

Lab 6. Развертывание с помощью Kustomize

Упражнение 1.

Шаг 1. Создайте каталог проекта demo и в нем следующую структуру каталогов:

```
> mkdir -p {overlays/{dev,prod},base}
```

```
> touch {base/kustomization.yaml,overlays/{dev,prod}/kustomization.yaml}
```

demo

```
├─ base
|   └─ kustomization.yaml
└─ overlays
    ├─ dev
    |   └─ kustomization.yaml
    └─ prod
        └─ kustomization.yaml
```

Шаг 2. Определение образа контейнера

Сначала зададим версию образа контейнера. Для примера создадим простой манифест развертывания base/deploy.yaml:

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
  name: nginx-deployment
```

```
spec:
```

```
  selector:
```

```
    matchLabels:
```

```
      app: nginxdeployment
```

```
  replicas: 2
```

```
  template:
```

```
    metadata:
```

```
      labels:
```

Building Kubernetes Applications

```
app: nginxdeployment  
spec:  
  containers:  
  - name: nginxdeployment  
    image: nginx:1.25.0  
    ports:  
    - containerPort: 80
```

Шаг 3. В каталоге base файл kustomization.yaml позволяет изменить тег:

```
apiVersion: kustomize.config.k8s.io/v1beta1
```

```
kind: Kustomization
```

```
resources:
```

```
- deploy.yaml
```

```
images:
```

```
- name: nginx
```

```
  newTag: latest
```


Теперь, чтобы увидеть, как он анализируется, мы можем использовать команду:

kubectl kustomize base

Шаг 4. Произведите развертывание.

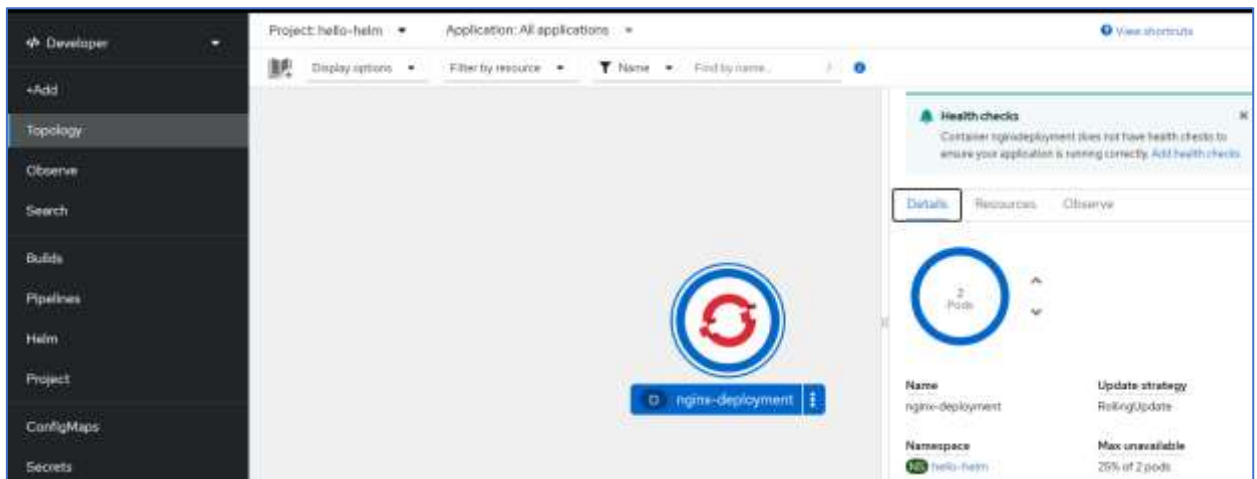
В командной строке перейдите в каталог проекта и произведите развертывание. Для этого выполните команду:

```
kubectl apply -k base
```



```
Windows PowerShell  
PS C:\Users\Michael\demo> kubectl apply -k base  
deployment.apps/nginx-deployment created  
PS C:\Users\Michael\demo>
```


Шаг 4. Перейдите в веб-консоль и в перспективе разработчика перейдите в раздел Topology



Упражнение 2. Patching: Использование ConfigMapGenerator

Kustomize предоставляет два способа добавления ConfigMap в одном файле kustomization: либо путем объявления ConfigMap в качестве ресурса, либо путем объявления ConfigMap из ConfigMapGenerator.

Прим. Формат файла kustomization.yaml имеет следующий вид

```
# declare ConfigMap as a resource
```

```
resources:
```

```
- configmap.yaml
```

```
# declare ConfigMap from a ConfigMapGenerator
```

```
configMapGenerator:
```

```
- name: a-configmap
```

```
files:
```

```
  # configfile is used as key
```

```
  - configs/configfile
```

```
  # configkey is used as key
```

```
  - configkey=configs/another_configfile
```

ConfigMaps, объявленные как ресурс, обрабатываются так же, как и другие ресурсы. Kustomize не добавляет хэш к имени ConfigMap. ConfigMap, объявленный из ConfigMapGenerator, обрабатывается по-разному. К имени добавляется хеш, и любое изменение в ConfigMap вызовет непрерывное обновление.

Подробнее: <https://github.com/kubernetes-sigs/kustomize/blob/master/examples/configGeneration.md>

Шаг 1. Рассмотрим пример с configMapGenerator

Добавим configMapGenerator в файл **base/kustomization.yaml**:

```
apiVersion: kustomize.config.k8s.io/v1beta1
```

```
kind: Kustomization
```

```
resources:
```

```
- deploy.yaml
```

```
images:
```

```
- name: nginx
```

```
  newTag: latest
```

```
configMapGenerator:
```

```
- name: config
```

```
  literals:
```

```
    - common="Common Variable"
```

И затем в **overlays/dev/kustomization.yaml**:

```
apiVersion: kustomize.config.k8s.io/v1beta1
```

```
kind: Kustomization
```

```
resources:
```

```
- ../../base
```

```
configMapGenerator:
```

```
- name: config
```

```
  behavior: merge
```

```
  literals:
```

```
    - custom="My Custom Config"
```

Шаг 2. Теперь можно изменить deployment в base, чтобы добавить common configMap:

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

Building Kubernetes Applications

name: nginx-deployment

spec:

selector:

matchLabels:

app: nginxdeployment

replicas: 2

template:

metadata:

labels:

app: nginxdeployment

spec:

containers:

- name: nginxdeployment

image: nginx:1.25.0

ports:

- containerPort: 80

env:

- name: **COMMON**

valueFrom:

configMapKeyRef:

name: config

key: common

Шаг 3. Делаем парсинг конфига:

kubectl kustomize overlays/dev

Упражнение 3. Patching: Использование ConfigMap.

Шаг 1. Сделаем постановку с помощью исправления, примененного к ConfigMap.

Добавим в overlays/prod/kustomization.yaml:

```
apiVersion: kustomize.config.k8s.io/v1beta1
```

```
kind: Kustomization
```

```
resources:
```

```
- ../../base
```

```
patchesStrategicMerge:
```

```
- map.yaml
```

И создадим файл overlays/prod/map.yaml следующего содержания:

```
apiVersion: v1
```

```
kind: ConfigMap
```

```
metadata:
```

```
  name: config
```

```
data:
```

```
  altGreeting: "Have a pineapple!"
```

```
  enableRisky: "true"
```

Шаг 2. Делаем парсинг конфига:

```
kubectl kustomize overlays/prod
```

Шаг 3. Развернем конфигурацию prod

```
kubectl apply -k ./overlays/prod/
```