AVIATOR

A demo application to learn about containers and orchestration engines

Just a silly application to iterate with

NAVIGATION

? is for Help

Escape is for overview

Arrows bottom right tell where to go next

PRE-REQUISITES

We need some software available

Several options

LAUNCH A VM

All batteries included

```
$ openstack server create --image UUID \
--flavor m2.medium --key-name YOURKEY YOURID-handson

$ ssh root@YOURID-handson
```

MANUAL INSTALL

Docker

```
$ yum install -y yum-utils  device-mapper-persistent-data lvm2
$ yum-config-manager --add-repo https://download.docker.com/li
$ yum install -y docker-ce
$ systemctl start docker
```

Kubernetes

```
$ curl -LO https://storage.googleapis.com/kubernetes-release/r
$ mv kubectl ~/docker
$ chmod 755 ~/docker/kubectl
```

Helm

```
$ wget https://kubernetes-helm.storage.googleapis.com/helm-v2.
$ tar zxvf helm-v2.7.2-linux-amd64.tar.gz
$ mv linux-amd64/helm ~/docker
```

CLONE THE REPOSITORY

Fork the repository in your personal gitlab

https://gitlab.cern.ch/cloud-infrastructure/aviator

Clone the repository locally

```
$ git clone https://:@gitlab.cern.ch:8443/YOURID/aviator.git
```

TOPICS

- A Bit of Theory
- Docker Basics
- OpenStack Magnum
- Kubernetes
- Application Lifecycle
- Other

A BIT OF THEORY

REPRODUCIBILITY FOR DEVELOPMENT ENVIRONMENTS AND DEPLOYMENTS

- Package all software dependencies
  - eg Python and C library dependencies
- Standardize packaging accross platforms/archritectures
- Manage processes instead of machines

CONTAINERS AND CONTAINER IMAGES

- Applications and their dependencies are packaged in tarballs, "docker/container images"
  - Docker v2 format https://docs.docker.com/registry/spec/manifest-v2-2/
  - OCI format https://github.com/opencontainers/image-spec
- Applications run as "Linux Containers"
  - Leverage features of the Linux Kernel

ISOLATION

- Limit what a process can see
  - Network
  - Files
  - Other PIDs
- Limit what a process can do
  - Change file ownership
  - Change process user id
  - Bind to a port
  - Trace calls
  - ...

NAMESPACES

- Feature of the Linux kernel that partitions kernel resources such that one set of processes sees one set of resources while another set of processes sees a different set of resources
- Mount, PID, Network, IPC, UTS, User, Time, Cgroup
- https://man7.org/linux/man-pages/man7/namespaces.7.html

CAPABILITIES

- Linux capabilities are special attributes in the Linux kernel that grant processes and binary executables specific privileges that are normally reserved for processes whose effective user ID is 0 (The root user, and only the root user, has UID 0).
- CAP_CHOWN, CAP_NET_BIND_SERVICE, CAP_SETUID, CAP_SETGID, ...
- https://man7.org/linux/man-pages/man7/capabilities.7.html

RESOURCE REQUESTS AND LIMITS

- Ensure a process has enough resources and won't starve the unedelying host/HW
  - CPU shares
  - Memory
  - PIDs

CGROUPS

- Linux kernel feature that limits, accounts for, and isolates the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes
- cpu, memory, pids ...
- https://man7.org/linux/man-pages/man7/cgroups.7.html

DEMO

Useful Links

https://github.com/lizrice/containers-from-scratch

(https://github.com/strigazi/containers-from-scratch)

https://ericchiang.github.io/post/containers-from-scratch/

DOCKER BASICS

It all starts with a run

```
$ docker run -it alpine sh
/ #
```

We're now in a different process world

```
/ # ps auxw
PID   USER     TIME   COMMAND
    1 root       0:00 sh
    6 root       0:00 ps auxw
```

And filesystem, even networking

```
/ # cat /etc/passwd
/ # ip addr
```

Though we do share a kernel

```
/ # uname -a
Linux 56b4c4d6d6d8 3.10.0-693.21.1.el7.x86_64 #1 SMP Wed Mar 7
```

When we exit, it's gone... right?

```
/ # exit
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CR
```

Maybe not

```
$ docker ps --all
CONTAINER ID        IMAGE               COMMAND               CR
29c24c5847f9        alpine                                  "sh"
$ docker rm -f 29c24c5847f9
```

A service would look similar

```
$ docker run --name mynginx -p 1234:80 -d nginx
```

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND
298dc0089df9        nginx               "nginx -g 'daemon ..."
```

Should be really there

```
$ curl http://localhost:1234
```

And we can kill it

```
$ docker rm -f mynginx
```

Host networking can be of use, sometimes

```
$ docker run --net=host --name mynginx -d nginx
```

No network namespace, listening on the host

```
$ curl http://localhost
```

And cleanup as usual

```
$ docker rm -f mynginx
```

CONTAINER REGISTRIES

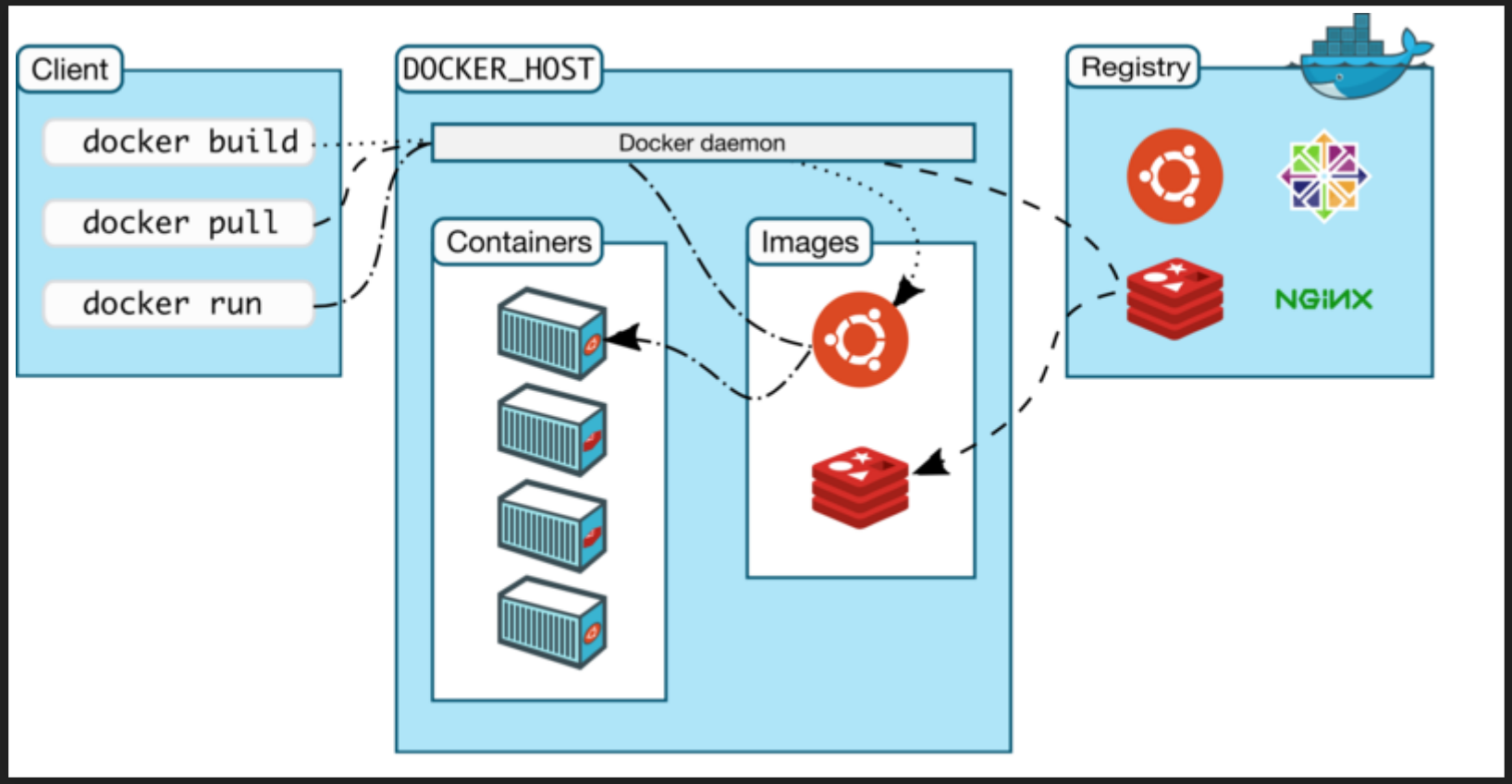
- DockerHub https://hub.docker.com/explore/
- Quay.io https://quay.io/search
- CERN Registry https://registry.cern.ch/

DOCKER IMAGES

Application units for sharing and deployment

- Dockerfile
- Layered
- Hosted locally or in shared online repositories

Architecture



DOCKERFILE

One command == One layer

```
FROM golang:1.9.2

WORKDIR /

ADD main.go index.html aviator.png /

RUN go get -d -v .

RUN go build .

CMD ["/aviator"]
```

Dockerfile

BUILD

golang was our base image, we build a new one on top

```
$ docker build -t aviator .

$ docker images
REPOSITORY          TAG                 IMAGE ID            CR
aviator             latest              22b01c5e71cb        48
golang              1.9.2               1a34fad76b34        4
```

INSPECT

Where are my layers?

```
$ docker images --all
REPOSITORY          TAG                 IMAGE ID            CREATE
none                none                85e6e677bb7f        13 sec
aviator             latest              256b23989de5        13 sec
none                none                b71e4c2bbf47        22 sec
none                none                f8205be1377c        23 sec
none                none                b54be6539976        23 sec
golang              1.9.2               1a34fad76b34        4 week
```

TAGGING

'latest' is just the default when nothing is given

Same build? Same image id, no rebuild

```
$ docker build -t aviator:mytag .

$ docker images
REPOSITORY          TAG                 IMAGE ID            CR
aviator             latest              22b01c5e71cb        2
aviator             mytag               22b01c5e71cb        2
golang              1.9.2               1a34fad76b34        4
```

MULTI-STAGE BUILDS

Reuse previous builds (COPY --from=builder)

```
FROM golang:1.9.2 AS builder
WORKDIR /
ADD main.go /
RUN go get -d -v .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o
CMD ["/aviator"]

FROM alpine:latest
WORKDIR /
RUN apk --no-cache add ca-certificates
ADD index.html aviator.png /
COPY --from=builder /aviator /
CMD ["/aviator"]
```

Dockerfile

SMALL RUNTIME IMAGES

740MB vs 11MB for the runtime image (!!)

```
$ docker build -t aviator:small -f Dockerfile.multi .

$ docker images
REPOSITORY          TAG                 IMAGE ID            CR
aviator             small               3caadbeed7b5        2
none                none                00c4fd9c5705        2
aviator             latest              256b23989de5        2
aviator             mytag               256b23989de5        2
alpine              latest              e21c333399e0        9
golang              1.9.2               1a34fad76b34        5
```

RUN

Run locally, interactivily

```
$ docker run --rm --name aviator -p 80:80 aviator:small
```

Or in the background (-d), and give it a name

```
$ docker run -d --rm --name aviator -p 80:80 aviator:small
a85efd4fa0eeef45a89051ab2426e143ac54ad3cbb1505552e08402ff4d377

$ docker ps
CONTAINER ID        IMAGE               COMMAND             CR
a85efd4fa0ee        aviator:small       "/aviator"          4
```

Check it at http://localhost

ACCESS

Run a command in a running container, even a shell

```
$ docker exec aviator ps auxw

$ docker exec aviator ip addr

$ docker exec -it aviator sh
/ #
```

LOGGING

If the application writes to stdout/err, it's easy as

```
$ docker logs -f aviator

2017/12/08 14:04:13 next /root/next
2017/12/08 14:04:14 next /root/next
2017/12/08 14:04:15 next /root/next
2017/12/08 14:04:16 next /root/next
```

DESTROY

-f will also stop it if it was running

```
$ docker rm -f aviator

$ docker ps
CONTAINER ID        IMAGE               COMMAND             CR
```

PUSHING

Share your images on public repositories

If not given, docker.io (dockerhub) is assumed

```
$ docker build -t registry.cern.ch/YOURID/aviator:latest .

$ docker login registry.cern.ch

$ docker push registry.cern.ch/YOURID/aviator:latest
```

Check it in gitlab https://gitlab.cern.ch/YOURID/aviator

CLEANUP

Prune gets rid of local containers, caches, ...

```
$ docker system prune
```

Option -a to prune also images (not right now)

EXERCISE 1.1

Run a second aviator instance on port 9999

http://localhost:9999

EXERCISE 1.2

Check the stats of running containers

TIP: docker --help

EXERCISE 1.3

Check the history of an existing image

TIP: docker --help

EXERCISE 1.4

Copy a file to or from a running container

TIP: docker --help

OPENSTACK MAGNUM

Container Orchestration Engine

```
$ . Personal\ rbritoda-openrc.sh

$ openstack coe cluster template list

| 0996cb4f-cf85-419c-b30b-c91e3e5e3739 | kubernetes-1.19.6-1
| 06e9283f-1326-4591-931e-d13c03033bdf | kubernetes-1.19.6-1-m
| 200327fe-10b0-4685-96c1-ecda717638c2 | kubernetes-1.20.4-4
| 262992e0-459b-4183-a8bf-08e8af079c17 | kubernetes-1.20.4-4-m
| c49e8156-46be-42db-a6ff-66b7e9c96532 | kubernetes-1.21.1-1
| 79b48601-74e6-4f5f-883b-ac2f37fbbc89 | kubernetes-1.21.1-1-m
```

CLUSTER TEMPLATES

```
$ openstack coe cluster template show kubernetes-1.21.1-1 --fi
        | master_flavor_id      | m2.medium
        | flavor_id             | m2.medium
        | labels                | {u'kube_tag': u'v1.21.1-1',
```

CREATE

Let's create a kubernetes cluster

```
$ openstack coe cluster create \
    --cluster-template kubernetes-1.21.1-1 \
    --node-count 1 \
    --keypair YOURKEYPAIR \
    YOURID-handson-kub

Request to create cluster 615aa816-0ec5-4be9-a205-a9cdcccc0554

$ openstack coe cluster list
+--------------------------------------+-----------------------
| uuid                                 | name
+--------------------------------------+-----------------------
| 615aa816-0ec5-4be9-a205-a9cdcccc0554 | YOURID-handson-kub
+--------------------------------------+-----------------------
```

ACCESS

Config to fetch credentials

```
$ mkdir -p ~/clusters/YOURID-handson-kub
$ cd ~/clusters/YOURID-handson-kub
$ openstack coe cluster config YOURID-handson-kub
```

Access using native clients

```
$ export KUBECONFIG=`pwd`/config

$ kubectl get node
NAME                                        STATUS    ROLES
YOURID-handson-kub-6dc43wjvbdcv-minion-0    Ready     none
```

SCALE

Up or down, change the node_count

```
$ openstack coe cluster update YOURID-handson-kub replace node
Request to update cluster YOURID-handson-kub has been accepted

$ openstack coe cluster list
| b34d0745-ade8-4d0a-94ba-96febf9e30fd | YOURID-handson-kub
```

LABELS

Enable and disable features, override defaults

Available Labels

```
$ openstack coe cluster create --merge-labels --labels nvidia_
```

NODE GROUPS

Heteregeneous clusters with distinct sets of nodes

```
$ openstack coe nodegroup list YOURID-handson-kub

$ openstack coe nodegroup create YOURID-handson-kub \
    mynodegroup --node-count 1 --flavor m2.large \
    --merge-labels --labels availability_zone=cern-geneva-a \
    --role large
```

EXERCISE 2.1

Check label differences between two different kubernetes templates

KUBERNETES

RESOURCES

Pod, Service, Volume, Namespace

ReplicaSet, Deployment, StatefulSet, DaemonSet

Secret, Job

DEPLOYMENT

Define the application containers and metadata

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: aviator
  labels:
    app: aviator
spec:
  replicas: 5
  selector:
    matchLabels:
      app: aviator
  template:
    metadata:
      labels:
        app: aviator
```

deployment.yaml

SERVICE (L4)

A load balanced entrypoint to an application
(ClusterIP, NodePort, LoadBalancer)

```yaml
apiVersion: v1
kind: Service
metadata:
  name: aviator
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
      name: aviator
  selector:
    app: aviator
```

deployment.yaml

INGRESS (L7)

A HTTP based load balancer entrypoint

```yaml
kind: Ingress
metadata:
  name: aviator
spec:
  rules:
    - host: "*.cern.ch"
      http:
        paths:
          - path: /aviator
            backend:
              serviceName: aviator
              servicePort: 80
```

deployment.yaml

CREATE

Single create command for all manifests

```
$ kubectl create -f kubernetes/deployment.yaml
```

Recording allows later rollback

```
$ kubectl create --record -f kubernetes/deployment.yaml
```

CHECK

Overview of our deployment

```
$ kubectl get deployment
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
aviator   1         1         1            0           20s
```

Or details of a specific resource

```
$ kubectl describe deployment/aviator
```

Check all resources

```
$ kubectl get all
...
```

Check system resources

```
$ kubectl -n kube-system get all
...
```

Access

```
$ kubectl get service
NAME         TYPE           CLUSTER-IP       EXTERNAL-IP
aviator      LoadBalancer   10.254.229.138   137.138.226.89
kubernetes   ClusterIP      10.254.0.1       <none>
```

The service should be available at the given IP:

http://137.138.226.89

SCALE

Deployments are backed by Pods and Replica Sets

```
$ kubectl get deployment
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
aviator   1         1         1            1           7m
```

Which makes it very easy to scale

```
$ kubectl scale --replicas=3 deployment/aviator
```

ISOLATION

With namespaces, run multiple instances of one app

```
$ kubectl create namespace canary
$ kubectl -n canary create -f kubernetes/deployment.yaml
$ kubectl -n canary get deployment
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
aviator   1         1         1            1           20s

$ kubectl get deployment --all-namespaces
```

LIVE CHANGES

Useful, even if better to track them in manifests

```
$ kubectl edit deployment aviator
```

ADVANCED SCHEDULING

Selectors, Taints and Tolerations, Priorities, Affinities, Disruption Budgets

And so much more...

EXERCISE 4.1

Change the number of replicas of aviator to 10

TIP: look for both 'scale' and 'edit' commands

EXERCISE 4.2

Check logs from our aviator

TIP: not very different from the docker command

EXERCISE 4.3

Delete a pod, see that it a new one is launched

TIP: Check the age of all pods to see there's a new one

EXERCISE 4.4

Try to get an interactive shell in one of the pods

TIP: kubectl --help

EXERCISE 4.5

Check if the cluster has an internal DNS

TIP: Ping 'aviator' from inside one of the pods

APPLICATION LIFECYCLE

Helm to improve our flying skills

Manages application on a kubernetes cluster

QuickStart

CHARTS

Go templates for Kubernetes manifests

```
    spec:
      containers:
        - name: {{ .Chart.Name }}
          image: "{{ .Values.image.repository }}:{{ .Values.im
          imagePullPolicy: {{ .Values.image.pullPolicy }}
          ports:
            - containerPort: {{ .Values.service.port }}
```

deployment.yaml

VALUES

Data for deployment configurations

```
replicaCount: 1
image:
  repository: registry.cern.ch/cloud/aviator
  tag: small
  pullPolicy: IfNotPresent
```

values.yaml

DEPLOYMENT

One per application, with versions and history

```
$ helm install aviator helm/aviator

$ helm ls
NAME        NAMESPACE    REVISION    UPDATED
aviator     default      1           2021-07-08 14:20:10.349800
```

LOOPS

Make our aviator loop

```
$ vim main.go
        nextPath := nextPath(current, size, true)
```

Rebuild with a new tag

```
$ docker build -t registry.cern.ch/YOURID/aviator:loops .
$ docker push registry.cern.ch/YOURID/aviator:loops
```

VALIDATE

Check the feature works in a separate instance / namespace

```
$ helm -n loops install aviator helm/aviator --set image.tag=l
```

```
$ vim values.yaml
image:
  repository: registry.cern.ch/YOURID/aviator
  tag: loops

$ helm -n loops install aviator helm/aviator --values values.y
```

```
$ helm ls -A
NAME        NAMESPACE    REVISION    UPDATED
aviator     default      1           2021-07-08 14:20:10.349800
aviator     loops        1           2021-07-08 14:29:27.565208
```

ROLLOUT

Reviewed and checked, rollout to the main instance

```
$ helm upgrade aviator helm/aviator/ --set image.tag=loops
$ helm ls -A
NAME                    REVISION    UPDATED
aviator-production      2           Wed Mar 21 20:45:32 2018
aviator-v2              1           Wed Mar 21 20:42:12 2018

$ helm history aviator
REVISION    UPDATED                     STATUS          CHART
1           Thu Jul  8 14:20:10 2021    superseded      aviator-
2           Thu Jul  8 14:38:42 2021    deployed        aviator-
```

ROLLBACK

Loops make me dizzy...

```
$ helm rollback aviator 1
$ helm history aviator
REVISION    UPDATED                     STATUS        CHART
1           Thu Jul  8 14:20:10 2021    superseded    aviator-
2           Thu Jul  8 14:24:10 2021    superseded    aviator-
3           Thu Jul  8 14:38:42 2021    deployed      aviator-
```

EXERCISE 5.1

Check the deployments, pods, ... that were created

TIP: use the kubernetes client

EXERCISE 5.2

Delete the 'feature' validation deployment from above

TIP: using helm, never use kubectl for helm deployments

EXERCISE 5.3

Update the number of replicas of our default deployment

EXERCISE 5.4

Check the status information of our default deployment

TIP: helm --help

EXERCISE 5.5

Download the current kubernetes manifests of our default deployment

TIP: helm --help

OTHER TOPICS

GitOps, Prometheus

REFERENCES

- Cloud Docs
- CERN Container Webinars
- CNCF Landscape
- Artifact Hub