Deploying the First Release

In this lesson, we will deploy our first release to get started with Namespaces.

WE'LL COVER THE FOLLOWING

- Looking into the Definition
- Altering the Definition
- Verification

Looking into the Definition

We'll start by deploying the `go-demo-2` application and use it to explore Namespaces.

```
cat ns/go-demo-2.yml
```

The **output** is as follows.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: go-demo-2
  annotations:
    kubernetes.io/ingress.class: "nginx"
    ingress.kubernetes.io/ssl-redirect: "false"
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
spec:
  rules:
  - host: go-demo-2.com
    http:
      paths:
      - path: /demo
        backend:
          serviceName: go-demo-2-api
          servicePort: 8080

---

apiVersion: apps/v1
kind: Deployment
```

```
metadata:
  name: go-demo-2-db
spec:
  selector:
    matchLabels:
      type: db
      service: go-demo-2
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        type: db
        service: go-demo-2
        vendor: MongoLabs
    spec:
      containers:
      - name: db
        image: mongo:3.3

---

apiVersion: v1
kind: Service
metadata:
  name: go-demo-2-db
spec:
  ports:
  - port: 27017
  selector:
    type: db
    service: go-demo-2

---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: go-demo-2-api
spec:
  replicas: 3
  selector:
    matchLabels:
      type: api
      service: go-demo-2
  template:
    metadata:
      labels:
        type: api
        service: go-demo-2
        language: go
    spec:
      containers:
      - name: api
        image: vfarcic/go-demo-2
        env:
        - name: DB
          value: go-demo-2-db
        readinessProbe:
          httpGet:
            path: /demo/hello
            port: 8080
          periodSeconds: 1
        livenessProbe:
          httpGet:
            path: /demo/hello
            port: 8080

---

apiVersion: v1
kind: Service
metadata:
  name: go-demo-2-api
spec:
  ports:
  - port: 8080
  selector:
    type: api
    service: go-demo-2
```

The definition is the same as the one we used before, so we'll skip the explanation of the YAML file. Instead, we'll jump right away into the deployment.

Altering the Definition

Unlike previous cases, we'll deploy a specific tag of the application. If this would be a Docker Swarm stack, we'd define the tag of the `vfarcic/go-demo-2` image as an environment variable with the default value set to `latest`.

Unfortunately, Kubernetes does not have that option. Since we don't believe that it is a good idea to create a different version of the YAML file for each release, we'll use `sed` to modify the definition before passing it to `kubectl`.

Using `sed` to alter Kubernetes definitions is not a good solution. We should use a templating solution like, for example, Helm. However, we are focusing purely on Kubernetes.

Helm and other third-party products are out of the scope of this course. So, we'll have to do with a workaround in the form of `sed` commands.

```
IMG=vfarcic/go-demo-2

TAG=1.0

cat ns/go-demo-2.yml \
    | sed -e \
    "s@image: $IMG@image: $IMG:$TAG@g" \
    | kubectl create -f -
```

- We declared environment variables `IMG` and `TAG`.
- We `cat` the YAML file and piped the output to `sed`. It, in return, replaced `image: vfarcic/go-demo-2` with `image: vfarcic/go-demo-2:1.0`.
- The modified definition was piped to `kubectl`.

> When the `-f` argument is followed with a dash (`-`), `kubectl` uses standard input (`stdin`) instead of a file. In our case, that input is the YAML definition altered by adding the specific tag (`1.0`) to the `vfarcic/go-demo-2` image.

Verification

Let's confirm that the deployment rolled out successfully.

```
kubectl rollout status \
    deploy go-demo-2-api
```

We'll check whether the application is deployed correctly by sending an HTTP request. Since the Ingress resource we just created has the `host` set to `go-demo-2.com`, we'll have to "fake" it by adding `Host: go-demo-2.com` header to the request.

```
curl -H "Host: go-demo-2.com" \
    "http://$(minikube ip)/demo/hello"
```

The **output** is as follows.

```
hello, release 1.0!
```

The reason we jumped through so many hoops to deploy a specific release will be revealed soon. For now, we'll assume that we're running the first release in production.

In the next lesson, we will explore the system level objects running inside a Kubernetes cluster.