Máster
Architecture & Engineering

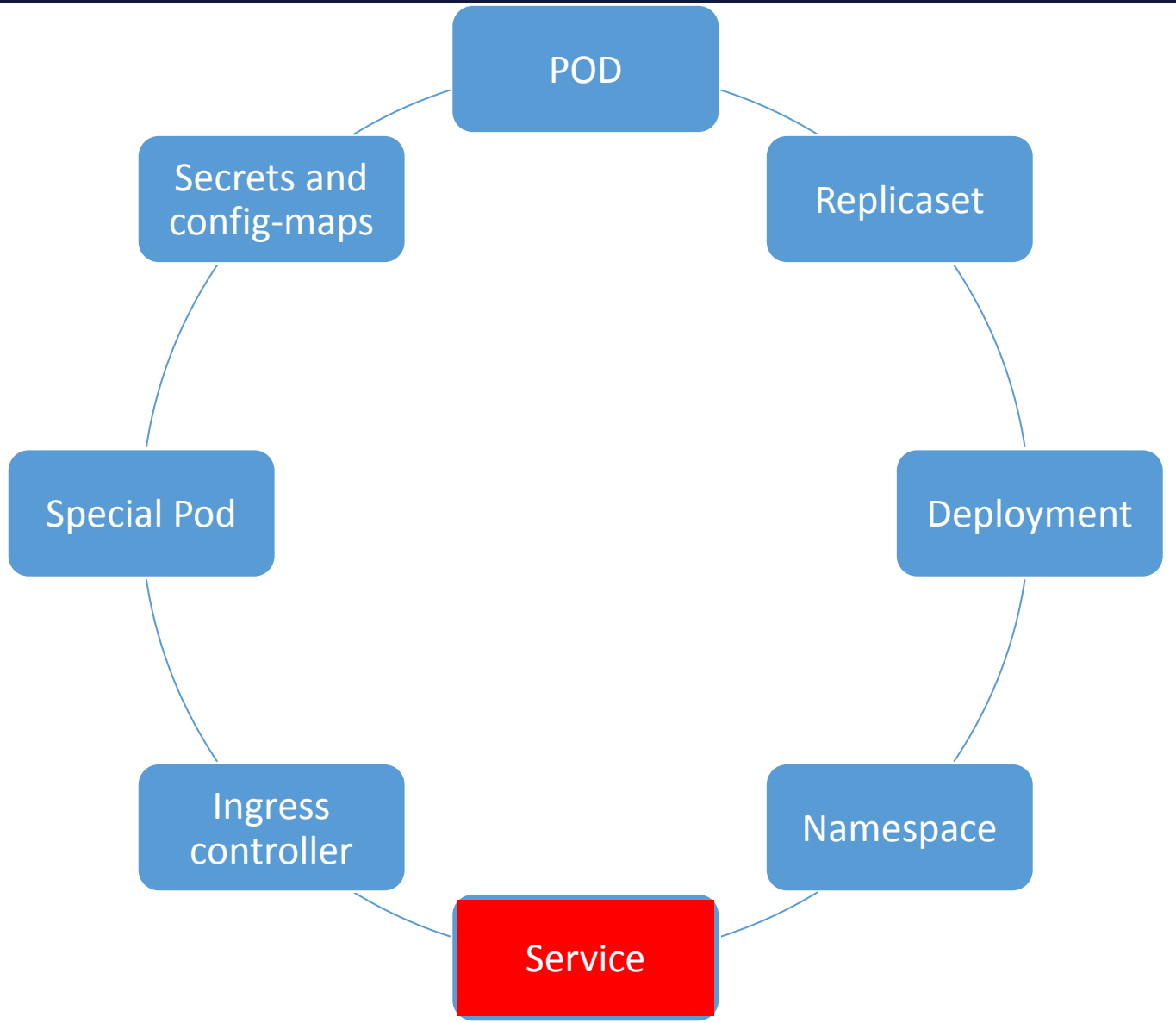
Cristabel Talavera Arriola

Kubernetes

Indice

- Componentes in K8S
  - Servicios
- Networking
  - Ingress Controller
  - Special Pods
- Scaling
- Storage
- Monitorización y Logging
- Tolerancias y Afinidades en K8S

Componentes en K8S

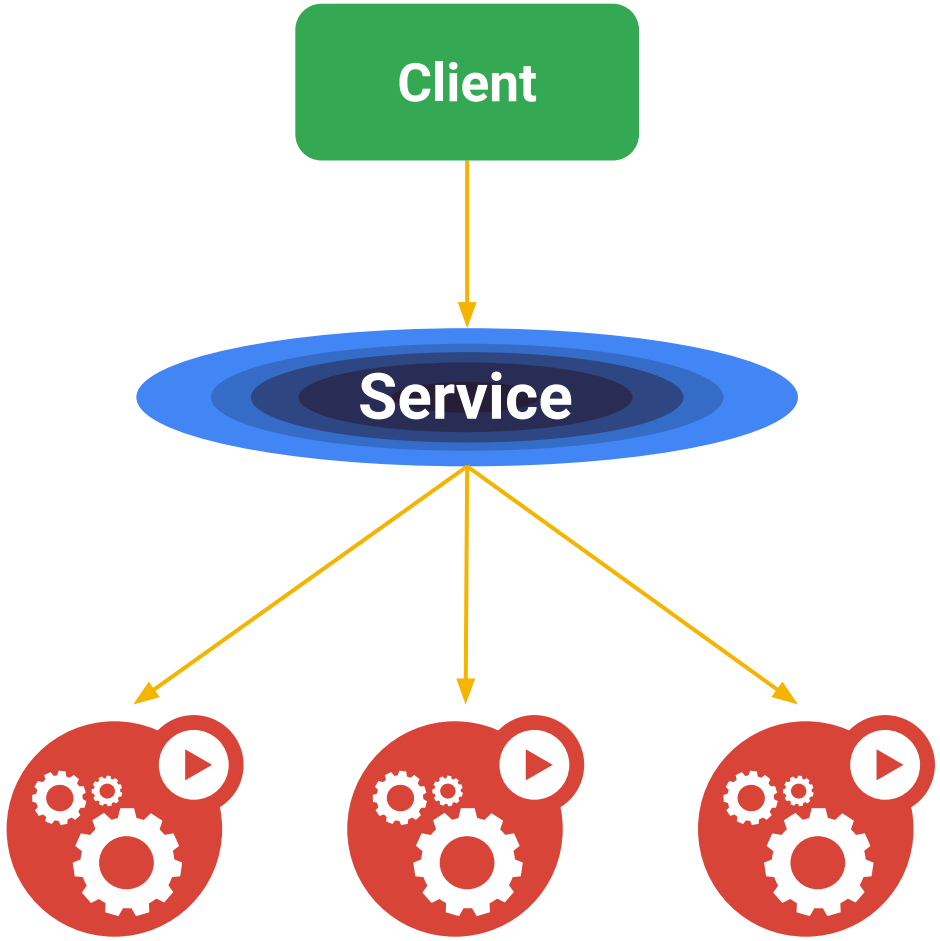


Services

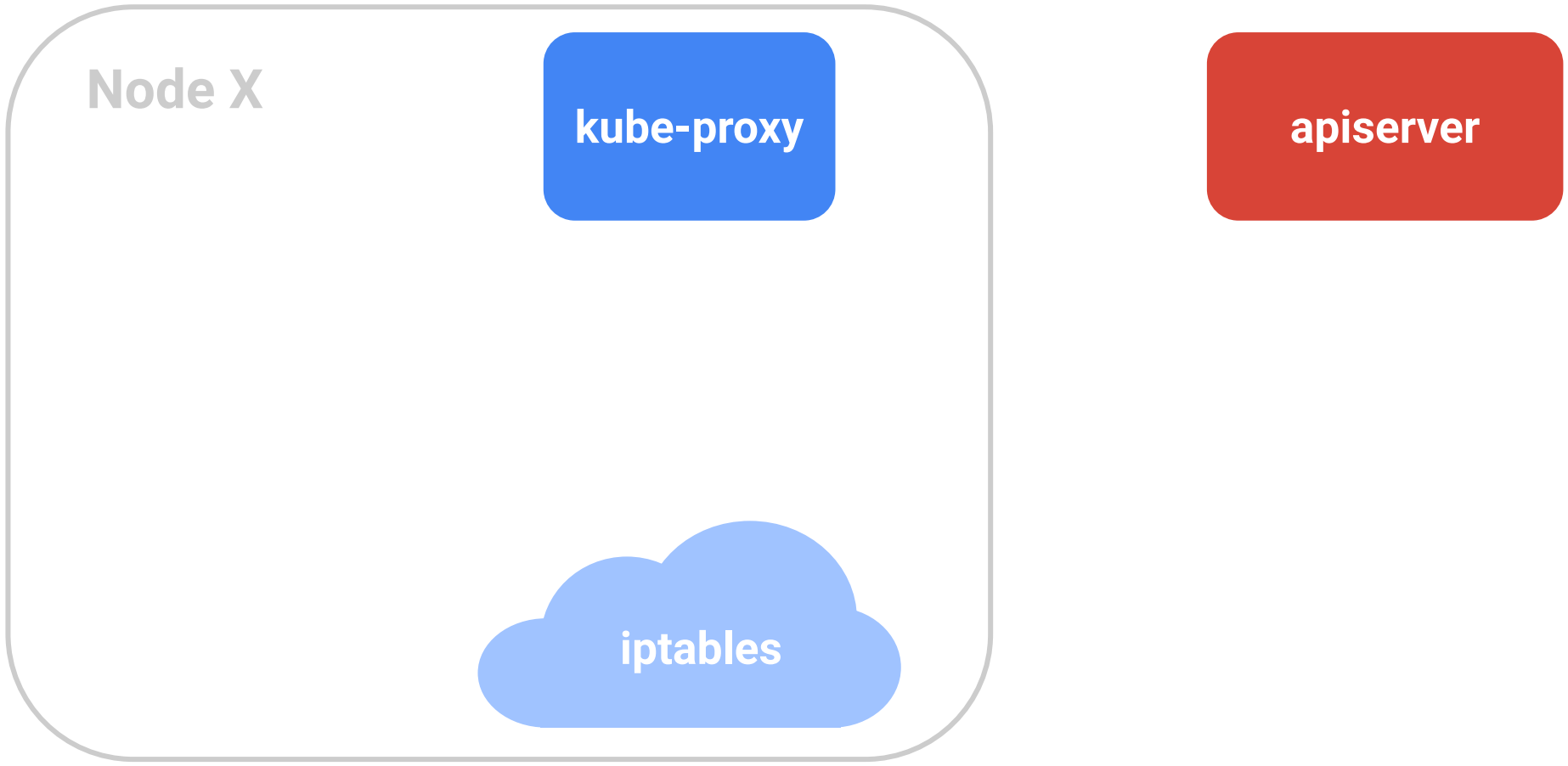
- A service defines a logical set of pods (your *microservice)*
- They are essentially a virtual **load balancer** in front of pods
- *Load balancing is the process of distributing workloads across multiple servers, collectively known as a server cluster. The main purpose of load balancing is to prevent any single server from getting overloaded and possibly breaking down. In other words, load balancing improves service availability and helps prevent downtimes.*

Services

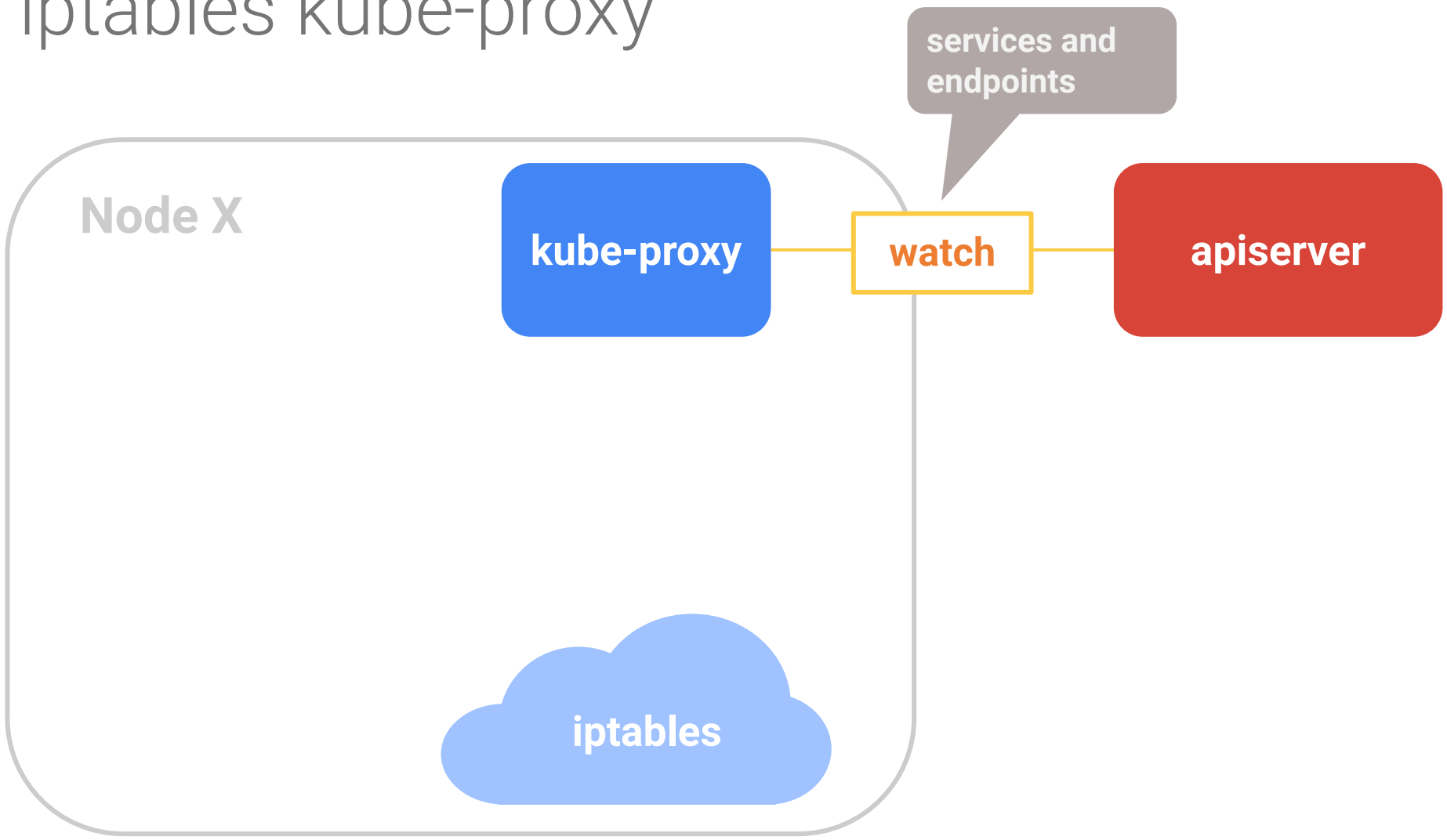
- A group of pods that work together
  - Grouped by a selector
- Defines access policy
  - “Load balanced” or “headless”
- Can have a stable virtual IP and port
  - Also a DNS name
- VIP is managed by kube-proxy
  - Watches all services
  - Updates iptables when backends change
  - Default implementation — can be replaced!
- Hides complexity



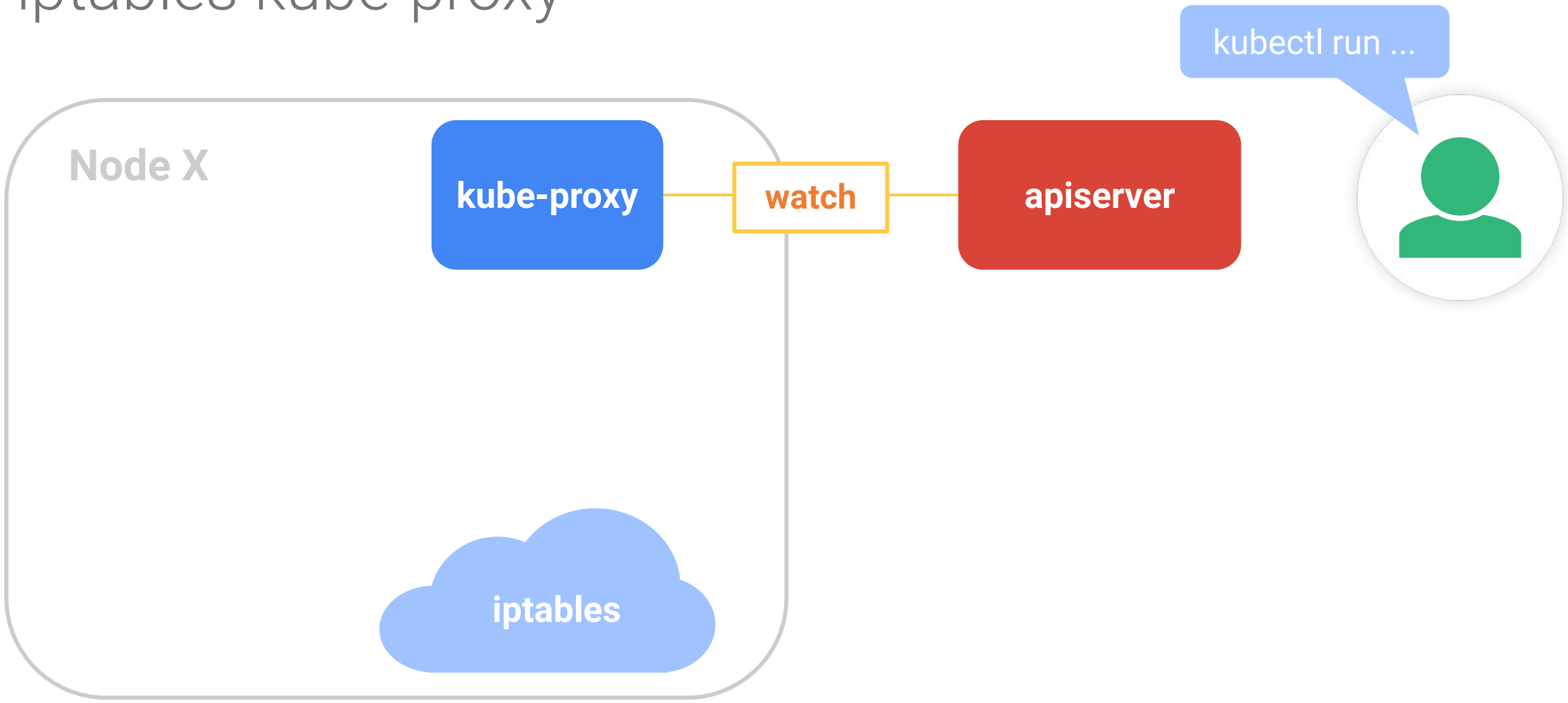
iptables kube-proxy



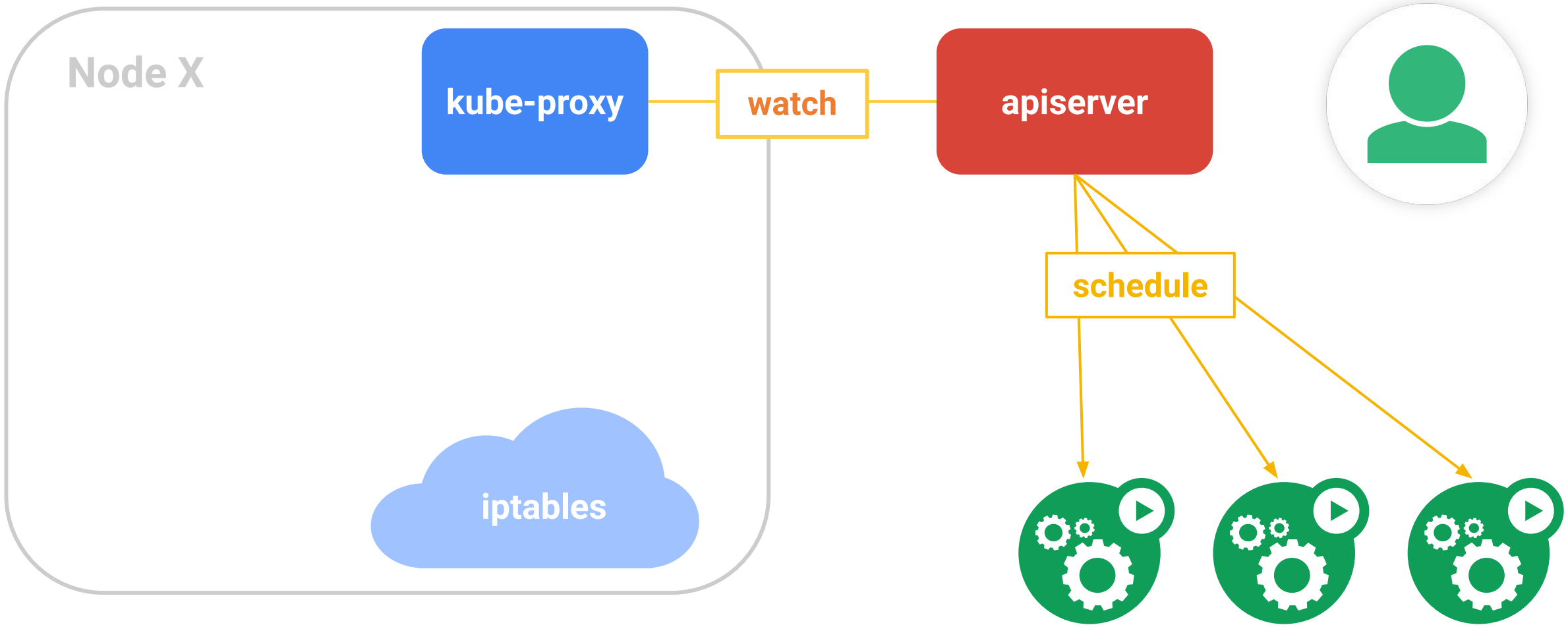
iptables kube-proxy



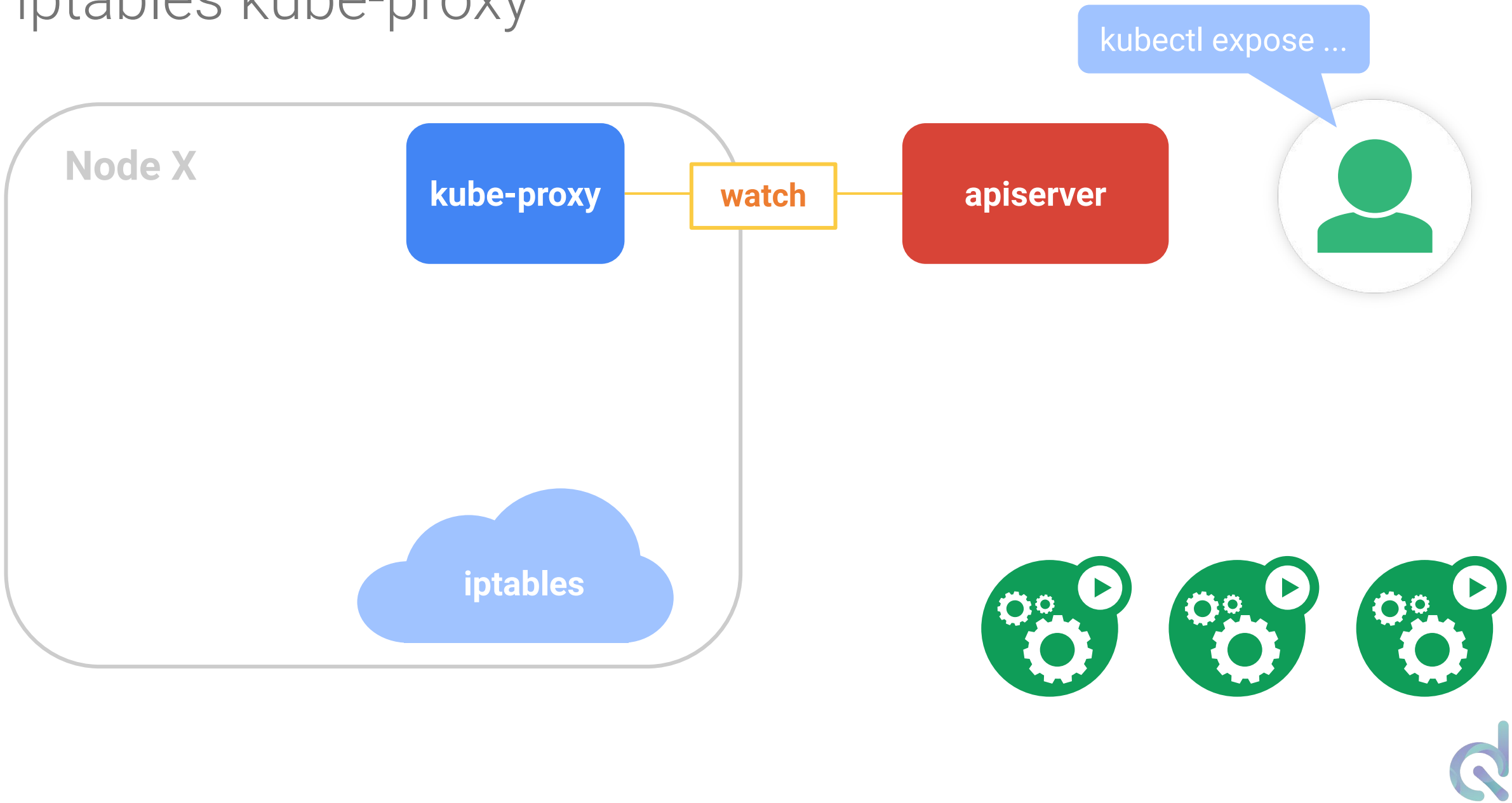
iptables kube-proxy



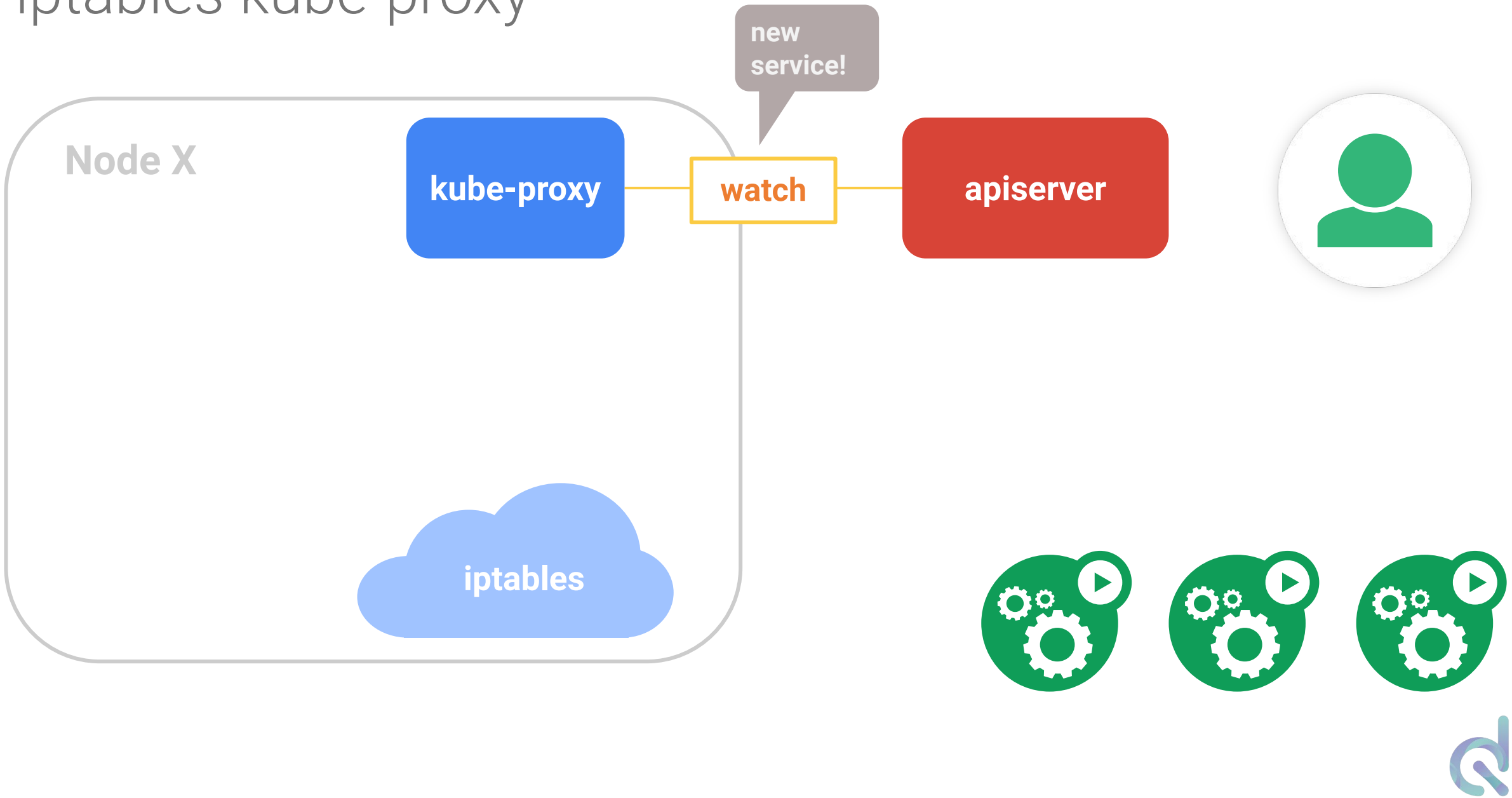
iptables kube-proxy



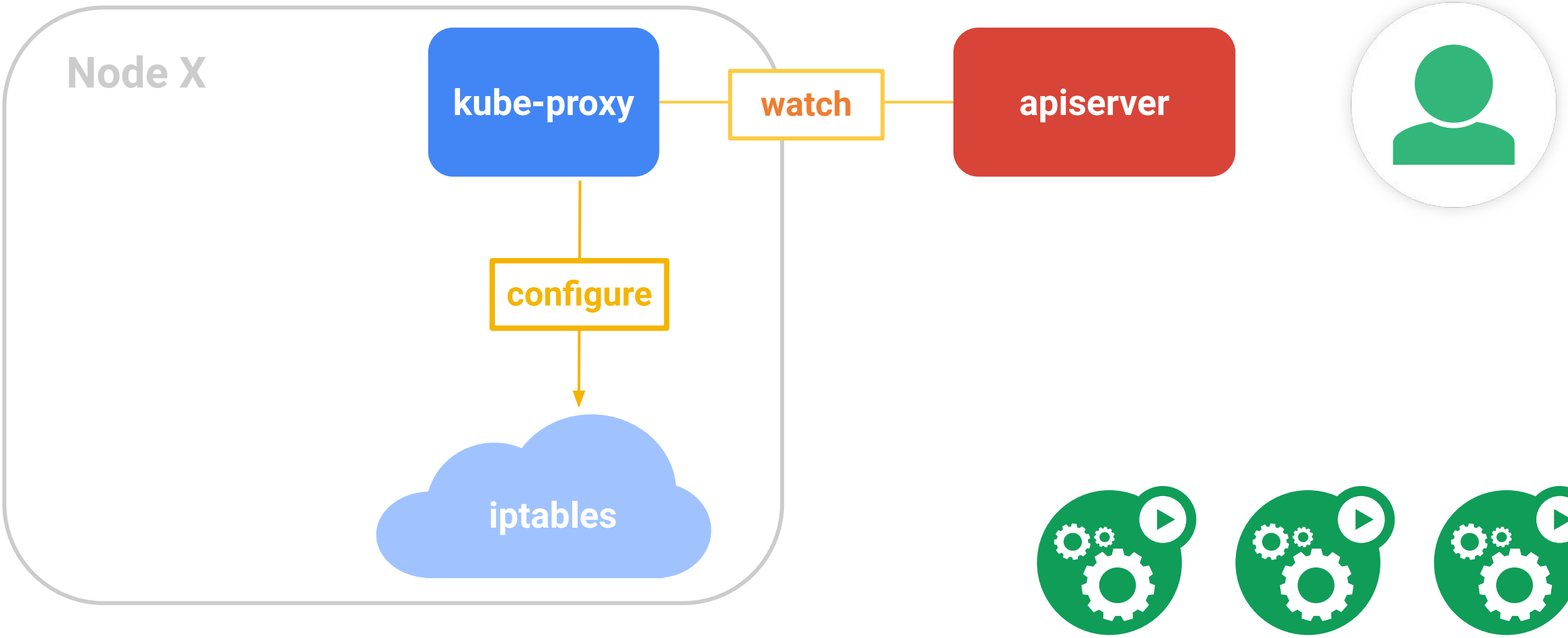
iptables kube-proxy



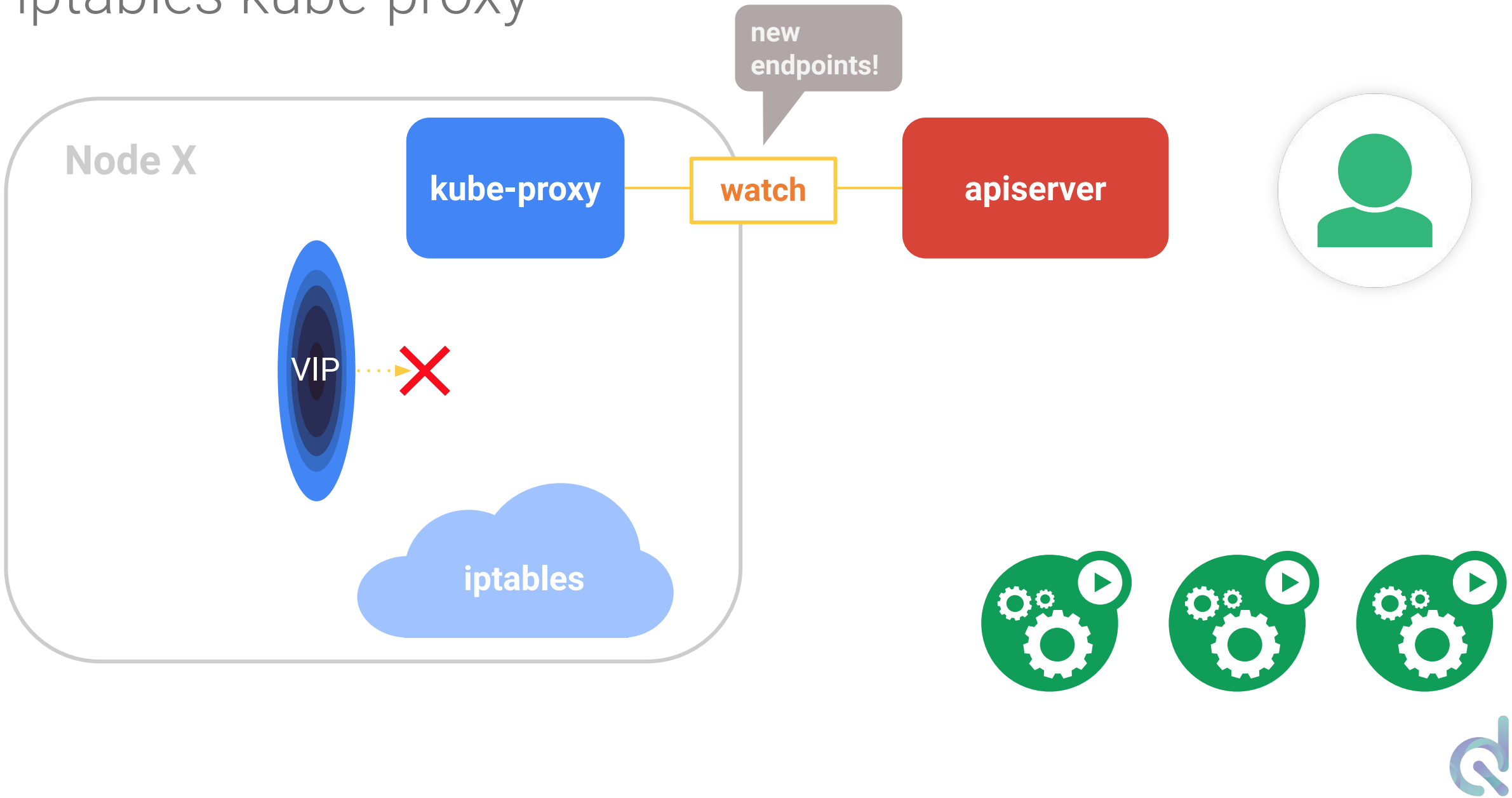
iptables kube-proxy



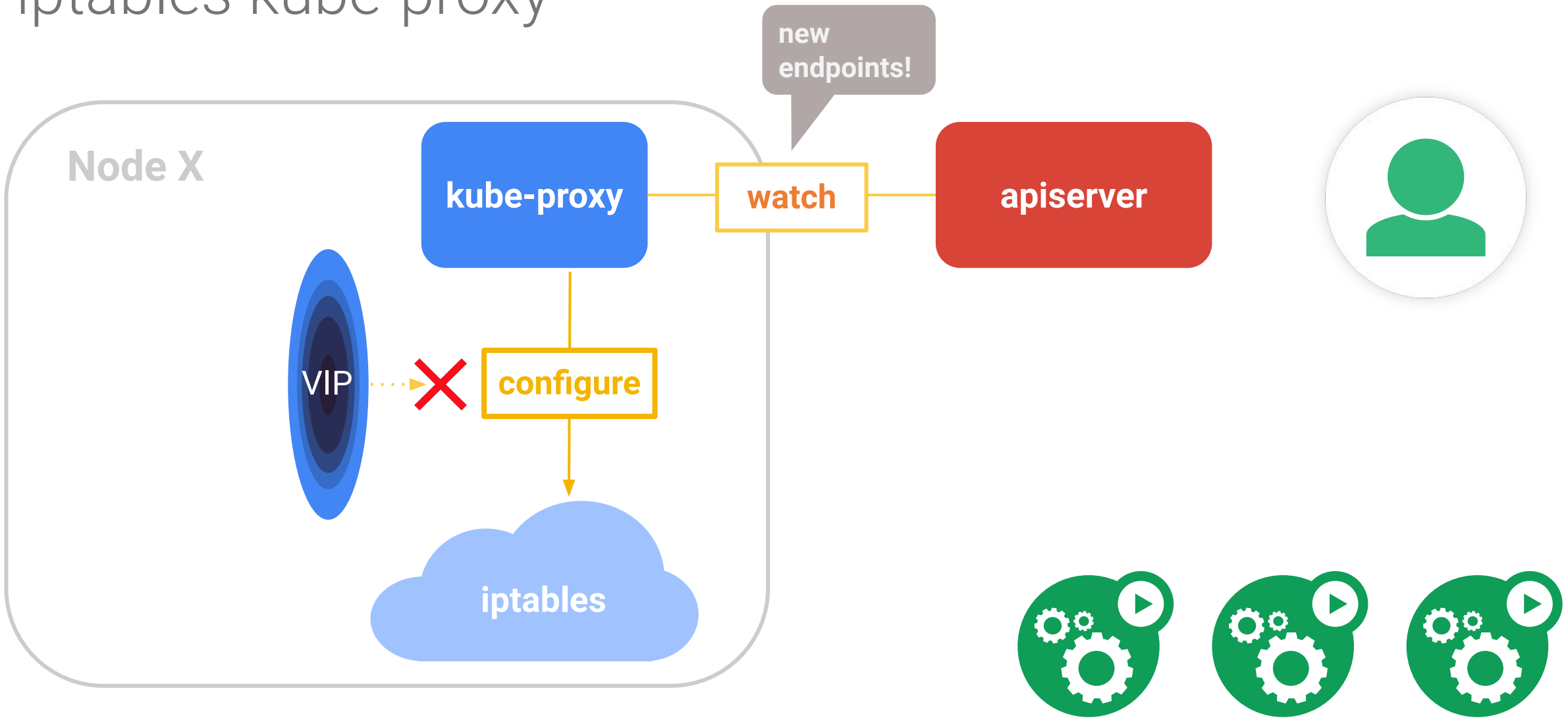
iptables kube-proxy



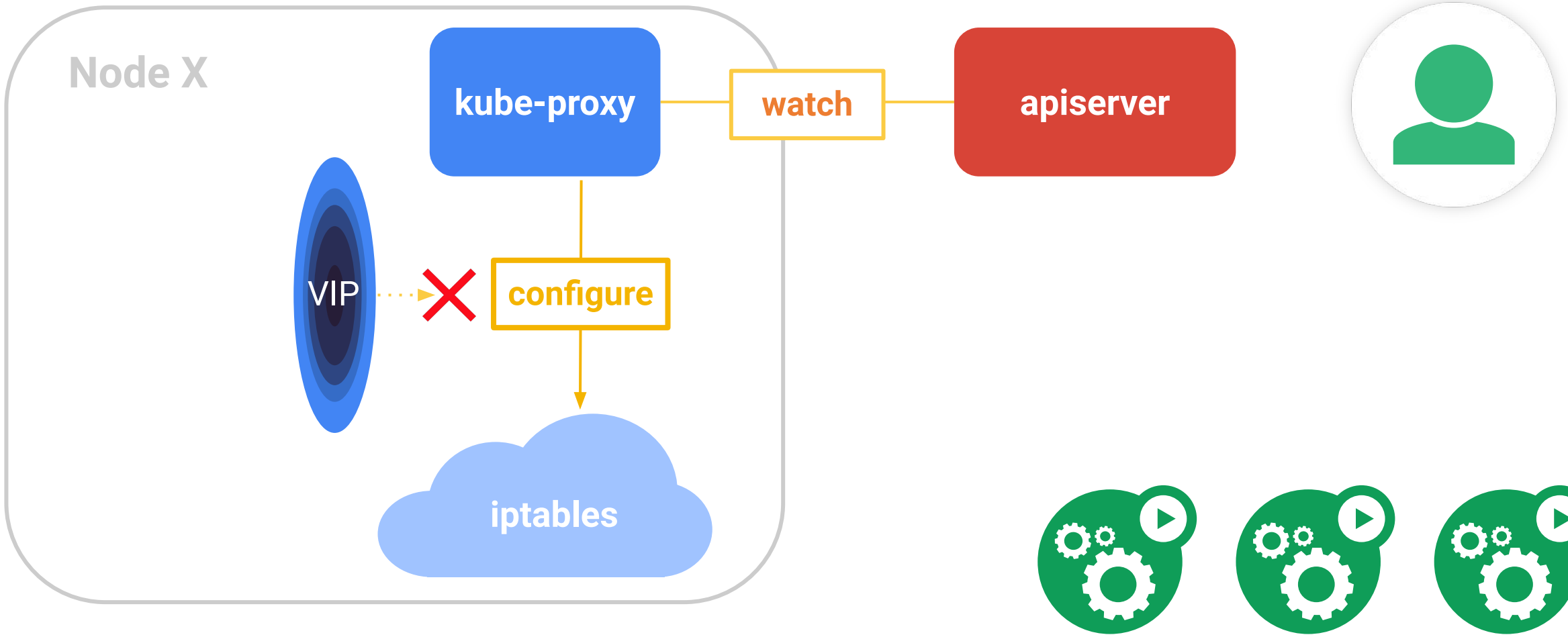
iptables kube-proxy



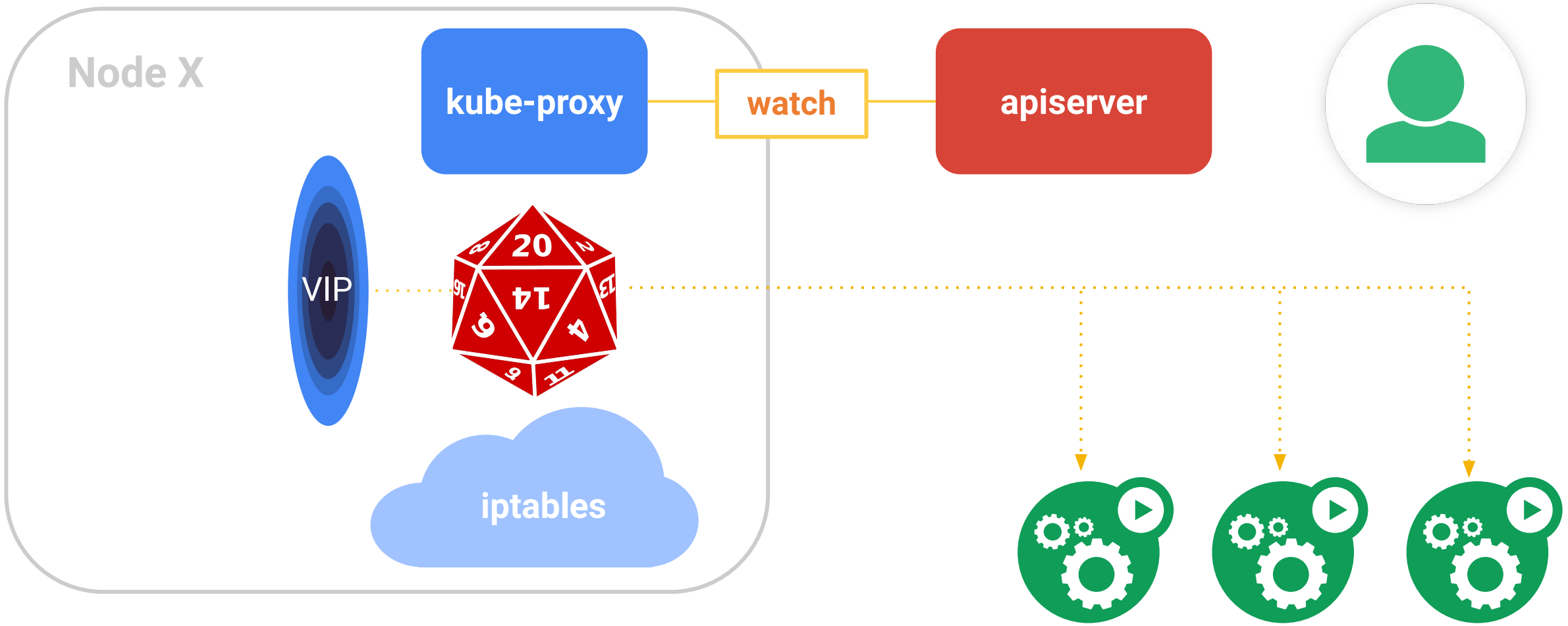
iptables kube-proxy



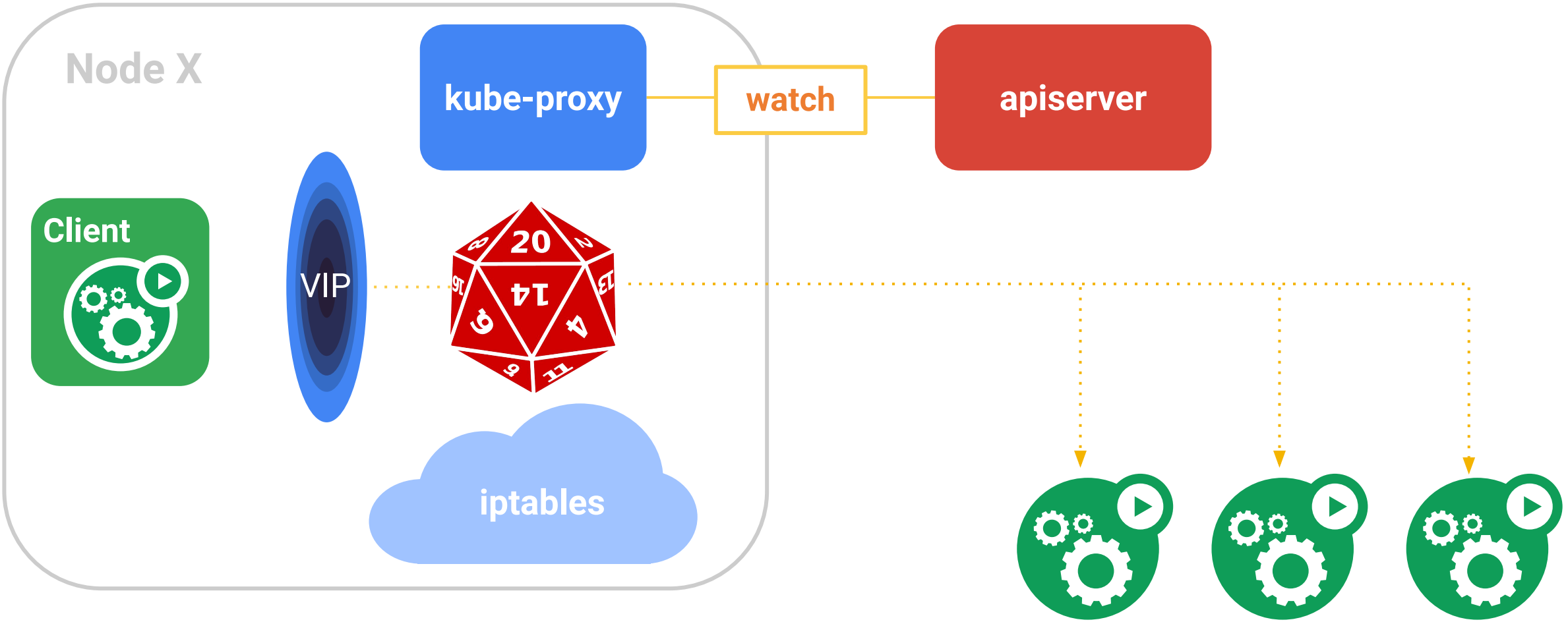
iptables kube-proxy



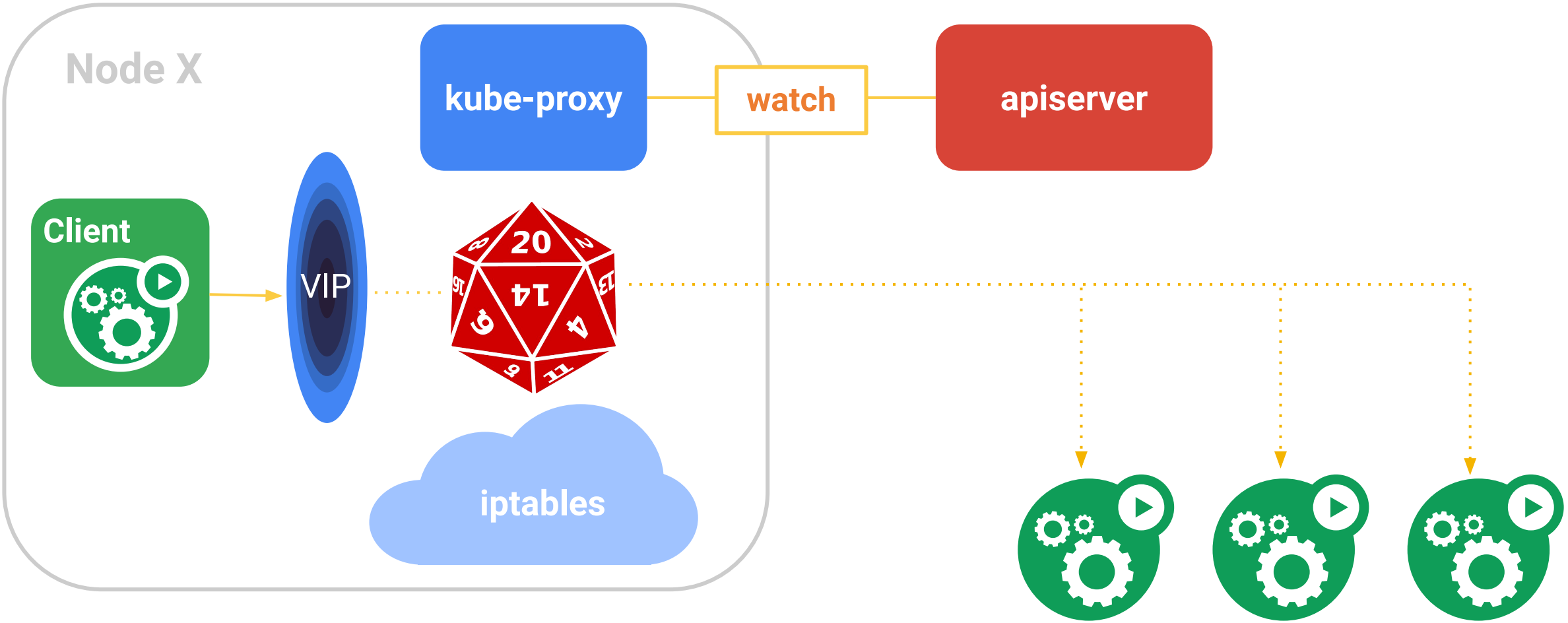
iptables kube-proxy



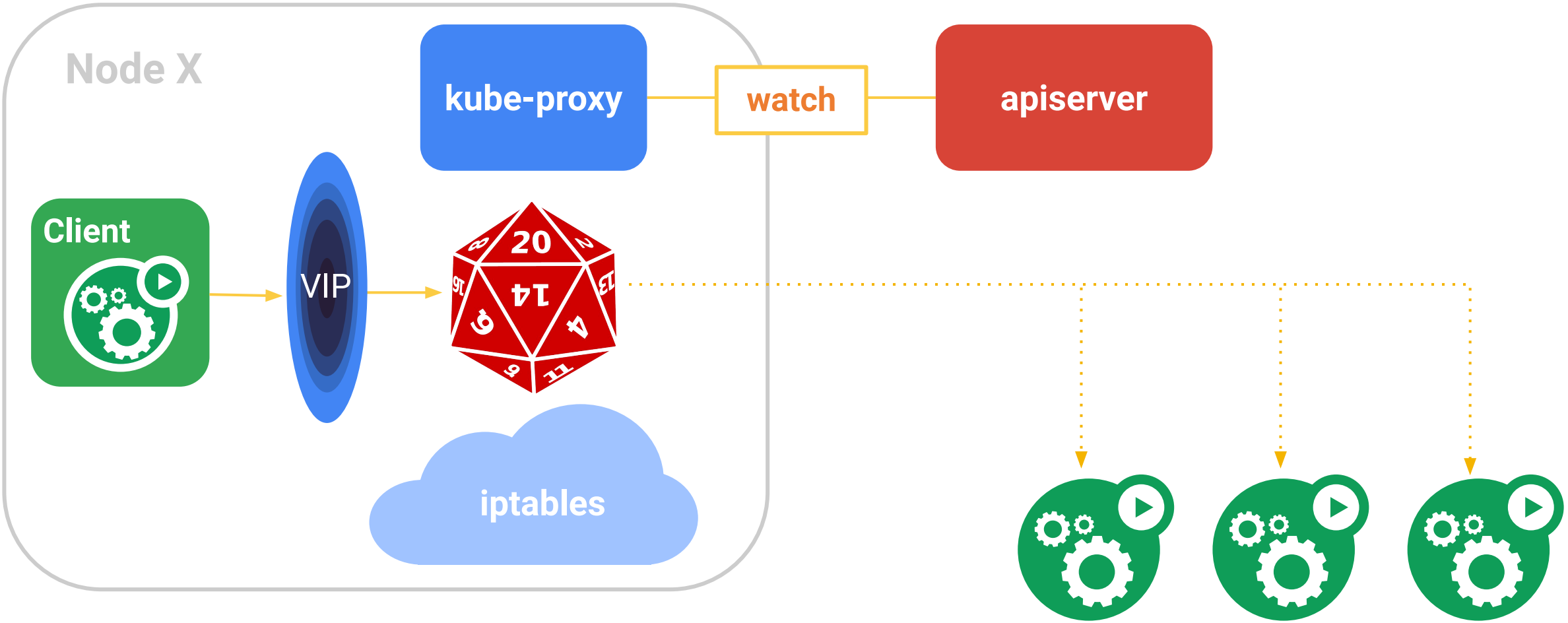
iptables kube-proxy



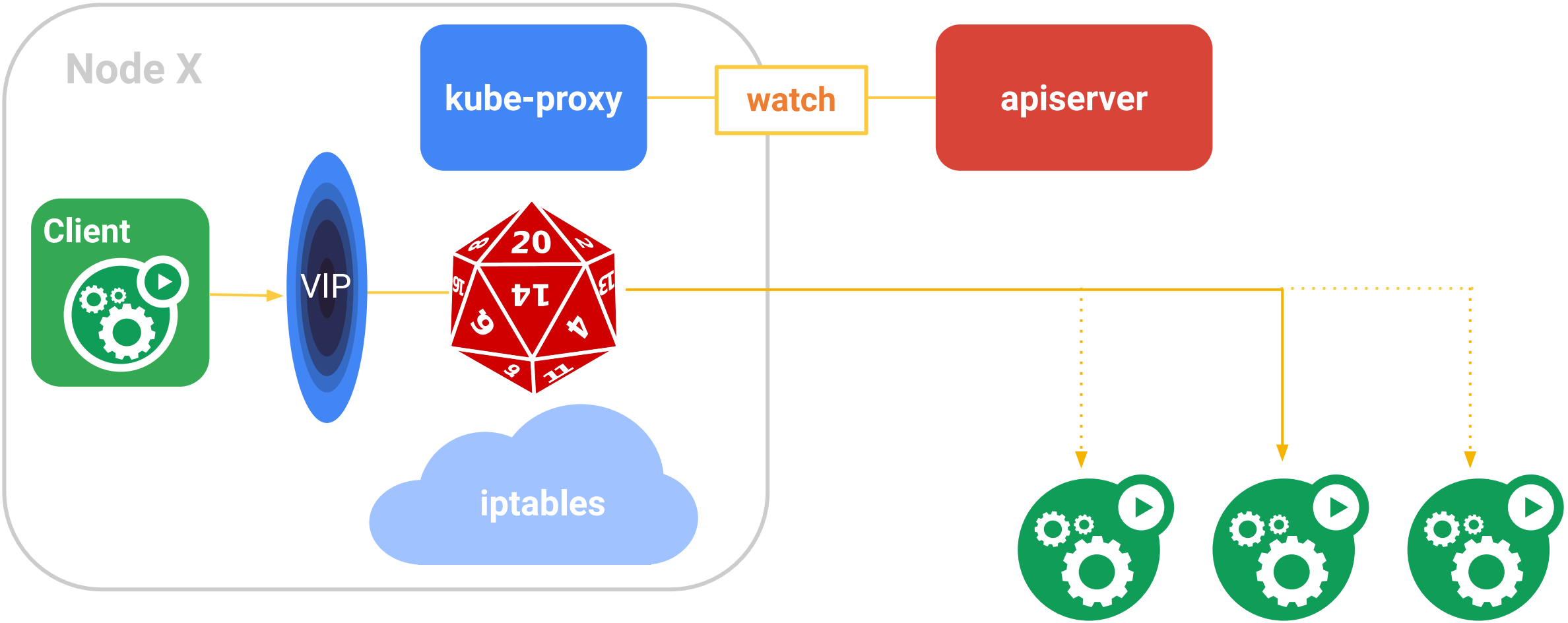
iptables kube-proxy



iptables kube-proxy



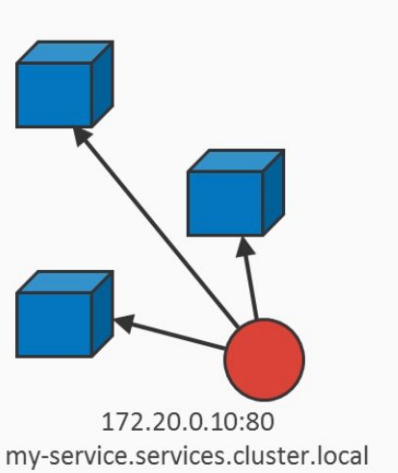
iptables kube-proxy



Service Types

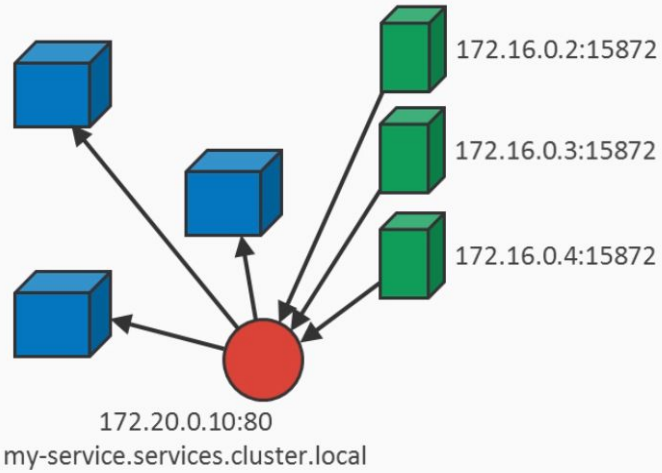
- **ClusterIP**:
  - Exposes the service on a cluster-internal IP. Choosing this value makes the service only reachable from within the cluster
- **NodePort**:
  - Exposes the service on each Node's IP at a static port (the NodePort)
  - Connect from outside the cluster by requesting <NodeIP>:<NodePort>
- **LoadBalancer**:
  - Exposes the service externally using a cloud provider's load balancer

Service Types



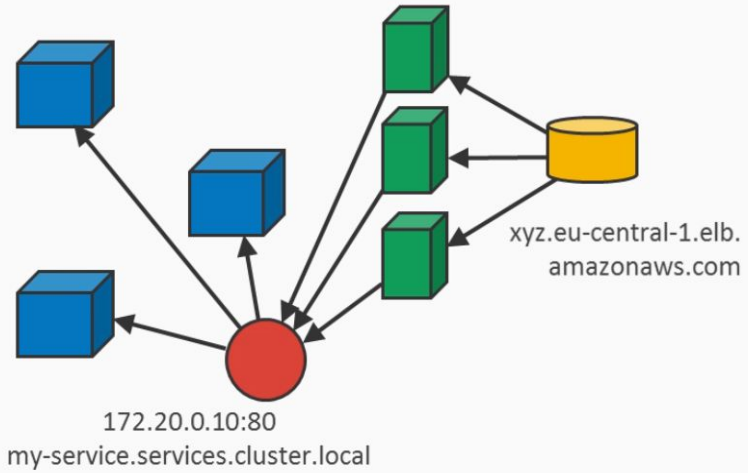
ClusterIP

- Stable internal IP
- Stable internal DNS



NodePort

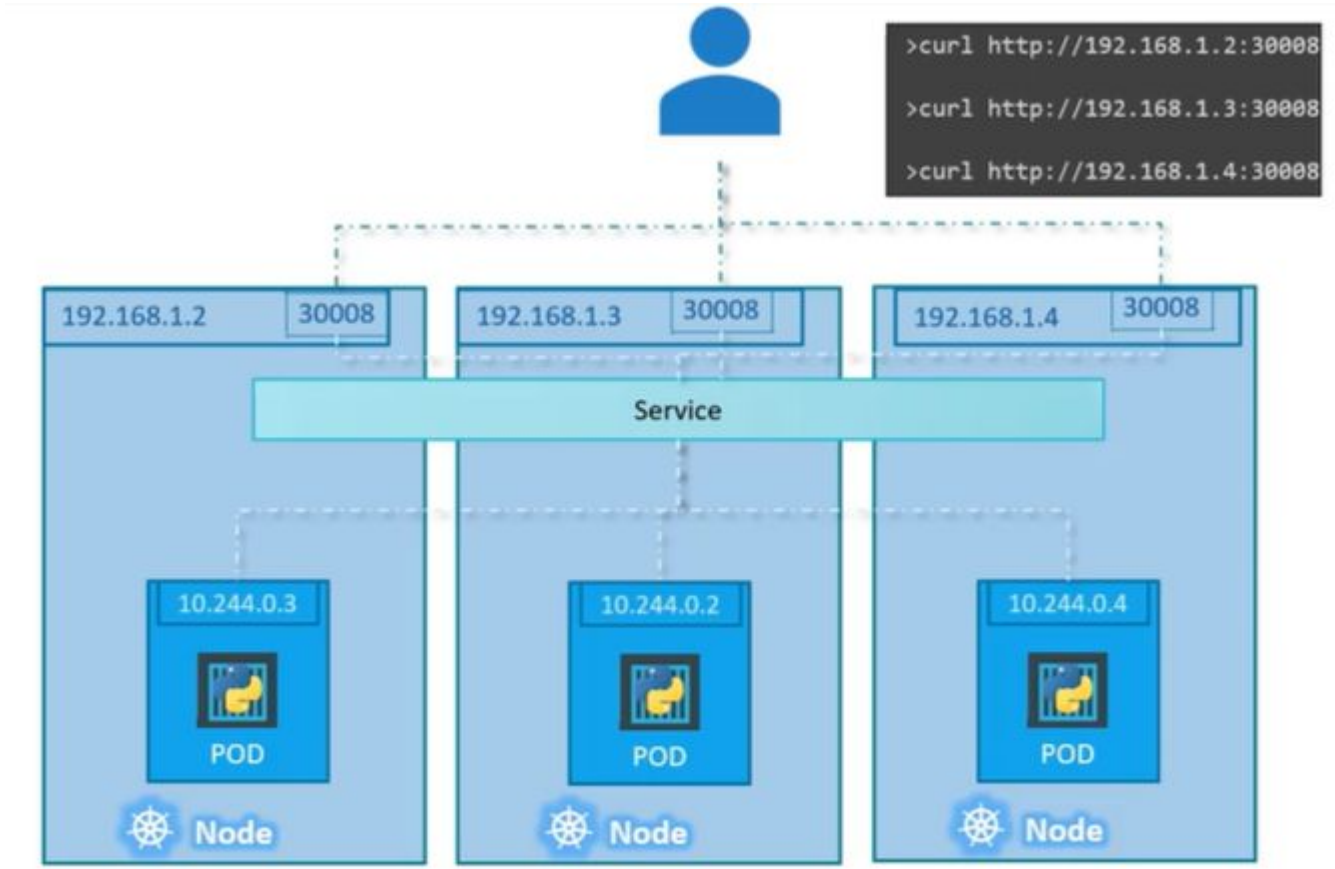
- As ClusterIP, but additionally
- Every Node gets a public TCP port forwarding to that service



LoadBalancer

- As NodePort, but additionally
- A load balancer (must be supported by cloud provider) is created to allow external incoming traffic

Service with NodePort



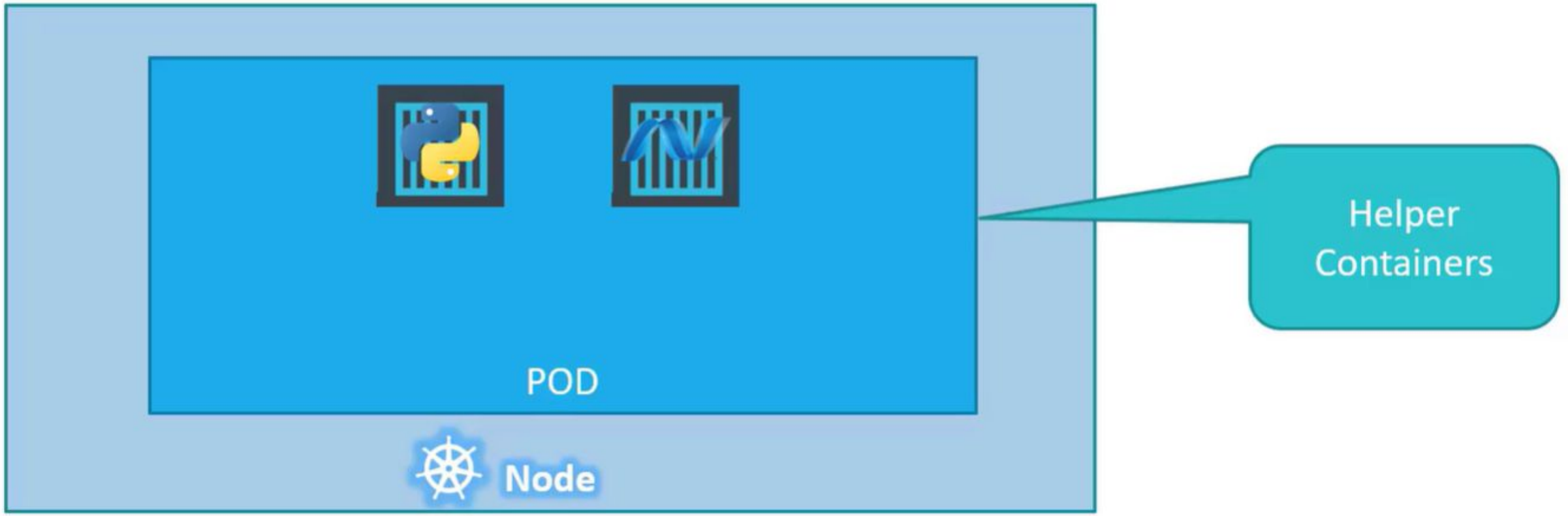
Kubernetes manifest: Service

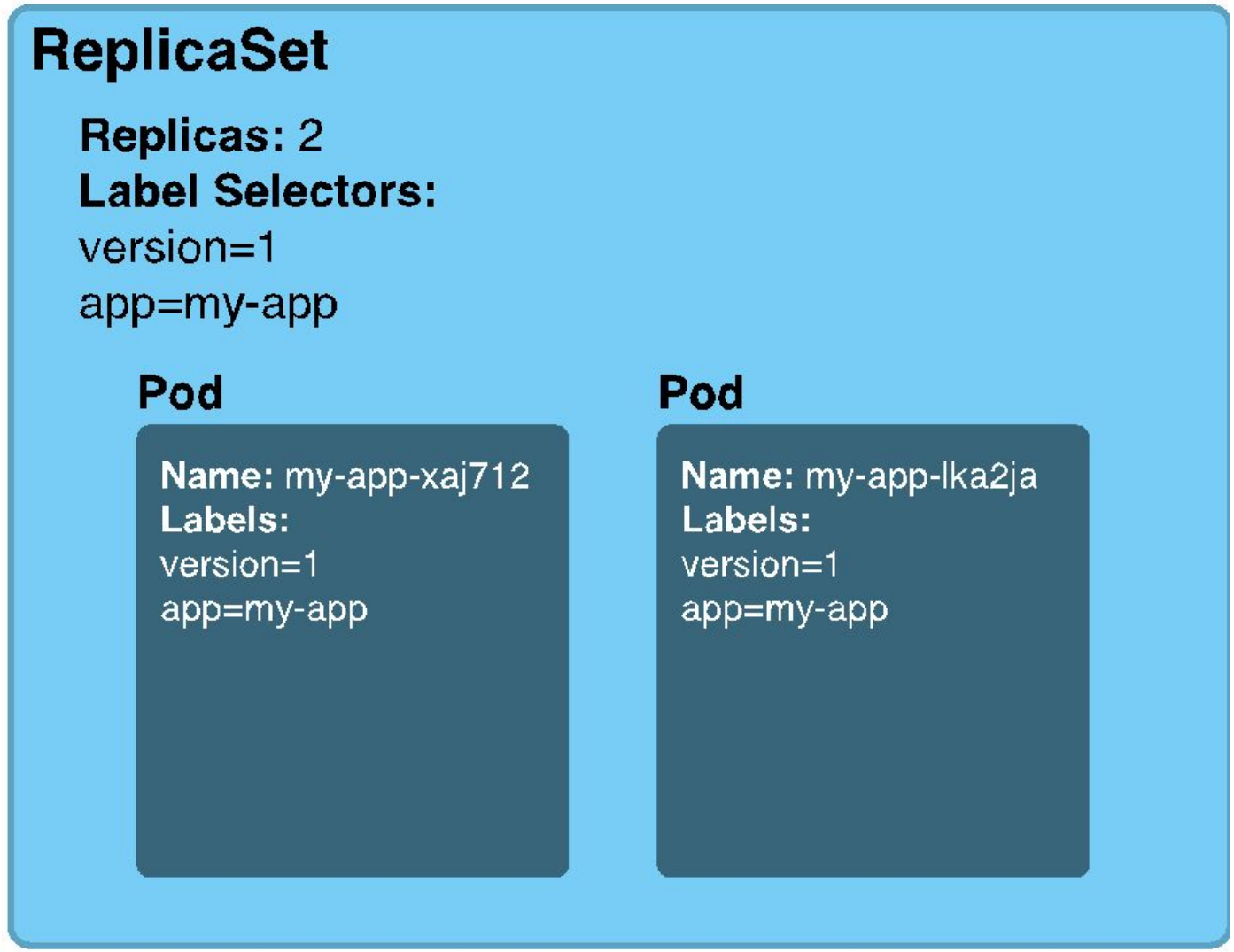
```
apiVersion: v1
kind: Service
metadata:
  name: my-first-service
spec:
  selector:
    app: web
  type: ClusterIP
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

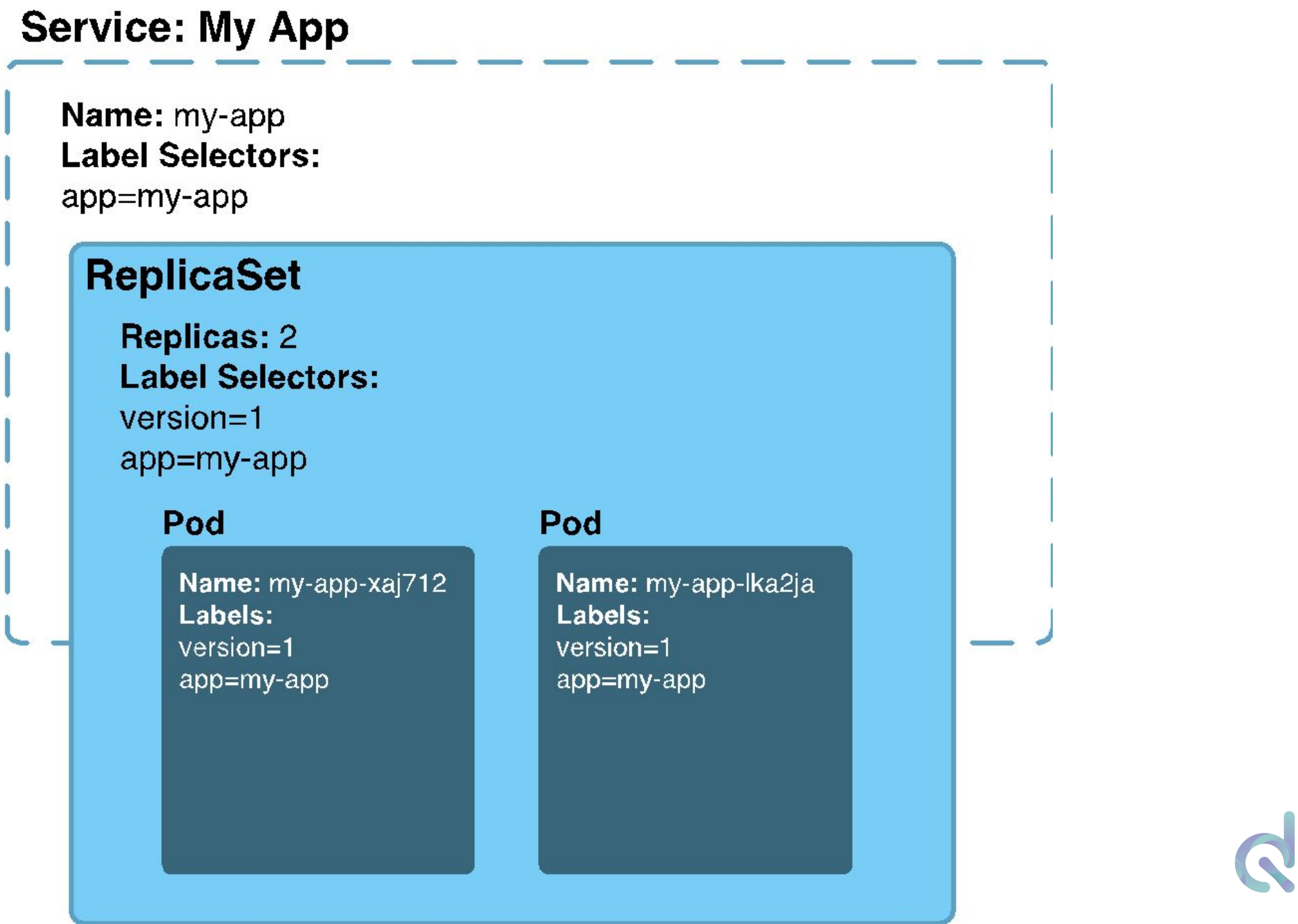
```
apiVersion: v1
kind: Service
metadata:
  name: my-first-service
spec:
  selector:
    app: web
  type: NodePort
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
    nodePort: 30008
```

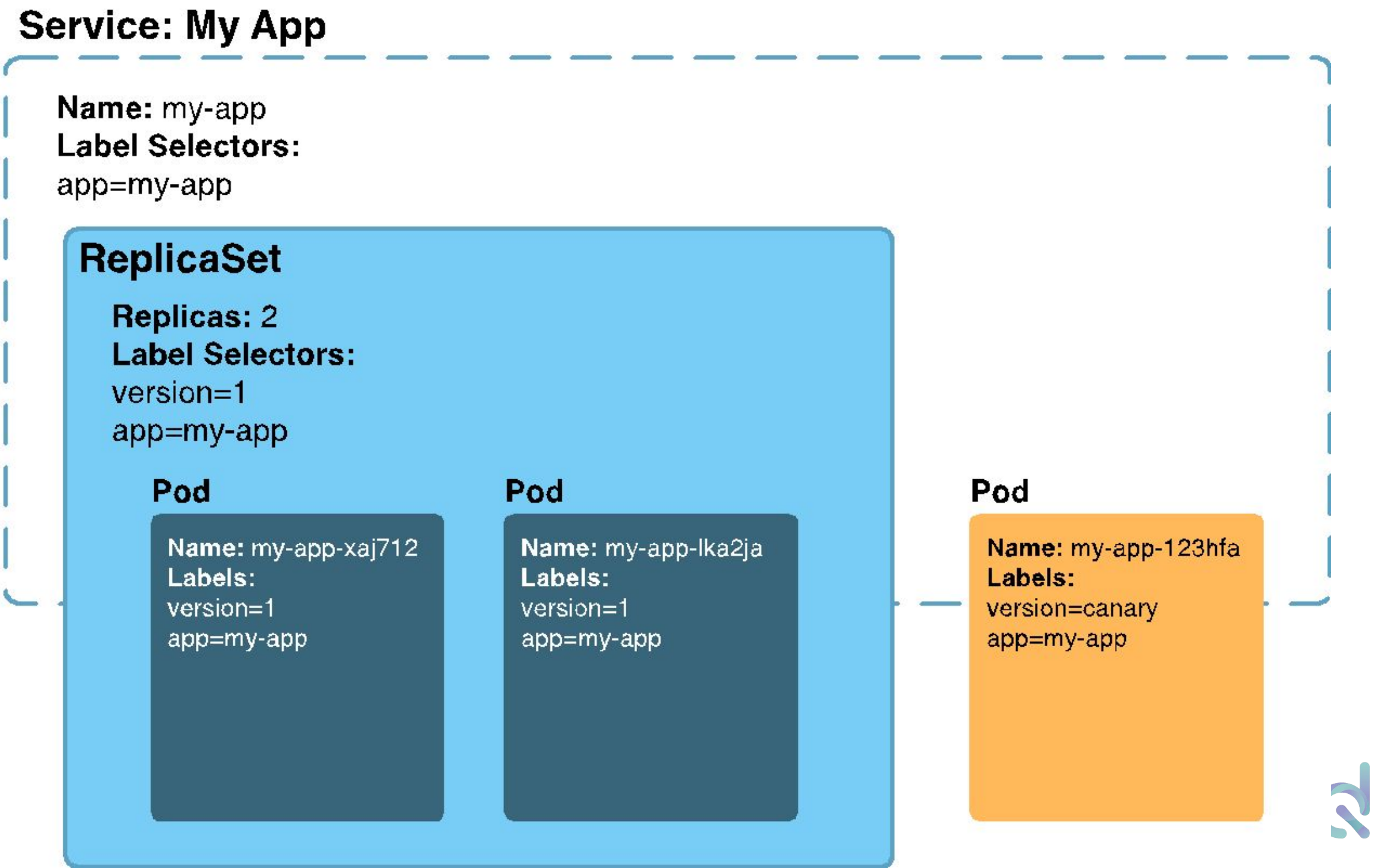
Pod

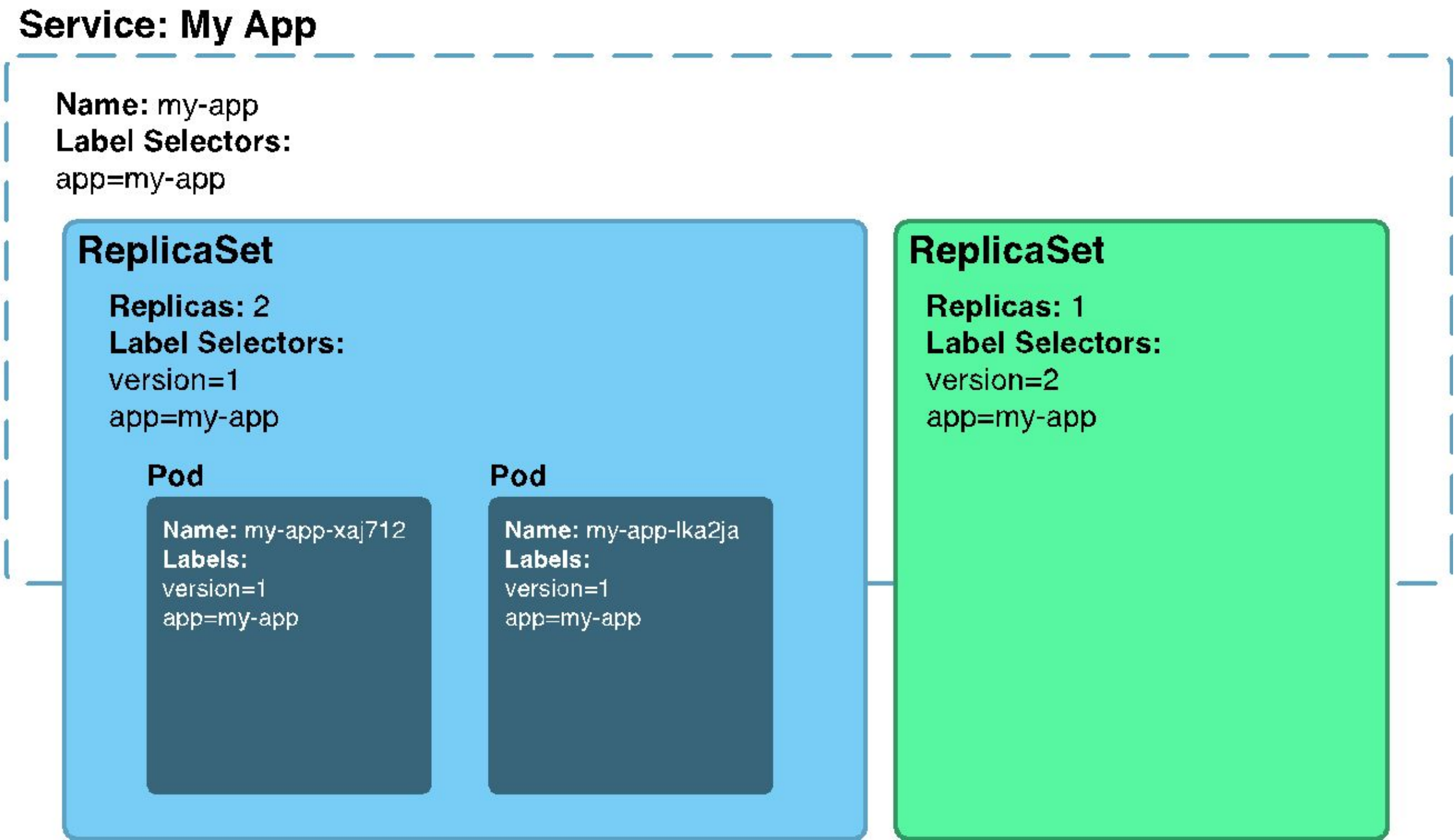
Name: my-app-xaj712
Labels:
version=1
app=my-app

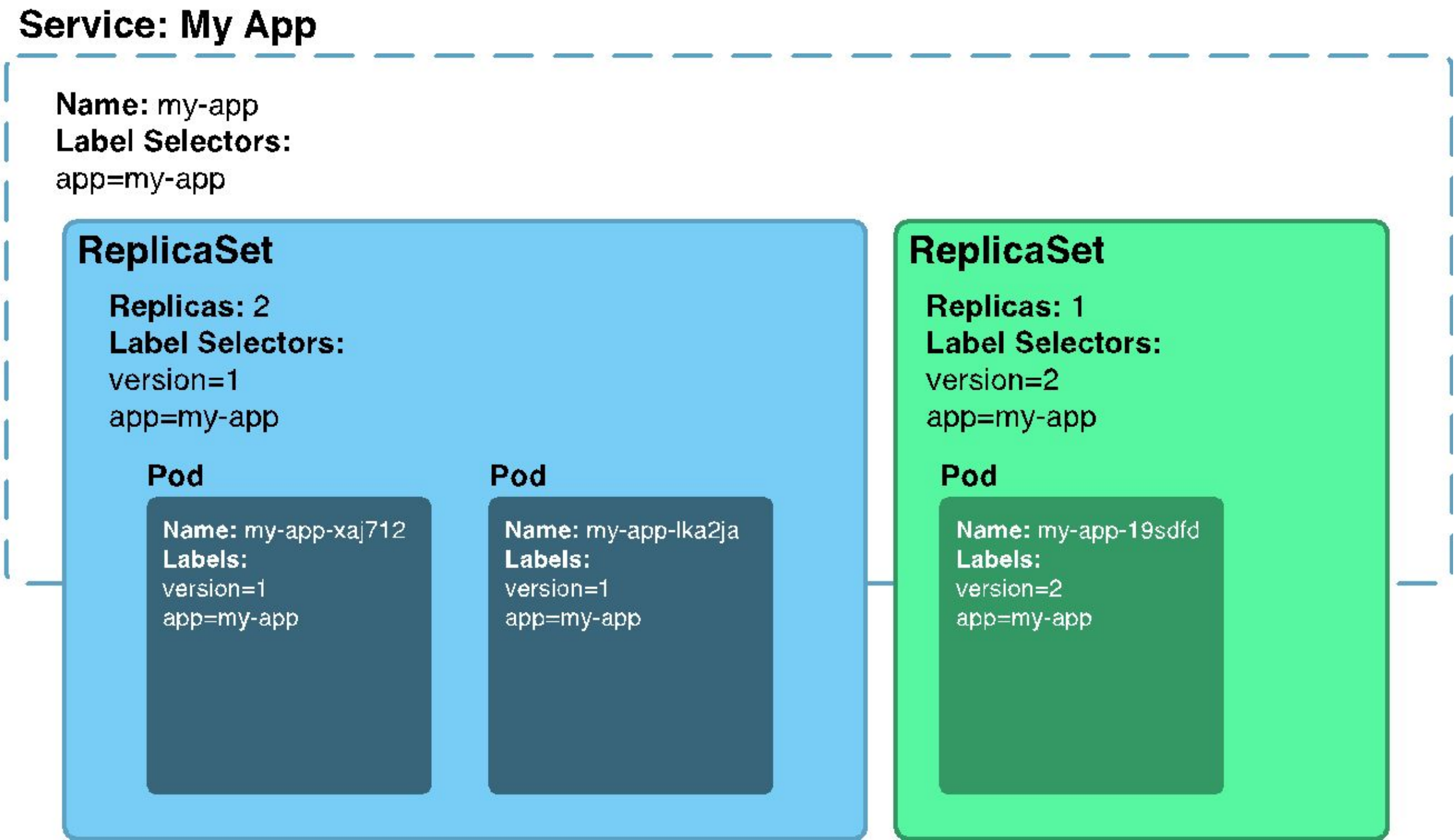

POD
Node
Helper
Containers

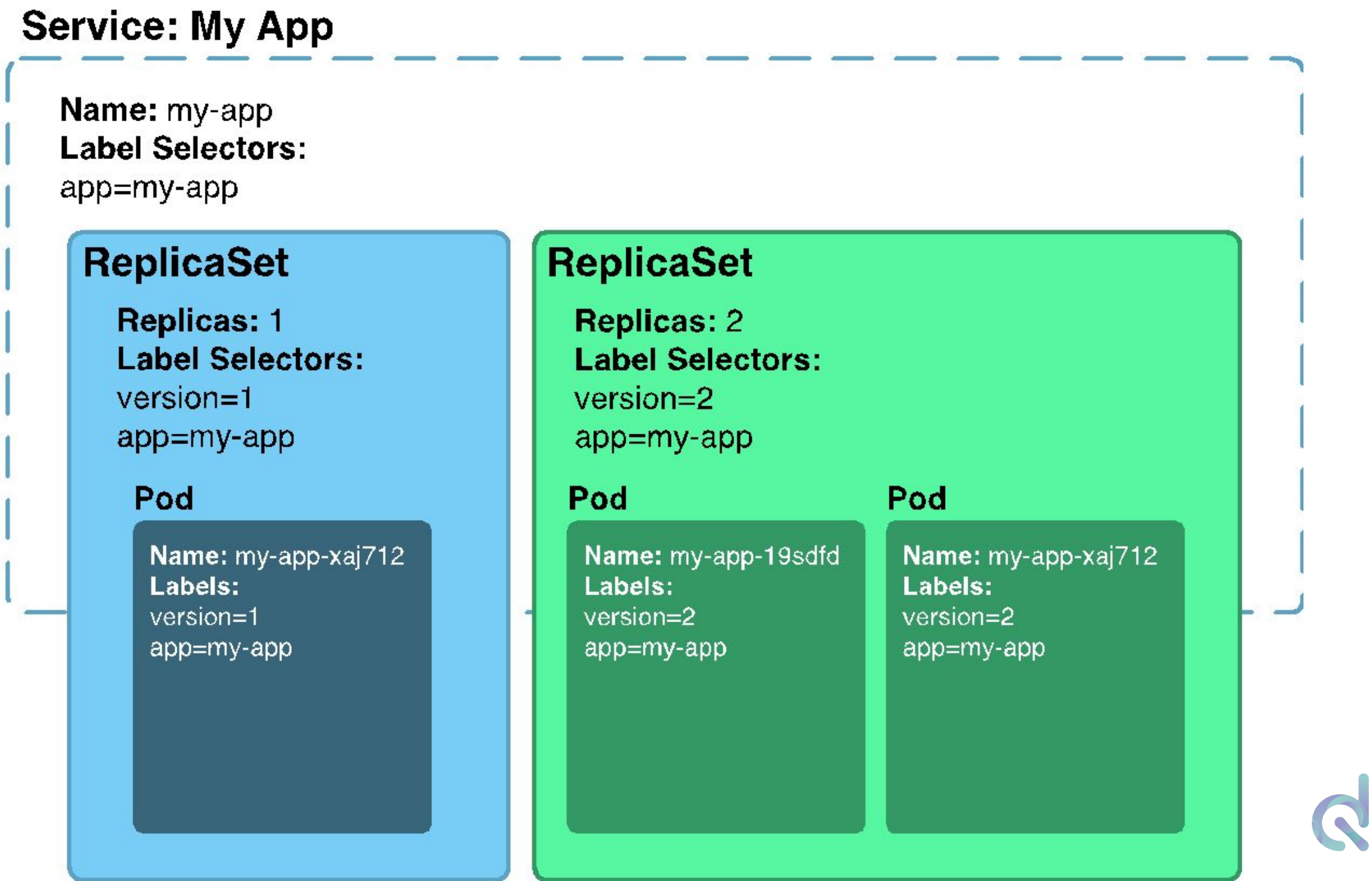

ReplicaSet
Replicas: 2
Label Selectors:
version=1
app=my-app
Pod
Name: my-app-xaj712
Labels:
version=1
app=my-app
Pod
Name: my-app-lka2ja
Labels:
version=1
app=my-app

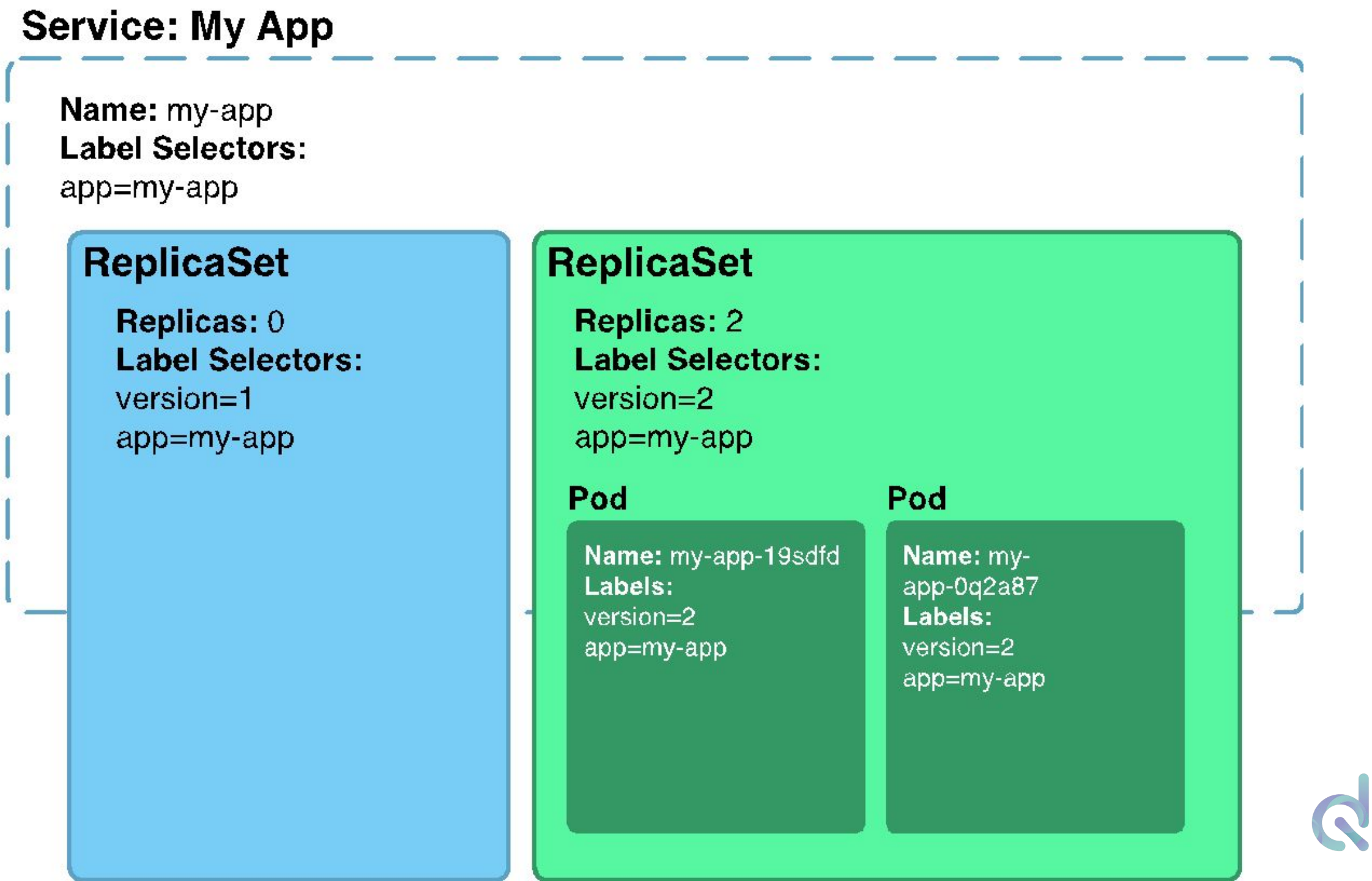

Service: My App
Name: my-app
Label Selectors:
app=my-app
ReplicaSet
Replicas: 2
Label Selectors:
version=1
app=my-app
Pod
Name: my-app-xaj712
Labels:
version=1
app=my-app
Pod
Name: my-app-lka2ja
Labels:
version=1
app=my-app


Service: My App
Name: my-app
Label Selectors:
app=my-app
ReplicaSet
Replicas: 2
Label Selectors:
version=1
app=my-app
Pod
Name: my-app-xaj712
Labels:
version=1
app=my-app
Pod
Name: my-app-lka2ja
Labels:
version=1
app=my-app
Pod
Name: my-app-123hfa
Labels:
version=canary
app=my-app


Service: My App
Name: my-app
Label Selectors:
app=my-app
ReplicaSet
Replicas: 2
Label Selectors:
version=1
app=my-app
Pod
Name: my-app-xaj712
Labels:
version=1
app=my-app
Pod
Name: my-app-lka2ja
Labels:
version=1
app=my-app
ReplicaSet
Replicas: 1
Label Selectors:
version=2
app=my-app


Service: My App
Name: my-app
Label Selectors:
app=my-app
ReplicaSet
Replicas: 2
Label Selectors:
version=1
app=my-app
Pod
Name: my-app-xaj712
Labels:
version=1
app=my-app
Pod
Name: my-app-lka2ja
Labels:
version=1
app=my-app
ReplicaSet
Replicas: 1
Label Selectors:
version=2
app=my-app
Pod
Name: my-app-19sdfd
Labels:
version=2
app=my-app


Service: My App
Name: my-app
Label Selectors:
app=my-app
ReplicaSet
Replicas: 1
Label Selectors:
version=1
app=my-app
Pod
Name: my-app-xaj712
Labels:
version=1
app=my-app
ReplicaSet
Replicas: 2
Label Selectors:
version=2
app=my-app
Pod
Name: my-app-19sdfd
Labels:
version=2
app=my-app
Pod
Name: my-app-xaj712
Labels:
version=2
app=my-app


Service: My App
Name: my-app
Label Selectors:
app=my-app
ReplicaSet
Replicas: 0
Label Selectors:
version=1
app=my-app
ReplicaSet
Replicas: 2
Label Selectors:
version=2
app=my-app
Pod
Name: my-app-19sdfd
Labels:
version=2
app=my-app
Pod
Name: my-app-0q2a87
Labels:
version=2
app=my-app

Práctica 10 - Servicios en K8S

- **Desplegar una aplicación con un deployment, por ejemplo usamos la imagen** xstabel/dotnet-core-publish

  Esperar que el POD esté en estado “Ready”:

- **Exponer el Servicio**

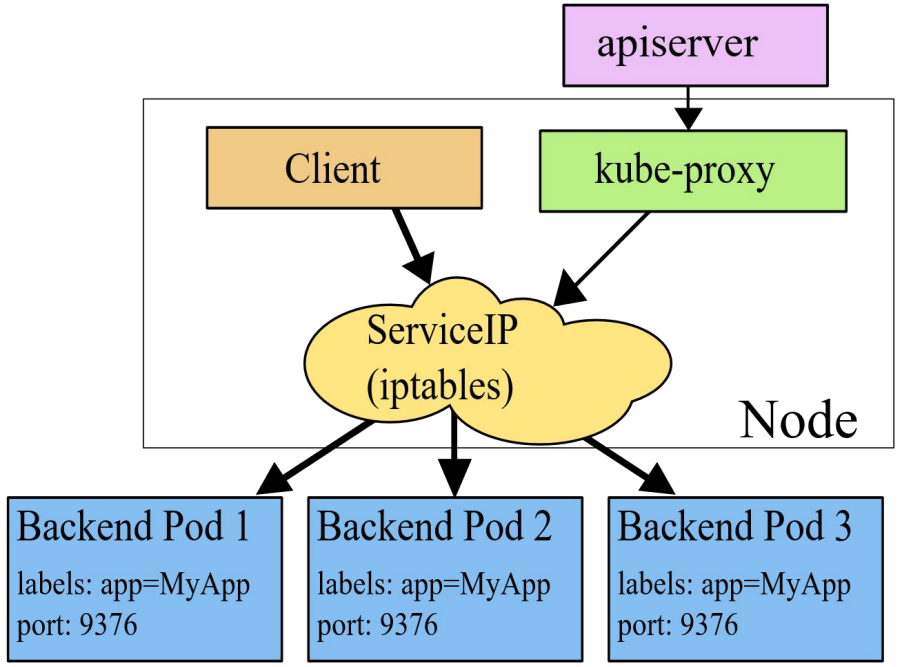
- **Acceder al servicio … hacemos un Curl al endpoint**

Networking

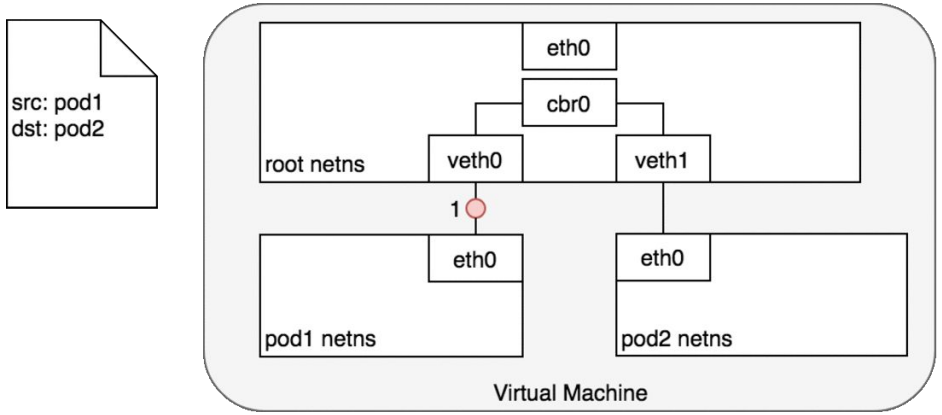
Networking in Kubernetes

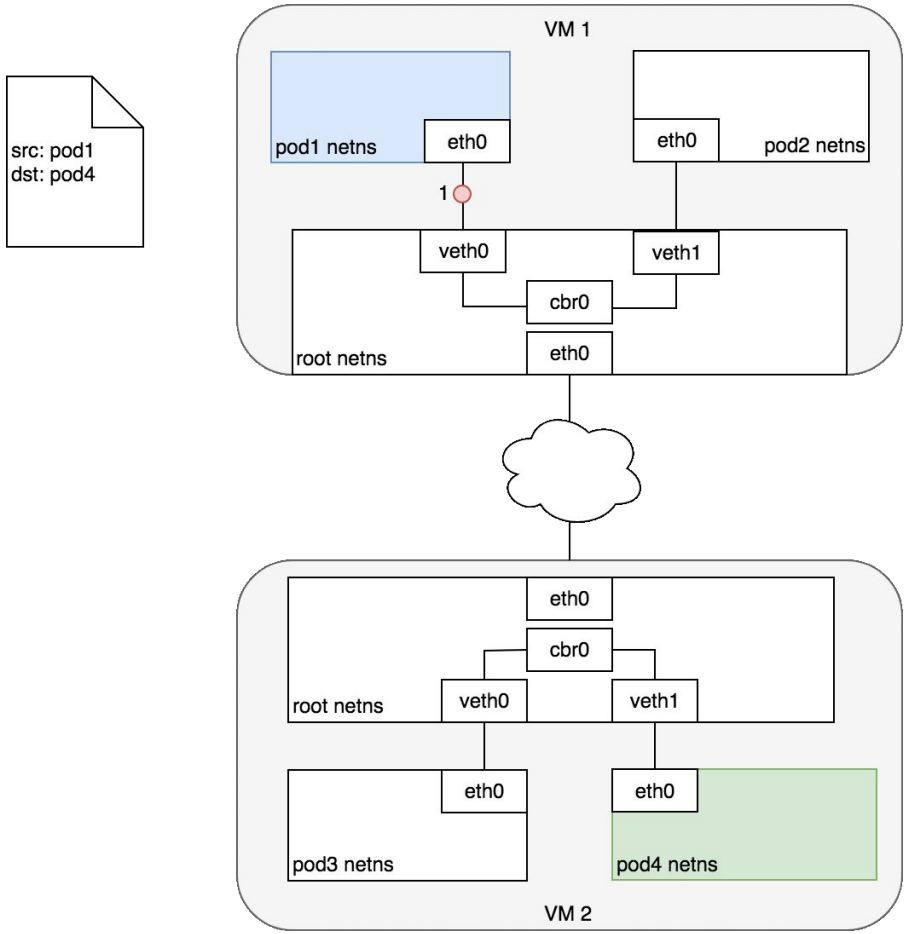
Kubernetes Networking model introduces 3 methods of communications:

- **Pod-to-Pod** communication directly by IP address. Kubernetes has a Pod-IP wide metric simplifying communication.
- **Pod-to-Service** Communication – Client traffic is directed to service virtual IP by iptables rules that are modified by the kub-proxy process (running on all hosts) and directed to the correct Pod.
- **External-to-Internal** Communication – external access is captured by an external load balancer which targets nodes in a cluster. The Pod-to-Service flow stays the same.

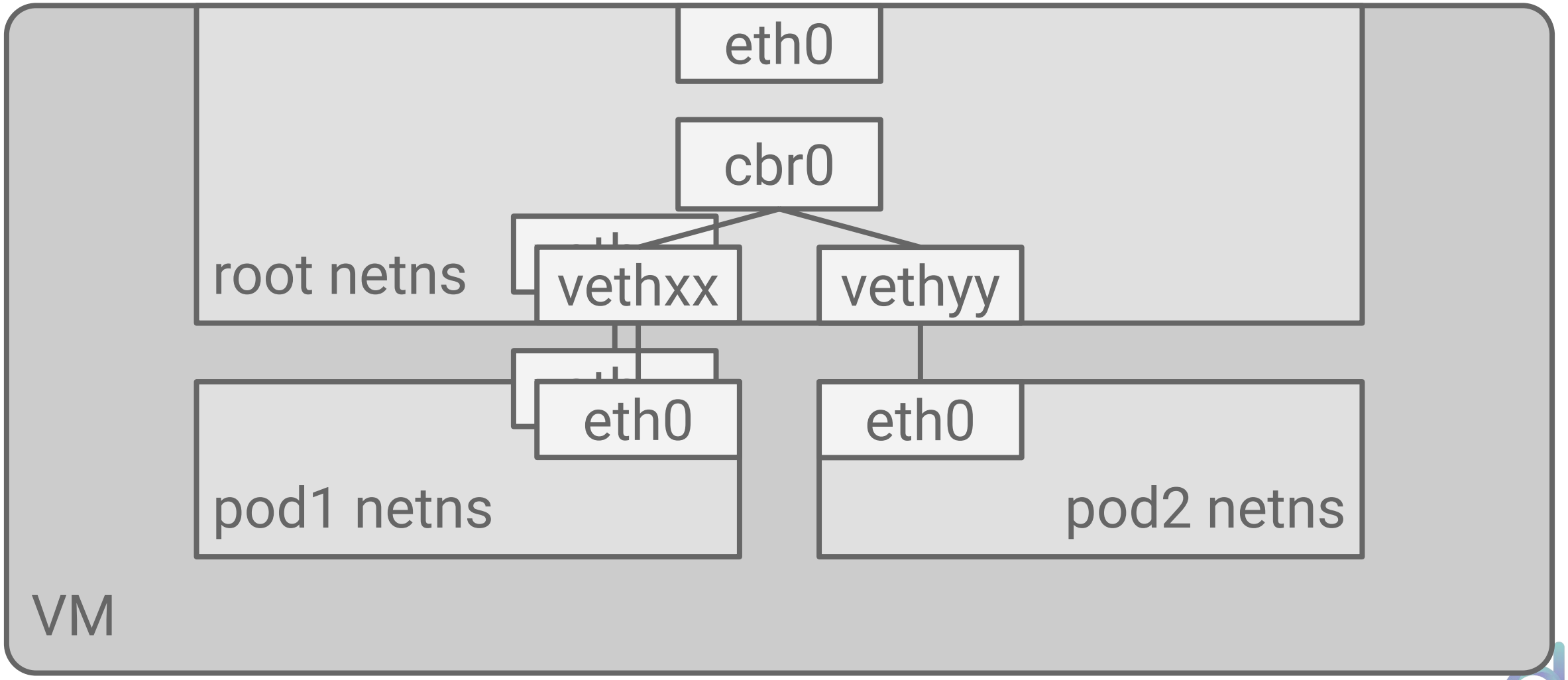


Networking in Kubernetes



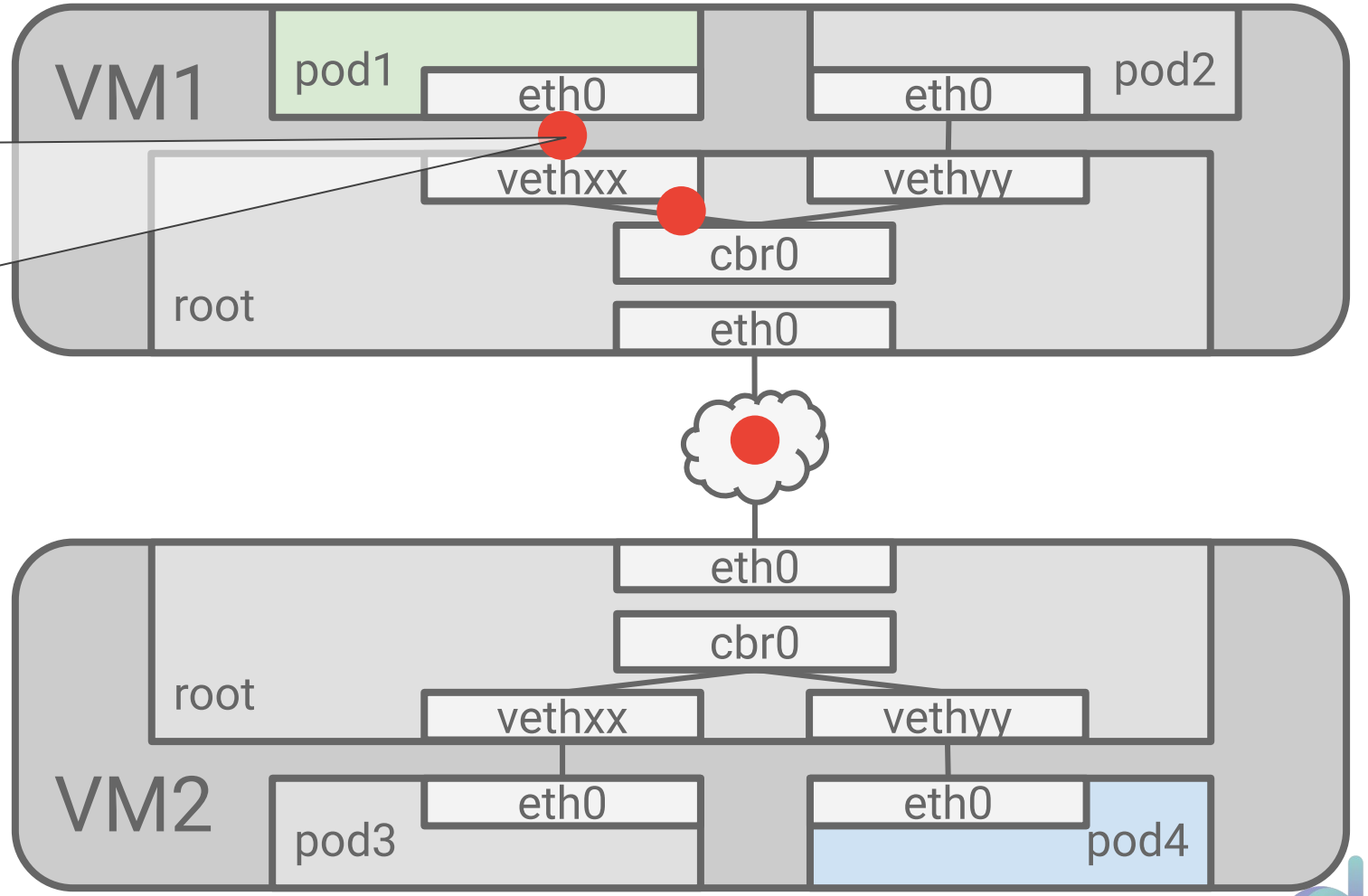


Pod networking



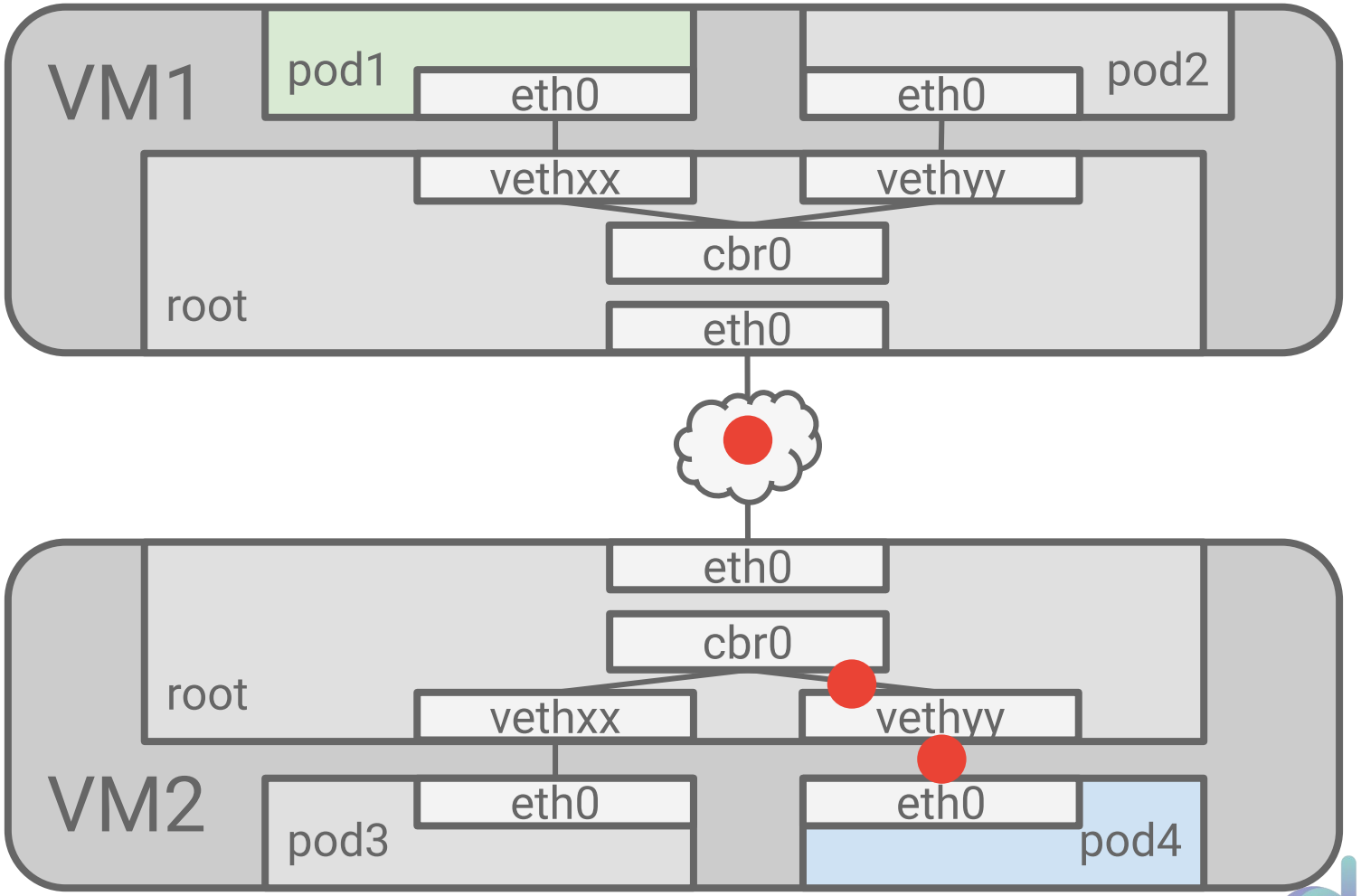
Life of a packet: pod-to-pod

src: pod1
dst: pod4



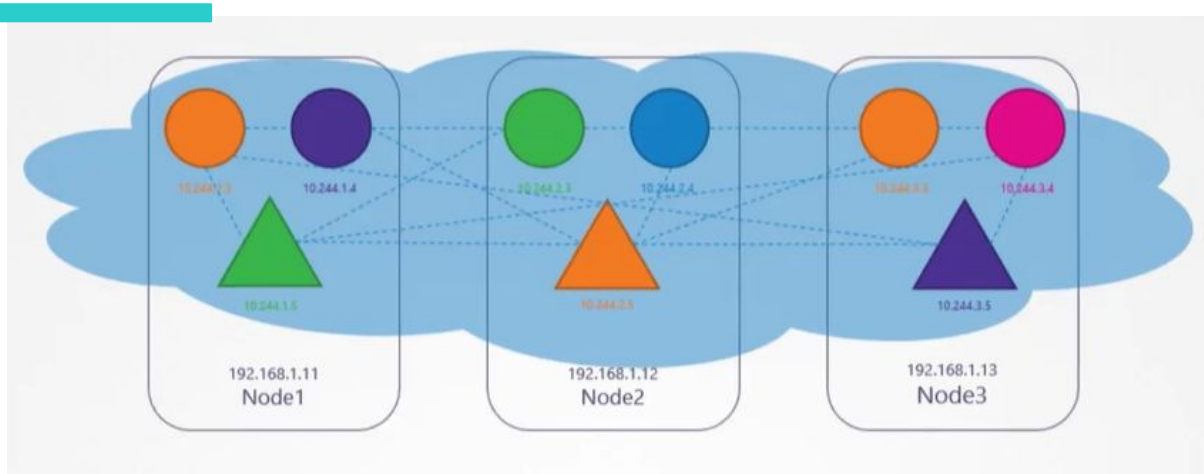
Google Cloud

Life of a packet: pod-to-pod



Google Cloud

Network Policies



Allow and Deny: ingress and egress rules

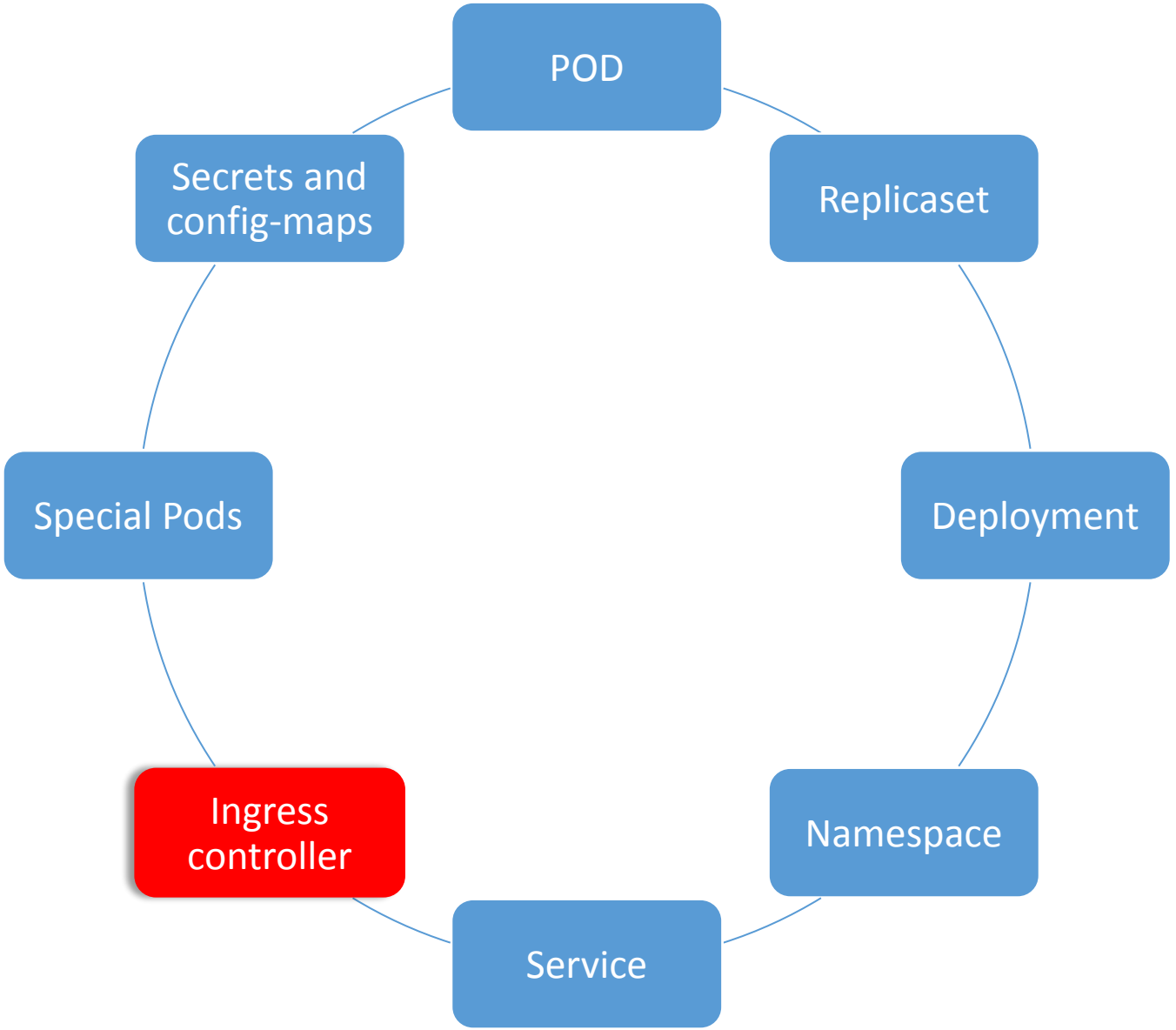
Solutions that support Network Policies:

- Kube-router
- Calico
- Romana
- Weave-net

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: db-policy
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          name: api-pod
      namespaceSelector:
        matchLabels:
          name: prod
    ports:
    - protocol: TCP
      port: 3306
```

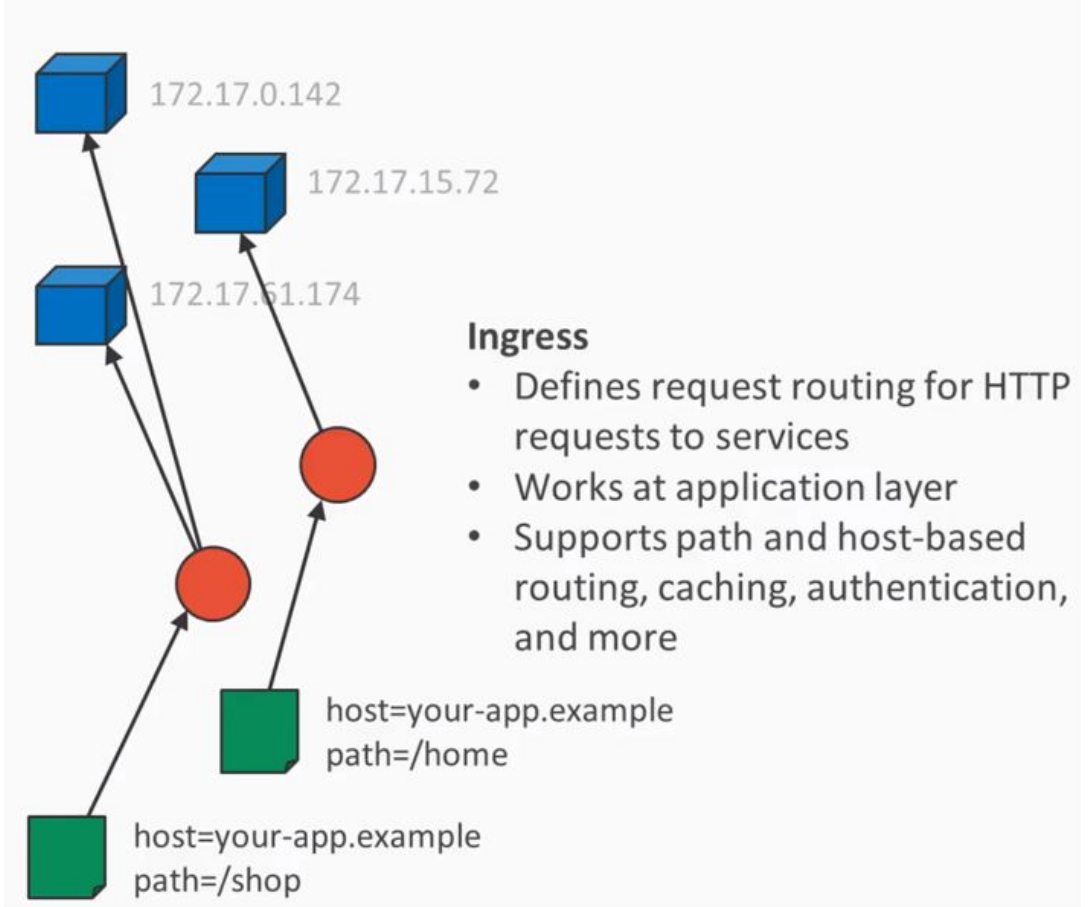
```yaml
policyTypes:
- Ingress
- Egress
ingress:
- from:
  - podSelector:
      matchLabels:
        name: api-pod
  ports:
  - protocol: TCP
    port: 3306
egress:
- to:
  - ipBlock:
      cidr: 192.168.5.10/32
  ports:
  - protocol: TCP
    port: 80
```

Componentes en K8S



Ingresses

- Typically, services and pods have IPs only routable by the cluster network
- All traffic that ends up at an edge router is either dropped or forwarded elsewhere
- Ingress is a collection of rules that allow inbound connections to reach the cluster services
- An Ingress controller is responsible for fulfilling the Ingress, usually with a loadbalancer, though it may also configure your edge router or additional frontends to help handle the traffic in an HA manner



Ingress Controller

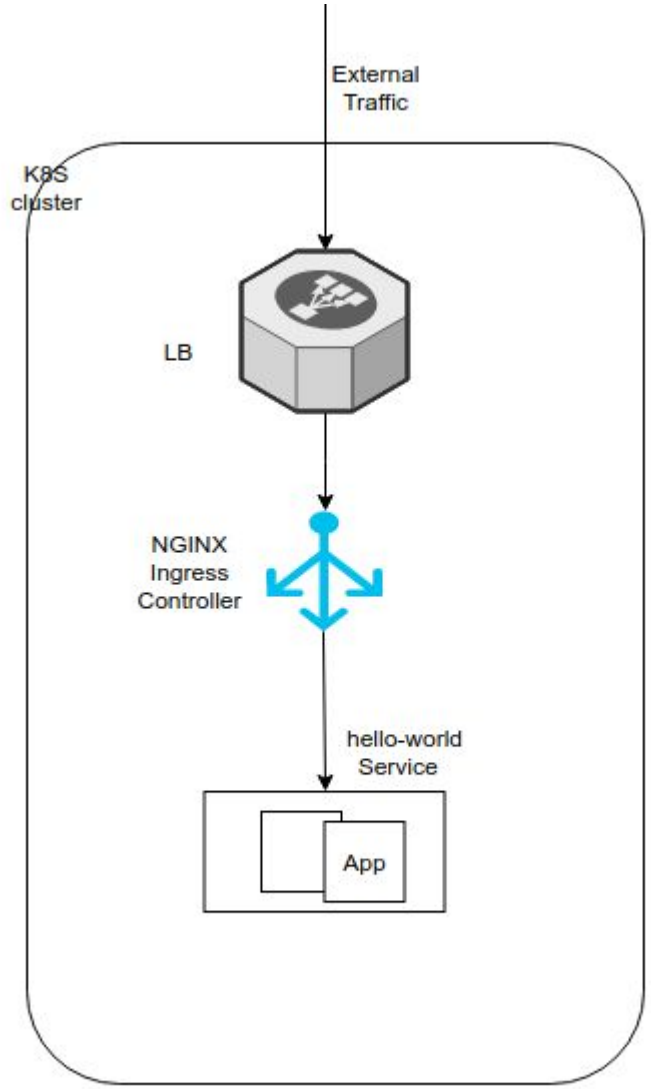
Using Helm to deploy a NGINX ingress controller

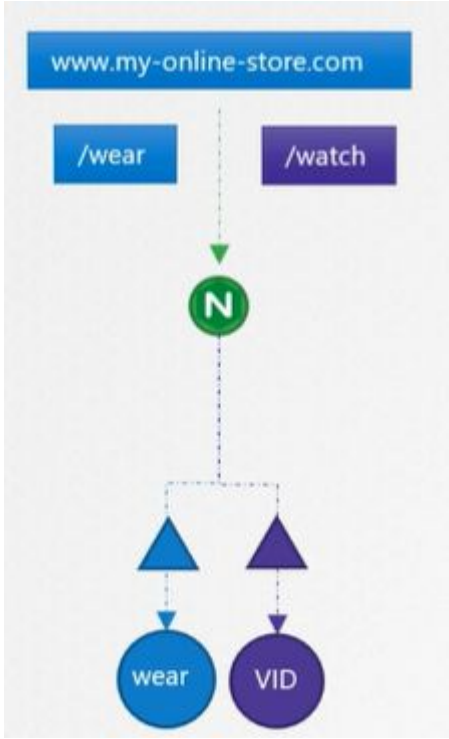
kubectl create namespace ingress-basic

helm install nginx-ingress stable/nginx-ingress \
--namespace ingress-basic \
--set controller.replicaCount=2 \
--set controller.nodeSelector."beta\.kubernetes\.io/os"=linux \
--set defaultBackend.nodeSelector."beta\.kubernetes\.io/os"=linux \
--set controller.service.loadBalancerIP="STATIC_IP" \...

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress-tripviewer
  namespace: api-dev
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
 #   nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
  rules:
  - http:
      paths:
      - backend:
          serviceName: poisvc
          servicePort: 80
        path: /api/poi(.*)
      - backend:
          serviceName: tripssvc
          servicePort: 80
        path: /api/trips(.*)
      - backend:
          serviceName: userprofilesvc
          servicePort: 80
        path: /api/user(.*)
      - backend:
          serviceName: userjavasvc
          servicePort: 80
        path: /api/user-java(.*)
```

Ingress Controller





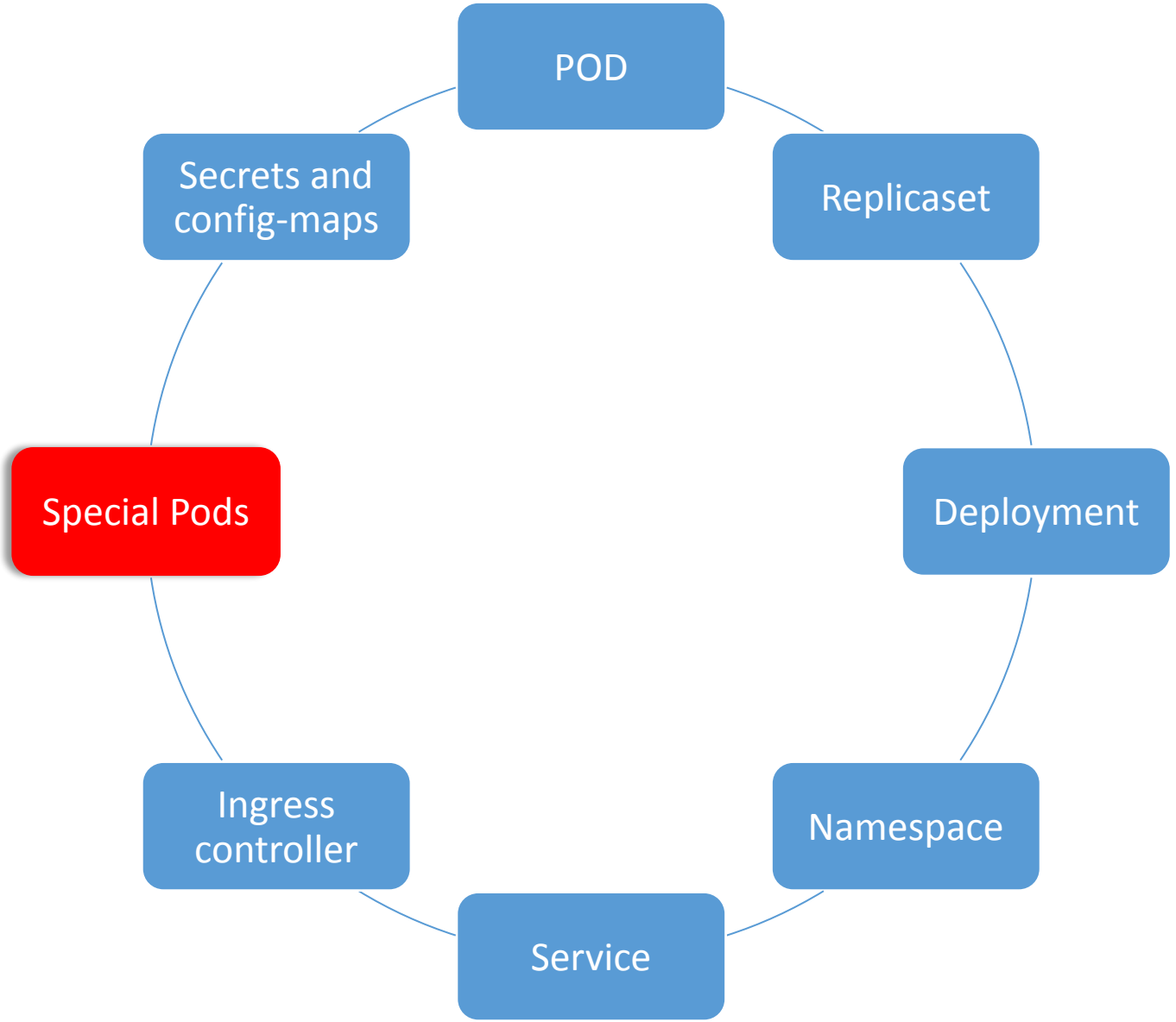
Práctica 11 Ingress Controller

- **Habilitar Ingress controller en minikube**

  $ minikube addons enable ingress

- **Verificar que el ingress controller está corriendo**
- **Desplegar una aplicación**
- **Exponer el servicio**
- **Crear un ingress que envíe tráfico a la aplicación**
- **Verificar el ingress**
- **Desplegar otra aplicación imagen gcr.io/google-samples/hello-app:2.0 , exponer servicio y añadir al ingress**

Componentes en K8S



Daemonsets

- Ensures that all (or some) nodes run a copy of a pod
- Doesn't care if a node is marked as unschedulable
- Can make pods even if the scheduler is not running

StatefulSets

- Used for workloads that require consistent, persistent hostnames e.g. etcd-01, etcd-02
- One PersistentVolume storage per pod
- StatefulSet example - zookeeper
- StatefulSet example - cockroachdb

Jobs

- Creates one or more pods and ensures that a specified number of them successfully terminate
- 3 types of jobs
  - Non-parallel Jobs - e.g. single pod done when exit is successful
  - Parallel Jobs with a fixed completion count - e.g. one successful pod for each value in the range 1 to .spec.completions
  - Parallel Jobs with a work queue that coordinate with each other and terminate when one pod exits successfully

Scaling

Escalar aplicaciones en K8S

- Los objetos ReplicaSet en K8S nos sirven para indicar la cantidad de instancias de un POD en la plataforma.
- Los ReplicaSets se pueden especificar directamente en los despliegues (Deployment YAML)
- También se puede configurar el autoescalado dependiendo de los recursos disponibles.

Escalar aplicaciones en K8S

- El Horizontal POD Autoscaler (HPA) es una funcionalidad que permite ajustar las réplicas de un POD para que cumpla con la utilización que se ha especificado.
- Existen algunas opciones de configuración que permiten mantener los recursos en un estado aceptable.

```
kubectl autoscale <Deployment-name> --min=10 --max=15 --cpu-percent=80
```

Práctica 12 - Autoescalado

- Se puede escalar cualquiera de nuestras aplicaciones utilizando HPA, en este ejercicio usaremos este deployment

  https://k8s.io/examples/application/php-apache.yaml

- Aplicar HPA a nuestro despliegue, de min 1 max 10 y cuando la CPU esté a 40%

- Probar HPA con mucha carga, vamos a mandar muchas peticiones a nuestra aplicación web de Apache

kubectl run -i --tty load-generator --rm --image=busybox --restart=Never -- /bin/sh -c ‘while sleep 0.01; do wget -q -O- http://php-apache; done’

Vemos que pasa con nuestro HPA como va cambiando kubectl get hpa XXXX -o yaml

More K8S features…

Role Based Access Control

- Allows for dynamic policies against k8s API for authZ
- Can create both ”Roles” and “ClusterRoles”
- To grant permissions:
  - Use ”RoleBindings” for within namespaces
  - Use “ClusterRoleBindings” for cluster-wide access
  - Can contain users, groups, service accounts
  - Control access to specific resources or API verbs

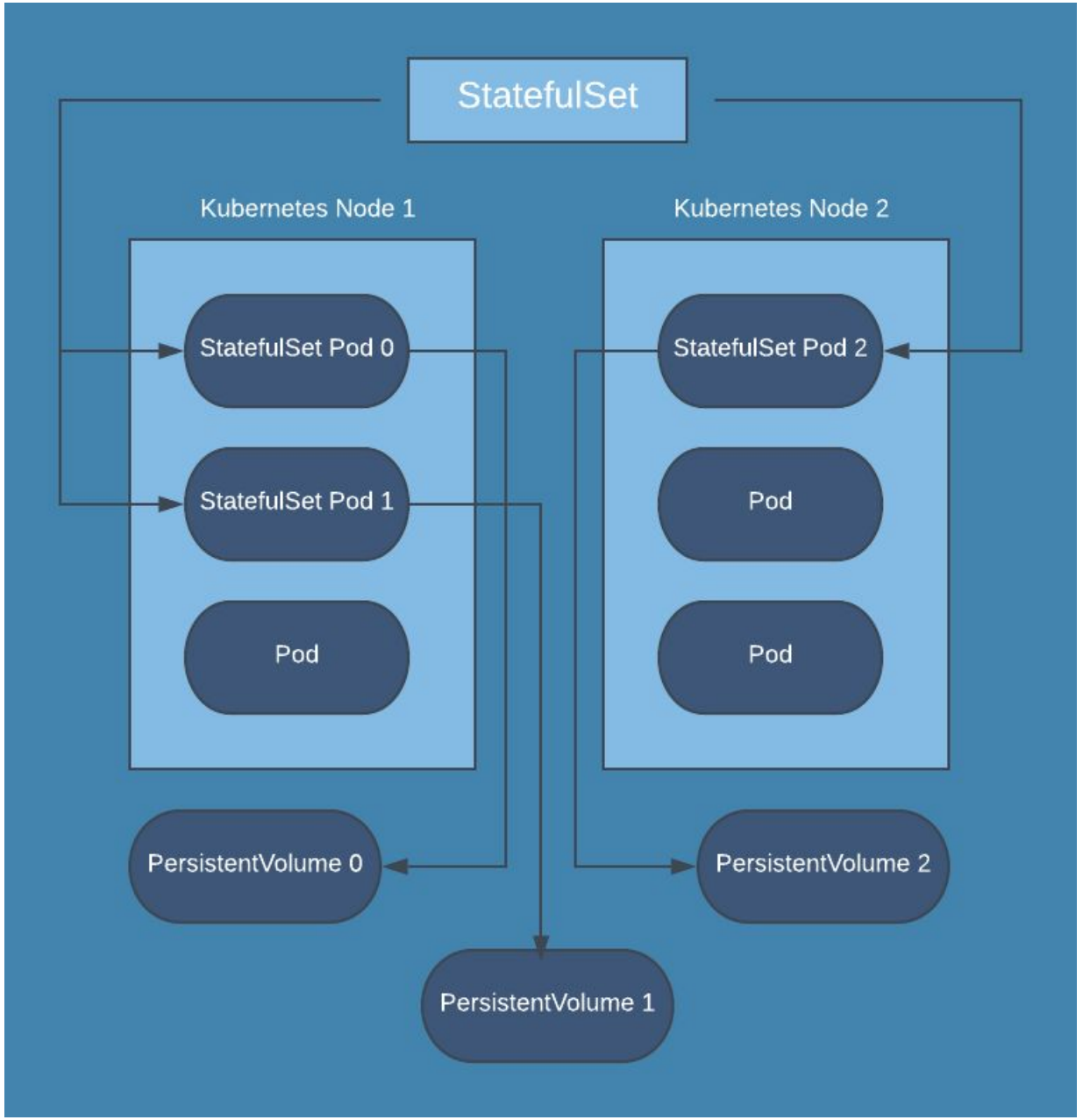
```
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["batch", "extensions"]
  resources: ["jobs"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

StatefulSet

- StatefulSet is the workload API object used to manage stateful applications.
- Manages the deployment and scaling of a set of Pods, and provides guarantees about the ordering and uniqueness of these Pods.

Headless Services

- Sometimes you don’t need load-balancing and a single Service IP. In this case, you can create what are termed “headless” Services, by explicitly specifying "None" for the cluster IP (.spec.clusterIP).
- You can use a headless Service to interface with other service discovery mechanisms, without being tied to Kubernetes’ implementation.



Operators

Operators extend Kubernetes functionality

- Extend the power of Kubernetes, in a far more manageable and accessible fashion than before. As the coreOS announcement introducing the original etcd Operator and Prometheus Operator said, “stateless is easy, **stateful is hard**. A great example of this is database operators, which allow Kubernetes users to safely deploy and manage certain databases without needing to build their own workarounds.

Operators systematize human knowledge as code

- Kubernetes enables the automation of the infrastructure (and corresponding operational burden of managing that infrastructure) necessary for running containerized applications

Operators

Operators enable standardize

- Like templates that can be reused and adapted to automate application management, with no need to reinvent the wheel.

Operators Improve resiliency

- Operators are simplifying the process highly complex distributed database management by defining the installation, scale, updates, and management lifecycle of a stateful clustered application.

https://operatorhub.io/

Storage

Volúmenes en Kubernetes

Los volúmenes se asocian a los Pods para conservar datos y únicamente el POD puede acceder.

En K8S se soportan distintos tipos de volúmenes:

- Distintos storages de proveedores Cloud
- SAN-type, file systems etc.
- Soporte local sólo para caso de testing como en **minikube**.
- Cloud Provider: Azure Files and Azure Disks, AWS EBS, Google compute engine Persistent Disk

Algunos tipos de volúmenes podrían proveer la capacidad de compartir ficheros entre Pods.

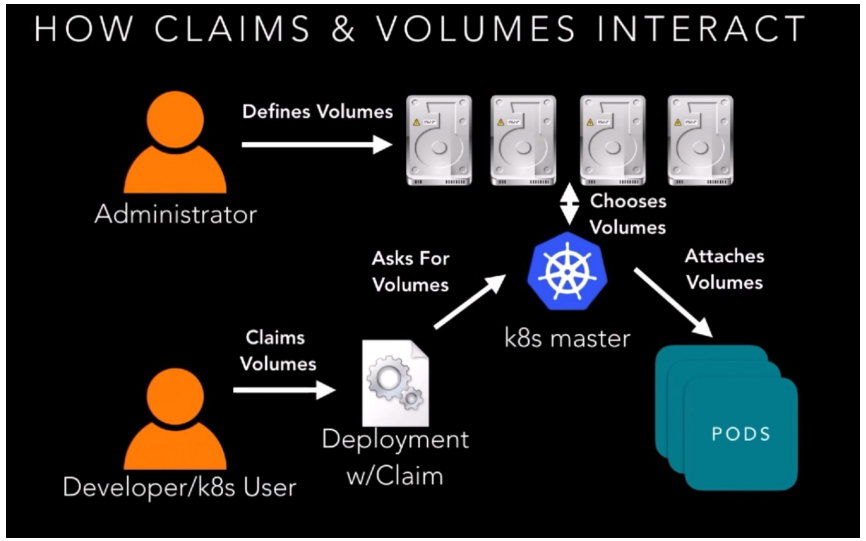
Volúmenes persistentes

Usando volúmenes persistentes

- Un volumen persistente está diseñado para proveer la funcionalidad de persistencia de datos.
- El flujo de uso de los volúmenes:
  - Aprovisionar un volumen persistente. Disco virtual o HW específico
  - Establecer una claim: PersistentVolumeClaim, esto sirve para solicitar un storage por un usuario/POD
- De esta forma, aprovisionando volumen solicitando las claims de volumen y ejecutando el POD, es como K8S pone a disposición un storage persistente.
- En el PersistentVolumeClaim se especifica el nombre, el tamaño, tipo de storage…

Volúmenes en Kubernetes

- Definir el volumen
- Solicitar un volumen:Claims Volume
- La plataforma escoge entre los
- Definidos.
- Adjuntar el volumen al Pod.



En la definición del Pod se especifica dónde y qué montar:

- En el campo spec.volumes **se indica el volumen que se necesita**.
- En el campo spec.containers.volumeMounts, **dónde montarlo**.

Volúmenes persistentes

Los PODs usan el PersistentVolumeClaims para solicitar el storage físico definido por los volúmenes persistentes.

K8S usa las claims para buscar el volumen definido que satisfaga los requerimientos.

```yaml
kind: PersistentVolume
apiVersion: v1
metadata:
  name: task-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: task-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```

Práctica 13 - Storage

- Creemos un PVC de 1Gi veamos lo que ocurre por detrás y limpiemos.
- Crear nuestra stateful app para que emplee volúmenes persistentes.
  - Partimos de una imagen NGINX k8s.gcr.io/nginx-slim:0.8
  - Creamos un volume claim de acceso "ReadWriteOnce"
  - Montamos el volumen en el path de apps: `/usr/share/nginx/html`
- Validar la creación de los PV y PVClaims

Monitoring

What indicators matter in Kubernetes

- Cluster health: total number of nodes, pods, etc
- Node health: available compute resources, health
- Application health
- Tooling Examples
  - Datadog
  - Prometheus
  - Cloud-specific: Azure Monitoring, CloudWatch (AWS)

Logging

- Pod logs: applications must log to standard out!
- Cluster-wide **event logs**
- **Logging forwarder solutions**
  - fluentd
  - logstash
- Log indexers
  - Splunk
  - ELK stack
  - Cloud providers solutions

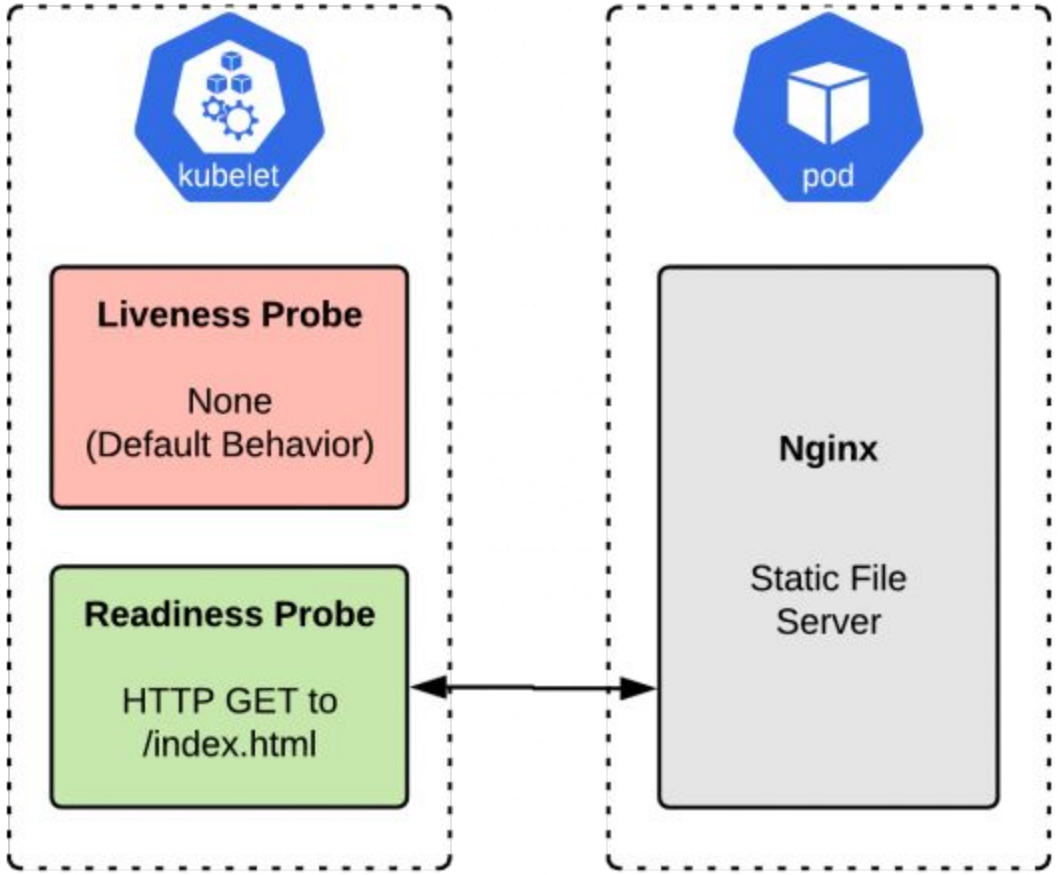
Health Checks

- Existen dos tipos de Health Checks:
  - Readiness Probes: cuando el Pod haya cargado lo que necesita internamente en la imagen y está listo para recibir requests de servicios externos.
  - Liveness Probes: determina la salud del Pod, que puede seguir recibiendo requests.

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: tomcat-deployment
spec:
  selector:
    matchLabels:
      app: tomcat
  replicas: 4
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        image: tomcat:9.0
        ports:
        - containerPort: 8080
        livenessProbe:
          httpGet:
            path: /
            port: 8080
          initialDelaySeconds: 30
          periodSeconds: 30
        readinessProbe:
          httpGet:
            path: /
            port: 8080
          initialDelaySeconds: 15
          periodSeconds: 3
```

Práctica 14

- Añadir livenessProbes
- Añadir readinessProbes



Monitoring/Logging your cluster

- Log Everything to stdout / stderr
- Key Metrics:
  - Node metrics (CPU Usage, Memory Usage, Disk Usage, Network Usage)
  - Kube_node_status_condition
  - Pod memory usage / limit; memory_failures_total
    - container_memory_working_set_bytes
  - Pod CPU usage average / limit
  - Filesystem Usage / limit
  - Network receive / transmit errors

Monitoring, Logging, and Troubleshooting

There are many free and paid packages and services for K8S and beyond:

- Performance analytics with Kubernetes dashboard
- Central logging
- Detecting problems at the node level
- Troubleshooting scenarios
- Using **Prometheus**

Monitoring on Minikube

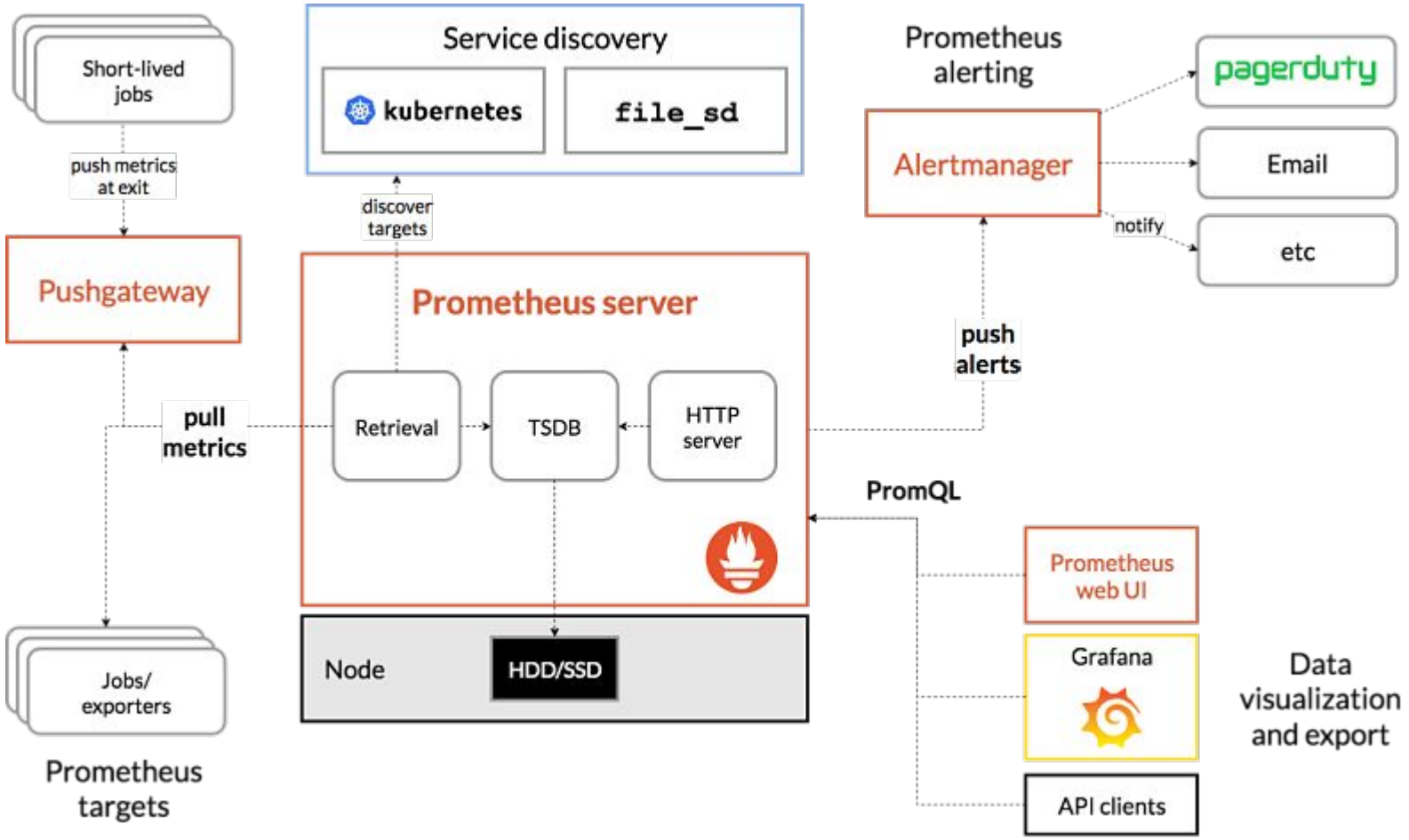
- In most of the Cloud-based platforms it comes enabled.
- On Minikube, you'll have to enable the add-on
  - Run the following command:

    $ minikube addons enable metrics
  - Wait some minutes

Monitoring - Prometheus

- Prometheus is a pull-based system. It sends an HTTP request, a so-called scrape. The response to this scrape request is stored and parsed in storage along with the metrics for the scrape itself.

- The storage is a custom database on the Prometheus server and can handle a massive influx of data. It's possible to monitor thousands of machines simultaneously with a single server.



Monitoring on Minikube

- Install Prometheus using Helm
  - Install HELM

    https://helm.sh/docs/intro/install/

    $ helm version

    $ helm repo add stable https://kubernetes-charts.storage.googleapis.com/
  - Install prometheus

$helm install prometheus stable/prometheus --namespace monitoring --dry-run

Monitoring using Prometheus

```
$ export POD_NAME=$(kubectl get pods --namespace monitoring -l "app=prometheus,component=server" -o jsonpath="{.items[0].metadata.name}")
```

```
$ kubectl --namespace monitoring port-forward $POD_NAME 9090
```

Consultas en Prometheus: https://prometheus.io/docs/prometheus/latest/querying/basics/

Monitoring using Grafana

$helm install stable/grafana

1. Get your 'admin' user password by running:

$ kubectl get secret --namespace default mygrafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo

2. The Grafana server can be accessed via port 80 on the following DNS name from within your cluster:

$ mygrafana.default.svc.cluster.local

3. Get the Grafana URL to visit by running these commands in the same shell:

$ export POD_NAME=$(kubectl get pods --namespace default -l "app=grafana,release=mygrafana" -o jsonpath="{.items[0].metadata.name}")

$ kubectl --namespace default port-forward $POD_NAME 3000

4. Login with the password from step 1 and the username: admin

Monitoring using Grafana

List minikube service to get Grafana URL

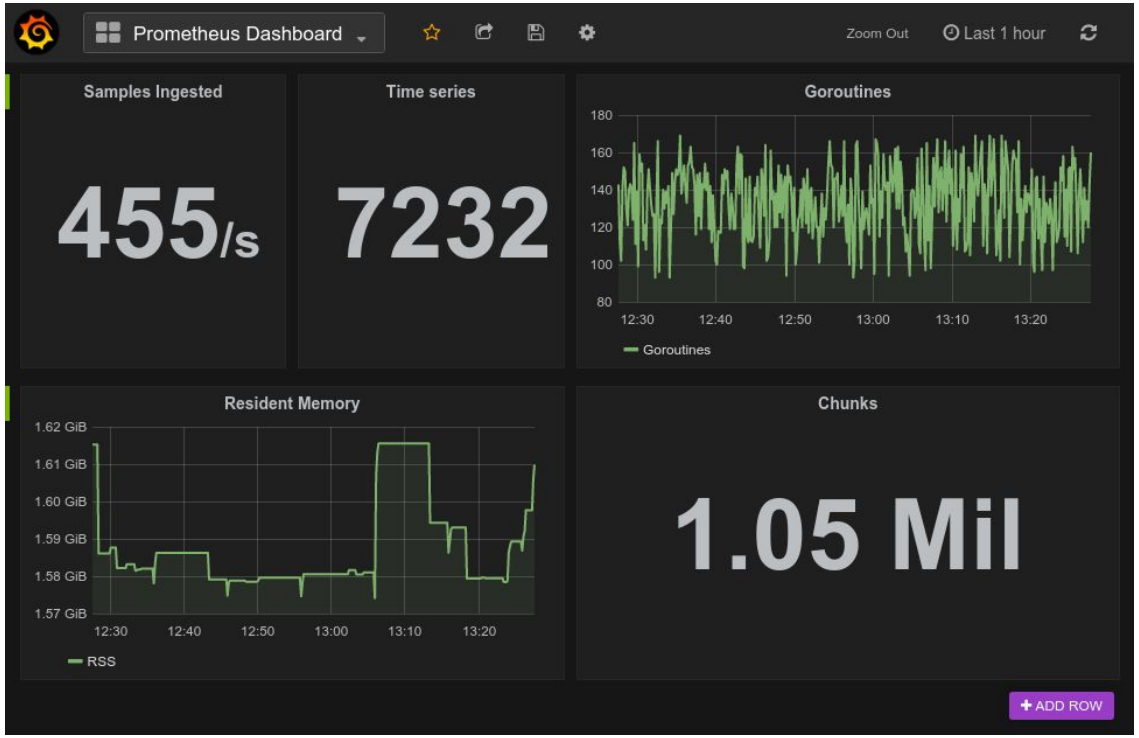
$minikube service list

- Sample using Grafana Dashboards downloading JSON (https://grafana.com/grafana/dashboards/10856)
- Open http://grafana_url/dashboard/import
- Upload JSON, configure Prometheus as Data source

Monitoring: Using Prometheus and Grafana

Once the Prometheus Operator is up and running along with Grafana and the Alertmanager, you can access their UIs and interact with the different components:

- Prometheus UI on node port 30900
- Alertmanager UI on node port 30903
- Grafana on node port 30902
- Prometheus supports a dizzying array of metrics to choose from.



Práctica 15

- Habilitamos monitoring en nuestro entorno
- Mandamos carga a nuestra app veamos las queries en prometheus y grafana

Tolerancias y Afinidades en K8S

Nodes Selections

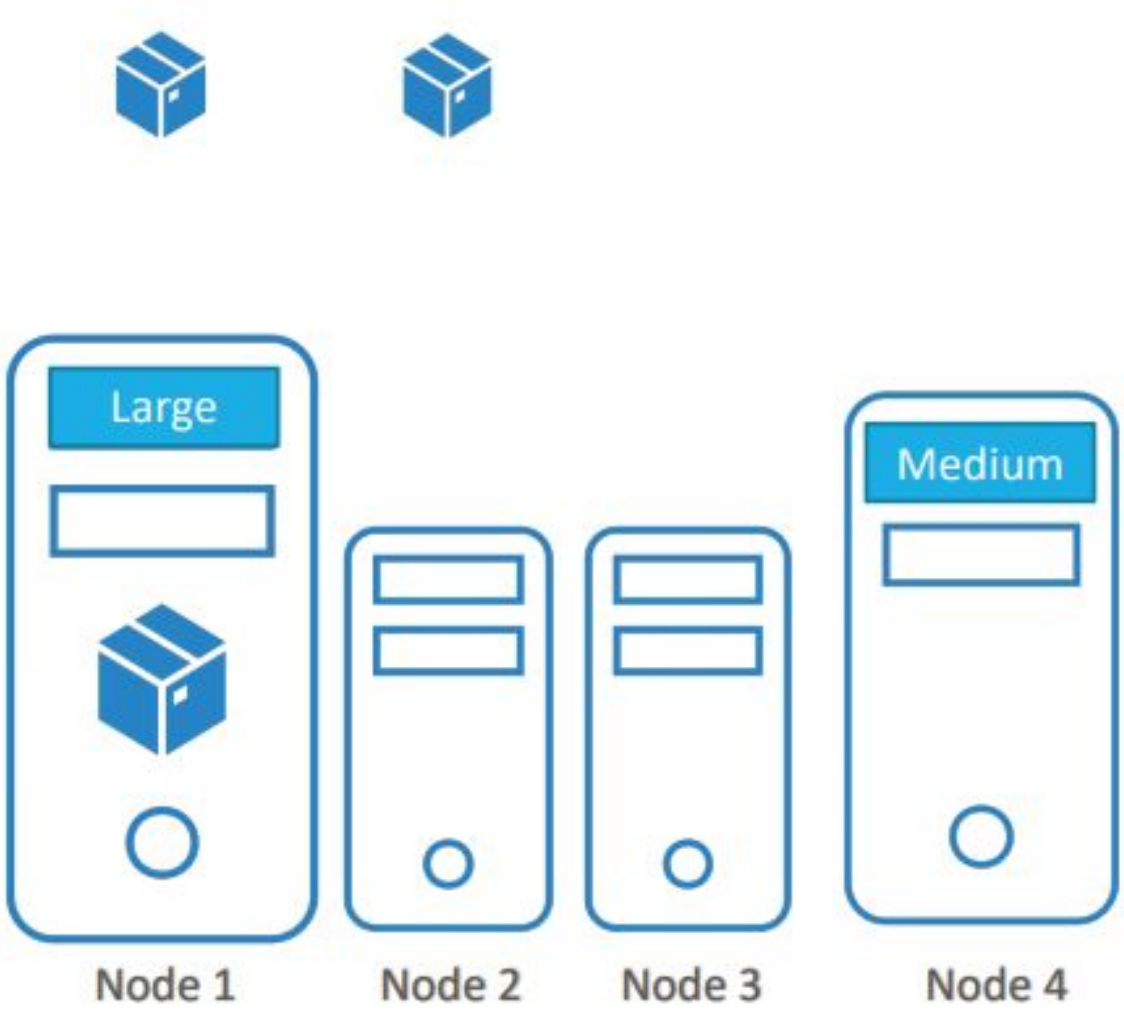
Node Selector

- Label selectors can also be used to identify a subset of the nodes in a Kubernetes cluster that should be used for scheduling a particular Pod.
- By default, all nodes in the cluster are potential candidates for scheduling, but by filling in the spec.nodeSelector field in a Pod or PodTemplate, the initial set of nodes can be reduced to a subset.

kubectl label node aks-nodepool1 hardware:highmem

```yaml
kind: Pod
apiVersion: v1
metadata:
  name: tf-mnist
spec:
  containers:
  - name: tf-mnist
    image: microsoft/samples-tf-mnist-demo:gpu
    resources:
      requests:
        cpu: 0.5
        memory: 2Gi
      limits:
        cpu: 4.0
        memory: 16Gi
    nodeSelector:
      hardware: highmem
```

Node Selector



Large or Medium?
Not small

Node Affinity

- The notion of affinity was added to node selection via the affinity structure in the Pod spec. Affinity is a more complicated structure to understand, but it is significantly **more flexible if you want to express more complicated scheduling policies**.
- Types:
  - requiredDuringSchedulingIgnoredDuringExecution
  - preferredDuringSchedulingIgnoredDuringExecution
  - *requiredDuringSchedulingRequiredDuringExecution(planned)*

kubectl label node aks-nodepool1 hardware:highmem

```
kind: Pod
apiVersion: v1
metadata:
  name: tf-mnist
spec:
  containers:
  - name: tf-mnist
    image: microsoft/samples-tf-mnist-demo:gpu
    resources:
      requests:
        cpu: 0.5
        memory: 2Gi
      limits:
        cpu: 4.0
        memory: 16Gi
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: hardware
            operator: In
            values: highmem
```

Taints and Tolerations

Taints and Tolerations

- Scheduler can use taints and tolerations to restrict what workloads can run on nodes.
- A **taint is applied to a node** that indicates only specific pods can be scheduled on them.
- A **toleration is then applied to a pod** that allows them to tolerate a node's taint.

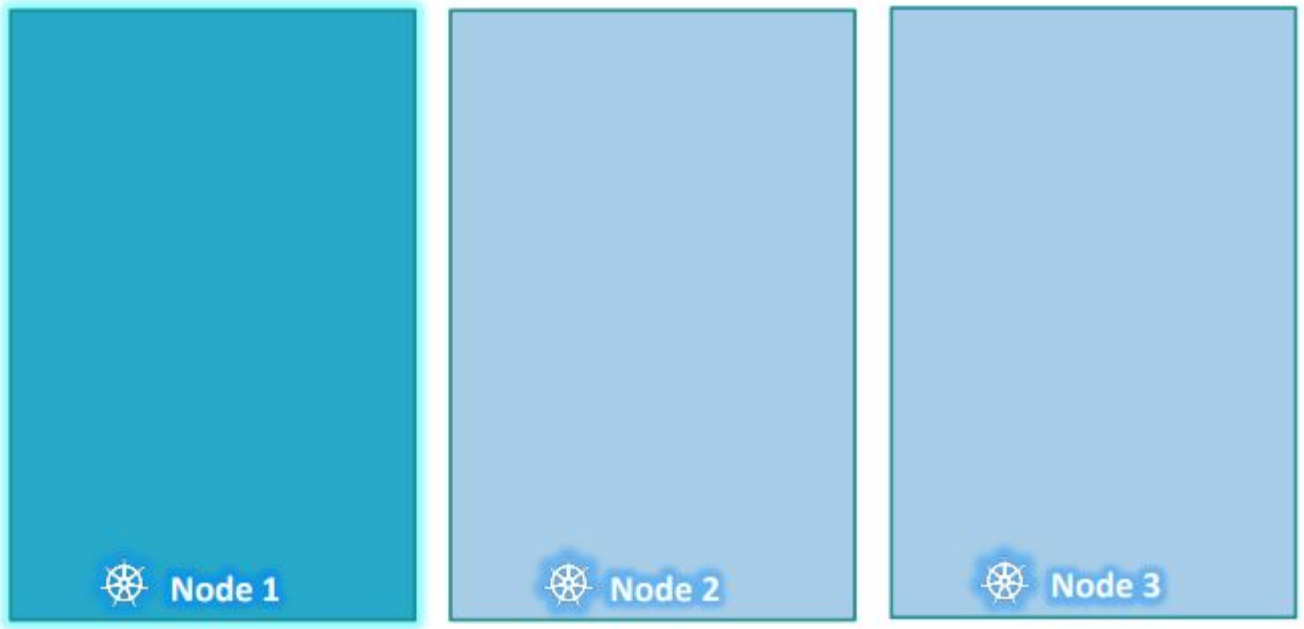
kubectl taint node aks-nodepool1 sku=gpu:NoSchedule

```
kind: Pod
apiVersion: v1
metadata:
  name: tf-mnist
spec:
  containers:
  - name: tf-mnist
    image: microsoft/samples-tf-mnist-demo:gpu
    resources:
      requests:
        cpu: 0.5
        memory: 2Gi
      limits:
        cpu: 4.0
        memory: 16Gi
  tolerations:
  - key: "sku"
    operator: "Equal"
    value: "gpu"
    effect: "NoSchedule"
```

```
kubectl describe node node01  | grep -i taint
```

Taints and Tolerations

Taint=blue



Muchas gracias

datahack
INSPIRING BIG DATA