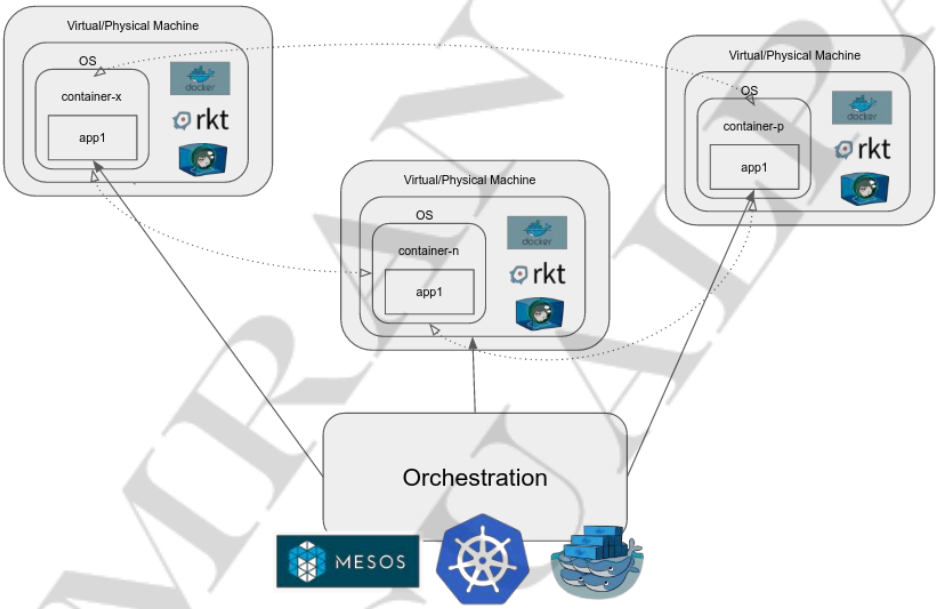
XV. Kubernetes

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.

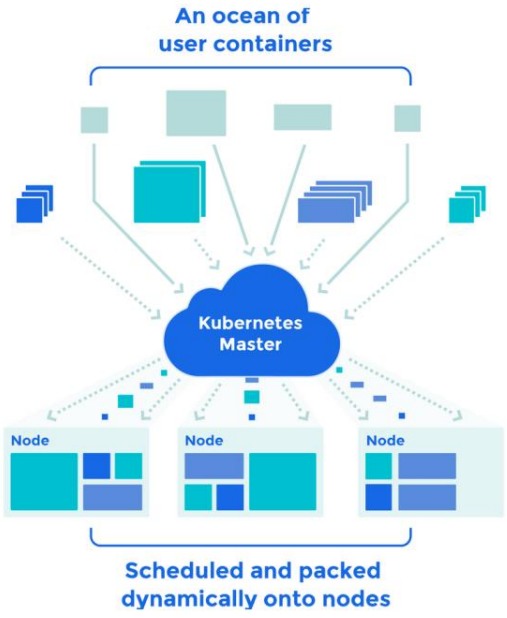
2. Why Kubernetes?

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:

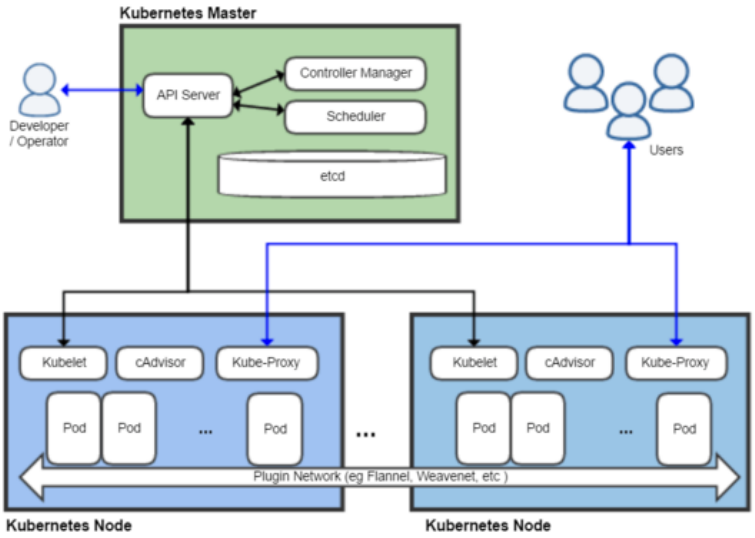
- Co-locating helper processes, facilitating composite applications and preserving the one-application-per-container model
- Mounting storage systems
- Distributing secrets
- Checking application health
- Replicating application instances
- Using Horizontal Pod Autoscaling
- Naming and discovering
-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

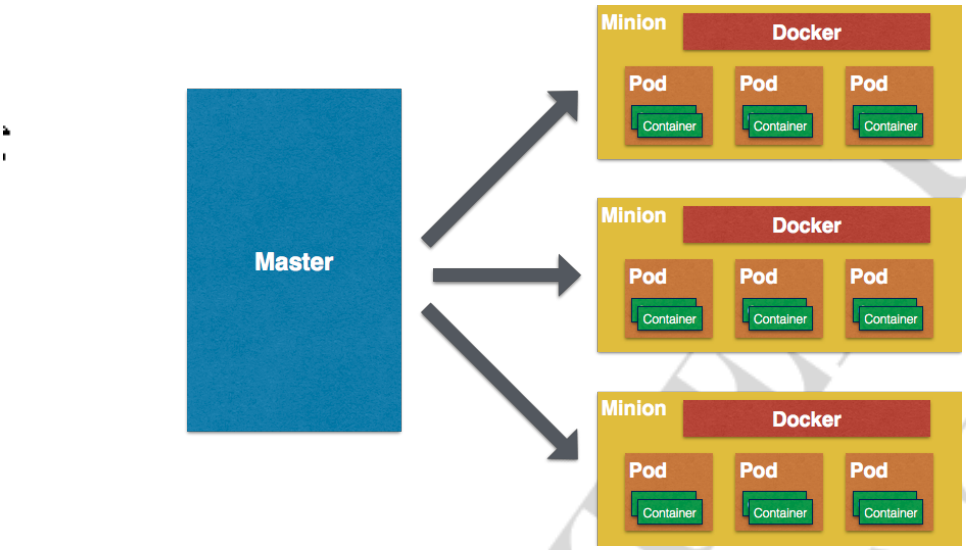
5. Kubernetes Architecture

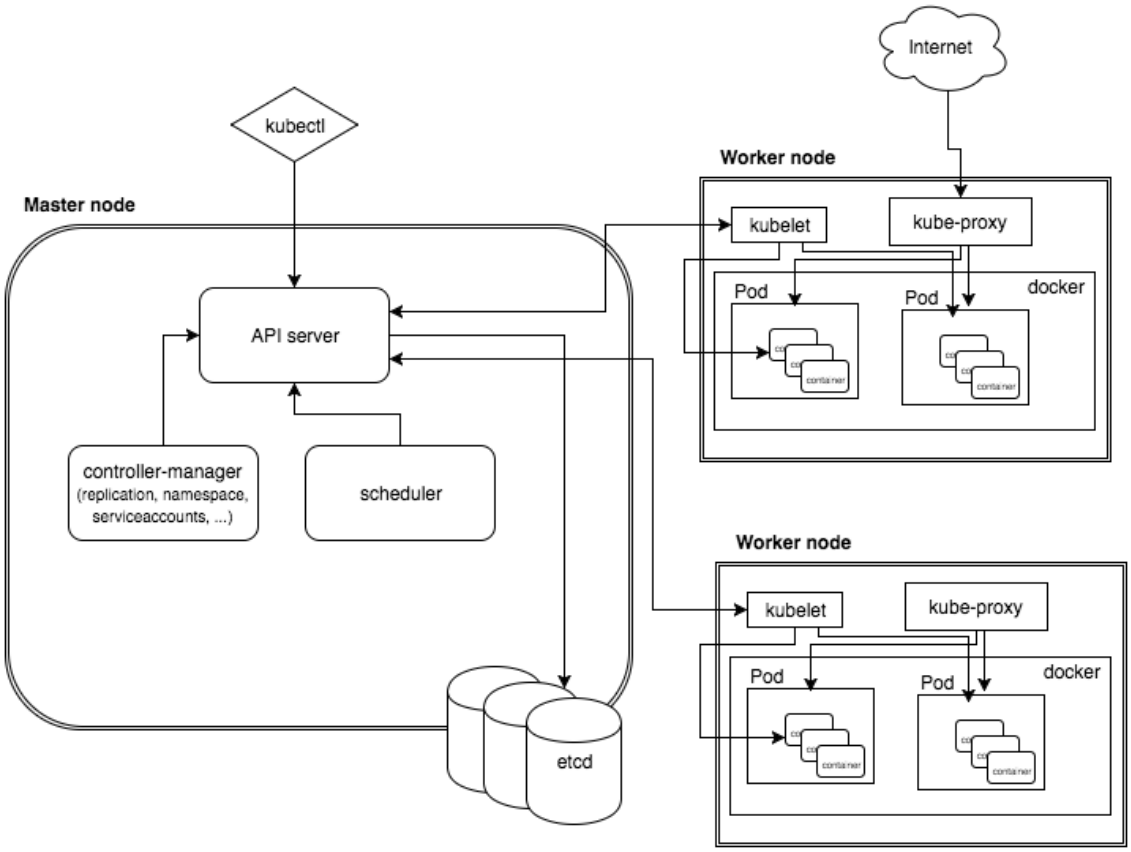
Kubernetes is a stack of services that work together to manage all the hosts, Kubernetes cluster is what we call them together AKA K8s. All those services or components are shown in the below three diagrams.

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

Visualpath Training & Consulting.
Flat no: 205, Nilgiri Block,Aditya Enclave, Ameerpet, Hyderabad, Phone No: - +91-970 445 5959, 961 824 5689 E-Mail ID : online.visualpath@gmail.com, Website : www.visualpath.in.

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

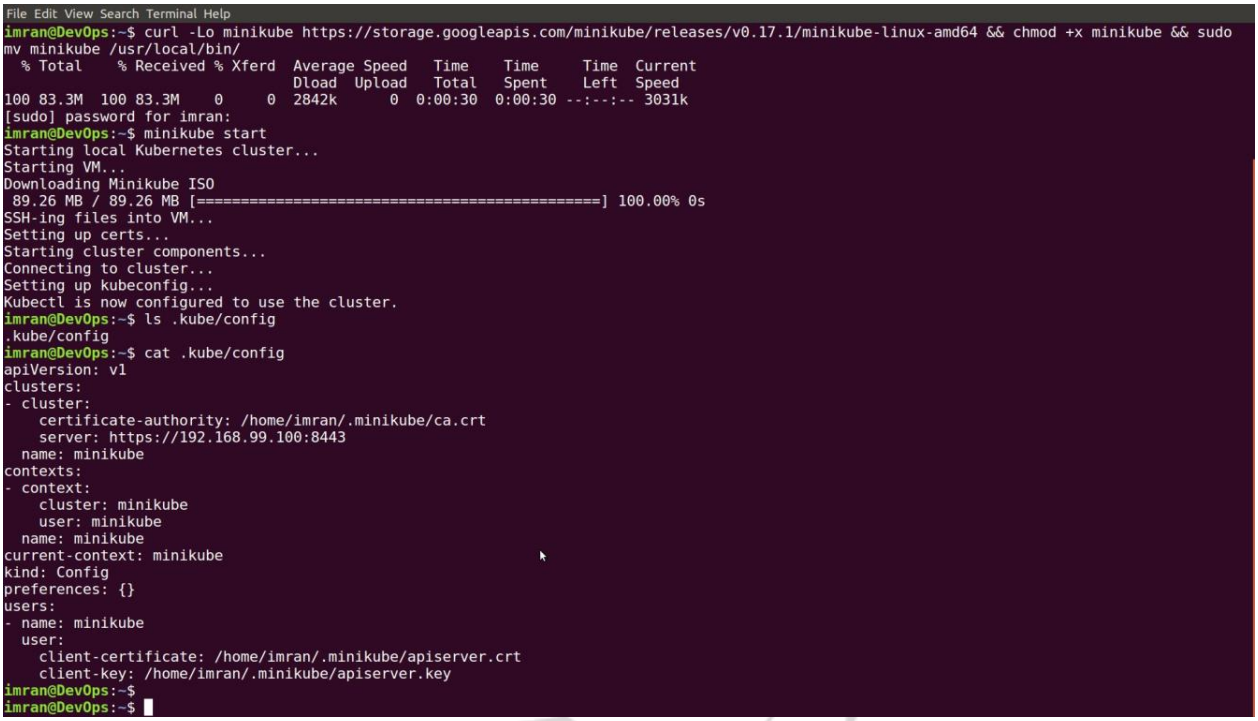
8. Minikube setup locally.

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
```

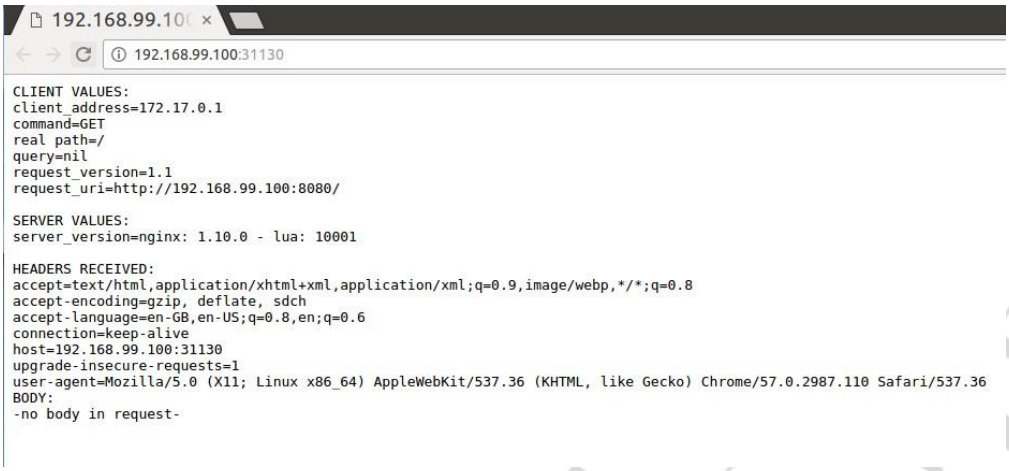
- Run kubectl command to verify if it's working.

```
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
-

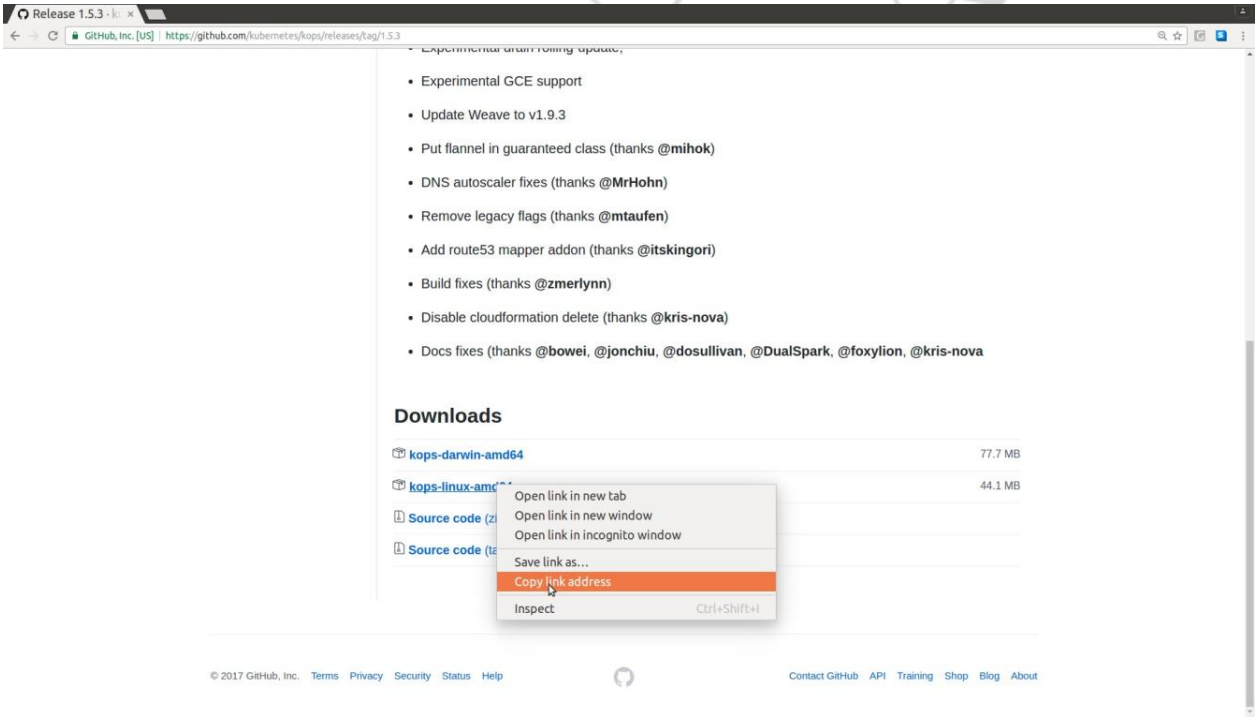
We will setup a Linux vm with vagrant+virtualbox and setup Kops in the vm.

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.

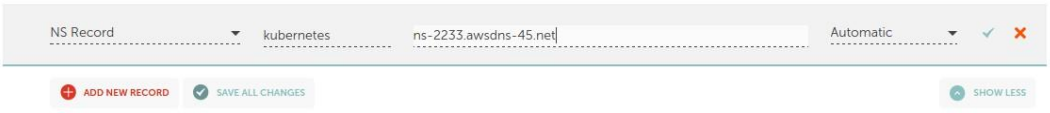
NS Record setup with Domain provider.

- 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-release/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.

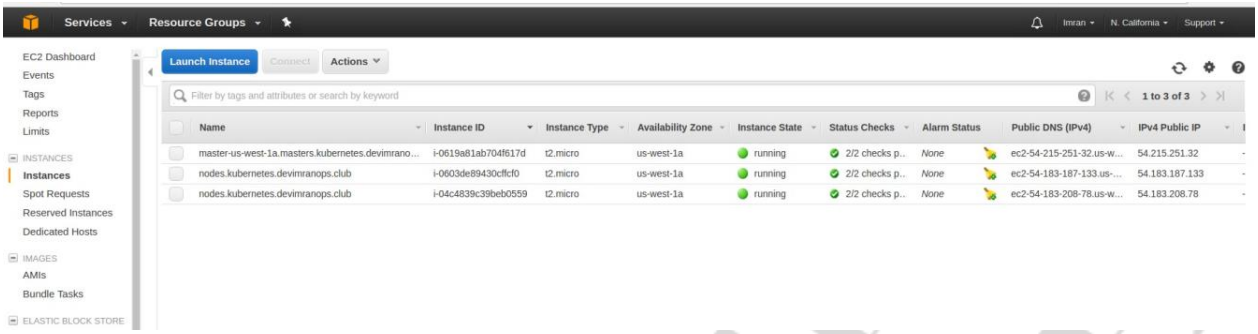
Cluster has not yet created, follow next command to create it.

```
Must specify --yes to apply changes

Cluster configuration has been created.

Suggestions:
 * list clusters with: kops get cluster
 * edit this cluster with: kops edit cluster kubernetes.devimranops.club
 * edit your node instance group: kops edit ig --name=kubernetes.devimranops.club nodes
 * edit your master instance group: kops edit ig --name=kubernetes.devimranops.club master-us-west-1a

Finally configure your cluster with: kops update cluster kubernetes.devimranops.club --yes

root@DevImranOps:~#
```

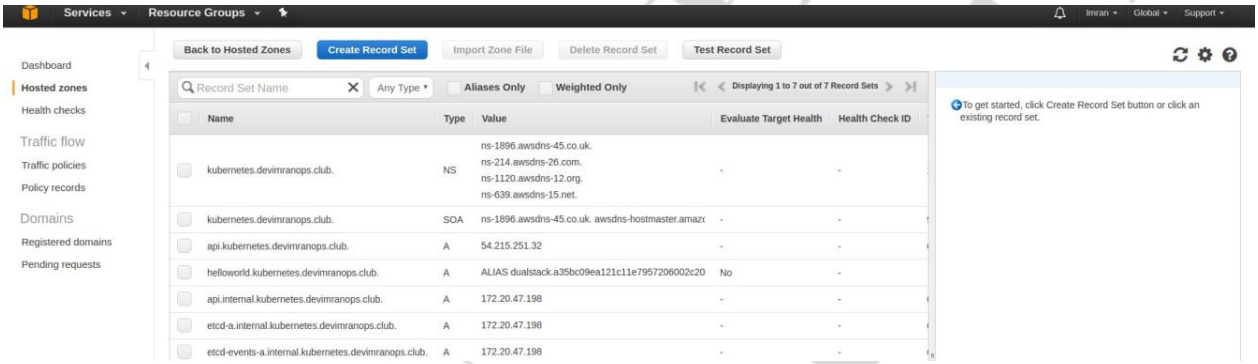
- Kops update cluster to create the cluster.

```
File Edit View Search Terminal Help
A s3 bucket is required to store cluster state information.
root@DevImranOps:~# kops update cluster kubernetes.devimranops.club --yes --state=s3://kops-state-86
I0330 18:12:19.022553    2871 executor.go:91] Tasks: 0 done / 56 total; 27 can run
I0330 18:12:20.430752    2871 vfs_castore.go:422] Issuing new certificate: "kubecfg"
I0330 18:12:20.590799    2871 vfs_castore.go:422] Issuing new certificate: "master"
I0330 18:12:20.771600    2871 vfs_castore.go:422] Issuing new certificate: "kubelet"
I0330 18:12:26.910766    2871 executor.go:91] Tasks: 27 done / 56 total; 12 can run
I0330 18:12:29.447253    2871 executor.go:91] Tasks: 39 done / 56 total; 15 can run
I0330 18:12:33.262695    2871 launchconfiguration.go:310] waiting for IAM instance profile "masters.kubernet
es.devimranops.club" to be ready
I0330 18:12:33.263549    2871 launchconfiguration.go:310] waiting for IAM instance profile "nodes.kubernetes
.devimranops.club" to be ready
I0330 18:12:45.146373    2871 executor.go:91] Tasks: 54 done / 56 total; 2 can run
I0330 18:12:46.170759    2871 executor.go:91] Tasks: 56 done / 56 total; 0 can run
I0330 18:12:46.170816    2871 dns.go:141] Pre-creating DNS records
I0330 18:12:50.787335    2871 update_cluster.go:208] Exporting kubecfg for cluster
Kops has set your kubectl context to kubernetes.devimranops.club

Cluster is starting.  It should be ready in a few minutes.

Suggestions:
 * validate cluster: kops validate cluster
 * list nodes: kubectl get nodes --show-labels
 * ssh to the master: ssh -i ~/.ssh/id_rsa admin@api.kubernetes.devimranops.club
 * read about installing addons: https://github.com/kubernetes/kops/blob/master/docs/addons.md

root@DevImranOps:~#
root@DevImranOps:~#
root@DevImranOps:~#
root@DevImranOps:~#
```

- Cluster setup would have been initiated, verify it by from AWS.



- Verify Route 53 subdomains.



- Validate cluster with kops command.

```
root@DevImranOps:~# kops validate cluster --state=s3://kops-state-86
Using cluster from kubectl context: kubernetes.devimranops.club

Validating cluster kubernetes.devimranops.club

INSTANCE GROUPS
NAME                    ROLE    MACHINETYPE     MIN     MAX     SUBNETS
master-us-west-1a       Master  t2.micro        1       1       us-west-1a
nodes                   Node    t2.micro        2       2       us-west-1a

NODE STATUS
NAME                                            ROLE    READY
ip-172-20-46-167.us-west-1.compute.internal     node    True
ip-172-20-47-198.us-west-1.compute.internal     master  True
ip-172-20-53-5.us-west-1.compute.internal       node    True

Your cluster kubernetes.devimranops.club is ready
root@DevImranOps:~#
```

- Kops creates ~/.kube/config file where it writes all the config details for KUBECTL.

```
File  Edit  View  Search  Terminal  Help

root@DevImranOps:~# cat ~/.kube/config
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUMxekNDQWIrZ0F3SUJBZ0lRRUZFU0tSQThyUk1zaFpGbElSDdLekFOQmdrcWhraUc5d
zBCQVFzRkFEQVYKTVJNd0VRWURWUVFERXdwcmRXSmxjbTVsZEdWek1CNFhEVEUzTURNeU9ERTRNVEl5TUZvdOZURVRNQkVHQTFVRUF4TUthM1ZpWl
hKdVpYUmxjekNDQVNJd0RRWUpLb1pJaHZjTkFRRUJCUUFECmdnRVBBRENDQVFvQ2dnRUJBTHIvWW93ZUtCbHFMRGpXRVhMak5GTmF5T2tUTFUyekJIWE5GczVhcldGb3dKRzEKcXpvS3R
YMG1Qd3c3aW5GdkFuQk96OGtGTCszVjIxaFFXeG1pWGpZcGptWDNpQTcvVHhwVGJTQ3gzdm9JSEY1SgprSnk3TThCSDRyMVdhaDYwVURQRCtQV2VWak9xcmZUUnFBMVJBMldiUzBTZTdT
VS9MVWJpUitvbmJUalRVNzRzCk5ZOUhvUVJKd3BJa0VTeHhQUG5tL25iSTNzbmVzQ3A0ZGNqdkZBUzEzV0xXNkozSVZacGlZNXV6RExsYUZyMDQKTlJEM2hjZjQ1eU1lWEdwOGdmQVhPS
2Y3cVpqZHdjRU5wV3NBNnVNUElLZW1UNkkrR1RkZTNnWk8zRG9ZenhKdgp0TUladzFOaVJDc0lYZzAwdXRwelp1WnY2M2lsSjBEOXdpUGhEM2tDQXdFQUFhTWpNQ0V3RGdZRFZSMFBBUU
gvCkJBUURBZ0VHTUE4R0ExVWRFd0VCL3dRRk1BTUJBZjh3RFFZSktvWklodmNOQVFFTEJRQURnZ0VCQUR6Yk41MjUKd3NDSy9HMHh1MDgwMUYwYzduVGJJSW5VSUriNTRvdXZnOHZkdEp
UWUthcEhDb1hQSHMwZm44MHk3SUpXd3dZWQpveTZSMndWN0pNandlN2cvbVFweHAvSmdkVlk0UzJPRmpjbTZtTzVtV1N1WFpZME9sU1RtSHY4OXdLM25uUmR6CmJwdHdTUDFRaGI3ZXNJ
bW5oS3FDYU54N1F4b3JoRVpLN1VqK0x0aEkxQlRLdGM4WWpNdGRoL1FSRTZKVVpScGcKa21hbFJzMXlWUEhuTXpoZ3RKMHdIajJTcnlBajJzSHJYcmdtQjVsTkNjZkFQbUEwemRjOFgvb
00xZjNoOGhZcApDMWQ1RUN6K01MQyt2VkxZeGdjb1FSRlFUWDVmU1JTeVN3d1c3Sk01NllFT0pVM0tPZFk1bEIvTjk0cUkraTFJCm8wZUltR3ZYdDlZa01iRT0KLS0tLS1FTkQgQ0VSVE
lGSUNBVEUtLS0tLQo=
    server: https://api.kubernetes.devimranops.club
  name: kubernetes.devimranops.club
contexts:
- context:
    cluster: kubernetes.devimranops.club
    user: kubernetes.devimranops.club
  name: kubernetes.devimranops.club
current-context: kubernetes.devimranops.club
kind: Config
preferences: {}
users:
- name: kubernetes.devimranops.club
  user:
    client-certificate-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUM0akNDQWNxZ0F3SUJBZ0lNRkxDN2FlaXFZVnovYi9MNU1BMEdDU3FHU0liM0RRRUJDd1VB
TUJVeEV6QVIKQmdOVkJBTVRDbXQxWW1WeWJtVjBaWE13SGhjTk1UY3dNekk0TVRneE1qSXpXaGNOTWpjd016STRNVGd4TWpJegpXakFTTVJBd0RnWURWUVFERXdkcmRXSmxZMlpuTUlJQ
klqQU5CZ2txaGtpRzl3MEJBUUVGQUFPQ0FROEFNSUlCCkNnS0NBUUVBb0M2N2x0cEZkNlNPNjZlNjY2elVrSHpQWWtOS3dWTHVXU3ZoT0tpV0Q4VENPNEFMNTJzY2x4bnAKU3Vzc1N4V3
VCRjIrR1IranF4NWRDR09BQ3FvT0VaemdpQTJxWHpxRWh5UUNaL1FLZ3NzYlkwdVd2cC83dzVHcAoxWWY1U2pWTGJOWk4wdVNsTEUwbU1EanV3UDBhUC92bjZKY3hhMGMyb3d5RTZRSkJ
kTGdmK3VCNlBZRXJJYVE5CmpoVnFldktHc0RJVUxNdUZtLzBhRGxvemxuR05xdFFzSFg3R3F0T0VvVFBHaDV3NDdZZWJHaU9SK0VJUDU0L1EKcVhUYmhkUU5MeG5oQ1RDRUlvVUorSGQ3
QXhQZWpXN1hXVnJCbGlqazBJRjRxa2NZYzU2UEx4WnUybG5ibDF2RQpVeWxkY0Q2cS9FVjFuZS9sS0NFSmxvbHJvbFZ1eHdJREFRQUJvelV3TXpBT0JnTlZIUThCQWY4RUJBTUNCNEF3C
kV3WURWUjBsQkF3d0NnWUlLd1lCQlFVSEF3SXdEQVlEVlIwVEFRSC9CQUl3QURBTkJna3Foa2lHOXcwQkFRc0YKQUFPQ0FRRUFBQ01KaHdzdER3a3oyTEcrVlJGUTk1RnArbmNCZnJOWj
l1djhtZjlPZ1M2eVR4K1dwM0ZNOTUrYQpDekt2SWp3TzJMczJUUkFuWlhNN2NiNXNiVFNFS0V4M09XNU4vcVJISXF1SWtWcTF0d3JwNWM5YXFOb0JWVEE0CnNGMk5OaHpQVmkrZXp4SEx
ZUnBSWGRqQW5oOTZHc0hNcS9PQ05DaWk0V1p6SFZLdmsxZ2EwM0RBajc1enNXOHAKczFudnBwY2lQL3JoTHFzcnJCOHFlVDdTUmVjek9sNWxncWVCRE9QMzk1UUlWMDBHTnhWUDNRajVu
dFpnWDZwaQpRdVJrUFV5THFTN3RCakVkdmh0MENsa2poeVFFNXJERm9VYXdENGlXRU53MU5ldit1cVNTbXlHdCtWVUZVb2dqClhpTmFjVU1hUlZVNnJkREhLTW1HUmNOMlM1K2FHQT09C
i0tLS0tRU5EIENFUlRJRklDQVRFLS0tLS0K
    client-key-data: LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFb3dJQkFBS0NBUUVBb0M2N2x0cEZkNlNPNjZlNjY2elVrSHpQWWtOS3dWTHVXU3ZoT0tpV0Q4
```

- We can now start using kubectl command to interact with our kubernetes cluster.

Check the all the nodes in cluster.

```
root@DevImranOps:~# kubectl get nodes
NAME                                        STATUS         AGE
ip-172-20-46-167.us-west-1.compute.internal   Ready          14m
ip-172-20-47-198.us-west-1.compute.internal   Ready,master   15m
ip-172-20-53-5.us-west-1.compute.internal     Ready          14m
root@DevImranOps:~#
```

- Let’s run a container on this cluster.

```
File  Edit  View  Search  Terminal  Help
root@DevImranOps:~# kubectl run hello-minikube --image=gcr.io/google_containers/echoserver:1.4 --port=8080
deployment "hello-minikube" created
root@DevImranOps:~# kubectl expose deployment hello-minikube --type=NodePort
service "hello-minikube" exposed
root@DevImranOps:~# kubectl get service
NAME            CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
hello-minikube   100.64.74.8   <nodes>       8080:31496/TCP   14s
kubernetes      100.64.0.1    <none>        443/TCP          22m
root@DevImranOps:~#
```

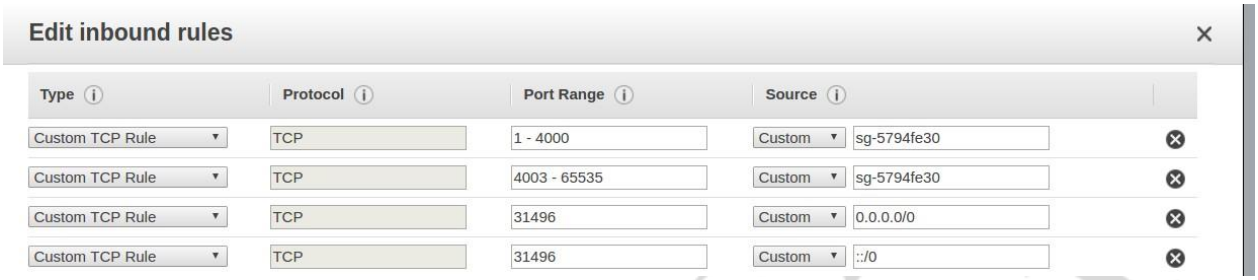
- Deployment and service created, service is exposed on port 31496.

We must allow port 31496 in Security group of Master node to access this service.

- Go to AWS Ec2 => Select Master node => Security groups =>masters.kubernetes.

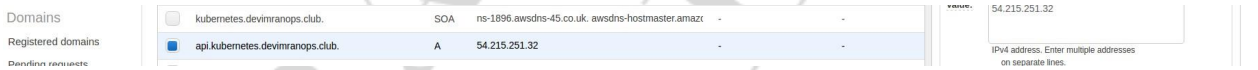
Inbound => Edit.

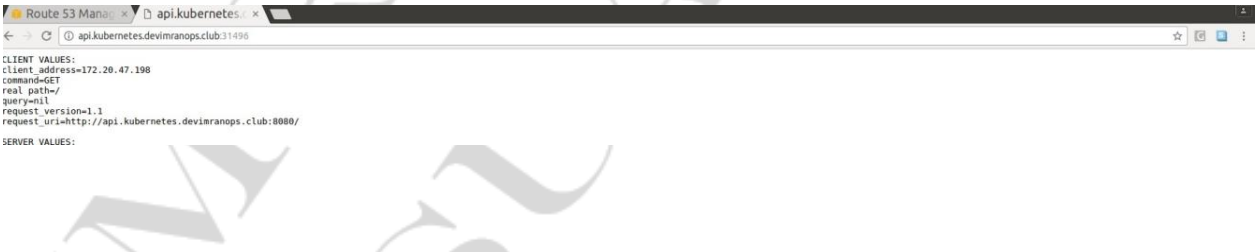
Add the port number and source from anywhere/MyIP.



- Get the master node Public IP and access it from browser on the exposed port.

- Instead of Public IP of master node we can also use names assigned by Route53.





- If you are not using the cluster you can delete it.

10. Containers & Images

In this exercise, we will build a docker image for nodejs app.

Will use dockerhub as registry and will run that image on our kubernetes cluster.

Check the docker-demo directory. Verify three files shown below.

- Dockerfile

```
# cat Dockerfile
FROM node:4.6
WORKDIR /app
ADD . /app
RUN npm install
EXPOSE 3000
CMD npm start
```

- index.js

```
# cat index.js
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('Hello World!');
});

var server = app.listen(3000, function () {
  var host = server.address().address;
  var port = server.address().port;

  console.log('Example app listening at http://%s:%s', host, port);
});
```

- package.json

```
# cat package.json
{
  "name": "myapp",
  "version": "0.0.1",
  "private": true,
```

```
  "scripts": {
    "start": "node index.js"
  },
  "engines": {
    "node": "^4.6.1"
  },
  "dependencies": {
    "express": "^4.14.0",
    "mysql": "^2.10.2"
  }
}
```

- Install docker.

```
root@DevImranOps:~/kube/docker-demo# apt-get install docker.io
Reading package lists... Done
Building dependency tree
Reading state information... Done
docker.io is already the newest version (1.12.6-0ubuntu1~16.04.1).
```

- - Build Image.

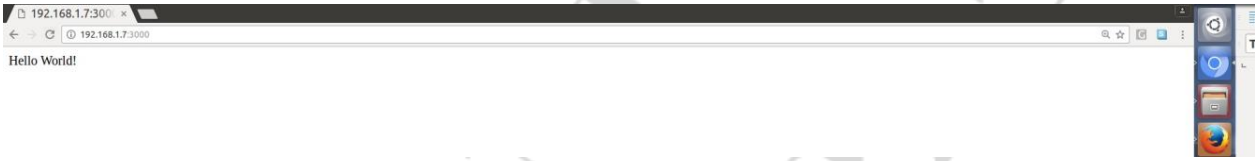
Images name should match with your dockerhub account/reponame. For example my account name in dockerhub is “visualpath”, so my image name is visualpath/k8s-demo.

```
File  Edit  View  Search  Terminal  Help

root@DevImranOps:~/kube# cd docker-demo/
root@DevImranOps:~/kube/docker-demo# cat Dockerfile
FROM node:4.6
WORKDIR /app
ADD . /app
RUN npm install
EXPOSE 3000
CMD npm start
root@DevImranOps:~/kube/docker-demo# docker build -t visualpath/k8s-demo  .
Sending build context to Docker daemon 90.11 kB
Step 1 : FROM node:4.6
 ---> e834398209c1
Step 2 : WORKDIR /app
 ---> Using cache
 ---> d808ac4780ac
Step 3 : ADD . /app
 ---> Using cache
 ---> 304268d58a39
Step 4 : RUN npm install
 ---> Using cache
 ---> 8573156bdb29
Step 5 : EXPOSE 3000
 ---> Using cache
 ---> ac8a61ac3689
Step 6 : CMD npm start
 ---> Using cache
 ---> 03be3f739a76
Successfully built 03be3f739a76
root@DevImranOps:~/kube/docker-demo#
```

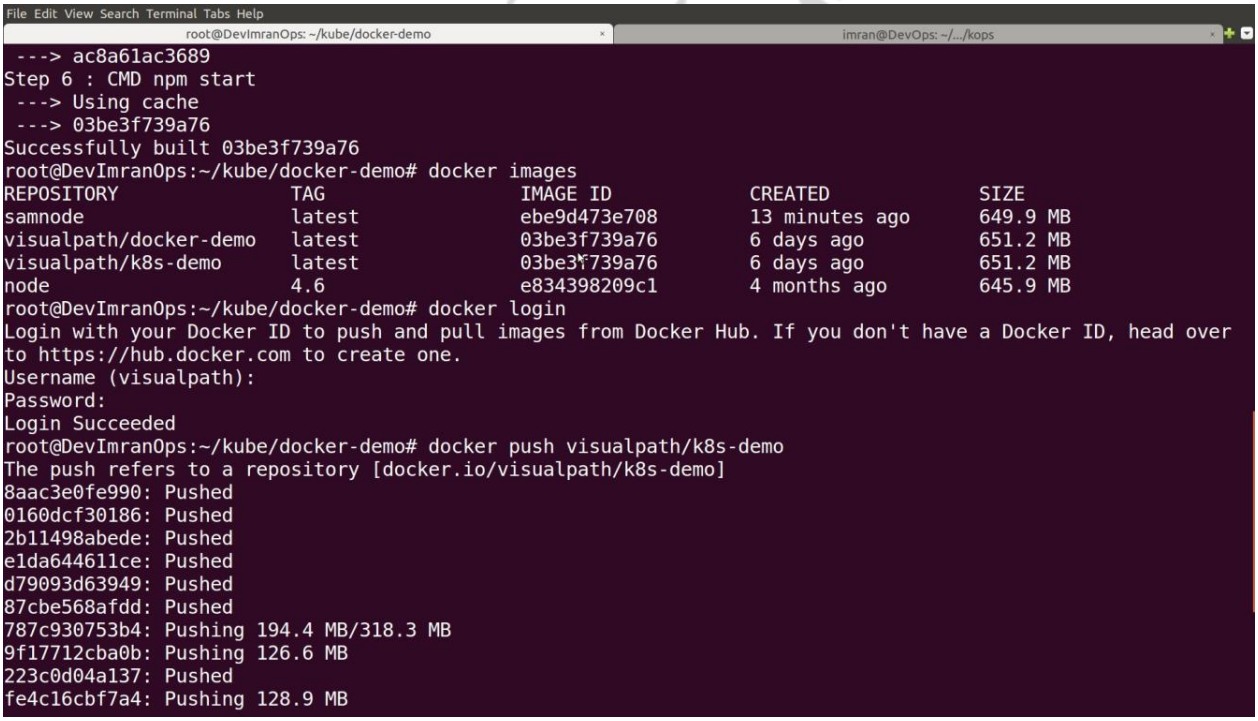
- Run image locally and test.

- Test it from browser

Enter IP address of your vm(kops vm) where your container is running and port number.



- Push Image to Dockerhub



11. First App on Kubernetes.

- Let’s run our newly built application on the Kubernetes cluster
- Before we can launch a container, we need to create a POD definition.
  - A Pod describes an application running on Kubernetes.
  - A can contain one or more tightly coupled containers, that make up the app.
    - Those apps can easily communicate with each other using their local port numbers.

  Our app only has one container.

- Read pod definition.

```
root@DevImranOps:~/kube# cat first-app/helloworld.yml
apiVersion: v1
kind: Pod
metadata:
  name: nodehelloworld.example.com
  labels:
    app: helloworld
spec:
  containers:
  - name: k8s-demo
    image: visualpath/k8s-demo
    ports:
    - name: nodejs-port
      containerPort: 3000
root@DevImranOps:~/kube#
```

- Use kubectl to create pod on cluster.

```
root@DevImranOps:~/kube# kubectl create -f first-app/helloworld.yml
pod "nodehelloworld.example.com" created
root@DevImranOps:~/kube# kubectl get pod
NAME                        READY     STATUS    RESTARTS   AGE
nodehelloworld.example.com   1/1       Running   0          3m
```

```
root@DevImranOps:~/kube# kubectl describe pod nodehelloworld.example.com
Name:           nodehelloworld.example.com
Namespace:      default
Node:           ip-172-20-34-251.us-west-1.compute.internal/172.20.34.251
Start Time:     Sat, 01 Apr 2017 19:18:20 +0000
Labels:         app=helloworld
Status:         Running
IP:             100.96.2.2
Controllers:    <none>
```

- Useful Kubectl 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

- Executing commands on container.

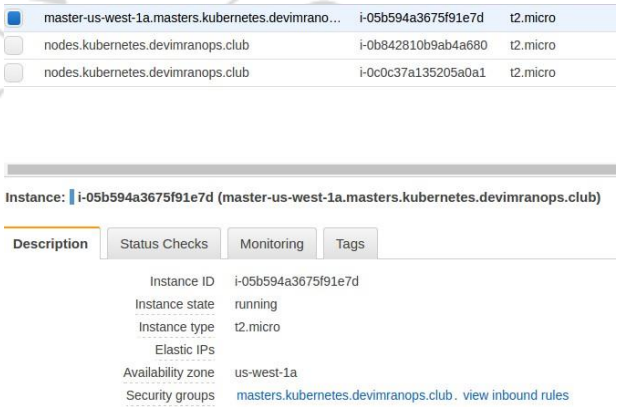
```
File  Edit  View  Search  Terminal  Help
root@DevImranOps:~/kube# kubectl exec nodehelloworld.example.com -- ls /app
Dockerfile
docker-compose.yml
index-db.js
index.js
misc
node_modules
package.json
root@DevImranOps:~/kube# kubectl exec nodehelloworld.example.com -- touch  /app/testfile.txt
root@DevImranOps:~/kube# kubectl exec nodehelloworld.example.com -- ls /app
Dockerfile
docker-compose.yml
index-db.js
index.js
misc
node_modules
package.json
testfile.txt
root@DevImranOps:~/kube#
```

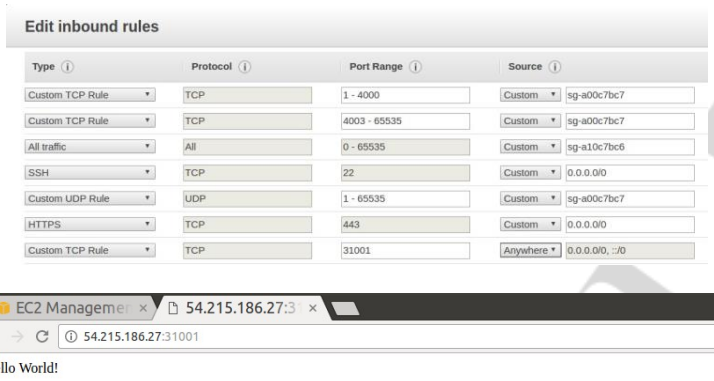
12. Services

- Creating service for our app.

```
File Edit View Search Terminal Help
root@DevImranOps:~/kube# cat first-app/helloworld-nodeport-service.yml
apiVersion: v1
kind: Service
metadata:
  name: helloworld-service
spec:
  ports:
  - port: 31001
    nodePort: 31001
    targetPort: nodejs-port
    protocol: TCP
  selector:
    app: helloworld
  type: NodePort
root@DevImranOps:~/kube# kubectl create -f first-app/helloworld-nodeport-service.yml
service "helloworld-service" created
root@DevImranOps:~/kube# kubectl get service
NAME                 CLUSTER-IP     EXTERNAL-IP   PORT(S)           AGE
helloworld-service   100.68.5.83    <nodes>       31001:31001/TCP   5s
kubernetes           100.64.0.1     <none>        443/TCP           35m
root@DevImranOps:~/kube#
```

- Accessing service from outside world.
  - Open MasterNode Security group from AWS.
  - Allow port 31001 from MyIP/Anywhere. **Port could be different**, please check your service port(kubectl get service).
  - Open browser enter your master node public IP and 31001 port.





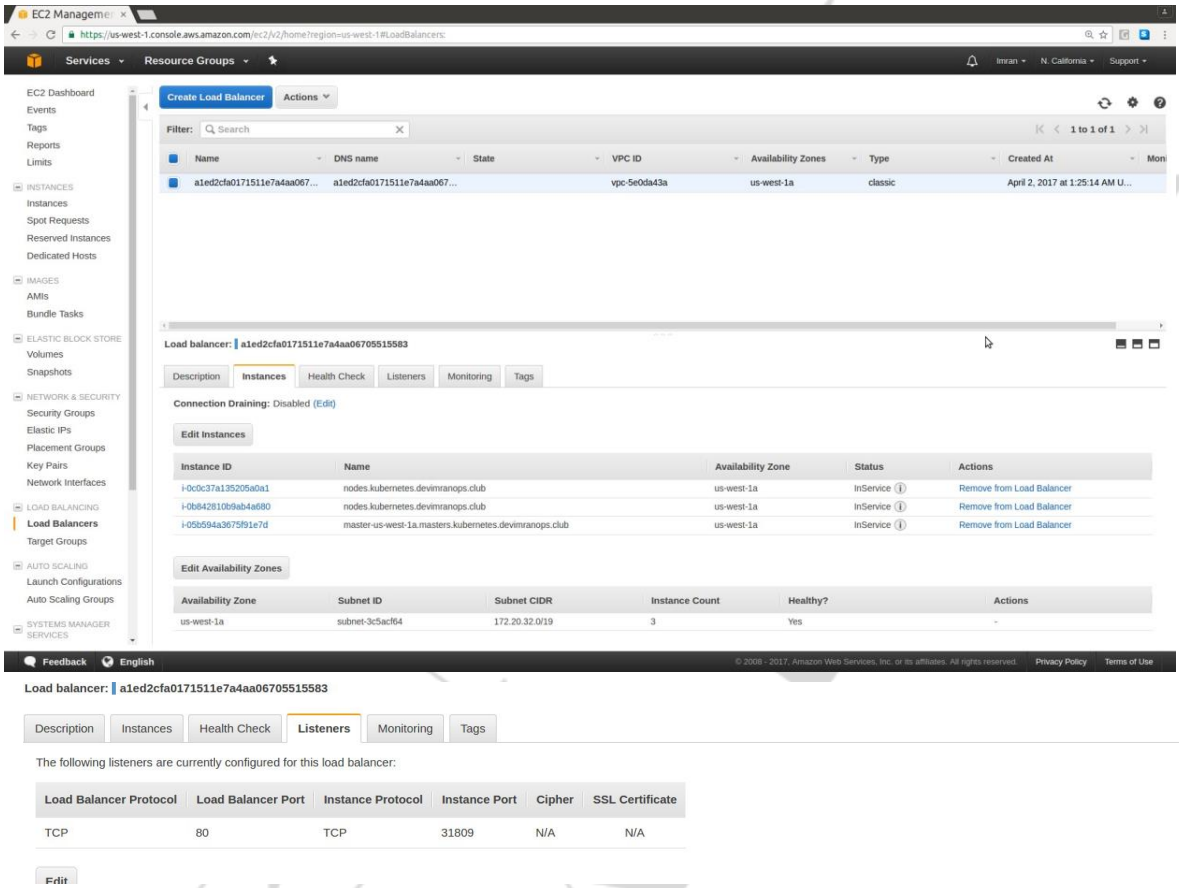
- Deleting service.

```
root@DevImranOps:~/kube# kubectl get svc
NAME                 CLUSTER-IP    EXTERNAL-IP   PORT(S)           AGE
helloworld-service   100.68.5.83   <nodes>       31001:31001/TCP   14m
kubernetes           100.64.0.1    <none>        443/TCP           49m
root@DevImranOps:~/kube# kubectl delete service helloworld-service
service "helloworld-service" deleted
root@DevImranOps:~/kube#
```

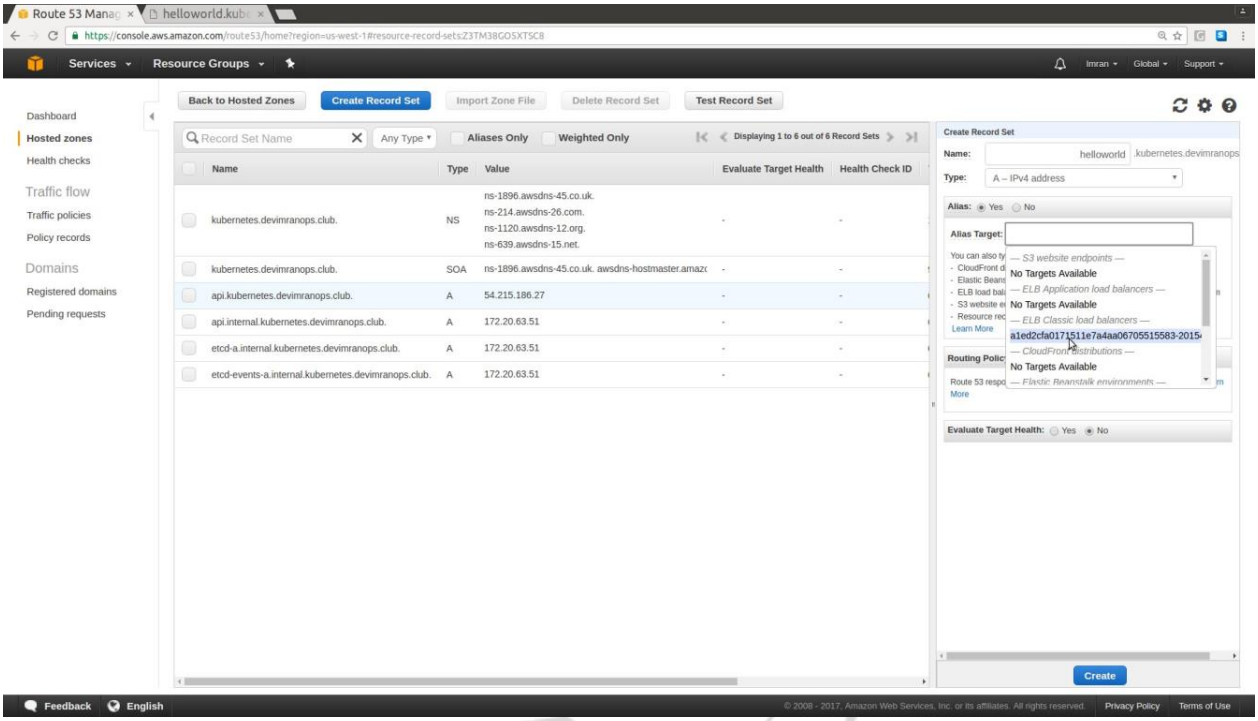
- Creating AWS ELB as a Service for our pod.

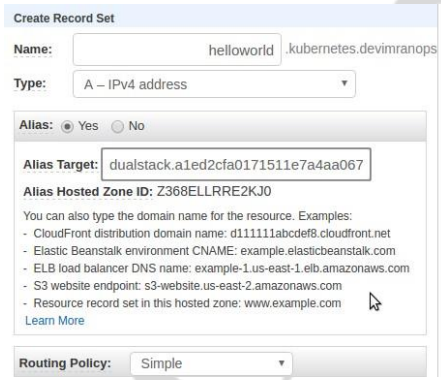
```
File Edit View Search Terminal Help

root@DevImranOps:~/kube# cat first-app/helloworld-service.yml
apiVersion: v1
kind: Service
metadata:
  name: helloworld-service
spec:
  ports:
  - port: 80
    targetPort: nodejs-port
    protocol: TCP
  selector:
    app: helloworld
  type: LoadBalancer
root@DevImranOps:~/kube# kubectl create -f first-app/helloworld-service.yml
service "helloworld-service" created
root@DevImranOps:~/kube# kubectl get service
NAME                CLUSTER-IP      EXTERNAL-IP        PORT(S)        AGE
helloworld-service  100.71.244.70   a1ed2cfa01715...   80:31809/TCP   11s
kubernetes          100.64.0.1      <none>             443/TCP        51m
root@DevImranOps:~/kube# kubectl describe service helloworld-service
Name:                   helloworld-service
Namespace:              default
Labels:                 <none>
Selector:               app=helloworld
Type:                   LoadBalancer
IP:                     100.71.244.70
LoadBalancer Ingress:   a1ed2cfa0171511e7a4aa06705515583-201543200.us-west-1.elb.amazonaws.com
Port:                   <unset> 80/TCP
NodePort:               <unset> 31809/TCP
Endpoints:              100.96.2.2:3000
Session Affinity:       None
Events:
  FirstSeen     LastSeen        Count   From                    SubObjectPath   Type            Reason      Message
  ---------     --------        -----   ----                    -------------   --------        ------      -------
  27s           27s             1       {service-controller }                   Normal          CreatingLoadBalancer        Creating load bal
ancer
  24s           24s             1       {service-controller }                   Normal          CreatedLoadBalancer Created load balancer
root@DevImranOps:~/kube#
root@DevImranOps:~/kube#
root@DevImranOps:~/kube#
```

- Verify ELB from AWS.



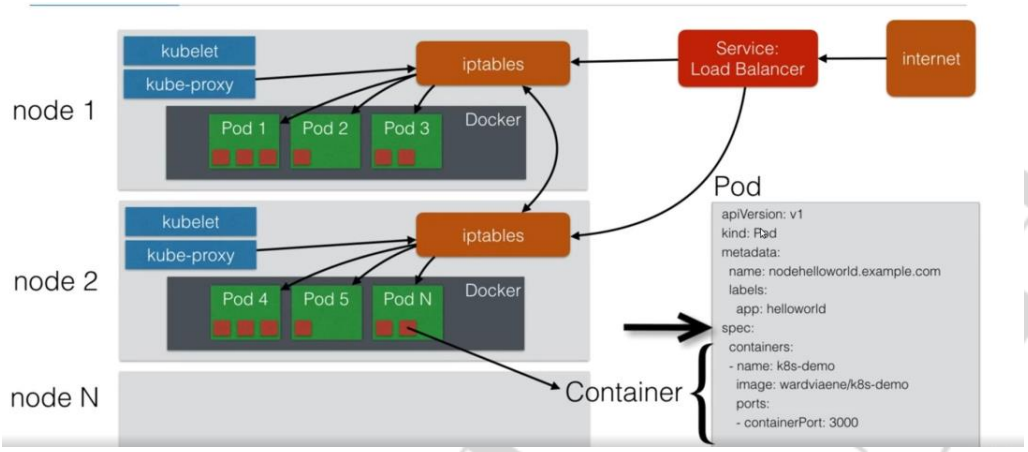
- Assign DNS name to our ELB from Route 53.
  - Go to AWS route53 dashboard => Hosted Zone => Create Record Set
    => Name:helloworld => Alias:Yes => ELB Classic load balancer
  - Select your ELB and use helloworld.kubernetes.devimranops.club (diff domain in your case.)





Visualpath Training & Consulting.
Flat no: 205, Nilgiri Block,Aditya Enclave, Ameerpet, Hyderabad, Phone No: - +91-970 445 5959, 961 824 5689 E-Mail ID : online.visualpath@gmail.com, Website : www.visualpath.in.

13. Node Architecture.



14. Scaling Pods

Scaling

- If your application is stateless you can horizontally scale it.
  - Stateless means your application does not store files/data on local file system.
  - All Databases are statefull. They store database files locally.
- Most Web applications can be made stateless.
  - Session Management needs to be done outside the container
  - Any file that need to be saved can't be saved locally on the computer

- Scaling in Kubernetes can be done using the Replication Controller.
- The replication controller will ensure a specified number of pod replicas will run at all time
- A pod created with the replica controller will automatically be replaced if they fil, 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.

◆ Replicating our pod two times.

```
root@DevImranOps:~/kube# cat replication-controller/helloworld-repl-controller.yml
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: visualpath/k8s-demo
        ports:
        - name: nodejs-port
          containerPort: 3000
```

◆ Check pod & rc (replication controller) with kubectl get & kubectl describe command.

```
File  Edit  View  Search  Terminal  Help
root@DevImranOps:~/kube# kubectl create -f replication-controller/helloworld-repl-controller.yml
replicationcontroller "helloworld-controller" created
root@DevImranOps:~/kube# kubectl get rc
NAME                   DESIRED   CURRENT   READY     AGE
helloworld-controller   2         2         2         8s
root@DevImranOps:~/kube# kubectl get pod
NAME                         READY     STATUS    RESTARTS   AGE
helloworld-controller-514j9   1/1       Running   0          14s
helloworld-controller-htdc3   1/1       Running   0          14s
root@DevImranOps:~/kube# kubectl describe pod helloworld-controller-514j9
Name:           helloworld-controller-514j9
Namespace:      default
Node:           ip-172-20-34-251.us-west-1.compute.internal/172.20.34.251
Start Time:     Sat, 01 Apr 2017 20:44:52 +0000
Labels:         app=helloworld
Status:         Running
IP:             100.96.2.4
Controllers:    ReplicationController/helloworld-controller
Containers:
  k8s-demo:
    Container ID:       docker://8e501fbf12b96963e092a0b5e5623707807a3262f24fba13211c825156b4ca1e
    Image:              visualpath/k8s-demo
    Image ID:           docker-pullable://visualpath/k8s-demo@sha256:877837260a5e78becebb3ed03f0e72ba3acfae9d6fd8f0cc0d83c4a82889d1b1
    Port:               3000/TCP
    Requests:
      cpu:              100m
    State:              Running
      Started:          Sat, 01 Apr 2017 20:44:55 +0000
    Ready:              True
    Restart Count:      0
    Volume Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-348lc (ro)
    Environment Variables:      <none>
Conditions:
  Type          Status
  Initialized   True
  Ready         True
  PodScheduled  True
Volumes:
  default-token-348lc:
    Type:       Secret (a volume populated by a Secret)
```

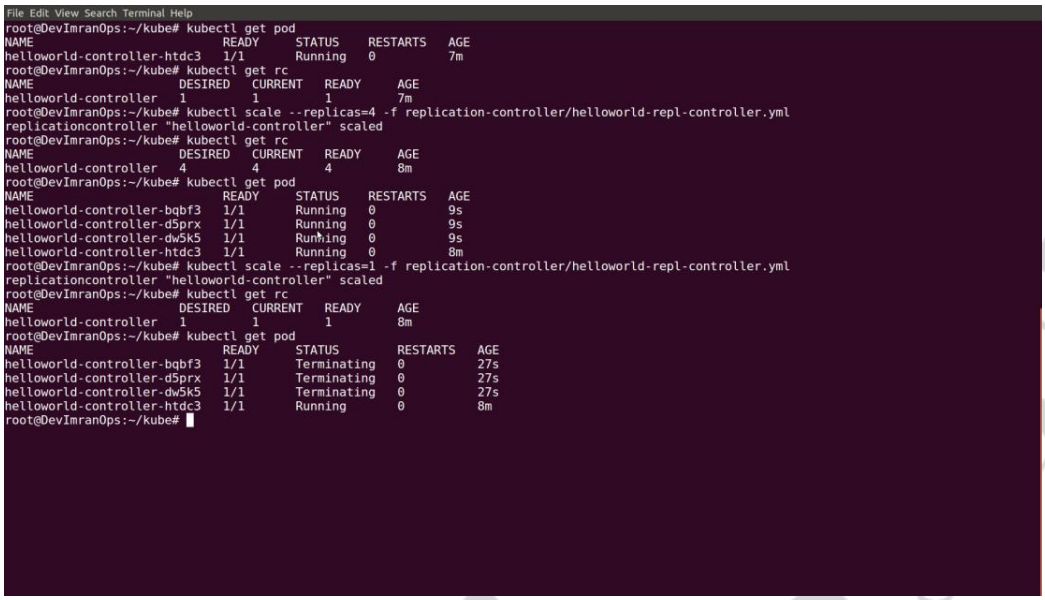
- Delete one pod to see scaling.

```
root@DevImranOps:~/kube# kubectl get pod
kuNAME                         READY     STATUS    RESTARTS   AGE
helloworld-controller-514j9   1/1       Running   0          1m
helloworld-controller-htdc3   1/1       Running   0          1m
root@DevImranOps:~/kube# kubectl delete pod helloworld-controller-514j9
pod "helloworld-controller-514j9" deleted
root@DevImranOps:~/kube# kubectl get pod
NAME                         READY     STATUS        RESTARTS   AGE
helloworld-controller-514j9   1/1       Terminating   0          1m
helloworld-controller-8lmxh   1/1       Running       0          4s
helloworld-controller-htdc3   1/1       Running       0          1m
```

- Scale Up/Down the replication controller.

```
root@DevImranOps:~/kube# kubectl get rc
NAME                   DESIRED   CURRENT   READY     AGE
helloworld-controller   2         2         2         3m
root@DevImranOps:~/kube# kubectl scale --replicas=4 rc helloworld-controller
replicationcontroller "helloworld-controller" scaled
root@DevImranOps:~/kube# kubectl get pod
NAME                         READY     STATUS    RESTARTS   AGE
helloworld-controller-8lmxh   1/1       Running   0          2m
helloworld-controller-htdc3   1/1       Running   0          4m
helloworld-controller-hvzbr   1/1       Running   0          8s
helloworld-controller-qtc93   1/1       Running   0          8s
root@DevImranOps:~/kube# kubectl scale --replicas=1 rc helloworld-controller
replicationcontroller "helloworld-controller" scaled
root@DevImranOps:~/kube# kubectl get pod
NAME                         READY     STATUS        RESTARTS   AGE
helloworld-controller-8lmxh   1/1       Terminating   0          2m
helloworld-controller-htdc3   1/1       Running       0          4m
helloworld-controller-hvzbr   1/1       Terminating   0          24s
helloworld-controller-qtc93   1/1       Terminating   0          24s
root@DevImranOps:~/kube#
```

- Scale Up/Down with the replication controller's definition file.



15. Deployments

Replication Set

- Replica set is the next generation Replication Controller.
- It supports a new selector that can do selection based on filtering according a set of values.
  - 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

- 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 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.
- With a deployment object, you can:
  - Create a deployment (e.g deploying an app)
  - Update a deployment (e.g deploying a latest version)
  - Do rolling updates (zero downtime deployments)
  - Roll back to a previous version
  - Pause/Resume a deployment (e.g to roll-out to only a certain percentage)

- Sample Deployment Definition.

```
root@DevImranOps:~/kube# cat deployment/helloworld.yml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: helloworld-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: helloworld
    spec:
      containers:
      - name: k8s-demo
        image: visualpath/k8s-demo
        ports:
        - name: nodejs-port
          containerPort: 3000
root@DevImranOps:~/kube#
```

- 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:
  - Create deployment
  - Get info on deployment, Replication set (rs), pod, rollout status

```
root@DevImranOps:~/kube# kubectl create -f deployment/helloworld.yml
deployment "helloworld-deployment" created
root@DevImranOps:~/kube# kubectl get deployment
NAME                   DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
helloworld-deployment   3         3         3            0           27s
root@DevImranOps:~/kube# kubectl get rs
NAME                              DESIRED   CURRENT   READY     AGE
helloworld-deployment-706070162   3         3         3         37s
root@DevImranOps:~/kube# kubectl get pods
NAME                                    READY     STATUS    RESTARTS   AGE
helloworld-deployment-706070162-8n3bs   1/1       Running   0          46s
helloworld-deployment-706070162-nlpdl   1/1       Running   0          46s
helloworld-deployment-706070162-q350d   1/1       Running   0          46s
root@DevImranOps:~/kube# kubectl get pods --show-labels
NAME                                    READY     STATUS    RESTARTS   AGE       LABELS
helloworld-deployment-706070162-8n3bs   1/1       Running   0          59s       app=helloworld,pod-template-hash=706070162
helloworld-deployment-706070162-nlpdl   1/1       Running   0          59s       app=helloworld,pod-template-hash=706070162
helloworld-deployment-706070162-q350d   1/1       Running   0          59s       app=helloworld,pod-template-hash=706070162
root@DevImranOps:~/kube# kubectl rollout status deployment helloworld-deployment
deployment "helloworld-deployment" successfully rolled out
root@DevImranOps:~/kube#
```

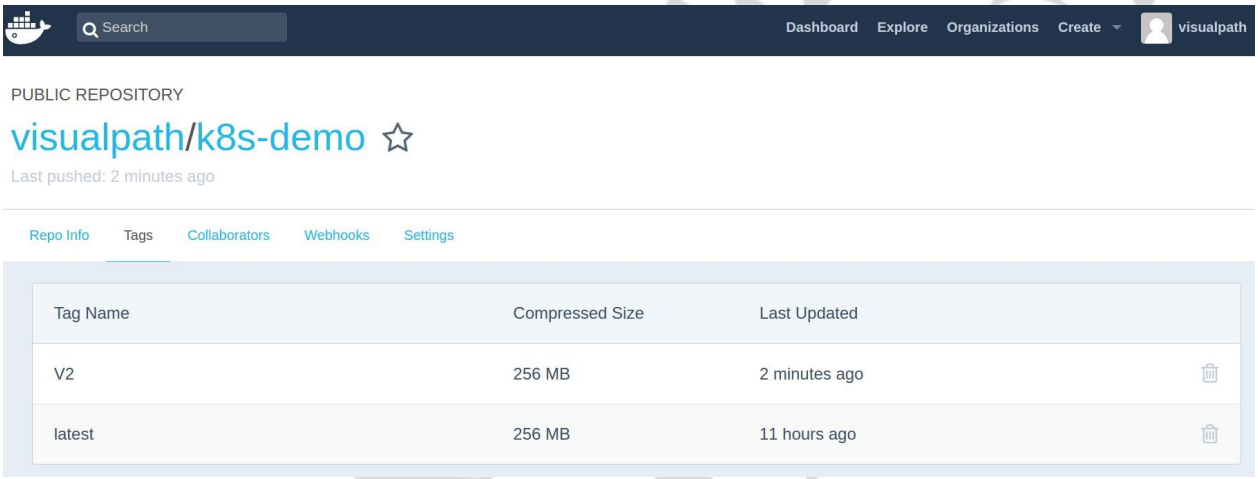
- Build k8s-demo image with next version, V2.
  - Copy docker-demo directory content into another directory.
  - Edit index.js file, update string in res.send to something like “version 2”.
  - Build the image with tag V2.
  - Push the latest image to dockerhub registry.
  - Verify from dockerhub

```
root@DevImranOps:~/kube/docker-demo# ls
docker-compose.yml  Dockerfile  index-db.js  index.js  misc  package.json
root@DevImranOps:~/kube/docker-demo# cp * -r /tmp/hellov2/
root@DevImranOps:~/kube/docker-demo# cd /tmp/hellov2/
root@DevImranOps:/tmp/hellov2# vi index.js
```

```
root@DevImranOps:/tmp/hellov2# cat index.js
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('Hello Everybody, this is version 2');
});

var server = app.listen(3000, function () {
  var host = server.address().address;
  var port = server.address().port;

  console.log('Example app listening at http://%s:%s', host, port);
});
```

```
root@DevImranOps:/tmp/hellov2# docker build -t visualpath/k8s-demo:V2 .
Sending build context to Docker daemon 8.704 kB
Step 1 : FROM node:4.6
 ---> e834398209c1
Step 2 : WORKDIR /app
 ---> Using cache
 ---> d808ac4780ac
Step 3 : ADD . /app
 ---> 7f2f05cfe161
Removing intermediate container 356b8e9ec00a
Step 4 : RUN npm install
 ---> Running in 5f01e5862183
npm info it worked if it ends with ok
```

```
root@DevImranOps:/tmp/hellov2# docker images
REPOSITORY                TAG                 IMAGE ID            CREATED             SIZE
visualpath/k8s-demo       V2                  39b2b75413a5        2 minutes ago       651.2 MB
<none>                    <none>              5418af191c46        7 minutes ago       651.2 MB
samnode                   latest              ebe9d473e708        11 hours ago        649.9 MB
visualpath/docker-demo    latest              03be3f739a76        6 days ago          651.2 MB
node                      4.6                 e834398209c1        4 months ago        645.9 MB
root@DevImranOps:/tmp/hellov2# docker push visualpath/k8s-demo:V2
The push refers to a repository [docker.io/visualpath/k8s-demo]
2987eec31172: Pushed
996cf1e370b0: Pushed
2b11498abede: Layer already exists
e1da644611ce: Layer already exists
d79093d63949: Layer already exists
87cbe568afdd: Layer already exists
787c930753b4: Layer already exists
9f17712cba0b: Layer already exists
223c0d04a137: Layer already exists
fe4c16cbf7a4: Layer already exists
V2: digest: sha256:8496f6a052ca8b43eb97a6aa48fa4b17a8386bec148ec628051611bd5e47d2c3 size: 2420
```



- Expose the existing deployment with NodePort.

```
root@DevImranOps:~/kube# kubectl expose deployment helloworld-deployment --type=NodePort
service "helloworld-deployment" exposed
root@DevImranOps:~/kube# kubectl get service
NAME                     CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
helloworld-deployment    100.70.171.200   <nodes>       3000:32400/TCP   10s
kubernetes               100.64.0.1       <none>        443/TCP          35m
root@DevImranOps:~/kube# kubectl describe service helloworld-deployment
Name:                    helloworld-deployment
Namespace:               default
Labels:                  app=helloworld
Selector:                app=helloworld
Type:                    NodePort
IP:                      100.70.171.200
Port:                    <unset> 3000/TCP
NodePort:                <unset> 32400/TCP
Endpoints:               100.96.1.6:3000,100.96.2.4:3000,100.96.2.5:3000
Session Affinity:        None
No events.
```

- NodePort is 32400 as per above output.
- Allow this port in Security group of master node so that we can connect to this service.

- Verify Master node IP and exposed port.

- Upgrade the image version to V2 from Deployment.

```
root@DevImranOps:~/kube# kubectl set image deployment/helloworld-deployment k8s-demo=visualpath/k8s-demo:V2
deployment "helloworld-deployment" image updated
root@DevImranOps:~/kube# kubectl get pod
NAME                                  READY     STATUS        RESTARTS   AGE
helloworld-deployment-3758606681-8dg52   1/1       Terminating   0          1m
helloworld-deployment-3758606681-bg12n   1/1       Terminating   0          1m
helloworld-deployment-3758606681-p423x   1/1       Terminating   0          1m
helloworld-deployment-3966748500-8grfs   1/1       Running       0          4s
helloworld-deployment-3966748500-9p0hq   1/1       Running       0          3s
helloworld-deployment-3966748500-r9t6d   1/1       Running       0          4s
root@DevImranOps:~/kube# kubectl rollout status deployment helloworld-deployment
deployment "helloworld-deployment" successfully rolled out
root@DevImranOps:~/kube#
```

- Verify from browser & curl.


```
root@DevImranOps:~/kube# curl 54.183.128.169:32400
Hello Everybody, this is version 2root@DevImranOps:~/kube#
```

- Deployment Rollout History.

```
root@DevImranOps:~/kube#
root@DevImranOps:~/kube# kubectl rollout history deployment helloworld-deployment
deployments "helloworld-deployment"
REVISION        CHANGE-CAUSE
1               <none>
3               <none>
4               <none>
5               <none>
```

- Rollback previous version.

```
root@DevImranOps:~/kube# kubectl rollout undo deployment helloworld-deployment
deployment "helloworld-deployment" rolled back
root@DevImranOps:~/kube# kubectl get pod
NAME                                  READY     STATUS        RESTARTS   AGE
helloworld-deployment-3758606681-bnntb   1/1       Running       0          7s
helloworld-deployment-3758606681-ltx5r   1/1       Running       0          9s
helloworld-deployment-3758606681-wb0w6   1/1       Running       0          9s
helloworld-deployment-3966748500-8grfs   1/1       Terminating   0          15m
helloworld-deployment-3966748500-9p0hq   1/1       Terminating   0          15m
helloworld-deployment-3966748500-r9t6d   1/1       Terminating   0          15m
root@DevImranOps:~/kube# kubectl rollout status deployment helloworld-deployment
deployment "helloworld-deployment" successfully rolled out
root@DevImranOps:~/kube# curl 54.183.128.169:32400
Hello Everybody!root@DevImranOps:~/kube#
```

- Increasing rollout history limit from just two to many.

```
root@DevImranOps:~/kube# kubectl edit deployment helloworld-deployment
deployment "helloworld-deployment" edited
```

```
File  Edit  View  Search  Terminal  Tabs  Help
root@DevImranOps: ~/kube                    imran@DevOps: ~/.../kops
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "6"
  creationTimestamp: 2017-04-02T05:14:12Z
  generation: 6
  labels:
    app: helloworld
  name: helloworld-deployment
  namespace: default
  resourceVersion: "5429"
  selfLink: /apis/extensions/v1beta1/namespaces/default/deployments/helloworld-deployment
  uid: 35897346-1763-11e7-a421-066ce5fa5d9e
spec:
  replicas: 3
  revisionHistoryLimit: 50
  selector:
    matchLabels:
      app: helloworld
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: helloworld
    spec:
      containers:
      - image: visualpath/k8s-demo:latest
        imagePullPolicy: Always
:wq
```

```
root@DevImranOps:~/kube# kubectl rollout history deployment helloworld-deployment
deployments "helloworld-deployment"
REVISION        CHANGE-CAUSE
1               <none>
3               <none>
5               <none>
6               <none>
```

```
root@DevImranOps:~/kube# kubectl rollout undo deployment helloworld-deployment --to-revision=1
deployment "helloworld-deployment" rolled back
root@DevImranOps:~/kube# curl 54.183.128.169:32400
```

- Rollout with version number.

16. More about 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 Deployments, when updating the image version, pods are terminated and new pods take the place of older pods.

- That's 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.

- 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 pods
  - A ClusterIP: a virtual IP address only reachable from within the cluster.
  - 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.

◆ Create a helloworld pod.

```
File  Edit  View  Search  Terminal  Tabs  Help
root@DevImranOps: ~/kube                                    imran@DevOps: ~/.../kops
    Image:              visualpath/k8s-demo
    Image ID:           docker-pullable://visualpath/k8s-demo@sha256:14051338fef44e6e37b82faeaeaa15bdceff2dd4f2d7af448772d793bc795a2a
    Port:               3000/TCP
    Requests:
      cpu:              100m
    State:              Running
      Started:          Sun, 02 Apr 2017 06:03:24 +0000
    Ready:              True
    Restart Count:      0
    Volume Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-h7kw0 (ro)
    Environment Variables:      <none>
Conditions:
  Type          Status
  Initialized   True
  Ready         True
  PodScheduled  True
Volumes:
  default-token-h7kw0:
    Type:       Secret (a volume populated by a Secret)
    SecretName: default-token-h7kw0
QoS Class:      Burstable
Tolerations:    <none>
Events:
  FirstSeen     LastSeen        Count   From                                                    SubObjectPath                   Type        R
eason          Message
  ---------     --------        -----   ----                                                    -------------                   --------    -
-----          -------
  36s           36s             1       {default-scheduler }                                                                    Normal      S
cheduled       Successfully assigned nodehelloworld.example.com to ip-172-20-63-195.us-west-1.compute.internal
  35s           35s             1       {kubelet ip-172-20-63-195.us-west-1.compute.internal}   spec.containers{k8s-demo}       Normal      P
ulling         pulling image "visualpath/k8s-demo"
  34s           34s             1       {kubelet ip-172-20-63-195.us-west-1.compute.internal}   spec.containers{k8s-demo}       Normal      P
ulled          Successfully pulled image "visualpath/k8s-demo"
  34s           34s             1       {kubelet ip-172-20-63-195.us-west-1.compute.internal}   spec.containers{k8s-demo}       Normal      C
reated         Created container with docker id 111b8b3b7dda; Security:[seccomp=unconfined]
  34s           34s             1       {kubelet ip-172-20-63-195.us-west-1.compute.internal}   spec.containers{k8s-demo}       Normal      S
tarted         Started container with docker id 111b8b3b7dda
root@DevImranOps:~/kube#
```

- Create a Service for the above pod, selector value is the name of the pod as highlighted.

```
root@DevImranOps:~/kube# cat first-app/helloworld-nodeport-service.yml
apiVersion: v1
kind: Service
metadata:
  name: helloworld-service
spec:
  ports:
  - port: 31001
    nodePort: 31001
    targetPort: nodejs-port
    protocol: TCP
  selector:
    app: helloworld
  type: NodePort
root@DevImranOps:~/kube# kubectl create -f first-app/helloworld-nodeport-service.yml
service "helloworld-service" created
root@DevImranOps:~/kube# kubectl get service
NAME                CLUSTER-IP       EXTERNAL-IP   PORT(S)           AGE
helloworld-service  100.67.119.123   <nodes>       31001:31001/TCP   2m
kubernetes          100.64.0.1       <none>        443/TCP           1h
```

- Nodeport is explicitly defined in the service definition, observe in below screenshot.

```
root@DevImranOps:~/kube# cat first-app/helloworld-nodeport-service.yml
apiVersion: v1
kind: Service
metadata:
  name: helloworld-service
spec:
  ports:
  - port: 31001
    nodePort: 31001
    targetPort: nodejs-port
    protocol: TCP
  selector:
    app: helloworld
  type: NodePort
```

- We can use svc as a shortform for service.

```
root@DevImranOps:~/kube# kubectl get svc
NAME                CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
helloworld-service  100.67.119.123  <nodes>       31001:31001/TCP  5m
kubernetes          100.64.0.1      <none>        443/TCP          1h
root@DevImranOps:~/kube# kubectl describe svc helloworld-service
Name:                   helloworld-service
Namespace:              default
Labels:                 <none>
Selector:               app=helloworld
Type:                   NodePort
IP:                     100.67.119.123
Port:                   <unset> 31001/TCP
NodePort:               <unset> 31001/TCP
Endpoints:              100.96.2.21:3000
Session Affinity:       None
No events.
```

- Delete and recreate service will change the internal cluster IP/virtual IP.

```
File  Edit  View  Search  Terminal  Tabs  Help
root@DevImranOps: ~/kube                    imran@DevOps: ~/.../kops
root@DevImranOps:~/kube# kubectl describe svc helloworld-service
Name:                   helloworld-service
Namespace:              default
Labels:                 <none>
Selector:               app=helloworld
Type:                   NodePort
IP:                     100.67.119.123
Port:                   <unset> 31001/TCP
NodePort:               <unset> 31001/TCP
Endpoints:              100.96.2.21:3000
Session Affinity:       None
No events.
root@DevImranOps:~/kube# kubectl delete svc helloworld-service
service "helloworld-service" deleted
root@DevImranOps:~/kube# kubectl create -f first-app/helloworld-service.yml
service "helloworld-service" created
root@DevImranOps:~/kube# kubectl get svc
NAME                CLUSTER-IP       EXTERNAL-IP        PORT(S)        AGE
helloworld-service  100.69.176.221   ac6d9af47176f...   80:32678/TCP   7s
kubernetes          100.64.0.1       <none>             443/TCP        2h
root@DevImranOps:~/kube# kubectl describe svc helloworld-service
Name:                   helloworld-service
Namespace:              default
Labels:                 <none>
Selector:               app=helloworld
Type:                   LoadBalancer
IP:                     100.69.176.221
LoadBalancer Ingress:   ac6d9af47176f11e7a421066ce5fa5d9-897774110.us-west-1.elb.amazonaws.com
Port:                   <unset> 80/TCP
NodePort:               <unset> 32678/TCP
Endpoints:              100.96.2.21:3000
Session Affinity:       None
Events:
  FirstSeen     LastSeen        Count   From                    SubObjectPath   Type            Reason                  Message
  ---------     --------        -----   ----                    -------------   --------        ------                  -------
  15s           15s             1       {service-controller }                   Normal          CreatingLoadBalancer    Creating load balance
r
  13s           13s             1       {service-controller }                   Normal          CreatedLoadBalancer     Created load balancer
root@DevImranOps:~/kube#
```

17. Labels

- Labels are key/value pair that can be attached to objects.
  - Labels are like tags in AWS.
- You can label your objects, for instance your pod, following an organizational structure.
  - Key: environment – Value: dev/staging/qa/prod
  - Key: department – Value: engineering/finance/marketing

```
root@DevImranOps:~/kube# cat first-app/helloworld.yml
apiVersion: v1
kind: Pod
metadata:
  name: nodehelloworld.example.com
  labels:
    app: helloworld
spec:
```

Lables are not unique and multiple labels cab be added to one object. Once labels are attached to an object, you can use filters to narrow down results. This is called as Label Selectors.

Using Label Selectors, you can use matching expression to match labels. For example, a particular pod can only run on a node labelled with “environment” equals “development”.

You can use labels to tag nodes. Once nodes are tagged, you can use label selectors to let pods only run on specific nodes.

- Selecting a specific node from deployment definition.

```
root@DevImranOps:~/kube# cat deployment/helloworld-nodeselector.yml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: helloworld-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: helloworld
    spec:
      containers:
      - name: k8s-demo
        image: wardviaene/k8s-demo
        ports:
        - name: nodejs-port
          containerPort: 3000
      nodeSelector:
        hardware: high-spec
```

- Tag hardware: high-spec is not on any node from our cluster.

```
root@DevImranOps:~/kube# kubectl get nodes --show-labels
NAME                                         STATUS         AGE       LABELS
ip-172-20-44-214.us-west-1.compute.internal   Ready          2h        beta.kubernetes.io/arch=amd64,beta.kubernetes.io/instance-type=t2.micr
o,beta.kubernetes.io/os=linux,failure-domain.beta.kubernetes.io/region=us-west-1,failure-domain.beta.kubernetes.io/zone=us-west-1a,kubernetes
.io/hostname=ip-172-20-44-214.us-west-1.compute.internal
ip-172-20-58-217.us-west-1.compute.internal   Ready,master   2h        beta.kubernetes.io/arch=amd64,beta.kubernetes.io/instance-type=t2.micr
o,beta.kubernetes.io/os=linux,failure-domain.beta.kubernetes.io/region=us-west-1,failure-domain.beta.kubernetes.io/zone=us-west-1a,kubernetes
.io/hostname=ip-172-20-58-217.us-west-1.compute.internal,kubernetes.io/role=master
ip-172-20-63-195.us-west-1.compute.internal   Ready          2h        beta.kubernetes.io/arch=amd64,beta.kubernetes.io/instance-type=t2.micr
o,beta.kubernetes.io/os=linux,failure-domain.beta.kubernetes.io/region=us-west-1,failure-domain.beta.kubernetes.io/zone=us-west-1a,kubernetes
.io/hostname=ip-172-20-63-195.us-west-1.compute.internal
```

- Create our deployment which has nodeSelector tag looking for hardware: high-spec tag.

```
root@DevImranOps:~/kube# cat deployment/helloworld-nodeselector.yml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: helloworld-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: helloworld
    spec:
      containers:
      - name: k8s-demo
        image: visualpath/k8s-demo
        ports:
        - name: nodejs-port
          containerPort: 3000
      nodeSelector:
        hardware: high-spec
root@DevImranOps:~/kube# kubectl create -f deployment/helloworld-nodeselector.yml
deployment "helloworld-deployment" created
root@DevImranOps:~/kube# kubectl get pod
NAME                                     READY     STATUS    RESTARTS   AGE
helloworld-deployment-2524499403-6pb8z   0/1       Pending   0          9s
helloworld-deployment-2524499403-9c76j   0/1       Pending   0          9s
helloworld-deployment-2524499403-qz0t3   0/1       Pending   0          9s
```

```
File  Edit  View  Search  Terminal  Tabs  Help
root@DevImranOps: ~/kube                    imran@DevOps: ~/.../kops
NAME                                     READY     STATUS    RESTARTS   AGE
helloworld-deployment-2524499403-6pb8z   0/1       Pending   0          9s
helloworld-deployment-2524499403-9c76j   0/1       Pending   0          9s
helloworld-deployment-2524499403-qz0t3   0/1       Pending   0          9s
nodehelloworld.example.com               1/1       Running   0          1h
root@DevImranOps:~/kube# kubectl describe pod helloworld-deployment-2524499403-6pb8z
Name:           helloworld-deployment-2524499403-6pb8z
Namespace:      default
Node:           /
Labels:         app=helloworld
                pod-template-hash=2524499403
Status:         Pending
IP:
Controllers:    ReplicaSet/helloworld-deployment-2524499403
Containers:
  k8s-demo:
    Image:      visualpath/k8s-demo
    Port:       3000/TCP
    Requests:
      cpu:      100m
    Volume Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-h7kw0 (ro)
    Environment Variables:      <none>
Conditions:
  Type          Status
  PodScheduled  False
Volumes:
  default-token-h7kw0:
    Type:       Secret (a volume populated by a Secret)
    SecretName: default-token-h7kw0
QoS Class:      Burstable
Tolerations:    <none>
Events:
  FirstSeen     LastSeen        Count   From                    SubObjectPath   Type            Reason                  Message
  ---------     --------        -----   ----                    -------------   --------        ------                  -------
  4m            5s              19      {default-scheduler }                    Warning         FailedScheduling        pod (helloworld-deplo
yment-2524499403-6pb8z) failed to fit in any node
fit failure summary on nodes : MatchNodeSelector (3), PodToleratesNodeTaints (1)
root@DevImranOps:~/kube#
```

- ✓ Observe the pod creation status is pending.
- ✓ Deployment is looking for a node with tag high-spec, which is not found.
- ✓ If we don't have such tag then pod creation will be pending.

- Let’s Tag a node with hardware: high-spec.

```
root@DevImranOps:~/kube# kubectl get nodes
NAME                                         STATUS         AGE
ip-172-20-44-214.us-west-1.compute.internal   Ready          3h
ip-172-20-58-217.us-west-1.compute.internal   Ready,master   3h
ip-172-20-63-195.us-west-1.compute.internal   Ready          3h
root@DevImranOps:~/kube# kubectl label nodes ip-172-20-44-214.us-west-1.compute.internal hardware=high-spec
node "ip-172-20-44-214.us-west-1.compute.internal" labeled
root@DevImranOps:~/kube# kubectl get nodes --show-labels
NAME                                         STATUS         AGE       LABELS
ip-172-20-44-214.us-west-1.compute.internal   Ready          3h        beta.kubernetes.io/arch=amd64,beta.kubernetes.io/instance-type=t2.micr
o,beta.kubernetes.io/os=linux,failure-domain.beta.kubernetes.io/region=us-west-1,failure-domain.beta.kubernetes.io/zone=us-west-1a,hardware=h
igh-spec,kubernetes.io/hostname=ip-172-20-44-214.us-west-1.compute.internal
ip-172-20-58-217.us-west-1.compute.internal   Ready,master   3h        beta.kubernetes.io/arch=amd64,beta.kubernetes.io/instance-type=t2.micr
```

- Pod will automatically scheduled once the label is available.

```
root@DevImranOps:~/kube# kubectl get pods
NAME                                      READY     STATUS    RESTARTS   AGE
helloworld-deployment-2524499403-6pb8z    1/1       Running   0          11m
helloworld-deployment-2524499403-9c76j    1/1       Running   0          11m
helloworld-deployment-2524499403-qz0t3    1/1       Running   0          11m
nodehelloworld.example.com                1/1       Running   0          1h
root@DevImranOps:~/kube#
```

```
File  Edit  View  Search  Terminal  Tabs  Help
root@DevImranOps: ~/kube                                  imran@DevOps: ~/.../kops
root@DevImranOps:~/kube# kubectl describe pod helloworld-deployment-2524499403-6pb8z
Name:           helloworld-deployment-2524499403-6pb8z
Namespace:      default
Node:           ip-172-20-44-214.us-west-1.compute.internal/172.20.44.214
Start Time:     Sun, 02 Apr 2017 07:44:19 +0000
Labels:         app=helloworld
                pod-template-hash=2524499403
Status:         Running
IP:             100.96.1.13
Controllers:    ReplicaSet/helloworld-deployment-2524499403
Containers:
  k8s-demo:
    Container ID:       docker://d23cadaade95e576e2693db75c8cd82cb31444006144326a54460fc3452c144b
    Image:              visualpath/k8s-demo
    Image ID:           docker-pullable://visualpath/k8s-demo@sha256:14051338fef44e6e37b82faeaeaa15bdceff2dd4f2d7af448772d793bc795a2a
    Port:               3000/TCP
    Requests:
      cpu:              100m
    State:              Running
      Started:          Sun, 02 Apr 2017 07:44:21 +0000
    Ready:              True
    Restart Count:      0
    Volume Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-h7kw0 (ro)
    Environment Variables:      <none>
Conditions:
  Type          Status
  Initialized   True
  Ready         True
  PodScheduled  True
Volumes:
  default-token-h7kw0:
    Type:       Secret (a volume populated by a Secret)
    SecretName: default-token-h7kw0
QoS Class:      Burstable
Tolerations:    <none>
Events:
  FirstSeen     LastSeen        Count   From                    SubObjectPath   Type            Reason                  Message
  ---------     --------        -----   ----                    -------------   --------        ------                  -------
  12m           5m              29      {default-scheduler }                    Warning         FailedScheduling        pod (helloworld-deployment-2524499403-6pb8z) failed to fit in an
y node
fit failure summary on nodes : MatchNodeSelector (3), PodToleratesNodeTaints (1)
  4m    4m      1       {default-scheduler }                                    Normal  Scheduled       Successfully assigned helloworld-deployment-2524
499403-6pb8z to ip-172-20-44-214.us-west-1.compute.internal
  4m    4m      1       {kubelet ip-172-20-44-214.us-west-1.compute.internal}   spec.containers{k8s-demo}       Normal  Pulling         pulling image "visualpath/k8s-demo"
  4m    4m      1       {kubelet ip-172-20-44-214.us-west-1.compute.internal}   spec.containers{k8s-demo}       Normal  Pulled          Successfully pulled image "visualpath/k8s-demo"
  4m    4m      1       {kubelet ip-172-20-44-214.us-west-1.compute.internal}   spec.containers{k8s-demo}       Normal  Created         Created container with docker id d23cadaade95; S
ecurity:[seccomp=unconfined]
  4m    4m      1       {kubelet ip-172-20-44-214.us-west-1.compute.internal}   spec.containers{k8s-demo}       Normal  Started         Started container with docker id d23cadaade95
```

18. Secrets

Secrets provides a way in kubernetes to distribute Credentials, keys and passwords to the pods.

Kubernetes itself uses this secrets mechanism to provide the credentials to access the internal API.

Secrets can be used in 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 the volumes, you have files.

◆ Base 64 encoding, encoding username & password.

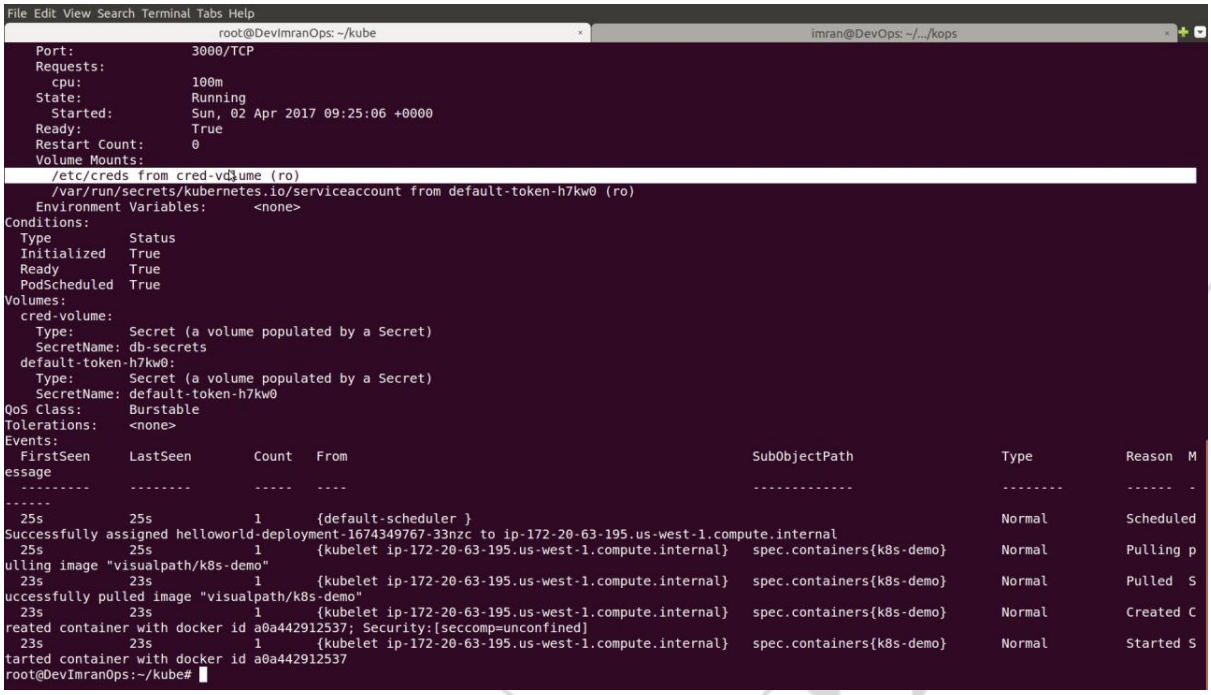
```
root@DevImranOps:~/kube# echo -n "root" | base64
cm9vdA==
root@DevImranOps:~/kube# echo -n "password" | base64
cGFzc3dvcmQ=
root@DevImranOps:~/kube#
```

◆ Creating Secret.

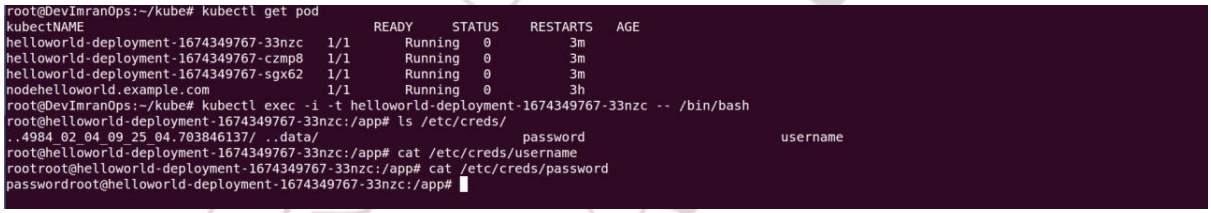
```
root@DevImranOps:~/kube# cat deployment/helloworld-secrets.yml
apiVersion: v1
kind: Secret
metadata:
  name: db-secrets
type: Opaque
data:
  username: cm9vdA==
  password: cGFzc3dvcmQ=
root@DevImranOps:~/kube# kubectl create -f deployment/helloworld-secrets.yml
secret "db-secrets" created
```

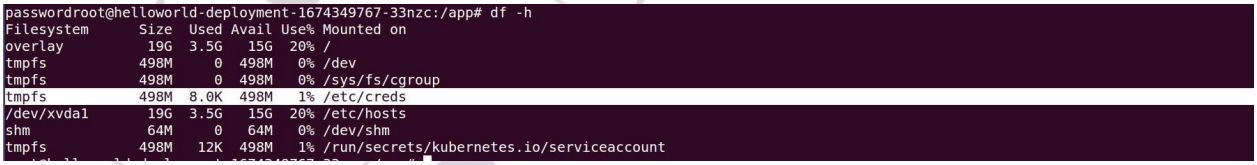
◆ Creating deployment which mounts our secret(db-secrets) as a volume.

◆ Describe shows the mounted volumes, highlighted is our db-secret mounted as volume.



- Login to any of the pod and check mount point /etc/creds to verify our db-secrets.





19. WordPress Deployment.

- Creating secrets for WordPress deployment.

```
root@DevImranOps:~/kube# cat wordpress/wordpress-secrets.yml
apiVersion: v1
kind: Secret
metadata:
  name: wordpress-secrets
type: Opaque
data:
  db-password: cGFzc3dvcmQ=
root@DevImranOps:~/kube# kubectl create -f wordpress/wordpress-secrets.yml
secret "wordpress-secrets" created
root@DevImranOps:~/kube#
```

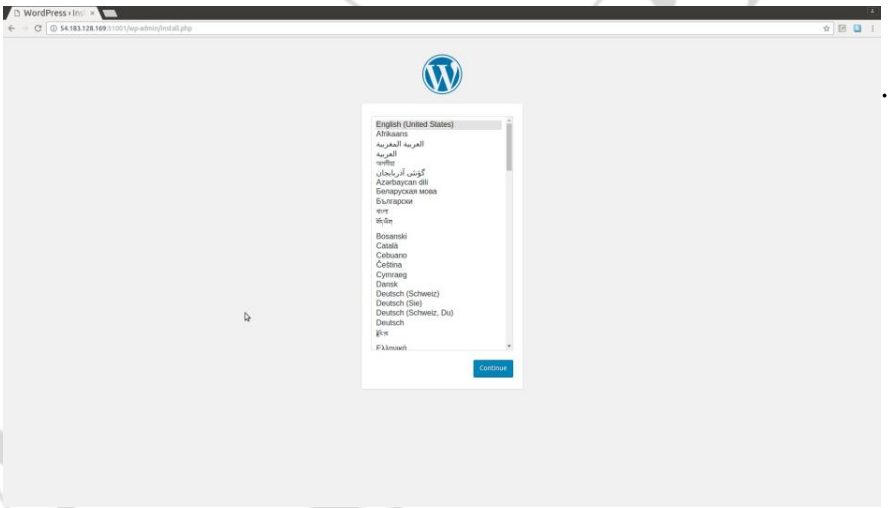
- Creating work press deployment.

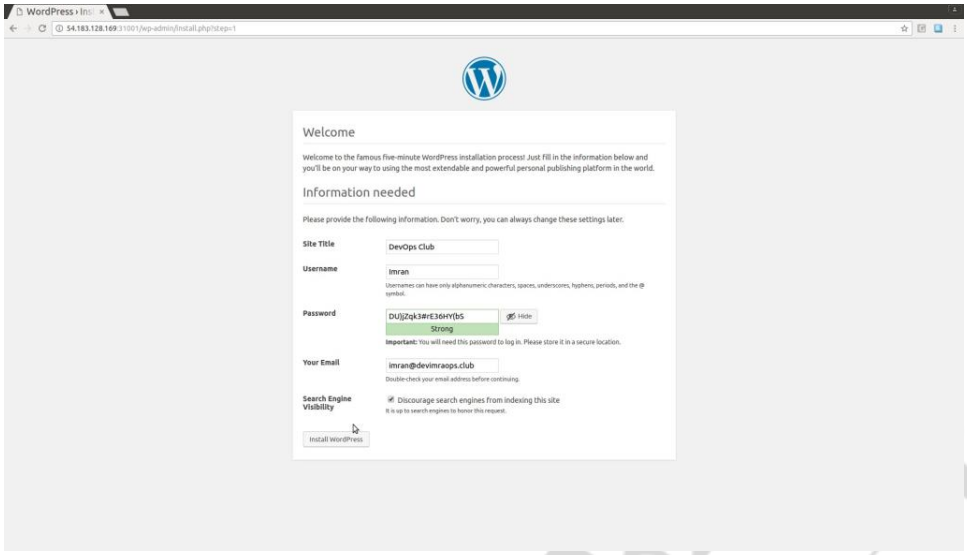
```
File  Edit  View  Search  Terminal  Tabs  Help
root@DevImranOps: ~/kube                    imran@DevOps: ~/.../kops
root@DevImranOps:~/kube# cat wordpress/wordpress-single-deployment-no-volumes.yml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: wordpress-deployment
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: wordpress
    spec:
      containers:
      - name: wordpress
        image: wordpress:4-php7.0
        ports:
        - name: http-port
          containerPort: 80
        env:
          - name: WORDPRESS_DB_PASSWORD
            valueFrom:
              secretKeyRef:
                name: wordpress-secrets
                key: db-password
          - name: WORDPRESS_DB_HOST
            value: 127.0.0.1
      - name: mysql
        image: mysql:5.7
        ports:
        - name: mysql-port
          containerPort: 3306
        env:
          - name: MYSQL_ROOT_PASSWORD
            valueFrom:
              secretKeyRef:
                name: wordpress-secrets
                key: db-password
root@DevImranOps:~/kube# kubectl create -f wordpress/wordpress-single-deployment-no-volumes.yml
deployment "wordpress-deployment" created
root@DevImranOps:~/kube#
root@DevImranOps:~/kube#
```

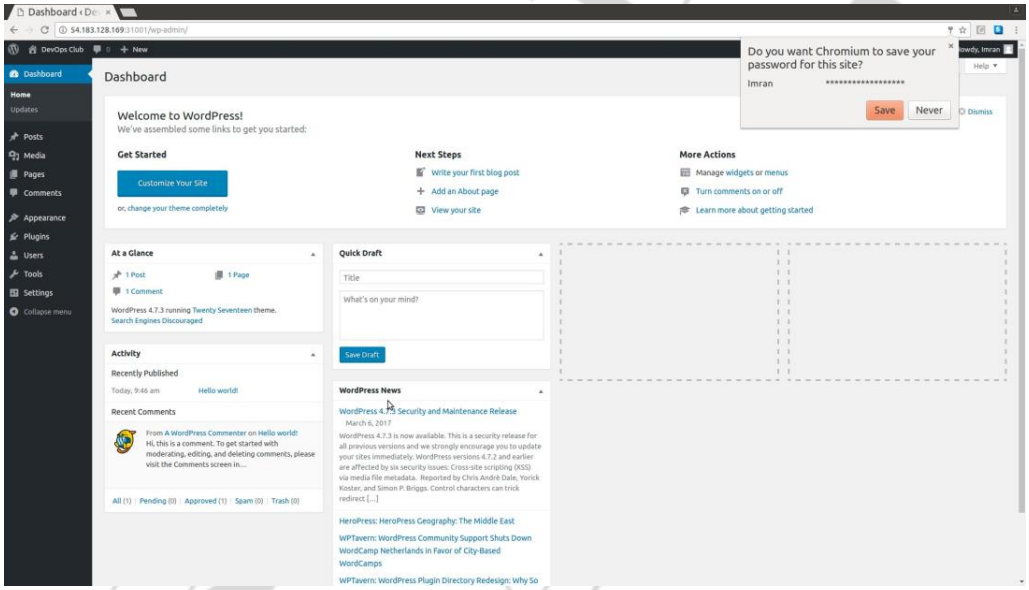
- Creating a service for WordPress.

```
root@DevImranOps:~/kube# cat wordpress/wordpress-service.yml
apiVersion: v1
kind: Service
metadata:
  name: wordpress-service
spec:
  ports:
  - port: 31001
    nodePort: 31001
    targetPort: http-port
    protocol: TCP
  selector:
    app: wordpress
  type: NodePort
root@DevImranOps:~/kube# kubectl create -f wordpress/wordpress-service.yml
service "wordpress-service" created
root@DevImranOps:~/kube# kubectl get svc
NAME                CLUSTER-IP      EXTERNAL-IP   PORT(S)           AGE
kubernetes          100.64.0.1      <none>        443/TCP           5h
wordpress-service   100.64.1.146    <nodes>       31001:31001/TCP   1m
root@DevImranOps:~/kube#
```

- Accessing your WordPress deployment from outside world.
  - Access your master node public IP on port 31001 as shown above in service port.
  - Master node Security group should allow access on port 31001 from MyIP/Anywhere.







Pods or containers are stateless, they does not save data if killed or dead.

We will delete our WordPress pod as a demo to show that behaviour.

```
root@DevImranOps:~/kube# kubectl get pod
NAME                                READY     STATUS    RESTARTS   AGE
wordpress-deployment-1969810892-z76tv   2/2       Running   0          16m
root@DevImranOps:~/kube# kubectl delete pod delete wordpress-deployment-1969810892-z76tv
pod "wordpress-deployment-1969810892-z76tv" deleted
Error from server (NotFound): pods "delete" not found
root@DevImranOps:~/kube# kubectl get pod
NAME                                READY     STATUS    RESTARTS   AGE
wordpress-deployment-1969810892-1hkkz   2/2       Running   0          7s
root@DevImranOps:~/kube#
```

- ✓ Pod get recreated by deployment replication set if we delete it manually.
- ✓ All the data from previous pod gets deleted as we delete the pod.

- ✓ Check the WordPress app from browser, we would have lost the previous installation.