Pod Deep Dive
The Interesting Bits

Cloud Native Summit Munich
July 21st 2025

🐘 @Marcus@k8s.social | 🌐 MarcusNoble.com | 🦋 @averagemarcus.bsky.social

Hi 👋,

I'm Marcus Noble!

I'm a platform engineer at Giant Swarm working on release engineering, CI/CD and general Kubernetes development.

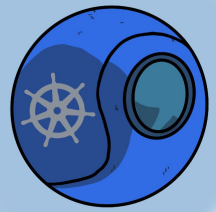
I run a monthly newsletter - CloudNative.Now

I'm a CNCF & Civo Ambassador.

7+ years experience running Kubernetes in production environments.

🐘 @Marcus@k8s.social | 🌐 MarcusNoble.com | 🦋 @averagemarcus.bsky.social


So what is a "Pod"?
Containers
Containers
Images
Containers
Runtime
Lifecycle & Health
Lifecycle & Health
Probes
Lifecycle & Health
Hooks
Lifecycle & Health
Conditions
Config
Config
Environment Variables
Config
Volumes
Scheduling
Scheduling
Resource Allocation
Scheduling
Node Assignment
Scheduling
Taints & Tolerations
Scheduling
Scheduler Logic
Networking



So what is a “Pod”?

Pods are the smallest deployable units of computing that you can create and manage in Kubernetes.

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

So what is a “Pod”?

- Smallest deployable unit of computing
- A wrapper around ***one or more*** containers (or WASM functions)
- Managed by the Kubernetes scheduler and assigned to nodes at runtime
- Workloads within a pod all share the same runtime context (Linux namespaces, cgroups, network, etc)
- Designed to be relatively ephemeral and disposable
- Mostly immutable (only changes to `image`, `activeDeadlineSeconds` and additions to `tolerations` are allowed)
    - v1.33 graduated in-place Pod resize to beta that allows changes to `resources` also

So what is a “Pod”?

But really, it’s much more than that! **Too much to cover in fact!**

So let’s talk about the interesting, weird or surprising bits!

Containers

Sidecar Containers

- Enabled by default from **v1.29** (feature gate `SidecarContainers`)
- “Disguised” as initContainers 🤷
- Launched when Pod scheduled, continues running until main application containers have fully stopped, then kubelet terminates all sidecars
- Supports `readinessProbe` (unlike normal initContainers) and used to determine the ready state of the Pod
- Termination handled more harshly than app containers - **`SIGTERM`** followed by **`SIGINT`** before graceful exit likely

```yaml
apiVersion: v1
kind: Pod
metadata:
    name: "tiny-pod"
spec:
    initContainers:
    - name: logshipper
      image: alpine
      restartPolicy: Always
```

Only allowed value

On the container, not the Pod

Ephemeral Containers

- Designed for debugging
- Must be added via a special `ephemeralcontainers` handler, not via the Pod spec (e.g. kubectl debug)
- Not supported on static pods
- Can target specific container process namespaces with optional `targetContainerName` property
- No support for `ports`, `probes`, `resources` or `lifecycle` on the container spec

```
kubectl debug tiny-pod -it \
    --image=alpine         \
    --target=nginx
```

```
apiVersion: v1
kind: Pod
metadata:
  name: "tiny-pod"
spec:
  ephemeralContainers:
  - name: debugger-67t9x
    image: alpine
    targetContainerName: nginx
```

Read only

Pause Container

- Every Pod includes an empty pause container which bootstraps the Pod with the cgroups, reservations and namespaces before the defined containers are created
- This container can be thought of as a “parent container” for all the containers within your Pod and will remain even if workload containers crash, ensuring namespaces and networking remain available
- The pause container is always present but *not visible* via the Kubernetes API
- Can be seen if you query directly on the node, e.g. with containerd:

```
~ # ctr -n k8s.io containers list | grep pause

03bfc9fa4bd0aebb0a9f84b1aad680f4b7    gsoci.azurecr.io/giantswarm/pause:3.9         io.containerd.runc.v2
05df0356a344c23d02afbff797742c67bd    gsoci.azurecr.io/giantswarm/pause:3.9         io.containerd.runc.v2
066e0c8ee2962f276c4b7bb7d505e63f5b    gsoci.azurecr.io/giantswarm/pause:3.9         io.containerd.runc.v2
0a6685e4d54e94c4acc36dbbb1a2b356de    gsoci.azurecr.io/giantswarm/pause:3.9         io.containerd.runc.v2
```

Images

Image Pull Policy

- `IfNotPresent` - Fetches the image if not already on the node (default if you use a tag/sha)
- `Always` - Will always fetch the image (default if you use the tag as “latest” or omit the tag)
    - The cache mechanism compares the image layers from the registry and only pulls those missing
- `Never` - Will not attempt to fetch the image, it must be loaded onto the node by some other means

```yaml
apiVersion: v1
kind: Pod
metadata:
    name: "tiny-pod"
spec:
    containers:
    - name: "nginx"
      image: "nginx:v1.2.3"
      imagePullPolicy: "Always"
```

Image Tags - SHA

- Recommended best practice
- SHA-based image tag ensure ***exactly*** the same image is used each time, *even if tag it overwritten*
- If SHA is used, the tag is completely ignored and may no longer match the SHA! ⚠️
    - Be careful with automated dependency updaters - make sure the sha is also updated!

```yaml
apiVersion: v1
kind: Pod
metadata:
    name: "tiny-pod"
spec:
    containers:
    - name: "nginx"
      image: "nginx:1.25.1@sha256:9d6b58feebd2db...2072c9496"
      imagePullPolicy: "Always"
```

Meaningless / Ignored

RuntimeClass

RuntimeClass

- Allows for multiple runtimes in a single cluster
- If unset, uses default **Container Runtime Interface** (CRI) configured on the node
- If set, must point to a `RuntimeClass` resource name and have the **CRI handler** configured up on the node
- The `scheduling` property of the `RuntimeClass` ensures Pods are scheduled onto nodes with that runtime available (based on label selectors)

```yaml
apiVersion: v1
kind: Pod
metadata:
    name: "tiny-pod"
spec:
    runtimeClassName: "crio-runtime"
    containers:
    - name: "demo"
      image: "nginx:latest"
---
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
    name: "crio-runtime"
scheduling:
    nodeSelector:
        runtime: "crio"
handler: "crio"
```

RuntimeClass

- Allows for multiple runtimes in a single cluster
- If unset, uses default **Container Runtime Interface** (CRI) configured on the node
- If set, must point to a `RuntimeClass` resource name and have the **CRI handler** configured up on the node
- The `scheduling` property of the `RuntimeClass` ensures Pods are scheduled onto nodes with that runtime available (based on label selectors)
- Can be used for **WASM** (web assembly) runtimes, not just containers

```yaml
apiVersion: v1
kind: Pod
metadata:
    name: "wasm-pod"
spec:
    runtimeClassName: "wasmedge"
    containers:
    - name: "demo"
      image: "my-wasm-demo:latest"
---
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
    name: "wasmedge"
scheduling:
    nodeSelector:
        runtime: "wasmedge"
handler: "wasmedge"
```

Static Pods

Static Pods

- Managed directly by the Kubelet, not the API server
- Defined as static manifests either:
  - on disk of the node in the directory defined by `--pod-manifest-path`
  - or referenced from an URL using the `--manifest-url` flag
- The Kubelet automatically tries to create a “mirror Pod” on the API for each static Pod so that they are visible when querying the API server but they cannot be modified via the API
- Pod names get the node name as a suffix (e.g. `kube-scheduler-control-plane-1`)
- Cannot refer to other resources (e.g. ConfigMaps)
- The Kubelet watches the static directory and reconciles when files are changed/added/removed

File stored on the host node disk

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: kube-scheduler
 namespace: kube-system
spec:
 containers:
   - name: kube-scheduler
     image: kube-scheduler:v1.32.0
     command:
       - kube-scheduler
```

Lifecycle Hooks

Lifecycle Hooks

- Guaranteed to trigger *at least once* but may be called multiple times.
- `postStart`
  - Runs immediately after container is created but no guarantee that it will execute before the container's ENTRYPOINT.
  - The container isn't marked as "running" until this completes.
- `preStop`
  - Runs immediately before the container is terminated.
- Hook mechanisms available:
  - `exec` - perform command in container
  - `httpGet` - perform an HTTP GET request to the container

Example based on the Cilium chart provided by Bitnami

```yaml
apiVersion: v1
kind: Pod
metadata:
   name: cilium-agent
spec:
   containers:
   - name: cilium-agent
     image: "cilium:latest"
     lifecycle:
       postStart:
         exec:
           command:
           - /bin/bash
           - -ec
           - |
               if [[ "$(iptables-save | grep -E -c 'AWS-SNAT-CHAIN)"
                   iptables-save | grep -E -v 'AWS-SNAT-CHAIN' | ipta
               fi
       preStop:
         exec:
           command:
           - /opt/bitnami/scripts/cilium/uninstall-cni-plugin.sh
           - /host
```

Conditions & Readiness Gates

Conditions

Kubelet manages the following Pod Conditions:

- `PodScheduled` - the Pod has been scheduled to a node
- `PodReadyToStartContainers` - (beta feature) the Pod sandbox has been created and networking configured
- `ContainersReady` - all containers in the Pod are ready
- `Initialized` - all the initContainers have completed
- `Ready` - all containers ready and probes successfully passing

Each status condition may also contain...

- A 🤖machine readable `reason` property and
- A 🧑human readable `message` property

...that can be used for debugging.

```yaml
apiVersion: v1
kind: Pod
metadata:
   name: "demo-pod"
spec:
 ...
status:
   conditions:
   - type: Ready
     status: "False"
     lastProbeTime: null
   - type: PodScheduled
     status: "True"
     lastProbeTime: null
```

Readiness Gates

When container status and probes aren't enough to determine is a Pod really is ready then there is `readinessGates`!

These must me handled by some **external application** that patches the status of the Pod once the readiness gate condition is met.

Example usage: AWS Load Balancer supports readiness gates to indicate a pod is registered to the ALB/NLB.

```yaml
apiVersion: v1
kind: Pod
metadata:
   name: "aws-alb-example"
spec:
 readinessGates:
   - conditionType: "target-health.elbv2.k8s.aws/k8s-readines-perf1000-7848e5026b"
status:
 conditions:
   - type: "target-health.elbv2.k8s.aws/k8s-readines-perf1000-7848e5026b"
     status: "False"
     message: "Initial health checks in progress"
     reason: "Elb.InitialHealthChecking"
```

Config

Config

Environment Variables

Environment Variables

- Hardcoded & Dynamic, leveraging other environment variables with the `$(ENV)` syntax

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: "demo-pod"
spec:
 containers:
 - name: demo
   image: nginx
   env:
   - name: NAME
     value: "World"
   - name: GREETING
     value: "Hello, $(NAME)"
```

Environment Variables

- Hardcoded & Dynamic, leveraging other environment variables with the `$(ENV)` syntax
- The **Downward API** allows exposing properties from the Pod fields as env vars.
  Not all fields are valid but you can use fields from the Pod's metadata, spec, limits and status.

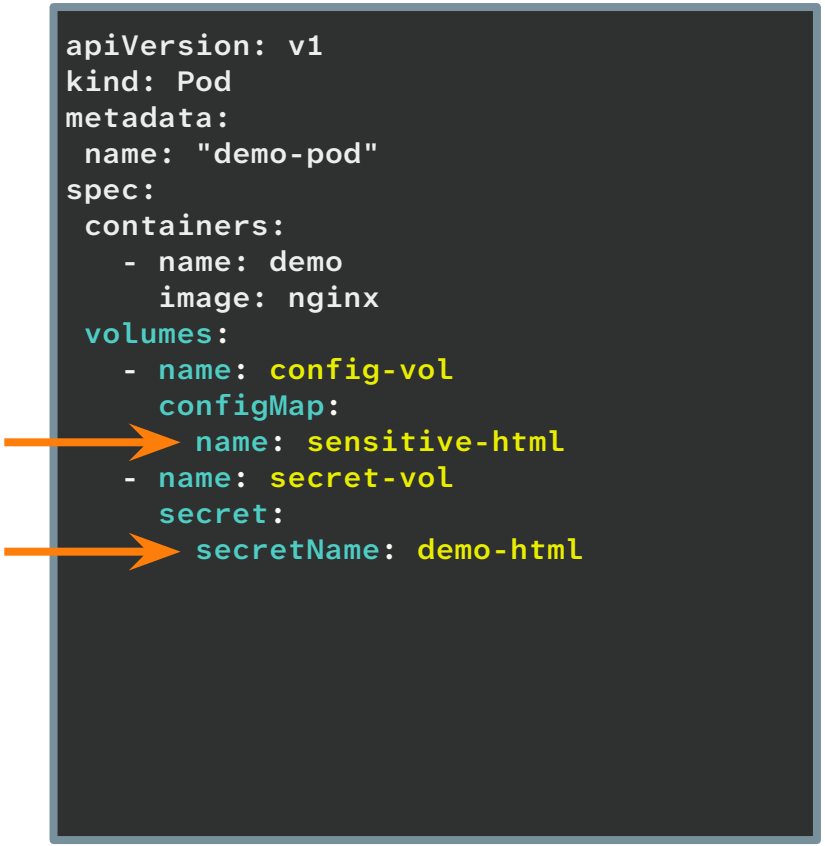
```yaml
apiVersion: v1
kind: Pod
metadata:
 name: "demo-pod"
spec:
 containers:
 - name: demo
   image: nginx
   env:
   - name: POD_NAME
     valueFrom:
       fieldRef:
         fieldPath: metadata.name
   - name: NODE_NAME
     valueFrom:
       fieldRef:
         fieldPath: spec.nodeName
   - name: POD_IP
     valueFrom:
       fieldRef:
         fieldPath: status.podIP
   - name: CONTAINER_MEM_LIMIT
     valueFrom:
       resourceFieldRef:
         containerName: demo
         resource: limits.memory
```

Config

Volumes

Volumes

- ConfigMaps vs. Secrets - `name` vs. `secretName`



Volumes

- ConfigMaps vs. Secrets - `name` vs. `secretName`
- Downward API

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: "demo-pod"
spec:
 containers:
   - name: demo
     image: nginx
     volumeMounts:
       - name: podinfo
         mountPath: /etc/podinfo
 volumes:
   - name: podinfo
     downwardAPI:
       items:
         - path: "labels"
           fieldRef:
             fieldPath: metadata.labels
         - path: "annotations"
           fieldRef:
             fieldPath: metadata.annotat
```

Becomes the filename

Volumes

- ConfigMaps vs. Secrets - `name` vs. `secretName`
- Downward API
- EmptyDir

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: "demo-pod"
spec:
 containers:
   - name: demo
     image: nginx
 volumes:
   - name: cache-volume
     emptyDir:
       medium: Memory
       sizeLimit: 500Mi
```

Recommended to avoid filling host node disk

Volumes

- ConfigMaps vs. Secrets - `name` vs. `secretName`
- Downward API
- EmptyDir
- Projected

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: "demo-pod"
spec:
 containers:
   - name: demo
     image: nginx
     volumeMounts:
     - name: web-content
       mountPath: /usr/share/nginx/html
 volumes:
   - name: web-content
     projected:
       sources:
       - configMap:
           name: web-index
           items:
             - key: index.html
               path: index.html
       - configMap:
           name: error-pages
```

Entire contents of ConfigMap data

Volumes

- ConfigMaps vs. Secrets - `name` vs. `secretName`
- Downward API
- EmptyDir
- Projected
- Image (KEP #4639)
  - Alpha in v1.31, Beta in v1.33, disabled by default
  - Allows mounting an OCI image as a volume
  - Pull secrets handled the same as container images
  - Container runtime needs to support it (CRI-O and Containerd have initial support available)

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: "demo-pod"
spec:
 containers:
   - name: demo
     image: nginx
     volumeMounts:
     - name: oci-content
       mountPath: /usr/share/nginx/html
       readOnly: true
 volumes:
 - name: oci-content
   image:
     reference: quay.io/crio/artifact:v1
     pullPolicy: IfNotPresent
```

Scheduling

Scheduling

Resource Allocation

🐘 @Marcus@k8s.social | 🌐 MarcusNoble.com | 🦋 @averagemarcus.bsky.social

Very brief overview!

Resource Requests & Limits

- Requests - min resources free on node to be scheduled
- Limits - enforced amount of resources a container has
  - CPU - enforced by CPU throttling
  - Memory - enforced by kernel out of memory (OOM) terminations

```yaml
apiVersion: v1
kind: Pod
metadata:
   name: "demo-pod"
spec:
   containers:
   - name: demo
     image: nginx
     resources:
       requests:
         memory: "64Mi"
         cpu: "250m"
       limits:
         memory: "128Mi"
         cpu: "500m"
```

Resource Requests & Limits

- Requests - min resources free on node to be scheduled
- Limits - enforced amount of resources a container has
  - CPU - enforced by CPU throttling
  - Memory - enforced by kernel out of memory (OOM) terminations
- Custom resource types can be managed by 3rd party controllers (e.g. `nvidia.com/gpu`)
  - Requests & limits must be the same
  - You cannot specify requests without limits

For GPU resources

```yaml
apiVersion: v1
kind: Pod
metadata:
   name: "demo-pod"
spec:
   containers:
   - name: demo
     image: nginx
     resources:
       requests:
         nvidia.com/gpu: 1
       limits:
         nvidia.com/gpu: 1
```

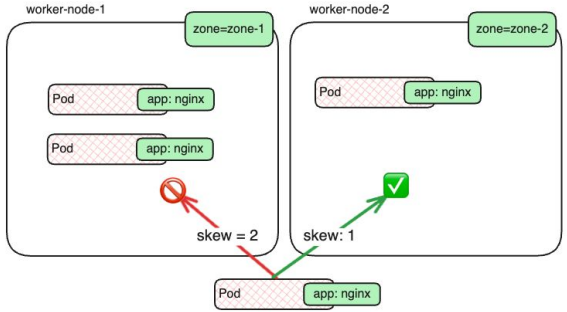
Resource Requests & Limits

- Requests - min resources free on node to be scheduled
- Limits - enforced amount of resources a container has
  - CPU - enforced by CPU throttling
  - Memory - enforced by kernel out of memory (OOM) terminations
- Custom resource types can be managed by 3rd party controllers (e.g. `nvidia.com/gpu`)
  - Requests & limits must be the same
  - You cannot specify requests without limits
- Pod limit and requests are calculated from the sum of all the containers
  - v1.32 introduces a new alpha (disabled by default) feature that supports pod-level resource specification

```yaml
apiVersion: v1
kind: Pod
metadata:
   name: "demo-pod"
spec:
   resources:
     requests:
       memory: "100Mi"
     limits:
       memory: "200Mi"
   containers:
   - name: demo
     image: nginx
```

Node Assignment

- `topologySpreadConstraints` - more control over the spread of Pods across a cluster when scaling replicas
  - `skew` = number of pods in topology - min pods in any topology
  - `topologyKey` - the label on nodes to use as the groupings (usually failure domains)
  - `whenUnsatisfiable` - Either `DoNotSchedule` or `ScheduleAnyway`
  - No guarantee the constraints remain satisfied when Pods are removed (e.g. scaling down)
  - Combined with other node assignment strategies (e.g. affinity)




```yaml
apiVersion: v1
kind: Pod
metadata:
   name: "demo-pod"
   labels:
       app: nginx
spec:
   topologySpreadConstraints:
     - maxSkew: 1
       topologyKey: zone
       whenUnsatisfiable: DoNotSchedule
       labelSelector:
           matchLabels:
               app: nginx
   containers:
       - name: demo
         image: nginx
```

DNS

- Usually, depending on the DNS mechanism used in the cluster (e.g. CoreDNS), each Pod also gets an A record
  - E.g. `172-17-0-3.default.pod.cluster.local`
- A Pods hostname is set to its `metadata.name` by default but can be overridden with the `spec.hostname` property and an additional subdomain set with `spec.subdomain`
  - E.g. `my-demo.example.default.svc.cluster.local`
  - (This doesn't mean other Pods can resolve that hostname)
- Add extra entries to the Pods `/etc/hosts` file with `spec.hostAliases`

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: demo
spec:
  hostname: my-demo
  subdomain: example
  setHostnameAsFQDN: true
  hostAliases:
  - ip: "127.0.0.1"
    hostnames:
    - "demo.local"
  containers:
  - name: demo
    image: nginx
```

Scheduling

Scheduler Logic

🐘 @Marcus@k8s.social | 🌐 MarcusNoble.com | 🦋 @averagemarcus.bsky.social

Priority & Preemption

- Pods can be given a priority to indicate their importance compared to other Pods. If a Pod is unable to be scheduled and has a higher priority than already scheduled Pods the scheduler will evict (preempt) the lower priority to make room
- `PriorityClass` resource is used to define possible priorities in a cluster
- PodDisruptionBudget are handled on a best-effort basis and not guaranteed to be honoured
- You can avoid preempting lower priority Pods by setting `preemptionPolicy: Never` on the `PriorityClass`
    - This effects the scheduler queue but doesn't cause pods to be evicted

```yaml
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
   name: high-priority
value: 1000000
globalDefault: false
description: These pods are important

---

apiVersion: v1
kind: Pod
metadata:
   name: demo
spec:
   priorityClassName: high-priority
   containers:
        - name: demo
          image: nginx
```

Multiple / Alternative Schedulers

- `schedulerName` - indicates which scheduler a Pod should be managed by
- In not set, or set to `default-scheduler` then the built-in Kubernetes scheduler is used

```yaml
apiVersion: v1
kind: Pod
metadata:
   name: demo
spec:
   schedulerName: custom-scheduler
   containers:
       - name: demo
         image: nginx
```

Networking

DNS Policy

- Define how the Pods DNS configuration defaults should be specified:
  - `Default` - inherits the nodes DNS resolution config
  - `ClusterFirst` - matches against in-cluster resources first before sending forwarding to an upstream nameserver

  - `ClusterFirstWithHostNet` - should be used when using host network otherwise the Pod will fallback to `Default`
  - `None` - ignores all DNS config from cluster and expects all to be set via `dnsConfig`

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: demo
spec:
 dnsPolicy: ClusterFirst
 containers:
 - name: demo
   image: nginx
```

DNS Config

- More control over the DNS settings used within a Pod
- `nameservers` - a list of IPs to use as the DNS servers (max 3)
- `searches` - list of DNS search domains to use for hostname lookup, merged into the base search domains generated (max 32)
- `options` - a list of name/value pairs to define extra DNS configuration options

```
apiVersion: v1
kind: Pod
metadata:
 name: demo
spec:
 containers:
 - name: busybox
   image: nginx
   dnsPolicy: "None"
   dnsConfig:
     nameservers:
       - 192.0.2.1
     searches:
       - ns1.svc.cluster.example
       - my.dns.custom
     options:
       - name: ndots
         value: "2"
       - name: edns0
```

Only use the below config

```
# cat /etc/resolv.conf
nameserver 192.0.2.1
search ns1.svc.cluster.example my.dns.custom
options ndots:2 edns0
```

Wrap-up

Slides and resources available at: https://go-get.link/cnsmunich

Thoughts, comments and feedback:

MarcusNoble.com

k8s.social/@Marcus

@averagemarcus.bsky.social