

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“САНКТ – ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИТМО”

КУРСОВАЯ РАБОТА

Тема: Исследование возможностей отказоустойчивости приложений, эксплуатируемых в Kubernetes

Работу выполнила Фрид Зинаида Александровна студент группы Р4223

Руководитель Штенников Дмитрий Геннадьевич

Работа защищена " _____ " _____ 2025 г.

с оценкой _____

Подписи членов комиссии: _____

Санкт-Петербург, 2025 г

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1. ТЕОРЕТИЧЕСКИЙ ОБЗОР	4
1.1. Основные концепции Kubernetes	4
1.2. Механизмы отказоустойчивости.....	4
1.3. Стратегии отказоустойчивости	5
2. ДИЗАЙН ЭКСПЕРИМЕНТА И УСТАНОВКА ОКРУЖЕНИЯ.....	7
2.1. Описание эксперимента	7
2.2. Развёртывание окружения	9
3. РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТА ДОСТУПНОСТИ ПРИЛОЖЕНИЯ.....	16
ЗАКЛЮЧЕНИЕ	19
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ	20

ВВЕДЕНИЕ

В современном мире информационных технологий, где цифровизация и автоматизация процессов становятся ключевыми факторами успеха, надежность и отказоустойчивость приложений играют критическую роль. Одной из наиболее популярных и мощных платформ для управления контейнеризированными приложениями является Kubernetes. Эта система оркестрации контейнеров предоставляет широкий спектр возможностей для обеспечения высокой доступности, масштабируемости и устойчивости приложений.

Цель данного исследования — изучить возможности отказоустойчивости приложений, эксплуатируемых в Kubernetes [1]. В рамках работы будут рассмотрены основные механизмы и стратегии, которые Kubernetes предлагает для обеспечения непрерывной работы приложений в условиях сбоев и непредвиденных обстоятельств. Мы проанализируем такие аспекты, как репликация, балансировка нагрузки, автоматическое восстановление и распределение ресурсов, а также рассмотрим лучшие практики и рекомендации по их применению.

Важность данного исследования обусловлена тем, что в условиях растущей сложности ИТ-инфраструктуры и увеличения объема обрабатываемых данных, обеспечение отказоустойчивости становится неотъемлемой частью стратегии любой организации. Kubernetes, благодаря своей гибкости и мощным инструментам, предоставляет разработчикам и администраторам возможность создавать надежные и устойчивые системы, способные выдерживать высокие нагрузки и минимизировать время простоя.

1. ТЕОРЕТИЧЕСКИЙ ОБЗОР

Kubernetes — это открытая платформа для автоматизации развертывания, масштабирования и управления контейнеризированными приложениями. Одной из ключевых особенностей Kubernetes является его способность обеспечивать высокую доступность и отказоустойчивость приложений. В этом теоретическом обзоре мы рассмотрим основные механизмы и стратегии, которые Kubernetes предлагает для достижения этих целей.

1.1. Основные концепции Kubernetes

Pods: Основная единица развертывания в Kubernetes. Pod может содержать один или несколько контейнеров, которые совместно используют ресурсы и сетевые пространства.

ReplicaSets: обеспечивают поддержание заданного количества реплик Pods, что позволяет автоматически восстанавливать Pods в случае их сбоя.

Deployments: управляют развертыванием и обновлением Pods, обеспечивая плавное обновление приложений без простоев.

Services: обеспечивают стабильный сетевой доступ к Pods, даже если они перемещаются или перезапускаются.

1.2. Механизмы отказоустойчивости

1. Репликация.

ReplicaSets и Deployments позволяют автоматически создавать и удалять Pods для поддержания заданного количества реплик. Это обеспечивает высокую доступность приложений, даже если отдельные Pods выходят из строя.

Horizontal Pod Autoscaler (HPA) автоматически масштабирует количество Pods на основе метрик, таких как использование CPU или пользовательские метрики.

2. Балансировка нагрузки.

Services в Kubernetes обеспечивают балансировку нагрузки между Pods, что позволяет равномерно распределять трафик и предотвращать перегрузку отдельных Pods.

Ingress Controllers предоставляют дополнительные возможности для управления входящим трафиком, включая SSL-терминацию и маршрутизацию на основе URL.

3. Автоматическое восстановление.

Kubernetes автоматически перезапускает Pods, которые завершились с ошибкой, и перемещает Pods на другие узлы в случае сбоя узла.

Liveness и Readiness Probes позволяют Kubernetes проверять состояние Pods и автоматически перезапускать или удалять Pods, которые не проходят проверки.

4. Распределение ресурсов.

Resource Requests и Limits позволяют задавать минимальные и максимальные ресурсы (CPU, память) для Pods, что помогает предотвратить перегрузку узлов и обеспечивает стабильную работу приложений.

Node Affinity и Anti-Affinity позволяют контролировать распределение Pods по узлам, что помогает избежать концентрации Pods на одном узле и повышает отказоустойчивость.

1.3. Стратегии отказоустойчивости

1. Многозональное развертывание.

Развертывание Pods в нескольких зонах доступности (Availability Zones) позволяет обеспечить высокую доступность приложений даже в случае сбоя одной зоны.

Pod Anti-Affinity помогает распределить Pods по разным зонам, минимизируя риск одновременного сбоя всех Pods.

2. Резервное копирование и восстановление.

Использование инструментов для резервного копирования данных и состояния приложений, таких как Velero, позволяет быстро восстановить приложения в случае катастрофы.

StatefulSets обеспечивают управление состоянием приложений, что позволяет сохранять и восстанавливать состояние Pods.

3. Мониторинг и алертинг.

Интеграция с системами мониторинга, такими как Prometheus и Grafana, позволяет отслеживать состояние кластера и приложений в реальном времени.

Настройка алертов помогает оперативно реагировать на потенциальные проблемы и предотвращать сбои.

Kubernetes предоставляет мощные инструменты и механизмы для обеспечения отказоустойчивости приложений. Использование репликации, балансировки нагрузки, автоматического восстановления и распределения ресурсов позволяет создавать надежные и устойчивые системы. Стратегии многозонального развертывания, резервного копирования и мониторинга дополнительно повышают устойчивость приложений к сбоям. Внедрение этих практик и технологий помогает организациям обеспечить высокую доступность и надежность своих приложений в условиях динамично меняющейся ИТ-инфраструктуры.

2. ДИЗАЙН ЭКСПЕРИМЕНТА И УСТАНОВКА ОКРУЖЕНИЯ

В данном разделе будет приведено описание эксперимента по исследованию отказоустойчивости приложений, эксплуатируемых в Kubernetes, а также полная инструкция по созданию тестового окружения для выполнения данного исследования.

2.1. Описание эксперимента

Для исследования возможностей отказоустойчивости приложений необходимо развернуть тестовое приложение в кластере Kubernetes на нескольких Pods, предоставив возможность влиять на состояние этих модулей, например, путем удаления Pods. Это активирует механизмы отказоустойчивости Kubernetes, эффективность которых и требуется изучить.

В качестве метрики эффективности отказоустойчивости предлагается использовать время доступности приложения, выраженное в процентах от общего времени его работы. Это можно реализовать путем создания внешней периодической проверки, которая будет регистрировать процент успешных обращений к приложению за определенное время.

Таким образом, можно будет измерить доступность приложения в зависимости от количества развернутых реплик, подавая на приложение постоянную нагрузку в течение одинакового временного промежутка. Это продемонстрирует эффективность восстановления приложения в Kubernetes.

В качестве тестового приложения будет использован простой веб-сервер, отображающий список файлов в директории, где он был запущен. Это можно легко реализовать с помощью Python-пакета `http.server`.

Влияние на стабильность работы приложения в кластере будет осуществляться путем удаления запущенных реплик тестового приложения с помощью специального ПО, развернутого внутри кластера Kubernetes —

Kubedoom [2]. Это приложение позволяет играть в Doom, выполняемом внутри контейнера, и удалять работающие реплики приложения внутри кластера.



Рис. 1 Интерфейс Kubedoom

Во время исследования отказоустойчивости приложения предполагается выполнение нескольких шагов для каждой итерации эксперимента:

1. конфигурируется необходимое число реплик тестового приложения внутри кластера;
2. выполняется запуск kubedoom, подключение через VNC Client и подготовка игрока (ввод чит-кодов и т. д.); игрок занимает позицию для влияния на кластер;
3. осуществляется запуск анализирующего ПО, которое выполняет как мониторинг доступности, так к небольшую DDoS-атаку на тестовое приложение внутри кластера в течение 2 минут;
4. одновременно с запуском анализирующего ПО игрок начинает отключать реплики тестового приложения в течение 2 минут, наблюдая за поведением кластера;

5. по истечении 2 минут игрок фиксирует рассчитанную доступность приложения и повторяет эксперимент для нового числа реплик тестового приложения

2.2. Развёртывание окружения

Тестирование отказоустойчивости приложения в кластере будет выполнено на Ubuntu 22.04, кластер будет поднят при помощи пакета Minikube [3], который позволяет быстро развернуть локальный кластер Kubernetes на одном устройстве для изучения и разработки под Kubernetes. Однако, данное ПО не предназначено для использования в продуктивной среде.

Ресурсы, выданные кластеру, составляют 2 ядра Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz, 3 Gb оперативной памяти и 3 worker-ноды. Команда создания такого кластера представлена на листинге 1.

```
minikube start --cpus=2 --memory=3000 -nodes=3 -driver=kvm2
```

Листинг 1. Создание локального кластера Minikube

Параметры:

--cpus=2 — количество виртуальных процессоров, выделенных для кластера.

--memory=3000 — объём оперативной памяти, выделенной для кластера (в МБ).

--nodes=3 — количество узлов в многоузловом кластере.

--driver=kvm2 — драйвер для управления виртуальными машинами (в данном случае KVM2).

Далее установим в кластер Kubernetes Dashboard для простой проверки текущего состояния и конфигурации кластера. На рисунке 2 отображен вывод информации о подключенных к кластеру узлах.

Name	Labels	CPU Ready	CPU requests (cores)	CPU limits (cores)	CPU capacity (cores)	Memory requests (bytes)	Memory limits (bytes)	Memory capacity (bytes)	Pods	Create
minikube-m03	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/os: linux kubernetes.io/arch: amd64	True	100.00m (5.00%)	100.00m (5.00%)	2.00	50.00Mi (1.72%)	50.00Mi (1.72%)	2.83Gi	2 (1.82%)	10 minutes ago
minikube-m02	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/os: linux kubernetes.io/arch: amd64	True	100.00m (5.00%)	100.00m (5.00%)	2.00	50.00Mi (1.72%)	50.00Mi (1.72%)	2.83Gi	3 (2.73%)	11 minutes ago
minikube	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/os: linux kubernetes.io/arch: amd64	True	850.00m (42.50%)	100.00m (5.00%)	2.00	220.00Mi (7.58%)	220.00Mi (7.58%)	2.83Gi	11 (10.00%)	7 hours ago

Рис. 1 Узлы кластера

Для установки в кластер тестового приложения создадим файл `web-deployment.yaml`, в котором опишем конфигурацию деплоимента с тестовым приложением таким образом, чтобы наше приложение внутри кластера было доступно на 8000 порту и было развёрнуто только на `worker`-нодах при помощи параметра `nodeAffinity`. На листинге 2 приведена конфигурация данного деплоимента.

В данной конфигурации указано создание приложения в числе 20 реплик из контейнера. В спецификации пода так же указан параметр `affinity`, в котором описана проверка правила на отсутствие специальной метки, указывающей на то, что текущий узел является управляющим. Если такой метки нет, то узел подходит для размещения пода приложения. На рисунке 3 в первом столбце вывода команды указано получившееся распределение подов по узлам после применения такой конфигурации [4].

```
(venv) rustam@ubuntu:~/Desktop/course_paper$ kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
web-8558b69556-4f5wb	1/1	Running	0	24s	10.244.2.62	minikube-m03	<none>	<none>
web-8558b69556-55rgt	1/1	Running	0	7m	10.244.1.55	minikube-m02	<none>	<none>
web-8558b69556-msvp4	1/1	Running	0	24s	10.244.2.61	minikube-m03	<none>	<none>
web-8558b69556-p8r2r	1/1	Running	0	7m2s	10.244.1.53	minikube-m02	<none>	<none>
web-8558b69556-p9tl6	1/1	Running	0	7m45s	10.244.1.49	minikube-m02	<none>	<none>

Рис. 2 Распределение подов приложения по узлам

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web
  labels:
    app: web
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: node-role.kubernetes.io/control-plane
                    operator: DoesNotExist
      containers:
        - name: web-container
          image: saitam700/kube-stress:simple-web-server
          ports:
            - name: http
              containerPort: 8000
          livenessProbe:
            httpGet:
              path: /
              port: 8000
            initialDelaySeconds: 3
            periodSeconds: 3
```

Листинг 2. Конфигурация тестового приложения в файле web-deployment.yaml

После установки тестового приложения в кластер необходимо опубликовать его для доступа извне кластера, так как для целей эксперимента необходимо измерять время доступности приложения в условиях интенсивной нагрузки на него. Лучше всего для этой задачи подойдёт абстракция сервиса (Service), которую предоставляет Kubernetes. Сервис типа NodePort публикует определённый внутренний порт кластера в один из портов, который будет

доступен извне кластера на ip-адресе, зарезервированном для доступа к этому сервису. Далее на листинге 3 приведена конфигурация данного сервиса.

```
apiVersion: v1
kind: Service
metadata:
  name: web-service
  namespace: default
spec:
  type: NodePort
  selector:
    app: web
  ports:
    - protocol: TCP
      port: 8000
      targetPort: 8000
      nodePort: 31111
```

Листинг 3. Конфигурация сервиса для доступа к приложению web извне кластера

После выполнения команд

```
cd course-paper-manifests/
kubectl apply -f web-deployment.yaml
kubectl apply -f web-service.yaml
kubectl apply -f kubedoom-service.yaml
```

станет возможным проверить работу приложения через браузер. Для этого узнаем внешний ip-адрес сервиса при помощи команды

```
minikube service web-service.
```

Данная команда возвращает ip-адрес сервиса, прикреплённый к сети кластера minikube внутри docker адаптера. При переходе по указанному ip-адресу будет загружена страница с отображением одного из файлов каталога, показанная на рисунке 5.

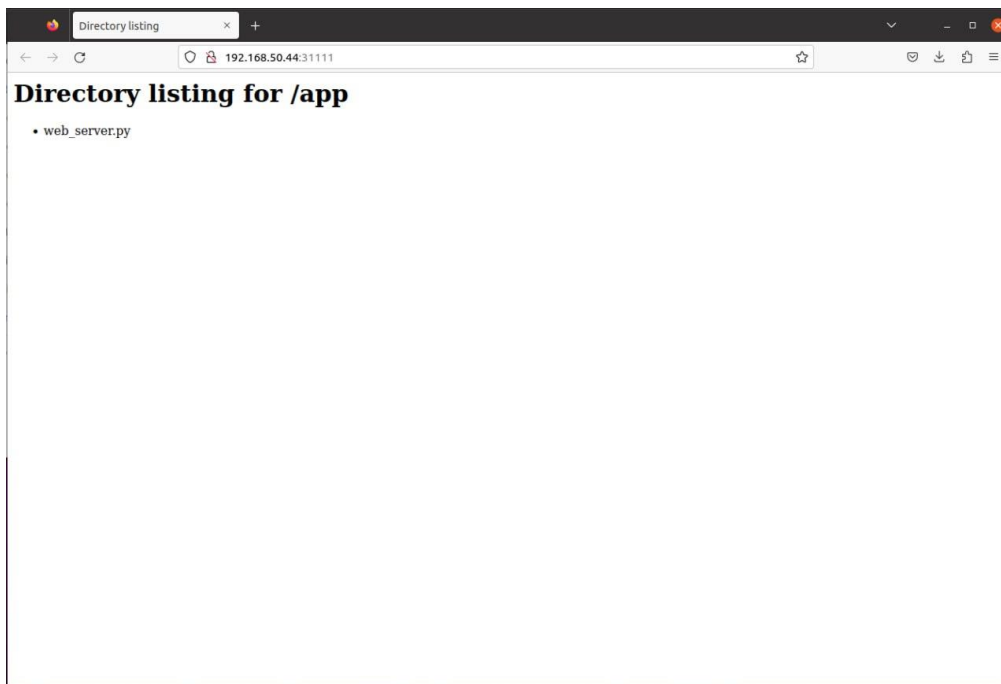


Рис. 3 Тест доступа к приложению через сервис web-service

Установка Kubedoom в кластер выполняется с использованием деплоймента из репозитория разработчика при помощи команды

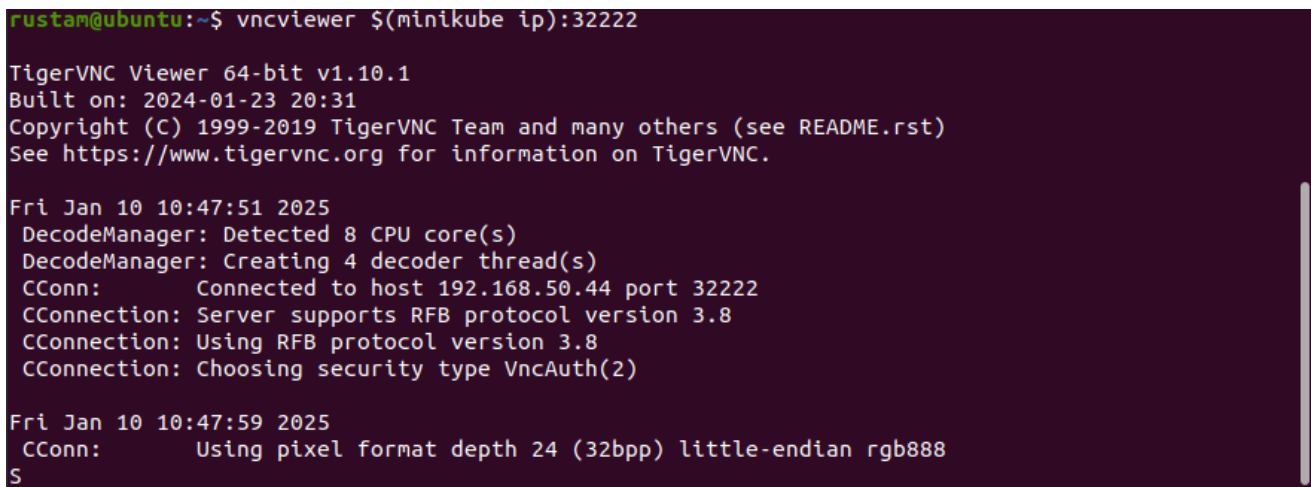
```
kubectl apply -k manifest/
```

Данное приложение будет развёрнуто на порту 5900 внутри кластера, поэтому, чтобы получить к нему доступ извне кластера [5], аналогично необходимо создать для него сервис с типом NodePort [6], который обеспечит к нему доступ через VNC Клиент для игрока. На листинге 4 приведена конфигурация сервиса kubedoom-service для этого случая.

```
apiVersion: v1
kind: Service
metadata:
  name: kubedoom-service
  namespace: kubedoom
spec:
  type: NodePort
  selector:
    app: kubedoom
  ports:
    - protocol: TCP
      port: 5900
      targetPort: 5900
      nodePort: 32222
```

Листинг 4. Конфигурация сервиса для доступа к приложению Kubedoom извне кластера

После установки Kubedoom и создания kubedoom-service станет возможно подключиться к данному приложению через VNC-клиент. На рисунке 6 показано подключение к Kubedoom.



```
rustam@ubuntu:~$ vncviewer $(minikube ip):32222

TigerVNC Viewer 64-bit v1.10.1
Built on: 2024-01-23 20:31
Copyright (C) 1999-2019 TigerVNC Team and many others (see README.rst)
See https://www.tigervnc.org for information on TigerVNC.

Fri Jan 10 10:47:51 2025
DecodeManager: Detected 8 CPU core(s)
DecodeManager: Creating 4 decoder thread(s)
CConn:      Connected to host 192.168.50.44 port 32222
CConnection: Server supports RFB protocol version 3.8
CConnection: Using RFB protocol version 3.8
CConnection: Choosing security type VncAuth(2)

Fri Jan 10 10:47:59 2025
CConn:      Using pixel format depth 24 (32bpp) little-endian rgb888
S
```

Рис. 4 Подключение к Kubedoom

Тестирующее приложение будет выполняться при помощи python-скрипта, осуществляя запросы к серверу по адресу 192.168.50.44:31111 конкурентно в 16 потоках исполнения, измеряя при этом общее время стабильной работы приложения.

В результате выполнения указанных шагов мы получим тестовое окружение, которое можно использовать для тестирования отказоустойчивости в зависимости от числа реплик тестового приложения. Изменять число реплик возможно при помощи команды

```
kubectl scale --replicas= <число_реплик> deployment/web
```

На рисунке 7 показано готовое окружение для теста. В терминале запущен процесс выполнения тестирования доступности приложения, правое окно показывает интерфейс Kubernetes-dashboard, в нижнем левом углу запущен VNC клиент с Kubedoom.

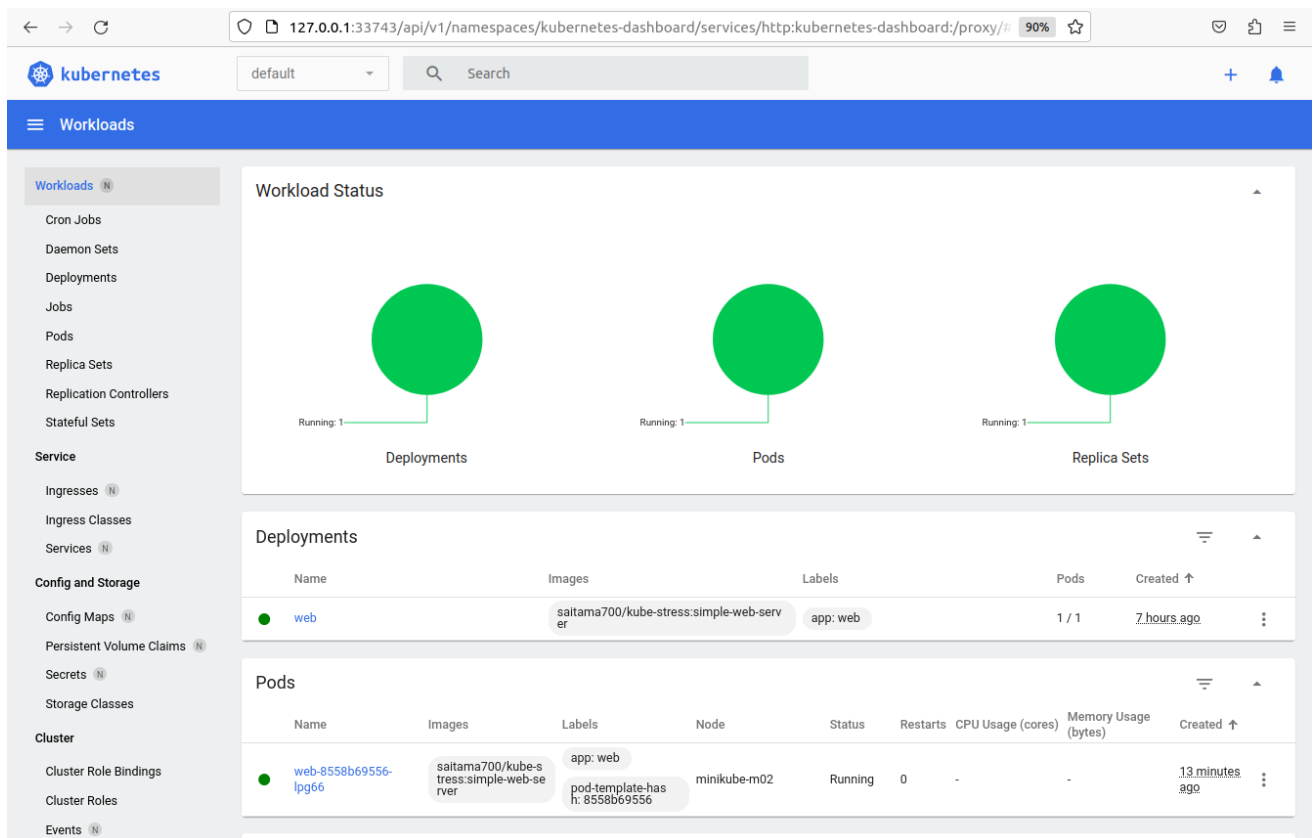


Рис. 5 Тестовое окружение

3. РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТА ДОСТУПНОСТИ ПРИЛОЖЕНИЯ

Для оценки доступности приложения был написан код на языке Python, который представлен на Листинге 5.

```
import asyncio
import aiohttp
import time
import subprocess

total_requests = 0
total_successes = 0
uptime_total = 0

def get_minikube_ip():
    """
    Получение IP-адреса Minikube с помощью команды `minikube ip`.
    """
    try:
        result = subprocess.check_output(["minikube", "ip"], text=True).strip()
        return result
    except subprocess.CalledProcessError as e:
        print(f"Ошибка при получении IP-адреса Minikube: {e}")
        return None

async def make_load_test(host: str, session: aiohttp.ClientSession):
    """
    Function for ddosing host specified with result saving
    :param session:
    :param host:
    :return:
    """

    requests_made = 0
    successes = 0
    uptime = 0

    while True:
        try:
            requests_made += 1
            last_attempt = time.time_ns()

            async with session.get(host) as response:
                await response.text()
                if response.ok:
                    successes += 1
                    uptime = uptime + time.time_ns() - last_attempt

        except asyncio.CancelledError:
            # print("Sync results")
            global total_requests
            global total_successes
            global uptime_total
            total_requests += requests_made
            total_successes += successes
            uptime_total += uptime
```



```

        break
    except Exception as e:
        pass
    # print(f"{e}")

async def main(host: str, threads: int, experiment_time: int):
    """
    Entrypoint for start of the testing
    :return:
    """

    tasks = []

    timeouts = aiohttp.ClientTimeout(
        total=60, connect=15, sock_connect=15, sock_read=15
    )
    async with aiohttp.ClientSession(timeout=timeouts) as session:
        for i in range(threads):
            tasks.append(
                asyncio.create_task(
                    coro=make_load_test(host=host, session=session),
                    name=f"Task_{i}",
                )
            )

        await asyncio.sleep(experiment_time)

        for task in tasks:
            task.cancel()
            try:
                await task
            except asyncio.CancelledError:
                print(f"{task.get_name()} was cancelled")

        # print(f"Availability time {total_successes * 100 / total_requests:.2f} %")
        print(f"{total_requests / experiment_time:.2f} requests/s")
        print(f"{(uptime_total/threads)*100 / (experiment_time*10**9) :.2f} %
uptime")
        # print(total_requests)

# Получение IP Minikube
minikube_ip = get_minikube_ip()
if minikube_ip:
    asyncio.run(
        main(
            host=f"http://{minikube_ip}:31111/",
            threads=20,
            experiment_time=120,
        )
    )
else:
    print("Не удалось получить IP Minikube. Проверьте доступность команды
`minikube ip`.")

```

Листинг 5. Тестирование доступности приложения

Описанный ранее в разделе 2 эксперимент был выполнен последовательно с различным числом требуемых реплик приложения: поочерёдно использовались значения 1, 2, 4, 8, 16. Результаты представлены в таблице.

Таблица. Результаты эксперимента отказоустойчивости

Число реплик приложения, шт	Доступность, %	Средняя частота запросов, запросов/сек
1	62.57	748.22
2	91.12	941.83
4	99.44	956.80
8	99.78	971.51
16	99.83	985.19

Из результатов эксперимента можно установить, что для обеспечения довольно высокой доступности приложения достаточно 4 реплик приложения в условиях симулированной ситуации. В то же время нельзя сказать, что каждое испытание эксперимента было выполнено в одинаковых условиях, как того требует проведение экспериментального исследования: невозможно было обеспечить одинаковую интенсивность отключения подов приложения в каждом из испытаний.

Тем не менее, результаты эксперимента согласуются с ожидаемой доступностью приложения из информации о применении данной технологии в производственной практике.

Видео экспериментов доступно на диске [7].

ЗАКЛЮЧЕНИЕ

В ходе данного исследования были тщательно изучены возможности отказоустойчивости приложений, эксплуатируемых в Kubernetes. Ключевой целью работы было понять, как Kubernetes справляется с различными сбоями и обеспечивает высокую доступность и надежность приложений.

Kubernetes предоставляет мощные инструменты для обеспечения отказоустойчивости, такие как репликация, балансировка нагрузки, автоматическое восстановление и распределение ресурсов. Эти механизмы позволяют приложениям оставаться доступными и функциональными даже в условиях сбоев.

Проведенные эксперименты с изменением количества реплик приложения продемонстрировали, что Kubernetes эффективно справляется с отказами и обеспечивает высокую доступность приложений. Время восстановления после сбоев было минимальным, что подтверждает надежность платформы.

В качестве метрики эффективности отказоустойчивости использовалось время доступности приложения, выраженное в процентах от общего времени работы. Результаты показали, что Kubernetes обеспечивает высокую доступность приложений, даже при значительных нагрузках и сбоях.

В частности, путём выполнения эксперимента с переменным числом реплик приложения удалось убедиться в том, что Kubernetes обеспечивает надёжную платформу для эксплуатации приложений в условиях кластера, которая обеспечивает высокую производительность.

Все файлы конфигурации Kubernetes доступны в репозитории GitHub [8].

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Command line проху. [Электронный ресурс], kubernetes.io — URL: <https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/#command-line-proxy> (дата обращения: 01.01.2025).
2. Kube DOOM. [Электронный ресурс], GitHub — URL: <https://github.com/storax/kubedoom> (дата обращения: 01.01.2025).
3. Minikube Documentation. [Электронный ресурс], minikube.sigs.k8s.io — URL: <https://github.com/kubernetes/dashboard/blob/master/docs/user/access-control/creating-sample-user.md> (дата обращения: 01.01.2025)
4. Assigning Pods to Nodes. [Электронный ресурс], kubernetes.io — URL: <https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#node-affinity> (дата обращения: 01.01.2025).
5. service-access-application-cluster. [Электронный ресурс], kubernetes.io — URL: <https://kubernetes.io/docs/tasks/access-application-cluster/service-access-application-cluster/> (дата обращения: 01.01.2025).
6. Kubernetes. Service. [Электронный ресурс], kubernetes.io — URL: <https://kubernetes.io/docs/concepts/services-networking/service/#type-nodeport> (дата обращения: 01.01.2025).
7. Записи экспериментов [Электронный ресурс], Google Drive — URL: <https://drive.google.com/drive/folders/1VqcuCw3IaPVtw4j6ePRywP895PkENoX1> (дата обращения: 26.01.2025).
8. Репозиторий проекта [Электронный ресурс], GitHub — URL: https://github.com/zina-frid/kube_lab (дата обращения: 26.01.2025).