Kubernetes Workshop

Wojciech Barczynski (wbarczynski.pro@gmail.com)

Contents

1 Prerequiments

You need to feel good with Command Line Interface. You should understand what Docker is.

- Workstation with Linux or OSX recommended.
- Software
  - k3s
  - Kubernetes CLI
  - Docker
- Tools
  - jq (stedolan.github.io/jq/)
- Good to have
  - hub.docker.com account or alternative docker repository

1.1 How to install

- K3S - github.com/k3s-io/k3s
- Kubernes CLI - kubernetes.io/docs/tasks/tools/

1.2 Verify the setup

```
$ k3d cluster create --port "8080:8080@loadbalancer" \
                     --port "8000:80@loadbalancer" \
                     'k8s-w10i-workshop'

$ kubectl config use-context k3d-k8s-w10i-workshop

$ kubectl cluster-info

Kubernetes control plane is running at https://0.0.0.0:60602
CoreDNS is running at https://...
Metrics-server is running at https://...
```

1.3 Kubernetes CLI Basics

Let's learn first some basics regarding the *kubectl*:

```
kubectl get <ARTIFACT>
kubectl describe <ARTIFACT>
```

From the kubect ref kubernetes.io/docs/reference/kubectl/overview:

```
kubectl [command] [TYPE] [NAME] [flags]
```

1. List the nodes (underlaying machines or virualmachines running k8s):

```
$ kubectl get nodes
```

What the names of our nodes are? . . .

2. Let's learn more about the node:

```
# the name of the node you saw from previous
# command
$ kubectl get nodes k3d-k8s-w10i-workshop-server-0
$ kubectl get nodes k3d-k8s-w10i-workshop-server-0 -o wide

# notice:
#   most of services have shortnames:
$ kubectl get no k3d-k8s-w10i-workshop-server-0
$ kubectl get node k3d-k8s-w10i-workshop-server-0
$ kubectl get nodes k3d-k8s-w10i-workshop-server-0
```

You can find the abbreviations with the command:

```
$ kubectl api-resources
```

3. Get more details:

```
$ kubectl describe nodes k3d-k8s-w10i-workshop-server-0
```

Note down:

- Container Runtime Version: . . .
- What the namespaces we have: . . .
- Note down name of two pods:
  - . . .
  - . . .

4. YAML and JSON output

```
$ kubectl get node k3d-k8s-w10i-workshop-server-0 -o yaml
$ kubectl get node k3d-k8s-w10i-workshop-server-0 -o json
```

Use *jq* to get the *kubeletVersion*, write down below:
. . .

5. Notice, `kubectl` provides support for jsonpath:

```
$ kubectl get node k3d-k8s-w10i-workshop-server-0 \
  -o jsonpath="{.status.daemonEndpoints.kubeletEndpoint.Port}"

$ kubectl get node k3d-k8s-w10i-workshop-server-0 \
  -o jsonpath="{.metadata.labels}"
```

Write down a command with jsonpath to get information on how many CPU we have allocated to our minikube:
. . .

6. All Kubernetes resources have labels attached:

```
# show me nodes that have the following label
$ kubectl get no  -l 'kubernetes.io/hostname'

# show me nodes running on linux
$ kubectl get no  -l 'kubernetes.io/os=linux'
```

Please find another label, you could select our node and run the command.

7. Recommendations for your local setup:

- `alias k=kubectl` or `alias kb=kubectl` (more ideas on github.com/prezto-contributions/prezto-kubectl)
- `kubectx` and `kubens` - github.com/ahmetb/kubectx

If you do not want to install kubectx, create an alias that will let you quickly check to which of kubernetes clusters, you are "connected":
`alias kctx='kubectl config current-context'`

2 Kubectl configuration file

Your kubectl configuration is in `${HOME}/.kube/config`, it contains tokens, certificates, aliases etc. You will need to edit this file very seldom.

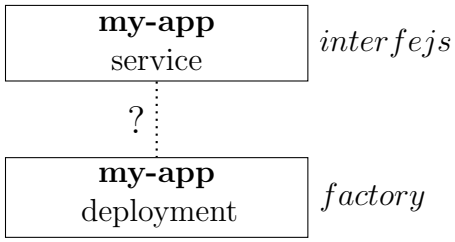
1. View `${HOME}/.kube/config`.

2. Find *certificate-authority*.

3. Note the main sections:
. . .
. . .
. . .

3 Task at Hand

Our goal today will be deployment of `intro-app` on Kubernetes. It is a simple web service. We will start with learning about the Kubernetes deployment and service as shown on the picture.



4 What are the namespaces?

Let us see first how to organize our apps in the Kubernetes with `namespaces`.

```
$ kubectl get ns
$ kubectl get namespaces
```

Notice:

- you can create namespaces to better organize your components
- you might define resource restrictions per namespaces
- namespace effects the service name:

  <service-name>.<namespace-name>.svc.cluster.local.

  More about it later.

To change the selected namespace for our commands:

```
$ kubectl config set-context \
  $(kubectl config current-context) \
  --namespace <namespace-name>
```

You can specify namespace explicitly with the kubectl CLI:

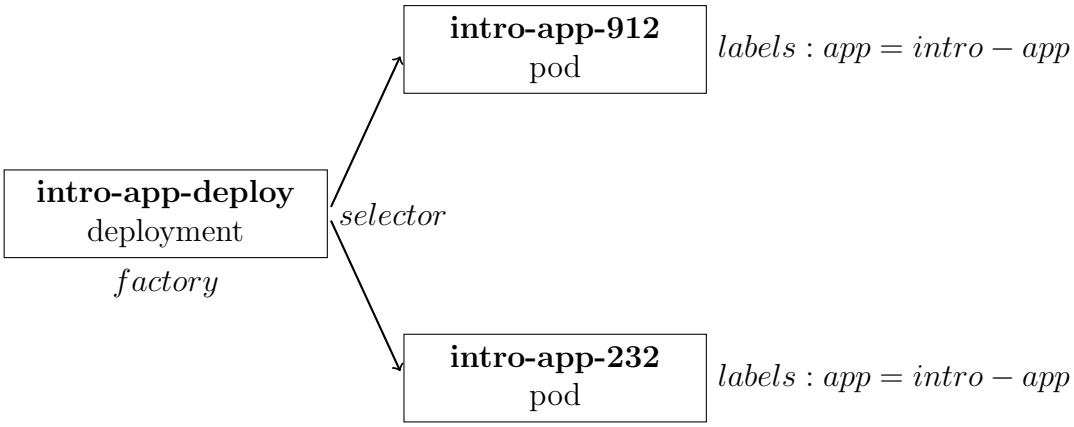
```
$ kubectl get pods --namespace=kube-system
$ kubectl get pods -n kube-system
$ kubectl get po -n default
```

Notice: you can check `kubectl api-resources` to see which resources are namespaced and which not.

5 Kubernetes deployment yaml

Let's get instances of our application running. We use an application docker image built on the top of official nginx (hub.docker.com/_/nginx), you will find Dockerfiles in `manifests/dockers`.

0. The Kubernetes deployment resource as a factory that creates pods based on a template definition. The deployment uses labels to "find" its pods. If any pod is missing, Kubernetes will schedule the missing number of pods.



Notice: `deployment` (apps/v1) uses `replicasets` (`k get rs`) under the hood.

1. Let's understand the deployment manifest `manifests/kube-deployment.yaml`(showing a minimal manifest):

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: intro-app-deploy
  labels:
    app_deploy: intro-app
spec:
  replicas: 1
  selector:
    matchLabels:
```

```
      app: intro-app
  template:
    metadata:
      labels:
        app: intro-app
    spec:
      containers:
      - name: app
        image: wojciech11/api-status:1.0.0
        env:
          - name: DB_NAME
            value: user
        ports:
        - containerPort: 80
```

Notice: the postfix `-deploy` is not the best practise, just to make it more explicit what-is-what during the training.

Hints:

- if your repo is private, you need to define `imagePullSecrets` (check official docs[1]).
- You can forge Kubernetes to pull Docker image every time with `imagePullPolicy: Always`, not recommended but might be helpful during the development. You MUST NOT use this setting in **production**.

2. Let's get our application running by creating the Kubernetes deployment that will start the app for us:

```
# Declarative object configuration (we use it in CI/CD)
$ kubectl apply -f manifests/kube-deployment.yaml
deployment.apps/intro-app-deploy created

# Imperative object configuration (alternative to the previous command):
$ kubectl create -f manifests/kube-deployment.yaml
```

[1] kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/

```
deployment.apps/intro-app-deploy created

# the command will delete and create new deployment
$ kubectl replace -f manifests/kube-deployment.yaml
```

there are also the imperative commands:

```
# create a deployment with nginx image
$ kubectl create deployment nginx --image nginx
```

If you want to know more about the difference, checks docs:

- Imperative object configuration - kubernetes.io/docs/concepts/overview/object-management-kubectl/imperative-config,
- Declarative object configuration - kubernetes.io/docs/concepts/overview/object-management-kubectl/declarative-config/.

Trade-offs between them - kubernetes.io/docs/concepts/overview/working-with-objects/object-management/.

Notice, `kubectl` has a diff command:

```
$ kubectl diff -R -f manifests/ingress
```

3. List deployments to see whether there is your deployment resource:

```
# deploy, deployment, deployments
$ kubectl get deploy

NAME               READY   UP-TO-DATE   AVAILABLE   AGE
intro-app-deploy   1/1     1            1           19s

$ kubectl get deploy -o wide
# notice the selector
```

4. Check the details about the Kubernetes deployment:

```
$ kubectl describe deploy <DEPLOYMENT_NAME>
```

Notice fields for update strategy (more about it later) and replicas.

5. Where is our app? The pods:

```
# "po" = "pod" or "pods"
$ kubectl get po
$ kubectl get po -n default

$ kubectl describe po <POD_NAME>
```

6. Find the following information:

- How many containers are in the pod? . . .
- What is the IP of your app pod? . . .
- What is ReplicatSet? . . .
- Ready? . . .
- Restart Count? . . .
- Events? . . .

Notice: we will discus other fields – QoS, Conditions (Ready), Node-Selector, and Tolerations – later.

7. Increase the number of pods, modify the YAML manifest and apply the changes. Check wherher you see more pods `kubectl get po`. If the answer is yes, please scale down your deployment to one pod instance.

8. What does it happen when you delete one of the pods?

```
$ kubectl delete po intro-app-65db4-...
$ kubectl get po
```

6 Connect to your app with port forwarding

We can see the app running but does it work? We did not expose our application outside the cluster, so we need another mechanism at this moment. Kubernetes provide us `port-forward` to forward port from the pod to local machine port.

1. Find the port our application listen on[2].

2. Setup the port forwarding:

```
kubectl port-forward <POD_NAME> <LOCAL_PORT_NUMBER>:<POD_PORT_NUMBER>

# let's choose 8080 as the local port
$ kubectl port-forward <POD_NAME> 8080:<POD_PORT_NUMBER>

# in a separate terminal:
$ curl 127.0.0.1:8080

<html>
<h1>1.0.0</h1>
</html>

# if you prefer httpie (httpie.io)
$ http 127.0.0.1:8080
```

You will use `port-forward` very often as one of the first steps of many debugging sessions for why-client-cannot-connect-to-my-service-from-outside.

The `port-forward` can take also as a parameter also Kubernetes service or deployment[3].

Let's learn about services and ingresses first, later we see how we can modify our deployment and update the application.

[2]Notice: the deployment manifest does not enforce the port number the application listen on.

[3]kubernetes.io/docs/tasks/access-application-cluster/port-forward-access-application-cluster

7 Get your application logs

Let's keep the port-forwarding running and open the second terminal to examinate logs:

```
$ kubectl logs <POD NAME>
```

```
$ kubectl logs <POD NAME> -f # -f as in tail -f ;)
```

Send few requests with `curl` to see new entries in the logs. You will use this command a lot, so check the command help `k logs --help`.

8 Opening console in your container

Sometimes, you would like to check your appliacation from within the container. Let's see how we can achieve it with Kubernetes

1. Get the console:

```
# check the pod name
$ kubectl exec -it intro-app-... -- /bin/bash
```

Let's print env variables:

```
$ kubectl exec -it intro-app-65d /bin/bash

root@intro#$ printenv

root@intro#$ printenv | grep DB
```

2. Add tool for debugging - curl:

```
root@intro#$ apt-get update && apt-get install -qq curl
```

3. Does it work?

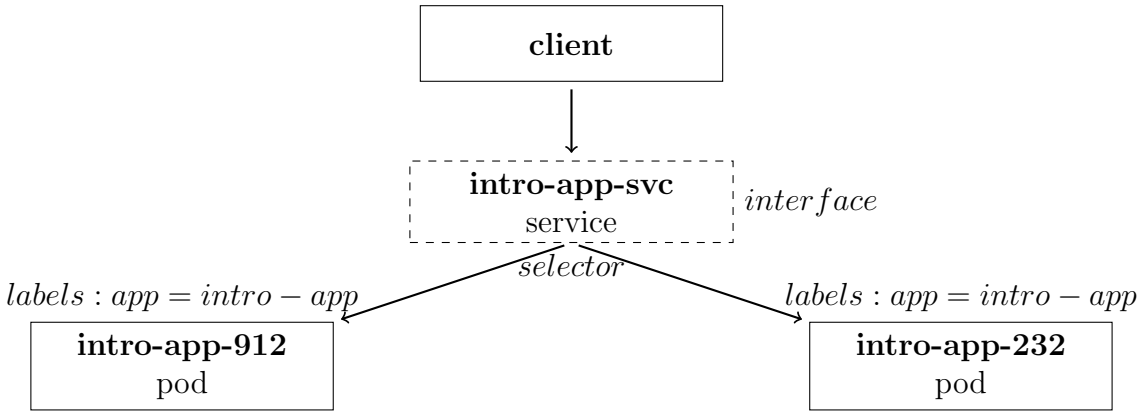
```
# does it work?
root@intro#$ curl 127.0.0.1

# can we get outside
root@intro#$ curl -I wbarczynski.pl

# can we reach other services:
root@intro#$ telnet kube-dns.kube-system 53
```

4. Assuming we resolve our issue, let's clean up by deleting the pod we install utilites for debugging:

```
$ kubectl delete po intro-app-65db4-...
```

9 Kubernetes Service

Our factory, I mean the deployment defines how we create our application instanes as pods. The service, how we expose it to be consumed. We have three types of Services: LoadBalancer, ClusterIP (the most commonly used), NodePort, and ExternalName (CNAME to an external service).



1. Let's go through the (pretty basic) manifest `manifests/kube-service.yaml` (again the *-svc* prefix for the clarity):

```
apiVersion: v1
kind: Service
metadata:
  name: intro-app-svc
  labels:
    me: wojtek
spec:
  ports:
  - port: 8080
    targetPort: 80
    protocol: TCP
    name: http
  selector:
    app: intro-app
  type: LoadBalancer
```

2. Deploy:

```
$ kubectl create -f manifests/kube-service.yaml
$ kubectl apply -f manifests/kube-service.yaml
```

3. Let's call our service through loadbalancer that we exposed on `8080`:

```
# http 127.0.0.1:8080
$ curl -s -D - 127.0.0.1:8080
HTTP/1.1 200 OK
Server...

<html>
<h1>1.0.0</h1>
</html>
```

Notice: on AWS, Azure, or GCP, we would get the loadbalancer created and public IP assigned. You would than use annotations to specify the loadbalancer configuration, for example: docs.aws.amazon.com/eks/latest/userguide/network-load-balancing.html.

4. Let's list the services and get more details about our newly created service:

```
$ kubectl get services
$ kubectl get svc -o wide
$ kubectl describe svc intro-app-svc
```

Please note down:

- Endpoints (where this IP comes from?) . . .
- Selector . . .
- IP . . .

5. Short recap with the trainer - service types:

- ClusterIP with and without IP (headless)
- LoadBalanced

- ExternalName

6. How does the service work? Let's use busybox[4] docker to see how we can access the service from a different app.

```
$ kubectl run busybox --image=busybox:1.28.4 --rm \
    --restart=OnFailure -ti -- /bin/nslookup intro-app-svc

$ kubectl run -i --tty busybox-wget --rm \
    --image=busybox:1.28.4 -- sh

# what the port is?
/# wget -O- intro-app-svc:YOUR_SRV_PORT
/# wget -O- intro-app-svc.default
/# wget -O- intro-app-svc.default.svc

/# wget -O- traefik.kube-system
```

By the way, the service info is also injected as environment variables. I did not have so far a need to use this information:

```
$ kubectl run busybox --image=busybox \
  --rm --restart=OnFailure -ti -- printenv | grep -i intro_app_svc
```

You could also run nslookup from within our app pods:

```
$ kubectl exec -it intro-app-deploy-5d556d9f4b-vslp9 \
    -- /bin/bash

/# apt-get update && apt-get install dnsutils -qq
/# nslookup intro-app-svc
```

7. You will use ClusterIP service with Ingress controller more often than Loadbalancer service.

[4]"The Swiss Army Knife of Embedded Linux" - `hub.docker.com/_/busybox`

10 Modyfing kubernetes deployment and service

Avoid editing files on kubernetes, always modify a yaml and apply the changes.

1. Change the number of pods running to 2 with:

```
$ kubectl edit deploy YOUR_DEPLOYMENT
```

```
$ kubectl get po
```

2. Change the value of label `me` to your name in the service definition.

3. Modify the `kube-deployment.yml` to get 3 pods, use: `kubectl apply -f`

4. Add one more label to service.

5. What does happen if we add one more selector, apply it:

```
apiVersion: v1
kind: Service
metadata:
  name: intro-app-svc
  labels:
    me: wojtek
spec:
  ports:
  - port: 8080
    targetPort: 80
    protocol: TCP
  selector:
    app: intro-app
    break: the-connection-with-pods
  type: LoadBalancer
```

Can we connect?

```
# again, let's connect through LB
$ curl -s -D - 127.0.0.1:8080
```

What has changed?

```
$ kubectl describe svc intro-app-svc
```

Notice: very very very common issue that selectors do not match labels.

6. Fix your service.

11 Updating service

Let's update our app from the version `1.0.0` to `2.0.0`:

1. Change in the `kube-deployment.yaml` and apply changes.

2. You can also change it with set image setting the image by the CLI[5]:

```
$ kubectl set image deployment/<DEPLOYMENT_NAME> \
  <CONTAINER_NAME>=<DOCKER_IMAGE_NAME>:<VERSION>
```

3. Change two times from `1.0.0` to `2.0.0` and back:

```
$ curl -s -D - 127.0.0.1:8080
```

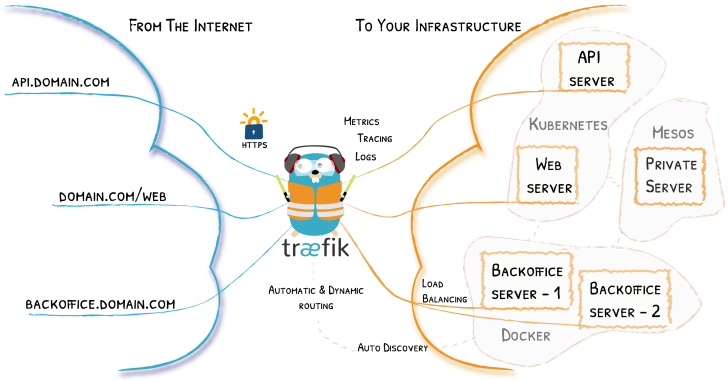
4. Let's scale it up:

```
$ kubectl scale --replicas=2 deployment/intro-app-deploy
```

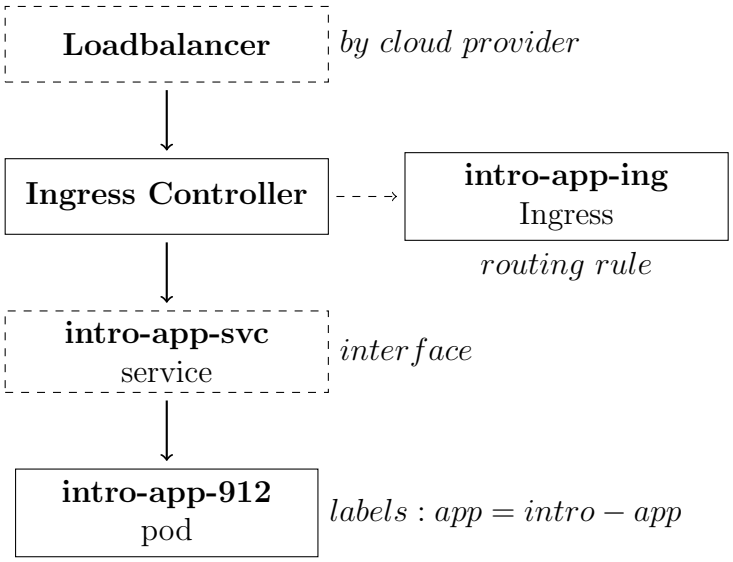
[5] hint: `$ kubectl get deploy -o wide`

12 Kubernetes Ingress

The purpose of the ingress controller is to provide reverse proxy service to hide details how our application is implemented:



In most of scenarios, you will use the ingress controller, so we will have:



1. Let's set up the neccessary configuration for already installed `traefik` in the cluster:

```
# let's remote the previous service definition
$ kubectl delete -f manifests/kube-service.yaml
```

```
# check the files before applying
$ kubectl apply -f  manifests/ingress
```

2. Let go over the `manifest/ingress/kube-ingress.yaml`:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: intro-app-ing
  annotations:
    kubernetes.io/ingress.class: traefik
    traefik.ingress.kubernetes.io/router.middlewares:
    ↪  default-intro-app-ing@kubernetescrd
spec:
  rules:
  - host: my.app
    http:
      paths:
        - path: /echo
          pathType: Prefix
          backend:
            service:
              name: intro-app-svc
              port:
                number: 80
```

3. Now, it is time to call our service. You can print the logs from your pod to verify that the requests are reaching your pod. Let's access our service as our customers would do:

```
$ curl --header 'Host: my.app' http://127.0.0.1:8000/echo
```

4. Now, after verifing that our ingress is working, let's look closer into the ingress:

```
# "ing" or "ingress"
$ kubectl get ing
$ kubectl describe ing <ingress name>
```

5. Let's open a dashboard of our ingress controller - `traefik`:

```
$ kubectl get po -n kube-system
$ kubeclt get po -l 'app.kubernetes.io/instance=traefik'
$ kubectl describe po traefik-97b44b794-tf5kf -n kube-system
k port-forward -n kube-system traefik-97b44b794-tf5kf 9000:9000
```

Open in your browser: http://127.0.0.1:9000/dashboard/, choose `HTTP`, and select our rule from the list.

13 Containers vs Pods

Please answer the following questions:

- How many containers can a Pod has?
- Do containers share disk?
- Do container share port space?
- What does `1/1` mean in the output of `kubectl get po`?

14 Fail-over

Let's see what happens when our application crashes.

1. Open console.

2. Force restart:

```
# should work
kill 1

# always works
kill -9 1
```

Repeat 5 times. Observer the output from: `kubectl get po`.

15 How to debug in nutshell

Good to ship a minimum of debugging tools in your container, such as, curl or telnet.
Happy debugging path:

```
$ kubectl describe ing
$ kubectl describe svc
$ kubectl exec -it <pod name> /bin/bash

# curl, telnet, ...
$ kubectl describe po <pod name>

$ kubectl logs <pod name>
$ kubectl logs <pod name> -f
$ kubectl logs <pod name> --tail=100

$ kubectl logs -n kube-system <pod for your ingress controller>

$ kubectl get events
```

Notice: To improve the observability, start with monitoring (e.g., Prometheus) and Kubernetes probes (we will cover them later).

16 Kubernetes configmap

With configmaps, we can deliver values for environment variables or files. In our scenario, we will change the index.hml that we server from our nginx:

1. Copy `index.html`:

```
$ cp manifests/dockers/site-1.0.0/index.html index.html
```

2. Edit `index.html` Add your name after the version number:

```
<html>
<h1>1.0.0-Natalia</h1>
</html>
```

3. Let's create a configmap:

```
$ kubectl create cm intro-app-index-html --from-file index.html
```

4. Check the commands:

```
# "cm" "configmap"
$ kubectl describe cm intro-app-index-html
$ kubectl get cm intro-app-index-html -o yaml
```

5. To make our new index file available, we need to mount it:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: intro-app-deploy
  labels:
    app_deploy: intro-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: intro-app
  template:
    metadata:
      labels:
        app: intro-app
    spec:
      containers:
      - name: app
        image: wojciech11/api-status:1.0.0
        ports:
        - containerPort: 80
        volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: "html-content"
      volumes:
      - name: html-content
        configMap:
          name: intro-app-index-html
```

You can modify your kube-deployment.yaml in the manifest directory or:

```
$ kubectl get -o yaml deploy intro-app-deploy > tmp_deployment.yaml
# edit tmp_deployment.yaml
$ kubectl apply -f tmp_deployment.yaml
```

5. Let's do a smoke test:

```
$ curl --header 'Host: my.app'  "http://127.0.0.1:8000/echo"
```

6. We can also set environment values, let's create new configmap:

```
$ kubectl create configmap intro-app \
  --from-literal=db.name=mydb
```

7. .. and use it:

```
        env:
        - name: DB_NAME
          valueFrom:
            configMapKeyRef:
              name: intro-app
              key: db.name
```

8. Open a console in your pods and check whether the ENV variable is set:

```
\# printenv | grep DB_NAME
```

9. You may also declare all of your env variables in configmap and load all of them:

```
      envFrom:
      - configMapRef:
          name: db-config
```

and the corresponding configmap:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: db-config
  namespace: default
data:
  DB_NAME: mydb
  DB_USERNAME: myuser
```

also:

```
$ kubectl create configmap intro-app \
    --from-literal=DB_NAME=mydb \
    --from-literal=DB_USERNAME=myuser
```

Recomendation: Keep it simple, start with env values hardcoded in deployment if the values do not change between the environments.

17 Kubernetes secret

Secrets are very similar to configmaps. They provide better security (kind-of) than configmaps.

1. Create a secret with database password:

```
$ kubectl create secret generic intro-app-secret \
  --from-literal="db.password=nomoresecrets"

# what changed?
$ kubectl get secret intro-app-secret -o yaml
```

Hint: below[6]

2. Bind it to an environment variable in the deployment:

```
env:
  - name: DB_PASSWORD
    valueFrom:
      secretKeyRef:
        name: intro-app-secret
        key: db.password
```

3. Please deliver `cert.crt` to your application and mount it at `/usr/secret`, find how to do it find on kubernetes.io/docs/concepts/configuration/secret/ :

```
$ echo "CERT" > cert.crt
$ kubectl create secret generic intro-app-cert \
    --from-file cert.crt
```

[6] `printf nomoresecrets | base64`

18 Your own app

We have prepared two simple application in Python and Golang, your task is to deploy one of them on the Kubernetes cluster. Let's use docker hub for hosting our image. For this purpose please create an user on hub.docker.com.

Before please delete all the resources from the previous exercises:

```
# find resources we created so far and delete them:
$ kubectl get ingress
$ kubectl get svc
$ kubectl get deploy
$ kubectl get cm
$ kubectl get secret
```

Chose one applications (see in the `exercise` directory):

- Python
- Golang

Create, following the instruction from the trainer, Kubernetes resources to run the app[7]:

- Deployment with MGO_USERNAME as an env variable
- ConfigMap with a field MGO_NAME and MGO_PORT
- Secret for MGO_PASSWORD
- Service
- Ingress: path `/hello` to `/`
- Ingress: path `/hostname` to `/hostname`

Please test whether you can call your application:

```
$ curl "http://127.0.0.1:8000/hello"
$ curl "http://127.0.0.1:8000/hostname"
```

[7]Probably we should use here the DSN (Data Source Name) to define the connectio to the database. What are challenges of using DSN in K8S?

19 Quickstart templating

We will cover CI/CD in a separate workshop, here is a quick start. Let's learn about the most popular approaches

1. envsubst or other general templating solution
2. Helm
3. kustomize

1. envsubst or other general templating solution. approach. Let's imagine, we want to generate ingress resources with differenet host names depending on the environment.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: login-app
spec:
  rules:
  - host: ${HOST}
    http:
      paths:
        - path: /login
          backend:
            serviceName: login-app
            servicePort: 80
```

and the command:

```
$ export HOST=example.com
$ envsubst < my-k8s.tmpl.yaml > my-k8s.yaml
```

Let's see how it works. Please compare first `templating/simple/` with `templating/vanilla/` (together with your instructor), and follow the instructions from `README.md` in the `simple/` directory.

2. Helm[8] as a templating engine.
Please compare:

- `templating/simple/`
- `templating/helm/`

and follow the instructions from `README.md` in the `helm/` directory.

```
$ cd templating/helm/deployment

$ export APP_NAME=my-app
$ export APP_VERSION=snapshot-9911

$ helm template "${APP_NAME}" deployment \
   --set image.tag="${APP_VERSION}" \
   --version "${APP_VERSION}"

# in CI/CD, dry run for safety :)
$ helm template "${APP_NAME}" deployment \
   --set image.tag="${APP_VERSION}" \
   --version "${APP_VERSION}" | kubectl apply -f - --dry-run=server

# in CI/CD, again --dry-run for safety
$ helm install "${APP_NAME}" deployment \
   --set image.tag="${APP_VERSION}" \
   --version "${APP_VERSION}" --dry-run --debug
```

[8]Notice: Helm has functionality beyond just templating.

2. kustomize - overlay:

```
.
├── base
│   ├── kube-deployment.yaml
│   ├── kube-service.yaml
│   └── kustomization.yaml
├── dev
│   ├── image.yaml
│   ├── kustomization.yaml
│   └── scale.yaml
├── production
│   ├── image.yaml
│   ├── kustomization.yaml
│   └── scale.yaml
└── staging
    ├── image.yaml
    ├── kustomization.yaml
    └── scale.yaml
```

and the commands:

```
$ cd manifest/kustomize/kubernetes

$ export ENV_NAME=dev
$ kubectl kustomize ${ENV_NAME}

$ kubectl kustomize staging
$ kubectl kustomize production
```

You will find a complete example in `templating/kustomize`. Please follow `README.md`.

Lean towards:

1. If it is not going to change, do not template it.
2. Push as many setting as possible to the app repo as possible.
3. You can go very far, even if you create your Kubernetes secrets manually in the beginning[9].

[9] gitcrypt or a script that fetches secrets from your secret manager of choice.

20 Volumes

We will talk more about volumes in the training (02), during the discussion about `Statefulsets`[10]. If you need an cache for your application, you can use the `emptyDir` (kubernetes.io/docs/concepts/storage/volumes/#emptydir):

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
  volumes:
  - name: cache-volume
    emptyDir: {}
```

21 Next

- Liveness/Readiness probes - github.com/wojciech12/talk_zero_downtime_deployment_with_kubernetes
- Resource, Limits and QoS
- RBAC

22 Outlook

What could be the next steps in learning k8s.
What you could learn next.
Next course *Intermediate (Developer)*:

[10] `https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/`

1. Liveness/Readiness probes
2. Monitoring with Prometheus
3. Resource and Limits, QoS for your pods, schedule policies
4. Statefulsets
5. DaemonSets
6. Taints and Tolerations
7. Node affinity

Observability plus:

1. Monitoring
2. Logging
3. Traceability

Selected projects from CloudNative/K8S community:

1. ArgoCD
2. Istio
3. conftest - www.conftest.dev

Advance (Developer):

1. Zero-downtime deployment strategies
2. Horizontal scaling (beta: vertical pod scaling for the pets)
3. Continuous Deployment and Integration with Gitlab or other tool

Network and Security:

1. RBAC deep dive
2. Networking - Internal Loadbalancing - https://kubernetes.io/docs/concepts/services-networking/
3. Restricting Egress/Ingress with Network Policies

Kubernetes customization

1. Write your first CRD
2. Operators
3. Plugins to kubectl

CloudNative Ecosystem

1. Observability: Prometheus stack
2. Observability: EFK
3. Observability: Tracing
4. Ingress Controllers: Traefik, ... , standard and controller-specific annotations
5. Cert-manager
6. Operators for etcd and Vault

More

1. Operators for ...
2. Kubeless

Optionals

1. Google Kubernetes Engine - GKE
2. Azure Kubernetes Service - AKS
3. Amazon Elastic Kubernetes - EKS

Trainings and Consultancy: `wbarczynski.pro@gmail.com`