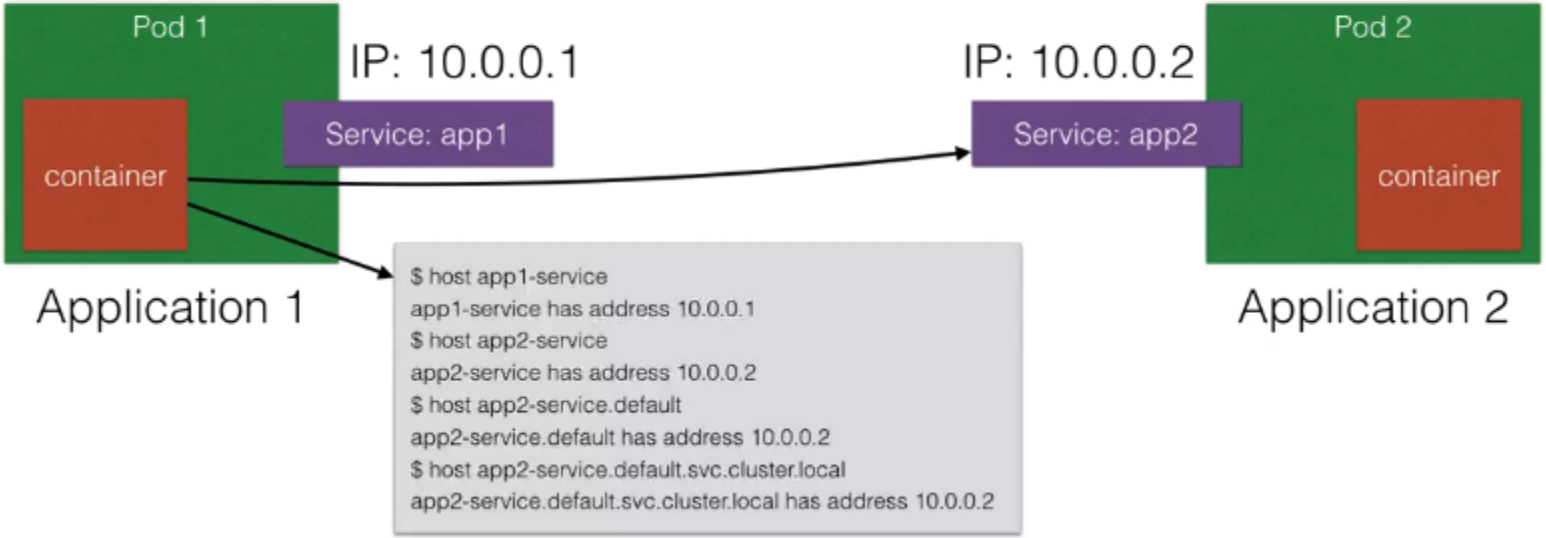

Service Discovery

DNS

- As of Kubernetes 1.10, DNS is a **built-in** service launched automatically using addon manger
- The DNS Service can be used within pods to find other service running on the same cluster
- Multiple containers **within 1 pod** don’t need this service, as they can contact each other **directly**
- A container in the same pod can **connect / communicate** to other containers directly using **localhost:port**

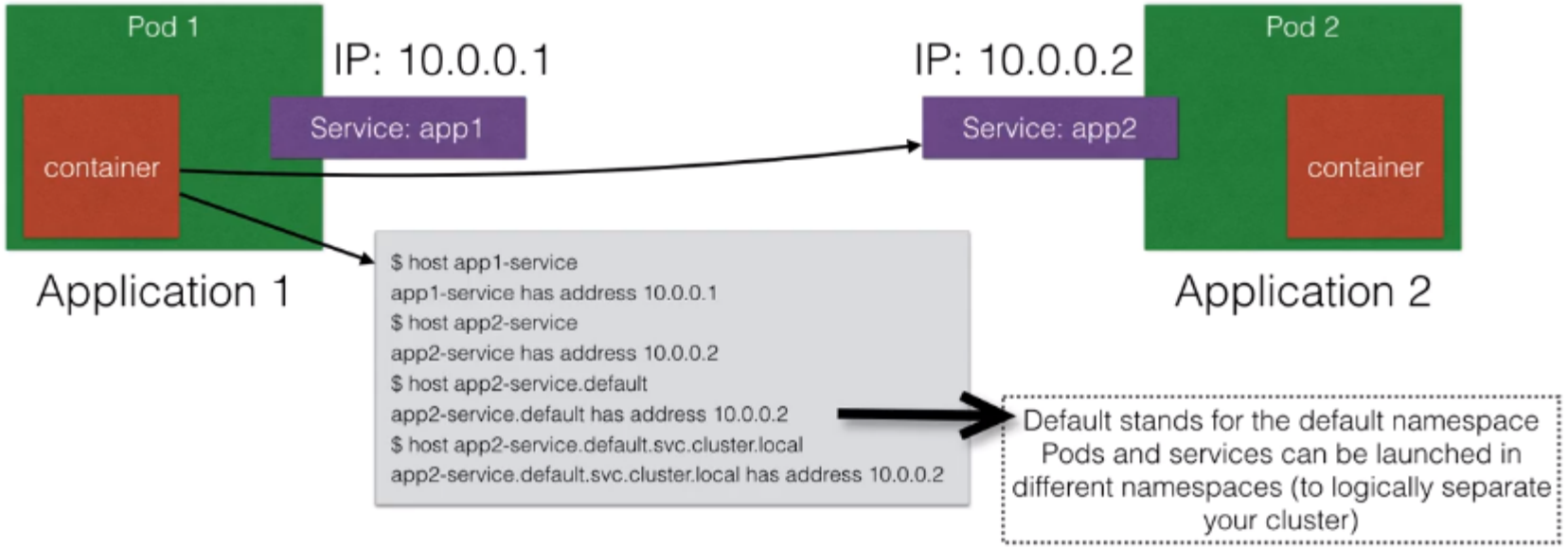
DNS

- An example of how app 1 could reach app 2 using DNS:

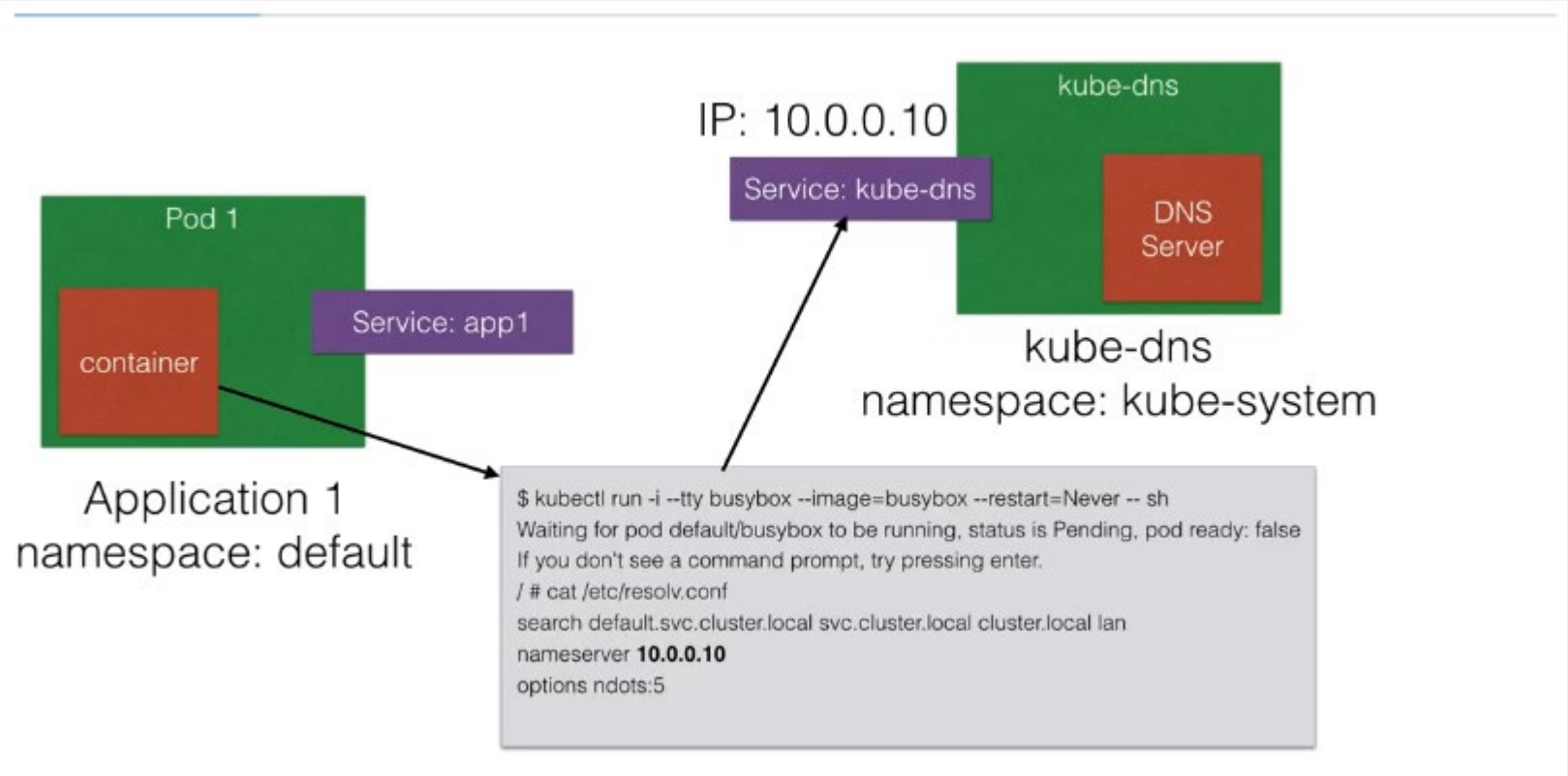


DNS

- An example of how app 1 could reach app 2 using DNS:



DNS – How does it work ?



Demo Placeholder

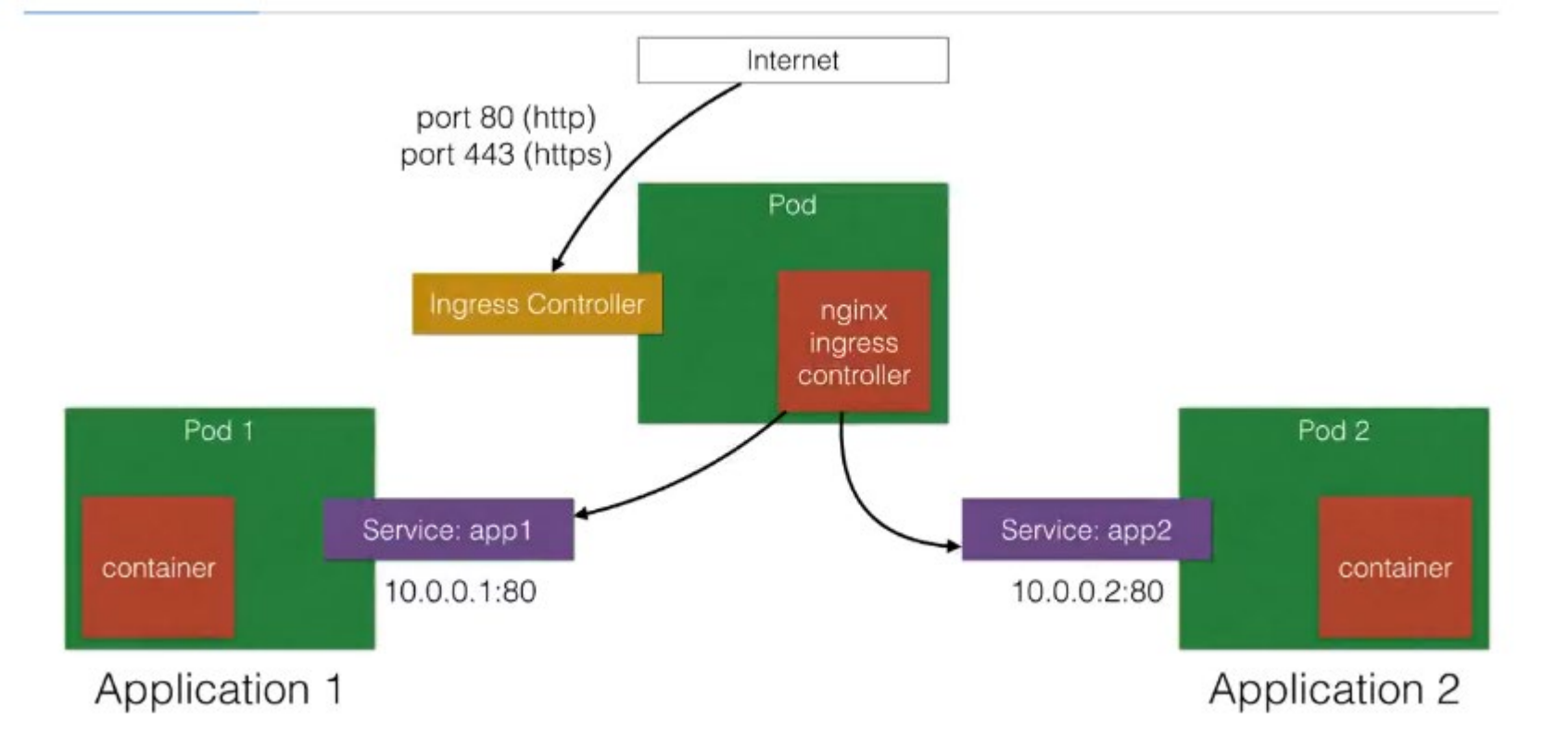
Service Discovery

Ingress

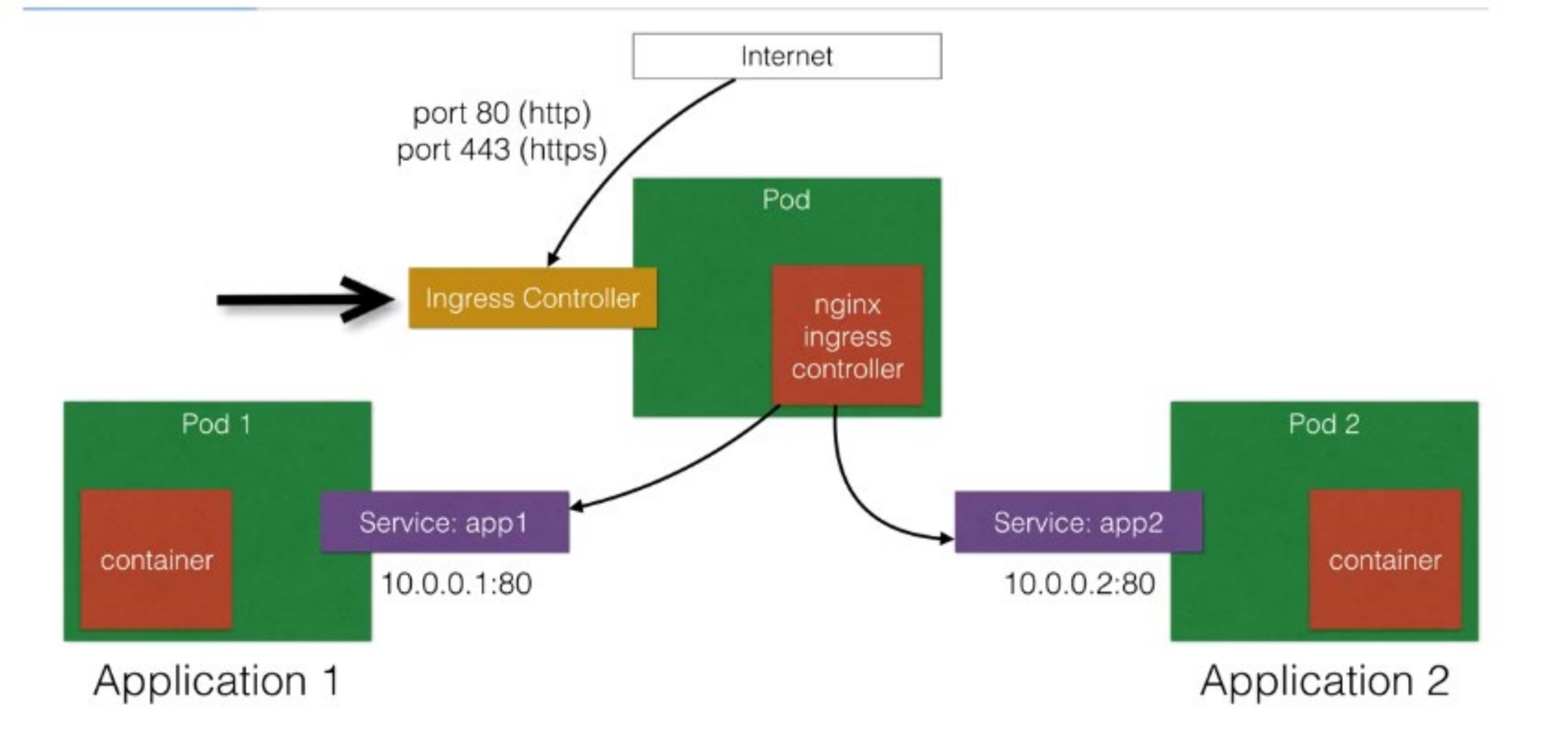
Ingress

- Ingress is a solution available since kubernetes 1.1 that allows **inbound connection** to the cluster
- **I**t's an alternative to the external LoadBalancer & NodePorts
  - Ingress allows you to **easily expose services** that need to be accessible from **outside** to the **cluster**
- With ingress you can run your own **ingress controller** (basically a LoadBalancer) within the kubernetes cluster
- There are a default ingress controllers available, or you can **write your own** ingress controller

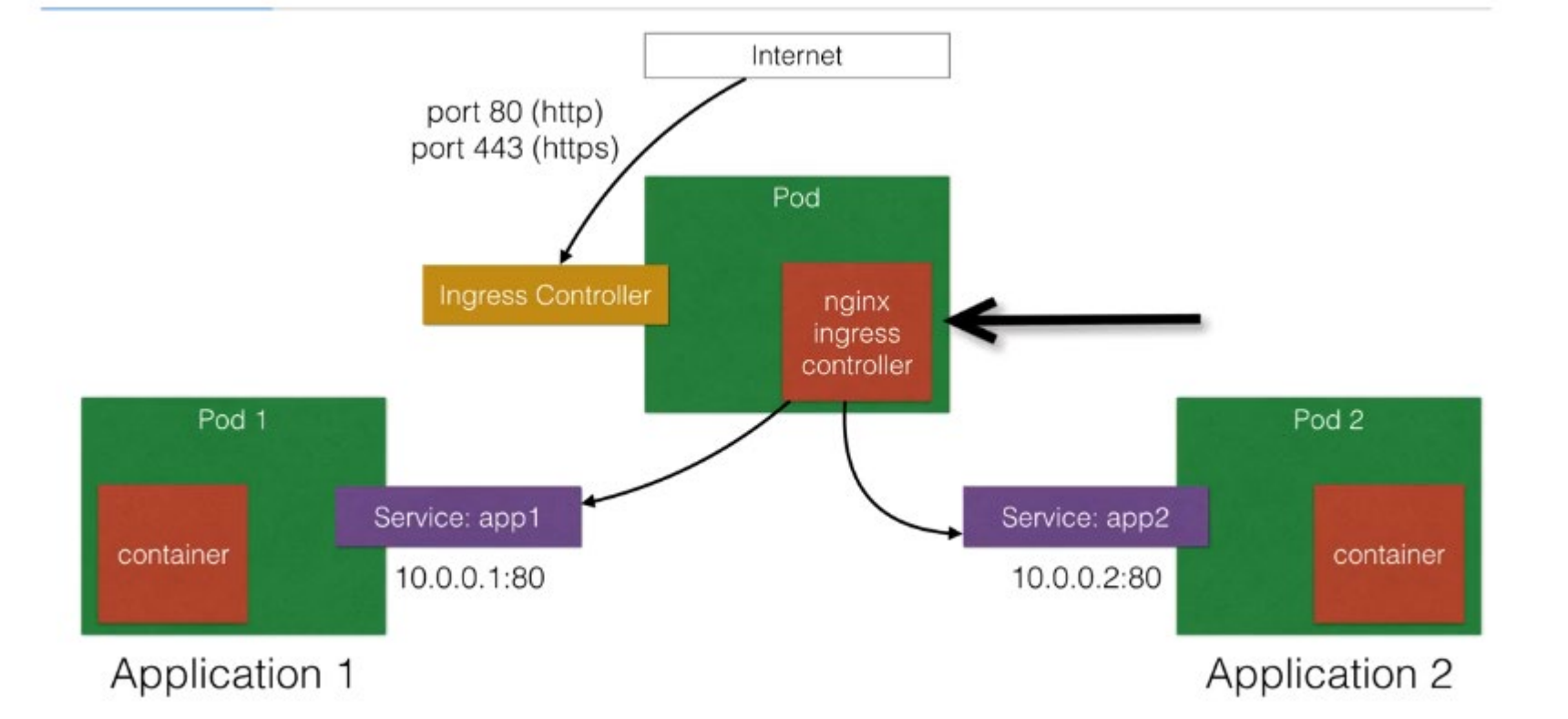
Ingress


Internet
port 80 (http)
port 443 (https)
Pod
Ingress Controller
nginx
ingress
controller
Pod 1
container
Service: app1
10.0.0.1:80
Pod 2
Service: app2
10.0.0.2:80
container
Application 1
Application 2

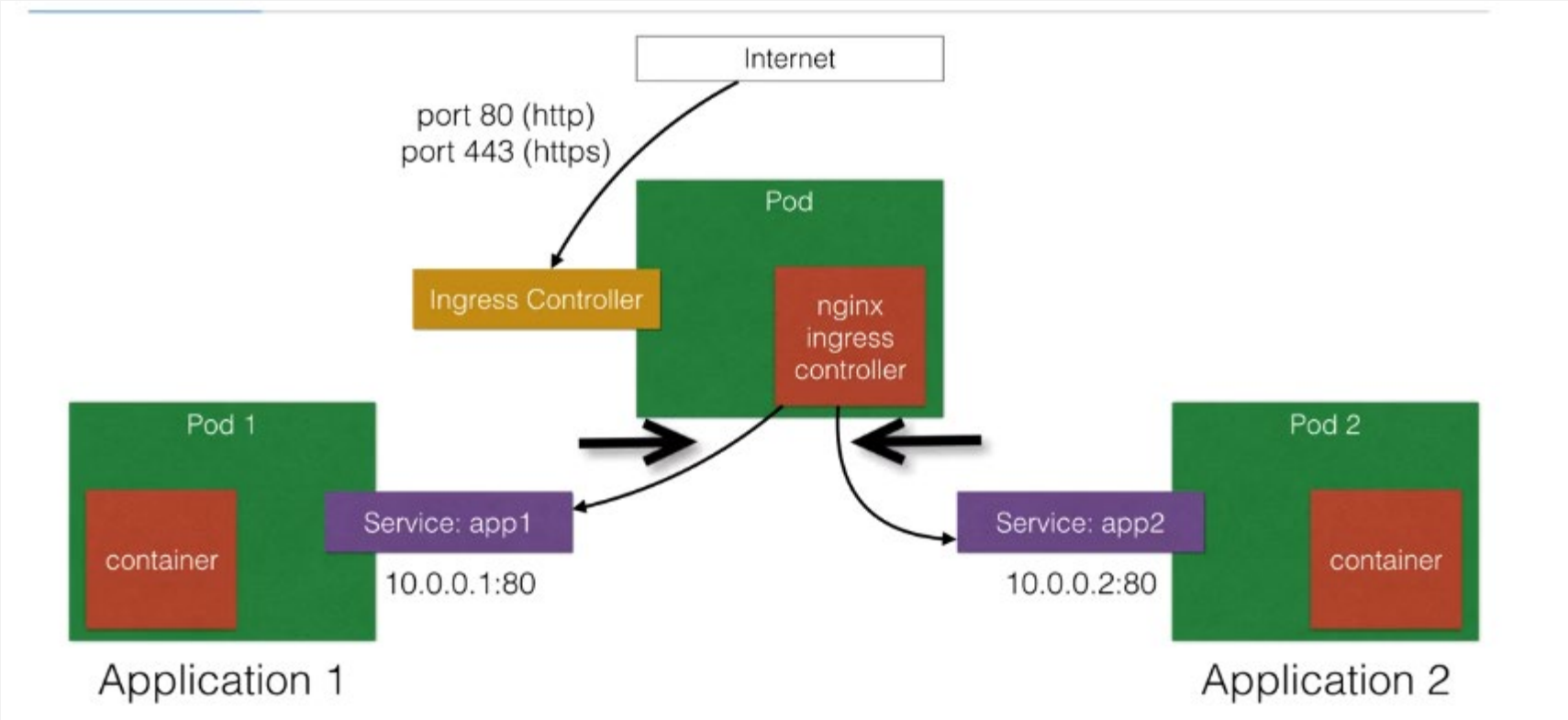
Ingress


Internet
port 80 (http)
port 443 (https)
Pod
Ingress Controller
nginx
ingress
controller
Pod 1
container
Service: app1
10.0.0.1:80
Pod 2
container
Service: app2
10.0.0.2:80
Application 1
Application 2

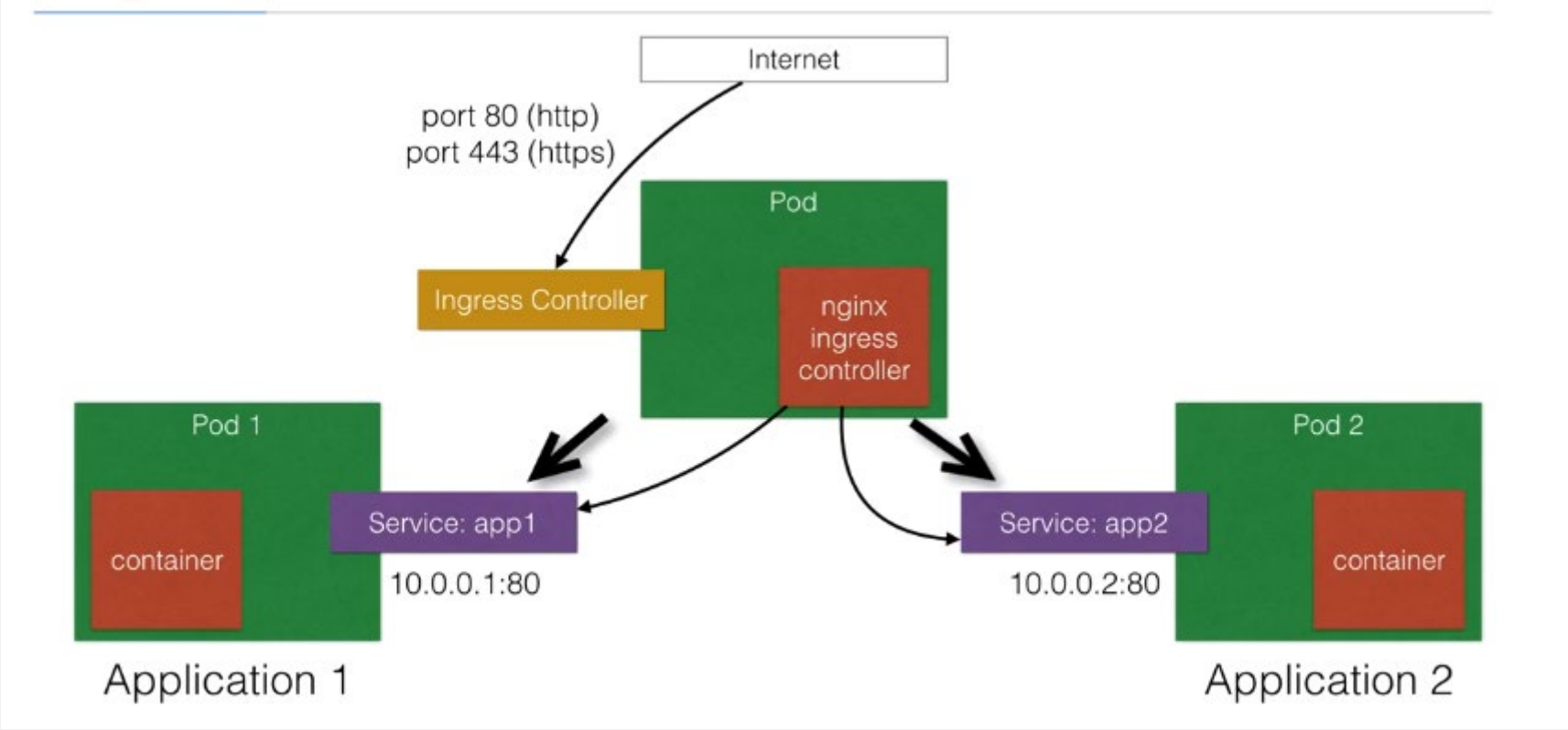
Ingress


Internet
port 80 (http)
port 443 (https)
Pod
Ingress Controller
nginx
ingress
controller
Pod 1
container
Service: app1
10.0.0.1:80
Pod 2
Service: app2
10.0.0.2:80
container
Application 1
Application 2

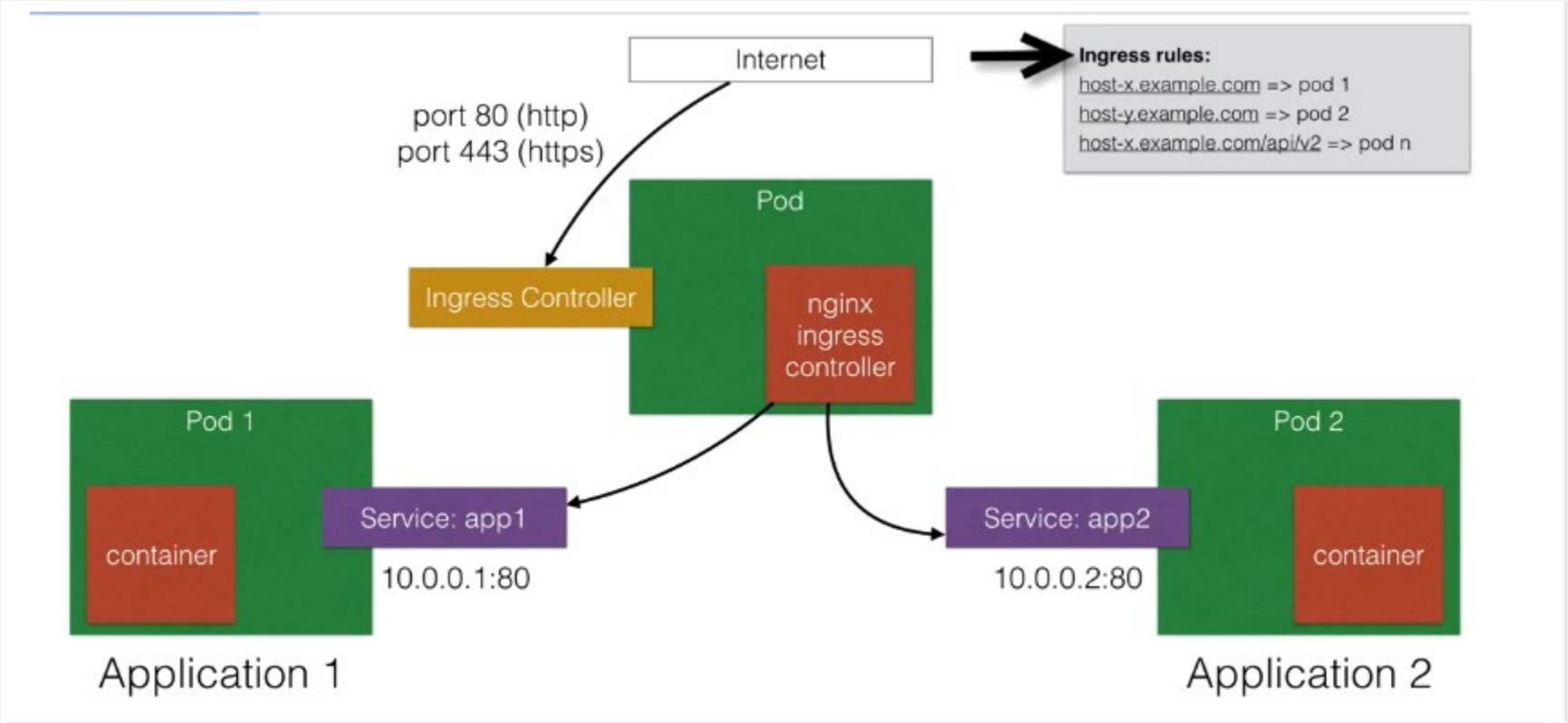
Ingress


Internet
port 80 (http)
port 443 (https)
Pod
Ingress Controller
nginx
ingress
controller
Pod 1
container
Service: app1
10.0.0.1:80
Pod 2
container
Service: app2
10.0.0.2:80
Application 1
Application 2

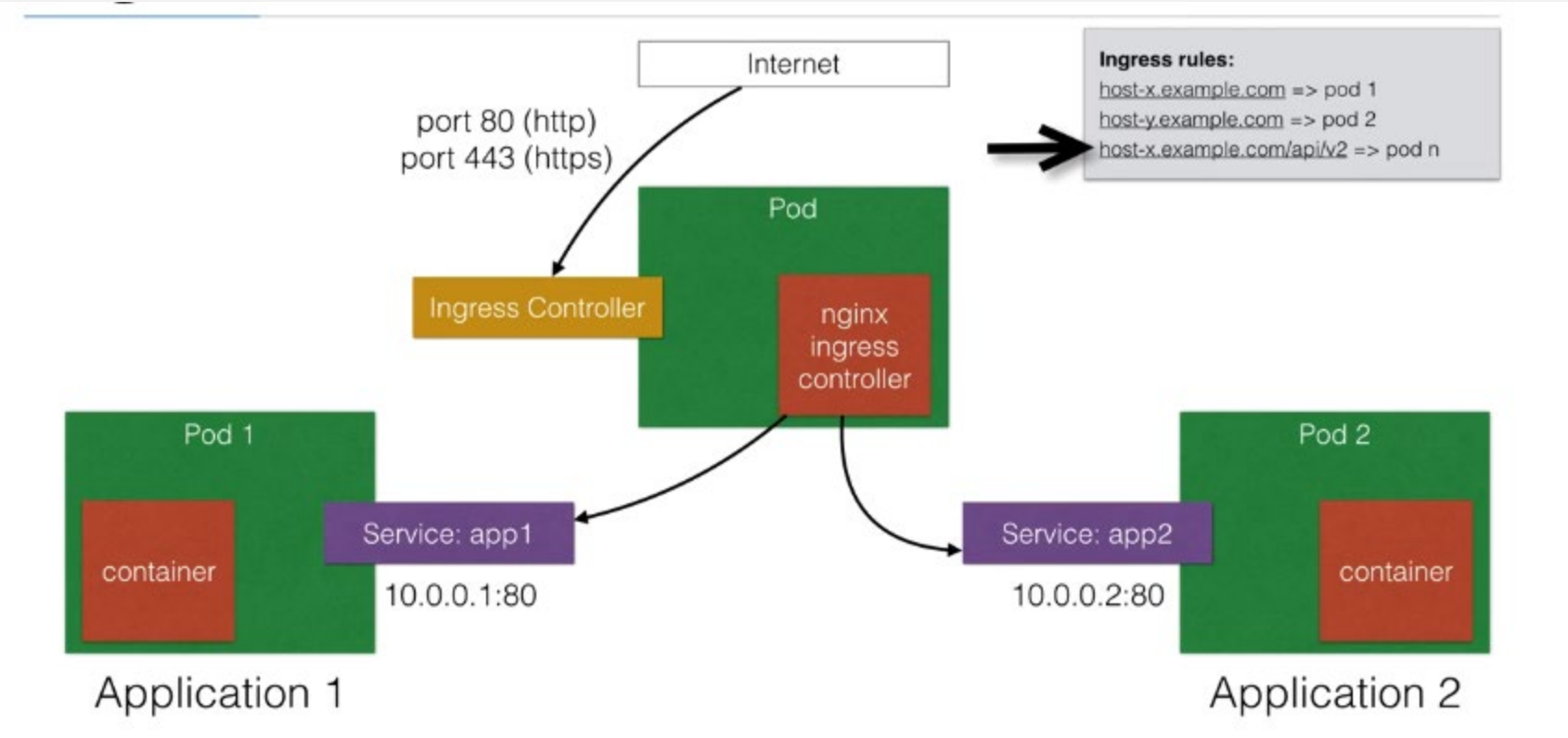
Ingress


Internet
port 80 (http)
port 443 (https)
Pod
Ingress Controller
nginx
ingress
controller
Pod 1
container
Service: app1
10.0.0.1:80
Service: app2
10.0.0.2:80
Pod 2
container
Application 1
Application 2

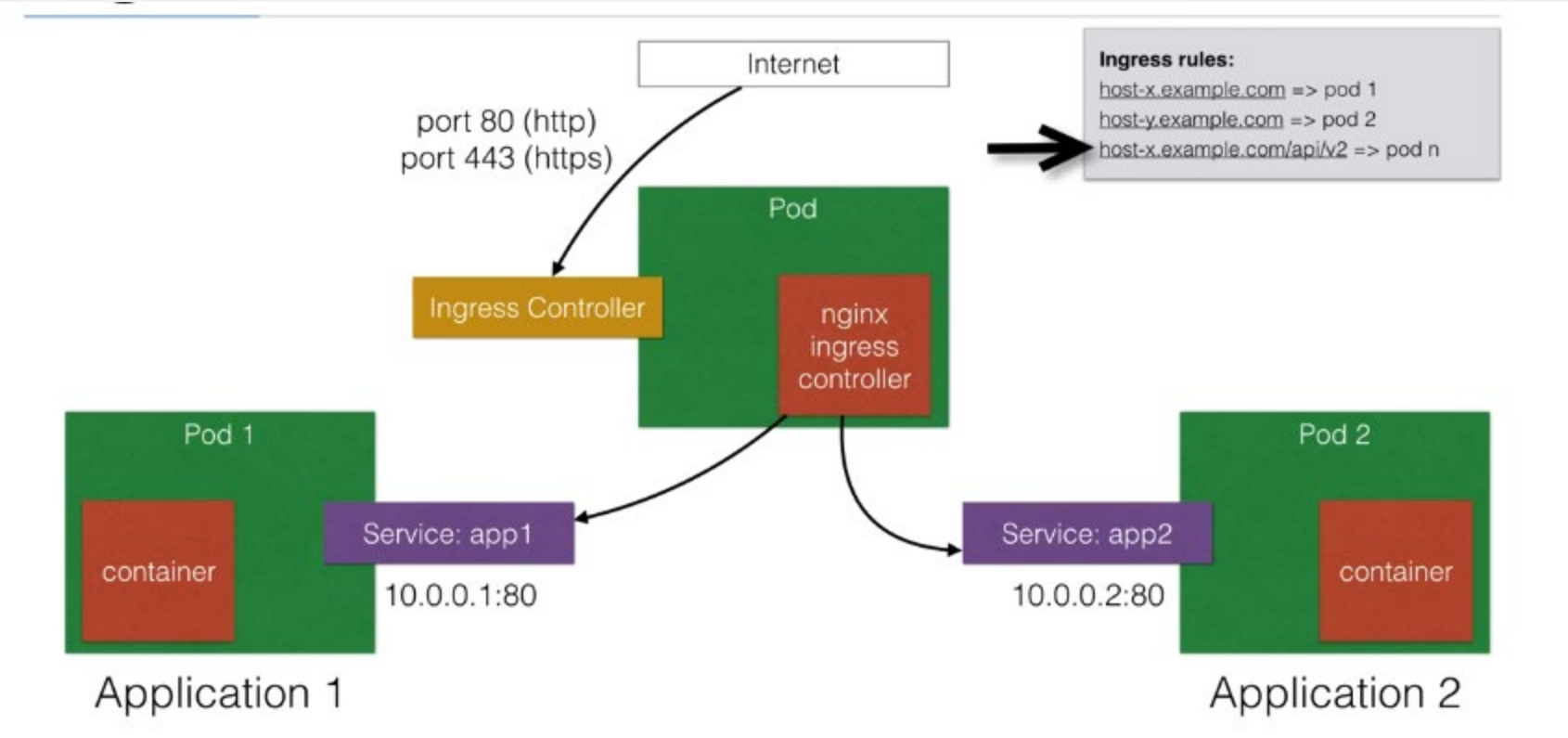
Ingress


Internet
Ingress rules:
host-x.example.com => pod 1
host-y.example.com => pod 2
host-x.example.com/api/v2 => pod n
port 80 (http)
port 443 (https)
Pod
Ingress Controller
nginx ingress controller
Pod 1
container
Service: app1
10.0.0.1:80
Pod 2
container
Service: app2
10.0.0.2:80
Application 1
Application 2

Ingress

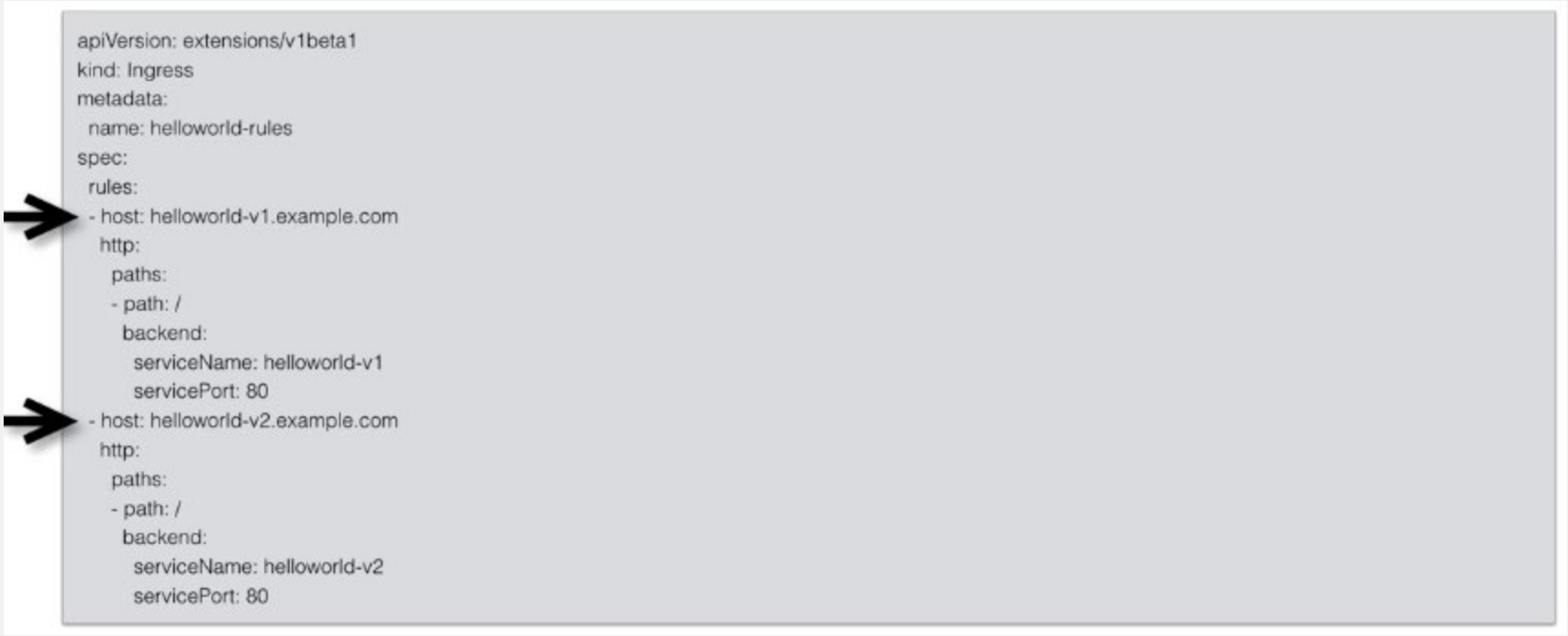

Internet
port 80 (http)
port 443 (https)
Ingress rules:
host-x.example.com => pod 1
host-y.example.com => pod 2
host-x.example.com/api/v2 => pod n
Pod
Ingress Controller
nginx ingress controller
Pod 1
container
Service: app1
10.0.0.1:80
Application 1
Service: app2
10.0.0.2:80
Pod 2
container
Application 2

Ingress


Internet
port 80 (http)
port 443 (https)
Ingress rules:
host-x.example.com => pod 1
host-y.example.com => pod 2
host-x.example.com/api/v2 => pod n
Pod
Ingress Controller
nginx ingress controller
Pod 1
container
Service: app1
10.0.0.1:80
Application 1
Service: app2
10.0.0.2:80
Pod 2
container
Application 2

Ingress

- You can create ingress rules using the ingress objects



```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: helloworld-rules
spec:
  rules:
  - host: helloworld-v1.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: helloworld-v1
          servicePort: 80
  - host: helloworld-v2.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: helloworld-v2
          servicePort: 80
```

Demo Placeholder

Ingress Controller

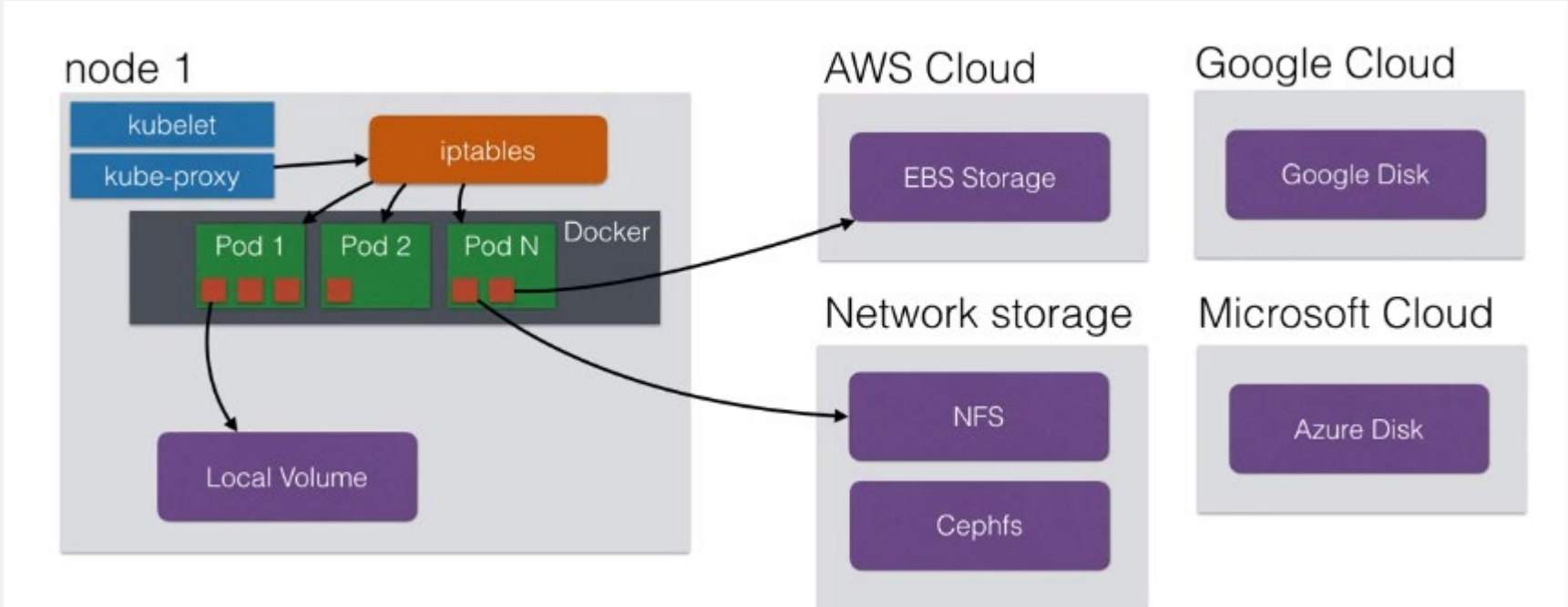

Volumes

Volumes

- Volumes in kubernetes allow you to **store data outside the container**
- When a containers **stops,** all data on the container itself is **lost**
  - That's why up until now I've been using **stateless** apps: apps that don't keep a **local** state, but store theirs state in an **external service**
    - External Service like a database, cashing server (e.g. MySQL, AWS S3)
- Persistent Volumes in Kubernetes allow you **attach a volume** to a container that will **exists** even when the **container** stops

Kubernetes Volumes

- Volumes can be attached using different volumes plugins:

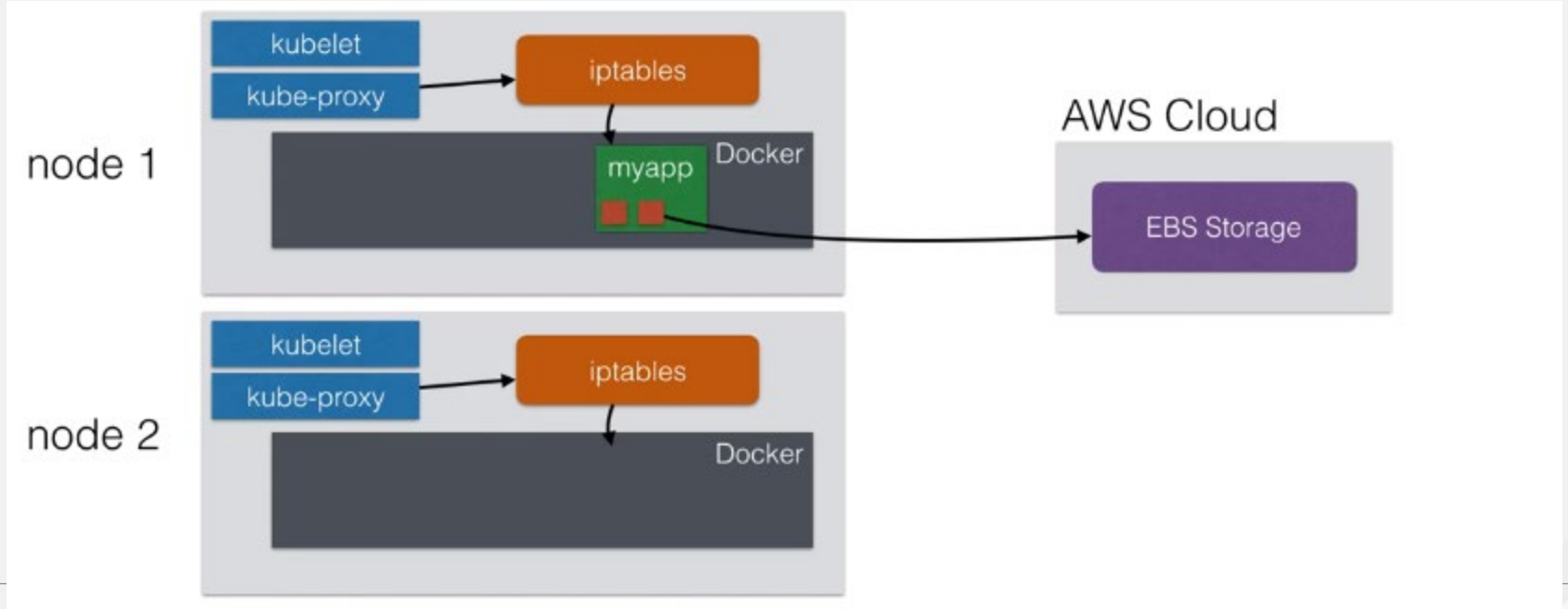


Volumes

- Using volumes, you could deploy **applications with state** on your cluster
  - Those applications need to read/write to files on the **local filesystem** that need to be persistent in time
- You could run a MySQL database using persistent volumes
  - Although this might not be ready for production (yet)
  - Volumes are new since the June 2016 release in Kubernetes

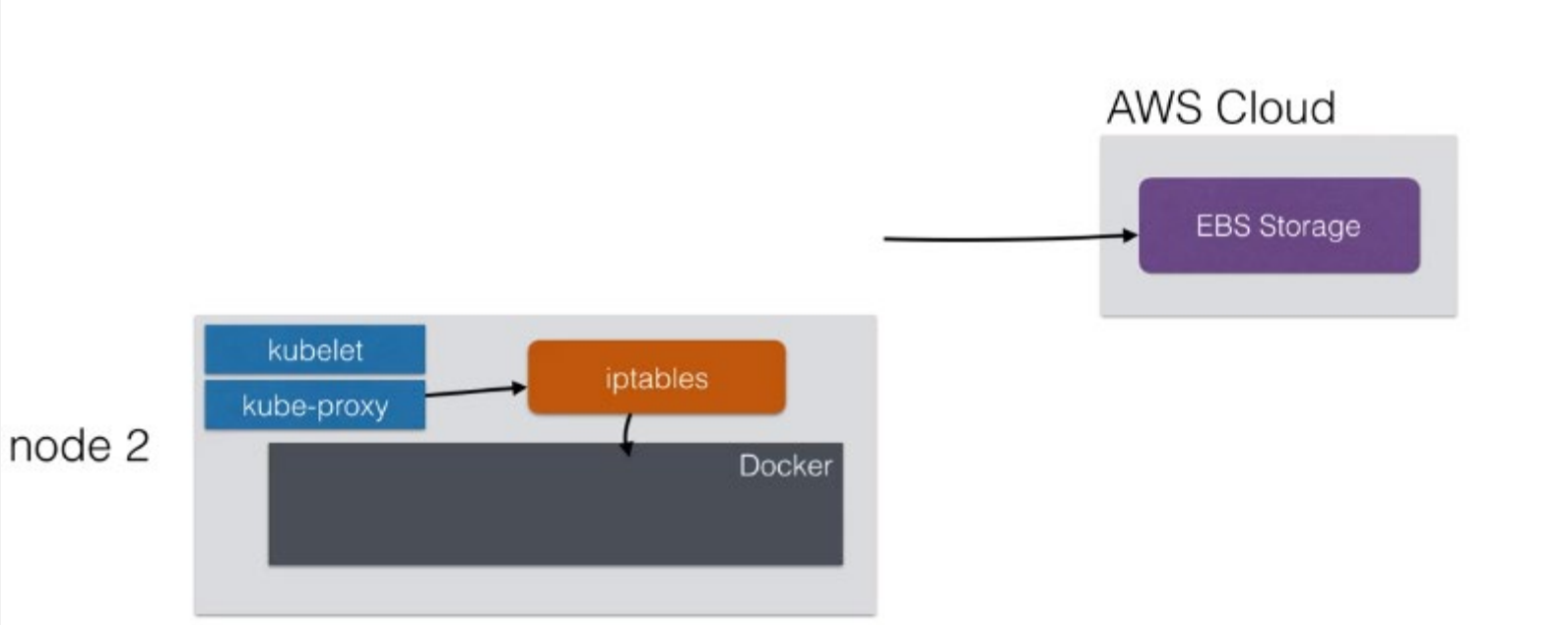
Volumes

- If your **node stops** working, the pod can be rescheduled on another node, and the volumes can be attached to new node



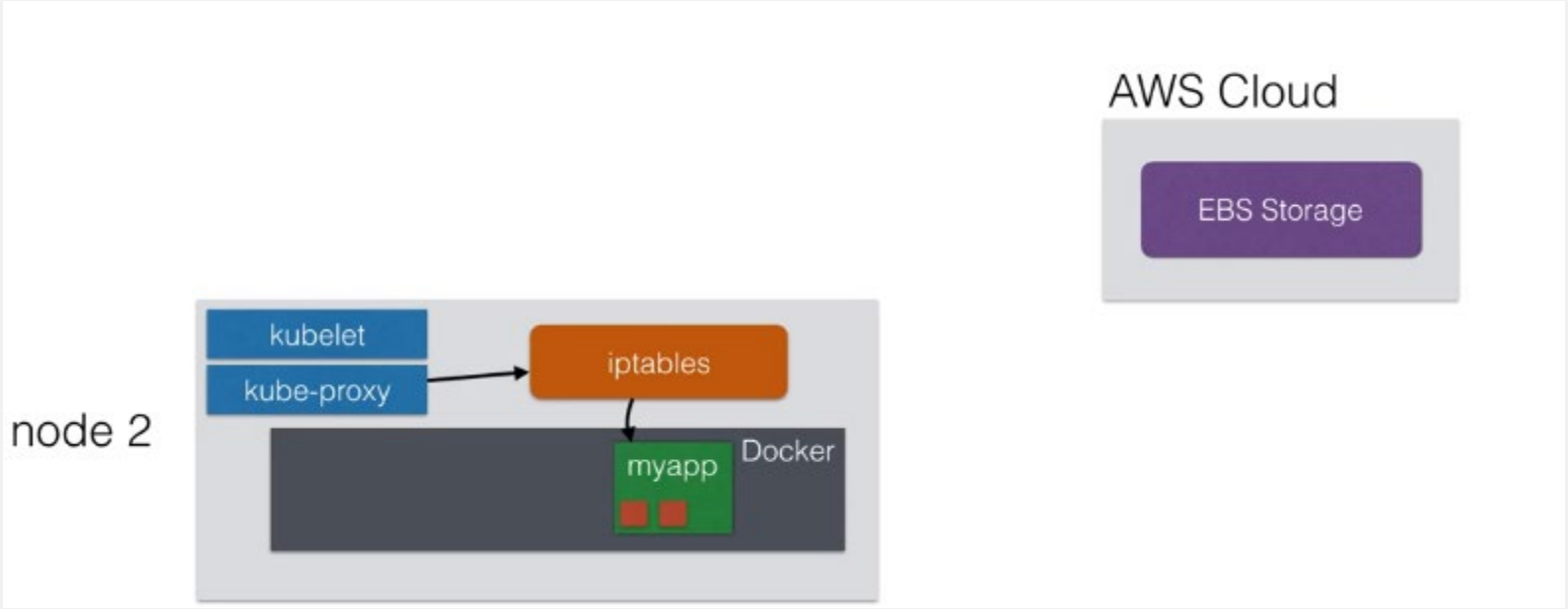
Volumes

- If your **node stops** working, the pod can be rescheduled on another node, and the volumes can be attached to new node



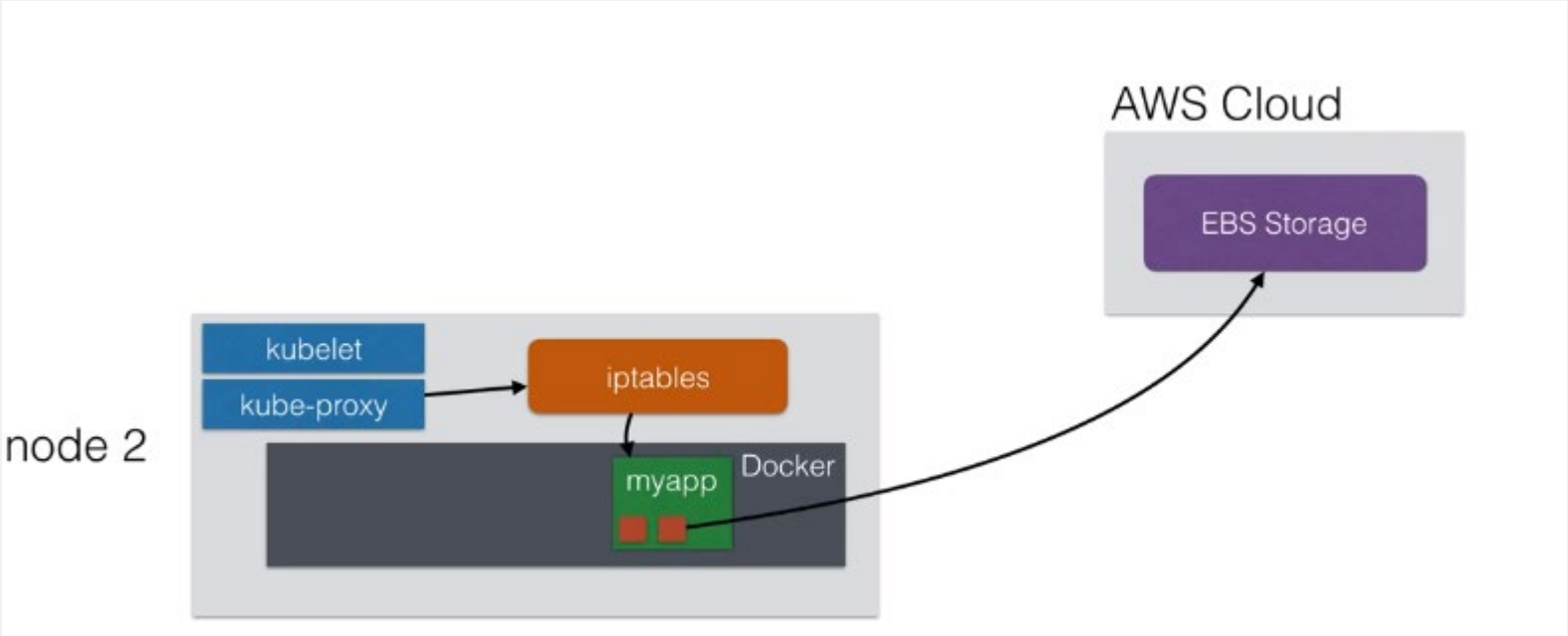
Volumes

- If your **node stops** working, the pod can be rescheduled on another node, and the volumes can be attached to new node



Volumes

- If your **node stops** working, the pod can be rescheduled on another node, and the volumes can be attached to new node




Pod State
redhat.

Pod State

- Pods have a status field, which you see when you do *kubectl get pods*:

```
$ kubectl get pods -n kube-system
NAME                                                        READY     STATUS    RESTARTS   AGE
dns-controller-7cc97fb976-4b9nt                             1/1       Running   0          4h
etcd-server-events-ip-172-20-38-169.eu-west-1.compute.internal   1/1       Running   0          4h
etcd-server-ip-172-20-38-169.eu-west-1.compute.internal     1/1       Running   0          4h
kube-apiserver-ip-172-20-38-169.eu-west-1.compute.internal  1/1       Running   0          4h
```

- In this scenario all pods are in the **running status**
  - This means that the **pod has been bound to a node**
  - All **containers have been created**
  - **At least one container** is still **running**, or is starting/restarting

Pod State

- Other valid statuses are:
  - **Pending**: Pod has been **accepted** but is **not running**
    - Happens when the container image is still **downloading**
    - If the pod cannot be scheduled because of **resource constraints**, it'll also be in this status
  - **Succeeded**: All containers within this pod have been **terminated successfully** and will not be restarted

Pod State

- Other valid statuses are:
  - **Failed:** All containers within this pod have been **Terminated**, and at least one container returned a failure code
    - The failure code is the **exit code** of the process when a container terminates
  - **Unknown:** The **state of the pod couldn't be determined**
    - A **network error** might have been occurred (for example the node where the pod is running on is down)

Pod State

- You can get the pod conditions using kubectl describe pod PODNAME

```
$ kubectl describe pod kube-apiserver-ip-172-20-38-169.eu-west-1.compute.internal -n kube-system
[…]
Conditions:
  Type           Status
  Initialized    True
  Ready          True
  PodScheduled   True
```

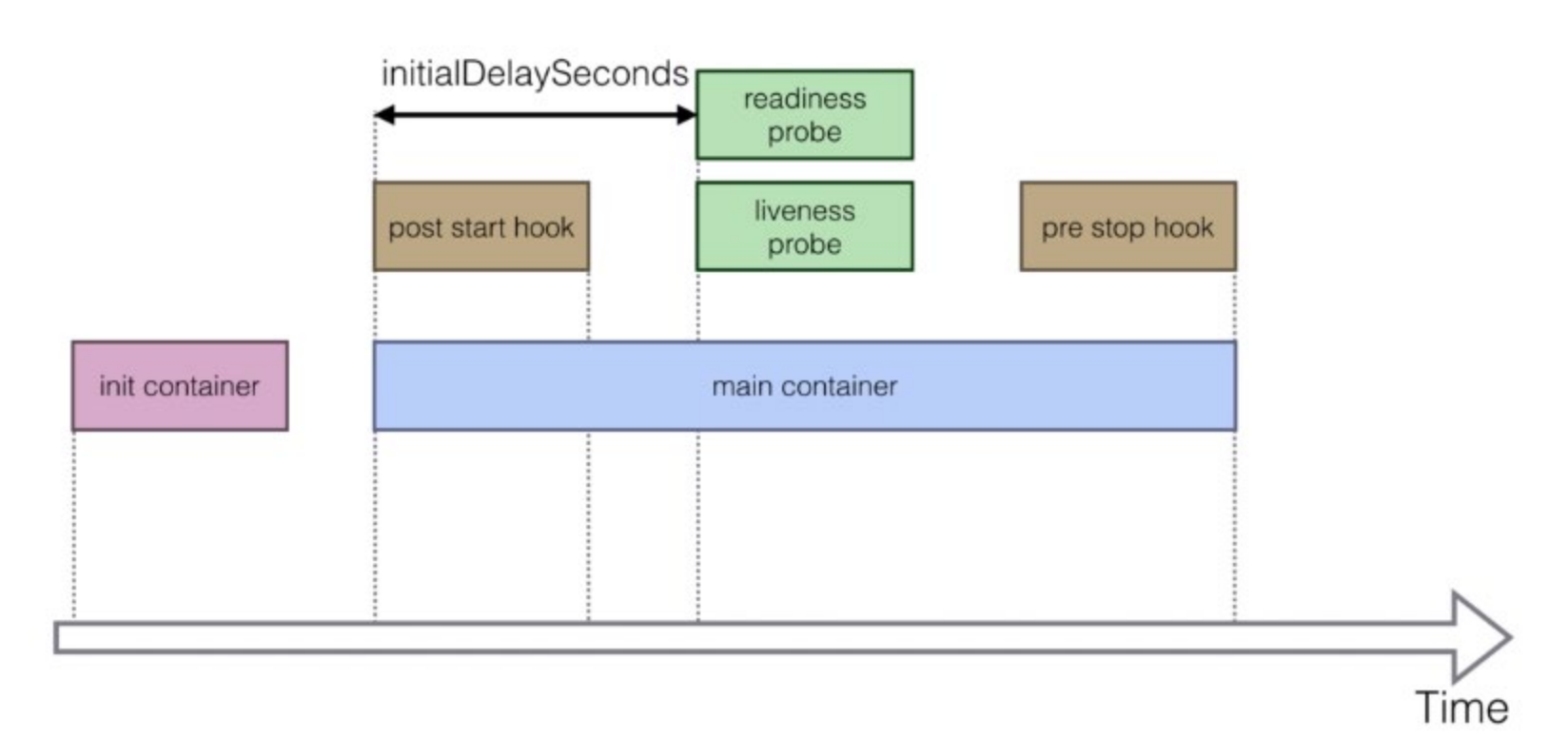
- These are conditions which the pod has passed
  - In this example, Initialized, Ready, and PodScheduled

Pod State

- There are 5 different types of PodConditions:
  - **PodScheduled**: the pod has been scheduled to a node
  - **Ready**: Pod can serve requests and is going to be added to matching Services
  - **Initialized**: the initialization containers have been started successfully
  - **Unschedulable**: the Pod can't be scheduled (for example due to resource constraints)
  - **ContainersReady**: all containers in the pod are ready


Pod LifeCycle
redhat.

Pod Lifecycle



Demo


StatefulSets
redhat.

Stateful Sets

- Pet Sets was a **new feature** starting from Kubernetes 1.3, and got renamed to StatefulSets which is stable since Kubernetes 1.9
- It is introduced to be able to run **stateful applications**:
  - That need a **stable pod hostname** (instead of podname-randomstring)
    - Your podname will have a sticky identity, using an index, e.g. podname-0 podname-1 and podname-2 (and when a pod gets rescheduled, it'll keep that identity)
  - Statefulsets allow **stateful apps stable storage** with volumes based on their ordinal number (podname-**x**)
    - **Deleting** and/or **scaling** a **StatefulSet down** will not delete the volumes associated with the StatefulSet (preserving data)

Stateful Sets

- A StatefulSet will allow your stateful app to use **DNS** to find other **peers**
  - Cassandra clusters, ElasticSearch clusters, use **DNS** to find other members of the cluster
    - for example: **cassandra-0.cassandra** for all pods to reach the first node in the cassandra cluster
  - Using StatefulSet you can run for instance 3 cassandra nodes on Kubernetes named cassandra-0 until cassandra-2
  - If you wouldn't use StatefulSet, you would get a dynamic hostname, which you wouldn't be able to use in your configuration files, as the name can always change

Stateful Sets

- A StatefulSet will also allow your stateful app to **order the startup and teardown**:
  - Instead of randomly terminating one pod (one instance of your app), you'll know which one that will go
    - When **scaling up** it goes from 0 to n-1 (n = replication factor)
    - When **scaling down** it starts with the highest number (n-1) to 0
  - This is useful if you first need to **drain** the data from a node before it can be shut down

Stateful Sets

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: helloworld-statefull
  labels:
    app: helloworld-statefull
spec:
  serviceName: helloworld-statefull
  replicas: 3
  selector:
    matchLabels:
      app: helloworld-statefull
  template:
    metadata:
      labels:
        app: helloworld-statefull
    spec:
      containers:
      - name: k8s-demo
        image: amitvashist7/k8s-tiny-web
        ports:
        - name: node-port
          containerPort: 80
```


Deamon Sets
redhat.

Daemon Sets

- Daemon Sets ensure that **every single node** in the Kubernetes cluster runs the same pod resource
    - This is useful if you want to **ensure** that a certain pod is running on every single kubernetes node
- When a node is **added** to the cluster, a new pod will be **started** automatically
- Same when a node is **removed**, the pod will not be **rescheduled** on another node


Packaging and Deploying
on Kubernetes
redhat.


Helm
redhat.

Helm

- Helm the best way to find, share and use software built for Kubernetes
  (definition from https://helm.sh/)
- Helm is a **package manager** for Kubernetes
- It helps you to manage Kubernetes **applications**
- Helm is maintained by the **CNCF - The Cloud Native Computing Foundation** (together with Kubernetes, fluentd, linkerd, and others)
  - It is now maintained in collaboration with **Microsoft**, **Google**, **Bitnami** and the **helm contributor community**

Helm

- To start using helm, you first need to download the **helm client**
- You need to run “helm init” to **initialize helm** on the Kubernetes cluster
  - This will install **Tiller**
  - If you have **RBAC installed** (recent clusters have it enabled now by default), you’ll also need add a **ServiceAccount and RBAC rules**
- After this, helm is ready for use, and you can **start installing charts**

Helm

- Helm uses a packaging format called **charts**
  - A **chart** is a **collection** of **files** that **describe** a set of Kubernetes **resources**
  - A single chart can **deploy an app**, a piece of software, or a database
  - It can have dependencies, e.g. to install wordpress chart, you need a mysql chart
  - You can write **your own chart** to deploy your application on Kubernetes using helm

Helm

- Charts use **templates** that are typically developed by a package maintainer
- They will generate **yaml** files that Kubernetes understands
- You can think of templates as dynamic yaml files, which can contain logic and variables

Helm

- This is an example of a **template within a chart**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  drink: {{ .Values.favoriteDrink }}
```

- The favoriteDrink value can then be overridden by the user when running helm install

Helm – Common Commands

Command	Description
helm init helm reset	Install tiller on the cluster Remove tiller from the cluster
helm install	Install a helm chart
helm search	search for a chart
helm list	list releases (installed charts)
helm upgrade	upgrade a release
helm rollback	rollback a release to the previous version

Demo Placeholder

Let’s set helm.


Helm
Create your own charts
redhat.

Helm Charts

- You can **create helm charts** to **deploy your own apps**
- It's the **recommended** way to deploy your applications on Kubernetes
  - Packaging the app, allows you **deploy the app in 1 command** (instead of using kubectl create / apply)
  - Helm allows for **upgrades** and **rollbacks**
  - Your helm chart is **version controlled**

Helm Charts

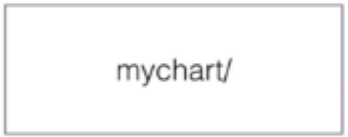
- To create the files necessary for a new chart, you can enter the command:

  helm create *mychart*

Helm Charts

- To create the files necessary for a new chart, you can enter the command:

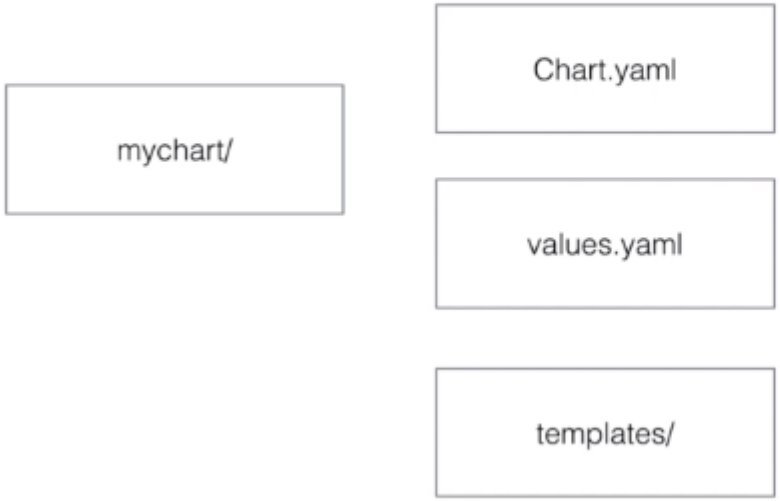
  helm create *mychart*



Helm Charts

- To create the files necessary for a new chart, you can enter the command:

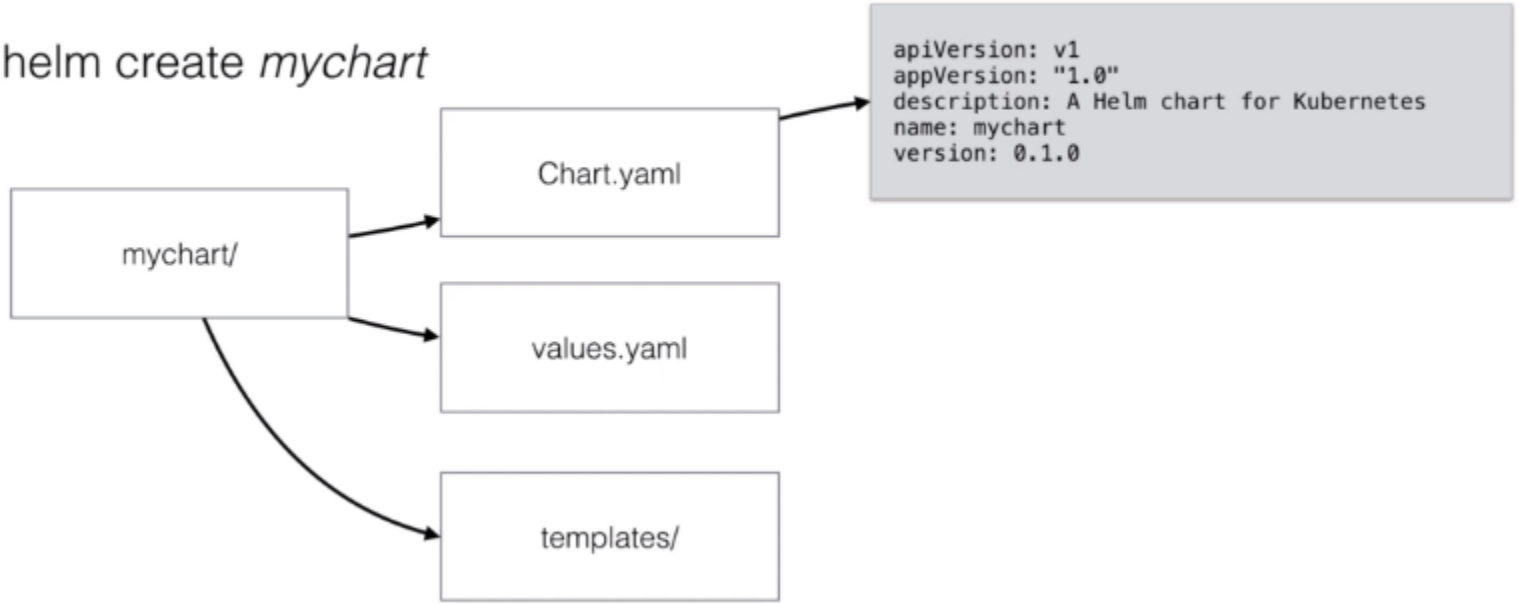
  helm create *mychart*



Helm Charts

- To create the files necessary for a new chart, you can enter the command:

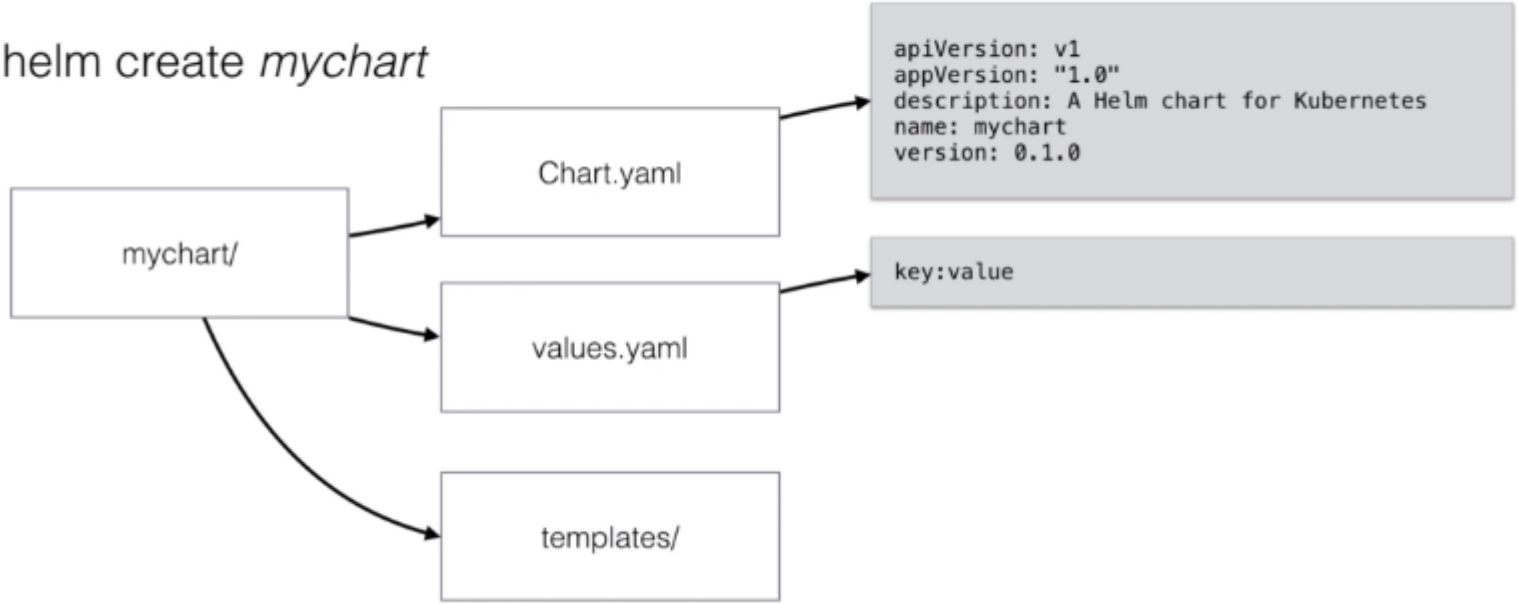
helm create *mychart*



Helm Charts

- To create the files necessary for a new chart, you can enter the command:

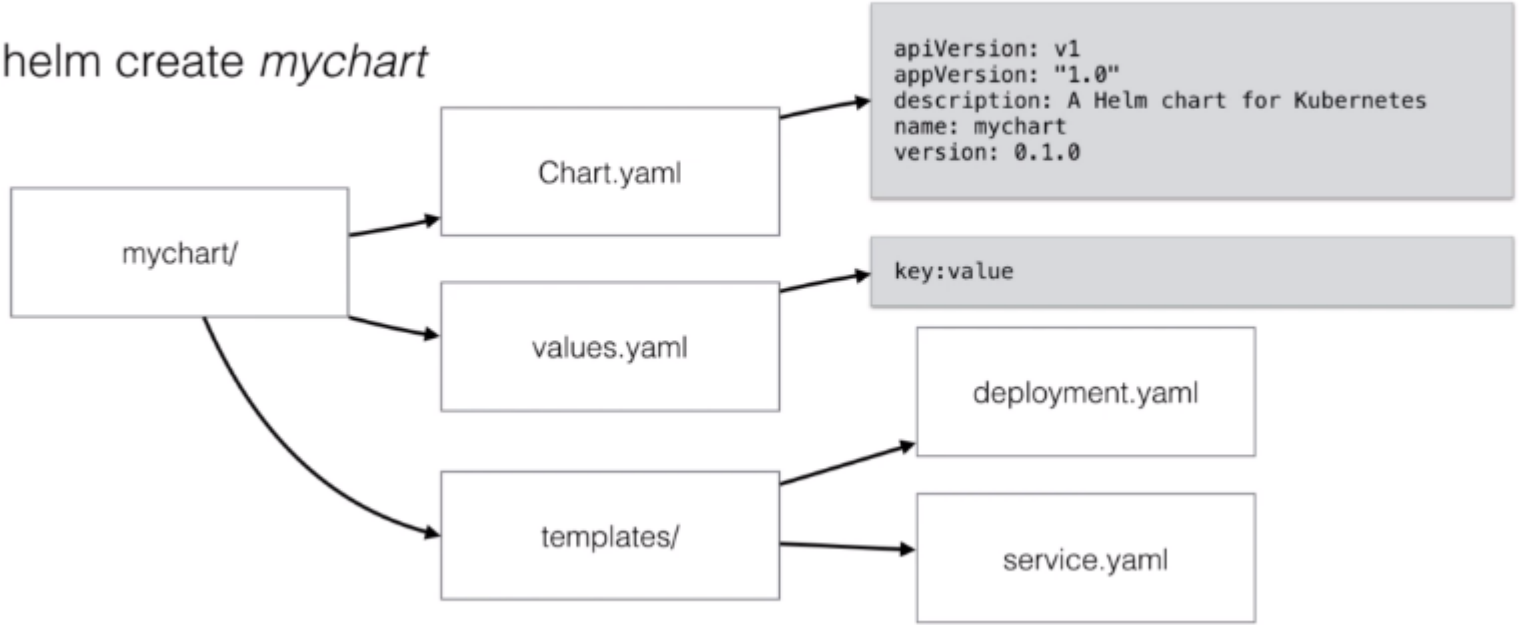
helm create *mychart*



Helm Charts

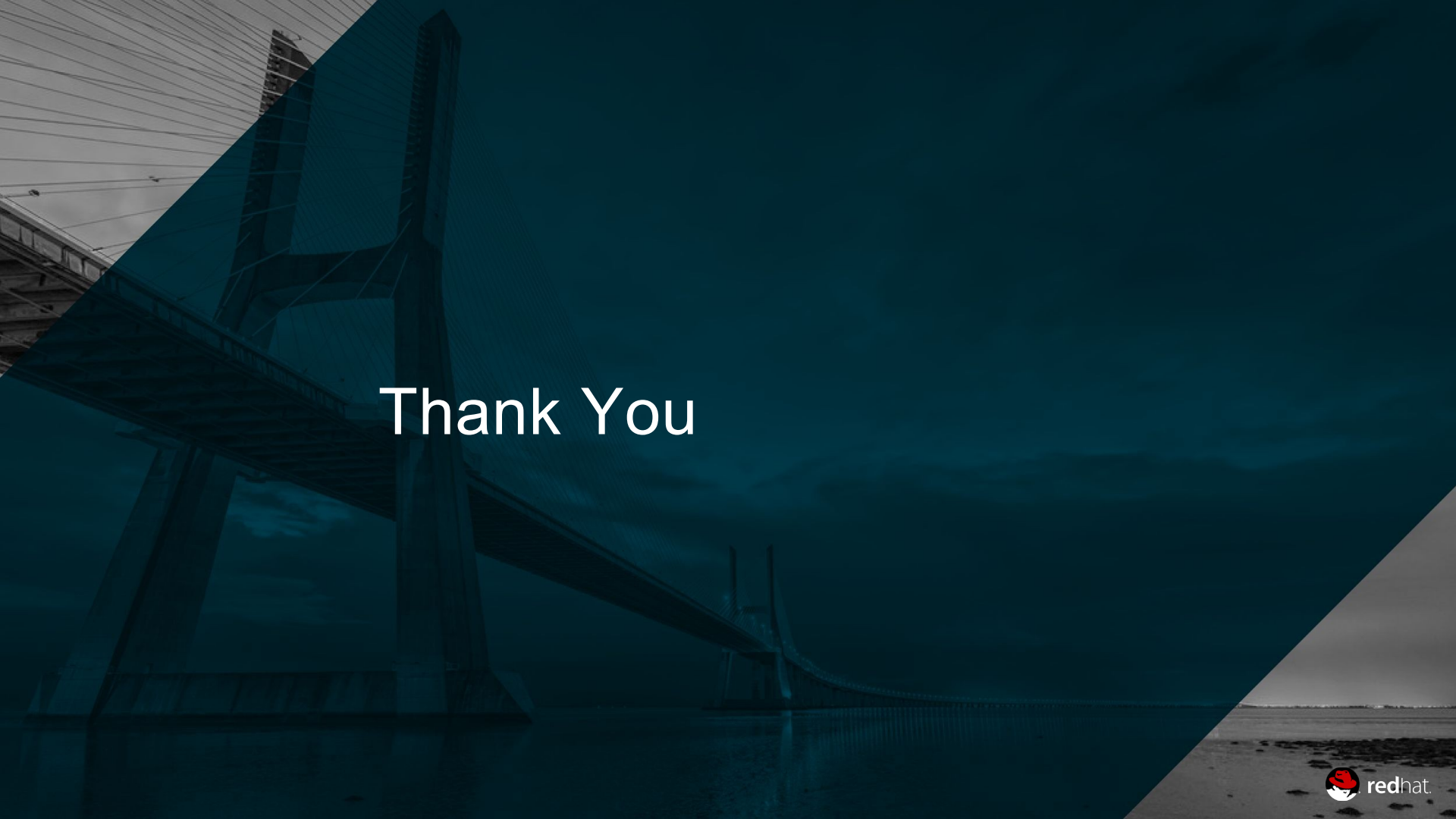
- To create the files necessary for a new chart, you can enter the command:

helm create *mychart*



Demo Placeholder

Let’s create our own Charts


Thank You
redhat.


Serverless on Kubernetes
redhat.

What is Serverless

- Public Cloud providers often provide Serverless capabilities in which you can **deploy functions**, rather than instances or containers
  - Azure Functions
  - AWS Lambda
  - Google Cloud Functions
- With these products, you **don't need to manage the underlying infrastructure**
- The functions are also **not "always-on"** unlike containers and instances, which can greatly the reduce cost of serverless if the function doesn't need to be executed a lot

What is Serverless

- Serverless in public cloud can **reduce the complexity, operational costs, and engineering time to get code running**
  - You don't need to manage a Windows/Linux distribution
  - You don't need to build containers
  - You only pay for the time your function is running
  - A developer can "just push" the code and does not to worry about many operational aspects
    - Although "cold-starts", the time for a function to start after it has not been invoked for some time, can be an operational issue that needs to be taken care of

What is Serverless

- This is an example of a AWS Lambda Function:

```
exports.handler = function(event, context) {
 context.succeed("Hello, World!");
};
```

- You'd still need to setup when the code is being executed
- For example in AWS you would use the API Gateway, to setup a URL that will invoke this function when visited

Serverless in Kubernetes

- Rather than using containers to start applications on Kubernetes, you can also use **Functions**
- Currently, the **most popular projects** enabling functions are:
  - OpenFaas
  - Kubeless
  - Fission
  - OpenWhisk
- You can install and use any of the projects to let developers launch functions on your Kubernetes cluster
- As an **administrator**, you'll **still need to manage the underlying infrastructure**, but from a **developer** standpoint, he/she will be able to **quickly and easily deploy functions on Kubernetes**


Kubeless
redhat.

Kubeless

- Kubeless is a Kubernetes-native framework (source: https://github.com/kubeless/kubeless/)
  - It leverages the Kubernetes resources to provide auto-scaling, API routing, monitoring, etc
- It uses **Custom Resource Definitions** to be able to create functions
- It's **open source** and **non-affiliated** to any commercial organization
- It has a UI available for developers to deploy functions

Kubeless

- With kubeless you deploy a function in your preferred language
- Currently, the **following runtimes are supported**:
  - Python
  - NodeJS
  - Ruby
  - PHP
  - .NET
  - Golang
  - Others

Kubeless

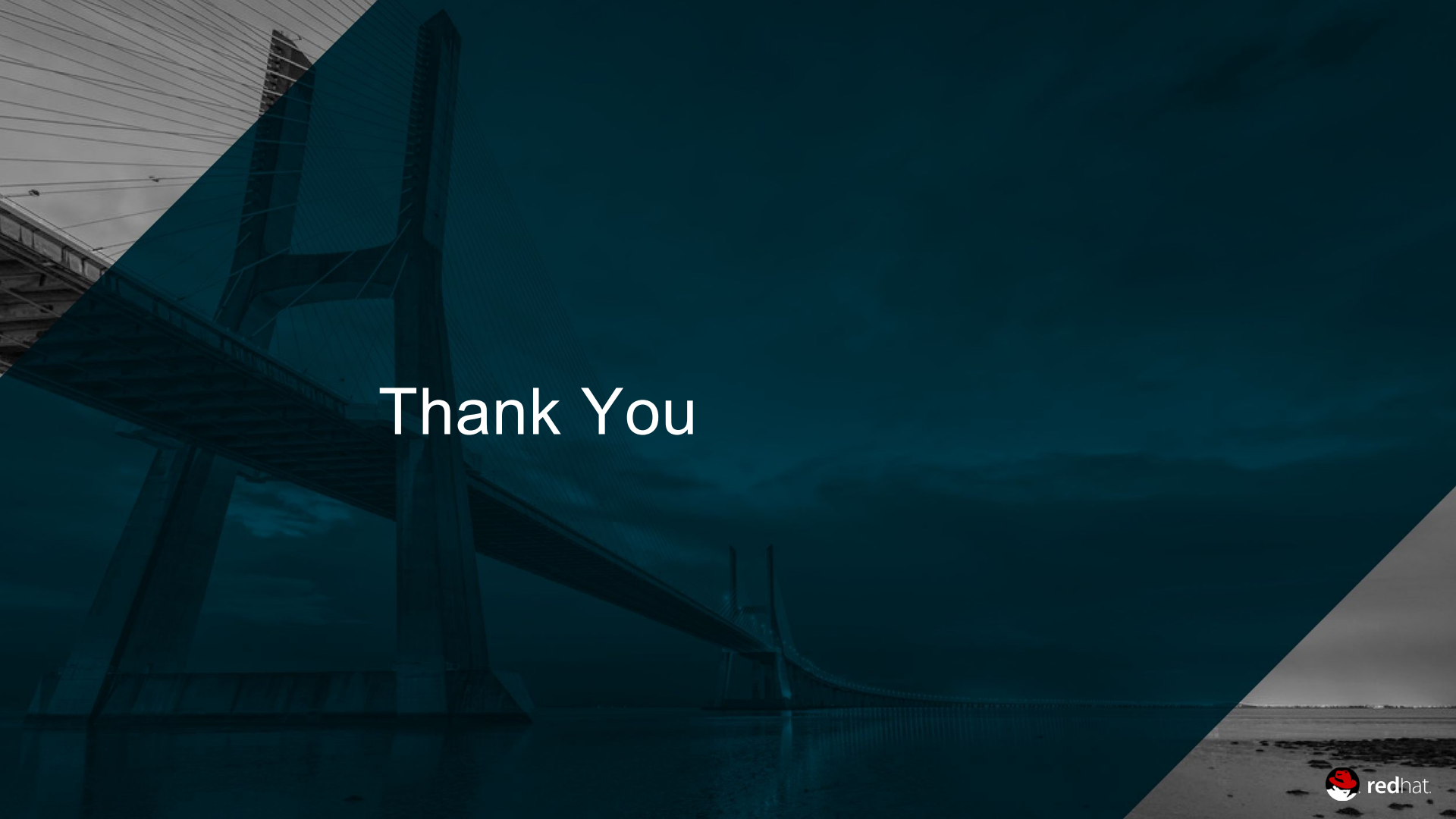
- Once you deployed your function, you'll need to determine how it'll be **triggered**
- Currently, the **following function are supported**:
  - **HTTP functions**
    - HTTP functions gets executed when an HTTP endpoint is triggered
    - You write a function and return the text/HTML that needs to be displayed in the browser
  - Scheduled function

Kubeless

- Once you deployed your function, you need to determine how it'll be **triggered**
- Currently, the **following function are supported**:
  - PubSub (Kafka or NATS)
    - Triggers a function when data is available in Kafka / NATS
  - AWS Kinesis
    - Triggers based on data in AWS Kinesis (similar to Kafka)

Demo Placeholder

Let’s create our own Charts

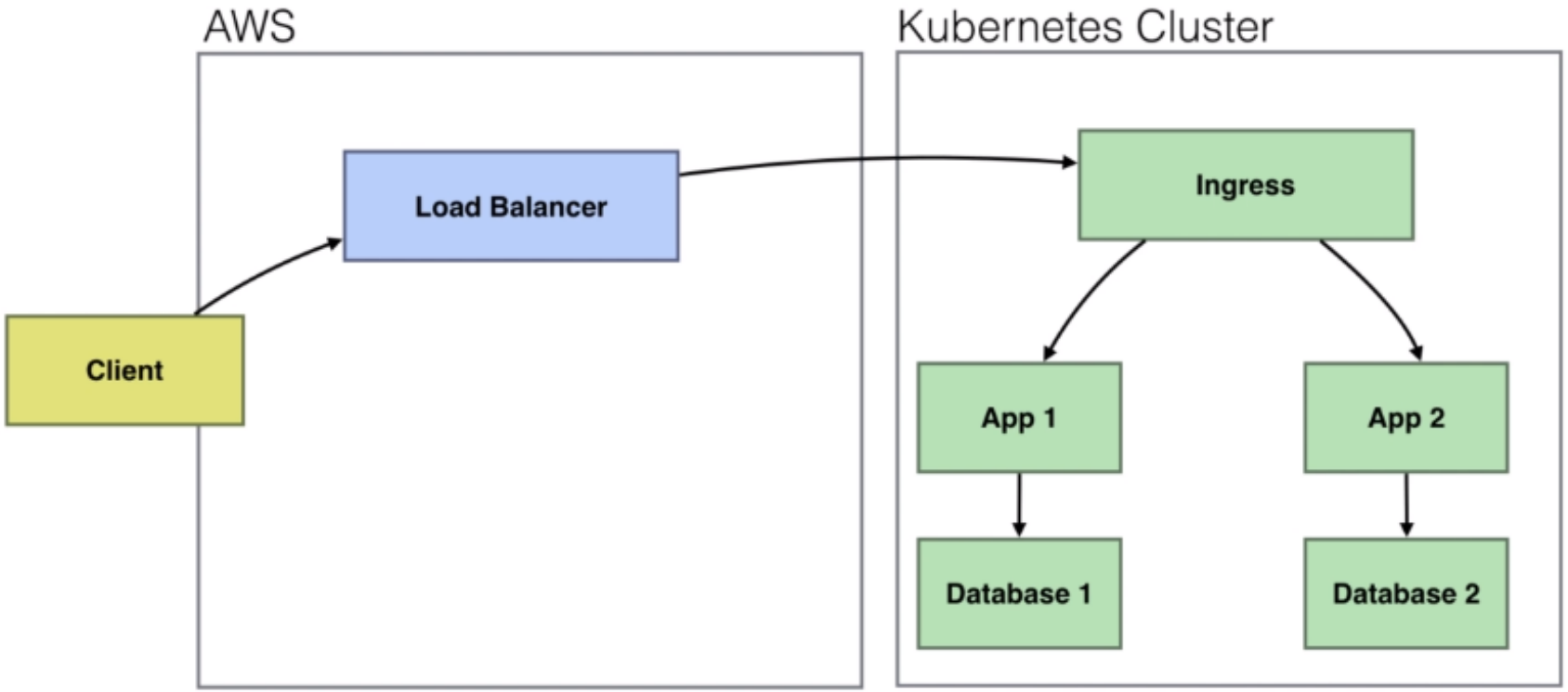

Thank You
redhat.


MicroServices
redhat.

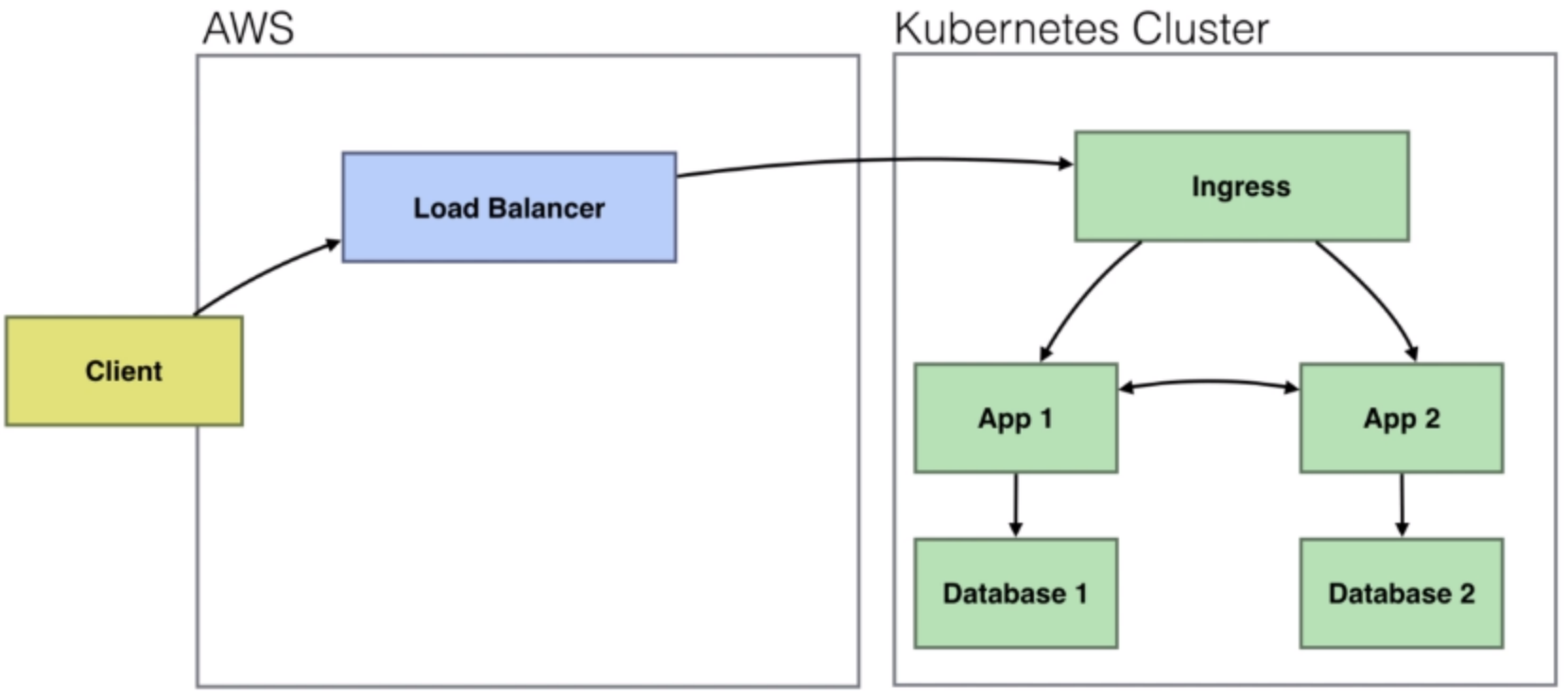
Microservices

- Kubernetes makes it **easy to deploy a lot of diverse applications**
- Those applications can be **monoliths** that don't have anything to do with each other, or **microservices**, small services that make up one application
- The **microservices architecture** is **increasingly popular**
- This approach allows developers to split up the application in **multiple independent parts**
- Having to manage microservices can put an **operational strain** on the engineering team

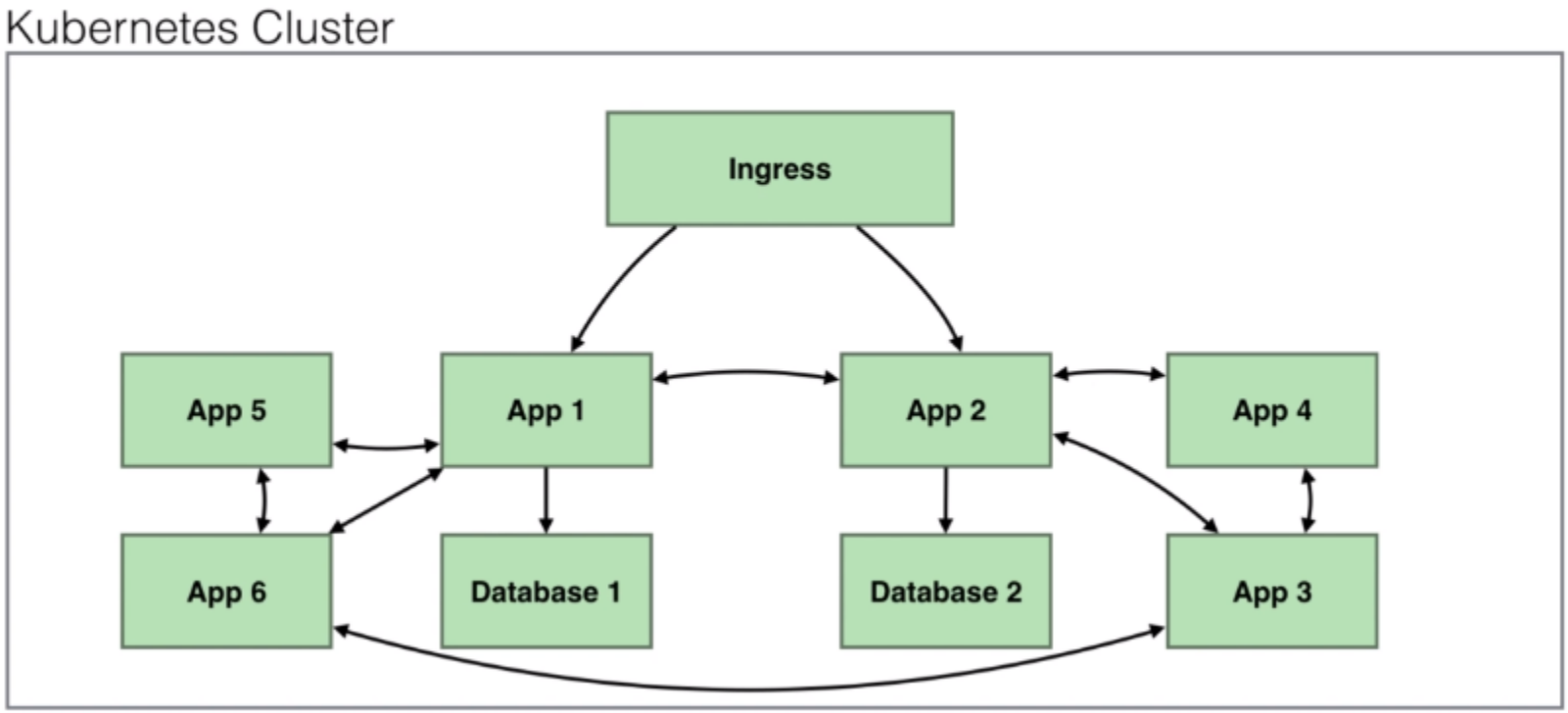
Microservices - Monoliths



Microservices

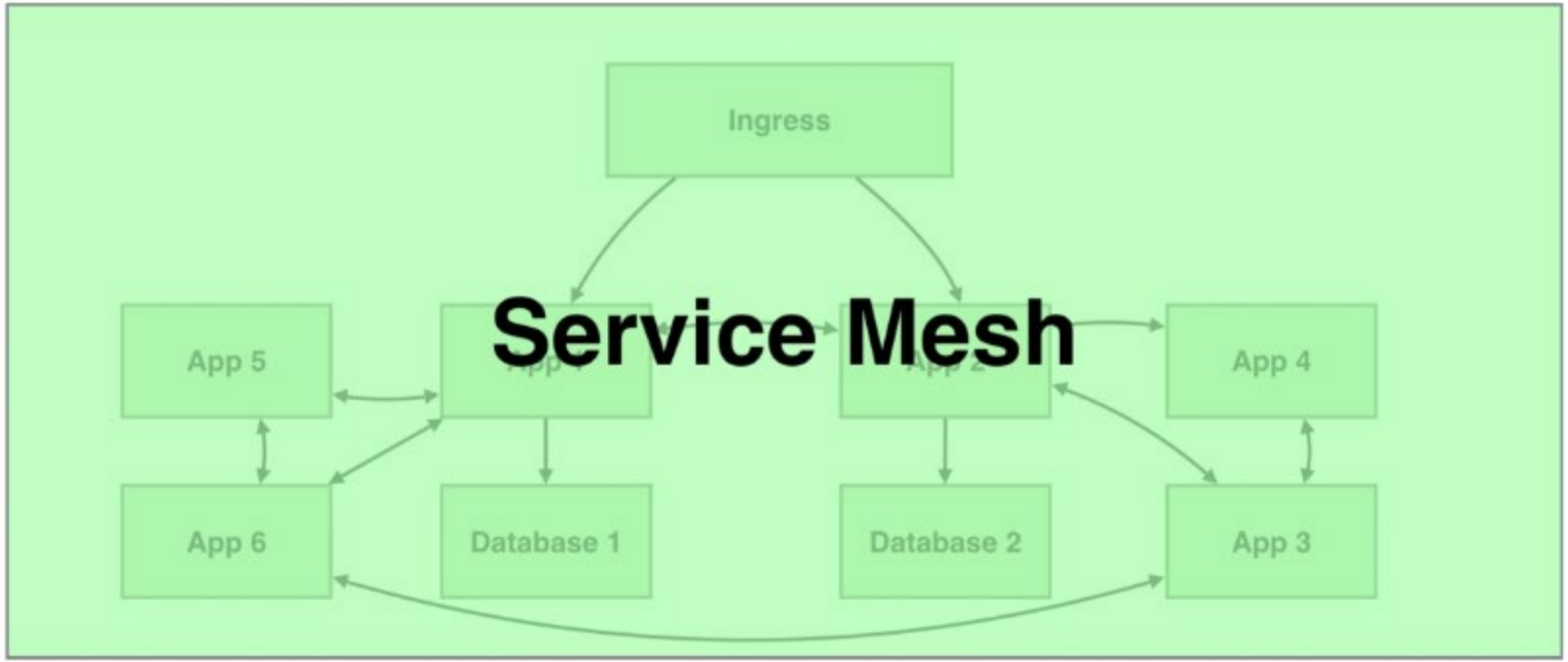


Microservices

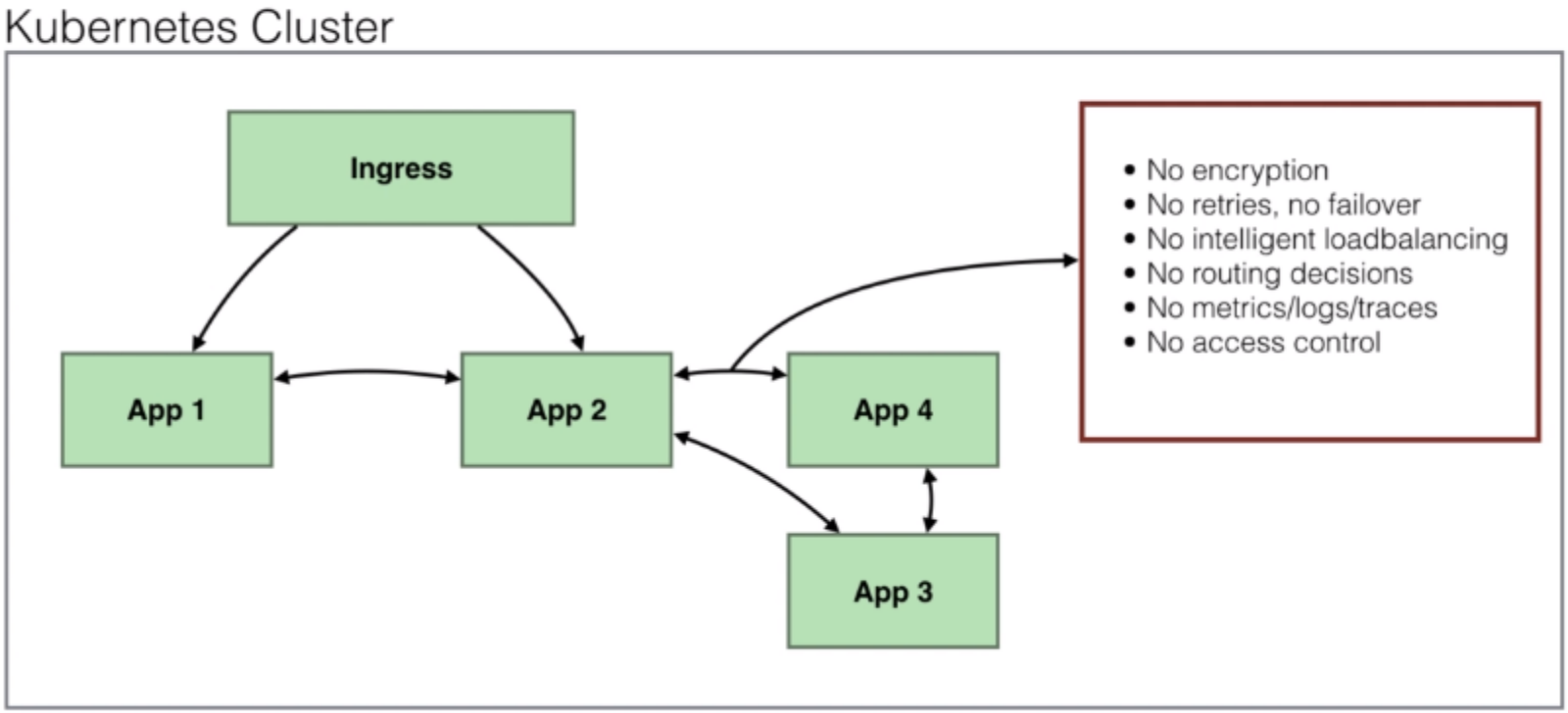


Microservices

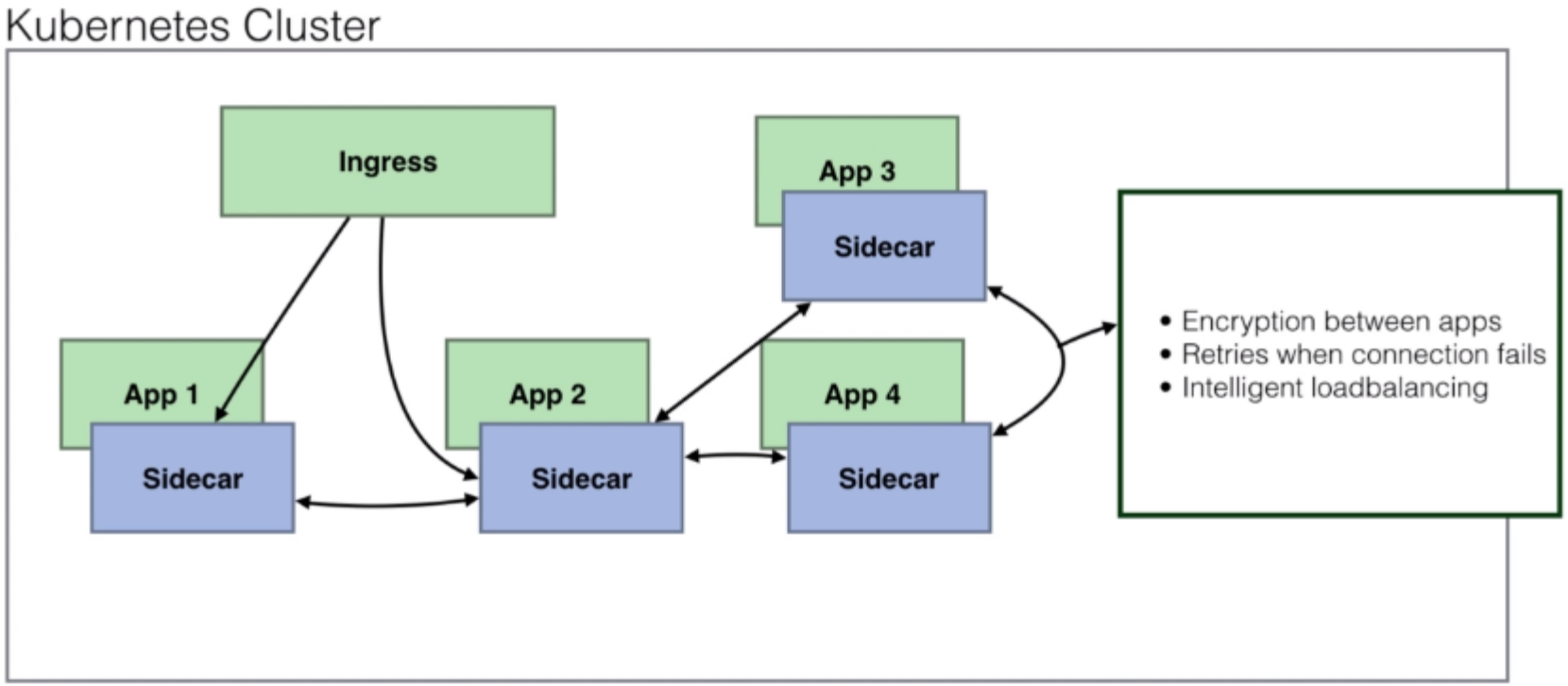
Kubernetes Cluster



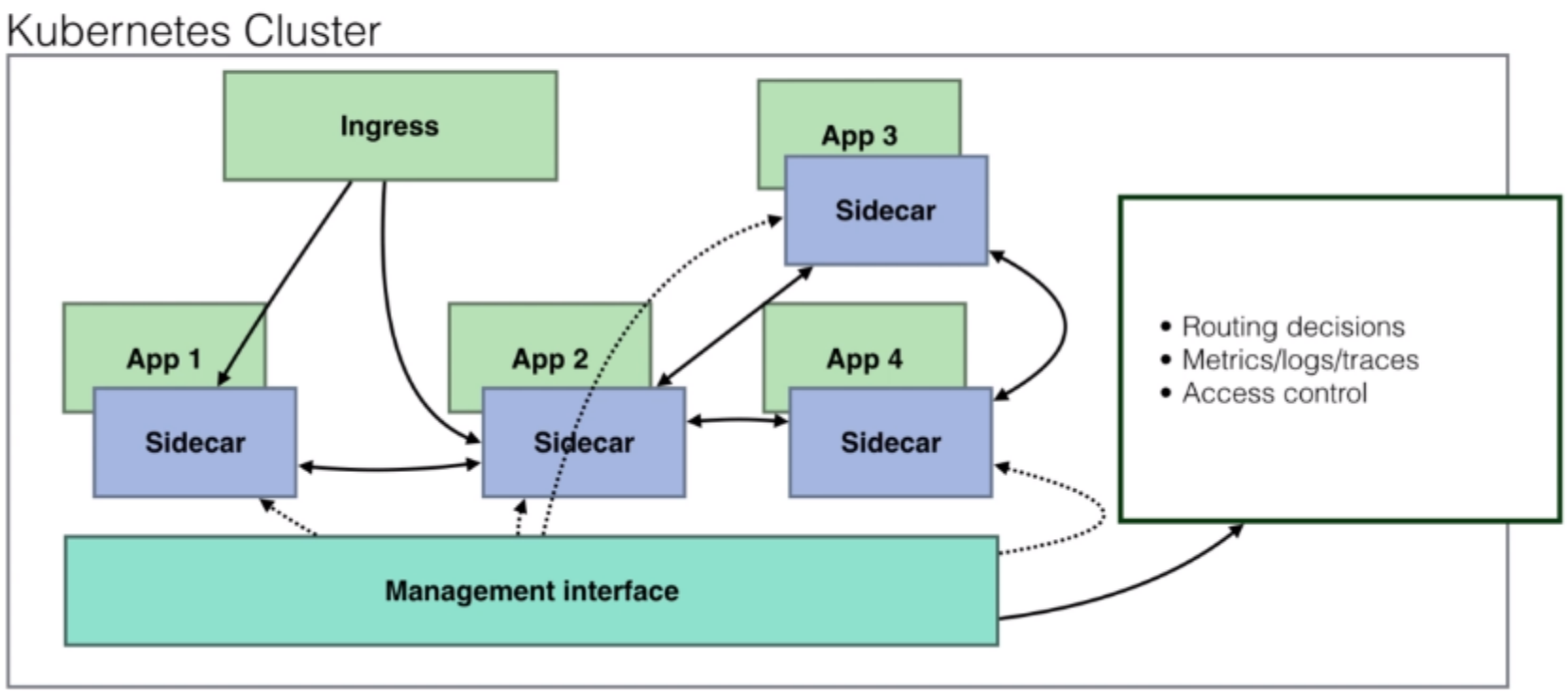
Microservices



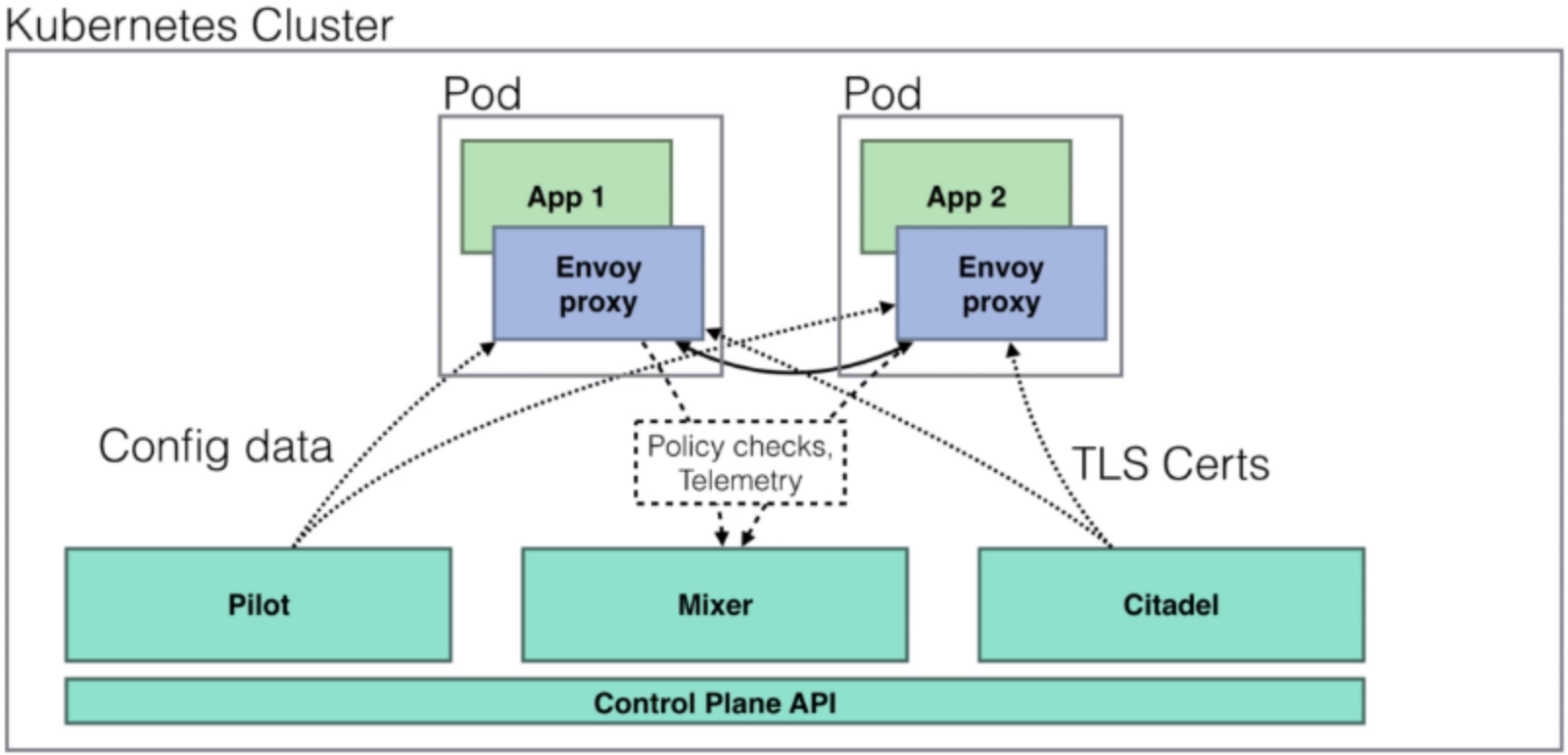
Microservices



Microservices

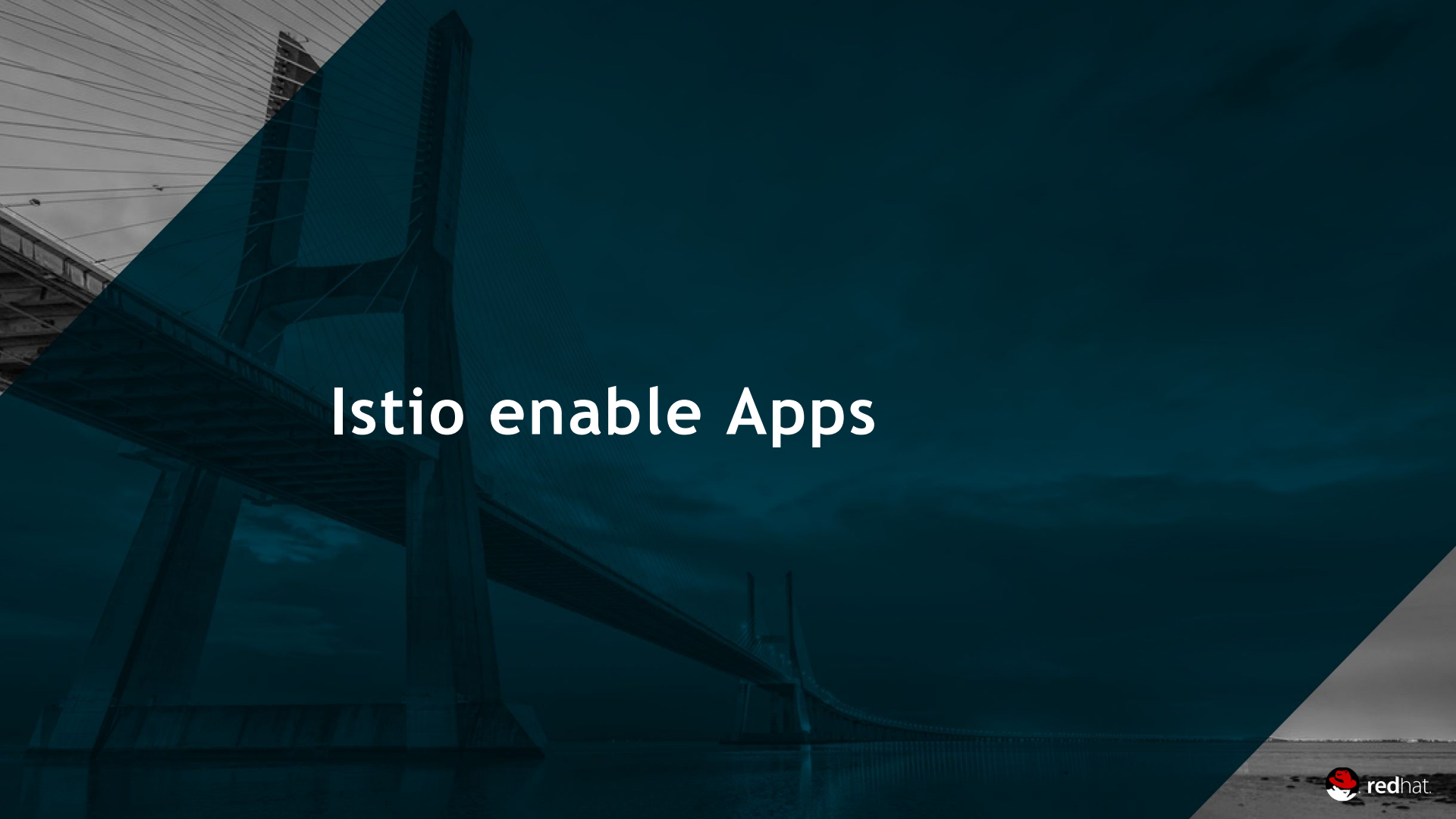


Istio

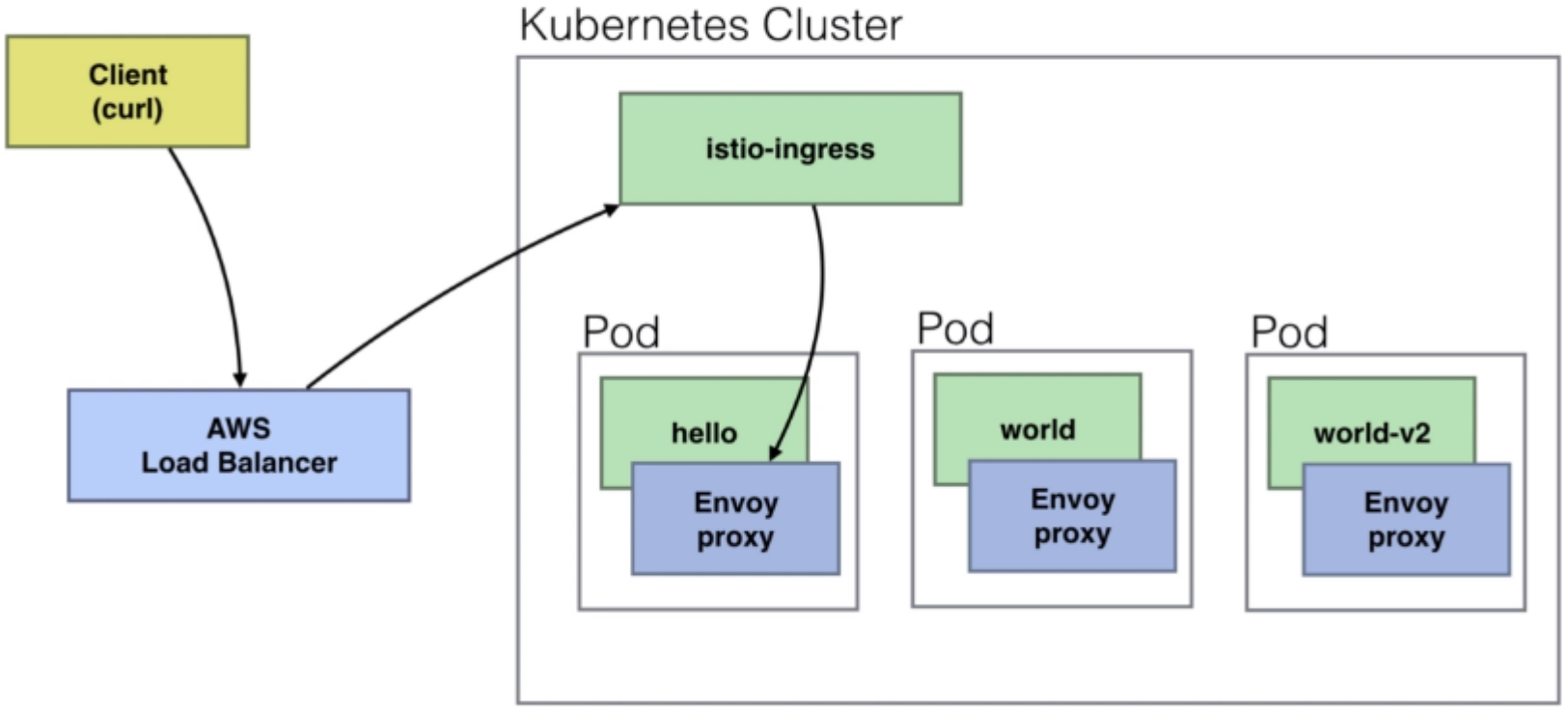

Kubernetes Cluster
Pod
Pod
App 1
Envoy proxy
App 2
Envoy proxy
Config data
Policy checks, Telemetry
TLS Certs
Pilot
Mixer
Citadel
Control Plane API

Demo Placeholder

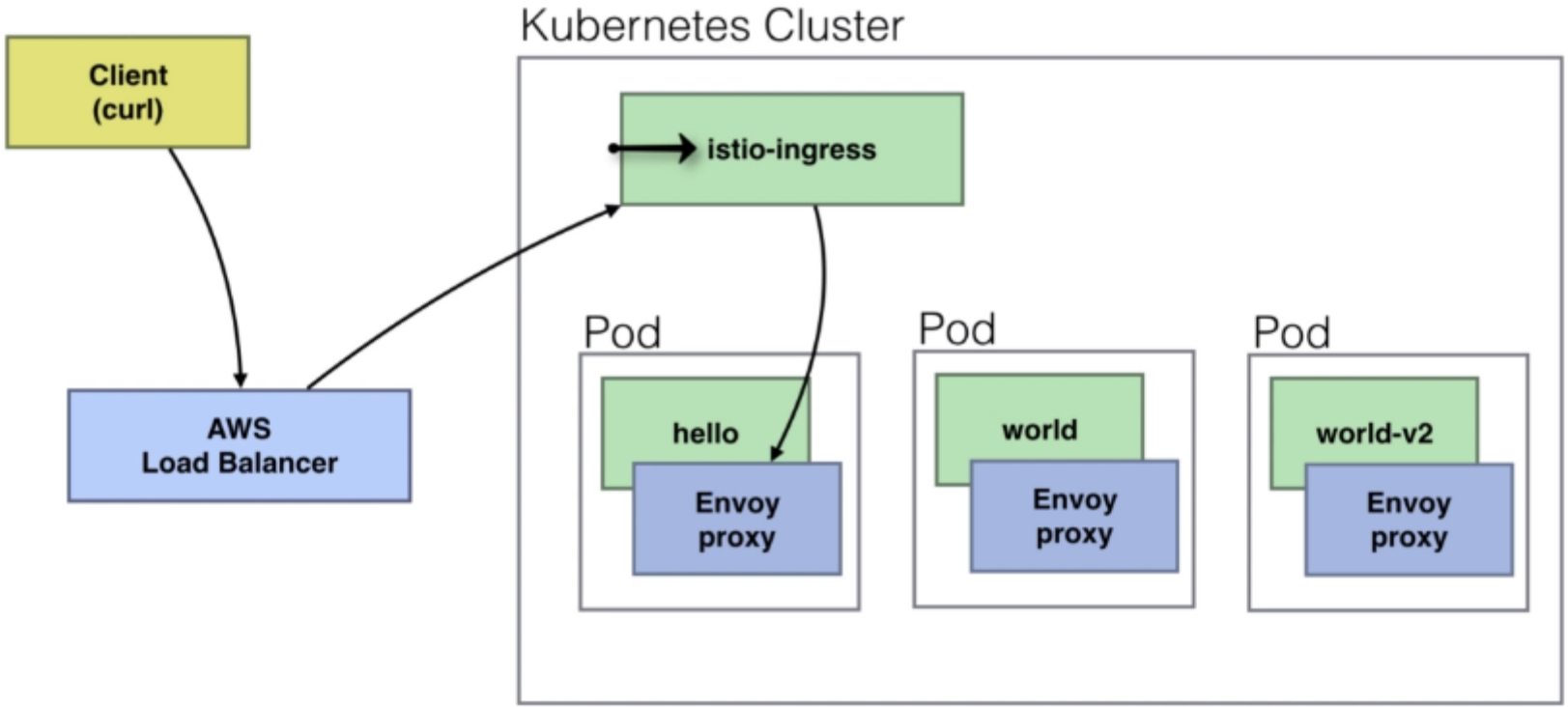
Istio Installation


Istio enable Apps
redhat.

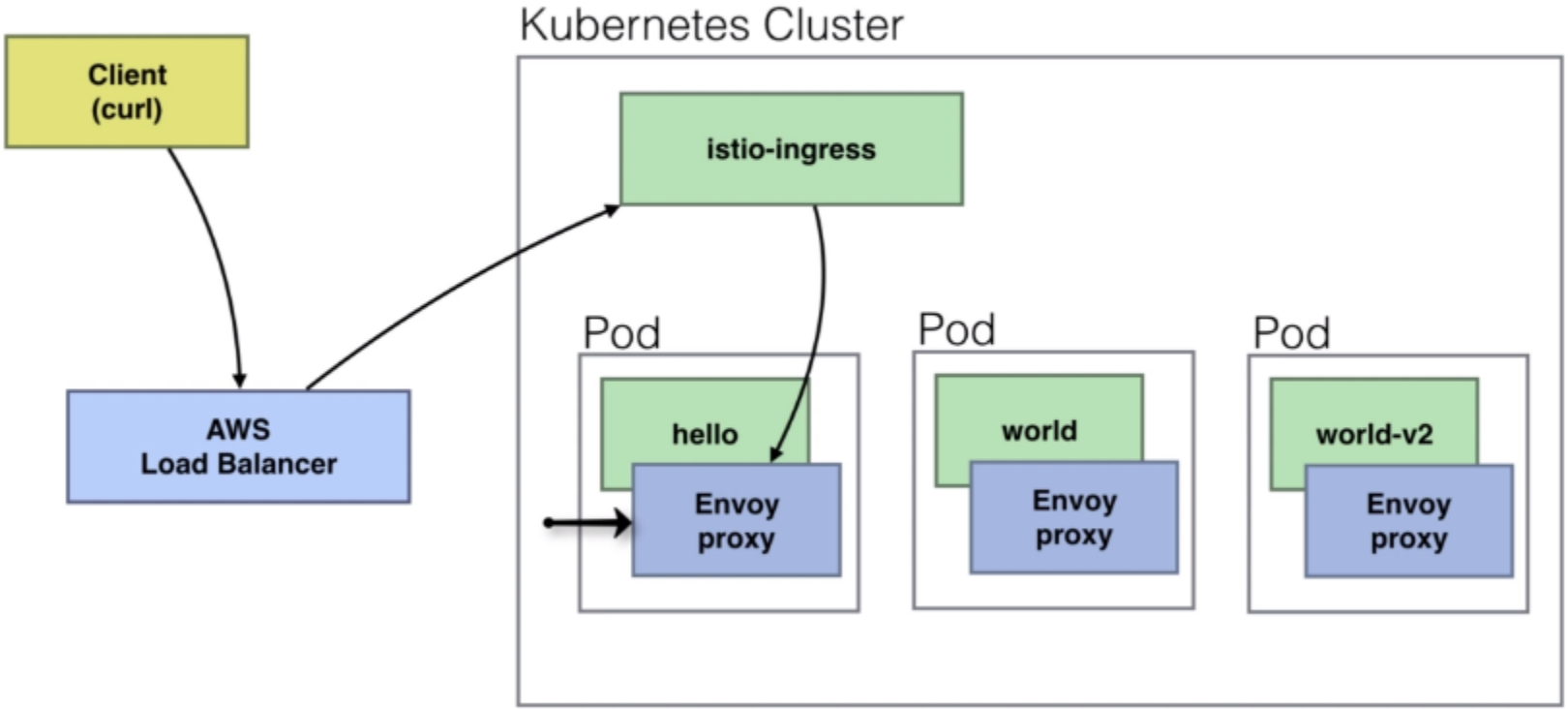
hello world app



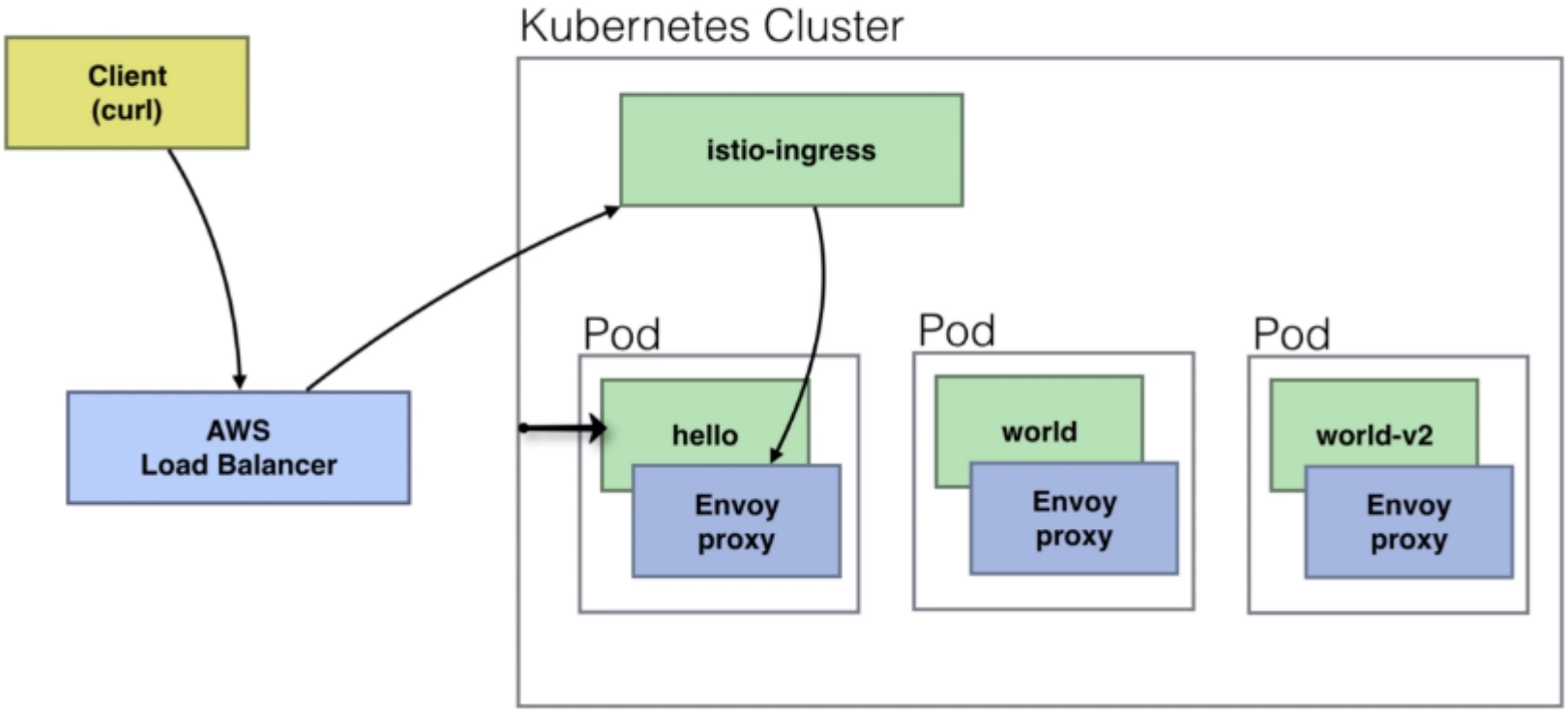
hello world app



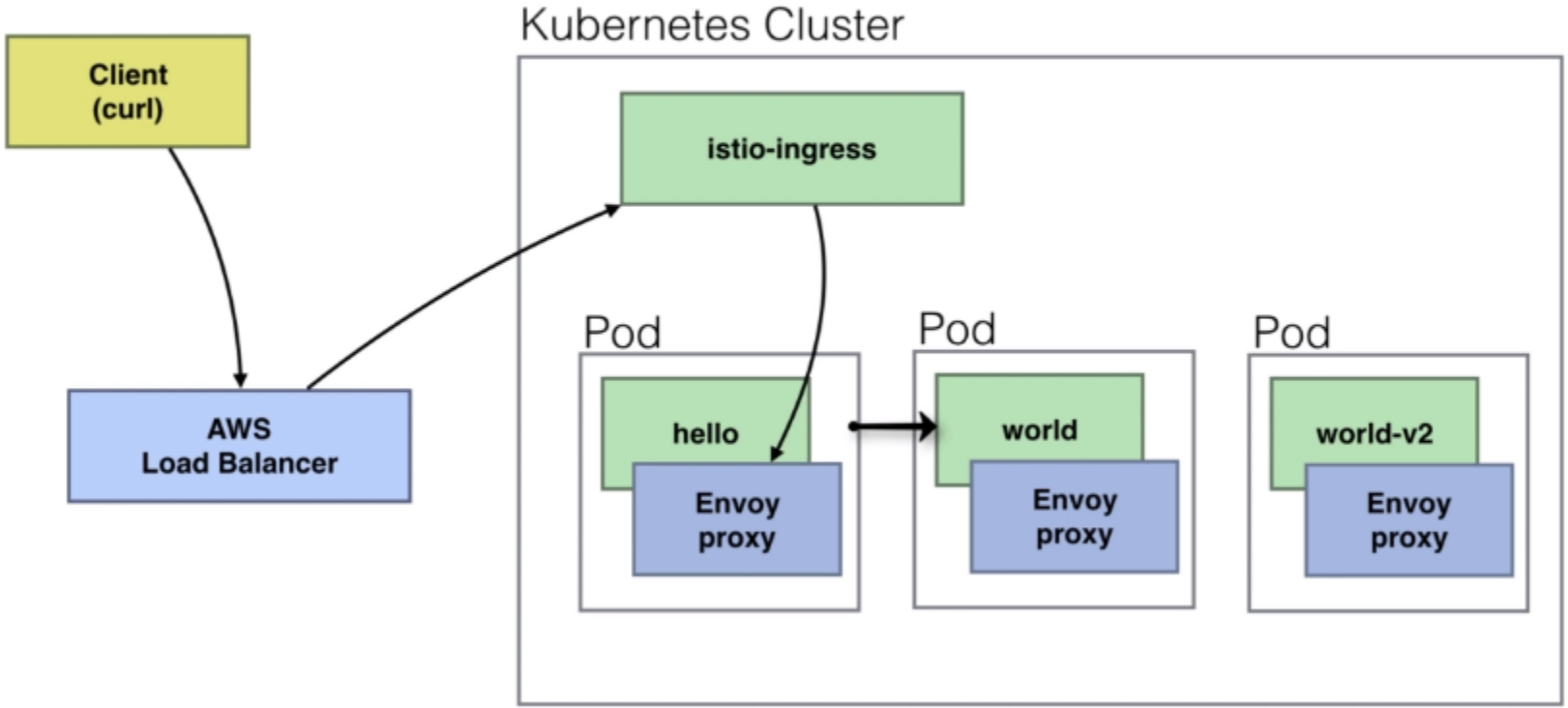
hello world app



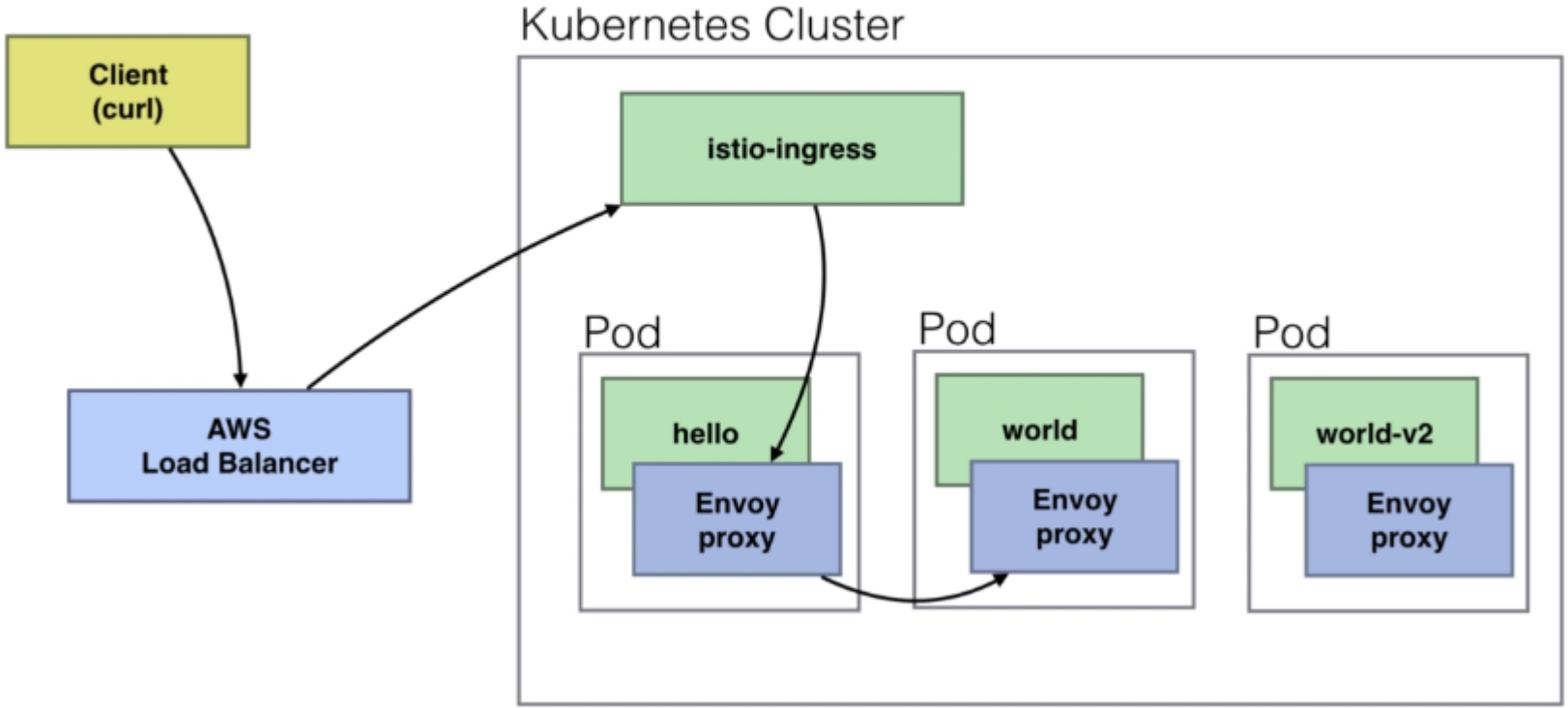
hello world app



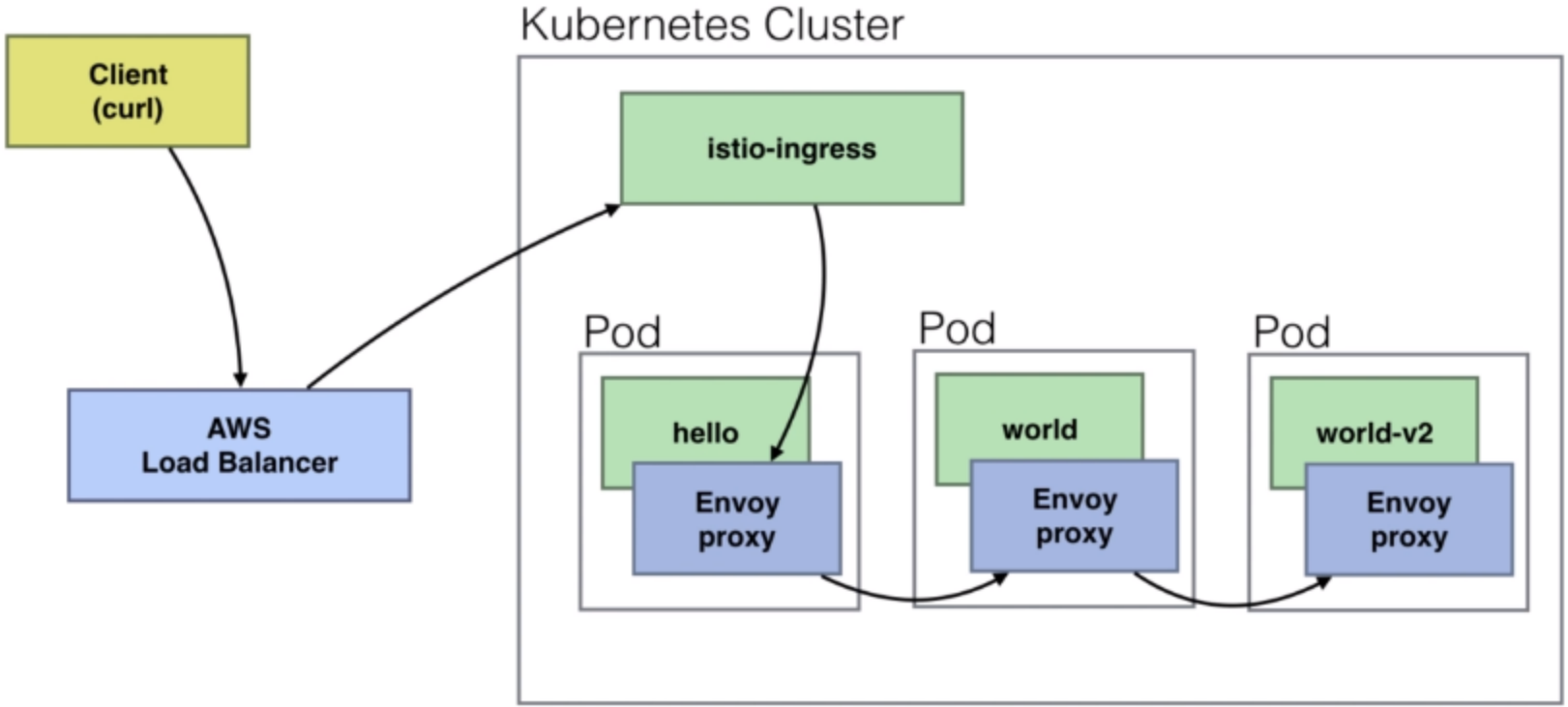
hello world app



hello world app



hello world app

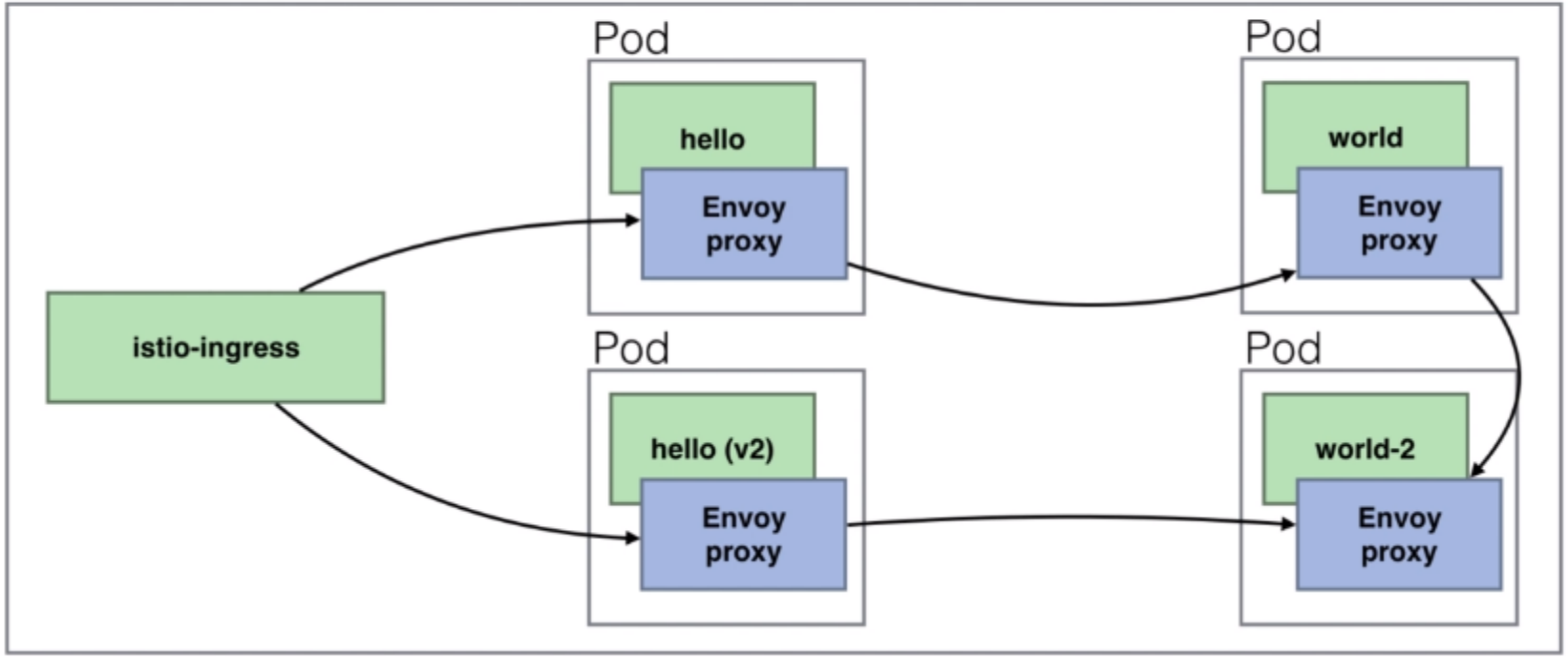


Demo Placeholder

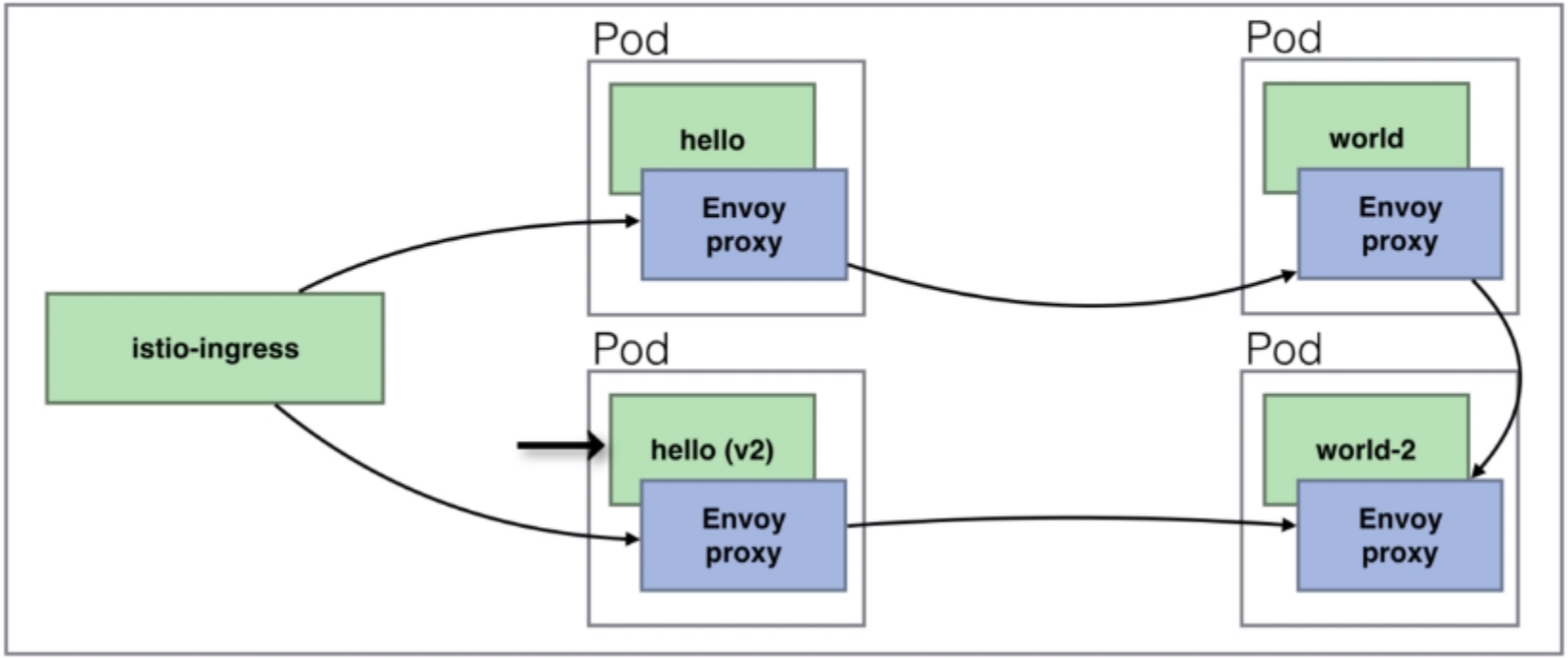
Setup HelloWord App with istio enabled

Routing

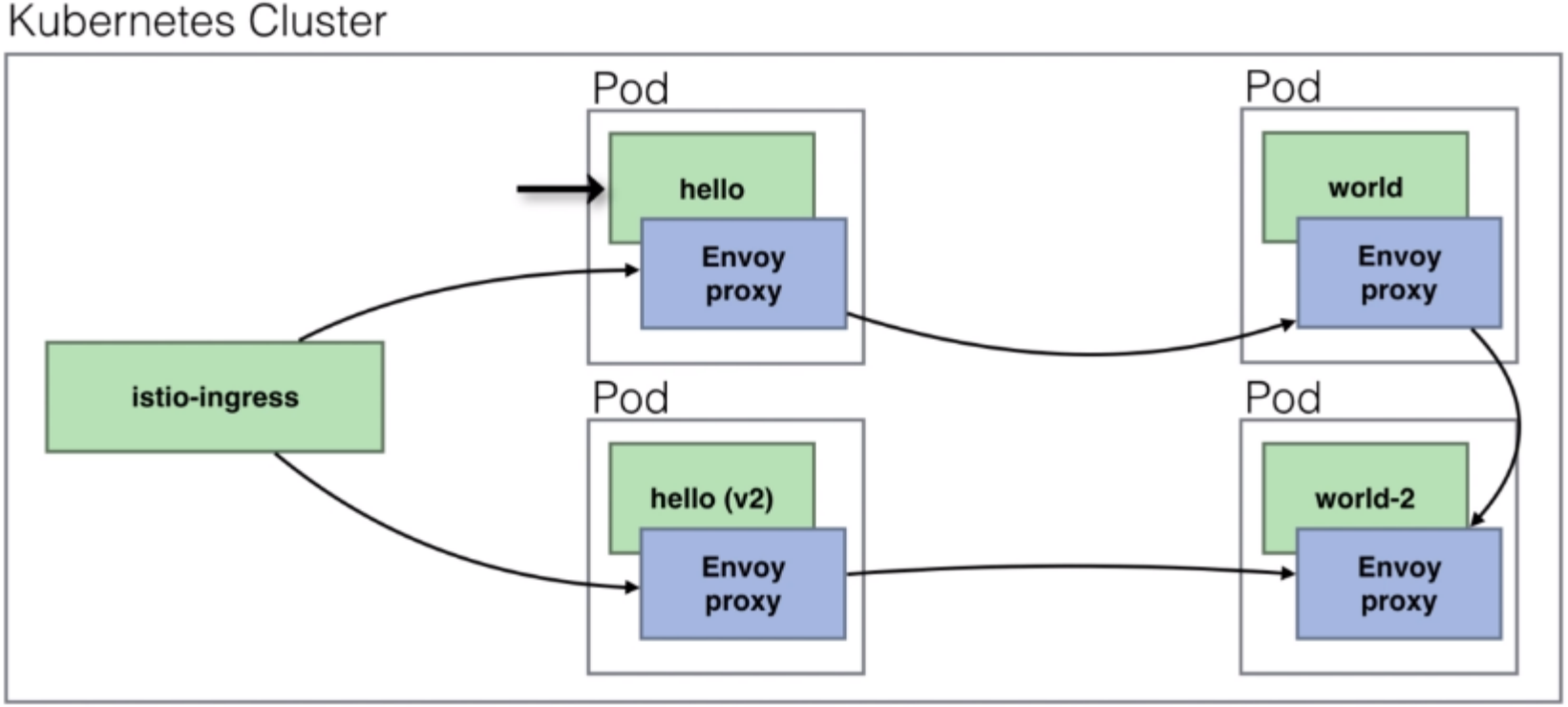
hello world app - v2



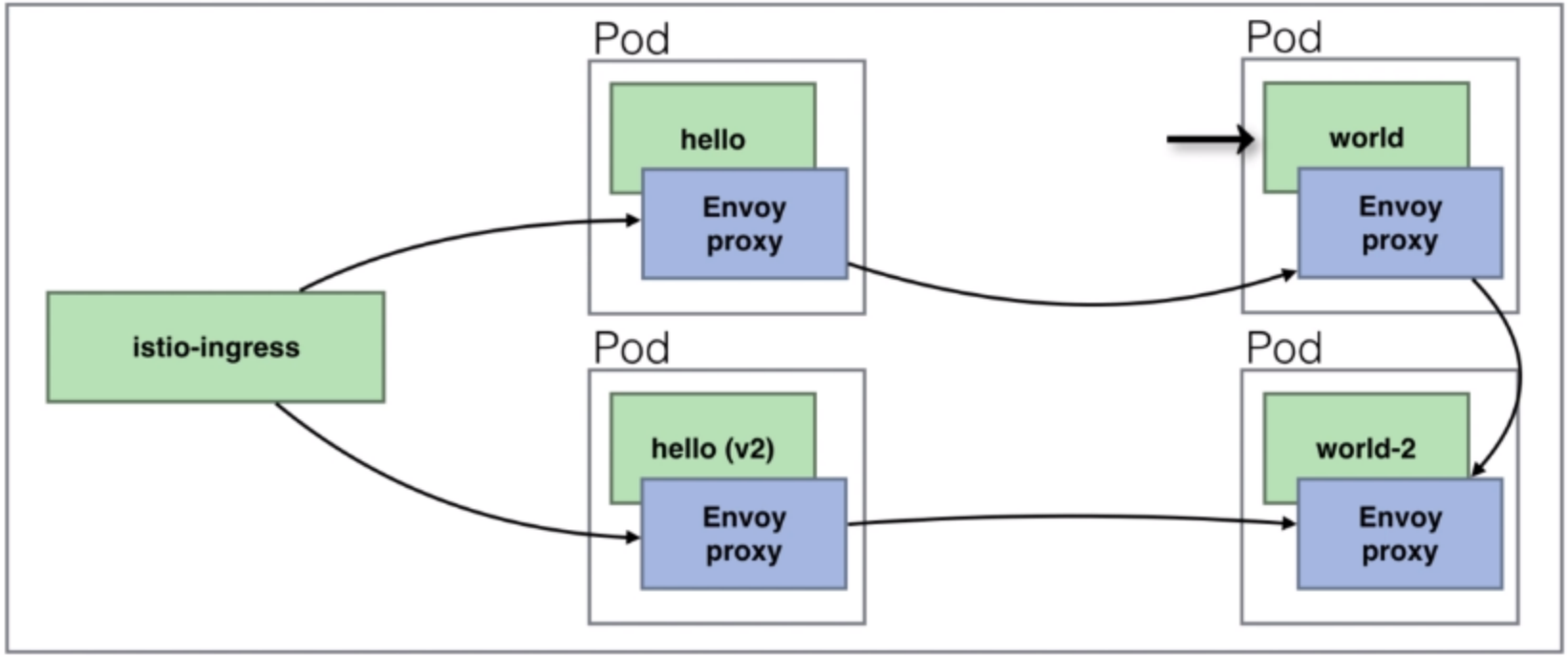
hello world app - v2



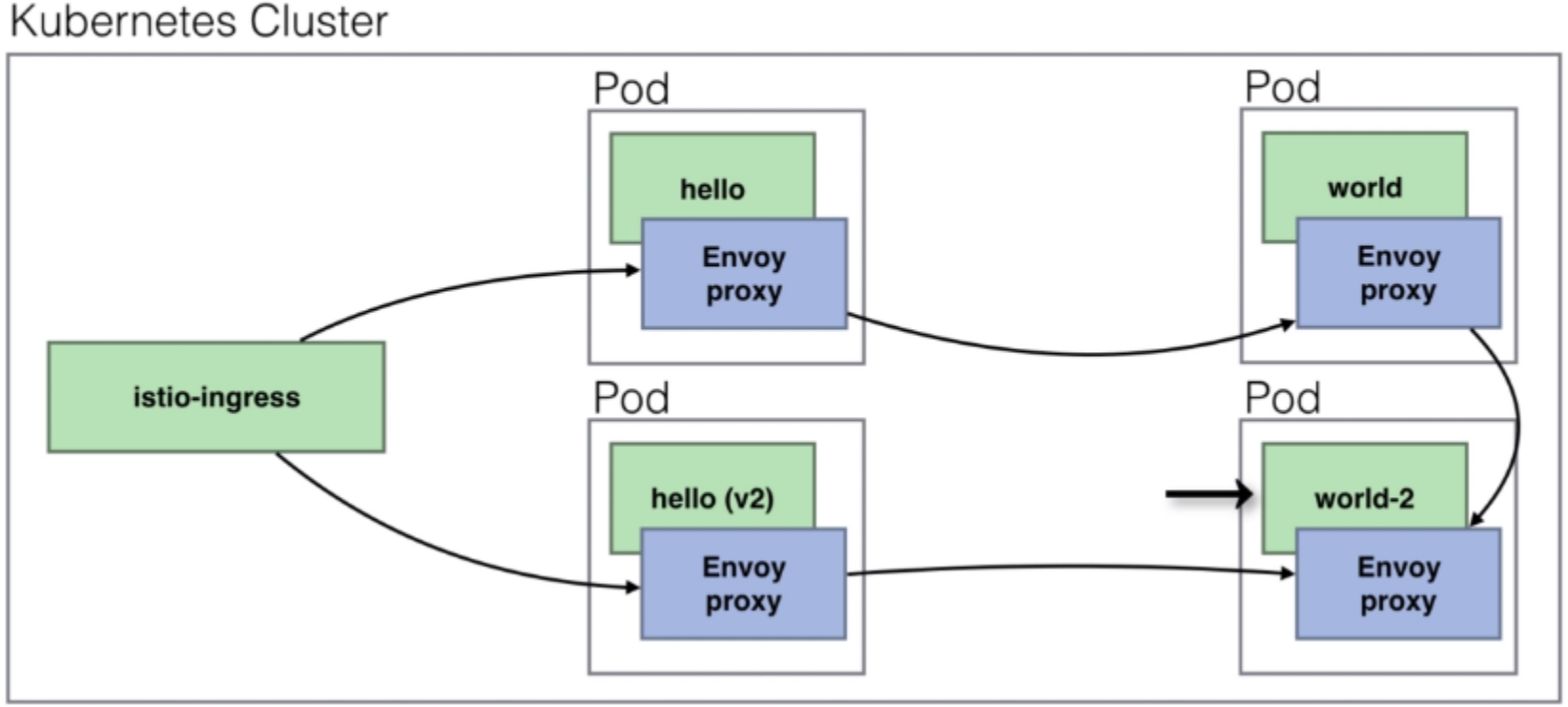
hello world app - v2



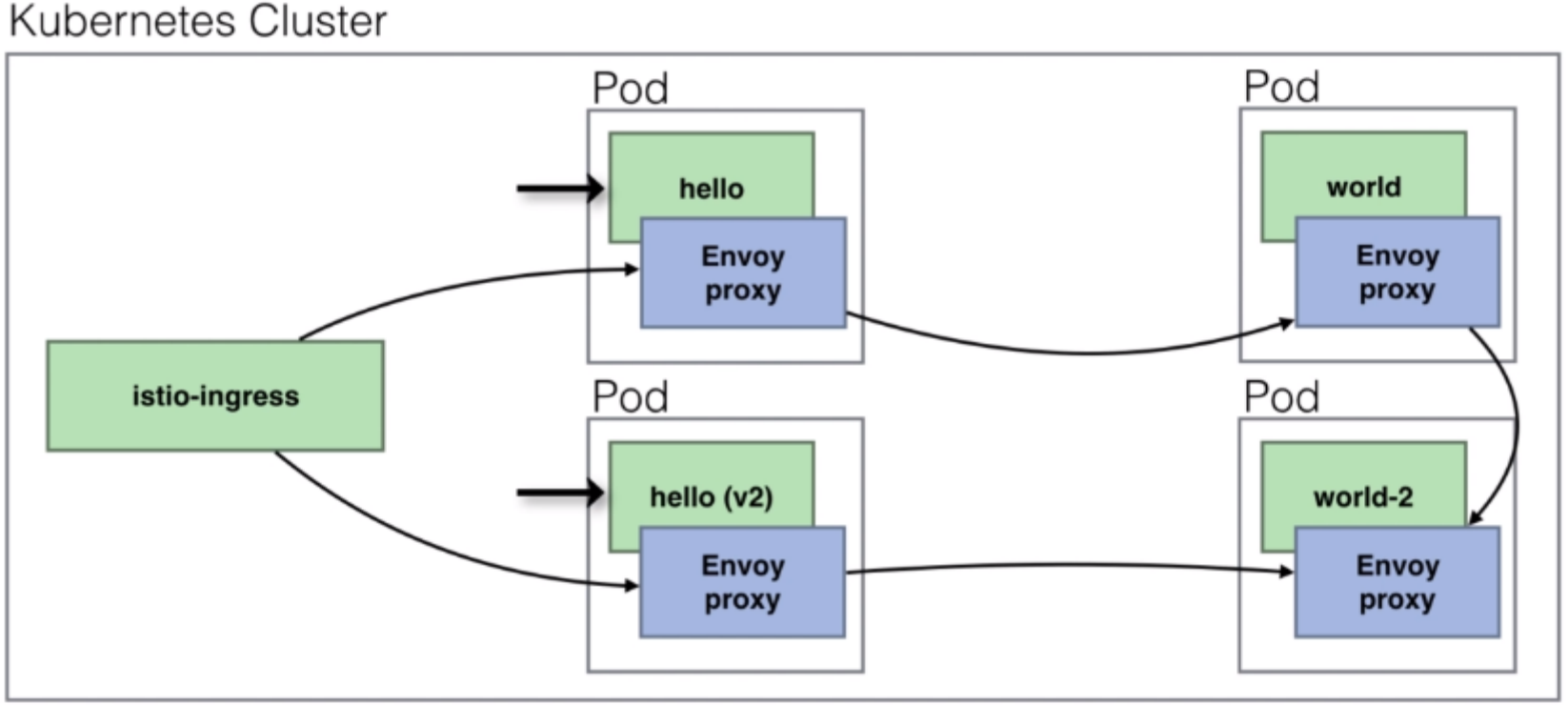
hello world app - v2



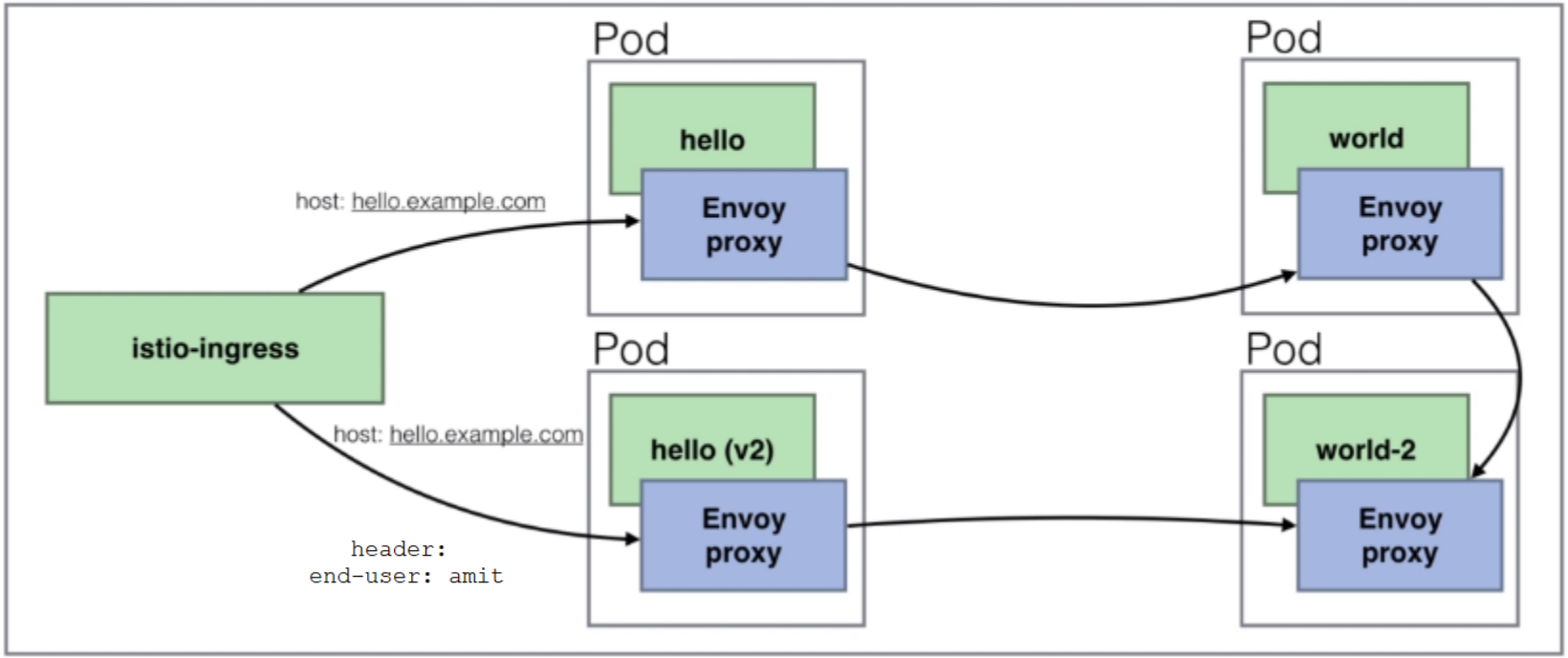
hello world app - v2



hello world app - v2



hello world app - v2

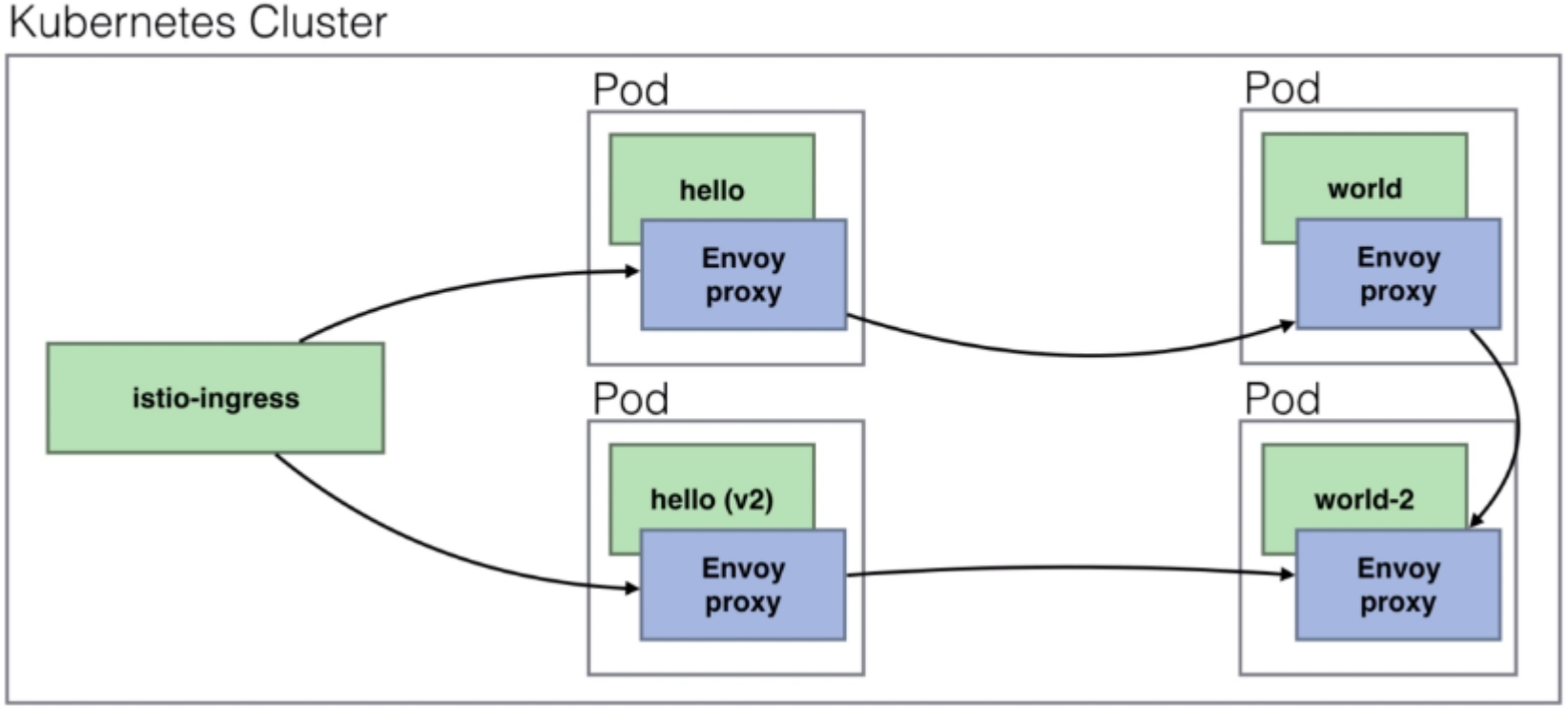


Demo Placeholder

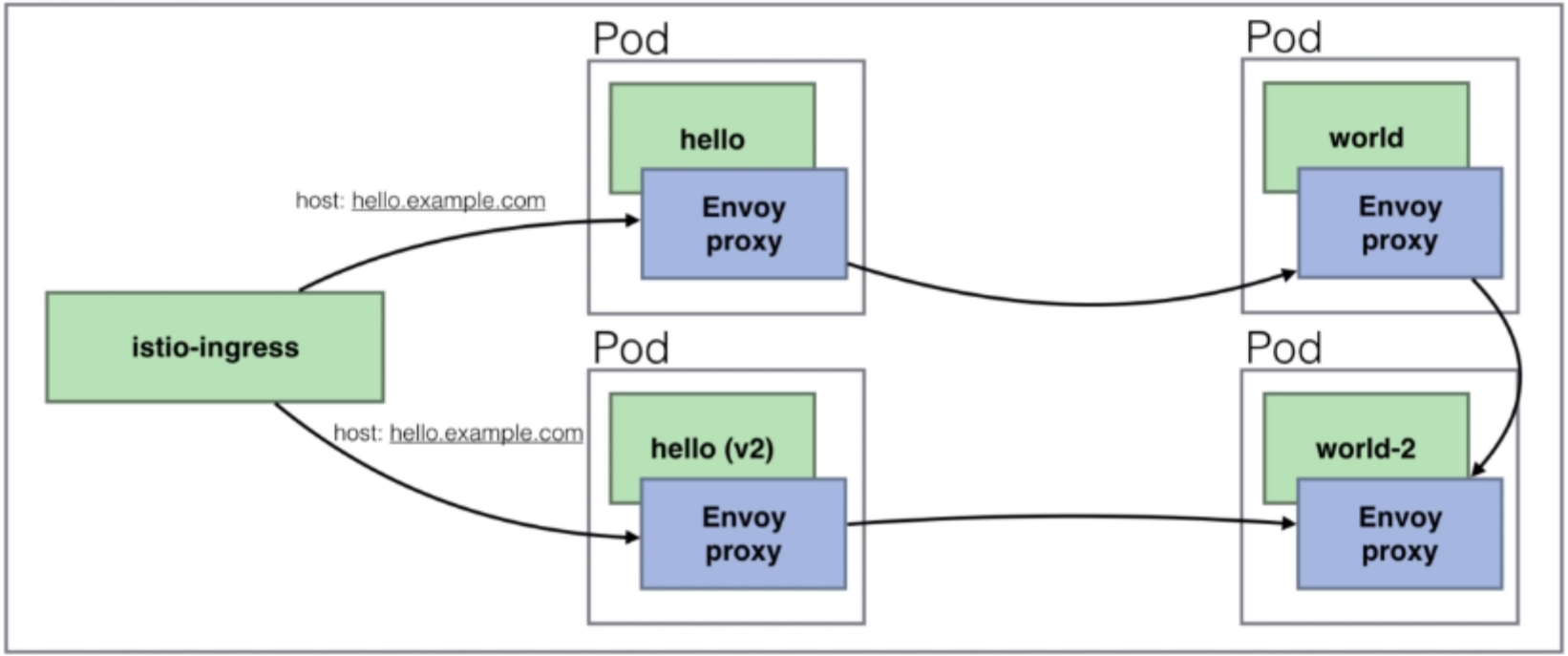
Hello & Hello-v2 with advance routing


Canary Deployments

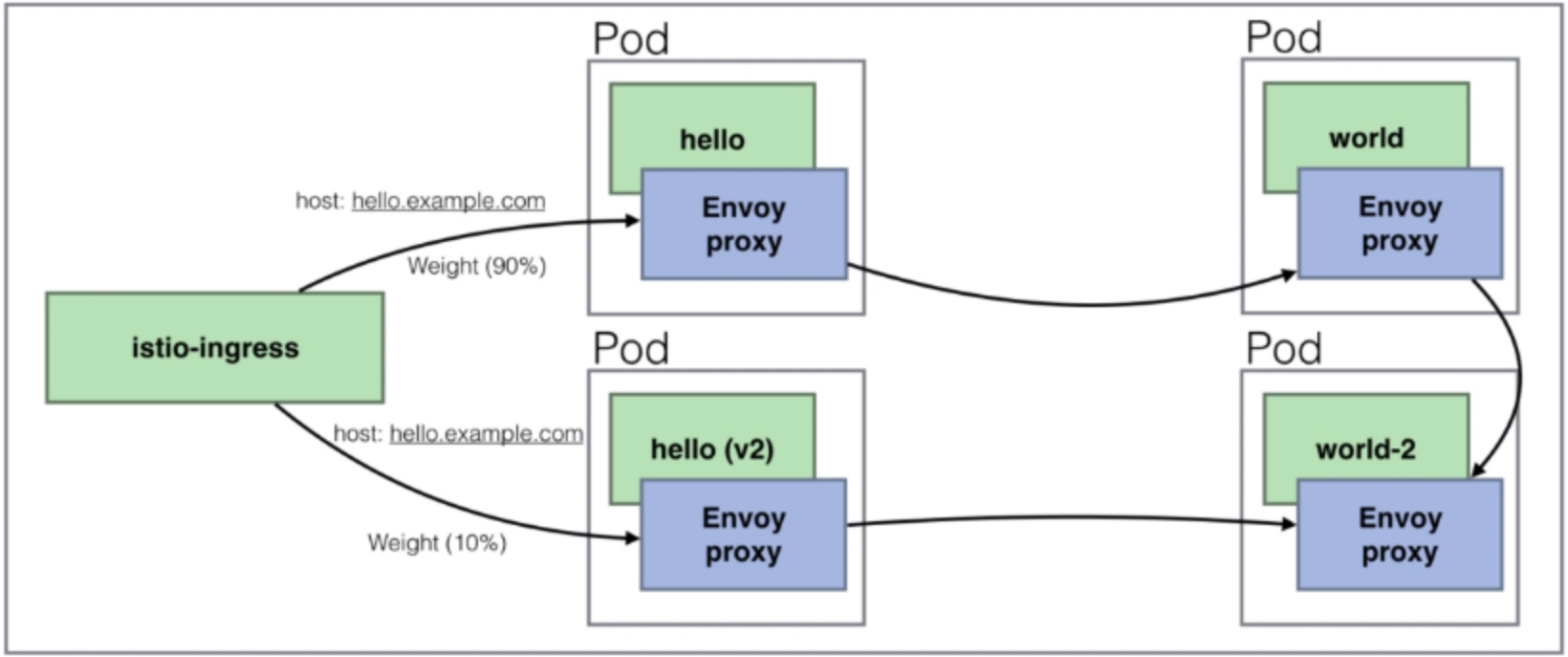
hello world app - v2



hello world app - v2

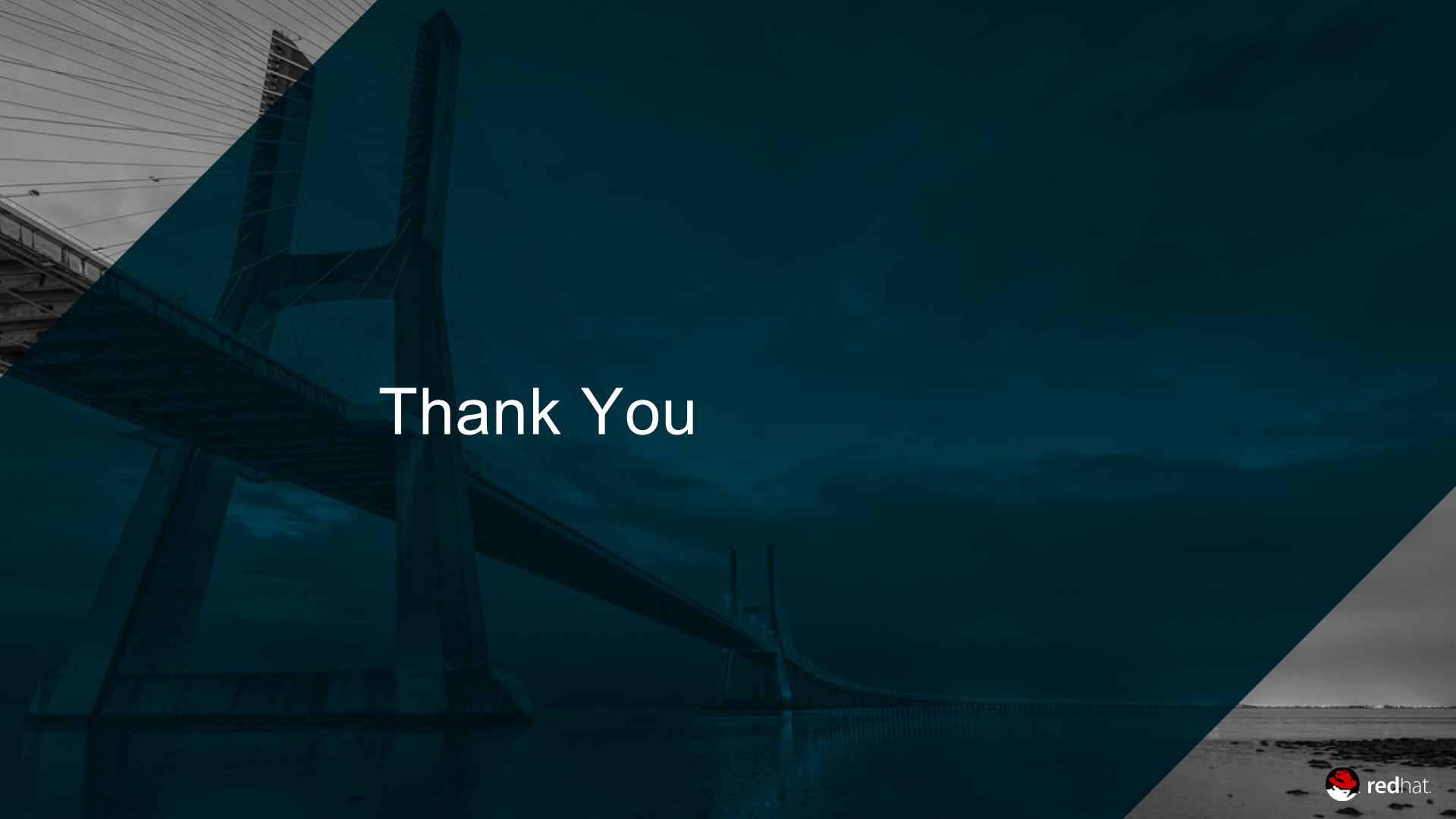


hello world app - v2



Demo Placeholder

Canary Deployment


Thank You
redhat.