Kubernetes Workshop

Part 3: Kubernetes Basics

What will we do for the next hour?

- Getting started with Kubernetes
  - Using Kubernetes in your own cluster

- Learn the basic features:
  - Deploying, Scheduling, Scaling and Discovering and more.

- After this, you will know enough to use Kubernetes as a software developer.

Check if your kubernetes cluster is available

```
$ kubectl get nodes
NAME                                        STATUS      AGE
gke-cluster-1-default-pool-1cf4645c-9fa5    Ready       2h
gke-cluster-1-default-pool-1cf4645c-equw    Ready       2h
```
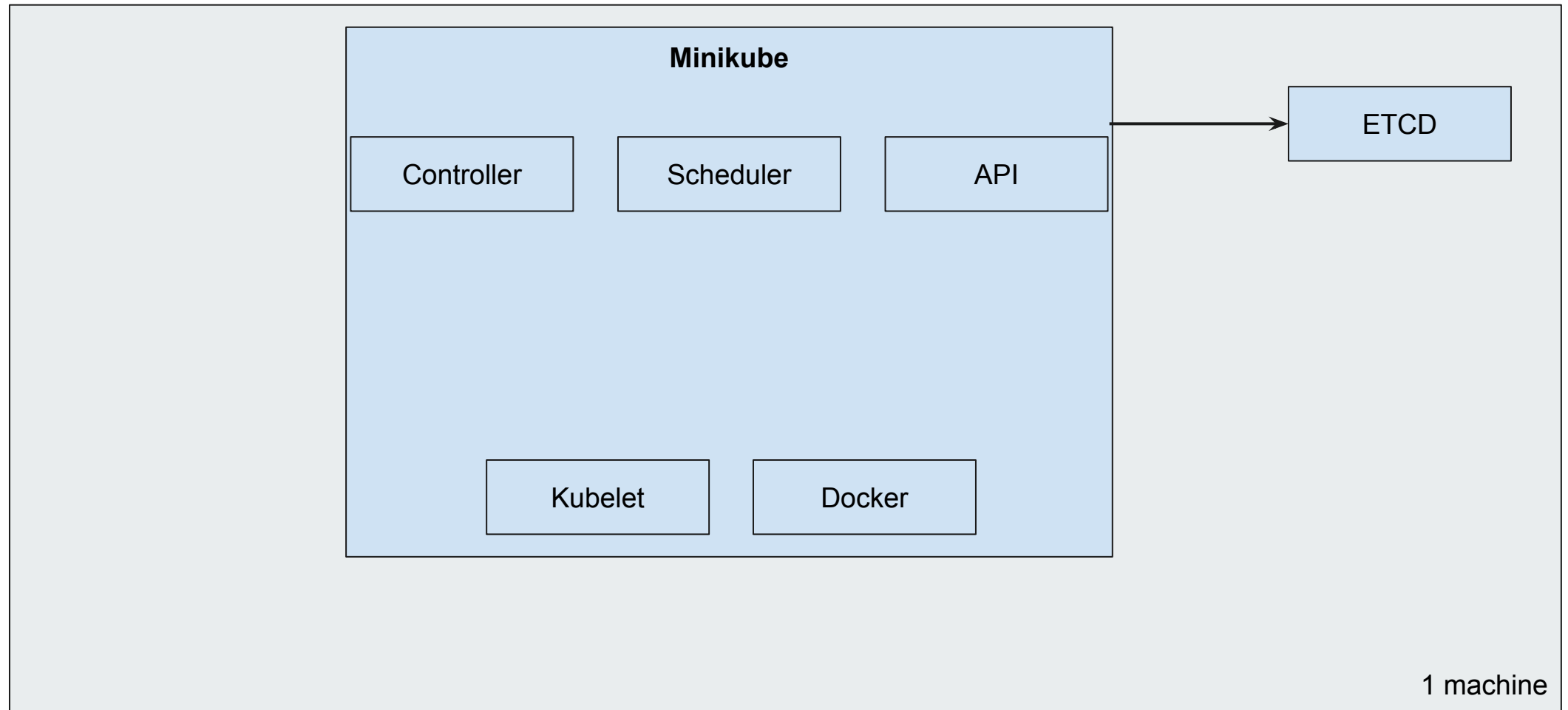
Shows the nodes of your cluster

```
$ kubectl version
Client Version: version.Info{Major:"1", Minor:"2", GitVersion:"v1.2.4", GitTreeState:"clean"}
Server Version: version.Info{Major:"1", Minor:"2", GitVersion:"v1.2.4", GitTreeState:"clean"}
```
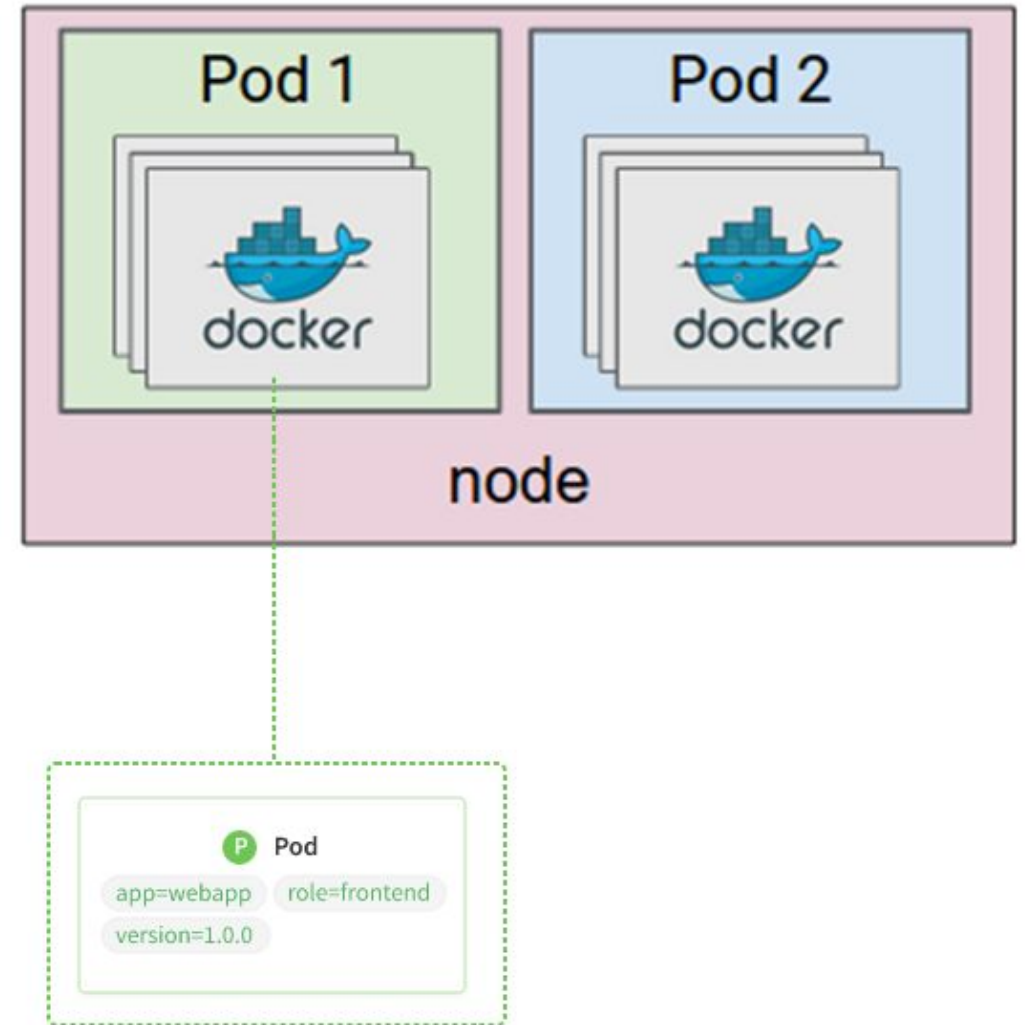
Our minikube environment

Next up

We will start with the 3 basic kubernetes concepts:
- Pods
- Replication controllers
- Services

Kubernetes Pods

- A pods is the smallest deployable unit in Kubernetes.

- A pod can contain one or more Docker containers.

- Every Pod had an own IP address, and containers in a Pod can access each other through localhost

Pod commands

```
$ cd examples
$ kubectl create -f nginx.pod.yaml
pod "nginx" created
```

```
$ kubectl get pods
NAME        READY       STATUS              RESTARTS    AGE
nginx       0/1         ContainerCreating   0           4s
```

```
$ kubectl describe pod nginx
Name:       nginx
Namespace:  default
Node:
gke-cluster-1-default-pool-1cf4645c-9fa5/10.132.0.2
Start Time: Mon, 23 May 2016 23:44:50 +0200
...
```

```
$ cat nginx.pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
```

Kubernetes Replication controllers

- Replication controllers can create, scale control pods.

- Creating a replication controller is almost similar as creating a pod, except that you can chose the replication count.

- Kubernetes will do its best to maintain the desired replication count.
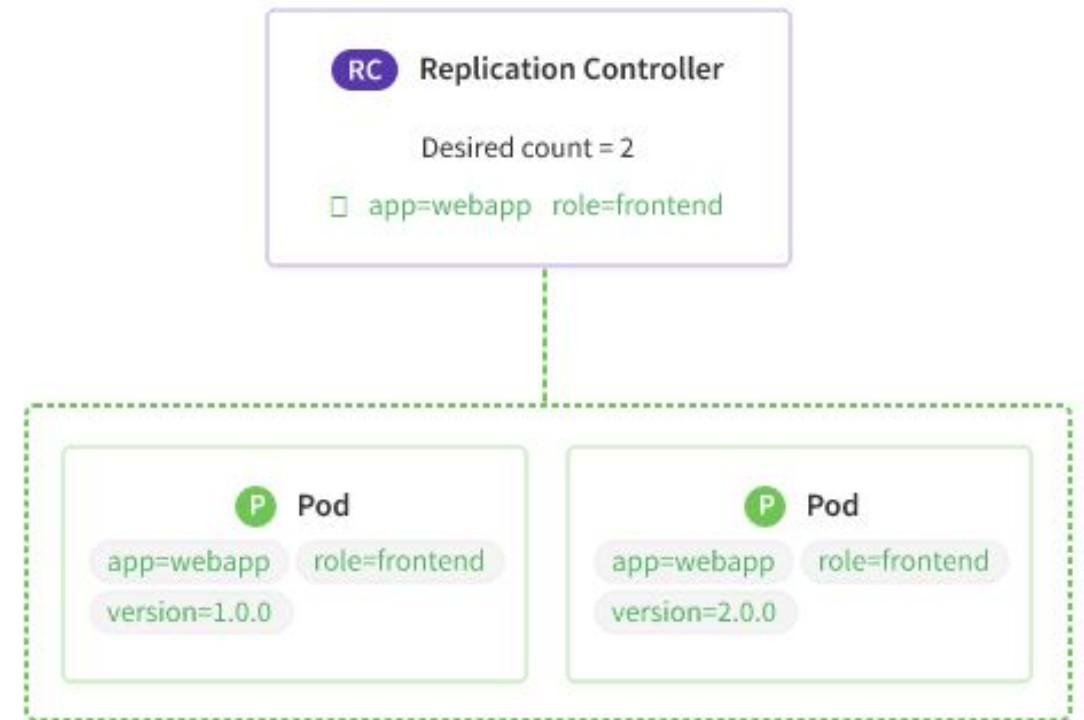
Replication controller commands

```
$ cd examples
$ kubectl create -f nginx.rc.yaml
replicationcontroller "nginx" created
```

```
$ kubectl get rc
NAME        DESIRED   CURRENT    AGE
nginx       2         2          25s
```

```
$ kubectl get pods
NAME            READY      STATUS     RESTARTS    AGE
nginx           1/1        Running    0           15m
nginx-szq0g     1/1        Running    0           55s
```

```
$ cat nginx.rc.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
```

Kubernetes services

- Kubernetes services provide a stable IP address to reach the pods.

- A Kubernetes service serves as a load balancer, traffic shifter and/or service discovery.

- The kubernetes service is connected to pods with the use of selectors.

- Kubernetes services can also be accessed with DNS (by using the plug-in)
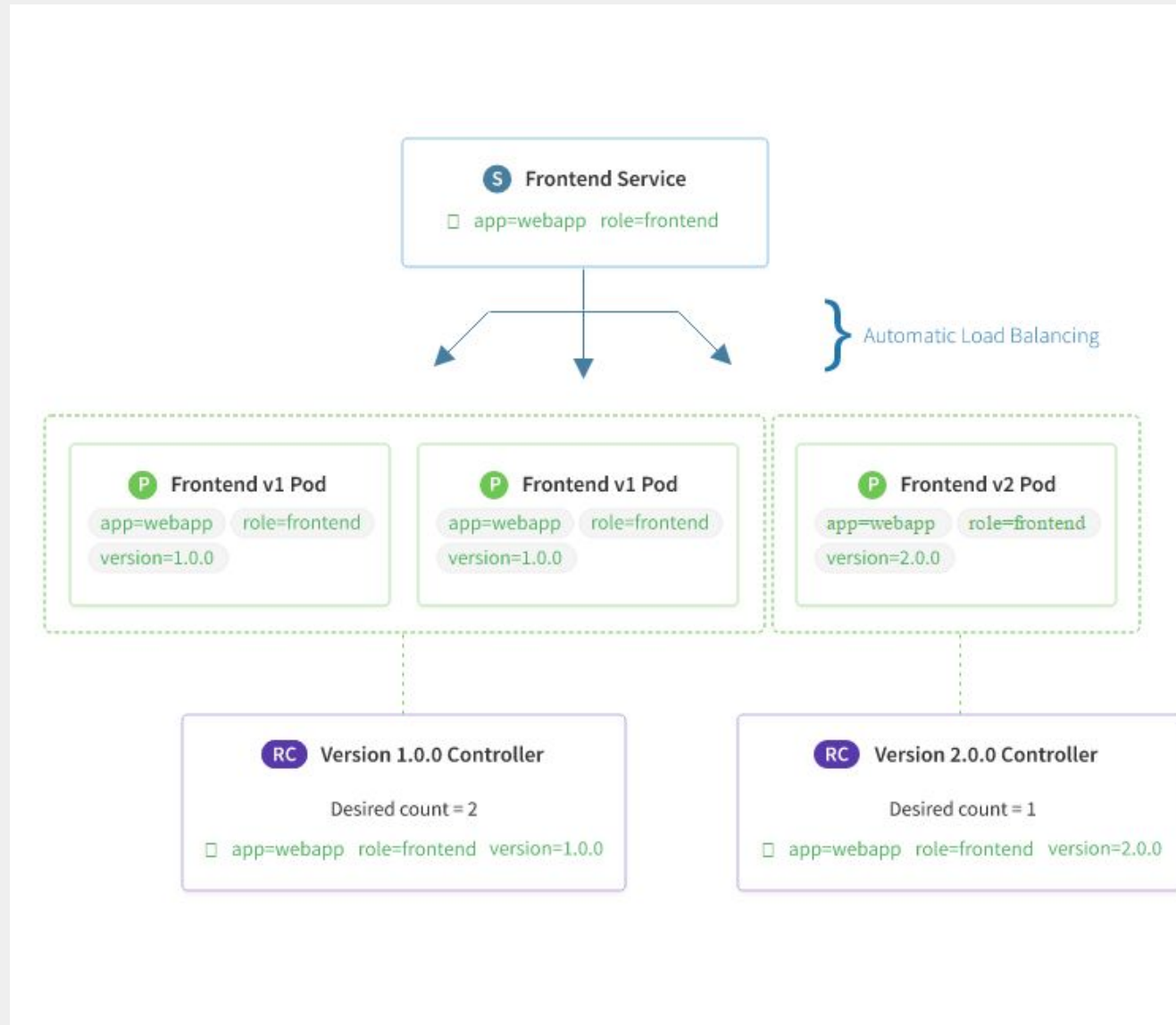
Kubernetes services commands

```
$ cd examples
$ kubectl create -f nginx.service.json
service "nginx" created
```

```
$ kubectl get service
NAME            CLUSTER-IP       EXTERNAL-IP      PORT(S)     AGE
nginx-service   10.15.245.215    104.155.2.94     80/TCP      2m
```

```
$ kubectl get endpoints
NAME            ENDPOINTS                    AGE
nginx-service   10.12.1.4:80,10.12.2.4:80    1m
```

★ The Info Support network only allows access to port 80 and 443. So make sure to only pick those ports.

```
$ cat nginx.service.json
{
    "kind": "Service",
    "apiVersion": "v1",
    "metadata": {
        "name": "nginx-service"
    },
    "spec": {
        "selector": {
            "app": "nginx"
        },
        "type": "LoadBalancer",
        "ports": [
            {
                "protocol": "TCP",
                "port": 80,
                "targetPort": 80
            }
        ]
    }
}
```

Exercise: Deploy **Jenkins** in Kubernetes

To summarize:

```
$ kubectl create -f <filename>
```

```
$ kubectl get pod                    / $ kubectl get po
$ kubectl get replicationcontroller / $ kubectl get rc
$ kubectl get service
$ kubectl get endpoints              / $ kubectl get ep
```

```
$ kubectl describe pod ...
$ kubectl describe rc ...
$ kubectl describe service ...
```

```
$ kubectl delete pod ...
$ kubectl delete rc ...
$ kubectl delete service ...
```

For more information:
- **$ kubectl --help**
- http://kubernetes.io/docs/user-guide/

★ There are examples available in the examples directory on the Desktop.

Solution: Deploy **Jenkins** in Kubernetes

```
$ cat jenkins.rc.yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: jenkins
spec:
  replicas: 2
  selector:
    app: jenkins
  template:
    metadata:
      name: jenkins
      labels:
        app: jenkins
    spec:
      containers:
      - name: jenkins
        image: jenkins
        ports:
        - containerPort: 80
```

```
$ cat jenkins.service.json
{
    "kind": "Service",
    "apiVersion": "v1",
    "metadata": {
        "name": "jenkins-service"
    },
    "spec": {
        "selector": {
            "app": "jenkins"
        },
        "type": "LoadBalancer",
        "ports": [
            {
                "protocol": "TCP",
                "port": 80,
                "targetPort": 8080
            }
        ]
    }
}
```

Delete pods, rc or services

```
$ kubectl delete pod <pod-name>
```

```
$ kubectl delete rc <rc-name>
```

```
$ kubectl delete service <service-name>
```

Next up

- Deployment
- Container properties
- DNS

Kubernetes Deployments

- Kubernetes Deployments are a declarative way of deploying your pods.

- Instead of running the **kubectl create -f** command, you can also deploy pods through Kubernetes deployments.

- Kubernetes Deployments can be rolled back, paused and resumed.

```
# to create a deployment
$ kubectl create -f nginxv1.deployment.yaml --record
```

```
$ cat nginxv1.deployment.yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Updating Kubernetes deployments

```
# to update a deployment
$ kubectl apply -f nginxv2.deployment.yaml --record
```

```
# to view the deployment status
$ kubectl get deployment
```

```
# to check the deployment history
$ kubectl rollout history deployment nginx-deployment
deployments "nginx-deployment":
REVISION CHANGE-CAUSE
1        kubectl create -f nginxv1.deployment.yaml --record
2        kubectl apply -f nginxv2.deployment.yaml
```

```
# to rollback
$ kubectl rollout undo deployment/nginx-deployment --to-revision=1
```

```
$ cat nginxv2.deployment.yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.9.1
        ports:
        - containerPort: 80
```

Container properties

- The container field in the pod configuration supports a lot of properties, for example:
  - args - Container arguments
  - ports - Container ports
  - env - Environment variables
  - imagePullPolicy - Defines which strategy to use to pull an image
  - volumeMounts - To mount volumes

A full list can be found at:
http://kubernetes.io/docs/api-reference/v1/definitions/#_v1_container

```
$ cat nginx.pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
```

DNS

- Your Kubernetes cluster is configured with DNS support (SkyDNS)

- The default configuration for DNS is:
  <servicename>.<namespace>.svc.cluster.local

- So you can access nginx with:
  nginx-service.default.svc.cluster.local

- To find out which DNS records exists, use the following command:
  $ kubectl get endpoints

Exercise: Deploy MySQL + PhpMyAdmin in Kubernetes

To summarize:

```
# to create a deployment
$ kubectl create -f nginxv1.deployment.yaml --record
```

```
# to update a deployment
$ kubectl apply -f nginxv2.deployment.yaml --record
```

```
# to view the deployment status
$ kubectl get deployment
```

- The default configuration for DNS is:

  <servicename>.<namespace>.svc.cluster.local

For more information:
- **$ kubectl --help**
- http://kubernetes.io/docs/user-guide/
- http://kubernetes.io/docs/api-reference/v1/definitions/#_v1_container

★ The container field in the pod configuration supports a lot of properties, for example:
  ○ args - Container arguments
  ○ ports - Container ports
  ○ env - Environment variables
  ○ volumeMounts - To mount volumes

Next up

We've finished the basics for Kubernetes, but we still have advanced topics to cover:
- High availability
- Liveness probes / Readiness probes (health checks)
- Resource Quotas
- Automatic scaling
- Node selectors
- Static pods / daemon sets
- Volumes / Secrets
- Debugging
- Namespaces
- Identity / Authorization