T6 - DevOps

T-DOP-600

DevOps - Kubernetes

Bootstrap

1.2

EPITECH.

DevOps - Kubernetes

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.
- Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

+ Step 0: Install

- Virtualbox or VMware
- Docker:
  - Docker: https://docs.docker.com/install/linux/docker-ce/fedora/
  - Docker Machine: https://docs.docker.com/machine/install-machine/, Minikube dependency
- Kubectl:
  - CLI for sending commands to k8s
  - https://kubernetes.io/docs/tasks/tools/install-kubectl/
  - awesome bash autocompletion: `$ kubectl completion bash`
- Minikube:
  - a tool for starting a 1 node k8s cluster, locally
  - https://kubernetes.io/docs/tasks/tools/install-minikube/

+ Step 0: About minikube

Minikube start a VM into VirtualBox (by default) and installs a Kubernetes cluster inside.
Start cluster:

```
minikube start

minikube status
minikube ip

# only the first time:
kubectl config current-context
kubectl config use-context minikube
```

Delete cluster:

```
minikube stop
minikube delete
```

+ Step 0: k8s vocabulary

- kubernetes nodes:
  - host, machine, vm, bare-metal…
  - `$ kubectl get nodes`
- kubernetes namespaces:
  - separated environments
  - `$ kubectl get namespaces`
  - at least 2 namespace on a fresh new cluster: `default`, `kube-system`
- kubernetes pods:
  - container or group of container
  - `$ kubectl get pods`
  - `$ kubectl get pods --namespace=kube-system`

+ Step 1: Create a pod

Create a pod with the following Docker image: `samber/hello-world-nodejs` (https://hub.docker.com/r/samber/hello-world-nodejs).

```
# hello-world.pod.yaml

apiVersion: v1
kind: Pod
metadata:
  name: hello-world
spec:
  containers:
  - name: hello-world
    image: samber/hello-world-nodejs
```

Terminal

```
~/T-DOP-600> kubectl apply -f hello-world.pod.yaml
```

Then wait few seconds for container deployment.

Then:

- Can you list existing pods in your k8s cluster ?
- Can you print more details about this pod ?
- Can you fetch and follow the container logs ?
- Can you execute the 'date' command inside the running container ?
- Can you delete this pod ?

+ Step 2: environment variables

- Can you update `hello-world.pod.yaml` and add to the container the following environment variable: `PORT=8080` ?
- Delete pod and apply again the new configuration.
- Check if it worked:
  - execute `env` inside container
  - or `kubectl describe pod <pod-id>`

+ Step 3: Expose ports

This small application is a web server. Can you ask Kubernetes to expose the port 8080 ?
Delete and apply `hello-world.pod.yaml` again ;)

This port has been exposed only inside the container. To test your configuration, execute:

```
Terminal
~/T-DOP-600> kubectl port-forward [pod-id] 8080:8080
```

```
Terminal
~/T-DOP-600> curl localhost:8080/
```

Port-forward command built a tunnel between your computer and Kubernetes cluster.

- Can you print full configuration of the container with `kubectl describe` ?
- Do you see the exposed port ?
- Do you see the container IP ?

+ Step 4: Create an internal DNS

Thanks to the previous step, other services can request our HTTP server on :8080.
But if your container dies, IP address will change. DNS are much better!

Create a k8s "Service" of type `ClusterIP`, linked to the `hello-world` app.

```
Terminal
~/T-DOP-600> kubectl apply -f hello-world.service.yaml
```

If you print informations of the created services, you might see the container IP in the `Endpoint` field.

```
Terminal
~/T-DOP-600> kubectl describe service [service-name] | grep Endpoint
```

Now, you should be able to ping your service from everywhere in the Kubernetes cluster:

```
Terminal
~/T-DOP-600> ping [service-name].[namespace].svc.cluster.local
```

+ Step 5: Volumes

Can you attach a 512MB disk volume to the `hello-world` container ?

+ Step 6: From pods to deployments

In the future, you will use k8s `deployments` instead of `pods`.

Deployments offer more features, such as:

- Replication policies
- Deployment history (and rollback!)
- Auto-scaling rules
- …

Can you convert your `hello-world.pod.yaml` into a deployment ?

+ TIPS AND TRICKS

You can get a full web UI for kubernetes with:

```
Terminal
~/T-DOP-600> minikube dashboard
```

Most used `kubectl` commands:

```
Terminal
~/T-DOP-600> kubectl apply -f [my-yaml-file]
```

```
Terminal
~/T-DOP-600> kubectl get [resource-type]
```

```
Terminal
~/T-DOP-600> kubectl describe [resource-type] [resource-id]
```

```
Terminal
~/T-DOP-600> kubectl delete [resource-type] [resource-id]
```

```
Terminal
~/T-DOP-600> kubectl port-forward [service-id] 8080:8080
```

Kubectl auto-completion is awesome:

- `kubectl [tab]`
- `kubectl describe [tab]`
- `kubectl describe pod [tab]`
- `kubectl describe pod hello-world`
- `kubectl get pods --namespace [tab]`

Useful 3rd-party tools:

- kubectx
- kubens

+ Troubleshooting with k8s

Get info about existing resource:

```
Terminal
~/T-DOP-600> kubectl describe [resource-id]
```

Open a shell into a running container:

```
Terminal
~/T-DOP-600> kubectl exec -it [resource-id] -c [container-name] sh
```

Test network from a running container:

```
Terminal
~/T-DOP-600> kubectl exec -it [resource-id] -c [container-name] - - ping
my-service.default.svc.cluster.local
```

```
Terminal
~/T-DOP-600> kubectl exec -it [resource-id] -c [container-name] - - curl
my-service.default.svc.cluster.local:8080/
```

Test port:

By default, exposed ports are available only inside k8s cluster.
So we need to build a tunnel.

```
Terminal
~/T-DOP-600> kubectl port-forward [resource-id] 8080:80
```

From your linux:

```
Terminal
~/T-DOP-600> curl localhost:8080/
```

Port-forward will never, ever, replace a VPN between 2 datacenters: only for debugging!

Check link between k8s service and pods

```
~/T-DOP-600> kubectl describe service [service-id] | grep Endpoint
```

RTFM

https://kubernetes.io/docs/concepts/