Netzwerk Sicherheit für Anwendung in einer OpenShift Umgebung

Robert Baumgartner
Senior Solution Architect OpenShift & Middleware
Red Hat Austria

Agenda

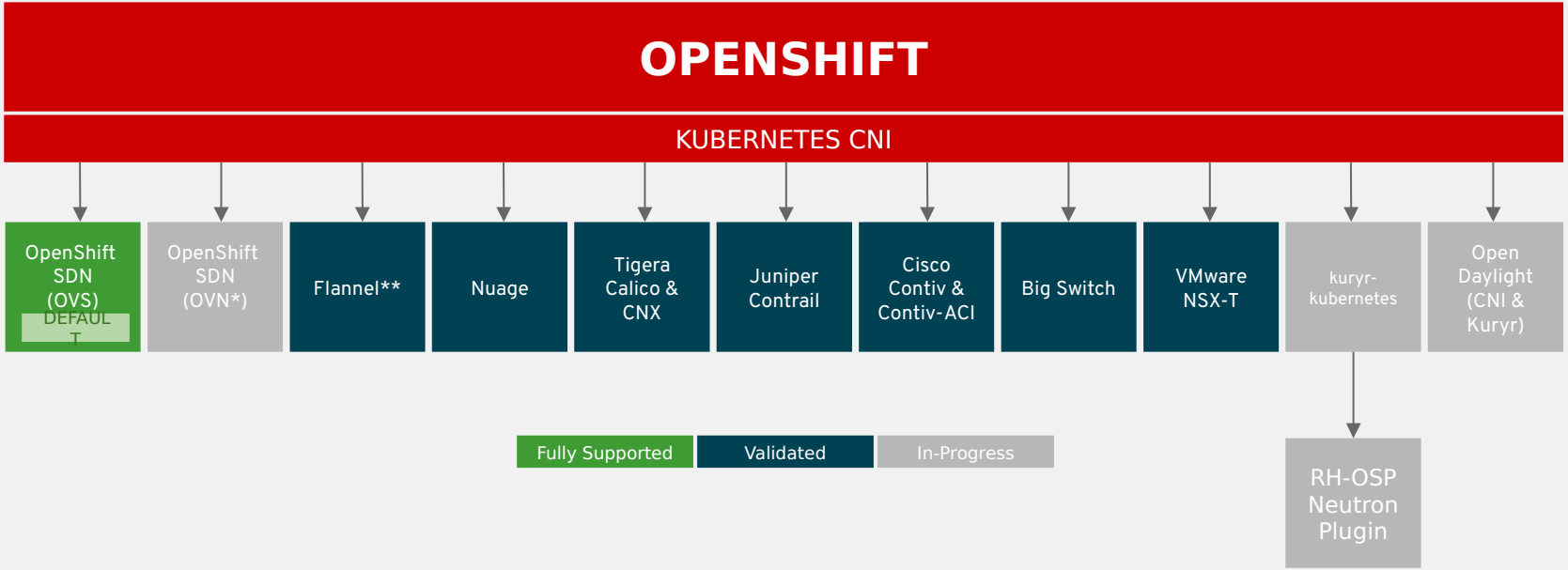
OpenShift SDN Overview

Typical network security questions for OpenShift

- Restricting traffic across tiers
- Handling network zones and isolation
- Securing Egress
- Securing Ingress
- Securing communications between OpenShift Nodes
- Application Network Security
- Istio / OpenShift Service Mesh

OpenShift SDN Overview

OpenSift uses OCI



* Coming as default in OCP 4.1
** Flannel is minimally verified and is supported only and exactly as deployed in the OpenShift on OpenStack reference architecture

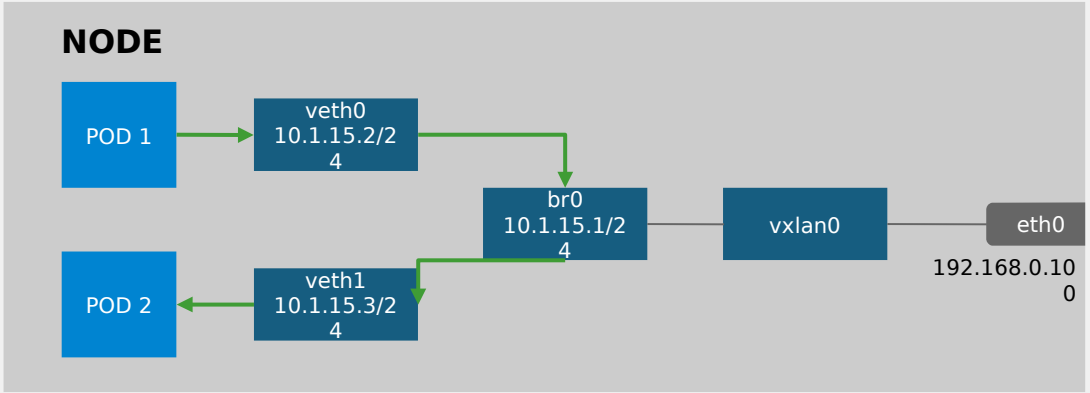
OpenShift Networking

Software Defined Networking (SDN) for pod-pod communication

- Configures overlay network using Open vSwitch (OVS)
- Three types of plugins
  - **ovs-subnet** : flat network every pod can talk to every other pod
  - **ovs-multitenant**: project level isolation for pod-pod communication. Unique VNID per project
    You can join projects to get them the same VNID
    'default' project (VNID 0) privileged to communicate with other pods
  - **ovs-networkpolicy**: fine-grained isolation using network policy objects

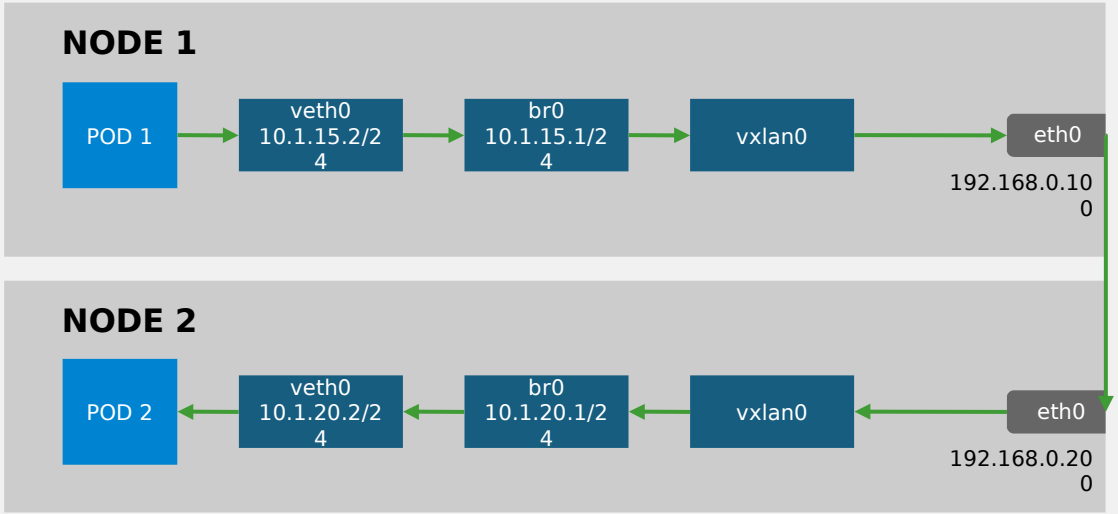
OPENSHIFT SDN - OVS PACKET FLOW

Container to Container on the Same Host



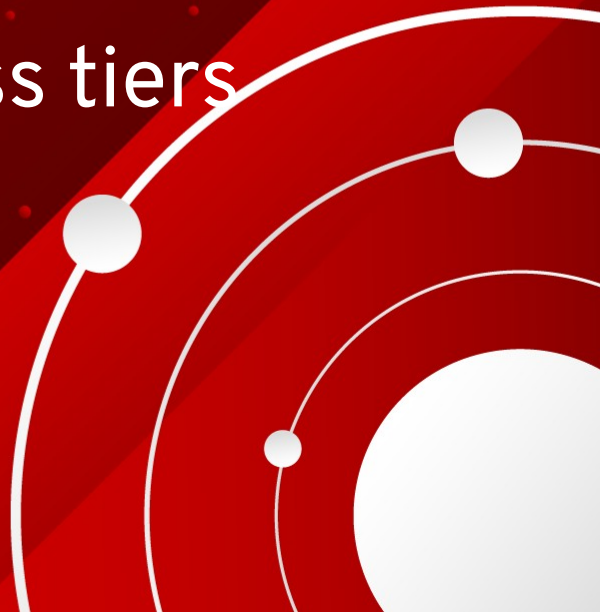
OPENSHIFT SDN - OVS PACKET FLOW

Container to Container on the Different Hosts

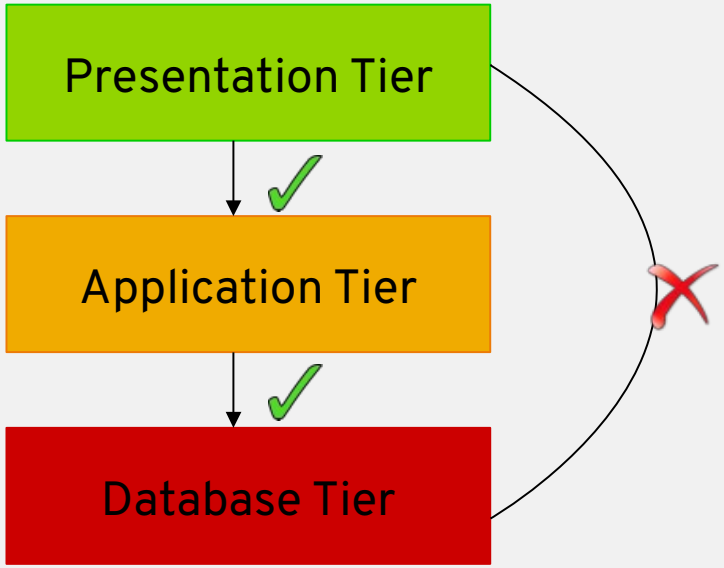


Typical Network Scenarios and OpenShift Solutions

1. Restricting traffic across tiers



Traffic Restrictions Across Application Tiers



Allowed connections

Disallowed connections

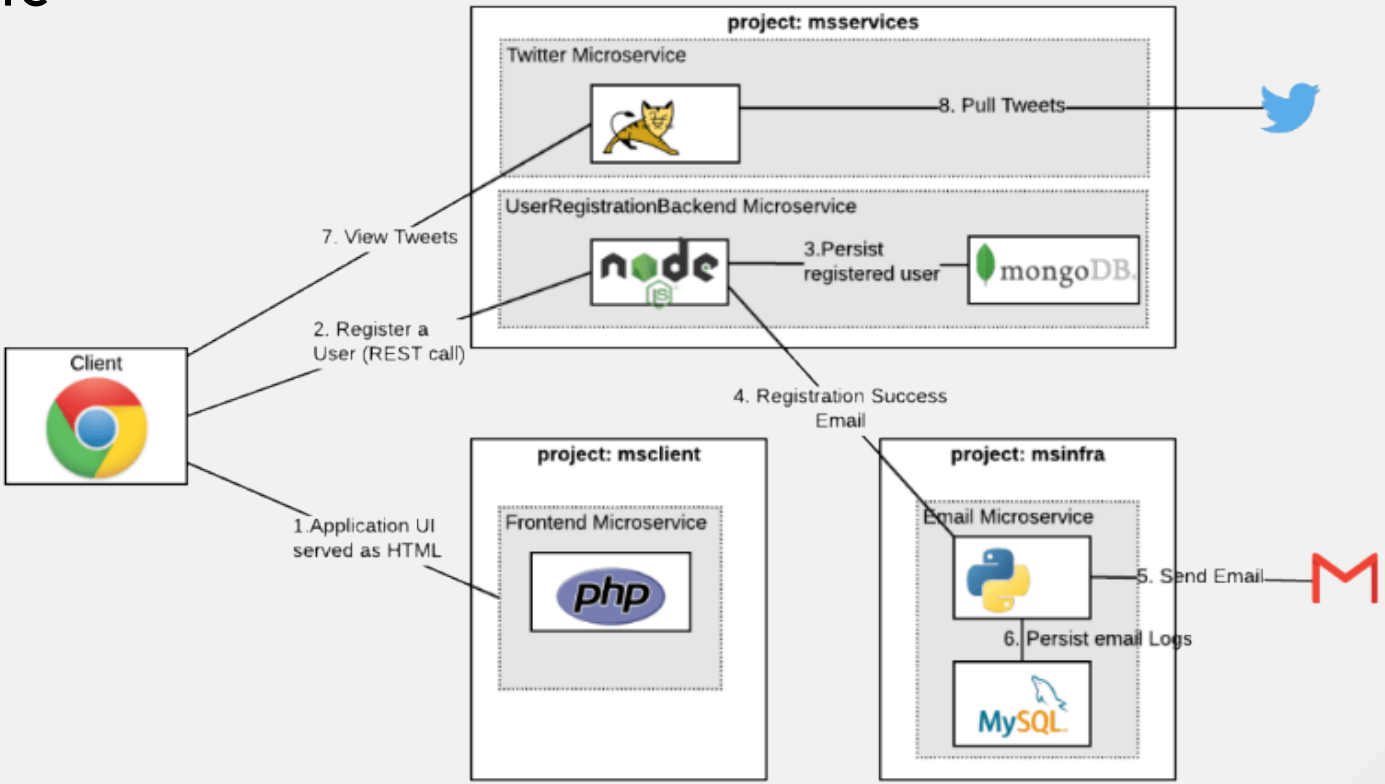
In the world of OpenShift, how can we restrict traffic across Application Tiers?

Network Policy Objects - Introduction

Enables **Microsegmentation**

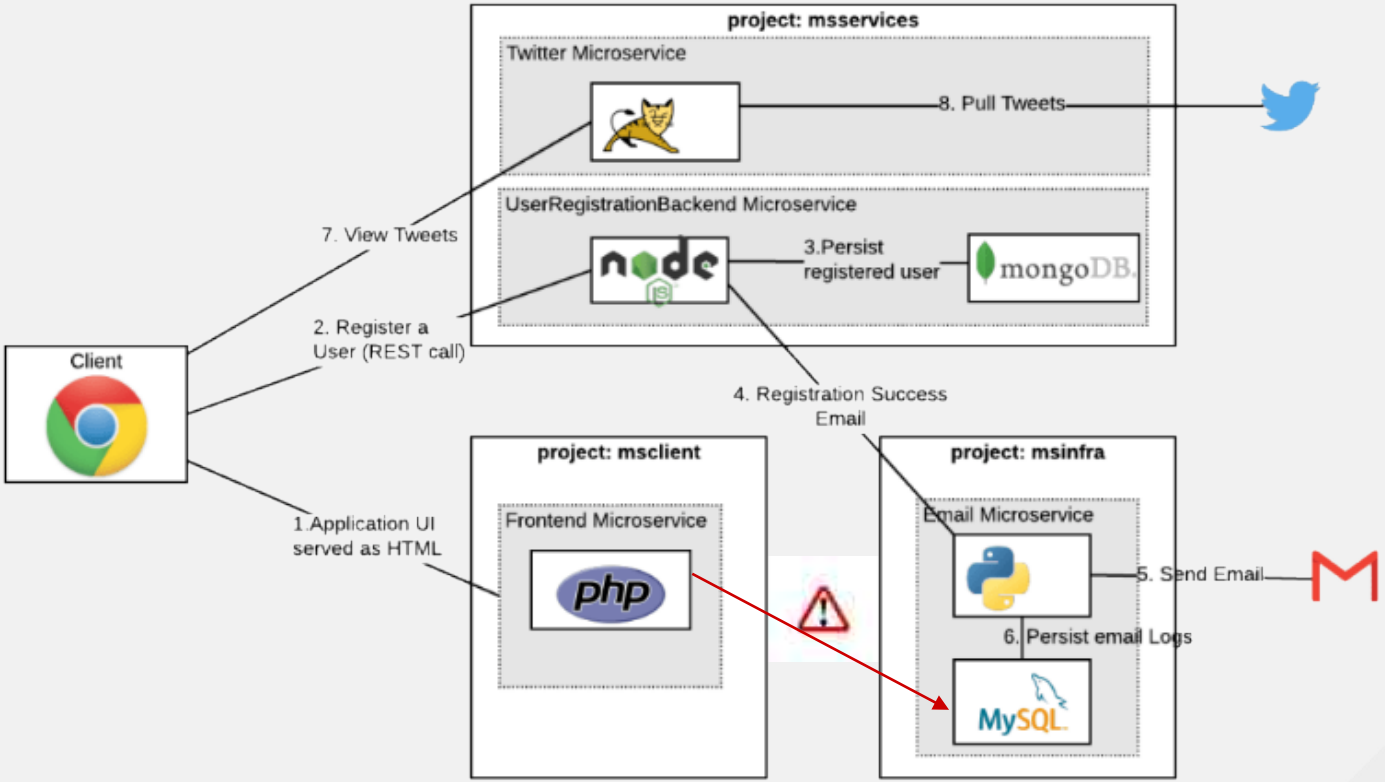
- Allows configuring individual policies at the Pod Level
- Apply to ingress traffic for pods and services
- Allows restricting traffic between the pods within a project/namespace
- Allows traffic to specific pods from other projects/namespaces

Example

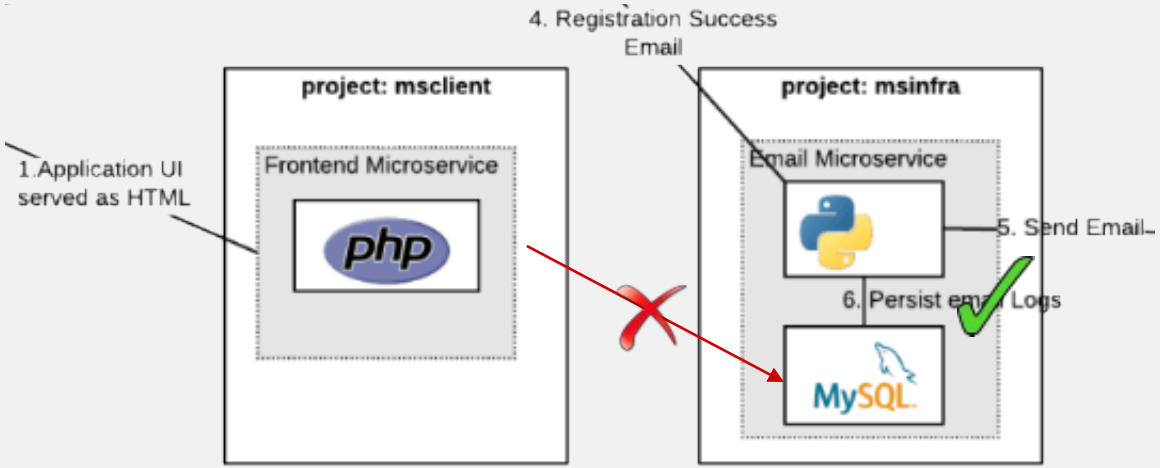


Video: https://blog.openshift.com/network-policy-objects-action/

Hack


project: msservices
Twitter Microservice
8. Pull Tweets
UserRegistrationBackend Microservice
3.Persist registered user
mongoDB.
7. View Tweets
2. Register a User (REST call)
Client
4. Registration Success Email
project: msclient
Frontend Microservice
php
1.Application UI served as HTML
project: msinfra
Email Microservice
5. Send Email
6. Persist email Logs
MySQL

Network Policy Objects to Rescue

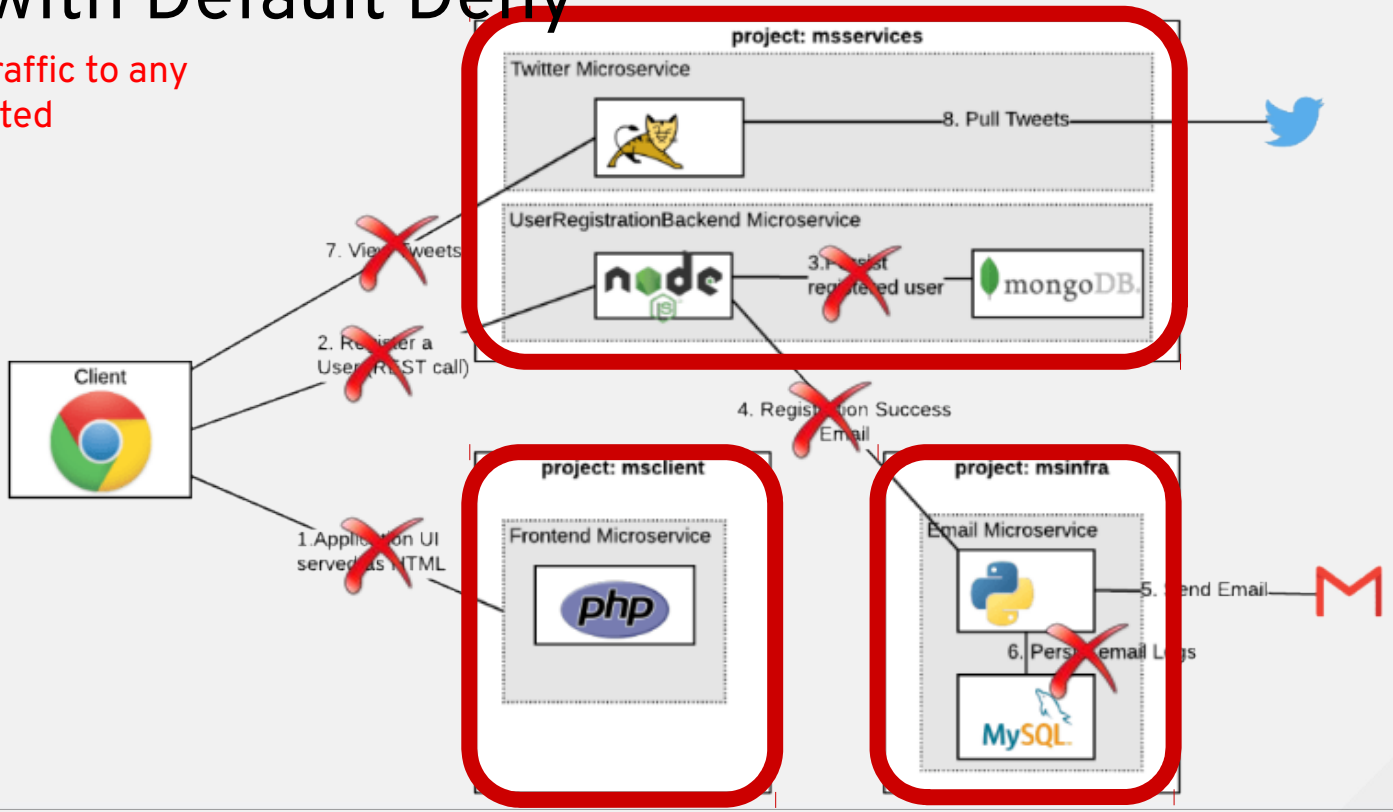


Allow MySqlDB connection from Email Service

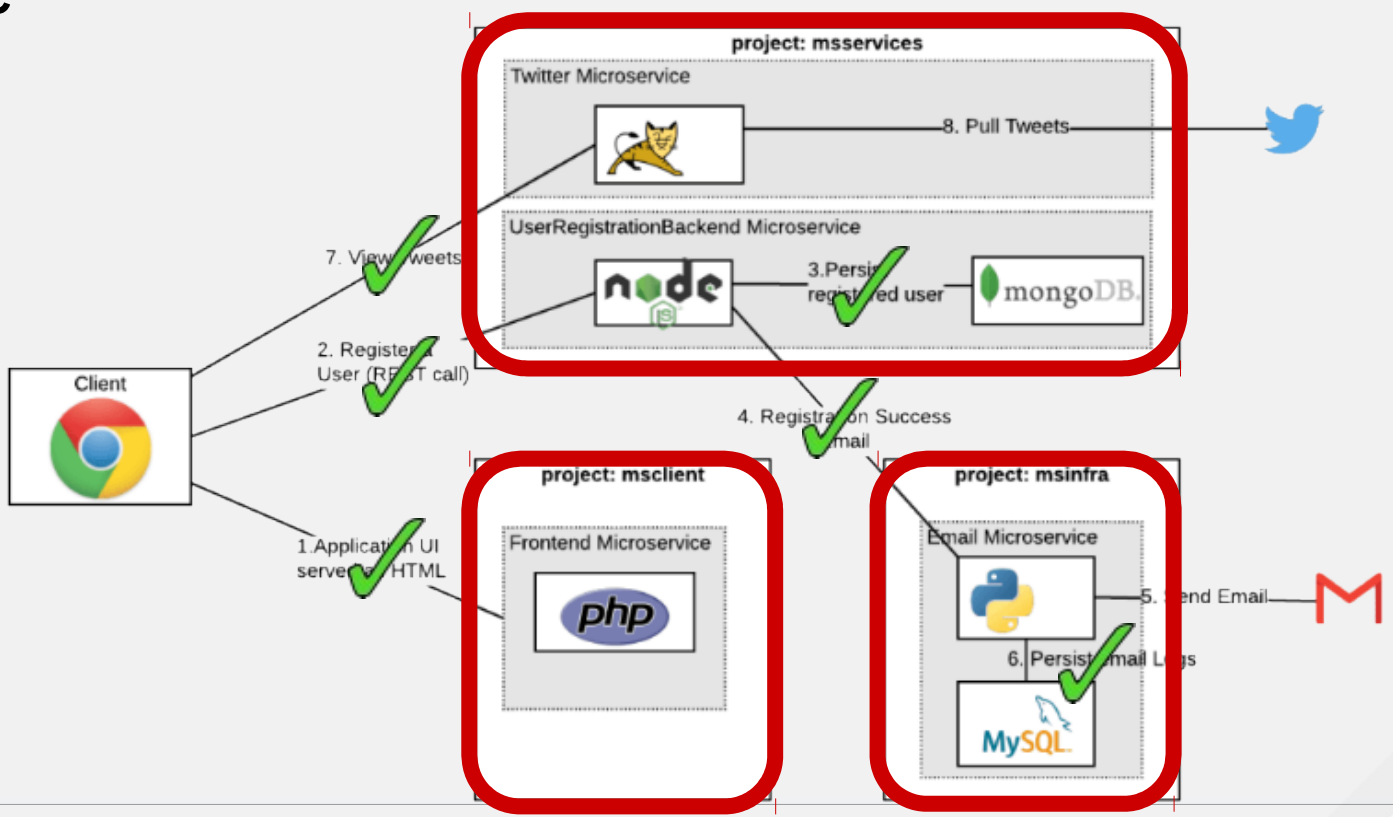
```
kind: NetworkPolicy
apiVersion: extensions/v1beta1
metadata:
  name: allow-3306
spec:
 podSelector:
   matchLabels:
     app: mysql
 ingress:
 - from:
   - podSelector:
       matchLabels:
         app: emailsvc
   ports:
   - protocol: TCP
     port: 3306
```

Start with Default Deny

All ingress traffic to any pods is rejected

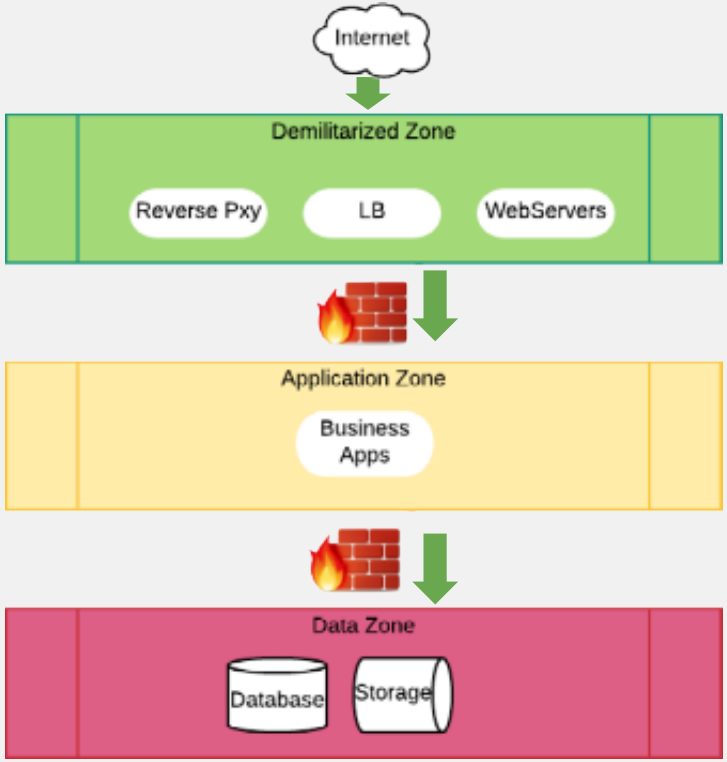


Add Network Policies To Allow Specific Incoming Traffic



2. Isolating zones

Network Zones separated by Firewalls



External traffic allowed to touch DMZ

Holes punched in firewalls to allow specific traffic from

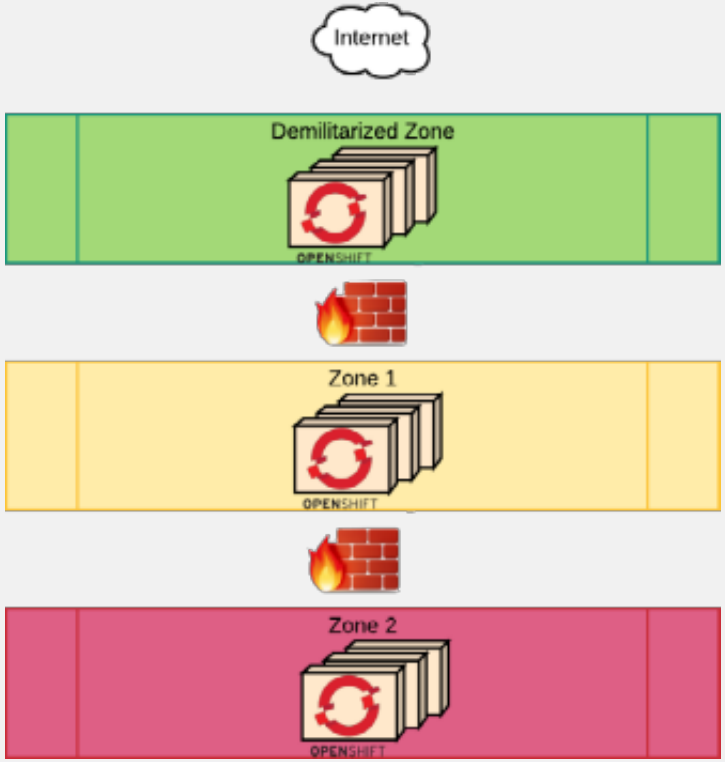
DMZ to Application Zone

and from

Application Zone to Data Zone

How do I setup OpenShift here?

Option 1: OpenShift cluster per Zone

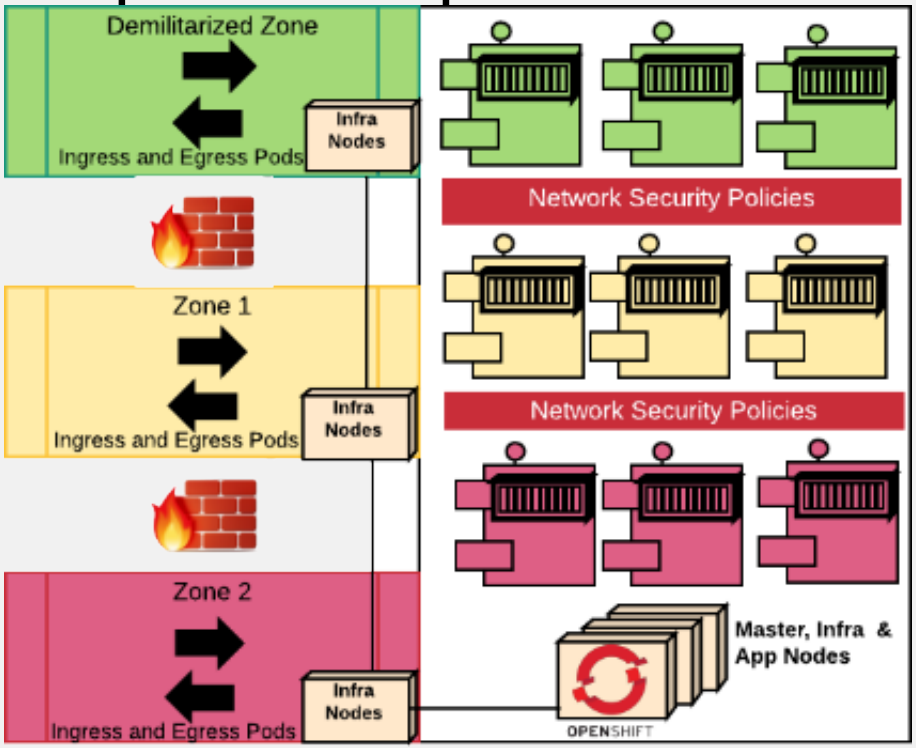


Useful to demonstrate compliance with Security Standards and Regulations

Additional actions needed to protect Master APIs, and other URLs in DMZ that are not supposed to be exposed to Internet

Cost of maintenance is high

Option 2: OpenShift Cluster covering Multiple Zones



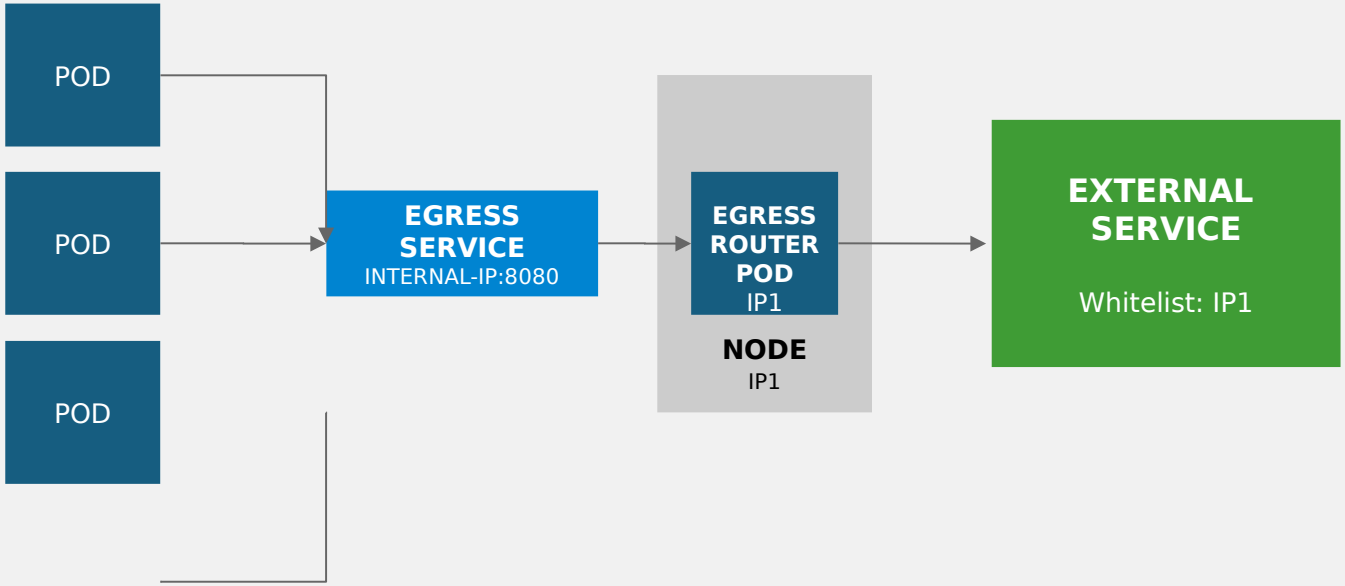
Application pods run on one OpenShift Cluster. Microsegmented with Network Security policies.

Infra Nodes in each zone run Ingress and Egress pods for specific zones

If required, physical isolation of pods to specific nodes is possible with node-selectors. But that defeats the purpose of a shared cluster. Microsegmentation with SDN is the way to go.

3. Securing Egress

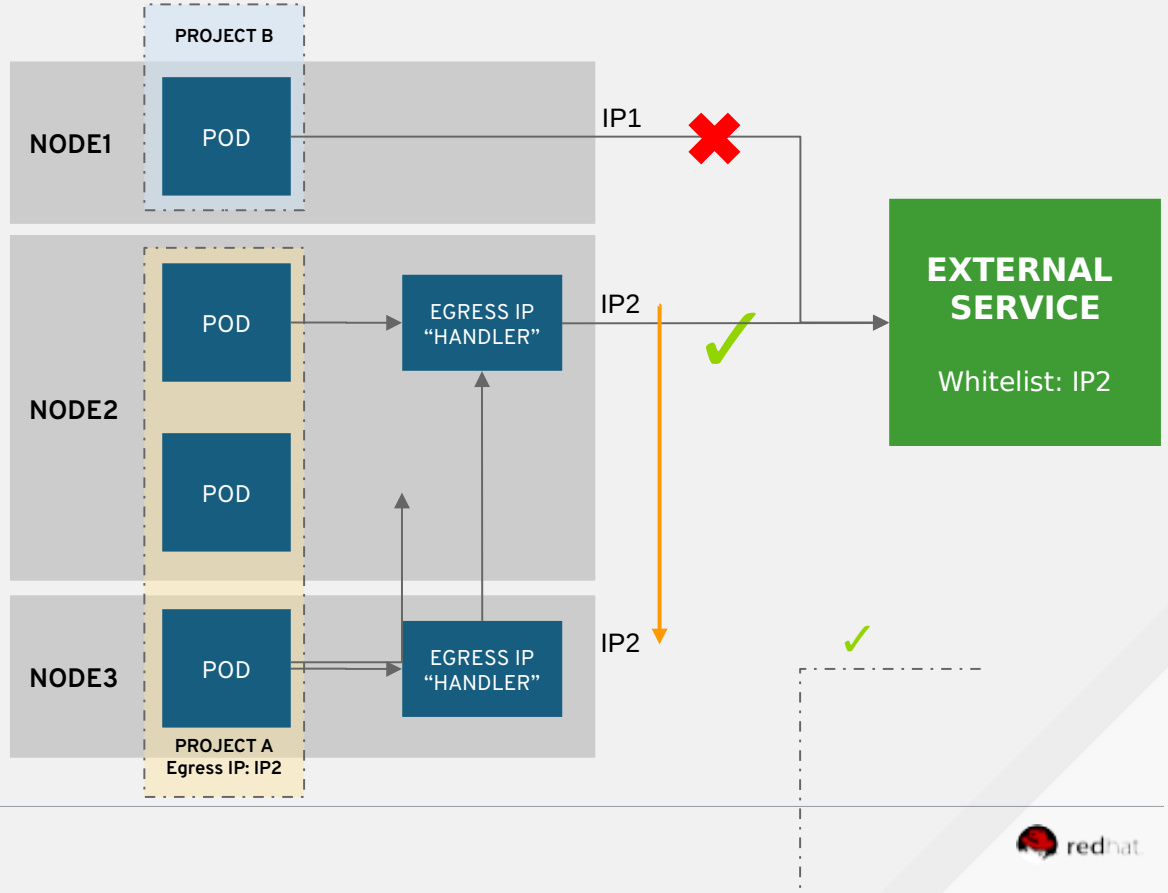
Control Outgoing Traffic
Source IP with Egress Router



Namspace-wide Egress IP

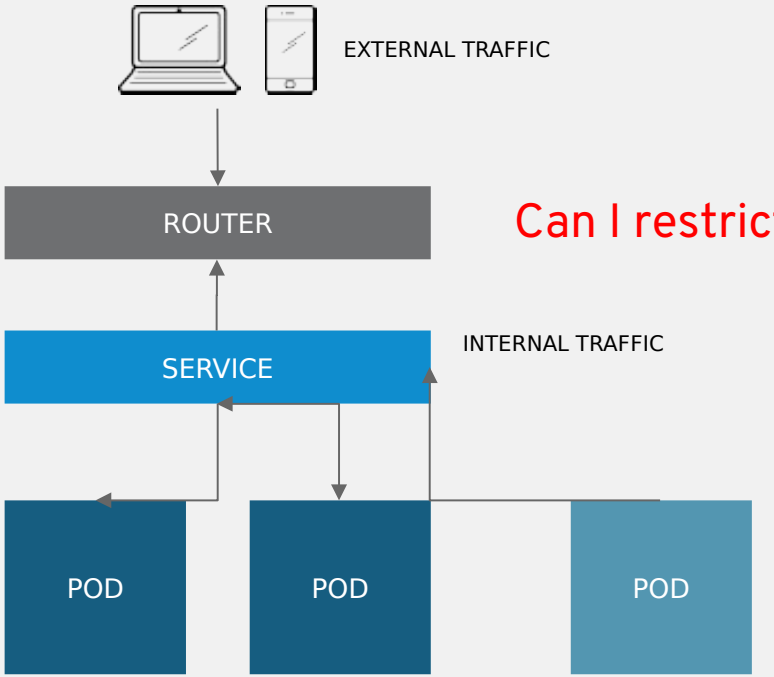
Projects are automatically allocated a single egress IP on a node in the cluster and that IP is automatically migrated from a failed node to a healthy node.

Full automatic



4. Securing Ingress

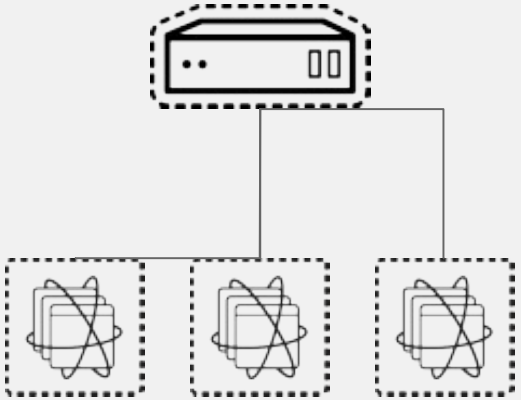
ROUTE EXPOSES SERVICES EXTERNALLY



Can I restrict access to route?

Routing and External Load-Balancing

- Pluggable routing architecture
  - HAProxy Router
  - F5 Router
- 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 Specific IP Whitelists

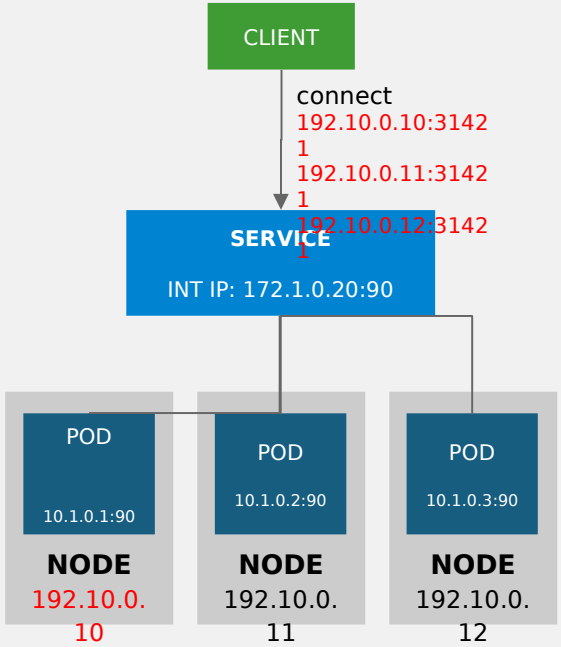
- Restrict access to a route to a select IP address(es)
- Annotate the route with the whitelisted/allowed IP addresses
- Connections from any other IPs are blocked

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10 192.168.1.11
```

What about ingress traffic on ports that are not 80 or 443?

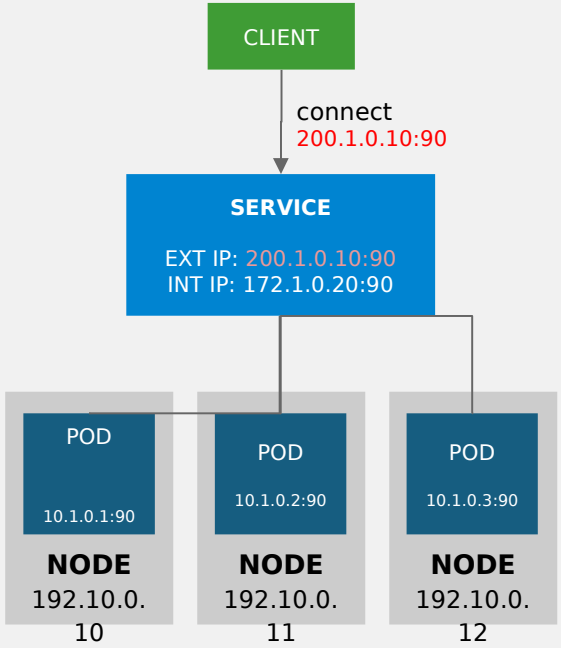
Using NodePort as Ingress to Service

- 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
- Every exposed service uses a port on all the nodes in a cluster. Are there alternatives?



Assigning External IP to a Service 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

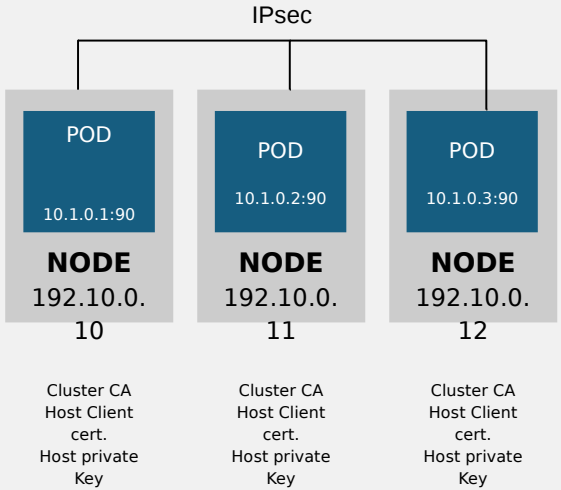


5. Securing communications between OpenShift nodes

Secured Communications between Hosts

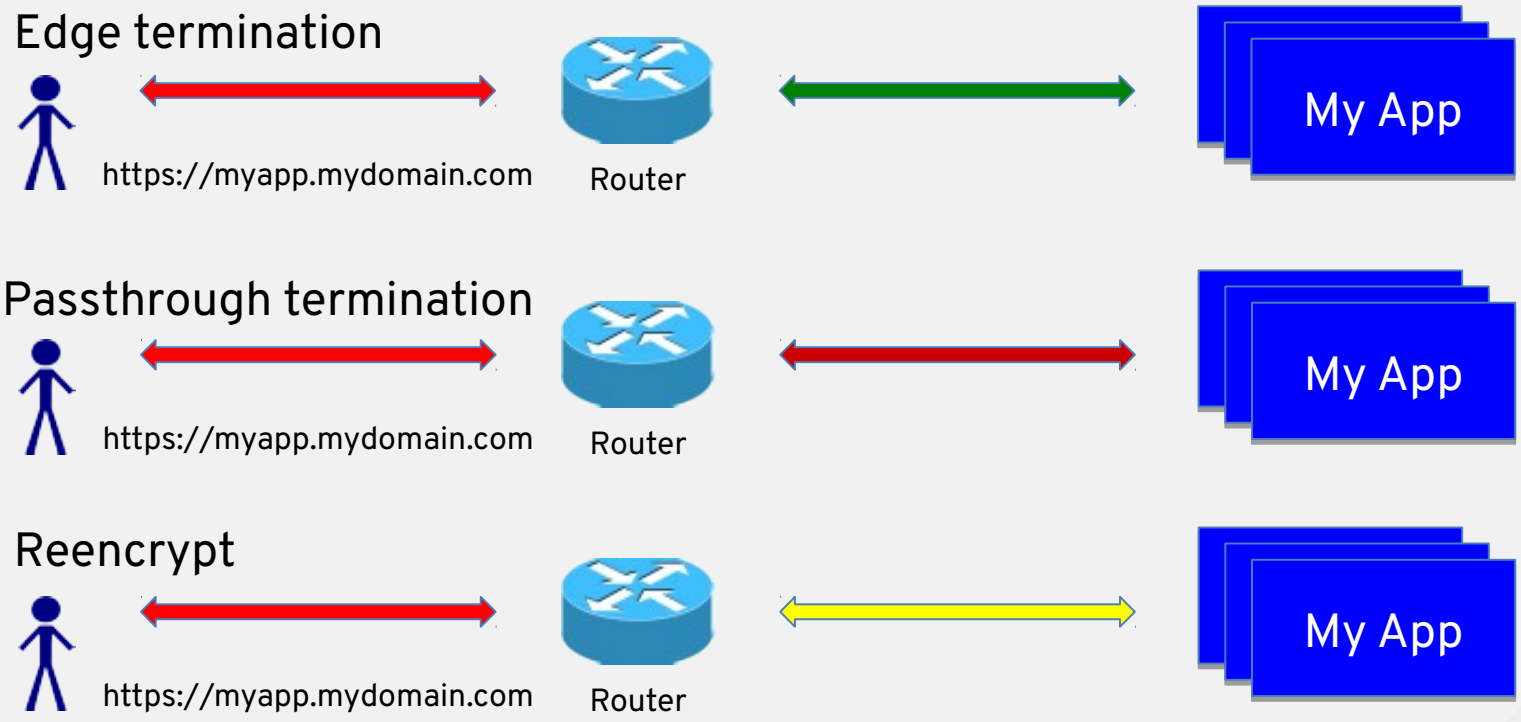
Secures cluster communications with IPsec

- Encryption between all Master and Node hosts (L3)
- Uses OpenShift CA and existing certificates
- Simple setup via policy defn
  - Groups (e.g. subnets)
  - Individual hosts



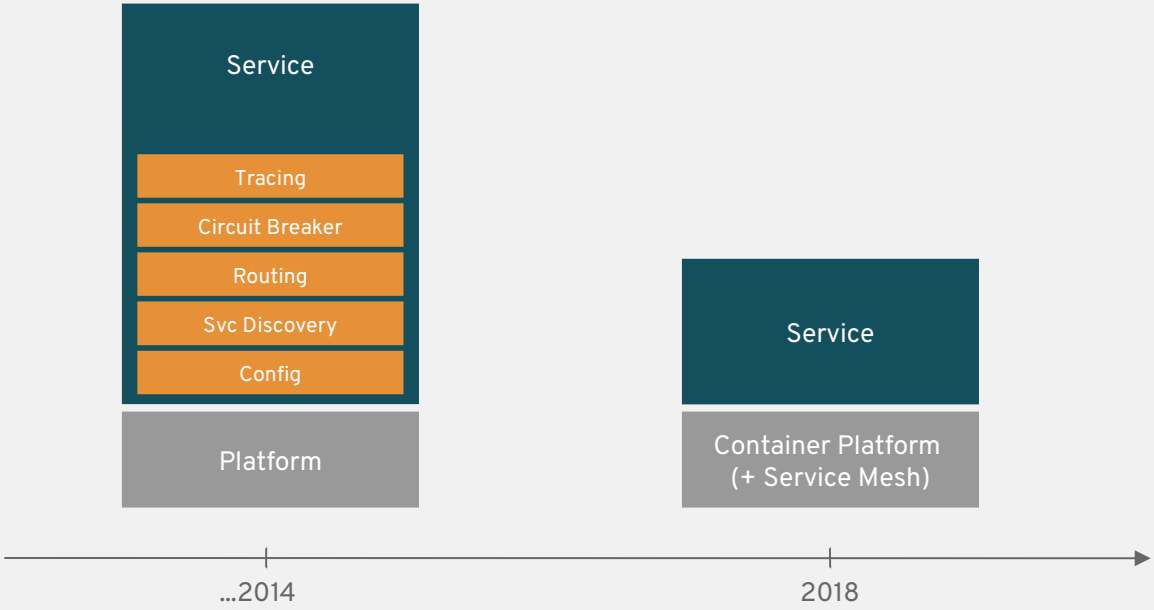
6. Security at Application Level

SSL at Ingress (with OpenShift Routes)

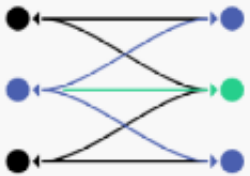


7. Application network security with Istio

Microservice Evolution



Service Mesh

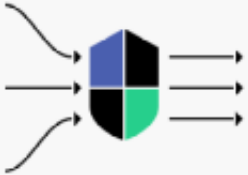


Connect

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

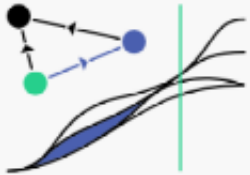
Secure

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



Control

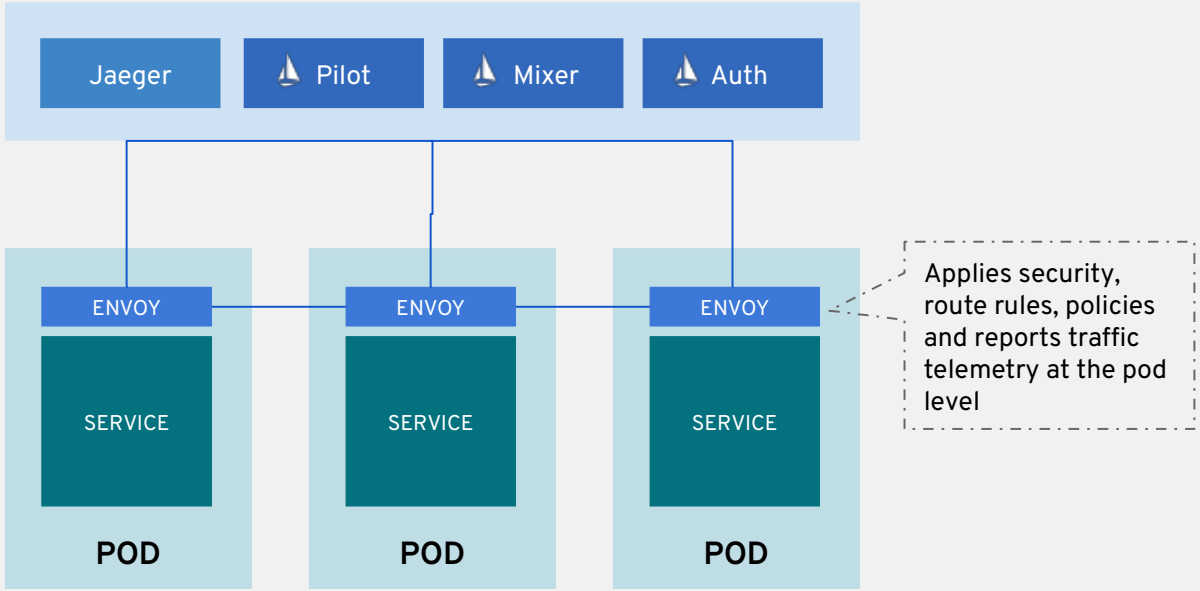
Apply policies and ensure that they're enforced, and that resources are fairly distributed among consumers.



Observe

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

Service Mesh Architecture



Questions?

THANK YOU


plus.google.com/+RedHat


facebook.com/redhatinc


linkedin.com/company/red-hat


twitter.com/RedHat

youtube.com/user/RedHatVideos