Kubernetes Containers: Sidecar and init containers, ephemeral debug environments

Check GitHub for helpful DevOps tools:

Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics

Ask Personal AI Document assistant to learn interactively (FASTER)!

1 Download PDF

2 Go to website

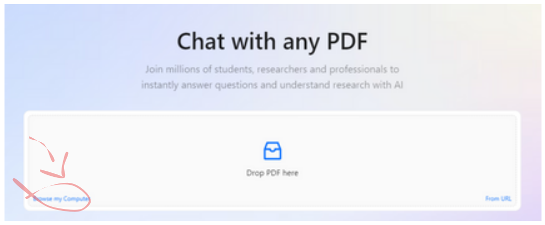
3 Browse file

4 Chat with Document

1 https://github.com/MichaelRobotics/DevOpsTools/blob/main/KubernetesContainers.pdf

Click there to go to ChatPdf website 2

3



Chat with any PDF

Join millions of students, researchers and professionals to instantly answer questions and understand research with AI

Ask questions about document! 4

Complety new to Linux and Networking?

Essential for this PDF is a thorough knowledge of networking. I highly recommend the HTB platform's networking module, which offers extensive information to help build a comprehensive understanding.

HTB - Your Cyber Performance Center

We provide a human-first platform creating and maintaining high performing cybersecurity individuals and organizations.

https://www.hackthebox.com/

What is Kubernetes?

Kubernetes is an open-source platform that automates the deployment, scaling, and management of containerized applications. It helps manage clusters of nodes running containers, ensuring efficient and reliable operation.

How Kubernetes clusters are made?

Kubernetes clusters consist of a control plane and multiple worker nodes. The control plane manages cluster operations, while worker nodes run the actual container workloads.

Why and When use Kubernetes

Kubernetes is ideal for deploying scalable, resilient, and automated containerized applications. It is used when managing multiple containers across different environments is necessary.

Example: Running a microservices-based e-commerce platform that scales up during peak hours.

System Requirements

- RAM: 2 GB per node (1 GB can work for testing but may lead to limited performance)
- 10 GB free storage
- Ubuntu

Kubernetes: Main components & packages

- **kube-apiserver:** Central management component that exposes the Kubernetes API; acts as the front-end for the cluster.
- **etcd:** Distributed key-value store for storing all cluster data, ensuring data consistency across nodes.
- **kube-scheduler:** Assigns pods to available nodes based on resource requirements and policies.
- **kube-controller-manager:** Manages core controllers that handle various functions like node status, replication, and endpoints.
- **kubelet:** Agent that runs on each node, responsible for managing pods and their containers.
- **kube-proxy:** Manages networking on each node, ensuring communication between pods and services within the cluster.

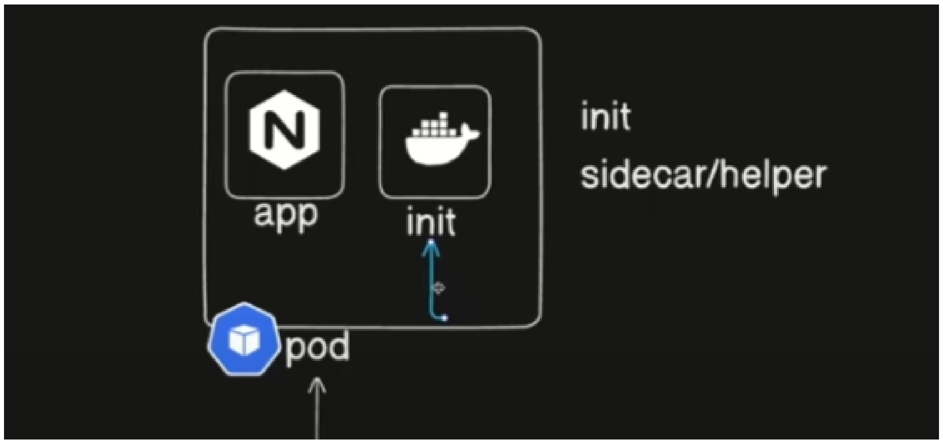
Kubernetes Containers: Intro

1) Why use additional containers in pods?

Sidecar containers solve the need for modularity by running auxiliary tasks (e.g., logging, proxying) alongside the main container. This decouples support functions from application code, promoting reusability and maintainability.

Init containers handle one-time initialization tasks like setup or dependency checks. By separating these from the main container, they simplify application images and ensure a consistent start-up environment.

Ephemeral debug environments allow temporary containers to be attached to pods for live troubleshooting without restarting or disrupting services, enabling safer and more efficient debugging.



Kubernetes Containers: Init container

Init containers run before the main containers in a Pod to perform setup tasks like initializing configurations, downloading dependencies, or preparing the environment. They execute sequentially and must complete successfully before the main containers start, ensuring the Pod is properly initialized. Once their tasks are done, they terminate and free up resources.

Lets test Init container functionality by creating pod:

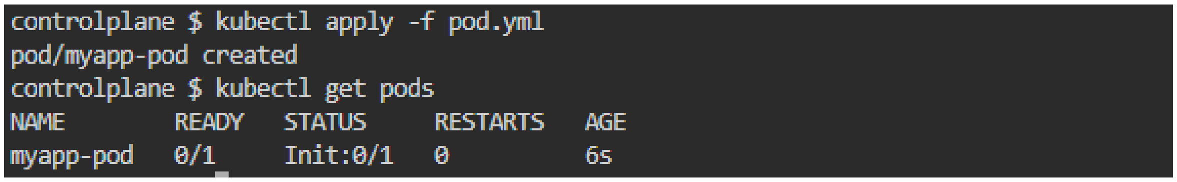
```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app.kubernetes.io/name: MyApp
spec:
  containers:
  - name: myapp-container
    image: busybox:1.28
    env:
    - name: FIRSTNAME
      value: "Piyush"
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
  initContainers:
  - name: init-myservice
    image: busybox:1.28
    command: ['sh', '-c']
    args: ['until nslookup myservice.default.svc.cluster.local; do echo waiting for myservice; \
sleep 2; done']
```

name manifest pod.yaml and apply:

```
kubectl apply -f pod.yaml
```

get pods

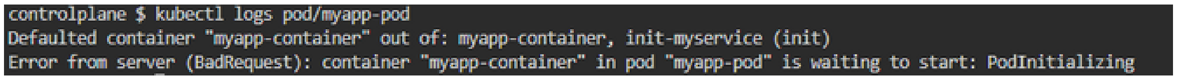
```
kubectl get pods
```



```
controlplane $ kubectl apply -f pod.yml
pod/myapp-pod created
controlplane $ kubectl get pods
NAME        READY   STATUS     RESTARTS   AGE
myapp-pod   0/1     Init:0/1   0          6s
```

Check why pod is not in running status:

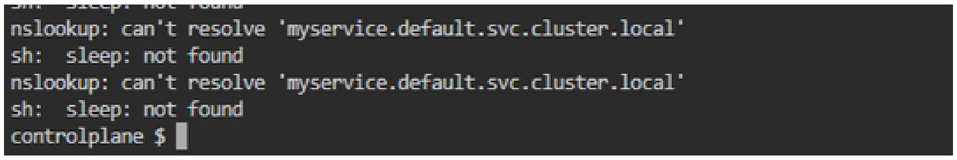
```
kubectl logs pod/myapp-pod
```



```
controlplane $ kubectl logs pod/myapp-pod
Defaulted container "myapp-container" out of: myapp-container, init-myservice (init)
Error from server (BadRequest): container "myapp-container" in pod "myapp-pod" is waiting to start: PodInitializing
```

Pod is in PodInitializing state. That means, something prevents pod container to start. Check logs of Init container

```
kubectl logs pod/myapp-pod -c init-myservice
```



```
nslookup: can't resolve 'myservice.default.svc.cluster.local'
sh:  sleep: not found
nslookup: can't resolve 'myservice.default.svc.cluster.local'
sh:  sleep: not found
controlplane $
```

Init container waits for myservice service. Nslookup cannot reach its DNS entry. As specified in InitContainer pod yaml section.

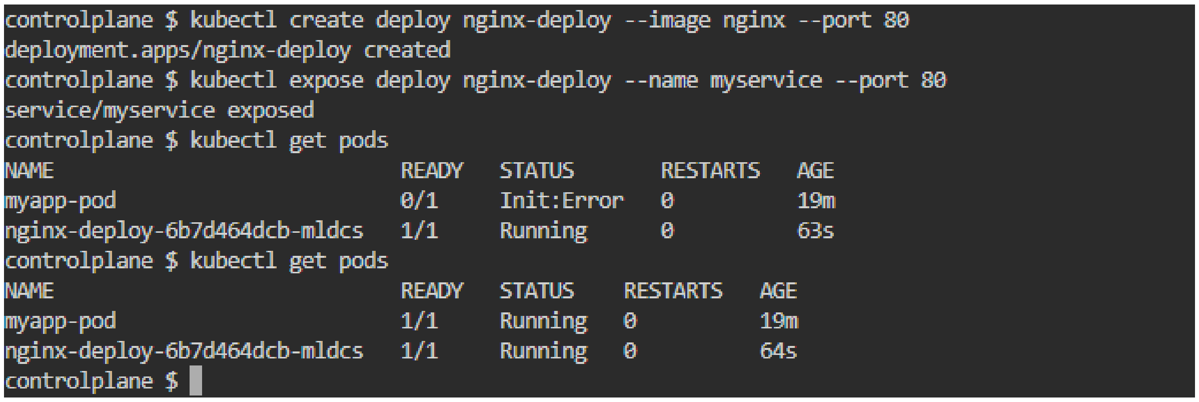
```
 initContainers:
 - name: init-myservice
 image: busybox:1.28
 command: ['sh', '-c']
 args: ['until nslookup myservice.default.svc.cluster.local; do echo waiting for myservice; \
sleep 2; done']
```

Create deployment and service pointing to pod inside this deployment:

```
kubectl create deploy nginx-deploy --image nginx --port 80

kubectl expose deploy nginx-deploy --name myservice --port 80
```

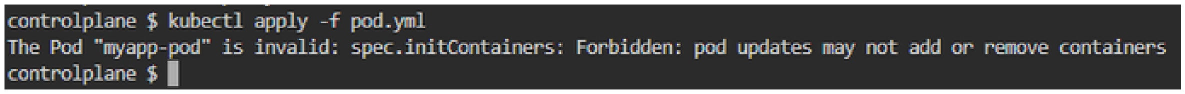
Now check pods. InitContainer requirement should be fulfilled and pod in running state:



```
controlplane $ kubectl create deploy nginx-deploy --image nginx --port 80
deployment.apps/nginx-deploy created
controlplane $ kubectl expose deploy nginx-deploy --name myservice --port 80
service/myservice exposed
controlplane $ kubectl get pods
NAME                           READY   STATUS       RESTARTS   AGE
myapp-pod                      0/1     Init:Error   0          19m
nginx-deploy-6b7d464dcb-mldcs   1/1     Running      0          63s
controlplane $ kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
myapp-pod                      1/1     Running   0          19m
nginx-deploy-6b7d464dcb-mldcs   1/1     Running   0          64s
controlplane $
```

Now, try to add new init container into pod yaml

```
- name: init-mydb
  image: busybox:1.28
  command: ['sh', '-c']
  args: ['until nslookup mydb.default.svc.cluster.local; do echo waiting for mydb; sleep 2; done']
```



```
controlplane $ kubectl apply -f pod.yml
The Pod "myapp-pod" is invalid: spec.initContainers: Forbidden: pod updates may not add or remove containers
controlplane $
```

Update failed. Containers inside a pod cannot be deleted or created.

Kubernetes Containers: Sidecar container

A common use case for a sidecar container is log aggregation. It collects logs from the main application and forwards them to a centralized system, such as Elasticsearch or Fluentd. This decouples logging from the application, improving scalability and flexibility.

```
apiVersion: v1
kind: Pod
metadata:
  name: sidecar-logging-pod
spec:
  containers:
    - name: main-app
      image: nginx:latest
      volumeMounts:
        - name: shared-logs
          mountPath: /var/log/app
    - name: log-forwarder
      image: fluentd:latest
      args: ["-c", "/fluentd/etc/fluent.conf"]
      volumeMounts:
        - name: shared-logs
          mountPath: /var/log/app
  volumes:
    - name: shared-logs
      emptyDir: {}
```

The containers section defines two containers: main-app, which runs the nginx web server and mounts the shared volume shared-logs at /var/log/app to store logs, and log-forwarder, a sidecar container running Fluentd to forward logs.

The log-forwarder uses the Fluentd configuration file specified in its arguments and mounts the same shared volume to access logs written by the main app. The volumes section defines the shared-logs volume as an emptyDir, allowing both containers to read and write to it while the pod is running,

Kubernetes Containers: Ephemeral Debug container

Ephemeral containers in Kubernetes are temporary containers added to existing Pods for debugging without altering the Pod's original specification. They allow troubleshooting by sharing the same resources and namespaces as the target container.

Debugging directly in production Pods is risky, as it can compromise stability and security. Production Pods should run only application-specific processes, with no shell access or debugging tools, enforced by solutions like Kubermor.

Instead of accessing production Pods, create a debug container that shares the target container's environment. Since containers can't be deleted individually, copy the Pod, debug it with an ephemeral container, and delete the debug Pod once the issue is resolved.

create simple pod

```
kubectl create pod my-simple-pod --image=nginx
```

Acces it with debug container

```
kubectl  debug  my-simple-pod --image alpine \
--stdin --tty --share-processes --copy-to silly-demo-debug
```

Thats all.

Debugging production pods risks changes, performance issues, and security vulnerabilities.

common troubleshooting

1) Init Container Failing

Cause: Errors or stuck init container.
Solution: Review init container logs (kubectl logs <pod-name> -c <init-name>), check dependencies, and Pod events

2) Ephemeral Debug Environment Not Starting

Cause: Configuration or resource issues.
Solution: Check debug container config, resources, and logs. Use kubectl describe pod <pod-name> for details.

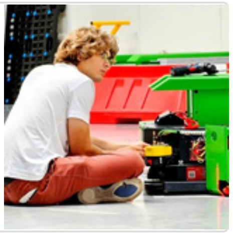
3) Sidecar Container Overwriting Logs

Cause: Shared log volumes.
Solution: Ensure separate log paths or configure log rotation for both containers.

4) Check my Kubernetes Troubleshooting series:

Michael Robotics
Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats skill information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics



Learn more about Kubernetes

Check Kubernetes and piyushsachdeva - great docs!

Setup a Multi Node Kubernetes Cluster

kubeadm is a tool to bootstrap the Kubernetes cluster

https://github.com/piyushsachdeva/CKA-2024/tree/main/Resources/Day27

Kubernetes Documentation

This section lists the different ways to set up and run Kubernetes

https://kubernetes.io/docs/setup/

Share, comment, DM and check GitHub for scripts & playbooks created to automate process.

Check my GitHub

Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats skill information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics

PS.

If you need a playbook or bash script to manage KVM on a specific Linux distribution, feel free to ask me in the comments or send a direct message!