Kubernetes Classroom

Kubernetes for Beginners · Lab

Mouna Rouini · Mariem Cherif · Skander Soltane · Adam Lahbib

Feb. 15, 2023 @ INSAT

Contents

1 Lab Objectives

This is a beginner lab by Jérôme Petazzoni. The goal of this lab is to learn the basics of Kubernetes and how to use it to deploy applications.

2 Definitions

2.1 What is kubectl?

Kubectl is a command-line tool that is used to control Kubernetes clusters. It is a tool that is used to deploy and manage applications on Kubernetes. It is a command-line tool that is used to control Kubernetes clusters. It is a tool that is used to deploy and manage applications on Kubernetes.

2.2 What is Kubeadm?

Kubeadm is a tool that is used to bootstrap a Kubernetes cluster. It is a command-line tool that is used to initialize a Kubernetes cluster. It is a tool for bootstrapping a minimal Kubernetes control plane, which consists of the API server, controller manager, and scheduler. It also installs the networking solution and the kube-proxy daemon.

3 Part 1: Kubernetes Cluster Setup

3.1 Start the cluster

In the master terminal, we run the following command to initialize the cluster:

```
kubeadm init --apiserver-advertise-address $(hostname -i)
```

- `kubeadm init` is a command in Kubernetes that is used to initialize a new Kubernetes cluster. It is a tool for bootstrapping a minimal Kubernetes control plane, which consists of the API server, controller manager, and scheduler.

```
[node1 ~]$  kubeadm init --apiserver-advertise-address $(hostname -i)
Initializing machine ID from random generator.
I0215 20:38:35.040207    2516 version.go:251] remote version is much newer: v1.26.1; falling back to: stable-1.20
[init] Using Kubernetes version: v1.20.15
[preflight] Running pre-flight checks
        [WARNING Service-Docker]: docker service is not active, please run 'systemctl start docker.service'
        [WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the Docker cgroup driver. The recommended driver is "systemd". Please fo
llow the guide at https://kubernetes.io/docs/setup/cri/
        [WARNING FileContent--proc-sys-net-bridge-bridge-nf-call-iptables]: /proc/sys/net/bridge/bridge-nf-call-iptables does not exist
[preflight] The system verification failed. Printing the output from the verification:
KERNEL_VERSION: 4.4.0-210-generic
DOCKER_VERSION: 20.10.1
OS: Linux
CGROUPS_CPU: enabled
CGROUPS_CPUACCT: enabled
CGROUPS_CPUSET: enabled
CGROUPS_DEVICES: enabled
```

```
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:
```

We copy the whole line that starts with `kubeadm join` and paste it in the workers' terminals.

- `kubeadm join` is a command in Kubernetes that is used to join a new worker node to an existing Kubernetes cluster. This command is typically run on the new worker node after it has been provisioned and prepared with the necessary software and configuration.

```
[node2 ~]$ kubeadm join 192.168.0.8:6443 --token ln0mic.ybdsicrp6aivirz4 \
>     --discovery-token-ca-cert-hash sha256:576d022a0a64af78012c28bac53bbf342581a7e08874cd693a7f3ea17eecdf2c
[preflight] Running pre-flight checks
        [WARNING Service-Docker]: docker service is not active, please run 'systemctl start docker.service'
        [WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the Docker cgroup driver. The recommended driver is "systemd". Please fo
llow the guide at https://kubernetes.io/docs/setup/cri/
        [WARNING FileContent--proc-sys-net-bridge-bridge-nf-call-iptables]: /proc/sys/net/bridge/bridge-nf-call-iptables does not exist
[preflight] The system verification failed. Printing the output from the verification:
KERNEL_VERSION: 4.4.0-210-generic
DOCKER_VERSION: 20.10.1
```

```
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

RTNETLINK answers: File exists
```

We can verify in the master, by using this command `kubectl get nodes`.

```
[node1 ~]$ kubectl get no
NAME     STATUS     ROLES    AGE   VERSION
node1    NotReady   master   55s   v1.14.9
node2    NotReady   <none>   20s   v1.14.9
```

This is what we have so far:

```
[node1 ~]$ kubectl get po
No resources found.
[node1 ~]$ kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   74s
```

3.2 Initializing Cluster Networking

```
[node1 ~]$ kubectl apply -f https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemonset-k8s-1.11.yaml
serviceaccount/weave-net created
clusterrole.rbac.authorization.k8s.io/weave-net created
clusterrolebinding.rbac.authorization.k8s.io/weave-net created
role.rbac.authorization.k8s.io/weave-net created
rolebinding.rbac.authorization.k8s.io/weave-net created
daemonset.apps/weave-net created
[node1 ~]$
```

- `kubectl apply` is a command in Kubernetes that is used to create or update resources in a Kubernetes cluster. It reads a YAML or JSON file that contains a configuration for a Kubernetes resource, such as a deployment, pod, service, or config map, and applies the configuration to the cluster.

And with this, our cluster is ready to use.

3.3 Getting the application source code

We clone the application source code from the following repository:

```
[node1 ~]$ git clone https://github.com/dockersamples/dockercoins
Cloning into 'dockercoins'...
remote: Enumerating objects: 102, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 102 (delta 0), reused 3 (delta 0), pack-reused 97
Receiving objects: 100% (102/102), 121.82 KiB | 0 bytes/s, done.
Resolving deltas: 100% (28/28), done.
```

3.4 Running the application

```
[node1 ~]$ cd ~/dockercoins
[node1 dockercoins]$ docker-compose up
Creating network "dockercoins_default" with the default driver
Building rng
Step 1/5 : FROM python:3-alpine
3-alpine: Pulling from library/python
63b65145d645: Pull complete
781eddb6f342: Pull complete
1ae1f1ea756f: Pull complete
5b3e7c82c61d: Pull complete
369973018634: Pull complete
Digest: sha256:84630610c68e7c97384bc6e10f5490ab7b8398c30cdfffefa139ae20c3407cda
```

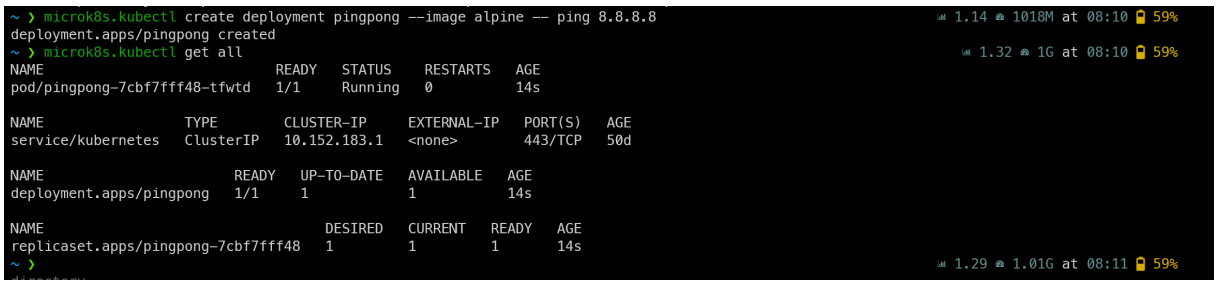
A Docker compose file is used to define and run multi-container Docker applications. It is a YAML file that defines how Docker containers should behave in production. It is a YAML file that defines how Docker containers should behave in production.

4 Part 2: Running our first containers on Kubernetes

For this part, I will use **microk8s**, a lightweight Kubernetes that works on Linux, Windows, and macOS.

4.1 Starting a simple pod with `kubectl run` or `kubectl create`

To create a deployment with a single pod that will ping 8.8.8.8 enter the following:



- `kubectl run` is a command in Kubernetes that is used to run a particular image on the cluster. It is a command that is used to run a particular image on the cluster.
- `kubectl create` is a command in Kubernetes that is used to create a resource from a file or from stdin. It is a command that is used to create a resource from a file or from stdin.

A Namespace is a virtual cluster backed by the same physical cluster.

A Pod is a group of one or more containers, with shared storage/network, and a specification for how to run the containers.

A Deployment is a Kubernetes object that is used to describe the desired state of a set of pods.

A Service is a Kubernetes object that is used to expose an application running on a set of pods as a network service.

A ReplicaSet is a Kubernetes object that is used to ensure that a specified number of pod replicas are running at any given time.

4.2 Viewing container output

```
~ ❯ microk8s.kubectl logs pod/pingpong-7cbf7fff48-tfwtd                    1.29  1.01G at 08:11  59%
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=114 time=123.267 ms
64 bytes from 8.8.8.8: seq=1 ttl=114 time=121.550 ms
64 bytes from 8.8.8.8: seq=2 ttl=114 time=121.184 ms
64 bytes from 8.8.8.8: seq=3 ttl=114 time=120.898 ms
64 bytes from 8.8.8.8: seq=4 ttl=114 time=120.297 ms
64 bytes from 8.8.8.8: seq=6 ttl=114 time=122.644 ms
64 bytes from 8.8.8.8: seq=7 ttl=114 time=123.341 ms
64 bytes from 8.8.8.8: seq=8 ttl=114 time=123.293 ms
64 bytes from 8.8.8.8: seq=9 ttl=114 time=121.275 ms
64 bytes from 8.8.8.8: seq=10 ttl=114 time=123.659 ms
64 bytes from 8.8.8.8: seq=11 ttl=114 time=127.032 ms
64 bytes from 8.8.8.8: seq=12 ttl=114 time=121.725 ms
64 bytes from 8.8.8.8: seq=13 ttl=114 time=136.729 ms
64 bytes from 8.8.8.8: seq=14 ttl=114 time=125.003 ms
64 bytes from 8.8.8.8: seq=15 ttl=114 time=122.433 ms
64 bytes from 8.8.8.8: seq=17 ttl=114 time=124.463 ms
64 bytes from 8.8.8.8: seq=18 ttl=114 time=122.762 ms
64 bytes from 8.8.8.8: seq=19 ttl=114 time=123.188 ms
64 bytes from 8.8.8.8: seq=20 ttl=114 time=126.923 ms
64 bytes from 8.8.8.8: seq=21 ttl=114 time=126.365 ms
64 bytes from 8.8.8.8: seq=22 ttl=114 time=122.524 ms
64 bytes from 8.8.8.8: seq=23 ttl=114 time=122.956 ms
64 bytes from 8.8.8.8: seq=24 ttl=114 time=123.448 ms
64 bytes from 8.8.8.8: seq=26 ttl=114 time=121.985 ms
64 bytes from 8.8.8.8: seq=27 ttl=114 time=120.836 ms
64 bytes from 8.8.8.8: seq=28 ttl=114 time=120.534 ms
64 bytes from 8.8.8.8: seq=29 ttl=114 time=130.788 ms
64 bytes from 8.8.8.8: seq=30 ttl=114 time=122.018 ms
64 bytes from 8.8.8.8: seq=31 ttl=114 time=144.174 ms
64 bytes from 8.8.8.8: seq=32 ttl=114 time=123.157 ms
64 bytes from 8.8.8.8: seq=33 ttl=114 time=122.517 ms
64 bytes from 8.8.8.8: seq=34 ttl=114 time=120.720 ms
```

4.3 Streaming logs in realtime

```
~ ❯ microk8s.kubectl logs pod/pingpong-7cbf7fff48-tfwtd --tail 1 --follow    0.98  1G at 08:14  59%
64 bytes from 8.8.8.8: seq=250 ttl=114 time=123.115 ms
64 bytes from 8.8.8.8: seq=251 ttl=114 time=121.100 ms
64 bytes from 8.8.8.8: seq=252 ttl=114 time=125.732 ms
```

4.4 Scaling our application

To scale up the deployment to three pods running in parallel, we use the “scale deployment” command

```
~ ❯ microk8s.kubectl scale deployment.apps/pingpong --replicas 8          ✘ INT took 21s  0.96  1G at 08:15  59%
deployment.apps/pingpong scaled
~ ❯ microk8s.kubectl get deployments                                      1.27  1018M at 08:18  59%
NAME       READY   UP-TO-DATE   AVAILABLE   AGE
pingpong   4/8     8            4           8m16s
~ ❯                                                                       1.45  993M at 08:19  59%
```

Now, what would happen if we delete a pod from the deployment?

```
~ ❯ microk8s.kubectl get po                                                                                    1.45  993M at 08:19  59%

NAME                       READY   STATUS    RESTARTS   AGE
pingpong-7cbf7fff48-tfwtd   1/1     Running   0          9m31s
pingpong-7cbf7fff48-6bz7z   1/1     Running   0          94s
pingpong-7cbf7fff48-66k75   1/1     Running   0          94s
pingpong-7cbf7fff48-sqls6   1/1     Running   0          94s
pingpong-7cbf7fff48-c5gl8   1/1     Running   0          94s
pingpong-7cbf7fff48-jns2m   1/1     Running   0          94s
pingpong-7cbf7fff48-ts7b9   1/1     Running   0          94s
pingpong-7cbf7fff48-kqr6q   1/1     Running   0          94s
~ ❯                                                                                    took 5s  1.53  990M at 08:20  59%
~ ❯ microk8s.kubectl delete pod pingpong-7cbf7fff48-sqls6                                       1.53  989M at 08:20  59%
pod "pingpong-7cbf7fff48-sqls6" deleted
~ ❯ microk8s.kubectl get po                                                            took 31s  1.58  979M at 08:21  59%
NAME                       READY   STATUS    RESTARTS   AGE
pingpong-7cbf7fff48-tfwtd   1/1     Running   0          10m
pingpong-7cbf7fff48-6bz7z   1/1     Running   0          2m27s
pingpong-7cbf7fff48-66k75   1/1     Running   0          2m27s
pingpong-7cbf7fff48-c5gl8   1/1     Running   0          2m27s
pingpong-7cbf7fff48-jns2m   1/1     Running   0          2m27s
pingpong-7cbf7fff48-ts7b9   1/1     Running   0          2m27s
pingpong-7cbf7fff48-kqr6q   1/1     Running   0          2m27s
pingpong-7cbf7fff48-jnd55   1/1     Running   0          33s
~ ❯                                                                                    1.58  974M at 08:21  59%
```

We can see that the pod we are deleting is gone, and another one has been created in the last 33 seconds; this is happening because we have instructed Kubernetes, no matter what, always to have a deployment with eight pods.

4.5 Cleaning up

```
~ ❯ microk8s.kubectl delete deployment.apps/pingpong                                           1.58  974M at 08:21  59%
deployment.apps "pingpong" deleted
~ ❯ microk8s.kubectl get all                                                                   1.46  977M at 08:24  59%
NAME                           READY   STATUS        RESTARTS   AGE
pod/pingpong-7cbf7fff48-jnd55   1/1     Terminating   0          3m42s
pod/pingpong-7cbf7fff48-6bz7z   1/1     Terminating   0          5m36s
pod/pingpong-7cbf7fff48-c5gl8   1/1     Terminating   0          5m36s
pod/pingpong-7cbf7fff48-66k75   1/1     Terminating   0          5m36s
pod/pingpong-7cbf7fff48-jns2m   1/1     Terminating   0          5m36s
pod/pingpong-7cbf7fff48-ts7b9   1/1     Terminating   0          5m36s
pod/pingpong-7cbf7fff48-kqr6q   1/1     Terminating   0          5m36s
pod/pingpong-7cbf7fff48-tfwtd   1/1     Terminating   0          13m

NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.152.183.1   <none>        443/TCP   50d
~ ❯                                                                                    1.46  982M at 08:24  59%
```

In few seconds, everything will be gone!

```
~ ❯ microk8s.kubectl get all                                                                   1.46  982M at 08:24  59%
NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.152.183.1   <none>        443/TCP   50d
~ ❯                                                                                    1.52  1.01G at 08:27  59%
```

5 Part 3: Exposing Containers

5.1 Service Types

- ClusterIP service is a service issues a virtual IP address that is only accessible from within the cluster. It is a service that is used to expose a set of Pods as a network service. It is a service that is used to expose a set of Pods as a network service.
- NodePort service is a service that exposes the application on a port on each node in the cluster. It is a service that is used to expose a set of Pods as a network service. It is a service that is used to expose a set of Pods as a network service.
- LoadBalancer service is a service that exposes the application using a cloud provider's load balancer. It is a service that is used to expose a set of Pods as a network service. It is a service that is used to expose a set of Pods as a network service.
- ExternalName service is a service that maps a service to a DNS name. It is a service that is used to expose a set of Pods as a network service. It is a service that is used to expose a set of Pods as a network service.

5.2 Running containers with open ports

Let's use a yaml file this time, and set replicas to 4

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: elastic-deployment
  labels:
    app: elastic
spec:
  replicas: 4
  selector:
    matchLabels:
      app: elastic
  template:
```

```
    metadata:
      labels:
        app: elastic
    spec:
      containers:
      - name: elastic
        image: docker.elastic.co/elasticsearch/elasticsearch:7.2.0
        ports:
        - containerPort: 9200
        - containerPort: 9300
        env:
          - name: discovery.type
            value: "single-node"
```

Since it is taking some time and resources, we actually set replicas to 1 for our tests.

```
~/test ❯ microk8s.kubectl get all                                                    1.22  1.26G at 09:16  59%
NAME                                          READY   STATUS    RESTARTS   AGE
pod/elastic-deployment-64bbd55fc6-f4mjb   1/1     Running   0          9m50s

NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.152.183.1   <none>        443/TCP   50d

NAME                                     READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/elastic-deployment   1/1     1            1           9m50s

NAME                                                DESIRED   CURRENT   READY   AGE
replicaset.apps/elastic-deployment-64bbd55fc6   1         1         1       9m50s
```

5.3 Exposing our deployment

```
~/test ❯ microk8s.kubectl expose deployment.apps/elastic-deployment --port 9200      2.31  444M at 09:21  59%
service/elastic-deployment exposed
~/test ❯                                                                             2.78  459M at 09:21  59%
```

Let's check!

```
~/test ❯ IP=$(microk8s.kubectl get svc elastic-deployment -o go-template --template '{{ .spec.clusterIP }}')   2.28  439M at 09:24  59%
~/test ❯ echo $IP                                                                    2.28  436M at 09:24  59%
10.152.183.122
~/test ❯ curl http://$IP:9200/                                                       2.31  436M at 09:24  59%
{
  "name" : "elastic-deployment-64bbd55fc6-f4mjb",
  "cluster_name" : "docker-cluster",
  "cluster_uuid" : "_fOIzvT4TEWTr2fWSRje3g",
  "version" : {
    "number" : "7.2.0",
    "build_flavor" : "default",
    "build_type" : "docker",
    "build_hash" : "508c38a",
    "build_date" : "2019-06-20T15:54:18.811730Z",
    "build_snapshot" : false,
    "lucene_version" : "8.0.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
~/test ❯                                                                             2.31  440M at 09:24  59%
```

5.4 Cleaning up

```
~/test ❯ microk8s.kubectl delete deployment.apps/elastic-deployment                                  2.31  440M at 09:24  59%
deployment.apps "elastic-deployment" deleted
~/test ❯ microk8s.kubectl get all                                                                    1.83  449M at 09:28  59%
NAME                         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/kubernetes           ClusterIP   10.152.183.1     <none>        443/TCP    50d
service/elastic-deployment   ClusterIP   10.152.183.122   <none>        9200/TCP   6m58s
~/test ❯ microk8s.kubectl get all                                                                    1.85  1.39G at 09:28  59%
NAME                         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/kubernetes           ClusterIP   10.152.183.1     <none>        443/TCP    50d
service/elastic-deployment   ClusterIP   10.152.183.122   <none>        9200/TCP   7m6s
~/test ❯ microk8s.kubectl delete service/elastic-deployment                                          1.81  1.4G at 09:28  59%
service "elastic-deployment" deleted
~/test ❯ microk8s.kubectl get all                                                                    1.82  1.39G at 09:28  59%
NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.152.183.1   <none>        443/TCP   50d
~/test ❯                                                                                             1.82  1.38G at 09:29  59%
ancestor directory
```

6 Part 4: Registries

6.1 Docker

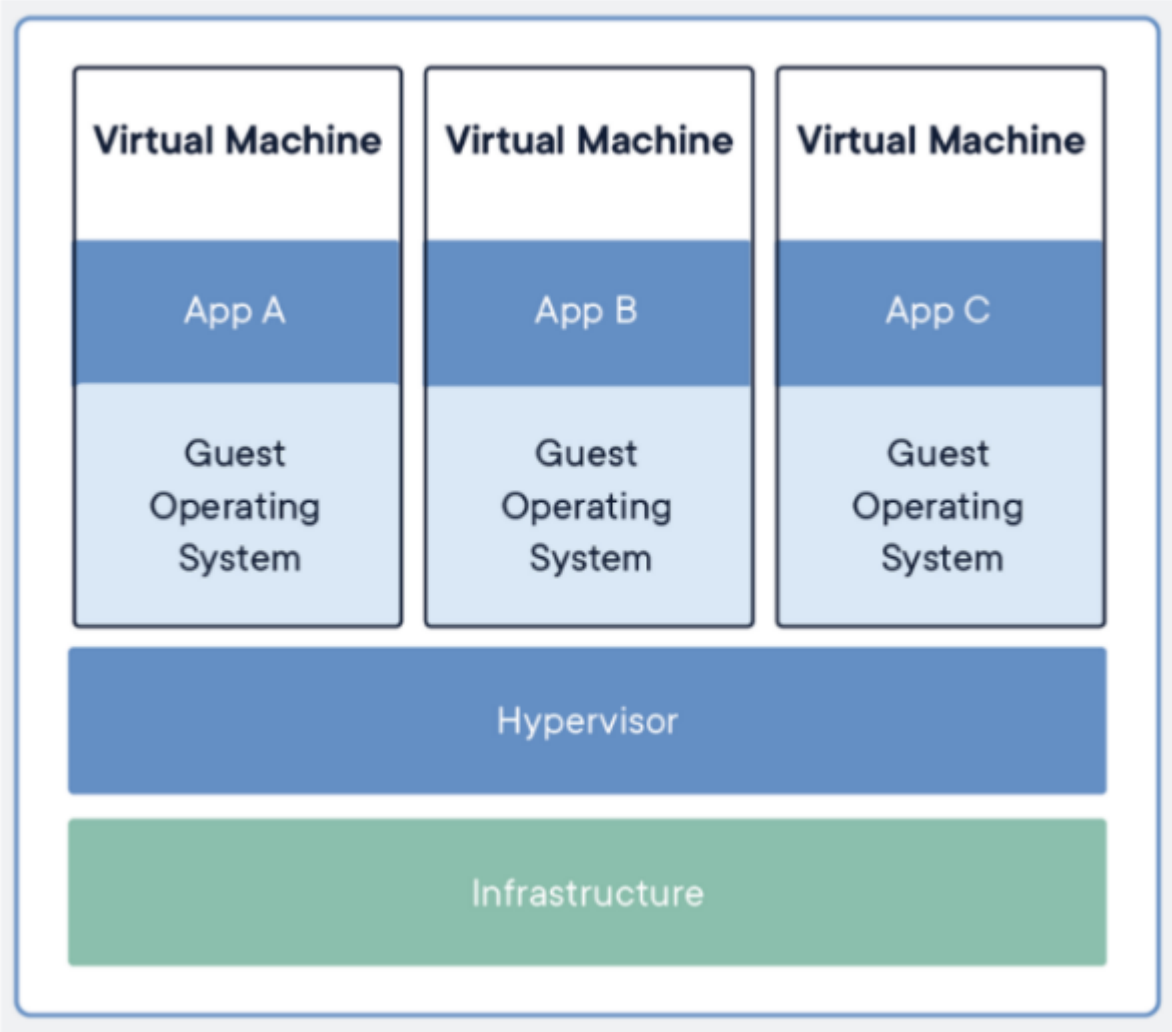
Docker is a tool that can package software into containers that run reliably in any environment. But What is a container? And why do you need one?

6.2 Virtual Machines vs Containers

6.2.1 Virtual Machines

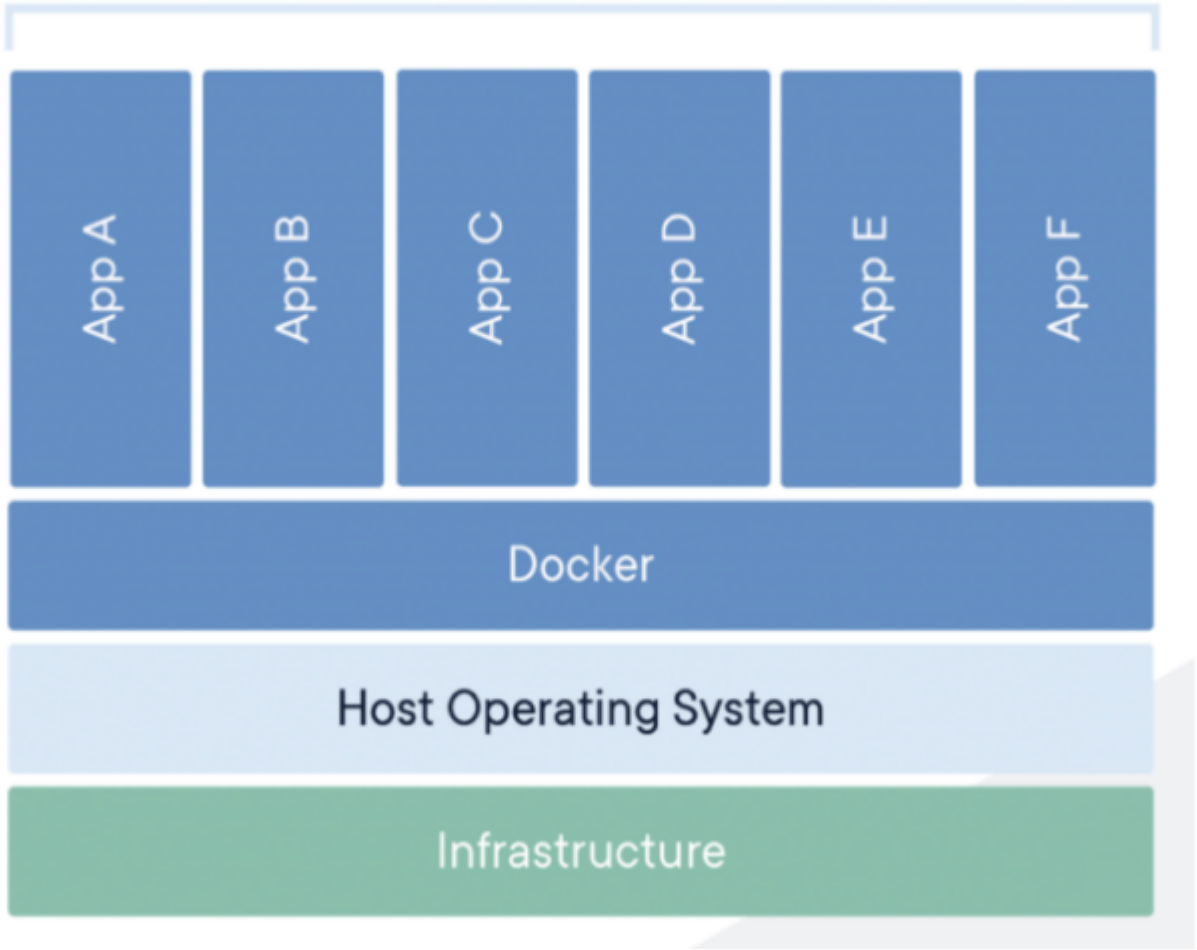
A virtual machine is the virtualization/emulation of a computer system. Virtual machines are based on computer architectures and provide the functionality of a physical computer. Their implementations may involve specialized hardware, software, or a combination. In brief, VMs Isolate applications and allocate resources to run that application.

- VMs can be shared as images.
- Aren't dependent on the Host OS.
- Multiple VMs can be run simultaneously using a Hypervisor.



6.2.2 Containers

A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.



6.2.3 Why use Docker?

- Develop applications that work on any OS.
- Easy to share applications among teams.
- Easy to scale across multiple servers.
- Large applications can be broken into multiple containers – one for each microservice.
- A great solution for Cloud Computing.
- Big community and library of Docker Images.

6.2.4 Docker Files And Docker Images

A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using docker build users can create an automated build that executes several command-line instructions in succession.

Docker images are read-only templates that you build from a set of instructions written in your Dockerfile. Docker images are the basis of containers.

Docker containers run off Docker images.

6.3 Building and pushing images to DockerHub

```
~/dockercoins/stacks on master ❯ export USERNAME=adamlahbib                                  1.03 1.32G at 09:41 59%
~/dockercoins/stacks on master ❯ docker-compose -f dockercoins.yml build                    0.94 1.33G at 09:42 59%
redis uses an image, skipping
Building rng
Sending build context to Docker daemon  3.584kB
Step 1/5 : FROM python:3-alpine
3-alpine: Pulling from library/python
63b65145d645: Pull complete
781eddb6f342: Pull complete
1ae1f1ea756f: Pull complete
5b3e7c82c61d: Pull complete
369973018634: Pull complete
Digest: sha256:84630610c68e7c97384bc6e10f5490ab7b8398c30cdfffefa139ae20c3407cda
Status: Downloaded newer image for python:3-alpine
 ---> 85da6554f50c
```

```
~/dockercoins/stacks on master ❯ docker-compose -f dockercoins.yml push          took 10m 50s 0.73 1.27G at 10:22 59%
Pushing rng (adam/rng:latest)...
The push refers to repository [docker.io/adam/rng]
aea5ac9a48d4: Preparing
4c530265ab91: Preparing
8ab058a1a5ad: Preparing
db7c33b45b7b: Preparing
6c173e500bd5: Preparing
f7cd9720fc7e: Waiting
7cd52847ad77: Waiting
```

6.4 Deploying all the things

```
~/dockercoins/stacks on master ❯ for SERVICE in hasher rng webui worker; do                  1.51 1.26G at 10:54 59%
  microk8s.kubectl run $SERVICE --image=$USERNAME/$SERVICE -l app=$SERVICE
done
pod/hasher created
pod/rng created
pod/webui created
pod/worker created
```

```
~/dockercoins/stacks on master ❯ microk8s.kubectl get all                                   2.04 1.21G at 11:20 59%
NAME         READY   STATUS    RESTARTS   AGE
pod/redis    1/1     Running   0          36m
pod/rng      1/1     Running   0          5m4s
pod/hasher   1/1     Running   0          3m49s
pod/webui    1/1     Running   0          3m33s
pod/worker   1/1     Running   0          3m14s

NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.152.183.1   <none>        443/TCP   50d
~/dockercoins/stacks on master ❯                                                            1.99 1.18G at 11:20 59%
directory
```

6.5 Is this working?

Let's check `rng` logs

```
~/dockercoins/stacks on master ❯ microk8s.kubectl logs pod/rng                                  1.90 1.19G at 11:21 59%
 * Serving Flask app 'rng' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://10.1.205.237:80/ (Press CTRL+C to quit)
```

6.6 Exposing Services

6.6.1 Exposing Services Internally

```
~/dockercoins/stacks on master ❯ microk8s.kubectl expose pod redis --port 6379                  1.51 1.2G at 11:22 59%
service/redis exposed
~/dockercoins/stacks on master ❯ microk8s.kubectl expose pod rng --port 80                      1.54 1.2G at 11:23 59%
service/rng exposed
~/dockercoins/stacks on master ❯ microk8s.kubectl expose pod hasher --port 80                   1.70 1.19G at 11:23 59%
service/hasher exposed
~/dockercoins/stacks on master ❯                                                               1.73 1.18G at 11:23 59%
```

6.6.2 Exposing services for external access

```
~/dockercoins/stacks on master ❯ microk8s.kubectl create service nodeport webui --tcp=80 --node-port=30001
service/webui created
~/dockercoins/stacks on master ❯ microk8s.kubectl get svc                                       1.68 1.19G at 11:25 59%
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.152.183.1     <none>        443/TCP        50d
redis        ClusterIP   10.152.183.63    <none>        6379/TCP       2m48s
rng          ClusterIP   10.152.183.49    <none>        80/TCP         2m36s
hasher       ClusterIP   10.152.183.162   <none>        80/TCP         2m30s
webui        NodePort    10.152.183.209   <none>        80:30001/TCP   18s
~/dockercoins/stacks on master ❯ curl http://10.152.183.209                                     1.62 1.18G at 11:25 59%
Found. Redirecting to /index.html
~/dockercoins/stacks on master ❯ curl http://10.152.183.209/index.html                          1.54 1.2G at 11:26 59%
<!doctype html>
<html>
<head>
<title>DockerCoin Miner WebUI</title>
<link rel="stylesheet" type="text/css" href="rickshaw.min.css">
<style>
#graph {
    background-color: #eee;
    width: 800px;
    height: 400px;
}
#tweet {
    color: royalblue;
}
</style>
<script src="jquery-1.11.3.min.js"></script>
<script src="d3.min.js"></script>
<script src="rickshaw.min.js"></script>
<script>
```

7 Conclusion

In this lab, we will learn how to deploy a simple application on Kubernetes. We will learn how to create a Kubernetes cluster, deploy an application, and expose it to the outside world. We will also learn how to scale the application and update it with zero downtime.

We also learned how to build and push images to DockerHub, and deploy them on Kubernetes.