Автоматизация процесса развёртывания приложений в кластере Kubernetes на основе применения методик CI/CD

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Автоматизация процессов в современной IT-индустрии играет важную роль в том числе в сфере развертывания приложений. Это повышает скорость на всех стадиях разработки и внедрения программного продукта, надежность, снижает риски при обновлениях и экономит ресурсы. Автоматизация в данном вопросе может быть достигнута за счет применения методологии DevOps и практик CI/CD, реализуемая посредством различных современных soft-технологии, одной из которых является цифровая платформа Kubernetes.

Исходя из вышесказанного можно сформировать актуальность исследования, которая связана с возрастающей потребностью в автоматизации развертывания приложений в Kubernetes через CI/CD, что уменьшает риски человеческого фактора и обеспечивает воспроизводимость процессов за счёт подхода "Infrastructure as Code" — инфраструктурного кода, снижающего применения ручных настроек.

Цель данной выпускной квалификационной работы состоит в создании автоматизированной системы тестирования, сборки и развертывания на основе Kubernetes и методологий CI/CD.

Для реализации цели были сформированы следующие задачи:

1. Описать методологию DevOps и практики CI/CD.
2. Выполнить обзор платформы Kubernetes.
3. Показать основные принципы развертывания приложений в Kubernetes.
4. Спроектировать и настроить виртуальный сервер и инфраструктуру.
5. Реализовать способ развертывания приложения в Kubernetes.
6. Разработать механизм автоматизации развертывания приложения и реализовать механизм на конкретном примере.
7. Протестировать корректность работы автоматизированного развертывания.

1 DEVOPS МЕТОДОЛОГИЯ И ЕЕ РЕАЛИЗАЦИЯ ЧЕРЕЗ ПЛАТФОРМУ KUBERNETES

1.1 Понятие DevOps методологии

DevOps (сокращение от англ. Development и Operations) — это общепринятое наименование методологии в сфере IT, направленной на объединение процессов разработки (Dev) и эксплуатации (Ops) программного продукта или услуги. Такая синергия позволяет, в том числе:

- ускорить выпуск продукта;
- повысить качество продукта;
- обеспечить стабильность работы продукта в целом.

Образно DevOps методологию можно охарактеризовать как целую философию работы с приложениями, в основе которой лежит всесторонняя автоматизация.

Реализация происходит с использованием практик CI/CD, где

- CI (Continuous Integration) — непрерывная интеграция, при которой изменения кода автоматически интегрируются в один программный проект. Каждая интеграция сопровождается автоматизированной сборкой и запуском тестов (юнит-тестов, интеграционных тестов);

- CD (Continuous Delivery) — непрерывная поставка — практика, которая расширяет принципы CI, добавляя автоматизацию процесса развертывания приложений в тестовые или staging-окружения. При этом код всегда находится в состоянии, готовом к релизу, но финальное развертывание в production выполняется вручную после дополнительных проверок.

Инфраструктура как код (Infrastructure as Code (IaC)) в контексте DevOps — это подход к управлению и обеспечению инфраструктуры с помощью программного кода, а не ручных настроек. IaC позволяет описывать серверы, сети, хранилища и другие компоненты в виде декларативных или императивных

конфигурационных файлов, которые можно версионировать, тестировать и развертывать автоматически [12].

Одним из важнейших условий успешности разрабатываемого программного продукта на рынке является его ускоренный выход, что можно реализовать посредством автоматизации всех этапов разработки с использованием конвейеров DevOps CI и CD.

Конвейером (в англ. варианте Pipeline) называется автоматизированная последовательность этапов (шагов, стадий) через которые проходит разрабатываемый командой специалистов код от коммита до размещения продукта на платформе, доступной для пользователя. Конвейеры обеспечивают:

1) стандартизацию процесса сборки, тестирования и доставки ПО;

2) существенно снижают риск ошибки за счет минимизации ручного вмешательства;

3) упрощают процесс объединения вновь написанного кода с существующим кодом;

4) запускают на каждом шаге различные типы тестов;

5) запускают и развертывают вновь созданный код в протестированной версии ПО для конечного пользователя.

GitOps — это современная методология управления инфраструктурой и приложениями, которая использует Git-репозиторий в качестве единого источника истины для всех конфигураций и изменений. Основная идея заключается в том, что любое изменение в системе — будь то обновление приложения, настройка инфраструктуры или модификация политик безопасности — сначала вносится в Git, после чего автоматически применяется в целевой среде с помощью специализированных инструментов [12].

Этот подход обеспечивает прозрачность, контроль версий и возможность аудита всех изменений, так как вся история модификаций сохраняется в системе контроля версий. GitOps особенно популярен в экосистеме Kubernetes, где он позволяет согласовывать желаемое состояние кластера (описанное в манифестах

YAML, Helm-чартах или Kustomize-конфигурациях) с его фактическим состоянием.

Ключевые принципы GitOps включают декларативность (описание того, что должно быть развернуто, а не как), автоматизацию синхронизации, а также непрерывный мониторинг расхождений между зафиксированными конфигурациями и реальным состоянием системы.

Преимущества GitOps проявляются в повышении надежности развертываний, ускорении процессов отката до стабильных версий и улучшении коллаборации между командами разработки и эксплуатации. Благодаря использованию привычных для разработчиков инструментов (таких как Git и pull request), методология сокращает количество ручных операций и снижает риск человеческих ошибок при управлении сложными распределенными системами.

1.2 Обзор устройства платформы Kubernetes

Платформа Kubernetes является полезным механизмом в реализации методологии DevOps, она представляет собой цифровую среду обеспечивающую оркестрирование контейнеризованными рабочими задачами и сервисами. Платформа портативная, расширяемая, с широко доступным инструментарием, разнообразным сервисом и всесторонней поддержкой. У Kubernetes доступный для общего пользования цифровой код и развитая, быстрорастущая экосистема.

Принцип работы Kubernetes заключается в запуске на Узлах (Nodes) контейнеров приложений в Подах (Pods). Узлы — виртуальные или физические машины - включают в себя сервисы, которые необходимы для запуска Подов. Эти сервисы управляются Control Plane (управляющий слой). Несколько Узлов составляют кластер, но в условиях ограниченных ресурсов Узел в кластере может быть один [6].

Под — минимальная неделимая единица развертывания, «обертка» для одного или нескольких контейнеров. Внутри Пода все контейнеры имеют общий

IP-адрес и порт, для них создано собственное сетевое пространство, хранилище и спецификации запуска. Таким образом, Под обеспечивает логический хостинг конкретного приложения и интеграцию взаимосвязанных процессов [3].

На практике Поды используются в двух основных сценариях. Наиболее распространенный вариант — Под с одним контейнером, где под выполняет роль обертки, а Kubernetes управляет жизненным циклом именно Пода, а не отдельного контейнера. Такой подход обеспечивает стандартизированное взаимодействие с оркестратором независимо от специфики контейнеризированного приложения.

Второй сценарий предполагает размещение нескольких контейнеров в одном Поде, что актуально для случаев, когда приложения тесно связаны между собой и требуют совместной работы в общем пространстве. Например, основной контейнер приложения может работать в паре с sidecar-контейнером, отвечающим за логирование или мониторинг. Однако такая практика применяется ограниченно, только когда контейнеры действительно представляют собой единое целое с точки зрения функциональности.

Деплоймент (от англ. Deployment - «развертывание») предоставляет декларативные обновления для Подов и ReplicaSets. Можно описывать желаемое состояние в Деплойменте, и контроллер Деплойментов изменяет реальное состояние на желаемое в управляемом темпе.

Отдельной задачей является управления хранилищем. Она отличается от управления вычислительными ресурсами. Подсистема персистентного хранилища (PersistentVolume) предоставляет API для пользователей и администраторов, абстрагируя детали предоставления хранилища от его использования. Для этого вводятся два новых API-ресурса: PersistentVolume и PersistentVolumeClaim.

PersistentVolume (PV, персистентное хранилище) — это выделенное хранилище в кластере, подготовленное администратором или динамически созданное с использованием Storage Classes. PV является ресурсом кластера, подобно узлу (node). PV используют те же плагины для работы с томами, что и обычные Volumes, но их жизненный цикл не зависит от отдельных Pod, которые

их используют [4,11].

PersistentVolumeClaim (PVC, запрос на выделение хранилища) — это запрос на хранилище от пользователя. Аналогично тому, как Под потребляют ресурсы узлов, PVC потребляют ресурсы PV. Под могут запрашивать определенные уровни ресурсов (CPU и память), а PVC — конкретный размер и режимы доступа [4,11].

Хотя PVC позволяют пользователям работать с абстрактными ресурсами хранилища, часто для разных задач требуются PV с различными характеристиками (например, производительностью). Администраторам кластера необходимо предоставлять разнообразные PV, отличающиеся не только размером и режимами доступа, без раскрытия пользователям деталей их реализации. Для этих целей существует ресурс StorageClass.

В Kubernetes под сервисом (Service) понимается абстракция, позволяющая обеспечить доступ к сетевому приложению, развернутому в виде одного или нескольких Подов кластера. Основное назначение сервисов в Kubernetes заключается в том, что вам не требуется изменять существующее приложение для работы с незнакомым механизмом обнаружения сервисов. Сервис позволяет сделать эту группу Подов доступной в сети для взаимодействия с клиентами [13].

Если вы используете Деплоймент для запуска приложения, он может динамически создавать и удалять Поды. Поды в Kubernetes создаются и удаляются для соответствия желаемому состоянию кластера. Они являются временными (эфемерными) ресурсами — не следует рассчитывать, что отдельный Под будет надежным и долговечным.

За маршрутизацию трафика отвечает ресурс Ingress, который предоставляет доступ к HTTP и HTTPS-маршрутам извне кластера к сервисам внутри кластера. Маршрутизация трафика контролируется правилами, определёнными в ресурсе Ingress.

Ingress может быть настроен для: предоставления сервисам внешне доступных URL-адресов, балансировки нагрузки трафика, терминирования

SSL/TLS-соединений, организации виртуального хостинга по имени домена

Ingress-контроллер отвечает за обработку Ingress-ресурсов, как правило, с использованием балансировщика нагрузки. Однако он также может конфигурировать пограничный маршрутизатор (edge router) или дополнительные фронтенды для обработки входящего трафика.

1.3 Особенности развертывания приложений в Kubernetes

Развертывания приложения в Kubernetes можно реализовать с помощью инструмента командной строки kubectl и YAML или JSON файлами, содержащими декларативную конфигурацию приложения — манифестами [1]. Kubectl — это инструмент командной строки для коммуникации с компонентами управляющего слоя Kubernetes с помощью Kubernetes API [2]. Пример манифеста можно увидеть на рисунке 1.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Рисунок 1 — Пример YAML манифеста ресурса deployment

В манифесте указываются метаданные (название и пометки определяющие конкретное приложение) и спецификация ресурса. Спецификация для каждого

ресурса будут отличаться, для deployment они содержат следующие данные:

- количество реплик приложения, которые нужно будет развернуть;

- шаблон манифеста ресурса Под, содержащий информацию об имени Пода, образе контейнера, который нужно использовать, и при необходимости переменные окружения;

- метки, идентифицирующие ресурс;

- различные аннотации.

Однако развертывание приложений в Kubernetes посредством манифестов, в особенности для больших кластеров, может оказаться проблематичным в связи с запутанностью и потенциальными синтаксическими ошибка в YAML файлах, которые могут привести к неудаче при развертывании из-за опечатки или неправильной конфигурации. Управление большим количеством манифестов вручную также непродуктивно и подвержено человеческим ошибкам.

Для решения данной проблемы были разработаны многочисленные инструменты шаблонизации, наиболее популярный из которых — Helm — менеджер пакетов для Kubernetes. Подобно тому, как пакетный менеджер операционной системы упрощает установку программного обеспечения, Helm упрощает развертывание приложений и ресурсов в кластерах Kubernetes, позволяя реализовать эту задачу одной командой. Пример команды Helm можно увидеть на рисунке 2 [5].

```
$ helm install happy-panda bitnami/wordpress
```

Рисунок 2 — Пример команды Helm

На данном рисунке happy-panda – это название приложения, а bitnami/wordpress – название чарта, который нужно использовать при развертывании.

Helm использует чарты (charts) для упаковки приложений. Чарт — это пакет

Helm, содержащий всю необходимую информацию для установки набора ресурсов Kubernetes в кластер [9].

Для генерации Kubernetes манифестов Helm использует шаблоны — файлы написанные на языке программирования Go. Шаблоны позволяют динамически создавать ресурсы Kubernetes с помощью подстановки значений из файла values.yaml или аргументов командной строки.

Helm – инструмент для шаблонизирования манифестов Kubernetes, но установка многоуровневых приложений или групп приложений может оказаться громоздкой задачей. Helmfile адресует это проблему предоставляя простой, но эффективный способ декларативного развертывания чартов Helm.

Helmfile — это декларативный инструмент для развертывания Helm-чартов. Он расширяет возможности Helm, предоставляя продвинутые функции оркестрации через YAML-конфигурационный файл (как правило helmfile.yaml), пример которого можно увидеть на рисунке 3 [10].

```yaml
repositories:
 - name: prometheus-community
   url: https://prometheus-community.github.io/helm-charts

releases:
- name: prom-norbac-ubuntu
  namespace: prometheus
  chart: prometheus-community/prometheus
  set:
  - name: rbac.create
    value: false
```

Рисунок 3 — Пример файла helmfile.yaml

С его помощью можно:

- хранить каталог value-файлов для чартов и отслеживать изменения в системе контроля версий (например, Git);

- интегрировать конфигурации с CI/CD для автоматического применения изменений;

- периодически синхронизировать состояния, чтобы избежать расхождений

между средами.

Helmfile сравнивает список чартов Helm и их значения, прописанные в конфигурационном файле, с настоящим состоянием приложений в кластере Kubernetes, и Helm реагирует на любую разницу, развертывая приложения с актуальной конфигурацией [14].

Выводы по главе 1

В главе 1 выпускной квалификационной работы выполнено следующее:

1. Дано понятие методологии DevOps и практики автоматизации CI/CD.

2. Рассмотрены принципы работы, основы и ресурсы платформы Kubernetes.

3. Представлены особенности развертывания приложений в Kubernetes, как путем нативных манифестов, так и с использованием инструмента шаблонизации Helm.

2 АВТОМАТИЗАЦИЯ РАЗВЕРТЫВАНИЯ ПРИЛОЖЕНИЯ В KUBERNETES

2.1 Постановка задачи. Проектирование и настройка виртуального сервера и инфраструктуры

Основной задачи практической части работы является разработка механизма автоматизации развертывания приложения и реализация механизма на конкретном примере.

В связи с ограниченными ресурсами было принято решение развернуть K3S кластер (K3S — это дистрибутив Kubernetes) на один узел. Для хранения данных использовалась СУБД PostgreSQL, для управления базами данных — DB Operator. Чтобы декларативно прописать развертывание инфраструктуры использовался Helmfile, для Vaultwarden был сделан Helm chart и настроена автоматическая сборка контейнера с помощью Github Actions, а для развертывания настроен Argo CD.

На первом этапе реализации процесса автоматизации была создана и настроена виртуальная машина (ВМ). В качестве операционной системы принято решение использовать Alt-p11-jeos — которая является легковесной версией ОС AltLinux-p11, разрабатываемой российской компанией Базальт СПО.

Сама виртуальная машина была развернута в Proxmox Virtual Environment — системе виртуализации с открытым исходным кодом, основанной на Debian GNU/Linux, разрабатываемой австрийской фирмой Proxmox Server Solutions GmbH.

После создания ВМ был настроен конфиг OpenSSH для удалённого управления сервером через интернет. Далее было произведено полное обновление системы с помощью менеджера пакетов apt (Advanced Packaging Tool), работающего в операционных системах Debian.

Второй этап реализации задачи включал развертывание кластера K3S. K3S предоставляет инсталляционный скрипт, представляющий из себя удобный способ установки в качестве сервиса в системах, основанных на systemd. Ниже представлена команда для установки:

curl -sfL https://get.k3s.io | INSTALL_K3S_VERSION='v1.32.2+k3s1'
INSTALL_K3S_EXEC="server" sh -s - --disable traefik --disable servicelb --disable local-storage ,
где

- команда “curl -sfL https://get.k3s.io” скачивает установочный скрипт;
- sh -s запускает скрипт после установки;
- переменная “INSTALL_K3S_VERSION='v1.32.2+k3s1'” указывает версию;
- переменная “INSTALL_K3S_EXEC='server'” указывает на тип узла;
- флаги “--disable traefik”,“--disable servicelb” и “--disable local-storage” отключают установку перечисленных сервисов.

Kubectl — это инструмент командной строки для коммуникации с компонентами управляющего слоя Kubernetes с помощью Kubernetes API (набор правил и протоколов). При установке K3S на сервер также устанавливается и Kubectl. На локальное устройство был скачан бинарный файл с Kubectl.

Для настройки конфигурационного файла была использована Ansible роль для создания пользователя Kubernetes. Ansible — система управления конфигурациями, написанная на языке программирования Python, с использованием декларативного языка разметки для описания конфигураций.

Для большей стабильности системы конфигурации приложений были прописаны декларативно с использованием Helmfile. Структуру файлов можно увидеть на рисунке 4.

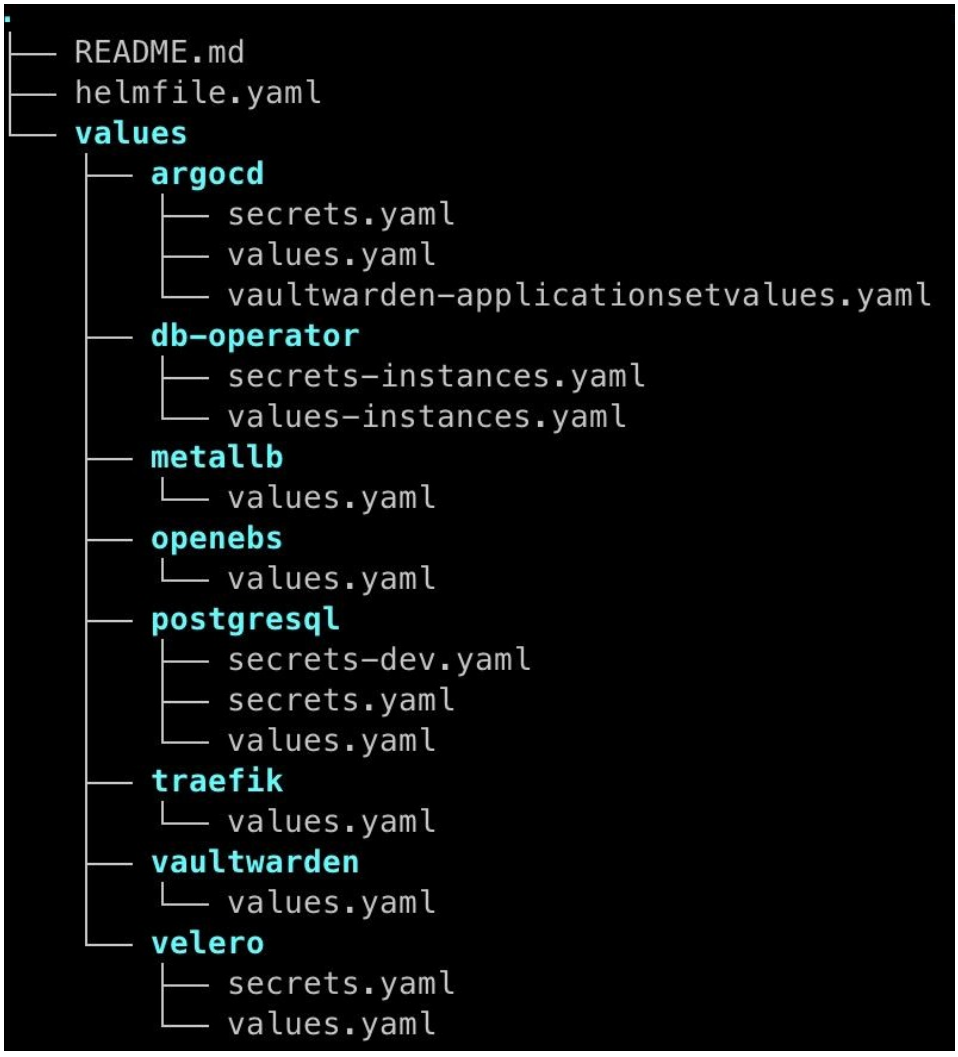

Рисунок 4 - Структура файлов в директории Helmfile

На рисунке 4 в файле helmfile.yaml находится перечень релизов. В файлах values.yaml прописана конфигурация приложений. В файлах secrets.yaml зашифрована чувствительная часть конфигурации.

Для шифрования конфиденциальных данных было принято решение использовать комбинацию Age и Sops. Age — это простой, современный и безопасный инструмент для шифрования файлов, а Sops — гибкий инструмент для управления секретами, который может работать с YAML файлами.

Чтобы использовать Age и Sops необходимо сперва установить их на свою рабочую станцию, в рамках работы это было выполнено с помощью пакетного менеджера Brew.

Сначала был создан ключ Age, для этого есть команда «age-keygen», она генерирует приватный и публичный ключи. Потом эти ключи были сохранены в текстовый файл. Чтобы Sops знал расположение файла, содержащего ключи, нужно задать переменную окружения, указывающую путь до этого файла. Это можно сделать с помощью команды, предоставленной ниже:

export SOPS_AGE_KEY_FILE=~/.config/sops/age/key.txt .

Сам путь должен соответствовать расположению файла с ключами. Данная команда работает в рамках сессии в терминале, чтобы команда работа во всех сессиях её необходимо прописать в файле

Чтобы получить зашифрованный секрет есть команда:

sops --encrypt values/redis/secrets.yaml .

Путь должен соответствовать расположению файла, который планируется зашифровать. Эта команда выводит в терминал зашифрованный файл, который потом можно скопировать и вставить в соответствующий секрет. С зашифрованными файлами можно удобно работать в Visual Studio Code, так как там есть плагин для работы с Sops.

Для хранения данных было принято решение использовать PostgreSQL. Это бесплатная СУБД с открытым исходным кодом, которая позволяет хранить, управлять и извлекать данные. Причина выбора - стабильная популярность, это можно увидеть на рисунке 5 [8].

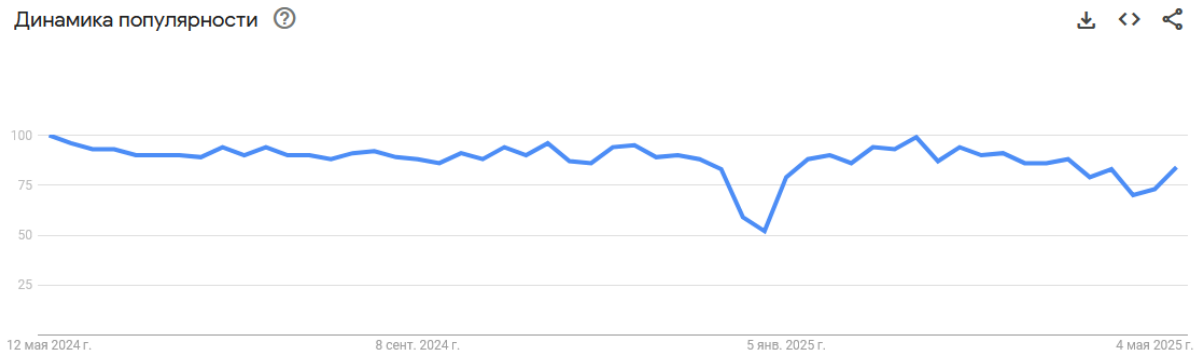

Рисунок 5 - Популярность запроса PostgreSQL в Google

Для облегчения управление инстанциями PostgreSQL и MySQL для приложений, развернутых в Kubernetes, есть инструмент DB Operator. Оператор создает БД и делает их доступными с помощью пользовательских ресурсов. Он разработан для поддержки создания тестовых окружений по запросу в CI/CD конвейерах.

Следующий шаг — развертывание приложений в кластер с помощью Helmfile. Для этого есть команды [10]:

- helmfile apply — применяет изменения в кластер;

- helmfile sync — синхронизирует конфигурацию кластера.

Сперва принято синхронизировать конфигурацию командой «sync», так как это позволяет избежать ошибки с отсутствием CRD. В дальнейшем для применения изменений можно использовать «apply», потому что это действие будет занимать меньше времени.

После успешного выполнения команды можно проверить состояние развернутых приложений с помощью Kubectl. Для этого есть команды [3]:

- Kubectl get pod — позволяет получить список развернутых Подов в текущем пространстве имен. При этом не обязательно искать конкретно «pod», можно выбрать любой ресурс;

- Kubectl describe pod $name — позволяет узнать текущий статус пода, где $name — имя пода. Также можно использовать для любого ресурса;

- Kubectl logs $name — позволяет получить логи пода.

- Флаг --namespace — позволяет указать пространство, которое должен смотреть Kubectl.

2.2 Создание Helm чарта для Vaultwarden

Helm чарт (Chart) — это формат упаковки приложений для Kubernetes, который содержит все необходимые ресурсы и настройки для их развертывания.

С помощью Helm CLI можно создать чарт, за это отвечает команда:

```
Helm Create $ChartName    .
```

Эта команда создает папку с чартом и все необходимые файлы. В рамках проекта эта папка находится в форке Vaultwarden, так как это упрощает работу, но для проектов с большим количеством чартов создается отдельная библиотека. На рисунке 6 можно увидеть структуру папки charts, в которой находится Helm чарт проекта.

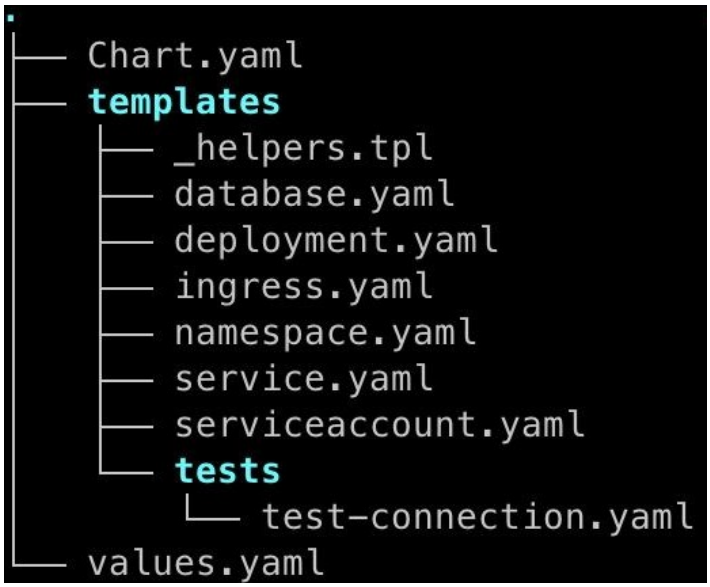

Рисунок 6 - Структура чарта

В структуру чарта на рисунке 6 входят следующие файлы:

- файл Chart.yaml содержит информацию о чарте: название, описание и версию;

- в папке Templates находятся шаблоны манифестов ресурсов Kubernetes, которые будут созданы в процессе развертывания приложения;

- файл values.yaml содержит дефолтные значения для переменных, которые подставляются в шаблоны. Значения этих переменных используются если значения для них не было указано при развертывании с помощью Helm CLI или в качестве файла values.yaml для Helmfile.

Ниже представлены два рисунка, где на рисунке 7 показана часть шаблона манифеста deployment.yaml, а на рисунке 8 — значения из файла values.yaml, которые подставляются в шаблон.

```
env:
{{- range $key, $value := .Values.env }}
  - name: {{ $key  }}
    value: {{ quote $value }}
{{- end }}
```

Рисунок 7 - Отрывок из шаблона deployment.yaml

```
env:
  DOMAIN: https://example.net
  SIGNUPS_ALLOWED: 'true'
  SIGNUPS_VERIFY: 'false'
  WEB_VAULT_ENABLED: 'true'
  ADMIN_TOKEN: qwerty123
```

Рисунок 8 - Отрывок из файла values.yaml

В шаблоне для подстановки значений используется ключевое слово range для итерации по элементам коллекций. Ключ $key — это название переменной из файла values.yaml, а $value — значение. Результат после подстановки можно увидеть на рисунке 9, где представлена часть манифеста с подставленными значениями [9].

```
- env:
  - name: ADMIN_TOKEN
    value: qwerty123
  - name: DOMAIN
    value: https://example.net
  - name: SIGNUPS_ALLOWED
    value: "true"
  - name: SIGNUPS_VERIFY
    value: "false"
  - name: WEB_VAULT_ENABLED
    value: "true"
  - name: DATABASE_URL
```

Рисунок 9 - Отрывок манифеста

Для автоматического развертывания приложения системой CD был сделан шаблон манифеста database.yaml. Это пользовательский ресурс для создания БД, за управление которым отвечает DB Operator. Без этого шаблона не получится динамически создавать базы данных.

Чтобы протестировать чарт в Helm CLI есть команда «helm template». Она подставляет значения из файла values.yaml в шаблон, формирует манифесты и выводит их в терминале. Также для анализа файлов можно использовать линтер Yamllint — это инструмент специально разработанный для поиска ошибок в синтаксисе или форматировании в YAML файлах.

Следующий шаг, после того как чарт реализован и протестирован — отправка фиксаций в удаленный репозиторий. Это позволит скачивать и использовать чарт при развертывании приложения на сервере.

2.3 Проектирование конвейера CI/CD

Для проектирования программы были сделаны UML диаграммы. Первая диаграмма — диаграмма деятельности, она позволяет описать шаги, которые выполнит программа. Ее можно увидеть на рисунке 10.

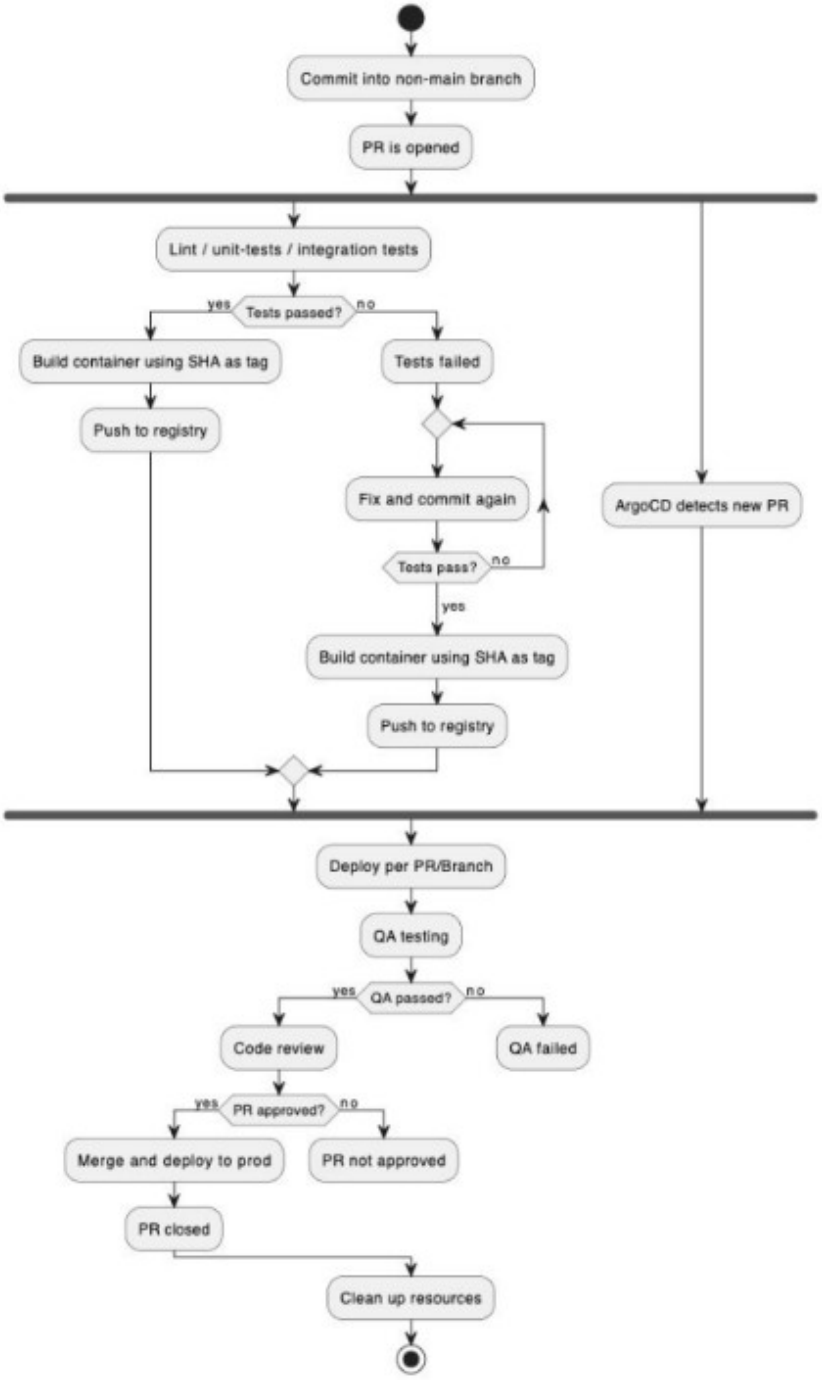

Рисунок 10 - Диаграмма деятельности

Начало работы вызывается созданием запроса на включение изменений в репозиторий, которые предлагаются в ветви. Для этого необходимо сделать commit в не основную ветку, а затем открыть pull request. Commit позволяет совершить запись изменений в репозиторий, а pull request — создать запрос на изменения кода.

Далее начинается работа программы. Вот шаги, которые выполняются программой:

1. Проводится тестирование приложения.

2. При успешном выполнение тестов приложение собирается в контейнер, и собранные образы отправляются в реестр контейнеров. Было принято решение использовать ghcr для хранения образов. Чтобы программа могла автоматически выбирать образ контейнера, тэг задается с помощью GitHub SHA.

3. Следующий шаг — развертывание приложения в тестовой среде. Для этого в Argo CD есть два пользовательских ресурса Kubernetes: Application и ApplicationSet. Application позволяет статично создать приложение, в то время как ApplicationSet дает возможность динамически создавать приложения, следовательно для реализации проекта больше подходит ApplicationSet.

Для каждого нового PR создается свое тестовое окружения, что позволяет развернуть приложение без участия человека.

Для развертывания необходимо создать пространство имен и БД. Argo CD поддерживает автоматическое создание пространств имен, а для создание БД используется DB Operator.

4. После успешного развертывания нужно выполнить обзор кода.

5. Далее — развертывание приложения в производственную среду. Процесс схож с развертыванием в тестовую среду, но пространство имен и БД — статичны.

6. Последний шаг — удаление тестовой среды при закрытии PR.

Благодаря диаграмме деятельности на рисунке 10 получилось понятно описать процессы.

Следующая диаграмма — диаграмма вариантов использования. Она отражает отношения между акторами и прецедентами. Ее можно увидеть на рисунке 11.

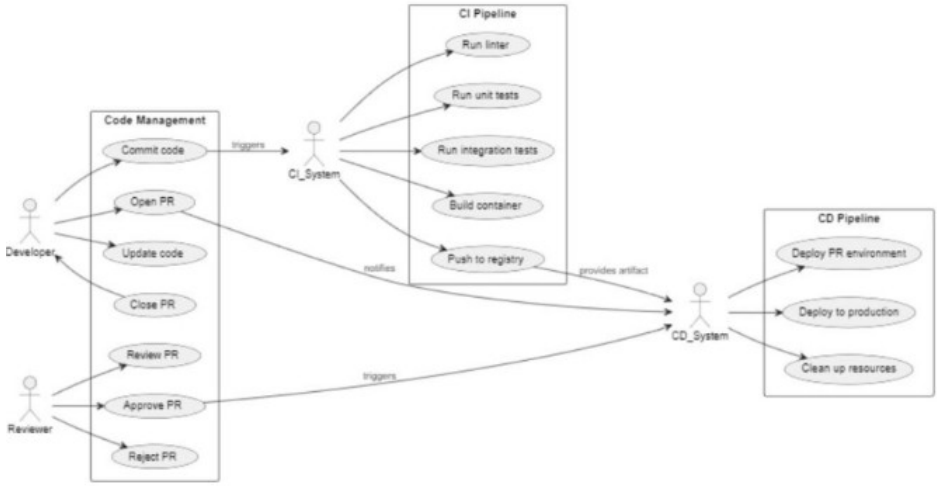

Рисунок 11 - Диаграмма вариантов использования

К акторам относятся: разработчик, рецензент, система CI и система CD. Диаграмма показывает не только связь между акторами и процессами, но и связь между участниками процесса.

Последняя диаграмма - диаграмма последовательности. Она используется для более детального описания логики сценариев использования. Ее можно увидеть на рисунке 12.

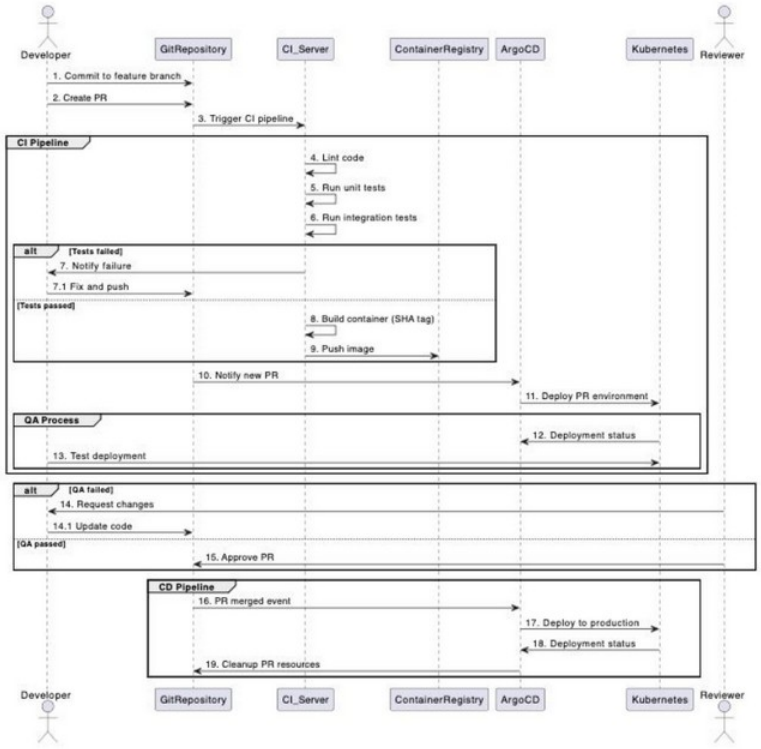

Рисунок 12 - Диаграмма последовательности

С помощью этой диаграммы представленной на рисунке 12 получилось понятно продемонстрировать все интеграции и взаимодействия в рамках проектируемой программы.

Использование UML помогло:

- структурировать логику работы системы до начала кодирования;
- выявить избыточные или недостающие функциональные блоки;
- сократить время на рефакторинг, так как ключевые сценарии были продуманы заранее.

2.4 Настройка конвейера CI для сборки Vaultwarden с помощью Github Actions

GitHub Actions - это проприетарная платформа непрерывной интеграции и поставки, разработанная компанией GitHub, Inc. Она позволяет автоматизировать сборку, тестирование и развертывание.

Github Actions имеет встроенную систему плагинов Actions (от англ. действия), которые могут выполнять разные функции и быть написаны на разных языках программирования. Это позволяет передавать параметры в уже написанные плагины, вместо того чтобы полностью описывать пайплайн.

Файлы с описанием работы CI находятся в репозитории с приложением по пути .github/workflows. Ниже, на рисунке 13, представлен файл, в котором написана работа CI для сборки образа и дальнейшей отправки в регистр контейнеров.

```
name: Build and Push Docker Image
on:
  pull_request:
jobs:
  docker:
    runs-on: ubuntu-latest
    steps:
      -
        name: Checkout code
        uses: actions/checkout@v4
      -
        name: Set up QEMU
        uses: docker/setup-qemu-action@v3
      -
        name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3
      -
        name: Build and push
        uses: macbre/push-to-ghcr@master
        with:
          image_name: ${{ github.repository }}
          github_token: ${{ secrets.TOKEN }}
          context: .
          image_tag: ${{ github.sha }}
```

Рисунок 13 - Файл с описанием работы CI

В представленном на рисунке 13 файле содержится следующая информация:

- «on: pull_request» позволяет автоматически вызывать работу CI конвейера при открытии PR. Steps — шаги, которые выполняет конвейер:

- «Checkout code» смотрит код репозитория;

- «Set up QEMU» отвечает за настройку виртуальной машины, на которой выполняется CI;

- «Set up Docker Buildx» настраивает Docker, платформу для работы с контейнерами;

- «Build and push» отвечает за сборку образа и отправку в регистр контейнеров ghcr. Для этого шага указаны переменные, необходимые для корректной сборки и доступа к регистру контейнеров. Для сборки тестового образа в качестве тэга используется GIT SHA, а для образа использующегося в

производственной среде - тэг latest.

Github Actions дает возможность интерактивно смотреть логи по мере работе конвейера и показывает статус выполнения шагов.

После успешной сборки и отправки образа его можно скачать на виртуальную машину из регистра контейнеров.

2.5 Настойка Argo CD

Argo CD представляет собой декларативный инструмент непрерывной доставки (Continuous Delivery), реализующий принципы GitOps для управления приложениями в Kubernetes-кластерах. Разработанный как часть экосистемы Argo, этот инструмент обеспечивает автоматическую синхронизацию состояния кластера с конфигурациями, хранящимися в системе контроля версий (например, Git-репозитории) [7].

Основная идея Argo CD заключается в использовании Git как единственного источника истины для всех конфигураций и манифестов развертывания. Инструмент постоянно отслеживает различия между фактическим состоянием кластера и желаемым состоянием, описанным в репозитории, автоматически или по запросу приводя кластер в соответствие с декларативными описаниями.

Argo CD поддерживает различные инструменты описания инфраструктуры, включая нативные Kubernetes-манифесты, Helm-чарты и Kustomize-конфигурации. Благодаря встроенному веб-интерфейсу и CLI, он предоставляет прозрачный механизм наблюдения за состоянием приложений, позволяя быстро выявлять и устранять расхождения между ожидаемой и текущей конфигурацией.

Особенностью Argo CD является его способность работать в multi-tenant и multi-cluster окружениях, обеспечивая безопасность через механизмы RBAC и интеграцию с SSO-провайдерами. Инструмент особенно востребован в средах, где критически важны воспроизводимость развертываний, контроль изменений и

соответствие требованиям compliance.

В отличие от традиционных CI/CD-пайплайнов, Argo CD реализует pull-подход, когда кластер самостоятельно получает актуальные конфигурации, что повышает безопасность и снижает нагрузку на систему сборки.

ApplicationSets представляют собой способ декларативного описания динамически развертываемого приложения. В рамках данной работы, все конфигурация приложения описана в файле со значениями vautlwarden-applicationsetvalues.yaml [7].

За начало работы CD конвейера в Argo CD отвечают генераторы. Пример генератора можно увидеть на рисунке 14, данный генератор работает при создании PR в указанном репозитории.

```yaml
generators:
- pullRequest:
    github:
      owner: jack-lull
      repo: vaultwarden
      tokenRef:
        secretName: github-token
        key: token
```

Рисунок 14 — Генератор

Для создания ApplicationSet используется шаблон соответствующего пользовательского ресурса, этот шаблон представлен на рисунке 15.

```yaml
template:
  metadata:
    name: 'vaultwarden-pr-{{ .number }}'
    labels:
      project: 'vaultwarden-pr'
    annotations: {}
  spec:
    project: 'default'
    source:
      repoURL: https://github.com/jack-lull/vaultwarden
      targetRevision: '{{ .head_sha }}'
      path: chart
    destination:
      server: https://kubernetes.default.svc
      namespace: 'app-prewiev-{{ .number }}'
    syncPolicy:
      automated:
        prune: false
        selfHeal: false
      syncOptions:
      - CreateNamespace=true
```

Рисунок 15 - Шаблон ApplicationSet

В шаблоне ApplicationSet содержится следующая информация:

- metadata — общие сведения о проекте, к примеру название;

- в source указывается путь до Helm чарта;

- destination указывает, где ApplicationSet должен создать приложение;

- syncOptions: CreateNamespace=true дает ApplicationSet право динамически создавать пространства имен.

Когда значения для ApplicationSet готовы, нужно принять изменения с помощью команды «helmfile apply». Чтобы проверить работу CD можно создать PR в репозитории и проверить успешно ли выполняется развертывание приложения.

2.6 Тестирование работы конвейера CI/CD

Тестирование проводилось на всех этапах работы по автоматизации развертывания приложения. Все выявленные ошибки устранялись по мере возникновения. Основные выявленные ошибки — синтаксические.

Далее, с помощью разработанного конвейером CI/CD, было принято решение произвести обновление приложения Vaultwarden. В приложение была встроена картинка, которую пользователь видит при получение ошибки 404 «Return to the web vault?» (рисунок 16).

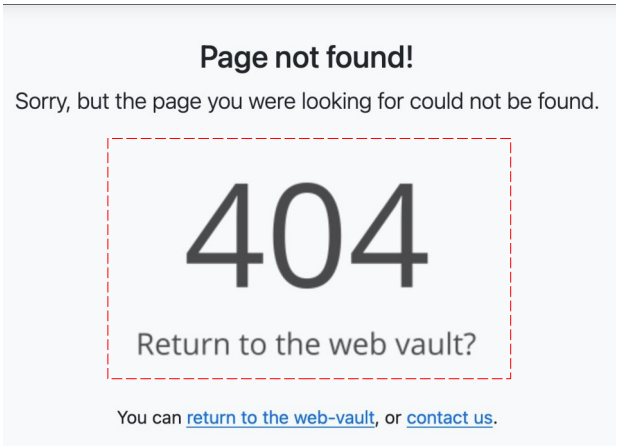

Рисунок 16 - Страница до изменений

На рисунке XX красной пунктирной линией выделена область, которую предполагается заменить на другое изображение.

Чтобы протестировать работу конвейера, файл, содержащий данное изображение, был изменен на файл с новым изображением.

Далее изменения были отправлены в удаленный репозиторий и был открыт новый PR. Это начало работу конвейера. После успешного выполнения работы конвейера появилось развернутое тестовое приложения. С помощью команды

kubectl port-forward pod $podName 8080:80,

где $podName – название Пода;

был получен доступ к Поду с локального устройства. После перехода на Web страницу по ссылке «localhost:8080/nonexistent», вместо исходного, отобразилось измененное изображение (рисунок 17).

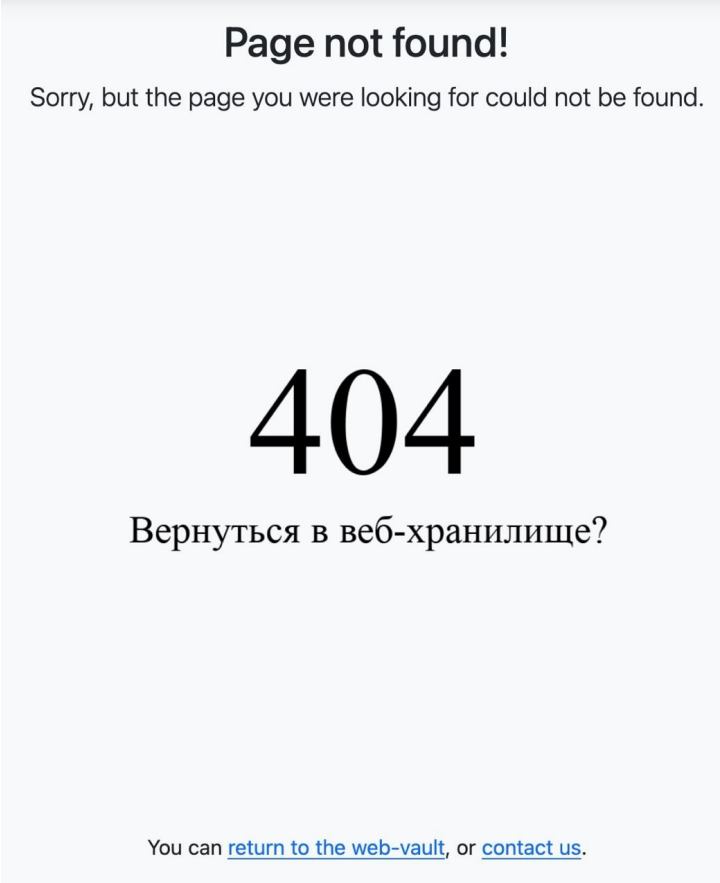

Рисунок 17 — Страница после изменения

Изменение изображения доказывает работу конвейера для тестового окружения, чтобы проверить производственное окружение нужно фиксировать изменения в основную ветку. После замены файла в основной ветке собрался контейнер с тэгом latest. Для обновления производственной версии приложения Vaultwarden проведена синхронизация в UI Argo CD. Разработку механизма автоматизации развертывания приложения и реализацию механизма можно считать успешной.

Выводы по главе 2

В главе 2 выпускной квалификационной работы выполнено следующее:

1. Спроектирован и настроен виртуальный сервер на базе операционной системы Alt-p11-jeos. В кластере K3S организована необходимая для автоматического развертывания приложения инфраструктура: СУБД PostgreSQL, DB Operator, Argo CD.

2. Реализован Helm чарт и все необходимые для приложения шаблоны манифестов.

3. Спроектирован механизм работы конвейера CI/CD с использованием унифицированного языка моделирования UML.

4. Реализована автоматическая сборка и отправка образа контейнера с приложением с использованием платформы GitHub Actions.

5. Настроена конфигурация декларативного инструмента непрерывной доставки Argo CD для динамического развертывания тестовой и производственной версии приложения в кластер.

6. Корректность работы конвейера протестирована путем обновления приложения.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы было реализовано следующее:

1. Дано понятие методологии DevOps и практики автоматизации CI/CD.

2. Рассмотрены принципы работы, основы и ресурсы платформы Kubernetes.

3. Представлены особенности развертывания приложений в Kubernetes, как путем нативных манифестов, так и с использованием инструмента шаблонизации Helm.

4. Спроектирован и настроен виртуальный сервер на базе операционной системы Alt-p11-jeos. В кластере K3S организована необходимая для автоматического развертывания приложения инфраструктура: СУБД PostgreSQL, DB Operator, Argo CD.

5. Реализован Helm чарт и все необходимые для приложения шаблоны манифестов.

6. Спроектирован механизм работы конвейера CI/CD с использованием унифицированного языка моделирования UML.

7. Реализована автоматическая сборка и отправка образа контейнера с приложением с использованием платформы GitHub Actions.

8. Настроена конфигурация декларативного инструмента непрерывной доставки Argo CD для динамического развертывания тестовой и производственной версии приложения в кластер.

9. Корректность работы конвейера протестирована путем обновления приложения.

В результате успешно достигнута поставленная цель - создание автоматизированной системы тестирования, сборки и развертывания на основе Kubernetes и методологий CI/CD.

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

DevOps— Development and Operations

БД — База данных

СУБД — Система управления базами данных

ВМ — Виртуальная машина

CI — Continuous integration

CD — Continuous delivery

CRD — Custom resource definition

ghcr — GitHub Container Registry

PR — pull request

CLI — Command Line Interface

UI — User Interface

СЛОВАРЬ ТЕРМИНОВ

Ресурс представляет собой конечную точку (endpoint) в API Kubernetes, которая хранит коллекцию объектов API определённого типа. Например встроенный ресурс pods содержит коллекцию объектов Pod

Пользовательский ресурс (Custom Resource) — это расширение Kubernetes API, которое не входит в стандартную поставку Kubernetes. Он позволяет настраивать конкретный кластер под определенные задачи. При этом многие базовые функции Kubernetes сейчас реализованы через такие ресурсы, что делает систему более модульной [4,11].

CRD (Custom Resource Definition) – это способ расширить API Kubernetes, добавляя в него собственные типы ресурсов. Это позволяет создавать и управлять объектами, которые не являются стандартными для Kubernetes, но могут быть необходимы для конкретного приложения или инфраструктуры [3].

Секрет (Secret) - это объект, содержащий небольшое количество конфиденциальных данных, например пароли, токены или ключи. Секреты схожи с ConfigMaps, но предназначены для хранения чувствительных данных.

Операторы представляют собой программные расширения для Kubernetes, которые используют пользовательские ресурсы для управления приложениями и их компонентами. Они работают в соответствии с принципами Kubernetes, в частности, реализуя контрольный цикл (control loop).

Логи (или log-файлы) - это текстовые файлы, в которые автоматически записывается информация о работе системы, программы или сервера.

SHA - это хэш сумма, определяющая последний коммит в git репозиторий и являющаяся идентификатором этого коммита.

Тэг образа контейнера (также называемый tag) - это метка, используемая для идентификации конкретной версии образа.

UML, или Unified Modeling Language, — это унифицированный язык

моделирования. Его используют, чтобы создавать диаграммы и схемы для визуализации процессов и явлений.

Артефакты в программировании — вспомогательные (как правило, созданные в процессе исполнения программного обеспечения) элементы продукта, так или иначе входящие в его состав.

Манифесты Kubernetes по сути являются файлами в формате YAML или JSON, которые описывают желаемое состояние объектов API Kubernetes в кластере [3].

CI/CD-пайплайн (CI/CD pipeline) расшифровывается как «конвейер непрерывной интеграции и непрерывного развертывания».

Интерфейс командной строки (англ. Command line interface, CLI) или командная оболочка — способ взаимодействия между человеком и компьютером путём отправки компьютеру команд, представляющих собой последовательность символов.

UI (User Interface) - это пользовательский интерфейс, то есть всё, что видит и с чем взаимодействует пользователь при использовании приложения или сайта. Это визуальная часть, включающая цвета, шрифты, иконки, кнопки и другие элементы, определяющие внешний вид.

Дистрибутив (от английского слова distribute - распространять) - это форма распространения программного обеспечения. Он представляет собой набор файлов, архивов и других компонентов, необходимых для установки и работы программы или операционной системы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Бёрнс Б., Вильяльба Э., Штребель Д., Эвенсон Л. Kubernetes: лучшие практики. Раскрой потенциал главного инструмента в отрасли. — Спб:. Питер, 2021. — 288 с.

2. Вьяс Д., Лав К. Kubernetes изнутри — ДМК Пресс, 2022. — 365 с.

3. Документация по Kubernetes [Электронный ресурс]. URL: https://kubernetes.io/ru/docs/home/ (дата обращения: 24.04.2025).

4. Лукша М. Kubernetes в действии / пер. с англ. А. В. Логунов. – ДМК Пресс, 2019. – 672 с

5. Маркелов А. А. Введение в технологии контейнеров и Kubernetes. – ДМК Пресс, 2019. – 194 с.

6. Сайфан Д. Осваиваем Kubernetes. Оркестрация контейнерных архитектур. – Спб:. Питер, 2019. — 400 с.

7. Argo CD - Declarative GitOps CD for Kubernetes [Электронный ресурс]. URL: https://argo-cd.readthedocs.io/en/stable/ (дата обращения: 7.04.2025).

8. Google Trends [Электронный ресурс]. URL: https://trends.google.ru/trends/explore (дата обращения: 15.04.2025).

9. Helm Docs [Электронный ресурс]. URL: https://helm.sh/docs/ (дата обращения: 30.04.2024).

10. Helmfile Documentation [Электронный ресурс]. URL https://helmfile.readthedocs.io/en/latest/ (дата обращения: 13.04.2025).

11. Luksa M. Kubernetes in action. - Simon and Schuster, 2017. - 624 с.

12. Paavola E. Managing Multiple Applications on Kubernetes Using GitOps Principles / E. Paavola. – 2021. – 41 с.

13. Poulton N. The kubernetes book. - NIGEL POULTON LTD, 2023. - 476 с.

14. What is Helmfile and How Does it Work? [Электронный ресурс]. URL https://blogs.vmware.com/tanzu/what-is-helmfile-and-how-does-it-work/ (дата обращения: 25.04.2025).