Kubernetes

Training

Who am I? Guillem Hernández Sola

- DevOPS
  - Code
    - `Code development and review, version control tools, code merging`
  - Build
    - `Continuous integration tools, build status`
  - Test
    - `Test and results determine performance`
  - Package
    - `Artifact repository, application pre-deployment staging`
  - Release
    - `Change management, release approvals, release automation`
  - Configure
    - `Infrastructure configuration and management, Infrastructure-as-Code tools`
  - Monitor
    - `Applications performance monitoring, end-user experience`

- Software Quality Assurance Engineer, Test Automation Engineer
  - specialised in WebDriver and Appium
  - Certified ISTQB

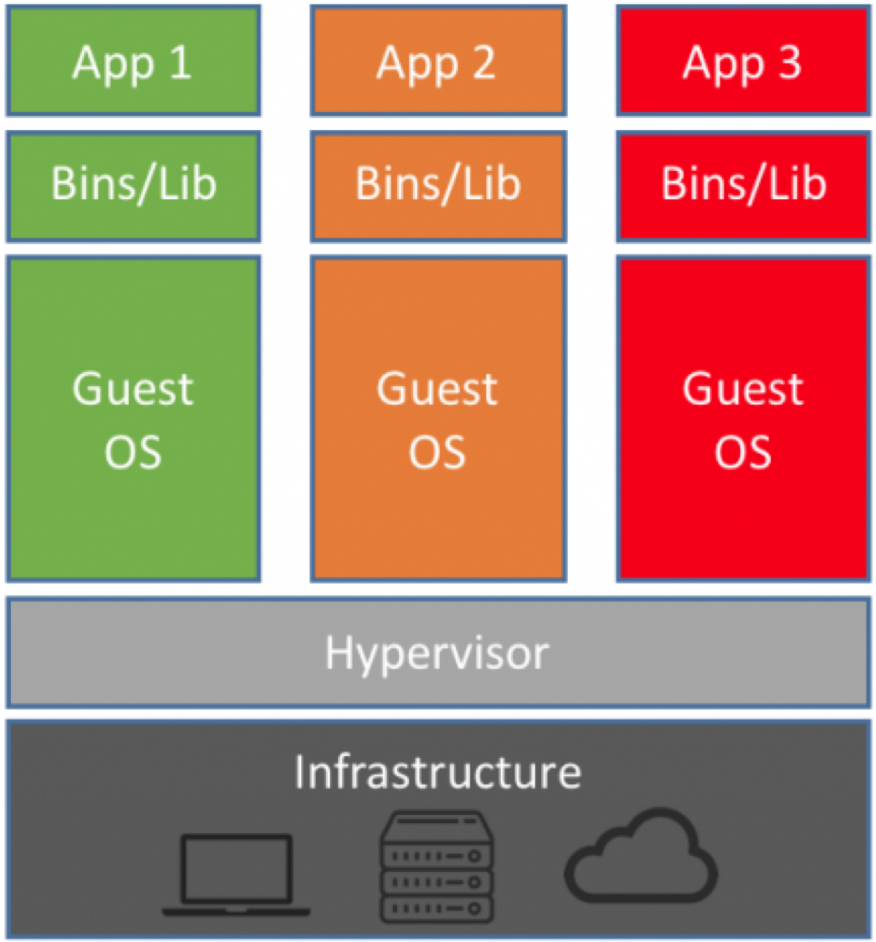
Presentation

- Some questions:
  - Name, Role in the company
  - Motivation: why are you here?
  - Technologies used in your projects
  - Experience and background

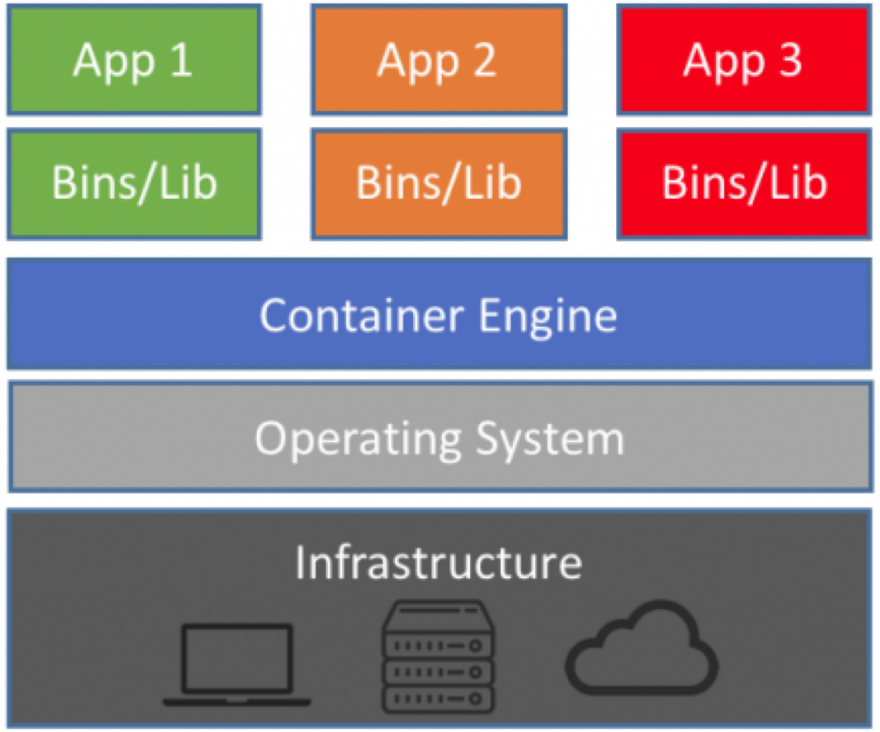
Agenda

- Complete Training agenda (4 days)
  - Containers management
    - Create a container
    - Package a container using Docker
    - Remote Docker images management (Google Container Registry, AWS)
  - Kubernetes and Remote Docker images management
    - Overview: What is Kubernetes?
    - Kubernetes Architecture
    - Provision a complete Kubernetes cluster using remote repositories
    - Deploy and manage Docker containers using kubectl
    - Convert applications into microservices using Kubernetes Deployments and Services
  - Deploying to Kubernetes
    - Create a Kubernetes deployment
    - Trigger, pause, resume, and rollback updates
    - Understand and build canary deployments
  - Continuous Deployment with Jenkins and Docker
    - Provision Jenkins in your Kubernetes cluster
    - Create a Jenkins pipeline
    - Implement a canary deployment using Jenkins

Virtual Machine vs Containers



Machine Virtualization



Containers

Docker

- Docker is the most popular container software
  - An alternative to Docker is rkt (also supported on Kubernetes)
- Docker engine
  - Docker runtime
  - Software to make run docker images
- Docker Hub
  - Online service to store and fetch docker images
  - Also allows you to build docker images online

Docker benefits

- Isolation: you ship a binary with all the dependencies
  - » No more it works on my machine, but not in production
- Closer parity between dev, QA and production environments
- Docker makes developments team able to ship faster
- You can run the same docker image, unchanged, on laptops, data center VMs and Cloud providers
- Docker uses Linux Containers (a kernel feature) for operating system-level isolation

Create my environment for the training

- Checkout the example code from:
  - https://bitbucket.org/itnove/startusingkubernetes
- Just clone the repo
  - git clone https://bitbucket.org/itnove/startusingkubernetes
- And then follow the commands:
  - cd startusingkubernetes
  - vagrant up
- To check if everything works, just enter to Vagrant Box using ssh:
  - your@terminal $ vagrant ssh
  - vagrant@k8s $ date (check if it works)

Checking the installation of the app

- Enter to the environment using ssh:
  - your@terminal $ vagrant ssh (from the project root startusingkubernetes)
- Checking a regular code from the Example NodeJS App:
  - vagrant@k8s $ git clone https://bitbucket.org/itnove/the-example-app-nodejs.git
  - vagrant@k8s $ cd the-example-app.nodejs
  - vagrant@k8s $ npm install
- To start the express server, run the following
  - npm run start:dev
- On the host machine, type http://locahost:3000 and you have to see a website
- After running all this, check your docker installation
  - vagrant@k8s $ docker version

Dockerizing the app

- Start building the container:
  - vagrant@k8s $ git clone https://bitbucket.org/itnove/the-example-app-nodejs.git
  - vagrant@k8s $ cd the-example-app.nodejs
  - vagrant@k8s $ docker build -t the-example-app.nodejs .
  - vagrant@k8s $ docker run -p 3000:3000 -d the-example-app.nodejs
- Uploading your container to the Docker (you need hub.docker.com credentials)
  - vagrant@k8s $ docker login
  - vagrant@k8s $ docker images
  - vagrant@k8s $ docker tag bb38976d03cf yourhubusername/verse_gapminder:firsttry
  - vagrant@k8s $ docker push yourhubusername/verse_gapminder:firsttry

Kubernetes

Day 1

What is Kubernetes?

- The question:
  - » What is Kubernetes for you?

What is Kubernetes (1/2)

- Kubernetes is an open source orchestration system for Docker Containers
  - » Schedule containers on a cluster of machines
  - » Run multiple containers on one machine
  - » Run long services (like web applications)
  - » Kubernetes will manage the state of these containers
    - × `Can start the container on specific nodes`
    - × `Will restart a container when it gets killed`
    - × `Can move containers from one node to another node`

What is Kubernetes (2/2)

- Instead of just running a few docker containers on one host manually, Kubernetes is a platform that will manage the containers for you
- Kubernetes clusters can start with one node until thousands of nodes
- Some other popular docker orchestrators are:
  - Docker Swarm
  - Mesos

Kubernetes Advantatges

- Kubernetes runs anywhere
  - On-premise (your metal)
  - Public (Google Cloud, AWS)
  - Hybrid: Public & Private
- Highly modular
- Open source
- Great Community
- Backed by Google

Kubernetes Setup

- Kubernetes should really be able to run anywhere
- There are more integration for some Cloud Providers (AWS & GKE)
  - » Volumes and external load balancers work only with supported cloud providers
- We will first use minikube to quickly spin up a local single machine with a Kubernetes Cluster
- If we have time, we will see how to spin up a cluster on AWS using kops
  - » kops can be used to spin up a highly available production cluster

Minikube setup – Some comments

- Minikube is a tool that makes it easy to run Kubernetes locally
- Minikube runs a single-node Kubernetes cluster inside a Linux VM
  - » It is NOT the configuration we downloaded using Vagrant
- It is aimed on users who want to just test it out or use it for development
- It cannot spin up a production cluster, it's a one node machine with no high availability
  - » Check Github for Minikube

Our first example

- We spin up our Vagrant environment
  - vagrant@k8s $ vagrant ssh
  - vagrant@k8s $ kubectl apply -f https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')
  - vagrant@k8s $ kubectl taint nodes --all node-role.kubernetes.io/master-
- When it is ready, we type the following command:
  - vagrant@k8s $ kubectl run hello-minikube --image=gcr.io/google_containers/echoserver:1.4 --port=8080
  - vagrant@k8s $ kubectl expose deployment hello-minikube --type=NodePort

Kubectl first commands

- vagrant@k8s $ kubectl get nodes
- vagrant@k8s $ kubectl get pods
- vagrant@k8s $ kubectl get services

Our app on our cluster

- Let’s run our newly build application on the new Kubernetes Cluster
- Before we can launch a container based on the image, we need to create a pod definition
  - A pod describes an application running on Kubernetes
  - A pod can contain one or more tightly coupled containers, that make up the app
    - `Those apps can easily communicate which each other using their local por numbers`
  - Our app only has one container

Create a pod

- Create a file 01-pod-helloworld.yml with the pod definition

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nodehelloworld.itnove.com
  labels:
    app: helloworld
spec:
  containers:
  - name: k8s-demo
    image: itnove/k8s-example
    ports:
    - name: nodejs-port
      containerPort: 3000
```

Useful commands

Command	Description
kubectl get pod	Get information about all running pods
kubectl describe pod <pod>	Describe one pod
kubectl expose pod <pod> --port=444 --name=frontend	Expose the port of a pod (creates a new service)
kubectl port-forward <pod> 8080	Port forward the exposed pod port to your local machine
kubectl attach <podname> -i	Attach to the pod
kubectl exec <pod> -- command	Execute a command on the pod
kubectl label pods <pod> mylabel=awesome	Add a new label to a pod
kubectl run -i --tty busybox --image=busybox --restart=Never -- sh	Run a shell in a pod - very useful for debugging

Running our pod on Kubernetes

- We will run it using the following commands:
  - vagrant@k8s $ kubectl get node
  - vagrant@k8s $ git clone https://bitbucket.org/itnove/startusingkubernetes.git
  - vagrant@k8s $ cd startusingkubernetes/examples
  - vagrant@k8s $ cat 01-pod-helloworld.yml
  - vagrant@k8s $ kubectl create –f 01-pod-helloworld.yml
  - vagrant@k8s $ kubectl get pod
  - vagrant@k8s $ kubectl describe pod nodehelloworld.itnove.com
  - vagrant@k8s $ kubectl port-forward nodehelloworld.itnove.com 8081:3000 (*)
  - vagrant@k8s $ kubectl expose pod nodehelloworld.itnove.com –type=NodePort –name nodehelloworld-service
  - vagrant@k8s $ kubectl get services

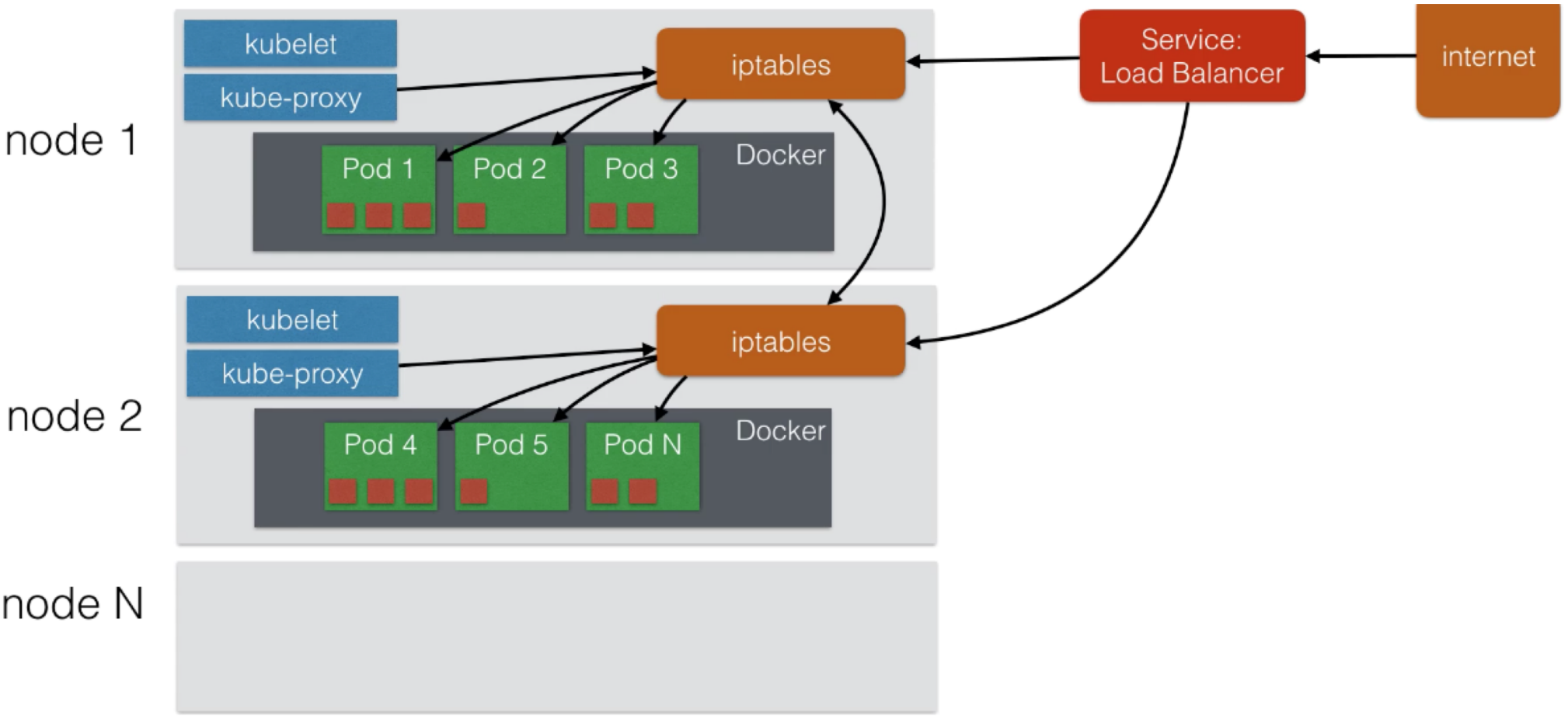
More commands

- More commands:
  - vagrant@k8s $ kubectl exec nodehelloworld.itnove.com -- ls
  - vagrant@k8s $ kubectl describe service nodehelloworld-service
    - `Get ip:port from the cluster`
  - vagrant@k8s $ kubectl run –i –tty busybox –image=busybox –restart=Never – sh
    - `I connect via telnet to ip:3000 from the other pod`

Kubernetes Basics

Training

Architecture Overview



Scaling Pods (1/2)

- If your application is stateless you can horizontally scale it
  - Stateless = your application doesn't have a state, it doesn't write any local files / keep local sessions
  - All traditional databases (MySQL, Postgres) are stateful, they have database files that can't be split over multiple instances
- Most web applications can be made stateless:
  - Session management needs to be done outside the container
  - Any files that need to be saved can't be saved locally on the container
- Our example app is stateless, if the same app would run multiple times and it doesn't change state
- Best practices at 12factor.net

Scaling pods (2/2)

- Later we will explain how to use volumes to still run stateful apps
  - Those stateful apps can't horizontally scale, but you can run them in a single container and vertically scale (allocate more CPU / Memory / Disk)
- Scaling in Kubernetes can be don using the Replication Controller
- The replication controller will ensure a specified number of pod replicas will run at all time
- A pods created with the replica controller will automatically be replaced if they fail, get deleted or are terminated
- Using the replication controller is also recommended if you just want to make sure 1 pod is always running even after reboots
  - You can then run replication controller with just 1 replica
  - This makes sure that the pod is always running

Replication controller example

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: helloworld-controller
spec:
  replicas: 2
  selector:
    app: helloworld
  template:
    metadata:
      labels:
        app: helloworld
    spec:
      containers:
      - name: k8s-demo
        image: itnove/k8s-example
        ports:
        - name: nodejs-port
          containerPort: 3000
```

Replication Controller Examples

- Commands:
  - vagrant@k8s $ kubectl create –f 02-replication-controller.yml
  - vagrant@k8s $ kubectl get pods
  - vagrant@k8s $ kubectl describe pods
  - vagrant@k8s $ kubectl delete pod helloworld-controller-wedsa2
  - vagrant@k8s $ kubectl scale –replicas=4 –f 02-replication-controller.yml
  - vagrant@k8s $ kubectl get pods
  - vagrant@k8s $ kubectl get rc
  - vagrant@k8s $ kubectl scale –replcias=1 rc/helloworld-controller
  - vagrant@k8s $ kubectl get pods

Replication Set

- Replica Set is the next-generation Replication Controller
- It supports a new selector that can do selection based on filtering according to a set of valuers
  - » E.g. ”environment” either “dev” or “qa”
  - » Not only based on equality, like the Replication Controller
    - × `E.g.”environment” == “dev”`
- This Replica Set, rather than the Replication Controller, is used by the Deployment Object

Deployments (1/2)

- A deployment declaration in Kubernetes allows you to do app deployments and updates
- When using the deployment object, you define the state of your application
  - » Kubernetes will then make sure the clusters matches your desired state
- Just using the replication controller or replication set might be cumbersome to deploy apps
  - » The Deployment Object is easier to use and gives you more possibilities

Deployments (2/2)

- With a deployment object you can:
  - Create a deployments
  - Update a deployment
  - Do rolling updates
  - Roll back to a previous version
  - Pause / Resume a deployment

Deployment example

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: helloworld-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: helloworld
    spec:
      containers:
      - name: k8s-demo
        image: itnove/k8s-example
        ports:
        - name: nodejs-port
          containerPort: 3000
```

Deployment useful commands

Command	Description
kubectl get deployments	Get information on current deployments
kubectl get rs	Get information about the replica sets
kubectl get pods --show-labels	get pods, and also show labels attached to those pods
kubectl rollout status deployment/helloworld-deployment	Get deployment status
kubectl set image deployment/helloworld-deployment k8s-demo=k8s-demo:2	Run k8s-demo with the image label version 2
kubectl edit deployment/helloworld-deployment	Edit the deployment object
kubectl rollout status deployment/helloworld-deployment	Get the status of the rollout
kubectl rollout history deployment/helloworld-deployment	Get the rollout history
kubectl rollout undo deployment/helloworld-deployment	Rollback to previous version
kubectl rollout undo deployment/helloworld-deployment --to-revision=n	Rollback to any version version

Deployment example commands (1/2)

- Commands
  - vagrant@k8s $ kubectl create –f 03-deployment.yml
  - vagrant@k8s $ kubectl get deployments
  - vagrant@k8s $ kubectl get rs
  - vagrant@k8s $ kubectl get pods –show-labels
  - vagrant@k8s $ kubectl rollout status deployment/helloworld-deployment
- I will push a new version from our demo app
  - vagrant@k8s $ kubectl explose deployment helloworld-deployment –type=NodePort
  - vagrant@k8s $ kubectl get service
  - vagrant@k8s $ kubectl describe service helloworld-deployment (check Endpoints)

Deployment example commands (2/2)

- Commands for the rollout:
  - vagrant@k8s $ kubectl set image deployment/helloworld-deployment k8s-demo=itnove/k8s-example:new_version
  - vagrant@k8s $ kubectl rollout status deployment/helloworld-deployment
  - vagrant@k8s $ kubectl get pods
  - vagrant@k8s $ kubectl rollout history deployment/helloworld-deployment
  - vagrant@k8s $ kubectl rollout undo deployment/helloworld-deployment
  - vagrant@k8s $ kubectl rollout status deployment/helloworld-deployment
  - vagrant@k8s $ kubectl edit deployment/helloworld-deployment (change revisionHistoryLimit to 100)
  - vagrant@k8s $ kubectl set image deployment/helloworld-deployment k8s-demo=itnove/k8s-example:older_version
  - vagrant@k8s $ kubectl rollout undo deployment/helloworld-deployment –to-revision=3

Services

- Pods are very dynamic, they come and go on the Kubernetes Cluster
  - When using a Replication Controller, pods are terminated and created during scaling operations
  - When using Deployment, when updating the image version, pods are terminated and new pods take the place of older pods
- That is why Pods should never be accessed directly, but always through a Service
- A service is the logical bridge between the “mortal” pods and other services or end-users

Services (1/2)

- When using the “kubectl expose” command earlier, you created a new Service for your pod, so it could be accessed externally
- Creating a service will create an endpoint for your pod(s):
  - A ClusterIP: a virtual IP address only reachable from within the cluster (this is the default)
  - A NodePort: a port that is the same on each node that is also reachable externally
  - A LoadBalancer: a LoadBalancer created by the cloud provider that will route external traffic to every node on the NodePort (ELB on AWS)

Services (2/2)

- The options just shown only allow you to create virtual IPs or ports
- There is also a possibility to use DNS names
  - ExternalName can provide a DNS name for the service
  - E.g. for service discovery using DNS
  - This only works when the DNS add-on is enabled

Service example

- Example of a Service Definition (also created using kubectl expose)

```yaml
apiVersion: v1
kind: Service
metadata:
  name: helloworld-service
spec:
  ports:
  - port: 31001
    nodePort: 31001
    targetPort: nodejs-port
    protocol: TCP
  selector:
    app: helloworld
  type: NodePort
```

Services commands

- Commands:
  - vagrant@k8s $ kubectl create –f 01-helloworld.yml
  - vagrant@k8s $ kubectl get pods
  - vagrant@k8s $ kubectl describe nodehelloworld.itnove.com
  - vagrant@k8s $ kubectl create –f 04-services.yml
  - vagrant@k8s $ kubectl delete svc helloworld-service
  - vagrant@k8s $ kubectl create –f 04-services.yml
  - vagrant@k8s $ kubectl describe svc helloworld-service

Labels

- Labels are key/value pairs that can be attached to objects
  - Labels are like tags in AWS or other cloud providers, used to tag resources
- You can label your objects, for instance your pod, following an organizational structure
  - Key: environment – Value: dev / staging / qa / prod
  - Key: environment – Value: engineering / finance / marketing
- Labels are not unique and multiple labels can be added to one object
- Once labels are attached to an object, you can use filters to narrow down results. This is called Label selectors
- Using Label Selectors, yo can use matching expressions to match labels

Node Labels

- You can also use labels to tag nodes
- Once nodes are tagged, you can use label selectors to let pods only run on specific nodes
- There are 2 steps required to run a pod on a specific set of nodes:
  - First you tag the node
  - Then you add a nodeSelector to your pod configuration

Node Selector using Labels

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: helloworld-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: helloworld
    spec:
      containers:
      - name: k8s-demo
        image: itnove/k8s-example
        ports:
        - name: nodejs-port
          containerPort: 3000
      nodeSelector:
        hardware: high-spec
```

NodeSelector Commands

- Commands:
  - vagrant@k8s $ kubectl get nodes –show-labels
  - vagrant@k8s $ kubectl create –f 05-nodeselector.yml
  - vagrant@k8s $ kubectl get deployments
  - vagrant@k8s $ kubectl describe pod
  - vagrant@k8s $ kubectl label nodes name_of_the_node hardware=high-spec

Health Checks

- If your application malfunctions, the pod and container can still be running, but the application might not work anymore
- To detect and resolve problems with your application, you can run health checks
- You can run 2 different type of health checks
  - Running a command in the container periodically
  - Periodic checks on a URL (HTTP)
- The typical production application behind a load balaner should always have health checks implemented in some way to ensure availability and resiliency of the app

Health check example

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: helloworld-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: helloworld
    spec:
      containers:
      - name: k8s-demo
        image: itnove/k8s-example
        ports:
        - name: nodejs-port
          containerPort: 3000
        livenessProbe:
          httpGet:
            path: /
            port: nodejs-port
          initialDelaySeconds: 15
          timeoutSeconds: 30
```

Health Check Commands

- Commands:
  - vagrant@k8s $ kubectl create –f 06-healthcheck.yml
  - vagrant@k8s $ kubectl get pods
  - vagrant@k8s $ kubectl describe helloworld-deployment-pod-name
  - vagrant@k8s $ kubectl edit deployment/helloworld-deployment

Readiness Probe (1/2)

- Besides livenessProbes, you can also use readinessProbe on a container within a Pod
- livenessProbes: indicates whether a container is running
  - » If the check fails, the container will be restarted
- readinessProbes: indicates whethere the container is ready to serve requests
  - » If the check fails, the container will not be restarted, but the Pod's IP address will be removed from the Service, so it will no serve any requests anymore

Readiness Probe (2/2)

- The readiness test will make sure that at startup, the pod will only receive traffic when the test succeeds
- You can use these probes in conjuction, and you can configure different tests for them
- I fyou container always exits when something goes wrong, you don’t need a livenessProbe
- In general, you configure both the livenessProbe and the readinessProbe

Liveness Readiness Example

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: helloworld-readiness
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: helloworld
    spec:
      containers:
      - name: k8s-demo
        image: itnove/k8s-example
        ports:
        - name: nodejs-port
          containerPort: 3000
        livenessProbe:
          httpGet:
            path: /
            port: nodejs-port
          initialDelaySeconds: 15
          timeoutSeconds: 30
        readinessProbe:
          httpGet:
            path: /
            port: nodejs-port
          initialDelaySeconds: 15
          timeoutSeconds: 30
```

Liveness Readiness Commands

- Commands:
  - vagrant@k8s $ kubectl create –f 06-healthcheck.yml && watch -n1 kubectl get pods
  - vagrant@k8s $ kubectl create –f 07-readiness.yml && watch –n1 kubectl get pods

Pod state (1/2)

- Pod status field: high level status
- Pod condition: the condition of the pod
- Container State: state of container(s) itself
- Pods have a status field, you can see it when you do kubectl get pods
- For example if a pod has Running status
  - This meand that the pod has been bound to a node
  - All containers have been created
  - At least one container is still running, or is starting / restarting

Pod state (2/2)

- Other valid statuses are:
  - » Pending: Pod has been accepted but is not running
    - × `Happens when the container image is still downloading`
    - × `If the pod cannot be scheduled because of resource constraints, it will also be in this status`
  - » Succeeded: All containers within this pod have been terminated successfully and will not be restarted
  - » Failed: All containers within this pod have been Terminated, and at least one container returned a failure code
    - × `The failure code is the exit code of the process when a container terminates`
  - » Unknown: The state of the pod couldn't be determined
    - × `A network error might have been occurred (for example the node where the pod is running on is down)`

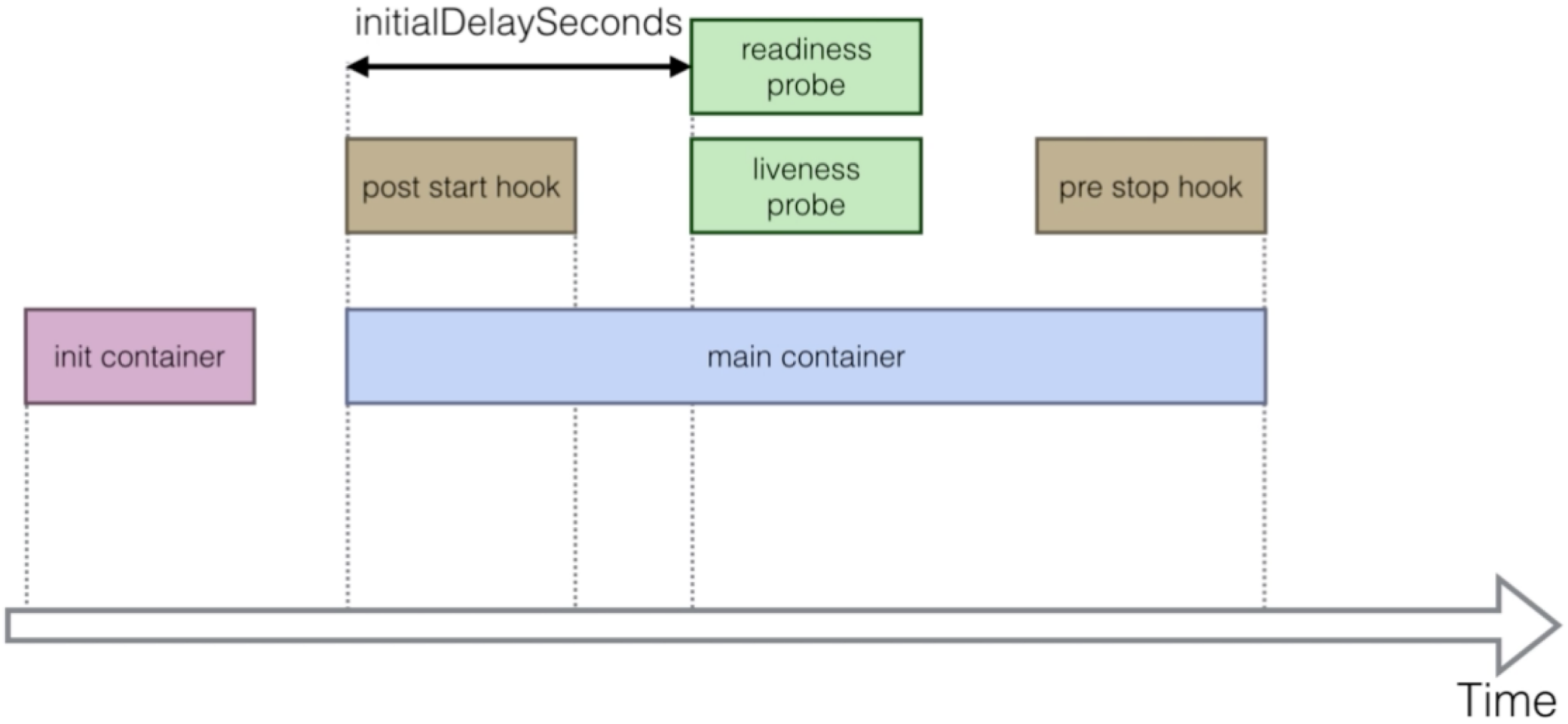
Pod Conditions

- 5 types of PodConditions:
  - PodScheduled: the pod has been scheduled to a noe
  - Ready: Pod can serve requests and is going to be added to matching Services
  - Initialized: the initialization containers have been started successfully
  - Unschedulable: the Pod can't be scheduled (for resource constraints for example)
  - ContainersReady: all containers in the pod are ready

Container state

- There’s also a container state coming directly from Docker Engine
- Container state can be Running, Terminated or Waiting

Pod Lifecycle



Pod LifeCycle Example

```yaml
kind: Deployment
apiVersion: apps/v1beta1
metadata:
  name: lifecycle
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: lifecycle
    spec:
      initContainers:
      - name:           init
        image:          busybox
        command:       ['sh', '-c', 'sleep 10']
      containers:
      - name: lifecycle-container
        image: busybox
        command: ['sh', '-c', 'echo $(date +%s): Running >> /timing && echo "The app is running!" && /bin/sleep 120']
        readinessProbe:
          exec:
            command: ['sh', '-c', 'echo $(date +%s): readinessProbe >> /timing']
          initialDelaySeconds: 35
        livenessProbe:
          exec:
            command: ['sh', '-c', 'echo $(date +%s): livenessProbe >> /timing']
          initialDelaySeconds: 35
          timeoutSeconds: 30
        lifecycle:
          postStart:
            exec:
              command: ['sh', '-c', 'echo $(date +%s): postStart >> /timing && sleep 10 && echo $(date +%s): end postStart >> /timing']
          preStop:
            exec:
              command: ['sh', '-c', 'echo $(date +%s): preStop >> /timing && sleep 10']
```

Pod Lifecycle commands

- Commands (need 2 terminals from k8s) – screen -r
  - Terminal 1
    - `vagrant@k8s $ watch –n1 kubectl get pods`
  - Terminal 2
    - `vagrant@k8s $ kubectl create –f 08-lifecycle.yml`
    - `vagrant@k8s $ kubectl exec –it lifecycle-name –- tail /timings -f`

Secrets (1/2)

- Secrets provides a way in Kubernetes to distribute credentials, keys, passwords or “secret” data to the pods
- Kubernetes itself uses this Secrets mechanism to provide the credentials to access the internal API
- You can also use the same mechanism to provide secrets to your application
- Secrets is one way to provide secrets, native to Kubernetes
  - » There are still other ways your container can get its secrets if you don’t want to use Secrets (e.g. using an external vault services in your app)

Secrets (2/2)

- Secrets can be used in the following ways:
  - Use secrets as environment variables
  - Use secrets as a file in a pod
    - `This setup uses volumes to be mounted in a container`
    - `In this volume you have files`
    - `Can be used for instance for dotenv files or your app can just read this file`
  - Use an external image to pull secrets (from a private image registry)

Secrets example

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: db-secrets
type: Opaque
data:
  username: cm9vdA==
  password: cGFzc3dvcmQ=
```

Secrets commands

- vagrant@k8s $ kubectl create –f 09-secrets.yml
- vagrant@k8s $ kubectl create -f 10-secrets-volumes.yml
- vagrant@k8s $ kubectl get pods
- vagrant@k8s $ kubectl describe pod helloworld-podname
- vagrant@k8s $ kubectl exec helloworld-deployment-2233 –i –t -- /bin/bash
- Inside the pod $ cat /etc/creds/password

Set up a Wordpress using Secrets

- Not fully working due to stateful (not data persistent yet)
  - examples/wordpress/wordpress-secrets.yml
  - examples/wordpress/wordpress-single-no-volumes.yml
  - examples/wordpress/wordpress-service.yml
- Commands:
  - vagrant@k8s $ kubectl create –f wordpress/wordpress-secrets.yml
  - vagrant@k8s $ kubectl create –f wordpress/wordpress-single-no-volumes.yml
  - vagrant@k8s $ kubectl create –f wordpress/wordpress-service.yml
  - vagrant@k8s $ kubectl delete pod/wordpres-deployment-namepod3121

DNS

- DNS is a built-in service launched automatically using the addon manager
  - The addons are in the /etc/kubernetes/addons directory on master node
- The DNS service can be used within pods to find other services running on the same cluster
- Multiple containers within 1 pod don't need this service, as they can contact each other directly
  - A container in the same pod can connect the port of the other container directly using localhost:port
- To make DNS work, a pod will need a Service definition

DNS example

- Commands:
  - vagrant@k8s $ kubectl create –f 011-service-discovery-secrets.yml
  - vagrant@k8s $ kubectl create –f 012-database.yml
  - vagrant@k8s $ kubectl create –f 013-database-service.yml
  - vagrant@k8s $ kubectl create –f 014-helloworld-db.yml
  - vagrant@k8s $ kubectl create –f 015-helloworld-db-service.yml
  - vagrant@k8s $ kubectl exec database –i –t – mysql –u root –p
  - vagrant@k8s $ kubectl run –i –tty busybox –image=busybox –restart=Never – sh
  - nslookup inside the busybox checking DNS
  - telnet IP 3000 – GET /
  - vagrant@k8s $ kubectl delete pod busybox

ConfigMap

- Configuration parameters that are not secret, can be put in a ConfigMap
- The input is again key-value pairs
- The ConfigMap key-value pairs can then be read by the app using:
  - Environment variables
  - Container commandline arguments in the Pod configuration
  - Using volumes
- A ConfigMap can also contain full configuration files
  - E.g. webserver config file
- This file can then be mounted using volumes where the application expects its config file
- This way you can “inject” configuration settings into containers without changing the container itself

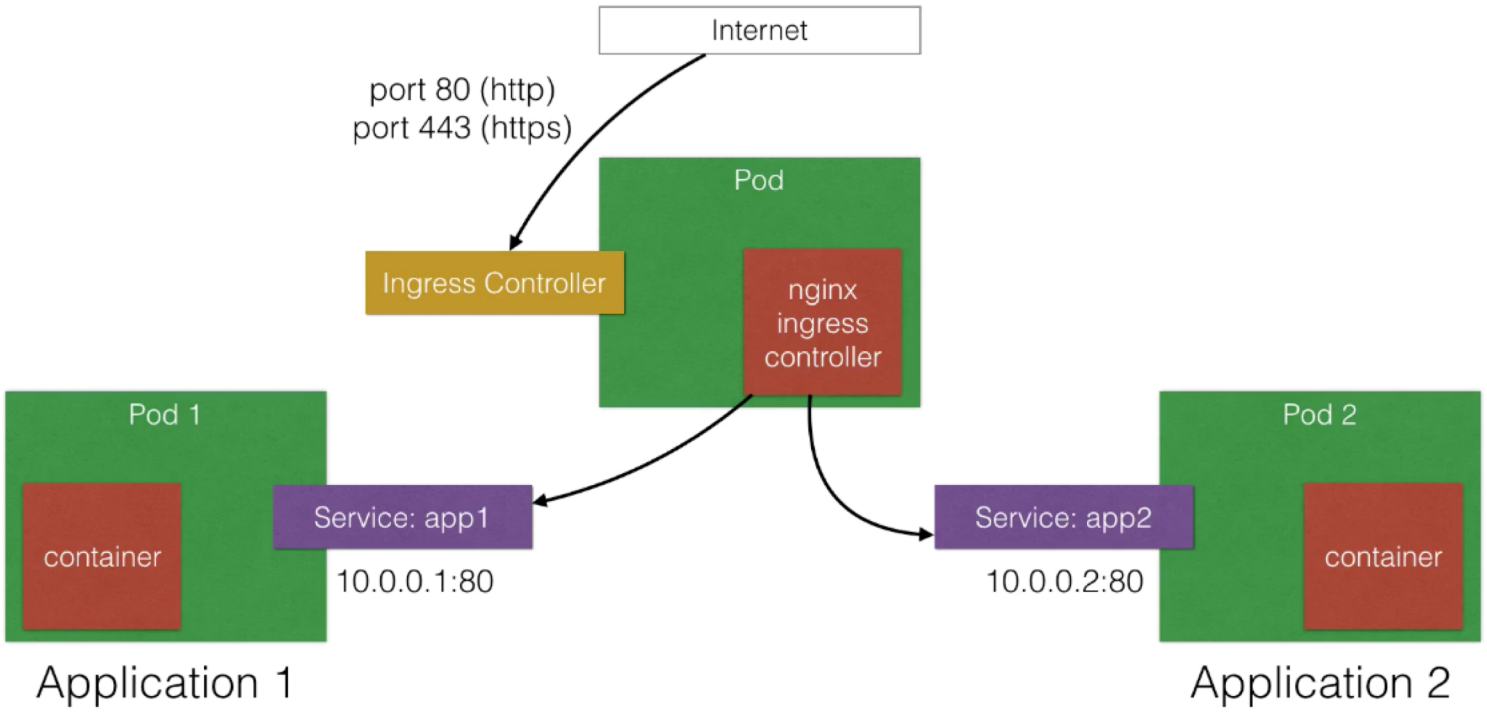
ConfigMap example

- Commands (Before check files form configmap folder):
  - vagrant@k8s $ cat configmap/reverseproxy.conf
  - vagrant@k8s $ kubectl create configmap nginx-config –from-file=configmap/reverseproxy.conf
  - vagrant@k8s $ kubectl get configmap
  - vagrant@k8s $ kubectl get configmap nginx-config –o yaml
  - vagrant@k8s $ kubectl create –f configmap/nginx.yml
  - vagrant@k8s $ kubectl create –f configmap/nginx-service.yml
  - vagrant@k8s $ curl http://ip:port –vvv
  - vagrant@k8s $ kubectl exec –i –t helloworld-nginx –c nginx – bash
  - root@helloworld-nginx # cat /etc/nginx/conf.d/reverseproxy.conf

Ingress Controller

- Ingress is a solution available since Kubernetes 1.1 that allows inbound connections to the cluster
- It is an alternative to the external LoadBalancer and NodePorts
  - » Ingress allows you to easily expose services that need to be accessible from outside to the cluster
- With ingress you can run your own ingress controller (basically a loadbalancer) within the Kubernetes cluster
- There are a default ingres controllers available or you can write your own ingress controller

Ingress Architecture example



Ingress Commands

- Check files from ingress folder
- Commands:
  - vagrant@k8s $ kubectl create –f ingress/ingress.yml
  - vagrant@k8s $ kubectl create –f ingress/nginx-ingress-controller.yml
  - vagrant@k8s $ kubectl create –f ingress/echoservice.yml
  - vagrant@k8s $ kubectl create –f ingress/helloworld-v1.yml
  - vagrant@k8s $ kubectl create –f ingress/ helloworld-v2.yml
  - (*) vagrant@k8s $ curl ip –H 'Host: helloworld-v1.example.com'
  - (*) vagrant@k8s $ curl ip –H 'Host: helloworld-v2.example.com'

Daemon Sets (1/2)

- Daemon Sets ensure that every single node in the Kubernetes cluster runs the same pod resource
  - » This is useful if you want to ensure that a certain pod is running on every single kubernetes node
- When a node is added to the cluster, a new pod will be started automatically
- Same when a node is removed, the pod will not be rescheduled on another node

Daemon Sets (2/2)

- Typical use cases:
  - Logging aggregators
  - Monitoring
  - Load Balancers / Reverse Proxies / API gateways
  - Running a daemon that only needs one instance per physical instance

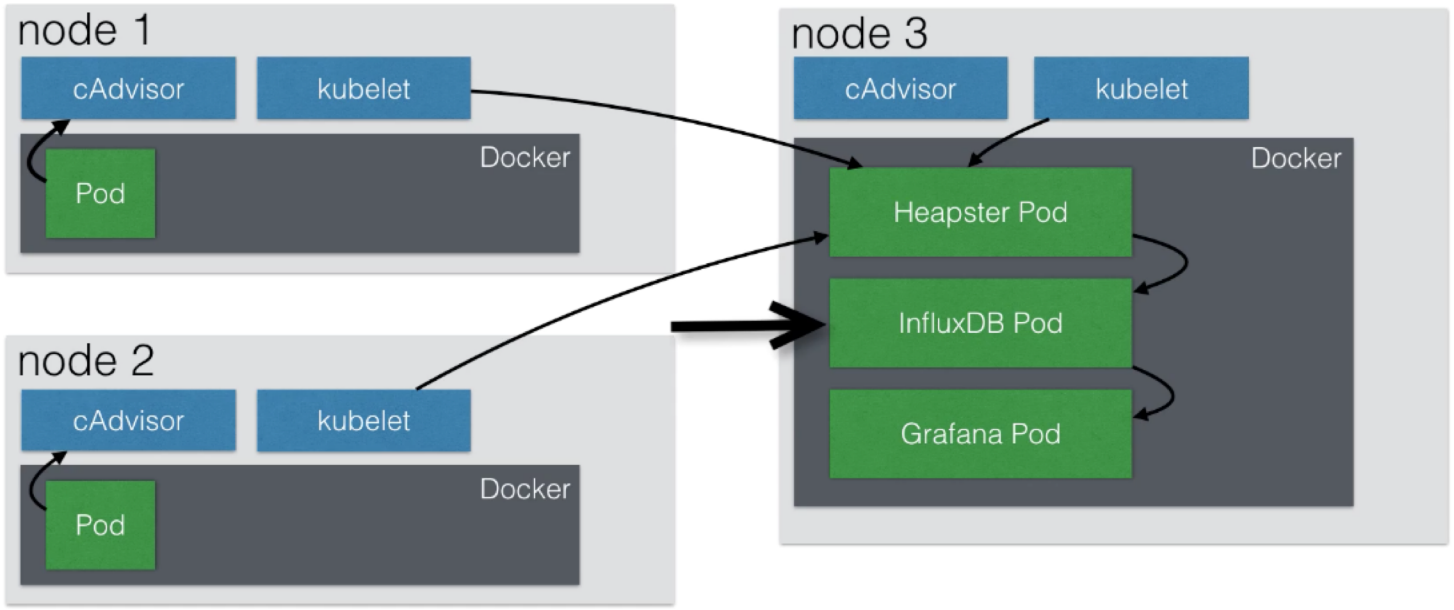
Resource Usage Monitoring

- Heapster enables Container Cluster Monitoring and Performance Analysis
- It is providing a monitoring platform for Kubernetes
- It is a prerequisite if you want to do pod auto-scaling in Kubernetes
- Heapster export clusters metrics via REST Endpoints
- You can use different backends with Heapster
  - » We use InfluxDB but others like Google Cloud Monitoring/Logging and Kafka are also possible

Resource Usage Monitoring

- Visualizations (graphs) can be shown using Grafana
- All these technologies (Heapster, InfluxDB and Grafana) can be started in pods

Resource Usage Monitoring



Metrics Server (Still evolving)

- Commands:
  - vagrant@k8s $ cd examples/metrics-server
  - vagrant@k8s $ kubectl create –f .
  - vagrant@k8s $ kubectl top
  - vagrant@k8s $ kubectl top node
  - vagrant@k8s $ kubectl top pod
  - vagrant@k8s $ kubectl run hello-minikube --image=gcr.io/google_containers/echoserver:1.4 --port=8080
  - vagrant@k8s $ kubectl expose pod nodehelloworld.itnove.com –type=NodePort –name nodehelloworld-service
  - vagrant@k8s $ kubectl top node
  - vagrant@k8s $ kubectl top pod

Kubernetes

Training

Setting up a load balancer for our app (on AWS)

- In real world scenario, you need to be able to access the app from outside the cluster
- On AWS, you can easily add an external Load Balancer
- This AWS Load Balancer will route the traffic to the correct pod in Kubernetes
- There are other solutions for other cloud providers that don't have a Load Balancer
  - » Your own haproxy / nginx load balancer in front of your cluster
  - » Expose ports directly

Web UI

- In general, you can access the Kubernetes Web UI at https://<kubernetes-master-ip>/ui
- If you cannot access it (for instance if it is not enabled on your deploy type), you can install I manually using:
  - vagrant@k8s $ kubectl create –f https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/recommended/kubernetes-dashboard.yaml
- If a password is asked, you can retrieve the password by entering:
  - vagrant@k8s $ kubectl config view

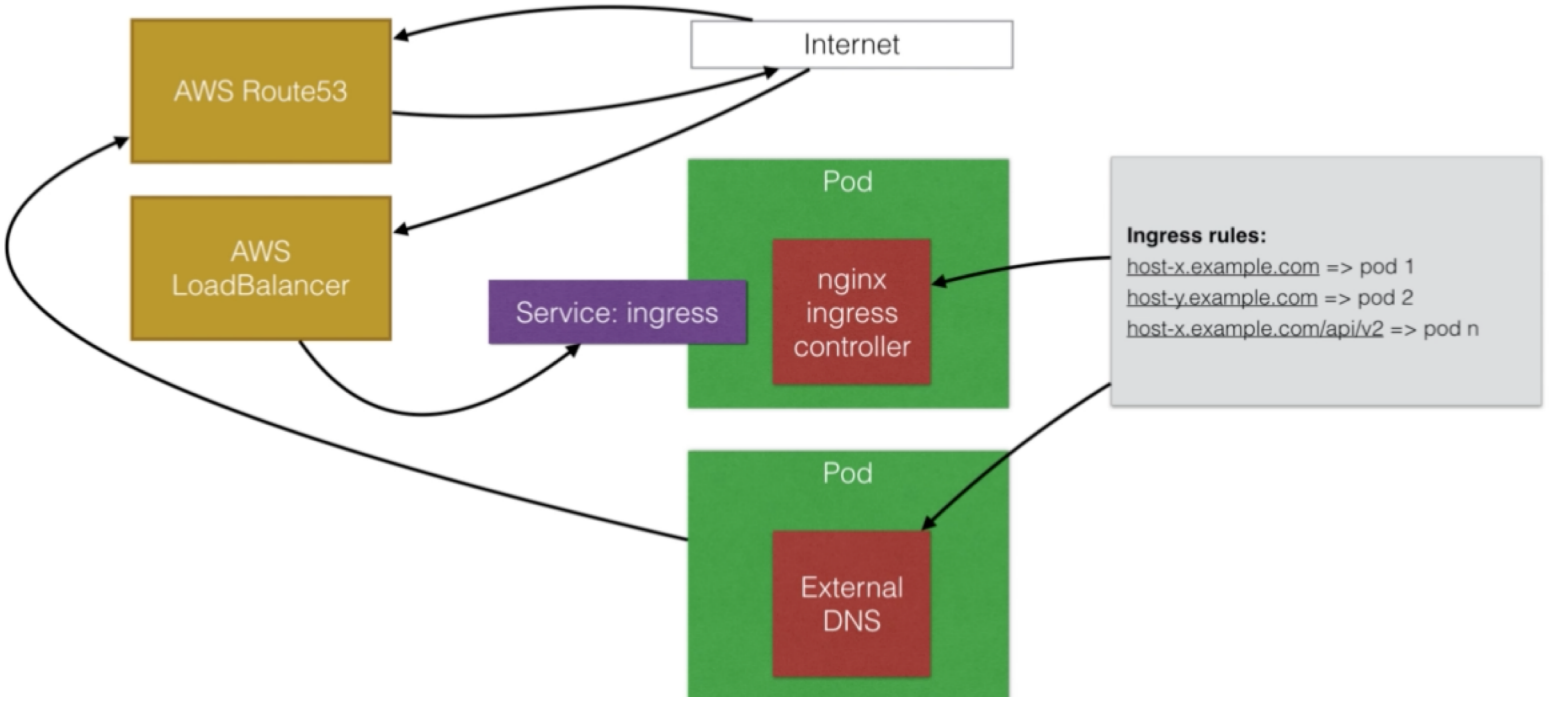
Web UI

- Commands:
  - vagrant@k8s $ kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/recommended/kubernetes-dashboard.yaml
  - vagrant@k8s $ kubectl create –f 011-dashboard.yml
- Get login token
  - vagrant@k8s $ kubectl –n kube-system get secret | grep admin-user
  - vagrant@k8s $ kubectl -n kube-system describe secret admin-user-token-<id-displayed-in-the-former-command>
- Login to the Dashboard
  - Go to http://api.yourdomain.com:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/#!/login
  - Login: admin Password: the password that is listed in ~/.kube/config (open file in editor and look for "password: …"
  - vagrant@k8s $ kubectl config view
  - Choose for login token (from ~/.kube/config) and enter the login token from the previous step

External DNS

- On public cloud providers, you can use ingress controller to reduce the cost of you LoadBalancers
  - You can use 1 LoadBalancers that captures all the external traffic and send it to the ingress controller
  - The ingress controller can be configured to route the different traffic to all your apps based on HTTP rules (host and prefixes)
  - This only works for HTTP(s)-based applications
- This tool will automatically create the necessary DNS records in your external DNS server (like route53
- For every hostname that you use in ingress, it will create a new record to send traffic to your loadbalancer
- The major DNS providers are supported: Google CloudDNS, Route53, AzureDNS, CloudFlare, DigitalOcean, etc,....
- Other setups are also possible without ingress controllers

External DNS

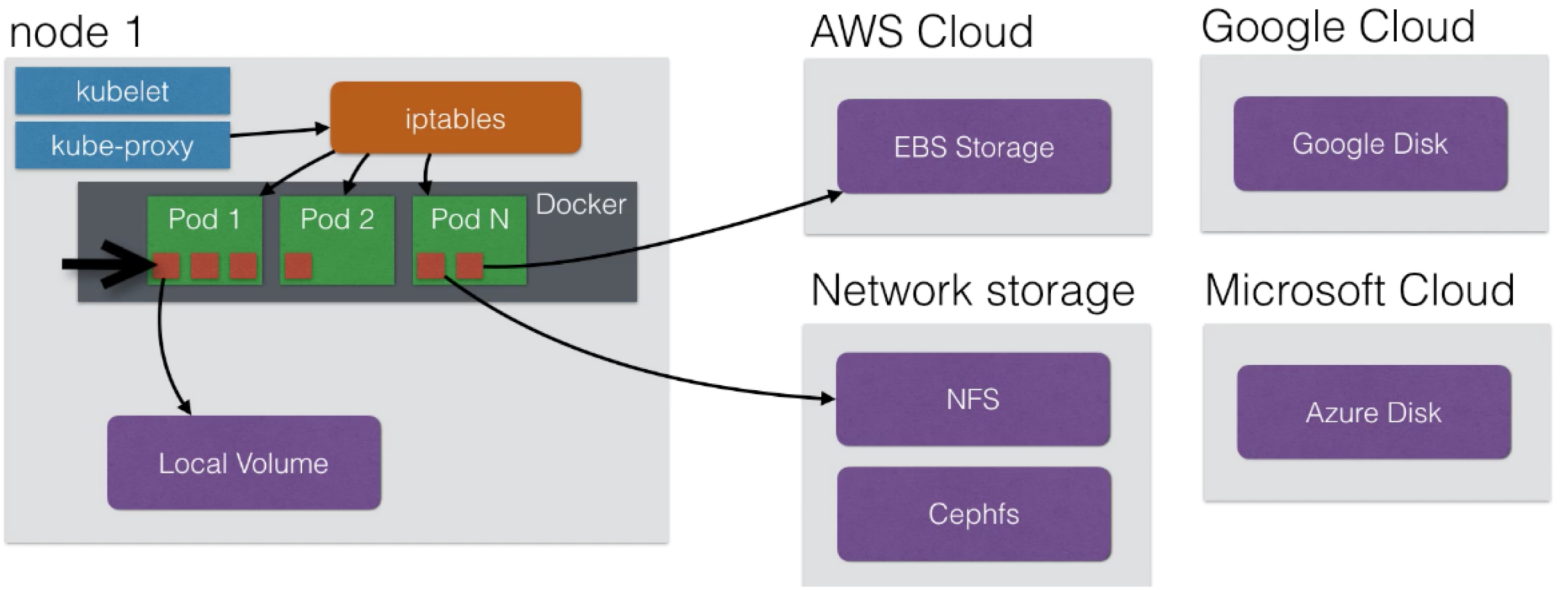


External DNS

Volumes

- Volumes in Kubernetes allow you to store data outside the container
- When a container stops, all data on the container itself is lost
  - That is why up until we've been using stateless apps: apps that do not keep a local state, but store their state in an external service
    - `External service like a database, caching server (e.g. Mysql, AWS S3)`
- Persistent Volumes in Kubernetes allow you attach a volume to a container that will exists even when the container stops

Volumes possible architecture

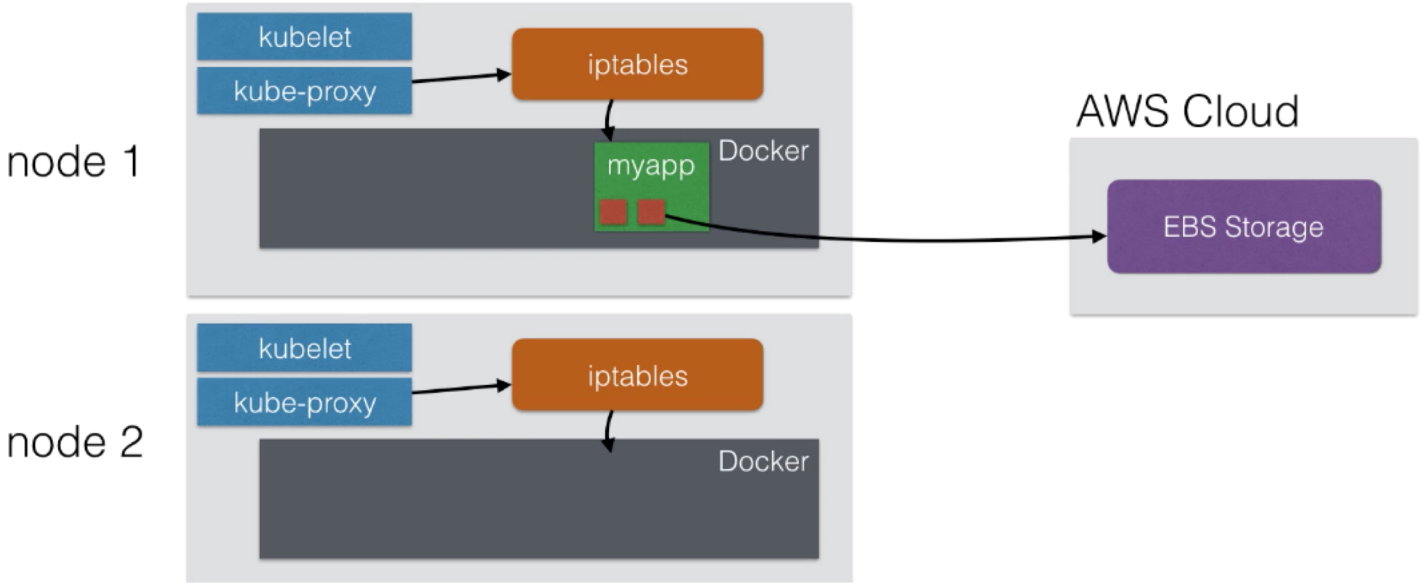


Volumes

- Using volumes, you could deploy applications with state on your cluster
  - » Those applications need to read/write to files on the local filesystem that need to be persistent in time
- For example, you could run a MySQL database using persistent volumes

Volumes

- If your node stops working, the pod can be rescheduled on another node and the volume can be attached to the new node



Pod Presets

- Pod presets can inject information into pods at runtime
  - » Pod Presets are used to inject Kubernetes Resources like Secrets, ConfigMaps, Volumes and Environment variables
- Imagine you have 20 applications you want to deploy, and they all need to get a specfic credential
  - » You can edit 20 specifications and add credentials or…
  - » You can use presets to create 1 Preset Object and inject to credentials to all pods
- When injecting Environment variables and VolumeMounts, the Pod Preset will apply the changes to all containers in the pod

Pod Presets

- You can use more than PodPreset, they'll all be applied to matching pods
- If there is a conflict, the Pod Preset will not be applied to the pod
- Pod Preset can match zero or more Pods
  - » It is possible that no pods are currently matching, but that matching pods will be launched at later time

Pod Preset

- Commands (note it is alpha at the moment, execution may be does not work)
- We have to add alpha plugins to the cluster, but we will try when we start with kops
  - https://kubernetes.io/docs/concepts/workloads/pods/podpreset/
  - vagrant@k8s $ kubectl create -f pod-presets/pod-presets.yaml
  - vagrant@k8s $ kubectl create -f pod-presets/deployments.yaml

StatefulSets

- StatefulSets is introduced to be to run stateful applications
  - » That need a stable pod hostname
    - × `Your podname will have a sticky identity using an index for example`
  - » Statefulsets allow stateful apps stable storage with volumes based on their ordinal number (podname-x)
    - × `Deleting and/or scaling a StatefulSet down will not delete the volumes associated in order to preserve data`
- A StatefulSet will allow your stateful app to use DNS to find other peers
  - » Cassandra clusters, ElasticSearch clusters, use DNS to find other members of the cluster
- Using StatefulSet you can run 3 instances of Cassandra

StatefulSets

- A StatefulSet will also allow your stateful app to order the startup and teardown:
  - Instead of randomly terminating one pod, you will know which one will go
    - `When scaling up it goes from 0 to n-1 (n=replication factor)`
    - `When scaling down it start with n-1 to 0`
  - This is useful if you first need to drain the data from a node before it can be shut down

StatefulSets

- Commands:
  - vagrant@k8s $ kubectl create –f examples/statefulsets/statefulsets.yml
  - vagrant@k8s $ kubectl get pod
  - vagrant@k8s $ kubectl exec –it cassandra-0 – nodetool status