МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

СӘТБАЕВ УНИВЕРСИТЕТІ

Институт кибернетики и информационной технологии
Кафедра “Программной инженерии”

Абдыбаев Азамат Ерланулы

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

На соискание академической степени магистра

Название диссертации	Архитектура и оркестрация микросервисов в системе оптимизации транспортной логистики
Направление подготовки	6M070400 – Вычислительная техника и программное обеспечение

Научный руководитель
PhD технический директор
ТОО “Вериграм”
_____Кеншимов Ч.
«__» _____ 2020 г.

Оппонент
канд. тех. наук, профессор
_____ Богданчиков А.
«__» _____ 2020 г.

Нормоконтроль
лектор
_____
Ж.М.Алибиева
«__» _____ 2020 г.

ДОПУЩЕН К ЗАЩИТЕ
Заведующий кафедрой ПИ
Доктор PhD,
_____ Тұрдалыұлы М.
«__»_____ 2020 г.

Алматы 2020

Институт кибернетики и информационных технологий

Кафедра Программная инженерия

Специальность: 6М070400 – Вычислительная техника и программное обеспечение

УТВЕРЖДАЮ
Заведующий кафедрой ПИ
Доктор PhD,
_____Тұрдалыұлы М.
«___» _____ 2020 г.

ЗАДАНИЕ
на выполнение магистерской диссертации
магистранту Абдыбаеву Азамату

Тема диссертации: “Архитектура и оркестрация микросервисов в системе оптимизации транспортной логистики”

Срок сдачи законченной диссертации «___» _____

ГРАФИК
подготовки магистерской диссертации

Наименование разделов, перечень разрабатываемых вопросов	Сроки представления научному руководителю	Примечание
Раздел 1. Подход		
Раздел 2. Основные эксперименты		

Консультации по проекту с указанием относящихся к ним разделов проекта

Раздел	Консультант, (уч. степень, звание)	Сроки	Подпись
Нормоконтроль	Ж.М.Алибиева, Лектор кафедры программная инженерия		

Дата выдачи задания "____"_____ 2020 г.

Заведующий кафедрой _____Тұрдалыұлы М.

Научный руководитель _____ Кеншимов Ч.

Задание принял к исполнению магистрант _____ Абдыбаев А.

Дата "_____"_____ 2020 г.

АННОТАЦИЯ

В данной работе рассматривается основные модели сетевого доступа, а также приводится пример реализации контейнерных технологий на предприятии с использование системы оркестраций контейнеров. Помимо этого рассматривается варианты реализации поставки продукта и использование полного цикла разработки программного обеспечения, включающий такие этапы как сборка, тестирование и релиз. Также в данной диссертационной работе рассматривается интеграция мониторинга и логирования системы в существующую инфраструктуру продукта.

АҢДАТПА

Бұл жұмыста біз желіге қол жеткізудің негізгі модельдерін қарастырамыз, сонымен қатар контейнерді құру жүйесін қолданатын кәсіпорында контейнерлік технологияларды енгізу мысалын қарастырамыз. Сонымен қатар, өнімді жеткізуді жүзеге асыру және құрастыру, тестілеу және шығару сияқты кезеңдерді қамтитын бағдарламалық жасақтаманың толық циклын қолдану нұсқалары қарастырылады. Сондай-ақ, бұл диссертацияда өнімнің қолданыстағы инфрақұрылымына жүйелік тіркеу мен бақылау интеграциясы жатыр.

SUMMARY

In this paper, we consider the main models of network access, as well as an example of the implementation of container technologies in an enterprise using a container orchestration system. In addition, options for implementing the delivery of the product and the use of a full software development cycle, including such stages as assembly, testing and release, are considered. Also in this dissertation is the integration of monitoring and system logging into the existing product infrastructure.

СОДЕРЖАНИЕ

Введение

Облачные вычисления это модель сетевого доступа, которая достигается с помощью виртуализации. В данную модель входят такие понятия как: администрирование ресурсов на уровне железа и программ, обработка запросов, построение бизнес логики. На данные момент времени облачные технологии популярны в использовании. Главные преимущества это относительно недорогие цены за услуги хостинга, широкая доступность, ряд готовых инструментов в создания инфраструктуры, обеспечения безопасности, отказоустойчивости и масштабируемости.
В целом, облачные технологии подразделяются на несколько подвидов по типам доступа:

- Публичное облако
- Приватное облако
- Общедоступное облако
- Гибридное облако

Публичное облако - самый распространенный вид облако, где есть готовые и необходимые инструменты для настройки инфраструктуры. Данные вид является превалирующим и самым используемым на рынке. Примерами служат такие популярные сервисы как: Digital Ocean, Google Cloud Platform, Microsoft Azure, IBM, Amazon Web Services. Почти все из них имеют бесплатный кредитный баланс для того, чтобы пользователи могли опробовать сервис в качестве инфраструктуры.
Приватное облако - аналогичная технология вышеописанной, за исключением того, что ограниченные круг пользователей может иметь доступ в данное облако. Зачастую частные компании разворачивают все необходимые сервисы на голом железе которое доступно с определенных IP адресов, тем самым достигается безопасность и производительность.
Общедоступное облако - инфраструктура, которая контролируется группой организаций с одинаковыми потребностями, политикой безопасности и уровнем доступа
Гибридное облако - инфраструктура, которая является смесью вышеописанных типов. Необходимость данного типа является результатом сложности той или иной инфраструктурой и списком используемых технологий.
Контейнер - стандартизированная единица программного обеспечения, которая содержит в себе весь код программы и необходимые зависимости,

для его быстрого запуска вне зависимости от типа операционной системы. Контейнер очень легковесен в программном понятии и изолирован от окружающей среды. Изображение контейнера это предопределенная файловая система, которая при запуске превращается в контейнер. Если проводить аналогию в ООП, то контейнер это экземпляр класса, а изображение контейнера - класс. Вышеописанные характеристики делают контейнер главным инструментом, которым можно управлять, разграничивать ресурсы, настраивать сетевой трафик внутри облака.

Изолированность ресурсов(таких как оперативная память, жесткие диски, работа процессов) достигается с помощью **виртуализации**. Непосредственно виртуализация работают при помощи программного обеспечения, которая разделяет ресурсы компьютера на уровне железа. С помощью таких программ как: VirtualBox, Parallels, которых можно запустить виртуальные операционные системы на основной операционной системе. Докер делает это в автоматическом режиме. То есть по своей сути контейнер - обрезанная виртуальная машина которая может полноценно работать. Для управления контейнером используется программа оркестрации контейнеров. **Docker** - самое популярное решения для контейнеризации приложения с возможность автоматизационного развертывания. Для управления контейнерами необходимо иметь систему оркестрации контейнеров. На данные момент есть много различных программ для оркестраций контейнеров. Среди них выделяют:

- Docker Swarm - нативное открытое программное обеспечение, которое поддерживается компанией Docker
- Kubernetes - программное обеспечения разработанное Google и презентованное в 2014.

Kubernetes обладает рядом преимуществ надо Docker Swarm. В данные преимущества входят как более абстрактный подход к созданию и управлению инфраструктурой, так и большое количество открытых инструментов для обеспечения масштабируемости, высокой доступности, CI/CD.

Преимущества виртуализации

Повышение изоляции

Ограничение одной или группы тесно связанных служб собственной виртуальной машиной;

Снижение вероятности сбоев от взаимного влияния программ;

Безопасность

Распределение задач администрирования — возможность ограничить права каждого администратора только самыми необходимыми;

Снижение потенциальных вредных последствий взлома какой-либо из служб.

Распределение ресурсов — каждая машина получает столько ресурсов, сколько ей необходимо, но не более того.

Приоритезация задач:

Выделение памяти по требованию;

Гибкое распределение сетевого трафика между машинами;

Распределение дисковых ресурсов;

Постоянная доступность

Есть возможность live-миграции машин;

Плавный апгрейд критических серверов.

Повышение качества администрирования

Возможность выполнения регрессионных тестов;

Возможность экспериментирования и исследования.

Постановка задач

На момент прихода в компанию Релог, инфраструктура состояла из большого количества отдельных серверов, которые “хостились” на Digital Ocean, FirstByte, Azure.

Это обусловило сложностями, связанными с хостинг провайдером. В этот список входят такие пункты как:

- Хаотичный разброс приложений на разных провайдеров хостинга
- Не настроенный и децентрализованный SSH доступ
- Разные версии операционных система
- Беспорядочный, а иногда отсутствующий CI/CD
- Опыт работы с FirstByte показывает, что ресурсы распределялись не всегда честно, а порой возникали проблемы с сетью
- Большие затраты на IP адреса и сервера
- Постоянное администрирование 20+ серверов на предмет выявления проблем
- Отсутствие мониторинга нагрузки серверов по отдельности и в целом
- Отсутствие единого места, куда бы выгружались логи серверов

Со стороны инфраструктуры проекта, возникали следующие трудности:

- Отсутствие CI/CD
- Отсутствие горизонтального и вертикального масштабирования сервисов и монолита
- Откат релизов по необходимости
- Резервное копирование происходило с большой задержкой по времени
- Неавтоматизированная чистка старых резервных копий
- Плавающая производительность, что выливалось в качество исполнения алгоритма в временной характеристике
- Децентрализованная инфраструктура, из за которой приходилось очень часто, перед началом выполнения той или иной задачи разбираться с доступом на сервере.
- Некорректно работающее тестовое окружение, которое приходилось “накатывать”, поверх существующих зависимостей
- Отсутствие полноценного тестового окружения, которое бы покрывало все микросервисы
- Разделение ресурсов между разными сервисами

- Отсутствие резервного копирования базы данных некоторых сервисов
- Отсутствие отказоустойчивости

Были рассмотрены следующие статьи:

1) Docker Swarm and Kubernetes in Cloud Computing Environment
Arsh Modak1 , Prof. S.D.Chaudhary2 , Prof. P.S.Paygude3 , Prof. S.R.Idate4
1,2,3,4 Dept. of Information Technology, BVUCOEP, India

2) Techniques to Secure Data on Cloud: Docker Swarm or Kubernetes
Proceedings of the Third International Conference on Trends in Electronics and Informatics (ICOEI 2019) IEEE Xplore Part Number: CFP19J32-ART; ISBN: 978-1-5386-9439-8

Поставленные цели

- Мигрировать приложения в контейнерные технологии
- Интегрировать логирование системы
- Сделать единую точку входа в инфрастуктуру
- Настроить полноценный CI/CD
- Использовать масштабирование посредством контейнерных технологий
- Установить не более 2-3 физических серверов, где бы располагалась вся инфраструктура
- Подключить мониторинг системы на уровне железа и на уровне контейнеров
- Реализовать данные задачи в системе оркестарции контейнеров

Характеристики Kubernetes

- Приложение устанавливаются как комбинация pods, services, deployments. Что дает гибкую настройку.
- Единая модель управлением сети. Настройка открытости и закрытости того или иного сервиса по отдельности. Легкая конфигурация
- Автоматическое масштабирование посредством декларации yaml файла. Возможно настроить масштабирование по необходимости(при загрузке серверов)
- Умная отказоустойчивость достигается посредством концепции master - slave. При отказе какого-либо узла, происходит автоматическое развертывание приложений на других.
- Возможность настроить many-masters архитектуры. Что приведете к гарантированной отказоустойчивости.
- Конфигурируемая сущность Ingress из под коробки, благодаря которой можно настроить балансировку
- Интеграция любой системы логирования или мониторинга

Характеристики Docker Swarm

- Приложение устанавливаются как микросервисы.
- Приложения взаимодействуют посредством модели «bride». Для того, чтобы связать 2 сервиса необходимо ручная настройка посредством docker-compose
- Ручное масштабирование. Посредством cli.
- Отсутствие отказоустойчивости из под коробки. Необходимость ручной настройки отказоустойчивость.
- Балансировки нет, необходимо подключить сторонние решения.
- Ручной мониторинг и легирование только при помощи Reimann

Согласно вышеописанным статьям kubernetes дает следующие преимущества:

- Горизонтальное и вертикальное масштабирование
- Изолированность и параметризация ресурсов, за счет Docker образов
- Более высокий уровень абстракции между инфраструктуры
- Полноценный CI/CD
- Выпуск “релизов”
- Zero-Downtime
- Blue-Green deployment
- Canary deployment
- Изолированность ресурсов на абстрактном уровне
- Role Based Access в кластер
- Централизованная инфраструктура
- Сетевая модель взаимодействия микросервисов
- Репликация экземпляров сервиса из под коробки Kuberenetes
- Относительно быстрая настройка тестового окружения
- Качественная обработка Unit tests
- Сбор мониторинг и метрики сервисов
- Абстракция инфраструктуры
- Отказоустойчивость

Поэтому было решено мигрировать монолит, микросервисы, базы данных в Kubernetes кластер.

Перед миграцией сервисов необходимо было решить следующие проблемы.

Настроить Dockerfile для каждого из сервисов

Для данного этапа, необходимо было ответить на вопрос: Что представляет из себя сервис?
Бинарный файл или интерпретируемая программа.
Собрать кодовую в базу в одном месте и настроить команду для запуска docker контейнера
В случае интерпретируемой программы Dockerfile выглядит примерно следующим образом:

```
FROM git-registry.relog.kz/docker-images/node-alpine

RUN mkdir /app
WORKDIR /app

COPY . .

RUN npm run build
CMD [ "npm","run", "prod" ]
```

В случае бинарного файла:

```
FROM maven:3.6.0-jdk-10

WORKDIR /src/app

COPY . /src/app

CMD java -Xms8198m -Xmx8198m -jar -Dorg.slf4j.simpleLogger.defaultLogLevel=debug target/PlanningWorker-*.jar
```

Настройка для сборки и тестирования

На данном этапе используется GitLab CI. Для этого необходимо создать Runner(программа которая умеет выполнять bash скрипты) и настроить тэги для правильной работы. Далее необходимо написать отдельные stage-ы

Сборка:

```yaml
image: git-registry.relog.kz/docker-images/node-alpine

variables:
  KUBECONFIG: /etc/deploy/config

stages:
  - install
  - tests
  - build
  - deploy

before_script:
  - export TAG="1.1.0-$CI_PIPELINE_ID-$(echo $CI_BUILD_REF | cut -c1-7)"
  - export FULL_VERSION="1.1.0-$CI_PIPELINE_ID-$(echo $CI_BUILD_REF | cut -c1-7)"

npm:
  stage: install
  only:
    - dev
    - master
  tags:
    - npm

  cache:
    key: ${CI_COMMIT_REF_SLUG}
    paths:
    - node_modules

  artifacts:
    expire_in: 1 day
    paths:
      - node_modules

  script:
    - npm install
```

Тестирование:

```
tests:
  stage: tests
  only:
    - dev
    - master
  services:
    - name: bitnami/mongodb:4.0.8
      alias: mongodb
  variables:
    MONGODB_ROOT_PASSWORD: hiddenvalue
    MONGODB_USERNAME: hiddenvalue
    MONGODB_DATABASE: hiddenvalue
    MONGODB_PASSWORD: hiddenvalue
    NODE_ENV: test
    PORT: "3003"
    MONGO_URL: "mongodb://hiddenvalue:hiddenvalue@mongodb:27017/hiddenvalue"

  script:
    - npm t
```

Запушить в container registry готовый к релизу образ:

```
docker-image:
  stage: build
  image: docker:latest
  only:
    - dev
    - master
  tags:
    - docker
  script:
    - docker login -u gitlab-ci-token -p $CI_BUILD_TOKEN $CI_REGISTRY
    - docker build -t $CI_PROJECT_PATH .
    - docker tag $CI_PROJECT_PATH $CI_REGISTRY_IMAGE
    - docker tag $CI_PROJECT_PATH $CI_REGISTRY_IMAGE:$TAG
    - docker push $CI_REGISTRY_IMAGE
    - docker push $CI_REGISTRY_IMAGE:$TAG
    - docker rmi $CI_PROJECT_PATH
    - docker rmi $CI_REGISTRY_IMAGE
    - docker rmi $CI_REGISTRY_IMAGE:$TAG
```

Постоянное хранилище данных

В связи с тем, что контейнеры не сохраняют данные между релизами, необходимо было решить проблему хранения данных.

То есть необходимо должна ли та или иная информация сохраняться между релизами или создаваться каждый раз при релизе. К примеру фотографии пользователей необходимо сохранять, тогда как nginx кэш созданный во время работы необходимо удалять между релизами. Для этого необходимо использовать логический типа файловой системы Volume. Volume представляет из себя файловую систему, которую можно создать, указать уровень доступ для данной файловой системы и подключить в контейнеру при старте.

Пример подключения volume

```
- name: hidden-value
    image: hidden-value
    replicas: 1
    rollingUpdate:
     maxSurge: 1
     maxUnavailable: 0
    env:
    - name: PORT
      value: "4000"
    - name: HOST
      value: "https://hidden-value"
    - name: PATH_TO_FILE
      value: /var/www/
    - name: TOKEN
      value: "hidden-value"
    ports:
    - name: http
      port: 4000
    resources:
      limits:
        cpu: 0.5
        memory: "0.5Gi"
      requests:
        cpu: 0.3
        memory: "0.2Gi"
    volumeMounts:
    - mountPath: /var/www
      name: media
    volumes:
    - name: media
      persistentVolumeClaim:
        claimName: relog-media-files
    ingress:
    - host: hidden-value
      tls:
        secretName: relog-tls
```

Репозиторий

В ходе работы были переведены все микросервисы и монолит в kubernetes с работающим CI/CD. В этот список состоит из:

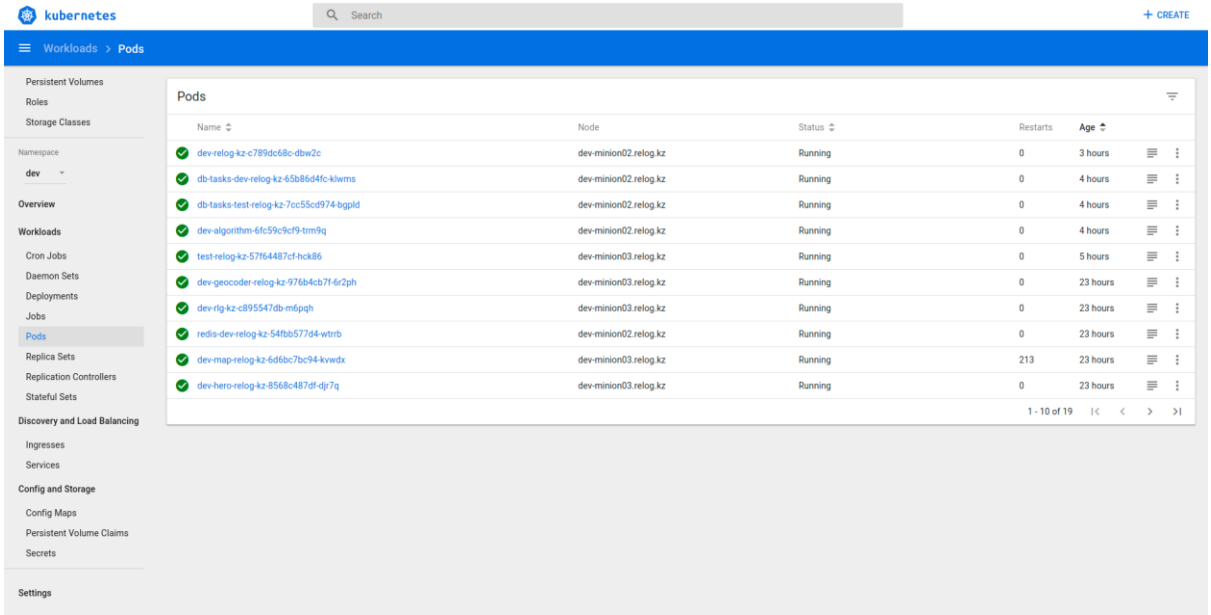
- Группа Applications - здесь находится главное облачное приложение и коробки, в их числе:
  - Основной монолит
  - Коробка #4
  - Коробка #3
  - Коробка #2
  - Коробка #1
- Группа Microservices
  - Media-files - микросервис, который хранит и принимает файлы(картинки), для дальнейшей раздачи статического контента
  - Db Tasks - микросервис, в котором можно выполнять операции по обновлению коллекций в базе данных. Также работает **cron task**, для выявления неверных координат.
  - Gazer - микросервис мониторинга клиента.
  - Geocoder - микросервис геокодирования(прямого и обратного). Работает с такими сервисами как: yandex, here, 2gis.
  - GraphHopper Router - роутер для построения маршрута из точки А в точку Б. Строит маршрут для коробок.
  - Google Algorithm - недоработанный алгоритм базирующийся на **Google OR Tools**
  - Algo Tester - микросервис тестирования алгоритма, также можно посмотреть результаты того или иного планирования в разделе "Планирования"
  - Epic Server - микросервис для взаимодействия с сторонними сервисами.
  - Bitly Server - микросервис создания коротких ссылок
  - Relog Admin - админ панель. В целом здесь создается и настраивается компания, также можно создавать единоразовые и периодические оплаты.
  - Time Zone Master - получение сдвига часовых поясов по координатам(longitude, lattitude)
  - Jobs Server - микросервис, работающий с excel документами клиентов.
  - OSRM Router - Микросервис для построение мрашрута из точки А в точку Б, а также для вычислений матрицы попарных расстояний для алгоритма(используется главным алгоритмом и облачным приложением)

○ Path filter - Микросервис, принимает массив позиций на карте и исходя из этих данным выстраивает более приемлемый маршрут(работает с историей передвижений водителя)

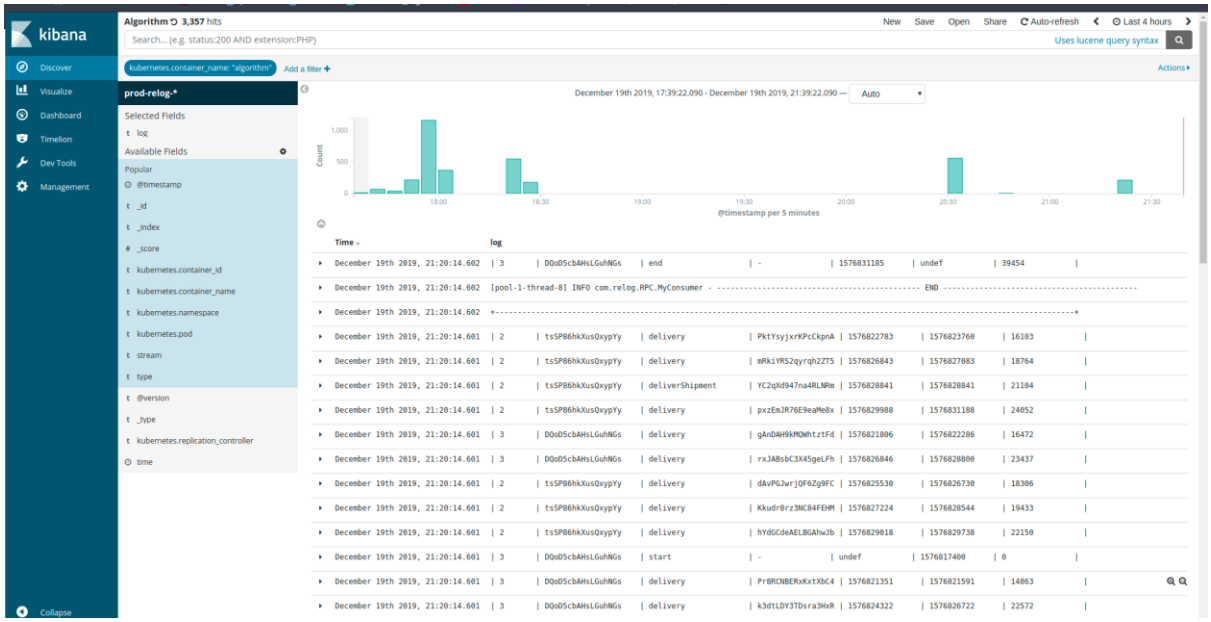
○ Algorithm - Микросервис, в котором выполняется планирование заявок(распределение заявок и водителей)

- Группа Docker-image - прекомпилированные docker образы
- Группа Deployment - с единственным репозиторием для релиза.

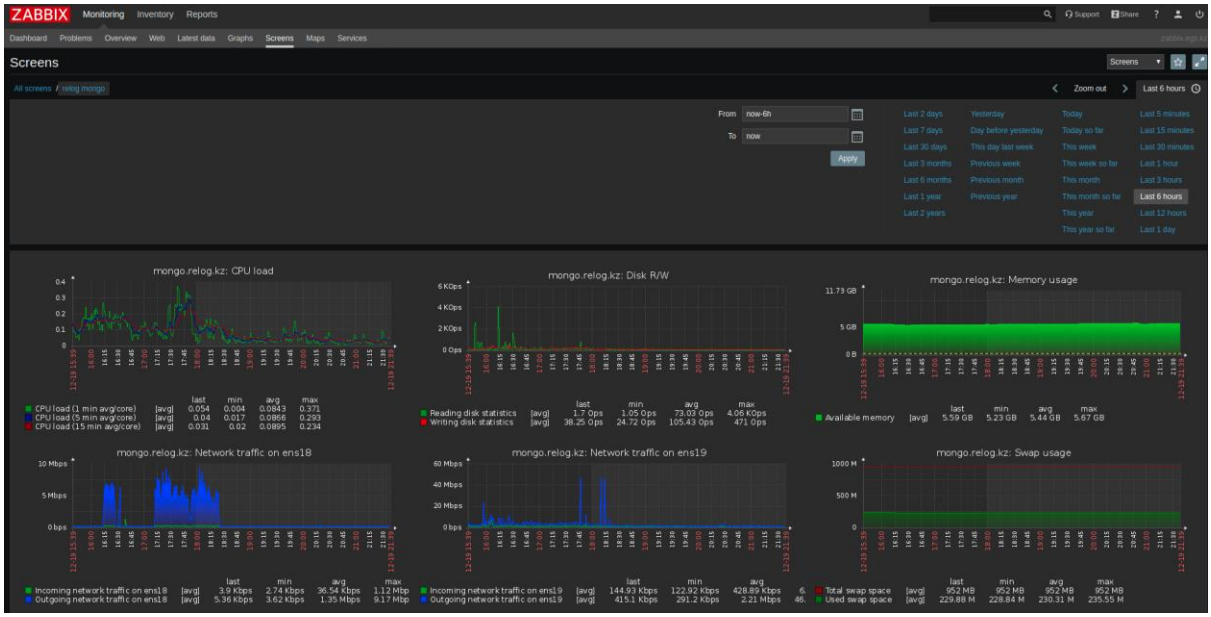
Также помимо этого было настроено тестовое окружение, дублирующее боевое окружение в kubernetes кластере



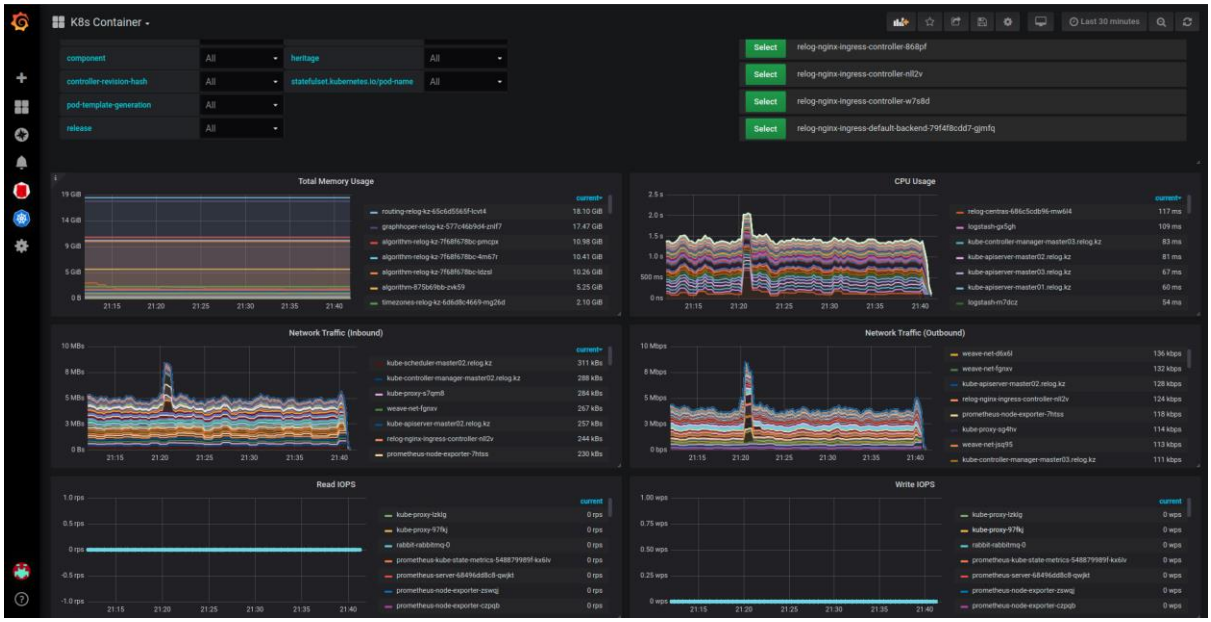
Настроена kibana для централизованного сбора логов всех сервисов



Настроен Zabbix для сбора метрики ресурсов на уровне железа:



Настроена Grafana для сбора метрики ресурсов ну уровне контейнера:



Стэк используемых технологий и фреймворков:

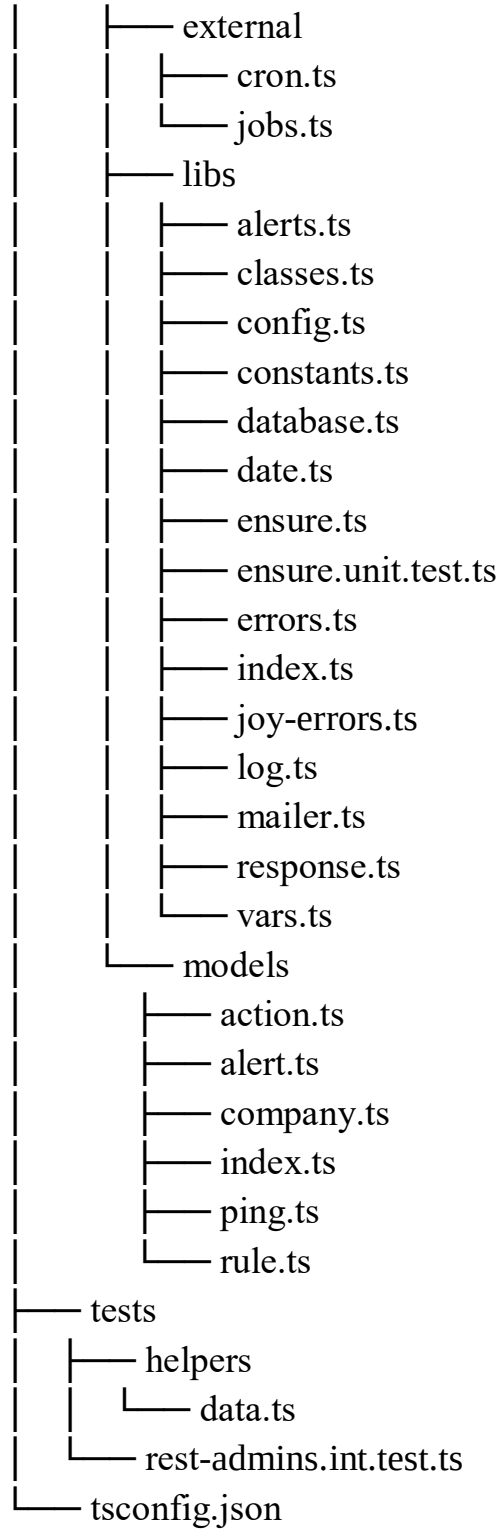
- Go
- Java
- Javascript
- React
- Redux
- Meteor
- NodeJS
- Redis
- Webpack
- Python
- Gitlab
- Git
- Kubernetes
- Docker
- Helm
- Nginx
- Ansible
- Proxmox
- Grafana
- Elastic search, Kibana Logstash
- MongoDb
- Operations Log
- RabbitMQ

- Ansible
- Zero downtime deployment
- High Availability Cluster
- Prometheus
- HAProxy
- Prometheus

Микросервис Gazer

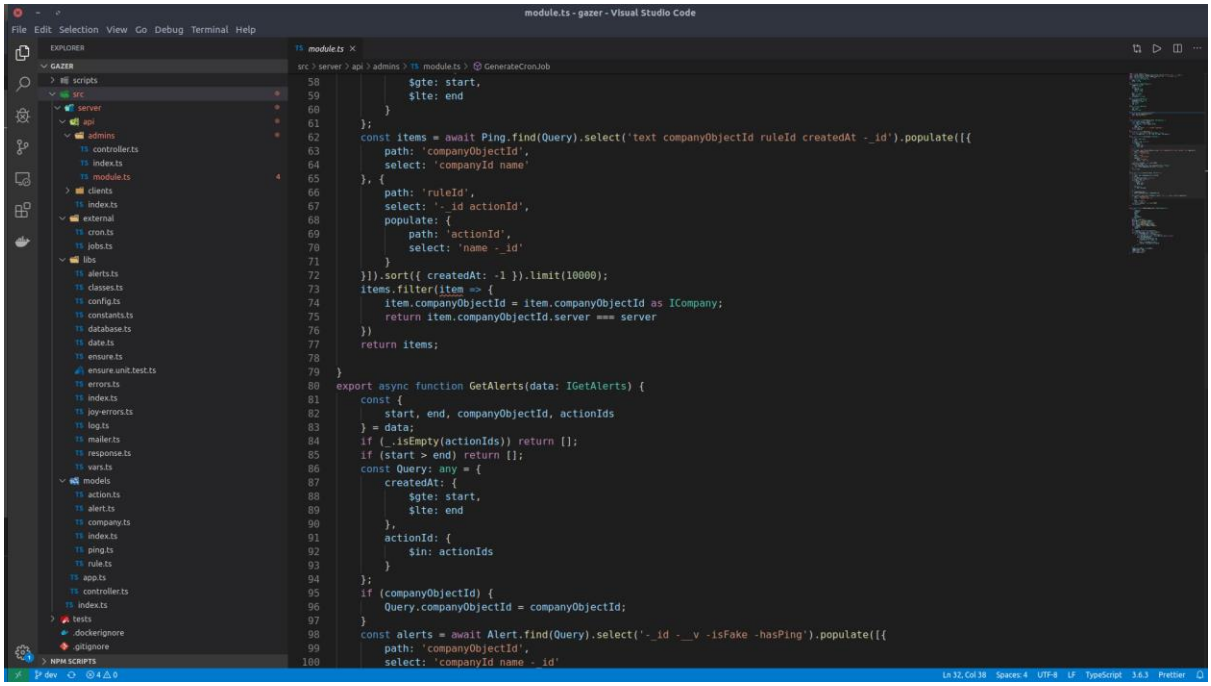
Также в ходе работы в Relog были разработаны различные микросервисы. Один из них - Gazer. Мониторинг активности клиентов. Результат команды tree в корне проекта:

```
cifer@pc:repos/gazer ‹dev›$ tree
.
├── Dockerfile
├── jest.config.js
├── nodemon.json
├── package.json
├── README.md
├── scripts
│   ├── cifer.sh
│   └── test.sh
├── src
│   ├── index.ts
│   └── server
│       ├── api
│       │   ├── admins
│       │   │   ├── controller.ts
│       │   │   ├── index.ts
│       │   │   └── module.ts
│       │   ├── clients
│       │   │   ├── controller.ts
│       │   │   ├── index.ts
│       │   │   └── module.ts
│       │   └── index.ts
│       ├── app.ts
│       ├── controller.ts
```



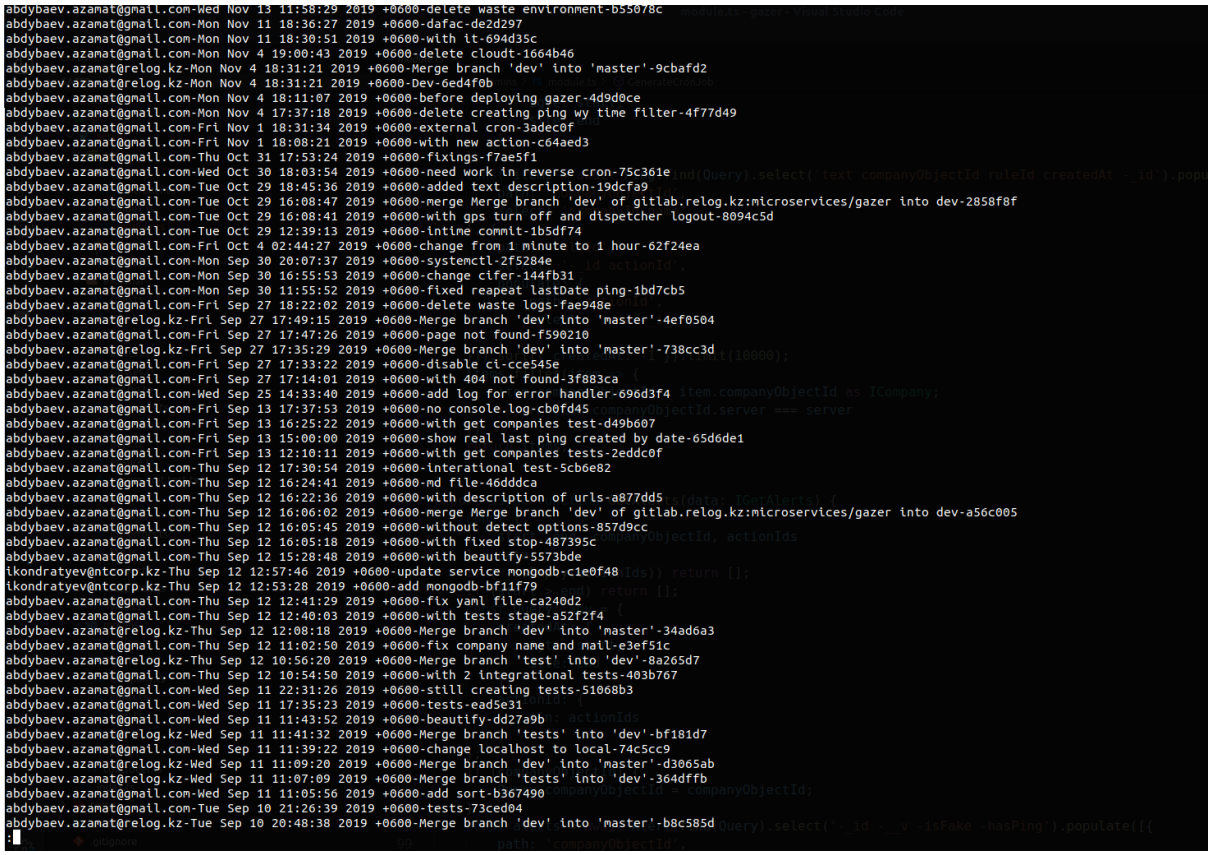
11 directories, 43 files

Пример кода:



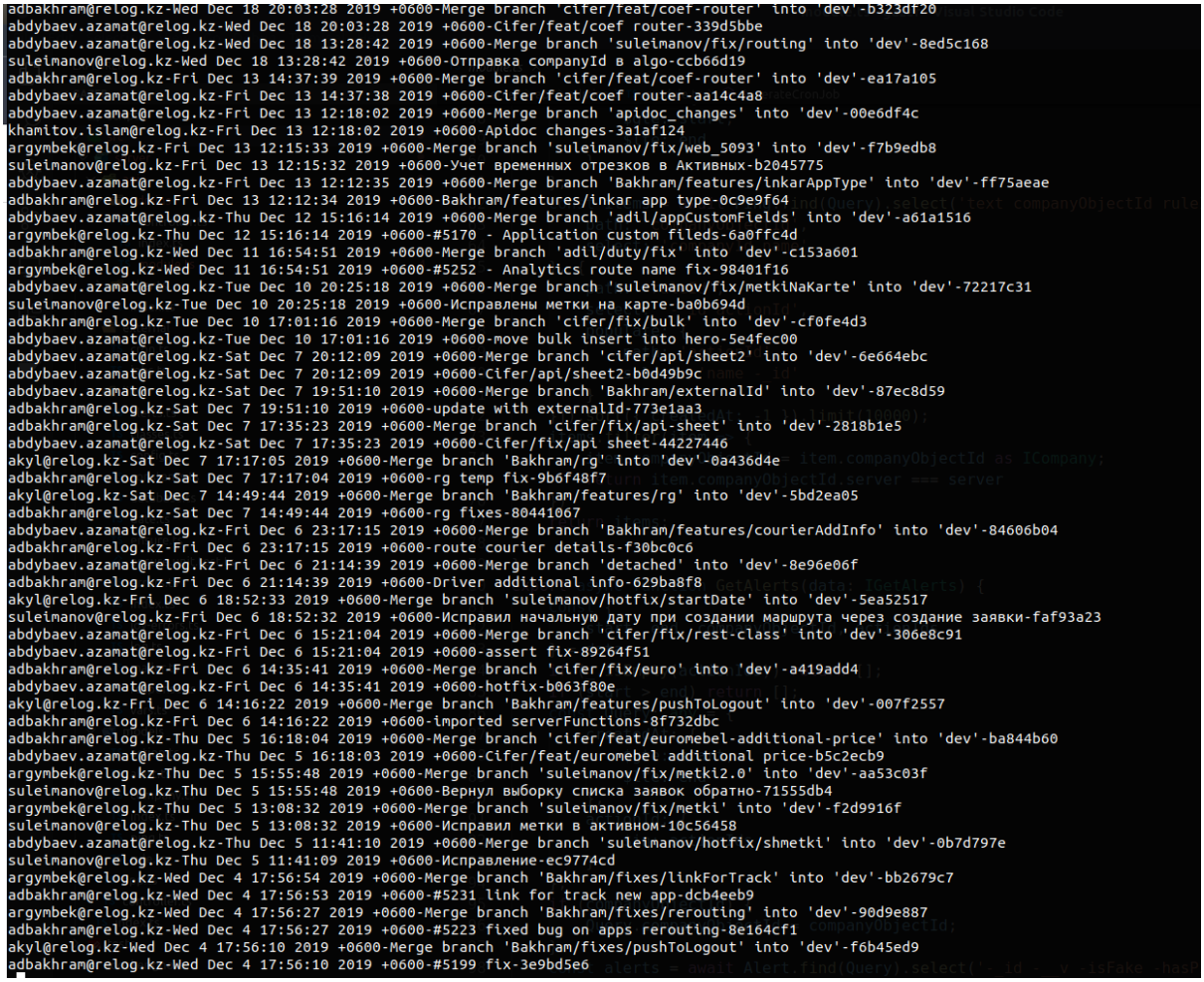
Пример выполнения команды(просмотр коммитов в формате: автор - время коммита - комментарий коммита)

cifer@pc:repos/gazer ‹dev›$ git log --pretty=format:"%ae-%cd-%s-%h "



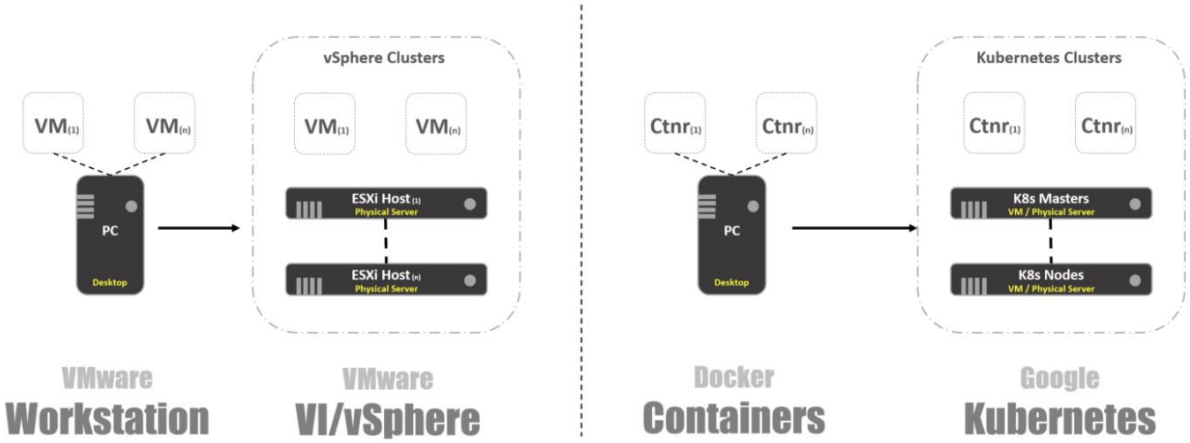
Также были проделаны работы в основном монолите.

cifer@pc:repos/relog ‹dev›$ git log --pretty=format:"%ae-%cd-%s-%h "

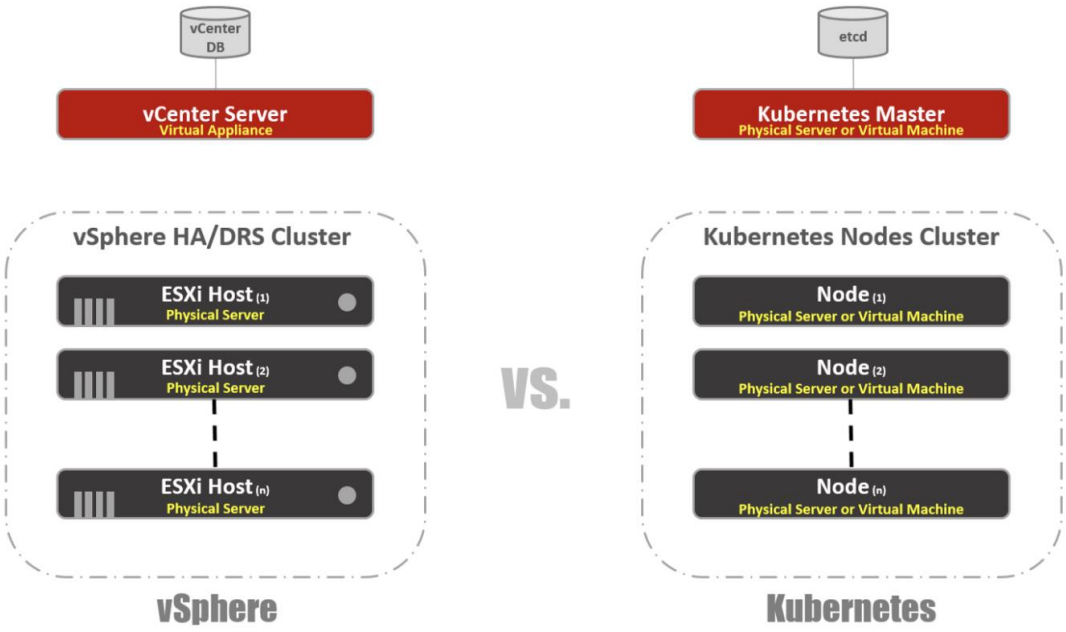

adbakhram@relog.kz-Wed Dec 18 20:03:28 2019 +0600-Merge branch 'cifer/feat/coef-router' into 'dev'-b323df20
abdybaev.azamat@relog.kz-Wed Dec 18 20:03:28 2019 +0600-Cifer/feat/coef router-339d5bbe
abdybaev.azamat@relog.kz-Wed Dec 18 13:28:42 2019 +0600-Merge branch 'suleimanov/fix/routing' into 'dev'-8ed5c168
suleimanov@relog.kz-Wed Dec 18 13:28:42 2019 +0600-Отправка companyId в algo-ccb66d19
adbakhram@relog.kz-Fri Dec 13 14:37:39 2019 +0600-Merge branch 'cifer/feat/coef-router' into 'dev'-ea17a105
abdybaev.azamat@relog.kz-Fri Dec 13 14:37:38 2019 +0600-Cifer/feat/coef router-aa14c4a8
abdybaev.azamat@relog.kz-Fri Dec 13 12:18:02 2019 +0600-Merge branch 'apidoc_changes' into 'dev'-00e6df4c
khamitov.islam@relog.kz-Fri Dec 13 12:18:02 2019 +0600-Apidoc changes-3a1af124
argymbek@relog.kz-Fri Dec 13 12:15:33 2019 +0600-Merge branch 'suleimanov/fix/web_5093' into 'dev'-f7b9edb8
suleimanov@relog.kz-Fri Dec 13 12:15:32 2019 +0600-Учет временных отрезков в Активных-b2045775
abdybaev.azamat@relog.kz-Fri Dec 13 12:12:35 2019 +0600-Merge branch 'Bakhram/features/inkarAppType' into 'dev'-ff75aeae
adbakhram@relog.kz-Fri Dec 13 12:12:34 2019 +0600-Bakhram/features/inkar app type-0c9e9f64
abdybaev.azamat@relog.kz-Thu Dec 12 15:16:14 2019 +0600-Merge branch 'adil/appCustomFields' into 'dev'-a61a1516
argymbek@relog.kz-Thu Dec 12 15:16:14 2019 +0600-#5170 - Application custom fileds-6a0ffc4d
adbakhram@relog.kz-Wed Dec 11 16:54:51 2019 +0600-Merge branch 'adil/duty/fix' into 'dev'-c153a601
argymbek@relog.kz-Wed Dec 11 16:54:51 2019 +0600-#5252 - Analytics route name fix-98401f16
abdybaev.azamat@relog.kz-Tue Dec 10 20:25:18 2019 +0600-Merge branch 'suleimanov/fix/metkiNaKarte' into 'dev'-72217c31
suleimanov@relog.kz-Tue Dec 10 20:25:18 2019 +0600-Исправлены метки на карте-ba0b694d
adbakhram@relog.kz-Tue Dec 10 17:01:16 2019 +0600-Merge branch 'cifer/fix/bulk' into 'dev'-cf0fe4d3
abdybaev.azamat@relog.kz-Tue Dec 10 17:01:16 2019 +0600-move bulk insert into hero-5e4fec00
abdybaev.azamat@relog.kz-Sat Dec 7 20:12:09 2019 +0600-Merge branch 'cifer/api/sheet2' into 'dev'-6e664ebc
abdybaev.azamat@relog.kz-Sat Dec 7 20:12:09 2019 +0600-Cifer/api/sheet2-b0d49b9c
abdybaev.azamat@relog.kz-Sat Dec 7 19:51:10 2019 +0600-Merge branch 'Bakhram/externalId' into 'dev'-87ec8d59
adbakhram@relog.kz-Sat Dec 7 19:51:10 2019 +0600-update with externalId-773e1aa3
adbakhram@relog.kz-Sat Dec 7 17:35:23 2019 +0600-Merge branch 'cifer/fix/api-sheet' into 'dev'-2818b1e5
abdybaev.azamat@relog.kz-Sat Dec 7 17:35:23 2019 +0600-Cifer/fix/api sheet-44227446
akyl@relog.kz-Sat Dec 7 17:17:05 2019 +0600-Merge branch 'Bakhram/rg' into 'dev'-0a436d4e
adbakhram@relog.kz-Sat Dec 7 17:17:04 2019 +0600-rg temp fix-9b6f48f7
akyl@relog.kz-Sat Dec 7 14:49:44 2019 +0600-Merge branch 'Bakhram/features/rg' into 'dev'-5bd2ea05
adbakhram@relog.kz-Sat Dec 7 14:49:44 2019 +0600-rg fixes-80441067
abdybaev.azamat@relog.kz-Fri Dec 6 23:17:15 2019 +0600-Merge branch 'Bakhram/features/courierAddInfo' into 'dev'-84606b04
adbakhram@relog.kz-Fri Dec 6 23:17:15 2019 +0600-route courier details-f30bc0c6
abdybaev.azamat@relog.kz-Fri Dec 6 21:14:39 2019 +0600-Merge branch 'detached' into 'dev'-8e96e06f
adbakhram@relog.kz-Fri Dec 6 21:14:39 2019 +0600-Driver additional info-629ba8f8
akyl@relog.kz-Fri Dec 6 18:52:33 2019 +0600-Merge branch 'suleimanov/hotfix/startDate' into 'dev'-5ea52517
suleimanov@relog.kz-Fri Dec 6 18:52:32 2019 +0600-Исправил начальную дату при создании маршрута через создание заявки-faf93a23
abdybaev.azamat@relog.kz-Fri Dec 6 15:21:04 2019 +0600-Merge branch 'cifer/fix/rest-class' into 'dev'-306e8c91
abdybaev.azamat@relog.kz-Fri Dec 6 15:21:04 2019 +0600-assert fix-89264f51
adbakhram@relog.kz-Fri Dec 6 14:35:41 2019 +0600-Merge branch 'cifer/fix/euro' into 'dev'-a419add4
abdybaev.azamat@relog.kz-Fri Dec 6 14:35:41 2019 +0600-hotfix-b063f80e
akyl@relog.kz-Fri Dec 6 14:16:22 2019 +0600-Merge branch 'Bakhram/features/pushToLogout' into 'dev'-007f2557
adbakhram@relog.kz-Fri Dec 6 14:16:22 2019 +0600-imported serverFunctions-8f732dbc
adbakhram@relog.kz-Thu Dec 5 16:18:04 2019 +0600-Merge branch 'cifer/feat/euromebel-additional-price' into 'dev'-ba844b60
abdybaev.azamat@relog.kz-Thu Dec 5 16:18:03 2019 +0600-Cifer/feat/euromebel additional price-b5c2ecb9
argymbek@relog.kz-Thu Dec 5 15:55:48 2019 +0600-Merge branch 'suleimanov/fix/metki2.0' into 'dev'-aa53c03f
suleimanov@relog.kz-Thu Dec 5 15:55:48 2019 +0600-Вернул выборку списка заявок обратно-71555db4
argymbek@relog.kz-Thu Dec 5 13:08:32 2019 +0600-Merge branch 'suleimanov/fix/metki' into 'dev'-f2d9916f
suleimanov@relog.kz-Thu Dec 5 13:08:32 2019 +0600-Исправил метки в активном-10c56458
abdybaev.azamat@relog.kz-Thu Dec 5 11:41:10 2019 +0600-Merge branch 'suleimanov/hotfix/shmetki' into 'dev'-0b7d797e
suleimanov@relog.kz-Thu Dec 5 11:41:09 2019 +0600-Исправление-ec9774cd
argymbek@relog.kz-Wed Dec 4 17:56:54 2019 +0600-Merge branch 'Bakhram/fixes/linkForTrack' into 'dev'-bb2679c7
adbakhram@relog.kz-Wed Dec 4 17:56:53 2019 +0600-#5231 link for track new app-dcb4eeb9
argymbek@relog.kz-Wed Dec 4 17:56:27 2019 +0600-Merge branch 'Bakhram/fixes/rerouting' into 'dev'-90d9e887
adbakhram@relog.kz-Wed Dec 4 17:56:27 2019 +0600-#5223 fixed bug on apps rerouting-8e164cf1
akyl@relog.kz-Wed Dec 4 17:56:10 2019 +0600-Merge branch 'Bakhram/fixes/pushToLogout' into 'dev'-f6b45ed9
adbakhram@relog.kz-Wed Dec 4 17:56:10 2019 +0600-#5199 fix-3e9bd5e6

Понимание Kubernetes в сравнении с vSphere

Kubernets для контейнеров - это как vSphere для виртуальных машин в современном центре обработки данных. Если вы использовали рабочую станцию VMware в начале 2000-х годов, вы знаете, что это решение было серьезно воспринято для решений центров обработки данных. С появлением VI / vSphere с хостами VCenter и ESXi мир виртуальных машин сильно изменился. Сегодня Kubernets делает то же самое с миром контейнеров, предоставляя возможность работать и управлять контейнерами в производстве. И поэтому мы начнем сравнивать с Kubernetes, чтобы понять детали этой распределенной системы, ее функции и технологии.



Как и vSpere и ESXi являются хостами. Концепция Kubernetes имеет мастеров и узлы. В этом контексте Master эквивалентен k8s vCenter в том смысле, что он представляет собой плоскость управления распределенной системой. Это также точка входа API, с которой вы взаимодействуете при обработке своей рабочей нагрузки. Точно так же K8 Node действуют как вычислительные ресурсы, такие как хосты ESXi. Их задача - запустить рабочую нагрузку (в случае с K8s мы называем их стручками). Узлы могут быть виртуальными машинами или физическими серверами. Конечно, в vSphere ESXi хост всегда должен быть физическим.

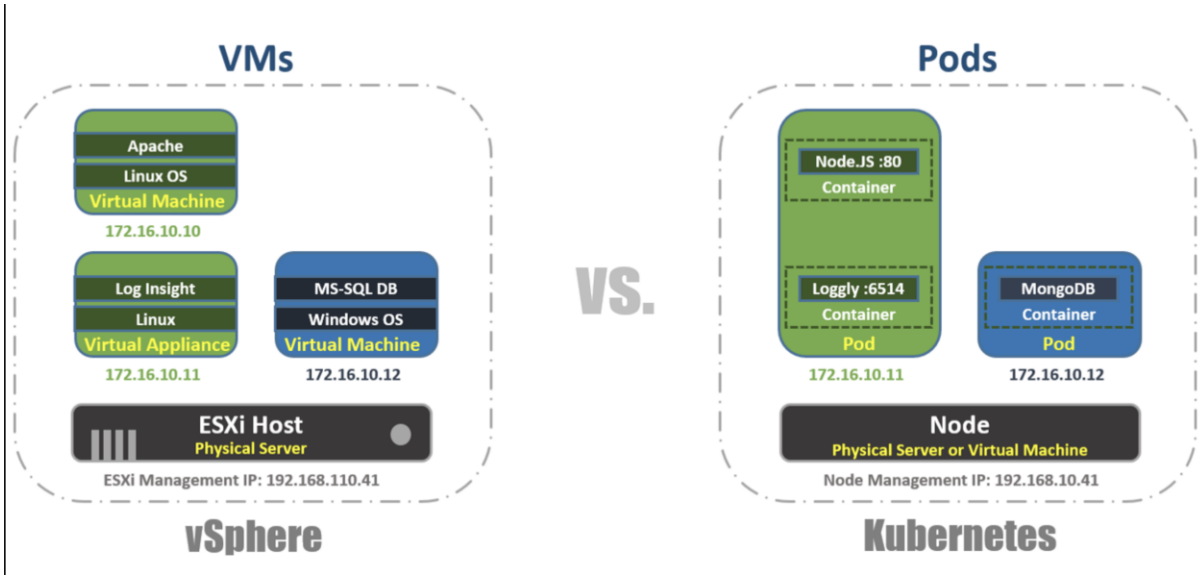


K8s есть key-value хранилище, называемое “etcd”. Это хранилище похоже на базу данных vCenter, где вы сохраняете желаемую конфигурацию кластера, которой хотите придерживаться.

Что касается различий: на Master K8s вы также можете запустить рабочие нагрузки, а на vCenter — нет. vCenter — это Virtual Appliance, выделенный только для управления. В случае K8s Master считается вычислительным ресурсом, но запускать на нем Enterprise приложения не очень хорошая идея.

В vSphere виртуальная машина является логической границей операционной системы. В Kubernetes Pod’ы являются границами для контейнеров, как и ESXi хост, на котором может работать одновременно несколько виртуальных машин. На каждой Node может работать несколько Pod’ов. Каждый Pod получает маршрутизируемый IP-адрес, как и виртуальные машины, для взаимодействия Pod’ов друг с другом.

В vSphere приложения запускаются внутри ОС, а в Kubernetes приложения запускаются внутри контейнеров. Виртуальная машина может одновременно работать только с одной ОС, а Pod может запускать несколько контейнеров.

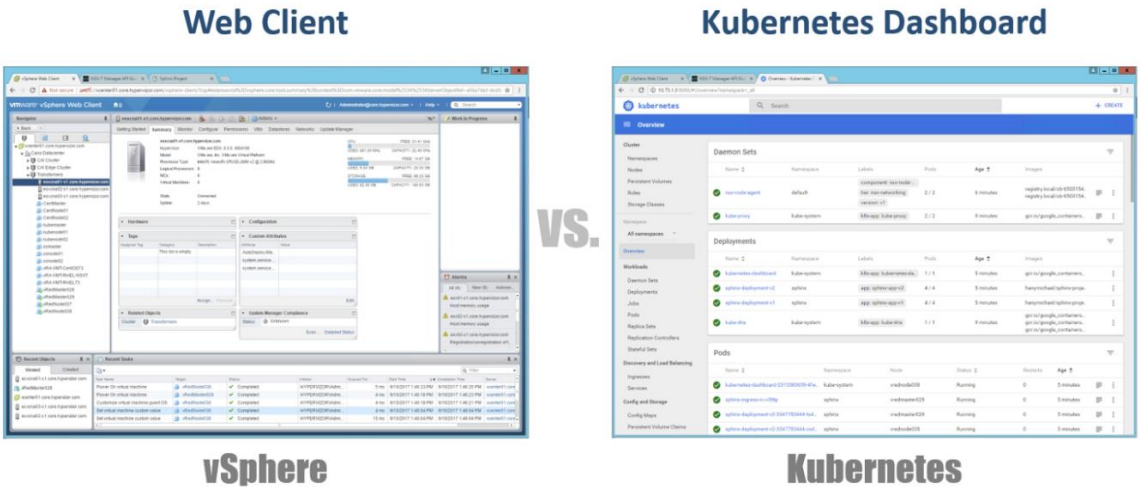


Вот так можно вывести список Pod'ов внутри кластера K8s, используя инструмент kubectl через CLI, проверить работоспособность Pod'ов, их возраст, IP-адрес и Node'ы, на которых они сейчас работают

```
C:\Users\Hany
λ kubectl.exe get po -o wide --all-namespaces
NAMESPACE     NAME                                        READY   STATUS    RESTARTS   AGE     IP              NODE
default       nginx-ingress-rc-g1f5l                      1/1     Running   0          11m     10.105.0.66     vkubemaster007
default       nsx-ncp-sgqt3                               1/1     Running   0          11m     192.168.80.23   vkubemaster007
default       nsx-node-agent-k5fnv                        2/2     Running   0          5m      192.168.80.24   vkubenode018
default       nsx-node-agent-n2nhz                        2/2     Running   0          11m     192.168.80.23   vkubemaster007
default       nsx-node-agent-sssdp                        2/2     Running   0          4m      192.168.80.27   vkubenode019
default       nsx-node-agent-t3g4v                        2/2     Running   0          5m      192.168.80.25   vkubenode020
default       nsx-node-agent-tz68z                        2/2     Running   0          5m      192.168.80.26   vkubenode017
kube-system   etcd-vkubemaster007                         1/1     Running   0          11m     192.168.80.23   vkubemaster007
kube-system   kube-apiserver-vkubemaster007               1/1     Running   0          10m     192.168.80.23   vkubemaster007
kube-system   kube-controller-manager-vkubemaster007      1/1     Running   0          10m     192.168.80.23   vkubemaster007
kube-system   kube-dns-692378583-22hx4                    3/3     Running   0          11m     10.105.0.2      vkubemaster007
kube-system   kube-proxy-b45r8                            1/1     Running   0          5m      192.168.80.24   vkubenode018
kube-system   kube-proxy-g7kh5                            1/1     Running   0          11m     192.168.80.23   vkubemaster007
kube-system   kube-proxy-s1tc5                            1/1     Running   0          4m      192.168.80.27   vkubenode019
kube-system   kube-proxy-xfhl6                            1/1     Running   0          5m      192.168.80.25   vkubenode020
kube-system   kube-proxy-zpfvb                            1/1     Running   0          5m      192.168.80.26   vkubenode017
kube-system   kube-scheduler-vkubemaster007               1/1     Running   0          10m     192.168.80.23   vkubemaster007
sphinx        sphinx-deployment-v1-3129483026-0q0sr       1/1     Running   0          10m     10.105.0.194    vkubemaster007
sphinx        sphinx-deployment-v1-3129483026-hd18j       1/1     Running   0          10m     10.105.0.195    vkubemaster007
sphinx        sphinx-deployment-v1-3129483026-p6z7c       1/1     Running   0          10m     10.105.0.196    vkubemaster007
sphinx        sphinx-deployment-v1-3129483026-xq4jv       1/1     Running   0          10m     10.105.0.197    vkubemaster007
sphinx        sphinx-deployment-v2-3347783444-4gtqh       1/1     Running   0          10m     10.105.0.200    vkubemaster007
sphinx        sphinx-deployment-v2-3347783444-fp3vb       1/1     Running   0          10m     10.105.0.199    vkubemaster007
sphinx        sphinx-deployment-v2-3347783444-mg0x8       1/1     Running   0          10m     10.105.0.201    vkubemaster007
sphinx        sphinx-deployment-v2-3347783444-zvb9x       1/1     Running   0          10m     10.105.0.198    vkubemaster007
sphinx        sphinx-ingress-rc-ngmwz                     1/1     Running   0          10m     10.105.0.202    vkubenode017
```

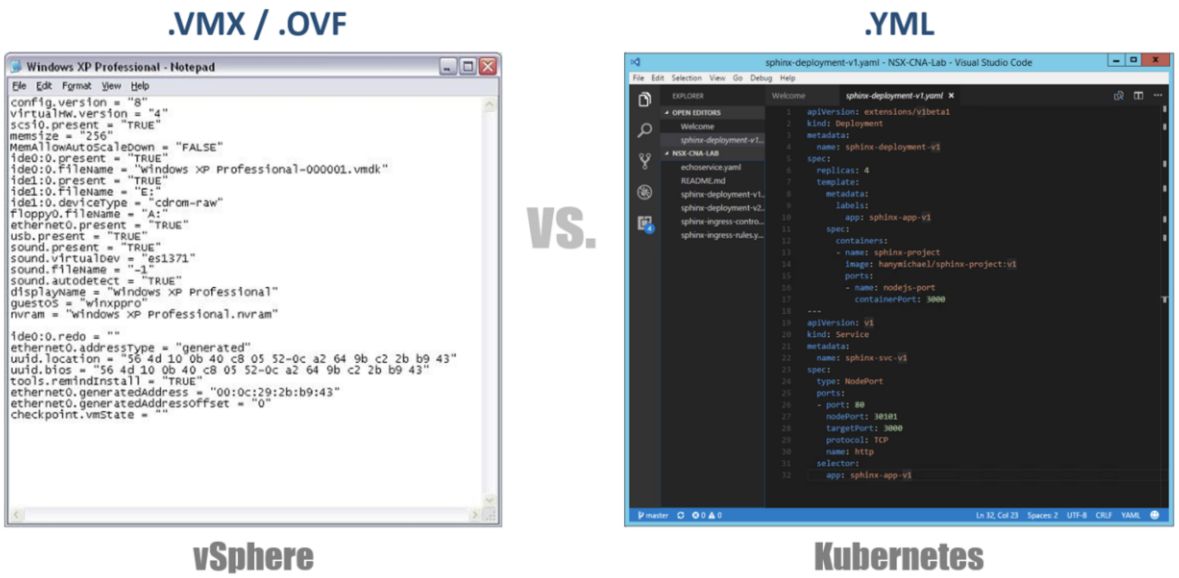
В vSphere мы используем Web клиент для управления большинством (если не всеми) компонент нашей виртуальной инфраструктуры. Для Kubernetes аналогично, с использованием Dashboard. Это хороший Web портал на основе графического интерфейса, к которому вы можете получить доступ через браузер так же, как и с Web клиентом vSphere. Из предыдущих разделов видно, что вы можете управлять своим кластером K8s с помощью команды kubeclt из CLI. Это всегда спорно, где вы будете проводить большую часть своего времени в CLI или в графическом Dashboard. Так как последний с каждым днем становится все более мощным инструментом. Лично я считаю, что Dashboard очень удобен для быстрого мониторинга

состояния или отображения деталей различных компонентов K8s, позволяя не вводить длинные команды в CLI.

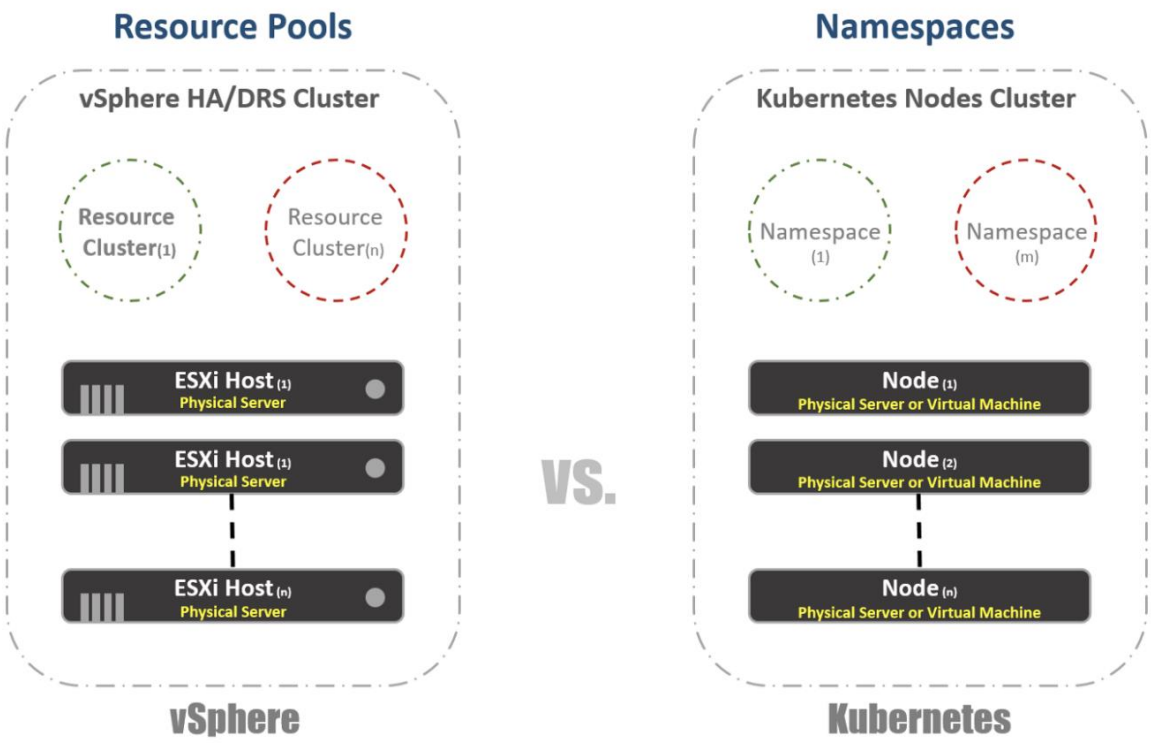


Одна из очень важных концепций в Kubernetes — желаемое состояние конфигураций. Вы декларируете, что вы хотите почти для любого компонента Kubernetes через YAML файл, и вы создаете все это, используя kubectl (или через графический Dashboard) в качестве желаемого состояния. Начиная с этого момента Kubernetes всегда будет стремиться сохранить ваше окружение в заданном, рабочем состоянии. Например, если вы хотите иметь 4 реплики одного Pod'а, K8s будет продолжать следить за этими Pod'ами, и в случае, если один из них умер или у Node, на которых он работал, возникли проблемы, K8s самовосстановится и автоматически создаст этот Pod в другом месте.

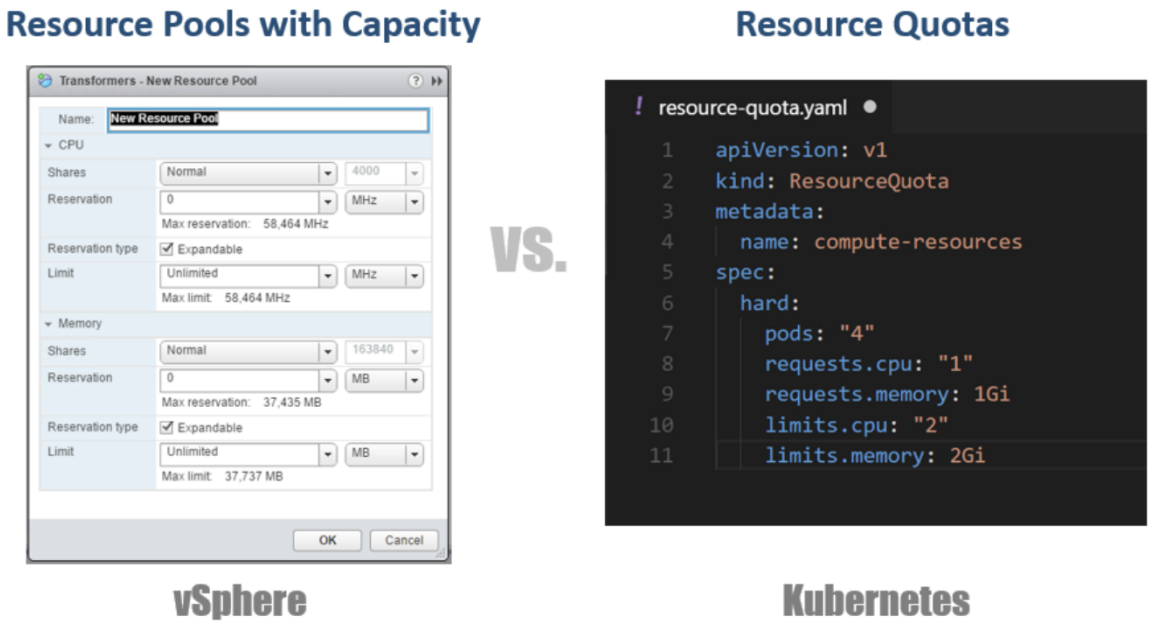
Возвращаясь к нашим YAML файлам конфигурации, вы можете рассматривать их как файл .VMX для виртуальной машины или дескриптор .OVF для Virtual Appliance, который вы хотите развернуть в vSphere. Эти файлы определяют конфигурацию рабочей нагрузки/ компонента, который вы хотите запустить. В отличие от файлов VMX/OVF, которые являются эксклюзивными для виртуальных VMs/Appliances, файлы конфигурации YAML используются для определения любого компонента K8s, такого как ReplicaSets, Services, Deployments и т.д. Рассмотрим это в следующих разделах.



В vSphere у нас есть физические ESXi хосты, логически сгруппированные в кластеры. Эти кластеры можем разделить на другие виртуальные кластеры, называемые “Resource Pools”. Эти “пулы” в основном используются для ограничения ресурсов. В Kubernetes у нас есть что-то очень похожее. Мы называем их “Namespaces”, они также могут использоваться для обеспечения ограничения ресурсов, что будет отражено в следующем разделе. Однако чаще всего “Namespaces” используются как средство multi-tenancy для приложений (или пользователей, если вы используете общие K8s кластеры). Это также один из вариантов, с помощью которого можно выполнить сетевую сегментацию с помощью NSX-T. Рассмотрим это в следующих публикациях.

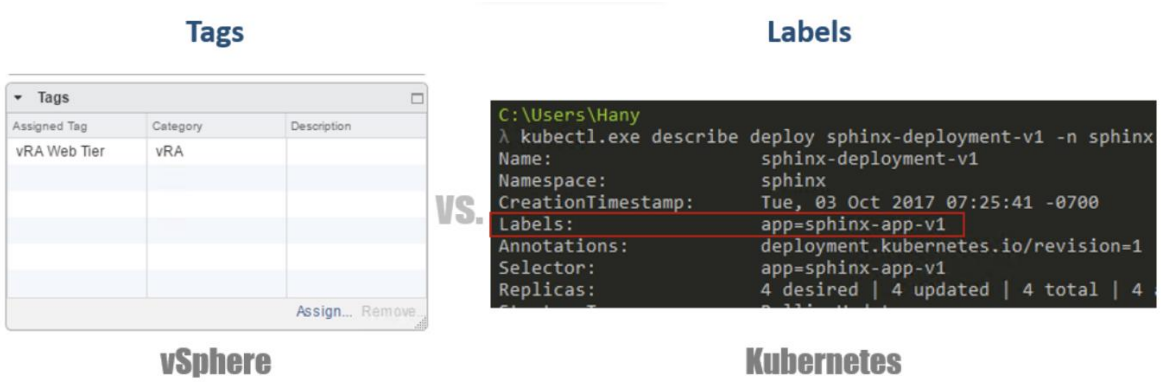


Namespaces в Kubernetes обычно используются как средство сегментации. Другим вариантом использования Namespaces является распределение ресурсов. Этот вариант называется “Resource Quotas”. Как следует из предыдущих разделов, определение этого происходит в конфигурационных YAML файлах, в которых объявляется желаемое состояние. В vSphere, как видно на скриншоте ниже, определяем это из настроек Resource Pools.

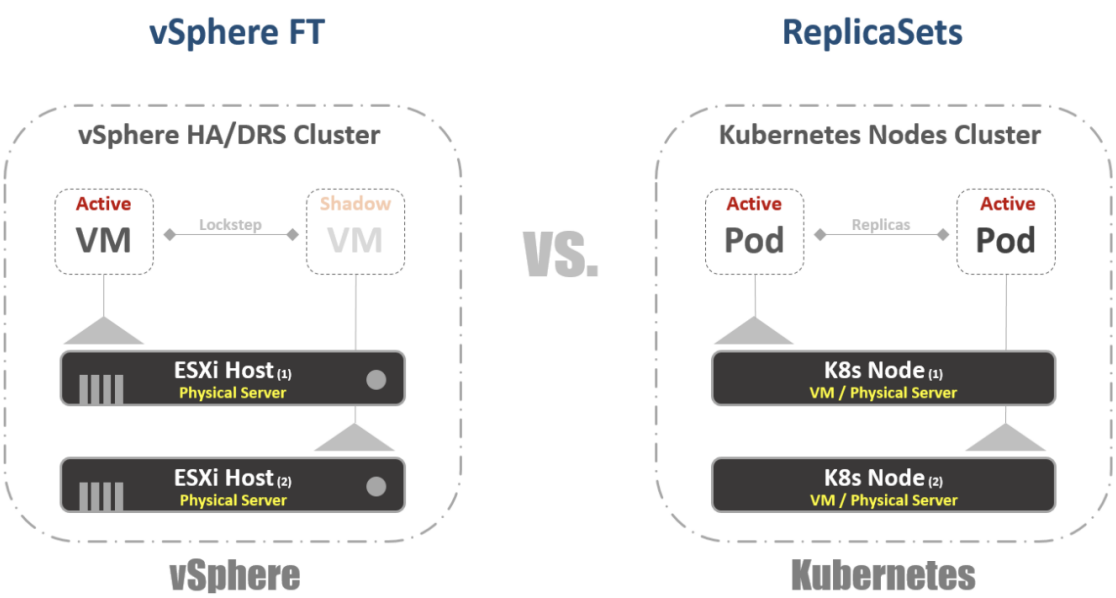


Идентификация рабочих нагрузок

В первом случае мы используем понятия Tags для определения (или группировки) аналогичных рабочих нагрузок, а во втором используем термин “Labels”. В случае с Kubernetes идентификация рабочих нагрузок является обязательной.



Если вы были или являетесь большим поклонником vSphere FT, как я, вам понравится эта функция в Kubernetes, несмотря на некоторые различия в двух технологиях. В vSphere это виртуальная машина с работающим теневым экземпляром, запущенным на другом хосте. Мы записываем инструкции на основной виртуальной машине и воспроизводим их на теневой виртуальной машине. Если основная машина перестала работать то, теневая виртуальная машина включается немедленно. Затем vSphere пытается найти другой ESXi хост, чтобы создать новый теневой экземпляр виртуальной машины для сохранения той же избыточности. В Kubernetes у нас есть нечто очень похожее. ReplicaSets — это количество, которое вы указываете для запуска нескольких экземпляров Pod’ов. Если один Pod выходит из строя, другие экземпляры доступны для обслуживания трафика. В то же время, K8s попытается запустить новый Pod на любой доступной Node, чтобы поддерживать желаемое состояние конфигурации. Основное отличие, как вы, возможно, уже заметили, состоит в том, что в случае с K8s Pod’ы всегда работают и обслуживают трафик. Они не являются теневыми рабочими нагрузками.



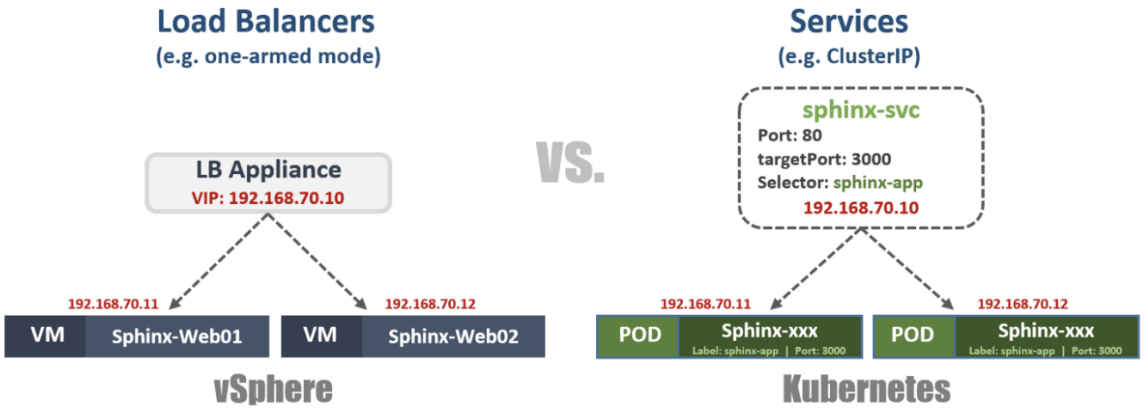
Балансировки нагрузки

Хотя это может и не быть встроенной функцией в vSphere, все же очень и очень часто нужно запускать балансировщики нагрузки на платформе. В мире vSphere есть виртуальные или физические балансировщики нагрузки для распределения сетевого трафика между несколькими виртуальными машинами. Может быть много разных режимов конфигурации, но давайте предположим, что мы имеем в виду One-Armed конфигурацию. В этом случае вы балансируете нагрузку East-West трафиком на свои виртуальные машины.

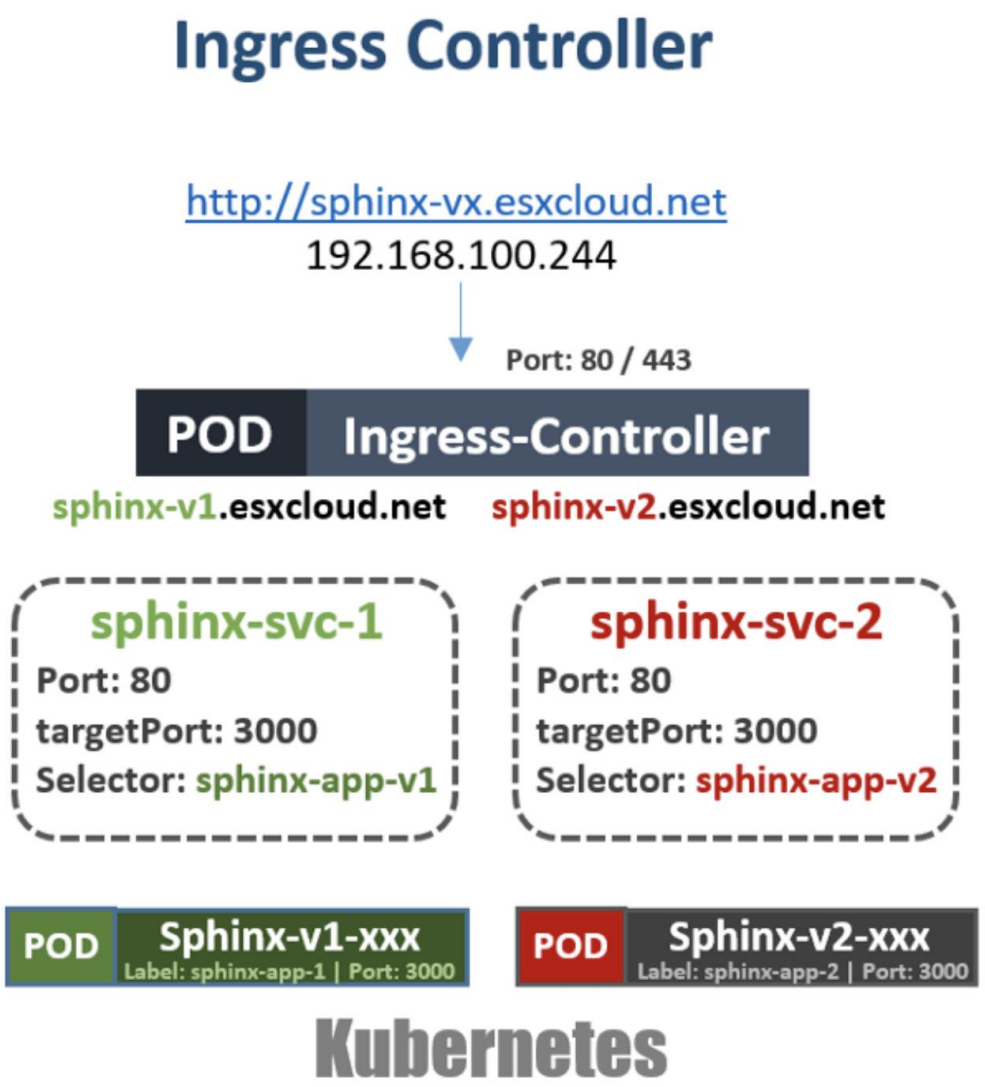
Точно также в Kubernetes есть понятия “Services”. Service в K8s также можно использовать в разных режимах конфигурации. Давайте выберем конфигурацию “ClusterIP’, чтобы сравнить ее с One-Armed Load Balancer. В этом случае Service в K8s будет иметь виртуальный IP-адрес (VIP), который всегда статичен и не изменяется. Этот VIP будет распределять трафик между несколькими Pod’ами. Это особенно важно в мире Kubernetes, где по своей природе Pod’ы эфемерны, вы теряете IP-адрес Pod’а в тот момент, когда он умирает или удаляется. Таким образом, вы всегда должны обеспечивать статичный VIP.

У Service есть много других конфигураций, например, “NodePort”, где вы назначаете порт на уровне Node’ы, а затем выполняете port-address-translation преобразование для Pod’ов. Существует также “LoadBalancer”,

где вы запускаете экземпляр Load Balancer от стороннего или облачного провайдера.



В Kuberentes есть еще один очень важный механизм балансировки нагрузки, который называется “Ingress Controller”. Вы можете считать его in-line application балансировщиком нагрузки. Основная идея заключается в том, что Ingress Controller (в форме Pod’a) будет запущен с видимым извне IP-адресом. Этот IP-адрес может иметь нечто вроде Wildcard DNS записи. Когда трафик попадает на Ingress Controller с использованием внешнего IP-адреса, он проверяет заголовки и определяет с помощью набора правил, которые вы предварительно установили, какому Pod’у принадлежит это имя. Например: sphinx-v1.esxcloud.net будет направлен на Service sphinx-svc-1, а sphinx-v2.esxcloud.net будет направлен на Service sphinx-svc2 и т.д.



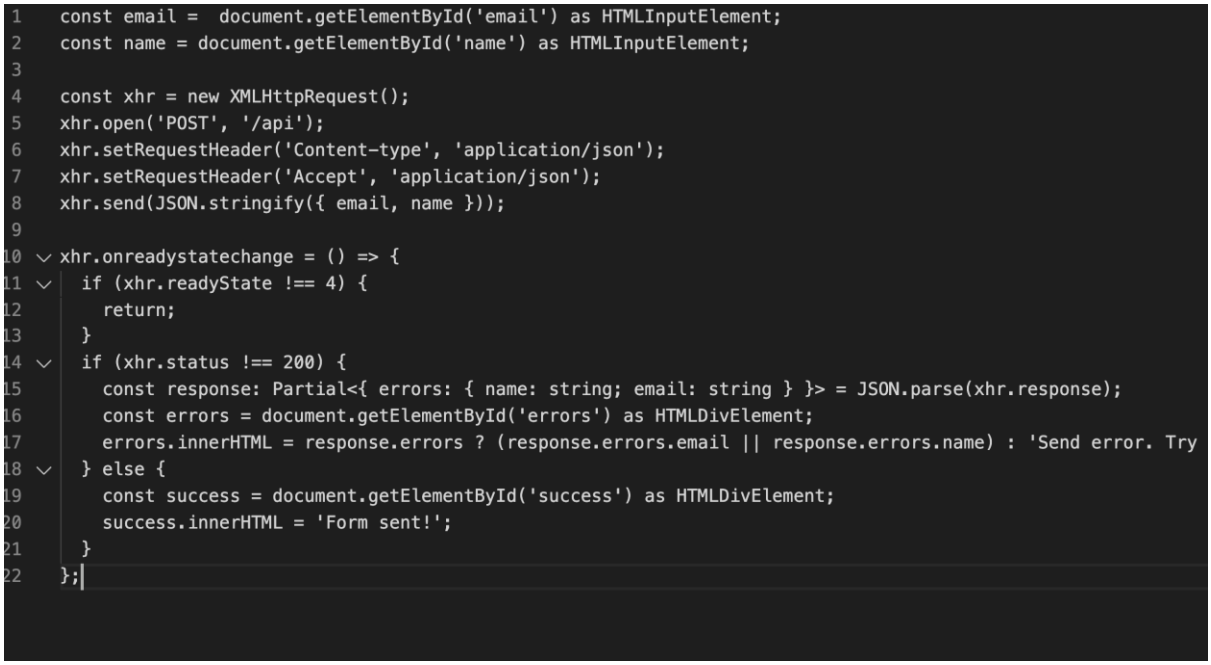
Конфигурация Typescript и Webpack

TypeScript — это расширенная версия JavaScript, главной целью которого является упрощение разработки крупных JS-приложений. Этот язык добавляет много новых принципов — классы, дженерики, интерфейсы, статические типы, что позволяет разработчикам использовать разные инструменты, такие как статический анализатор или рефакторинг кода.

Иногда появляются задачи, по разработке небольшого приложения, где не нужны современные фронтенд фреймворки. И как не был прост React или

Vue, задачи обычно сводятся, что нужно сверстать страничку и добавить форму, которая будет отправлять данные на указанный endpoint.

В данном случае, TS/JS кода будет не больше 20 строк кода:



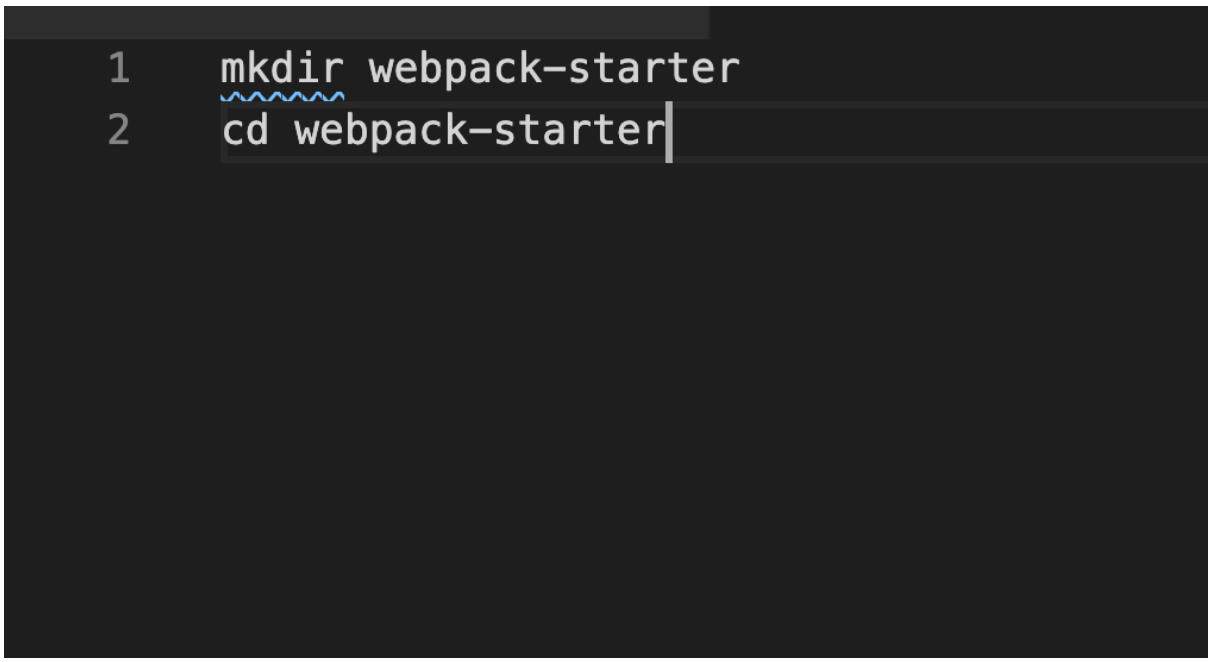
```
const email =  document.getElementById('email') as HTMLInputElement;
const name = document.getElementById('name') as HTMLInputElement;

const xhr = new XMLHttpRequest();
xhr.open('POST', '/api');
xhr.setRequestHeader('Content-type', 'application/json');
xhr.setRequestHeader('Accept', 'application/json');
xhr.send(JSON.stringify({ email, name }));

xhr.onreadystatechange = () => {
  if (xhr.readyState !== 4) {
    return;
  }
  if (xhr.status !== 200) {
    const response: Partial<{ errors: { name: string; email: string } }> = JSON.parse(xhr.response);
    const errors = document.getElementById('errors') as HTMLDivElement;
    errors.innerHTML = response.errors ? (response.errors.email || response.errors.name) : 'Send error. Try
  } else {
    const success = document.getElementById('success') as HTMLDivElement;
    success.innerHTML = 'Form sent!';
  }
};
```

И разворачивать полноценный фреймворк нет смысла. Для данного кода можно отказаться от всех современных благ разработки, если бы не одно но — Необходимо сверстать страничку. А это значит, что нужен CSS препроцессор, нужна автоматическая перезагрузка при изменении кода, а также сборка и оптимизация кода.
И вместо того, чтобы использовать полноценный фреймворк можно сделать простую сборку на webpack.
Основные требования к сборке:

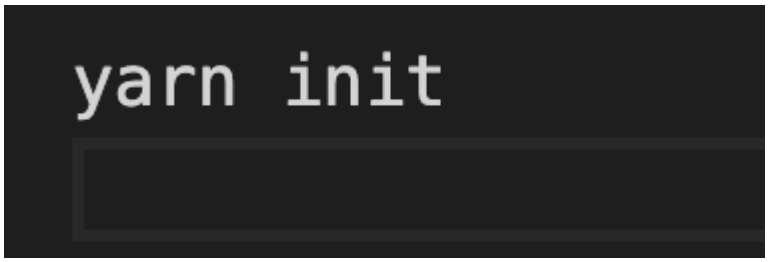
- Typescript — компиляция js
- Livereload — автоматическая перезагрузка
- SCSS — компиляция scss в css
- Html optimize — оптимизация html для prod версии
- Assets — правильные пути к используемым ресурсам (images, fonts, video и т.д)

Создание сборки

Для создания сборки создадим новый node проект:



Инициализируем проект:




```
→  Projects mkdir webpack-starter
→  Projects cd webpack-starter
→  webpack-starter yarn init
yarn init v1.22.4
question name (webpack-starter):
question version (1.0.0):
question description: Demo webpack starter
question entry point (index.js):
question repository url:
question author:
question license (MIT):
question private:
success Saved package.json
Done in 23.07s.
```

Установка зависимостей

Добавим основные зависимости для webpack:

yarn add -D webpack webpack-cli webpack-dev-server webpack-merge

Добавим Typescript:
yarn add -D typescript @types/node

Так как необходимы лоадеры для typescript, обработки обычных файлов(assets), файлов стилей и html, добавим лоадеры:

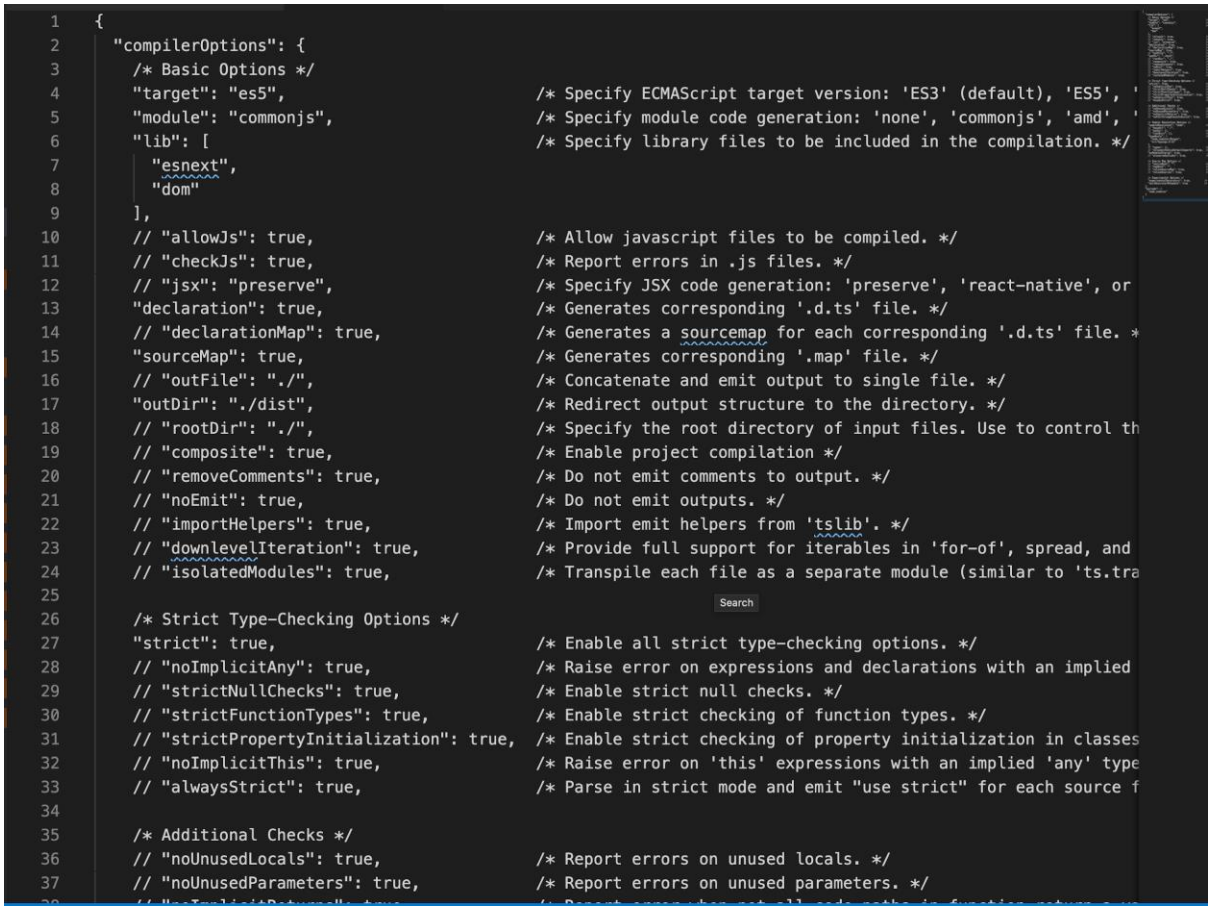
yarn add -D style-loader sass-loader node-sass ts-loader file-loader css-loader postcss-loader postcss-import postcss-cssnext cssnano sugarss

Так как папки для dev и prod нужно чистить, добавим необходимые плагины:
yarn add -D clean-webpack-plugin copy-webpack-plugin mini-css-extract-plugin html-webpack-plugin tsconfig-paths-webpack-plugin

И для контроля качества кода, добавим tslint и prettier:
yarn add -D tslint tslint-loader prettier

Добавление конфигураций

Сначала добавим tsconfig:



```
{
  "compilerOptions": {
    /* Basic Options */
    "target": "es5",                          /* Specify ECMAScript target version: 'ES3' (default), 'ES5', '
    "module": "commonjs",                     /* Specify module code generation: 'none', 'commonjs', 'amd', '
    "lib": [                                  /* Specify library files to be included in the compilation. */
      "esnext",
      "dom"
    ],
    // "allowJs": true,                       /* Allow javascript files to be compiled. */
    // "checkJs": true,                       /* Report errors in .js files. */
    // "jsx": "preserve",                     /* Specify JSX code generation: 'preserve', 'react-native', or
    "declaration": true,                      /* Generates corresponding '.d.ts' file. */
    // "declarationMap": true,                /* Generates a sourcemap for each corresponding '.d.ts' file. *
    "sourceMap": true,                        /* Generates corresponding '.map' file. */
    // "outFile": "./",                       /* Concatenate and emit output to single file. */
    "outDir": "./dist",                       /* Redirect output structure to the directory. */
    // "rootDir": "./",                       /* Specify the root directory of input files. Use to control th
    // "composite": true,                     /* Enable project compilation */
    // "removeComments": true,                /* Do not emit comments to output. */
    // "noEmit": true,                        /* Do not emit outputs. */
    // "importHelpers": true,                 /* Import emit helpers from 'tslib'. */
    // "downlevelIteration": true,            /* Provide full support for iterables in 'for-of', spread, and
    // "isolatedModules": true,               /* Transpile each file as a separate module (similar to 'ts.tra

    /* Strict Type-Checking Options */
    "strict": true,                           /* Enable all strict type-checking options. */
    // "noImplicitAny": true,                 /* Raise error on expressions and declarations with an implied
    // "strictNullChecks": true,              /* Enable strict null checks. */
    // "strictFunctionTypes": true,           /* Enable strict checking of function types. */
    // "strictPropertyInitialization": true,  /* Enable strict checking of property initialization in classes
    // "noImplicitThis": true,                /* Raise error on 'this' expressions with an implied 'any' type
    // "alwaysStrict": true,                  /* Parse in strict mode and emit "use strict" for each source f

    /* Additional Checks */
    // "noUnusedLocals": true,                /* Report errors on unused locals. */
    // "noUnusedParameters": true,            /* Report errors on unused parameters. */
```

Конфигурация postcss:

```
module.exports = ({ file, options, env }) => ({
  parser: false,
  plugins: {
    'postcss-import': {},
    'postcss-cssnext': {},
    'cssnano':  env === 'production'  ? {} : false
  }
});
```

Настройка линтеров:

```
{
  "extends": ["plugin:prettier/recommended"],
  "rules": {
    "max-len": [
      "error",
      {
        "code": 140
      }
    ],
    "prettier/prettier": "error",
    "semi": ["error", "always"],
    "quotes": ["error", "single"]
  },
  "parserOptions": {
    "ecmaVersion": 2020
  },
  "plugins": ["prettier"],
  "env": {
    "es6": true
  }
}
```

Настройка webpack:
Общая настройка будет следующей:

```
const path = require('path');
const CleanWebpackPlugin = require('clean-webpack-
plugin').CleanWebpackPlugin;
const CopyWebpackPlugin = require('copy-webpack-plugin');
const MiniCssExtractPlugin = require('mini-css-extract-plugin');
const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
  entry: './src/main.ts',
  output: {
```

```
  path: path.resolve(__dirname, 'dist'),
  filename: 'bundle-[hash].js'
},
resolve: {
  extensions: ['.ts', '.tsx', '.js']
},
devServer: {
  contentBase: 'dist',
  compress: true,
  port: 3000
},
module: {
  rules: [
    {
      test: /\.ts$/,
      enforce: 'pre',
      loader: 'tslint-loader',
      options: {/* Loader options go here */}
    },
    {
      test: /\.tsx?$/,
      loader: 'ts-loader'
    },
    {
      test: /\.(sa|sc|c)ss$/,
      use: [
        {
          loader: MiniCssExtractPlugin.loader,
          options: {
            hmr: process.env.NODE_ENV === 'development',
          },
        },
        'css-loader',
        'postcss-loader',
        'sass-loader',
      ],
    },
```

```
      {
        test: /\.(svg|woff|woff2|ttf|eot|otf)([\?]?.*)$/,
        loader: 'file-loader?name=assets/fonts/[name].[ext]',
      }
    ],
  },
  plugins: [
    new CleanWebpackPlugin(),
    new CopyWebpackPlugin([
      {
        from: 'src/index.html',
        to: './index.html'
      },
      {
        from: 'src/assets/**/*',
        to: './assets',
        transformPath(targetPath, absolutePath) {
          return targetPath.replace('src/assets', '');
        }
      },
    ]),
    new HtmlWebpackPlugin({
      template: 'src/index.html',
      minify: {
        collapseWhitespace: true,
        removeComments: true,
        removeRedundantAttributes: true,
        useShortDoctype: true
      }
    }),
    new MiniCssExtractPlugin({
      filename: 'style-[hash].css',
      allChunks: true
    })
  ]
};
```

Сборка приложения

Для сборки приложения добавим в package.json 3 команды:

```
"scripts": {
  "serve": "webpack-dev-server --mode=development",
  "build": "webpack",
  "build:prod": "webpack --config webpack.config.prod.js"
}
```

В результате выполнения команды создались 3 файла:

- index.html
- bundle-[hash].js
- style-[hash].css

В сборке используется hash, чтобы при обновлении файлов на сервере, браузер перезагрузил файлы, а не брал их из кеша.

ЗАКЛЮЧЕНИЕ

Были выполнены поставленные задачи(проблемы).Таким образом в результате работы была произведена миграция всех микросервисов и монолита в **Kubernetes**. Дальнейшая поставка и разработка продукта выполнялась в полном цикле: сборка, тестирование, релиз. Помимо этого была настроена масштабируемость и отказоустойчивость сервисов. Отказоустойчивость была настроена при помощи 3 мастера ноды. Также в инфраструктуру входит системы мониторинга и логирование, такие как **Zabbix**, **ELK, Grafana.**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Gary Gruver, Starting and Scaling DevOps in the Enterprise, 2016
2. Gigi Sayfan, Mastering Kubernetes: Master the art of container management by using the power of Kubernetes / Gigi Sayfan - 2nd Edition, 2018
3. Fernando Doglio, Pro REST API Development with Node.js / Fernando Doglio La Paz, Canelones Uruguay - 2015
4. Simon Holmes, Getting MEAN with Mongo, Express, Angular, and Node / Simon Holmer - 2015
5. John Paul Miles, Continuous Integration and Deployment with Gitlab, Docker-compose, and DigitalOcean / John Paul Miles - medium.com, Aug. 2017
6. Феофантов К.В., Власов А.В., Афанасьев Г.И., СОЗДАНИЕ КЛАСТЕРА POSTGRESQL В СРЕДЕ KUBERNETES / Современные научные исследования и инновации. 2017. № 2. С. 131-134.
7. Сурай О.В., МАСШТАБИРОВАНИЕ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ НА БАЗЕ KUBERNETES / Актуальные научные исследования в современном мире. 2018. № 4-10 (36). С. 118-122