

Cloud Computing Applications and Services

University of Minho

Guide 3

Container Orchestration and Scheduling

2024

Having configured and deployed an application with Docker, you may have noticed that the container approach is quite appealing. So appealing in fact that it may make sense to use it for large scale application deployment. More specifically, deployments where one may want to grow or shrink computational and storage resources provided for the application according to actual workloads and demand. To achieve this goal in an automated fashion, container orchestration and scheduling platforms have emerged. Kubernetes is one of the most widely known examples of such platforms, and is now being offered as a service by several cloud providers, such as Google Cloud.

In this guide, you will go through the steps of configuring and deploying a *Kubernetes* cluster, as the Cloud Provider (☁️). Then, you will use the *Kubernetes* cluster to deploy the *Swap* application, taking the role of the Application Developer (👩🏫).

Recall these symbols from previous guides:

- ☁️ Tasks to be performed by the cloud provider;
- 👩🏫 Tasks to be performed by the application developer;

The symbol 🖥️ signals tasks that need to be performed in a specific machine (myvm).

Part I

Kubernetes

Kubernetes (<https://kubernetes.io>), also known as K8s, is an open-source system for automating the deployment, scaling, and management of containerized applications. Take a look at the slides accompanying this guide and remember the theoretical classes on K8s!

K8s documentation is available at: <https://kubernetes.io/docs/home/>.

I-1 Setup

I-1.1 K8s Cluster Configuration and Initialization ☁️

A K8s cluster is a group of computing nodes that run containerized applications. It usually comprises a control plane that maintains the cluster's desired state and one or more nodes that run applications.

1. Start by setting up four VMs with Vagrant:


- controlplane*, *node1* and *node2* will form the K8s cluster. **Note:** If your host server does not have enough resources, you can run only *controlplane* and *node1* VMs.
- myvm* will be used to run Ansible playbooks and deploy Swap in the K8s cluster.

```
vagrant up myvm controlplane node1 node2
```

- Now, let us install and configure a K8s cluster. Inspect the new playbook and roles provided along this guide for the *CloudProvider* project and observe the new modifications and new Ansible concepts introduced in this guide.

- The inventory (*hosts* file) now has two more groups: the *control_planes*, composed of the *controlplane* VM, and the *k8s*, composed of the *controlplane* and the two *node* VMs. Notice how the *install_k8s.yml* playbook invokes different roles for the different groups.

- Make sure the IPs defined on the *hosts* file are correct according to your setup.
- At the *group_vars/k8s.yml* file ensure that the variable *m_iface* is set to the private network interface of your VMs. Use the command *ip add* to check the interface name.
- Test Ansible connectivity to each host with the Ping module: `ansible -m ping all`

- Connect to **myvm** and use this playbook to automatically install Docker and K8s on the *controlplane*, *node1* and *node2* VMs and setup a K8s cluster :

```
ansible-playbook install_k8s.yml
```

Tip: Run the following commands to store your private key password at the ssh agent, avoiding Ansible asking for your key password every time:

```
eval "$(ssh-agent -s)"
ssh-add <private_key_path>
```

- Connect to the *controlplane* (e.g. `ssh vagrant@192.168.56.100`) and verify if the cluster was correctly setup (i.e. if it has the 3 nodes):

```
kubectl get nodes -o wide
```

I-1.2 Setup local kubectl

Kubectl is a command line tool provided by Kubernetes for communicating with the cluster's control plane. Let us install and configure kubectl **on myvm** so we can remotely access the K8s cluster.

- Install kubectl on *myvm* following the instructions at <https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/#install-kubectl-binary-with-curl-on-linux>.

Note: Make sure you select the right architecture according to your setup (i.e., x86-64 or ARM64).

- Configure local kubectl to connect to the remote k8s cluster:

```
mkdir -p ~/.kube/
ssh vagrant@<CONTROL_PLANE_IP> kubectl config view --raw > ~/.kube/config
```

- Test the connection from *myvm* by checking the K8s nodes:

```
kubectl get nodes
```

I-1.3 Persistent Volumes Configuration

Persistent Volumes (PV) are used to ensure Pods' data persistence. Next, we will create two local PVs (one per node) and a Storage Class (SC) to define their storage type, provisioning, and behavior.

- Inspect the Ansible role *k8s-storage* provided along with the *CloudProvider* project and observe how we can use Ansible to automate the creation of K8s storage classes and persistent volumes.

The file *templates/k8s-sc.yml* specifies a Storage Class (*local-storage*) with the *volumeBindingMode* equal to *WaitForFirstConsumer* to instruct K8s to wait to bind a Persistent Volume Claim (PVC) until the Pod (e.g., MySQL) using it is scheduled. The file *templates/k8s-node-pv.yml* specifies the configurations of a PV. Note that our role (*tasks/main.yml*) will try to create one PV for each node: *pv-1* is assigned to *node1* and *pv-2* is assigned to *node2*.

Try running this role with the following command:

```
ansible-playbook install_k8s.yml -t k8s-storage
```

2. Use the command `kubectl get sc` to list existing SCs and validate it was correctly created.
3. Use the commands `kubectl get pv` to list existing PVs and validate they were correctly created.

I-2 Tasks

Now that, as a *cloud provider*, we have set up a K8s cluster and configured storage to be used by applications, let us move on to the *App developer* role and deploy the Swap application in this cluster.

I-2.1 Database Deployment

The YAML files to deploy MySQL on K8s are provided along with this guide (*k8s-mysql-yamls*). Be sure to inspect and understand them while doing the steps below. Remember to consult K8s documentation.

1. First, create a Persistent Volume Claim (PVC), to which the MySQL Pod will be bound, to request for a piece of storage. Inspect and create the PVC defined on *mysql-pvc.yml*:

```
kubectl apply -f mysql-pvc.yml
```

Verify if it was correctly created with `kubectl get pvc`. Note that its status remains “*Pending*”.

2. Next, set up a MySQL database for the Swap application. The file *mysql-deployment.yml* defines a Deployment object that specifies the Pod template for deploying MySQL and is expecting a PVC named *mysql-pvc*. Run the following command to create the MySQL Deployment:

```
kubectl apply -f mysql-deployment.yml
```

3. Use command `kubectl get all` to verify the objects that were created. It should show a *ReplicaSet*, a *Deployment*, and a *Pod* for MySQL. Explore the following commands to get information about a specific MySQL object

```
kubectl get deployment [DEPLOYMENT_NAME]
kubectl get replicaset [REPLICASET_NAME]
kubectl get pod [POD_NAME]
```

4. Check again the status of the PVC. If correctly configured, the status should now appear as “*Bound*”, and the volume column should have the name of one of the previously created PVs (*pv-1* or *pv-2*).
5. Rerun the command `kubectl get all` and verify if all MySQL objects are ready.
6. Now, run the command `kubectl apply -f mysql-service.yml` to create a Service for exposing MySQL to the other pods in the cluster. When executing the command `kubectl get svc`, there should be a Service named *mysql-service* with the type ClusterIP.
7. Run the following command to verify if you can access the MySQL database:

```
kubectl exec -it <MYSQL_POD_NAME> -- mysql -u<MYSQL_USER> -p<MYSQL_PASSWORD> <<< "SHOW
DATABASES;"
```

8. Explore the `--selector` option to select objects with a specific label. For example, the command `kubectl get all --selector=tier=database` will show the objects with the label *tier=database*.

I-2.2 Webserver Deployment

Now, let us deploy the webserver component of the Swap application. For this, create a Deployment object and a Service object for Swap. Take into account the following guidelines:

1. The Swap Pod should be configured to use the Docker image `ascnuminho/swap:u24` and should set the Environment Variables `DB_HOST`, `DB_DATABASE`, `DB_USERNAME`, `DB_PASSWORD` to the correct values. Also, it should expose the port to which the Swap application is listening (port 8000).
2. The Swap Service object should be of type *NodePort* to allow opening a specific port (e.g., 30007) on all the Nodes and forward any external traffic sent to this port to the Swap Service.
3. Use `kubectl` to deploy the aforementioned K8s objects. Use the commands `kubectl describe pod <SWAP_POD_NAME>` and `kubectl logs <SWAP_POD_NAME>` for troubleshooting.
4. Try it out! Access Swap from your browser. The URL should contain the IP of one of the K8s cluster's nodes and the port defined on the Swap Service (e.g. `<IP_CONTROL_PLANE>:30007`). Note that you can also access Swap through the IP addresses of Node VMs.
5. Use the command `kubectl exec -it <SWAP_POD_NAME> -- php artisan db:seed` to seed the database.
6. Explore the command `kubectl delete <OBJECT_TYPE> <OBJECT_NAME>` to delete Swap and MySQL objects and undeploy the application. Do not forget to delete MySQL's PVC.

Part II

Automation of tasks

You still had to do some manual tasks in the previous steps of this guide, namely when deploying MySQL and Swap's Webserver. Let us use Ansible, once again, to automate these few last tasks and make the process of deploying Swap in K8s fully automated.

II-1 Setup 📌

1. Start the VMs:

```
vagrant up myvm controlplane node1 node2
```

II-2 Tasks 📋

Now, let us further automate Swap's deployment.

1. Start by inspecting the Ansible playbook *k8s-swap-install.yml* and the role *k8s-mysql* provided along with the *AppDeveloper* Project. Observe the tasks defined in these and the Ansible modules and commands used and compare them with the manual steps done in Section I-2.1.
2. Notice the use of Ansible templates on the *k8s-mysql* role. *Templates* can be used to dynamically generate text-based files using templates, variables, and facts for configuration and other purposes. In this specific case, we are using templates to generate the YAML files for the MySQL K8s objects. Find more about Ansible templates at https://docs.ansible.com/ansible/latest/collections/ansible/builtin/template_module.html.
3. Install Kubernetes package for Python on *myvm*, which is necessary for running the Ansible k8s modules: `sudo apt install python3-kubernetes`

4. Run the *k8s-swap-install.yml* playbook to deploy MySQL database:

```
ansible-playbook k8s-swap-install.yml -t mysql
```

Use the command `kubectl get all` to verify if all MySQL objects were correctly created.

5. Leverage the provided *k8s-mysql* role as inspiration and fill in the *k8s-swap* role based on the manual steps done in Section I-2.2. Specifically, the *k8s-swap* role should allow for the following:
 - (a) creating a *Deployment* and a *Service* for Swap.
 - (b) seeding Swap's database when the option `seed_db` is `true`

Test your role by running the following commands:

```
ansible-playbook k8s-swap-install.yml -t swap
ansible-playbook k8s-swap-install.yml -t swap -e seed_db=true
```

Extras

1. Explore ConfigMaps for decoupling environment-dependent configurations from containerized applications, allowing them to remain portable across environments.
<https://kubernetes.io/docs/concepts/configuration/configmap/>
2. Explore Secrets for storing sensitive data such as passwords, tokens, or keys.
<https://kubernetes.io/docs/concepts/configuration/secret/>
3. Build another Ansible playbook for undeploying the application (e.g., *k8s-swap-uninstall.yml*). The playbook should allow the undeploy of the webserver and the database components separately and have the option to delete or not the application's data.

Learning outcomes

Hands-on experience with software container technology and orchestration (Docker and Kubernetes).
Deployment of a distributed application on top of a container platform.