Side Car With Openshift

Draft

Laurent Valeyre

August 2018

Contents

1 The Project

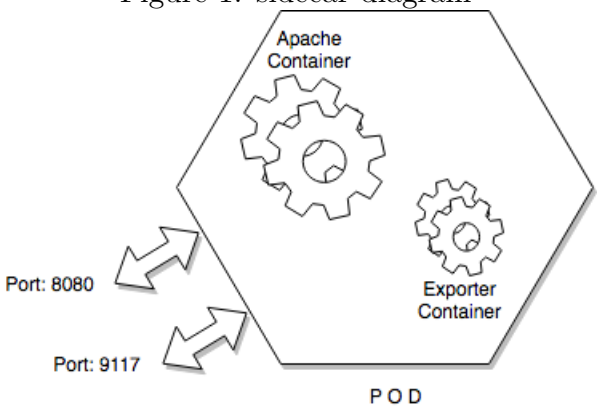
1.1 Introduction

The *Sidecar* concept provide 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.

```
apiVersion: v1
kind: Secret
metadata:
name: github-secret
namespace: sidecar
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
```

or if we use a *ssh key*,we generate the *key*, create the secret, and link the secret to the right project

```
  $ ssh-keygen -C "openshift-source-builder/repo@github" \
  -f repo-at-github -N ''
  $ oc secrets new-sshauth repo-at-github \
  --ssh-privatekey=repo-at-github
$ oc secrets link builder repo-at-github
```

When we create a new application based on this repository, it desn't work. We have to set the new *build*

```
$ oc set build-secret --source bc/mysite repo-at-github
```

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

1.5 New Project

Firstly, we create a new project

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

1.6 New Build

To obtain our image, we firstly

```
$ oc new-build http://192.168.0.8:8880/spike/faye.git \
--name faye
```

But to resolve the issue based on the credential, we'll attribute the *login/password* defined before and retstart the build process.

```
$ oc set build-secret --source bc/faye github-secret
$ oc start-build faye
```

or directly

```
$  oc new-build http://192.168.0.8:8880/spike/faye.git \
--source-secret github-secret
--name faye
```

1.7 New Application

It's time to create our application

```
$ oc new-app faye \
--name fayeapp
$ oc status
$ oc expose service faye
$ oc get pod
$ oc get all name --selector app=cdnselect
```

2 The Side Car

2.1 Export

We firstly export our *project*.

```
$ oc get --export is,bc,dc,svc -o yaml > export.yaml
```

2.2 ImageStream

We delete *resourceVersion, selfLink and uid*. In status, we keep dockerImageRepository (set to "")

2.3 BuildConfig

We delete *resourceVersion, selfLink and uid*. We delete in spec.triggers.imageChange *lastTriggeredImageID*

2.4 DeploymentConfig

We replace spec.template.spec.containers.image by *faye* in the first container
We add in spec.template.spec.container

```
- name: apache-exporter
image: previousnext/apache-exporter
```

```
command: [ "apache_exporter", \
"-scrape_uri", \
"http://127.0.0.1:8080/server-status/?auto" ]
ports:
- containerPort: 9117
```

2.5 Service

We delete *resourceVersion, selfLink and uid.*

We add in spec.ports

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

3 Geek Method

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

3.1 DeploymentConfig

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

```
$ oc get dc
$ oc edit dc/faye
```

```
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: faye
        ports:
          - containerPort: 8080
            protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
```

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

3.2 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 getsvc
$ oc edit svc/faye
```

```
spec:
  ports:
    - name: 8080-tcp
```

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