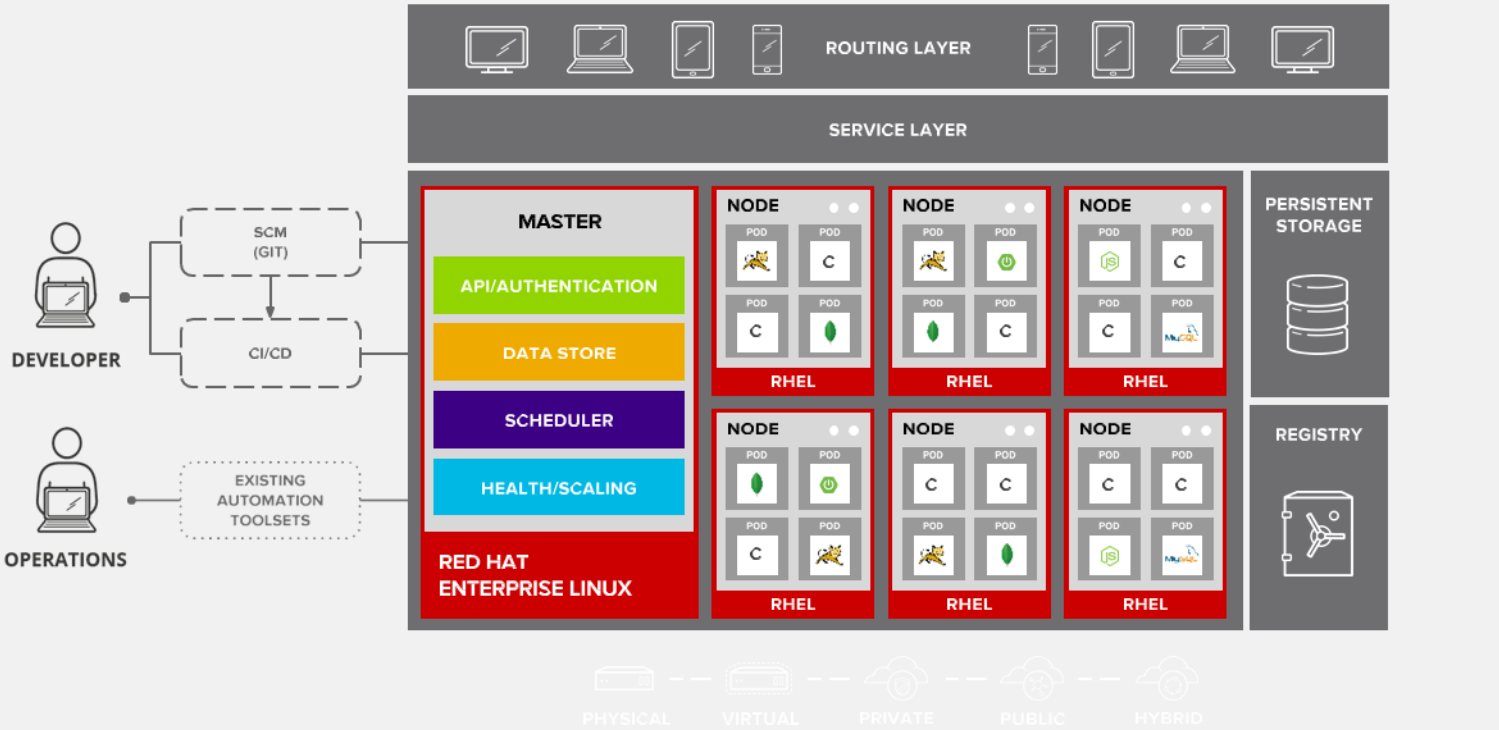

Kubernetes Basics
redhat.


Kubernetes Architecture
redhat.

Kubernetes Architecture




Core primitives

Kubernetes objects

Kubernetes contains a number of abstractions that represent the state of your system: deployed containerized applications and workloads, their associated network and disk resources, and other information about what your cluster is doing.

These abstractions are represented by objects in the Kubernetes API.

The basic Kubernetes objects include:

- Pod
- Volume
- Service
- Namespace

Pods

A Pod is the basic building block of Kubernetes–the smallest and simplest unit in the Kubernetes object model that you create or deploy. A Pod represents a running process on your cluster.

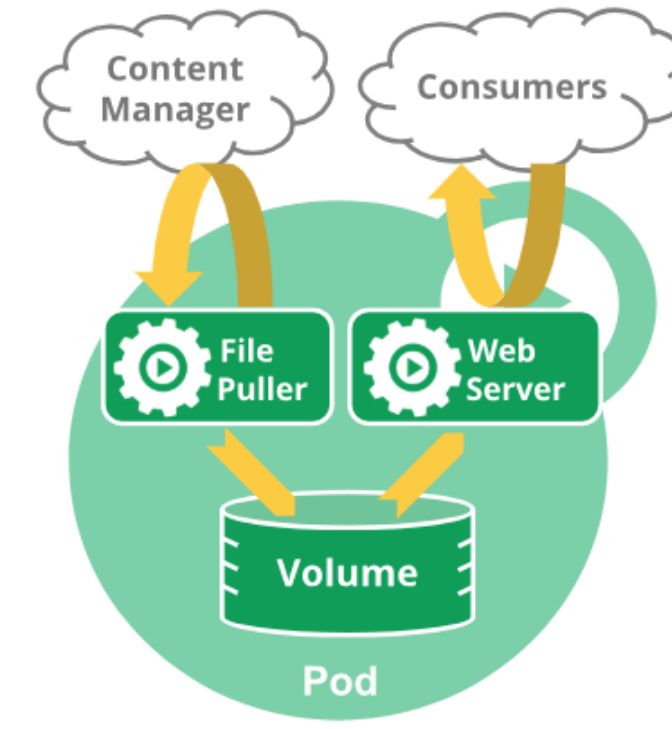
Small group of containers & volumes

Tightly coupled
- The atom of replication & placement

“Logical” host for containers
- Each pod gets an IP address
- Share data: localhost, volumes, IPC, etc.

Facilitates composite applications
- Mix and match components, languages, etc.
- Preserves 1:1 app to image

Example: data puller & web server



Volumes

First, when a Container crashes, kubelet will restart it, but the files will be lost - the Container starts with a clean state. Second, when running Containers together in a Pod it is often necessary to share files between those Containers.

Storage automatically attached to pod

- Local scratch directories created on demand
- Cloud block storage
  - GCE Persistent Disk
  - AWS Elastic Block Storage
- Cluster storage
  - File: NFS, Gluster, Ceph
  - Block: iSCSI, Cinder, Ceph
- Special volumes
  - Git repository
  - Secret

Critical building block for higher-level automation

Services

Kubernetes Pods are mortal. They are born and when they die, they are not resurrected. ReplicaSets in particular create and destroy Pods dynamically

A group of pods that work together

- grouped by a label selector

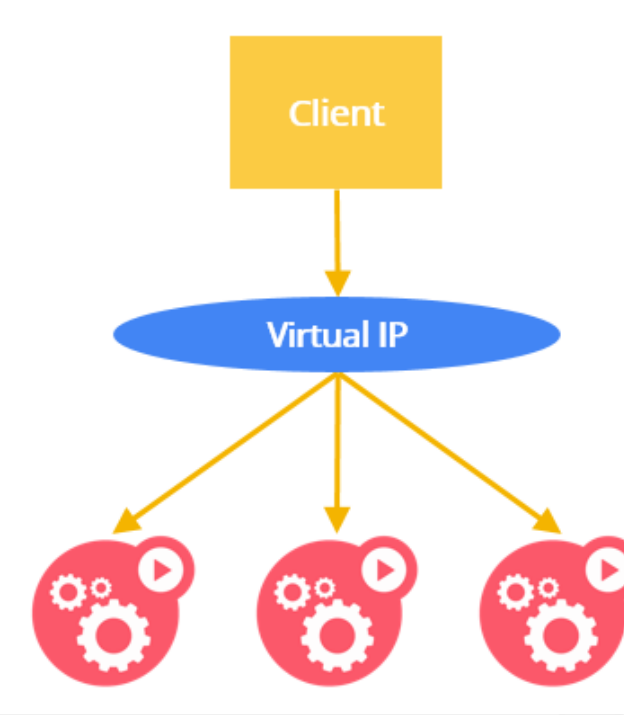
Publishes how to access the service

- DNS name
- DNS SRV records for ports (well known ports work, too)
- Kubernetes Endpoints API

Defines access policy

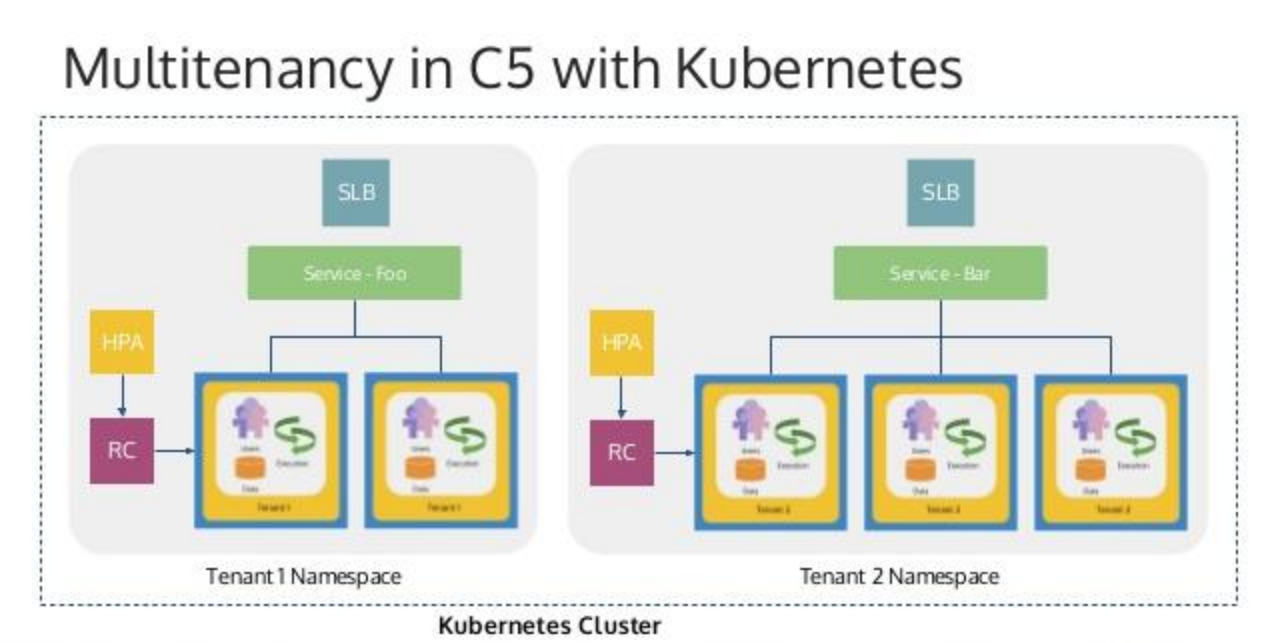
- Load-balanced: name maps to stable virtual IP
- “Headless”: name maps to set of pod IPs

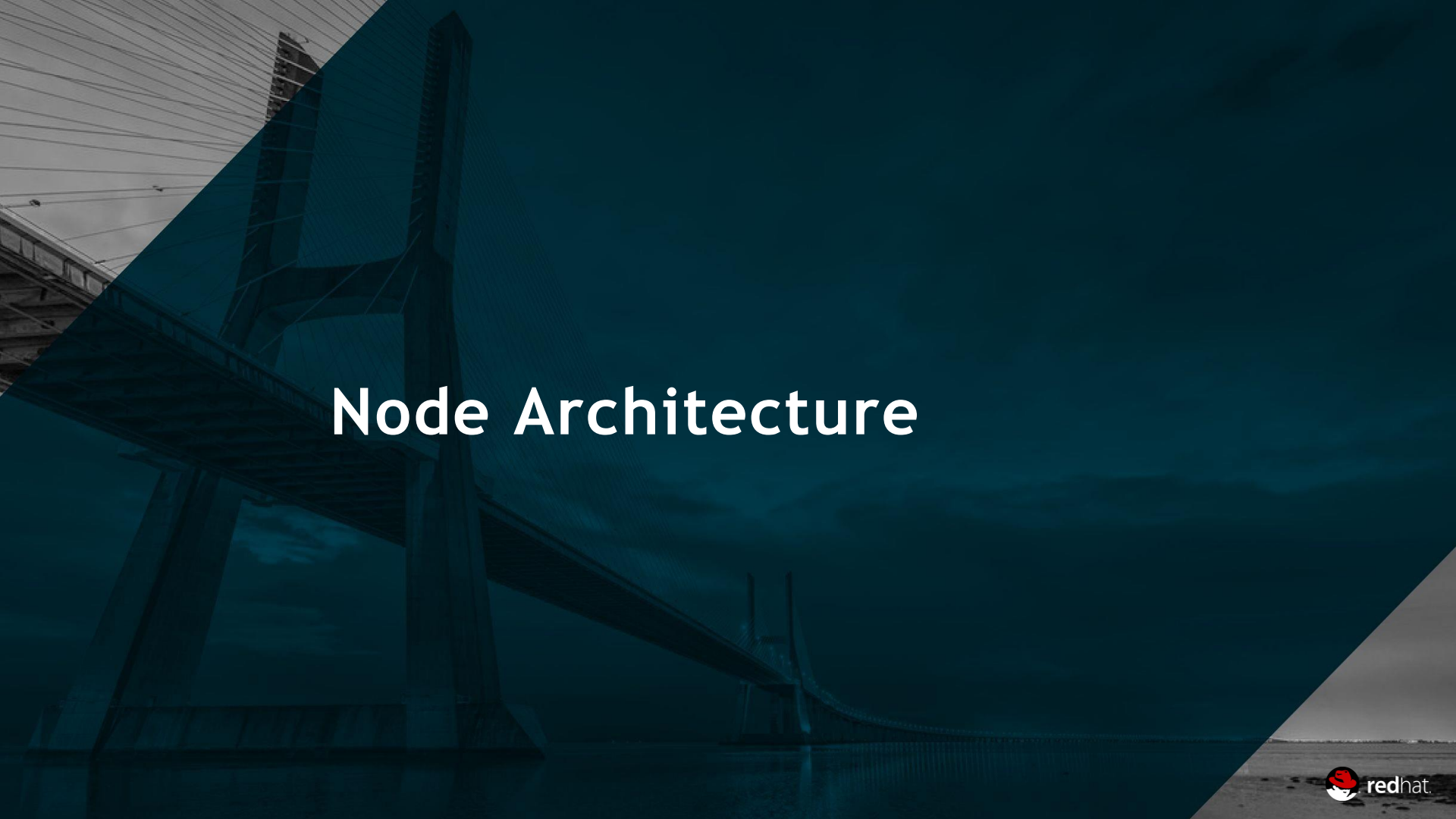
Decoupled from Pods and ReplicationControllers



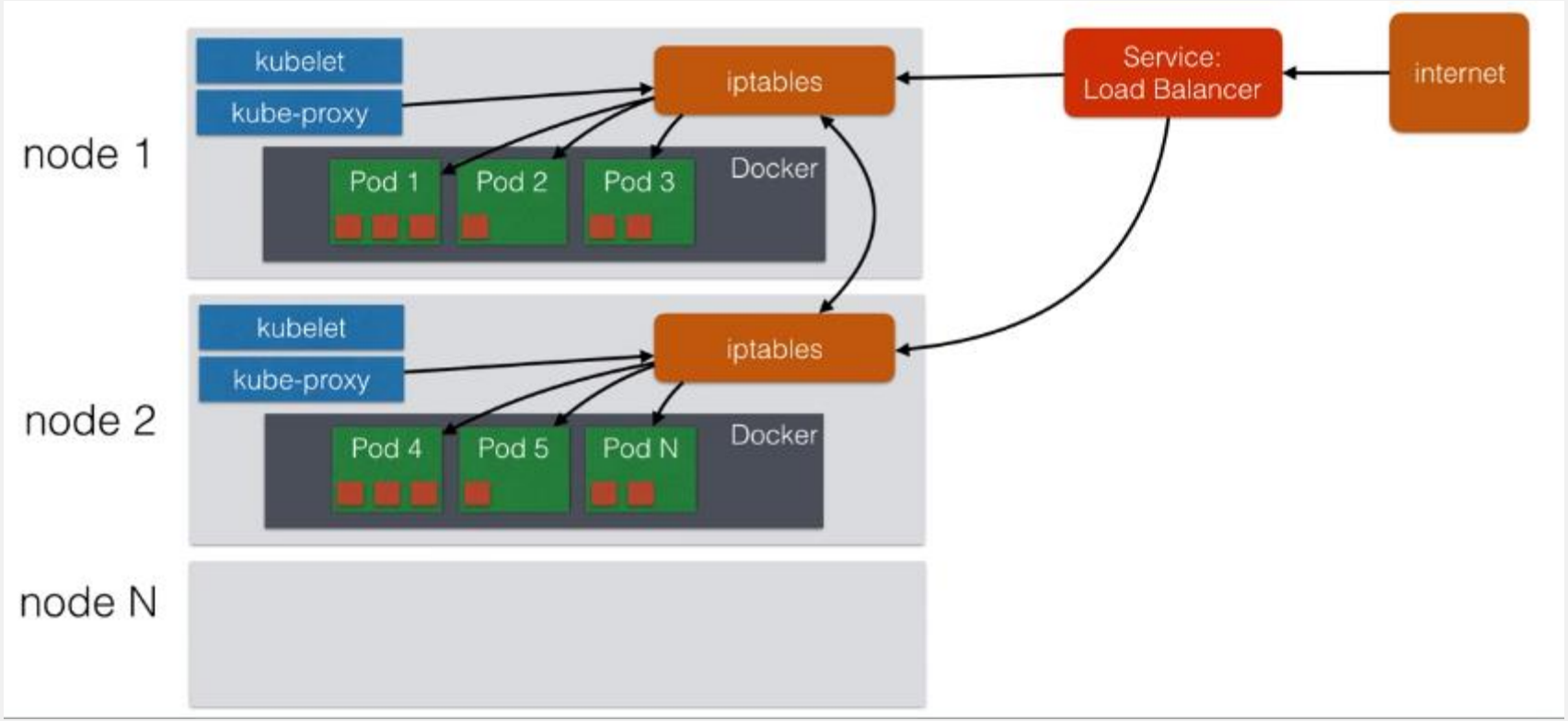
Namespaces

Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called namespaces.

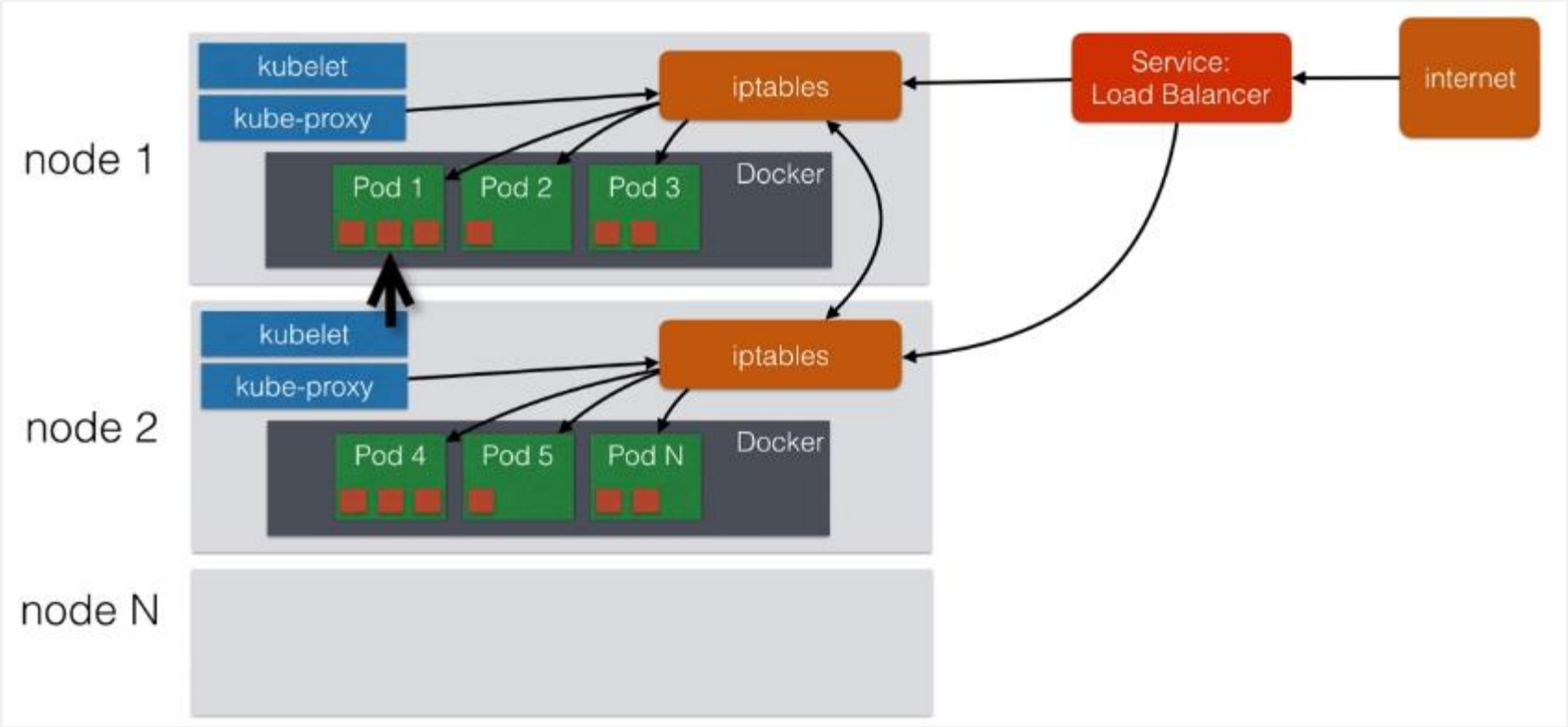



Node Architecture

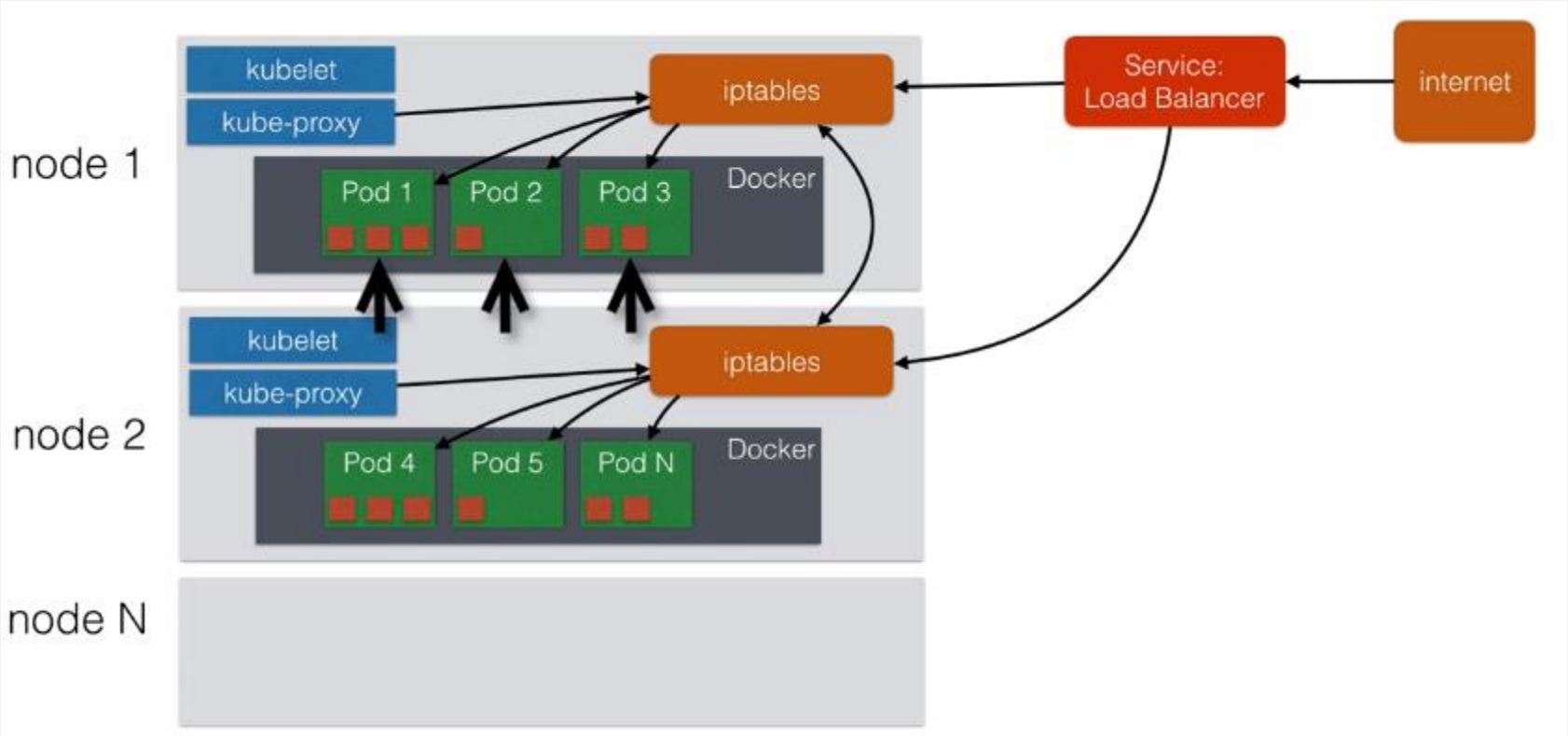
Architecture overview



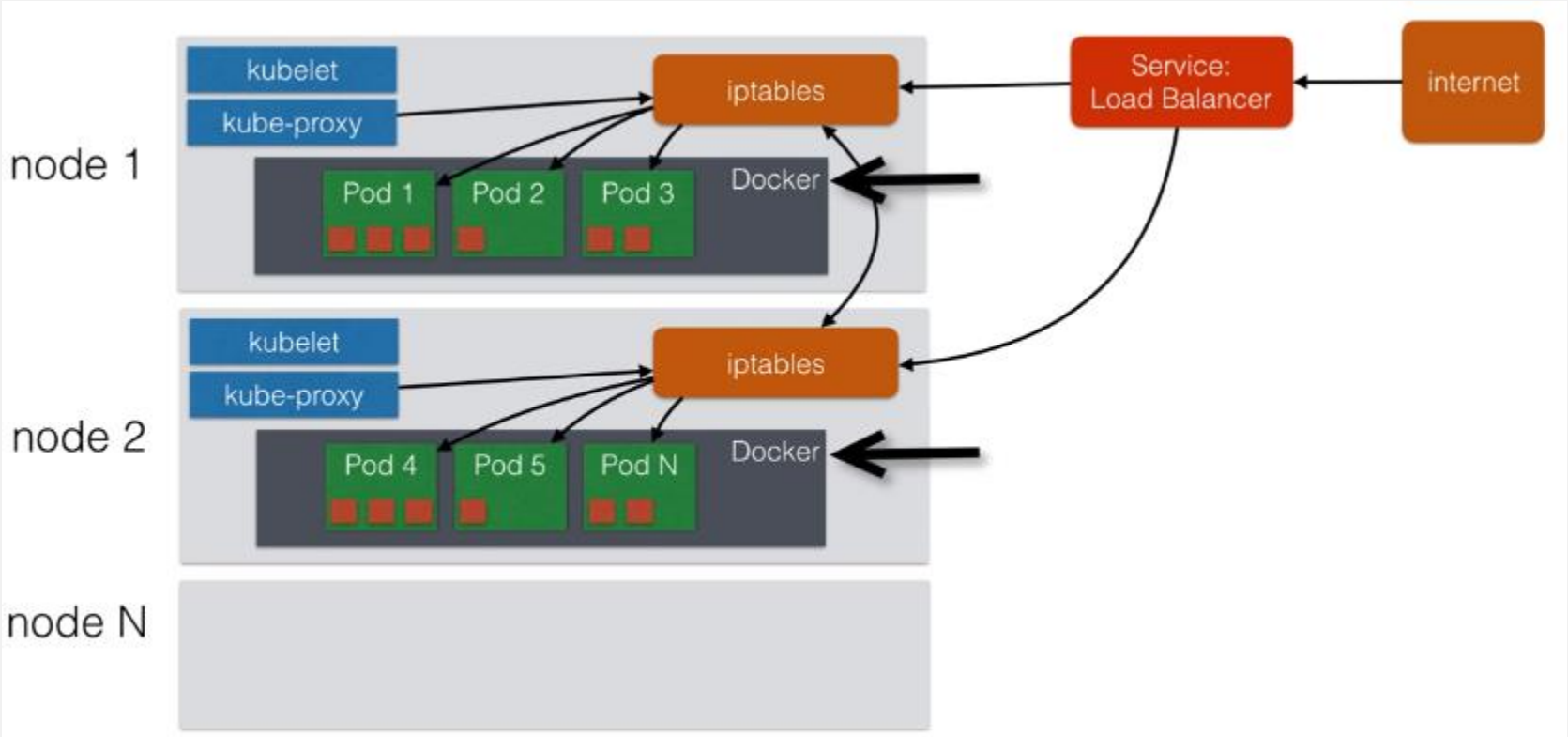
Architecture overview



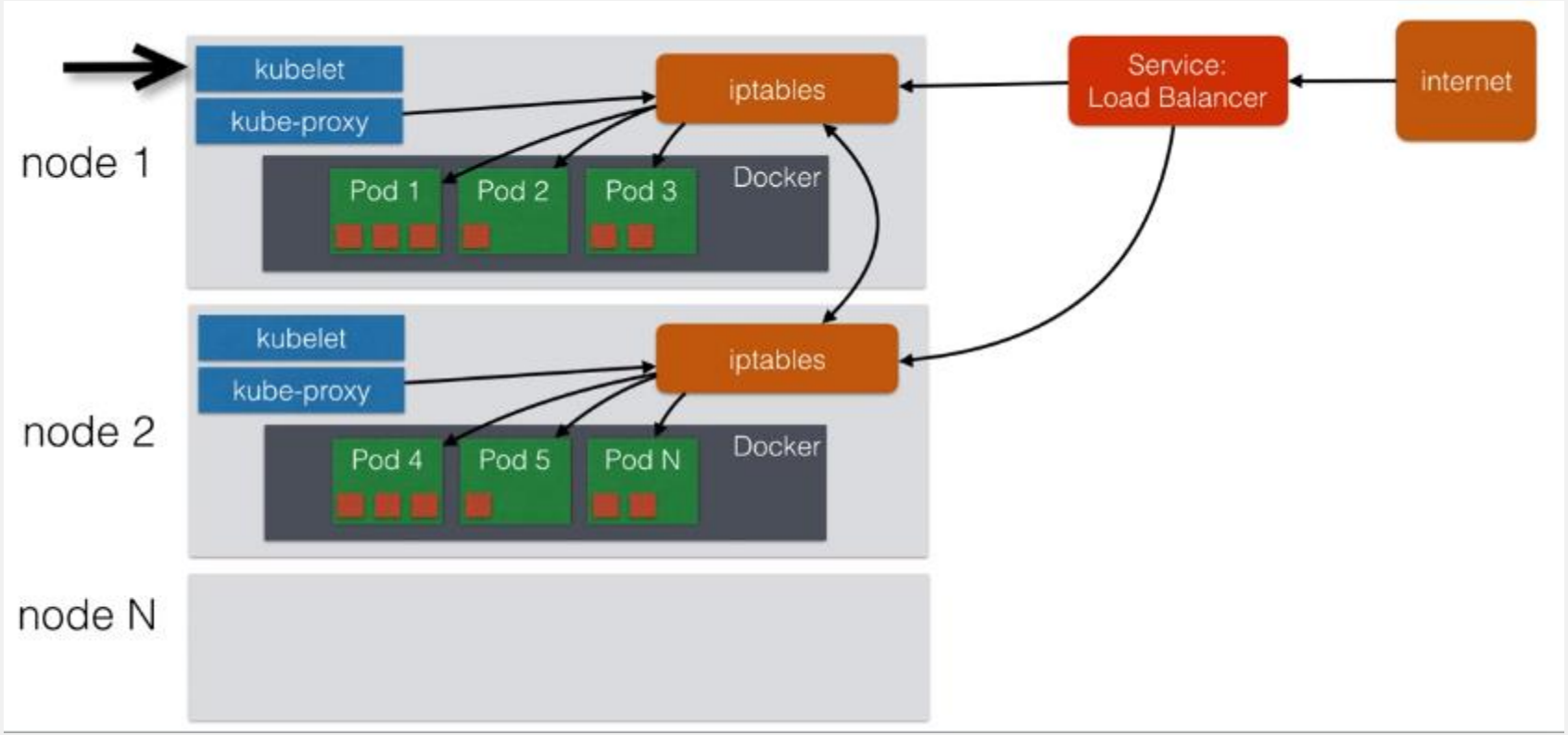
Architecture overview



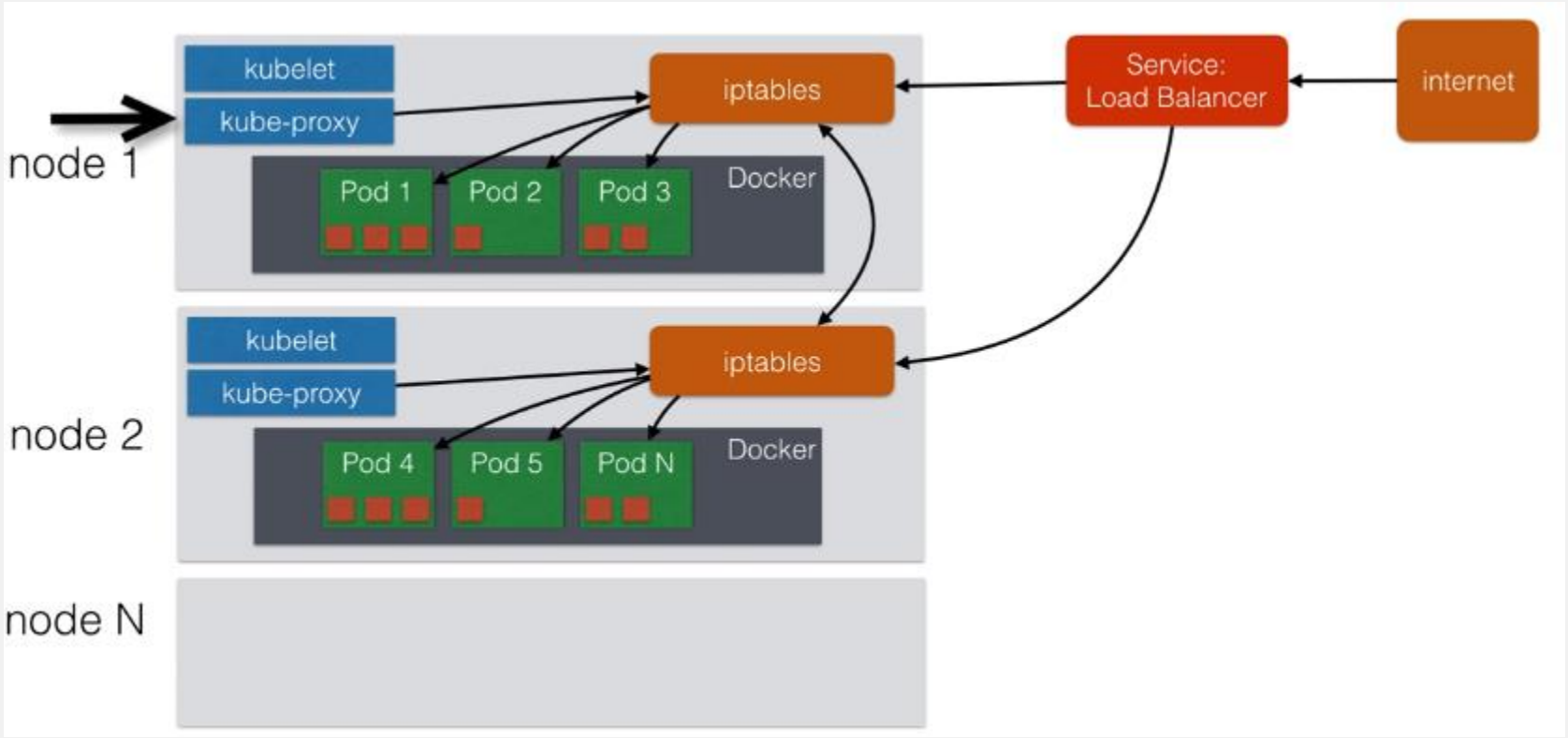
Architecture overview



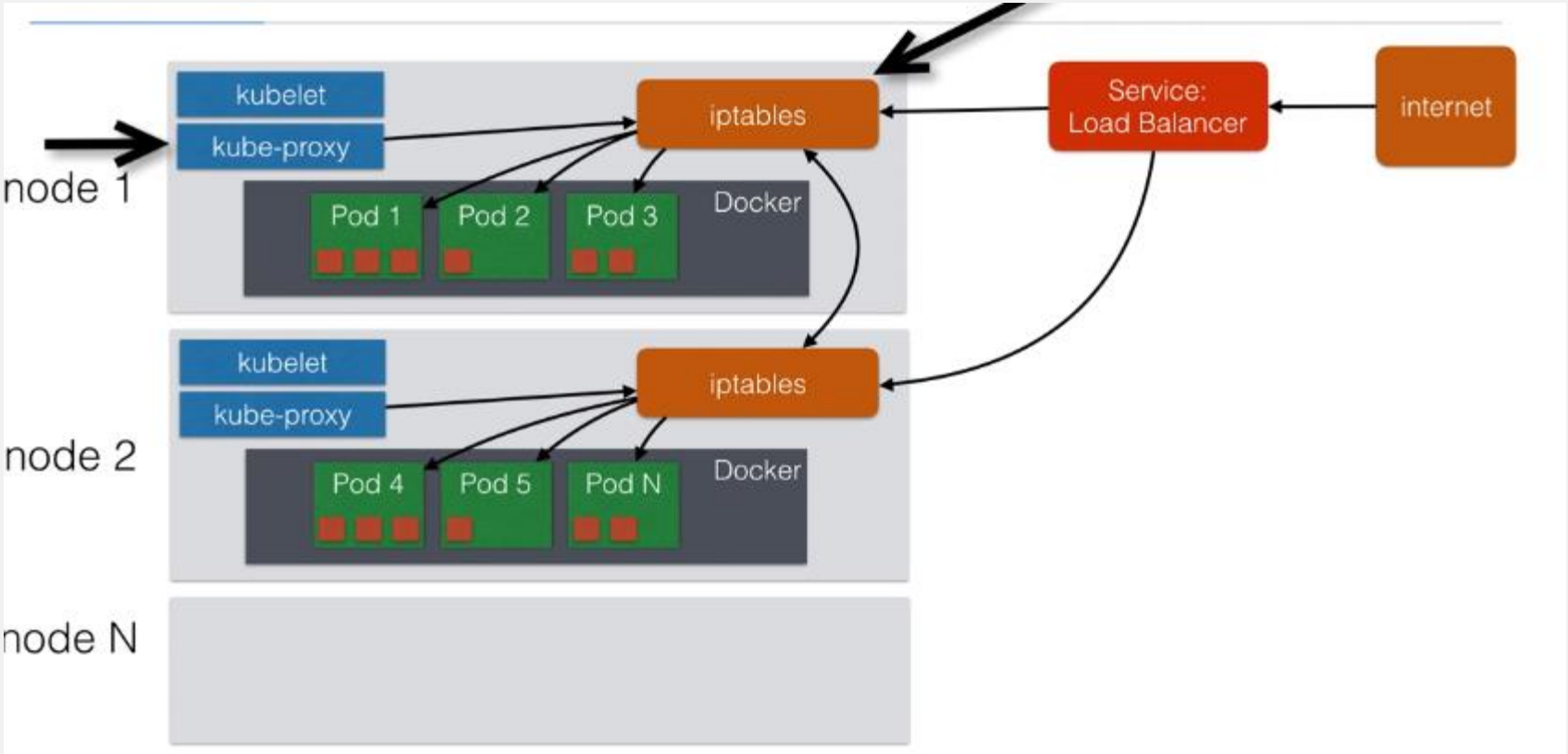
Architecture overview



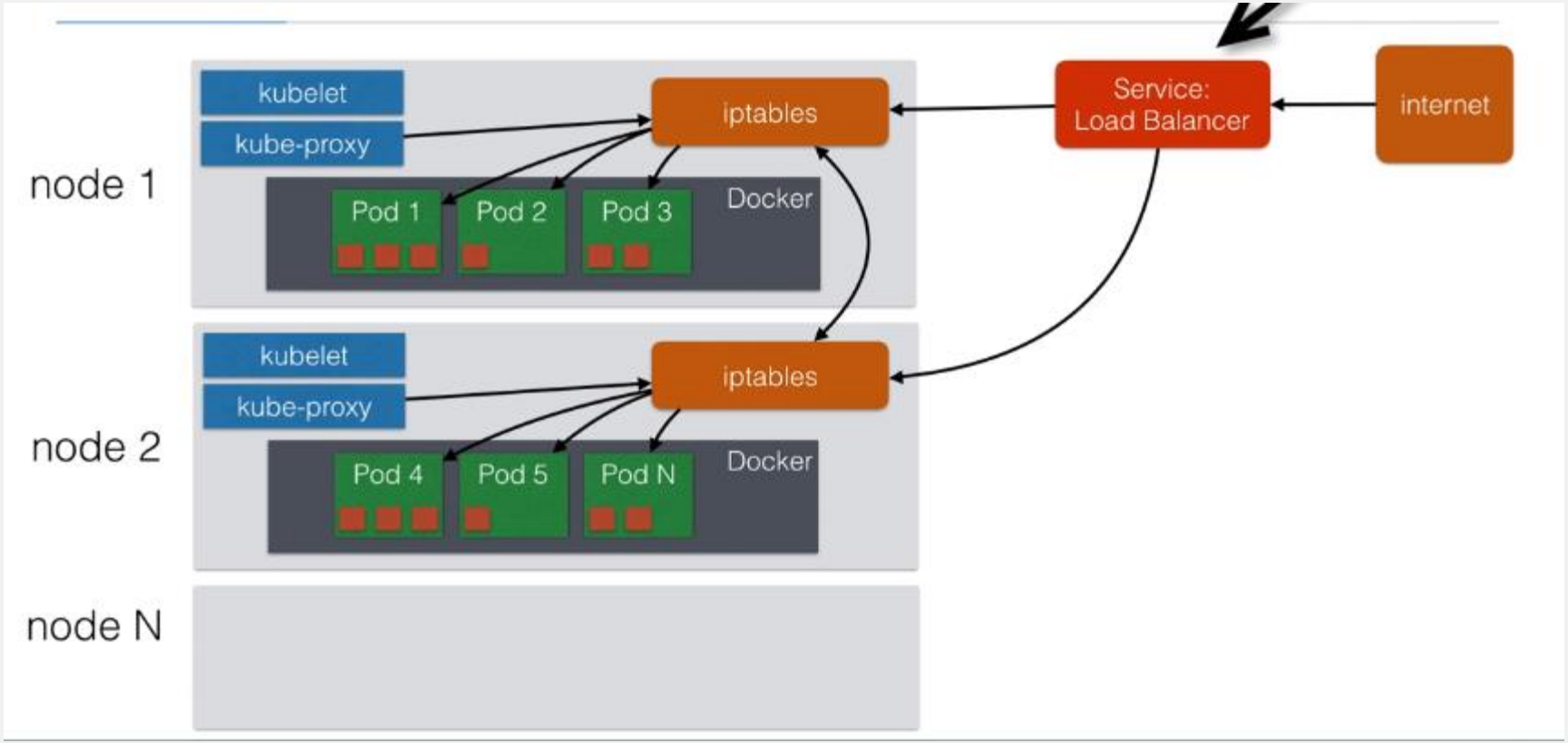
Architecture overview



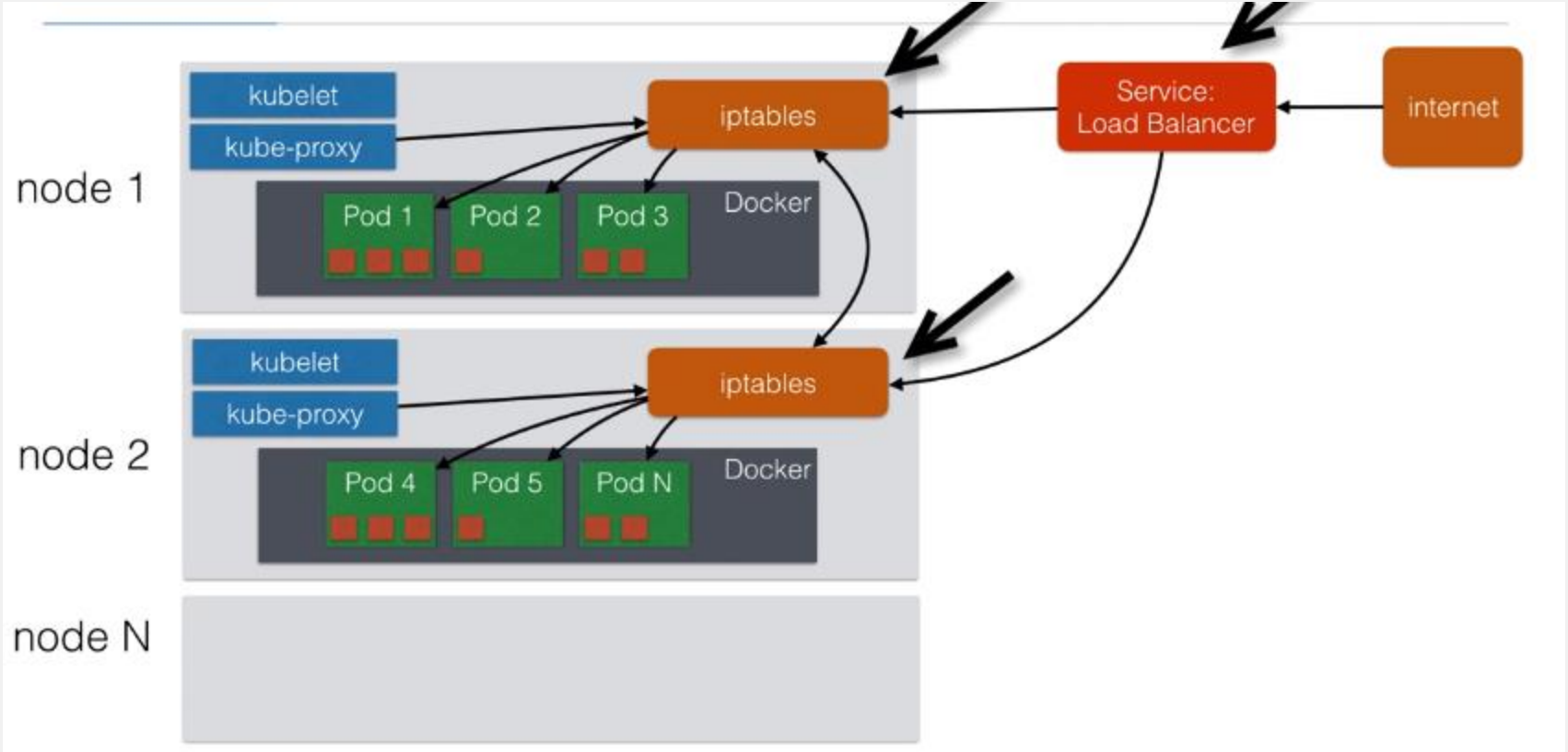
Architecture overview



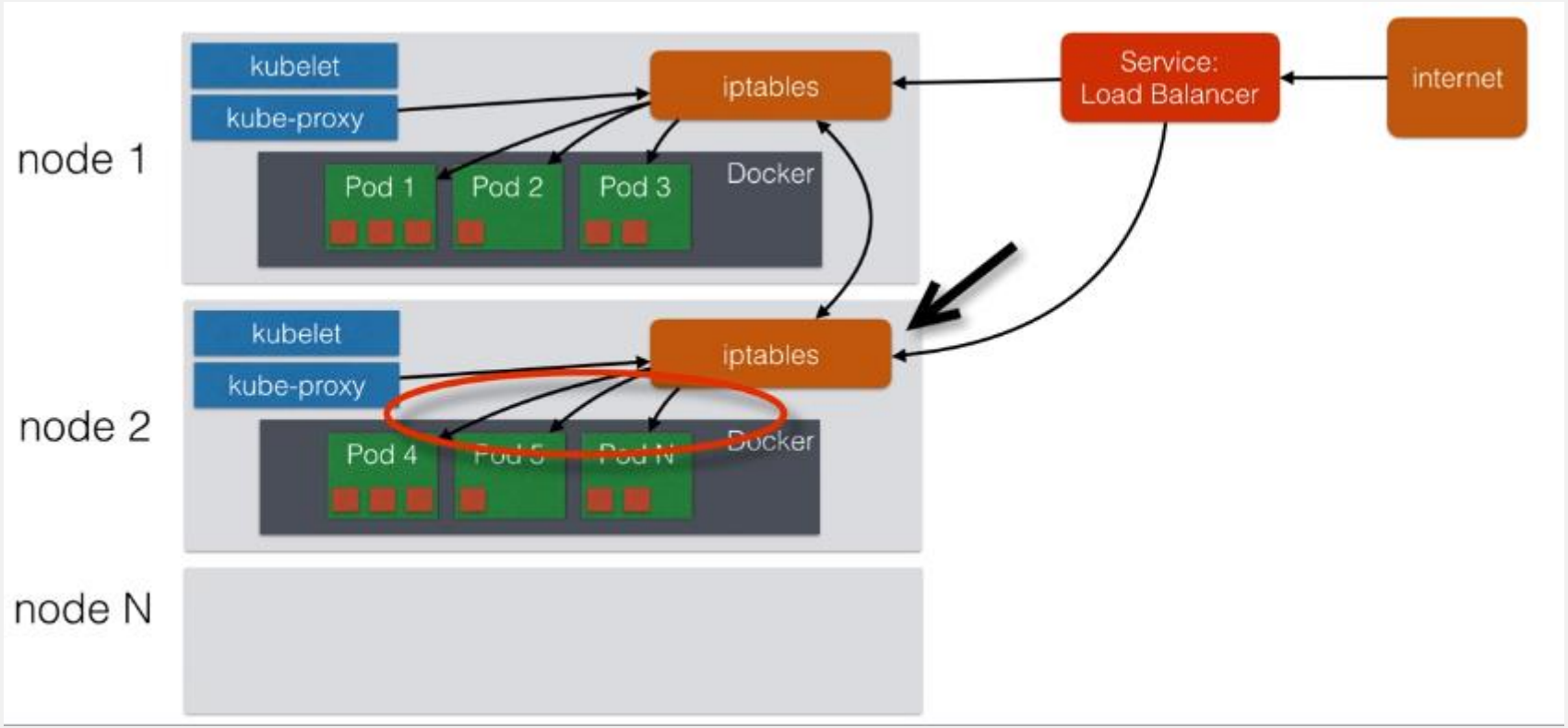
Architecture overview



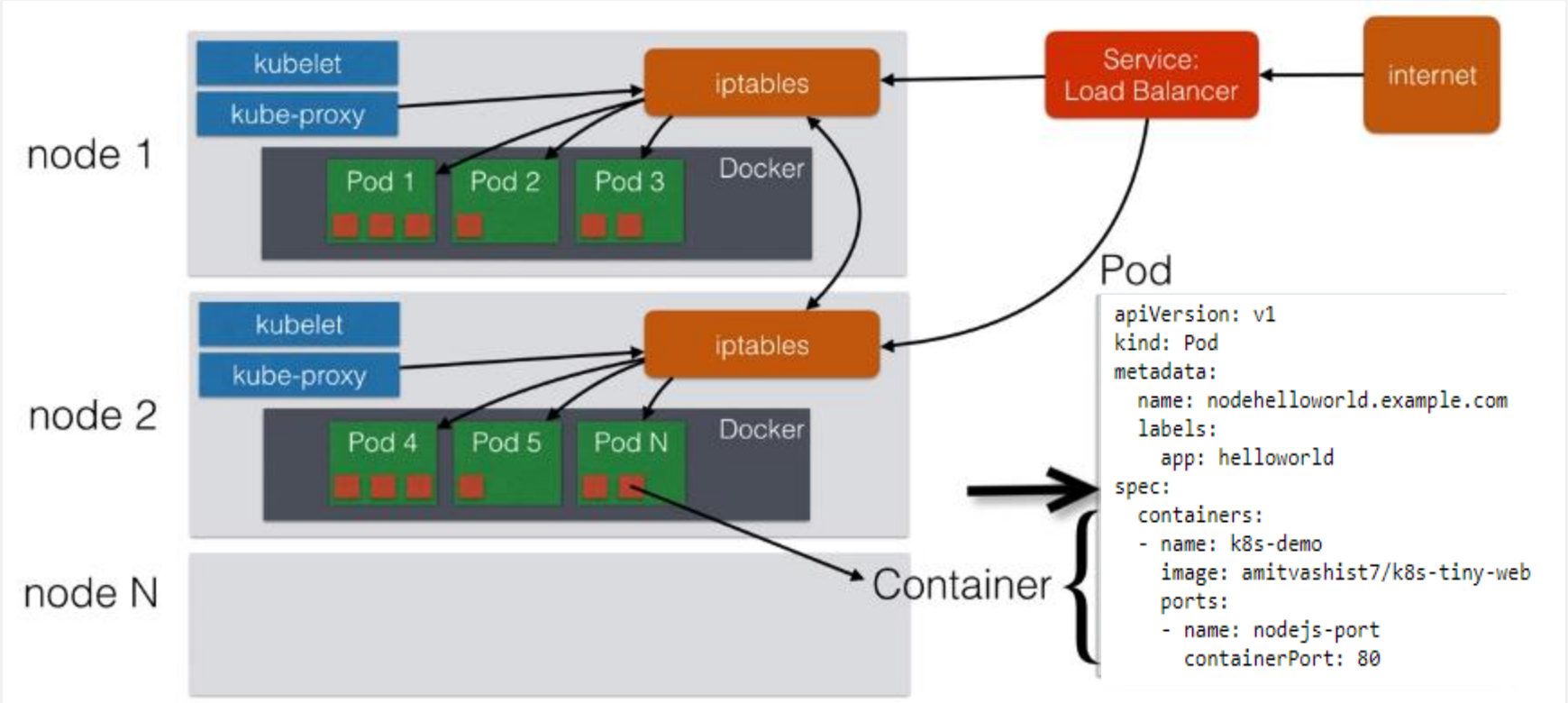
Architecture overview



Architecture overview



Architecture overview



Scaling pods

Scaling

- If your application is **stateless** you can horizontally scale it
  - Stateless = your applicate doesn't have a **state**, it doesn't **write** any **local file** / keeps local session.
  - All traditional database (MySQL, Postgres) are **stateful**, they have database files that can't be split over multiple session.
- Most **Web Application** can be made stateless:
  - **Session Management** needs to be done outside the container (MemCache, Redis etc)
  - Any files that need to be saved **can't be saved locally** on the container

Scaling

- Our example app is **stateless**, if the same app would run multiple times, it doesn't change state.
- Later in this course I'll explain how to use **volumes** to still run stateful apps
  - Those stateful apps can't horizontally scale, but you can run them in single container & vertically scale (allocate more CPU/ Memory/ Disk)

Scaling

- Scaling in Kubernetes can be done using the **Replication Controller**
- The Replication Controller will **ensure** a specified number of **pod replicas** will run at all times
- A Pods created with Replication Controller will **automatically** be **replaced** if they fail, get deleted, or are terminated.
- Using the Replication Controller is also **recommended** if you just want to make sure **1 pod** is always running, even after reboot.
    - You can then run a replication controller with just **1 replica**
    - This makes sure that the pod is always running.

Scaling

- To Replicate our example app 2 times

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: helloworld-controller
spec:
  replicas: 2
  selector:
    app: docket-get-started
  template:
    metadata:
      labels:
        app: docket-get-started
    spec:
      containers:
      - name: docket-get-started
        image: amitvashist7/get-started:part2
        ports:
        - name: nodejs-port
          containerPort: 80
```

Demo Placeholder

Horizontally scale a pod with replication controller


Deployments

Replica Set

- **Replica set** is the next-get Replication Controller
- It support a new selector that can do the selection based on **filtering** according a **set of values**
  - Example: Environnent Ethier "Dev" or "QA«
  - not only based on equality, like replication controller
    - e.g "Environment" == "Dev“
- This **Replica Set**, rather than the replication controller, is used by the Deployment Object

Deployment

- A Deployment declaration in Kubernetes allows you to do app **deployments** and **updates**
- When using the deployment object, you define the **state** of your application
  - Kubernetes will then make sure the cluster matches your **desired** state

- Just using the **replication-controller** or **replication set** might be **cumbersome** to deploy apps
  - The **Deployment Object** is easier to use and gives you more possibilities

Deployment

- With a deployment object you can:
  - **Create** a deployment (e.g. deploying an app)
  - **Update** a deployment (e.g. deploying an new version)
  - Do **rolling updates** (Zero downtime deployment)
  - **Roll Back** to previous version
  - **Pause/Resume** a deployment (e.g. to roll-out to only certain percentage)

Deployment

- This is an example of a deployment

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: helloworld-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: helloworld
    spec:
      containers:
      - name: k8s-demo
        image: amitvashist7/k8s-tiny-web
        ports:
        - name: node-port
          containerPort: 80
```

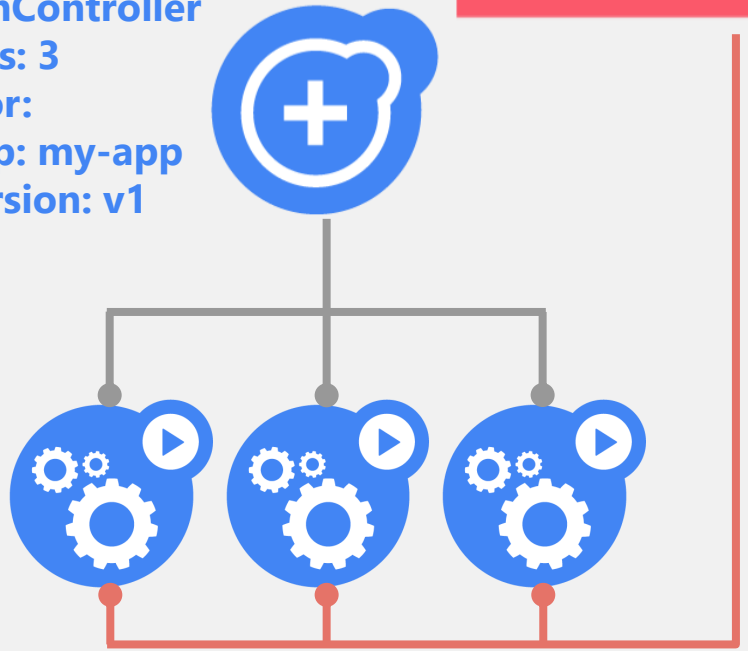
Useful Commands

Command	Description
kubectl get deployment	Get info on current deployments
kubectl get rs	Get info about the replica sets
kubectl get pods --show-labels	Get info about the labels attached to those pods
kubectl rollout status deployment/helloworld-deployment	Get deployment status
kubectl set image deployment/helloworld-deployment k8s-demo=k8s-demo:2	Run k8s-demo with image label version
kubectl edit deployment/helloworld-deployment	Edit the deployment Object
kubectl rollout history deployment/helloworld-deployment	Get the rollout history
kubectl rollout undo deployment/helloworld-deployment	Rollback to pervious version
kubectl rollout undo deployment/helloworld-deployment --to-revision=n	Rollback to any pervious version

Rolling Updates

Service
- **app: my-app**

ReplicationController
- **replicas: 3**
- **selector:**
  - **app: my-app**
  - **version: v1**



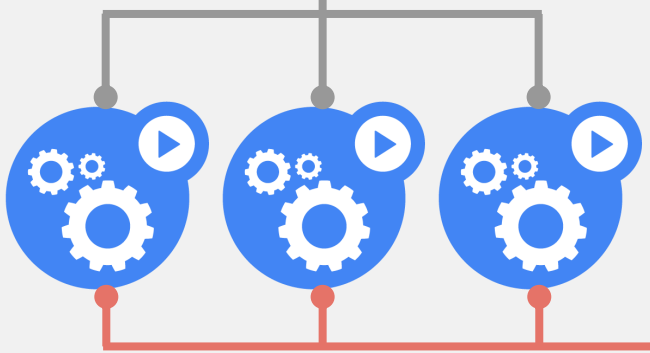
Live-update an application

```
$ kubectl rolling-update \
   my-app-v1 my-app-v2 \
   --image=image:v2
```

Rolling Updates

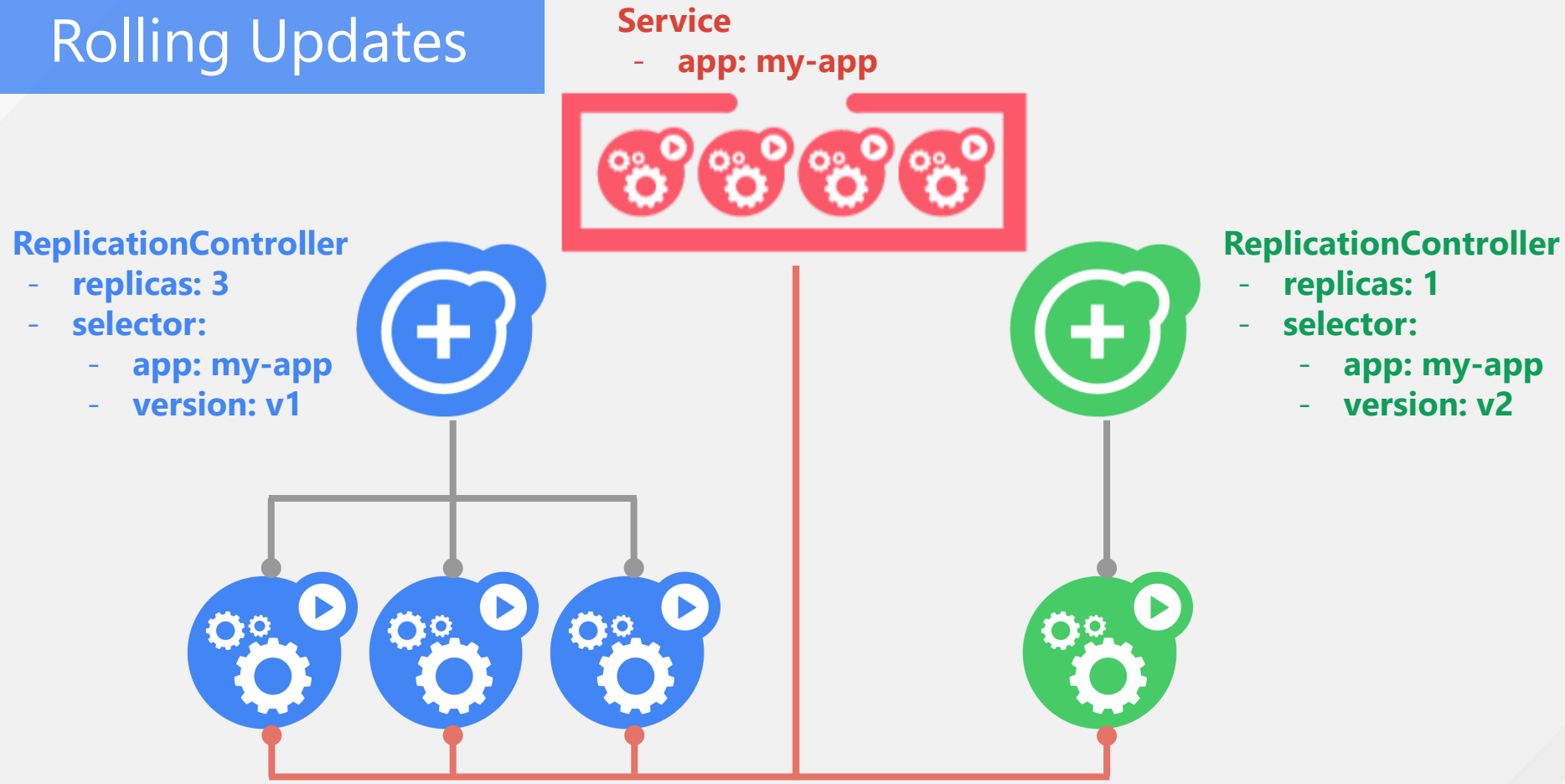
Service
- app: my-app

ReplicationController
- replicas: 3
- selector:
  - app: my-app
  - version: v1

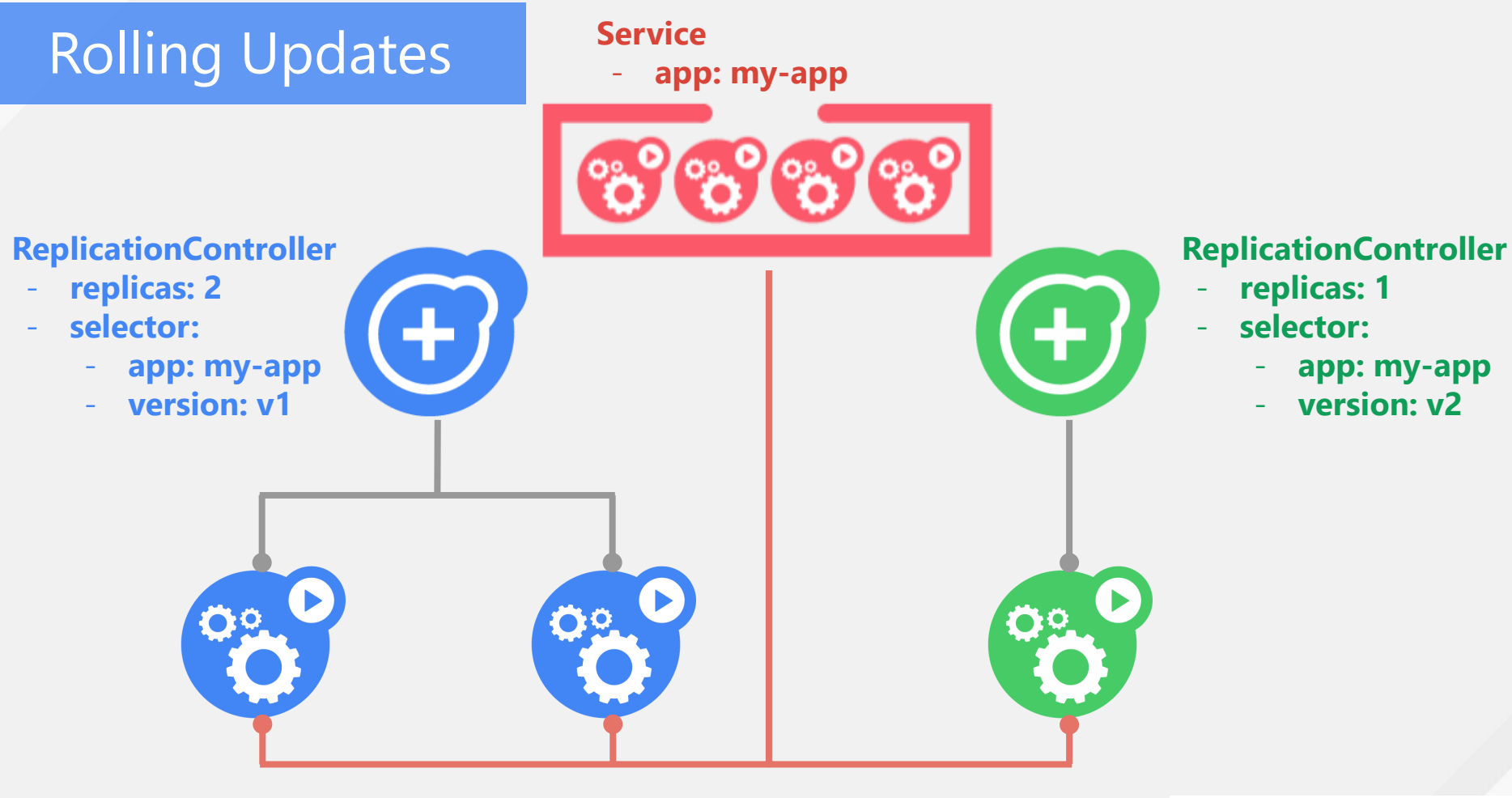


ReplicationController
- replicas: 0
- selector:
  - app: my-app
  - version: v2

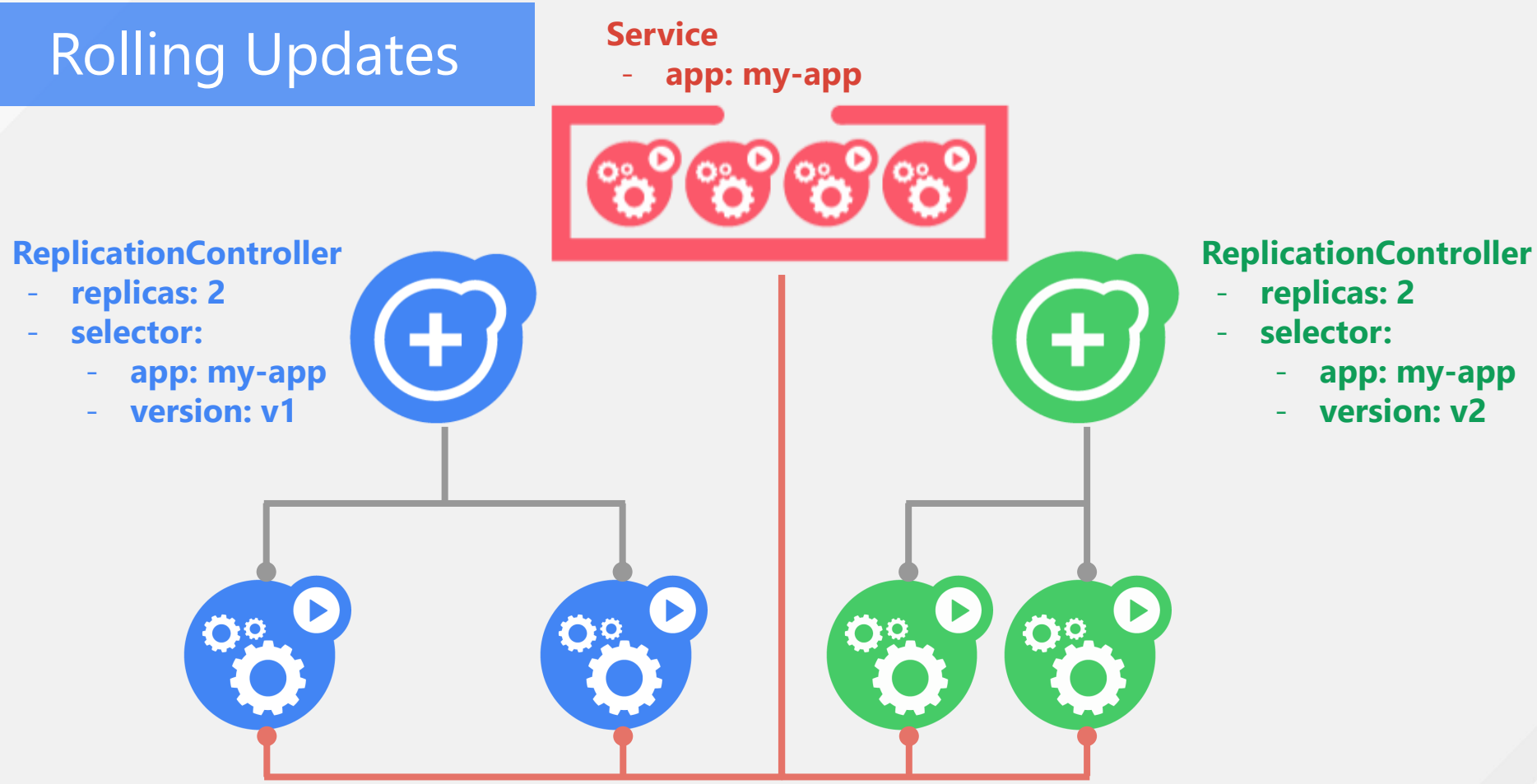
Rolling Updates



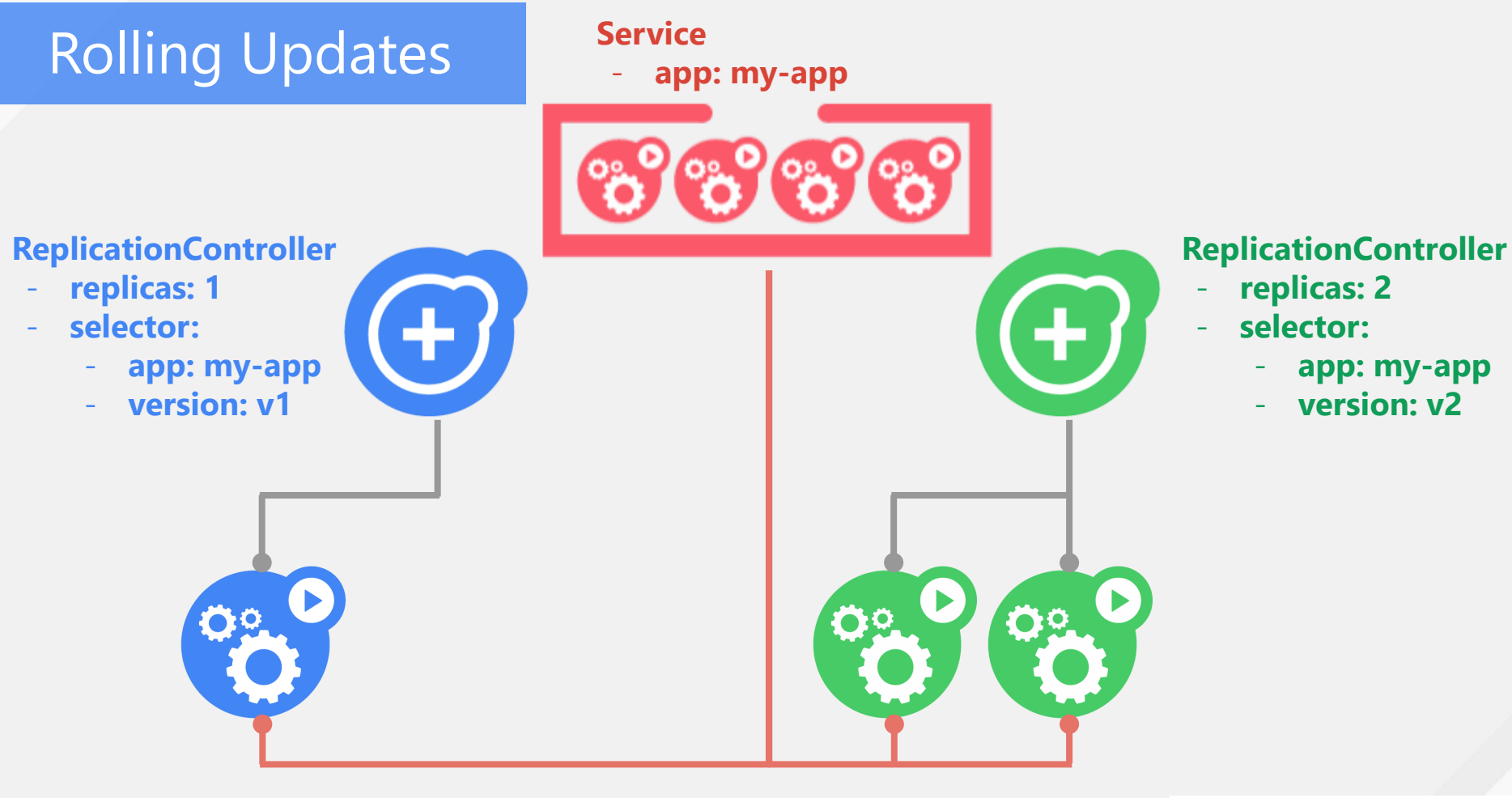
Rolling Updates



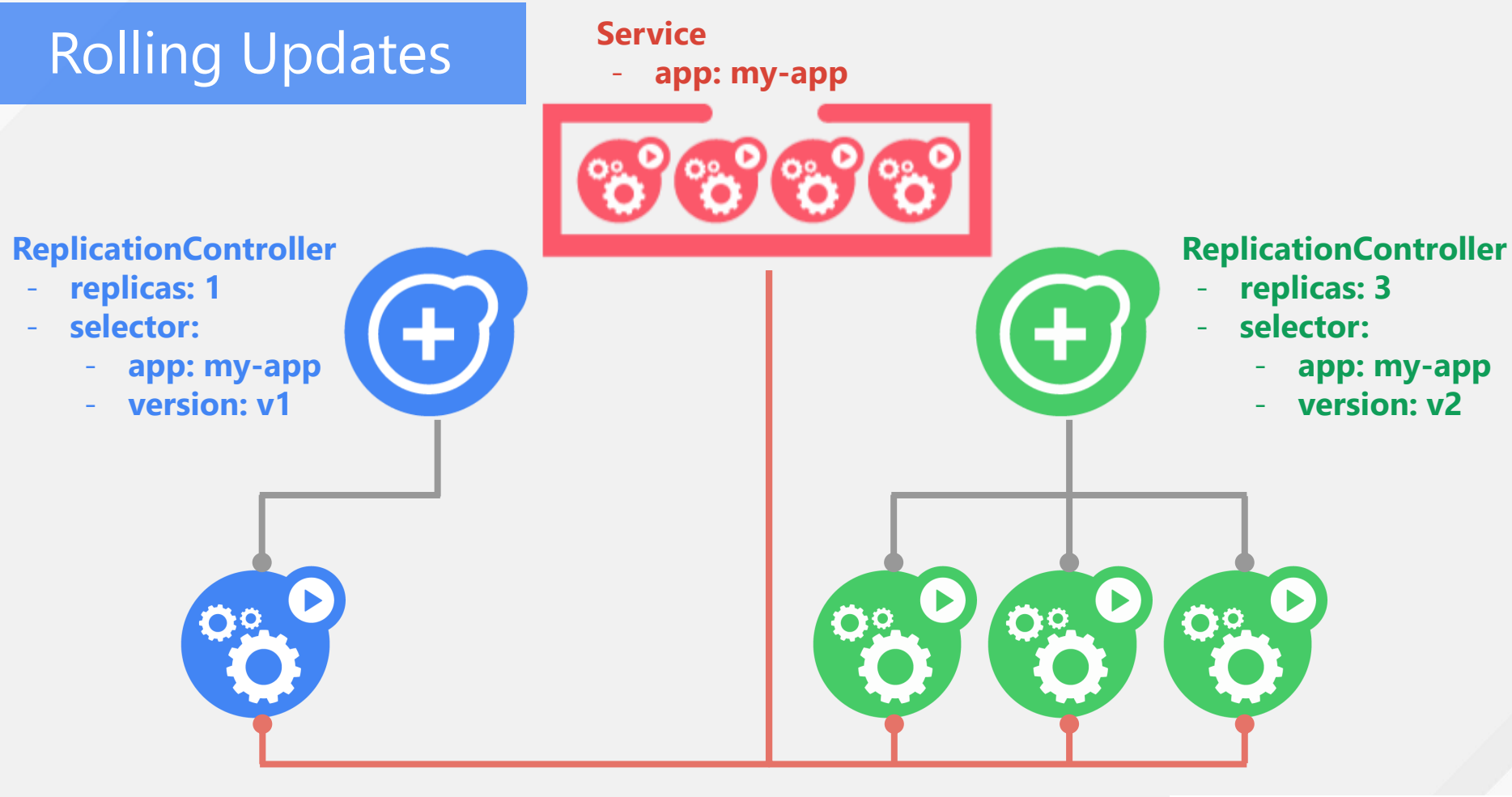
Rolling Updates



Rolling Updates



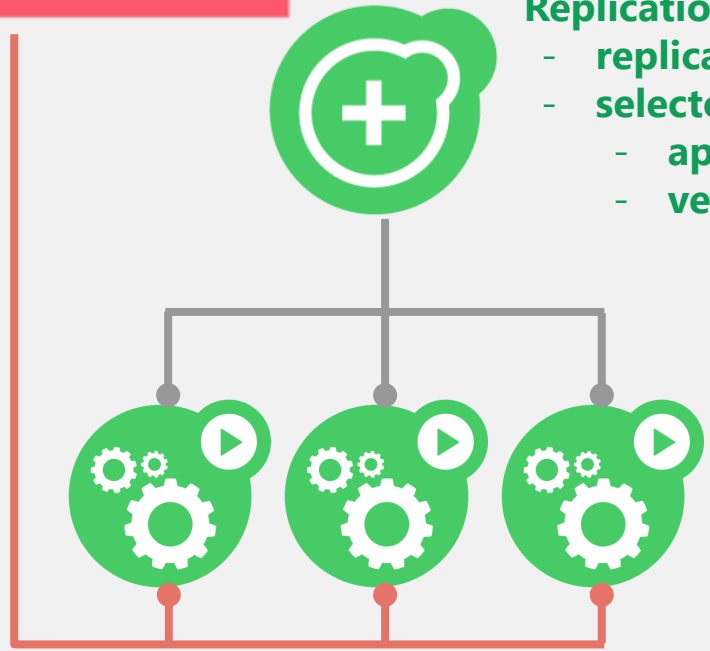
Rolling Updates



Rolling Updates

Service
- app: my-app

ReplicationController
- replicas: 0
- selector:
  - app: my-app
  - version: v1



ReplicationController
- replicas: 3
- selector:
  - app: my-app
  - version: v2

Demo Placeholder

A deployment

Services

Services

- **Pods** are very **dynamic**, they come and go on the kubernetes cluster
  - When using a **replication-controller**, pods are **terminated** and created during scaling operations
  - When using **Deployments**, when **updating** the image version, pods are **terminated** and new pods take the place of older pods
- That's why pods should never be accessed directly, but always through a **Service**
- A Service is **logical bridge** between the "mortal" pods and other **services** or **end-users**

Services

- When using the "kubectl expose" command earlier, you created a new service for your pod, so it could be accessed externally
- Create a service will create an endpoints for your pod(s)
  - A **ClusterIP**: a virtual IP address only reachable from within the cluster. (this is default)
  - A **Nodeport**: a port that is the same on each node that is also reachable externally
  - A **LoadBalancer**: a LoadBalancer created by the cloud provider that will route the external traffic to every node on the NodePort (ELB on AWS)

Services

- The option just shown only allow you to create **virtual IPs** or **Ports**
- There is also a possibilities to use **DNS Name**
  - **ExternalName** can provide a DNS name for the service
  - e.g for service discovery using DNS
  - This only works when the **DNS add-on** is enabled
- I will discuses this later in a separate lecture

Services

- This is an example of a Service definition (also created using kubectl expose):

```
---------------------------------------
apiVersion: v1
kind: Service
metadata:
  name: helloworld-service
spec:
  ports:
  - port: 31001
    nodePort: 31001
    targetPort: nodejs-port
    protocol: TCP
  selector:
    app: helloworld
  type: NodePort
---------------------------------------
```

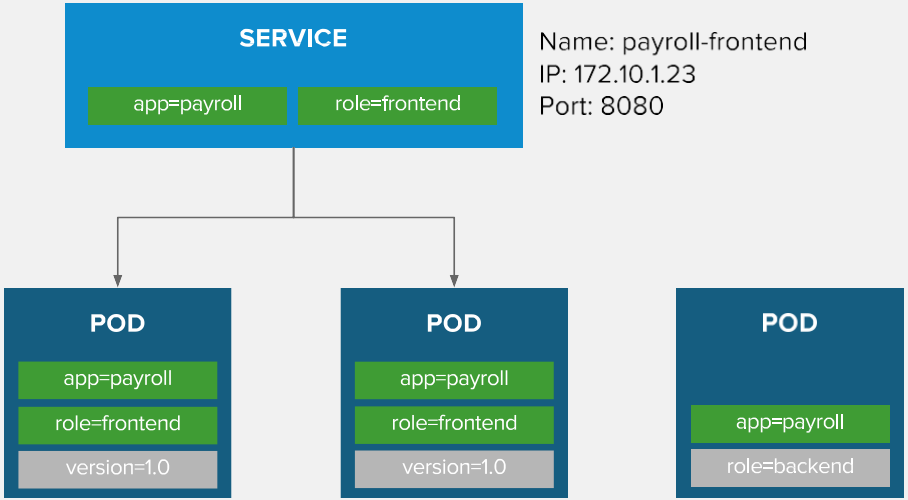
- **Note:** By default service can run only between port 30000 - 32767, but you could change this behavior by adding the --service-node-port-range= argument to the kube-apiserver (in the init scripts)

Demo Placeholder

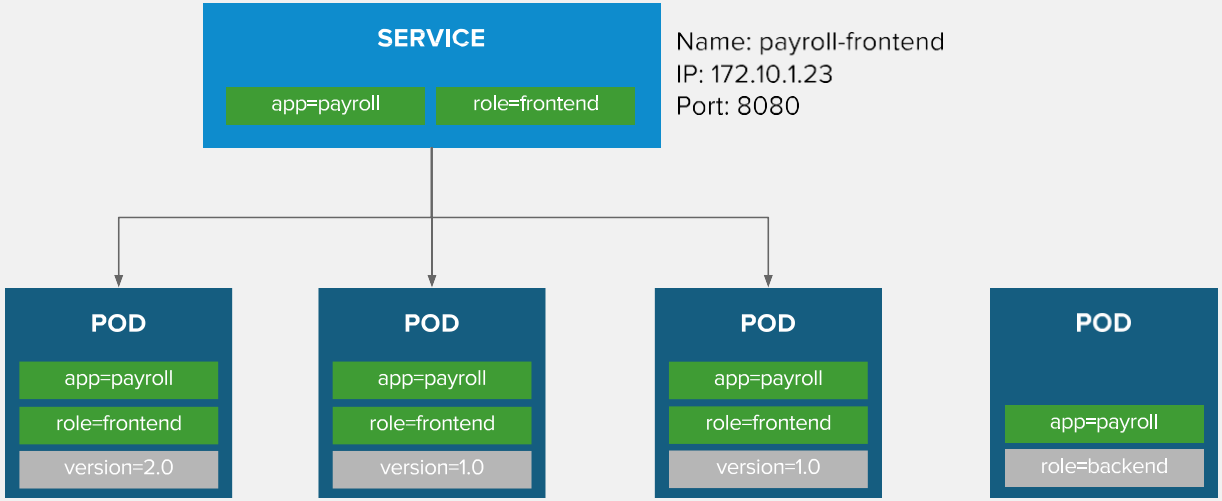
A new service


NETWORKING
redhat.

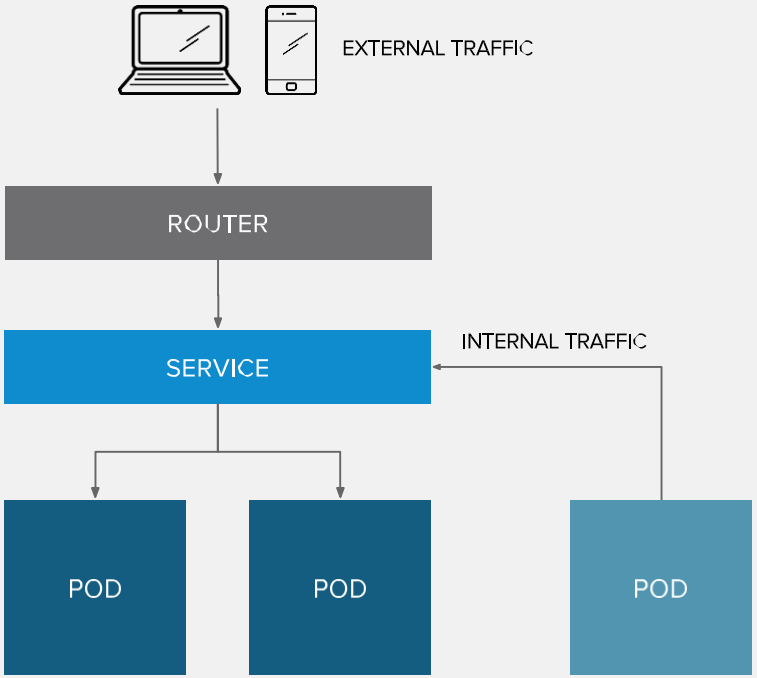
BUILT-IN SERVICE DISCOVERY INTERNAL LOAD-BALANCING



BUILT-IN SERVICE DISCOVERY INTERNAL LOAD-BALANCING

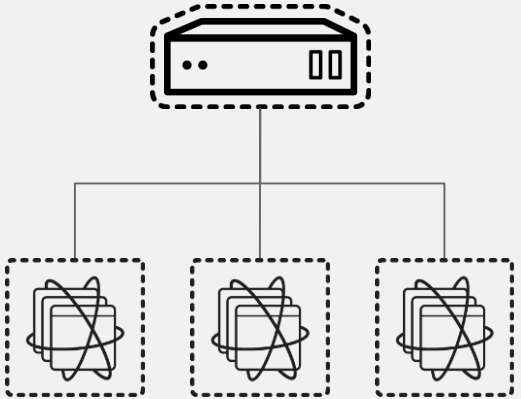


ROUTE EXPOSES SERVICES EXTERNALLY



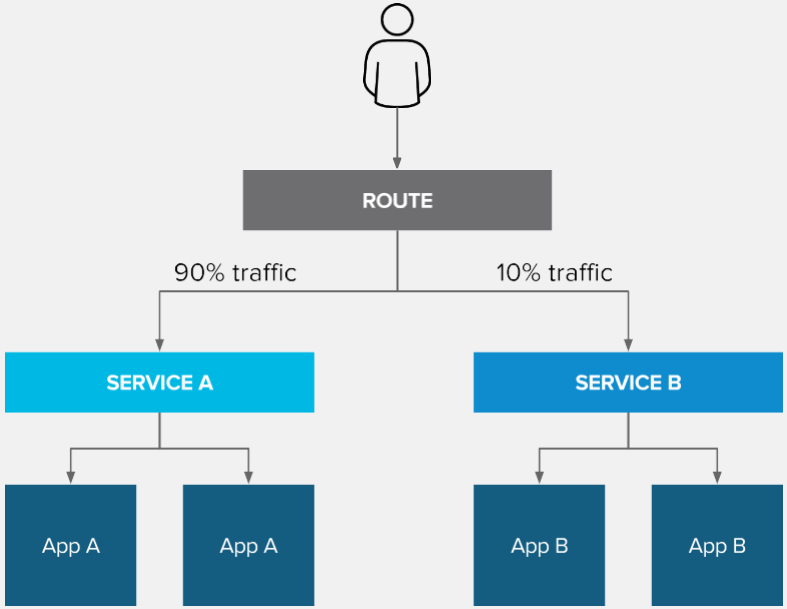
ROUTING AND EXTERNAL LOAD-BALANCING

- 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



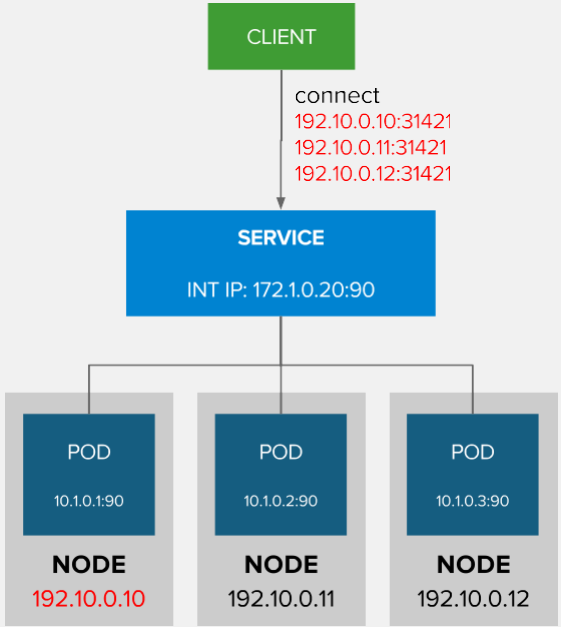
ROUTE SPLIT TRAFFIC

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



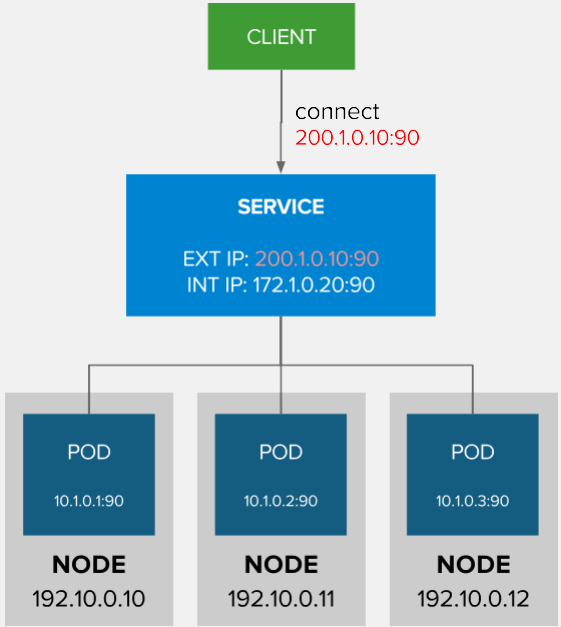
EXTERNAL TRAFFIC TO A SERVICE ON A RANDOM PORT WITH NODEPORT

- 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

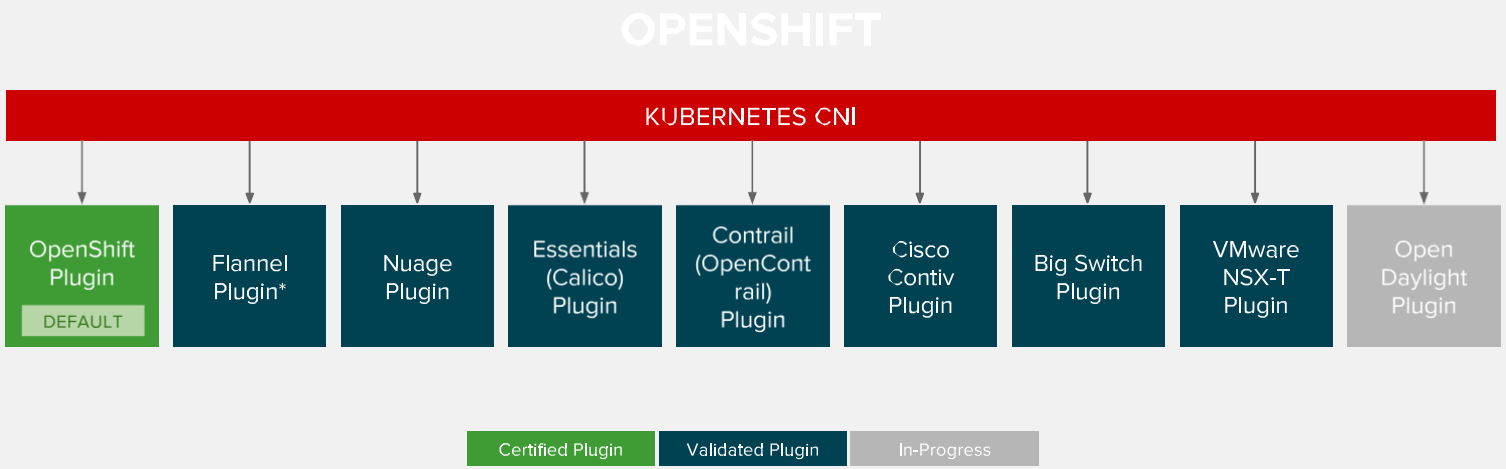


EXTERNAL TRAFFIC TO A SERVICE ON ANY PORT WITH INGRESS

- 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



K8S NETWORK PLUGINS



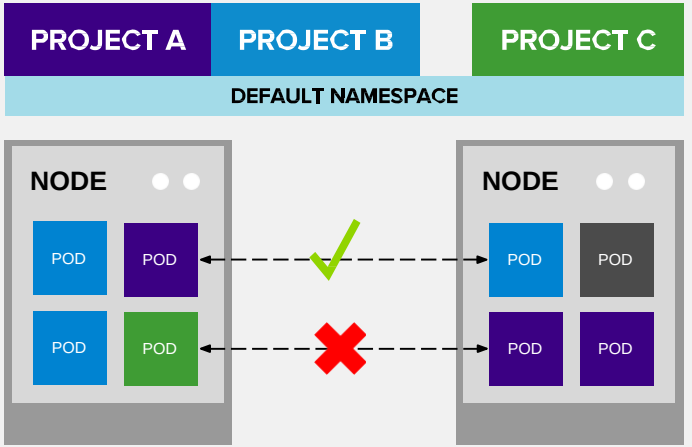
K8S SDN

FLAT NETWORK (Default)

- All pods can communicatewith each other acrossprojects

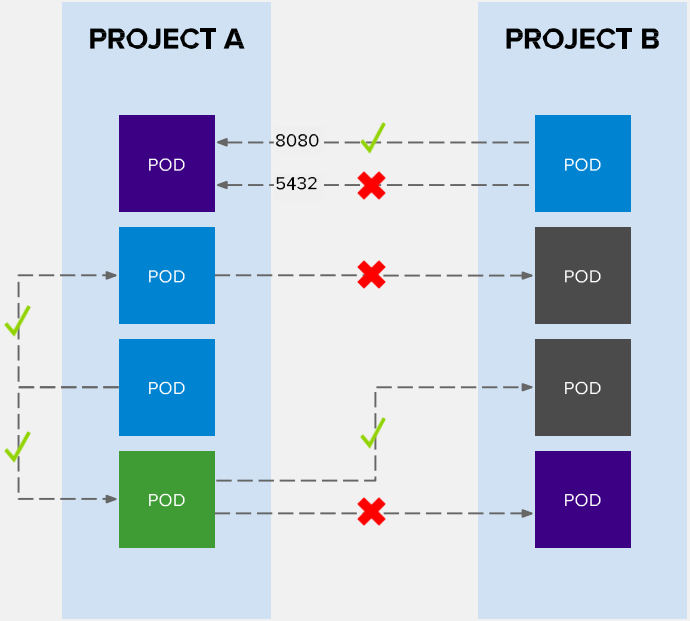
MULTI-TENANT NETWORK

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



Multi-Tenant Network

K8S SDN - NETWORK POLICY



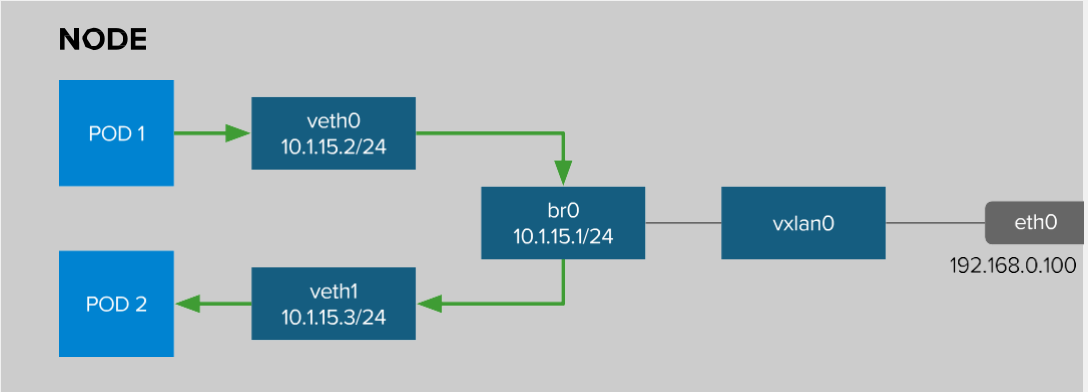
Example Policies

- 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
```

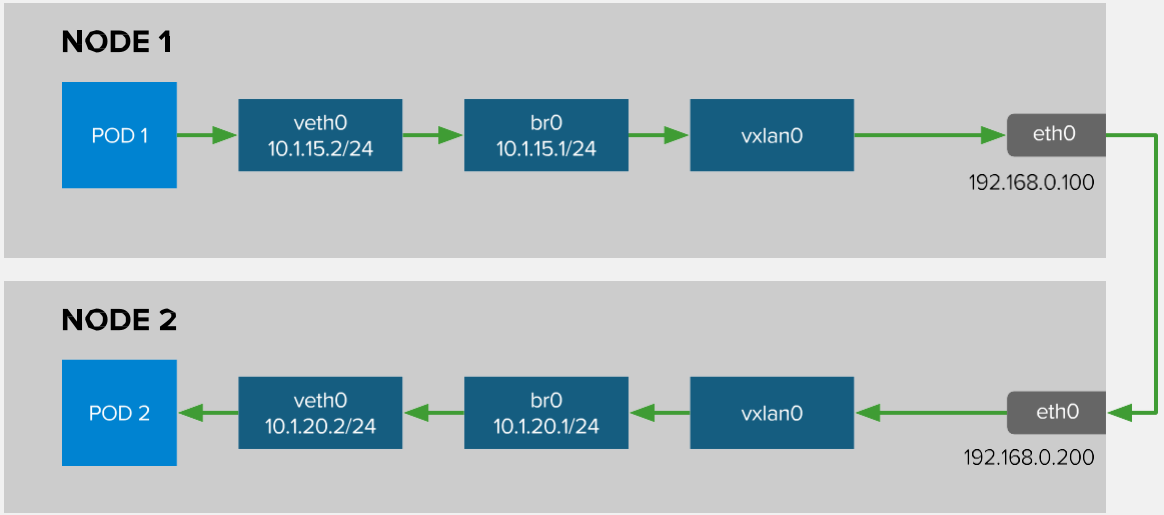
K8S SDN - OVS PACKET FLOW

Container to Container on the Same Host

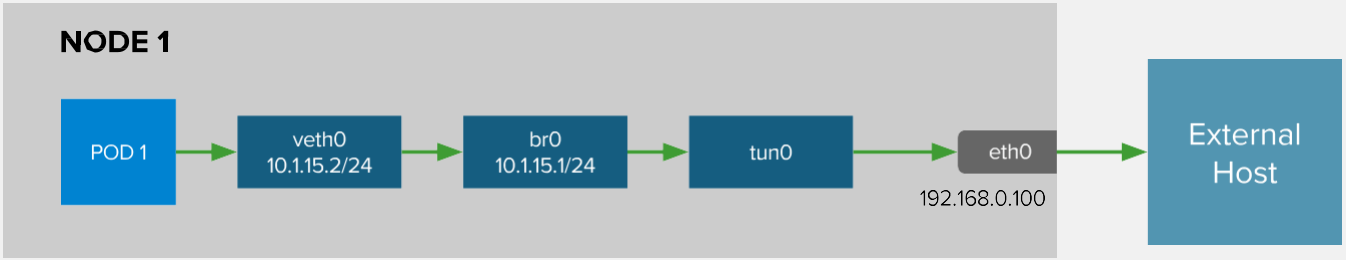


K8S SDN - PACKET FLOW

Container to Container on the Different Hosts



Container Connects to External Host



Labels

Labels

- Labels are key/values pairs that can be attached to objects
  - Labels are like **tags** in AWS or other cloud providers, used to tag resources
- You can **label** your **objects**, for instance your pod, following an org. structure
  - **Key**: environments - **Value**: Dev\ UAT\ QA\ PROD
  - **Key**: department - **Value**: R&D\ finance \ marketing
- In our previous examples, I already have been using labels to tag pods

```
---------------------------
metadata:
  name: nodehelloworld.example.com
  labels:
    app: helloworld
---------------------------
```

Labels

- Labels are **not unique** & **multiple labels** can be added to one object
- Once labels are attached to an object, you can use filters to narrow down results
  - This is called **Label Selector**
- Using Label Selector, you can use **matching expressions** to match labels
  - For instance, a particular pod can only run on a node labeled with "environment" equal "Dev".
  - More complex matching: "environment" in "Dev" or "QA".

Node Labels

- You can also use labels to tag **nodes**
- Once nodes are tagged, you can use **label selectors** to let pods only run on **specified nodes**

- There are **2 steps** required to run a pod on specific set of nodes:
  - First you **tag** the node
  - Then you add a **nodeSelector** to your pod configuration

Labels

User-provided key-value attributes

Attached to any API object

Generally represent **identity**

Queryable by **selectors**

- think SQL '*select … where …*'

The **only** grouping mechanism

Selectors

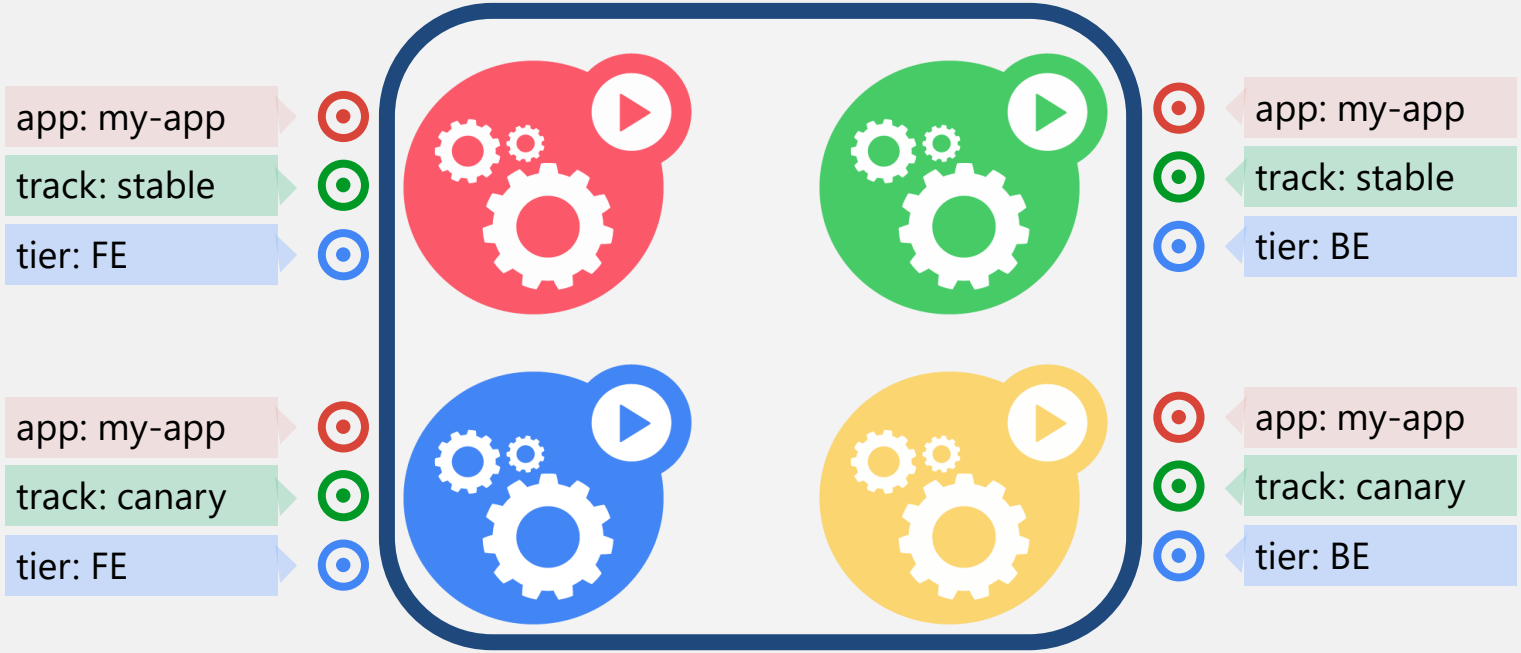
app: my-app
track: stable
tier: FE

app: my-app
track: stable
tier: BE

app: my-app
track: canary
tier: FE

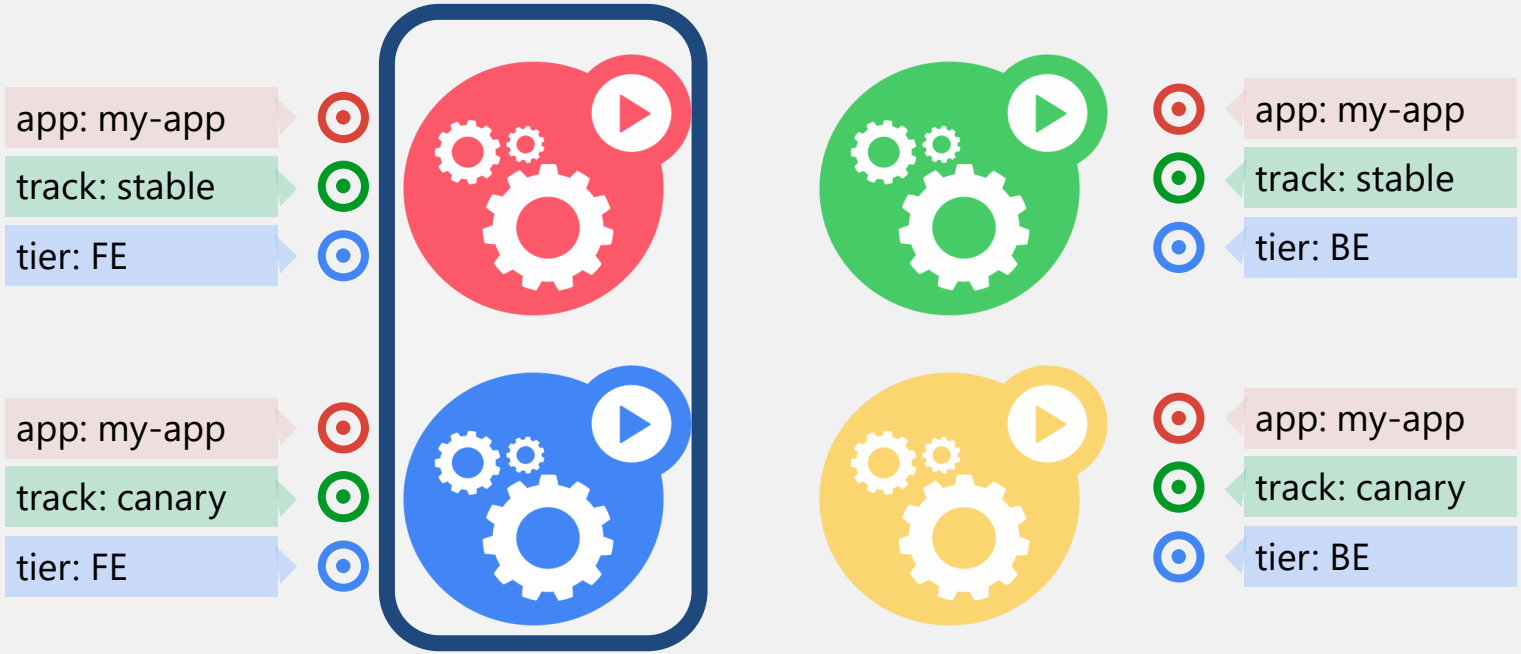
app: my-app
track: canary
tier: BE

Selectors



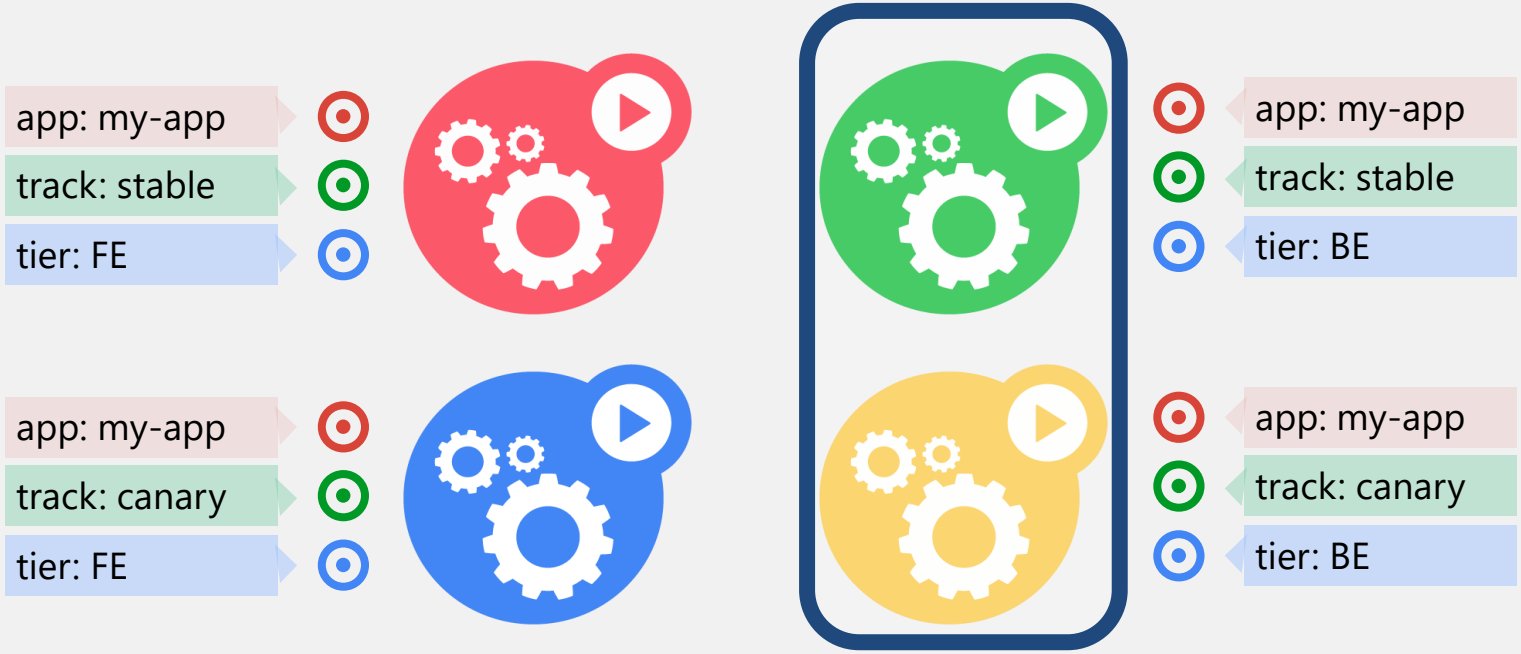
app = my-app

Selectors



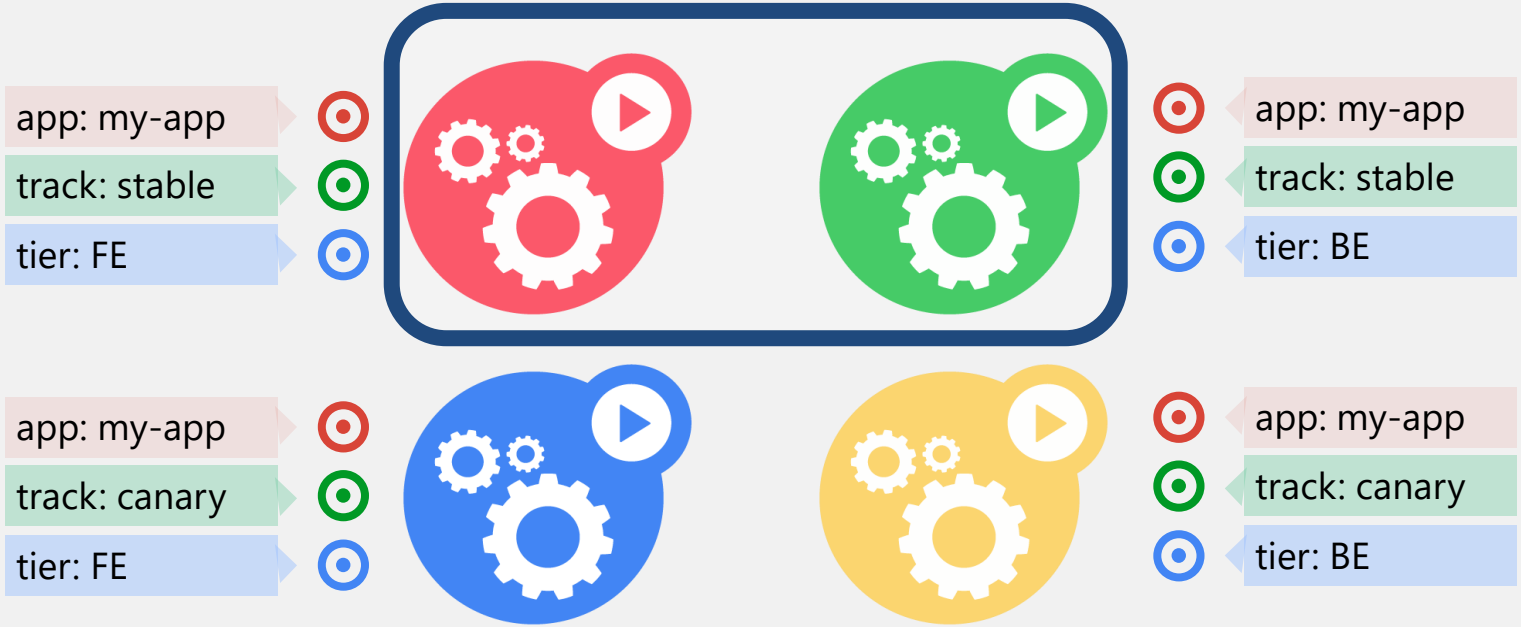
app = my-app, tier = FE

Selectors



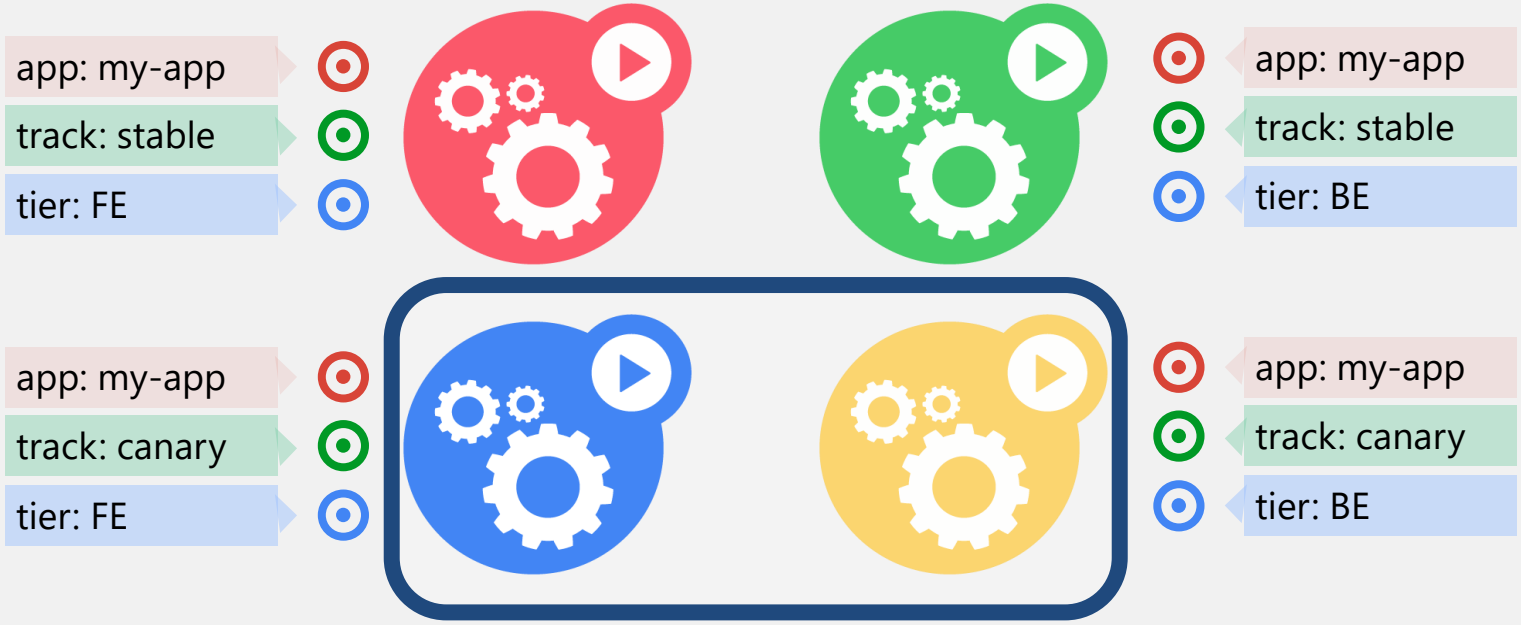
app = my-app, tier = BE

Selectors



app = my-app, track = stable

Selectors



app = my-app, track = canary

Node Labels

- First Step, add a label or multiple labels to your nodes:

```
# kubectl label nodes worker01 hardware=high
# kubectl label nodes worker02 hardware=low
```

- Secondly, add a pod that uses those labels:

```
----------------------------------------
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: helloworld-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: helloworld
    spec:
      containers:
      - name: k8s-demo
        image: amitvashist7/k8s-tiny-web
        ports:
        - name: nodejs-port
          containerPort: 80
```

Demo Placeholder

Node Selector using labels

Health Checks

Health Checks

- If your application **malfunctions**, the pod & container can still be running, but the application might not work anymore.
- To **detect** & **resolve** problems with your application, you can run **health checks**
- You can run 2 different type of health
  - Running a **command** in the container **periodically**
  - Periodic checks on the a **URL** (HTTP)
- The Typical production application behind a load balancer should always have **health checks** implemented in some way to ensure **availability** & **resiliency** of the app.

Health Checks

- This is how a health check looks like on our example container:

```
------------------------------
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: helloworld-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: helloworld
    spec:
      containers:
      - name: k8s-demo
        image: amitvashist7/k8s-tiny-web
        ports:
        - name: nodejs-port
          containerPort: 80
        livenessProbe:
          httpGet:
            path: /
            port: nodejs-port
          initialDelaySeconds: 15
          timeoutSeconds: 30
------------------------------
```

Demo Placeholder

Performing health checks


Secrets
redhat.

Secrets

- Secrets provides a way in kubernetes to distribute **credentials**, **Keys**, **passwords** or "**secret**" **data** to the pods
- Kubernetes itself uses this Secrets mechanism to provide the credentials to access the internal API
- You can also use the **same mechanism** to provide secret to your application
- Secrets is one way to provide secret, native to Kubernetes
  - There are still **other ways** your container can get the its secrets if you don't want to use Secrets (e.g using an **external vault services** in your app)

Secrets

- Secrets can be used in the following ways:
  - Use Secrets as **environment varia**bles
  - Use secrets **as a file** in a pod
    - This setup uses **volumes** to be mounted in a container
    - In this volume you have **files**
    - Can be used for instance for **dotenv** files or you app can just read this file

Secrets

- To generate secrets using files:

```
# echo -n "root" > ./username.txt
# echo -n "password" > ./password.txt

# kubctl create secret generic db-user-pass --from-file=./username.txt --from-file=./password.t
```

- A Secret can also be an SSH Key or an SSL certificate

```
# kubctl create secret generic ssl-certificate --from-file=ssh-privatekey=~/.ssh/id_rsa  --ssl-cert=ssl-cert=mysslcert.
```

Secrets

- To generate secret using yaml definitions:

secrets-db-secret.yml

```
---------------------------------
apiVersion: v1
kind: Secret
metadata:
  name: db-secrets
type: Opaque
data:
  username: cm9vdA==
  password: cGFzc3dvcmQ=
---------------------------------
```

```
# echo -n 'root' | base64
cm9vdA==
# echo -n 'password' | base64
cGFzc3dvcmQ=
```

- After creating the yml file, you can use kubectl create:

```
# kubectl create -f secrets-db-secret.yml
```

Using Secrets

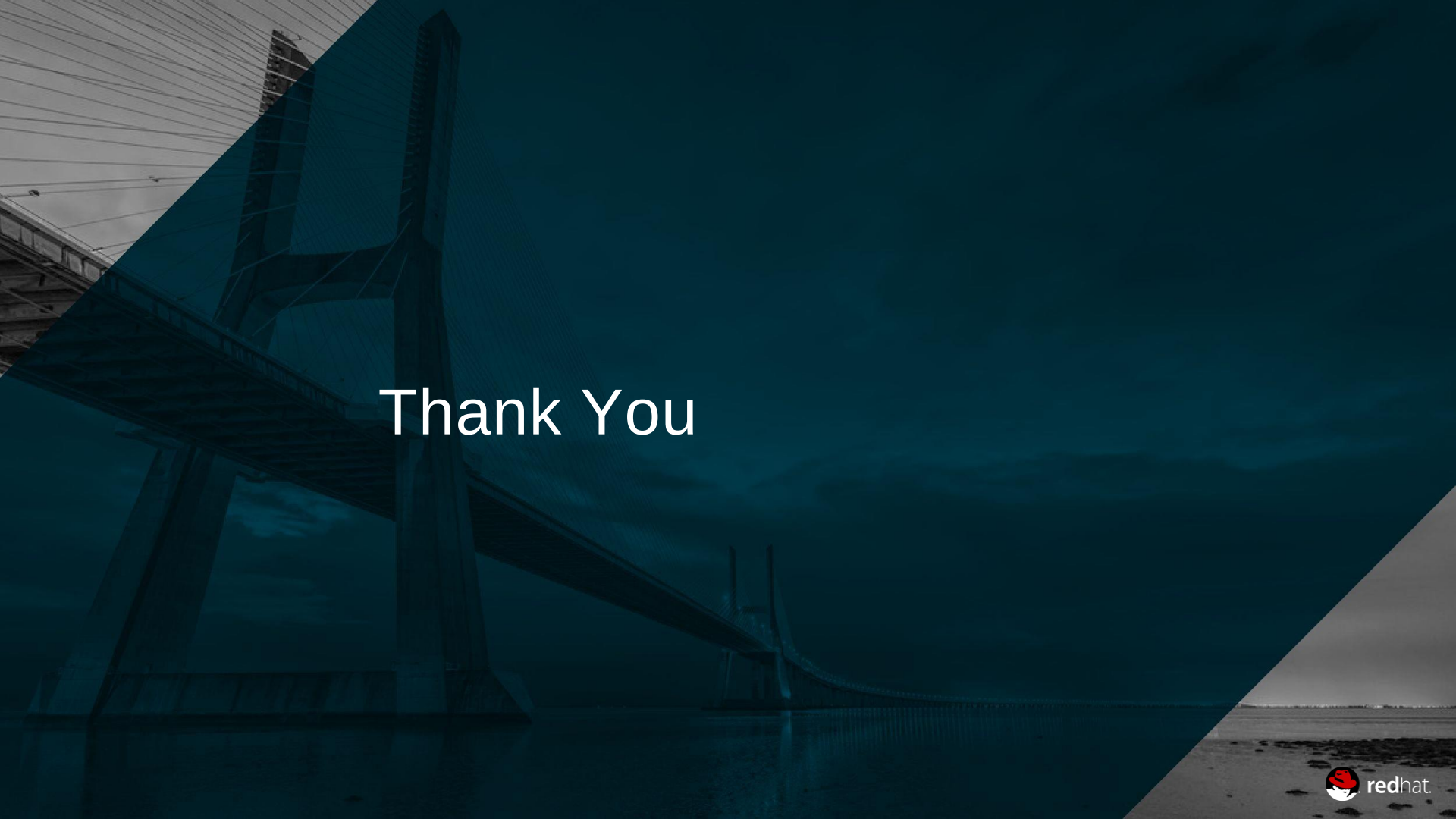
- You can create a pod that expose the secret as environment variables:

```
env:
        - name: MYSQL_HOST
          value: database-service
        - name: MYSQL_USER
          value: root
        - name: MYSQL_PASSWORD
          valueFrom:
            secretKeyRef:
              name: helloworld-secrets
              key: rootPassword
        - name: MYSQL_DATABASE
          valueFrom:
            secretKeyRef:
              name: helloworld-secrets
              key: database
```

Demo Placeholder

Secrets

Setting up Wordpress


Thank You
redhat.