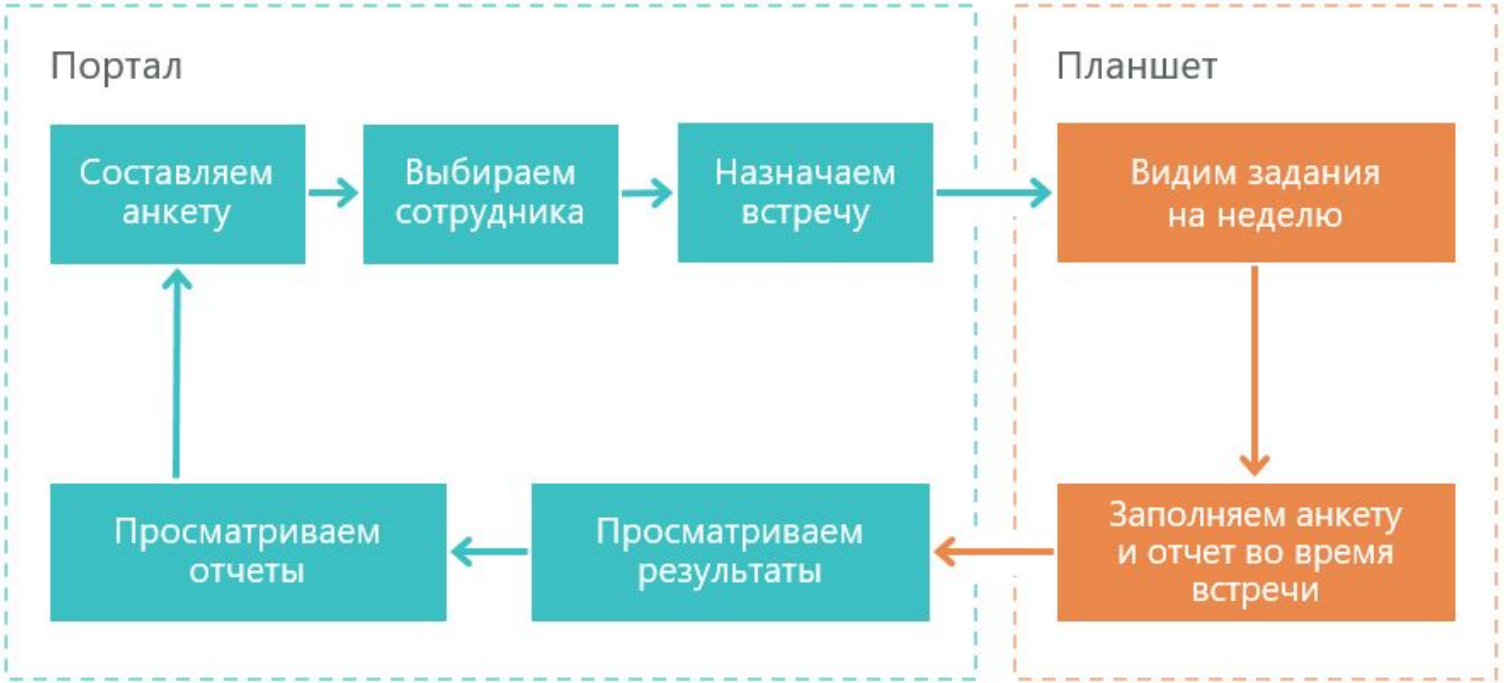
.Net Core и kubernetes

опыт создания приложения

Бизнес-кейс


Портал
Составляем анкету
Выбираем сотрудника
Назначаем встречу
Просматриваем отчеты
Просматриваем результаты
Планшет
Видим задания на неделю
Заполняем анкету и отчет во время встречи

Технологии


kubernetes


.NET
Core


TFS

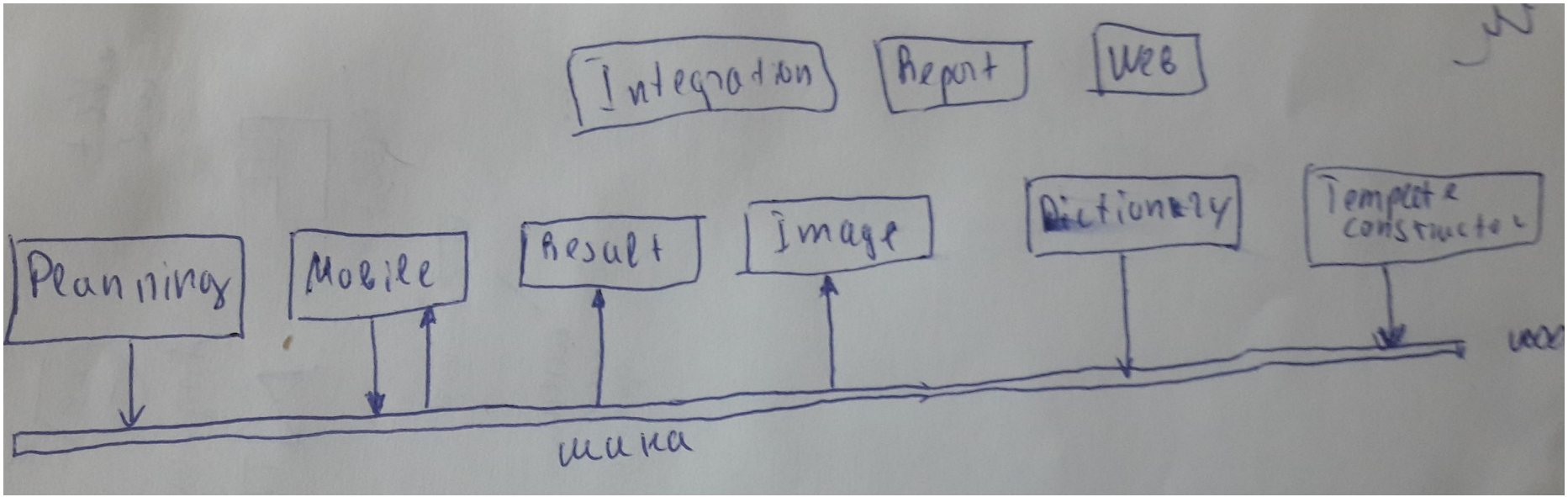

docker


RabbitMQ


ANGULARJS
by Google

Переход на .Net Core

Архитектура решения

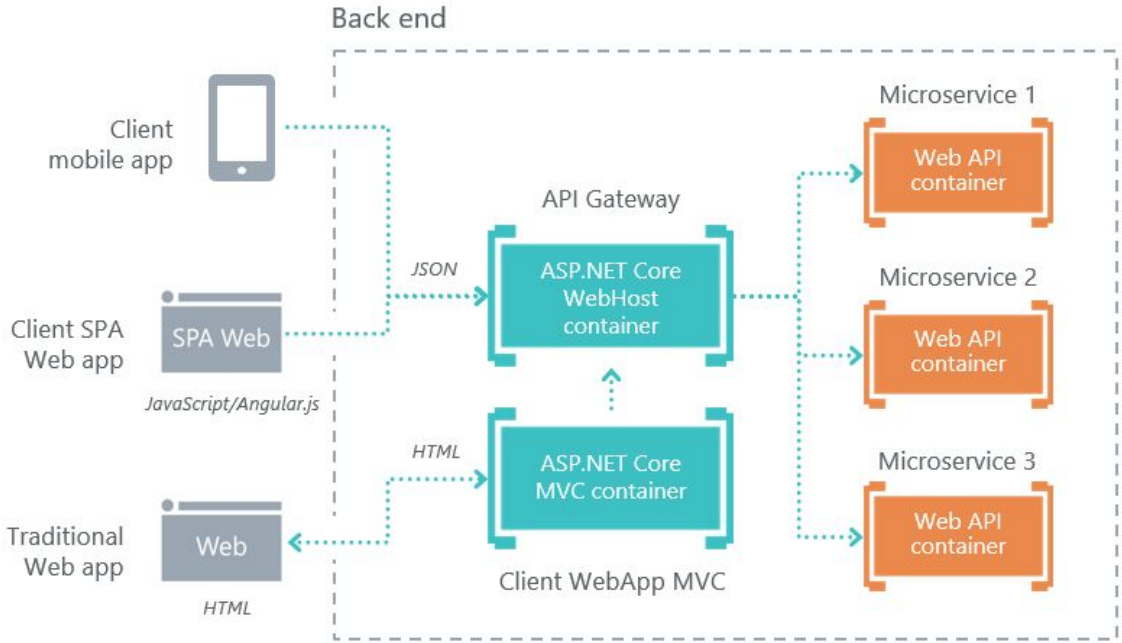


Взаимодействия с внешними потребителями

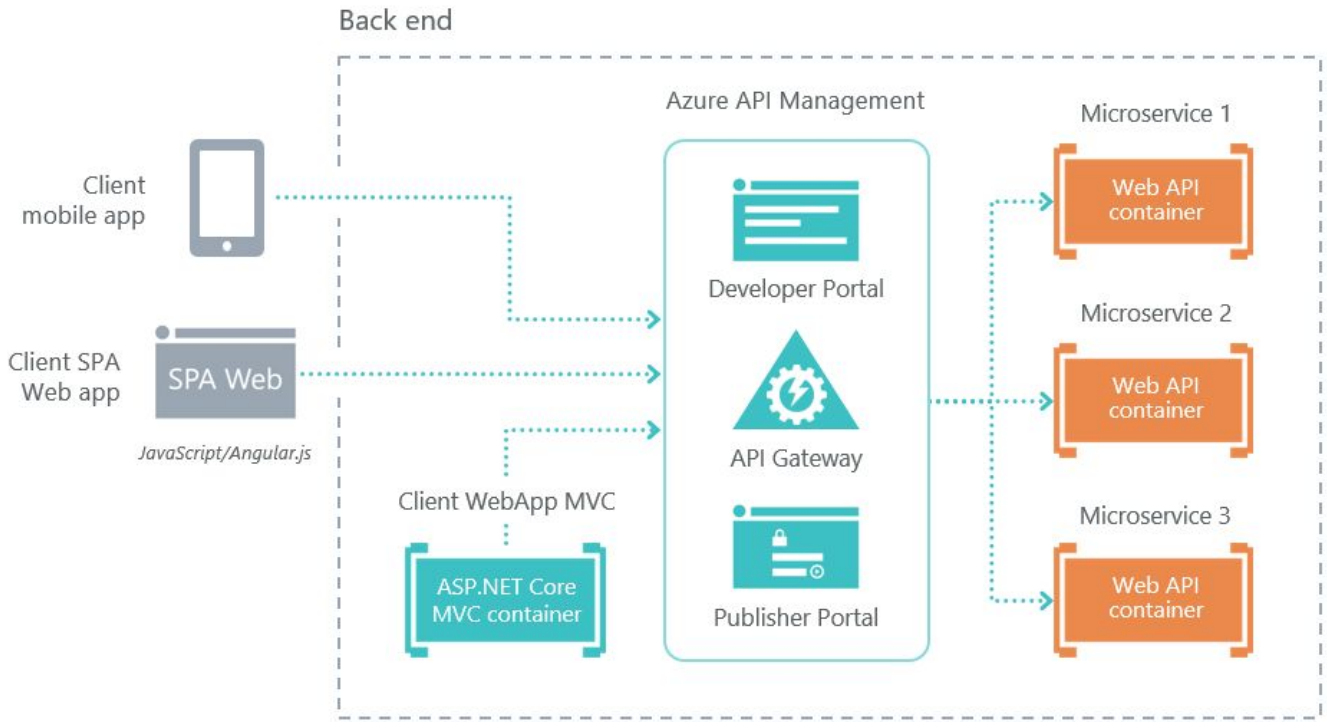
Варианты:

1. API Gateway service
2. API Gateway with Azure API Management
3. Direct Client-To-Microservice communication

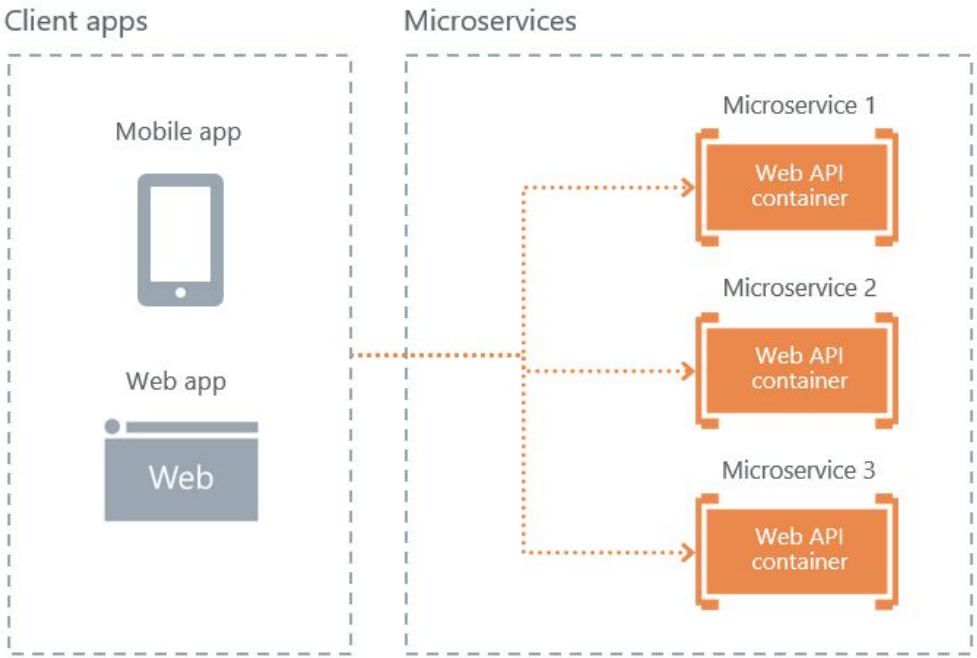
API Gateway service



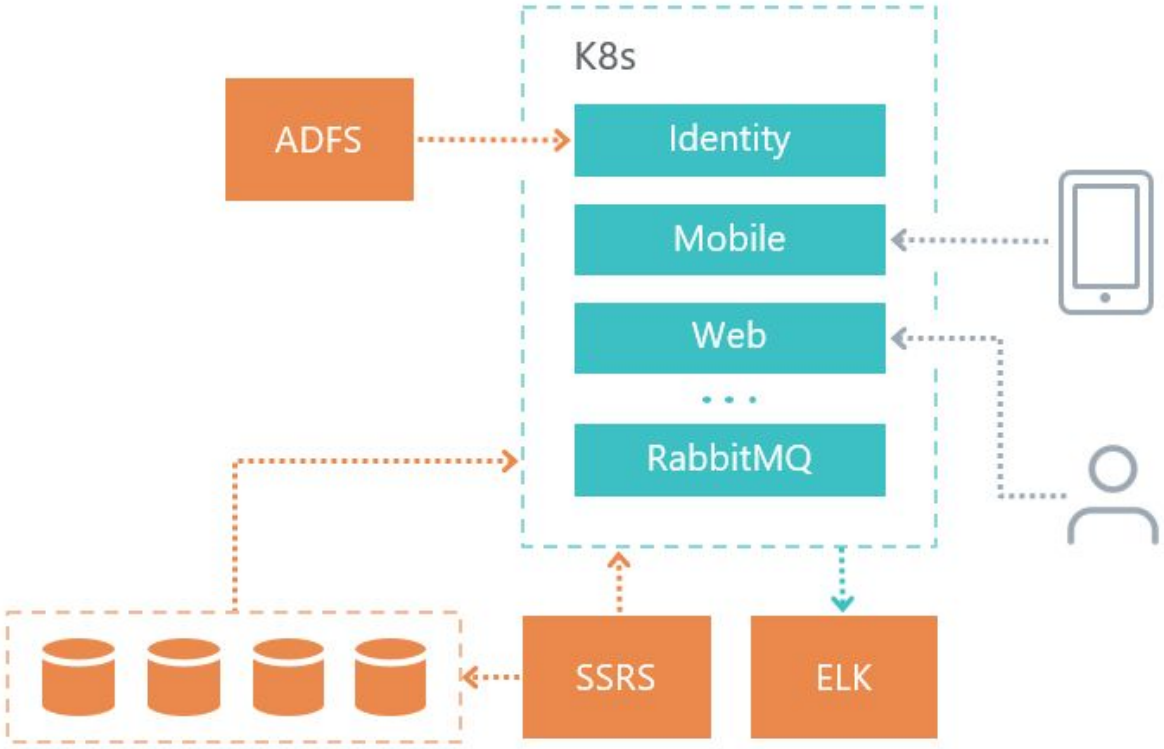
API Gateway with Azure API Management



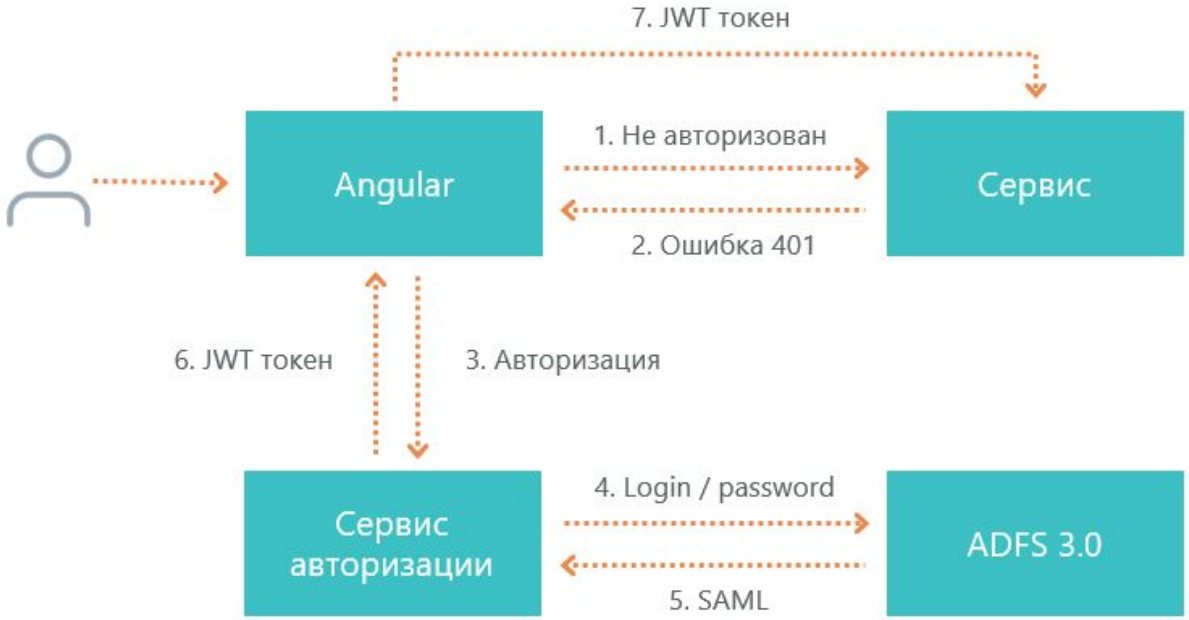
Direct Client-To-Microservice communication



Физическая архитектура

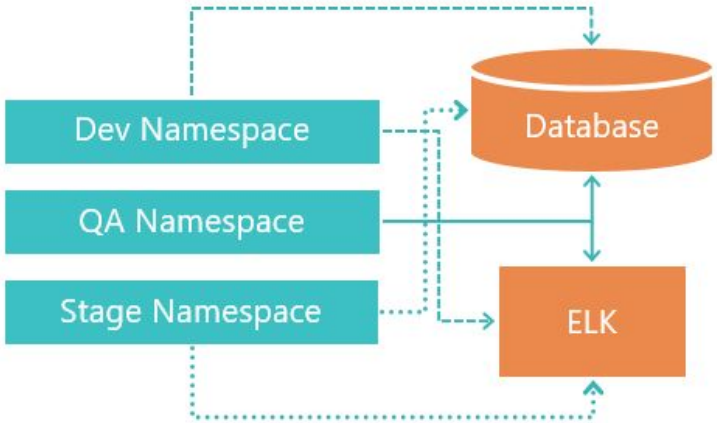


Реализация авторизации

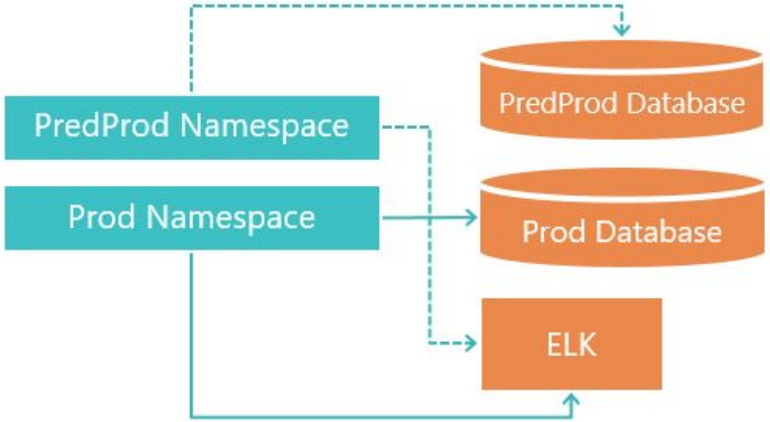


Как мы организовали окружения

Kubernetes корпоративный



Kubernetes заказчика



Развертывание микросервиса в kubernetes

1. Secret
2. Deployement
3. Service (если необходимо)

Развертывание deployment

kubectl apply -f .\imtob-etr-it-dictionary-api.yml --namespace=DEV

```yaml
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: imtob-etr-it-dictionary-api
spec:
  replicas: 1
  template:
    metadata:
      labels:
        name: imtob-etr-it-dictionary-api
    spec:
      containers:
        - name: imtob-etr-it-dictionary-api
          image: nexus3.eastbanctech.ru:18085/etr-it-dictionary-api:18289
          resources:
            requests:
              memory: "256Mi"
            limits:
              memory: "512Mi"
          volumeMounts:
          - name: secrets
            mountPath: /app/secrets
            readOnly: true
      volumes:
      - name: secrets
        secret:
          secretName: secret-appsettings-dictionary
      imagePullSecrets:
        - name: regsecret-imtob
```

Развертывание service

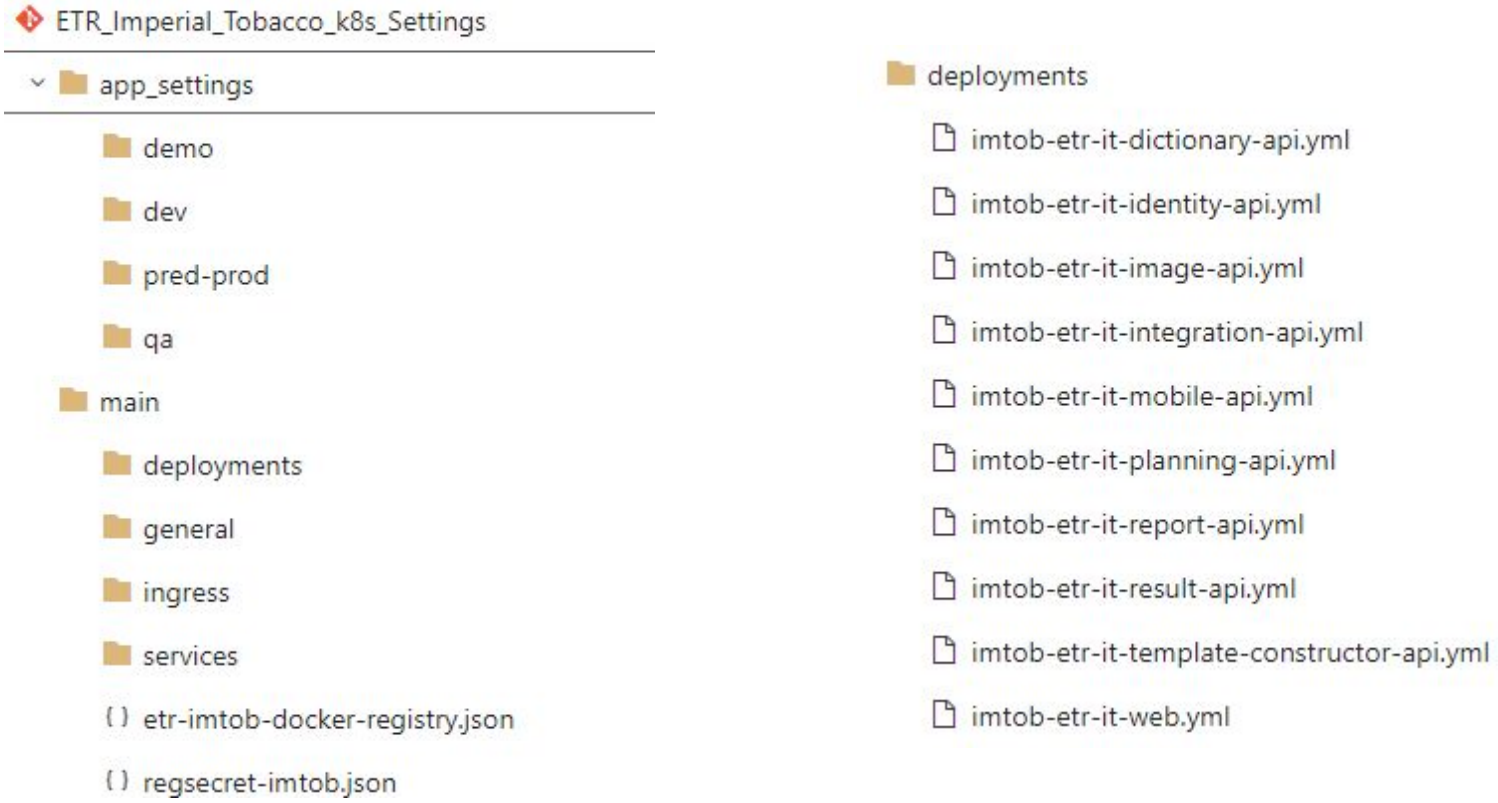
kubectl apply -f .\imtob-etr-it-dictionary-api-services.yml --namespace=DEV

```yaml
apiVersion: v1
kind: Service
metadata:
  name: imtob-etr-it-image-api-services
spec:
  ports:
  - name: http
    port: 80
    targetPort: 80
    protocol: TCP
  selector:
    name: imtob-etr-it-image-api
```

Развертывание микросервиса в kubernetes

1. Secret
2. Deployement
3. Service (если необходимо)

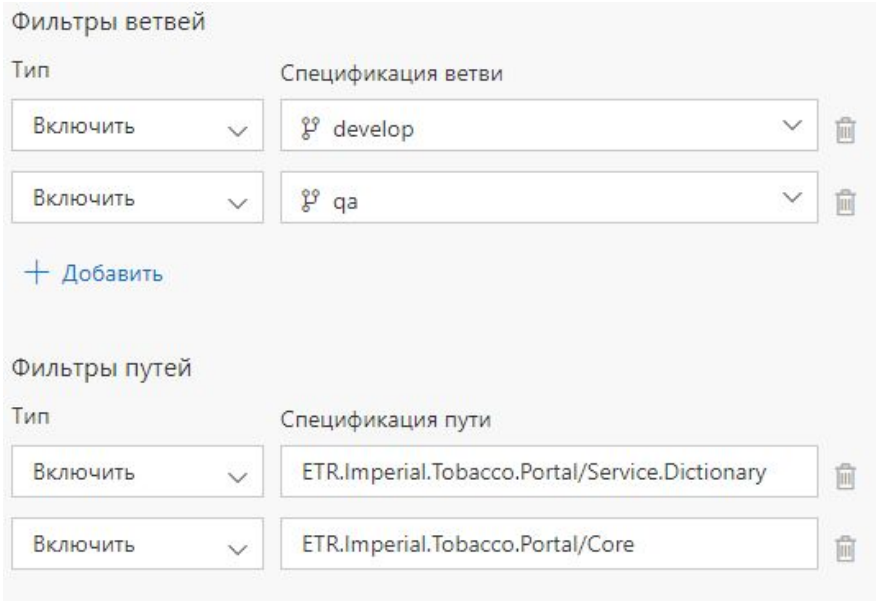
Структура YAML-конфигураций



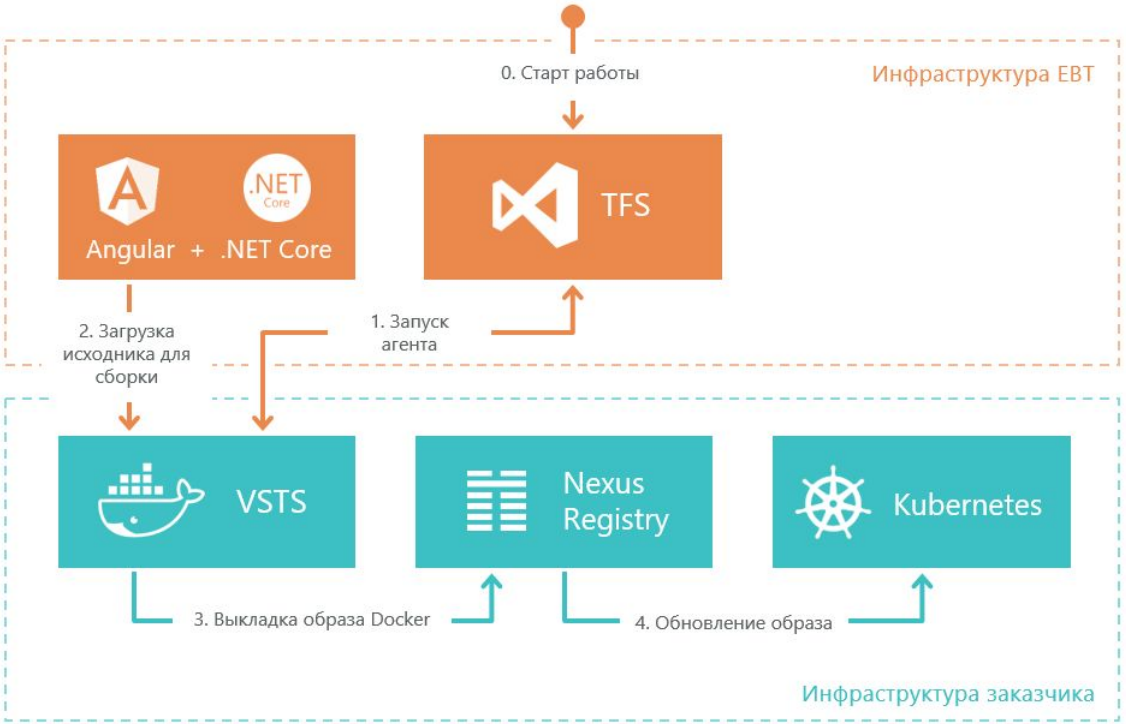
Скрипты развертывания

```
kubectl apply -f .\imtob-etr-it-image-api.yml --namespace=DEV
kubectl apply -f .\imtob-etr-it-mobile-api.yml --namespace=DEV
kubectl apply -f .\imtob-etr-it-planning-api.yml --namespace=DEV
kubectl apply -f .\imtob-etr-it-result-api.yml --namespace=DEV
kubectl apply -f .\imtob-etr-it-web.yml --namespace=DEV
kubectl apply -f .\imtob-etr-it-report-api.yml --namespace=DEV
kubectl apply -f .\imtob-etr-it-template-constructor-api.yml --namespace=DEV
kubectl apply -f .\imtob-etr-it-dictionary-api.yml --namespace=DEV
kubectl apply -f .\imtob-etr-it-integration-api.yml --namespace=DEV
kubectl apply -f .\imtob-etr-it-identity-api.yml --namespace=DEV
```

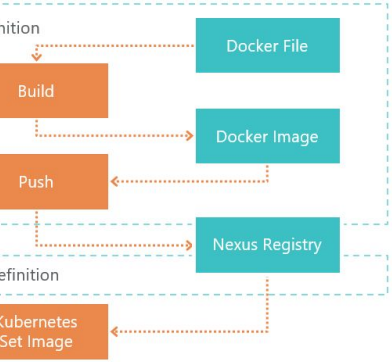
Что нам предоставляет TFS для организации CI

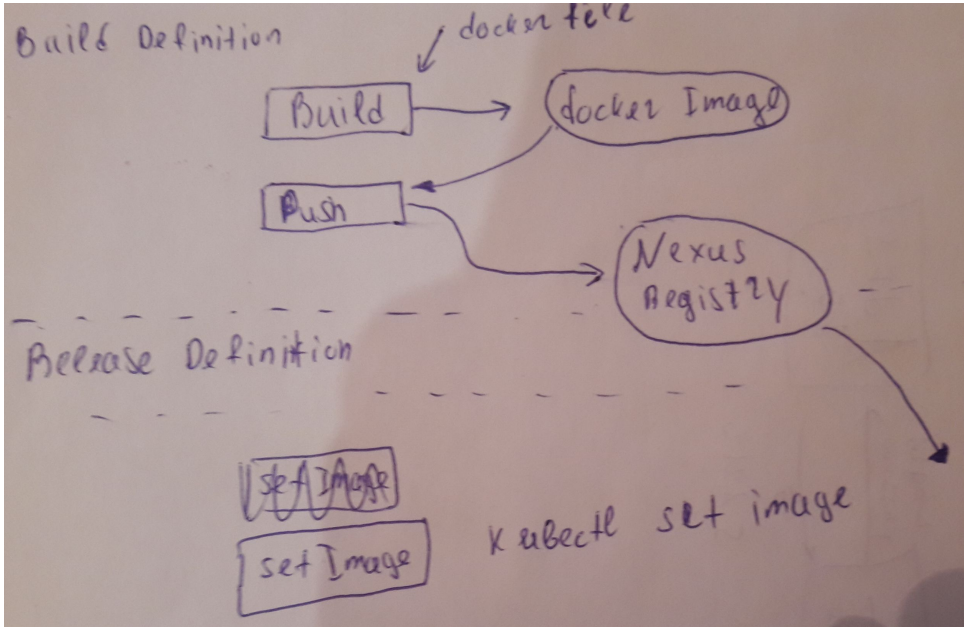


VSTS-агент и развертывание решения в окружении заказчика

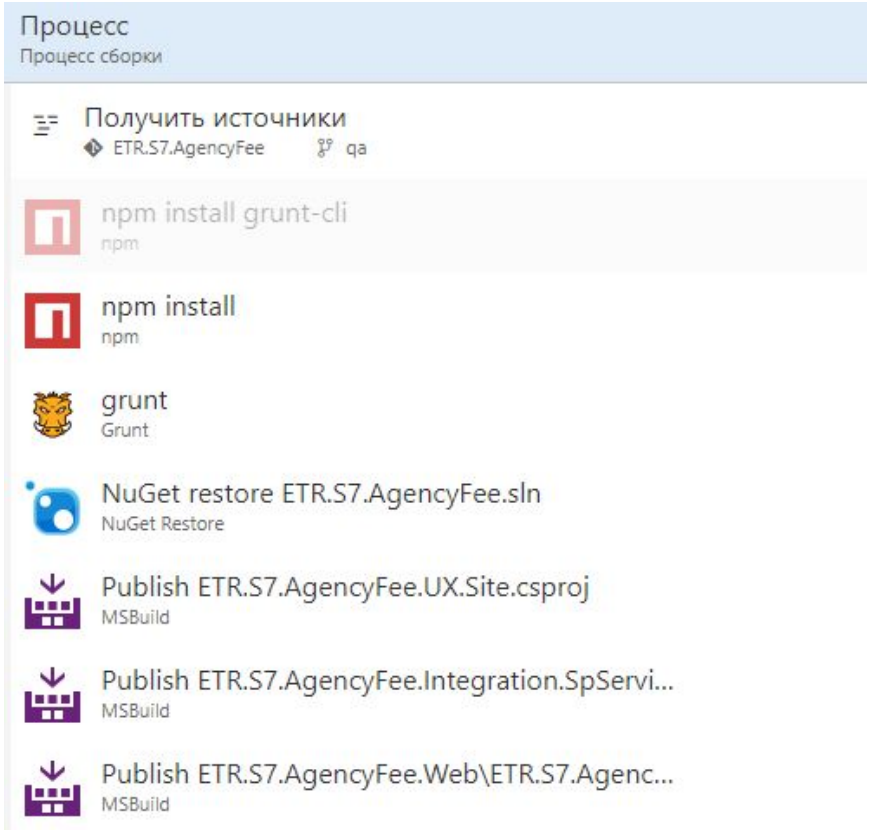


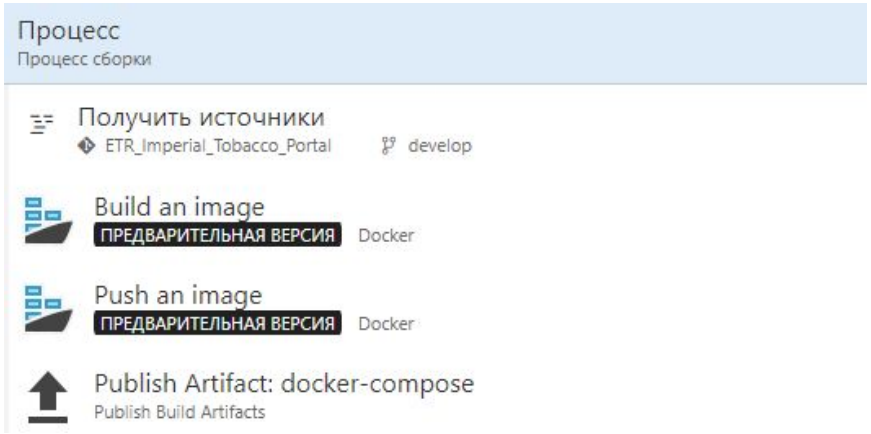
Взаимодействие VSTS-агента и Nexus Registry



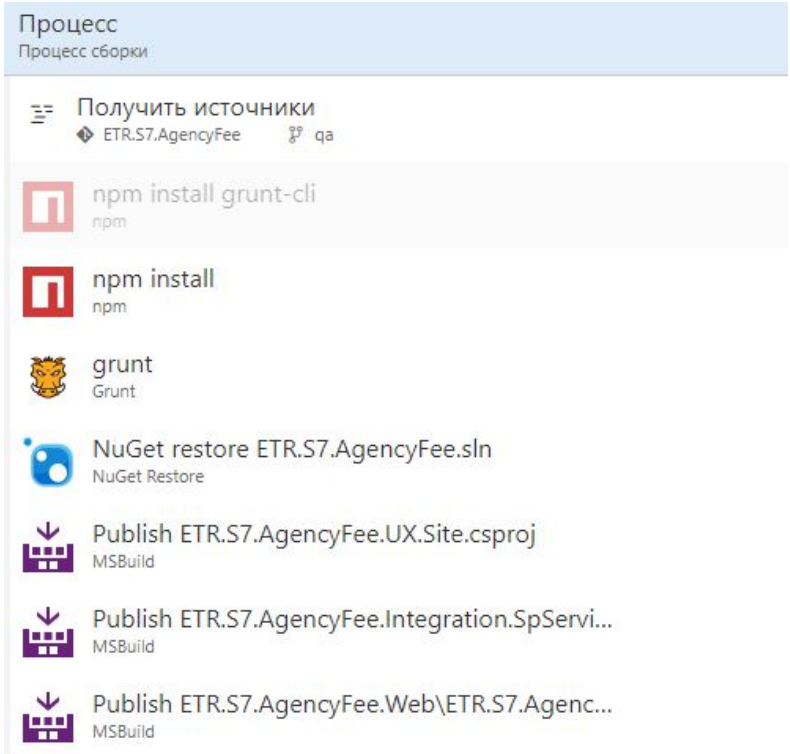


Build Definition





Процесс сборки с помощью Dockerfile




```
FROM microsoft/aspnetcore:2.0 AS base
WORKDIR /app
EXPOSE 80

FROM microsoft/aspnetcore-build:2.0 AS build
WORKDIR /src
COPY ETR.Imperial.Tobacco.Portal.sln ./
COPY Web/ETR.IT.Portal.Web/ETR.IT.Portal.Web.csproj Web/ETR.IT.Portal.Web/
RUN dotnet restore -nowarn:msb3202,nu1503
COPY . .
WORKDIR /src/Web/ETR.IT.Portal.Web/ClientApp
RUN npm run i
WORKDIR /src/Web/ETR.IT.Portal.Web/ClientApp
RUN npm run build:dev
WORKDIR /src/Web/ETR.IT.Portal.Web
RUN dotnet build -c Release -o /app

FROM build AS publish
RUN dotnet publish -c Release -o /app

FROM base AS final
WORKDIR /app
COPY --from=publish /app .
ENTRYPOINT ["dotnet", "ETR.IT.Portal.Web.dll"]
```

Итоги использования TFS и K8S

В результате мы получили простой и прозрачный CI/CD:

- Разделение части разработки и развертывания.
  Сборка описывается в Dockerfile и лежит на плечах разработчика.

- При настройке CI/CD не нужно знать о деталях и особенностях сборки - работа ведется только с Dockerfile.

- Автоматическое определение зависимостей

Хранение конфигурации сервисов

1. Настраиваем загрузку конфигурации из папки secrets

```
var configuration = new ConfigurationBuilder()
    .AddJsonFile($"appsettings.json", true)
     .AddJsonFile("secrets/appsettings.secrets.json", optional: true)
     .Build();
```

2. Делаем mount папки secrets в kubernetes в настройках deployment-а

3. Добавляем новый secret в kubernetes

```
kubectl create secret generic secret-appsettings-dictionary
    --from-file=./Dictionary/appsettings.secrets.json --namespace=XXX
```

Настройки RabbitMQ в K8S

Проблема: (суть)

Как решали: читайте статью Скриншот + (QR) - завтра будут.

Тесты

1. InMemory EF для Unit-тестирования каждого микросервиса в момент сборки.
2. После развертывания в DEV выполняются интеграционные тесты - сценариев newman-а

Мониторинг

Модифицированный ELK стэк:

- Внутри приложения - Serilog
- Данные отправляем напрямую в Elasticsearch
- Kibana - для разработчиков инженеров поддержки

Работа с состоянием

Получилось несколько БД. У которых внутри простая структура.

Для поддержания ее используется EF - Migrations

Минусы выбранного подхода

1. Высокий порог входа
2. Микросервисы → более сложное проектирование
3. Не все реализовано для Docker
4. Много Release Definition

Плюсы подхода

1. Инфраструктура как код
2. Масштабирование функционала и производительности
3. Микросервисы очень хорошо изолированы
4. Быстрая доставка изменений

Выводы для нас

- На .NET Core можно и нужно реализовывать промышленные решения
- K8S действительно облегчил жизнь, упростил обновление сред, облегчает конфигурирование сервисов
- TFS можно использовать для реализации CI/CD для Linux