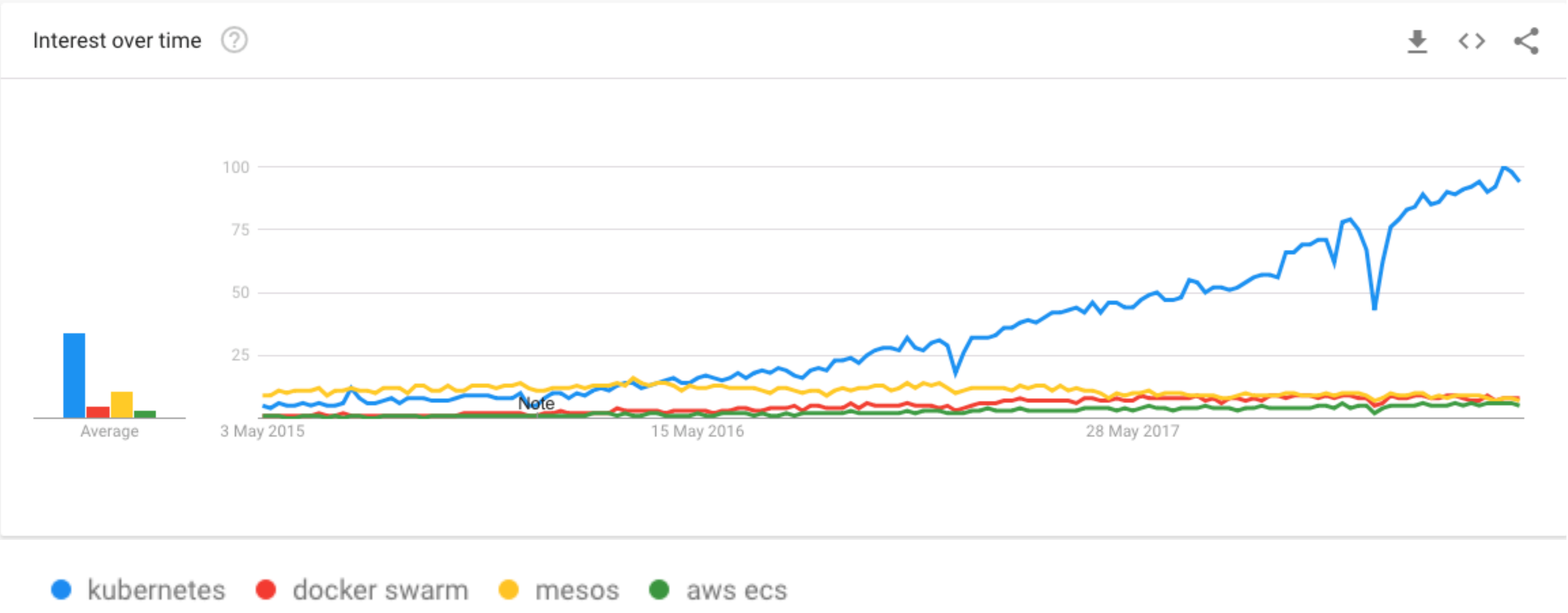
Kubernetes

On-Prem or Cloud Agnostic Kubernetes

Introvideo

On-Prem / Cloud Agnostic Kubernetes

Kubernetes, the **most popular** orchestration tool!



On-Prem / Cloud Agnostic Kubernetes

- Only start this course:
  - If you finished the Course **“Learn DevOps: The Complete Kubernetes Course”**
    - Which you can find on the Udemy marketplace
  - Or, if you are an experienced Kubernetes user
    - I will assume in this course that you know how to run apps on Kubernetes, and know the tooling around this

Course Overview

Kubernetes topic	Technology
Installing kubernetes on-prem	kubeadm
File, Block, and Object storage	Kubernetes Operators, Rook with ceph
Managing SSL (HTTPS apps & endpoints)	cert-manager
LDAP authentication	Dex with LDAP
Service Mesh, Load Balancing and proxying	Envoy, Istio
Networking	Calico
Secret store	Vault
PaaS	OpenShift Origin

Who am I

- My name is Edward Viaene
- I am a **consultant** and **trainer** in Cloud and Big Data technologies
- I’m a big advocate of **Agile** and **DevOps techniques** in all the projects I work on
- I held various roles from **banking** to **startups**
- I have a **background** in Unix/Linux, Networks, Security, Risk, and distributed computing
- Nowadays I specialize in everything that has to do with **Cloud** and **DevOps**

On-Prem or Cloud Agnostic Kubernetes

Introduction

On-Prem or Cloud Agnostic Kubernetes

- Only start this course:
  - If you finished the Course **“Learn DevOps: The Complete Kubernetes Course”**
    - Which you can find on the Udemy marketplace
  - Or, if you are an experienced Kubernetes user
    - I will assume in this course that you know how to run apps on Kubernetes, and know the tooling around this

Course Overview

Kubernetes topic	Technology
Installing kubernetes on-prem	kubeadm
File, Block, and Object storage	Kubernetes Operators, Rook with ceph
Managing SSL (HTTPS apps & endpoints)	cert-manager
LDAP authentication	Dex with LDAP
Service Mesh, Load Balancing and proxying	Envoy, Istio
Networking	Calico
Secret store	Vault
PaaS	OpenShift Origin

Course objectives

- To be able to **use** Kubernetes on-prem or in a Cloud Agnostic way
    - This allows you to **use** Kubernetes in an enterprise environment
- After this course you should be able to deploy Kubernetes anywhere
    - using your own integrations
    - like storage, certificates, authentication, and so on

Who am I

- My name is Edward Viaene

- I am a **consultant** and **trainer** in Cloud and Big Data technologies

- I'm a big advocate of **Agile** and **DevOps techniques** in all the projects I work on

- I held various roles from **banking** to **startups**

- I have a **background** in Unix/Linux, Networks, Security, Risk, and distributed computing

- Nowadays I specialize in everything that has to do with **Cloud** and **DevOps**

Online Training

- **Online training** on Udemy
  - **DevOps**, **Distributed Computing**, **Cloud, Terraform, Big Data (Hadoop)**
  - Using online video lectures
  - 45,000+ enrolled students in 100+ countries

Kubernetes Introduction

Trail map

Kubernetes installation

using kubeadm

Kubeadm

- **Kubeadm** is a **toolkit** by Kubernetes to create a cluster
- It works on **any deb / rpm compatible Linux OS**, for example Ubuntu, Debian, Redhat, or CentOS
  - This is the main **advantage** of kubeadm, because **a lot of tools are OS / Cloud specific**
- The tool itself is still in beta (Q1 2018), but is expected to become **stable** somewhere **this year**
- It's very **easy to use** and lets you spin up your Kubernetes cluster in just a **couple of minutes**

Kubeadm

- Kubeadm supports **bootstrap tokens**
  - Those are **simple tokens** that can be used to **create a cluster**, or to **join nodes** later on
  - The **tokens** are of the **format** *abcdef.0123456789abcdef*
- Kubeadm supports upgrading/downgrading clusters
- It **does not install a networking solution**
  - You have to install a **C**ontainer **N**etwork **I**nterface - compliant network solution yourself using kubectl apply (as I will show in the demo)

Kubeadm prerequisites

- deb / rpm compatible system (or CoreOS' Container Linux)
- **2 GB** of memory
- **2 CPUs** for the master node
- Network connectivity between the nodes
  - Can be a **private network** (internal IP addresses)
  - Or **public routable internet IP addresses** (in this case its best to use a firewall to only allow access within the cluster and to the users)
- Typically, you need minimal 2 nodes (one master node and one to schedule pods on)

Demo

- In the demo I will use DigitalOcean to spin up droplets (VMs)
- You can get $10 worth of credits if you use the following link to sign up:
  - https://m.do.co/c/007f99ffb902
- A **2 GB memory droplet currently costs $10 / month**, but it is **billed per hour**, so you can run 2x 2GB RAM droplets for half a month or 4x 2GB RAM droplets for bit more than a week

Demo

installing Kubernetes using kubeadm

Operators

Operators

- An Operator is a method of **packaging**, **deploying**, and **managing** a Kubernetes Application (definition: https://coreos.com/operators/)

- It puts **operational knowledge** in an application

    - It brings the user **closer to the experience of managed cloud services**, rather than having to know all the specifics of an application deployed to Kubernetes

    - Once an Operator is deployed, it can be **managed using Custom Resource Definitions** (arbitrary types that extend the Kubernetes API)

- It also provides a great way to deploy Stateful services on Kubernetes (because a lot of complexities can be hidden from the end-user)

Custom Resource Definitions

- Custom Resource Definitions (CRDs) are **extensions of the Kubernetes API**
- It allows the Kubernetes user to use **custom objects** (the objects you use in yaml files), and create / modify / delete those objects on the cluster
  - For example: you could run a *kubectl create* on a yaml file containing a custom database object, to spin up a database on your cluster
- The custom objects are **not necessarily available on all clusters**
  - They can be dynamically registered / deregistered
  - **Operators include CRDs**
    - By adding an Operator, you'll register these custom resource definitions

Operators example

- **etcd, Rook, Prometheus, and Vault** are examples of technologies can be deployed as an Operator

- Let’s use **etcd** as an example (etcd is a distributed key value store)

    - Once the etcd operator is deployed, a new etcd cluster can be created by using the following yaml file:

```
apiVersion: "etcd.database.coreos.com/v1beta2"
kind: "EtcdCluster"
metadata:
  name: "example-etcd-cluster"
spec:
  size: 3
  version: "3.2.13"
```

Operators example

- Resizing the cluster is now just a matter of changing the yaml file:

```
apiVersion: "etcd.database.coreos.com/v1beta2"
kind: "EtcdCluster"
metadata:
  name: "example-etcd-cluster"
spec:
  size: 5  # was 3 previously
  version: "3.2.13"
```

- After making the edit, the changes can be applied using *kubectl apply*

- The same is true for **the version** number: to upgrade the etcd cluster you just change the version, enter *kubectl apply*, and the etcd cluster will be upgraded

Operators

- Using operators **simplifies deployment and management** a lot

- This example was using an etcd cluster, but more software is being released using operators for Kubernetes

  - For example the PostgreSQL operator by Zalando (https://github.com/zalando-incubator/postgres-operator)

  - Or the MySQL Operator, providing you with a simple API to create a MySQL Database (https://github.com/oracle/mysql-operator):

```
apiVersion: mysql.oracle.com/v1
kind: MySQLCluster
metadata:
  name: myappdb
```

Operators

- You can also build your own operators, using the following tools:
  (Source: https://coreos.com/operators/)
  - **The Operator SDK:** makes it easy to build an operator, rather than having to learn the Kubernetes API specifics
  - **Operator Lifecycle Manager:** oversees installation, updates, and management of the lifecycle of all the operators
  - **Operator Metering:** Usage reporting
- I'll be using **Operators** in this course
  - The next lectures will be about **Rook**, which will be deployed using Operators

Rook

Introduction to Rook

Rook

- Rook is an **open source orchestrator** for **distributed storage systems** running in Kubernetes. (definition: https://rook.io/docs/rook/master/)
  - Rook allows you to **use storage systems on Kubernetes clusters** (that cannot use public cloud storage, or want to be cloud agnostic)
  - If you're on the public cloud, it's very easy to **attach a storage volume to a pod**, to allow your app to persist its data, even when the pod or node shuts down
  - It's not that easy when you're **not on the major cloud providers** like AWS / Azure / Google Cloud
- **Rook** wants to make it easy for you to use a **storage system**, even when you're not on one of those major cloud providers, or using an **on-prem cluster**

Rook

- Rook **automates** the **configuration**, **deployment**, **maintenance** of distributed storage software

- This way, you **don’t need to worry** about the **difficulties of setting up storage systems**

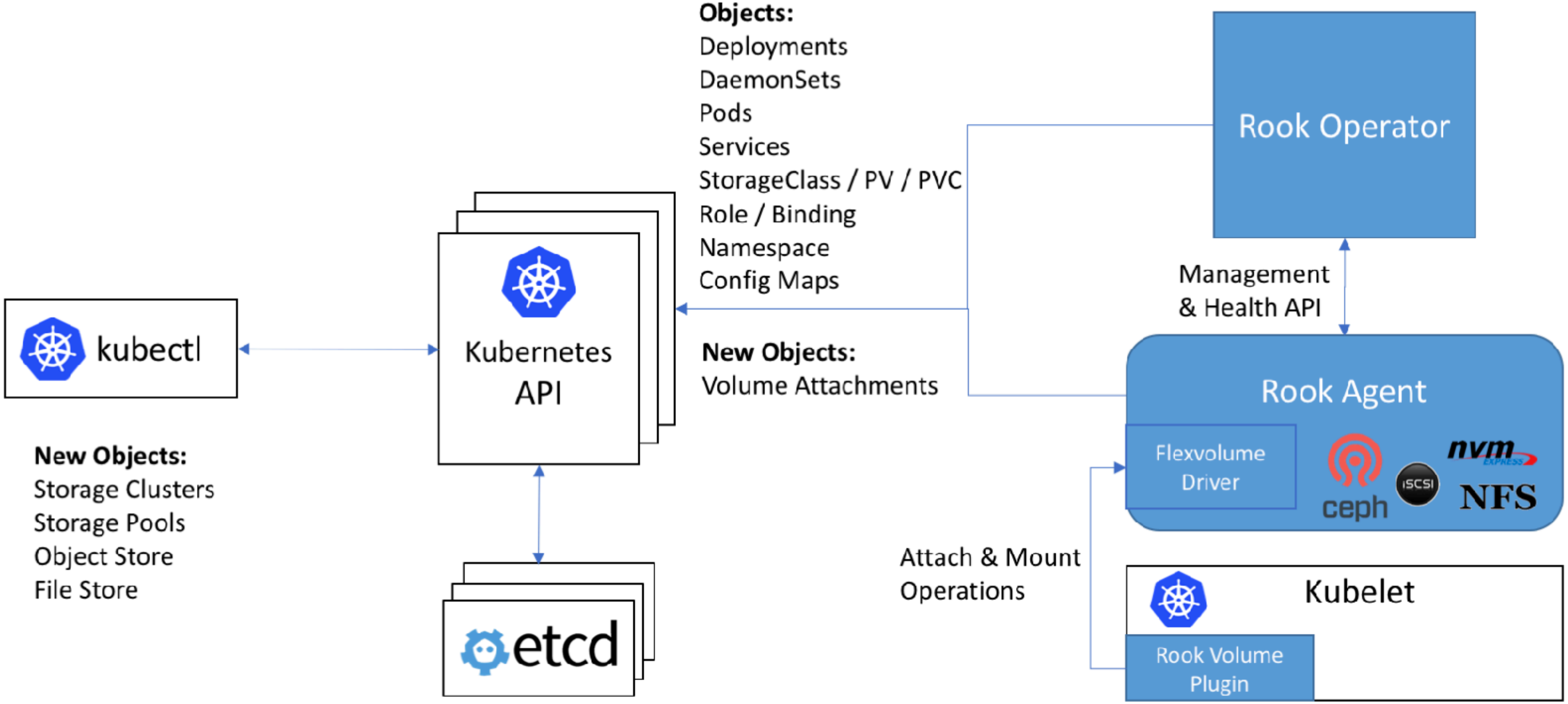
  - Rook will **orchestrate** all this management for you

- Rook is currently (early 2018) in alpha, but Rook already looks very promising and will sure be stable at some point soon

- Currently Rook uses **Ceph** as underlying storage, but **Minio** and **CockroachDB** are also available

  - **More storage engines** will be added in future releases

Rook



Source: https://rook.io/docs/rook/master/

Ceph

Ceph

- Ceph is software that provides **object, file, and block storage**
- It’s **open source**
- It’s **distributed without a single point of failure**
- Ceph **replicates** its data to make it **fault tolerant** (a node can fail, and you still have your data available)
- It’s **self-healing** and **self-managing**
- Scalable to exabyte level

Ceph

- Ceph provides **3 different types of storage**:
  - **File Storage:** to **store files and directories**, similar to accessing files over Networking File System (NFS), or using a Network Attached Storage (NAS), or EFS (Elastic File System) on AWS
  - **Block Storage:** like a hard drive, to store data using a filesystem. A database needs block storage. Examples are a SAN (Storage Area Network, which can provide block storage to servers), or EBS (Elastic Block Storage) on AWS
    - Typical use case is to store files for your OS, storage for databases, etc
  - **Object Storage:** To store any type of data as an **object**, identified by a **key**, with the possibility to add **metadata**. This type of storage lends itself to be **distributed** and **scalable**. For example, AWS S3 provides Object Storage
    - Can be used to store **unstructured data** like pictures, website assets, videos, log files, etc

Ceph Components

- Ceph has multiple components:
  - **Ceph Monitor** (min 3): **maintains a map of the cluster state** for the other ceph components (the daemons) to communicate. Also responsible for **authentication** between daemons and clients
  - **Ceph Manager daemon**: responsible to **keep track of runtime metrics** and the cluster state
  - **Ceph OSDs** (Object Storage Daemon, min 3): **stores the data**, is responsible for **replication**, **recovery**, **rebalancing** and provides information for the monitoring daemons
  - MDSs (**Ceph Metadata Server**): stores metadata for the **Ceph FileSystem** storage type (not for block/object storage types)
- Ceph stores data as **objects** within **logical storage pools**
- It uses the **CRUSH algorithm** (Controlled Replication Under Scalable Hashing), which allows Ceph to be scalable

Ceph Components

- Object storage in Ceph is provided by the **distributed object storage mechanism within Ceph**

- Ceph software libraries (librados) provides clients access to the “Reliable Autonomic Distributed Object Store” (RADOS)

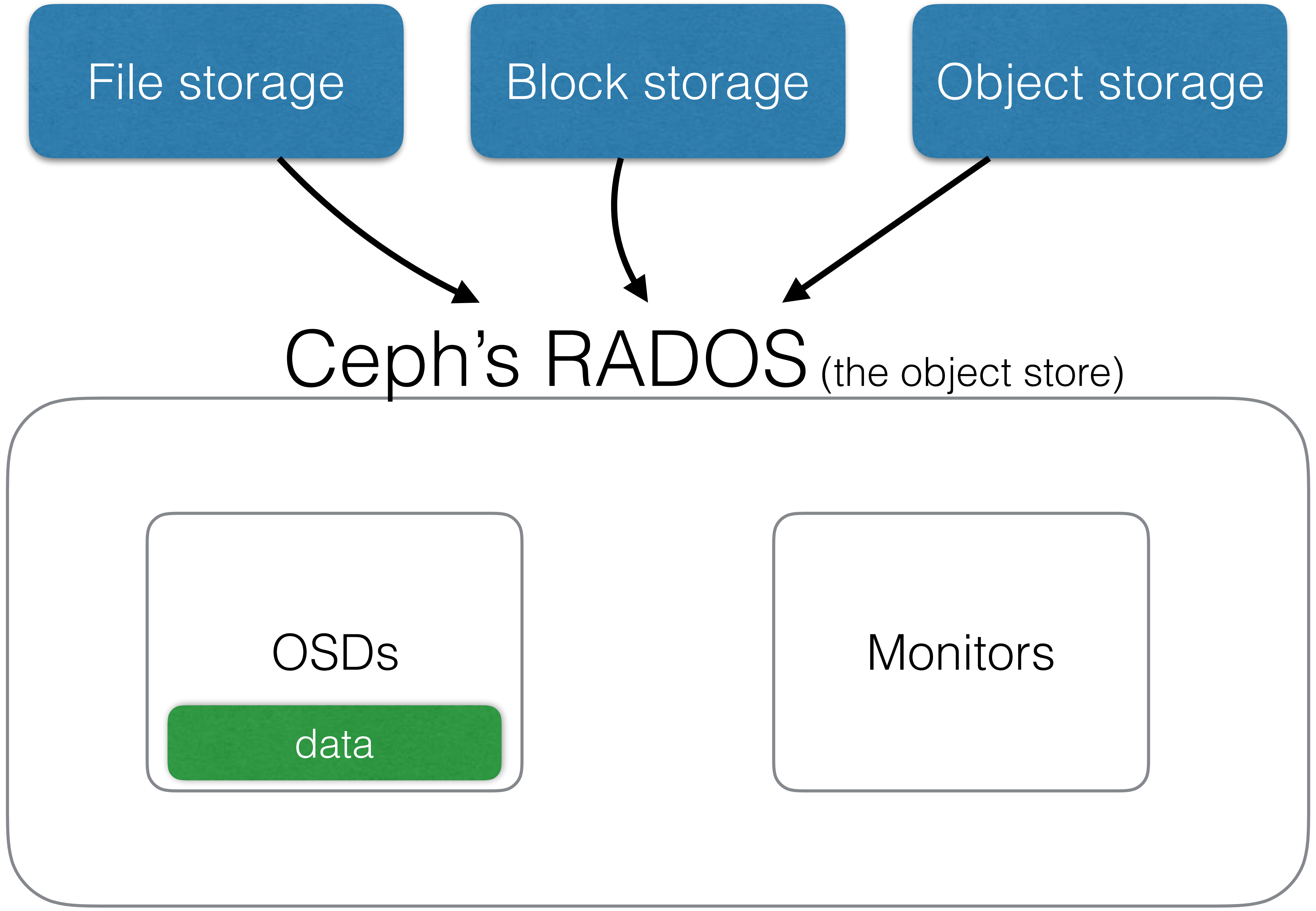
  - RADOS provides a **reliable, autonomous, distributed object store** comprised of **self-healing, self-managing, intelligent storage** nodes
    (definition source: http://docs.ceph.com/docs/master/architecture/)

  - There is also a RESTful interface that can provide an AWS S3 compatible interface to this object store

Ceph Components

- Ceph **block storage** is provided by Ceph’s RADOS Block Device (RBD)
- RBD is built on top of the same **Ceph Object Storage**
    - Ceph stores **the block images** as **objects** in the object store
    - It’s also built on librados, the software library

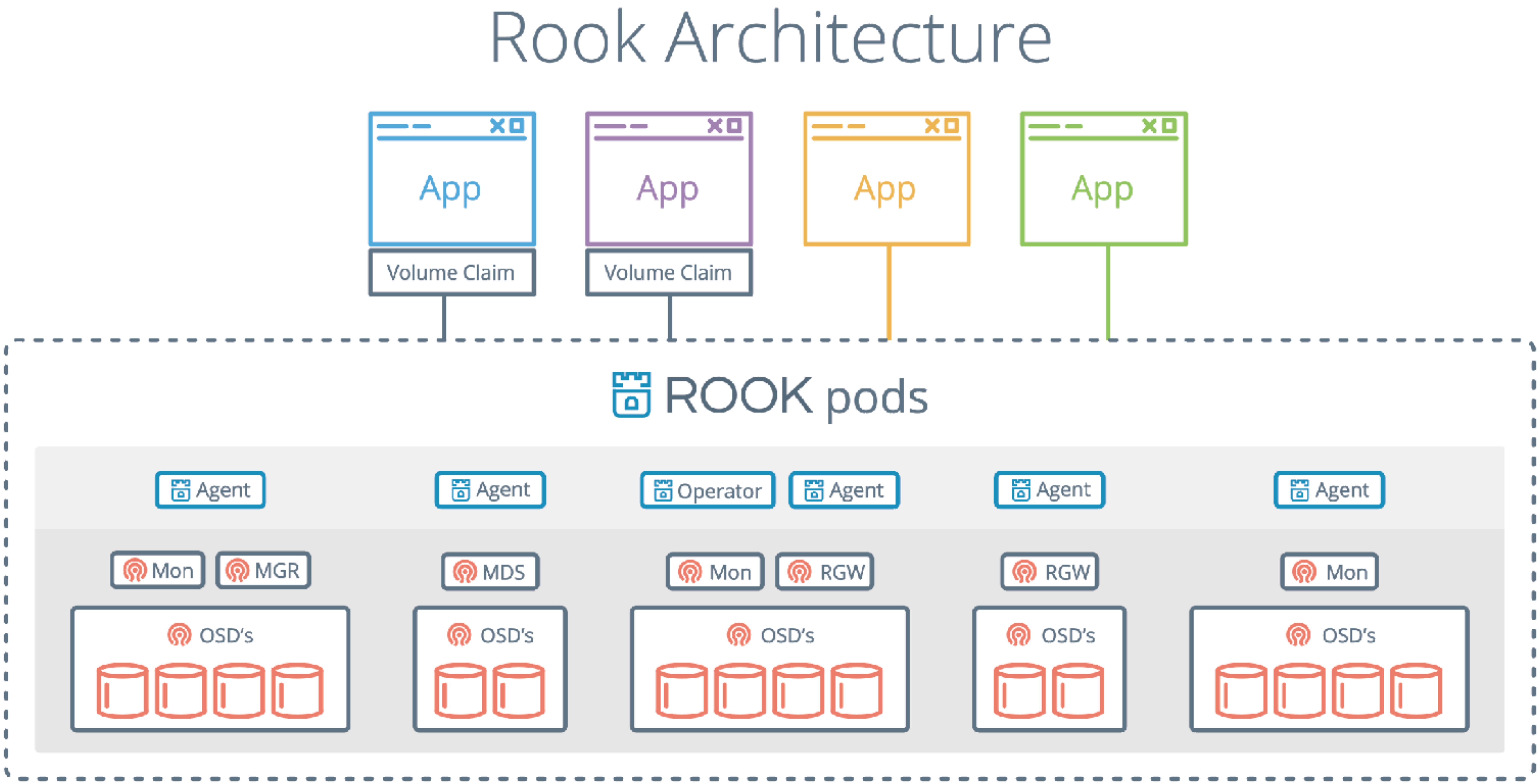
Ceph Components



- Data can come in from Ceph's **file storage**, **block storage**, or **object storage**, and Ceph will **store this data as an object in Ceph**
- Each object is stored within the Object Storage Device (OSD)
- The OSDs will run on multiple nodes
- They will handle the read/write operations to their underlying storage

Ceph with Rook

Ceph on Rook



Source: https://rook.io/docs/rook/master/

Ceph on Rook

- Rook supports all 3 types of storage: **block, file and object storage**
- I will use in the demo **Ceph** for all these types, but **other backends** are also a possibility
  - Rook will do a good job to **abstract this away from you**, so most of the configuration is nicely hidden from you
  - You ‘ll be able to use the **kubernetes yaml files** to set configuration options

Deployment steps

- First, you'll need to deploy the **rook operator**
    - Using the provided yaml files
    - Using the helm chart
- Then, you can **create the rook cluster**
    - Also using yaml definitions (this time using: apiVersion: rook.io/v1alpha1)
    - This will use the rook operator rather than the Kubernetes API
- After that, **block / file / object storage can be configured**
    - Using the rook API and Kubernetes storage API - using this storage API means using rook storage will become as easy as using for example AWS EBS or NFS

Demo

Ceph with Rook

Demo

Rook Object Storage

Demo

Rook Shared File System

cert-manager

Cert-manager

- If you want to use a **secure http connection** (https), you need to have **certificates**

- Those certificates can be **bought**, or can be **issued** by some **public cloud providers**, like AWS's Certificate Manager

- Managing SSL / TLS certificates yourself often **takes a lot of time** and are **time consuming to install and extend**

    - You also **cannot issue your own certificates** for production websites, as they are not trusted by the common internet browsers (Chrome, IE, …)

- **Cert-manager** can **ease** the **issuing** of certificates and the **management** of it

Cert-manager

- Cert-manager can use letsencrypt
- Let’s encrypt is a **free**, **automated** and **open** Certificate Authority
  (definition: https://letsencrypt.org/)
  - Let’s encrypt can issue certificates for free for your app or website
  - You’ll need to prove to *let’s encrypt* that you are the owner of a domain
  - After that, they’ll issue a certificate for you
  - The certificate is **recognized by major software vendors and browsers**

Cert-manager

- Cert-manager can **automate the verification process** for let’s encrypt

- With *Let’s encrypt* you’ll also have to **renew certificates** every **couple of months**

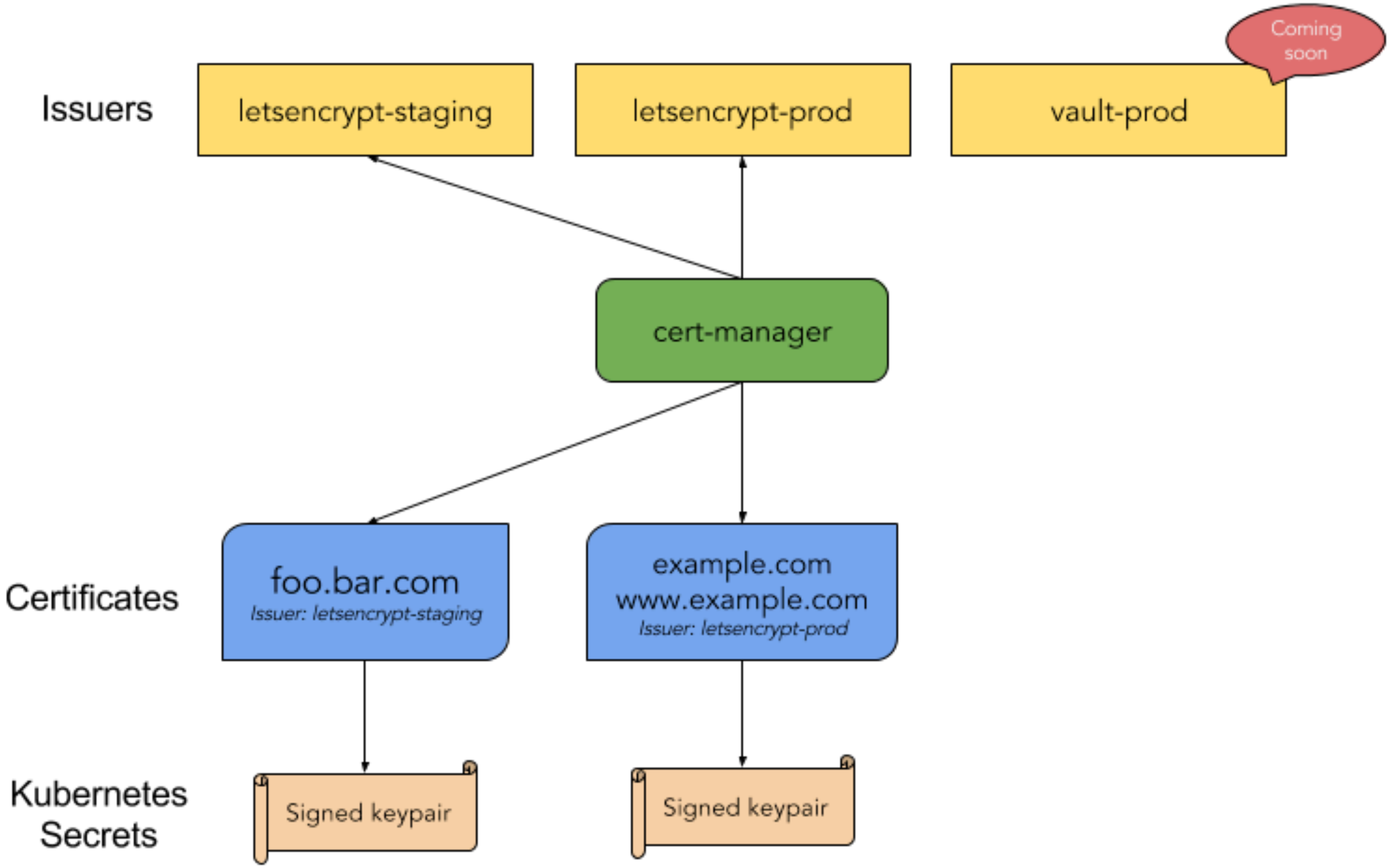
- Cert-Manager will **periodically check the validity** of the certificates and will **start the renewal process** if necessary

- *Let’s encrypt* in combination with cert-manager **takes away a lot of hassle** to deal with certificates, allowing you to **secure your endpoints** in an easy, affordable way

Cert-manager

- You can only issue certificates for a **domain name you own**
- You'll need to have a domain name like xyz.com
  - If you were using a domain name to bring up your cluster for my first Kubernetes course, the **"Complete Kubernetes Course"**, you can **re-use this domain**
  - Otherwise, you can **get one for free** from www.dot.tk or other providers
  - Or, you can **buy one** through namecheap.com / AWS route53 / any other provider that sells domain names
  - Less popular extensions only cost **a few dollars**

Cert-manager



From: https://cert-manager.readthedocs.io/en/latest/index.html

Demo

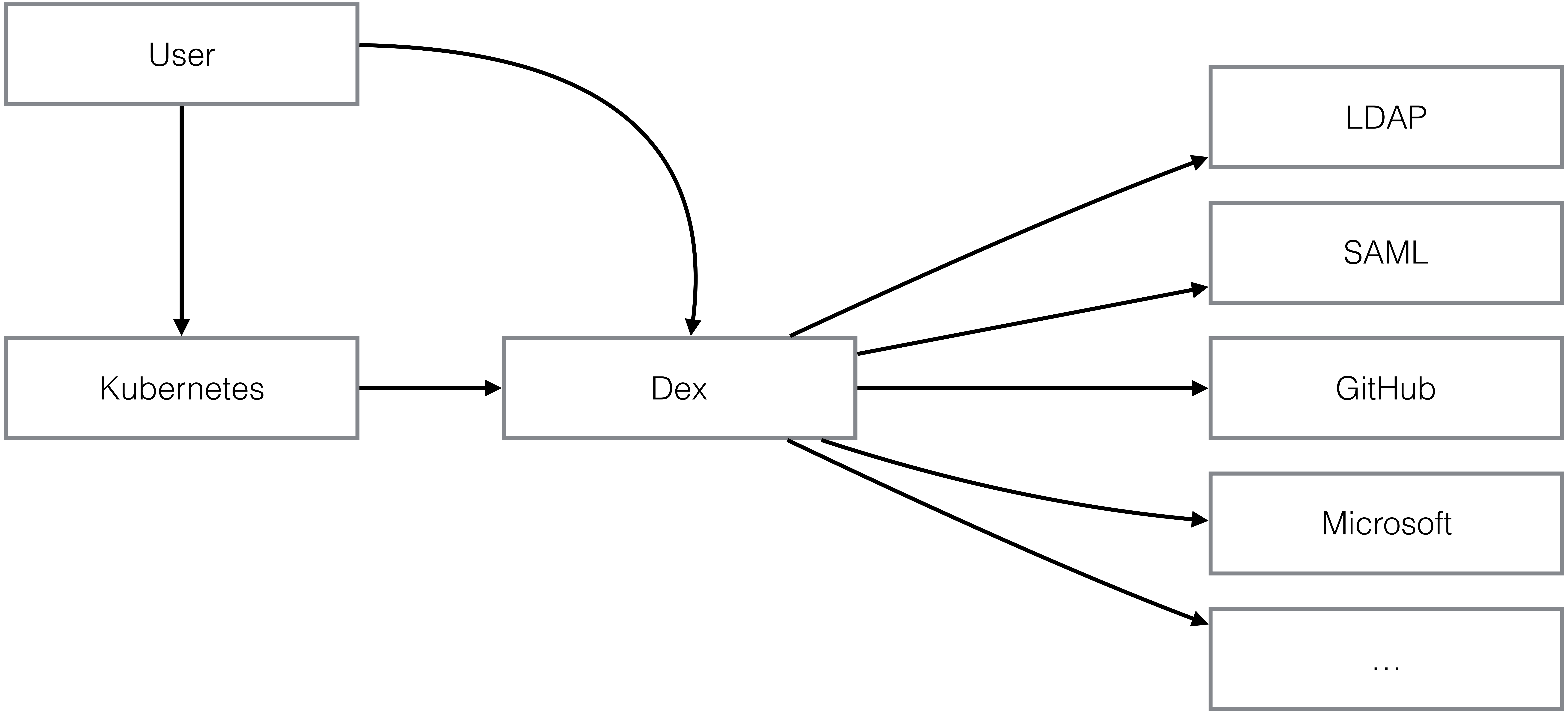
cert-manager

Dex

Dex

- Dex is **Identity Service**
  - It uses **OpenID Connect** (OIDC)
- **Kubernetes** can use **Dex** to **authenticate** its **users** (using OIDC)
- Dex uses **connectors to authenticate a user** using another Identity Provider
  - This allows you to use Dex to authenticate users in Kubernetes using LDAP, SAML, GitHub, Microsoft, and others

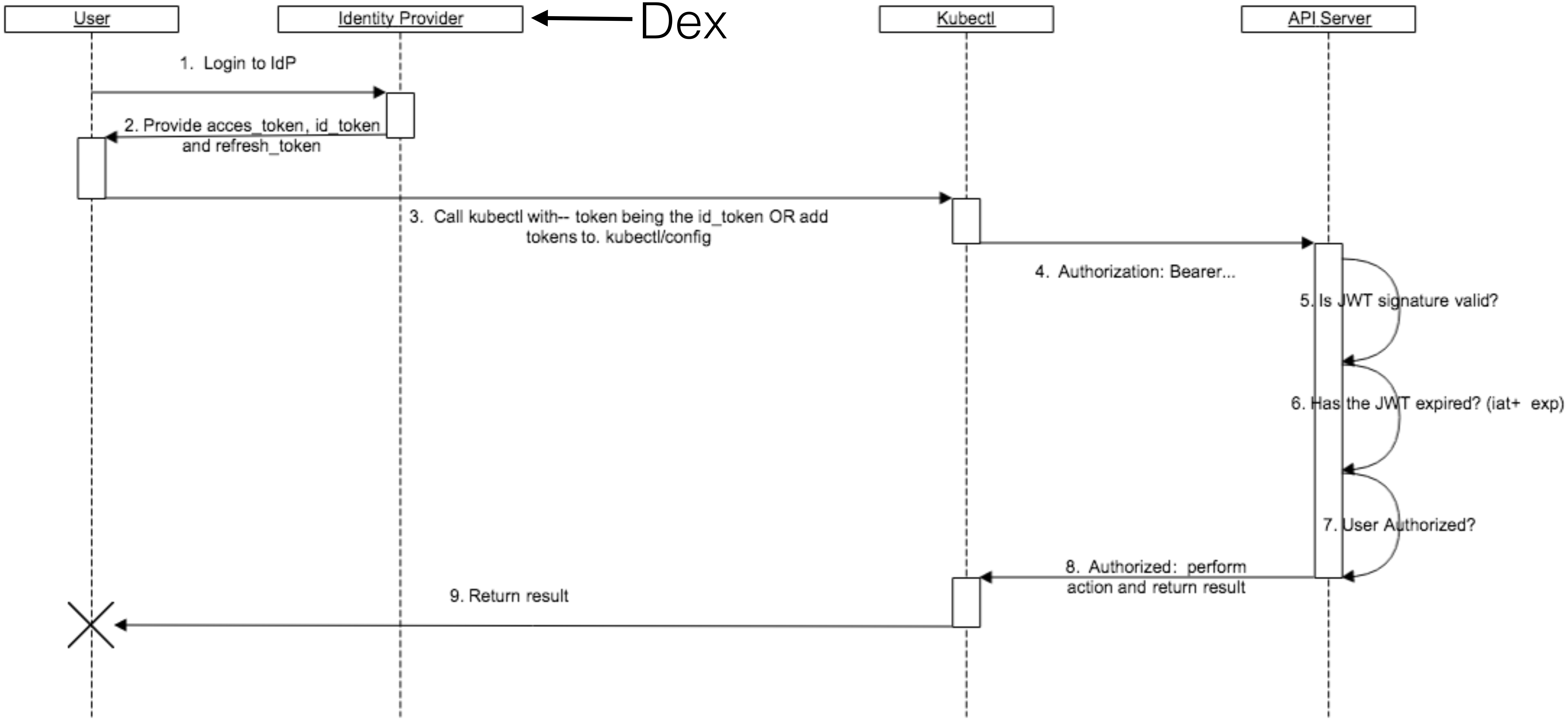
Dex Architecture



Dex

- Most companies already have a **user directory**, using **OpenLDAP**, Microsoft **Active Directory** (which is LDAP compatible), or similar products
  - LDAP stands for **Lightweight Directory Access Protocol**
- It's less common for companies to already have an OpenID Connect (OIDC) implementation that you can use
- That's why you have to use software like Dex, that will **act as a bridge** between what **enterprises offer for authentication**, and what Kubernetes can use today
- Dex can use LDAP, but there are also **other connectors** you could use if your company doesn't use LDAP

Kubernetes OIDC


User
Identity Provider
Dex
Kubectl
API Server
1. Login to IdP
2. Provide acces_token, id_token and refresh_token
3. Call kubectl with-- token being the id_token OR add tokens to. kubectl/config
4. Authorization: Bearer...
5. Is JWT signature valid?
6. Has the JWT expired? (iat+ exp)
7. User Authorized?
8. Authorized: perform action and return result
9. Return result

Source: https://kubernetes.io/docs/reference/access-authn-authz/authentication/

Demo

dex

Demo

dex - LDAP

Istio - Envoy

Envoy

- When you break up a **monolith application** (1 codebase), into **micro-services** (multiple codebases), you end up with **lots of services** that need to be able to **communicate** with each other

- These communications between services need to be able to be **fast**, **reliable and flexible**

- To be able to implement this, you need a **service mesh**

    - A **service mesh** is an **infrastructure layer** for handling these service-to-service communications

    - This is usually implemented using proxies

    - Proxies manage these communications and ensure they're **fast, reliable and flexible**

Envoy

- **Envoy** is a such a **proxy**
    - It is designed for **cloud native applications**
- Was originally built at **Lyft**
- Envoy is a **High Performance distributed proxy** written in C++
- You can see it as an **iteration** of the NGINX, HAProxy, hardware / cloud load balancers
- It's comparable with **Linkerd** (which is explained in my Advanced Kubernetes course)
    - While there's a lot of **overlap**, each solution has its **own distinct features**

Envoy Features

- **Small memory** footprint
- HTTP/2 and gRPC support
    - It’s a **transparent** HTTP/1.1 to HTTP/2 proxy
        - Not all browsers support HTTP/2 yet, so **incoming requests** can be HTTP/1.1, but **internally requests can be HTTP/2**
- **Advanced Loadbalancer Features** (automatic retries, circuit braking, rate limiting, request shadowing, zone load balancing, …)
- **Configuration** can be **dynamically managed** using an **API**
- Native support for **distributed tracing**

Comparison to linkerd

- **Linkerd** has **more features**, but that comes at a price of higher cpu and memory footprint
    - Linkerd is built on top of **Netty and Finagle** (JVM based), whereas **Envoy is written in C++**
    - If you're looking for **more features**, you might want to look at Linkerd, if you're looking for speed and low resource utilization, **Envoy wins**
        - **Istio**, discussed next, can give you the best of both worlds
- Linkerd **integrates** with **Consul and Zookeeper** for service discovery
- Envoy **supports hot reloading** using an API, Linkerd does not (by design)

Istio

Istio

- Istio is an **open platform** to **connect**, **manage**, and **secure microservices** (Definition: https://istio.io/docs/concepts/what-is-istio/overview.html)

- Key capabilities include:

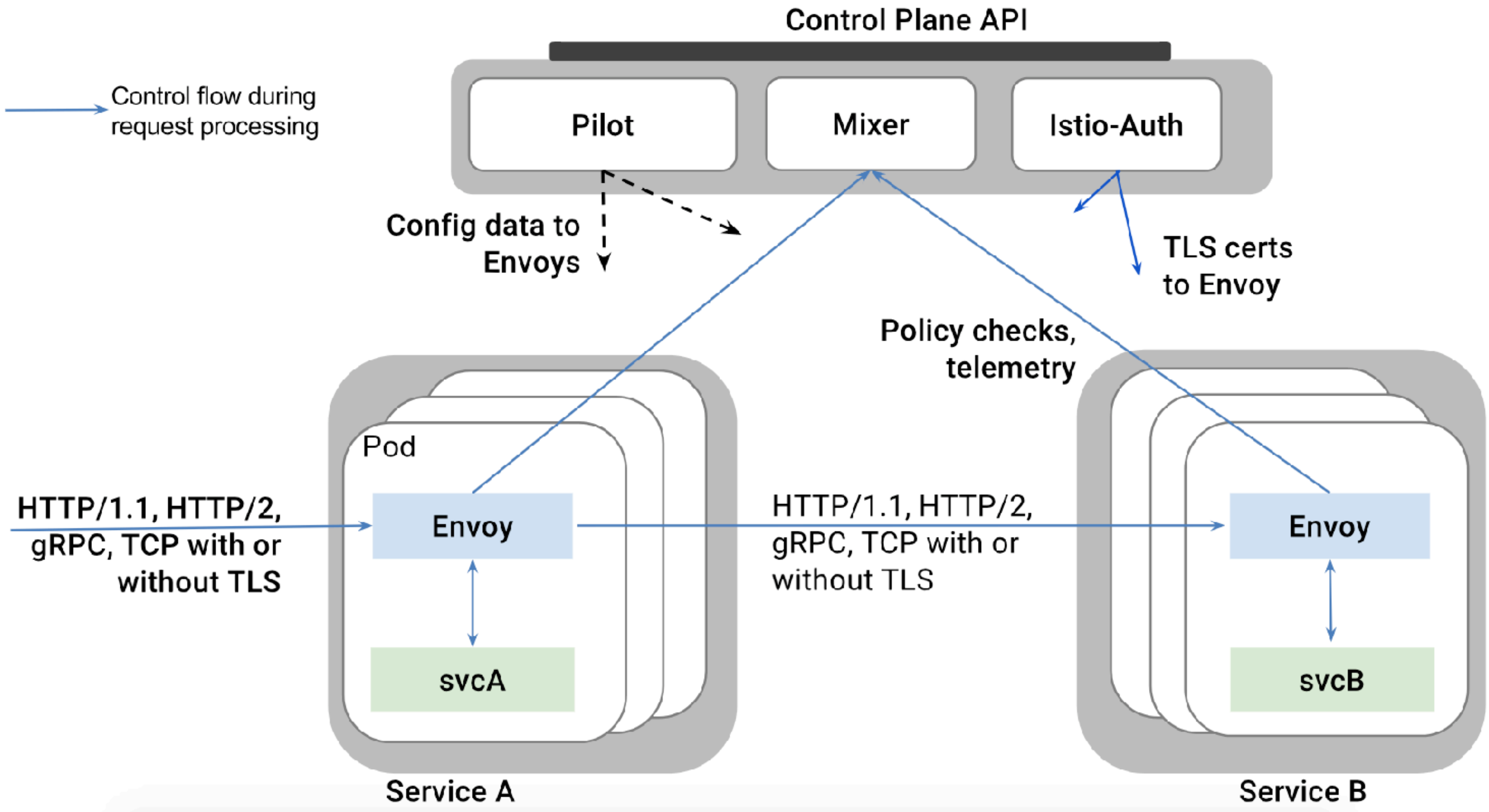
  - It supports Kubernetes

  - Can **control traffic between** services, can make it more **robust** and **reliable**

  - Can show you **dependencies** and the **flow** between services

  - Provides **access policies** and **authentication** within your service mesh

Istio Architecture



from: https://istio.io/docs/concepts/what-is-istio/overview.html)

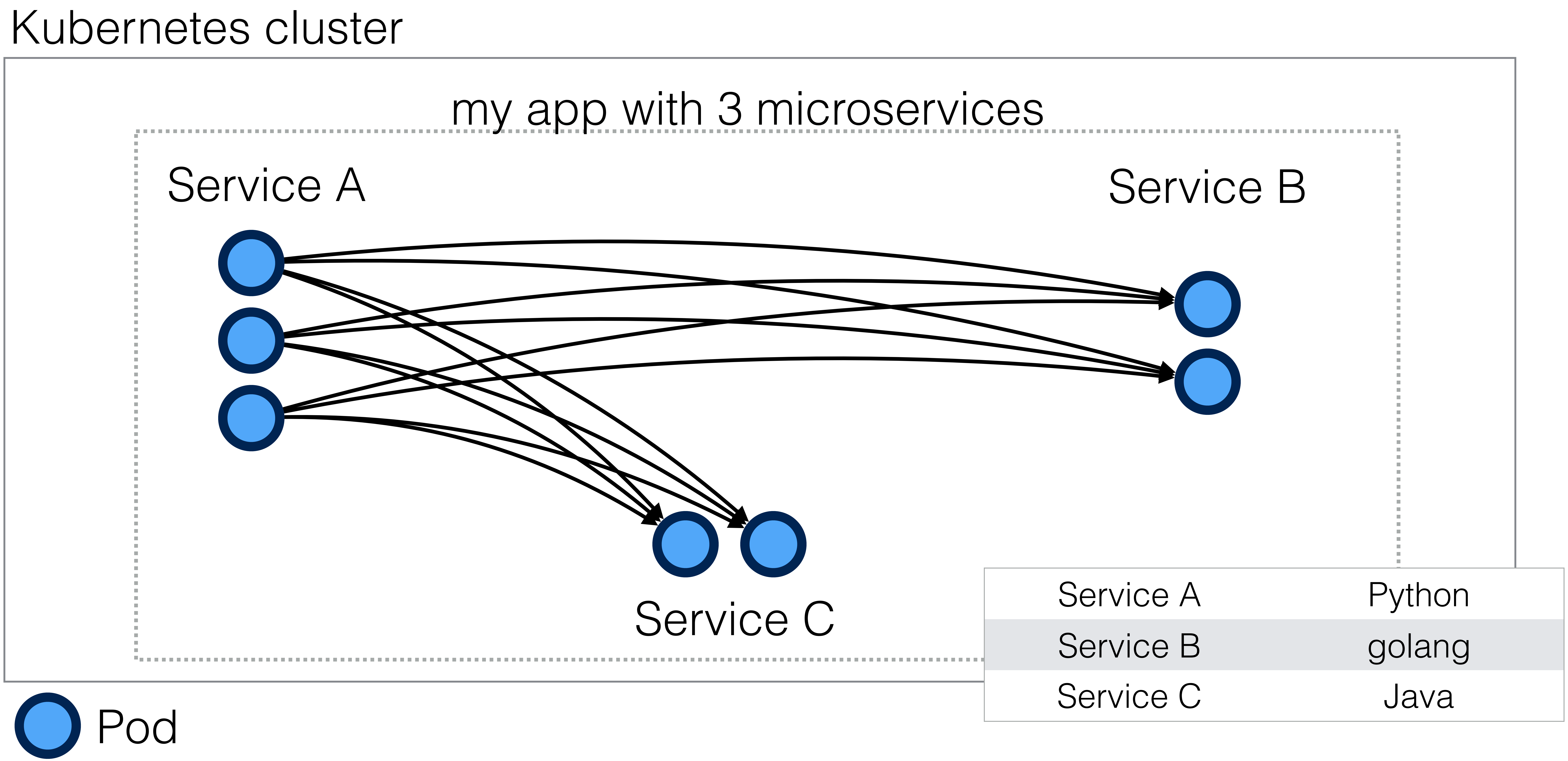
Istio Components

- Envoy (data plane)
  - Istio uses the **Envoy proxy** in its **data plane**
  - It uses a **sidecar deployment**, which means a deployment along the application (a one to one relation between app/pod and proxy)
- Mixer (control plane)
  - Responsible for enforcing **access control** and **usage policies**
  - Collects **telemetry** data from Envoy

Istio Components

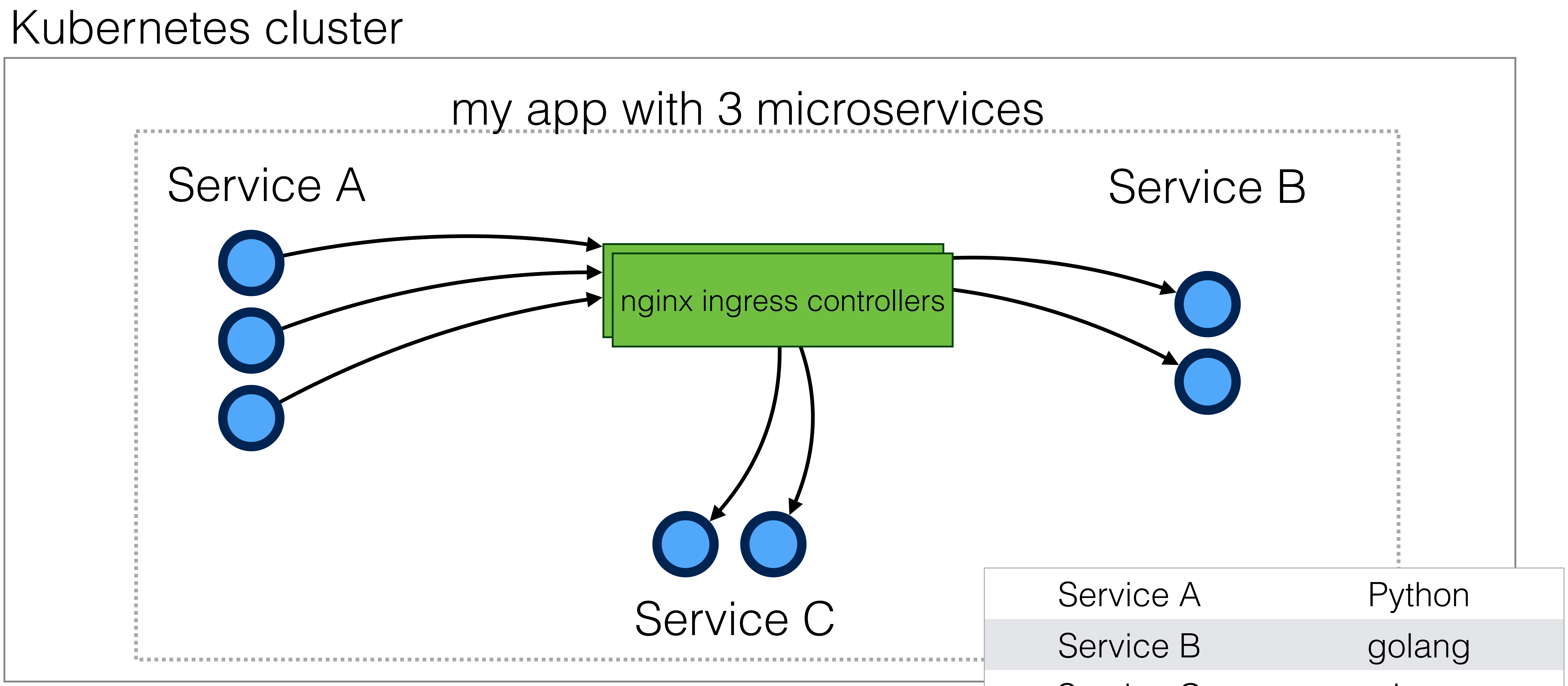
- Pilot (control plane)
    - Responsible for **service discovery**, **traffic management** and **resiliency**
        - A/B tests and canary deployments
        - Timeouts, retries, circuit brakers
    - It does this by converting Istio rules to Envoy configurations
- Istio Auth (control plane)
    - Service-to-service and end-user **authentication** using **mutual TLS**

Service Mesh



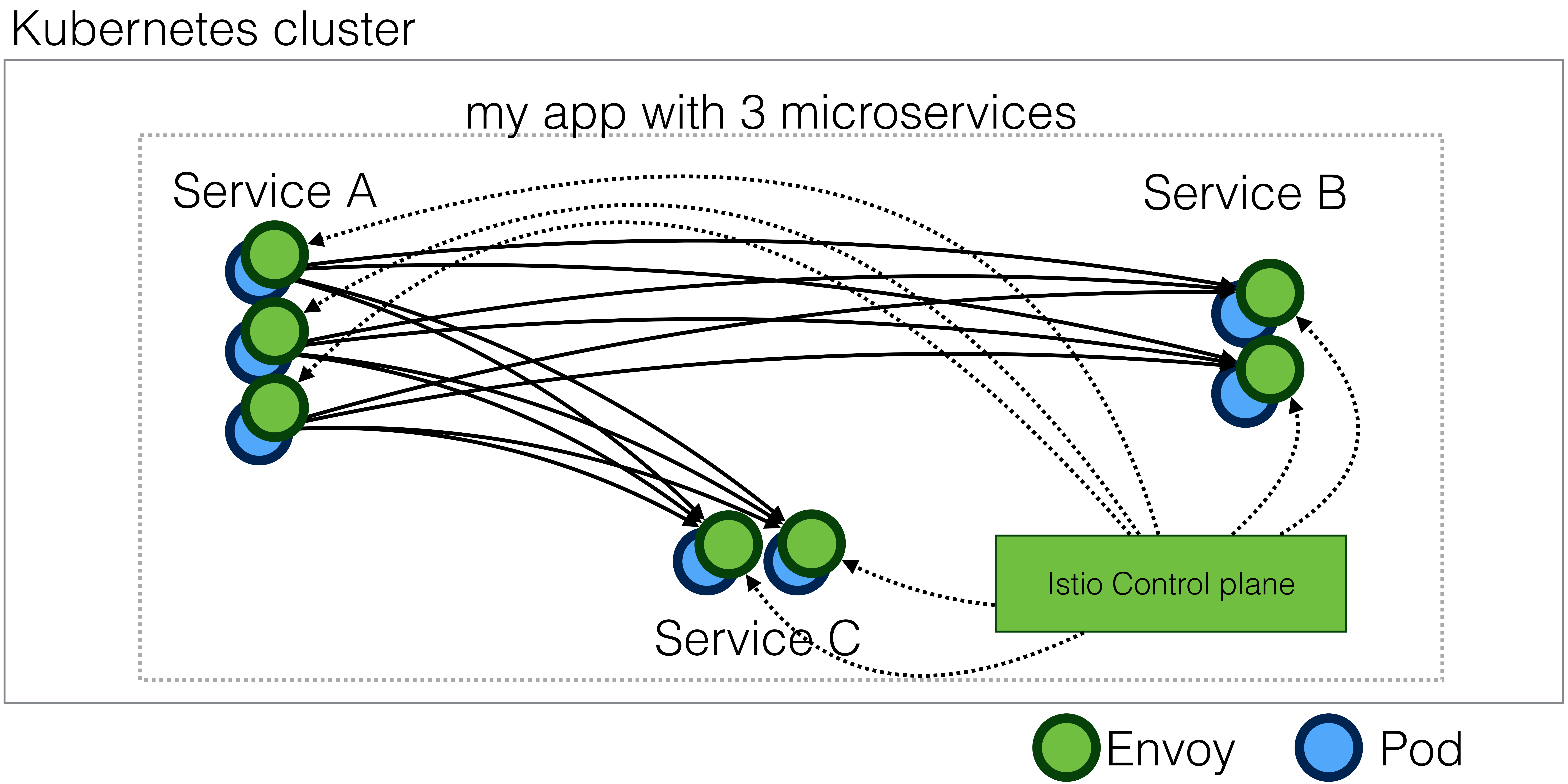
Service A	Python
Service B	golang
Service C	Java

Service Mesh - ingress



Service A	Python
Service B	golang
Service C	Java

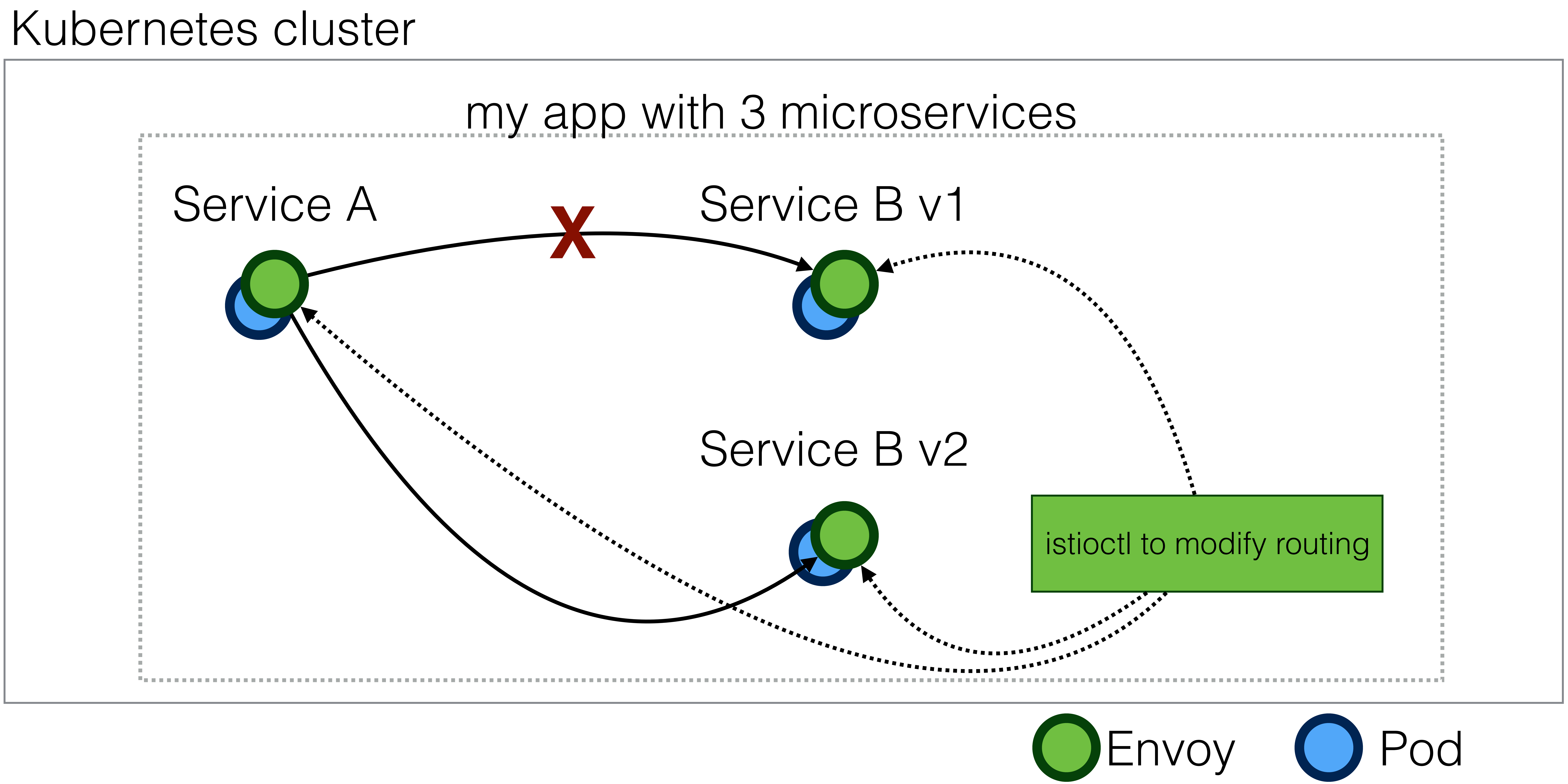
Service Mesh - istio



Demo

istio demo

traffic management



Demo

istio demo - traffic management

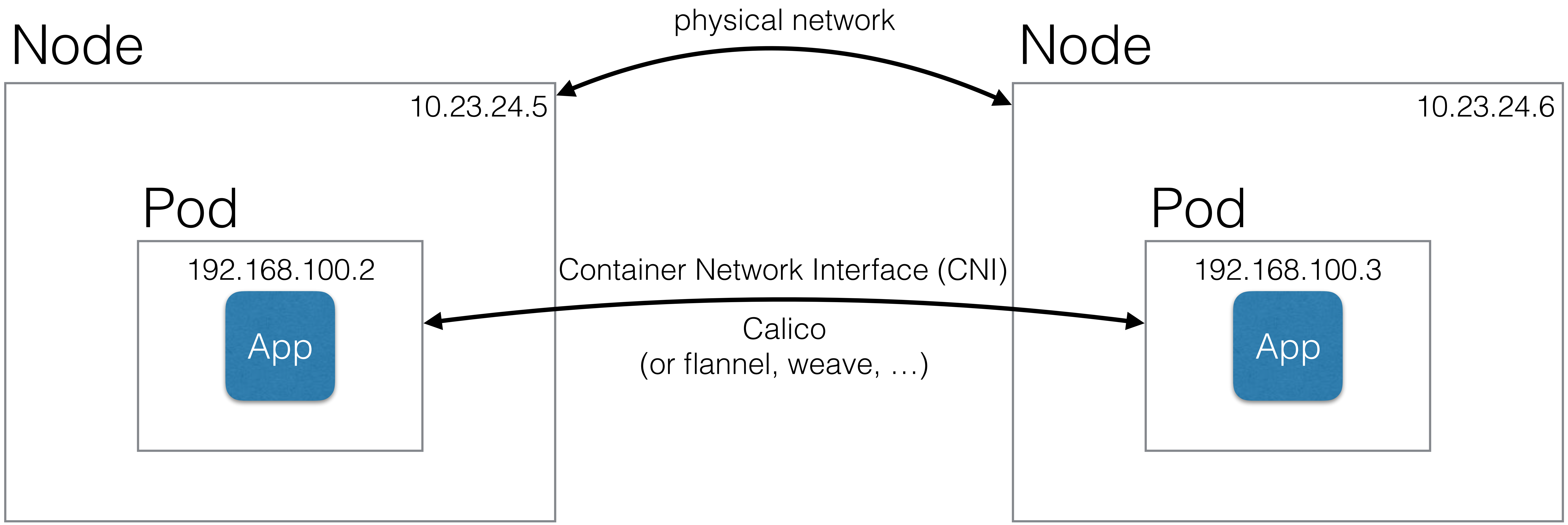
Demo

istio demo - distributed tracing

Calico

Calico

- Calico provides **secure network connectivity** for **containers** and virtual machine workloads. (Definition: https://docs.projectcalico.org/v3.1/introduction/)



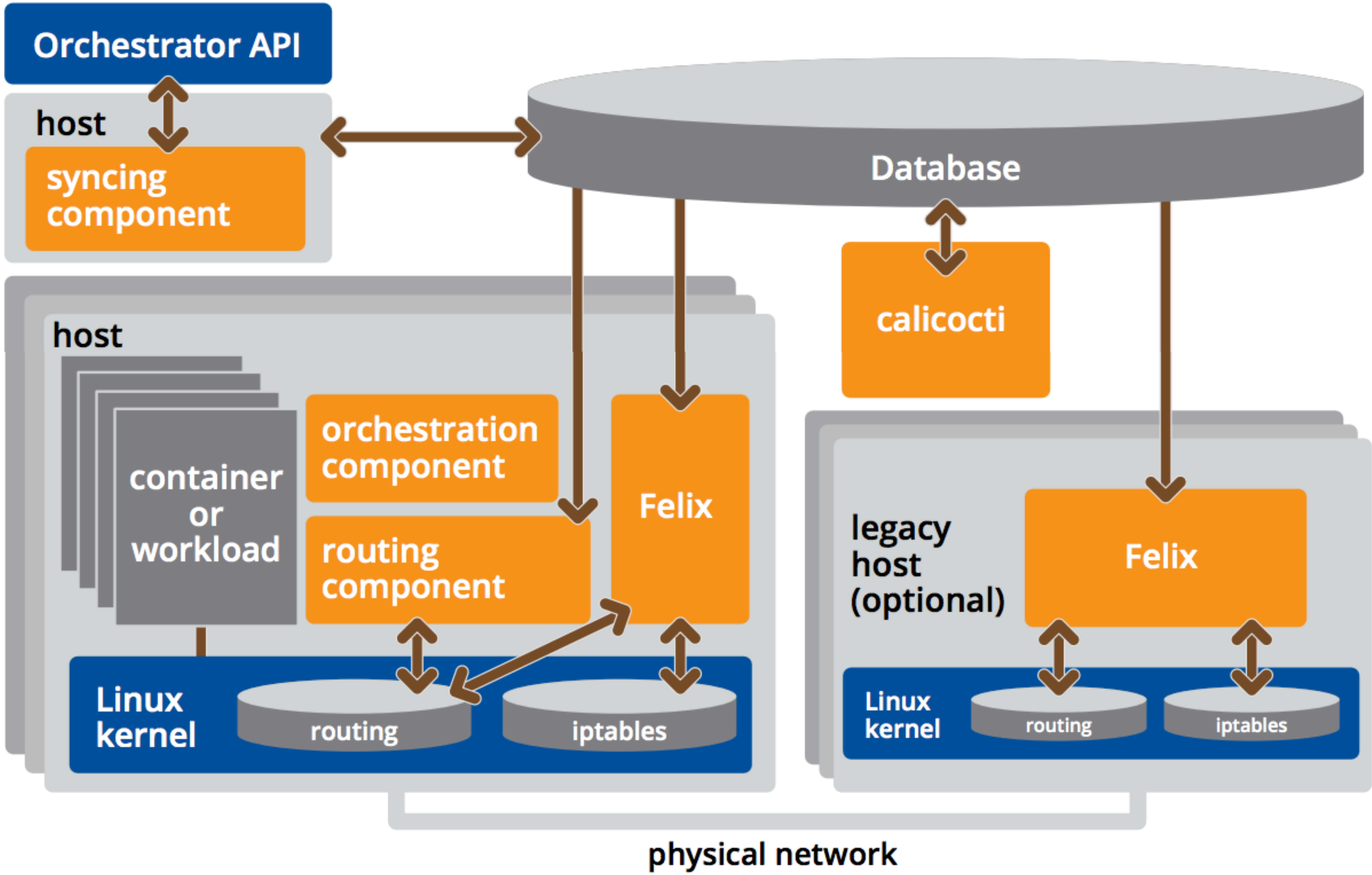
Calico

- Calico is a **Software Defined Network**, with a simplified model, with cloud-native in mind
- Calico creates a flat **Layer 3 network** using **BGP** (Border Gateway Protocol) as **routing mechanism**
  - BGP is also used as the “internet routing protocol” to route between providers (it’s a proven, scalable technology)
- **Policy driven network security** using the **Kubernetes Network Policy API**
  - Fine-grain control over the network, using the same Kubernetes API (using yaml files) as you’re used to
- **Only use overlay if necessary**, reducing overhead and increasing performance
  - An overlay network does **IP encapsulation**, but often those IP packets can be **routed without adding those extra headers** to IP packets

Calico

- Works with **Kubernetes**, but also with **OpenStack**, **Mesos**, and others
- Uses **etcd as backend** (Kubernetes also uses etcd - a distributed key value store using Raft consensus)
- Works on **major cloud providers** like AWS, GCE (also Kubernetes Engine), Azure (ACS)
    - Will also support the hosted kubernetes services AWS EKS and Azure AKS when they'll be GA
- Works well within **enterprise environments**
    - Either **without overlay**
    - With **IP-in-IP tunneling**
    - Or using an **overlay** (VxLAN) network like **Flannel**

Calico



Source: https://www.projectcalico.org/wp-content/uploads/2018/01/ProjectCalico.v3.datasheet.pdf

Calico

- Calicoctl
    - Allows you to **manage the Calico network** and **security policy**
- Felix
    - **Daemon** that runs on every machine (calico-node **DaemonSet**)
    - Responsible for
        - **programming routes and ACL** on the nodes itself
        - **Interface management** (interacts with kernel - think about MAC address / IP level configuration)
    - Reports on **health** and **state** of the network

Calico

- **BGP Client** (BIRD)
    - Runs **next to Felix** (still within the calico-node DaemonSet)
    - **Reads routing state** that Felix programmed and **distributes this information to other nodes**
        - Basically that's what BGP needs to do, it needs to make the **other nodes aware of routing information** to ensure **traffic** is **efficiently routed**
- **BGP Route Reflector**
    - All BGP clients are connected to each other, which may become a limiting factor
    - In larger deployments, a BGP route reflector might be setup, which acts as a **central point where BGP clients connect to** (instead of having a mesh topology)

Calico

- Once Calico is setup, you can **create a network policy in Kubernetes**

- You can first create a **network policy to deny all access to all pods** (then afterwards you can open the ports that are needed):

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-all
  namespace: apps
spec:
  podSelector:
    matchLabels: {}
```

- At this point the pods are isolated, you’ll not be able to connect from one pod to another anymore

Calico

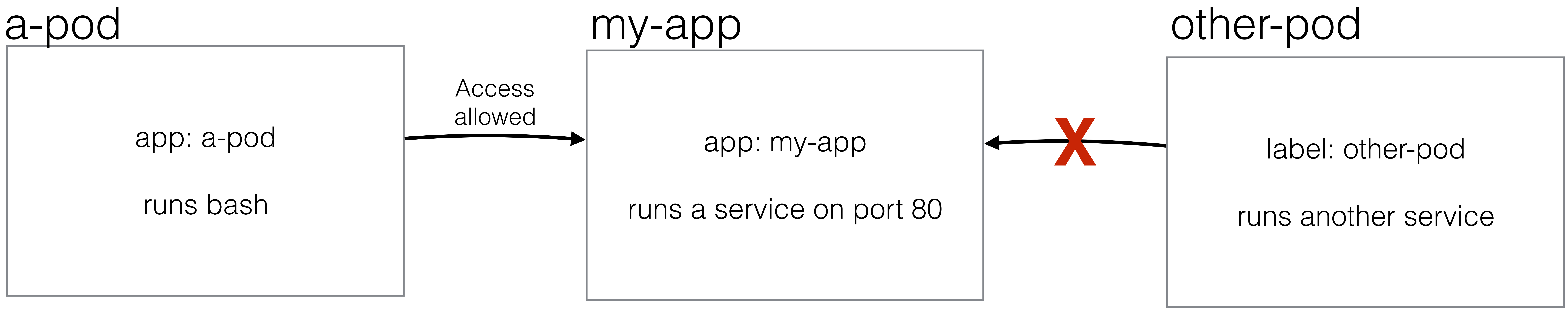
- **Isolated vs non-isolated**
  - By default pods are **non-isolated**
    - Pods **accept traffic from any source**
  - By having a **network policy** with a **selector** that selects them (the previous one selects all pods), network access is denied by default
    - The pod now becomes **isolated**
    - Only connections that are defined in the **network policy** are allowed
  - This is on a namespace basis

Calico

- You can now add a new rule to enable network access to a pod:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-my-app
  namespace: apps
spec:
  podSelector:
    matchLabels:
      app: my-app
  ingress:
    - from:
      - podSelector:
          matchLabels:
            app: a-pod
```

Calico



a-pod

app: a-pod

runs bash

Access allowed

my-app

app: my-app

runs a service on port 80

other-pod

label: other-pod

runs another service

```
spec:
  […]
  ingress:
    - from:
      - podSelector:
          matchLabels:
            app: a-pod
```

```
spec:
  podSelector:
    matchLabels:
      app: my-app
  […]
```

Demo

Calico example

Demo

Calico egress example

Vault

Managing credentials in a distributed environment

Vault

- Vault is a **tool for managing secrets**
    - For example: passwords, API keys, SSH keys, certificates
- It’s **opensource** and released by **HashiCorp** (like Vagrant, terraform, and other well known tools)
- Some use cases are:
    - General Secret Storage
    - Employee Credential Storage (Sharing credentials, but using audit log, with ability to roll over credentials)
    - API key generation for scripts (Dynamic Secrets)
    - Data Encryption / Decryption

Vault Features

- **Secure Secret Storage**
  - **Encrypted key-value pairs** can be stored in Vault
- **Dynamic Secrets**
  - Vault can create **on-demand secrets** and **revoke them after a period of time** (when the client lease is up)
  - For example **AWS credentials** to access an S3 bucket
- **Data Encryption**
  - Vault can encrypt / decrypt data without storing it

Vault Features

- **Leasing and Renewal**
  - Secrets in Vault have a **lease** (a time to live)
  - When the lease is up, the **secret** will be **revoked** (deleted)
  - Clients can ask for a **renewal** (a new secret) using an API
- **Revocation**
  - Easy **revocation features**
  - For example, all secrets of a **particular user** can be removed

Vault Operator

- In April 2018 CoreOS released the Vault Operator

- It allows you to easily deploy Vault on Kubernetes

- It allows you to configure and maintain Vault using the Kubernetes API (using yaml files and kubectl)

- It gives you a **good alternative** to **secret management tools** on **public cloud** (like the AWS Secrets Manager or AWS Parameter store)

Demo

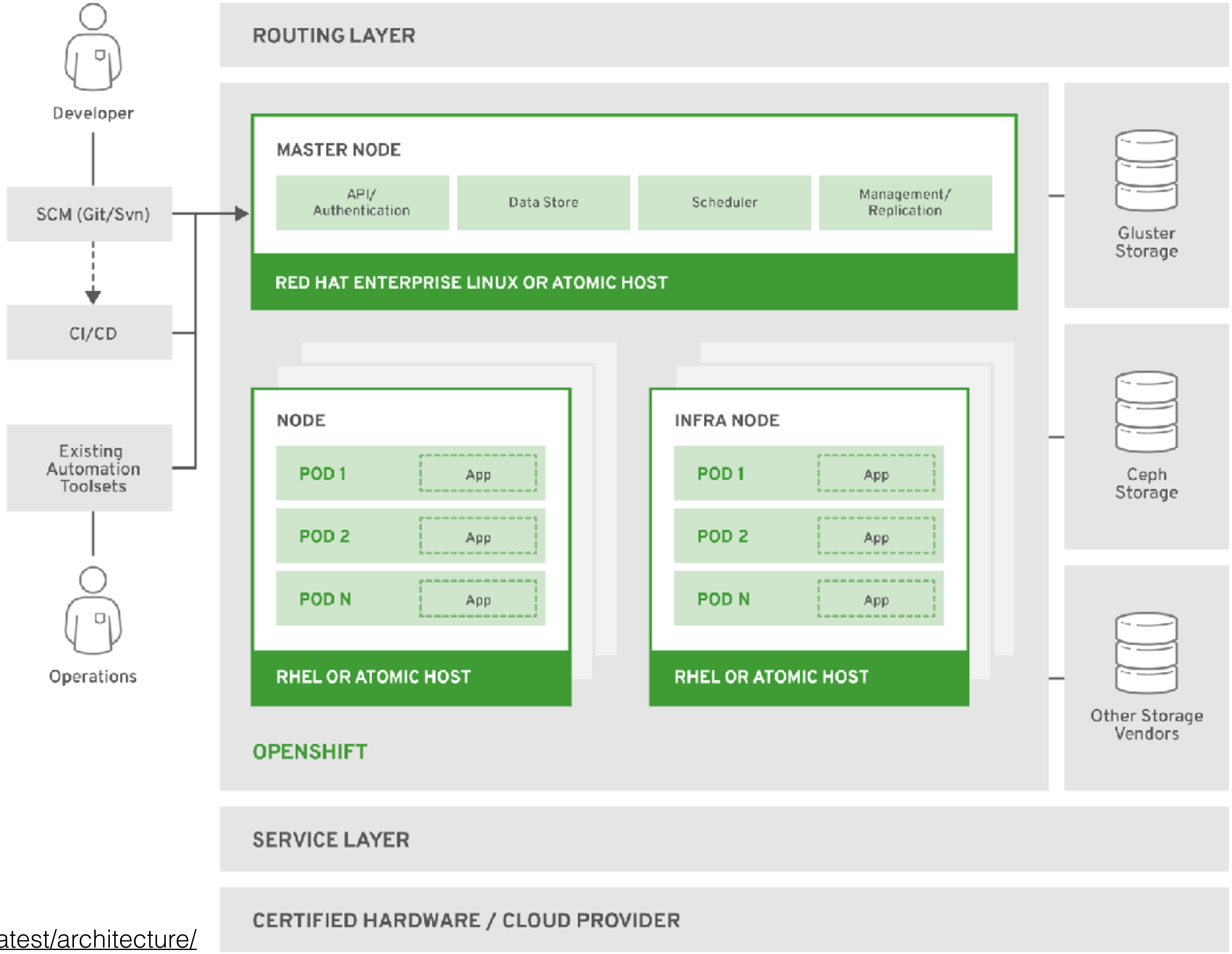
Vault

Openshift Origin

OpenShift Origin

- OpenShift Origin is a **distribution of Kubernetes**
- It is optimized for **continuous application development** and **multi-tenancy**
- It adds **developer** and **operations** centric tools
- OpenShift Origin is the **upstream community project** that powers Openshift
- It uses **Kubernetes**, **Docker**, and **Project Atomic** (a container operating system)
- Definitions: https://www.openshift.org/

OpenShift Origin Architecture



Source: https://docs.openshift.org/latest/architecture/

OpenShift Origin

- Openshift also has a quick setup by running **openshift in a container**
- With “oc cluster up” you can bring up the Kubernetes cluster with the **Openshift frontend**
- Using the web frontend, you can **create projects and applications**
  - For example, you can start a **NodeJS** project or a **MySQL** database
  - Those projects will use a **git repository** to build and push the docker image
  - Everything happens **behind the scenes**, it’s very developer focussed
- You can also **integrate with Jenkins** by putting Jenkinsfiles in your project

OpenShift Origin

- The **Developer experience is great**, because it **hides** the **complexities** of Kubernetes
- That means that Openshift has its own **implementation** of:
  - Handling **storage** (for example ceph)
  - Handling **authentication** (you plugin into Openshift)
  - Integrating **CI**, **Ingress, loadbalancing, etc**
- This can be a pro or a con (you don’t have to worry about the implementation, but you also have to follow Openshift’s way of doing things)

OpenShift Origin

- When to use Openshift?

  - If you need a **complete system** that **integrates CI/CD** and **Kubernetes** (and hosted platforms are no option)

  - If you don't want to design and develop a custom delivery platform for your developers, but are OK with using **Openshift's way of doing things**

  - If you "just want to **let your developers run apps on Kubernetes**"

  - If you **already are using Redhat**, and you'd like to get a on-prem PaaS offering with the support Redhat provides

Demo

Installing OpenShift

Demo

Running an application