Openshift

Introduction to the Side Car

Laurent Valeyre

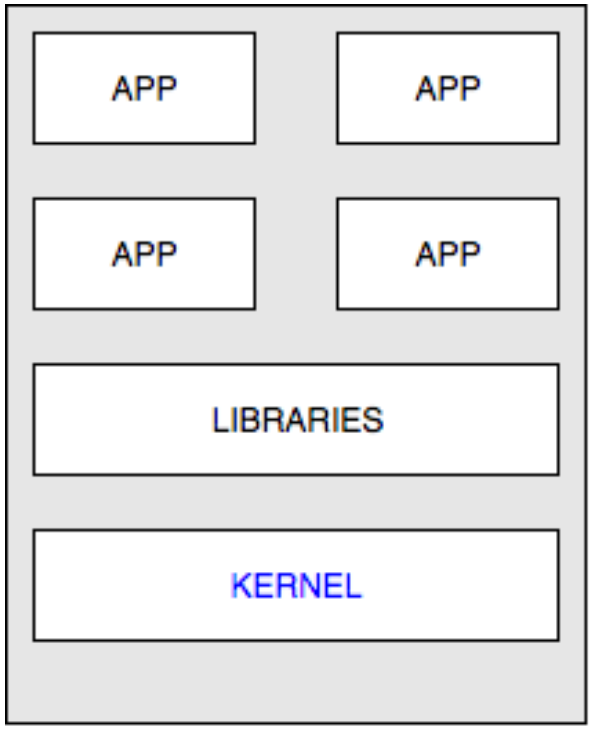
Orange

August 2018

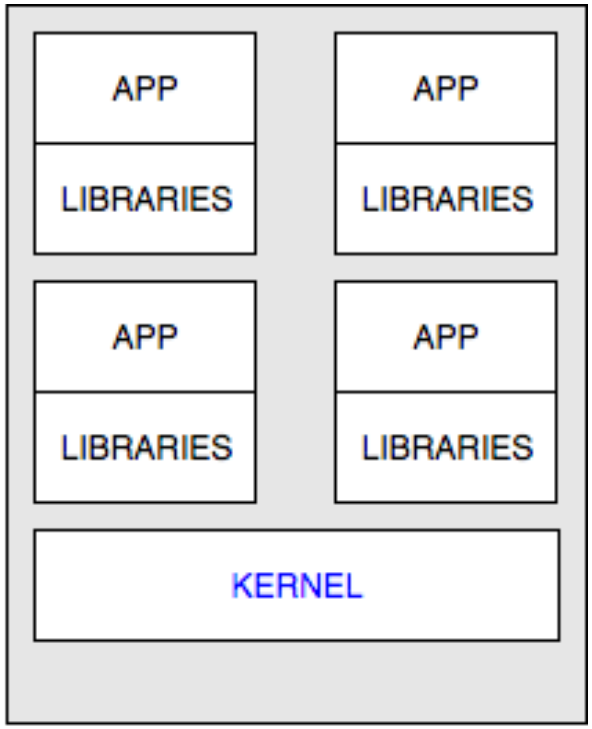
Table of contents

Containers

Figure: Why Containers



Applications on host
heavyweight, non-portable
Relies on OS package manager



Deploy containers
Small and fast, portable
Uses OS-Level virtualization

Understanding Pods

- A *Pod* is the smallest and simplest unit in the Kubernetes object model that we create or deploy.

Understanding Pods

- A *Pod* is the smallest and simplest unit in the Kubernetes object model that we create or deploy.
- A Pod represents a running process on our cluster.

Understanding Pods

- A *Pod* is the smallest and simplest unit in the Kubernetes object model that we create or deploy.
- A Pod represents a running process on our cluster.
  - A *Pod* encapsulates an application container (or, in some cases, multiple containers)

Understanding Pods

- A *Pod* is the smallest and simplest unit in the Kubernetes object model that we create or deploy.
- A Pod represents a running process on our cluster.
  - A *Pod* encapsulates an application container (or, in some cases, multiple containers)
  - A *Pod* storages resources

Understanding Pods

- A *Pod* is the smallest and simplest unit in the Kubernetes object model that we create or deploy.
- A Pod represents a running process on our cluster.
  - A *Pod* encapsulates an application container (or, in some cases, multiple containers)
  - A *Pod* storages resources
  - A *Pod* has a unique network IP

Understanding Pods

- A *Pod* is the smallest and simplest unit in the Kubernetes object model that we create or deploy.
- A Pod represents a running process on our cluster.
  - A *Pod* encapsulates an application container (or, in some cases, multiple containers)
  - A *Pod* storages resources
  - A *Pod* has a unique network IP
- A *Pod* represents a unit of deployment:

Main ways

Pods in a Kubernetes cluster can be used in two main ways:

Main ways

Pods in a Kubernetes cluster can be used in two main ways:

- Pods that run a single container (most common Kubernetes use case)

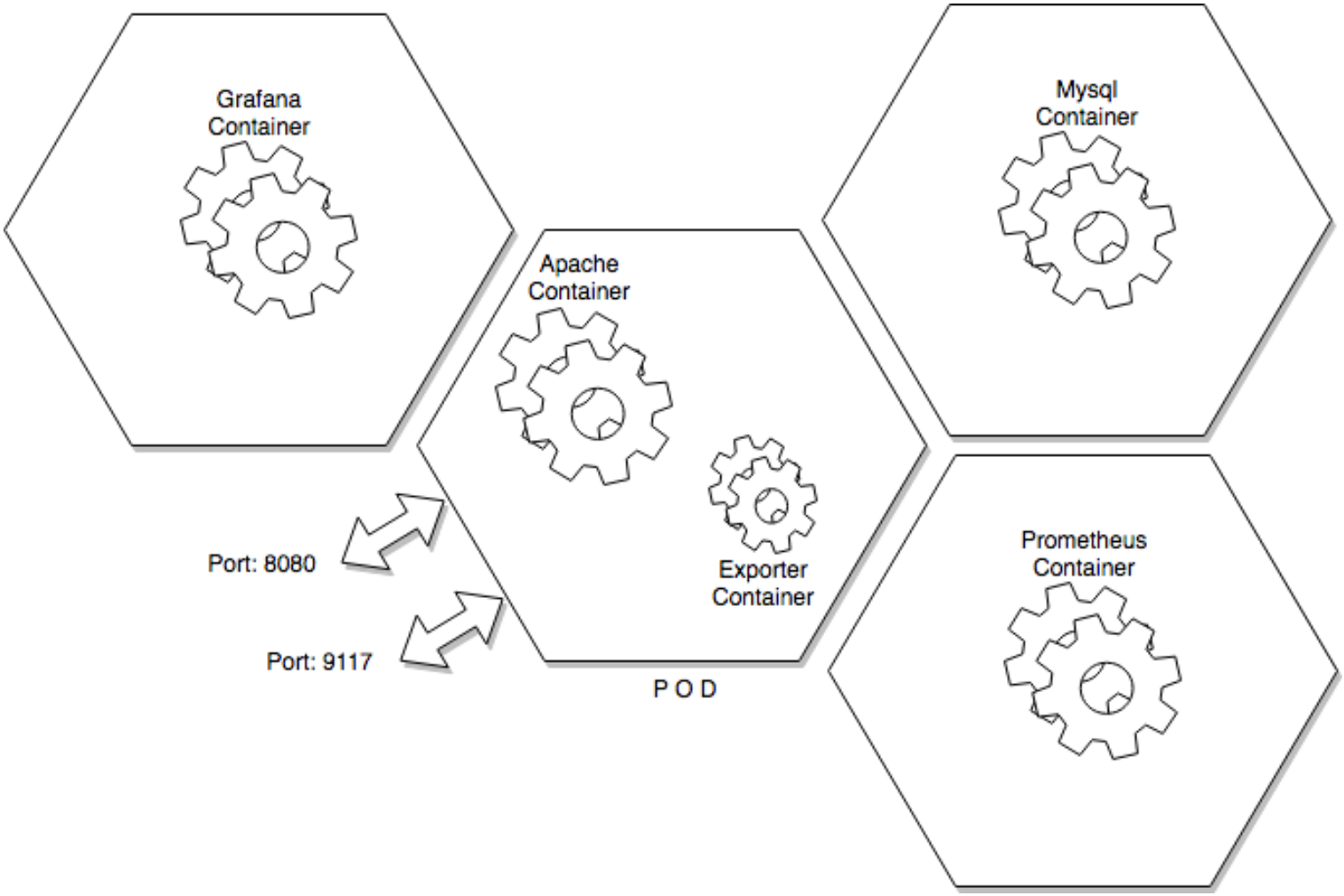
Main ways

Pods in a Kubernetes cluster can be used in two main ways:

- Pods that run a single container (most common Kubernetes use case)
- Pods that run multiple containers that need to work together(encapsulate an application composed of multiple co-located containers that tightly coupled)

Patterns for Composite Containers: Sidecar

Figure: schema of our Sidecar



Apache status

Module to enable the output statistic of *Apache*.

```
<Location /server-status>
  SetHandler server-status
  Order deny,allow
  Allow from all
</Location> ExtendedStatus On>
```

Figure: status.conf

This module has to be copied in the */etc/apache2/mods-enabled/* directory.

Dockerfile

The *Dockerfile* includes the copy of the *Apache* module
Important to add the switching between *root* and *1001* user

```
FROM ubuntu:latest
USER root
...
RUN a2enmod status
COPY status.conf /etc/apache2/mods-enabled/
EXPOSE 8080
USER 1001
CMD ["/usr/sbin/apache2ctl", "-DFOREGROUND"]
```

Figure: Dockerfile

Secret Access

And because the credential of *GITLAB*, we'll use the login/password based on the token initialized in our profile

```
apiVersion: v1
kind: Secret
metadata:
  name: gitlab-secret
  namespace: sidecar
type: kubernetes.io/basic-auth
data:
  username: c3Bpa2U=
  password: dmFsZW50aW5l
```

Figure: gitlab-secret.yaml

Secret Access

The *username* and *password* are encoded to Base64 format. Finally we load the new *secret*

```
$ echo -n 'spike' | base64
c3Bpa2U=
$ echo -n 'valentine' | base64
dmFsZW50aW5l
$ oc create -f gitlab-secret.yaml
```

New Project

It's time to create our new project *sidecar*, similar to a namespace

```
$ oc new-project sidecar \
--display-name='Side Car Project' \
--description='Side Car Project'
```

New Application

It's time to create our application

```
$ oc new-app https://gitlab.forge.orange-labs.fr/\
laov6410/cdnselect.git --name sidecar
$ oc set build-secret --source bc/sidecar gitlab-secret
$ oc expose service sidecar
$ oc get all name --selector app=sidecar
```

Item To Modify

2 parts will be modified to adapted to our application

- DeploymentConfig
- Service

DeploymentConfig

We edit *DeploymentConfig*

```
$ oc edit dc/sidecar
```

and we add

```
spec:
  containers:
  - name: apache-exporter
    image: previousnext/apache-exporter
    command: [ "apache_exporter", "-scrape_uri", \
    "http://127.0.0.1:8080/server-status/?auto" ]
    ports:
    - containerPort: 9117
```

Service

We edit *service*

```
$ oc edit svc/sidecar
```

```
spec:
  ...
  - name: 9117-tcp
    port: 9117
    protocol: TCP
    targetPort: 9117
```

9117 is The port related to *exporter apache*

Finally

Finally we create our new application from this *yaml* file

```
$ oc get svc --selector "app=selector"
NAME       TYPE          CLUSTER-IP       PORT(S)
selector   ClusterIP     172.30.127.98    8080/TCP,9117/TCP
$ oc describe dc/sidecar
```

Et voila...

Pull Image

We pull the image for *Prometheus* from the public Docker Hub registry.

```
$ oc new-app prom/prometheus
$ oc new-app grafana/grafana
$ oc expose service prometheus
$ oc expose service grafana
```

Image and configuration decoupled

```
global:
  scrape_interval:     5s
  evaluation_interval: 5s

scrape_configs:
  - job_name: 'apache-exporter'
    scheme: http
    static_configs:
    - targets: ['faye:9117']
      labels: {'host': 'cOmerade'}
```

Prometheus Configmap

We create the *configmap* including the prometheus configuration and finally edit the *deploymentconfig* prometheus

```
$ oc create configmap prom-config \
--from-file=prometheus.yml
$ oc edit dc/prometheus
```

DeploymentConfig and ConfigMap

Add these 2 blocks of code

```
- name: prom-config-volume
  configMap:
    name: prom-config
    defaultMode: 420
```

```
- name: prom-config-volume
  mountPath: /etc/prometheus/
```

Grafana Configmap

We create the *configmap* including the grafana configuration

```
$ oc create configmap grafana-config \
--from-file=grafana.ini
$ oc edit dc/grafana
```

DeploymentConfig and ConfigMap

Modification in *spec-containers*

```
- name: grafana-config
  configMap:
    name: grafana-config
    defaultMode: 420
```

```
- name: grafana-config
  mountPath: /etc/grafana/
```

links

https://kubernetes.io/blog/2015/06/the-distributed-system-toolkit-patterns/
https://www.robustperception.io/openshift-and-prometheus
http://widerin.net/blog/official-grafana-docker-image-on-openshift/