Microsoft

Building the future of Distributed Systems with Kubernetes

Erik St. Martin
Sr. Cloud Developer Advocate
@erikstmartin



About Me

Go

- Using since 2011
- Co-Organizer GopherCon
- Co-Author Go in Action
- Co-Host Go Time

Kubernetes

- Using since 2014
- Contributed to Docker and Kubernetes
- Created SkyDNS (predecessor and library for kube-dns)
- Contributed heavily to Virtual Kubelet



What is Kubernetes?

- Container Orchestration Platform?

What is Kubernetes?

- Container Orchestration Platform?
- Infrastructure as a Service Platform?

What is Kubernetes?

- Container Orchestration Platform?
- Infrastructure as a Service Platform?
- Framework / Library for building Distributed Systems?

“Computers aren’t the thing… They’re the thing that gets us to the thing.”

- Joe MacMillan

(Halt and Catch Fire S1:E1)

“Kubernetes isn’t the thing…
It’s the thing that gets us to the thing.”

Customizing Kubernetes



How does Kubernetes work?

Reconciliation

Desired State



Actual State

Kubernetes Objects

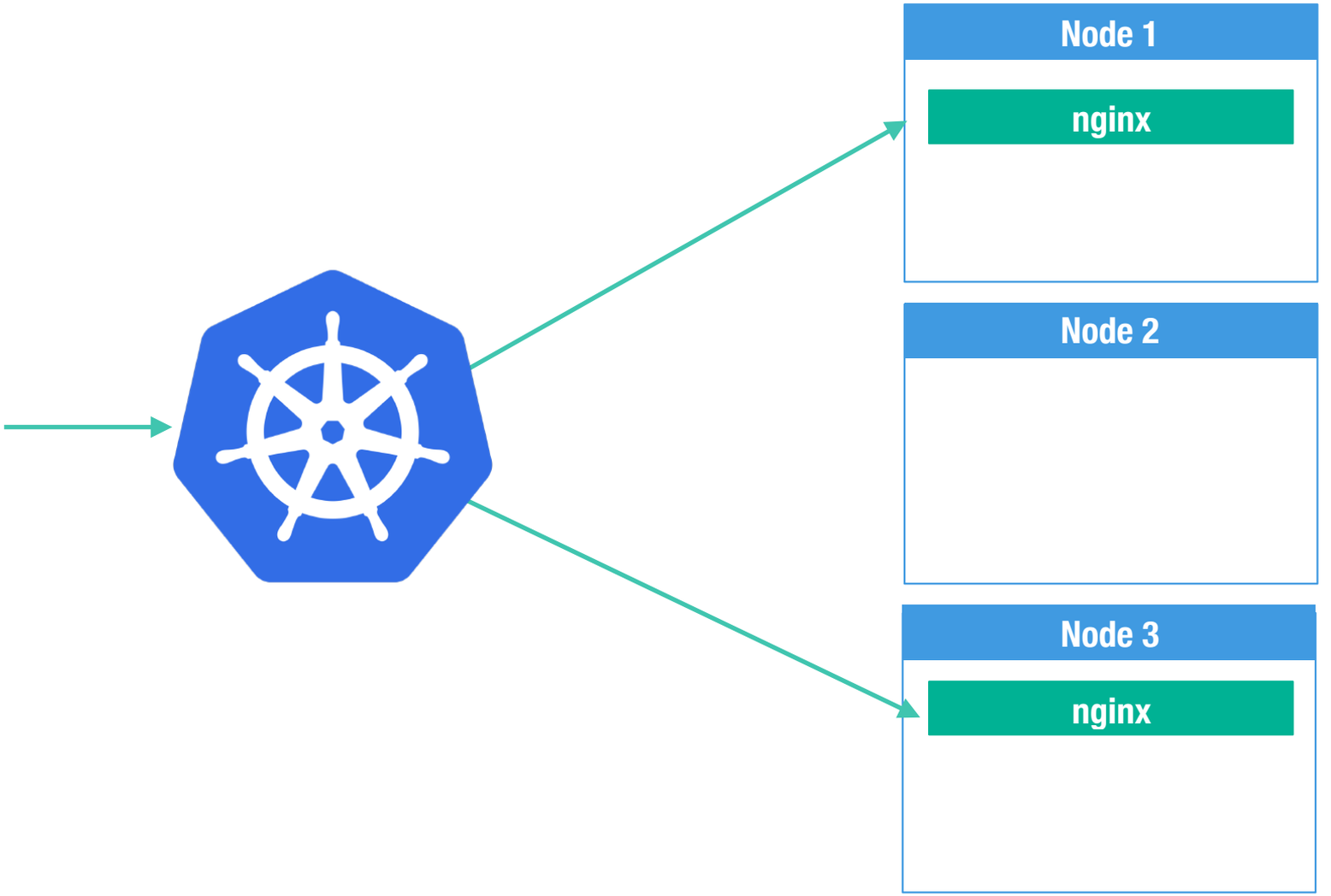
- ReplicaSet
- StatefulSet
- DaemonSet
- Job
- Deployment
- Node
- Pod
- Service
- Ingress
- Namespace
- ConfigMap
- Secret
- Volume
- PersistentVolume
- PersistentVolumeClaim

Spec Files

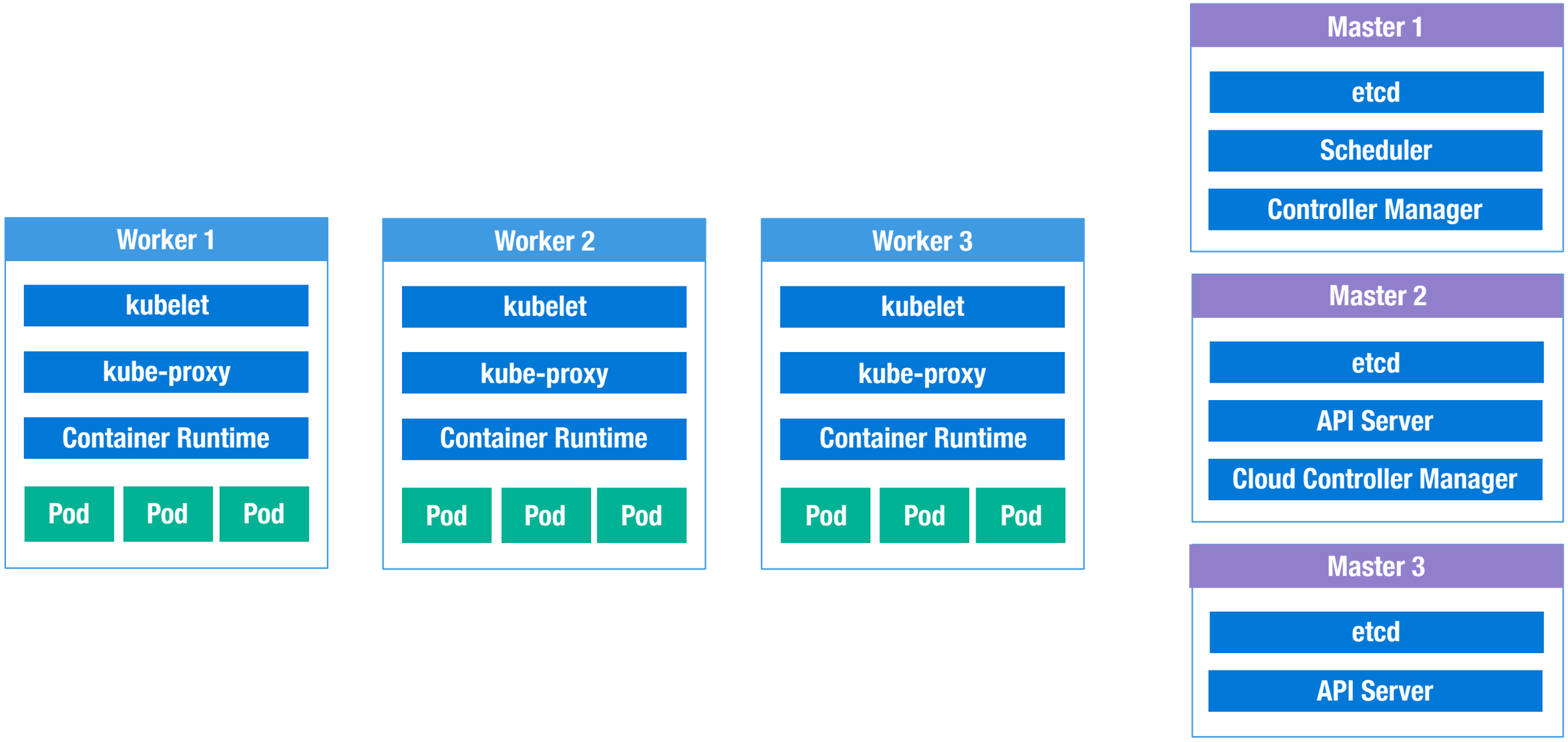
```
  apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
         - containerPort: 80
```

Reconciliation

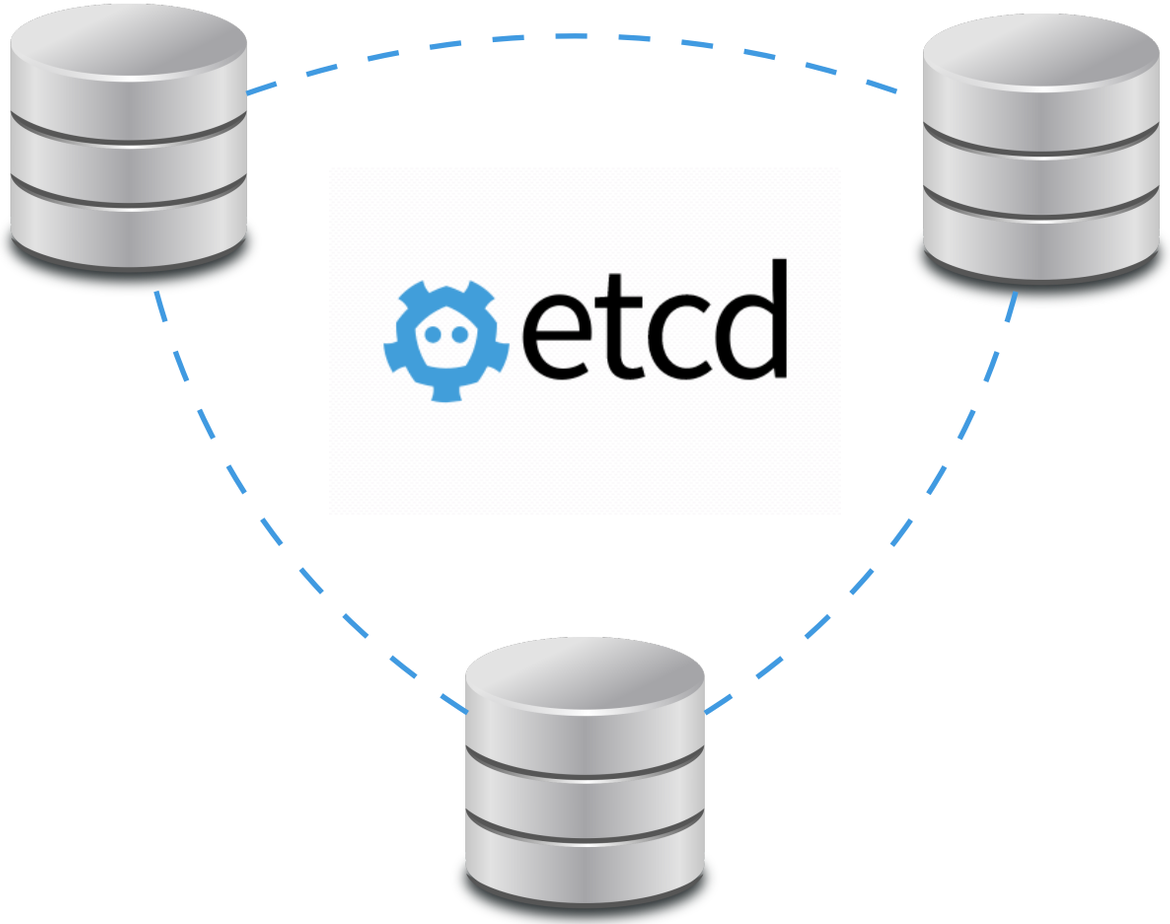
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
```



Kubernetes Control Plane

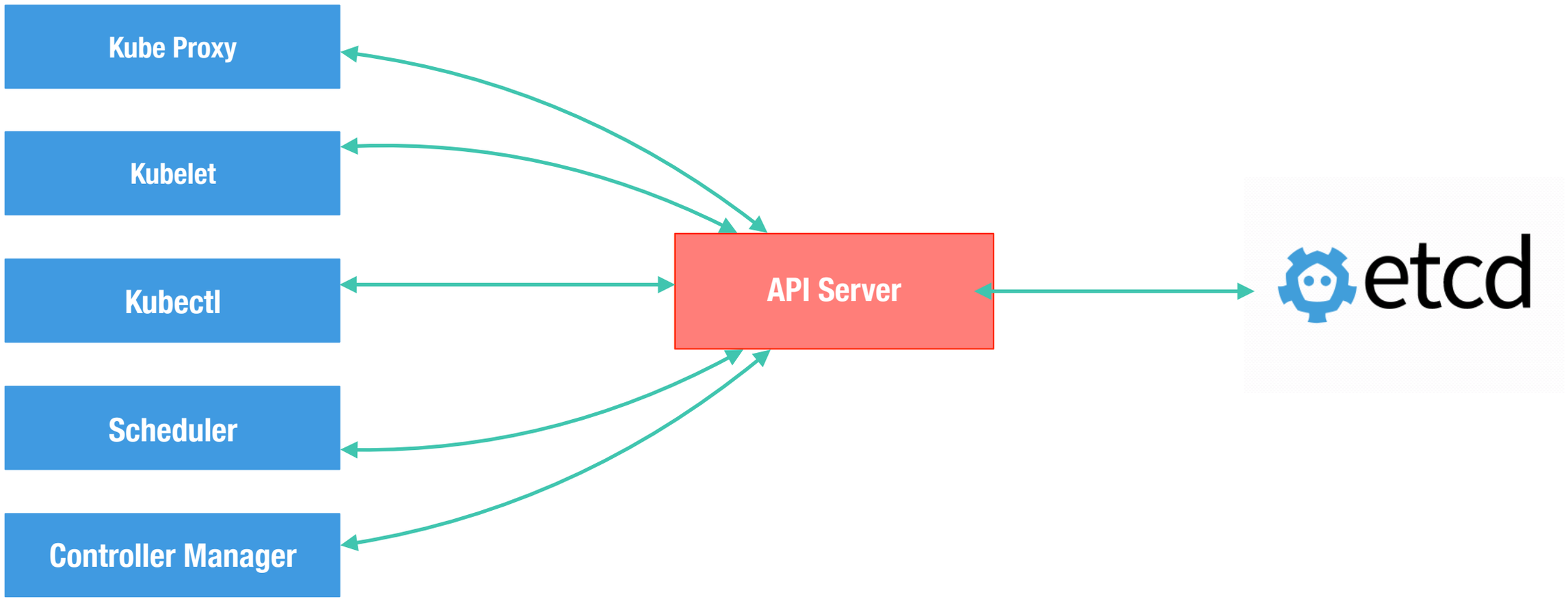


Etcd



- Distributed
- Fault Tolerant
- Versioning
- Watches
- Distributed Locks
- Leadership Election

API Server


Kube Proxy
Kubelet
Kubectl
Scheduler
Controller Manager
API Server
etcd

API Server

Why would you want to create your own?

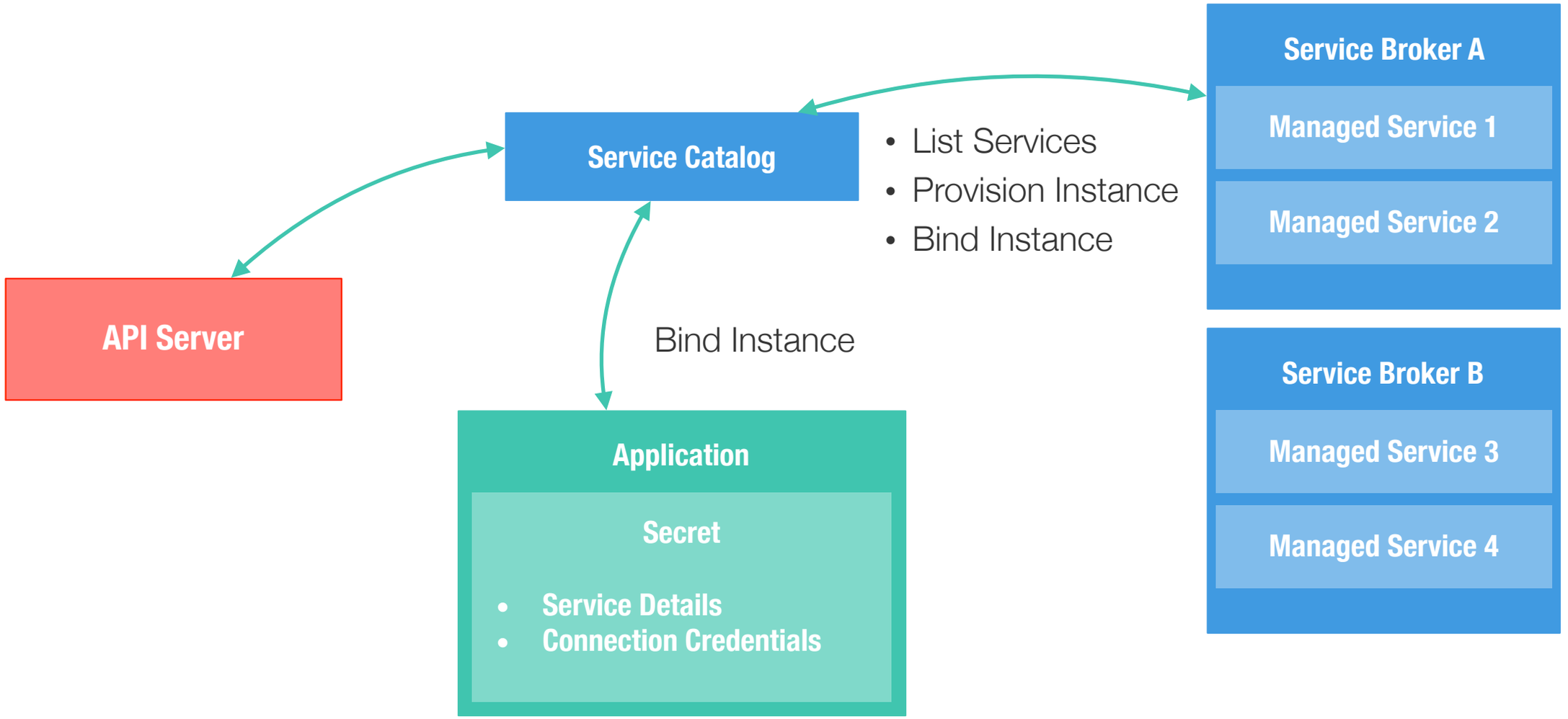
- API Server Aggregation (Service Catalog, etc)
- Custom Resource Definitions

Open Service Broker



- Connect applications to hosted services and partner services using Kubernetes API
- Provisioning, Deprovisioning, Binding, Unbinding
- Currently supports (ACI, Cosmos DB, MySQL, Postgres, Event Hubs, Key Vault, Redis, SQL Database, Search, Service Bus, Storage)

Service Catalog & Service Broker



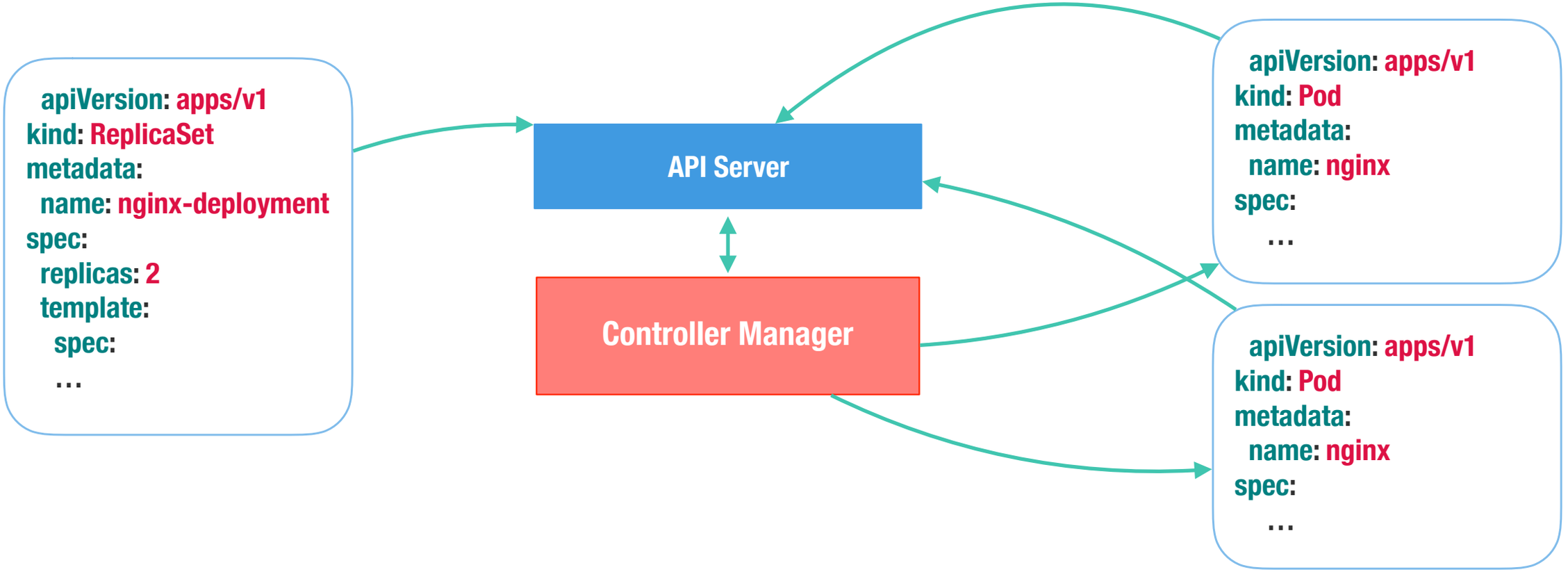
Open Service Broker (mysql instance)

```yaml
apiVersion: servicecatalog.k8s.io/v1beta1
kind: ServiceInstance
metadata:
  name: example-mysql-instance
  namespace: default
spec:
  clusterServiceClassExternalName: azure-mysql
  clusterServicePlanExternalName: basic50
  parameters:
    location: eastus
    resourceGroup: demo
    firewallRules:
    - startIPAddress: "0.0.0.0"
      endIPAddress: "255.255.255.255"
      name: "AllowAll"
```

Open Service Broker (mysql binding)

```yaml
apiVersion: servicecatalog.k8s.io/v1beta1
kind: ServiceBinding
metadata:
  name: example-mysql-binding
  namespace: default
spec:
  instanceRef:
    name: example-mysql-instance
  secretName: example-mysql-secret
```

Controller Manager



Controller Manager

Node Controller: Responsible for noticing and responding when nodes go down.

Replication Controller: Responsible for maintaining the correct number of pods for every replication controller object in the system.

Endpoints Controller: Populates the Endpoints object (that is, joins Services & Pods).

Service Account & Token Controllers: Create default accounts and API access tokens for new namespaces.

Cloud Controller Manager

Node Controller: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding

Route Controller: For setting up routes in the underlying cloud infrastructure

Service Controller: For creating, updating and deleting cloud provider load balancers

Volume Controller: For creating, attaching, and mounting volumes, and interacting with the cloud provider to orchestrate volumes

Controllers / Operator Pattern

Why would you want to create your own?

- Implement support for a different cloud provider
- Decouple applications and resource management from spec definitions
- Provide additional self-service infrastructure to your organization (databases,
- services, monitoring, etc), and can leverage things like RBAC out of the box
- Abstracting operational knowledge (Operator Pattern)

https://github.com/kubernetes/sample-controller

Kube-lego

Automatically request LetsEncrypt certificates for your services

Service Spec

```
metadata:
  annotations:
    kubernetes.io/tls-acme: “true"
```

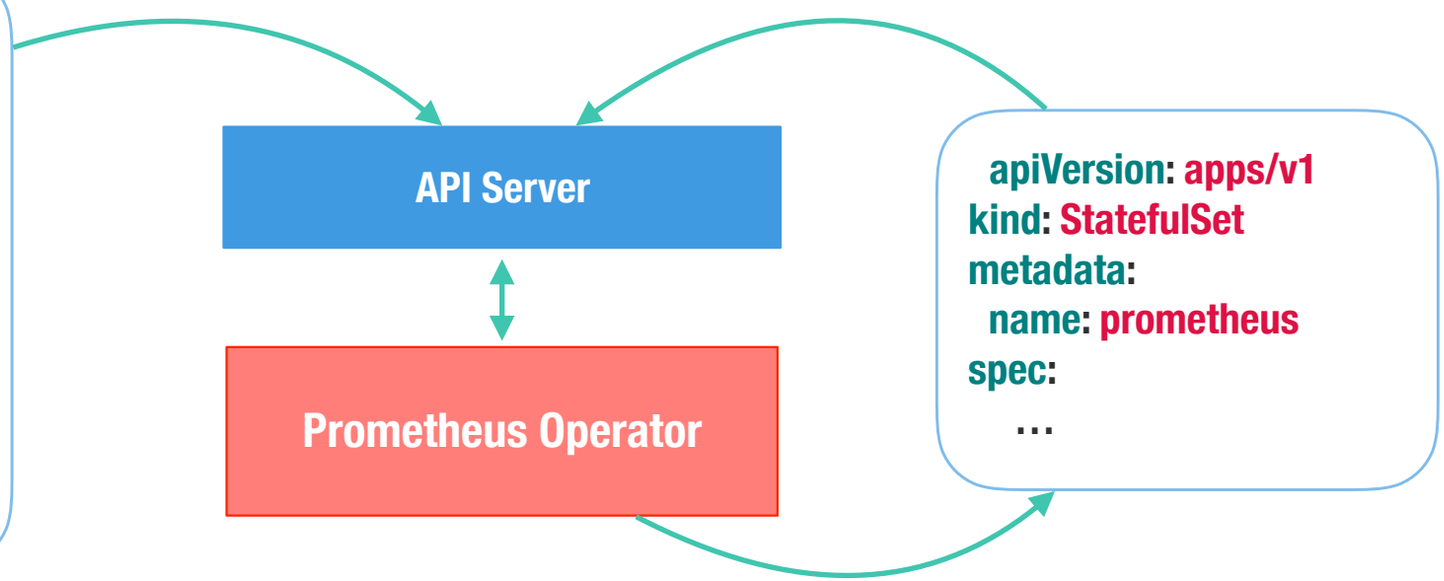
Kube-lego

Ingress Spec

```
spec:
  tls:
  - secretName: mysql-tls
    hosts:
    - phpmyadmin.example.com
    - mysql.example.com
  - secretName: postgres-tls
    hosts:
    - postgres.example.com
```

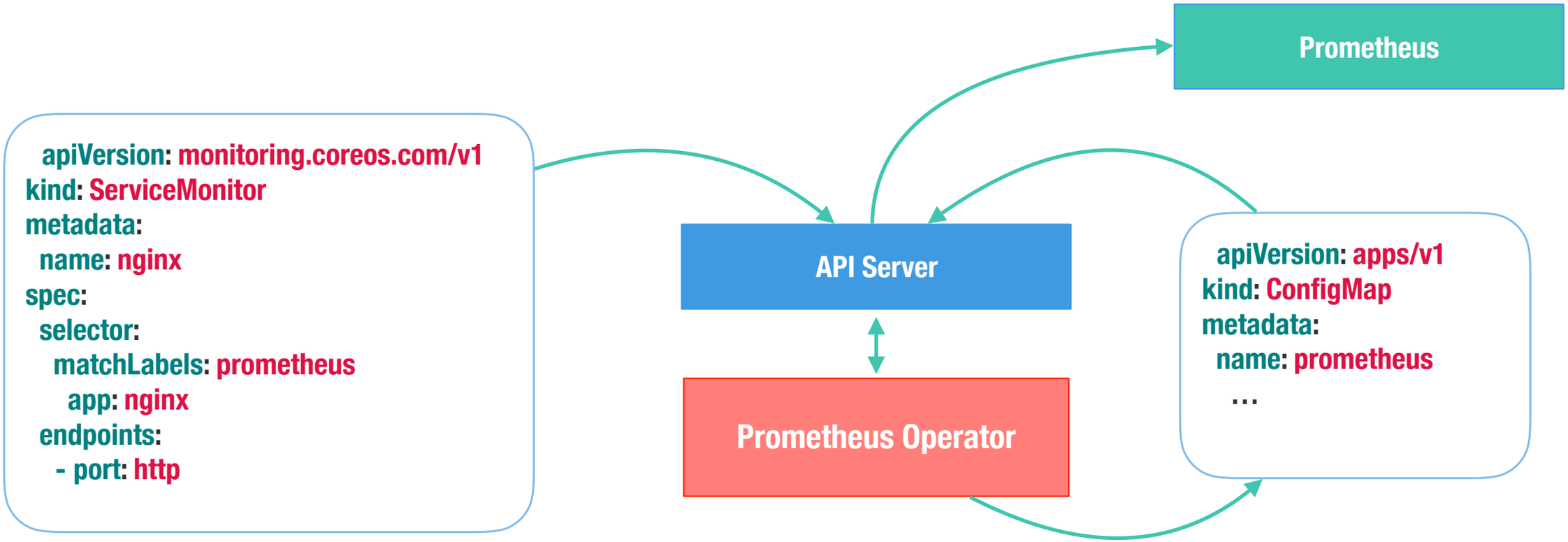
Prometheus Operator

```
apiVersion: monitoring.coreos.com/v1
kind: Prometheus
metadata:
  name: prometheus
spec:
  replicas: 2
  serviceAccountName: prometheus
  alerting:
    alertmanagers:
...
```

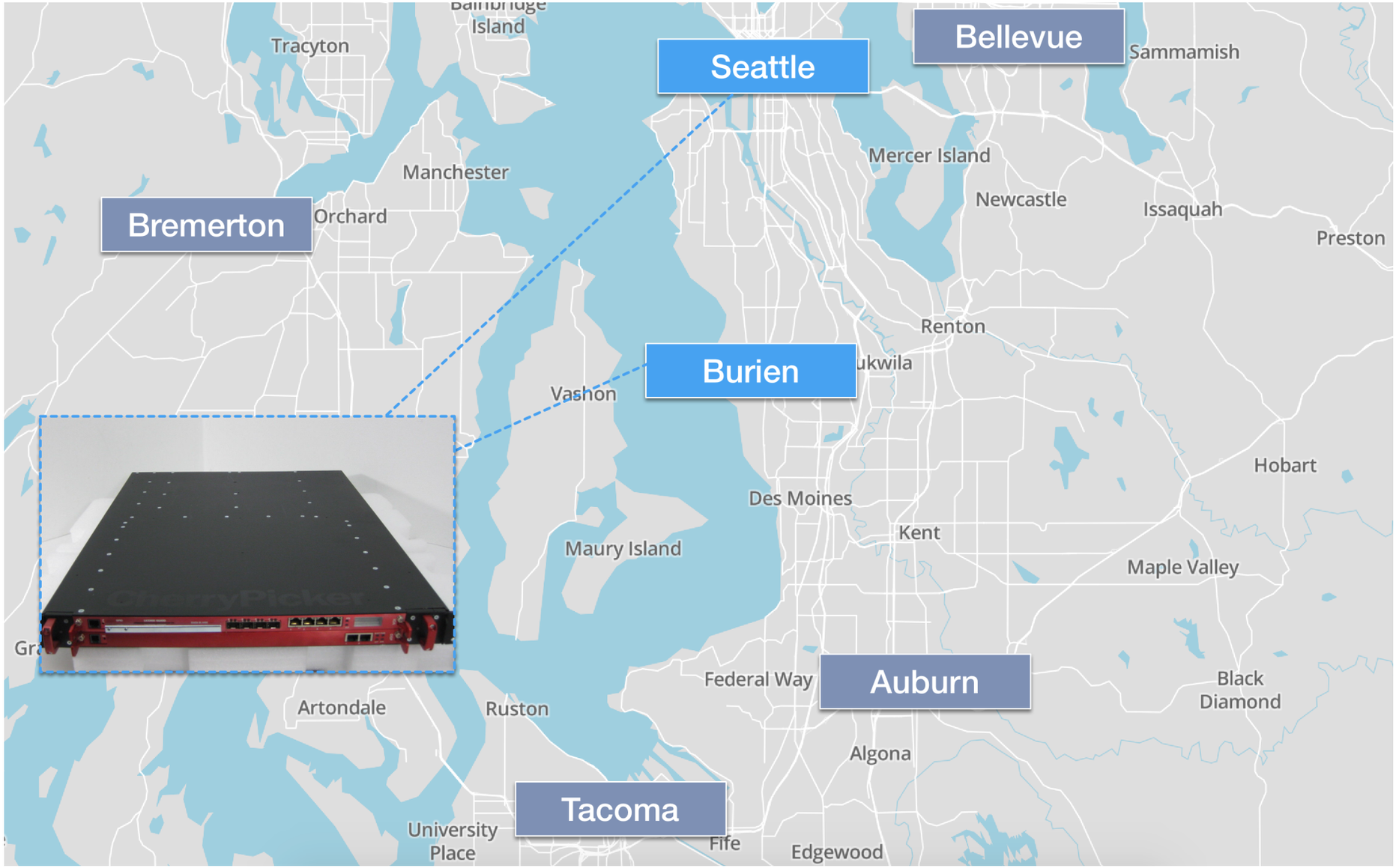



```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: prometheus
spec:
  ...
```

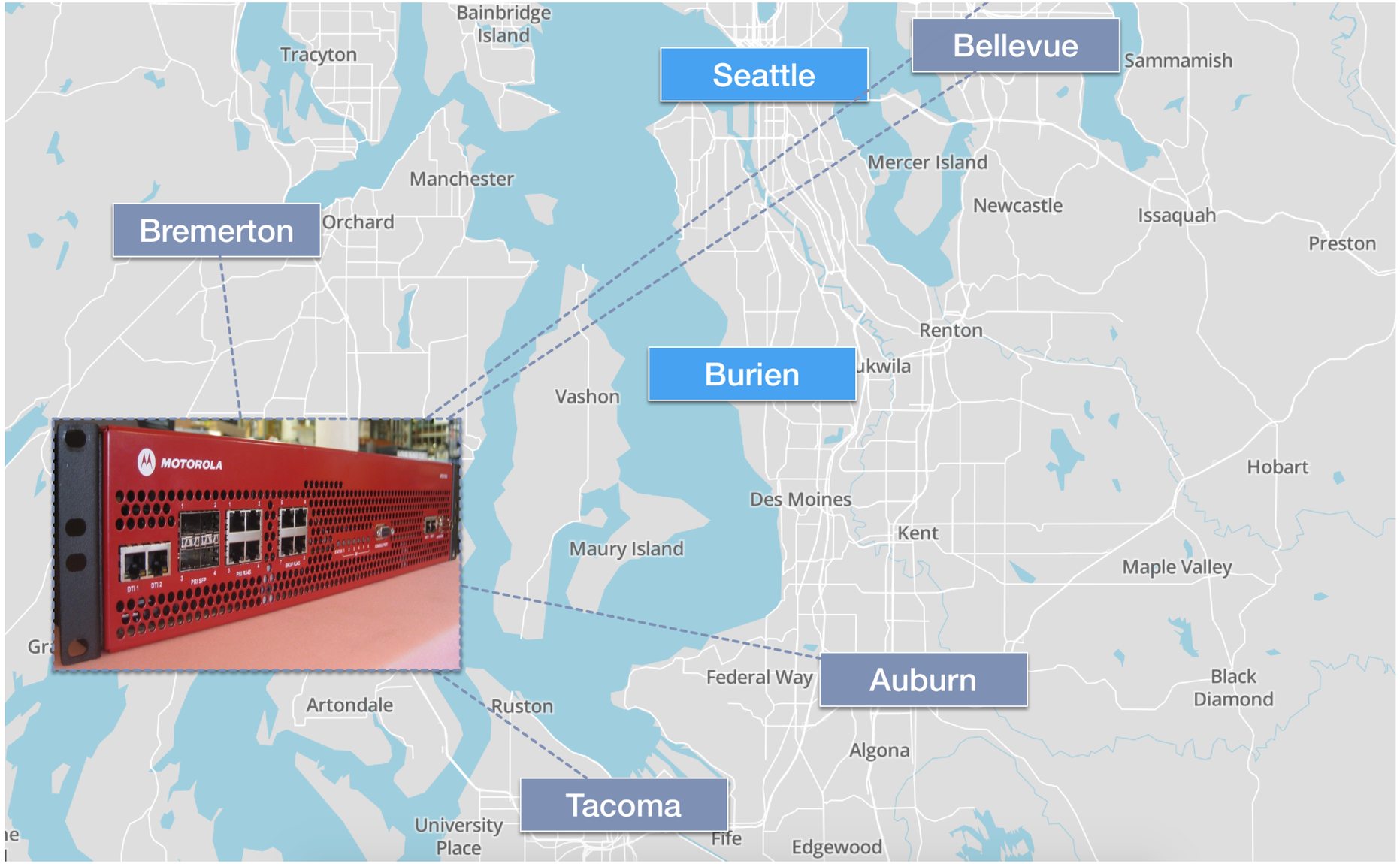
Prometheus Operator



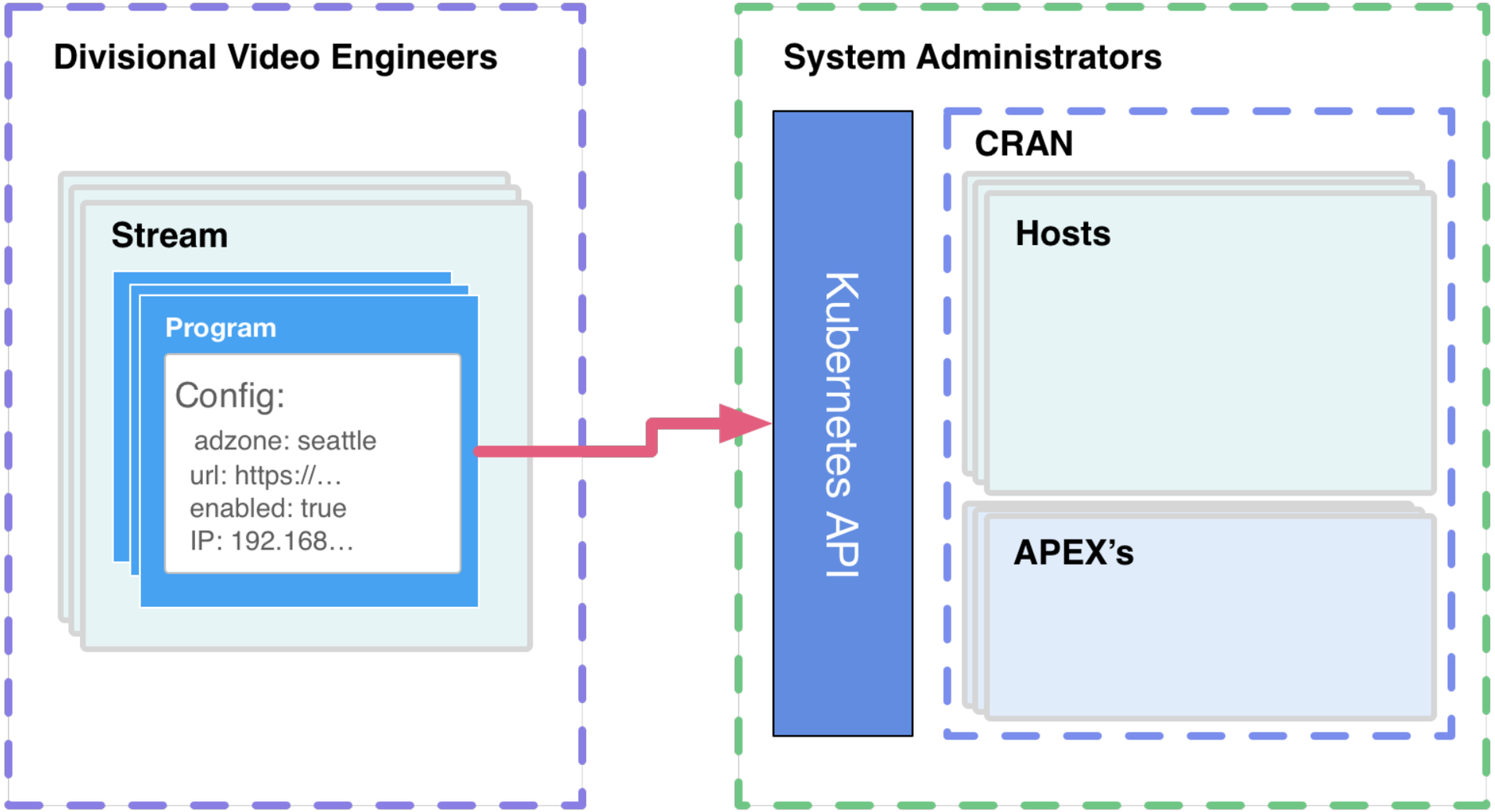
CRDs / Custom Controllers



CRDs / Custom Controllers



CRDs / Custom Controllers



CRDs vs API Aggregation

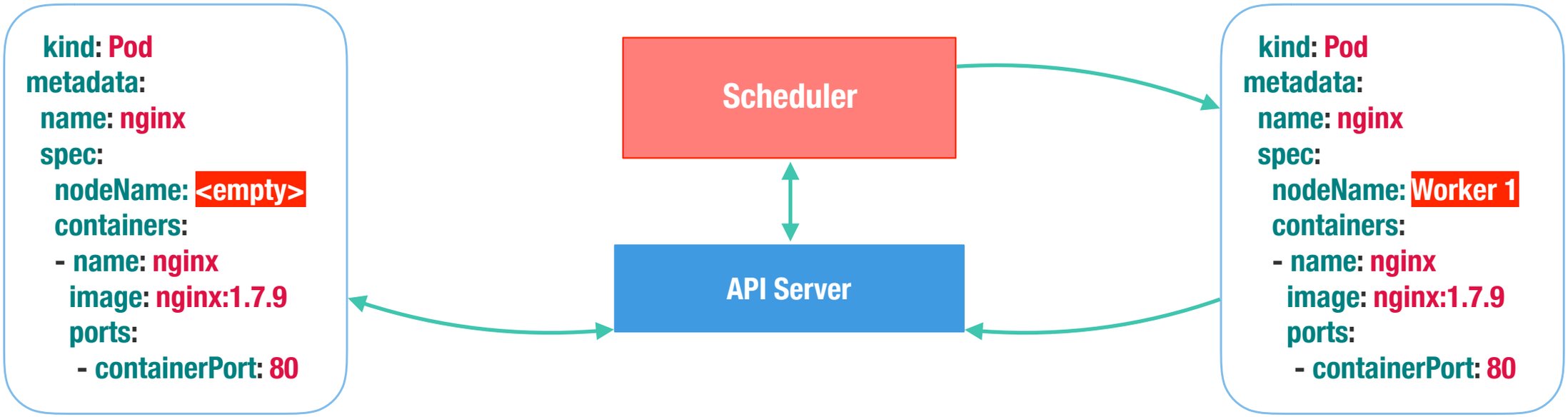
CRD

- Can use any programming language
- No need to handle multiple API versions
- Can do minimal validation (beta 1.9)
- Supports Scale and Status sub-resources

API Aggregation

- Must use Go
- Must handle multiple API versions
- Any validation you want
- Implement any sub-resources you'd like, including things like exec, attach, etc

Scheduler


kind: Pod
metadata:
name: nginx
spec:
nodeName: <empty>
containers:
- name: nginx
image: nginx:1.7.9
ports:
- containerPort: 80
Scheduler
API Server
kind: Pod
metadata:
name: nginx
spec:
nodeName: Worker 1
containers:
- name: nginx
image: nginx:1.7.9
ports:
- containerPort: 80

Scheduler

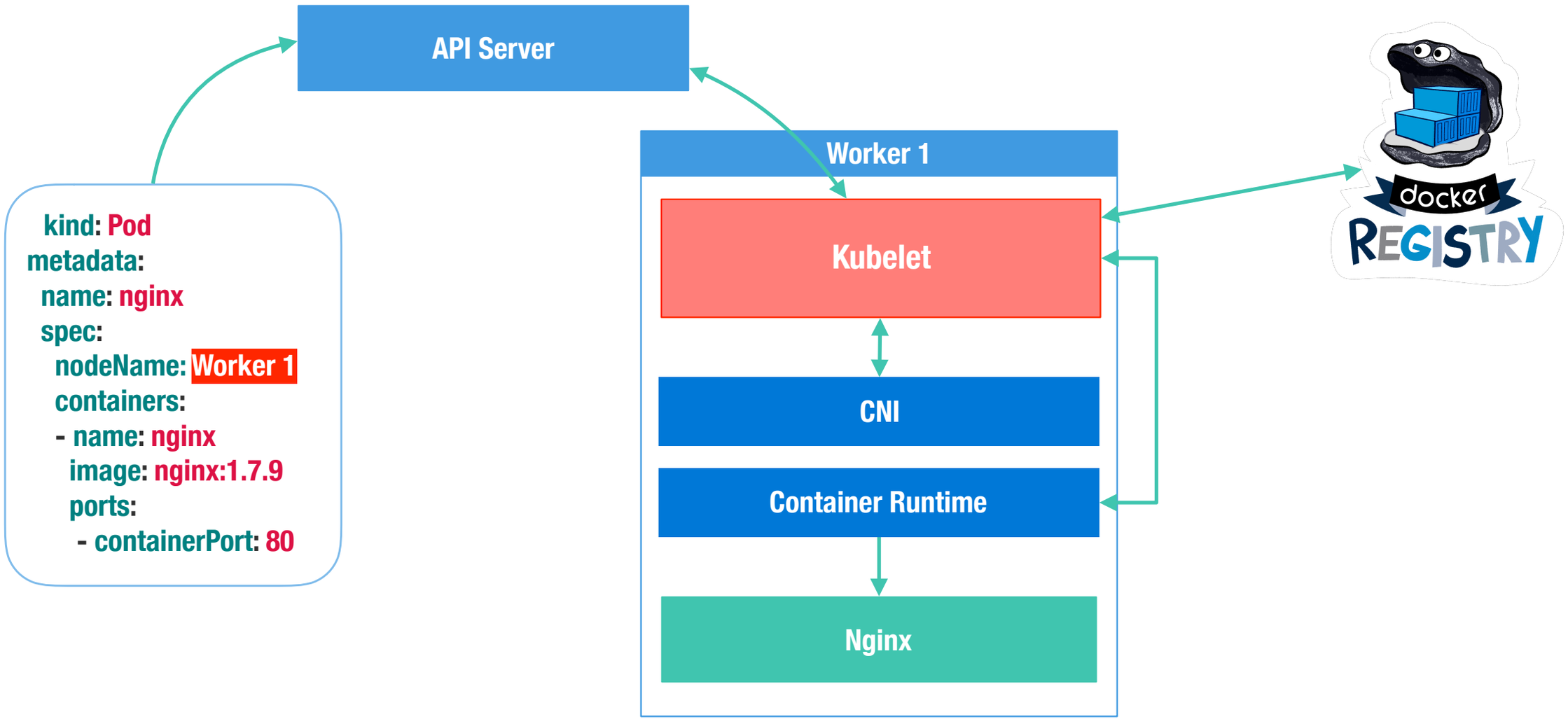
Why would you want to create your own?

- You need outside information for scheduling decisions
- Network based scheduling
- Running workloads close to the data it needs
- Hardware Encryption modules that are pre-primed with data, etc

Kubelet

- Calls the configured container runtime
- Pulls images from the image registry associated with pods assigned to this node.
- Creates and mounts volumes within a container
- Injects environment variables and updates volumes with Secrets, ConfigMaps, and the Downward API
- Updates the API Server with the latest Pod statuses
- Provides an API for the API Server to call for things like
  - kubectl exec
  - kubectl attach
  - kubectl logs
  - metrics used by the scheduler and dashboard
- Configures Pod networking (CNI)

Kubelet

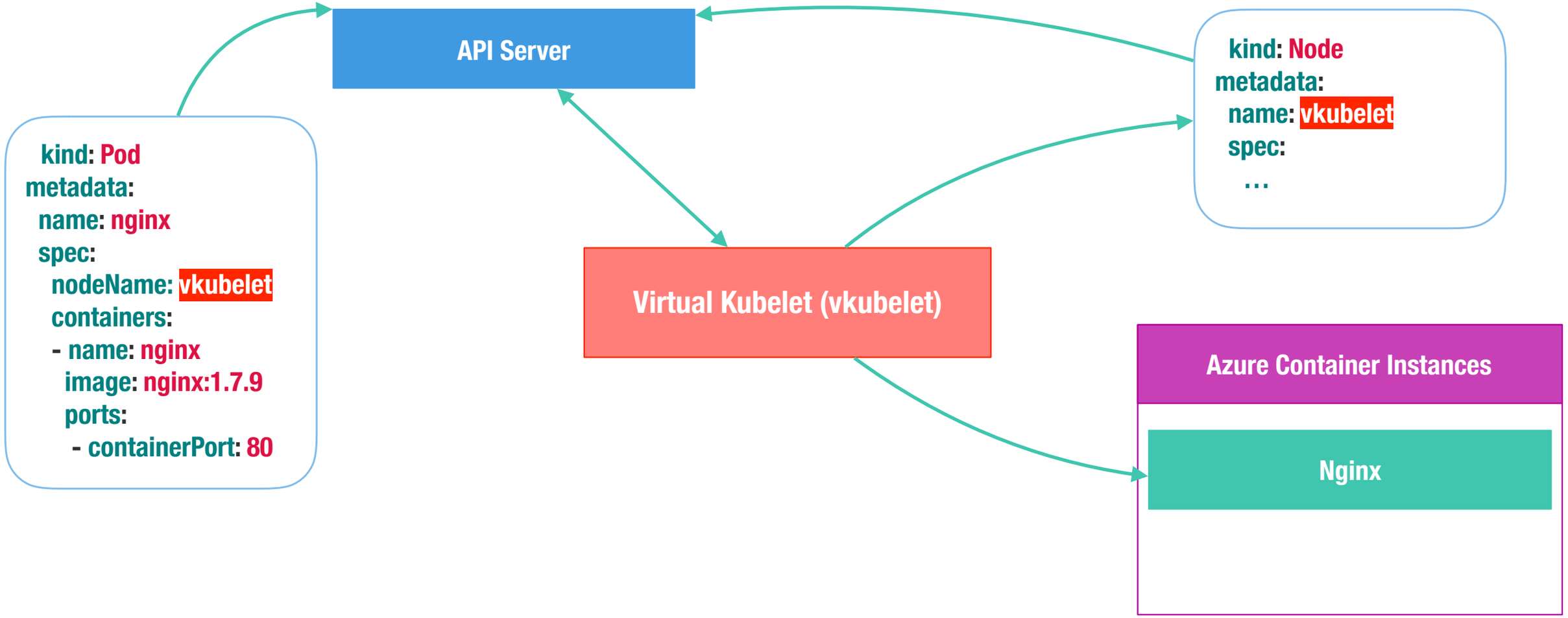

API Server
kind: Pod
metadata:
name: nginx
spec:
nodeName: Worker 1
containers:
- name: nginx
image: nginx:1.7.9
ports:
- containerPort: 80
Worker 1
Kubelet
CNI
Container Runtime
Nginx
docker
REGISTRY

Kubelet

Why would you want to create your own?

- You probably don’t
- but… we did anyway (Virtual Kubelet)

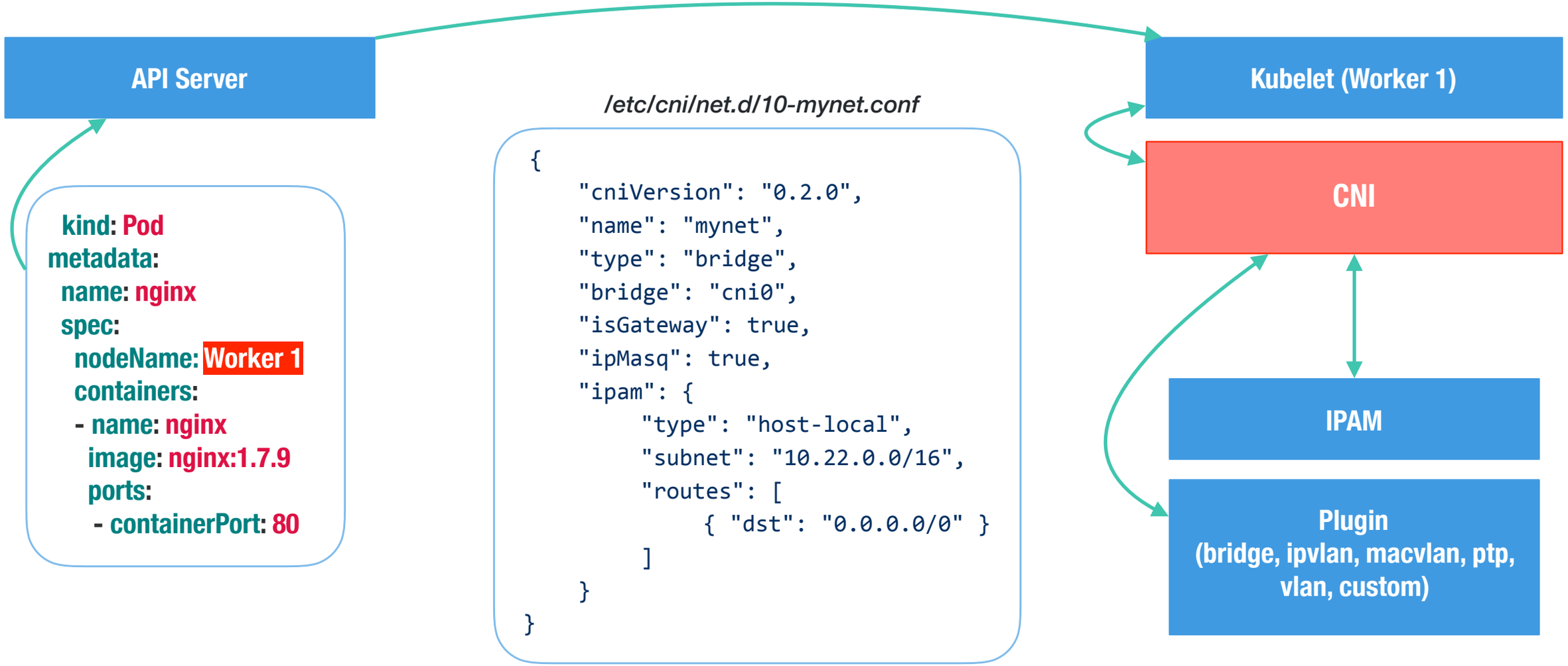
Virtual Kubelet



Virtual Kubelet - Why?

- Batch Jobs
- CI/CD
- Serverless
- Bursting

CNI

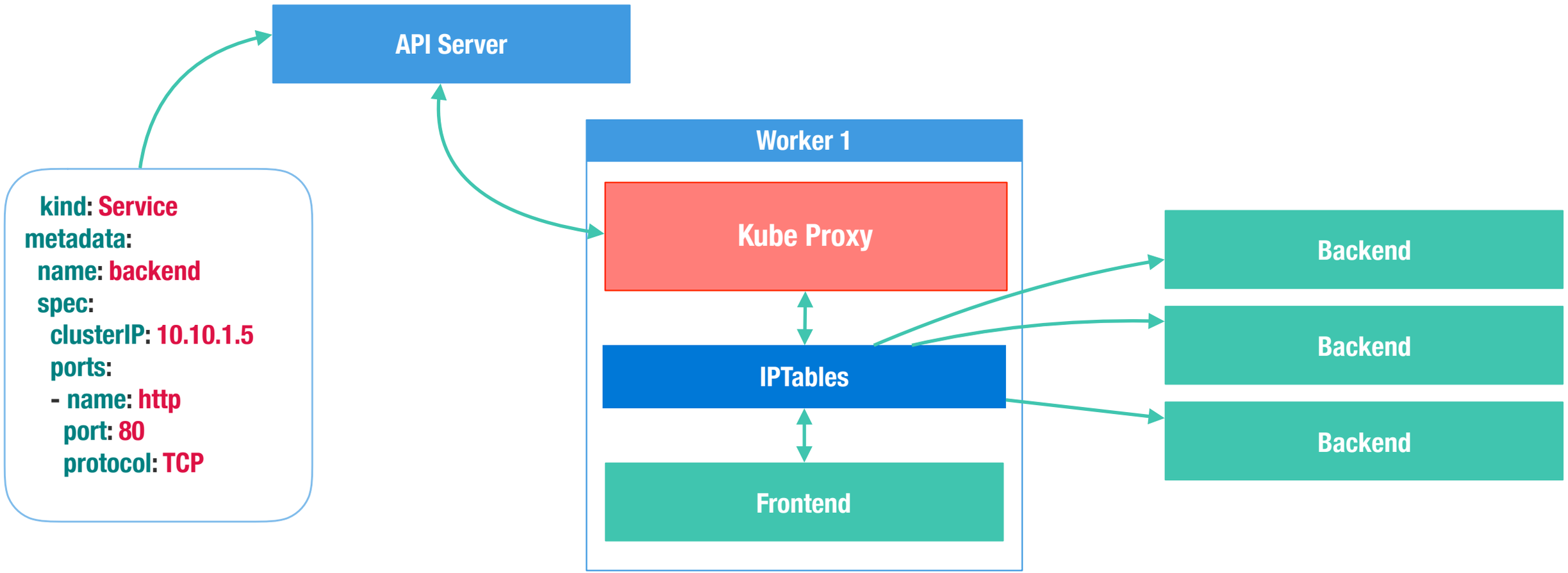

API Server
kind: Pod
metadata:
name: nginx
spec:
nodeName: Worker 1
containers:
- name: nginx
image: nginx:1.7.9
ports:
- containerPort: 80
/etc/cni/net.d/10-mynet.conf
{
"cniVersion": "0.2.0",
"name": "mynet",
"type": "bridge",
"bridge": "cni0",
"isGateway": true,
"ipMasq": true,
"ipam": {
"type": "host-local",
"subnet": "10.22.0.0/16",
"routes": [
{ "dst": "0.0.0.0/0" }
]
}
}
Kubelet (Worker 1)
CNI
IPAM
Plugin
(bridge, ipvlan, macvlan, ptp, vlan, custom)

CNI

Why would you want to create your own?

- Multiple interfaces (CNI-Genie, Multus)
- Custom Routing (BGP, NetConf, host/container routes, etc)
- SDN (OpenFlow, etc)
- Firewall configuration

Kube Proxy


API Server
kind: Service
metadata:
name: backend
spec:
clusterIP: 10.10.1.5
ports:
- name: http
port: 80
protocol: TCP
Worker 1
Kube Proxy
IPTables
Frontend
Backend
Backend
Backend

But wait, there's more!

- Extended Resources (Custom resource limits)
- initContainers (run a container before the rest of the pod starts, can have access to things the app containers don't, like secrets)
- Initializers and Finalizers (force constraints)
- readiness and liveness probes
- node affinity / anti-affinity
- pod affinity / anti-affinity

Istio

- Can modify headers and inject distributed tracing, etc
- Policy Enforcement
- Intelligent Request Routing A/B tests, etc
- Timeouts
- Bounded retries with timeout budgets and variable jitter between retries
- Limit number of concurrent connections and requests to upstream services
- Active (periodic) health checks on each member of the load balancing pool
- Fine-grained circuit breakers (passive health checks) – applied per instance in the load balancing pool

Abstracting Kubernetes



Helm



- Package manager for Kubernetes
- Abstracts multiple resources into an "application"
- Supports upgrades & rollbacks of the entire application
- Repository of pre-created charts (https://hub.kubeapps.com/)

OpenFaaS

- Functions as a Service - backed by Kubernetes



OPENFAAS

Draft



- Draft makes it easy to build applications that run on Kubernetes.
- Draft packs consist of a Dockerfile and a Helm Chart that demonstrates best practices for deploying applications of a given language

Pachyderm



- Open Source data pipelining and data management layer for Kubernetes
- Orchestrates workloads
- Manages input and output data to ensure it is available to the write code at the write time
- Ensures that jobs that require or could benefit from GPU acceleration end up on nodes with GPU resources.
- Data Versioning and Provenance

Metaparticle



- Metaparticle is a standard library for cloud native applications on Kubernetes.
- Democratizes the development of distributed systems.
- Collection of libraries that enable programmers to build and deploy containers using code that feels familiar to them.
- Aims to use language level features to add new capabilities to existing programming languages.

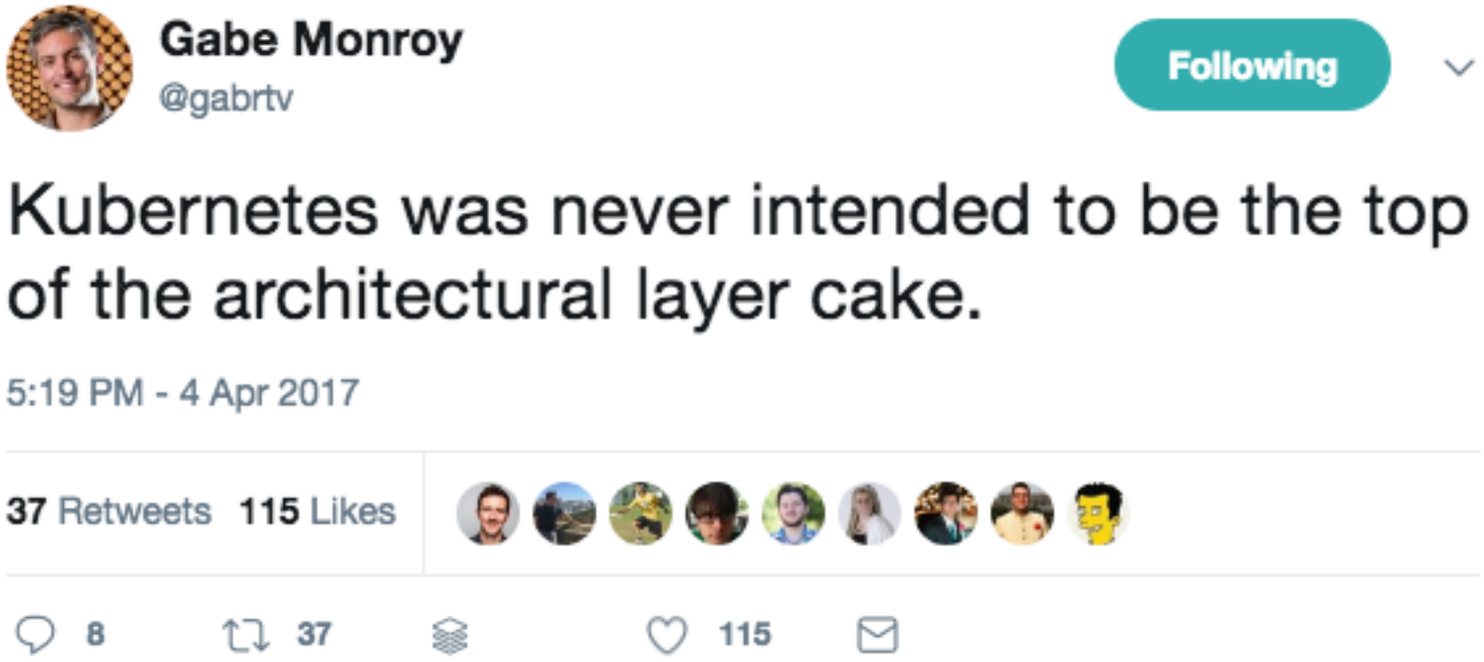
Metaparticle

```
metaparticle.Containerize(
    &metaparticle.Runtime{
        Ports:    []int32{port},
        Executor: "metaparticle",
        Replicas: 3,
    },
    &metaparticle.Package{
        Name:       “web-demo",
        Repository: "docker.io/myuser",
        Builder:    "docker",
        Publish:    true,
    },
    func() {
        http.HandleFunc("/", handler)
        err := http.ListenAndServe(fmt.Sprintf(":%d", port), nil)
        if err != nil {
            log.Fatal("Couldn't start the server: ", err)
        }
    })
```

Metaparticle

```
metaparticle.Containerize(
    &metaparticle.Runtime{
        Ports:    []int32{port},
        Executor: "metaparticle",
        Shards: 3,
        URLShardPattern: "^\\/users\\/([^\\/]*)\\/.*",
    },
    &metaparticle.Package{
        Name:       “shard-demo",
        Repository: "docker.io/myuser",
        Builder:    "docker",
    },
    func() {
        http.HandleFunc("/", handler)
        err := http.ListenAndServe(fmt.Sprintf(":%d", port), nil)
        if err != nil {
            log.Fatal("Couldn't start the server: ", err)
        }
    })
```

The Future


Gabe Monroy
@gabrtv
Following
Kubernetes was never intended to be the top of the architectural layer cake.
5:19 PM - 4 Apr 2017
37 Retweets 115 Likes
8
37
115

Thank You!

Twitter: @erikstmartin
Blog: erikstmartin.com
GitHub: github.com/erikstmartin



Questions?

No question is a bad question.