Министерство науки и высшего образования РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный** **исследовательский политехнический университет»**

Факультет: Прикладной математики и механики

Кафедра: Кафедра вычислительной математики, механики и биомеханики

Направление: 09.04.02 «Информационные технологии и системная инженерия»

Программа: «Информационные технологии и системная инженерия»

**Отчет**

по дисциплинам

**«Программирование с использованием облачных платформ»** и

**«Разработка высоконагруженных web-приложений»**

Выполнил:

ИТСИ-19-1м

студент гр.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Хатямов Р.К.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Ф.И.О.)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

*(подпись)*

Принял:

Ассистент, Истомин Д.А.

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(должность, ФИО руководителя)*

*\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(оценка) (подпись)*

*\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(дата)*

**Пермь 2020**

# Содержание

[Практика 3](#_Toc58776364)

[1. Реализация балансировки и отказоустойчивости с использованием Nginx, Docker 3](#_Toc58776365)

[2. Реализация разделяемого хранилища данных с использованием Redis 4](#_Toc58776366)

[3. Реализация балансировки нагрузки с использованием очередей Kafka 4](#_Toc58776367)

[4. Реализация партиционирования с использованием Postgres 4](#_Toc58776368)

[5. Реализация веб-приложение с использованием протокола S3 5](#_Toc58776369)

[6. Реализация веб-приложения с технологий computer vision 5](#_Toc58776370)

[7. Реализация запуска приложения в Kubernetes 6](#_Toc58776371)

[Теория 7](#_Toc58776372)

[Презентация на тему «Мониторинг и логгирование (ELK Stack, дополнительно Zabbix и др.)» 7](#_Toc58776373)

[Презентация на тему «AWS Game tech» 12](#_Toc58776374)

# Практика

## Реализация балансировки и отказоустойчивости с использованием Nginx, Docker

Для реализации балансировки и отказоустойчивости с использованием Nginx требовалось:

* Разработать приложение, содержащее HTTP endpoint, при обращении к которому возвращается ответ вида {“counter”: “1”}. При каждом обращении счетчик должен увеличиваться.
* Запустить несколько экземпляров данного приложения на разных портах (возможно использование Docker, но не обязательно)
* Запустить Nginx, который балансирует нагрузку между запущенными веб-сервисами
* Реализовать и проанализировать различные алгоритмы балансировки: round robin, hash, least conn, least time, random, и т.д.
* Реализовать и проанализировать функционал реализации отказоустойчивости (fail\_timeout, max\_fails) при отключении одного или нескольких веб-сервисов
* Проанализировать результаты нагрузочного тестирования с использованием Apache benchmark и/или Apache JMeter.

Во всех облачных платформах имеется возможность запуска приложений с использованием Kubernetes. Рассмотрение Docker является предварительным шагом перед запуском приложения в Kubernetes.

Для реализации использования Docker необходимо было:

* Пройти interactive tutorial по Docker
* Создать веб-сервис
* Создать Dockerfile и запустить приложение в Docker
* Создать docker-compose.yaml и запустить несколько контейнеров с использованием docker-compose

## Реализация разделяемого хранилища данных с использованием Redis

Применение данного хранилища рассматривается для реализации хранения разделяемого состояния между экземплярами веб-сервисов. Также, рассматривается концепция no sql хранилища данных.

Для реализации данной лабораторной работы требовалось:

* Пройти интерактивный redis tutorial
* Запустить redis
* Доработать приложение из предыдущей лабораторной работы таким образом, чтобы счетчик входящих запросов хранился в redis.

## Реализация балансировки нагрузки с использованием очередей Kafka

Основная цель – рассмотреть применимость очередей для ограничения максимального количества входящих запросов в единицу времени.

Для реализации данной лабораторной работы требовалось:

* Запустить kafka
* Реализовать веб-сервис, который с неравномерной скоростью записывает сообщения в очередь. Например, загружает данные из файла и записывает их в очередь.
* Реализовать веб-сервис, который с определенной задержкой обрабатывает данные в очереди.

## Реализация партиционирования с использованием Postgres

Для реализации данной лабораторной работы требовалось:

* Необходимо запустить СУБД с поддержкой партиционирования (например, postgres)
* Создать таблицу с использованием партиционирования

Написать запросы для:

* вставки данных;
* добавления и удаления партиций;
* создания глобальных и локальных индексов;
* выборки данных и использованием индексов.

## Реализация веб-приложение с использованием протокола S3

Протокол S3 поддерживается множеством облачных провайдеров. Протокол рассматривается для реализации простого key-value хранилища данных. Можно использовать, например, Java SDK от AWS. Либо реализовать обращение с использованием HTTP запросов.

 Для реализации данной лабораторной работы требовалось:

* Создать ресурс, поддерживающий протокол S3 у любого облачного провайдера
* Реализовать веб-приложение, которое позволяет:
  + Работать с bucket: создавать, удалять, просматривать список
  + Работать с данными в bucker: создавать, удалять, просматривать данные в bucket

## Реализация веб-приложения с технологий computer vision

В данной работе рассматривается возможность использования готовых сервисов, реализующий функционал computer vision.

Для реализации данной лабораторной работы требовалось создать веб-приложение, позволяющее:

* Загрузить изображение
* Отобразить информацию о находящихся объектах на изображении

## Реализация запуска приложения в Kubernetes

Необходимо реализовать запуск приложение в Kubernetes с использованием облачного провайдера. В исключительных случаях можно запустить Kubernetes локально.

Для реализации данной лабораторной работы требуется:

* Пройти interactive tutorial по Kubernetes/minikube
* Создать yaml файлы для работы с Kubernetes
* Управлять развёртыванием контейнером с использованием kubectl

# Теория

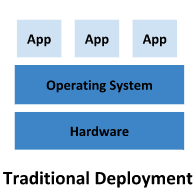
## Презентация на тему «Kubernetes»

Kubernetes – это портативная расширяемая платформа с открытым исходным кодом для управления контейнеризованными рабочими нагрузками и сервисами, которая облегчает как декларативную настройку, так и автоматизацию.

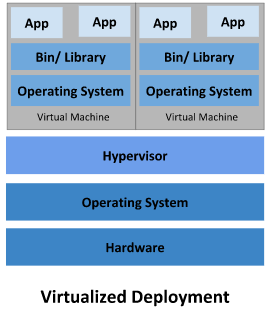
В 2014 году Google открыл исходный код Kubernetes.

Kubernetes имеет несколько этапов развития:

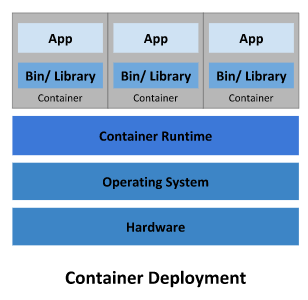
**Традиционная эра развертывания.** Ранее организации запускали приложения на физических серверах. Не было способа определить границы ресурсов для приложений. Это вызывало проблемы с распределением ресурсов.



**Эра виртуального развития.** Виртуализация позволила запускать несколько виртуальных машин на одном физическом сервере. Виртуализация изолирует приложения между виртуальными машинами и обеспечивает определенный уровень безопасности.



**Эра контейнеров.** Контейнеры похожи на виртуальные машины**,** но у них есть свойства изоляции для совместного использования операционной системы (ОС) между приложениями. Поэтому контейнеры считаются легкими.



Преимущества Kubernetes:

* Гибкое создание и развертывание приложений: простота и эффективность создания образа контейнера по сравнению с использованием образа виртуальной машины.
* Непрерывная разработка, интеграция и развертывание: обеспечивает надежную и частую сборку и развертывание образа контейнера с быстрым и простым откатом (благодаря неизменности образа).
* Изоляция ресурсов: предсказуемая производительность приложения.
* Грамотное использование ресурсов: высокая эффективность и компактность.

Kubernetes дает вам фреймворк для гибкой работы распределенных систем. Он занимается масштабированием и обработкой ошибок в приложении, предоставляет шаблоны развертывания и многое другое.

## Презентация на тему «Kubernetes»

***Yandex IoT Core*** – это облачный отказоустойчивый MQQT-брокер, позволяющий наладить двустороннюю защищенную связь между устройствами и локальными или облачными ресурсами.

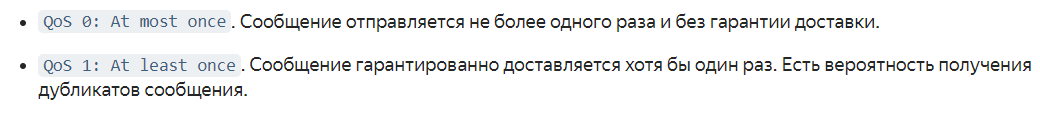
**Основные элементы сервиса** – устройство и реестр, обменивающиеся данными с помощью MQQT-брокера.

Основные элементы сервиса:

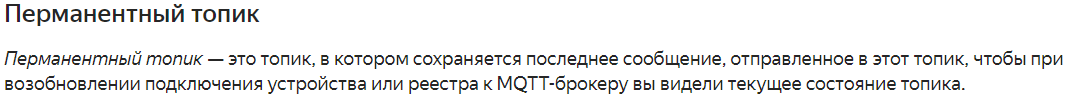
* **Устройство** – экземпляр физического устройства, представленный в сервисе в виде сертификата и имени устройства (например, датчик температуры)
* **Реестр** – набор устройств, логически связанных между собой. Реестр может читать данные с устройств и отправлять им различные команды.
* **MQTT-брокер** – центральный узел для обмена сообщениями между устройствами и реестрами. Обмен данными происходит только через брокер.
* **Топик** – канал получения и отправки сообщений между устройствами и реестрами.

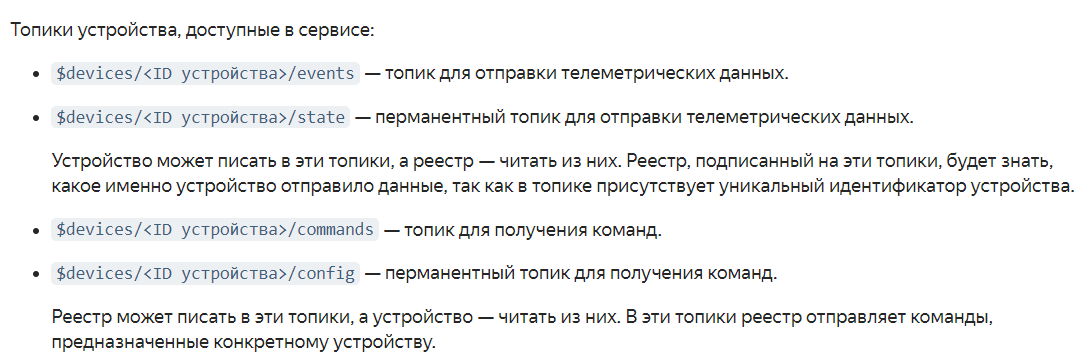
MQTT-брокер отвечает за получение и обработку сообщений, а также контролирует их доставку MQTT-клиентам (устройствам или реестрам), при этом обеспечивая необходимый уровень качества обслуживания (QoS).

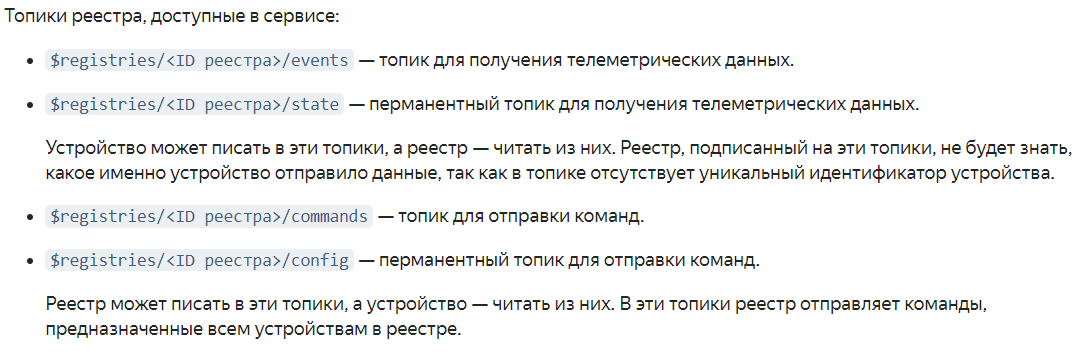
Сервис от Яндекс поддерживаем следующие уровни QoS:



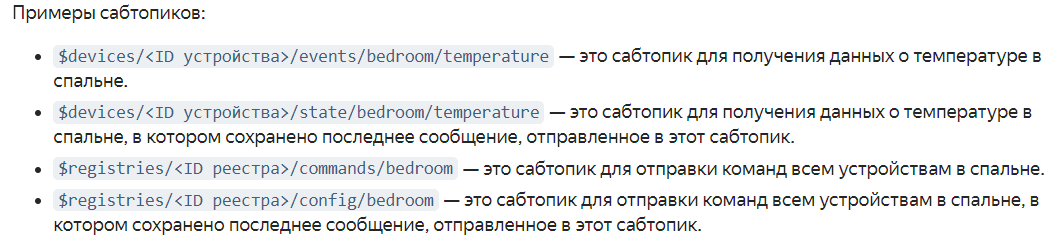
В сервисе есть топики устройств и топики реестра. Топиками устройства подписываются сообщения, которые предназначены для определенного устройства, а топиками реестра — сообщения, которые предназначены для всех устройств этого реестра.







Сабтопик – пользовательский топик, созданный в рамках существующих в сервисе топиков устройств и реестров.

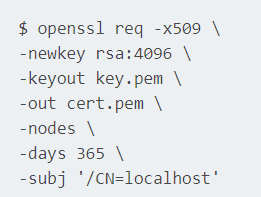


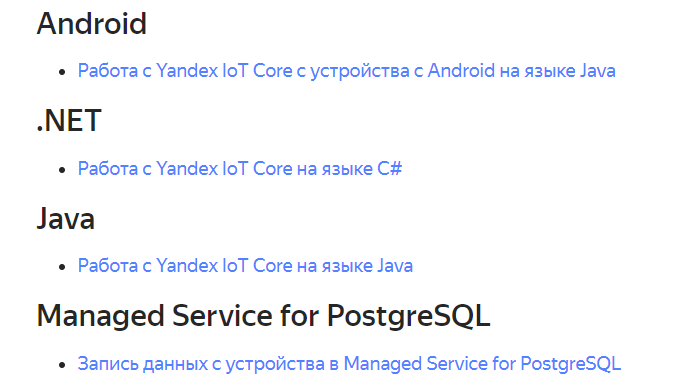
Начало работы:

Для начала работы с сервисом IoT от Яндекс необходимо:

* Создать X.509-сертификаты;
* Создать реестр;
* Создать устройство;
* Настроить процесс обмена сообщениями между ними.

Устройство и реестр взаимодействуют с помощью X.509 сертификата. Сертификат можно создать, например, при помощи программы OpenSSL. Он имеет следующий вид:





В сервисе Яндекс IoT действуют следующие ограничения:

* Квоты – организационные ограничения, которые можно изменить по запросу в техническую поддержку.
* Лимиты – технические ограничения, обусловленные особенностями архитектуры Yandex.Cloud. Изменение лимитов невозможно.

