Lab Kubernetes

Introduction and Prerequisites

This laboratory is to

- Learn usage of Kubernetes through the command-line tool kubectl
- Exercise to define, deploy, run and scale web-application using pods, services and deployments
- Verify elasticity and resilience features of the Container management functionality

In this lab, you will first gain access to a Kubernetes cluster through the kubectl command line tool. The Kubernetes cluster as well as the command line tool is part of an image available to you on our OpenStack cluster.

- Name of the image: CCP1-EN-Kube-Lab_microk8s_minikube

The first task is to deploy a Redis key-value store service provided as example. The goal of the first part is to deploy a complete version of the application (Frontend + API + Redis) using pods. The second part of the lab will require you to make the application resilient to failures. You will deploy multiple pods with a Deployment for the Frontend and API services.

The following resources and tools are required for this laboratory session:

- Access to Ned cluster
- Any modern web browser

Time

The entire session will take 90 minutes.

Task 0 - Preparing your Kubernetes Cluster

In addition to a fully fledged Kubernetes (aka k8s) installation, there are some smaller versions meant for proof of concepts (PoCs) and development. The most well known are microk8s and minikube. Both versions are provided in the lab image but only microk8s is supported for the tasks below.

Start a VPN session to ZHAW and access the ZHAW Cloud called “Ned”. Deploy a lab VM instance from the preconfigured Image “CCP1-EN-KUBE-Lab-microk8s-minicube”. Note this for configuring your lab instance:

- Follow the instructions on how to use Ned (see Moodle)
- Use at least flavor m1.medium
- Remember to configure a key for SSH access
- Associate a floating IP
- Open up port `30080` for TCP connections by adding a security group to the machine
- SSH into the machine with the key you configured, e.g.: `ssh -i “path-to-your-ssh-key” ubuntu@160.85.37.155`
- Start your Kubernetes cluster with: `ubuntu@kube-lab-instance:~$ microk8s.start`
- In case of “`error: snap "microk8s" has "auto-refresh" change in progress`” wait until the auto-refresh process is completed.
- Verify your single-node cluster with the following command: `microk8s.kubectl cluster-info`
- Output on the terminal:
  ```
  Kubernetes control plane is running at https://...
  CoreDNS is running at https://...
  ```

The code templates required for the lab are in the folder `/home/ubuntu/kube`. In the following, the term “kubectl” is referring to the `microk8s.kubectl` command.
Hint: you could also create an alias in your Linux terminal, e.g. for bash add the following line to ~/.bashrc `alias kubectl='microk8s.kubectl'` and source your config by `$ source ~/.bashrc`

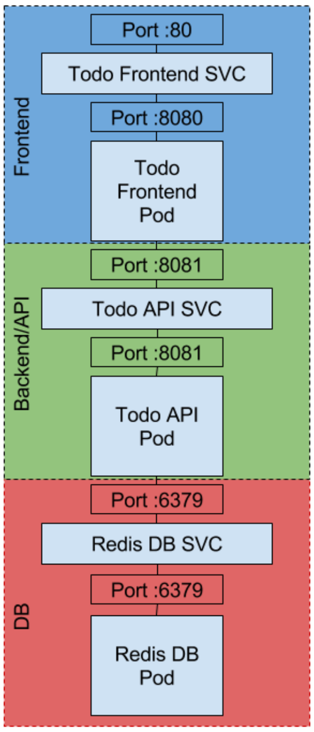
Task 1 – Deploy the application

In this task, you will set up the environment and deploy an example application to the Kubernetes cluster.

The goal of the provided example application is to store and edit a ToDo list. The application consists of a simple Web UI (Frontend) that uses a REST API (API service) to access a Key Value storage service (Redis). Ports to be used for pods and services are shown in the figure.

The required docker images are provided on Docker hub:

- Frontend: icclabcna/ccp2-k8s-todo-frontend
- API backend: icclabcna/ccp2-k8s-todo-api
- Redis: redis:6



Create the configuration and deploy the three tiers of the application to the Kubernetes cluster.

Deploy the Redis service and pod

Use the following commands to deploy Redis using the provided configuration files:

```
kubectl create -f redis-svc.yaml
kubectl create -f redis-pod.yaml
```

and verify it is up and running and on which ports using the command `kubectl get all`

Deploy the ToDo-API service and pod

Using the `redis-svc.yaml` file as an example and information from `api-pod.yaml`, create the `api-svc.yaml` configuration file for the API service. The service has to expose port 8081 and connect to the port of the API pod.

- Deploy and verify the API-service and pod (similar to the Redis ones) and verify that they are up and running on the correct ports.

Deploy the ToDo-Frontend service

Using the `redis-svc.yaml` file as an example, create the `frontend-svc.yaml` configuration file for the Frontend service.
Note, unlike the Redis and API services, the Frontend needs to be accessible from outside the Kubernetes cluster as a web server **on port 30080 of the host** (i.e., the VM).

- What needs to be different?
- Deploy the service using `kubectl`
- Make sure you have opened the `30080` port on the VM

Deploy the Frontend Pod

Using the `api-pod.yaml` file as an example, create the `frontend-pod.yaml` configuration file that starts the frontend docker container in a pod.

- Docker image for frontend container on Docker Hub is `icclabcna/ccp2-k8s-todo-frontend`

Note that the container runs on port: 8080
It also needs to be initialized with the following `ENVIRONMENT VARIABLES` (check how `api-pod.yaml` defines environment variables):

- `API_ENDPOINT_URL`: URL where the API can be accessed, for instance like http://localhost:9000
    - What value must be set for this URL?
      Hint: remember that anything you define as a service will be assigned a cluster-wide addressable DOMAIN and a PORT.
        - `api-svc.default.svc.cluster.local:8081 (cluster internal)`
- Deploy the pod using kubectl.

Verify the ToDo application

Now you can verify if the todo application is working correctly.

- What will be the public URL of your frontend services? (remember you have created a service that should listen on port 30080 of your VM)
- Access the public URL of the service with a browser. You should be able to access the complete application.

Troubleshooting

Several things can be misconfigured. Remember that there are two service dependencies:

- the Frontend forwarding requests to the (not externally accessible) API service
- The API service accessing the Redis service (also only accessible from within the cluster)

You can look at the pods logs to find out where the problem is.
A handy way to debug is to login to a container and see whether the required services are addressable.

You can see the logs of a pod using:

```
kubectl logs -f pod_name
```

You can run a command on a (container in a) pod using:

```
kubectl exec -it pod_name <command>
```

E.g. `kubectl exec -it pod-name bash` to start a bash inside the container.

Task 2 - Add and exercise resilience

By now you should have understood the general principle of configuring, running and accessing applications in Kubernetes. However, the above application has no support for resilience. If a container (resp. pod) dies, it stops working. Next, we add some resilience to the application.

Subtask – Add Deployments

In this task you will create Deployments that will spawn Replica Sets as Health-Management components.
Converting a Pod to be managed by a Deployment is quite simple.

- Have a look at an example of Deployment described here: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/
- Create Deployment versions of your application configurations (e.g. redis-deploy.yaml instead of redis-pod.yaml) and modify/extend them to contain the required Deployment parameters.
  - Ensure that 2 instances of the API and Frontend are running at any time.
  - Use only 1 instance for the Redis-Server. Why?
- Delete all application pods (using `kubectl delete pod ...`) and replace them with deployment versions.
- Verify that the application is still working, and the Replica Sets are in place. (kubectl get all, kubectl get pods, kubectl describe ...)

Subtask – Verify the functionality of the Replica Sets

In this subtask you will intentionally kill (delete) pods and verify that the application keeps working and the replica set is doing its task.

Hint: You can monitor the status of a component by repeatedly executing `kubectl describe rs <rs-name>` or use `kubectl get all` to see a list of rs/pods. You should also verify if the application stays available by continuously reloading your browser window.

- What happens if you delete a Frontend or API pod? How long does it take for the system to react?
- What happens when you delete the Redis pod?
- How can you change the number of instances temporarily to 3? Hint: look for scaling in the deployment documentation.
- What autoscaling features are available? Which metrics are used?
- How can you update a component? (see `update` in the deployment documentation)

Cleanup

At the end of the lab session:

- Delete all the pods, deployments and services of your group
- If no longer needed, halt your instance and disassociate your Floating IP for further use

Additional Documentation

Microk8s

- MicroK8s is the smallest, fastest, fully-conformant Kubernetes that tracks upstream releases and makes clustering trivial. MicroK8s is great for offline development, prototyping, and testing. Use it on a VM as a small, cheap, reliable k8s for CI/CD. It’s also the best production grade Kubernetes for appliances. Develop IoT apps for k8s and deploy them to MicroK8s on your Linux boxes.
- https://microk8s.io/docs

Minikube

- minikube is local Kubernetes, focusing on making it easy to learn and develop for Kubernetes.
- https://minikube.sigs.k8s.io/docs/start/

General Kubernetes documentation can be found on the following pages:

- User Guide explaining the main concepts: https://kubernetes.io/docs/user-guide/
- What is a Pod: https://kubernetes.io/docs/concepts/workloads/pods/pod/
- What is a Service: https://kubernetes.io/docs/concepts/services-networking/service/
- “Publishing services - service types” https://kubernetes.io/docs/concepts/services-networking/service/#publishing-services-service-types
- What is a Deployment: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/
- HowTo define environment variables: https://kubernetes.io/docs/tasks/configure-pod-container/define-environment-variable-container/
- How-To’s for do typical tasks in kubernetes: https://kubernetes.io/docs/tasks/
- Interactive kubectl command reference: https://kubernetes.io/docs/user-guide/kubectl