kubernetes

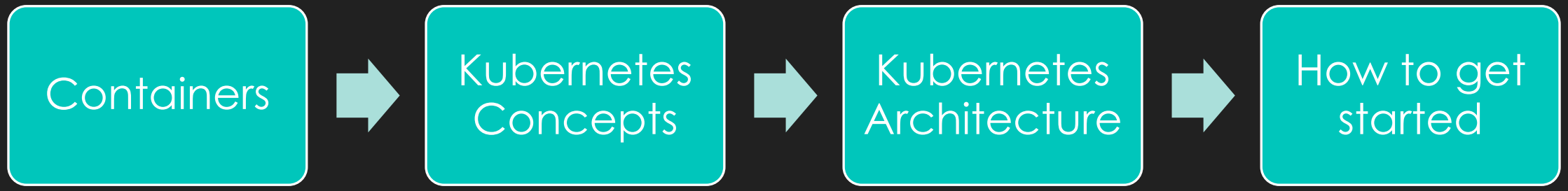
Kubernetes Explained

Garrett Woodworth

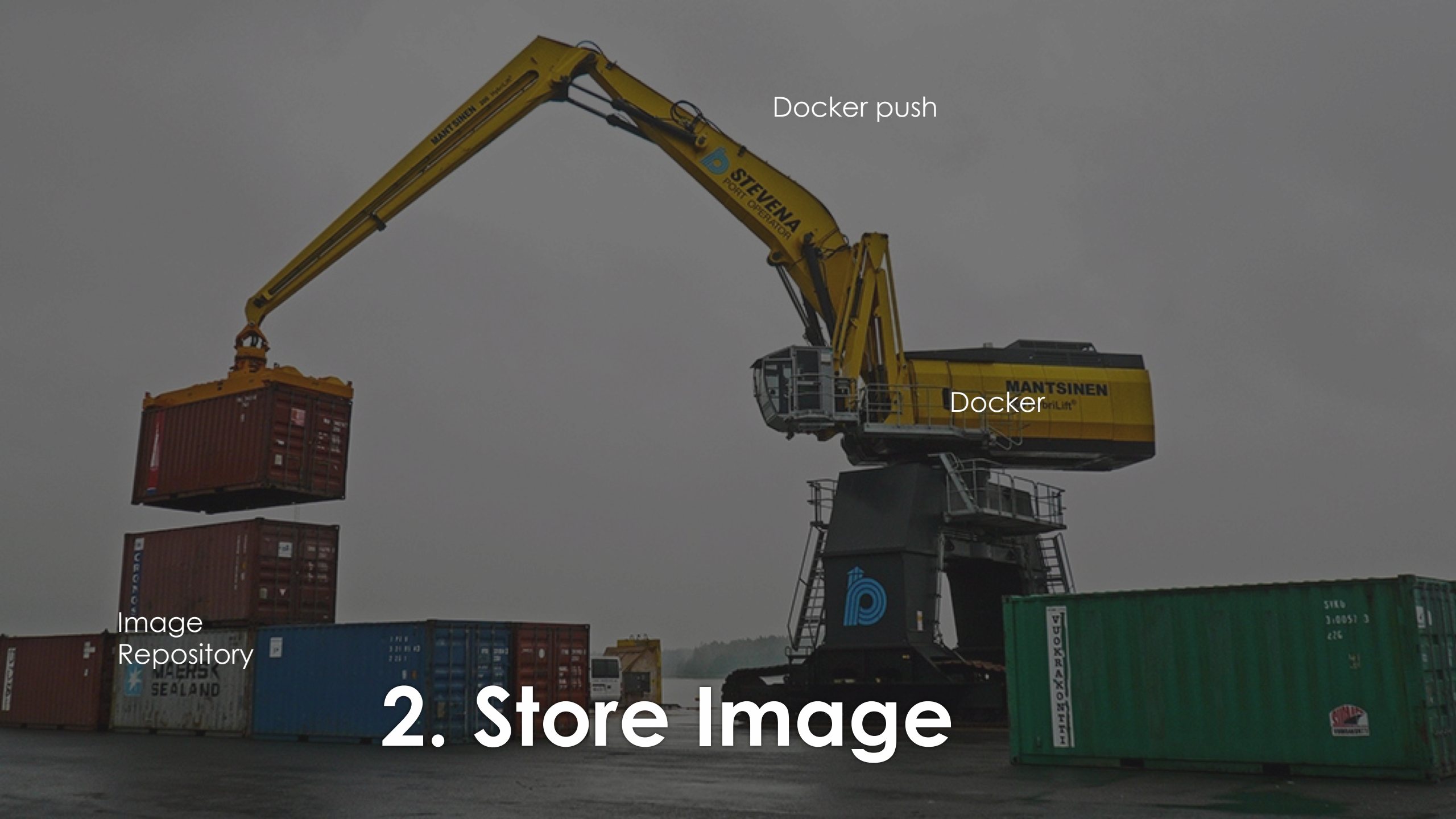
IBM Z Technical Specialist

garrett.lee.woodworth@ibm.com

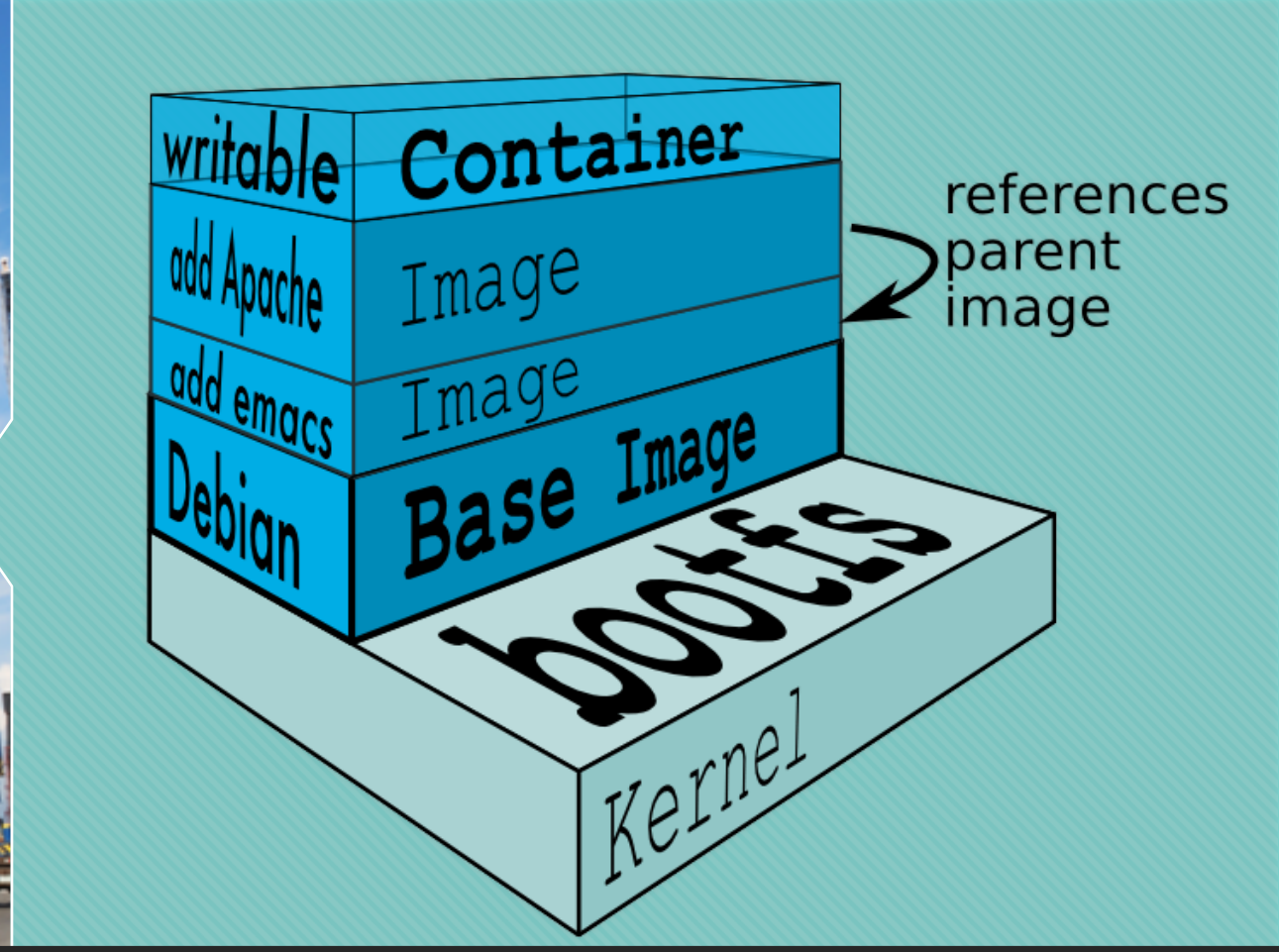
Roadmap


Containers
Kubernetes Concepts
Kubernetes Architecture
How to get started

1. Build Image



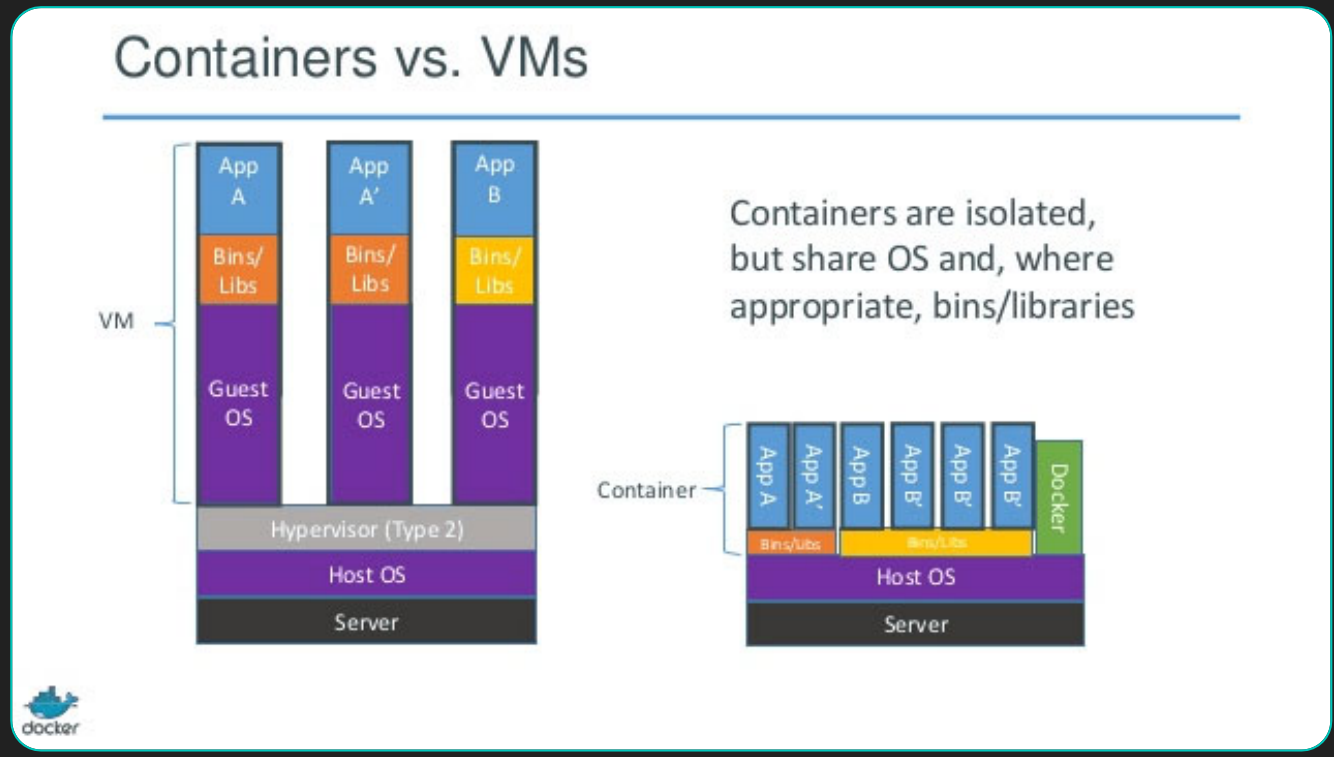
2. Store Image



3. Run Image as Container on Host Machine

Docker Use

- **Portable**, independently packaged images for apps/services which can be used across Linux distros
- **Lightweight** Namespace Isolation and cgroup resource limits for rapid deployment
- **Storage pooling** of host os and applicable bins/libraries
- Manifest lists -> support for multiple architectures (i.e. s390x, power, x86) [up to developers to enable for a specific container]



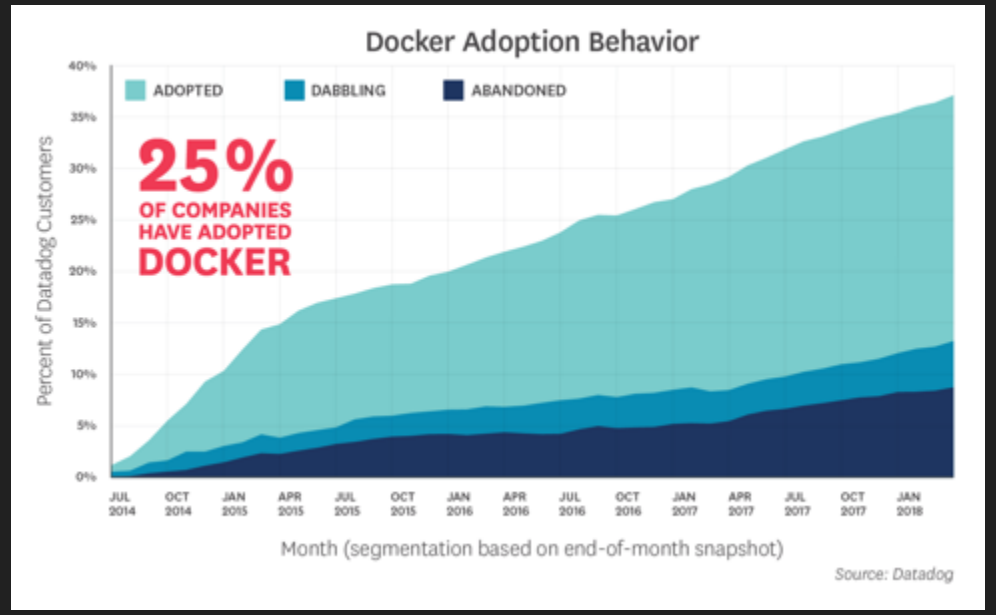
Why Orchestration (short)

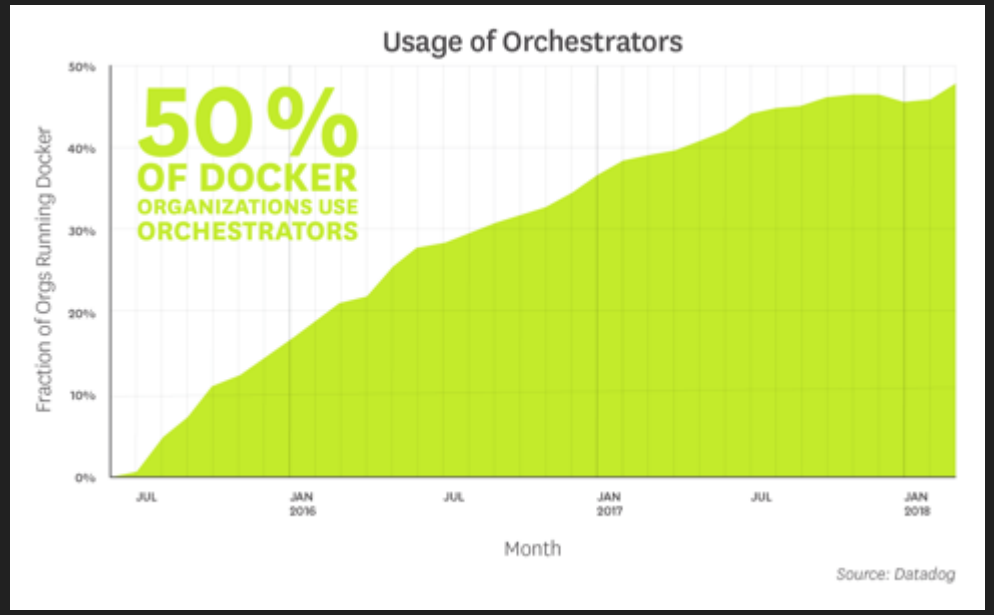
Why Orchestration (long) ?

Scaling Out Containers (especially for microservices) leads to management issues

- What happens when a container dies? (recovery)
- How do I rollout new versions of my application?
- How do I expose containers to the outside world (port conflicts become a problem if using one host)?
- How do I scale my application and load-balance calls to it?
- How do I secure access to my containers?
- How do I manage credentials for my applications?
- How can I manage my containers across nodes (machines / vms in my datacenter or cloud) from one control plane to better utilize resources (resource pooling -> improved scheduling -> improved utilization)?

....



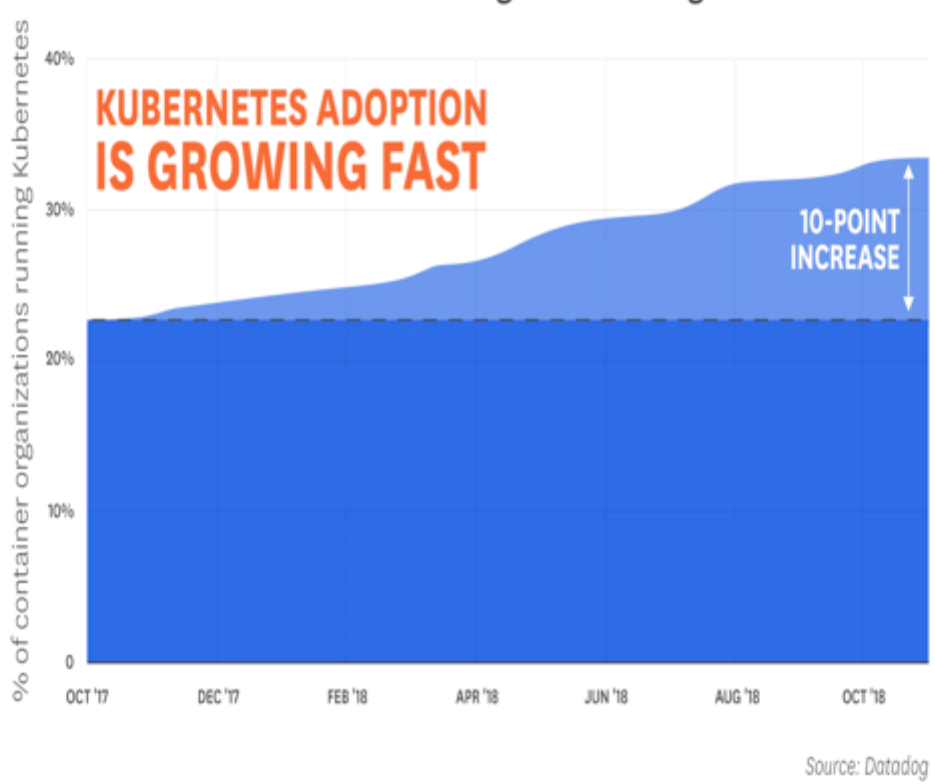


Docker Adoption and Container Orchestration

- Over time companies are realizing the need for container orchestration from the beginning

Why Kubernetes?

- Open source
- High community involvement
- Good starting point (Many tutorials, start with docker)
- Availability (Cloud Private, IKS, EKS, GKS, AKS, OpenShift, etc.)
- Standard (Automation)



#	Company	Commits to Docker by Company Commits
	*independent	20521
1	Docker	17068
2	Red Hat	2252
3	dotCloud	2115
4	IBM	1247
5	Huawei	1235
6	Microsoft	1226
7	ZTE Corporation	503
8	DaoCloud	431
9	Google	410

Source: Stackalytics

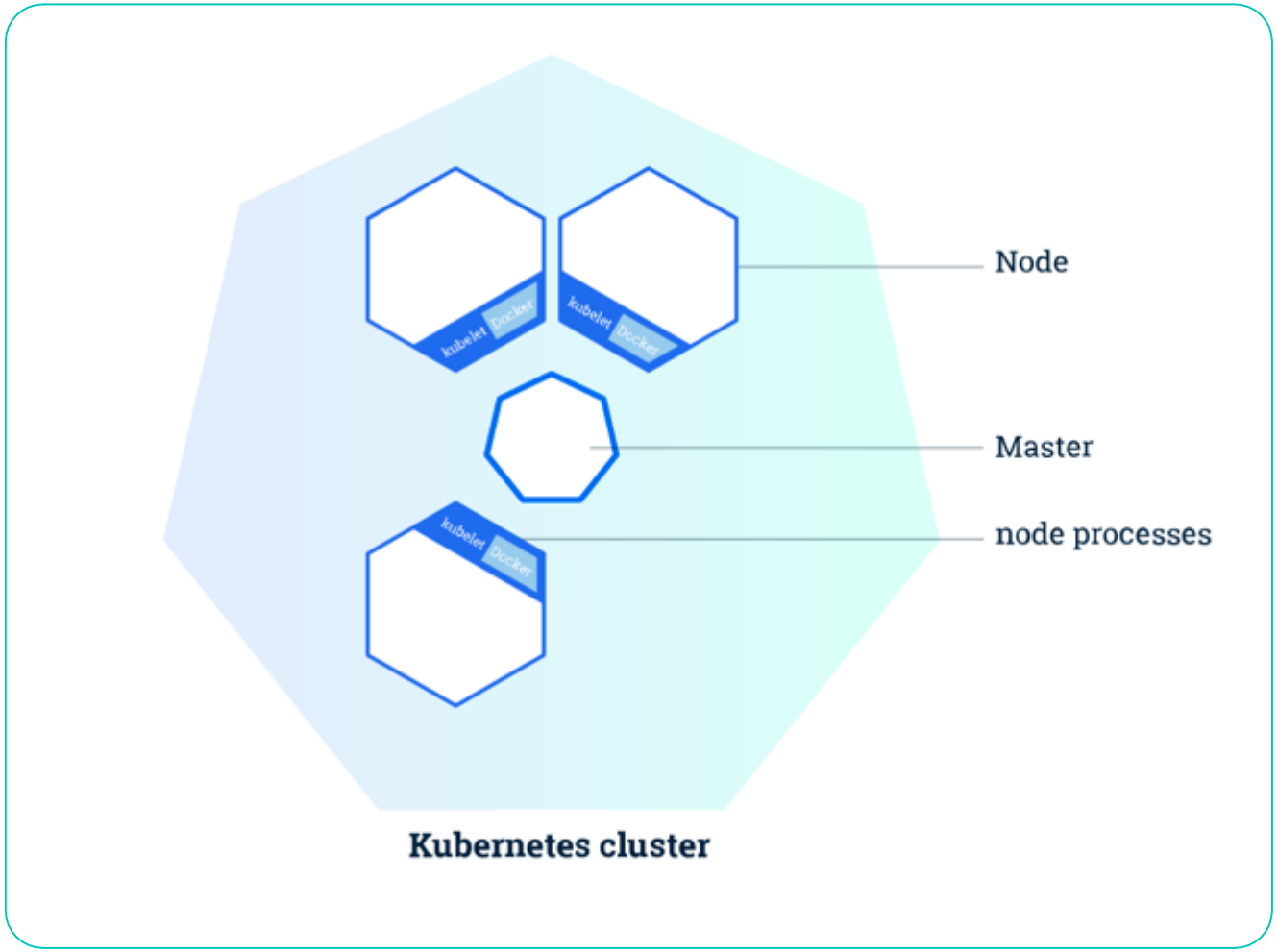
#	Company	Commits to Kubernetes by Company	Commits
1	Google		19627
2	Red Hat		7806
	*independent		6929
3	Huawei		1658
4	ZTE Corporation		1133
5	Microsoft		705
6	FathomDB		587
7	CoreOS		505
8	VMware		500
9	Fujitsu		482
10	IBM		465

Source: Stackalytics

Whose Behind these Open Source Projects?

Cluster (Simple)

- A group of masters and nodes working together to form one addressable unit
- Masters run system containers (containers to maintain the system)
- Nodes (workers) handle the application containers that run containerized workloads (containers scheduled by the platform) as well as some Kubernetes system containers specific for nodes

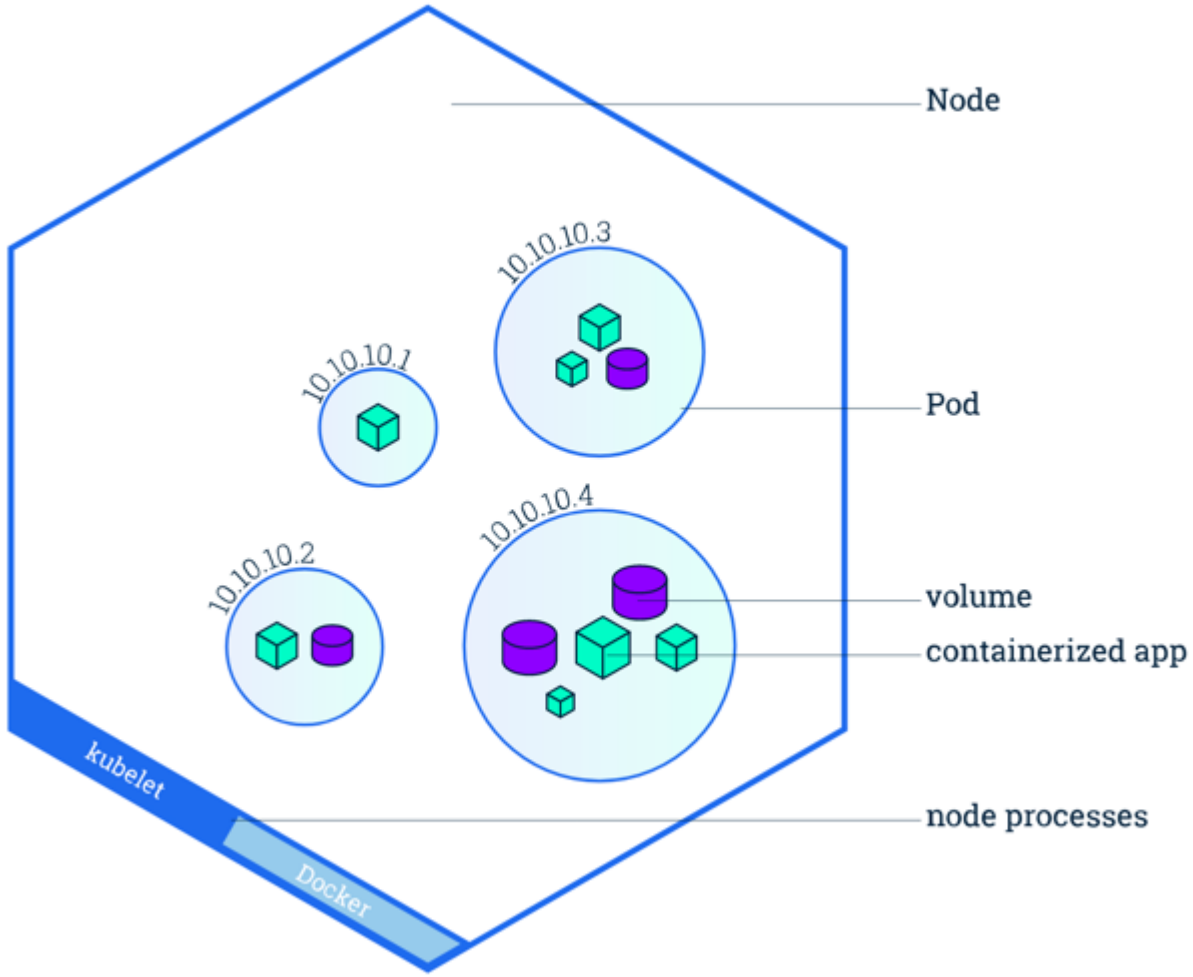


Kubernetes Namespaces (not Linux namespaces)

- Virtual clusters all connected to the same "physical" cluster
- End user sees the cluster via current namespace (i.e. kubectl get pods will show pods in current namespace)
  - Divide cluster resources between different users
  - Resources (i.e. pods, replicasets, etc.) scoped by namespace (resource names unique within namespace)
- Initial Kubernetes Namespaces
  1. default: objects with no defined namespace
  2. kube-system: objects created by Kubernetes system (i.e. helm-api, tiller, logging, kube-dns, etc.)
  3. kube-public: originally configured as readable by all users (even unauthenticated users) to make certain resources visible cluster-wide
- Make and view namespaces
  - *kubectl create namespace hi* [makes new namespace called hi]
  - *kubectl get namespace* [returns all namespaces on the "physical" cluster]

Kubernetes Resources

- Kubernetes is a declarative system: you declare how you want the system to be (spec = desired state) and Kubernetes monitors it to keep it that way (status = current state)
- Kubernetes specifically attempts to reconcile the desired state with the current stat
- Each object also has an api field and all actions go through the kubernetes api server
- You can use kubectl to perform actions which become wrapped api calls
- Decoupling resources is an emphasis
- In order to see what a specific resource is and what field it takes you can always use (kubectl explain resource) such as (kubectl explain node) to get a description of it and find out its api version and what fields it takes

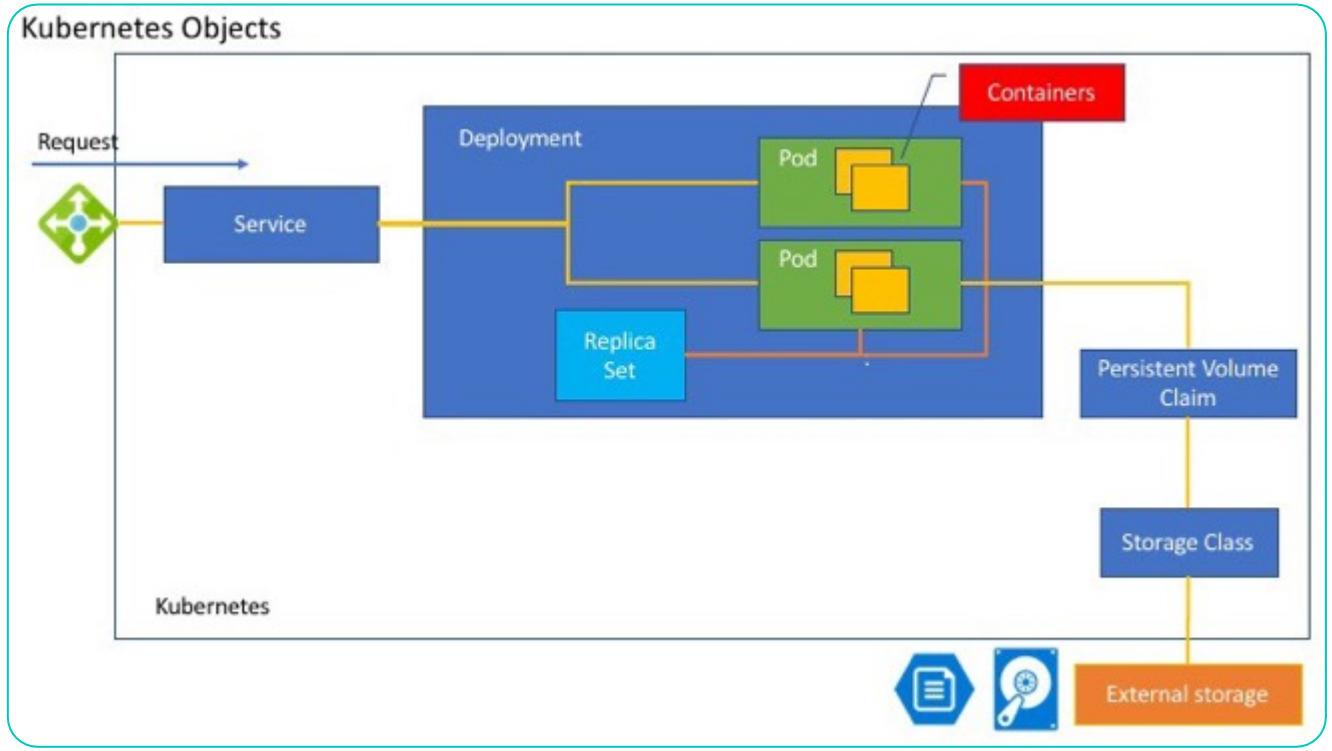


Nodes

- Workers in the cluster where application containers are scheduled and run

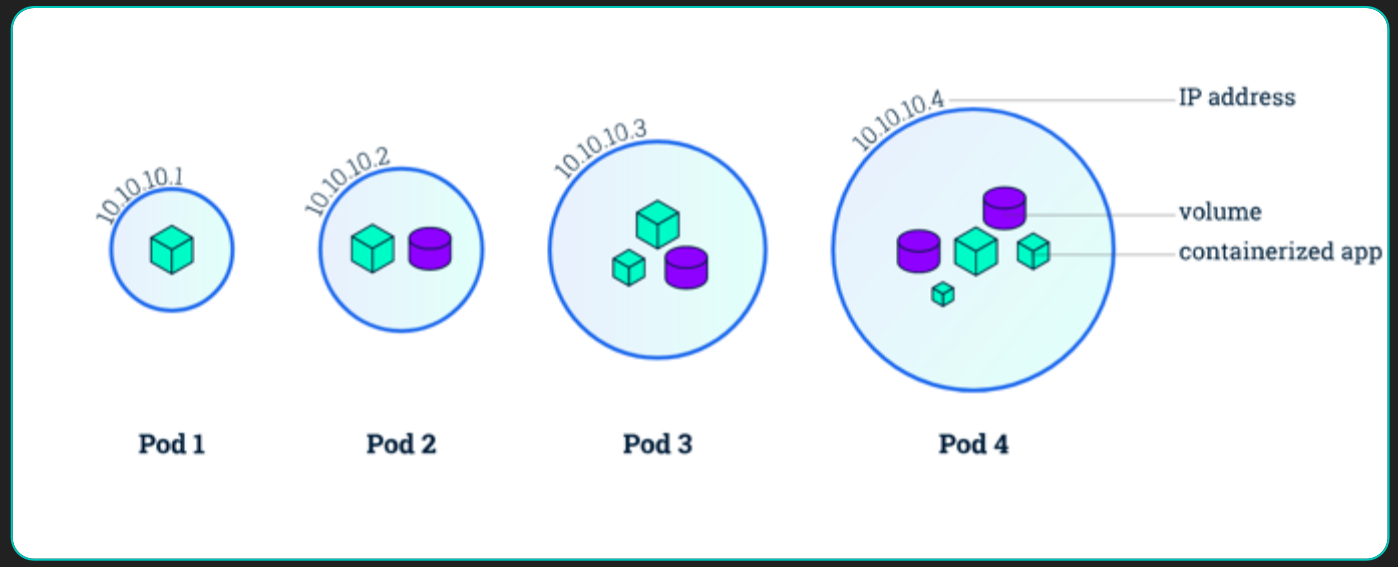
Simplified Big Picture

- Pod – Set of containers running in same execution environment/context (smallest unit in kubernetes) [containers in pod share some Linux namespaces (Network, IPC, and PID if enabled) but each have own cgroup]
- ReplicaSet – makes sure correct number and types of pods are available
- Deployment –Manages replica sets for ease of new app version rollout.
- Service – Provides access point for pods/deployment as well as load balancing
- Persistent Volume Claim – provides storage volumes to container runtime (i.e. docker) by binding to persistent volumes
- Storage Class – groups storage so that it can be dynamically selected and provisioned
- Persistent Volume - Set of external storage defined to kubernetes



Pods

- 1+ containers running in same execution context [containers in pod share some Linux namespaces (Network, IPC, and PID if enabled) and storage volumes but each have own cgroup]
- smallest unit in kubernetes
- Each have unique ip
- Replaceable (if they die they are recreated by their replicaset)



Replica Sets

- How many copies of a pod do I want?
- Makes sure the proper number of copies exist by checking the desired state (x number of pods with the current state) [self-healing]
- Best practice means creating replicaset with a deployment

Deployments

- Manages Replicasets
- Versioned Rollouts and Rollbacks
- User creates and updates state declaration for pods and replicasets via deployment
- Add parameters for scheduling such as resource requests

Deployment Example

Create deployment from file (deployment.yaml) with:

kubectl apply –f deployment.yaml

```yaml
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
```

Daemonsets

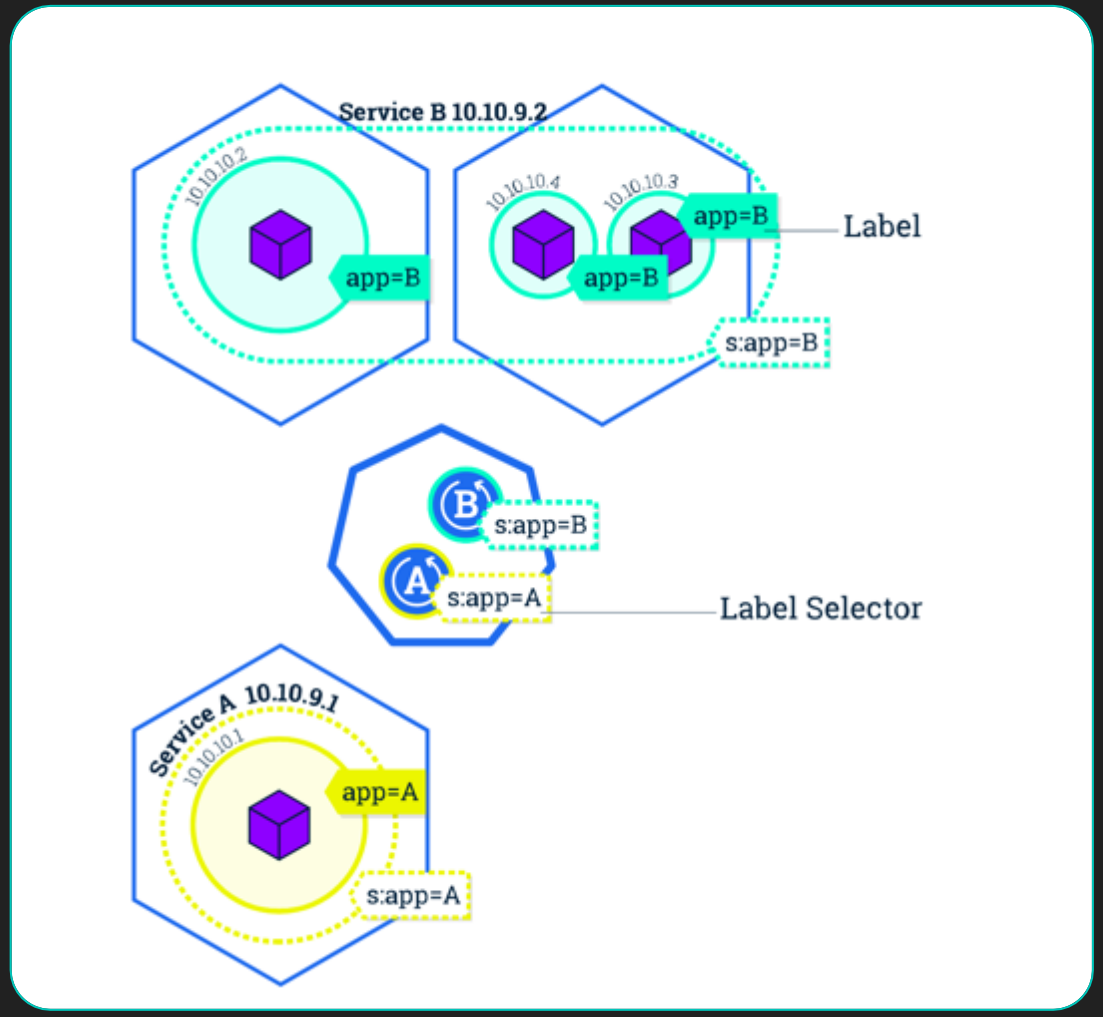
- One pod on every node in host group [specified by labels] (Things like logging controllers, monitoring, etc.)
- As nodes are added pods are added to them by daemonset via control loop

Jobs

- Run until completion (Cronjobs can run at specific times) [One-shot]
- Can think of as batch workloads

Services

- Serves as long-lasting frontend for backend pods
- Provides level 4 (TCP/IP) load balancing to pods based on number of connections (TCP/IP level)



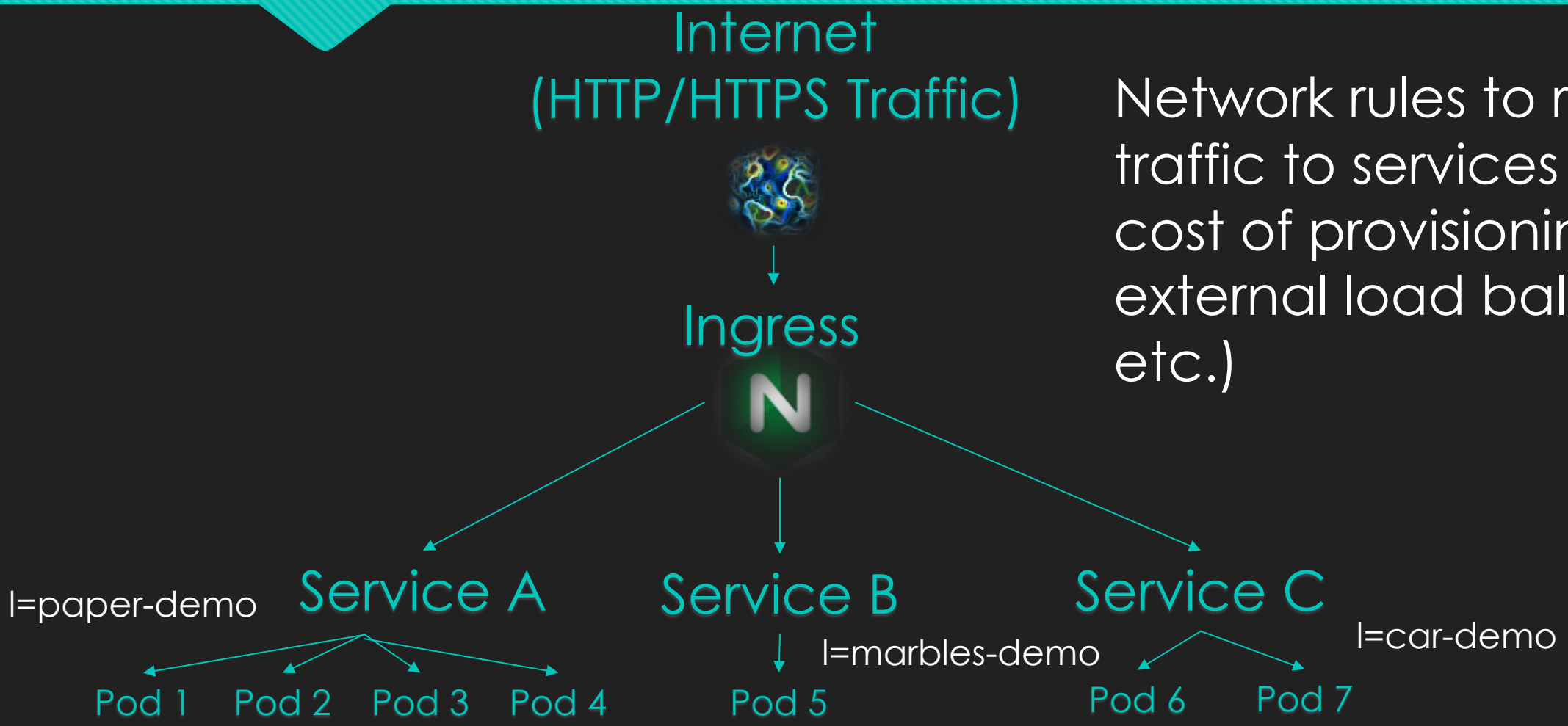
Services Example

Create service from file (service..yaml) with:

kubectl apply –f service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

Ingress



Network rules to route traffic to services (saves cost of provisioning external load balancers, etc.)

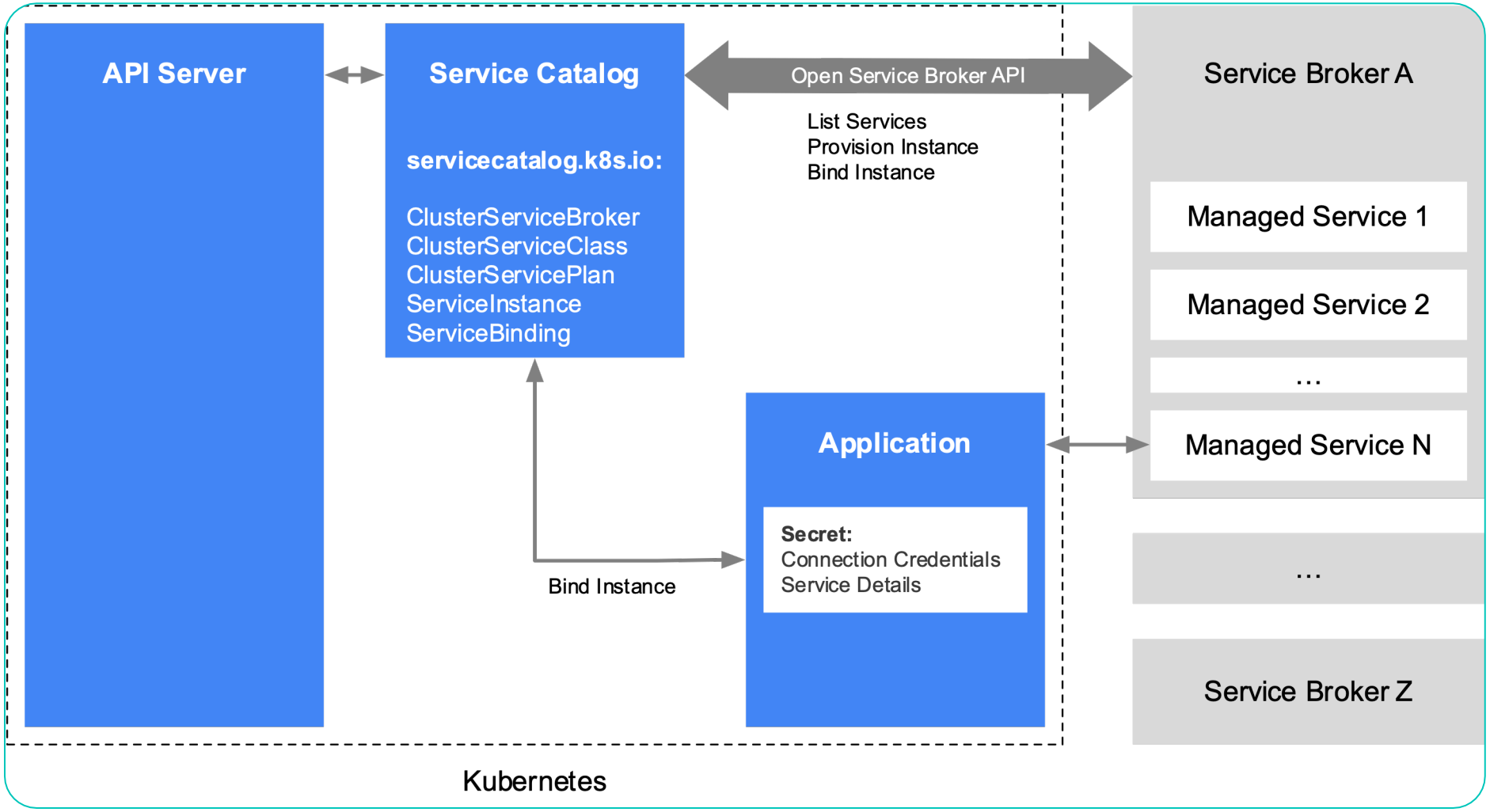
Ingress Example

Create ingress from file (ingress.yaml) with:

kubectl apply –f ingress.yaml

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: simple-fanout-example
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        backend:
          serviceName: service1
          servicePort: 4200
      - path: /bar
        backend:
          serviceName: service2
          servicePort: 8080
```

- Service catalog installs the service catalog.k8s.io API and provides resources listed under service catalog.k8s.io
- ClusterServiceBroker connect to provisioning service
- ClusterServiceClass = the list of services provided



Service Catalog

- Catalog of services provided by Service Brokers (contact point for set of services managed by third party)

Operators: Controlling your cluster from the inside

Write code to extend Kubernetes to automate tasks

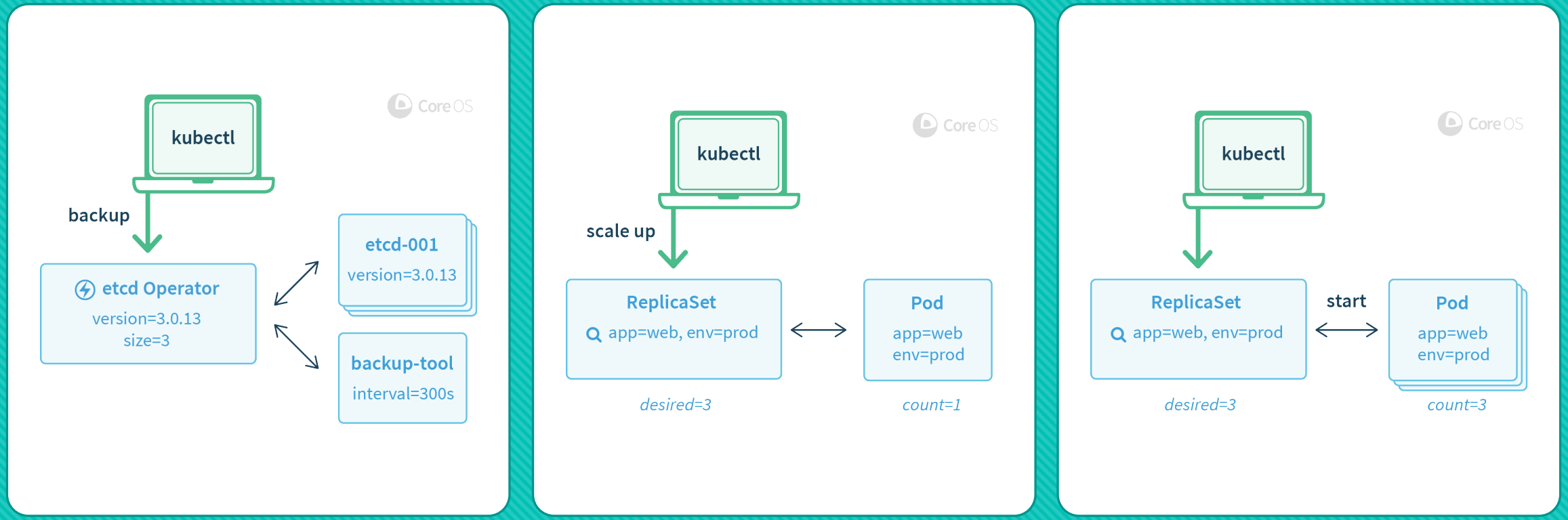
Uses CRDs (Custom Resource Definitions) to define application resources

Popular applications made and deployed via operators for best-practices deployment

Ease of sharing via operator hub

Operator Principles

- Application's operational knowledge -> configuration resource + control loop
- Install controller with deployment or stateful set
- Create CRD (Custom Resource Definition) to use for resource
- Use k8s resources instead of re-inventing the wheel
- Backwards compatible
- Applications should run independently of operator's existence
- Versioning
- Chaos monkey testing for failures
- See Introducing Operators



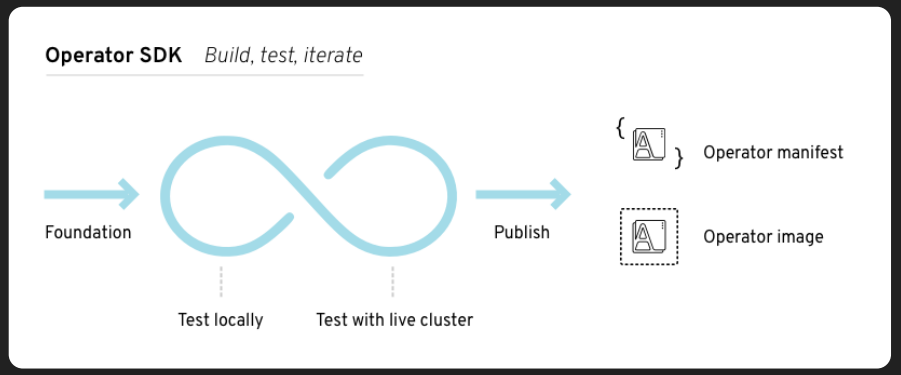
Operator Flow:
employs controller pattern (i.e. ReplicaSet, etc.) for your application's operation

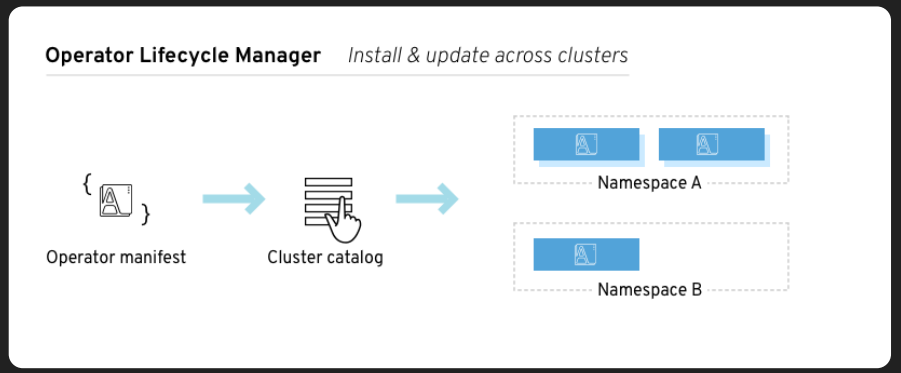
Building Operators

Leverage Existing Tools

- Operator Framework
- Metacontroller w/ self-implemented webhooks
- KUDO (Kubernetes Universal Declarative Operator)
- Kubebuilder

Publish your Operator on Operator Hub



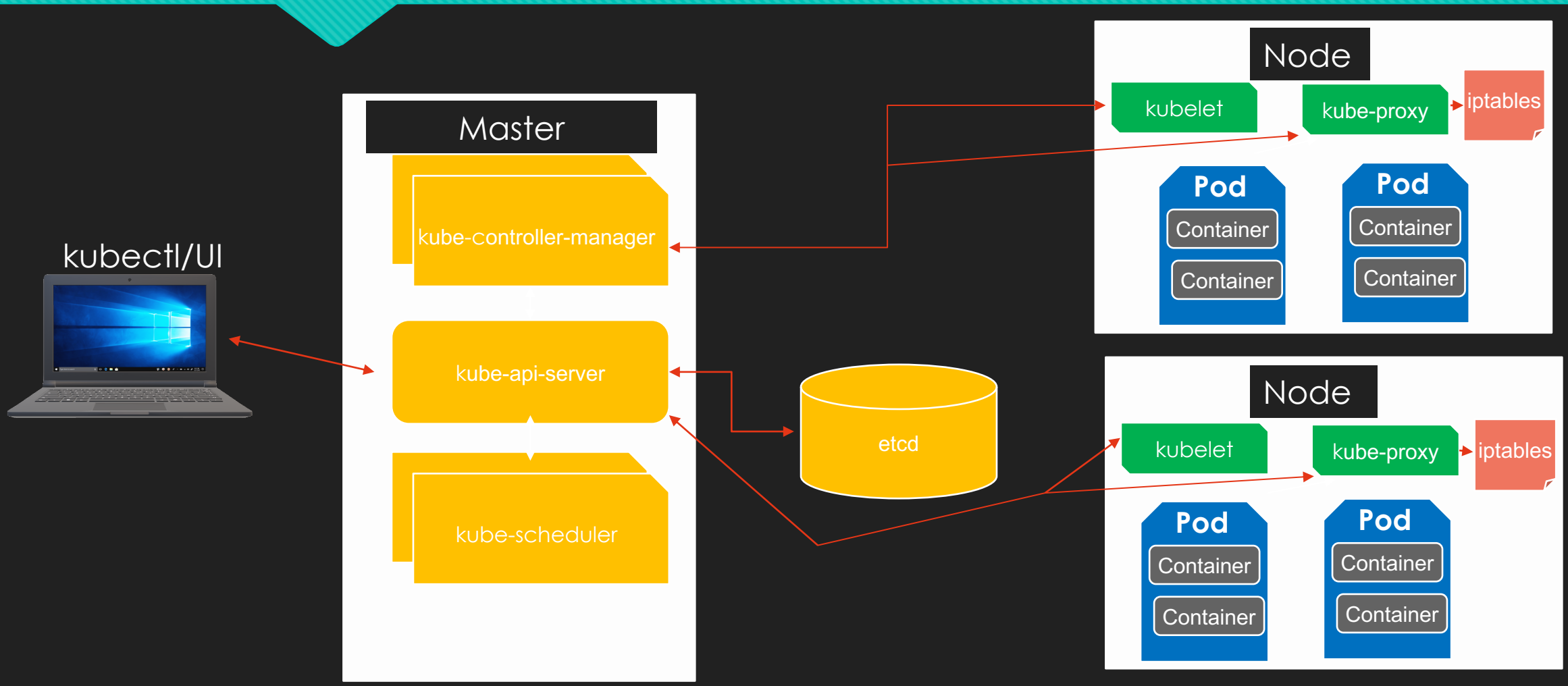


Operator Framework

- **Operator SDK**: Enables developers to build operators based on their expertise without requiring knowledge of Kubernetes API complexities.
- **Operator Lifecycle Management**: Oversees installation, updates, and management of the lifecycle of all of the operators (and their associated services) running across a Kubernetes cluster.
- **Operator Metering** (joining in the coming months): Enables usage reporting for operators that provide specialized services.

- See Introducing the Operator Framework
- See Operator Framework GitHub

Basic "Physical" Kubernetes Cluster Architecture



Kube-api-server

- Exposes versioned Kubernetes API
- Validates data for API call
- Only component to interact directly with datastore (etcd) where the state of resources are stored
- Once changes occur, notifies other components
- Multiple instances can be deployed for resiliency and handle requests simultaneously
- Everything is authenticated to the api-server via either certificates (node components) or tokens (users, service accounts)

etcd

- Distributed highly available key store which stores data for your entire cluster
- Have a backup plan for etcd in your environment just in case
- Only talks to kube-api-server
- Uses RAFT consensus protocol (etcd-raft)

Kube-scheduler

- Waits for new pods and assigns them to nodes
- Can use different schedulers for different pods based on their requirements such as **Poseidon-Firmament**
- Takes into account
  - individual and collective resource requirements
  - hardware/software/policy constraints
  - affinity and anti-affinity specifications
  - data locality
  - inter-workload interference
  - Deadlines

Kube-controller-manager

- Manage resources via control loops that check the desired state vs the current state and attempt to realize the desired state in the cluster via actions through communication with the API Server
- Manages controllers such as: (endpoints controller, namespace controller, serviceaccounts controller, ReplicaSet, Deployments, StatefulSets. DaemonSet, Garbage Collection, TTL Controller for Finished Resources, Jobs, CronJob, etc.)
- Multi-threaded daemon embedded with core control loops (i.e. those specified above and more) as well as option for multiple controller-managers for high availability.

Kubelet

- Agent running on each node (worker) in the cluster
- Communicates with api-server and container runtime (i.e. docker, containerd, etc.) to ensure pods specified in pod spec are running and healthy
- Coordinates interactions with the nodes such as:
  - Network (CNI)
  - Image lifecycle
  - Pod lifecycle
  - Storage Volumes
  - Quality of Service
  - Metrics/Logging

Kube-proxy

- Provides TCP/UDP level 4 (transport) load-balancing for traffic across pods connected to a service (via labels and selectors)
- Uses iptables by default (IPVS option available for better performance on clusters with 1000s of pod running in a a cluster) to map connections from services to pod ip addresses

Networking Kubernetes Basic

- Within a pod
  - containers share networking and ipc namespace meaning they can communicate over localhost using one port space and make ipc calls like if they were on the same vm
- Pod to Pod
  - Each pod has unique ip [endpoint] (which is the same as seen from the outside and inside)
  - A pod on one node can communicate with all pods on all other nodes with NAT
  - Agents on a node can communicate with all pods on that node
  - pods in the host network of a node can communicate with all pods on all nodes without NAT
  - Overlay networks [CNI] used to accomplish this such as flanneld or calico
- Pod to Service Communications (clusterip:internal_port -> endpoint:targetPort on pod) is the cluster internal-only option
- External to Service Communications (external_node_ip:node_port -> clusterip:internal_port -> endpoint:targetPort on pod) is one of many options

Kubernetes: Where to Start Resources

Kubernetes Basics Tutorials

- https://kubernetes.io/docs/tutorials/kubernetes-basics/

Setting up Kubernetes Using Docker to try it out

- https://docs.docker.com/docker-for-mac/#kubernetes

Kubernetes Tutorials

- https://kubernetes.io/docs/tutorials/

Kubernetes Docs

- https://kubernetes.io/docs/home/

Kubernetes by Example (Simple Overview of Kubernetes)

- http://kubernetesbyexample

OpenShift interactive learning tutorials (automatically spins up an OpenShift cluster for hands-on exercises)

- https://learn.openshift.com/introduction/

OpenShift on Linux on IBM Z trial

- https://cloud.redhat.com/openshift/install