Kubernetes - Beyond a Black Box

A humble peek into one level below your running application in production

About The Author

- **Hao (Harry) Zhang**
- **Currently:** Software Engineer, LinkedIn, Team Helix @ Data Infrastructure
- **Previously:** Member of Technical Staff, Applatix, Worked with Kubernetes, Docker and AWS
- **Previous blog series:** “Making Kubernetes Production Ready”
  - **Part 1, Part 2, Part 3**
  - Or just Google “Kubernetes production”, “Kubernetes in production”, or similar
- **Connect with me on LinkedIn**

Copyright © 2017 Hao Zhang All rights reserved.

Motivation

- It's one of the hottest cluster management project on Github
- It has THE largest developer community
- State-of-art architecture designed mainly by people from Google who have been working on container orchestration for 10 years
- Collaboration of hundreds of engineers from tens of companies

Copyright © 2017 Hao Zhang All rights reserved.

Introduction

This set of slides is a humble dig into one level below your running application in production, revealing how different components of Kubernetes work together to orchestrate containers and present your applications to the rest of the world.

The slides contains **80+** external links to Kubernetes documentations, **blog posts, Github issues, discussions, design proposals, pull requests, papers, source code files** I went through when I was working with Kubernetes - which I think are valuable for people to understand how Kubernetes works, Kubernetes design philosophies and why these design came into places.

Copyright © 2017 Hao Zhang All rights reserved.

Outline - Part I

- A high level idea about what is Kubernetes
- Components, functionalities, and design choice
  - API Server
  - Controller Manager
  - Scheduler
  - Kubelet
  - Kube-proxy
- Put it together, what happens when you do `kubectl create`

Copyright © 2017 Hao Zhang All rights reserved.

Outline - Part II

- An analysis of controlling framework design
  - **Micro-service Choreography**
  - **Generalized Workload and Centralized Controller**
- Interfaces for production environments
- High level workload abstractions
  - Strength and limitations
- Conclusion

Copyright © 2017 Hao Zhang All rights reserved.

Part I

Overview

A high level idea about what is Kubernetes

“Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.”

– Kubernetes Official Website

Manage Apps, Not Machines

Manages Your Apps

- App Image Management
  - Where to **Run**
  - When to **Get**, when to **Run**, and when to **Discard**
- App Image Usage
  - Image Service: Image lifecycle management
  - Runtime: Run image as application

Copyright © 2017 Hao Zhang All rights reserved.

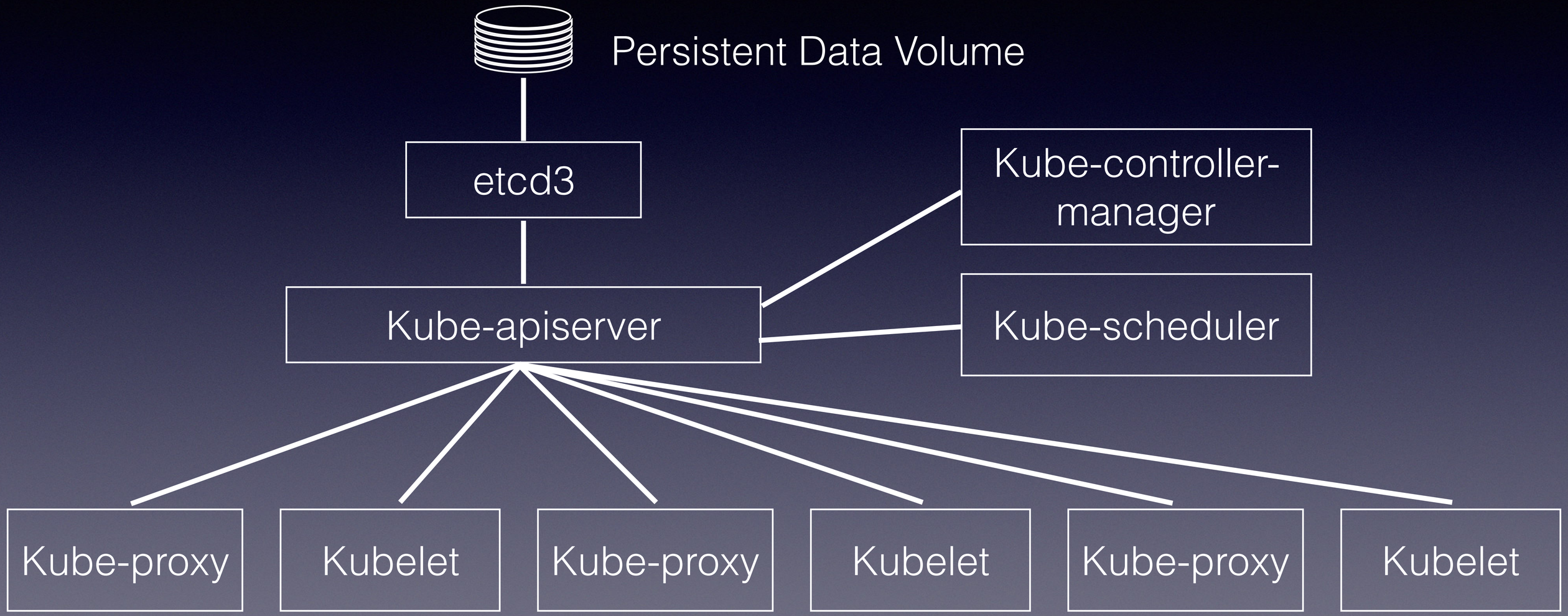
Manage Apps, Not Machines

Manages Things Around Your Apps

- High level workload abstractions
  - Pod, Job, Deployment, StatefulSet, etc.
  - “If container resembles **atom**, Kubernetes provides **molecules**” - Tim Hockin
- Storages
  - Data volumes
- Network
  - IP, Port mapping, DNS, …
- Monitoring, scaling, communicating, …

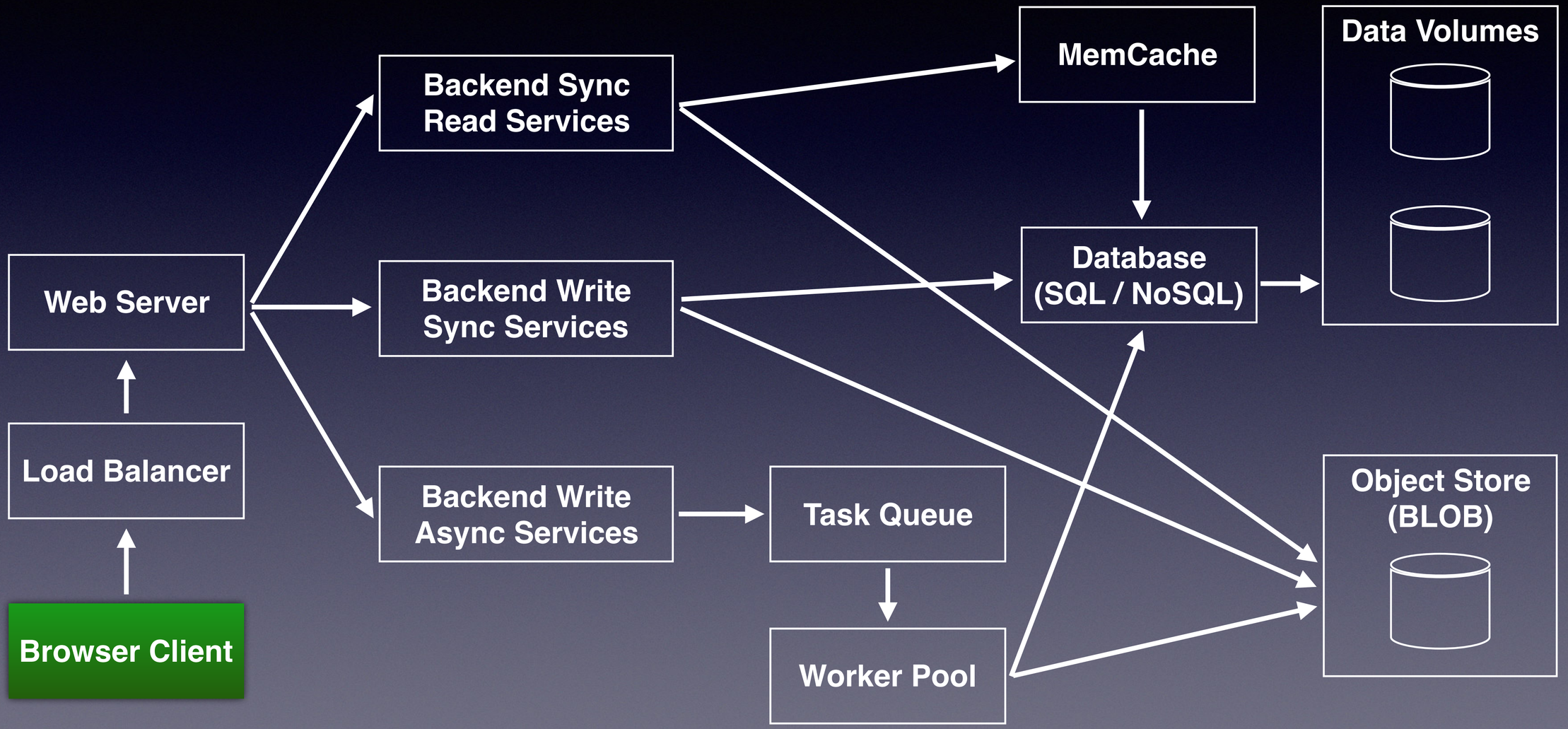
Copyright © 2017 Hao Zhang All rights reserved.

Kubernetes Control Plane



Copyright © 2017 Hao Zhang All rights reserved.

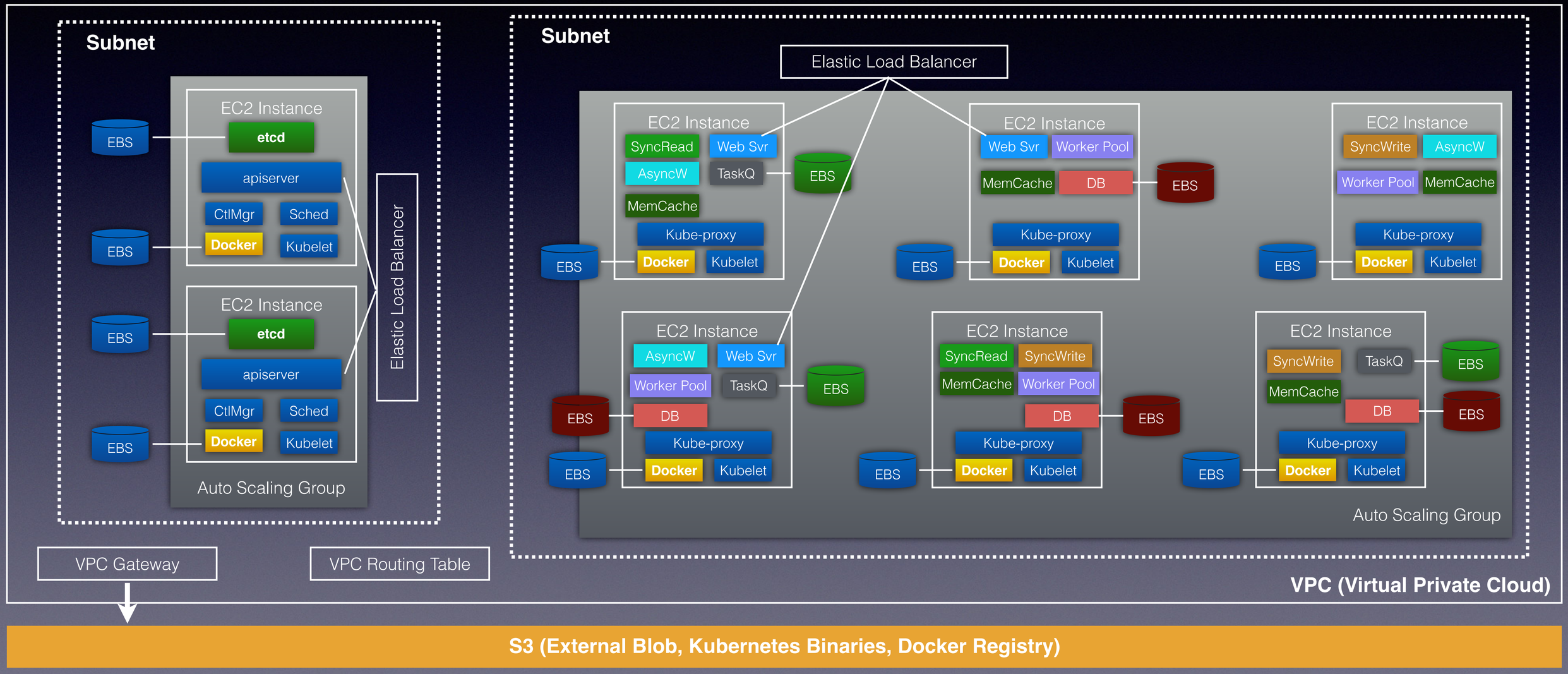
A Simplified Typical Web Service



Copyright © 2017 Hao Zhang All rights reserved.

Web Service On Kube (AWS)

Note: This chart is a rough illustration about how different production components might sit in cloud.
It does not reflect any strict production configurations such as high-availability, fault tolerance, machine load balancing etc.



S3 (External Blob, Kubernetes Binaries, Docker Registry)

Copyright © 2017 Hao Zhang All rights reserved.

Kubernetes Architecture

Components, Functionalities and Design Choice

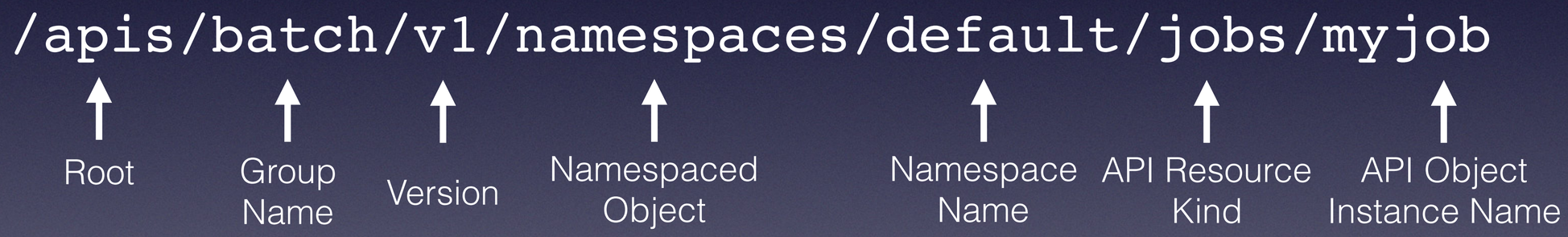
Kubernetes REST API

- Every piece of information in Kubernetes is an API object (a.k.a Cluster metadata)
  - Node, Pod, Job, Deployment, Endpoint, Service, Event, PersistentVolume, ResourceQuota, Secrets, ConfigMap, ……
  - Schema and data model explicitly defined and enforced
  - Every API object has a UID and version number
- Micro-services can perform CRUD on API objects through REST
- Desired state in cluster is achieved by **collaborations of separate autonomous entities reacting on changes of one or more API object(s) they are interested in**

Copyright © 2017 Hao Zhang All rights reserved.

Kubernetes REST API

- The API Space
  - Different API Groups
    - Usually a group represents a set of features, i.e. batch, apps, rbac, etc.
  - Each group has its own version control



More Readings:
- Extending APIs design spec: https://github.com/kubernetes/community/blob/master/contributors/design-proposals/api-machinery/extending-api.md
- Customer Resource Documentation: https://kubernetes.io/docs/tasks/access-kubernetes-api/extend-api-custom-resource-definitions/
- Extending core API with aggregation layer: https://kubernetes.io/docs/concepts/api-extension/apiserver-aggregation/
- Kubernetes API Spec: https://raw.githubusercontent.com/kubernetes/kubernetes/release-1.7/api/openapi-spec/swagger.json
- Kubernetes API Server Deep Dives:
  - https://blog.openshift.com/kubernetes-deep-dive-api-server-part-1/
  - https://blog.openshift.com/kubernetes-deep-dive-api-server-part-2/

Copyright © 2017 Hao Zhang All rights reserved.

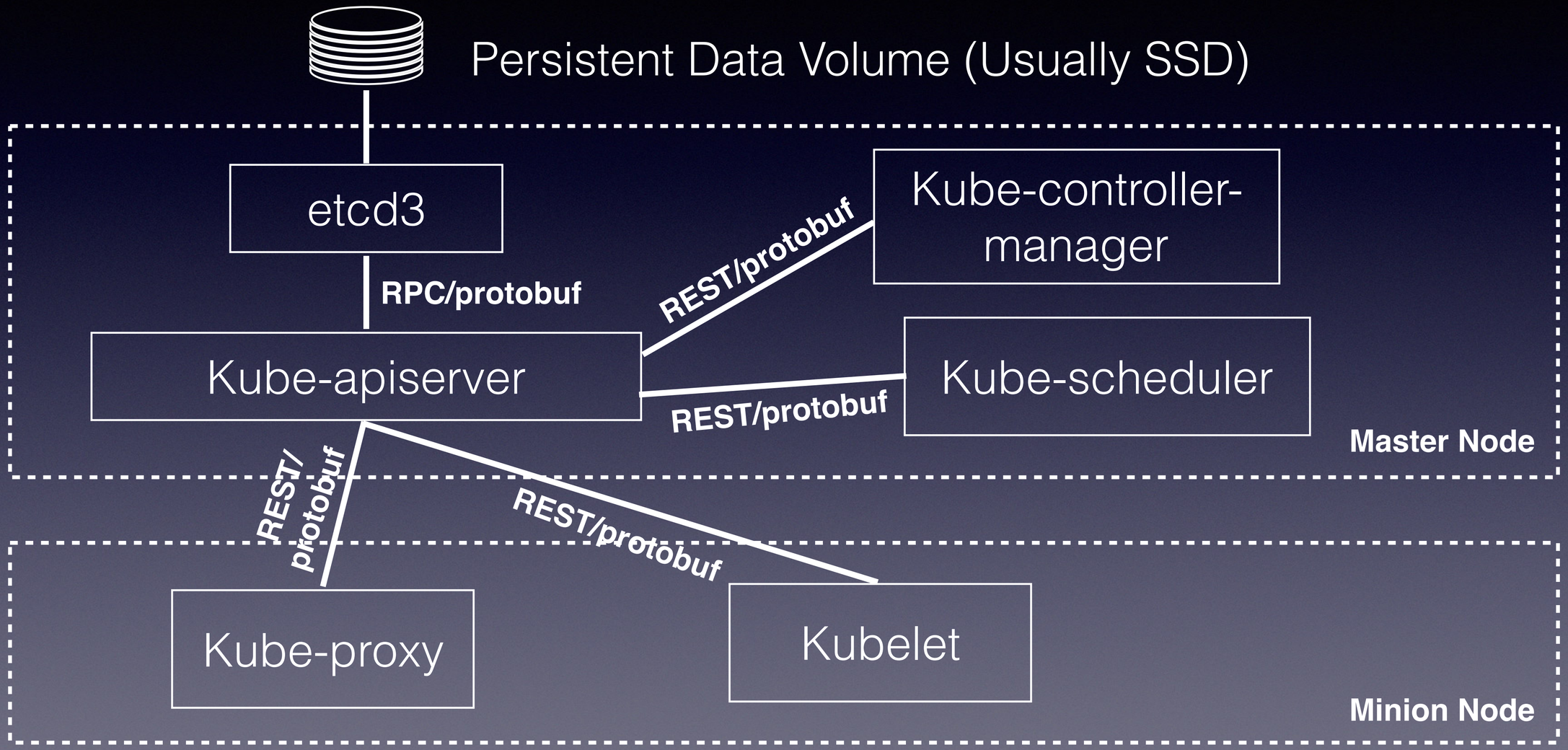
Kubernetes REST API

- The API Object Definition
  - 5 major fields: apiVersion, kind, metadata, spec and status
    - A few exceptions such as v1Event, v1Binding, etc
  - **Metadata** includes name, namespace, label, annotation, version, etc
  - ***Spec*** describes **desired state**, while ***status*** describes **current state**

```
$ kubectl get jobs myjob --namespace default --output yaml
apiVersion: batch/v1
kind: job
metadata:
    name: myjob
    namespace: default
spec:
    (Describing desired “job” object, i.e. Pod template, how many runs, etc.)
status:
    (“job” object current status, i.e. ran # times already, etc.)
```

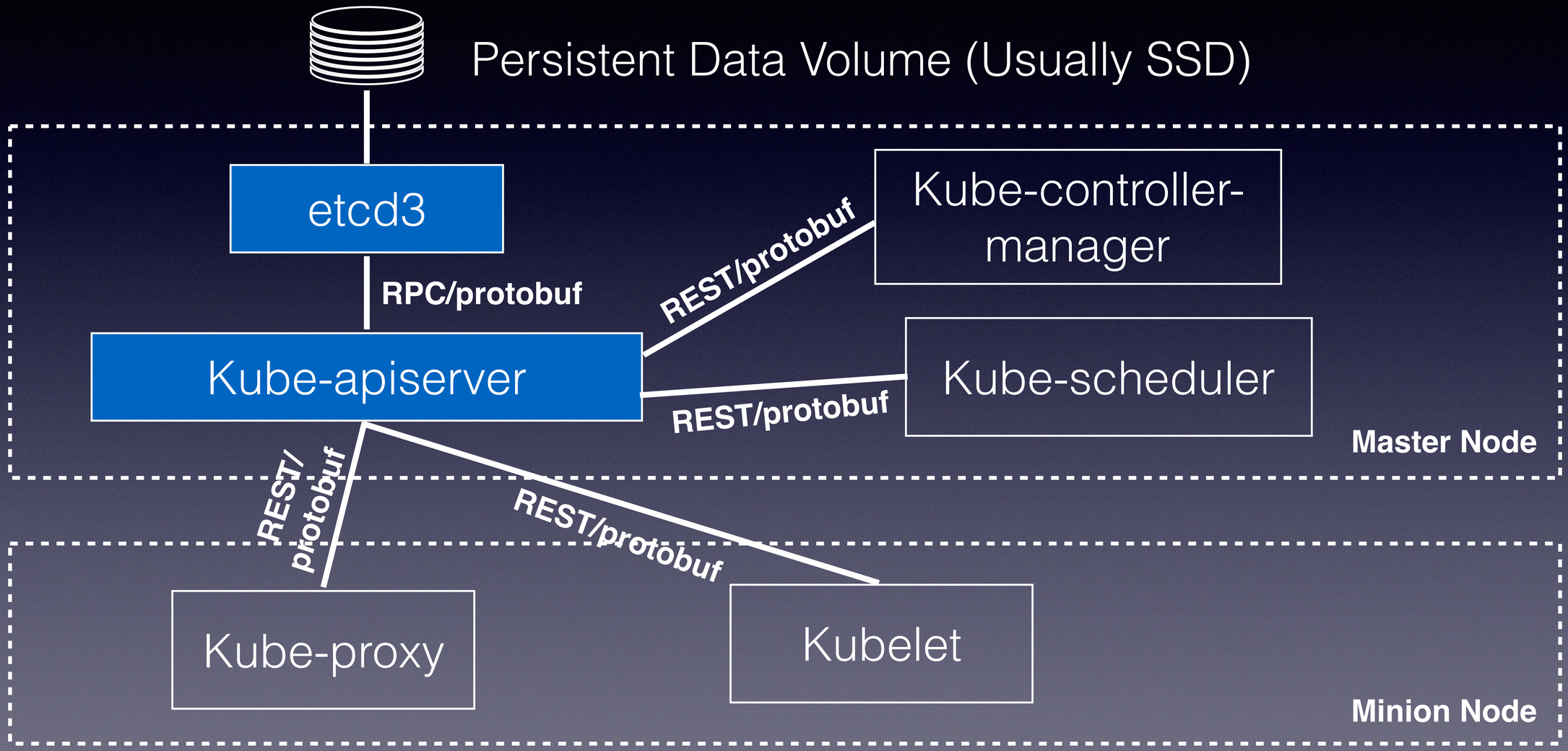
Copyright © 2017 Hao Zhang All rights reserved.

Kubernetes Control Plane



Copyright © 2017 Hao Zhang All rights reserved.

Kube-apiserver & Etcd



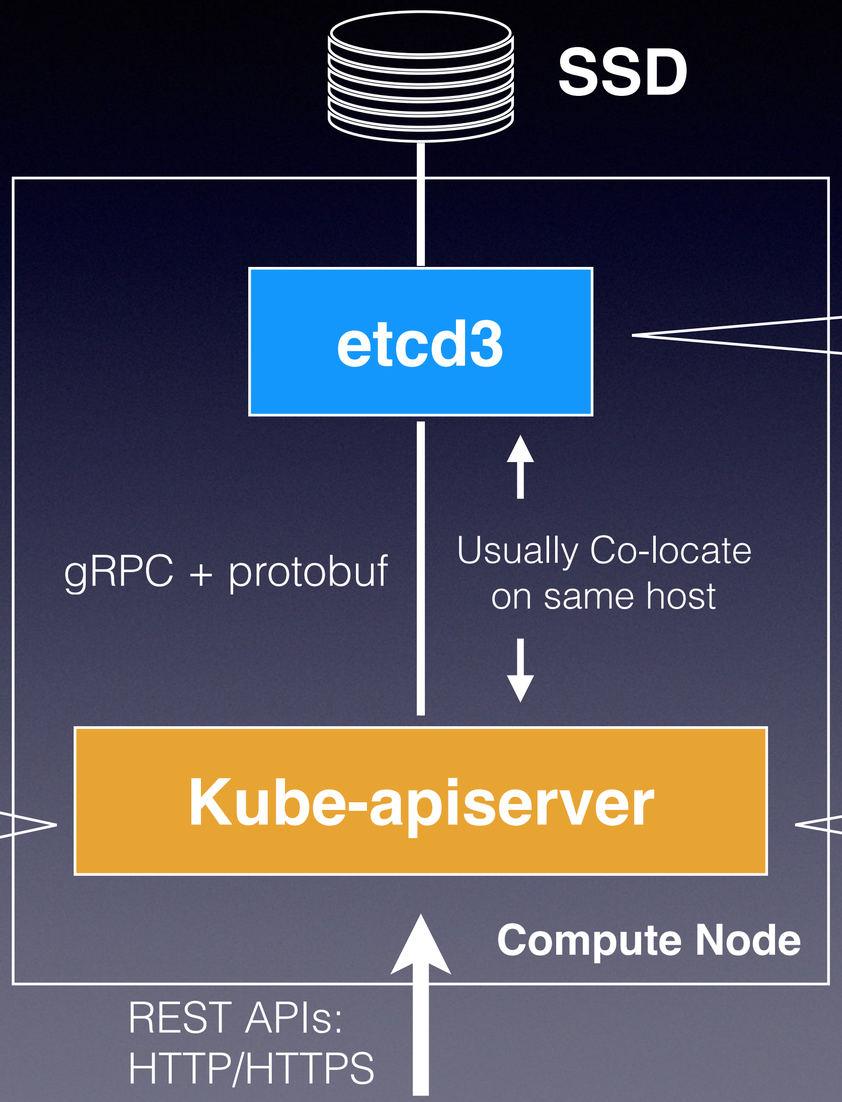
Copyright © 2017 Hao Zhang All rights reserved.

Kube-apiserver & Etcd

General Functionalities:
- Authentication
- Admission control (RBAC)
- Rate limiting
- Connection pooling
- Feature grouping
- Pluggable storage backend
- Pluggable customer API definition
- Client tracing and API call stats

For API Objects:
- CRUD Operations
- Defaulting
- Validation
- Versioning
- Caching

For Containers:
- Application Proxying
- Stream Container Logs
- Execute Command in Container
- Metrics



Etcd as metadata store:
- Watchable distributed K-V store
- Transaction and locking support
- Scalable and low memory consumption
- RBAC support
- Uses Raft for leader election
- R/W on all nodes of Etcd cluster
- …

WRITE API Call Process Pipeline:
1. Route through mux to the routine
2. Version conversions
3. Validation
4. Encode
5. Storage backend communication

READ API calls are process roughly opposite of WRITE API calls.

Copyright © 2017 Hao Zhang All rights reserved.

Kube-apiserver & Etcd

- API Server caching
  - Decoding object from Etcd is a huge burden
    - Without caching, 50% CPU spent on decoding objects read from etcd, 70% of which is on convert K/V (as of Etcd2)
    - Also adds pressure on Golang's GC (Lots of unused K/V)
  - In-memory Cache for READs (Watch, Get and List)
    - Decode only when etcd has more up-to-date version of the object

Copyright © 2017 Hao Zhang All rights reserved.

Kube-apiserver & Etcd

Citation and More Readings:

- Etcd! Etcd3!
  - Discussion about pluggable storage backend (ZK, Consul, Redis, etc): https://github.com/kubernetes/kubernetes/issues/1957
  - More on “Why etcd?” (Could be biased): https://github.com/coreos/etcd/blob/master/Documentation/learning/why.md
  - Upgrade to etcd3: https://github.com/kubernetes/kubernetes/issues/22448
  - Etcd3 Features: https://coreos.com/blog/etcd3-a-new-etcd.html

- Kubernetes & Etcd
  - Kubernetes’ Etcd usage: http://cloud-mechanic.blogspot.com/2014/09/kubernetes-under-hood-etcd.html
  - Raft - the consensus protocol behind etcd: https://speakerdeck.com/benbjohnson/raft-the-understandable-distributed-consensus-protocol/
  - Protocol Buffer: https://developers.google.com/protocol-buffers/

- Setting up HA Kubernetes cluster: https://kubernetes.io/docs/admin/high-availability/#clustering-etcd

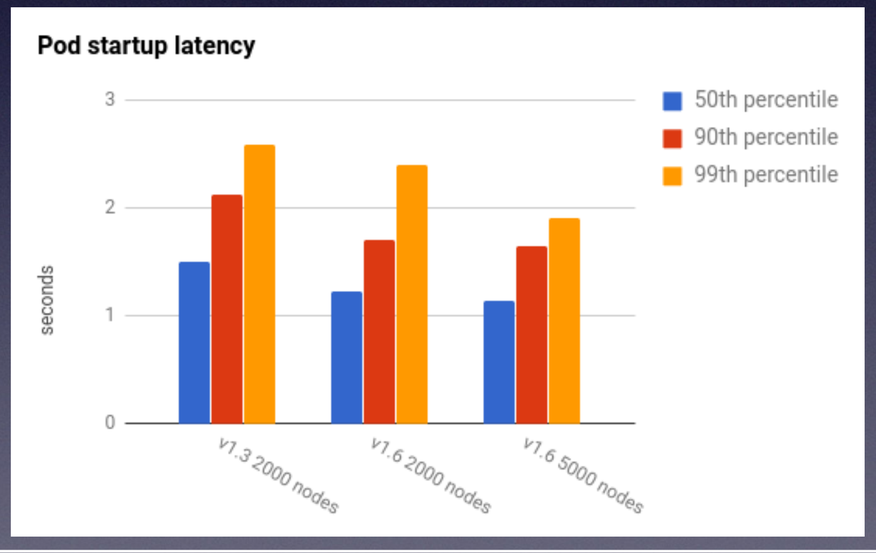
- Kubernetes API
  - Kubernetes API Convention: https://github.com/kubernetes/community/blob/master/contributors/devel/api-conventions.md

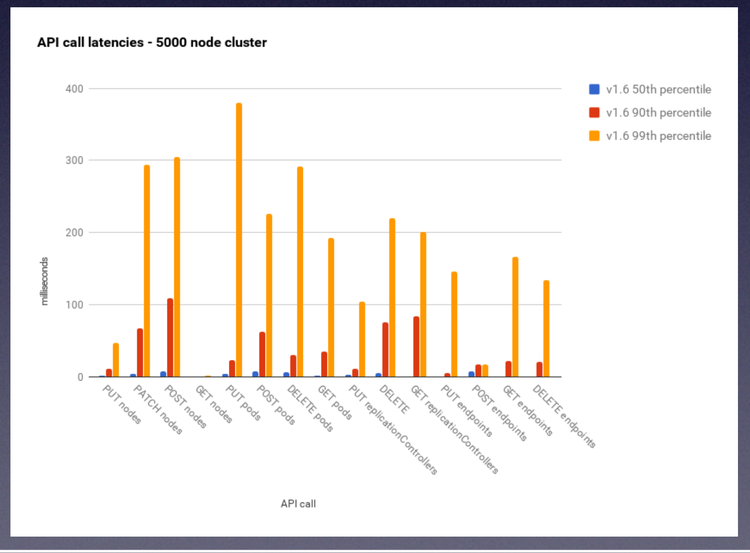
- API Server Optimizations and Improvements
  - Discussion about cache: https://github.com/kubernetes/kubernetes/issues/6678, Watch Cache Design Proposal
  - Discussion about optimizing etcd connection: https://github.com/kubernetes/kubernetes/issues/7451
  - Optimizing GET, LIST, and WATCH: https://github.com/kubernetes/kubernetes/issues/4817, https://github.com/kubernetes/kubernetes/issues/6564
  - Serve LIST from memory: https://github.com/kubernetes/kubernetes/issues/15945
  - Optimizing get many pods: https://github.com/kubernetes/kubernetes/issues/6514

Copyright © 2017 Hao Zhang All rights reserved.

Kube-apiserver & Etcd

- Kubernetes' Performance SLO
  - API Responsiveness: 99% API calls < 1s
  - Pod Startup Time: 99% Pods and their containers start < 5s (With pre-pulled images)
  - 5000 Nodes, 150,000 Pods, ~300,000 Containers (1~1.5M API objects)
- Resource (Single master set up for 1000 nodes as of v1.2): 32 Cores, 120G Memory





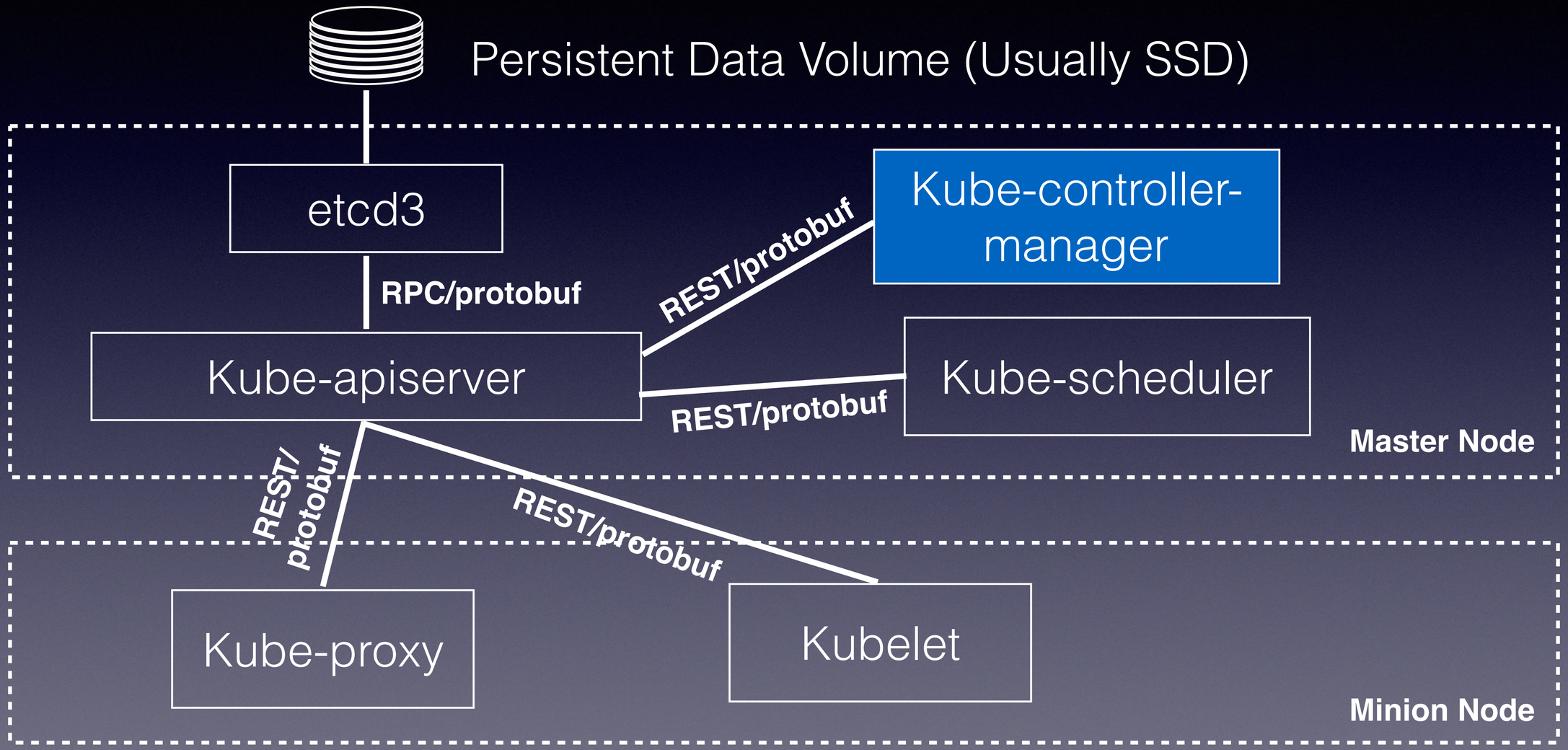
Citation and More Readings:

✤ Kubernetes 1.2 Scalability: http://blog.kubernetes.io/2016/03/1000-nodes-and-beyond-updates-to-Kubernetes-performance-and-scalability-in-12.html

✤ Kubernetes 1.6 Scalability: http://blog.kubernetes.io/2017/03/scalability-updates-in-kubernetes-1.6.html

Copyright © 2017 Hao Zhang All rights reserved.

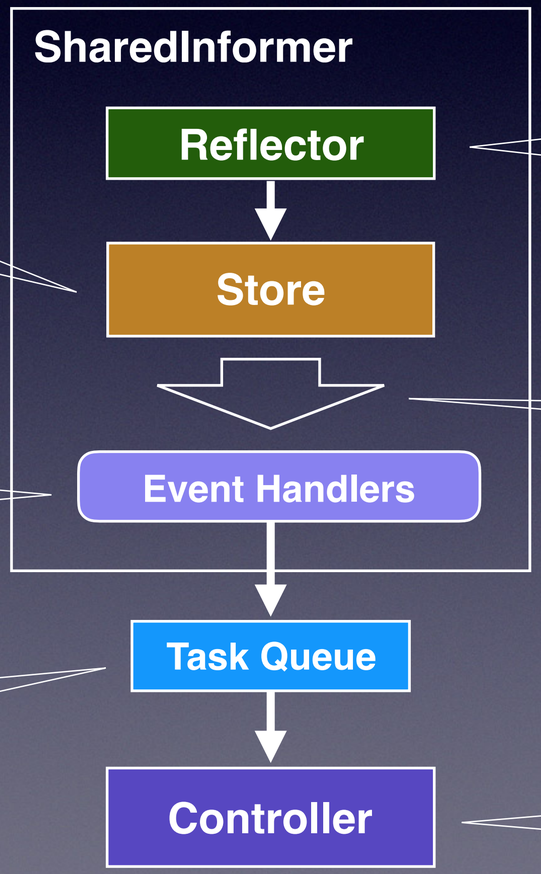
Kube-controller-manager



Copyright © 2017 Hao Zhang All rights reserved.

Kube-controller-manager

- Framework: Reflector, Store, SharedInformer, and Controller
  - One of the optimizations in 1.6 to make it scale from 2k to 5k nodes



- In memory read-only cache
  - Shared among informers
  - Optimizes GET and LIST

- ListAndWatch on a particular resource
  - All instance of a resource
  - List upon startup and connection broken
  - List: get current states
  - Watch: get updates
  - Push ListAndWatch results into store
  - Store is always up-to-date

- Controller registers event handlers to the shared informer of the resource it is interested in
- A single SharedInformer has a pool of event handlers registered by different controllers

- Cache Update Broadcast
  - Events: OnAdd, OnUpdate, OnDelete
  - Store is **at least as fresh** as notification

- Blocking, MT Safe FIFO Queue
  - Additional feature such as delayed scheduling, failure counts, etc.

- Worker pool of a particular resource
  - Reconciliation controlling loops
  - Action is computed based on DesiredState and CurrentState

Copyright © 2017 Hao Zhang All rights reserved.

Kube-controller-manager

- Reconciliation Controlling Loop: Things will finally go right

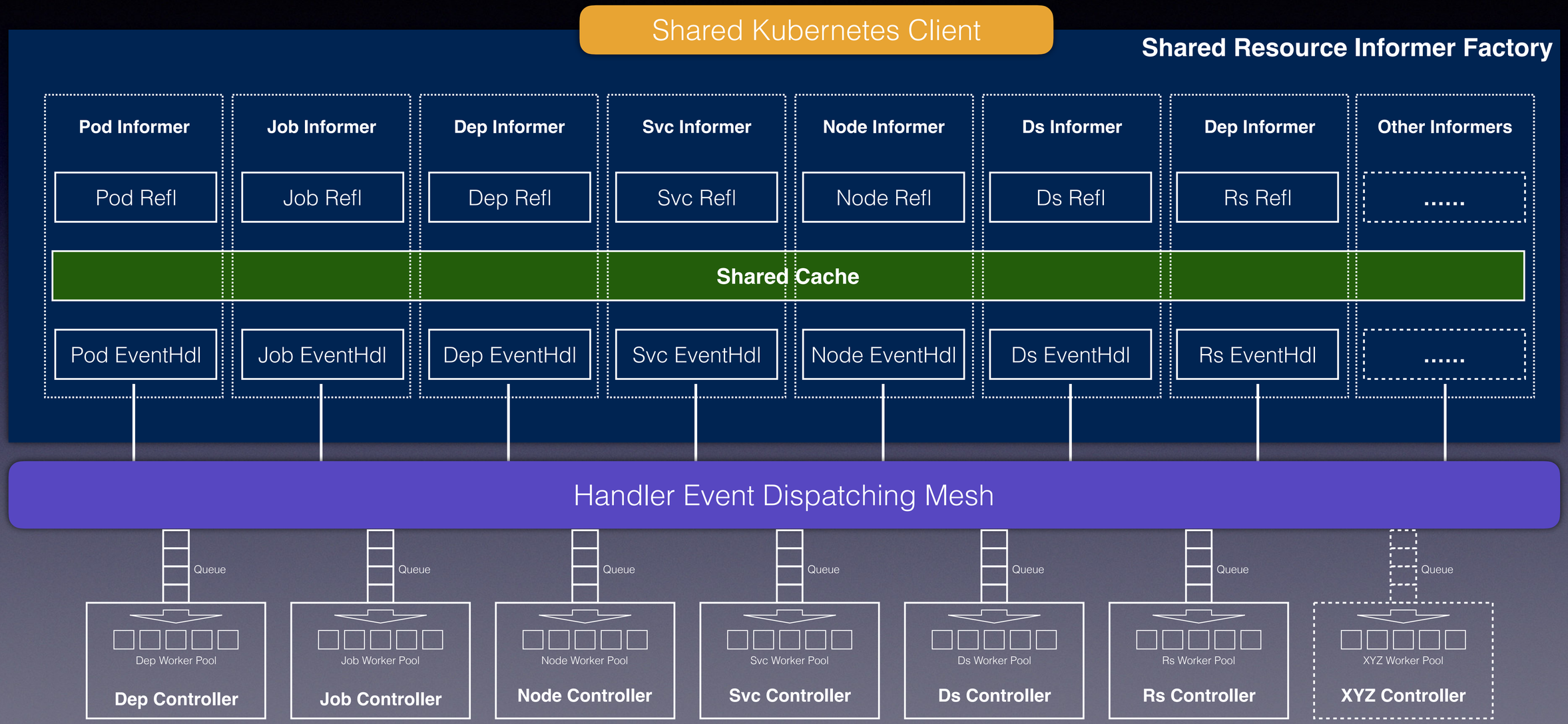
```
func (c *Controller) Run(poolSize int, stopCh chan struct{}) {
    // start worker pool
    for i := 0; i < poolSize; i++ {
        // runWorker will loop until "something bad" happens. The .Until will
        // then rekick the worker after one second
        go wait.Until(c.runWorker, 1 * time.Second, stopCh)
    }
    // wait until we're told to stop
    <-stopCh
}

func (c *Controller) runWorker() {
    // hot loop until we're told to stop.
    for c.processNextWorkItem() {}
}

func (c *Controller) processNextWorkItem() {
    obj = c.queue.Get()
    // Reconciliation between current and desired state
    err = c.makeChange(obj.CurrentState, obj.DesiredState)
    if !err {
        return
    }
    // Cool down and retry using delayed scheduling upon error
    c.queue.AddRateLimited(obj)
    return
}
```

Copyright © 2017 Hao Zhang All rights reserved.

Kube-controller-manager



Copyright © 2017 Hao Zhang All rights reserved.

Kube-controller-manager

- 28 Controllers as of 1.8, usually deployed as a Pod
- All autonomous micro-services: easily turn on/off
- HA via leader election
- Shared Kubernetes client
  - Limit QPS, manage session/HTTPConnection, auth, retry with jittered backoff
- InformerFactory (SharedInformer)
  - Reduce API Call, ensure cache consistency, lighter GET/LIST
- Reflector / Store / Informer / Controller framework
  - Easy, separated development of different controllers as micro services
- Worker Pool
  - Individually configurable agility and resource consumption

Copyright © 2017 Hao Zhang All rights reserved.

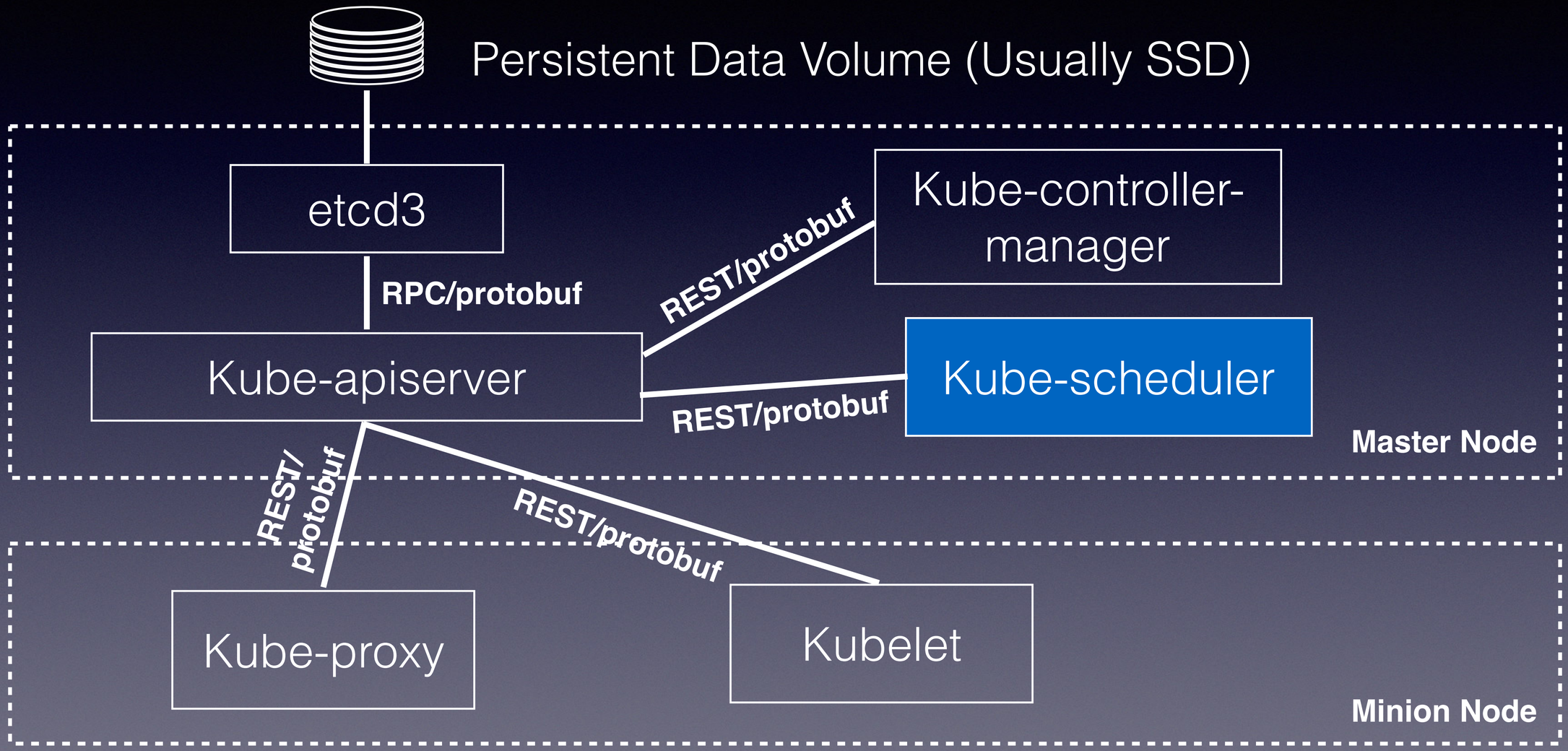
Kube-controller-manager

More Readings:

- Controller Dev Guide: https://github.com/kubernetes/community/blob/master/contributors/devel/controllers.md
- Controller Manager Main: https://github.com/kubernetes/kubernetes/blob/master/cmd/kube-controller-manager/app/controllermanager.go
- Shared Informer: https://github.com/kubernetes/kubernetes/blob/master/staging/src/k8s.io/client-go/tools/cache/shared_informer.go
- Controller Config: https://github.com/kubernetes/kubernetes/blob/master/staging/src/k8s.io/client-go/tools/cache/controller.go
- Reflector: https://github.com/kubernetes/kubernetes/blob/master/staging/src/k8s.io/client-go/tools/cache/reflector.go
- List and Watch: https://github.com/kubernetes/kubernetes/blob/master/staging/src/k8s.io/client-go/tools/cache/listwatch.go
- All Controller Logics: https://github.com/kubernetes/kubernetes/tree/master/pkg/controller

Copyright © 2017 Hao Zhang All rights reserved.

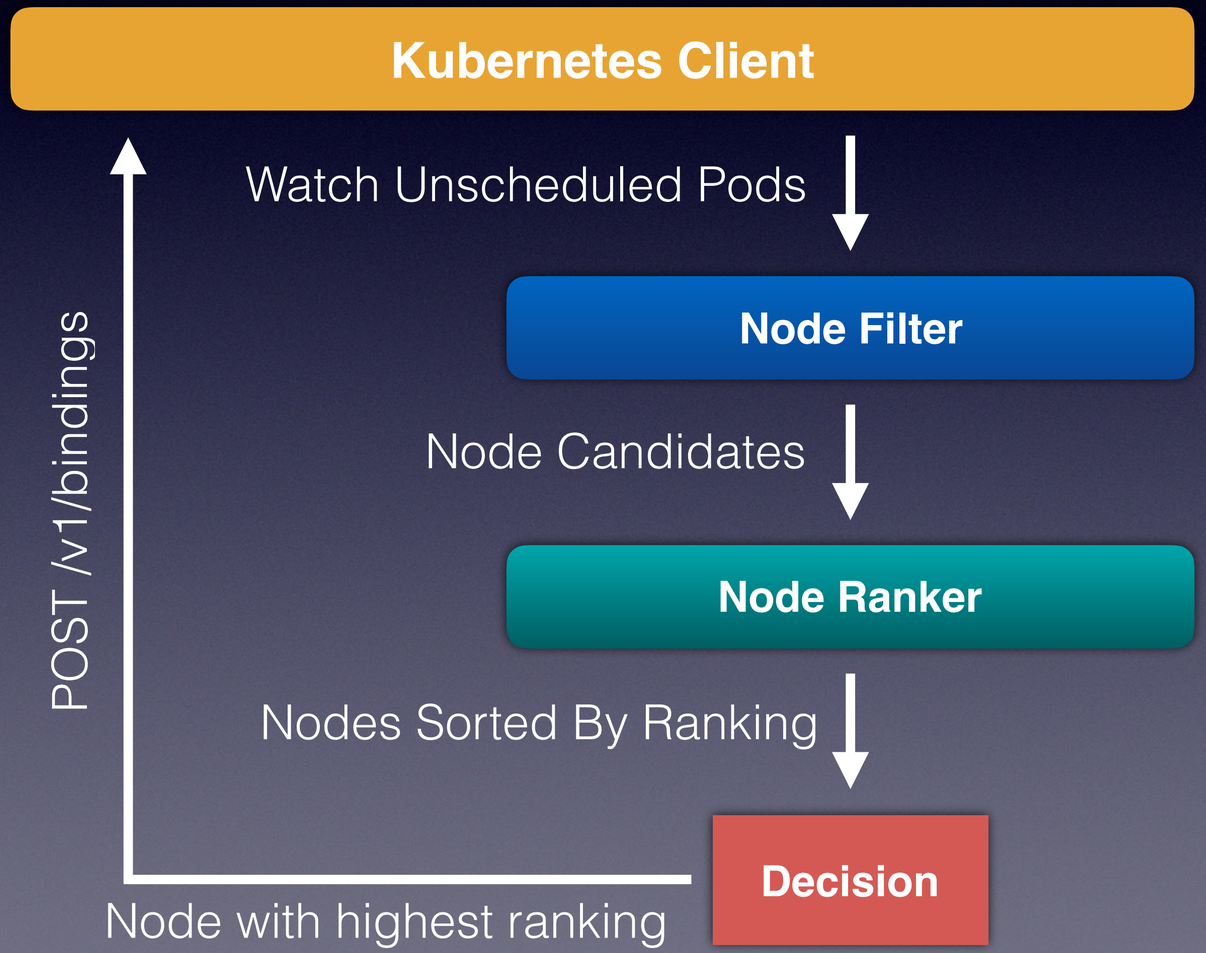
Kube-scheduler



Copyright © 2017 Hao Zhang All rights reserved.

Kube-scheduler

- Scheduling Algorithm



- Filter Out Impossible Node
  - **NodeIsReady**: Node should be in Ready state
  - **CheckNodeMemoryPressure**: Node can be over-provisioned, filter out node with memory pressure
  - **CheckNodeDiskPressure**: Node reporting host disk pressure should not be considered
  - **MaxNodeEBSVolumeCount**: If a Pod needs a volume, filter out nodes reach max volume count
  - **MatchNodeSelector**: If Pod has a nodeSelector, it should only consider nodes with that label
  - **PodFitsResources**: Node should have resource quota
  - **PodFitsHostPort**: Pod's HostPort should be available
  - **NoDiskConflict**: We should have volume this Pod wants
  - **NoVolumeZoneConflict**: Filter out nodes with zone conflict
  - **.....**

- Give Every Node a Score
  - **LeastRequestedPriority**: Nodes "more idle" is preferred
  - **BalancedResourceAllocation**: Balance node utilization
  - **SelectorSpreadPriority**: Replicas should be spread out
  - **Affinity/AntiAffinityPriority**: "I prefer to stay away / co-locate from some other categories of Pods"
  - **ImageLocalityPriority**: Node is preferred if it has the image
  - **.....**

Copyright © 2017 Hao Zhang All rights reserved.

Kube-scheduler

- Default scheduler
  - Sequential scheduler
  - Some application / topology awareness
    - User can define affinity, anti-affinity, node-selector, etc
  - Schedule 30,000 Pods onto 1,000 Nodes within 587 Seconds (In v1.2)
  - HA via leader election
- Supports multiple schedulers
  - Pod can choose the scheduler that schedules it
  - Schedulers share same pool of nodes, race condition is possible
  - Kubelet can reject Pods in case of conflict, and triggers reschedule
- Easy to write user's own scheduler or just extend algorithm provider for default scheduler

Copyright © 2017 Hao Zhang All rights reserved.

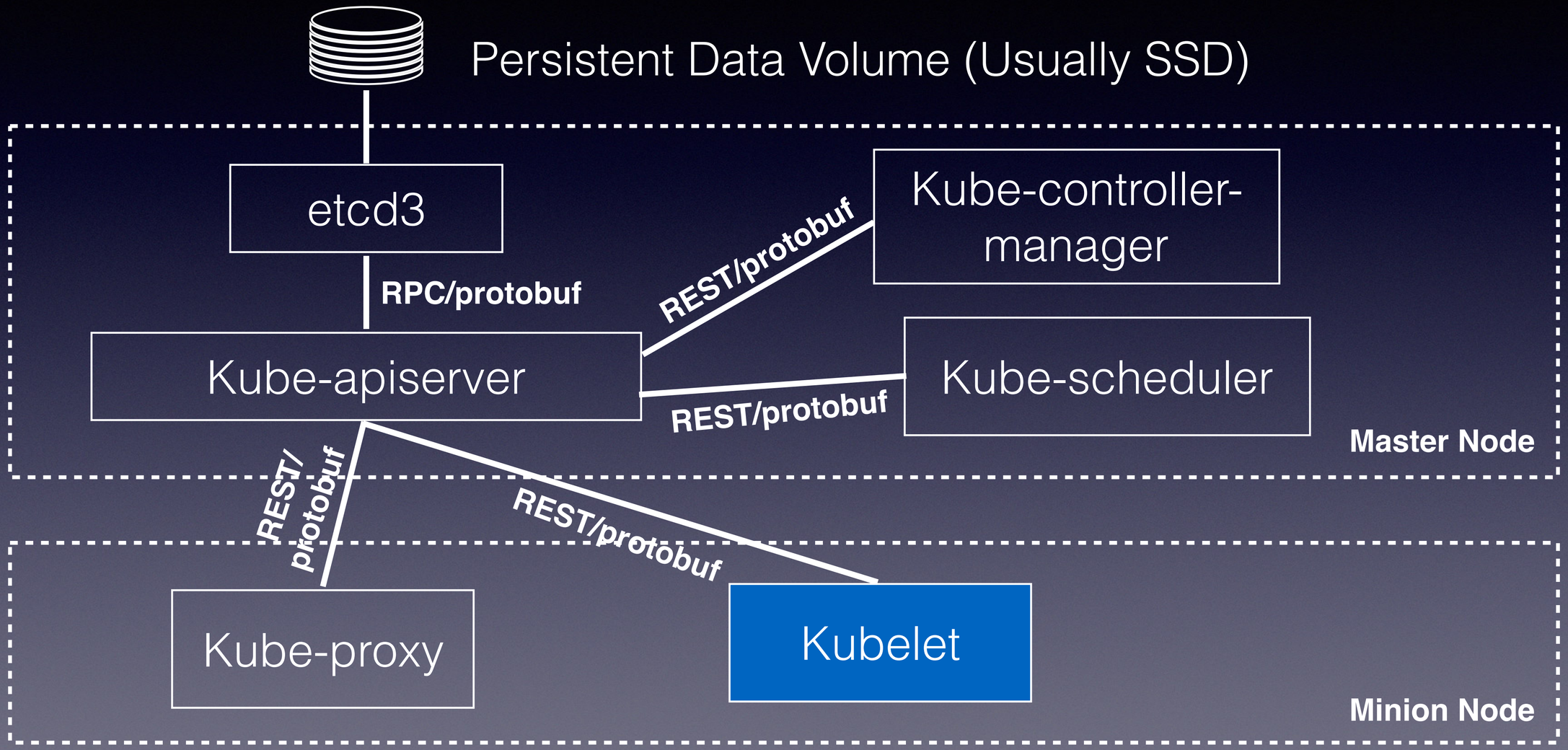
Kube-scheduler

More Readings:

- Documentations:
  - Scheduler Overview: https://github.com/kubernetes/community/blob/master/contributors/devel/scheduler.md
  - Guaranteed scheduling through re-scheduling: https://kubernetes.io/docs/tasks/administer-cluster/guaranteed-scheduling-critical-addon-pods/
  - Default Scheduling Algorithm: https://github.com/kubernetes/community/blob/master/contributors/devel/scheduler_algorithm.md
- Blogs
  - Scheduler Optimization by CoreOS: https://coreos.com/blog/improving-kubernetes-scheduler-performance.html
  - Advanced Scheduling and Customized Scheduler After 1.6: http://blog.kubernetes.io/2017/03/advanced-scheduling-in-kubernetes.html
- Critical Code Snippets
  - Scheduling Predicates: https://github.com/kubernetes/kubernetes/blob/master/plugin/pkg/scheduler/algorithm/predicates/predicates.go
  - Scheduling Defaults: https://github.com/kubernetes/kubernetes/blob/master/plugin/pkg/scheduler/algorithmprovider/defaults/defaults.go
- Designs, Specs, Discussions
  - Scheduling Feature Proposals: https://github.com/kubernetes/community/tree/master/contributors/design-proposals/scheduling

Copyright © 2017 Hao Zhang All rights reserved.

Kubelet



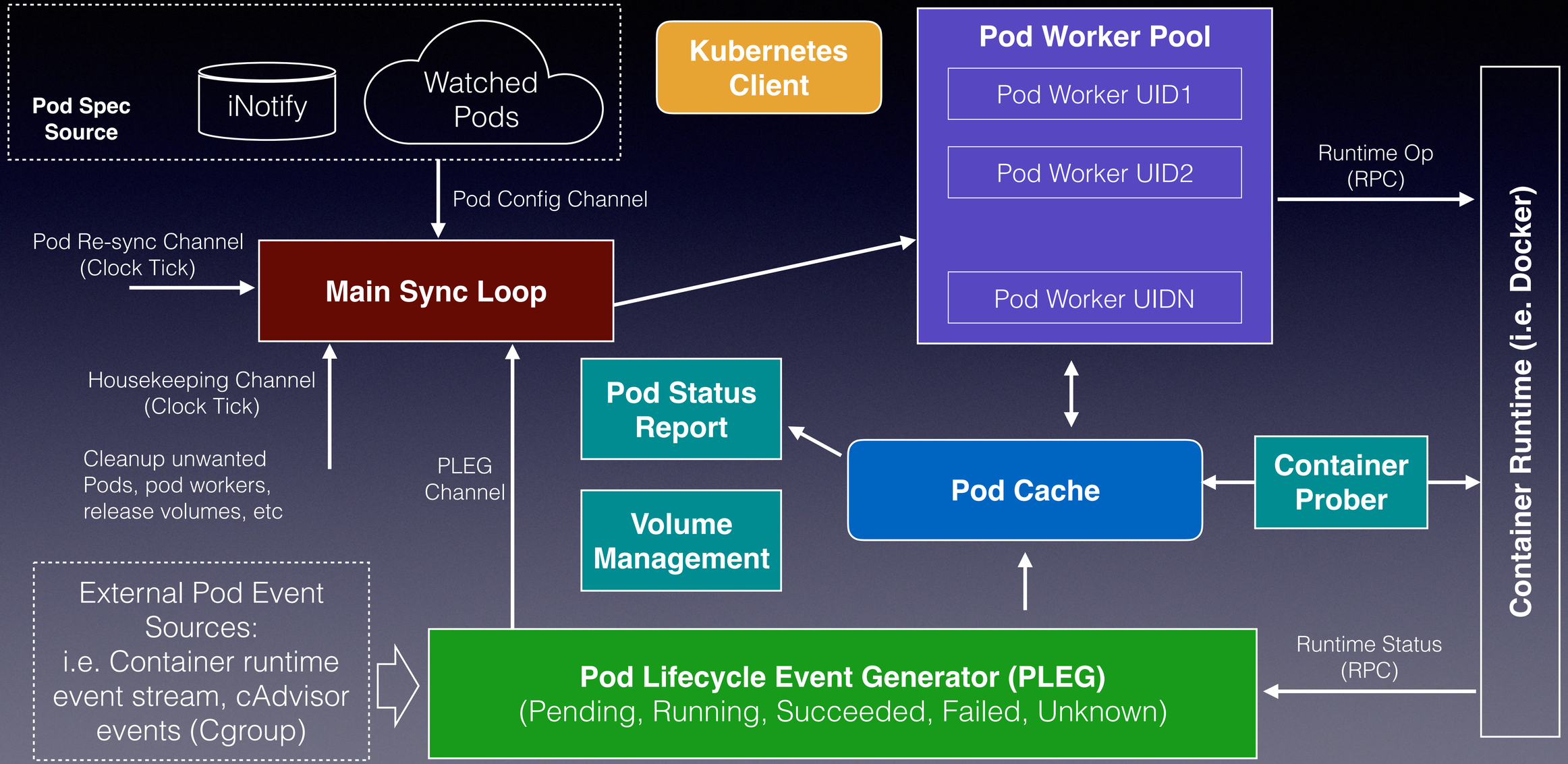
Copyright © 2017 Hao Zhang All rights reserved.

Kubelet

- Kubernetes' node worker
- Ensure Pods run as described by spec
  - Report readiness and liveness
- Node-level admission control
  - Reserve resource, limite # of Pods, etc.
- Report node status to API server (v1Node)
  - Machine info, resource usage, health, images
- Host stats (Cgroup metrics via cAdvisor)
- Communicate with container through runtime interface
  - Stream logs, execute commands, port-forward, etc
- Usually not run as a container, but a service under Systemd

Copyright © 2017 Hao Zhang All rights reserved.

Kubelet



Copyright © 2017 Hao Zhang All rights reserved.

Kubelet

- Main Controller Loop
  - Listen on events from channels and react
- PLEG (Pod Lifecycle Event Generation)
  - Listen and List on external sources, i.e. container runtime, cAdvisor
  - Translate container status changes to Pod Events
- Pod Worker
  - Interact with container runtime
  - Move Pod from current state (v1Pod.status) to desired state (v1Pod.spec)

Copyright © 2017 Hao Zhang All rights reserved.

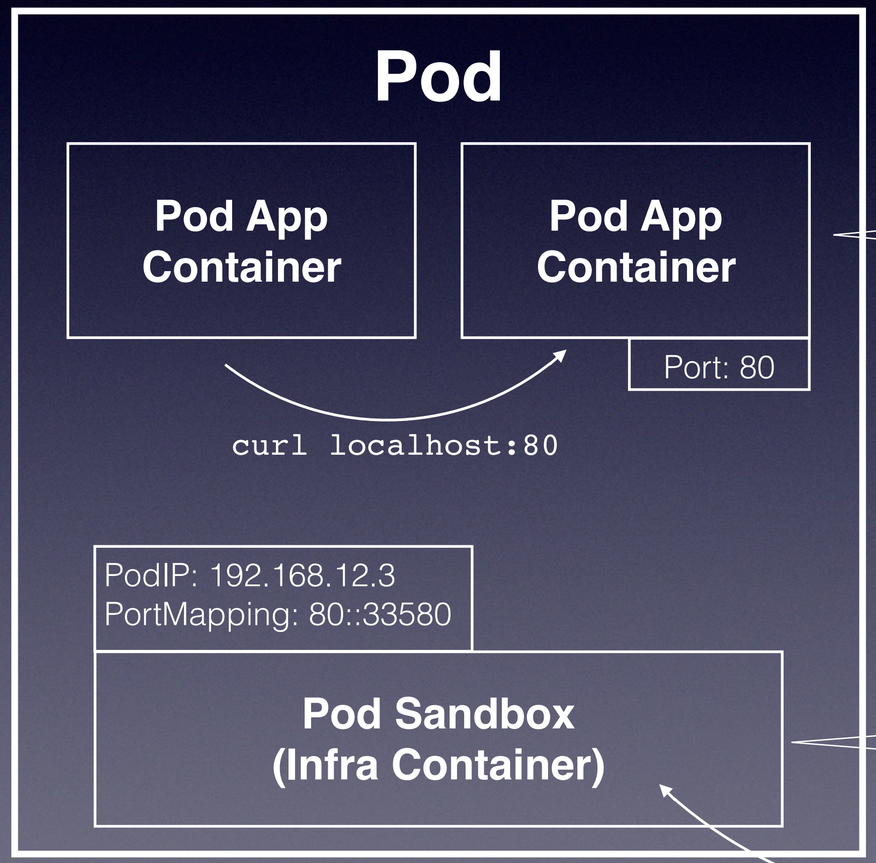
Kubelet

- Container Prober
  - Probe container liveness and healthiness using methods defined by spec
  - Publish event (v1Event) and put container in CrashLoopBackoff upon probing failures
- Volume Manager
  - Local directory management, device mount/unmount, device initialization, etc
- cAdvisor
  - Container matrix collection and reporting, events about kernel Cgroups

Copyright © 2017 Hao Zhang All rights reserved.

Kubelet

- How Pod is set up



- **App Containers**
- Use Sandbox as Cgroup parent
  - Share same “localhost”
  - `dockerService.CreateContainer()`
  - `kubeGenericRuntimeManager. startContainer()`

- **Sandbox**: An isolated environment with resource constraints
- Impl. Based on runtime
  - Could be VM, Linux Cgroup, etc
- Kubelet sets the following for docker impl.
  - Cgroup, Port mapping, Linux security context
  - Holds resolv.conf that is shared by all containers in Pod
  - `dockerService.makeSandboxDockerConfig()`

Copyright © 2017 Hao Zhang All rights reserved.

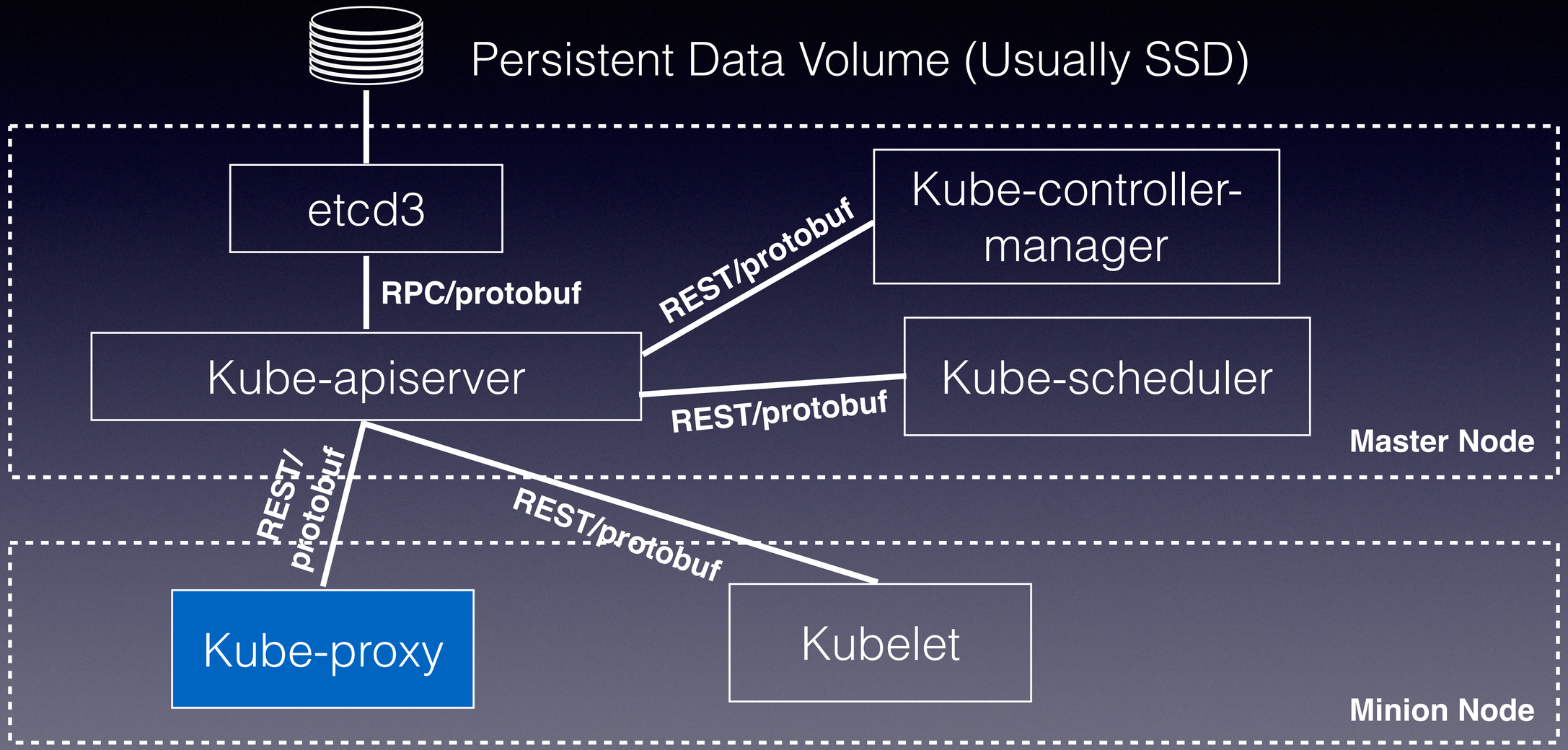
Kubelet

More Readings:

- Kubelet Main: https://github.com/kubernetes/kubernetes/blob/master/pkg/kubelet/kubelet.go
  - Kubelet main function: `Kubelet.Run()`
  - Kubelet sync Loop channel event processing: `Kubelet.syncLoopIteration()`
  - Kubelet sync Pod: `Kubelet.syncPod()`
- Pod Worker: https://github.com/kubernetes/kubernetes/blob/master/pkg/kubelet/pod_workers.go
- Container Runtime Sync Pod: https://github.com/kubernetes/kubernetes/blob/master/pkg/kubelet/kuberuntime/kuberuntime_manager.go
  - Sync Pod using CRI: `kubeGenericRuntimeManager.SyncPod()`
- Container Runtime Start Container: https://github.com/kubernetes/kubernetes/blob/master/pkg/kubelet/kuberuntime/kuberuntime_container.go
  - How kubelet setup container: `kubeGenericRuntimeManager.startContainer()`
- Implementation of PodSandbox using Docker: https://github.com/kubernetes/kubernetes/blob/master/pkg/kubelet/dockershim/docker_sandbox.go
  - Create Pod sandbox: `dockerService.RunPodSandbox()`
- PLEG Design Proposal: https://github.com/kubernetes/community/blob/master/contributors/design-proposals/node/pod-lifecycle-event-generator.md
- Pod Cgroup Hierarchy: https://github.com/kubernetes/community/blob/master/contributors/design-proposals/node/pod-resource-management.md#cgroup-hierachy
- All Kubelet related design docs: https://github.com/kubernetes/community/tree/master/contributors/design-proposals/node

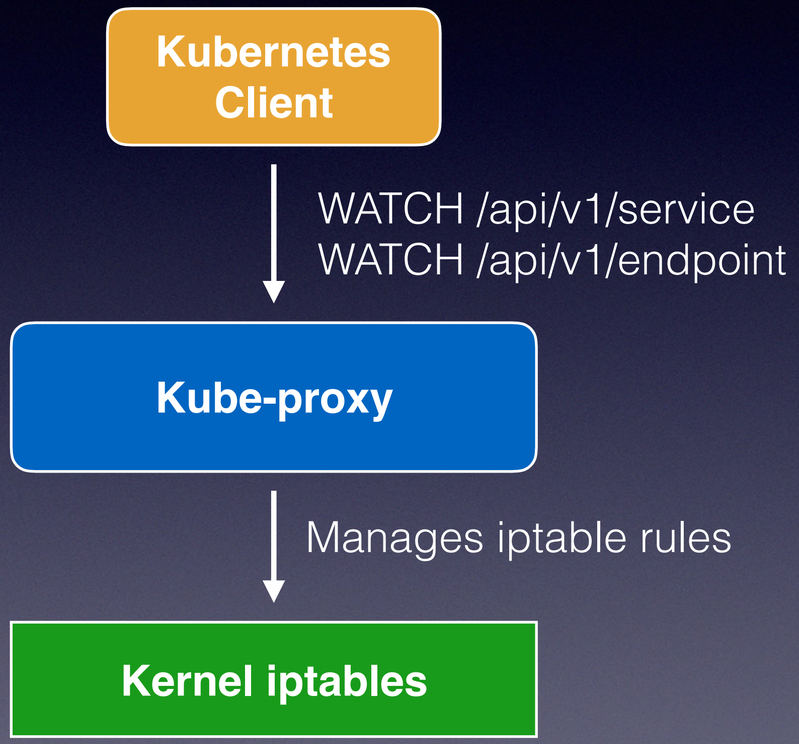
Copyright © 2017 Hao Zhang All rights reserved.

Kube-proxy



Copyright © 2017 Hao Zhang All rights reserved.

Kube-proxy



- Daemon on every node
- Watch on changes of Service and Endpoint
- Manipulate iptable rules on host to achieve service load balancing (one of the implementations)

More Readings:

- v1Service object and different types of proxy implementations: https://kubernetes.io/docs/concepts/services-networking/service/#virtual-ips-and-service-proxies
- iptable proxy implementation: https://github.com/kubernetes/kubernetes/blob/master/pkg/proxy/iptables/proxier.go
  - `Proxier.syncProxyRules()`
- Other built-in proxy implementations: https://github.com/kubernetes/kubernetes/tree/master/pkg/proxy
- Some early discussions about Service object and implementations: https://github.com/kubernetes/kubernetes/issues/1107

Copyright © 2017 Hao Zhang All rights reserved.

Put It Together

What will happen after `kubectl create`

Put it together

- What will happen when you do the following?

```
$ kubectl create -f myapp.yaml
Deployment nginx-deployment created
$
```

```
$ cat myapp.yaml
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 9376
```

Copyright © 2017 Hao Zhang All rights reserved.

Kubernetes Reaction Chain

Note: this is a rather high-level idea about Kubernetes reaction chain. It hides some details for illustration purpose and is different than actual behavior

Kubectl

Deployment Controller
WATCH v1Deployment, v1ReplicaSet

ReplicaSet Controller
WATCH v1ReplicaSet, v1Pod

Scheduler
WATCH v1Pod, v1Node

Kubelet-node1
WATCH v1Pod on node1

CRI

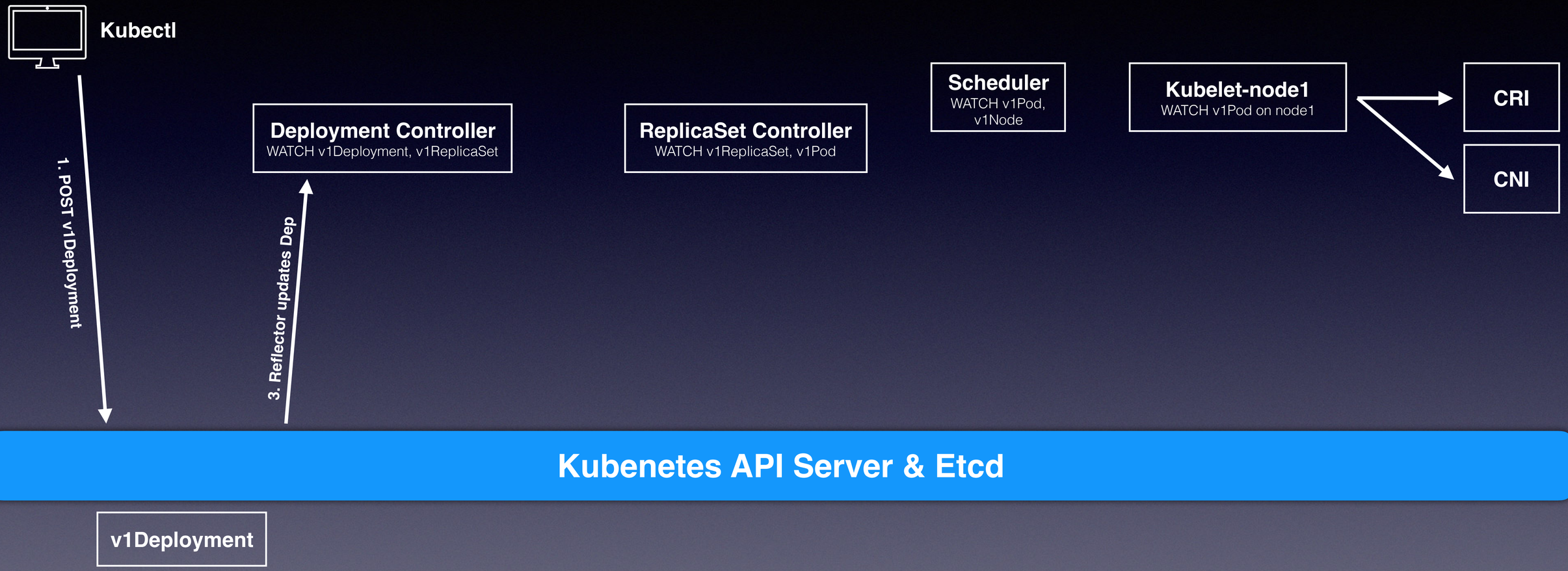
CNI

Kubenetes API Server & Etcd

Copyright © 2017 Hao Zhang All rights reserved.

Kubernetes Reaction Chain

Note: this is a rather high-level idea about Kubernetes reaction chain. It hides some details for illustration purpose and is different than actual behavior

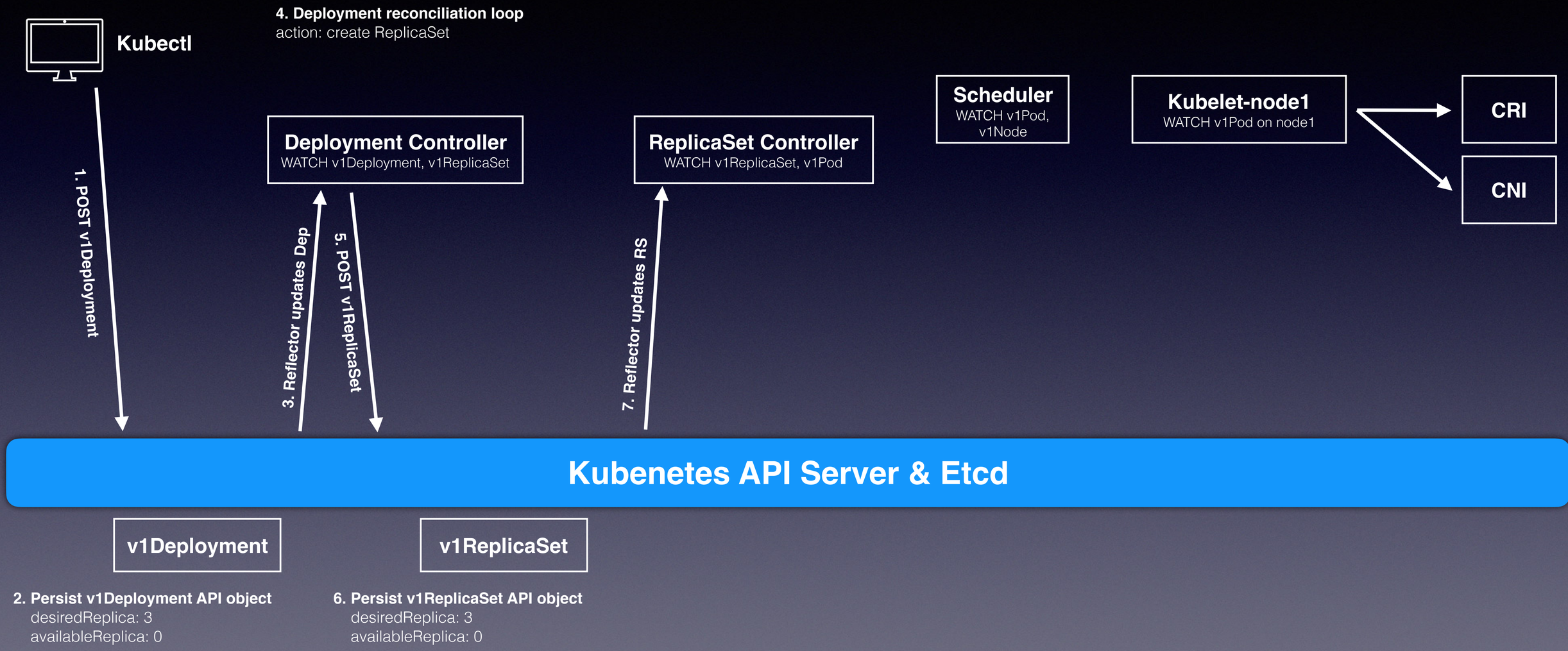


2. Persist v1Deployment API object
desiredReplica: 3
availableReplica: 0

Copyright © 2017 Hao Zhang All rights reserved.

Kubernetes Reaction Chain

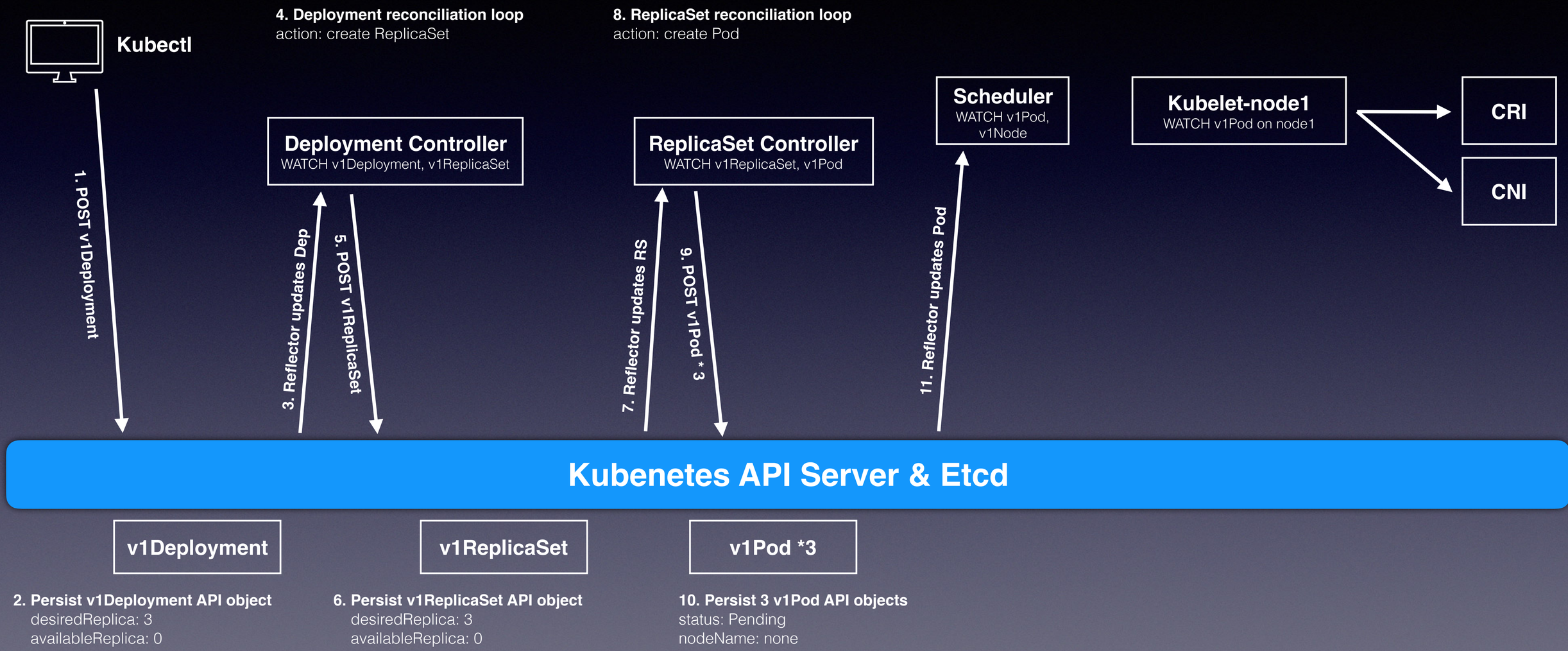
Note: this is a rather high-level idea about Kubernetes reaction chain. It hides some details for illustration purpose and is different than actual behavior



Copyright © 2017 Hao Zhang All rights reserved.

Kubernetes Reaction Chain

Note: this is a rather high-level idea about Kubernetes reaction chain. It hides some details for illustration purpose and is different than actual behavior



2. Persist v1Deployment API object
desiredReplica: 3
availableReplica: 0

4. Deployment reconciliation loop
action: create ReplicaSet

6. Persist v1ReplicaSet API object
desiredReplica: 3
availableReplica: 0

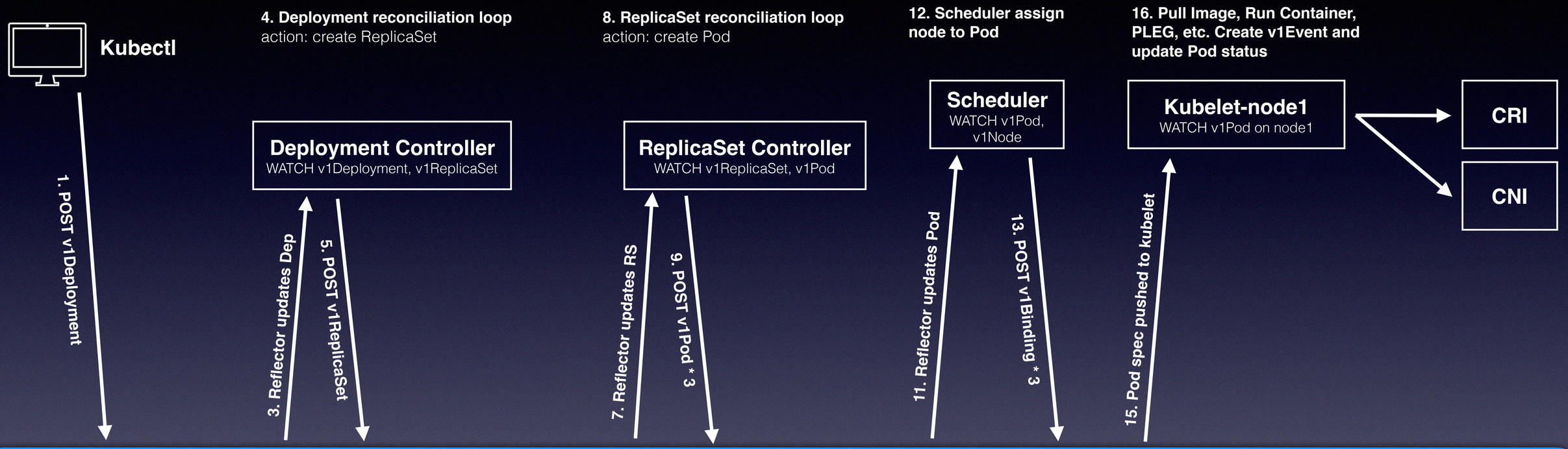
8. ReplicaSet reconciliation loop
action: create Pod

10. Persist 3 v1Pod API objects
status: Pending
nodeName: none

Copyright © 2017 Hao Zhang All rights reserved.

Kubernetes Reaction Chain

Note: this is a rather high-level idea about Kubernetes reaction chain. It hides some details for illustration purpose and is different than actual behavior



4. Deployment reconciliation loop
action: create ReplicaSet

8. ReplicaSet reconciliation loop
action: create Pod

12. Scheduler assign node to Pod

16. Pull Image, Run Container, PLEG, etc. Create v1Event and update Pod status

Kubenetes API Server & Etcd

v1Deployment

2. Persist v1Deployment API object
desiredReplica: 3
availableReplica: 0

v1ReplicaSet

6. Persist v1ReplicaSet API object
desiredReplica: 3
availableReplica: 0

v1Pod *3

10. Persist 3 v1Pod API objects
status: Pending
nodeName: none

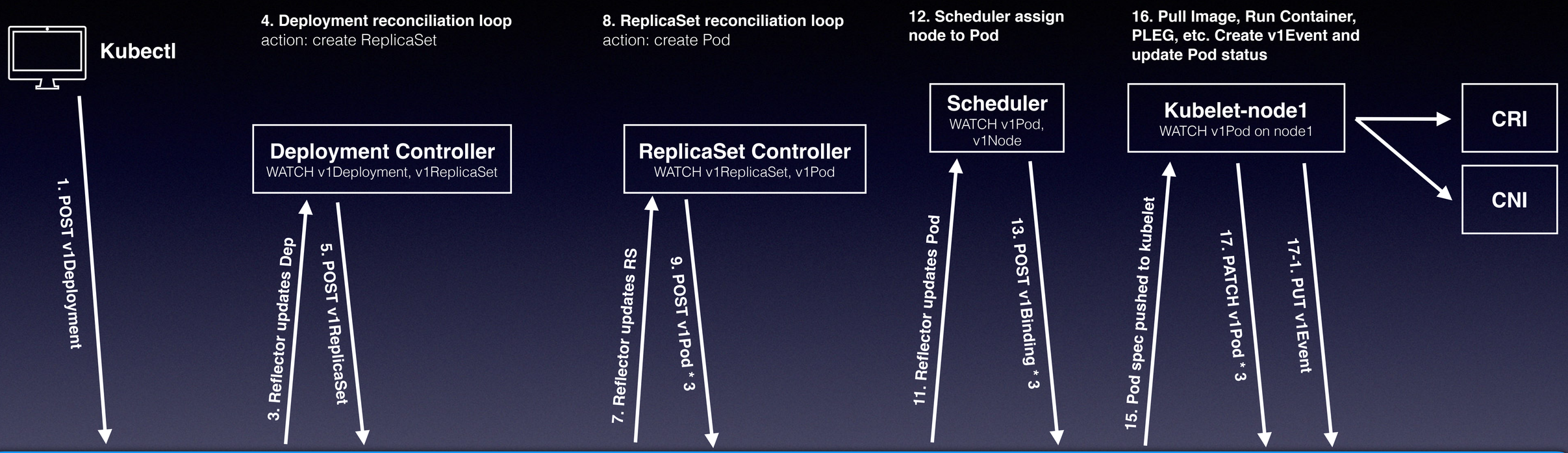
v1Binding * 3

14. Persist 3 v1Binding API objects
name: PodName
targetNodeName: node1

Copyright © 2017 Hao Zhang All rights reserved.

Kubernetes Reaction Chain

Note: this is a rather high-level idea about Kubernetes reaction chain. It hides some details for illustration purpose and is different than actual behavior



4. Deployment reconciliation loop
action: create ReplicaSet

8. ReplicaSet reconciliation loop
action: create Pod

12. Scheduler assign node to Pod

16. Pull Image, Run Container, PLEG, etc. Create v1Event and update Pod status

Kubenetes API Server & Etcd

v1Deployment

2. Persist v1Deployment API object
desiredReplica: 3
availableReplica: 0

v1ReplicaSet

6. Persist v1ReplicaSet API object
desiredReplica: 3
availableReplica: 0

v1Pod *3

10. Persist 3 v1Pod API objects
status: Pending
nodeName: none

17. Update 3 v1Pod API objects
status: PodInitializing -> Running
nodeName: node1

v1Binding * 3

14. Persist 3 v1Binding API objects
name: PodName
targetNodeName: node1

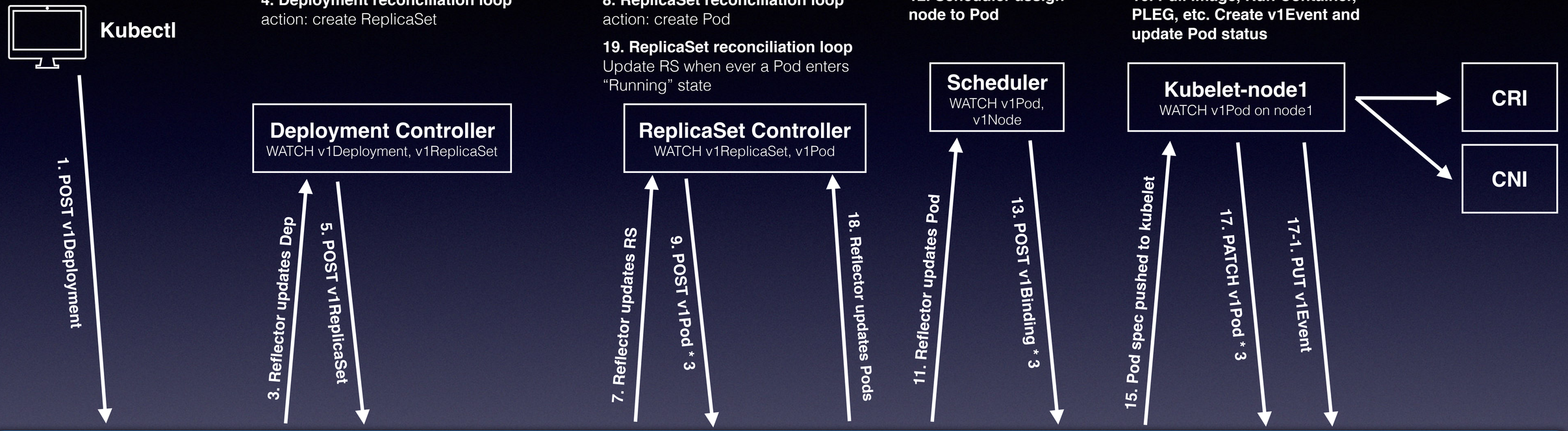
v1Event * N

17-1. Several v1Event objects created
e.g. ImagePulled, CreatingContainer, StartedContainer, etc., controllers, scheduler can also create events throughout the entire reaction chain

Copyright © 2017 Hao Zhang All rights reserved.

Kubernetes Reaction Chain

Note: this is a rather high-level idea about Kubernetes reaction chain. It hides some details for illustration purpose and is different than actual behavior



4. Deployment reconciliation loop
action: create ReplicaSet

8. ReplicaSet reconciliation loop
action: create Pod

19. ReplicaSet reconciliation loop
Update RS when ever a Pod enters "Running" state

12. Scheduler assign node to Pod

16. Pull Image, Run Container, PLEG, etc. Create v1Event and update Pod status

Kubenetes API Server & Etcd

v1Deployment

2. Persist v1Deployment API object
desiredReplica: 3
availableReplica: 0

v1ReplicaSet

6. Persist v1ReplicaSet API object
desiredReplica: 3
availableReplica: 0

v1Pod *3

10. Persist 3 v1Pod API objects
status: Pending
nodeName: none

17. Update 3 v1Pod API objects
status: PodInitializing -> Running
nodeName: node1

v1Binding * 3

14. Persist 3 v1Binding API objects
name: PodName
targetNodeName: node1

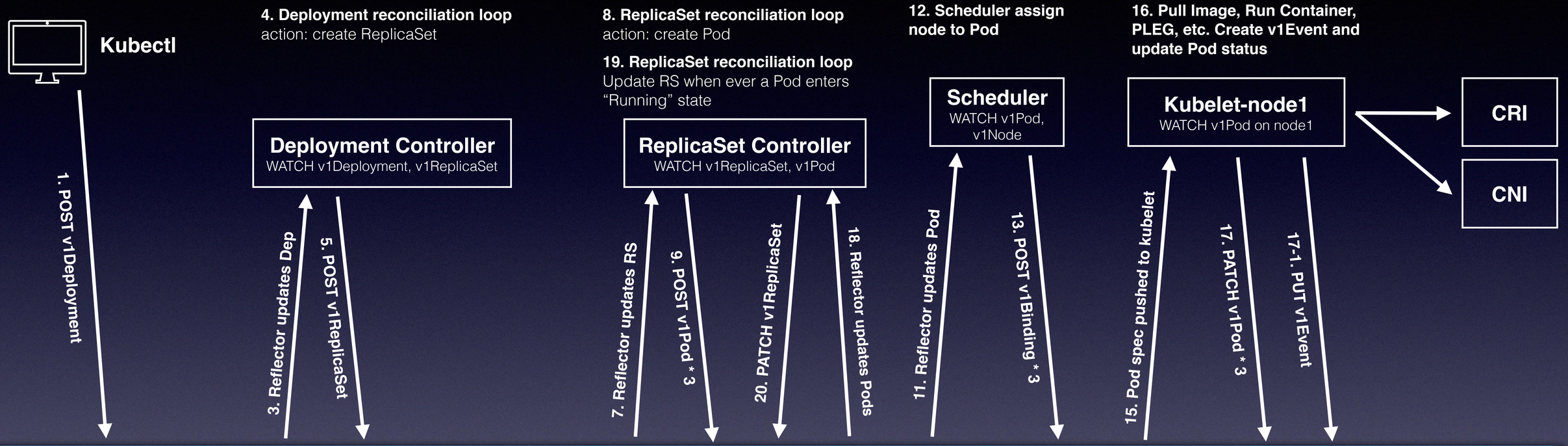
v1Event * N

17-1. Several v1Event objects created
e.g. ImagePulled, CreatingContainer, StartedContainer, etc., controllers, scheduler can also create events throughout the entire reaction chain

Copyright © 2017 Hao Zhang All rights reserved.

Kubernetes Reaction Chain

Note: this is a rather high-level idea about Kubernetes reaction chain. It hides some details for illustration purpose and is different than actual behavior



Kubenetes API Server & Etcd

v1Deployment

2. Persist v1Deployment API object
desiredReplica: 3
availableReplica: 0

v1ReplicaSet

6. Persist v1ReplicaSet API object
desiredReplica: 3
availableReplica: 0

21. Update v1ReplicaSet API object
desiredReplica: 3
availableReplica: 0 -> 3

v1Pod *3

10. Persist 3 v1Pod API objects
status: Pending
nodeName: none

17. Update 3 v1Pod API objects
status: PodInitializing -> Running
nodeName: node1

v1Binding * 3

14. Persist 3 v1Binding API objects
name: PodName
targetNodeName: node1

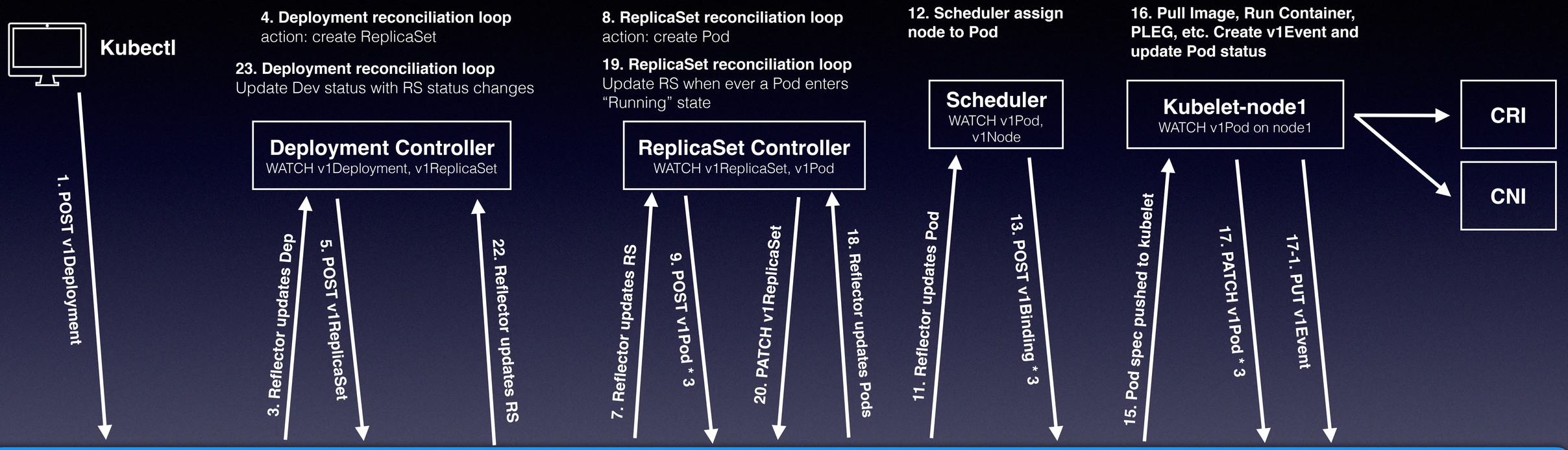
v1Event * N

17-1. Several v1Event objects created
e.g. ImagePulled, CreatingContainer, StartedContainer, etc., controllers, scheduler can also create events throughout the entire reaction chain

Copyright © 2017 Hao Zhang All rights reserved.

Kubernetes Reaction Chain

Note: this is a rather high-level idea about Kubernetes reaction chain. It hides some details for illustration purpose and is different than actual behavior



Kubenetes API Server & Etcd

v1Deployment

2. Persist v1Deployment API object
desiredReplica: 3
availableReplica: 0

v1ReplicaSet

6. Persist v1ReplicaSet API object
desiredReplica: 3
availableReplica: 0

21. Update v1ReplicaSet API object
desiredReplica: 3
availableReplica: 0 -> 3

v1Pod *3

10. Persist 3 v1Pod API objects
status: Pending
nodeName: none

17. Update 3 v1Pod API objects
status: PodInitializing -> Running
nodeName: node1

v1Binding * 3

14. Persist 3 v1Binding API objects
name: PodName
targetNodeName: node1

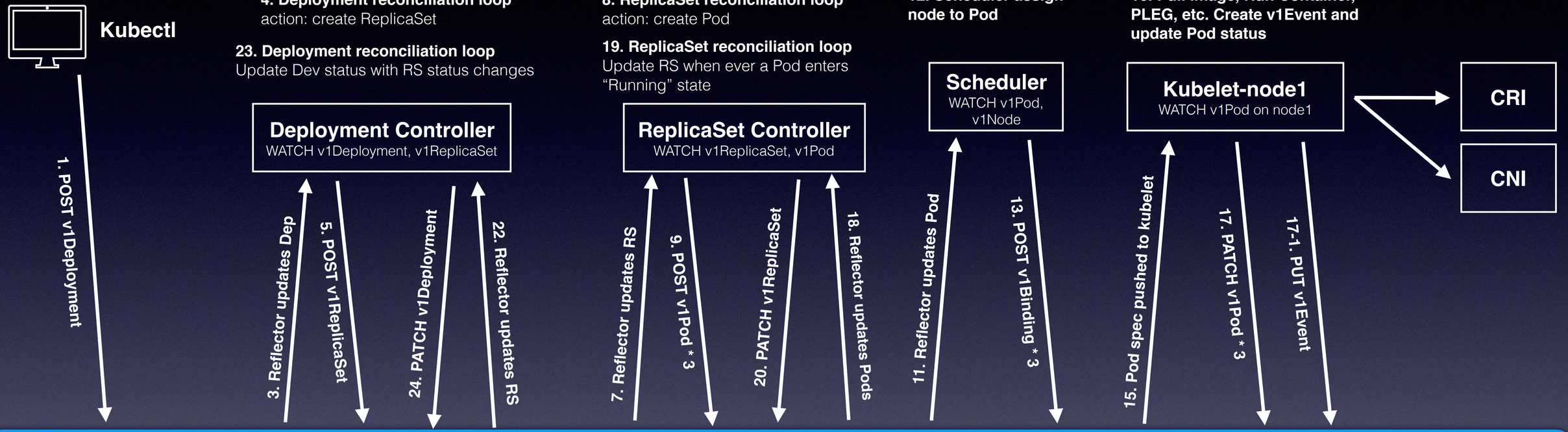
v1Event * N

17-1. Several v1Event objects created
e.g. ImagePulled, CreatingContainer, StartedContainer, etc., controllers, scheduler can also create events throughout the entire reaction chain

Copyright © 2017 Hao Zhang All rights reserved.

Kubernetes Reaction Chain

Note: this is a rather high-level idea about Kubernetes reaction chain. It hides some details for illustration purpose and is different than actual behavior



Kubenetes API Server & Etcd

v1Deployment

2. Persist v1Deployment API object
desiredReplica: 3
availableReplica: 0

25. Update v1Deployment API object
desiredReplica: 3
availableReplica: 0 -> 3

v1ReplicaSet

6. Persist v1ReplicaSet API object
desiredReplica: 3
availableReplica: 0

21. Update v1ReplicaSet API object
desiredReplica: 3
availableReplica: 0 -> 3

v1Pod *3

10. Persist 3 v1Pod API objects
status: Pending
nodeName: none

17. Update 3 v1Pod API objects
status: PodInitializing -> Running
nodeName: node1

v1Binding * 3

14. Persist 3 v1Binding API objects
name: PodName
targetNodeName: node1

v1Event * N

17-1. Several v1Event objects created
e.g. ImagePulled, CreatingContainer, StartedContainer, etc., controllers, scheduler can also create events throughout the entire reaction chain

Copyright © 2017 Hao Zhang All rights reserved.

To be continued in Part II