ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
**«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»**
Институт компьютерных наук и технологий
Высшая школа программной инженерии

# Отчет о прохождении учебной ознакомительной практики

Савчук Алексей Анатольевич

2 курс, 3530904/00004

09.03.04 «Программная инженерия»

**Место прохождения практики:**

ООО «Санкт-Петербургский центр разработок ЕМС», дистанционный формат

**Сроки практики:** 20.06.22 – 15.07.22

**Руководитель практической подготовки от ФГАОУ ВО «СПбПУ»:**

Шемякин Илья Александрович, старший преподаватель ВШПИ, ИКНТ

**Руководитель практической подготовки от профильной организации:**

Саламатов Михаил Александрович, ведущий программный менеджер по маркетингу

**Оценка:** зачтено

Руководитель практической подготовки
от ФГАОУ ВО «СПбПУ»: /Шемякин И.А./

Руководитель практической подготовки
от профильной организации: /Саламатов М.А./

Обучающийся: /Савчук А.А./

Дата: 15.07.22

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
**«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»**
Институт компьютерных наук и технологий
Высшая школа программной инженерии

# ИНДИВИДУАЛЬНЫЙ ПЛАН (ЗАДАНИЕ И ГРАФИК) ПРОВЕДЕНИЯ ПРАКТИКИ

Савчук Алексей Анатольевич

Направление подготовки (код/наименование):09.03.04 «Программная инженерия»
Профиль (код/наименование):
09.03.04_01 «Технология разработки и сопровождения качественного программного продукта»
Вид практики учебная
Тип практики ознакомительная
Место прохождения практики:
ООО «Санкт-Петербургский центр разработок ЕМС», дистанционный формат

Руководитель практической подготовки от ФГАОУ ВО «СПбПУ»:

Шемякин Илья Александрович, старший преподаватель ВШПИ, ИКНТ
*(Ф.И.О., уч.степень, должность)*

Руководитель практической подготовки от профильной организации:

Саламатов Михаил Александрович, ведущий программный менеджер по маркетингу
*(Ф.И.О., должность)*

**Рабочий график проведения практики**

Сроки практики: с 20.06.22 по 15.07.22

| № п/п | Этапы практики | Вид работ | Сроки прохождения этапа практики |
|---|---|---|---|
| 1 | Организационный этап | Выбор задачи для работы на практике. Встреча для разъяснения целей, задач, содержания и порядка прохождения практики, инструктаж по технике безопасности, выдача сопроводительных документов по практике | 20.06 |
| 2 | Основной этап | Работа над задачей. Дополнительные встречи для демонстрации промежуточных результатов и решения возникших проблем. | 21.06 – 14.07 |
| 3 | Заключительный этап | Представление результатов работы над задачей, написание отчета | 15.07 |

Обучающийся: /Савчук А.А./

Руководитель практической подготовки
от ФГАОУ ВО «СПбПУ»: /Шемякин И.А./

**Согласовано**
Руководитель практической подготовки
от профильной организации: /Саламатов М.А./

# Разработка прототипа для демонстрации работы средств CI/CD в Kind окружении

# Содержание

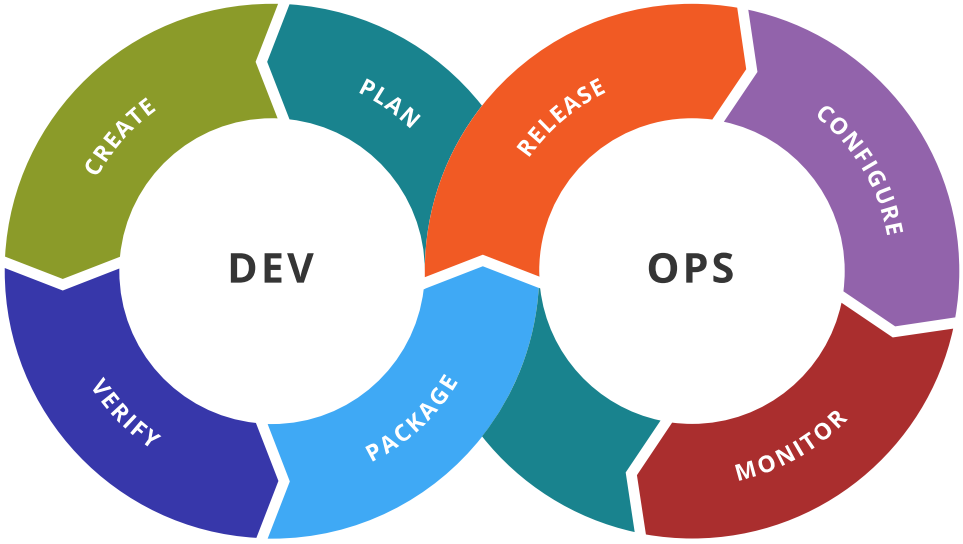
# 1 Вступительная часть

## 1.1 Формулировка проблемы

Большая часть процессов разработки программного продукта может быть автоматизирована. Существует большое число методологий, набор принципов и практик, следование которым делает процесс разработки, тестирования, развертывания приложений более простым и безопасным.

Одна из таких практик – CI/CD. Это комбинация двух принципов – Continious Integration (CI) и Continious Delivery (CD). *Непрерывная интеграция* (CI) – автоматическое включение в готовый проект написанного кода через его сборку, упаковку и тестирование. *Непрерывная поставка* (CD) – автоматическое развертывание приложения в целевом окружении.

Цель выполнения этого проекта – знакомство с некоторыми средствами CI/CD в процессе разработки небольшого веб-приложения.



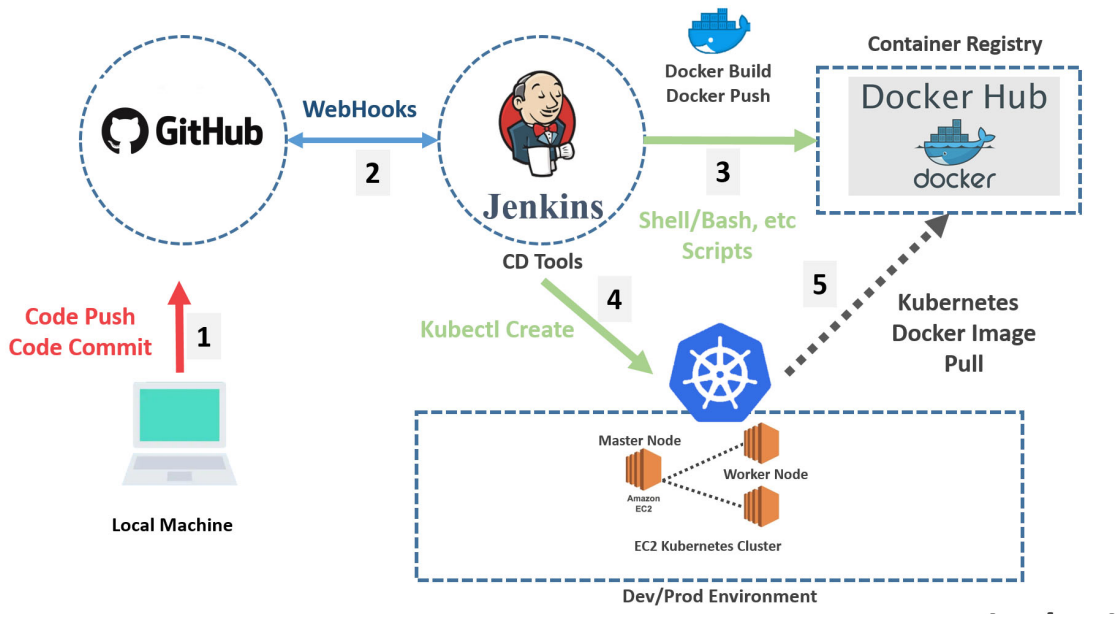

## 1.2 Постановка задачи

Разработать прототип для демонстрации работы средств CI/CD в Kind окружении. Дополнительно: приложение должно использовать интерфейс Baremetal CSI.

# 2 Основная часть

## 2.1 Подходы к решению

Приведем пример небольшого CI/CD пайплайна, который можно считать достаточно простым.




Опишем этапы:

1. **Code Commit / Code Push**. Разработчик завершает работу над новой частью приложения и добавляет изменения в репозиторий (здесь – Github) с кодом проекта.

2. **Receive GitHub Webhook**. CI/CD сервер (здесь – Jenkins) получает оповещение о том, что нужно запустить процесс сборки, после чего должен вернуть статус пайплайна. Есть несколько стратегий отслеживания изменений в репозитории проекта, например, *Polling SCM* или *Webhook Events*.

3. **Build Image / Push Image**. На сервере происходит сборка, тестирование кода и его развертывание в тестовом или рабочем окружении. Здесь в роли окружения выступает Kubernetes кластер, состоящий из нескольких машин.

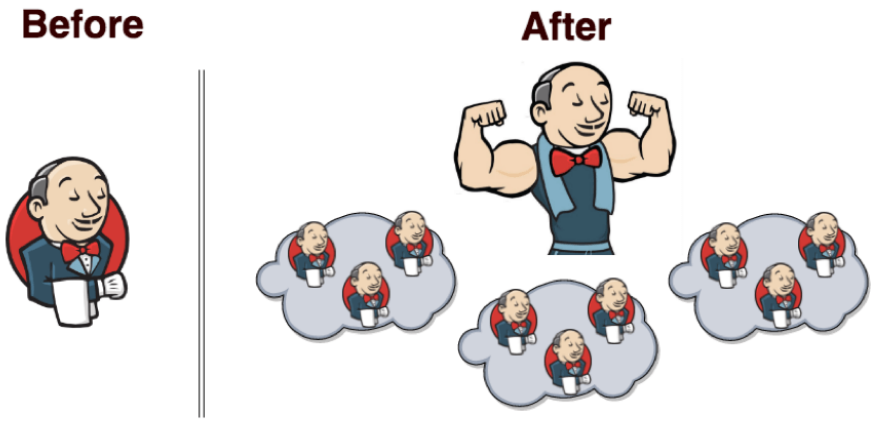
4. **Call the Kubernetes API**. На одном из этапов пайплайна Jenkins вызывает Kubernetes API и просит развернуть в кластере приложение, образ которого был только что собран и загружен в репозиторий (публичный или приватный).

5. **Pull Image / Deploy on Cluster**. Kubernetes создает указанные ресурсы и скачивает образ приложения из репозитория. Здесь можно провести некоторую работу, чтобы убедиться, что приложение было успешно развернуто и работает корректно, после чего вернуть статус и артефакты пайплайна.

## 2.2 Архитектура

Архитектура разработанного прототипа похожа на описанную ранее, но есть несколько изменений. В качестве способа отслеживания изменений в репозитории проекта используется не Webhooks Events, а Polling SCM. Сервер проверяет репозиторий в соответствии с заданным расписанием. Если в код проекта были внесены изменения, то Jenkins запускает процесс сборки. При этом между внесением изменений и началом сборки может пройти некоторое время, зависящее от расписания. В этом состоит принципиальное отличие Polling SCM от Webhooks Events, когда само внесение изменений порождает процесс сборки.

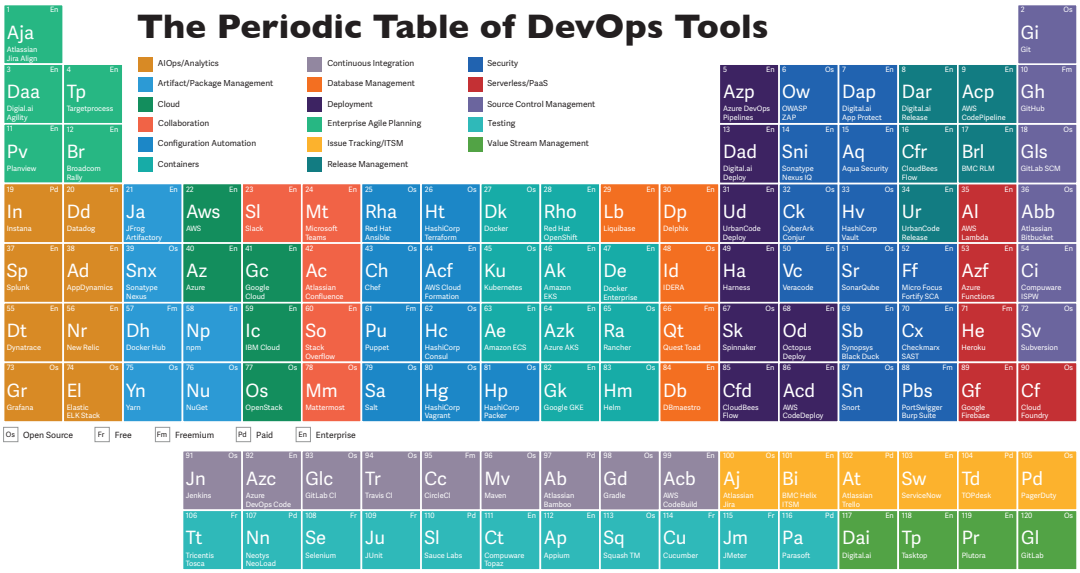
В схеме, приведенное ранее, не указано, в каком окружении работает Jenkins сервер. Есть несколько способов запуска Jenkins сервера: 1) прямо «на железе», 2) в Docker контейнере, 3) в Kubernetes кластере.




Для разработки прототипа был выбран последний вариант – запуск Jenkins внутри Kubernetes кластера. Основным преимуществом такого решения являются возможность запуска большого числа линий сборок параллельно и удобство взаимодействия с кластером.

## 2.3 Использованные инструменты

Существует большое число инструментов, которые используют DevOps-инженеры в своей работе.




Помимо DevOps-инструментов, был использован ряд средств для веб-разработки. Полный список всех использованных инструментов:

- Языки программирования и программные средства:
  - HTML, CSS, JavaScript
  - Node.js
  - Docker
- Средства взаимодействия с Kubernetes кластером:
  - Инструмент для запуска локальных Kubernetes кластеров в Docker окружении – Kind
  - Инструмент командной строки для взаимодействия с кластером – Kubectl
  - Пакетный менеджер для Kubernetes – Helm

- Официальная клиентская библиотека Python для Kubernetes

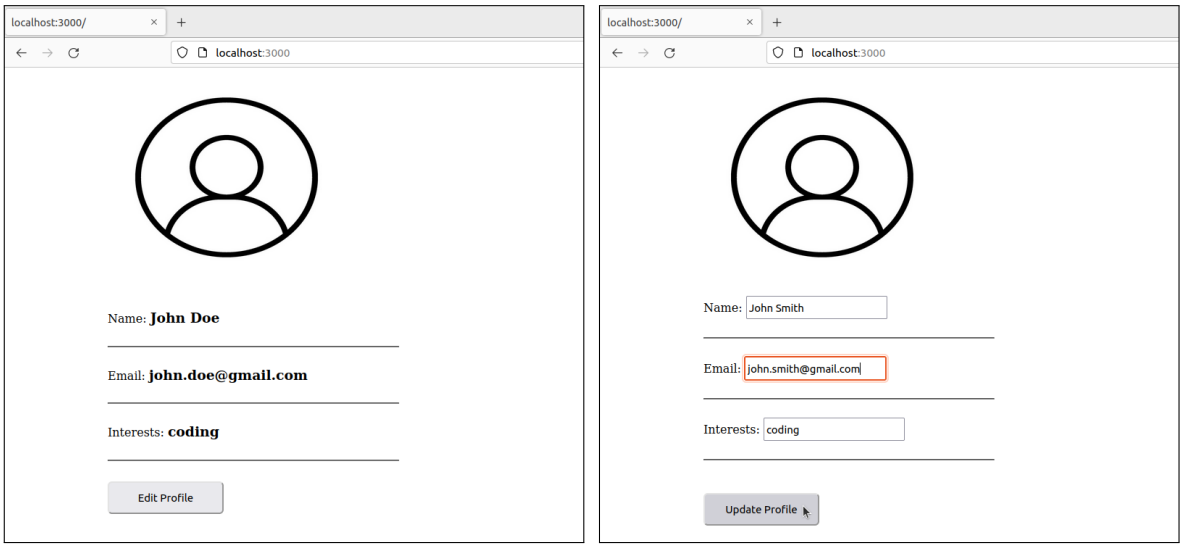
- Система Jenkins и использованные плагины:
  - Установке Jenkins внутри Kubernetes кластера – Jenkins on Kubernetes
  - Плагин для интеграции Kubernetes и Jenkins – Kubernetes plugin for Jenkins

## 2.4 Реализация

Разработанный прототип состоит из простого веб-приложения, взаимодействующего с базой данных, и пайплайна, исполняемого с помощью Jenkins, инструмента CI/CD.

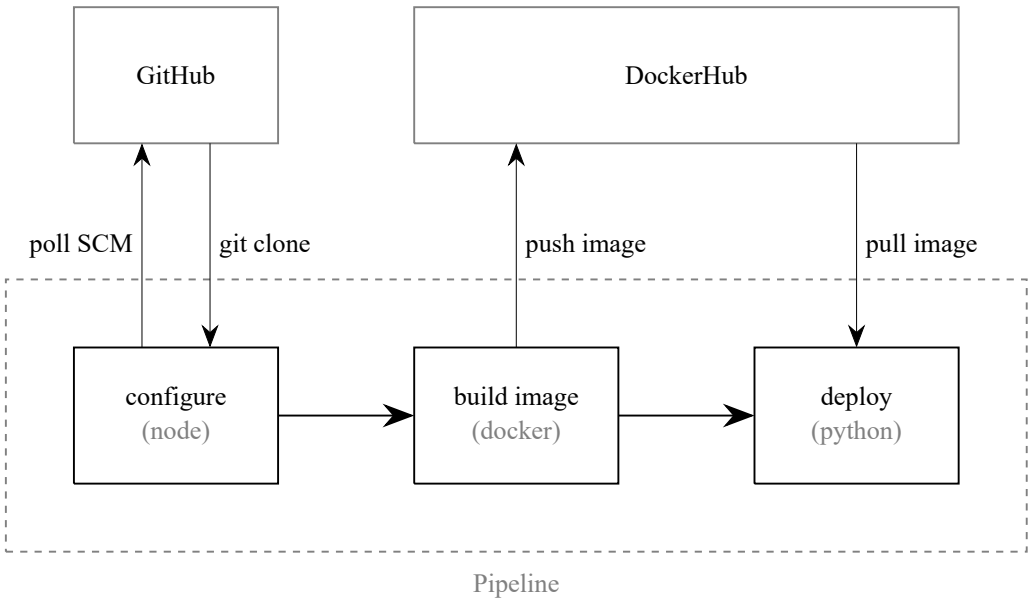
### Веб-приложение

Было написано простое веб-приложение с помощью следующих инструментов: HTML, CSS, JavaScript, Node.js. Программа взаимодействует с NoSQL базой данных MongoDB. Приложение распространяется в виде Docker образа. В нашем случае оно будет запущено внутри Kubernetes кластера (локальный кластер, созданный с помощью инструмента Kind).




### Пайплайн

Среди нескольких способов запуска Jenkins – прямо на компьютере, в контейнере и в Kubernetes кластере – был выбран последний. Установка Jenkins внутри Kubernetes кластера была осуществлена в соответствии с официальным руководством. Для интеграции Jenkins и Kubernetes был установлен специальный плагин Kubernetes Plugin for Jenkins. Для написания сценариев, взаимодействующих с кластером, была использована официальная клиентская библиотека Python для Kubernetes.




Написанный пайплайн, включает в себя следующие этапы:

1. **Конфигурация проекта (configure)**.

   На этом этапе с помощью инструмента npm (Node Package Manager) скачиваются все необходимые зависимости для веб-приложения.

2. **Сборка Docker образа (build image)**.

   На этом этапе происходит сборка Docker образа веб-приложения в соответствии с файлом Dockerfile, который находится в папке с проектом, затем образ загружается в DockerHub репозиторий.

3. **Развертывание приложения в кластере (deploy)**.

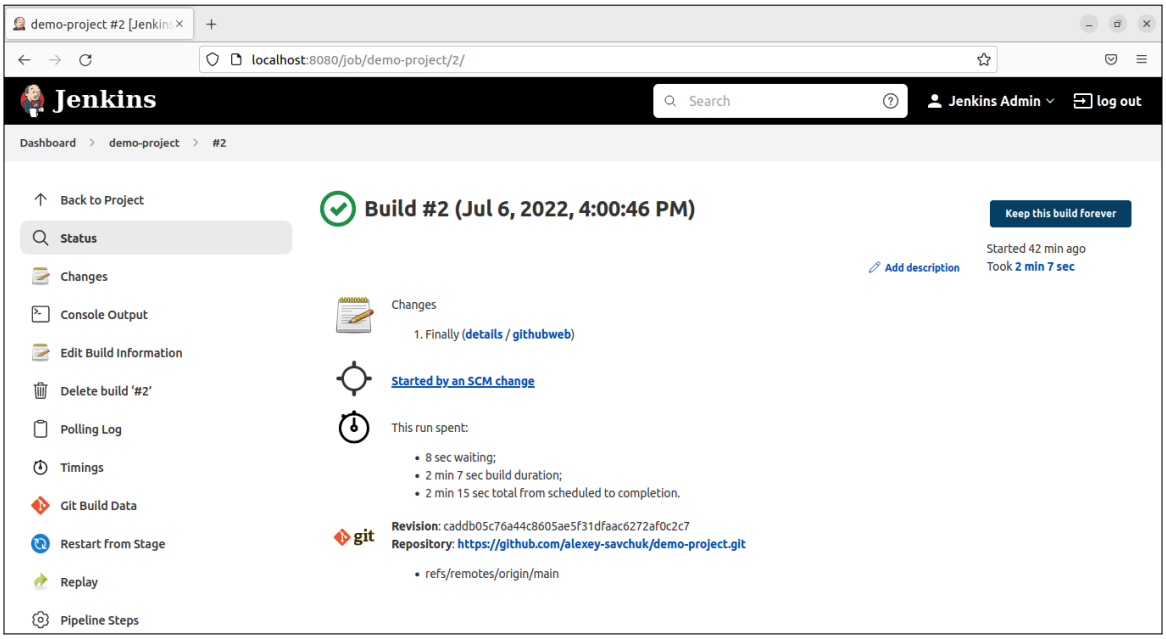
   На этом этапе происходит развертывание приложения в Kubernetes кластере.

Каждый этап пайплайна запускается в отдельном контейнере. Конфигурации этих контейнеров, а также все команды записаны в файле Jenkinsfile в формате декларативного пайплайн.

```
stages {
  stage('Configure') {
    steps {
      container(name: 'node', shell: '/bin/bash') {
        sh 'cd app && npm install'
      }
    }
  }
  stage('Build Image') {
    steps {
      container('docker') {
        sh '''
          docker build -t ${IMAGE_NAME} .
          echo $DOCKERHUB_CREDS_PSW | docker login --username $DOCKERHUB_CREDS_USR --password-stdin
          docker push ${IMAGE_NAME}
          docker logout
        '''
      }
    }
  }
  stage('Deploy') {
    steps {
      container(name: 'build-env', shell: '/bin/bash') {
        sh 'apply_deployment_from_yaml kube/app.yaml'
        sh 'timeout 5m rollout_deployment_status kube/app.yaml'
      }
    }
  }
}
```

## 2.5 Полученные результаты

Jenkins сервер видит изменения, внесенные в код проекта, и запускает сборку. Сборка завершается без ошибок: создается Docker образ веб-приложения, которое потом успешно разворачивается внутри кластера.




Kubernetes кластер после успешного прохождения пайплайна выглядит следующим образом.

```
Every 1.0s: kubectl get --all-namespaces pod                                  ubuntu: Wed Jul  6 09:57:02 2022

NAMESPACE           NAME                                         READY   STATUS    RESTARTS   AGE
default             app-deployment-7d9f7758b5-hn5vc              1/1     Running   0          7h1m
default             csi-baremetal-controller-6f976cd4fd-rf79j    4/4     Running   0          7h26m
default             csi-baremetal-node-controller-6dcd46d9df-cddgq   1/1     Running   0          7h26m
default             csi-baremetal-node-kernel-5.4-bdfqq          4/4     Running   0          7h26m
default             csi-baremetal-operator-659bfcb457-rn4wv      1/1     Running   0          7h26m
default             csi-baremetal-se-patcher-dqx8n               1/1     Running   0          7h26m
default             csi-baremetal-se-pbxn8                       1/1     Running   0          7h26m
default             mongo-express-78fcf796b8-zj9ln               1/1     Running   5          7h26m
default             mongodb-deployment-6c5499fc57-twbjt          1/1     Running   0          7h26m
jenkins             jenkins-0                                    2/2     Running   0          7h26m
kube-system         coredns-f9fd979d6-7rwgw                      1/1     Running   0          7h27m
kube-system         coredns-f9fd979d6-nfkgf                      1/1     Running   0          7h27m
kube-system         etcd-test-control-plane                      1/1     Running   0          7h27m
kube-system         kindnet-jtcls                                1/1     Running   0          7h26m
kube-system         kindnet-xsclj                                1/1     Running   0          7h27m
kube-system         kube-apiserver-test-control-plane            1/1     Running   0          7h27m
kube-system         kube-controller-manager-test-control-plane   1/1     Running   0          7h27m
kube-system         kube-proxy-4skmb                             1/1     Running   0          7h26m
kube-system         kube-proxy-l8278                             1/1     Running   0          7h27m
kube-system         kube-scheduler-test-control-plane            1/1     Running   0          7h25m
local-path-storage  local-path-provisioner-547f784dff-j7xpj      1/1     Running   0          7h27m
```

# 3 Заключительная часть

## 3.1 Выводы

Был разработан минимально рабочий прототип для демонстрации работы средств CI/CD в Kind окружении. Были использованы ранее изученные средства Docker и Kubernetes, а также изучена CI/CD система Jenkins.

## 3.2 Дальнейшие шаги

Дальнейшие шаги в работе над прототипом могут быть связаны, например, с настройкой более тесной интеграции GitHub и Jenkins (GitHub Webhooks) и добавлением уведомлений о статусе пайплайна, приходящих на электронную почту разработчика (Email Extension Plugin), в Slack (Slack Plugin) или в Discord (Discord Notifier Plugin).

## 3.3 Исходный код проекта

Репозиторий с исходным кодом проекта доступен по ссылке.