Kubernetes

Manage a cluster of Linux containers in multi-cloud world, public, private or hybrid.

Luciano Faustino , Tiago Katcipis

Agenda

- Introduction
- Why Kubernetes ?
- Concepts
- Deploying
- Service Management
- Monitoring

Introduction

What is Kubernetes?

- Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts
- Scale your applications on the fly
- Seamlessly roll out new features
- Optimize use of your hardware by using only the resources you need
- Kubernetes is: lean, portable, extensible, self-healing

History

- The Kubernetes project was started by Google in 2014
- Kubernetes builds upon a decade and a half of experience that Google has with running production workloads at scale, combined with best-of-breed ideas and practices from the community
- Google's Borg system

research.google.com/pubs/pub43438.html (https://research.google.com/pubs/pub43438.html)

kubernetes.io/community/ (http://kubernetes.io/community/)

Why Kubernetes?

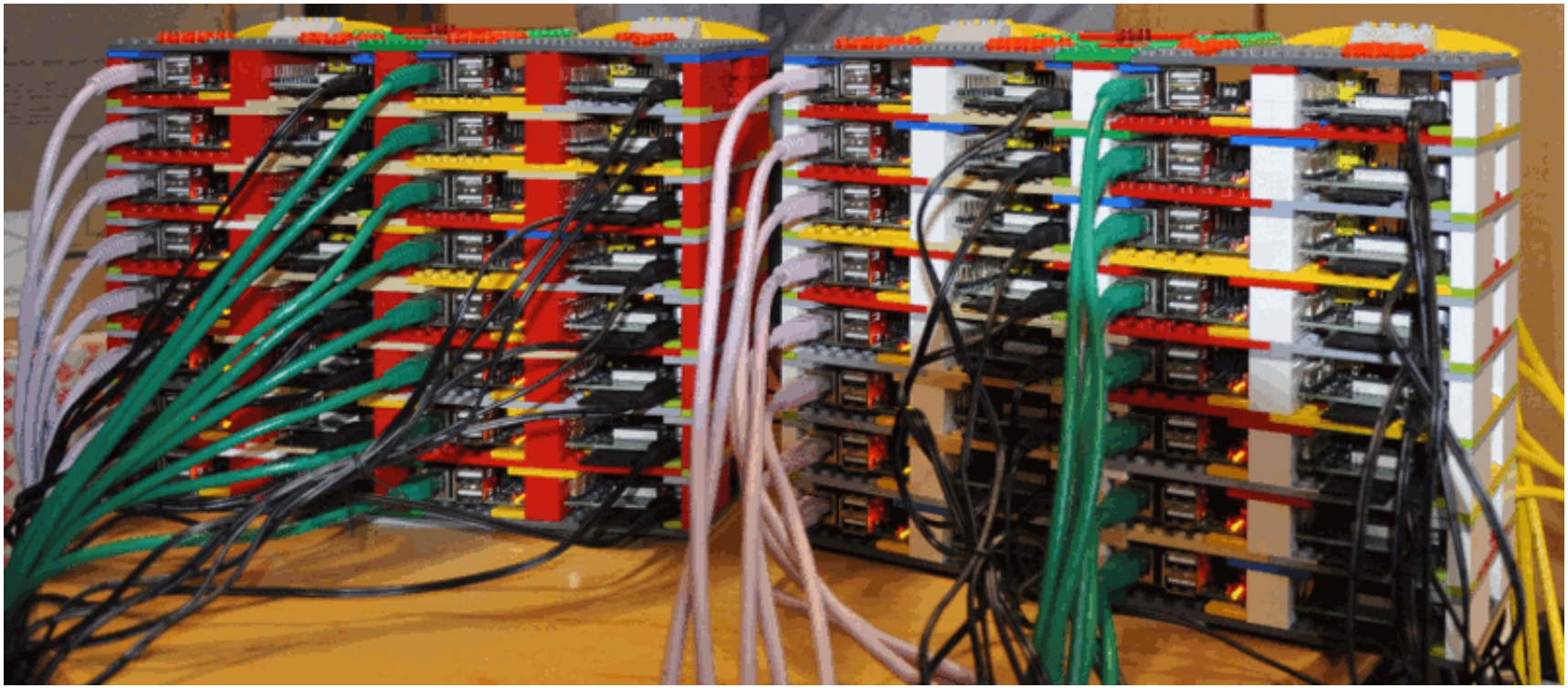
Some problems solved

- Keep services running
- Stop services if necessary
- Schedule services around a cluster
- Load balance
- Service discovery
- Health checking
- Data management
- Support to autoscale
- Support to rolling updates
- Multi cloud

Kubernetes Concepts

Cluster

- A cluster is a set of physical or virtual machines and other infrastructure resources used by Kubernetes to run your applications



Node

- Is a worker machine in Kubernetes, previously known as Minion
- Node may be a VM or physical machine, depending on the cluster
- Each node has the services necessary to run Pods and is managed by the master components

Pods

- It is a set of containers
- They share all resources (context)
- Each pod has its own IP, that is shared through all containers inside it
- Models a logical host (that is why each one has its own IP)

Labels

- key/value pairs that are attached to objects, such as pods.
- Used to organize and select groups of objects
- Labels enable users to map their own organizational structures onto system objects in a loosely coupled fashion, without requiring clients to store these mappings

Example labels:

```
"release" : "stable", "release" : "canary"
```

or

```
"environment" : "dev", "environment"* : "qa", "environment"* : "production"
```

Label Selector

- Unlike names and UIDs, labels do not provide uniqueness. In general, we expect many objects to carry the same label(s)
- Via a label selector, the client/user can identify a set of objects

Example labels selector:

```
environment = production
```

The former selects all resources with key equal to environment and value equal to production

```
environment in (production, qa)
```

The first example selects all resources with key equal to environment and value equal to production or qa

Replication Controllers

- Ensures that a specified number of pod "replicas" are running at any one time
- If there are too many, it will kill some
- If there are too few, it will start more
- Replaces pods that are deleted or terminated for any reason, such as in the case of node failure
- Think of it similarly to a cluster aware process supervisor
- Useful to perform rolling updates

Services

- Defines a logical set of Pods and a policy by which to access them
- Offers a virtual-IP-based bridge to Services which redirects to the backend Pods
- Service consumers always see the same IP, even if pods IPs changes
- Pods set that composes a service is built using a Label Selector

Volumes

- The Kubernetes Volume abstraction (Persistent Volume)
- Kubernetes supports several types of Volumes: emptyDir, hostPath, gcePersistentDisk, awsElasticBlockStore, nfs, iscsi, glusterfs, rbd, gitRepo, secret, persistentVolumeClaim

Secrets

- Objects of type secret are intended to hold sensitive information, such as passwords, OAuth tokens, and ssh keys
- Putting this information in a secret is safer and more flexible than putting it verbatim (text plain) in a pod definition or in a docker image

Names

- Names are generally client-provided
- Only one object of a given kind can have a given name at a time
- Names are the used to refer to an object in a resource URL

```
GET /api/v1/pods/some-name
```

Namespaces

- Intended for use in environments with many users spread across multiple teams, or projects
- A way to divide cluster resources between multiple uses
- Kubernetes starts with two initial namespaces: default, kube-system

Annotations

- Like labels, annotations are key-value maps
- Possible information that could be recorded in annotations:

```
lightweight rollout tool metadata
```

or

```
build/release/image information (timestamps, release ids, git branch,
PR numbers, image hashes, registry address, etc.)
```

or

```
phone/pager number(s) of person(s) responsible, or directory entry where
that info could be found, such as a team website, etc.
```

Deploying

* Provisioning cluster nodes

* Configuring a Kubernetes compatible network

* Deploying Kubernetes services and client tools

* Kubernetes can run on a range of platforms

  github.com/kubernetes/kubernetes/blob/master/docs/getting-started-guides/README.md#table-of-solutions (https://github.com/kubernetes/kubernetes/blob/master/docs/getting-started-guides/README.md#table-of-solutions)

Operation System

* CoreOS

CoreOS is a minimal operating system that supports popular container systems out of the box. The operating system is designed to be operated in clusters.

coreos.com (https://coreos.com)

Why?

- Cluster
- Minimal
- Fastpatch
- Bootstrap with cloud-config
- Container

Automation Tools

* Cloud Machine

This is a Go Project that should be used to create a cloud environment.The app will create volumes and instance through AWS, although in the next future it'll be possible use other backends like Microsoft Azure, Google Cloud Platform, etc.

github.com/NeowayLabs/cloud-machine (https://github.com/NeowayLabs/cloud-machine)

Why?

- Simple
- Fast
- Without Bugs
- Write in Golang
- ????

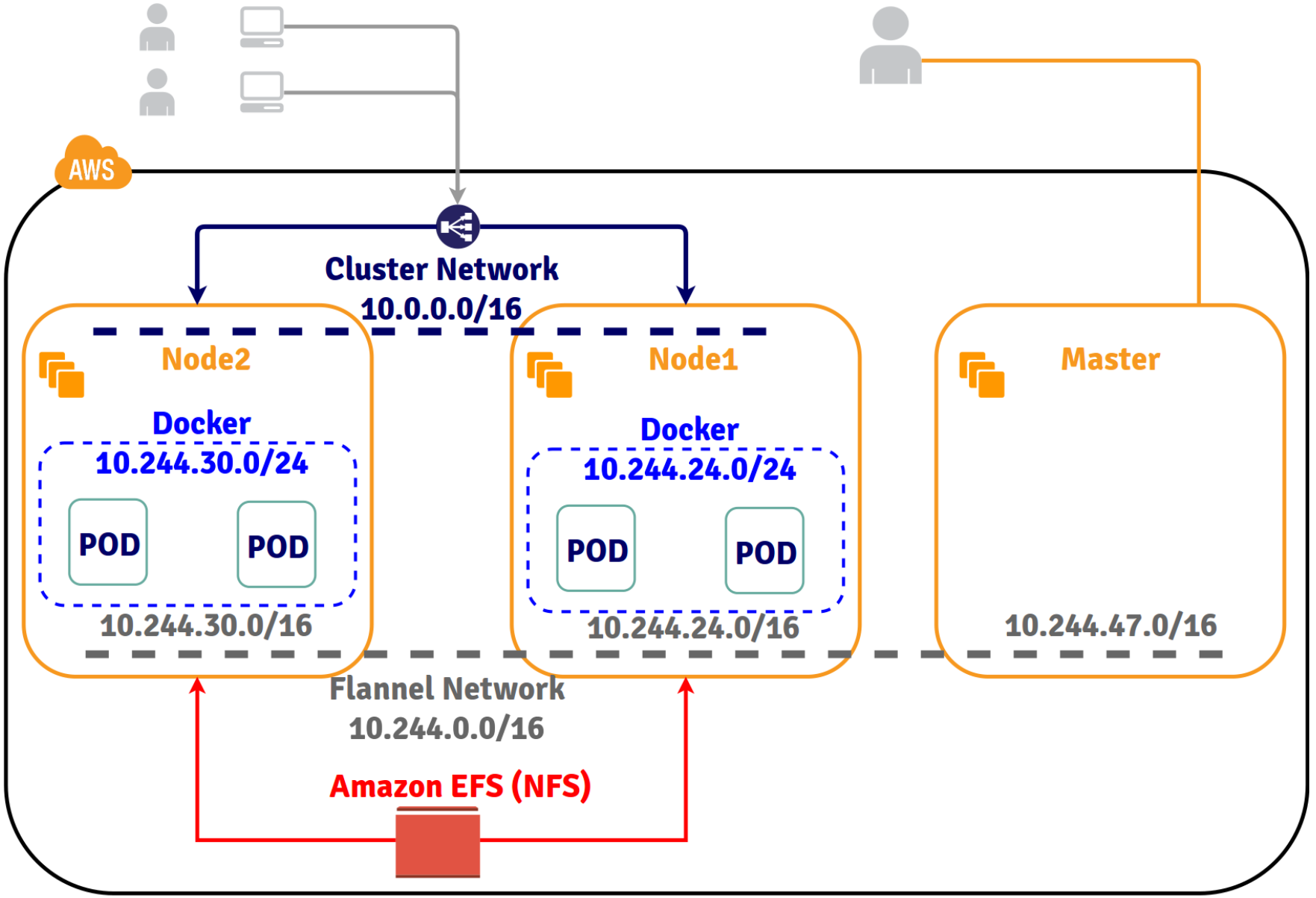
Kubernetes Control Plane (Master)

- **Kubernetes API Server** : The apiserver serves up the Kubernetes API
- **Scheduler** : Watches the apiserver for unscheduled pods and schedules them onto healthy nodes based on resource requirements
- **Controller Manager Server** : All other cluster-level functions
- **etcd** : A distributed consistent key-value store for shared configuration and service discovery. All persistent master state is stored in an instance of etcd
- **flannel** : Is a very simple overlay network. Provide SDN for Kubernetes
- **Docker Mirror** : Keep most of the image fetch traffic on your local network

Kubernetes Node

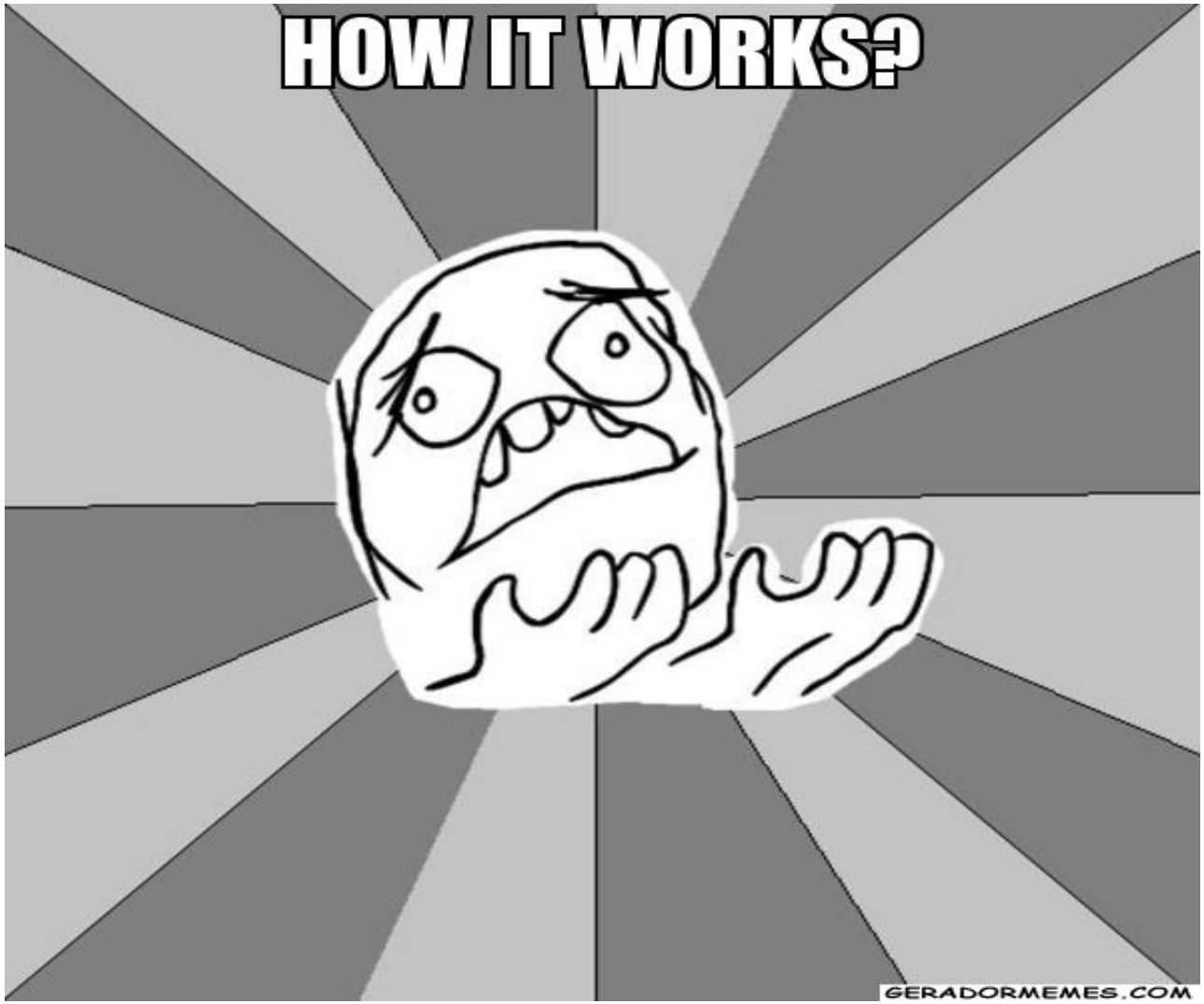
- **Kubelet** : Manages pods and their containers, their images, their volumes, etc
- **Kube-Proxy** : Simple network proxy and load balancer
- **etcd** : A distributed consistent key-value store for shared configuration and service discovery. All persistent master state is stored in an instance of etcd
- **flannel** : Is a very simple overlay network. Provide SDN for Kubernetes
- **Docker** : Linux Container :-)

Kubernetes Network



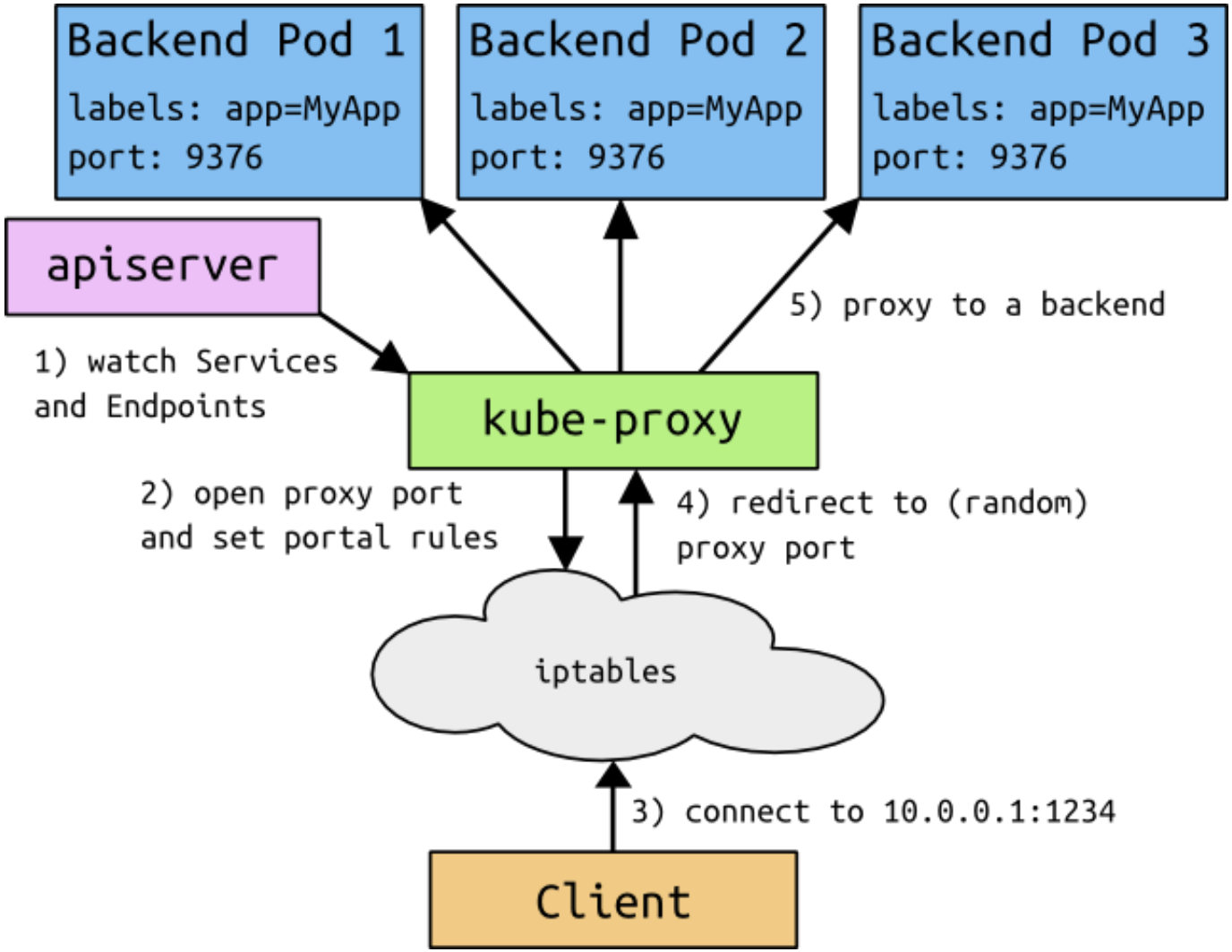

Text

Cluster Network



Cluster Network

Node Network Inside



Service Management

Running a service

```
kubectl -s http://{HOST}:{PORT} create -f k8s/rc/nginx.yaml

# cat k8s/rc/nginx.yaml
```

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx
spec:
  replicas: 2
  selector:
    app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

Exposing a service

```
kubectl -s http://{HOST}:{PORT} create -f k8s/svc/nginx.yml

cat k8s/svc/nginx.yml
```

```
apiVersion: v1
kind: Service
metadata:
  labels:
    name: nginx-service
  name: nginx-service
spec:
  deprecatedPublicIPs:
    - 192.168.33.11
  ports:
    - port: 80 # the port that this service should serve on
      # the container on each pod to connect to, can be a name
      # (e.g. 'www') or a number (e.g. 80)
      targetPort: 80
      protocol: TCP
  # just like the selector in the replication controller,
  # but this time it identifies the set of pods to load balance
  # traffic to.
  selector:
    app: nginx
```

Service Discovery

- Kubernetes supports 2 primary modes of finding a Service

* Environment variables

- When a Pod is run on a Node, the kubelet adds a set of esvironment variables for each active Service
- Supports both Docker links compatible variables and simpler **{SVCNAME}_SERVICE_HOST** and **{SVCNAME}_SERVICE_PORT** variables

For example, the Service **"redis-master"** which exposes TCP port 6379

```
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 Discovery

***** DNS

- An optional (though strongly recommended) cluster add-on is a DNS server
- The DNS server watches the Kubernetes API for new Services and creates a set of DNS records for each

For example, if you have a Service called **"redis-master"** in Kubernetes Namespace **"my-ns"**

a DNS record for **"redis-master.my-ns"** is created.

- Kubernetes also supports DNS SRV (service) records for named ports

You can do a **DNS SRV** query for **"_http._tcp.redis-master.my-ns"** to discover the port number for **"http"**

Load balance

Data management

Persistent Volume

```
kubectl -s http://{HOST}:{PORT} create -f k8s/pv/efs.yml

cat k8s/pv/efs.yml
```

```
  apiVersion: v1
  kind: PersistentVolume
  metadata:
    name: pv0001
  spec:
    capacity:
      storage: 50Gi
    accessModes:
      - ReadWriteOnce
    persistentVolumeReclaimPolicy: Recycle
    nfs:
      path: /
      server: us-west-2a........efs.us-west-2.amazonaws.com
```

Persistent Volume Claim

```
kubectl -s http://{HOST}:{PORT} create -f k8s/pvc/efs.yml

cat k8s/pvc/efs.yml
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: efs
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
```

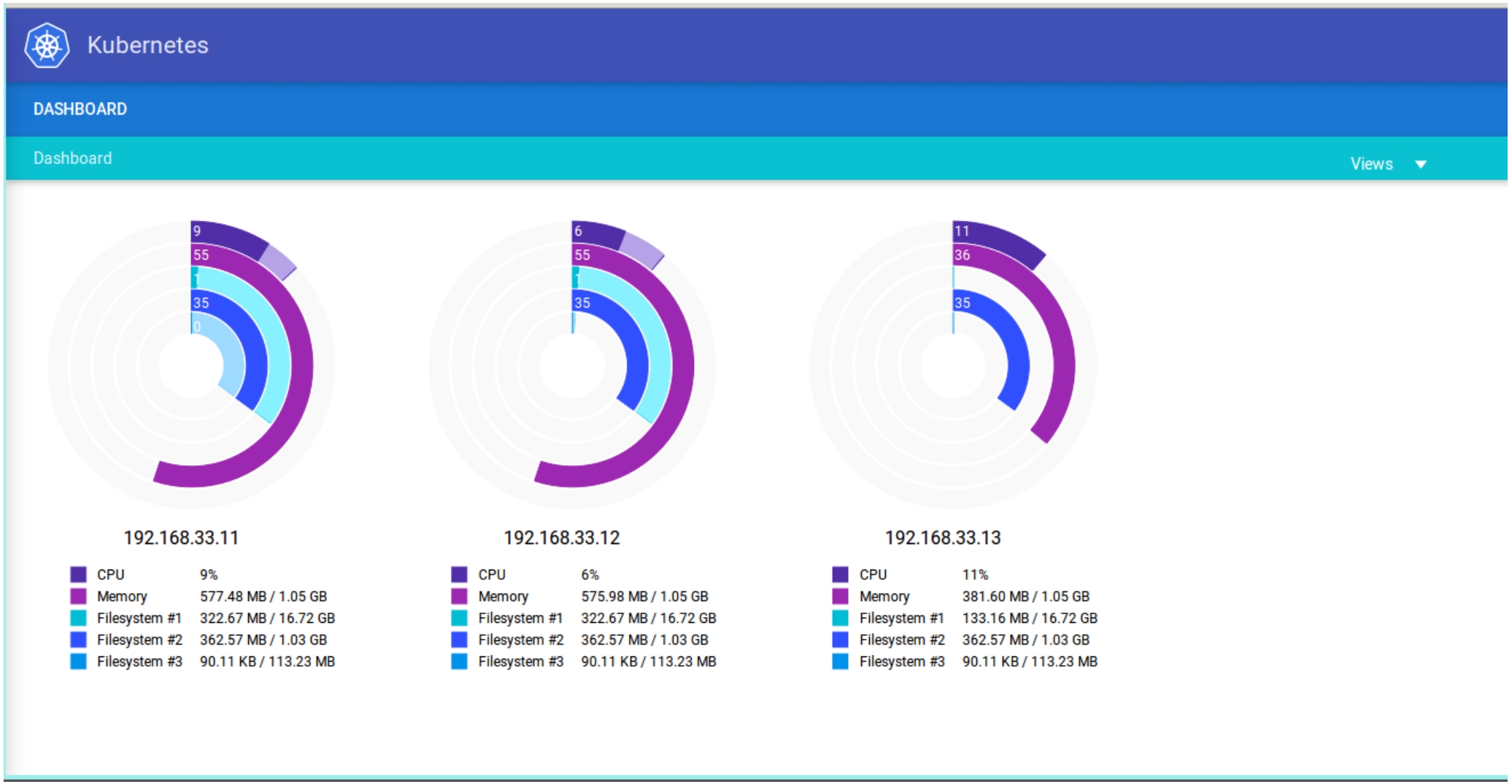
Usage

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        volumeMounts:
        - mountPath: /data/www
          name: nfs-storage
      volumes:
      - name: nfs-storage
        persistentVolumeClaim:
          claimName: ebs
```

Health Checking

Monitoring

```
http://{HOST}:{PORT}/ui/
```



Thank you

Luciano Faustino , Tiago Katcipis
lucianoborguetti@gmail.com , tiagokatcipis@gmail.com

(mailto:lucianoborguetti@gmail.com%20%20%20%20%20,%20tiagokatcipis@gmail.com)

https://github.com/lborguetti , https://github.com/katcipis

(https://github.com/lborguetti%20%20,%20https://github.com/katcipis)