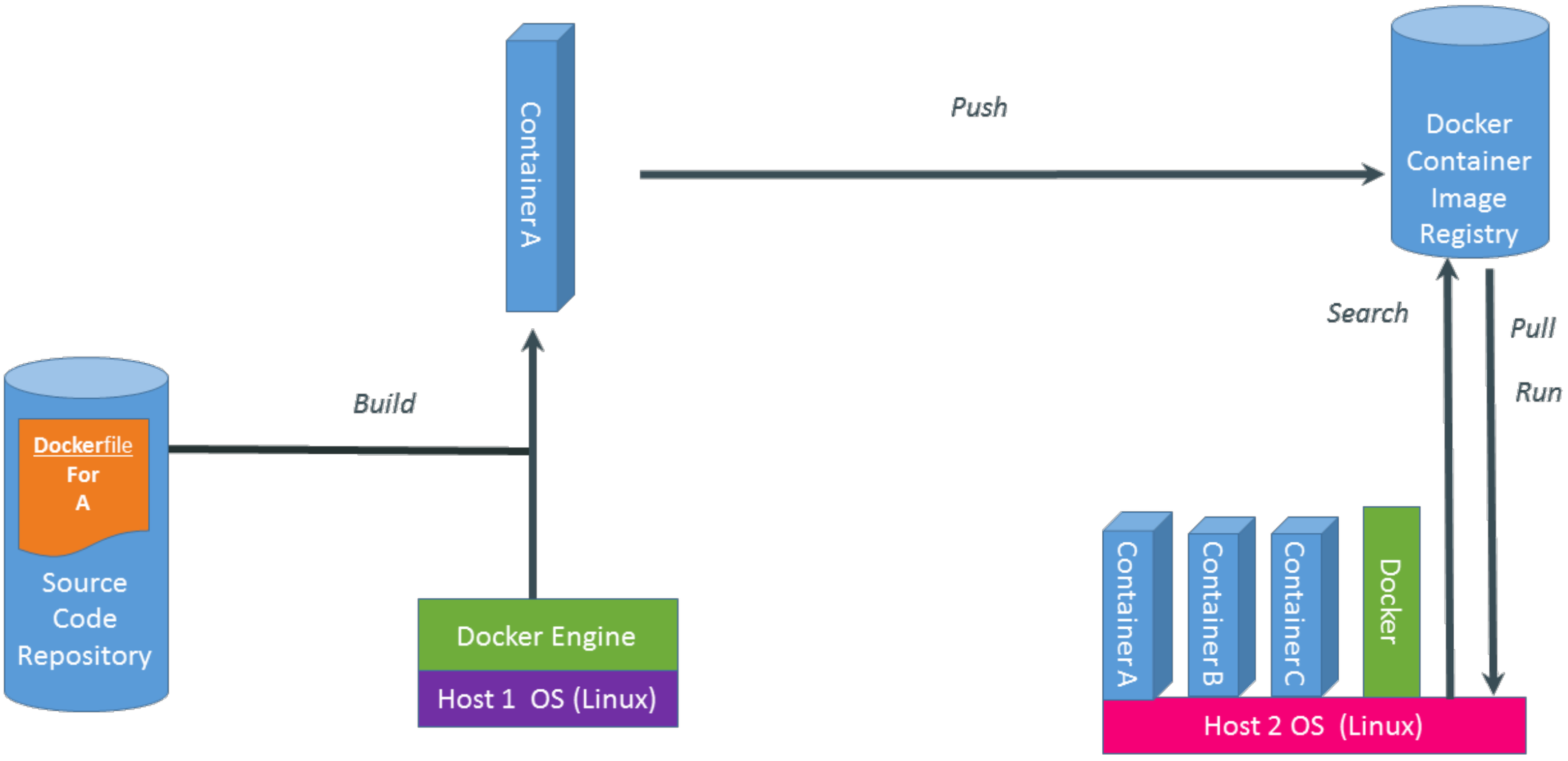
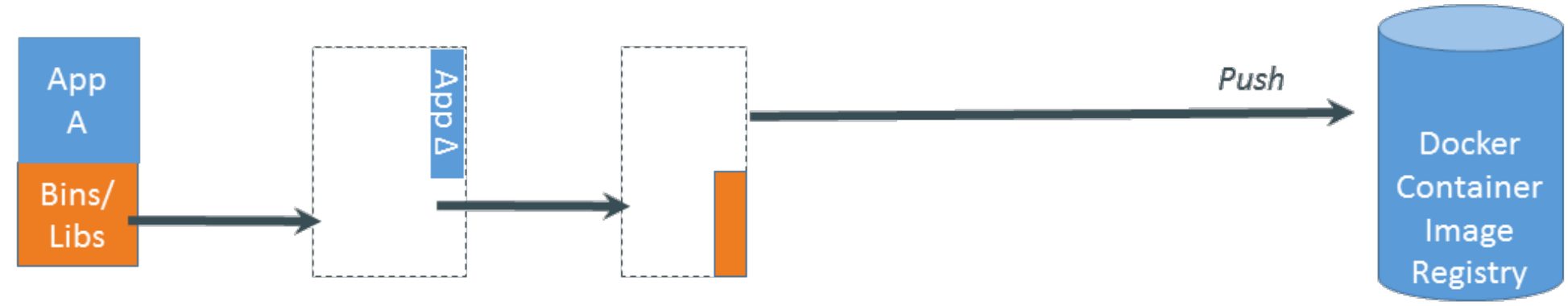
Docker Image Registry

- Registry containing docker images
  - Local registry on the same host
  - Docker Hub Registry: Globally shared
    - https://hub.docker.com/search?q=
  - Private registry on docker.com

What are the Basics of a Docker System?



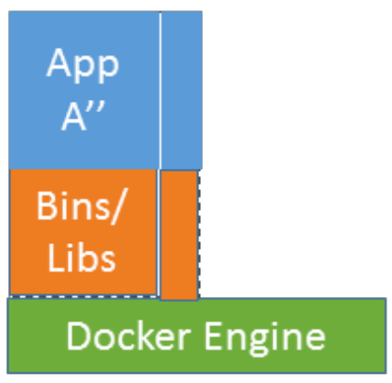
Changes and Updates



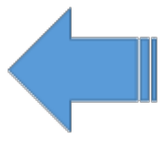
Base Container Image

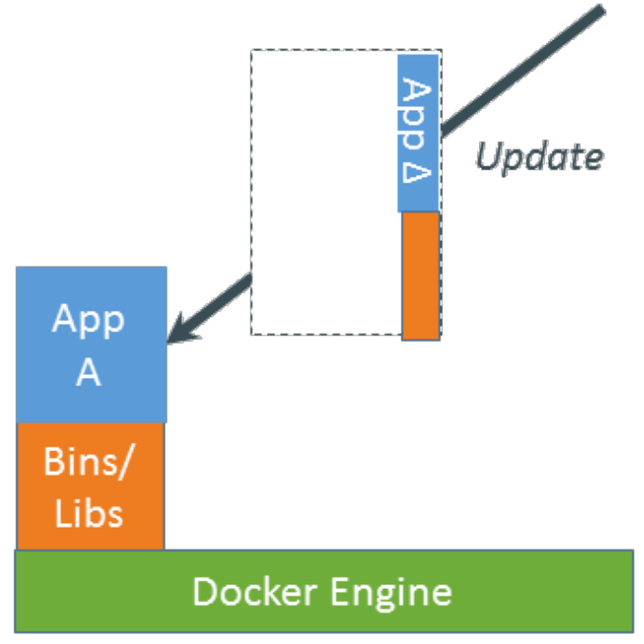
Container Mod A'

Container Mod A''

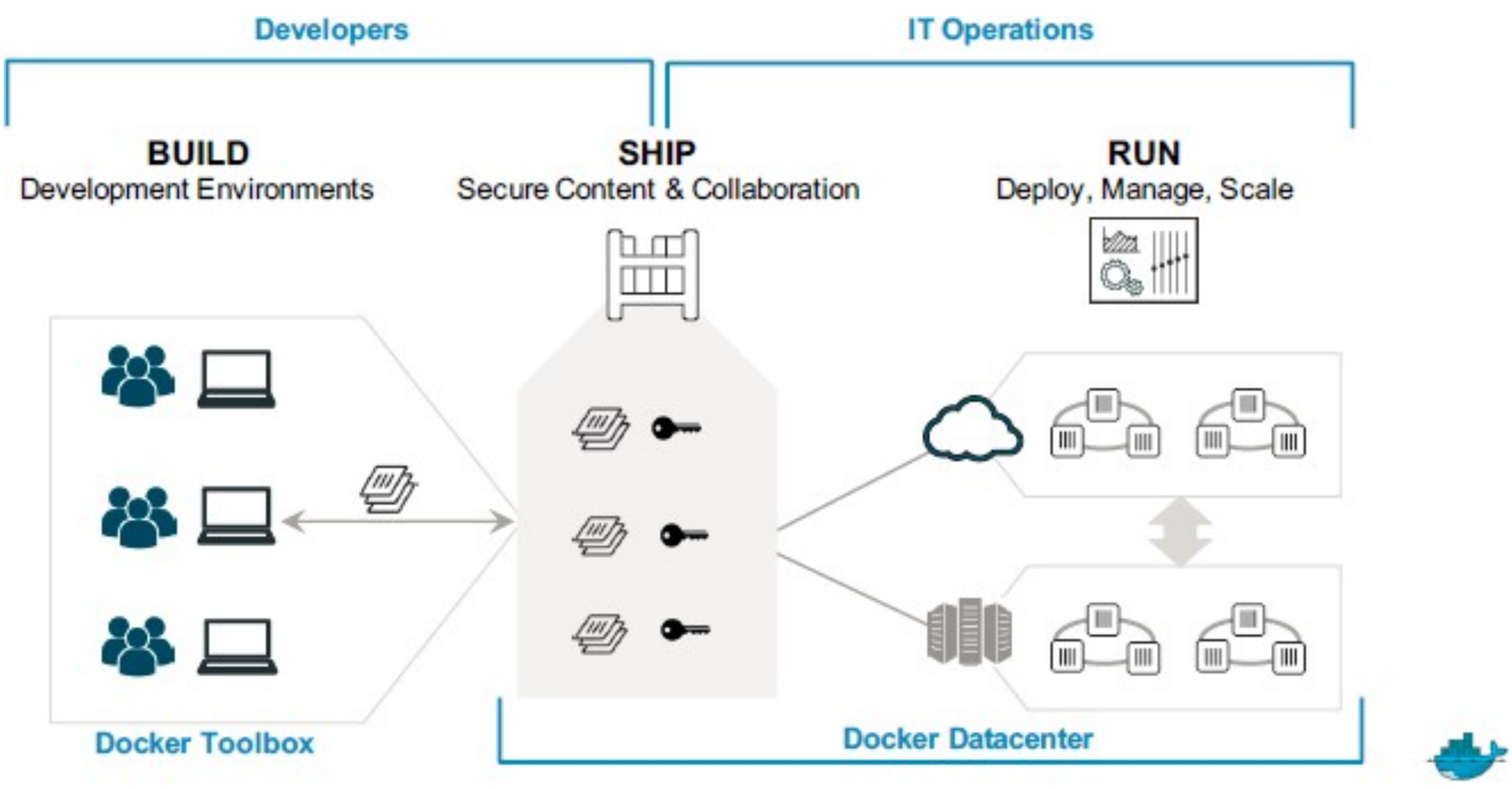


Host is now running A''





Host running A wants to upgrade to A''.
Requests update. Gets only diffs

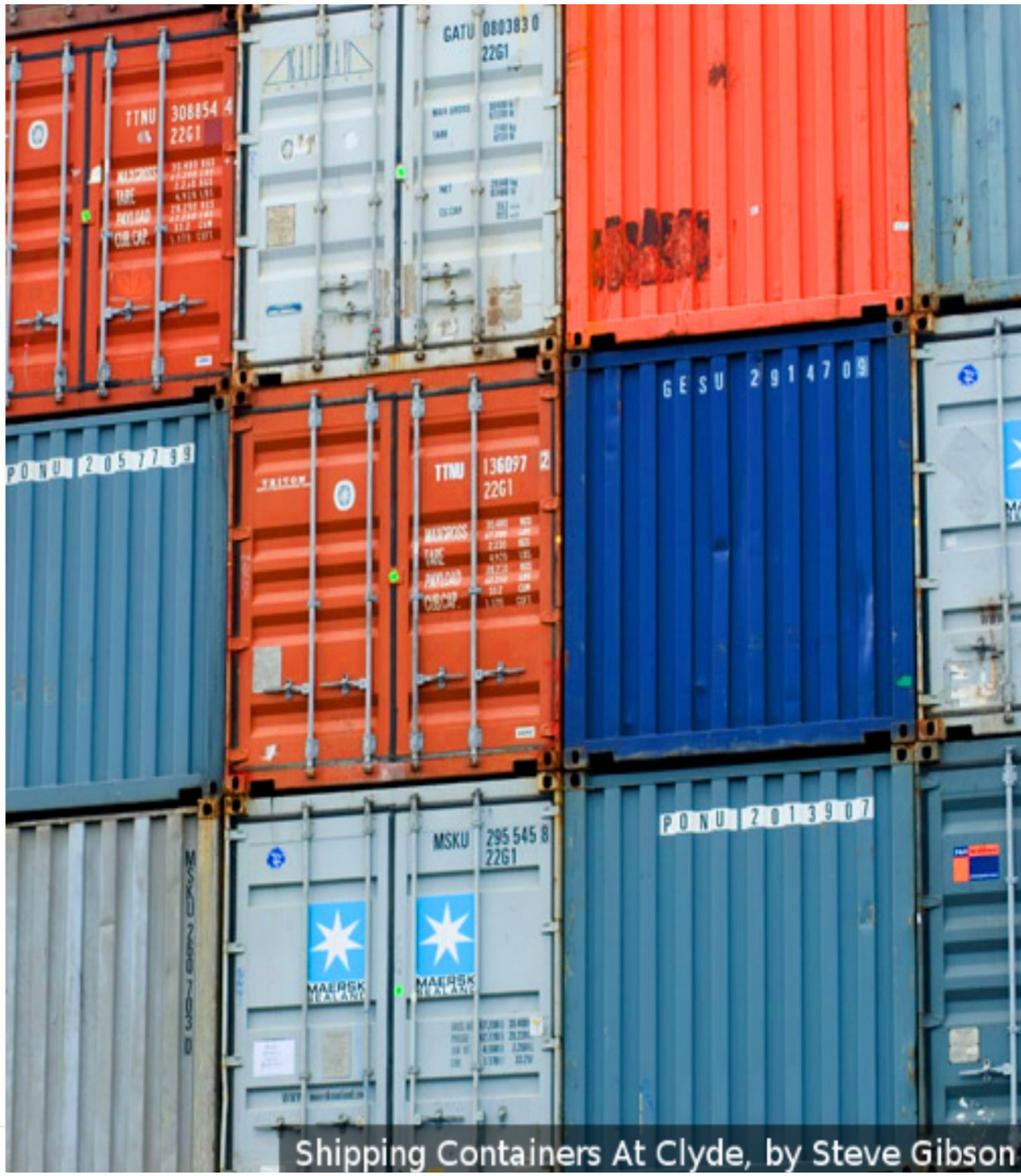

Developers
IT Operations
BUILD
Development Environments
SHIP
Secure Content & Collaboration
RUN
Deploy, Manage, Scale
Docker Toolbox
Docker Datacenter

Managing Container Clusters:

vs.

Everything at Google runs in containers:

- Gmail, Web Search, Maps, ...
- MapReduce, MillWheel, Pregel, ...
- Colossus, BigTable, Spanner, ...
- Even **Google's Cloud Computing product GCE itself**: VMs run in containers



Shipping Containers At Clyde, by Steve Gibson

Container orchestration

We need more than just packing and isolation

Scheduling: Where should my containers run?

Lifecycle and health: Keep my containers running despite failures

Discovery: Where are my containers now?

Monitoring: What's happening with my containers?

Auth{n,z}: Control who can do things to my containers

Aggregates: Compose sets of containers into jobs

Scaling: Making jobs bigger or smaller

...

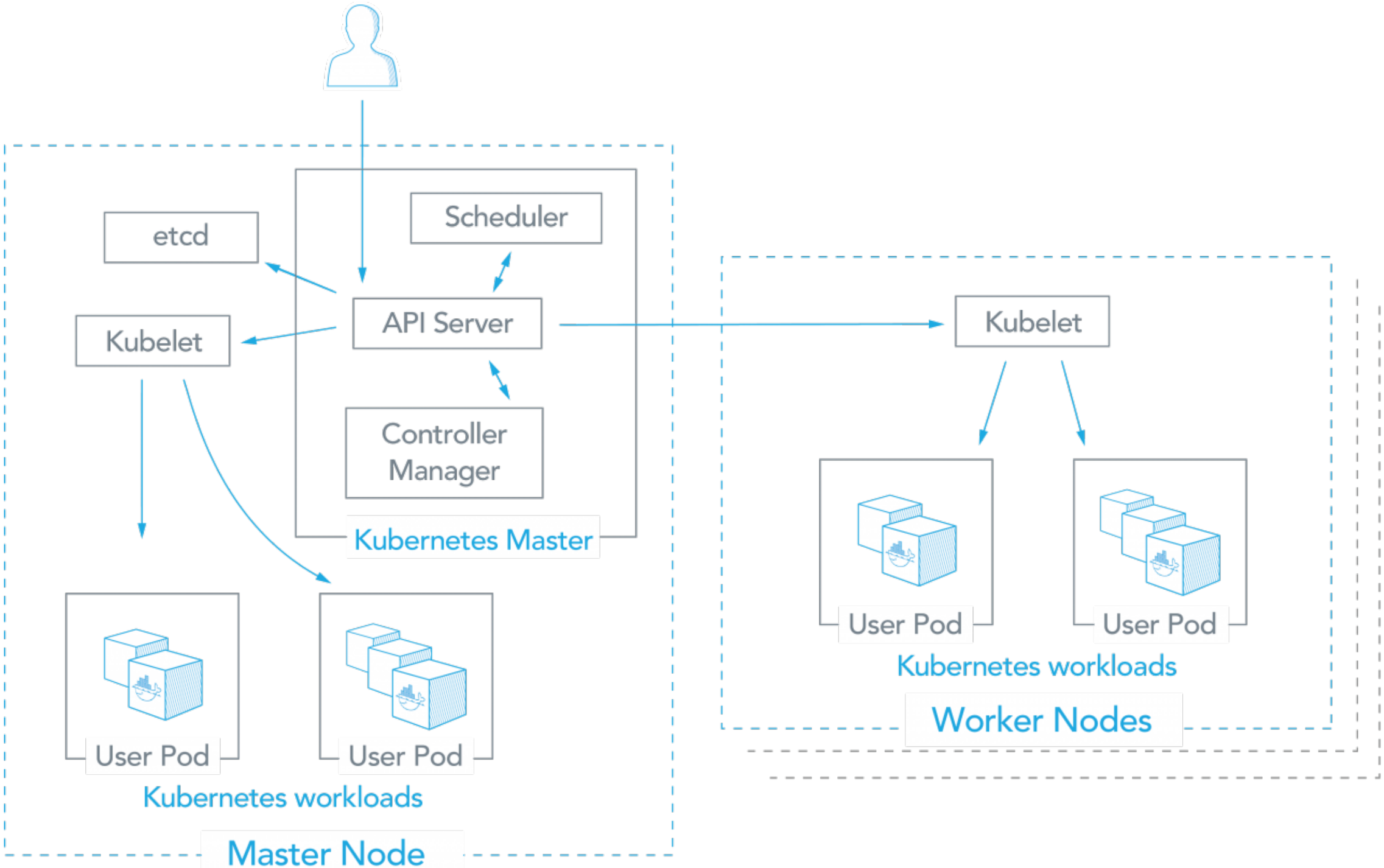
Open Source Containers: Kubernetes

Greek for *“Helmsman”*; also the root of the word *“Governor”* and *“cybernetic”*

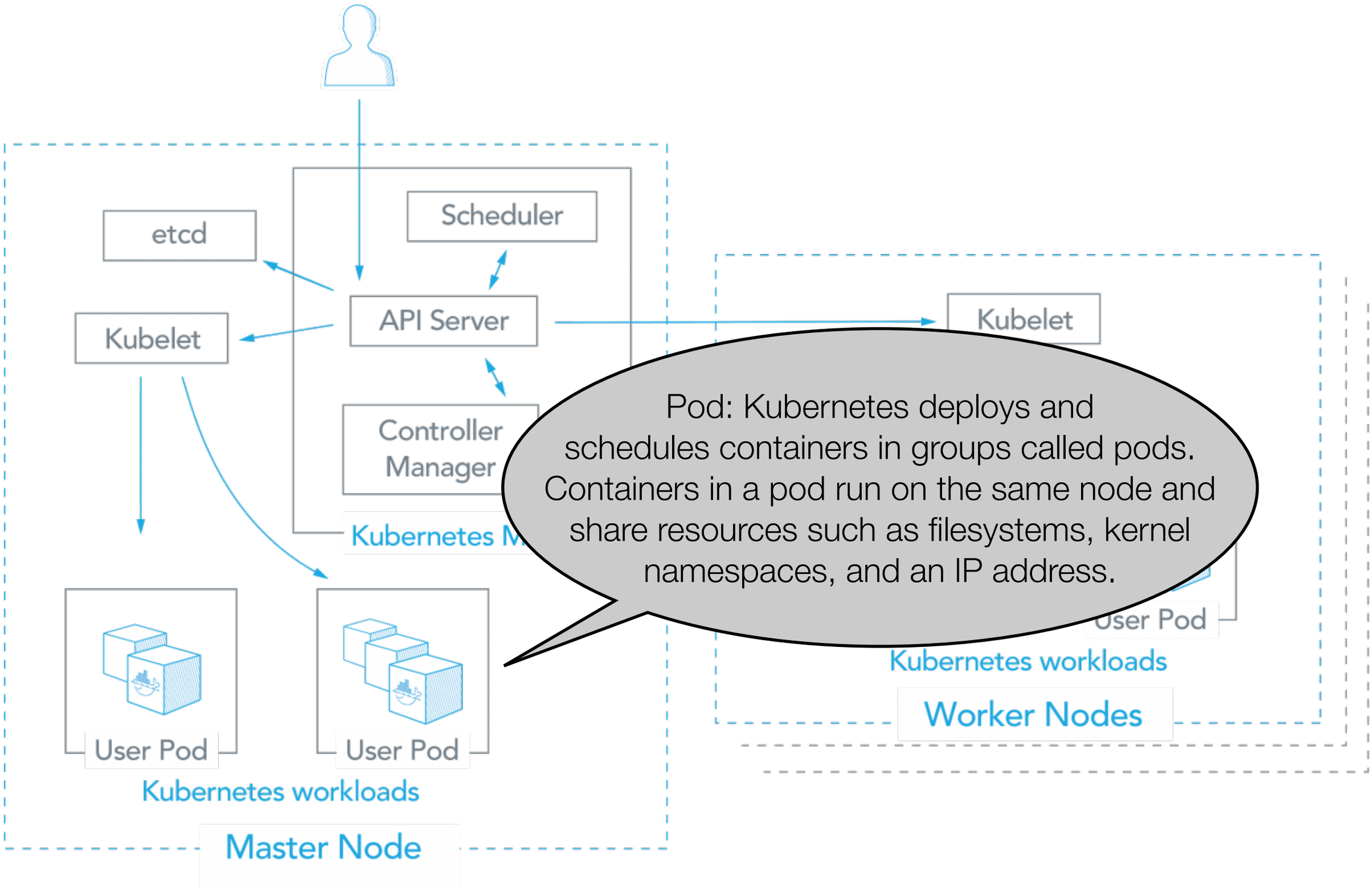
- Container orchestrator
- Builds on Docker containers
  - also supporting other container technologies
- Multiple cloud and bare-metal environments
- Supports existing OSS apps
  - cannot require apps becoming cloud-native
- Inspired and informed by Google’s experiences and internal systems
- **100% Open source**, written in **Go**

Let users manage **applications**, not machines

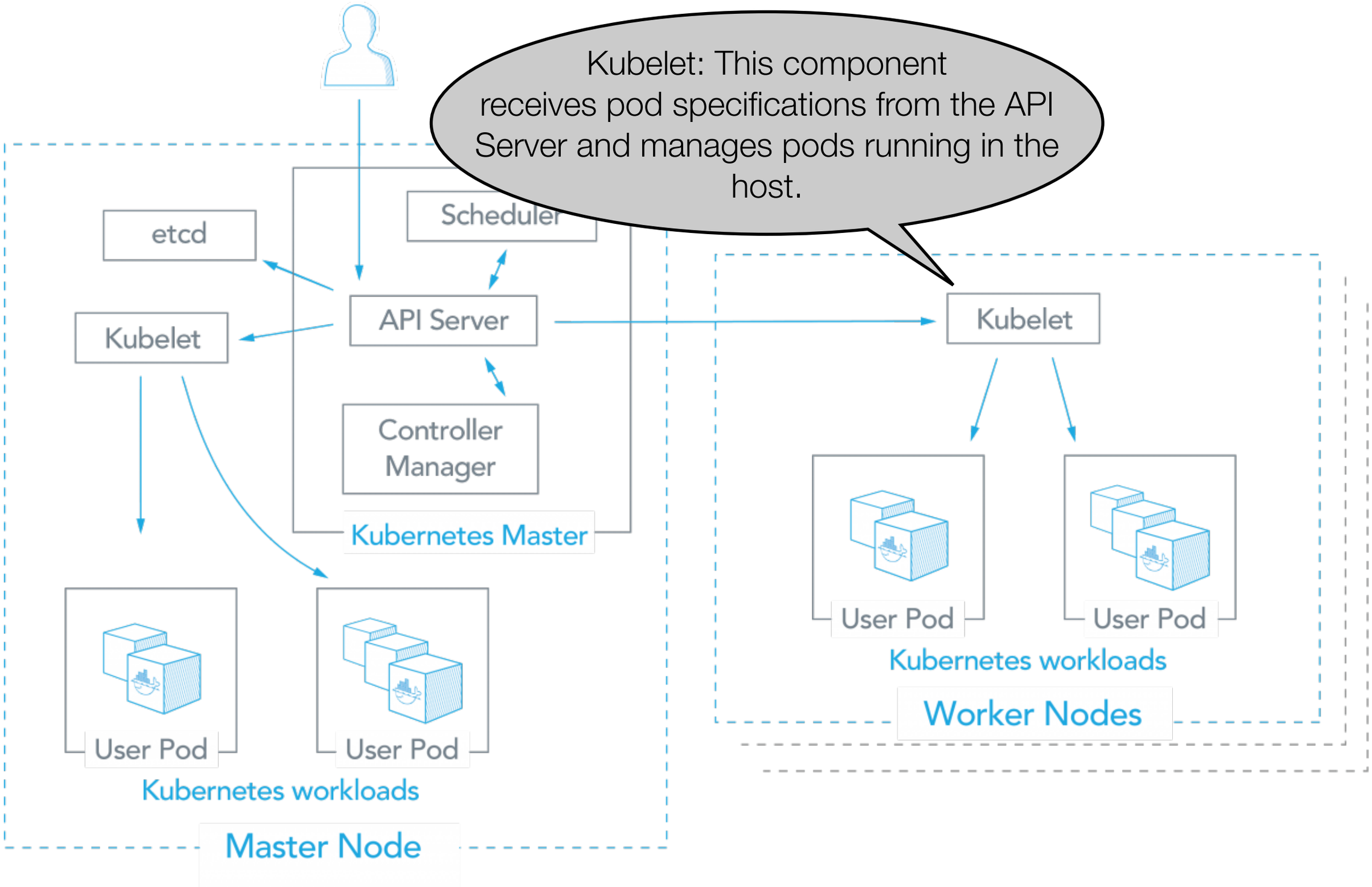
Architecture



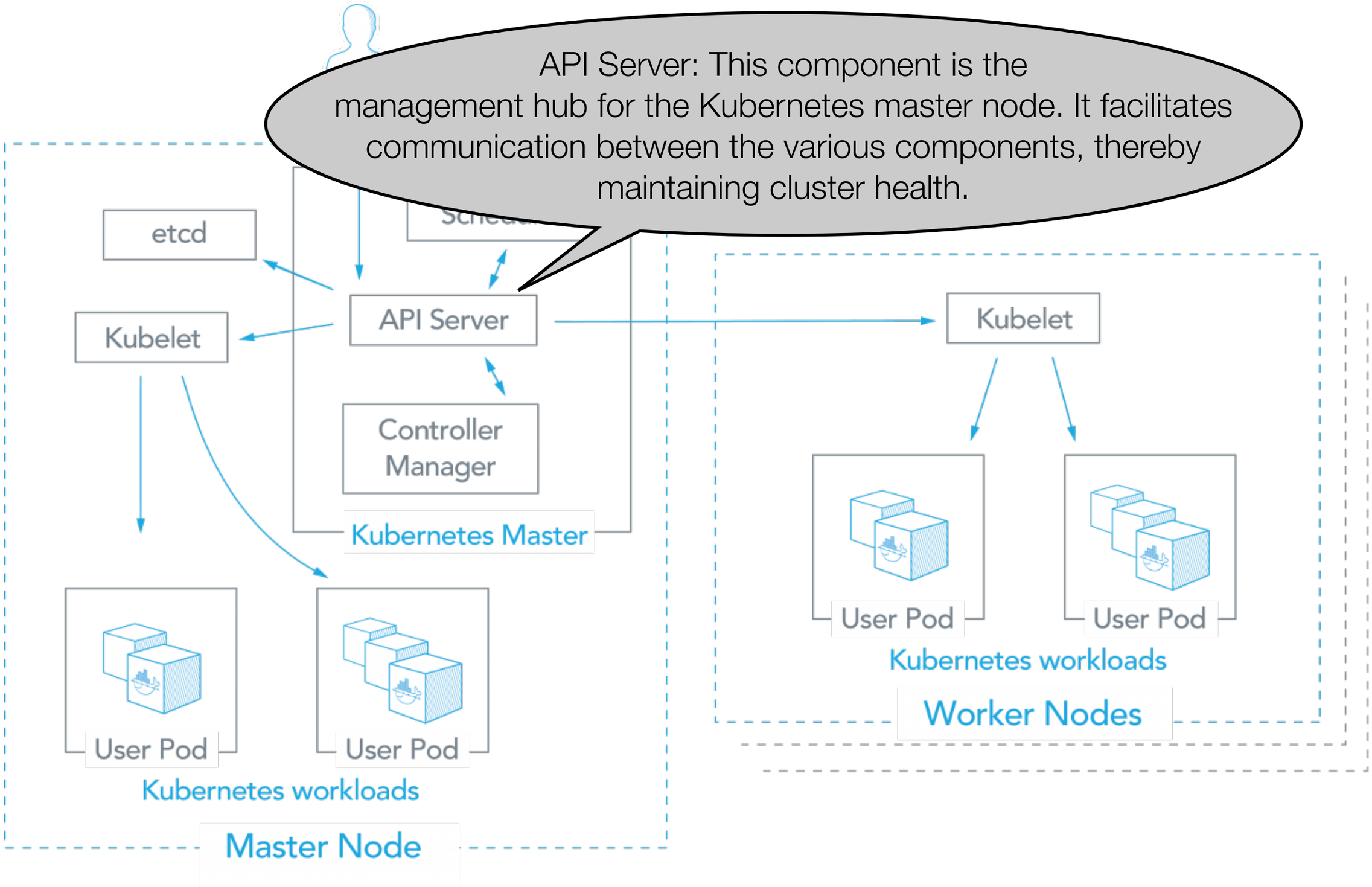
Architecture



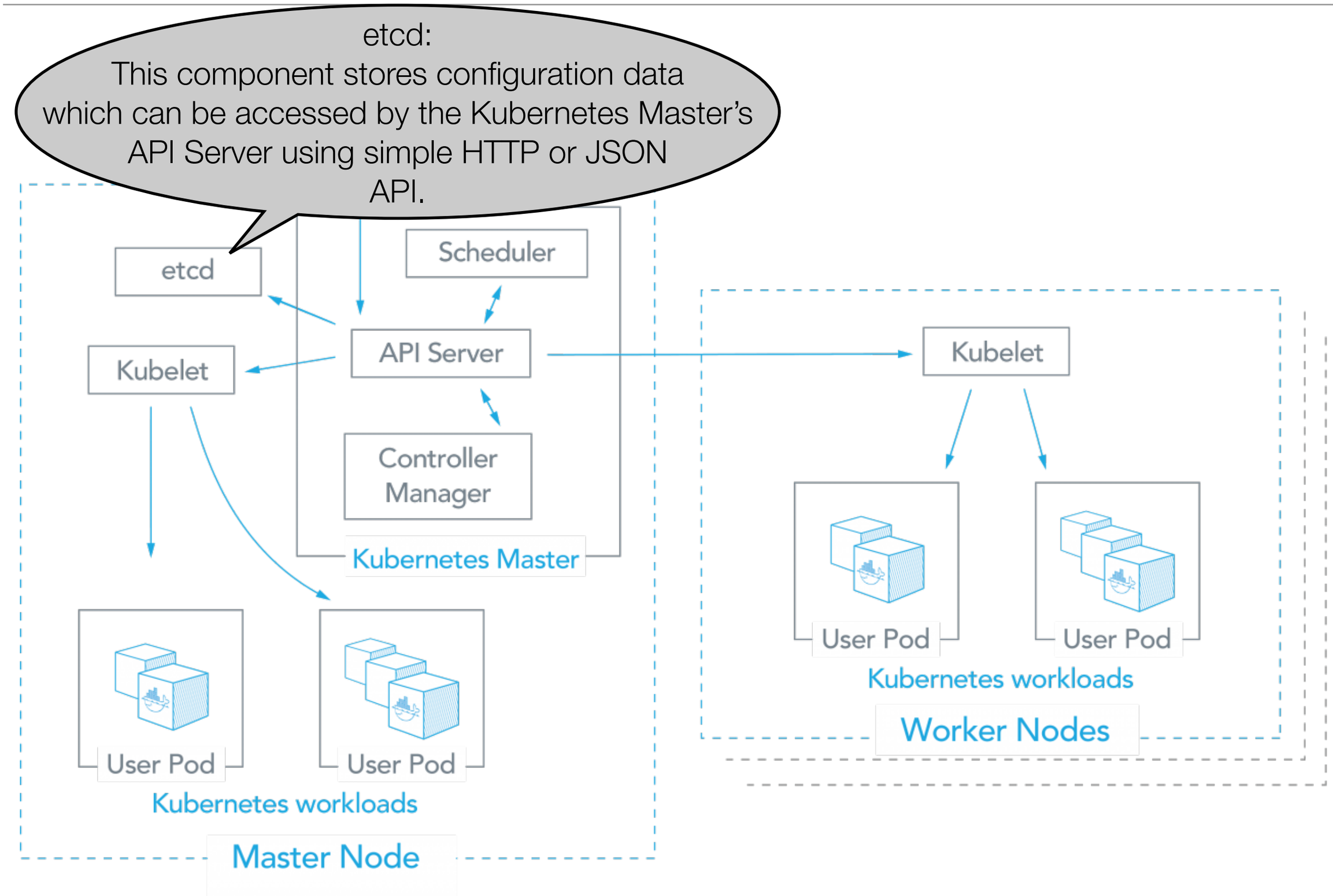
Architecture



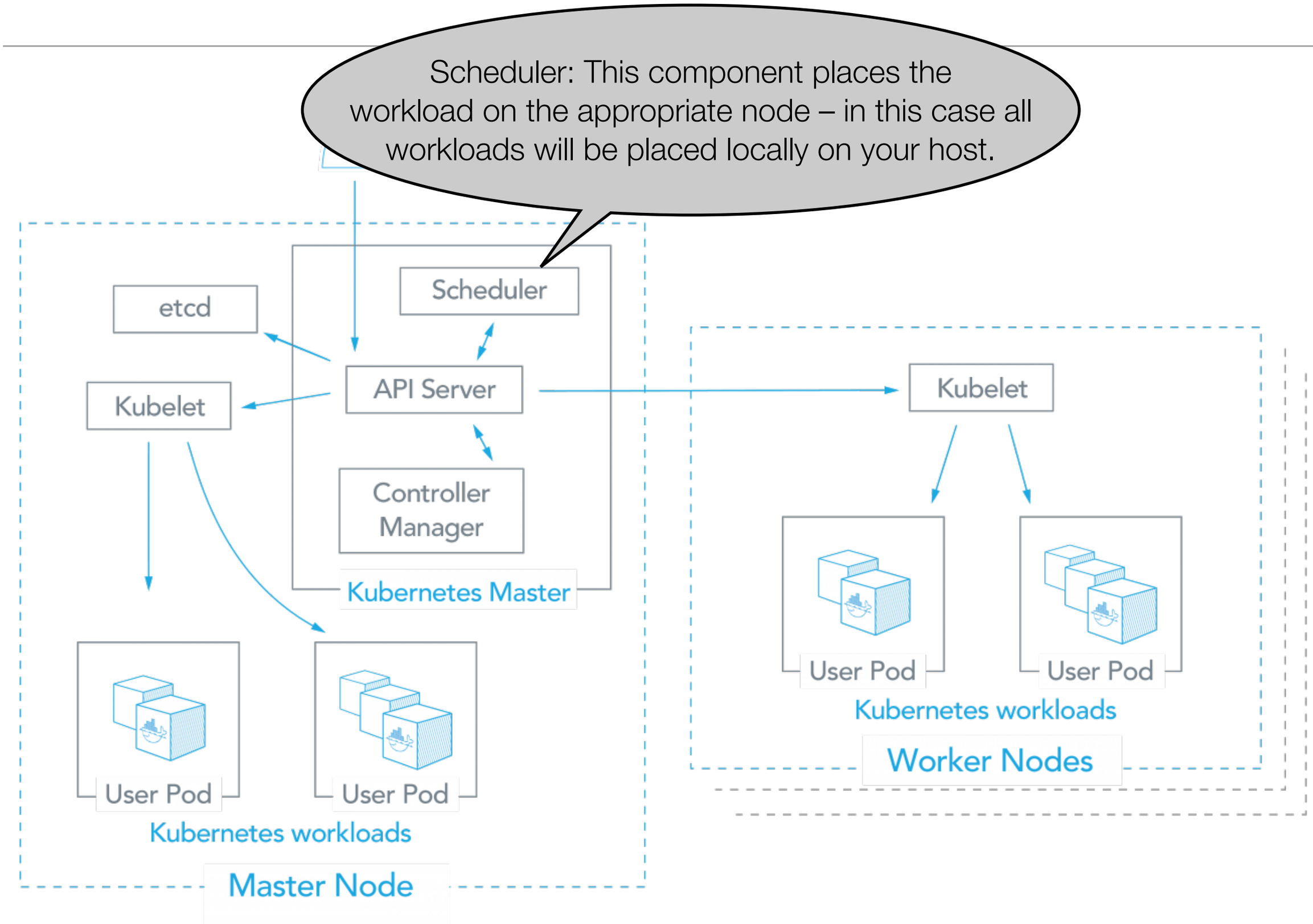
Architecture



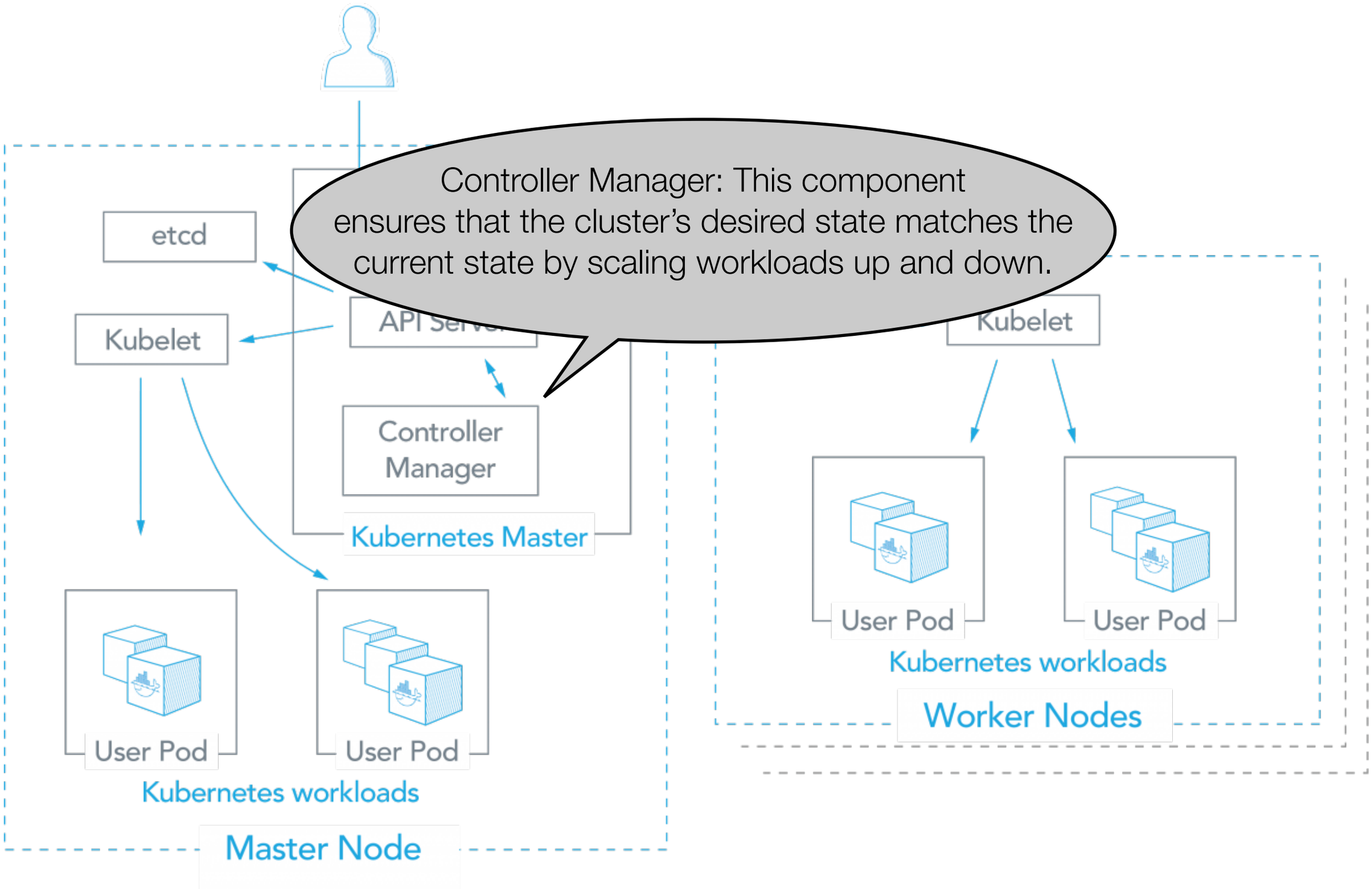
Architecture



Architecture

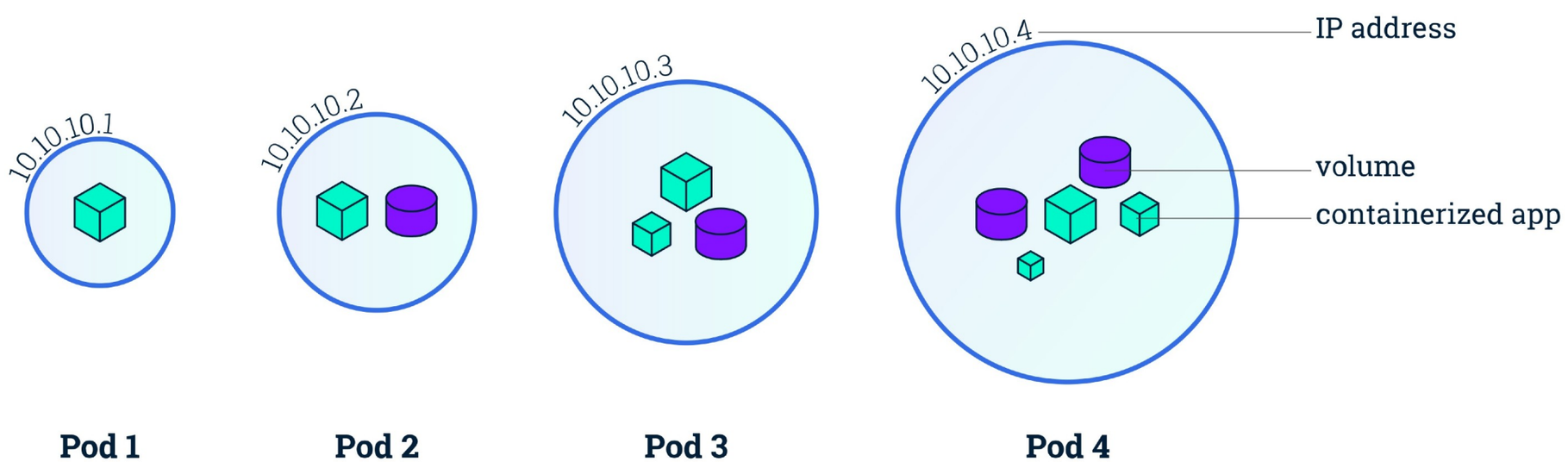


Architecture



Pod

- a Kubernetes abstraction that represents a group of one or more application containers, and some shared resources for those containers
  - Shared storage, as Volumes
  - Networking, as a unique cluster IP address
- Information about how to run each container, such as the container image version or specific ports to use



Pod

- Simplest unit in Kubernetes
- Represents processes running in your cluster
- Encapsulates a container (or sometimes multiple)
- Replicating a Pod serves to scale an application horizontally

```
apiVersion: v1
kind: Pod
metadata:
      name: nginx
spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Pod Lifecycle

Once scheduled to a node, pods do not move

- restart policy means restart **in-place**

Pods can be observed *pending*, *running*, *succeeded*, or *failed*

- *failed* is **really** the end - no more restarts
- no complex state machine logic

Pods are **not rescheduled** by the scheduler or apiserver

- even if a node dies
- controllers are responsible for this
- keeps the scheduler **simple**

ReplicaSet

- Maintains a set of identical Pods
- Definition consists of:
  - Number of replicas
  - Pod template
  - Selector to identify which Pods it can acquire
- Generally encapsulated by a Deployment

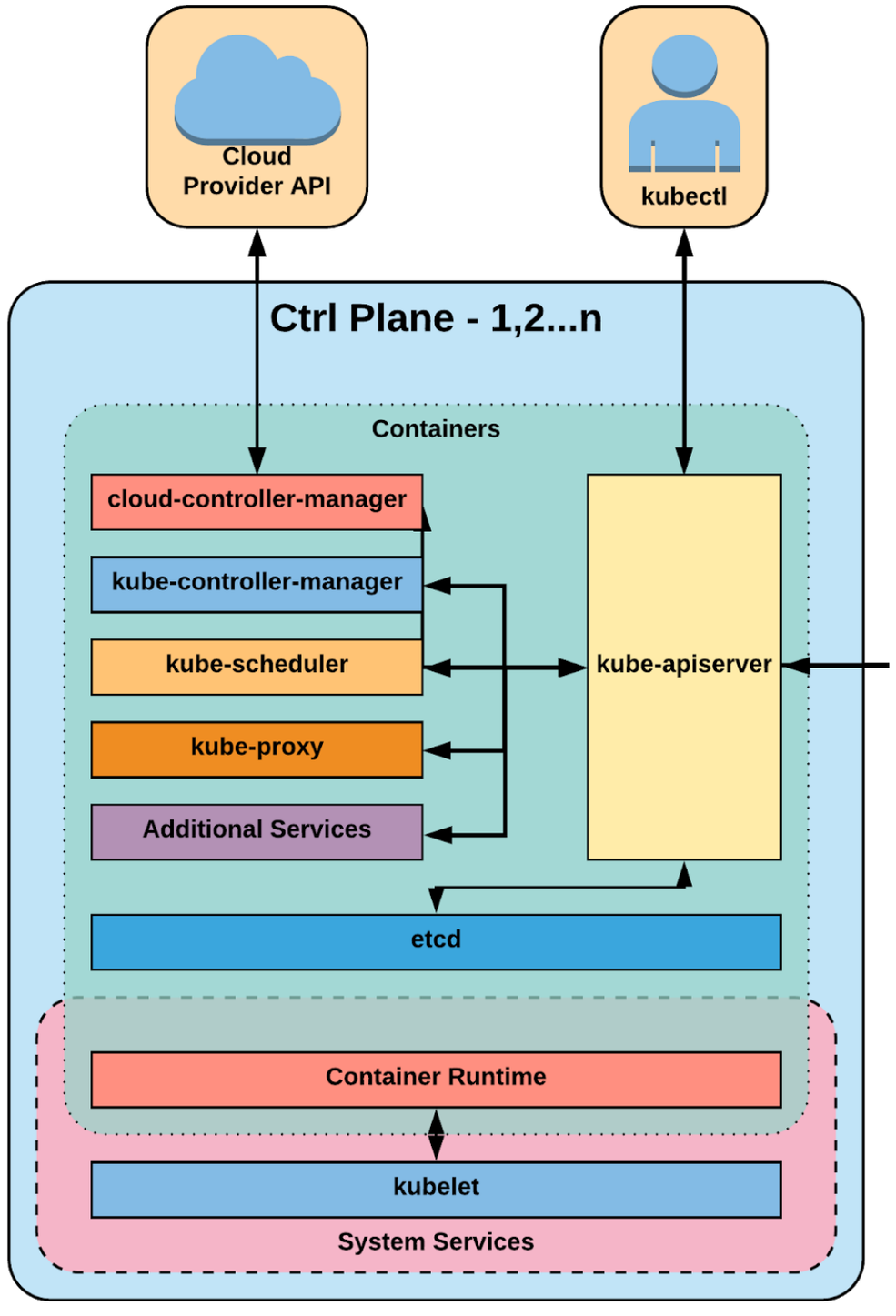
```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-replicaset
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Deployment

- Provides updates for Pods and ReplicaSets
- Runs multiple replicas of your application
- Suitable for stateless applications

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Control Plane



kube-apiserver

- Provides a forward facing REST interface into the Kubernetes control plane and datastore

- All clients and other applications interact with Kubernetes **strictly** through the API Server

- Acts as the gatekeeper to the cluster by handling authentication and authorization, request validation, mutation, and admission control in addition to being the front-end to the backing datastore

kube-controller-manager

- Monitors the cluster state via the apiserver and **steers the cluster towards the desired state**

- **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.

kube-scheduler

- Component on the master that watches newly created pods that have no node assigned, and selects a node for them to run on
- Factors taken into account for scheduling decisions include individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference and deadlines

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

etcd

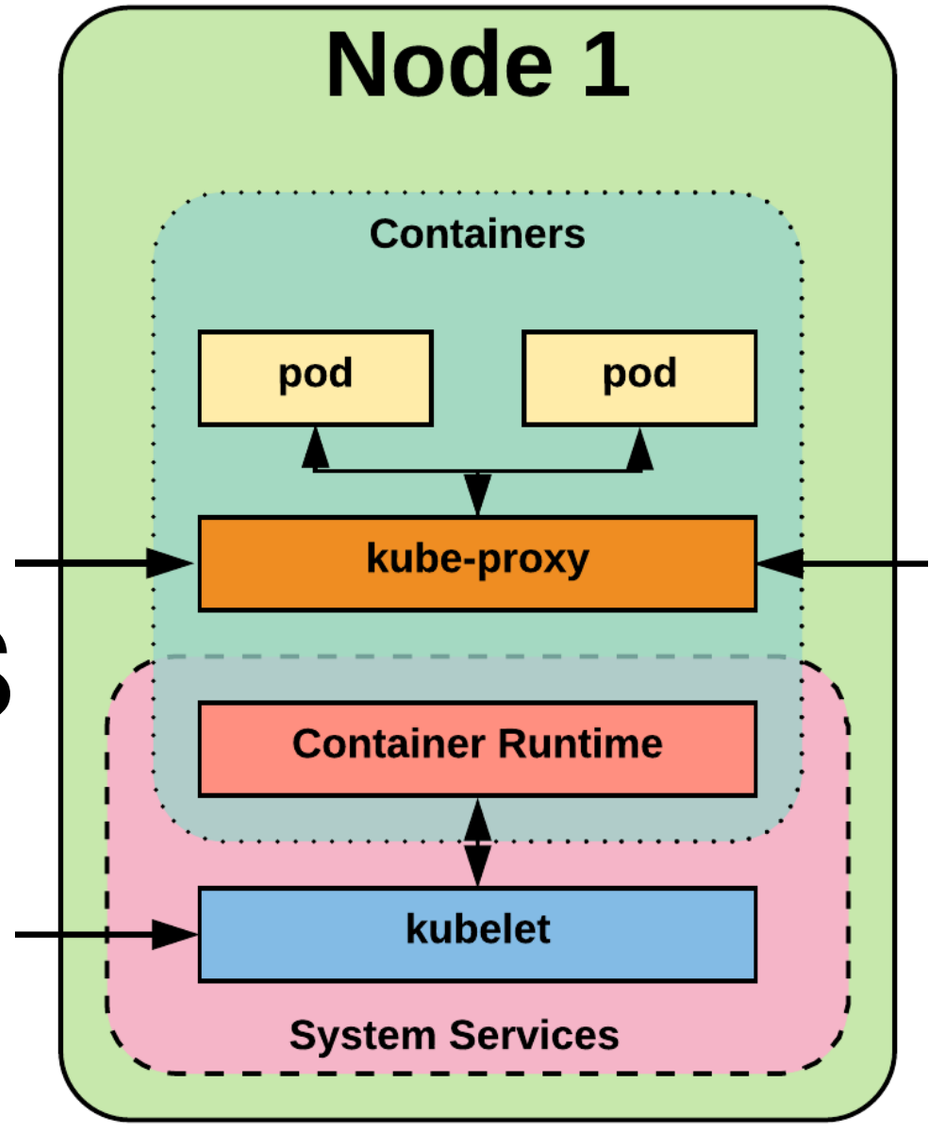
- etcd: an atomic key-value store that uses Raft consensus

- Backing store for all control plane metadata

- Provides a strong, consistent and highly available key-value store for persisting cluster state

- Stores objects and config information

Node Components



kubelet

- An agent that runs on each node in the cluster. It makes sure that containers are running in a pod.

- The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy.

kube-proxy

- Manages the network rules on each node.
- Performs connection forwarding or load balancing for Kubernetes cluster services.

Container Runtime Engine

- A container runtime is a CRI (Container Runtime Interface) compatible application that executes and manages containers.
  - Containerd (docker)
  - Cri-o
  - Rkt
  - Kata (formerly clear and hyper)
  - Virtlet (VM CRI compatible runtime)

Borg

Borg

Cluster management system at Google that achieves high utilization by:

- Admission control
- Efficient task-packing
- Over-commitment
- Machine sharing

The User Perspective

- Allocs
  - Reserved set of resources
- Priority, Quota, and Admission Control
  - Job has a priority (preempting)
  - Quota is used to decide which jobs to admit for scheduling
- Naming and Monitoring
  - 50.jfoo.ubar.cc.borg.google.com
  - Monitoring health of the task and thousands of performance metrics

Scheduling a Job

```
job hello_world = {
  runtime = { cell = “ic” } //what cell should run it in?
  binary = ‘../hello_world_webserver’ //what program to run?
  args = { port = ‘%port%’ }
  requirements = {
    RAM = 100M
    disk = 100M
    CPU = 0.1
  }
  replicas = 10000
}
```

Borg <—> Kubernetes

Directly derived

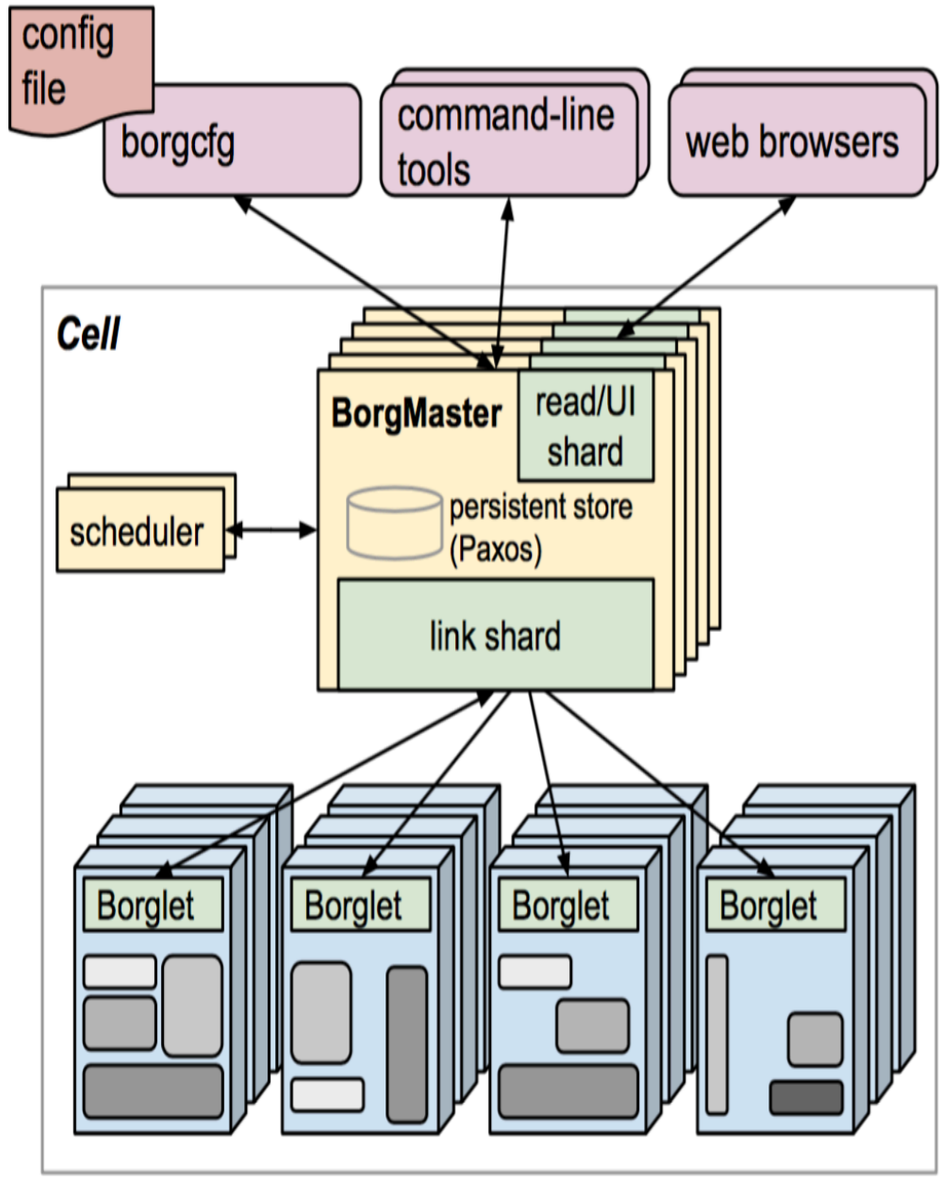
- Borglet => Kubelet
- alloc => pod
- Borg containers => docker
- Declarative specifications

loosely inspired by Borg

- Job => labels
- managed ports => IP per pod
- Monolithic master => micro-services

Borg Architecture

- Borgmaster
  - Main Borgmaster process & Scheduler
  - Five replicas
- Borglet
  - Manage and monitor tasks and resource
  - Borgmaster polls Borglet every few seconds



Borg's Allocation Algorithms and Policies

Advanced Bin-Packing algorithms:

- Avoid stranding of resources

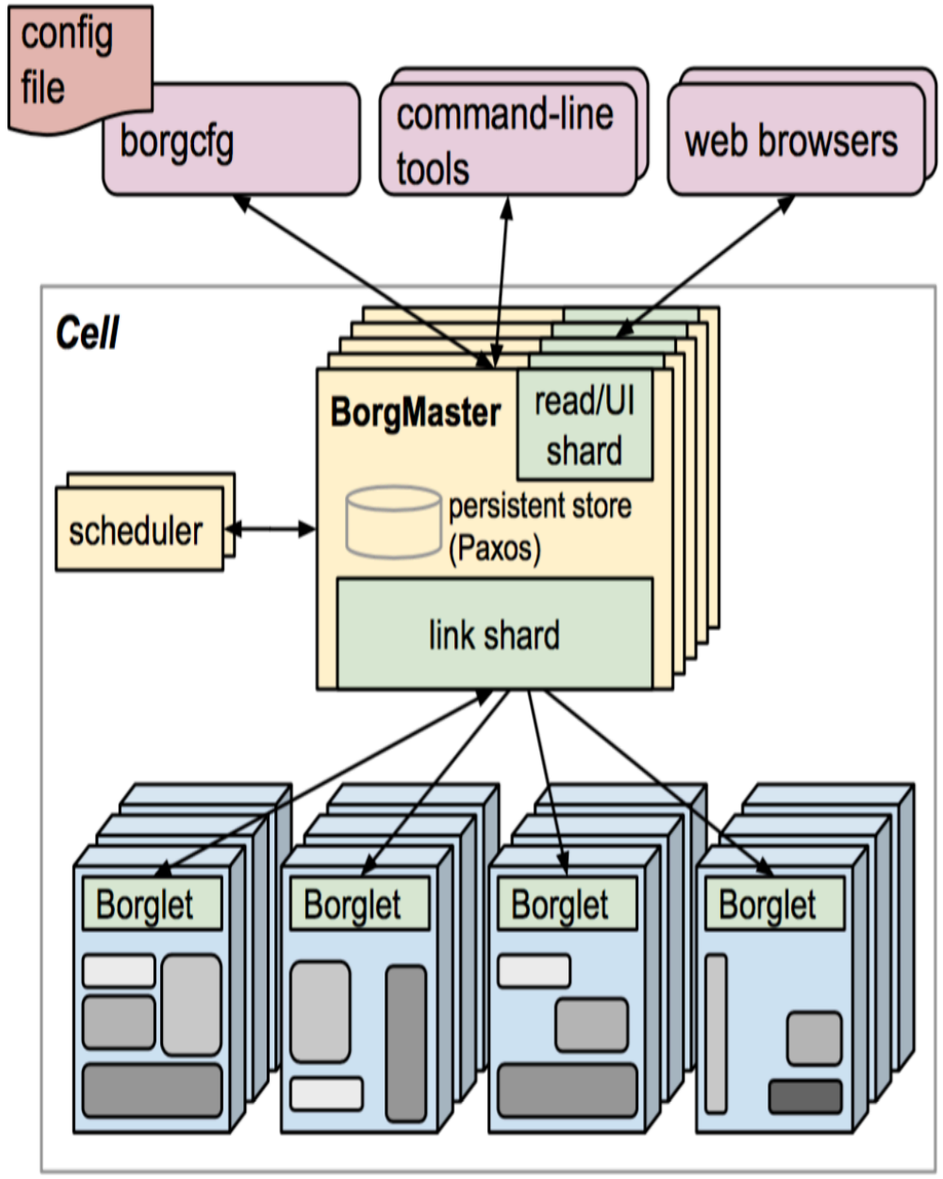
Evaluation metric: Cell-compaction

- Find smallest cell that we can pack the workload into…
- Remove machines randomly from a cell to maintain cell heterogeneity

Evaluated various policies to understand the cost, in terms of extra machines needed for packing the same workload

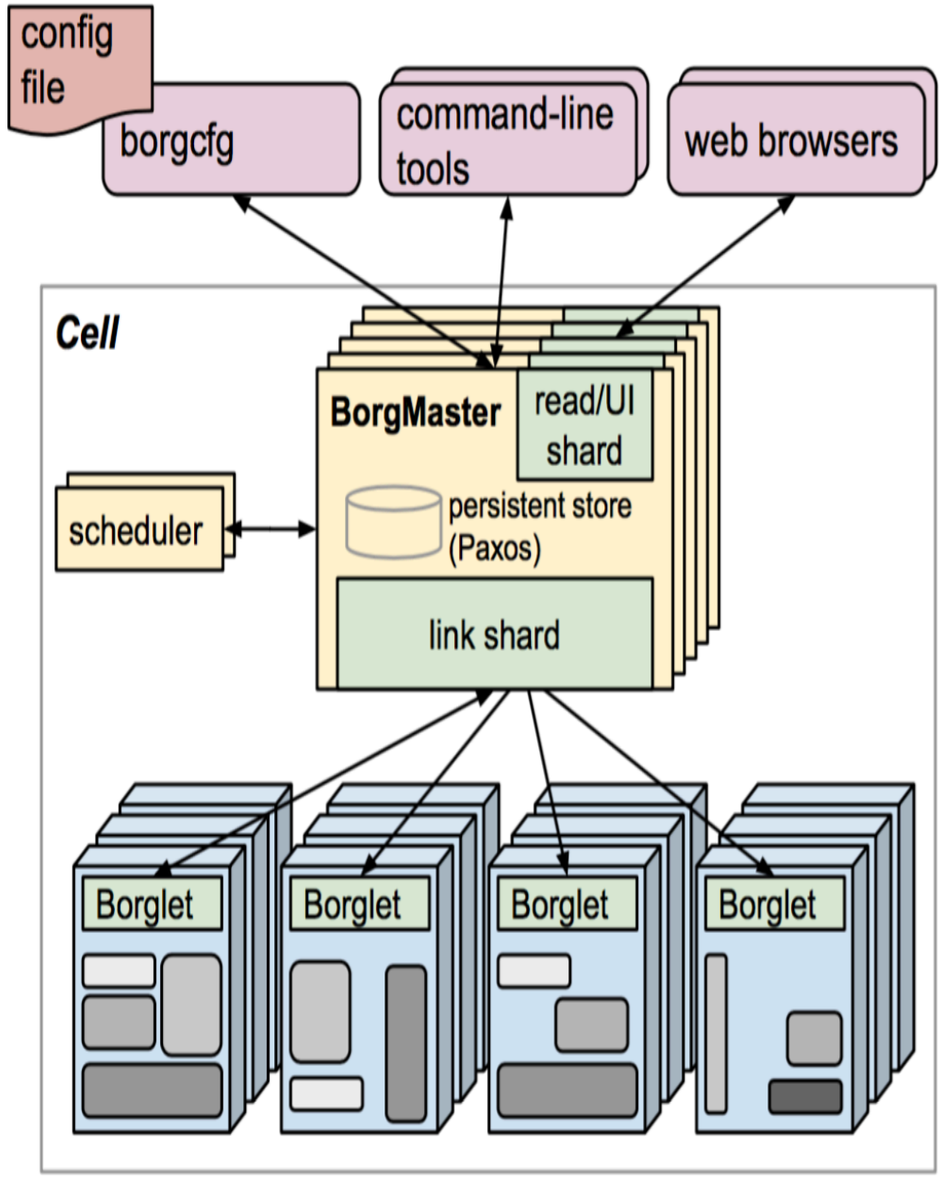
Scheduling

- Feasibility checking: find machines for a given job
- Scoring: pick one machines
  - User prefs & built-in criteria
    - Minimize the number and priority of the preempted tasks
    - Picking machines that already have a copy of the task's packages
    - spreading tasks across power and failure domains
    - Packing by mixing high and low priority tasks



Scalability

- Separate scheduler
- Separate threads to poll the Borglets
- Partition functions across the five replicas
- Score caching
- Equivalence classes
- Relaxed randomization



Next: Hadoop