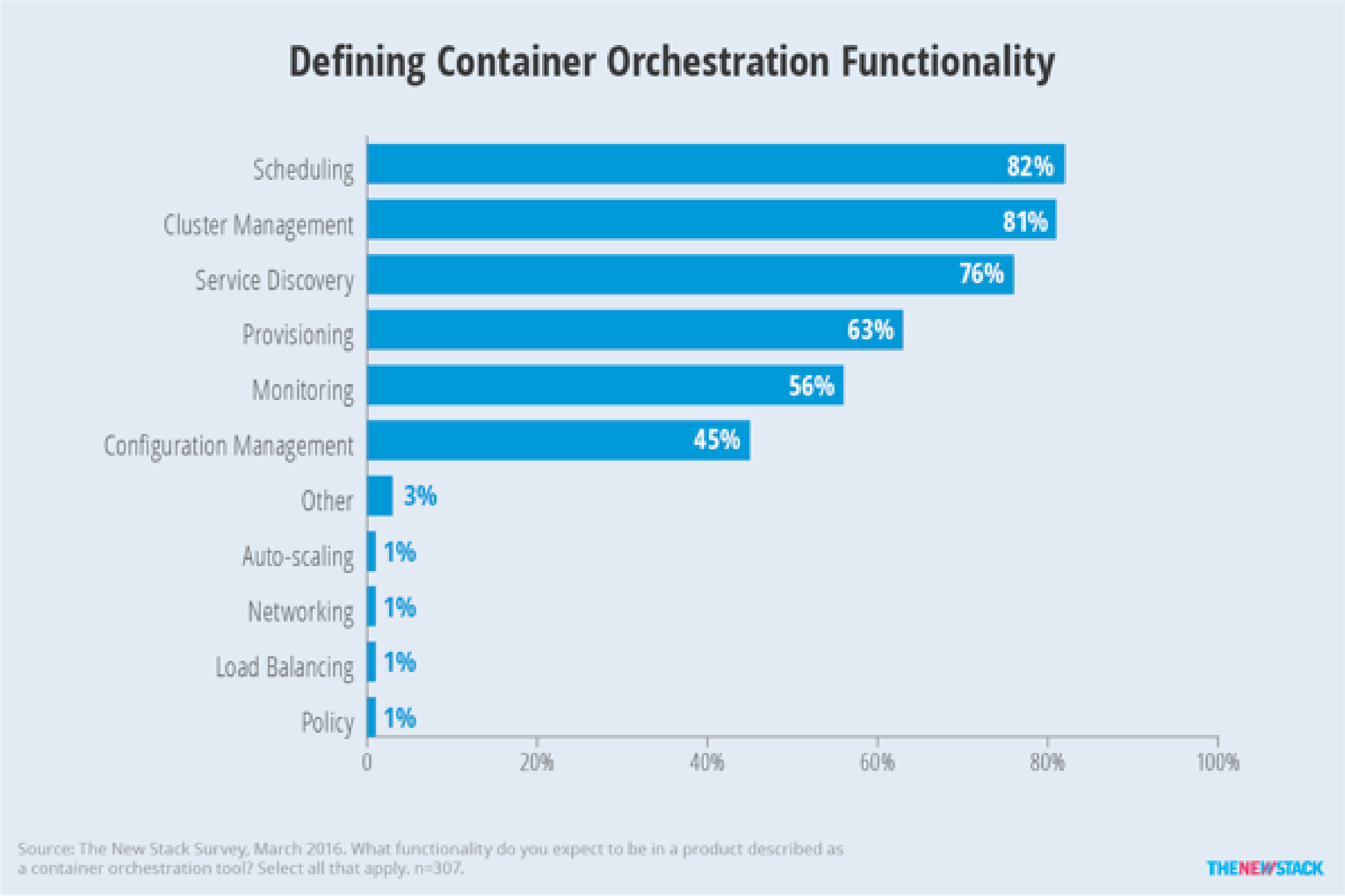
IBM Cloud Containers Workshop

Part2 - Orchestration

Introduction to Orchestration

- Container Orchestration = Scheduling + Cluster management + Discovery

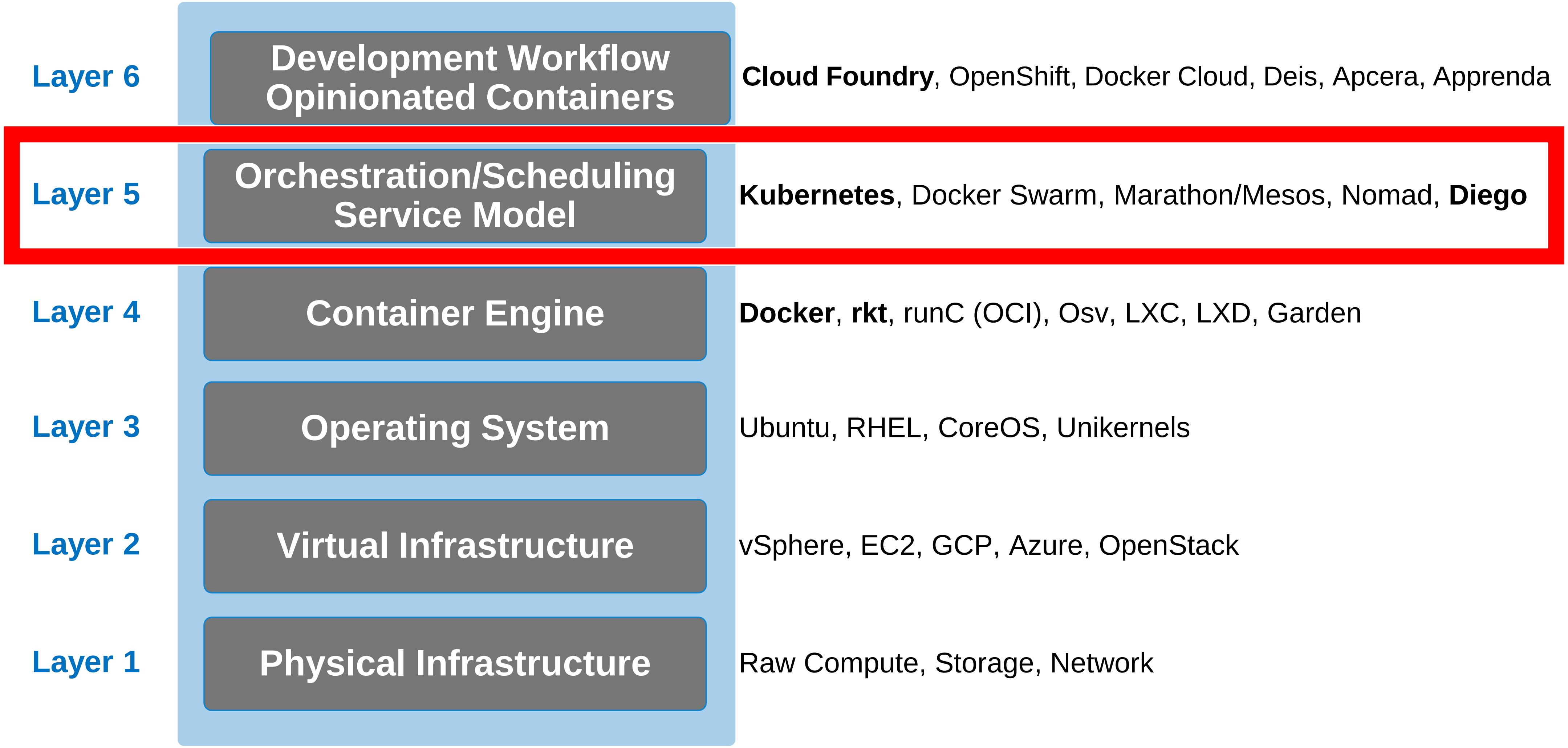


Defining Container Orchestration Functionality

Source: The New Stack Survey, March 2016. What functionality do you expect to be in a product described as a container orchestration tool? Select all that apply. n=307.

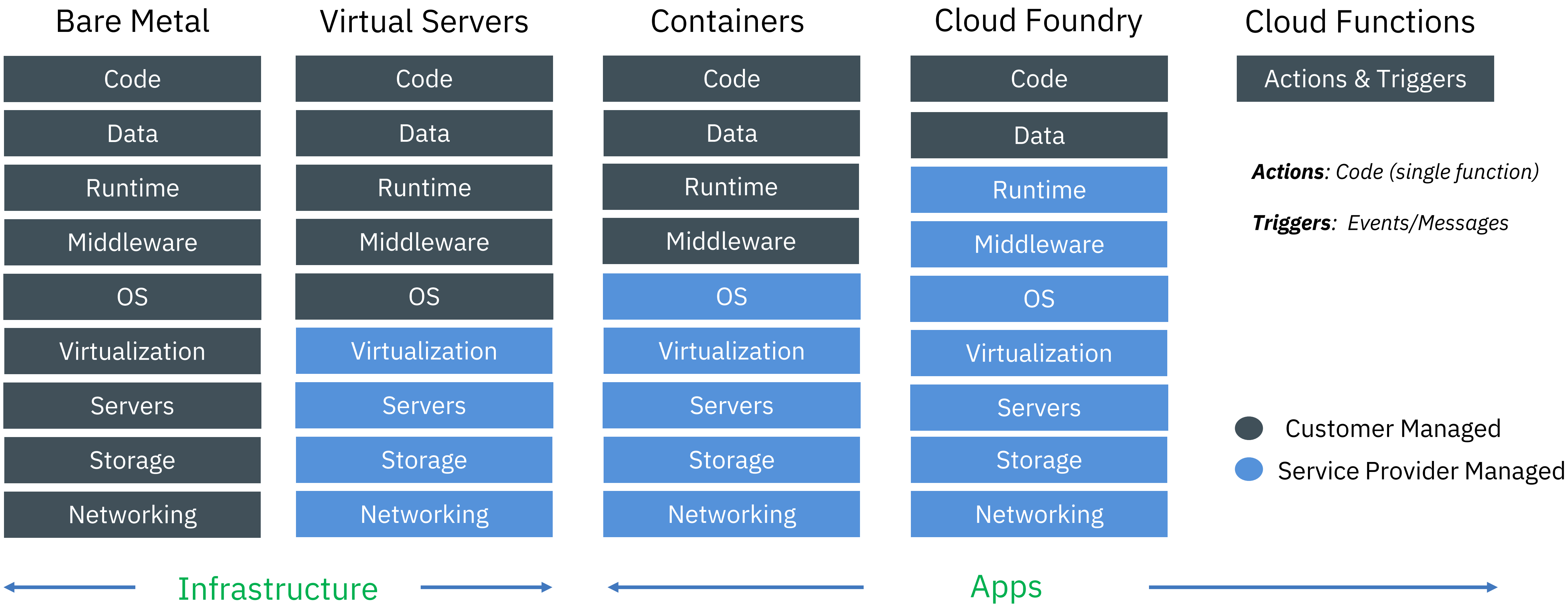
Figure 3: Only 45 percent of respondents consider configuration management to be part of a container orchestration product.

Container Orchestration with Kubernetes



Layer 6	**Development Workflow Opinionated Containers**	**Cloud Foundry**, OpenShift, Docker Cloud, Deis, Apcera, Apprenda
Layer 5	**Orchestration/Scheduling Service Model**	**Kubernetes**, Docker Swarm, Marathon/Mesos, Nomad, **Diego**
Layer 4	**Container Engine**	**Docker**, **rkt**, runC (OCI), Osv, LXC, LXD, Garden
Layer 3	**Operating System**	Ubuntu, RHEL, CoreOS, Unikernels
Layer 2	**Virtual Infrastructure**	vSphere, EC2, GCP, Azure, OpenStack
Layer 1	**Physical Infrastructure**	Raw Compute, Storage, Network

• Cloud Computing – Levels of Responsibility



Bare Metal	Virtual Servers	Containers	Cloud Foundry	Cloud Functions
● Code	● Code	● Code	● Code	● Actions & Triggers
● Data	● Data	● Data	● Data	
● Runtime	● Runtime	● Runtime	🔵 Runtime	
● Middleware	● Middleware	● Middleware	🔵 Middleware	
● OS	● OS	🔵 OS	🔵 OS	
● Virtualization	🔵 Virtualization	🔵 Virtualization	🔵 Virtualization	
● Servers	🔵 Servers	🔵 Servers	🔵 Servers	
● Storage	🔵 Storage	🔵 Storage	🔵 Storage	
● Networking	🔵 Networking	🔵 Networking	🔵 Networking	

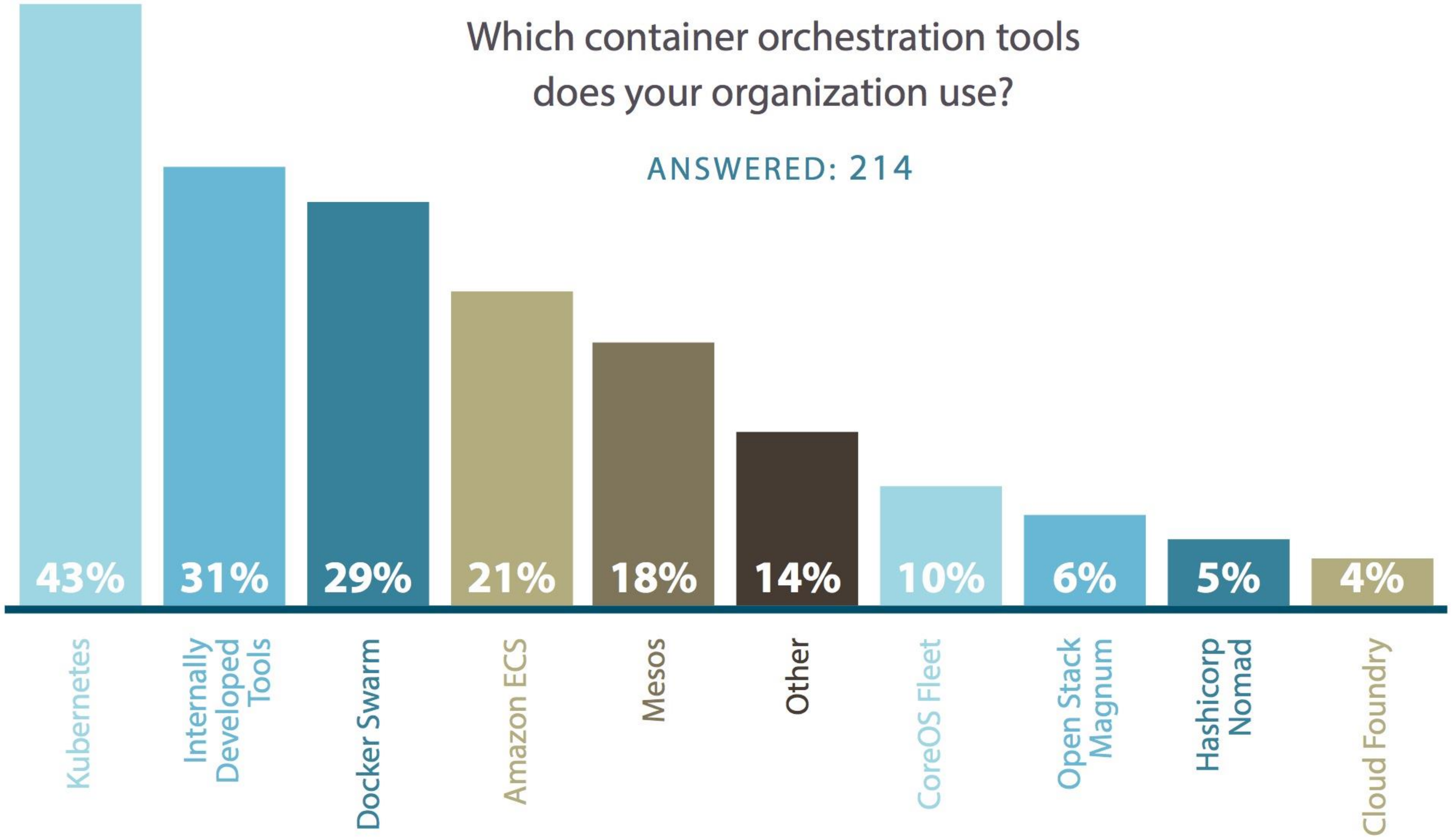
***Actions**: Code (single function)*

***Triggers**: Events/Messages*

● Customer Managed

🔵 Service Provider Managed

Different Orchestration Tools

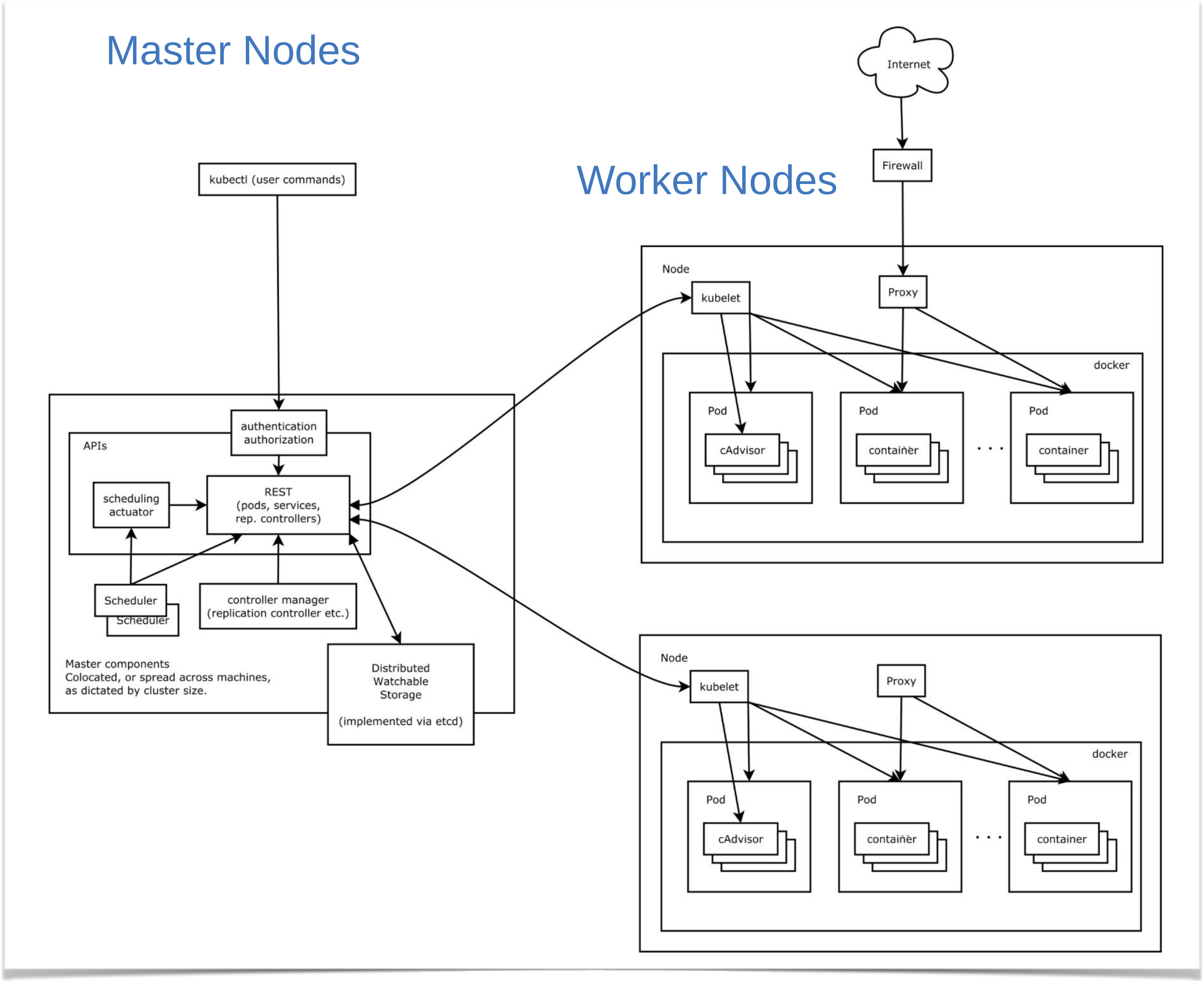


What is Kubernetes – K8S?

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

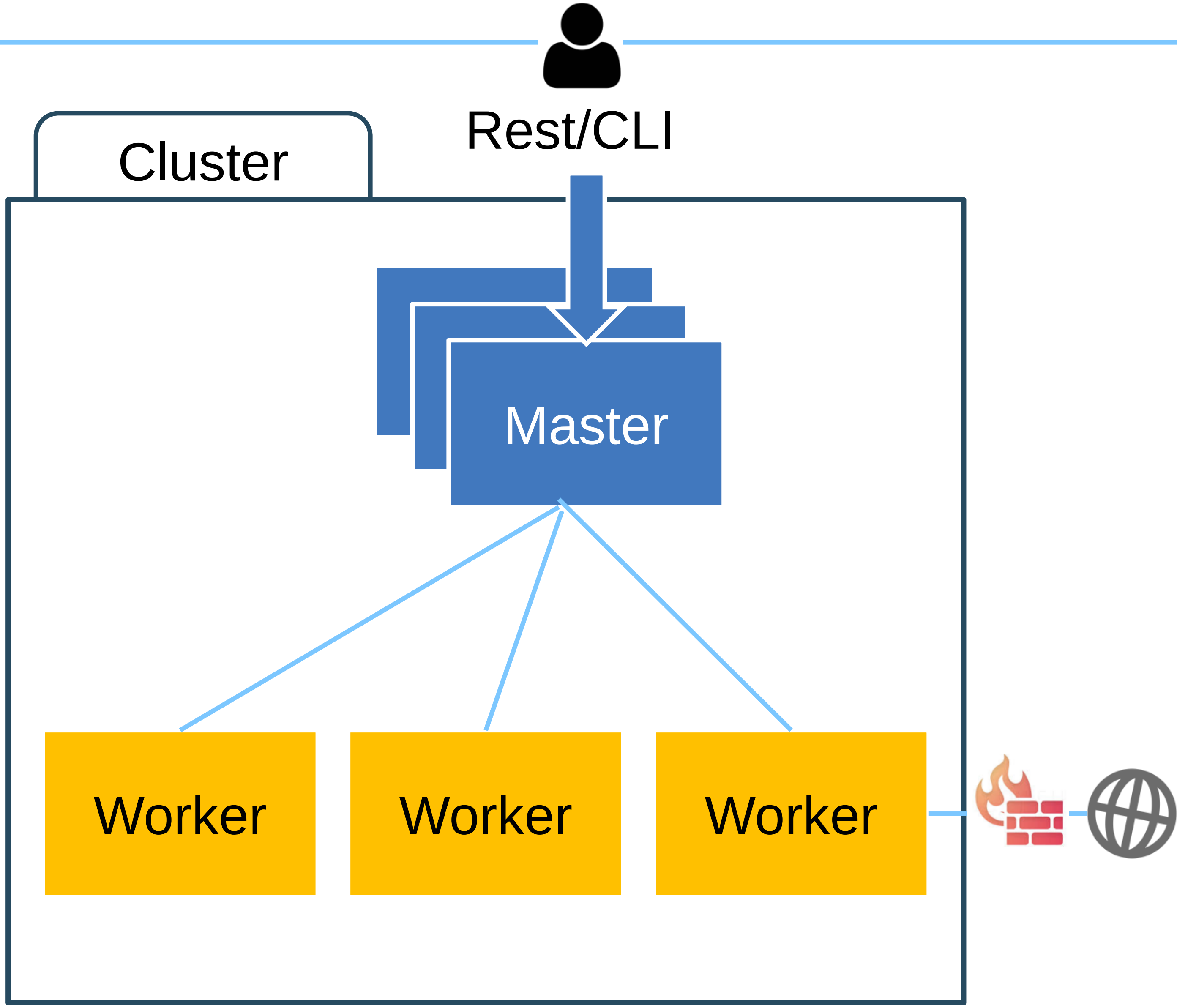
- **Container orchestrator**
- Runs and manages containers
- Supports **multiple cloud and bare-metal** environments
- Inspired and informed by Google's experiences and internal systems
- **100% Open source**, written in Go
- **Manage applications**, not machines
- Rich ecosystem of plug-ins for scheduling,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

Kubernetes Overview



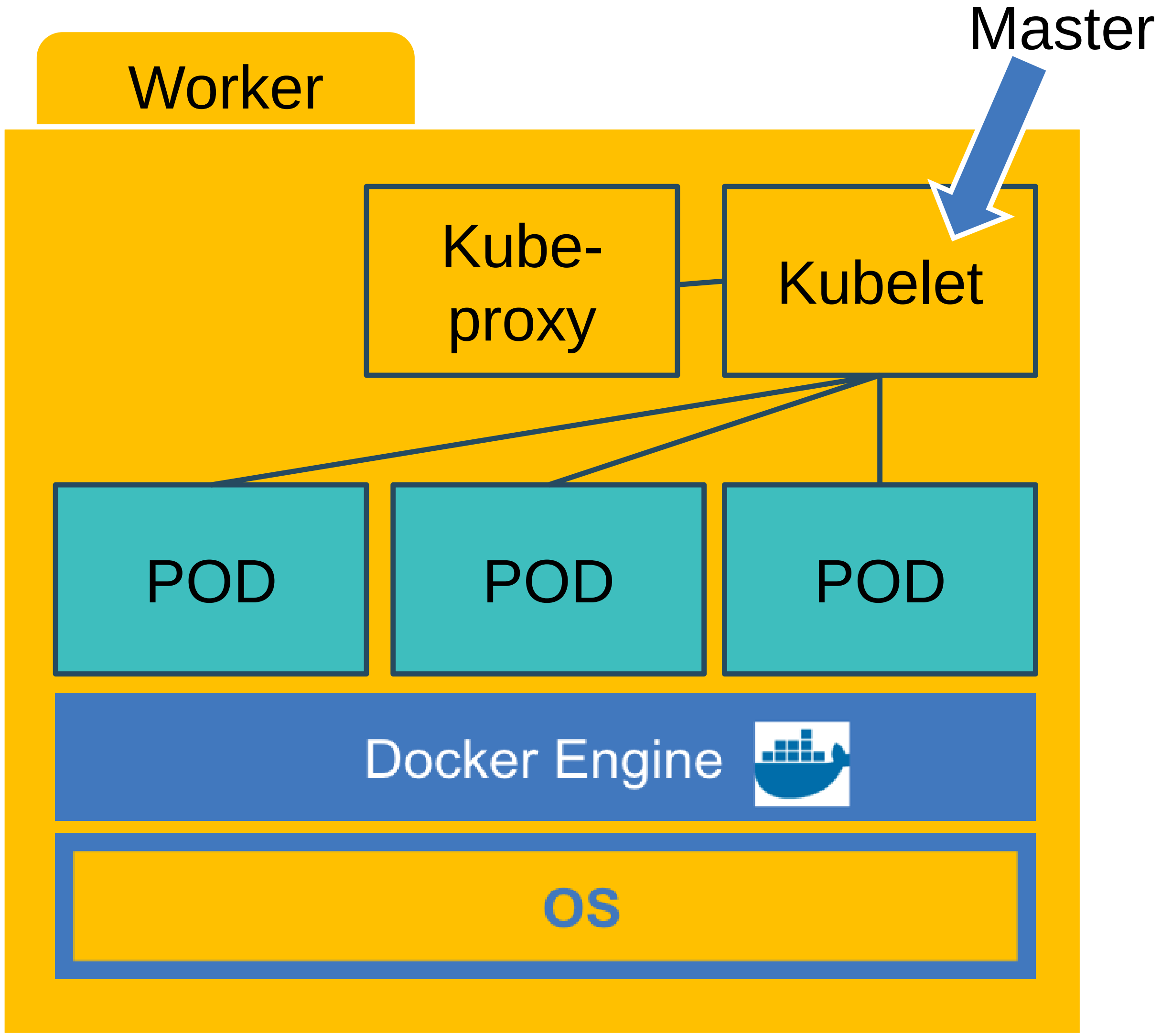
Cluster & Nodes

- 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 – where the applications run



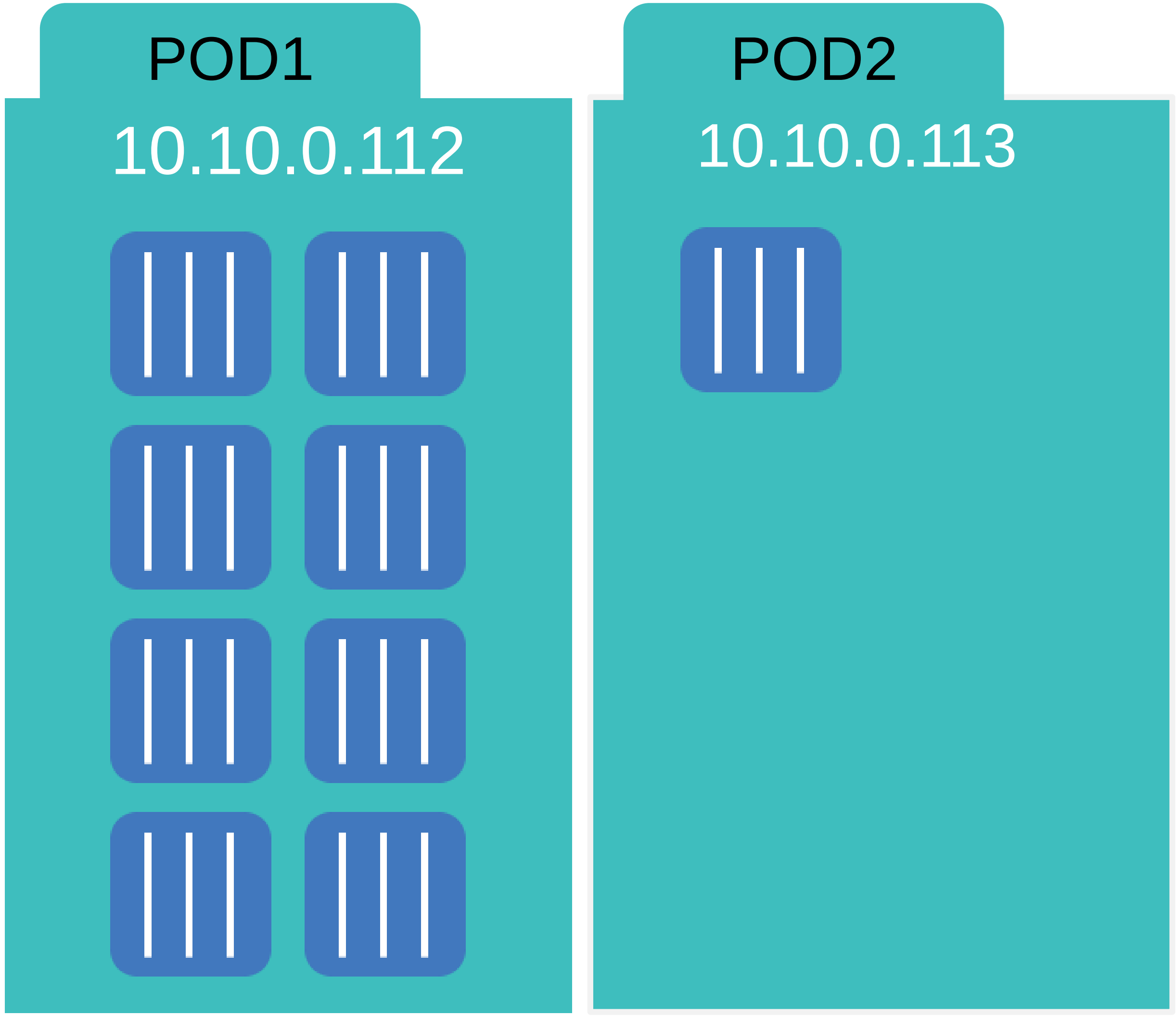
Worker Nodes

- 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)
- **Docker Engine** Host



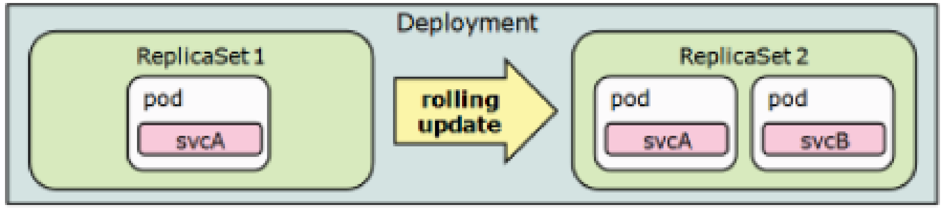
PODs

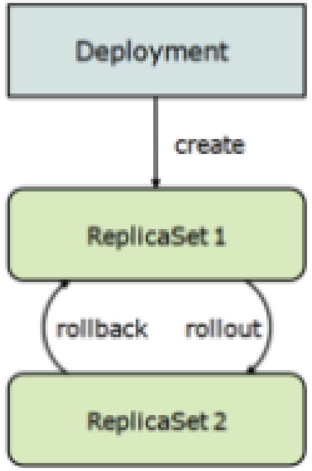
- Smallest **deployment unit** in K8s
- **Collection of containers** that run on a worker node
- Each has its **own IP**
- Pod shares a PID namespace, network, and hostname
- Inside a POD, form the network point of view, containers are in Network Container Mode (see Networking in Part #1)
- Important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



Controllers

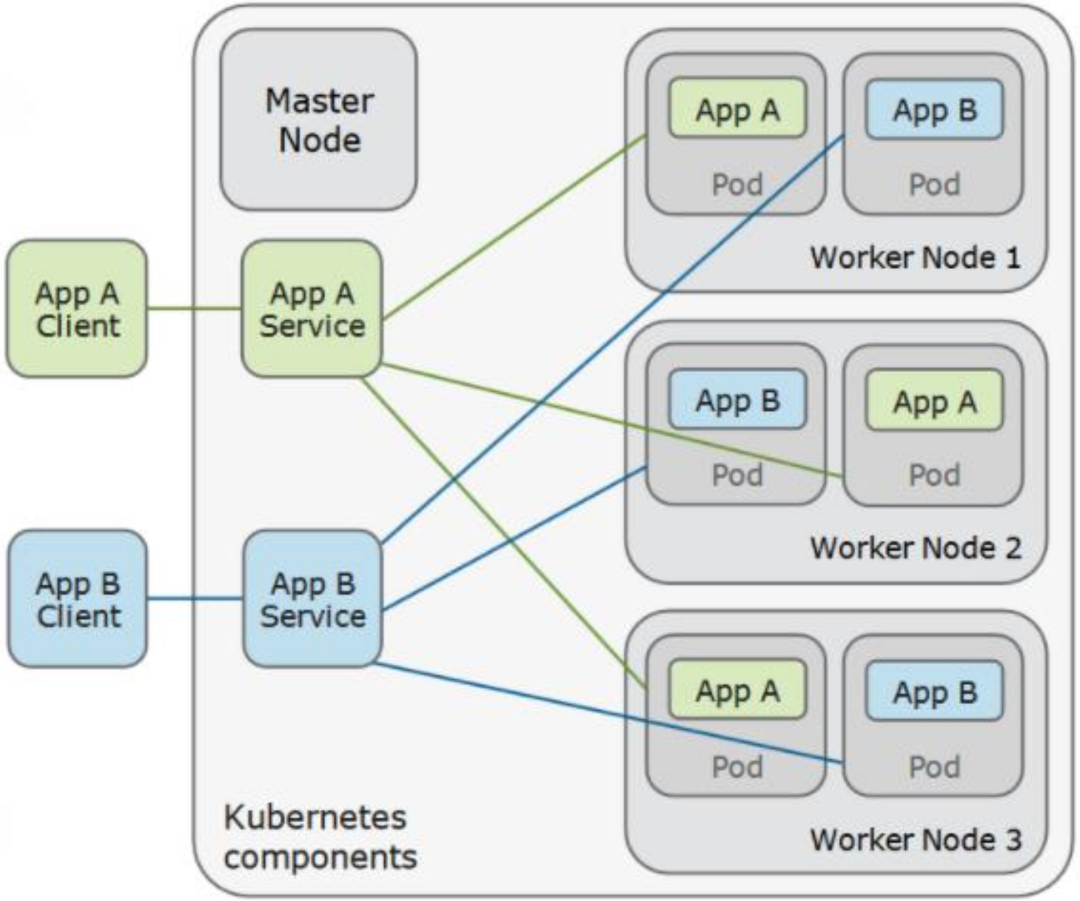
- They control the **workloads**
- Different types : CronJob, DaemonSet, **Deployment**, Jobs, **Replicaset**, Replication-Controller
- A **Deployment** controller provides declarative updates for **Pods** and **ReplicaSets**.
- A ReplicaSet is a template that describes specifically what each pod should 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. The service propagates state and networking information to all worker nodes.
  - 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
  - ExternalName: maps the service to an external name, such as abc.xyz.example.com



Labels and Selectors

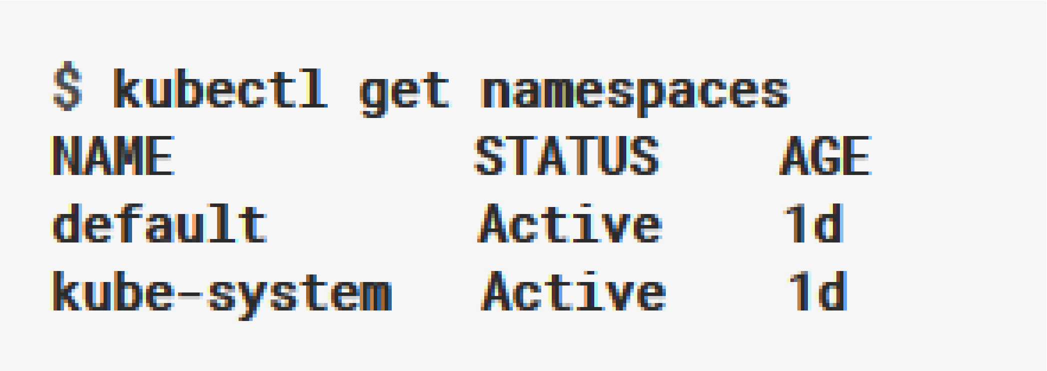
- Labels
  - **Metadata** assigned to K8s resources
  - Key-value pairs for identification
  - Critical to K8s as it relies on querying the cluster for resources that have certain labels
- **Selectors**
  - Equality (= or not =)
  - Set Based (in or not in)

```
"metadata": {
        "labels": {
                environmemnt: prod,
                app: invoice
} }
```

```
$ kubectl get pods -l
environment=prod,app=invoice
```

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. Higher level resource objects are scoped within namespaces. Low level resources (nodes, persistent volumes and namespaces themselves) are not in namespaces. 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.



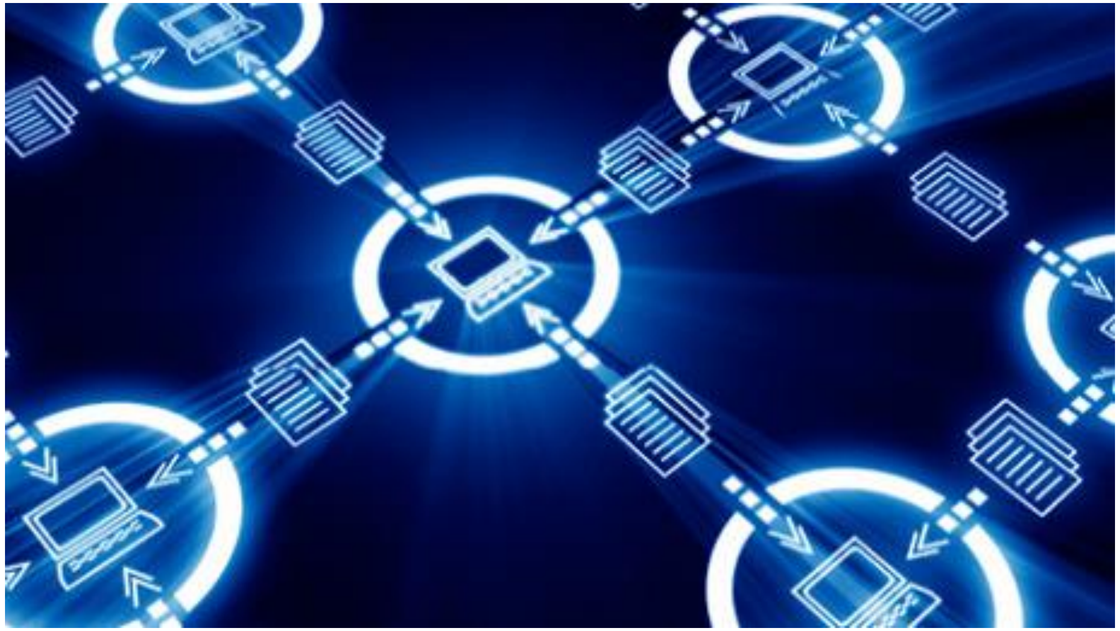
```
$ kubectl get namespaces
NAME          STATUS    AGE
default       Active    1d
kube-system   Active    1d
```

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

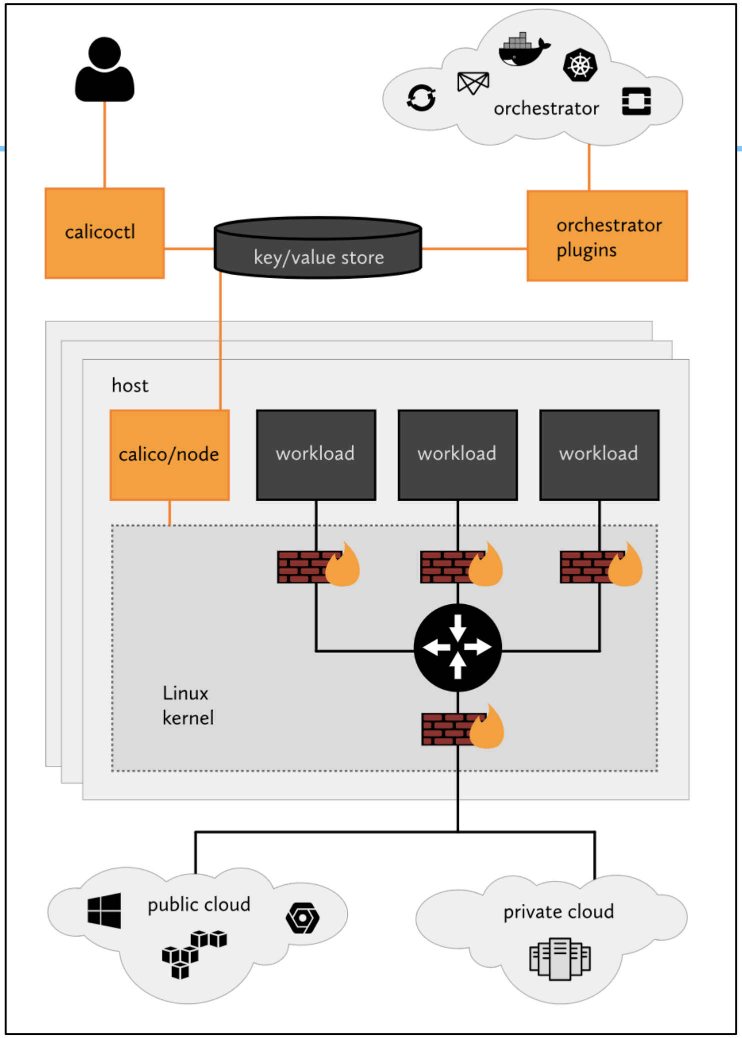
- Network policy
  - A Network policy is a specification of how groups of pods are allowed to communicate with each other and other network endpoints.
  - It uses labels to select pods and define rules which specify what traffic is allowed to the selected pods.



Calico

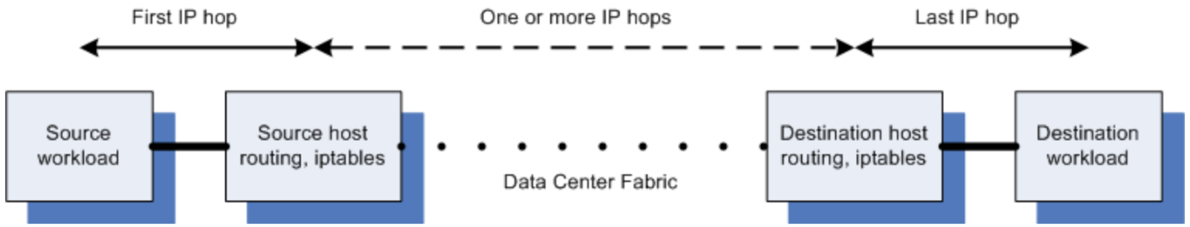
- A new approach to **virtual networking** and **network security** for **containers, VMs, and bare metal** services, that provides a rich set of security enforcement capabilities running on top of a highly scalable and efficient virtual network

- The **calico/node** Docker container runs on the Kubernetes master and each Kubernetes node in the cluster
- The **calico-cni plug-in** integrates directly with the Kubernetes kubelet process on each node to discover which pods have been created, and adds them to Calico networking
- The calico/kube-policy-controller container runs as a pod on top of Kubernetes and implements the **NetworkPolicy** API

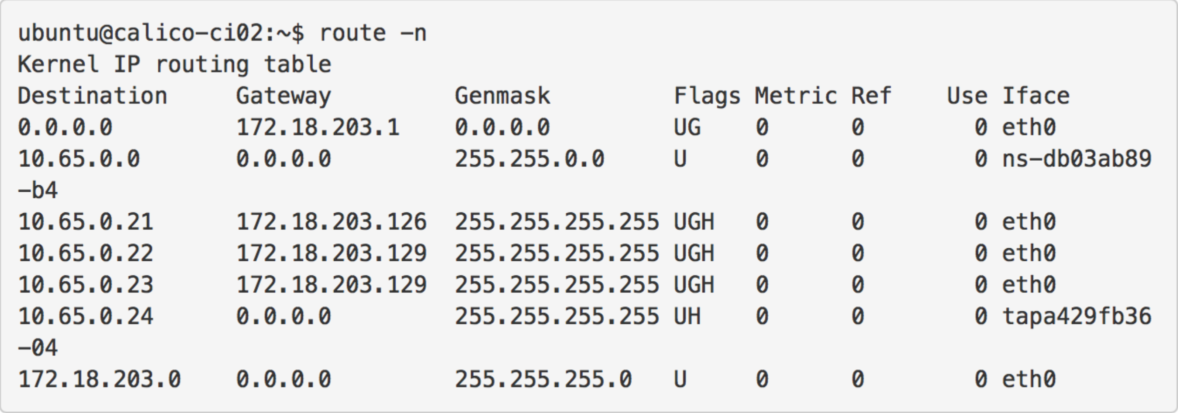


PROJECT CALICO

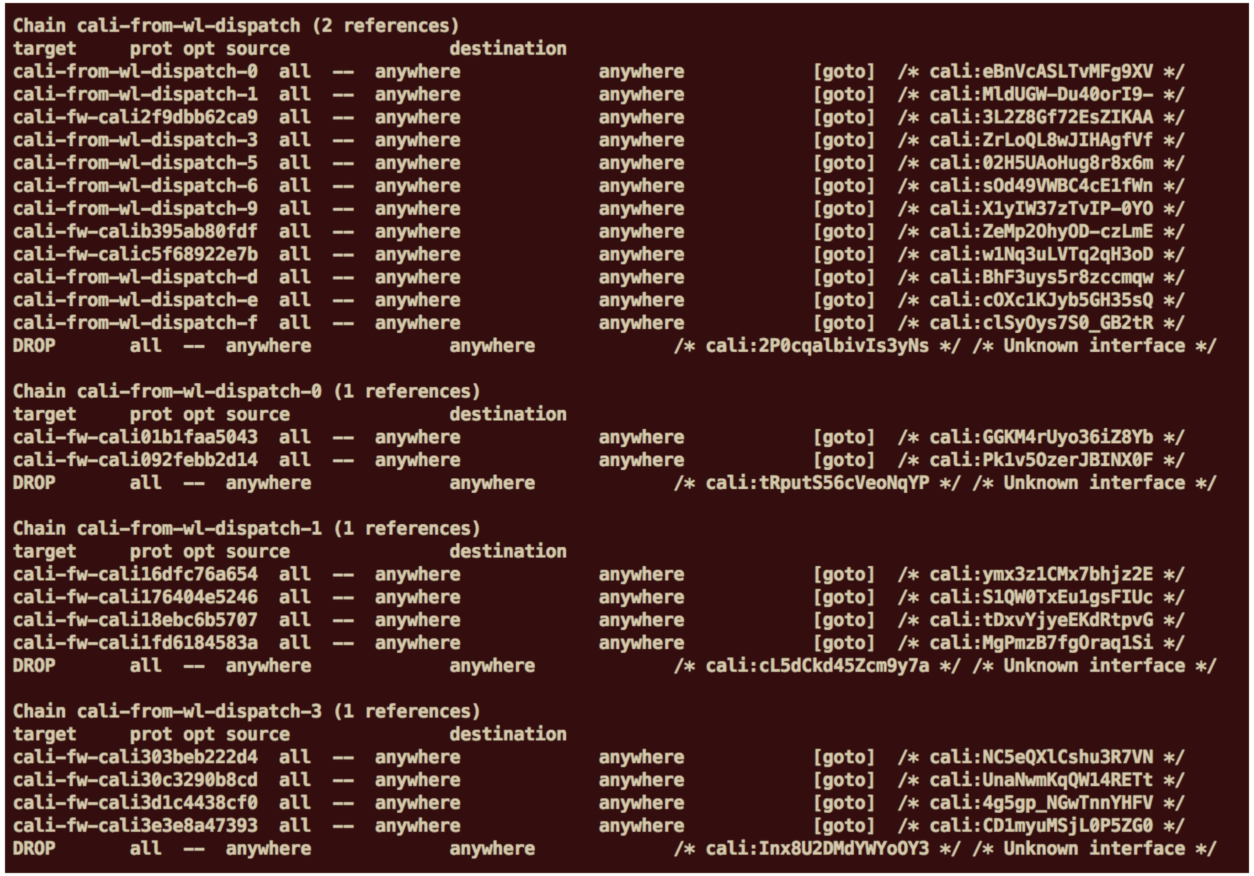
Calico Data Path: IP Routing and IPTABLES



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 */
```

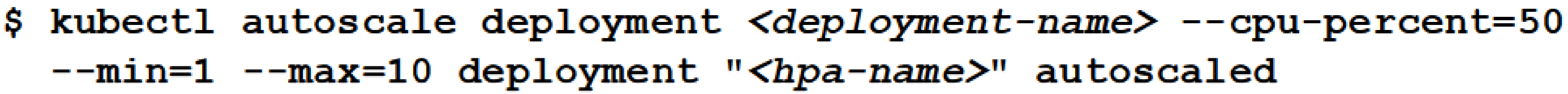
Service discovery

- 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.

Autoscaling

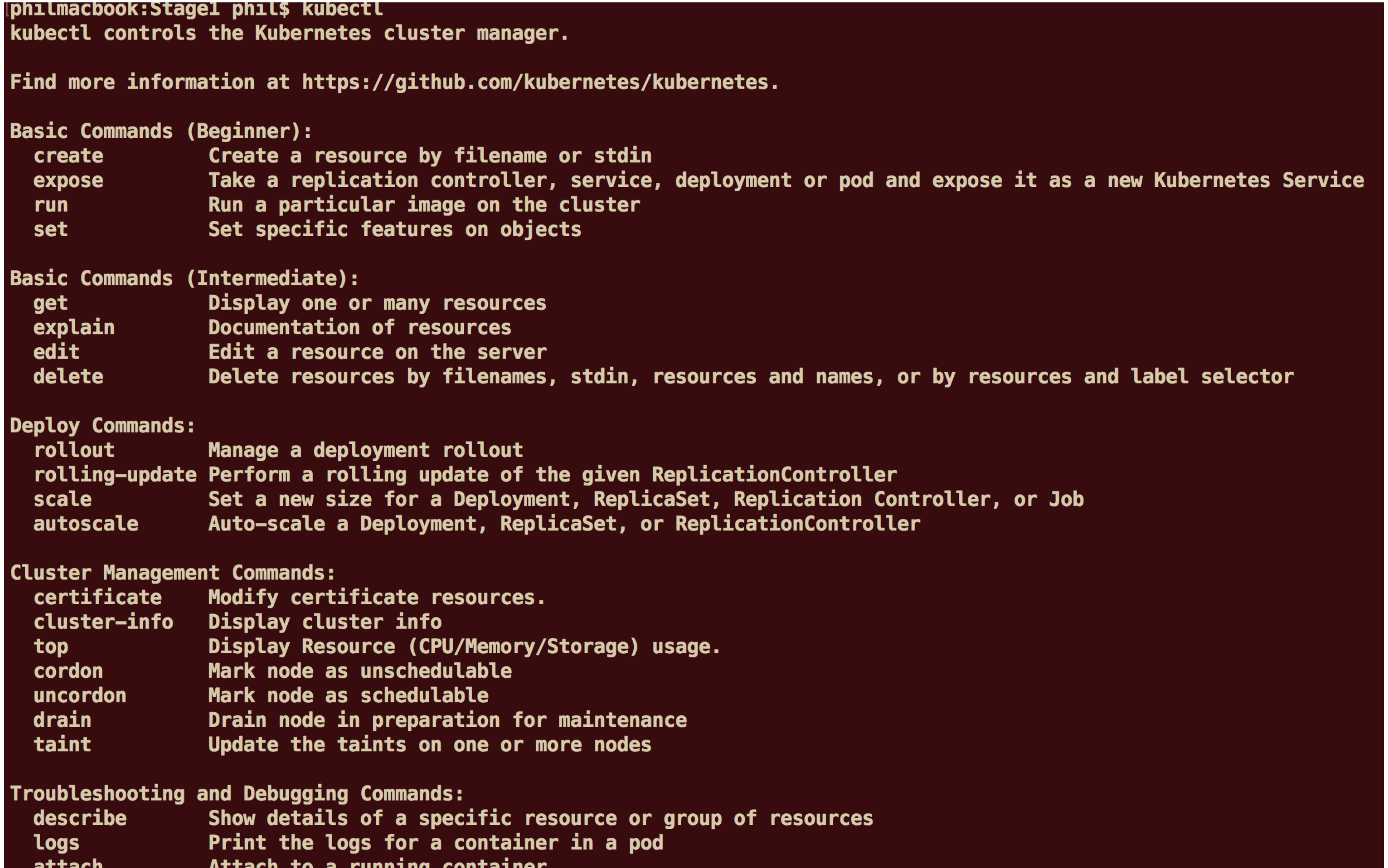
- Kubernetes implements autoscaling through Horizontal Pod Autoscaling (HPA). HPA automatically scales the number of pods in a replication controller, deployment, or replica set by matching the observed average CPU utilization to a specified target. HPA can also autoscale based on application-provided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

kubectl Commands

kubectl run hello-world-deployment --image=registry.ng.bluemix.net/production/hello-world:2



```
philmacbook:Stage1 phil$ kubectl
kubectl controls the Kubernetes cluster manager.

Find more information at https://github.com/kubernetes/kubernetes.

Basic Commands (Beginner):
  create         Create a resource by filename or stdin
  expose         Take a replication controller, service, deployment or pod and expose it as a new Kubernetes Service
  run            Run a particular image on the cluster
  set            Set specific features on objects

Basic Commands (Intermediate):
  get            Display one or many resources
  explain        Documentation of resources
  edit           Edit a resource on the server
  delete         Delete resources by filenames, stdin, resources and names, or by resources and label selector

Deploy Commands:
  rollout        Manage a deployment rollout
  rolling-update Perform a rolling update of the given ReplicationController
  scale          Set a new size for a Deployment, ReplicaSet, Replication Controller, or Job
  autoscale      Auto-scale a Deployment, ReplicaSet, or ReplicationController

Cluster Management Commands:
  certificate    Modify certificate resources.
  cluster-info   Display cluster info
  top            Display Resource (CPU/Memory/Storage) usage.
  cordon         Mark node as unschedulable
  uncordon       Mark node as schedulable
  drain          Drain node in preparation for maintenance
  taint          Update the taints on one or more nodes

Troubleshooting and Debugging Commands:
  describe       Show details of a specific resource or group of resources
  logs           Print the logs for a container in a pod
  attach         Attach to a running container
```

(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
```

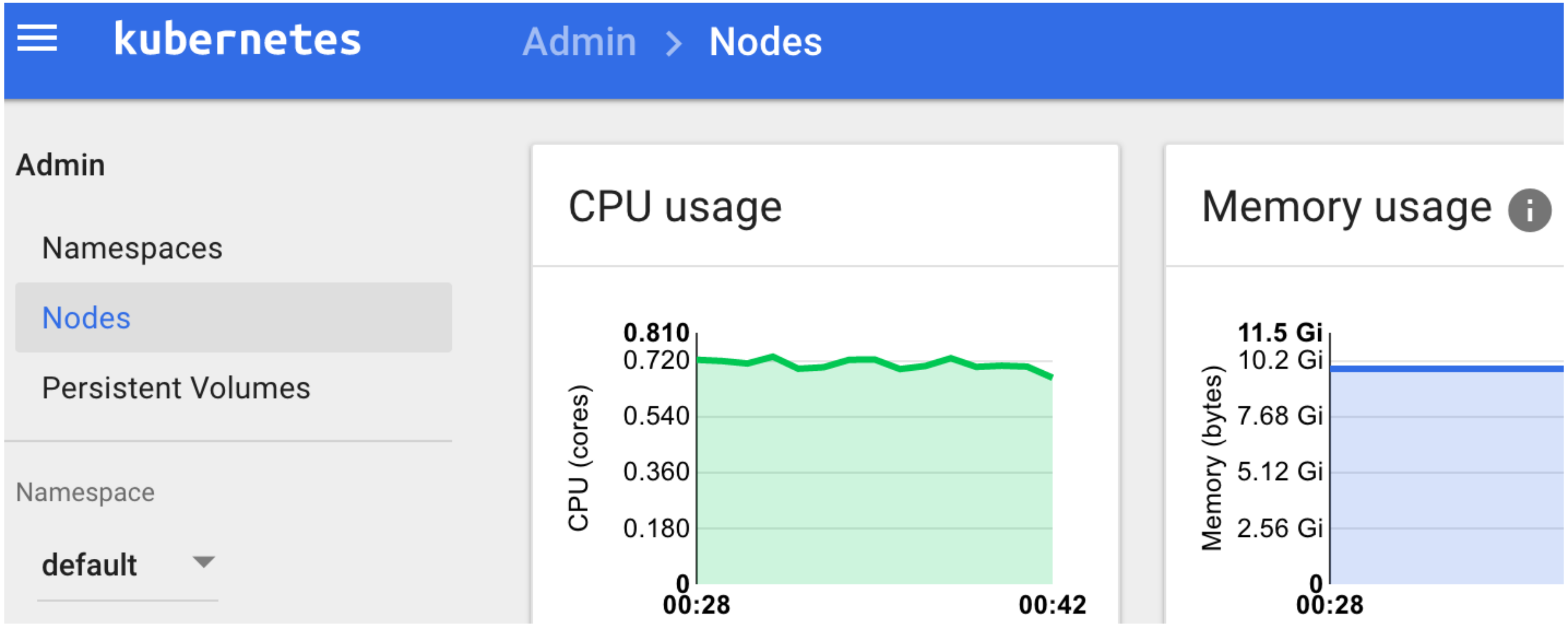
kubectl delete -f filename.yml

```
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
```

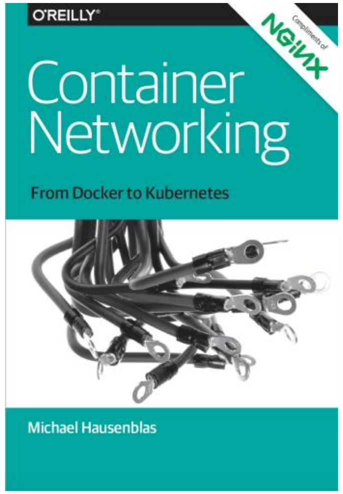
Monitoring Containers

- **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



Jonathan Baier
Getting Started with Kubernetes
Second Edition
Harness the power of Kubernetes to manage Docker deployments with ease
Packt

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

IBM Cloud Kubernetes Service

Intelligent Scheduling

Automated rollouts and rollbacks

Service discovery & load balancing

Secret & configuration management

Self-healing

Horizontal scaling

Design Your Own Cluster

Container Security & Privacy

Simplified Cluster Management

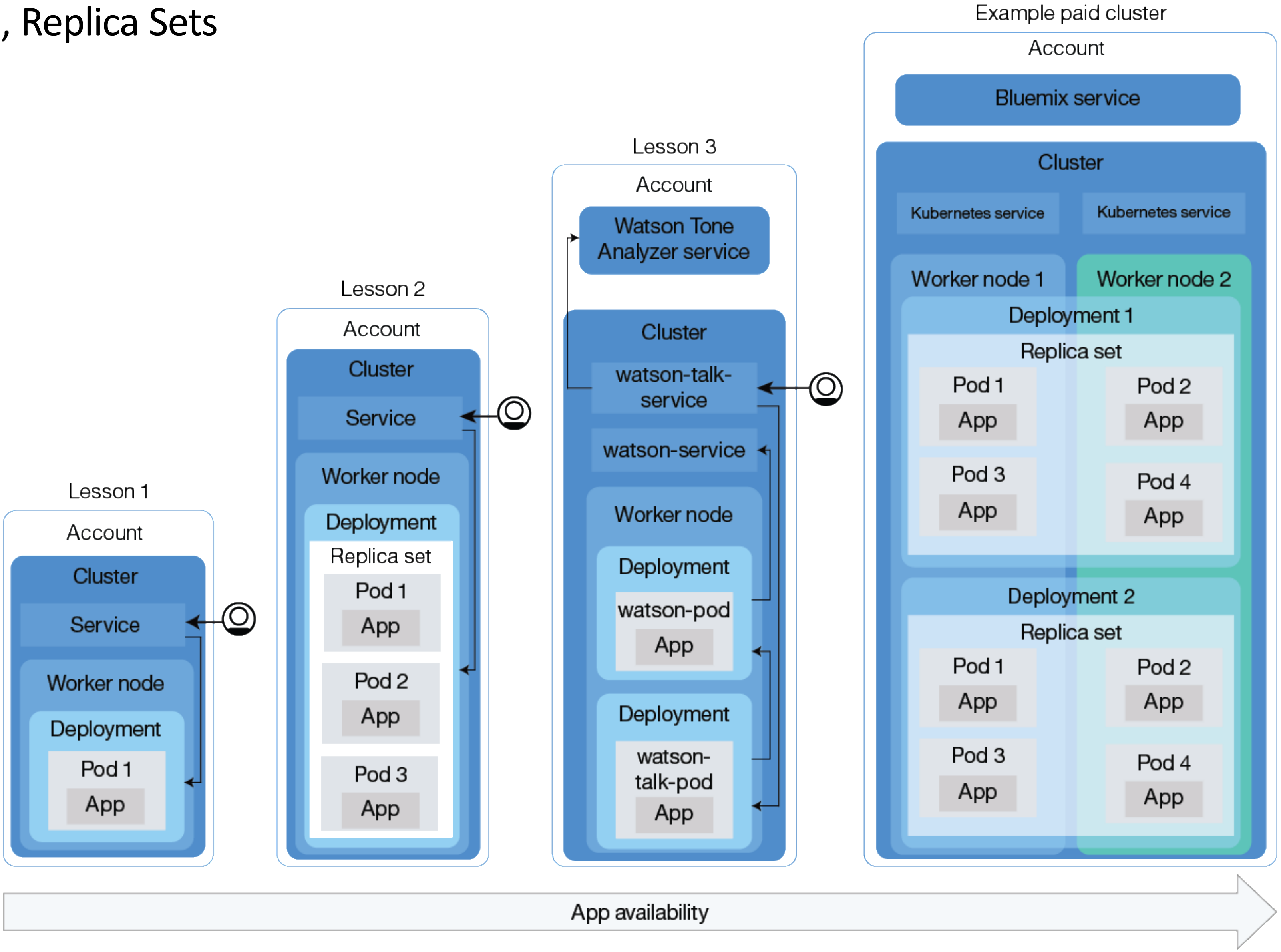
Native Kubernetes Experience

Leverages IBM Cloud & Watson

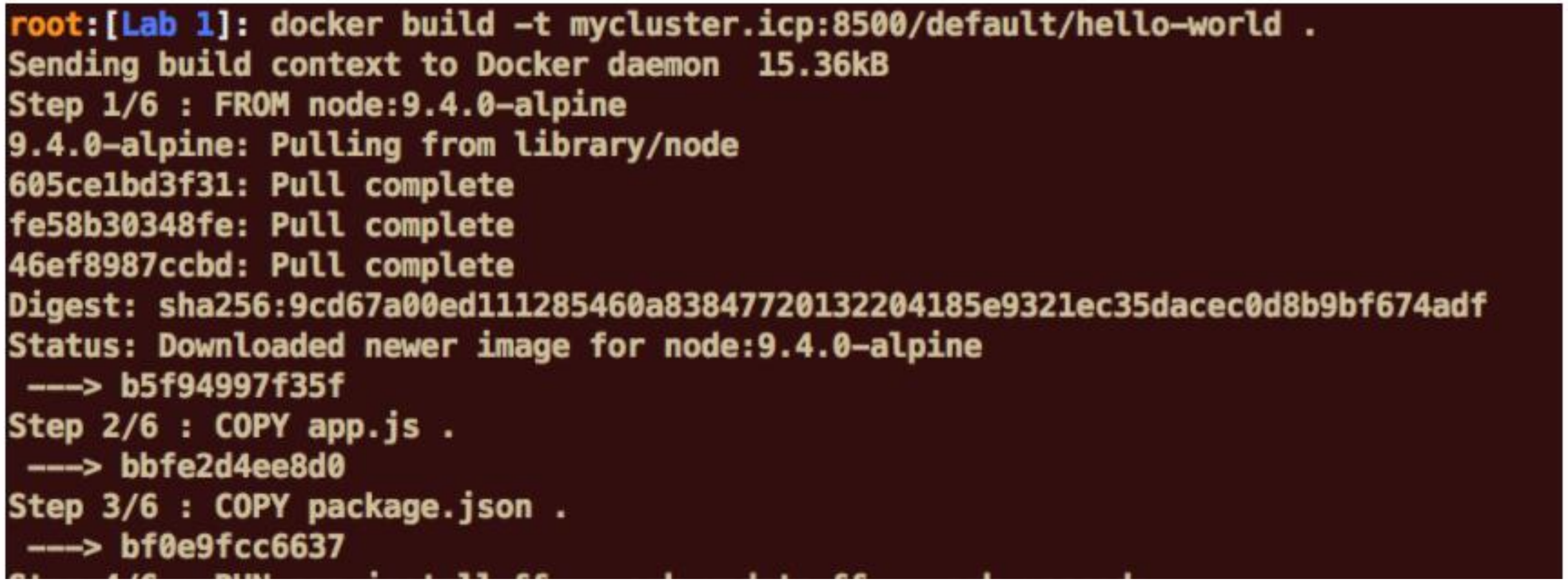
Integrated Operational Tools

Container Application Resiliency – High Availability Patterns

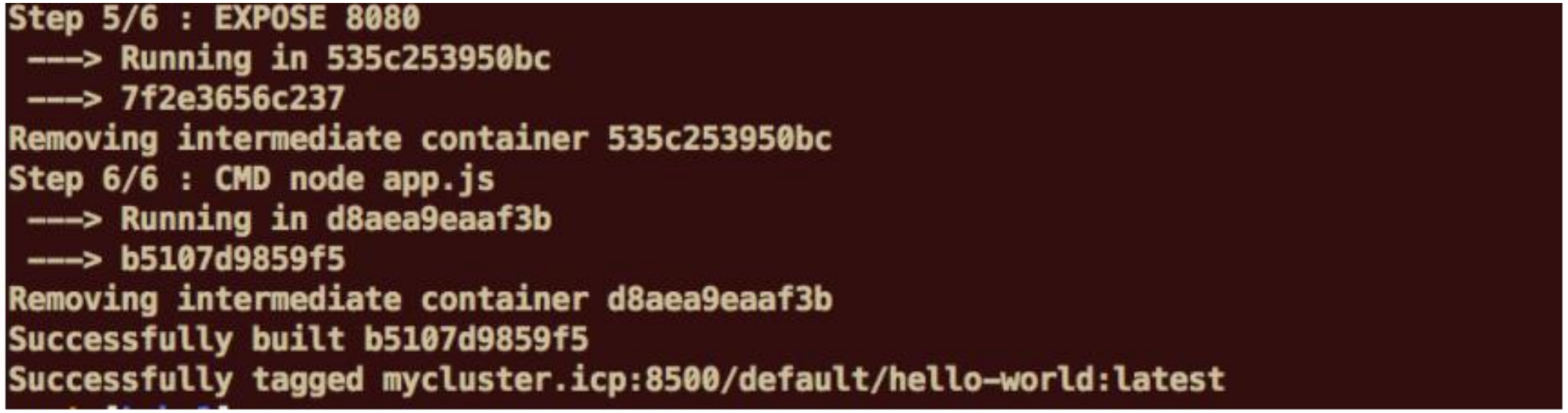
Leveraging container Pods, Replica Sets and Worker Nodes



Example : Docker Build

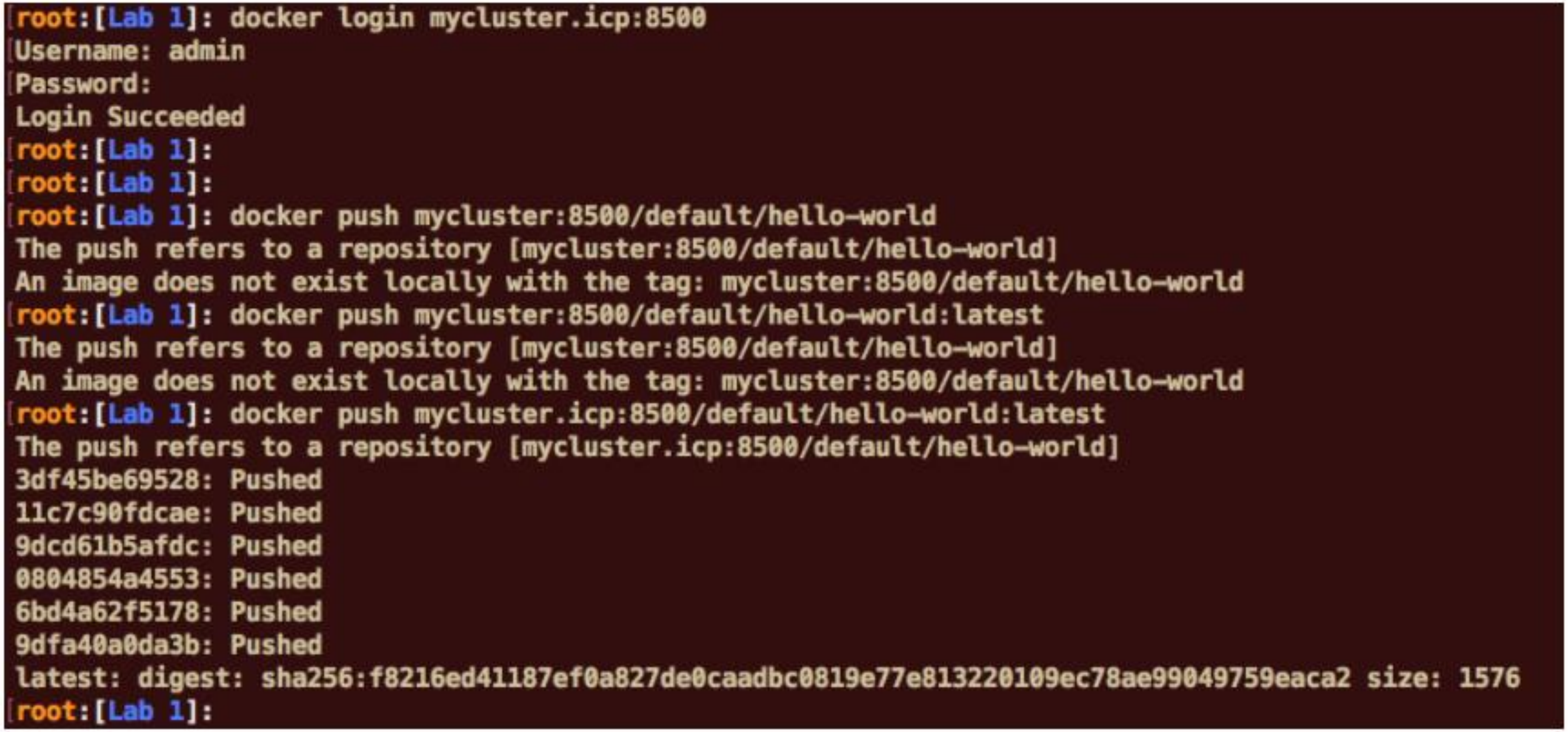


```
root:[Lab 1]: docker build -t mycluster.icp:8500/default/hello-world .
Sending build context to Docker daemon  15.36kB
Step 1/6 : FROM node:9.4.0-alpine
9.4.0-alpine: Pulling from library/node
605ce1bd3f31: Pull complete
fe58b30348fe: Pull complete
46ef8987ccbd: Pull complete
Digest: sha256:9cd67a00ed111285460a83847720132204185e9321ec35dacec0d8b9bf674adf
Status: Downloaded newer image for node:9.4.0-alpine
 ---> b5f94997f35f
Step 2/6 : COPY app.js .
 ---> bbfe2d4ee8d0
Step 3/6 : COPY package.json .
 ---> bf0e9fcc6637
```



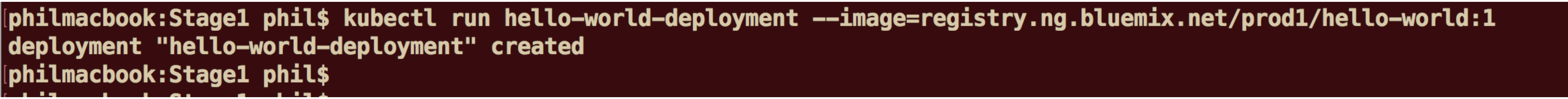
```
Step 5/6 : EXPOSE 8080
 ---> Running in 535c253950bc
 ---> 7f2e3656c237
Removing intermediate container 535c253950bc
Step 6/6 : CMD node app.js
 ---> Running in d8aea9eaaf3b
 ---> b5107d9859f5
Removing intermediate container d8aea9eaaf3b
Successfully built b5107d9859f5
Successfully tagged mycluster.icp:8500/default/hello-world:latest
```

Example : Docker Push

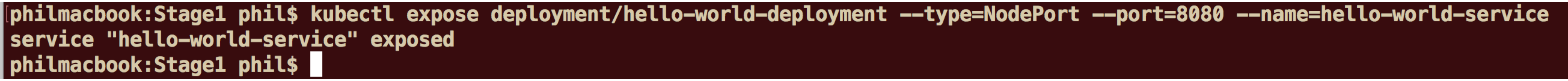


```
[root:[Lab 1]: docker login mycluster.icp:8500
[Username: admin
[Password:
 Login Succeeded
[root:[Lab 1]:
[root:[Lab 1]:
[root:[Lab 1]: docker push mycluster:8500/default/hello-world
 The push refers to a repository [mycluster:8500/default/hello-world]
 An image does not exist locally with the tag: mycluster:8500/default/hello-world
[root:[Lab 1]: docker push mycluster:8500/default/hello-world:latest
 The push refers to a repository [mycluster:8500/default/hello-world]
 An image does not exist locally with the tag: mycluster:8500/default/hello-world
[root:[Lab 1]: docker push mycluster.icp:8500/default/hello-world:latest
 The push refers to a repository [mycluster.icp:8500/default/hello-world]
 3df45be69528: Pushed
 11c7c90fdcae: Pushed
 9dcd61b5afdc: Pushed
 0804854a4553: Pushed
 6bd4a62f5178: Pushed
 9dfa40a0da3b: Pushed
 latest: digest: sha256:f8216ed41187ef0a827de0caadbc0819e77e813220109ec78ae99049759eaca2 size: 1576
[root:[Lab 1]:
```

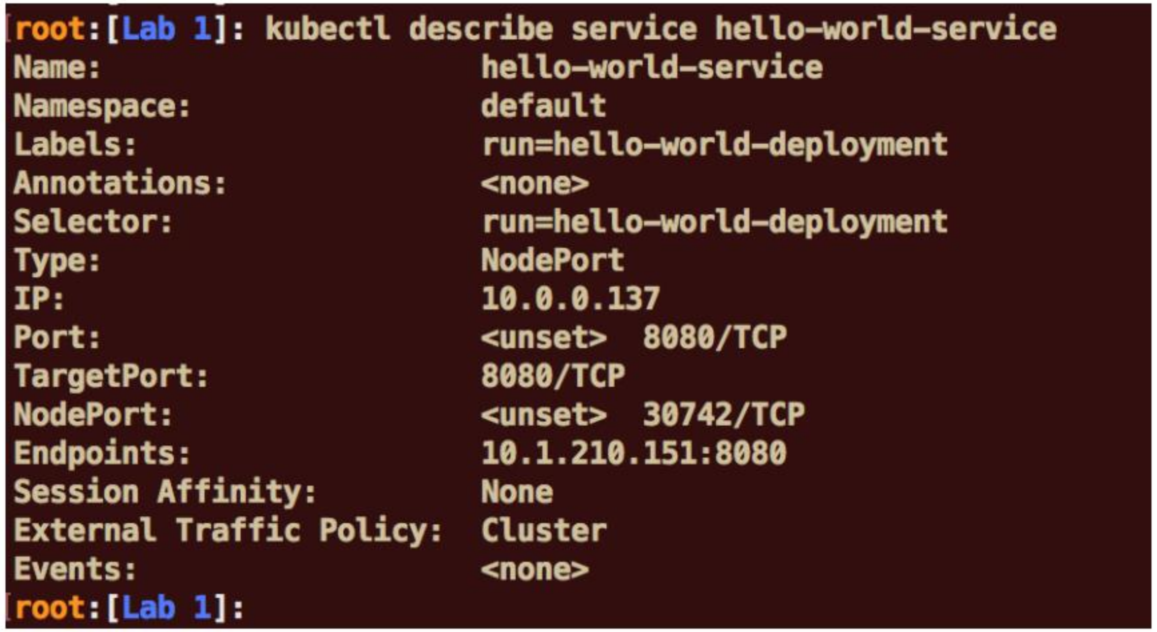
Example : Kubernetes Deployment, Expose, Describe



```
philmacbook:Stage1 phil$ kubectl run hello-world-deployment --image=registry.ng.bluemix.net/prod1/hello-world:1
deployment "hello-world-deployment" created
philmacbook:Stage1 phil$
```

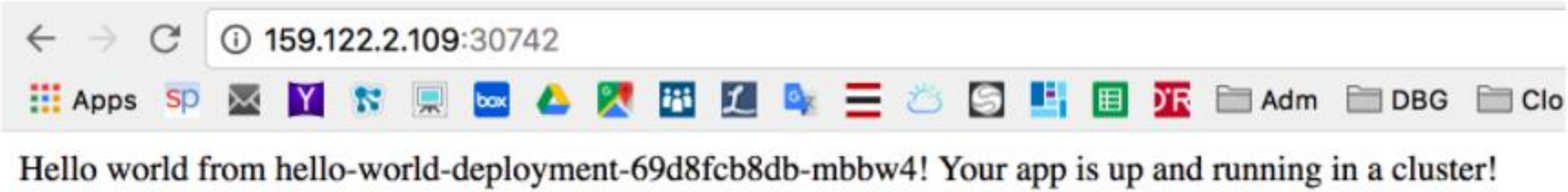


```
philmacbook:Stage1 phil$ kubectl expose deployment/hello-world-deployment --type=NodePort --port=8080 --name=hello-world-service
service "hello-world-service" exposed
philmacbook:Stage1 phil$
```



```
root:[Lab 1]: kubectl describe service hello-world-service
Name:                      hello-world-service
Namespace:                 default
Labels:                    run=hello-world-deployment
Annotations:               <none>
Selector:                  run=hello-world-deployment
Type:                      NodePort
IP:                        10.0.0.137
Port:                      <unset>  8080/TCP
TargetPort:                8080/TCP
NodePort:                  <unset>  30742/TCP
Endpoints:                 10.1.210.151:8080
Session Affinity:          None
External Traffic Policy:   Cluster
Events:                    <none>
root:[Lab 1]:
```

Example : Access to the app



Hello world from hello-world-deployment-69d8fcb8db-mbbw4! Your app is up and running in a cluster!