Journal of Computer Engineering
ISSN 10003428

Original Research Article
Vol 8, Issue 11, 2019 | ISSN: 1000-3428

Investigating the Deployment and Scalability of Sidecar Containers in Kubernetes Environments

Rohan K. Patel and Dhruv A. Shah

Department of Computer Science, University of California, Los Angeles (UCLA), Los Angeles, CA, USA; and Department of Information Technology, Indian Institute of Technology (IIT) Bombay, Mumbai, India

Abstract This paper presents an overview of Kubernetes, an open-source container orchestration engine facilitating automated deployment, scaling, and management of containerized applications. Central to Kubernetes' architecture are pods, serving as the foundational unit for application deployment and management. Pods encapsulate containers alongside associated resources such as storage, IP addresses, and runtime configurations. While single-container pods are prevalent, multi-container pods, such as those employing the sidecar container pattern, offer enhanced functionality and flexibility. This paper delves into the sidecar container pattern, detailing its implementation through an illustrative project, thereby providing readers with comprehensive insights into Kubernetes' architecture and practical application within containerized environments.

1. Introduction

KUBERNETES is an open-source container orchestration engine for automating deployment, scaling, and management of containerized applications. A pod is the basic building block of Kubernetes application. Kubernetes manages pods instead of containers and pods encapsulate containers. A pod may contain one or more containers, storage, IP addresses, and options that govern how containers should run inside the pod. A pod that contains one container refers to a single container pod and it is the most common Kubernetes use case. A pod that contains multiple co-related containers refers to a multi-container pod. There are a few patterns for multi-container pods one of them is the sidecar container pattern. In this post, we will see this pattern in detail with an example project.

- What is Sidecar Container
- Other Patterns
- Example Project
- Test With Deployment Object • How to Configure Resource Limits
- When should we use this pattern?
- Summary
- Conclusion

2. What Is Sidecar Container

Sidecar containers are the containers that should run along with the main container in the pod. This sidecar pattern extends and enhances the functionality of current containers without changing it. Nowadays, We know that we use container technology to wrap all the dependencies for the application to run anywhere. A container does only one thing and does that thing very well.

Imagine that you have a pod with a single container working very well and you want to add some functionality to the current container without touching or changing, how can you add

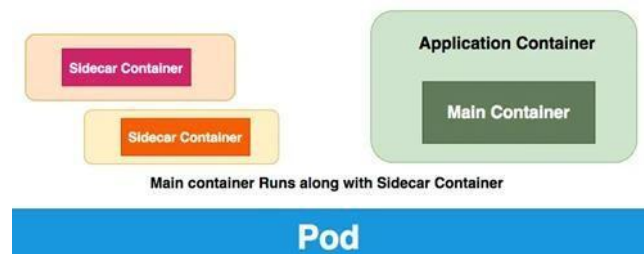
© 2019 The Authors. Open Access under CC BY 4.0.

How to cite: Rohan K. Patel and Dhruv A. Shah (2019). Investigating the Deployment and Scalability of Sidecar Containers in Kubernetes Environments. Journal of Computer Engineering, 8(11), 14–16. DOI: https://doi.org/10.5281/zenodo.19341051

the additional functionality or extending the current functionality? This sidecar container pattern really helps exactly in that situation.current container

Journal of Scientific and Engineering Research

without touching or changing, how can you add the additional functionality or extending the current functionality? This sidecar container pattern really helps exactly in that situation.



If you look at the above diagram, you can define any number of containers for Sidecar containers and your main container works along with it successfully. All the Containers will be executed parallelly and the whole functionality works only if both types of containers are running successfully. Most of the time these sidecar containers are simple and small that consume fewer resources than the main container.

3. Other Patterns

There are other patterns that are useful for everyday Kubernetes workloads.

- Init Container Pattern
- Adapter Container Pattern
- Ambassador Container Pattern

4. Example Project

Here is an example project you can clone and run it on your machine. You need to install Minikube as a prerequisite. https://github.com/bbachi/k8s-sidecar-container-pattern.git

Let's implement a simple project to understand this pattern. Here is a simple pod that has main and sidecar containers. The main container is nginx serving on port **80** that takes the index.html from the volume mount **workdir** location. The Sidecar container with the image busybox creates logs in the same location with a timestamp. Since the Sidecar container and main container run parallel Nginx will display the new log information every time you hit in the browser.

https://gist.github.com/bbachi/7e1a5896e60606d18a5303813a97d577#file-pod-yml

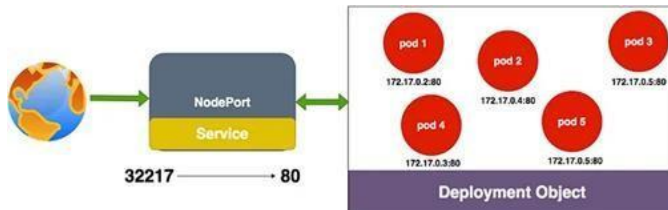
// create the pod kubectl create -f pod.yml// list the pods kubectl get po// exec into pod kubectl exec -it sidecar-
container-demo -c main-container -- /bin/sh# apt-get update && apt-get install -y curl # curl localhost You can install curl and query the local host and check the response.

```
Bhargavs-MacBook-Pro:k8s-sidecar-container-pattern bhargavbachina$ kubectl create -f pod.yml
pod/sidecar-container-demo created
Bhargavs-MacBook-Pro:k8s-sidecar-container-pattern bhargavbachina$ kubectl get po
NAME                     READY   STATUS    RESTARTS   AGE
sidecar-container-demo   2/2     Running   0          11s
Bhargavs-MacBook-Pro:k8s-sidecar-container-pattern bhargavbachina$ kubectl exec -it sidecar-container-demo -c main-container -- /bin/sh
# apt-get update && apt-get install -y curl
Get:1 http://deb.debian.org/debian buster InRelease [122 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://deb.debian.org/debian buster-updates/main amd64 Packages [7868 B]
Get:6 http://security.debian.org/debian-security buster/updates/main amd64 Packages [226 kB]
Fetched 8379 kB in 3s (3273 kB/s)
Reading package lists... Done
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 0 not upgraded.
# curl localhost
echo Sat Sep 5 22:28:18 UTC 2020 Hi I am from Sidecar container
echo Sat Sep 5 22:28:23 UTC 2020 Hi I am from Sidecar container
echo Sat Sep 5 22:28:28 UTC 2020 Hi I am from Sidecar container
echo Sat Sep 5 22:28:33 UTC 2020 Hi I am from Sidecar container
echo Sat Sep 5 22:28:38 UTC 2020 Hi I am from Sidecar container
echo Sat Sep 5 22:28:43 UTC 2020 Hi I am from Sidecar container
echo Sat Sep 5 22:28:48 UTC 2020 Hi I am from Sidecar container
echo Sat Sep 5 22:28:53 UTC 2020 Hi I am from Sidecar container
echo Sat Sep 5 22:28:58 UTC 2020 Hi I am from Sidecar container
echo Sat Sep 5 22:29:03 UTC 2020 Hi I am from Sidecar container
# curl localhost
echo Sat Sep 5 22:28:18 UTC 2020 Hi I am from Sidecar container
echo Sat Sep 5 22:28:23 UTC 2020 Hi I am from Sidecar container
echo Sat Sep 5 22:28:28 UTC 2020 Hi I am from Sidecar container
echo Sat Sep 5 22:28:33 UTC 2020 Hi I am from Sidecar container
echo Sat Sep 5 22:28:38 UTC 2020 Hi I am from Sidecar container
echo Sat Sep 5 22:28:43 UTC 2020 Hi I am from Sidecar container
echo Sat Sep 5 22:28:48 UTC 2020 Hi I am from Sidecar container
echo Sat Sep 5 22:28:53 UTC 2020 Hi I am from Sidecar container
echo Sat Sep 5 22:28:58 UTC 2020 Hi I am from Sidecar container
echo Sat Sep 5 22:29:03 UTC 2020 Hi I am from Sidecar container
echo Sat Sep 5 22:29:08 UTC 2020 Hi I am from Sidecar container
echo Sat Sep 5 22:29:13 UTC 2020 Hi I am from Sidecar container
echo Sat Sep 5 22:29:18 UTC 2020 Hi I am from Sidecar container
#
```

Figure 1: Testing Sidecar Container

5. Test with Deployment Object

Let's create a deployment object with the same pod specification with 5 replicas. I have created a service with the port type NodePort so that we can access the deployment from the browser. Pods are dynamic here and the deployment controller always tries to maintain the desired state. That's why you can't have one static IP Address to access the pods so that you have to create a service that exposes the static port to the outside world. Internally service maps to port 80 based on the selectors. You will see that in action in a while.



Let's look at the below deployment object where we define one main container and two sidecar containers. All the containers run in parallel. The two sidecar containers create logs in the location **/var/log.** The main container Nginx serves those log files as when we hit the NGINX web server from port 80. You will see that in action in a while. https://gist.github.com/bbachi/7d6c40fc8f660eed243f7e9cd31d99c8#file-manifest-yml

Let's follow these commands to test the deployment.

// create a deployment kubectl create -f manifest.yml// list the deployment, pods, and service kubectl get deploy -o wide kubectl get po -o widekubectl get svc -o wide

```
Bhargavs-MacBook-Pro:k8s-sidecar-container-pattern bhargavbachina$ kubectl create -f manifest.yml
deployment.apps/nginx-webapp created
service/nginx-webapp created
Bhargavs-MacBook-Pro:k8s-sidecar-container-pattern bhargavbachina$ kubectl get deploy -o wide
NAME           READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS                                            IMAGES                  SELECTOR
nginx-webapp   0/5     5            0           14s   sidecar-container1,sidecar-container2,main-container   busybox,busybox,nginx   app=nginx-webapp
Bhargavs-MacBook-Pro:k8s-sidecar-container-pattern bhargavbachina$ kubectl get pod -o wide
NAME                            READY   STATUS    RESTARTS   AGE   IP            NODE       NOMINATED NODE   READINESS GATES
nginx-webapp-7c8b4d4f8d-9qmdm   3/3     Running   0          28s   172.17.0.3    minikube   <none>           <none>
nginx-webapp-7c8b4d4f8d-b9rnv   3/3     Running   0          28s   172.17.0.5    minikube   <none>           <none>
nginx-webapp-7c8b4d4f8d-d446v   3/3     Running   0          28s   172.17.0.13   minikube   <none>           <none>
nginx-webapp-7c8b4d4f8d-kgnvx   3/3     Running   0          28s   172.17.0.12   minikube   <none>           <none>
nginx-webapp-7c8b4d4f8d-x7hnm   3/3     Running   0          28s   172.17.0.4    minikube   <none>           <none>
Bhargavs-MacBook-Pro:k8s-sidecar-container-pattern bhargavbachina$ kubectl get svc -o wide
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE   SELECTOR
kubernetes     ClusterIP   10.96.0.1       <none>        443/TCP        78d   <none>
nginx-webapp   NodePort    10.101.115.82   <none>        80:32123/TCP   35s   app=nginx-webapp
Bhargavs-MacBook-Pro:k8s-sidecar-container-pattern bhargavbachina$ kubectl cluster-info
Kubernetes master is running at https://192.168.64.2:8443

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

Figure 3: Deployment in action

In the above diagram, you can see 5 pods running in different IP addresses and the service object maps the port **32123** to port **80**. You can access this deployment from the browser from the Kubernetes master IP address **192.168.64.2** and the service port **32123**. http://192.168.64.2:32123

You can even test the pod with the following commands.

// exec into main container of the pod kubectl exec -it nginx-webapp-7c8b4d4f8d-9qmdm -c main-container -- /bin/sh// install curl # apt-get update && apt-get install -y curl # curl localhost

Figure 4: Sidecar Pattern in action

6. How to Configure Resource Limits

Configuring resource limits is very important when it comes to Sidecar containers. The main point we need to understand here is all the containers run in parallel so when you configure resource limits for the pod you have to take that into consideration.

- The sum of all the resource limits of the main containers as well as sidecar containers (Since all the containers run in parallel)

7. When Should We Use This Pattern

These are some of the scenarios where you can use this pattern.

- Whenever you want to extend the functionality of the existing single container pod without touching the existing one.
- Whenever you want to enhance the functionality of the existing single container pod without touching the existing one.
- You can use this pattern to synchronize the main container code with the git server pull.
- You can use this pattern for sending log events to the external server.
- You can use this pattern for network-related tasks.

8. Summary

- A pod that contains one container refers to a single container pod and it is the most common Kubernetes use case.
- A pod that contains Multiple co-related containers refers to a multi-container pod.
- The Sidecar container pattern is one of the patterns that we use regularly for extending or enhancing pre-existing containers.
- Sidecar containers run in parallel with the main container. So that you need to consider resource limits of sidecar containers while defining request/resource limits for the pod.
- The application containers and Sidecar containers run in parallel which means all the containers run at the same time. So that you need to sum up all the request/resource limits of the containers while defining request/resource limits for the pod.
- You should configure health checks for sidecar containers as main containers to make sure they are healthy.
- All the pods in the deployment object don’t have static IP addresses so that you need a service object to expose yourself to the outside world.
- The service object internally maps to the port container port based on the selectors.
- You can use this pattern where your application or main containers need extending or enhancing the current functionality.

9. Conclusion

It is invaluable to comprehend established Kubernetes patterns, particularly regarding sidecar containers. When implementing sidecar containers, it is essential to ensure their simplicity and small footprint, as resource utilization is aggregated when defining resource limits for the pod. Furthermore, configuring health checks for sidecar containers is imperative to maintain overall pod health. Consequently, understanding the appropriate scenarios for consolidating functionality within the main container versus employing separate containers is crucial for effective application deployment and management in Kubernetes environments.

References

[1]. Official Docker Guides https://docs.docker.com/get-started/overview/
[2]. Official Kubernetes Docs https://kubernetes.io/docs/home/
[3]. Container Design Patterns https://kubernetes.io/blog/2016/06/container-design-patterns/