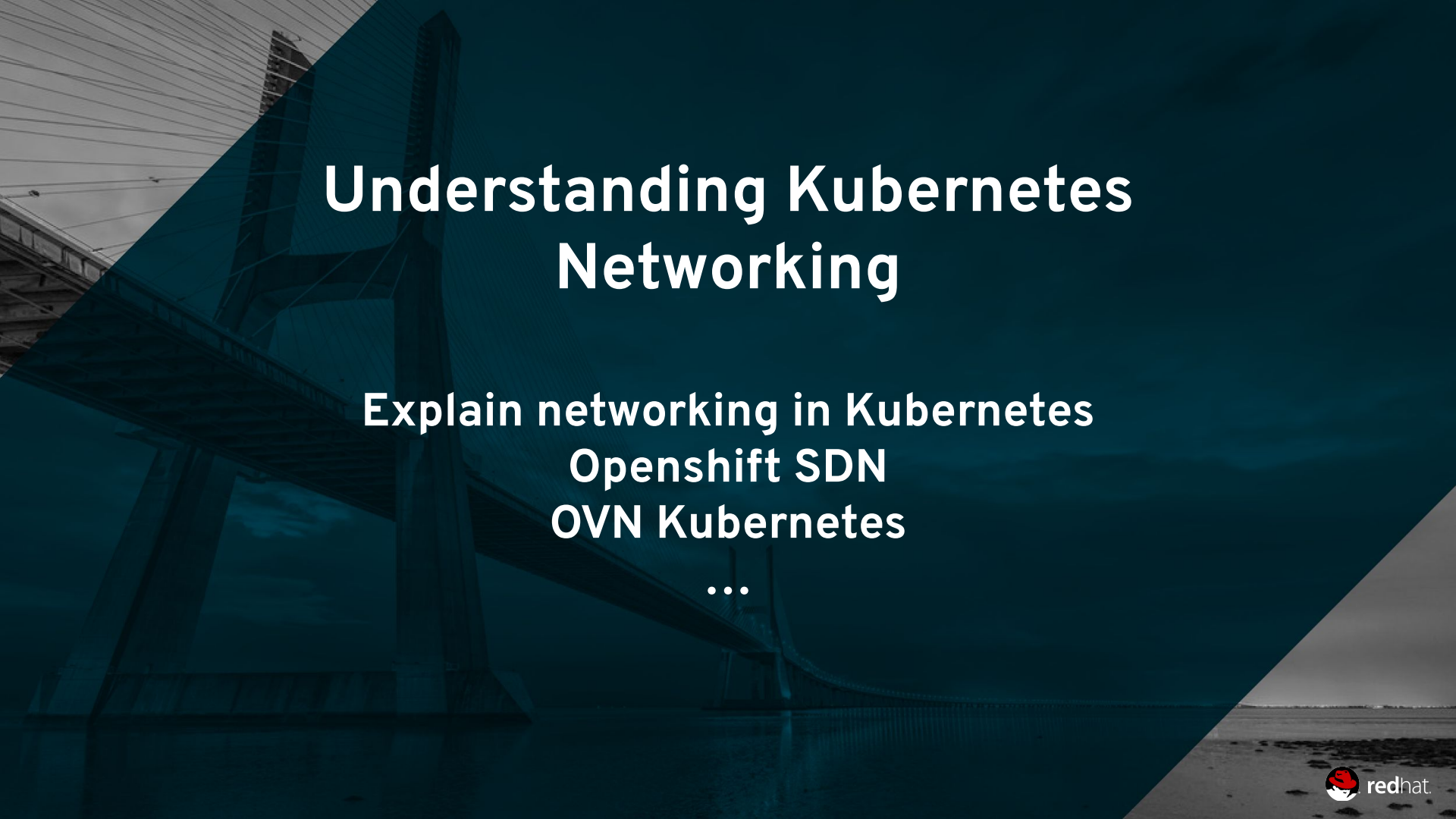
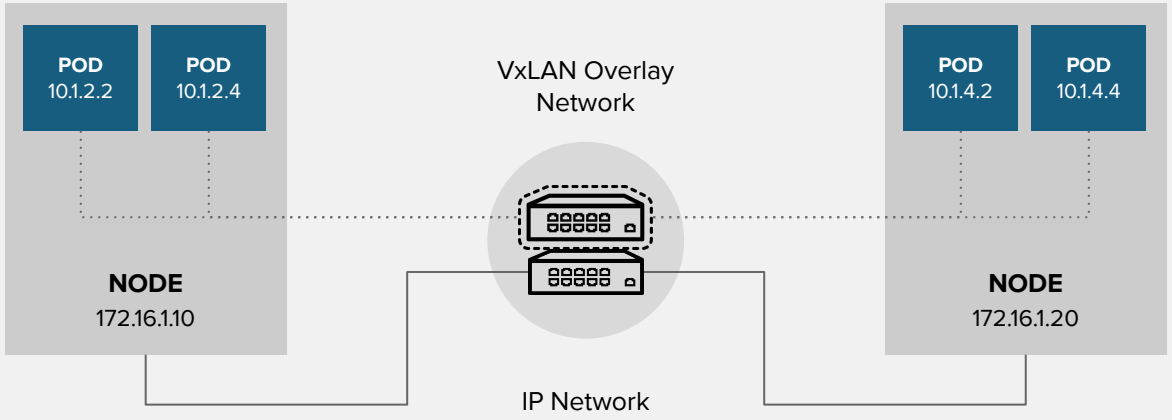
OPENSHIFT NETWORKING


Understanding Kubernetes Networking
Explain networking in Kubernetes
Openshift SDN
OVN Kubernetes
...
redhat.

OPENSHIFT NETWORKING

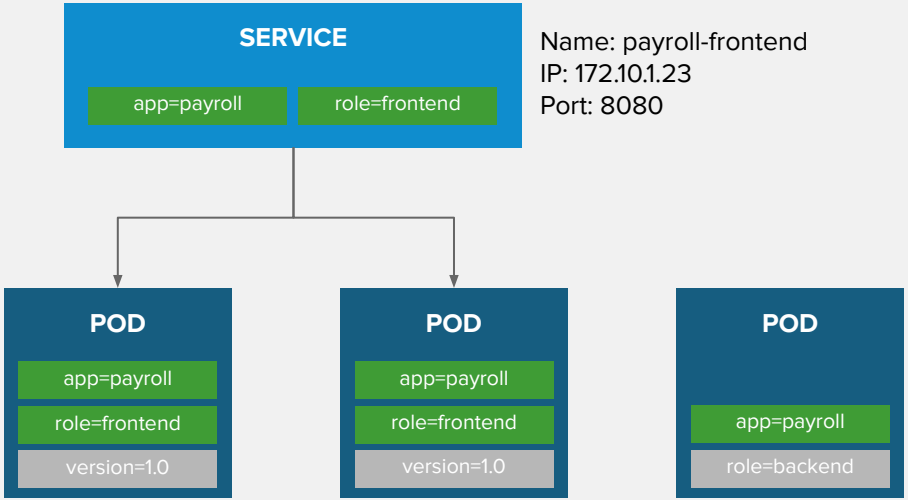
- DNS : Provide name resolution for each Pod
- Service : Virtual access point targeting several Pods
  - Access by IP or DNS, internally or externally
  - Load balance flows to target Pods
  - Dynamic selection of target Pods (dynamic discovery)
- Flow control :
  - Control flows between Pods/Namespaces (Multi-tenancy)
  - Control flows entering/leaving Kubernetes
- External Flows :
  - How to access Pods from outside Kubernetes
  - How to filter on firewalls, flows coming from Kubernetes
  - Give direct access to node network

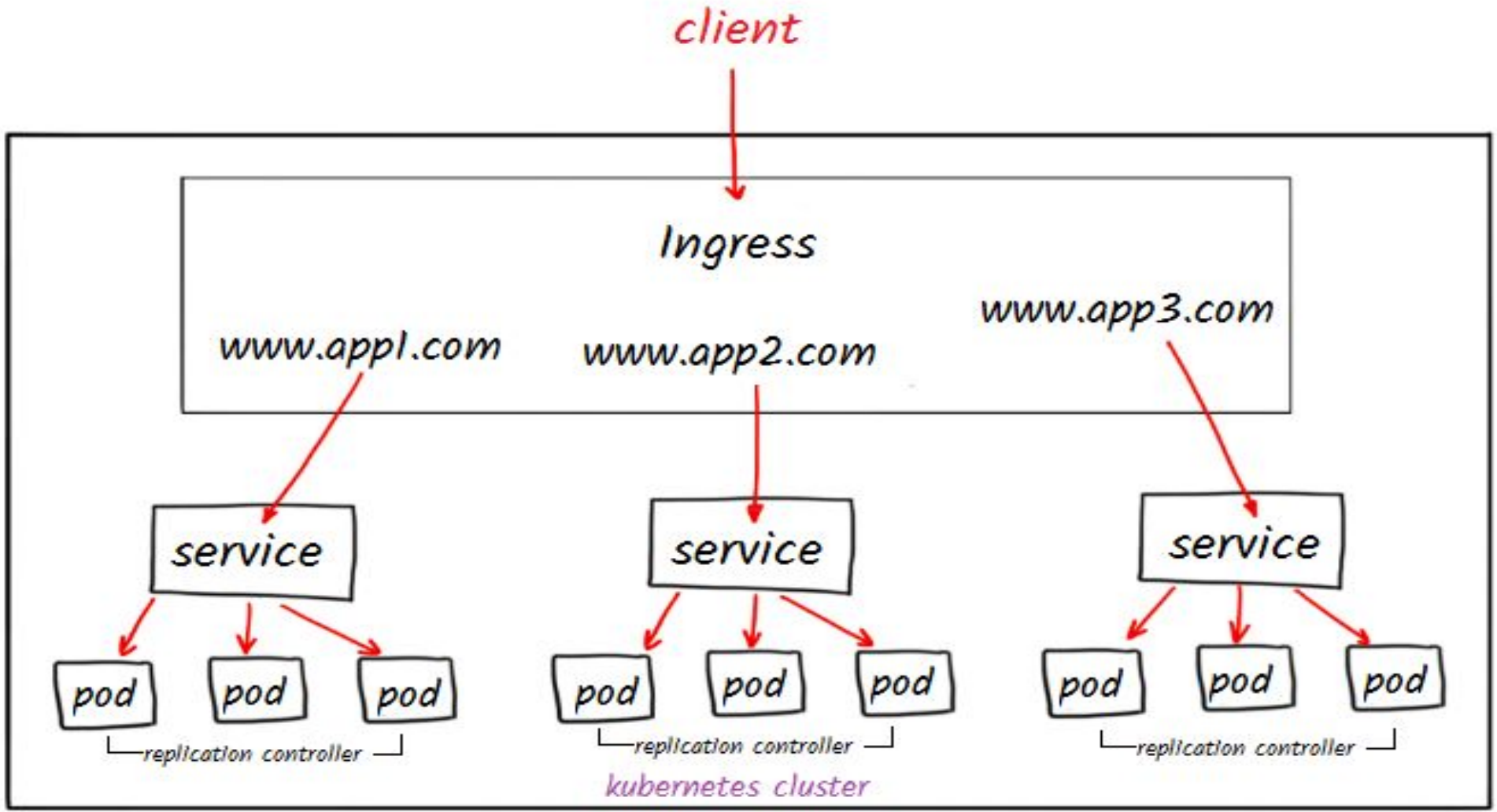
POD to POD COMMUNICATION



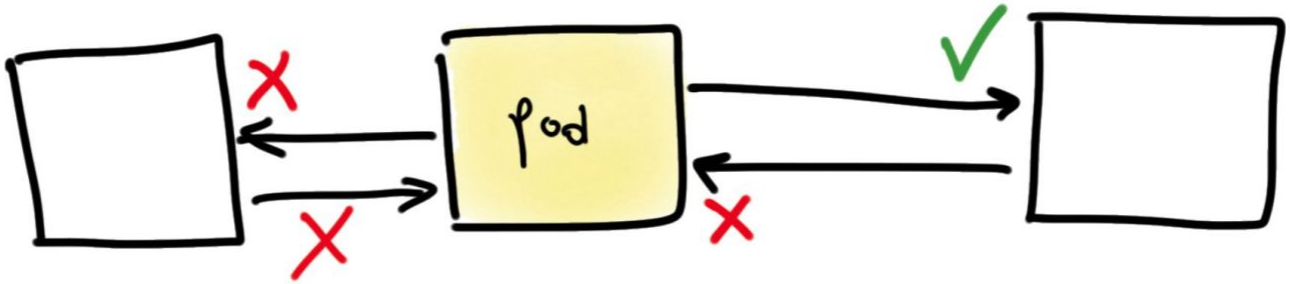
Like a VPN built around Nodes and Pods

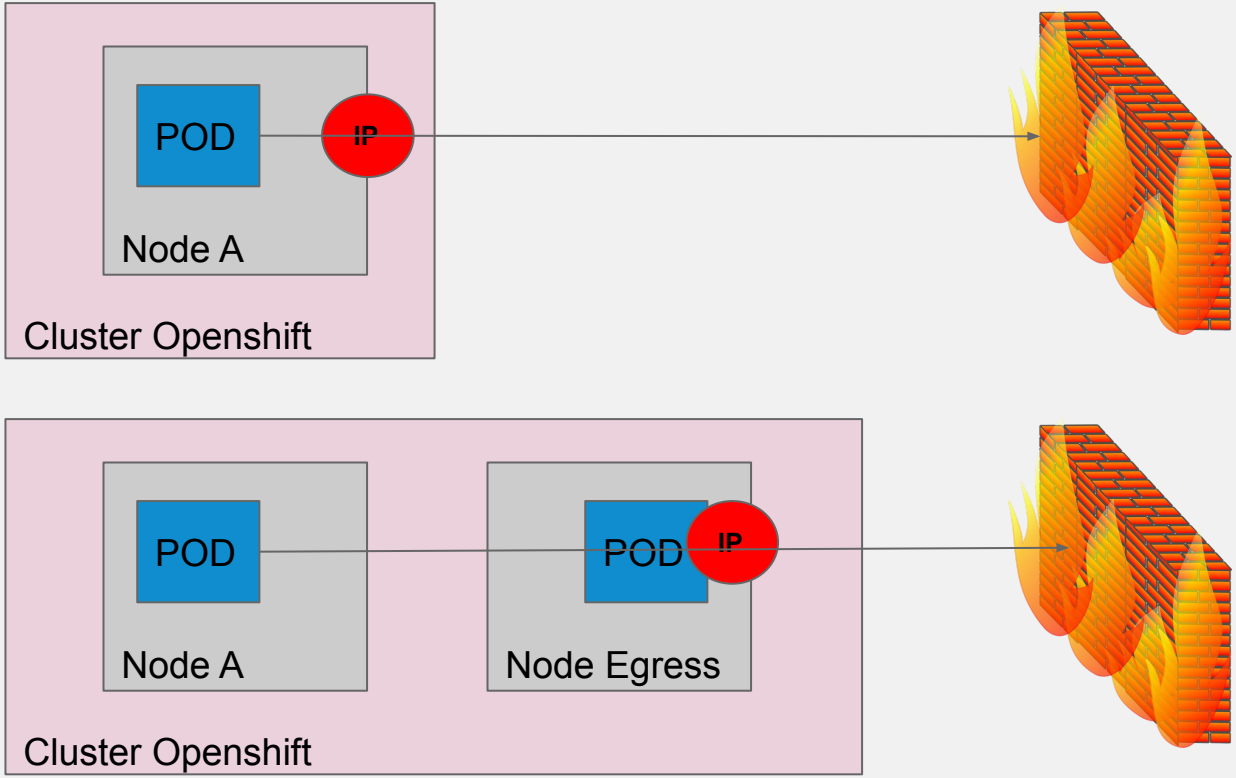
BUILT-IN SERVICE DISCOVERY
INTERNAL LOAD-BALANCING




client
Ingress
www.app1.com
www.app2.com
www.app3.com
service
service
service
pod
pod
pod
pod
pod
pod
pod
pod
pod
replication controller
replication controller
replication controller
kubernetes cluster

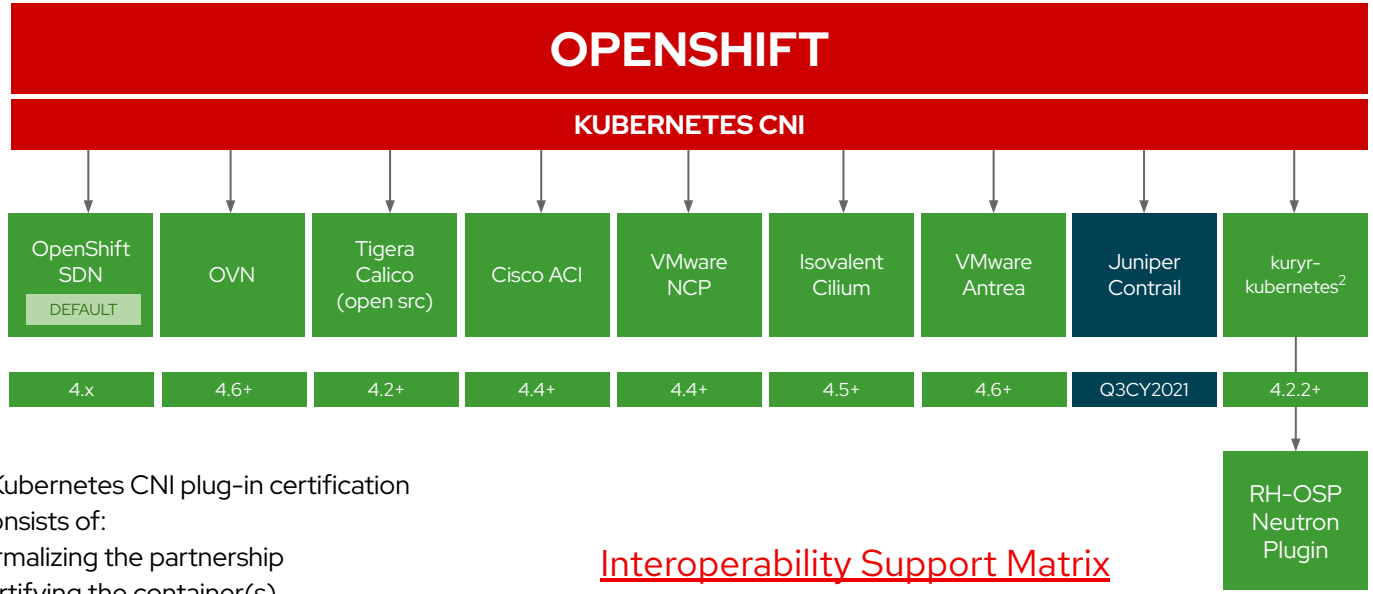
network policies control traffic from/to pods





How to filter flows outside Openshift

OpenShift Networking Plug-ins



3rd-party Kubernetes CNI plug-in certification primarily consists of:

1. Formalizing the partnership
2. Certifying the container(s)
3. Certifying the Operator
4. Successfully passing the same Kubernetes networking conformance tests that OpenShift uses to validate its own SDN

Interoperability Support Matrix

Fully Supported | Tech Preview | Cert In-Progress


DNS inside the Cluster

DNS: Pod Name Resolution

1. Container is configured with:

   ```
   dnsPolicy: ClusterFirst
   ```

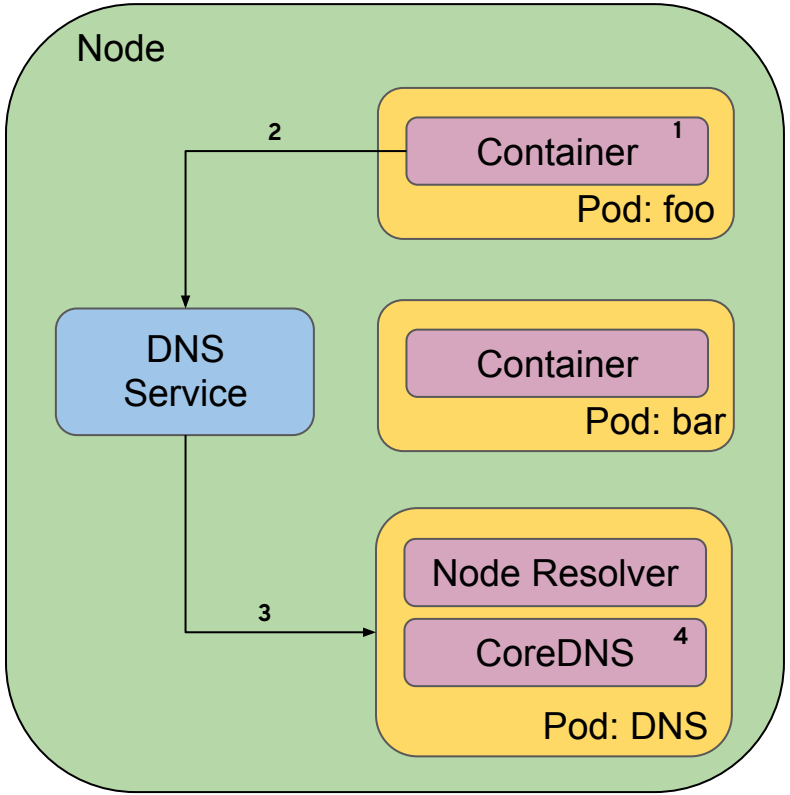
   Resulting in containers with /etc/resolv.conf:

   ```
   nameserver 172.30.0.10
   ```
   *

   Container “foo” looks-up “bar” in local cache. “bar” doesn’t exist, so “foo” uses /etc/resolv.conf to figure out how to resolve “bar”.

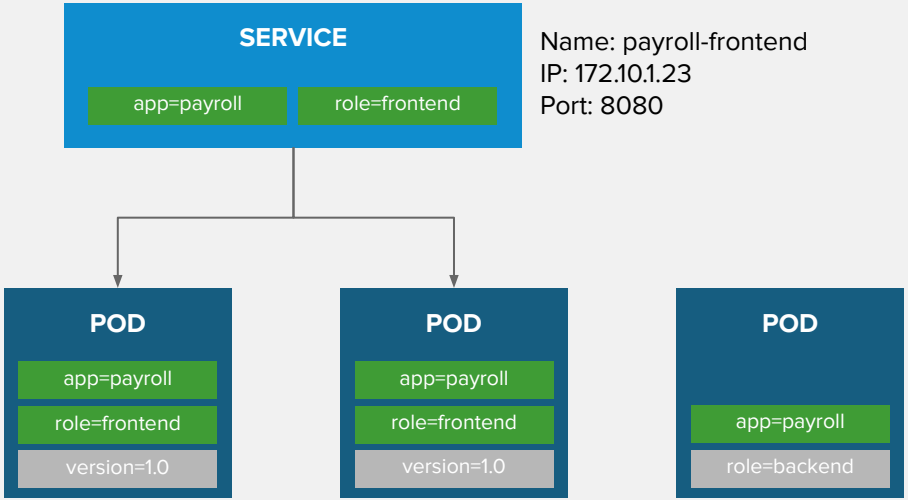
2. Container “foo” queries nameserver 172.30.0.10 for “bar”.
3. The query gets proxied to the DNS pod
4. CoreDNS resolves bar to “10.128.0.100”.

* Default serviceNetwork is 172.30.0.0/16



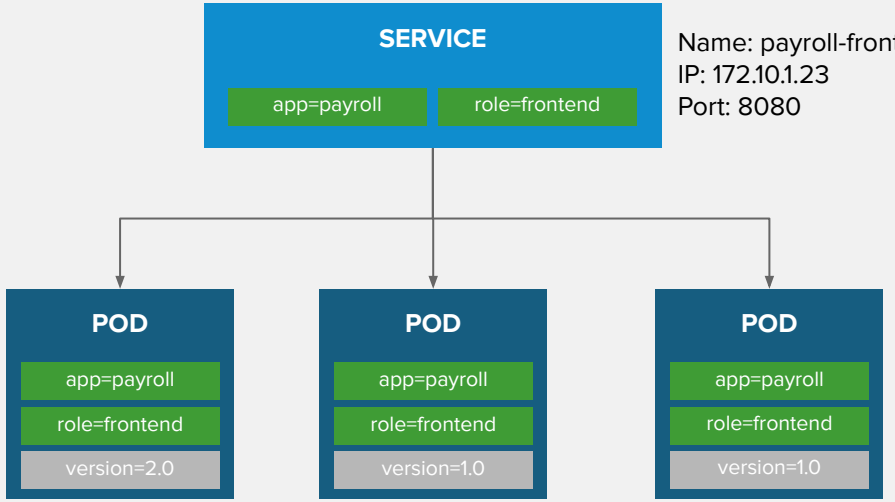

Load balancing inside the Cluster

BUILT-IN SERVICE DISCOVERY
INTERNAL LOAD-BALANCING



BUILT-IN SERVICE DISCOVERY INTERNAL LOAD-BALANCING

- When a new pod is deployed on the cluster with the same labels as the service selector, it is automatically added to the list of endpoints load-balanced by the service. If one of the pod gets removed, the service immediately removes it from the list of endpoints and does not send traffic to it anymore.
- Service selector allows load-balancing between multiple versions of the application pods (version=1.0 and version=2.0) in order to achieve A/B testing and other deployment patterns out-of-the-box



- On the slide, the service is configured to discover pods and load-balance traffic between pods that have both app=payroll and role=frontend labels set. Since both v1 and v2 of the frontend pods are deployed and have those labels set, ⅓ of the traffic would be to sent to the frontend v2 to achieve canary release or A/B testing for example.


Network Micro-segmentation
redhat.

OPENSHIFT Network Multi-tenancy

FLAT NETWORK (Default) : OVS

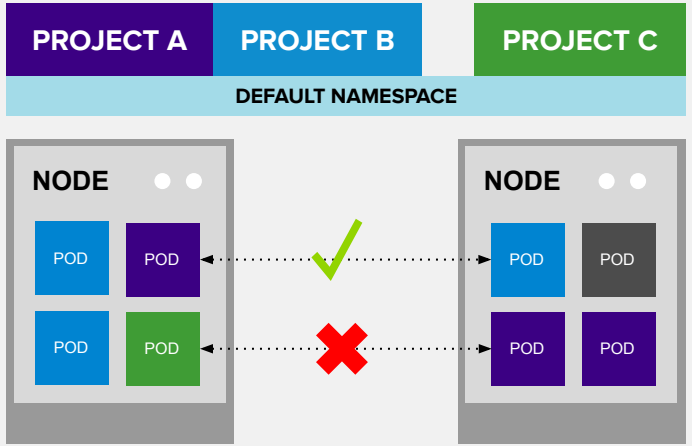
- All pods can communicate with each other across projects

MULTI-TENANT NETWORK: OVS

- Project-level network isolation
- Multicast support
- Egress network policies

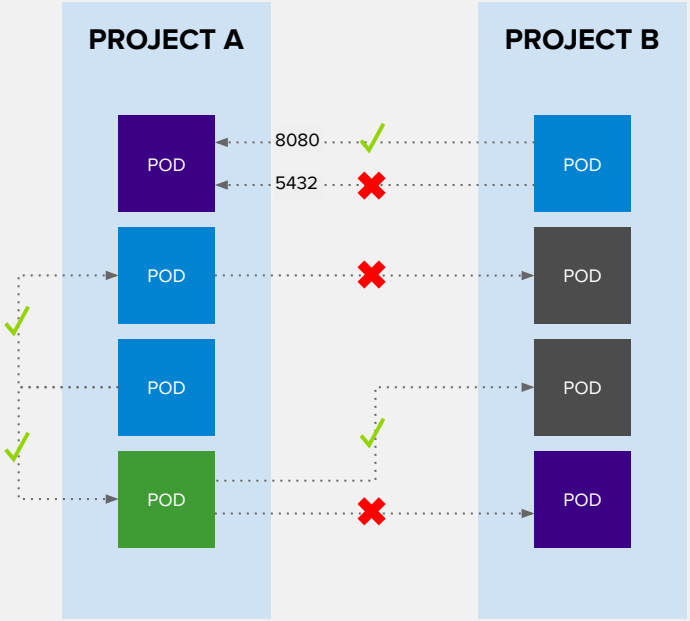
NETWORK POLICY : OVS+OVN

- Granular policy-based isolation



Multi-Tenant Network

OPENSHIFT SDN - NETWORK POLICY



Example Policies

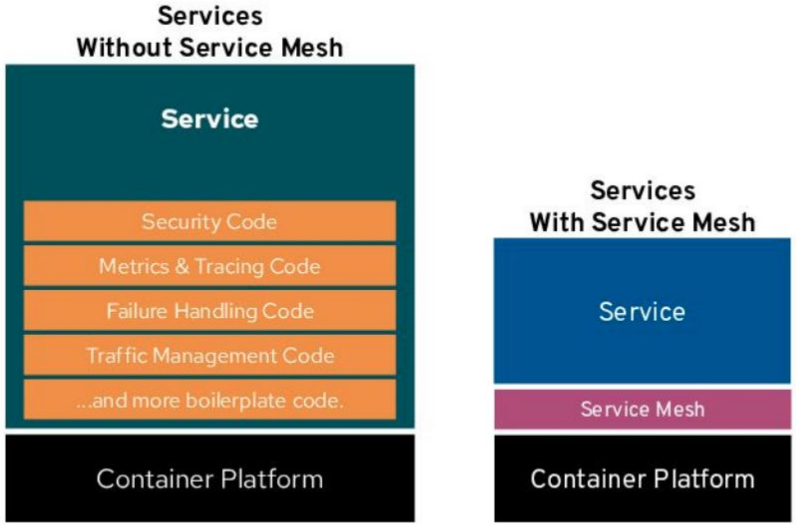
- Allow all traffic inside the project
- Allow traffic from green to gray
- Allow traffic to purple on 8080

```
apiVersion: extensions/v1beta1
kind: NetworkPolicy
metadata:
  name: allow-to-purple-on-8080
spec:
  podSelector:
    matchLabels:
      color: purple
  ingress:
  - ports:
    - protocol: tcp
      port: 8080
```

Why Service Mesh?

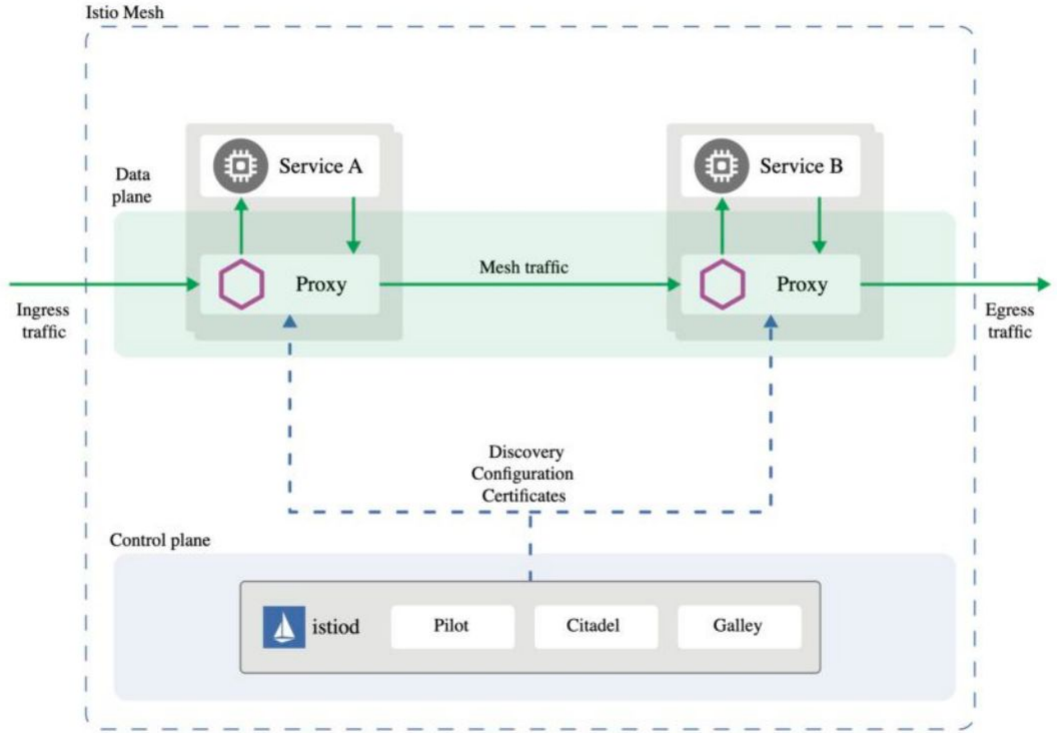
An Abstraction for Microservice Challenges

- Service Mesh solve distributed systems challenges at **a common infrastructure layer.**
- This reduces boilerplate code and copy/paste errors across services.
- Enforces common policies across all services.
- Removes the obligation to implement cross cutting concerns from developers.



Red Hat

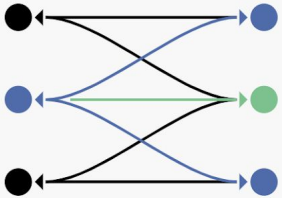
Istio Service Mesh - Architecture



Red Hat

Istio Service Mesh

A modern way to manage the complexity of microservice applications

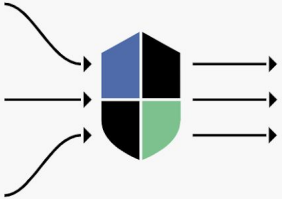


Connect

Intelligently control the flow of traffic and API calls between services, conduct a range of tests, and upgrade gradually with red/black deployments.

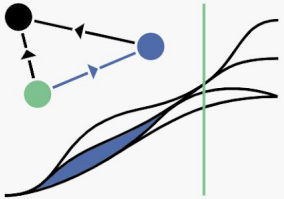
Secure

Automatically secure your services through managed authentication, authorization, and encryption of communication between services.



Control

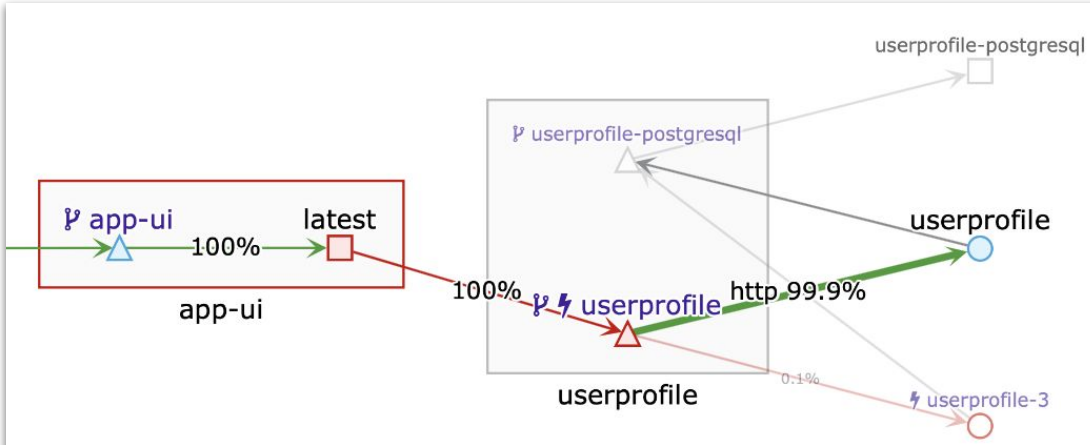
Apply policies and ensure that they're enforced, and that resources are fairly distributed among consumers.

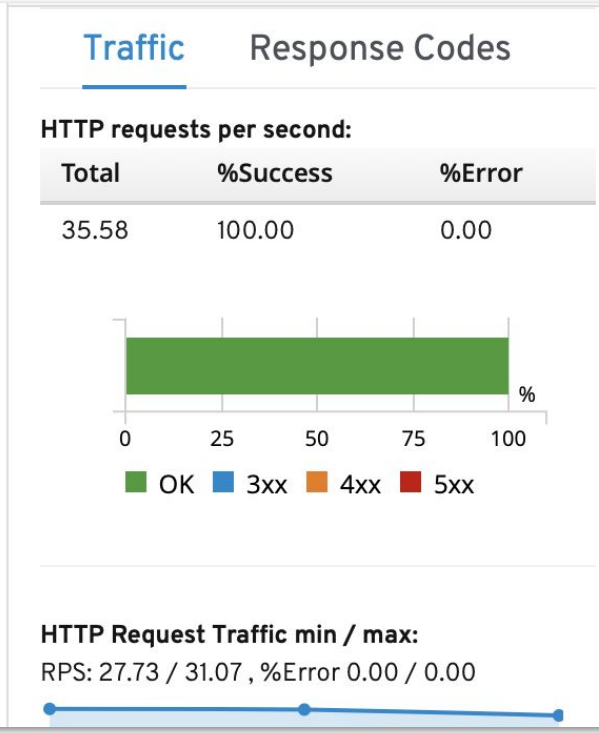


Observe

See what's happening with rich automatic tracing, monitoring, and logging of all your services.

Handling Partial Failures with the Service Mesh



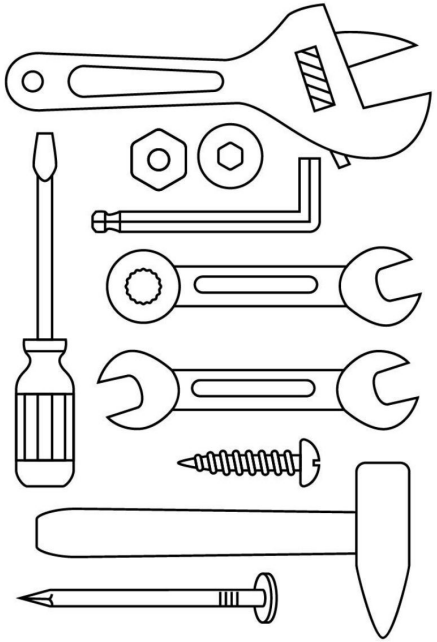



```
26      trafficPolicy:
27        connectionPool:
28          http:
29            http1MaxPendingRequests: 1
30            maxRequestsPerConnection: 1
31        outlierDetection:
32          consecutiveErrors: 1
33          interval: 1s
34          baseEjectionTime: 10m
35          maxEjectionPercent: 100
```

YAML configurable,
(Kubernetes native)

Red Hat

Features needed to meet the challenges of microservices



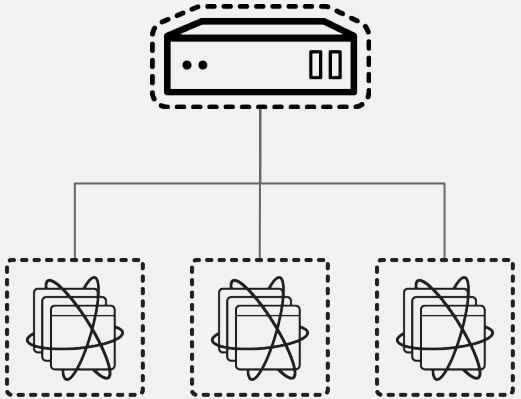
SERVICE DISCOVERY
CIRCUIT BREAKING AND BULKHEADS
RATE LIMITING
MIRRORING / TRAFFIC SHIFTING
VERSION BASED ROUTING
AUTOSCALING
STAGED ROLLOUTS
CANARY DEPLOYMENTS
BLUE/GREEN DEPLOYMENTS
DISTRIBUTED TRACING
VISUAL SERVICE HEALTH
DISTRIBUTED LOGGING
COLLECTING AND VISUALIZING METRICS
CHAOS ENGINEERING
SERVICE SECURITY
CONTAINER BUILD AUTOMATION

Table Stakes for doing microservices

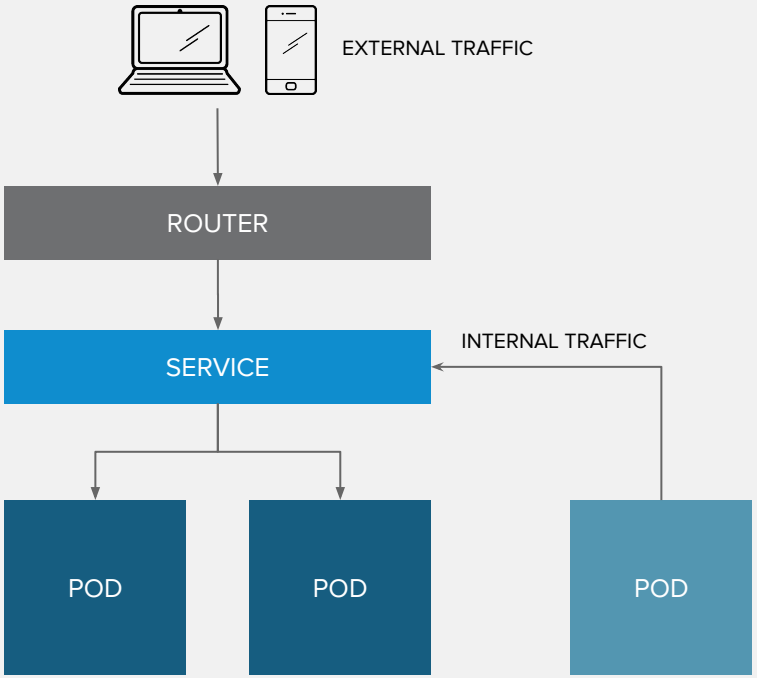

Getting Traffic In the Cluster
redhat.

ROUTING AND EXTERNAL LOAD-BALANCING

- Pluggable routing architecture
  - HAProxy Router
  - F5 Router
  - NGINX
- Multiple-routers with traffic sharding
- Router supported protocols
  - HTTP/HTTPS
  - WebSockets
  - TLS with SNI
- Non-standard ports via cloud load-balancers, external IP, and NodePort



ROUTE EXPOSES SERVICES EXTERNALLY

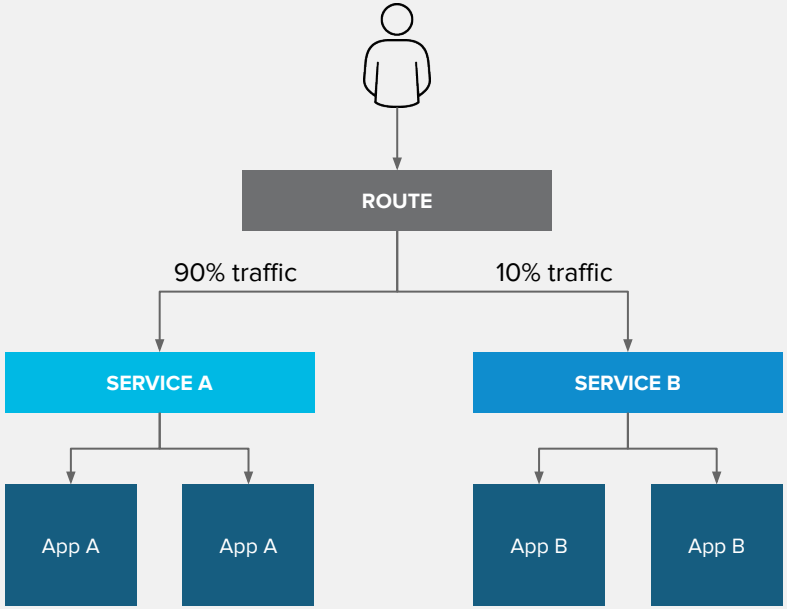


Routes vs Ingress

Feature	Ingress	Route
Standard Kubernetes object	X	
External access to services	X	X
Persistent (sticky) sessions	X	X
Load-balancing strategies (e.g. round robin)	X	X
Rate-limit and throttling	X	X
IP whitelisting	X	X
TLS edge termination	X	X
TLS re-encryption	X	X
TLS passthrough	X	X
Multiple weighted backends (split traffic)		X
Generated pattern-based hostnames		X
Wildcard domains		X

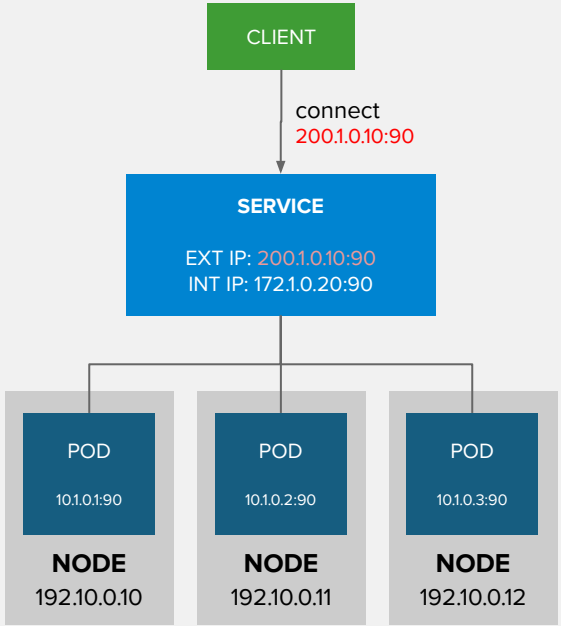
ROUTE SPLIT TRAFFIC

Split Traffic Between Multiple Services For A/B Testing, Blue/Green and Canary Deployments



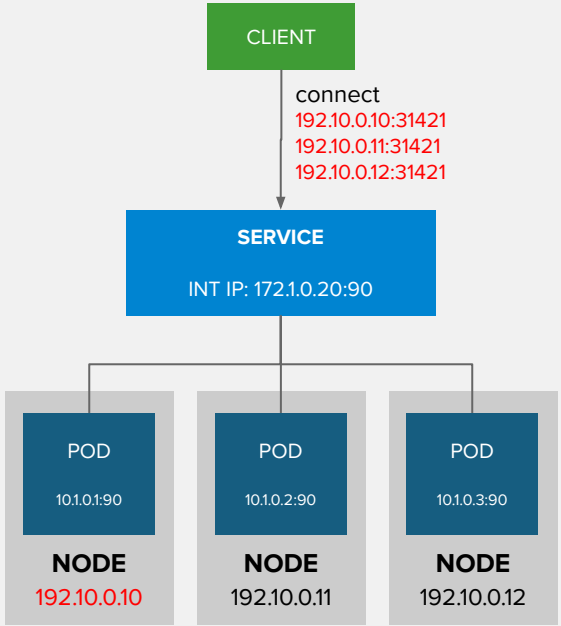
EXTERNAL TRAFFIC TO A SERVICE ON ANY PORT WITH INGRESS

- Access a service with an external IP on any TCP/UDP port, such as
  - Databases
  - Message Brokers
- Automatic IP allocation from a predefined pool using Ingress IP Self-Service
- IP failover pods provide high availability for the IP pool



EXTERNAL TRAFFIC TO A SERVICE ON A RANDOM PORT WITH NODEPORT

- NodePort binds a service to a unique port on all the nodes
- Traffic received on any node redirects to a node with the running service
- Ports in 30K-60K range which usually differs from the service
- Firewall rules must allow traffic to all nodes on the specific port



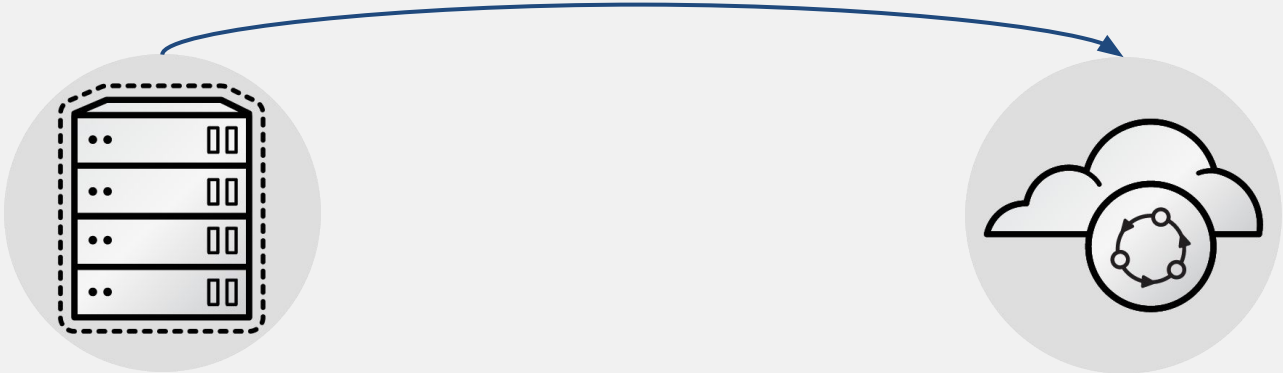

Getting Traffic Out of the Cluster

Getting Traffic Out of the Cluster

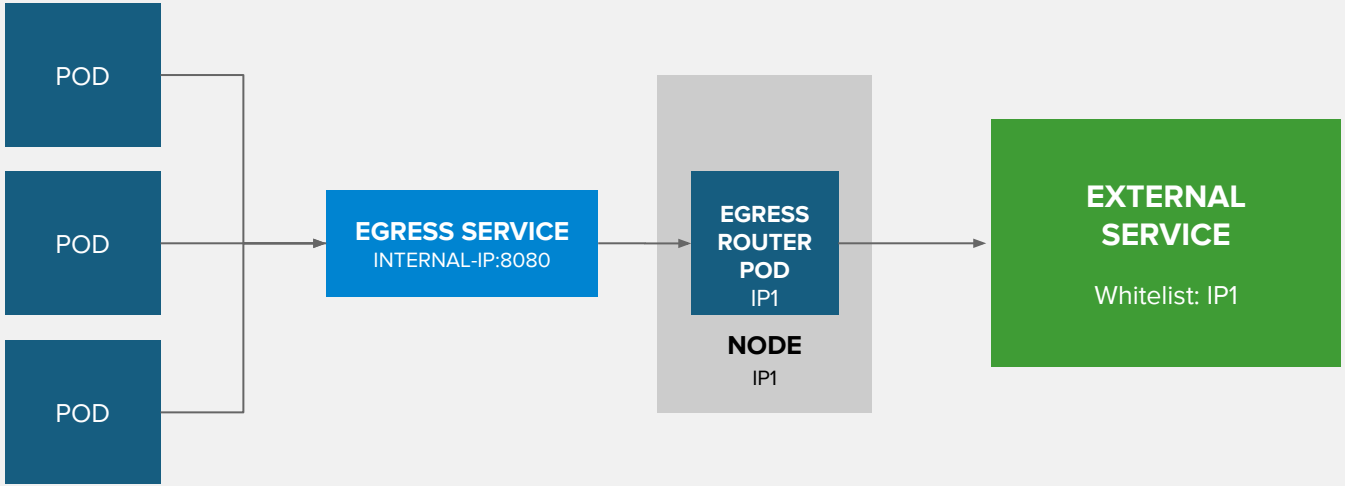
- Default: Traffic is NAT'd to the host IP of the Node running the Pod

Getting Traffic Out of the Cluster

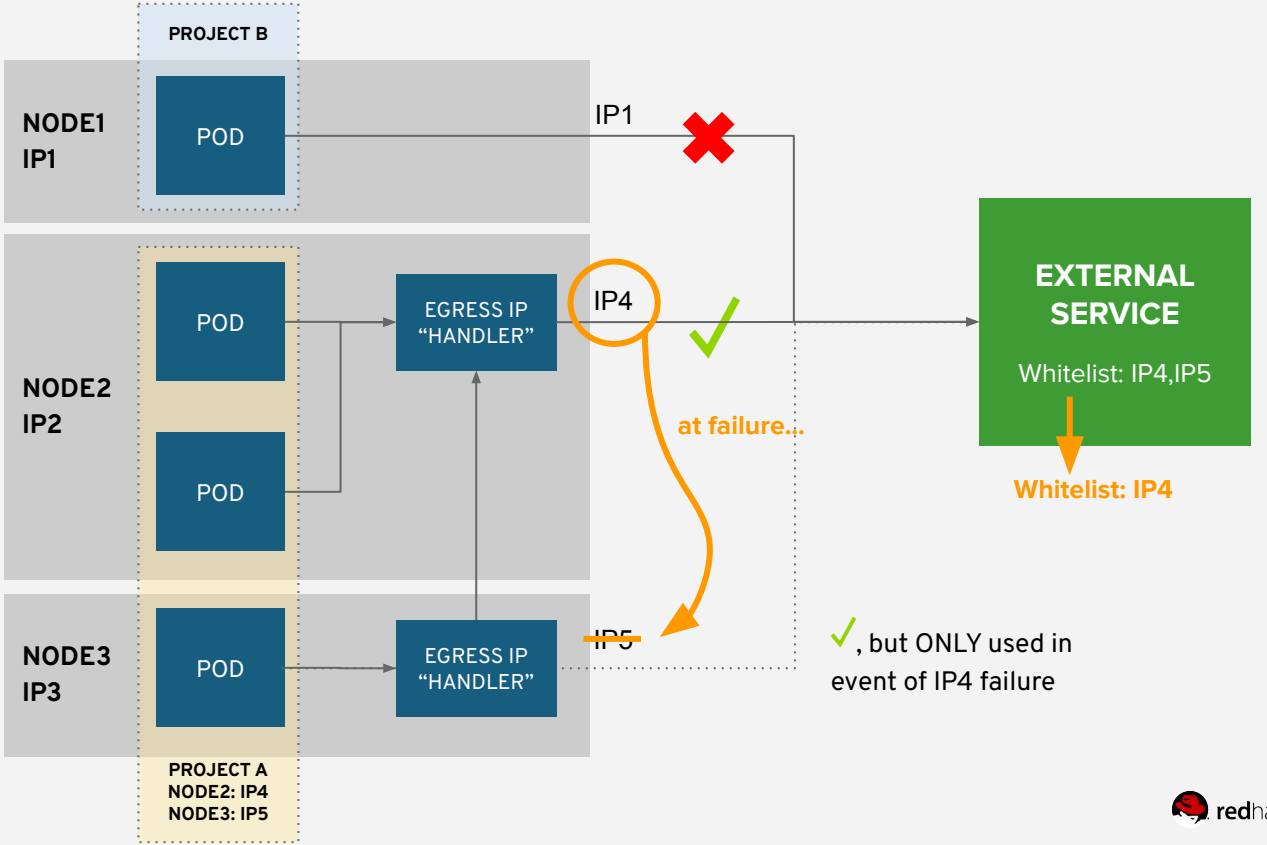
- Can control egress traffic in two ways
  - **Firewall** - using egress firewall allows you to enforce the acceptable outbound traffic - Still NAT'd
  - **Router** - An egress router allows you to create identifiable services to send traffic to a specific destination ensuring the destination treat as if originating from a known source



CONTROL OUTGOING TRAFFIC SOURCE IP WITH EGRESS ROUTER



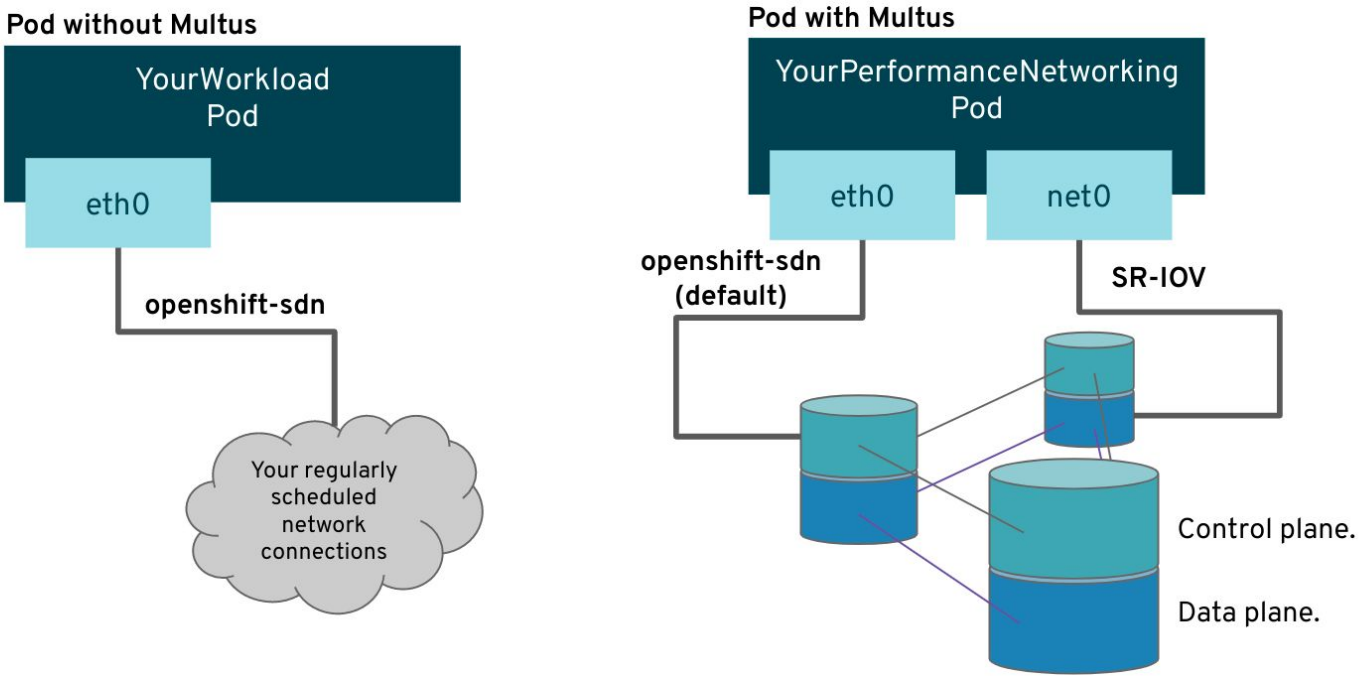
Fully Automatic, HA Namespace-Wide Egress IP



redhat.

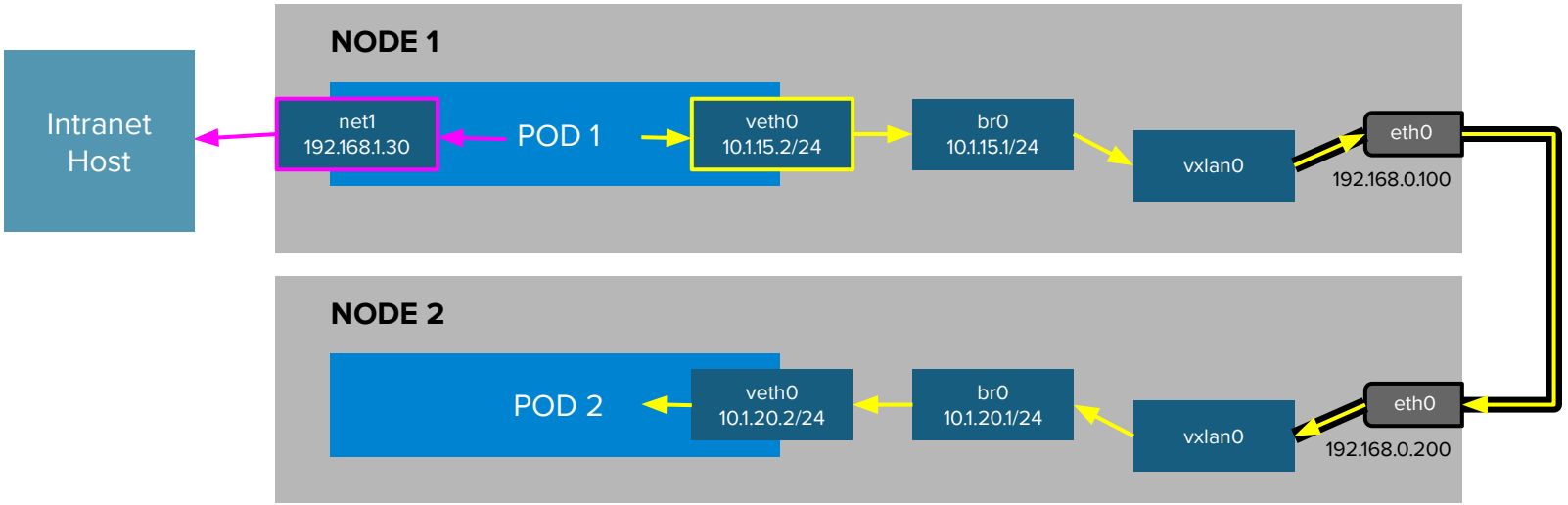
Using direct access to a host network

WHAT MULTUS DOES



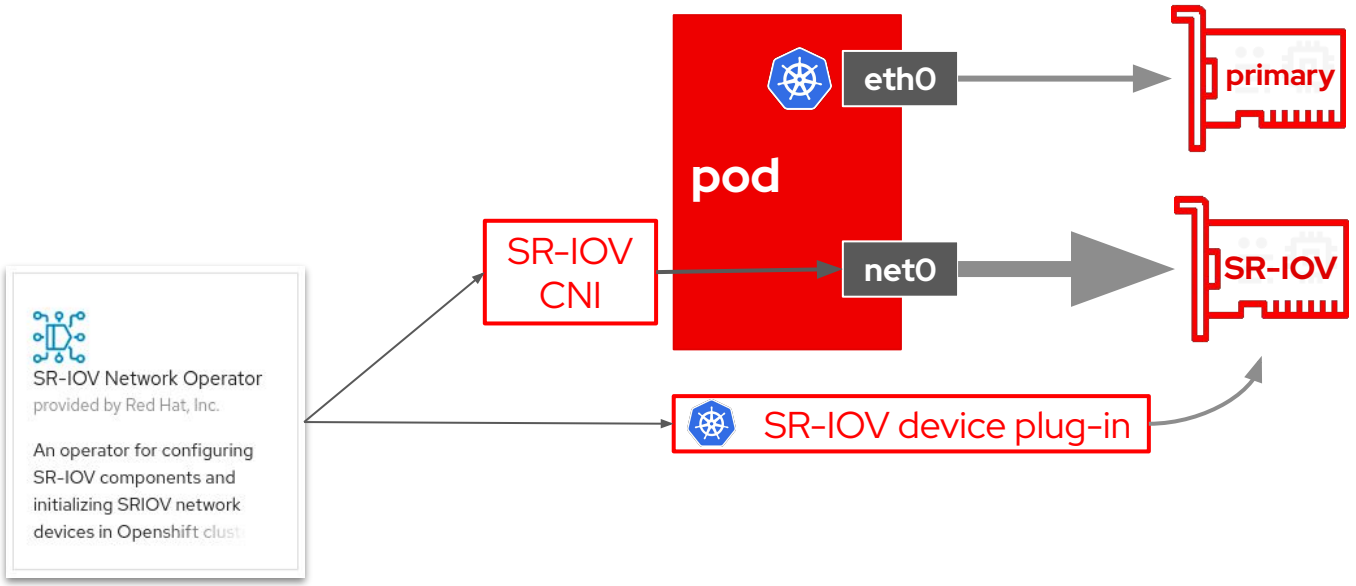
Red Hat

Multus Packet Flow



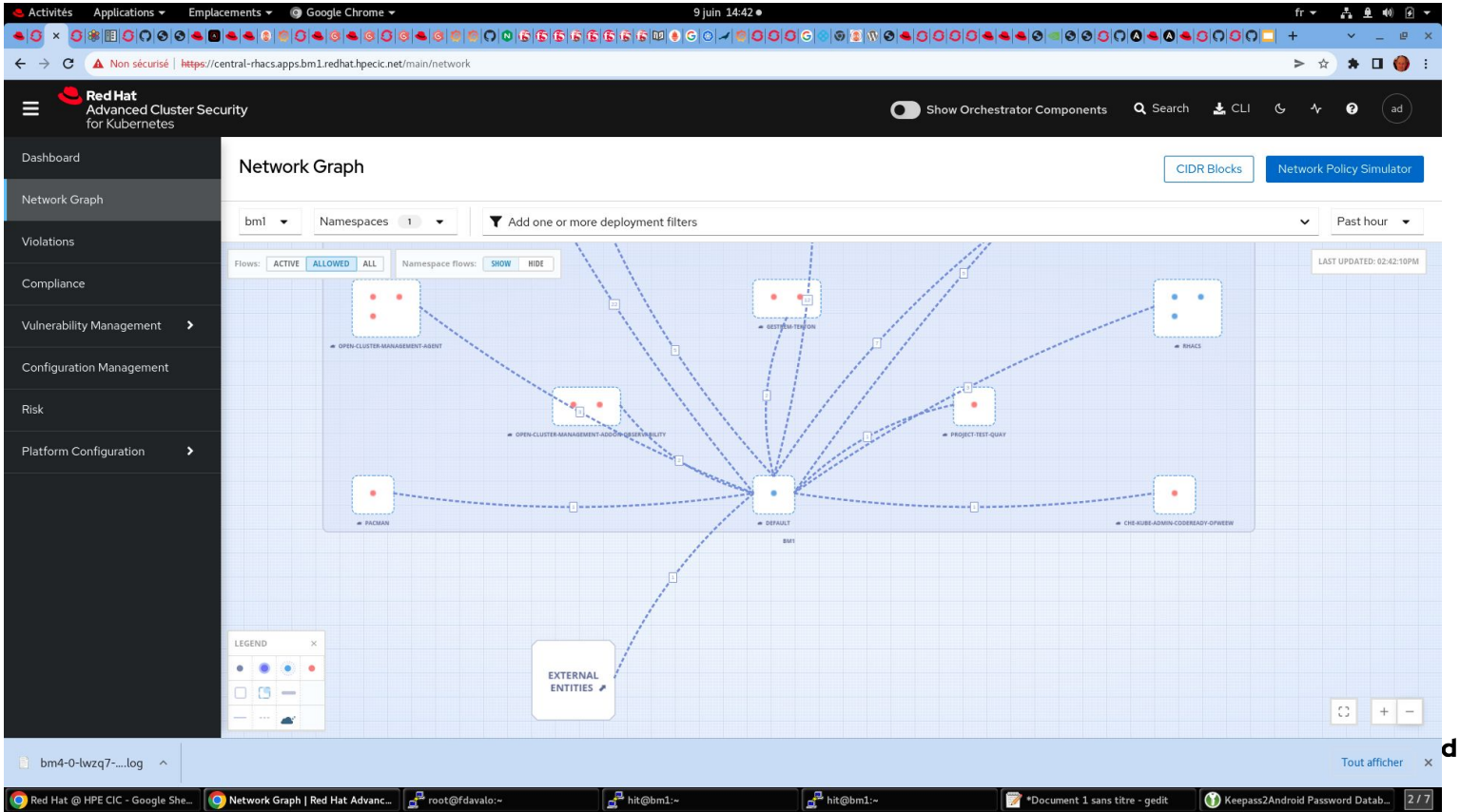
Red Hat

SR-IOV

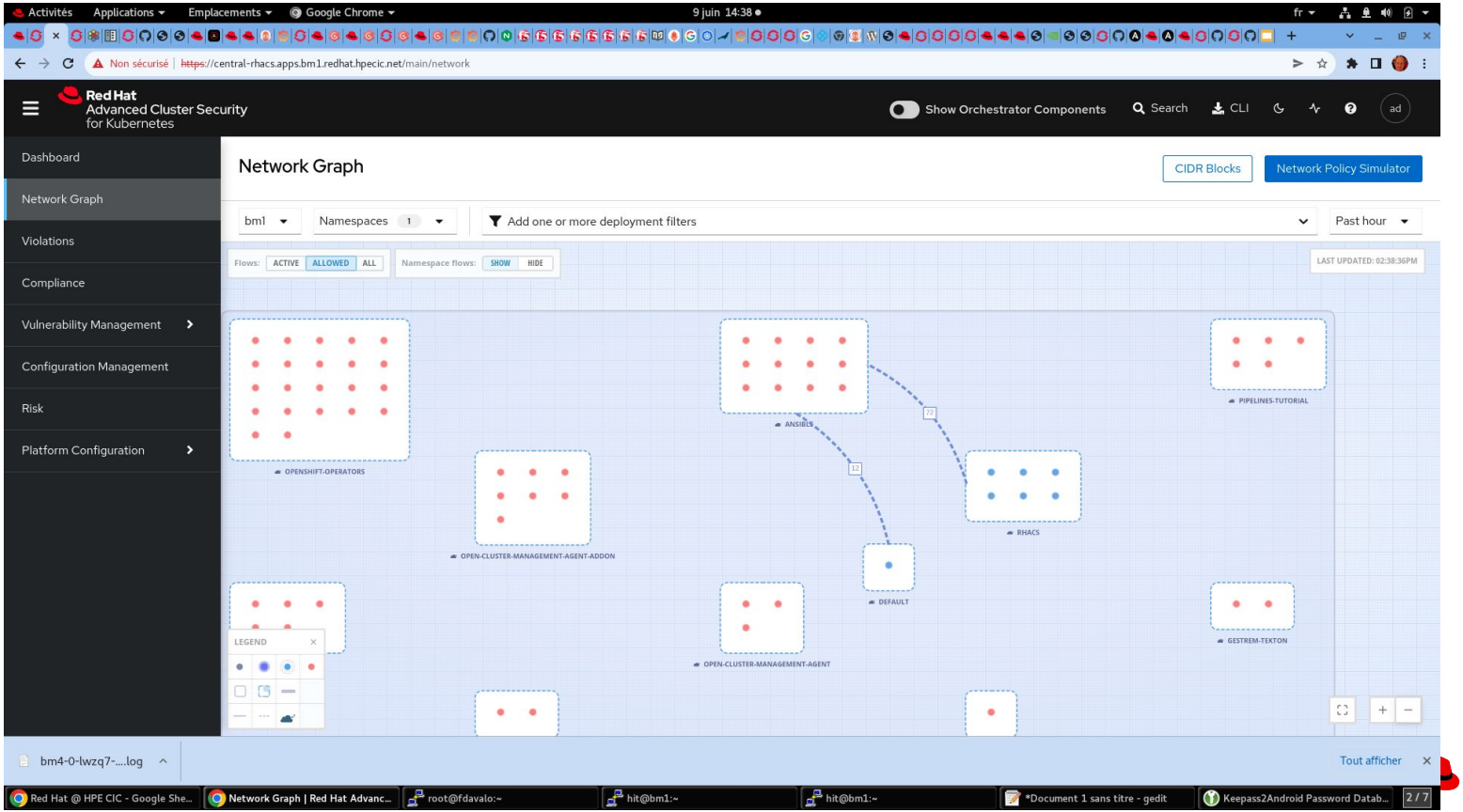



Advanced Cluster Security
Network Policy Management
redhat.

ACS Network Graph



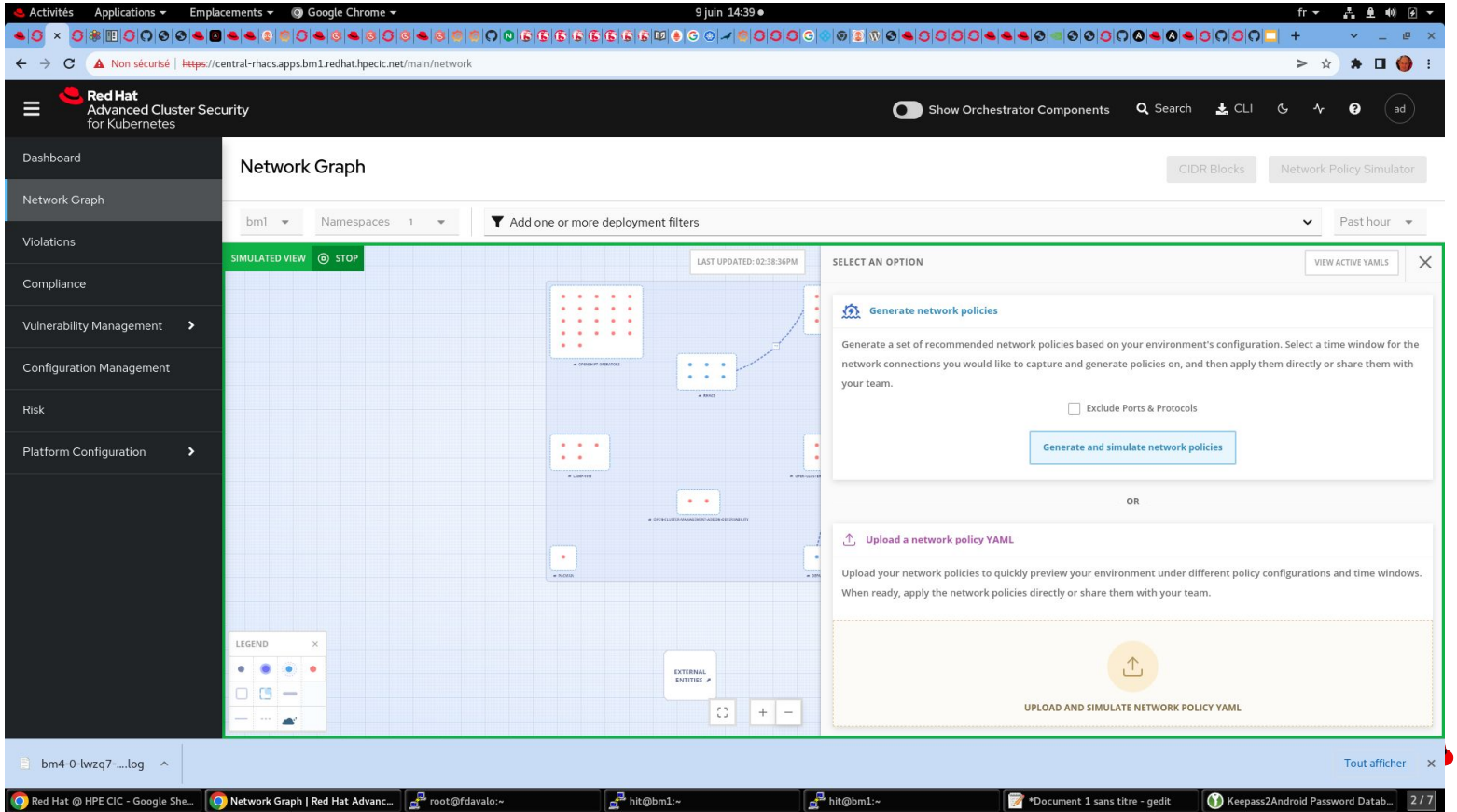
ACS Network Graph



Red Hat

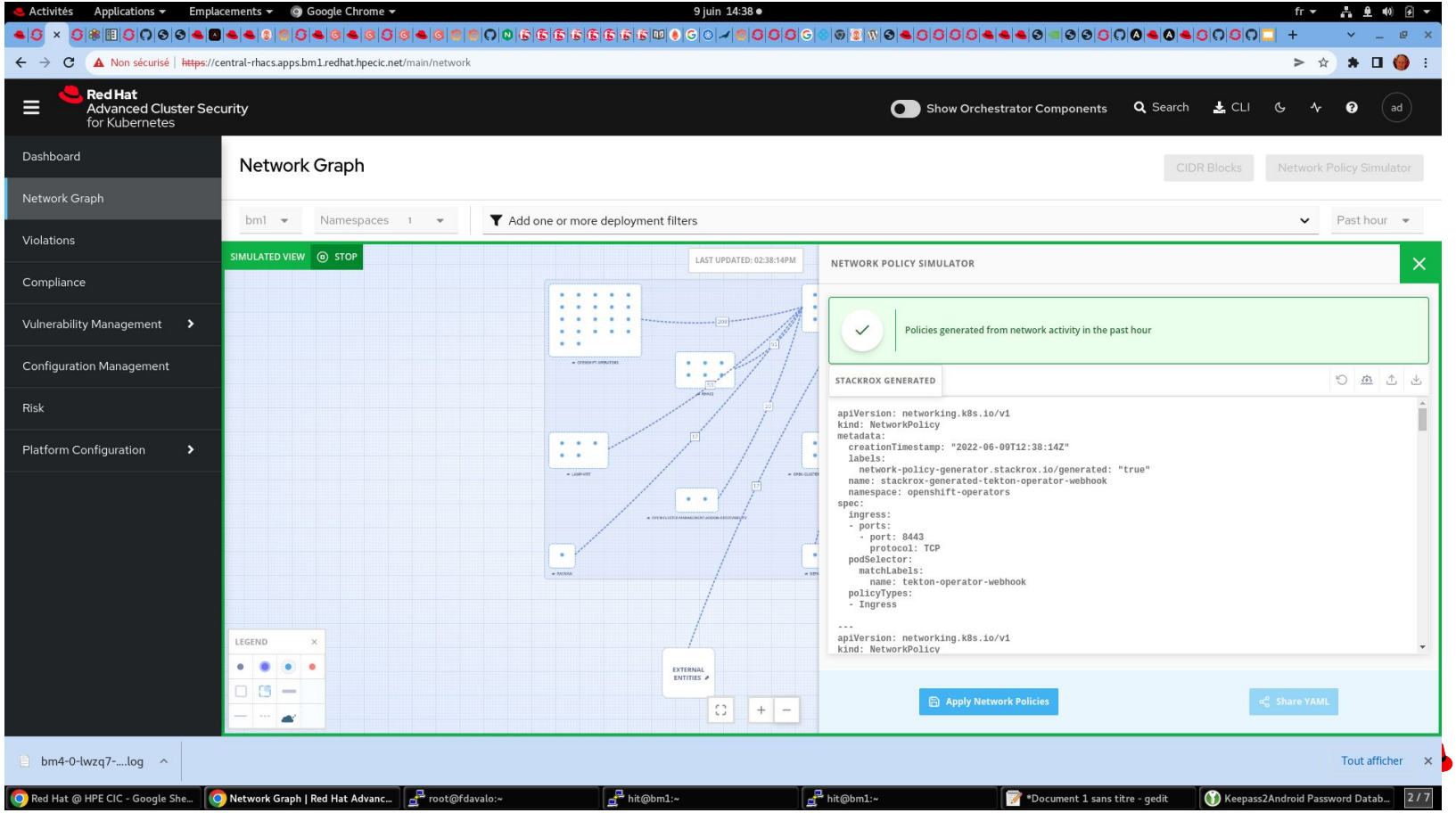
ACS

Generate Network Policies / Upload and simulate Network Policy



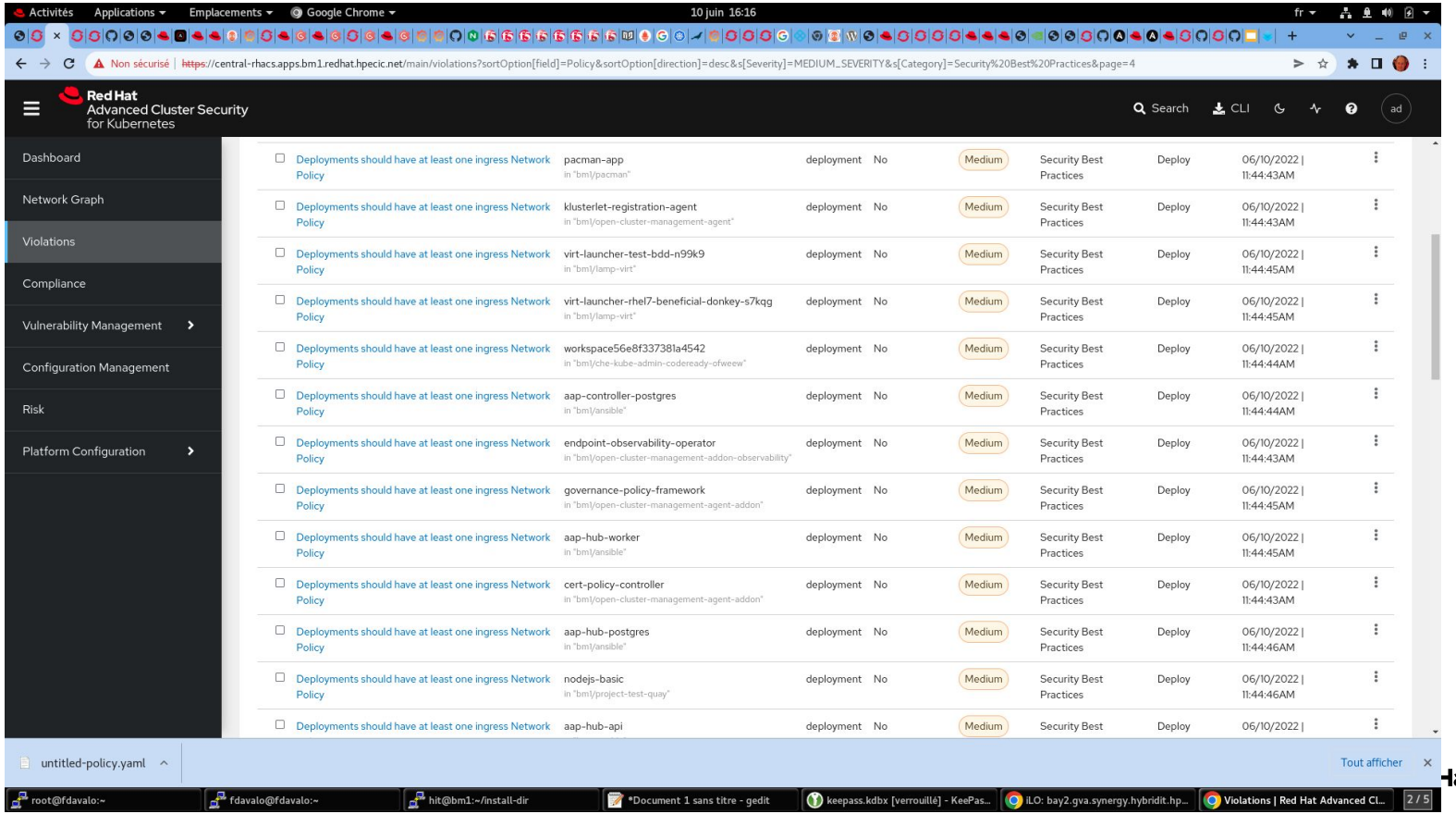
Red Hat

Network Policy Simulator



Red Hat

Network policy enforcement on namespaces




Monitoring network flows

Observability

Networking

Unified Experience

Network Traffic Metrics and Tracing

Whether one cluster or one hundred, developers and cluster administrators require seamless connectivity across applications.

Security Everywhere

Network Policy and Governance

Security and regulatory compliance requires governance of traffic in, around, and out of networks.

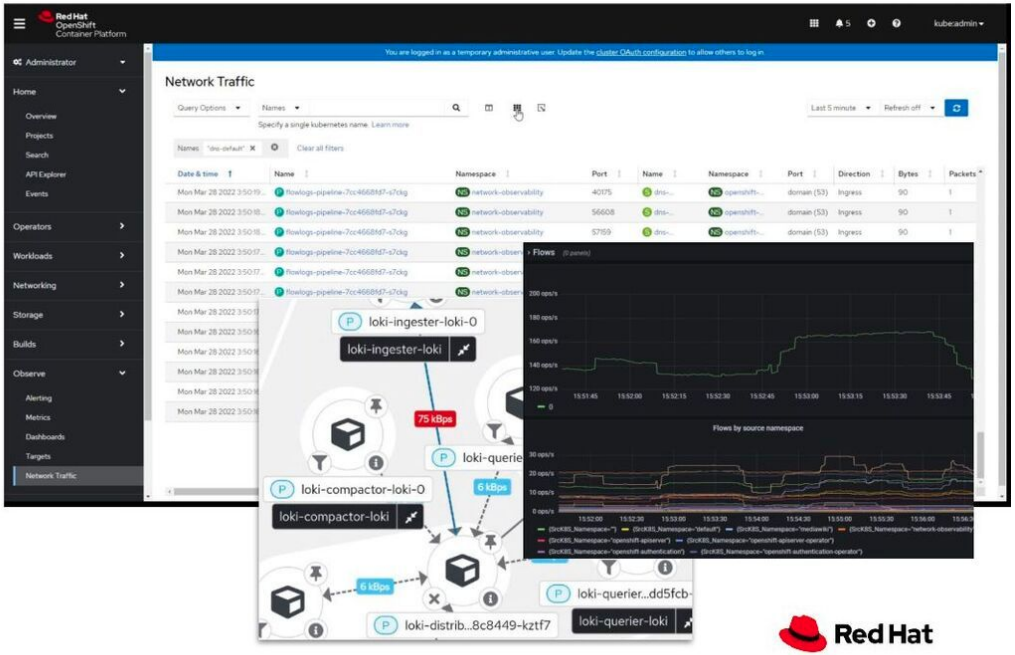
Platform Consistency

Network Traffic Flow and Topology

Developers and administrators require a common understanding of their traffic within and across cluster boundaries.

Network Flow Data – New Insight & Presentation

Tabular Netflow data, NOC Dashboard, Pod/Service/Node-specific Topology, New Metrics, Export options



Red Hat

What is Network Observability?

Network Observability provides real-time visualization and insight into your network.

It goes beyond traditional monitoring which includes metrics, logs, and tracing.

Monitoring tells you what happened. Observability tells you why it happened.

Six Functional Areas of Network Observability

Network flow visibility

What traffic is flowing in, around, and out of my network?

Network topology

What physical and virtual resources are connected to my network?

Network metrics

High-level performance metrics to indicate the overall health of my network

Network tracing

What path does specific traffic take through various interfaces to its final destination?

Network policy

Rules governing the flow of traffic in, around, and out of my network

Network traffic capture

Capture + mirror to an external location for debugging, audit, or lawful capture

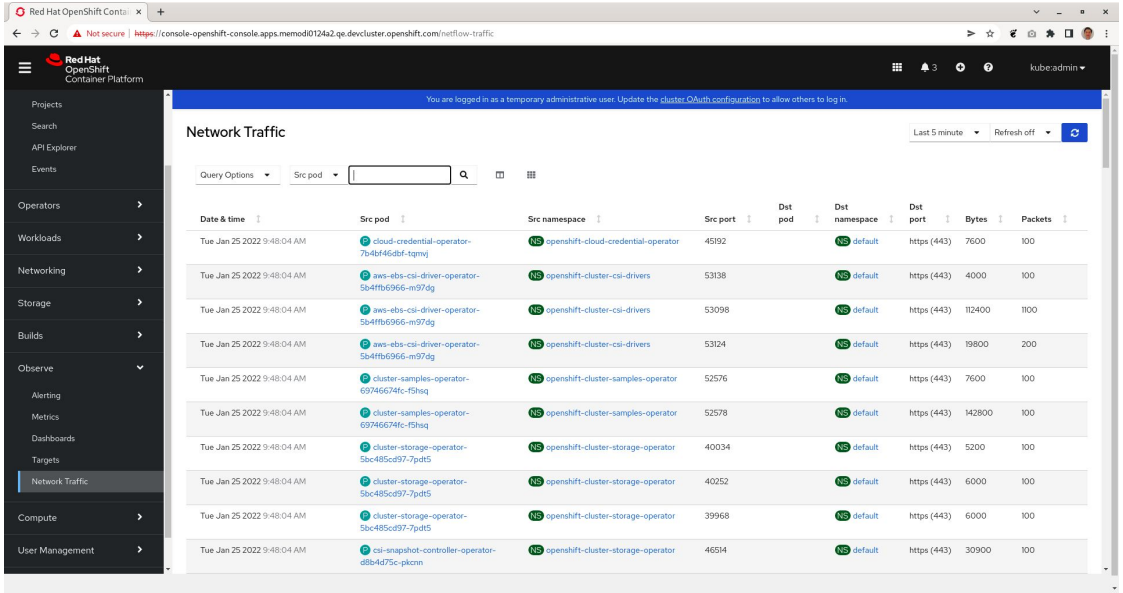
Network Observability in OCP 4.10

(Dev Preview)

- Collect and store NetFlows
  - GoFlow2-Kube Enricher
- NetFlow table and graphs
  - Network Observability UI Plugin
- Network Observability Operator

- Single cluster
- OVN-Kubernetes only
- kubeadmin (not developer)

NetFlow Table in OCP 4.10

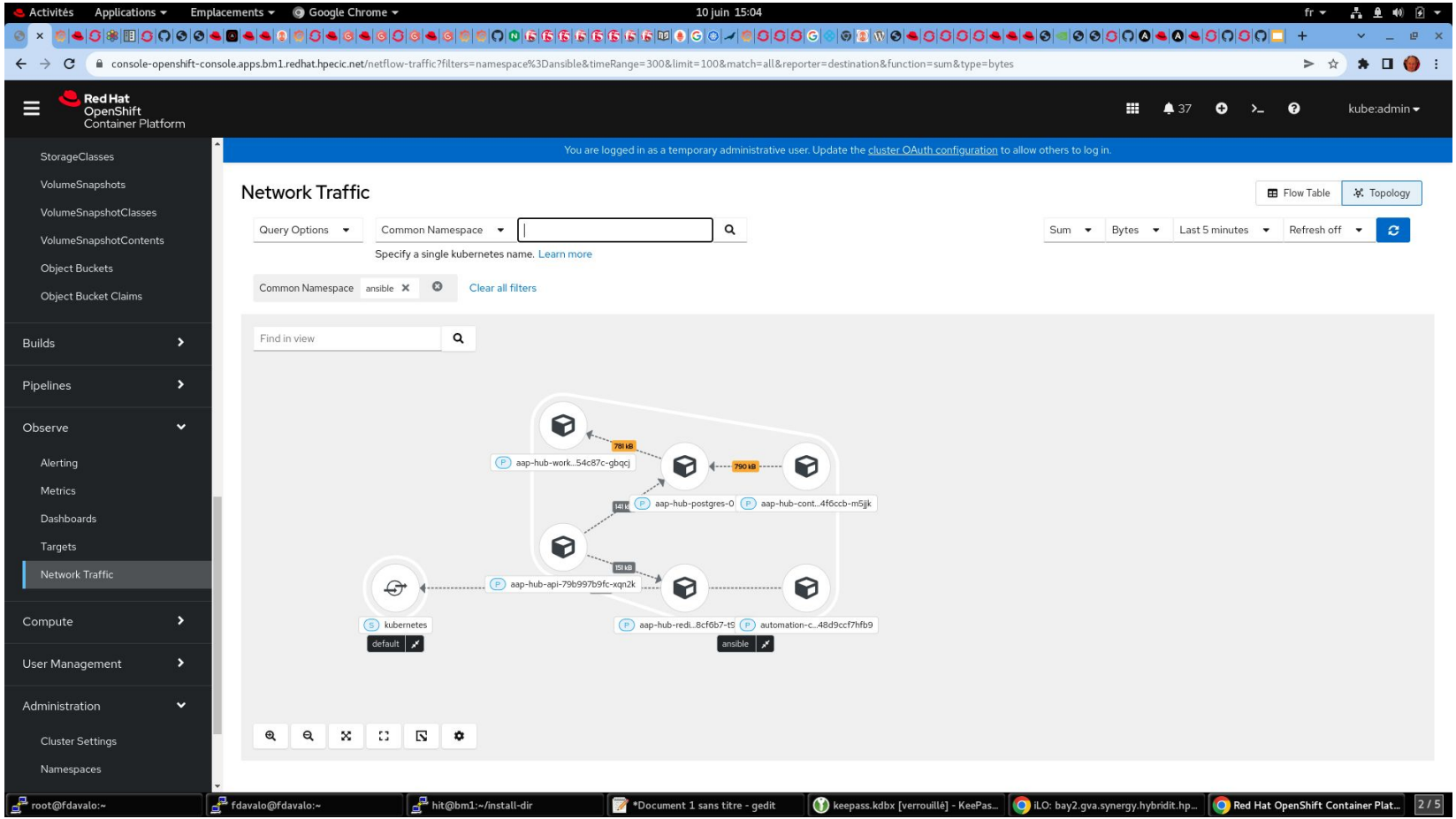


Features

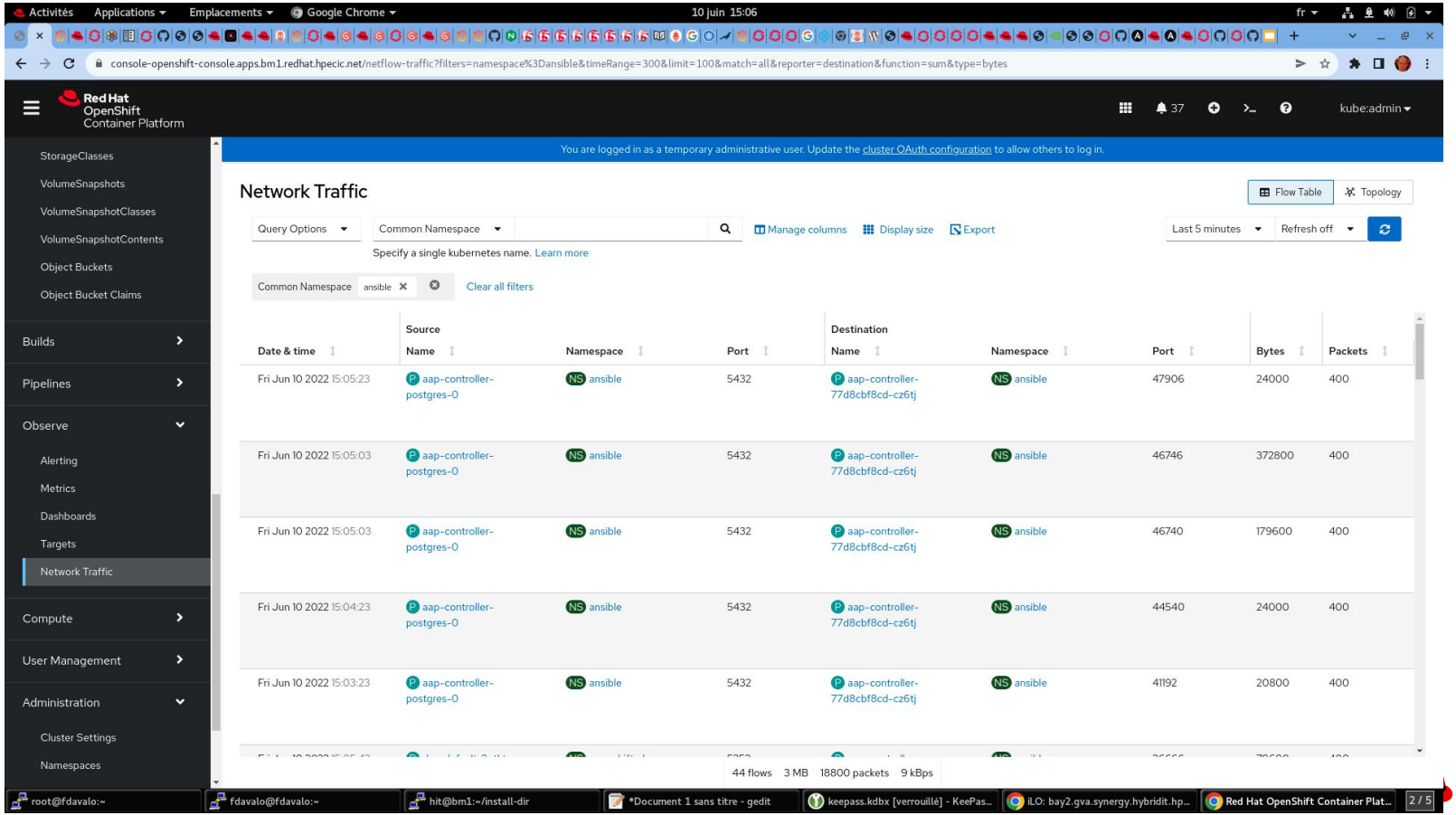
- Pod-to-pod traffic
- Namespaces / projects
- Map port to service names
- Column selection
- Filtering
- Sorting
- Time range
- Refresh interval

Red Hat

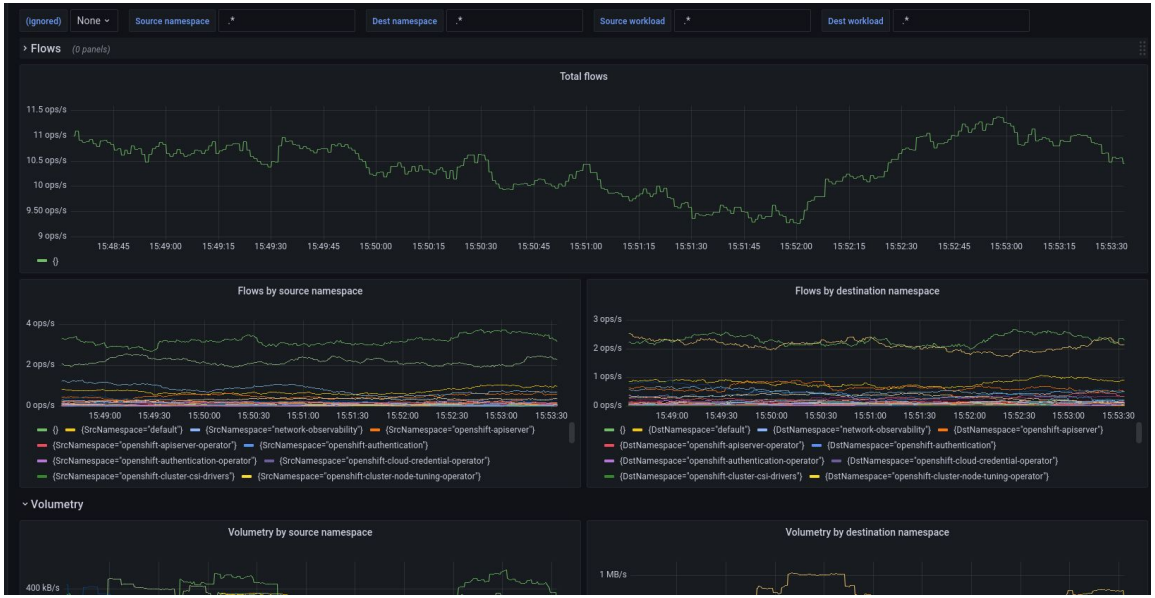
Network observability operator



Network observability operator



NetFlow Graphs in OCP 4.10



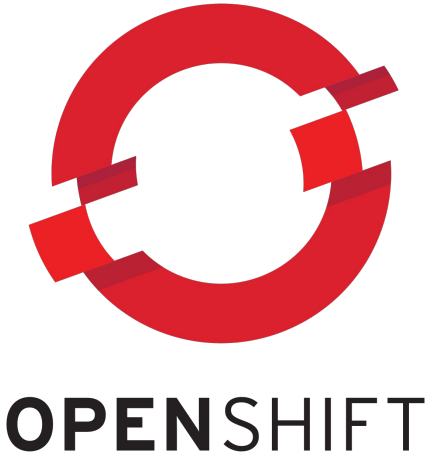
Features

- Leverages Grafana
- Create your own graphs
- Provides dashboards

Red Hat

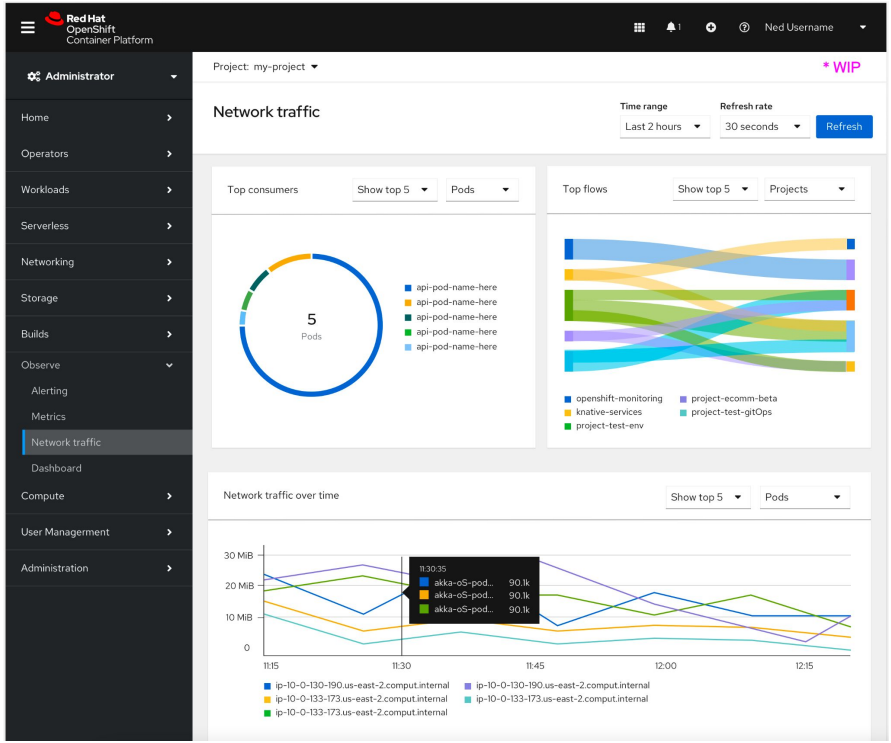
Network Observability in OCP 4.11 and Beyond

(Tech Preview)



- Retrieve NetFlows using eBPF
- Topology
- Dashboards on top pod talkers, layer 4 services, ...
- Apache Kafka for scalability
- Productization
- Persistent storage

OCP 4.11 Dashboard



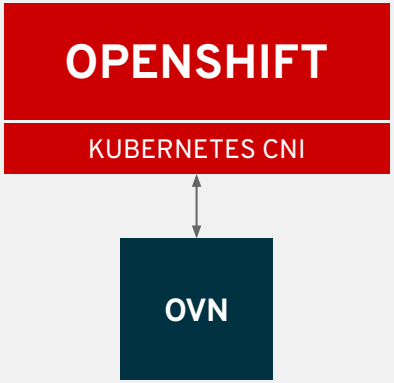
Features

- Top pod talkers
- Graphical pod-to-pod communication
- Network traffic over time

Red Hat


OVN Kubernetes
redhat.

Open Virtual Network (OVN)



Next-Gen Default OpenShift SDN

- An implementation of virtual networking via Open vSwitch project
- Developer Community
  - SDN portfolio consolidation / common network tech (RH-OCP, RH-OSP, RHV)
- Acceleration and enablement of customer-driven feature requirements
  - Egress IP per pod
  - Distributed Ingress/Egress firewall
  - Distributed services LB
  - Multi-Network/Interface
  - Heterogeneous clusters w/ Windows nodes
  - Capability to span on-prem & cloud nodes
  - Traffic isolation / Multi-tenancy
  - DPDK support
  - Encrypted tunnels
  - IPv6 / DHCPv6
  - QoS, Control/Data plane separation
  - ...

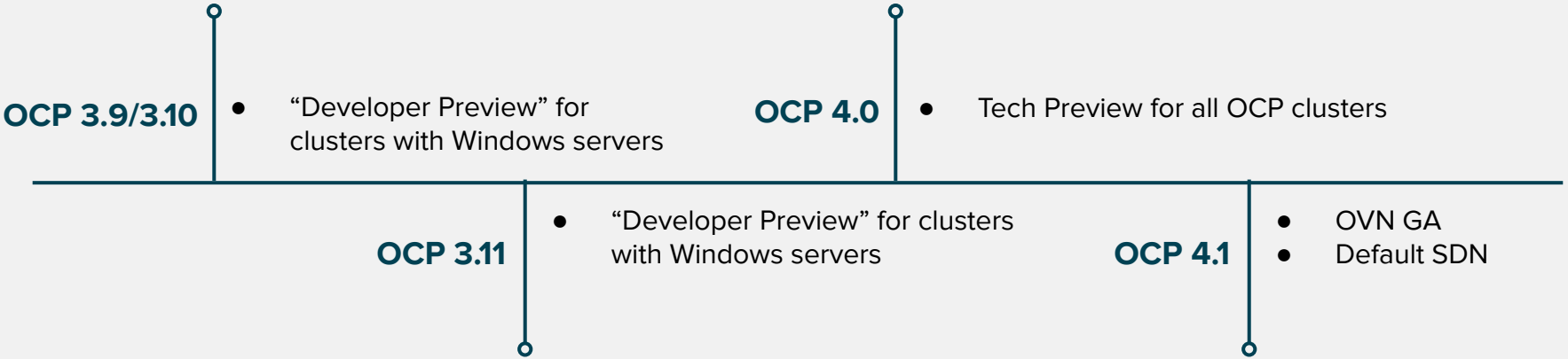
Advantages of OVN Kubernetes

- The core OVN project is also used by OpenStack and RHEV
  - We get their bugfixes, performance improvements, and features
- Red Hat has a team working on OVN who understands it deeply
- Other companies are also using (or planning to use) OVN-Kubernetes
  - The project was started by NVIDIA and VMware; Red Hat joined later
- Effectively, our team is much larger now, giving us more ability to implement new features like: IPv6, hybrid Linux/Windows clusters, more powerful NetworkPolicy support, etc

Hey… Wait a minute…

OpenShift SDN	OVN Kubernetes
veth pairs	veth pairs
OVS bridge	OVS bridge
Central controller / host-ipam	Central controller / host-ipam
VXLAN tunnels	Geneve tunnels
OVS flows for NetworkPolicy	OVS flows for NetworkPolicy
IPTables for Services	OVN LBs for Services
IPTables for NAT	OVS for NAT

OVN Enablement Timeline

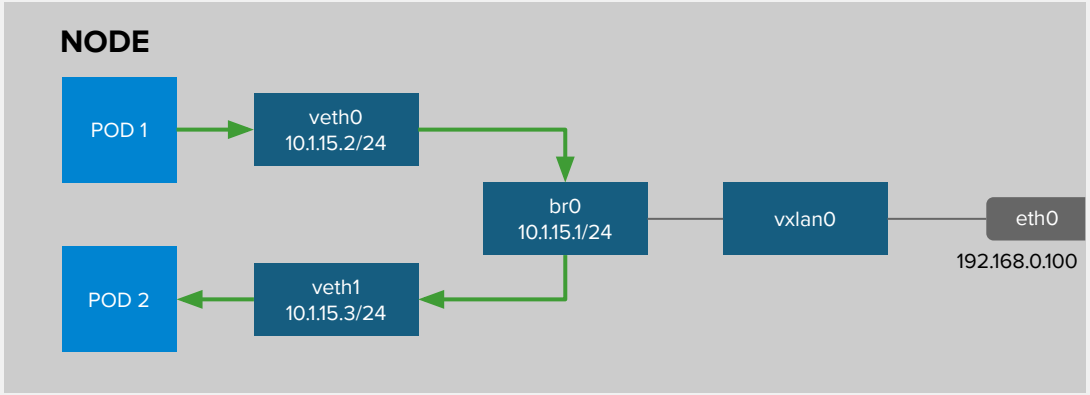


OVN maturity starting at 4.9, stability phase enforced, OVN default CNI could be 4.12


Openshift SDN Flows
redhat.

OPENSHIFT SDN - OVS PACKET FLOW

Container to Container on the Same Host



OPENSHIFT SDN - OVS PACKET FLOW

Container to Container on the Different Hosts

OPENSHIFT SDN - OVS PACKET FLOW

Container Connects to External Host