Kubernetes Deep Dive

A FRAMEWORK FOR DATA INTENSIVE COMPUTING

Agenda

Intro / Prep Environments

Day 1: Docker Deep Dive

Day 2: Kubernetes Deep Dive

Docker Recap

- Linux containers
- Docker API
- Images
- Containers
- Registry

Docker Recap

- ❑ Containers run on **single** Docker host
- ❑ Containers are **ephemeral**
- ❑ Nothing watchdogs the containers
- ❑ Containers can have external persistence
- ❑ Containers do not contain
- ❑ Operating system **matters**

Why Docker Matters

- Application distribution
- Dependency management
- Application density
- Reduced management overhead in terms of VMs
- On premise, hybrid, public cloud

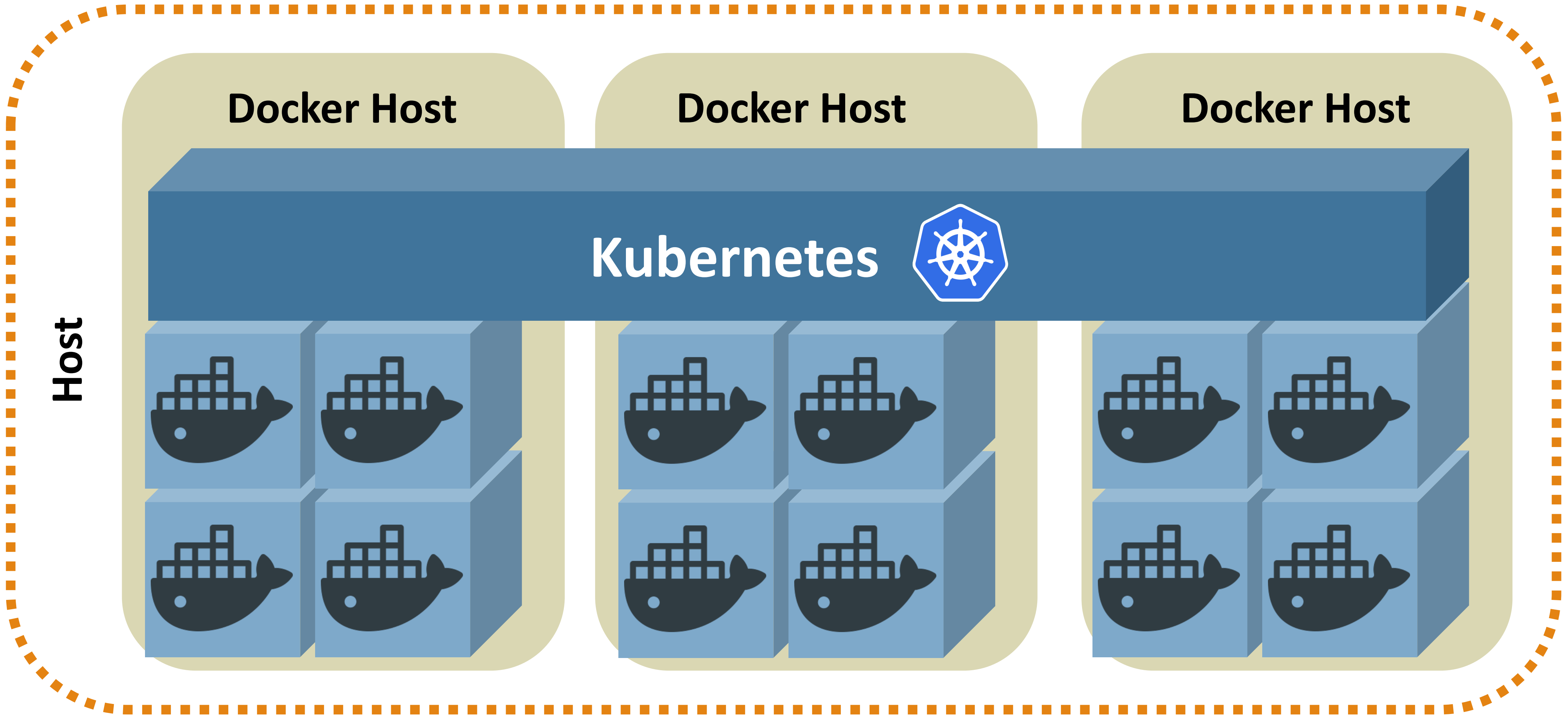
Why you Win with Docker-Based Solutions

- Immutable infrastructure
- DevOps
- CI/CD

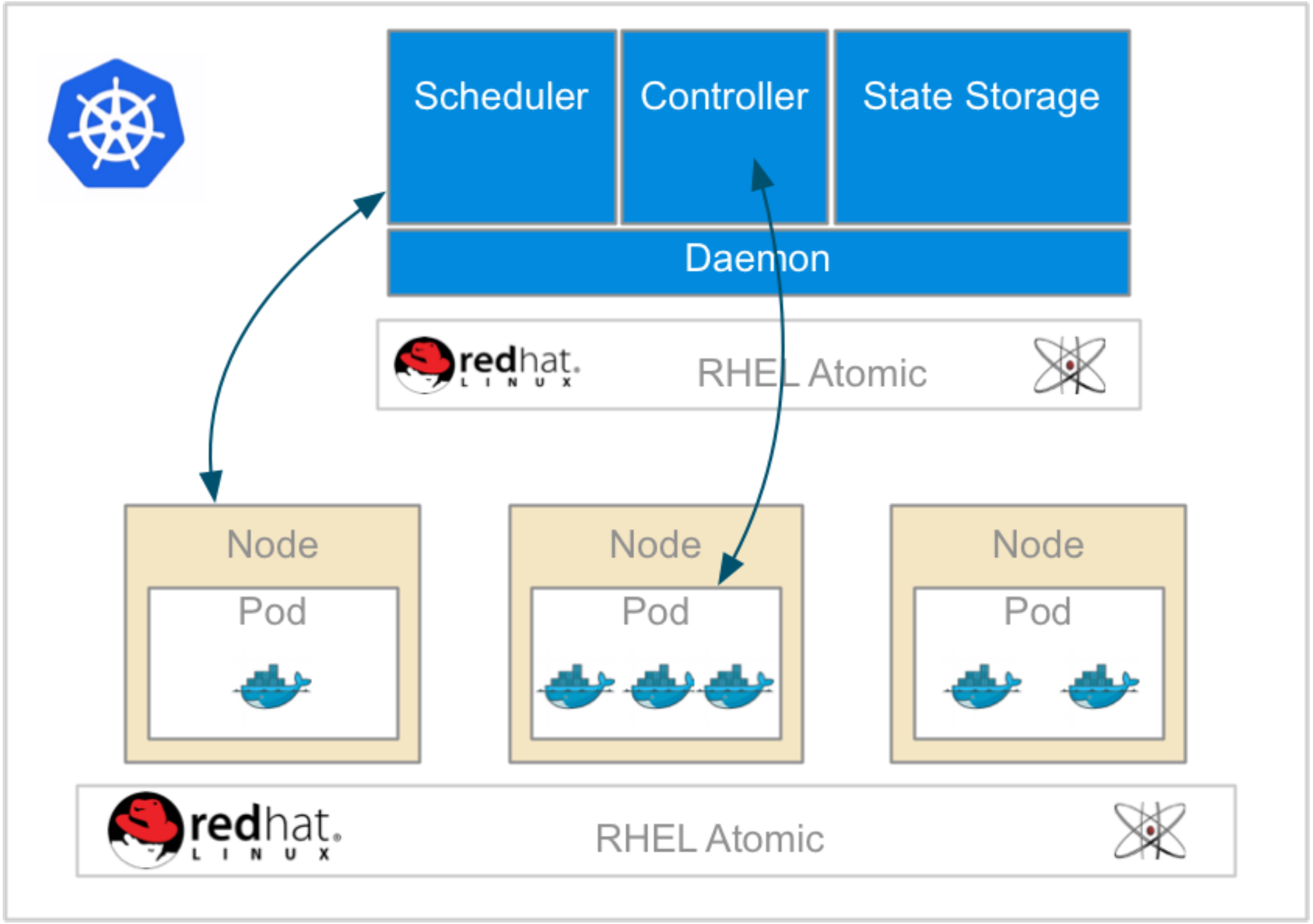
Enter Kubernetes

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

Simple Kubernetes Architecture



Overall Kubernetes



Containerize all the Things

- **Everything** at Google runs in containers!
- Everything means, **Gmail**, search, Google Maps
- With over 2 billion containers **a week**
- And of course, Google **Cloud** Services

What is a Kube?

- ❑ "Helmsman of a Ship"
- ❑ Containers at Google
- ❑ Large-scale cluster management at Google with Borg
- ❑ Open source 6/2014
- ❑ GKE…
- ❑ Red Hat - a top contributor
- ❑ Written in **golang** completely
- ❑ Do we need this?

Kube is Kubernetes

- ❑ Different way to look at managing instances: scale
- ❑ Design for failure
- ❑ Efficient / Lean/ Simple
- ❑ Portability
- ❑ Extensible
- ❑ How to place containers on a cluster
- ❑ Smart placement
- ❑ How to interact with a system that does placement

What Does Kubernetes Do?

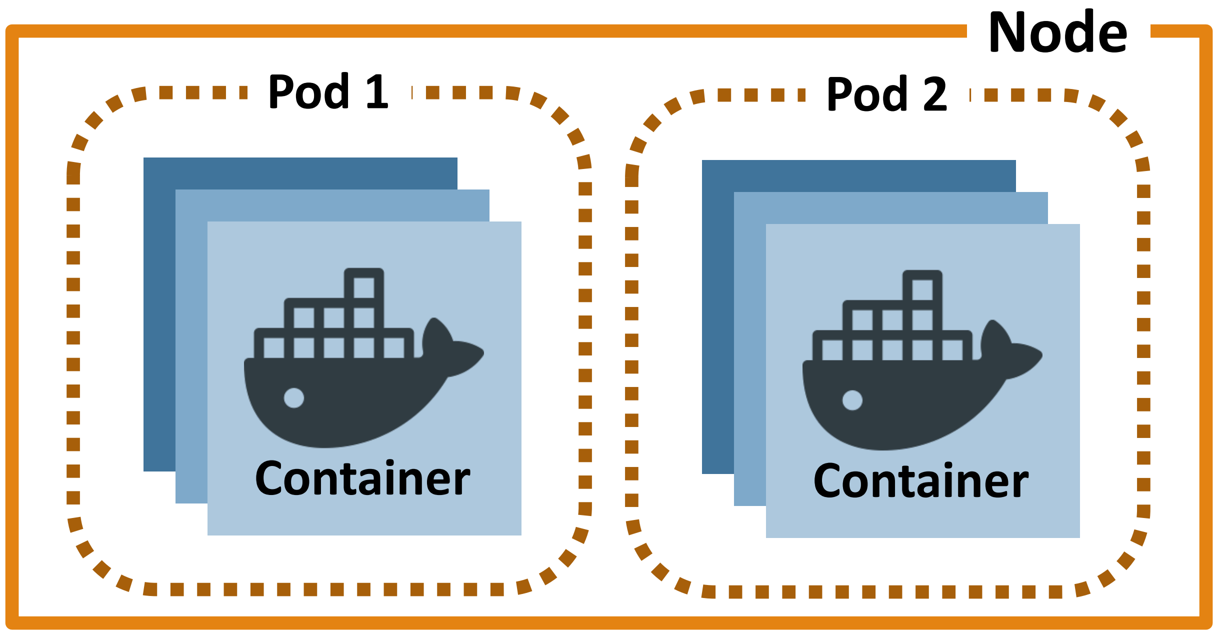
- Different than configuration management
- Immutable infrastructure principles
- What to do when containers fail?
- Containers will fail
- Cluster security authZ/authN
- Scaling
- Grouping/Aggregates

Why is it Important

- Managing containers by hand is harder than VMs: It won’t scale
- Automate the boilerplate stuff
- Runbooks → Scripts → Config management → Scale
- Decouple application from machine
- Applications run on resources
- Kubernetes manages this interaction of applications and resources
- Manage applications, not machines
- What about legacy apps?

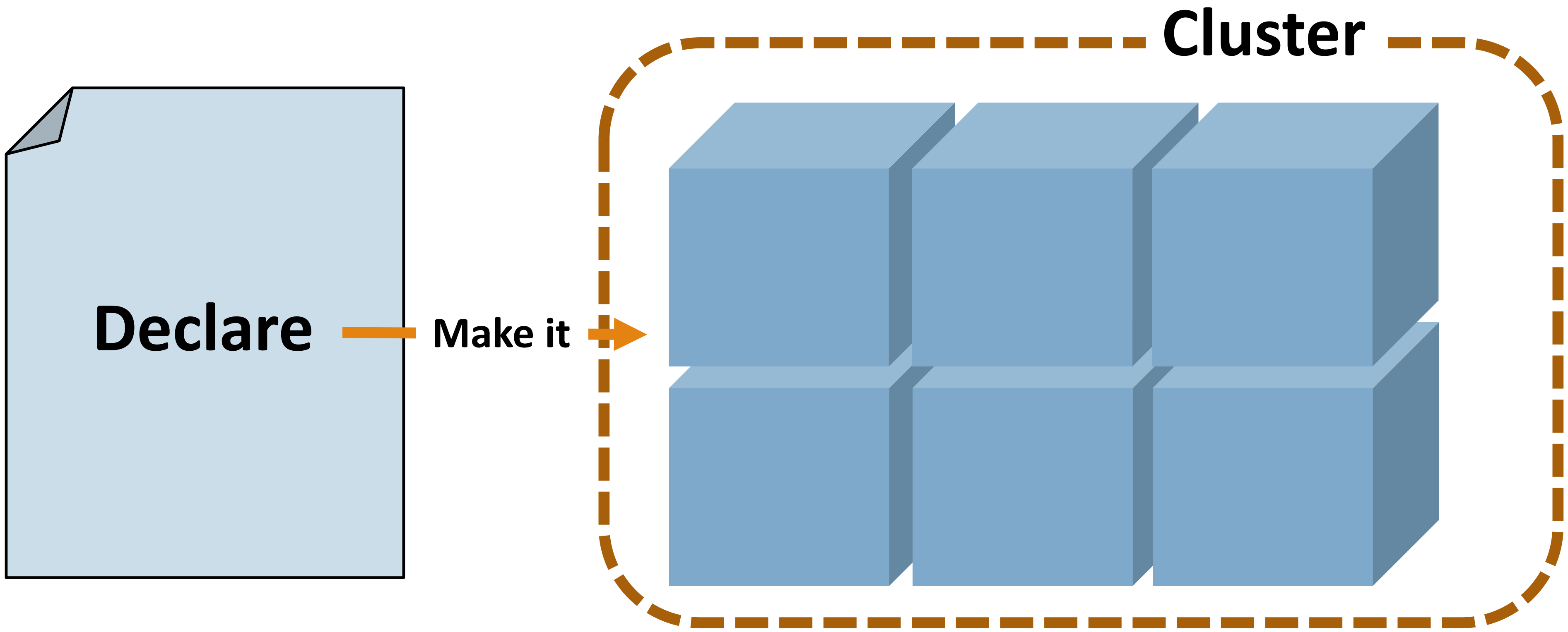
Kubernetes Core Concepts

- ❑ Simplicity, Simplicity, Simplicity
- ❑ Pods
- ❑ Labels / Selectors
- ❑ Replication Controllers
- ❑ Services



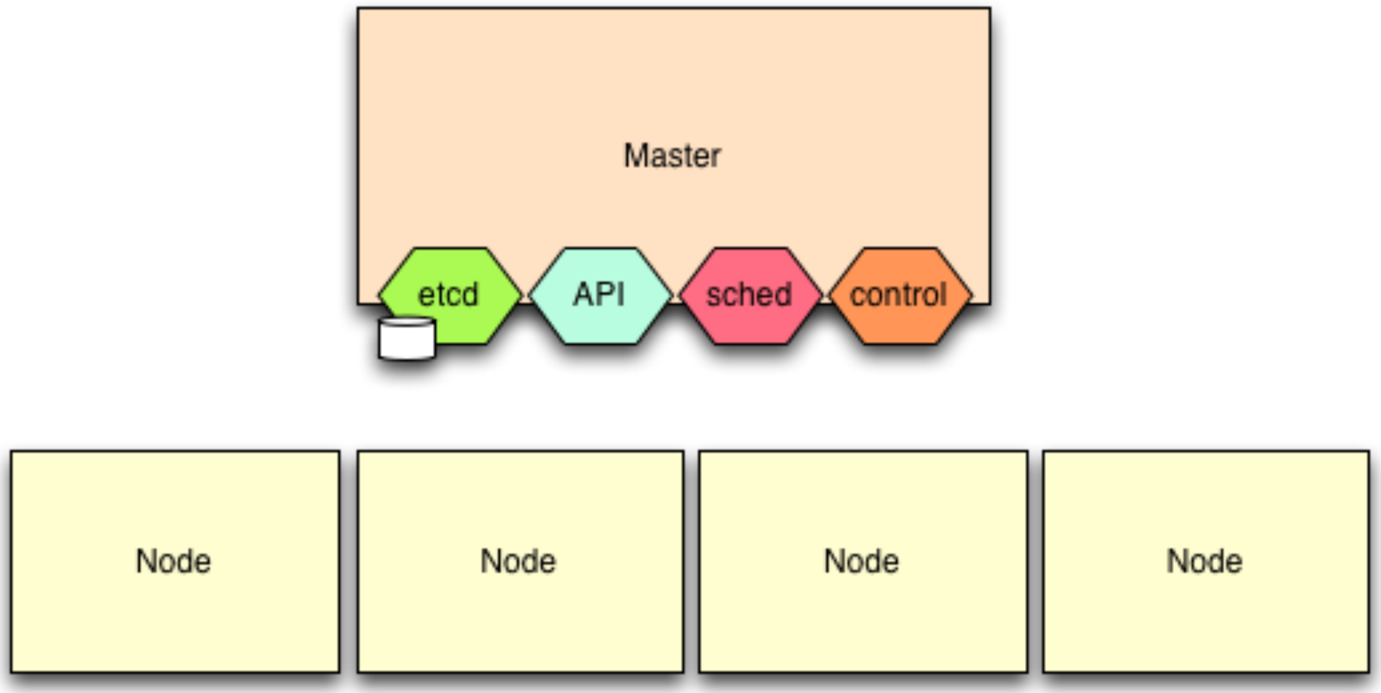
See the API at `http://kubernetes.io/third_party/swagger-ui/`

Reconciliation of End State



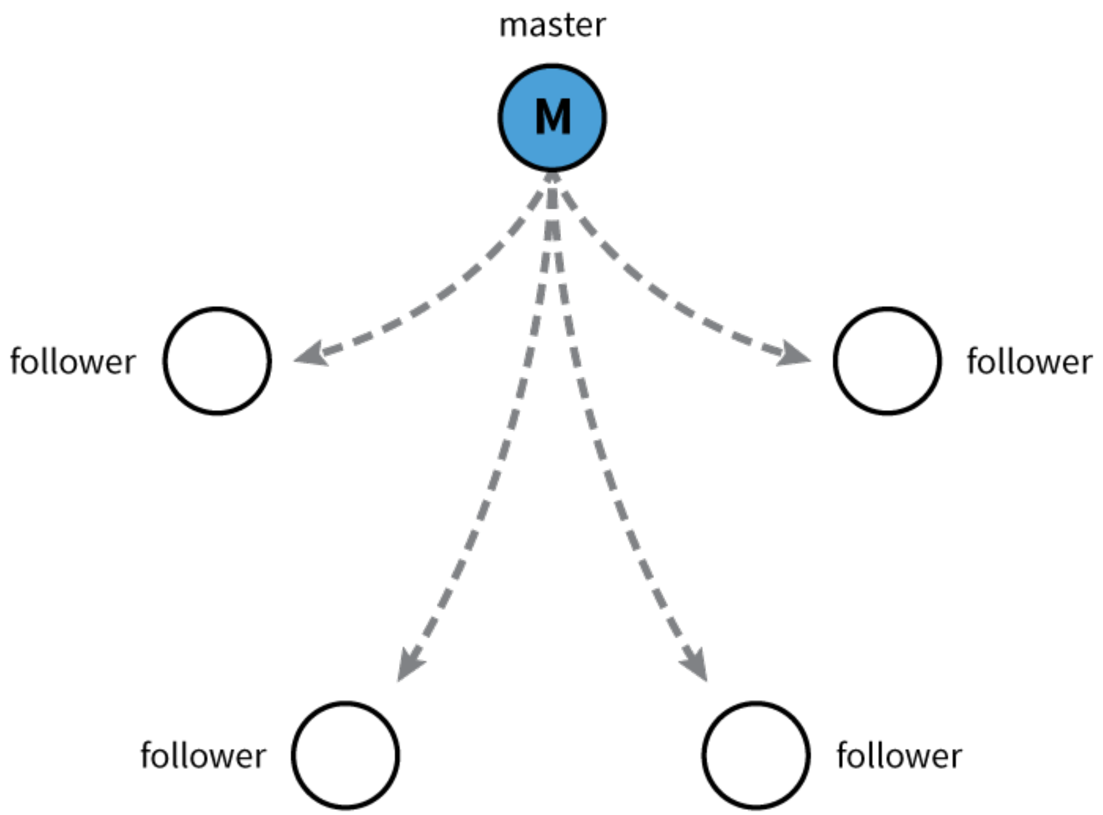
Kubernetes Control Plane

- Controller manager
- `etcd`
- API Server
- Scheduler



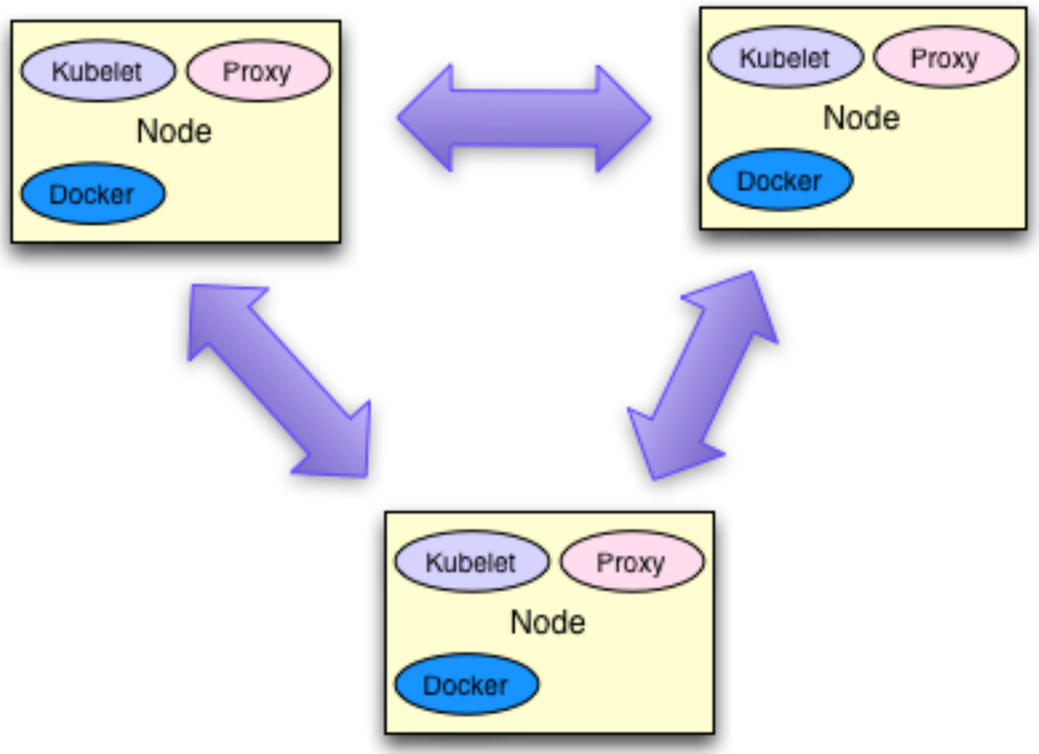
etcd

- Open source project started at CoreOS
- Distributed database
- CAP Theorem? == CP
- Raft algorithm/protocol
- Watchable
- `etcd` provides HA data store

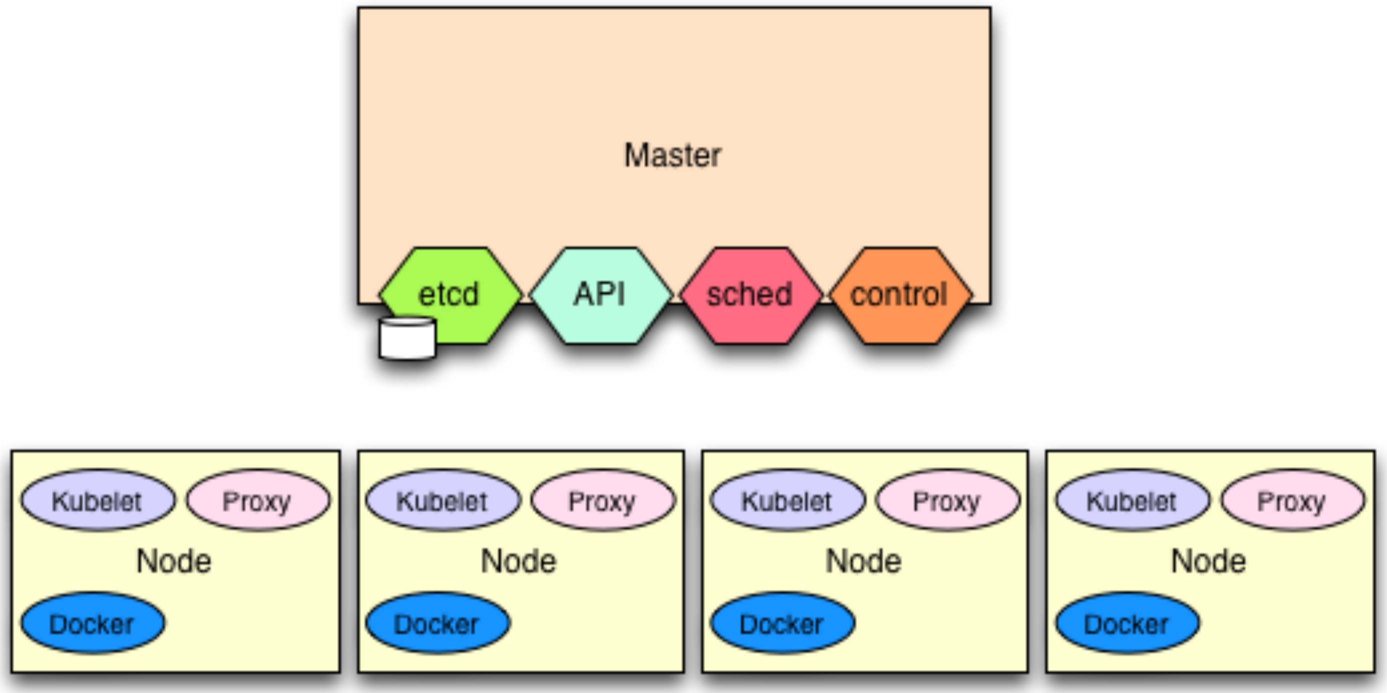


Kubernetes Nodes

- Nodes are VMs or physical hosts
- Nodes need connectivity between them
- Ideally same network, data center, and availability zone



Kubernetes Nodes



Kubernetes Nodes

- Kubelet
  - Watches for pods to be assigned to node
  - Mount volumes
  - Install secrets
  - Runs the pod (via Docker)
  - Reports pod status / node status
- Kube-Proxy
  - Connection forwarding
  - Kube services
- Docker

Cluster Add-Ons

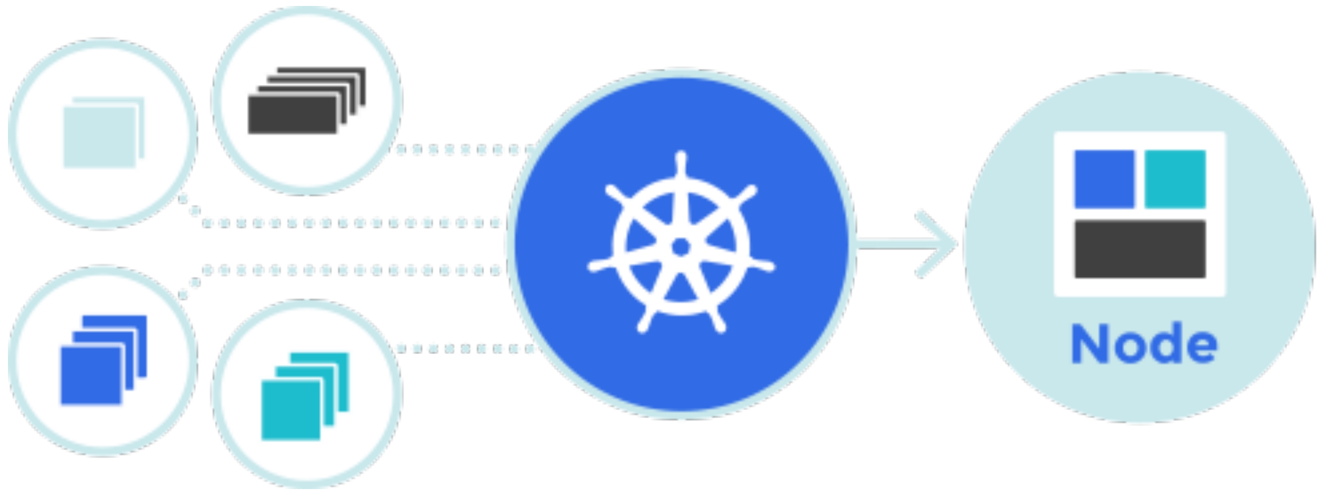
- ❑ Monitoring
- ❑ DNS
- ❑ UI
- ❑ Logging

DEMO

- Guestbook Demo
- https://github.com/kubernetes/kubernetes/blob/release-1.0/examples/guestbook/README.md

Set up Kubernetes

- ❑ Lab prerequisites in place – if any
- ❑ Verify you have most recent version of Oracle‘s VirtualBox
- ❑ Install Kubernetes
- ❑ Understand the architecture
- ❑ Smoke test



Final Output

```
Waiting for each minion to be registered with cloud provider
Validating we can run kubectl commands.
NAME      READY     STATUS    RESTARTS   AGE
Connection to 127.0.0.1 closed.
Kubernetes cluster is running.  The master is running at:
https://10.245.1.2
The user name and password to use is located in ~/.kubernetes_vagrant_auth.
calling validate-cluster
Found 1 nodes.
        NAME         LABELS                              STATUS
     1  10.245.1.3   kubernetes.io/hostname=10.245.1.3   Ready
Validate output:
NAME                 STATUS    MESSAGE              ERROR
controller-manager   Healthy   ok                   nil
scheduler            Healthy   ok                   nil
etcd-0               Healthy   {"health": "true"}   nil
Cluster validation succeeded
Done, listing cluster services:
Kubernetes master is running at https://10.245.1.2
KubeDNS is running at https://10.245.1.2/api/v1/proxy/namespaces/kube-system/services/kube-
dns
KubeUI is running at https://10.245.1.2/api/v1/proxy/namespaces/kube-system/services/kube-ui
```

Kubernetes Deep Dive

LET'S DIVE DEEPER NOW THAT WE KNOW THAT BASICS

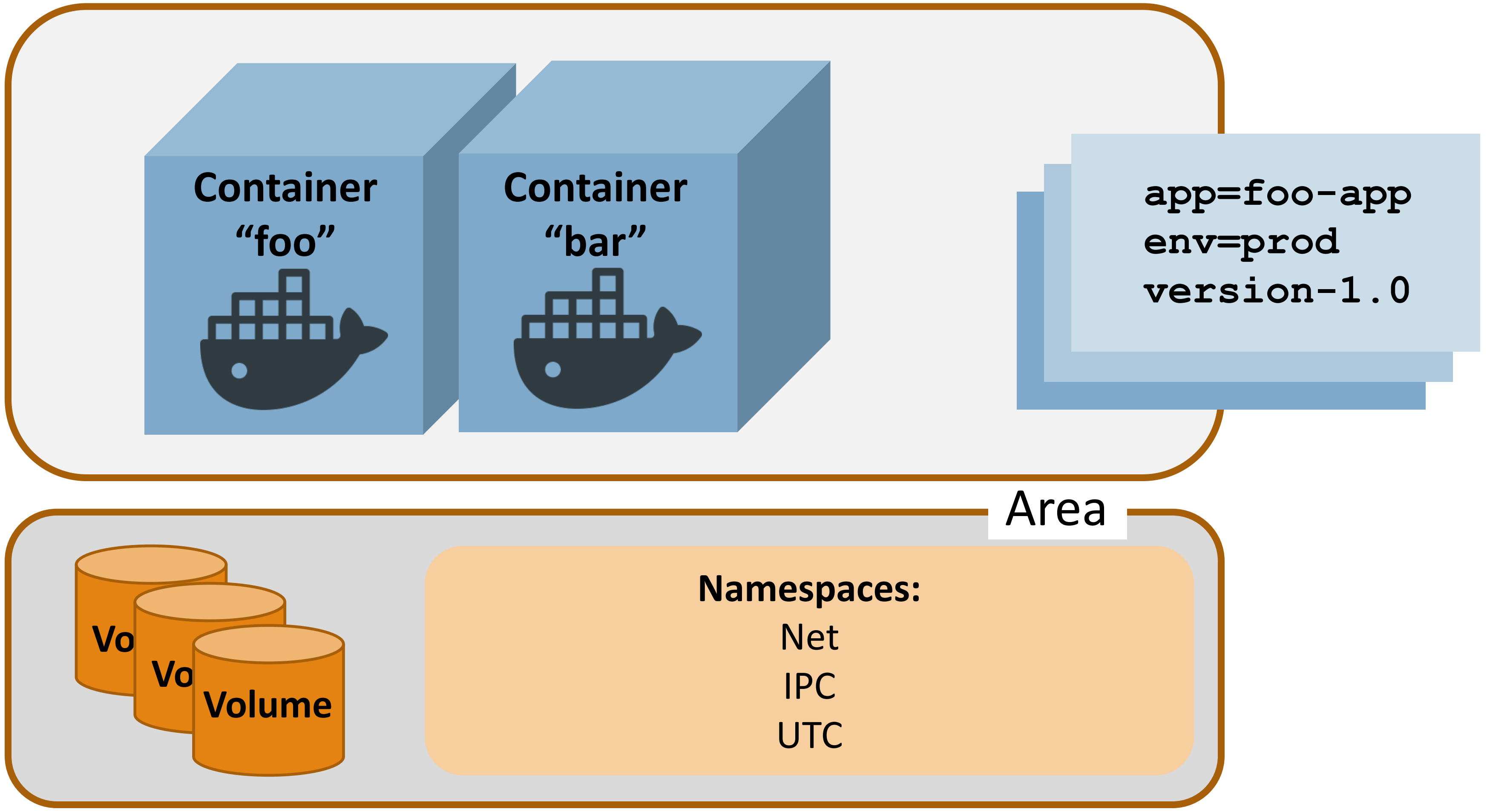
Kubernetes Core Concepts

- ❑ Pods
- ❑ Labels/Selectors
- ❑ Replication Controllers
- ❑ Services

Kubernetes Pods

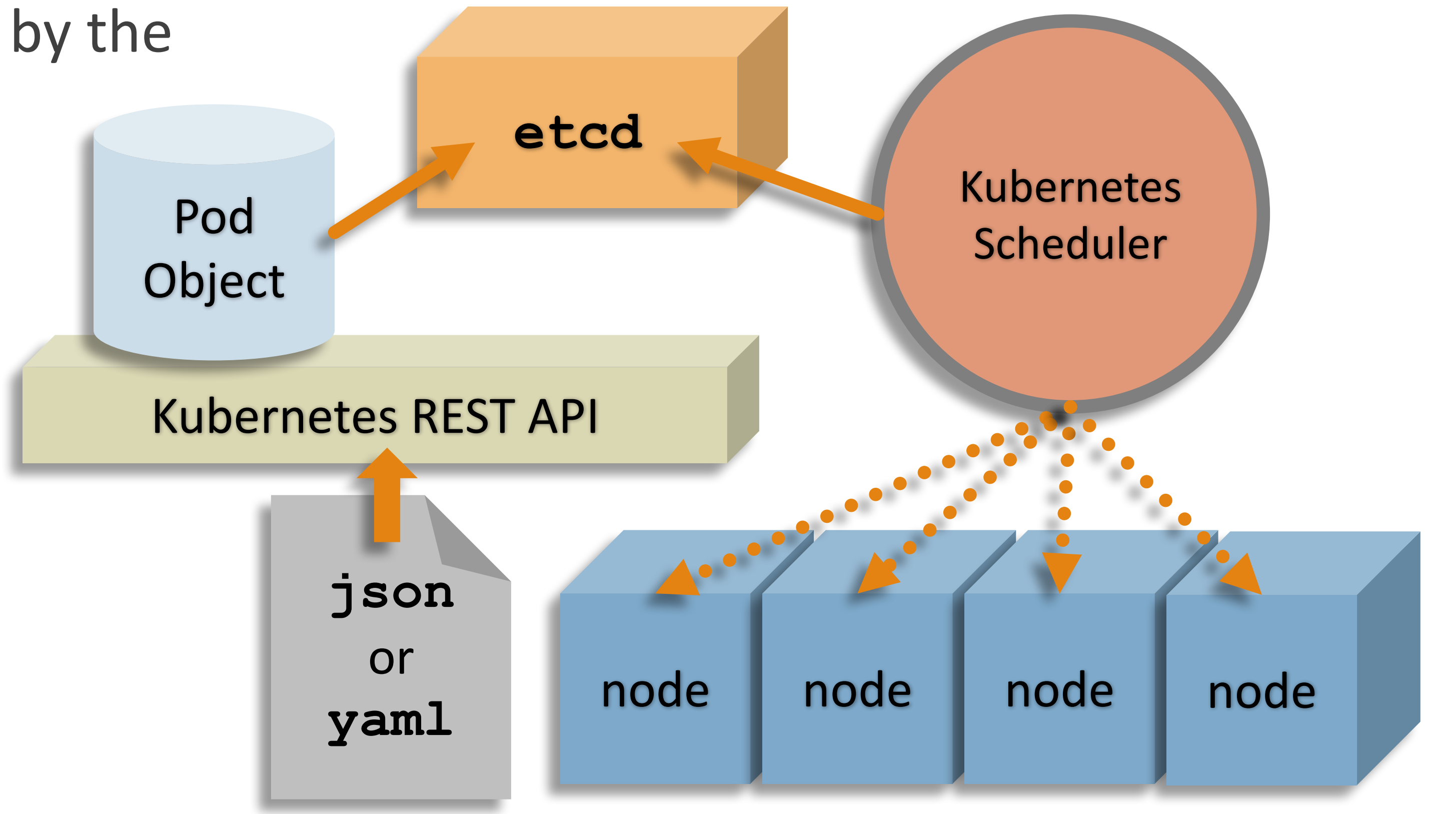
- A pod is one or more Docker containers
- Ensures collocation and shared fate
- Pods are scheduled and do not move nodes
- Docker containers share resources within the pod
  - Volumes
  - Network / IP
  - Port space
  - CPU / Mem allocations
- Pod health probes
  - Readiness
  - Liveness

Kubernetes Pods



Kubernetes Pods

- Kubernetes Scheduler reads from `etcd`
- Each node is scheduled by the Scheduler



Pod Probes

- Out of the box
  - Readiness
  - Liveness
- Probe types
  - ExecAction
  - TCPSocketAction
  - HTTPGetAction
- Results
  - Success
  - Failure
  - Unknown

Pod Restart Policies

- Pods restart on single node
- Only replication controller reschedules
- RestartPolicy
  - Always (default)
  - OnFailure
  - Never
- Applies to all containers in the pod
- For replication controllers, only Always applicable

Restart Policies Examples

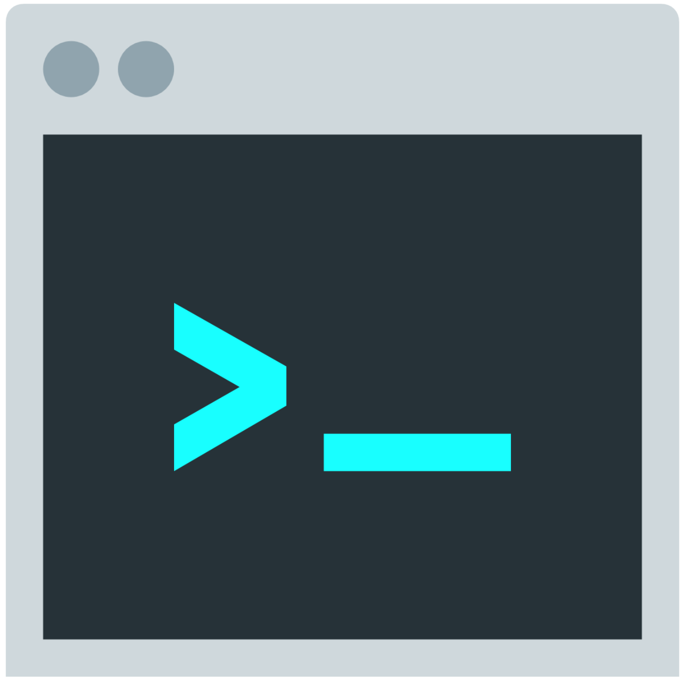
- Pod is Running: 1 container and container exits **success**
  - Always: restart container, pod stays Running
  - OnFailure: pod becomes Succeeded
  - Never: pod becomes Succeeded
- Pod is Running: 1 container and container exits **failure**
  - Always: restart container, pod stays Running
  - OnFailure: restart container, pod stays Running
  - Never: pod becomes Failed

Restart Policies Examples

- Pod is Running: 2 containers and container 1 exits failure
  - Always: restart container, pod stays Running
  - OnFailure: restart container, pod stays Running
  - Never: pod stays Running
- When container 2 exits from the above example
  - Always: restart container, pod stays Running
  - OnFailure: restart container, pod stays Running
  - Never: pod becomes Failed

Termination Messages Practices

- Help debug problems by putting messages into "termination log"
- Default location `/dev/termination-log`



Your First Pod

```
apiVersion: v1
kind: Pod
metadata:
 labels:
   name: influxdb
 name: influxdb
spec:
 containers:
   - image: docker.io/tutum/influxdb:latest
     name: influxdb
     ports:
       - containerPort: 8083
         name: admin
         protocol: TCP
       - containerPort: 8086
         name: http
         protocol: TCP
```

Deploy the `influxdb` Pod

- You can either copy/paste locally, or download the pod resource file like this:

```
 curl -s -L https://raw.githubusercontent.com/christian-
posta/docker-kubernetes-workshop/master/demos/hands-on-
pods/influxdb-simple.yaml -O influxdb-simple.yaml
```

- Once you have the file locally downloaded, run the `kubectl` command (`./cluster/kubectl.sh`) like this:

```
kubectl create -f influxdb-simple.yaml
```

Deploy the `influxdb` Pod

- ❑ Wait for the Docker image to download and the pod to start.
- ❑ We can watch the process, like this:

```
kubectl get pod --watch
```

- ❑ And watch the Docker events in another window, like this:

```
docker events
```

  - ❑ NOTE: You'll need to be logged into minion-1 directly to see the Docker events:

Interact with Your First Kubernetes Pod

- Let’s begin interacting by using our `describe` command, like this:

```
kubectl describe pod/influxdb
```

Interact with Your First Kubernetes Pod

- Output:

```
Name:                       influxdb
Namespace:                  demo-pods
Image(s):                   docker.io/tutum/influxdb:latest
Node:                       10.245.1.3/10.245.1.3
Labels:                     name=influxdb
Status:                     Running
Reason:
Message:
IP:                         10.246.1.7
Replication Controllers:            <none>
Containers:
  influxdb:
    Image:                  docker.io/tutum/influxdb:latest
    State:                  Running
      Started:              Sun, 18 Oct 2015 17:09:16 -0700
    Ready:                  True
    Restart Count:          0
Conditions:
  Type              Status
  Ready             True
Events:
<truncated>
```

Interact with Your First Kubernetes Pod

- Now, let’s try to ping:

```
ping 10.246.1.7
```

- Output:

```
PING 10.246.1.7 (10.246.1.7) 56(84) bytes of data.
64 bytes from 10.246.1.7: icmp_seq=1 ttl=63 time=2.43 ms
64 bytes from 10.246.1.7: icmp_seq=2 ttl=63 time=8.41 ms
64 bytes from 10.246.1.7: icmp_seq=3 ttl=63 time=1.27 ms
64 bytes from 10.246.1.7: icmp_seq=4 ttl=63 time=2.23 ms
^C
--- 10.246.1.7 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 1.271/3.587/8.411/2.819 ms
```

Interact with Your First Kubernetes Pod

- Now, let’s issue a command to that IP address.

```
export INFLUX_NODE=10.246.1.7
```

- These commands must be run from either `master` or `minion1`:

```
curl -G "http://$INFLUX_NODE:8086/query" --data-urlencode
"q=CREATE DATABASE kubernetes"
```

```
curl -X POST "http://$INFLUX_NODE:8086/write?db=kubernetes"
-d 'pod,host=node0 count=10'
```

```
curl -G "http://$INFLUX_NODE:8086/query?pretty=true" --data-
urlencode "db=kubernetes" --data-urlencode "q=SELECT
SUM(count) FROM pod"
```

Adding Readiness and Liveness Checks

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    name: influxdb
  name: influxdb
spec:
  containers:
    - image:
docker.io/tutum/influxdb:latest
      name: influxdb
      readinessProbe:
        httpGet:
          path: /ping
          port: 8086
        initialDelaySeconds: 5
        timeoutSeconds: 1
...
```

```
...
livenessProbe:
        httpGet:
          path: /ping
          port: 8086
        initialDelaySeconds: 15
        timeoutSeconds: 1
      ports:
        - containerPort: 8083
          name: admin
          protocol: TCP
        - containerPort: 8086
          name: http
          protocol: TCP
```

Adding Readiness and Liveness Checks

- First, let's stop and remove our previous `influxdb` pod since we'll be re-doing it this time with health-checking:

```
kubectl stop pod/influxdb
```

- Curl the pod resource as we did before, like this:

```
curl -s -L https://raw.githubusercontent.com/christian-
posta/docker-kubernetes-workshop/master/demos/hands-on-
pods/influxdb-readiness.yaml -O influxdb-readiness.yaml
```

- Use the `kubectl` command line again to create our pod, just like before:

```
kubectl create -f influxdb-readiness.yaml
```

Adding Resource Constraints to Our Pods

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    name: influxdb
  name: influxdb
spec:
  containers:
    - image:
docker.io/tutum/influxdb:latest
      name: influxdb
      resources:
        limits:
          cpu: "500m"
          memory: "128Mi"
      readinessProbe:
        httpGet:
          path: /ping
          port: 8086
...
```

```
...
initialDelaySeconds: 5
        timeoutSeconds: 1
    livenessProbe:
      httpGet:
        path: /ping
        port: 8086
      initialDelaySeconds: 15
      timeoutSeconds: 1
    ports:
      - containerPort: 8083
        name: admin
        protocol: TCP
      - containerPort: 8086
        name: http
        protocol: TCP
```

Adding Resource Constraints to Our Pods

- Let’s stop our previous pod, download the new manifest and run it:

```
kubectl stop pod/influxdb
```

- Now, `curl` the pod resource, just like before:

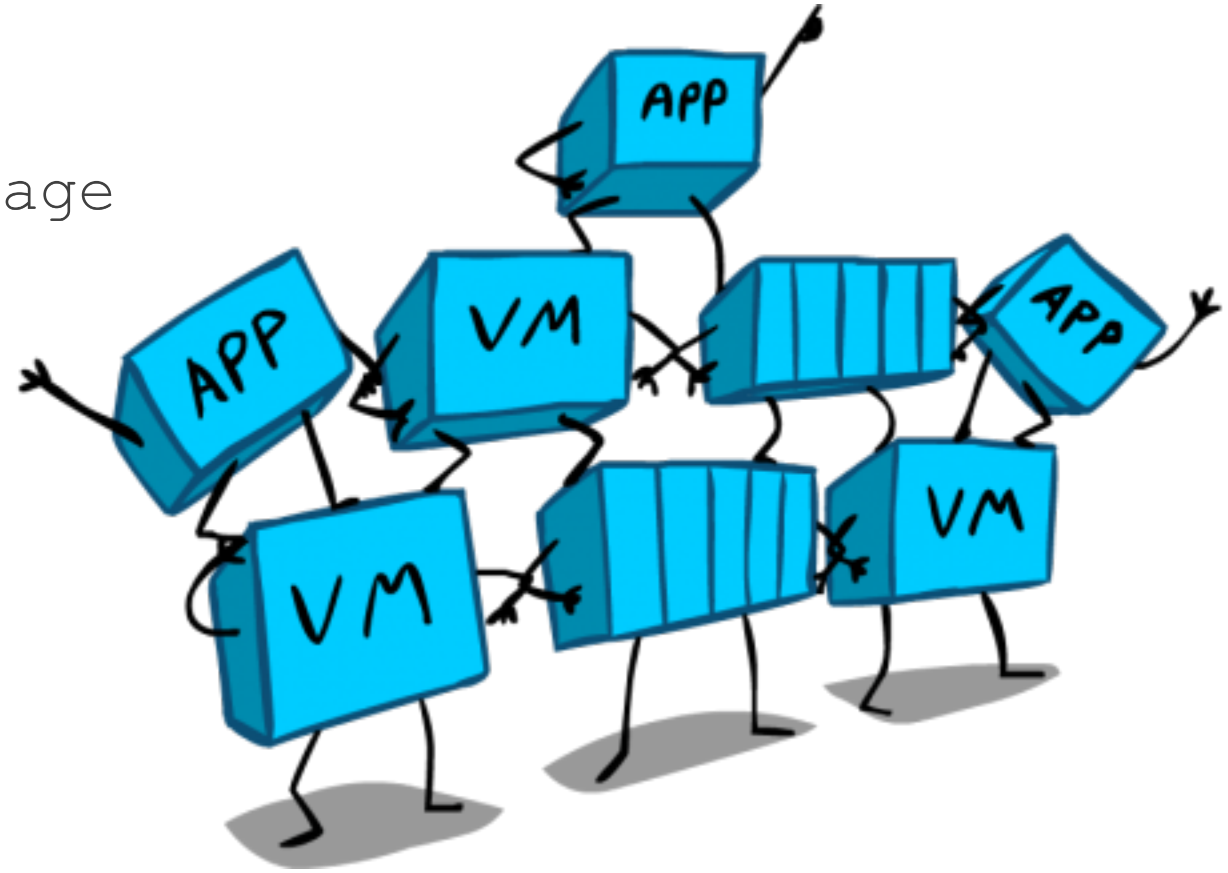
```
curl -s -L https://raw.githubusercontent.com/christian-posta/docker-kubernetes-workshop/master/demos/hands-on-pods/influxdb-resource-limits.yaml -O influxdb-resource-limits.yaml
```

- Use the `kubectl` command line again to create our pod:

```
kubectl create -f influxdb-resource-limits.yaml
```

Stateful Containers

- ❑ emtpyDir
- ❑ hostpath
- ❑ gcePersistentDisk
- ❑ awsElasticBlockStorage
- ❑ nfs
- ❑ glusterfs
- ❑ gitRepo



Stateful Containers

- Let’s stop our previous pod, download the new manifest and run it:

```
 curl -s -L https://raw.githubusercontent.com/christian-
posta/docker-kubernetes-workshop/master/demos/hands-on-
pods/local-persistent-volume.yaml | kubectl create -f -
```

- Now we’re going to create a "claim" for persistent volume space (details explained in class):

```
 curl -s -L https://raw.githubusercontent.com/christian-
posta/docker-kubernetes-workshop/master/demos/hands-on-
pods/local-pvc.yaml | kubectl create -f -
```

Stateful Containers

- Make sure your `influxdb` pod is stopped, like this:

```
kubectl stop pod influxdb
```

- Now, deploy your new pod that uses the `PersistentVolume`:

```
curl -s -L https://raw.githubusercontent.com/christian-
posta/docker-kubernetes-workshop/master/demos/hands-on-
pods/influxdb-local-pv.yaml | kubectl create -f -
```

Stateful Containers

- Follow along:

```
export INFLUX_NODE=10.246.1.7
```

```
curl -G "http://$INFLUX_NODE:8086/query" --data-urlencode
"q=CREATE DATABASE kubernetes"
```

```
curl -X POST "http://$INFLUX_NODE:8086/write?db=kubernetes" -d
'pod,host=node0 count=10'
```

```
curl -X POST "http://$INFLUX_NODE:8086/write?db=kubernetes" -d
'pod,host=node1 count=9'
```

```
curl -X POST "http://$INFLUX_NODE:8086/write?db=kubernetes" -d
'pod,host=node2 count=12'
```

```
curl -G "http://$INFLUX_NODE:8086/query?pretty=true" --data-
urlencode "db=kubernetes" --data-urlencode "q=SELECT SUM(count)
FROM pod"
```

Stateful Containers

- Restart:

```
curl -s -L https://raw.githubusercontent.com/christian-posta/docker-kubernetes-workshop/master/demos/hands-on-pods/influxdb-local-pv.yaml | kubectl create -f -
```

- And query:

```
curl -G "http://$INFLUX_NODE:8086/query?pretty=true" --data-urlencode "db=kubernetes" --data-urlencode "q=SELECT SUM(count) FROM pod"
```

Pods in Production

- Uses `PersistentVolumes` for stateful pods/containers
- Uses Kubernetes secrets to distribute credentials
- Leverages readiness and liveness probes
- Considesr resources (CPU/Mem) limits and quotas
- Uses termination messages

Grouping

ORGANIZING YOUR CLUSTER

Grouping

- Decouple application dimensions from infrastructure deployments
  - Tiers
  - Release Tracks
  - Grouping of micro services
- Kubernetes clusters are dynamic
- Predicated on failure
- Need a way to identify groups of services
- Be able to add/remove from group on failures
- Query-able
- Kubernetes Labels

What are Kubernetes Labels?

- Arbitrary metadata
- Attach to any API object
  - eg, Pods, ReplicationControllers, Endpoints, etc
- Simple Key = Value Assignments
- Can be queried with selectors

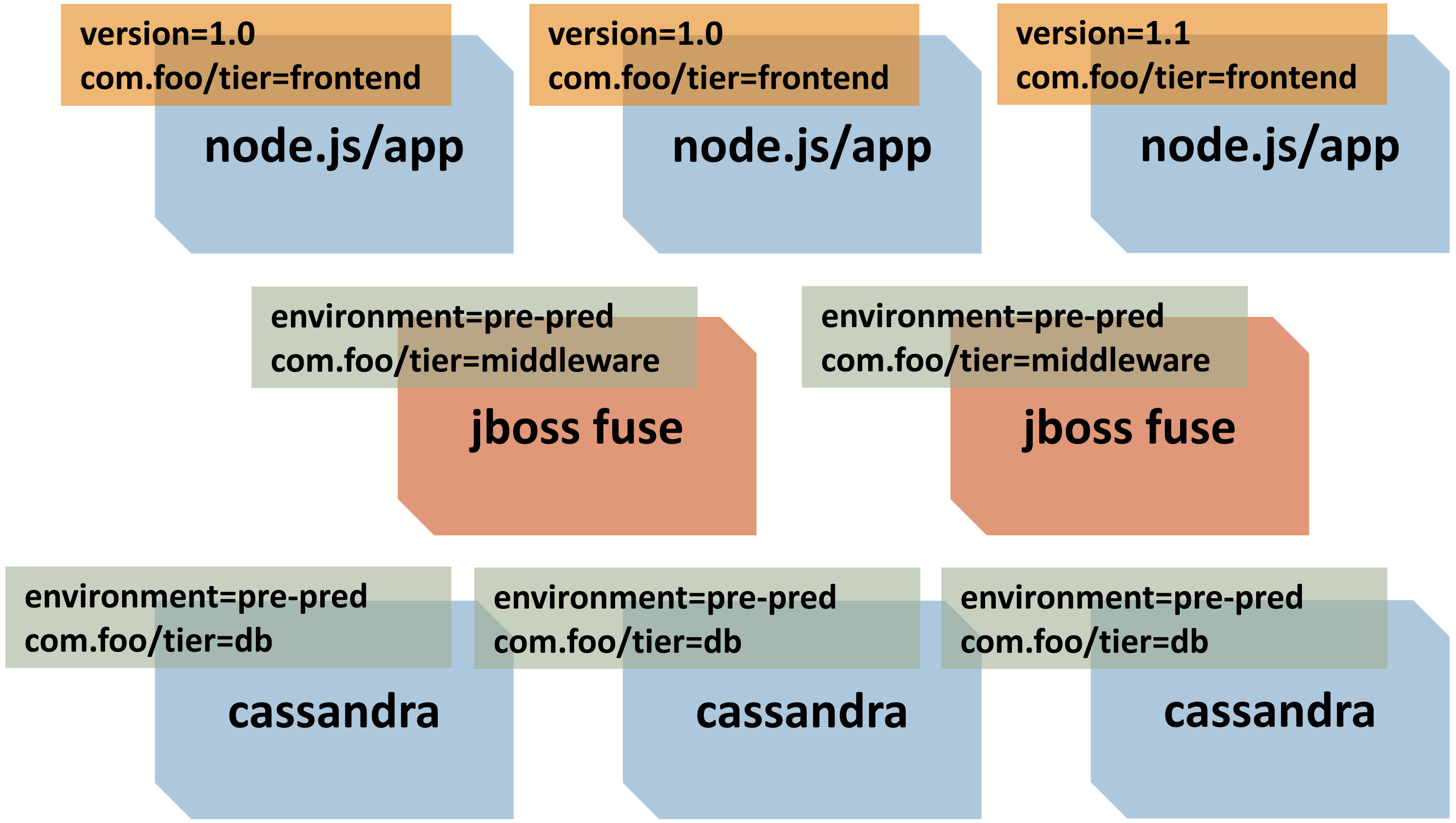
Label Examples

- ❑ Release = Stable, Release = Canary
- ❑ Environment = Dev, Environment = Qa, Environment = Prod
- ❑ Tier = Frontend, Tier = Backend, Tier = Middleware
- ❑ Partition = Customer A, Partition = Customer B, Etc.
- ❑ Used to group “like” objects — no expectation of uniqueness

Keys and Values

- The structure of a key:
  - `<prefix>/<name>`
  - example key `org.apache.hadoop/partition`
- Prefix can be up to 253 characters
- Name can be up to 63 characters
- Values can be ([a-z0-9A-Z]) with dashes (-), underscores (_), dots (.)
- Values can be up to 63 characters

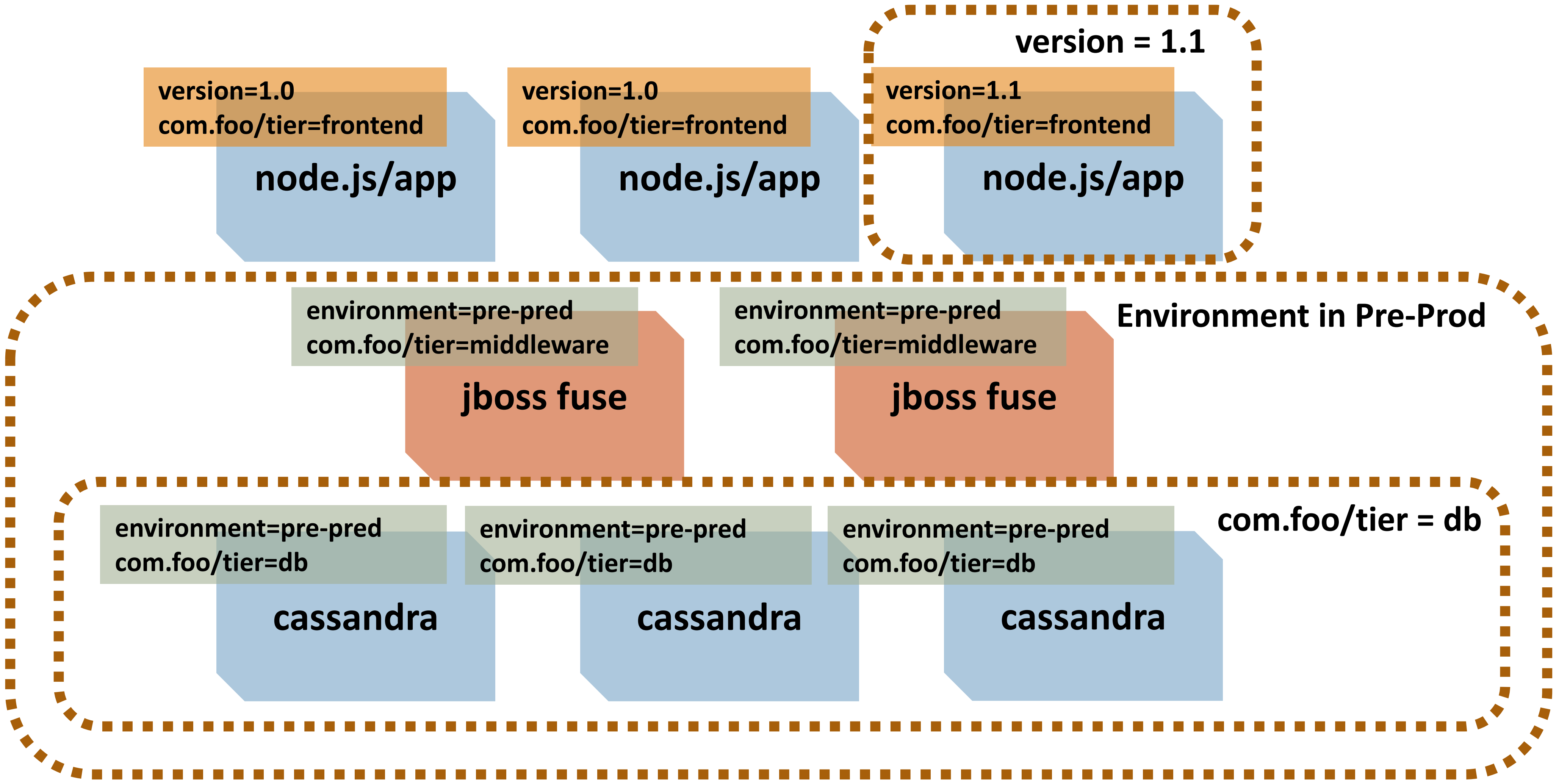
Example Labels



Selectors for Grouping

- Labels are query-able metadata
- Selectors can do the queries
- Equality based
  - Environment = Production
  - Tier != Frontend
  - Combinations: "tier != frontend, version = 1.0.0"
- Set based
  - Environment in (production, pre-production)
  - Tier notin (frontend, backend)
  - Partition

Example Selectors



Annotations

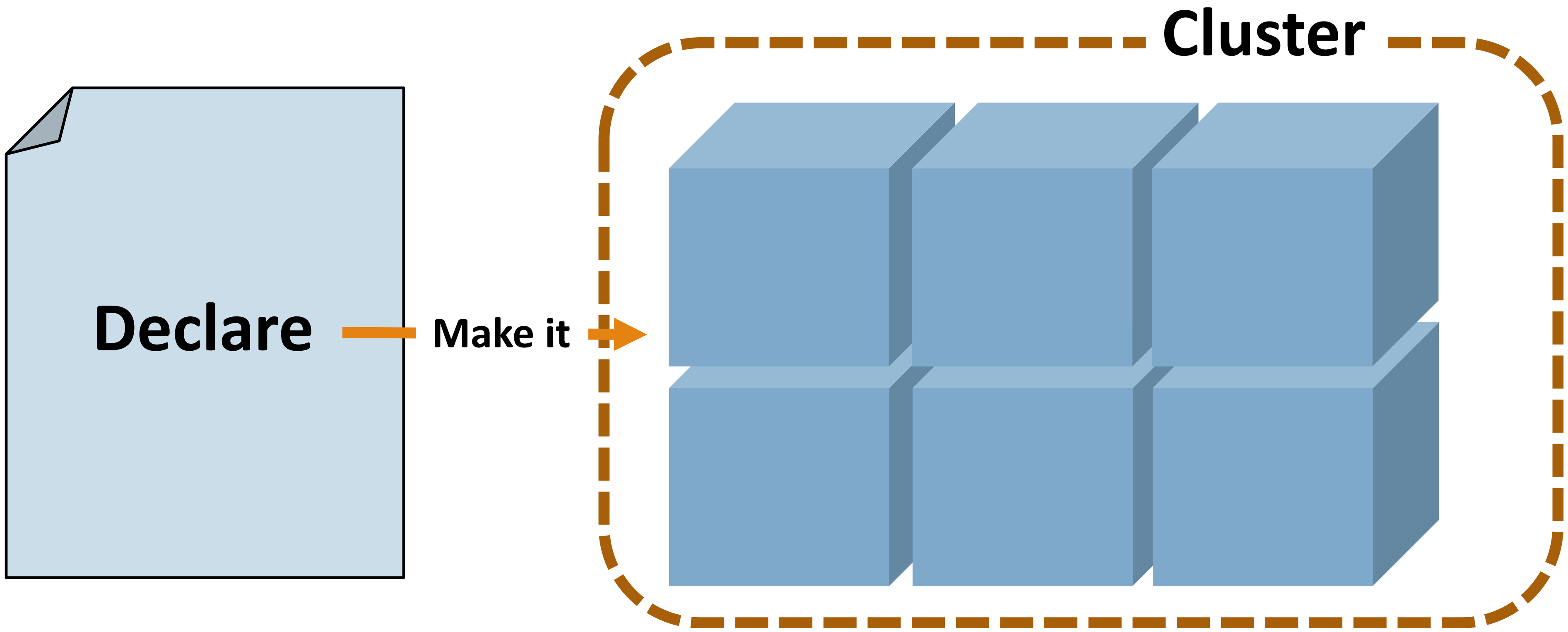
- Attaching useful metadata to objects - not query-able
- Data that you want to know to build context - easy to have it close
  - Author information
  - Build date/timestamp
  - Links to SCM
  - PR numbers/Gerrit patch sets
- Annotations are not query-able
- Can build up tooling around annotations

What Should you Label and Annotate?

- Everything!

Scaling and Ensuring Cluster Invariants

Replication Controllers



Replication Controllers

- ❑ Can configure the number of **replicas** of a pod
- ❑ Will be scheduled across applicable nodes
- ❑ Replication controllers can change **replica** value to scale up/down
- ❑ Which pods are scaled depends on RC selector
- ❑ Labels and selectors are the backbone of grouping

Replication Controllers

- Can even do complicated things with labels/RCs
- Take a pod out of cluster by changing its label
- Can have multiple RCs so no SPOF
- If a pod is unhealthy
  - RC can kill and restart

Replication Controller

```
apiVersion: v1
kind: ReplicationController
metadata:
  labels:
    app: inspector
    track: stable
  name: inspector
spec:
  replicas: 1
  selector:
    app: inspector
    track: stable
  template:
    metadata:
      labels:
        app: inspector
        track: stable
...
```

```
...
spec:
      containers:
      - image:
b.gcr.io/kuar/inspector:1.0.0
        imagePullPolicy:
IfNotPresent
        name: inspector
        resources: {}
        terminationMessagePath:

     /dev/termination-log
      dnsPolicy: ClusterFirst
      restartPolicy: Always
status:
  observedGeneration: 1
  replicas: 1
```

Replication Controller

```
curl -s -L https://raw.githubusercontent.com/christian-posta/docker-
kubernetes-workshop/master/demos/replication-controllers/inspector-rc.yaml
| kubectl create -f -
```

```
root@server $ kubectl get rc
CONTROLLER   CONTAINER(S)   IMAGE(S)                          SELECTOR                     REPLICAS
inspector    inspector      b.gcr.io/kuar/inspector:1.0.0     app=inspector,track=stable   1
```

Scale Up

We want to change the value of the replicas for our inspector replication controller to scale up:

```
kubectl scale rc inspector --replicas=5
```

If successful you should see:

```
scaled
```

Now do a `get pods` to see

```
root@server $ kubectl get pod
NAME              READY     STATUS    RESTARTS   AGE
inspector-3sh6q   1/1       Running   0          13m
inspector-5djgp   1/1       Running   0          13m
inspector-9b68m   1/1       Running   0          13m
inspector-ohaod   1/1       Running   0          13m
inspector-r5bqo   1/1       Running   0          19m
```

Scale Down

Let’s scale the inspector app down to two pods:

```
kubectl scale rc inspector --replicas=2
```

Now when you get the pods, there should just be two:

```
root@server $ kubectl get pod
NAME                 READY      STATUS     RESTARTS    AGE
inspector-5djgp      1/1        Running    0           13m
inspector-r5bqo      1/1        Running    0           19m
```

Autoscaling

- Ongoing work in the community
  https://github.com/kubernetes/kubernetes/blob/master/docs/proposals/autoscaling.md
- Use existing monitoring tools to provide scaling

Loadbalancing Service Discovery

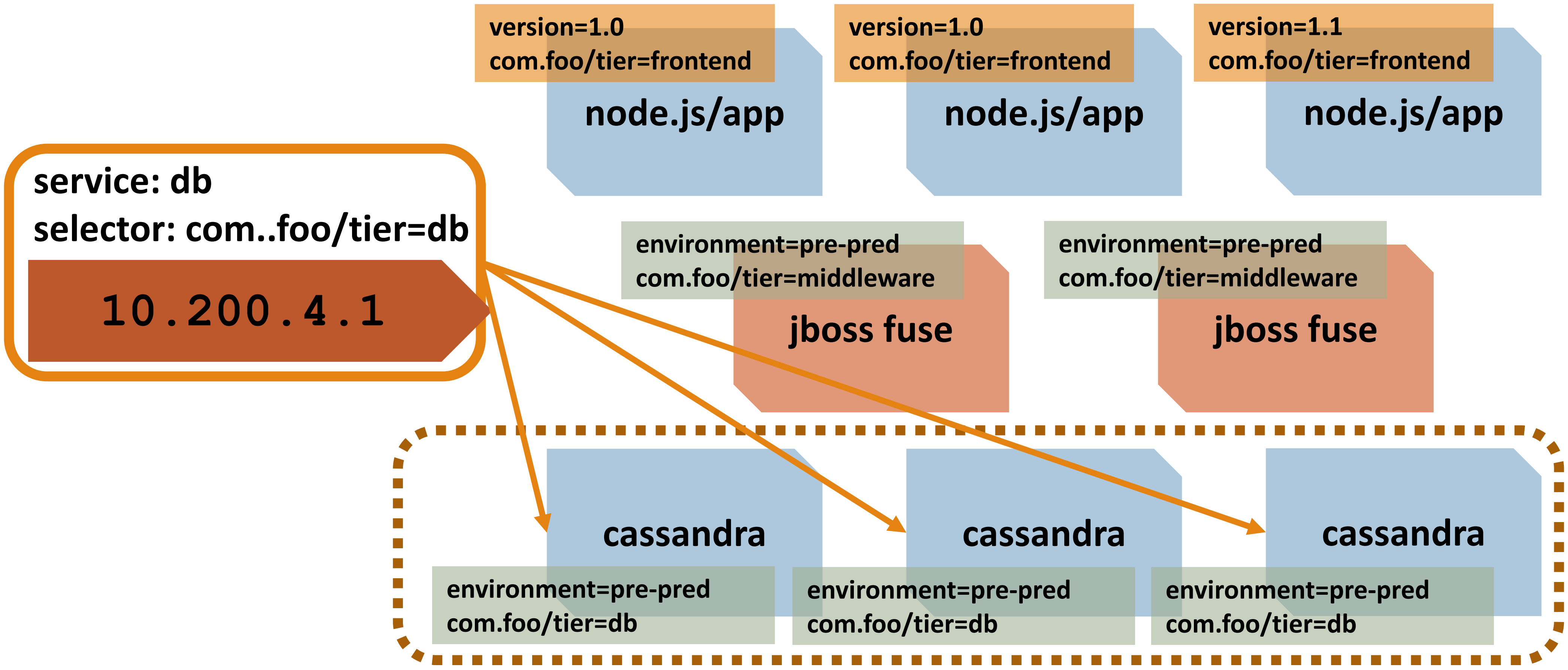
You Have Lots of Pods

- Pods have their own IP address
- Pods can come and go; they’re fungible
- Replication controllers maintain replica invariants
- So you have lots of Pods and IPs, which do you use?
- Use labels for grouping
- Kubernetes services does the heavy lifting of finding Pods

Kubernetes Services

- Decouple providers and accessors of services
- Don’t depend on Pod IPs directly
- Need to find pods that match certain selection criteria
  - mmm… labels and selectors again :)
- Pods can live on any node
- Use a single IP that doesn’t change
- Virtual IP load balancing and discovery

Kubernetes Services



Kubernetes Services

- Example:

```
apiVersion: v1
kind: Service
metadata:
  name: inspector
  labels:
    app: inspector
spec:
  type: NodePort
  selector:
    app: inspector
  ports:
  - name: http
    nodePort: 36000
    port: 80
    protocol: TCP
```

Kubernetes Services – No Selectors

- Use selectors to locate existing Kubernetes pods
- Use a service without selectors to point to external services
  - DB running outside of Kubernetes
- Example:

```
kind: "Service"
  apiVersion: "v1"
  metadata:
    name: "my-service"
  spec:
    ports:
      -
        protocol: "TCP"
        port: 80
        targetPort: 9376
```

Kubernetes Services – No Selectors

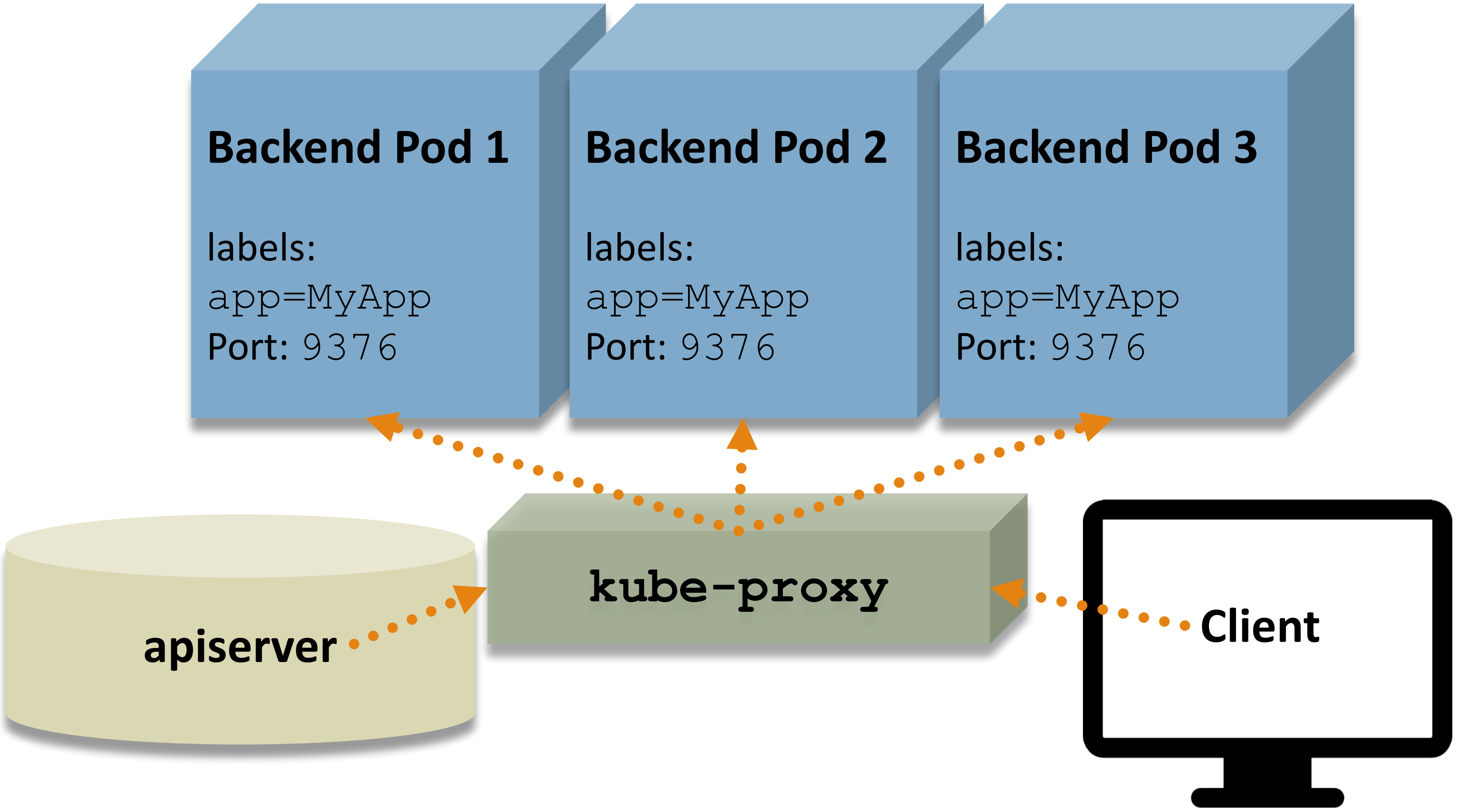
- Endpoint objects will not be created
  - Will need to make those and point them where you want:
- Example:

```
kind: "Endpoints"
 apiVersion: "v1"
 metadata:
   name: "my-service"
 subsets:
   -
     addresses:
       -
         IP: "1.2.3.4"
     ports:
       -
         port: 80
```

How it Works

- When a new service and/or endpoint object is created in `etcd`
- Also, `kube-proxy` opens a port on the node
- Connections to that port will be proxy-ed to the pods
- And `kube-proxy` maintains `iptables` routing from the `clusterIP` (VIP) to the node port

How it Works


Backend Pod 1
labels:
app=MyApp
Port: 9376
Backend Pod 2
labels:
app=MyApp
Port: 9376
Backend Pod 3
labels:
app=MyApp
Port: 9376
kube-proxy
apiserver
Client

Discovering Services from Collaborators

- Example, UI pods need to use DB pods
  - `{svcname}_SERVICE_HOST`
  - `{svcname}_SERVICE_PORT`
  - Implies an ordering requirement - service must have been created first
- DNS - as a cluster add on (recommended)
  - `{svcname}.{nsname}`
  - e.g.: `my-service.my-namespace` would be how you find a service in the `my-namespace` namespace

```
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
```

Service Types

- ClusterIP
  - Default, create an internal IP that can be used within pods for service discovery
- NodePort - open a port on all of the nodes that the service listens on (can be accessed from outside the cluster)
  - Set type == “NodePort”
  - Will allocate a port in the default range (can be configured): `30000-32767`
  - Will set `spec.ports.nodePort` for you
  - Can set it manually and it will try to use this fixed port, or it will fail
  - Can then use your own load balancers and point to this specific port (external access)
- LoadBalancer — useful for cloud providers that expose external load balancers

Combine Service and RC Resources

```
kind: "List"
 apiVersion: "v1"
 items:
   - kind: "Service"
     apiVersion: "v1"
     metadata:
       name: "mock"
       labels:
         app: "mock"
         status: "replaced"
     spec:
       ports:
         - protocol: "TCP"
           port: 99
           targetPort: 9949
       selector:
         app: "mock"
```

```
- kind: "ReplicationController"
      apiVersion: "v1"
      metadata:
        name: "mock"
        labels:
          app: "mock"
          status: "replaced"
      spec:
        replicas: 1
        selector:
          app: "mock"
        template:
          metadata:
            labels:
              app: "mock"
          spec:
            containers:
              - name: "mock-container"
                image: "kubernetes/pause"
                ports:
                  - containerPort: 9949
                    protocol: "TCP"
```

Create a Service

- Let's create a service now to play around with:

```
 curl -s -L https://raw.githubusercontent.com/christian-
posta/docker-kubernetes-
workshop/master/demos/services/inspector-svc.yaml | kubectl
create -f -
```

  - NOTE: Notice we are piping the results to `kubectl`

- This service exposes to a NodePort on the Node.

- To see which port is exposed we can `get` that info, like this:

```
 kubectl get svc/inspector -o template -t "{{.spec.ports}}"
```

Output from Inspector Service

- Now, what happens when we do the following:

```
curl -G http://10.245.1.3:30727/
```

Output from Inspector Service

- Output:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="css/bootstrap.min.css">
  </head>
  <div class="container">
    <body>
      <h1>Inspector</h1>
      <p>
       <b>Home</b> |
       <a href="./env">Environment</a> |
       <a href="./mnt">Mounts</a> |
       <a href="./net">Network</a>
      </p>
      <hr/>
      <p>Version: Inspector 1.0.0</p>
      </ul>
    </body>
  </div>
</html>
```

Output from Inspector Service

- That's now `curl /net`, like this:

```
curl -G http://10.245.1.3:30727/net
```

- Output:

```
<snipped>
         <tr>
           <td>eth0</td>
           <td>02:42:0a:f6:01:0e</td>
           <td><ul class="list-unstyled">
                 <li>10.246.1.14/24</li>
                 <li>fe80::42:aff:fef6:10e/64</li>
               </ul>
           </td>
         </tr>
      </snipped>
```

Lab

End of Chapter