KUBERNETES – SECURITY

Github repo..

https://github.com/vsaini44/KubernetesRepo.git

Overview

In this Module, you'll learn how to allow pods to access the resources of the node they're running on. You'll also learn how to configure the cluster so users aren't able to do whatever they want with their pods.

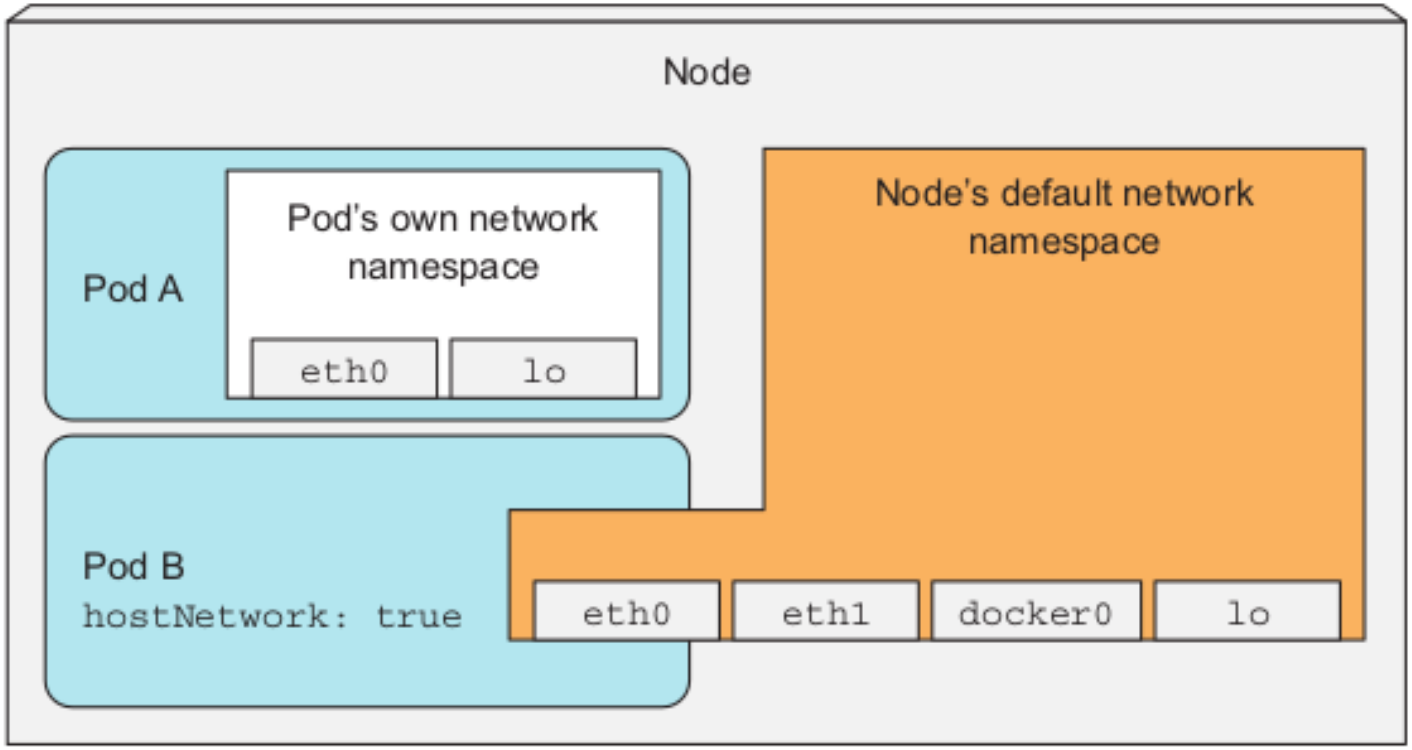
Then, we will learn how to secure the network the pods use to communicate.

Host node namespace

Containers in a pod usually run under separate Linux namespaces, which isolate their processes from processes running in other containers or in the node's default namespaces.

Certain pods (usually system pods) need to operate in the host's default namespaces, allowing them to see and manipulate node-level resources and devices

Host node’s namespace

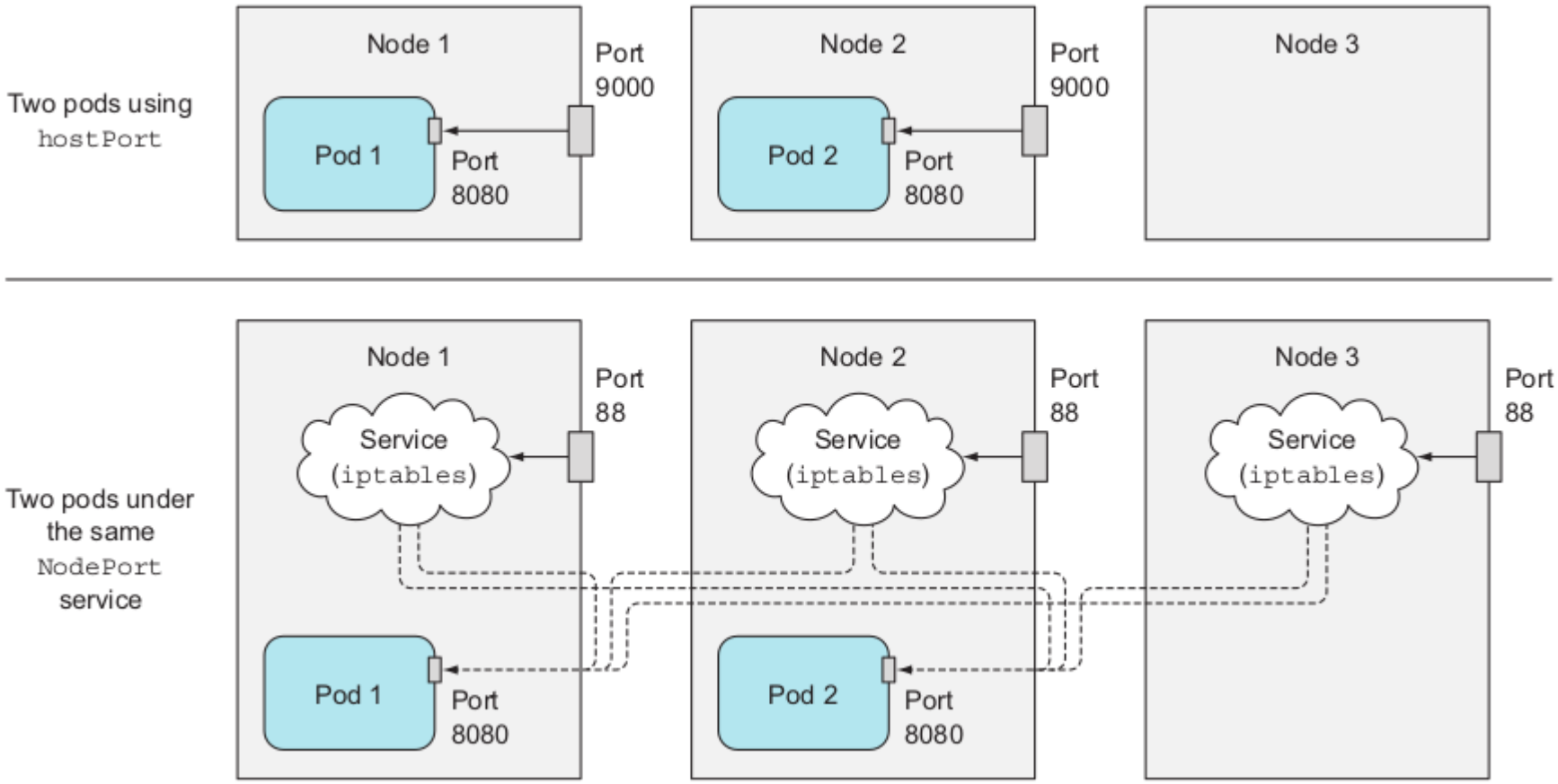


Binding host port

A related feature allows pods to bind to a port in the node's default namespace, but still have their own network namespace.

This is done by using the hostPort property in one of the container's ports defined in the spec.containers.ports field.

HostPort v/s Nodeport



Security Context

runAsUser : To run a pod under a different user ID than the one that's baked into the container image

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-as-user-guest
spec:
  containers:
  - name: main
    image: alpine
    command: ["/bin/sleep", "999999"]
    securityContext:
      runAsUser: 405
```

Security Context

runasNonRoot : Although containers are mostly isolated from the host system, running their processes as root is still considered a bad practice

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-run-as-non-root
spec:
  containers:
  - name: main
    image: alpine
    command: ["/bin/sleep", "999999"]
    securityContext:
      runAsNonRoot: true
```

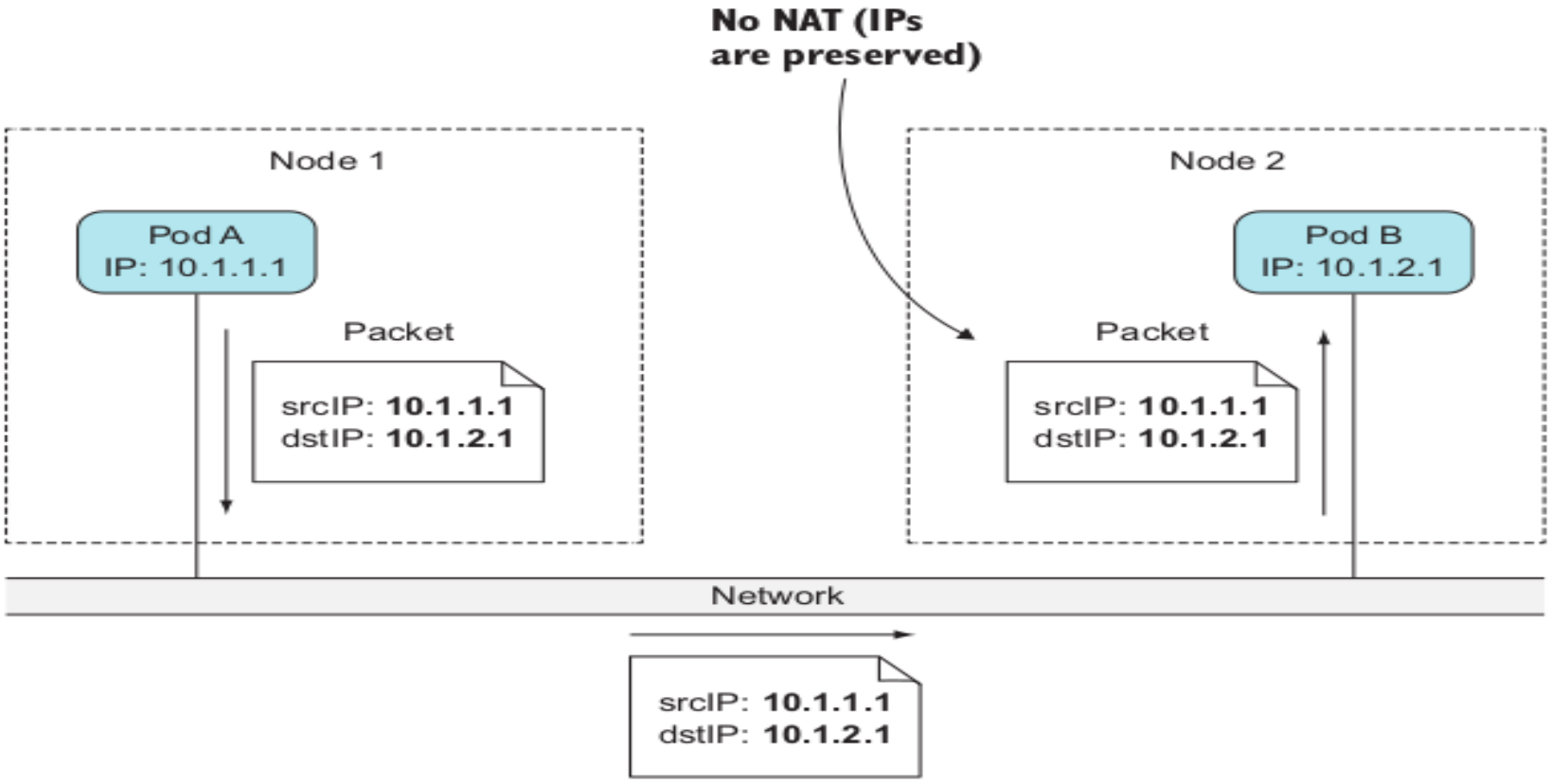
Security Context

privileged : To get full access to the node's kernel, the pod's container runs in privileged mode.

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-privileged
spec:
  containers:
  - name: main
    image: alpine
    command: ["/bin/sleep", "999999"]
    securityContext:
      privileged: true
```

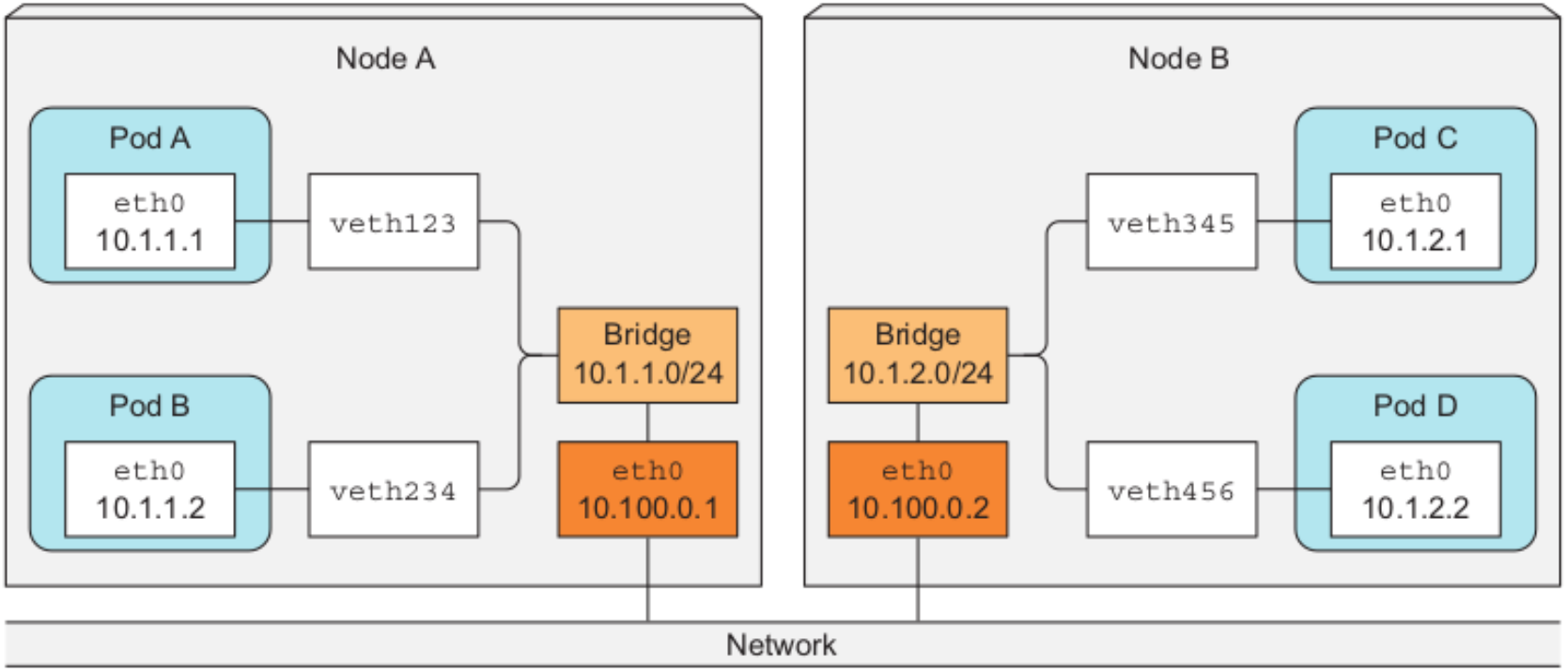
Understanding Network

This is what kubernetes wants ..



Understanding Network

This is what we have



Container Network Interface

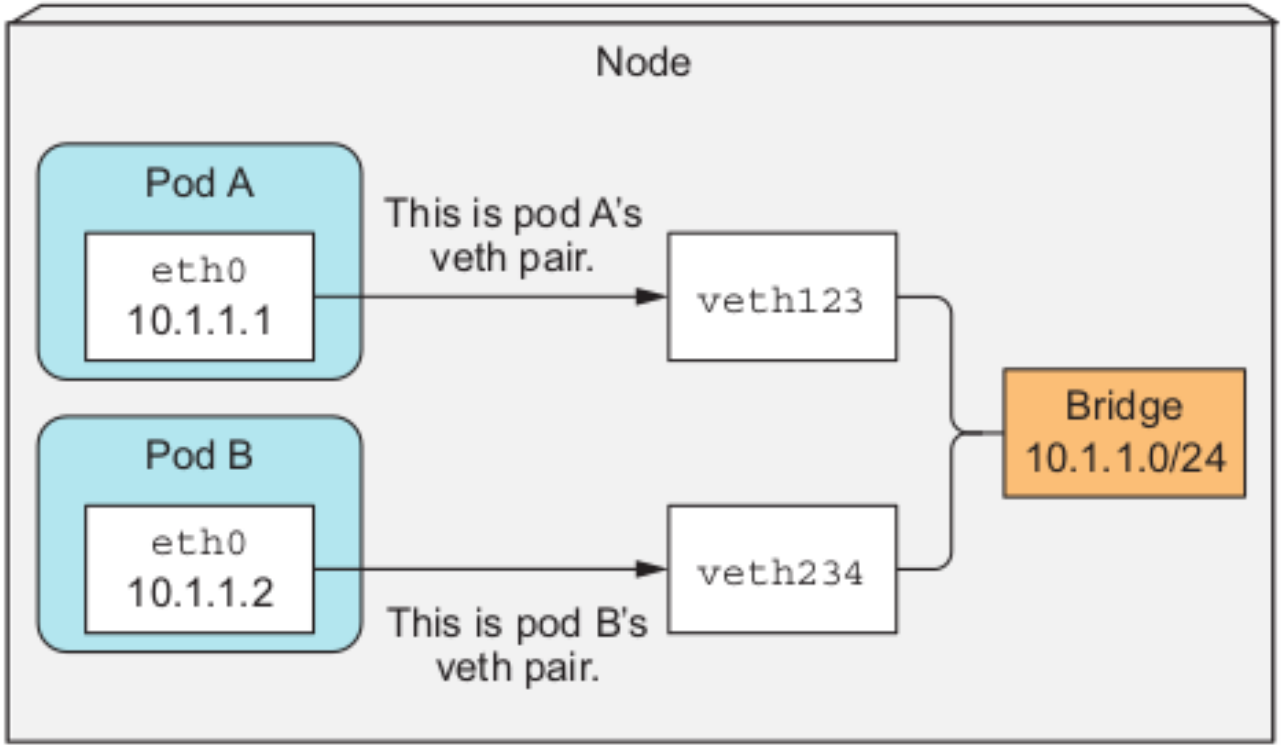
CNI Project plugins allow makes the nodes appear as though they’re connected to the same network switch, regardless of the actual underlying network topology, no matter how complex it is.

Packets sent from the pod are encapsulated and sent over the network to the node running the other pod, where they are de-encapsulated and delivered to the pod in their original form.

Plugins

- Calico
- Flannel
- Romana
- Weave Net
- And others ..

Understanding networking



The pause container

A piece of infrastructure that enables many networking features in Kubernetes is known as the pause container.

This container runs alongside the containers defined in a Pod and is responsible for providing the network namespace that the other containers share

The flannel CNI

Flannel is one of the most straightforward network providers for Kubernetes.

It operates at Layer 3 and offloads the actual packet forwarding to a backend such as VxLAN or IPSec. It assigns a large network to all hosts in the cluster and then assigns a portion of that network to each host.

Routing between containers on a host happens via the usual channels, and Flannel handles routing between hosts using one of its available options.

The Calico CNI

Calico operates at Layer 3 and assigns every workload a routable IP address.

It prefers to operate by using BGP without an overlay network for the highest speed and efficiency, but in scenarios where hosts cannot directly communicate with one another, it can utilize an overlay solution such as VxLAN or IP-in-IP.

The network Policy

By default, pods in a given namespace can be accessed by anyone.

Network policy offers us the solution of network isolation in a namespace.