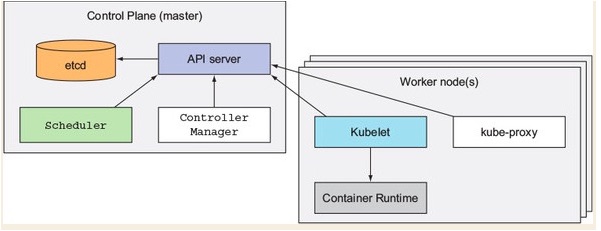
## Kubernetes introduction

* Kubernetes is an open source orchestrator for deploying containerized applications.
* It is used to deploy distributed systems, as well as machine learning, big data and other batch workloads.
* Applications are defined and deployed using a simple declarative syntax.
* Provides Automatic scheduling of application components to servers, automatic configurations, supervision and failure-handling.
* Abstracts infrastructure, exposes the whole cluster as single computational resource.
* Provides Higher utilization of infrastructure.

## Kubernetes introduction

* Makes it easier to perform zero-downtime updates of your software.
* Load balances traffic across a number of replicas of your service.
* Provides tools for discovery of services so that you can build loosely coupled microservices architecture.
* Kubernetes is widely used across public and private clouds.



## Components of kubernetes

* Consistes of
  + Master node
  + Number of worker nodes

## Components of kubernetes

Master Node

* Kubernetes Control plane.
* Apps are submitted to Master node
* It holds the state of cluster

Worker Node

- Runs the application.

## Master node

API Server

* All control plane communications are handled through API Server.

Scheduler

* Assigns worker nodes for application deployment.

Controller Manager

* Keeps track for worker nodes
* Handles node failure

Etcd

- Distributed data store for Persistently storing cluster configuration.

## Worker nodes

Docker - container runtime which runs containers.

Kubelet - manages containers on the node.

Kubeproxy - load balances network traffic between application components.

## To run application in kubernetes

* Package app into one or more container images.
* Push images to an registry

- Post the description of your app to API server.

## Description consists of

* Container image or images that contain components for application.
* Which ones need to be run co-located.
* How many copies (replicas) need to run.
* Which of the components provide services to internal or external clients and should be exposed single IP address and made discoverable to other components.

## Scaling the number of copies

* You can decide to increase or decrease the number of copies and kubernetes will spin up additional ones or stop excess ones.
* Kubernetes can automatically keep adjusting the number based on realtime metric such as CPU, memory consumption or queries per second.
* If kubernetes is running in cloud infrastructure, where adding additional nodes is as easy as requesting them through cloud providers API, kubernetes can even automatically scale the whole cluster size up and down based on needs of deployed application.
* Kubernetes can detect if the new version is bad and stop rollout immediately.

## Creating sharing and running container image

* If you want to run an application in kubernetes, you need to package it as containers.
* Install docker
* Docker daemon is run inside VM
* Docker client on host OS
* Docker client is used to run various docker commands.
* Public docker registry contains ready to use container images.

## Creating Docker file for the image

* List of instructions to build an image.
* It needs to be in same directory as application (app.js)

FROM node:7

ADD app.js /app.js

ENTRYPOINT ["node", "app.js"]

* FROM line defines the base image. node container image, tag 7
* You are adding app.js file from your local directory into the root directory in the image.
* ENTRYPOINT defines the command to be executed, when image is run.

## BUILDING THE CONTAINER IMAGE

$ docker build -t kubia .

* Build the image called kubia, based on contents for current directory.
* Docker will look for Dockerfile in the current directory and build image as per instructions in Dockerfile.
* Content of the whole directory is uploaded to docker daemon and image is built there.
* Docker will pull the base image (node:7) from public image repository.
* Image is composed of multiple layers.
* When building an image, a new layer is created for each command in Dockerfile.

$ docker images - lists all locally stored images.

## Pushing image to image registry

* Publicly available registries, <http://hub.docker.com>, [quay.io](http://quay.io), Google container registry
* you need to retag your image according to Docker Hub's rules.
* Image's repository name should start with Docker Hub ID. Ex: dgunjetti/kubia

$ docker tag kubia dgunjetti/kubia

$ docker images | head

Pushing image to docker hub

* login with your user Id with docker login command
* $ docker push dgunjetti/kubia

## Setting up a kubernetes cluster

* You have your app packaged inside a container image and made available through Docker hub, you can deploy it in a kubernetes cluster.
* <https://cloud.google.com/container-engine/docs/before-you-begin>
* Create a cluster for 3 nodes.
* $ gcloud container clusters create kubia --num-nodes 1 --machine-type f1-micro --region asia-south1
* $ kubectl run kubia --image=dgunjetti/kubia --port=8080 --generator=run/v1
* --generator will create a replication controller.

## Introducing pods

* Kubernetes does not deal with containers directly. Instead it uses a concept of multiple co-located containers. This group of container is called Pod.
* A Pod is the atomic unit of scheduling in a Kubernetes cluster.
* All containers in a Pod land on same machine in the cluster.
* Each pod is like a logical machine with its own IP, hostname and process running a single application.
* All containers in pod share the same linux namespace.
* $ kubectl get pods

## Understanding how application is run on kubernetes

* What happens when we run application on kubernetes
* $ kubectl run kubia --image=luksa/kubia --port=8080 --generator=run/v1
* Kubectl sends a http request to API server to create a new ReplicationController object.
* The ReplicationController created a new pod, which is scheduled to one of the worker node by Scheduler.
* Kubelet on saw that pod was scheduled to it and instructed Docker to pull the specified image from the registry.
* After downloading image, Docker creates and runs the container.

## Accessing your web application

* Each pod gets its own IP address, but this address is internal to cluster and is not accessible from outside.
* To make the pod accessible from outside, you will expose it through a Service object.
* You will create a special service of type LoadBalancer.
* By creating LoadBalancer type service, an external load balancer will be created and you can connect to the pod through load balancer public IP.

Creating a Service object

* $ kubectl expose rc kubia --type=LoadBalancer --name=kubia-http

- $ kubectl get svc - get to know the external IP address.

- $ curl 104.155.74.57:8080

## Splitting multi-tier application into multiple pods

* Split frontend to one pod and backend to other pod.
* Frontend component have completely different scaling requirement than backend, so we tend to scale them individually.