Lab ARCH – Lab Environment Setup

Introduction and Prerequisites

In this exercise, you will learn how to set up and use the laboratory infrastructure and environment used in the coming lab exercises. During the course, we will be using Kubernetes as the foundation in many configurations and different setups.

A flexible environment to quickly spin up preconfigured clusters and deploy the required components is required. Each student must be able to experiment with his own cluster to run the exercises, experience the effects and verify the results.

Therefore, we will be using k3d as an orchestrator to spin up lightweight Kubernetes distributions based on k3s. It is using Docker in Docker technology (i.e., running containers within containers) to spin up virtual Kubernetes node containers (similar to VMs or physical servers) and start the required services and applications/pods within these node containers.

The objective of this lab session is to:

- Setting up the tooling required to manage the cluster and work with Kubernetes
- Learn how to spin up and tear down clusters.
- Investigate the anatomy of and Kubernetes cluster.
- Make applications and services available.

Required Resources and Tools

The recommended environment for this lab is to use the virtual machine provided on the ZHAW InIT OpenStack cluster (APU) at https://apu.cloudlab.zhaw.ch. It is possible to perform the labs on your local machine, but given the diverse platforms used by students, we cannot ensure that everything will work as it should. Also, the environment might interfere with other Kubernetes distributions or tools running on your computer.

The following software is required for this Lab (and most CCP2 labs):

- Installed Docker environment. This is available in a virtual machine available on the APU cluster, based on the `2022-12_Ubuntu_Jammy_with_Docker` image.
- `git` and `curl` CLI clients (available on the preconfigured VM)
- An ssh client on your local machine to connect to the VM
  - **Mac / Unix / Linux**: `ssh` (and `ssh-keygen`) is required.
    Both commands are part of the OpenSSH-Package which is preinstalled on most distiributions. If not, use the package manager to install it.
  - **Windows**: On Windows10/11 install the OpenSSH-Client from Microsoft
    Or use `ssh` und `ssk-keygen` from the Windows Subsystem for Linux (WSL)
    As a last resort, you can also use PuTTY and PuTTYgen.

The credentials required to access the Lab-Infrastructure are provided on Moodle

References

- k3d – run k3s cluster in docker: https://k3d.io/
- Arkade – utility for managing K8s CLI tools: https://github.com/alexellis/arkade
- kubectl – cheet sheet: https://kubernetes.io/docs/reference/kubectl/cheatsheet/

Task 1 - Set up the OpenStack Environment and create a Lab-VM

For most of the CCP2 laboratory exercises, a single Virtual Machine will be sufficient. This VM can be run on the InIT Cloudlab OpenStack infrastructure. This task will guide you through setting up the OpenStack environment, create the Lab-VM and make it accessible from your local computer.

Attendees of CCP1 already know the procedure and may go through the instructions in a faster pace.
The Environment and Lab-VM specification is as follows:

Security Group "CCP2":

● Egress,	Any protocol,	Any port,	to 0.0.0.0/0
● Ingress,	ICMP,	Any port,	from 0.0.0.0/0
● Ingress,	TCP,	22 (SSH), 80 (HTTP), 443(HTTPS), 5000 (Image-Registry), 6443 (K8S-API), 8080, 8443 (Application Ports)	from 0.0.0.0/0

Lab-VM Configuration:

● Image:	2024-01_Ubuntu_Jammy_with_Docker
● Flavor:	m1.large
● Network:	internal2
● Sec Groups:	CCP2
● Key Pair:	Your Public SSH-Key
● Floating-IP:	from the external2 network

Subtask 1.1: OpenStack account configuration

Login to OpenStack Dashboard on the APU Cluster: https://apu.cloudlab.zhaw.ch.
Use the credentials provided on Moodle. If this is your first login to APU, you should

- Change the language settings in the user menu at the top right (Einstellungen/Settings > Benutzereinstellungen/User Settings > Sprache/Language) to "English (en)". We will use the English names in all lab sessions.
- Next, you must also change the initial password. In User Settings, select "Change Password" and change it to a unique secure password. This is independent of the ZHAW user credentials.

If you are new to OpenStack, you should explore the UI. It is basically separated into the three main Infrastructure Components **Compute** (to manage VMs), **Volumes** (to manage storage), and **Networking** (to manage virtual networks). In this course, we will only use basic components like Compute Instances (VMs), Security Groups (Firewall-Rules) and Floating-IPs (Public IP address mapping / NAT).

Subtask 1.2: Setup SSH keypair

Connection to your virtual machines will be done using Secure Shell (SSH). Therefore, you need to create an SSH-Keypair on your computer and upload the Public-Key to the OpenStack-Cluster, so it can be injected into your virtual machines. OpenStack accepts SSH keys of type RSA or the newer elliptic curve based EdDSA (Ed25519), but rejects the older DSS algorithms.
Detailed instructions how to create and use SSH key pairs with OpenStack can be found here.

- If you already have an RSA or EdDSA key-pair, you may use this and skip the next step.
- If not, create your own key-pair using OpenSSH, e.g. for an EdDSA key-pair
  ```
  ssh-keygen -t ed25519 -f ~/.ssh/apu
  ```
  This creates two files, `apu` and `apu.pub` in your `$HOME/.ssh/` directory
  - The generated key pair has a public part (file typically ending with `.pub` extension) and a private part which is meant never to be shared with anyone.
  - If you do not have an OpenSSH installation (Linux/Unix, macOS, Windows OpenSSH/WSL) you fill find the instructions for PuTTY and PuTTYgen here or here

- Register your key pair with your account on https://apu.cloudlab.zhaw.ch
  - Use Project > Compute > Key Pairs > Import Public Key
  - Give it a suitable name, select Key-Type SSH and copy/paste the whole content of your Pubic-Key file (.pub) to the text field.

Subtask 1.3: Set up the CCP2 Security Group

A Security Group is a set of firewall-rules, which can be assigned to virtual machines and configure, which traffic is allowed from and to the VM.

Every OpenStack project comes with a Default security group that allows all traffic from inside a virtual machine to go out (outbound/egress) but blocks all external traffic from coming in (inbound/ingress).

The goal of this subtask is to create a CCP2 Security Group allowing the connections we will use for the labs.

- Login to the APU dashboard and make sure you are in your CCP2 project (menu bar, top left)
- Select Network > Security Groups > Create Security Group.
- Create a security group with name CCP2 and add the rules in the above configuration sheet. Add a separate entry for each ingress TCP port (you may choose from the set of predefined rules in the Add Rule dialog or configure a custom rule).

Note: Security groups are dynamic, which means that they can be changed at any time and have immediate effect on all the VMs they are assigned to.

Subtask 1.4: Create the Lab-VM

Now everything is ready to create the Lab-VM. This VM will be used in multiple labs and should not be deleted each time. But in case you have trouble, it gets inaccessible, etc., you may delete and recreate it at any time.

- Start the Launch Instance Wizard (Compute > Instances > Launch Instance)
- Select a name of your choice, e.g., "CCP2 Lab".
- In "Source" choose Boot Source "Image" and select the **`2024-01_Ubuntu_Jammy_with_Docker`** image (⬆).
  The volume size will be automatically set to the required size when you choose the image.
- Select Flavor **`m1.large`**.
- In "Networks" attach the VM to the '**`internal2`**' network.
- Add the **`CCP2`** Security Group you created before
- In "Key Pair" make sure to select your Public SSH key.
- Finally, you can Launch the Instance.

  The VM is now created and booted. It should show up in the Compute > Instances view
  You can check the configuration, status, logs, etc. by clicking on the instance name.

Once the VM is created and booted, it is only available on the cluster's **`internal2`** network (IP address range 10.10.0.0/16). To make it accessible from the ZHAW network, you need to assign it a public IP address, which is called Floating-IP in OpenStack (Traffic to this IP-Address will then be forwarded to the VM → NAT).

- In the VMs Action-Menu select "Associate Floating IP" and select an available IP address.
  If no floating IP addresses are allocated, you may add one using the + Button. Select an address from the **`external2`** pool.

Now you should be able to login to the VM from your computer using SSH:

`$ ssh -i ~/.ssh/apu ubuntu@160.85.253.X` (replace the X with your Floating IP address)

Because you will often connect to this VM using SSH, you may add a 'Host' entry like the following to your `~/.ssh/config` file:

```
…
Host ccp2-lab
  HostName 160.85.253.X
  User ubuntu
  IdentityFile ~/.ssh/apu
```

With this, you should be able to connect to the VM using the command:

```
$ ssh ccp2-lab
```

Make sure you can connect to the VM using SSH before continuing with the next task.

Task 2 - Add tools and resources to the lab VM

In this task you will set up the environment on the VM with the required tools and resources.

Setup up the VM as follows:

- Connect to the virtual machine using ssh (see previous task).
  - The following commands will be executed on the VM as standard user ubuntu. Commands requiring root access will use `sudo` to get privileged access.
- Clone the Lab-Repository to the home directory on the VM:
  `$ git clone https://github.zhaw.ch/CCP2/CCP2-ARCH-Lab.git`
  and change to the CCP2-ARCH-Lab folder.
- To speed up the installation, a setup-script (`setup-tools.sh`) is provided in this folder. The script will do the following:
  - Verify that Docker is installed.
  - Install the ubuntu `net-tools` package providing essential network commands like `netstat, ifconfig, arp, hostname`, etc.
  - Install and configure the `arkade` command
    Arkade simplifies installing the DevOps tools we will use in this and upcoming labs. It installs the tools in the users `~/.arcade/bin` folder to not conflict with the system tools, therefore the setup-script adds this directory to the search path ($PATH)
  - It uses Arkade to install some basic Kubernetes tools like `kubectl`, `k3d`, `k9s`,...
    Later in the course, we will use arkade to install additional tools. You may check out the extensive list of supported tools using: `arkade get`
  - Finally, it will add bash-completions for the tools, to support tab-completion on sub-commands and options.
- Execute the setup script: `./setup-tools.sh`
  and follow the installation process. The script only installs the tools, if they are not yet available, so you can also start it multiple times if something fails.
- The installed basic commands will be used to manage and inspect the Kubernetes clusters and applications.
  (if the commands are not found, you may run the command `source ~/.profile` or relogin to configure the shell `PATH`)
  - **`kubectl`** is the standard Kubernetes CLI tool. Verify that it works using:
    `kubectl version --short`
    Kubectl is extensible using plugins and has many sub-commands. You can always use `kubctl help` to get additional information about the sub-commands, e.g. use `kubectl help config` to get information about the config sub-command.
    The following cheat-sheet may help to find the correct command for a specific task
    https://kubernetes.io/docs/reference/kubectl/cheatsheet/
  - **`k3d`** (https://k3d.io) is a Kubernetes cluster manager, which runs clusters nodes in Docker Containers using the k3s distribution. This is a very fast and flexible way to set up lightweight clusters for testing or experiments in seconds. Later in this lab we will set up a cluster to show the process.
  - **`k9s`** (https://k9scli.io/) is a terminal-based UI to navigate, observe and manage applications in a Kubernetes cluster. It gives you some better insight into the status and lets you interact with the components. The following cheat sheet helps to use k9s
    https://k9scli.io/topics/commands/

You now have a working environment in a VM that is ready to investigate managing K8s clusters.

Task 3 - Create and inspect your first cluster

In this task, you will create a first simple cluster and inspect its anatomy.

- First, make sure your workspace is empty, no cluster is running:
  - `k3d cluster list` should show an empty list.
  - `kubectl cluster-info` should show a connection error.
  - `docker ps` shows no running containers.

- Now we will create a cluster (ccp2) with 1 master node (server) and 2 worker nodes (agents):

  `k3d cluster create ccp2 --servers 1 --agents 2`

  (In this case, the `--servers` argument is optional because 1 is the default)

  The command will output the steps to build the cluster:

```
INFO[0000] Prep: Network
INFO[0000] Created network 'k3d-ccp2'
INFO[0000] Created image volume k3d-ccp2-images
INFO[0000] Starting new tools node...
INFO[0001] Starting Node 'k3d-ccp2-tools'
INFO[0001] Creating node 'k3d-ccp2-server-0'
INFO[0004] Creating node 'k3d-ccp2-agent-0'
INFO[0005] Creating node 'k3d-ccp2-agent-1'
INFO[0007] Creating LoadBalancer 'k3d-ccp2-serverlb'
INFO[0008] Using the k3d-tools node to gather environment information
INFO[0008] HostIP: using network gateway 172.19.0.1 address
INFO[0008] Starting cluster 'ccp2'
INFO[0008] Starting servers...
INFO[0008] Starting Node 'k3d-ccp2-server-0'
INFO[0016] Starting agents...
INFO[0016] Starting Node 'k3d-ccp2-agent-0'
INFO[0016] Starting Node 'k3d-ccp2-agent-1'
INFO[0027] Starting helpers...
INFO[0028] Starting Node 'k3d-ccp2-serverlb'
INFO[0036] Injecting records for hostAliases (incl. host.k3d.internal) and for 4 network members into
CoreDNS configmap...
INFO[0039] Cluster 'ccp2' created successfully!
```

  After a few seconds, the cluster will be available. You can verify the k3d managed clusters using:

```
$ k3d cluster list

NAME    SERVERS    AGENTS    LOADBALANCER
ccp2    1/1        2/2       true
```

  Also, the state of the cluster nodes can be listed using:

```
$ k3d node list

NAME                 ROLE           CLUSTER    STATUS
k3d-ccp2-agent-0     agent          ccp2       running
k3d-ccp2-agent-1     agent          ccp2       running
k3d-ccp2-server-0    server         ccp2       running
k3d-ccp2-serverlb    loadbalancer   ccp2       running
```

  This shows the single Kubernetes Master (server) and the two Worker (agent) nodes created and a Loadbalancer-Node (nginx based), which forwards requests from the host (VM) to the cluster.

  That the cluster is operational can be verified using: **`kubectl cluster-info`**

  You will see three running services (api-server - K8s control plane, CoreDNS and Metrics) and their respective endpoints. But what happened? What does the cluster look like?

- Let's check the above output of the build and analyze the steps (incl. commands to verify):
  1. First, it pulls the images for the k3s distribution and k4d helper from the registry.
     `docker images`
  2. Then the Docker environment is set up
     - a cluster-specific bridge network (k3d-ccp2) is created
       `docker network list`
       get details using: **`docker network inspect k3d-ccp2`**
     - an image caching volume (k3d-ccp2-images) is created
       `docker volume list`
       get details using: `docker volume inspect k3d-ccp2-images`
     - cluster node containers are created
       `docker ps --format 'table {{.ID}}\t{{.Image}}\t{{.Names}}\t{{.Ports}}'`
       shows 4 running container nodes (matching the k3d nodes above):
       - 1 server node, 2 agents and 1 server load-balancer
       - the fifth container 'k3d-tools' was temporarily used to download the images
  3. The Kubernetes cluster is built by running the master services on the server node and the worker services on the agent nodes using k3s.
  4. The load-balancer node is configured to forward traffic from the host to the cluster.
     The **`docker ps`** command shows, that currently only the Kubernetes api-server is made available from the host, by default on a random port around `35XXX`.
  5. Finally, the kubectl context (k3d-ccp2) is created for the admin user of the new cluster and set as current (active): **`kubectl config get-contexts`**
     Config details (e.g. cluster/user info for contexts):
     `kubectl config view`
     `k3d kubeconfig get` allows obtaining configurations from running clusters and **`merge`** them into other kubeconfig files.
     `kubectl config use-context…` would allow switching the current context to another cluster

  Because only K8S API traffic is forwarded to the cluster, its usability is limited.

- Let's tear down the cluster and set up a new one, which allows web-traffic and is available externally.

```
$ k3d cluster delete ccp2
INFO[0000] Deleting cluster 'ccp2'
INFO[0004] Deleting cluster network 'k3d-ccp2'
INFO[0004] Deleting 1 attached volumes...
INFO[0004] Removing cluster details from default kubeconfig...
INFO[0004] Removing standalone kubeconfig file (if there is one)...
INFO[0004] Successfully deleted cluster ccp2!
```

- Last, make sure your workspace is empty again, no cluster is running:

● `k3d cluster list`	should show an empty list.
● `kubectl cluster-info`	should show a connection error.
● `docker ps`	shows no running containers.
● `docker volume ls`	should be empty.
● `docker network ls`	the k3d-ccp2 network should be gone

Task 4 - Start the cluster with external access

In this task, you will create a cluster which is available from the ZHAW network and deploy a first application.

Subtask 4.1: Create a simple cluster with a configured load balancer

- Manually create the new cluster with a fixed API-Port and the load balancer configured to forward Web-Traffic to the ingress-proxy:

  ```
  $ k3d cluster create ccp2 --api-port 6443 -p "8080:80@loadbalancer" -p
  "8443:443@loadbalancer" --agents 2
  ```

  Now, the docker port mapping for the serverlb node is showing the forwarded ports:

  ```
  $ docker ps --format 'table {{.ID}}\t{{.Image}}\t{{.Names}}\t{{.Ports}}'
  ```

  ```
  CONTAINER ID   IMAGE                            NAMES               PORTS
  71a075f47f8e   ghcr.io/k3d-io/k3d-proxy:5.4.8   k3d-ccp2-serverlb   0.0.0.0:6443->6443/tcp,
  0.0.0.0:8080->80/tcp, :::8080->80/tcp, 0.0.0.0:8443->443/tcp, :::8443->443/tcp
  fbdbaa4637d5   rancher/k3s:v1.25.6-k3s1         k3d-ccp2-agent-1
  88a011f1252d   rancher/k3s:v1.25.6-k3s1         k3d-ccp2-agent-0
  e686b4af1f99   rancher/k3s:v1.25.6-k3s1         k3d-ccp2-server-0
  ```

  And the ingress-proxy is already available locally:

  `curl http://localhost:8080/`

  Because no application is deployed, a 404-error message is returned.

  Same for the encrypted endpoint **`curl -k https://localhost:8443`**.

  The CCP2 security-group of the VM allows access to port 8080 and 8443. Therefore, the load balancer/proxy is also available externally. Try accessing the page from a Web browser using `http://<your-vm-floating-ip>:8080` (should also return a 404-error response)
- Before we now deploy an app, we will discover a third way to create clusters using a config file. Therefore, we will remove this simple manual cluster again using
  `$ k3d cluster delete ccp2`

Subtask 4.2: Create a Cluster using a configuration file

- Configuration of a k3d cluster can be done completely using command line arguments, as above. `k3d help cluster create` shows the full list of possible configuration options. But this can get very extensive.
  Therefore, k3d supports using a configuration file (https://k3d.io/v5.4.8/usage/configfile/)
  In the lab repository, such a configuration is provided (`k3d-ccp2-config.yaml`). It configures additional components like a local image registry, a volume driver for local storage and some extended configuration. Inspect the file using an editor:
  - Most of the parameters are similar to the cli arguments (see the specification for details) e.g., `server`, `agents`, `hostPort`, …
  - The **`image`** parameter allows specifying which K8S version to use
  - The **`kubeAPI`** settings specify how to reach the K8S API server
    `hostIP: 0.0.0.0` specifies, that it will be listening on all IP-Addresses of the host.
    `host` should be a public available domain name. If you want to reach it from the ZHAW network. Here we are using the pseudo domain **`nip.io`**, which returns the IP address specified in the subdomain part, i.e. looking up the domain `*.160.85.253.9.nip.io` will always return the IP address 160.85.253.9 for all wildcard subdomains (*).

Adapt the `host` domain (replace the <x>) to match your VMs floating IP address, e.g. `api.160.85.253.`**`9`**`.nip.io`

- ○ The Load Balancer will now forward the standard web ports 80 and 443 to the ingress proxy.

- Make sure the previous cluster is deleted (`k3d cluster list`)
- Create the new cluster using `k3d cluster create --config k3d-ccp2-config.yaml`
  The first time it may complain, that the directory `/home/ubuntu/storage` is missing, but it will be created in the process.
- Now `kubectl cluster-info` shows the API endpoint with a valid domain name and your ingress should be available on the default web ports (80, 443). e.g., http://xyz.160.85.253.X.nip.io should return a response (still 404) because no application is deployed.

Subtask 4.3: Deploy a simple app

Next, we deploy the **`hostname`** app, which simply returns the name of the host/node it is running on.

- Apply the application resources for hostname-app.yaml:
  `$ kubectl apply -f hostname-app.yaml`
  The file contains three Kubernetes resource specifications. A Deployment which will run 3 replicas of the app., a Service to make the deployment available cluster internally and an Ingress to forward external traffic to the service.
  We will learn, in the next labs, how this all works.
- Try to access the web app on the VM using: `curl http://hostname.127.0.0.1.nip.io`
  - What does it return? Try multiple requests.
  - Is it accessible from outside the VM (e.g., your laptop)? Why?
- Update the `hostname-app.yaml` manifest and uncomment the second ingress rule at the end. Also change the domain name to match your VMs floating IP.
- Reapply the resource specification:
  `$ kubectl apply -f hostname-app.yaml`
- Verify that the hostname app is reachable from your browser.

Subtask 4.4: Optional - Access the API server from your local computer

To access the VMs cluster API server from your local computer, you need to:

- Install `kubectl` on your local computer (using brew, chocolatey, arkade, ..)
- Extract the kubeconfig on the VM for the ccp2 cluster:
  `$ k3d kubeconfig get ccp2 > ccp2-kubeconfig.yaml`
- Copy the file to your computer and merge the ccp2 config to the existing config in `~/.kube/config`
  (see https://jacobtomlinson.dev/posts/2019/how-to-merge-kubernetes-kubectl-config-files/)
- set the context to k3d-ccp2: **`kubectl config set-context k3d-ccp2`**
- Now you should be able to access the cluster on your VM, e.g., `kubectl cluster-info`
- Also, all other tools using kubeconfig will work, e.g., **`k9s`**

If you recreate the cluster on the VM, you must repeat the process because the certificates to access the cluster change.

Cleanup

Open ports are always a security risk. At the end of the lab-session you may stop the cluster

- k3d cluster stop ccp2

and restart it when you continue:

- k3d cluster start ccp2