Sidecar With Openshift

Patterns for composite Containers

Laurent Valeyre

August 2018

Contents

1 The Project

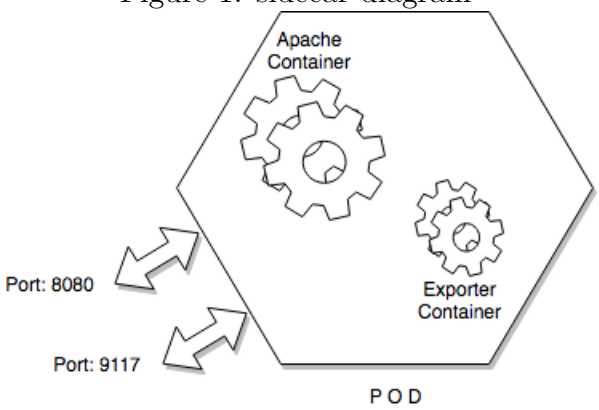
1.1 Introduction

The *Sidecar container* or *adapter container* or *ambassador container* concept provides a separation and focus on services that reduces spaghetti dependencies and untestable components. Building an application from modular containers means thinking about symbiotic groups of containers that cooperate to provide a service, not one container per service. In Kubernetes, the embodiment of this modular container service is a Pod.

A Pod is a group of containers that share resources like file systems, kernel namespaces and an IP address. The Pod is the atomic unit of scheduling in a Kubernetes cluster.

1.2 schema

Figure 1: sidecar diagram



Sidecar containers extends and enhance the *main* container, they take existing container and make them better. As an exemple, consider a container that runs the Apache web server. Add a different container that provide statistics of the system with a exporter and you have built exporter to deploy. But you've done it in a modular manner where the exporter can be built by a different team.

1.3 Status On Apache

status.conf

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

and *Dockerfile*

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

1.4 Secret Access

We firstly define our *secret file*. If the access is based on a login/password. It enables *openshift* to access at the repository to pull sources.

```
apiVersion: v1
kind: Secret
metadata:
  name: gitlab-secret
  namespace: cdnapi
type: kubernetes.io/basic-auth
data:
```

```
  username: c3Bpa2U=
  password: dmFsZW50aW5l
```

username and *password* are defined with the command

```
$ echo -n 'spike' | base64
c3Bpa2U=
$ echo -n 'valentine' | base64
dmFsZW50aW5l
```

and we run

```
$ oc create -f gitlab-secret.yaml
```

In case of our *GITLAB*, we have to use the login and password based on the *Deploy Tokens* generated by our repository.

1.5 New Project

Firstly, we create a new project

```
$ oc new-project cdnapi \
--display-name='CDN API Project' \
--description='CDN API Project'
```

1.6 New Application

It's time to create our application

```
$ oc new-app https://gitlab.forge.orange-labs.fr/\
laov6410/cdnselect.git --name cdnapi
```

```
$ oc set build-secret --source bc/cdnapi gitlab-secret
$ oc expose service cdnapi
$ oc get all name --selector app=cdnapi
```

2 The Patern Sidecar

The solution concists to create the application without the sidecar, and at the end to modify the *DeploymentConfig* and *Service*.

2.1 DeploymentConfig

The *DeploymentConfig* to add the *exporter-apache* image

```
$ oc get dc
$ oc edit dc/cdnapi
```

```
spec:
  containers:
      - command:
          - apache_exporter
          - '-scrape_uri'
          - 'http://127.0.0.1:8080/server-status/?auto'
        image: previousnext/apache-exporter
        imagePullPolicy: Always
        name: apache-exporter
        ports:
          - containerPort: 9117
            protocol: TCP
      - image: 172.30.220.103:5000/.......
        imagePullPolicy: Always
        name: ....
```

The modification of the *DeploymentConfig* will be followed by a new *build* of the image.

2.2 Service

In the common case, we don't have to modify the service, but in this case, we must access to the exporter service through the port *9117*.

```
$ oc get svc
$ oc edit svc/cdnapi
```

```
spec:
  ports:
    - name: 8080-tcp
      port: 8080
      protocol: TCP
      targetPort: 8080
    - name: 9117-tcp
      port: 9117
      protocol: TCP
      targetPort: 9117
```

Finally, we check modification with

```
$ oc get svc
$ oc describe svc/cdnapi
$ oc get dc
$ oc describe dc/cdnapi
```

Et voila...
Enjoy