Podman Special Event ～ OpenShift Lounge+ TALKs ～ (Nov 16, 2023)

NTT

Kubernetes in Rootless Podman

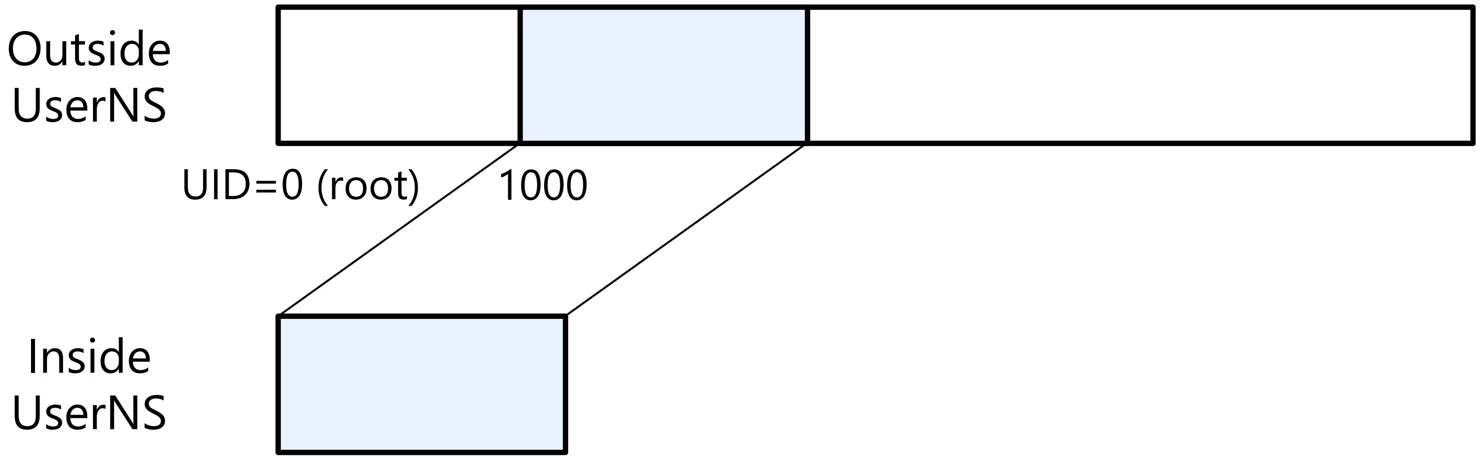
Akihiro Suda, NTT

Rootless containers

- A technique to run container runtimes as a non-root user
- Available for LXC, Docker, Podman, containerd, etc.
- Mitigates potential vulnerabilities of container runtimes
  - Even if it gets compromised, it will not affect files and processes owned by other user IDs
  - Less chance of having stealth malware, as the kernel, firmware, etc., are protected
  - No ARP spoofing/DNS spoofing in the physical network
    https://blog.aquasec.com/dns-spoofing-kubernetes-clusters

Rootless containers

- Implemented by using User Namespaces
  - A feature of the Linux kernel
  - Maps the root in the UserNS to a non-root user outside the UserNS
  - dnf, apt-get, etc. just work, because they think they are running as the root



Rootless Kubernetes

- Began in 2018 https://twitter.com/_AkihiroSuda_/status/1019570064385642498
  - As old as Rootless Docker (pre-release at that time) and Rootless Podman
- The changes to Kubernetes was merged in Kubernetes v1.22 (Aug 2021)
  - Feature gate: "KubeletInUserNamespace" (Alpha)

KubeletInUserNamespace feature gate

- Slightly misnomer; it refers to running all the node components (kubelet, kube-proxy, CRI, CNI, OCI) in UserNS
- Root-in-UserNS is similar to the root, but has no permission for:
  - some sysctls
  - dmesg
- The feature gate allows ignoring these permission errors

  https://github.com/search?q=repo%3Akubernetes%2Fkubernetes%20KubeletInUserNamespace&type=code

How to run Rootless Kubernetes

The easiest way to run Rootless Kubernetes today is to wrap a Kubernetes node in a Rootless container (such as Rootless Podman)

- kind
- minikube
- Usernetes (Gen2)

kind (Kubernetes in Docker)

- https://kind.sigs.k8s.io/
- The most typical way to run Kubernetes in Docker (and in Podman)
- Supports multi-node, but only on a single host
  - 1 kind container = 1 Kubernetes node
- Not intended to be used for production environments

kind (Kubernetes in Docker): Usage

- A few of steps needs to be executed by the root
  - These steps are needed for minikube, Usernetes, etc. too

Needs cgroup v2
(RHEL >= 9, etc.)

```
# Allow limiting CPU, memory, etc. via cgroups
cat <<EOF | sudo tee \
/etc/systemd/system/user@.service.d/delegate.conf
[Service]
Delegate=cpu cpuset io memory pids
EOF
sudo systemctl daemon-reload
```

kind (Kubernetes in Docker): Usage

- A few of steps needs to be executed by the root
  - These steps are needed for minikube, Usernetes, etc. too

```
# Load extra kernel modules
cat <<EOF | sudo tee /etc/modules-load.d/iptables.conf
ip6_tables
ip6table_nat
ip_tables
iptable_nat
EOF
systemctl restart systemd-modules-load.service
```

kind (Kubernetes in Docker): Usage

- https://kind.sigs.k8s.io/docs/user/rootless/

```
export KIND_EXPERIMENTAL_PROVIDER=podman

kind create cluster

kubectl get pods -A
```

minikube

- https://minikube.sigs.k8s.io/

- Originally designed for running Kubernetes in VM

- Supports kind-like mode too

minikube: Usage

- https://minikube.sigs.k8s.io/docs/drivers/podman/

```
minikube config set rootless true

minikube start --driver=podman --container-runtime=crio

kubectl get pods -A
```

- Make sure to set “rootless” property, otherwise minikube executes podman with sudo

Usernetes

- https://github.com/rootless-containers/usernetes
- Rootless Kubernetes, since 2018
  - **Gen 1** (2018-2023): “The hard way”
  - **Gen 2** (2023-): depends on Rootless (Docker|Podman|nerdctl) for simplicity
- Supports real multi-node clusters with VXLAN

Usernetes: Gen 1 vs Gen 2

NTT

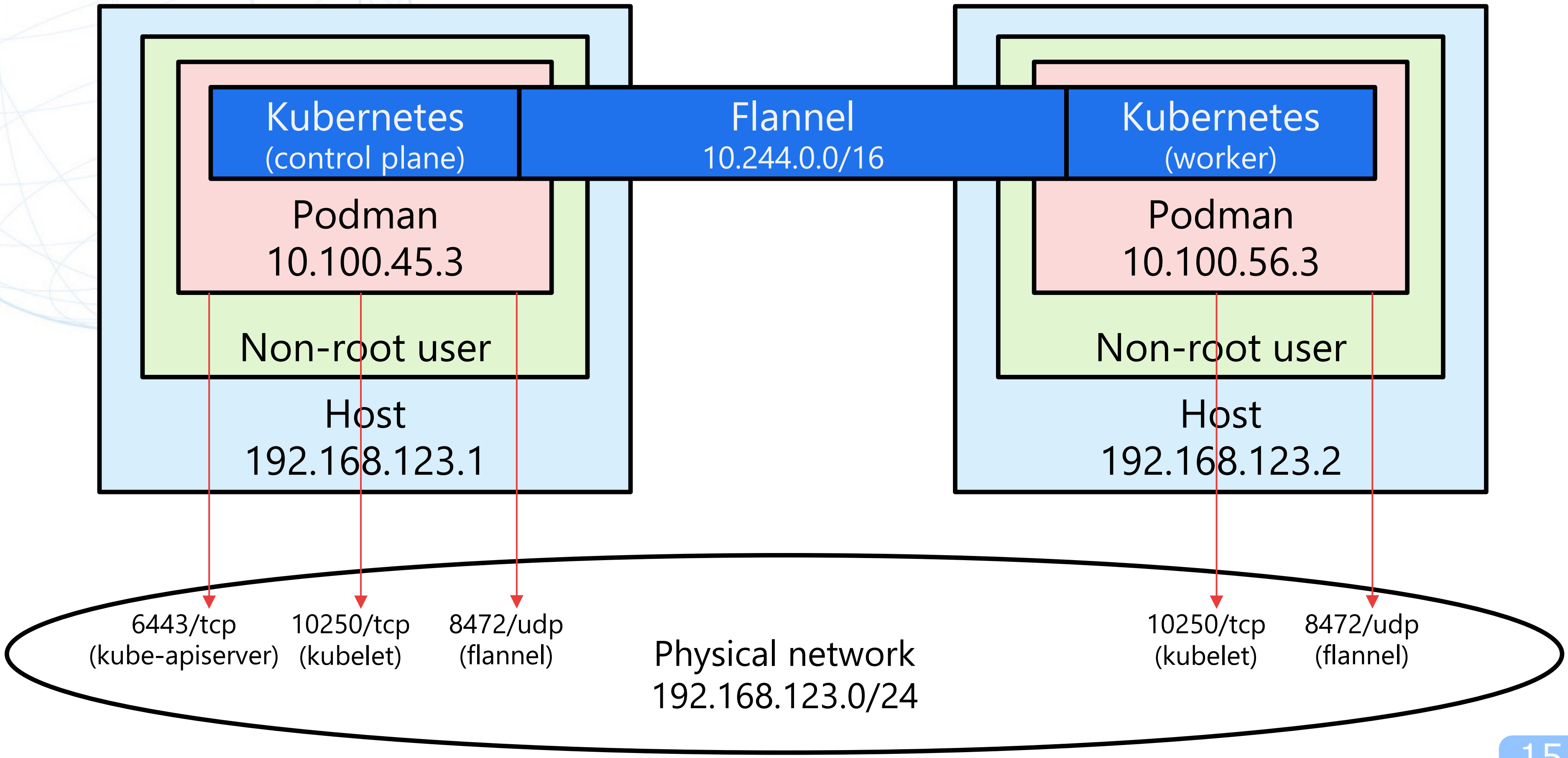
"The hard way"

Similar to `kind` and minikube, but supports real multi-node

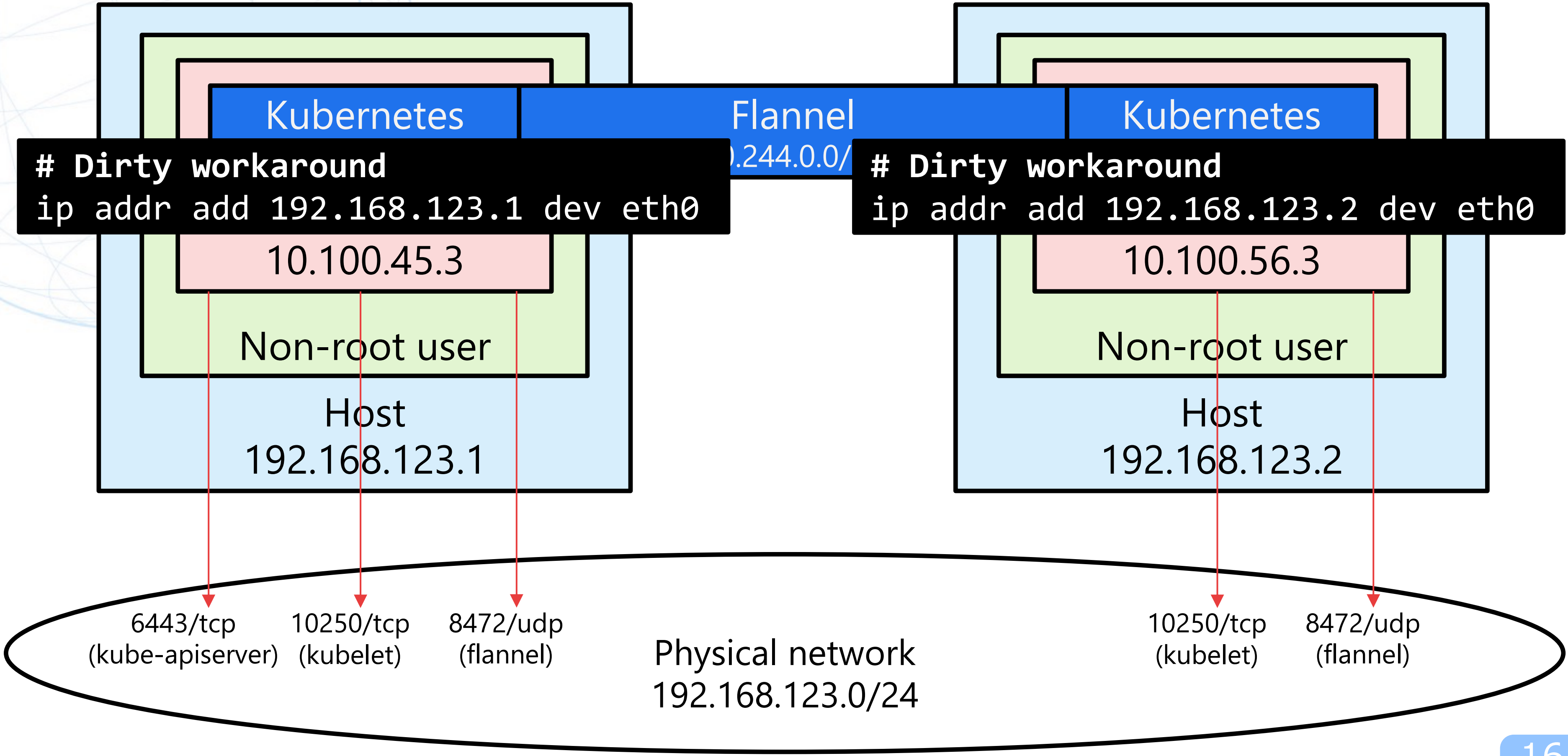
	Gen 1 (2018-2023)	Gen 2 (2023-)
Host dependency	RootlessKit	Rootless Docker, Rootless Podman, or Rootless nerdctl (contaiNERD CTL)
Supports kubeadm	No	Yes
Supports multi-node (multi-host)	Yes, but practically No, due to complexity	Yes
Supports hostPath volumes	Yes	Yes, for most paths, but needs an extra config

Usernetes (Gen 2): How it works

NTT



Usernetes (Gen 2): How it works



```
# Dirty workaround
ip addr add 192.168.123.1 dev eth0
```

```
# Dirty workaround
ip addr add 192.168.123.2 dev eth0
```

Usernetes (Gen 2): Usage

Set `CONTAINER_ENGINE=podman` if multiple container engines are installed on the host

```
# Bootstrap the first node
make up
make kubeadm-init
make install-flannel
```

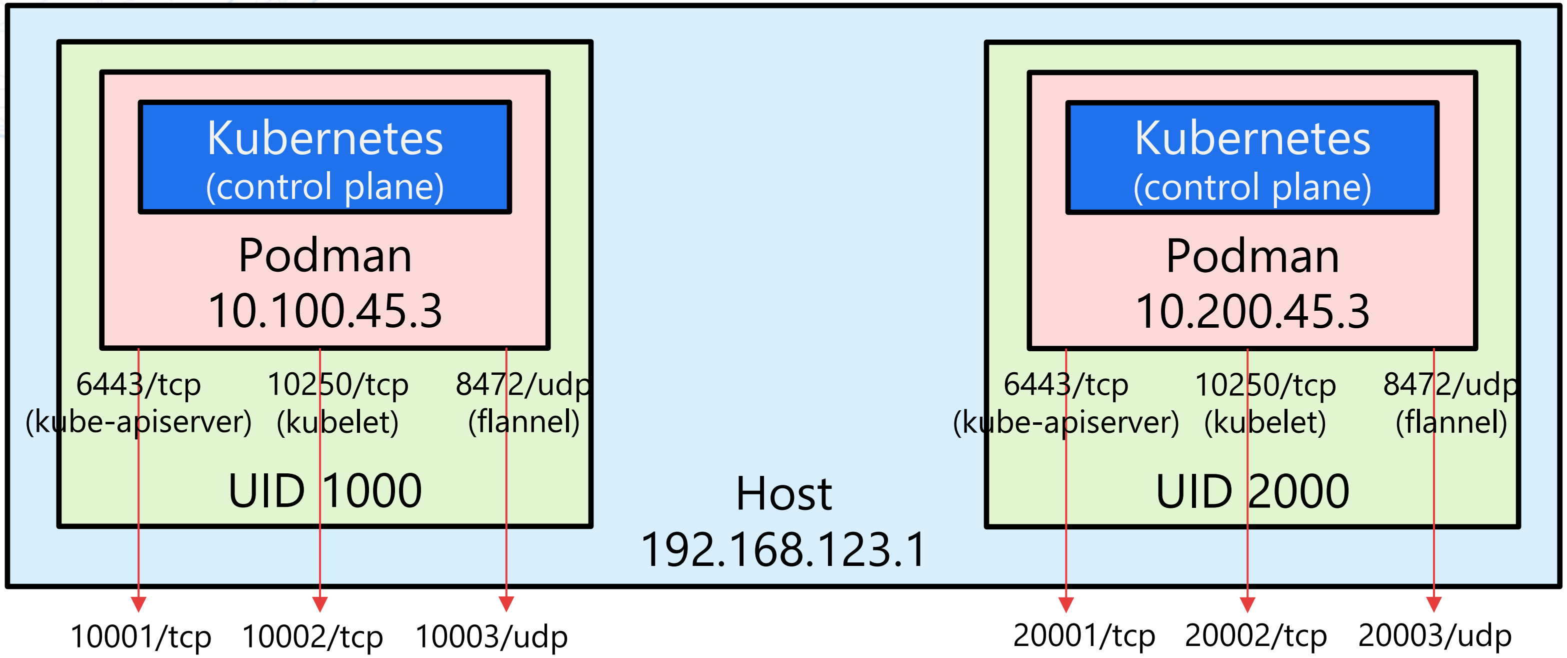
```
# Enable kubectl
make kubeconfig
export KUBECONFIG=$(pwd)/kubeconfig
kubectl get pods -A
```

```
# Multi-node
make join-command
scp join-command another-host:~/usernetes
ssh another-host make -C ~/usernetes up kubeadm-join
```

Future works

Multi-tenancy using multiple user IDs and multiple TCP ports

- A single host will be able to join multiple clusters



Future works

Promote “KubeletInUserNamespace” gate from alpha to beta (and then GA)

- The blocker was how to test the gate in the upstream CI
- WIP: https://github.com/kubernetes/test-infra/pull/31085
  - Spawns rootless `kind` machines using Google Compute Engine

Future works

Eliminate the overhead of user-mode TCP/IP

(slirp4netns, RootlessKit, and pasta)

- POC: https://github.com/rootless-containers/bypass4netns
- Captures socket-related syscalls in containers using seccomp_unotify(2), and replaces the socket FDs with ones that are created in the host network namespace
- Unsolved question: how to support VXLAN?
  VXLAN is implemented in the kernel, so VXLAN calls cannot be captured with seccomp_unotify(2)

Future works

Support running okd (OpenShift) in Rootless Podman

- Help wanted from the OpenShift community