Методическое Пособие по Курсу

“DevOps Программирование: Практический Аспект”

Полезные ссылки на методические материалы для создания и управления развернутым кластером :

http://178.140.10.58:8083/read/4391/pdf

https://libgen.gs/edition.php?id=138721270

https://github.com/bregman-arie/devops-exercises

https://bookflow.ru/infrastruktura-kak-kod-praktiki-dlya-devops-inzhenerov/

Цель курса: Сформировать у студентов комплексное понимание и практические навыки работы с ключевыми инструментами и практиками DevOps, используя VirtualBox и Proxmox VE для создания собственной лабораторной среды.

Основные инструменты: VirtualBox, Proxmox VE, Debian, Git, Docker, GitHub Actions, Terraform, Ansible.

Часть 1: Подготовка Рабочей Среды и Базовых Инструментов

Модуль 1.1: Установка VirtualBox и Создание Debian VM для Proxmox

Цель: Установить VirtualBox на Windows 10 и создать виртуальную машину с ОС Debian, на которую будет установлен Proxmox VE.

Предварительные требования:

1. **Операционная система:** Windows 10 (или новее).
2. **ISO-образ Debian:** Скачан с debian.org/download (выберите amd64-netinst.iso).
3. **ISO-образ Proxmox VE:** Скачан с proxmox.com/en/downloads (“Proxmox VE ISO Installer”).

Шаг 1: Установка VirtualBox

1. Скачайте последнюю версию VirtualBox с virtualbox.org/wiki/Downloads.
2. Запустите установщик (VirtualBox-x.x.x-....exe).
3. Следуйте инструкциям мастера установки, принимая настройки по умолчанию.
4. При установке может потребоваться разрешение на установку драйверов — разрешите.
5. После установки запустите VirtualBox.

Шаг 2: Создание Виртуальной Машины для Proxmox

1. В VirtualBox нажмите **“Создать” (New)**.
2. **Имя и Операционная система:**
    - **Имя:** debian-for-proxmox
    - **Папка машины:** Выберите место для хранения файлов ВМ.
    - **Тип:** Linux
    - **Версия:** Debian (64-bit)
    - Нажмите **“Далее”**.
3. **Размер памяти:**
    - Установите минимум 4096 МВ (4 ГБ), так как на этой ВМ будет работать Proxmox и другие сервисы.
    - Нажмите **“Далее”**.
4. **Жесткий диск:**
    - Выберите **“Создать новый виртуальный жесткий диск”**.
    - Нажмите **“Создать”**.

5. **Тип файла виртуального диска:**
    - Выберите **“VDI (VirtualBox Disk Image)”** (по умолчанию).
    - Нажмите **“Далее”**.
6. **Формат хранения:**
    - Выберите **“Динамический виртуальный жесткий диск”**.
    - Нажмите **“Далее”**.
7. **Расположение и размер файла:**
    - Размер диска: Установите минимум 50 GB (для Debian + Proxmox + VM).
    - Нажмите **“Создать”**.

Шаг 3: Настройка Сетевого Адаптера для Proxmox VM

1. Выберите созданную ВМ debian-for-proxmox в списке VirtualBox.
2. Нажмите **“Настроить” (Settings)**.
3. Перейдите в раздел **“Сеть” (Network)**.
4. **Адаптер 1:**
    - Убедитесь, что **“Включить сетевой адаптер”** установлено.
    - **Тип подключения:** Выберите **“Сеть NAT” (NAT Network)**.
        - **Почему NAT Network?** NAT Network создает отдельную виртуальную подсеть для ваших ВМ. Это позволит им общаться друг с другом и выходить в интернет, но они не будут напрямую доступны из вашей домашней сети, что безопаснее.
        - **Настройка NAT Network:**

a. В главном окне VirtualBox: **“Файл” (File) -> “Настройки сети” (Network Settings)**.

b. Перейдите на вкладку **“NAT Networks”**.

c. Нажмите кнопку **”+”** для добавления новой NAT Network.

d. Дайте ей имя (например, DevOpsNet).

e. В разделе **“IPv4 Network CIDR”** укажите подсеть, например, 192.168.56.0/24. Это будет диапазон IP-адресов для ваших ВМ.

f. Нажмите **“OK”**.

- Теперь вернитесь к настройкам вашей VM debian-for-proxmox, в **“Сеть” -> “Адаптер 1” -> “Тип подключения”** выберите **“NAT Network”** и укажите созданную сеть **“DevOpsNet”**.
- Нажмите **“OK”**.

Шаг 4: Подключение ISO-образа Debian

1. Выберите VM debian-for-proxmox, нажмите **“Настроить” (Settings)**.
2. Перейдите в **“Носители” (Storage)**.
3. В дереве устройств нажмите на значок CD/DVD рядом с **“Контроллер: IDE”**.
4. В разделе **“Атрибуты”** нажмите на значок диска справа от поля **“Оптический привод”**, выберите **“Выбрать файл диска…”** и найдите скачанный ISO-образ Debian Server.
5. Нажмите **“OK”**.

Шаг 5: Установка Debian на VM

1. Выберите debian-for-proxmox, нажмите **“Запустить” (Start)**.
2. В консоли VM пройдите установку Debian:
    - Язык: English, Местоположение: United States, Клавиатура: American English.
    - **Сеть:**
        - **Hostname:** debian-proxmox-host
        - **Configure network automatically:** Yes. VirtualBox NAT Network должен выдать IP из диапазона 192.168.56.x. **ЗАПИШИТЕ IP-адрес.**
    - **Пароли:** Задайте надежные пароли для root и пользователя devops_user.
    - **Partition disks:** Guided - use entire disk.
    - **Software selection:** Выберите SSH server и Standard system utilities.
    - **GRUB boot loader:** Установите на /dev/sda.
3. После завершения установки перезагрузите VM (VirtualBox предложит извлечь ISO).

Шаг 6: Определение IP-адреса VM и Проверка SSH-доступа

1. После загрузки Debian, войдите как devops_user и выполните ip a. Запишите IP-адрес (обычно 192.168.56.x).
2. На вашей рабочей машине откройте терминал.
3. Подключитесь по SSH: ssh devops_user@<IP_адрес_вашей_Debian_VM> (например, ssh devops_user@192.168.56.101).
4. Введите пароль devops_user. При первом подключении введите yes.

5. Если вы увидели командную строку, SSH-доступ работает.

Практикум:

- **Задание 1.1.1:** Установите VirtualBox.
- **Задание 1.1.2:** Скачайте ISO Debian Server.
- **Задание 1.1.3:** Создайте VM debian-proxmox-host в VirtualBox, настройте NAT Network (192.168.56.0/24) и подключите к ней VM.
- **Задание 1.1.4:** Установите Debian на debian-proxmox-host, запишите IP-адрес и пароли.
- **Задание 1.1.5:** Проверьте SSH-доступ к debian-proxmox-host с вашей рабочей машины.

Модуль 1.2: Установка Proxmox VE на Debian VM

Цель: Установить Proxmox VE как операционную систему на нашу Debian VM.

Предварительные требования:

1. Успешно развернутая VM debian-proxmox-host с SSH-доступом.
2. SSH-клиент на вашей рабочей машине.

Шаг 1: Подключение к Debian VM

1. Подключитесь к debian-proxmox-host по SSH: ssh devops_user@<IP_адрес_вашей_Debian_VM>.

Шаг 2: Настройка Репозиториев Proxmox VE

1. **Добавление репозитория PVE (No-Subscription):**

- Proxmox VE по умолчанию требует подписку для доступа к полным репозиториям. Для тестирования и обучения мы будем использовать репозиторий “no-subscription”.
- Создайте файл /etc/apt/sources.list.d/pve-install-repo.list:

sudo nano /etc/apt/sources.list.d/pve-install-repo.list

- Добавьте следующие строки:
- deb [arch=amd64] http://download.proxmox.com/debian/pve bullseye pve-no-subscription
- deb [arch=amd64] http://ftp.debian.org/debian bullseye main contrib
- deb [arch=amd64] http://ftp.debian.org/debian bullseye-updates main contrib

deb [arch=amd64] http://security.debian.org/debian-security bullseye-security main contrib # Если вам нужен доступ к security репозиторию

  - **Примечание:** Замените bullseye на bookworm или другую актуальную версию Debian, если это необходимо (например, если вы устанавливали более новую версию Debian).
- Сохраните файл (Ctrl+O, Enter) и выйдите (Ctrl+X).

Шаг 3: Обновление Системы и Установка Proxmox VE

1. Обновите список пакетов:

sudo apt update

2. Обновите установленные пакеты (для обеспечения совместимости):

sudo apt dist-upgrade -y

3. Установите Proxmox VE:

sudo apt install proxmox-ve postfix open-iscsi -y

- proxmox-ve: Основной пакет.
- postfix: Почтовый сервер (для уведомлений).
- open-iscsi: Для iSCSI-соединений (не всегда требуется, но часто устанавливается).

4. В процессе установки postfix может запросить конфигурацию. Выберите **“Internet Site”** и укажите системный hostname (например, debian-proxmox-host).

Шаг 4: Настройка Загрузчика GRUB

1. Proxmox VE использует специфичные настройки загрузчика. Замените запись в GRUB:

sudo nano /etc/default/grub

2. Найдите строку GRUB_CMDLINE_LINUX_DEFAULT="quiet" и измените её на:
3. GRUB_CMDLINE_LINUX_DEFAULT="quiet intel_iommu=on iommu=pt" # Для Intel
4. # Или для AMD:

GRUB_CMDLINE_LINUX_DEFAULT="quiet amd_iommu=on iommu=pt"

- intel_iommu=on / amd_iommu=on и iommu=pt важны для работы виртуализации в VirtualBox.

5. Сохраните файл (Ctrl+O, Enter, Ctrl+X).
6. Обновите конфигурацию GRUB:

sudo update-grub

Шаг 5: Перезагрузка и Доступ к Веб-интерфейсу Proxmox

1. Перезагрузите систему:

sudo reboot

2. После перезагрузки, **выйдите из SSH-сессии**.
3. На вашей рабочей машине откройте браузер и перейдите по адресу https://<IP_адрес_вашей_Debian_VM>:8006.
    - *Пример:* https://192.168.56.101:8006
4. При предупреждении о сертификате, нажмите “Дополнительно” -> “Перейти на … (не безопасно)”.
5. Введите:
    - **Username:** root
    - **Password:** Пароль root, который вы задали при установке Debian.
    - **Realm:** Proxmox VE authentication server.
6. Нажмите “Login”.

Практикум:

- **Задание 1.2.1:** Подключитесь к debian-proxmox-host по SSH.
- **Задание 1.2.2:** Добавьте репозиторий Proxmox VE (pve-install-repo.list).
- **Задание 1.2.3:** Обновите систему и установите proxmox-ve.
- **Задание 1.2.4:** Настройте GRUB (/etc/default/grub) для поддержки IOMMU.
- **Задание 1.2.5:** Перезагрузитесь и войдите в веб-интерфейс Proxmox VE (https://<IP_адрес_VM>:8006).

Модуль 1.3: Развертывание Debian VM в Proxmox VE

Цель: Создать виртуальную машину с ОС Debian внутри Proxmox VE для дальнейшей работы.

Предварительные требования:

1. Успешно работающий Proxmox VE и доступ к веб-интерфейсу.
2. ISO-образ Debian Server (amd64-netinst.iso), загруженный в Proxmox.
3. SSH-клиент на вашей рабочей станции.

Шаг 1: Загрузка ISO-образа Debian в Proxmox

1. В веб-интерфейсе Proxmox VE выберите ваш узел Proxmox (proxmox-node-1).
2. Перейдите в Storage.
3. Выберите ваше основное хранилище (например, local).
4. Нажмите **Upload**, выберите ISO-образ Debian и дождитесь завершения.

Шаг 2: Создание Виртуальной Машины (VM)

1. Нажмите **Create VM**.
2. **General:**
    - **Node:** Ваш узел Proxmox (proxmox-node-1).
    - **Name:** debian-devops-node-1
    - Нажмите **Next**.
3. **OS:**
    - **ISO Image:** Выберите хранилище и ваш ISO-образ Debian.
    - **Guest OS:** Linux, Version: Debian 11 (Bullseye).
    - Нажмите **Next**.

4. **System:**
    - **SCSI Controller:** VirtIO SCSI single.
    - Нажмите **Next**.
5. **Hard Disk:**
    - **Storage:** Выберите хранилище (например, local-lvm).
    - **Disk size (GiB):** 30.
    - Нажмите **Next**.
6. **CPU:**
    - **Cores:** 2.
    - **Type:** host.
    - Нажмите **Next**.
7. **Memory:**
    - **Memory (MiB):** 2048 (2 ГБ).
    - Нажмите **Next**.
8. **Network:**
    - **Bridge:** vmbr0 (это стандартный сетевой мост Proxmox, который соединяет ВМ с сетью, которую настроил сам Proxmox).
    - **Model:** VirtIO (paravirtualized).
    - Нажмите **Next**.
9. **Confirm:** Проверьте настройки и нажмите **Finish**.

Шаг 3: Установка Debian на VM

1. Выберите debian-devops-node-1, нажмите **Start**, затем **Console**.
2. Пройдите установку Debian:

- Язык, местоположение, клавиатура: English, United States, American English.
- **Сеть:**
  - **Hostname:** debian-devops-node-1
  - **Configure network automatically:** Yes. Proxmox VE будет выдавать IP-адреса из своей сети vmbr0. **ЗАПИШИТЕ IP-адрес**, который получит ВМ (он будет в той же подсети, что и ваш Proxmox сервер, но отличаться).
- **Пароли:** Задайте надежные пароли для root и пользователя devops_user.
- **Partition disks:** Guided - use entire disk.
- **Software selection:** Выберите SSH server и Standard system utilities.
- **GRUB boot loader:** Установите на /dev/sda.

3. После завершения установки перезагрузите ВМ.

Шаг 4: Отключение ISO-образа

1. В Proxmox: debian-devops-node-1 -> Hardware -> CD/DVD Drive.
2. Нажмите **Edit**, выберите **Do not use any media**. Нажмите **OK**.

Шаг 5: Определение IP-адреса VM и Проверка SSH-доступа

1. Подключитесь к консоли ВМ, выполните ip a и запишите IP-адрес eth0 (или ens18).
2. На вашей рабочей станции откройте терминал.
3. Подключитесь по SSH: ssh devops_user@<IP_адрес_вашей_Debian_VM> (например, ssh devops_user@192.168.1.101).

4. Введите пароль devops_user. При первом подключении введите yes.
5. Если увидели командную строку, SSH-доступ работает.

Практикум:

- **Задание 1.3.1:** Загрузите ISO Debian в Proxmox.
- **Задание 1.3.2:** Создайте VM debian-devops-node-1 в Proxmox с указанными параметрами.
- **Задание 1.3.3:** Установите Debian, запишите IP-адрес ВМ и пароли, отключите ISO.
- **Задание 1.3.4:** Проверьте SSH-доступ к debian-devops-node-1 с вашей рабочей машины.

Модуль 1.4: Подключение к Git-Системе с Debian VM

Цель: Настроить безопасное подключение к GitHub с Debian VM с помощью SSH-ключей.

Предварительные требования:

- Успешно развернутая VM debian-devops-node-1 в Proxmox с SSH-доступом.
- Аккаунт на GitHub.

Шаг 1: Генерация SSH-ключа на Debian VM

1. Подключитесь к debian-devops-node-1 по SSH (ssh devops_user@<IP_адрес_вашей_Debian_VM>).
2. Выполните команду:

ssh-keygen -t rsa -b 4096 -C "ваш_email@example.com"

- Замените ваш_email@example.com на ваш email.

3. На запрос пути к файлу нажмите Enter.
4. Введите **надежную парольную фразу** и подтвердите её.

Шаг 2: Проверка прав доступа к ключам

1. Выполните: ls -la ~/.ssh
2. Убедитесь, что права на id_rsa — -rw-------, на id_rsa.pub — -rw-r--r--, на .ssh — drwx------.
3. Если нет, исправьте:

chmod 700 ~/.ssh

chmod 600 ~/.ssh/id_rsa

chmod 644 ~/.ssh/id_rsa.pub

Шаг 3: Копирование Публичного Ключа

1. Выполните: cat ~/.ssh/id_rsa.pub
2. Скопируйте весь вывод (начинается с ssh-rsa ...).

Шаг 4: Добавление Ключа в GitHub

1. На GitHub: Profile icon -> Settings -> SSH and GPG keys.
2. Нажмите “New SSH key”.
3. **Title:** Debian VM Key
4. **Key:** Вставьте скопированный публичный ключ.
5. Нажмите “Add SSH key”.

Шаг 5: Тестирование Подключения к GitHub

1. В терминале Debian VM выполните: ssh -T git@github.com
2. Введите yes (если нужно), затем вашу SSH-парольную фразу.

3. Вы должны увидеть сообщение Hi ваш_аккаунт! You've successfully authenticated...

Шаг 6: Клонирование Репозитория

1. На GitHub скопируйте SSH-URL вашего репозитория (например, git@github.com:ваш_аккаунт/репозиторий.git).
2. В терминале Debian VM выполните:

git clone git@github.com:ваш_аккаунт/репозиторий.git

3. Введите вашу SSH-парольную фразу, если потребуется.

Практикум:

- **Задание 1.4.1:** Сгенерируйте SSH-ключ на Debian VM, проверьте права.
- **Задание 1.4.2:** Скопируйте публичный ключ и добавьте его в GitHub.
- **Задание 1.4.3:** Протестируйте подключение к GitHub (ssh -T git@github.com).
- **Задание 1.4.4:** Клонируйте ваш репозиторий на Debian VM.

Модуль 1.5: Установка Docker на Debian VM

Цель: Установить Docker Engine на вашу Debian VM для работы с контейнерами.

Предварительные требования:

- Успешно настроенный SSH-доступ к debian-devops-node-1.

Шаг 1: Обновление списка пакетов

1. Подключитесь к debian-devops-node-1 по SSH.

2. Выполните:

```
sudo apt update
```

Шаг 2: Установка необходимых пакетов

1. Выполните:

```
sudo apt install apt-transport-https ca-certificates curl software-properties-common gnupg lsb-release -y
```

Шаг 3: Добавление GPG-ключа Docker

1. Выполните:

```
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

Шаг 4: Добавление репозитория Docker

1. Выполните:

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/debian \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Шаг 5: Установка Docker Engine

1. Обновите список пакетов снова:

```
sudo apt update
```

2. Установите Docker:

```
sudo apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin -y
```

Шаг 6: Добавление пользователя в группу docker

1. Чтобы запускать Docker без sudo, добавьте пользователя:

sudo usermod -aG docker $USER

2. **ВАЖНО:** Для применения изменений **выйдите из SSH-сессии и подключитесь заново.**

Шаг 7: Проверка установки Docker

1. После переподключения к debian-devops-node-1, выполните:

docker --version

docker run hello-world

2. Должен появиться текст приветствия от Docker.

Практикум:

- **Задание 1.5.1:** Установите Docker на debian-devops-node-1, следуя шагам.
- **Задание 1.5.2:** Добавьте пользователя devops_user в группу docker, переподключитесь и проверьте работу Docker, запустив docker run hello-world.

Модуль 1.6: Сборка Docker-образа Вашего Приложения

Цель: Научиться создавать Docker-образ для вашего приложения с помощью Dockerfile.

Предварительные требования:

- Установленный Docker на debian-devops-node-1.

- Простой проект (например, статичный HTML-файл или простой Python-скрипт).

Шаг 1: Создание Директории для Приложения

1. В вашей debian-devops-node-1 создайте директорию:

```
mkdir my_app && cd my_app
```

2. Создайте простой файл:
    - Для веб-сервера: echo "<h1>Hello from Docker Container!</h1>" > index.html
    - Для Python: echo "print('Hello from Docker Python!')" > app.py

Шаг 2: Создание Dockerfile

1. Создайте файл Dockerfile в этой директории: nano Dockerfile
2. **Пример Dockerfile для статического сайта (Nginx):**

```
FROM nginx:latest

COPY index.html /usr/share/nginx/html/

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

3. **Пример Dockerfile для Python:**

```
FROM python:3.9-slim

WORKDIR /app

COPY app.py .

CMD ["python", "app.py"]
```

Шаг 3: Сборка Docker-образа

1. В директории с Dockerfile выполните:

```
docker build -t my-app-image:v1 .
```

   - my-app-image:v1 – имя и тег вашего образа.

Шаг 4: Запуск Контейнера из Образа

1. Запустите контейнер (пример для Nginx):

```
docker run -d -p 8081:80 --name my-app-container my-app-image:v1
```

   - -p 8081:80 – пробрасывает порт 8081 вашей Debian VM на порт 80 контейнера.

2. Для Python-приложения (если порт 5000):

```
docker run -d -p 5000:5000 --name my-app-container my-app-image:v1
```

Шаг 5: Проверка Работы

1. Откройте браузер и перейдите
   на http://<IP_адрес_вашей_Debian_VM>:8081 (для Nginx)
   или http://<IP_адрес_вашей_Debian_VM>:5000 (для Python).
2. Проверьте список контейнеров: docker ps

Практикум:

- **Задание 1.6.1:** Создайте директорию my_app на debian-devops-node-1, добавьте index.html или app.py.
- **Задание 1.6.2:** Создайте соответствующий Dockerfile.
- **Задание 1.6.3:** Соберите образ (docker build) и запустите контейнер (docker run).
- **Задание 1.6.4:** Проверьте доступность приложения через IP вашей Debian VM.

Часть 2: Автоматизация CI/CD с GitHub Actions

Модуль 2.1: Настройка GitHub Secrets

Цель: Безопасное хранение учетных данных и ключей для использования в GitHub Actions.

Предварительные требования:

- Репозиторий вашего проекта на GitHub.
- Аккаунт на Docker Hub.
- SSH-ключ без парольной фразы, сгенерированный на вашей рабочей машине (для развертывания).

Шаг 1: Docker Hub Учетные Данные

1. Зарегистрируйтесь на Docker Hub (hub.docker.com).
2. Создайте Access Token в настройках Docker Hub (Security -> Access Tokens).
3. В репозитории GitHub: Settings -> Secrets and variables -> Actions.
4. Добавьте секреты:
    - **Name:** DOCKERHUB_USERNAME
    - **Secret:** Ваш логин на Docker Hub.
    - **Name:** DOCKERHUB_TOKEN
    - **Secret:** Ваш Docker Hub Access Token.

Шаг 2: SSH-ключ для Развертывания

1. На вашей рабочей машине сгенерируйте SSH-ключ **без парольной фразы**:

ssh-keygen -t rsa -b 4096 -C "github_actions_deploy_key" -f ~/.ssh/github_actions_deploy_key

2. Скопируйте содержимое файла ~/.ssh/github_actions_deploy_key.pub.
3. Добавьте этот публичный ключ в GitHub: Settings -> SSH and GPG keys -> New SSH key (Title: GitHub Actions Deploy Key).
4. Добавьте приватный ключ (~/.ssh/github_actions_deploy_key) в Secrets GitHub как SSH_PRIVATE_KEY.

Практикум:

- **Задание 2.1.1:** Создайте учетные данные Docker Hub и добавьте их в Secrets GitHub.
- **Задание 2.1.2:** Сгенерируйте SSH-ключ без парольной фразы для развертывания, добавьте публичный ключ в GitHub и приватный ключ в Secrets GitHub.

Модуль 2.2: Создание CI/CD Пайплайна с GitHub Actions

Цель: Автоматизировать сборку, тестирование, сборку Docker-образа и развертывание приложения.

Предварительные требования:

- Настроенные Secrets на GitHub.
- Репозиторий проекта на GitHub.
- Рабочая Debian VM (debian-devops-node-1) с установленным Docker.

Шаг 1: Создание Workflow-файла

1. В корне вашего репозитория создайте директорию .github/workflows/.

2. Внутри создайте файл main.yml.

Шаг 2: Написание YAML-кода Workflow

```
name: CI/CD Pipeline


on:
  push:
    branches: [ main ] # Запускать при push в main
  pull_request:
    branches: [ main ] # Запускать при pull request в main


jobs:
  build-and-test:
    runs-on: ubuntu-latest # Среда выполнения
    steps:
    - name: Checkout code
      uses: actions/checkout@v3 # Скачать код проекта


    - name: Set up Python 3.9 # Пример для Python, адаптируйте под ваш язык
      uses: actions/setup-python@v3
      with:
        python-version: '3.9'


    - name: Install dependencies and test
```

```
    run: |
      python -m pip install --upgrade pip
      pip install flake8 pytest # Пример зависимостей
      flake8 . # Проверка стиля кода
      pytest # Запуск тестов

  docker-build-push:
    needs: build-and-test # Запускать после build-and-test
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Set up Docker Buildx
      uses: docker/setup-buildx-action@v2

    - name: Log in to Docker Hub
      uses: docker/login-action@v2
      with:
        username: ${{ secrets.DOCKERHUB_USERNAME }}
        password: ${{ secrets.DOCKERHUB_TOKEN }}

    - name: Build and push Docker image
```

```
    uses: docker/build-push-action@v4
    with:
      context: . # Путь к Dockerfile
      push: true
      tags: ${{ secrets.DOCKERHUB_USERNAME }}/my-app:latest # ВАШ ОБРАЗ!

  deploy-to-vm:
    needs: docker-build-push # Запускать после docker-build-push
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Set up SSH Agent
      uses: webfactory/ssh-agent@v0.7.0
      with:
        ssh-private-key: ${{ secrets.SSH_PRIVATE_KEY }}

    - name: Add host key to known_hosts
      run: ssh-keyscan <IP_адрес_вашей_Debian_VM> >> ~/.ssh/known_hosts # Убедиться, что IP ВМ известен
```

```
- name: Deploy to server
  env:
    SSH_HOST: <IP_адрес_вашей_Debian_VM>
    SSH_USER: devops_user
  run: |
    ssh ${{ env.SSH_HOST }} "
      cd ~/my_app && # Перейти в директорию приложения
      docker stop my-app-container || true && # Остановить старый контейнер, если есть
      docker rm my-app-container || true &&   # Удалить старый контейнер
      docker pull ${{ secrets.DOCKERHUB_USERNAME }}/my-app:latest && # Скачать новый образ
      docker run -d -p 8081:80 --name my-app-container ${{ secrets.DOCKERHUB_USERNAME }}/my-app:latest # Запустить новый контейнер
    "
```

- **ВАЖНО:**
  - Адаптируйте build-and-test под ваш язык и инструменты тестирования.
  - Укажите правильный путь к вашему Dockerfile в docker-build-push.
  - Замените <IP_адрес_вашей_Debian_VM>, devops_user, ~/my_app, 8081:80 на ваши реальные значения.
  - Используйте ваш образ Docker Hub (${{ secrets.DOCKERHUB_USERNAME }}/my-app:latest).

Шаг 3: Загрузка и Проверка

1. Закоммитьте файл main.yml в .github/workflows/.
2. Перейдите на вкладку “Actions” в GitHub, чтобы увидеть выполнение пайплайна.

Практикум:

- **Задание 2.2.1:** Создайте директорию .github/workflows/ и файл main.yml.
- **Задание 2.2.2:** Напишите код Workflow, включающий job’ы build-and-test, docker-build-push, deploy-to-vm.
- **Задание 2.2.3:** Закоммитьте файл и проверьте успешность выполнения всего пайплайна. Убедитесь, что ваше приложение обновляется на Debian VM.

Часть 3: Оркестрация Контейнеров с Kubernetes

Модуль 3.1: Установка Kubernetes (Minikube) и kubectl

Цель: Развернуть локальный Kubernetes-кластер с помощью Minikube и установить инструмент управления kubectl.

Предварительные требования:

1. **Рабочая станция:** Linux, macOS или Windows.
2. **Docker:** Установлен на рабочей станции.
3. **SSH-клиент:** Установлен.

Шаг 1: Установка kubectl

1. **Linux/macOS:**

curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl" *# Для Linux*

 # curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/darwin/amd64/kubectl" # Для macOS

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

rm kubectl

2. **Windows:** Скачайте kubectl.exe с kubernetes.io/docs/tasks/tools/install-kubectl-windows/ и добавьте в PATH.
3. Проверьте установку: kubectl version --client

Шаг 2: Установка Minikube

1. **Linux/macOS:**

curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 *# Для Linux*

 # curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-darwin-amd64 # Для macOS

sudo install minikube-linux-amd64 /usr/local/bin/minikube

rm minikube-linux-amd64

2. **Windows:** Скачайте minikube-windows-amd64.exe с minikube.sigs.k8s.io/docs/start/ и добавьте в PATH.
3. Проверьте установку: minikube version

Шаг 3: Запуск Minikube

1. В терминале выполните:

minikube start --driver=docker

- --driver=docker указывает использовать Docker как бэкенд для Minikube.

Шаг 4: Проверка Статуса Кластера

1. Проверьте статус Minikube: minikube status
2. Проверьте kubectl: kubectl cluster-info

Практикум:

- **Задание 3.1.1:** Установите kubectl и minikube.
- **Задание 3.1.2:** Запустите Minikube (minikube start --driver=docker).
- **Задание 3.1.3:** Проверьте kubectl, убедившись, что он работает с Minikube (kubectl cluster-info).

Модуль 3.2: Развертывание Приложения в Kubernetes

Цель: Развернуть Docker-образ приложения в Minikube с помощью Kubernetes-манифестов.

Предварительные требования:

- Успешно запущенный Minikube.
- Ваш Docker-образ, опубликованный на Docker Hub.

Шаг 1: Создание Kubernetes-манифестов

1. В корне вашего проекта создайте директорию k8s/.

2. Создайте k8s/deployment.yml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app-deployment
  labels:
    app: my-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app-container
        image: ваш_dockerhub_username/my-app:latest # ВАШ ОБРАЗ!
        ports:
        - containerPort: 80 # Порт вашего приложения в контейнере
```

- Замените ваш_dockerhub_username/my-app:latest на ваш Docker Hub образ.

3. Создайте k8s/service.yml:

```
apiVersion: v1
kind: Service
metadata:
  name: my-app-service
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80 # Порт, который будет доступен извне кластера
      targetPort: 80 # Порт внутри контейнера
  type: NodePort # Для Minikube это лучший вариант
```

Шаг 2: Применение Манифестов

1. В директории k8s/ выполните:

```
kubectl apply -f deployment.yml
kubectl apply -f service.yml
```

Шаг 3: Проверка Развертывания

1. Проверьте статус Pod’ов: kubectl get pods
2. Проверьте статус Service: kubectl get services
3. Чтобы получить доступ к приложению, выполните:

minikube service my-app-service --url

Эта команда выдаст URL, по которому можно открыть ваше приложение.

Шаг 4: Автоматизация Обновления Репозитория Git

- **Задача:** После внесения изменений в код, сборки нового Docker-образа и его пуша на Docker Hub, нужно обновить deployment.yml в репозитории, указав новый тег образа, и закоммитить это изменение.
- **Решение:** GitHub Actions может автоматизировать этот процесс.
    i. **Скрипт обновления deployment.yml:** Создайте скрипт (например, scripts/update_k8s_image.sh) в вашем репозитории.

```
#!/bin/bash

IMAGE_NAME="$1"

K8S_DEPLOYMENT_FILE="k8s/deployment.yml"


git config --global user.name 'GitHub Actions'

git config --global user.email 'actions@github.com'


# Обновляем image в deployment.yml

sed -i "s|image:.*|image: ${IMAGE_NAME}|g" ${K8S_DEPLOYMENT_FILE}


# Добавляем и коммитим изменения

git add ${K8S_DEPLOYMENT_FILE}

if git diff --staged --quiet; then

  echo "No changes to commit."
```

else

git commit -m "Update Kubernetes image to ${IMAGE_NAME}"

 # Для отправки коммита, нужно настроить Git credentials в GitHub Actions (например, PAT)

 # git push origin main

fi

- **Интеграция в GitHub Actions:** В job docker-build-push после успешного пуша образа, добавьте шаг для выполнения этого скрипта и коммита изменений. Для этого понадобится настроить Git credentials в GitHub Actions.

Практикум:

- **Задание 3.2.1:** Создайте k8s/deployment.yml и k8s/service.yml с вашим Docker Hub образом.
- **Задание 3.2.2:** Разверните приложение в Minikube (kubectl apply -f ...).
- **Задание 3.2.3:** Получите URL приложения (minikube service my-app-service --url) и проверьте его работу.
- **Задание 3.2.4 (Автоматизация обновления):** Создайте скрипт scripts/update_k8s_image.sh. Интегрируйте его в GitHub Actions для автоматического обновления deployment.yml и коммита.

Модуль 3.3: Интеграция Kubernetes с GitHub Actions

Цель: Автоматизировать развертывание в Kubernetes из CI/CD пайплайна.

Предварительные требования:

- Успешно настроенный Minikube и kubectl.
- Успешный CI/CD пайплайн с публикацией Docker-образа.

- Секрет KUBE_CONFIG_DATA в GitHub Secrets.

Шаг 1: Настройка Секрета KUBE_CONFIG_DATA

1. Получите конфигурацию Minikube: minikube kubectl -- config view --raw
2. Скопируйте вывод.
3. В GitHub репозитории: Settings -> Secrets and variables -> Actions -> New repository secret.
4. **Name:** KUBE_CONFIG_DATA
5. **Secret:** Вставьте скопированную конфигурацию.

Шаг 2: Обновление Workflow (main.yml)

1. Добавьте новый job deploy-to-k8s в ваш файл .github/workflows/main.yml.

```
# ... (предыдущие jobs: build-and-test, docker-build-push) ...

  deploy-to-k8s:

    needs: docker-build-push # Запускать после публикации образа

    runs-on: ubuntu-latest

    steps:

    - name: Checkout code

      uses: actions/checkout@v3


    - name: Set up Kubectl

      uses: azure/setup-kubectl@v3
```

```
    env:
      KUBE_CONFIG_DATA: ${{ secrets.KUBE_CONFIG_DATA }} # Используем секрет


  - name: Deploy to Kubernetes
    run: |
      # Обновляем образ в deployment.yml на актуальную версию
      sed -i "s|image: ваш_dockerhub_username/my-app:latest|image: ${{ secrets.DOCKERHUB_USERNAME }}/my-app:latest|g" k8s/deployment.yml
      # cat k8s/deployment.yml # Для отладки


      kubectl apply -f k8s/deployment.yml
      kubectl apply -f k8s/service.yml
```

- **ВАЖНО:** Убедитесь, что image: в deployment.yml соответствует вашему Docker Hub образу.

Шаг 3: Загрузка и Проверка

1. Закоммитьте обновленный main.yml и файлы k8s/deployment.yml, k8s/service.yml.
2. Перейдите на вкладку “Actions” в GitHub.
3. Проверьте статус Pod’ов и Service в Minikube: kubectl get pods, kubectl get services.
4. Получите URL для доступа к приложению: minikube service my-app-service --url.

Практикум:

- **Задание 3.3.1:** Настройте секрет KUBE_CONFIG_DATA.
- **Задание 3.3.2:** Добавьте job deploy-to-k8s в main.yml.
- **Задание 3.3.3:** Закоммитьте изменения и проверьте успешность развертывания в Minikube.

Часть 4: Управление Конфигурациями и IaC

Модуль 4.1: Основы Terraform

Цель: Научиться описывать и управлять инфраструктурой с помощью кода с использованием Terraform.

Предварительные требования:

- Рабочая станция с установленным Git.
- Терминал.

Шаг 1: Установка Terraform

1. Скачайте Terraform с terraform.io/downloads для вашей ОС.
2. Распакуйте архив.
3. Добавьте исполняемый файл terraform в директорию, находящуюся в вашем PATH (например, /usr/local/bin/ в Linux/macOS, или добавьте путь в PATH в Windows).

 # Пример для Linux/macOS

sudo mv terraform /usr/local/bin/

4. Проверьте установку: terraform version

Шаг 2: Создание Первой Terraform-Конфигурации (Пример с Docker)

1. Создайте новую директорию: mkdir terraform-docker-example && cd terraform-docker-example
2. Инициализируйте Git: git init
3. Создайте файл main.tf:

```
# Указываем провайдер Docker
provider "docker" {}

# Создаем Docker-образ Nginx
resource "docker_image" "nginx" {
  name = "nginx:latest"
  keep_local = false
}

# Создаем Docker-контейнер
resource "docker_container" "nginx_container" {
  name = "my-nginx-container-tf"
  image = docker_image.nginx.image_id
  ports {
    internal = 80
    external = 8080
  }
  depends_on = [docker_image.nginx]
}
```

Шаг 3: Инициализация Terraform

1. В директории проекта выполните:

terraform init

   - Эта команда загрузит провайдер Docker.

Шаг 4: Планирование и Применение

1. Посмотрите, что Terraform собирается сделать:

terraform plan

2. Примените изменения:

terraform apply

   - Введите yes для подтверждения.

3. Проверьте работу: Откройте http://localhost:8080 в браузере.

Шаг 5: Удаление Ресурсов

1. Удалите созданную инфраструктуру:

terraform destroy

   - Введите yes для подтверждения.

Практикум:

- **Задание 4.1.1:** Установите Terraform.
- **Задание 4.1.2:** Создайте директорию terraform-docker-example, main.tf с примером выше.
- **Задание 4.1.3:** Выполните terraform init, terraform plan, terraform apply.
- **Задание 4.1.4:** Проверьте работу контейнера Nginx.
- **Задание 4.1.5:** Удалите ресурсы с помощью terraform destroy.

Модуль 4.2: Основы Ansible

Цель: Автоматизировать настройку серверов и установку ПО с помощью Ansible.

Предварительные требования:

- Рабочая станция с установленным Git.
- Успешный SSH-доступ к debian-devops-node-1.
- Установленный Ansible на вашей рабочей станции.

Шаг 1: Установка Ansible

1. **Linux (Debian/Ubuntu):**

sudo apt update && sudo apt install ansible -y

2. **macOS:**

brew install ansible

3. Проверьте установку: ansible --version

Шаг 2: Создание Inventary-файла

1. Создайте директорию для Ansible: mkdir ansible-project && cd ansible-project
2. Создайте файл inventory.ini:

[debian_servers]

debian-devops-node-1 ansible_host=<IP_адрес_вашей_Debian_VM> ansible_user=devops_user

   - Замените <IP_адрес_вашей_Debian_VM> на реальный IP.

Шаг 3: Создание Playbook для Установки Nginx

1. Создайте файл nginx_install.yml:

```
---
- name: Install Nginx on Debian
  hosts: debian_servers
  become: yes # Использовать sudo

  tasks:
  - name: Update apt cache
    apt:
      update_cache: yes

  - name: Install Nginx
    apt:
      name: nginx
      state: present

  - name: Ensure Nginx is running and enabled
    service:
      name: nginx
      state: started
      enabled: yes
```

Шаг 4: Запуск Playbook

1. Выполните:

ansible-playbook -i inventory.ini nginx_install.yml

- Ansible подключится по SSH, выполнит задачи, возможно, запросит SSH-парольную фразу.

Шаг 5: Проверка Работы

1. Откройте браузер: http://<IP_адрес_вашей_Debian_VM>. Вы должны увидеть страницу Nginx.

Практикум:

- **Задание 4.2.1:** Установите Ansible.
- **Задание 4.2.2:** Создайте inventory.ini для вашей Debian VM.
- **Задание 4.2.3:** Создайте nginx_install.yml и запустите его.
- **Задание 4.2.4:** Проверьте, что Nginx установлен и работает на вашей Debian VM.

Часть 3: Интеграция Инструментов (CI/CD + Kubernetes)

Модуль 3.1: Установка Kubernetes (Minikube) и kubectl

Цель: Развернуть локальный Kubernetes-кластер с помощью Minikube и установить инструмент управления kubectl.

Предварительные требования:

1. **Рабочая станция:** Linux, macOS или Windows.
2. **Docker:** Установлен на рабочей станции.
3. **SSH-клиент:** Установлен.

Шаг 1: Установка kubectl

1. **Linux/macOS:**

curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl" *# Для Linux*

 # curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/darwin/amd64/kubectl" # Для macOS

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

rm kubectl

2. **Windows:** Скачайте kubectl.exe с kubernetes.io/docs/tasks/tools/install-kubectl-windows/ и добавьте в PATH.
3. Проверьте установку: kubectl version --client

Шаг 2: Установка Minikube

1. **Linux/macOS:**

curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 *# Для Linux*

 # curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-darwin-amd64 # Для macOS

sudo install minikube-linux-amd64 /usr/local/bin/minikube

rm minikube-linux-amd64

2. **Windows:** Скачайте minikube-windows-amd64.exe с minikube.sigs.k8s.io/docs/start/ и добавьте в PATH.
3. Проверьте установку: minikube version

Шаг 3: Запуск Minikube

1. В терминале выполните:

minikube start --driver=docker

- --driver=docker указывает использовать Docker как бэкенд для Minikube.

Шаг 4: Проверка Статуса Кластера

1. Проверьте статус Minikube: minikube status
2. Проверьте kubectl: kubectl cluster-info

Практикум:

- **Задание 3.1.1:** Установите kubectl и minikube.
- **Задание 3.1.2:** Запустите Minikube (minikube start --driver=docker).
- **Задание 3.1.3:** Проверьте kubectl, убедившись, что он работает с Minikube (kubectl cluster-info).

Модуль 3.2: Развертывание Приложения в Kubernetes

Цель: Развернуть Docker-образ приложения в Minikube с помощью Kubernetes-манифестов.

Предварительные требования:

- Успешно запущенный Minikube.
- Ваш Docker-образ, опубликованный на Docker Hub.

Шаг 1: Создание Kubernetes-манифестов

1. В корне вашего проекта создайте директорию k8s/.
2. Создайте k8s/deployment.yml:

apiVersion: apps/v1

kind: Deployment

```
metadata:
  name: my-app-deployment
  labels:
    app: my-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app-container
        image: ваш_dockerhub_username/my-app:latest # ВАШ ОБРАЗ!
        ports:
        - containerPort: 80 # Порт вашего приложения в контейнере
```

- Замените ваш_dockerhub_username/my-app:latest на ваш Docker Hub образ.

3. Создайте k8s/service.yml:

```
apiVersion: v1
```

```
kind: Service
metadata:
  name: my-app-service
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80 # Порт, который будет доступен извне кластера
      targetPort: 80 # Порт внутри контейнера
  type: NodePort # Для Minikube это лучший вариант
```

Шаг 2: Применение Манифестов

1. В директории k8s/ выполните:

```
kubectl apply -f deployment.yml
kubectl apply -f service.yml
```

Шаг 3: Проверка Развертывания

1. Проверьте статус Pod’ов: kubectl get pods
2. Проверьте статус Service: kubectl get services
3. Чтобы получить доступ к приложению, выполните:

```
minikube service my-app-service --url
```

Эта команда выдаст URL, по которому можно открыть ваше приложение.

Шаг 4: Автоматизация Обновления Репозитория Git

- **Задача:** После внесения изменений в код, сборки нового Docker-образа и его пуша на Docker Hub, нужно обновить deployment.yml в репозитории, указав новый тег образа, и закоммитить это изменение.
- **Решение:** GitHub Actions может автоматизировать этот процесс.

Скрипт обновления deployment.yml: Создайте скрипт (например, scripts/update_k8s_image.sh) в вашем репозитории.

```
#!/bin/bash

IMAGE_NAME="$1"

K8S_DEPLOYMENT_FILE="k8s/deployment.yml"


git config --global user.name 'GitHub Actions'

git config --global user.email 'actions@github.com'


# Обновляем image в deployment.yml

sed -i "s|image:.*|image: ${IMAGE_NAME}|g" ${K8S_DEPLOYMENT_FILE}


# Добавляем и коммитим изменения

git add ${K8S_DEPLOYMENT_FILE}

if git diff --staged --quiet; then

  echo "No changes to commit."

else

  git commit -m "Update Kubernetes image to ${IMAGE_NAME}"
```

 # Для отправки коммита, нужно настроить Git credentials в GitHub Actions (например, PAT)

 # git push origin main

fi

Интеграция в GitHub Actions: В job docker-build-push после успешного пуша образа, добавьте шаг для выполнения этого скрипта и коммита изменений. Потребуется настроить Git credentials (например, с помощью Personal Access Token GitHub) в GitHub Actions для выполнения git push.

Практикум:

- **Задание 3.2.1:** Создайте k8s/deployment.yml и k8s/service.yml с вашим Docker Hub образом.
- **Задание 3.2.2:** Разверните приложение в Minikube (kubectl apply -f ...).
- **Задание 3.2.3:** Получите URL приложения (minikube service my-app-service --url) и проверьте его работу.
- **Задание 3.2.4 (Автоматизация обновления):** Создайте скрипт scripts/update_k8s_image.sh. Интегрируйте его в GitHub Actions для автоматического обновления deployment.yml и коммита.

Модуль 3.3: Интеграция Kubernetes с GitHub Actions

Цель: Автоматизировать развертывание в Kubernetes из CI/CD пайплайна.

Предварительные требования:

- Успешно настроенный Minikube и kubectl.
- Успешный CI/CD пайплайн с публикацией Docker-образа.

- Секрет KUBE_CONFIG_DATA в GitHub Secrets.

Шаг 1: Настройка Секрета KUBE_CONFIG_DATA

1. Получите конфигурацию Minikube: minikube kubectl -- config view --raw
2. Скопируйте вывод.
3. В GitHub репозитории: Settings -> Secrets and variables -> Actions -> New repository secret.
4. **Name:** KUBE_CONFIG_DATA
5. **Secret:** Вставьте скопированную конфигурацию.

Шаг 2: Обновление Workflow (main.yml)

1. Добавьте новый job deploy-to-k8s в ваш файл .github/workflows/main.yml.

```
# ... (предыдущие jobs: build-and-test, docker-build-push) ...


  deploy-to-k8s:
    needs: docker-build-push # Запускать после публикации образа
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v3


    - name: Set up Kubectl
      uses: azure/setup-kubectl@v3
```

```
    env:
      KUBE_CONFIG_DATA: ${{ secrets.KUBE_CONFIG_DATA }} # Используем секрет


  - name: Deploy to Kubernetes
    run: |
      # Обновляем образ в deployment.yml на актуальную версию
      sed -i "s|image: ваш_dockerhub_username/my-app:latest|image: ${{ secrets.DOCKERHUB_USERNAME }}/my-app:latest|g" k8s/deployment.yml
      # cat k8s/deployment.yml # Для отладки


      kubectl apply -f k8s/deployment.yml
      kubectl apply -f k8s/service.yml
```

- **ВАЖНО:** Убедитесь, что image: в deployment.yml соответствует вашему Docker Hub образу.

Шаг 3: Загрузка и Проверка

1. Закоммитьте обновленный main.yml и файлы k8s/deployment.yml, k8s/service.yml.
2. Перейдите на вкладку “Actions” в GitHub.
3. Проверьте статус Pod’ов и Service в Minikube: kubectl get pods, kubectl get services.
4. Получите URL для доступа к приложению: minikube service my-app-service --url.

Практикум:

- **Задание 3.3.1:** Настройте секрет KUBE_CONFIG_DATA.
- **Задание 3.3.2:** Добавьте job deploy-to-k8s в main.yml.
- **Задание 3.3.3:** Закоммитьте изменения и проверьте успешность развертывания в Minikube.

Часть 4: Управление Конфигурациями и IaC

Модуль 4.1: Основы Terraform

Цель: Научиться описывать и управлять инфраструктурой с помощью кода с использованием Terraform.

Предварительные требования:

- Рабочая станция с установленным Git.
- Терминал.

Шаг 1: Установка Terraform

1. Скачайте Terraform с terraform.io/downloads для вашей ОС.
2. Распакуйте архив.
3. Добавьте исполняемый файл terraform в директорию, находящуюся в вашем PATH (например, /usr/local/bin/ в Linux/macOS, или добавьте путь в PATH в Windows).

 # Пример для Linux/macOS

sudo mv terraform /usr/local/bin/

4. Проверьте установку: terraform version

Шаг 2: Создание Первой Terraform-Конфигурации (Пример с Docker)

1. Создайте новую директорию: mkdir terraform-docker-example && cd terraform-docker-example
2. Инициализируйте Git: git init
3. Создайте файл main.tf:

```
# Указываем провайдер Docker
provider "docker" {}

# Создаем Docker-образ Nginx
resource "docker_image" "nginx" {
  name = "nginx:latest"
  keep_local = false
}

# Создаем Docker-контейнер
resource "docker_container" "nginx_container" {
  name = "my-nginx-container-tf"
  image = docker_image.nginx.image_id
  ports {
    internal = 80
    external = 8080
  }
  depends_on = [docker_image.nginx]
}
```

Шаг 3: Инициализация Terraform

1. В директории проекта выполните:

terraform init

   - Эта команда загрузит провайдер Docker.

Шаг 4: Планирование и Применение

1. Посмотрите, что Terraform собирается сделать:

terraform plan

2. Примените изменения:

terraform apply

   - Введите yes для подтверждения.

3. Проверьте работу: Откройте http://localhost:8080 в браузере.

Шаг 5: Удаление Ресурсов

1. Удалите созданную инфраструктуру:

terraform destroy

   - Введите yes для подтверждения.

Практикум:

- **Задание 4.1.1:** Установите Terraform.
- **Задание 4.1.2:** Создайте директорию terraform-docker-example, main.tf с примером выше.
- **Задание 4.1.3:** Выполните terraform init, terraform plan, terraform apply.
- **Задание 4.1.4:** Проверьте работу контейнера Nginx.
- **Задание 4.1.5:** Удалите ресурсы с помощью terraform destroy.

Модуль 4.2: Основы Ansible

Цель: Автоматизировать настройку серверов и установку ПО с помощью Ansible.

Предварительные требования:

- Рабочая станция с установленным Git.
- Успешный SSH-доступ к debian-devops-node-1.
- Установленный Ansible на вашей рабочей станции.

Шаг 1: Установка Ansible

1. **Linux (Debian/Ubuntu):**

sudo apt update && sudo apt install ansible -y

2. **macOS:**

brew install ansible

3. Проверьте установку: ansible --version

Шаг 2: Создание Inventary-файла

1. Создайте директорию для Ansible: mkdir ansible-project && cd ansible-project
2. Создайте файл inventory.ini:

[debian_servers]

debian-devops-node-1 ansible_host=<IP_адрес_вашей_Debian_VM> ansible_user=devops_user

   - Замените <IP_адрес_вашей_Debian_VM> на реальный IP.

Шаг 3: Создание Playbook для Установки Nginx

1. Создайте файл nginx_install.yml:

```
---
- name: Install Nginx on Debian
  hosts: debian_servers
  become: yes # Использовать sudo

  tasks:
  - name: Update apt cache
    apt:
      update_cache: yes

  - name: Install Nginx
    apt:
      name: nginx
      state: present

  - name: Ensure Nginx is running and enabled
    service:
      name: nginx
      state: started
      enabled: yes
```

Шаг 4: Запуск Playbook

1. Выполните:

ansible-playbook -i inventory.ini nginx_install.yml

    - Ansible подключится по SSH, выполнит задачи, возможно, запросит SSH-парольную фразу.

Шаг 5: Проверка Работы

1. Откройте браузер: http://<IP_адрес_вашей_Debian_VM>. Вы должны увидеть страницу Nginx.

Практикум:

- **Задание 4.2.1:** Установите Ansible.
- **Задание 4.2.2:** Создайте inventory.ini для вашей Debian VM.
- **Задание 4.2.3:** Создайте nginx_install.yml и запустите его.
- **Задание 4.2.4:** Проверьте, что Nginx установлен и работает на вашей Debian VM.