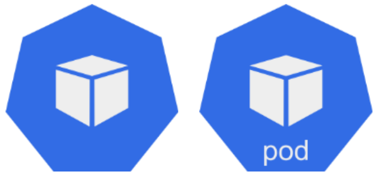
Kaloom™ Copyright 2021

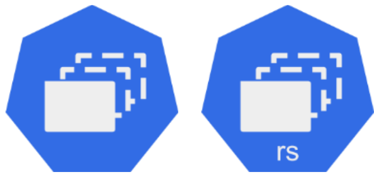
Kubernetes Networking Semantics

Per Andersson

Pod and Workloads



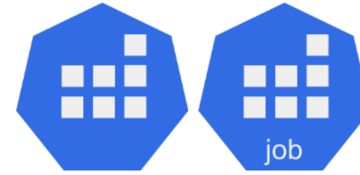
Pod: Pod is a collection of containers that can run on a host. This resource is created by clients and scheduled onto hosts.



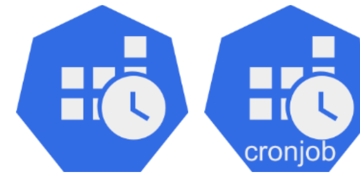
ReplicaSet: ReplicaSet ensures that a specified number of pod replicas are running at any given time.

Deployment: Deployment enables declarative updates for Pods and ReplicaSets.

DaemonSet: DaemonSet represents the configuration of a daemon set.



Job: Job represents the configuration of a single job.



CronJob: A CronJob manages time based Job, namely:

- once at a specified point in time
- repeatedly at a specified point in time

StatefulSet: StatefulSet represents a set of pods with consistent identities. Identities are defined as: network, storage.

Kaloom™ Copyright 2021

kaloom

Network Objects

Ingress: Ingress is a collection of rules that allow inbound connections to reach the endpoints defined by a backend. An Ingress can be configured to give services externally-reachable urls, load balance traffic, terminate SSL, offer name based virtual hosting etc.

Service: Service is a named abstraction of software service (for example, mysql) consisting of local port (for example 3306) that the proxy listens on, and the selector that determines which pods will answer requests sent through the proxy.

EndpointSlices: Endpoints and Endpointslices are a collections of endpoints that implement the actual service.

Network Policy: NetworkPolicy describes what network traffic is allowed for a set of Pods.

Kaloom™ Copyright 2021

‹kaloom›

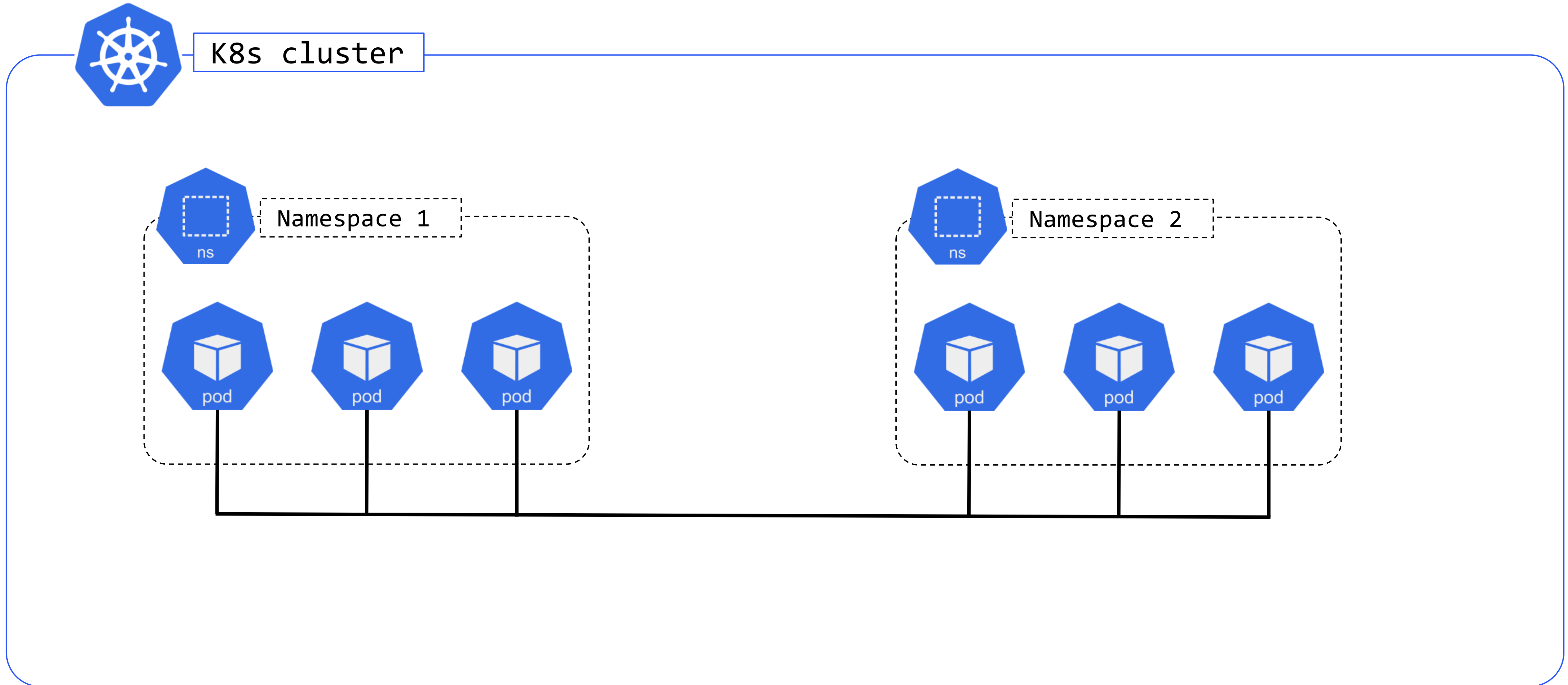
Abstraction of K8s networking

- The manifest used to define Kubernetes entities are typically free of any sort of IP address information.
  - Service
  - Network Policy
  - Ingress
  - Pod
  - Workload Resources
    - Deployment
    - ReplicaSet
    - StatefulSet
    - DaemonSet
    - Job
    - CronJob
- The basic semantics of Kubernetes and the information found in the manifest defines the connectivity rules and behavior

- All entities belong to a namespace
- All entities have a name that is unique in that namespace
- All entities have a unique identifier (UID)
- The identity can be simplified to type<namespace, name>
  - namespace<name>
  - service<namespace, name>
  - networkPolicy<namespace, name>
  - ingress<namespace, name>
  - pod<namespace, name>
  - deployment<namespace, name>
  - replicaSet<namespace, name>
  - daemonSet<namespace, name>
  - job<namespace, name>
  - cronJob<namespace, name>

Kaloom™ Copyright 2021

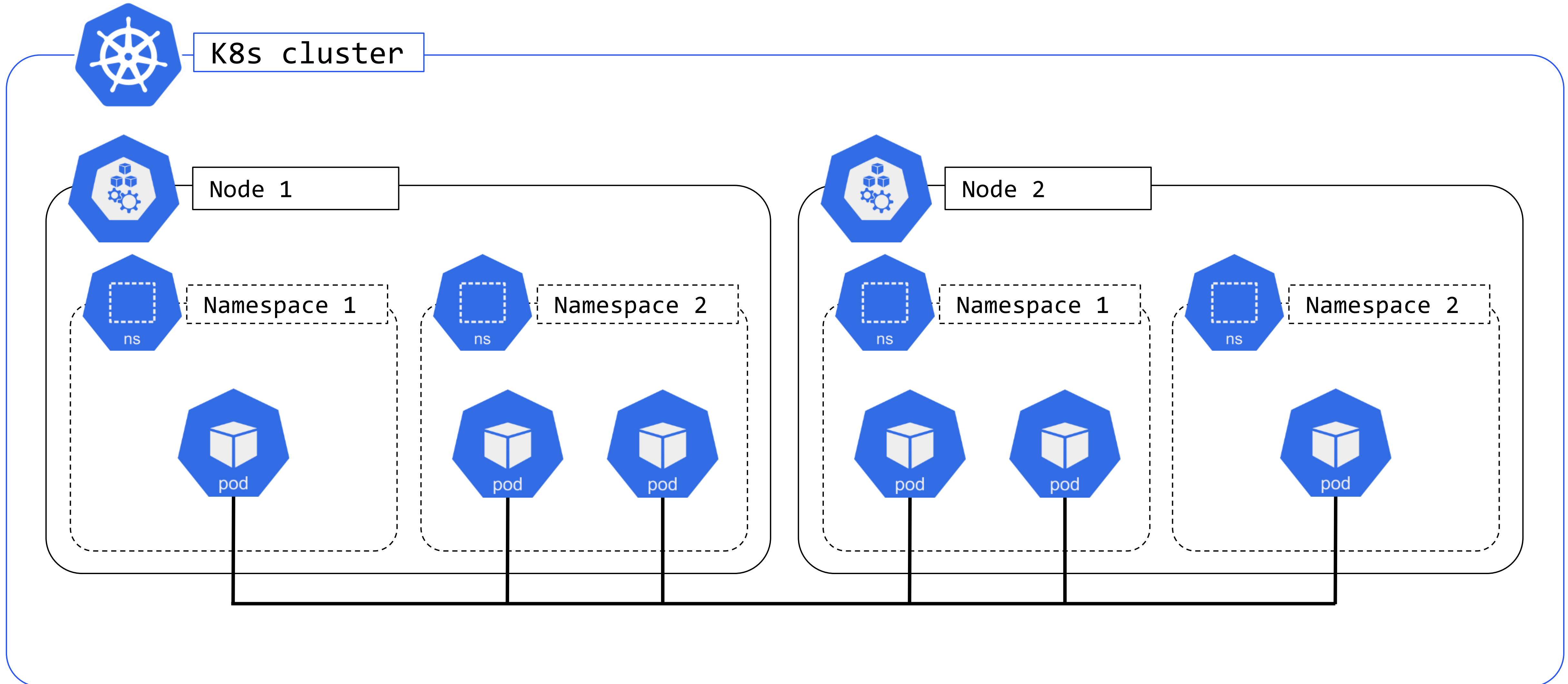
All pods can communicate with each other*



Kaloom™ Copyright 2021

kaloom

All pods can communicate with each other*



Kaloom™ Copyright 2021

kaloom

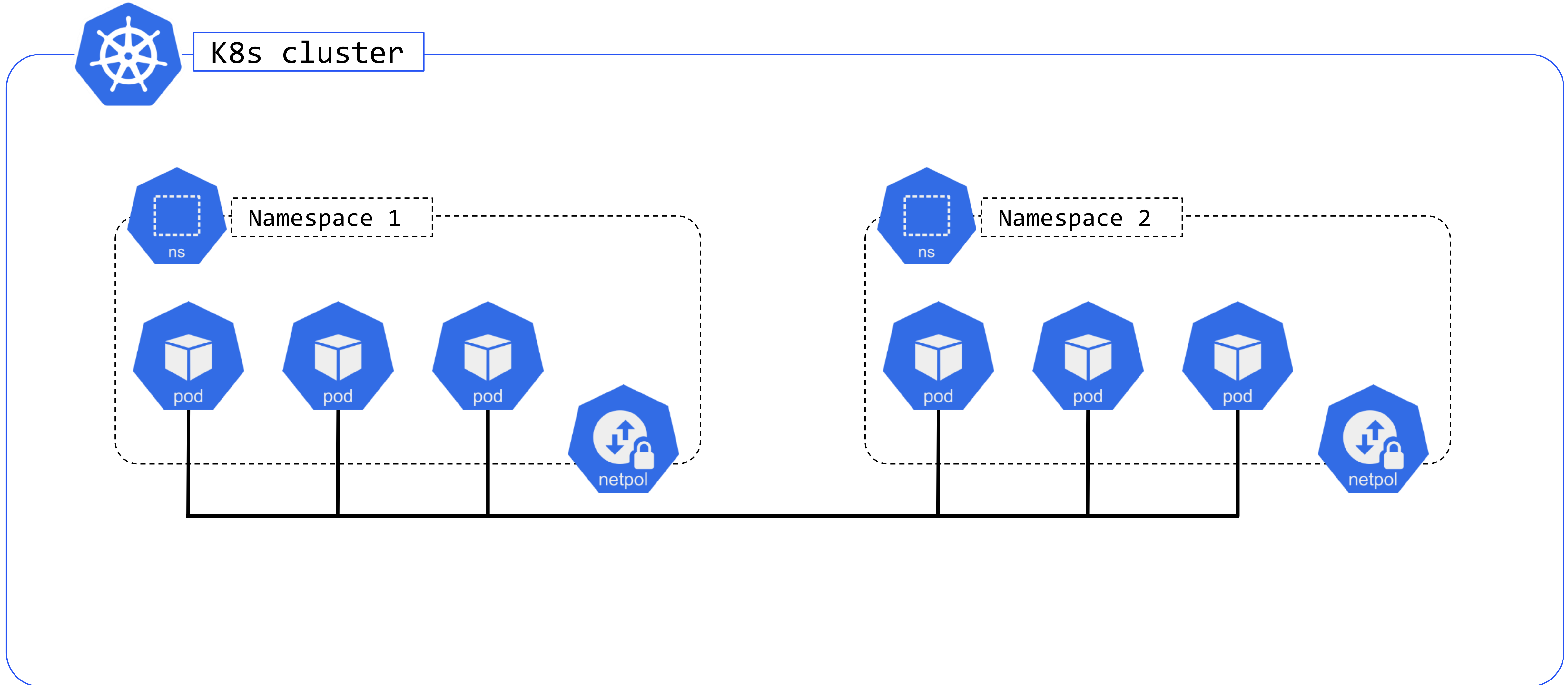
Pod lifecycle and Pod Communication

- The Pod life cycle can be abstracted to
  - addPod(pod<namespace, name>)
  - removePod(pod<namespace, name>)
- The Pod communication can be abstracted to two connection primitives
  - openPodConnection(sourcePod, destinationPod, protocol, port) =>
    <true, connectionId> or <false, 0>
  - closePodConnection(connectionId)
- A connection can be described as
  - newConnection(Pod1, Pod2, protocol, port) =>
    connection<connectionId, Pod1, Pod2>
  - deleteConnection(connectionId)

Kaloom™ Copyright 2021

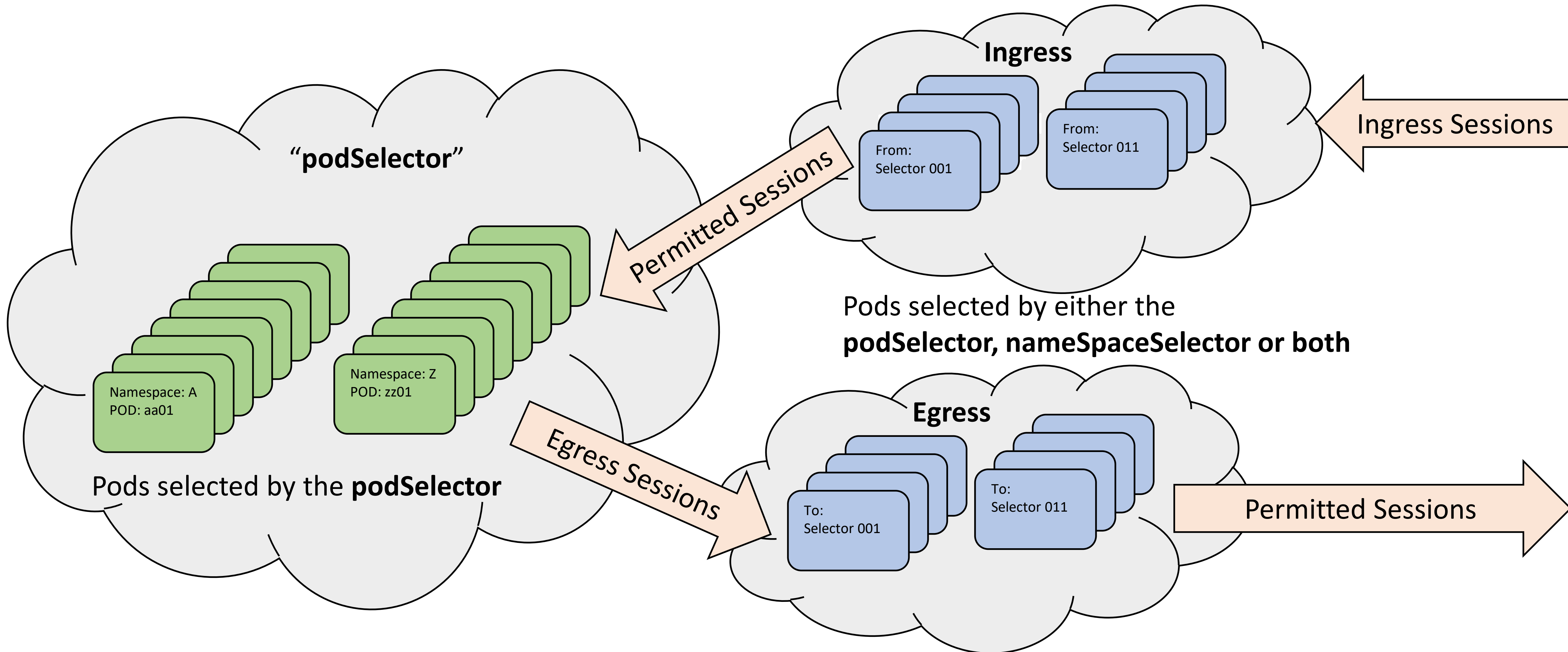
kaloom

All pods can communicate with each other, if there is Network Policy that allows it



Kaloom™ Copyright 2021

Network Policy model



Kaloom™ Copyright 2021

kaloom

Network Policy examples

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
    name: default-deny-ingress
spec:
    podSelector: {}
    policyTypes:
    - Ingress
```

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
    name: default-deny-egress
spec:
    podSelector: {}
    policyTypes:
    - Egress
```

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
    name: default-deny-all
spec:
    podSelector: {}
    policyTypes:
    - Ingress
    - Egress
```

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
    name: allow-all-ingress
spec:
    podSelector: {}
    ingress:
    - {}
    policyTypes:
    - Ingress
```

```yaml
apiVersion: networking.k8s.io/v1 - Ingress
kind: NetworkPolicy
metadata:
    name: allow-all-egress
spec:
    podSelector: {}
    egress:
    - {}
    policyTypes:
    - Egress
```

```yaml
apiVersion: networking.k8s.io/v1 - Ingress
kind: NetworkPolicy
metadata:
    name: allow-all
spec:
    podSelector: {}
     ingress:
    - {}
    egress:
    - {}
    policyTypes:
    - Ingress
    - Egress
```

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
    name: web-deny-all
spec:
    podSelector:
        matchLabels:
            app: web
    ingress: []
    policyTypes:
    - Ingress
    - Egress
```

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
    name: web-allow-all-ns-monitoring
    namespace: default
spec:
    podSelector:
        matchLabels:
            app: web
    ingress:
    - from:
    # chooses all pods in namespaces
    # labelled with team=operations

        - namespaceSelector:
            matchLabels:
                team: operations
    # chooses pods with type=monitoring
          podSelector:
            matchLabels:
                team: monitoring
```

Kaloom™ Copyright 2021

kaloom™

Network Policy

- The NetworkPolicy spec has all the information needed to define a particular network policy in the given namespace.
  - **podSelector**: Each NetworkPolicy includes a podSelector which selects the grouping of pods to which the policy applies. An empty podSelector , ”podSelector: {} selects all pods in the namespace.
  - **policyTypes**: Each NetworkPolicy includes a policyTypes list which may include either Ingress, Egress, or both. The **policyTypes** field indicates whether or not the given policy applies to ingress traffic to selected pod, egress traffic from selected pods, or both. If no **policyTypes** are specified on a NetworkPolicy then by default Ingress will always be set and Egress will be set if the NetworkPolicy has any egress rules.
  - **ingress**: Each NetworkPolicy may include a list of whitelist ingress rules. Each rule allows traffic which matches both the **from** and **ports** sections.
  - **egress**: Each NetworkPolicy may include a list of whitelist egress rules. Each rule allows traffic which matches both the **to** and **ports** sections.
- **to** and **from** selectors, there are four kinds of selectors that can be specified in an **ingress** from section or **egress** to section:
  - **podSelector**: This selects particular Pods in the same namespace as the NetworkPolicy which should be allowed as ingress sources or egress destinations.
  - **namespaceSelector**: This selects particular namespaces for which all Pods should be allowed as ingress sources or egress destinations.
  - **namespaceSelector** *and* **podSelector**: A single **to/from** entry that specifies both **namespaceSelector** and **podSelector** selects particular Pods within particular namespaces.
  - **ipBlock**: This selects particular IP CIDR ranges to allow as ingress sources or egress destinations. These should be cluster-external IPs, since Pod IPs are ephemeral and unpredictable.

Kaloom™ Copyright 2021

kaloom

Abstract Network Policy Filter System

- addNetworkPolicy(networkPolicy<namespace, name , policy)
- updateNetworkPolicy(networkPolicy<namespace, name , policy)
- removeNetworkPolicy(networkPolicy<namespace, name)
- allowNewConnectionFromPod(destinationPod, sourcePod, protocol, destinationPort)
- allowNewConnectionToPod(sourcePod, destinationPod, protocol, destinationPort)

- The ipBlock part of to and from selectors is ignored for now
- This policy filter system is not dependent on
  - The number of pod replicas
  - The number of pod interfaces
  - The number of pod network attachments
  - Which interface an ip address is configured too
- The policy filter system is dependent on
  - Pod manifests
  - Pod labels
  - Label selectors in the Network Policies
- It is only updated when
  - policy filters are added, updated or removed
  - Labels used in policy filters are changed
- It is easy to extend with functionality
  - show how policies are related
  - which policies that applies towards a
    - namespace
    - Service
    - Workload entities
    - individual pods

Kaloom™ Copyright 2021

kaloom

Pod Communication with Network Policies

- The allowNewConnectionToPod is used to check outgoing egress connectivity from the "sourcePod"
  - This is done by matching the "sourcePod" towards the NetworkPolicy PodSelector and the "destinationPod", "protocol" and "port" towards the NetworkPolicy egress to rules
- The allowNewConnectionToPod is used to check incomming ingress connectivity towards the "destinationPod"
  - This is done by matching the "destinationPod" towards the NetworkPolicy PodSelector and the "sourcePod", "protocol" and "port" towards the NetworkPolicy ingress from rules

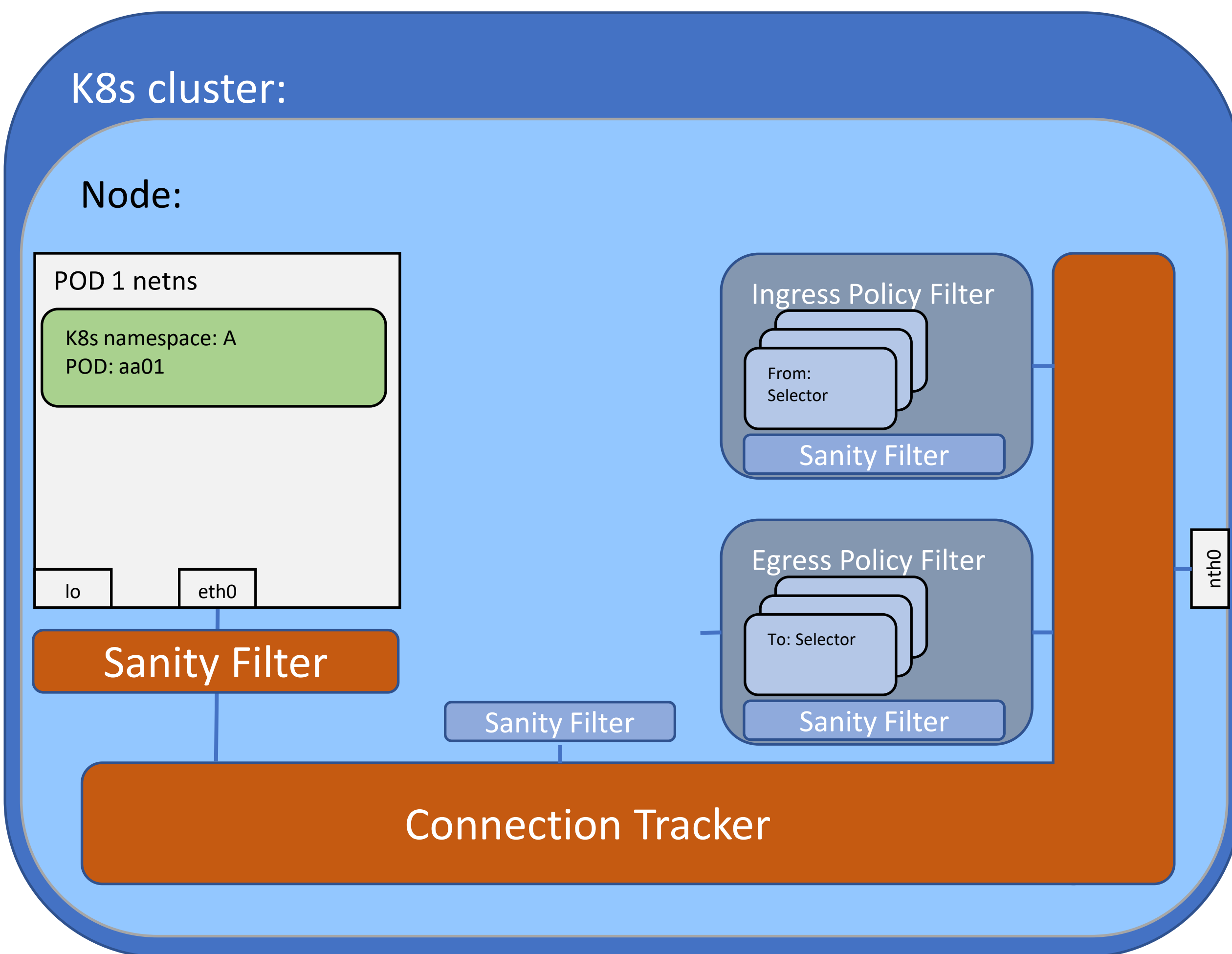
- openPodConnection must be updated to support the NW policy check

```
openPodConnection(sourcePod, destinationPod, protocol, port)
{
    if
            allowNewConnectionToPod(
                    sourcePod,
                    destinationPod,
                    protocol,
                    destinationPort)
            &&
            allowNewConnectionFromPod(
                    destinationPod,
                    sourcePod,
                    protocol,
                    destinationPort)
    then
            connection = newConnection(sourcePod,
                            destinationPod,
                            protocol,
                            destinationPort)
            return <true, connection.connectionId>
    else
            return <false, 0>
}
```

Kaloom™ Copyright 2021

kaloom

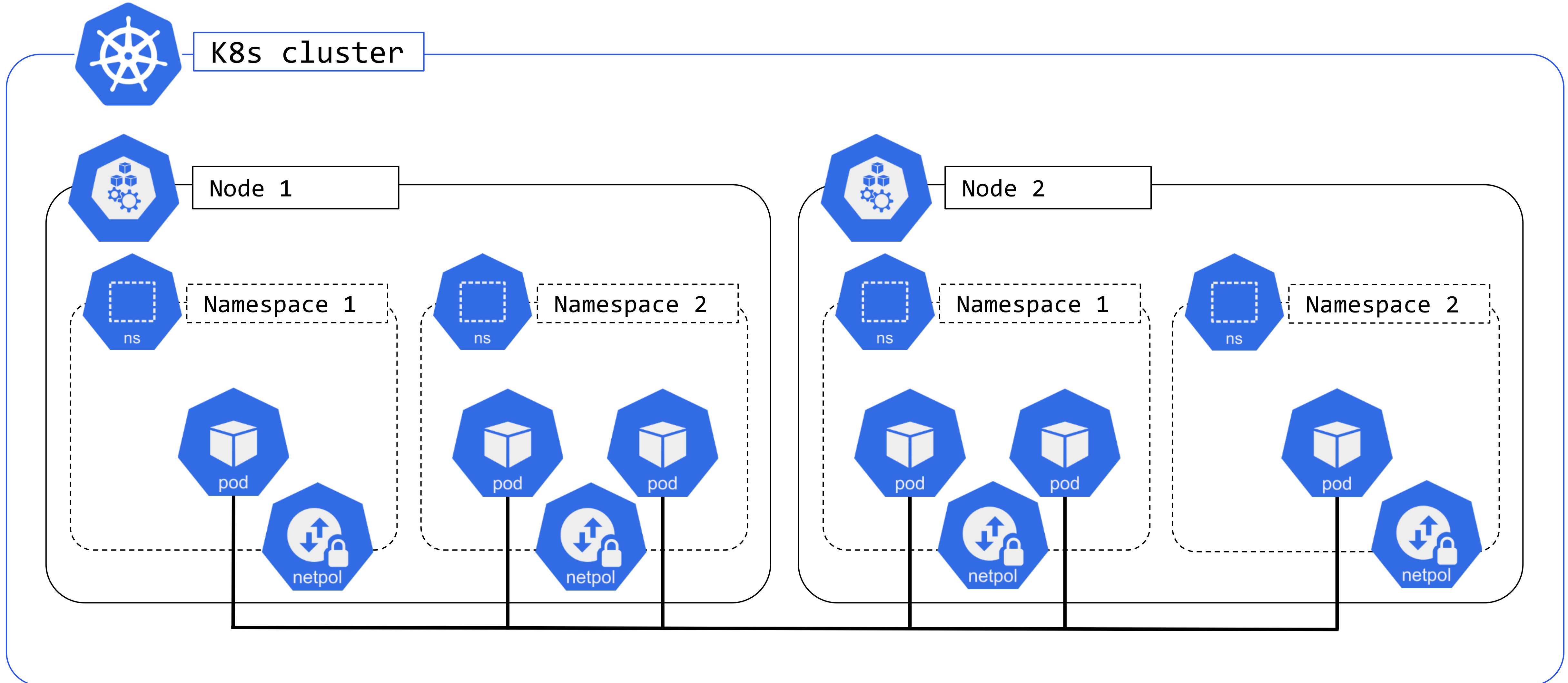
Simple POD Network Policy reference model



- Connection Tracker
  - Any packet matching an existing connection can pass through to its destination
  - Any packet arriving from the Ingress or Egress Policy Filter is sent to its destination
  - Any packet arriving from eth0 that does not match an existing connection and is classified as able to initiate a new connection is passed to the Egress Policy Filter or otherwise dropped.
  - Any packet arriving from nth0 that does not match an existing connection and is classified as able to initiate a new connection is passed to the Ingress Policy Filter or otherwise dropped
- Ingress Policy Filter
  - The packet is checked if it matches any of the **from** selectors, if successful, the new bi-directional connection is registered in the Connection Tracker and the packet is send back to the connection tracker to be forwarded to its destination , other wise the packet is dropped
- Egress Policy Filter
  - The packet is checked if it matches any of the **to** selectors, if successful, the new bi-directional connection is registered in the Connection Tracker and the packet is send back to the connection tracker to be forwarded to its destination , other wise the packet is dropped
- The source port is never used in filters, it is typically ephemeral and requires more complex expressions to specify

Kaloom™ Copyright 2021

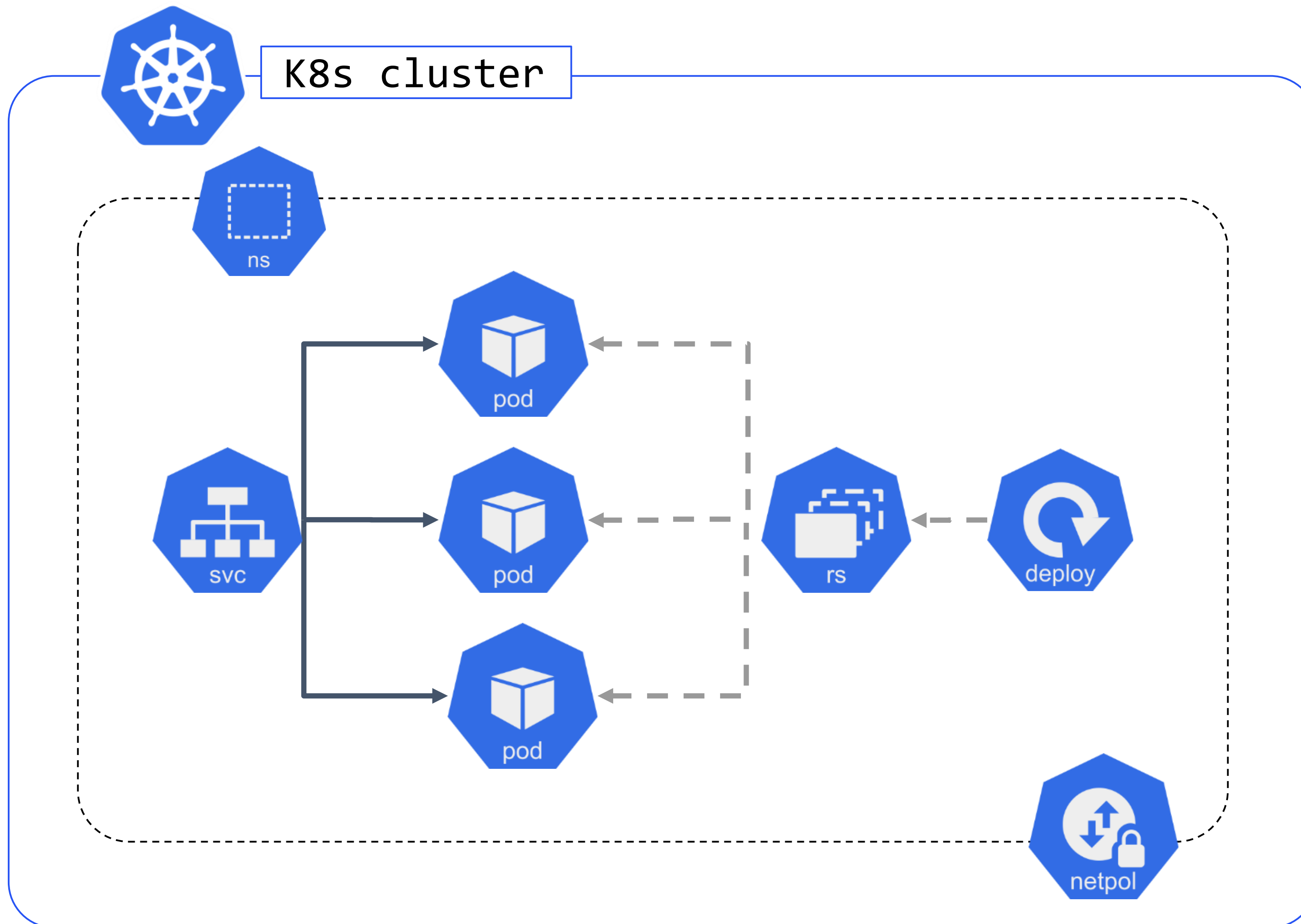
All pods can communicate with each other, if there is Network Policy that allows it



Kaloom™ Copyright 2021

‹kaloom›

Services



- Define a Service

```
apiVersion: v1
kind: Service
metadata:
        name: my-service
spec:
  selector:
        app: my-app
  ports:
    -protocol: TCP
        port: 231
        targetPort: 123
```

- Define a Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
        name: my-app-deployment
spec:
  selector:
    matchLabels:
        app: my-app
    replicas: 3
    template:
        metadata:
            labels:
                app: my-app
    spec:
        containers:
            - name: my-app
              image: my-app-1.2.3
              ports:
                - containerPort: 123
```

Kaloom™ Copyright 2021

kaloom

Service lifecycle and Service to Pod mapping

- Service is the basic construct used to load balance Kubernetes applications.
  - It is built in and works in conjunction with workload resources like Deployment, StatefulSet and DaemonSet to provide application scalability and load balancing.
  - A Service is mapped to a set of Pods by using label and label selectors, the mapping can be simplified though by defining a Service <-> Pod mapping service
- The Service life cycle can be abstracted to
  - addService(service<namespace, name>)
  - removeService(servcice<namespace, name>)
- The mapping service can be abstracted to six primitives
  - addPod2ServiceMapping(pod<namespace, name>, service<namespace, name>)
  - removePod2ServiceMapping(pod<namespace, name>, service<namespace, name>)
  - removePod2ServiceMappings(pod<namespace, name>)
  - removeService2PodMappings(service<namespace, name>)
  - lookupPod2ServiceMappings(pod<namespace, name>) => {service<namespace, name>*}
  - lookupService2PodMappings(Service<namespace, name>) => {pod<namespace, name>*}

Kaloom™ Copyright 2021

Pod and Service lifecycle updates and service to pod communication

- Pod lifecycle: removePod must clean up the Service mappings

```
removePod(pod<namespace, name>)
{
        removePod2ServiceMappings(pod<namespace, name>)
 }
```

- Service lifecycle: removeService must clean up the Pod mappings

```
removeService(service<namespace, name>)
{
        removeService2PodMappings(service<namespace, name>)
 }
```

- Service to pod communication can be abstracted to one primitive
  - openServiceConnection(
      sourcePod,
      destinationService,
      protocol,
      port)
    => <true, connectionId> or <false, 0>

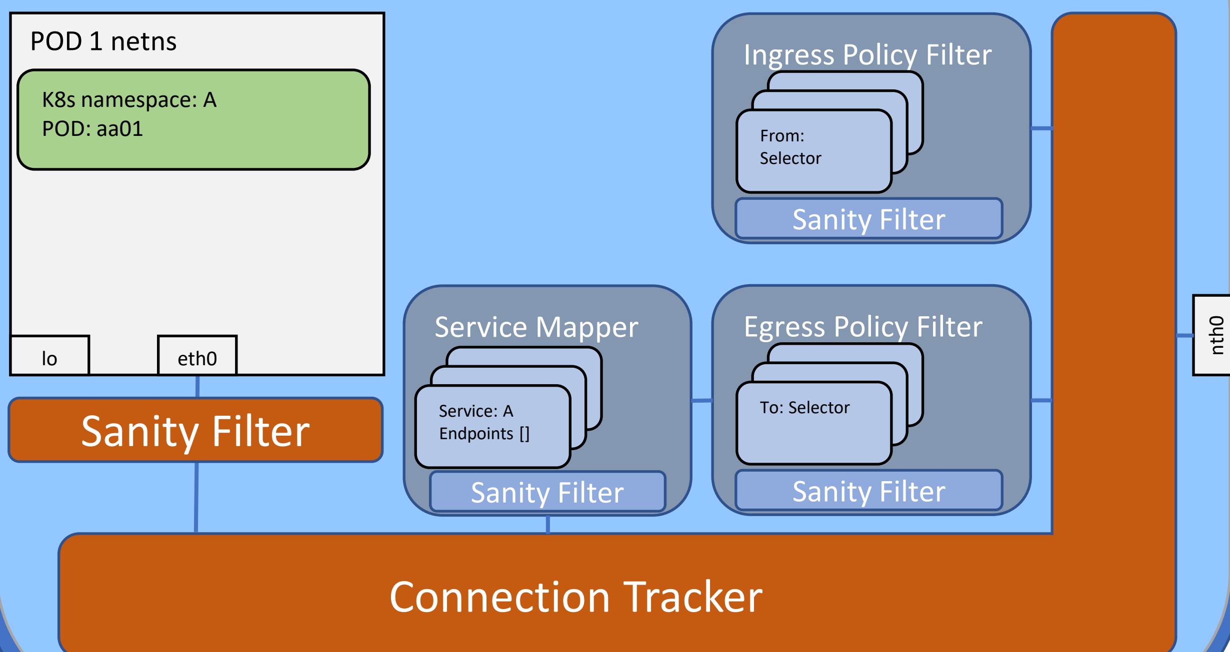
OpenServiceConnection can then specified as

```
openServiceConnection(sourcePod,
                      destinationService,
                      protocol,
                      port)
 {
        podSet pods =
        lookupService2PodMappings(destinationService)
        if isEmptySet(pods) then
                return <false, 0>
        else
                return openPodConnection(sourcePod,
                                         pods.SelectOnePod(),
                                         protocol,
                                         port)
 }
```

Kaloom™ Copyright 2021

kaloom

Simple POD Network Policy reference model



- Connection Tracker
  - Any packet matching an existing connection can pass through to its destination
  - Any packet arriving from the Ingress or Egress Policy Filter is sent to its destination
  - Any packet arriving from eth0 that does not match an existing connection and is classified as able to initiate a new connection is passed to the Egress Policy Filter or otherwise dropped.
  - Any packet arriving from nth0 that does not match an existing connection and is classified as able to initiate a new connection is passed to the Ingress Policy Filter or otherwise dropped
- Service Mapper
  - The packets are matched against the service definitions, if one is found then one of the pod instances in the endpoints list is selected as the destination for the new connection
  - The packet is checked by the Egress Policy Filter, using selected pod instance as the destination pod, , if successful, the new bi-directional session is registered in the Connection Tracker and the packet is send back to the connection tracker to be forwarded to its destination , other wise the packet is dropped
- Ingress Policy Filter
  - The packet is checked if it matches any of the **from** selectors, if successful, the new bi-directional connection is registered in the Connection Tracker and the packet is send back to the connection tracker to be forwarded to its destination , other wise the packet is dropped
- Egress Policy Filter
  - The packet is checked if it matches any of the **to** selectors, if successful, the new bi-directional connection is registered in the Connection Tracker and the packet is send back to the connection tracker to be forwarded to its destination , other wise the packet is dropped
- The source port is never used in filters, it is typically ephemeral and requires more complex expressions to specify

Kaloom™ Copyright 2021

kaloom

Basic Kubernetes Network Semantics

- The basic Network semantics can be summarized in
  - All Pods can communicate with each other, if there is a network policy rule that allows it
  - The Network Policy regulate connectivity between Pods
  - A Service can load balance connectivity towards a set of Pods

Kaloom™ Copyright 2021

Summary Pod and Service Abstractions

Pod lifecycle

- addPod(pod<namespace, name>)
- removePod(pod<namespace, name>)

Service lifecycle

- addService(service<namespace, name>)
- removeService(servcice<namespace, name>)

Connection lifecyle

- newConnection(Pod1, Pod2, protocol, port) =>
  connection<connectionId, Pod1, Pod2>
- deleteConnection(connectionId)

Pod Communication

- openPodConnection(sourcePod,
  destinationPod,
  protocol,
  port) =>
  <true, connectionId> or <false, 0>

Service Communication

- openServiceConnection(sourcePod,
  destinationService,
  protocol,
  port) =>
  <true, connectionId> or <false, 0>

Pod to Service mapping

- addPod2ServiceMapping(pod<namespace, name>,
  service<namespace, name>)
- removePod2ServiceMapping(pod<namespace, name>,
  service<namespace, name>)
- removePod2ServiceMappings(pod<namespace, name>)
- removeService2PodMappings(service<namespace, name>)
- lookupPod2ServiceMappings(pod<namespace, name>) =>
  {service<namespace, name>*}
- lookupService2PodMappings(Service<namespace, name>) =>
  {pod<namespace, name>*}

Kaloom™ Copyright 2021

kaloom

Abstract Network Policy Filter System

- Network Policy primitives
  - addNetworkPolicy(
    networkPolicy<namespace, name , policy)
  - updateNetworkPolicy(
    networkPolicy<namespace, name , policy)
  - removeNetworkPolicy(
    networkPolicy<namespace, name)
  - allowNewConnectionFromPod(
    destinationPod,
    sourcePod,
    protocol,
    destinationPort)
  - allowNewConnectionToPod(
    sourcePod,
    destinationPod,
    protocol,
    destinationPort)
  - allowNewConnectionFromIpAddress(
    sourceIpAddress,
    destinationPod,
    protocol,
    destinationPort)
  - allowNewConnectionToIpAddress(
    sourcePod,
    destinationIpAddress,
    protocol,
    destinationPort)

- This policy filter system is not dependent on
  - The number of pod replicas
  - The number of pod interfaces
  - The number of pod network attachments
  - Which interface an ip address is configured to
- The policy filter system is dependent on
  - Pod manifests
  - Pod labels
  - Label selectors in the Network Policies
- It is only updated when
  - policy filters are added, updated or removed
  - Labels used in policy filters are changed
- It is easy to extend with functionality
  - show how policies are related
  - which policies that applies towards a
    - namespace
    - service
    - workload entity
    - individual pods

Kaloom™ Copyright 2021

kaloom

Definition of primitives

- **removePod**

```
removePod(pod<namespace, name>)
{
        removePod2ServiceMappings(pod<namespace, name>)
        removePod2AddressMappings(pod<namespace, name>)
}
```

- **removeService**

```
removeService(service<namespace, name>)
{
        removeService2PodMappings(service<namespace, name>)
        removeService2AddressMappings(service<namespace, name>)
}
```

- **openServiceConnection**

```
openServiceConnection(sourcePod,
                      destinationService,
                      protocol,
                      port)
    {
        podSet pods =
        lookupService2PodMappings(destinationService)
        if isEmptySet(pods) then {
                return <false, 0>
        } else {
                return openPodConnection(sourcePod,
                                         pods.SelectOnePod(),
                                         protocol,
                                         port)
        }
    }
```

- **openPodConnection**

```
openPodConnection(sourcePod, destinationPod, protocol, port)
{
        if
                allowNewConnectionToPod(
                        sourcePod,
                        destinationPod,
                        protocol,
                        destinationPort)
                &&
                allowNewConnectionFromPod(
                        destinationPod,
                        sourcePod,
                        protocol,
                        destinationPort)
        then
                connection = newConnection(sourcePod,
                                           destinationPod,
                                           protocol,
                                           destinationPort)
                return <true, connection.connectionId>
        else
                return <false, 0>
}
```

Kaloom™ Copyright 2021

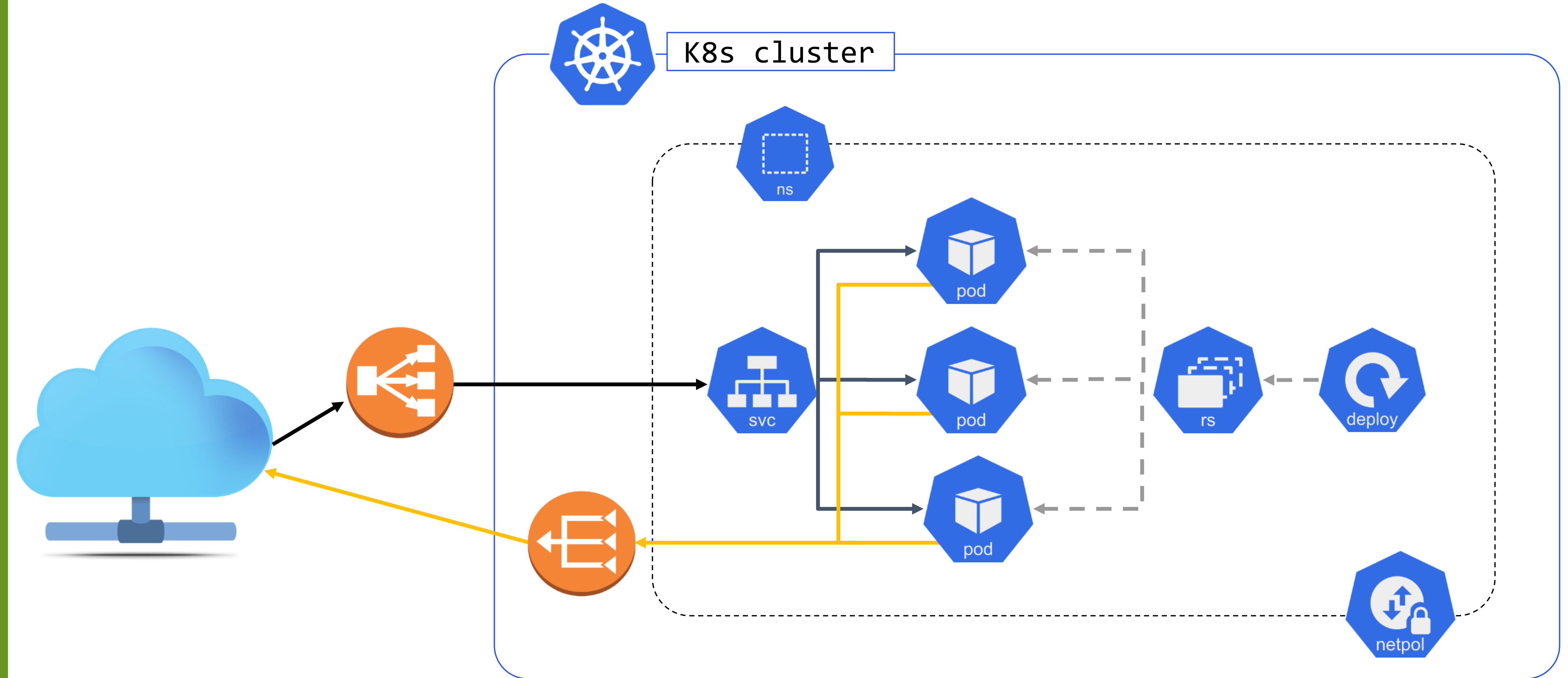
Extend model with support for single network and pods with single Network Attachment with single IP address

- Every Pod should have one IP address added to the eth0 interface that is attached to the common network
- Every Service is assigned one Virtual IP address
- Network Policy must support the IpBlock in egress and ingress rules
- Add support for connectivity to and from cluster external entities

Kaloom™ Copyright 2021

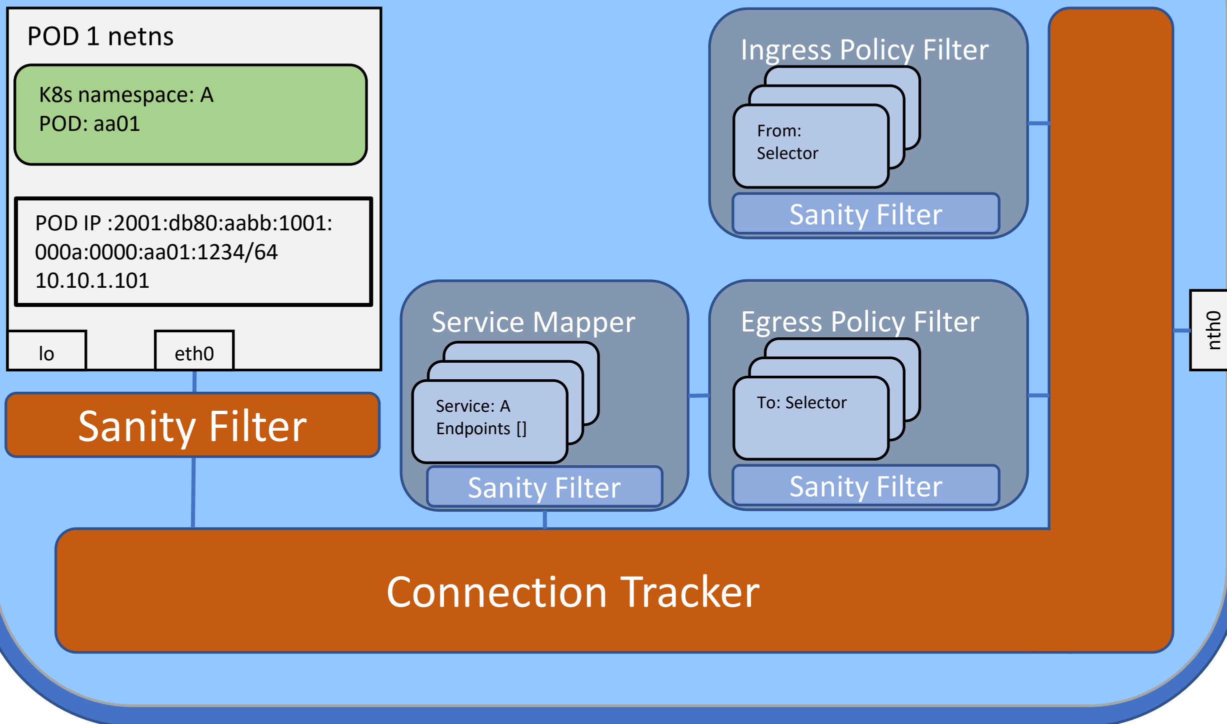
‹kaloom›

Networking with SNAT and external load balancer



Kaloom™ Copyright 2021
kaloom

Simple POD Network Policy reference model



- Connection Tracker
  - Any packet matching an existing connection can pass through to its destination
  - Any packet arriving from the Ingress or Egress Policy Filter is sent to its destination
  - Any packet arriving from eth0 that does not match an existing connection and is classified as able to initiate a new connection is passed to the Egress Policy Filter or otherwise dropped.
  - Any packet arriving from nth0 that does not match an existing connection and is classified as able to initiate a new connection is passed to the Ingress Policy Filter or otherwise dropped
- Sanity Filter
  - All packets that are received from eth0 are checked to ensure that the source IP address is assigned to either eth0 or lo
- Service Mapper
  - All packets arriving are checked by the sanity filter to ensure that the destination IP address is an address that is a service address
  - The packets are matched against the service definitions, if one is found then one of the pod instances in the endpoints list is selected as the destination for the new connection
  - The packet is checked by the Egress Policy Filter, using selected pod instance address as the destination address, , if successful, the new bi-directional session is registered in the Connection Tracker and the packet is send back to the connection tracker to be forwarded to its destination, other wise the packet is dropped
- Ingress Policy Filter
  - All packets arriving are checked by the sanity filter to ensure that the destination IP address is an address that is assigned to either eth0 or lo
  - The packet is checked if it matches any of the **from** selectors, if successful, the new bi-directional session is registered in the Connection Tracker and the packet is send back to the connection tracker to be forwarded to its destination , other wise the packet is dropped
- Egress Policy Filter
  - All packets arriving are checked by the sanity filter to ensure that the source IP address is a pod IP address assigned to eth0
  - The packet is checked if it matches any of the **to** selectors, if successful, the new bi-directional session is registered in the Connection Tracker and the packet is send back to the connection tracker to be forwarded to its destination , other wise the packet is dropped

Kaloom™ Copyright 2021

kaloom

Map ip addresses to Pod And Services

- Add a mapping service from Pod to ip addresses
  - addAddress2PodMapping(ipAddress, pod<namespace, name>)
  - removeAddress2PodMapping(ipAddress, pod<namespace, name>)
  - removePod2AddressMappings(pod<namespace, name>)
  - lookupPod2AddressMappings(pod<namespace, name>) => {ipAddress*}
  - lookupAddress2PodMapping(ipAddress) => {pod<namespace, name>?}
- Add a mapping service from Service to ip addresses
  - addAddress2ServiceMapping(ipaddress, service<namespace, name>)
  - removeAddress2ServiceMapping(ipaddress , service<namespace, name>)
  - removeService2AddressMappings(service<namespace, name>)
  - lookupService2AddressMappings(service<namespace, name>) => {ipAddress*}
  - lookupAddress2ServiceMapping(ipAddress => {service<namespace, name>?}

Kaloom™ Copyright 2021

New primitives and lifecycle updates

- Communication must be extended with two new primitives
  - openConnectionWithIpAddresses(sourceIpAddress,
    destinationIpAddress,
    protocol,
    port) => <true, connectionId> or <false, 0>
  - newConnection(Pod1, ipAddress, protocol, port) =>
    connection<connectionId, Pod1, Pod2>
- Network Policy system must be extended with two more primitives
  - allowNewConnectionFromIpAddress(
    sourceIpAddress,
    destinationPod,
    protocol,
    destinationPort)
  - allowNewConnectionToIpAddress(
    sourcePod,
    destinationIpAddress,
    protocol,
    destinationPort)
- Pod lifecycle: removePod must clean up the Address mappings

```
removePod(pod<namespace, name>)
{
        removePod2ServiceMappings(pod<namespace, name>)
        removePod2AddressMappings(pod<namespace, name>
}
```

- Service lifecycle: removeService must clean up the Address mappings

```
removeService(service<namespace, name>)
{
        removeService2PodMappings(service<namespace, name>)
        removeService2AddressMappings(service<namespace, name>)
}
```

Kaloom™ Copyright 2021

Definition of openConnectionFromIpAddresses

```
openConnectionFromIpAddresses(
                sourceIpAddress,
                destinationIpAddress,
                protocol,
                port)
{
                // find out if addresses maps to pod or services
                podSet sourcePod = lookupAddress2PodMapping(sourceIpAddress)
                serviceSet destService = lookupAddress2ServiceMapping(destinationIpAddress)
                podSet destPod

                if destService.isEmpty() then {// find out if destination address maps to a pod
                                destPod = lookupAddress2PodMapping(destinationIpAddress)
                } else {
                                destPod = lookupService2PodMappings(destService.selectOnePod())
                }
                connection con
                if destPod.isEmpty() && sourcePod.isNotEmpty() &&
                                allowNewConnectionToIpAddress(sourcePod[0],
                                                                destinationIpAddress,
                                                                protocol,
                                                                destinationPort) then
                                con = newConnection(sourcePod[0],
                                                destinationIpAddress,
                                                protocol,
                                                destinationPort)
                if sourcePod.isEmpty() then {
                                if allowNewConnectionFromIpAddress(sourceIpAddress,
                                                                destPod[0],
                                                                protocol,
                                                                destinationPort)  then

                                con = newConnection(destPod[0],
                                                sourceIpAddress,
                                                protocol,
                                                destinationPort)
                } else {
                                if allowNewConnectionFromPod(sourcePod[0],
                                                                destPod[0],
                                                                protocol,
                                                                destinationPort) then
                                con = newConnection(sourcePod[0],
                                                destPod[0],
                                                protocol,
                                                destinationPort)
                }

                if con.notEmpty() then
                                return <true, connection.connectionId>
                else
                                return <false, 0>
        }
```

Kaloom™ Copyright 2021

kaloom™

Where to go from here

- Step 1: Multi Network and multi network attachments
- Step 2: Overlapping ip address spaces
- Step 3: Service load balancing for secondary networks
- Step 4: Service chaining

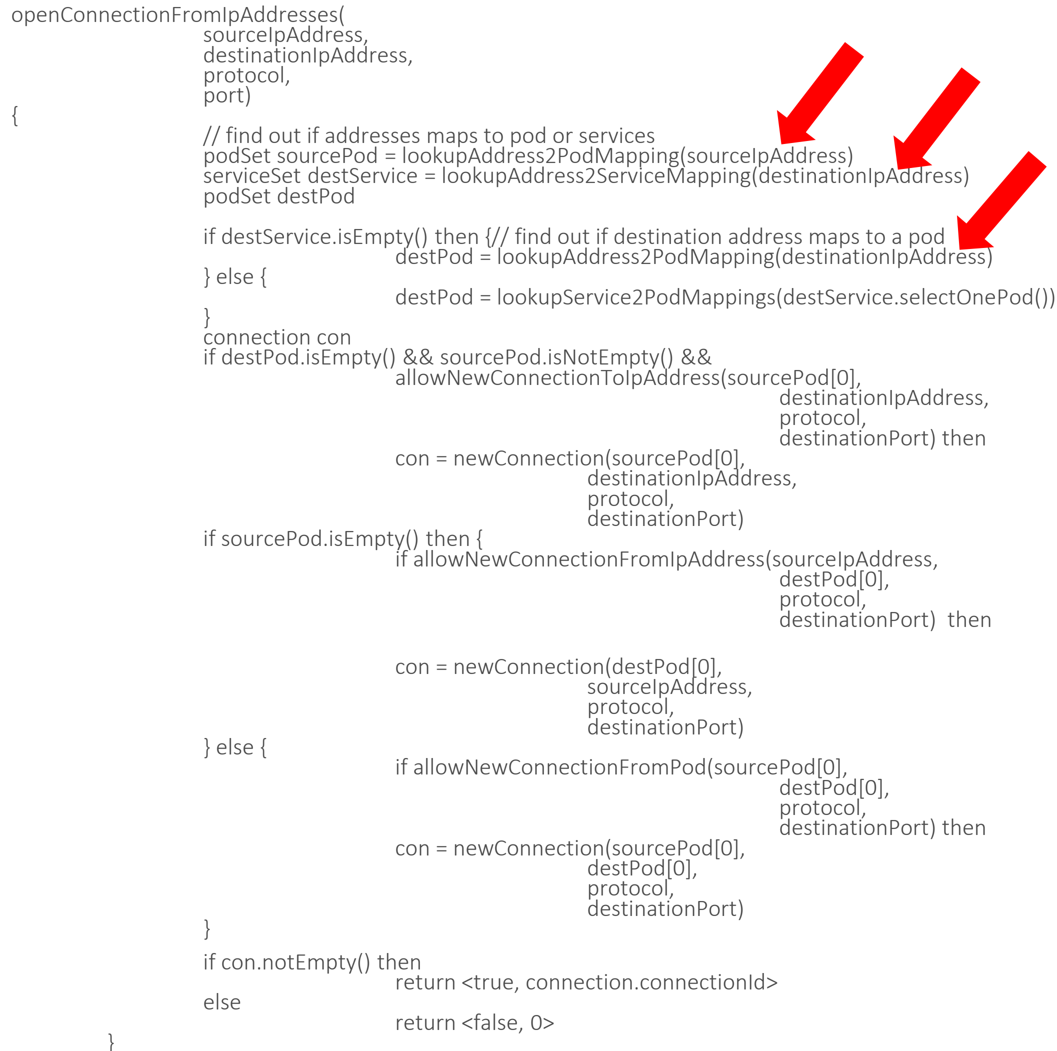
Kaloom™ Copyright 2021

Step 1:Extend model with support for multi network and pods with multi–Network Attachments with multiple IP addresses

- A Pod can have multiple network attachments towards one or more networks
- A Pod can have one or more ip addresses assigned to each Network attachment

Kaloom™ Copyright 2021

Let's look at openConnectionFromIpAddresses again



```
openConnectionFromIpAddresses(
                sourceIpAddress,
                destinationIpAddress,
                protocol,
                port)
{
                // find out if addresses maps to pod or services
                podSet sourcePod = lookupAddress2PodMapping(sourceIpAddress)
                serviceSet destService = lookupAddress2ServiceMapping(destinationIpAddress)
                podSet destPod

                if destService.isEmpty() then {// find out if destination address maps to a pod
                                destPod = lookupAddress2PodMapping(destinationIpAddress)
                } else {
                                destPod = lookupService2PodMappings(destService.selectOnePod())
                }
                connection con
                if destPod.isEmpty() && sourcePod.isNotEmpty() &&
                                allowNewConnectionToIpAddress(sourcePod[0],
                                                                destinationIpAddress,
                                                                protocol,
                                                                destinationPort) then
                                con = newConnection(sourcePod[0],
                                                destinationIpAddress,
                                                protocol,
                                                destinationPort)
                if sourcePod.isEmpty() then {
                                if allowNewConnectionFromIpAddress(sourceIpAddress,
                                                                destPod[0],
                                                                protocol,
                                                                destinationPort)  then

                                con = newConnection(destPod[0],
                                                sourceIpAddress,
                                                protocol,
                                                destinationPort)
                } else {
                                if allowNewConnectionFromPod(sourcePod[0],
                                                                destPod[0],
                                                                protocol,
                                                                destinationPort) then
                                con = newConnection(sourcePod[0],
                                                destPod[0],
                                                protocol,
                                                destinationPort)
                }
                if con.notEmpty() then
                                return <true, connection.connectionId>
                else
                                return <false, 0>
        }
```

- Important to understand that everything is based on that
  - Addresses are unique
  - An address can only be assigned once
- From that it is possible to conclude several interesting things
  - It does not matter how many addresses that are mapped to a Pod!!!
  - The only thing that matters is that lookupAddress2PodMapping only can return zero or one Pod
  - The number of addresses returned by lookupPod2AddressMapping is irrelevant for the Kubernetes Network Semantics
    - But it depends on the network topology for service to pod mapping*
- The Kubernetes Network semantics is not changed by
  - The number networks used in the system
  - The number of network attachments in a pod
  - The number of networks attached to a pod
  - The number of IP addresses assigned to an interface/network attachment
- That said, the network environment in any multi homed Pod can become very challenging
  - How to handle routes, VRF, overlapping address spaces…..

Kaloom™ Copyright 2021

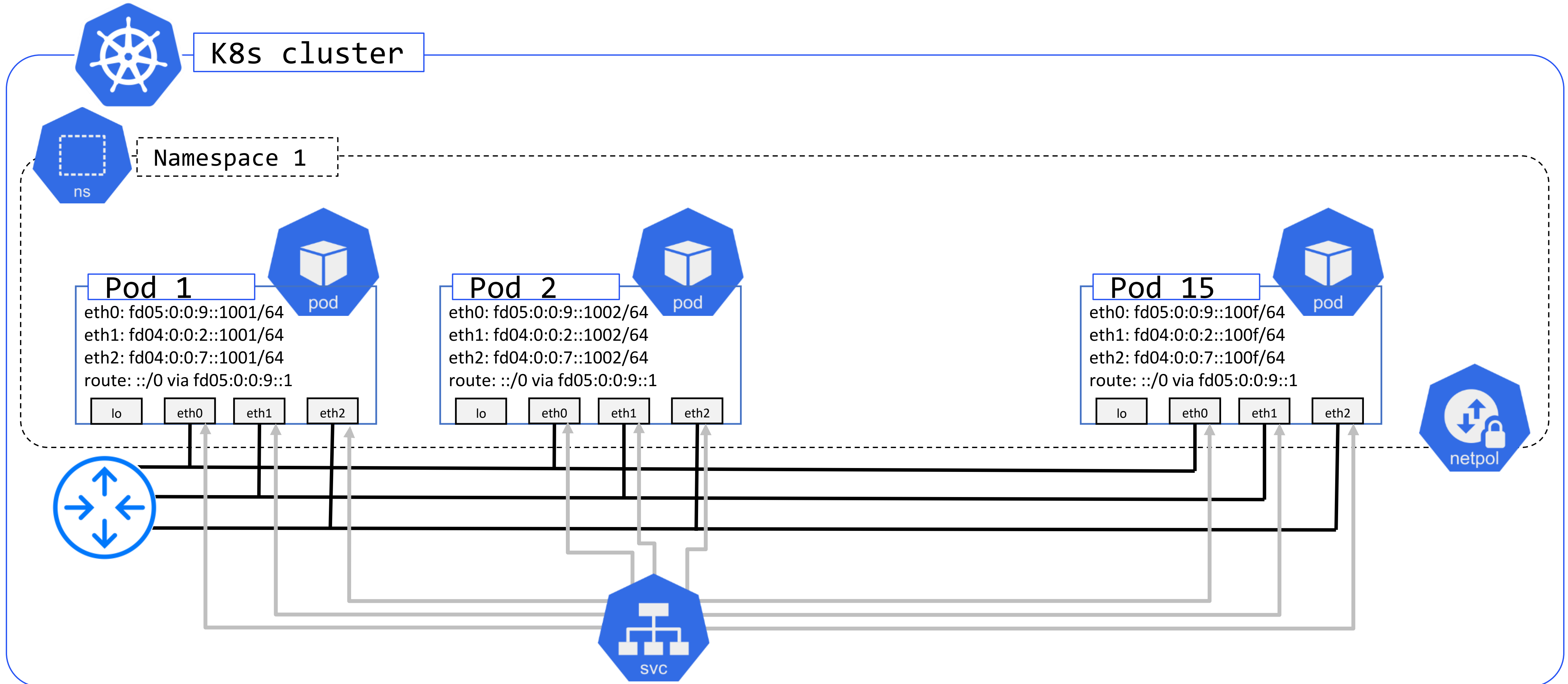
‹kaloom›

Two scenarios

- L3 reachability between all networks (type 1)
  - Network Policy model works for all addresses on all networks
  - Service lookup model works, service can be mapped to any pod address on any network and any interface
- No L3 reachability between all networks (type 2)
  - Network Policy model works for all addresses on all networks
  - Service lookup model only works for addresses assigned to eth0 interface in Pod
    - Possible to extend service manifest with annotation to handle this. This will be defined later in “Step 4”

Kaloom™ Copyright 2021

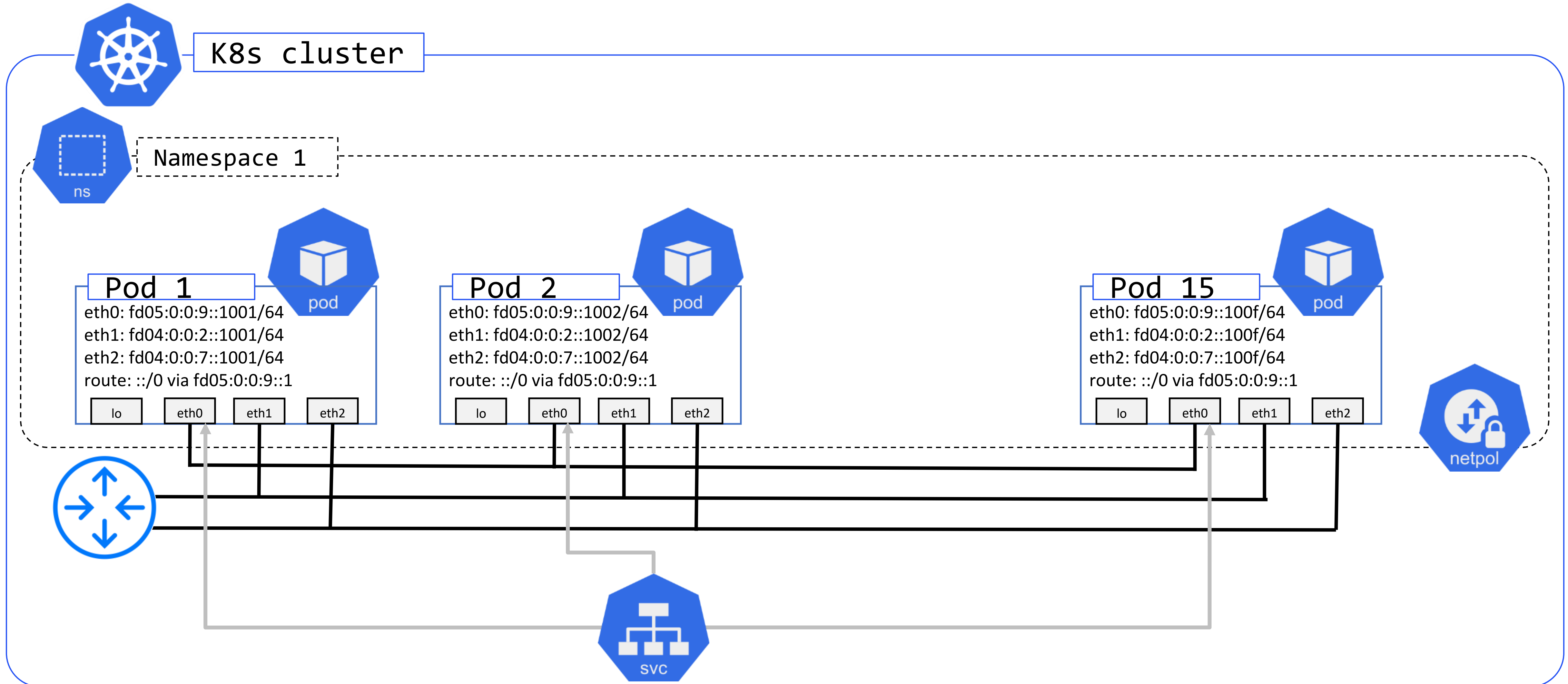
Multi Network type 1



Kaloom™ Copyright 2021

‹kaloom›

Multi Network type 2



Kaloom™ Copyright 2021

kaloom

Step 2: Overlapping address space

- What is an ip address domain
  - How do we express it
  - How do we link it into the current model?
  - How is it found and known at runtime?

Kaloom™ Copyright 2021

What is an ip address domain: How do we express it?

- The simplest way to describe an address domain is to say that the addresses used within that domain are unique and well defined.
- The address domain is an abstract entity, it is not part of a packet, but there are well established conventions and rules around them
- NAT type of functions can be used to translate addresses between two domains
- Routers can be used to protect a domain from receiving packets with not wanted destination addresses
- Firewalls can block packets with not wanted source or destination addresses

Kaloom™ Copyright 2021

What is an ip address domain: How do we link it into the current model?

- When an address is assigned
  - you must know which address domain/overlapping address space it belongs to
- IPAM use address space
  - This address space must be unique within the current domain
    - do not assign addresses from the same prefix in uncoordinated way
  - Interfaces are assigned ip addresses
    - CNI
    - DHCP
    - Route Advertisement
  - Interfaces are attached to networks
  - Networks and IPAM in K8s are tied together through the Network Configuration Specification
- The ip address domain should be added as an attribute to the CNI "Network Configuration Specification"

Kaloom™ Copyright 2021

‹kaloom›

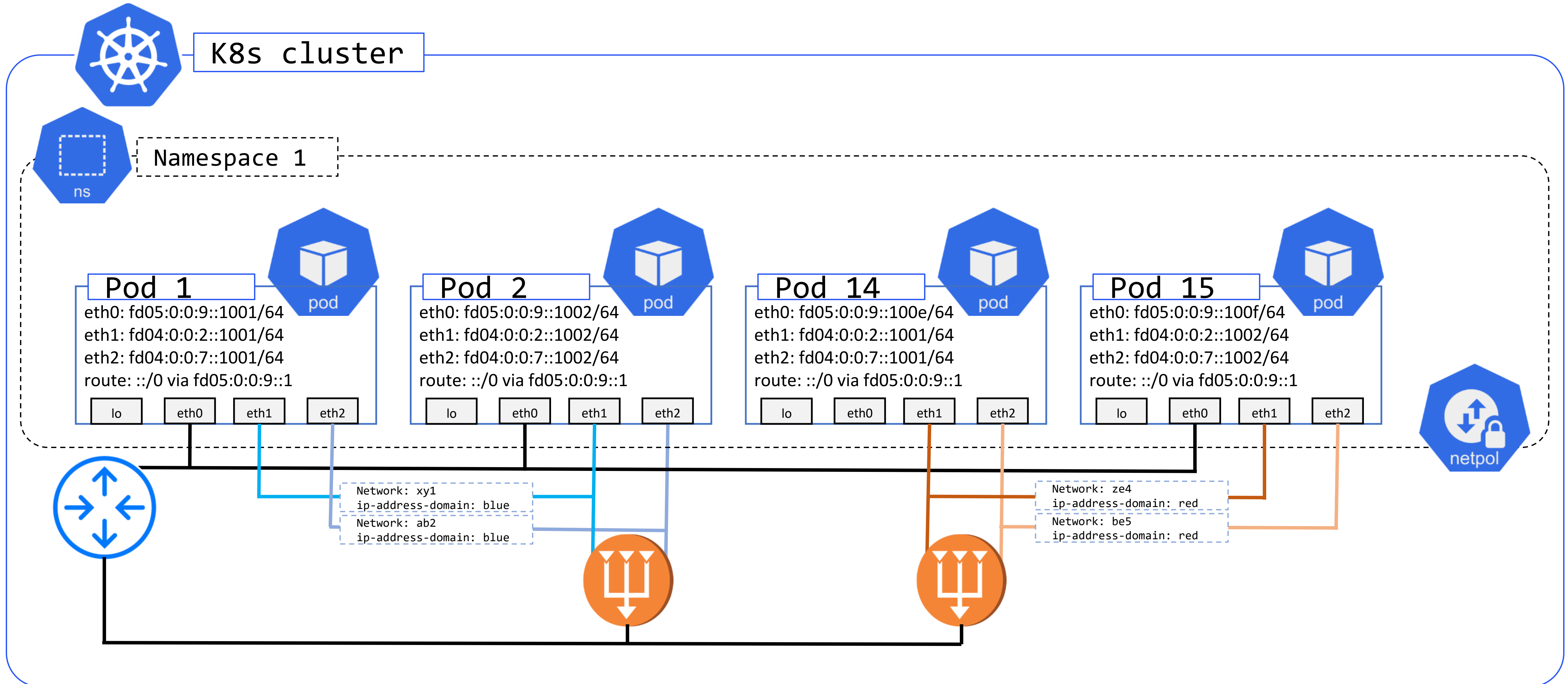
What is an ip address domain: How is it found and known at runtime?

- IpAddressDomain <-> Network <-> interface <-> address
  - An interface is tied to a network, the network belongs to an ip address domain
  - It is therefore known which ip address domain each individual interface in a POD belongs to, and it is easy to map which domain that should be used in lookups for both egress and ingress filters
- Add two new network constructs
  - ip_address_domain<name>
  - network<name, ip_address_domain_name>

Kaloom™ Copyright 2021

kaloom

Ip address domain for separate address spaces



Kaloom™ Copyright 2021

‹kaloom›

Pod, Service and Network Policy Abstraction with support for multiple ip address spaces

Pod and Service Communication

- openConnectionWithIpAddresses(
  sourceIpAddressSpaceName,
  sourceIpAddress,
  destinationIpAddressSpaceName,
  destinationIpAddress,
  protocol,
  port) =>
  <true, connectionId> or <false, 0>

Network Policy primitives

- allowNewConnectionFromIpAddress(
  sourceAddressSpaceName,
  sourceIpAddress,
  destinationPod,
  protocol,
  destinationPort)
- allowNewConnectionToIpAddress(
  sourcePod,
  destinationIpAddressSpaceName,
  destinationIpAddress,
  protocol,
  destinationPort)

Pod to ip addresses mapping

- addAddress2PodMapping(ipAddressSpaceName,
  ipAddress,
  pod<namespace, name>)
- removeAddress2PodMapping(ipAddressSpaceName,
  ipAddress,
  pod<namespace, name>)
- removePod2AddressMappings(pod<namespace, name>)
- lookupPod2AddressMappings(pod<namespace, name>) =>
  {< ipAddressSpaceName ,ipAddress>*}
- lookupAddress2PodMapping(ipAddressSpaceName, ipAddress) =>
  {pod<namespace, name>?}

Service to ip address mapping

- addAddress2ServiceMapping(ipAddressSpaceName,
  ipaddress ,
  service<namespace, name>)
- removeAddress2ServiceMapping(ipAddressSpaceName,
  ipaddress ,
  service<namespace, name>)
- removeService2AddressMappings(service<namespace, name>)
- lookupService2AddressMappings(service<namespace, name>) =>
  {< ipAddressSpaceName ,ipAddress>*}
  lookupAddress2ServiceMapping(ipAddressSpaceName, ipAddress) =>
  {service<namespace, name>?}

Kaloom™ Copyright 2021

openConnectionFromIpAddresses with support for multiple ip address spaces

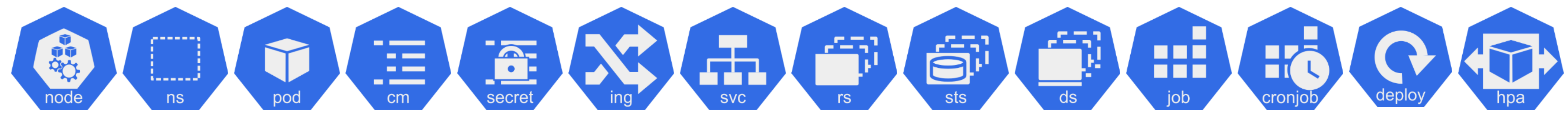
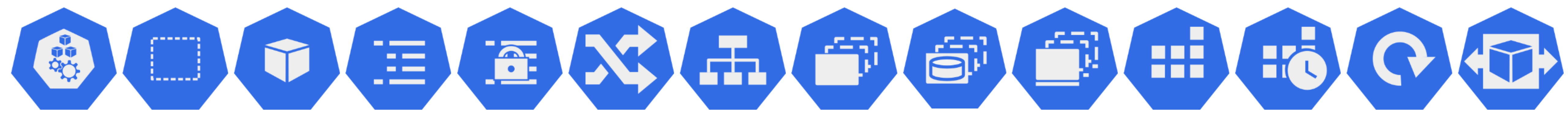
```
openConnectionFromIpAddresses(
                        sourceIpAddress,
                        destinationIpAddress,
                        protocol,
                        port)
{
                        // find out if addresses maps to pod or services
                        podSet sourcePod = lookupAddress2PodMapping(sourceIpAddressSpaceName,
                                                                        sourceIpAddress)
                        serviceSet destService = lookupAddress2ServiceMapping(destinationIpAddressSpaceName ,
                                                                        destinationIpAddress)
                        podSet destPod

                        if destService.isEmpty() then {// find out if destination address maps to a pod
                                destPod = lookupAddress2PodMapping(destinationIpAddressSpaceName,
                                                                        destinationIpAddress)
                        } else {
                                destPod = lookupService2PodMappings(destService.selectOnePod())
                        }
                        connection con
                        if destPod.isEmpty() && sourcePod.isNotEmpty() &&
                                allowNewConnectionToIpAddress(sourcePod[0],
                                                                        destinationIpAddressSpaceName,
                                                                        destinationIpAddress,
                                                                        protocol,
                                                                        destinationPort) then
                                con = newConnection(sourcePod[0],
                                                destinationIpAddress,
                                                protocol,
                                                destinationPort)
                        if sourcePod.isEmpty() then {
                                if allowNewConnectionFromIpAddress(sourceIpAddressSpaceName,
                                                                        sourceIpAddress,
                                                                        destPod[0],
                                                                        protocol,
                                                                        destinationPort)  then

                                con = newConnection(destPod[0],
                                                sourceIpAddressSpaceName,
                                                sourceIpAddress,
                                                protocol,
                                                destinationPort)
                        } else {
                                if allowNewConnectionFromPod(sourcePod[0],
                                                                        destPod[0],
                                                                        protocol,
                                                                        destinationPort) then
                                con = newConnection(sourcePod[0],
                                                destPod[0],
                                                protocol,
                                                destinationPort)
                        }
                        if con.notEmpty() then
                                return <true, connection.connectionId>
                        else
                                return <false, 0>
                }
```

Kaloom™ Copyright 2021

kaloom

Work in progress after this slide

Kaloom™ Copyright 2021


node
ns
pod
cm
secret
ing
svc
rs
sts
ds
job
cronjob
deploy
hpa


netpol
quota
limits
sc
vol
pv
pvc
rb
crb
role
c.role
sa
user
group

ep
k-proxy
kubelet
c-c-m
c-m
sched.
api
etcd
master
psp
crd