

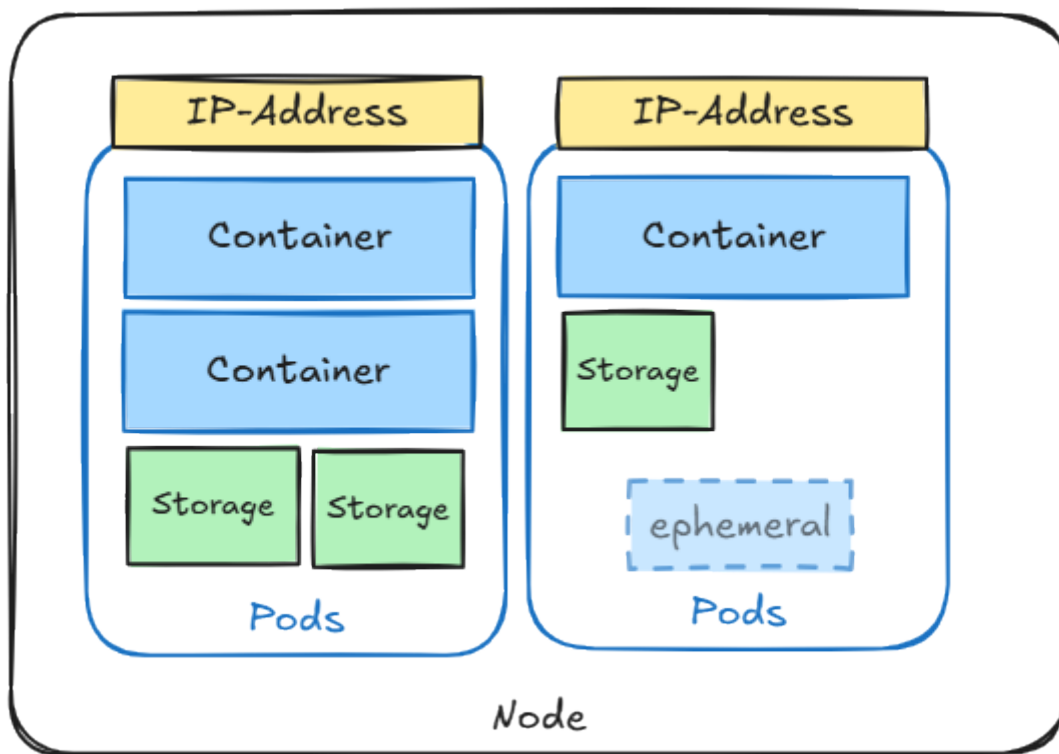
Pod



Useful Links:

- [Kubernetes Official Documentation](#)
- [Viewing Pods and Nodes](#)

Architecture:



Detailed Description:

A Pod is the smallest deployable unit in Kubernetes and serves as the basic building block for running applications in the cluster. Each Pod encapsulates one or more containers, which share the same resources such as storage, networking, and compute. Containers within a Pod are tightly coupled, meaning they always run together on the same node and

share the same network namespace, allowing them to communicate with each other using `localhost`.

Typically, a Pod has a single container, but it can host sidecar containers that assist the main application container with additional tasks like logging, monitoring, or proxying requests. Pods are ephemeral by nature, designed to be replaceable and scaled according to workload demands through higher-level Kubernetes abstractions like Deployments or StatefulSets.

Key characteristics of Pods include:

- **Shared Networking:** All containers in a Pod share the same IP address and port space.
- **Shared Storage:** Volumes attached to a Pod are shared among all its containers.
- **Lifecycle Management:** Pods are managed by controllers like Deployments, ReplicaSets, and DaemonSets to ensure desired state is maintained.

Init containers in Kubernetes run before the main app container starts in a pod.

- Prepare the environment (e.g., set up files or check conditions)
- Run once and finish before the main app starts
- Are useful for tasks that your main app doesn't handle well or should not have access to

Sidecar containers run alongside the app container in a pod to enhance its functionality without modifying the main app. They can share resources and help with tasks like logging, monitoring, or proxying.

Ephemeral containers are temporary containers that you can add to an existing Pod to troubleshoot or inspect it. Unlike regular containers, they are not part of the initial setup and cannot be restarted. They're useful when you need to debug or run commands in a Pod that's already running.

Command Reference Guide:

```
# Query running pods
```

```
kubectl get pods
```

```
# Query detailed information about pods
```

```
kubectl get pods -o wide
```

```
# Create single pod
```

```
kubectl run nginx --image=nginx
```

```
# Run image / pass environment and command
```

```
kubectl run --image=ubuntu ubuntu --env="KEY=VALUE" -- sleep infinity
```

```
# Get yaml configuration for the resource
```

```
kubectl run nginx --image=nginx --dry-run=client -o yaml | tee nginx.yaml
```

```
# Get specific information of any yaml section
```

```
kubectl explain pod.spec.restartPolicy
```

```
# Create pod resource from yaml configuration file
```

```
kubectl create -f nginx.yaml
```

```
# Apply pod resource from yaml configuration
```

```
kubectl apply -f nginx.yaml
```

```
# Delete pod resource without waiting for graceful shutdown of application (--now)
```

```
kubectl delete pod/nginx pod/ubuntu --now
```

```
# Get full resource description using describe
```

```
kubectl describe pod/nginx
```

```
# Get logs for a specific container in the pod
```

```
kubectl logs pod/nginx -c nginx
```

```
# If a pod fails use -p to get previous logs for a specific container in the pod
```

```
kubectl logs pod/nginx -c nginx -p
```

```
# Get shell from running container
```

```
kubectl exec --stdin --tty nginx -- /bin/bash
```

```
kubectl exec --stdin --tty nginx -c container1 -- /bin/bash # get access to specific container
```

```
# Combine pod creation
```

```
kubectl run nginx --image=nginx --dry-run=client -o yaml | tee nginx.yaml
```

```
kubectl run ubuntu --image=ubuntu --dry-run=client -o yaml | tee ubuntu.yaml
```

```
{ cat nginx.yaml; echo "---"; cat ubuntu.yaml; } | tee multi_pods.yaml
```

```
kubectl apply -f multi_pods.yaml
```

fail-pod-deploy.yaml:

```
apiVersion: v1
kind: Pod
metadata:
  name: blocked-pod
spec:
  restartPolicy: Never
  initContainers:
    - name: init-fail
      image: busybox
      command: ["sh", "-c", "exit 1"]
  containers:
    - name: app-container
      image: nginx
```

success-on-retry-pod-deploy.yaml:

```
apiVersion: v1
kind: Pod
metadata:
  name: blocked-pod
spec:
  restartPolicy: Always
  initContainers:
    - name: init-fail
      image: busybox
      command: ["sh", "-c", "if [ ! -f /data/ready ]; then touch /data/ready; sleep 10; exit 1; else exit 0; fi"]
  volumeMounts:
    - name: shared-data
      mountPath: /data
  containers:
    - name: app-container
      image: nginx
      volumeMounts:
        - name: shared-data
          mountPath: /data
```

```
volumes:  
- name: shared-data  
  emptyDir: {}
```

sidecar-pod-deploy.yaml:

```
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: myapp  
  labels:  
    app: myapp  
spec:  
  replicas: 1  
  selector:  
    matchLabels:  
      app: myapp  
  template:  
    metadata:  
      labels:  
        app: myapp  
    spec:  
      containers:  
        - name: myapp  
          image: alpine:latest  
          command: ['sh', '-c', 'while true; do echo "$(date) logging $((($RANDOM)))" >> /opt/logs.txt; sleep 5;  
done']  
          volumeMounts:  
            - name: data  
              mountPath: /opt  
        - name: logshipper  
          image: alpine:latest  
          command: ['sh', '-c', 'tail -F /opt/logs.txt']  
          volumeMounts:  
            - name: data  
              mountPath: /opt
```

volumes:

- name: data
- emptyDir: {}

Create init container that will fail. App container will not start

```
kubect! apply -f fail-pod-deploy.yaml && watch kubect! describe -f blocked.yaml
```

Create init container that will succeed on second try.

```
kubect! apply -f success-on-retry-pod-deploy.yaml && watch kubect! describe -f blocked.yaml
```

Run app container along with sidecar helper container

```
kubect! apply -f sidecar-pod-deploy.yaml && watch kubect! logs $(kubect! get pods -l app=myapp -o jsonpath='{.items[0].metadata.name}') --all-containers=true
```

Run ephemeral container (If you only need to inspect and debug the running Pod)

```
kubect! run ephemeral-demo --image=busybox --restart=Never -- sleep 100000
```

```
kubect! debug -it ephemeral-demo --image=busybox:1.28 --target=ephemeral-demo
```

```
kubect! describe pod ephemeral-demo
```

Copy and Add a New Container (If you need to change the environment or add more debugging tools)

```
kubect! run myapp --image=busybox:1.28 --restart=Never -- sleep 1d
```

```
kubect! debug myapp -it --image=ubuntu --share-processes --copy-to=myapp-debug
```

```
kubect! get pods
```

Revision #14

Created 22 November 2024 19:25:21 by Admin

Updated 30 November 2024 17:04:33 by Admin