Kubernetes Pod internals
with the fundamentals of Containers

Hyojun Jeon (https://hyojun.me)

What is a Kubernetes Pod?

What is a Kubernetes Pod?

- The smallest deployable unit in Kubernetes
- A group of containers

What is a Kubernetes Pod?

- The smallest deployable unit in Kubernetes
- A group of containers

→ To understand Kubernetes Pods,
we need to understand containers

A container is
a “process” running on an “isolated environment”

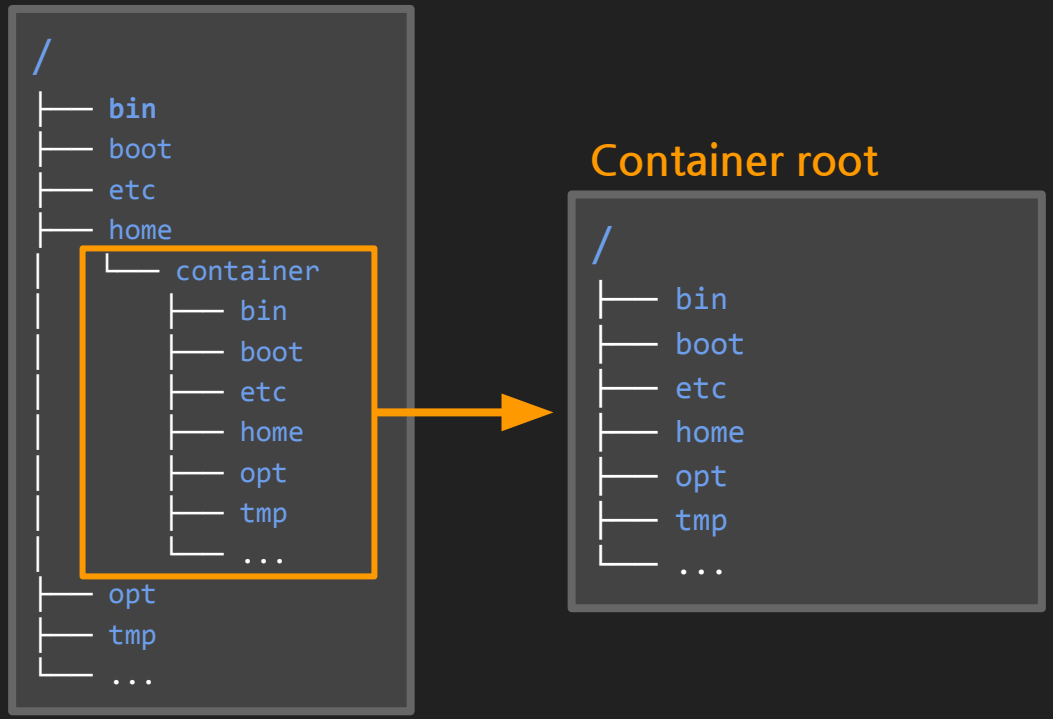
How are containers isolated?

- Root directory isolation (chroot)
- Linux namespaces
  - Mount (mnt)
  - Process ID (pid)
  - Network (net)
  - Interprocess Communication (ipc)
  - Unix Time-Sharing (uts)
  - User ID (user)
- Control groups (cgroup)
- OverlayFS
- … etc.

→ Let's find out what these are.

How are containers isolated?

(1) Isolating root directory (chroot)



How are containers isolated?

(1) Isolating root directory (chroot)

```
05:16ubuntu@ubuntu>container> pwd
/home/ubuntu/workspace/container-lecture/home/container
05:16ubuntu@ubuntu>container> tree -L 2 ./
./
├── bin
│   ├── bash
│   └── ls
├── lib
│   ├── libc.so.6
│   ├── libdl.so.2
│   ├── libpcre.so.3
│   ├── libpthread.so.0
│   ├── libselinux.so.1
│   └── libtinfo.so.5
└── lib64
    └── ld-linux-x86-64.so.2

3 directories, 9 files
05:16ubuntu@ubuntu>container> sudo chroot ./ /bin/bash
bash: warning: setlocale: LC_ALL: cannot change locale (en_US.UTF-8)
bash-4.4# ls
bin  lib  lib64
```

bash / ls binaries

bash / ls dependencies

chroot

Sets an isolated root directory (a new root path) for a process and its children

```
$ chroot <NEWROOT> <COMMAND>
```

How are containers isolated?

(2) Linux Namespaces

Linux Namespace

→ A kernel feature to isolate system resources between processes

$ lsns -p <pid>

→ Lists the namespaces of the specified process.

```
06:26ubuntu@ubuntu>container> sudo lsns -p 95359
        NS TYPE   NPROCS   PID USER COMMAND
4026531835 cgroup    150     1 root /sbin/init maybe-ubiquity
4026531837 user      150     1 root /sbin/init maybe-ubiquity
4026532493 mnt         1 95359 root sleep 3600
4026532494 uts         1 95359 root sleep 3600
4026532495 ipc         1 95359 root sleep 3600
4026532496 pid         1 95359 root sleep 3600
4026532498 net         1 95359 root sleep 3600
```

This process is running on "cgroup", "user" namespaces of the init process.

Isolated namespaces for this process

How are containers isolated?

(2) Linux Namespaces

unshare

→ Runs a process with isolated namespaces.

```
# Run a process(/bin/bash) with an isolated mount namespace(-m option).
$ unshare -m /bin/bash
# Run a process (/bin/bash) with isolated mount (-m) and IPC (-i) namespaces
$ unshare -m -i /bin/bash
```

How are containers isolated?

(3) Mount (mnt) namespace

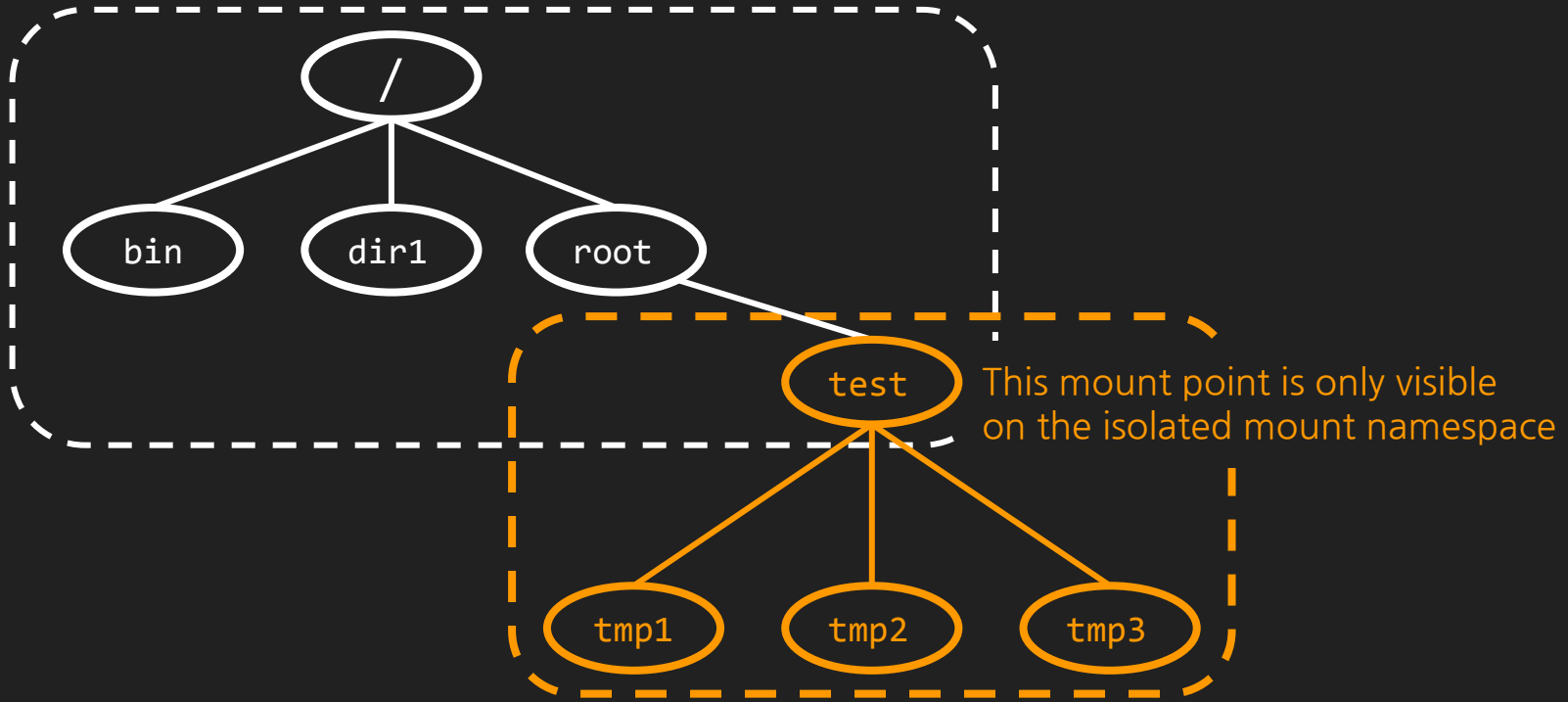
Mount

A command to attach a file system to the big file tree, to make it accessible in Unix systems.

```
# mount -t <type> <device> <dir>

# e.g. Mount “tmpfs”(temporary file storage) into “/root/test”
$ mount -t tmpfs tmpfs /root/test
```

How are containers isolated?

(3) Mount (mnt) namespace



How are containers isolated?

(3) Mount (mnt) namespace

Mount namespace

Allows processes to have different mount points

```
$ echo $$
1111
$ unshare -m /bin/bash
$ echo $$
2222
$ mkdir -p test && mount -t tmpfs tmpfs /root/test
$ df | grep test
tmpfs           2.0G     0  2.0G   0% /root/test
$ exit
$ df | grep test
```

Show the current process ID

Run `/bin/bash` with an isolated mount namespace(`-m` option)

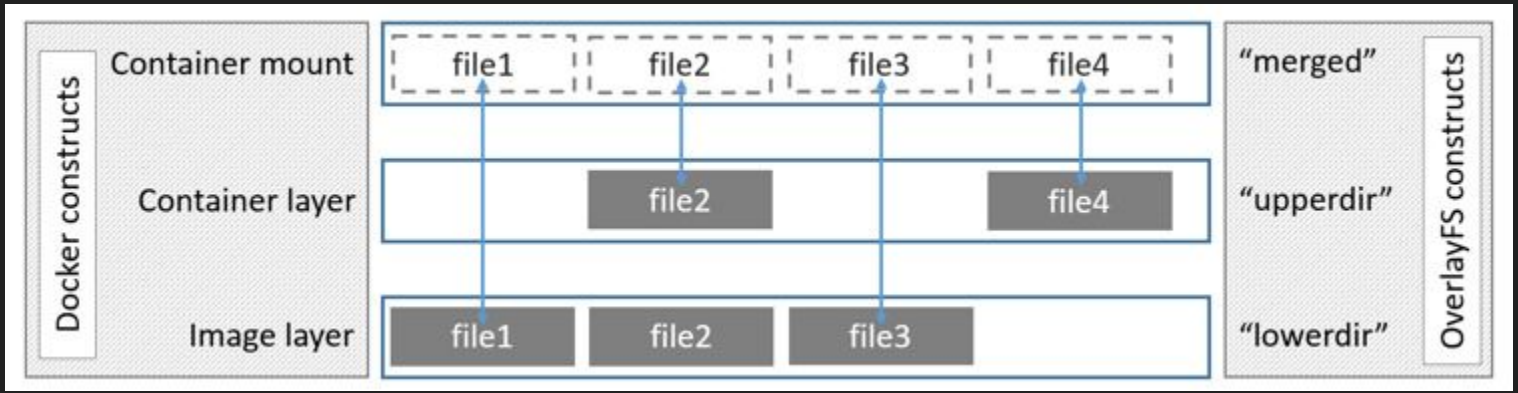
Mount tmpfs(temporary file storage) into `/root/test`

Exit `/bin/bash`(isolated mount namespace) and then check the file systems. The file system mounted into `/root/test` shouldn't be visible.

How are containers isolated?

(3) Mount (mnt) namespace + OverlayFS

OverlayFS storage driver

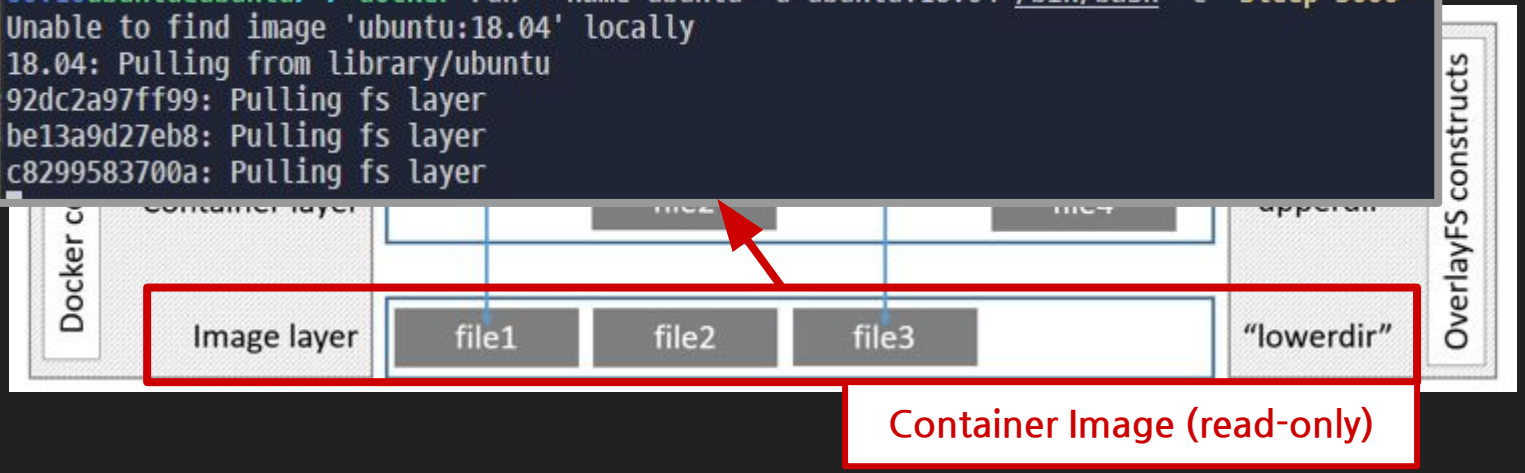


How are containers isolated?

(3) Mount (mnt) namespace + OverlayFS

OverlayFS storage driver

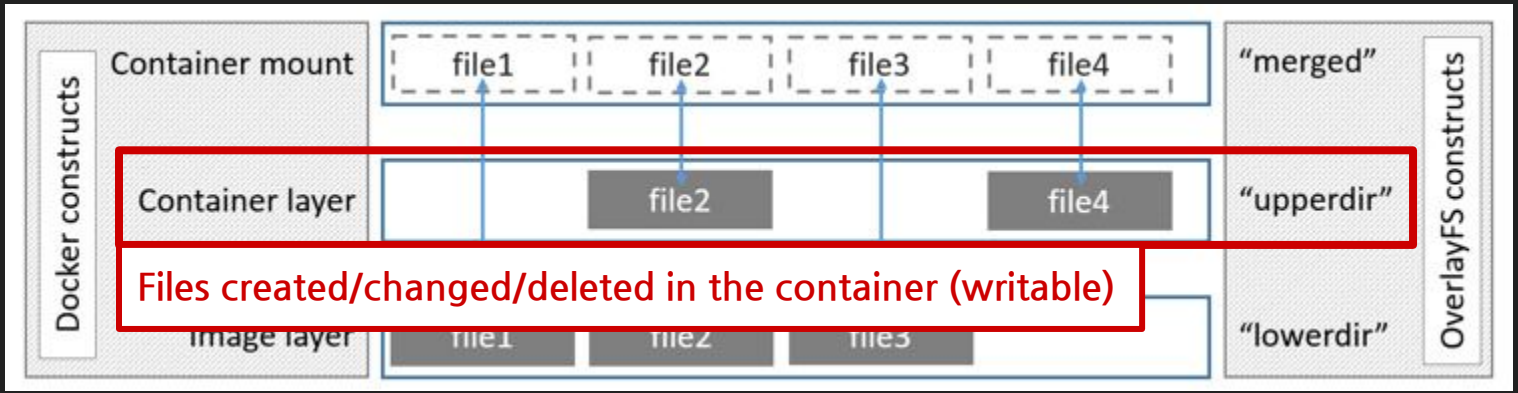
```
00:16ubuntu@ubuntu›~› docker run --name ubuntu -d ubuntu:18.04 /bin/bash -c "sleep 3600"
Unable to find image 'ubuntu:18.04' locally
18.04: Pulling from library/ubuntu
92dc2a97ff99: Pulling fs layer
be13a9d27eb8: Pulling fs layer
c8299583700a: Pulling fs layer
```



How are containers isolated?

(3) Mount (mnt) namespace + OverlayFS

OverlayFS storage driver



How are containers isolated?

(3) Mount (mnt) namespace + OverlayFS

OverlayFS storage driver

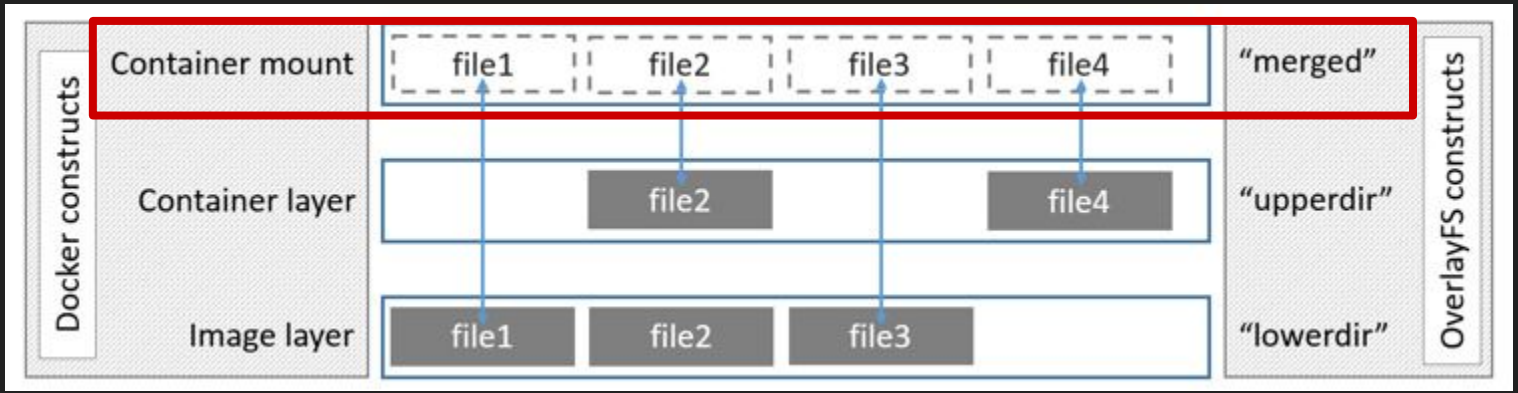


Image Layer + Container Layer = merged = The final mounted file system

How are containers isolated?

(3) Mount (mnt) namespace + OverlayFS

OverlayFS storage driver

```
$ docker inspect ubuntu | jq ".[].GraphDriver"
{
  "Data": {
    "LowerDir": "/var/lib/docker/overlay2/.../diff",
    "MergedDir": "/var/lib/docker/overlay2/.../merged",
    "UpperDir": "/var/lib/docker/overlay2/.../diff",
    "WorkDir": "/var/lib/docker/overlay2/.../work"
  },
  "Name": "overlay2"
}
```

*workdir

How are containers isolated?

(3) Mount (mnt) namespace + OverlayFS

```
00:34✘ubuntu@ubuntu〉~〉 docker exec -ti ubuntu /bin/bash
root@84c2fb9b121e:/# df -h
Filesystem      Size  Used Avail Use% Mounted on
overlay          98G   25G   69G  26% /
tmpfs            64M     0   64M   0% /dev
tmpfs           2.0G     0  2.0G   0% /sys/fs/cgroup
shm              64M     0   64M   0% /dev/shm
/dev/sda2        98G   25G   69G  26% /etc/hosts
```

How are containers isolated?

(4) Process ID (pid) namespace

Container is

a “Process” running on an “isolated environment”

```
03:30ubuntu@ubuntu)~) docker run --name ubuntu -d ubuntu:18.04 /bin/bash -c "sleep 3600"
e51f36f7e189508fbb0093dd0b3de82655a7582b1e85a10a04901602f6cf8688
03:30ubuntu@ubuntu)~) ps -ef | grep sleep
root       64387   64348  0 03:30 ?        00:00:00 sleep 3600
```

From inside the container, it looks like a virtual machine.
But from the outside (host), it's just a process.

How are containers isolated?

(4) Process ID (pid) namespace

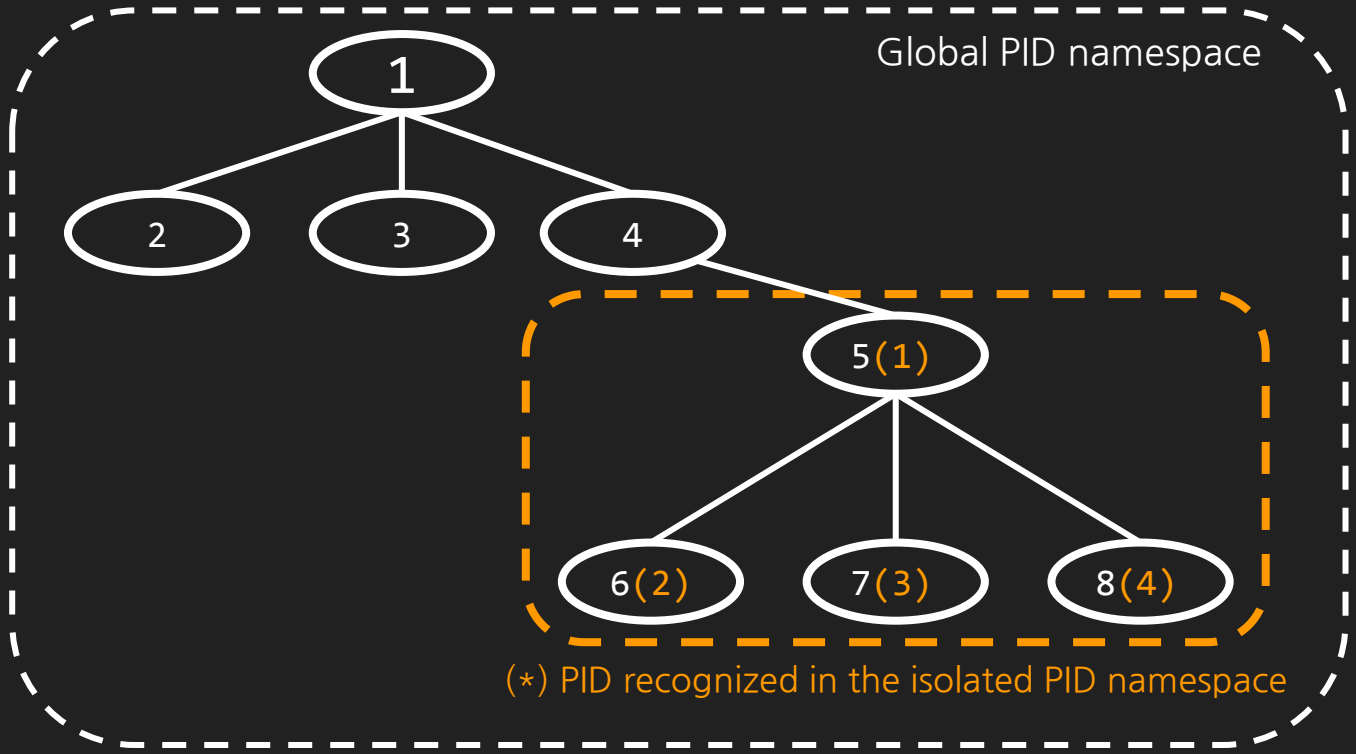
For the same process,
The PIDs are different between outside and inside the container.

```
07:59ubuntu@ubuntu>~> docker run --name ubuntu -d ubuntu:18.04 /bin/bash -c "sleep 3600"
d1d0b10115f92762407544170332 4fa99d756d833045fbb6108ddb7ba5fd5bf5
07:59ubuntu@ubuntu>~> ps -ef | grep sleep
root      115679  115645  0 07:59 ?        00:00:00 sleep 3600
ubuntu    115734  115095  0 07:59 pts/1    00:00:00 grep --color=auto --exclude-dir=.bzr
=.svn --exclude-dir=.idea --exclude-dir=.tox sleep
07:59ubuntu@ubuntu>~> docker exec ubuntu ps
    PID TTY          TIME CMD
      1 ?        00:00:00 sleep
      8 ?        00:00:00 ps
```

Outside(host) PID=115679
Inside(container) PID=1

How are containers isolated?

(4) Process ID (pid) namespace



How are containers isolated?

(4) Process ID (pid) namespace

```
# Run a process (/bin/bash) with an isolated PID namespace (-p option).
$ unshare -f -p /bin/bash
$ echo $$  # Show the current PID
1
```

Inside a container running on an isolated PID namespace,
the first executed process (the entrypoint) always has the PID of 1.

How are containers isolated?

(5) Inter-Process Communication (ipc) namespace

Isolates Inter-Process Communication (based on System V)

- System V IPC
  - Shared memory(shm)
  - Semaphores
  - POSIX message queues(/proc/sys/fs/mqueue)
- IPC objects are visible only to the processes on the same namespace

How are containers isolated?

(6) Network (net) namespace

Isolates network interfaces, routing tables, and firewall rules.

```
# Create a network namespace named `test-ns`
$ ip netns add test-ns
$ ip netns list
test-ns

# Create a virtual ethernet interface pair (veth1, veth2)
# Add `veth1` to `test-ns` namespace, and `veth2` to the network namespace of PID 1
$ ip link add veth1 netns test-ns type veth peer name veth2 netns 1
```

How are containers isolated?

(6) Network (net) namespace

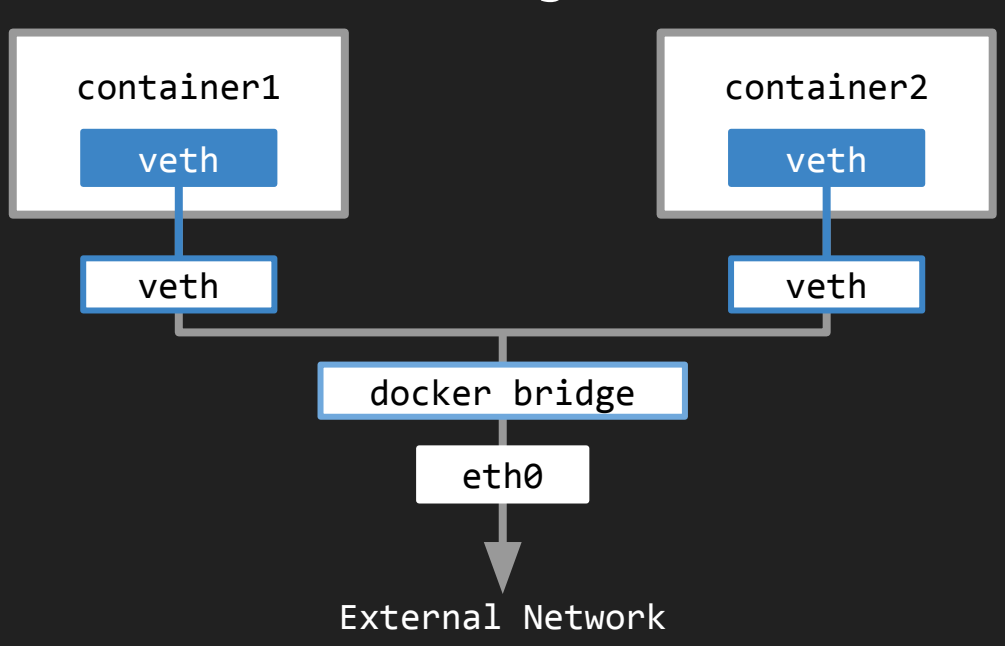
Isolates network interfaces, routing tables, and firewall rules.

```
# On the new namespace `test-ns`, run the command “ip link list” to list network interfaces.
# In `test-ns`, there are only `veth1` and loopback interfaces.
$ ip netns exec test-ns ip link list

1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
8: veth1@if7: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/ether 2a:aa:60:ee:27:d4 brd ff:ff:ff:ff:ff:ff link-netnsid 0

# On the host default network namespace, run the command “ip link list”.
$ ip link list

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
(... omit ...)
7: veth2@if8: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/ether d2:a1:90:78:3c:4b brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

How are containers isolated?

(6) Network (net) namespace

Docker's networking



Containers are running on an isolated network namespace, connected to the host via veth peers.

The veths are connected to the “docker bridge” on the host.
To communicate with the outside world,
the containers have to go through the bridge.

How are containers isolated?

(7) Unix Time-Sharing (uts) namespace

Unix Time-Sharing?

This comes from the concept of sharing computing resources among multiple users.

Multiple users are sharing the same machine,
but we want to make them feel like they're using separate machines.

How are containers isolated?

(7) Unix Time-Sharing (uts) namespace

Unix Time-Sharing?

This comes from the concept of sharing computing resources among multiple users.

Multiple users are sharing the same machine,
but we want to make them feel like they're using separate machines.

➡ Make a namespace for each user to isolate the hostnames!

How are containers isolated?

(7) Unix Time-Sharing (uts) namespace

hostname, domainname isolation

```
$ hostname
ubuntu
```
Show the current hostname

```
$ unshare -u /bin/bash
```
Run /bin/bash on an isolated UTS namespace

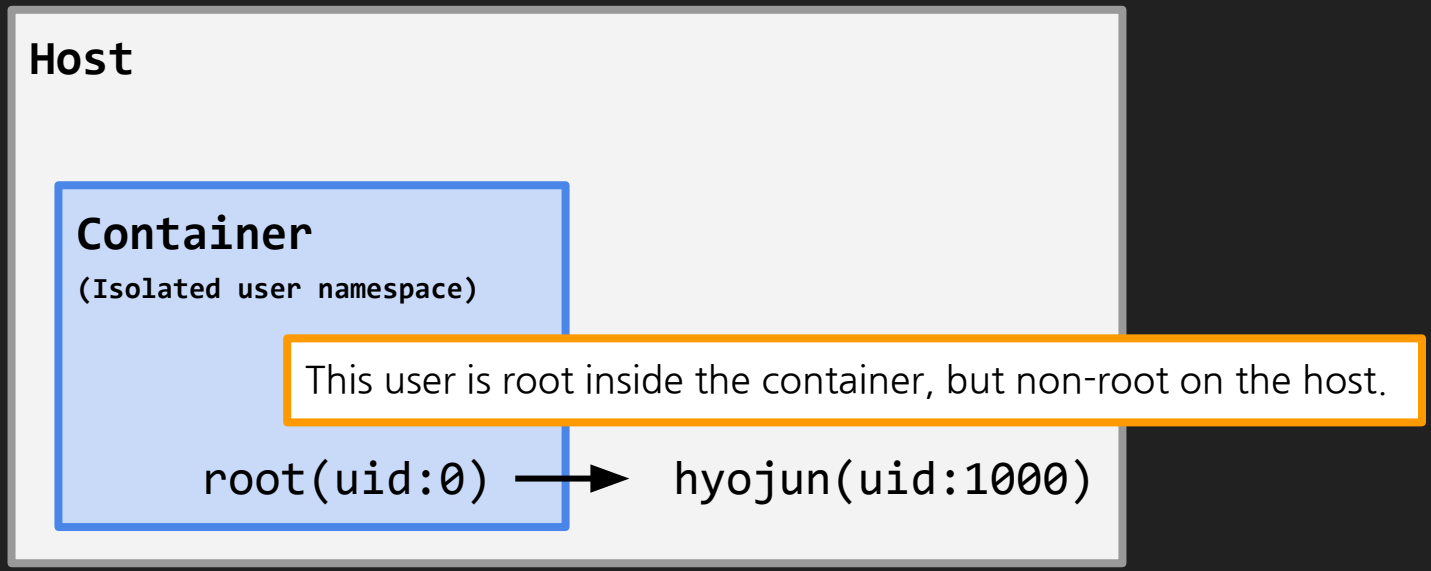
```
$ hostname hyojun
$ hostname
hyojun
```
Change hostname to “hyojun” and then show the current hostname

```
$ exit
$ hostname
ubuntu
```
Exit bash, and then show the current hostname.
= it keeps the original hostname(ubuntu)
(The hostname was changed only in the process where the UTS namespace was isolated.)

How are containers isolated?

(8) User ID (user) namespace

Map a different uid for a host user



How are containers isolated?

(8) User ID (user) namespace

However, in Docker, the user namespace is not isolated.

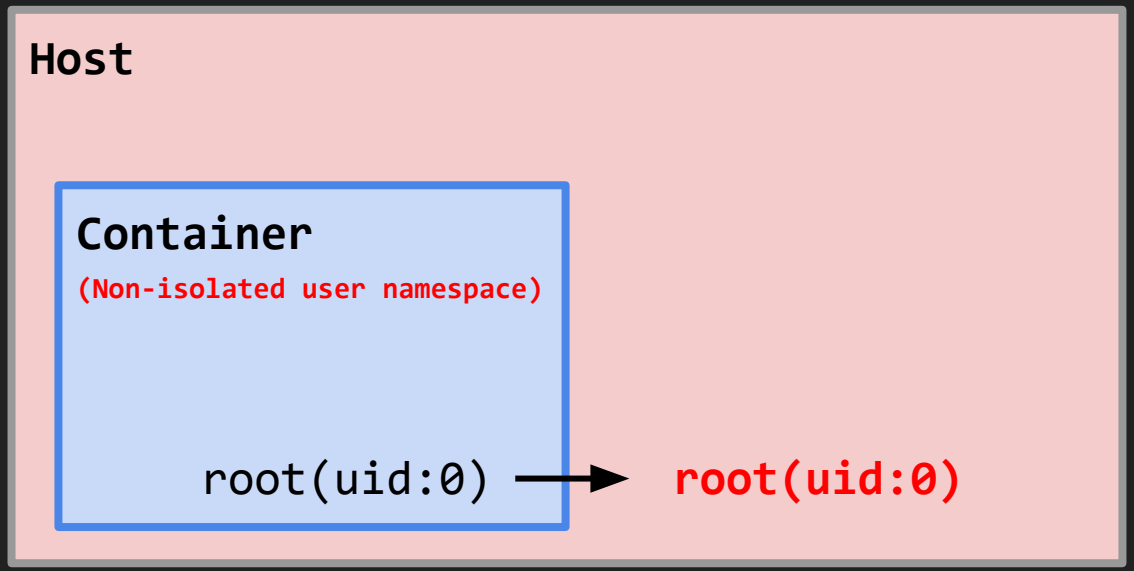
Docker container use the namespace of host PID=1 by default

```
06:26ubuntu@ubuntu>container> sudo lsns -p 95359
        NS TYPE   NPROCS   PID USER COMMAND
4026531835 cgroup    150     1 root /sbin/init maybe-ubiquity
4026531837 user      150     1 root /sbin/init maybe-ubiquity
4026532493 mnt         1 95359 root sleep 3600
4026532494 uts         1 95359 root sleep 3600
4026532495 ipc         1 95359 root sleep 3600
4026532496 pid         1 95359 root sleep 3600
4026532498 net         1 95359 root sleep 3600
```

How are containers isolated?

(8) User ID (user) namespace

If the user namespace of the host is shared to its containers,
the users inside the containers can exercise the authority of the same uid on the host.



How are containers isolated?

(8) User ID (user) namespace

This is the command you've run at least once after installing Docker.

```
$ sudo usermod -aG docker <your-user>
```

> After installing Docker,
> add a user to docker group so that non-root users can run docker

How are containers isolated?

(8) User ID (user) namespace

This is the command you've run at least once after installing Docker.

```
$ sudo usermod -aG docker <your-user>
```

WARNING!

- After installing Docker,
  add a user to docker group so that non-root users can run docker

How are containers isolated?

(8) User ID (user) namespace

If you bind the host's “/” root directory to the container…

```
non-root$ docker run -ti -v /:/host ubuntu:18.04 /bin/bash
```

Users without root privileges can exercise root authority
to the bound host root directory through Docker.
This is because the container root uid 0 has the same uid on the host.

How are containers isolated?

(8) User ID (user) namespace

Why doesn't Docker isolate the user namespace?

- Compatibility issues with sharing PID and Network namespaces.
- Compatibility issues with external volumes or drivers that do not support user mapping.
- The complexity of ensuring access rights for the files bound from the host, if the host uid and the container uid differ.
- However, while the root on a unisolated user namespace has *almost* the same permissions as the host's root, it does not include *all* permissions.

How are containers isolated?

(8) User ID (user) namespace

Kubernetes does not support user namespace isolation yet, either.

A good article to read on this topic:
https://kinvolk.io/blog/2020/12/improving-kubernetes-and-container-security-with-user-namespaces/

How are containers isolated?

(8) User ID (user) namespace

When not isolating the user namespace…

- Restrict only trusted users to run the container runtime (e.g. Docker).
- Make sure that the container's processes do not run as the root user.
  - Assign a specific user and group to run processes
- Do not bind any of the host's important directories, to prevent containers from accessing them.

Kubernetes provides security settings based on the same principles.

https://kubernetes.io/docs/concepts/policy/pod-security-policy/#users-and-groups

How are containers isolated?

(9) Control group (cgroup)

A Linux kernel feature to limit and isolate resource allocations among process groups

- CPU
- Memory
- Network
- Disk

Limit CPU and memory usage…
Prioritize network traffic, …
Provide statistics on usage, or etc.

How are containers isolated?

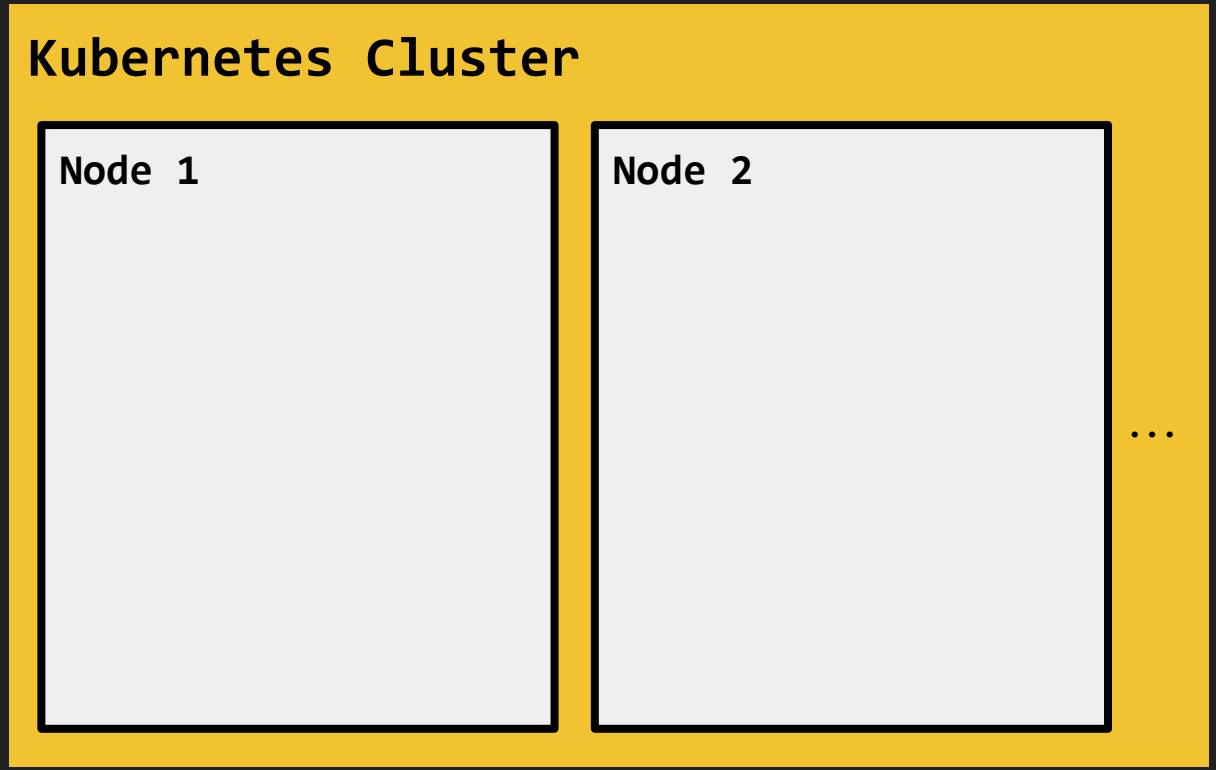
Recap

- A container is a **process** running on an **isolated environment**.
- Isolated environments are implemented through **Linux namespaces**.
  - Mount (mnt)
  - Process ID (pid)
  - Network (net)
  - Interprocess Communication (ipc)
  - Unix Time-Sharing (uts)
  - User ID (user)
- Processes' resource usage are limited through **cgroups**.

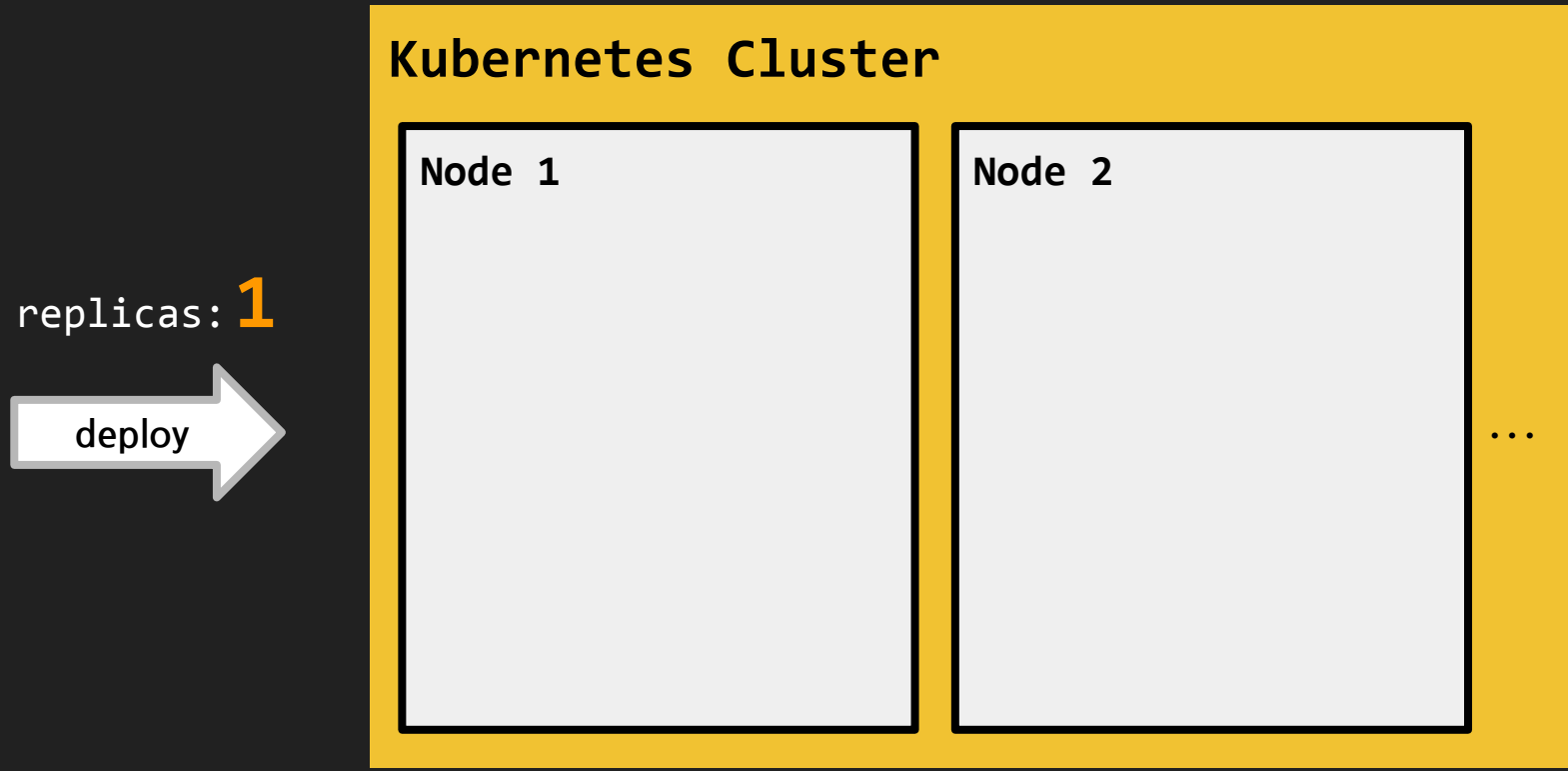
What is a Kubernetes Pod?

- The smallest deployable unit in Kubernetes
- A group of containers

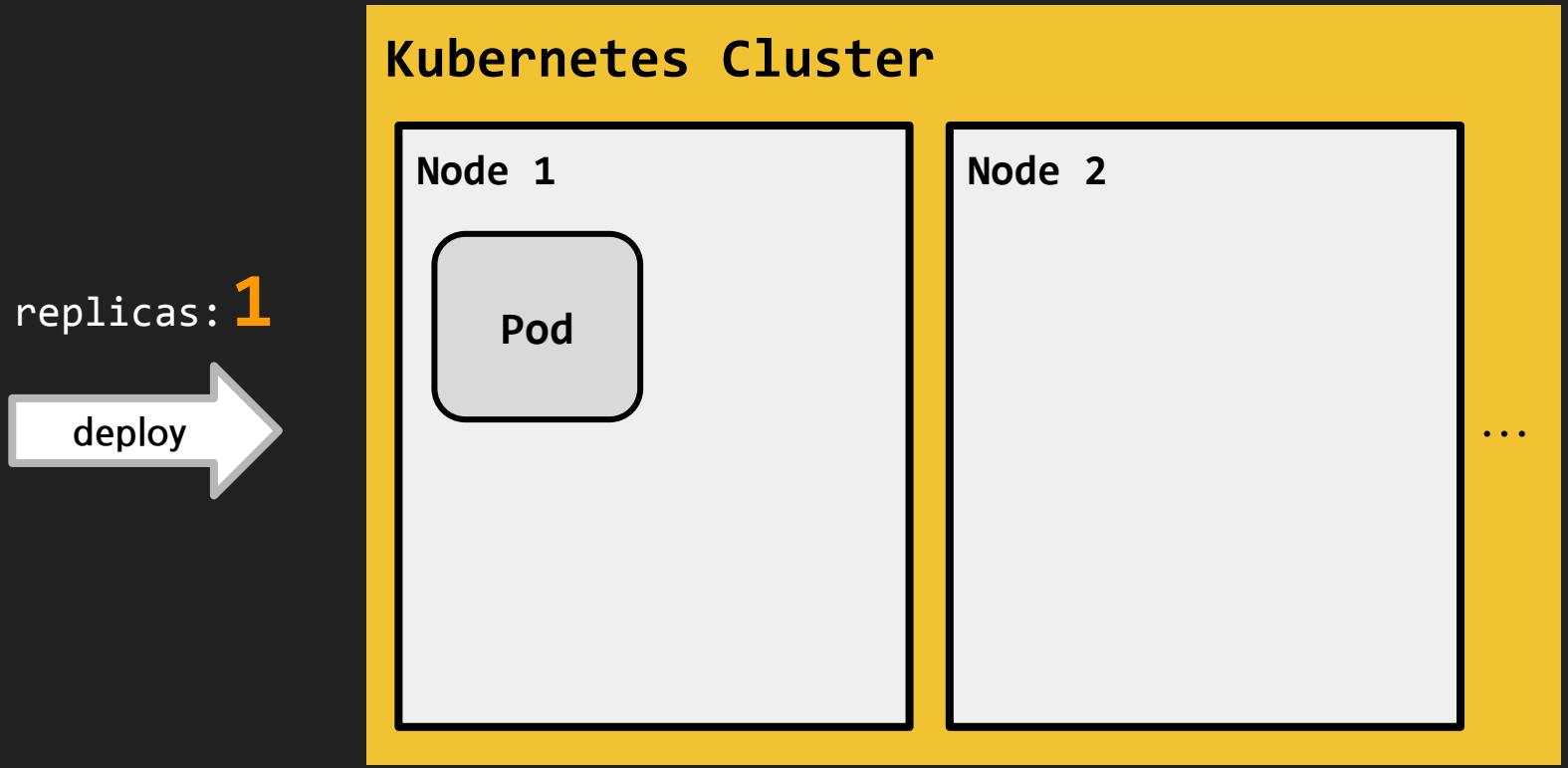
K8s applications are deployed as Pods.



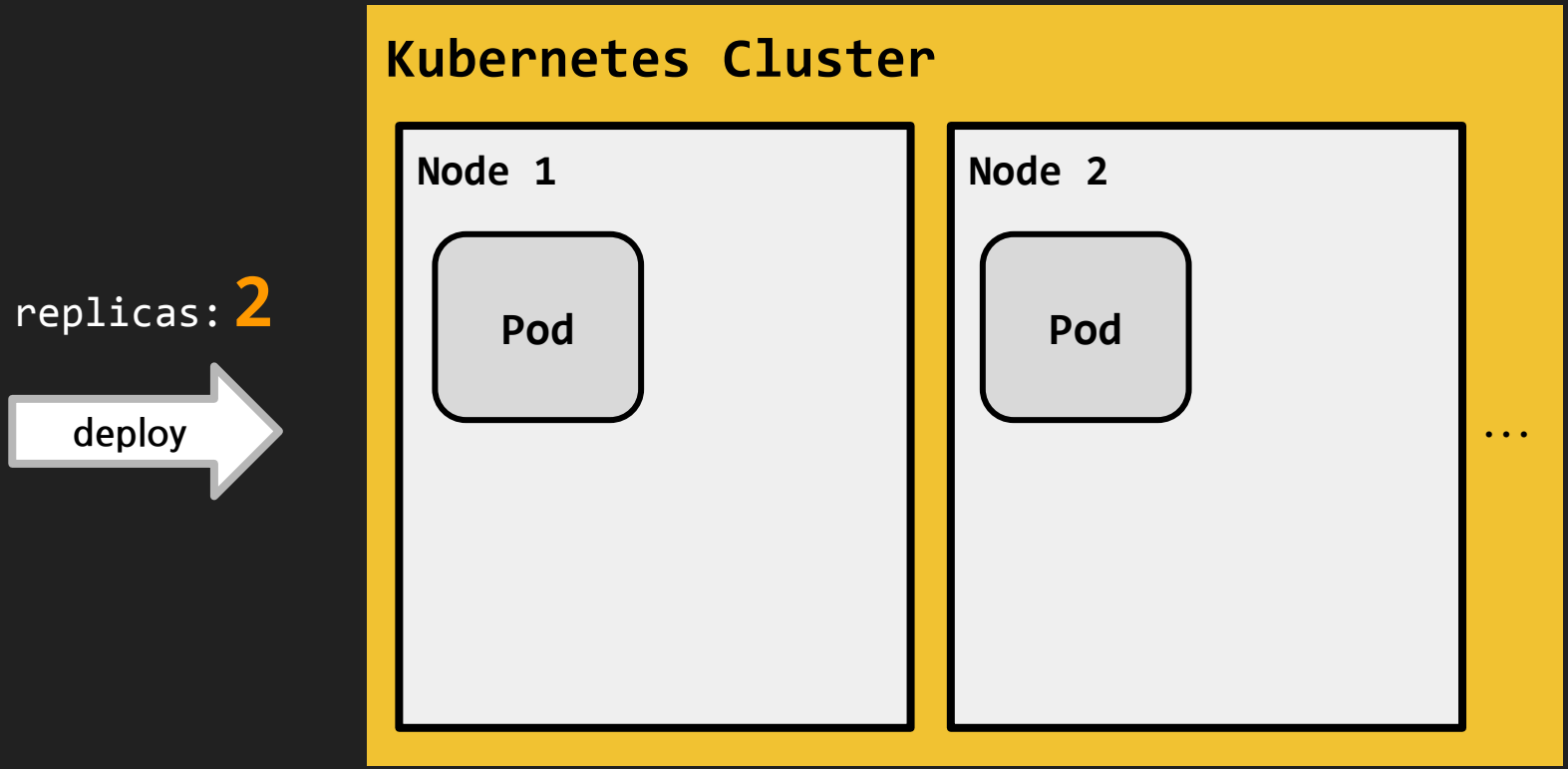
K8s applications are deployed as Pods.



K8s applications are deployed as Pods.



K8s applications are deployed as Pods.



“The smallest deployable unit”?

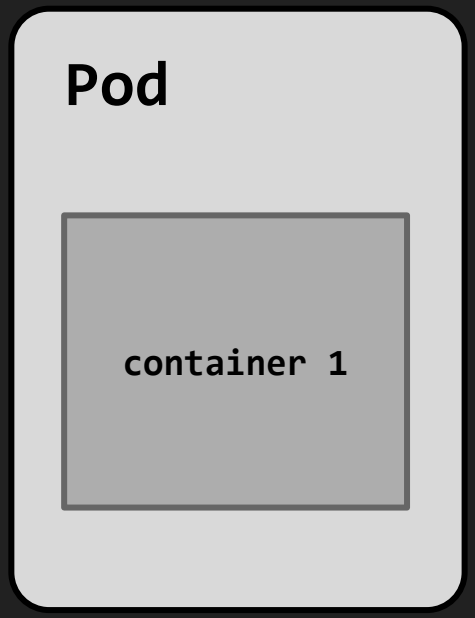
- In most cases, Pods are usually managed by using the below types of workload resources.
  - **Job** - Manages Pods that are executed once and terminated when the task is completed.
  - **ReplicaSet** - Ensures that the specified number of Pods(replica) are running.
  - **DaemonSet** - Ensures that only one pod running for each node
  - **StatefulSet** - Manages Pods running stateful applications
  - **Deployment** - Manages deployment of Pod, ReplicaSet updates

A pod is the most basic and smallest unit created and managed by Kubernetes.

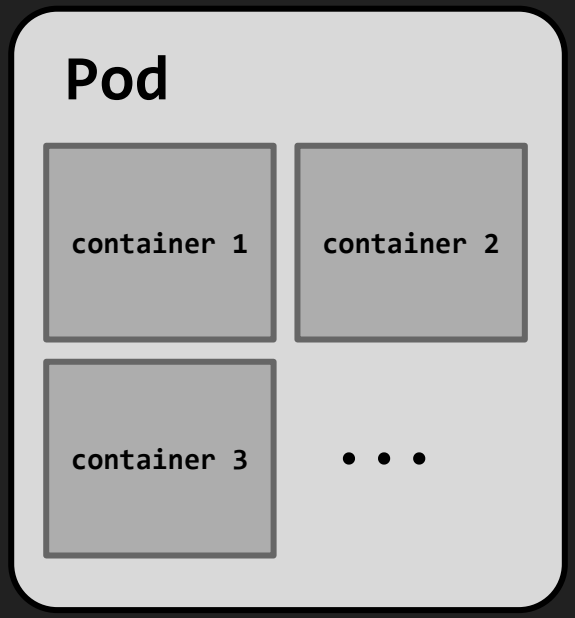
What is a Kubernetes Pod?

- The smallest deployable unit in Kubernetes
- The group of containers

There can be more than one container in a pod.

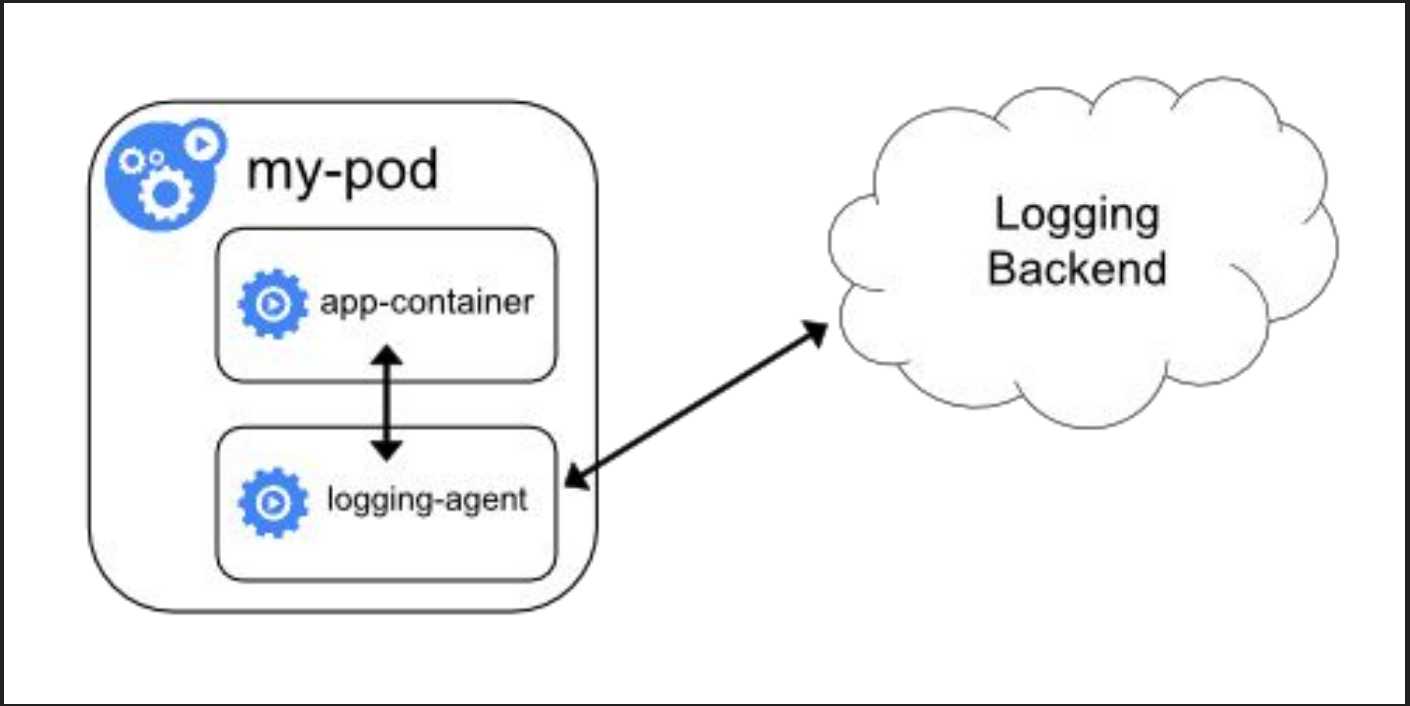


A pod with a single container



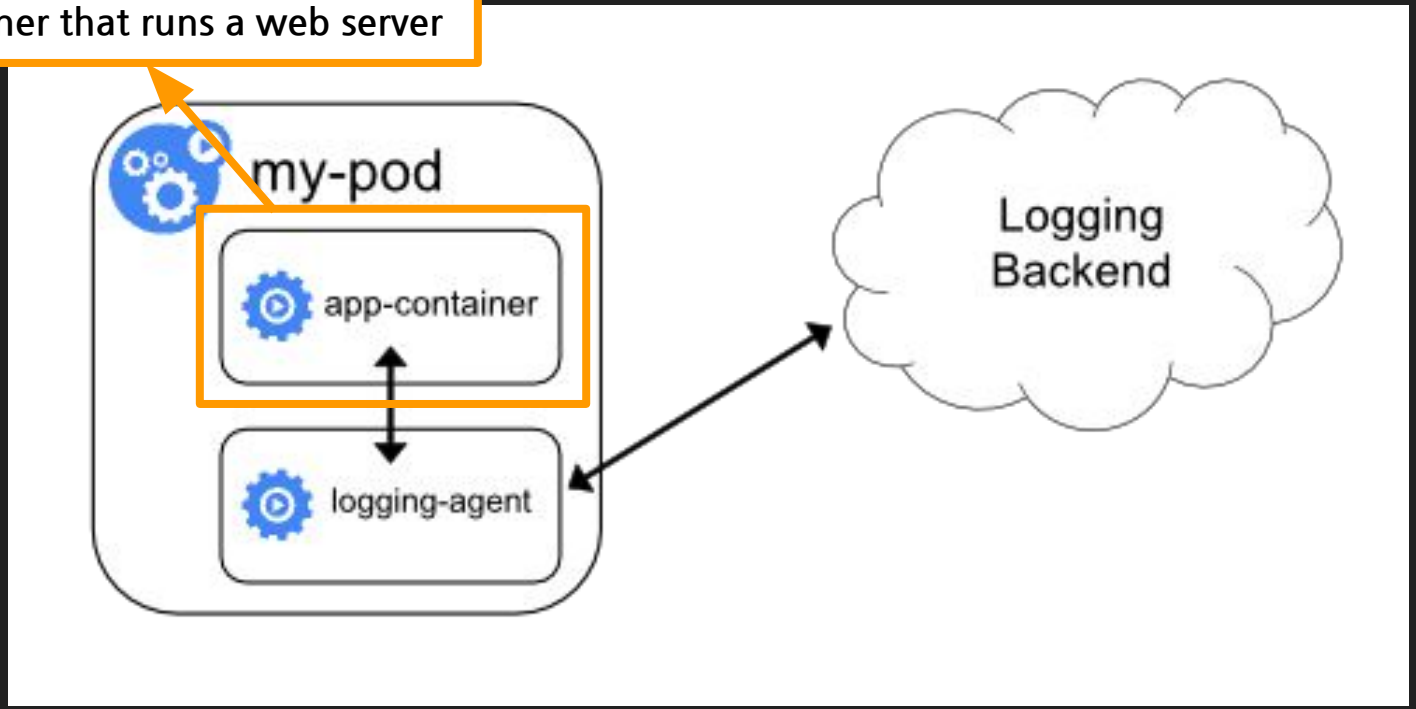
A pod with multiple containers

A case of running multiple containers in a pod



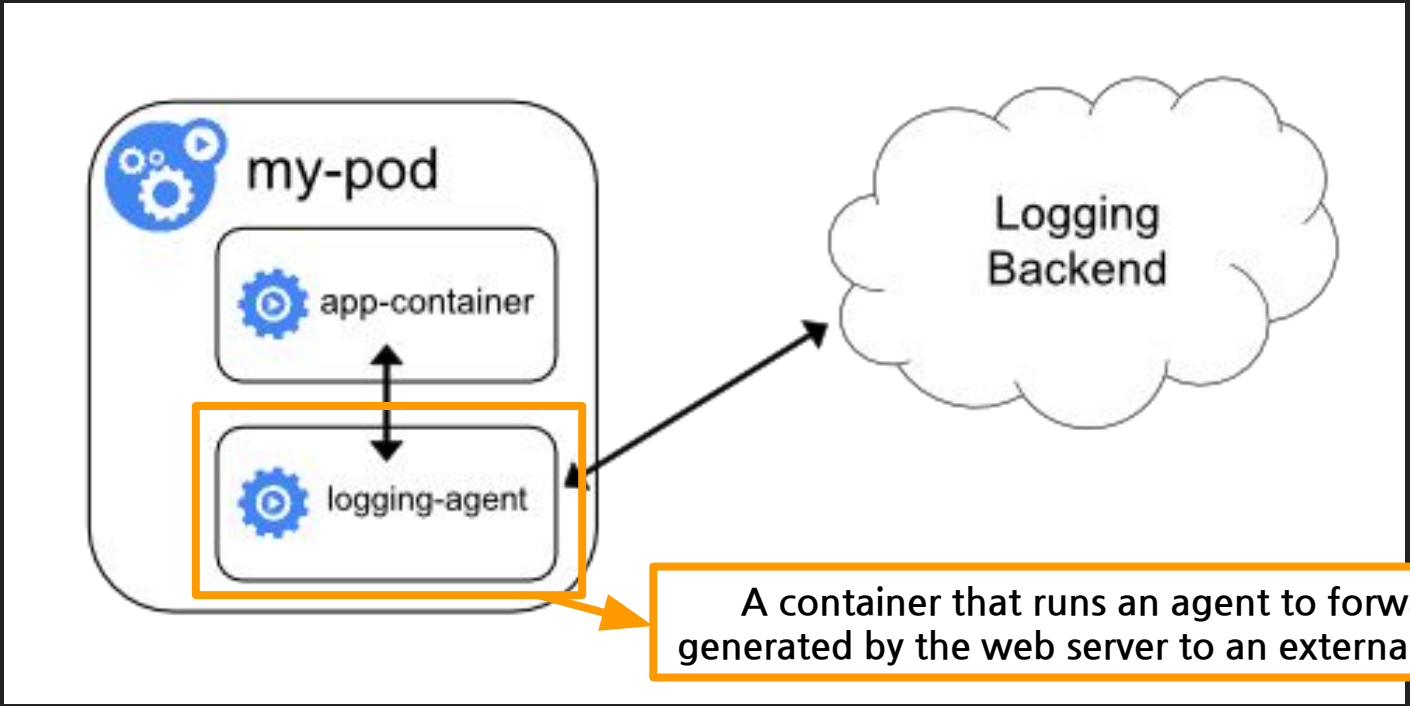
https://kubernetes.io/docs/concepts/cluster-administration/logging/#sidecar-container-with-a-logging-agent

A case of running multiple containers in a pod



https://kubernetes.io/docs/concepts/cluster-administration/logging/#sidecar-container-with-a-logging-agent

A case of running multiple containers in a pod



https://kubernetes.io/docs/concepts/cluster-administration/logging/#sidecar-container-with-a-logging-agent

When a pod consists of multiple containers...

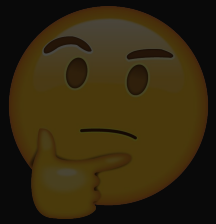
- One **Primary Container** that plays the main role
- One or more **Sidecar Containers**
  - It serves supporting features for the primary container
    e.g. Monitoring, Logging, etc...

"Sidecar" attached to a motorcycle

https://pixabay.com/photos/bmw-500-old-motorcycle-sidecar-4344066/

That seems complicated…

Can't we run all processes in one container?



That seems complicated...

Can't we run all processes in one container?

You can, but that's not recommended.

In a container, it’s recommended to run a single process

- We learned that a container is a process that runs in an isolated environment.
- The first process executed in the isolated PID namespace is pid 1.

The state of the first process run in the container

=

Container’s life

If there are multiple processes running inside a container...

Even if the container is running,
we cannot guarantee that all desired processes are running fine.

The state of all processes running in the container

≠

Container's life

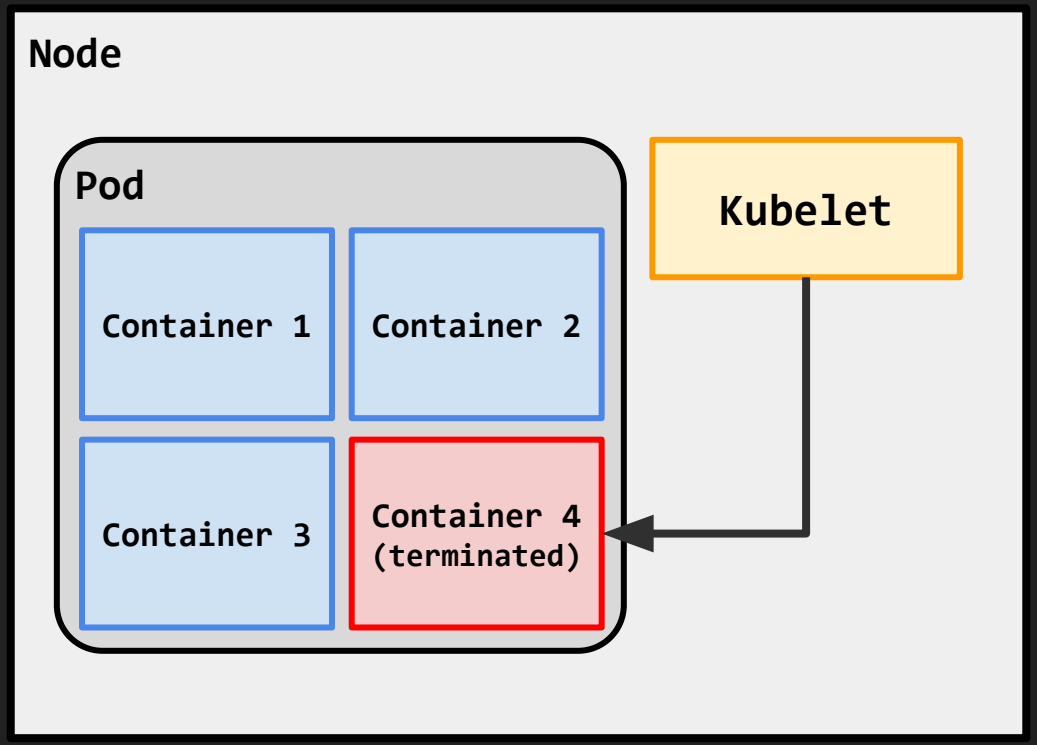
When a specific container of a Kubernetes Pod is terminated

```
spec:
  template:
    (...)
    spec:
      containers:
      - name: hello
        image: busybox
        command: ['sh', '-c', 'sleep 3600']
      restartPolicy: Always
    (...)
```

Don't worry, Kubernetes restarts the container according to the declared restartPolicy.

- Always
- OnFailure
- Never

When a specific container of a Kubernetes Pod is terminated



Retries with exponential back-off delay
(10s, 20s, 40s,⋯ up to 5 minutes)

Considerations for configuring Pods

- Should the containers run on the same node?
  (Containers in the same Pod always run in the same node)
- Should the containers scale horizontally by the same number?
  (Pod are the units of scalability)
- Should the containers be deployed together as a group?

Isolation between containers in a Kubernetes Pod

- A pod is a group of containers.
- If so… what's shared and what's isolated between containers in the same Pod?

Isolation between containers in a Kubernetes Pod

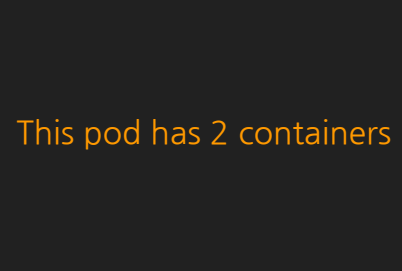
- A pod is a group of containers.
- If so… what's shared and what's isolated between containers in the same Pod?

We learned the fundamentals of containers…
Let's take a look!

Isolation between containers in a Kubernetes Pod

two-containers-pod.yml

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: two-containers-pod
spec:
  template:
    # This is the pod template
    spec:
      containers:
      - name: hello
        image: busybox
        command: ['sh', '-c', 'echo "first container" && sleep 3600']
      - name: hello2
        image: busybox
        command: ['sh', '-c', 'echo "second container" && sleep 3600']
      restartPolicy: OnFailure
    # The pod template ends here
```



Isolation between containers in a Kubernetes Pod

(Kubernetes v1.20.2)

```
vagrant@control-plane:~$ kubectl apply -f two-containers-pod.yml
job.batch/two-containers-pod created
vagrant@control-plane:~$ kubectl get pods -o wide
NAME                       READY   STATUS    RESTARTS   AGE   IP           NODE
two-containers-pod-9pvvt   2/2     Running   0          20s   10.244.2.4   worker-node2
```

you can check the containers of the Pod on this node

Isolation between containers in a Kubernetes Pod

```
vagrant@worker-node2:~$ ps -ef | grep sleep
root        9318    9295  0 20:01 ?        00:00:00 sleep 3600
root        9382    9356  0 20:01 ?        00:00:00 sleep 3600
vagrant    10647   10383  0 20:04 pts/0    00:00:00 grep --color=auto sleep
vagrant@worker-node2:~$ sudo lsns -p 9318
        NS TYPE   NPROCS   PID USER COMMAND
4026531835 cgroup    121     1 root /sbin/init
4026531837 user      121     1 root /sbin/init
4026532214 ipc         3  9182 root /pause
4026532217 net         3  9182 root /pause
4026532304 mnt         1  9318 root sleep 3600
4026532305 uts         1  9318 root sleep 3600
4026532306 pid         1  9318 root sleep 3600
vagrant@worker-node2:~$ sudo lsns -p 9382
        NS TYPE   NPROCS   PID USER COMMAND
4026531835 cgroup    121     1 root /sbin/init
4026531837 user      121     1 root /sbin/init
4026532214 ipc         3  9182 root /pause
4026532217 net         3  9182 root /pause
4026532307 mnt         1  9382 root sleep 3600
4026532308 uts         1  9382 root sleep 3600
4026532309 pid         1  9382 root sleep 3600
```

The cgroup, user namespace are not isolated.

*cgroup namespace

Isolation between containers in a Kubernetes Pod

```
vagrant@worker-node2:~$ ps -ef | grep sleep
root        9318    9295  0 20:01 ?        00:00:00 sleep 3600
root        9382    9356  0 20:01 ?        00:00:00 sleep 3600
vagrant    10647   10383  0 20:04 pts/0    00:00:00 grep --color=auto sleep
vagrant@worker-node2:~$ sudo lsns -p 9318
        NS TYPE   NPROCS   PID USER COMMAND
4026531835 cgroup    121     1 root /sbin/init
4026531837 user      121     1 root /sbin/init
4026532214 ipc         3  9182 root /pause
4026532217 net         3  9182 root /pause
4026532304 mnt         1  9318 root sleep 3600
4026532305 uts         1  9318 root sleep 3600
4026532306 pid         1  9318 root sleep 3600
vagrant@worker-node2:~$ sudo lsns -p 9382
        NS TYPE   NPROCS   PID USER COMMAND
4026531835 cgroup    121     1 root /sbin/init
4026531837 user      121     1 root /sbin/init
4026532214 ipc         3  9182 root /pause
4026532217 net         3  9182 root /pause
4026532307 mnt         1  9382 root sleep 3600
4026532308 uts         1  9382 root sleep 3600
4026532309 pid         1  9382 root sleep 3600
```

The mnt, uts, pid namespace are isolated
(These are not shared even for containers running on the same pod)

Isolation between containers in a Kubernetes Pod

```
vagrant@worker-node2:~$ ps -ef | grep sleep
root        9318    9295  0 20:01 ?        00:00:00 sleep 3600
root        9382    9356  0 20:01 ?        00:00:00 sleep 3600
vagrant    10647   10383  0 20:04 pts/0    00:00:00 grep --color=auto sleep
vagrant@worker-node2:~$ sudo lsns -p 9318
        NS TYPE   NPROCS   PID USER COMMAND
4026531835 cgroup    121     1 root /sbin/init
4026531837 user      121     1 root /sbin/init
4026532214 ipc         3  9182 root /pause
4026532217 net         3  9182 root /pause
4026532304 mnt         1  9318 root sleep 3600
4026532305 uts         1  9318 root sleep 3600
4026532306 pid         1  9318 root sleep 3600
vagrant@worker-node2:~$ sudo lsns -p 9382
        NS TYPE   NPROCS   PID USER COMMAND
4026531835 cgroup    121     1 root /sbin/init
4026531837 user      121     1 root /sbin/init
4026532214 ipc         3  9182 root /pause
4026532217 net         3  9182 root /pause
4026532307 mnt         1  9382 root sleep 3600
4026532308 uts         1  9382 root sleep 3600
4026532309 pid         1  9382 root sleep 3600
```

The ipc and net namespaces are shared between the containers in the pod

→ It's possible to do IPC like using shared memory between containers.

→ The IP addresses and ports are shared between containers. It means that you have to be careful of port conflicts on containers in the same pod.

Pause? What is this?

Pause Container?

```
vagrant@worker-node2:~$ docker ps ¦ grep two-containers
ae2b5c6f7882        busybox                  "sh -c 'echo \"second…"
e8670003df44        busybox                  "sh -c 'echo \"first …"
19802e369987        k8s.gcr.io/pause:3.2     "/pause"
```

The Pause container creates and holds isolated IPC and Network namespaces.

→ The rest of the containers share these namespaces.

This is to prevents issues in shared namespaces in the pod,
even when a container running a user application terminates unexpectedly.

Pause Container?

1. Terminates when SIGINT or SIGTERM is given, without doing anything.

```c
static void sigdown(int signo) {
  psignal(signo, "Shutting down, got signal");
  exit(0);
}

static void sigreap(int signo) {
  while (waitpid(-1, NULL, WNOHANG) > 0)
    ;
}

int main(int argc, char **argv) {
  int i;
  for (i = 1; i < argc; ++i) {
    if (!strcasecmp(argv[i], "-v")) {
      printf("pause.c %s\n", VERSION_STRING(VERSION));
      return 0;
    }
  }

  if (getpid() != 1)
    /* Not an error because pause sees use outside of infra containers. */
    fprintf(stderr, "Warning: pause should be the first process\n");

  if (sigaction(SIGINT, &(struct sigaction){.sa_handler = sigdown}, NULL) < 0)
    return 1;
  if (sigaction(SIGTERM, &(struct sigaction){.sa_handler = sigdown}, NULL) < 0)
    return 2;
  if (sigaction(SIGCHLD, &(struct sigaction){.sa_handler = sigreap,
                                             .sa_flags = SA_NOCLDSTOP},
                NULL) < 0)
    return 3;

  for (;;)
    pause();
  fprintf(stderr, "Error: infinite loop terminated\n");
  return 42;
}
```

Pause Container?

1. Terminates when SIGINT or SIGTERM is given, without doing anything.

2. Plays the role of reaping zombie processes. (if using PID namespace sharing option)

```c
static void sigdown(int signo) {
  psignal(signo, "Shutting down, got signal");
  exit(0);
}

static void sigreap(int signo) {
  while (waitpid(-1, NULL, WNOHANG) > 0)
    ;
}

int main(int argc, char **argv) {
  int i;
  for (i = 1; i < argc; ++i) {
    if (!strcasecmp(argv[i], "-v")) {
      printf("pause.c %s\n", VERSION_STRING(VERSION));
      return 0;
    }
  }

  if (getpid() != 1)
    /* Not an error because pause sees use outside of infra containers. */
    fprintf(stderr, "Warning: pause should be the first process\n");

  if (sigaction(SIGINT, &(struct sigaction){.sa_handler = sigdown}, NULL) < 0)
    return 1;
  if (sigaction(SIGTERM, &(struct sigaction){.sa_handler = sigdown}, NULL) < 0)
    return 2;
  if (sigaction(SIGCHLD, &(struct sigaction){.sa_handler = sigreap,
                                             .sa_flags = SA_NOCLDSTOP},
                NULL) < 0)
    return 3;

  for (;;)
    pause();
  fprintf(stderr, "Error: infinite loop terminated\n");
  return 42;
}
```

PID namespace sharing on Kubernetes

If there is a risk of Zombie process occurring in some containers,

You can delegate the Zombie process reaping role to the Pause container by activating the Kubernetes “PID namespace sharing” option.

Reference link: https://www.ianlewis.org/en/almighty-pause-container

- PID namespace sharing is enabled by default in Kubernetes v1.7.
- However, from v1.8, it is disabled again due to compatibility issues with containers that depend on the init system.
    - https://github.com/kubernetes/kubernetes/issues/48937

PID namespace sharing on Kubernetes

two-containers-pod.yml

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: two-containers-pod
spec:
  template:
    # This is the pod template
    spec:
      shareProcessNamespace: true
      containers:
      - name: hello
        image: busybox
        command: ['sh', '-c', 'echo "first container" && sleep 3600']
      - name: hello2
        image: busybox
        command: ['sh', '-c', 'echo "second container" && sleep 3600']
      restartPolicy: OnFailure
    # The pod template ends here
```

PID namespace sharing on Kubernetes

```
vagrant@worker-node1:~$ ps -ef | grep sleep
root        1009     983  0 21:01 ?        00:00:00 sleep 3600
root        1083    1050  0 21:01 ?        00:00:00 sleep 3600
vagrant     1378    1297  0 21:01 pts/0    00:00:00 grep --color=auto sleep
vagrant@worker-node1:~$ sudo lsns -p 1009
        NS TYPE   NPROCS   PID USER COMMAND
4026531835 cgroup    117     1 root /sbin/init
4026531837 user      117     1 root /sbin/init
4026532214 ipc         3   877 root /pause
4026532215 pid         3   877 root /pause
4026532217 net         3   877 root /pause
4026532304 mnt         1  1009 root sleep 3600
4026532305 uts         1  1009 root sleep 3600
```

pause container's PID namespace

```
vagrant@worker-node1:~$ docker exec e077f522d6fa ps
PID   USER     TIME  COMMAND
    1 root      0:00 /pause
    6 root      0:00 sleep 3600
   11 root      0:00 sleep 3600
   22 root      0:00 ps
```

In the container running "sleep" process,
you can see the other container processes running on the same pod.

PID 1 is always "pause" process when enabling PID namespace sharing.

Creating a pod without Kubernetes (Exercise)

```
# Docker version 19.03.15
# Run a `pause` container
$ docker run -d --ipc="shareable" --name pause k8s.gcr.io/pause:3.2

# Run a container executing `sleep` command on the ipc, net, pid namespaces of pause container
$ docker run -ti --rm -d --name sleep-busybox \
    --net=container:pause \
    --ipc=container:pause \
    --pid=container:pause \
    busybox sleep 3600

# Run a container executing `ps` command on the ipc, net, pid namespaces of pause container.
# We can see the other container's processes(pause, sleep) because the pid namespace is shared.
$ docker run --rm --name ps-busybox \
     --net=container:pause \
     --ipc=container:pause \
     --pid=container:pause \
     busybox ps

# Let's take a look the namespaces of "sleep-busybox" container
$ ps -ef | grep sleep
$ sudo lsns -p <PID>
```

Comparing with Kubernetes Pod (Exercise)

practice.yml

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: practice
spec:
  template:
    # This is the pod template
    spec:
      shareProcessNamespace: true
      containers:
      - name: sleep
        image: busybox
        command: ['sh', '-c', 'sleep 3600']
      restartPolicy: OnFailure
    # The pod template ends here
```

```
$ kubectl apply -f practice.yml

# Determine which node the pod is running on
$ kubectl get pods -o wide

# In the node,
# let's take a look at the `sleep` container's namespaces
node# $ ps -ef | grep sleep
node# $ sudo lsns -p <PID>
```

Recap: Kubernetes Pod Concept

- What is a pod?
  - The smallest unit that can be deployed in Kubernetes
    - Pods are managed by various types of resources (Job, ReplicaSet, etc.)
- A group of one or more containers
  - running a single container
  - running multiple containers
    - Primary Container
    - Sidecar Containers

Recap: Kubernetes Pod Concept

- It's not recommended to run multiple processes in one container
  - Even if the container is running,
    we cannot guarantee that all processes are running well.
- When a container in a Kubernetes Pod is shut down,
  Kubelet restarts the container according to the restartPolicy.
- Considerations to configure pod
  - Should the containers run on the same node?
  - Should the containers scale horizontally by the same number?
  - Should the containers be deployed together as a group?

Recap: Kubernetes Pod Concept

- Isolation between containers in Kubernetes Pods
  - shared namespaces with Host → **cgroup, user**
  - shared namespaces with the containers in the same pod → **ipc, net**
  - Isolated namespaces for each container → **mount, uts, pid**
    - pid namespace sharing is optional
- The pause container?
  - Creates and holds isolated IPC and Network namespace.
  - Plays the role of reaping zombie processes when enabling PID namespace sharing.

Thanks

Special thanks to June Oh for the linguistic review.