Introduction to Kubernetes

Short description

Lab focus: Getting experience with Kubernetes
Credits: This is not a graded assignment but finishing it will help you a lot with the next graded assignments.

Intro

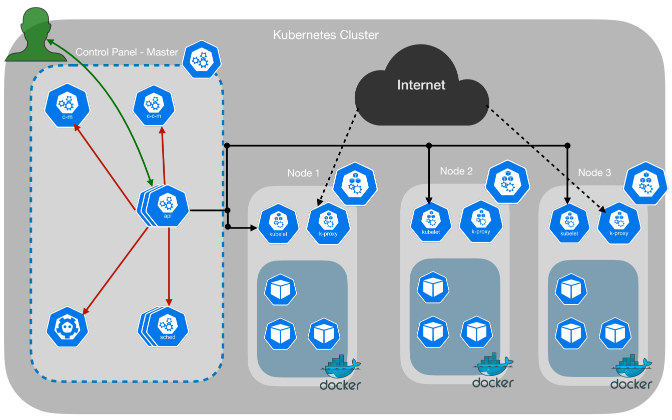


Figure 1: The components of Kubernetes.

The Figure 1 shows the components of a cluster running Kubernets. Specifically we notice the following parts:

- *Cluster*: It is a collection of compute, storage and networking resources that Kubernetes uses to run the various workloads that comprises your system.
- *Nodes*: It is a single physical or virtual host. One of them is the *master* node and the others are the *workers*.
- ***Pods***: It is the smallest unit that can be scheduled as deployment in Kubernetes. Each worker is responsible of running a pod. A pod is a collection of one or more containers, and hence they are always scheduled together. All the containers inside the pod have the same IP address and port number. Also, they serve as Kubernetes' core unit framework. Finally, they provide a great solution of managing groups of closely related containers that depend on each other and need to cooperate on the same host.
- *kublet*: Manages the communication between node and master.
- *kube-proxy*: Allows network traffic to come in and then be redirected into the various pods.
- *API*: The API server is the front end for the Kubernetes control panel.
- *etcd*: Kubernetes uses it to store the entire cluster state. It is a high-reliable distribute key-value store.
- *Kube-scheduler*: Watches for newly created Pods with no assigned nodes, and creates a node for them to run.

Why Kubernetes is important? The abstractions in Kubernetes allow you to deploy containerised applications to a cluster without tying them specifically to individual machines. Kubernetes automates the distribution and scheduling of application containers across a cluster in a more efficient way.

Module Coordinator: Dimitris Chatzopoulos - School of Computer Science - UCD - dimitris.chatzopoulos@ucd.ie
Teaching Assistant: Joana Tirana - School of Computer Science - UCD - joana.tirana@ucdconnect.ie

Exercise One - Create a Cluster

Since a real physical cluster is not available, Kubernetes provide a tool that allow us to test several deployments and functionalities. This tool is Minikube, a lightweight Kubernetes implementation that creates a VM * on your local machine and deploys a simple cluster containing only one node. The Minikube CLI provides basic bootstrapping operations for working with your cluster, including start, stop, status, and delete.
* As you will see in the downloading procedure you can either use the VM configuration or run Minikube as a container. **Select the container option!**

Task 1:

Install Minikube (https://minikube.sigs.k8s.io/docs/start) to create a single node cluster on your local machine. Choose the option that uses Docker.

Start the Minikube cluster:

```
minikube start
```

Use kubectl to interact with Kubernetes. Try the following commands to learn the status of your cluster:

```
kubectl cluster-info
```

```
kubectl get nodes
```

Task 2:

Create you own *namespace*.

Kubernetes Namespace is a mechanism that enables you to organize resources. It is like a virtual cluster inside the cluster. A namespace isolates the resources from the resources of other namespaces. Here is the list of the initial namespaces:

- **kube-system**: System processes like Master and kubectl processes are deployed in this namespace; thus, it is advised not to create or modify the namespace.
- **kube-public**: This namespace contains publicly accessible data like a configMap containing cluster information.
- **kube-node-lease**: This namespace is the heartbeat of nodes. Each node has its associated lease object. It determines the availability of a node.
- **default**: This is the namespace that you use to create your resources by default.

You can view the list of the namespaces using the following command:

```
kubectl get namespaces
```

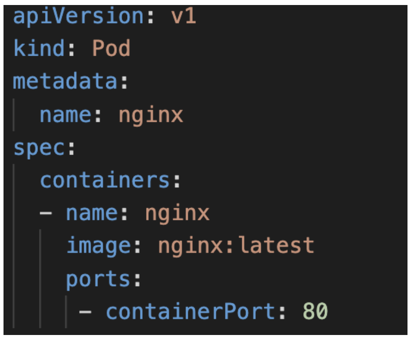
Try to create your own namespace using a Yaml file.

Task 3

: Create a pod

A pod is a collection of one or more running containers, allowing for simple container movement within a cluster. The containers inside the pod share a single IP address, as well as the pod's storage, network, and any other requirements. In Figure 2 is an example of a yaml file that describes the creation of single-container pod. The pod will include a container of a nginx image.

Then use the following command to start your pod:



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
```

Figure 2: Example of Yaml file to create a pod with a single container.

```
kubectl create -f <file-name>
```

You can view the running pods using the following command:

```
kubectl get pods
```

And with the following command you can get all the details about the pod:

```
kubectl describe pods
```

Try to create a pod that contains more than one container (You can use whichever images you prefer).

Task 4 : Create a service.

A Service is one of the three controllers of a pod (the other two are Jobs, and Stateful Sets). The controllers differ to how they handle a pod that is terminated. For instance, the service usually deploys services that we want to be always up and running. Also, with the service we are able to expose our application outside from the cluster's world and view its content.

Use the given code to start a service that runs a pod with a nginx container.

```
kubectl apply -f nginx.yaml
```

Use the correspond `kubectl get` commands to check if the pod, service, and deployment is running.
Then in order to have access to the website run the following command:

```
kubectl port-forward service/nginx 8080:80
```

Open your browser to http://localhost:8080/ and check the website.

Exercise Two - Converting the Docker-compose flow to Kubernetes

Delete the previous deployment and make sure that there are no pods/services/deployments running on your Minikube cluster. In the previous practical, we used docker-compose to build multi-container apps. Unfortunately, you cannot use the same configuration file format to run multi-container pods into Kubernetes. However, there are tools that allows us to convert a docker-compose file into a Kubernetes.

Task 1: Install Kompose. This is the tool that we will use to convert the docker-compose file to Kubernetes.

Task 2: Use the source code from the first Docker tutorial (https://docs.docker.com/compose/gettingstarted/). You have to make some adjustments to the docker-compose file. You should make the following changes:

(a) For the *web* service you cannot use any more the build command that points to the local Dockerfile. You should publish your image to your repository as *username-to-registry/image-name*. Then change the `build` declaration to `image` and insert the name of the registry you just created.

(b) Add to the web service a "always" restart policy.

(c) You should define the ports for the redis service. The default configuration is: `"6379:6379"`

Task 3: Convert the docker-compose file to Kubernetes, and create a deployment to your cluster.

HINT ;) Follow the documentation from Kompose.

Task 4: Find the services using the appropriate kubectl command, and get its output with the command:

```
minikube service <service-name>
```

This will automatically open your browser to the nodes proxy address.
HINT ;) If everything was implemented right, you will see the same output with the one you saw when you used the docker-compose.