

Университет ИТМО

**Облачные системы и услуги**

**Лабораторная работа №4\***

Выполнили: Данилов Владислав Андреевич K34201**,**

Ледванов Вадим Дмитриевич K34201,

Смирнов Кирилл Андреевич K34201

**Преподаватель:** Мигулаева Татьяна Алексеевна

Санкт-Петербург

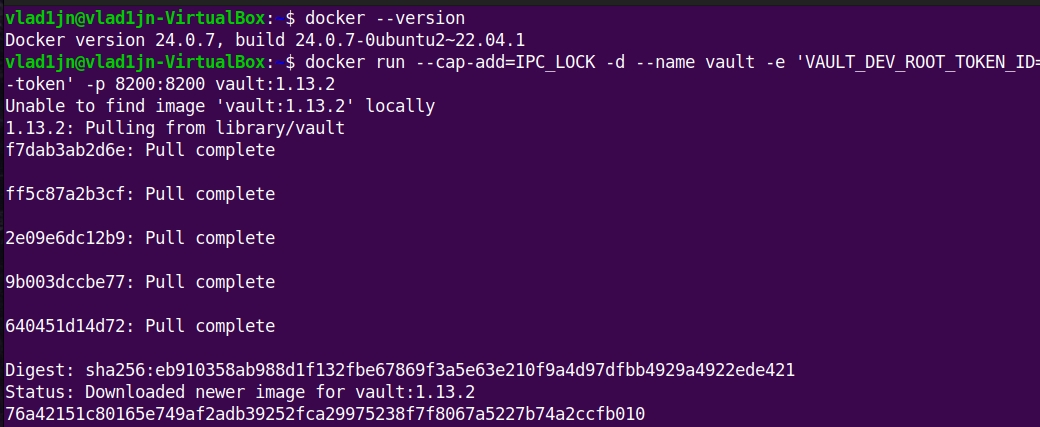
2024

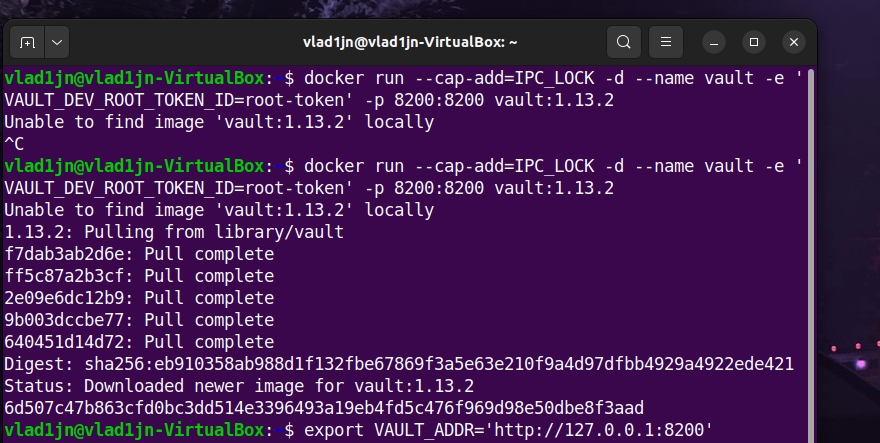
**Цель работы:**

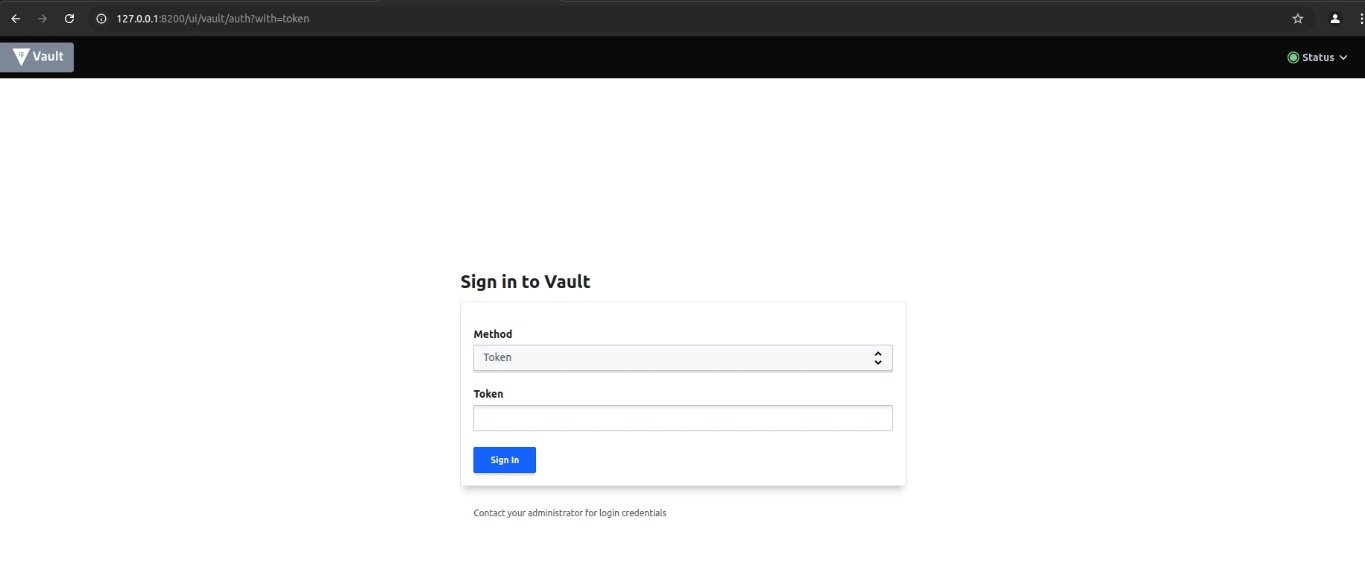
В работе, рассмотрим установку HashiCorp Vault, настройку аутентификации, управление доступом к секретам и использование этих секретов в Jenkins пайплайне, не отображая их в логах.

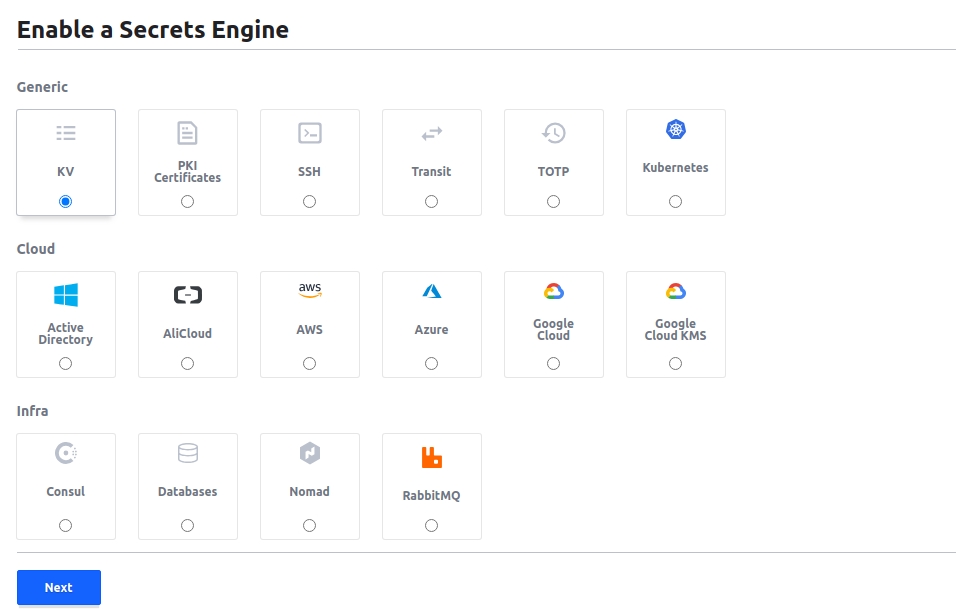
**Ход работы:**

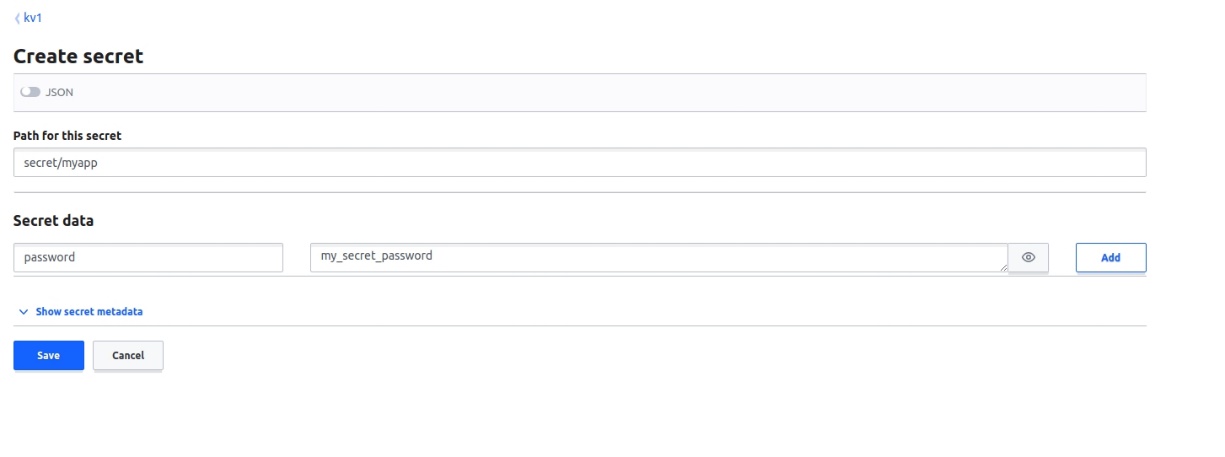
**Задание 1**.

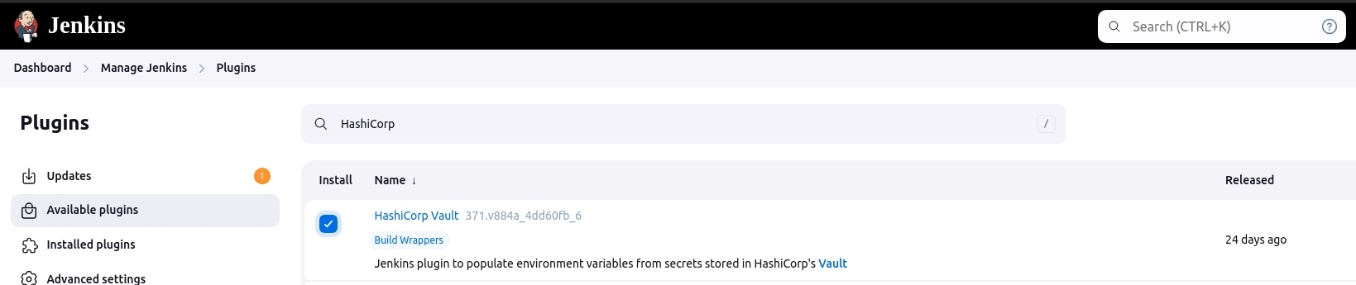
Поднимаем HashiCorp Vault в в контейнере Docker:

Далее, присваиваем Http-адрес для контейнера и открываем Vault в Chrome и авторизуемся по токену:

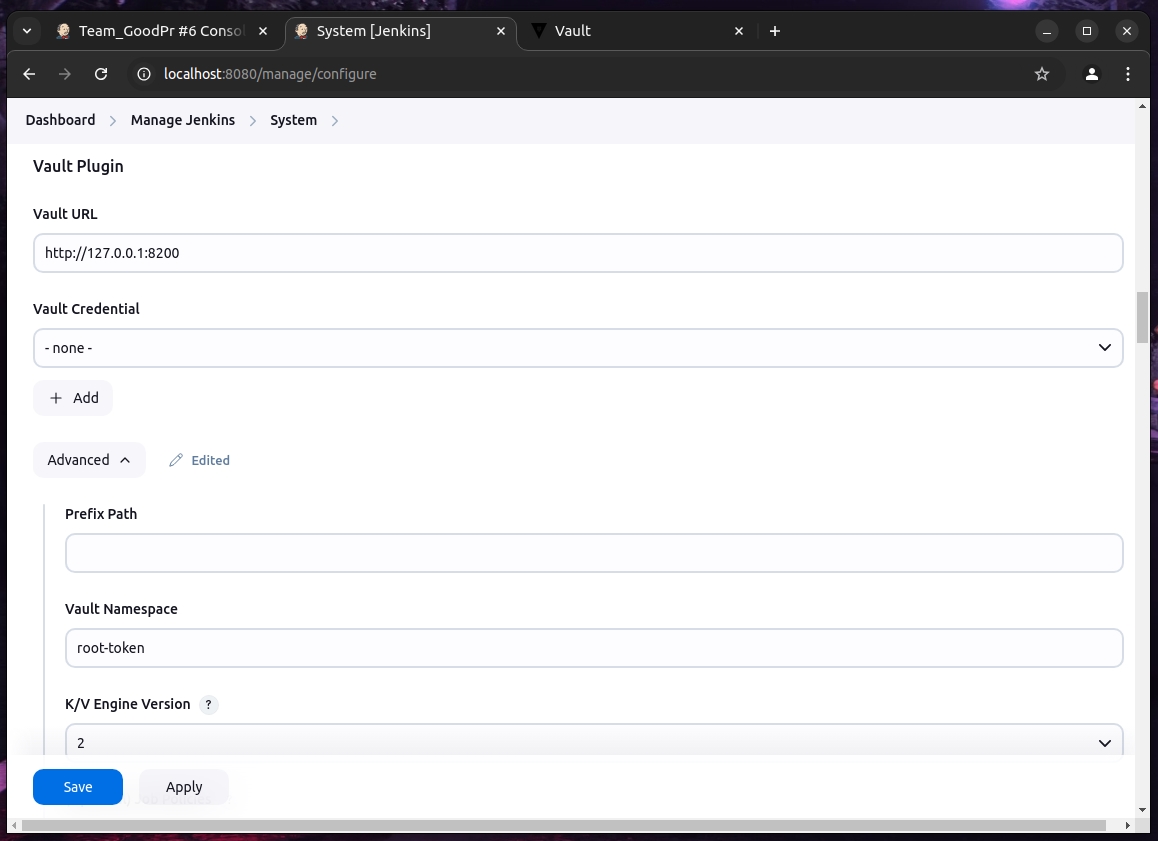
На странице ниже выбираем метод TOKEN и вводим ‘root-token’:

Далее, инициализируем Secrets Engine. Выбираем KV и вводим название:

Необходимо ввести путь к секрету и его данные, в формате ключ- значение:

Чтобы Jenkins мог подключаться к Vault и запрашивать секреты, нужно установить плагин HashiCorp Vault для Jenkins. Из менеджера Jenkins устанавливаем ‘HashiCorp Vault Plugin’:

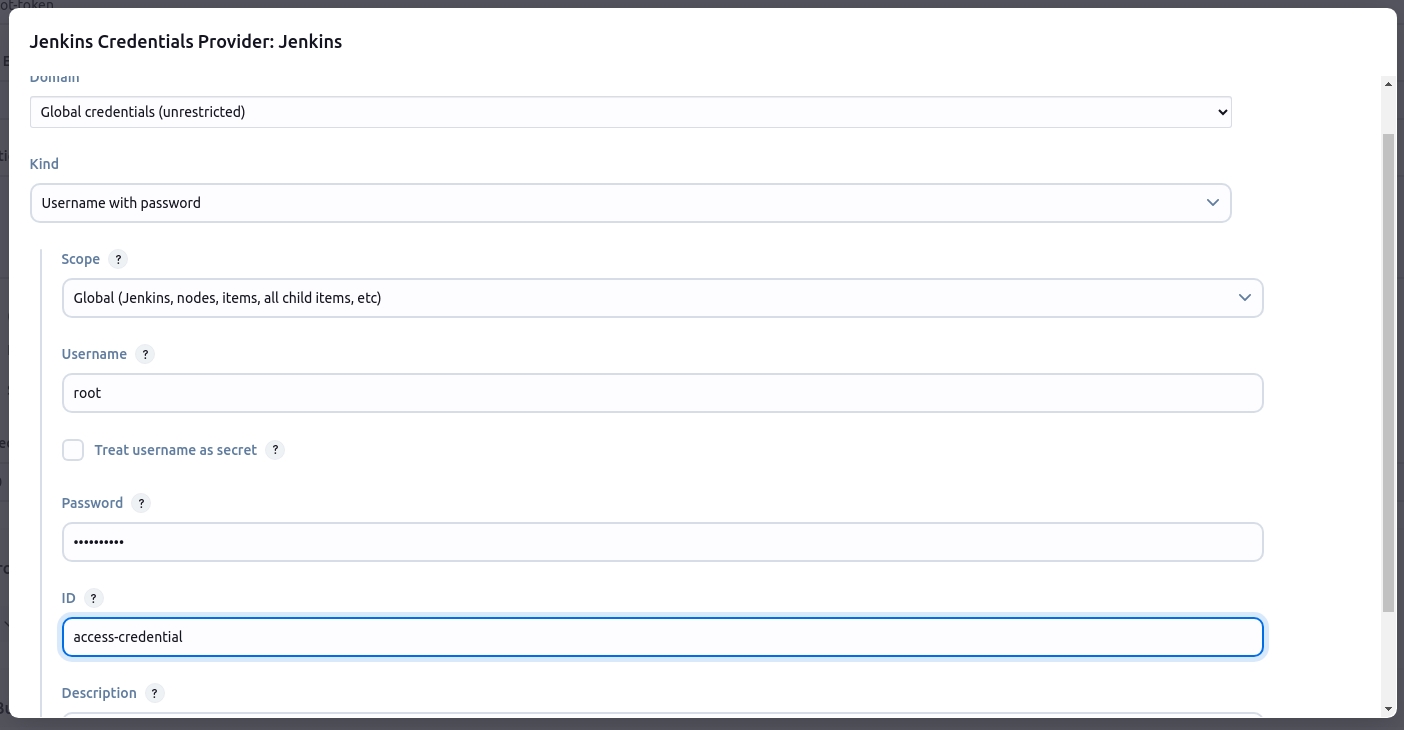
После установки плагина для обеспечения их взаимодействия открываем ‘Configure System’ в менаджере.

Необходимо добавить новую конфигурацию Jenkins для Vault. В ней указываются значения URL, Vault Token, Engine Version. Они были настроены раньше. Первичная настройка взаимодействия представлена ниже:

Чтобы Jenkins мог инициализировать обращение к хранилищу секретов, производим заполнение полей в параметре Credentials (его значение ‘-none-’) на рисунке выше.

Указаны параметры, соответсвующие текущей настройке и сервисам на каждом Localhost:

* Username: ‘root’ (оно уже создано в в HashiCorp Vault)
* в поле Password: ‘root-token’ - токен доступа, созданный для Jenkins.
* ID: явно указываем уникальную строку – “ vault-access-credentials ”

Описанные параметры применяются к Jenkins на скриншоте ниже:

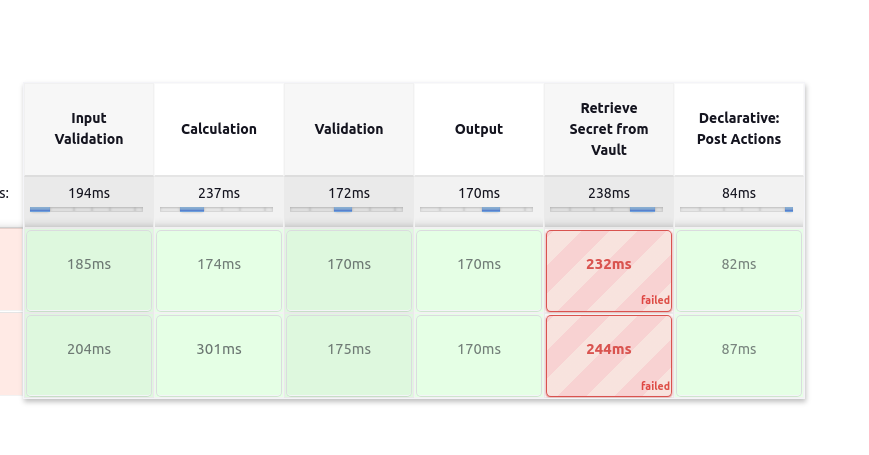
После заполнения всех полей нажмите OK для сохранения учетных данных.

Переходим к практическим запускам пайплайна из более ранней работы. В последний Stage был добавлен блок ‘withVault(’. В нем добавлена опция “credentialsId: ' access-credential’ ”, где ‘access-credential’ — это ID учетных данных, которые были создали для Vault.

Полный код пайплайна:



Jenkins, по задумке и документации, должен получить секрет из Vault, не выводя его в логах.

На практике получаем ошибку в этом Stage:

Происходит это вследствие особенностей взаимодействия HashiCorp Vault и Jenkins в контейнеризированной среде Docker:

1. Jenkins, работающий в Linux, не может получить к нему прямой доступ. Если сеть Docker Vault настроена неправильно, Jenkins в Linux не сможет отправлять запросы в Vault, используя локальный хост или обычные порты.
2. Jenkins не может отправлять запросы к Vault через localhost или нестандартные замапленные порты.
3. Даже если Jenkins сможет подключиться к контейнеру Vault, существует проблема с защитой и сохранением конфиденциальных данных. Это может привести к тому, что личная информация будет сохранена в журналах или местах, доступных из среды пайплайна Jenkins.
4. Эта конфигурация не подходит для сред с использованием секретов, поскольку в ней не используются токены или другие методы аутентификации, необходимые подключаемому модулю Jenkins Vault. Тогда он не будет иметь адекватной безопасности и защиты секрета

В заключении я приведу несколько доводов, Почему это решение не починить:

* пайплайн Jenkins не сможет подключиться к Vault должным образом: HashiCorp Vault работает в контейнере Docker на доп. Localhost, а Jenkins — как приложение на Linux.
* Vault и Jenkins в данной системе не имеют защищенного соединения и безопасного метода аутентификации

**Задание 2**.

(Копия из Readme)

Причины, по которой этот способ (HashiCorp Vault работает в контейнере Docker на доп. Localhost, а Jenkins — как приложение на Linux.) – красивый:

1. Контроль доступа к секретам со стороны HashiCorp Vault
2. Когда мы запускаете Vault в контейнере Docker, это создаёт дополнительную защиту. Vault работает сам по себе и управляется Docker-сетью. Это значит, что другие программы не могут легко получить доступ к Vault.
3. Docker позволяет применять настройки сети по средствам маппинга портов, ограничивая доступ к Vault.
4. Vault может создавать временные токены или учётные данные, которые истекают со временем или отключаются: Это снижает риск несанкционированного доступа

В заключении,

Ниже приведены причины, по которым хранение секретов в CI/CD-переменных репозитория — плохая практика:

Хранение секретов (паролей, токенов) в переменных CI/CD-пайплайна связано с рисками. Они могут попасть в логи, где их увидят все, у кого есть доступ к системе CI/CD. Кроме того, изменить такие секреты вручную сложно, а встроенных инструментов для контроля доступа, ротации и мониторинга нет, что усложняет управление безопасностью. Вследствие этого нужно усложнять систему дополнительными надстройками.

**Вывод:**

Таким образом, Vault в Docker позволяет создать надёжную систему управления секретами, в то время как хранение их в CI/CD-переменных — менее безопасная практика, которая может привести к утечке конфиденциальных данных и усложнить управление доступом.