Introduction to

kubernetes

About Us

Péter Megyesi

PhD in Telecommunications @ BME

- Worked with 5G technology
- SDN & NFV → Cloud Native Network Functions
- Graduated in the EIT Digital Doctoral School

Co-founder & CTO @ LeanNet Ltd.

- Consulting, training, implementing
- Cloud Native, Microservices, DevOps

Dávid Szabó

✉ megyesi@leannet.eu

twitter.com/M3gy0

linkedin.com/in/M3gy0

szabo@leannet.eu ✉

twitter.com/szabvid

linkedin.com/in/szabvid

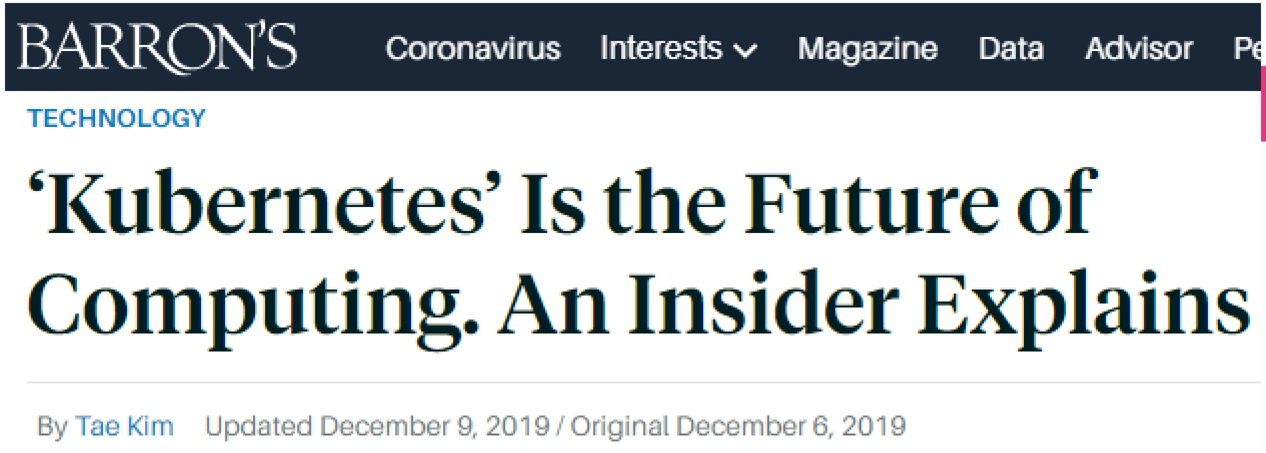
Housekeeping

1. Join to Slack channel:
   - https://hwswkubernetes.slack.com

2. Every code is on GitLab
   - Send us your GitLab username on Slack, we will ad you to the repo

Course Outline

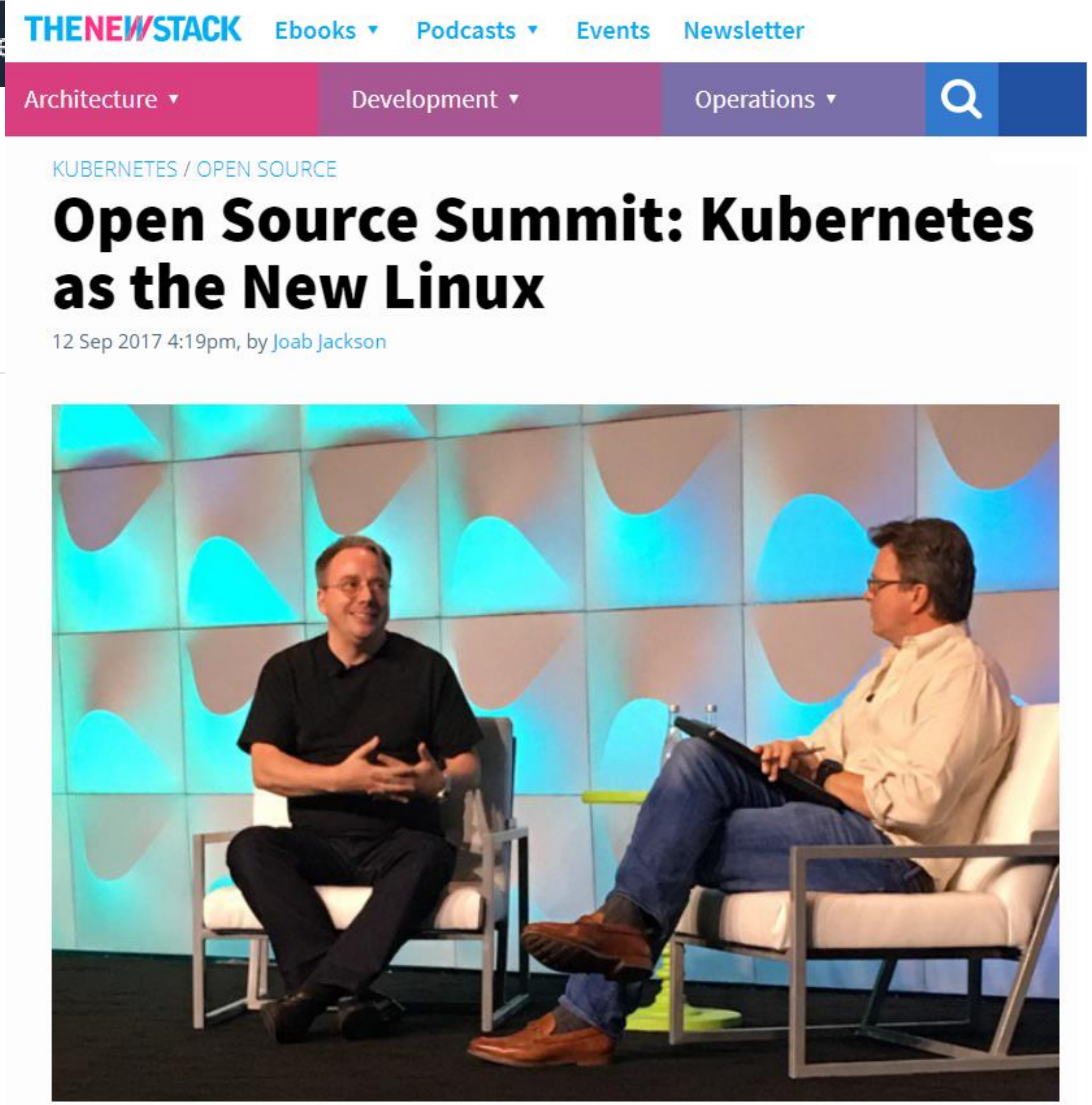
1. What is Kubernetes?
   - Components
   - Installation
2. Basics of Docker
   - Namespaces
   - Building and running Docker images
3. Pods and Deployments
   - Running basic workloads in Kubernetes
   - Scale, Update, Rollback
4. Advanced Pod configuration
   - Args, Envs, ConfigMaps, Secrets
   - Init- and sidecar containers
   - Scheduling and debugging
5. Networking in Kubernetes
   - What are network plugins?
   - Service abstraction and ingress
6. Persistent storage
   - Basics of storage: block vs. object vs. file system
   - StoragesClass, PVC, PV
7. Security
   - RBAC: Roles, ServiceAccounts, RoleBindings
   - Security context and network security policy
8. Advanced topics
   - Helm
   - Custom resources and operators



TECHNOLOGY

‘Kubernetes’ Is the Future of Computing. An Insider Explains

By Tae Kim   Updated December 9, 2019 / Original December 6, 2019

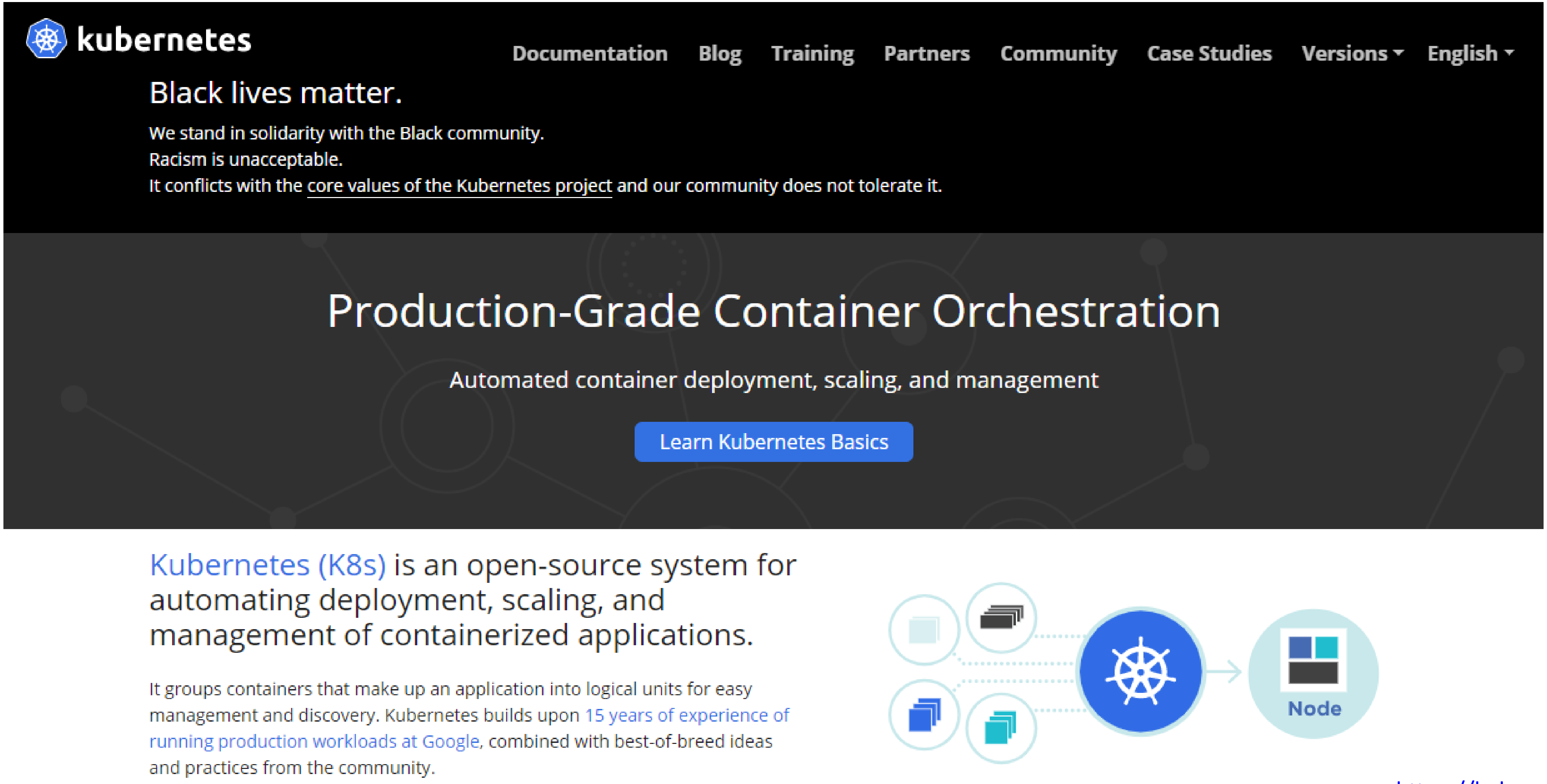


KUBERNETES / OPEN SOURCE

Open Source Summit: Kubernetes as the New Linux

12 Sep 2017 4:19pm, by Joab Jackson

What is Kubernetes?



Black lives matter.

We stand in solidarity with the Black community.
Racism is unacceptable.
It conflicts with the core values of the Kubernetes project and our community does not tolerate it.

Production-Grade Container Orchestration

Automated container deployment, scaling, and management

Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications.

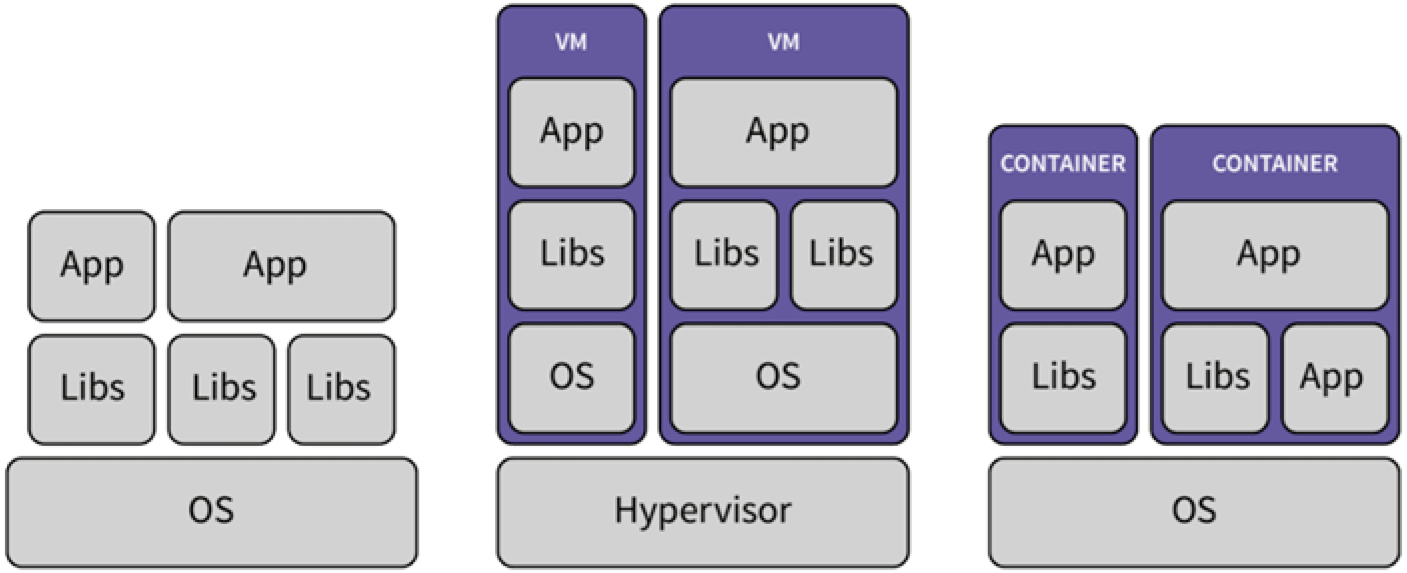
It groups containers that make up an application into logical units for easy management and discovery. Kubernetes builds upon 15 years of experience of running production workloads at Google, combined with best-of-breed ideas and practices from the community.

https://kubernetes.io/

What is a Container?

Containers are an application-centric way to deliver high-performing, scalable applications on the infrastructure of your choice

- A **bundle** of the **application** code along with its **runtime** and **dependencies**
- It creates an **immutable isolated executable** environment, also known as container image
- It can be **deployed** on the platform of your choice, such as desktops, servers, VMs or in the cloud



Bare Metal

- Deploy in months
- Live for years

Virtualization

- Deploy in minutes/hours
- Live for weeks

Containers

- Deploy in seconds
- Live for hours/days

- Deploy in milliseconds
- Live for seconds

LXC LXD

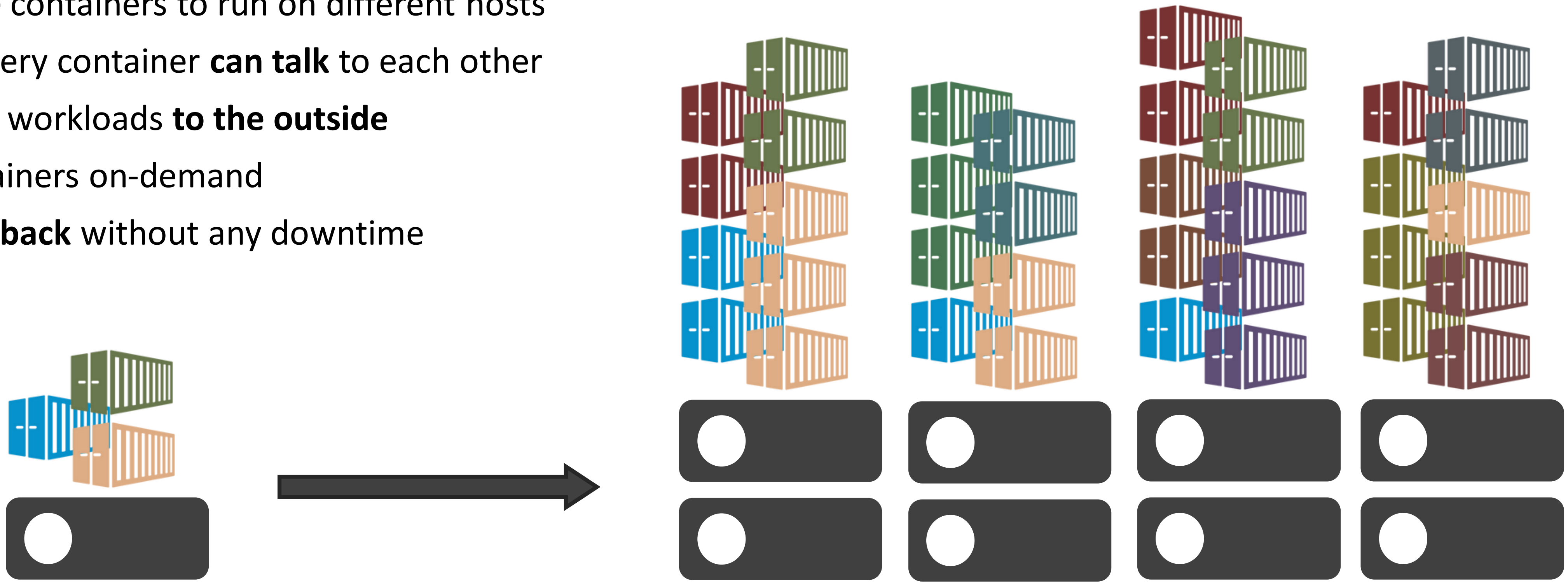
What is a Container Orchestration?

Running a **few containers** on a server is easy ☺

Running **hundreds of containers** on dozens of servers can be handled manually ☹

So container orchestration:

- group hosts together to form a **cluster**
- optimally **schedule** containers to run on different hosts
- **guarantees** that every container **can talk** to each other
- can **expose** certain workloads **to the outside**
- can **scale** out containers on-demand
- can do **update/rollback** without any downtime



And I Bet You'll Need Orchestration

Why? Because all these companies have built a similar systems:

- Sigma by Alibaba Group
- Apollo by amazon.com
- Apache Mesos Apache MESOS
- Matrix by Baidu 百度
- Cloud Foundry CLOUD FOUNDRY
- Fleet by CoreOS
- Swarm by docker
- Tupperware by f

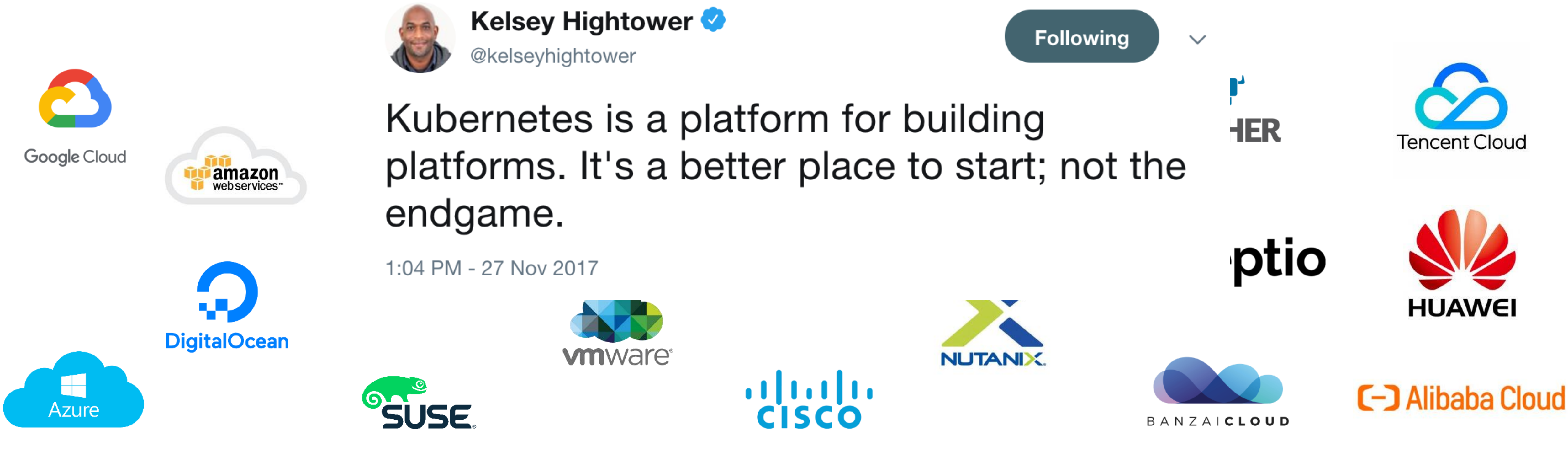
- Borg and Omega by Google
- Nomad by HashiCorp
- Platform Symphony by IBM
- Triton by Joyent
- v3 Infra by lyft
- Service Fabric by Microsoft
- Titus by NETFLIX

- Cattle by RANCHER
- OpenShift v2 by redhat
- Helios by Spotify
- Gaia by Tencent 腾讯
- Aurora by twitter
- Peloton by UBER

@dankohn1: Stitching Things Together – KubeCon EU '19 Keynote

So Why Kubernetes?

- **Open-source system** for automating deployment, scaling, and management of containerized applications
- Builds upon **15 years of experience** of running production workloads at **Google**
- Designed to be **extendable**, so that future solutions could be integrated into it
- **Avoids** vendor or cloud-provider **lock-in**, trademark the **CNCF** (Cloud Native Computing Foundation)
- Basically the whole IT industry is backing it:

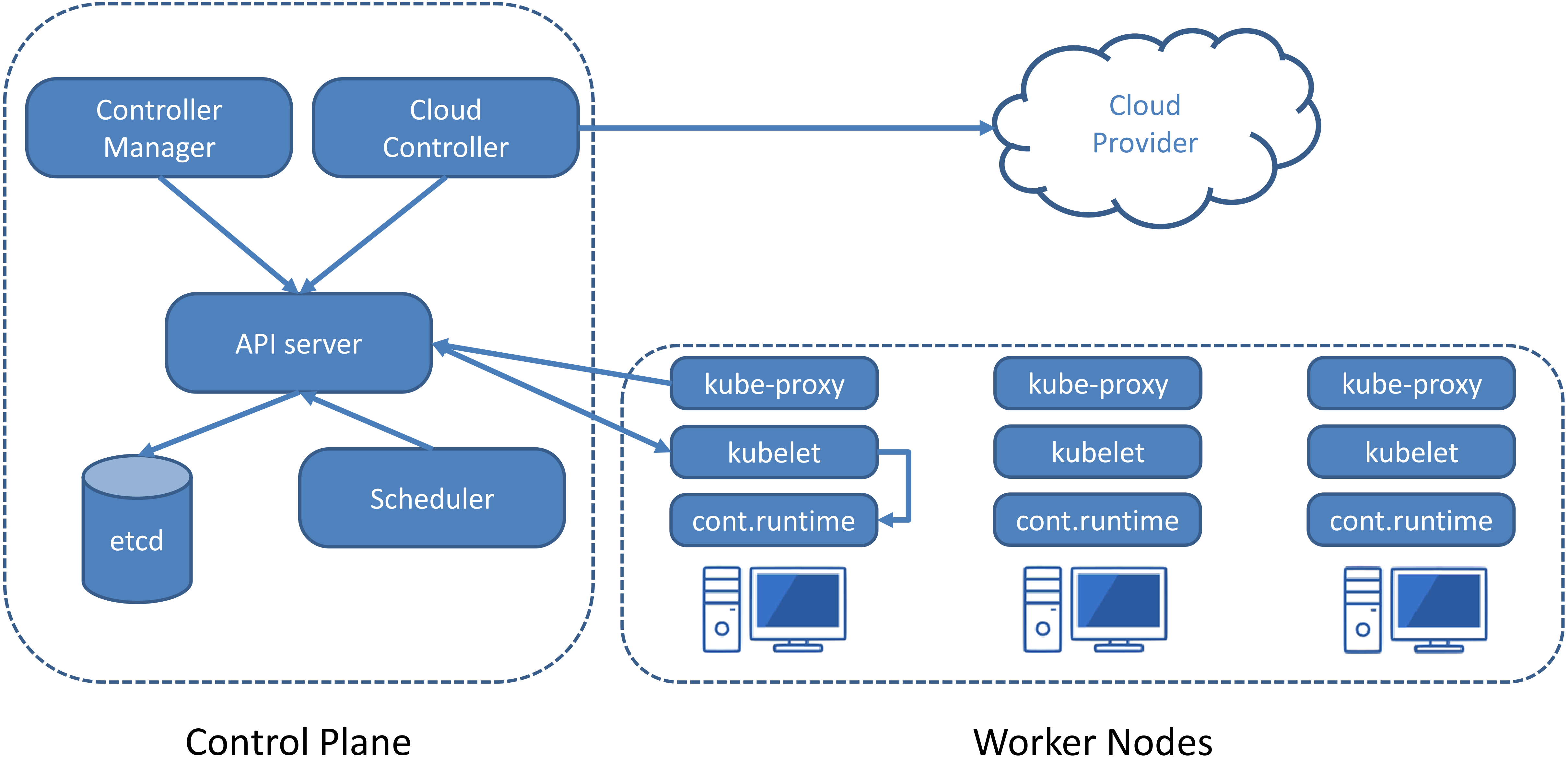


Kelsey Hightower @kelseyhightower

Kubernetes is a platform for building platforms. It's a better place to start; not the endgame.

1:04 PM - 27 Nov 2017

Kubernetes Architecture



Kubernetes API Server

HTTP server that exposes the Kubernetes API

Front end for the Kubernetes control plane:

- Validates and configures data for the **API objects**
- Services REST operations and provides the frontend to the cluster's shared state
- All other components interact solely via the API Server

Designed to scale horizontally:

- It scales by deploying more instances
- You can run several instances of kube-apiserver and balance traffic between those instances

Basic Objects in Kubernetes

Pod

- Unit of deployment
- Group of one or more containers
- Container = unit of packaging

ReplicaSet

- Groups uniform Pods
- Ensures availability and scalability

Deployment

- Groups uniform ReplicaSets
- Ensures updates and rollbacks

Jobs

- Run to completion Pods

Services

- Collection of pods exposed as an endpoint

Ingress

- Represent an HTTP(S) endpoint inside the cluster that is accessible externally

PersistentVolume

- Represent a persistent block volume backed by a (usually HA) storage unit

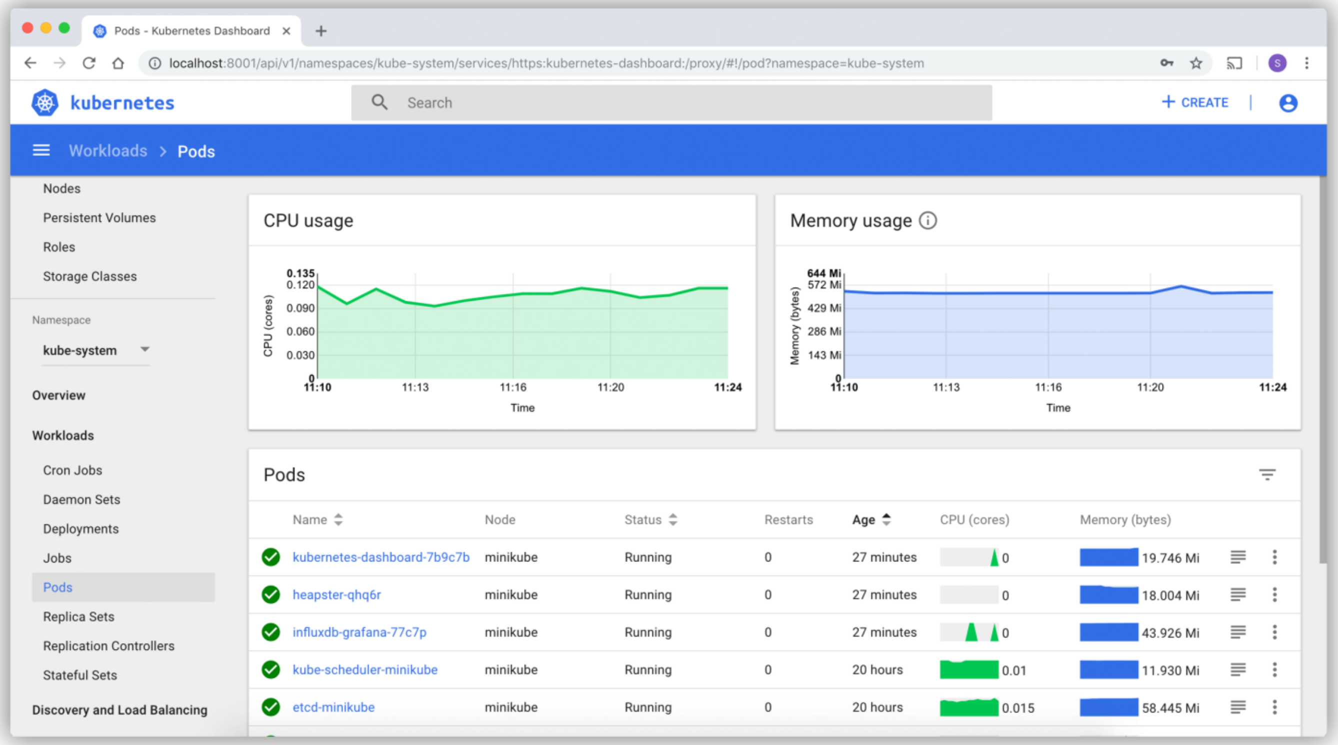
ConfigMap

- Mountable read-only config files for Pods

Interacting with the API Server

`kubectl`

- The official CLI client
- Most widely used



Pods

Name	Node	Status	Restarts	Age	CPU (cores)	Memory (bytes)
kubernetes-dashboard-7b9c7b	minikube	Running	0	27 minutes	0	19.746 Mi
heapster-qhq6r	minikube	Running	0	27 minutes	0	18.004 Mi
influxdb-grafana-77c7p	minikube	Running	0	27 minutes	0	43.926 Mi
kube-scheduler-minikube	minikube	Running	0	20 hours	0.01	11.930 Mi
etcd-minikube	minikube	Running	0	20 hours	0.015	58.445 Mi

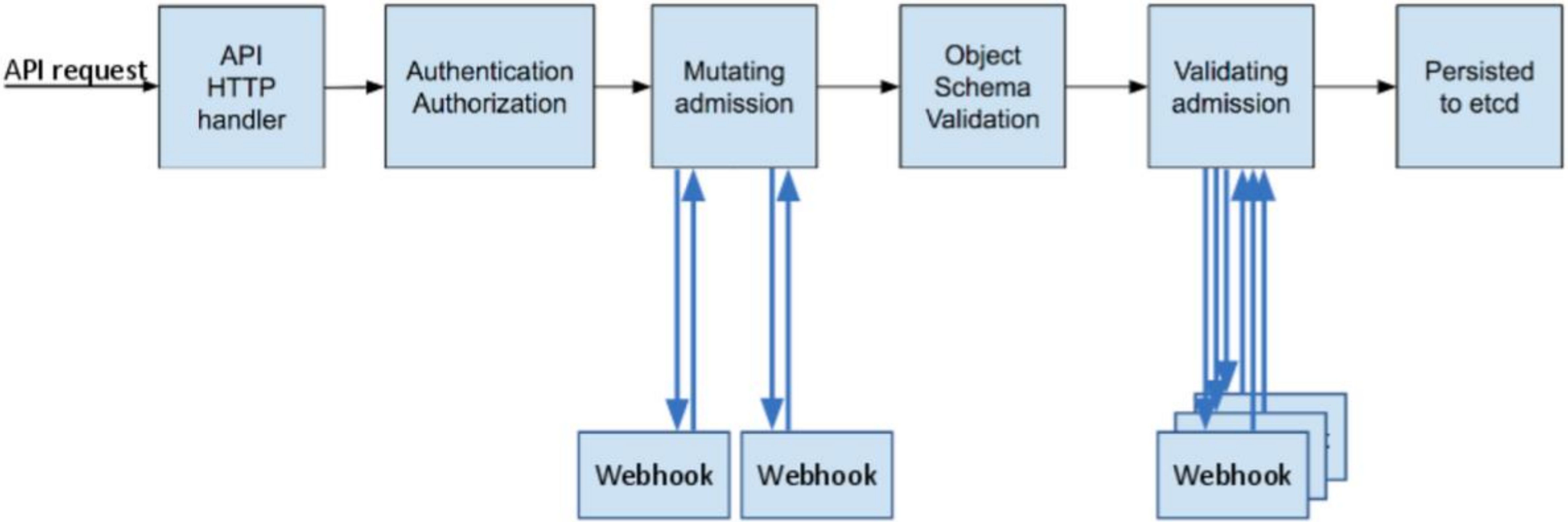
GUI

- https://github.com/kubernetes/dashboard

REST API in JSON format

- `curl`
- Language specific libraries
  - Official: Go, Python, Java, .NET, JavaScript, Heskell
  - Community: Clojure, Lisp, TypeScript, Perl, PHP, Ruby, Rust, Scala, Elixir
  - https://kubernetes.io/docs/reference/using-api/client-libraries/

Admission Control in the API Server



What is ETCD?

Distributed, reliable key-value store

Features:

- Written in Go
- HTTP REST API with optional SSL client certificate authentication
- Store data in hierarchically organized directories, as in a standard filesystem
- Watch specific keys or directories for changes and react to changes in values

High availability is based on distributed consensus:

- Raft algorithm: https://raft.github.io/
- Use odd number of nods: (n-1)/2 nodes can be unavailable for the consensus to work

What is the Scheduler?

Watches for newly created Pods with no assigned node, 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
- deadlines

What is the Controller Manager?

Control Plane component that runs controller processes

Controller:

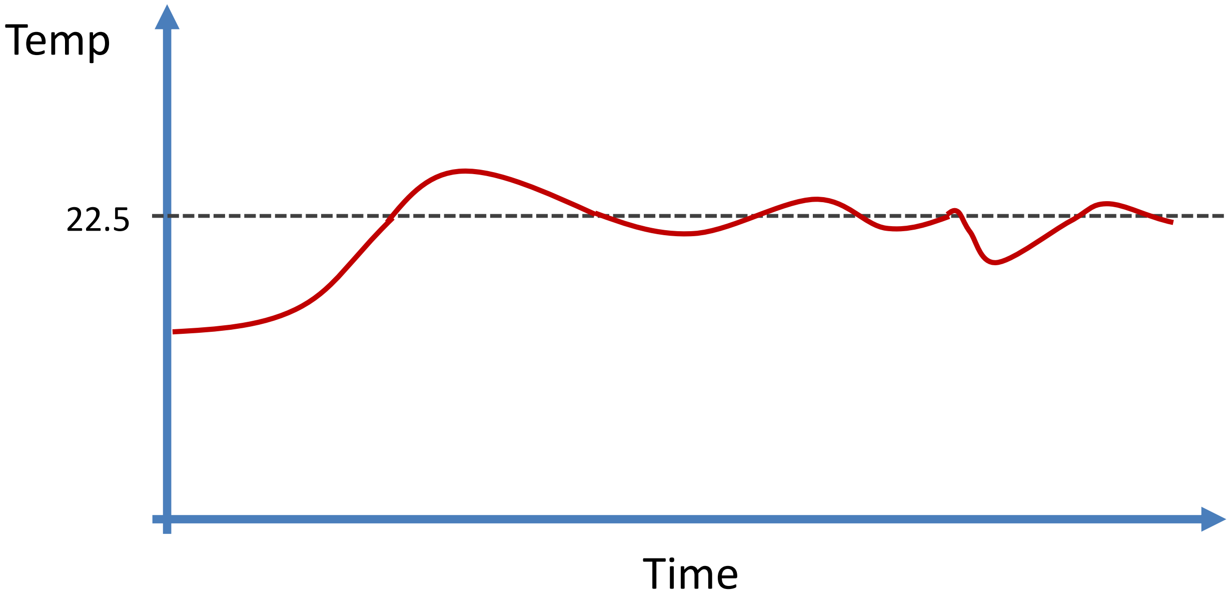
- A control loop that **watches** the shared **state** of the cluster through the API Server
- Makes changes attempting to **move the current state towards the desired state**

Logically, each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process:

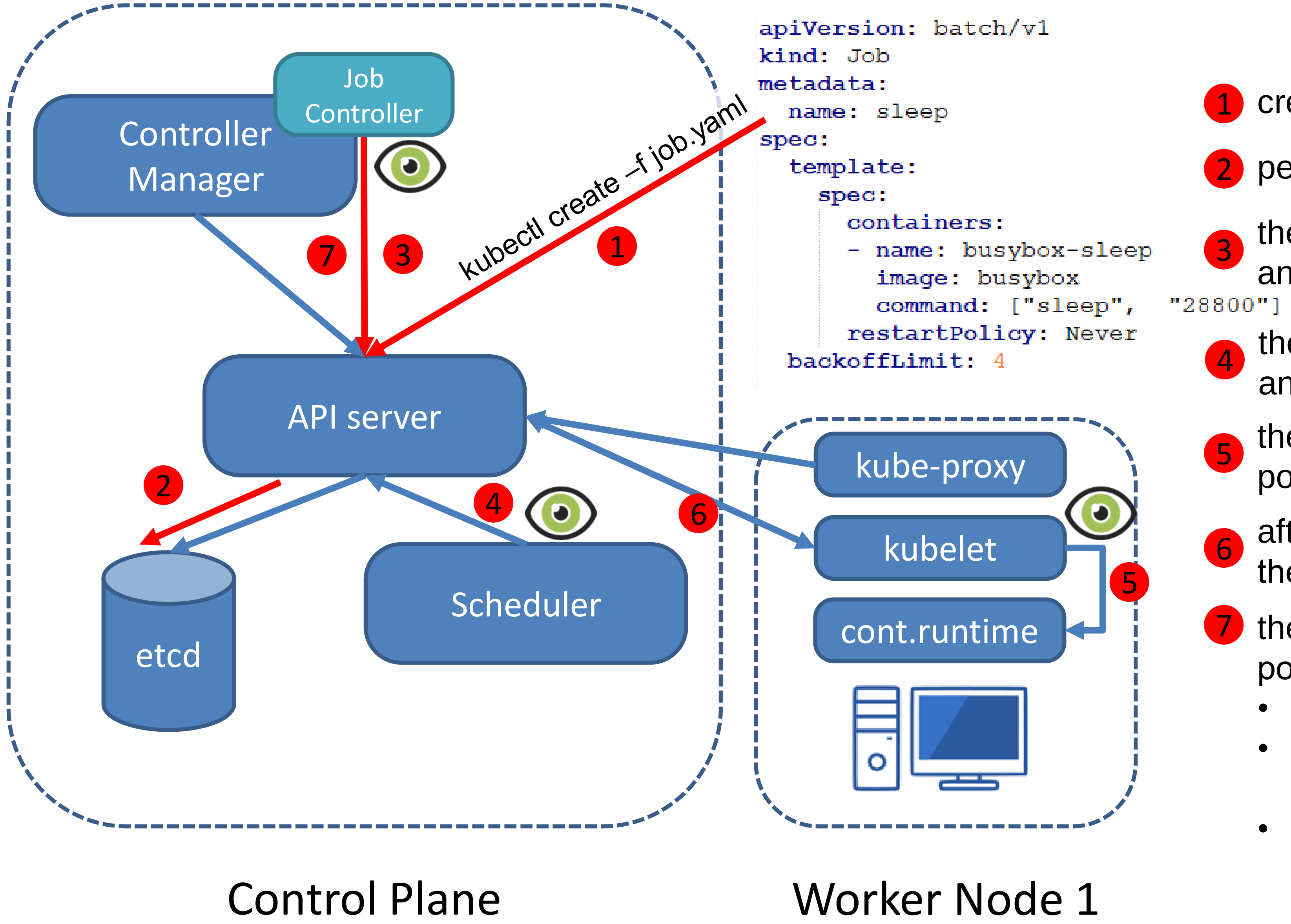
- Node controller: responsible for noticing and responding when nodes go down
- Replication controller: responsible for maintaining the correct number of pods for every replication controller object in the system
- Endpoints controller: populates the Endpoints object (that is, joins Services & Pods)

https://github.com/kubernetes/kubernetes/tree/master/pkg/controller
https://github.com/kubernetes/kube-controller-manager

Control Loops



Control Loops in Kubernetes



```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: sleep
spec:
  template:
    spec:
      containers:
      - name: busybox-sleep
        image: busybox
        command: ["sleep",  "28800"]
      restartPolicy: Never
  backoffLimit: 4
```

1 create a job via an **API call**

2 persist state to ***etcd***

3 the **job controller** sees the newly crated job and creates a pod based on the template

4 the **scheduler** sees that there is a pod without an assigned node, so it does the assignment

5 the **kubelet** on the node sees that there is a pod assigned to it, and starts it via the CRI

6 after the container exits, **kubelet** reports back the exit code to the pod’s state

7 the **job controller** notices the change in the pod’s state and makes a decision

- exit code = 0 → marks the job as **complete**
- exit code ≠ 0 and reached the backoff limit: → marks the job as **failed**
- else → increases the count and **starts a new pod** (back to step 4)

Kubernetes Controllers

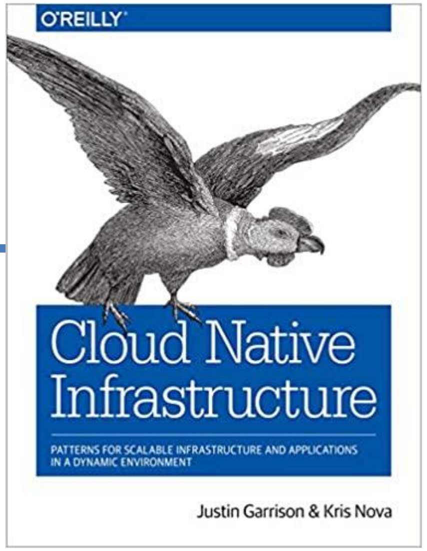
Watch the changes in the API server and compares the **desired state** with the **current state**

If these states **differ** they **carry out actions** to bring the current state closer to the desired state

working with desired states is also referred as
declarative infrastructure

acting on the deference between the desired state and the current state
is also referred as
reconcile pattern

foundation of every **cloud native infrastructure**


O'REILLY
Cloud Native Infrastructure
PATTERNS FOR SCALABLE INFRASTRUCTURE AND APPLICATIONS IN A DYNAMIC ENVIRONMENT
Justin Garrison & Kris Nova

What is the Kubelet?

Agent that runs on each node in the cluster
It makes sure that containers are running in a Pod

Uses the reconcile pattern:

- Watches the API server
- Reads the PodSpecs that are assigned to it's node
- Ensures that the containers described in those PodSpecs are running and healthy
- (Note: can also run *static Pods* whose spec comes from the local file system, not from the API server)

Only manages Kubernetes :

- You can run other (Docker) containers on the same node, Kubelet won't touch them

What are the Cloud Controllers?

Lets you link your cluster into your cloud provider's API

Embeds cloud-specific control logic:

- Runs controllers that are specific to only your cloud provider
- Examples: AWS, GCP, Azure, DigitalOcean, OpenStack, etc.

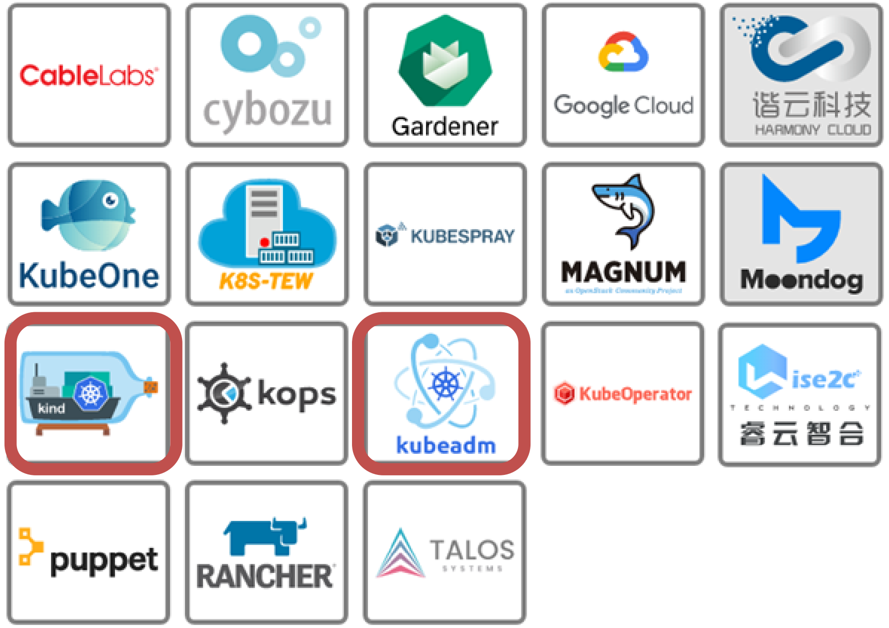
Usual jobs:

- Node controller: for checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding
- Storage controller: for setting up persistent volumes in you cloud provider
- Route controller: for setting up routes in the underlying cloud infrastructure
- Service controller: for creating, updating and deleting cloud provider load balancers

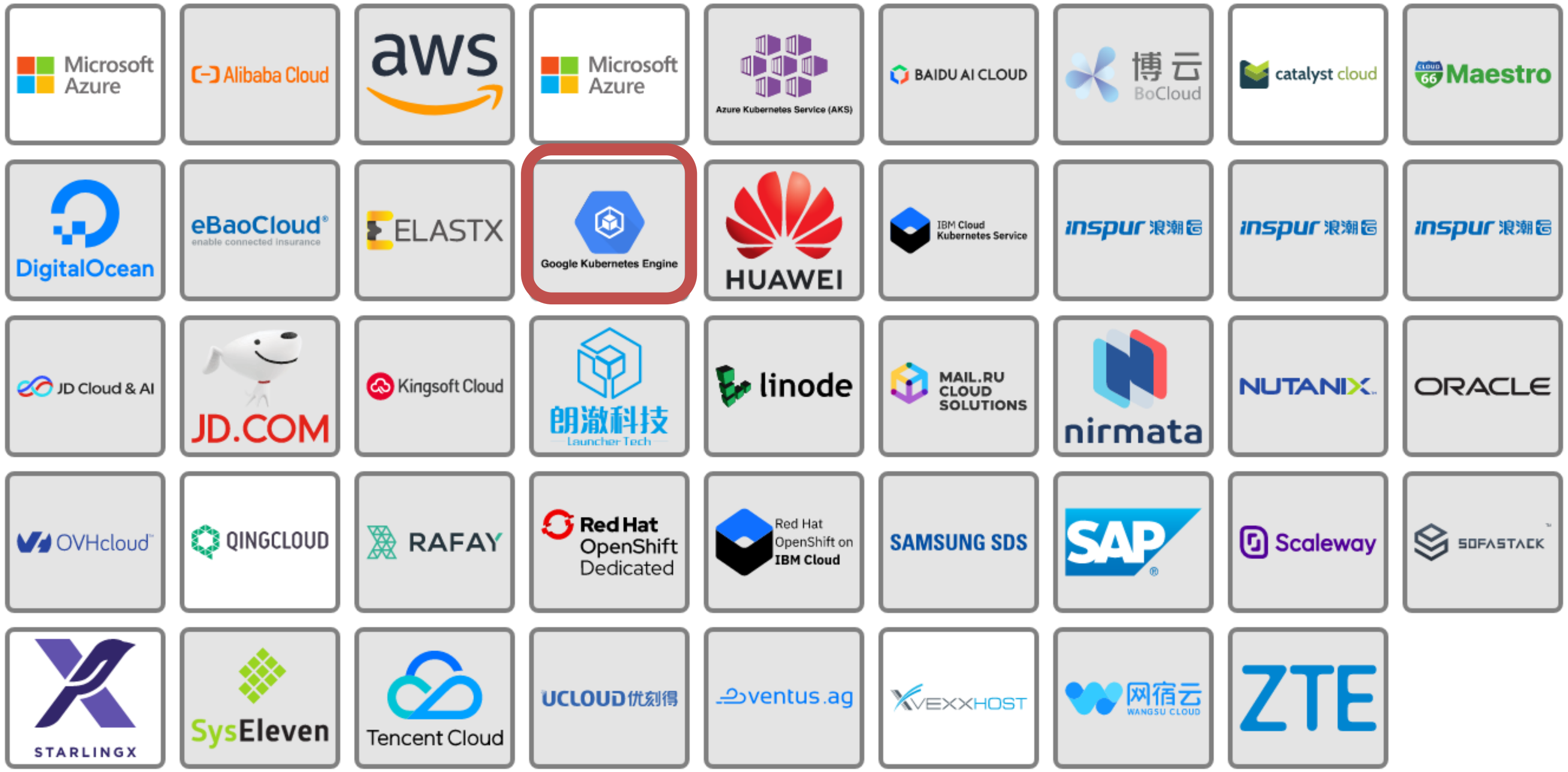
The Many Ways to Install Kubernetes

Where to Start: Installers and Hosted Kubernetes

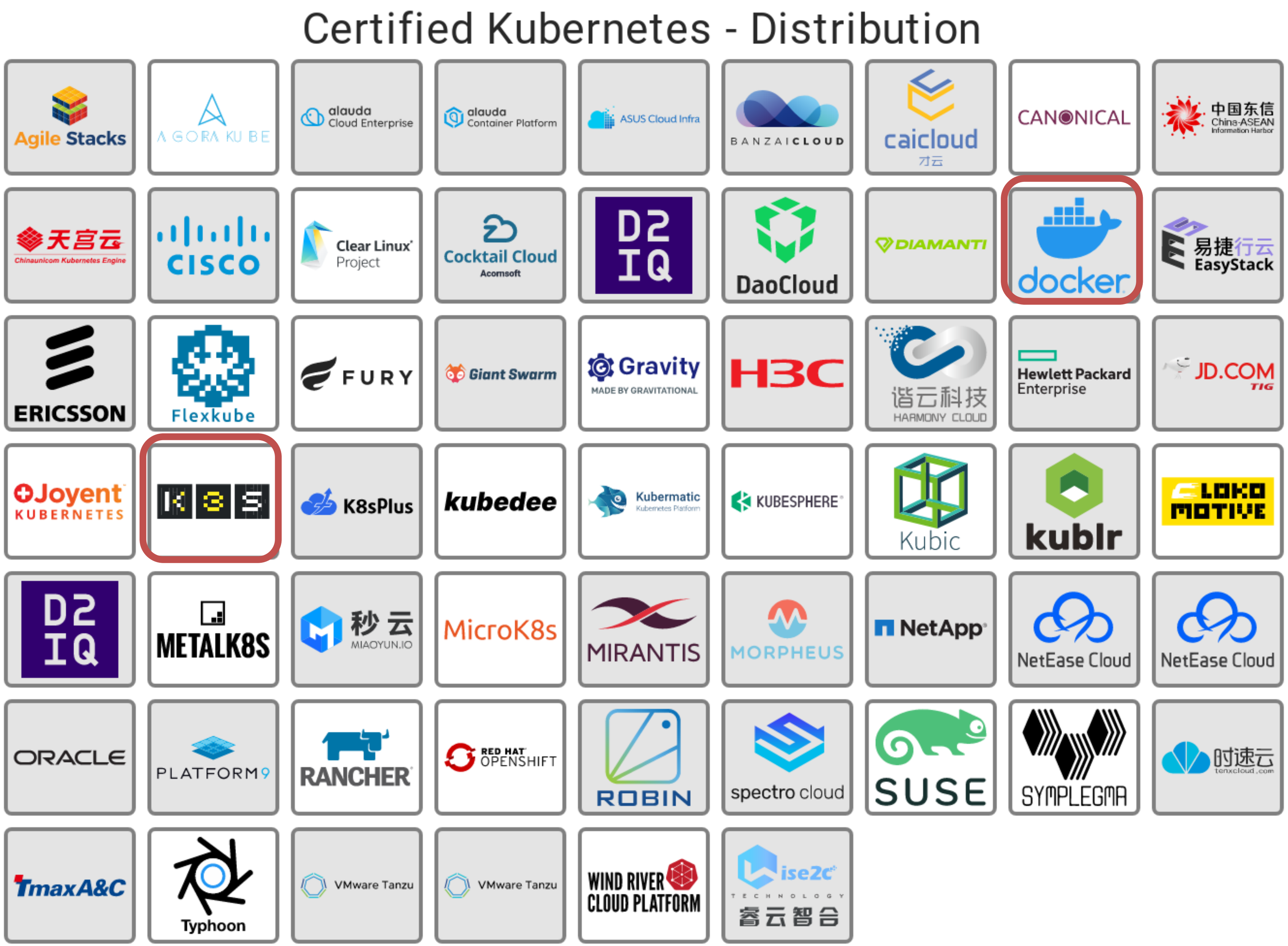
Certified Kubernetes - Installer



Certified Kubernetes - Hosted



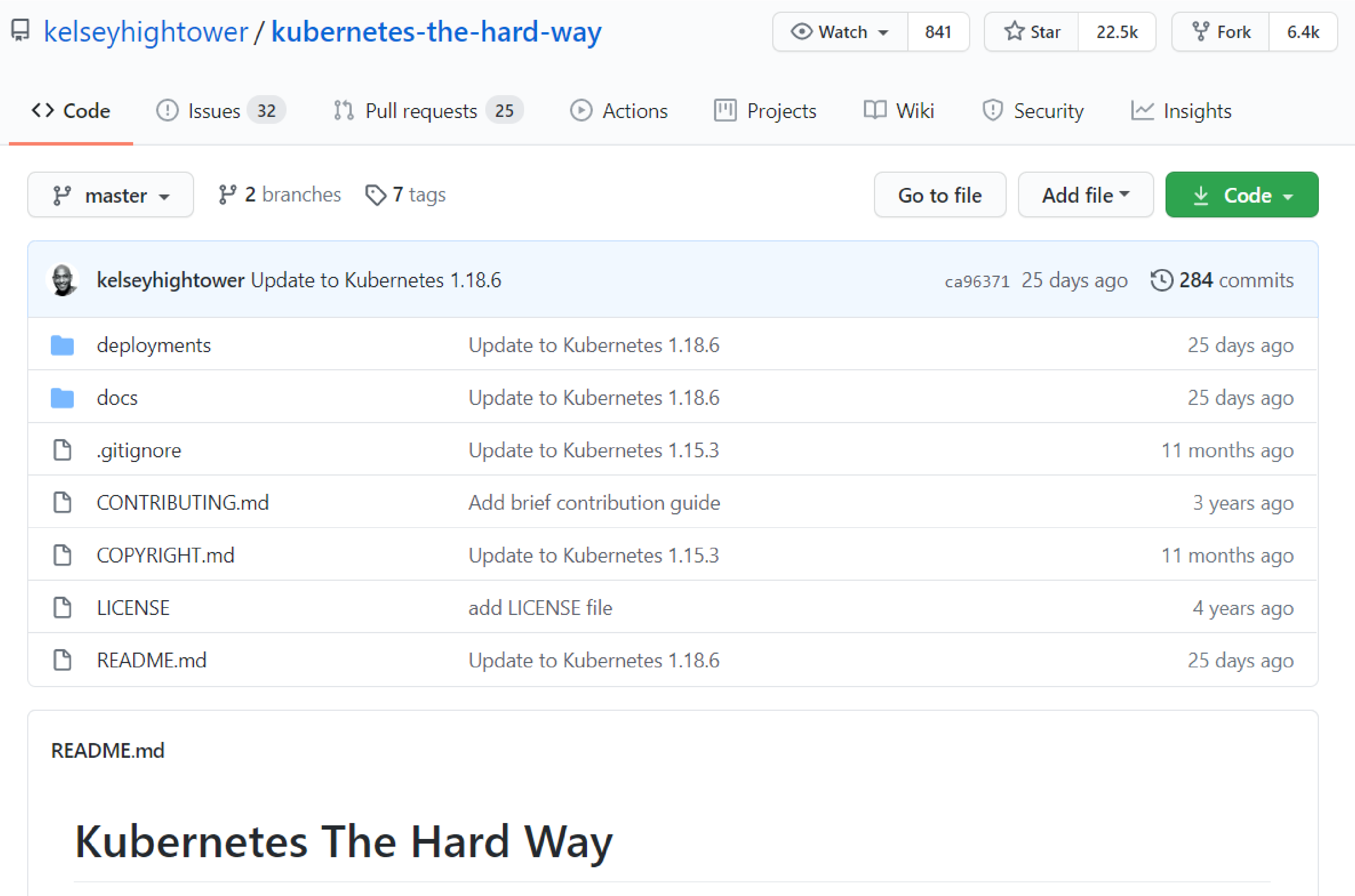
Certified Kubernetes Distributions



Certified Kubernetes - Distribution

Agile Stacks	AGORAKUBE	alauda Cloud Enterprise	alauda Container Platform	ASUS Cloud Infra	BANZAICLOUD	caicloud 才云	CANONICAL	中国东信 China-ASEAN Information Harbor
天宫云 Chinaunicom Kubernetes Engine	CISCO	Clear Linux Project	Cocktail Cloud Acornsoft	D2 IQ	DaoCloud	DIAMANTI	docker	易捷行云 EasyStack
ERICSSON	Flexkube	FURY	Giant Swarm	Gravity MADE BY GRAVITATIONAL	H3C	谐云科技 HARMONY CLOUD	Hewlett Packard Enterprise	JD.COM TIG
Joyent KUBERNETES	K3S	K8sPlus	kubedee	Kubermatic Kubernetes Platform	KUBESPHERE	Kubic	kublr	LOKOMOTIVE
D2 IQ	METALK8S	秒云 MIAOYUN.IO	MicroK8s	MIRANTIS	MORPHEUS	NetApp	NetEase Cloud	NetEase Cloud
ORACLE	PLATFORM9	RANCHER	RED HAT OPENSHIFT	ROBIN	spectro cloud	SUSE	SYMPLEGMA	时速云 tenxcloud.com
TmaxA&C	Typhoon	VMware Tanzu	VMware Tanzu	WIND RIVER CLOUD PLATFORM	ise2c TECHNOLOGY 睿云智合			

+1: Kubernetes the Hard Way



kelseyhightower Update to Kubernetes 1.18.6 ca96371 25 days ago 284 commits

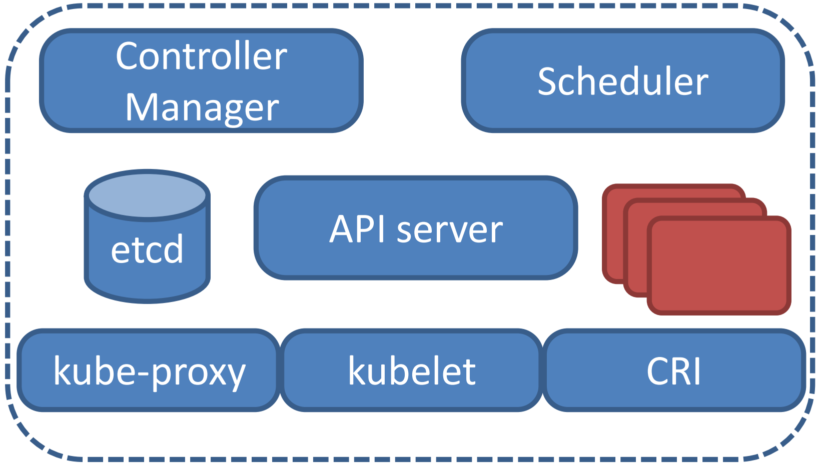
deployments	Update to Kubernetes 1.18.6	25 days ago
docs	Update to Kubernetes 1.18.6	25 days ago
.gitignore	Update to Kubernetes 1.15.3	11 months ago
CONTRIBUTING.md	Add brief contribution guide	3 years ago
COPYRIGHT.md	Update to Kubernetes 1.15.3	11 months ago
LICENSE	add LICENSE file	4 years ago
README.md	Update to Kubernetes 1.18.6	25 days ago

README.md

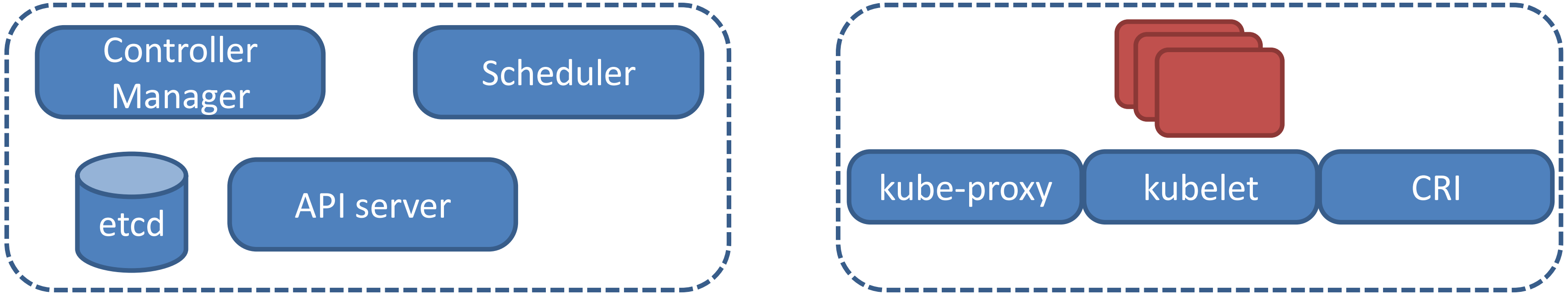
Kubernetes The Hard Way

Deployment Options – Non-HA

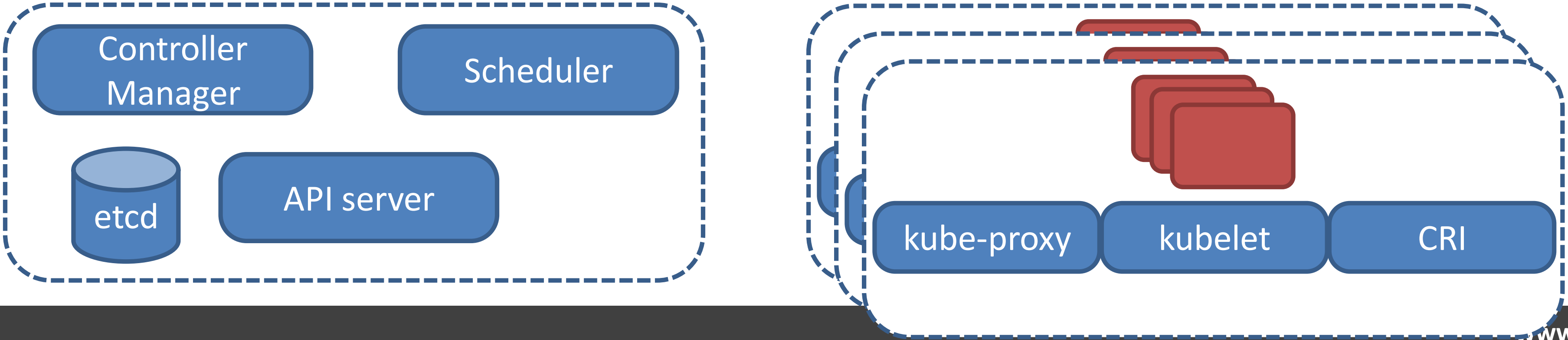
Single node



Single Master
Single Worker

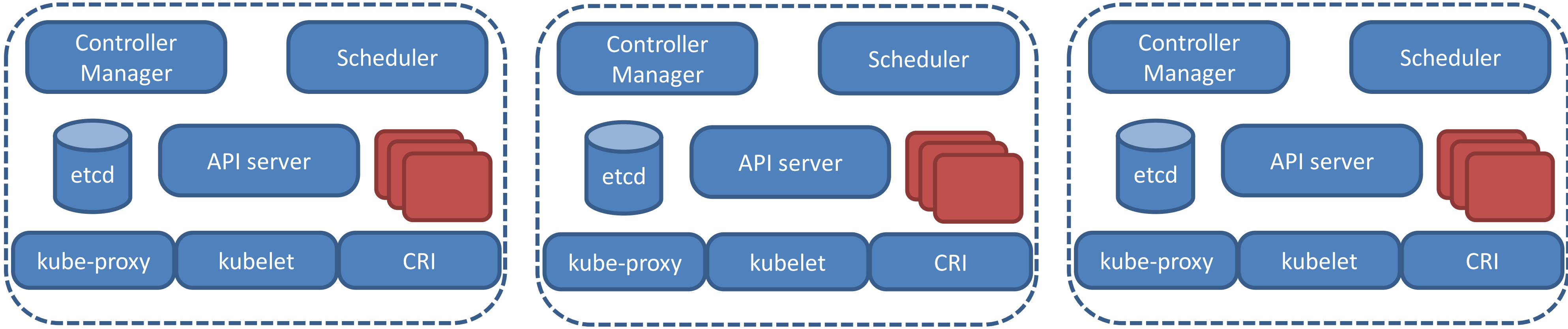


Single Master
Multiple Worker

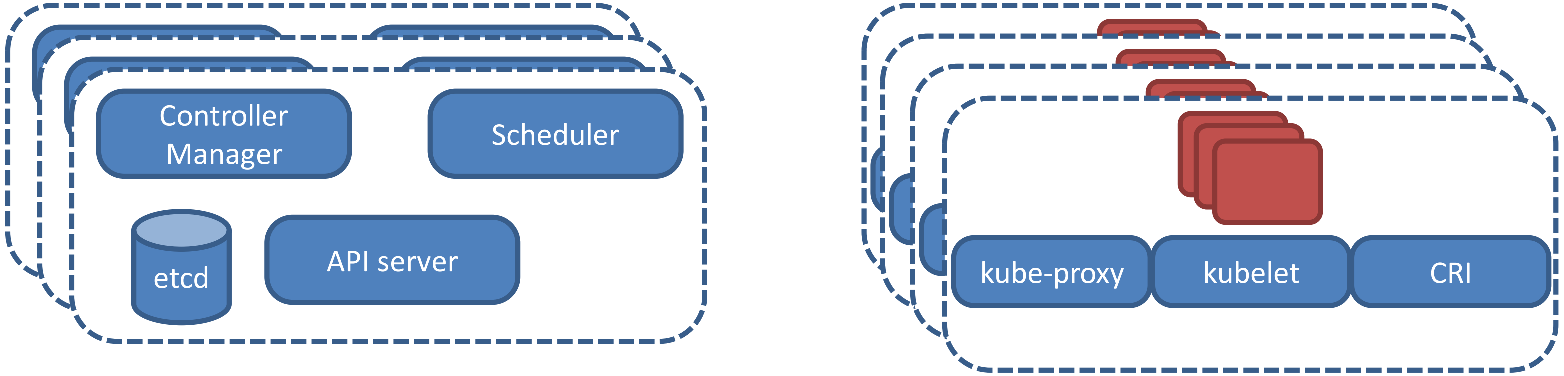


Deployment Options – HA

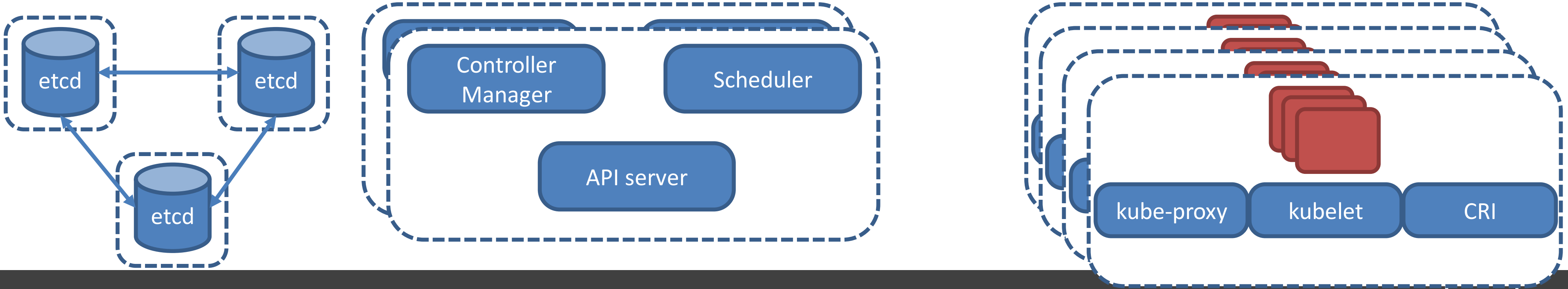
3 Node HA
Master = Worker



True HA
3 Master Node
Multiple Workers



ETCD Cluster
2+ Master Nodes
Multiple Workers



Packaging Options

Separately or together:

- You can choose to package all the control plane components together in one Go binary
  - Just like in the case of the Controller Manager, which runs multiple independent controller processes
- Good examples: HyperKube, K3s

Packaging:

- Static Go binary
- .dep / .rpm package
- Container

Scheduling:

- systemd
- Docker daemon
- Kubernetes

Packaging Options - Kubeadm

Separately or together:

- You can choose to package all the control plane components together in one Go binary
  - Just like in the case of the Controller Manager, which runs multiple independent controller processes
- Good examples: HyperKube, K3s

Packaging:

- Static Go binary
- .dep / .rpm package → kubelet
- Container → API server, etcd, controller, scheduler, kube-proxy

Scheduling:

- systemd → kubelet
- Docker daemon
- Kubernetes → API server, etcd, controller, scheduler, kube-proxy

Packaging Options – K3s

Separately or together:

- You can choose to package all the control plane components together in one Go binary
  - Just like in the case of the Controller Manager, which runs multiple independent controller processes
- Good examples: HyperKube, K3s

Packaging:

- Static Go binary -> kubelet, kubectl, API server, SQLite/etcd, controller, scheduler, kube-proxy, flannel, containerd
- .dep / .rpm package
- Container

Scheduling:

- systemd
- Docker daemon
- Kubernetes

Packaging Options - Rancher

Separately or together:

- You can choose to package all the control plane components together in one Go binary
    - Just like in the case of the Controller Manager, which runs multiple independent controller processes
- Good examples: HyperKube, K3s

Packaging:

- Static Go binary
- .dep / .rpm package
- Container

Scheduling:

- systemd
- Docker daemon
- Kubernetes

Packaging Options – Kubespray (Ansible Playbook)

Separately or together:

- You can choose to package all the control plane components together in one Go binary
  - Just like in the case of the Controller Manager, which runs multiple independent controller processes
- Good examples: HyperKube, K3s

Packaging:

- Static Go binary
- .dep / .rpm package
- Container

Scheduling:

- systemd
- Docker daemon
- Kubernetes

Versions in Kubernetes

Version numbering: **x.y.z** (see also: https://semver.org/)

- x: major version
- y: minor version
- z: patch

History:

- **1.0** came at in 10 July 2015
- The current releases is **1.19**, which come out 26 August 2020
- No future plan for **2.0**

Skew policy:

- The newest and oldest **kube-apiserver** instances must be within one minor version (e.g. 1.19 with 1.18)
- **kubelet** must not be newer than **kube-apiserver**, and may be up to two minor versions older (e.g. 1.19 → 1.19,1.18,1.17)
- **controller-manager**, **scheduler**, and **cloud-controller** must not be newer than the **kube-apiserver** instances they communicate with. They are expected to match the kube-apiserver minor version, but may be up to one minor version older (to allow live upgrades)
- **kubectl** is supported within one minor version (older or newer) of **kube-apiserver** (e.g. 1.19 → 1.20, 1.19, 1.18)

https://kubernetes.io/docs/setup/release/version-skew-policy/
https://github.com/kubernetes/community/blob/master/contributors/design-proposals/release/versioning.md

First Pod: Nginx

```
kubectl get pods

kubectl run nginx --image=nginx

kubectl get pods

kubectl get pods –o wide

curl $(kubectl get pod nginx -o jsonpath={.status.podIP})
```

Before We Really Begin: What are YAML Files??

YAML: YAML Ain't Markup Language

YAML is a **human friendly** data serialization standard for all programming languages:

- Used for expressing key-value structures
- Superset of JSON
- Commonly used for configuration files

Syntax:

- Whitespace indentation (tab not allowed)
- Comments begin with ‘***#***’
- List members are denoted by a leading hyphen (-)
- An associative array entry is represented using colon space in the form *key: value* (with one entry per line)
- Strings (scalars) are ordinarily unquoted, but may be enclosed in double-quotes ("), or single-quotes (')

```
foo: bar
list:
- member1
- member2
- member2
object:
  key1: value1
  key2: “value 2“
  key3: 123
  key4: “123“
```

First Pod Using a YAML File

Create a file called **`nginx.yaml`**:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    run: nginx
spec:
  containers:
  - image: nginx
    name: nginx
```

Then run:

```
kubectl apply –f nginx.yaml
```

Kubernetes Object Structure

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    run: nginx
spec:
  containers:
  - image: nginx
    name: nginx
```

Kubernetes Object Structure

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    run: nginx
spec:
  containers:
  - image: nginx
    name: nginx
```

imply different levels of stability and support:

- alpha
- beta
- stable

Kubernetes Object Structure

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    run: nginx
spec:
  containers:
  - image: nginx
    name: nginx
```

Resource type, e.g. Pod, Service, ReplicaSet, Deployment, Ingress, etc.

Kubernetes Object Structure

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    run: nginx
spec:
  containers:
  - image: nginx
    name: nginx
```

Information about pod that can be used for managing it

Kubernetes Object Structure

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    run: nginx
spec:
  containers:
  - image: nginx
    name: nginx
```

each object has it,
unique for that type of resource:

- only one Pod can have “name: www”
- but a Service or container can have “name: www” as well

Kubernetes Object Structure

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    run: nginx
spec:
  containers:
  - image: nginx
    name: nginx
```

Information that is specific to the given kind of resource.

Kubernetes Object Structure

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    run: nginx
spec:
  containers:
  - image: nginx
    name: nginx
```

helps you tag and then select certain resources in Kubernetes

Labels and Selectors in Kubernetes

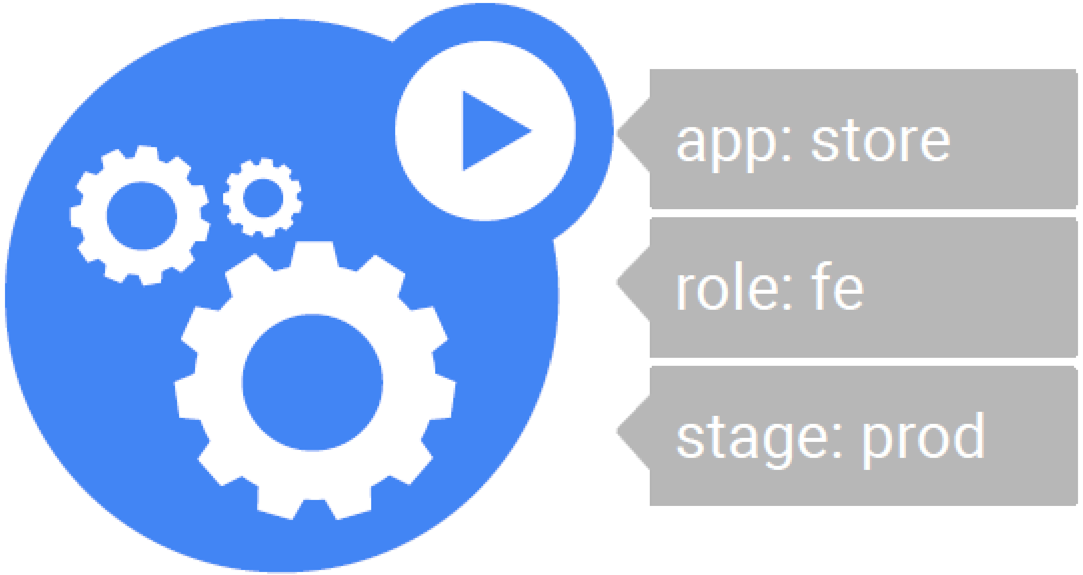
Metadata (key-value) which can be attached to any API resource

Labels: identification

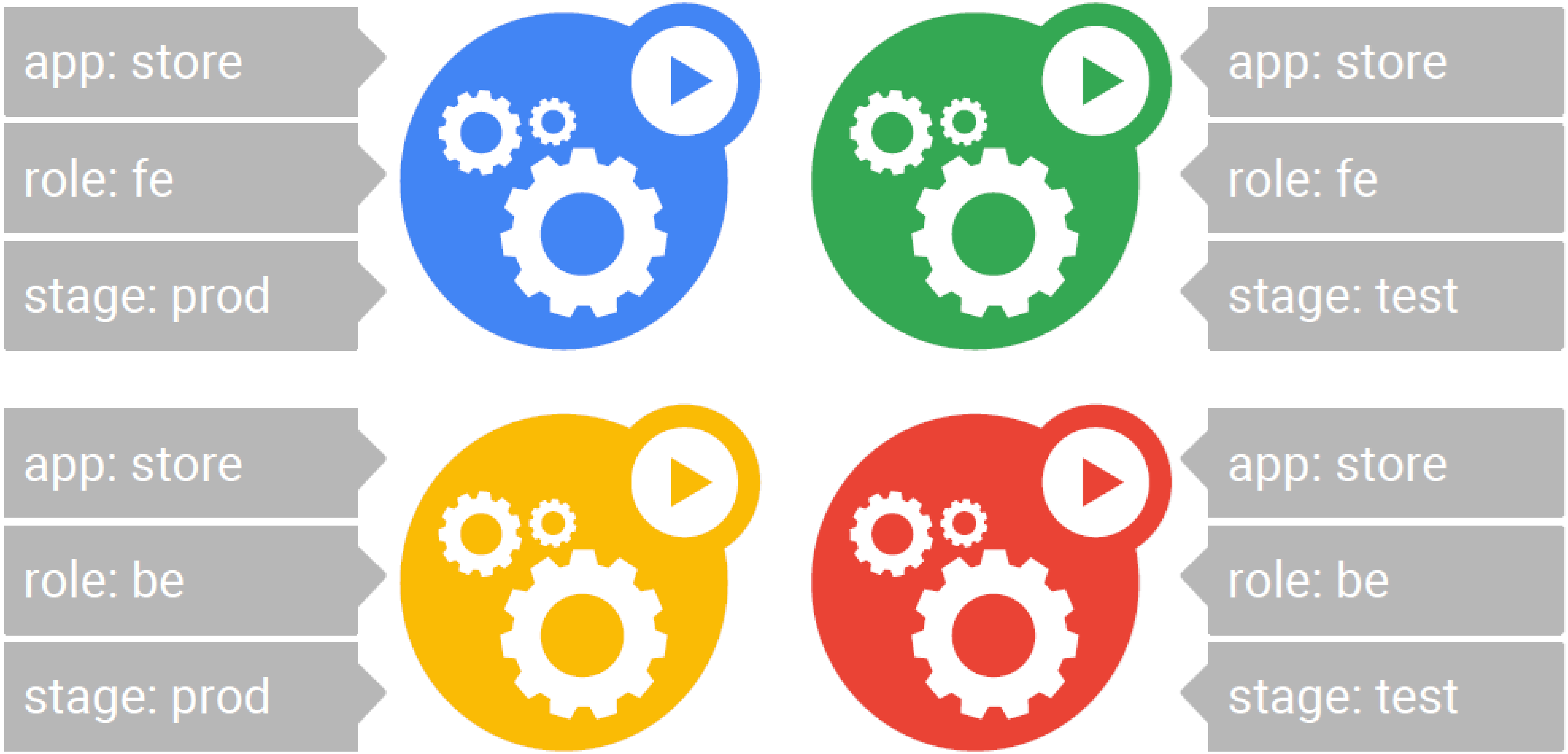
- Allow users to define how to group resources
- Examples: app name, tier (frontend/backend), stage (dev/test/prod)

Selectors: express which objects to act upon

- Think “select ... where”
- Provides very loose coupling
- Users can manage groups however they need



Example for Selectors



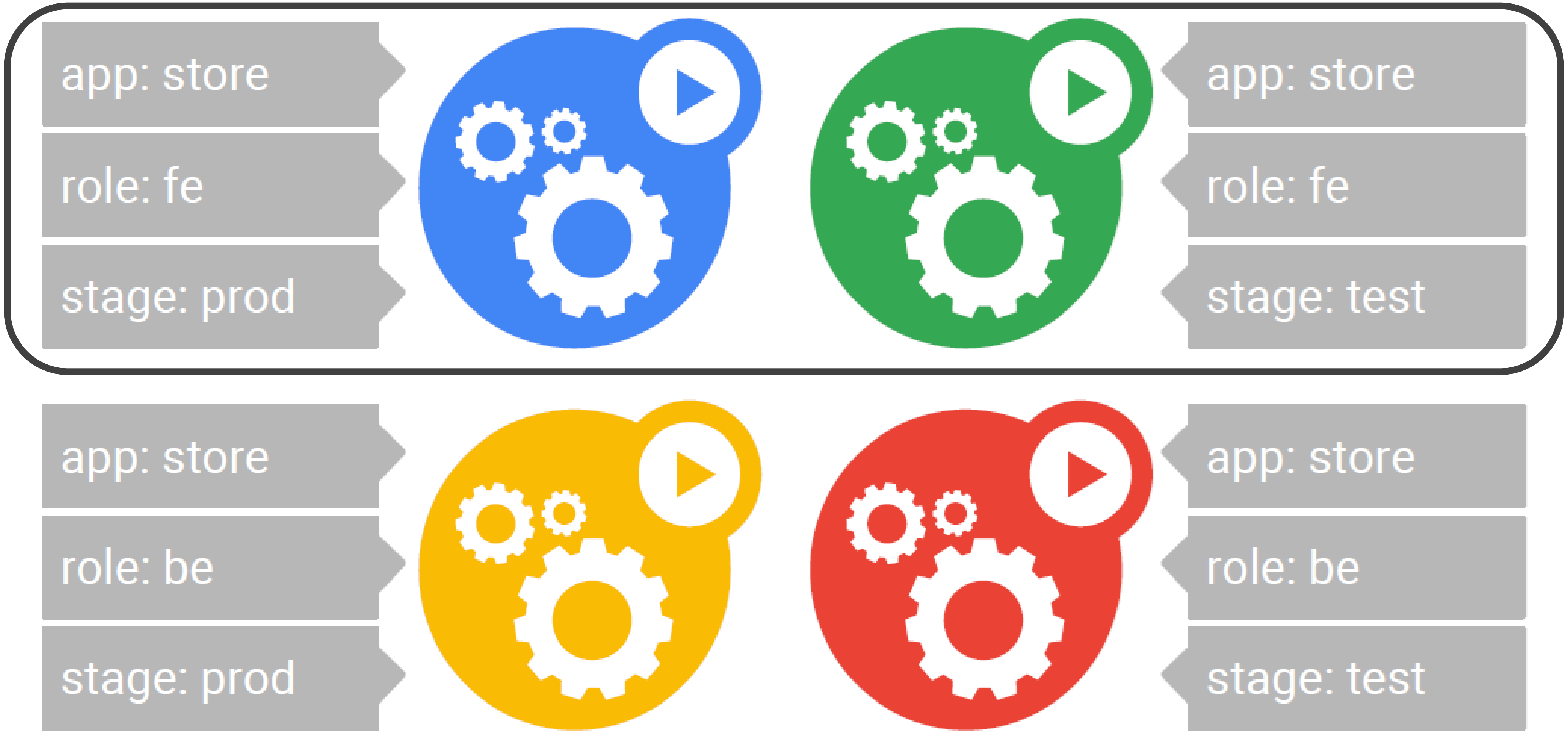
app: store
role: fe
stage: prod

app: store
role: fe
stage: test

app: store
role: be
stage: prod

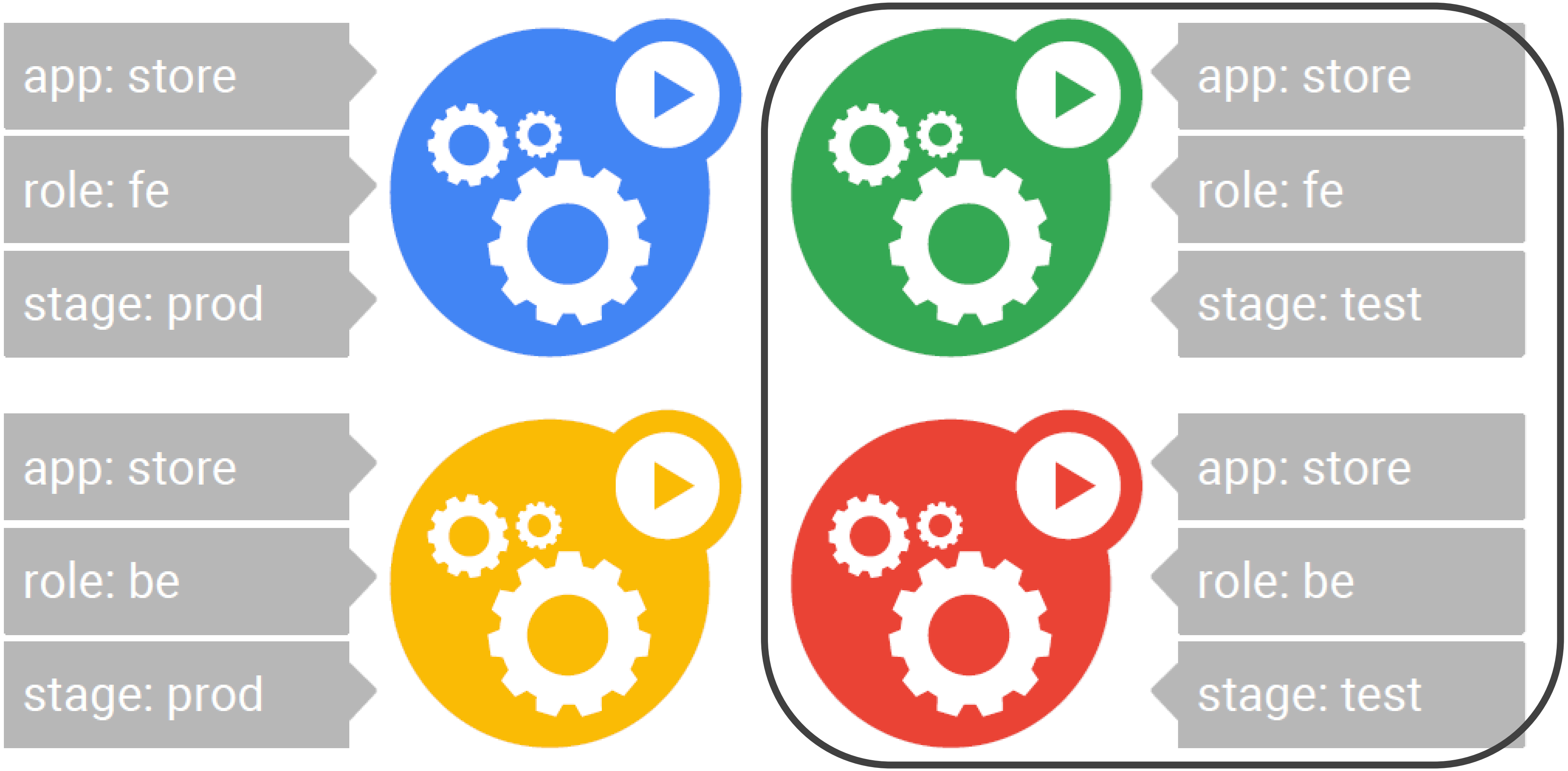
app: store
role: be
stage: test

Example for Selectors



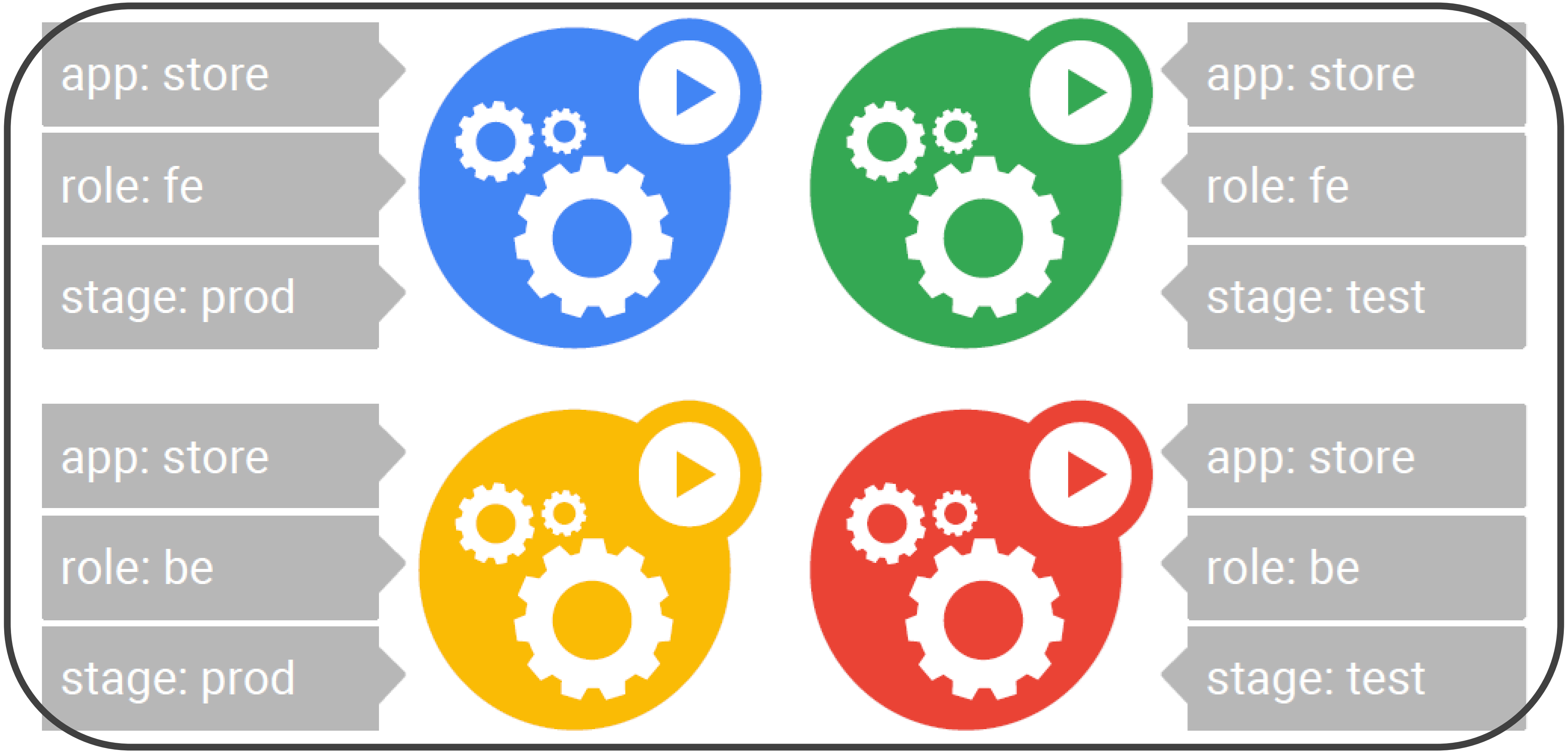
app=store, role=fe

Example for Selectors



app=store, stage=test

Example for Selectors



app=store

First Deployment

Create a file called **`nginx-deployment.yaml`**:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    run: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      run: nginx
  template:
    metadata:
      labels:
        run: nginx
    spec:
      containers:
      - image: nginx
        name: nginx
```

Then run:

```
kubectl apply -f nginx-deployment.yaml
```

First Service

Create a file called **`nginx-service.yaml`**:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  selector:
    run: nginx
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

Then run:

```
kubectl apply –f nginx-service.yaml
```

First Application: Sock Shop

```
kubectl create namespace sock-shop
```

```
kubectl apply -f https://raw.githubusercontent.com/microservices-demo/microservices-demo/master/deploy/kubernetes/complete-demo.yaml
```

will generate an error ☺

```
kubectl apply -f https://raw.githubusercontent.com/rexx4314/microservices-demo/patch-1/deploy/kubernetes/complete-demo.yaml
```

First Application: Sock Shop – Access it Via Ingress

Create a file called **`ingress.yaml`**:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: sock-shop-ingress
  namespace: sock-shop
spec:
 rules:
  - host: sock-shop.leannet.eu
    http:
      paths:
      - backend:
          service:
            name: front-end
            port:
              number: 80
```

Then run:

```
kubectl apply –f ingress.yaml
```

Outlook

CLOUD NATIVE
COMPUTING FOUNDATION

What is the Cloud Native Computing Foundation?

CNCF is a Linux Foundation umbrella project

Founded in 2015:

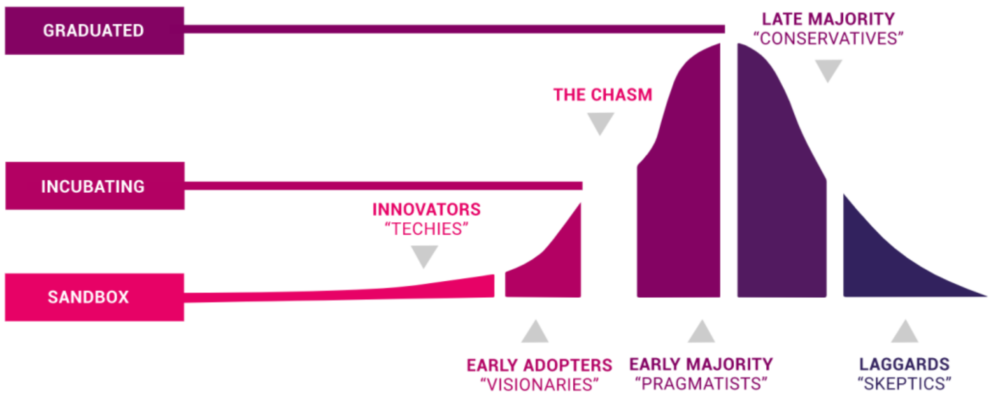
- Founded together with the announcement of Kubernetes 1.0
- Founding members: Google, CoreOS, Mesosphere, Red Hat, Twitter, Huawei, Intel, Cisco, IBM, Docker, VMware

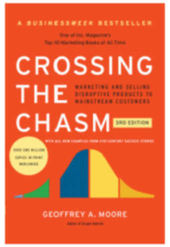
Mission:

- The Foundation's mission is to make cloud native computing ubiquitous
- Cloud native technologies empower organizations to **build and run scalable applications** in modern, dynamic environments such as **public, private, and hybrid clouds**. **Containers, service meshes, microservices, immutable infrastructure, and declarative APIs** exemplify this approach
- These techniques enable **loosely coupled systems** that are **resilient, manageable, and observable**. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil
- CNCF seeks to drive adoption of this paradigm by fostering and **sustaining an ecosystem of open source, vendor-neutral projects**, democratizing state-of-the-art patterns to make these innovations accessible for everyone

https://github.com/cncf/foundation/blob/master/charter.md

CNCF Projects





Graduated

Kubernetes
Orchestration

Prometheus
Monitoring

Envoy
Network Proxy

CoreDNS
Service Discovery

containerd
Container Runtime

Fluentd
Logging

Jaeger
Distributed Tracing

Vitess
Storage

TUF
Software Update Spec

Helm
Package Management

Harbor
Registry

Incubating

OpenTracing
Distributed Tracing API

gRPC
Remote Procedure Call

CNI
Networking API

Notary
Security

NATS
Messaging

Linkerd
Service Mesh

Rook
Storage

etcd
Key/Value Store

Open Policy Agent

CRI-O
Container Runtime

TiKV
Key/Value Store

CloudEvents
Serverless

Falco
Container Security

Argo
Continuous Integration & Deployment

Dragonfly
Image Distribution

SPIFFE
Identity Spec

SPIRE
Identity

Contour
High performance ingress controller

Telepresence
Tooling

OpenMetrics
Metrics Spec

Cortex
Monitoring

Buildpacks
Packaging Spec

Virtual Kubelet
Nodeless

KubeEdge
Edge

Brigade
Scripting

Network Service Mesh

OpenTelemetry
Telemetry Specification

OpenEBS
Storage

Thanos
Monitoring

Flux
GitOps

in-toto
Security

Strimzi
Kafka Operator

KubeVirt
VM Operator

Longhorn
Storage

ChubaoFS
Storage

KEDA
Event-driven autoscaling

Service Mesh Interface
Service Mesh

Volcano
High Performance Workloads

KUDO
Workload Orchestration

Crossplane
Infrastructure Management

CNI-Genie
Orchestrator Deployment

Keptn
Control Plane for Continuous Delivery

Cloud Custodian
Rules Engine

Dex
OpenID Connect

LitmusChaos
Chaos Engineering

Artifact Hub
Package Management Directory

Kuma
Control Plane for Service Mesh and Microservices

Parsec
Platform AbstRaction for SECurity

BFE
Load Balancing

Serverless Workflow Specification

ChaosMesh

Other Linux Foundation Umbrella Projects

ASWF /* ACADEMY SOFTWARE FOUNDATION

CLOUD NATIVE COMPUTING FOUNDATION

CD.FOUNDATION

FINOS

HYPERLEDGER

GraphQL Foundation

LF AI

LF EDGE

LF ENERGY

LF PUBLIC HEALTH

OpenJS Foundation

OPEN MAINFRAME PROJECT

presto

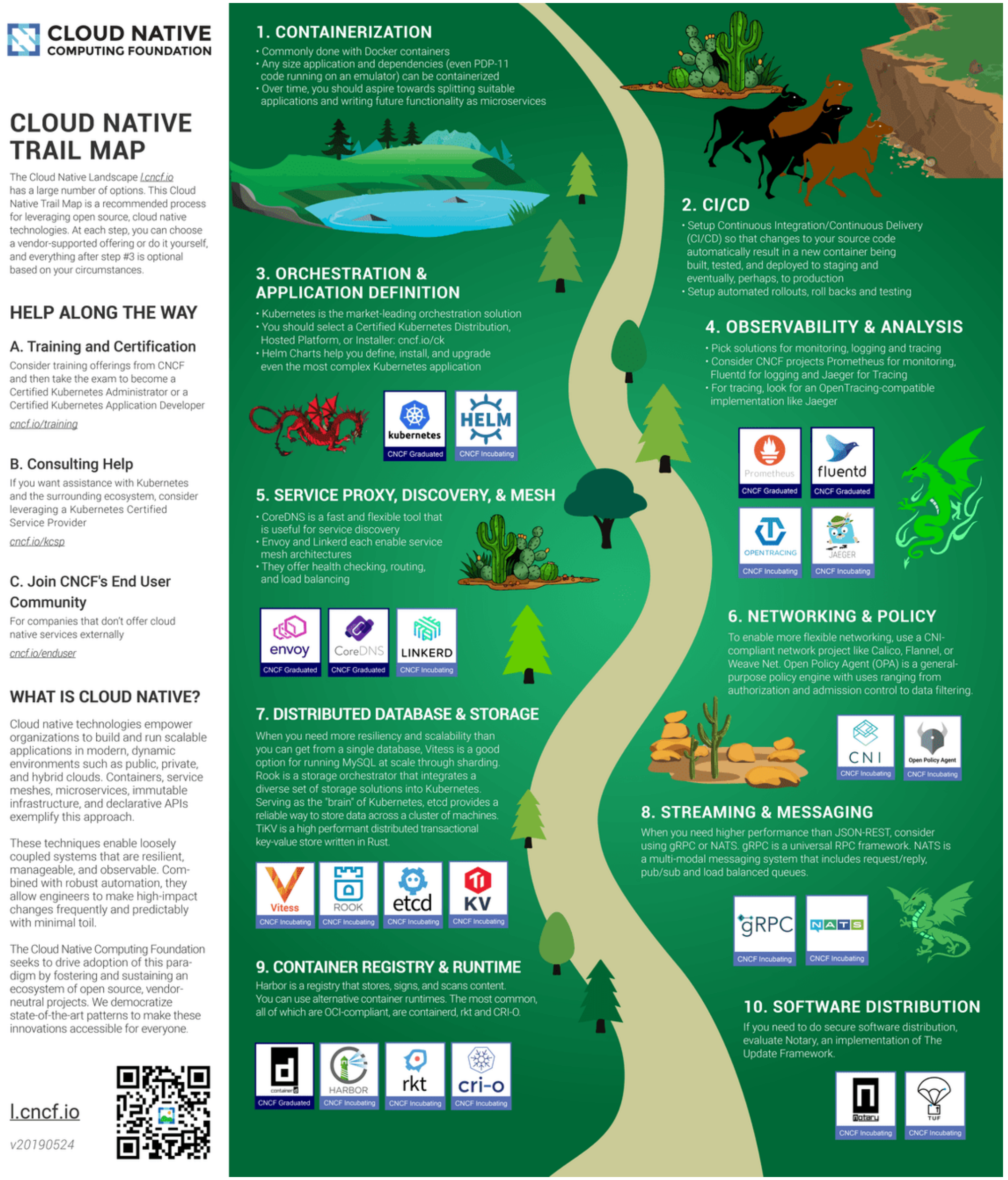
TARS Foundation

URBAN COMPUTING FOUNDATION

LF NETWORKING

OpenSSF OPEN SOURCE SECURITY FOUNDATION

Cloud Native Trail Map



CLOUD NATIVE
COMPUTING FOUNDATION

CLOUD NATIVE TRAIL MAP

The Cloud Native Landscape *l.cncf.io* has a large number of options. This Cloud Native Trail Map is a recommended process for leveraging open source, cloud native technologies. At each step, you can choose a vendor-supported offering or do it yourself, and everything after step #3 is optional based on your circumstances.

HELP ALONG THE WAY

A. Training and Certification

Consider training offerings from CNCF and then take the exam to become a Certified Kubernetes Administrator or a Certified Kubernetes Application Developer

cncf.io/training

B. Consulting Help

If you want assistance with Kubernetes and the surrounding ecosystem, consider leveraging a Kubernetes Certified Service Provider

cncf.io/kcsp

C. Join CNCF's End User Community

For companies that don't offer cloud native services externally

cncf.io/enduser

WHAT IS CLOUD NATIVE?

Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.

These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.

The Cloud Native Computing Foundation seeks to drive adoption of this paradigm by fostering and sustaining an ecosystem of open source, vendor-neutral projects. We democratize state-of-the-art patterns to make these innovations accessible for everyone.

l.cncf.io

v20190524

1. CONTAINERIZATION

- Commonly done with Docker containers
- Any size application and dependencies (even PDP-11 code running on an emulator) can be containerized
- Over time, you should aspire towards splitting suitable applications and writing future functionality as microservices

2. CI/CD

- Setup Continuous Integration/Continuous Delivery (CI/CD) so that changes to your source code automatically result in a new container being built, tested, and deployed to staging and eventually, perhaps, to production
- Setup automated rollouts, roll backs and testing

3. ORCHESTRATION & APPLICATION DEFINITION

- Kubernetes is the market-leading orchestration solution
- You should select a Certified Kubernetes Distribution, Hosted Platform, or Installer: cncf.io/ck
- Helm Charts help you define, install, and upgrade even the most complex Kubernetes application

4. OBSERVABILITY & ANALYSIS

- Pick solutions for monitoring, logging and tracing
- Consider CNCF projects Prometheus for monitoring, Fluentd for logging and Jaeger for Tracing
- For tracing, look for an OpenTracing-compatible implementation like Jaeger

5. SERVICE PROXY, DISCOVERY, & MESH

- CoreDNS is a fast and flexible tool that is useful for service discovery
- Envoy and Linkerd each enable service mesh architectures
- They offer health checking, routing, and load balancing

6. NETWORKING & POLICY

To enable more flexible networking, use a CNI-compliant network project like Calico, Flannel, or Weave Net. Open Policy Agent (OPA) is a general-purpose policy engine with uses ranging from authorization and admission control to data filtering.

7. DISTRIBUTED DATABASE & STORAGE

When you need more resiliency and scalability than you can get from a single database, Vitess is a good option for running MySQL at scale through sharding. Rook is a storage orchestrator that integrates a diverse set of storage solutions into Kubernetes. Serving as the "brain" of Kubernetes, etcd provides a reliable way to store data across a cluster of machines. TiKV is a high performant distributed transactional key-value store written in Rust.

8. STREAMING & MESSAGING

When you need higher performance than JSON-REST, consider using gRPC or NATS. gRPC is a universal RPC framework. NATS is a multi-modal messaging system that includes request/reply, pub/sub and load balanced queues.

9. CONTAINER REGISTRY & RUNTIME

Harbor is a registry that stores, signs, and scans content. You can use alternative container runtimes. The most common, all of which are OCI-compliant, are containerd, rkt and CRI-O.

10. SOFTWARE DISTRIBUTION

If you need to do secure software distribution, evaluate Notary, an implementation of The Update Framework.

Okay, But Should I Use Kubernetes???

I’m certain that you will use it at some point in the future!

Not a good reason:

- I’ve heard it’s cool since Google made this!
- Want to be cloud native so I must use this new stuff
- My boss told me…

But definitely use it if:

- If you already using containers and CI/CD in your DevOps processes
- Your application needs more resilience and scalability
- You have micro(ish)services environment, written in multiple languages, that needs more than one server
- You want build cloud native application that are cloud agnostic (e.g. can run in every cloud, even in on-prem)
- You need to apply more advance deployment patterns (e.g. canary, blue/green), since you current one is too slow