Building an on premise Kubernetes cluster

DANNY TURNER

Outline

What is K8s?

Why (not) run k8s?

Why run our own cluster?

Building what the public cloud provides

Kubernetes

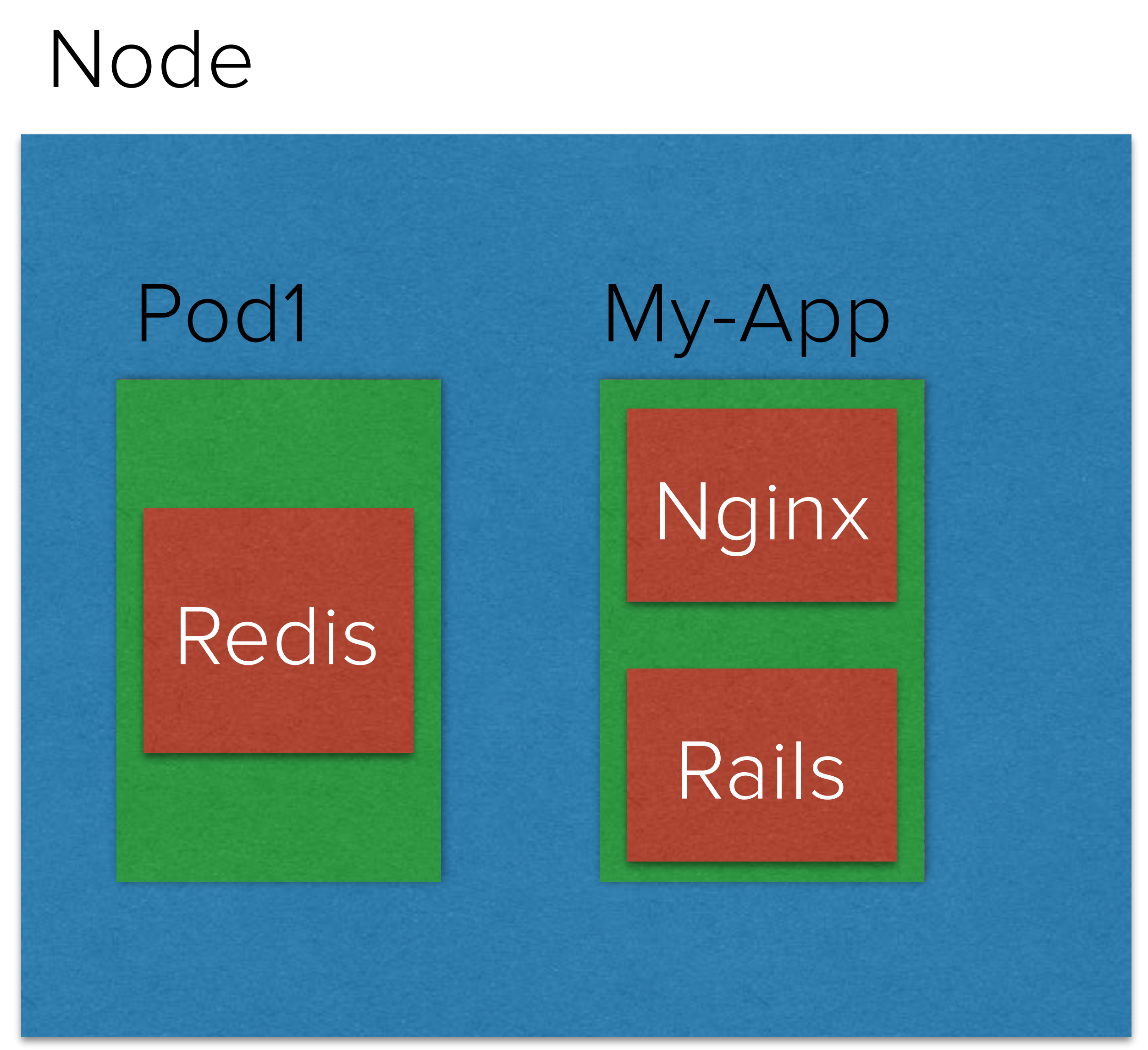
- Open-Source Container Management Platform
  - Deploying
  - Scaling
  - Share Hardware
- Service Discovery
- Configuration Management

Kubernetes Terms

- Node
  - Server

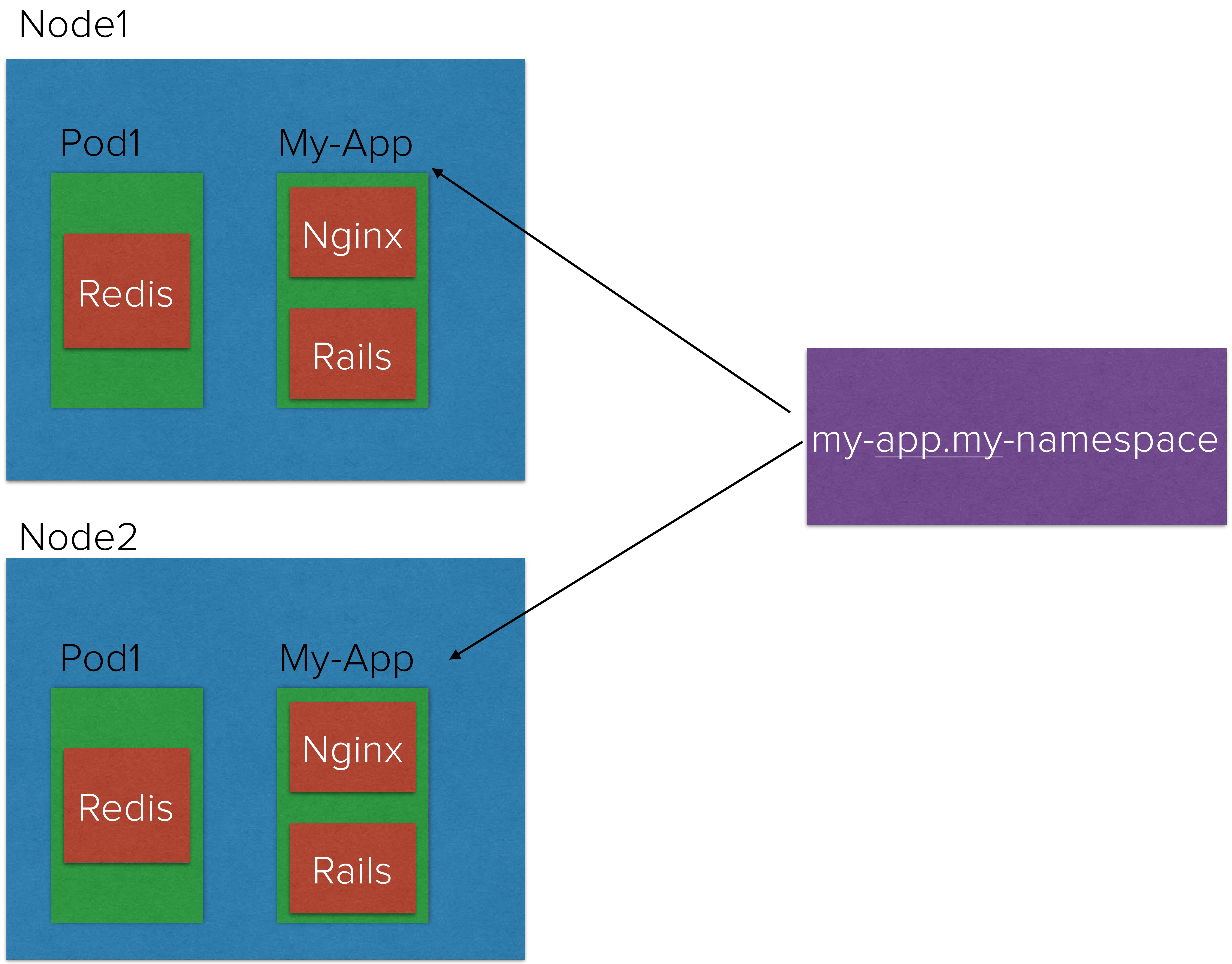
Kubernetes Terms

- Node
    - Server
- Pod
    - 1 or more containers
    - Redis
    - Rails & nginx



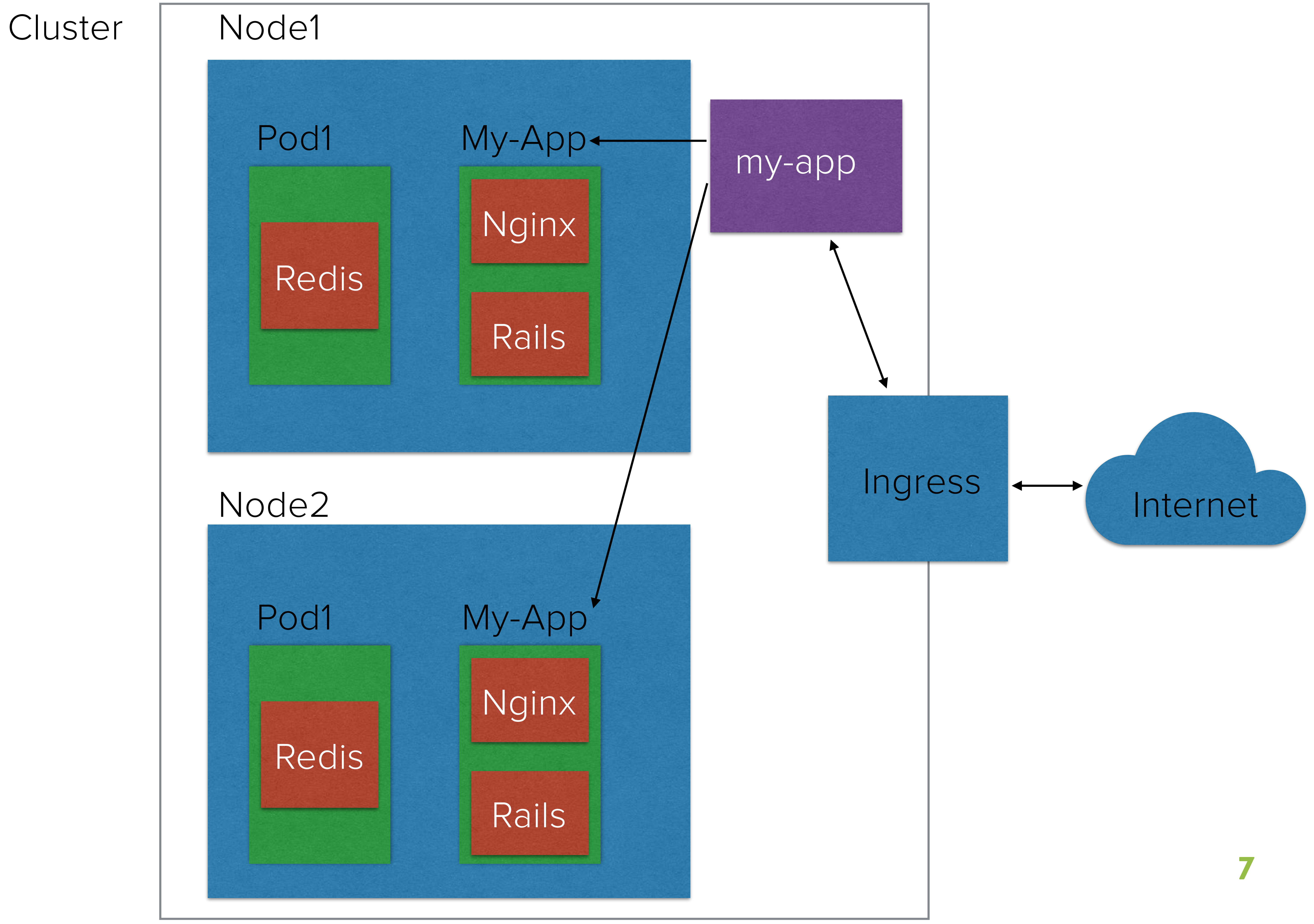
Kubernetes Terms

- Node
  - Server
- Pod
  - 1 or more containers
  - Redis
  - Rails & nginx
- Service
  - DNS name for 1 or more pods



Kubernetes Terms

- Node
  - Server
- Pod
  - 1 or more containers
  - Redis
  - Rails & nginx
- Service
  - DNS name for 1 or more pods
- Ingress
  - Bridge into the cluster



Why Kubernetes

- We already use containers
- We have our container management system
    - Only runs our monolith
    - Scaling unit is a host
    - Not open source

Why not run K8s

- Long running Jobs
    - DB migration
- Fixed scheduling assumptions
    - Number of workers per server
- Exposing internal services to external tools
    - Stateful services like redis/DBs

Why build our own

- We have 2 data centers filled with hardware

Why build our own

- We have 2 data centers filled with hardware
- Cloud Pricing might not be competitive at scale
    - Hard to determine op-ex of running a DC

Why build our own

- We have 2 data centers filled with hardware
- Cloud Pricing might not be competitive at scale
- One change at at time
    - Easy to connect to resources outside of k8s but in the DC

Why build our own

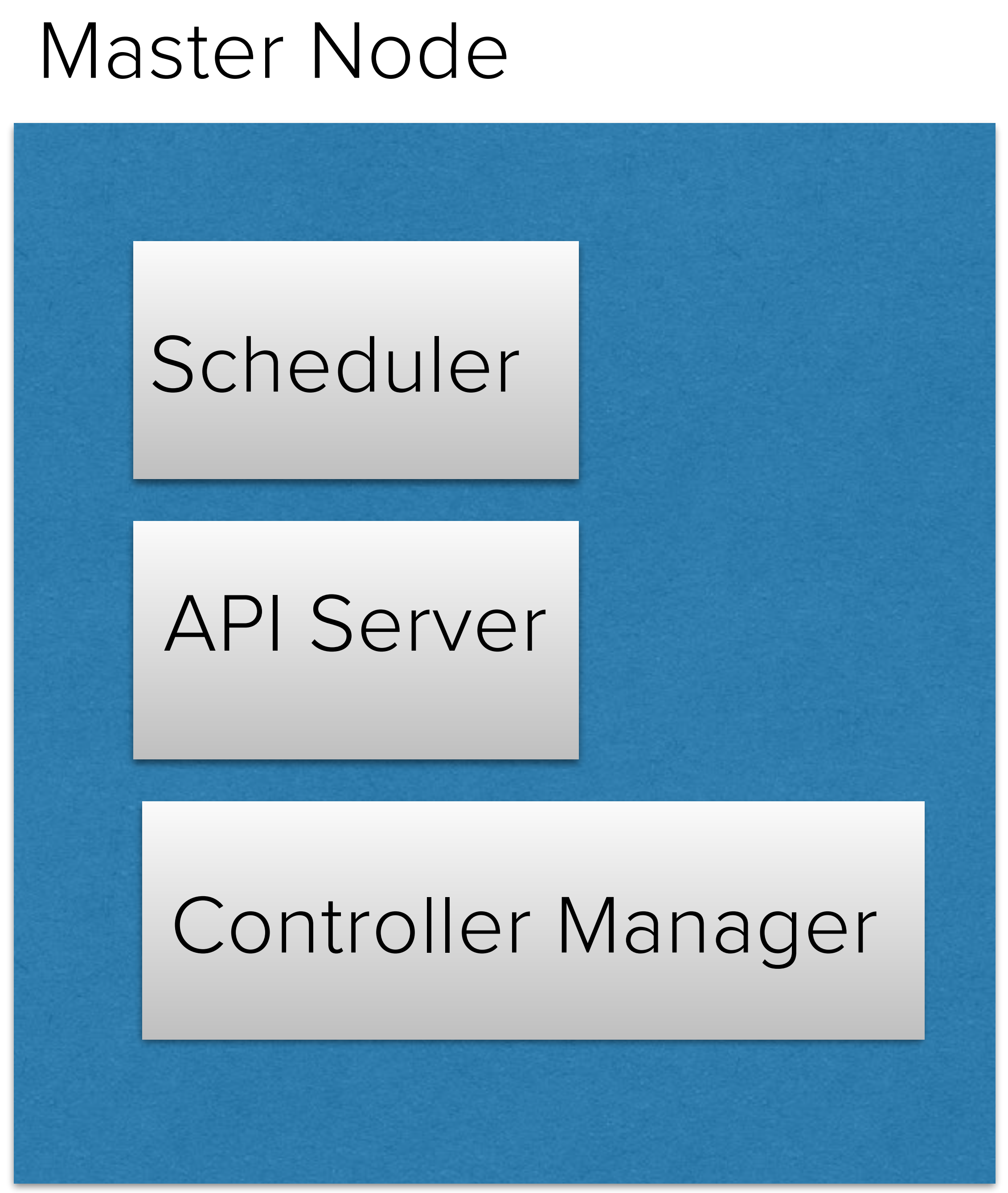
- We have 2 data centers filled with hardware
- Cloud Pricing might not be competitive at scale
- One change at at time
- Security & Privacy
    - DC doesn’t need secure communication between servers
    - Trusting our data in 3rd party hands

On Premise work

- Master Node
- ETCD
- Networking & Ingress
- Persistent Storage

Master Components

- Assigns pods to nodes
- IPs to pods and services
- Health Checks
- Cluster is frozen w/o master node
    - cluster wont change itself
    - external forces can still happen

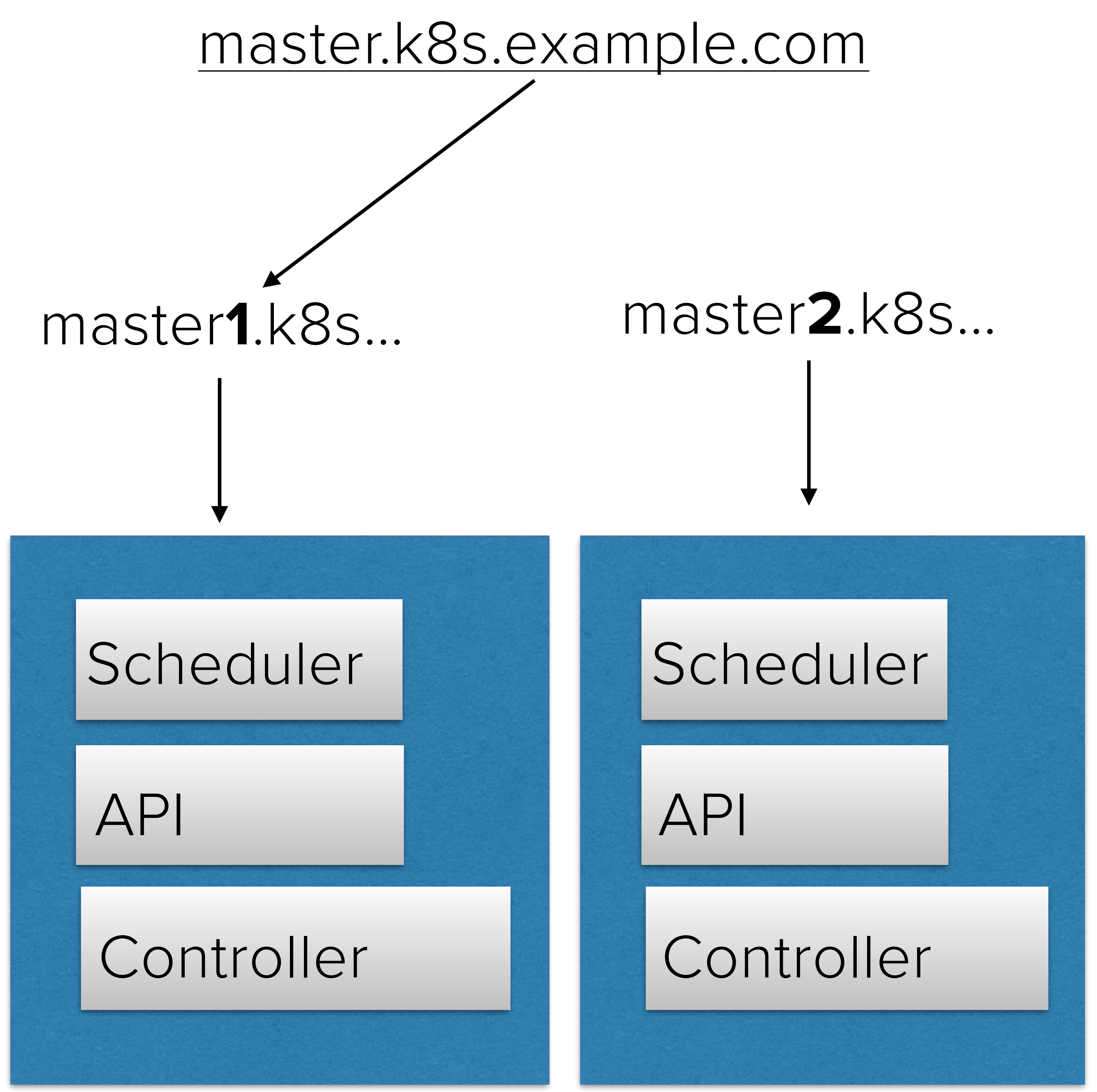


(High) Availability Strategies

- Start a new one after detecting a failure
    - Bottleneck: time to spin up a new master node
- Run multiple at once
    - Components are stateless and have leader election built-in
    - Bottleneck: failover strategy

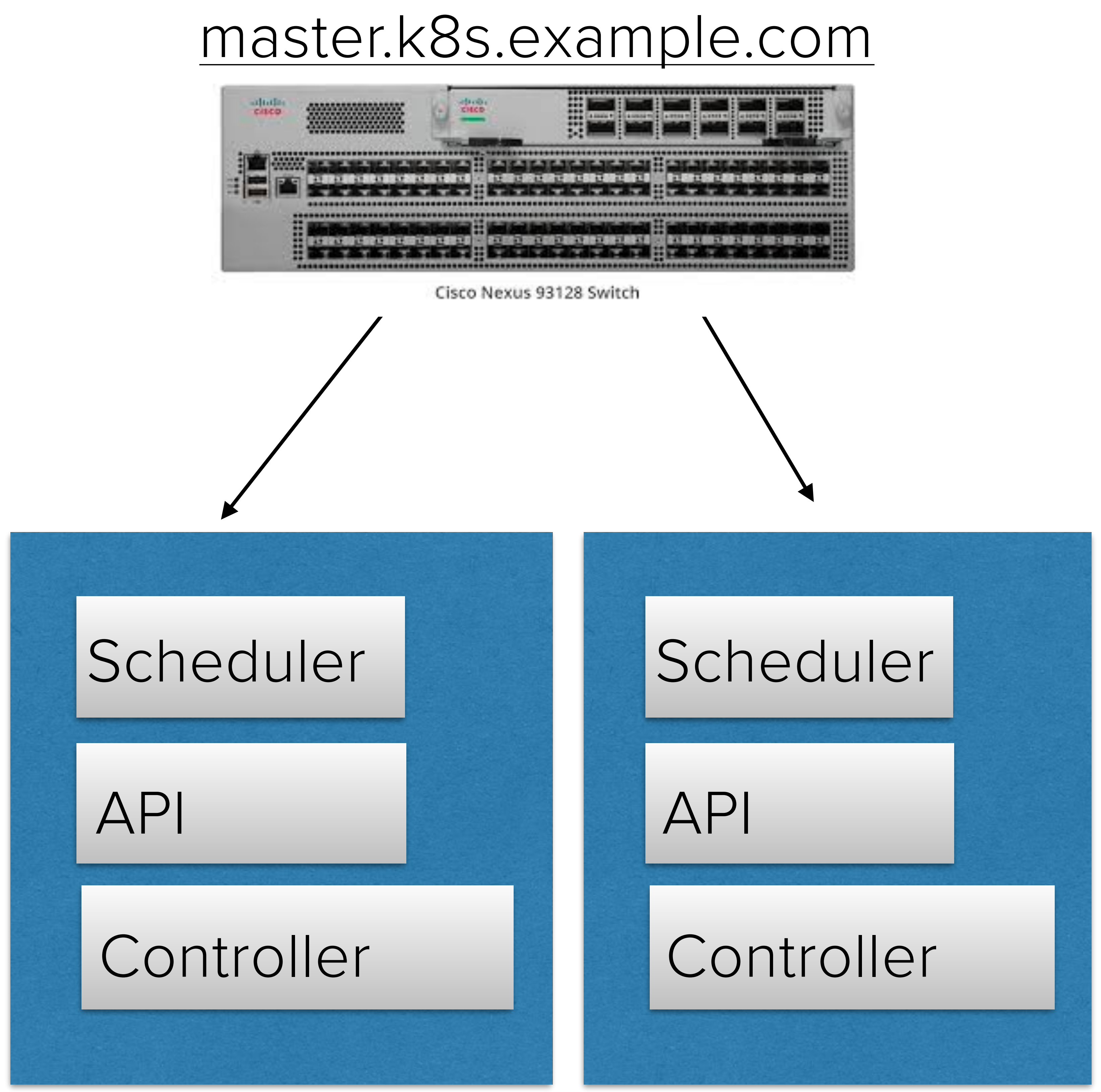
Multi-Master

- CNAME your master
    - Bottleneck: DNS propagation / timeouts



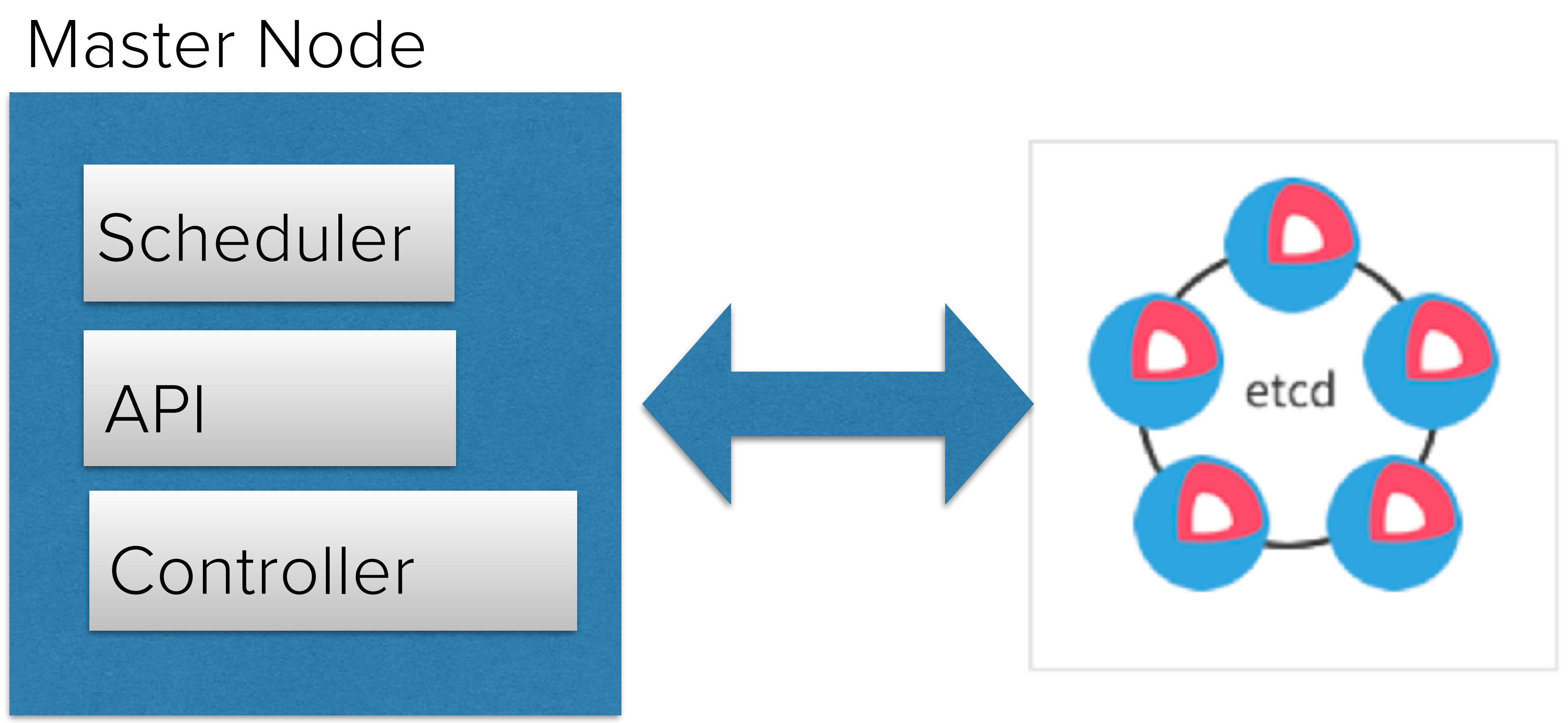
Multi-Master

- CNAME your master
  - Bottleneck: DNS propagation / timeouts
- Send requests to all the masters
  - ECMP to a Virtual-IP via an A-Record
  - Health checks on your masters!
  - Bottleneck: time to withdraw from ECMP group



etcd

- K8s data lives here
- Quorum is life
  - k8s frozen when quorum is lost
- Can be run on the master nodes
  - Limits scaling
  - Makes the servers pets not cattle



etcd

- Member discovery
  - Static configs
  - chef searches
  - SRV Records

```
$ etcd --name infra0 --initial-advertise-peer-urls http://10.0.1.10:2380 \
  --listen-peer-urls http://10.0.1.10:2380 \
  --listen-client-urls http://10.0.1.10:2379,http://127.0.0.1:2379 \
  --advertise-client-urls http://10.0.1.10:2379 \
  --initial-cluster-token etcd-cluster-1 \
  --initial-cluster infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2380,infra2=http://10.
0.1.12:2380 \
  --initial-cluster-state new
```

--discovery-srv etcd.example.com

```
$ dig +noall +answer SRV _etcd-server._tcp.example.com
_etcd-server._tcp.example.com. 300 IN  SRV  0 0 2380 infra0.example.com.
_etcd-server._tcp.example.com. 300 IN  SRV  0 0 2380 infra1.example.com.
_etcd-server._tcp.example.com. 300 IN  SRV  0 0 2380 infra2.example.com.
```

etcd

- Member discovery
  - Static configs
  - chef searches
  - SRV Records
- Backups
  - Live snapshots

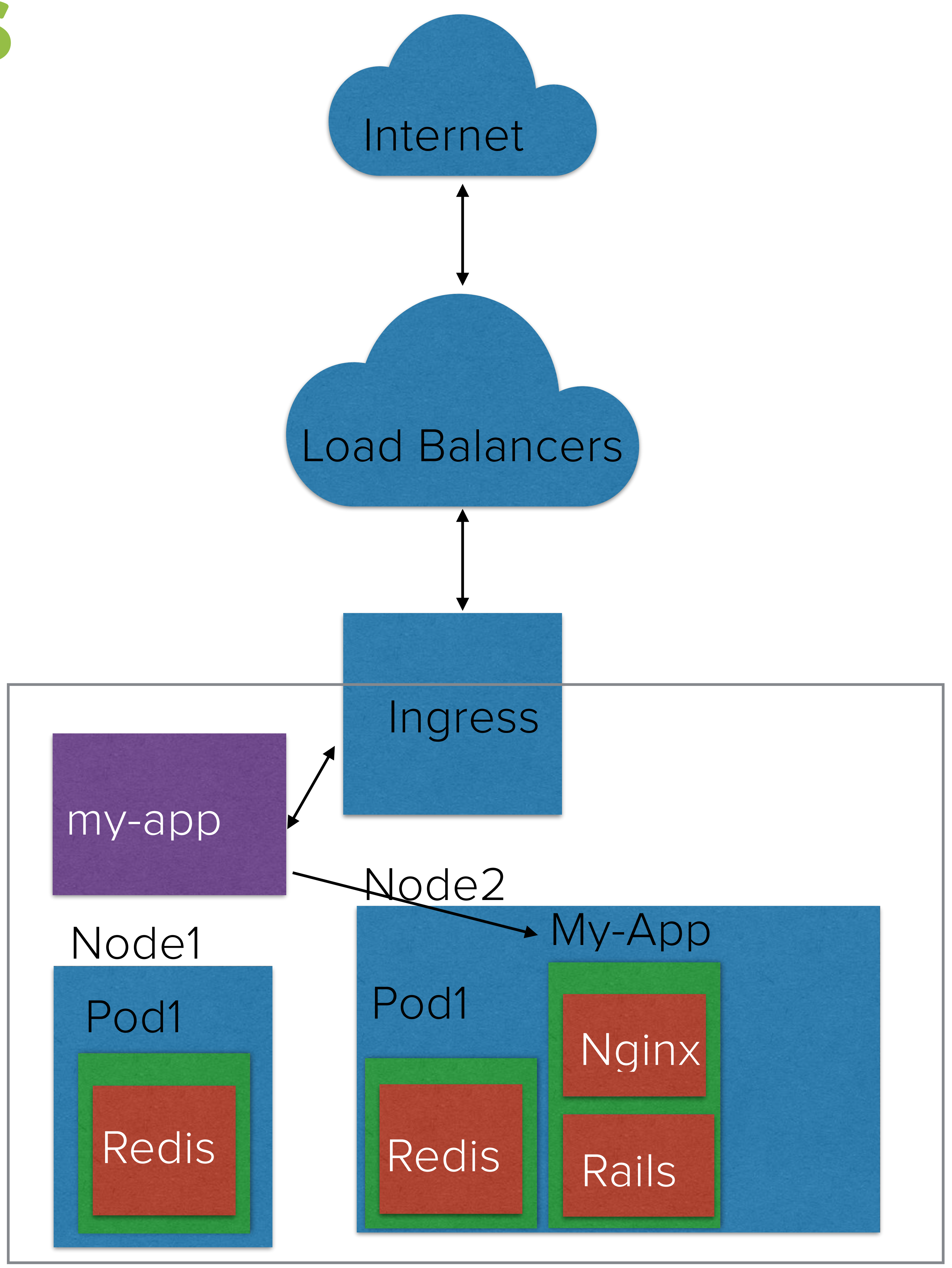
```
$ etcd --name infra0 --initial-advertise-peer-urls http://10.0.1.10:2380 \
  --listen-peer-urls http://10.0.1.10:2380 \
  --listen-client-urls http://10.0.1.10:2379,http://127.0.0.1:2379 \
  --advertise-client-urls http://10.0.1.10:2379 \
  --initial-cluster-token etcd-cluster-1 \
  --initial-cluster infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2380,infra2=http://10.
0.1.12:2380 \
  --initial-cluster-state new
```

```
--discovery-srv etcd.example.com
```

```
$ dig +noall +answer SRV _etcd-server._tcp.example.com
_etcd-server._tcp.example.com. 300 IN  SRV  0 0 2380 infra0.example.com.
_etcd-server._tcp.example.com. 300 IN  SRV  0 0 2380 infra1.example.com.
_etcd-server._tcp.example.com. 300 IN  SRV  0 0 2380 infra2.example.com.
```

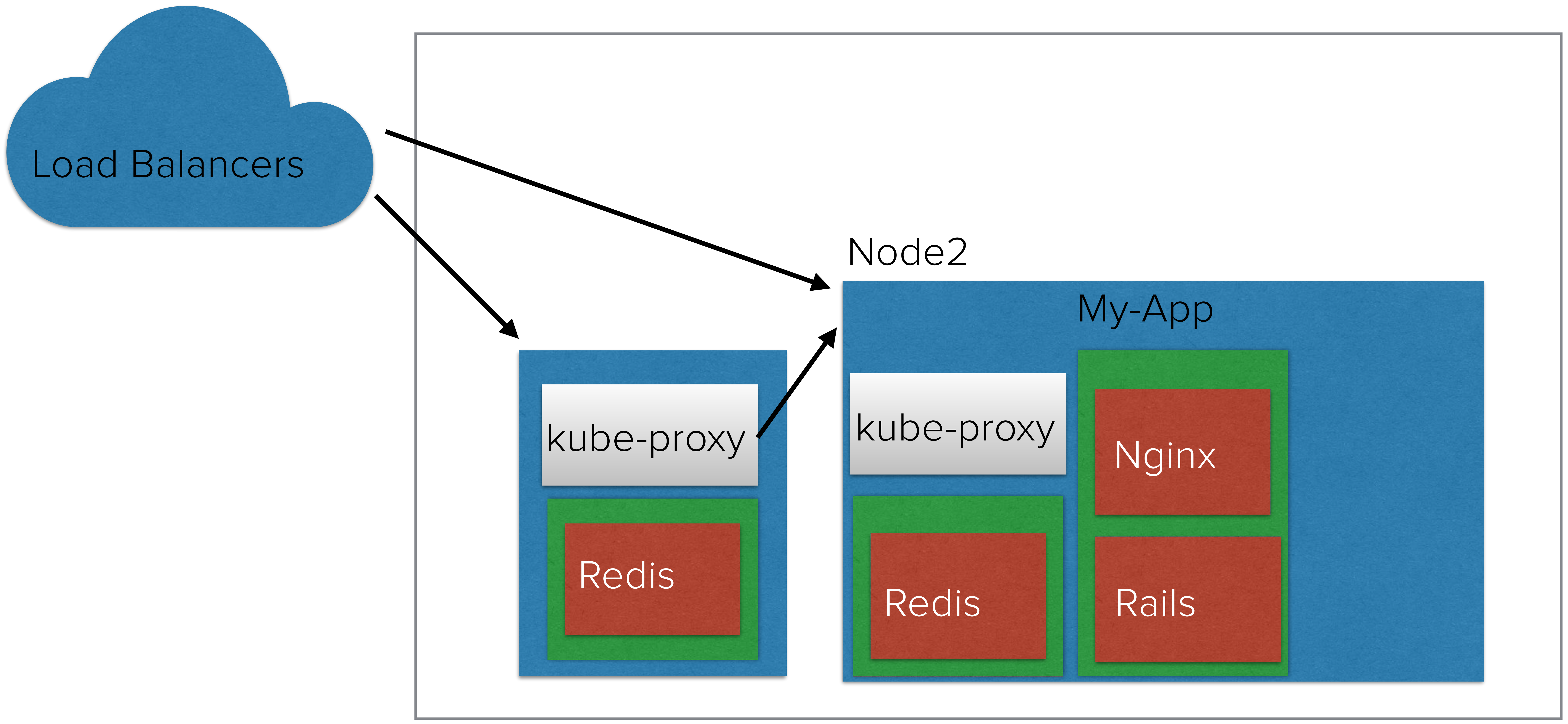
Ingress

- Bridge between the internet and a service
- Ingress Controller + nginx
    - Each deploy caused nginx to reload
- We already have a load balancing tier



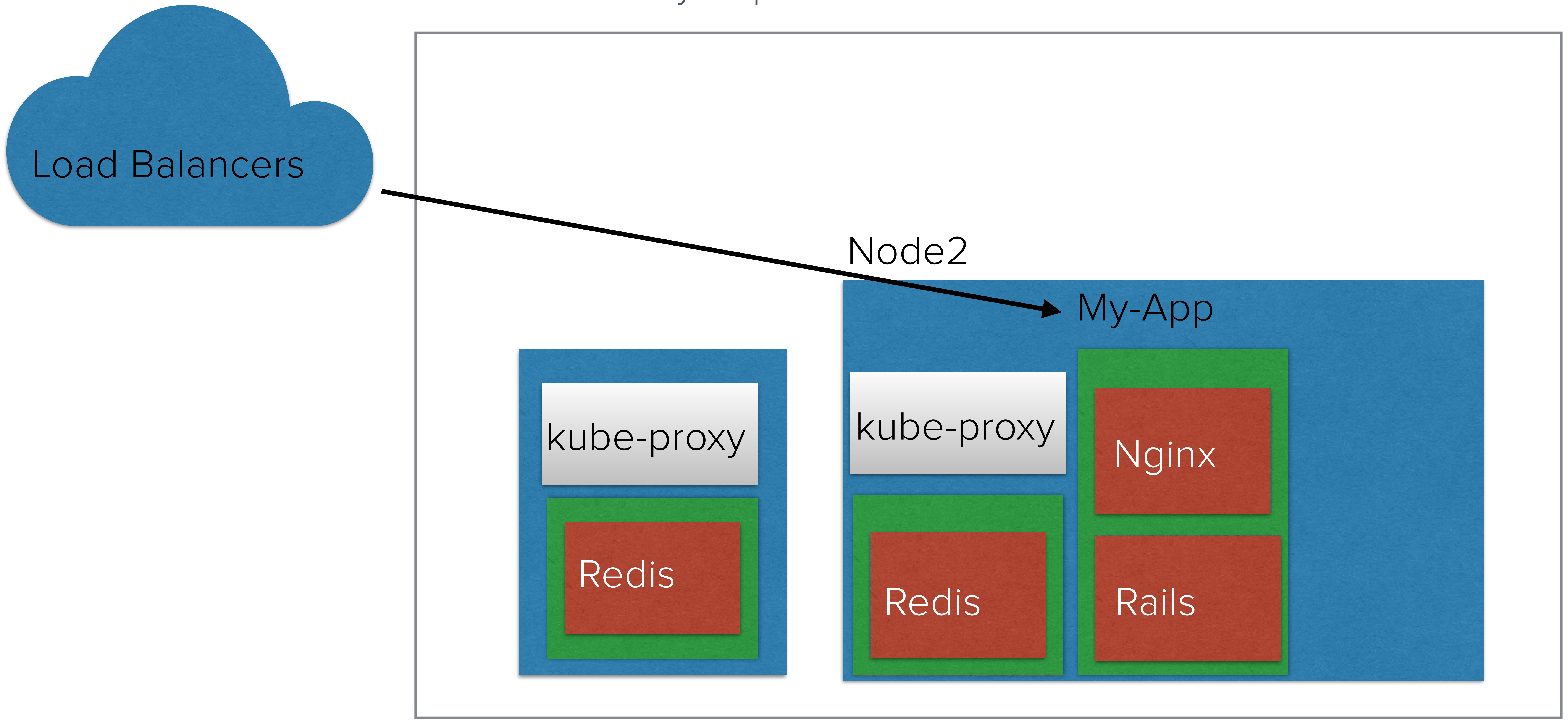
Ingress

- Services can be exposed on every host at a known port



Ingress

- Services can be exposed on every host at a known port
- Route directly to pods



Persistent Storage (Volumes)

- Persistent Volume Claims
- Distributed Storage System
    - GlusterFs / Ceph RBS
- Same nodes as k8s Cluster
    - Better use of hardware
    - Servers are pets once again
- Just buy a SAN?

Successful Failure

- We ran production traffic on our on-premise cluster
- Yet, we decided to use the cloud instead
    - Upgrades were painful
    - Solving a lot of problems ourselves
    - We were becoming experts at more things not less

QUESTIONS?

Check out our blog at **engineering.shopify.com**
Follow us on Twitter at **@shopifyeng**

DANNY TURNER

shopify

Networking

- All to all communication
  - Pod & Service IPs
- Routing
  - Calico (Software BGP)
- BGP Peer with top of rack switches
  - 1 peer per server
  - Calico custom filters