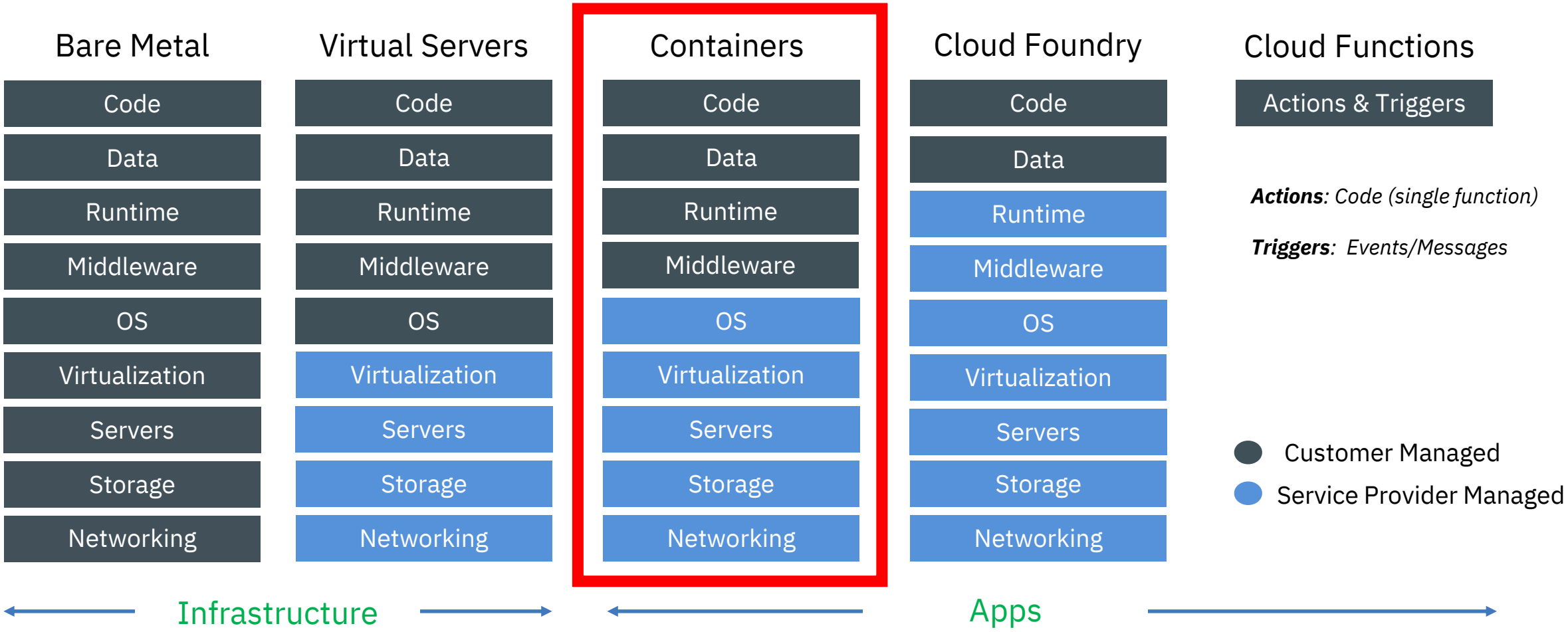
IBM Cloud Containers Workshop

Part2 - Orchestration

Cloud Computing – Levels of Responsibility



Why do you need Container Orchestration?

Pets

VS.

Cattle

Introduction to Orchestration

- Container Orchestration = Scheduling + Cluster management + Discovery

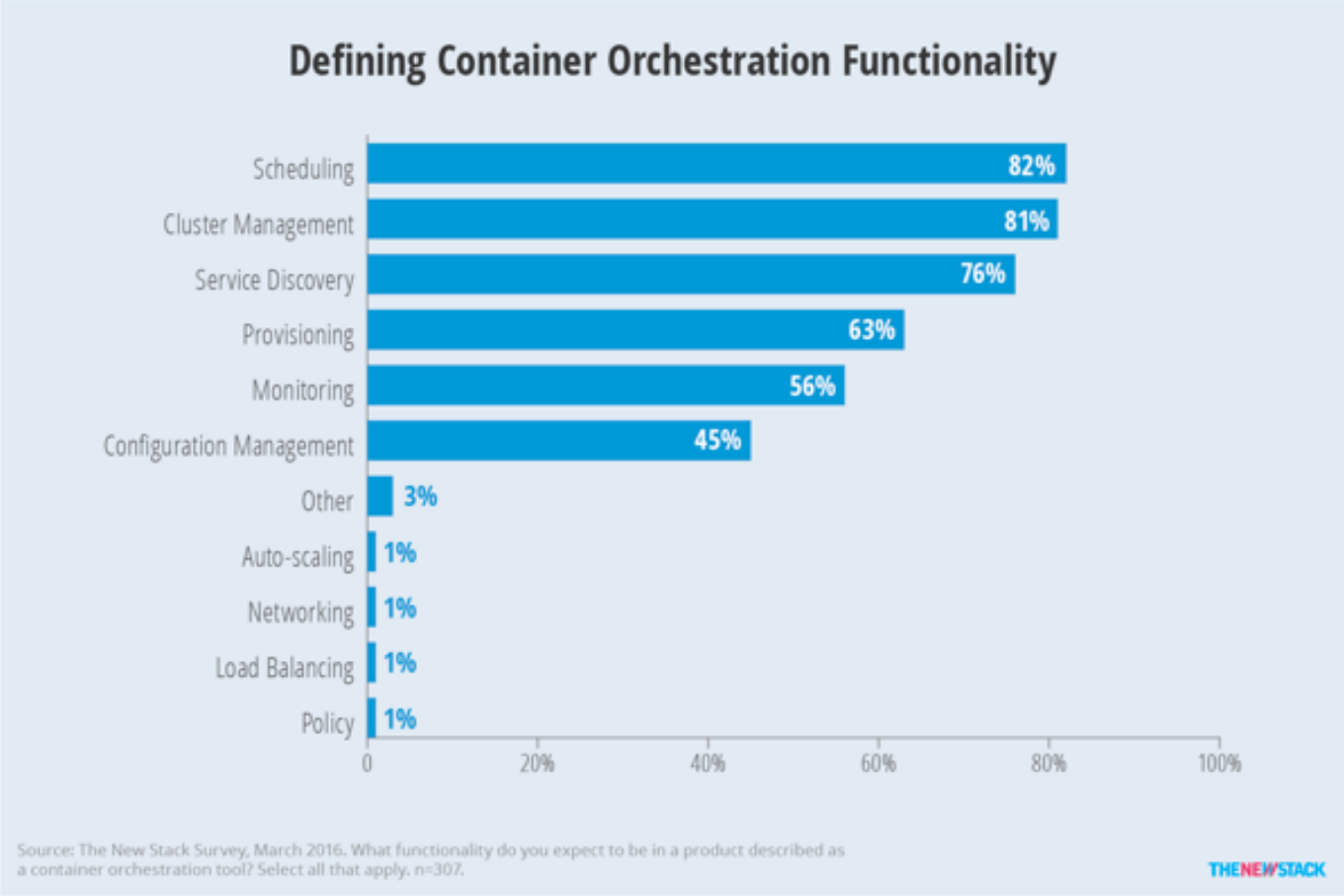
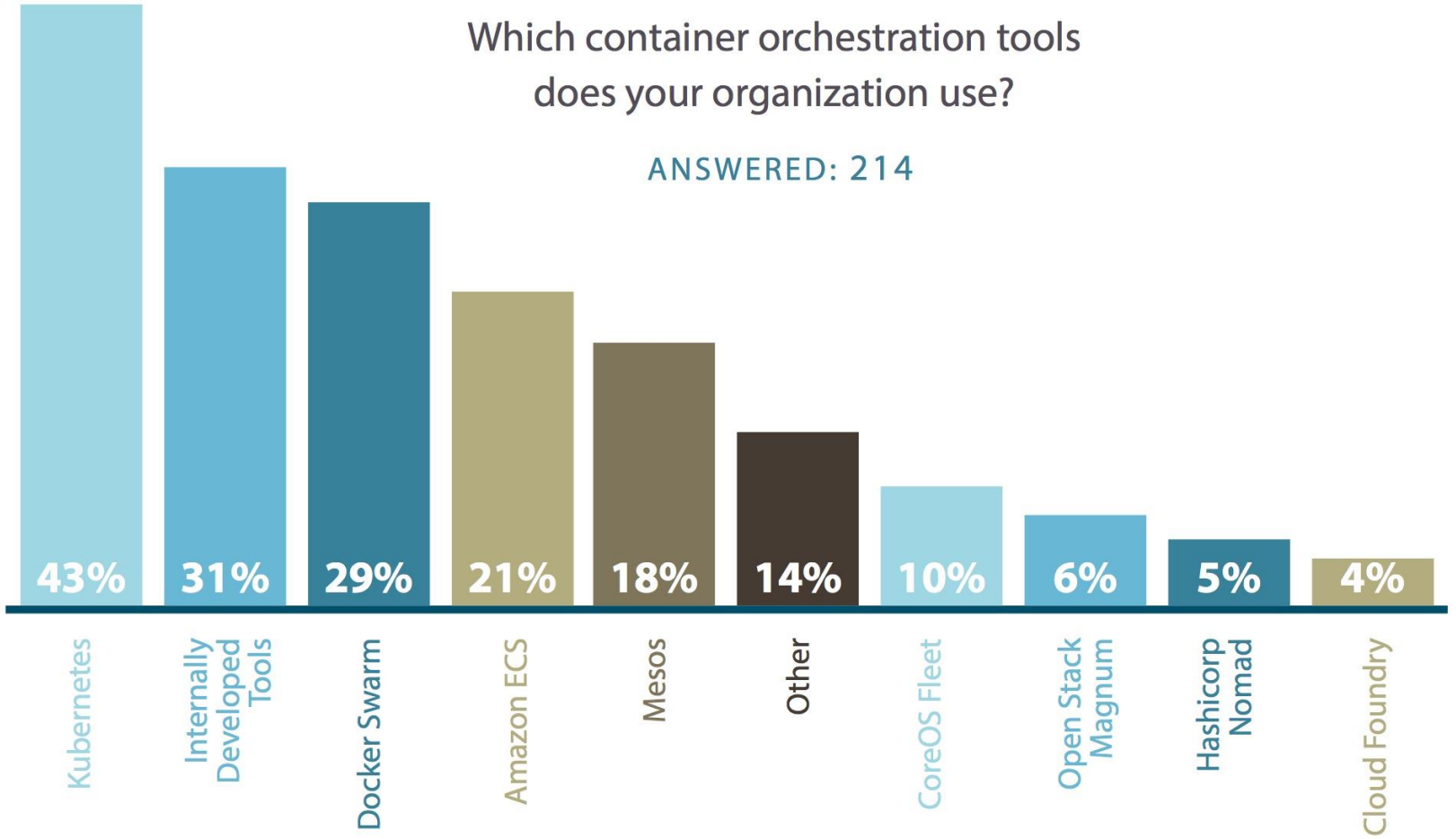
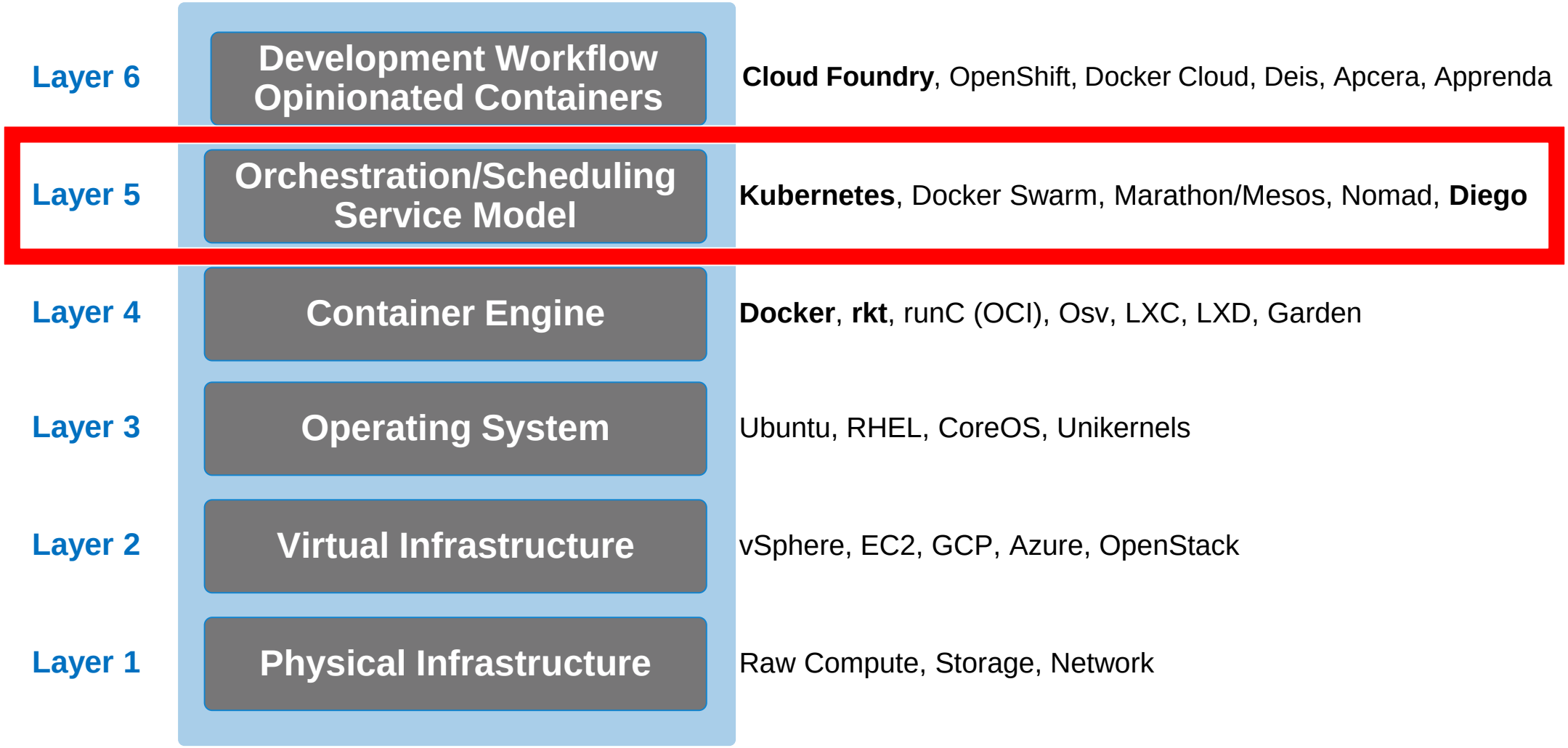


Figure 3: Only 45 percent of respondents consider configuration management to be part of a container orchestration product.

Different Orchestration Tools



Container Orchestration

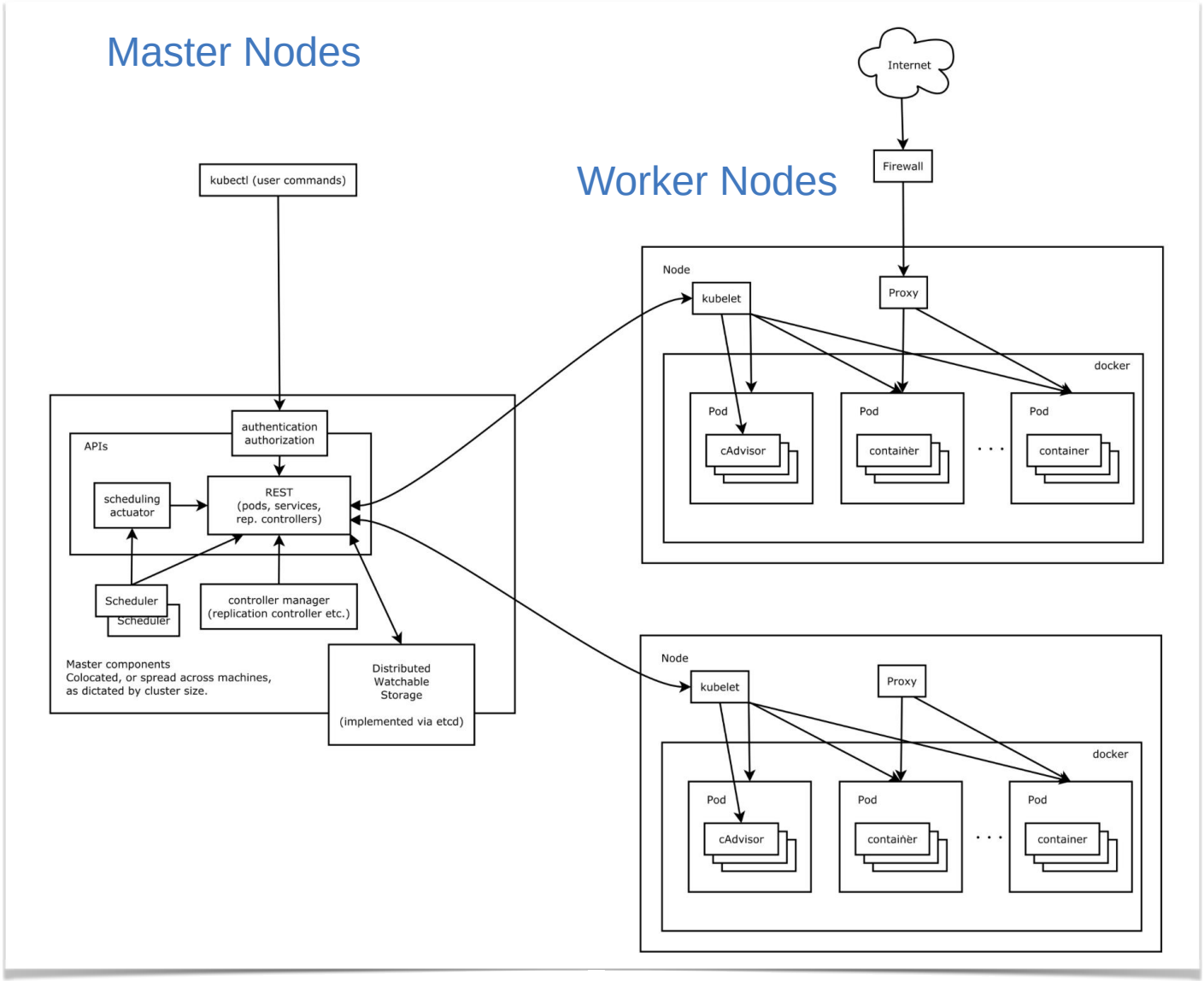


What is Kubernetes – K8S?

- Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts, providing container-centric infrastructure.

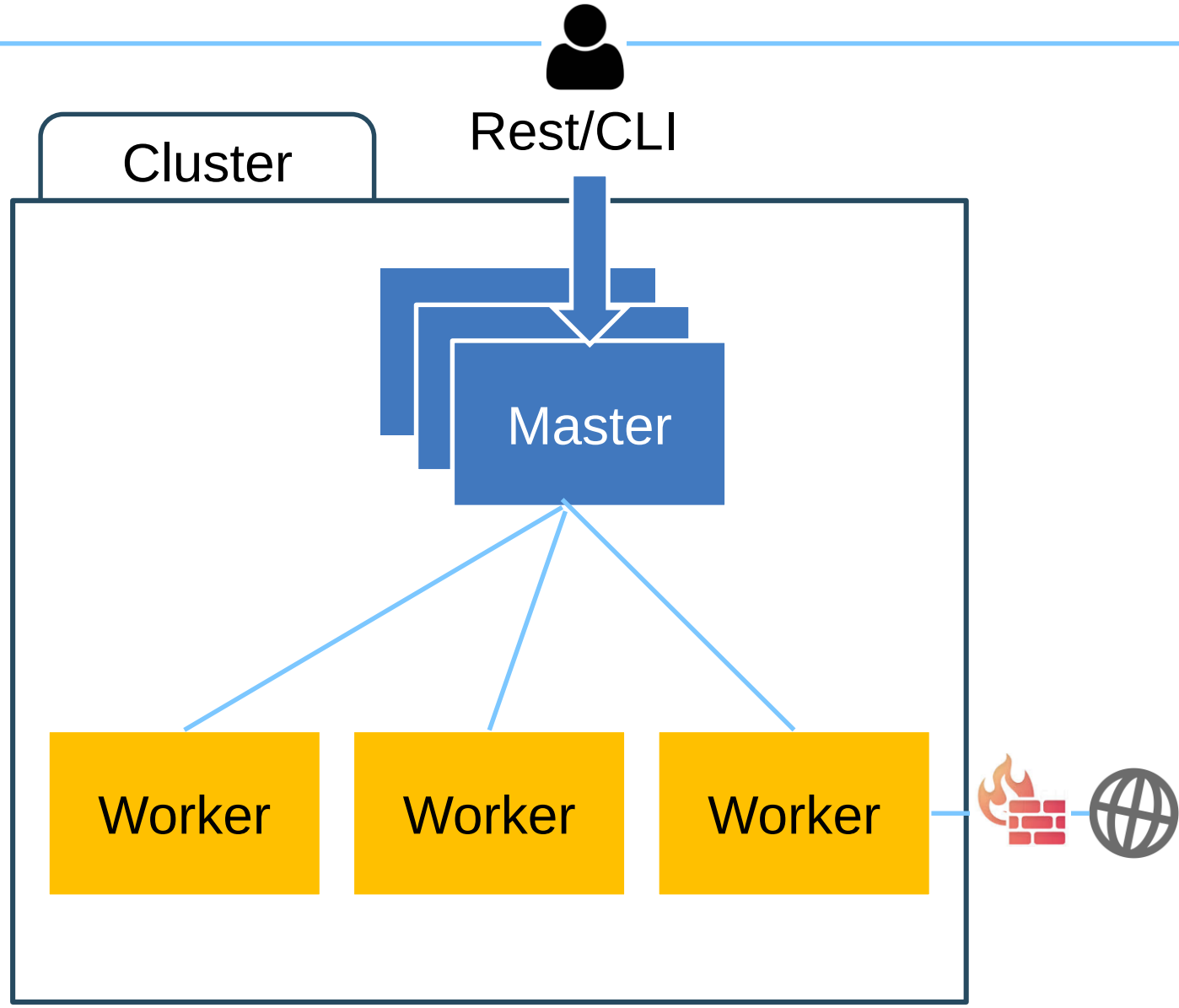
- **Container orchestrator**
- Runs and manages containers
- Supports **multiple cloud and bare-metal** environments
- Inspired and informed by Google's experiences and internal systems
- **100% Open source**, written in Go
- **Manage applications**, not machines
- Rich ecosystem of plug-ins for scheduling, storage, networking

Kubernetes Overview



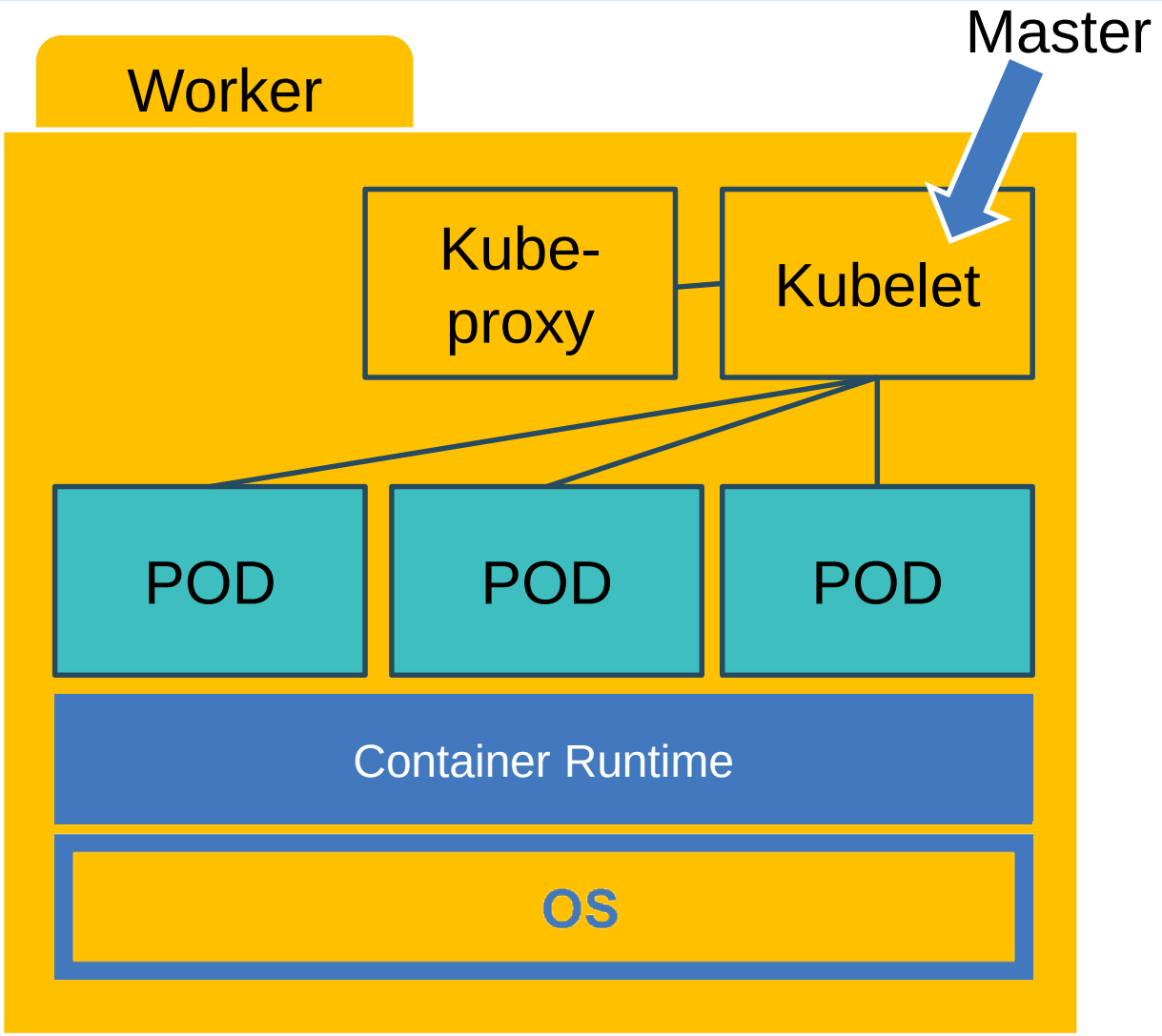
Cluster & Nodes

- Cluster is a set of nodes
- Nodes = hosts that run Kubernetes applications
- Master Node(s) – controls and manages the cluster :
  - Etcd
  - API Server
  - Controller Manager
  - Scheduler



Worker Nodes

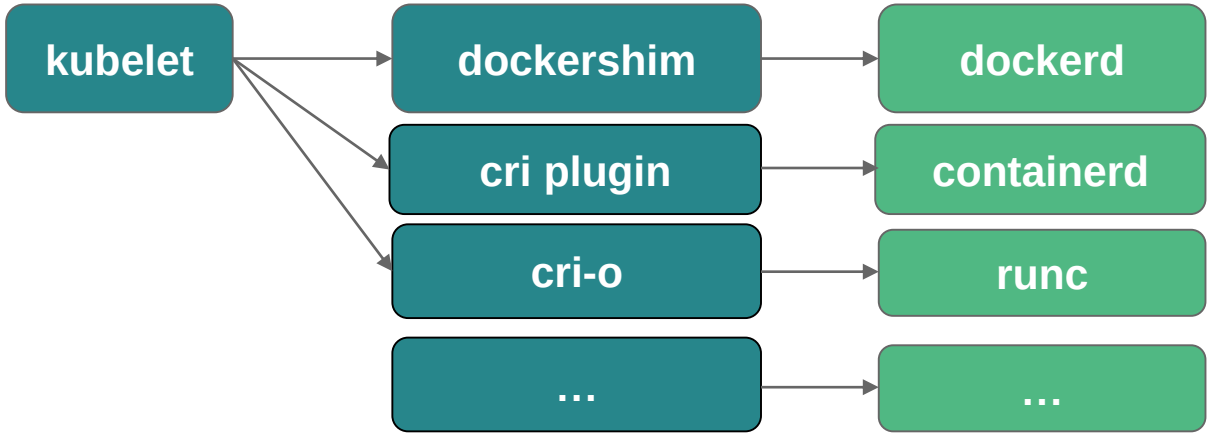
- This is where your applications are running in **PODs**
- Host Kubernetes services
  - Runs the **kubelet** agent to control the node from the master
  - **Kube-proxy** (network proxy service responsible for routing activities for inbound or ingress traffic to the PODs)
  - **Container Engine** on host



Kubernetes **doesn't run** containers

https://github.com/kubernetes/kubernetes/tree/release-1.4/pkg/kubelet/dockershim

Kubernetes **doesn't run** containers



CLOUD NATIVE COMPUTING FOUNDATION

OPEN CONTAINER INITIATIVE

The Kubernetes CRI

Monday, December 19, 2016

Introducing Container Runtime Interface (CRI) in Kubernetes

Editor's note: this post is part of a series of in-depth articles on what's new in Kubernetes 1.5

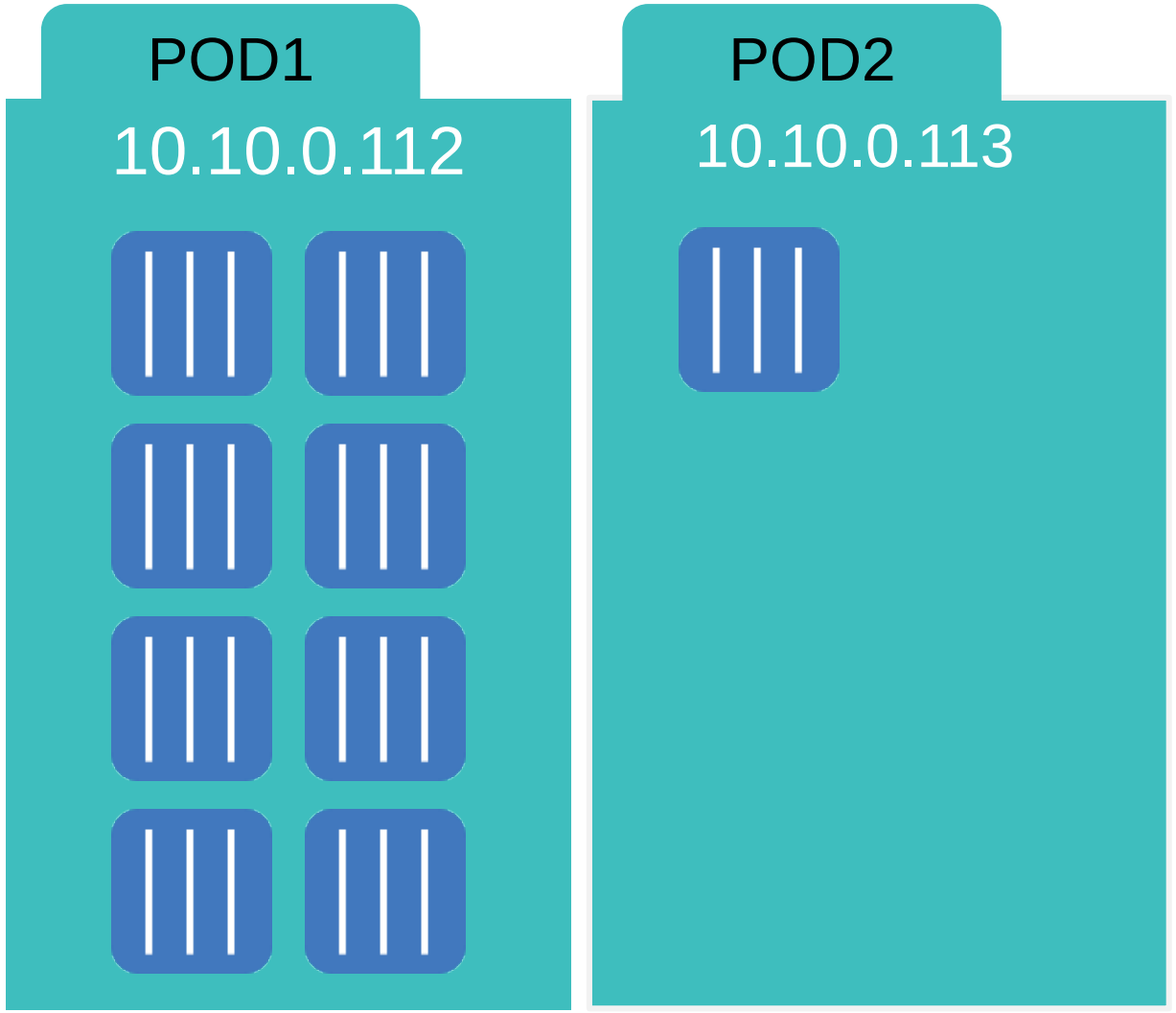
At the lowest layers of a Kubernetes node is the software that, among other things, starts and stops containers. We call this the "Container Runtime". The most widely known container runtime is Docker, but it is not alone in this space. In fact, the container runtime space has been rapidly evolving. As part of the effort to make Kubernetes more extensible, we've been working on a new plugin API for container runtimes in Kubernetes, called "CRI".

What is the CRI and why does Kubernetes need it?

Each container runtime has it own strengths, and many users have asked for Kubernetes to support more runtimes. In the Kubernetes 1.5 release, we are proud to introduce the Container Runtime Interface (CRI) -- a plugin interface which enables kubelet to use a wide variety of container runtimes, without the need to recompile. CRI consists of a protocol buffers and gRPC API, and libraries, with additional specifications and tools under active development. CRI is being released as Alpha in Kubernetes 1.5.

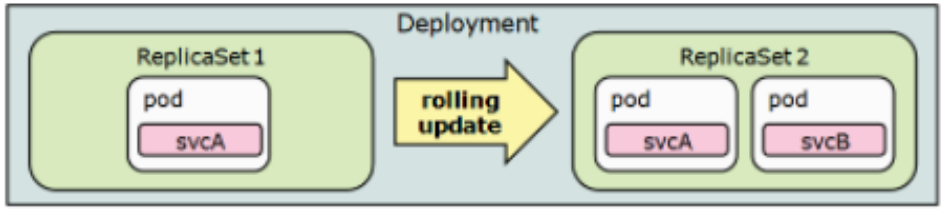
PODs

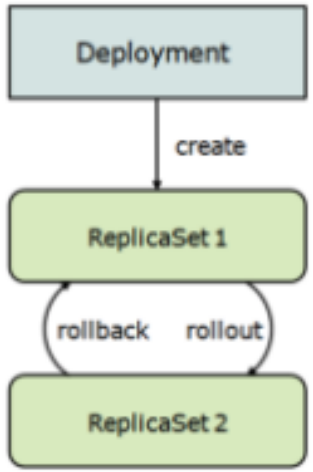
- Smallest **deployment unit** in K8s
  - **Single or colocated containers** that run on a worker node
- Each has its **own IP**
  - Pod shares a PID namespace, network, and hostname
- Inside a POD, form the network point of view, containers are in Network Container Mode (see Networking in Part #1)
- Important : check Network Ports



Controllers

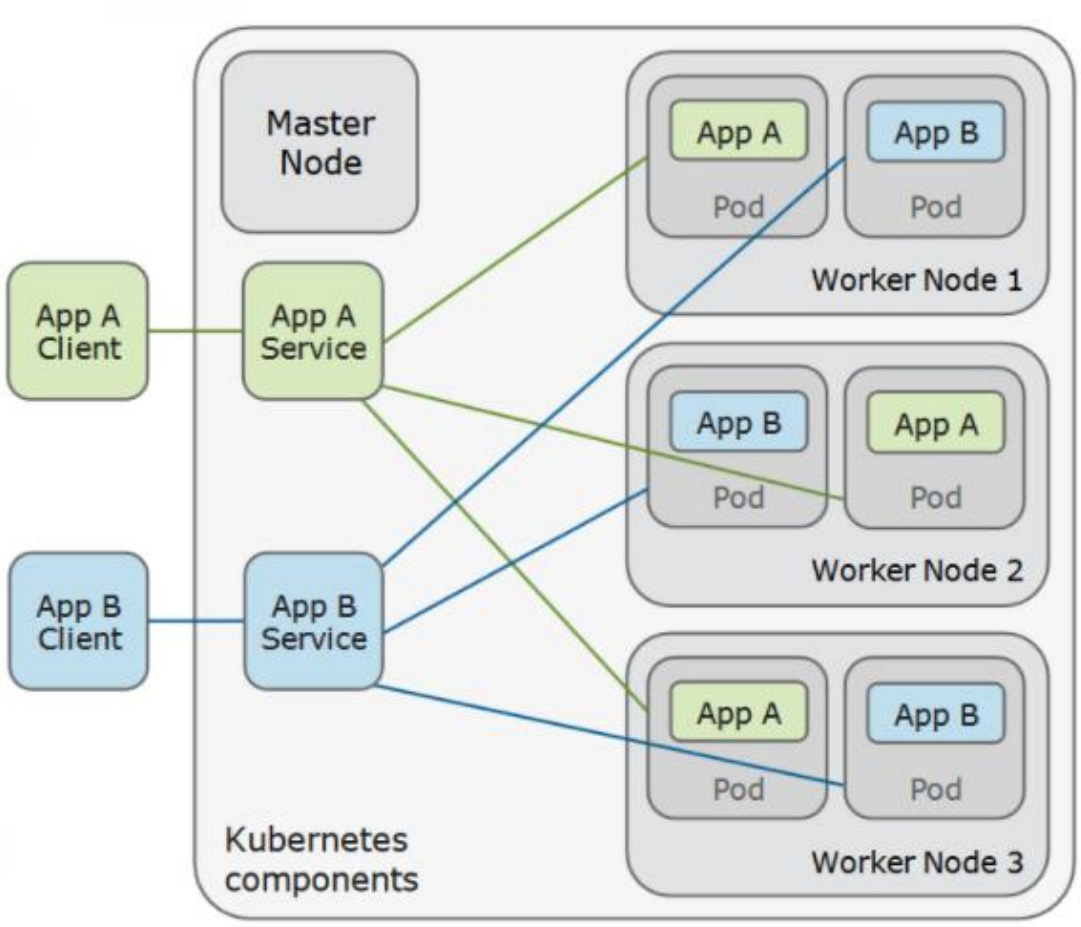
- They control the **workloads**
- Different types :
  - Jobs, CronJob,
  - DaemonSet
  - Deployment, Replicaset
- **Deployment** :
  - A **Deployment** controller provides **declarative** updates for **Pods** and **ReplicaSets**.
  - A ReplicaSet is a template that describes specifically what each pod should contain and it ensures that a specified number of pod replicas are running at any given time. It ensures **availability** and **scalability**





Kubernetes Services

- A service is a collection of pods **exposed as an endpoint**.
  - **ClusterIP**: exposes the cluster's internal IP (no communication with the outside)
  - **NodePort**: exposes the service on each node's IP at a static port
  - **LoadBalancer**: exposes the service externally by using a cloud provider's load balancer
- The service propagates state and networking information to all worker nodes.



Labels and Selectors

- Labels
  - **Metadata** assigned to K8s resources
  - Key-value pairs for identification
  - Critical to K8s as it relies on querying the cluster for resources that have certain labels
- **Selectors**
  - Equality (= or not =)
  - Set Based (in or not in)

Example labels:

- `"release" : "stable"`, `"release" : "canary"`
- `"environment" : "dev"`, `"environment" : "qa"`, `"environment" : "production"`
- `"tier" : "frontend"`, `"tier" : "backend"`, `"tier" : "cache"`
- `"partition" : "customerA"`, `"partition" : "customerB"`
- `"track" : "daily"`, `"track" : "weekly"`

```
$ kubectl get pods -l environment=prod,release=stable
```

Names and Namespaces

- Each resource object by type has a unique **name**.
- To achieve resource isolation, each **namespace** is a virtual cluster within the physical cluster.
  - Within a namespace, names of resources must be unique, but not across namespaces.
  - Namespaces can divide cluster resources.
- There are two initial namespaces:
  - *default* - This is the namespace for objects with no other namespace.
  - *kube-system* - The namespace for objects created by the Kubernetes system.

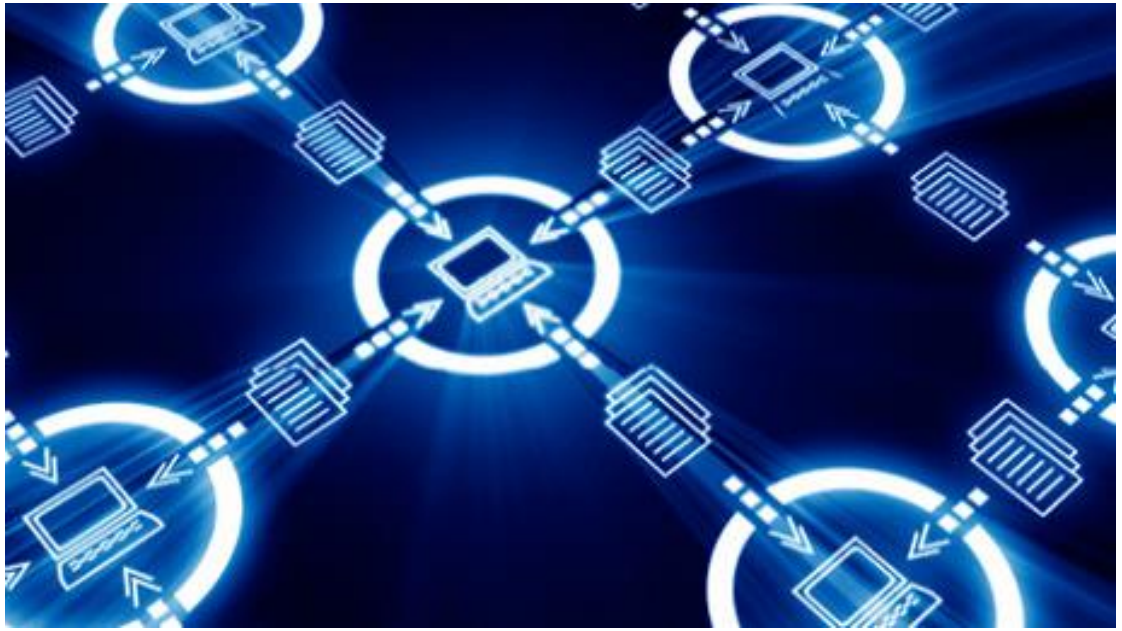
```
$ kubectl get namespaces
NAME          STATUS    AGE
default       Active    1d
kube-system   Active    1d
```

Configuration components

- **ConfigMaps**
  - ConfigMaps allow you to decouple configuration artifacts from image content to keep containerized applications portable.
- **Secrets**:
  - Sensitive info that containers need to consume
  - Encrypted in special volumes mounted automatically

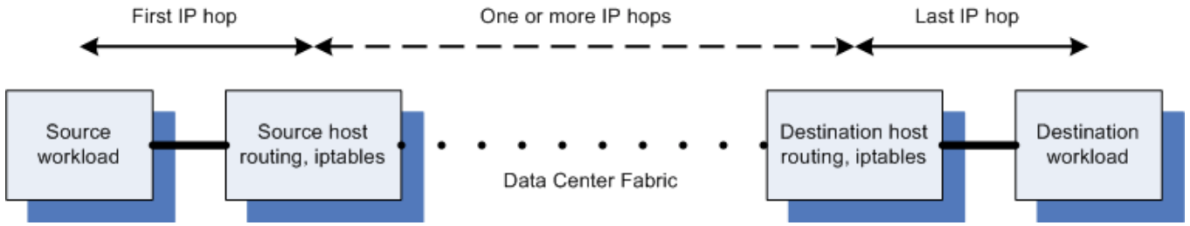
Configuration components

- **Network policy**
  - A Network policy is a specification of how groups of pods are allowed to communicate with each other and other network endpoints.
  - It uses labels to select pods and define rules which specify what traffic is allowed to the selected pods.



Calico Data Path: IP Routing and IPTABLES

- The calico/kube-policy-controller container runs as a pod on top of Kubernetes and implements the **NetworkPolicy** API



Suppose that IPv4 addresses for the workloads are allocated from a datacenter-private subnet of 10.65/16, and that the hosts have IP addresses from 172.18.203/24. If you look at the routing table on a host you will see something like this:

```
ubuntu@calico-ci02:~$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.18.203.1    0.0.0.0         UG    0      0        0 eth0
10.65.0.0       0.0.0.0         255.255.0.0     U     0      0        0 ns-db03ab89
-b4
10.65.0.21      172.18.203.126  255.255.255.255 UGH   0      0        0 eth0
10.65.0.22      172.18.203.129  255.255.255.255 UGH   0      0        0 eth0
10.65.0.23      172.18.203.129  255.255.255.255 UGH   0      0        0 eth0
10.65.0.24      0.0.0.0         255.255.255.255 UH    0      0        0 tapa429fb36
-04
172.18.203.0    0.0.0.0         255.255.255.0   U     0      0        0 eth0
```

```
Chain cali-from-wl-dispatch (2 references)
target     prot opt source               destination
cali-from-wl-dispatch-0  all  --  anywhere             anywhere             [goto]  /* cali:eBnVcASLTvMFg9XV */
cali-from-wl-dispatch-1  all  --  anywhere             anywhere             [goto]  /* cali:MldUGW-Du40orI9- */
cali-fw-cali2f9dbb62ca9  all  --  anywhere             anywhere             [goto]  /* cali:3L2Z8Gf72EsZIKAA */
cali-from-wl-dispatch-3  all  --  anywhere             anywhere             [goto]  /* cali:ZrLoQL8wJIHAgfVf */
cali-from-wl-dispatch-5  all  --  anywhere             anywhere             [goto]  /* cali:02H5UAoHug8r8x6m */
cali-from-wl-dispatch-6  all  --  anywhere             anywhere             [goto]  /* cali:sOd49VWBC4cE1fWn */
cali-from-wl-dispatch-9  all  --  anywhere             anywhere             [goto]  /* cali:X1yIW37zTvIP-0YO */
cali-fw-calib395ab80fdf  all  --  anywhere             anywhere             [goto]  /* cali:ZeMp2OhyOD-czLmE */
cali-fw-calic5f68922e7b  all  --  anywhere             anywhere             [goto]  /* cali:w1Nq3uLVTq2qH3oD */
cali-from-wl-dispatch-d  all  --  anywhere             anywhere             [goto]  /* cali:BhF3uys5r8zccmqw */
cali-from-wl-dispatch-e  all  --  anywhere             anywhere             [goto]  /* cali:cOXc1KJyb5GH35sQ */
cali-from-wl-dispatch-f  all  --  anywhere             anywhere             [goto]  /* cali:clSyOys7S0_GB2tR */
DROP       all  --  anywhere             anywhere             /* cali:2P0cqalbivIs3yNs */ /* Unknown interface */

Chain cali-from-wl-dispatch-0 (1 references)
target     prot opt source               destination
cali-fw-cali01b1faa5043  all  --  anywhere             anywhere             [goto]  /* cali:GGKM4rUyo36iZ8Yb */
cali-fw-cali092febb2d14  all  --  anywhere             anywhere             [goto]  /* cali:Pk1v5OzerJBINX0F */
DROP       all  --  anywhere             anywhere             /* cali:tRputS56cVeoNqYP */ /* Unknown interface */

Chain cali-from-wl-dispatch-1 (1 references)
target     prot opt source               destination
cali-fw-cali16dfc76a654  all  --  anywhere             anywhere             [goto]  /* cali:ymx3z1CMx7bhjz2E */
cali-fw-cali176404e5246  all  --  anywhere             anywhere             [goto]  /* cali:S1QW0TxEu1gsFIUc */
cali-fw-cali18ebc6b5707  all  --  anywhere             anywhere             [goto]  /* cali:tDxvYjyeEKdRtpvG */
cali-fw-cali1fd6184583a  all  --  anywhere             anywhere             [goto]  /* cali:MgPmzB7fgOraq1Si */
DROP       all  --  anywhere             anywhere             /* cali:cL5dCkd45Zcm9y7a */ /* Unknown interface */

Chain cali-from-wl-dispatch-3 (1 references)
target     prot opt source               destination
cali-fw-cali303beb222d4  all  --  anywhere             anywhere             [goto]  /* cali:NC5eQXlCshu3R7VN */
cali-fw-cali30c3290b8cd  all  --  anywhere             anywhere             [goto]  /* cali:UnaNwmKqQW14RETt */
cali-fw-cali3d1c4438cf0  all  --  anywhere             anywhere             [goto]  /* cali:4g5gp_NGwTnnYHFV */
cali-fw-cali3e3e8a47393  all  --  anywhere             anywhere             [goto]  /* cali:CD1myuMSjL0P5ZG0 */
DROP       all  --  anywhere             anywhere             /* cali:Inx8U2DMdYWYoOY3 */ /* Unknown interface */
```

Service discovery

Services need to discover each other dynamically, to get IP address and port detail to communicate with other services in the cluster

- Kubernetes provides two options for internal service discovery :
  - **Environment variable**: When a new Pod is created, environment variables from older services can be imported. This allows services to talk to each other. This approach enforces ordering in service creation.
  - **DNS**: Every service registers to the DNS service; using this, new services can find and talk to other services. Kubernetes provides the **kube-dns** service for this.

```
REDIS_MASTER_SERVICE_HOST=10.0.0.11
REDIS_MASTER_SERVICE_PORT=6379
REDIS_MASTER_PORT=tcp://10.0.0.11:6379
REDIS_MASTER_PORT_6379_TCP=tcp://10.0.0.11:6379
REDIS_MASTER_PORT_6379_TCP_PROTO=tcp
REDIS_MASTER_PORT_6379_TCP_PORT=6379
REDIS_MASTER_PORT_6379_TCP_ADDR=10.0.0.11
```

Autoscaling

- Kubernetes implements autoscaling through Horizontal Pod Autoscaling (HPA). HPA automatically scales the number of pods in a replication controller, deployment, or replica set by matching the observed average CPU utilization to a specified target. HPA can also autoscale based on application-provided metrics.

- Metrics to drive this are fetched in two ways: direct Heapster access and REST client access. Kubernetes Heapster enables container cluster monitoring and performance analysis.. The default configuration is to:

- query every 30 seconds

- maintain 10% tolerance

- wait 3 minutes after scale-up
- wait another 5 minutes after scale-down
- The following kubectl command creates an HPA instance that maintains between 1 and 10 replicas of the pod controlled by the deployment. Additionally, the command maintains an average CPU utilization across all pods of 50%.

```
$ kubectl autoscale deployment <deployment-name> --cpu-percent=50
  --min=1 --max=10 deployment "<hpa-name>" autoscaled
```

(a few) kubectl commands

- CLUSTER :
  - Get the state of your cluster
    - $ kubectl cluster-info
  - Get all the nodes of your cluster
    - $ kubectl get nodes
- DEPLOYMENTS :
  - Get deployments from namespace «kube-system »
    - $ kubectl get deployments -n kube-system
  - Get details from deployment
    - $ kubectl describe deploy <NAME_OF_DEPLOYMENT>
- SERVICES :
  - Get info about the services of your cluster
    - $ kubectl get services
  - Get full config info about a Service
    - $ kubectl get service <NAME_OF_SERVICE> -o json
  - Delete a Service
    - $ kubectl delete service NAME_OF_THE_SERVICE
- PODS :
  - Get info about the pods of your cluster
    - $ kubectl get pods –all-namespaces
  - Get the IP of a Pod
    - $ kubectl get pod <NAME_OF_POD> -template={{.status.podIP}}
  - Delete a Pod
    - $ kubectl delete pod NAME

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

kubectl apply -f filename.yml

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: hw-demo-deployment
spec:
  replicas: 3
  template:
    metadata:
      name: pod-liveness-http
      labels:
        run: hw-demo-health
        test: hello-world-demo
```

```
spec:
      containers:
        - name: hw-demo-container
          image: ” registry.ng.bluemix.net/pod1/hello-world:2"
          imagePullPolicy: Always
          livenessProbe:
            httpGet:
              path: /healthz
              port: 8080
            initialDelaySeconds: 5
            periodSeconds: 5
```

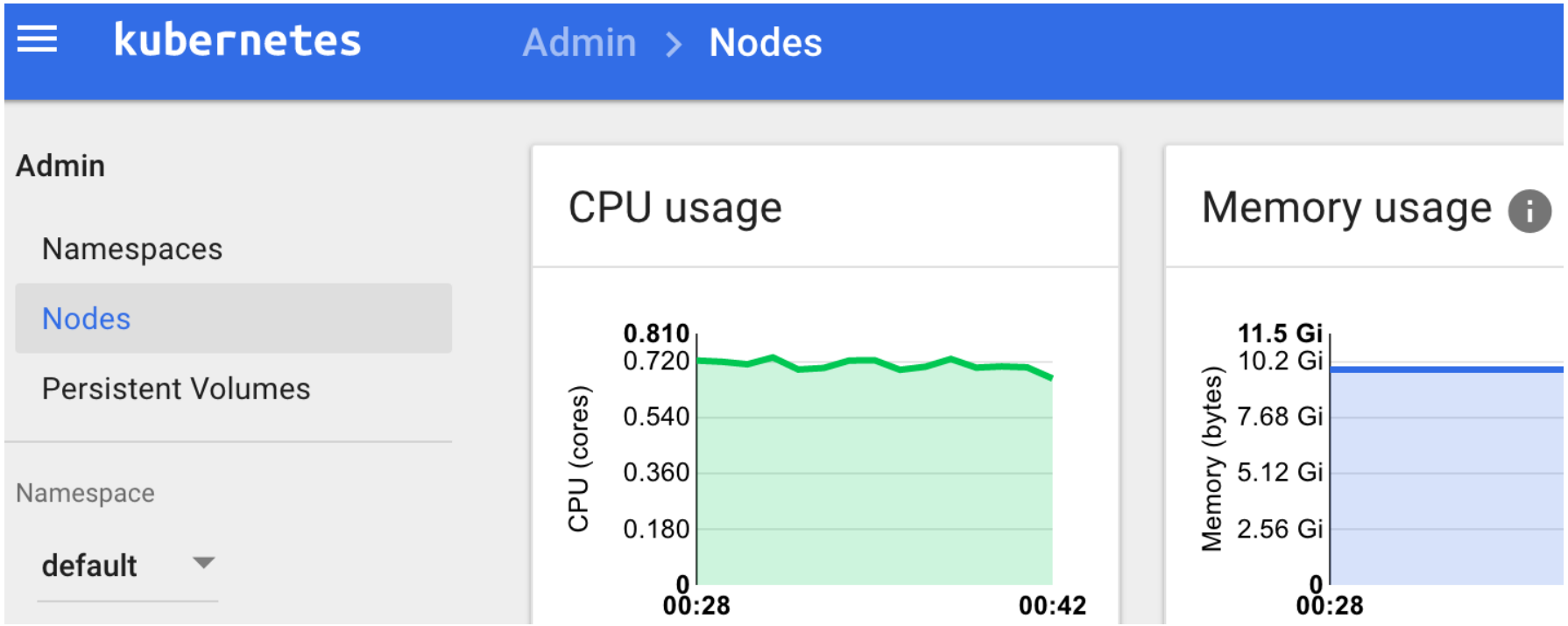
kubectl delete -f filename.yml

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: hw-demo-deployment
spec:
  replicas: 3
  template:
    metadata:
      name: pod-liveness-http
      labels:
        run: hw-demo-health
        test: hello-world-demo
```

```
spec:
      containers:
        - name: hw-demo-container
          image: ” registry.ng.bluemix.net/pod1/hello-world:2"
          imagePullPolicy: Always
          livenessProbe:
            httpGet:
              path: /healthz
              port: 8080
            initialDelaySeconds: 5
            periodSeconds: 5
```

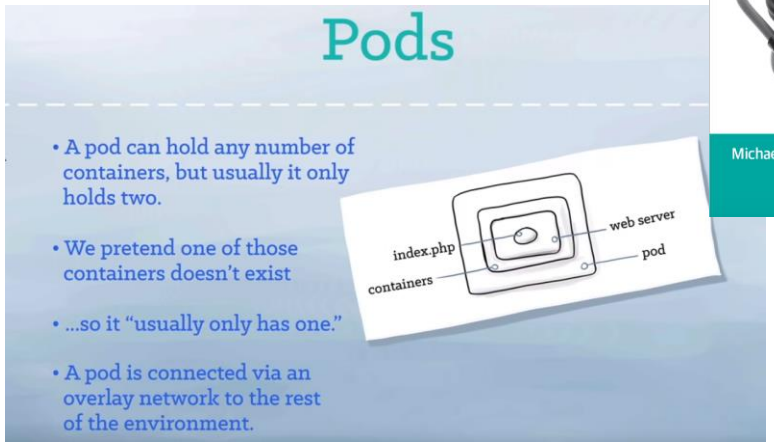
Monitoring Containers

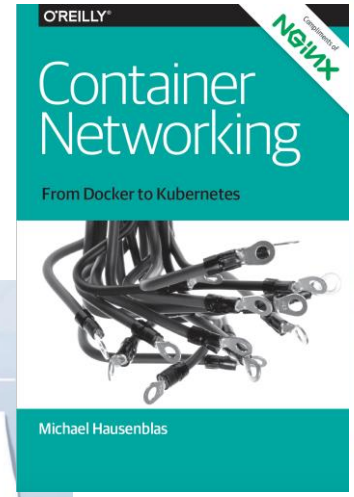
- **Integrated** logging and monitoring on IBM Cloud based on ELK stack
- Native Kubernetes **dashboard** or API



Books, eBooks and links

- Getting Started with Kubernetes - Second Edition
- Mastering Kubernetes
- Container Networking





- https://kubernetes.io/docs/tutorials/kubernetes-basics/
- https://kubernetes.io/docs/concepts/
- https://towardsdatascience.com/key-kubernetes-concepts-62939f4bc08e
- https://www.youtube.com/watch?v=4ht22ReBjno