CoreOS

Self-Hosted Kubernetes
How and Why

Diego Pontoriero

Github/Slack: @diegs | diegs@coreos.com

We are CoreOS

kubernetes

KUBERNETES COMMUNITY

Top 3 contributor

Lead 6 SIGs

Creators of etcd

TECTONIC

ENTERPRISE KUBERNETES

Enterprise-ready

Automated operations

Cloud agnostic and hybrid

CoreOS

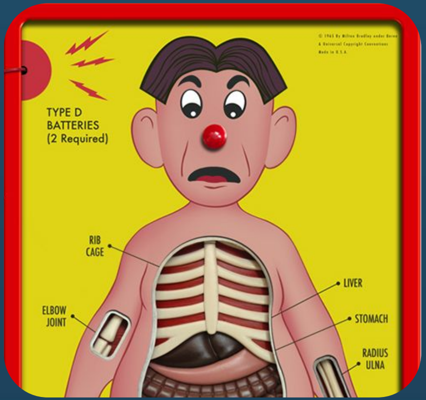
CONTAINERIZATION SUCCESS

Enterprise support

Field Engineering

Educational Services

Who this talk is for



Cluster Operators

Kubernetes Contributors

People who enjoy clever hacks

What is self-hosted Kubernetes?

What is self-hosted Kubernetes?

```
$ kubectl -n kube-system get deployments
NAME                      DESIRED   CURRENT
kube-controller-manager   2         2
kube-dns                  1         1
kube-scheduler            2         2
```

```
$ kubectl -n kube-system get daemonsets
NAME               DESIRED   CURRENT   NODE SELECTOR
kube-apiserver     1         1         node-role.kubernetes.io/master=
```

```
$ kubectl -n kube-system get secrets
NAME                           TYPE
kube-apiserver                 Opaque
kube-controller-manager        Opaque
```

A talk in three parts

Why self-hosted?

How does it work?

What’s next?

Part 1: Why self-hosted?

Why self-host Kubernetes?

- Leverage Kubernetes’ strengths
- Simplified, unified node management
- Streamlined, robust cluster lifecycle management

Desirable control plane properties

- Scales up and down automatically
- Handles node failures gracefully
- Safely rolls out new versions
- Rollback on upgrade failures

And what about...

- Advanced networking
- RBAC
- Health checking & monitoring
- Resource allocation & accounting

Simplified node management

Minimal on-host requirements:

Kubelet

Container Runtime

Credentials

kubeconfig, etc.

No distinction between masters and workers!

So how do we select masters?

Add a label to nodes you want to run “master” workloads:

```
$ kubectl label node n1 master=true
```

Or have the kubelet start as a master:

```
--node-labels=master=true
```

Any node can become a master at any time!

Streamlined lifecycle management

```
$ kubectl apply -f kube-apiserver.yaml
$ kubectl apply -f kube-scheduler.yaml
$ kubectl apply -f kube-controller-manager.yaml
$ kubectl apply -f kube-proxy.yaml
```

Better yet: **automate**.

Part 2: How does it work?

How does self-hosted Kubernetes work?

Three main areas to solve:

Bootstrapping

Upgrades

Disaster Recovery

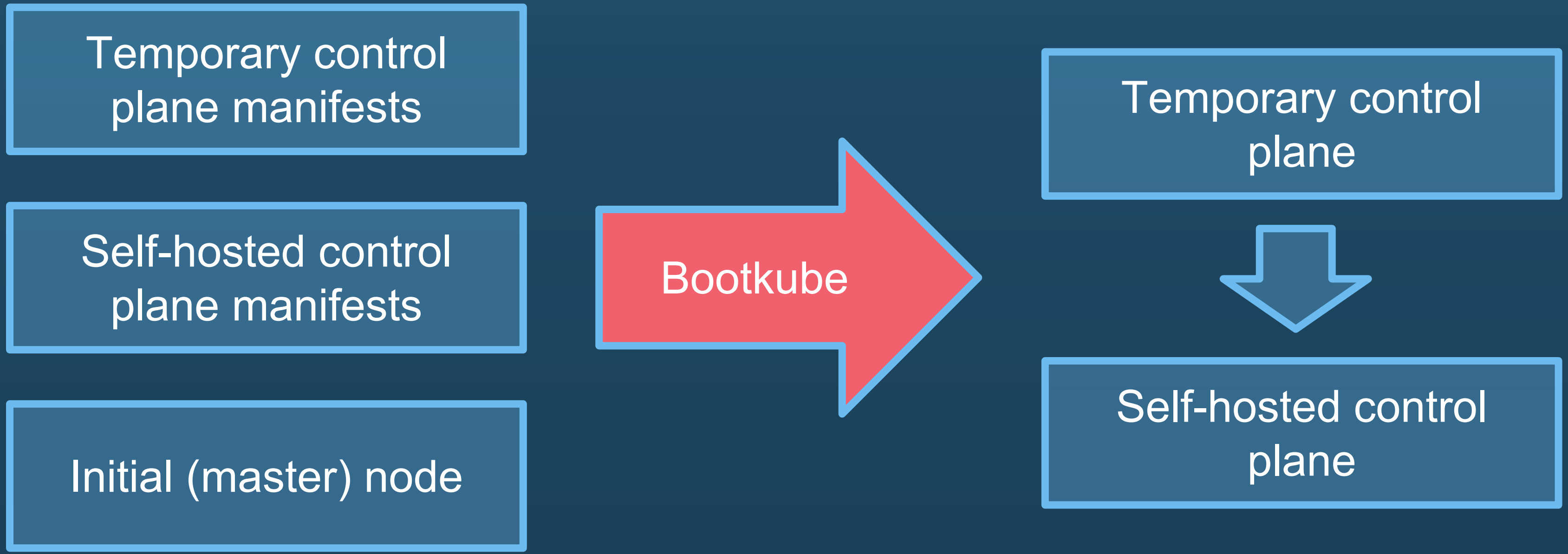
How it works: Bootstrapping

How it works: Bootstrapping

- Control plane runs as DaemonSets and Deployments…
- …but we need a control plane to create DaemonSets and Deployments

Clever Hack #1: Use a temporary, static control plane to bootstrap a self-hosted cluster

Bootkube



Temporary control plane manifests

Self-hosted control plane manifests

Initial (master) node

Bootkube

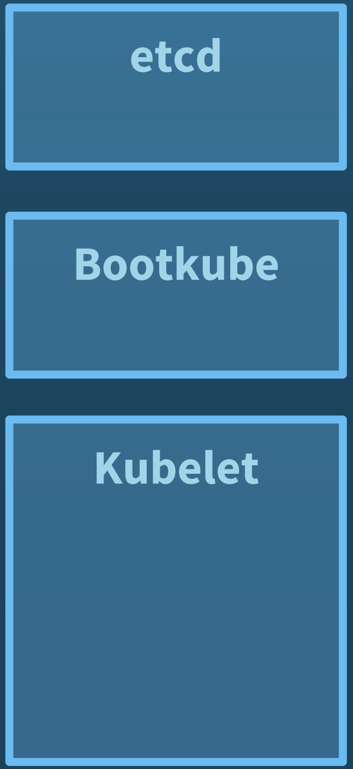
Temporary control plane

Self-hosted control plane

CoreOS https://github.com/kubernetes-incubator/bootkube

Bootstrapping illustrated

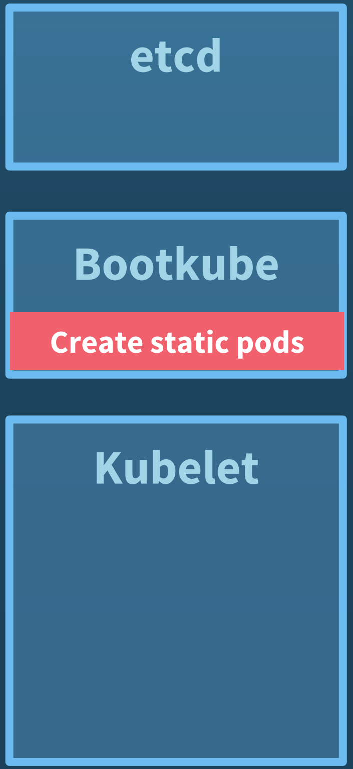
(Special thanks to Aaron Levy for the original version of these slides)



etcd

Bootkube

Kubelet

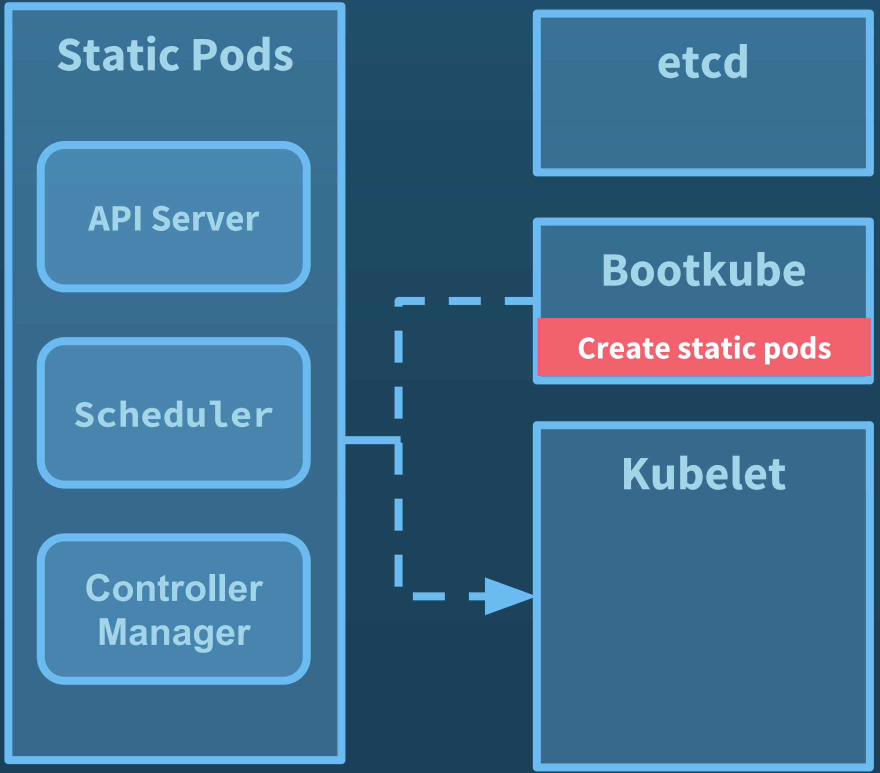


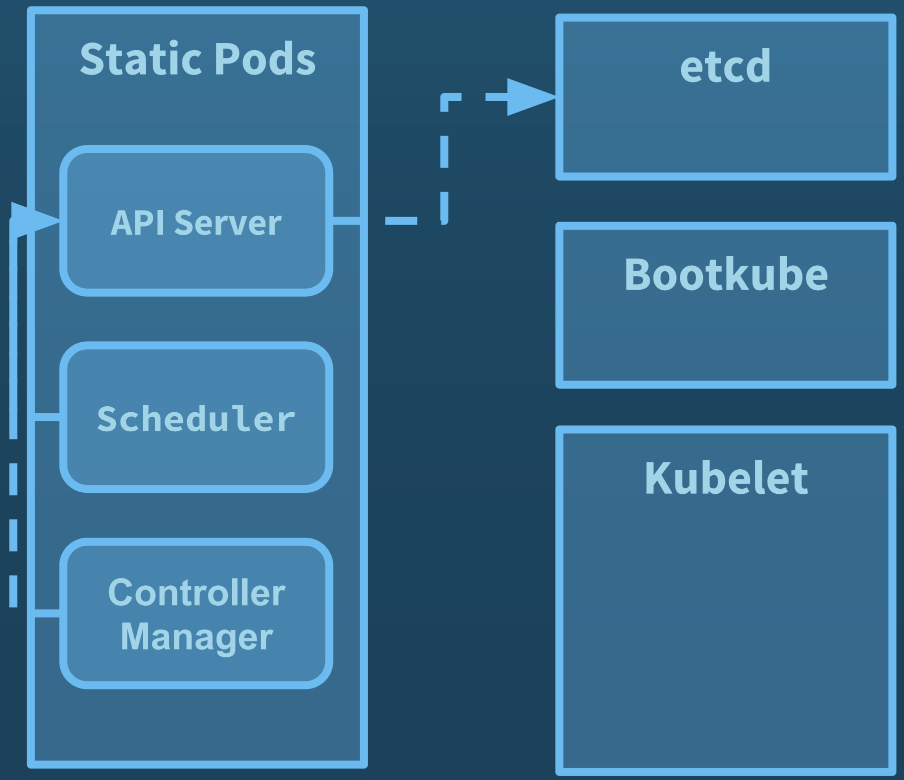
etcd

Bootkube

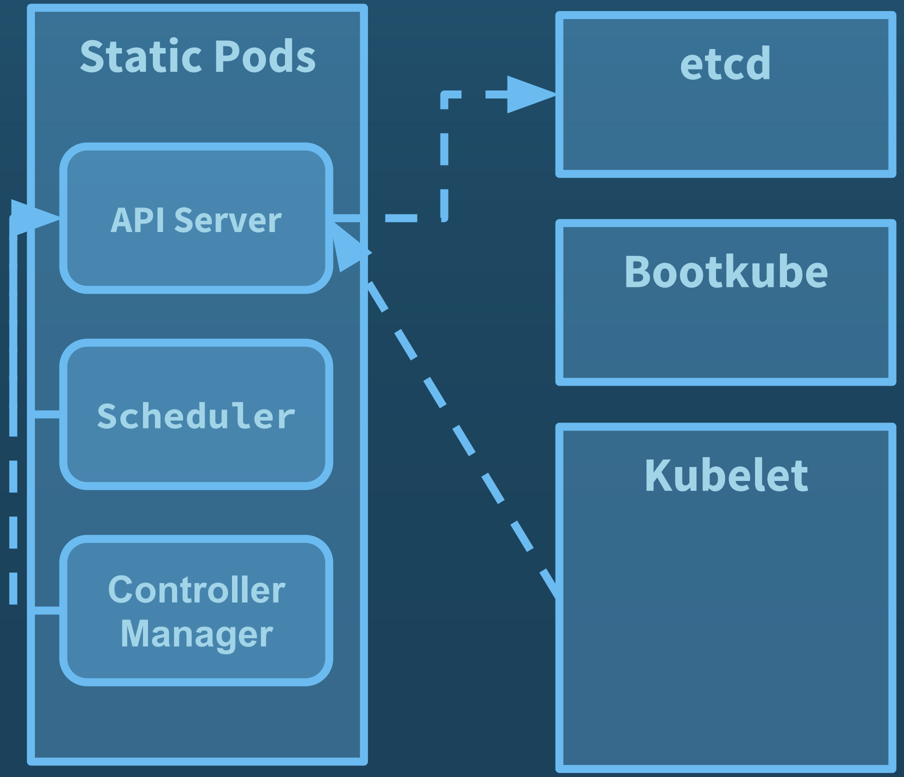
Create static pods

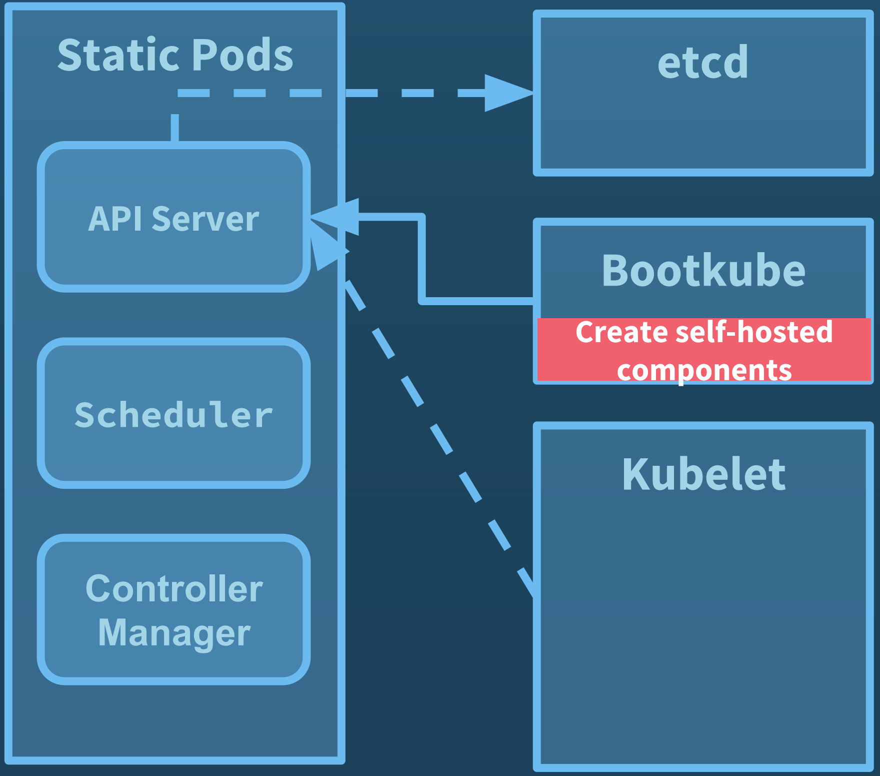
Kubelet


Static Pods
API Server
Scheduler
Controller Manager
etcd
Bootkube
Create static pods
Kubelet


Static Pods
API Server
Scheduler
Controller Manager
etcd
Bootkube
Kubelet

CoreOS


Static Pods
API Server
Scheduler
Controller Manager
etcd
Bootkube
Kubelet



Static Pods

API Server

Scheduler

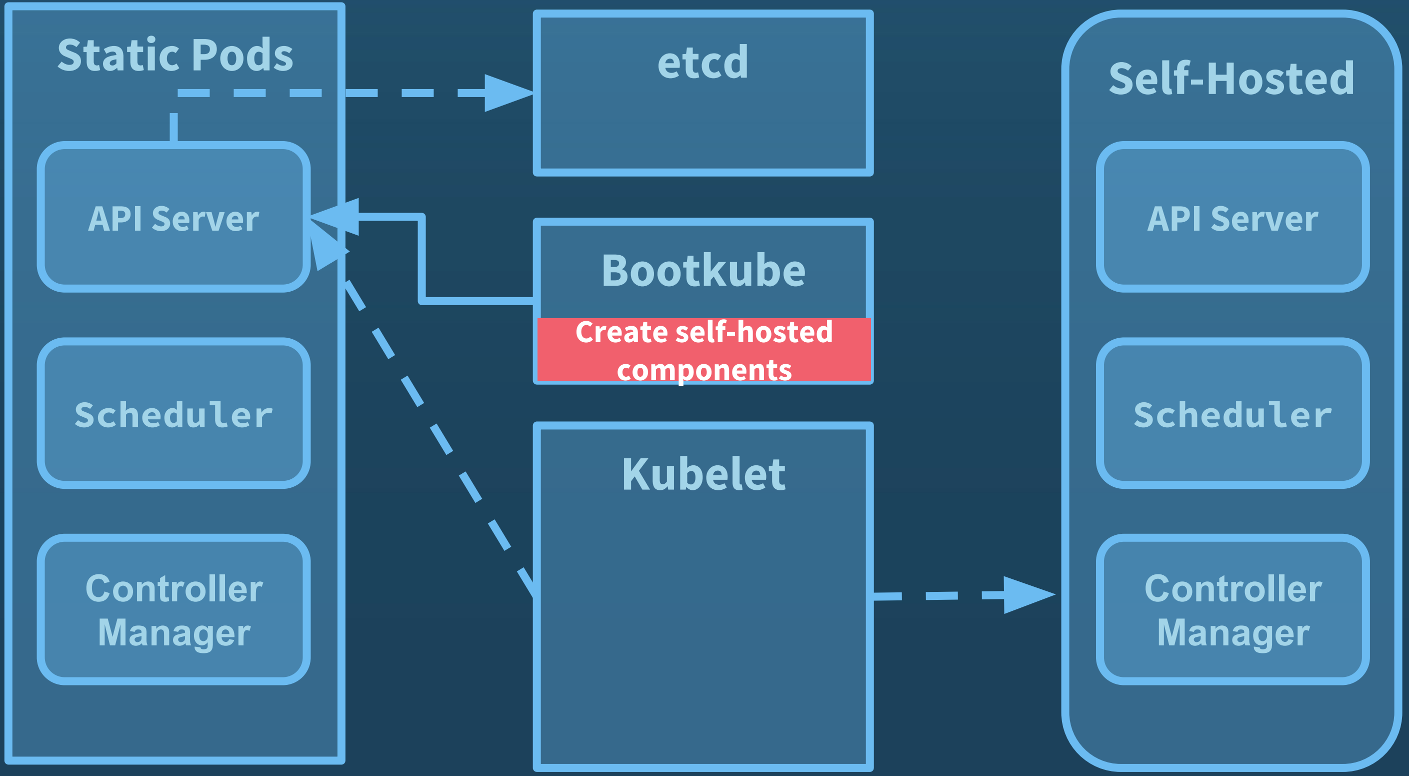
Controller Manager

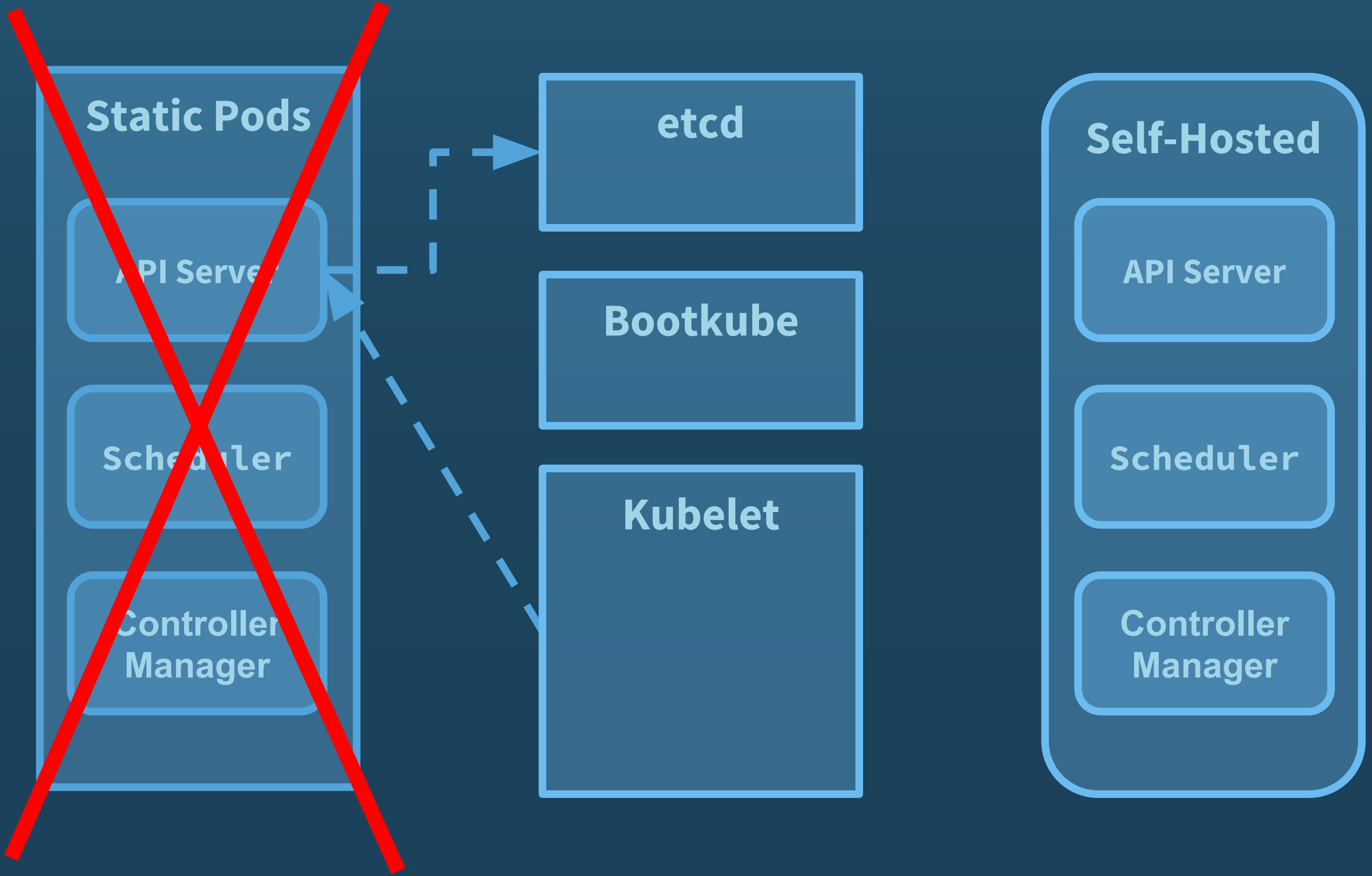
etcd

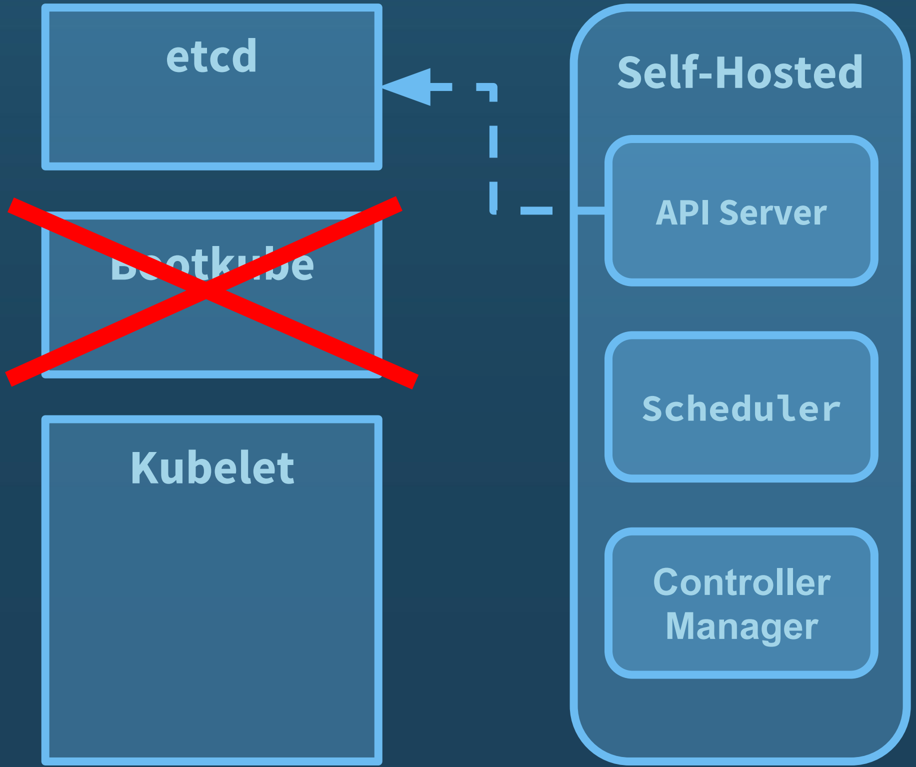
Bootkube

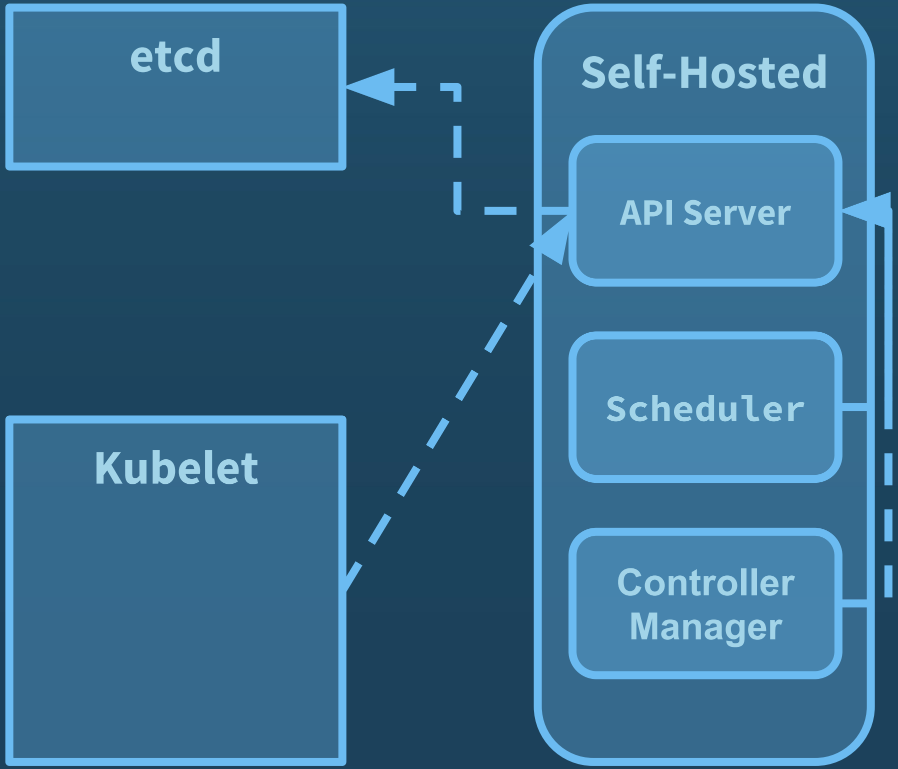
Create self-hosted components

Kubelet


Static Pods
API Server
Scheduler
Controller Manager
etcd
Bootkube
Create self-hosted components
Kubelet
Self-Hosted
API Server
Scheduler
Controller Manager


Static Pods
API Server
Scheduler
Controller Manager
etcd
Bootkube
Kubelet
Self-Hosted
API Server
Scheduler
Controller Manager


etcd
Bootkube
Kubelet
Self-Hosted
API Server
Scheduler
Controller Manager


etcd
Self-Hosted
API Server
Scheduler
Controller Manager
Kubelet

How it works: Upgrades

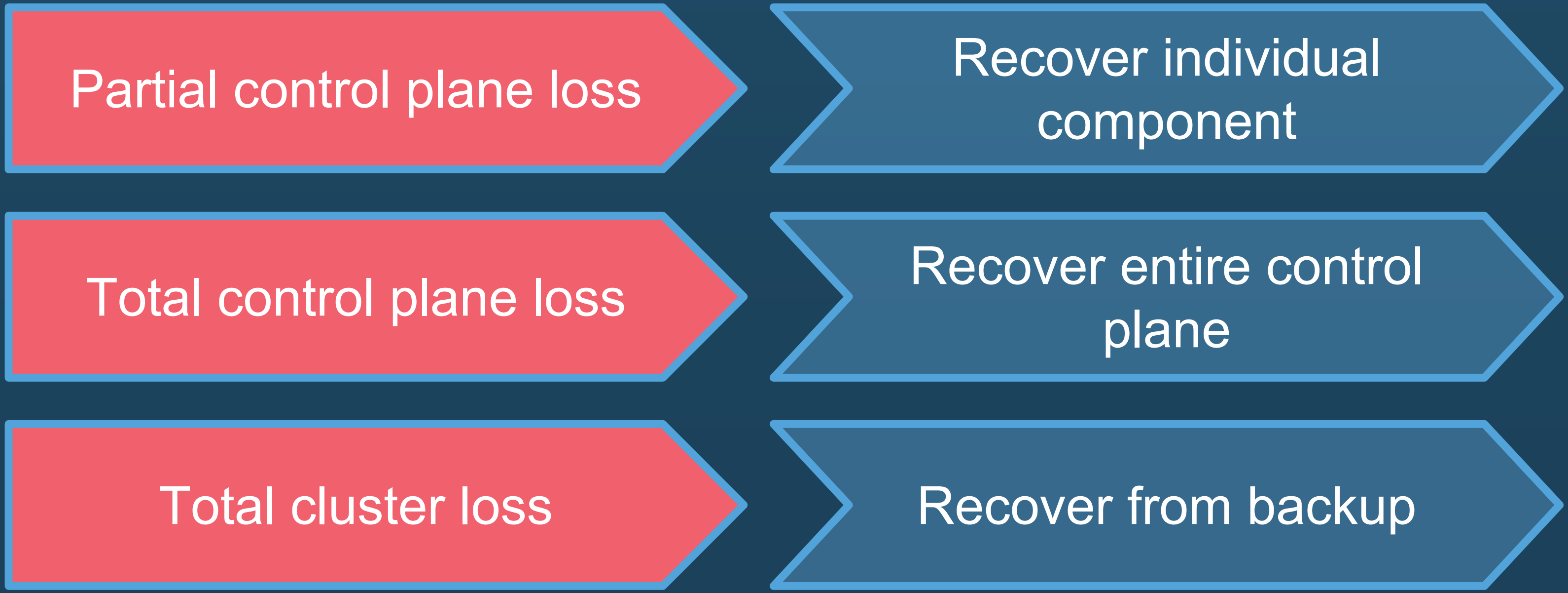
How it works: Upgrades

```
$ kubectl edit -n kube-system daemonsets/kube-apiserver
apiVersion: apps/v1beta2
kind: DaemonSet
metadata:
  name: kube-apiserver
  namespace: kube-system
spec:
  template:
    spec:
      containers:
      - name: kube-apiserver
        image: gcr.io/google_containers/hyperkube:v1.8.4
        command:
        - /hyperkube
        - apiserver
```

How it works: Disaster recovery

How it works: Disaster recovery

Failure modes:



Partial control plane loss → Recover individual component

Total control plane loss → Recover entire control plane

Total cluster loss → Recover from backup

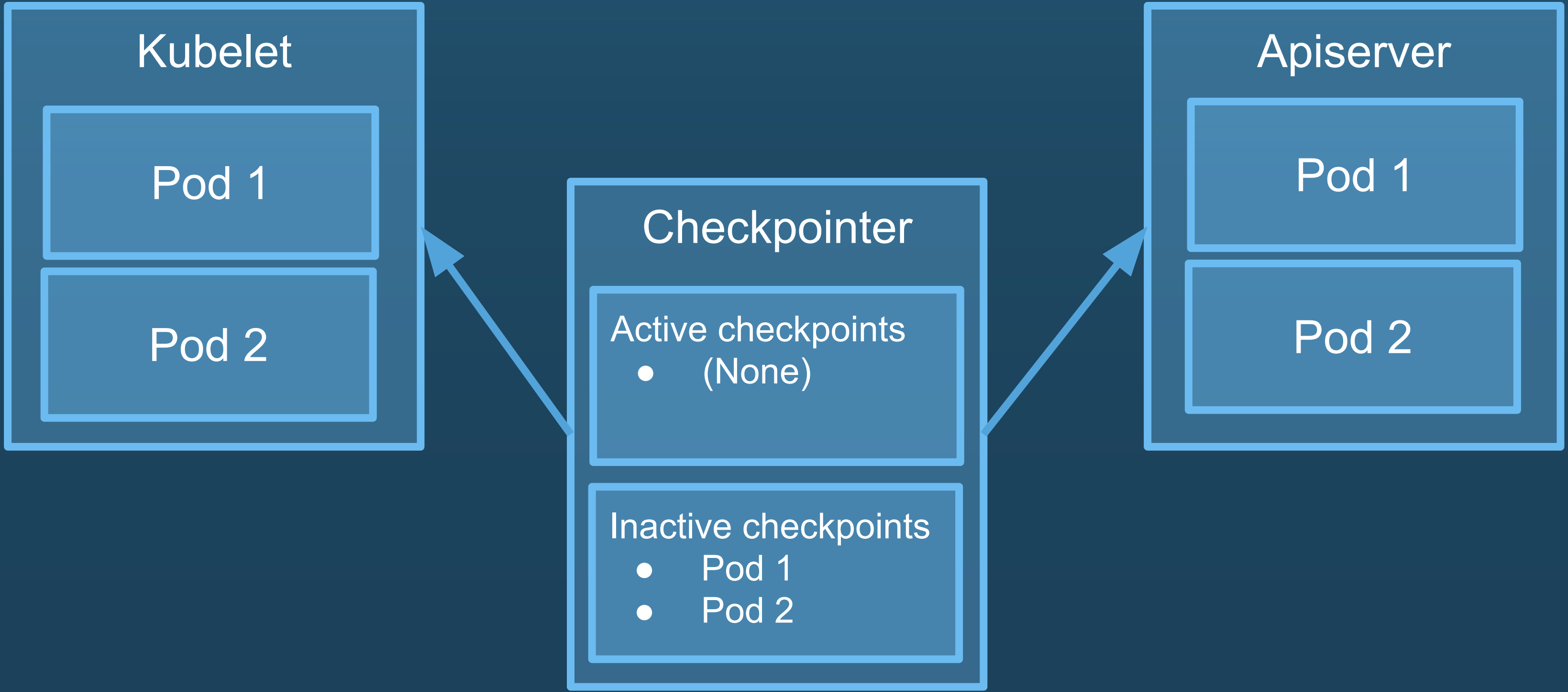
Pod checkpointer

- Keen observers may have noticed a trick during the upgrade demo

- How do you upgrade apiservers? How do you handle master node reboots?

Clever Hack #2: Run a “checkpointer” daemon to run static pods when the control plane is non-functional

Pod checkpointer: how it works



Kubelet

Pod 1

Pod 2

Checkpointer

Active checkpoints

- (None)

Inactive checkpoints

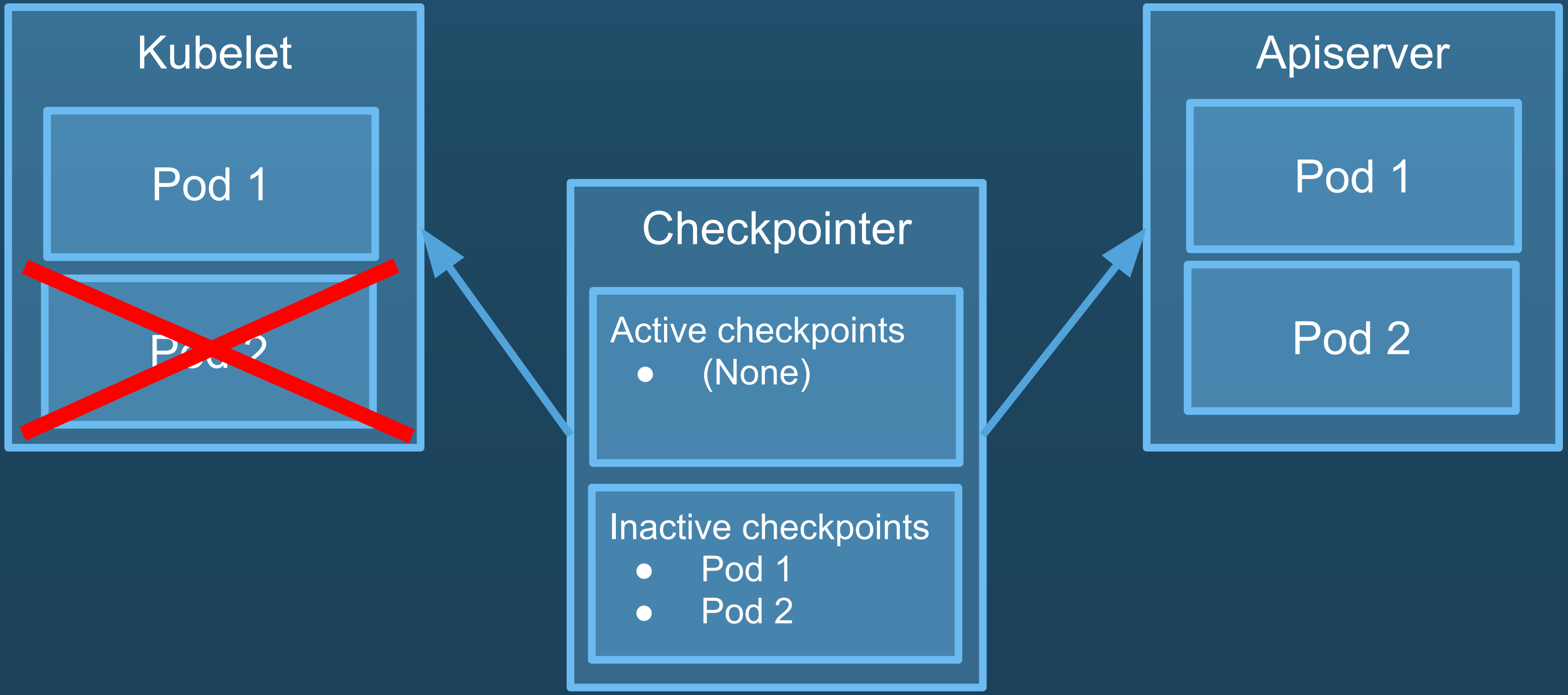
- Pod 1
- Pod 2

Apiserver

Pod 1

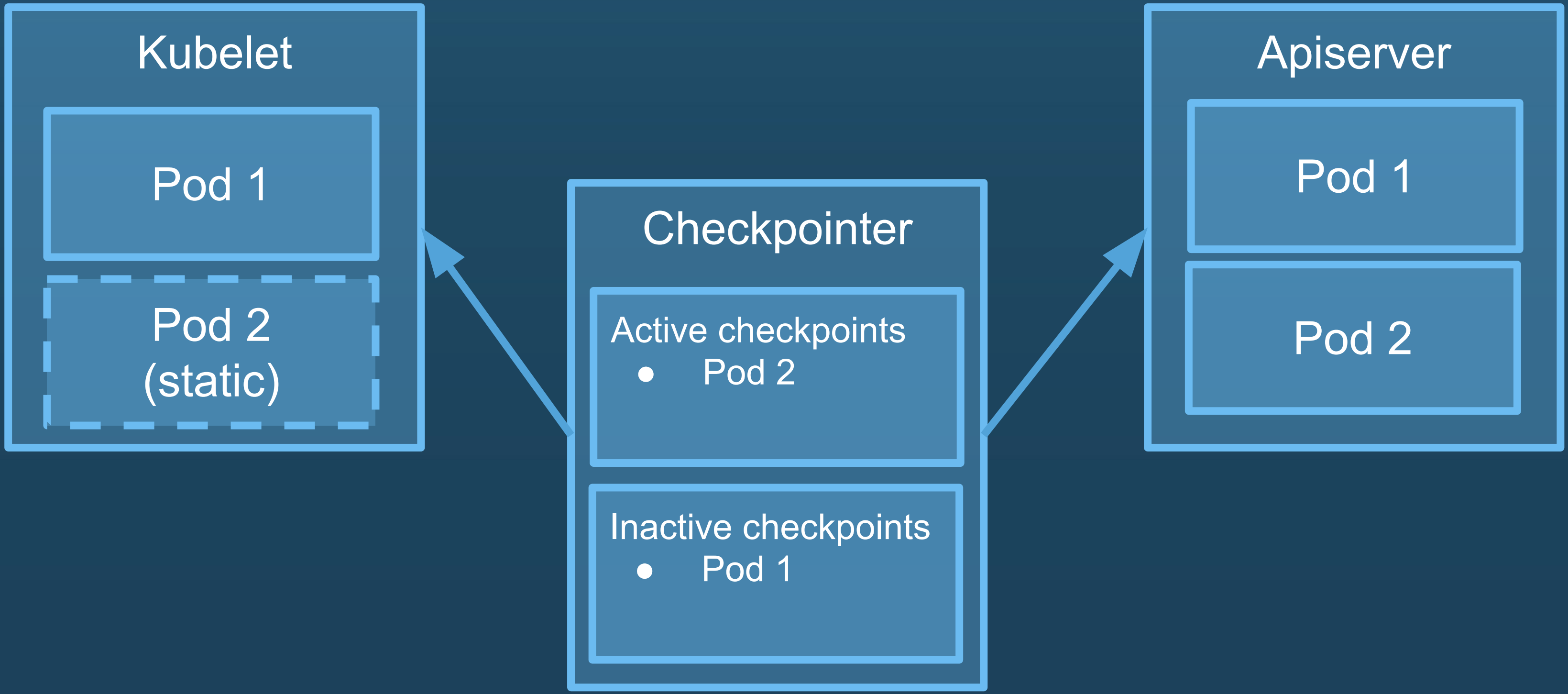
Pod 2

Pod checkpointer: how it works



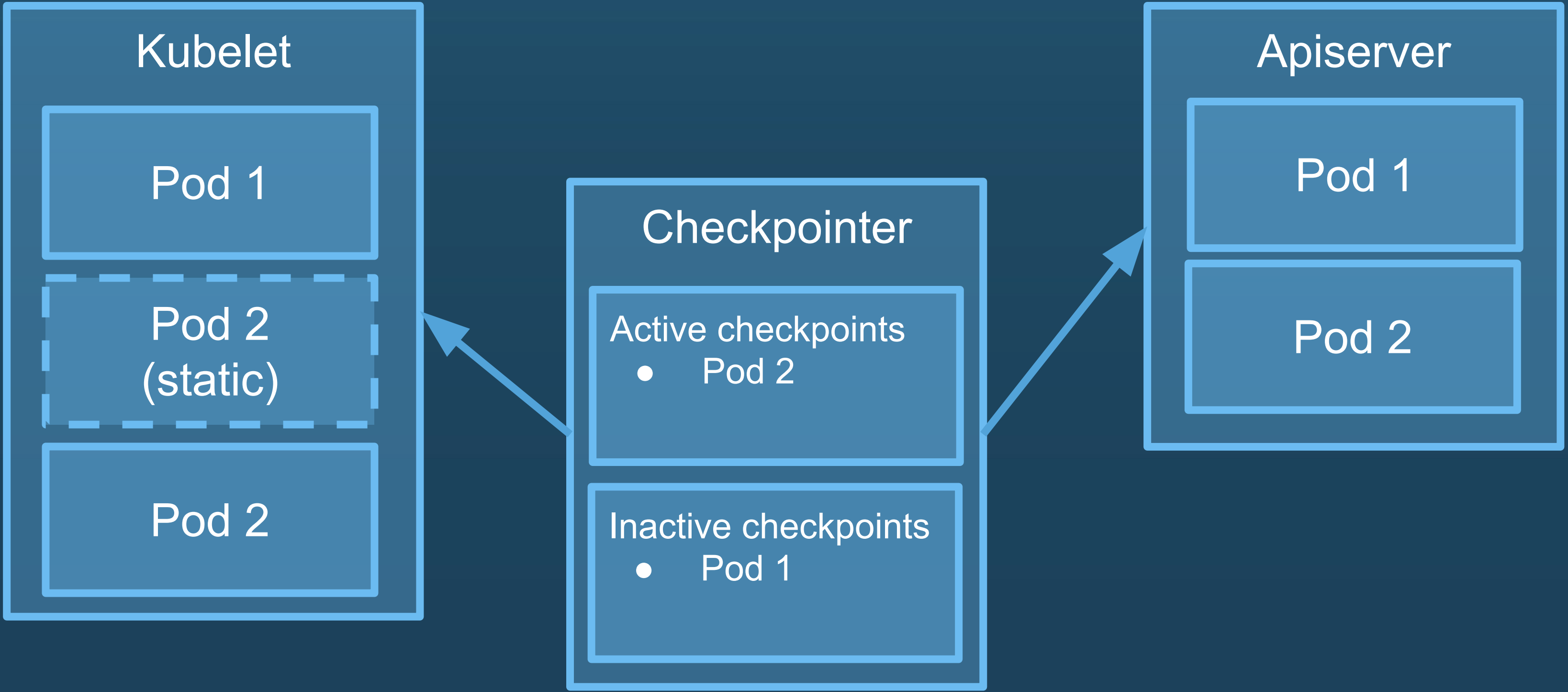
CoreOS

Pod checkpointer: how it works



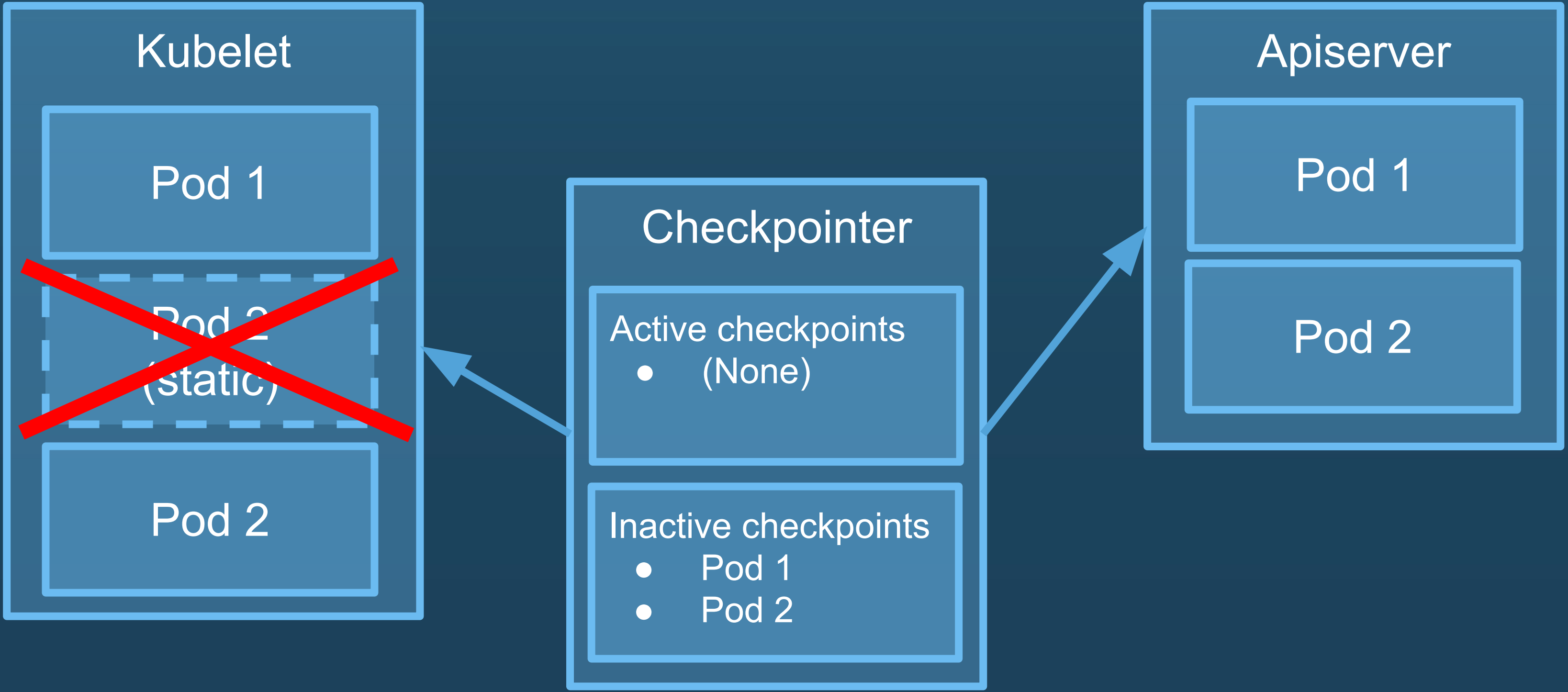
CoreOS

Pod checkpointer: how it works



CoreOS

Pod checkpointer: how it works



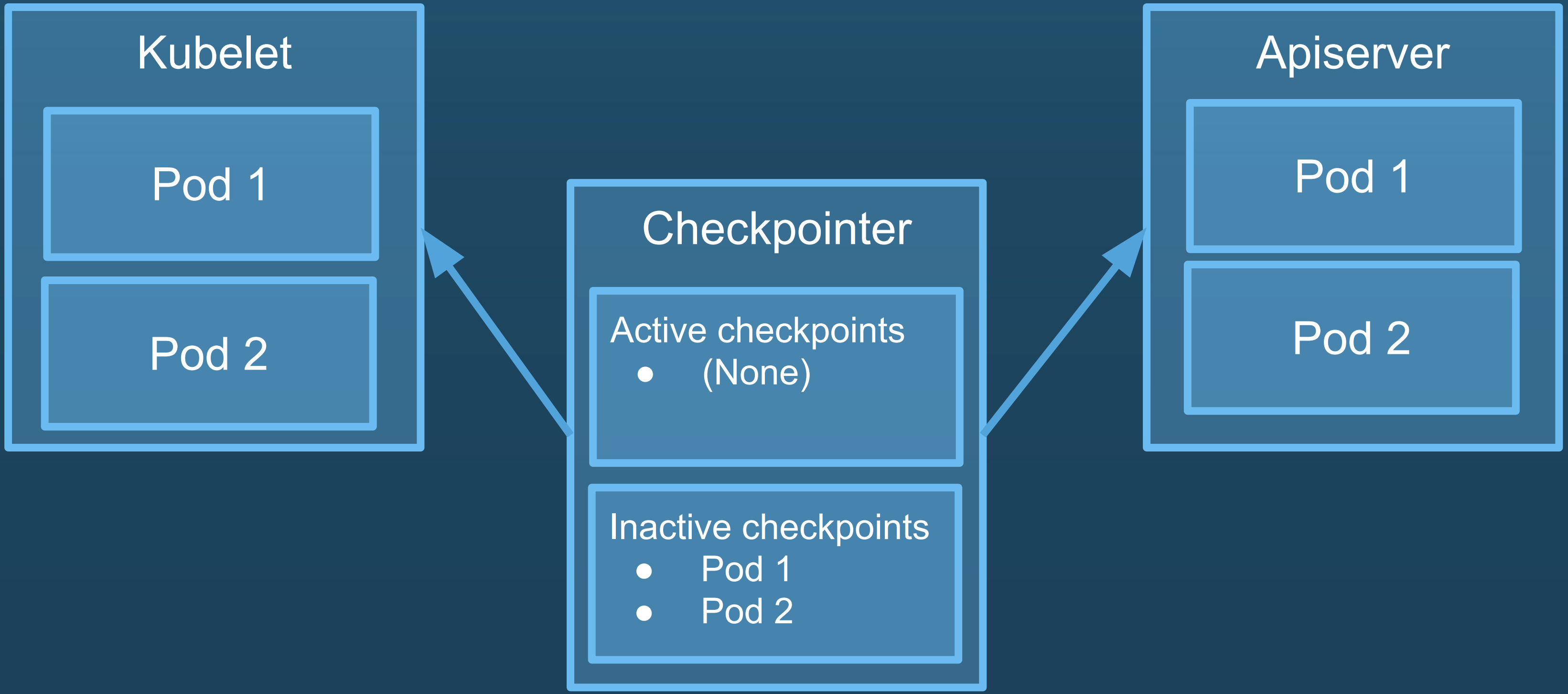
Active checkpoints

- (None)

Inactive checkpoints

- Pod 1
- Pod 2

Pod checkpointer: how it works



Kubelet

Pod 1

Pod 2

Checkpointer

Active checkpoints

- (None)

Inactive checkpoints

- Pod 1
- Pod 2

Apiserver

Pod 1

Pod 2

Bootkube Recover

- Checkpointer doesn't save us from all outages
  - e.g. need a functioning control plane to fix what's broken

- If only there was a way to "jumpstart" the cluster...

Clever Hack #3: Use bootkube to extract manifests and create another temporary control plane

Part 3: What’s next?

Automated operations

Cluster Upgrades

- Automate `kubectl apply` with `client-go`
- Fine-grained control over rollout and ordering
- Pre- and post-upgrade operations

Kubelet Upgrades

- Run node agent as a DaemonSet
- Install new Kubelet and container runtime

Configuration Management

- Evolve cluster configuration over time
- Change control plane settings? Sure!
- Deploy new network overlay? Sure!

Node management

Self-Healing

- Operator communicates with node agents to perform recovery operations

Autoscaling

- Provision/de-provision masters as needed
- Newly joining nodes can ask: “what should I be?”

Node Identity

- TLS bootstrapping provides identities to new nodes as they join the cluster
  - https://github.com/kubernetes-incubator/bootkube/pull/663

Self-hosting in upstream Kubernetes

- Kubeadm: support for self-hosted clusters
  - https://github.com/kubernetes/kubeadm/issues/127
- Kubelet: built-in pod checkpointer
  - https://github.com/kubernetes/features/issues/378
- Help needed! See #sig-cluster-lifecycle

Thanks!

QUESTIONS?

diegs@coreos.com

Github/Slack: @diegs

LONGER CHAT?

Let’s talk! Meet us at booth D2

More events: coreos.com/community