Создайте новый GET-запрос для всех сетевых устройств в сети.

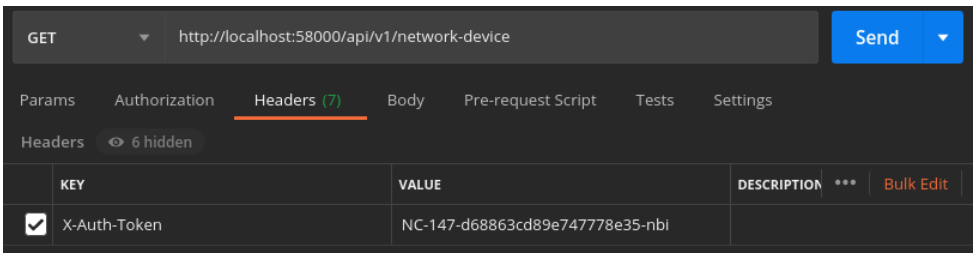
В Postman нажмите на знак плюс, чтобы создать новый **Untitled Request**.

Введите URL **http://localhost:58000/api/v1/network-device**.

Ниже поля URL нажмите **Headers**.

Под последним KEY щелкните поле **Key** и введите **X-Auth-Token**.

В поле **Value** введите значение **ваш_serviceTicket**.



Шаг 2: Отправьте GET-запрос.

Нажмите кнопку Send, чтобы отправить GET-запрос на PT-Controller0.

Вы должны получить ответ со списком данных, которые контроллер имеет для девяти сетевых устройств в сети. Ответ для первого устройства показан здесь.


```
{
        "response": [
                {
                        "collectionStatus": "Managed",
                        "connectedInterfaceName": [
                                "GigabitEthernet0/0/0",
                                "GigabitEthernet0",
                                "FastEthernet0"
                        ],
                        "connectedNetworkDeviceIpAddress": [
                                "192.168.101.1",
                                "192.168.101.254",
                                "192.168.101.100"
                        ],
                        "connectedNetworkDeviceName": [
                                "R1",
                                "NetworkController",
                                "Example Server"
                        ],
                        "errorDescription": "",
                        "globalCredentialId":                "53046ecc-88c3-49f6-9626-
                        ca8ab9db6725",
                        "hostname": "SWL1",
                        "id": "CAT1010BT47-uuid",
                        "interfaceCount": "29",
                        "inventoryStatusDetail": "Managed",
                        "lastUpdateTime": "6",
                        "lastUpdated": "2020-06-11 19:07:29",
                        "macAddress": "000C.CF42.2B11",
                        "managementIpAddress": "192.168.101.2",
                        "platformId": "3650",
                        "productId": "3650-24PS",
                        "reachabilityFailureReason": "",
                        "reachabilityStatus": "Reachable",
                        "serialNumber": "CAT1010BT47-",
                        "softwareVersion": "16.3.2",
                        "type": "MultiLayerSwitch",
                        "upTime": "1 hours, 6 minutes, 49 seconds"
                },
<вывод опущен>
        ],
        "version": "1.0"
```

}

Шаг 3: Дублируйте GET-запрос и измените его для всех хостов в сети.

В Postman щелкните правой кнопкой мыши вкладку GET-запроса хоста и выберите Duplicate Tab.

Вся информация в тикете будет одинаковой, за исключением URL. Просто измените network-device на host:

http://localhost:58000/api/v1/host.

Шаг 4: Отправьте GET-запрос.

Нажмите кнопку **Send**, чтобы отправить запрос **GET** на **PT-Controller0**.

Вы должны получить ответ со списком данных, которые контроллер имеет для шести хост-устройств в сети.

Ответ для первого устройства показан здесь.


```
{
        "response": [
                {
                        "connectedAPMacAddress": "",
                        "connectedAPName": "",
                        "connectedInterfaceName": "GigabitEthernet1/0/24",
                        "connectedNetworkDeviceIpAddress": "192.168.102.2",
                        "connectedNetworkDeviceName": "SWL2",
                        "hostIp": "192.168.102.3",
                        "hostMac": "00E0.F96C.155B",
                        "hostName": "PC4",
                        "hostType": "Pc",
                        "id": "PTT08108MO8-uuid",
                        "lastUpdated": "2020-06-11 13:00:56",
                        "pingStatus": "SUCCESS"
                },
<вывод опущен>
        ],
        "version": "1.0"
}
```


Шаг 5: Закройте Postman, чтобы освободить память в виртуальной машине DEVASC.

Часть 5: Отправка REST-запросов с помощью VS Code

В этой части вы будете использовать сценарий Python в VS Code для отправки тех же запросов API, которые вы отправляли в Postman.

Однако вы также будете использовать циклы Python for для разбора JSON и отображения только определенных пар ключ-значение.

Шаг 1: Используйте скрипт для запроса заявки на обслуживание.

Откройте VS Code.

Нажмите **File > Open Folder...** и перейдите в каталог devnet-src/ptna.

Нажмите OK.

Обратите внимание, что в панели EXPLORE слева отображаются три скрипта: **01_get-ticket.py, 02_get-networkdevice.py** и **03_get-host.py.** Просмотрите код

каждого из них. Обратите внимание, что программы для сетевых устройств и хостов требуют, чтобы вы заменили значение **ваш_serviceTicket** на значение, которое Packet Tracer дал вам, когда вы сделали его запрос. Запросите новый тикет на обслуживание, чтобы увидеть работу скрипта **01_get-ticket.py**.

```
devasc@labvm:~/labs/devnet-src/ptna$ python3 01_get-ticket.py
Ticket request status: 201
The service ticket number is: your_serviceTicket
devasc@labvm:~/labs/devnet-src/ptna$
```

Замените значение **your_serviceTicket** в файлах **02_get-network-device.py** и **03_get-host.py** на значение которое дал вам Packet Tracer.

Шаг 2: Используйте сценарий для запроса списка сетевых устройств.

Ранее в Postman вызов API сетевого устройства возвращал список всех девяти сетевых устройств и всю информацию, доступную для каждого устройства. Однако скрипт **02_get-network-device.py** печатает только значения ключей, которые интересуют программиста: имя хоста, **platformId** и **managementIpAddress**.

В окне терминала запустите сценарий **02_get-network-device.py**.

```
devasc@labvm:~/labs/devnet-src/ptna$ python3 02_get-network-device.py
Request status: 200
SWL1 3650 192.168.101.2
R1 ISR4300 192.168.1.2
R3 ISR4300 192.168.2.1
SWR1 3650 10.0.1.2
SWR2 3650 10.0.1.3
R2 ISR4300 192.168.2.2
SWL2 3650 192.168.102.2
SWR4 3650 10.0.1.5
SWR3 3650 10.0.1.4
devasc@labvm:~/labs/devnet-src/ptna$
```

Шаг 3: Используйте сценарий для запроса списка хост-устройств.

Аналогично, разработчик решил перечислить конкретную информацию для каждого из шести хост-устройств, подключенных к сети.

В окне терминала запустите скрипт **03_get-host.py**.

```
devasc@labvm:~/labs/devnet-src/ptna$ python3 03_get-host.py
Request status: 200
PC4 192.168.102.3 00E0.F96C.155B GigabitEthernet1/0/24
PC3 10.0.2.129 0004.9A42.C245 GigabitEthernet1/0/24
PC1 10.0.1.129 00E0.A330.3359 GigabitEthernet1/0/22
PC2 10.0.2.130 0060.47C1.A4DB GigabitEthernet1/0/23
Admin 10.0.1.130 0050.0FCE.B095 GigabitEthernet1/0/21
Example Server 192.168.101.100 000A.413D.D793 GigabitEthernet1/0/3
devasc@labvm:~/labs/devnet-src/ptna$
```

Часть 6: Отправка REST-запросов внутри Packet Tracer

В этой части вы будете использовать те же сценарии с одной небольшой правкой для отправки тех же API-запросов внутри Packet Tracer, которые вы отправляли из VS Code.

Шаг 1: Создание проекта в Packet Tracer

В программе Packet Tracer щелкните **Admin PC**.

Перейдите на вкладку **Programming**.

В настоящее время проекта нет. Нажмите кнопку **New**.

Введите REST APIs в качестве имени и выберите **Empty - Python** в качестве шаблона.

Нажмите кнопку **Create**.

Теперь проект **REST APIs (Python)** создан с пустым сценарием **main.py**.

Шаг 2: Измените скрипт для запуска внутри Packet Tracer.

Доступ от одного приложения к другому на одной и той же хост-машине требует, чтобы номер порта был указан в URL-адресе. Однако Packet Tracer имитирует реальную сеть. В реальном мире вы обычно не указывать номер порта при выполнении запросов API. Кроме того, вы будете использовать доменное имя или IP-адрес в URL.

В VS Code скопируйте код для файла 03_get-host.py

На вкладке **Admin > Programming** дважды щелкните по **main.py**, чтобы открыть его.

Вставьте код в скрипт **main.py**.

Измените api_url. Замените **localhost:58000/api/v1/host** на **192.168.101.254/api/v1/host**.

Изменения автоматически сохраняются. Нажмите кнопку Выполнить. Вывод Packet Tracer не совсем точно имитирует то, что вы видите в командной строке Linux. Однако вы должны увидеть похожие результаты, как показано ниже

```
Starting REST APIs (Python)...
('Request status: ', 200)
('PC4',     '\t',     '192.168.102.3',     '\t',     '00E0.F96C.155B',     '\t',
'GigabitEthernet1/0/24')
('PC3',     '\t',     '10.0.2.129',     '\t',     '0004.9A42.C245',     '\t',
'GigabitEthernet1/0/24')
('PC1',     '\t',     '10.0.1.129',     '\t',     '00E0.A330.3359',     '\t',
'GigabitEthernet1/0/22')
('PC2',     '\t',     '10.0.2.130',     '\t',     '0060.47C1.A4DB',     '\t',
'GigabitEthernet1/0/23')
('Admin',     '\t',     '10.0.1.130',     '\t',     '0050.0FCE.B095',     '\t',
'GigabitEthernet1/0/21')
('Example  Server',  '\t',  '192.168.101.100',  '\t',  '000A.413D.D793',  '\t',
'GigabitEthernet1/0/3')
REST APIs (Python) finished running.
```

Скопируйте и вставьте **02_get-network-device.py** в файл **main.py**. Измените URL и запустите его.

```
REST APIs (Python) finished running.
Starting REST APIs (Python)...
('Request status: ', 200)
('SWL1', '\t', '3650', '\t', '192.168.101.2')
('R1', '\t', 'ISR4300', '\t', '192.168.1.2')
('R3', '\t', 'ISR4300', '\t', '192.168.2.1')
('SWR1', '\t', '3650', '\t', '10.0.1.2')
('SWR2', '\t', '3650', '\t', '10.0.1.3')
```

```
('R2', '\t', 'ISR4300', '\t', '192.168.2.2')
('SWL2', '\t', '3650', '\t', '192.168.102.2')
('SWR4', '\t', '3650', '\t', '10.0.1.5')
('SWR3', '\t', '3650', '\t', '10.0.1.4')
REST APIs (Python) finished running
```

ПРИЛОЖЕНИЕ Б

Проектные работы.

В данном приложении рассматриваются пример проектных работ. Для каждой работы представлены описание, задачи, процедура возможного выполнения и порядок предоставления отчета. Весь процесс выполнения сопровождается четкими указаниями, это позволяет успешнее воспринимать материалы.

Помимо приобретения отличных технических навыков, вам также важно развить другие навыки, которые можно использовать на рабочем месте. Одним из способов достижения этой цели являются учебные проекты и мероприятия, которые имитируют навыки и знания, необходимые для участия в реальной работе.

В этом приложении используется проектная методология для развития технических навыков и "soft" навыков в увлекательной, я надеюсь, и содержательной форме. Проект состоит из пяти проектных работ, которые вы будете выполнять в командах. Вы создадите свою рабочую команду, организуете свои усилия, создадите и протестируете программные приложения. Показав свою работу другим и обеспечивать обратную связь с другими командами и членами команд.

Студенты, которые развивают свои навыки обучения (учатся учиться), гораздо лучше подготовлены к сегодняшней все более сложной рабочей среде, чем те, кто этого не делает. Умение учиться — это один из "навыков 21 века" к которым также относятся:

- Креативность и инновации;
- Критическое мышление и решение проблем;
- Коммуникация;
- Сотрудничество.

Компетенции, которые требуются на новом рынке труда, быстро меняются. Старый способ иметь высококвалифицированных специалистов в одной конкретной области, или другая крайность — иметь специалиста широкого профиля без глубоких знаний в какой-либо специализации, больше не эффективен. Новые рабочие места в современных организациях требуют многопрофильных навыков, которые обеспечивают сочетание двух предыдущих типов специалистов.

Предполагается, что во время проектной деятельности студенты будут работать в команде для совместной работы:

- Оценить проблему и требования клиента;
- Провести первоначальное исследование;
- Разработать решение;
- Создайте прототип;
- Протестируйте, оцените и измените решение;

- Оценивать, повторять и улучшать решение;
- Проведите заключительную презентацию своего решения и проанализируйте полученные знания.

План действий для этих проектов может быть разным, включая тип отрасли, задачи, которые необходимо решить, и роли команды. В этом проекте три сценария; однако ваш преподаватель может выбрать другой сценарий или позволить командам создать свои собственные сценарии

Результаты обучения

Технические навыки:

- Использование новейших концепции, навыков проектирования и разработки программного обеспечения для создания приложения.
- Практика удаленного сотрудничества с другими членами команды, используя коммуникационное программное обеспечение и систему контроля версий.
- Создание систему для интеграции и развертывания новых функций и улучшений итеративным способом.
- Итеративно разрабатывать решение с помощью методов быстрого создания прототипов.
- Создание процессов для развертывания приложений в автоматизированной среде.
- Создание кода, используя передовые методы разработки программного обеспечения и проектирования.
- Использование сетевого программирования и автоматизации для удовлетворения требований заказчика.

Soft навыки:

- Применение навыков критического мышления и решения проблем в реалистичной рабочей обстановке.
- Использование методов создания идей, такие как мозговой штурм.
- Использование средств презентации для предложения деталей проекта.
- Работа вместе в междисциплинарных командах для достижения командных целей.

Другие значимые результаты:

- Создание группы продуктов, о которых можно рассказать потенциальным работодателям.
- Развитие human networking в сфере IT.

Проектная деятельность №1

Формирование команды

Задачи проектной работы:

- Выбор ведущего и записывающего.
- Проведите упражнение "растопить лед".
- Обсудите эффективную работу в команде.
- Выберите роль для каждого члена команды.

Порядок выполнения работы:

Часть 1: Выбор ведущего и записывающего

Существует несколько методов формирования команд. Несмотря на некоторые различия, большинство из них сходятся в том, что первый шаг в формировании успешной команды включает в себя период знакомства.

Для начала вы выберете двух человек, которые будут помогать в процессе формирования команды. У этих людей будут только эти роли для данного мероприятия. Позже вы выберете новые роли как часть agile-команды.

Выберите две роли для формирования команды:

- Ведущий, который будет следить за тем, чтобы у каждого была возможность высказаться и внести свой вклад в этот процесс, и чтобы ни один человек не доминировал в обсуждении.
- Записывающий, который будет делать записи и обобщать этот процесс в рубрике.

Уделите несколько минут, чтобы всей командой просмотреть рубрику проекта DEVASC для этой деятельности. Рубрики будут использоваться на протяжении всего проекта, чтобы помочь вам записать свой процесс и подготовиться к презентации. Записывающий будет фиксировать эту информацию в листе рубрик для этой деятельности.

Роль ведущего заключается в том, чтобы руководить командой во время следующих частей мероприятия:

- Задавать вопросы для "растопки льда";
- Обсуждение требований проекта;
- Обсуждение работы в команде;
- Подчеркивание навыков и опыта каждого участника.

Примечание: *Ведущий и Записывающий должны участвовать в работе вместе с другими членами команды.*

Часть 2: Упражнение "Растопка-льда".

В качестве "растопки льда", ведущий попросит каждого из участников представиться и выбрать вопрос, например:

- Какое ваше любимое хобби?
- Что вы больше всего любите делать в одиночестве?
- Какой первый концерт вы посетили?
- Какие безумные занятия вы мечтаете когда-нибудь попробовать?
- Каково ваше представление о веселье?

Рубрика/Задание:

- Выберете название команды и запишите имена членов команды в таблицу.
- Запишите вопросы, которые были заданы, и ответы членов команды в рубрике.

Часть 3: Обсуждение эффективной работы в команде.

Ведущий должен прочитать команде вслух следующий параграф о командной работе:

Работа в команде обычно определяется как группа людей, работающих вместе для достижения общей цели. Более точное определение звучит следующим образом:

"Группа людей работает вместе слаженно, для достижения общей цели, создавая позитивную рабочую атмосферу и поддерживая друг друга, чтобы объединить индивидуальные сильные стороны и различия для повышения эффективности работы команды". - Статья: Определение командной работы, Счастливый менеджер

Пожалуйста, уделите время чтению приведенного ниже списка и подумайте о том, как вы будете участвовать в проекте в качестве члена команды. Обсудите, что каждый из этих пунктов означает в контексте ежедневного взаимодействия проектных команд.

Успешные команды разделяют следующие установки:

- Готовность помогать друг другу
- Разделять общее видение
- Позитивное отношение
- Поддержка и поощрение
- Достижение консенсуса
- Открытые коммуникации
- Эффективное разрешение конфликтов

Ведущий должен попросить каждого члена команды сказать, что для него значит работа в команде.

Часть 4: Выделите навыки и опыт каждого члена команды

Как член команды (включая Ведущего и Записывающего) вы должны найти роль, которая подходит именно вам. Кратко расскажите о своих навыках, включая технические и "soft" скиллы. Успех любого проекта зависит как от хорошей организации и координации, так и от сочетания технических знаний и навыков членов команды.

- Над чем вам нравится работать?
- О чем вам нравится узнавать?

Рубрика/Задание: Для каждого члена команды кратко опишите его навыки и интересы.

Шаблон отчёт:

Название команды: _____

Команда и роли:

	Имя	Навыки, Интересы
Ведущий		
Записывающий		
Член команды		
Член команды		

Вопросы с целью “растопить лёд”:

Член команды	Вопрос	Ответ

Проектная деятельность №2

Формирование Agile команды

Задачи проектной работы:

- Выбор проекта.
- Создание Agile команды.
- Распределение проектных ролей.
- Выберите каналы связи, которые будут способствовать работе команды.

Быть в сфере технологий сегодня означает совместную и эффективную работу в составе многофункциональной команды. Чтобы подготовиться к постоянным и ускоряющимся темпам изменений, вам необходимо:

- Критически подходить к вопросам, творчески решать проблемы;
- Работать в сотрудничестве;
- Четко общаться в различных медиа-каналах;
- Быстро осваивать постоянно меняющиеся технологии;
- Справляться с потоком информации.

Быстрые изменения в нашем мире требуют от вас гибкости, инициативы, лидерства, когда это необходимо, и производства чего-то нового и полезного.

Предыстория

Вас наняли на должность младшего инженера по автоматизации, чтобы помочь вашей компании автоматизировать ее ИТ-инфраструктуру. Вы ежедневно работаете с кодом автоматизации на языке Python прямо на своем компьютере. Однако ваша команда расширяется за счет новых членов, и теперь вам всем нужен способ совместной работы над кодом. Ваш руководитель требует, чтобы все члены команды перенесли код со своих компьютеров на GitHub и сотрудничали с командой через платформу GitHub.

Часть 1: Выбор ролей в команде Agile

Теперь ваша команда создаст формальную Agile команду. У каждого члена вашей будет своя роль в этом проекте. Конкретные задачи, которые каждый член команды будет выполнять в ходе проекта, будут определены позже.

Роли команды Agile могут различаться в зависимости от:

- Размера команды;
- Характера задачи;
- Передовой практики и политики компании.

Scrum-лидер

Как минимум, в agile-команде есть "scrum-лидер". Scrum -лидер помогает получить все необходимые ресурсы и берет на себя инициативу в решении любых вопросов или проблем, которые могут возникнуть у команды. В этой роли используются "soft навыки" управления проектами и коммуникации.

На этом этапе ваша команда выберет scrum-лидера. Обязанности scrum-лидера включают:

1. Устранение любых препятствий и проблем, которые могут возникнуть.

2. Создание среды, в которой команда может эффективно работать и общаться.
3. Решение вопросов, связанных с динамикой команды, для обеспечения поддержки и совместной работы членов команды.
4. Защита команды от внешних помех и отвлечений и выполнение роли посредника между командой и владельцем проекта (преподавателем).
5. Обеспечение документирования информации, требуемой рубрикой. (Фактическое документирование может осуществляться регистратором).

Рубрика/Задание:

- Запишите имя вашего scrum-лидера.
- Как ваша команда выбрала scrum-лидера?

Примечание: *Ваш преподаватель может выбрать сменного scrum-лидер.*

Фиксатор

Выберите кого-нибудь в качестве фиксатора. Этот человек будет отвечать за ввод и ведение рубрик проекта. Другие обязанности регистратора включают:

- Запись всех дней и времени встреч;
- Совместная работа с scrum-лидером для обеспечения точности и полноты информации в рубриках.

Примечание: *В небольшой команде (менее четырех человек) scrum-лидер может также взять на себя роль регистратора.*

Прочие роли

Провидите исследование, найдя в Интернете "роли agile-команды" и обсудите в команде, нужны ли еще какие-либо роли на данном этапе. Ваша команда всегда может вносить изменения и коррективы в ходе проекта.

Часть 2: Командные коммуникации

На последнем этапе этой работы вы решаете, как ваша команда будет общаться и сотрудничать во время этого процесса. Обсудите некоторые из различных инструментов, которые доступны каждому, такие как:

- WebEx Teams или другая платформа для обмена мгновенными сообщениями, например, Slack или Discord;
- Webex Meetings или другая платформа для видеосвязи, например, MS Teams или Discord;
- Электронная почта;
- Запланированные встречи.

Задокументируйте методы, которые будет использовать ваша команда, и убедитесь, что все члены команды имеют доступ к этим инструментам.

Шаблон Отчёта:

Название Команды: _____

Scrum Лидер
Scrum Лидер: _____

Как ваша команда выбирала скрам-лидера?
_____
_____

Ввели ли вы какие-либо другие роли в команде agile?
_____
_____

Коммуникации в команде
Какие средства коммуникации решила использовать ваша команда и с какой целью?

Название	Цель

Проектная деятельность №3

Социальное программирование

В этом задании вы выполните следующие задачи:

- Выберите свой проект.
- Определите функции, которые вы обязуетесь реализовать в своем приложении.

Часть 1: Выберите проект приложения

У вашей команды есть выбор: либо внести усовершенствования в существующее приложение, либо создать новое приложение. Выберите один из вариантов, описанных ниже.

Ваша команда получила определенный бюджет и ограничение по времени. Бюджетом управляет владелец продукта, который выделил финансирование для вашей команды и необходимые ресурсы. Ваш менеджер также запланировал определенное количество спринтов для завершения работы над этими функциями.

Вариант 1: Расширение возможностей – MapQuest

У вас на компьютере есть код из работы "Интеграция REST API в приложение на Python" (Лабораторная работа 4.9.2). Ваш руководитель попросил:

- Перенос кода с вашего компьютера в систему контроля версий для совместной работы;
- Усовершенствования функций.

Ваш руководитель получил запрос от отдела маркетинга на добавление новых функций в приложение MapQuest API. Ваш руководитель поручил вашей команде сделать усовершенствование функций в этом приложении. Эти усовершенствования могут включать изменения пользовательского интерфейса, такие как улучшение форматирования вывода данных (с помощью цветов, таблиц и т.д.), предоставление пользователю дополнительных опций, таких как данные с использованием метрической системы или миль, или тип данных, которые вы предоставляете пользователю. Любые другие улучшения, которые вы считаете необходимыми, будут отличать вашу работу от работы других команд.

Ваш менеджер также хотел бы получить список других улучшений для будущего пересмотра. Эти элементы называются бэклог. Этот список будет использован для Проектной деятельности 4.

***Примечание**: Вы будете использовать финальный (окончательный) код из лабораторной работы "Интеграция REST API в приложение Python". В этой работе вы использовали API направления MapQuest для получения данных в формате JSON, разобрали данные и отформатировали их для пользователя.*

Вариант 2: Новая разработка - создание приложения для адреса IPv4/IPv6

Ваш руководитель получил запрос от инженерного отдела на создание прототипа нового приложения, которое будет предоставлять информацию об IP-адресах сетевым специалистам.

Найдите в Интернете REST API, которые извлекают текущий публичный IPv4-адрес и IPv6-адрес пользователя, например ipapi.co или ipsstack.com. Используя общедоступные API, создайте приложение Python, которое отображает и форматирует информацию о текущей публичной IP-адресации компьютера.

В зависимости от выбранного вами API, вы также можете получить информацию о геолокации, провайдере (ISP), ASN (Autonomous System Number) ISP и код страны.

Ваш руководитель также хотел бы получить список других усовершенствований для будущей редакции. Это так называемые резервные элементы. Этот список будет использован для Проектной деятельности 4.

Примечание: *Цели и конкретные задачи вашего проекта будут зависеть от возможностей, предоставляемых выбранным вами API IP-информации.*

Часть 2: Роли команды

Теперь, когда вы выбрали проект приложения, вы должны определить, какой вклад в проект может внести каждый член команды. Возможно, некоторые члены команды обладают большим опытом в определенных областях, чем другие. Это может стать отличной возможностью для перекрестного обучения. Помните, что в успешной команде люди поддерживают, поощряют и готовы помогать другим членам команды.

На этом этапе scrum-лидер будет способствовать процессу:

- Выявление всех членов команды, которые могут обладать знаниями, опытом или быть авторитетами в определенной области или теме, имеющей отношение к проекту.
- Разработка стратегии для выполнения задач.
- Создание плана проекта для выполнения задач в соответствии с графиком.
- Определение инструментов, необходимых для продвижения вперед.

Рубрика/Задание: Задокументируйте роли членов вашей команды, а также любые дополнительные знания и навыки, которые они могут привнести в этот проект.

Часть 3: Рабочая нагрузка и статус проекта

Обсудите объем работы по проекту и временные обязательства команды. У членов команды есть и другие обязанности, и определение ожиданий поможет избежать любых будущих конфликтов и обеспечить своевременное завершение проекта. Ваш менеджер сообщит вам количество времени, которое было выделено на выполнение ваших задач.

На протяжении всего проекта scrum-лидеру будет важно управлять индивидуальной рабочей нагрузкой и статусом каждого проекта. Во многих случаях эта роль может быть возложена на человека, который обладает опытом в этой области и чья единственная работа заключается в управлении проектом.

Скрам-лидер должен будет содействовать процессу:

- Ведение списка конкретных задач.

- По возможности разбитин задачи на более мелкие.
- Определение приоритетов для каждой задачи.
- Управление статусом каждой задачи и любым влиянием, которое она может оказать на завершение других задач и проекта.

Скрам-лидер может взять на себя роль менеджера проекта или поручить выполнение этих задач другому члену команды. Для этого можно использовать простое облачное приложение, такое как Google Sheets, Google Docs или Yandex Документы, позволяющее членам команды постоянно обновлять статус своих индивидуальных задач.

Рубрика/Задание: Дайте краткое описание стратегии вашей команды для завершения этого проекта.

Часть 4: Настройка GitHub для совместной работы

Как упоминалось ранее, ваш руководитель попросил, чтобы ваша команда использовала GitHub для совместной работы над кодом Python.
Использование GitHub.com:

- Создайте репозиторий
- Добавьте коллаборaторов
- Создавайте ветки для членов команды
- Работа с pull requests, code review, слияние и т.д.

Примечание*: Ваш преподаватель может попросить вас дать ссылку на репозиторий GitHub.*

Рубрика/Задание: Запишите ссылку на ваш репозиторий GitHub. Опишите, как GitHub использовался для создания ответвлений, добавления членов команды, выполнения запросов и т.д.

Часть 5: Разработка приложения

В этой части проектной деятельности будет завершено приложение. Постоянно обращайтесь к процессам и инструментам, обсуждавшимся ранее. Продолжайте рассматривать эти вопросы:

- Сохраняются ли в команде открытые коммуникации и позитивный настрой?
- Подбадривают ли члены команды и поддерживают ли они друг друга?
- Осуществляется ли мониторинг проекта с обновлением статуса каждой задачи?
- Есть ли конфликты или проблемы с рабочей нагрузкой?
- Достижимы ли технические цели?

Теперь ваша команда готова приступить к работе:

- Разработка конкретных задач
- Объединение кода команды
- Проверка работоспособности кода (обеспечение качества).

Кроме того, scrum-лидер несет ответственность за то, чтобы команда:

- Совместная работа
- Выполнение заданий по графику

Часть 6: Презентация

Создайте бизнесовую презентацию о выбранном вами проекте и вашей работе по его завершению.

Рубрика/Задание: Ваша презентация должна включать:

- Информацию о вашем приложении, в том числе о том, какие функции включила ваша команда.
- Причины, по которым ваша команда решила использовать именно эти функции в вашем приложении.
- Код приложения, включая комментарии и документацию. Ваши комментарии и документация должны быть достаточными для того, чтобы любая другая команда могла продолжить проект, если потребуется. Другая команда должна быть в состоянии понять приложение, ваши особенности и то, как продолжить работу над проектом.
- Демонстрация приложения
- Список будущих улучшений (бэклог)
- Точки рефлексии — с какими проблемами вы столкнулись во время работы над этим мероприятием, как вы нашли решения, чему научились и т.д.

Шаблон Отчёта:

Название Команды: _____

Выбор приложения:

Выберите приложение для социального кодирования, выбранное вашей командой:

- Вариант 1: Расширение возможностей кода Lab 4.9.2
- Вариант 2: Новое приложение - приложение адресов IPv4/IPv6

По каким причинам ваша команда выбрала этот вариант?

_____

_____

Опишите приложение вашей команды и его результаты. Каковы конкретные цели этого приложения?

_____
_____
_____
_____
_____
_____

Запишите роли и навыки членов вашей команды

Член команды	Роль/знания/навыки

Стратегия/план проекта
Дайте краткое описание стратегии вашей команды по завершению этого проекта.

_____
_____
_____
_____

Использование GitHub для совместной работы
Какова ссылка на ваш репозиторий GitHub?

_____

Опишите, как использовался GitHub:
Созданные ветки

_____
_____
_____

Добавление членов команды

_____
_____
_____

Выполнение процессов pull requests, code review, merge и т.д.

_____
_____
_____

Окончательные результаты
Презентация

Создайте презентацию о выбранном вами проекте. Ваша презентация должна включать:

- Информацию о вашем приложении, в том числе о том, какие функции включила ваша команда
- Причины, по которым ваша команда решила использовать именно эти функции в вашем приложении
- Код приложения, включая комментарии и документацию. Ваши комментарии и документация
- Должны быть достаточными для того, чтобы любая другая команда могла продолжить работу над проектом, если потребуется. Другая команда должна быть в состоянии понять приложение, ваши особенности и то, как продолжить работу над проектом.
- Демонстрация приложения
- Список будущих улучшений (бэклог)
- Точки рефлексии - с какими проблемами вы столкнулись во время работы над этим заданием, как вы нашли решения, чему научились и т.д.

Опубликуйте свою презентацию SlideShare https://www.slideshare.net

Проектная деятельность №4

Автоматизированное тестирование и развертывание программного обеспечения

Задачи проектной работы:

- Разработать процесс автоматизированного тестирования.
- Реализовать процесс автоматизированного тестирования.

Отдел маркетинга получил очень благоприятные отзывы и рецензии о вашем приложении. Ваш руководитель выделил финансирование и ресурсы для вашей команды, чтобы добавить в приложение некоторые функции, оставшиеся в прошлом.

Пока ваша команда продолжает работать над кодом, каждый член команды вносит свои изменения в общую базу кода приложения на GitHub. Несмотря на то, что вы стараетесь проводить ручную проверку кода для каждого коммита или запроса на внесение изменений, иногда изменения члена команды нарушают некоторые функциональные возможности приложения. Во многих случаях такие проблемы выявляются только после развертывания и запуска приложения. Это обычно приводит к длительным сессиям по устранению неполадок и замедляет разработку.

Чтобы устранить описанные выше проблемы и облегчить разработку, вашей команде было предложено использовать некоторые лучшие практики из методологии DevOps. Разработайте процесс тестирования, который будет

автоматически запускаться GitHub при каждом новом коммите в кодовой базе. Только если все тестовые случаи пройдены, зафиксированное изменение может быть объединено или использовано в дальнейшем. Ваш руководитель рекомендовал вам в дополнение к Jenkins изучить другие инструменты CI/CD, такие как GitHub Actions, CircleCI и т.д., которые могут помочь вам построить этот автоматизированный CI/CD pipeline.

- Какие функции ваша команда выбрала для реализации из вашего бэклога?
- Каковы конкретные цели этих функций?
- Почему были выбраны именно эти функции?
- Задокументируйте роли, знания и навыки членов вашей команды, если что-то изменилось в связи с этой новой проектной деятельностью.
- Предоставьте краткое описание стратегии вашей команды по завершению этого проекта, если что-то изменилось в связи с этой новой проектной деятельностью.

Окончательные результаты

На этом этапе ваш проект завершен. Ваша команда представит его своему руководителю:

- Презентация
- Командная деятельность и рефлексия

Презентация

Рубрика/Задание: Создайте презентацию о выбранном вами проекте. Ваша презентация должна включать:

- Код приложения с новыми возможностями
- Показать изменения, опубликованные на GitHub
- Предоставить используемые тестовые случаи
- Описать функциональность CI/CD конвейера
- Точки размышления - с какими проблемами вы столкнулись во время работы над этим заданием, как вы нашли решения, чему научились и т.д.

Командная деятельность и рефлексия

Достижение/Рубрика: Ваш руководитель заинтересован в том, чтобы знать, как все продолжают работать вместе как команда. Вот список вопросов от вашего руководителя:

- Что вам понравилось в работе в команде? Что получилось хорошо?
- С какими проблемами в команде вы столкнулись и как вы их решили?
- С какими техническими проблемами вы столкнулись и как вы их решили?

- Как каждый член команды отвечал за свою работу в отдельности и за работу команды в целом?
- Каков был процесс принятия решений в вашей команде?
- В целом, какова была динамика работы команды и какие уроки были извлечены?

Шаблон Отчёта:

Название Команды: _____

Опишите приложение вашей команды:
Какие функции ваша команда выбрала для реализации из вашего бэклога?

_____
_____
_____

Каковы конкретные цели этих функций?

_____
_____
_____

Почему были выбраны именно эти характеристики?

_____
_____
_____

Запишите роли и навыки членов вашей команды

Член команды	Роль/знания/навыки

Стратегия/план проекта
Дайте краткое описание стратегии вашей команды по завершению этого проекта

_____
_____
_____

Окончательные результаты
Презентация

Предоставьте следующие материалы (которые также будут частью вашей презентации):
Приложение и Git:

- Код приложения с новыми возможностями;
- Показать изменения, перенесенные на GitHub;
- Тестовые кейсы;
- Функциональность CI/CD pipeline'а

Вспомните, с какими проблемами вы столкнулись во время работы над этим проектом. Как вы находили решения и чему вы научились?

Командная деятельность и рефлексия
Доставляемый материал/рубрика: Ваш руководитель заинтересован в том, чтобы знать, как все работали вместе как команда.
Вот список вопросов от вашего менеджера:
Что вам понравилось в работе в команде? Что хорошо получалось?

_____
_____
_____

С какими проблемами в коллективе вы сталкивались и как вы их решали?

_____
_____
_____

С какими техническими проблемами вы столкнулись и как вы их решили?

_____
_____
_____

Как каждый член команды отчитывался за свою работу индивидуально и за работу команды в целом?

_____
_____
_____

Каков был процесс принятия решений вашей командой?

_____
_____
_____

В целом, какова была динамика работы команды и какие уроки были извлечены?

Проектная деятельность №5

Программирование и автоматизация сети

Задачи проектной работы:

- Разработка нового приложение для автоматизации сети.
- Использование моделирования данных с помощью YANG.
- Реализация автоматизации с помощью NETCONF.
- Автоматизация процесса с помощью Python.

В этом проекти вы будете работать над новым приложением. Ваша новая ИТ-команда состоит из разработчиков программного обеспечения, сетевых инженеров и специалистов по кибербезопасности, все они имеют опыт работы и соответствующие сертификаты, подтверждающие их компетентность, а как иначе). Ваша компания стремится:

- Более оперативно реагировать на потребности и требования клиентов
- Иметь более короткие циклы разработки (более оперативно реагировать на изменения, обновления, требования клиентов);
- Увеличить частоту развертывания;
- Иметь более надежные релизы;
- Более тесно увязать с бизнес-целями;
- Использовать автоматизацию везде, где это возможно.

Команда использует общую философию, культуру, дисциплину и набор целостных методологий для предоставления и поддержки продуктов и услуг для внешних и внутренних клиентов. Ваш руководитель хочет, чтобы ваша команда использовала подход DevOps, чтобы продемонстрировать способность команды внедрять необходимые инструменты в вашей сети.

На вашей первой встрече с руководителем ваша команда определила множество новых функций и усовершенствований, которые могут упростить работу сети. У вас есть возможность изучить новые проблемы, но ваша первоочередная задача - поддержка инженеров поддержки первого уровня (L1 SE), которые ежедневно получают звонки от клиентов. Клиенты обычно требуют каких-то изменений в сети. Одним из распространенных запросов является необходимость изменить описание интерфейса маршрутизатора.

Команда L1 SE не имеет доступа к управлению сетевой инфраструктурой. Они передают запросы инженерам сетевой команды L2, которые ежедневно тратят драгоценное время на простые изменения конфигурации вручную. Ваш руководитель хочет найти способ оптимизировать эти процессы и поручил вашей команде разработать систему, которая позволит команде L1

SE запускать некоторые из этих изменений с помощью простого чат-бота (или веб-интерфейса, консольного приложения и т.д.).

Поскольку конфигурирование является одним из требований, ваша команда решила использовать NETCONF. Ваша команда создаст автоматизированный процесс, который продемонстрирует, как инженеры поддержки L1 могут вносить обновления в сетевые устройства.

Оперативная задача — автоматизировать следующий процесс:

- Проверка текущей конфигурацию.
- Внесение трёхи изменений в конфигурацию.
- Проверка изменения.
- Проверка новой конфигурацию.
- Отправка уведомления группе WebEx SE Teams (L1 и L2 SE) с сообщением об обновлении.

Технические задания:

- Использовать моделирование данных с помощью YANG.
- Реализация автоматизации сети с помощью NETCONF.
- Автоматизация процесса с помощью Python.

Достижение/Рубрика:

- Какие изменения ваша команда решила реализовать в running-config?
- Почему были выбраны именно эти функции?
- Документируйте роли, знания и навыки членов вашей команды, если что-то изменилось в связи с этой новой проектной деятельностью.
- Предоставьте краткое описание стратегии вашей команды по завершению этого проекта, если что-то изменилось в связи с этой новой проектной деятельностью.

Окончательные результаты

На этом этапе ваш проект завершен. Ваша команда представит его своему руководителю:

- Презентация
- Командная деятельность и рефлексия

Презентация

Рубрика/Задание: Создайте презентацию. Ваша презентация должна включать:

- Код приложения
- Описание выбранной модели YANG
- Приведите пример вывода работающего кода, показывающего:
    - running-config до изменений
    - running-config после изменений

    - Сообщение, отправленное командам WebEx
- Точки рефлексии — с какими проблемами вы столкнулись во время работы над этим заданием, как вы нашли решения, чему научились и т.д.

Командная деятельность и рефлексия
Доставляемый материал/рубрика: Ваш руководитель заинтересован в том, чтобы знать, как все работали вместе как команда. Вот список вопросов от вашего менеджера:

- Что вам понравилось в работе в команде? Что получилось хорошо?
- С какими проблемами в команде вы столкнулись и как вы их решили?
- С какими техническими проблемами вы столкнулись и как вы их решили?
- Как каждый член команды отвечал за свою работу в отдельности и за работу команды в целом?
- Каков был процесс принятия решений в вашей команде?
- В целом, какова была динамика работы команды и какие уроки были извлечены?

Шаблон отчета:

Название Команды: _____

Опишите приложение вашей команды:
Какие изменения ваша команда решила внести в run-config?
_____
_____
_____
Почему были выбраны именно эти фичи?
_____
_____
_____

Запишите роли и навыки членов вашей команды

Член команды	Роль/знания/навыки

Стратегия/план проекта
Дайте краткое описание стратегии вашей команды по завершению этого проекта.

_____
_____
_____
_____

Окончательные результаты
Презентация
Предоставьте следующие доказательства (которые также будут частью вашей презентации):

- Код приложения
- Используемая модель YANG
- Пример вывода работающего кода, показывающий:
    - running-config до изменений
    - run-config после изменений
    - Сообщение, отправленное командам WebEx

Подумайте о том, с какими проблемами вы столкнулись во время работы над этим проектом. Как вы находили решения
и чему вы научились?

_____
_____
_____
_____

Командная деятельность и рефлексия
Доставляемый материал/рубрика: Ваш руководитель заинтересован в том, чтобы знать, как все работали вместе как команда.
Вот список вопросов от вашего менеджера:
Что вам понравилось в работе в команде? Что хорошо получалось?

_____
_____
_____

С какими проблемами в коллективе вы сталкивались и как вы их решали?

_____
_____
_____

С какими техническими проблемами вы столкнулись и как вы их решили?

_____
_____
_____

Как каждый член команды нес ответственность индивидуально и за команду в целом?

_____

_____

_____

Каков был процесс принятия решений вашей командой?

_____

_____

_____

В целом, какова была динамика работы команды и какие уроки были извлечены?

_____

_____

_____

ПРИЛОЖЕНИЕ В

Дополняющий теоритический материал.

В данном приложении представлен теоретический материал, дополняющий основную часть ВКР. Все темы, рассмотренные в пояснительной записке, расширены дополнительными параграфами, которые из соображений, не превзойти рекомендуемый размер работы, опубликован в этои приложении, давая дополнительную информацию о темах более широко, что позволяет успешнее воспринимать материалы.

Для приобретения отличных технических навыков, вам также важно иметь широкий кругозор. Одним из способов достижения этой цели обзорно ознакомится с технологиями, после чего приступить к их отработке на практе.

Приложение дополняет следующие используется проектная методология для развития технических навыков и "soft" навыков в увлекательной, я надеюсь, и содержательной форме. Проект состоит из пяти проектных работ, которые вы будете выполнять в командах. Вы создадите свою рабочую команду, организуете свои усилия, создадите и протестируете программные приложения. Показав свою работу другим и обеспечивать обратную связь с другими командами и членами команд.

Дополненная Глава 1
Жизненный цикл разработки программного обеспечения (SDLC)

Фаза Требований и Анализа

Цель фазы требований и анализа — ответить на несколько уровней вопросов. Они начинаются с изучения текущей ситуации, потребностей и ограничений заинтересованных сторон, существующей инфраструктуры и т.д. Они определяют, какую проблему (проблемы) должно решить новое программное обеспечение.

После того, как проблема определена, можно изучить более конкретные вопросы, чтобы определить, где и в каком контексте новое программное обеспечение будет жить.

Когда ответы на такие вопросы собраны, самое время приступить к изучению более точных требований, сосредоточившись на желаемых функциях и пользовательском опыте (UX).

Наконец, команда начинает оценивать варианты архитектуры для создания самого программного обеспечения. Для большинства корпоративных приложений это означает множество итераций определения требований к передней и задней части программного обеспечения. Вам также потребуется обеспечить точки интеграции для других приложений, а также сервисы для управления жизненным циклом.

После сбора требований команда анализирует результаты, чтобы определить следующее:

- Возможно ли разработать программное обеспечение в соответствии с этими требованиями, и можно ли это сделать в рамках бюджета?
- Существуют ли какие-либо риски для графика разработки, и если да, то какие?
- Как будет проводиться тестирование программного обеспечения?
- Когда и как будет осуществляться поставка программного обеспечения?

По завершении этой фазы классический метод водопада предполагает создание документа Software Requirement Specification (SRS), в котором излагаются требования к программному обеспечению и объем работ, и тщательное подтверждение этого с заинтересованными сторонами.

Фазы разработки и имплементации

Дизайн

На этапе проектирования в качестве исходных данных обычно берется документ SRS, подготовленный на этапе анализа требований. На этапе проектирования архитекторы и разработчики программного обеспечения проектируют программное обеспечение на основе предоставленного SRS.

По завершении этапа проектирования команда создает документы высокоуровневого проектирования (HLD) и низкоуровневого проектирования (LLD). HLD дает "вид с высоты 10 000 футов" на предлагаемое программное

обеспечение. Он описывает общую архитектуру, компоненты и их взаимосвязи, а также может содержать дополнительные детали. В документе LLD, основанном на документе HLD, более подробно описывается архитектура отдельных компонентов, протоколы, используемые для связи между ними, а также перечисляются необходимые классы и другие аспекты проекта.

Имплементация

На этапе имплементации (реализации) в качестве исходных данных обычно берутся HLD и LLD, полученные на этапе проектирования.

Фазу реализации часто называют фазой кодинга или разработки. На этом этапе разработчики берут проектную документацию и пишут код в соответствии с этим проектом. На этом этапе создаются все компоненты и модули, что делает реализацию самой длительной фазой жизненного цикла. Во время этой фазы инженеры по тестированию (QA) также пишут план тестирования.

По завершении фазы внедрения функциональный код, реализующий все требования заказчика, готов к тестированию.

Фазы Тестирования, Деплоя и Поддержки

Тестирование

На этапе тестирования в качестве исходных данных обычно используется программный код, полученный на этапе имплементации. На этом этапе инженеры по тестированию переносят этот код в специально подготовленную тестовую среду, с целью пройтись по подготовленному ими плану тестирования. План тестирования — это документ, содержащий список всех тестов, которые необходимо выполнить, чтобы охватить все функции и функциональность программного обеспечения, как указано в требованиях заказчика, иными словами это check-list или to-do. В дополнение к функциональному тестированию инженеры по тестированию также выполняют:

- Интеграционное тестирование;
- Перфоманс тестирование;
- Тестирование безопасности.

Если код не проходит тест, инженер по тестированию выявляет баг, ошибка в компьютерной программе, который передается разработчикам. После исправления бага инженеры по тестированию повторно тестируют программу. Этот обмен мнениями между инженерами по тестированию и разработчиками продолжается до тех пор, пока весь код не пройдет все тесты.

По завершении этапа тестирования теоретически высококачественное, без ошибок, рабочее программное обеспечение готово к производству. На практике такое случается редко. Разработчики научились тестировать более эффективно, встраивать тестирование в автоматизированные рабочие процессы и тестировать программное обеспечение на разных уровнях

детализации и абстракции: от мельчайших низкоуровневых определений функций до масштабных объединений компонентов. Они также узнали, что программное обеспечение никогда не бывает без ошибок, вместо этого его необходимо сделать наблюдаемым, протестировать в производстве и сделать устойчивым, чтобы оно оставалось доступным и работало адекватно, несмотря на проблемы.

Деплой

На этапе деплоя в качестве исходных данных используется программное обеспечение, полученное на этапе тестирования; программное обеспечение устанавливается в продакшен среде. Если проблем с развертыванием нет, менеджер продукта работает с архитекторами и квалифицированными инженерами, чтобы решить, готово ли программное обеспечение к выпуску.

В конце этапа развертывания конечная часть программного обеспечения передается клиентам и другим конечным пользователям.

Поддержка

На этапе поддержки (технического обслуживания или пост-продакшена) команда:

- Обеспечивает поддержку клиентов
- Исправляет ошибки, обнаруженные в производстве
- Работает над усовершенствованием программного обеспечения
- Собирает новые запросы от клиентов

По завершении этапа сопровождения команда готова к работе над следующей итерацией и версией программного обеспечения, что возвращает процесс к началу SDLC и фазе анализа требований.

Методы Agile

Бэклог

Роль владельца продукта заключается в создании бэклога. Этот бэклог состоит из всех функций программного обеспечения в виде списка приоритетов. Функции в бэклоге являются результатом фазы "Требования и анализ" и включают функции, которые не обязательно будут в ближайшем релизе. Новые функции могут быть добавлены в бэклог в любое время, а владелец продукта может изменить приоритеты бэклога на основе отзывов клиентов.

Пользовательские истории

Когда функция приближается к вершине списка приоритетов, она разбивается на более мелкие задачи, называемые пользовательскими историями. Каждая пользовательская история должна быть достаточно маленькой, чтобы одна команда могла завершить ее в течение одного спринта. Если она слишком велика для завершения в течение одного спринта, команда должна разбить ее на более мелкие части. Поскольку программное обеспечение должно быть готово к сдаче в конце каждого спринта, пользовательская история также должна соответствовать этим правилам.

История пользователя — это простое изложение того, что нужно пользователю (или роли) и почему. Предлагаемый шаблон для истории пользователя выглядит следующим образом:

Как <пользователь|роль>, я хотел бы <действие>, чтобы <значение|выгода>

Завершение пользовательской истории требует завершения всех этапов SDLC. Сама история пользователя уже должна иметь требования, определенные владельцем продукта, а команда, взявшаяся за эту историю пользователя, должна придумать дизайн для задачи, реализовать его и протестировать.

Scrum-команды

Scrum-команды — это кросс-функциональные, совместные, самоуправляемые и самодостаточные команды. В идеале в таких Scrum-группах не должно быть больше 10 человек, но они должны быть достаточно большими, чтобы закончить историю пользователя в течение спринта.

Каждый день каждая scrum-команда должна проводить ежедневный standup. Стендап — это собрание, которое должно длиться не более 15 минут и проходить в одно и то же время каждый день. Фактически, оно называется "standup", потому что в идеале оно должно быть достаточно коротким, чтобы команда могла выполнить его без необходимости садиться.

Цель ежедневного стендапа заключается в том, чтобы все члены команды были в курсе того, что каждый человек сделал с момента последнего стендапа, над чем они собираются работать до следующего стендапа и какие препятствия мешают им завершить свою задачу. Scrum-мастер проводит такие стендапы, и его задача - сообщать о препятствиях и/или помогать их устранять.

Lean разработка программного обеспечения

Устраняйте потери

Отходы — это все, что не добавляет ценности для клиентов. Однако определение ценности субъективно, поскольку ее определяет клиент. Устранение отходов — это самый фундаментальный принцип бережливого производства, из которого вытекают все остальные принципы.

Чтобы устранить отходы, вы должны понимать, что такое отходы. Отходы — это все, что не добавляет прямой ценности для клиента. Существует семь видов отходов при разработке программного обеспечения:

- Частично выполненная работа;
- Дополнительные процессы;
- Дополнительные функции;
- Переключение задач;
- Ожидание;
- Процесс;

– Дефекты.

Частично выполненная работа

Частично выполненная работа — это отходы, поскольку:

– Она не добавляет никакой ценности для клиента;
– Время и ресурсы, затраченные на выполнение этой работы, можно было бы использовать для чего-то, что представляет ценность для клиента;
– Работа обычно не поддерживается, поэтому со временем она устаревает.

Дополнительные процессы

Лишние процессы — это просто куча бумажной работы. В результате они являются пустой тратой по тем же причинам, что и частично выполненная работа.

Дополнительные функции

Если клиент не просил об этом, это не приносит ему пользы. Это может быть приятно иметь, но лучше использовать ресурсы для создания именно того, что нужно клиентам.

Переключение задач

Человеку нужно время, чтобы переключить свое внимание на другую задачу, и это время, потраченное на переключение контекста, является пустой тратой. Переключение задач тратит время ресурса (человека), поэтому назначение ресурса на несколько проектов является пустой тратой времени.

Ожидание

Многие люди согласятся с тем, что по определению ожидание — это большая трата времени. Таким образом, любой тип задержки — это пустая трата времени. Примерами задержки в разработке программного обеспечения являются задержки на следующих этапах:

– начало проекта;
– получение необходимых ресурсов (персонала);
– получение требований от заказчика;
– утверждения документации;
– получение ответов;
– принятие решений;
– внедрение;
– тестирование;

Процесс

Бережливая разработка программного обеспечения определяет процесс для двух вещей: людей и артефактов. Процесс для людей — это когда людям нужно физически дойти от своего стола до другого члена команды, чтобы задать вопрос, сотрудничать и так далее. Когда они перемещаются от своего стола, тратится не только время, необходимое для того, чтобы добраться до места назначения, но и время переключения задачи.

Процесс артефактов — это когда, простите за такую конструкцию, документы или код переходят от одного человека к другому. Чаще всего документ не содержит всей информации для следующего сотрудника, поэтому либо ему приходится собирать информацию заново, либо передача требует времени, что является потерей времени.

Дефекты

Незамеченные дефекты (иначе называемые ошибками) являются расточительством из-за их последствий. Дефект может вызвать эффект снежного кома в других функциях, поэтому время, затраченное на его отладку, является пустой тратой. Кроме того, для клиента ценность программного обеспечения снижается, когда он сталкивается с проблемами, поэтому функция в конечном итоге оказывается бесполезной.

Улучшение обучения с помощью коротких спринтов

Чтобы иметь возможность доработать программное обеспечение, необходимо часто проводить короткие итерации рабочего программного обеспечения. Благодаря большему количеству итераций:

- Разработчики учатся быстрее;
- Клиенты могут быстрее получить обратную связь;
- Функции могут быть скорректированы таким образом, чтобы они приносили клиентам больше пользы.

Принимайте решение как можно позже

Когда существует неопределенность, лучше отложить принятие решения до как можно более позднего этапа процесса, потому что лучше основывать решения на фактах, а не на мнениях или домыслах.

Кроме того, когда решение еще не принято, программное обеспечение создается гибким, чтобы учесть неопределенность. Такая гибкость позволяет разработчикам вносить изменения, когда решение будет принято — или в будущем, если требования изменятся.

Поставить как можно быстрее

Быстрее поставлять программное обеспечение:

- Позволяет клиентам получать обратную связь;
- Позволяет разработчикам повысить эффективность обучения;
- Предоставляет клиентам функции, которые им нужны сейчас;
- Не позволяет клиентам передумать;
- Заставляет всех быстрее принимать решения;
- Производит меньше отходов.

Вы заметите, что каждая из этих причин использует по крайней мере один из ранее обсуждавшихся принципов бережливого производства.

Наделите команду полномочиями

У каждого человека есть свои знания, поэтому позвольте им принимать решения в своей области. В сочетании с другими принципами, такими как

устранение отходов, несвоевременное принятие решений и быстрая доставка, у других не будет времени принимать решения за команду.

Встройте целостность

Целостность программного обеспечения — это когда программное обеспечение в точности соответствует потребностям клиента. Другой уровень целостности заключается в том, что программное обеспечение сохраняет свою полезность для клиента.

Оптимизируйте итоговый продукт

Хотя одним из принципов является расширение возможностей команды, каждый эксперт должен сделать шаг назад и увидеть общую картину. Программное обеспечение должно быть создано целостно. Ценность программного обеспечения пострадает, если каждый эксперт сосредоточится только на своей компетенции и не будет учитывать последствия своих решений для остальной части программного обеспечения.

Дополненная Глава 2

Пример паттерн Model-View-Controller (MVC)

Паттерн проектирования Model-View-Controller (MVC) иногда рассматривается как архитектурный паттерн проектирования представлен на рисунке 40. Его цель — упростить разработку приложений, которые зависят от графических пользовательских интерфейсов. MVC разграничивает код и ответственность на три различных компонента: модель, представление и контроллер. Каждый компонент взаимодействует друг с другом в одном направлении. Этот шаблон проектирования широко используется в пользовательских интерфейсах и веб-приложениях.

Рисунок 40 — Схема Model-View-Controller

Компоненты

- **Модель** — является структурой данных приложения и отвечает за управление данными, логикой и правилами приложения. Она получает входные данные от контроллера.
- **Представление** — это визуальное представление данных. Может быть несколько представлений одних и тех же данных.
- **Контроллер** — является посредником между моделью и представлением. Он принимает пользовательский ввод и манипулирует им, чтобы он соответствовал формату модели или представления.

Работа Model-View-Controller выглядит следующим образом:

1. Пользователь предоставляет входные данные;
2. Контроллер принимает входные данные и манипулирует ими;
3. Контроллер отправляет обработанные данные в модель;
4. Модель принимает обработанные данные, обрабатывает их и отправляет выбранные данные (в строгой форме MVC, через контроллер) в представление;
5. Представление принимает выбранные данные и отображает их пользователю;
6. Пользователь видит обновленные данные, полученные в результате его ввода.

Преимущество паттерна проектирования Model-View-Controller заключается в том, что каждый компонент может быть построен параллельно. Поскольку каждый компонент абстрагирован, единственная информация, которая нужна каждому компоненту, — это интерфейс ввода и вывода для двух других компонентов. Компонентам не нужно знать о реализации внутри других компонентов. Более того, поскольку каждый компонент зависит только от входных данных, которые он получает, компоненты можно использовать повторно до тех пор, пока другие компоненты предоставляют данные в соответствии с правильным интерфейсом.

Системы Контроля Версий

Типы Систем Контроля Версий

Контроль версий, также называемый системой контроля версий, контролем ревизий или контролем исходных текстов, — это способ управления изменениями в наборе файлов с целью сохранения истории этих изменений. Вспомните, сколько раз вы делали копию файла перед его изменением, на случай, если захотите вернуться к оригиналу. Контроль версий делает все это за вас.

Системы контроля версий хранят основной набор файлов и историю изменений в репозитории, также известном как репа. Чтобы внести изменения в файл, человек должен получить рабочую копию репозитория на свою

локальную систему. Рабочая копия — это личная копия файлов, в которую человек может вносить изменения, не затрагивая других. К преимуществам контроля версий относятся:

- Обеспечивает совместную работу — несколько человек могут одновременно работать над проектом (набором файлов), не отменяя изменений друг друга;
- Учет и видимость — вы знаете, кто, когда и почему внёс изменения.
- Работа в изолированном режиме — создание новых функций независимо друг от друга без влияния на существующее программное обеспечение;
- Безопасность — файлы могут быть отменены в случае ошибки.
- Работа в любом месте — файлы хранятся в репозитории, поэтому на любом устройстве может быть рабочая копия.

Типы систем контроля версий

Существует три типа систем контроля версий:

- Локальная;
- Централизованная;
- Распределенная.

Локальная система контроля версий

Как следует из названия, локальная система контроля версий (Local Version Control System сокр. LVCS) отслеживает файлы в пределах локальной системы. Локальная система контроля версий заменяет сценарий "сделать копию файла перед дальнейшим редактированием" (Рисунок 41). Основной задачей локальной системы контроля версий является возможность возврата к предыдущей версии. Этот тип контроля версий не предназначен для решения большинства перечисленных выше задач.

Локальные системы контроля версий используют простую базу данных для отслеживания всех изменений в файле. В большинстве случаев система хранит дельту между двумя версиями файла, а не сам файл. Когда пользователь хочет вернуть файл, дельта меняется на противоположную, чтобы получить требуемую версию.

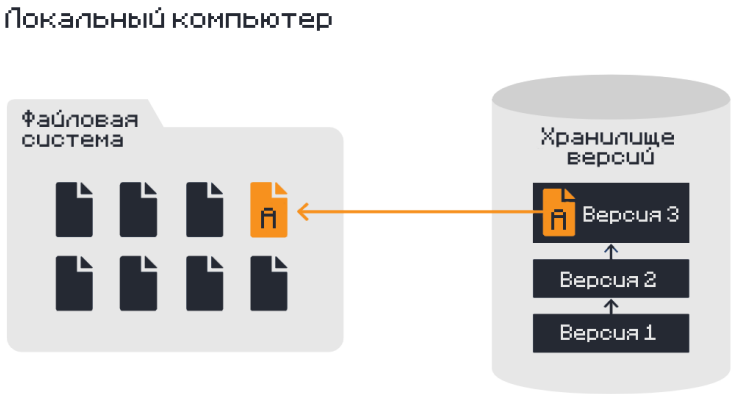


Рисунок 41 — Локальная система контроля версий

Централизованная система контроля версий

Централизованные системы контроля версий (Centralized Version Control Systems сокр. CVCS) используют модель сервер-клиент (Рисунок 42). Репозиторий (также известный как репа), который является единственной копией набора файлов и истории, хранится в централизованном месте, на сервере. Каждый раз, когда человек хочет внести изменения в файл, он должен сначала получить рабочую копию файла из хранилища в свою собственную систему — клиент.

В централизованной системе контроля версий только один человек может одновременно работать над определенным файлом. Чтобы обеспечить соблюдение этого ограничения, человек должен проверить файл, что блокирует файл и предотвращает его изменение кем-либо другим. Когда человек завершает внесение изменений, он должен проверить файл, что позволяет применить изменения человека к основной копии в репозитории, пометить новую версию и разблокировать файл для внесения изменений другими.

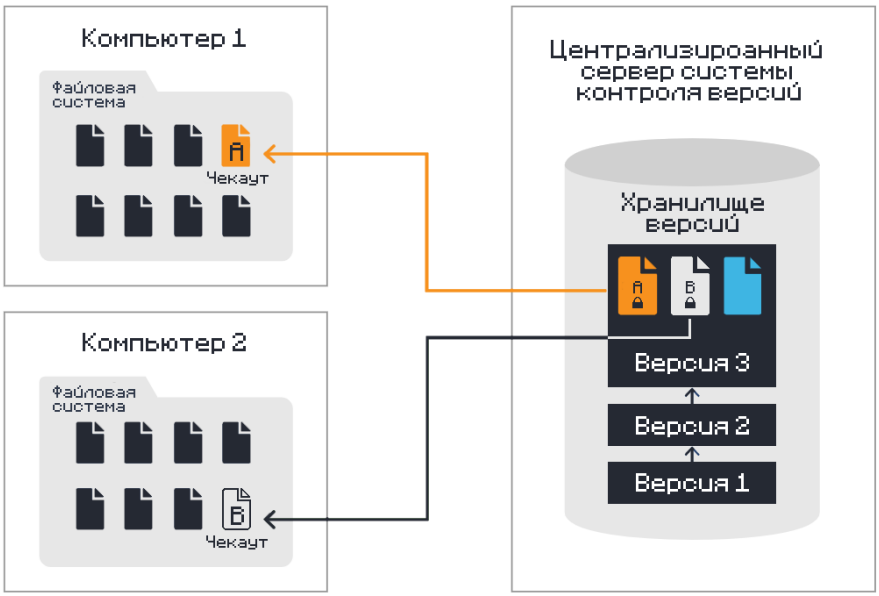


Рисунок 42 — Централизованная система контроля версий

Распределенная система контроля версий

Распределенная система контроля версий (Distributed Version Control System сокр. DVCS) представляет собой одноранговую модель. Репозиторий может храниться на клиентской системе, но обычно он хранится в службе хостинга репозиториев. Когда человек хочет внести изменения в файл, он должен сначала клонировать весь репозиторий на свою систему. Это включает в себя набор файлов, а также всю историю файлов. Преимущество этой модели заключается в том, что полный репозиторий будет находиться на нескольких системах и может быть использован для восстановления репозитория в службе хостинга репозиториев, если произойдет такое событие, как повреждение данных.

В распределенной системе контроля версий каждый человек может работать над любым файлом, даже в одно и то же время, поскольку локальный файл в рабочей копии — это то, что изменяется. В результате блокировка файла не требуется. Когда человек заканчивает вносить изменения, он передает файл в основное хранилище, которое находится в службе хостинга хранилищ, и система контроля версий обнаруживает любые конфликты между изменениями файлов.

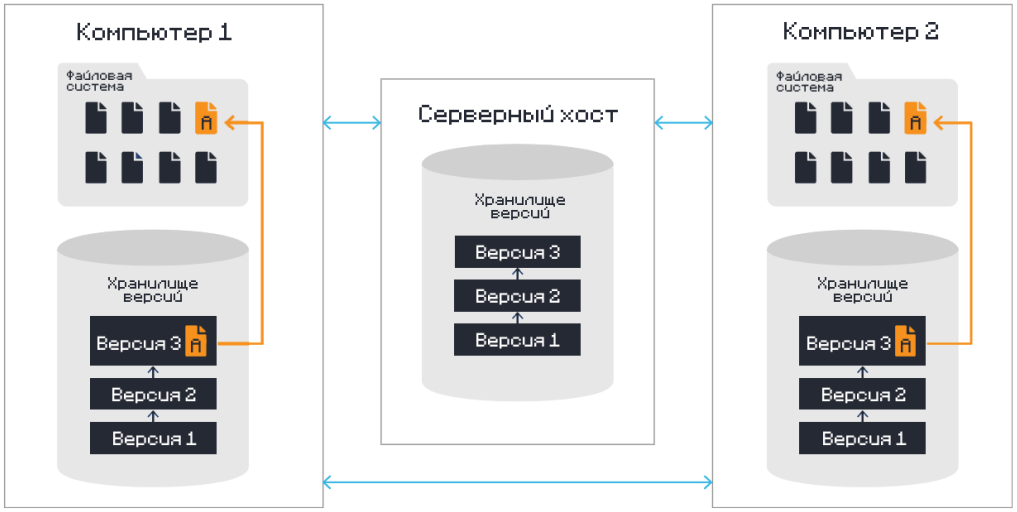


Рисунок 43 — Распределенная система контроля версий

Git

Git — это реализация распределенной системы контроля версий с открытым исходным кодом, которая в настоящее время является последней тенденцией в разработке программного обеспечения. Git:

- Легко осваивается;
- Может работать со всеми типами проектов, включая проекты крупных предприятий;
- Имеет высокую производительность;
- Создан для совместных проектов;
- Гибкий;
- Имеет небольшую площадь;
- Обладает всеми преимуществами распределенной системы контроля версий;
- Бесплатный;

На клиентской машине должен быть установлен клиент Git. Он доступен для MacOS, Windows и Linux/Unix. Хотя некоторые клиенты Git поставляются с базовым графическим интерфейсом, основное внимание в Git'е уделяется интерфейсу командной строки, о котором мы подробно расскажем позже.

Одно из ключевых отличий Git от других систем контроля версий заключается в том, что Git хранит данные в виде снимков (snapshots), а не

различий (разница между текущим файлом и предыдущей версией). Если файл не меняется, git использует ссылку на последний моментальный снимок в системе вместо того, чтобы делать новый идентичный моментальный снимок проиллюстрированно на рисуках 44 и 45.

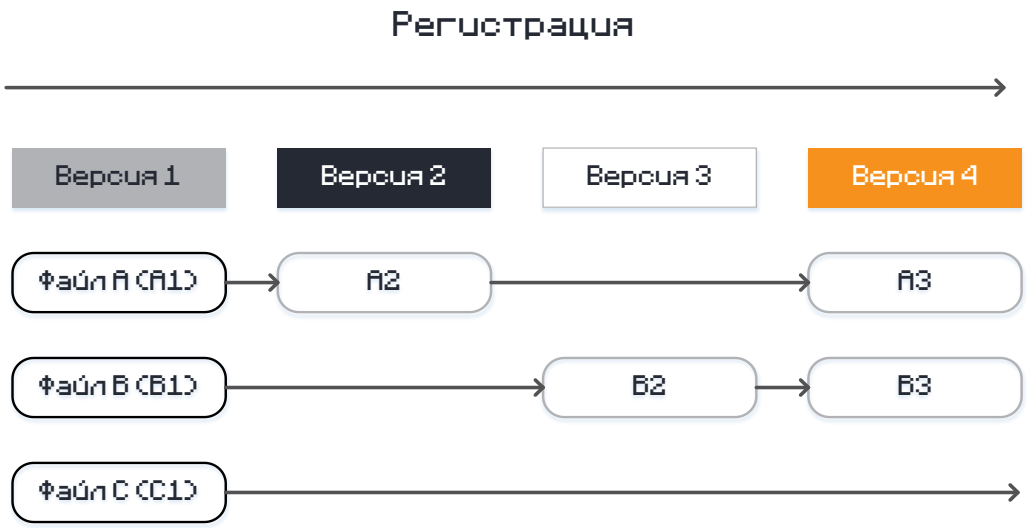


Рисунок 44 — Файловое представление

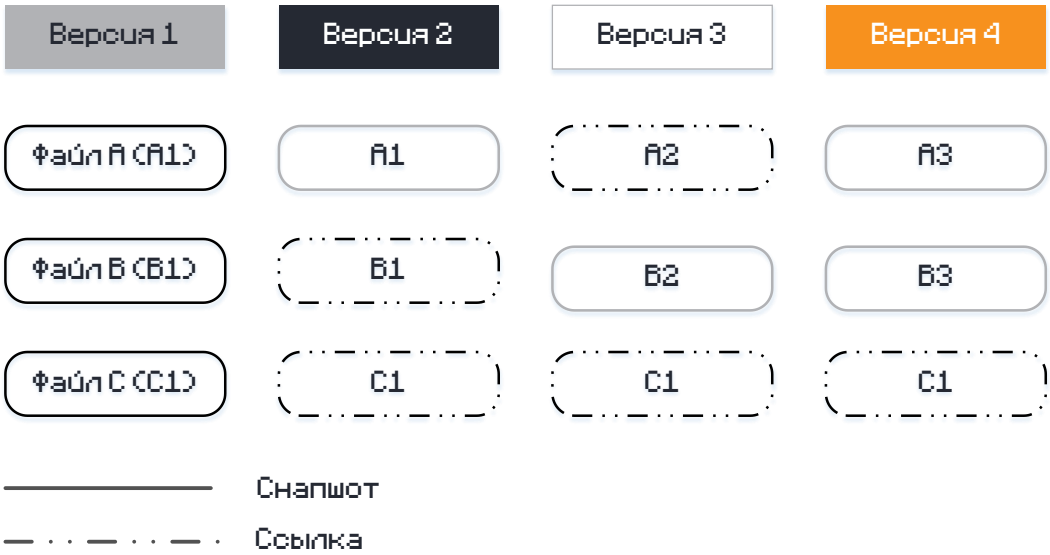


Рисунок 45 —Представление версий

Три этапа Git

В Git'е есть три “этапа” (Рисунок 46):

- репозиторий (каталог .git);
- рабочий каталог;
- область хранения.

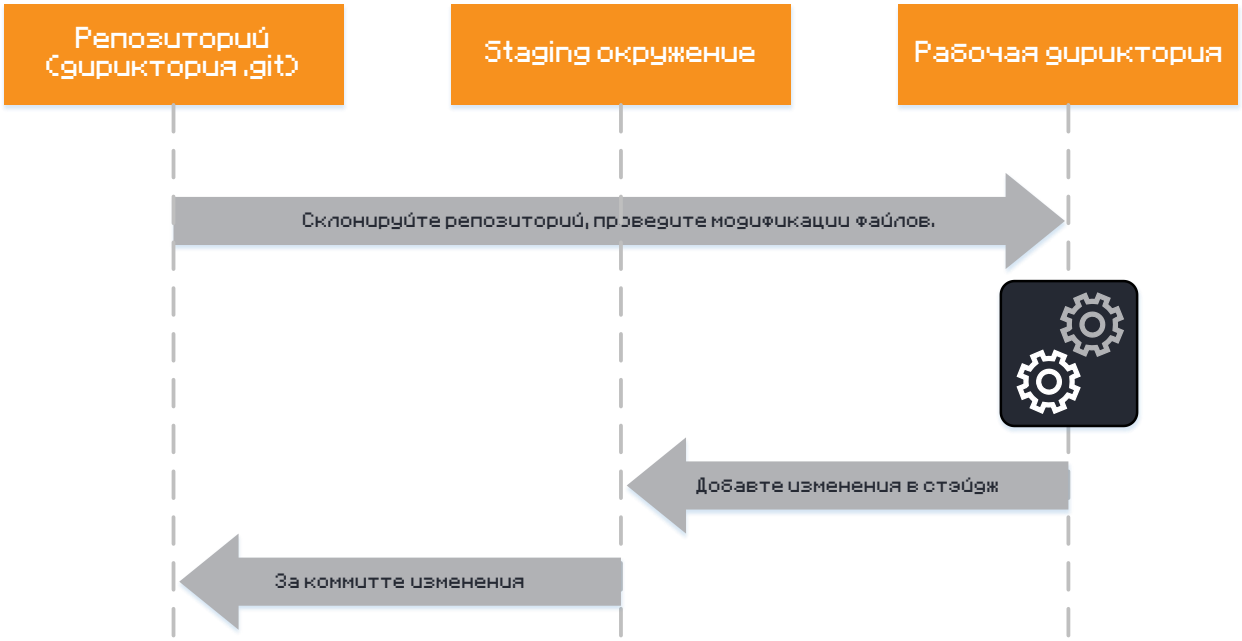


Рисунок 46 — “Этапы” .git

Репозиторий (каталог .git)

Поскольку Git является распределенной системой контроля версий, каждый клиент имеет полную копию репозитория. Когда проект становится репозиторием Git, создается скрытый каталог .git, который, по сути, и является репозиторием. В каталоге .git хранятся метаданные, такие как файлы (сжатые), коммиты и журналы (история коммитов).

Рабочий каталог

Рабочий каталог — это папка, видимая в файловой системе. Это копия файлов в хранилище. Эти файлы могут быть изменены, и изменения видны только пользователю клиента. Если файловая система клиента будет повреждена, эти изменения будут потеряны, но основное хранилище останется нетронутым.

Staging

В области постановки хранится информация о том, что пользователь хочет добавить/обновить/удалить в хранилище. Пользователю не обязательно добавлять все свои измененные файлы в stage/repo; он может выбрать определенные файлы. Хотя эта область называется областью, на самом деле это просто индексный файл, расположенный в каталоге **.git**.

Три состояния

Поскольку в Git'е есть три стадии, для файла Git'а существует три соответствующих состояния:

- **committed** — версия файла была сохранена в репозитории (каталог .git);
- **modified** — файл изменился, но не был добавлен в область постановки или зафиксирован в хранилище;
- **staged** — измененный файл готов к фиксации в хранилище.

Локальные и удаленные репозитории

В Git есть два типа репозиториев — локальные и удаленные.

Локальный репозиторий хранится в файловой системе клиентской машины, на которой выполняются команды git.

Удаленный репозиторий хранится не на клиентской машине, а где-нибудь в другом месте, обычно на сервере или в службе хостинга репозиториев. Удаленные репозитории опциональны и обычно используются, когда проект требует совместной работы команды с несколькими пользователями и клиентскими машинами.

Удаленные репозитории можно рассматривать как "централизованный" репозиторий для Git, но это не делает его CVCS. Удаленный репозиторий с Git продолжает оставаться DVCS, поскольку удаленный репозиторий будет содержать полный репозиторий, включающий код и историю файлов. Когда клиентская машина клонирует репозиторий, она получает полный репозиторий без необходимости его блокировки, как в CVCS.

После клонирования локального репозитория из удаленного репозитория или создания удаленного репозитория из локального репозитория эти два репозитория независимы друг от друга до тех пор, пока изменения содержимого не будут применены к другой ветке посредством ручного выполнения команды Git.

Что такое Ветвление?

Ветвление позволяет разработчикам работать над кодом независимо, не затрагивая основной код в репозитории. Когда создается репозиторий, код автоматически помещается в ветвь, называемую Master или толерантно Main. Ветвление представлино на рисунке 47

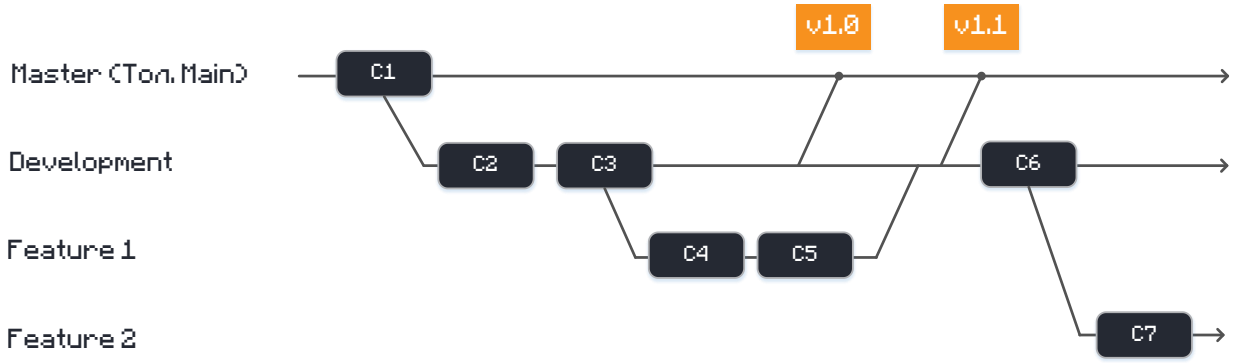


Рисунок 47 — Пример ветвления

У разработчиков может быть несколько веток независимых друг от друга. Ветвление позволяет:

- работать над фичами независимо друг от друга, получая при этом преимущества распределенной системы контроля версий;
- работать над несколькими фичами одновременно;
- экспериментировать с кодом;
- хранить production, development и feature код отдельно;

– Обеспечивать стабильность главной ветки Main.

Ветви могут быть локальными или удаленными, и их можно удалять. Локальные ветви облегчают опробование различных реализаций кода, поскольку ветвь можно использовать, если она успешна, и удалять, если нет. Слияние ветки с родительской веткой не является обязательным.

В отличие от других систем контроля версий, создание ветвей в Git'е облегчено, а переключение между ветвями происходит практически мгновенно. Хотя ветви часто визуально рисуются как отдельные пути, ветви Git'а по сути являются просто указателями на соответствующий коммит рис 48.

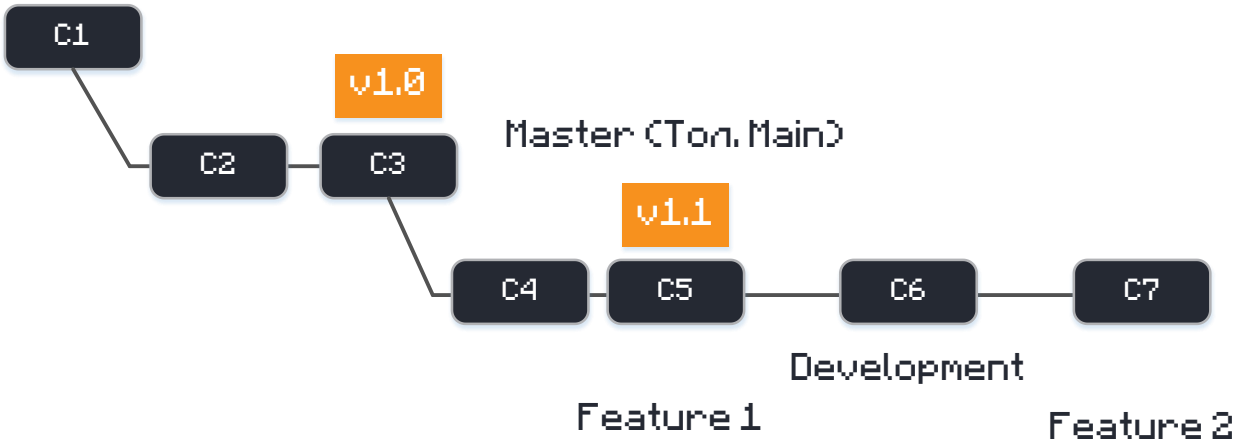


Рисунок 48 — Пример истории коммитов

Ветви похожи на развилку дорог, где они начинают с кода и истории в точке отклонения, а затем самостоятельно строят свой собственный путь с новыми коммитами. В результате ветви имеют свою собственную историю, область постановки и рабочий каталог. Когда пользователь переходит из одной ветви в другую, код в его рабочем каталоге и файлы в области постановки изменяются соответствующим образом, но каталоги репозитория (.git) остаются неизменными.

По возможности старайтесь использовать ветки, а не обновлять код непосредственно в мастер-ветке, чтобы предотвратить случайные обновления, ломающие код.

GitHub и другие поставщики

Работа с проектами с использованием Git часто ассоциируется с GitHub, но Git и GitHub — это не одно и то же. Git является реализацией распределенного контроля версий и предоставляет интерфейс командной строки. GitHub - это сервис, предоставляемый компанией Microsoft, который реализует услугу хостинга репозиториев с помощью Git.

В дополнение к функциональности распределенного контроля версий и управления исходным кодом Git, GitHub также предоставляет дополнительные возможности, такие как:

– code review;
– документация;
– project management;
– отслеживание багов;
– запрос фич.

GitHub развился для поддержки многих форм совместной разработки кода, включая:

– частные репозитории, видимые только определенным командам;
– проекты "социального программирования", которые являются публичными, но их участники могут оставаться анонимными;
– широкомасштабные проекты с открытым исходным кодом с большим количеством участников, число которых иногда исчисляется тысячами.

Чтобы дать возможность владельцам проектов управлять такими сильно различающимися сценариями, GitHub ввел понятие "**pull request**". Pull request — это способ формализации запроса соавтора на рассмотрение изменений, таких как новый код, правки существующего кода и т.д., в ветке соавтора с целью их включения в main или другие курируемые ветки проекта. Идиома pull request в настоящее время повсеместно реализована в службах хостинга Git.

GitHub — не единственный сервис хостинга репозиториев, использующий Git, другие включают GitLab и Bitbucket (Рисунок 49).

Рисунок 49 — Сервисы хостинга репозиториев

Команды Git

Настройка Git

После установки Git на клиентскую машину его необходимо настроить. Git предоставляет команду `git config` для получения и установки глобальных настроек Git'a или опций репозитория.

Чтобы настроить Git, используйте опцию `--global` для установки начальных глобальных настроек.

Команда: `git config --global` ключ значение

Использование опции `--global` приведет к записи в глобальный файл `~/.gitconfig`.

Чтобы каждый пользователь мог отвечать за свои изменения кода, при установке Git необходимо задать имя и email пользователя. Для этого используйте следующие команды:

```
$ git config --global user.name "<имя пользоввателя>"
$ git config --global user.email "<email>"
```

, где `<имя пользователя>` и `<email>` — имя пользователя и адрес электронной почты, соответственно.

Создание нового репозитория Git

Любой проект (папка) в локальной файловой системе пользователя может стать репозиторием Git. Git предоставляет команду `git init` для создания пустого Git-репозитория или превращения существующей папки в Git-репозиторий. Когда новый или существующий проект становится репозиторием Git, в папке проекта создается скрытый каталог `.git`. Помните, что каталог `.git` — это репозиторий, в котором хранятся метаданные, такие как сжатые файлы, история фиксации и область хранения. Помимо создания каталога .git, Git также создает основную ветвь.

Команда: `git init`

Чтобы сделать новый или существующий проект репозиторием Git, выполните следующую команду:

```
$ git init <дириктория проекта>
```

, где `<директория проекта>` — это абсолютный или относительный путь к новому или существующему проекту. Для нового Git-репозитория сначала будет создан каталог по указанному пути, а затем каталог `.git`.

Создание Git-репозитория не приводит к автоматическому отслеживанию файлов в проекте. Для отслеживания файлов необходимо явно добавить их во вновь созданный репозиторий. Подробнее о том, как добавлять файлы в репозиторий, будет рассказано позже.

Рисунок 50 — Git-репозиторий

Получение существующего репозитория Git

Команда: `git clone <репозиторий> [целевой каталог]`

где `<репозиторий>` — это местоположение репозитория, который нужно клонировать. Git поддерживает четыре основных транспортных протокола для доступа к `<репозиторию>`: Local, Secure Shell (SSH), Git и HTTP. Параметр `[целевой каталог]` является необязательным и представляет собой абсолютный или относительный путь к месту хранения клонированных файлов. Если вы не укажете каталог проекта, git скопирует репозиторий в то место, где вы выполнили команду.

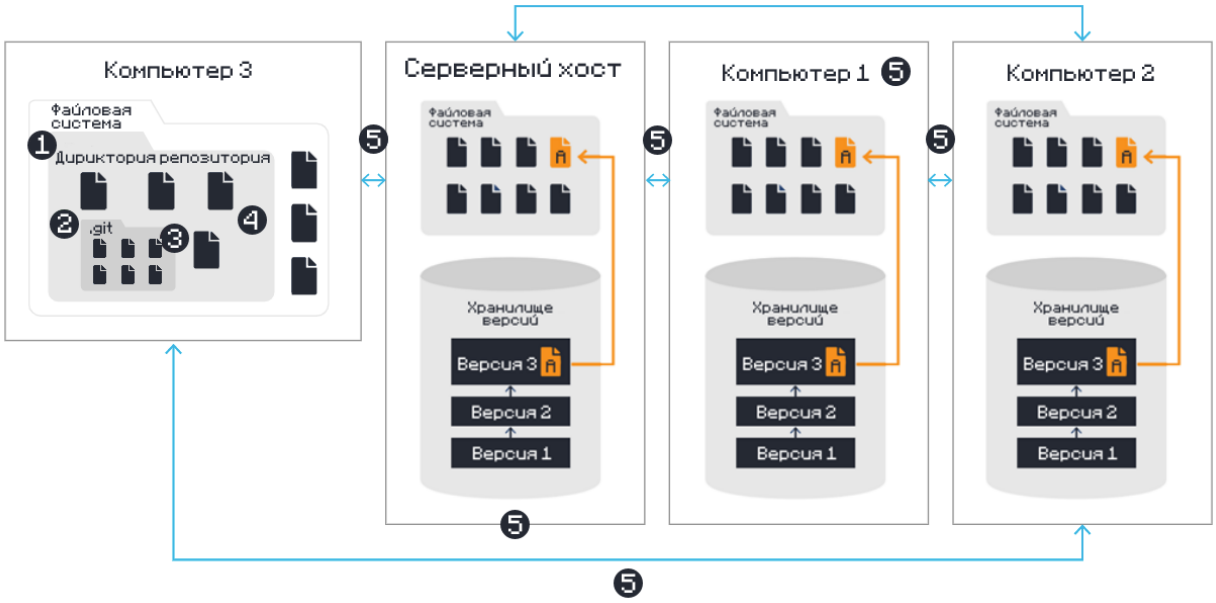


Рисунок 51 — Процесс клонирования репозитория

Когда вы выполняете команду git clone, Git:

1. Создает рабочий каталог в локальной файловой системе с именем репозитория или указанным именем, если оно указано;
2. Создает каталог .git внутри вновь созданной папки;

3. Копирует метаданные репозитория во вновь созданный каталог .git;
4. Создает рабочую копию последней версии файлов проекта;
5. Дублирует структуру ветвей клонированного удаленного репозитория и позволяет отслеживать изменения, внесенные в каждую ветвь, локально и удаленно — это включает создание и проверку локальной активной ветви, "форкнутой" из текущей активной ветви клонированного репозитория.

Просмотр измененных файлов в рабочей директории

Что было изменено в рабочем каталоге? Какие файлы были добавлены в область постановки? Git предоставляет команду `git status` для получения списка файлов, имеющих различия между рабочим каталогом и родительской ветвью. Сюда входят недавно добавленные неотслеживаемые файлы и удаленные файлы. Она также предоставляет список файлов, которые находятся в области постановки. Обратите внимание, что разница рассчитывается на основе последнего коммита, который локальный клон скопировал из родительской ветви в репозитории Git, а не обязательно последней версии в удаленном репозитории. Если с момента клонирования репозитория в него были внесены изменения, Git не будет учитывать эти изменения.

Команда: `git status`

Помимо списка файлов, вывод команды `git status` предоставляет дополнительную информацию, такую как:

- Текущая ветвь рабочего каталога
- Количество коммитов, на которое рабочий каталог отстает от последней версии родительской ветви
- Инструкции по обновлению локального репозитория и по постановке/отмене файлов.

Сравнение изменений между файлами

Хотите узнать, что было изменено в файле или в чем разница между двумя файлами? Git предоставляет команду git diff, которая по сути является общим инструментом сравнения файлов.

Команда: `git diff`

Поскольку эта команда является общим инструментом сравнения файлов, она включает множество опций для сравнения. При использовании этой команды файл не обязательно должен быть отслеживаемым файлом Git.

Например, вы можете:

1. Показать изменения между версией файла в рабочем каталоге и последней фиксацией, которую локальный клон скопировал из родительской ветви в репозитории Git:

```
$ git diff <путь к файлу>
```

2. Показать изменения между версией файла в рабочем каталоге и конкретной фиксацией из истории файлов:

```
$ git diff <идентификатор коммита> <путь к файлу>
```

3. Показать изменения между двумя коммитами файла из истории файлов. `<путь к файлу>` — это абсолютный или относительный путь к файлу для сравнения, а `<идентификатор коммита>` — это имя версии файла для сравнения.

```
$ git diff <коммита id 1> <коммит id 2> <путь к файлу>
```

4. Показать изменения между двумя файлами в рабочем каталоге или на диске.

```
$ git diff <путь к файлу 1> <путь к файлу 2>
```

Добавление и удаление файлов

Добавление и удаление файлов

После внесения изменений в файл в рабочем каталоге он должен сначала попасть в stage, прежде чем его можно будет обновить в репозитории Git. Git предоставляет команду git add для добавления файла(ов) в stage. Эти добавляемые в staging файлы могут включать новые неотслеживаемые файлы, существующие отслеживаемые файлы, которые были изменены, или даже отслеживаемые файлы, которые должны быть удалены из хранилища. Измененные файлы не нужно добавлять в рабочий каталог, если только изменения не должны быть добавлены в хранилище.

Команда: `git add`

Эта команда может быть использована более одного раза до обновления репозитория Git (с помощью commit). Кроме того, один и тот же файл может быть добавлен в stage несколько раз до фиксации. В stage добавляются только те файлы, которые указаны в команде git add.

Чтобы добавить один файл в staging:

```
$ git add <путь к файлу>
```

Чтобы добавить несколько файлов в staging, где `<путь к файлу>` — это абсолютный или относительный путь к файлу, который нужно добавить в область постановки, и может принимать подстановочные знаки.

```
$ git add <путь к файлу 1> ... <путь к файлу n>
```

Чтобы добавить все измененные файлы в staging:

```
$ git add .
```

Помните, что Git имеет три стадии, поэтому добавление файлов в область постановки — это только первый шаг двухэтапного процесса обновления репозитория Git (Рисунок 52).

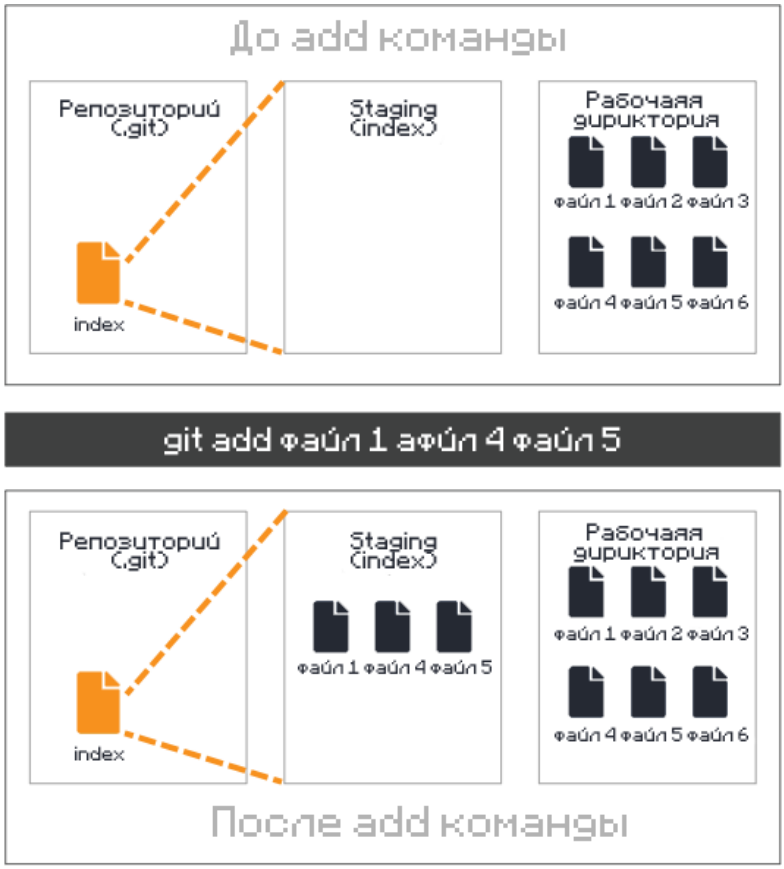


Рисунок 52 — git add

Удаление файлов из репозитория Git

Существует два способа удаления файлов из репозитория Git.

Первый способ:

Git предоставляет команду git rm для удаления файлов из репозитория Git. Эта команда добавит удаление указанного файла (файлов) в область хранения. Она не выполняет второй шаг обновления самого репозитория Git.

Команда: `git rm`

Чтобы удалить указанный файл(ы) из рабочего каталога и добавить это изменение в область постановки, используйте следующую команду:

```
$ git rm <путь к файлу 1> ... <путь к файлу n>
```

где `<путь к файлу>` — абсолютный или относительный путь к файлу, который нужно удалить из репозитория Git.

Чтобы добавить указанный файл(ы) для удаления в область хранения, не удаляя сам файл(ы) из рабочего каталога, используйте следующую команду:

```
$ git rm --cached <путь к файлу 1> ... <путь к файлу n>
```

Эта команда не будет работать, если файл уже находится в области хранения с изменениями (Рисунок 53).

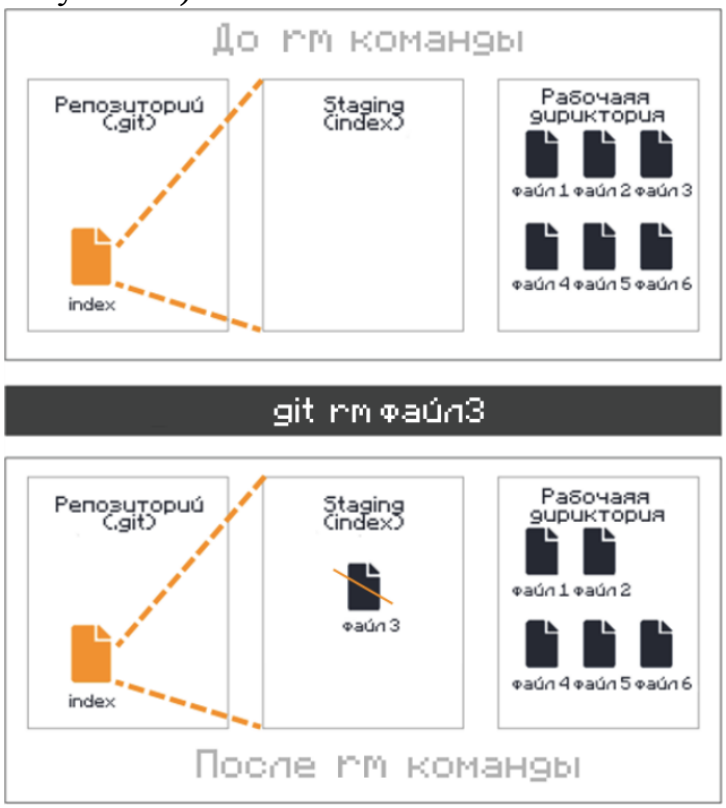


Рисунок 53 — git rm

Второй способ:

Он состоит из двух шагов. Сначала используем команду filesystem для удаления файла(ов). Затем добавим файл в stage с помощью команды Git `git add`, которая обсуждалась ранее.

```
$ rm <путь к файлу 1> ... <путь к файлу n>
$ git add <путь к файлу 1> ... <путь к файлу n>
```

Этот двухшаговый процесс эквивалентен использованию команды `git rm <путь к файлу 1> ... <путь к файлу n>`. Использование этой опции не позволяет сохранить файл в рабочем каталоге.

Обновление репозиториев

Обновление локального репозитория с изменениями в Staging

Помните, что в Git изменения в файле проходят три стадии: рабочий каталог, staging и репозиторий. Передача изменений содержимого из рабочего каталога в staging может быть выполнена с помощью команды `git add`, но как обновления попадают в репозиторий? Git предоставляет команду `git commit` для обновления локального репозитория изменениями, которые были добавлены в область постановки.

Команда: `git commit`

Эта команда объединяет все изменения содержимого в области постановки в один коммит и обновляет локальный репозиторий Git. Этот новый коммит становится последним изменением в репозитории Git. Если существует удаленный Git-репозиторий, он не будет изменен этой командой. Чтобы зафиксировать изменения из staging, используйте следующую команду:

```
$ git commit
```

Хорошей практикой разработки программного обеспечения является добавление сообщения коммита, чтобы объяснить причину изменений. Чтобы зафиксировать изменения из области постановки с сообщением, используйте следующую команду:

```
$ git commit -m "<сообщение>"
```

Если команда `git commit` будет выполнена без содержимого в staging, Git вернет сообщение, и с репозиторием Git ничего не произойдет, поскольку ничего не было изменено, что разумно (Рисунок 54).

Рисунок 54 — git commit

Обновление “удаленного” (в контексте расстояния) р епозитория

Для того чтобы поделиться изменениями содержимого локального Gitрепозитория с другими, “удаленный” Git-репозиторий должен быть обновлен вручную. Git предоставляет команду `git push` для обновления “удаленного” Git-репозитория изменениями содержимого из локального Git-репозитория.

Команда : `git push`

Эта команда не будет успешно выполнена, если существует конфликт с добавлением изменений из локального Git-репозитория в “удаленный” Git репозиторий. Конфликты возникают, когда два человека редактируют одну и ту же часть одного и того же файла. *Например, если вы клонируете репозиторий, а кто-то другой вносит изменения раньше вас, ваше внесение может создать конфликт. Конфликты должны быть разрешены, прежде чем* `git push` *будет успешным.*

Чтобы обновить содержимое локального репозитория до определенной ветки в “удаленном” репозитории, используйте следующую команду:

```
$ git push origin <имя ветки>
```

Чтобы обновить содержимое локального хранилища в master ветви “удаленного” хранилища, выполните следующую команду:

```
$ git push origin master
```

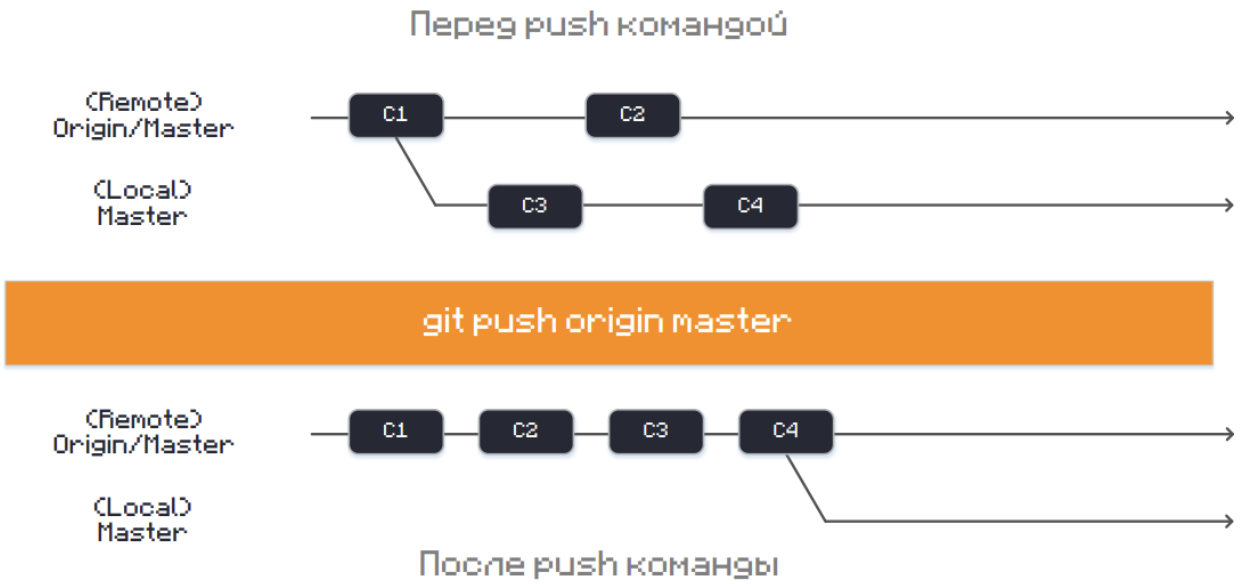


Рисунок 55 — git push

Обновление локальной копии репозитория

Локальные копии Git-репозитория не обновляются автоматически, когда другой участник вносит обновление в удаленный Git-репозиторий. Обновление локальной копии репозитория — это ручной шаг. Git предоставляет команду git pull для получения обновлений из ветки или репозитория. Эта команда также может быть использована для интеграции локальной копии с неродительской ветвью.

Команда: `git pull`

Говоря более подробно о команде git pull, то при ее выполнении происходят следующие действия:

1. Локальный репозиторий (каталог .git) обновляется последними коммитами, историей файлов и так далее из “удаленного” Git-репозитория. (Это эквивалентно команде Git git fetch);
2. Рабочая директория и ветка обновляется последним содержимым из пункта 1. (Это эквивалентно команде Git git merge);
3. На локальной ветке создается единый коммит с изменениями из пункта 1. Если существует конфликт слияния, его нужно будет разрешить;
4. Рабочая директория обновляется актуальным содержимым.

Чтобы обновить локальную копию репозитория Git из родительской ветки, выполните следующую команду

```
$ git pull
```

или

```
$ git pull origin
```

Чтобы обновить локальную копию репозитория Git из определенной ветки, выполните следующую команду:

```
$ git pull <имя ветки>
```

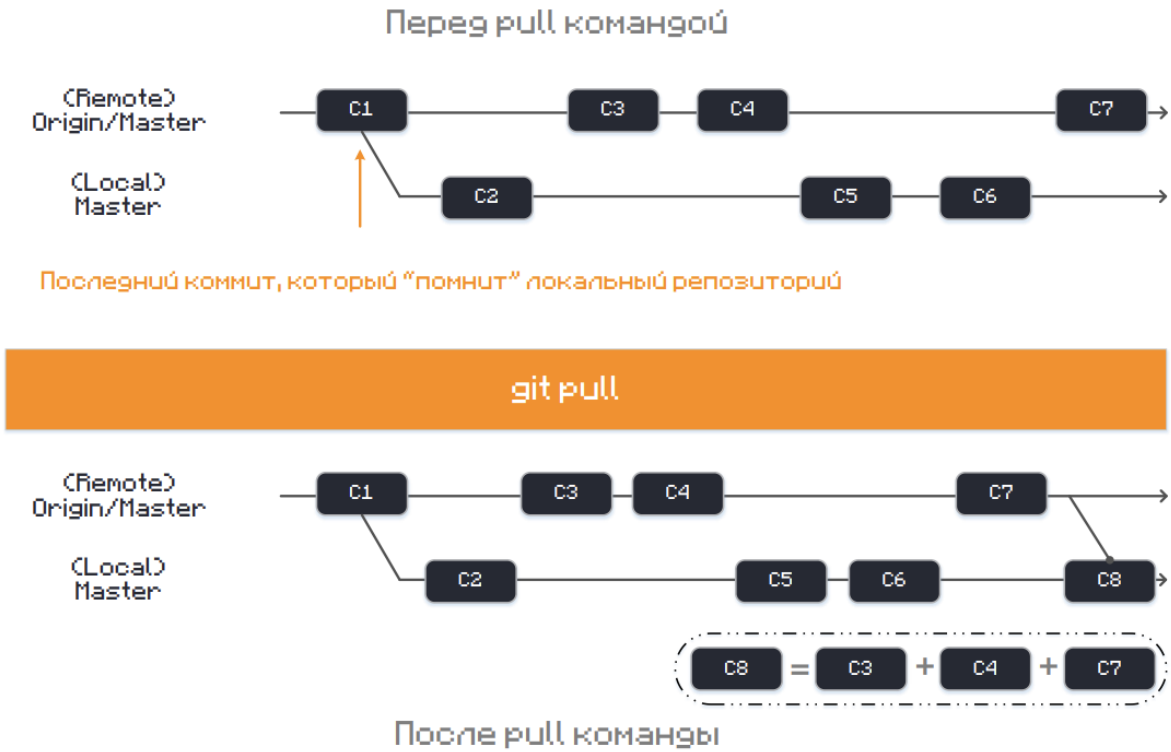


Рисунок 56 — git pull

Особенности ветвления

Создание и Удаление ветки

Ветви — очень полезная функция Git'a. Как обсуждалось ранее, существует множество преимуществ использования ветвей, но одно из основных заключается в том, что они позволяют вносить функции и изменения независимо от основного кода (ветви master).

Вариант 1

Git предоставляет команду git branch для отображения, создания или удаления ветви.

Команда : `git branch`

Чтобы создать ответвление, используйте следующую команду:

```
$ git branch <родительская ветвь> <имя создаваемой ветви>
```

где `<родительская ветвь>` — это ветвь, от которой нужно “отвестись”, а `<имя создаваемой ветви>` — это действительно имя, которым нужно назвать новую ветвь.

При использовании этой команды для того чтобы “отвестись”, Git создаст ответвление, но автоматически не поменяет текущую ветку, на это ответвление. Вам следует использовать команду `git switch` , чтобы переключить рабочий каталог на новую ветвь.

Вариант 2

Git предоставляет команду git checkout для переключения ветвей путем обновления рабочего каталога содержимым ветви.

Команда : `git checkout`

Чтобы создать ветку и переключиться на неё, используйте следующую команду:

```
$ git checkout –b <родительская ветвь> <имя создаваемой ветви>
```

Удаление ветки

Чтобы удалить локальную ветвь, выполните следующую команду:

```
$ git branch -d <имя удаляемой ветви>
```

Отображение ветвей

Чтобы получить список всех локальных ветвей, выполните следующую команду:

```
$ git branch
```

или

```
$ git branch –list
```

Слияние веток

Ветви расходятся друг с другом, когда в них вносятся изменения после их создания. Чтобы перенести изменения из одной ветви (исходной) в другую (целевую), необходимо их объединить. Когда Git объединяет ветвь, он берет изменения/коммиты из исходной ветви и применяет их к целевой ветви. Во время слияния изменяется только целевая ветвь. Исходная ветвь не трогается и остается неизменной. Например:

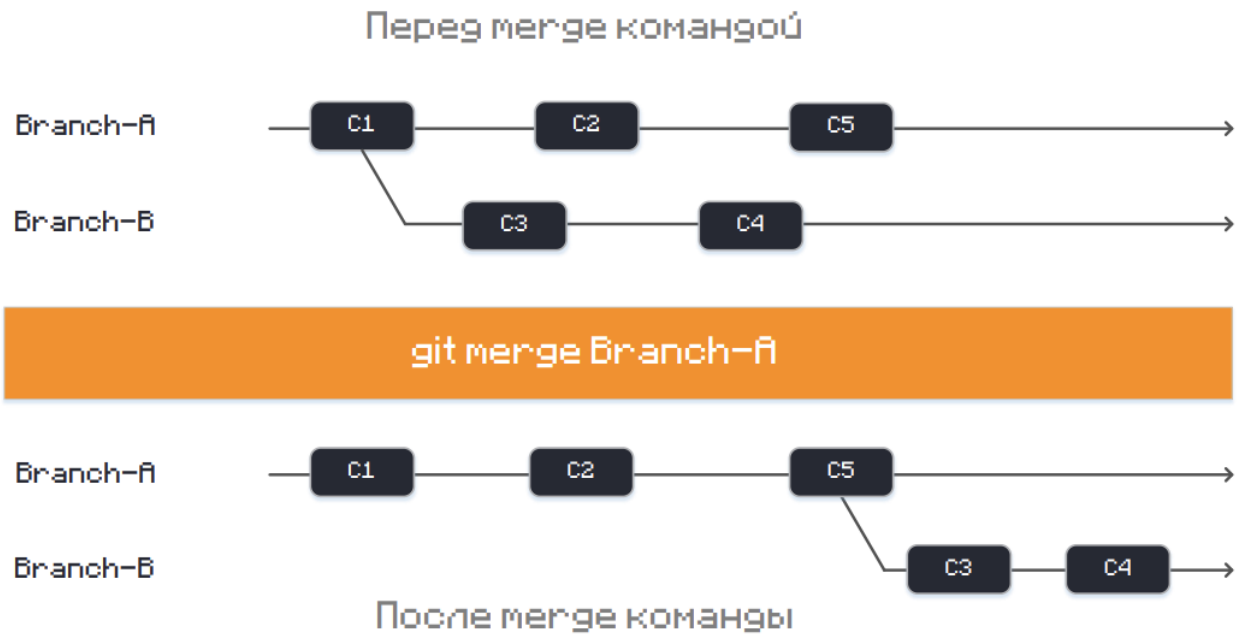


Рисунок 57 — git merge

1. При коммите#1 ветвь B ответвилась от ветви A.
2. После того как ветви разошлись, кто-то добавляет коммит#2 в ветвь A. Ветвь B не получает этих изменений.
3. Кто-то добавляет коммит#3 и коммит#4 в ветвь B. Ветвь A не получает этих изменений.
4. Кто-то добавляет коммит#5 в ветвь A. Ветвь B не получает этих изменений.
5. Теперь ветви A и B разошлись на два коммита каждая.
6. Предположим, что ветвь B хочет получить изменения из ветви A, так как она расходится (коммит#2 и коммит#5). Итак, ветвь A — это исходная ветвь, а ветвь B - целевая ветвь. В этом примере укажем, что коммиты были изменениями в разных файлах. В результате, коммит#2 и коммит#5 применяются к ветви B, а ветвь A остается прежней. Это называется ускоренным слиянием.

Слияние ветвей в режиме FAST-FORWARD

FAST-FORWARD или Быстрое слияние — это возможность алгоритма Git автоматически и без конфликтов применить изменения/коммиты из исходной ветви (ветвей) к целевой ветви. Обычно это возможно, когда в объединяемых 51 ветвях изменяются разные файлы или, когда изменяются разные строки одного файла. FAST-FORWARD слияние — это наилучший сценарий при выполнении слияния.

При нём Git объединяет различные коммиты из исходной ветки в целевую ветку. Поскольку ветви, по сути, являются просто указателями на коммиты, при быстром слиянии просто перемещается указатель, представляющий HEAD целевой ветви, а не добавляется новый коммит.

Обратите внимание, что для выполнения FAST-FORWARD слияния Git должен быть в состоянии объединить все существующие коммиты, не встречая никаких конфликтов.

Конфликты слияния

Изменение одного и того же файла в разных ветках, подлежащих слиянию, увеличивает вероятность возникновения конфликта слияния. Конфликт слияния — ситуация, когда Git не может выполнить FAST-FORWARD, потому что он не знает, как автоматически применить изменения из ветвей вместе для файла (файлов). В такой ситуации, разработчик должен вручную устранить эти конфликты, прежде чем ветви будут объединены. Ручное исправление конфликта добавляет новый коммит в целевую ветвь, содержащий коммиты из исходной ветви, а также исправленный конфликт(ы) слияния (Рисунок 58).

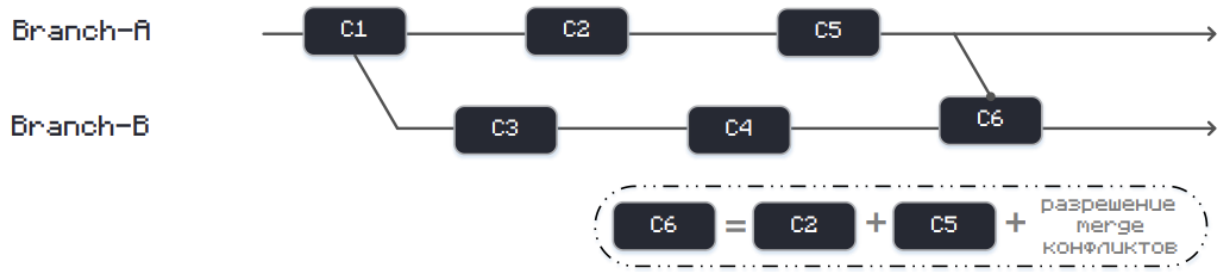


Рисунок 58 — merge conflict

Осуществление слияния

Git предоставляет команду git merge для объединения двух или более веток вместе.

Команда: `git merge`

Чтобы объединить ветку с текущей веткой/репозиторием, используйте следующую команду:

```
$ git merge <имя ветви>
```

где `<имя ветви>` — это исходная ветвь, которая объединяется с текущей ветвью.

При использовании команды git merge целевой веткой должна быть текущая ветка/хранилище, поэтому для слияния ветки в ветку, которая не является текущей веткой/хранилищем клиента, используйте следующие команды:

```
$ git checkout <имя целевой ветви>
$ git merge <имя ветви источника>
```

где `<имя целевой ветви>` — целевая ветвь, а `<имя ветви источника>` — исходная ветвь.

Чтобы объединить более одной ветви в текущую ветвь/репозиторий клиента, используйте следующую команду:

```
$ git merge <имя ветви 1> ... <имя ветви n>.
```

Это называется осьминожьим слиянием (octopus merge) рисунок 59.

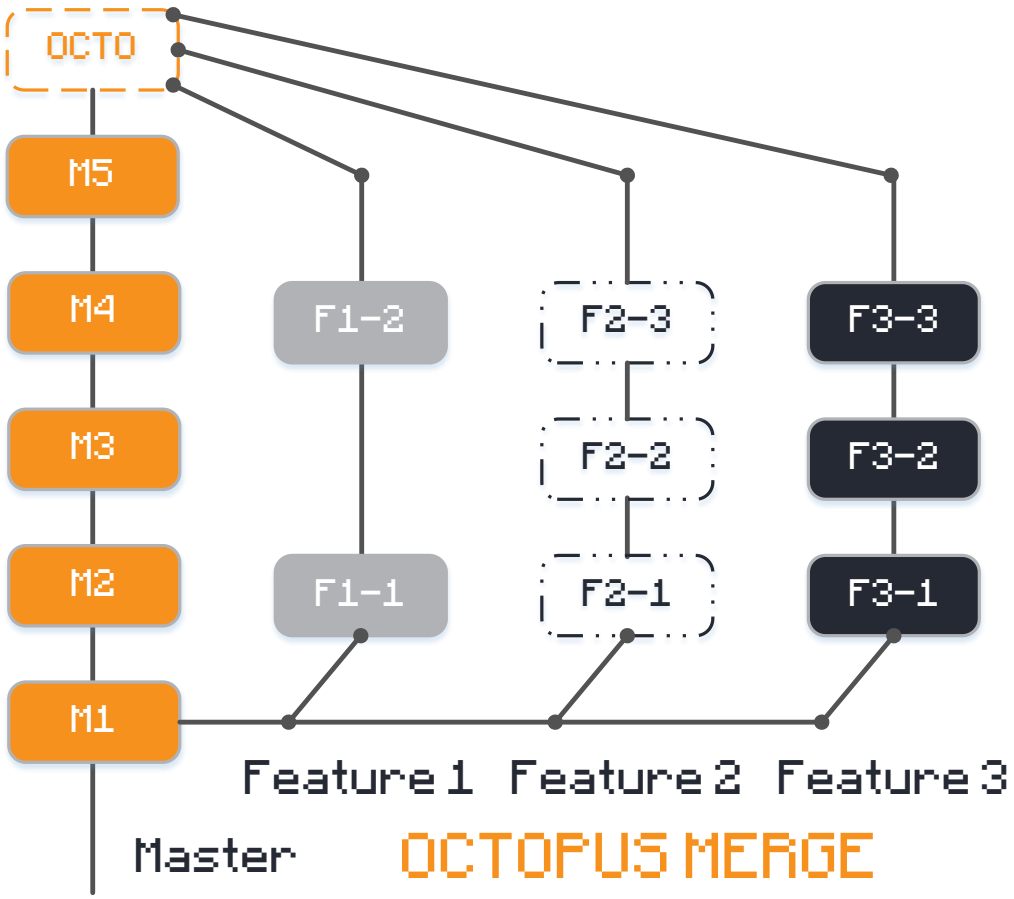


Рисунок 59 — octopus merge

Файлы .diff

Что такое `.diff`? Разработчики используют файл `.diff`, чтобы показать, как изменились две разные версии файла. Благодаря использованию специальных символов, этот файл может быть прочитан другими системами для интерпретации обновления файлов. Разница используется для реализации изменений путем сравнения и слияния двух версий. Некоторые проекты требуют, чтобы изменения представлялись через файл `.diff` в виде патча. Поскольку все это находится в одном файле, его называют унифицированным `diff`.

Символы и значения в унифицированном diff-файле показаны ниже:

- `+` : Указывает, что строка была добавлена;
- `-` : Указывает, что строка была удалена;
- `/dev/null` : Показывает, что файл был добавлен или удален;
- или "`пусто`": Дает контекстные линии вокруг измененных строк;
- `@@` : Визуальный индикатор начала следующего блока информации. В пределах изменений для одного файла их может быть несколько;
- `index` : Отображает сравниваемые коммиты.

Пример `diff` для файла с именем `check-network.yml`:

```
diff --git a/check-network.yml b/check-network.yml
index 09b4f0c..b1978ca 100644
--- a/check-network.yml
+++ b/check-network.yml
@@ -4,7 +4,7 @@
   roles:
     - ansible-pyats
   vars:
-    snapshot_file: "{{ inventory_hostname }}_bgp.json"
+    snapshot_file: "{{ inventory_hostname }}_routes.json"
   tasks:
   - set_fact:
       snapshot_data: "{{ lookup('file', snapshot_file) | from_json }}"
@@ -13,7 +13,7 @@
 #      var: snapshot_data
 #
   - pyats_parse_command:
-      command: show ip route bgp
+      command: show ip route
       compare: "{{ snapshot_data }}"
     register: command_output
```

Сигналом может быть "+" или "-" в зависимости от порядка следования хэшей. В данном формате есть три строки, показанные выше и ниже точно измененной строки для контекста, но вы можете заметить различия, сравнив строку "-" со строкой "+". Одним из изменений в этом патче является изменение имени файла моментального снимка, заменяя `...bgp.json` на `...routes.json`.

```
- snapshot_file: "{{ inventory_hostname }}_bgp.json"
+ snapshot_file: "{{ inventory_hostname }}_routes.json"
```

Вы всегда можете посмотреть разницу между двумя файлами из GitHub Pull Request в виде унифицированного diff, добавив `.diff` к URL GitHub.

Основы написания кода.

Методы, функции, модули и классы

Может показаться, что это просто — бросить код в файл и заставить его работать. Однако с ростом размера и сложности проекта, а также с привлечением других разработчиков (и заинтересованных сторон), необходимы дисциплинированные методы и лучшие практики, которые помогут разработчикам писать лучший код и легче сотрудничать вокруг него.

Одна вещь, с которой согласны большинство разработчиков, — это стремление писать чистый код. Но что такое чистый код?

Чистый код

Чистый код — это результат того, что разработчики стараются сделать свой код простым для чтения и понимания другими разработчиками. Что такое "чистый код" — вопрос несколько субъективный, но вот некоторые общие принципы:

- Является ли форматирование кода аккуратным, и следует ли оно общепринятой практике форматирования для соответствующего компьютерного языка (языков), и/или отвечает ли оно специфическим требованиям институционального, проектного и/или командного "руководства по стилю"?
    1. Используются ли ВСЕ табуляции или ВСЕ пробелы?
    2. Используется ли одинаковое количество табуляций или пробелов на каждый уровень отступа во всем файле? (В некоторых языках, например, в Python, это является обязательным требованием).
    3. Есть ли в нем отступы в нужных местах?
    4. Используется ли в нем последовательное форматирование синтаксиса, например, расположение фигурных скобок ({})?
- Являются ли имена переменных, объектов и другие имена, используемые в коде, интуитивно понятными?
- Организован ли код таким образом, чтобы он имел смысл? Например, сгруппированы ли декларации: функции вверху, основной код внизу или как-то иначе, в зависимости от языка и контекста?
- Документирован ли код внутренними комментариями?
- Служит ли каждая строка кода какой-либо цели? Удалили ли вы весь неиспользуемый код?
- Написан ли код так, чтобы общий код можно было использовать повторно, и чтобы весь код можно было легко протестировать на модулях?

Особенности Чистого кода:

- стандартизированное форматирование и интуитивно понятное именование для удобства чтения, понимания и поиска;

- общая организация для передачи информации о намерениях и облегчения поиска вещей;
- модульность для облегчения тестирования и повторного использования
- встроенные и другие комментарии;
- другие характеристики, которые помогают сделать код самодокументируемым.

Теоретически, другие разработчики должны иметь возможность понимать, использовать и изменять ваш код, не задавая вам вопросов. Это значительно ускоряет разработку, позволяя повторное использование, отладку, анализ безопасности, обзор и объединение кода, а также другие процессы. Это позволяет давно существующим проектам (например, проектам с открытым исходным кодом) с большей уверенностью включать ваш код. И это позволяет организациям продолжать эффективно использовать ваш код после того, как вы перешли на другие проекты или работу.

В отличие от этого, “нечистый” код быстро превращается в технический долг. *(Технический долг — это метафора программной инженерии, обозначающая накопленные в программном коде или архитектуре проблемы, связанные с пренебрежением к качеству при разработке программного обеспечения и вызывающие дополнительные затраты труда в будущем.)* Его можно не поддерживать или использовать с полной уверенностью. Это код, который необходимо рефакторить, или удалить и переписать, что дорого и требует много времени.

Каковы другие причины, по которым разработчики хотят писать чистый код?

1. Чистый код легче понять, он более компактен и лучше организован, что, как правило, приводит к тому, что код работает правильно (меньше ошибок) и выполняет требуемые функции.
2. Чистый код, будучи модульным, как правило, легче тестировать с помощью автоматизированных методов, таких как фреймворки модульного тестирования.
3. Чистый код, будучи стандартизированным, легче сканировать и проверять с помощью автоматизированных инструментов, таких как линтеры, или инструментов командной строки, таких как grep, awk и sed.
4. Он просто выглядит красивее.

Теперь, когда вы понимаете цели написания чистого кода, мы можем углубиться в лучшие практики его написания. В частности, рассмотрим, как разбивать код на методы и функции, модули и классы.

Методы и Функции

Методы и функции имеют одну и ту же концепцию; это блоки кода, которые при выполнении выполняют задачи, простите за тавтологию. Если метод или функция не выполняются, эти задачи не будут выполнены. Хотя абсолютных правил не существует, вот несколько стандартных лучших практик для определения того, следует ли инкапсулировать часть кода (в метод или функцию):

- Код, выполняющий дискретную задачу, даже если она выполняется только один раз, может быть кандидатом на инкапсуляцию. Классическими примерами являются служебные функции, которые оценивают входные данные и возвращают логический результат (например, сравнивают две строки по длине), выполняют операции ввода-вывода (например, считывают файл с диска) или переводят данные в другие формы (например, разбирают и обрабатывают данные). В этих случаях вы инкапсулируете для ясности и проверяемости, а также для возможного повторного использования или расширения в будущем.
- Код задачи, который используется более одного раза, вероятно, должен быть инкапсулирован. Если вы обнаружите, что копируете несколько строк кода, вероятно, это должен быть метод или функция. Вы можете приспособить небольшие вариации в использовании, используя логику для оценки передаваемых параметров (см. ниже).

Самое мощное в методах и функциях то, что их можно написать один раз и выполнять сколько угодно раз. При правильном использовании методы и функции упрощают ваш код, а значит, снижают вероятность ошибок.

Синтаксис функции в Python:

```
# Определение функции
def functionName:
    ... блоки кода...
# Вызов функции
functionName()
```

Аргументы и Параметры

Еще одной особенностью методов и функций является возможность выполнения кода на основе значений переменных, передаваемых при выполнении. Они называются аргументами. Для того чтобы использовать аргументы при вызове метода или функции, метод или функция должны быть написаны так, чтобы принимать эти переменные в качестве параметров.

Параметры могут быть любого типа, и каждый параметр в методе или функции может иметь свой тип данных. Аргументы, передаваемые методу или функции, должны соответствовать типу (типам) данных, ожидаемому методом или функцией.

В зависимости от языка программирования, некоторые языки требуют, чтобы тип данных был определен в параметре *(так называемые "сильно типизированные" языки)*, а некоторые разрешают это делать опционально.

Даже если спецификация типа параметра не является обязательной, обычно это хорошая идея. Это облегчает повторное использование кода, так как легче понять, какие параметры ожидает метод или функция. Это также делает сообщения об ошибках более понятными. Ошибки несоответствия типов легко исправить, в то время как неправильный тип, переданный в функцию, может вызвать ошибки, которые трудно понять, находясь глубже в коде.

Параметры и аргументы добавляют гибкость методам и функциям. Иногда параметр — это просто булев флаг, который определяет, должны ли выполняться определенные строки кода в методе или функции. Думайте о параметрах как о входе в метод или функцию.

Синтаксис функции с использованием аргументов и параметров в Python:

```
# Определение функции
def functionName(parameter1,...,parameterN):
    # Вы можете использовать параметры так же, как локальные
переменные
    ...блоки кода...
# Вызов функции
functionName("argument1", 4, {"argument3":"3"})
```

В приведенном выше примере функции передаются строка, целое число (или номер) и объект, содержащий ключ и значение.

Return выражения

Методы и функции выполняют задачи, а также могут возвращать значение. Во многих языках возвращаемое значение задается с помощью ключевого слова `return`, за которым следует переменная или выражение. Это называется оператором возврата. Когда оператор возврата выполняется, возвращается его значение, а весь код ниже него пропускается. В обязанности строки кода, вызывающей метод или функцию, входит получение значения `return`, но это не является обязательным.

Синтаксис функции с оператором возврата в Python:

```
# Определение функции
def functionName(parameter1,...,parameterN):
    # Вы можете использовать параметры так же, как локальные
переменные
    ...блоки кода...
    someVariable = parameter1 * parameter2
    return someVariable
# Вызов функции
myVariable = functionName("argument1", 4, {"argument3":"3"})
```

В приведенном выше примере возвращаемым значением будет строка "argument1argument1argument1argument1argument1", поскольку Python позволяет объединять строки с помощью оператора умножения.

Пример функции

Допустим, ваш исходный код выглядит следующим образом:

```
# Посчитайте длинны окружностей для кругов с радиусами 2, 5 и 7
radius1 = 2
radius2 = 5
radius3 = 7
# Формула длинны окружности: c = pi * диаметра
# Формула диаметра d = 2 * радиус
pi = 3.14 # (В данном примере pi будет фиксировано)
circumference1 = pi * radius1 * 2
print ("Длинна окружности с радиусом " + str(radius1) + " равна " +
str(circumference1))
circumference2 = pi * radius2 * 2
print ("Длинна окружности с радиусом " + str(radius2) + " равна " +
str(circumference2))
circumference3 = pi * radius3 * 2
print ("Длинна окружности с радиусом " + str(radius3) + " равна " +
str(circumference3))
```

Как вы можете видеть, здесь много дублирующегося кода.

Используя параметризированные методы, можно очистить код:

```
# Посчитайте длинны окружностей для кругов с радиусами 2, 5 и 7
# В этом примере, circumference и
# printCircumference имена функций.
# 'radius' – это параметр, который
# может быть использован в функции.
# Эта функция возвращает значение
# circumferenceValue в место откуда
# эта функция была вызвана.
def circumference(radius):
    # Формула длинны окружности: c = pi * диаметра
    # Формула диаметра d = 2 * радиус
    pi = 3.14 # (В данном примере pi будет фиксировано)
    circumferenceValue = pi * radius * 2
    return circumferenceValue


def printCircumference(radius):
    myCircumference = circumference(radius)
    print ("Длинна окружности с радиусом " + str(radius) + " равна " +
str(myCircumference))
radius1 = 2
radius2 = 5
radius3 = 7
```

```
# В приведенной ниже строке кода значение radius1 (2)
# аргумент функции printCircumference
printCircumference(radius1)
printCircumference(radius2)
printCircumference(radius3)
```

Обратите внимание, что версия кода, в которой используются функции, параметры и аргументы, не содержит дублирующего кода. Кроме того, используя функции, вы можете маркировать блоки кода, что делает их назначение более понятным. Если бы этот пример был более сложным и в каждой функции было бы много строк кода, то наличие блоков кода, продублированных в файле три раза, значительно усложнило бы понимание.

Методы VS. Функции

Если методы и функции имеют одну и ту же концепцию, почему они называются по-разному? Разница между методами и функциями заключается в том, что функции — это отдельные блоки кода, а методы — блоки кода, связанные с объектом, обычно для объектно-ориентированного программирования.

Пример метода

Код выше можно изменить, чтобы превратить функцию в метод, получив тот же результат:

```
# Посчитайте длинны окружностей для кругов с радиусами 2, 5 и 7
# В этом примере есть класс с именем Circle.
# Классы мы рассмотрим позже. circumference
# и printCircumference имена методов класса.
# Они возвращают значение circumferenceValue
# в место откуда этот метод был вызван.
class Circle:
    def __init__(self, radius):
        self.radius = radius


    def circumference(self):
        # Формула длинны окружности: c = pi * диаметра
        # Формула диаметра d = 2 * радиус
        pi = 3.14 # (В данном примере pi будет фиксированно)
        circumferenceValue = pi * self.radius * 2
        return circumferenceValue


    def printCircumference(self):
        myCircumference = self.circumference()
        print ("Длинна окружности с радиусом " + str(self.radius) + "
равна " + str(myCircumference))
```

```
radius1 = 2
radius2 = 5
radius3 = 7
# Поскольку Circle - это класс, он должен быть
# оснащён значением радиуса.
circle1 = Circle(radius1)
# Поскольку printCircumference – это метод, он должен
# вызываться используя синтакс [экземпляр класса].[метод]
# Вызов printCircumference() работать не будет :/
circle1.printCircumference()
circle2 = Circle(radius2)
circle2.printCircumference()
circle3 = Circle(radius3)
circle3.printCircumference()
```

Модули

Модули — это способ создания независимых и самодостаточных частей кода, которые можно использовать повторно. Разработчики обычно используют модули для разделения большого проекта на более мелкие части. Таким образом, код легче читать и понимать, а каждый модуль можно разрабатывать параллельно без конфликтов. Модуль упаковывается в один файл. Помимо того, что он доступен для интеграции с другими модулями, он должен работать самостоятельно.

Модуль состоит из набора функций и обычно содержит интерфейс для интеграции с другими модулями. По сути, он является библиотекой и не может быть внедрен.

Пример модуля

Ниже представлен модуль с набором функций, сохраненных в сценарии под названием `circleModule.py`.

```
# Дан радиус, выведете длину окружности данного круга.
# Формула длинны окружности: c = pi * 2 * radius
class Circle:
    def __init__(self, radius):
    self.radius = radius


    def circumference(self):
        pi = 3.14
        circumferenceValue = pi * self.radius * 2
        return circumferenceValue

    def printCircumference(self):
        myCircumference = self.circumference()
        print("Длинна окружности с радиусом " + str(self.radius) + "
равна " + str(myCircumference))
```

Приложение, которое существует в том же каталоге, что и `circleModule.py`, может использовать этот модуль, импортируя его, инстанцируя класс, а затем используя точечную нотацию для вызова его функций, как показано ниже:

```
from circleModule import Circle

# Первый экземпляр класса Circle.
circle1 = Circle(2)
# Вызовите функцию printCircumference для инстанцированного класса
circle1.
circle1.printCircumference()
# Еще два инстанцирования и вызова методов для класса Circle.
circle2 = Circle(5)
circle2.printCircumference()
circle3 = Circle(7)
circle3.printCircumference()
```

Классы

Объектно-ориентированное программирование (ООП), как оно изначально задумывалось, основано на некоторых формально определенных свойствах: инкапсуляции, абстракции данных, полиморфизме и наследовании.

В большинстве языков ООП и в Python классы являются средством объединения данных и функциональности. Каждое объявление класса определяет новый тип объекта.

Инкапсуляция функциональности вместе с хранением данных в одной структуре также обеспечивает один из аспектов абстракции данных. Функции, определенные в классе, называются методами класса. Классы могут иметь переменные класса и переменные объекта. При создании нового объекта класса создаются новые члены данных класса и члены данных объекта (переменные). Новые классы могут быть определены на основе существующих, ранее определенных классов, так что они наследуют свойства, члены данных и функциональность (методы).

Как и другие структуры данных и переменные Python, объекты инстанцируются (создаются) при первом использовании, а не объявляются заранее. Класс может быть инстанцирован (создан) несколько раз, и каждый раз со своими собственными значениями атрибутов данных, специфичных для объекта. (Классы Python также поддерживают переменные класса, которые являются общими для всех объектов класса). Вне области имени класса ссылки на методы и атрибуты данных класса выполняются с использованием точечной нотации: [экземпляр класса].[имя метода].

***Примечание**: В отличие от других языков ООП, в Python нет возможности создавать "приватные" переменные класса или внутренние методы. Однако, по соглашению, методы и переменные с одним предшествующим знаком подчеркивания (_) считаются приватными и не должны использоваться или упоминаться за пределами класса.*

Code Review и тестирование

Что такое Code Review и почему вам следует это делать?

Code Review — процесс просмотра кодовой базы разработчики, её подмножества или конкретных изменений кода и предоставления ими обратной связи. Этих разработчиков часто называют reviewer'ами. По возможности лучше иметь более несколько ревьюеров (Рисунок 26).

А лучше всего, если они понимают назначение кода, тогда они могут дать качественный и релевантный отзыв. Ревьюеры будут оставлять комментарии, обычно автору кода, о том, что, по их мнению, необходимо исправить. Поскольку многие комментарии могут быть субъективными, автор сам решает, нужно ли устранять замечание, при условии, что ревьюер(ы) согласится(тся), если оно не будет исправлено. Этот процесс просмотра кода происходит только после того, как изменения в коде завершены и протестированы.

Цель этих "обзоров" — убедиться, что окончательный код:

- легко читается;
- легко понять;
- следует лучшим практикам кодирования;
- использует правильное форматирование;
- не содержит ошибок;
- имеет надлежащие комментарии и документацию;
- чистый.

Проведение code review приносит пользу всей команде. Автор получает информацию от ревьюеров и узнает о лучших практиках, других способах реализации кода и различных стилях кодирования. В результате обзора автор учится на своих ошибках и в следующий раз сможет написать более качественный код. Обзоры кода нужны не только младшим разработчикам, это отличный процесс обучения для всех.

Code review также передают знания о коде между разработчиками. Если ревьюерам придется работать над этим фрагментом кода в будущем, они будут лучше понимать, как он работает.

Code review — способ доработать рабочий код или выявить потенциальные ошибки, что повышает качество кода. В целом, наличие еще одного взгляда на код никогда не помешает.

Типы Code Review

Формальный

Формальный code review — это серия встреч с целью обзора всей кодовой базы. На этих встречах разработчики просматривают код строка за строкой, подробно обсуждая каждую из них. Этот тип обзора кода способствует обсуждению между всеми рецензентами.

Формальный code review позволяет ревьюерам прийти к консенсусу, что может привести к улучшению обратной связи. Вы можете проводить новый обзор кода каждый раз, когда устраняются замечания.

Детали встреч, такие как участники, комментарии и комментарии, которые будут рассмотрены, документируются. Этот тип обзора кода часто называют инспекцией Фагана и он характерен для проектов, использующих водопадную методологию разработки программного обеспечения.

Современная адаптация формального обзора кода заключается в проведении одной встречи для обзора только изменений кода. Таким образом, код может только выиграть от живого обсуждения между рецензентами. Это иногда называют прохождением.

Ревью на основе изменений

Code review на основе изменений, также известный как обзор кода с помощью инструментов, рассматривает код, который был изменен в результате ошибки, пользовательской истории, фич, коммитов и т. д.

Для определения изменений кода, которые необходимо просмотреть, обычно используется инструмент экспертной проверки кода, который выделяет изменения кода. Этот тип обзора кода инициируется разработчиками, которые внесли изменения в код и отвечают за устранение согласованных замечаний. В процессе обзора ревьюеры обычно выполняют обзор независимо друг от друга и предоставляют комментарии с помощью инструмента коллегиального обзора кода.

Он позволяет легко определить фактические изменения кода, подлежащие рассмотрению, и дает возможность нескольким ревьюерам получить разносторонний взгляд на код.

Ревью из-за-плеча

Code review из-за-плеча — это именно то, на что он похож. Ревьюер смотрит через плечо разработчика, написавшего код. Разработчик, написавший код, строка за строкой просматривает изменения кода, а ревьюер дает обратную связь.

При таком методе, если исправление не является сложным, код может быть изменен на месте, чтобы ревьюер мог сразу же пересмотреть его. Преимущество такого ревью "через плечо" заключается в том, что происходит прямое взаимодействие между автором кода и тем, кто этот код проверяет, что позволяет обсудить правильность исправления. Недостатком этого типа обзора кода является то, что в нем обычно участвует только один ревьюер, поэтому комментарии могут быть односторонними.

Email Pass-Around

Email Pass-Around может происходить после автоматической отправки электронных писем системами управления исходным кодом, в момент его проверки. Когда письма отправлены, другие разработчики должны просмотреть изменения кода, которые были сделаны в этой проверке. Недостатком этого типа обзора кода является то, что иногда один чек-ин

может быть лишь частью всего изменения кода, поэтому он может не включать надлежащий контекст для полного понимания изменений кода.

Тестирование

Зачем разработчики тестируют программное обеспечение? Простой ответ — с целью убедиться, что оно работает так, как должно работать. За этим ответом скрывается множество нюансов и деталей.

Начнем с того, что тестирование программного обеспечения классически подразделяется на две общие категории:

- Функциональное тестирование направлено на определение того, правильно ли работает программное обеспечение. Ведет ли оно себя так, как задумано в логическом смысле, начиная с самых низких уровней детализации, изучаемых при модульном тестировании, и заканчивая более высокими уровнями сложности, изучаемыми при интеграционном тестировании?
- Нефункциональное тестирование изучает удобство использования, производительность, безопасность, отказоустойчивость, соответствие требованиям, локализацию и многие другие вопросы. Этот тип тестирования позволяет выяснить, соответствует ли программное обеспечение своему назначению, обеспечивает ли оно запланированную ценность и минимизирует ли риск.

Вы можете подумать, что функциональное тестирование происходит на ранних этапах цикла разработки, а нефункциональное тестирование начинается после того, как части программного обеспечения построены или даже завершены. Это неверно. Некоторые виды нефункционального тестирования (например, определение соответствия конкретного языка, библиотеки с открытым исходным кодом или компонента требованиям проекта или стандарта) должны проводиться задолго до завершения разработки.

При разработке программного обеспечения "Agile" предпочтение отдается высоко адаптируемому, минимально планируемому созданию и расширению минимально жизнеспособного продукта (MVP) в течение коротких спринтов. Это означает, что продукт существует в той или иной форме с самого начала процесса. А это значит, что его можно с самого начала подвергать как функциональным, так и нефункциональным тестам.

Некоторые разработчики выступают за использование тестирования в качестве основы для управления разработкой программного обеспечения. Это означает, что требования к дизайну записываются в виде тестов, а затем пишется программное обеспечение для прохождения этих тестов. Это называется Test-Driven Development (TDD).

Давайте рассмотрим некоторые методы и инструменты для тестирования строк кода, блоков, функций и классов.

Unit-тестирование

Детальное функциональное тестирование небольших фрагментов кода (строк, блоков, функций, классов и других компонентов в изоляции) обычно называется Unit (*модульным)* тестированием. Современные разработчики обычно автоматизируют этот вид тестирования с помощью соответствующих фреймворков, кторые представляют собой программное обеспечение, позволяющее делать утверждения о тестируемых условиях и определять, справедливы ли эти утверждения в определенный момент выполнения. **Например:**

```
a = 2 + 2
assert a == 4
```

Ключевое слово `assert` фактически является родным для `Python`. В данном случае утверждение вернет true, потому что `2 + 2` действительно равно `4`. С другой стороны, если бы выражение выглядело так:

```
assert a == 5
```

Оно вернуло бы false и вызовало ошибку.

Сбор утверждений и составление отчетов по тестам упрощается с помощью фреймворков для тестирования. Некоторые примеры тестовых фреймворков для Python включают:

- `unittest` — это фреймворк, включенный в Python по умолчанию. Он позволяет создавать коллекции тестов в виде методов, расширяющих стандартный класс TestCase.
- `PyTest` — это фреймворк, который легко добавляется в Python (из репозиториев `pip: pip3 install pytest`). `PyTest` может выполнять тесты `unittest` без модификации, но он также упрощает тестирование, позволяя разработчикам создавать тесты в виде простых функций, не прибегая к методам класса. PyTest используется некоторыми более специализированными наборами тестов, такими как PyATS от Cisco.

Ниже мы рассмотрим некоторые различия между ними.

Простое unit-тестирование с помощью PyTest

`PyTest` удобен тем, что он автоматически выполняет любые скрипты, которые начинаются с `test_` или заканчиваются `_test.py`, а внутри этих скриптов автоматически выполняет любые функции, начинающиеся с '`test_`' или '`tests_`'. Таким образом, мы можем протестировать часть кода (например, функцию), скопировав ее в файл, импортировав `pytest`, добавив соответствующие функции тестирования (имена которых начинаются с `tests_`), сохранив файл под именем, которое также начинается с '`tests_`', и запустив его с помощью `PyTest`.

Предположим, мы хотим протестировать функцию `add5()` , которая прибавляет 5 к переданному значению и возвращает результат:

```
def add5(v):
    myval = v + 5
    return myval
```

Мы можем сохранить функцию в файле `tests_mytest.py`. Затем импортировать `pytest` и написать функцию, содержащую тесты, под названием `tests_add5()` :

```
# файл tests_mytest.py
import pytest

def add5(v):
    myval = v + 5
    return myval


def tests_add5():
    r = add5(1)
    assert r == 6
    r = add5(5)
    assert r == 10
    r = add5(10.102645)
    assert r == 15.102645
```

Тесты в нашей тестовой функции используют стандартное ключевое слово Python `assert`. `PyTest` скомпилирует и сообщит об этих результатах, как при сборе тестовых элементов из файла (предварительный этап, на котором `PyTest` исследует собственный анализ кода Python и сообщает о правильном использовании типов и других проблемах, возникающих до выполнения), так и при выполнении функции `tests_add5()`.

Затем у нас появляется возможность запустить тесты, используя:

```
pytest tests_mytest.py
```

И результат, в консоли будет выглядеть примерно так:

```
============================= test session starts
==============================
platform darwin -- Python 3.8.1, pytest-5.3.5, py-1.8.1, pluggy-0.13.1
rootdir: /home/tana/python/mytest
collected 1 item
tests_mytest.py .
[100%]
============================== 1 passed in 0.01s
==============================
```

Заметим, что хотя тестируемая функция, безусловно, *примитивнейшая*, многие production-ready программы содержат функции, которые, подобно этой, выполняют математические операции над своими аргументами. Как правило, эти функции вызываются функциями более высокого уровня,

которые затем выполняют дополнительную обработку возвращаемых значений.

Если в функции нижнего уровня допущена ошибка, в результате которой она возвращает плохой результат, это, скорее всего, отразится на выводе более высокого уровня. Но из-за всех промежуточных обработок может быть трудно или невозможно найти источник ошибки *(или даже заметить, произошла ли ошибка)*, глядя на вывод этих функций более высокого уровня или на вывод программы в целом.

Это одна из причин, почему детальное модульное тестирование необходимо для разработки надежного программного обеспечения. А также это причина, по которой модульные тесты должны добавляться каждый раз, когда вы добавляете что-то существенное в код любого уровня, а затем повторно выполняться при каждом внесенном вами изменении.

Простое unit-тестирование с помощью `unittest`

Фреймворк `unittest` требует иного синтаксиса, чем `PyTest`. Для `unittest` необходимо проклассифицировать встроенный класс `TestCase` и тестировать, переопределяя его встроенные методы или добавляя новые методы, имена которых начинаются с 'test_'. Пример кода модульного тестирования, приведенный выше, может быть изменен для работы с `unittest` следующим образом:

```
import unittest

def add5(v):
    myval = v + 5
    return myval


class tests_add5(unittest.TestCase):
    def test_add5(self):
        self.assertEqual(add5(1),6)
        self.assertEqual(add5(5),10)
        self.assertEqual(add5(10.102645),15.102645)


if __name__ == '__main__':
    unittest.main()
```

Как и в `PyTest`, для начала вы импортируете модуль `unittest`. Далее следует ваша функция.

Чтобы проклассифицировать класс `TestCase`, передайте его в свой собственный (производный) класс тестов (снова названный `tests_add5`, хотя теперь это класс, а не функция), в результате чего последний наследует все характеристики первого.

Далее используйте метод `assertEqual` из `unittest` (это один из множества встроенных методов тестирования) так же, как вы использовали родной `assert`

из Python в примере PyTest. По сути, вы запускаете свою функцию с разными аргументами и проверяете, совпадают ли возвращаемые значения с ожидаемыми.
Последняя строка — это стандартный способ разрешить выполнение нашей программы в командной строке, вызвав ее главную функцию, которая в данном случае определена unittest.

Сохранив этот файл например, снова как tests_mytest.py, добавив аргумент -v для получения подробного отчета можно получить следующий вывод:

```
python3 tests_mytest.py -v

test_add5 (__main__.tests_add5) ... ok
----------------------------------------------------------------------
Ran 1 test in 0.000s
OK
```

Интеграционное тестирование

После модульного тестирования идёт интеграционное тестирование, которое позволяет убедиться, что все отдельные блоки, которые вы создавали, правильно сочетаются друг с другом и образуют целостное приложение.

Например, предположим, что приложение, которое вы пишете, должно обращаться к локальной веб-службе для получения данных конфигурации, включая имя соответствующего узла базы данных. Возможно, вы захотите проверить значения переменных, установленных при вызове этих функций. Если бы вы использовали PyTest, вы могли бы сделать это следующим образом:

```
import requests   # модуль для веб-запросов

def get_config():
    return requests.get("http://localhost/get_config").content


def set_config(dbhost):
    requests.get("http://localhost/config_action?dbhost="+dbhost)

save_dbhost = ""

def setUp():
    global save_dbhost
    save_dbhost = get_config()


def tearDown():
    global save_dbhost
    set_config(save_dbhost)
```

```
def test_setconfig():
    setUp()
    set_config("TESTVAL")
    assert get_config() == "ESTVAL"
    tearDown()
```

Обратите внимание на то, что метод `test_setconfig()` намеренно вызывает функцию `setUp()` перед запуском тестов и функцию `tearDown()` после этого. В `unittest` методы `setUp()` и `tearDown()` предоставляются классом `TestCase`, могут быть переопределены в определенном вами подклассе и выполняются автоматически.

Запуск этого кода с помощью PyTest может выдать такой результат:

```
============================= test session starts ==============================
platform linux2 -- Python 2.7.15+, pytest-3.3.2, py-1.5.2, pluggy-0.6.0
rootdir: /home/ubuntu/deploysample, inifile:
collected 1 item
test_sample_app.py F                                                     [100%]
=================================== FAILURES ===================================
_______________________________ test_setconfig ________________________________
    def test_setconfig():
        setUp()
        set_config("TESTVAL")
>       assert get_config() == "ESTVAL"
E       AssertionError: assert 'TESTVAL' == 'ESTVAL'
E         - TESTVAL
E         ? -
E         + ESTVAL
test_sample_app.py:21: AssertionError
------------------------------- Captured log call -------------------------------
connectionpool.py          225 DEBUG    Starting new HTTP connection (1):
localhost:80
connectionpool.py          437 DEBUG    http://localhost:80 "GET /get_config
HTTP/1.1" 200 7
connectionpool.py          225 DEBUG    Starting new HTTP connection (1):
localhost:80
connectionpool.py          437 DEBUG    http://localhost:80 "GET
/config_action?dbhost=TESTVAL HTTP/1.1" 200 30
connectionpool.py          225 DEBUG    Starting new HTTP connection (1):
localhost:80
connectionpool.py          437 DEBUG    http://localhost:80 "GET /get_config
HTTP/1.1" 200 7
=========================== 1 failed in 0.09 seconds ===========================
```

Если исправить неработающий тест, то увидите, что все работает идеально:

```
============================= test session starts ==============================
platform linux2 -- Python 2.7.15+, pytest-3.3.2, py-1.5.2, pluggy-0.6.0
rootdir: /home/ubuntu/deploysample, inifile:
collected 1 item
test_sample_app.py .                                                     [100%]
========================== 1 passed in 0.07 seconds ============================
```

Опять же, интеграционные тесты должны запускаться до внесения изменений за прошедший день, при внесении значительных изменений и

перед завершением работы за день. Если вы используете CI *(Continuous Integration)*, все найденные ошибки должны быть исправлены до того, как вы сделаете что-либо еще.

Разработка через тестирование (TDD)

Создание небольших, простых модульных и интеграционных тестов вокруг небольших фрагментов кода помогает двумя способами:

- гарантирует, что модули соответствуют своему назначению. Другими словами, убеждаетесь, что блоки выполняют то, что диктуют требования, в контексте вашего развивающегося решения.
- позволяет выявлять ошибки на месте и исправлять их на ранней стадии, избавляя от проблем в дальнейшем при тестировании или использовании частей решения более высокого порядка, которые зависят от этих компонентов.

Первое так же важно, как и второе, поскольку позволяет тестированию подтвердить дизайн системы или, если это не удается, направить локальный рефакторинг, более широкий редизайн или пересмотр требований.

Тестирование для проверки замысла проекта в свете требований подразумевает, что вы должны написать код тестирования до написания кода приложения. Выразив требования в коде тестирования, вы можете писать код приложения до тех пор, пока он не пройдет тесты, которые вы создали в коде тестирования.

Это принцип Test-Driven Development или Разработки через тестирование. Основная схема TDD - это пятиэтапный повторяющийся процесс:

1. Создайте новый тест (добавив его к существующим тестам, если они уже существуют). Идея здесь заключается в том, чтобы зафиксировать некоторые требования к единице кода приложения, которую вы хотите произвести.
2. Запустите тесты и посмотрите, не провалятся ли они по неожиданным причинам. Если это происходит, исправьте тесты. Обратите внимание, что ожидаемые неудачи здесь являются приемлемыми (например, если ваш новый тест не сработает, потому что функция, которую он призван тестировать, еще не существует, это будет приемлемой неудачей на данном этапе).
3. Напишите код приложения для прохождения нового теста. Правило здесь таково: не добавлять в приложение ничего, кроме того, что требуется для прохождения теста.
4. Запустите тесты, чтобы проверить, не провалились ли они. Если это так, исправьте код приложения и попробуйте снова.
5. Рефакторить и улучшать код приложения. Каждый раз проводите тесты заново и исправляйте код приложения, если возникли сбои.

Действуя таким образом, тестовый пакет ведет и развивается в ногу с вашим приложением. Процесс может происходить построчно, обеспечивая

очень высокое покрытие тестов и высокую уверенность в том, что и тестовый пакет, и приложение корректны в любой точке остановки. Совместное развитие кода тестов и приложения таким образом:

- Обязывает разработчиков последовательно думать о требованиях (и о том, как отразить их в тестах);
- Помогает прояснить и ограничить то, что должен делать код (потому что он просто должен проходить тесты), ускоряя разработку и поощряя простоту и хорошее использование паттернов проектирования;
- Обязательное создание высокотестируемого кода. Это код, который, например, разбивает операции на чистые функции, которые можно тестировать изолированно, в любом порядке и т.д.

Вникаем в форматы данных

Форматы данных

Rest API позволяют обмениваться информацией с удаленными службами и оборудованием. Также, как и интерфейсы, построенные на основе этих API, включая специализированные средства интерфейса командной строки и интеграционные комплекты разработки программного обеспечения (SDK) для популярных языков программирования.

При управлении этими API с помощью программного обеспечения полезно иметь возможность получать и передавать информацию в формах, соответствующих стандартам, машиночитаемых и человекочитаемых. Это позволит вам:

- **Легко использовать** готовые программные компоненты и/или встроенные языковые средства для преобразования сообщений в формы, которыми легко манипулировать и из которых можно извлекать данные, например, в структуры данных, характерные для используемого вами языка программирования. Вы также можете преобразовывать их в другие стандартные форматы, которые могут понадобиться вам для различных целей;
- **Легко писать код** для составления сообщений, которые могут принимать удаленные объекты;
- **Самостоятельно читать и интерпретировать** полученные сообщения, чтобы убедиться, что ваше программное обеспечение обрабатывает их правильно, и вручную составлять тестовые сообщения для отправки удаленным объектам;
- **Легче обнаруживать** "неправильные" сообщения, вызванные ошибками передачи или другими ошибками, мешающими коммуникации.

Сегодня тремя наиболее популярными стандартными форматами для обмена информацией с удаленными API являются `XML`, `JSON` и `YAML`. Стандарт `YAML` был создан как надмножество `JSON`, поэтому любой легальный документ

JSON может быть разобран и преобразован в эквивалентный YAML, и (с некоторыми ограничениями и исключениями) наоборот. XML, более старый стандарт, не так прост для разбора, и в некоторых случаях он лишь частично (или совсем не) конвертируется в другие форматы. Поскольку XML является более старым, инструменты для работы с ним уже достаточно совершенны.

Разбор XML, JSON или YAML является частым требованием при взаимодействии с интерфейсами прикладного программирования (API). Часто встречающийся шаблон в реализации REST API выглядит следующим образом:

1. Аутентификация, обычно путем отправки комбинации пользователь/пароль и получения токена с истекающим сроком действия для использования при аутентификации последующих запросов.
2. Выполнение GET-запроса к заданной конечной точке (аутентификация по мере необходимости) для получения состояния ресурса, запрашивая XML/JSON/YAML в качестве выходного формата.
3. Изменить возвращаемый XML/JSON/YAML.
4. Выполнить POST (или PUT) к той же конечной точке (опять же, аутентифицируясь по мере необходимости), чтобы изменить состояние ресурса, снова запрашивая XML/JSON/YAML в качестве выходного формата и интерпретируя его по мере необходимости, чтобы определить, была ли операция успешной.

XML

XML — расширяемый язык разметки является производным от языка структурированной, обобщенной разметки (SGML), а также родителем языка гипертекстовой разметки (HTML). XML — это общая методология для обертывания текстовых данных в симметричные теги для обозначения семантики. Имена файлов XML обычно заканчиваются на ".xml".

Пример XML-документа

Например, простой XML-документ может выглядеть следующим образом:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- Перечень экземпляров -->
<vms>
    <vm>
        <vmid>0101af9811012</vmid>
        <type>t1.nano</type>
    </vm>
    <vm>
        <vmid>0102bg8908023</vmid>
        <type>t1.micro</type>
    </vm>
</vms>
```

Этот пример моделирует информацию, которую вы можете получить от API для управления облачными вычислениями, в котором перечислены существующие виртуальных машин.

Тело документа XML

На данный момент игнорируйте первую строку документа, которая является специальной частью, известной как пролог (подробнее об этом ниже), и вторую строку, которая содержит комментарий. Оставшаяся часть документа называется телом.

Обратите внимание, что отдельные элементы данных в теле (читаемые строки символов) окружены симметричными парами тегов: открывающий тег, окруженный символами `<` и `>`, и закрывающий тег, похожий на него, но с символом "`/`" (косая черта) перед именем закрывающего тега.

Обратите внимание, что некоторые пары тегов окружают несколько экземпляров помеченных данных (например, тег `<vm>` и соответствующий тег `<vm>`). Основное тело документа в целом всегда окружено крайней парой тегов (например, пара тегов `<vms>...</vms>`) или корневой парой тегов.

Структура тела документа напоминает дерево, от корня которого отходят ветви, содержащие возможные дальнейшие ответвления, и, наконец, узлы листьев, содержащие фактические данные. Двигаясь вверх по дереву, каждая пара тегов в XML-документе имеет родительскую пару тегов, и так далее, пока вы не достигнете корневой пары тегов.

Определяемые пользователем имена тегов

Имена тегов XML определяются пользователем. Если вы составляете XML для собственного приложения, лучше всего выбирать имена тегов, которые четко выражают смысл элементов данных, их взаимосвязи и иерархию. Имена тегов можно повторять по мере необходимости, чтобы заключить в них несколько элементов данных (или маркированных групп элементов) одного типа. Например, можно создать несколько пар тегов `<vm>`, чтобы заключить в них дополнительные группировки `<vmid>` и `<type>`.

При потреблении XML из API имена тегов и их значения обычно документируются поставщиком API и могут представлять собой использование, определенное в схеме публичного пространства имен.

Кодировка специальных символов

Данные в `XML` передаются в виде читаемого текста. Как и в большинстве языков программирования, кодировка специальных символов в полях данных `XML` представляет определенные трудности.

Например, поле данных не может содержать текст, включающий символы `<` или `>`, используемые в `XML` для разграничения тегов. Если вы пишете собственный `XML` (без схемы), то для кодирования таких символов обычно используются кодировки сущностей `HTML`. В этом случае символы можно заменить эквивалентными им кодировками сущностей `<` и `>`. Подобную стратегию можно использовать для представления широкого спектра специальных символов, лигатур и других сущностей.

Обратите внимание, что, если вы используете XML в соответствии с требованиями схемы или определенного словаря и иерархии имен тегов (это часто бывает при взаимодействии с API, такими как NETCONF), вам не разрешается использовать сущности HTML. В редких случаях, когда требуются специальные символы, вы можете использовать их числовые представления, которые для знаков "меньше" и "больше" - < и > соответственно.

Чтобы избежать необходимости индивидуального поиска и преобразования специальных символов, можно включать целые необработанные строки символов в XML-файлы, окружая их так называемыми блоками CDATA. Вот пример:

```
<hungarian_sentence>
    <![CDATA[Minden személynek joga van a neveléshez.]]>
</hungarian_sentence>
```

XML Пролог

Пролог XML — это первая строка в XML-файле. Она имеет специальный формат, заключенный в скобки <? и ?>. Она содержит имя тега xml и атрибуты, указывающие версию и кодировку символов. Обычно версия - "1.0", а кодировка символов в большинстве случаев - "UTF-8", в противном случае - "UTF-16". Включение пролога и кодировки может быть важным для того, чтобы ваши XML-документы надежно интерпретировались синтаксическими анализаторами, редакторами и другими программами.

Комментарии в XML

Файлы XML могут включать комментарии, используя ту же конвенцию комментирования, которая используется в документах HTML. Например:

```
<!-- This is an XML comment. It can go anywhere -->
```

XML Атрибуты

XML позволяет встраивать атрибуты в теги для передачи дополнительной информации. В следующем примере номер версии XML и кодировка символов находятся внутри тега xml. Однако элементы vmid и type также могут быть включены в тег xml в качестве атрибутов:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Перечень экземпляров -->
<vms>
    <vm vmid="0101af9811012" type="t1.nano" />
    <vm vmid="0102bg8908023" type="t1.micro"/>
</vms>
```

Здесь есть несколько моментов, на которые следует обратить внимание:

- Значения атрибутов всегда должны заключаться в одинарные или двойные кавычки.
- Элемент может иметь несколько атрибутов, но только один атрибут для каждого конкретного имени.

– Если элемент не имеет содержимого, можно использовать сокращенное обозначение, при котором косая черта ставится внутри открытого тега, а не в закрывающем теге.

Пространства имён XML

Некоторые сообщения и документы XML должны включать ссылки на определенные пространства имен *(namespace)*, с целью определения конкретных имён и то, как они должны использоваться в различных контекстах. Пространства имен определяются IETF и другими органами Интернета, организациями и другими субъектами, а их схемы обычно размещаются в Интернете в виде общедоступных документов. Идентифицируются с помощью унифицированных имен ресурсов (URN), используемых для того, чтобы сделать постоянные документы доступными без необходимости для ищущего знать их местоположение.

Приведенный ниже пример показывает использование пространства имен, определенного как значение атрибута xmlns, для утверждения того, что содержимое удаленного вызова процедуры XML должно интерпретироваться в соответствии с устаревшим стандартом NETCONF 1.0. Этот пример показывает инструкцию NETCONF по удаленному вызову процедуры в XML. Атрибуты в открывающем теге rpc обозначают идентификатор сообщения и пространство имен XML, которое должно использоваться для интерпретации смысла содержащихся тегов. В данном случае вы просите удаленный объект завершить определенную сессию. XML-схема NETCONF документирована IETF.

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <kill-session>
        <session-id>4</session-id>
    </kill-session>
</rpc>
```

Интерпретация XML

В приведенном выше примере то, что представлено, подразумевается, как список или одномерный массив (называемый "экземплярами") объектов (каждый из которых идентифицируется как "экземпляр" с помощью тегов в скобках). Каждый объект экземпляра содержит две пары ключ-значение, обозначающие уникальный идентификатор экземпляра и тип сервера VM. Семантически эквивалентная структура данных Python может быть объявлена следующим образом:

```
vms [
        {
            {"vmid": "0101af9811012"},
            {"type": "t1.nano"}
        },
        {
            {"vmid": "0102bg8908023"},
```

```
            {"type": "t1.micro"}
        }
    ]
```

Проблема заключается в том, что в XML нет возможности намеренно указать, что определенное расположение тегов и данных должно интерпретироваться как список. Поэтому мы должны интерпретировать намерение автора XML при переводе. Сопоставление древовидных структур XML с более эффективными представлениями, возможными в различных компьютерных языках, требует понимания семантики данных. Это в меньшей степени относится к более современным форматам данных, таким как JSON и YAML, которые структурированы таким образом, что хорошо отображаются на распространенные простые и составные типы данных в популярных языках программирования.

В этом случае теги <vm>, заключающие в скобки данные каждого экземпляра (id и тип), сворачиваются в пользу использования обычных скобок ({}) для группировки. В результате вы получаете список объектов Python (которые можно назвать "объектами vm", но которые не названы явно в этом объявлении), каждый из которых содержит два подобъекта, каждый из которых содержит пару ключ/значение.

JSON

JSON, или JavaScript Object Notation — это формат данных, полученный на основе способа записи сложных объектных литералов в JavaScript (который, в свою очередь, похож на способ записи объектных литералов в Python). Имена файлов JSON обычно заканчиваются на ".json". Вот пример файла JSON, содержащий несколько пар ключ/значение. Обратите внимание, что два значения являются текстовыми строками, одно — булевым значением, а два — массивами:

```
{
    "edit-config":
    {
        "default-operation": "merge",
        "test-operation": "set",
        "some-integers": [2,3,5,7,9],
        "a-boolean": true,
        "more-numbers": [2.25E+2,-1.0735],
    }
}
```

Основные типы данных JSON

Основные типы данных JSON включают числа (записанные как положительные и отрицательные целые числа, как плавающие числа с десятичной дробью или в научной нотации), строки, логические ("true" и "false") или нули (значения, оставленные пустыми).

Объекты JSON

Как и в JavaScript, отдельные объекты в JSON состоят из пар ключ/значение, которые могут быть окружены скобками, по отдельности:

```
{"keyname": "value"}
```

В этом примере изображен объект со строковым значением (например, слово 'value'). Число или булево значение не будет заключено в кавычки.

```
{"number": 228}
{"boolean": false}
```

Мапы и Списки JSON

Объекты также могут содержать несколько пар ключ/значение, разделенных запятыми, создавая структуры, эквивалентные сложным объектам JavaScript или словарям Python. В этом случае каждая отдельная пара ключ/значение не нуждается в собственном наборе скобок, но весь объект нуждается. В приведенном выше примере ключ "edit-config" идентифицирует объект, содержащий пять пар ключ/значение.

Составные объекты JSON могут быть глубоко вложенными, со сложной структурой.

JSON также может выражать упорядоченные массивы (или "списки") данных или объектов. В приведенном выше примере ключи "some-integers" и "more-numbers" идентифицируют такие массивы.

Отсутствие комментариев в JSON

В отличие от XML и YAML, JSON не поддерживает какой-либо стандартный метод включения в код не разобранных комментариев *(unparsed comments)*.

Незначительные Пробелы

Пробельные символы в JSON не имеют значения, и файлы могут быть отступы с использованием табуляции или пробелов, как предпочтительно, или вообще без них *(что на самом деле является плохой идеей)*. Это делает формат эффективным *(пробелы могут быть удалены без ущерба для смысла)* и надежным для использования в командных строках и в других “сценариях”, где пробельные символы могут быть легко потеряны при операциях удаления и вставки.

YAML

YAML, сокращение от "YAML Ain't Markup Language", представляет собой супермножество *(математика)* JSON, разработанное для ещё более лёгкой читаемости человеком. Он становится всё более распространенным форматом для файлов конфигурации, и особенно для написания декларативных шаблонов автоматизации для таких инструментов, как Ansible.

Как супермножество JSON, парсеры YAML, как правило, могут разбирать документы JSON (но не наоборот). Из-за этого YAML лучше JSON в некоторых задачах, включая возможность встраивать JSON непосредственно (включая

кавычки) в файлы YAML. JSON тоже можно вставлять в JSON-файлы, но кавычки должны быть экранированы обратными слешами \" или закодированы как HTML-символы &quote;

Вот версия файла JSON, выраженная в YAML. Давайте с его в помощью постараемся понять, как работает YAML:

```
---
edit-config:
  a-boolean: true
  default-operation: merge
  more-numbers:
    - 225.0
    - -1.0735
  some-integers:
    - 2
    - 3
    - 5
    - 7
    - 9
  test-operation: set
...
```

Структура файла YAML

Как показано в примере, файлы YAML традиционно открываются тремя тире (--- на одной строке) и заканчиваются тремя точками (... также на одной строке). YAML также поддерживает понятие нескольких "документов" внутри одного физического файла, в данном случае отделяя каждый документ тремя тире на отдельной строке.

Типы данных YAML

Основные типы данных YAML включают числа (записываемые как целые положительные и отрицательные числа, как плавающие числа с десятичной дробью или в научной системе счисления), строки, логические (true и false) или нули (значение, оставленное пустым).

Строковые значения в YAML часто остаются без кавычек. Кавычки требуются только тогда, когда строки содержат символы, имеющие значение в YAML. Например, { , скобка, за которой следует пробел, указывает на начало карты. Обратные косые черты и другие специальные символы или строки также необходимо учитывать. Если вы окружите текст двойными кавычками, то сможете вывести специальные символы в строке с помощью выражений с обратной косой чертой, например, \n для новой строки.

YAML также предлагает удобные способы кодирования многострочных строковых литералов (подробнее ниже).

Основные объекты

В YAML основные (и сложные) типы данных приравниваются к ключам. Ключи обычно не заключаются в кавычки, хотя их можно заключить в кавычки, если они содержат двоеточия (:) или некоторые другие специальные

символы. Ключи также не обязательно должны начинаться с буквы, хотя обе эти особенности противоречат требованиям большинства языков программирования, поэтому по возможности от них лучше воздержаться.

Двоеточие (:) разделяет ключ и значение:

```
my_integer: 2
my_float: 2.1
my_exponent: 2e+5
'my_complex:key' : "значение\n заключённое в кавычки"
0.2 : "А вы тоже верите, что это ключ?"
my_boolean: true
my_null: null # в противном случае, может быть интерпретирована
как пустая строка
```

Отступы в YAML и структура файлов

В YAML не используются скобки или содержащие пары тегов, вместо этого иерархия обозначается с помощью отступов. Элементы, отступающие под меткой, являются "членами" этого элемента с меткой.

Величина отступа зависит от вас. Там, где требуется отступ, можно использовать не более одного пробела, хотя лучшей практикой является использование двух пробелов на каждый уровень отступа. Главное — быть абсолютно последовательным и использовать пробелы, а не табуляцию.

Мапы и Списки

YAML легко представляет более сложные типы данных, такие как карты, содержащие несколько пар ключ/значение (эквивалент словарей в Python) и упорядоченные списки.

Мапы *(Maps, Хэш-таблицы, отображения или ассоциативный массив как вам больше нравится)* обычно выражаются в нескольких строках, начиная с метки key и двоеточия, за которыми следуют члены, отступающие на последующие строки:

```
mymap:
  myfirstkey: 1337
  mysecondkey: Пажилой Гиббон
```

Списки (массивы) представляются аналогичным образом, но с опционально индексированными членами, перед которыми ставится тире и пробел:

```
mylist:
  - 1
  - 2
  - 3
```

Мапы и списки также могут быть представлены в так называемом "flow синтаксисе", который очень похож на JavaScript или Python:

```
mymap: { myfirstkey: 1337, mysecondkey: Пажилой Гиббон}
mylist: [1, 2, 3]
```

Длинные строки

Вы можете представлять длинные строки в YAML, используя "сворачивающийся" синтаксис, в котором предполагается, что разрывы строк будут заменены пробелами при parse/consume файла, или не сворачивающийся синтаксис. Длинные строки не могут содержать экранированные специальные символы, но теоретически могут содержать двоеточия, хотя некоторые программы не соблюдают это правило.

```
mylongstring: >
  Here in my garage, just bought this new Lamborghini here.
  It’s fun to drive up here in the Hollywood hills.
  But you know what I like more than materialistic things? Knowledge.
myotherlongstring: |
  In fact, I’m a lot more proud of these seven new bookshelves
  that I had to get installed to hold two thousand new books that I
bought.
  It’s like the billionaire Warren Buffett says,
  “the more you learn, the more you earn.”
```

Обратите внимание на разницу в двух приведенных выше примерах. Индикатор больше чем (>) дает нам синтаксис складывания *(Разрывы будут заменены пробелами)*, а pipe (|) — нет *(Разрывы будут сохранятсяь как новые строки)*.

Коментарии

Комментарии в YAML могут быть вставлены в любое место, кроме литерала длинной строки, и им предшествует знак октотропа и пробел:

```
# это пажилой комментарий
```

Дополнительные возможности YAML

YAML имеет множество дополнительных возможностей, которые чаще всего встречаются при использовании его в контексте конкретных языков, таких как Python, или при конвертации в JSON или другие форматы. Например, YAML 1.2 поддерживает схемы и теги, которые можно использовать для однозначной интерпретации значений. Например, чтобы заставить число интерпретироваться как строка, можно использовать строку !!str, которая является частью схемы YAML "Failsafe":

```
mynumericstring: !!str 0.1415
```

Парсинг и Сериализация

Парсинг или Разбор означает анализ сообщения, разбиение его на составные части и понимание их назначения в контексте. Когда сообщения передаются между компьютерами, они передаются в виде потока символов, что фактически является строкой. Её необходимо разобрать на семантически эквивалентную структуру данных, содержащую данные распознанных типов (таких как целые числа, плавающие числа, строки и т.д.), прежде чем приложение сможет интерпретировать данные и действовать с ними.

Сериализация примерно противоположна синтаксическому анализу. Например, для передачи информации через интерфейс `REST` вас могут попросить взять локально хранящиеся данные (например, данные, хранящиеся в словарях `Python`) и вывести их в виде эквивалентного `JSON`, `YAML` или `XML` в строковой форме для представления удаленному ресурсу.

Пример

Например, представьте, что вы хотите отправить начальный запрос на некоторую удаленную конечную точку `REST`, чтобы узнать о состоянии запущенных служб. Для этого, как правило, необходимо пройти аутентификацию в `REST API`, указав свое имя пользователя (электронную почту), а также ключ разрешения, полученный в результате предыдущей транзакции. Вы могли бы хранить имя пользователя и ключ в словаре `Python`, например, так:

```
auth = {
    "user": {
        "username": "myemail@mydomain.com",
        "key": "90823ff08409408aebcf4320384"
    }
}
```

Но `REST API` требует, чтобы эти значения были представлены в виде `XML` в строковой форме, добавленной к вашему запросу в качестве значения пары ключ/значение под названием "auth":

```
https://myservice.com/status/services?auth=<XML string containing
username and key>
```

Сам `XML`, возможно, должен принять такой формат, при этом значения ключей `Python` преобразуются в одноименные пары тегов, заключающие значения данных:

```
<user>
    <username>myemail@mydomain.com</username>
    <key>90823ff08409408aebcf4320384</key>
</user>
```

Обычно вы используете функцию сериализации (из библиотеки Python) для вывода структуры данных auth в виде строки в формате XML, добавляя ее в запрос:

```
import dicttoxml    # библиотека для сериализации
import requests     # библиотека http-запросов

auth = {            # Словарь, содержащий информацию об
аутентификации
    "user": {
        "username": "myemail@mydomain.com",
        "key": "90823ff07409408aebcf4320384"
    }
}

get_services_query = "https://myservice.com/status/services"
xmlstring = dicttoxml(auth)       # преобразование словаря в XML в
строковой форме
myresponse = requests.get(get_services_query,auth=xmlstring)  #
служба запросов
```

В этот момент служба может ответить, установив переменную myresponse в строку, подобную следующей, содержащую имена и статусы служб в формате XML:

```
<services>
    <service>
        <name>Service A</name>
        <status>Running</status>
    </service>
    <service>
        <name>Service B</name>
        <status>Idle</status>
    </service>
</services>
```

Затем вам нужно будет разобрать XML, чтобы извлечь информацию в форму, к которой Python мог бы получить удобный доступ.

```
import untangle     # библиотека xml парсера

myreponse_python = untangle.parse(myresponse)

print(myreponse_python.services.service[1].name.cdata,
      myreponse_python.services.service[1].status.cdata)
```

В этом случае библиотека untangle будет разбирать XML в словарь, корневой элемент которого (services) содержит список (service[]) пар ключ/значение объектных элементов, обозначающих имя и статус каждой службы. Затем можно получить доступ к значению 'cdata' элементов, чтобы

получить текстовое содержимое каждого узла листа `XML`. Приведенный выше код выведет:

```
Service B  Idle
```

Популярные языки программирования, такие как Python, обычно включают простые в использовании функции синтаксического анализа, которые могут принимать данные, возвращаемые функцией ввода-вывода, и создавать семантически эквивалентную внутреннюю структуру данных, содержащую допустимые типизированные данные. На стороне исходящих данных они содержат сериализаторы, которые делают обратное, превращая внутренние структуры данных в семантически эквивалентные сообщения, отформатированные как символьные строки.

Резюме

Разработка программного обеспечения

Жизненный цикл разработки программного обеспечения (SDLC) — это процесс разработки программного обеспечения, начиная с идеи и заканчивая поставкой. Этот процесс состоит из шести фаз. Каждая фаза использует результаты предыдущей фазы: 1. Требования и анализ, 2. Проектирование, 3. Реализация, 4. Тестирование, 5. Развертывание и 6. Поддержка. Три популярные модели разработки программного обеспечения: водопад, Agile и Lean:

- Водопад — традиционная модель разработки программного обеспечения. Каждый этап не может накладываться друг на друга и должен быть завершен до перехода к следующему этапу.
- Agile Scrum — В регби термин "схватка" описывает момент игры, когда игроки собираются вместе и пытаются завладеть мячом. Методология Scrum фокусируется на небольших, самоорганизующихся командах, которые ежедневно проводят короткие встречи и работают в итеративных спринтах, постоянно адаптируя результаты для удовлетворения меняющихся требований.
- Lean — основанный на бережливом производстве, метод Lean подчеркивает устранение напрасных усилий при планировании и выполнении, а также снижение когнитивной нагрузки на программиста.

Дизайн Паттерны Программ

Модели проектирования программного обеспечения — это лучшие практические решения для решения общих проблем при разработке программного обеспечения. Паттерны проектирования не зависят от языка. В своей книге "Паттерны проектирования" "Банда четырех" разделила паттерны на три основные категории:

- **Порождающие** — паттерны, используемые для руководства, упрощения и абстрагирования создания программных объектов в

масштабе. *Простыми словами, они беспокоятся о гибком создании объектов без внесения в программу лишних зависимостей.*

- **Структурные** — паттерны, описывающие надежные способы использования объектов и классов для различных видов программных проектов. *То есть показывают различные способы построения связей между объектами.*
- **Поведенческие** — паттерны, описывающие, как объекты могут взаимодействовать и работать вместе для решения привычных задач в программной инженерии. Или по-человечески заботятся об эффективной коммуникации между объектами.

Паттерн проектирования Наблюдатель — конструкция уведомления о подписке, которая позволяет объектам (наблюдателям или подписчикам) получать события, когда происходят изменения в объекте (субъекте или издателе), за которым они наблюдают.

Шаблон проектирования Model-View-Controller (MVC) иногда рассматривается как архитектурный шаблон проектирования. Его цель — упростить разработку приложений, которые зависят от графических пользовательских интерфейсов.

Системы Контроля Версий

Контроль версий — это способ управления изменениями в наборе файлов с целью сохранения истории этих изменений. Существует три типа систем контроля версий: Локальная, Централизованная и Распределенная.

Git — это реализация распределенной системы контроля версий с открытым исходным кодом. Git имеет два типа репозиториев — локальный и удаленный. Разветвление позволяет пользователям работать над кодом независимо, не затрагивая основной код в репозитории. В дополнение к функциональности распределенного контроля версий и управления исходным кодом Git, GitHub также предоставляет дополнительные возможности, такие как: обзор кода, документация, управление проектами, отслеживание ошибок и запросы функций. После установки Git на клиентскую машину его необходимо настроить. Git предоставляет команду git config для получения и установки глобальных настроек Git или опций репозитория. Git имеет множество других команд, которые вы можете использовать, включая множество опций ветвления. Разработчики используют специальный файл .diff, чтобы показать, как изменились две разные версии файла.

Основы написания кода.

Чистый код — это результат того, что разработчики стараются сделать свой код легко читаемым и понятным для других разработчиков. Методы и функции имеют одну и ту же концепцию; это блоки кода, которые выполняют задачи при выполнении. Если метод или функция не выполняются, то эти задачи не будут выполнены. Модули — это способ создания независимых и самодостаточных частей кода, которые можно использовать повторно. В большинстве языков ООП и в Python классы являются средством объединения

данных и функциональности. Каждое объявление класса определяет новый тип объекта.

Code Review и тестирование

Code review — это когда разработчики просматривают кодовую базу, подмножество кода или конкретные изменения кода и предоставляют отзывы. Наиболее распространенные типы процессов обзора кода включают: Формальный обзор кода, обзор кода на основе изменений, обзор кода "через плечо" и обход по электронной почте.

Тестирование программного обеспечения подразделяется на две общие категории: функциональное и нефункциональное. Детальное функциональное тестирование небольших фрагментов кода (строк, блоков, функций, классов и других компонентов в изоляции) обычно называется модульным тестированием. После модульного тестирования идет интеграционное тестирование, которое позволяет убедиться в том, что все эти отдельные части правильно сочетаются друг с другом, образуя полноценное приложение. Разработка, ориентированная на тестирование (иногда называемая Test-First Development) — это тестирование для подтверждения замысла проекта в свете требований. Это означает написание кода тестирования до написания кода приложения. Выразив требования в тестовом коде, разработчики могут писать код приложения до тех пор, пока он не пройдет тесты.

Вникаем в форматы данных

Сегодня тремя наиболее популярными стандартными форматами для обмена информацией с удаленными API являются XML, JSON и YAML.

Расширяемый язык разметки (XML) является производным от структурированного, обобщенного языка разметки (SGML), а также родителем языка гипертекстовой разметки (HTML). XML — это общая методология для обертывания текстовых данных в симметричные теги для обозначения семантики. Имена файлов XML обычно заканчиваются на ".xml".

JavaScript Object Notation JSON) — формат данных, основанный на способе записи сложных объектных литералов в JavaScript. Имена файлов JSON обычно заканчиваются на ".json".

YAML Ain't Markup Language (YAML) — это супермножество JSON, разработанное для еще более легкой читаемости человеком.

Разбор означает анализ сообщения, разбиение его на составные части и понимание их назначения в контексте. Сериализация — это примерно противоположность разбору.

Дополненная Глава 3

Если вы планируете автоматизировать свою сеть (или даже только ее часть), вам понадобится умение создавать и устранять неполадки в интерфейсах прикладного программирования (API). API определяют способ взаимодействия пользователей, разработчиков и других приложений с компонентами приложения. Поскольку API-интерфейсов так много, вы должны быть уверены, что ж/б понимаете их основы. В этом модуле основное внимание мы уделим REST API. Cisco имеет множество линеек продуктов с такими интерфейсами для различных случаев использования, будь то сбор данных, вызов служб или прослушивание событий.

Стили проектирования API

Типы стилей дизайна

API могут предоставляться одним из двух способов: синхронно или асинхронно. Вам необходимо понимать эту разницу между, поскольку приложение, использующее API, по-разному управляет ответом в зависимости от его дизайна. Каждый дизайн имеет свою цель, но также и свой набор сложностей на стороне клиента и/или сервера. Набор API в продукте может состоять как из синхронных, так и асинхронных конструкций, где каждая конструкция API не зависит от других. Однако для лучшей практики логика, лежащая в основе дизайна, должна быть последовательной.

Синхронные API

Билеты продаются в порядке поступления и обслуживания. Это синхронный процесс (Рисунок 60).

Рисунок 60 — Синхронные API

Синхронные API отвечают на запрос напрямую, обычно предоставляя данные (или другой подходящий ответ) немедленно.

Когда API являются синхронными?

API обычно проектируются как синхронные, когда данные для запроса легко доступны, например, когда данные хранятся в базе данных или во внутренней памяти. Сервер может мгновенно получить эти данные и немедленно ответить на запрос.

Преимущества синхронного дизайна API

Синхронные API позволяют приложению получать данные немедленно. Если API разработан правильно, приложение будет иметь более высокую производительность, поскольку все происходит быстро. Однако если API спроектирован неправильно, запрос API станет узким местом, поскольку приложению придется ждать ответа.

Обработка на стороне клиента

Приложение, выполняющее API-запрос, должно дождаться ответа, прежде чем выполнять какие-либо дополнительные задачи по выполнению кода.

Асинхронные API

Асинхронные API предоставляют ответ, свидетельствующий о том, что запрос был получен, но этот ответ не содержит фактических данных. Сервер обрабатывает запрос, что может занять некоторое время, и посылает уведомление (или запускает обратный вызов) с данными после обработки запроса. После этого клиент может действовать на основе полученных данных (Рисунок 61).

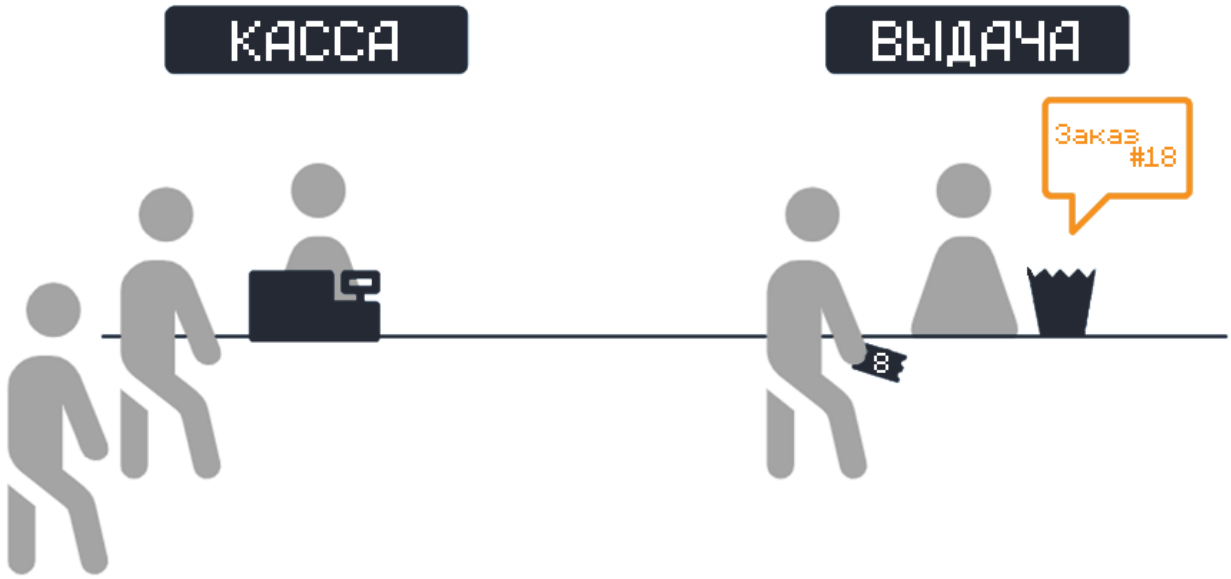


Рисунок 61 — Асинхронные API

Когда API являются асинхронными?

API обычно проектируются асинхронными, когда запрос представляет собой действие, на обработку которого серверу требуется некоторое время, или если данные не являются легкодоступными. Например, если серверу необходимо сделать запрос к удаленной службе для получения данных, он не может гарантировать, что получит их немедленно, чтобы отправить обратно

клиенту. То, что API является асинхронным, не обязательно означает, что клиент не получит данные немедленно. Это означает лишь то, что немедленный ответ с данными не гарантирован.

Преимущества асинхронного API

Асинхронные API позволяют приложению продолжать выполнение без блокировки на время, необходимое серверу для обработки запроса. В результате производительность приложения может быть выше, поскольку оно может работать в многозадачном режиме и выполнять другие запросы. Однако излишнее или чрезмерное использование асинхронных вызовов может иметь обратный эффект на производительность.

Обработка на стороне клиента

При асинхронной обработке, API на стороне сервера определяет то, что вы хотите сделать на стороне клиента. Иногда клиент может создать слушатель или механизм обратного вызова для получения этих уведомлений и обработки их при получении. В зависимости от дизайна приложения, клиенту может также понадобиться очередь для хранения запросов, чтобы поддерживать порядок их обработки. В других конструкциях API клиенту необходим механизм опроса для получения информации о состоянии и ходе выполнения запроса.

Архитектурные стили API

Обычные архитектурные стили

Приложение определяет, как третьи стороны взаимодействуют с ним, что означает, что не существует "стандартного" способа создания API. Однако, несмотря на то, что приложение технически может предоставлять бессистемный интерфейс, лучшей практикой является следование стандартам, протоколам и определенным архитектурным стилям. Это значительно упрощает изучение и понимание API для потребителей API, поскольку концепции уже знакомы.

Три наиболее популярных архитектурных стиля API — это RPC, SOAP и REST.

RPC

Удаленный вызов процедур (RPC) — это модель запрос-ответ, которая позволяет приложению (выступающему в роли клиента) осуществлять процедурный вызов другого приложения (выступающего в роли сервера). Серверное приложение обычно расположено на другой системе в сети.

При использовании RPC клиент обычно не знает, что запрос процедуры выполняется удаленно, поскольку запрос выполняется на уровне, скрывающем эти детали. Для клиента эти вызовы процедур являются просто действиями, которые он хочет выполнить. Другими словами, для клиента удаленный вызов процедуры — это просто метод с аргументами. Когда он вызывается, метод выполняется, и результаты возвращаются.

На рисунке 62 показаны два столбца. Столбец слева обозначен как Caller (клиентский процесс), а столбец справа — как Callee (серверный процесс). Сплошная стрелка идет от вызывающего процесса к точке процедуры вызова. Сплошная линия с надписью request указывает на точку получения запроса со стороны вызывающей стороны. Над этой точкой проходит пунктирная линия от callee к точке с надписью wait for request и еще одна пунктирная линия вниз к точке receive request. Через точку с надписью process request в колонке callee проходит сплошная линия со стрелкой, указывающей на send response в колонке callee. В колонке вызывающего абонента есть сплошная линия со стрелкой, идущая от отправленного ответа к полученному ответу, с надписью response. На стороне вызывающей стороны есть пунктирная линия, идущая от точки процедуры вызова к точке получения ответа и через точку с надписью wait for response. Под колонкой вызывающей стороны находится сплошная стрелка, направленная вниз, через точку с надписью возобновить выполнение. Под точкой отправки ответа на стороне получателя есть пунктирная линия, проходящая через точку с надписью wait for request.

Рисунок 62 — RPC клиент-серверная модель запроса/ответа

При распространённом использовании RPC клиент делает синхронный запрос к серверу и блокируется, пока сервер обрабатывает запрос. Когда сервер завершает обработку запроса, он направляет ответ клиенту, который разблокирует свой процесс. (Это не относится к асинхронным запросам).

RPC — это стиль API, который может применяться к различным транспортным протоколам. Примеры реализации включают:

- XML-RPC
- JSON-RPC
- NFS (сетевая файловая система)
- Простой протокол доступа к объектам (SOAP)

SOAP

SOAP — это протокол обмена сообщениями на основе XML, разработанный компанией Microsoft. Он используется для связи мульти платформенных приложений. SOAP обычно используется с HyperText Transfer Protocol (HTTP), но может быть применен и к другим протоколам. SOAP является независимым, расширяемым и нейтральным.

Независимый

SOAP был разработан для того, чтобы все типы приложений могли взаимодействовать друг с другом. Приложения могут быть созданы с использованием различных языков программирования, могут работать на различных операционных системах и быть настолько разными, насколько это возможно.

Расширяемый

SOAP сам по себе считается приложением XML, поэтому на его основе можно создавать расширения. Расширяемость означает, что вы можете добавлять такие функции, как надежность и безопасность.

Нейтральный

SOAP может использоваться по любому протоколу, включая HTTP, SMTP, TCP, UDP или JMS.

Сообщения SOAP

Сообщение SOAP — это просто XML-документ, который может содержать четыре элемента:

1. Envelope;
2. Header;
3. Body;
4. Fault.

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header/>
    <soap:Body>
        <soap:Fault>
            <faultcode>soap:Server</faultcode>
            <faultstring>Query request too large.</faultstring>
        </soap:Fault>
    </soap:Body>
</soap:Envelope>
```

Пример SOAP-сообщения

`Envelope` должен быть корневым элементом XML-документа. В `Envelope` предоставленное пространство имен говорит о том, что XML-документ является сообщением `SOAP`.

Header

Заголовок является необязательным элементом, но при наличии, должен быть первым дочерним элементом элемента `Envelope`. Как и большинство

других заголовков, он содержит специфическую для приложения информацию, такую как авторизация, специфические атрибуты SOAP или любые атрибуты, определенные приложением.

Body

Тело содержит данные, переданные получателю. Эти данные должны быть в формате XML и находиться в собственном пространстве имен.

Fault

Элемент `Fault` также является необязательным, но при наличии в документе должен быть дочерним элементом элемента `Body`. В сообщении SOAP может быть только один элемент `Fault`. Элемент `Fault` предоставляет информацию об ошибке и/или статусе.

REST

Representational **S**tate **T**ransfer (REST) — это архитектурный стиль, автором которого является американский компьютерный ученый Рой Томас Филдинг , о котором он рассказывает в 5 главе своей докторской диссертации "Архитектурные стили и проектирование сетевых архитектур программного обеспечения" вышедшей 2000 году. Рой определяет REST как гибридный стиль, полученный из нескольких сетевых архитектурных стилей, описанных им в другом месте работы, "в сочетании с дополнительными ограничениями, которые определяют единый интерфейс соединителя".

Мы подробно рассмотрим REST в следующих двух разделах, но сейчас давайте обратимся к основам.

В диссертации Филдинга он устанавливает шесть ограничений, применяемых к элементам архитектуры:

- Модель клиент-сервер (Client-Server);
- Отсутствие состояния (Stateless);
- Кэширование (Cache);
- Единообразие интерфейса (Uniform Interface);
- Слои (Layered System)
- Код по требованию (Code-On-Demand).

Эти шесть ограничений могут быть применены к любому протоколу, и в ситуации, когда все эти требования применены, вы часто услышите, что это RESTful.

Клиент-сервер

Клиент и сервер должны быть независимы друг от друга, что позволяет создавать клиент для различных платформ и упрощает компоненты на стороне сервера рисунок 63.

Рисунок 63 — REST модель клиент-сервер

Отсутствие состояния

Запросы от клиента к серверу должны содержать всю информацию, необходимую серверу для выполнения запроса. Сервер не может содержать состояния сеанса (Рисунок 64).

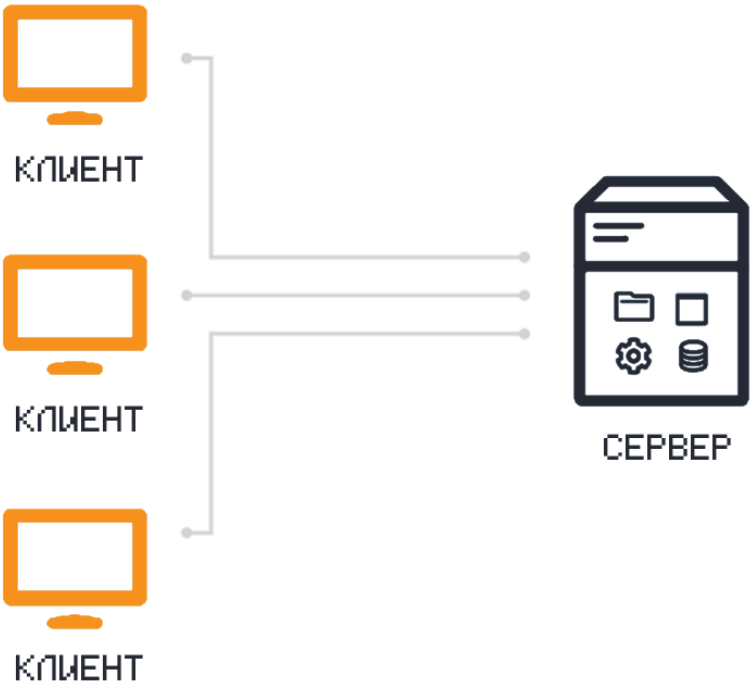


Рисунок 64 — REST Отсутствие состояния

Кэш

В ответах от сервера должно быть указано, является ли ответ кэшируемым или некэшируемым. Если он кэшируемый, клиент может использовать данные из ответа для последующих запросов (Рисунок 65).

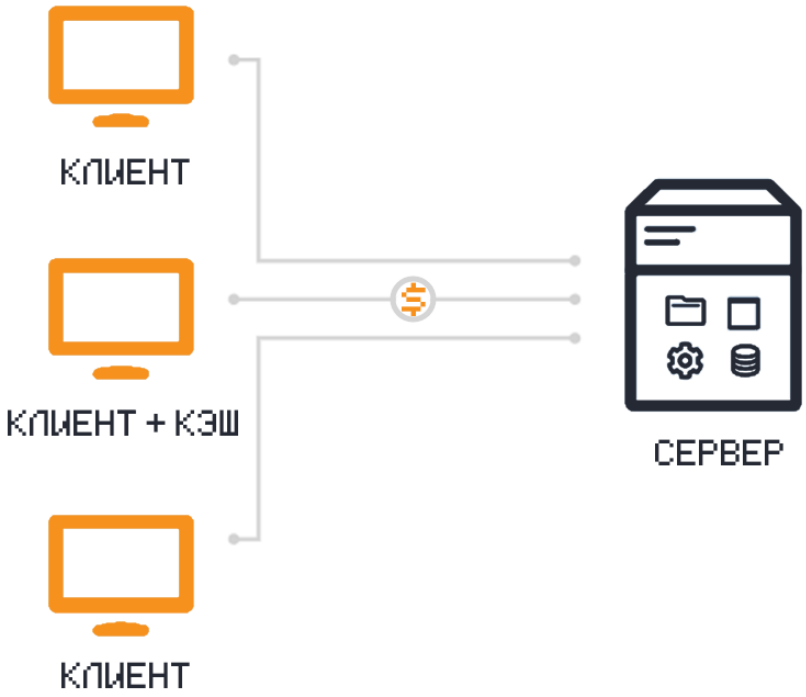


Рисунок 65 — REST Кэширование

Единообразный интерфейс

Интерфейс между клиентом и сервером должен соответствовать этим четырем принципам:

- **Идентификация ресурсов** — ресурс должен быть идентифицирован в запросе как отдельный объект, к которому сервер будет обращаться и которым он будет манипулировать. Ресурсом может быть любая информация, например, документ, изображение, человек, коллекция других ресурсов и так далее. Например, в запросе на изменение пароля пользователя должен быть идентифицирован отдельный пользователь.
- **Манипулирование ресурсами через представления** — Клиент получает от сервера представление ресурса. Это представление должно содержать достаточно данных или метаданных, чтобы клиент мог манипулировать ресурсом. Например, запрос, предназначенный для заполнения профиля пользователя, должен содержать информацию о профиле. Представление может быть точной копией ресурса на сервере или даже его упрощенной версией, но не может быть, например, просто идентификатором дополнительного ресурса.
- **Само описательные сообщения** — каждое сообщение должно содержать всю информацию, необходимую получателю для обработки сообщения. Примерами информации могут быть:
    1. Тип протокола;
    2. Формат данных сообщения;
    3. Запрашиваемая операция.

– **Гипермедиа как механизм состояния приложения** — Данные, отправляемые сервером, должны включать дополнительные действия и ресурсы, доступные клиенту для получения дополнительной информации о ресурсе.

Многослойная система

Система состоит из различных иерархических слоев, в которых каждый слой предоставляет услуги только тому слою, который находится над ним. В результате он потребляет услуги от нижележащего слоя .

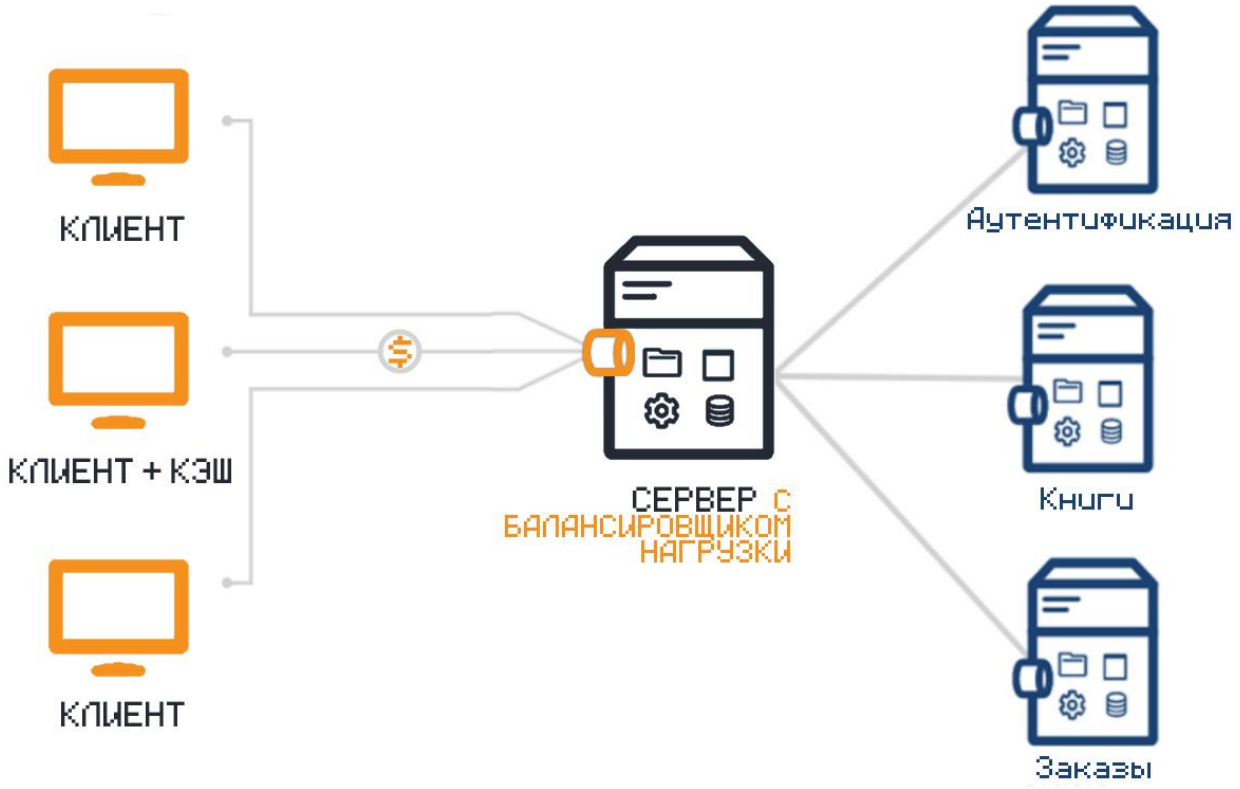


Рисунок 66 — REST Многослойная система

Код по требованию

Это ограничение является необязательным и относится к тому факту, что информация, возвращаемая службой REST, может включать исполняемый код (например, JavaScript) или ссылки на такой код, предназначенный для полезного расширения функциональности клиента. Например, платежная служба может использовать REST для предоставления ссылок на свои опубликованные библиотеки JavaScript для осуществления платежей. Затем эти файлы JavaScript могут быть загружены и (если они признаны надежными) выполнены клиентским приложением. Это избавляет разработчиков клиента от необходимости создавать и поддерживать отдельный код обработки платежей, а также управлять изменениями зависимостей, которые могут время от времени ломать такой код.

Ограничение является необязательным, поскольку выполнение стороннего кода создает потенциальные риски безопасности, а также потому, что брандмауэры и другие инструменты управления политиками в некоторых случаях могут сделать выполнение стороннего кода невозможным.

Введение в REST API

Тело

Тело ответа REST API — это данные, которые клиент запросил у REST API. Тело не является обязательным, однако, если данные предоставляются в теле, тип данных указывается в заголовке с помощью ключа Content-Type. Если запрос REST API оказался неудачным, тело ответа может содержать дополнительную информацию о проблеме или действие, которое необходимо предпринять для успешного выполнения запроса.

Пагинация ответов

Некоторые API, например, API поиска, могут нуждаться в отправке огромного количества данных в ответе. Чтобы уменьшить использование пропускной способности сети, такие API разбивают данные ответа на страницы.

Пагинация ответа позволяет разбить данные на фрагменты. Большинство API, в которых реализована постраничная разбивка, позволяют запрашивающей стороне указать, сколько элементов он хочет получить в ответе. Поскольку существует несколько фрагментов, API также должен позволить запрашивающему указать, какой фрагмент ему нужен. Для API не существует стандартного способа реализации пагинации, но большинство реализаций используют параметр запроса, чтобы указать, какую страницу возвращать в ответе.

Сжатые данные ответа

Когда серверу необходимо отправить очень большой объем данных, которые не могут быть постранично отображены, сжатые данные являются еще одним способом снижения пропускной способности.

Сжатие данных может быть запрошено клиентом через сам запрос API. Чтобы запросить сжатие данных, запрос должен добавить поле Accept-Encoding в заголовок запроса. Допустимыми значениями являются:

- gzip;
- compress;
- deflate;
- br;
- identity;
- *;

Если сервер не может обеспечить ни один из запрошенных типов сжатия, он отправит ответ с кодом состояния `406 -- Not acceptable`.

Если сервер выполнит сжатие, он отправит ответ со сжатыми данными и добавит поле Content-Encoding в заголовок ответа. Значение Content-Encoding — это тип сжатия, который был использован, что позволяет клиенту распаковать данные соответствующим образом.

Использование диаграмм последовательностей с REST API

Диаграммы последовательности используются для объяснения последовательности обменов или событий. Они представляют собой сценарий упорядоченного набора событий. Их также называют диаграммами событий. Хотя один запрос REST API может служить для получения информации или инициирования изменений в системе, чаще всего взаимодействие с конкретной службой REST API представляет собой последовательность запросов. По этой причине диаграммы последовательности часто используются для объяснения запросов/ответов REST API и асинхронной активности.

Формализованные диаграммы последовательности тесно связаны со стандартизированной системой моделирования, известной как унифицированный язык моделирования (UML), и считаются ее подмножеством. UML включает стандартизированные подходы к другим аспектам статических ресурсов и процессов, в том числе стандартизированные способы диаграммы и объяснения пользовательских интерфейсов, определений классов и объектов, а также поведения взаимодействия. Диаграммы последовательностей являются одним из способов диаграммы поведения взаимодействия.

В стандартной диаграмме последовательности, такой как приведенная ниже, ось Y — это упорядоченное и немасштабированное время, с нулевым временем (`t=0`) в верхней части и временем, увеличивающимся по направлению к нижней части. Если стрелка или обмен находится ниже по диаграмме, то он происходит после тех, которые находятся выше.

Ось X состоит из линий жизни, представленных вертикальными линиями с заголовками, и обменов или сообщений, представленных горизонтальными стрелками. Линия жизни — это любой элемент, который может взаимодействовать, получая или генерируя сообщение. По традиции, внешние инициаторы последовательности, такие как пользователь, клиент или веб-браузер, располагаются в левой части диаграммы. Такие элементы, как файловые системы, базы данных, постоянные хранилища и так далее, располагаются в правой части. Промежуточные службы, такие как веб-сервер или конечные точки API, располагаются в центре.

Одним из полезных аспектов диаграмм последовательности является то, что пользователи могут сосредоточиться на взаимодействии только между двумя жизненно важными элементами. Например, в то время как остальная часть диаграммы может помочь установить контекст, пользователи REST API могут сосредоточиться на взаимодействии между клиентом и внешним

интерфейсом, или, как показано ниже, **обработчиком Host HTTP/S Service API endpoint**.

Приведенная ниже последовательность API (Рисунок 67) представляет собой три упрощенных сценария.

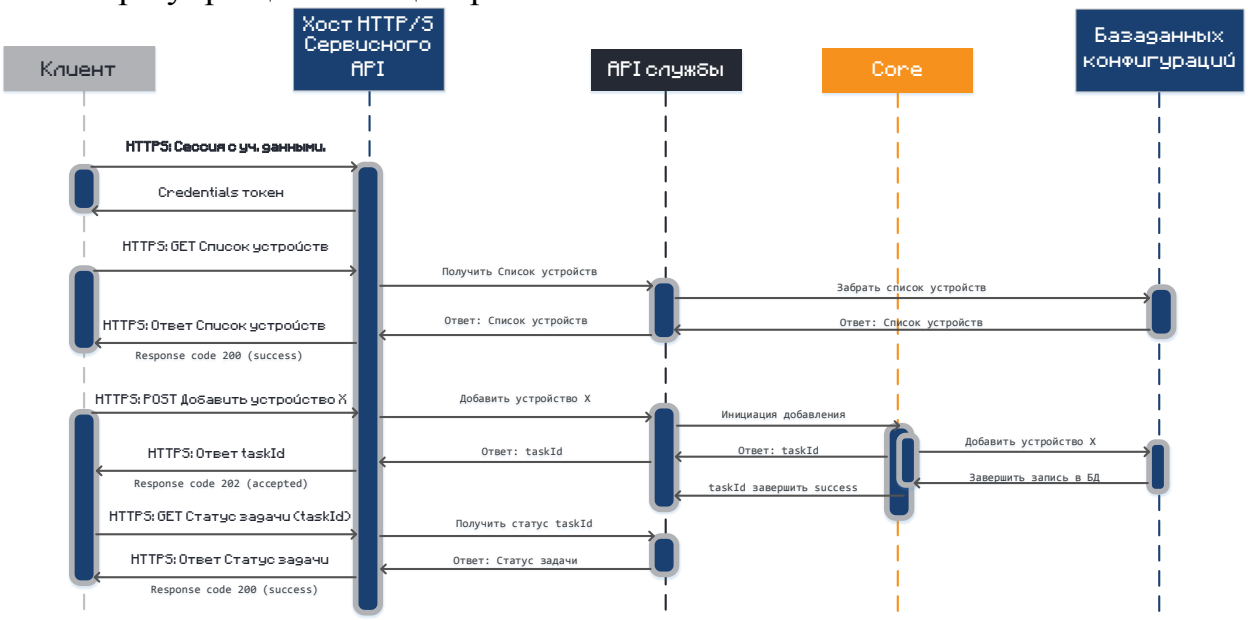


Рисунок 67 — Диаграмма последовательности запросов/ответов API

Клиент слева может быть Python приложением, тестовой средой POSTMAN или любым другим генератором запросов API. Клиент отправляет запрос в формате HTTPS на внешний компонент сервера. Этот компонент обычно передает запрос соответствующему обработчику API, который называется API Services. Справа от API Services находится элемент с простым названием Core, который является общим для основной логики обработки или платформы.

API Services может обрабатывать некоторые запросы напрямую или интерпретировать и затем пересылать часть или весь запрос на Core. Самый правый столбец, показанный здесь как База конфигураций, может быть любым хранилищем или даже процессом обмена сообщениями или регистрации для связи с другой системой.

В этом примере показаны три отдельные последовательности: создание сеанса, получение списка устройств и создание устройства.

Создание сессии: Начальный запрос обозначен как HTTPS: Сессия с уч. данными (credentials). Здесь логика создания HTTPS-сессии находится во внешнем обработчике конечной точки Host HTTP/S Service API.

Получение списка устройств: Второй запрос от клиента — это запрос списка устройств от платформы. В этой последовательности запрос передается в модуль API Services, который содержит логику для прямого запроса к базе данных конфигурации, получения списка устройств и возврата списка обработчику конечной точки. Обработчик затем упаковывает содержимое в ответ HTTPS и возвращает его клиенту вместе с кодом состояния HTTP, указывающим на успех.

Эти первые две последовательности демонстрируют синхронный обмен, в котором за запросом следует ответ, и задача полностью завершается с успехом или неудачей.

Создание устройства: Третья последовательность начинается с POST-запроса на создание устройства. В этом случае запрос переходит в API Services, которые затем пересылают запрос в логику Core, которая начинает работать над запросом. API-сервисы ждут только сообщения от ядра, подтверждающего, что процесс создания устройства начался. Ядро предоставляет хэндл (TaskId), который идентифицирует работу и позволяет проследить, была ли работа завершена. Значение TaskId передается в ответе обратно клиенту. HTTP-ответ сообщает клиенту только хэндл (TaskId) и то, что запрос был инициирован, с кодом ответа 202 (Accepted). Это означает, что запрос был принят и что работа продолжается.

Логика Core продолжает выполняться. Она обновляет базу данных конфигурации, а затем информирует службы API о завершении работы. Позднее клиент может захотеть подтвердить выполнение задачи. Клиент делает это с помощью запроса статуса задачи. Службы API могут ответить информацией о статусе завершения, а также о статусе успеха или неудачи для этой задачи. Поскольку запрашиваемая работа не была завершена до получения ответа от клиента, это взаимодействие считается асинхронным.

Аутентификация в REST API

Аутентификация REST API

Merriam-Webster определяет аутентификацию как *"...действие, процесс или метод подтверждения реальности, истинности или подлинности чего-либо (например, личности)"*. В целях безопасности большинство REST API требуют аутентификации, чтобы посторонние пользователи не могли создавать, обновлять или удалять информацию неправильно или злонамеренно, а также получать доступ к информации, которая не должна быть общедоступной. Без аутентификации REST API позволяет любому получить доступ к функциям и системным сервисам, которые были открыты через этот интерфейс. Требование аутентификации — это та же концепция, что и требование ввода имени пользователя/пароля для доступа к административной странице приложения.

Некоторые API не требуют аутентификации. Обычно они доступны только для чтения и не содержат критически важной или конфиденциальной информации.

Аутентификация vs Авторизация

При работе с REST API важно понимать разницу между аутентификацией и авторизацией, поскольку эти два термина часто используются как взаимозаменяемые или неправильно. Знание разницы поможет вам устранить любые проблемы, связанные с безопасностью запроса REST API.

Аутентификация

Аутентификация — это акт проверки личности пользователя. Пользователь доказывает, что он тот, за кого себя выдает. *Например, когда вы едете в аэропорт, вы должны предъявить удостоверение личности (рисунок 68), выданное правительством, или использовать биометрические данные, чтобы доказать, что вы тот человек, за которого себя выдаете.*

Рисунок 68 — Аутентификация подтверждает личность пользователя

Авторизация

Авторизация — это доказательство пользователем того, что у него есть разрешения на выполнение запрашиваемого действия на данном ресурсе. *Например, когда вы идете на концерт или кино, вам достаточно предъявить билет (Рисунок 69), чтобы доказать, что вам разрешен вход. Вам не обязательно подтверждать свою личность.*

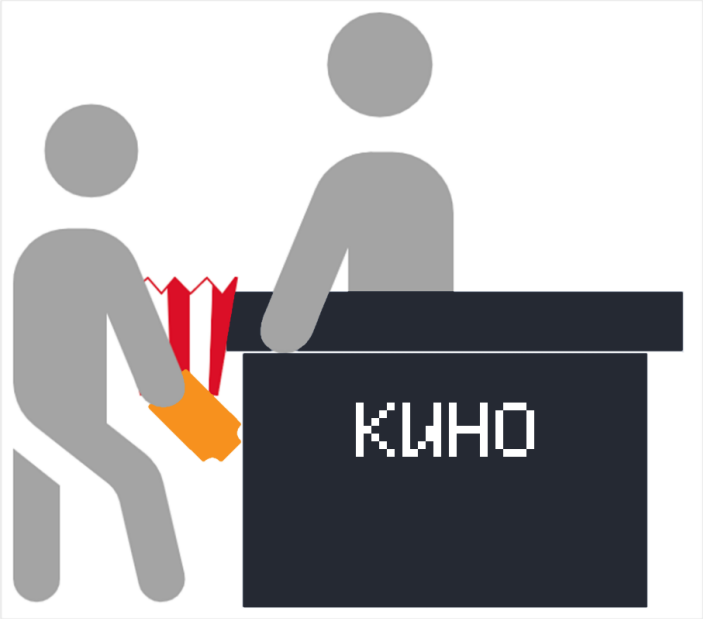


Рисунок 69 — Авторизация определяет доступ пользователя

Механизмы аутентификации

К общим типам механизмов аутентификации относятся Базовая, Bearer и API Ключ.

Базовая аутентификация

Базовая аутентификация, также известная как Basic Auth, использует стандартную схему аутентификации Basic HTTP. Basic Auth передает учетные данные в виде пар имя пользователя/пароль, разделенных двоеточием (:) и закодированных с помощью Base64.

В запросе REST API информация о Basic Auth будет представлена в заголовке:

```
Authorization: Basic <username>:<password>
```

Basic Auth — это простейший механизм аутентификации. Он крайне небезопасен, если только не используется в паре с запросами, использующими HTTPS, а не HTTP. Хотя учетные данные кодируются, они не шифруются. Расшифровать учетные данные и получить пару имя пользователя/пароль очень просто.

Bearer аутентификация

Bearer аутентификаци, также известная как аутентификация по токену, использует стандартную схему аутентификации Bearer HTTP. Она является более безопасной, чем базовая аутентификация, и обычно используется с OAuth (рассмотрим позже) и Single Sign-On (SSO). Аутентификация на предъявителя использует маркер предъявителя, который представляет собой строку, сгенерированную сервером аутентификации, таким как Identity Service (IdS).

В запросе REST API информация об аутентификации носителя будет предоставлена в заголовке:

```
Authorization: Bearer <bearer token>
```

Как и базовая аутентификация, аутентификация по токену должна использоваться с HTTPS.

API Ключ

API Ключ, также называемый API-токеном, — это уникальная буквенно-цифровая строка, генерируемая сервером и присваиваемая пользователю. Чтобы получить уникальный ключ API, пользователь обычно входит на портал, используя свои учетные данные. Этот ключ обычно присваивается один раз и не регенерируется. Все запросы REST API для этого пользователя должны содержать назначенный ключ API в качестве формы аутентификации. Как и в случае с другими типами аутентификации, API-ключи безопасны только при использовании HTTPS.

API-ключи предназначены для аутентификации, но часто используются не по назначению в качестве механизма авторизации.

API-ключи бывают двух типов: публичные и приватные.

Публичный ключ API может быть предоставлен в общее пользование и дает пользователю доступ к подмножеству данных и API. Не передавайте закрытый ключ, поскольку он аналогичен вашему имени пользователя и паролю. Большинство ключей API не имеют срока действия, поэтому, если ключ не может быть отозван или регенерирован, в случае его распространения или компрометации, любой человек с таким ключом может получить неограниченный доступ к системе, как и вы.

В запросе REST API ключ может быть предоставлен несколькими различными способами:

- Строка запроса: Рекомендуется только для открытых ключей API
- Заголовок: Используется ключ авторизации или пользовательский ключ `Authorization: <API Key` или `Authorization: APIkey <API Key>` или `APIkey: <API Key>`
- Данные тела: Использует уникальный ключ в качестве идентификатора `Content-Type: application/json`
- Cookie: Использует уникальный ключ в качестве идентификатора `Cookie: API_KEY=<API Key>`

Механизмы авторизации

Open Authorization, также известная как OAuth, объединяет аутентификацию и авторизацию. OAuth был разработан как решение для небезопасных механизмов аутентификации. Обладая повышенной безопасностью по сравнению с другими вариантами, она обычно является рекомендуемой формой аутентификации/авторизации для REST API.

Существует две версии OAuth, называемые просто OAuth 1.0 и OAuth 2.0. Большинство современных REST API реализуют OAuth 2.0. Обратите внимание, что OAuth 2.0 не имеет обратной совместимости.

Согласно определению, данному в документе OAuth 2.0 Authorization Framework, "OAuth 2.0 Authorization Framework позволяет стороннему приложению получить ограниченный доступ к HTTP-сервису либо от имени владельца ресурса, организуя взаимодействие по утверждению между владельцем ресурса и HTTP-сервисом, либо позволяя стороннему приложению получить доступ от своего имени".

По сути, OAuth 2.0 позволяет предварительно зарегистрированным приложениям получать разрешение на выполнение запросов REST API от имени пользователя без необходимости сообщать свои учетные данные самому приложению. OAuth позволяет пользователю предоставлять учетные данные непосредственно серверу авторизации, обычно поставщику идентификационных данных (IdP) или службе идентификации (IdS), для получения маркера доступа, который может быть передан приложению.

Этот процесс получения маркера называется потоком. Затем приложение использует этот маркер в REST API в качестве аутентификации носителя. Затем веб-служба для REST API проверяет сервер авторизации,

чтобы убедиться, что маркер действителен и что запрашивающий имеет право выполнять запрос.

Ограничение скорости API

Стандартного способа реализации ограничения скорости не существует, но распространенные алгоритмы включают:

- Leaky bucket;
- Token bucket;
- Fixed window counter;
- Sliding window counter.

Leaky bucket

Алгоритм leaky bucket помещает все входящие запросы в очередь запросов в том порядке, в котором они были получены (Рисунок 70). Входящие запросы могут поступать с любой скоростью, но сервер будет обрабатывать запросы из очереди с фиксированной скоростью. Если очередь запросов переполнена, запрос отклоняется.

Рисунок 70 — Leaky bucket

При использовании этого алгоритма клиент должен быть готов к задержке ответа или отклонению запроса. Визуальное представление алгоритма Leaky bucket представлено на рисунке 71

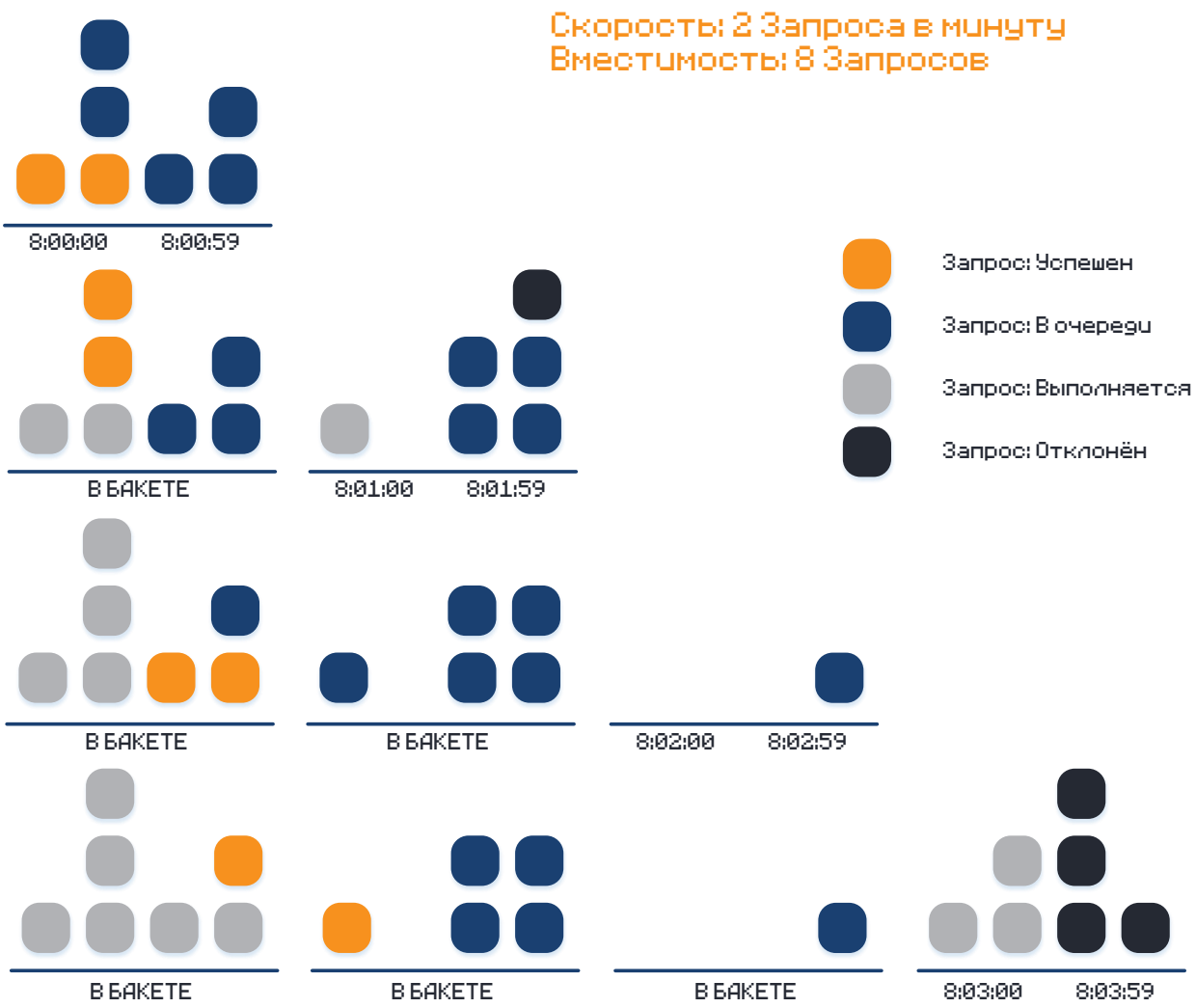


Рисунок 71 — Модель алгоритма Leaky bucket

Token bucket

Алгоритм Token bucket (Рисунок 72) дает каждому пользователю определенное количество токенов, монеток на иллюстрации, которые он может использовать в течение определенного промежутка времени, и эти жетоны накапливаются до тех пор, пока не будут использованы. Когда клиент делает запрос, сервер проверяет бакет, чтобы убедиться, что в нем есть хотя бы один токен. Если это так, он извлекает его и обрабатывает запрос. Если отсутствует, запрос отклоняется.

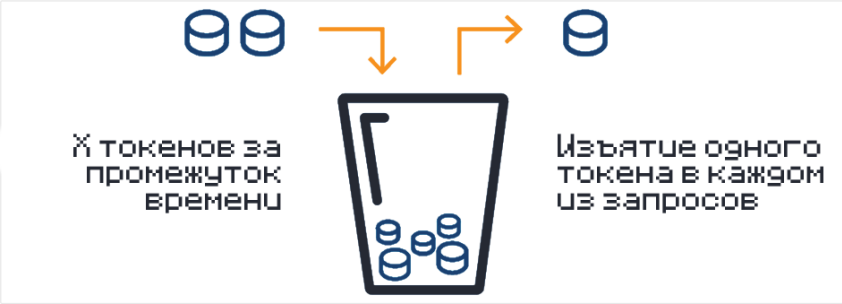


Рисунок 72 — Принцип работы алгоритма Token bucket

Все запросы, сделанные до пополнения запасов токенов, будут отклонены. После пополнения токенов пользователь может снова выполнять запросы.

Например, API, использующий алгоритм Token Bucket, устанавливает ограничение по скорости в 10 запросов на клиента в час. Если клиент сделает 11 запросов в течение часа, 11-й запрос будет отклонен, потому что монеток уже не осталось. С другой стороны, если клиент не делает никаких запросов в течение 6 часов, он может сделать сразу 60 запросов, потому что жетоны уже накоплены.

При таком алгоритме клиент должен подсчитать, сколько токенов у него есть на данный момент, чтобы избежать отклоненных запросов. Он также должен обработать потенциально отклоненные запросы, встроив механизм повторных попыток, когда жетоны будут пополнены (Рисунок 43).

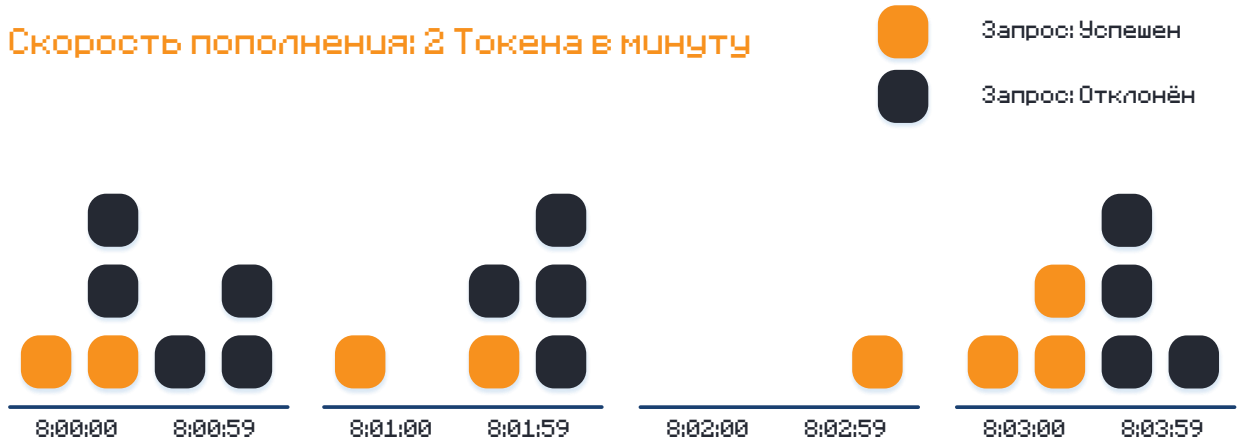


Рисунок 73 — Модель алгоритма Token bucket

Fixed window counter

Алгоритм Fixed window counter похож на алгоритм Token bucket, за исключением двух основных отличий:

- В нем используется счетчик, а не набор токенов.
- Счётчик не может накапливаться.

В этом алгоритме фиксированному окну времени присваивается счетчик, показывающий, сколько запросов может быть обработано за этот период. Когда сервер получает запрос, счетчик для текущего окна времени проверяется, чтобы убедиться, что он не равен нулю. Когда запрос обрабатывается, счетчик уменьшается. Если лимит для данного окна времени достигнут, все последующие запросы в этом окне времени будут отклонены. Когда начнется следующее окно времени, счетчик будет установлен на заранее определенное значение, и запросы снова можно будет обрабатывать.

Если вернуться к предыдущему примеру с 10 запросами в час, то 11-й запрос в течение часа будет отклонен, но после 6 часов без запросов клиент сможет сделать только 10 запросов за один час, потому что эти "неиспользованные" запросы не были накоплены.

При таком алгоритме клиент должен знать, когда начинается и заканчивается временное окно, чтобы знать, сколько запросов можно сделать за этот промежуток времени. Так же, как и в алгоритме "Token bucket", клиент должен создать механизм повторных запросов, чтобы он мог повторить запросы, когда начнется следующее окно времени.

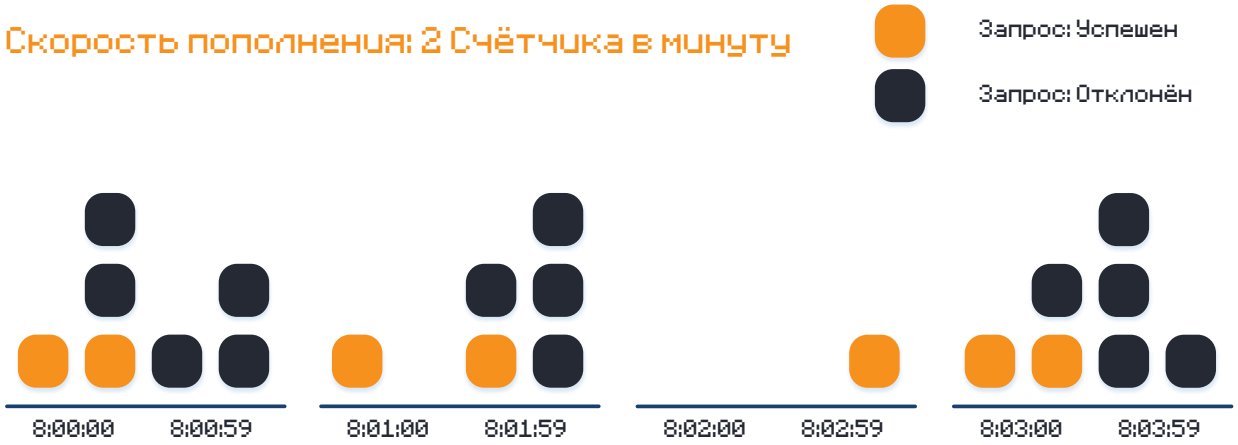


Рисунок 74 — Модель алгоритма Fixed window counter

Sliding window counter

Алгоритм Sliding window counter позволяет выполнить фиксированное количество запросов за определенный промежуток времени. Этот промежуток времени не является фиксированным окном, и счетчик не пополняется, когда окно начинается снова. В этом алгоритме сервер хранит метку времени, когда был сделан запрос. Когда поступает новый запрос, сервер подсчитывает, сколько запросов уже было сделано с начала окна до текущего времени, чтобы определить, следует ли обработать запрос или отклонить его (Рисунок 45). *Например, если скорость составляет пять запросов в минуту, при получении нового запроса сервер проверяет, сколько запросов было сделано за последние 60 секунд. Если пять запросов уже было сделано, то новый запрос будет отклонен.*

При таком алгоритме клиенту не нужно знать, когда начинается и заканчивается временное окно. Он просто должен убедиться, что на момент запроса лимит скорости не был превышен. Клиенту также необходимо разработать способ задержки запросов, если это необходимо, чтобы не превысить допустимую скорость, и, конечно, учесть отклоненные запросы.

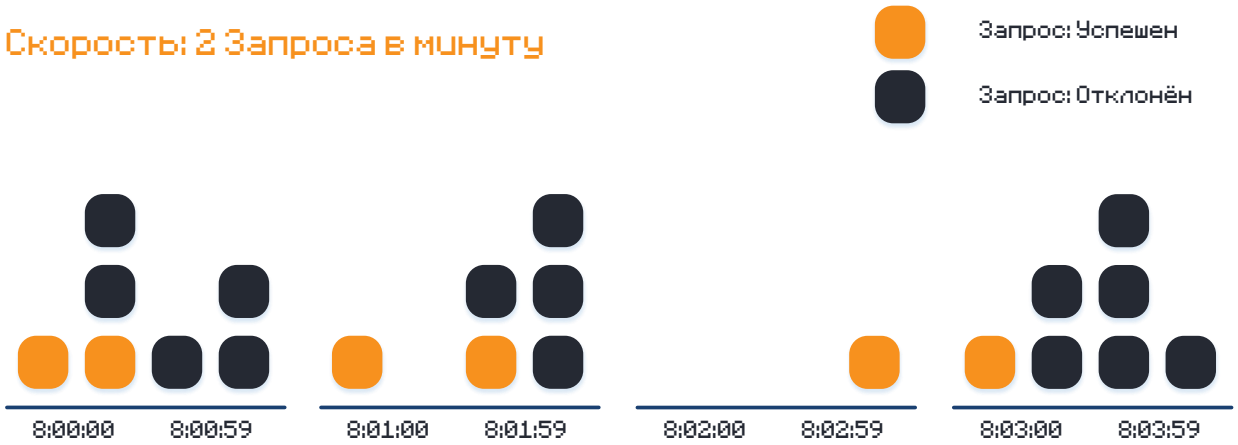


Рисунок 75 — Модель алгоритма Sliding window counter

Знание предельной скорости

В документации API обычно содержится подробная информация об ограничении скорости и единицах времени. Кроме того, многие API с ограничением скорости добавляют информацию об ограничении скорости в заголовок ответа. Поскольку стандарта не существует, пара ключ-значение, используемая в заголовке, может отличаться в разных API. Некоторые часто используемые ключи включают:

- `X-RateLimit-Limit`: максимальное количество запросов, которое может быть сделано за определенную единицу времени;
- `X-RateLimit-Remaining`: Количество оставшихся запросов, которые запросчик может сделать в текущем окне ограничения скорости;
- `X-RateLimit-Reset`: Время, когда окно ограничения скорости будет сброшено.

Клиент может использовать эту информацию для отслеживания того, сколько еще запросов API он может сделать в текущем окне, что поможет ему избежать превышения лимита скорости.

Увеличение предельной скорости

Если лимит скорости превышен, сервер автоматически отклоняет запрос и отправляет ответ HTTP, информирующий пользователя. Кроме того, обычно ответ, содержащий ошибку "превышен лимит скорости", также включает в себя значимый код состояния HTTP. К сожалению, поскольку стандарта для такого взаимодействия не существует, сервер может выбирать, какой код состояния отправить. Наиболее часто используемыми кодами состояния HTTP являются `429: Too Many Requests` или `403: Forbidden`; убедитесь, что ваш клиент настроен для конкретного API, который он использует.

Работа с Webhooks

Что такое Webhook?

Вебхук — это обратный вызов HTTP или HTTP POST на указанный URL, который уведомляет ваше приложение, когда определенная активность или "событие" произошло в одном из ваших ресурсов на платформе. Концепция проста. Подумайте о том, чтобы попросить кого-то: "Скажите мне немедленно, если произойдет X". Этот "кто-то" — поставщик Вебхуков, а вы — приложение.

Вебхуки позволяют приложениям получать данные в режиме реального времени, поскольку они запускаются определенными действиями или событиями. Благодаря вебхукам приложения становятся более эффективными, поскольку им больше не нужен механизм опроса. Механизм опроса — это способ многократного запроса информации от службы до тех пор, пока не будет выполнено определенное условие. Представьте, что вам нужно спрашивать кого-то снова и снова: "X уже произошел? X уже произошел?". Раздражает, правда? Это и есть опрос. Опрос снижает производительность как клиента, так и сервера из-за многократной обработки запросов и ответов. Кроме того, опрос не работает в режиме реального времени, поскольку опросы происходят через фиксированные интервалы времени. Если событие произошло сразу после последнего опроса, ваше приложение не узнает об изменении статуса до истечения интервала опроса и следующего опроса.

Webhooks также известны как reverse API, поскольку приложения подписываются на сервер webhook, регистрируясь у провайдера webhook. В процессе регистрации приложение предоставляет URI, который будет вызываться сервером при возникновении целевого действия или события. Этот URI обычно представляет собой API на стороне приложения, который сервер вызывает при срабатывании веб-хука.

Когда срабатывает webhook, сервер отправляет уведомление, становясь вызывающей стороной и делая запрос к предоставленному URI. Этот URI представляет собой API для приложения, а приложение становится вызывающей стороной и выполняет запрос. В результате для webhook’ов в роли меняются местами: сервер становится клиентом, а клиент — сервером. Несколько приложений могут подписаться на один сервер веб-хуков.

Получение Вебхуков

Чтобы получить уведомление от провайдера вебхуков, приложение должно соответствовать определенным требованиям:

- Приложение должно быть постоянно запущено, чтобы получать HTTP POST-запросы.
- Приложение должно зарегистрировать URI у провайдера вебхуков, чтобы провайдер знал, куда отправлять уведомления при наступлении целевых событий.

В дополнение к этим двум требованиям приложение должно обрабатывать входящие уведомления от сервера webhook.

Поскольку в работе с вебхуками участвуют третьи стороны, иногда бывает сложно убедиться, что все работает правильно. Существует множество бесплатных онлайн-инструментов, которые гарантируют, что ваше приложение может получать уведомления от веб-хуков. Многие из них также могут предоставить вам предварительный просмотр содержимого уведомления вебхука. Эти инструменты можно найти по запросу "`webhook tester`", и они могут быть полезны при разработке и внедрении вашего приложения.

Устранение неполадок с вызовами API

Устранение неполадок при выполнении запросов REST API

Вы узнали о REST API и изучили их в лабораторных условиях. В представленных сценариях все работало правильно и всегда было успешным. Конечно, в реальной жизни все не всегда работает именно так. В какой-то момент вы обнаружите, что делаете запросы к REST API и не получаете ожидаемого ответа. В этой части вы узнаете, как устранить наиболее распространенные проблемы с REST API.

Обратите внимание, когда вы будете применять эти концепции и советы по устранению неполадок к конкретному REST API, убедитесь, что у вас под рукой есть справочное руководство по API и информация об аутентификации API; это значительно облегчит вашу работу.

Отсутствие ответа и HTTP статус кода от сервера API

Давайте начнём с простого вопроса: Правда или неправда: каждый запрос, отправленный на сервер REST API, возвращает ответ с кодом состояния. Правильный ответ: враньё. Хотя можно ожидать, что вы всегда будете получать следующие статус коды, `2xx`, `4xx` и `5xx`, существует множество случаев, когда сервер API не доступен или просто не отвечает. В таких случаях вы не получите заветный статус код, потому что сервер не может отправить ответ.

Хотя не отвечающий сервер, может оказаться большой проблемой, основную причину отсутствия ответа с легкостью можно определить, по сообщениям об ошибках, полученным в результате запроса.

Давайте рассмотрим некоторые советы по устранению неполадок, которые помогут вам понять, почему сервер API не отправил ответ и статус код, а также некоторые возможные способы устранения проблемы.

Ошибка на стороне клиента

В первую очередь необходимо проверить, нет ли у вас ошибки на стороне клиента; именно здесь у вас больше контроля в плане устранения проблемы.

Есть ли ошибка пользователя?

Чтобы изолировать ошибку от пользователя, можно задать следующие вопросы:

- Правильный ли URI?
- Работал ли этот запрос раньше?

При первом использовании REST API часто случается ошибка при вводе URI. Проверьте правильность URI запроса в справочном руководстве по данному API.

Пример неправильного URI

Чтобы проверить условие недопустимого URI, запустите похожий сценарий, подобный приведённому ниже, который просто делает запрос к URI.

```
import requests
uri = "sandboxdnac.zss.com/dna/intent/api/v1/network-device"
resp = requests.get(uri, verify = False)
```

Трассировка будет выглядеть следующим образом:

```
....requests.exceptions.MissingSchema: Invalid URL
'sandboxdnac.zss.com/dna/intent/api/v1/network-device': No schema
supplied. Perhaps you meant
http://sandboxdnac.cisco.com/dna/intent/api/v1/network-device?....
```

Пример неправильного доменного имени

Чтобы проверить условие неправильного доменного имени, запустите сценарий, подобный этому, который просто делает запрос на URI, имеющий неправильное доменное имя.

```
import requests
url = "https://sandboxdnac123.zss.com/dna/intent/api/v1/network-
device"
resp = requests.get(url, verify = False)
```

Трассировка в этом случае будет выглядеть следующим образом:

```
....requests.exceptions.ConnectionError:
HTTPSConnectionPool(host='sandboxdnac123.zss.com', port=443): Max
retries exceeded with url: /dna/intent/api/v1/network-device
(Caused by
NewConnectionError('<urllib3.connection.VerifiedHTTPSConnection
object at 0x109541080>: Failed to establish a new connection:
[Errno 8] nodename nor servname provided, or not known')).....
```

Есть ли проблема с подключением?

Если URI правильный, он все равно может быть недоступен. Задайте себе эти вопросы:

– Нет ли проблем с прокси-сервером, брандмауэром или VPN?
– Нет ли ошибки SSL?

Пример недействительного сертификата

Эта проблема может возникнуть только в том случае, если URI REST API использует защищенное соединение (HTTPS).
Когда схема URI это HTTPS, соединение будет выполнять SSL квитирование между клиентом и сервером для аутентификации друг друга. Это квитирование должно быть успешным до того, как запрос REST API будет отправлен на сервер API.

При использовании библиотеки `requests` в Python для выполнения запроса REST API, если квитирование SSL завершилось неудачно, в трассировке будет содержаться `requests.exceptions.SSLError`.

Например, если квитирование SSL завершилось неудачно из-за того, что проверка сертификата не прошла, то обратная трассировка будет выглядеть следующим образом:

```
requests.exceptions.SSLError: HTTPSConnectionPool(host='xxxx',
port=443): Max retries exceeded with url:
/dna/intent/api/v1/network-device (Caused by
SSLError(SSLCertVerificationError(1, '[SSL:
CERTIFICATE_VERIFY_FAILED] certificate verify failed: self signed
certificate (_ssl.c:1108)')))
```

В этой ситуации вы должны исправить недействительный сертификат.

Чтобы исправить этот фэйл в библиотеки запросов в Python, добавьте в запрос параметр verify.

```
resp = requests.get(url, verify = False)
```

Решение: Ошибки клиента обычно просто устранить, особенно если сообщения об ошибках в обратном пути указывают на возможную причину и возможное решение. Внимательно проанализируйте логи в поисках первопричины.

Ошибка на стороне сервера

После того как вы убедились в отсутствии ошибок на стороне клиента, следующее, что нужно проверить, — это ошибки на стороне сервера.

Работает ли сервер API?

Самое очевидное, с чего следует начать, — убедиться, что сам сервер API функционирует должным образом. Есть несколько вещей, которые вы обязаны спросить у себя:

1. Выключено ли питание?
2. Есть ли проблемы с кабелями?
3. Изменилось ли доменное имя?
4. Не отключена ли сеть?

Чтобы проверить, доступен ли IP-адрес, запустите сценарий, подобный этому, который просто делает запрос к URL-адресу и ждет ответа.

Предупреждение: При выполнении сценария, ниже возможна длительная задержка.

```
import requests
url = "https://209.165.209.225/dna/intent/api/v1/network-device"
resp = requests.get(url, verify = False)
```

***Примечание**: IP-адрес в этом сценарии является фиктивным, но сценарий дает тот же результат, что и недоступный сервер.*

Если сервер API не работает, вы получите длинное молчание, за которым последует трассировка, выглядящая следующим образом:

```
....requests.exceptions.ConnectionError:
HTTPSConnectionPool(host='209.165.209.225', port=443): Max retries
exceeded with url: /dna/intent/api/v1/network-device (Caused by
NewConnectionError('<urllib3.connection.VerifiedHTTPSConnection
object at 0x10502fe20>: Failed to establish a new connection:
[Errno 60] Operation timed out'))....
```

Если сервер работает нормально, вам нужно определить, есть ли причина, по которой клиент не получает ответа. Спросите себя:

- Доступны ли IP-адрес и доменное имя с места нахождения клиента в сети?
- Отправляет ли сервер API ответ, но клиент его не получает?

Чтобы проверить это условие, используйте инструмент захвата сети, чтобы увидеть, теряется ли ответ от сервера API в процессе взаимодействия между сервером API и клиентом. Если у вас есть доступ, просмотрите журналы сервера API, чтобы определить, был ли получен запрос, был ли он обработан и был ли отправлен ответ.

Решение: Проблемы на стороне сервера не могут быть решены со стороны клиента API. Для решения этой проблемы обратитесь к администратору сервера API.

Интерпретация кодов состояния

Поскольку REST API основаны на http, они также используют коды состояния http для уведомления клиентов API о результатах запроса.

Если не указано иное, код состояния является частью стандарта HTTP/1.1 (RFC 7231), что означает, что первая цифра кода состояния определяет класс ответа. (Последние две цифры не играют никакой роли в определении класса или категории, но помогают различать результаты). Существует 5 таких категорий:

- `1xx: Informational`: Запрос получен, продолжается обработка.
- `2xx: Success`: Действие было успешно получено, понято и принято.
- `3xx: Redirection`: Для завершения запроса необходимо выполнить дальнейшие действия.
- `4xx: Client Error`: Запрос содержит неправильный синтаксис или не может быть выполнен.
- `5xx: Server Error`: Серверу не удалось выполнить явно правильный запрос.

Обычно мы видим `2xx`, `4xx` и `5xx` от сервера REST API. Обычно можно найти основную причину ошибки, если понять значение кодов ответа. Иногда сервер API также предоставляет дополнительную информацию в теле ответа.

Для всех запросов, имеющих код состояния возврата, выполните следующие шаги для устранения ошибок:

- Шаг 1: Проверьте код возврата. Это может помочь вывести код возврата в вашем сценарии на этапе разработки.
- Шаг 2: Проверьте тело ответа. Выведите тело ответа во время разработки; чаще всего вы можете найти, что пошло не так, в ответном сообщении, отправленном вместе с кодом состояния.
- Шаг 3: Если вы не можете решить проблему с помощью двух вышеуказанных шагов, воспользуйтесь справочником кода состояния, чтобы понять определение кода состояния.

Статусы 2xx и 4xx

Давайте рассмотрим эти коды более подробно:

`2xx – Success`

Когда клиент получает код ответа 2xx, это означает, что запрос клиента был успешно получен, понят и принят. Однако вы всегда должны проверять, что ответ указывает на успех правильного действия и что скрипт делает то, что, по вашему мнению, он должен делать.

`4xx – Client side error`

Ответ 4xx означает, что ошибка произошла на стороне клиента. Некоторые серверы могут включать сущность, содержащую объяснение ошибки. Если это не так, вот несколько общих рекомендаций по устранению распространенных ошибок 4xx:

`400 – Bad Request`

Запрос не может быть понят сервером из-за неправильного синтаксиса. Проверьте синтаксис вашего API.

Одной из причин ошибки 400 Bad Request является сам ресурс. Дважды проверьте конечную точку и ресурс, к которому вы обращаетесь, не написали ли вы неправильно один из ресурсов или не забыли "s", чтобы сделать его

множественным, например, `/device` против `/devices` или `/interface` против `/interfaces`? Является ли URI правильно сформированным и полным?

Другой причиной ошибки 400 Bad Request может быть синтаксическая ошибка в объекте JSON, который представляет ваш POST-запрос.

Давайте рассмотрим пример воображаемого сервиса:

```
import requests
url = "http://myservice/api/v1/resources/house/rooms/id/"
resp = requests.get(url,auth=("person2","super"),verify = False)
print(resp.status_code)
print(resp.text)
```

Можете ли вы догадаться, в чем дело, просто прочитав код?
Этот пример возвращает код состояния 400. Серверная сторона также сообщает вам: "Поле id не предоставлено", поскольку id является обязательным для этого API-запроса. Чтобы убедиться в этом, нужно посмотреть документацию, лол.

```
import requests
url =
"http://myservice/api/v1/resources/house/rooms/id/1001001027331"
resp = requests.get(url,auth=("person2","super"),verify = False)
print(resp.status_code)
print(resp.text)
```

401 – Unauthorized

Это сообщение об ошибке означает, что сервер не смог аутентифицировать запрос.

Проверьте свои учетные данные, включая имя пользователя, пароль, ключ API, токен и так далее. Если с этими элементами проблем нет, возможно, вам стоит еще раз проверить URI запроса, поскольку сервер может отклонить доступ в случае неправильного URI запроса.

```
import requests
url = "http://myservice/api/v1/resources/house/rooms/all"
resp = requests.get(url,verify = False)
print(resp.status_code)
print(resp.text)
```

Запрос не предоставляет учетные данные. В коде необходимо добавить аутентификацию `auth=("person1", "great")`.

403 - Forbidden

В этом случае сервер распознает учетные данные аутентификации, но клиент не уполномочен выполнять запрос. Некоторые API имеют контроль доступа на основе ролей, и для выполнения определенных API требуется роль

супер-админа. Дополнительную информацию можно найти в справочном руководстве по API.

```
import requests
url = "http://myservice/api/v1/resources/house/rooms/id/1"
resp = requests.get(url,auth=("person1","great"),verify = False)
print(resp.status_code)
print(resp.text)
```

Почему этот воображаемый фрагмент кода не работает? Мы просто использовали имя пользователя person1 и пароль great, и это сработало в прошлом примере.

Код 403 не является проблемой аутентификации; сервер уверен в личности пользователя, просто у пользователя недостаточно привилегий для использования данного API. Если мы используем person2/super в качестве имени пользователя/пароля для аутентификации, это будет работать, потому что person2 имеет привилегии для выполнения этого API.

Аутентификация должна быть изменена, чтобы использовать person2/super вместо person1/great.

```
import requests
url = "http://myservice/api/v1/resources/house/rooms/id/1"
resp = requests.get(url,auth=("person2","super"),verify = False)
print (resp.status_code)
print (resp.text)
```

404 - Not Found

Сервер не нашел ничего, соответствующего URI запроса; проверьте URI запроса, чтобы убедиться в его правильности. Если код раньше работал, возможно, вам стоит проверить последнее справочное руководство по API, так как синтаксис API может меняться с течением времени.

Рассмотрим "get all interfaces". В названии сказано "all interfaces", поэтому вы пытаетесь использовать api/v1/interfaces, но получаете ошибку 404, поскольку запрос API на самом деле api/v1/interface.

```
import requests
url = "http://myservice/api/v1/resources/house/room/all"
resp = requests.get(url,auth=("person1","great"),verify = False)
print (resp.status_code)
print (resp.text)
```

Почему этот сценарий возвращает статус 404, и можете ли вы предположить, как это исправить?

Фактический URI — /api/v1/resources/house/rooms/all. Ресурс комнаты отсутствует, поэтому сервер не смог найти ресурс, соответствующий URI запроса.

```
import requests
url = "http://myservice/api/v1/resources/house/rooms/all"
resp = requests.get(url,auth=("person1","great"),verify = False)
print (resp.status_code)
print (resp.text)
```

405 - Method Not Allowed

В этом случае запрос был распознан сервером, но метод, указанный в запросе, был отклонен сервером. Вы можете проверить справочное руководство по API, чтобы узнать, какие методы ожидает сервер. Ответ сервера может также включать заголовок Allow, содержащий список допустимых методов для запрашиваемого ресурса.

Например, если вы по ошибке используете метод POST для API, который ожидает метод GET, вы получите ошибку 405.

```
import requests
url = "http://myservice/api/v1/resources/house/rooms/id/1"
resp = requests.post(url,auth=("person2","super"),verify = False)
print(resp.status_code)
print(resp.text)
```

406 - Not Acceptable

Эта ошибка указывает на то, что целевой ресурс не имеет текущего представления, которое было бы приемлемо для клиента. Сервер имеет данные, но не может представить их, используя любой из вариантов, перечисленных в заголовках Accept- клиента.
Например, клиент запрашивает изображения SVG: Accept: image/svg+xml

```
import requests
headers = {'Accept': 'image/svg+xml'}
url = "http://myservice/api/v1/resources/house/rooms/id/1"
resp =
requests.get(url,headers=headers,auth=("person2","super"),verify =
False)
print(resp.status_code)
print(resp.text)
```

В данном случае сервер не поддерживает запрошенный ресурс image/svg+xml, поэтому он отвечает ошибкой 406.

407 - Proxy Authentication Required

Этот код похож на 401 (Unauthorized), но указывает на то, что клиент должен сначала аутентифицироваться на прокси-сервере. В данном сценарии между клиентом и сервером находится прокси-сервер, и код ответа 407 указывает на то, что клиенту необходимо сначала пройти аутентификацию на прокси-сервере.

409 - Запрос не может быть выполнен из-за конфликта с текущим состоянием целевого ресурса

Например, конфликт редактирования, когда ресурс редактируется несколькими пользователями, приведет к ошибке 409. Повторный запрос может быть успешным, если конфликт будет разрешен сервером.

415 - Unsupported Media Type

В этом случае клиент отправил тело запроса в формате, который сервер не поддерживает. Например, если клиент отправляет XML на сервер, который принимает только JSON, сервер вернет ошибку 415.

```
import requests
headers = {"content-type":"application/xml"}
url = "http://myservice/api/v1/resources/house/rooms/id/1"
resp =
requests.get(url,headers=headers,auth=("person2","super"),verify =
False)
print(resp.status_code)
print(resp.text)
```

Опустите заголовок или добавьте заголовок {"content-type":"application/json"}

```
import requests
headers = {"content-type":"application/json"}
url = "http://myservice/api/v1/resources/house/rooms/id/1"
resp =
requests.get(url,headers=headers,auth=("person2","super"),verify =
False)
print(resp.status_code)
print(resp.text)
```

Это только самые распространенные коды ответа 4xx. Если вы встретите другие коды ответа 4xx, вы можете обратиться к RFC 2616, раздел 6.1, Status-Line или RFC 7231, раздел 6, Response Status Codes для получения дополнительной информации о том, что они означают.

Статусы 5xx

5xx ошибка на стороне сервера

500 - Internal Server Error

Эта ошибка означает, что сервер столкнулся с непредвиденным условием, которое не позволило ему выполнить запрос.

501 - Not Implemented

Эта ошибка означает, что сервер не поддерживает функциональность, необходимую для выполнения данного запроса. Например, сервер ответит кодом 501, если он не распознает метод запроса и поэтому не может поддерживать его для какого-либо ресурса.

502 - Bad Gateway

Эта ошибка означает, что сервер, действуя в качестве шлюза или прокси, получил недопустимый ответ от входящего сервера, к которому он обращался при попытке выполнить запрос.

503 - Service Unavailable

Этот код указывает на то, что в настоящее время сервер не может обработать запрос из-за временной перегрузки или планового обслуживания, которое, вероятно, будет устранено после некоторой задержки.

504 - Gateway Timeout

Эта ошибка означает, что сервер, действуя в качестве шлюза или прокси, не получил своевременного ответа от вышестоящего сервера, к которому ему необходимо было обратиться для завершения запроса.

Если вы получаете ошибку 500 или 501, проверьте справочное руководство по API, чтобы убедиться в правильности запроса. В случае других ошибок 5xx обратитесь к администратору сервера API, чтобы решить проблему.

Прежде чем приступить к устранению неполадок в API, необходимо иметь под рукой справочное руководство по API и справку о кодах состояния. Если вы не можете получить код ответа, вы, скорее всего, сможете определить причину из журнала ошибок в вашем скрипте. Если вам удалось получить код ответа, вы сможете определить основную причину ошибки, включая то, на чьей стороне ошибка - клиента или сервера, поняв код ответа.

Резюме

Знакомство с API

Интерфейс прикладного программирования (API) определяет способы взаимодействия пользователей, разработчиков и других приложений с компонентами приложения. API может использовать общие веб-интерфейсы или коммуникационные протоколы, а также собственные стандарты. Некоторые случаи использования — это задачи автоматизации, интеграция данных и функциональность. Большинство современных API встроены в продукт. Такие API обычно тщательно тестируются, надежны и иногда даже используются самим продуктом.

Стили проектирования API

API могут предоставляться синхронно или асинхронно. Синхронные API отвечают на запрос напрямую, обычно предоставляя данные (или другой подходящий ответ) немедленно. Асинхронные API предоставляют ответ, свидетельствующий о том, что запрос был получен, но этот ответ не содержит никаких фактических данных. Сервер обрабатывает запрос, что может занять некоторое время, и посылает уведомление (или запускает обратный вызов) с данными после обработки запроса. После этого клиент может действовать на основе полученных данных.

Архитектурные стили API

Три наиболее популярных архитектурных стиля API - это RPC, SOAP и REST.

Удаленный вызов процедур (RPC) — это модель "запрос-ответ", которая позволяет приложению (выступающему в роли клиента) осуществлять процедурный вызов другого приложения (выступающего в роли сервера). Серверное" приложение обычно расположено на другой системе в сети.

Простой протокол доступа к объектам (SOAP) — это протокол обмена сообщениями. Он используется для связи между приложениями, которые могут быть на разных платформах или созданы на разных языках программирования. Это протокол на основе XML, который обычно используется с транспортом HyperText Transfer Protocol (HTTP), но может быть применен и к другим протоколам. SOAP является независимым, расширяемым и нейтральным.

REpresentational State Transfer (REST) — это архитектурный стиль, который был создан как гибридный стиль, производный от нескольких сетевых архитектурных стилей, объединенных с дополнительными ограничениями, которые определяют единый интерфейс коннектора. Существует шесть ограничений, применяемых к элементам архитектуры: Клиент-сервер, Stateless, Cache, Uniform Interface, Layered System и Code-On-Demand. Эти шесть ограничений могут быть применены к любому протоколу, и когда они применяются, вы часто услышите, что это RESTful.

Введение в REST API

API веб-службы REST (REST API) — это интерфейс программирования, который взаимодействует через HTTP, придерживаясь принципов архитектурного стиля REST. Запросы REST API — это, по сути, HTTP-запросы, которые следуют принципам REST. Эти запросы — способ для приложения (клиента) попросить сервер выполнить определенную функцию. Ответы REST API — это, по сути, HTTP-ответы. Эти ответы передают результаты HTTP-запроса клиента. Ответ может содержать данные, которые были запрошены, означать, что сервер получил запрос, или даже информировать клиента о том, что с его запросом возникли проблемы. Диаграммы последовательности используются для объяснения последовательности обменов или событий. Они представляют собой сценарий упорядоченного набора событий. Их также называют диаграммами событий.

Аутентификация в REST API

В целях безопасности большинство REST API требуют аутентификации, чтобы случайные пользователи не могли ошибочно или злонамеренно создавать, обновлять или удалять информацию, а также получать доступ к информации, которая не должна быть общедоступной. Без аутентификации REST API позволяет любому получить доступ к функциям и системным сервисам, которые были открыты через интерфейс. Аутентификация — это акт проверки личности пользователя. Пользователь доказывает, что он тот, за кого себя выдает. Авторизация - это подтверждение пользователем того, что у него есть разрешения на выполнение запрошенного действия на данном ресурсе. К

распространенным типам механизмов аутентификации относятся Basic, Bearer и API Key. Открытая авторизация, также известная как OAuth, объединяет аутентификацию и авторизацию.

Ограничение скорости API

Использование ограничения скорости API — это способ для веб-службы контролировать количество запросов, которые пользователь или приложение может сделать за определенную единицу времени. К распространенным алгоритмам ограничения скорости относятся: Leaky bucket, Token bucket, счетчик с фиксированным окном и счетчик со скользящим окном. Многие API с ограничением скорости добавляют информацию об ограничении скорости в заголовок ответа. Поскольку стандарта не существует, пара ключ-значение, используемая в заголовке, может отличаться в разных API. Некоторые часто используемые ключи включают:

- X-RateLimit-Limit: максимальное количество запросов, которое может быть сделано за определенную единицу времени;
- X-RateLimit-Remaining: Количество оставшихся запросов, которые запросчик может сделать в текущем окне ограничения скорости;
- X-RateLimit-Reset: Время, когда окно ограничения скорости будет сброшено;
- Если лимит скорости превышен, сервер автоматически отклоняет запрос и отправляет HTTP-ответ, информирующий пользователя. Кроме того, обычно ответ, содержащий ошибку "Превышен лимит скорости", также включает в себя значимый код состояния HTTP.

Работа с Webhooks

Webhooks — это обратный вызов HTTP или HTTP POST на указанный URL, который уведомляет ваше приложение, когда определенное действие или "событие" произошло в одном из ваших ресурсов на платформе. Webhooks позволяют приложениям получать данные в режиме реального времени, поскольку они запускаются при определенных действиях или событиях. Благодаря веб-крючкам приложения становятся более эффективными, поскольку им больше не требуется механизм опроса. Чтобы получить уведомление от провайдера webhook, приложение должно быть постоянно запущено, чтобы принимать HTTP POST-запросы, и оно должно зарегистрировать URI у провайдера webhook, чтобы провайдер знал, куда отправлять уведомление при наступлении целевых событий. В дополнение к этим двум требованиям приложение должно обрабатывать входящие уведомления от сервера webhook.

Устранение неполадок с вызовами API

Прежде чем приступить к устранению неполадок, убедитесь, что у вас под рукой справочное руководство по API и информация об аутентификации API.

Неверно, что каждый запрос, отправленный на сервер REST API, возвращает ответ с кодом состояния. Хотя можно ожидать, что всегда будет

получен код состояния HTTP, такой как 2xx, 4xx и 5xx, существует много случаев, когда сервер API не может быть достигнут или не отвечает. В таких случаях вы не получите код состояния HTTP, потому что сервер API не может отправить ответ.

Хотя неотвечающий сервер может быть большой проблемой, основную причину неотвечающего сервера можно легко определить. Обычно можно определить, что пошло не так, по сообщениям об ошибках, полученным в результате запроса. Это может быть ошибка на стороне клиента, ошибка пользователя, неправильный URI, неправильный домен, проблема подключения, недействительный сертификат, ошибка на стороне сервера или проблема связи между сервером и клиентом. Как вы можете решить эту проблему? Правильно интерпретируя коды состояния. Коды 4xx - это ошибки на стороне клиента, а коды 5xx - ошибки на стороне сервера.

Дополненная Глава 4

Serverless вычисления

Начнем с важного момента: говорить, что приложения "бессерверные" — это здорово для маркетинга, но технически это не так. Конечно, ваше приложение работает на сервере. Оно просто работает на сервере, который вы не контролируете и о котором вам не нужно думать. Отсюда и название "serverless".

Бессерверные вычисления используют преимущества современной тенденции к созданию приложений, построенных на основе сервисов. То есть приложение обращается к другой программе или рабочей нагрузке для выполнения определенной задачи, чтобы создать среду, в которой приложения предоставляются по мере необходимости.

Это работает следующим образом (Рисунок 76):

Шаг 1. Вы создаете свое приложение.

Шаг 2. Вы развертываете свое приложение в виде контейнера, чтобы оно могло легко запускаться в любой подходящей среде.

Шаг 3. Вы развертываете этот контейнер у поставщика бессерверных вычислений, например, AWS Lambda, Google Cloud functions или даже во внутренней инфраструктуре Function as a Service. Это развертывание включает в себя спецификацию того, как долго функция должна оставаться неактивной, прежде чем она будет свернута.

Шаг 4. Когда это необходимо, ваше приложение вызывает функцию.

Шаг 5. Провайдер запускает экземпляр контейнера, выполняет необходимую задачу и возвращает результат.

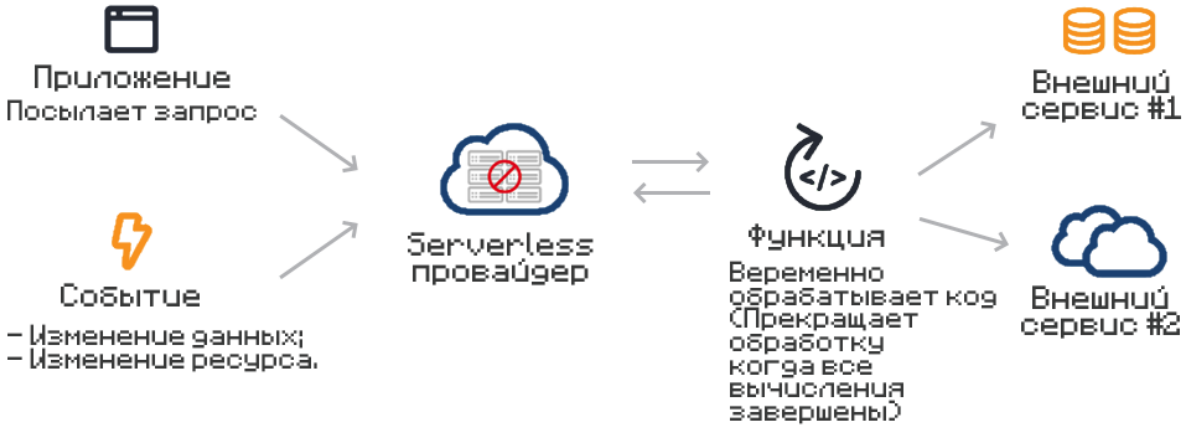


Рисунок 76 — Serverless вычисления

Serverless вычисления снимают ответственность за распределение ресурсов с разработчика и несут расходы только в момент выполнения приложения.

Хотя есть огромное преимущество в том, что вы платите только за те ресурсы, которые действительно используются, в отличие от виртуальной машины, которая может работать постоянно, даже когда ее мощности не нужны. Бессерверная модель вычислений означает, что вы не имеете никакого контроля над хост-машиной, поэтому с точки зрения безопасности она может оказаться неприемлемой.

Типы инфраструктуры

В первые дни существования компьютеров инфраструктура была довольно простой. Программное обеспечение, которое вы так тщательно писали, работало на одном компьютере. В конце концов, у вас появилась сеть, которая могла соединить несколько компьютеров вместе. Дальше все становилось все сложнее и сложнее. Давайте рассмотрим различные варианты создания инфраструктуры, например, различные типы облаков, и что каждое из них делает и не делает хорошо.

Локальная

Технически говоря, "локальный" означает любую систему, которая находится буквально в пределах вашего здания. В данном случае речь идет о традиционных центрах обработки данных, в которых размещаются отдельные машины, предоставляемые для приложений, а не об облаках, внешних или иных.

Эти традиционные инфраструктуры представляют собой центры обработки данных с серверами, предназначенными для отдельных приложений, или виртуальными машинами, которые, по сути, позволяют одному компьютеру действовать как несколько компьютеров.

Для эксплуатации традиционного локального центра обработки данных требуется заказать, получить, собрать в стойки ("racked and stacked") серверы, устройства хранения данных и сетевое оборудование, перевезти их на место,

подключить к электропитанию и передаче данных. Это оборудование должно быть обеспечено дополнительными услугами, такими как защита электропитания, охлаждение и предотвращение пожара. Затем необходимо логически настроить серверы в соответствии с их ролями, установить операционные системы и программное обеспечение, а также обеспечить обслуживание и мониторинг всего этого.

Вся эта работа по созданию инфраструктуры требует времени и усилий. Запросы на ресурсы должны проходить через операционную группу, что может привести к задержкам в несколько дней, недель или даже месяцев, пока новое оборудование будет получено, подготовлено и предоставлено.

Кроме того, масштабирование приложения обычно означает его перенос на более крупный сервер, что делает увеличение или уменьшение масштаба важным событием. Это означает, что приложение почти всегда либо тратит деньги на избыточные мощности, которые не используются, либо работает неэффективно из-за недостатка ресурсов.

Эти проблемы можно решить, перейдя на облачное решение.

Частное облако

Недостатки локальной инфраструктуры можно легко решить с помощью облачных вычислений. Облако — это система, которая обеспечивает самостоятельное предоставление вычислительных ресурсов, сетей и хранилищ.

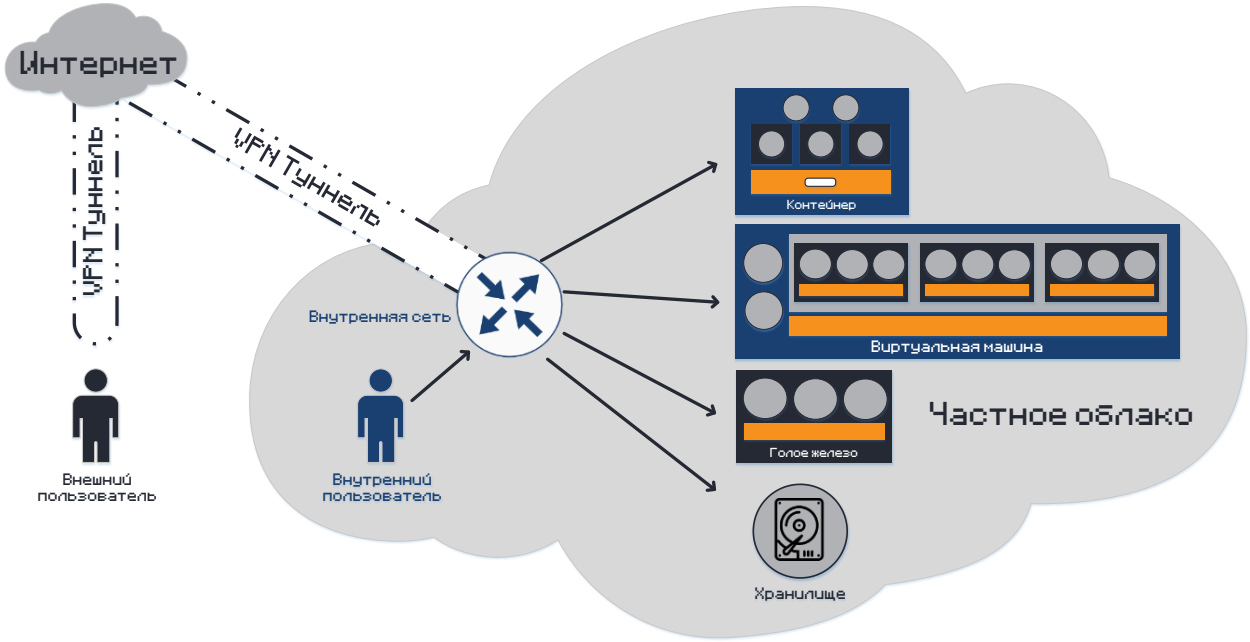


Рисунок 77 — Частное облако

В инфраструктуре частного облака (Рисунок 77) организация контролирует все ресурсы. Облако состоит из управляющей плоскости, которая позволяет выполнять запросы. Вы можете создать новую ВМ,

подключить том хранения, даже создать новую сеть и вычислительные ресурсы.

Облака предоставляют сервисный доступ к вычислительным ресурсам, таким как виртуальные машины, контейнеры и даже голый металл. Это означает, что пользователи могут войти в дашборд или использовать командную строку для самостоятельного запуска новых ресурсов, а не ждать, пока ИТ-отдел решит проблему. Такие платформы иногда называют инфраструктурой как услугой (IaaS). К распространенным платформам частных облаков относятся VMware (собственная разработка), OpenStack (с открытым исходным кодом) и Kubernetes (система оркестровки контейнеров). Базовая аппаратная инфраструктура для облаков может быть представлена обычными сетевыми "пустыми" серверами или более современными управляемыми "пустыми" или "гиперконвергентными" решениями физической инфраструктуры, такими как Cisco UCS и Cisco HyperFlex соответственно.

Частное облако отличается от других типов облаков тем, что все ресурсы внутри облака находятся под контролем вашей организации. В большинстве случаев частное облако располагается в вашем центре обработки данных, но технически это не является обязательным условием для того, чтобы называться "частным". Важно то, что все ресурсы, работающие на оборудовании, принадлежат организации-владельцу.

Преимущество частного облака заключается в том, что вы полностью контролируете место его расположения, что важно в ситуациях, когда существуют специальные нормативные требования, а также в том, что вам обычно не нужно беспокоиться о других рабочих нагрузках в системе.

С другой стороны, вам необходимо иметь операционную команду, которая будет управлять облаком и поддерживать его в рабочем состоянии.

Публичное облако

Публичное облако — это, по сути, то же самое, что и частное облако, но управляется оно провайдером публичного облака. В публичных облаках также могут работать такие системы, как OpenStack или Kubernetes, или же это могут быть конкретные собственные облака, такие как Amazon Web Services или Azure.

Клиенты публичного облака могут совместно использовать ресурсы с другими организациями: ваша ВМ может работать на одном узле с ВМ, принадлежащей кому-то другому. В качестве альтернативы провайдеры публичного облака могут предоставлять клиентам выделенную инфраструктуру. Большинство из них предоставляют несколько географически разделенных облачных "регионов", в которых могут быть размещены рабочие нагрузки. Это позволяет размещать рабочие нагрузки близко к пользователям (минимизируя задержки), поддерживать географическую избыточность (регионы Восточного и Западного побережья вряд ли будут отключены одновременно) и обеспечивать юрисдикционный контроль над тем, где хранятся данные. Рисунок 78

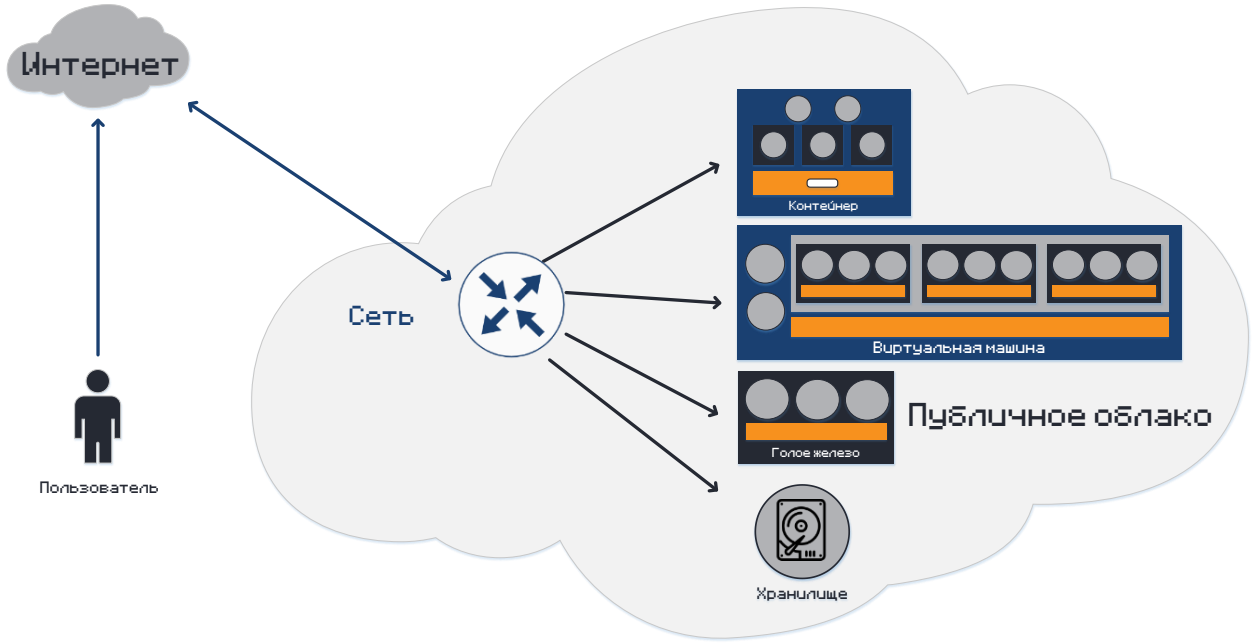


Рисунок 78 — Публичное облако

Публичные облака могут быть полезны, поскольку вам не нужно платить за оборудование, которое вы не собираетесь использовать, поэтому масштабирование может продолжаться практически до бесконечности. Так как вы платите только за те ресурсы, которые фактически используете, это решение может быть наиболее экономичным, поскольку ваше приложение никогда не исчерпает ресурсы, и вы не будете платить за ресурсы, которые не используете. Вам также не нужно беспокоиться о техническом обслуживании и эксплуатации оборудования; этим занимается провайдер общедоступного облака. Однако на практике, когда ваше облако достигает определенного размера, преимущества в стоимости исчезают, и вам лучше использовать частное облако.

У публичного облака есть один недостаток. Поскольку вы используете облако совместно с другими пользователями, вам, возможно, придется столкнуться с ситуацией, когда другие рабочие нагрузки занимают больше, чем их доля ресурсов.

Гибридное облако

Нетрудно догадаться, гибридное облако — это действительно комбинация двух различных типов облаков. Как правило, используется для объединения частного и публичного облака в рамках одного приложения (Рисунок 79).

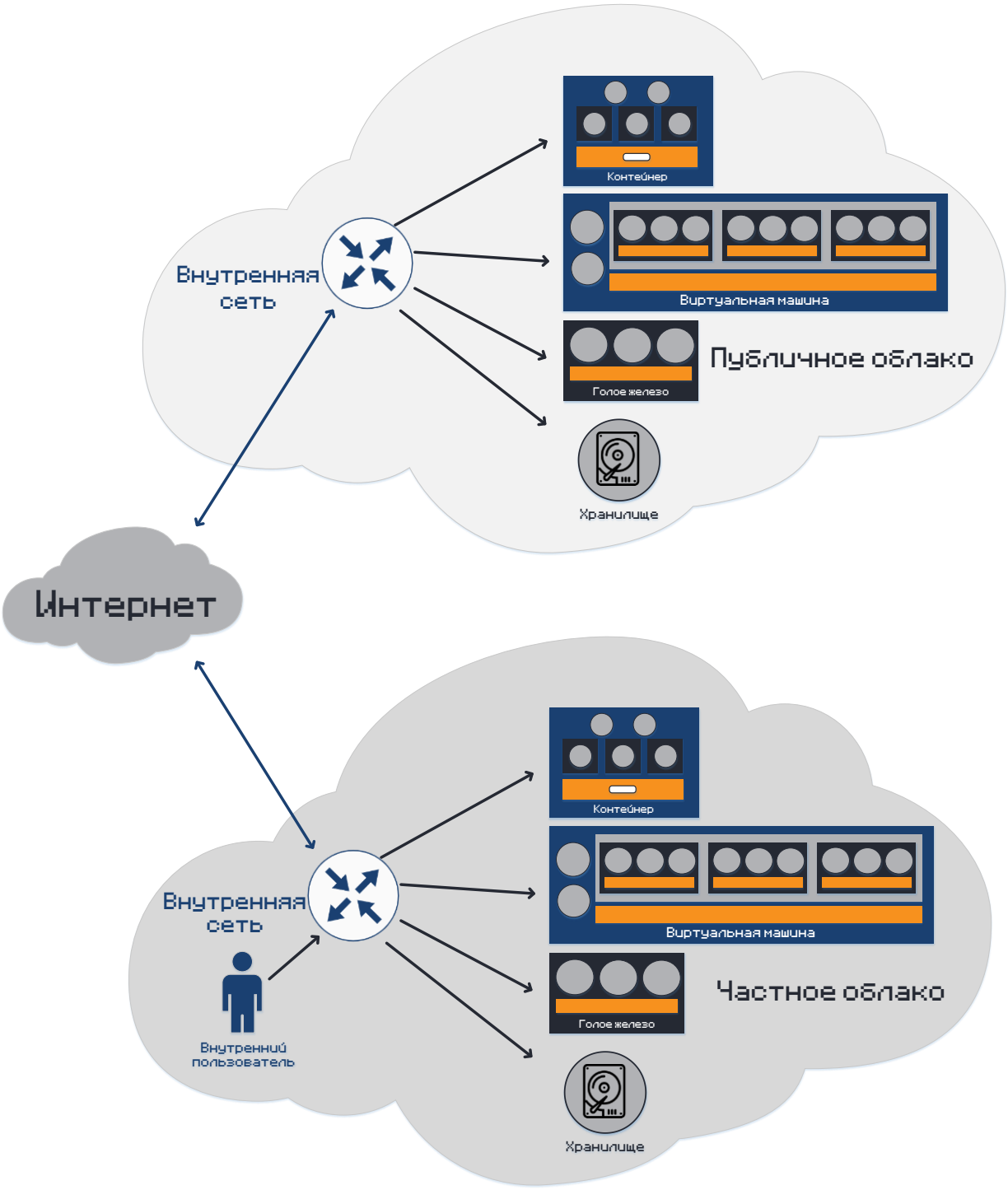


Рисунок 79 — Гибридное облако

Безопасность приложений

Обеспечение безопасности данных

Ни для кого не секрет, что безопасность является одной из главных проблем в современном мире. Это относится как к данным, так и к приложениям. Если одно защищено, а другое нет, то уязвимы оба.

В этой части мы рассмотрим некоторые вопросы, связанные с обеспечением безопасности данных и приложения.

Лучшие практики хранения зашифрованных данных

Когда речь идет о защите информации в состоянии покоя, необходимо принять во внимание несколько моментов.

Шифрование данных

Вы наверняка видели множество новостей об утечках данных. Как правило, речь идет о доступе отдельных лиц к данным, которые хранятся, но не защищены. В данном контексте это означает, что данные хранятся таким образом, что человек не только может получить доступ, но и легко увидеть и использовать данные.

В идеале неавторизованные лица или приложения никогда не получат доступ к вашим системам, но, очевидно, вы не можете этого гарантировать. Поэтому, когда человек с плохими намерениями (а это может быть и недовольный сотрудник, имеющий законный доступ) получает доступ к вашей базе данных, вы не хотите, чтобы он увидел нечто подобное (Таблица 7):

Таблица 7 — Данные пользователей

first_name	Last_name	username	password	birthdate
Валерий	Жмышенко	gladvalakas	aboba	13.09.1989
Сан	Саныч	obeme	abcixba	13.09.1955

Вместо этого было бы здорово если бы, они увидели что-то вроде этого (Таблица 8):

Таблица 8 — Данные пользователей, но лучше

first_name	Last_name	username	password	birthdate
Валерий	Жмышенко	gladvalakas	a6c79a27049109e472b246b5dfbe08aedff1e9e2259597e54032dbad4958d4ad	621648000
Сан	Саныч	obeme	485423b9cffbf74d5cb8aa5abe2c761da07c97e7f35304a1a3b03127d391702b	-451353600

Существует два метода шифрования данных: одностороннее шифрование и двустороннее шифрование.

Двустороннее шифрование — это буквально то, что звучит: вы шифруете данные с помощью ключа, а затем можете использовать этот ключ (или его разновидность) для расшифровки данных, чтобы получить их обратно в открытом виде. Вы можете использовать его для информации, доступ к которой вам необходим в исходном виде, например, медицинские записи или номера социального страхования.

Одностороннее шифрование проще, поскольку вы можете легко создать зашифрованное значение без необходимости использования определенного ключа, но не можете его расшифровать. Это можно использовать для

информации, которую не нужно извлекать, а только сравнивать, например, для паролей. Например, допустим, у вас есть пользователь `denis`, у которого пароль `neftejugansk`. Данные могут храниться вот так (Таблица 9):

Таблица 9 — Данные пользователей с одностороннее шифрованим

username	password
denis	@#$%SD@$$drw

В этом случае шифрование `neftejugansk` дает `@#$%SD@$$drw`, но нет никакого способа получить "`neftejugansk`" обратно. Чтобы проверить пароль Дениса, когда он снова войдет в систему, вам нужно будет сделать примерно следующее:

```
select * from users where username = ‘denis’ and
scrambled_password=scrambledversion(‘neftejugansk’)
```

В результате:

```
select * from users where username = ‘denis’ and
scrambled_password=’@#$%SD@$$drw’
```

Конечно, возникает вопрос: если вы собираетесь шифровать свои данные с помощью ключа, где безопасно хранить ключ? У вас есть несколько различных вариантов, от специализированного оборудования (хорошо, но сложно и дорого), использования службы управления ключами, например Amazon Key Management Service (использует специализированное оборудование, но проще и дешевле), до хранения в самой базе данных (что не является лучшей практикой, не имеет специализированного оборудования или физических характеристик и уязвимо для атак).

Уязвимости программного обеспечения

Когда заходит речь об уязвимостях программного обеспечения, необходимо беспокоиться о существующих своих и чужих уязвимостях.

Большинство разработчиков не являются экспертами в области безопасности, поэтому нередки случаи непреднамеренного внесения уязвимостей в ваше приложение. По этой причине существует ряд различных инструментов сканирования кода, таких как Bandit, Brakeman и VisualCodeGrepper, которые сканируют ваш код в поисках известных проблем. Эти проблемы могут быть встроены в код, который вы написали сами, или связаны с использованием других библиотек.

Эти “другие” библиотеки — то, как вы получаете чужие уязвимости. Даже в программном обеспечении, которое используется десятилетиями, могут быть проблемы, например, ошибка Heartbleed, обнаруженная в OpenSSL, программном обеспечении, на котором основана большая часть Интернета. Программное обеспечение существует с 1998 года, но ошибка появилась в

2012 году и пролежала незамеченной в течение двух лет, прежде чем её обнаружили и исправили.

Хранение слишком большого количества данных

Помните, что хакеры не могут получить то, что вы не храните. Например, если вам нужен только код авторизации кредитной карты для периодического выставления счетов, нет смысла хранить весь номер кредитной карты. Это особенно важно, когда речь идет о персональных данных, таких как номера социального страхования и дни рождения, а также о другой информации, которая может считаться "частной", например, истории пользователя.

Если данные не нужны вам для выполнения важных функций, не храните их.

Хранение данных в облаке

Помните, что при хранении данных в "облаке" вы, по определению, храните их на чужом компьютере. Хотя во многих случаях безопасность "облачных" поставщиков может быть лучше, чем у большинства предприятий, вы все равно сталкиваетесь с проблемой, что эти серверы находятся вне вашего контроля. Вы не знаете, кто из сотрудников имеет к ним доступ, и даже не знаете, что происходит с жесткими дисками, выведенными из эксплуатации. Это особенно актуально при использовании SDD в качестве хранилища, поскольку архитектурная структура твердотельного накопителя затрудняет или делает невозможным истинное удаление всех секторов. Убедитесь, что ваши облачные данные зашифрованы или защищены иным способом.

Роуминговые устройства

В мае 2006 года Департамент по делам ветеранов США потерял ноутбук, содержавший базу данных с личной информацией о 26,5 миллионах ветеранов и военнослужащих. Ноутбук в конечном итоге был восстановлен, но это по-прежнему является отличным примером того, почему информация должна храниться безопасным образом, особенно потому, что сейчас рабочая сила в мире гораздо более мобильна, чем в 2006 году.

Кроме того, приложения все чаще находятся на устройствах, которые даже более портативны, чем ноутбуки, таких как планшет и особенно мобильный телефон. Их просто легче потерять. Это могут быть даже не традиционные приложения, такие как базы данных, а приложения, ориентированные на конечного пользователя. Убедитесь, что ваши данные не остаются уязвимыми, зашифровав их при любой возможности.

Лучшие практики передачи данных

Данные также уязвимы при передаче. На самом деле, они могут быть еще более уязвимы из-за того, как устроен Интернет, где пакеты проходят через множество серверов (которые могут принадлежать или не принадлежать вам) на пути к конечному пункту назначения.

Такая структура делает ваши данные уязвимыми для атак типа "чел посередине", при которых сервер на пути следования может наблюдать, красть

и даже изменять данные по мере их прохождения. Для предотвращения этих проблем вы можете использовать:

- **SSH** — При подключении к вашим серверам всегда используйте безопасный протокол, такой как SSH, или защищенный shell, а не небезопасный протокол, такой как Telnet. SSH обеспечивает аутентификацию и шифрование сообщений между исходной и целевой машинами, что затрудняет или делает невозможным подглядывание за вашими действиями.
- **TLS** — В наши дни подавляющее большинство запросов к браузеру и от него используют протокол `https://` (а не `http://`). Первоначально этот протокол назывался SSL, или Secured Sockets Layer, но с годами он был постепенно заменен на TLS, или Transport Layer Security. TLS обеспечивает аутентификацию сообщений и более надежные шифры, чем его предшественник. При любой возможности вы должны использовать TLS.
- **VPN** — пожалуй, самое важное средство защиты вашего приложения. VPN позволяет держать весь трафик, связанный с приложением, внутри вашей сети, даже при работе с удаленными сотрудниками. Удаленный сотрудник подключается к VPN-серверу, который действует как прокси-сервер и шифрует весь трафик, идущий к пользователю и от него.

Использование VPN имеет несколько преимуществ. Во-первых, трафик к пользователю и от него не подвержен прослушиванию или манипуляциям, поэтому никто не сможет использовать это соединение для нанесения ущерба вашему приложению или сети. Во-вторых, поскольку пользователь по сути находится внутри частной сети, вы можете ограничить доступ к ресурсам разработки и развертывания, а также к ресурсам, которые не должны быть доступны конечным пользователям, таким как необработанные базы данных.

Что такое SQL-инъекция?

SQL-инъекция — это техника инъекции кода, используемая для атаки на приложения, управляемые данными, в которых вредоносные SQL-запросы вставляются в поле ввода для выполнения (например, для сброса содержимого базы данных злоумышленнику). SQL-инъекция должна использовать уязвимость в программном обеспечении приложения. Двумя примерами являются случаи, когда пользовательский ввод либо неправильно отфильтровывается на предмет символов экранирования строковых литералов, встроенных в SQL-запросы, либо пользовательский ввод не имеет строгой типизации и неожиданно выполняется. Инъекция SQL в основном известна как вектор атаки на веб-сайты, но может быть использована для атаки на любой тип базы данных SQL.

Атаки с использованием SQL-инъекций позволяют злоумышленникам подменить личность, подделать существующие данные, вызвать проблемы с отпиской, такие как аннулирование транзакций или изменение баланса,

позволить полностью раскрыть все данные в системе, уничтожить данные или сделать их недоступными, а также стать администраторами сервера базы данных.

SQL в веб-страницах

SQL-инъекция — одна из самых распространенных технологий взлома веб-страниц. Это размещение вредоносного кода в SQL-запросах через ввод данных на веб-странице. Обычно это происходит, когда вы просите пользователя ввести данные, например, имя пользователя/userid, а вместо имени/id пользователь дает вам SQL-запрос, который вы неосознанно запускаете в своей базе данных.

Посмотрите на следующий пример, который создает оператор SELECT путем добавления переменной (uid) в строку select. Переменная извлекается из пользовательского ввода с помощью `request.args("uid")`:

```
uid = request.args("uid");
str_sql = "SELECT * FROM Users WHERE UserId = " + uid;
```

Одним из примеров является SQL-инъекция на основе выражения 1=1, которое всегда верно (на языке SQL).

Посмотрите на код для создания SQL-оператора для выбора профиля пользователя по UID, с заданным UID UserProfile.

Если нет валидатора ввода для предотвращения ввода пользователем "неправильных" данных, пользователь может ввести данные следующим образом:

UID:

```
2022 OR 1=1
```

Выходной SQL-запрос будет выглядеть следующим образом:

```
SELECT * FROM UserProfiles WHERE UID = 2022 OR 1=1;
```

Приведенный выше оператор SQL является корректным, но вернет все строки из таблицы "UserProfiles", поскольку OR 1=1 всегда TRUE.

Что произойдет, если таблица "UserProfiles" содержит имена, электронные письма, адреса и пароли?

SQL-запрос будет выглядеть следующим образом:

```
SELECT UID, Address, Email, Name, Password FROM UserProfiles WHERE
UID = 2022 or 1=1;
```

Создатель вредоносной программы или хакер может получить доступ ко всем профилям пользователей в базе данных, просто введя в поле ввода 2022 OR 1=1.

Другой пример — SQL Injection на основе ""="" всегда истинно. Вот этот пример.

Имя пользователя: user_a Пароль: pass_user_a

Пример:

```
u_name = request.args("uid");
u_pass = request.args("password");
sql = 'SELECT * FROM UserProfiles WHERE Name ="' + u_name + '" AND
Pass ="' + u_pass + '"'
```

Вот ожидаемый SQL-запрос:

```
SELECT * FROM UserProfiles WHERE Name ="user_a" AND Pass
="pass_user_a"
```

Но хакер может получить доступ к именам пользователей и паролям в базе данных, просто вставив " OR ""="" в текстовое поле имени пользователя или пароля:

```
User Name:
" OR ""="
Password:
" OR ""="
```

Код вывода создаст правильный SQL-запрос на стороне сервера, как показано ниже:

Вывод:

```
SELECT * FROM UserProfiles WHERE Name ="" OR ""="" AND Pass ="" OR
""=""
```

Приведенный выше SQL действителен и вернет все строки из таблицы "Users", поскольку OR ""="" всегда TRUE.

SQL-инъекция на основе пакетных SQL-запросов

Большинство баз данных поддерживают пакетные операторы SQL. Пакет SQL-запросов — это группа из двух или более SQL-запросов, разделенных точкой с запятой.

Приведенный ниже оператор SQL вернет все строки из таблицы "UserProfiles", а затем удалит таблицу "UserImages".

Пример:

```
SELECT * FROM UserProfiles; DROP TABLE UserImages
```

Посмотрите на следующий пример:

```
uid = request.args("uid");
strSQL = "SELECT * FROM UserProfiles WHERE UID = " + uid;
```

Теперь вы вводите UID:

```
2022; DROP TABLE UserImages
```

SQL-запрос будет выглядеть следующим образом:

```
SELECT * FROM Users WHERE UID = 2022; DROP TABLE UserImages;
```

Надеюсь, эти примеры помогут вам разработать формы получения данных таким образом, чтобы избежать этих распространенных взломов системы безопасности.

Как обнаружить и не допустить SQL-инъекцию

Уязвимость SQL-инъекции существует потому, что некоторые разработчики не заботятся о проверке данных и безопасности. Существуют инструменты, которые могут помочь обнаружить недостатки и проанализировать код.

Open source решения

Чтобы облегчить обнаружение атаки SQL-инъекции, разработчики создали хорошие механизмы обнаружения. В качестве примера можно привести SQLmap или SQLninja.

Инструменты анализа исходного кода

Инструменты анализа исходного кода, также называемые инструментами статического тестирования безопасности приложений (Static Application Security Testing или SAST), предназначены для анализа исходного кода и/или скомпилированных версий кода с целью поиска недостатков безопасности.

Эти инструменты могут автоматически находить такие недостатки, как переполнение буфера, недостатки SQL Injection и другие.

Обнаружить и предотвратить SQL-инъекции можно с помощью брандмауэра базы данных. Брандмауэры баз данных — это разновидность брандмауэров веб-приложений, которые контролируют базы данных для выявления и защиты от атак на базы данных. Эти атаки в основном направлены на получение доступа к конфиденциальной информации, хранящейся в базах данных.

Работа с брандмауэром базы данных

Фильтрация SQL-инъекций работает примерно так же, как фильтры спама в электронной почте. Брандмауэры баз данных обнаруживают SQL-инъекции на основе количества некорректных запросов от хоста, наличия внутри запроса блоков OR и UNION или других.

Используйте готовые отчеты

Использование подготовленных операторов с привязкой переменных (также известных как параметризованные запросы) — это то, как всех разработчиков следует учить писать запросы к базе данных. Они просты в

написании и более понятны, чем динамические запросы. Параметризованные запросы заставляют разработчика сначала определить весь код SQL, а затем передать каждый параметр в запрос. Такой стиль кодирования позволяет базе данных различать код и данные, независимо от того, что вводит пользователь.

Подготовленные операторы гарантируют, что злоумышленник не сможет изменить замысел запроса, даже если SQL-команды будут вставлены злоумышленником. В безопасном примере ниже, если бы злоумышленник ввел идентификатор пользователя tom' или '1'='1, параметризованный запрос не был бы уязвимым и вместо этого искал бы имя пользователя, которое буквально соответствовало бы всей строке tom' или '1'='1.

```
// Получение имени клиента из параметра
String              custname = request.getParameter("custname");
// Выполнение проверки ввода для обнаружения атак
String              query = "SELECT account_balance FROM user_data
WHERE user_name = ? ";
PreparedStatement   pStatement = connection.prepareStatement( query );
pStatement.setString( 1, custname);
ResultSet           results = pStatement.executeQuery();
```

Использование хранимых процедур

Хранимые процедуры не всегда защищены от SQL-инъекций. Однако некоторые стандартные конструкции программирования хранимых процедур при безопасной реализации имеют тот же эффект, что и использование параметризованных запросов. Это норма для большинства языков хранимых процедур.

Они требуют от разработчика просто создавать SQL-операторы с параметрами, которые автоматически параметризуются, если только разработчик не делает что-то в значительной степени выходящее за рамки нормы. Разница между подготовленными операторами и хранимыми процедурами заключается в том, что код SQL для хранимой процедуры определяется и хранится в самой базе данных, а затем вызывается из приложения. Обе эти техники одинаково эффективны для предотвращения SQL-инъекций, поэтому ваша организация должна выбрать, какой подход имеет для вас наибольший смысл.

Примечание: *"Реализовано безопасно" означает, что хранимая процедура не содержит небезопасной динамической генерации SQL. Разработчики обычно не генерируют динамический SQL внутри хранимых процедур. Никто не запрещает так делать, но по возможности этого следует избегать. Если этого нельзя избежать, хранимая процедура должна использовать проверку ввода или соответствующее экранирование. Это необходимо для того, чтобы убедиться, что все вводимые пользователем данные в хранимую процедуру не могут быть использованы для внедрения SQL-кода в динамически сгенерированный запрос. Аудиторы всегда должны*

искать использование `sp_execute`, `execute` *или* `exec` *в хранимых процедурах SQL Server. Аналогичные рекомендации по аудиту необходимы и для подобных функций других производителей.*

Существует также несколько случаев, когда хранимые процедуры могут увеличить риск. Например, в MS SQL server есть три основные роли по умолчанию: `db_datareader`, `db_datawriter` и `db_owner`. До появления хранимых процедур DBA предоставляли права `db_datareader` или `db_datawriter` пользователю веб-службы, в зависимости от требований. Однако хранимые процедуры требуют прав на выполнение, роль, которая недоступна по умолчанию. В некоторых системах, где управление пользователями централизовано, но ограничено этими тремя ролями, все веб-приложения запускаются с правами `db_owner`, чтобы хранимые процедуры могли работать. Это означает, что в случае взлома сервера злоумышленник получает полные права на базу данных, тогда как раньше он мог иметь только доступ для чтения.

Валидация ввода по Whitelist

Различные части SQL-запросов не являются законными местами для использования переменных привязки, например, имена таблиц или столбцов, а также индикатор порядка сортировки (ASC или DESC). В таких ситуациях наиболее подходящим средством защиты является валидация ввода или переработка запроса. Что касается имен таблиц или столбцов, то в идеале эти значения приходят из кода, а не из пользовательских параметров.

Но если значения пользовательских параметров используются для обращения к различным именам таблиц и столбцов, то значения параметров должны быть сопоставлены с законными/ожидаемыми именами таблиц или столбцов, чтобы убедиться, что непроверенный пользовательский ввод не попадет в запрос. Обратите внимание, что это симптом плохого дизайна, и следует рассмотреть возможность полного переписывания.

Пример проверки имени таблицы

```
String tableName;
switch(PARAM):
      case "Value1":
            tableName = "fooTable";
                        break;
      case "Value2":
            tableName = "barTable";
                  break;
      ...
      default:
            throw new InputValidationException("unexpected value
provided" +  PARAM  + " for table name");
```

Затем `tableName` может быть непосредственно добавлено в SQL-запрос, поскольку теперь известно, что это одно из законных и ожидаемых значений для имени таблицы в этом запросе. Помните, что общие функции проверки таблиц могут привести к потере данных, поскольку имена таблиц используются в запросах, где они не ожидаются.

Для чего-то простого, например, порядка сортировки, лучше всего, если вводимые пользователем данные преобразуются в булево значение, а затем это булево используется для выбора безопасного значения для добавления в запрос. Это стандартная необходимость при создании динамических запросов.

```
public String someMethod(boolean sortOrder) {
     String SQLquery = "some SQL ... order by Salary " + (sortOrder ?
"ASC" : "DESC");`
...
```

Любой пользовательский ввод может быть преобразован в нестроку (например, дату, числовой, булев или перечислимый тип). Если преобразование выполняется перед добавлением в запрос или используется для выбора значения для добавления в запрос, это преобразование обеспечивает безопасное добавление в запрос.

Валидация ввода также рекомендуется в качестве вторичной защиты во ВСЕХ случаях. Более подробная информация о том, как реализовать надежную проверку ввода в белый список, описана в документе Open Web Application Security Project (OWASP) Input Validation Cheat Sheet.

Экранирование всех введенных пользователем данных

Этот метод следует использовать только в крайнем случае, когда ни один из вышеперечисленных способов не подходит. Валидация ввода, вероятно, является лучшим выбором, поскольку эта методика слаба по сравнению с другими средствами защиты, и мы не можем гарантировать, что она предотвратит все SQL Injection во всех ситуациях.

Эта техника позволяет избежать ввода данных пользователем, прежде чем поместить их в запрос. Его реализация зависит от конкретной базы данных. Обычно ее рекомендуется использовать только для модернизации устаревшего кода, когда внедрение проверки ввода не является экономически эффективным. Приложения, создаваемые с нуля, или приложения, требующие низкой устойчивости к риску, должны быть созданы или переписаны с использованием параметризованных запросов, хранимых процедур или какого-либо объектно-реляционного маппера (Object Relational Mapper сокр. ORM), который создает запросы за вас.

Экранирование работает следующим образом. Каждая СУБД поддерживает одну или несколько схем экранирования символов, специфичных для определенных типов запросов. Если вы экранируете все вводимые пользователем данные, используя соответствующую схему экранирования для используемой вами базы данных, СУБД не будет путать

эти данные с SQL-кодом, написанным разработчиком, что позволит избежать возможных уязвимостей SQL-инъекций.

Существуют некоторые библиотеки и инструменты, которые можно использовать для экранирования ввода. Например, OWASP Enterprise Security API или ESAPI — это бесплатная библиотека управления безопасностью веб-приложений с открытым исходным кодом, которая облегчает программистам написание приложений с пониженным риском.

Библиотеки ESAPI предназначены для того, чтобы облегчить программистам доработку системы безопасности в существующих приложениях и послужить надежной основой для новых разработок.

Дополнительные средства защиты

Помимо принятия одного из четырех основных средств защиты, также рекомендуется принять все эти дополнительные средства, чтобы обеспечить защиту на всю глубину. Этими дополнительными средствами защиты могут быть:

- Минимизация привилегий;
- Несколько пользователей базы данных;
- Представления SQL.

Эволюция парольных систем

Простые текстовые пароли

Первыми паролями были простые обычные пароли, хранившиеся в базах данных. Они позволяли нескольким пользователям, использующим один и тот же процессор, иметь уникальные настройки конфиденциальности. Это было до появления сложных хакерских сетей и программ для взлома паролей.
Правило для паролей с открытым текстом очень простое: Просто храните их. У вас есть база данных с таблицей для всех ваших пользователей, и она, вероятно, выглядит примерно так (id, имя пользователя, пароль). После создания учетной записи вы сохраняете имя пользователя и пароль в этих полях в виде простого текста, а при входе в систему извлекаете строку, связанную с введенным именем пользователя, и сравниваете введенный пароль с паролем из базы данных. Если они совпадают, вы впускаете пользователя. Все очень просто и легко реализуемо.

Следующий пример показывает, как создать/проверить новый профиль пользователя в формате plaintext:

```
#######################################Простой текст
#########################################################
@app.route('/signup/v1', methods=['POST'])
def signup_v1():
    conn = sqlite3.connect(db_name)
    c = conn.cursor()
    c.execute('''CREATE TABLE IF NOT EXISTS USER_PLAIN
           (USERNAME  TEXT    PRIMARY KEY NOT NULL,
            PASSWORD  TEXT    NOT NULL);''')
```

```
    conn.commit()
    try:
        c.execute("INSERT INTO USER_PLAIN (USERNAME,PASSWORD) "
                  "VALUES ('{0}',
'{1}')".format(request.form['username'], request.form['password']))
        conn.commit()
    except sqlite3.IntegrityError:
        return "username has been registered."
    print('username: ', request.form['username'], ' password: ',
request.form['password'])
    return "signup success"
```

Давайте проверим новую учетную запись пользователя с функцией логина в формате plaintext, и увидим, что все данные профиля пользователя находятся в формате plaintext.

```
def verify_plain(username, password):
    conn = sqlite3.connect('test.db')
    c = conn.cursor()
    query = "SELECT PASSWORD FROM USER_PLAIN WHERE USERNAME =
'{0}'".format(username)
    c.execute(query)
    records = c.fetchone()
    conn.close()
    if not records:
        return False
    return records[0] == password
@app.route('/login/v1', methods=['GET', 'POST'])
def login_v1():
    error = None
    if request.method == 'POST':
        if verify_plain(request.form['username'],
request.form['password']):
            error = 'login success'
        else:
            error = 'Invalid username/password'
    else:
        error = 'Invalid Method'
    return error
```

Очевидно, что простой текст — это небезопасный способ хранения паролей. Если ваша база данных будет взломана, пароли ваших пользователей будут напрямую доступны хакерам.

Хеширование паролей

Хранение паролей рискованно и сложно одновременно. Простой подход к хранению паролей заключается в создании таблицы в базе данных, которая сопоставляет имя пользователя с паролем. Когда пользователь входит в

систему, сервер получает запрос на аутентификацию с полезной нагрузкой, содержащей имя пользователя и пароль. Мы ищем имя пользователя в таблице и сравниваем предоставленный пароль с сохраненным паролем. При совпадении пользователь получает доступ к приложению. Надежность и устойчивость этой модели зависит от того, как хранится пароль. Самый простой, но и наименее безопасный формат хранения паролей — это открытый текст.

Хранение паролей в открытом тексте эквивалентно записи их на листе цифровой бумаги. Если злоумышленник проникнет в базу данных и украдет таблицу паролей, он сможет получить доступ к каждой учетной записи пользователя. Эта проблема усугубляется тем, что многие пользователи повторно используют или используют вариации одного пароля, что потенциально позволяет злоумышленнику получить доступ к другим службам, отличным от той, которая подвергается атаке. Атака может исходить изнутри организации. Инженер-программист, имеющий доступ к базе данных, может воспользоваться этим правом доступа, получить учетные данные в открытом виде и получить доступ к любой учетной записи.

Хэширование

Более надежным способом хранения пароля является его преобразование в данные, которые невозможно преобразовать обратно в исходный пароль. Этот механизм известен как хэширование.

Согласно словарному определению, хэширование означает "измельчение чего-либо на мелкие кусочки", чтобы это выглядело как "запутанный беспорядок". Это определение близко к тому, что хэширование представляет собой в вычислительной технике.

В криптографии хэш-функция — это математический алгоритм, который сопоставляет данные любого размера с битовой строкой фиксированного размера. Этот вход функции может называться сообщением или просто входом. Выходная строка функции фиксированного размера известна как хэш или дайджест сообщения. Как утверждает OWASP, хэш-функции, используемые в криптографии, обладают следующими ключевыми свойствами:

- Легко и практично вычислить хэш, но трудно или невозможно повторно сгенерировать исходный вход, если известно только хэш-значение.
- Трудно создать исходный вход, который бы соответствовал определенному желаемому выходу.

Таким образом, в отличие от шифрования, хэширование является односторонним механизмом. Данные, которые хэшируются, не могут быть практически "отэшированы".

Пример хэширования

Вот пример кода для импорта метода конструктора хэш-алгоритма SHA-256 из модуля hashlib:

```
from hashlib import sha256
# Создайте экземпляр класса sha256
h = sha256()
# Использует метод update() для обновления хэш-объекта
h.update(b'zsspassword1')
# Использует метод hexdigest() для получения дайджеста строки,
переданной в метод update()
hash = h.hexdigest()
# Дайджест - это выход хэш-функции.
# Выведите переменную hash, чтобы увидеть значение hash в консоли:
print(hash)
```

В качестве примера можно привести следующий вывод:

```
7e50d63260ffb0d2523f78b7e12b2790908900d84c80fe623529dcac307a5fa6
```

“Солёный пароль”

Многие пароли зашифрованы, но их можно угадать. Их добывают на взломанных сайтах и помещают в списки. Эти списки значительно облегчают взлом хэшированных паролей. Чтобы гарантировать уникальность паролей, повысить их сложность и предотвратить атаки на пароли даже при одинаковых входных данных, на вход хэш-функции добавляется соль (это просто случайные данные). Такой пароль называется соленым.

Использование криптографического хэширования для более надежного хранения паролей

Необратимые математические свойства хэширования делают его феноменальным механизмом для сокрытия паролей в покое и в движении. Еще одно важнейшее свойство, которое делает хэш-функции пригодными для хранения паролей, — это их детерминированность. Детерминированная функция — это функция, которая при одинаковых входных данных всегда дает один и тот же выход. Это очень важно для аутентификации, поскольку вам нужна гарантия того, что при заданном пароле всегда будет получен один и тот же хэш. В противном случае было бы невозможно последовательно проверять учетные данные пользователей с помощью этой техники.

Добавление соли для хеширования паролей

Соль добавляется в процесс хэширования для обеспечения уникальности хэша. Это увеличивает сложность без увеличения требований пользователя и смягчает атаки на пароли, такие как радужные таблицы. В криптографии под солевым хэшем понимается добавление случайных данных на вход хэш-функции для обеспечения уникального выхода, которым является хэш, даже если входные данные одинаковы. Следовательно, уникальный хэш, полученный в результате добавления соли, может защитить от различных векторов атак, замедляя при этом атаки по словарю и перебором.

Дополнительные факторы для аутентификации

Даже при использовании этих средств проверки надежности паролей однофакторная аутентификация все равно оставляет ваше предприятие

уязвимым. Хакеры могут взломать любой пароль, это может занять значительное время, но они могут это сделать. Более того, они всегда могут украсть пароли с помощью социальной инженерии. Таким образом, вам следует внедрить многофакторную аутентификацию (MFA). Включение других факторов аутентификации сбивает с толку хакеров, которые могли взломать ваш пароль.

Однофакторная аутентификация (SFA)

Однофакторная аутентификация является самой простой формой методов аутентификации. При использовании SFA человек использует одну учетную запись для проверки себя в Интернете. Наиболее популярным примером этого является соответствие пароля (учетных данных) имени пользователя. Сегодня большинство проверок используют этот тип аутентификации.

Каковы риски однофакторной аутентификации?

Хакеры могут слить пароли пользователей на веб-сайты. Без дополнительного фактора к паролю, подтверждающего вашу личность, злоумышленнику для получения доступа нужен только ваш пароль.

Часто пароль пользователя прост, чтобы его было легко запомнить. Чем проще пароль, тем легче его взломать. Злонамеренный пользователь может угадать ваш пароль, потому что он знает вас лично или потому что ему удалось узнать о вас определенные сведения, такие как дата рождения, любимый актер/актриса или кличка домашнего питомца. Злонамеренный пользователь также может взломать ваш пароль, используя бота для генерации правильной комбинации букв/цифр, соответствующей вашему простому секретному методу идентификации. В любом из этих случаев восстановление вашей учетной записи (учетных записей) будет сопряжено с трудностями. Надеюсь, ваш простой пароль не используется повторно в других онлайн-структурах.

Однофакторная аутентификация (SFA) быстро становится слабым звеном мер безопасности, подобно кросс-доменам. Растет число продуктов, веб-сайтов и приложений, которые предлагают двухфакторную и многофакторную аутентификацию. Будь то два фактора, три или более, MFA — это способ значительно усложнить доступ злоумышленников к вашим учетным записям.

Двухфакторная аутентификация (2FA)

Двухфакторная аутентификация использует ту же комбинацию пароль/имя пользователя, но с добавлением просьбы подтвердить личность человека с помощью чего-то, принадлежащего только ему, например, мобильного устройства. Проще говоря, для подтверждения личности используются два фактора.

Многофакторная аутентификация (MFA)

Многофакторная аутентификация (MFA) — это метод контроля доступа к компьютеру, при котором пользователю предоставляется доступ только после успешного предъявления нескольких отдельных доказательств механизму аутентификации. Обычно для MFA требуется как минимум две из

следующих категорий: знание (то, что пользователь знает), владение (то, что он имеет) и неотъемлемость (то, чем он является).

2FA — это подмножество этих категорий. Это просто разновидность MFA, для которой требуется только два доказательства, то есть два "фактора". Когда вы входите в Google, Tumblr или LinkedIn, или совершаете покупку на Ozon, вы можете использовать их двухэтапную проверку, требующую ваш пароль (то, что вы знаете) и специальный текст, отправленный на ваш телефон (то, что у вас есть). Если у вас нет пароля и телефона, вы не сможете войти в систему.

Взлом паролей

Техника, используемая для подбора пароля, позволяющего войти в систему, известна как "взлом" защиты, предусмотренной паролем.

Угадывание паролей

Угадывание пароля — это онлайн-методика, которая заключается в попытке аутентификации конкретного пользователя в системе. Угадывание пароля может быть обнаружено путем мониторинга системных журналов неудачных входов. Уровни обрезания используются для разграничения злонамеренных атак и случайного ввода пароля обычными пользователями. Уровни обрезания определяют минимальный пороговый уровень отчетности. В примере с угадыванием пароля можно установить уровень обрезания таким образом, чтобы система аудита предупреждала только в том случае, если неудачная аутентификация происходит чаще, чем пять раз в течение часа для определенного пользователя. Уровни обрезания могут помочь отличить атаки от законных ошибок; однако они также могут вызвать ложные отрицательные результаты, если злоумышленники смогут определить порог, ниже которого они должны действовать.

Предотвращение успешных атак на угадывание пароля обычно осуществляется с помощью блокировки учетных записей. Блокировка учетных записей используется для того, чтобы злоумышленник не смог просто угадать правильный пароль, перебрав большое количество потенциальных паролей.

Атака по словарю

В криптоанализе и компьютерной безопасности атака по словарю — это форма атаки грубой силы для поражения шифра или механизма аутентификации путем попытки определить ключ дешифрования или парольную фразу, перебирая сотни, а иногда и миллионы вероятных вариантов, таких как слова в словаре.

Атака по словарю основана на переборе всех строк в заранее составленном списке, обычно полученном из списка слов, например, из словаря (отсюда фраза "атака по словарю"). В отличие от атаки методом грубой силы, при которой систематически перебирается большая часть

пространства ключей, при атаке по словарю перебираются только те возможности, которые считаются наиболее вероятными.
Атаки по словарю часто успешны, потому что многие люди склонны выбирать короткие пароли, которые представляют собой обычные слова или распространенные пароли, или простые варианты, полученные, например, путем добавления цифры или знака препинания. Атаки по словарю относительно легко победить, например, используя парольную фразу или выбирая пароль, который не является простым вариантом слова, встречающегося в любом словаре или списке часто используемых паролей.

Атака по заранее составленному словарю или атака по радужной таблице

Можно достичь компромисса между временем и пространством, предварительно вычислив список хэшей словарных слов и сохранив их в базе данных, используя хэш в качестве ключа. Это требует значительного времени на подготовку, но позволяет выполнить атаку быстрее. Требования к хранению предварительно вычисленных таблиц когда-то были одной из основных статей расходов, но сегодня это не так важно из-за низкой стоимости дискового хранилища. Атаки по заранее вычисленным словарям особенно эффективны, когда требуется взломать большое количество паролей. Предварительно вычисленный словарь нужно составить только один раз, и после его завершения хэши паролей могут быть просмотрены почти мгновенно в любое время, чтобы найти соответствующий пароль. Более тонкий подход предполагает использование радужных таблиц, которые снижают требования к хранению данных за счет немного большего времени поиска. Атаки по заранее вычисленным словарям, или "атаки по радужным таблицам", можно предотвратить с помощью соли — метода, который заставляет словарь хэшей заново вычисляться для каждого искомого пароля, делая предварительное вычисление неосуществимым, при условии, что количество возможных значений соли достаточно велико.

Социальная инженерия

Социальная инженерия приминетельна ко взлому паролей предполагает, что человек убеждает или обманывает другого человека, чтобы тот предоставил злоумышленнику доступ.

Культура информационной безопасности

Поведение сотрудников может оказать большое влияние на информационную безопасность в организациях. Культурные концепции могут помочь различным сегментам организации эффективно работать в направлении информационной безопасности. В работе "Exploring the Relationship between Organizational Culture and Information Security Culture" дается следующее определение культуры информационной безопасности: "ISC — это совокупность моделей поведения в организации, которые способствуют защите информации всех видов". (Ссылка: Lim, J. S., Chang, S., Maynard, S., & Ahmad, A. (2009). Exploring the Relationship between Organizational Culture and Information Security Culture.)

Техники

Все методы социальной инженерии основаны на когнитивных предубеждениях, когда люди принимают решения на основе известных или предполагаемых атрибутов. Когнитивные предубеждения используются в различных комбинациях для создания методов атаки, например, для кражи личной информации сотрудника. Телефонные звонки и разговоры являются наиболее распространенным видом социальной инженерии. Другим примером атаки с помощью социальной инженерии может быть человек, выдающий себя за поставщика услуг, например, дезинсекторов, техников или инспекторов по безопасности, чтобы получить доступ к людям или информационным системам.

Техника может включать в себя завоевание доверия через общение в социальных сетях, что в конечном итоге приводит к запросу банковской информации или ценных паролей.

Другим примером может быть использование авторитетного, доверенного источника информации, например, доски объявлений компании, чтобы изменить номер службы поддержки на ложный номер для сбора информации. Когда сотрудник читает размещенную на доске объявлений, но фальшивую информацию и звонит по указанному номеру, он охотнее передает пароли или другие учетные данные, поскольку верит, что сообщение на доске настоящее.

Социальная инженерия в значительной степени опирается на шесть принципов влияния, сформулированных Робертом Чалдини в его книге "Влияние".

Шесть ключевых принципов влияния на человека

- **Взаимность** — наши социальные нормы подразумевают, что мы склонны возвращать услугу, когда нас об этом просят. Вы можете представить, как это хорошо работает в телефонных разговорах для получения информации.
- **Обязательства и последовательность** — люди берут на себя обязательства, будь то при личной встрече, в письменном виде или на веб-сайте, они с большей вероятностью будут выполнять их, чтобы сохранить свое представление о себе. Примером может служить флажок на всплывающем окне веб-сайта, гласящий: "Я зарегистрируюсь позже". Когда кто-то ставит галочку, даже если первоначальный стимул исчезает, он склонен соблюдать этот социальный контракт или идти вперед и регистрироваться, чтобы не чувствовать себя нечестным.
- **Социальное подтверждение** —люди видят, что кто-то другой что-то делает, например, смотрит вверх, другие останавливаются, чтобы сделать то же самое. Этот тип конформизма позволяет людям манипулировать другими людьми, заставляя их делиться информацией, которую они обычно не разглашают.
- **Авторитет** — люди почти всегда подчиняются авторитетным лицам, даже если их просят выполнить опасную или вредную для другого человека задачу. Этот принцип авторитета означает, что

злоумышленники, которые кажутся авторитетными или представляющими авторитетную фигуру, с большей вероятностью получат доступ.

- **Симпатия** — симпатичные люди способны убеждать других более эффективно. Этот принцип хорошо известен специалистам по продажам. Люди легко убеждаются знакомыми людьми, которые им нравятся.
- **Дефицит** — Когда люди верят или воспринимают, что что-то ограничено в количестве или доступно только в течение короткого времени, они будут действовать позитивно и быстро, чтобы заполучить желаемый товар или предложение.

Существует четыре вектора социальной инженерии, или направления атак, которые могут использовать эти принципы влияния.

- **Фишинг** означает, что человек обманным путем получает информацию, особенно через запросы финансовой информации. Часто попытки выглядят как настоящий веб-сайт или электронное письмо, но вместо этого ссылаются на сайт коллекторов.
- **Вишинг** означает "голосовой фишинг", поэтому он связан с голосовыми телефонными звонками с целью сбора частной личной информации для получения финансовой выгоды.
- **Смишинг** подразумевает использование текстовых сообщений SMS для срочности и просьбы выполнить определенное действие, например, перейти по поддельной ссылке или отправить информацию о счете.
- **Самозванство** включает в себя сценарии с личным присутствием, например, ношение униформы поставщика услуг для получения внутреннего доступа к зданию или системе.

Стойкость пароля

Стойкость пароля — мера эффективности пароля в противостоянии атакам взлома пароля. Стойкость пароля определяется:

Длина — количество символов, которые содержит пароль.

Сложность — означает, что в нем используется комбинация букв, цифр и символов.

Непредсказуемость — может ли злоумышленник легко его угадать?

Вот практический пример трех паролей (Рисунки 80-82):

student

Very Weak

7 characters containing: Lower case Upper case Numbers Symbols

Time to crack your password:
0.07 seconds

Review: Oh dear, using that password is like leaving your front door wide open. Your password is very weak because it is a dictionary word.

Рисунок 80 — Слабый пароль

student-zss

Medium

11 characters containing: Lower case Upper case Numbers Symbols

Time to crack your password:
6 hours

Review: Hmm, using that password is like locking your front door, but leaving the key under the mat. Your password is of medium strength because it contains a dictionary word.

Рисунок 81 — Пароль, используемый для гостевого входа

St4d3nt_Z$$

Very Strong

11 characters containing: Lower case Upper case Numbers Symbols

Time to crack your password:
4 thousand years

Review: Fantastic, using that password makes you as secure as Fort Knox.

Рисунок 82 — Пароль, который лучше менять раз в 3 месяца.

Средства проверки и подтверждения надежности паролей

Примером средства проверки надежности пароля является менеджер паролей. Этот тип инструмента - лучший способ убедиться, что вы используете надежный пароль. Как правило, инструмент проверки надежности пароля встроен в систему паролей в качестве входного валидатора, чтобы убедиться, что пароль пользователя совместим с последними рекомендациями по управлению идентификацией.

Лучшие практики

Существует несколько лучших практик для обеспечения безопасности попыток входа пользователей в систему. К ним относятся уведомление пользователей о подозрительном поведении и ограничение количества попыток ввода пароля и имени пользователя.

Руководство NIST по цифровой идентификации

Вот краткое содержание Руководства по цифровой идентификации NIST 800-63B:

1. Минимум 8 символов при установке человеком
2. Минимум 6 символов при установке системой/службой
3. Поддерживать максимальную длину не менее 64 символов
4. Должны поддерживаться все символы ASCII (включая пробел)
5. Усечение секрета (пароля) не должно выполняться при обработке
6. Проверять выбранный пароль по известным словарям паролей
7. Разрешать не менее 10 попыток ввода пароля до блокировки
8. Отсутствие требований к сложности
9. Отсутствие срока действия пароля
10. Никаких подсказок пароля
11. Никакой аутентификации на основе знаний, например, кто был вашим лучшим другом в школе?
12. Никаких SMS для двухфакторной аутентификации, вместо этого используйте одноразовый пароль из такого приложения, как Google Authenticator.

Резюме

Анализ вариантов развертывания с использованием различных моделей

Как правило, крупные организации используют четырехуровневую структуру: разработка, тестирование, staging и production. В первые дни существования компьютеров существовал только один способ развертывания программного обеспечения: вы просто устанавливали его на сам компьютер. Эта модель известна как "bare metal". Теперь у вас есть другие варианты, включая виртуальные машины, контейнеры и бессерверные вычисления. Под "локальным" понимается любая система, которая находится буквально в пределах вашего здания. Скорее всего, речь идет о традиционных центрах обработки данных, где размещаются отдельные машины, которые предоставляются для приложений, а не об облаках, внешних или иных. Облака предоставляют самообслуживаемый доступ к вычислительным ресурсам,

таким как виртуальные машины, контейнеры и даже "чистый металл". Преимущество частного облака заключается в том, что вы полностью контролируете место его расположения. Публичное облако - это, по сути, то же самое, что и частное облако, но управляется оно провайдером публичного облака. Клиенты публичного облака могут использовать ресурсы совместно с другими организациями. Гибридное облако - это комбинация двух различных типов облаков. Как правило, гибридное облако используется для объединения частного и публичного облака в рамках одного приложения. Граничное облако перемещает вычисления ближе к месту, где они необходимы. Вместо того чтобы транзакции проходили путь от конечного пользователя в Кливленде до основного облака в Орегоне, в Кливленде может быть промежуточное облако, пограничное облако. Краевое облако обрабатывает данные или транзакцию. Затем оно либо отправляет ответ обратно клиенту, либо проводит предварительный анализ данных и отправляет результаты в региональное облако, которое может находиться дальше.

Создание и развертывание приложения

Контейнер — способ инкапсуляции всего необходимого для запуска приложения, чтобы его можно было легко развернуть в различных средах. Docker — способ создания и запуска контейнера. “makefile” — файл, который утилита make использует для компиляции и сборки всех частей приложения. В Docker это простой текстовый файл, называемый Dockerfile. Он определяет шаги, которые команда сборки docker должна выполнить для создания образа, который затем может быть использован для создания целевого контейнера. Вот некоторые из доступных команд Dockerfile: `FROM`, `MAINTAINER`, `RUN`, `CMD`, `EXPOSE`, `ENV`, `COPY`, `ENTRYPOINT`, `VOLUME`, `USER`, `WORKDIR`, `ARG`, `ONBUILD`, `STOPSIGNAL` и `LABEL`. Вы можете использовать свой образ для создания нового контейнера и выполнения некоторой работы. Для этого нужно запустить образ:

```
$ docker run -d -P sample-app-image
1688a2c34c9e7725c38e3d9262117f1124f54685841e97c3c5225af88e30bfc5
```

В данном случае вы указываете несколько параметров. Параметр -d является сокращением от --detach и говорит, что вы хотите запустить его в фоновом режиме, а -P указывает Docker опубликовать его на портах, которые вы открыли (в данном случае 8080).

Вы можете сделать свой образ доступным для использования другими людьми, сохранив его в реестре образов. По умолчанию Docker использует реестр Docker Hub, хотя вы можете создать и использовать свой собственный реестр.

Среда разработки должна быть удобной для разработчика; она должна соответствовать производственной среде только там, где это уместно. Типичная среда разработки может состоять из любого количества инструментов, от IDE до баз данных и объектных хранилищ. Вы создали и

запустили пример приложения в практической лаборатории, и вы можете настроить его так, чтобы запустить пример приложения локально.

Continuous Integration/Continuous Deployment (CI/CD)

CI/CD — это философия развертывания программного обеспечения, которая занимает видное место в области DevOps. Сам DevOps — это коммуникация и уверенность в том, что все члены команды работают вместе для обеспечения бесперебойной работы. Идея непрерывной интеграции заключается в том, что вы и все остальные разработчики проекта постоянно объединяете свои изменения с основной ветвью существующего приложения. Это означает, что любой набор изменений невелик, а вероятность возникновения проблем низка. Если все используют основную ветвь, то каждый, кто проверяет код, будет иметь последнюю версию того, что разрабатывают остальные. Вот некоторые преимущества, которые дает использование CI/CD для разработки:

- Интеграция с методологиями agile
- Сокращение среднего времени решения проблемы (MTTR)
- Автоматизированное развертывание
- Выпуск функций с меньшими сбоями
- Улучшенное качество
- Улучшенное время выхода на рынок

Конвейер развертывания можно создать с помощью инструмента сборки, например, Jenkins. Эти конвейеры могут решать такие задачи, как сбор и компиляция исходного кода, тестирование и компиляция артефактов, таких как tar-файлы или другие пакеты. Основной единицей Jenkins является проект, также известный как “job”. Вы можете создавать job-ы, которые выполняют самые разные задачи: от получения кода из репозитория управления исходным кодом, например, GitHub, до создания приложения с помощью скрипта или инструмента сборки, его упаковки и запуска на сервере.

Сети для разработки приложений и безопасности

В наши дни необходимо учитывать сетевое взаимодействие во всех, кроме самых простых, случаях использования. Это особенно верно, когда речь идет об облачных и контейнерных развертываниях. Некоторые из приложений, которые необходимо учитывать при развертывании облака, включают: Брандмауэры, балансировщики нагрузки, DNS и обратные прокси. На самом базовом уровне брандмауэр принимает или отклоняет пакеты на основе IP-адресов и портов, к которым они адресованы. Балансировщик нагрузки принимает запросы и балансирует их, распределяя между несколькими серверами. DNS — это способ, с помощью которого серверы в Интернете переводят человекочитаемые имена в IP-адреса с возможностью машинной маршрутизации. IP-адреса необходимы для навигации в Интернете. Обратный прокси похож на обычный прокси; однако если обычный прокси работает для того, чтобы запросы с нескольких компьютеров выглядели так,

будто все они исходят от одного клиента, то обратный прокси работает для того, чтобы ответы выглядели так, будто все они исходят от одного сервера.

Безопасность приложений

Вы обязаны защитить данные, до момента работы над ними. Существует два метода шифрования данных: одностороннее шифрование и двустороннее шифрование. Данные также уязвимы при передаче. Когда ваши данные находятся в движении, они уязвимы для атак типа "человек посередине", при которых сервер по пути следования может наблюдать, украсть и даже изменить данные. Чтобы предотвратить это, вы можете использовать несколько методов, включая: SSH, TLS и VPN. SQL-инъекция — это метод инъекции кода, используемый для атаки на приложения, управляемые данными, при котором вредоносные SQL-запросы вставляются в поле ввода для выполнения. SQL-инъекция должна использовать уязвимость в программном обеспечении приложения. Уязвимость SQL-инъекции существует потому, что некоторые разработчики не заботятся о проверке данных и безопасности. Существуют инструменты, которые могут помочь обнаружить недостатки и проанализировать код. OWASP специализируется на предоставлении образования, инструментов и других ресурсов, чтобы помочь разработчикам избежать некоторых наиболее распространенных проблем безопасности в веб-приложениях. Ресурсы, предоставляемые OWASP, включают: инструменты, проекты кода и проекты документации.

XSS-атаки происходят, когда контент, переданный пользователем и не прошедший санитарную обработку, отображается другим пользователям. Наиболее очевидный вариант такой атаки — когда один пользователь отправляет комментарий, содержащий скрипт, выполняющий вредоносное действие, и у каждого, кто просматривает страницу комментариев, этот скрипт выполняется на его машине. Другой тип атаки, который имеет некоторые общие черты с XSS-атаками, — это CSRF. В обоих случаях злоумышленник намеревается, чтобы пользователь выполнил код злоумышленника, обычно даже не подозревая об этом. Разница в том, что CSRF-атаки обычно направлены не на целевой сайт, а на другой сайт, на котором пользователь уже прошел аутентификацию.

Первые пароли были простым открытым текстом, хранившимся в базах данных. Более надежным способом хранения пароля является его преобразование в данные, которые невозможно преобразовать обратно в исходный пароль, что называется хэшированием. Чтобы гарантировать уникальность паролей, повысить их сложность и предотвратить атаки на пароли даже при одинаковых входных данных, на вход хэш-функции добавляется соль (которая представляет собой просто случайные данные). 2FA использует ту же комбинацию пароль/имя пользователя, но с добавлением просьбы проверить, кем является человек, используя то, что принадлежит только ему, например, мобильное устройство. При использовании MFA пользователю предоставляется доступ только после успешного предъявления нескольких отдельных доказательств механизму аутентификации. Обычно для

MFA требуется как минимум две из следующих категорий: знание (то, что пользователь знает), владение (то, что он имеет) и неотъемлемость (то, чем он является).

Угадывание пароля — это онлайновая техника, предполагающая попытку аутентификации конкретного пользователя в системе. В криптоанализе и компьютерной безопасности атака по словарю - это форма атаки грубой силы для поражения шифра или механизма аутентификации путем попытки определить ключ расшифровки или парольную фразу путем перебора сотен, а иногда и миллионов вероятных вариантов. Предварительное вычисление списка хэшей словарных слов и хранение их в базе данных с использованием хэша в качестве ключа требует значительного времени на подготовку, но позволяет быстрее выполнить реальную атаку. Социальная инженерия для взлома паролей предполагает, что человек убеждает или обманывает другого человека для получения доступа. Стойкость пароля — это мера эффективности пароля в противостоянии атакам взлома пароля. Стойкость пароля определяется: длиной, сложностью и непредсказуемостью.

Дополненная Глава 5

Вы узнали об автоматизации масштабирования. Теперь пришло время научиться автоматизировать инфраструктуру сетей. Это серьезное изменение в том, как специалисты думают о сетях, от проектирования до создания и управления.

Конечно, существует множество инструментов для автоматизации многих задач, а также специализированные инструменты для моделирования сети для тестирования. Но еще более ценным для сетевых инженеров является скорость с точностью и безопасными возможностями самообслуживания. Puppet, Ansible и Chef отлично подходят для этих целей. Кроме того, pyATS и VIRL — это инструменты, созданные специально для работы с сетью.

Автоматизация инфраструктуры

Программно-определяемая инфраструктура: Случай для автоматизации

Программно-определяемая инфраструктура, также известная как облачные вычисления, позволяет разработчикам и операторам использовать программное обеспечение для запроса, конфигурирования, развертывания и управления пустым и виртуализированным вычислительным оборудованием, хранилищем и сетевыми ресурсами.

- Облачные вычисления также позволяют использовать более абстрактные платформы и сервисы, такие как база данных как сервис (DaaS), платформа как сервис (PaaS), бессерверные вычисления, оркестровка контейнеров и многое другое.
- Частные облака позволяют предприятиям гораздо эффективнее использовать дорогостоящее местное оборудование.

– Публичные и размещаемые частные облака позволяют компаниям арендовать мощности по мере необходимости, что позволяет им быстрее перемещаться и расти (или сокращаться), упрощая планирование и избегая инвестиций в основной капитал.

Преимущества облачных парадигм

– Самообслуживание (платформы по требованию) — облачные ресурсы могут быть доступны в течение нескольких часов или минут с момента возникновения потребности в них. Это ускоряет все этапы разработки и позволяет быстро масштабировать производственные мощности. Приложения могут быть масштабированы в регионе публичного облака, наборе услуг или у провайдера, который является наиболее экономически эффективным.
– Четкая спецификация, согласованность, повторяемость — разработчики могут фиксировать и стандартизировать уникальные конфигурации, поддерживая согласованность конфигурации платформ в процессе разработки, тестирования, постановки и производства. Развертывание заведомо хорошего приложения и конфигурации предотвращает ошибки, которые могут возникнуть при ручном изменении конфигурации платформы.
– Абстракция платформы — контейнерные технологии абстрагируют приложения и платформы друг от друга, инкапсулируя зависимости приложений и позволяя вашему контейнерному приложению работать в общем определенном окружении хоста.

Вызовы облачных парадигм

Разработчики должны уделять пристальное внимание дизайну, архитектуре и безопасности платформы. Облачные среды предъявляют новые требования к приложениям. Общедоступные и частные облачные платформы имеют различные пользовательские интерфейсы, API и причуды. Это означает, что пользователи не всегда могут относиться к облачным ресурсам как к товарам, которыми они должны быть на самом деле, особенно при попытке управлять облаками вручную.

Контроль доступа имеет решающее значение, поскольку пользователи облака с неправильными разрешениями могут нанести большой ущерб активам организации. Управление облачными разрешениями также может быть сложным, особенно в сценариях с ручным управлением.

Когда облачные ресурсы могут быстро самообслуживаться с помощью ручных операций, потреблением может быть трудно управлять, а затраты трудно подсчитать. Частные облака требуют частого аудита и процедур для вывода из эксплуатации неиспользуемой виртуальной инфраструктуры. Пользователи публичного облака могут быть удивлены неожиданными расходами, когда платные ресурсы оставляют, но не сносят.

Многие из этих проблем можно решить с помощью автоматизации.

Распределенные и динамические приложения: Еще один случай для автоматизации

Крупномасштабные корпоративные и общественные приложения могут нуждаться в управлении большими и переменными нагрузками трафика, вычислений и хранения данных.

- Приложения должны обеспечивать хороший опыт для пользователей.
- Они должны быть устойчивыми, высокодоступными, обеспечивать целостность и безопасность пользовательских данных, а также соответствовать местным нормативным требованиям в отношении места и способа хранения данных.
- Они должны эффективно расти (и сокращаться), чтобы соответствовать потребностям бизнеса, использовать тенденции и работать экономически эффективно.

Односерверные, "монолитные" архитектуры приложений, несмотря на концептуальную простоту, не очень хорошо отвечают этим потребностям. Один сервер является единой точкой отказа, ограничивает производительность и емкость, а также имеет ограниченные возможности обновления. Дублирование единственного сервера может увеличить емкость для очень простых приложений, но не подходит для приложений, требующих согласованности данных во всех экземплярах. И это не защитит данные пользователей, если произойдет сбой на сервере, на котором находятся их данные.

По этим и другим причинам современные архитектуры приложений становятся все более распределенными. Они строятся из небольших и относительно легких компонентов, которые называют микросервисами. Эти компоненты могут быть изолированы в контейнерах, соединены с помощью служб обнаружения и обмена сообщениями (которые абстрагируют сетевое взаимодействие) и поддерживаются устойчивыми, масштабируемыми базами данных (которые поддерживают состояние).

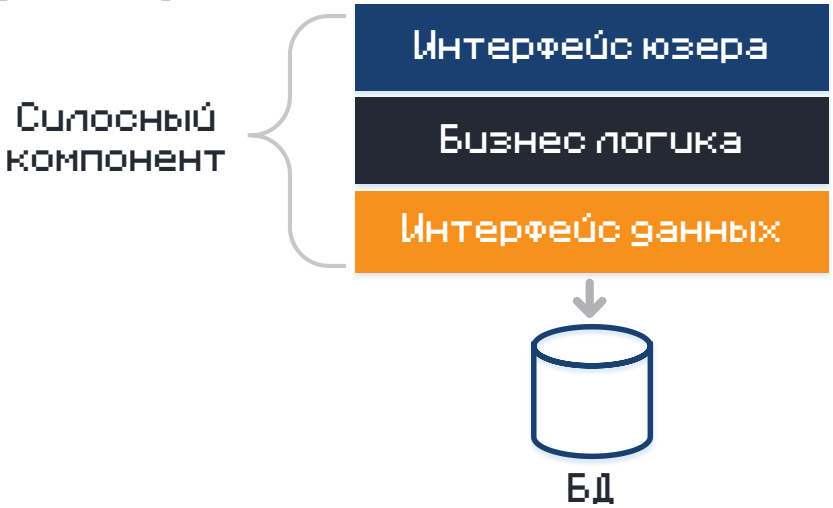


Рисунок 83 — Компонент

Монолитные приложения можно масштабировать только путем дублирования всего приложения на дополнительных машинах.

Приложения на базе микросервисов теоретически могут масштабироваться от сервиса к сервису по мере необходимости для удовлетворения требований к трафику или производительности. Такая архитектура помогает получить максимальную выгоду от гиперконвергентной инфраструктуры, которая позволяет точно настроить вычислительные системы, системы хранения данных, сетевые подключения и сервисы в соответствии с динамическими требованиями к нагрузке приложений.

Микросервисные приложения

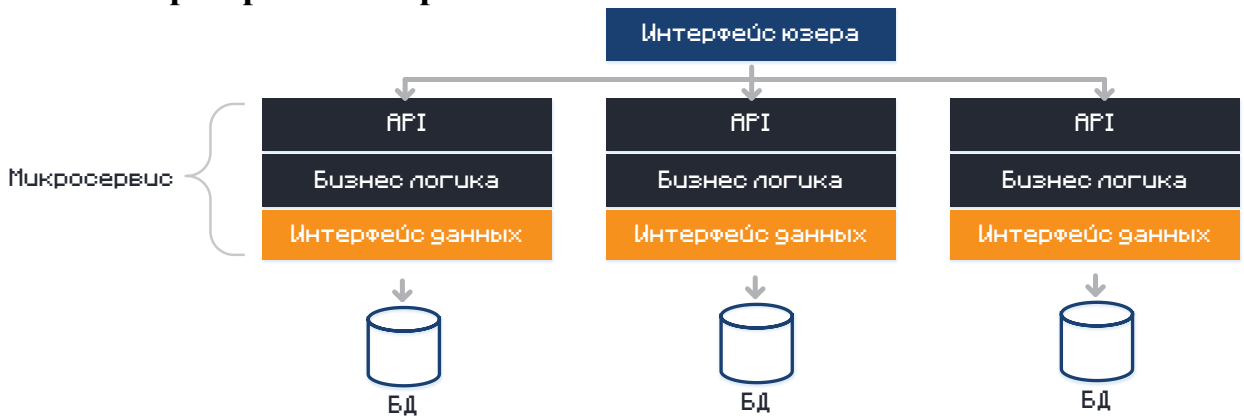


Рисунок 84 — Микросервис

Преимущества микросервисов

- **Масштабируемость** — Микросервисы с легкостью можно масштабировать и распределять между ними нагрузку по мере необходимости на многих сетевых серверах или даже в нескольких географически разделенных центрах обработки данных или регионах публичного облака. Это устраняет единые точки отказа и предоставляет столько мощностей для каждого конкретного типа задач, сколько требуют условия, везде, где это необходимо, устраняя перегрузки.
- **Средства автоматизации инфраструктуры** — динамичность приложений на базе микросервисов все чаще обеспечивается инфраструктурой. Такие контейнерные "оркестры", как Kubernetes или Mesos, автоматизируют масштабирование по требованию, самовосстановление и многое другое.

Вызовы микросервисов

- **Повышенная сложность** — Микросервисы подразумевают наличие множества движущихся частей, которые необходимо настроить и развернуть. Возникают более сложные операции, включая масштабирование по требованию, самовосстановление и другие функции.

– **Автоматизация является обязательным требованием** — Ручные методы не могут реально справиться со сложностью развертывания и управления динамическими приложениями и их платформами-оркестрами, с их высокоскоростными, автономными операциями и их преходящими и эфемерными кусочками.

Эти бизнесовые и технические потребности, тенденции и динамика побуждают разработчиков и администраторов повсеместно использовать автоматизацию для решения следующих задач:

– Управление всеми этапами создания, настройки, развертывания и управления жизненным циклом приложений. Это включает разработку, тестирование, staging и production.
– Управление программно-определяемыми инфраструктурами от имени создаваемых приложений.
– Вместе с вашими приложениями сохранять, обновлять и постоянно совершенствовать код автоматизации. Этот код помогает вам разрабатывать, тестировать, stage-ить, контролировать и эксплуатировать наши приложения в production масштабах и в различных средах. Вы всё чаще можете рассматривать весь этот код как единый рабочий продукт.

DevOps и SRE

Знакомство с DevOps и SRE

Чтобы full-stack автоматизация была действительно эффективной, она требует изменений в организационной культуре, включая разрушение исторически сложившихся разделений между разработкой (Dev) и операциями (Ops).

В этой теме мы рассмотрим историю подхода DevOps и основные принципы успешных организаций DevOps.

Разделение DevOps

Исторически сложилось так, что созданием приложений занимались разработчики программного обеспечения (Dev), а обеспечение работы приложений для пользователей и бизнеса было уделом ИТ-операторов (Ops) (Таблица 10).

Таблица 10 — Задачи Developers и Operators

Характеристика	Dev	Ops
Заботится о:	Приложениях по индивидуальному заказу и их работа	Приложениях и то, как они работают, плюс инфраструктура (исторически, аппаратное обеспечение), ОС, сеть, товарные сервисы, доступность, безопасность, масштабирование и обслуживание.
К бизнесу относятся как:	Центр прибыли: требует ресурсов	Центр затрат: предоставляет и учитывает ресурсы
Участвует в дежурствах:	Время от времени (и нарушается только тогда,	Регулярно

	когда проблемы передаются в отдел разработки)	
Производительность измеряется:	Абстрактно (включая плохие метрики, такие как строки кода/день)	Конкретно (соблюдение SLA, решение проблем)
Необходимые навыки:	Скорее глубокий, чем широкий: Языки, API, архитектура, инструменты, процесс, "фронтенд", "бэкенд" и т.д.	Скорее широкий, чем глубокий: Конфигурирование, администрирование, ОС, специфические для производителя административные CLI/API, shell, автоматизация, БД, безопасность и т.д.
Необходимое проворство:	Многое! Двигайтесь быстро, внедряйте инновации, ломайте вещи, чините потом.	Не так много! Инвестиции должны быть всесторонне обоснованы, ожидания оправданы. Длительные сроки — это нормально. Безопаснее сказать "нет", чем "да".

В традиционной экосистеме корпоративных ИТ до виртуализации разделение Dev и Ops казалось разумным. До тех пор, пока инфраструктура основывалась на инвестициях в капитальное оборудование (вручную устанавливаемое, предоставляемое, конфигурируемое и управляемое оборудование), она нуждалась в привратниках. Привратники Ops работали в иные временные рамки, чем пользователи и разработчики, и имели другие стимулы и проблемы.

Узкие места в наследстве

Традиционно обеспечение ресурсами проекта или масштабирование системы определяется планом, а не спросом. Запрос, покупка, установка, обеспечение и настройка серверов или сетевых мощностей и услуг для проекта могли занимать месяцы. При ограниченных физических ресурсах часто использовалось совместное использование ресурсов.

Отсутствие простых способов создания и разрушения изолированных сред и подключения означало, что организации склонны создавать долго работающие системы, которые становятся едиными точками отказа. Сети смешанного использования было сложно защитить, и они требовали тщательного постоянного управления.

В просторечии такие долго работающие, сложные системы называли "домашними животными", потому что люди давали им имена и заботились об этой единственной системе. Альтернативой является "скот", который представляет собой урезанные, эфемерные рабочие нагрузки и виртуальную инфраструктуру, созданную и разрушенную с помощью автоматизации. Этот метод гарантирует, что найдется новая система (или "корова"), которая возьмет на себя работу.

Слияние Dev и Ops

Сложные технологические организации уже несколько поколений мигрируют от этих исторических крайностей. Процесс ускорился с повсеместным внедрением виртуализации серверов, облачных технологий и Agile-разработки программного обеспечения. В начале 2000-х годов началось движение за то, чтобы рассматривать Dev и Ops как единое целое:

- Сделать разработчиков ответственными за развертывание и сопровождение
- Рассматривать виртуализированную инфраструктуру как код

Ключевые принципы DevOps

DevOps/SRE имеют множество основных принципов и лучших практик:

- Фокус на автоматизации
- Идея о том, что "отказ - это нормально"
- Переосмысление понятия "доступность" с точки зрения того, что может терпеть бизнес

Подобно тому, как Agile Development можно рассматривать как метод определения и контроля ожиданий руководства для проектов программного обеспечения, DevOps можно рассматривать как способ структурирования здоровой рабочей культуры для технологических частей бизнеса. Его также можно рассматривать как способ снижения затрат. Сделав сбои нормальным явлением и автоматизировав их устранение, можно перевести работу с дорогостоящих и стрессовых часов по вызову на запланированный рабочий день.

Автоматизация, отказ от труда и удержание талантов

Автоматизация обеспечивает скорость и повторяемость при устранении ручного, повторяющегося труда. Это позволяет техническим специалистам тратить время на решение новых проблем, увеличивая ценность для бизнеса за потраченное время.

От специалистов DevOps/SRE часто ожидают, что они будут посвящать значительную часть рабочего времени (50% и более в некоторых сценариях) предоставлению новых операционных возможностей и инженерному обеспечению надежного масштабирования, включая разработку инструментов автоматизации. Это сокращает время, затрачиваемое на вызовы и ручное вмешательство для понимания и устранения проблем.

Приобретение и удержание технических талантов требует от организаций сотрудничества с инноваторами, чтобы минимизировать скуку и стресс от малоценного труда, и работы по вызову, а также риск столкнуться с неизбежными технологическими сбоями в режиме "пожарной тревоги".

Это особенно важно для предприятий с изначально низкой маржой, которые получают прибыль за счет быстрого роста и больших масштабов и должны строго контролировать расходы, особенно на квалифицированный персонал. Однако для всех типов организаций важно, чтобы работа в области информационных технологий воспринималась как центр прибыли, а не как центр затрат.

Сбои — это нормально

Предположение о том, что сбои будут происходить, влияет на методологию разработки программного обеспечения. DevOps должен создавать продукты и платформы с большей отказоустойчивостью, более высокой задержкой, где

это возможно, лучшим самоконтролем, протоколированием, передачей сообщений об ошибках конечному пользователю и самовосстановлением.

Когда сбои всё же происходят и DevOps приходится вмешиваться, возникающие при этом действия следует рассматривать не просто как ремонтные работы, а как исследование для выявления и ранжирования процедурных кандидатов для новых раундов автоматизации.

SLO, SLI и бюджеты ошибок

Критически важными для культуры DevOps/SRE являются две взаимосвязанные идеи:

1. DevOps должен приносить измеримую, согласованную бизнес-ценность;
2. Статистическая реальность не позволяет сделать это идеально.

Эти идеи кодифицируются в цели уровня обслуживания (SLO), которые определяются в терминах реальных показателей, называемых индикаторами уровня обслуживания (SLI).

SLI разрабатываются с учетом практической реальности предоставления услуг клиентам: они могут представлять собой единый порог или предусматривать более сложные скобки для дальнейшей классификации отклоняющихся результатов. Например, в SLI может быть указано, что 99% запросов будут обработаны в течение 50 миллисекунд, а также может потребоваться сбор такой информации, как завершение одного запроса длительностью >50 мс или неудача конкретного запроса для вашего крупнейшего клиента.

Методология SLO/SLI позволяет дешевле и быстрее предоставлять бизнес-ценность, устраняя обязательство стремиться к совершенству в пользу создания того, что "достаточно хорошо". Она также может влиять на темпы, объем и другие аспекты разработки для обеспечения и улучшения адекватности.

Один из способов моделирования результатов SLO/SLI предполагает установление так называемого "бюджета ошибок" для сервиса на определенный период времени (день, неделя, месяц, квартал и т.д.), а затем вычитание из этого значения количества неудач в достижении SLO. Если бюджет ошибок превышен, можно принять разумные решения, например, замедлить темпы выпуска релизов до тех пор, пока не будут определены источники ошибок, сделаны и протестированы конкретные исправления.

Рисунок 85 — SLA против SLO и бюджет на ошибки

Для данной услуги имеет смысл взять на себя обязательство предоставлять услуги в пределах возможностей, но затем перевыполнить его. SLA (внешние соглашения) лучше всего устанавливать там, где их будет легко выполнить. SLO (целевые показатели фактической производительности) могут быть установлены выше. Бюджет ошибок — это разница между SLO и 100% доступностью.

Основные сценарии автоматизации

Введение в базовые сценарии автоматизации

Мощные инструменты автоматизации, такие как Ansible, Puppet и Chef, привносят в работу DevOps простоту использования, предсказуемость, дисциплину и способность работать в масштабе. Но это не означает, что вы не можете выполнять автоматизацию с помощью более простых инструментов, таких как Bash и Python. Инструменты автоматизации частично работают за счет обертывания функциональности оболочки, утилит операционной системы, функций API и других элементов плоскости управления для простоты, единообразия, расширения возможностей и совместимости в сценариях DevOps. Но инструменты все равно не решают всех проблем развертывания и конфигурирования.

Поэтому в каждом инструменте автоматизации есть одна или несколько функций, которые выполняют основные команды и сценарии на объектах и возвращают результаты. Например, в Ansible к таким функциям относятся command, shell и raw.

Иногда использование команд оболочки или сценариев может быть быстрее и проще. Часто это связано с тем, что многие реализации инструментов начинаются с перевода автоматизации, изначально написанной на Bash, Python или других языках, и вы хотите быстро и точно перенести эту функциональность в инструмент перед переносом и рефакторингом.

В общем, редко кто заглядывает в репозитории, поддерживаемые инструментами infra-as-code, и не находит там скриптов. Поэтому наличие этих навыков очень важно!

Основные инструменты для автоматизации сценариев

Оболочки распространены повсеместно, поэтому скрипты исторически являются основой автоматизации.

Bash

В Linux (и других операционных системах) оболочка взаимодействует с интерактивным вводом-выводом, файловой системой и межпроцессным взаимодействием. Это дает возможность отдавать команды, предоставлять входные данные для обработки и направлять выходные данные в цепочки мощных утилит.

Оболочка Bourne Again Shell (BASH) используется по умолчанию в большинстве дистрибутивов Linux. Из-за ее повсеместного распространения термины "Bash" и "оболочка" обычно используются как взаимозаменяемые.
Использование команд в сценарии Bash во многом аналогично их использованию непосредственно из командной строки. Разработка очень простых сценариев может сводиться к копированию выражений командной строки в файл после тестирования команд CLI на предмет их работоспособности.

В отличие от этого, Python или другой высокоуровневый, сложный язык для простой процедурной автоматизации обычно более сложен, и, возможно, его не стоит использовать для простых проектов.

Языки программирования помимо Bash

Сложные языки улучшают Bash при увеличении сложности и требований к масштабированию. Они особенно полезны при создании и эксплуатации виртуализированной инфраструктуры в облачных средах с использованием таких SDK, как AWS SDK для Python или AWS SDK для javascript в Node.js. Хотя Bash можно использовать для создания сценариев доступа к AWS CLI, вы можете использовать встроенные функции и библиотеки более сложных языков для разбора сложных возвращаемых наборов данных (таких как JSON), управления множеством параллельных операций, обработки ошибок, обработки асинхронных ответов на команды и многого другого.

Для разработки и выполнения сценариев на нужном вам языке может потребоваться установка и настройка этого языка на вашей системе разработки и на всех удаленных целевых машинах. Для доступа к утилитам системного уровня может потребоваться обращение к библиотекам (таким как библиотека os в Python), а затем обертывание команд Bash CLI в дополнительный синтаксис для выполнения. Вам также необходимо обрабатывать коды возврата, тайм-ауты и другие условия в предпочитаемой языковой среде.

Процедурная автоматизация

Использование Bash, Python или других языков для автоматизации обычно означает написание императивной процедуры. Императивная процедура — это упорядоченная последовательность команд, направленная на достижение цели. Последовательность может включать управление потоком, условия, функциональную структуру, классы и многое другое.

Такая процедурная автоматизация может быть очень мощной. Но она остается простой, только если вы знаете, как работают системные утилиты, CLIs/SDK и другие интерфейсы. Вы также должны знать о состоянии целевой системы.

Разработка процедуры

Как вы знаете, если вы создадите небольшой скрипт для установки и настройки части программного обеспечения на удаленной целевой системе, он

может сработать нормально в первый раз. Однако запустите его во второй раз, и ваш простой скрипт может испортиться. Он может выдать ошибку и остановиться, когда обнаружит, что приложение уже установлено, или, что ещё хуже, проигнорировать такую ошибку и продолжить вносить лишние изменения в конфигурационные файлы.

Чтобы сделать этот скрипт более безопасным, простым в использовании, более гибким и многоразовым, необходимо сделать его более умным и продуманным. Например, вы можете усовершенствовать его следующим образом:

- Определите, работает ли он в среде `Debian` или `CentOS`, и используйте правильный менеджер пакетов (`apt` или `yum`) и синтаксис.
- Определите, установлено ли уже ваше целевое приложение в соответствующей версии, и попробуйте установить его, только если его нет, остановившись в противном случае и не внося дальнейших изменений.
- Определите, сделал ли он копию каждого файла конфигурации, прежде чем изменять его, и используйте потоковые редакторы (`awk`, `sed` и т.д.) для внесения точных изменений, а не бездумно добавляйте текст в файлы конфигурации и надейтесь, что приложения, которые потребляют эти файлы, не сломаются.

По мере дальнейшего развития и совершенствования скриптов вам захочется, чтобы они выполняли некоторые из следующих задач:

- Обнаружение, инвентаризация и сбор информации о целевых системах, а также обеспечение того, чтобы скрипты делали это по умолчанию.
- Инкапсуляция сложности безопасной установки приложений. Создавать резервные копии и изменения конфигурационных файлов и перезапускать службы в многократно используемые формы, такие как дочерние скрипты, содержащие параметры, библиотеки функций и другую информацию.

Чтобы обеспечить эффективность сценариев и возможность их многократного использования, вы будете (Рисунок 86):

- Стандартизируйте порядок и представление параметров, флагов и ошибок.
- Создайте иерархию кода, которая логично и эффективно разделяет задачи.
- Создавайте высокоуровневые скрипты для всего развертывания и низкоуровневые для этапов развертывания.
- Отделяйте данные, специфичные для развертывания, от кода, делая код как можно более общим и пригодным для повторного использования.

Рисунок 86 — Пример простого процедурного сценария

Этот тип скриптов, как правило, опасен, если начальное состояние не полностью известно и не контролируется. Повторное применение тех же изменений к правильно настроенной системе может даже сломать ее.

Идемпотентность: повторяющаяся тема в автоматизации

В конечном счете, целью практически любого сценария является достижение желаемого состояния системы, независимо от начальных условий. Тщательно написанные процедурные скрипты и декларативные инструменты конфигурирования изучают цели перед выполнением задач на них, выполняя только те задачи, которые необходимы для достижения желаемого состояния.

Это качество программного обеспечения называется идемпотентностью. Существует несколько основных принципов идемпотентности, которым необходимо следовать (Рисунок 87):

- Убедитесь, что изменения, которые вы хотите внести, еще не были сделаны — Также известно, как "Сначала не навреди". Ничего не делать — почти всегда лучший выбор, чем сделать что-то неправильное и, возможно, невосстановимое.
- По возможности, перед внесением изменений перейдите к заведомо хорошему состоянию — Например, вам может понадобиться удалить и очистить ранние версии приложений перед установкой более поздних версий. В производственных средах infra-as-code этот принцип становится основой неизменяемости. Неизменяемость — это идея о том, что изменения никогда не вносятся в живые системы. Вместо этого автоматизируйте изменения и используйте их для создания совершенно новых, заведомо исправных компонентов с нуля.
- Проверяйте на идемпотентность: Будьте скрупулезны в создании автоматизации, свободной от побочных эффектов.
- Все компоненты процедуры должны быть идемпотентными — Только если известно, что все компоненты процедуры являются идемпотентными, процедура в целом может быть идемпотентной.

Рисунок 87 — Алгоритм приведения объекта к желаюмому состоянию.

Выполнение скриптов локально и удаленно

Для настройки удаленных систем необходимо получить доступ к ним и выполнить на них сценарии. Существует множество способов сделать это:

- Вы можете хранить скрипты локально, передавать их на целевые машины с помощью утилиты оболочки, например, `scp`, затем входить на удаленную машину с помощью ssh и выполнять их.
- Вы можете передавать скрипты на удаленную машину с помощью `cat | ssh` и выполнять их последовательно с другими командами, перехватывая и возвращая результаты на терминал одной командой.
- Вы можете установить клиент безопасной передачи файлов общего назначения, например, `SFTP`, затем использовать эту утилиту для подключения к удаленной машине, передачи, установки соответствующих разрешений, а затем выполнения вашего файла сценария.
- Вы можете хранить скрипты на веб-сервере, подключаться к удаленной машине и получать их с помощью `wget`, `curl` или других утилит, или хранить скрипты в репозитории `Git`. Установите `git` на удаленную машину, клонируйте репозиторий, проверьте ветку и выполните найденные там скрипты.
- Можно локально установить полноценное решение для удаленной работы, например, VNC или NoMachine, установить его сервер на целевой машине (для этого обычно требуется также установить

графическую среду рабочего стола), передать/скопировать, а затем выполнить сценарии.

- Если целевые устройства предоставляются на облачной платформе, обычно есть возможность внедрить конфигурационный скрипт с помощью той же команды CLI или действия WebUI, которое манифестирует платформу.

Большинство публичных облачных сервисов позволяют внедрять конфигурационные скрипты непосредственно в экземпляры виртуальных машин для выполнения во время загрузки.

Почти каждый разработчик в конечном итоге будет использовать эти и другие методы в тот или иной момент, в зависимости от задачи (задач), ограничений окружающей среды, доступа в интернет и других ограничений безопасности, а также институциональной политики.

Понимание этих методов и их применение на практике очень важно, поскольку процедурная автоматизация некоторых ручных процессов все еще может быть полезной, даже при использовании передовых инструментов развертывания для большинства задач DevOps. Для ясности, это не является хорошей практикой, но уровень развития инструментария еще не является совершенным или достаточно полным, чтобы решить все проблемы, с которыми вы можете столкнуться.

Облачная автоматизация

Облачные вычисления на базе инфраструктуры как сервиса (IaaS) являются типичной целью для автоматизации. Автоматизация облака позволяет вам предоставлять виртуализированные узлы, настраивать виртуальные сети и другие возможности подключения, запрашивать услуги, а затем развертывать приложения на этой инфраструктуре.

Поставщики облачных услуг и сообщества с открытым исходным кодом часто предоставляют специализированные подсистемы для популярных инструментов развертывания. Эти подсистемы извлекают полный перечень ресурсов из облачной структуры и обновляют его в режиме реального времени, пока автоматизация вносит изменения, что позволяет вам легче написать автоматизацию для управления этими ресурсами.

Вы также можете управлять облачными ресурсами с помощью скриптов, написанных на Bash, Python или других языках. В этом вам помогут многочисленные инструменты, которые упрощают доступ к целям автоматизации. К ним относятся:

- CLI и SDK, которые оборачивают REST и другие интерфейсы аппаратного обеспечения, объектов виртуальной инфраструктуры, плоскостей управления высшего порядка и облачных API. Это делает их функции доступными из оболочек (и через Bash скрипты) и внутри программ на Python.

– Инструменты командной строки и встроенные в Python анализаторы могут преобразовывать выходные данные JSON и YAML, возвращаемые CLI и SDK, в красивые, легко читаемые форматы и в собственные структуры данных Python для удобного манипулирования.

Облачные CLI и SDK

IaaS и другие типы инфраструктурных облаков также предоставляют CLI и SDK, позволяющие легко подключаться к их базовым интерфейсам, которые обычно основаны на REST.

VMware

Основным CLI VMware теперь является Datacenter CLI, который позволяет работать с vCenter Server API и VMware Cloud on AWS через командную строку. Он написан на языке Python и работает на Linux, Mac и Windows.

VMware также предлагает vSphere CLI для Linux и Windows, который позволяет управлять хостами виртуализации ESXi, серверами vCenter и предлагает подмножество команд DCLI. Компания также предлагает PowerCLI для Windows PowerShell, который предоставляет команды для сред vSphere, vCloud, vRealize Operations Manager, vSAN, NSX-T, VMware Cloud on AWS, VMware HCX, VMware Site Recovery Manager и VMware Horizon.

VMware также предлагает множество SDK для популярных языков (включая Python), предназначенных для vSphere Automation, vCloud Suite, vSAN и других продуктов.

OpenStack

Проект OpenStack предоставляет клиент OpenStack Client (OSC), написанный на языке Python. OSC позволяет получить доступ к API OpenStack Compute, Identity, Image, Object Storage и Block Storage.

При установке клиентов командной строки также устанавливается OpenStack Python SDK, что позволяет выполнять множество команд OpenStack на языке Python.

Наборы инструментов OpenStack также доступны для многих других популярных языков.

Cisco UCS — a bare metal cloud

Если вы знакомы с продуктами Cisco Compute, *не зря же мы в СПбГУТ вузе платиновым партнёром одноимённой кампании*, включая Unified Computing System (UCS), Hyperflex, UCS Manager и систему управления инфраструктурой Intersight, вы знаете, что они фактически являются шлюзом для глобального SaaS-управления инфраструктурой UCS/Hyperflex организации.

Основным API Cisco для этой инфраструктуры является Cisco Intersight RESTful API. Это OpenAPI-совместимый API, который можно опрашивать с помощью Swagger и других инструментов OpenAPI с открытым исходным кодом. Они позволяют создавать специализированные SDK для произвольных

языков и сред, а также упрощают задачу документирования API (и поддержки SDK).

Cisco предоставляет и поддерживает ряд SDK для Intersight RESTful API, включая SDK для Python и Microsoft PowerShell. Они также предоставляют ряд модулей Ansible.

Инструменты автоматизации

Важнейшие концепции

Идемпотентность: обзор

Идемпотентное программное обеспечение дает один и тот же желаемый результат при каждом запуске. В программном обеспечении для развертывания идемпотентность обеспечивает конвергенцию и композитивность. Идемпотентные компоненты развертывания позволяют вам:

- Легче собирать компоненты в коллекции, которые создают новые виды инфраструктуры и выполняют новые операционные задачи;
- Выполнять целые коллекции сборки/развертывания для безопасного устранения небольших проблем с инфраструктурой, выполнения инкрементных обновлений, изменения конфигурации или управления масштабированием.

Например, предположим, что неправильная конфигурация оператора вызывает проблему в сложном кластере Kubernetes. Idempotent позволяет вам вернуть последнее изменение в коде развертывания и перестроить кластер с нуля за пять минут, будучи полностью уверенным в том, что у вас будет рабочий продукт.

В этом заключается основная идея инфраструктуры как кода. Идемпотентность гарантирует, что ваша кодовая база достигнет желаемой цели, сходясь к последовательности зависимых целей. Таким образом, запуск вашей кодовой базы против базовой цели инфраструктуры приведет к идемпотентному результату — созданию новой или пересмотренной рабочей инфраструктуры.

Процедура vs декларативность

Процедурный код может обеспечить идемпотентность, но многие инструменты управления инфраструктурой, развертывания и оркестровки используют другой метод — создание декларативного кода. Декларатив — это статическая модель, представляющая желаемый конечный продукт. Эта модель используется промежуточным программным обеспечением, которое учитывает детали, специфичные для развертывания, изучает текущие обстоятельства и приводит реальную инфраструктуру в соответствие с моделью, используя наименее разрушительный и, как правило, наименее трудоемкий путь.

В настоящее время большинство популярных средств автоматизации характеризуются как процедурные или декларативные по своей сути. Например, Ansible и Puppet часто описываются как использующие

декларативные языки доменов (DSL), в то время как Chef считается более процедурным по своей сути.

Это несколько искусственное различие, поскольку все эти платформы (а также BASH, Python и т.д.) являются процедурными на самом низком уровне; Ansible основана на Python, Puppet и Chef построены на Ruby. Все они могут использовать как декларативные, так и процедурные методы по мере необходимости, и многие реальные задачи автоматизации требуют применения обоих подходов.

Типичная краткая декларация Ansible, которая устанавливает веб-сервер Apache на хост Ubuntu

В этом примере параметр state: present можно заменить на state: absent, чтобы удалить пакет, если он найден. Параметр update-cache: yes выполняет эквивалент apt-get update перед попыткой установки.

```
- name: Install Apache webserver
  apt:
    name: apache2
    state: present
    update-cache: yes
```

Обеспечение vs Конфигурация vs Деплой vs Оркестрация

Специалисты по эксплуатации склонны мыслить в терминах иерархии уровней инфраструктуры и связанных с ними задач-доменов:

- **Обеспечение** — это получение вычислительной, складской и сетевой инфраструктуры (реальной или виртуальной), обеспечение связи, введение ее в эксплуатацию и обеспечение готовности к использованию операторами и разработчиками (например, установка операционной системы, метрик машинного уровня, ключей ssh и самого низкого уровня операционного инструментария).
- **Конфигурирование** — означает установку базовых приложений и сервисов, а также выполнение операций, задач и тестов, необходимых для подготовки платформы низкого уровня к развертыванию приложений или платформы более высокого уровня.
- **Деплой** — включает создание, организацию, интеграцию и подготовку многокомпонентных приложений (например, кластеров баз данных) или платформ более высокого уровня (например, кластеров Kubernetes), часто на нескольких узлах.
- **Оркестрация** — может означать несколько вещей. В конкретном смысле она обычно относится к встроенной пользователем или платформой автоматизации, направленной на управление жизненным циклом рабочей нагрузки и динамическое реагирование на изменяющиеся условия (например, путем автомасштабирования или самовосстановления), особенно в контейнерных средах. В абстрактном смысле это понятие может относиться к процессам или рабочим

процессам, которые связывают задачи автоматизации для получения бизнес-преимуществ, например, самообслуживания.

Люди, пришедшие в операционную деятельность из разработки программного обеспечения, как правило, имеют более свободный взгляд на то, как следует использовать эти термины. Они склонны использовать термин развертывание для всего, что не является оркестровкой. Они проводят наиболее сильное различие между "вещами, которые нужно сделать, чтобы подготовить систему к тестированию/использованию" и "корректировками, которые система должна сделать автоматически или которые вас могут попросить сделать для нее".

Люди также используют фразу "управление конфигурацией" при описании инструментов автоматизации ИТ в целом. Это может означать одно из двух:

- Расширение понятия "конфигурация" как установка приложений и сервисов.
- Различие между конфигурацией и разработкой: области, где статические описания могут аккуратно представлять рабочие продукты, и области, в которых процессы должны быть явными и возможные побочные эффекты должны быть предвидены.

Statelessness

Автоматизация работает лучше всего, когда приложения могут быть сделаны без статических данных. Это означает, что их перераспределение на месте не приведет к уничтожению или потере данных, необходимых пользователям или операторам.

- Not stateless — Приложение, сохраняющее важную информацию в файлах или в базе данных на локальной файловой системе.
- Stateless — Приложение, которое сохраняет свое состояние в отдельной базе данных или предоставляет сервис, не требующий запоминания состояния между вызовами.

Обсуждение автоматизации всего стека (инфраструктура + приложения) в этой теме предполагает, что обсуждаемые приложения не имеют состояния и/или что вы, разработчик, выяснили, как сохранить состояние в вашем приложении, чтобы ваша автоматизация могла работать без разрушения.

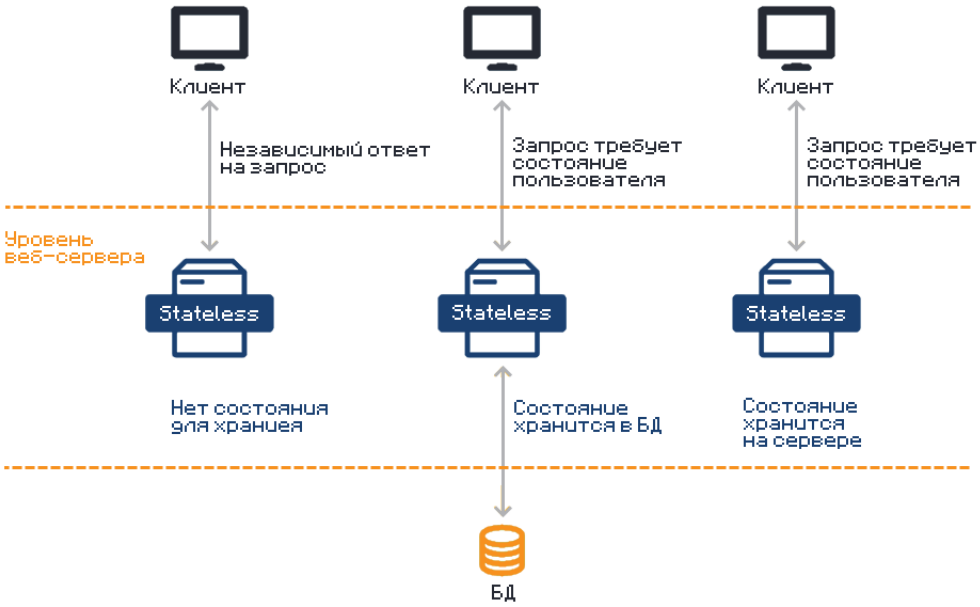


Рисунок 88 — Примеры Statelessness и Statefulness

На этой диаграмме (Рисунок 88) изображены три вида приложений:

- Stateless / No state to store —приложение требует только атомарных/синхронных взаимодействий между клиентом и сервером. Каждый запрос от клиента к серверу возвращает результат, полностью независимый от предыдущих и последующих запросов. Примером такого приложения является публичный веб-сервер, который по запросу браузера возвращает HTML-страницу, изображение или другие данные. Приложение может быть масштабировано путем дублирования серверов и данных за простым балансировщиком нагрузки.
- Stateless / Состояние хранится в базе данных — состояние пользователя хранится в базе данных, доступной для любого веб-сервера на среднем уровне. Примером такого применения является веб-сервер, которому необходимо знать о соответствии между идентификатором пользователя и его cookie. Можно свободно добавлять новые веб-серверы и копии веб-сайта, не нарушая текущие сессии пользователей и не требуя, чтобы каждый запрос от конкретного пользователя направлялся на конкретный сервер, поддерживающий его сессию.
- Stateful / Состояние хранится на сервере — запись состояния пользователя должна поддерживаться в течение серии транзакций. Примером такого применения является веб-сайт, требующий аутентификации. Приложению не разрешается обслуживать страницы пользователя, который не вошел в систему. Состояние пользователя обычно сохраняется путем предоставления клиенту идентификационного файла cookie, который возвращается на сервер с каждым новым запросом и используется для сопоставления с хранящимся там идентификатором. Это приложение нельзя масштабировать, просто добавляя серверы. Если вошедший в систему пользователь направляется на сервер, который не сохранил

идентификатор, соответствующий его cookie, этот сервер не распознает его как вошедшего в систему и отклонит его запрос.

Приложения, которым необходимо поддерживать состояние, являются неудобными кандидатами на автоматизацию полного стека, поскольку состояние будет уничтожено при специальной перестройке поддерживающей инфраструктуры. Они также не могут быть эффективно перемещены с одного пула ресурсов (например, с одного набора серверов приложений или хостов) на другой.

Популярные инструменты автоматизации

В настоящее время наиболее популярными инструментами являются Ansible, Puppet и Chef. Они обладают следующими характеристиками:

1. Относительно просты в освоении;
2. Доступны open-source версии;
3. Плагины и адаптеры позволяют прямо или косвенно управлять многими типами ресурсов: программно-определяемой пустой инфраструктурой (Cisco UCS, устройства Cisco), частным облаком (VMware, OpenStack) и публичным облаком (Amazon Web Services, Microsoft Azure, Google Cloud Platform).

Ansible

Ansible, вероятно, является наиболее популярным из современных решений для автоматизации. Он доступен с открытым исходным кодом, а также в версии с дополнительными функциями от IBM/Red Hat, которая называется Ansible Tower. Его название происходит от романов писательницы Урсулы К. ЛеГуин, в которых "ансибл" — это технология будущего, позволяющая мгновенно общаться на космических расстояниях.

Базовая архитектура Ansible очень проста и легка.

- Ansible control node работает практически на любой Linux-машине под управлением Python 2 или 3, включая ноутбук, виртуальную машину Linux, расположенную на ноутбуке любого типа, или на небольшой виртуальной машине, расположенной рядом с управляемыми облачными ресурсами. Все обновления системы выполняются на узле управления.
- Control node подключается к управляемым ресурсам по SSH. Через это соединение Ansible может:
    1. Выполнять команды shell на удаленном сервере или взаимодействовать с удаленным маршрутизатором или другим сетевым объектом через интерфейс REST.
    2. Внедрять скрипты Python в целевые объекты и удалять их после выполнения.
    3. При необходимости установите Python на целевые машины.

– Плагины позволяют Ansible собирать факты и выполнять операции на инфраструктуре, которая не может локально запускать Python, например, REST-интерфейсы облачных провайдеров.

Ansible в основном управляется из командной строки Bash, а код автоматизации разрабатывается и поддерживается с помощью любого стандартного текстового редактора. Atom — хороший выбор, поскольку он позволяет легко удаленно работать с кодом, хранящимся во вложенных системах каталогов.

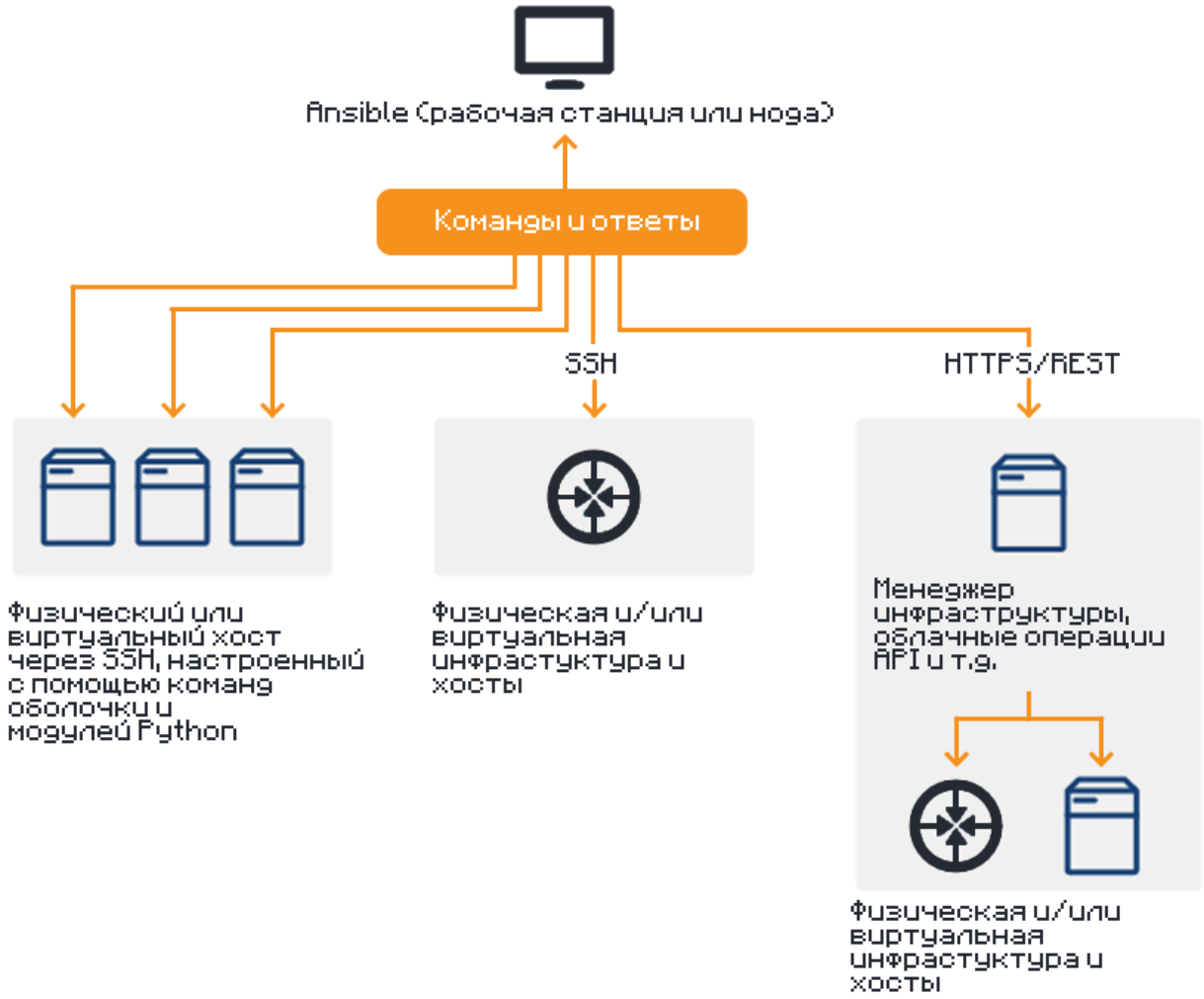


Рисунок 89 — Архитектура Ansible

Установка Ansible

Приложение узла управления Ansible устанавливается на машину Linux (часто виртуальную машину) из общедоступного репозитория пакетов. Чтобы установить Ansible на рабочую станцию, обратитесь к документации по установке, соответствующей вашему устройству.

Структура кода Ansible

В структуре кода Ansible работа разделяется на файлы YAML (.yml), которые содержат последовательность задач, выполняемых в порядке сверху

вниз. Типичная задача называет и параметризирует модуль, выполняющий работу, подобно вызову функции с параметрами.

В Ansible есть сотни готовых модулей Python, которые содержат функции и мета-функции уровня операционной системы. Некоторые модули, такие как raw, делают только одну вещь: они представляют команду в строковой форме оболочке, фиксируют код возврата и любой вывод консоли и возвращают их в доступных переменных. Модуль apt можно использовать для установки, удаления, обновления и изменения отдельных пакетов или списков пакетов на веб-сервере Linux, работающем под управлением Debian Linux. Если вы хотите узнать больше, документация по apt даст вам представление о масштабах и возможностях модулей Ansible.

Плейбуки и роли

Плейбук Ansible может быть написан как монолитный документ с серией модульных именованных задач. Чаще всего разработчики строят модель сложной задачи DevOps из низкоуровневых последовательностей задач (называемых "ролями"), а затем ссылаются на них в более высокоуровневых плейбуках, иногда добавляя дополнительные задачи на уровне плейбука.

Такое разделение задач имеет множество преимуществ.

- Ясность — При наличии небольшого контекста практически любой сможет интерпретировать плейбук более высокого уровня, содержащий ссылки на четко названные роли.
- Возможность повторного использования и совместного использования — Роли могут использоваться повторно и могут быть достаточно тесно связаны со спецификой инфраструктуры. Роли также потенциально могут использоваться совместно. Проект Ansible поддерживает хранилище определений ролей с открытым исходным кодом, называемое Ansible Galaxy.

Пример плейбука

Следующий пример плейбука создает трехузловой кластер Kubernetes на коллекции серверов:

- Устанавливает Python 2 на все серверы и выполняет общие шаги по настройке на всех узлах с помощью роли configure-nodes, устанавливает программное обеспечение Kubernetes и Docker в качестве контейнерного движка, а также настраивает Docker для работы с Kubernetes. Сами команды Ansible не показаны.
- Назначает один узел ведущим, устанавливает контейнерную сеть Weave (один из многих сетевых фреймворков, работающих с Kubernetes) и выполняет задачи по завершению работы.
- Присоединяет узлы k8sworker к узлу k8smaster.
- Утверждение become: true дает Ansible root-привилегии (через sudo) перед попыткой выполнения операции.

– Строка gather_facts: false в первой строфе предотвращает выполнение автоматического опросника системных фактов на целевой машине до установки Python. При выполнении последующих строф факты будут собираться автоматически, по умолчанию.

```
---
- hosts: all
  become: true
  gather_facts: False
  tasks:
    - name: install python 2
      raw: test -e /usr/bin/python || (apt -y update && apt install -y
python-minimal)
- hosts: all
  become: true
  roles:
    - configure-nodes
- hosts: k8smaster
  become: true
  roles:
    - create-k8s-master
    - install-weave
    - wrapup
- hosts:
    - k8sworker1
    - k8sworker2
  become: true
  roles:
    - join-workers
...
```

Организация проекта Ansible

Проекты Ansible обычно организованы в виде вложенной структуры каталогов, как показано ниже. Эта иерархия легко помещается под контроль версий и используется для инфраструктуры в стиле GitOps как код. Для примера см. раздел "Расположение каталогов" в документации Ansible.

Элементы иерархии папок Ansible

Иерархия папок/файлов Ansible включает следующие основные элементы:

– Файлы инвентаризации — организуют инвентаризацию ресурсов (например, серверов) под управлением. Это позволяет вам нацеливать развертывания на последовательность сред, таких как dev, test, staging, production. Для получения дополнительной информации о файлах инвентаризации см. раздел "Как создать инвентаризацию" в документации Ansible.
– Файлы переменных — файлы описывают значения переменных, которые относятся к группам хостов и отдельным хостам.

- Файлы библиотек и утилит — необязательные файлы содержат код Python для пользовательских модулей и утилит, которые они могут потребовать. Вы можете написать пользовательские модули и утилиты самостоятельно или получить их из Ansible Galaxy или других источников. Например, Ansible поставляется с большим количеством уже имеющихся модулей для управления основными функциями Cisco ACI, но также предоставляет учебники по составлению дополнительных пользовательских модулей для функций ACI, которые в настоящее время не охвачены.
- Основные файлы плейбука — написаны на языке YAML, эти файлы могут ссылаться друг на друга или на роли более низкого уровня.
- Папки и файлы ролей — дерево папок каждой роли объединяет ресурсы, которые в совокупности позволяют выполнить этап детальной настройки. Папка роли содержит папку /tasks с файлом задач main.yml. Она также содержит папку с файлами задач асинхронных обработчиков.

Ansible в масштабе

Управляющий узел Ansible (control node) спроектирован так, чтобы располагаться рядом с инфраструктурой, которой он управляет. Например, он может располагаться на виртуальной машине или в контейнере, работающем в той же подсети, что и управляемые ресурсы. Предприятия и организации с большим количеством управляемых хостов обычно используют много узлов управления Ansible, распределяя их по пулам инфраструктуры по мере необходимости.

Если вы не занимаетесь быстрой непрерывной доставкой, узлы Ansible даже не нужно поддерживать между развертываниями. Если код развертывания Ansible хранится в системе контроля версий, узлы управления можно запускать или создавать заново по мере необходимости.

Для крупных организаций существуют проблемы масштабирования, такие как гибкое и безопасное управление и контроль доступа ко многим узлам Ansible. Сюда также входит передача удаленных контроллеров под беспрепятственный и безопасный контроль централизованной автоматизации предприятия. Для этого существует два решения для плоскости управления:

- Коммерческий продукт Red Hat Ansible Tower предоставляет сложный веб-интерфейс, REST API и богатые возможности управления доступом на основе ролей.
- Альтернатива с открытым исходным кодом, сопоставимая по возможностям, проект AWX, дистрибутивом которого является Ansible Tower. Однако считается, что AWX представляет собой ветвь разработки, которая проходит минимальное тестирование и не доступна в виде подписанных двоичных файлов. Это может стать проблемой для многих предприятий.

Непрерывная доставка вокруг развертывания Ansible может быть выполнена с помощью любого инструмента автоматизации CI/CD общего назначения, такого как Jenkins или Spinnaker. Крупные, более сложные проекты часто используют фреймворк гейтинга с открытым исходным кодом Zuul, изначально разработанный в рамках проекта OpenStack и выделенный самостоятельно в 2018 году. Проект AWX, среди многих других, является проектом с Zuul-гейтингом.
Более масштабные внедрения Ansible также получат пользу от Ansible Vault — встроенной функции, позволяющей шифровать пароли и другую конфиденциальную информацию. Она представляет собой простую и легко управляемую альтернативу хранению конфиденциальной информации в плейбуках, ролях или где-либо еще в виде открытого текста.

Пример Ansible

Это упражнение позволит нам просмотреть структуру простой программы Ansible, которая получает информацию о контейнере, в котором находится демонстрационная среда. Обратите внимание, что Ansible обычно использует ssh для подключения к удаленным узлам и выполнения команд. В этом примере строка в файле playbook.yml верхнего уровня указывает Ansible запустить этот плейбук локально, без использования ssh.

```
connection: local
```

Обычно Ansible используется для выполнения задач развертывания и конфигурирования. Например, вы можете использовать его для создания простого веб-сайта на удаленном хосте. Давайте посмотрим, как это может работать.

Вы можете выполнить это упражнение самостоятельно, а можете просто читать. Если вы хотите выполнить это упражнение, вам понадобятся:

- Целевой хост под управлением совместимой операционной системы (например, сервер Ubuntu 18.04)
- SSH и аутентификация по ключу настроены на этом хосте
- Ansible, установленный на вашей локальной рабочей станции

Построение дерева файлов проекта Ansible

Для целей данного упражнения имя хоста целевой машины (с возможностью разрешения DNS) будет просто target. Если у вас установлен и настроен собственный целевой хост, подставьте имя своего хоста при создании файлов. Получив SSH-доступ к целевой машине, начните создавать базовую структуру папок для вашего проекта Ansible.

```
$ mkdir myproject
$ cd myproject
```

На корневом уровне проекта вам нужно:

– Файл инвентаризации, содержащий информацию о машине (машинах), на которой вы хотите развернуть систему.
– Файл верхнего уровня site.yml, содержащий наиболее абстрактный уровень инструкций по выполнению развертывания.
– Структура папок с ролями, содержащая вашу роль веб-сервера.

```
$ touch inventory
$ touch site.yml
$ mkdir roles
$ cd roles
$ ansible-galaxy init webservers
```

Создание файла инвентаризации

Ваш файл инвентаризации для этого проекта может быть очень простым. Пусть это будет DNS-разрешаемое имя хоста вашей целевой машины:

```
[webservers]
target # также может быть IP-адресом
```

Вы определяете группу под названием webservers и помещаете в нее имя хоста (или IP) вашей целевой машины. Вы можете добавить новые имена хостов/IP в этот блок группы или добавить дополнительные блоки групп, чтобы назначить хосты для более сложных развертываний. Имя webservers является полностью произвольным. Например, если у вас есть шесть серверов, которые вы хотите настроить общим способом, и вы настроите три сервера как веб-серверы и три как серверы баз данных, ваш реестр может выглядеть следующим образом:

```
[webservers]
target1
target2
target3
[dbservers]
target4
target5
target6
```

На самом деле вам не нужно создавать общую группу, поскольку Ansible предоставляет средства для применения общей конфигурации ко всем серверам в инвентаре, что вы увидите через некоторое время.

Плейбук верхнего уровня обычно описывает порядок, разрешения и другие детали, в соответствии с которыми применяются действия конфигурации нижнего уровня, определенные в ролях. Для данного примера проекта site.yml выглядит следующим образом:

```
---
- hosts: webservers
  become: true
  roles:
  - webservers
```

site.yml определяет, на каких хостах вы хотите выполнить операцию, и какие роли вы хотите применить к этим хостам. Строка become: true говорит Ansible, что вы хотите выполнять роли от имени root, через sudo.

Обратите внимание, что вместо hosts: webservers вы можете применить эту роль ко всем целевым хостам (что в данном случае будет работать нормально, так как у вас только один целевой хост), заменив строку:

```
- hosts: all
```

Создание роли веб-сервера

Следующий шаг — создание роли, которая устанавливает и настраивает ваш веб-сервер. Вы уже создали структуру папок для роли с помощью ansible-galaxy. Код для роли обычно содержится в файле main.yml в каталоге /tasks роли. Вы можете отредактировать файл roles/webserver/tasks/main.yml напрямую, чтобы он выглядел следующим образом:

```
---
- name: Perform updates and install apache2
  apt:
    name: apache2
    state: present
    update_cache: yes
- name: Insert new homepage index.html
  copy:
    src: index.html
    dest: /var/www/html
    owner: myname
    mode: '0444'
```

У роли есть две задачи:

- Развернуть Apache2.
- Скопировать новый файл index.html в HTML-корень Apache2, заменив им страницу index.html по умолчанию.

В строке apt: укажите имя пакета, его требуемое состояние и дайте указание модулю apt обновить свой кэш. По сути, вы выполняете sudo apt update перед установкой.

Во второй строфе процедура копирования Ansible перемещает файл из вашей локальной системы в каталог на целевой, а также изменяет его владельца и разрешения. Это эквивалентно:

```
$ chown myname index.html
$ chmod 444 index.html
```

Создание файла index.html

Конечно, вам также потребуется создать новый файл index.html. Команда Ansible copy предполагает, что такие файлы будут храниться в каталоге /files вызывающей роли, если не указано иное. Перейдите в этот каталог и создайте следующий файл index.html, сохранив после этого изменения:

```
<html>
    <head>
        <title>Мой сайт</title>
    </head>
<body>
    <h1>Дароу!</h1>
</body>
</html>
```

Запуск развертывания

Теперь вы готовы к запуску развертывания. Из каталога верхнего уровня вашего проекта вы можете сделать это с помощью оператора:

```
$ ansible- -i inventory -u myname -K site.yml
```

-i: называет ваш файл инвентаризации.
-u: называет вашего пользователя sudo.
-K: указывает Ansible запросить ваш пароль sudo, когда он начнет выполнение.
- site.yml — это файл, который управляет вашим развертыванием.

Если все в порядке, Ansible должен запросить у нас ваш пароль BECOME (пароль sudo), а затем выдать результаты, похожие на следующие:

```
BECOME password:
PLAY [webservers]
*****************************************************************
TASK [Gathering Facts]
*******************************************************
ok: [192.168.1.33]
TASK [webservers : Perform updates and install apache2]
****************************
changed: [192.168.1.33]
TASK [webservers : Insert new homepage index.html]
********************************
changed: [192.168.1.33]
PLAY RECAP
*********************************************************************
```

```
192.168.1.33                : ok=3    changed=2    unreachable=0
failed=0    skipped=0    rescued=0    ignored=0
```

И теперь, если вы зайдете на IP-адрес вашей целевой машины в браузере, вы должны увидеть вашу новую домашнюю страницу.

Обратите внимание, что Ansible предоставляет полный отчет о каждом выполнении, отмечая, был ли шаг действительно выполнен, или Ansible определил, что желаемая цель уже достигнута (В этом случае ничего не произошло, но шаг считается выполненным "нормально"). Это пример того, как Ansible поддерживает идемпотентность. Как правило, вы можете запускать развертывание Ansible столько раз, сколько необходимо, не вводя целевую систему в неизвестное состояние.

Пробежимся по Ansible CI/CD

Давайте рассмотрим пример, как если бы они были частью конвейера CI/CD.

Сотрудничающий с вами на GitHub разработчик коммитит изменение на сайте, например, в файле index.html.

Далее тесты в репозитории выполняют проверку синтаксиса и вменяемости, а также правила проверки кода, например, для каждого pull request.

Если тесты проходят: Принимает их коммит и уведомляет CI/CD-сервер для запуска тестов.

Если тесты не пройдены: Отклоняет коммит на основании неудачных тестов и просит отправить его повторно.

Далее система CI/CD, например Jenkins, подготавливает среду и запускает предопределенные тесты для любого плейбука Ansible. Она должна каждый раз указывать ожидаемую версию и устанавливать ее. Вот пример пайплайна:

```
$ pip install ansible==5.8.0
$ ansible-version
$ ansible-playbook main.yml --syntax-check
$ ansible-playbook -i staging-servers.cfg main.yml --check
$ ansible-playbook -i staging-servers.cfg main.yml -vvvv
```

С помощью этого пайплайна вы будете уверены:

- В файле `main.yml` нет синтаксических ошибок.
- Правила проверки кода соблюдаются еще до того, как код будет помещен в репозиторий.
- Все модули написаны правильно и доступны в окружении с параметром `-check`.
- Есть общая версия Ansible, например `5.8.0`, потому что она установлена в среде CI.
- Плейбук может запускаться в среде `staging` на серверах `staging`.
- Другие люди могут увидеть, что было запущено, используя параметр `-vvvv` с высокой степенью детализации.

После того как Jenkins выполнит задание, вы можете получить уведомление о том, что все готово для `staging`, и вы можете перенести эти изменения в `production` с помощью другого конвейера, на этот раз для push to production. Такова сила контроля версий, Ansible и многократного продвижения через окружения с помощью CI/CD.

Puppet

Puppet был основан как open source в 2005 году и коммерциализирован как Puppet Enterprise компанией Puppet Labs в 2011 году.

Основная архитектура Puppet имеет следующие характеристики:

- Назначенный сервер для размещения основных компонентов приложения:
    1. Puppet Server (исторически называемый "Puppet Master")
    2. Facter, служба сбора фактов
    3. PuppetDB, в которой могут храниться факты, каталоги узлов и история последних событий конфигурации.
- Безопасный клиент, также известный как Puppet Agent, установленный и настроенный на целевых машинах. Клиенты и сервер проходят взаимную аутентификацию с помощью самоподписанных сертификатов, а для передачи данных используется протокол SSL. Агенты собирают факты (под контролем службы Facter) и вносят изменения в конфигурацию по указанию Puppet Server.
- Для облачных API и оборудования, на котором не может работать агент, в Puppet есть модули, позволяющие устанавливать такие соединения.
- В масштабных реализациях, где под управлением находится множество устройств, не поддерживающих агентов, Puppet позволяет использовать прокси-агент, чтобы разгрузить работу по прямому подключению к CLI устройств и обмену информацией.

Операторы взаимодействуют с Puppet Server в основном через SSH и командную строку.

Puppet Server может быть виртуальной машиной или даже контейнером Docker для небольших самоучителей, и Puppet предоставляет компактную установку Docker для этих целей, называемую PUPPERWARE. Для создания Puppet Server на Linux доступны стандартные пакеты, что в настоящее время является единственным вариантом установки сервера. Агенты Puppet (также называемые клиентами) доступны для Linux, Windows и MacOS.

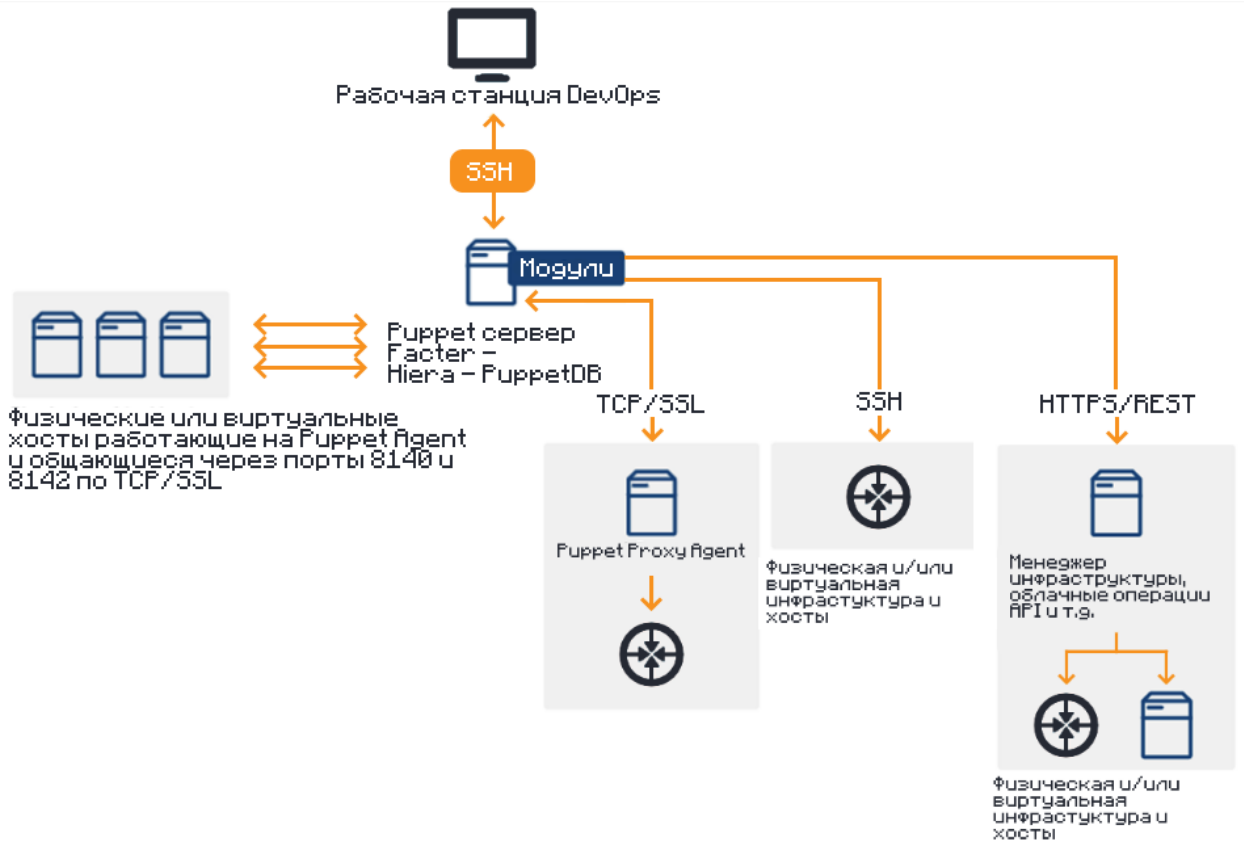


Рисунок 90 — Архитектура Puppet

Установка Puppet

Puppet Server требует достаточно мощного оборудования (или большой виртуальной машины), а также установки, настройки и тестирования клиента Network Time Protocol. В блогах можно найти множество статей о том, как установить Puppet, но проект Puppet быстро развивается, и сторонние статьи могут быть устаревшими или ненадежными.

Когда у вас запущен Puppet Server, вы можете начать установку агентов Puppet на хосты, которыми вы хотите управлять. Агенты должны иметь файл puppet.conf, настроенный на взаимодействие с Puppet Server. После запуска клиентской службы ее сертификат будет подписан сервером. Теперь сервер сможет собирать факты от клиента и обновлять состояние клиента с любыми данными.

Структура кода Puppet

Как и Ansible, Puppet позволяет хранить компоненты проекта или дискретной конфигурации в дереве каталогов (/etc/puppetlabs/code/environments). Дочерние папки создаются в соответствии с конфигурацией в puppet.conf или оператором.

В качестве примера, вы можете объявить environment = production в puppet.conf. Puppet создаст каталог для этой среды по умолчанию, который будет содержать подкаталог modules, в котором вы сможете хранить дочерние проекты и манифесты для того, что вам нужно собрать и настроить (/etc/puppetlabs/code/environments/production/modules).

Чтобы начать небольшой проект, вы можете создать папку внутри этой директории, а затем в этой папке создать другую под названием manifests, где будут храниться файлы манифестов, объявляющие операционные классы. Это единицы кода, описывающие операцию конфигурации. Файлы манифестов обычно заканчиваются расширением .pp и написаны на декларативном языке Puppet, который чем-то похож на Ruby и был вдохновлен форматом конфигурационных файлов Nagios.

Подобно Ansible и другим инструментам конфигурирования, Puppet предоставляет множество ресурсов, которые можно вызывать для определения конфигурационных действий, выполняемых на хостах и подключенной инфраструктуре. Основная идея очень похожа на практику Ansible, где класс, вызывающий ресурс, параметризуется так, чтобы функционировать идемпотентно, и применяется в контексте для получения одного и того же желаемого результата при каждом запуске.

Puppet поставляется с набором базовых ресурсов (шаблонов для выполнения конфигурационных действий), встроенных в систему. Множество дополнительных ресурсов для выполнения всевозможных операций с любыми видами инфраструктуры можно загрузить и установить из Puppet Forge с помощью команды puppet module.

Puppet в масштабе

Puppet Server в некоторой степени монолитен, и монолитная установка рекомендуется разработчиками (с открытым исходным кодом), которые утверждают, что правильно настроенный Puppet Server может управлять "до 4000" хостов.

Первым рекомендуемым шагом для размещения большего количества хостов является создание дополнительных "мастеров компиляции", которые компилируют каталоги для клиентских агентов и размещают их за балансировщиком нагрузки для распределения работы.

Пользователи Puppet Enterprise могут еще больше расширить возможности, заменив PuppetDB на отдельную базу данных PE-PostgreSQL. Продукт Puppet Enterprise предлагает множество других удобств, включая веб-консоль, позволяющую просматривать отчеты и журналы, а также выполнять некоторые виды конфигурации "наведи и щелкни".

Пример Puppet

В этом примере описано, как установить Puppet, а затем использовать Puppet для установки Apache 2 на устройство. Вы можете просто читать дальше, чтобы лучше понять Puppet.

Это приблизительно соответствует обычному рабочему процессу для операций Puppet в автоматизированной среде клиент/сервер. Обратите внимание, что модули могут быть совершенно общими и не содержать информации о конкретном сайте, а затем отдельно и многократно использоваться для настройки любого количества узлов или компонентов инфраструктуры. Поскольку модули и манифесты составляются в виде

текстовых файлов, их можно легко хранить согласованным образом в хранилище контроля версий, например, в Git.

Установка Puppet Server

Puppet Server требует достаточно мощного оборудования (или большой виртуальной машины), а также установки, настройки и тестирования клиента Network Time Protocol. Инструкции по установке сервера можно найти в документации Puppet

Установка Puppet Client

Когда у вас запущен Puppet Server, вы можете установить агенты Puppet на хост. Например, на системе Linux типа Debian вы можете установить Puppet Agent с помощью одной команды:

```
$ sudo apt-get install puppet-agent
```

Изменение

После установки Puppet Agent должен быть настроен на поиск Puppet Server. Добавьте следующие строки в файл `/etc/puppet/puppet.conf`:

```
[main]
certname = puppetclient
server = puppetserver
environment = production
runinterval = 15m
```

Это сообщает Клиенту имя хоста вашего сервера (определяется через /etc/hosts) и имя сертификата аутентификации, который вы сгенерируете на следующем этапе.

Запустите службу puppet на клиенте:

```
$ sudo /opt/puppetlabs/bin/puppet resource service puppet ensure=running
enable=true
```

Вы должны получить ответ, аналогичный следующему:

```
Notice: /Service[puppet]/ensure: ensure changed 'stopped' to 'running'
service { 'puppet':
  ensure => 'running',
  enable => 'true',
}
```

Обратите внимание на декларативный метод конфигурирования Puppet. Puppet использует себя для конфигурирования и активации своего сервиса. Обратите внимание на декларативный синтаксис, похожий на Ruby.

Выдача сертификатов

Агенты Puppet используют сертификаты для аутентификации на сервере перед получением своих конфигураций. Когда служба Client запускается в

первый раз, она посылает запрос на назначенный ей сервер, чтобы его сертификат был подписан, обеспечивая связь.
На сервере, выполнив команду ca list, вы получите список ожидающих сертификатов:

```
$ sudo /opt/puppetlabs/bin/puppetserver ca list
```

Ответ должен быть похож на следующий:

```
Requested                                                    Certificates:
    puppetclient                                                  (SHA256)
44:9B:9C:02:2E:B5:80:87:17:90:7E:DC:1A:01:FD:35:C7:DB:44:B6:36:6F:1F:C
C:DC:C2:E9:DD:72:61:E6:BE
```

Затем вы можете подписать сертификат, что позволит управлять удаленным узлом:

```
$ sudo /opt/puppetlabs/bin/puppetserver ca sign --certname puppetclient
```

Ответ:

```
Successfully signed certificate request for puppetclient
```

Теперь сервер и клиент надежно связаны и могут взаимодействовать. Это позволит Серверу собирать факты от Клиента, а вам — создавать конфигурации на Сервере, которые будут получены Клиентом и использованы для конвергенции его состояния (каждые 15 минут).

Создание конфигурации

Как и Ansible, Puppet позволяет хранить компоненты проекта или отдельной конфигурации в дереве каталогов:
Как и Ansible, Puppet позволяет хранить компоненты проекта или отдельной конфигурации в дереве каталогов:

```
/etc/puppetlabs/code/environments
```

Вспомогательные папки создаются в соответствии с конфигурацией в puppet.conf или оператором. В данном примере, объявив в `puppet.conf` `environment = production`, Puppet уже создал каталог для этого сайта по умолчанию, содержащий подкаталог `modules`, в котором мы можем хранить вспомогательные проекты и манифесты для вещей, которые нам нужно собрать и настроить.

```
/etc/puppetlabs/code/environments/production/modules
```

Теперь вы установите Apache2 на управляемом клиенте. Операции Puppet обычно выполняются от имени root, поэтому временно станьте root на сервере, введя:

```
$ sudo su -
```

Перейдите в папку /modules в среде /production:

```
$ cd /etc/puppetlabs/code/environments/production/modules
```

Создайте структуру папок, содержащую модуль install apache:

```
$ mkdir -p apache2/manifests
$ cd apache2/manifests
```

Внутри этой папки manifests создайте файл init.pp, который является зарезервированным именем для шага инициализации в модуле:

```
class apache2 {
  package { 'apache2':
    ensure => installed,
  }
  service { 'apache2':
    ensure  => true,
    enable  => true,
    require => Package['apache2'],
  }
}
```

Определение класса упорядочивает шаги, которые мы хотим выполнить:

Шаг 1. Вызвать ресурс пакета для установки именованного пакета. Если вы хотите удалить пакет, вы можете изменить ensure => installed на ensure => absent.

Шаг 2. Вызовите ресурс службы для запуска, если его требование (в данном случае, наличие Apache2) выполнено. Укажите ему убедиться, что служба доступна, а затем включите ее автоматический перезапуск при перезагрузке сервера.

Перейдите в соответствующую папку manifests:

```
$ cd /etc/puppetlabs/code/environments/production/manifests
```

Создайте файл site.pp, который вызывает модуль и применяет его к целевой машине:

```
node 'puppetclient' {
   include apache2
}
```

Развертывание конфигурации

У вас есть два варианта развертывания готовой конфигурации:

- Перезапуск Puppet Server приведет к тому, что манифесты будут скомпилированы и доступны Puppet Agent на указанном устройстве. Агент получит и применит их, установив Apache2 при следующем цикле обновления:
  ```
  $ systemctl restart puppetserver.service
  ```
- Для разработки и отладки можно вызвать Puppet Agent на целевой машине (в данном случае на нашей машине Puppet Client):
  ```
  $ sudo puppet agent -t
  ```

Агент немедленно опросит сервер, загрузит его каталог (конфигурации, которые на него ссылаются) и применит его. Результаты будут похожи на следующие:

```
root@target:/etc/puppetlabs/code/environments/production/manifests#
puppet agent -tInfo: Using configured environment 'production'Info:
Retrieving pluginfactsInfo: Retrieving pluginInfo: Retrieving
localesInfo: Caching catalog for puppetagentInfo: Applying
configuration version '1575907252'Notice:
/Stage[main]/Apache2/Package[apache2]/ensure: createdNotice: Applied
catalog in 139.88 seconds
```

После успешного развертывания приложения введите IP-адрес целевой машины в браузере. Должна открыться домашняя страница Apache2 по умолчанию.

Chef

Chef предоставляет полную систему для работы с инфраструктурой как с кодом. Продукты Chef частично лицензируются, но бесплатны для личного использования.

Продукты и решения Chef позволяют создавать, тестировать, организовывать, хранить в репозитории и выполнять на удаленных объектах инфраструктуру как код либо с отдельной рабочей станции Chef, либо опосредованно с центрального сервера Chef Infra Server. Вы должны знать основные компоненты Chef:

- Chef Workstation — автономная рабочая станция оператора, что может быть достаточно для небольших предприятий.
- Chef Infra Client (агент хоста) - Клиенты Chef Infra работают на хостах, получают шаблоны конфигурации и вносят необходимые изменения. Кулинарные книги (и прокси-клиенты) позволяют управлять оборудованием и ресурсами, которые не могут запустить Chef Infra Client локально (например, сетевые устройства).

– Сервер Chef Infra — отвечает на запросы от агентов Chef Infra на проверенных хостах и предоставляет обновления конфигурации, на основе которых агенты затем сходятся в конфигурации хоста.

Большинство задач конфигурирования также можно выполнять непосредственно между Chef Workstation и управляемыми узлами, и устройствами.

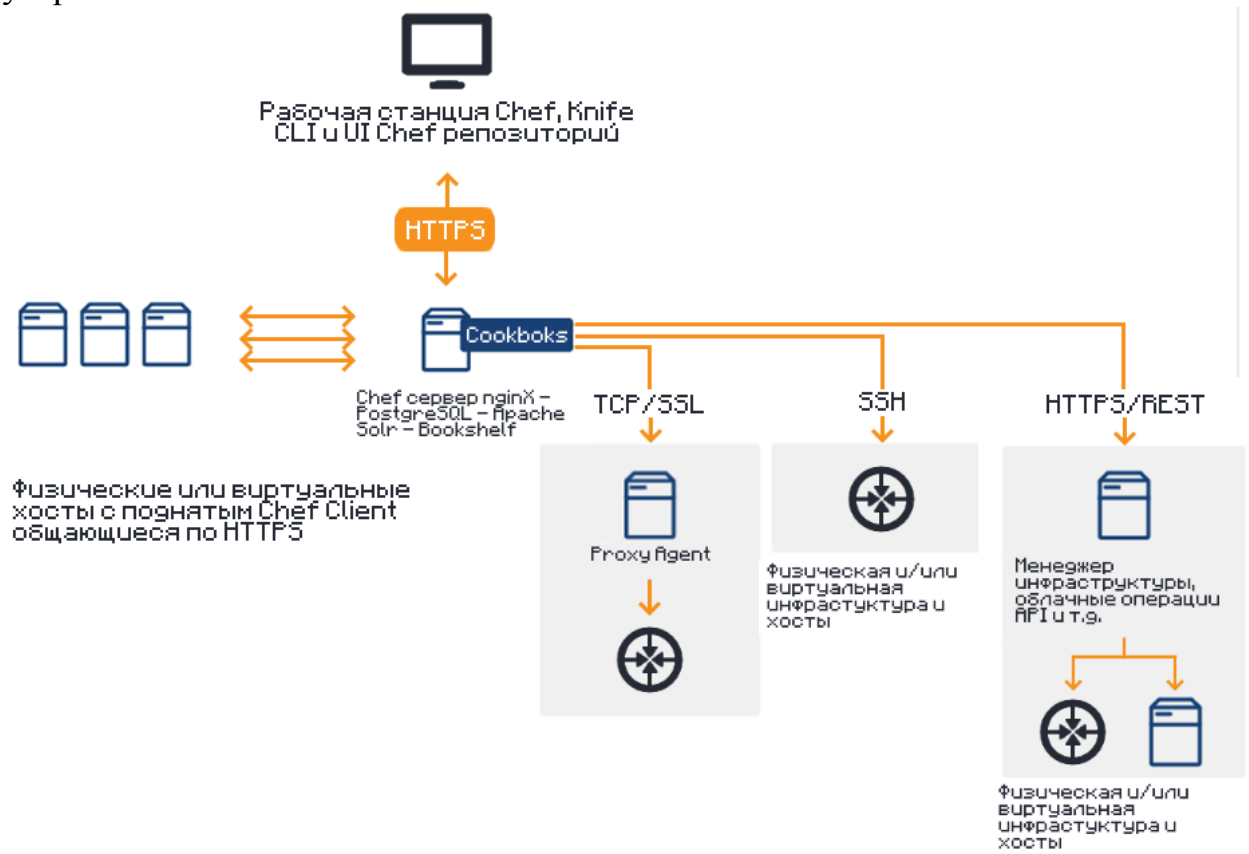


Рисунок 91 — Архитектура Chef

Компоненты рабочего места оператора Chef

Рабочая станция оператора Chef предоставляет:

– Инструменты командной строки для создания, тестирования и поддержки "поваренных книг" и применения их непосредственно на узлах.
– Взаимодействие с Chef Infra Servers для загрузки новых узлов и установки политики
– Test Kitchen, который представляет собой тестовый жгут
– ChefSpec, который моделирует эффекты кода перед внедрением изменений
– InSpec — система аудита и тестирования безопасности/соответствия нормативным требованиям.

Chef предоставляет сотни ресурсов для выполнения общих задач конфигурирования идемпотентным способом, а также Chef Supermarket,

поддерживаемый сообществом сайт обмена Cookbooks, пользовательскими ресурсами и другими решениями.

Код хранится в локальном репозитории, называемом chef-repo, который может быть синхронизирован с Git'ом для работы над инфраструктурой в масштабах предприятия. Внутри репозитория код организован в папки "cookbook", включающие "рецепты" (фактический код линейной конфигурации, написанный на расширенном Ruby синтаксисе Chef), отделенные атрибуты, пользовательские ресурсы, тестовый код и метаданные.

Доменно-специфический язык (DSL) Chef позволяет решать задачи конфигурации путем создания последовательности небольших шаблонов со скобками, каждый из которых объявляет ресурс и параметризует его. Ресурсы Chef имеют тенденцию быть более абстрактными, чем ресурсы Ansible или Puppet, что помогает решать кроссплатформенные задачи. Например, ресурс пакета может определить тип среды Linux, MacOS или Windows, в которой он работает, и выполнить необходимую установку с учетом особенностей платформы.

Установка Chef Workstation

Чтобы начать использовать Chef, хорошим первым шагом будет установка Chef Workstation, которая обеспечивает полную операционную среду. Workstation доступна для Linux и Windows. Для получения дополнительной информации обратитесь к странице загрузки Chef Workstation.

Когда Workstation установлена, вы можете сразу же использовать ее, чтобы начать вносить изменения в конфигурацию на доступных узлах. Прежде чем пытаться управлять целевым узлом с помощью Chef, полезно провести некоторую подготовку узла. Вы должны настроить SSH-ключ для доступа к узлу, а не использовать пароли. Также полезно (если вы не используете DNS) включить IP-адрес и имя хоста целевой машины в файл /etc/hosts вашей рабочей станции.

Запуск Chef в масштабе

Chef Infra Server был переписан несколько лет назад на языке Erlang, чтобы увеличить его возможности, позволяющие управлять примерно 10 000 хостов. Он может быть настроен на высокую доступность путем развертывания своих внешних сервисов (включая NGINX и логику приложений без статических данных) в массиве прокси-серверов с балансировкой нагрузки, которые подключаются к трехсерверному активному/активному кластеру, поддерживающему внутренние сервисы, такие как elasticsearch, etcd и PostgreSQL.

Chef также предлагает ряд продуктов, которые вместе решают большинство проблем, возникающих у предприятий при работе со все более сложными, масштабными, гибридными инфраструктурами. Его структуры chef-репо на рабочей станции гармонично сочетаются с Git, обеспечивая удобный контроль версий и совместную работу над кодом DevOps, а также упрощая переход к режимам infra-as-code. Его основная философия

непрерывного управления конфигурацией хорошо сочетается с целью непрерывного предоставления ИТ.
Встроенная в Chef система модульного тестирования Test Kitchen, симуляторы предварительного развертывания, а также сопутствующая система аудита и оценки безопасности InSpec предоставляют все остальные возможности специально разработанной системы разработки DevOps на основе тестирования.

Infrastructure as Code

Зачем хранить инфраструктуру как код?

На этом этапе пришло время ввести новый термин: **неизменяемость**. Это буквально означает "состояние неизменности", но на языке DevOps это означает поддержание систем полностью в виде кода, без каких-либо ручных операций над ними.

В этих темах несколько раз затрагивалась концепция отношения к инфраструктуре как к коду. Но до сих пор речь шла в основном о механике. Вы знаете, что имеет смысл автоматизировать развертывание полных стеков, которые представляют собой виртуальные инфраструктуры (вычислительные системы/хранилища/сети) плюс приложения. Мы познакомились с несколькими подходами к написанию базового кода автоматизации, с механикой инструментов автоматизации, а также с безопасным хранением кода и извлечением его из репозиториев контроля версий.

Теперь мы знакомы с идеей идемпотентности и смежными темами и видели, как можно составить код автоматизации, который очень безопасен для выполнения. Это код, который не ломает вещи, а вместо этого исправляет их и сходится к желаемому состоянию, описанному (частично или полностью) декларативной моделью развертываемой системы. Мы также узнали, как подобный код можно использовать для ускорения работы, решая проблемы методом грубой силы вместо детальной экспертизы и ошибочных, отнимающих много времени ручных операций.

GitOps: современная инфраструктура как код

Принятие обязательств по неизменяемости позволяет вам обращаться с кодовой базой автоматизации так же, как и с любым другим кодом приложения:

- Вы можете доверять тому, что кодовая база описывает то, что на самом деле работает на пустых металлических или облачных серверах.
- Вы можете управлять кодовой базой с помощью процедур Agile и структурированного использования контроля версий для сохранения ясности, и простоты.

Это и есть "GitOps", также называемый "operations by pull request.". В типичной схеме GitOps вы можете поддерживать репозиторий, например,

частный репозиторий на GitHub, с несколькими ветками, называемыми "Разработка", "Тестирование/UAT" и "Production".

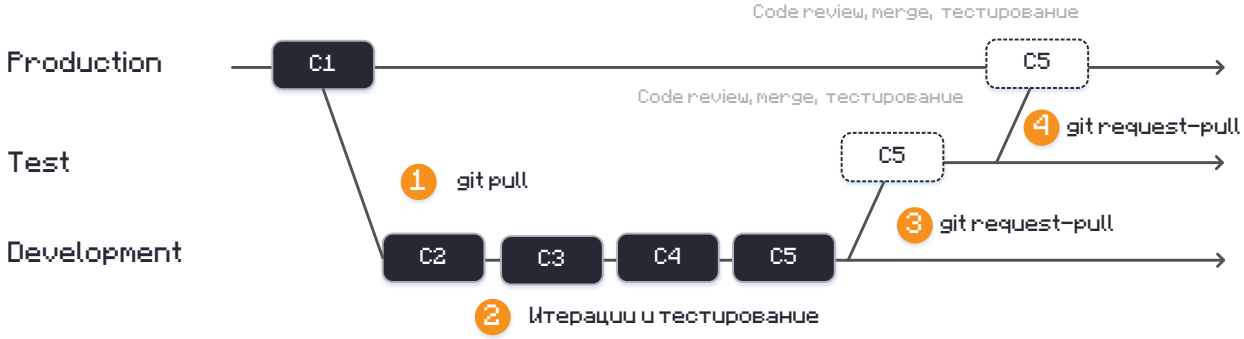


Рисунок 92 — GitOps это "Operations by Pull Request"

В этой модели каждая ветвь развертывается на отдельной инфраструктуре.

- **Development** — Разработчики вносят изменения в dev-ветку, оформляя коммиты и делая запросы на вытягивание. Эти запросы выполняются автоматизированным процессом стробирования и ставятся в очередь на автоматизированное тестирование. Операторы видят результаты автоматизированного тестирования, а разработчики проводят итерации до тех пор, пока тесты не пройдут. Затем изменения сливаются в ветку Test для следующего уровня проверки.
- **Test** — Когда изменения сливаются из ветки Development в Test, код развертывается в более крупной тестовой среде и подвергается более обширному набору автоматизированных тестов.
- **Production** — Протестированные изменения снова рассматриваются и сливаются в Production, откуда производится развертывание релиза.

К моменту фиксации, оценки, итераций и слияния тестируемого кода в ветку Production он проходит как минимум два полных цикла тестирования на последовательно более похожей на прод инфраструктуре, а также подвергается сознательной оценке со стороны нескольких экспертов. В результате получается код, который достаточно свободен от ошибок и хорошо работает. Эта операционная модель может быть использована для разработки широких возможностей самообслуживания и самовосстанавливающихся вычислительных систем/хранилищ/сетей/PaaS или для реализации крупномасштабных распределенных приложений с развитой контейнерной оркестрации.

Куда может привести вас GitOps?

Когда процедуры и рабочий процесс GitOps, gating/CI-CD, автоматизированное тестирование и другие компоненты внедрены, вы можете начать экспериментировать с элитными стратегиями развертывания, которые требуют безупречной автоматизации.

Blue/green deployments

Blue/green deployments — это метод сокращения или устранения времени простоя в производственных средах. Он требует поддержания двух

идентичных производственных сред (необязательно называть их синей и зеленой. Подойдут любые два цвета, например, красный и черный). Разработайте возможность быстрого перенаправления трафика приложений на ту или иную среду (с помощью автоматизации ACI, балансировки нагрузки, программируемого DNS или других средств).

Релиз развертывается в среде, которая в настоящее время не используется (зеленый цвет). После приемочного тестирования перенаправьте трафик на эту среду. Если возникнут проблемы, можно переключить трафик обратно на исходную среду (синюю). Если "зеленое" развертывание будет признано адекватным, ресурсы, принадлежащие "синему" развертыванию, могут быть освобождены, а роли поменяны местами для следующего релиза.

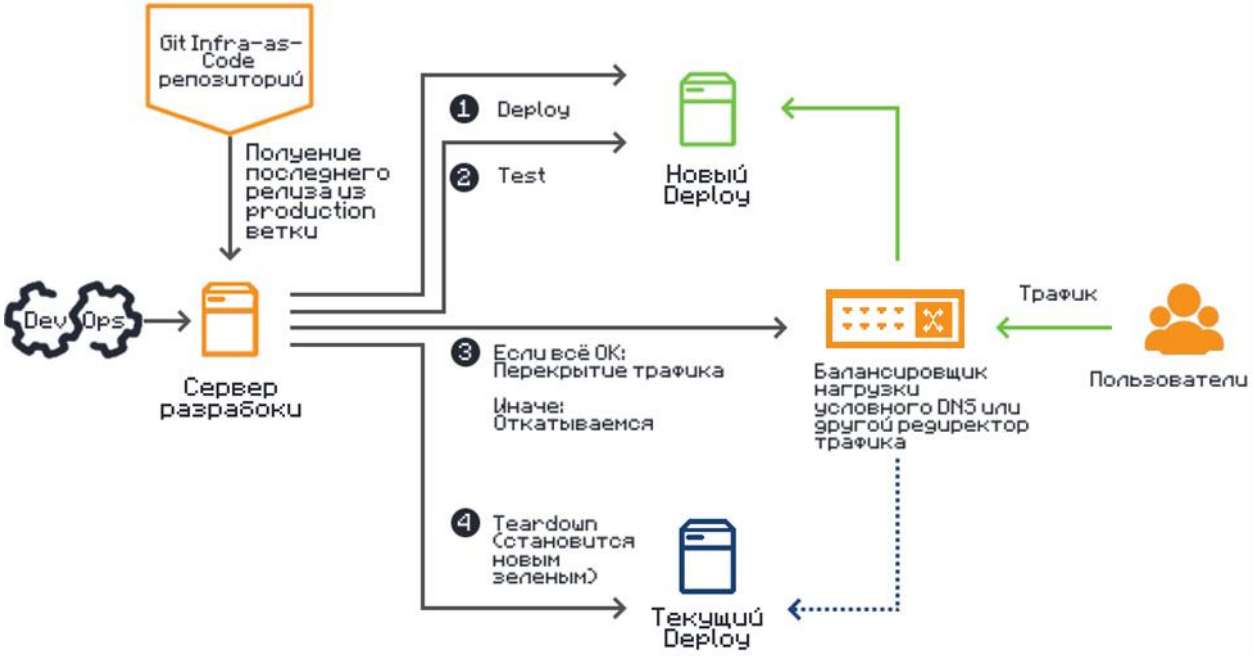


Рисунок 93 — Blue/green деплой

***Примечание**: Некоторые специалисты по DevOps различают blue/green и red/black стратегии. Они говорят, что в blue/green стратегии трафик постепенно переводится из одной среды в другую, так что в течение некоторого периода он попадает в обе системы; в то время как в red/black стратегии трафик отключается сразу. Некоторые продвинутые специалисты по DevOps, например, Netflix, практикуют версию red/black на уровне сервера, которую они называют "rolling red/black". При использовании "скользящего красного/черного" серверы в обеих средах постепенно обновляются, в итоге получается одна версия, отличная друг от друга, и ее также можно откатить по отдельности. Это означает, что в любой момент времени в обеих средах работают три (а не две) версии приложения или стека.*

Canary deployment

Canary deployment похоже на скользящее развертывание "blue/green", но несколько более деликатно. Миграция между старыми и новыми развертываниями осуществляется на основе каждого клиента (или даже пользователя), и миграции делаются намеренно, чтобы снизить риск и улучшить качество обратной связи. Некоторые клиенты могут быть очень заинтересованы в получении доступа к новым функциям и будут благодарны за предоставленную возможность. Они с радостью предоставят отзывы о релизах по эффективным каналам (Рисунок 93):

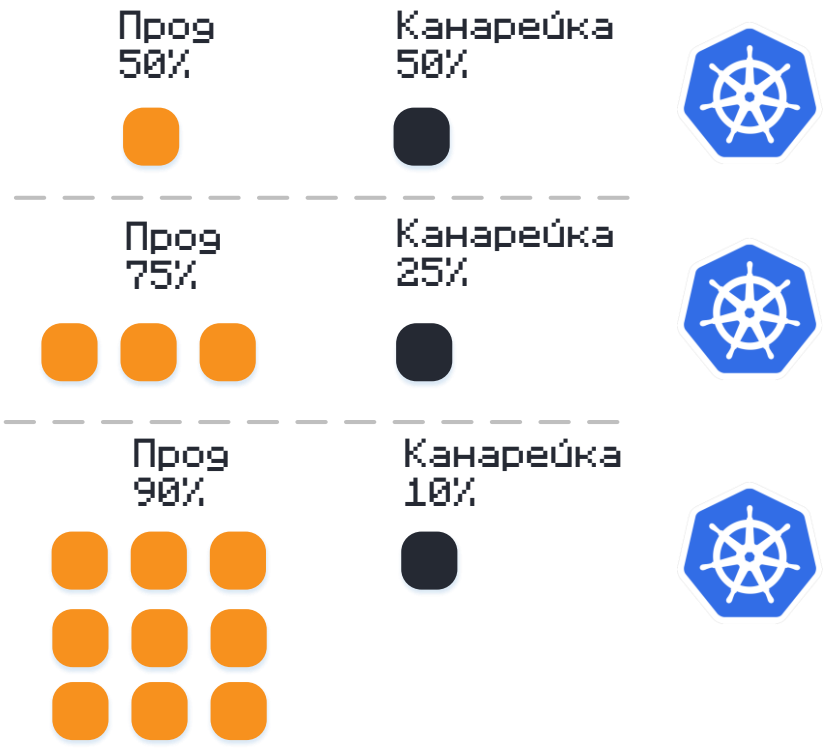


Рисунок 94 — Canary деплой

Автоматизация тестирования

Автоматизированное тестирование и валидация

В этой теме мы рассмотрим набор продуктов и практик, созданных компанией Cisco и ее сообществом пользователей для распространения автоматизации тестирования на программно-управляемую конфигурацию сети, а также для уменьшения или устранения неопределенности в отношении того, как будут функционировать и работать перспективные сетевые архитектуры после их полного внедрения.

Проблемы инфраструктуры тестирования

Инструменты автоматизации, такие как Ansible, Puppet, Chef и другие, частично решают эту проблему, превращая инфраструктуру в код. Но DevOps обычно нуждается в более тонких способах определения и внедрения инфраструктур, подтверждения того, что развернутые инфраструктуры работают так, как нужно, и проактивного обеспечения их бесперебойной работы. DevOps также нужны способы упреждающего принятия мер при

неизбежных сбоях, а также поиска и устранения проблем при возникновении ошибок.

При использовании инструментов модульного тестирования, таких как pytest, в тандеме с автоматизацией более высокого порядка и в сочетании с непрерывной доставкой (CI/CD), вы можете создать среду, в которой код будет автоматически тестироваться при внесении изменений.

Фреймворки модульного тестирования делают тесты частью вашей кодовой базы, следуя за кодом через коммиты разработчиков, запросы на перенос и ворота code-review до QA/test и Production. Это особенно полезно в средах разработки, управляемой тестами (TDD), где написание тестов — это непрерывный процесс, который фактически ведет разработку, автоматически поощряя очень высокий уровень покрытия тестами.

Трудности тестирования сети

Поведение и производительность реальной сети являются коллективными, поддерживаемыми конфигурациями многих отдельных частей оборудования и программного обеспечения.

В традиционных средах подключение и функциональность поддерживаются вручную на множестве отдельных единиц оборудования через различные интерфейсы. Часто операции требуют внесения изменений в работающее оборудование, передающее живой трафик. Это сложно, отнимает много времени, крайне опасно и чревато ошибками.

Неправильная конфигурация сети часто обнаруживается только косвенно, когда компьютеры не могут разговаривать друг с другом. Поскольку сети становятся более сложными и передают более разнообразный и чувствительный к производительности трафик, риски для безопасности и снижение производительности, которые трудно обнаружить и оценить количественно, становятся все более важными последствиями неправильной конфигурации.

Управление и тестирование сетей по-прежнему сложны, даже когда сетевые устройства и возможности подключения становятся программно адресуемыми и виртуализированными. Методы построения и конфигурирования сетей, конечно, меняются, но вам по-прежнему необходимо создавать коллективную архитектуру для безопасного подключения, детально затрагивая многочисленные интерфейсы устройств.

Тестирование программно-определяемых сетей (SDN)

Компания Cisco добилась огромных успехов в разработке программно-определяемых сетей (SDN) и промежуточного программного обеспечения, которые позволяют инженерам рассматривать физический сетевой коллектив как единый программируемый объект. В случае Cisco это включает в себя:

- Application Centric Infrastructure (ACI) — комплексное решение для центров обработки данных, которое работает поверх Nexus 9000 и устройств с поддержкой APIC. Оно позволяет абстрагировать и

программировать всю сетевую функциональность с помощью контроллера инфраструктуры политики приложений (APIC).
- Digital Network Architecture Center (Cisco DNA Center) — открытая, расширяемая, управляемая программным обеспечением архитектура для Catalyst 9000 и других устройств IOSXE для корпоративной сети.
- REST API и SDK обеспечивают интеграцию с такими инструментами автоматизации, как Ansible, Puppet и Chef.

Решения, подобные ACI, управляют всей сетью путем объединения моделей (часто написанных на декларативном синтаксисе, называемом YANG), которые представляют желаемые состояния функциональности и подключения. Промежуточное программное обеспечение позволяет модели гармонично работать с другими моделями, которые в настоящее время определяют состояние системы и требования к ресурсам. Инженеры реже взаимодействуют непосредственно с отдельными устройствами, хотя модели по-прежнему должны предоставлять достаточно подробную информацию для конфигурирования. Сложное, быстро меняющееся состояние больших инфраструктур может поддерживаться в виде кода, что позволяет:
- Быстрое преобразование в желаемые состояния при необходимости. Если устройство ломается и заменяется, его можно быстро реинтегрировать в существующую сеть и быстро восстановить его запрограммированную функциональность, а также поведение и производительность сети.
- Переносимость, так что, когда основное приложение перемещается из одного центра обработки данных в другой, его требуемая сетевая конфигурация сопровождает его.
- Контроль версий, CI/CD и другие инструменты для поддержания, развития и применения кодовой базы сети.

Эти инновации становятся все более популярными среди крупных организаций, операторов связи и других масштабных структур. Однако во многих случаях сети все ещё состоят из нескольких поколений разнообразных, гибридных, многовендорных физических и виртуальных инфраструктур, поэтому проблема преднамеренной конфигурации на уровне устройств все еще остается актуальной.

И даже при наличии сложной SDN контроллеры/организаторы SDN не всегда могут предотвратить неправильную конфигурацию. Они могут отклонить некачественный код, выполнить проверку на вменяемость перед применением изменений и сообщить, когда модели предъявляют требования, превышающие пороговые значения ресурсов, но кажущиеся легитимными изменения всё равно могут быть применены.

Решение для тестирования сети: pyATS

Python Automated Test System (pyATS) — это решение для тестирования и проверки сетевых устройств на языке Python, изначально разработанное компанией Cisco для внутреннего использования, затем ставшее общедоступным и частично открытым. pyATS можно использовать для проверки работоспособности ваших изменений до их внедрения в производство, а также для продолжения проверки и мониторинга в производстве для обеспечения бесперебойной работы.

Экология pyATS

pyATS зародился как низкоуровневая основа Python для системы тестирования в целом. Его библиотечная система более высокого уровня, Genie, предоставляет необходимые API и библиотеки, которые управляют сетевыми устройствами и взаимодействуют с ними, а также выполняют фактическое тестирование. Эти два компонента вместе образуют тестовое решение Cisco, которое мы знаем, как pyATS.

pyATS имеет несколько ключевых особенностей:

- Фреймворк и библиотеки pyATS могут быть использованы в любом коде Python.
- Она является модульной и включает такие компоненты, как:
    1. AEtest выполняет тестовые сценарии.
    2. Easypy — механизм выполнения, который обеспечивает параллельное выполнение нескольких сценариев, собирает журналы в одном месте и обеспечивает центральную точку, из которой можно вносить изменения в тестируемую топологию.
- CLI позволяет быстро опрашивать живые сети, извлекать факты и автоматизировать выполнение тестовых сценариев и других экспертиз. Это позволяет очень быстро отлаживать и исправлять проблемы в сетевых топологиях, созданных и поддерживаемых с помощью этих инструментов.

В SDN/облачных/виртуальных сетевых средах настройка может включать в себя фактическое построение топологии, а очистка — её удаление, повторное использование ресурсов платформы. Эта настройка и очистка может быть выполнена непосредственно с помощью кода pyATS. pyATS предоставляет огромную библиотеку интерфейсов к Cisco и другим инфраструктурам через ряд интерфейсов, включая низкоуровневые CLI и REST API, а также коннекторы к ACI и другим фреймворкам управления SDN более высокого порядка.

pyATS может использовать, анализировать и реализовывать топологии, описанные в JSON, в виде моделей YANG и из других источников, даже из электронных таблиц Excel.

pyATS также может быть интегрирован с инструментами автоматизации, такими как Ansible, для построения, обеспечения и разрушения. Однако лучшей практикой может быть обратная. Используйте

Ansible, Puppet или Chef для управления всей кодовой базой инфраструктуры, и пусть эти продукты вызывают Python (и pyATS) для решения деталей реализации сети. Эти инструменты также привлекают ACI или другое промежуточное ПО для упрощения задачи и позволяют раздельное хранение и версионирование YANG или других моделей, определяющих конкретные топологии.

Кроме того, вы можете вызвать pyATS косвенно несколькими способами (включая способы, требующие минимальных знаний программирования на Python).

Сетевое моделирование

Сетевое моделирование и VIRL

Моделирование сети предоставляет средства для тестирования сетевых конфигураций, отладки кода конфигурации, а также для работы и изучения инфраструктуры и API Cisco безопасным, удобным и не требующим больших затрат способом.

Cisco Virtual Internet Routing Laboratory (VIRL, произносится как "вайрал") — это коммерческий продукт, изначально разработанный для внутреннего использования в компании Cisco при широкой и активной поддержке сообщества. Теперь, в версии 2, VIRL может работать на "голом" металле или на больших виртуальных машинах на нескольких платформах гипервизоров (среди них ESXi и VMware Workstation 12+). Официальное название VIRL v 2.0 - Cisco Modeling Labs — Personal (или CML-Personal). Вы можете встретить упоминание инструмента как VIRL 2.0, так и CML - P.

Хотя VIRL не может дублировать производительность элитного оборудования или программных компонентов SDN, работающих в оптимальных производственных условиях, он идеально отражает функциональность Cisco. Виртуальное оборудование, работающее в VIRL, использует тот же код, который работает в реальных продуктах Cisco. Это делает VIRL идеальным инструментом для обучения, а также полезным механизмом для тестирования сетевых конфигураций и тонкой настройки автоматизации их создания и тестирования.

Компоненты VIRL и рабочий процесс

VIRL предоставляет локальный CLI для управления системой, REST-интерфейс для интеграции с автоматизацией, а также мощный пользовательский интерфейс, который предлагает полную графическую среду для построения и конфигурирования топологий моделирования.

Пользовательский интерфейс поставляется с несколькими образцами топологий для начала работы. Среди них — симуляция сети IOS с двумя маршрутизаторами, которую можно быстро сделать активной и изучить. Представление VIRL Design Perspective позволяет изменять существующие симуляции (после их остановки) или создавать новые, перетаскивая, опуская и соединяя сетевые объекты, конфигурируя их по ходу работы (Рисунок 95).

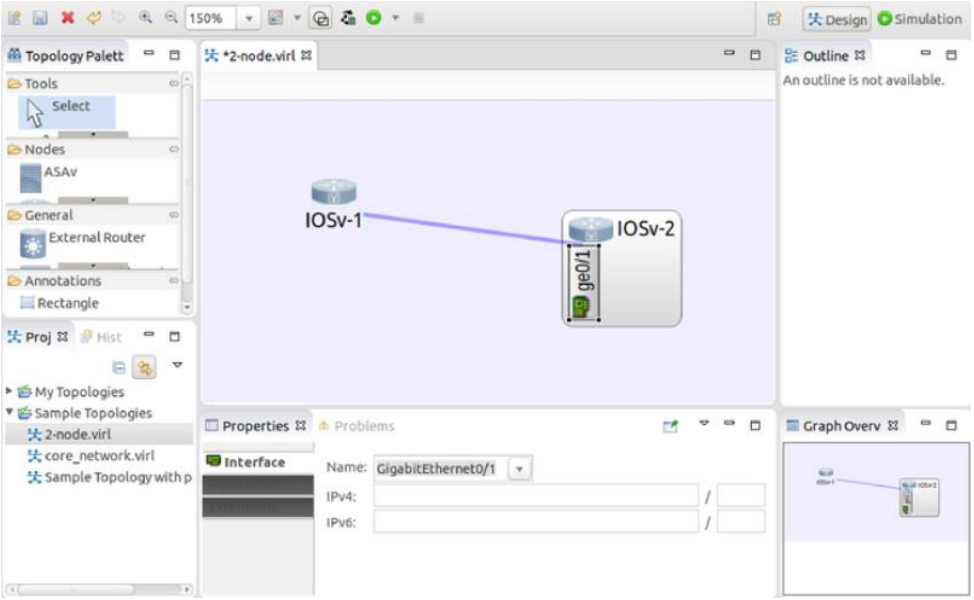


Рисунок 95 — VIRL демонстрирует визуализацию вашей симуляции

Визуализация имеет кликабельные элементы, которые позволяют исследовать конфигурацию устройств и вносить изменения через WebUI или подключаясь к элементам сети через консоль. Вы также можете извлечь конфигурации отдельных устройств или целые конфигурации моделируемой сети в виде файлов .virl.

VIRL файлы

VIRL также позволяет определять симуляции как код, что обеспечивает двустороннюю интеграцию с другими программными платформами для управления сетью и тестирования.

Родной формат конфигурации VIRL называется файлом .virl, который представляет собой человекочитаемый файл YAML. Файл .virl содержит полное описание маршрутизаторов IOS, конфигурации их интерфейсов и подключения (плюс другую информацию о конфигурации), учетные данные для доступа к ним и другие подробности. Эти файлы можно использовать для запуска симуляций через VIRL REST API, и вы можете конвертировать файлы .virl в файлы "тестовой площадки" и обратно для использования с PyATS и Genie.

В пользовательском интерфейсе VIRL вы выбираете симуляцию, заставляете VIRL прочитать конфигурацию устройства, а затем он составляет .virl-файл для его представления. VIRL предлагает сохранить топологию в новом файле, который затем можно открыть в редакторе для просмотра.

Файл .virl предоставляет метод определения того, произошло ли смещение конфигурации при моделировании. Простая команда diff может сравнить только что извлеченный файл .virl с оригинальным файлом .virl, использованным для запуска моделирования, и различия будут очевидны.

Эта техника сравнения известного конфигурационного манифеста с извлеченным манифестом, описывающим текущее состояние сети, помогает

отлаживать реальные сети, для которых доступны авторитетные, полные топологии PyATS.

Резюме

DevOps и SRE

Чтобы full-stack автоматизация была действительно эффективной, она требует изменений в организационной культуре, включая разрушение исторически сложившихся границ между отделами разработки (Dev) и эксплуатации (Ops). DevOps развивался и продолжает развиваться во многих местах параллельно. Некоторые ключевые события сформировали эту дисциплину в том виде, в котором мы знаем ее сегодня.

Ключевой момент 1: Site Reliability Engineering (SRE). Роль SRE призвана объединить дисциплины и навыки Dev и Ops.

Ключевой момент 2: Дебуа и "Agile Infrastructure" Дебуа был сторонником автоматизации виртуальной инфраструктуры, использования контроля версий для хранения кода развертывания инфраструктуры и применения методов Agile для разработки и обслуживания решений на уровне инфраструктуры.

Ключевой момент 3: Оллспоу и Хаммонд

Джон Оллспоу и Пол Хаммонд выступили с презентацией на VelocityConf в 2009 году. В ней описывались автоматизация, командная работа, разделение ответственности, прозрачность, доверие, взаимная подотчетность и практика коммуникаций.

DevOps/SRE имеют много основных принципов и лучших практик:

- Фокус на автоматизации
- Идея о том, что "Сбои — это нормально"
- Переосмысление понятия "доступность" с точки зрения того, что может выдержать бизнес

Основные сценарии автоматизации

Инструменты автоматизации частично работают за счет обертывания функциональности оболочки, утилит операционной системы, функций API и других элементов плоскости управления. Но инструменты все равно не решают всех проблем развертывания и конфигурирования. Поэтому в каждом инструменте автоматизации есть одна или несколько функций, которые выполняют основные команды и сценарии на объектах и возвращают результаты. Например, в Ansible к таким функциям относятся command, shell и raw. Такие инструменты автоматизации, как Ansible, Puppet или Chef, предлагают мощные возможности по сравнению с ситуативными стратегиями автоматизации с использованием BASH, Python или других языков программирования. Повелительная процедура — это упорядоченная последовательность команд, направленная на достижение цели. Последовательность может включать управление потоком, условия, функциональную структуру, классы и многое другое. Чтобы настроить

удаленные системы, необходимо получить к ним доступ и выполнить на них сценарии. Это можно сделать двумя способами (из нескольких): хранить скрипты локально, передавать их на целевые машины с помощью утилиты оболочки, например, scp, затем войти на удаленную машину с помощью ssh и выполнить их. Или вы можете передать сценарии на удаленную машину с помощью `cat | ssh` и выполнить их последовательно с другими командами, перехватывая и возвращая результаты на терминал одной командой. Облачные вычисления на базе инфраструктуры как услуги (IaaS) являются типичной целью для автоматизации. Автоматизация облачных вычислений позволяет вам предоставлять виртуализированные узлы, настраивать виртуальные сети и другие возможности подключения, запрашивать услуги, а затем развертывать приложения на этой инфраструктуре. IaaS и другие типы облачных инфраструктур также предоставляют CLI и SDK, позволяющие легко подключаться к их базовым интерфейсам, которые обычно основаны на REST.

Инструменты автоматизации

Три наиболее популярных инструмента автоматизации — это Ansible, Puppet и Chef. Такие инструменты автоматизации, как Ansible, Puppet или Chef, предлагают мощные возможности по сравнению с ситуативными стратегиями автоматизации с использованием BASH, Python или других языков программирования. Идемпотентное программное обеспечение дает один и тот же желаемый результат при каждом запуске. В программном обеспечении для развертывания идемпотентность обеспечивает конвергенцию и композитивность. Процедурный код может обеспечить идемпотентность, но многие инструменты управления инфраструктурой, развертывания и оркестровки используют другой метод — создание декларативного кода. Декларатив — это статическая модель, представляющая желаемый конечный продукт.

Infrastructure as Code

Неизменяемость буквально означает "состояние неизменности", но в языке DevOps это означает поддержание систем полностью в виде кода, без выполнения каких-либо ручных операций над ними. Принятие обязательств по неизменяемости позволяет вам обращаться с кодовой базой автоматизации так же, как с любым кодом приложения:

- Вы можете доверять тому, что кодовая база описывает то, что на самом деле работает на пустом металле или облачных серверах.
- Вы можете управлять кодовой базой с помощью Agile-процедур и структурированного использования контроля версий, чтобы все было ясно и просто.

Автоматизация тестирования

DevOps обычно нуждается в более тонких способах определения и внедрения инфраструктур, подтверждения того, что развернутые инфраструктуры работают так, как нужно, проактивного обеспечения их

бесперебойной работы, упреждающего принятия мер при неизбежных сбоях, а также поиска и устранения проблем при возникновении ошибок.

Если использовать инструменты модульного тестирования, такие как pytest, в тандеме с автоматизацией более высокого порядка и в сочетании с непрерывной доставкой (CI/CD), можно создать среду, в которой код будет автоматически тестироваться при внесении изменений.

Фреймворки модульного тестирования делают тесты частью вашей кодовой базы, следуя за кодом через коммиты разработчиков, запросы на перенос и ворота code-review до QA/test и Production. Это особенно полезно в средах разработки, управляемой тестами (TDD), где написание тестов - это непрерывный процесс, который фактически ведет разработку, автоматически поощряя очень высокий уровень покрытия тестами.

Не вошедшая Глава 6 — Сетевые технологит

Вступление

Зачем разработчику знать о сети, через которую проходит весь трафик приложений, а также между ними? Зачем сетевому инженеру, который уже знаком со сложными деталями и архитектурными решениями для сетевого трафика, знать о разработке приложений? Одно без другого не позволяет выполнить работу, когда речь идет об управлении и развертывании масштабных приложений, удовлетворяя при этом требованиям и ожиданиям по производительности. Давайте “заложим” фундамент сети и изучим чуть больше чем, только основы.

Введение в основы сетевых технологий

Пользователи сети просто хотят, чтобы она работала. Разработчики, обычно, более любопытны и часто готовы сами устранять проблемы с подключением. Сетевые администраторы извлекают пользу из методов, которые автоматически и программно управляют и развертывают сетевые конфигурации, включая сценарии "нулевого дня".

Производительность важна для всех, независимо от их точки зрения. С помощью автоматизации можно ускорить развертывание. С помощью мониторинга приложений вы сможете быстрее устранять неполадки. Знание способов устранения неисправностей сетевого подключения крайне важно, как для разработчиков, так и для администраторов, поэтому быстрое решение проблем имеет решающее значение для всех.

В этой теме рассматриваются фундаментальные элементы сети. Необходимо понять, какие стандарты используются для сетей, чтобы убедиться, что у вас есть необходимый словарный запас для обсуждения сетевых проблем и решений не только с любым членом команды, но и в приличном обществе. Четкое понимание уровней, через которые проходит сетевой трафик, позволит вам получить знания, необходимые для работы с сетями, приложениями и автоматизацией.

Что такое Сеть?

Сеть состоит из конечных устройств, таких как компьютеры, мобильные устройства и принтеры. Эти устройства соединены сетевыми устройствами, коммутаторами и маршрутизаторами. Сеть позволяет устройствам общаться друг с другом и обмениваться данными. Существует множество способов подключения к сети. Наиболее распространенными способами подключения к локальной сети (LAN), определенными Институтом инженеров по электротехнике и электронике (IEEE), являются проводные локальные сети Ethernet (IEEE 802.3) и беспроводные локальные сети (IEEE 802.11). Конечные устройства подключаются к сети с помощью сетевой интерфейсной карты Ethernet или беспроводной сети (NIC).

Сетевые карты Ethernet подключаются к сети через Registered Jack 45 (RJ-45) и кабели витой пары Ethernet. Беспроводные сетевые карты подключаются к сети с помощью беспроводных радиосигналов в частотном диапазоне 2,4 ГГц или, чаще всего, 5 ГГц.

Наборы протоколов

Набор протоколов — совокупность протоколов, которые работают вместе для предоставления комплексных услуг сетевой связи. С 1970-х годов существовало несколько различных наборов протоколов, некоторые из которых были разработаны организацией по стандартизации, а другие — различными поставщиками. Во время развития сетевых коммуникаций и Интернета существовало несколько конкурирующих наборов протоколов:

- **Internet Protocol Suite** или **TCP/IP** — *модель протокола Transmission Control Protocol/Internet Protocol (TCP/IP) для сетевых коммуникаций была создана в начале 1970-х годов и иногда называется моделью Интернета. Это наиболее распространенный и актуальный набор протоколов, используемый сегодня. Набор протоколов TCP/IP — это открытый стандартный набор протоколов, поддерживаемый рабочей группой по разработке Интернета (IETF);*
- **Open Systems Interconnection (OSI)** протоколы — *семейство протоколов, разработанных совместно в 1977 году Международной организацией по стандартизации (ISO) и Международным союзом электросвязи (ITU). Протоколы OSI включают семиуровневую модель, называемую эталонной моделью OSI. Эталонная модель OSI классифицирует функции протоколов. Сегодня OSI известна в основном благодаря своей многоуровневой модели. Протоколы OSI в значительной степени были заменены TCP/IP;*
- **AppleTalk** — *недолговечный пакет проприетарных протоколов, выпущенный компанией Apple Inc. в 1985 году для устройств Apple. В 1995 году Apple приняла TCP/IP вместо AppleTalk;*
- **Novell NetWare** — *недолговечный пакет проприетарных протоколов и сетевая операционная система, разработанная компанией Novell Inc. в 1983 году с использованием сетевого протокола IPX. В 1995 году Novell приняла TCP/IP для замены IPX.*

Сегодня для описания работы сети используются модель OSI и модель TCP/IP, показанные на рисунке.

На рисунке 96 показана модель OSI, а справа — модель TCP/IP. Модель OSI обозначена сверху вниз цифрами от 7 до 1 и следующими словами на каждом уровне: прикладной, презентационный, сеансовый, транспортный, сетевой, канальный и физический. Три верхних уровня модели o s i находятся напротив прикладного уровня модели TCP/IP. Транспортные уровни каждой модели находятся друг напротив друга. Сетевой уровень модели o s i находится справа напротив уровня Интернета. Уровни 1 и 2 модели OSI находятся напротив уровня доступа к сети модели TCP/IP.

Рисунок 96 — Модели OSI и TCP/IP

Модели OSI и TCP/IP используют уровни для описания функций и услуг, которые могут быть реализованы на данном уровне. Эти модели обеспечивают согласованность всех типов сетевых протоколов и служб, описывая, что должно быть сделано на определенном уровне, но не предписывая, как это должно быть сделано. Также описывается взаимодействие каждого уровня со слоями, расположенными непосредственно выше и ниже.

Обе модели могут использоваться со следующими различиями:

- Модель OSI нумерует каждый уровень;
- В модели TCP/IP используется один прикладной уровень для обозначения прикладного, презентационного и сеансового уровней OSI;
- В модели TCP/IP используется один уровень доступа к сети для обозначения канального и физического уровней OSI;

– В модели TCP/IP сетевой уровень OSI называется уровнем Интернета.

Рисунок 97 — OSI Уровень Передача данных

Форма, которую принимает часть данных на любом уровне, называется единицей данных протокола (PDU). В процессе инкапсуляции каждый последующий уровень инкапсулирует PDU, который он получает от вышележащего уровня, в соответствии с используемым протоколом. Когда сообщения передаются по сети, процесс инкапсуляции происходит сверху вниз, как показано на рисунке 98.

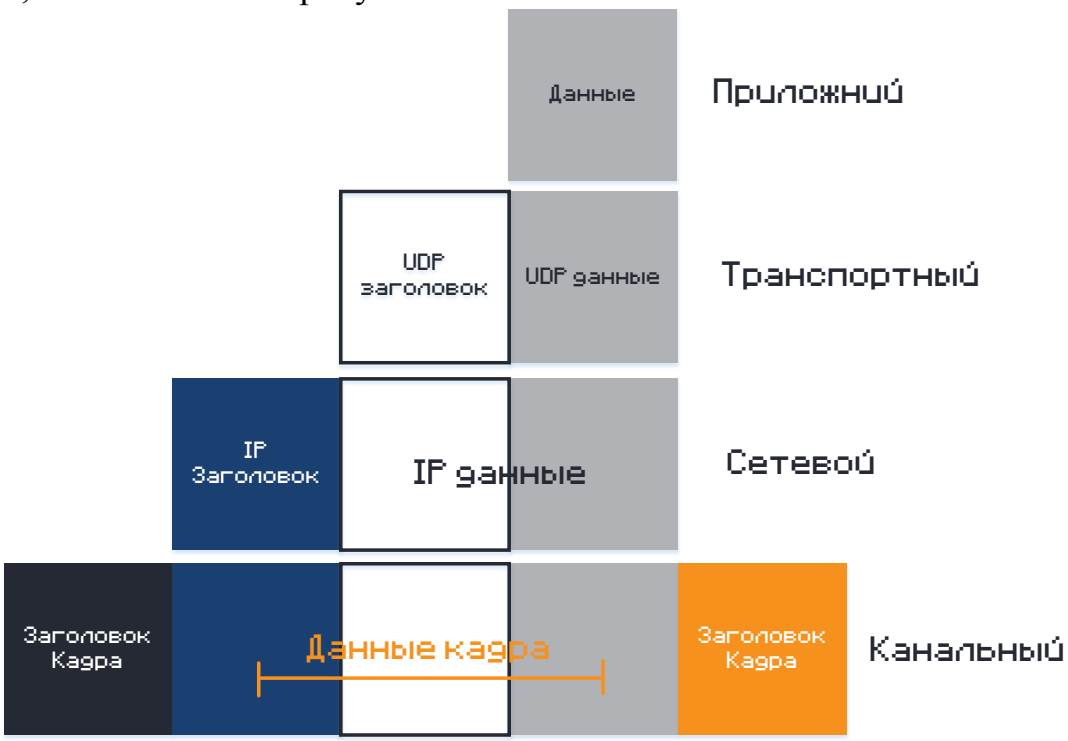


Рисунок 98 — Инкапсуляция данных на каждом уровне модели TCP/IP

На каждом этапе процесса PDU, *напомню Protocol Data Unit — обобщённое название фрагмента данных на разных уровнях модели OSI*, имеет разные имена, отражающее его новые функции. Как правило, PDU называются в соответствии со следующими уровнями:

– **Данные** — общий термин для PDU, используемых на прикладном уровне;
– **Сегмент** — PDU транспортного уровня;

- **Пакет** — PDU сетевого уровня;
- **Кадр** — PDU канального уровня;
- **Биты** — PDU физического уровня, используемый при физической передаче данных по среде передачи данных.

На каждом уровне информация верхнего уровня считается данными внутри инкапсулированного протокола. Например, **сегмент** транспортного уровня считается данными в **пакете** интернет-уровня. Затем **пакет** считается данными в **кадре** канального уровня.

Преимуществом многоуровневого процесса передачи данных является абстракция, которая может быть реализована с его помощью. Для каждого уровня могут быть разработаны различные протоколы, которые можно менять по мере необходимости. До тех пор, пока протокол обеспечивает функции, ожидаемые вышележащим уровнем, его реализация может быть абстрагирована и скрыта от других уровней. Абстракция протокола и услуг в этих моделях осуществляется посредством инкапсуляции.

В общем, приложение использует набор протоколов для передачи данных от одного узла к другому. Проходя по уровням, от верхнего до нижнего на отправляющем узле, а затем обратным путем от нижнего уровня до верхнего на принимающем узле, на каждом уровне происходит инкапсуляция данных.

На каждом уровне протоколы выполняют функции, собственно необходимые для данного уровня. Ниже приведу эту функциональность для каждого уровня модели OSI.

Физический уровень (уровень 1)

Этот уровень отвечает за передачу и прием необработанных битовых потоков. На этом уровне передаваемые данные преобразуются в электрические, радио- или оптические сигналы. Спецификации физического уровня определяют уровни напряжения, физическую скорость передачи данных, схему модуляции, расположение контактов для кабельных разъемов, спецификацию кабеля и многое другое. Ethernet, Bluetooth и универсальная последовательная шина (USB) являются примерами протоколов, которые имеют спецификации для физического уровня.

Канальный уровень (уровень 2)

Этот уровень обеспечивает связь между сетевыми картами в одной сети. Спецификация канального уровня определяет протоколы для установления и прерывания соединений, а также управление передачей данных между двумя физически подключенными устройствами. IEEE разработала несколько протоколов для канального уровня. Семейство протоколов IEEE 802, включающее Ethernet и беспроводные локальные сети (WLAN), подразделяет этот уровень ещё на два подуровня:

- **Medium Access Control (MAC)** — *подуровень MAC отвечает за управление тем, как устройства в сети получают доступ к среде передачи и получают разрешение на передачу данных.*

– **Logical Link Control (LLC)** — *отвечает за идентификацию и инкапсуляцию протоколов сетевого уровня, контроль проверки ошибок и синхронизацию кадров. Протоколы IEEE 802.3 Ethernet, 802.11 Wi-Fi и 802.15.4 ZigBee работают на канальном уровне. Подуровень MAC на канальном уровне критически важен в широковещательных средах (таких как беспроводная передача), в которых управление средой передачи должно быть тщательно реализовано.*

Сетевой уровень (уровень 3)

Этот уровень предоставляет услуги адресации и маршрутизации, позволяющие конечным устройствам обмениваться данными в сети. IP версии 4 (IPv4) и IP версии 6 (IPv6) являются основными протоколами адресации сетевого уровня. Такие протоколы, как Open Shortest Path First (OSPF) и Border Gateway Protocol (BGP), предоставляют услуги маршрутизации.

Для осуществления сквозной передачи данных через границы сети протоколы сетевого уровня выполняют две основные функции:

– Адресация — все устройства должны быть настроены на уникальный IP-адрес для идентификации в сети.
– Маршрутизация — протоколы маршрутизации предоставляют услуги по направлению пакетов к узлу назначения в другой сети. Чтобы попасть в другую сеть, пакет должен быть обработан маршрутизатором. Роль маршрутизатора заключается в выборе наилучшего пути и пересылке пакетов к узлу назначения в процессе, известном как маршрутизация. Пакет может пересечь множество маршрутизаторов, прежде чем достигнет узла назначения. Каждый маршрутизатор, через который пакет проходит, чтобы достичь узла назначения, называется хопом.

Сетевой уровень также включает протокол Internet Control Message Protocol (ICMP) для предоставления услуг обмена сообщениями, например, для проверки соединения с помощью команды `ping` или обнаружения пути между источником и пунктом назначения с помощью команды `traceroute`.

Транспортный уровень (уровень 4)

Транспортный уровень определяет службы для сегментации, передачи и повторной сборки данных для индивидуальных коммуникаций между конечными устройствами. Этот уровень имеет два протокола: Transmission Control Protocol (TCP) и User Datagram Protocol (UDP).

TCP обеспечивает надежность и управление потоком с помощью этих основных операций:

– Нумерация и отслеживание сегментов данных, переданные определенному хосту от определенного приложения;
– Подтверждение получения данных;
– Повторная передача не подтвержденных данных через определенное время;

– Упорядочивание данных, которые могут поступать в неправильном порядке;
– Отправка данных с эффективной скоростью, приемлемой для получателя.

TCP используется в таких приложениях, как базы данных, веб-браузеры и почтовые клиенты. TCP требует, чтобы все отправляемые данные поступали в пункт назначения в исходном состоянии. Любые недостающие данные могут испортить сообщение, сделав его неполным или нечитаемым.

UDP — это более простой протокол транспортного уровня, чем TCP. Он не обеспечивает надежность и управление потоком, а значит, требует меньшего количества полей заголовка. Датаграмма UDP могут обрабатываться быстрее, чем сегменты TCP.

UDP предпочтительнее для таких приложений, как передача голоса по IP (VoIP). Подтверждение и повторная передача замедлят доставку и сделают голосовой разговор неприемлемым. UDP также используется в приложениях типа "запрос-ответ", где объем данных минимален, а повторная передача может быть выполнена быстро. Служба доменных имен (DNS) использует UDP для такого типа транзакций.

Разработчики приложений должны выбрать подходящий тип транспортного протокола, исходя из требований приложений. Видео может передаваться по TCP или UDP. Приложения, передающие потоковое хранимое аудио и видео, обычно используют TCP. Приложение использует TCP для выполнения буферизации, зондирования полосы пропускания и управления перегрузками, чтобы лучше контролировать работу пользователя.

Сеансовый уровень (уровень 5)

Сеансовый уровень предоставляет приложениям механизмы для установления сеансов между двумя хостами. Через эти сквозные сеансы могут быть предложены различные услуги. Функции сеансового уровня отслеживают, чья очередь передавать данные, следят за тем, чтобы две стороны не пытались выполнить одну и ту же операцию одновременно, подхватывают неудачную передачу с того места, где она была неудачной, и завершают передачу. Сеансовый уровень явно реализован в приложениях, использующих удаленные вызовы процедур (RPC).

Уровень представления (уровень 6)

Презентационный уровень определяет контекст между объектами прикладного уровня. До сих пор уровни модели OSI в основном занимались перемещением битов от узла-источника к узлу-получателю. Уровень представления занимается синтаксисом и семантикой передаваемой информации, а также тем, как эта информация организована. На этом уровне проводится различие между тем, какой тип данных закодирован для передачи, например, текст, двоичные файлы или видео.

Уровень приложений (уровень 7)

Прикладной уровень — это уровень OSI, который находится ближе всего к конечному пользователю и содержит различные протоколы, обычно необходимые пользователям. Одним из широко используемых прикладных протоколов является HyperText Transfer Protocol (HTTP) и его защищенная версия HTTPS. HTTP/HTTPS лежит в основе Всемирной паутины (WWW). Обмен информацией между клиентским браузером и веб-сервером осуществляется с помощью HTTP. Когда браузер клиента хочет отобразить веб-страницу, он посылает имя страницы на сервер, на котором она находится, используя HTTP. Сервер отправляет обратно веб-страницу по протоколу HTTP. В течение многих лет были разработаны другие протоколы для передачи данных, электронной почты и другие.

Некоторые другие примеры протоколов, которые работают на прикладном уровне, включают File Transfer Protocol (FTP), используемый для передачи файлов между хостами, и Dynamic Host Configuration Protocol (DHCP), используемый для динамического назначения IP-адресов хостам.

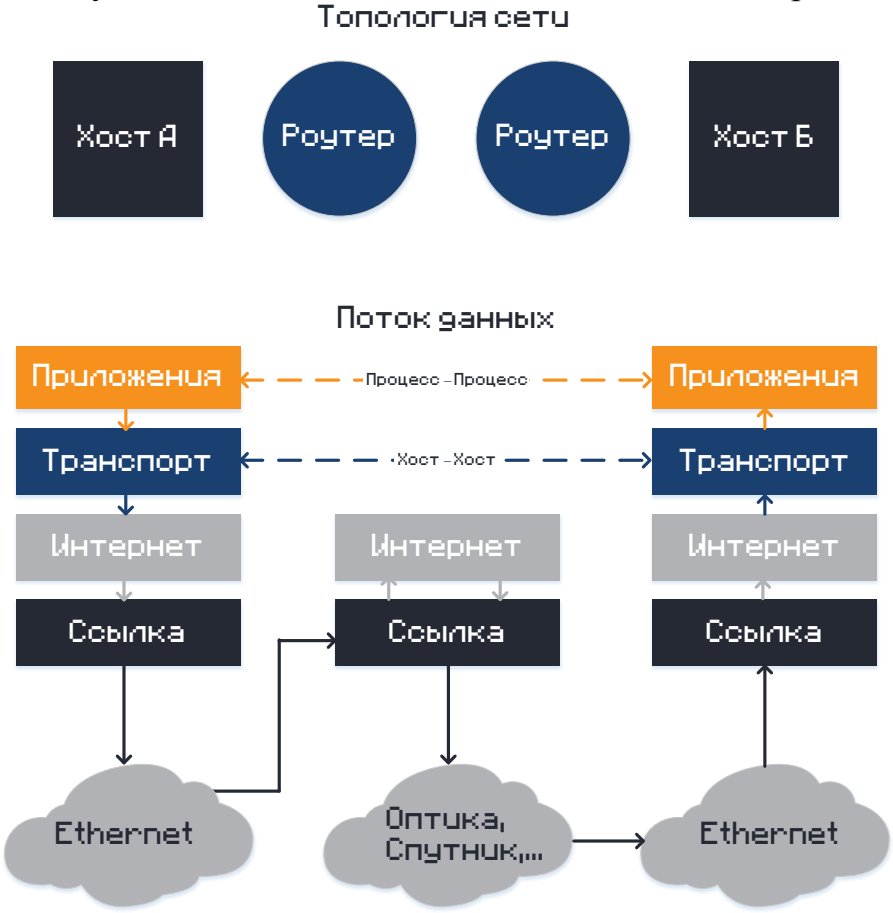


Рисунок 99 — Поток данных в многоуровневых моделях

Конечные устройства реализуют протоколы для всего "стека" уровней. Источник сообщения (данных) инкапсулирует данные с соответствующим заголовком/трейлером протокола на каждом уровне, а конечный получатель деинкапсулирует каждый заголовок/трейлер протокола для получения сообщения (данных).

Уровень доступа к сети (показан как "Ссылка" на иллюстрации выше) работает на уровне локальной сети, к которой подключено конечное устройство. Он занимается перемещением кадров от одной сетевой карты к другой сетевой карте в той же сети. На этом уровне работают коммутаторы Ethernet.

Интернет-уровень отвечает за передачу данных через потенциально несколько удаленных сетей. Соединение физически разрозненных сетей называется интернет-обработкой. Протоколы маршрутизации отвечают за отправку данных из сети источника в сеть назначения. Маршрутизаторы — это устройства, которые работают на интернет-уровне и выполняют функцию маршрутизации. Более подробно маршрутизаторы мы рассмотрим похже. IP работает на уровне Интернета в эталонной модели TCP/IP и выполняет две основные функции — адресацию и маршрутизацию.

Хосты идентифицируются по их IP-адресу. Для идентификации компьютеров хостов сети и определения их местоположения в сети в настоящее время поддерживаются две системы адресации. В IPv4 используются 32-битные адреса. Это означает, что приблизительно 4,3 миллиарда устройств могут быть идентифицированы. Сегодня к Интернету подключено гораздо больше, чем 4,3 миллиарда хостов, поэтому в конце 1990-х годов была разработана новая система адресации. IPv6 использует 128-битные адреса. Он был стандартизирован в 1998 году, а его внедрение началось в 2006 году. 128-битное адресное пространство IPv6 предоставляет 340 дециллионов адресов. В настоящее время в Интернете поддерживаются как IPv4, так и IPv6-адреса.

Вторая функция интернет-уровня — маршрутизация пакетов. Эта функция означает отправку пакетов от источника к месту назначения путем пересылки их на следующий маршрутизатор, который находится ближе к месту назначения. Благодаря этой функции интернет-уровень делает возможной работу в Интернете, соединяя различные IP-сети и, по сути, создавая Интернет. Передача IP-пакетов на интернет-уровне осуществляется с наилучшими усилиями и ненадежно. Любая повторная передача или исправление ошибок должны быть реализованы более высокими уровнями на конечных устройствах, обычно TCP.

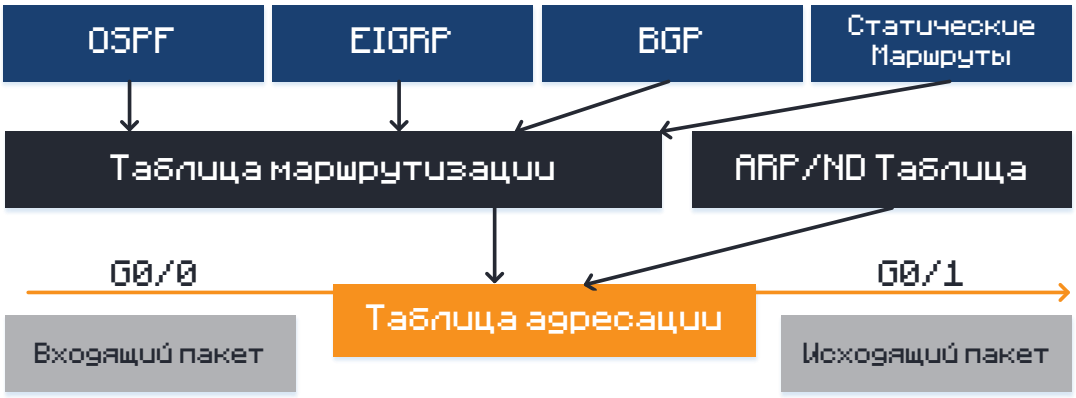


Рисунок 100 — Маршрутизация пакетов

Логика маршрутизатора управляется тремя функциональными плоскостями: плоскостью управления, плоскостью контроля и плоскостью данных. Каждая из них обеспечивает различные функциональные возможности:

- Плоскость управления — контролирует трафик, предназначенный для самого сетевого устройства. Примеры включают Secure Shell (SSH) и Simple Network Management Protocol (SNMP).
- Плоскость управления — обрабатывает трафик, необходимый для поддержания функциональности сетевой инфраструктуры. Плоскость управления состоит из приложений и протоколов между сетевыми устройствами, таких как протоколы маршрутизации OSPF, BGP и Enhanced Interior Gateway Routing Protocol (EIGRP). Плоскость управления обрабатывает данные в программном обеспечении.
- Плоскость данных — плоскость пересылки, которая отвечает за коммутацию пакетов в аппаратном обеспечении, используя информацию из плоскости управления. Плоскость данных обрабатывает данные в аппаратном обеспечении.

Уровень сетевых интерфейсов

Вникаем в уровень сетевых интерфейсов

Сеть состоит из конечных устройств, таких как компьютеры, мобильные устройства и принтеры, которые соединены сетевыми устройствами, такими как коммутаторы и маршрутизаторы. Сеть позволяет устройствам общаться друг с другом и обмениваться данными, как показано на рисунке 101.

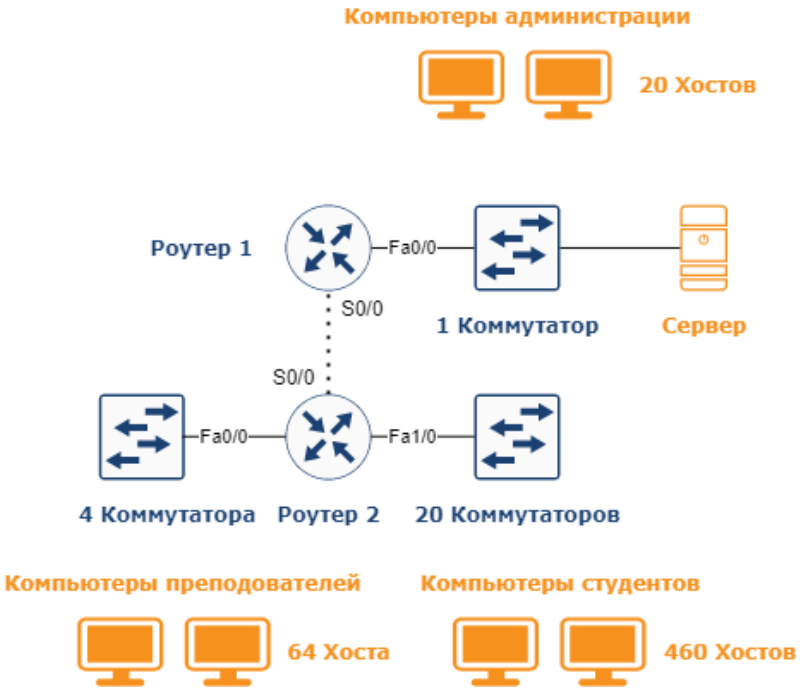


Рисунок 101 — Сетевая топология

На рисунке выше данные от компьютера студента к компьютеру преподавателя проходят через коммутатор к маршрутизатору (интерфейс

FastEthernet 1/0), затем к следующему коммутатору (интерфейс FastEthernet 0/0) и наконец к компьютеру преподавателя.

Все хосты и сетевые устройства, соединенные между собой в пределах небольшой физической области, образуют локальную сеть. Сетевые устройства, соединяющие локальные сети на больших расстояниях, образуют глобальную сеть (WAN).

Ethernet

Для соединения устройств в локальной сети требуется технологии, одной и наиболее распространенной из них является Ethernet —не просто тип кабеля или протокол, а целый сетевой стандарт, опубликованный IEEE. Ethernet — это набор рекомендаций и правил, которые позволяют различным сетевым компонентам работать вместе. Эти рекомендации определяют кабельные соединения и передачу сигналов на физическом и канальном уровнях модели OSI. Например, стандарты Ethernet рекомендуют различные типы кабелей и указывают максимальную длину сегмента для каждого типа.

Существует несколько типов сред, с которыми работает протокол Ethernet: коаксиальный кабель, витая медная пара, одномодовая и многомодовая fiber-оптика.

Биты, передаваемые по локальной сети Ethernet, организованы в кадры. Формат кадра Ethernet показан на рисунке.

На рисунке показаны 102 части кадра ethernet: преамбула, SFD, MAC-адрес назначения, MAC-адрес источника, EtherType, полезная нагрузка и FCS.

Рисунок 102 — Кадр Ethernet

В терминологии Ethernet контейнер, в который помещаются данные для передачи, называется кадром. Кадр содержит информацию заголовка, информацию трейлера и собственно передаваемые данные.

На рисунке выше показаны наиболее важные поля кадра Ethernet:

- Преамбула — это поле состоит из семи байт чередующихся 1 и 0, которые используются для синхронизации сигналов взаимодействующих компьютеров.
- Разделитель начала кадра (SFD) — это 1-байтовое поле, которое отмечает конец преамбулы и указывает на начало кадра Ethernet.
- MAC-адрес назначения — поле адреса назначения имеет длину шесть байт (48 бит) и содержит адрес сетевой карты в локальной сети, на которую отправляются инкапсулированные данные.
- MAC-адрес источника — поле адреса источника имеет длину шесть байт (48 бит) и содержит адрес сетевой карты устройства-отправителя.

- Тип — поле содержит код, идентифицирующий протокол сетевого уровня. Например, если протокол сетевого уровня - IPv4, то это поле имеет значение 0x0800, а для IPv6 - 0x086DD.
- Данные — это поле содержит данные, полученные от сетевого уровня на передающем компьютере. Затем эти данные отправляются в тот же протокол на компьютере назначения. Если длина данных меньше минимальной длины в 46 байт, для заполнения поля используется строка посторонних битов.
- Последовательность проверки кадра (FCS) — поле FCS включает механизм проверки, гарантирующий, что пакет данных был передан без повреждений.

MAC-адреса используются при передаче кадра через общий локальный носитель. Это обмен данными между сетевыми картами в одной сети. Если данные (инкапсулированный IP-пакет) предназначены для устройства в другой сети, MAC-адрес назначения будет адресом локального маршрутизатора (шлюза по умолчанию). Заголовок и конец Ethernet будут деинкапсулированы маршрутизатором. Пакет будет инкапсулирован в новый Ethernet-заголовок и конец с использованием MAC-адреса интерфейса выхода маршрутизатора в качестве MAC-адреса источника. Если следующим хопом является другой маршрутизатор, то MAC-адрес назначения будет MAC-адресом маршрутизатора следующего хопа. Если маршрутизатор находится в той же сети, что и место назначения пакета, MAC-адрес назначения будет MAC-адресом конечного устройства.

MAC Addresses

Все сетевые устройства в одной сети должны иметь уникальный MAC-адрес. MAC-адрес — это средство, с помощью которого данные направляются на соответствующее устройство назначения. MAC-адрес устройства — это адрес, который записывается в сетевую карту. Поэтому его также называют физическим адресом или burned in address (BIA).

MAC-адрес состоит из 12 шестнадцатеричных чисел, то есть имеет 48 бит. он состоит из двух основных компонентов. Первые 24 бита составляют OUI. Последние 24 бита составляют назначенный производителем адрес конечной станции, как показано на иллюстрации ниже:

- 24-битный OUI — OUI идентифицирует производителя сетевой карты. *IEEE регулирует присвоение номеров OUI.* 2 бита в OUI, имеют значение только при использовании в поле адреса назначения (DA) заголовка Ethernet;
- 24-битный, назначенный производителем, адрес конечной станции - Эта часть уникально идентифицирует оборудование Ethernet.

MAC-адрес может быть отображен любым из следующих способов:

- `0050.56c0.0001`

- `00:50:56:c0:00:01`
- `00-50-56-c0-00-01`

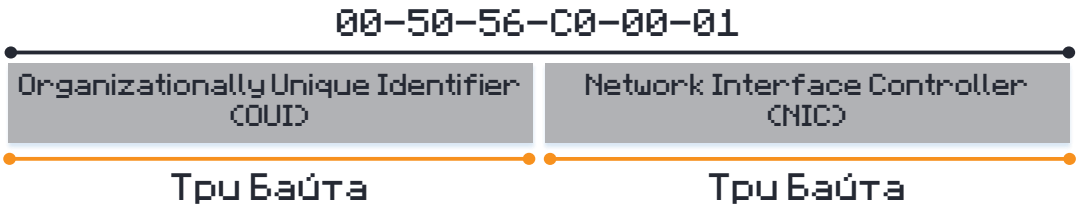


Рисунок 103 — Формат MAC-адреса

MAC-адреса назначения включают три основных типа сетевых коммуникаций:

- **Unicast** *(Одноадресная передача)* — передача, при которой кадр отправляется с одного узла и адресуется одному конкретному адресату. При одноадресной передаче есть только один отправитель и один получатель. Одноадресная передача является преобладающей формой передачи в локальных сетях и в Интернете;
- **Broadcast** *(Широковещательная передача)* — передача, при которой кадр отправляется с одного адреса на все остальные адреса. В этом случае есть только один отправитель, но информация отправляется всем подключенным получателям. Широковещательная передача необходима для отправки одного и того же сообщения всем устройствам в локальной сети, обычно используется, когда устройство ищет MAC-адрес получателя;
- **Multicast** *(Многоадресная передача)* — передача, при которой информация отправляется определенной группе устройств или клиентов. В отличие от широковещательной передачи, при многоадресной передаче клиенты должны быть членами многоадресной группы, чтобы получить информацию.

Коммутация

Коммутатор создает и поддерживает таблицу (называемую таблицей MAC-адресов), которая сопоставляет MAC-адрес назначения с портом, используемым для подключения к узлу. Таблица MAC-адресов хранится в памяти с возможностью адресации содержимого (Content Addressable Memory или CAM), что обеспечивает очень быстрый поиск.

Коммутатор динамически формирует таблицу MAC-адресов, изучая MAC-адрес источника кадров, полученных на порту. Коммутатор пересылает кадры путем поиска соответствия между MAC-адресом назначения в кадре и записью в таблице MAC-адресов. В зависимости от результата, коммутатор решает, задерживать или отклонять кадр. Если MAC-адрес назначения есть в таблице MAC-адресов, он отправит его через указанный порт. В противном случае он отправит его через все порты, кроме входящего порта.

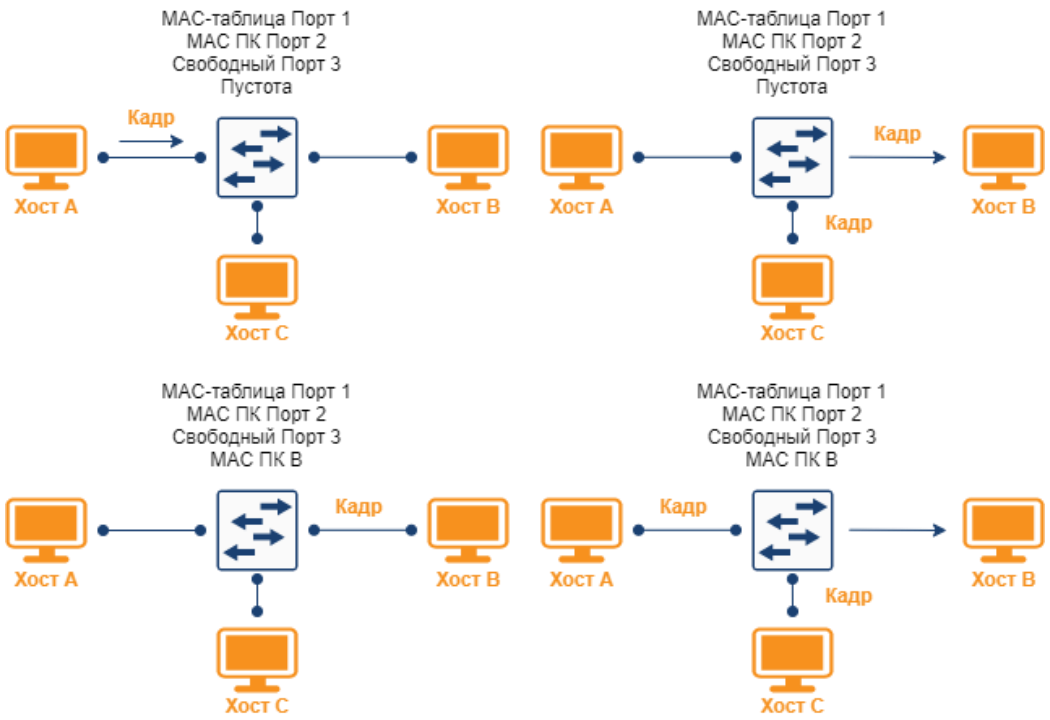


Рисунок 104 — Процесс коммутации.

На рисунке показаны четыре топологии. В каждой топологии есть коммутатор и три хоста (Хост A, Хост B и Хост C). Ниже описан процесс коммутации *(хост A посылает кадр хосту B)*:

1. В первой топологии, вверху слева, коммутатор получает кадр от Хоста A на порт 1.
2. Коммутатор заносит MAC-адрес источника и порт коммутатора, принявший кадр, в таблицу MAC-адресов.
3. Коммутатор проверяет таблицу на наличие MAC-адреса назначения. Поскольку адрес назначения неизвестен, коммутатор пересылает кадр на все порты, кроме порта, на котором он получил кадр. Во второй топологии, вверху справа, Хост B, MAC-адрес назначения, получает кадр Ethernet.
4. В третьей топологии, внизу слева, Хост B отвечает Хосту A с MAC-адресом назначения Хоста A.
5. Коммутатор заносит MAC-адрес источника Хоста B и номер порта коммутатора, принявшего кадр, в таблицу MAC-адресов. Адрес назначения кадра и связанный с ним порт известны в таблице MAC-адресов.
6. В четвертой топологии, внизу справа, коммутатор теперь может напрямую переслать этот кадр Хосту A через порт 1. Кадры между устройствами источника и назначения

Виртуальные локальные сети (VLAN)

Виртуальная локальная сеть (VLAN) используется для сегментации различных широковещательных доменов второго уровня на одном или нескольких коммутаторах. VLAN объединяет устройства в одной или

нескольких локальных сетях, которые настроены на взаимодействие так, как будто они подключены к одному проводу, хотя на самом деле они расположены в нескольких разных сегментах локальной сети. Поскольку сети VLAN основаны на логических, а не физических соединениях, они чрезвычайно гибкие.

Например, на рисунке 105 администратор сети создал три VLAN на основе функций пользователей: инженерная, маркетинговая и бухгалтерская. Обратите внимание, что устройства не обязательно должны находиться на одном этаже.

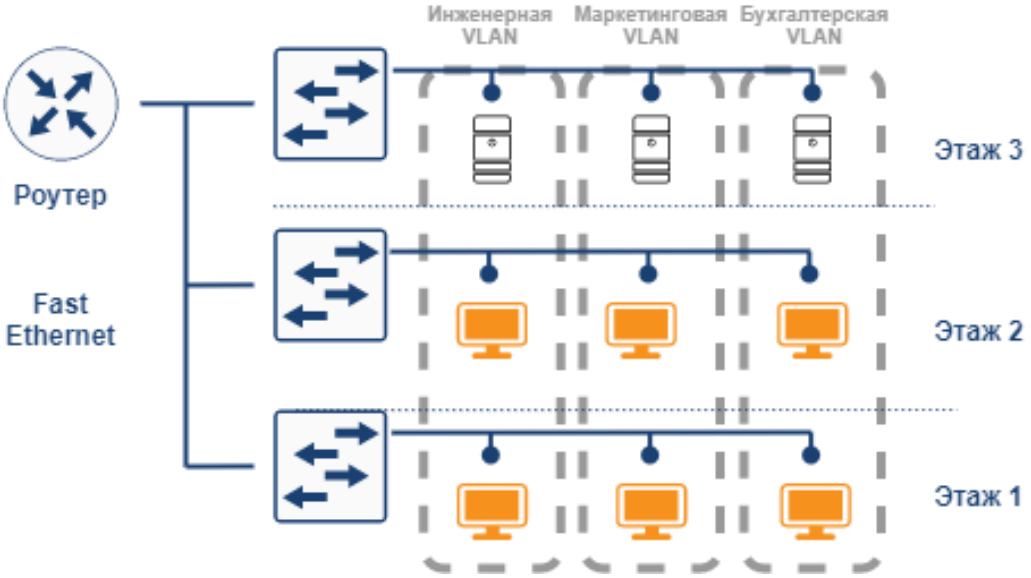


Рисунок 105 — Виртуальные локальные сети

VLAN создают широковещательные домены второго уровня. Широковещательные домены обычно ограничиваются маршрутизаторами, поскольку маршрутизаторы не пересылают широковещательные кадры. VLAN на коммутаторах второго уровня создают широковещательные домены на основе конфигурации коммутатора. Порты коммутатора назначаются на VLAN. Широковещательные кадры Уровня 2, полученные на порту коммутатора, передаются только на другие порты, принадлежащие к той же VLAN.

В коммутаторе можно создать одну или несколько виртуальных локальных сетей. Каждая VLAN, созданная в коммутаторе, определяет новый широковещательный домен. Трафик не может проходить непосредственно в другую VLAN (между широковещательными доменами) внутри коммутатора или между двумя коммутаторами. Для соединения двух разных VLAN необходимо использовать маршрутизатор или коммутатор 3-го уровня.

VLAN часто ассоциируются с IP-сетями или подсетями. Например, все конечные станции в определенной IP-подсети принадлежат к одной VLAN. Трафик между виртуальными локальными сетями должен быть маршрутизирован. Вы должны назначить принадлежность к VLAN (VLAN ID) порту коммутатора на основе каждого порта (это известно, как принадлежность к VLAN на основе интерфейса или статическая

принадлежность к VLAN). При создании VLAN на коммутаторе можно задать различные параметры, включая номер VLAN (VLAN ID) и имя VLAN.

Коммутаторы поддерживают 4096 VLAN в соответствии со стандартом IEEE 802.1Q, который определяет 12 бит (2^12=4096) для идентификатора VLAN.

Магистраль — это соединение "точка-точка" между двумя сетевыми устройствами, которое передает более одной сети VLAN. Магистраль VLAN распространяет сети VLAN на всю сеть. IEEE 802.1Q определяет "метку", которая вставляется в кадр и содержит идентификатор VLAN. Эта метка вставляется, когда кадр пересылается коммутатором на его выходной интерфейс. Метка удаляется коммутатором, принимающим кадр. Так коммутаторы узнают, членом какой VLAN является кадр.

Эти VLAN организованы в три диапазона: зарезервированный, обычный и расширенный (Таблица 11). Некоторые из этих VLAN распространяются на другие коммутаторы в сети при использовании протокола VLAN Trunking Protocol (VTP).

Таблица 11 — Диапазоны VLAN

VLAN	Диапазон	Использование
0, 4095	Зарезервировано	Только для системного использования. Вы не можете видеть или использовать эти VLAN.
1	Нормальный	По умолчанию Cisco. Вы можете использовать эту VLAN, но не можете удалить ее.
2 - 1001	Нормальный	Используется для сетей Ethernet VLAN; вы можете создавать, использовать и удалять эти сети VLAN.
1002 - 1005	Нормальный	Cisco по умолчанию для FDDI и Token Ring. Вы не можете удалить сети VLAN 1002-1005.
1006 - 4094	Расширенный	Только для сетей Ethernet VLAN.

Сетевой уровень интернета

Взаимосвязанные сети должны иметь способы связи, и интернет-обработка обеспечивает такой способ связи "между" (inter) сетями. Каждое устройство в сети имеет уникальный IP-адрес. IP-адрес и MAC-адрес используются для доступа и связи между всеми сетевыми устройствами. Без IP-адресов не было бы Интернета. Адрес IPv4 состоит из 32 бит, каждый октет (8 бит) представлен в виде десятичного значения, разделенного точкой. Такое представление называется точечной десятичной нотацией. Существует три типа адресов IPv4: сетевой адрес, адрес хоста и широковещательный адрес. Маска подсети IPv4 (или длина префикса) используется для разделения сетевой и хостовой частей адреса IPv4.

Таблица 12 — Двоичное значение для каждого октета IPv4

Значение	192	168	48	64
Двоичное	11000000	10101000	00110000	01000000
Значение	**64**	**100**	**36**	**254**
Двоичное	01000000	01100100	00100100	11111110

IPv6 разработан как преемник IPv4. IPv6 имеет более широкое 128-битное адресное пространство, обеспечивающее 340 недециллионов возможных адресов. Агрегация IPv6 prex, упрощенная перенумерация сетей и возможности IPv6 site multihoming обеспечивают иерархию адресации IPv6, которая позволяет осуществлять более эффективную маршрутизацию. Адреса IPv6 представлены в виде серии 16-битных шестнадцатеричных полей (гекстет), разделенных двоеточиями (:) в формате: x:x:x:x:x:x:x:x:x:x. Предпочтительный формат включает все шестнадцатеричные значения. Существует два правила, которые можно использовать для сокращения представления адреса IPv6: 1. Опустить ведущие нули в каждом шестнадцатеричном значении, и 2. Заменить одиночную строку гекстетов со всеми нулями двойным двоеточием (::) см. в таблице 13.

Таблица 13 — Сжатые IPv6 адреса

IPv6 address type	Preferred format	Compressed format
Unicast	2001:0:0:0:db8:800:200c:417a	2001::db8:800:200c:417a
Multicast	ff01:0:0:0:0:0:0:101	ff01::101
Loopback	0:0:0:0:0:0:0:1	::1
Unspecified	0:0:0:0:0:0:0:0	::

Одноадресный адрес IPv6 — это идентификатор для одного интерфейса на одном узле. Глобальный одноадресный адрес (GUA) (или агрегируемый глобальный одноадресный адрес) — это IPv6, аналогичный публичному адресу IPv4. Префикс глобальной маршрутизации — это префиксная, или сетевая, часть адреса, которая назначается провайдером, например, интернет-провайдером, клиенту или сайту. Поле ID Подсети — это область между префиксом глобальной маршрутизации и идентификатором интерфейса. Идентификатор интерфейса IPv6 эквивалентен хост-части адреса IPv4.

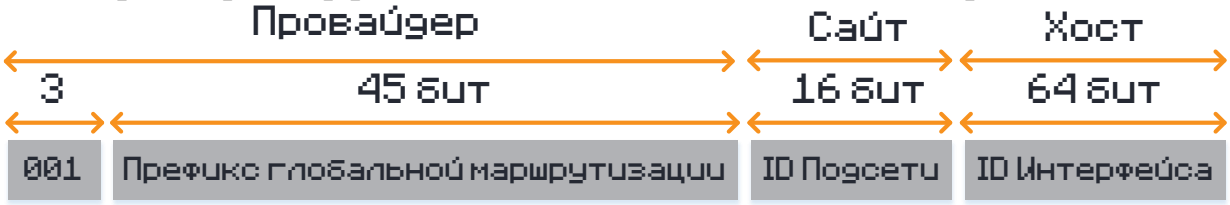


Рисунок 106 — Формат IPv6 GUA

Локальный адрес канала IPv6 (LLA) позволяет устройству взаимодействовать с другими устройствами с поддержкой IPv6 на том же канале и только на этом канале (подсети). Многоадресные адреса IPv6 аналогичны многоадресным адресам IPv4. Напомним, что многоадресная рассылка используется для отправки одного пакета в один или несколько пунктов назначения (многоадресная группа). Вот две распространенные многоадресные группы, назначенные в IPv6: ff02::1 All-nodes multicast group и ff02::2 All-routers multicast group.

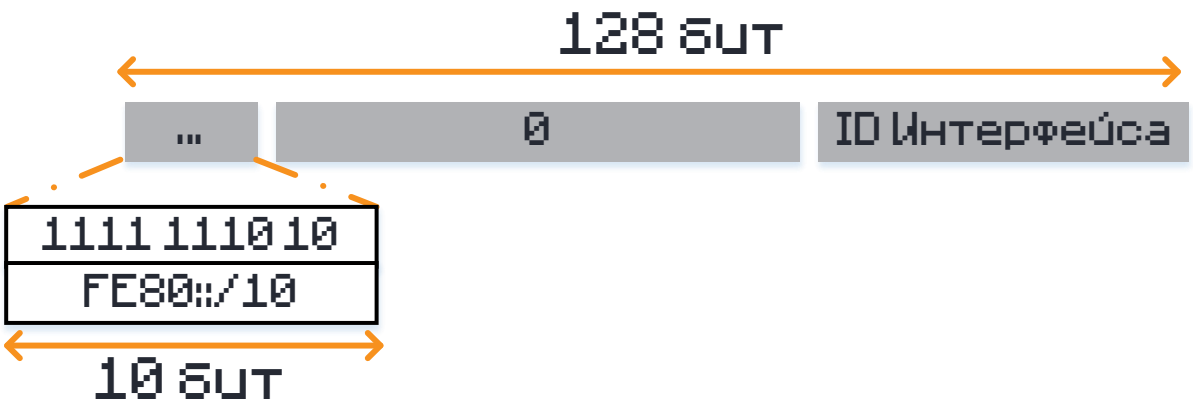


Рисунок 107 — Формат IPv6 LLA

Маршрутизатор — это сетевое устройство, функционирующее на интернет-уровне модели TCP/IP или сетевом уровне 3-го уровня модели OSI. Маршрутизация включает в себя пересылку пакетов между различными сетями. Для маршрутизации между сетями маршрутизаторы используют таблицу маршрутизации. Маршрутизатор обычно выполняет две основные функции: определение пути и маршрутизация или пересылка пакетов. Таблица маршрутизации может содержать следующие типы записей: непосредственно подключенные сети, статические маршруты, маршруты по умолчанию и динамические маршруты.

Сетевые устройства

Ключевым понятием в коммутации Ethernet является широковещательный домен. Широковещательный домен — это логическое разделение, в котором все устройства в сети могут связаться друг с другом посредством широковещательной передачи на канальном уровне. Теперь коммутаторы могут одновременно передавать и принимать данные. Коммутаторы выполняют следующие функции:

– Работают на уровне доступа к сети модели TCP/IP и канальном уровне уровня 2 модели OSI
– Фильтровать или рассылать кадры на основе записей в таблице MAC-адресов
– иметь большое количество высокоскоростных и полнодуплексных портов

Коммутатор работает в одном из следующих режимов коммутации: сквозной и с сохранением и пересылкой. Коммутаторы локальных сетей имеют высокую плотность портов, большие буферы кадров и быструю внутреннюю коммутацию.

Маршрутизаторы необходимы для связи с устройствами, не входящими в одну локальную сеть. Маршрутизаторы используют таблицы маршрутизации для направления трафика между различными сетями. Маршрутизаторы подключаются к различным сетям (или подсетям) через свои интерфейсы и имеют возможность маршрутизации трафика данных между ними.

Маршрутизаторы выполняют следующие функции:

- Они работают на интернет-уровне модели TCP/IP и сетевом уровне 3-го уровня модели OSI.
- Они маршрутизируют пакеты между сетями на основе записей в таблице маршрутизации.
- Они поддерживают большое количество сетевых портов, включая различные LAN и WAN медиапорты, которые могут быть медными или оптоволоконными. Количество интерфейсов на маршрутизаторах обычно намного меньше, чем на коммутаторах, но разнообразие поддерживаемых интерфейсов больше. IP-адреса настраиваются на интерфейсах.

Маршрутизаторы поддерживают три механизма пересылки пакетов: коммутация процессов, быстрая коммутация и CEF.

Межсетевой экран — это аппаратная или программная система, которая предотвращает несанкционированный доступ в сеть или из сети. Самым основным типом межсетевого экрана является безэталонный пакетный экранирующий экран. Вы создаете статические правила, которые разрешают или запрещают пакеты, основываясь на информации заголовка пакета. Межсетевой экран проверяет пакеты, когда они проходят через него, сравнивает их со статическими правилами и соответственно разрешает или запрещает трафик. Сетевой экран с контролем состояния пакетов выполняет ту же проверку заголовков, что и сетевой экран без контроля состояния пакетов, но при этом отслеживает состояние соединения. Для отслеживания состояния эти межсетевые экраны ведут таблицу состояний. Наиболее продвинутый тип firewall — это firewall прикладного уровня. В этом типе глубокая проверка пакета происходит вплоть до 7-го уровня модели OSI.

Балансировка нагрузки улучшает распределение рабочих нагрузок между несколькими вычислительными ресурсами, такими как серверы, кластеры серверов, сетевые каналы и т. д. Балансировка нагрузки сервера помогает обеспечить доступность, масштабируемость и безопасность приложений и услуг, распределяя работу одного сервера между несколькими серверами. На уровне устройств балансировщик нагрузки обеспечивает высокую доступность сети за счет поддержки: избыточности устройств, масштабируемости и безопасности. На уровне сетевых служб балансировщик нагрузки предоставляет расширенные услуги, поддерживая: высокую доступность служб, масштабируемость и безопасность на уровне служб.

Сетевые диаграммы отображают визуальное и интуитивное представление сети, как подключены все устройства, в каких зданиях, комнатах, шкафах они расположены, какой интерфейс подключается к данному конечному устройству и т.д. Как правило, существует два типа сетевых диаграмм: диаграммы физической связности второго уровня и диаграммы логической связности третьего уровня. Уровень 2, или диаграммы физической связности — это сетевые диаграммы, представляющие соединение портов между устройствами в сети. По сути, это визуальное представление

того, какой сетевой порт сетевого устройства подключается к какому сетевому порту другого сетевого устройства. Уровень 3, или диаграммы логической связности, - это сетевые диаграммы, отображающие IP-соединения между устройствами в сети.

Сетевые протоколы

Telnet и SSH, или Secure SHell, используются для подключения к удаленному компьютеру и входа в систему с помощью учетных данных. Telnet сегодня менее распространен, поскольку SSH использует шифрование для защиты данных, передаваемых по сетевому соединению, а безопасность данных является главным приоритетом. HTTP означает Hyper Text Transfer Protocol, а HTTPS добавляет ключевое слово "Secure" в конец аббревиатуры. Этот протокол узнаваем в веб-браузерах как протокол, который используется для подключения к веб-сайтам. NETCONF имеет стандартизированное значение порта — 830. RESTCONF не имеет зарезервированного значения порта, поэтому вы можете встретить различные реализации с разными значениями.

Dynamic Host Configuration Protocol (DHCP) используется для передачи информации о конфигурации хостам в сети TCP/IP. DHCP распределяет IP-адреса тремя способами: автоматическим, динамическим и ручным. Операции DHCP включают четыре сообщения между клиентом и сервером: обнаружение сервера, предложение аренды IP-адреса, запрос аренды IP-адреса и подтверждение аренды IP-адреса.

Протокол DNS определяет автоматическую службу, которая сопоставляет имена ресурсов с требуемым цифровым сетевым адресом. Он включает формат запросов, ответов и данных. Для обмена данными по протоколу DNS используется единый формат, называемый сообщением DNS. Сервер DNS хранит различные типы записей ресурсов, которые используются для разрешения имен. Эти записи содержат имя, адрес и тип записи.

Система SNMP состоит из трех элементов:

- SNMP-менеджер: система управления сетью (NMS)
- SNMP-агенты (управляемый узел)
- База управленческой информации (MIB)

Существует два основных запроса менеджера SNMP: get и set. Запрос get используется NMS для запроса данных на устройстве. Запрос set используется NMS для изменения переменных конфигурации устройства-агента. Ловушки - это незапрашиваемые сообщения, предупреждающие SNMP-менеджера о состоянии или событии в сети. SNMPv1 и SNMPv2c используют строки сообщества, которые контролируют доступ к MIB. Строки сообществ SNMP (включая только для чтения и чтения-записи) аутентифицируют доступ к объектам MIB. Рассматривайте MIB как "карту" всех компонентов устройства, которые управляются SNMP.

NTP используется для распределения и синхронизации времени между распределенными серверами времени и клиентами. Авторитетным

источником времени обычно являются радиочасы или атомные часы, подключенные к серверу времени. Серверы NTP могут взаимодействовать в нескольких режимах, включая: клиент/сервер, симметричный активный/пассивный и широковещательный.

Трансляция сетевых адресов (NAT) помогает решить проблему нехватки адресов IPv4. NAT работает путем сопоставления тысяч частных внутренних адресов с рядом публичных адресов. Сопоставляя внешние и внутренние адреса IPv4, NAT позволяет организациям с адресами, не имеющими глобальной маршрутизации, подключаться к Интернету, переводя адреса в глобально маршрутизируемое адресное пространство. NAT включает четыре типа адресов:

- Внутренний локальный адрес
- Внутренний глобальный адрес
- Внешний локальный адрес
- Внешний глобальный адрес

Типы NAT включают: статическую NAT, динамическую NAT и трансляцию адресов портов (PAT).

Что такое YANG?

YANG, сокращение от Yet Another Next Generation, как определено в RFC7519, является "языком моделирования данных, используемым для моделирования данных конфигурации и состояния, которыми манипулирует протокол сетевой конфигурации (NETCONF), удаленные вызовы процедур NETCONF и уведомления NETCONF".

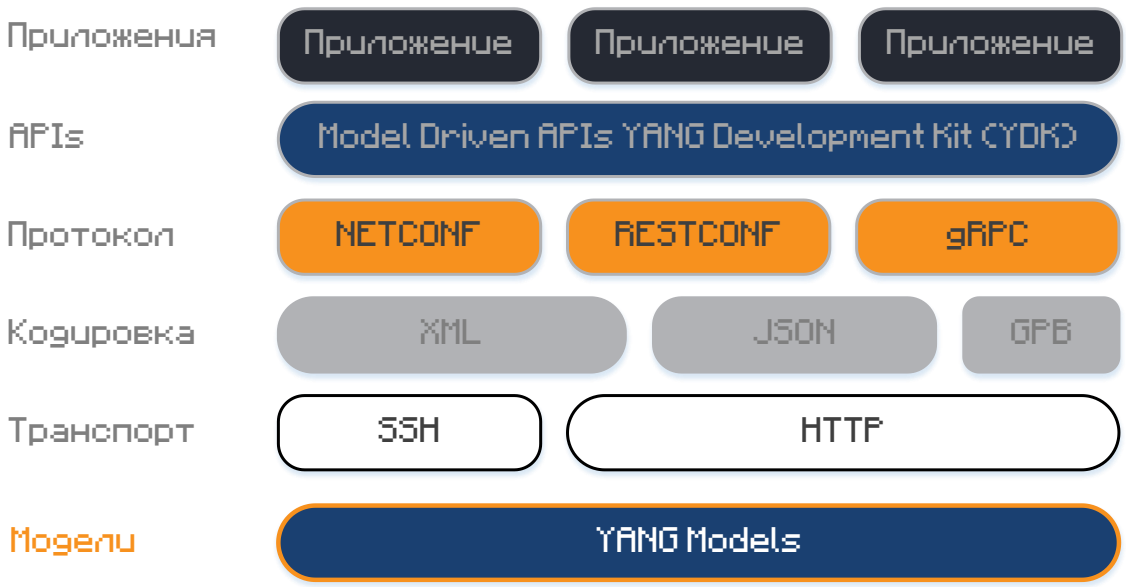


Рисунок 108 — YANG в стеке программирования на основе моделей

Модуль YANG определяет иерархии данных, которые могут быть использованы для операций на основе NETCONF, включая конфигурацию, данные состояния, RPC и уведомления. Это позволяет получить полное

описание всех данных, передаваемых между клиентом и сервером NETCONF. YANG также может использоваться с протоколами, отличными от NETCONF. Хотя YANG может описывать любую модель данных, изначально он был разработан для сетевых моделей данных.

В реальном мире существует два типа моделей YANG: открытые и собственные.

- Открытые модели YANG: Разработаны производителями и органами по стандартизации, такими как IETF, ITU, OpenConfig и т.д.. Они предназначены для независимости от базовой платформы и нормализации конфигурации сетевых устройств в зависимости от производителя.
- Встроенные модели: Разработаны поставщиками. Они связаны и предназначены для интеграции функций или конфигурации, относящихся только к данной платформе.

Почему нам нужен YANG для моделирования устройств?

YANG обеспечивает стандарт, но при этом допускает дальнейшее расширение описания. Нам нужен стандартный способ моделирования конфигурации сетевых устройств. YANG позволяет различным производителям сетевых устройств описывать тип, конфигурацию и состояние своих устройств, чтобы программно отобразить работу устройства.

Термины, используемые в YANG, определены в разделе 3 RFC6020. Вот некоторые часто используемые термины, которые необходимо понять, прежде чем погружаться в YANG:

- **anyxml:** Узел данных, который может содержать неизвестный фрагмент данных XML.
- **augment:** Добавляет новые узлы схемы к ранее определенному узлу схемы.
- **container**: Внутренний узел данных, который существует не более чем в одном экземпляре в дереве данных. Контейнер не имеет значения, а скорее является набором дочерних узлов.
- **data model**: Модель данных описывает, как данные представляются и как к ним осуществляется доступ.
- **data node**: Узел в дереве схемы, который может быть инстанцирован в дереве данных. Один из контейнера, листа, листа-списка, списка и anyxml.
- **data tree**: Инстанцированное дерево данных конфигурации и состояния устройства.
- **derived type**: Тип, который является производным от встроенного типа (например, uint32) или другого производного типа.
- **grouping**: Многократно используемый набор узлов схемы. Группировка может использоваться локально в модуле, в модулях, которые его включают, и в других модулях, которые импортируют из него. Утверждение группировки не является утверждением определения данных и, как таковое, не определяет никаких узлов в дереве схемы.

– **identifier**: Используется для идентификации различных типов элементов YANG по имени.
– **leaf**: Узел данных, который существует не более чем в одном экземпляре в дереве данных. Лист имеет значение, но не имеет дочерних узлов.
– **leaf-list**: Аналогичен узлу leaf, но определяет набор уникально идентифицируемых узлов, а не один узел. Каждый узел имеет значение, но не имеет дочерних узлов.
– **list**: Внутренний узел данных, который может существовать в нескольких экземплярах в дереве данных. Список не имеет значения, но имеет набор дочерних узлов.
– **module**: Модуль YANG определяет иерархию узлов, которые могут быть использованы для операций на основе NETCONF. Благодаря своим определениям и определениям, которые он импортирует или включает из других источников, модуль является самодостаточным и "компилируемым".
– **RPC**: Удаленный вызов процедур, используемый в протоколе NETCONF.
– **state data**: Дополнительные данные о системе, которые не являются данными конфигурации, такие как информация о состоянии только для чтения и собранная статистика [RFC4741].

Что такое NETCONF?

Конфигурация сети (NETCONF) — это протокол, определенный IETF RFC7519. Он предназначен для установки, манипулирования и удаления конфигурации сетевых устройств. На сегодняшний день это основной протокол, используемый в моделях данных YANG. Его операции реализуются поверх удаленного вызова процедур (RPC), обмен которыми осуществляется через безопасный транспортный протокол, такой как SSH.

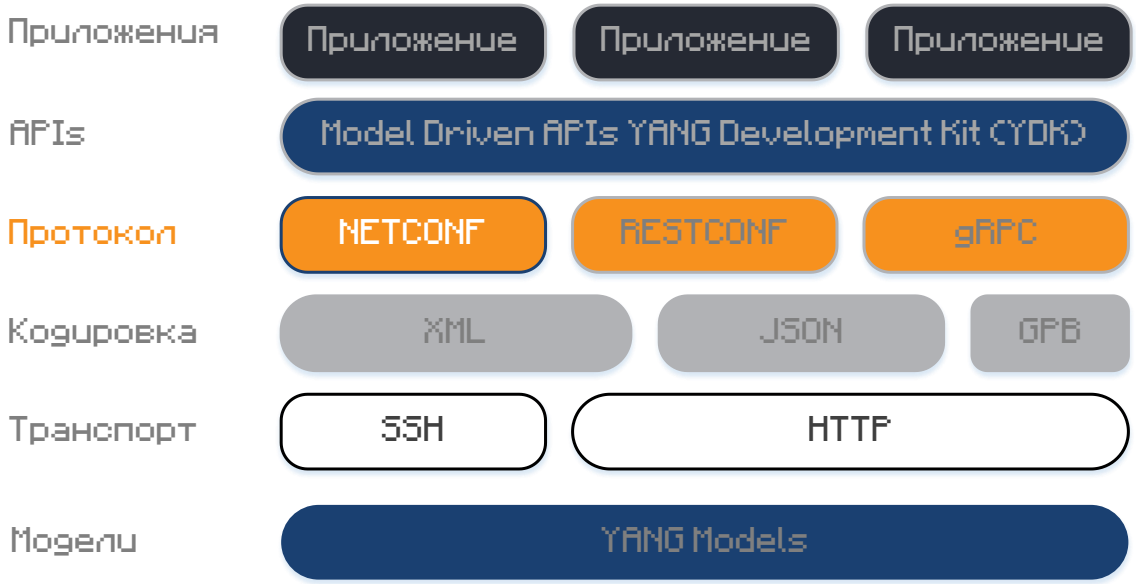


Рисунок 109 — NETCONF в стеке программирования на основе моделей

Протокол NETCONF использует кодировку данных на основе XML как для данных конфигурации, так и для сообщений протокола.
Протокол NETCONF предоставляет небольшой набор операций для управления конфигурациями устройств и получения информации о состоянии устройств. Базовый протокол предоставляет операции по получению, настройке, копированию и удалению хранилищ данных конфигурации.

Операции протокола NETCONF

Протокол NETCONF предоставляет набор операций для управления конфигурациями устройств и получения информации о состоянии устройств. Базовый протокол включает следующие операции протокола (Таблица 14):

Таблица 14 — Операции NETCONF

Операция	Описание
get	получить информацию о текущей конфигурации и состоянии устройства.
get-config	получить всю или часть указанной конфигурации.
edit-config	редактировать конфигурацию устройства.
copy-config	создать или заменить целое хранилище данных конфигурации другим полным хранилищем данных конфигурации.
delete-config	удалить конфигурацию в хранилище данных.
lock	блокировка всей системы конфигурационного хранилища данных устройства.
unlock	освободить блокировку конфигурации, полученную ранее с помощью операции блокировки.
close-session	запросить плавное завершение сеанса NETCONF.
kill-session	принудительное завершение сеанса NETCONF.

Что такое RESTCONF?

RESTCONF RFC 8040 определяет протокол и механизм для REST-подобного доступа к информации о конфигурации и управлению. Подобно NETCONF, он использует модели хранилищ данных и командные глаголы, определенные в протоколе сетевой конфигурации (NETCONF), инкапсулированные в HTTP-сообщения. Как и в NETCONF, язык YANG используется для определения синтаксиса структуры команд, а также семантики конфигурации хранилища данных, состояния и событий.

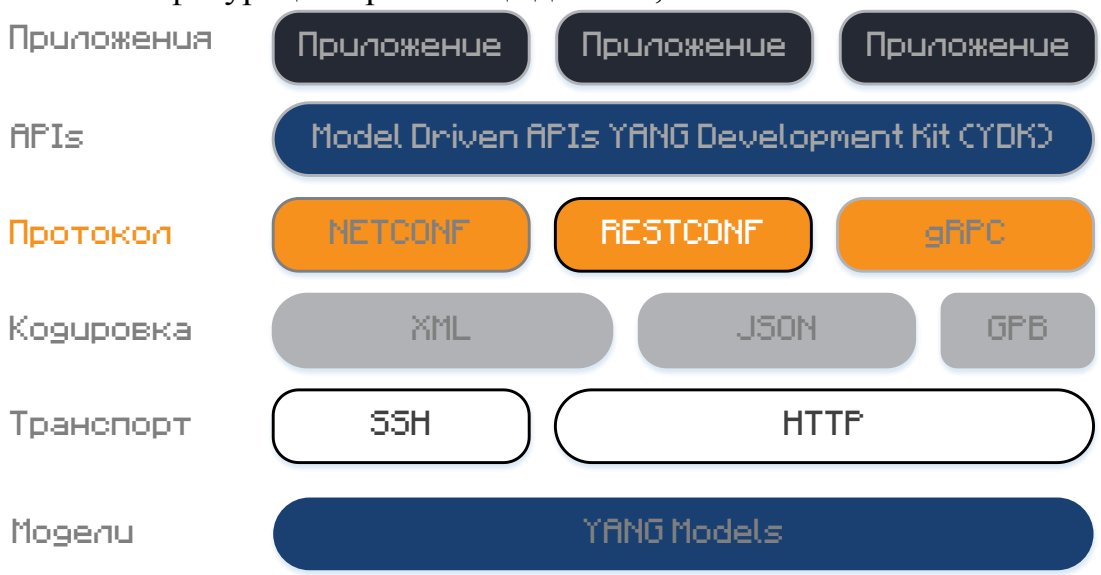


Рисунок 110 — RESTCONF в стеке модельно-ориентированного программирования

RESTCONF использует структурированные данные (XML или JSON) и YANG для предоставления REST-подобных API, обеспечивая программный доступ к устройствам. HTTP-команды GET, POST, PUT, PATCH и DELETE направляются в API RESTCONF для доступа к ресурсам данных, представленных моделями данных YANG. Эти базовые операции редактирования позволяют клиенту RESTCONF просматривать и изменять текущую конфигурацию.

RESTCONF не предназначен для замены NETCONF, а скорее для предоставления интерфейса HTTP, который следует принципам REST и совместим с моделью хранилища данных NETCONF. Сетевое устройство может одновременно обслуживать NETCONF и RESTCONF или предоставлять только одно или другое.

RESTCONF против NETCONF

В целом, NETCONF является более полным, гибким и сложным, чем RESTCONF. RESTCONF проще в освоении и использовании для инженеров с предыдущим опытом работы с REST API. Ниже перечислены различия между NETCONF и RESTCONF (Таблица 15):

- NETCONF поддерживает работающие хранилища данных и хранилища-кандидаты, в то время как RESTCONF поддерживает только работающее хранилище данных, поскольку любые правки в хранилище-кандидате немедленно фиксируются.
- RESTCONF не поддерживает получение или освобождение блокировки хранилища данных. Если хранилище данных имеет активную блокировку, операция редактирования RESTCONF будет неудачной.
- Редактирование RESTCONF — это транзакция, ограниченная одним вызовом RESTCONF.
- RESTCONF не поддерживает транзакции между несколькими устройствами.
- Валидация неявно присутствует в каждой операции редактирования RESTCONF, которая либо успешна, либо неудачна.

Таблица 15 — Сопоставление операций NETCONF и методов RESTCONF

Описание	NETCONF	RESTCONF
Создание ресурса данных	`<edit-config>, </edit-config>`	POST
Получение данных и метаданных	`<get-config>, <get> , </get-config>`	GET
Создание или замена ресурса данных	`<edit-config> (nc:operation="create/replace")`	PUT
Удаление ресурса данных	`<edit-config> (nc:operation="delete")`	DELETE