Stopping the app

When you hit ^C, Compose tries to gracefully terminate all of the containers. After ten seconds (or if you press ^C again) it will forcibly kill them.

The docker-compose.yml file

Here is the file used in the demo:

```
version: "2"


services:

  www:

    build: www

    ports:

      - 8000:5000

    user: nobody

    environment:

      DEBUG: 1

    command: python counter.py

    volumes:

      - ./www:/src


  redis:

    image: redis
```

Compose file versions

Version 1 directly has the various containers (www, redis...) at the top level of the file.

Version 2 has multiple sections:

- version is mandatory and should be "2".
- services are mandatory and corresponds to the content of the version 1 format.
- networks are optional and can define multiple networks on which containers can be placed.
- volumes are optional and can define volumes to be used (and potentially shared) by the containers.

Containers in docker-compose.yml

Each service in the YAML file must contain either build, or image.

- build indicates a path containing a Dockerfile.
- image indicates an image name (local, or on a registry).

The other parameters are optional.
They encode the parameters that you would typically add to docker run. Sometimes they have several minor improvements.

Container parameters

- command indicates what to run (like CMD in a Dockerfile).
- ports translate to one (or multiple) -p options to map ports. You can specify local ports (i.e. x:y to expose public port x).
- volumes translates to one (or multiple) -v options.

You can use relative paths here.

For the full list, check http://docs.docker.com/compose/yml/.

Compose commands

We already saw docker-compose up, but another one is docker-compose build. It will execute docker build for all containers mentioning a build path. It is common to execute the build and run steps in sequence:

```
$ docker-compose build && docker-compose up
```

Another common option is to start containers in the background:

```
$ docker-compose up -d
```

Check container status

It can be tedious to check the status of your containers with docker ps, especially when running multiple apps at the same time.
Compose makes it easier; with docker-compose ps you will see only the status of the containers of the current stack:

```
$ docker-compose ps
         Name                       Command               State           Ports
-------------------------------------------------------------------------------------
trainingwheels_redis_1   docker-entrypoint.sh redis ...   Up      6379/tcp
trainingwheels_www_1     python counter.py                Up      0.0.0.0:8000->5000/tcp
```

Cleaning up

If you have started your application in the background with Compose and want to stop it easily, you can use the kill command:

```
$ docker-compose kill
```

Likewise, docker-compose rm will let you remove containers (after confirmation):

```
$ docker-compose rm
Going to remove trainingwheels_redis_1, trainingwheels_www_1
Are you sure? [yN] y
Removing trainingwheels_redis_1...
Removing trainingwheels_www_1...
```

Alternatively, docker-compose down will stop and remove containers.

```
$ docker-compose down
Stopping trainingwheels_www_1 ... done
Stopping trainingwheels_redis_1


... done Removing trainingwheels_www_1 ... done Removing
trainingwheels_redis_1
... done
```

Special handling of volumes

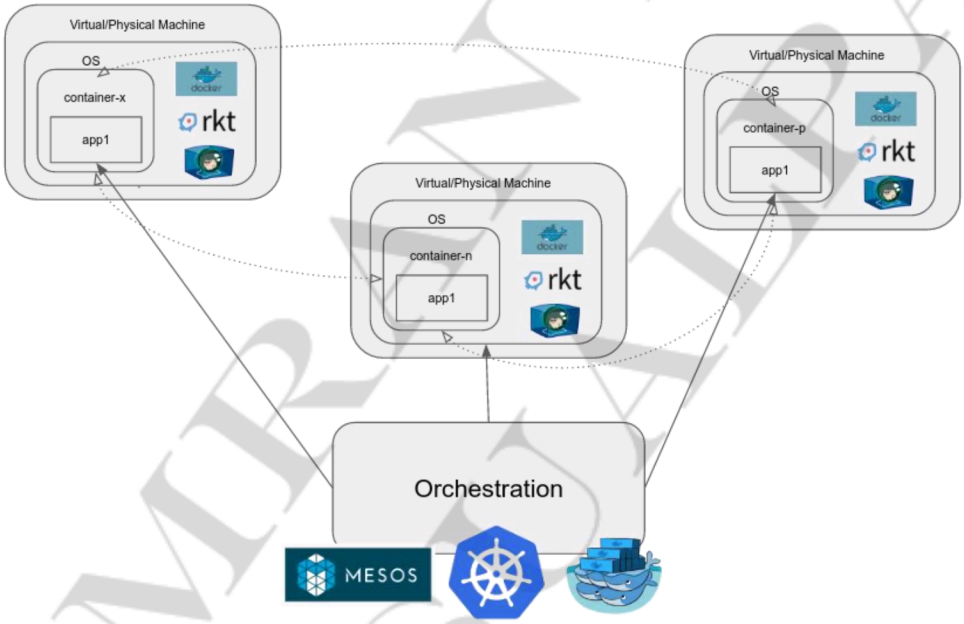
Compose is smart. If your container uses volumes, when you restart your application, Compose will create a new container, but carefully re-use the volumes it was using previously. This makes it easy to upgrade a stateful service, by pulling its new image and just restarting your stack with Compose.

XV. Kubernetes

1. Kubernetes Introduction

We have seen so far in Docker chapter that we can create images and run containers on the docker host. That is so cool its lightweight, fast and shippable but what about production. Can we run our applications on those lightweight containers or to ask you more specifically running all those containers on one host? Answer is it's not safe to put all your eggs in one basket.

So, we need cluster of docker hosts, which can be managed by some external application. Something that can schedule containers for us on the best suitable host in the cluster. It should also detect a failed container and fix the problem for us. So, we are looking for a Docker orchestration tool.



We have few docker orchestration tools and cloud services in the market as listed below.

Amazon ECS -- The Amazon EC2 Container Service (ECS) supports Docker containers and lets you run applications on a managed cluster of Amazon EC2 instances.

Azure Container Service (ACS) -- ACS lets you create a cluster of virtual machines that act as container hosts along with master machines that are used to manage your application containers.

Cloud Foundry's Diego -- Diego is a container management system that combines a scheduler, runner, and health manager. It is a rewrite of the Cloud Foundry runtime.

CoreOS Fleet -- Fleet is a container management tool that lets you deploy Docker containers on hosts in a cluster as well as distribute services across a cluster.

Docker Swarm -- Docker Swarm provides native clustering functionality for Docker containers, which lets you turn a group of Docker engines into a single, virtual Docker engine.

Google Container Engine -- Google Container Engine, which is built on Kubernetes, lets you run Docker containers on the Google Cloud platform. It schedules containers into the cluster and manages them based on user-defined requirements.

Kubernetes -- Kubernetes is an orchestration system for Docker containers. It handles scheduling and manages workloads based on user-defined parameters.

Mesosphere Marathon -- Marathon is a container orchestration framework for Apache Mesos that is designed to launch long-running applications. It offers key features for running applications in a clustered environment.

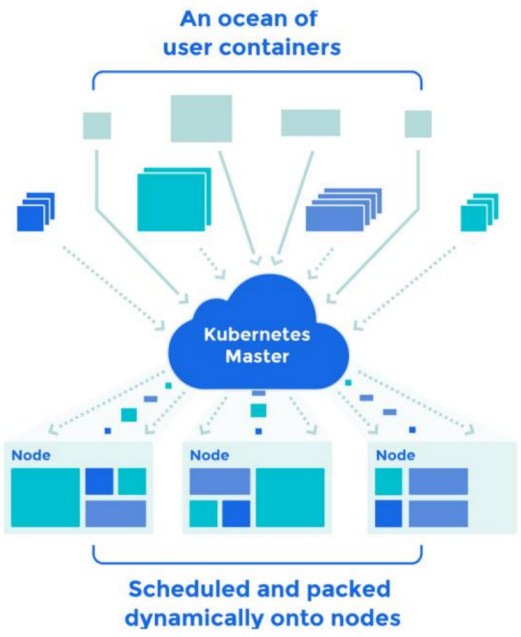
2. Why Kubernetes?

Among all of them we are focussing on Kubernetes in this tutorial because of below mentioned reasons

- Kubernetes is google's own project which they used to managed containers from past 10 years. So, huge applications experience and mature model is something we look for productions.
- Kubernetes is ranked among the best Container Orchestration tool in the market as per survey.
- It supports other container platform apart from docker like RKT.
- Kubernetes is an open source project, its distinct from other "vendor-driven" project like Swarm, Mesos, CloudFoundry. Docker swarn is also open source but its tightly integrated with other Docker tools.
- It can support a large number of applications. This has been a talking point for the Kubernetes community since the spring, when it announced the tool could run more than 1,000 nodes.

That's does not mean that other orchestartion tools are not as good as Kubernetes but we just wanted to try first the best among them and then we can try others if there are any shortcomings with Kubernetes.

3. What is Kubernetes?



As Per Kubernetes Documentation.

Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts, providing container-centric infrastructure.

With Kubernetes, you can quickly and efficiently respond to customer demand:

- Deploy your applications quickly and predictably.
- Scale your applications on the fly.
- Roll out new features seamlessly.
- Limit hardware usage to required resources only.

Kubenetes goal is to foster an ecosystem of components and tools that relieve the burden of running applications in public and private clouds.

Kubernetes is:

•**Portable**: public, private, hybrid, multi-cloud

•**Extensible**: modular, pluggable, hookable, composable

•**Self-healing**: auto-placement, auto-restart, auto-replication, auto-scaling

Google started the Kubernetes project 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.

4. What Kubernetes can do?

At a minimum, Kubernetes can schedule and run application containers on clusters of physical or virtual machines. Kubernetes provides the infrastructure to build a truly **container-centric** development environment.

Kubernetes satisfies a number of common needs of applications running in production, such as:

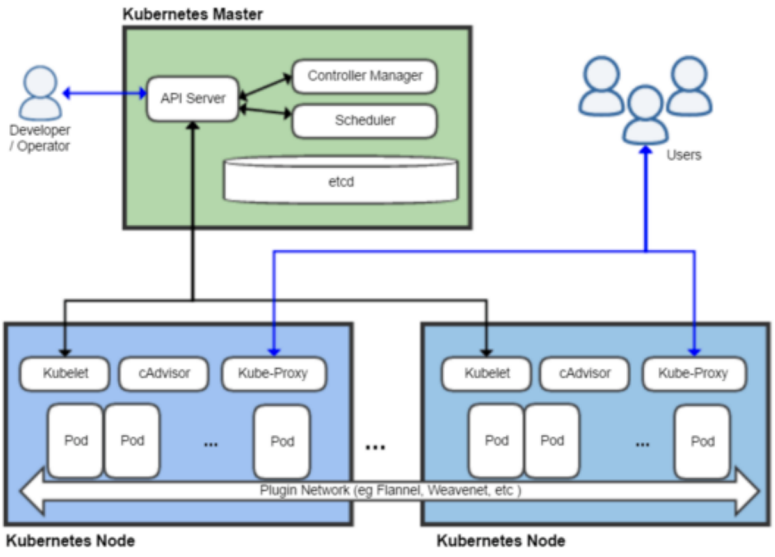
- Co-locating helper processes, facilitating composite applications and preserving the one-application-per-container model
- Mounting storage systems
- Distributing secrets
- Checking application health
- Replicating application instances
- Using Horizontal Pod Autoscaling
- Naming and discovering
- Balancing loads
- Rolling updates
- Monitoring resources
- Accessing and ingesting logs
- Debugging applications
- Providing authentication and authorization

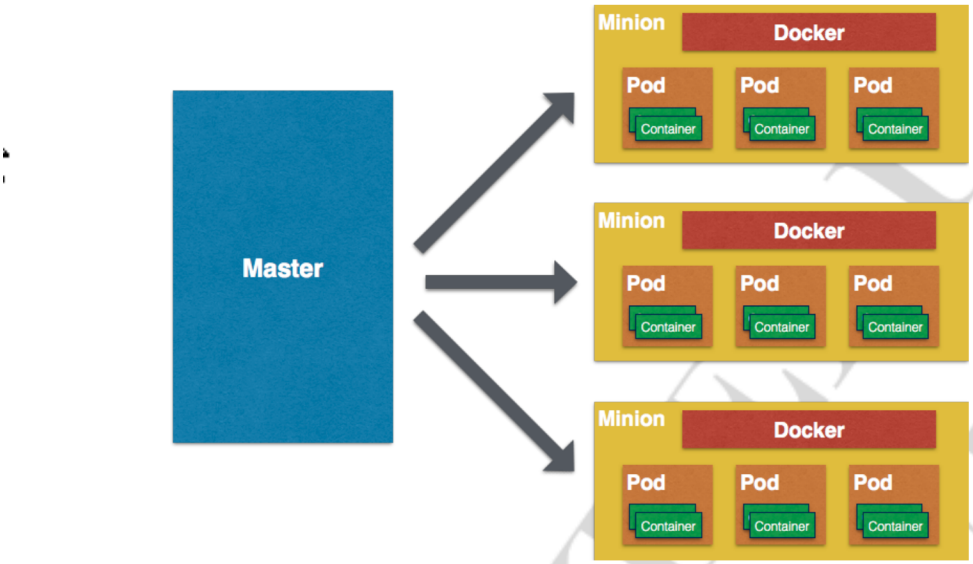
5. Kubernetes Architecture

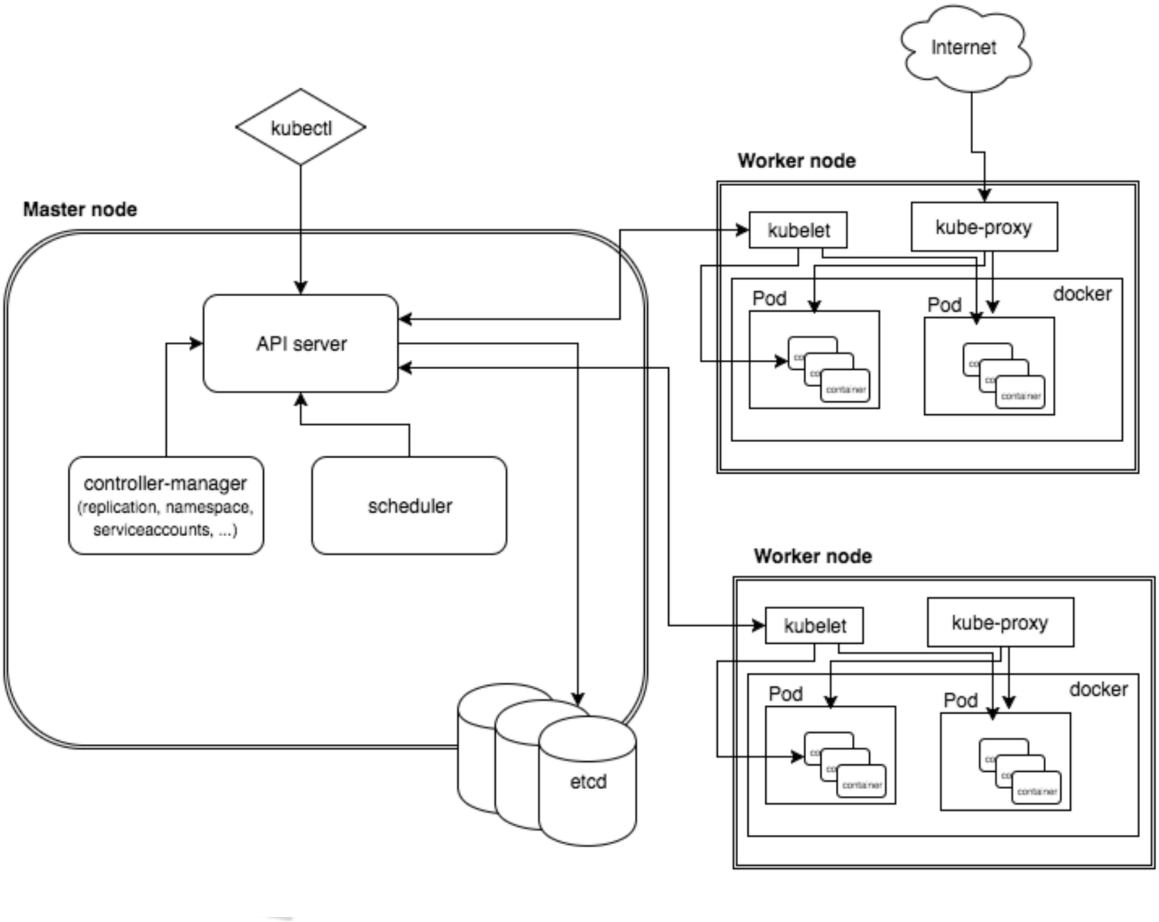
Kubernetes is a stack of services that work together to manage all the hosts, Kubernetes cluster is what we call them together AKA K8s. All those services or components are shown in the below three diagrams.

POD: A *pod* is a group of one or more containers (such as Docker containers), the shared storage for those containers, and options about how to run the containers. Pods are always co-located and co-scheduled, and run in a shared context. A pod models an application-specific "logical host" - it contains one or more application containers which are relatively tightly coupled — in a pre-container world, they would have executed on the same physical or virtual machine.

SERVICE: Kubernetes `Pods` are mortal. They are born and when they die, they are not resurrected. `ReplicationControllers` in particular create and destroy `Pods` dynamically (e.g. when scaling up or down or when doing rolling updates). While each `Pod` gets its own IP address, even those IP addresses cannot be relied upon to be stable over time. This leads to a problem. Service will be on top of the POD and will have a stable IP, it's like a load balancer, so not matter what nodes you have under load balancer and what their IP is it can be accessed by the Load balancer. Its similar for the Service in Kubernetes







Master Node Architecture

Etcd: It is an open source key-value store developed by CoreOs team. Kubernetes uses 'Etcd' to store the configuration data accessed by all nodes (minions and master) in the cluster.

Example of data stored by Kubernetes in etcd are jobs being scheduled, created and deployed pod/service details and state, namespaces and replication informations, etc.

Kube-ApiServer: The Kubernetes api-server generally validates the configuration data store in 'Etcd' and the details of the deployed container that are in agreement. It also provides a RESTful interface to make communication easy.

Kube-Schedule Server: The deployment of configured pods and services onto the nodes is done by the `scheduler` component.It is responsible for assigning task to minions in the cluster.Scheduler has the information regarding resources available on the members of the cluster, as well as the ones required for the configured service to run and hence is able to decide where to deploy a specific service.

Kube-Controller-Manager: It is generally responsible for handling the cluster level function such as replication controller. Whenever the desired state of the cluster changes it is written to Etcd and then the controller manager tries to bring up the cluster in the desired state. A controller uses apiserver to watch the shared state of the cluster and makes corrective changes to the current state to being it to the desired one. One example is R*eplication Controller* which takes care of the PODS in the system, if any POD fails it replaces that with a new POD.

Minion Node Architecture

Docker: One of the basic requirement of nodes is Docker. Docker is responsible for pulling down and running container from Docker images.

Kube-Proxy: Every node in the cluster runs a simple network proxy. Using proxy node in cluster routes request to the correct container in a node.

Kubelet: It is an agent process that runs on each node. It is responsible for managing pods and their containers. It deal with pods specifications which are defined in YAML or JSON format. Kubelet takes the pod specifications and checks whether the pods are running healthy or not.

Flannel: It is an overlay network that works on assigning a range of subnet address. It is used to assign IPs to each pods running in the cluster and to make the pod-to-pod and pod-to-services communications.

6. Kubernetes Setup

Kubernetes can be setup on a single linux machine or we can have Master and Nodes on seprate machine. As you would have guessed single node cluster is good for learning and testing but production grade Kubernetes Cluster is required to manage large scale applications and containers.
We will see both ways of setting up Kuberneter cluster. Setting up all the Kubernetes components manually is a very tedious task but there are few tools in the market that can automate kubernetes cluster deployment for us.

- **Minikube**
  Minikube sets up a single node cluster on a vm running on virtualbox. We can create and also manage the Kubernetes cluster with minikube.
- **KOPS**
  Kops sets up a production grade multinode Kubernetes cluster on AWS cloud, currently it supports only AWS provider.

7. Kubernetes Detailed Setup & Exercises

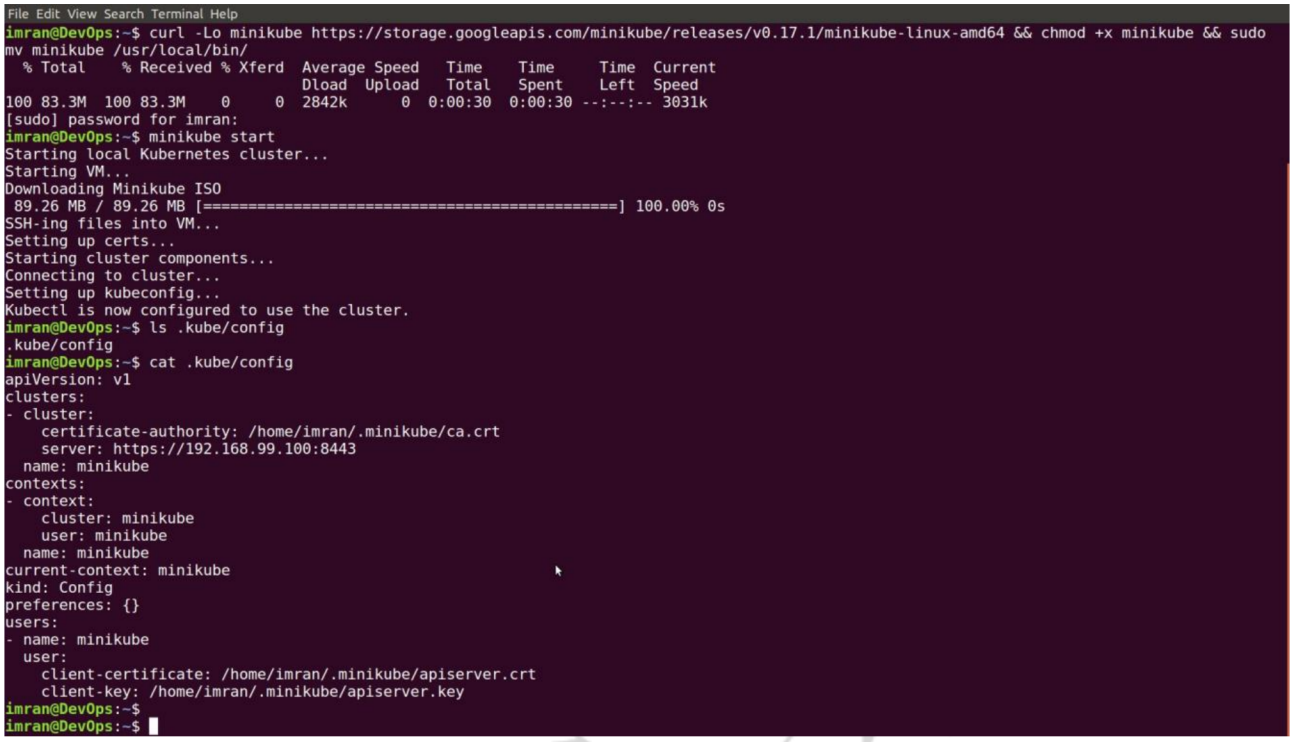
8. Minikube setup locally.

What is minikube?
Minukube runs a single node kubernetes cluster inside a VM on your laptop.
It's for learning and testing kubernetes, should not be used in production.

Setup Minikube.
Minikube can be downloaded from its github repo.

https://github.com/kubernetes/minikube/releases

We are going to setup minikube on a Linux machine so check the latest release url and download the package.
URL mentioned in screenshot was latest at the time of document preparation.
Virtualbox is the dependency for it, as by default it will create a vm on virtualbox and setup minikube on it.



Kubectl

`kubectl` is a command line interface for running commands against Kubernetes clusters.

Installing and Setting Up kubectl

Check the download page for the latest release url.

https://kubernetes.io/docs/tasks/kubectl/install/

- Kubectl will read the configuration from ~/.kube/config and will know the IP, Port, Auth other details to connect to kubenetes cluster.

```
imran@DevOps:~$ cat .kube/config
apiVersion: v1
clusters:
- cluster:
    certificate-authority: /home/imran/.minikube/ca.crt
    server: https://192.168.99.100:8443
  name: minikube
contexts:
- context:
    cluster: minikube
    user: minikube
  name: minikube
current-context: minikube
kind: Config
preferences: {}
users:
- name: minikube
  user:
    client-certificate: /home/imran/.minikube/apiserver.crt
    client-key: /home/imran/.minikube/apiserver.key
```

- Run kubectl command to verify if it's working.

```
imran@DevOps:~$ kubectl
kubectl controls the Kubernetes cluster manager.

Find more information at https://github.com/kubernetes/kubernetes.

Basic Commands (Beginner):
  create         Create a resource by filename or stdin
  expose         Take a replication controller, service, deployment or pod and expose it as a new Kubernetes Service
  run            Run a particular image on the cluster
  set            Set specific features on objects

Basic Commands (Intermediate):
  get            Display one or many resources
  explain        Documentation of resources
  edit           Edit a resource on the server
  delete         Delete resources by filenames, stdin, resources and names, or by resources and label selector

Deploy Commands:
  rollout        Manage a deployment rollout
  rolling-update Perform a rolling update of the given ReplicationController
  scale          Set a new size for a Deployment, ReplicaSet, Replication Controller, or Job
```

- Try the commands in below screenshot to test if the cluster is working.

```
imran@DevOps:~$ kubectl run hello-minikube --image=gcr.io/google_containers/echoserver:1.4 --port=8080
deployment "hello-minikube" created
imran@DevOps:~$ kubectl expose deployment hello-minikube --type=NodePort
service "hello-minikube" exposed
imran@DevOps:~$ minikube service hello-minikube --url
http://192.168.99.100:31130
imran@DevOps:~$
```

- Access the exposed service with IP and port you get.

That shows our kubernetes cluster is working, we are able to pull images and run a container and also access its service.

- If you want to stop your cluster you can run minikube stop and it will bring down minikube cluster.

9. Kops

Kubernetes Cluster on AWS cloud

What is Kops?

Kops lets you deploy production-grade, highly available, Kubernetes clusters from the command line. Deployment is currently supported on Amazon Web Services (AWS), with more platforms planned.

- Kubernetes Operations (kops) Production Grade K8s Installation, Upgrades, and Management
- The easiest way to get a production grade Kubernetes cluster up and running.
- Works on Mac OS/Linux
-

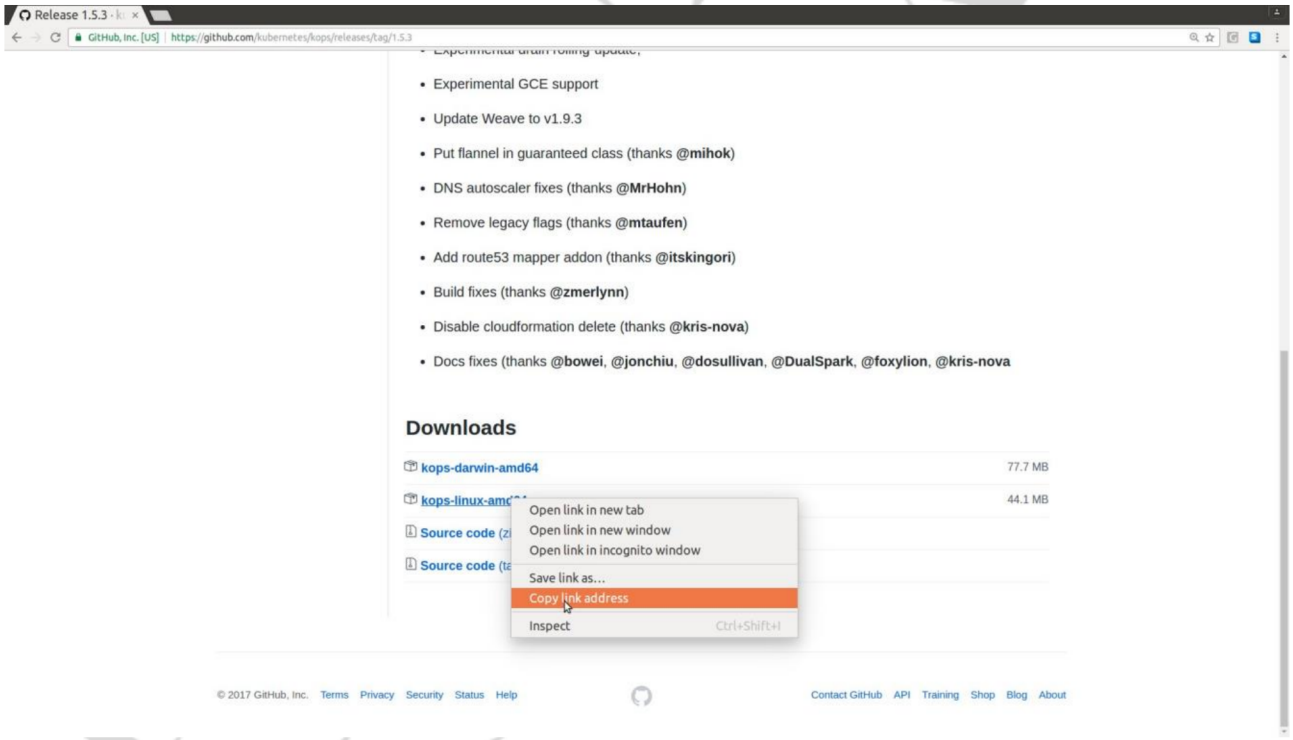
We will setup a Linux vm with vagrant+virtualbox and setup Kops in the vm.

Kops on VM

- Setup vagrant and virtualbox.
- Create a kops directory and bring up xenial64 vm.
- Bring up the vm

```
imran@DevOps:.../kuberbetes$ mkdir kops && cd kops
imran@DevOps:.../kops$ vagrant init ubuntu/xenial64
A `Vagrantfile` has been placed in this directory. You are now
ready to `vagrant up` your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
`vagrantup.com` for more information on using Vagrant.
imran@DevOps:.../kops$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'ubuntu/xenial64'...
==> default: Matching MAC address for NAT networking...
==> default: Checking if box 'ubuntu/xenial64' is up to date...
==> default: A newer version of the box 'ubuntu/xenial64' is available! You currently
==> default: have version '20170119.1.0'. The latest is version '20170328.0.0'. Run
==> default: `vagrant box update` to update.
==> default: Setting the name of the VM: kops_default_1490842068932_47692
==> default: Fixed port collision for 22 => 2222. Now on port 2200.
==> default: Clearing any previously set network interfaces...
```

- Copy the link of the latest Linux-amd64 version of Kops from github.



- Login to the vm and download Kops.

```
==> default: Mounting shared folders...
    default: /vagrant => /tmp/kuberbetes/kops
imran@DevOps:.../kops$ vagrant ssh
Welcome to Ubuntu 16.04.1 LTS (GNU/Linux 4.4.0-59-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

0 packages can be updated.
0 updates are security updates.


ubuntu@ubuntu-xenial:~$ wget https://github.com/kubernetes/kops/releases/download/1.5.3/kops-linux-amd64
--2017-03-30 02:59:06--  https://github.com/kubernetes/kops/releases/download/1.5.3/kops-linux-amd64
Resolving github.com (github.com)... 192.30.253.112
Connecting to github.com (github.com)|192.30.253.112|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://github-cloud.s3.amazonaws.com/releases/62091339/2cfa1aca-0533-11e7-9ae6-0b665d134100?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz
```

- Make is executable & move to /usr/local/bin.

```
ubuntu@ubuntu-xenial:~$ chmod u+x kops-linux-amd64 && sudo mv kops-linux-amd64 /usr/local/bin/
ubuntu@ubuntu-xenial:~$
```

- Install AWS cli

```
File  Edit  View  Search  Terminal  Help
ubuntu@DevImranOps:~$ sudo apt-get -qq update
ubuntu@DevImranOps:~$ sudo apt-get -qq -y install python-pip
ubuntu@DevImranOps:~$ sudo pip install awscli
The directory '/home/ubuntu/.cache/pip/http' or its parent directory is not owned by the current user and the cache has been disabled. Please
 check the permissions and owner of that directory. If executing pip with sudo, you may want sudo's -H flag.
The directory '/home/ubuntu/.cache/pip' or its parent directory is not owned by the current user and caching wheels has been disabled. check
the permissions and owner of that directory. If executing pip with sudo, you may want sudo's -H flag.
Collecting awscli
  Downloading awscli-1.11.68-py2.py3-none-any.whl (1.2MB)
    100% |████████████████████████████████| 1.2MB 963kB/s
Collecting s3transfer<0.2.0,>=0.1.9 (from awscli)
  Downloading s3transfer-0.1.10-py2.py3-none-any.whl (54kB)
    100% |████████████████████████████████| 61kB 6.5MB/s
Collecting botocore==1.5.31 (from awscli)
  Downloading botocore-1.5.31-py2.py3-none-any.whl (3.4MB)
    100% |████████████████████████████████| 3.4MB 391kB/s
Collecting rsa<=3.5.0,>=3.1.2 (from awscli)
```

AWS setup for Kops

- Create an S3 bucket in the region which you will use to store the state of KOPS.

Domain Name Setup.

We need a domain name and few sub domains, we can setup both with AWS route53.

I have domain name(devimranops.club) with namecheap & subdomains in Route53.

You can go with route53 for domain name and sub domain if you wish to.

- Register a domain with any domain name provider like Godaddy or Namecheap or Route53.

  I have domain devimranops.club which I will use to demonstrate all the exercises with Kubernetes.

Sub Domain Setup with AWS Route53.

- Go to AWS route53 dashboard => DNS management => Create Hosted Zone.
- Domain name => kubernetes.<Your Domain Name>
- Make a note of four ns values you see after creating subdomain.

NS Record setup with Domain provider.

- Go to your domain provider site.
- Add NS record for kubernetes sub domain.

For namecheap follow the below mentioned procedure.

Manage => Advanced DNS => Add new record.

You must add the four ns record values as we had four ns server name in Route53.

Dont use the ns server name mentioned in screenshot, use the one which you created with Route53.

AWS Cli setup.

- Create an IAM user with full access and download its credentials file.
- Execute aws configure command in our VM and enter AWS access key and Secret Key from credentials.csv file.

```
ubuntu@DevImranOps:~$ sudo -i
root@DevImranOps:~# aws configure
AWS Access Key ID [None]: AKIAJXEPCUDXBJKS3BOQ
AWS Secret Access Key [None]: 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
Default region name [None]:
Default output format [None]:
root@DevImranOps:~#
```

Setup Kubectl.

- Installing and Setting Up kubectl

Check the download page for the latest release url.

https://kubernetes.io/docs/tasks/kubectl/install/

```
root@DevImranOps:~# curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-re
lease/release/stable.txt)/bin/linux/amd64/kubectl
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 67.4M  100 67.4M    0     0  2885k      0  0:00:23  0:00:23 --:--:-- 2901k
root@DevImranOps:~# chmod u+x kubectl && mv kubectl /usr/local/bin/
root@DevImranOps:~#
```

- Setup SSH keys for cluster login.

```
root@DevImranOps:~# ssh-keygen -f .ssh/id_rsa
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in .ssh/id_rsa.
Your public key has been saved in .ssh/id_rsa.pub.
The key fingerprint is:
SHA256:oxGZrDQSA/J/WqzSNM0rh6M5XZY+MmPUSW9YcWDbvdQ root@DevImranOps
The key's randomart image is:
+---[RSA 2048]----+
|o.o      o.      |
|.. o . +.o.. .   |
|  o o = .o. o E  |
|   + *...  . .   |
|    =oB*S   .    |
|   o.BB+o.       |
|  .oB++.         |
|  .+*+o          |
|  oo + .         |
+----[SHA256]-----+
root@DevImranOps:~#
```

- Renaming kops binary & test it.

```
root@DevImranOps:~# mv /usr/local/bin/kops-linux-amd64 /usr/local/bin/kops
root@DevImranOps:~# kops
kops is kubernetes ops.
It allows you to create, destroy, upgrade and maintain clusters.

Usage:
  kops [command]

Available Commands:
  completion     Output shell completion code for the given shell (bash)
  create         Create a resource by filename or stdin
```

- Setup Google DNS name server (8.8.8.8, 8.8.4.4)in our VM so VM can resolve kubernetes.<Your Domain name>

```
root@DevImranOps:~# vi /etc/network/interfaces
```

```
#VAGRANT-BEGIN
# The contents below are automatically generated by Vagrant. Do not modify.
auto enp0s8
iface enp0s8 inet dhcp
    post-up route del default dev $IFACE || true
    dns-nameservers 8.8.8.8 8.8.4.4
#VAGRANT-END
```

Kubernetes.<Your domain name> should be resolved to an ns record value after this.

As displayed in the below screenshot its resolving for me.

```
root@DevImranOps:~# nslookup -type=ns kubernetes.devimranops.club
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
kubernetes.devimranops.club     nameserver = ns-1120.awsdns-12.org.
kubernetes.devimranops.club     nameserver = ns-1896.awsdns-45.co.uk.
kubernetes.devimranops.club     nameserver = ns-214.awsdns-26.com.
kubernetes.devimranops.club     nameserver = ns-639.awsdns-15.net.

Authoritative answers can be found from:

root@DevImranOps:~#
```

Setup Kubernetes Cluster

Now it's showtime, we will create Kubernetes Cluster on AWS with Kops command.

As of now we should have below mentioned setup ready from AWS.

- ✓ S3 bucket, my bucket name is kops-state-86.
- ✓ Complete domain name for our cluster, mine is kubernetes.devimranops.club.
- ✓ AWS cli setup with an Admin IAM user.
- ✓ AWS region, I am using us-west-1
- ✓ AWS zone, I am using us-west-1a

◆ We need to supply all the values to the kops command as shown below.

```
root@DevImranOps:~# kops create cluster --name=kubernetes.devimranops.club --state=s3://kops-state-86 --zone
s=us-west-1a --node-count=2 --node-size=t2.micro --master-size=t2.micro --dns-zone=kubernetes.devimranops.cl
ub
```

You should get output as below, which shows cluster configuration is created by kops.

Cluster has not yet created, follow next command to create it.