meshrr
Hierarchical Route Distribution @ Scale w/ Kubernetes

Jason R. Rokeach
Juniper Networks

NANOG 91 – Kansas City, MO
June 2024

Agenda

- Introduction
- Design
- Client Connectivity
- Use Cases
- Final Thoughts

meshrr

- *"meshrr is a scale-out, hierarchically-capable, BGP route reflector and route server approach using Juniper cRPD on Kubernetes."*
  - meshrr uses Kubernetes to orchestrate containerized route reflectors ("cRRs")
  - Concept could be extended to other containerized BGP daemons suitable for RR.

https://github.com/Juniper/meshrr

Why Kubernetes?

- meshrr is targeted at use cases requiring “more than a few” route reflectors or hierarchies thereof
- Kubernetes enables:
  - simple scaling and replication of workloads (cRRs) with built-in resource management
  - orchestrated network connectivity between & to workloads
  - integrated lifecycle management
  - deployment across multiple nodes and (if desired) geography

https://github.com/Juniper/meshrr

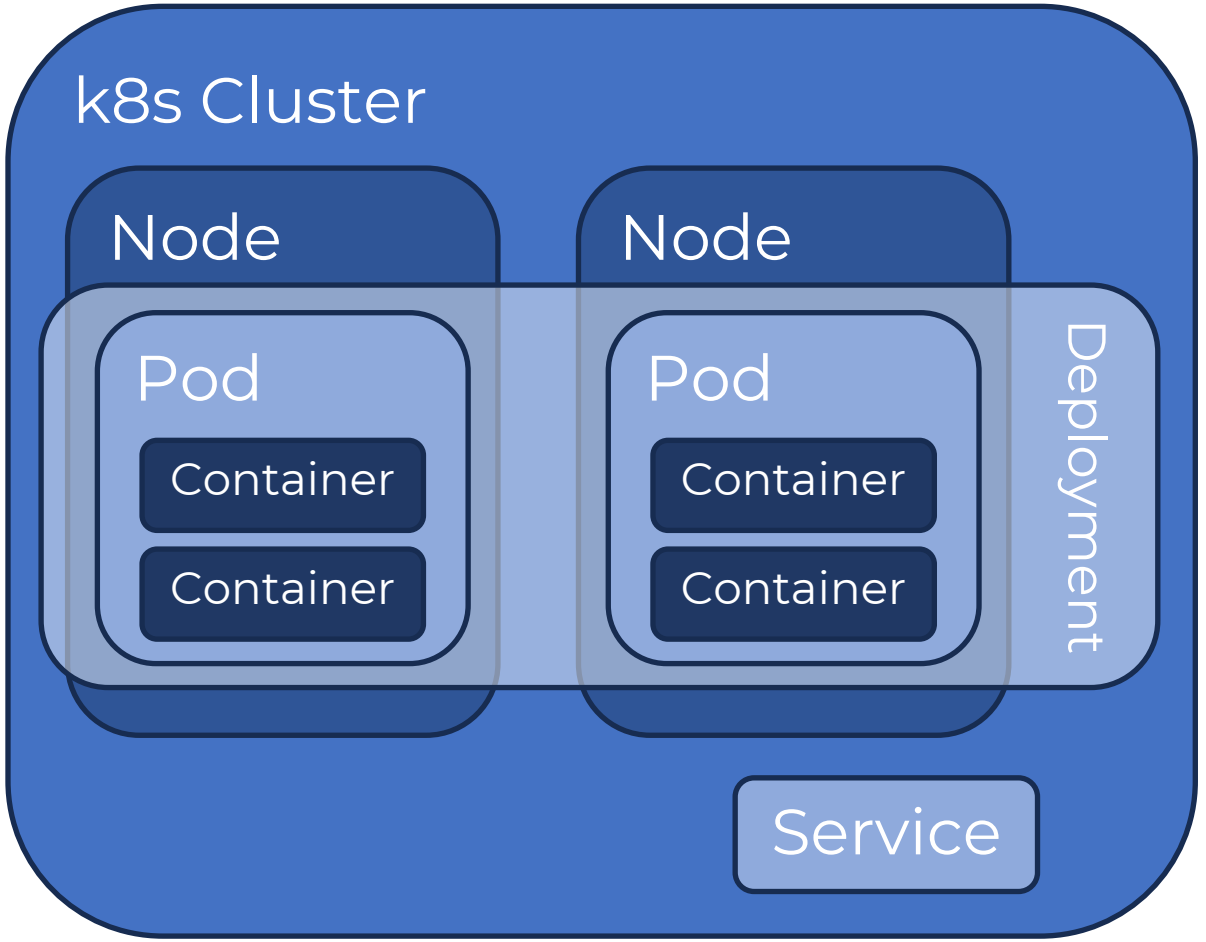
Inspiration

- Network operator required that different groups of routers discard routes based on policy specific to that group
- To avoid blackholing traffic based on best path selection (even with add-path), required either:
  - over 100 highly-peered routers in full mesh, or
  - many RRs, each with different routing policy, some of which only servicing 2 or 3 routers each

https://github.com/Juniper/meshrr

Kubernetes Terminology

- k8s: Kubernetes
- Node: “a worker machine in Kubernetes.” (VM or BMS)
- Cluster: “A set of nodes.”
- Container: “lightweight and portable executable image that contains software and all of its dependencies”
- Deployment: “manages a replicated application…”
- Pod: “… represents a set of running containers.”
- Service: “… exposing a network application that is running as one or more pods.”



Definitions from https://kubernetes.io/docs

What is meshrr?

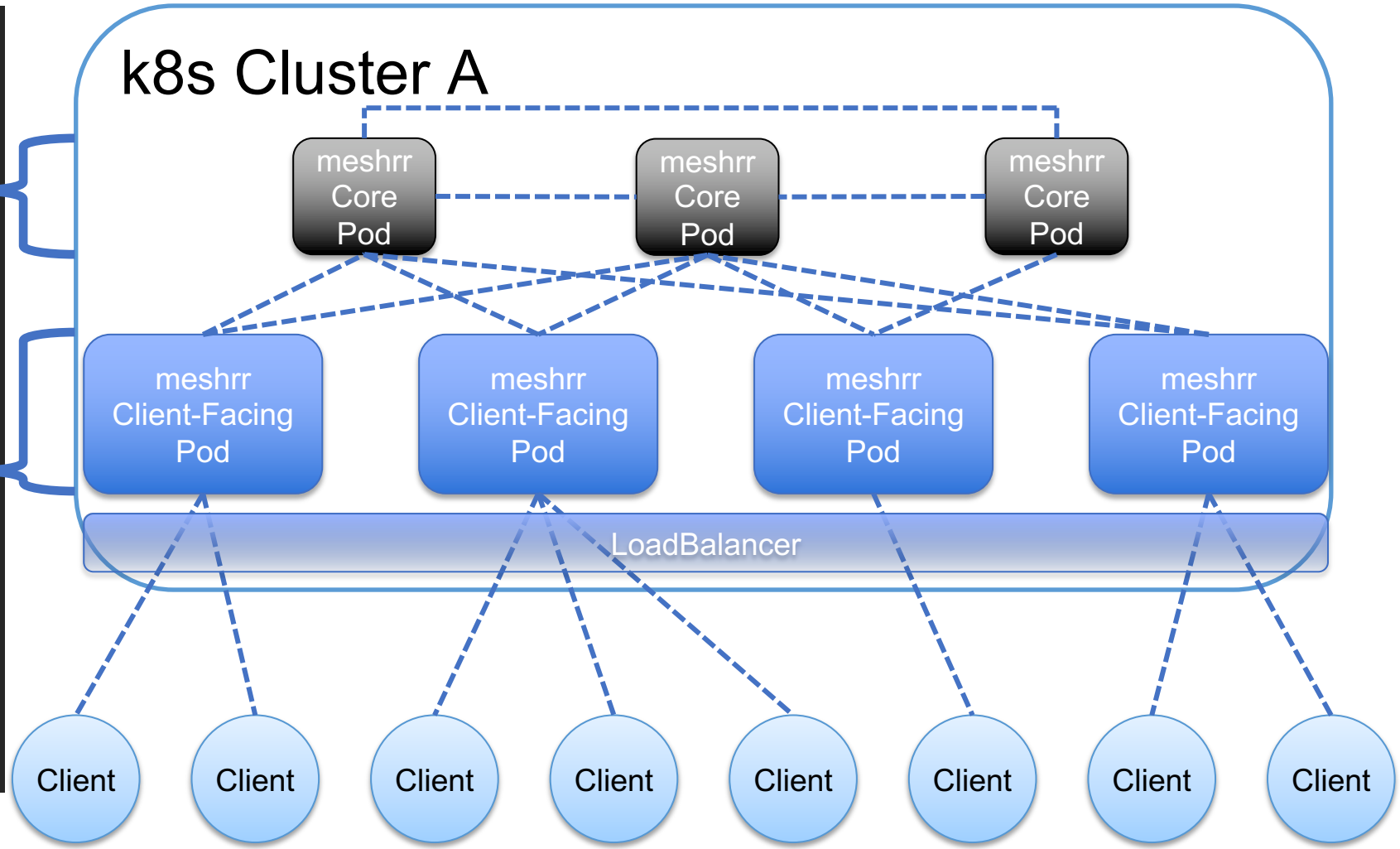
- Forms meshes of BGP route reflectors based on groups discovered by Kubernetes Services' DNS
- meshrr cRRs are *defined* by k8s manifests, not *configured* by CLI or automation tools for PNFs
- Simplest case: Form a single full mesh of route reflectors within a k8s cluster
- More complex cases:
  - Hierarchy of cRR groups with different configurations
  - Control client group membership and policy by IP address using Kubernetes services or external DNS

https://github.com/Juniper/meshrr

UC1: Scale-Out Hierarchical cRR

Core: **mesh** group between Core cRRs
subtractive group allows client-facing' cRRs to connect
Client-Facing: **mesh** group with max-peers 2 to Core cRRs
subtractive group for physical client connections

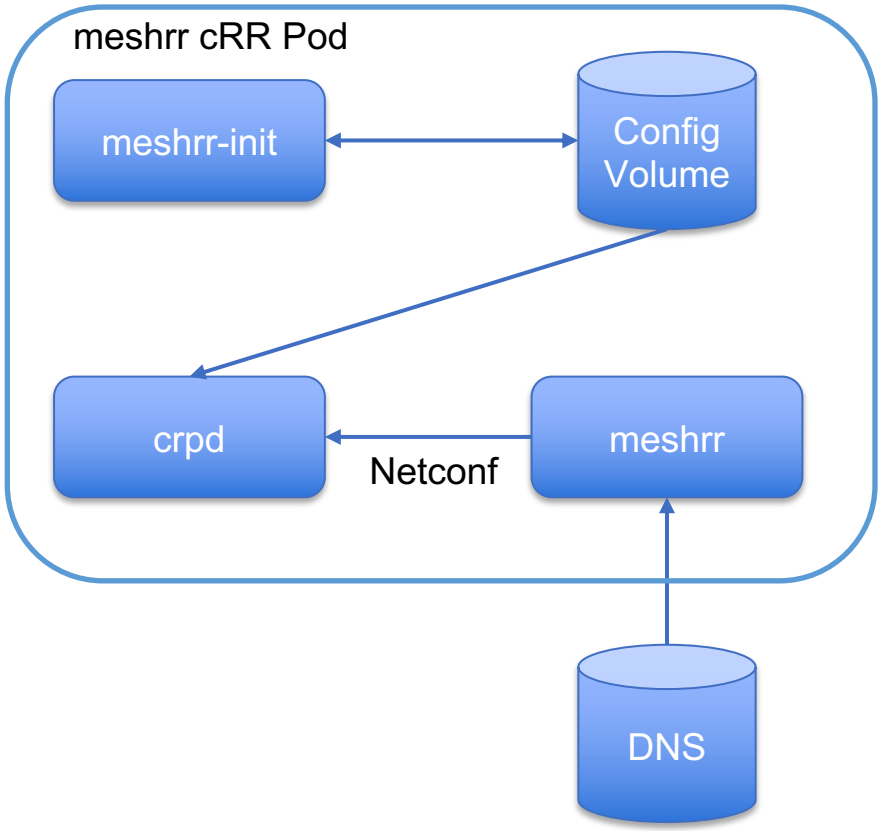
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: meshrr-core
spec:
  replicas: 3
<...>
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: meshrr-
clientfacing
spec:
  replicas: 4
<...>
```



Design

Single meshrr Pod

- meshrr-init (Init Container):
  - Creates router configuration from Jinja2 template, or derives configuration from persistent storage and updates variables
  - Creates pod-specific SSH pubkey
  - Image: ghcr.io/juniper/meshrr:0.2
- crpd:
  - Unmodified routing daemon
- meshrr:
  - Periodically checks DNS for updates to the dynamic list of cRRs in the k8s cluster and pushes changes to cRPD via Netconf
  - Image: ghcr.io/juniper/meshrr:0.2



Headless Services

- Provide the DNS to glue cRRs together
- The YAML displayed dynamically creates DNS records in k8s for *meshrr-core.default.svc.cluster.local* to point to all active pods with labels `app: meshrr` and `meshrr_region_core: "true”`

```yaml
apiVersion: v1
kind: Service
metadata:
  name: meshrr-core
spec:
  clusterIP: None
  ports:
  - name: bgp
    port: 179
    protocol: TCP
    targetPort: bgp
  selector:
    app: meshrr
    meshrr_region_core: "true"
  sessionAffinity: None
  type: ClusterIP
```

meshrr YAML Configuration

- Passed to meshrr containers as a volume mount
- General Parameters (Root PW, ASN)
- *mesh BGP groups* discover BGP peers through DNS and actively connect to them
  - Each mesh BGP group is represented in one or more Kubernetes Service resources, which provide DNS for this discovery process
  - Manually-defined endpoints in services or external DNS can also be used for group DNS resolution
- *subtractive BGP groups* accept a prefix in meshrr configuration and remove all discovered *mesh* neighbors from that prefix
  - Primarily for connections from physical RR clients or lower hierarchy
  - Configures BGP group(s) allowing a range, not explicit neighbors

meshrr Configuration Example

Applied to the meshrr and meshrr-init containers in a pod:

```
encrypted_root_pw: NOLOGIN # Can be a legitimate encrypted root PW or can be an impossible hash
asn: "65000”
bgpgroups:
- name: MESHRR-MESH
  type: mesh
  source:
    sourcetype: dns
    hostname: meshrr # FQDN for svc required if not in same namespace
- name: MESHRR-CLIENTS
  type: subtractive # Prefixes in multiple external-subtractive groups must not overlap
  prefixes:
  - 192.168.166.0/24
  # For routeserver use case, an AS range is needed; we don't set this for RR use case.
  # asranges:
  # - 65001-65500
- name: MESHRR-UPSTREAM
  type: mesh
  source:
    sourcetype: dns
    hostname: meshrr.core.svc.cluster.local # FQDN required if svc not in same namespace
  max_peers: 2 # Limits to only connecting to 2 peers from this group
```

Health Checks

- Current examples: Liveness + readiness probes on port 179
  - Readiness probes gate adding the pod into service
  - Liveness probes restart the pod's containers upon failure
- Adding a health check sidecar container could enable checks on BGP neighborship formation or telemetric data

```
livenessProbe:
  failureThreshold: 3
  initialDelaySeconds: 15
  periodSeconds: 2
  successThreshold: 1
  tcpSocket:
    port: bgp
  timeoutSeconds: 3
readinessProbe:
  failureThreshold: 3
  initialDelaySeconds: 5
  periodSeconds: 2
  successThreshold: 2
  tcpSocket:
    port: bgp
  timeoutSeconds: 3
```

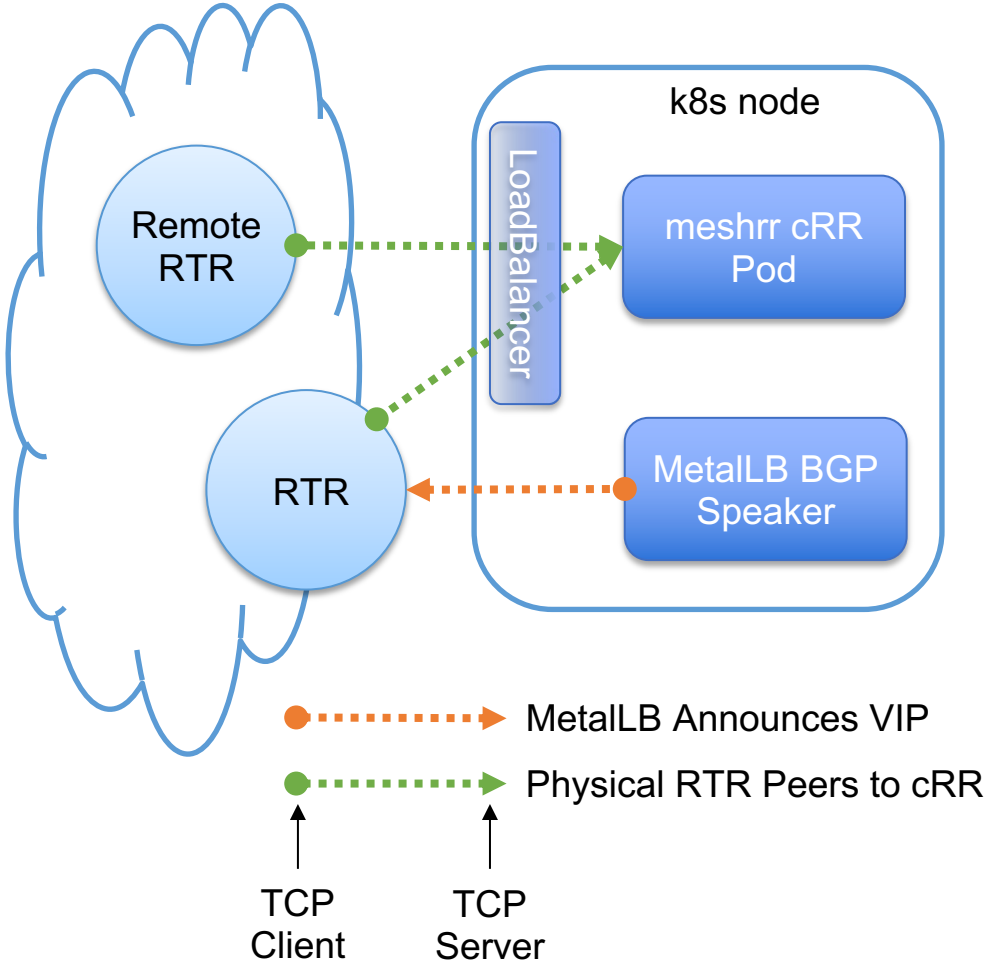
Client Connectivity

Outside BGP Client Connectivity

- MetalLB LoadBalancer services provide reachability to the pod
  - LoadBalancers provide dynamic connectivity from outside into k8s overlay to one or more pods.
  - Simplest k8s deployment: DNAT from outside to inside k8s cluster
- Alternative options, such as NodePorts, exist
  - Less dynamic than LoadBalancers

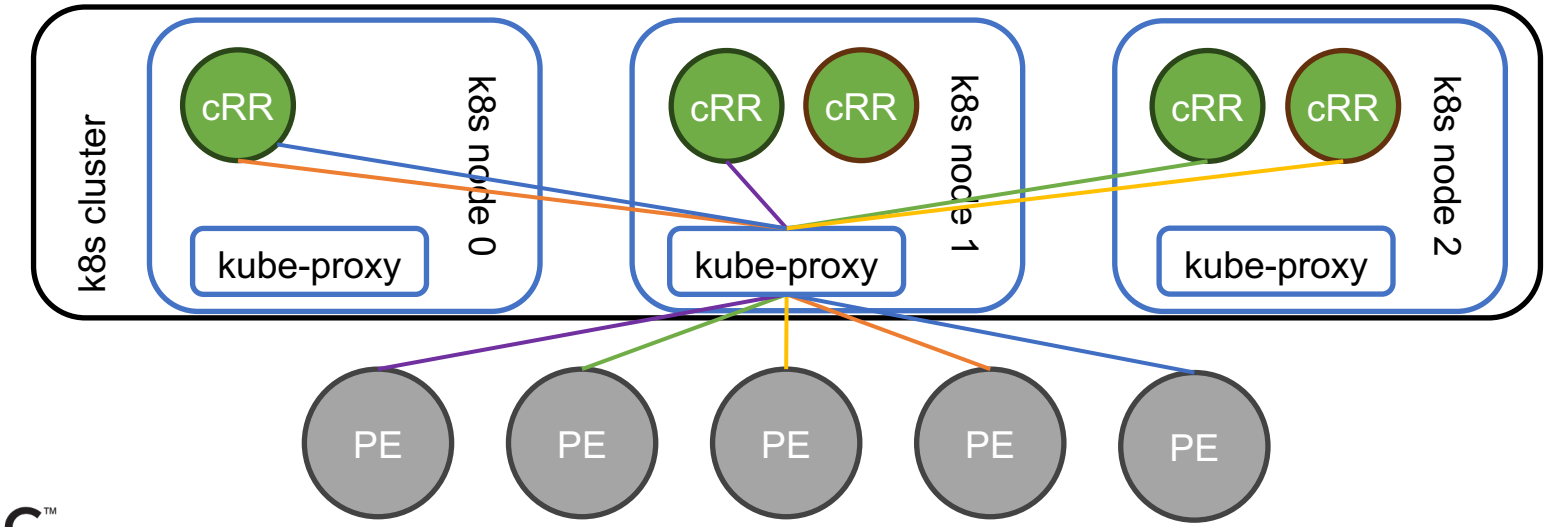
MetalLB: BGP Mode

- MetalLB peers via BGP from each k8s node to the directly connected physical router to announce VIP address(es) of the cRRs' LoadBalancer service(s).
- Multiple k8s nodes may permit traffic into the k8s cluster.
  - No state sync; can create churn based on physical network routing / ECMP



MetalLB: L2 Mode

- All traffic enters k8s cluster through single node
  - Single k8s node is bottleneck for network traffic
  - RRs can still be distributed across multiple pods / k8s nodes
- Eliminates churn due to routing changes on physical network – may be more appropriate if multiple cRR pods are being load balanced



externalTrafficPolicy: Cluster

- Distributes traffic to pods on any node in k8s cluster
- Source NATs RR client IP
- Pairs nicely with L2 Mode

```
user@R1> show bgp neighbor 172.19.1.1 | match "^ *Peer.*:"
Peer: 172.19.1.1+179 AS 65000  Local: 172.18.0.1+63221 AS 65000
  Peer ID: 10.42.4.5       Local ID: 172.18.0.1        Active Holdtime: 90

user@lothlorien-vm1:~/meshrr$ k get pods -o wide --field-selector status.podIP=10.42.4.5
NAME                          READY   STATUS    RESTARTS   AGE   IP           NODE
meshrr-lothlorien-a-7bdvm     2/2     Running   0          38m   10.42.4.5    lothlorien-vm3

user@lothlorien-vm1:~/meshrr$ k exec -it meshrr-lothlorien-a-7bdvm -c crpd - sh

# cli show bgp neighbor | egrep "^Peer|172.18.0.1" | grep -B1 172.18.0.1
Peer: 10.42.0.0+63146 AS 65000 Local: 10.42.4.5+179 AS 65000
  Peer ID: 172.18.0.1      Local ID: 10.42.4.5         Active Holdtime: 90
```

externalTrafficPolicy: Local

- Traffic from outside can only reach pods on entry node
- Eliminates Source NAT of RR client IP
- Pairs nicely with BGP mode
  - Route over the physical network directly to the k8s node hosting the cRR pod, rather than the k8s overlay.

```
RP/0/RP0/CPU0:R8#show bgp neigh 172.19.1.1 | i "router ID|host"
 Remote router ID 10.42.3.6
  Local host(configured): 172.18.0.8, Local port: 25563, IF Handle: 0x00000000
  Foreign host: 172.19.1.1, Foreign port: 179

user@lothlorien-vm1:~/meshrr$ k get pods -o wide --field-selector status.podIP=10.42.3.6
NAME                          READY   STATUS    RESTARTS   AGE   IP           NODE
meshrr-lothlorien-a-4afxb     2/2     Running   0          38m   10.42.3.6    lothlorien-vm2
user@lothlorien-vm1:~/meshrr$ k exec -it meshrr-lothlorien-a-4afxb -c crpd - sh

# cli show bgp neighbor 172.18.0.8 | egrep "Peer.*:.+\."
Peer: 172.18.0.8+25563 AS 65000 Local: 10.42.3.6+179 AS 65000
  Peer ID: 172.18.0.8       Local ID: 10.42.3.6          Active Holdtime: 90
```

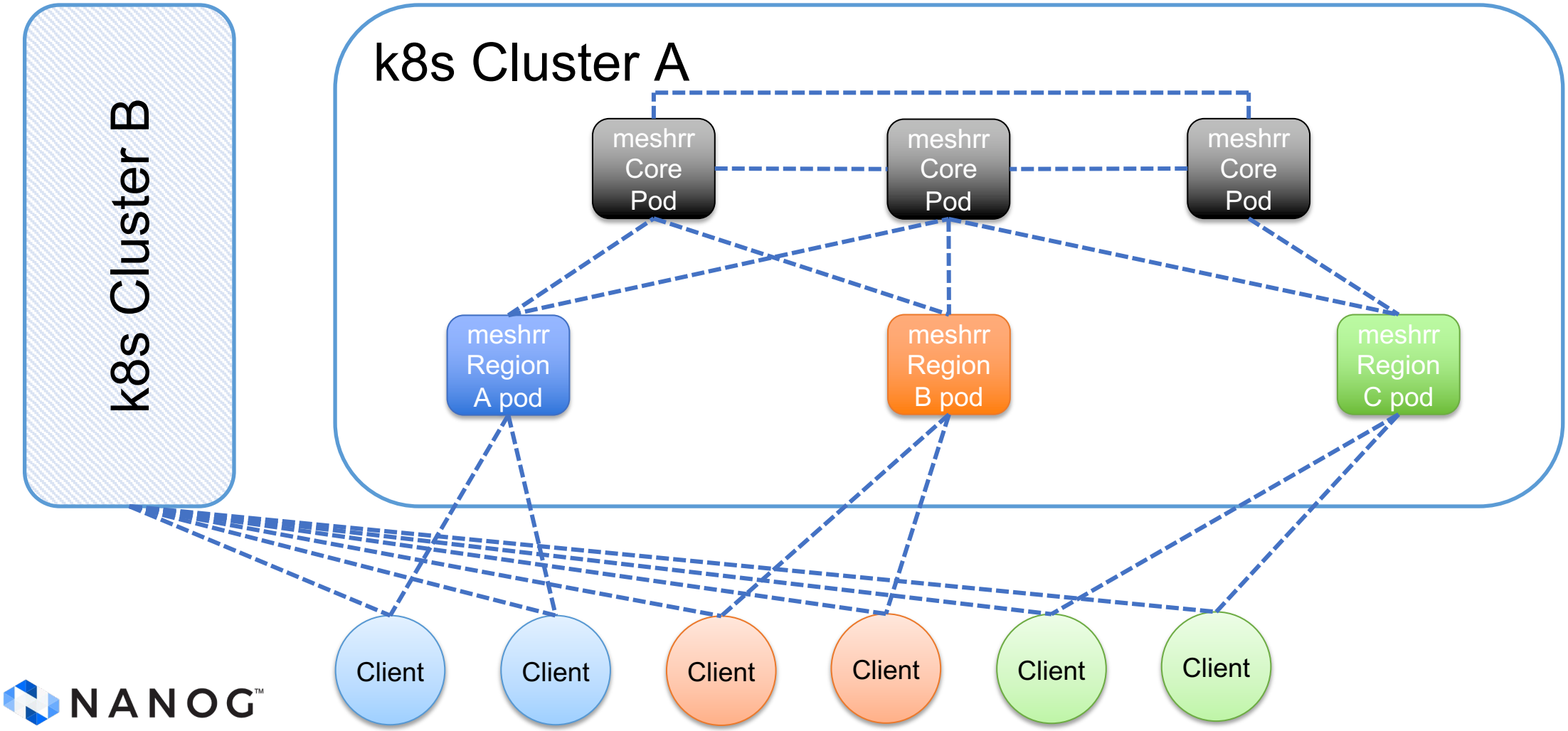
Use Cases

Basic Use Case Concepts

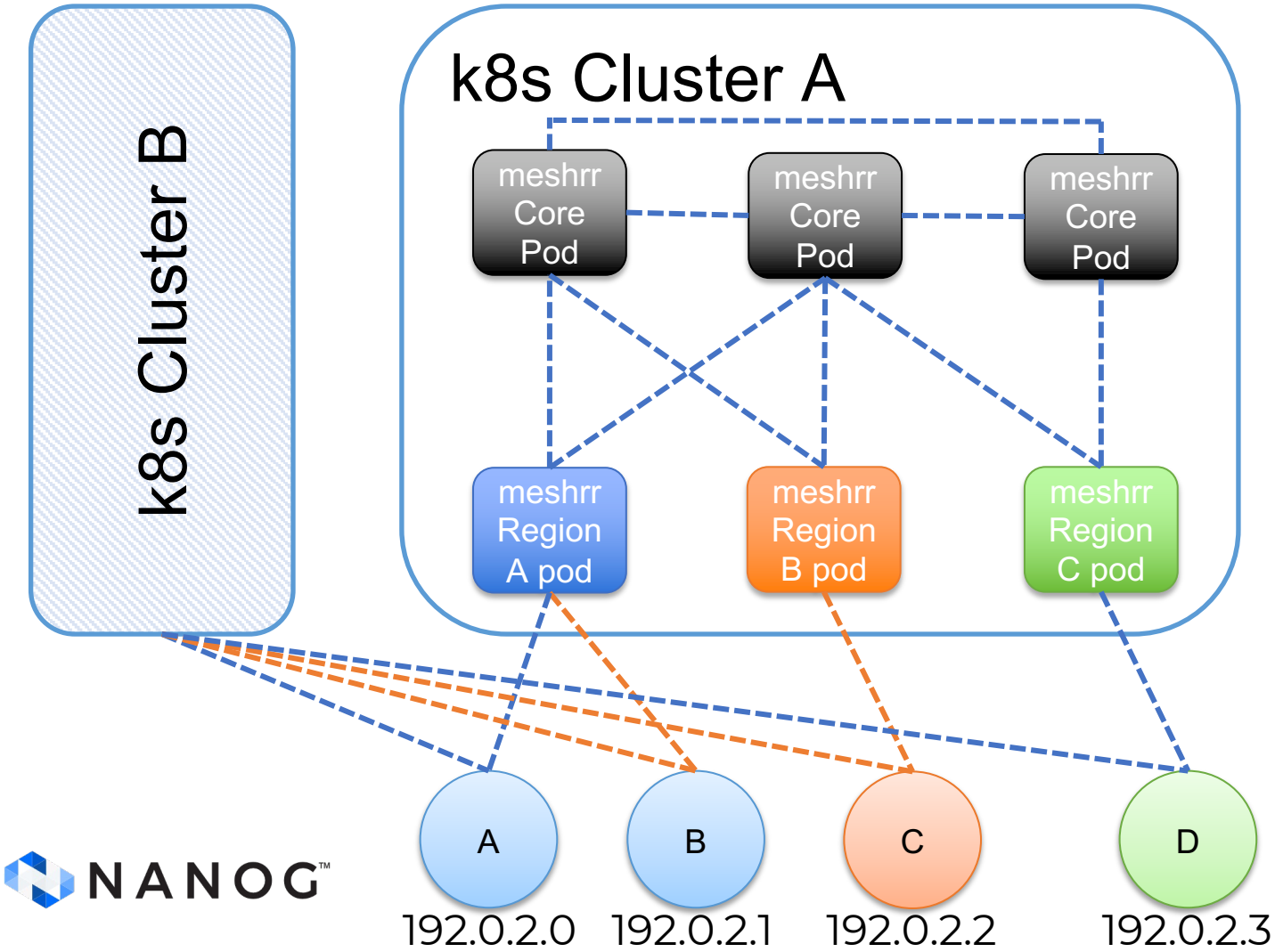
- DaemonSets: Replicate cRR on all nodes
  - E.g. could be used to offer “Anycast RRs”, assuming care taken to avoid churn from routing / ECMP changes.
- Statefulness and Persistence
  - Use StatefulSets and persistent volumes to retain configuration
  - Defeats some advantages if each cRR requires individual care
- Networking
  - LoadBalancers vs NodePorts
  - Overlay k8s network vs in-line k8s network

UC2: Regional Hierarchical cRRs

Core: **mesh** group between Core cRRs
subtractive group allows regions' cRRs to connect
Each Region: **mesh** group with max-peers 2 to Core cRRs
subtractive group allows clients to connect



UC3: Multi-Group Hierarchical cRR



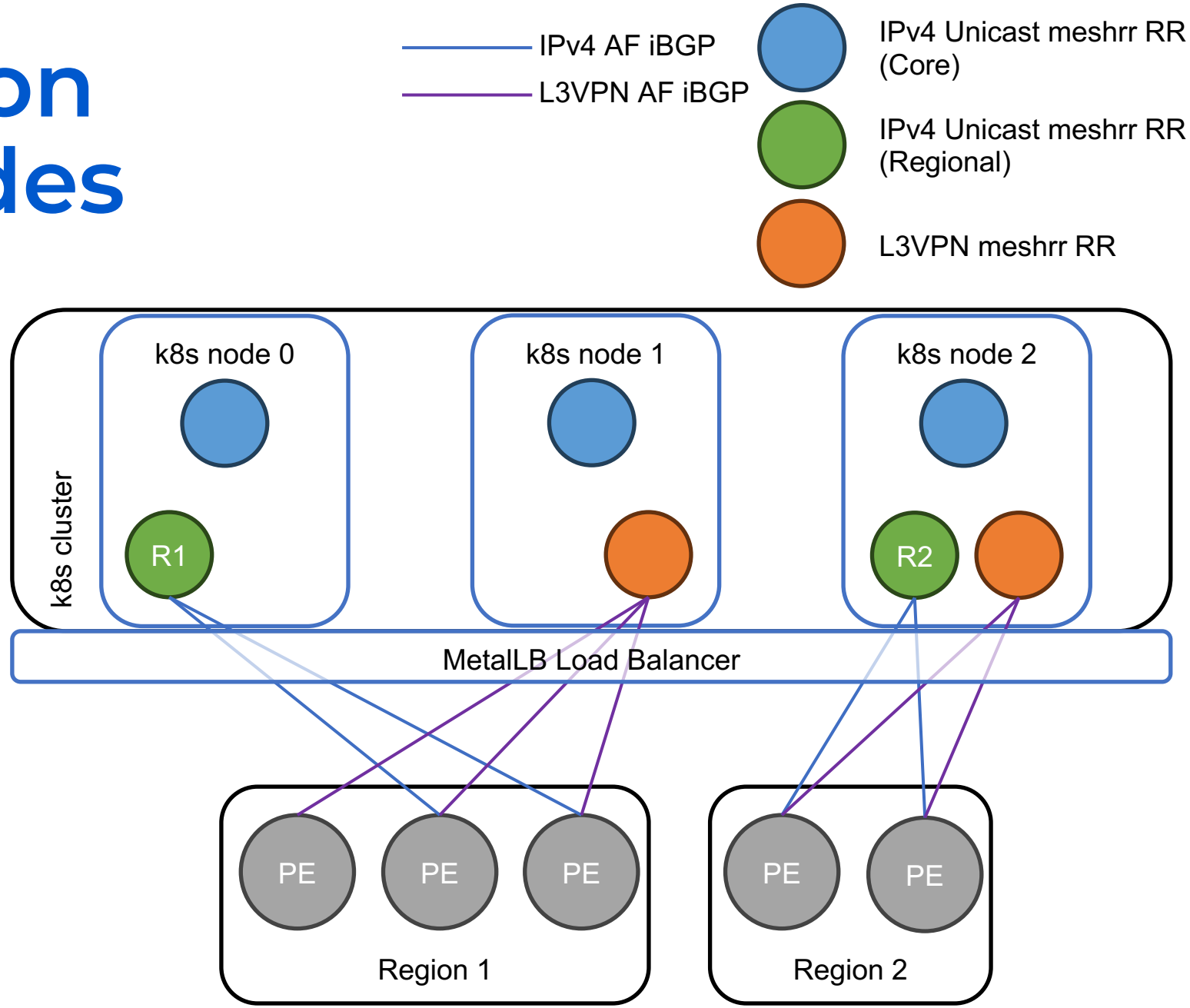
Default route only

k8s Manual Service Definition:

```
---
kind: "Service"
apiVersion: "v1"
metadata:
  name: "meshrr-defaultonly"
spec:
  clusterIP: None
  ports:
  - name: "bgp"
    protocol: "TCP"
    port: 179
    targetPort: 179
---
kind: "Endpoints"
apiVersion: "v1"
metadata:
  name: "meshrr-defaultonly"
subsets:
  - addresses:
    - ip: "192.0.2.1"
    - ip: "192.0.2.2"
    ports:
    - port: 179
      name: "bgp"
```

UC4: cRRs on Shared Nodes

Multiple, distinct cRRs can exist for different purposes and address families on the same k8s nodes



Final Thoughts

Participate

- Lab the solution
- Develop
  - BGP session-based Horizontal Pod Autoscaling support
  - Additional health checks with dedicated sidecar
  - Prestop hook to gracefully shut down BGP sessions
  - Refine examples to use multiple clusters instead of A/B sides
  - Peer discovery options other than DNS
- Discuss or create additional use cases and examples

https://github.com/Juniper/meshrr

Q: Is meshrr for you?

- A: meshrr is a concept and a starting place
- A: The concept is applicable in scenarios which:
  - require scaling to many RRs
  - creating meshes or hierarchies of more than a few RRs
- Benefits:
  - Multiple cRRs on shared nodes in resource-managed clusters
  - Consistent configuration and behavior
  - Simplified horizontal scaling and rolling updates
  - Dynamic discovery of neighbors
  - Auto-healing capabilities

Thank you

For more information or to participate:

https://github.com/Juniper/meshrr