



МИНОБРНАУКИ РОССИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«МИРЭА – Российский технологический университет»

**РТУ МИРЭА**

---

## **ОТЧЕТ ПО ПРАКТИКЕ В ПРОФИЛЬНОЙ СФЕРЕ**

**Тема практики:** «Реализация CI/CD-конвейера для автоматизации развертывания веб-сервера на базе Vagrant, Ansible, Jenkins и GitHub»

Отчет представлен к  
рассмотрению:

Студент группы КТСО-02-23

«31» мая 2025 г.

\_\_\_\_\_  
(подпись)

Верченко А.А.

Отчет утвержден.

Допущен к защите:

Руководитель практики

«31» мая 2025 г.

\_\_\_\_\_  
(подпись)

Тарланов А.Т.

Москва 2025 г.



МИНОБРНАУКИ РОССИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«МИРЭА – Российский технологический университет»

**РТУ МИРЭА**

---

## ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ НА ПРАКТИКУ В ПРОФИЛЬНОЙ СФЕРЕ

Студенту 2 курса учебной группы КТСО-02-23

Верченко Артёму Александровичу

Время практики: с 26 мая 2025 г. по 31 мая 2024 г. \_\_\_\_\_

Должность на практике: студент \_\_\_\_\_

**1. ЦЕЛЕВАЯ УСТАНОВКА:** Реализация CI/CD-конвейера для автоматизации развертывания веб-сервера на базе Vagrant, Ansible, Jenkins и GitHub

### **2. СОДЕРЖАНИЕ ПРАКТИКИ:**

2.1 Введение

2.2 Реализация CI/CD-конвейера для автоматизации развертывания веб-сервера

2.3 Заключение

**3. ОРГАНИЗАЦИОННО-МЕТОДИЧЕСКИЕ УКАЗАНИЯ:** в процессе практики рекомендуется использовать периодические издания и отраслевую литературу годом издания не старше 10 лет.

### **СОГЛАСОВАНО:**

Руководитель практики

«26» мая 2025 г.

Задание получил

«26» мая 2025 г.

\_\_\_\_\_  
(подпись) (Тарланов А.Т.)

\_\_\_\_\_  
(подпись) (Верченко А.А.)



МИНОБРНАУКИ РОССИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«МИРЭА – Российский технологический университет»

**РТУ МИРЭА**

**РАБОЧИЙ ГРАФИК ПРОВЕДЕНИЯ  
практики в профильной сфере**

студента Верченко А.А. 2 курса группы КТСО-02-23.

№	Сроки выполнения	Этап	Отметка о выполнении
1	26.05.2025	Подготовительный этап, включающий в себя организационное собрание (Вводная лекция о порядке организации и прохождения практики)	выполнено
2	с 26.05.2025 по 29.05.2025	Выполнение задания по практике в соответствии с выданным заданием студента. (Мероприятия по сбору, обработке и структурированию материала, выполнение поставленной задачи)	выполнено
3	с 29.05.2025 по 30.05.2025	Подготовка отчета по практике (Оформление материалов отчета в полном соответствии с требованиями на оформление учебных работ студентов)	выполнено
4	31.05.2025	Представление отчета по практике к защите посредством загрузки на портал ДПО ( <a href="https://online-dpo.mirea.ru/">https://online-dpo.mirea.ru/</a> )	выполнено

**Согласовано:**

Руководитель практики \_\_\_\_\_/Тарланов А.Т./

Обучающийся \_\_\_\_\_/ Верченко А.А./

## Содержание

<b>Введение.....</b>	<b>5</b>
Цель проекта.....	5
Основные задачи.....	5
Ожидаемые результаты.....	5
<b>Обзор используемых технологий.....</b>	<b>7</b>
Виртуализация: VirtualBox и Vagrant.....	7
Управление конфигурациями: Ansible.....	7
CI/CD: Jenkins.....	7
Система контроля версий: GitHub.....	8
Веб-сервер: Nginx.....	8
<b>Архитектура проекта.....</b>	<b>9</b>
Описание VMs.....	9
VM_1: Jenkins-сервер.....	9
VM_2: Ansible-контроллер.....	9
VM_3: Веб-сервер (Nginx).....	9
Схема взаимодействия между компонентами.....	10
<b>Настройка VMs.....</b>	<b>12</b>
Развертывание VMs с помощью Vagrant.....	12
Процесс запуска.....	13
<b>Установка и настройка Jenkins (VM_1).....</b>	<b>15</b>
Установка JDK и зависимостей.....	15
Установка Jenkins.....	16
Первоначальная настройка Jenkins (плагины, пользователи).....	18
<b>Настройка Ansible (VM_2).....</b>	<b>20</b>
Установка Ansible.....	20
Создание inventory-файла (inventory.ini).....	20
Разработка плейбука (deploy.yml) для развертывания Nginx.....	21
Создание шаблона веб-страницы (index.html).....	23
Проверка работы Ansible.....	23
<b>Интеграция с GitHub.....</b>	<b>24</b>
Создание нового репозитория.....	24
Клонирование репозитория на VM_2.....	25
<b>Настройка CI/CD Pipeline в Jenkins.....</b>	<b>25</b>
Создание Jenkinsfile.....	25
Настройка Jenkins Pipeline.....	27
<b>Тестирование.....</b>	<b>31</b>
Процесс тестирования.....	31
<b>Проблемы.....</b>	<b>33</b>
<b>Заключение.....</b>	<b>34</b>
Итоги реализации проекта.....	34
Преимущества автоматизации.....	34

Приложение.....	36
-----------------	----

# Введение

## Цель проекта

Целью данного проекта является создание автоматизированного CI/CD-конвейера для развертывания и обновления веб-сервера с использованием инструментов DevOps. Проект демонстрирует интеграцию виртуализации (VirtualBox/Vagrant), управления конфигурациями (Ansible), непрерывной интеграции и доставки (Jenkins) и контроля версий (GitHub) для обеспечения эффективного процесса разработки и оперативного внедрения изменений.

## Основные задачи

- Развернуть изолированную инфраструктуру из трёх виртуальных машин (Jenkins, Ansible, Nginx) с помощью Vagrant.
- Настроить Jenkins для автоматизации сборки и управления рабочими процессами через Pipeline.
- Реализовать автоматическое развёртывание веб-сервера (Nginx) с использованием Ansible-плейбуков.
- Интегрировать GitHub-репозиторий в процесс CI/CD для динамического обновления контента веб-страницы.
- Обеспечить проверку работоспособности системы: автоматический перезапуск сервера при изменениях в репозитории.

## Ожидаемые результаты

- Полностью автоматизированный процесс обновления веб-сервера: при внесении изменений в файл `index.html` в GitHub Jenkins инициирует выполнение Ansible-плейбука, который разворачивает обновленную версию страницы на VM\_3.
- Минимизация ручного вмешательства: настройка "один раз" с последующей автономной работой конвейера.
- Демонстрация взаимодействия ключевых DevOps-инструментов для решения реальной задачи: от управления инфраструктурой до непрерывной доставки.

# Обзор используемых технологий

В основе проекта лежит комбинация инструментов, обеспечивающих автоматизацию и управление жизненным циклом разработки. Каждый из них был выбран для решения конкретных задач, а их интеграция позволила создать гибкую и воспроизводимую инфраструктуру.

## Виртуализация: VirtualBox и Vagrant

VirtualBox и Vagrant стали основой для виртуализации. VirtualBox предоставил возможность эмуляции изолированных виртуальных машин, что критически важно для тестирования и воспроизведения одинаковых окружений на разных этапах проекта. Vagrant дополнил его, автоматизируя процесс создания и настройки ВМ через декларативный файл **Vagrantfile**. Это позволило быстро развернуть три машины (Jenkins, Ansible и Nginx) с предопределенными IP-адресами и сетевыми настройками, минимизировав ручную работу.

## Управление конфигурациями: Ansible

Ansible был выбран для управления конфигурациями благодаря своей простоте и отсутствию необходимости установки агентов на целевые узлы. Используя YAML-плейбуки, таких как **deploy.yml**, Ansible автоматически настраивал веб-сервер Nginx на третьей виртуальной машине, а также обеспечивал загрузку актуальной версии файла **index.html** напрямую из GitHub-репозитория. Его идиоматичность гарантировала, что повторный запуск плейбука не нарушит уже работающую конфигурацию.

## CI/CD: Jenkins

Jenkins выступил в роли центрального звена CI/CD-цепочки. С помощью Jenkins-пайплайна, описанного в **Jenkinsfile**, удалось автоматизировать процесс развёртывания: при обнаружении изменений в GitHub-репозитории, я говорил Jenkins запускать задачи, которые, в свою очередь, активировали Ansible-плейбуки для обновления веб-сервера. Гибкость Jenkins и поддержка плагинов позволили легко интегрировать его с другими инструментами, такими как GitHub и Ansible.

## Система контроля версий: GitHub

GitHub стал хранилищем для всего кода проекта, включая конфигурации, плейбуки и статический контент веб-страницы. Использование веб-хуков обеспечило мгновенное оповещение Jenkins о новых коммитах в репозитории, что стало триггером для автоматического перезапуска пайплайна. Это позволило синхронизировать изменения в коде с актуальным состоянием инфраструктуры.

## Веб-сервер: Nginx

Nginx был развернут на третьей виртуальной машине в качестве веб-сервера. Его лёгкость, производительность и простая настройка сделали его идеальным выбором для хостинга статической HTML-страницы. Каждое обновление `index.html` через GitHub немедленно отражалось на веб-сервере благодаря связке Jenkins и Ansible, что наглядно демонстрировало работу всего CI/CD-цикла.



# Архитектура проекта

## Описание VMs

В рамках проекта были развернуты три виртуальные машины (VMs), каждая из которых выполняет узкоспециализированную роль в цепочке CI/CD.

### VM\_1: Jenkins-сервер

Эта машина выступает в качестве центра управления процессами непрерывной интеграции и доставки. На нее устанавливается Jenkins — инструмент, который автоматизирует запуск задач (сборка, тестирование, деплой) на основе изменений в коде. Jenkins работает как оркестратор: он реагирует на события (например, обновление репозитория GitHub), запускает pipeline и передает управление Ansible для выполнения следующих этапов.

### VM\_2: Ansible-контроллер

Данная VM выполняет роль управляющего узла для Ansible — инструмента управления конфигурациями. Её ключевая задача — автоматизировать настройку других машин в инфраструктуре. На VM\_2 хранятся плейбуки (например, `deploy.yml`), которые описывают, как развернуть Nginx на VM\_3, а также файл инвентаризации (`inventory.ini`), указывающий на IP-адрес целевой машины (VM\_3).

После доработки проекта VM\_2 перестал использовать локальный файл `index.html`, а вместо этого загружает его напрямую из GitHub-репозитория, что обеспечивает синхронизацию контента с актуальной версией кода.

### VM\_3: Веб-сервер (Nginx)

Это конечная точка развертывания. На VM\_3 устанавливается Nginx — высокопроизводительный веб-сервер, который обслуживает статический контент (например, HTML-страницу). После выполнения Ansible-плейбука с VM\_2 сервер автоматически конфигурируется: копируются файлы из репозитория, настраиваются виртуальные хосты, и служба Nginx перезапускается. Любые изменения в `index.html`, зафиксированные в

GitHub, через цепочку Jenkins → Ansible попадают на эту машину, что позволяет мгновенно отображать обновленный контент.

## Схема взаимодействия между компонентами

Архитектура проекта построена по принципу последовательной автоматизации, где каждый этап инициируется событием:

### 1. Запуск виртуальных машин:

С помощью Vagrant и VirtualBox одновременно поднимаются все три VM. На этом этапе обеспечивается сетевая связность между ними (например, через NAT или мостовой интерфейс), чтобы Jenkins (VM\_1) мог обращаться к Ansible (VM\_2), а Ansible — к веб-серверу (VM\_3).

### 2. Работа Jenkins (VM\_1):

Имеется настройка Jenkins-задачи (Pipeline), которая привязана к GitHub-репозиторию через веб-хук. Далее при каждом коммите в репозиторий GitHub отправляет уведомление Jenkins, что запускает выполнение Pipeline. А в рамках Pipeline выполняется **Jenkinsfile**, который определяет этапы:

- a. Сборка: Проверка наличия изменений в репозитории.
- b. Деплой: Передача управления на VM\_2 для запуска Ansible-плейбука.

### 3. Действия Ansible (VM\_2):

Ansible получает от Jenkins сигнал на выполнение сценария. После сигнала, используя **inventory.ini**, Ansible устанавливает SSH-соединение с VM\_3, а плейбук **deploy.yml** выполняет следующие шаги:

- a. Скачивает из указанного репозитория GitHub актуальный **index.html**.
- b. Копирует файлы из репозитория на VM\_3 в директорию Nginx (например, **/var/www/html/**).
- c. Перезагружает службу Nginx для применения изменений.

### 4. Итог на VM\_3:

Веб-сервер начинает отдавать обновленную версию **index.html**, и теперь можно убедиться в успешности деплоя, открыв веб-страницу по IP-адресу VM\_3.

### 5\*. Визуализация потока:

GitHub (изменения кода)

- Jenkins (VM\_1, инициализирует Pipeline)
- Ansible (VM\_2, выполняет плейбук)
- Nginx (VM\_3, обновляет контент)

Такая архитектура обеспечивает полный цикл CI/CD: от фиксации кода до автоматического обновления продакшн-окружения. Каждый компонент изолирован, что соответствует принципам DevOps (например, разделение обязанностей, масштабируемость).

# Настройка VMs

Для реализации проекта были созданы три виртуальные машины (VM) с помощью инструментов VirtualBox и Vagrant. Основной задачей этого этапа было обеспечить изолированную среду для работы Jenkins, Ansible и веб-сервера, а также настроить корректное сетевое взаимодействие между ними.

## Развертывание VMs с помощью Vagrant

Конфигурация виртуальных машин была определена в файле Vagrantfile. Для каждой VM заданы:

**Базовая ОС:** Использован образ Ubuntu 22.04 (Jammy Jellyfish) для обеспечения совместимости с инструментами проекта.

**Ресурсы:**

1. VM\_1 (Jenkins): 2 ГБ оперативной памяти, 2 ядра CPU.
2. VM\_2 (Ansible): 1 ГБ оперативной памяти, 1 ядро CPU.
3. VM\_3 (Nginx): 1 ГБ оперативной памяти, 1 ядро CPU.

Пример фрагмента Vagrantfile для VM\_1 (полный файл будет в приложении):

```
Vagrant.configure("2") do |config|
  config.vm.define "jenkins" do |jenkins|
    jenkins.vm.box = "ubuntu/focal64"
    jenkins.vm.network "private_network", ip:
"192.168.56.10"
    jenkins.vm.provider "virtualbox" do |vb|
      vb.memory = "2048"
    end
  end
  # Аналогично для VM_2 и VM_3...
end
```

**Объяснение:**

Содержимое Vagrantfile написано на Ruby. Оно строится построчно, что происходит:

1. `Vagrant.configure("2") do |config|`

С этого начинается настройка Vagrant. Цифра 2 означает, что будет использована 2-ая версия (актуальная). А в

переменной `config` будут храниться все параметры виртуальных машин.

2. `config.vm.define "jenkins" do |jenkins|`

Тут создается виртуальная машина с именем *jenkins*. Всё что будет внутри этого блока (`do |jenkins| ... end`), относится к этой машине.

3. `jenkins.vm.box = "ubuntu/focal64"`

Тут указывается, какой образ ОС будет использоваться. В моем случае **Ubuntu 22.04**.

4. `jenkins.vm.network "private_network", ip: "192.168.56.10"`

Тут идет настройка сети: `private_network` означает, что эта VM будет видна только в изолированной локальной сети (соответственно машины вне Vagrant, не увидят ее).

Такой способ подходит в моем случае, так как проект — учебный, а у меня нет лишних денег на аренду облака.

5. `jenkins.vm.provider "virtualbox" do |vb|`

В этой строке указывается, что в качестве системы виртуализации будет использоваться **VirtualBox**. И опять, всё внутри этого блока (`do |vb| ... end`) — настройки для **VirtualBox**.

6. `vb.memory = "2048"`

Идет выделение 2 ГБ оперативной памяти.

7. В этой строке и следующих, будут завершаться предыдущие начатые блоки. По очереди: сначала `vb`, потом `jenkins`, и наконец последний `end`, закрывающий общий файл конфигурации **Vagrant**.

Нужно отметить, что после завершения блока `jenkins`, я, в своем проекте, начал ещё блоки `ansible` и `web`, для машин **VM\_2** и **VM\_3**, соответственно. Однако у них такой же синтаксис, поэтому нет смысла разбирать. (полный **Jenkinsfile** в приложении).

## Процесс запуска

1. Инициализация VMs:

Команда:

```
vagrant up
```

## 2. Проверка статуса:

Команда:

```
vagrant status
```

Результат:

```
archik@archik-TM1613:~/Documents/devops-project_3$ vagrant status
Current machine states:

jenkins                running (virtualbox)
ansible                running (virtualbox)
web                    running (virtualbox)

This environment represents multiple VMs. The VMs are all listed
above with their current state. For more information about a specific
VM, run `vagrant status NAME`.
```

## 3. Проверка подключения к VMs через SSH (например, для VM\_1):

Команда:

```
vagrant ssh jenkins
```

Результат:

```
archik@archik-1M1613:~/Documents/devops-project_3$ vagrant ssh jenkins
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-214-generic x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:       https://ubuntu.com/pro

System information as of Wed May 14 07:47:41 UTC 2025

System load:  0.18               Processes:            132
Usage of /:   6.9% of 38.7GB     Users logged in:     0
Memory usage: 29%               IPv4 address for enp0s3: 10.0.2.15
Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

18 updates can be applied immediately.
9 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

New release '22.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Fri May  9 12:10:22 2025 from 10.0.2.2
vagrant@ubuntu-focal:~$
```

## Установка и настройка Jenkins (VM\_1)

Для обеспечения непрерывной интеграции и доставки (CI/CD) на виртуальной машине VM\_1 был развернут Jenkins. Процесс установки и настройки включал следующие этапы:

### Установка JDK и зависимостей

1. Прежде чем что-либо устанавливать, необходимо обновить список доступных пакетов:

```
sudo apt update
```

2. Далее необходимо выполнить следующую команду:

```
sudo apt install fontconfig openjdk-21-jre
```

Данная команда выполняет установку двух пакетов:

- a. fontconfig — это библиотека для настройки и управления шрифтами в Linux. В данном случае она нужна чисто косметически, и не имеет реальной необходимости.
- b. openjdk-21-jre — это Java Runtime Environment (JRE) из проекта OpenJDK версии 21. Данный LTS-релиз является одним из последних в Java, и является рекомендованным для корректной работы с Jenkins.

Результат:

```
vagrant@ubuntu-focal:~$ sudo apt install fontconfig openjdk-21-jre
Reading package lists... Done
Building dependency tree
Reading state information... Done
fontconfig is already the newest version (2.13.1-2ubuntu3).
openjdk-21-jre is already the newest version (21.0.7+6-us1-0ubuntu1~20.04).
0 upgraded, 0 newly installed, 0 to remove and 15 not upgraded.
```

3. Теперь, чтобы убедиться, что всё установлено корректно, можно использовать следующую команду:

```
java --version
```

Результат:

```
vagrant@ubuntu-focal:~$ java --version
openjdk 21.0.7 2025-04-15
OpenJDK Runtime Environment (build 21.0.7+6-Ubuntu-0ubuntu120.04)
OpenJDK 64-Bit Server VM (build 21.0.7+6-Ubuntu-0ubuntu120.04, mixed mode, sharing)
```

## Установка Jenkins

После установки необходимых зависимостей, можно приступить к непосредственной установке Jenkins. Чтобы не ошибиться, можно использовать туториал по загрузке Jenkins на официальном сайте. Так как я скачиваю его на Linux, то использую соответствующий гайд:

1. Согласно гайду, вначале нужно выполнить команду:

```
sudo wget -O /etc/apt/keyrings/jenkins-keyring.asc
```



```
\
https://pkg.jenkins.io/debian-stable/jenkins.io-202
3.key
```

Данная команда выполняет ключевое действие для настройки Jenkins:

Скачиваем GPG-ключ репозитория Jenkins. Т.е. утилита `wget` загружает файл ключа по данному URL, а флаг `-O` указывает место, в которое следует сохранить ключ. Данный ключ нужен для проверки подлинности пакетов Jenkins.

2. Следующая команда в гайде:

```
echo "deb
[signed-by=/etc/apt/keyrings/jenkins-keyring.asc]"
\
  https://pkg.jenkins.io/debian-stable binary/ |
sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
```

Данная команда состоит из нескольких других, необходимых для подключения репозитория Jenkins:

Вначале формируется строка с данными репозитория:

```
deb      [signed-by=/etc/apt/keyrings/jenkins-keyring.asc]
https://pkg.jenkins.io/debian-stable binary/
```

Тут `"deb"` — указывает, что это репозитория с бинарными пакетами. `"[signed-by=...]"` — это ссылка на ранее загруженный GPG-ключ для проверки подлинности пакетов. `"https://pkg.jenkins.io/debian-stable"` — URL официального репозитория Jenkins. `"binary/"` — путь к пакетам внутри репозитория.

После идет сохранение в отдельный файл. Команда: `sudo tee /etc/apt/sources.list.d/jenkins.list`

создает файл `jenkins.list` в каталоге `/etc/apt/sources.list.d/`. А `> /dev/null` подавляет вывод в терминал.

3. Согласно гайду, далее, необходимо обновить информацию о пакетах, после чего можно наконец установить jenkins.

```
sudo apt-get update
sudo apt-get install jenkins
```

Результат:

```
vagrant@ubuntu-focal:~$ sudo apt-get install jenkins
Reading package lists... Done
Building dependency tree
Reading state information... Done
jenkins is already the newest version (2.504.1).
0 upgraded, 0 newly installed, 0 to remove and 15 not upgraded.
```

4. Теперь остается только запустить jenkins. Для этого используются команды:

```
sudo systemctl start jenkins
sudo systemctl enable jenkins
sudo systemctl status jenkins
```

Результат:

```
vagrant@ubuntu-focal:~$ sudo systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: ena
   Active: active (running) since Wed 2025-05-14 11:23:33 UTC; 56min ago
     Main PID: 638 (java)
       Tasks: 43 (limit: 2324)
      Memory: 410.6M
      CGroup: /system.slice/jenkins.service
              └─638 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkin

May 14 11:23:29 ubuntu-focal jenkins[638]: 2025-05-14 11:23:29.483+0000 [id=39]
May 14 11:23:29 ubuntu-focal jenkins[638]: 2025-05-14 11:23:29.715+0000 [id=39]
May 14 11:23:30 ubuntu-focal jenkins[638]: 2025-05-14 11:23:30.729+0000 [id=36]
May 14 11:23:33 ubuntu-focal jenkins[638]: 2025-05-14 11:23:33.122+0000 [id=38]
May 14 11:23:33 ubuntu-focal jenkins[638]: 2025-05-14 11:23:33.123+0000 [id=38]
May 14 11:23:33 ubuntu-focal jenkins[638]: 2025-05-14 11:23:33.748+0000 [id=38]
May 14 11:23:33 ubuntu-focal jenkins[638]: 2025-05-14 11:23:33.796+0000 [id=36]
May 14 11:23:33 ubuntu-focal jenkins[638]: 2025-05-14 11:23:33.918+0000 [id=36]
May 14 11:23:33 ubuntu-focal jenkins[638]: 2025-05-14 11:23:33.984+0000 [id=30]
May 14 11:23:33 ubuntu-focal systemd[1]: Started Jenkins Continuous Integration Serve
lines 1-19/19 (END)
```

## Первоначальная настройка Jenkins (плагины, пользователи)

После установки Jenkins и всех его зависимостей, нужно провести первоначальную настройку через веб-интерфейс.

Веб-интерфейс, в моем случае, доступен по адресу **192.168.56.10:8080**. На экране "приветствия", в первую очередь, необходимо ввести пароль, который можно найти на VM\_1 по адресу `/var/lib/jenkins/secrets/initialAdminPassword`.

Далее появилась возможность выбора установки плагинов Jenkins-а, я выбрал опцию "Initial suggested plugins" для установки базового набора (Git, Pipeline и др.).

Дальше появилось окно регистрации, где я успешно придумал имя и пароль, а также указал почту. После нужно было подтвердить URL сервера Jenkins (в моем случае я оставил <http://192.168.56.10:8080>).

На этом этапе у меня появилась возможность свободно пользоваться интерфейсом, и первым делом я отправился за дополнительными плагинами. В частности, для моего проекта необходим **Ansible**, для выполнения плейбуков.

## Настройка Ansible (VM\_2)

P.S. В процессе я пришел к новому решению задачи, и сделал загрузку index.html из репозитория GitHub. Но переписывать все уже написанные моменты, мне немного напряжно, поэтому напишу здесь. (это же отчет, а не курсач? 🤖)

Для автоматизации развёртывания веб-сервера на VM\_3 была выбрана платформа Ansible, работающая на VM\_2. Этот этап включал установку инструмента, настройку инвентаря, создание плейбука для управления конфигурацией и подготовку веб-страницы.

### Установка Ansible

На VM\_2, Ansible был установлен с использованием стандартных команд:

```
sudo apt update
```

```
sudo apt install ansible
```

Для уверенности можно проверить версию Ansible:

```
ansible --version
```

Результат:

```
vagrant@ubuntu-focal:~$ ansible --version
ansible 2.9.6
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/vagrant/.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 = 3.8.10 (default, Mar 18 2025, 20:04:55) [GCC 9.4.0]
```

### Создание inventory-файла (inventory.ini)

Файл **inventory.ini** нужен для определения целевых хостов, на которых Ansible выполняет задачи. В моем **inventory.ini** я указал IP-адрес VM\_3. Вот так выглядит файл:

```
[webservers]
```

```
192.168.56.12 ansible_user=vagrant
```

## Разработка плейбука (deploy.yml) для развертывания Nginx

Плейбук `deploy.yml` содержит задачи для установки и настройки Nginx на VM\_3:

1. В первую очередь идёт блок “заголовок плейбука”:

```
- name: Deploy Web Application
  hosts: webservers
  become: yes
```

Запуск происходит на группе хостов `webservers`. И выполняться все будет с правами `sudo` (`become: yes`).

2. Дальше блок “Обновления кэша пакетов”:

```
- name: Update apt cache
  apt:
    update_cache: yes
  tags: update
```

Тут обновляется информация о пакетах (некий аналог `apt update`).

3. После блок “Установки Nginx”:

```
- name: Install Nginx
  apt:
    name: nginx
    state: latest
  tags: nginx
```

Устанавливается последняя версия Nginx

4. И теперь можно перейти к блоку “Загрузки веб-страницы”:

После доработки проекта задача копирования `index.html` была изменена. Вместо локального файла Ansible стал забирать содержимое из репозитория:

```
- name: Download index.html from GitHub
  get_url:
    url:
      "https://raw.githubusercontent.com/.../index.html"
    dest: /var/www/html/index.html
    mode: '0644'
    force: yes
```

Скачивается HTML-файл из репозитория GitHub, и сохраняется в стандартную папку Nginx. Также я сделал принудительную перезапись уже существующего файла.

5. Ниже описан блок “Управление сервисов Nginx”:

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

В этом блоке прописана гарантия запуска сервиса. Также здесь включается автозапуск при загрузке.

6. И последний блок “Обработчик”:

```
handlers:
  - name: Restart Nginx
    service:
      name: nginx
      state: restarted
```

Обработчик нужен для перезагрузки Nginx.

## Создание шаблона веб-страницы (index.html)

Файл index.html был подготовлен как статическая страница для демонстрации работы веб-сервера. Его содержимое включает базовую разметку HTML с заголовком и текстом.

После доработки я просто перенес содержимое файла в репозиторий на GitHub, в соответствующий файл.

## Проверка работы Ansible

Перед интеграцией с Jenkins, хорошо бы выполнить ручную проверку плейбука. Для этого используется команда:

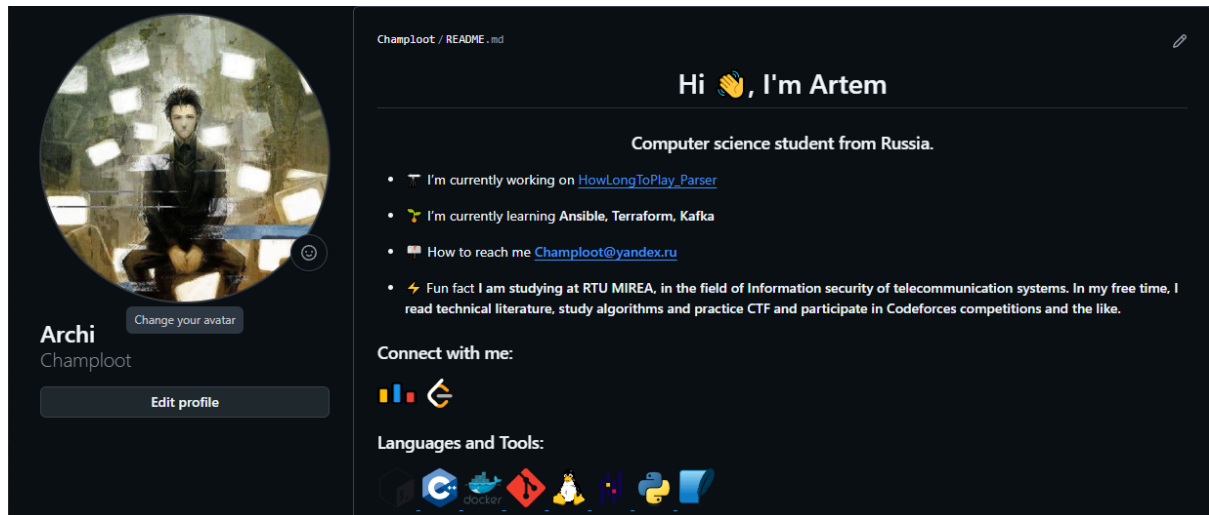
```
ansible-playbook -i inventory.ini deploy.yml
```

Результат:

```
vagrant@ubuntu-focal:~$ ansible-playbook -i ./ansible/inventory.ini ./ansible/deploy.yml
PLAY [Deploy Web Application] *****
TASK [Gathering Facts] *****
ok: [192.168.56.12]
TASK [Update apt cache] *****
changed: [192.168.56.12]
TASK [Install Nginx] *****
ok: [192.168.56.12]
TASK [Download index.html from GitHub] *****
ok: [192.168.56.12]
TASK [Ensure Nginx is running] *****
ok: [192.168.56.12]
PLAY RECAP *****
192.168.56.12 : ok=5    changed=1    unreachable=0    failed=0    skipped=0    re
scued=0    ignored=0
```

# Интеграция с GitHub

Хотя в этом проекте можно было обойтись и без GitHub, я решил что это также возможность показать навыки работы с ним. (А ещё засветить профиль)



## Создание нового репозитория

Для интеграции проекта с системой контроля версий GitHub был создан новый репозиторий, который стал центральным хранилищем для всех конфигурационных файлов и веб-контента. Репозиторий был инициализирован с базовой структурой, включая:

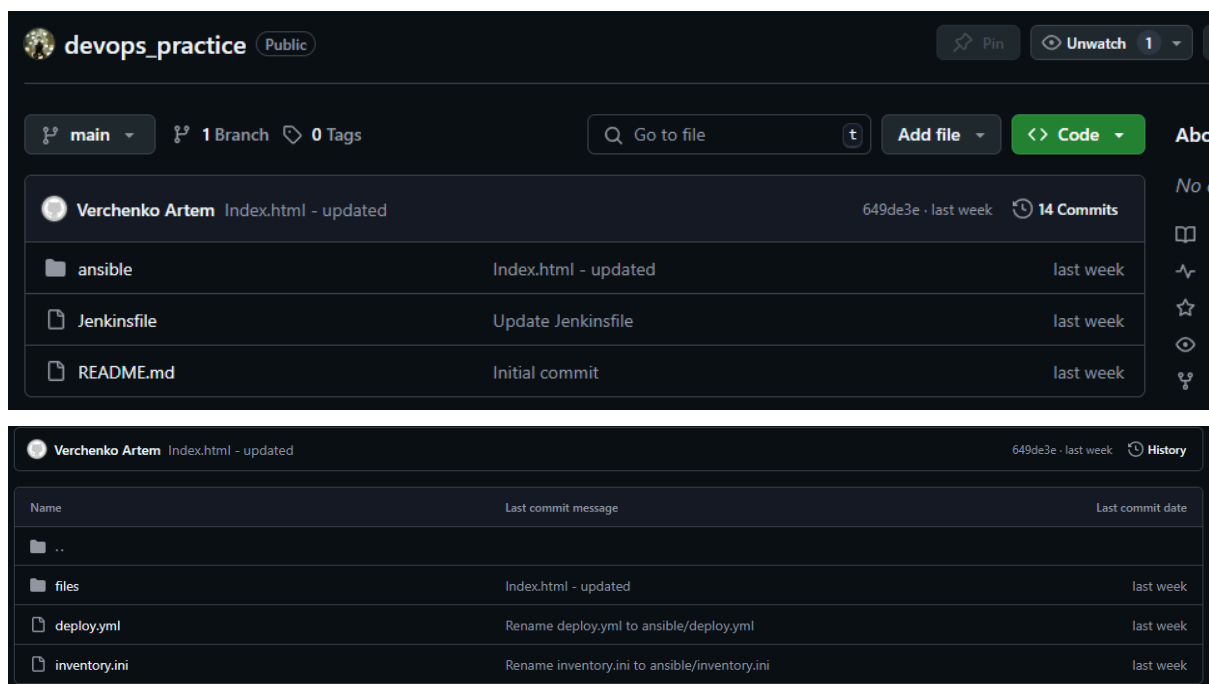
- Файл `index.html`, содержащий HTML-страницу для развертывания на веб-сервере.
- Каталог для Ansible-плейбуков (`ansible/`), где размещен файл `deploy.yml`.
- Файл `README.md` с описанием проекта и инструкциями по его запуску.

На этапе создания репозитория были настроены:

- **Ветвление:** Основная ветка `main` для стабильной версии кода.
- **Доступ:** Репозиторий я сделал публичным, чтобы избежать лишних заморочек.

После настройки все локальные файлы, включая Ansible-плейбуки и веб-контент, были загружены в репозиторий через команды `git add`, `git commit` и `git push`. Это позволило обеспечить версиюность и возможность отката изменений.





## Клонирование репозитория на VM\_2

Чтобы Ansible-контроллер (VM\_2) мог использовать актуальные файлы из GitHub, репозиторий был клонирован на эту виртуальную машину. Процесс включал следующие шаги:

1. Установка Git: На VM\_2 выполнена установка Git.
2. Клонирование репозитория:

Выполнена команда `git clone https://github.com/Champloot/devops_practice` в целевой директории на VM\_2. Это позволило получить локальную копию всех файлов, включая index.html и Ansible-плейбуки.

## Настройка CI/CD Pipeline в Jenkins

### Создание Jenkinsfile

Для автоматизации процесса сборки, развертывания и проверки в Jenkins был разработан Jenkinsfile, определяющий этапы пайплайна.

В первую очередь в файле расписан этап сборки. На этом этапе выполняются подготовительные действия: проверяется доступность целевой виртуальной машины (VM\_3) и наличие необходимых зависимостей. В рамках проекта данный этап включал:

- Подключение к VM\_2 для проверки актуальности репозитория GitHub.
- Клонирование или обновление репозитория с файлом `index.html`, чтобы убедиться, что изменения из GitHub загружены.

#### 1. Объявление пайплайна:

```
pipeline {
  agent any
```

`pipeline` — декларативное описание процесса сборки.  
`agent any` — список действий внутри этапа.

#### 2. Секция этапов:

```
stages {
  stage('Deploy') {
    steps {
```

`stage('Deploy')` — организация этапа, с названием "Deploy".  
`steps` — список действий внутри этапа.

#### 3. SSH-команда для деплоя

```
sh """
ssh -o StrictHostKeyChecking=no \
-i /var/lib/jenkins/.ssh/id_rsa \
vagrant@192.168.56.11 \
'ansible-playbook -i ansible/inventory.ini
ansible/deploy.yml'
"""
```

`-o StrictHostKeyChecking=no` — отключает проверку SSH-ключа для хоста. Это нужно для автоматизации.  
`-i` — указывает путь к приватному ключу Jenkins.  
Подключение к серверу `vagrant@192.168.56.11` (VM\_2).  
Запуск Ansible-плейбука `deploy.yml` с указанием инвентории `inventory.ini`.

## Настройка Jenkins Pipeline

Учитывая все вышесказанное, необходимо провести конечную настройку Jenkins.

Вначале необходимо создать Pipeline. Для этого на вкладке **Dashboard** выберем **New Item**. Откроется окно, в котором нужно выбрать название "Item-a" и его тип, в моем случае свой pipeline я назвал "pipeline\_".

Далее открывается окно настройки моего "pipeline\_". Там нужно настроить поле Definition:

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://github.com/Champloot/devops\_practice.git

Credentials ?

- none -

+ Add

Advanced

Тут я указал URL репозитория, из которого нужно будет брать Jenkinsfile.

Branches to build ?

Branch Specifier (blank for 'any') ?

main

Тут указал ветку разработки. В моем случае она вообще одна, но в реальном проекте, их, что логично, может быть больше.

Script Path ?

Jenkinsfile

☒ Lightweight checkout ?

А здесь указал путь в моем репозитории до Jenkinsfile. В моем случае файл находится в корне проекта, поэтому просто "Jenkinsfile".

После этого открывается страница Status моего "pipeline\_":

[Dashboard](#) > [pipeline\\_](#) >

Status

</> Changes

▶ Build Now

⚙️ Configure

🗑️ Delete Pipeline

📁 Stages

✎ Rename

🔍 Pipeline Syntax

✔️ pipeline\_

Add description

Permalinks

- [Last build \(#10\), 6 days 23 hr ago](#)
- [Last stable build \(#10\), 6 days 23 hr ago](#)
- [Last successful build \(#10\), 6 days 23 hr ago](#)
- [Last failed build \(#5\), 7 days 15 hr ago](#)
- [Last unsuccessful build \(#5\), 7 days 15 hr ago](#)
- [Last completed build \(#10\), 6 days 23 hr ago](#)

Builds

Filter

May 9, 2025

✔️ #10 12:06 PM

✔️ #9 11:52 AM

May 8, 2025

✔️ #8 8:47 PM

✔️ #7 8:44 PM

✔️ #6 8:35 PM

❌ #5 8:28 PM

❌ #4 8:19 PM

❌ #3 8:17 PM


❌ #2 8:16 PM



Тут сразу можно увидеть что я уже пытался выполнить pipeline. Сначала не очень удачно, потом без проблем. Всё это отображается соответствующими знаками (галочка и крестик). Тут также можно увидеть общую сводку о моем "pipeline\_": Последняя сборка, последняя успешная сборка и другое.


Также часто приходилось дебажить до этого неправильно настроенный pipeline, для этого нужно нажать на вкладку

Configure. Ещё тут можно удалить пайплайн, а самое главное тут можно его запустить.

Нажав на **Build Now** и перейдя на, в моем случае, 11-ый запуск, увидим:


 **#11 (May 16, 2025, 12:06:57 PM)**


Progress:  


 Add description

Keep this build forever


Started 19 sec ago  
Build has been executing for 19 sec

 Started by user [Artem Verchenko](#)

 This run spent 84 ms waiting in the queue.

 **git** **Revision:** 2511c3c4a67611311eff739a2edc9ea0aeeed9e89  
**Repository:** [https://github.com/Champloot/devops\\_practice.git](https://github.com/Champloot/devops_practice.git)

- origin/main

 **Changes**

1. Update deploy.yml ([details](#) / [githubweb](#))

Тут указаны номер билда, и дата его начала. Также тут видно прогресс и пользователя, запустившего проект (это я).

Открыв **Console Output** можно увидеть более подробный вывод. В частности он может показать успешные этапы сборки, а также, если была ошибка, может указать и ее.

Ещё раз покажу **Dashboard**:

+ New Item

📋 Build History

⚙️ Manage Jenkins

📅 My Views

✎ Add description

Build Queue ▼

No builds in the queue.

Build Executor Status 0/2 ▼

All

+

S	W	Name ↓	Last Success	Last Failure	Last Duration	
✓	☀	pipeline_	6 min 13 sec #11	7 days 15 hr #5	44 sec	▶

Icon: S M L

...

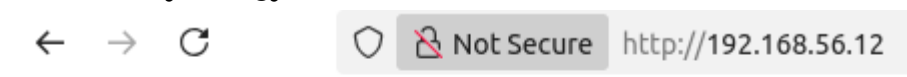
# Тестирование

Чтобы проверить работоспособность, я запущу все виртуальной машины. После этого открою в браузере VM\_3, где ожидаю увидеть содержимое `index.html`.

После этого, я поменяю `index.html` на основной машине (или прямо на GitHub), а после запущу в репозиторий на GitHub. Потом, я запущу Pipeline, и надеюсь увидеть в браузере содержимое нового `index.html`.

## Процесс тестирования

После очередного “`vagrant up`” и открытия в браузере адреса VM\_3 вижу следующее:



**Some text.**


Как несложно заметить видно “какой-то текст”. Теперь обновим файл и запустим пайплайн:

```

archik@archik-TM1613:~/Documents/devops-project_3$ git clone https://github.com/Champloot/devops_practice/blob/main/ansible/files/index.html
Cloning into 'index.html'...
fatal: repository 'https://github.com/Champloot/devops_practice/blob/main/ansible/files/index.html/' not found
archik@archik-TM1613:~/Documents/devops-project_3$ git clone https://github.com/Champloot/devops_practice.git
Cloning into 'devops_practice'...
remote: Enumerating objects: 54, done.
remote: Counting objects: 100% (54/54), done.
remote: Compressing objects: 100% (44/44), done.
remote: Total 54 (delta 13), reused 6 (delta 2), pack-reused 0 (from 0)
Receiving objects: 100% (54/54), 15.99 KiB | 1.78 MiB/s, done.
Resolving deltas: 100% (13/13), done.
archik@archik-TM1613:~/Documents/devops-project_3$ cd devops_practice/
archik@archik-TM1613:~/Documents/devops-project_3/devops_practice$ nano ansible/files/index.html
archik@archik-TM1613:~/Documents/devops-project_3/devops_practice$ cat ansible/files/index.html
<!DOCTYPE html>
<html>
<head>
  <title>DevOps Demo</title>
</head>
<body>
  <h1>Some text. It was updated!</h1>
</body>
</html>
archik@archik-TM1613:~/Documents/devops-project_3/devops_practice$ git add .
archik@archik-TM1613:~/Documents/devops-project_3/devops_practice$ git commit -m "update"
[main 283ab35] update
1 file changed, 1 insertion(+), 1 deletion(-)
archik@archik-TM1613:~/Documents/devops-project_3/devops_practice$ git push
Username for 'https://github.com': Champloot
Password for 'https://Champloot@github.com':
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 465 bytes | 232.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/Champloot/devops_practice.git
   f86e212..283ab35  main -> main
archik@archik-TM1613:~/Documents/devops-project_3/devops_practice$


```

Теперь запустим Pipeline:

 **#13 (May 16, 2025, 12:39:33 PM)**

Progress:



 Add description

Keep this build forever

Увидим результат:



 Not Secure http://192.168.56.12

**Some text. It was updated!**



# Проблемы

Основная проблема заключается в локальности моего проекта. В идеале было бы использовать облачные сервисы вроде AWS, но так как это либо дорого, либо попросту не доступно у нас в стране, я был вынужден мириться с таким решением.

Также у меня не получилось по-человечески настроить веб хуки между GitHub и Jenkins. То есть они друг о друге знают, но как сделать авто-деплой — я не знаю.

# Заключение

## Итоги реализации проекта

В ходе реализации проекта была успешно развернута инфраструктура из трёх виртуальных машин, каждая из которых выполняет определенную роль в CI/CD-процессе. С помощью Vagrant обеспечена быстрая и воспроизводимая настройка окружения, что позволило избежать проблем с совместимостью на разных этапах разработки. На VM\_1 установлен и настроен Jenkins, который стал центральным элементом автоматизации: через него был реализован Pipeline, запускающий задачи развертывания. На VM\_2 развернут Ansible, что дало возможность управлять конфигурацией веб-сервера (VM\_3) декларативно, используя плейбуки. Ключевым достижением стала интеграция с GitHub: Ansible теперь загружает актуальную версию [index.html](#) напрямую из репозитория, что обеспечивает мгновенное отображение изменений на веб-странице после обновления кода. В результате система демонстрирует полный цикл CI/CD — от внесения правок в репозиторий до их автоматического деплоя на продакшн-сервер.

## Преимущества автоматизации

Автоматизация процессов в проекте позволила существенно сократить время рутинных операций и минимизировать человеческий фактор. Например:

- Ansible устранил необходимость вручную настраивать Nginx на VM\_3 — все действия выполняются через предопределенные плейбуки, что гарантирует идентичность конфигураций при каждом развертывании.
  - Jenkins обеспечил единую точку управления Pipeline: задачи сборки, тестирования (при дальнейшем расширении) и деплоя выполняются последовательно и без вмешательства пользователя.
  - Интеграция с GitHub позволила синхронизировать исходный код с инфраструктурой: изменения в репозитории автоматически запускают Pipeline, что ускоряет feedback loop и упрощает внедрение новых версий продукта.
- Кроме того, использование Vagrant для создания идентичных виртуальных сред исключило проблему «работает на моей

машине», сделав процесс разработки предсказуемым и масштабируемым.

# Приложение

Jenkinsfile:

```
pipeline {
  agent any
  stages {
    stage('Deploy') {
      steps {
        sh """
            ssh -o StrictHostKeyChecking=no \
              -i /var/lib/jenkins/.ssh/id_rsa \
              vagrant@192.168.56.11 \
              'ansible-playbook -i ansible/inventory.ini
ansible/deploy.yml'
            """
      }
    }
  }
}
```

Deploy.yml:

```
- name: Deploy Web Application
  hosts: webservers
  become: yes

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

    - name: Install Nginx
      apt:
        name: nginx
        state: latest
      tags: nginx

    - name: Download index.html from GitHub
      get_url:
        url:
          "https://raw.githubusercontent.com/Champloot/devops_practice/
main/ansible/files/index.html"
```

```

    dest: /var/www/html/index.html
    mode: '0644'
    force: yes
    become: yes

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

handlers:
- name: Restart Nginx
  service:
    name: nginx
    state: restarted

```

Inventory.ini:

```

[webservers]
192.168.56.12 ansible_user=vagrant

```

Vagrantfile:

```

Vagrant.configure("2") do |config|

  config.vm.define "jenkins" do |jenkins|
    jenkins.vm.box = "ubuntu/focal64"
    jenkins.vm.network "private_network", ip: "192.168.56.10"
    jenkins.vm.provider "virtualbox" do |vb|
      vb.memory = "2048"
    end
  end

  config.vm.define "ansible" do |ansible|
    ansible.vm.box = "ubuntu/focal64"
    ansible.vm.network "private_network", ip: "192.168.56.11"
    ansible.vm.provider "virtualbox" do |vb|
      vb.memory = "2048"
    end
  end
end

```

```
config.vm.define "web" do |web|
  web.vm.box = "ubuntu/focal64"
  web.vm.network "private_network", ip: "192.168.56.12"
  web.vm.provider "virtualbox" do |vb|
    vb.memory = "2048"
  end
end
end
```

## Список литературы

1. Ким, Д., Хамбл, Д., Дебуа, П., Уиллис, Д. «Руководство по DevOps: Как добиться гибкости, надежности и безопасности мирового уровня в ИТ-системах». — М.: Альпина Пабlishер, 2017.
2. Моррис, К. «Инфраструктура как код: Управление серверами в облаке». — СПб.: Питер, 2018.
3. Хохштейн, Л. «Ansible: Практическое руководство». — М.: ДМК Пресс, 2019.
4. Vagrant — Официальная документация. URL: <https://www.vagrantup.com/docs>
5. Ansible — Официальная документация. URL: <https://docs.ansible.com>
6. Jenkins — Официальная документация. URL: <https://www.jenkins.io/doc/>
7. GitHub — Официальная документация. URL: <https://docs.github.com>
8. Nginx — Официальная документация. URL: <https://nginx.org/en/docs/>
9. VirtualBox — Руководство пользователя. URL: <https://www.virtualbox.org/manual/>