Multi-Container Pattern (Side car, Ambassador, Adapter)

[Edition 1]

[Last Update 200831]

For any issues/help contact : ***support@k21academy.com***

Contents

1 INTRODUCTION

There are three common design patterns and use-cases for combining multiple containers into a single pod. We'll walk through the **sidecar pattern**, the **adapter pattern**, and the **ambassador pattern**. Look to the end of the post for example YAML files for each of these.

The sidecar pattern consists of a main application—i.e. your web application—plus a helper container with a responsibility that is essential to your application, but is not necessarily part of the application itself.

The ambassador pattern is a useful way to connect containers with the outside world. An ambassador container is essentially a proxy that allows other containers to connect to a port on localhost while the ambassador container can proxy these connections to different environments depending on the cluster's needs.

This guide Covers:

1 Multi-Container Pattern – Sidecar Container

- Creating Multi-Container Pod with Shared Volume

2 Multi-Container Pattern – Ambassador Container

- Creating ConfigMap and Multi-Container Pod

3 Multi-Container Pattern – Adapter Container

- Creating ConfigMap and Multi-Container Pod

2 DOCUMENTATION

2.1 Kubernetes Documentation

1 Communicate Between Containers

https://kubernetes.io/docs/tasks/access-application-cluster/communicate-containers-same-pod-shared-volume/

2 Patterns for Composite Containers
https://kubernetes.io/blog/2015/06/the-distributed-system-toolkit-patterns/

3 Pods
https://kubernetes.io/docs/concepts/workloads/pods/

2.2 Linux Commands and VIM Commands

1. Basic Linux Commands
   https://maker.pro/linux/tutorial/basic-linux-commands-for-beginners
   https://www.hostinger.in/tutorials/linux-commands
2. Basic VIM Commands
   https://coderwall.com/p/adv71w/basic-vim-commands-for-getting-started
3. Popular VIM Commands
   https://www.keycdn.com/blog/vim-commands

3 MULTI-CONTAINER PATTERN – SIDECAR CONTAINER

3.1 Creating Multi-Container Pod with Shared Volume

1. Create a multi container pod with sidecar container pattern
2. Viewing the contents of multi-container.yaml file which has two container definition

```
$ vim multi-container.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: mc1
spec:
  volumes:
  - name: html
    emptyDir: {}
  containers:
  - name: 1st
    image: nginx
    volumeMounts:
    - name: html
      mountPath: /usr/share/nginx/html
  - name: 2nd
    image: debian
    volumeMounts:
    - name: html
      mountPath: /html
    command: ["/bin/sh", "-c"]
    args:
      - while true; do
          date >> /html/index.html;
          sleep 1;
        done
~
~
~
```

3. Deploying both the containers from multi-container.yaml file

```
$ kubectl create -f multi-container.yaml
```

```
root@kubeadm-master-01:~/Kubernetes#
root@kubeadm-master-01:~/Kubernetes#
root@kubeadm-master-01:~/Kubernetes# kubectl create -f multi-container.yaml
pod/mc1 created
root@kubeadm-master-01:~/Kubernetes#
```

4. Verify the pod status

```
$ kubectl get pods - w
$ Kubectl describe pod mc1
```

```
root@kubeadm-master-01:~/Kubernetes# kubectl get pods -w
NAME    READY   STATUS    RESTARTS   AGE
mc1     2/2     Running   0          23s
^Z
[1]+  Stopped                 kubectl get pods -w
```

```
root@kubeadm-master:/home/ubuntu/Kubernetes# kubectl describe pod mc1
Name:         mc1
Namespace:    default
Priority:     0
Node:         worker1/10.0.0.5
Start Time:   Fri, 02 Oct 2020 17:16:14 +0000
Labels:       <none>
Annotations:  <none>
Status:       Running
IP:           10.36.0.3
IPs:
  IP:  10.36.0.3
Containers:
  1st:
    Container ID:   docker://92e66344e0ba3f325870720a1bbb9772187d7e91e716526a6caabcb254487582
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:c628b67d21744fce822d22fdcc0389f6bd763daac23a6b77147d0712ea7102d0
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Fri, 02 Oct 2020 17:16:16 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /usr/share/nginx/html from html (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-fq86n (ro)
  2nd:
    Container ID:  docker://f8304750c1e1ce96cda17094ae430009298df4b5848cc76cd4c0d687da8a3af2
    Image:         debian
    Image ID:      docker-pullable://debian@sha256:439a6bae1ef351ba9308fc9a5e69ff7754c14516f6be8ca26975fb564cb7fb76
    Port:          <none>
    Host Port:     <none>
    Command:
      /bin/sh
      -c
    Args:
      while true; do date >> /html/index.html; sleep 1; done
    State:          Running
      Started:      Fri, 02 Oct 2020 17:16:26 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /html from html (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-fq86n (ro)
Conditions:
```

5. Checking the shared directory directly in the containers

```
$ kubectl exec mc1 -c 1st -- /bin/cat /usr/share/nginx/html/index.html
```

```
root@kubeadm-master-01:~/Kubernetes#
root@kubeadm-master-01:~/Kubernetes# kubectl exec mc1 -c 1st -- /bin/cat /usr/share/nginx/html/index.html
Tue Jun 16 17:12:08 UTC 2020
Tue Jun 16 17:12:09 UTC 2020
Tue Jun 16 17:12:10 UTC 2020
Tue Jun 16 17:12:11 UTC 2020
Tue Jun 16 17:12:12 UTC 2020
Tue Jun 16 17:12:13 UTC 2020
Tue Jun 16 17:12:14 UTC 2020
Tue Jun 16 17:12:15 UTC 2020
Tue Jun 16 17:12:16 UTC 2020
Tue Jun 16 17:12:17 UTC 2020
Tue Jun 16 17:12:18 UTC 2020
Tue Jun 16 17:12:19 UTC 2020
Tue Jun 16 17:12:20 UTC 2020
Tue Jun 16 17:12:21 UTC 2020
Tue Jun 16 17:12:22 UTC 2020
```

6. We can see that the sidecar container populates the index.html page for the main container serving as web server

3.2 Clean-up resources created in this lab exercise

```
$ kubectl delete -f  multi-container.yaml
```

4 MULTI-CONTAINER PATTERN – AMBASSADOR CONTAINER

4.1 Creating ConfigMap and Multi-Container Pod

1. Create a ConfigMap with the nginx configuration file

2. Viewing the contents of multi-pod-configmap.yaml file which has config map containing the nginx.conf file

```
$ vim multi-pod-configmap.yaml
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mc3-nginx-conf
data:
  nginx.conf: |-
    user  nginx;
    worker_processes  1;

    error_log  /var/log/nginx/error.log warn;
    pid        /var/run/nginx.pid;

    events {
        worker_connections  1024;
    }

    http {
        include       /etc/nginx/mime.types;
        default_type  application/octet-stream;

        sendfile        on;
        keepalive_timeout  65;

        upstream webapp {
            server 127.0.0.1:5000;
        }

        server {
            listen 80;

            location / {
                proxy_pass         http://webapp;
                proxy_redirect     off;
            }
        }
    }
~
```

3. Creating multi-container pod using Ambassador pattern

4. Viewing the contents of multi-pod-nginx.yaml file which has two container definition in a single pod

```
$ vim multi-pod-nginx.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: mc3
  labels:
    app: mc3
spec:
  containers:
  - name: webapp
    image: training/webapp
  - name: nginx
    image: nginx:alpine
    ports:
    - containerPort: 80
    volumeMounts:
    - name: nginx-proxy-config
      mountPath: /etc/nginx/nginx.conf
      subPath: nginx.conf
  volumes:
  - name: nginx-proxy-config
    configMap:
      name: mc3-nginx-conf
~
~
~
~
~
~
```

5. Deploying the resources from both multi-pod-configmap.yaml and multi-pod-nginx.yaml file

```
$ kubectl create -f multi-pod-configmap.yaml
$ kubectl create -f multi-pod-nginx.yaml
```

```
root@kubeadm-master-01:~/Kubernetes# kubectl create -f multi-pod-configmap.yaml
configmap/mc3-nginx-conf created
root@kubeadm-master-01:~/Kubernetes# kubectl create -f multi-pod-nginx.yaml
pod/mc3 created
```

6. Verify the pod status and ConfigMap resource creation

```
$ kubectl get pods
$ Kubectl describe pod mc3
```

```
root@kubeadm-master-01:~/Kubernetes# kubectl get pods mc3
NAME   READY   STATUS    RESTARTS   AGE
mc3    2/2     Running   0          14m
root@kubeadm-master-01:~/Kubernetes#
```

```
$ kubectl get cm
```

```
root@kubeadm-master-01:~/Kubernetes# kubectl get cm
NAME              DATA   AGE
mc3-nginx-conf    1      26m
root@kubeadm-master-01:~/Kubernetes#
```

7. Connect to the Ambassador container at Pod IP Address and Port 80. In below curl command provide your mc3 pod ip address

```
$ kubectl get pods -o wide
$ curl <Pod IP Address>
```

```
root@kubeadm-master:/home/ubuntu/Kubernetes# kubectl get pods -o wide
NAME          READY   STATUS        RESTARTS   AGE    IP          NODE      NOMINATED NODE   READINESS GATES
mc3           2/2     Running       0          2m     10.36.0.3   worker1   <none>           <none>
private-reg   1/1     Terminating   0          2d3h   10.32.0.6   worker2   <none>           <none>
root@kubeadm-master:/home/ubuntu/Kubernetes# curl 10.36.0.3
Hello world!root@kubeadm-master:/home/ubuntu/Kubernetes#
```

4.2 Clean-up resources created in this lab exercise

```
$ kubectl delete -f  multi-pod-configmap.yaml
$ kubectl delete -f  multi-pod-nginx.yaml
```

5 MULTI-CONTAINER PATTERN – ADAPTER CONTAINER

5.1 Creating ConfigMap and Multi-Container Pod

1. Create a ConfigMap with the nginx configuration file

2. Viewing the contents of multi-pod-configmap.yaml file which has config map containing the nginx.conf file

```
$ vim adapter-configmap.yaml
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-conf
data:
  default.conf: |
    server {
      listen       80;
      server_name  localhost;
      location / {
          root   /usr/share/nginx/html;
          index  index.html index.htm;
      }
      error_page   500 502 503 504  /50x.html;
      location = /50x.html {
          root   /usr/share/nginx/html;
      }
      location /nginx_status {
        stub_status;
        allow 127.0.0.1;  #only allow requests from localhost
        deny all;   #deny all other hosts
      }
    }
~
~
```

3. Creating multi-container pod using Adapter pattern

4. Viewing the contents of adapter-pod.yaml file which has two container definition in a single pod.

5. The Pod definition contains two containers; the nginx container, which acts as the application container, and the other is the adapter container. The adapter container uses the nginx/nginx-prometheus-exporter image which does the transformation of the metrics that Nginx exposes on /nginx_status to the Prometheus expected format

```
$ vim adapter-pod.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: webserver
spec:
  volumes:
  - name: nginx-conf
    configMap:
      name: nginx-conf
      items:
      - key: default.conf
        path: default.conf
  containers:
  - name: webserver
    image: nginx
    ports:
    - containerPort: 80
    volumeMounts:
    - mountPath: /etc/nginx/conf.d
      name: nginx-conf
      readOnly: true
  - name: adapter
    image: nginx/nginx-prometheus-exporter:0.4.2
    args: ["-nginx.scrape-uri","http://localhost/nginx_status"]
    ports:
    - containerPort: 9113
~
~
~
~
```

6. Deploying the resources from both apadpter-configmap.yaml and apdapter-pod.yaml file

```
$ kubectl create -f adapter-configmap.yaml
$ kubectl create -f adapter-pod.yaml
```

```
root@kubeadm-master:/home/ubuntu/Kubernetes-basics/newlabs# kubectl create -f adapter-configmap.yaml
configmap/nginx-conf created
root@kubeadm-master:/home/ubuntu/Kubernetes-basics/newlabs# kubectl create -f adapter-pod.yaml
pod/webserver created
```

7. Verify the pod status and ConfigMap resource creation

```
$ kubectl get pods
$ kubectl get configmap
```

```
root@kubeadm-master:/home/ubuntu/Kubernetes-basics/newlabs# kubectl get configmap
NAME                   DATA   AGE
example-redis-config   1      30d
my-config              1      30d
my-release-mariadb     1      21d
nginx-conf             1      13s
root@kubeadm-master:/home/ubuntu/Kubernetes-basics/newlabs# kubectl get pod
NAME          READY   STATUS    RESTARTS   AGE
private-reg   1/1     Running   0          12h
webserver     2/2     Running   0          10s
```

8. Get into the nginx container and install curl to be able to establish HTTP requests, and examine the /nginx_status

```
$ kubectl exec -it webserver bash
$ apt update && apt install curl -y
```

```
root@kubeadm-master:/home/ubuntu/Kubernetes-basics/newlabs# kubectl exec -it webserver bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl kubectl exec [POD] -- [COMMAND] instead.
Defaulting container name to webserver.
Use 'kubectl describe pod/webserver -n default' to see all of the containers in this pod.
```

```
root@webserver:/# apt update && apt install curl -y
Get:1 http://deb.debian.org/debian buster InRelease [121 kB]
Get:2 http://deb.debian.org/debian buster-updates InRelease [51.9 kB]
Get:3 http://security.debian.org/debian-security buster/updates InRelease [65.4 kB]
Get:4 http://deb.debian.org/debian buster/main amd64 Packages [7906 kB]
Get:5 http://security.debian.org/debian-security buster/updates/main amd64 Packages [234 kB]
Get:6 http://deb.debian.org/debian buster-updates/main amd64 Packages [7868 B]
Fetched 8387 kB in 2s (3525 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
3 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree
Reading state information... Done
curl is already the newest version (7.64.0-4+deb10u1).
0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
```

9. Curl and verify the nginx container status, serving at /nginx_status

```
$ curl localhost/nginx_status
```

```
root@webserver:/# curl localhost/nginx_status
Active connections: 2
server accepts handled requests
 2 2 2
Reading: 0 Writing: 1 Waiting: 1
```

10. Check the metrics reported by the nginx container

```
$ curl localhost:9313/metrics
```

```
root@webserver:/# curl localhost:9313/metrics
curl: (7) Failed to connect to localhost port 9313: Connection refused
```

11. Check the metrics reported by the adapter container. Prometheus is used for monitoring, and the nginx exporter exports monitoring metrics, such as number of request, request time and returned code

```
$ curl localhost:9113/metrics
$ exit
```

```
root@webserver:/# curl localhost:9113/metrics
# HELP nginx_connections_accepted Accepted client connections
# TYPE nginx_connections_accepted counter
nginx_connections_accepted 3
# HELP nginx_connections_active Active client connections
# TYPE nginx_connections_active gauge
nginx_connections_active 1
# HELP nginx_connections_handled Handled client connections
# TYPE nginx_connections_handled counter
nginx_connections_handled 3
# HELP nginx_connections_reading Connections where NGINX is reading the request header
# TYPE nginx_connections_reading gauge
nginx_connections_reading 0
# HELP nginx_connections_waiting Idle client connections
# TYPE nginx_connections_waiting gauge
nginx_connections_waiting 0
# HELP nginx_connections_writing Connections where NGINX is writing the response back to the client
# TYPE nginx_connections_writing gauge
nginx_connections_writing 1
# HELP nginx_http_requests_total Total http requests
# TYPE nginx_http_requests_total counter
nginx_http_requests_total 3
# HELP nginx_up Status of the last metric scrape
# TYPE nginx_up gauge
nginx_up 1
# HELP nginxexporter_build_info Exporter build information
# TYPE nginxexporter_build_info gauge
nginxexporter_build_info{gitCommit="f017367",version="0.4.2"} 1
root@webserver:/# curl localhost/nginx_status
Active connections: 2
server accepts handled requests
 4 4 4
Reading: 0 Writing: 1 Waiting: 1
root@webserver:/#
```

5.2 Clean-up resources created in this lab exercise

```
$ kubectl delete -f adapter-configmap.yaml
$ kubectl delete -f adapter-pod.yaml
```

6 SUMMARY

In this guide we Covered:

1. Multi-Container Pattern – Sidecar Container
    - Creating Multi-Container Pod with Shared Volume
2. Multi-Container Pattern – Ambassador Container
    - Creating ConfigMap and Multi-Container Pod
3. Multi-Container Pattern – Adapter Container
    - Creating ConfigMap and Multi-Container Pod