KUBERNETES QUICK START GUIDE

ABSTRACT
Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications. Topics covered in this ebook would give you quick start on basic concepts. To get the most of out of this, learners should have basic proficiency with command-line tools and Linux operating system environments, as well as Web server technologies such as Nginx. Also its recommend that you have systems operations experience, including deploying and managing applications, either on-premises or in a public cloud environment.

Shanmugam Karthikeyan
Upnxtblog.com

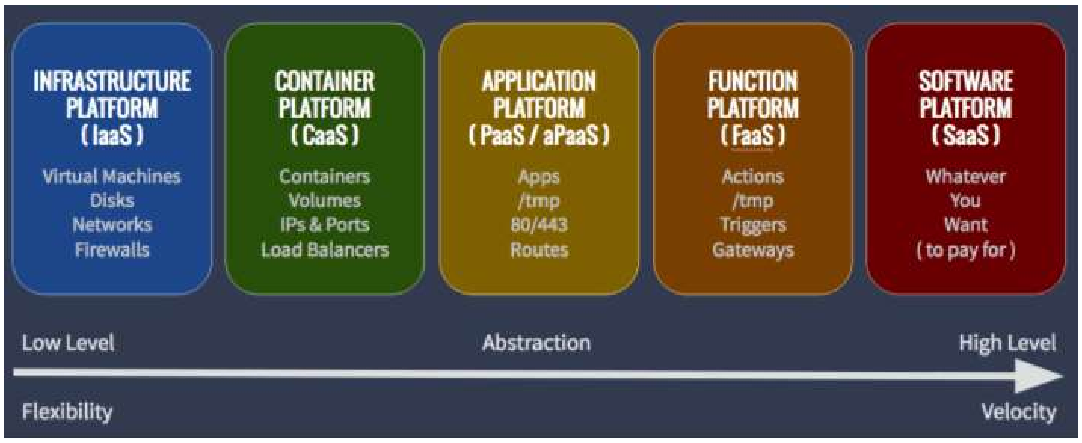
Kubernetes Quick Start Guide

www.upnxtblog.com

Contents

Introduction to Containers

Linux containers has been around since the early 2000s and architected into Linux in 2007. Due to small footprint and portability of containers, the same hardware can support an exponentially larger number of containers than VMs, dramatically reducing infrastructure costs and enabling more apps to deploy faster.But due to usability issues,it din't kickoff enough interest until Docker (2013) came into picture.To name a few prominent container platforms now – Docker,Kubernetes,Cloud Foundry *etc.,*



Cloud Platforms, their interfaces, and the scale of abstraction

Image – Mesosphere

1. Linux containers, contain applications in isolated fashion to keep them isolated from the host (OS) system that they run on.
2. Containers allow a developer to package up an application with all of the artifacts it needs, such as libraries and other dependencies, and ship it all out as one package.
3. It provide a consistent experience as developers and system administrators move code from development environments into production in a fast and replicable way.

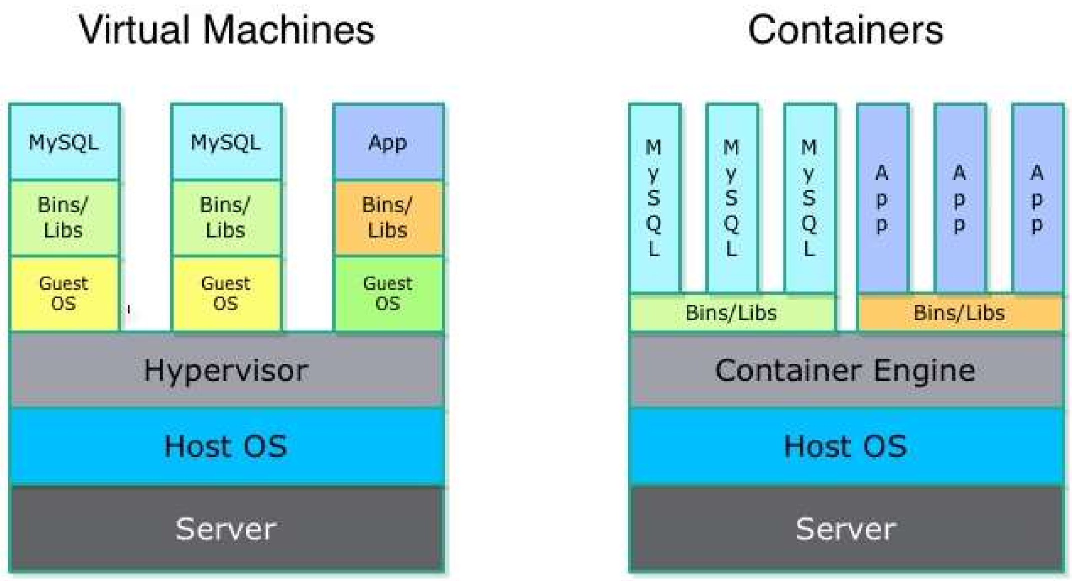
4. Containers don’t need to replicate an entire operating system, only the individual components they need in order to operate. This gives a significant performance boost and reduces the size of the application. They also operate much faster, as unlike traditional virtualization the process is essentially running natively on its host.
5. 

Image – LXC
6. Containers have also sparked an interest in *microservice architecture*, a design pattern for developing applications in which complex applications are broken down into smaller, composable services which work together. Each component is developed separately, and the application is then simply the sum of its constituent components. Each service, can live inside of a container, and can be scaled independently of the rest of the application as the need arises.
7. A bit about Docker platform : Its a utility designed to make it easier to create, deploy, and run applications by using containers. It is designed to benefit both developers and system administrators, making it a part of many DevOps (developers + operations) toolchains*(set of distinct software development tools that are linked)*. For developers, it means that they can focus on writing code without worrying about the system that it will be running on.Containers

allow a developer to package up an application with all of the artifacts it needs, such as libraries and other dependencies, and ship it all out as one package.The developer can be rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have *that could differ from the machine used for writing and testing the code*.For operations staff, Docker gives flexibility and potentially reduces the number of systems needed because of its small footprint and lower overhead.

8. Docker is different from standard virtualization,it is operating system level virtualization. Unlike hypervisor virtualization, where virtual machines run on physical hardware via an intermediation layer (hypervisor), containers instead run user space on top of an operating system's kernel. That makes them very lightweight and fast.
9. Benefits of Containers :
    1. Isolating applications and operating systems through containers.
    2. Providing nearly native performance as container manages allocation of resources in real-time.
    3. Controlling network interfaces and applying resources inside containers.
10. Limitations of containers
    1. All Containers are running inside the host system's Kernel and not with a different Kernel.
    2. Only allows Linux "guest" operating systems.
    3. Container is not a full virtualization stack like Xen, KVM, or libvirt.
    4. Security depends on the host system hence containers are not secure.
11. Risks due to containers
    1. Container breakout : If any one of the container breaks out,it can allow unauthorized access across containers, hosts or data centers etc., thus affecting all the containers hosted on the Host OS.

2. There could DDOS and cross-site scripting attacks on public facing containers hosted applications.
3. A container being forced to use up system resources in an attempt to slow or crash other containers.
4. If any of the compromised containers attempting to download additional malware, or scan internal systems for weaknesses or sensitive data,this can affect all the hosted containers.
5. Use of unsecure applications to flood the network and affect other container.

Now we know what containers are & why do we need them, also note deploying lots of containers does require sophisticated management, though. Luckily, there is a solution that simplifies this, it is Kubernetes. let's see what it has to offer.

Kubernetes Introduction

The name Kubernetes originates from Greek, it means helmsman or pilot, and is the root of governor and cybernetic. K8s is an abbreviation derived by replacing the 8 letters *"ubernete" with "8"*.Kubernetes has been built based upon 15 years of experience of running production workloads at Google, combined with best-of-breed ideas and practices from the community.It groups containers that make up an application into logical units for easy management and discovery

Kubernetes is a production-ready, open source platform designed with Google's accumulated experience in container orchestration, combined with best-of-breed ideas from the community. It is designed to automate deploying, scaling, and operating application containers.

Image – Kubernetes

Kubernetes coordinates a highly available cluster of computers that are connected to work as a single unit. The abstractions in Kubernetes allow you to deploy containerized applications to a cluster without tying them specifically to individual machines.

In short,Kubernetes is

- Portable: public, private, hybrid, multi-cloud
- Extensible: modular, pluggable, hookable, composable
- Self-healing: auto-placement, auto-restart, auto-replication, auto-scaling

Kubernetes Architecture – System & Abstractions

Following would help you to learn about the different parts of the Kubernetes system and the abstractions. Kubernetes automates the entire distribution and scheduling of application containers across a cluster in a more efficient way.

1. To interact with Kubernetes, there is an API layer (*Kubernetes API)* exposed same can be interacted using command-line interface via `kubectl`
2. Any Kubernetes cluster (example below) would have two types of resources:
    1. Master which controls the cluster
    2. Node are the workers nodes that runs applications

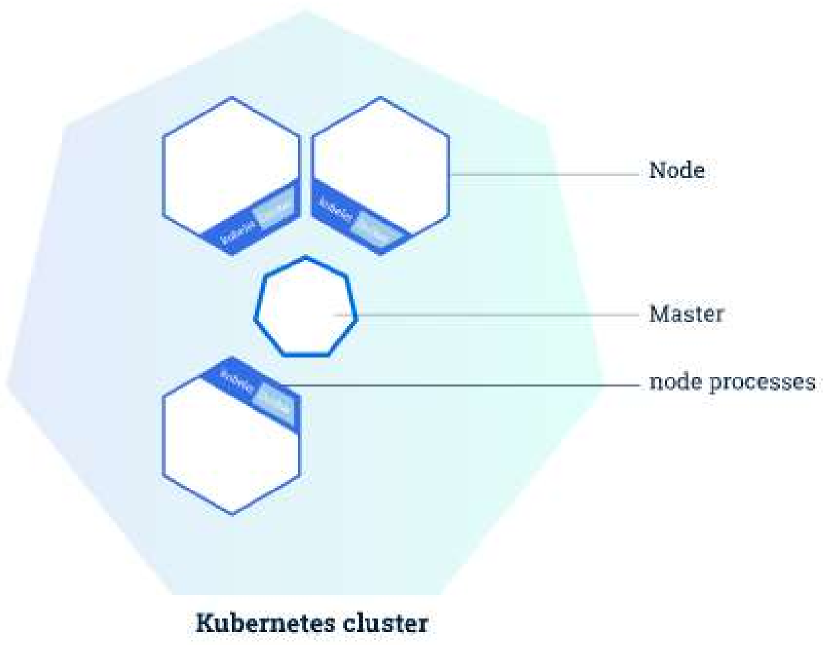


Image – Kubernetes cluster

3. The **Master** coordinates all activities in your cluster, such as scheduling applications, maintaining applications' desired state, scaling applications, and rolling out new updates.
4. Each **Mode** can be a VM or a physical computer that serves as a worker machine in a cluster.Each node has a **Kubelet**, which is an agent for managing the node and communicating with the Kubernetes master. The node should also have tools for handling container operations, such as Docker or rkt.
5. When any applications needs to be deployed on Kubernetes, **master** issues command to start the application containers. The master schedules the containers to run on the cluster's **nodes**.
6. The **nodes** communicate with the master using the Kubernetes API, which the master exposes. End users can also use the Kubernetes API directly to interact with the cluster.

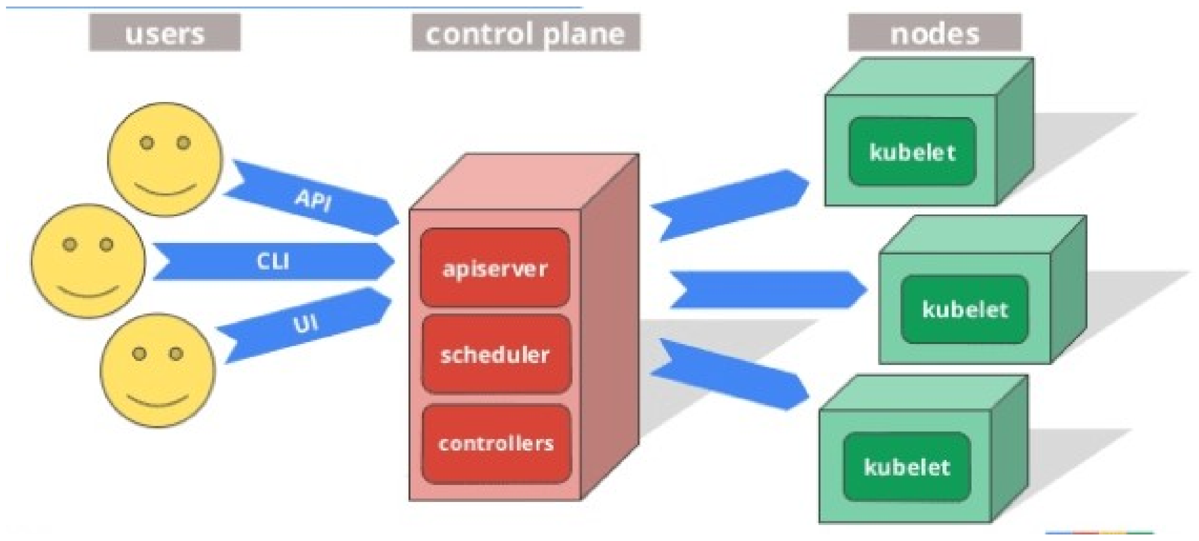


Image – Kubernetes Abstractions

Master components provide the cluster's control plane. Kubernetes Control Planeconsists of a collection of below processes on your cluster:

- Kubernetes Master collection of three processes kube-apiserver, kube-controller-manager and kube-scheduler.
  - kube-apiserver exposes the Kubernetes API. It is the front-end for the Kubernetes control plane.
  - kube-controller-manager runs controllers, which are the designed to handle routine tasks in the cluster.
- Each individual non-master node on the cluster runs two processes:
  - kubelet – this is to communicate with Kubernetes Master
  - kube-proxy – this is nothing but network proxy (Kubernetes networking services) on each node.
  - kube-scheduler is to keep watch for newly created pods that have no node assigned, and selects a node for them to run on.

Master components make global decisions about the cluster (like for example, scheduling applications), and detecting and responding to cluster events.

Apart from the above,there are other objects to represent the state of system,some of the basic Kubernetes objects include:

- Pod
- Service
- Volume
- Namespace
- Controllers

Kubernetes cluster can run on various platforms: from your laptop, to VMs on a cloud provider, to a rack of bare metal servers. To try with local Kubernetes setup, you can use Minikube. Minikube is a lightweight Kubernetes implementation that creates a VM on your local machine and deploys a simple cluster containing only one node.Minikube is available for Linux, macOS, and Windows systems. The Minikube CLI provides basic bootstrapping operations for working with your cluster, including start, stop, status, and delete.

There is also web-based Dashboard for Kubernetes clusters. It allows users to manage and troubleshoot applications running in the cluster, as well as the cluster itself.

Create new Kubernetes cluster

We have looked at the Introduction & key concepts of Kubernetes platform.Now in this chapter,we are going to Create new Kubernetes cluster using Minikube. Minikube is a lightweight Kubernetes implementation that creates a VM on your local machine and deploys a simple cluster containing only one node. Minikube is available for Linux, macOS, and Windows systems.

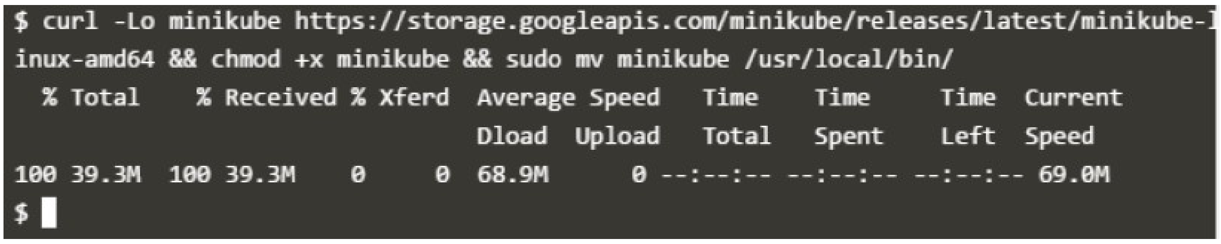
The Minikube CLI provides basic bootstrapping operations for working with your cluster, including start, stop, status, and delete.Also we are going to use kubectl utility to deploy and manage applications on Kubernetes. Also,you can inspect cluster resources; create, delete, and update components; and look at new cluster.

Here are the detailed steps:

Step #1. Minikube installation

Download the latest release with the command.This is for linux,if you're using other OS,please refer above link.

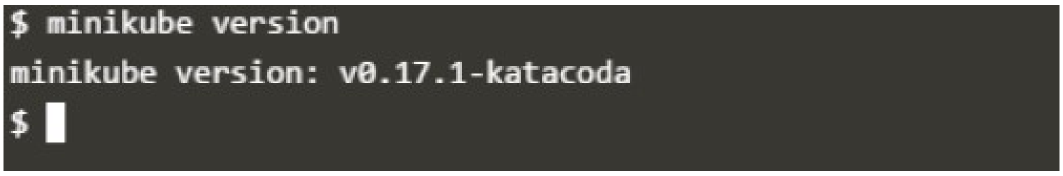
```
curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/
```



```
$ curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-l
inux-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 39.3M  100 39.3M    0     0  68.9M      0 --:--:-- --:--:-- --:--:-- 69.0M
$ █
```

Image – Minikube installation

Check the download by running the *minikube version* command:



```
$ minikube version
minikube version: v0.17.1-katacoda
$
```

Image – Check Minikube version

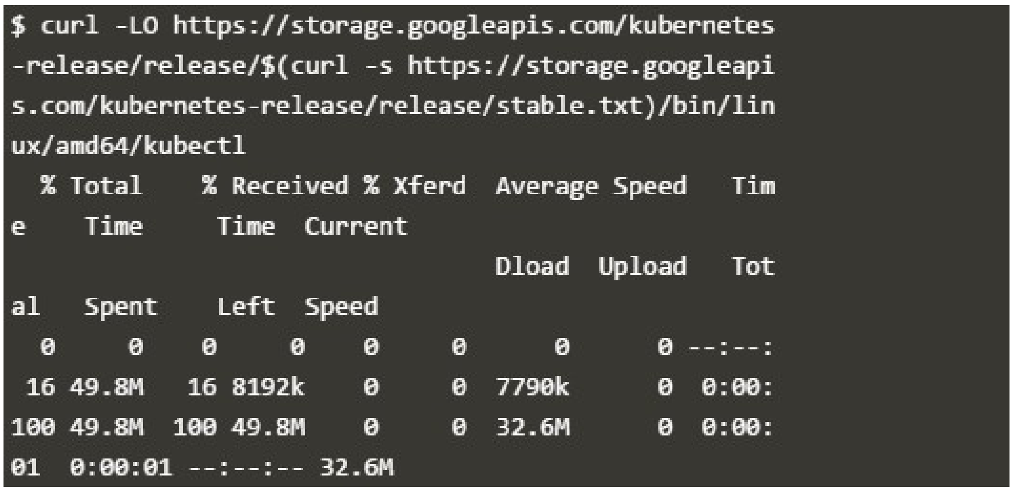
To check what are the available commands,try *minikube* from the terminal

```
$ minikube
Minikube is a CLI tool that provisions and manages single-node Kubernetes clusters optimized
 for development workflows.

Usage:
 minikube [command]

Available Commands:
start Starts a local kubernetes cluster.
stop Stops a running local kubernetes cluster.
version Print the version of minikube.
```

Step #2.kubectl installation.

Download the latest release with the command:

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl
```



```
$ curl -LO https://storage.googleapis.com/kubernetes
-release/release/$(curl -s https://storage.googleapi
s.com/kubernetes-release/release/stable.txt)/bin/lin
ux/amd64/kubectl
  % Total    % Received % Xferd  Average Speed   Tim
e    Time     Time  Current
                                 Dload  Upload   Tot
al   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:
 16 49.8M   16 8192k    0     0  7790k      0  0:00:
100 49.8M  100 49.8M    0     0  32.6M      0  0:00:
01  0:00:01 --:--:-- 32.6M
```

Image – kubectl Installation

Make the kubectl binary executable.

```
chmod +x ./kubectl
```

Move the binary in to your PATH.

```
sudo mv ./kubectl /usr/local/bin/kubectl
```

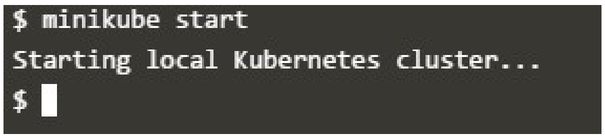
To check what are the available *kubectl* commands,run *kubectl* from the terminal

```
$ kubectl
kubectl controls the Kubernetes cluster manager.

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

Basic Commands (Beginner):
  create        Create a resource from a file or from 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
  run-container  Run a particular image on the cluster. This command is
deprecated, use "run" instead
```

```
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 the rollout of a resource
  rolling-update Perform a rolling update of the given ReplicationController
  scale         Set a new size for a Deployment, ReplicaSet, Replication
Controller, or Job
  autoscale      Auto-scale a Deployment, ReplicaSet, or ReplicationController

Cluster Management Commands:
  certificate    Modify certificate resources.
  cluster-info   Display cluster info
  top          Display Resource (CPU/Memory/Storage) usage.
  cordon        Mark node as unschedulable
  uncordon      Mark node as schedulable
  drain         Drain node in preparation for maintenance
  taint         Update the taints on one or more nodes

Troubleshooting and Debugging Commands:
  describe      Show details of a specific resource or group of resources
  logs          Print the logs for a container in a pod
  attach        Attach to a running container
  exec          Execute a command in a container
  port-forward   Forward one or more local ports to a pod
  proxy         Run a proxy to the Kubernetes API server
  cp           Copy files and directories to and from containers.
  auth          Inspect authorization

Advanced Commands:
  apply         Apply a configuration to a resource by filename or stdin
  patch         Update field(s) of a resource using strategic merge patch
  replace       Replace a resource by filename or stdin
  convert       Convert config files between different API versions

Settings Commands:
  label         Update the labels on a resource
  annotate      Update the annotations on a resource
  completion    Output shell completion code for the specified shell (bash or
zsh)

Other Commands:
  api-versions  Print the supported API versions on the server, in the form of
"group/version"
  config        Modify kubeconfig files
  help          Help about any command
  plugin        Runs a command-line plugin
  version       Print the client and server version information

Use "kubectl  --help" for more information about a given command.
Use "kubectl options" for a list of global command-line options (applies to all
commands).
```

Step #3. Create local cluster

Start the cluster, by running the *minikube start* command:

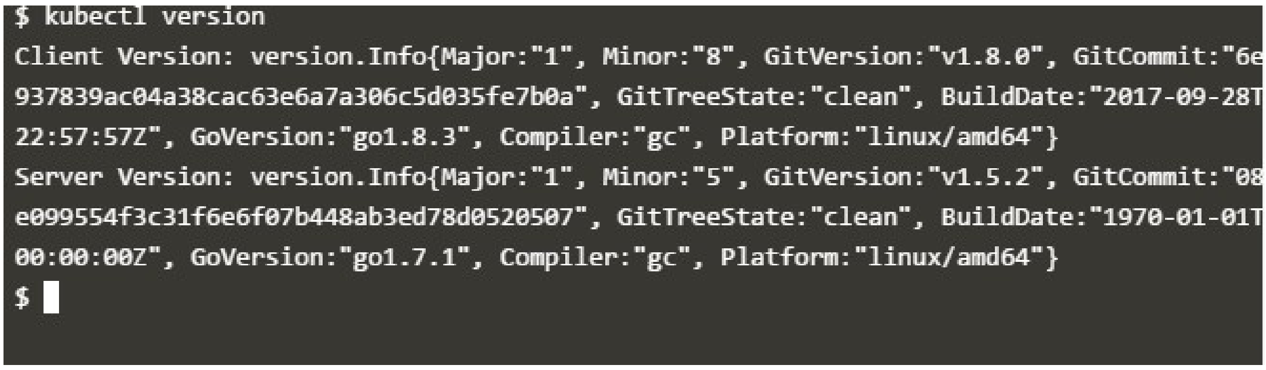


```
$ minikube start
Starting local Kubernetes cluster...
$
```

Image – Starting local cluster

Once the cluster is started,we have a running Kubernetes local cluster now.Minikube has started a virtual machine for you, and a Kubernetes cluster is now running in that VM.

To interact with Kubernetes we'll use the command line interface, *kubectl*. To check if *kubectl* is installed you can run the *kubectl version* command:

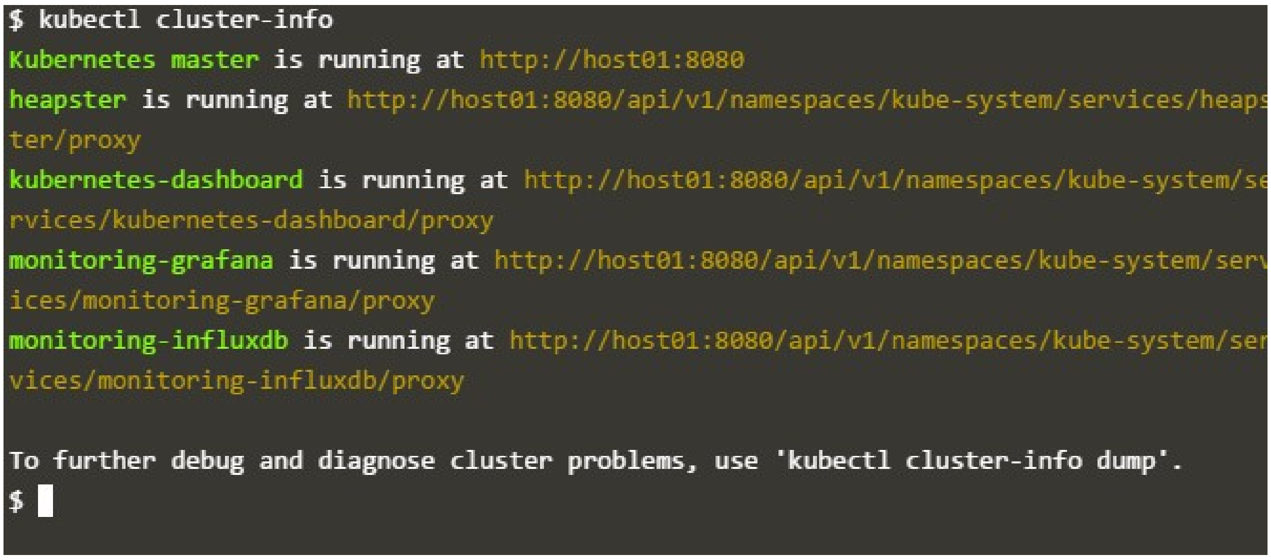


```
$ kubectl version
Client Version: version.Info{Major:"1", Minor:"8", GitVersion:"v1.8.0", GitCommit:"6e
937839ac04a38cac63e6a7a306c5d035fe7b0a", GitTreeState:"clean", BuildDate:"2017-09-28T
22:57:57Z", GoVersion:"go1.8.3", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"5", GitVersion:"v1.5.2", GitCommit:"08
e099554f3c31f6e6f07b448ab3ed78d0520507", GitTreeState:"clean", BuildDate:"1970-01-01T
00:00:00Z", GoVersion:"go1.7.1", Compiler:"gc", Platform:"linux/amd64"}
$
```

Image – kubectl version command

As you can see kubectl is configured and we can see that both the version of the client and as well as the server. The client version is the kubectl version; the server version is the Kubernetes version installed on the master. You can also see details about the build.

To view the cluster details, run *kubectl cluster-info*:



```
$ kubectl cluster-info
Kubernetes master is running at http://host01:8080
heapster is running at http://host01:8080/api/v1/namespaces/kube-system/services/heaps
ter/proxy
kubernetes-dashboard is running at http://host01:8080/api/v1/namespaces/kube-system/se
rvices/kubernetes-dashboard/proxy
monitoring-grafana is running at http://host01:8080/api/v1/namespaces/kube-system/serv
ices/monitoring-grafana/proxy
monitoring-influxdb is running at http://host01:8080/api/v1/namespaces/kube-system/ser
vices/monitoring-influxdb/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
$
```

Image – kubectl cluster-info

To view the nodes in the cluster, run the *kubectl get nodes* command:

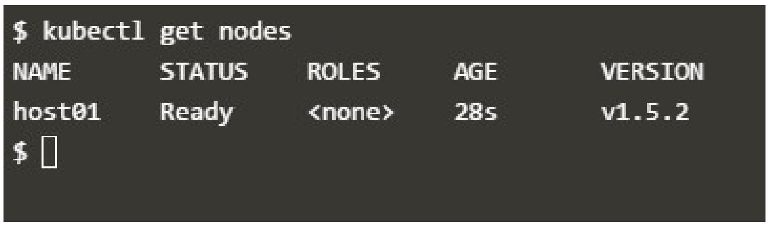

```
$ kubectl get nodes
NAME      STATUS    ROLES     AGE       VERSION
host01    Ready     <none>    28s       v1.5.2
$ 
```

Image – kubectl get nodes command

Step #4 : Deploy ngnix app to one of the nodes of the cluster

Let’s run our first app on Kubernetes with the *kubectl run* command. The run command creates a new deployment. We need to provide the deployment name and app image location *(include the full repository url for images hosted outside Docker hub)* ,currently I have provided ngnix image. If we want to run the app on a specific port so we could add the *–port* parameter as well.

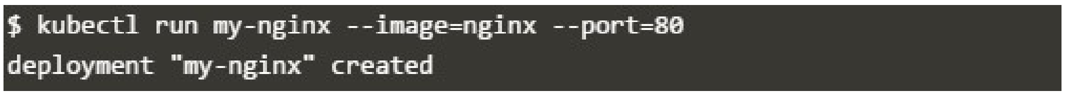
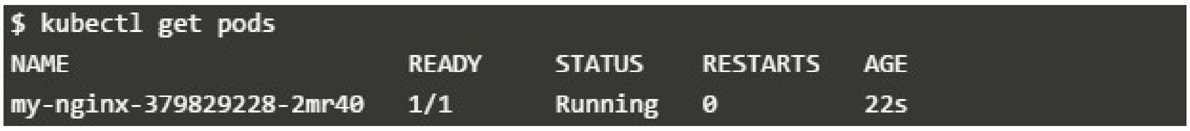

```
$ kubectl run my-nginx --image=nginx --port=80
deployment "my-nginx" created
```

Image – Kubernets deployment created

Congrats! We have just deployed first application by creating a deployment. Following is what the command has done for us:

1. Searched for a suitable node where an instance of the application could be run *(we have only 1 available node)*
2. Scheduled the ngnix application to run on that Node
3. Configured the cluster to reschedule the instance on a new Node when needed



```
$ kubectl get pods
NAME                     READY     STATUS    RESTARTS   AGE
my-nginx-379829228-2mr40 1/1       Running   0          22s
```

Image – kubectl get pods

Once the application instances are created, a Kubernetes Deployment Controller continuously monitors those instances. If the Node hosting an instance goes down or is deleted, the Deployment controller replaces it.

A Pod is a Kubernetes abstraction that represents a group of one or more application containers (such as Docker or rkt), and some shared resources for those containers. Those resources include:

- Shared storage, as Volumes
- Networking, as a unique cluster IP address
- Information about how to run each container, such as the container image version or specific ports to use

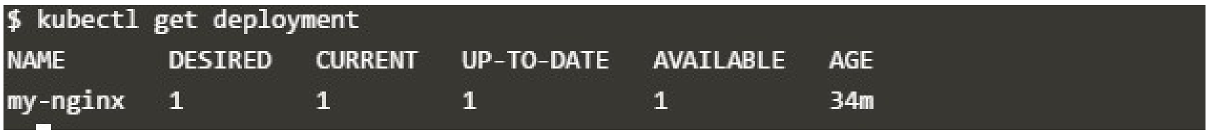
A Pod always runs on a Node. As discussed earlier,Node is a nothing but a worker machine in Kubernetes and may be either a virtual or a physical machine, depending on the cluster. Each Node is managed by the Master. A Node can have multiple pods, and the Kubernetes master automatically handles scheduling the pods across the Nodes in the cluster. The Master's automatic scheduling takes into account the available resources on each Node.

Every Kubernetes Node runs at least:

- Kubelet – responsible for communication between the Kubernetes Master and the Nodes
- Container runtime (like Docker, rkt)

For example, a Pod might include both the container with your ngnix app as well as a different container that feeds the data to be published by the ngnix webserver. The containers in a Pod share an IP Address and port space, are always co-located and co-scheduled, and run in a shared context on the same Node.

To list your deployments, use the *get deployments* command:



```
$ kubectl get deployment
NAME       DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
my-nginx   1         1         1            1           34m
```

Image – kubectl get deployments command

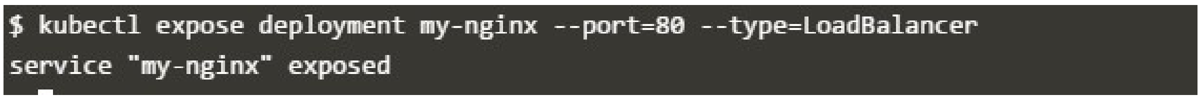
Here we can see that there is 1 deployment running a single instance of the app.

Some of the useful kubectl commands are below.

- kubectl get – *list resources*
- kubectl describe – *show detailed information about a resource*
- kubectl logs – *print the logs from a container in a pod*
- kubectl exec – *execute a command on a container in a pod*

Step #5 : Expose ngnix app outside of the cluster

To expose the app on to the outside world, use *expose deployment* command:



```
$ kubectl expose deployment my-nginx --port=80 --type=LoadBalancer
service "my-nginx" exposed
```

Image – kubectl expose deployment command

Pods that are running inside Kubernetes are running on a private, isolated network. By default they are visible from other pods and services within the same kubernetes cluster, but not outside that network. On some platforms (for example Google Compute Engine) the kubectl command can integrate with your cloud provider to add a public IP address for the pods, to do this run:

To see the ngnix landing page,you can check at the http://localhost:80

Also note in order to access your nginx landing page, you also have to make sure that traffic from external IPs is allowed. Do this by opening a firewall to allow traffic on port 80.

```
kubectl get services
```

This should print the service that has been created, and map an external IP address to the service. Where to find this external IP address will depend on the environment you run in. For instance, for Google Compute Engine the external IP address is listed as part of the newly created service and can be retrieved by running above command.

A Service in Kubernetes is an abstraction which defines a logical set of Pods and a policy by which to access them. Services enable a loose coupling between dependent Pods.A Service routes traffic across a set of Pods. Services are the abstraction that allow pods to die and replicate in Kubernetes without impacting your application. Discovery and routing among dependent Pods (such as the frontend and backend components in an application) is handled by Kubernetes Services.

Step #6 : Delete app

To delete the app,run *delete deployment* command

```
kubectl delete deployment my-nginx
```

In the next tutorial,we will learn how to scale & perform updates to the app on the cluster.

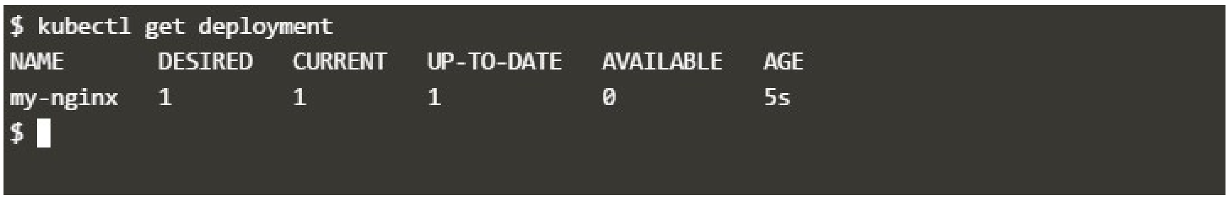
Scale & perform updates to the app on the cluster

We have looked at how to create local cluster,deploy an app and check the status of the deployments.In continuation to the series,in this post we are going to check how to scale & perform updates to applications running on Kubernetes cluster.

Step #1. Check the list of application deployment

If you can remember, in the last post we have deployed our nginx application using *run*command.So lets check the list of application deployments using *get deployments*command.

Run command would have created only one Pod for running our application. But in the real life scenario,when traffic increases, we will need to scale the application to keep up with user demand. Running multiple instances of an application will require a way to distribute the traffic to all of them. Services have an integrated load-balancer that will distribute network traffic to all Pods of an exposed Deployment. Services will monitor continuously the running Pods using endpoints, to ensure the traffic is sent only to available Pods.



```
$ kubectl get deployment
NAME       DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
my-nginx   1         1         1            0           5s
$ █
```

Image – kubectl get deployment command

To list your deployments use the *get deployments* command:

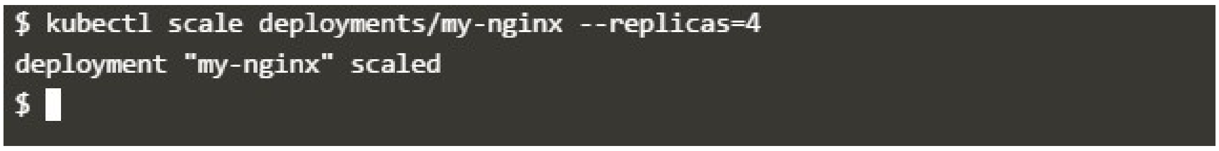
We should have 1 Pod. If not, run the command again. This shows:

- The DESIRED state is showing the configured number of replicas

- The CURRENT state show how many replicas are running now
- The UP-TO-DATE is the number of replicas that were updated to match the desired (configured) state
- The AVAILABLE state shows how many replicas are actually AVAILABLE to the users

Step #2. Scale up/down application deployment

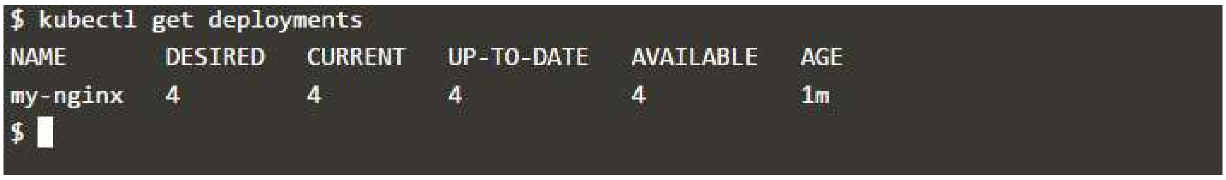
Now let's scale the Deployment to *4 replicas*. We are going to use the *kubectl scale*command, followed by the deployment type, name and desired number of instances:



```
$ kubectl scale deployments/my-nginx --replicas=4
deployment "my-nginx" scaled
$
```

Image – kubectl scale deployment command

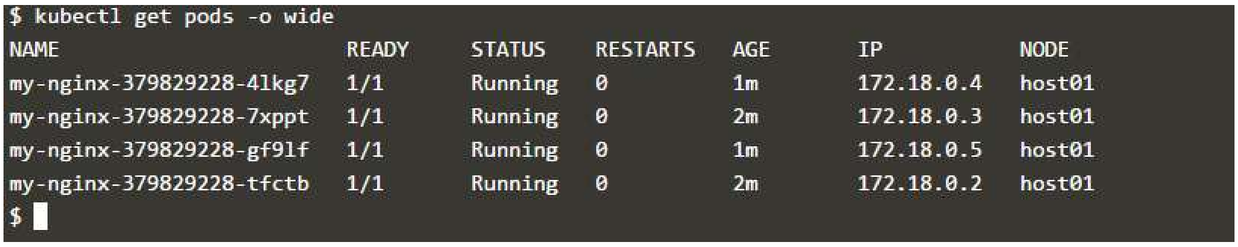
The change was applied, and we have 4 instances of the application available. Next, let's check if the number of Pods changed:



```
$ kubectl get deployments
NAME       DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
my-nginx   4         4         4            4           1m
$
```

Image – kubectl get deployments command

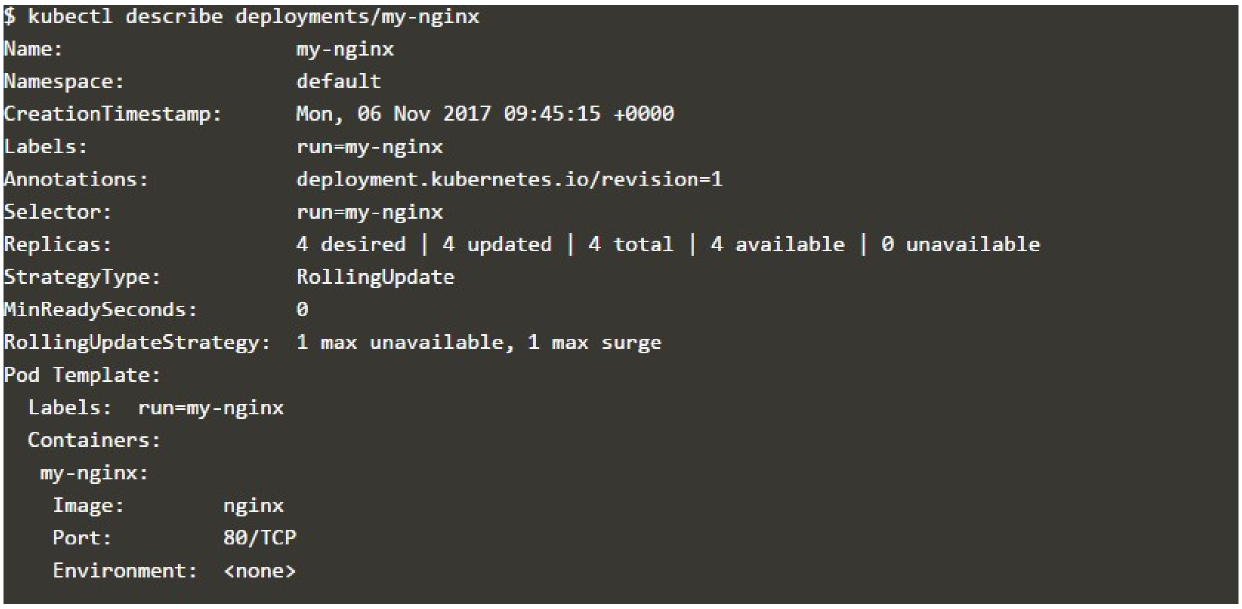
Now There should be 4 pods running in the cluster



```
$ kubectl get pods -o wide
NAME                         READY     STATUS    RESTARTS   AGE       IP           NODE
my-nginx-379829228-4lkg7     1/1       Running   0          1m        172.18.0.4   host01
my-nginx-379829228-7xppt     1/1       Running   0          2m        172.18.0.3   host01
my-nginx-379829228-gf9lf     1/1       Running   0          1m        172.18.0.5   host01
my-nginx-379829228-tfctb     1/1       Running   0          2m        172.18.0.2   host01
$
```

Image – kubectl get pods command

There are 4 Pods now, with different IP addresses. The change was registered in the Deployment events log. To check that, use the *describe* command:

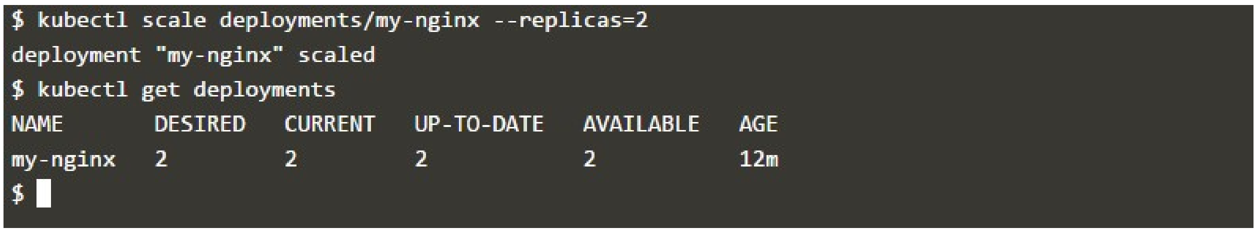


```
$ kubectl describe deployments/my-nginx
Name:                   my-nginx
Namespace:              default
CreationTimestamp:      Mon, 06 Nov 2017 09:45:15 +0000
Labels:                 run=my-nginx
Annotations:            deployment.kubernetes.io/revision=1
Selector:               run=my-nginx
Replicas:               4 desired | 4 updated | 4 total | 4 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  1 max unavailable, 1 max surge
Pod Template:
  Labels:  run=my-nginx
  Containers:
   my-nginx:
    Image:        nginx
    Port:         80/TCP
    Environment:  <none>
```

Image – kubectl describe command

You can also view in the output of this command that there are 4 replicas now.

To scale down the Service to 2 replicas, run again the *scale* command:



```
$ kubectl scale deployments/my-nginx --replicas=2
deployment "my-nginx" scaled
$ kubectl get deployments
NAME       DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
my-nginx   2         2         2            2           12m
$ █
```

Image – kubectl scale command

Step #3. Perform rolling updates to application deployment

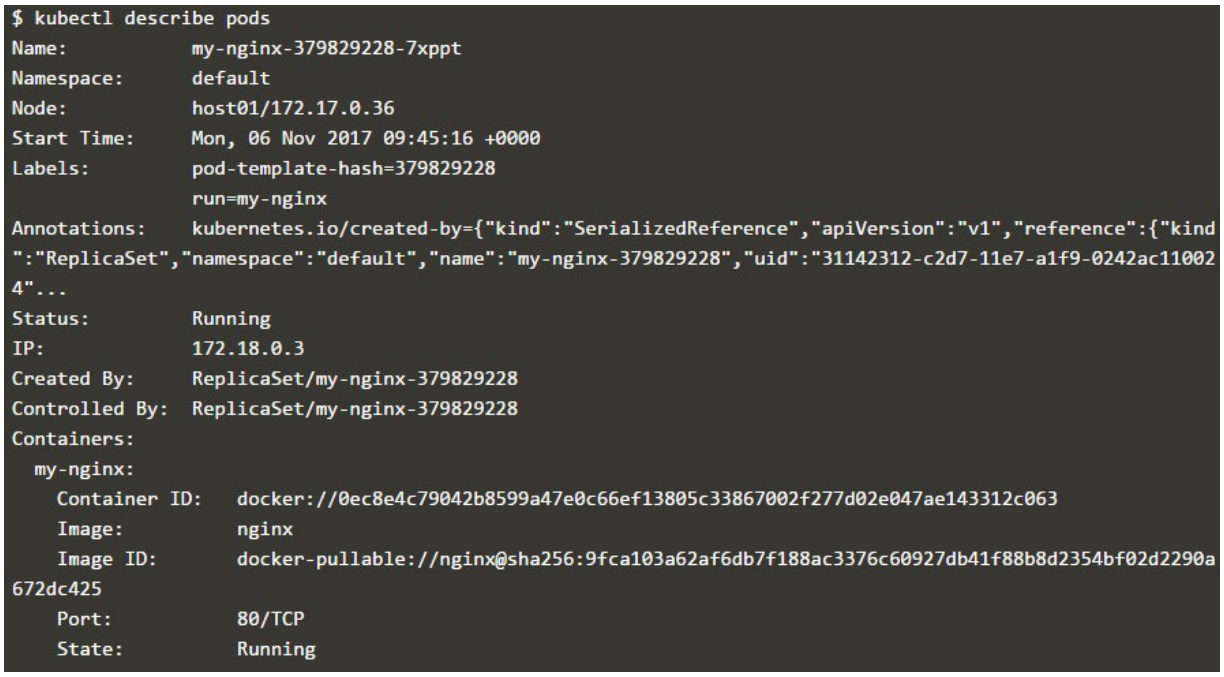
If you have multiple instances of an Application running, there could be scenarios where old instances can clash with the new instances and if you shutdown the cluster for updates,downtime could never be not acceptable.Users expect applications to be available all the time and developers are expected to deploy new versions of them several times a day.

In Kubernetes this is done with rolling updates. Rolling updates allow Deployments update to take place with zero downtime by incrementally updating Pods instances with new ones. The new Pods will be scheduled on Nodes with available resources.

Rolling updates allow the following actions:

- Promote an application from one environment to another (via container image updates)
- Rollback to previous versions
- Continuous Integration and Continuous Delivery of applications with zero downtime

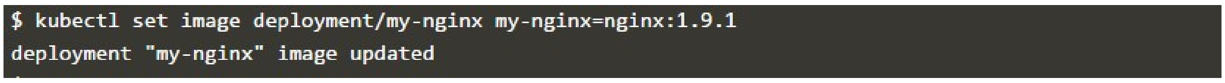
To view the current image version of the app, run a *describe* command against the Pods (look at the Image field):



```
$ kubectl describe pods
Name:           my-nginx-379829228-7xppt
Namespace:      default
Node:           host01/172.17.0.36
Start Time:     Mon, 06 Nov 2017 09:45:16 +0000
Labels:         pod-template-hash=379829228
                run=my-nginx
Annotations:    kubernetes.io/created-by={"kind":"SerializedReference","apiVersion":"v1","reference":{"kind
":"ReplicaSet","namespace":"default","name":"my-nginx-379829228","uid":"31142312-c2d7-11e7-a1f9-0242ac11002
4"...
Status:         Running
IP:             172.18.0.3
Created By:     ReplicaSet/my-nginx-379829228
Controlled By:  ReplicaSet/my-nginx-379829228
Containers:
  my-nginx:
    Container ID:   docker://0ec8e4c79042b8599a47e0c66ef13805c33867002f277d02e047ae143312c063
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:9fca103a62af6db7f188ac3376c60927db41f88b8d2354bf02d2290a
672dc425
    Port:           80/TCP
    State:          Running
```

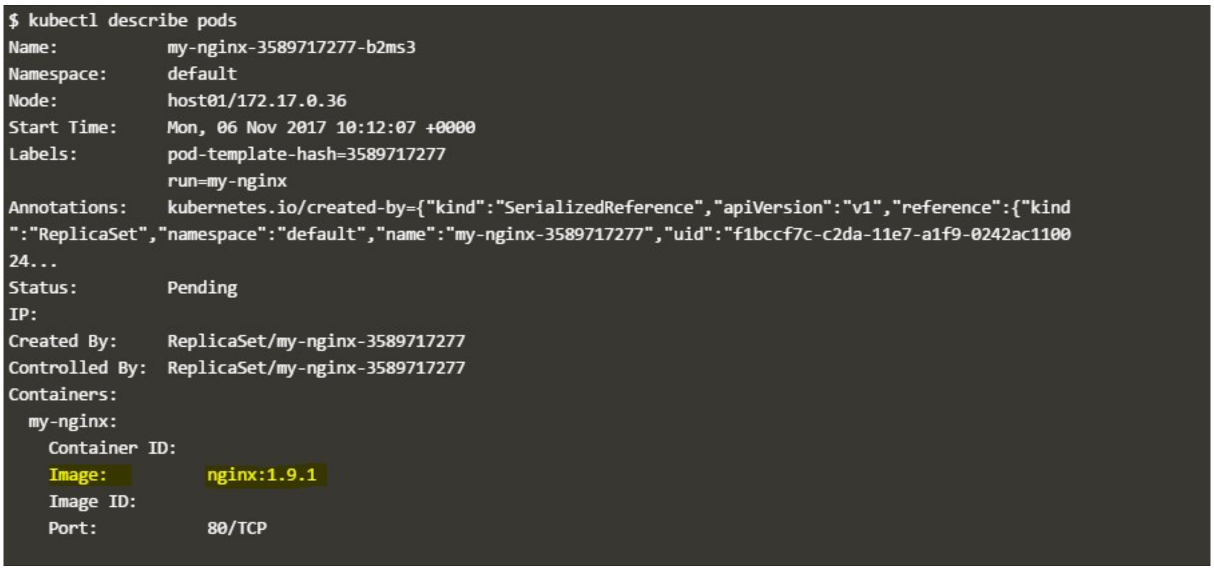
Image – kubectl describe command

To update the image of the application to new version, use the *set image* command, followed by the deployment name and the new image version:



```
$ kubectl set image deployment/my-nginx my-nginx=nginx:1.9.1
deployment "my-nginx" image updated
```

Image – kubectl set image command



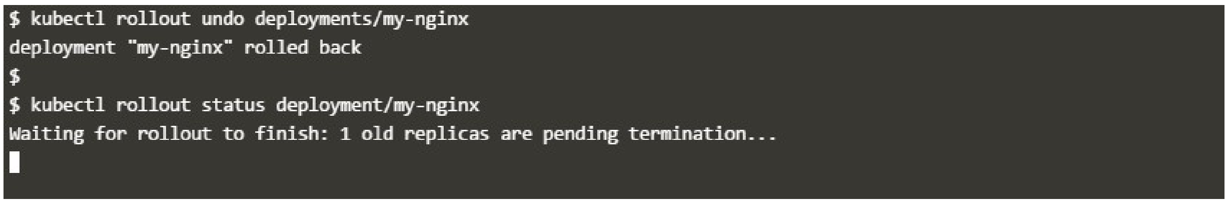
```
$ kubectl describe pods
Name:           my-nginx-3589717277-b2ms3
Namespace:      default
Node:           host01/172.17.0.36
Start Time:     Mon, 06 Nov 2017 10:12:07 +0000
Labels:         pod-template-hash=3589717277
                run=my-nginx
Annotations:    kubernetes.io/created-by={"kind":"SerializedReference","apiVersion":"v1","reference":{"kind
":"ReplicaSet","namespace":"default","name":"my-nginx-3589717277","uid":"f1bccf7c-c2da-11e7-a1f9-0242ac1100
24...
Status:         Pending
IP:
Created By:     ReplicaSet/my-nginx-3589717277
Controlled By:  ReplicaSet/my-nginx-3589717277
Containers:
  my-nginx:
    Container ID:
    Image:          nginx:1.9.1
    Image ID:
    Port:           80/TCP
```

Image – kubectl describe pods command

The command notified the Deployment to use a different image for your app and initiated a rolling update. Check the status of the new Pods, and view the old one terminating with the get pods command:

Step #4. Rollback updates to application deployment

Suppose if you want to roll out the updates we made, We'll use the *rollout undo*command:



```
$ kubectl rollout undo deployments/my-nginx
deployment "my-nginx" rolled back
$
$ kubectl rollout status deployment/my-nginx
Waiting for rollout to finish: 1 old replicas are pending termination...
█
```

Image – kubectl rollout undo command

The rollout command reverted the deployment to the previous known state. Updates are versioned and you can revert to any previously know state of a Deployment. List again the Pods:

After the rollout succeeds, you may want to *get* the Deployment.

Step #5. Cleanup

Finally you can clean up the resources you created in your cluster:

```
kubectl delete service my-nginx
kubectl delete deployment my-nginx
```

Create application deployment using yaml file

In the last post, we have learnt how to create & deploy the app to the Kubernetes cluster.Now in this post,we are going to learn how to create application deployment using yaml file also we can check on how to create services to control how application communicates.

Step #1.Create an nginx deployment

Using *Deployment* controller we can provide declarative updates for Pods and ReplicaSets. Create deployment.yaml file in your current folder like the below to describe the nginx deployment.

Kubernetes manifest file defines a desired state for the cluster, including what container images should be running.For example, this YAML file describes a Deployment that runs the nginx:latest Docker image

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

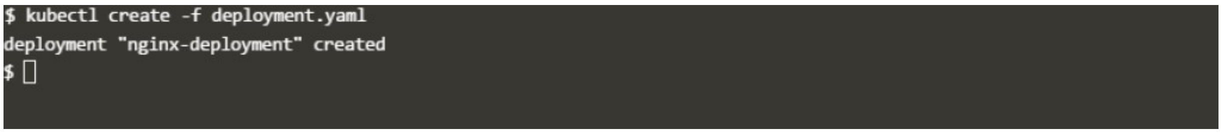
In this example:

- A Deployment named *nginx-deployment* is created, indicated by the metadata: name field.
- The Deployment creates *1 replicated Pods*, indicated by the replicas field.
- The Pod template's specification, or template: spec field, indicates that the Pods run one container, nginx, which runs the nginx Docker Hub latest image (you can also specify the version ex.1.7.9.)
- The Deployment opens port 80 for use by the Pods.

Step #2.Create Deployment based on the YAML file

- Based on the deployment described in deployment.yaml created in the previous step,create the deployment using *kubectl create* command

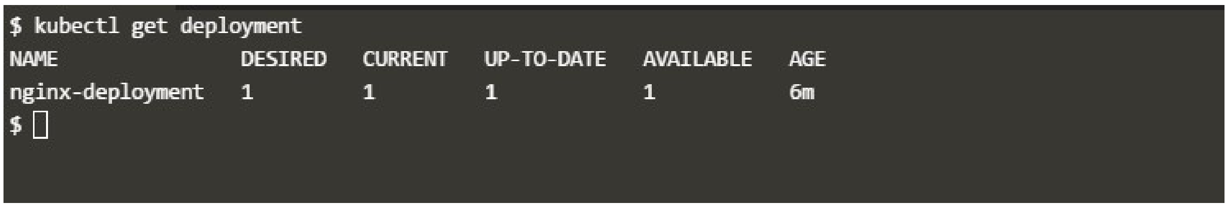
```
kubectl create -f deployment.yaml
```



```
$ kubectl create -f deployment.yaml
deployment "nginx-deployment" created
$
```

Image – Kubectl – Create deployment command

- Now that deployment is created,lets check the deployment information using *kubectl get deployment* command :



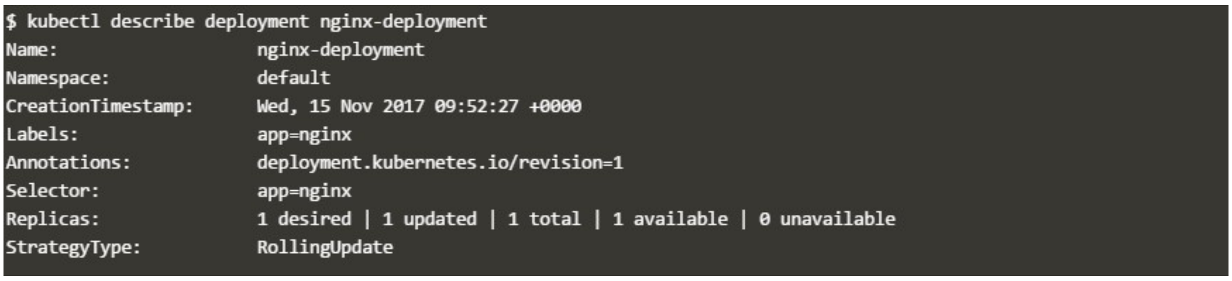
```
$ kubectl get deployment
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   1         1         1            1           6m
$
```

Image – Kubectl – get deployment command

When you inspect the Deployments in your cluster, the following fields are displayed:

- NAME lists the names of the Deployments in the cluster.
- DESIRED displays the desired number of *replicas* of the application, which you define when you create the Deployment. This is the *desired state*.
- CURRENT displays how many replicas are currently running.
- UP-TO-DATE displays the number of replicas that have been updated to achieve the desired state.
- AVAILABLE displays how many replicas of the application are available to your users.
- AGE displays the amount of time that the application has been running.

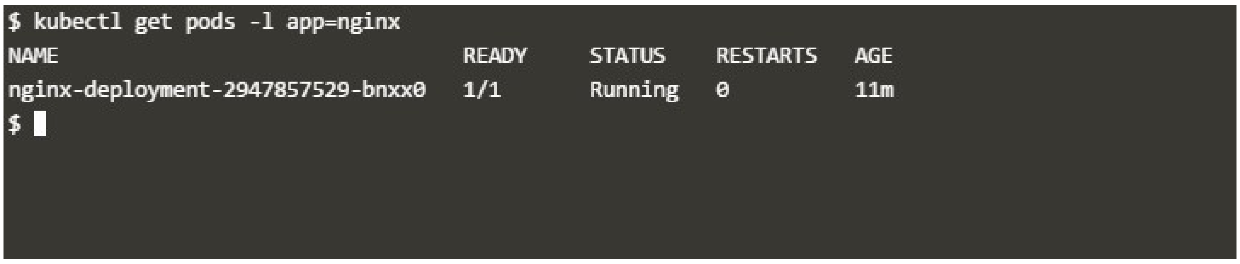
- Describe the deployment using *kubectl describe deployment* command to check the details on the deployment



```
$ kubectl describe deployment nginx-deployment
Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Wed, 15 Nov 2017 09:52:27 +0000
Labels:                 app=nginx
Annotations:            deployment.kubernetes.io/revision=1
Selector:               app=nginx
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
```

Image- Kubectl – describe deployment command

- Check the list of pods created by the deployment by using *kubectl get pods*command:



```
$ kubectl get pods -l app=nginx
NAME                                READY     STATUS    RESTARTS   AGE
nginx-deployment-2947857529-bnxx0   1/1       Running   0          11m
$ 
```

Image – Kubectl – get pods command

Step #3.Create service

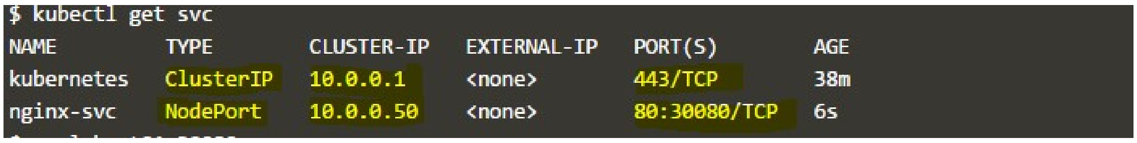
Kubernetes has powerful networking capabilities that control how applications communicate. These networking configurations can also be controlled via YAML.The Service selects all applications with the label *ngnix*. As multiple replicas, or instances, are deployed, they will be automatically load balanced based on this common label. The Service makes the application available via a NodePort.

Kubernetes Service is an abstraction which defines a logical set of Pods and a policy by which to access them (micro-service). The set of Pods targeted by a Service is determined by a Label Selector

```
apiVersion: v1
kind: Service
metadata:
 name: nginx-svc
 labels:
 app: nginx
spec:
 type: NodePort
 ports:
 - port: 80
 nodePort: 30080
 selector:
 app: nginx
```

Step #4.Deploy service

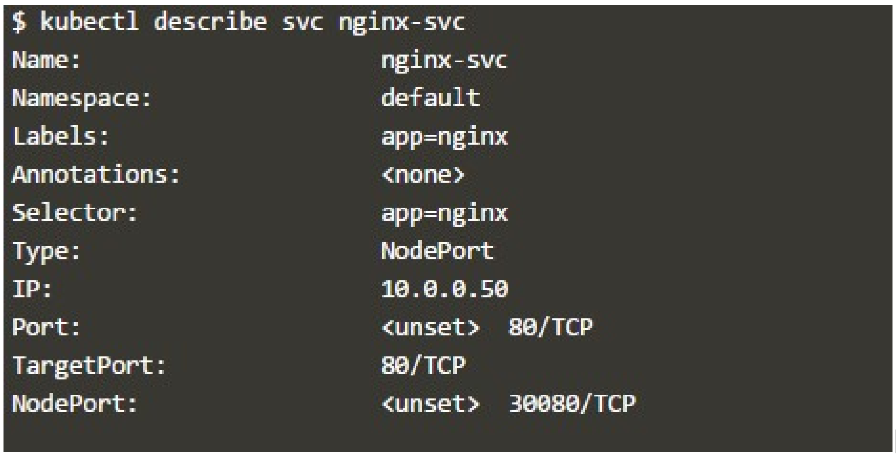
- Use *kubectl create command* to create new service based on the service.yaml file created in the previous step
- Check the details of all the Service objects deployed



```
$ kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.0.0.1     <none>        443/TCP        38m
nginx-svc    NodePort    10.0.0.50    <none>        80:30080/TCP   6s
```

Image – Kubectl – get Svc command command

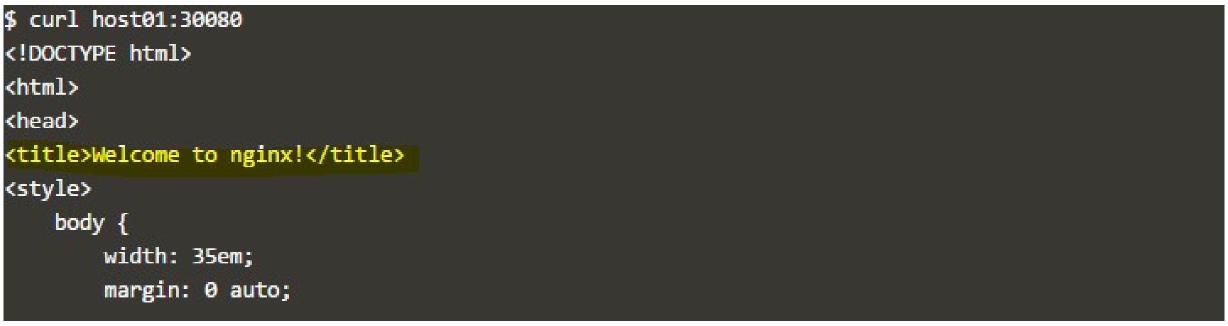
- Use describe command to discover more details about the configuration of all the Service objects deployed



```
$ kubectl describe svc nginx-svc
Name:                    nginx-svc
Namespace:               default
Labels:                  app=nginx
Annotations:             <none>
Selector:                app=nginx
Type:                    NodePort
IP:                      10.0.0.50
Port:                    <unset>  80/TCP
TargetPort:              80/TCP
NodePort:                <unset>  30080/TCP
```

Image – Kubectl – describe svc command command

Use curl command to Issuing requests to the port *30080*



```
$ curl host01:30080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
```

Image – curl command to issue request

Step #5.Update nginx deployment to have 4 replicas

Modify deployment.yaml to update the nginx deployment to have 4 replicas.

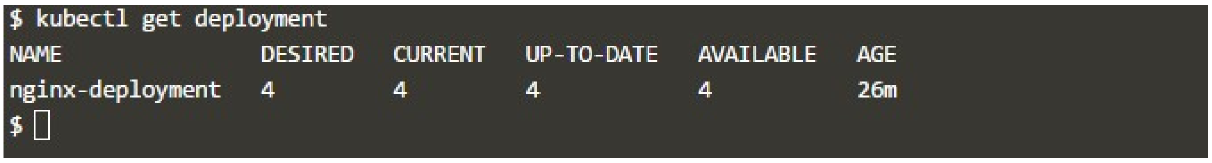
```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 4
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

Step #6.Apply the updated nginx deployment to have 4 replicas

Apply the changes using kubectl apply command

```
kubectl apply -f deployment.yaml
```

- Now that deployment is created,lets check the deployment information using *kubectl get deployment* command :



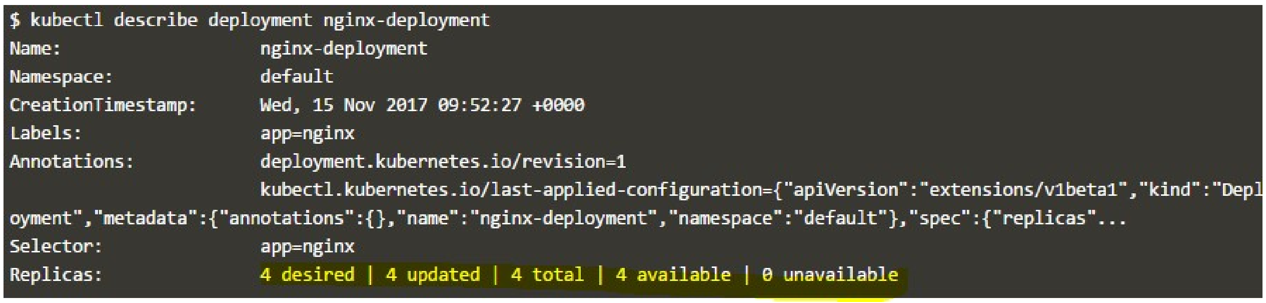
```
$ kubectl get deployment
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   4         4         4            4           26m
$ 
```

Image – Kubectl – get deployment command

When you inspect the Deployments in your cluster, the following fields are displayed:

- NAME lists the names of the Deployments in the cluster.
- DESIRED displays the desired number of *replicas* of the application, which you define when you create the Deployment. This is the *desired state*.
- CURRENT displays how many replicas are currently running.
- UP-TO-DATE displays the number of replicas that have been updated to achieve the desired state.
- AVAILABLE displays how many replicas of the application are available to your users.
- AGE displays the amount of time that the application has been running.

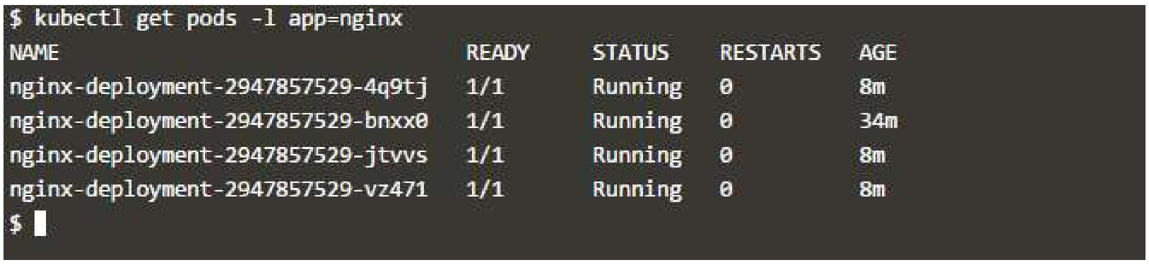
- Describe the deployment using *kubectl describe deployment* command to check the details on the deployment



```
$ kubectl describe deployment nginx-deployment
Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Wed, 15 Nov 2017 09:52:27 +0000
Labels:                 app=nginx
Annotations:            deployment.kubernetes.io/revision=1
                        kubectl.kubernetes.io/last-applied-configuration={"apiVersion":"extensions/v1beta1","kind":"Depl
oyment","metadata":{"annotations":{},"name":"nginx-deployment","namespace":"default"},"spec":{"replicas"...
Selector:               app=nginx
Replicas:               4 desired | 4 updated | 4 total | 4 available | 0 unavailable
```

Image – Kubectl – describe deployment command

- Check the list of pods created by the deployment by using *kubectl get pods*command:



```
$ kubectl get pods -l app=nginx
NAME                               READY     STATUS    RESTARTS   AGE
nginx-deployment-2947857529-4q9tj  1/1       Running   0          8m
nginx-deployment-2947857529-bnxx0  1/1       Running   0          34m
nginx-deployment-2947857529-jtvvs  1/1       Running   0          8m
nginx-deployment-2947857529-vz471  1/1       Running   0          8m
$ 
```

Image – Kubectl – get Pods command

As all the Pods have the same label selector, they'll be load balanced behind the Service NodePort deployed. Issuing requests to the port will result in different containers processing the request

Running Kubernetes on Microsoft Azure

Azure Container Service (AKS) is managed Kubernetes offering from Azure.As a hosted Kubernetes service, Azure handles all heavy lifting of all the complexity, operational overhead of managing a Kubernetes cluster for you.

As a managed Kubernetes service, AKS provides:

- Automated Kubernetes version upgrades and patching
- Easy cluster scaling
- Self-healing hosted control plane (masters)
- Cost savings – pay only for running agent pool nodes

In short, AKS would provide a container hosting environment by using open-source tools and technologies. To this end, standard Kubernetes API standard endpoints are exposed and you can leverage any software that is capable of talking to a Kubernetes cluster.Like for example,kubectl

Now we can see how to create simple cluster using AKS.This quickstart assumes a basic understanding of Kubernetes concepts, please refer earlier posts for understanding on Kubernetes & how to create,deploy & rollout updates to the cluster.

Step#1.Launch Azure Cloud Shell

Once you login to Azure Portal,on the upper right corner,there is an option to Azure Cloud Shell.Azure Cloud Shell is a free Bash shell has the Azure CLI preinstalled and configured to use with your account.

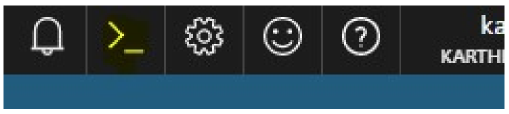
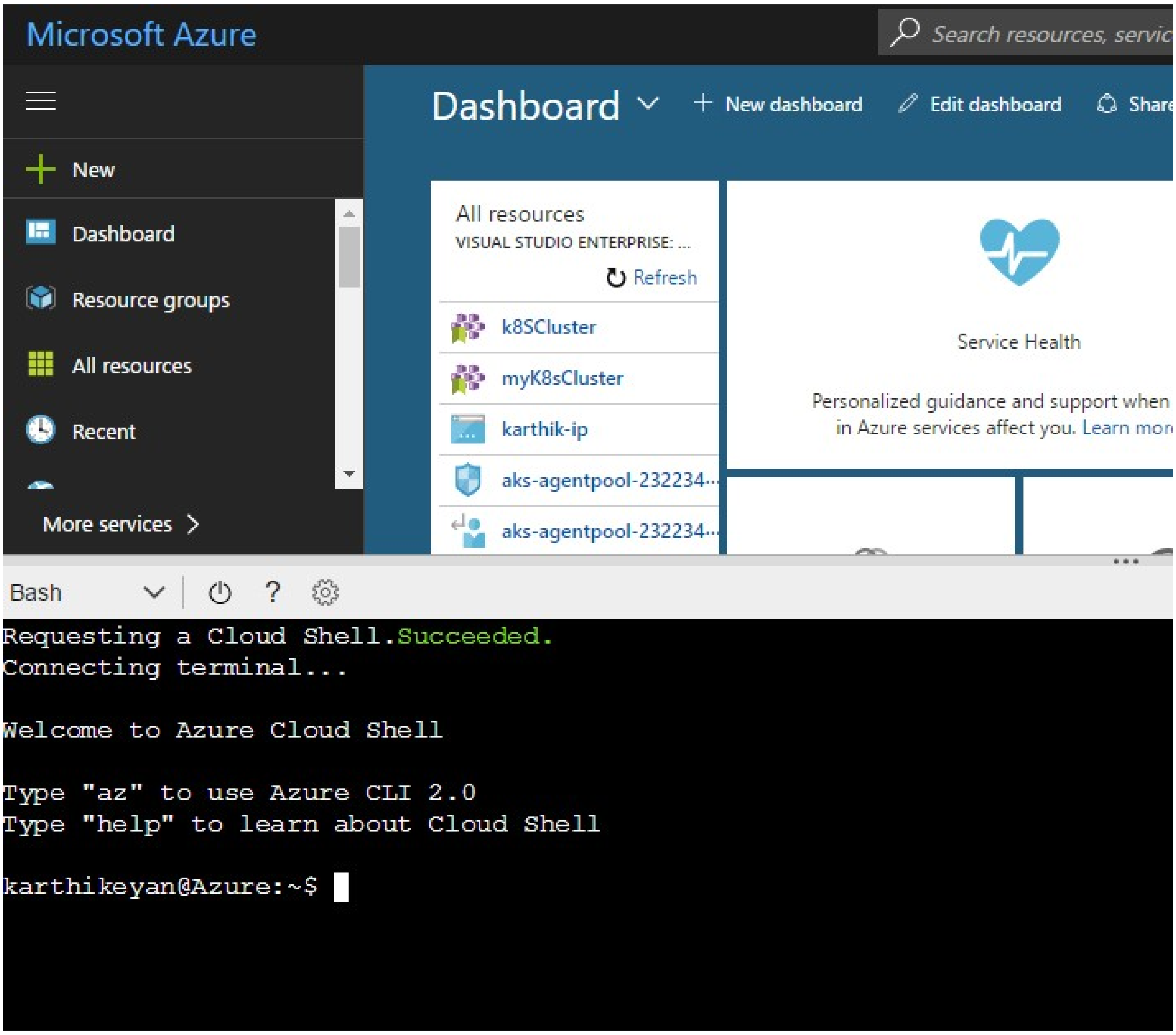


Image – Launch Azure Cloud Shell



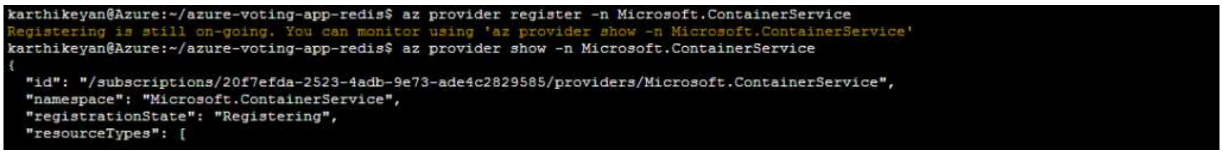
```
Requesting a Cloud Shell.Succeeded.
Connecting terminal...

Welcome to Azure Cloud Shell

Type "az" to use Azure CLI 2.0
Type "help" to learn about Cloud Shell

karthikeyan@Azure:~$
```

Image – Azure Portal

Step#2.Enable AKS Preview

From the Azure Shell,enable AKS preview by the below command.

```
az provider register -n Microsoft.ContainerService
```

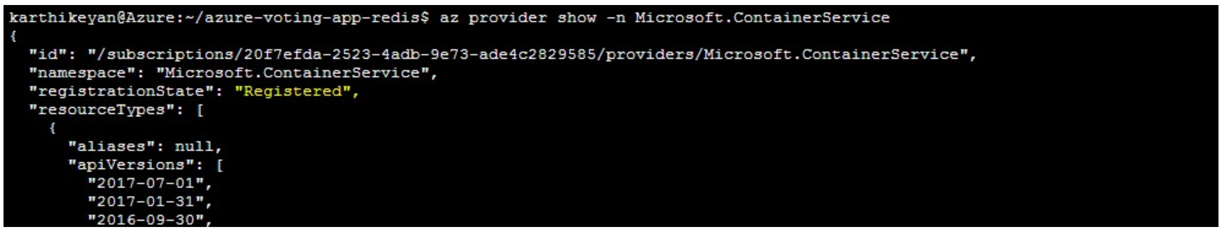


```
karthikeyan@Azure:~/azure-voting-app-redis$ az provider register -n Microsoft.ContainerService
Registering is still on-going. You can monitor using 'az provider show -n Microsoft.ContainerService'
karthikeyan@Azure:~/azure-voting-app-redis$ az provider show -n Microsoft.ContainerService
{
  "id": "/subscriptions/20f7efda-2523-4adb-9e73-ade4c2829585/providers/Microsoft.ContainerService",
  "namespace": "Microsoft.ContainerService",
  "registrationState": "Registering",
  "resourceTypes": [
```

Image – Enable AKS P review

On the registrationState field you can see that its ‘Registering’,you can check the status of the registration by following command.

```
az provider show -n Microsoft.ContainerService
```



```
karthikeyan@Azure:~/azure-voting-app-redis$ az provider show -n Microsoft.ContainerService
{
  "id": "/subscriptions/20f7efda-2523-4adb-9e73-ade4c2829585/providers/Microsoft.ContainerService",
  "namespace": "Microsoft.ContainerService",
  "registrationState": "Registered",
  "resourceTypes": [
    {
      "aliases": null,
      "apiVersions": [
        "2017-07-01",
        "2017-01-31",
        "2016-09-30",
```

Image – AKS Registration Successful

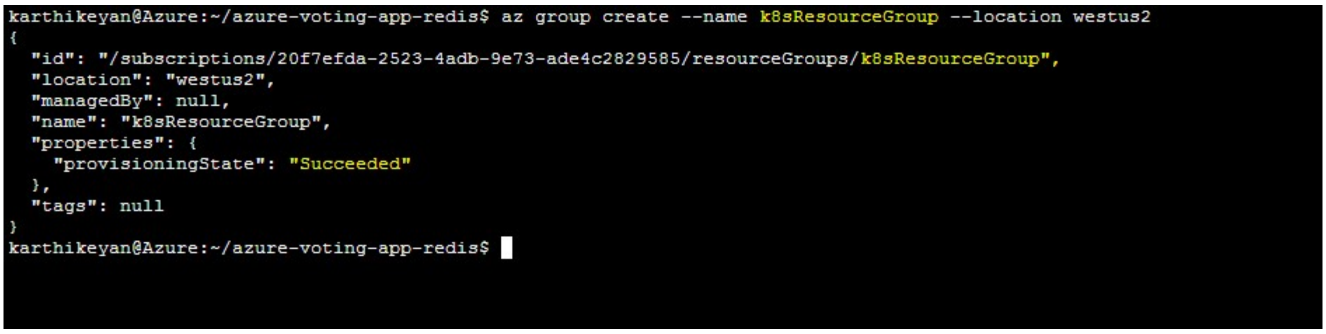
Once registration is complete, we are now ready to create a Kubernetes cluster with AKS.

Step#3.Resource group creation

Before we create the cluster & nodes,we would need to create Azure resource group which is nothing but a logical group in which Azure resources are deployed and managed.

Create new resource using below command.

az group create --name *k8SResourceGroup* --location *westus2*



```
karthikeyan@Azure:~/azure-voting-app-redis$ az group create --name k8sResourceGroup --location westus2
{
  "id": "/subscriptions/20f7efda-2523-4adb-9e73-ade4c2829585/resourceGroups/k8sResourceGroup",
  "location": "westus2",
  "managedBy": null,
  "name": "k8sResourceGroup",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null
}
karthikeyan@Azure:~/azure-voting-app-redis$
```

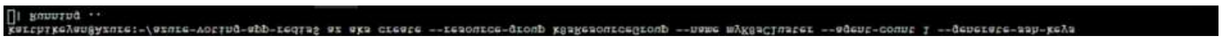
Image – Azure create new resource group command

Before proceeding to the next step,check the provisioningState on the ouput.It should be "Succeeded"

Step#4.Kubernetes cluster creation

Creates a sample cluster named *myK8sCluster* with one node.

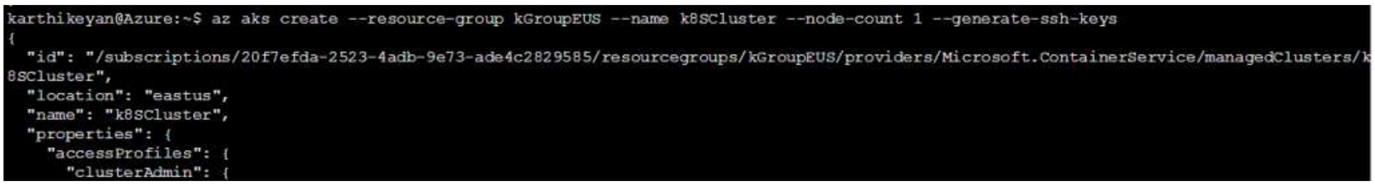
```
az aks create --resource-group k8SResourceGroup --name myK8sCluster --node-count 1 --generate-ssh-keys
```



```
karthikeyan@Azure:~/azure-voting-app-redis$ az aks create --resource-group k8sResourceGroup --name myK8sCluster --node-count 1 --generate-ssh-keys
 - Running ..
```

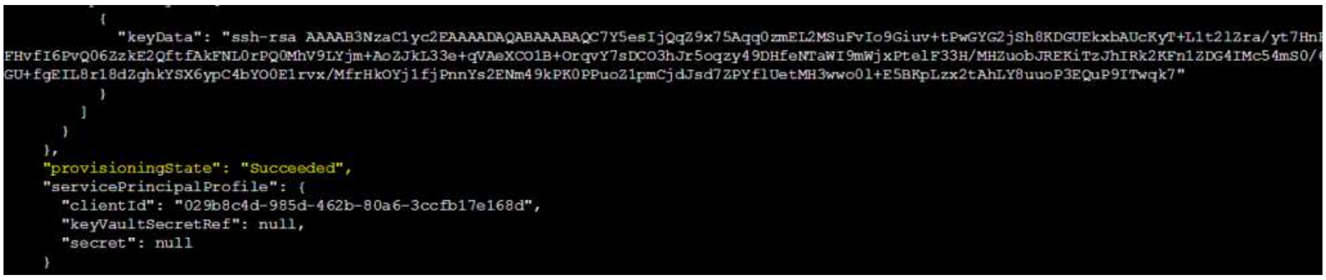
Image – Azure create cluster

Above command would take sometime to create the cluster, once the command completes and returns JSON-formatted information about the cluster.



```
karthikeyan@Azure:~$ az aks create --resource-group kGroupEUS --name k8SCluster --node-count 1 --generate-ssh-keys
{
  "id": "/subscriptions/20f7efda-2523-4adb-9e73-ade4c2829585/resourcegroups/kGroupEUS/providers/Microsoft.ContainerService/managedClusters/k
8SCluster",
  "location": "eastus",
  "name": "k8SCluster",
  "properties": {
    "accessProfiles": {
      "clusterAdmin": {
```

Image – JSON-formatted output about the cluster



```
        {
          "keyData": "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC7Y5esIjQqZ9x75Aqq0zmEL2MSuFvIo9Giuv+tPwGYG2jSh8KDGUEkxbAUcKyT+L1t21Zra/yt7HnE
FHvfI6PvQ06ZzkE2QftfAkFNL0rPQ0MhV9LYjm+AoZJkL33e+qVAeXCO1B+OrqvY7sDCO3hJr5oqzy49DHfeNTaWI9mWjxPtelF33H/MHZuobJREKiTzJhIRk2KFn1ZDG4IMc54mS0/
GU+fgEIL8r18dZghkYSX6ypC4bYO0E1rvx/MfrHkOYj1fjPnnYs2ENm49kPK0PPuoZ1pmCjdJsd7ZPYflUetMH3wwo01+E5BKpLzx2tAhLY8uuoP3EQuP9ITwqk7"
        }
      ]
    }
  },
  "provisioningState": "Succeeded",
  "servicePrincipalProfile": {
    "clientId": "029b8c4d-985d-462b-80a6-3ccfb17e168d",
    "keyVaultSecretRef": null,
    "secret": null
  }
```

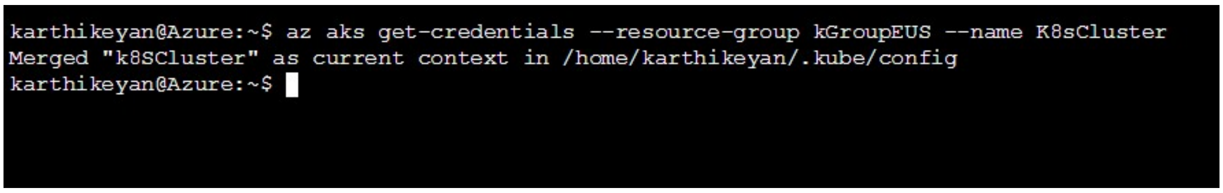
Image – JSON-formatted output about the cluster

Step#5.Connect to Kubernetes cluster

For us to connect, manage Kubernetes cluster, we are going to use kubectl, the Kubernetes command-line client.Azure cloud shell has already built-in kubectl so we don't have to install them separately.

To configure kubectl to connect to our Kubernetes cluster, run the following command. This step downloads credentials and configures the Kubernetes CLI to use them.

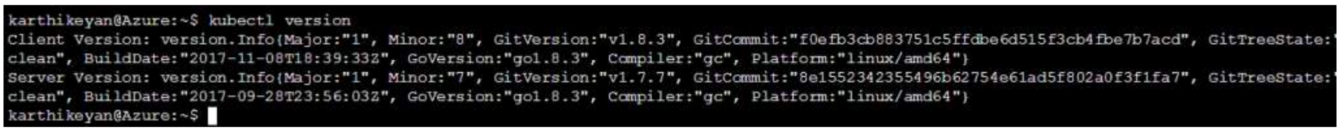
```
az aks get-credentials --resource-group k8SResourceGroup --name myK8sCluster
```



```
karthikeyan@Azure:~$ az aks get-credentials --resource-group kGroupEUS --name K8sCluster
Merged "k8SCluster" as current context in /home/karthikeyan/.kube/config
karthikeyan@Azure:~$
```

Image – Connect to Kubernetes cluster

To verify kubectl configuration, check the version of kubectl



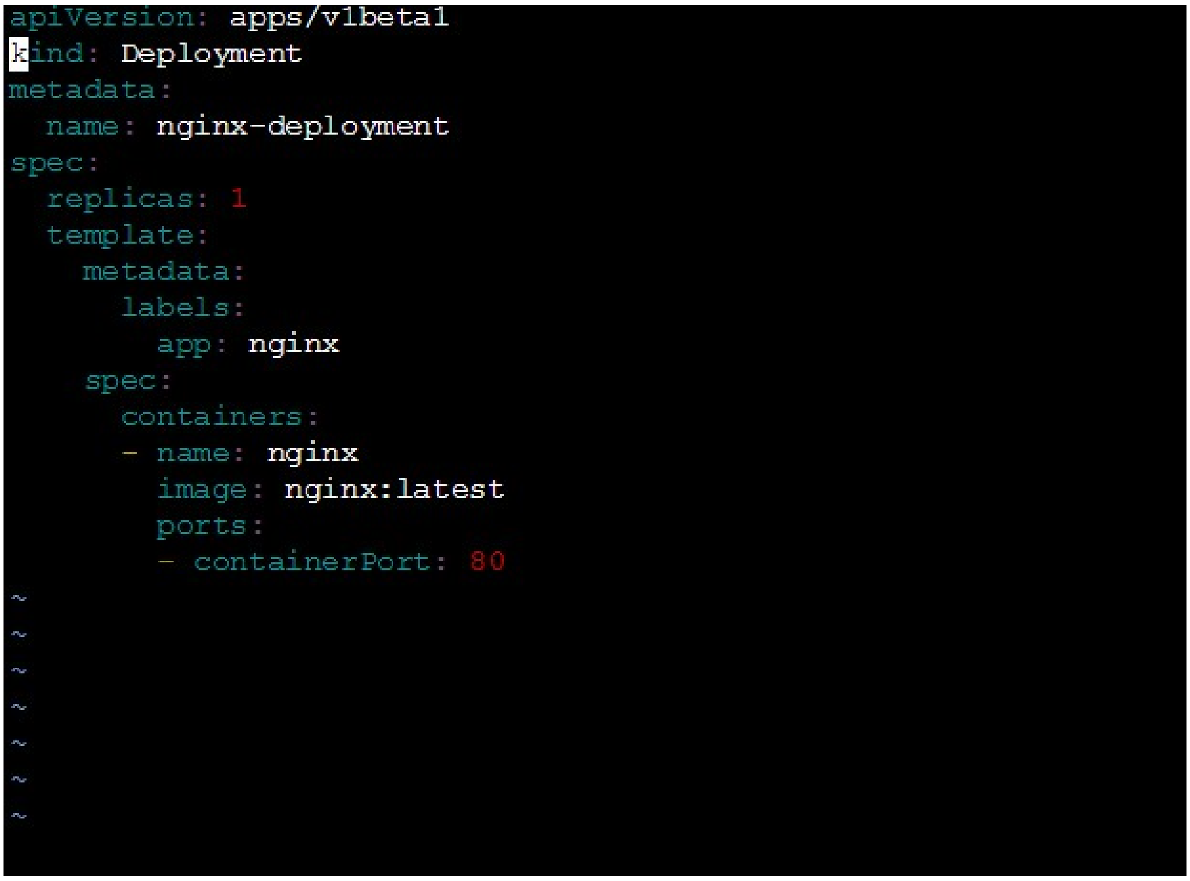
```
karthikeyan@Azure:~$ kubectl version
Client Version: version.Info{Major:"1", Minor:"8", GitVersion:"v1.8.3", GitCommit:"f0efb3cb883751c5ffdbe6d515f3cb4fbe7b7acd", GitTreeState:"
clean", BuildDate:"2017-11-08T18:39:33Z", GoVersion:"go1.8.3", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"7", GitVersion:"v1.7.7", GitCommit:"8e1552342355496b62754e61ad5f802a0f3f1fa7", GitTreeState:"
clean", BuildDate:"2017-09-28T23:56:03Z", GoVersion:"go1.8.3", Compiler:"gc", Platform:"linux/amd64"}
karthikeyan@Azure:~$
```

Image – kubectl version

Step#6. Deploy new application on the cluster

Now we are going to create new Kubernetes manifest file that defines a desired state for the cluster, including what container images should be running etc.,

Create file named Deployment.yml,you can use vi editor to create this file.



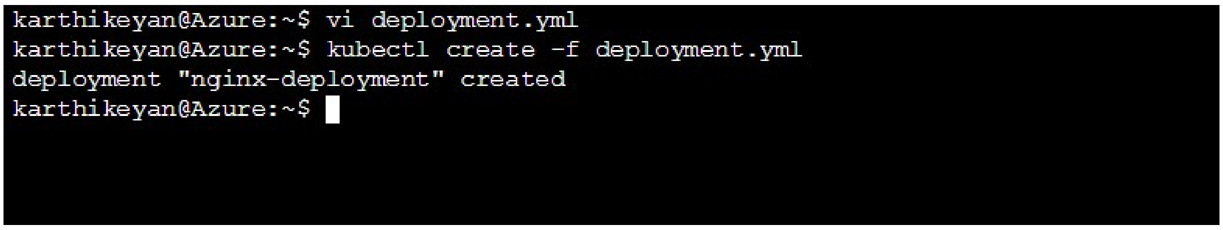
```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
~
~
~
~
~
~
~
```

Image – Deployment manifest file

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 1
  template:
```

```
metadata:
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
```

Use the kubectl create command to deploy the application.



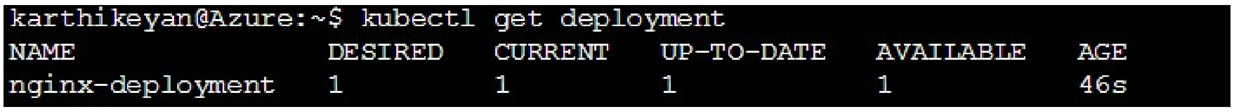
```
karthikeyan@Azure:~$ vi deployment.yml
karthikeyan@Azure:~$ kubectl create -f deployment.yml
deployment "nginx-deployment" created
karthikeyan@Azure:~$
```

Image – Create new Deployment

In this example:

- A Deployment named *nginx-deployment* is created, indicated by the metadata: name field.
- The Deployment creates *1 replicated Pods*, indicated by the replicas field.
- The Pod template's specification, or template: spec field, indicates that the Pods run one container, nginx, which runs the nginx Docker Hub latest image (you can also specify the version ex.1.7.9.)
- The Deployment opens port 80 for use by the Pods.

Now that deployment is created,lets check the deployment information using *kubectl get deployment* command :



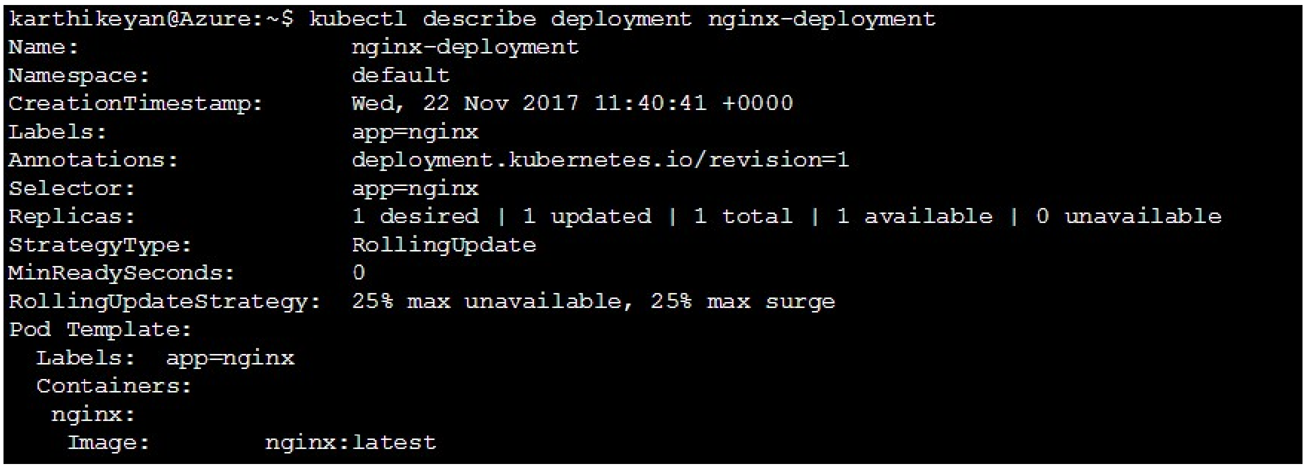
```
karthikeyan@Azure:~$ kubectl get deployment
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   1         1         1            1           46s
```

Image – kubectl get deployment command

When you inspect the Deployments in your cluster, the following fields are displayed:

- NAME lists the names of the Deployments in the cluster.
- DESIRED displays the desired number of *replicas* of the application, which you define when you create the Deployment. This is the *desired state*.
- CURRENT displays how many replicas are currently running.
- UP-TO-DATE displays the number of replicas that have been updated to achieve the desired state.
- AVAILABLE displays how many replicas of the application are available to your users.
- AGE displays the amount of time that the application has been running.

Describe the deployment using *kubectl describe deployment* command to check the details on the deployment.



```
karthikeyan@Azure:~$ kubectl describe deployment nginx-deployment
Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Wed, 22 Nov 2017 11:40:41 +0000
Labels:                 app=nginx
Annotations:            deployment.kubernetes.io/revision=1
Selector:               app=nginx
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx
  Containers:
   nginx:
    Image:        nginx:latest
```

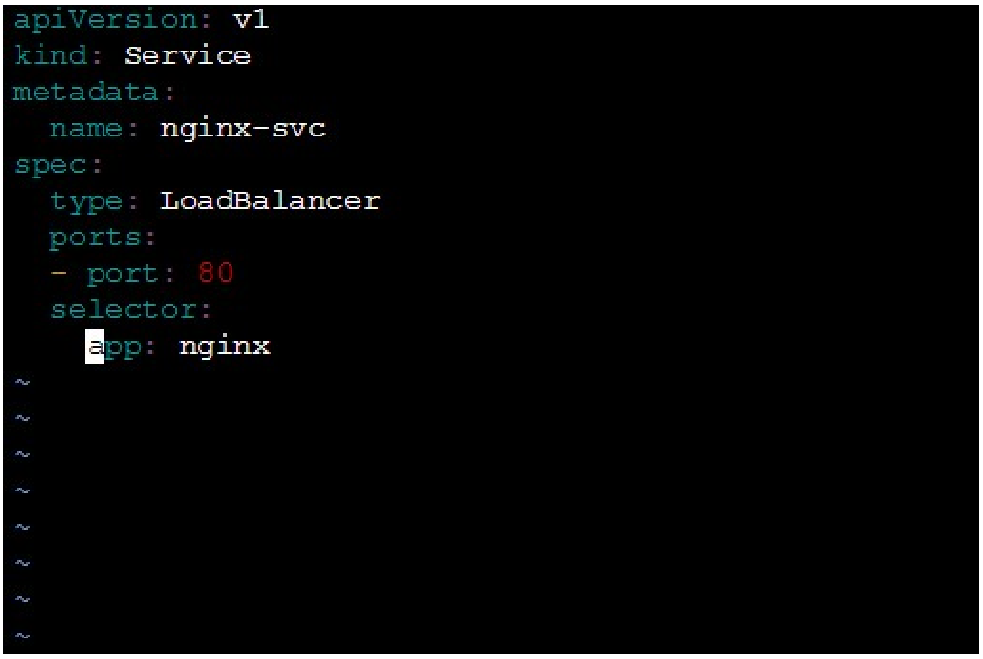
Image – kubectl describe deployment command

Step#7. Create service

Kubernetes Service is an abstraction which defines a logical set of Pods and a policy by which to access them (micro-service). The set of Pods targeted by a Service is determined by a Label Selector

Kubernetes has powerful networking capabilities that control how applications communicate. These networking configurations can also be controlled via YAML.The Service selects all applications with the label *ngnix*. As multiple replicas, or instances, are deployed, they will be automatically load balanced based on this common label. The Service makes the application available via a NodePort.

Create new file named svc.yml to define load balancers & apps.



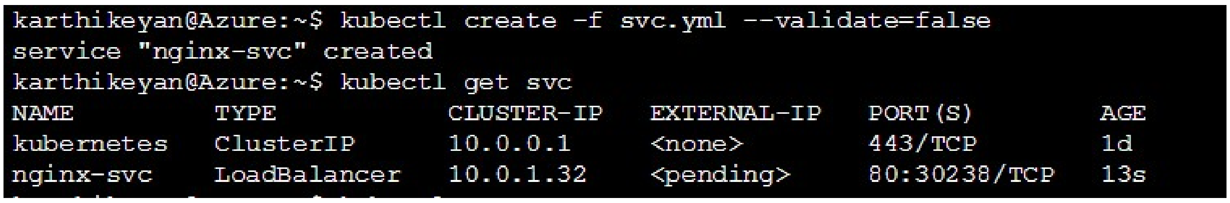
```
apiVersion: v1
kind: Service
metadata:
  name: nginx-svc
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: nginx
~
~
~
~
~
~
~
~
```

Image – Service Yaml file

```
apiVersion: v1
kind: Service
metadata:
 name: nginx-svc
 labels:
 app: nginx
spec:
 type: LoadBalancer
 ports:
 - port: 80
```

```
selector:
app: nginx
```

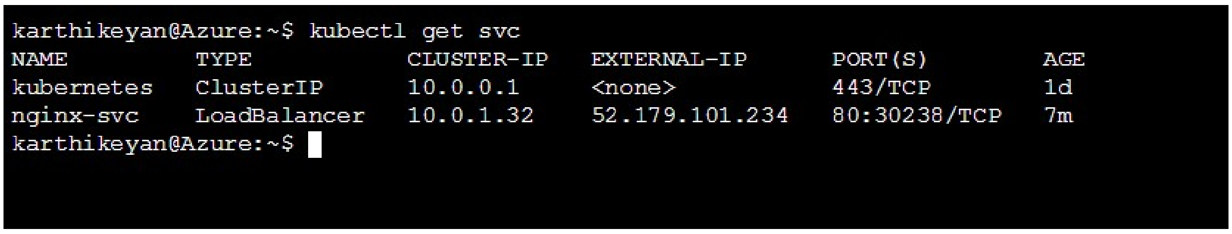
Use *kubectl create command* to create new service based on the svc.yaml file created in the previous step.



```
karthikeyan@Azure:~$ kubectl create -f svc.yml --validate=false
service "nginx-svc" created
karthikeyan@Azure:~$ kubectl get svc
NAME          TYPE           CLUSTER-IP    EXTERNAL-IP   PORT(S)        AGE
kubernetes    ClusterIP      10.0.0.1      <none>        443/TCP        1d
nginx-svc     LoadBalancer   10.0.1.32     <pending>     80:30238/TCP   13s
```

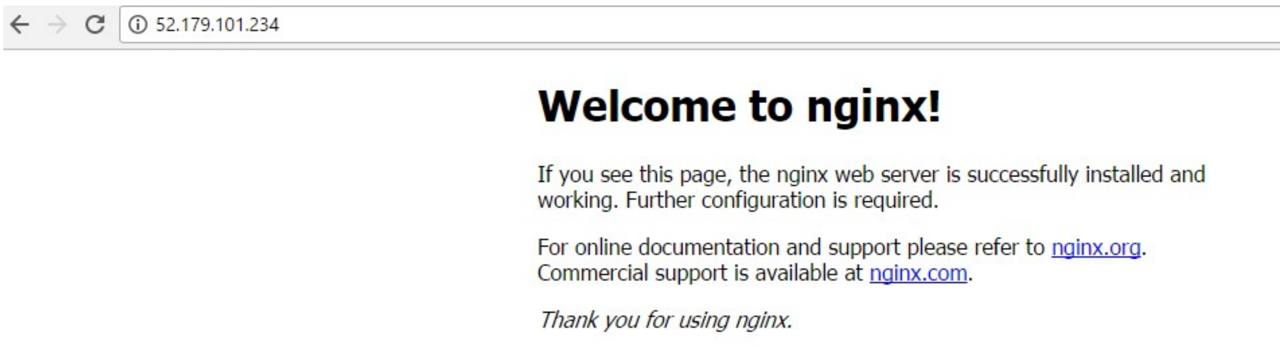
Image – Create new service

Once the External-IP is available,we can browse the ngnix page.



```
karthikeyan@Azure:~$ kubectl get svc
NAME          TYPE           CLUSTER-IP    EXTERNAL-IP      PORT(S)        AGE
kubernetes    ClusterIP      10.0.0.1      <none>           443/TCP        1d
nginx-svc     LoadBalancer   10.0.1.32     52.179.101.234   80:30238/TCP   7m
karthikeyan@Azure:~$
```

Image – Once External IP is assigned we can browse ngnix page



52.179.101.234

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

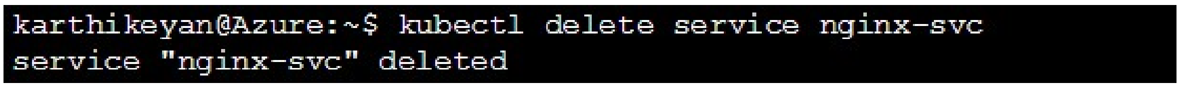
Thank you for using nginx.

Image – ngnix landing page

Step#8. Clean up

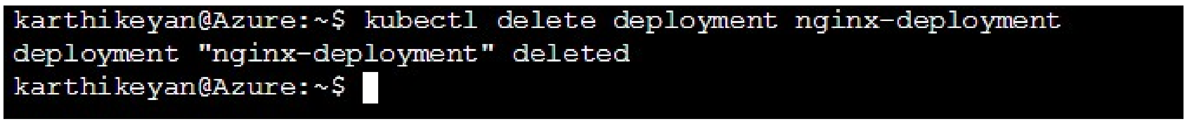
Finally now its time to clean up the resources what we have created:

```
kubectl delete service my-nginx
kubectl delete deployment my-nginx
```



```
karthikeyan@Azure:~$ kubectl delete service nginx-svc
service "nginx-svc" deleted
```

Image – kubectl detele service command



```
karthikeyan@Azure:~$ kubectl delete deployment nginx-deployment
deployment "nginx-deployment" deleted
karthikeyan@Azure:~$
```

Image – kubectl detele deployment command

Kubless – Kubernetes Native Serverless Framework

kubeless is a Kubernetes-native serverless framework that lets you deploy small bits of code without having to worry about the underlying infrastructure plumbing. It leverages Kubernetes resources to provide auto-scaling, API routing, monitoring, troubleshooting and more.

Kubeless

Image – Kubeless – Kubernetes-native serverless framework

Before we move on to tutorial, First – little bit of intro on Serverless,it allows developers to build and run applications and services without thinking about the servers actually running the code. Serverless services, or FaaS (Functions-as-a-Service) providers, instrument this concept by allowing developers to upload the code while taking care of deploying running and scaling it. AWS Lambda was the first one in the market to offer this kind.

Popular cloud providers that supports Function As A Service (FaaS) as follows:

- AWS via Lamdba Service
- Azure via Azure Functions
- Google via Google Cloud Functions

Kubeless aims to be an open source FaaS solution to clone the functionalities of AWS Lamdba/Google Cloud Functions.

For more details on Serverless & comparison, look up here.

How it works

Serverless services or FaaS lets you run code without provisioning or managing servers (but still servers are needed). You pay only for the compute time you consume there is no charge when your code is not running. You can run code for virtually any type of application or backend service all with zero administration. Just upload your code and FaaS provider would take care of everything required to run and scale your code with high availability. You can set up your code to automatically trigger from other services or call it directly from any web or mobile app.

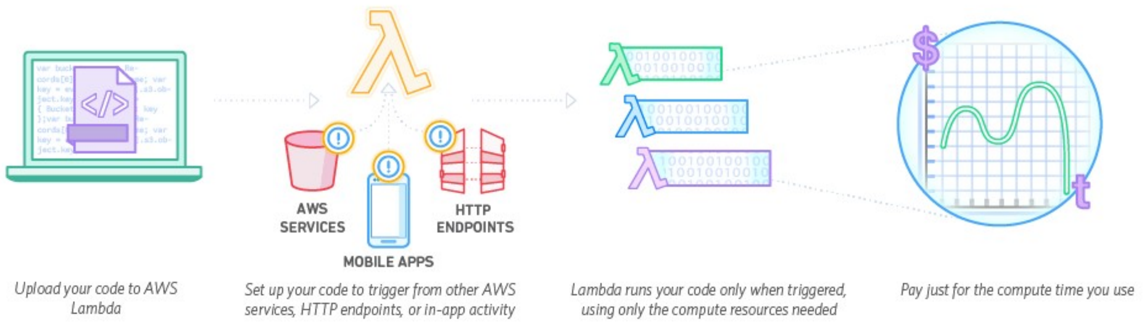

AWS SERVICES
MOBILE APPS
HTTP ENDPOINTS
Upload your code to AWS Lambda
Set up your code to trigger from other AWS services, HTTP endpoints, or in-app activity
Lambda runs your code only when triggered, using only the compute resources needed
Pay just for the compute time you use

Image – AWS Lambda / How it works

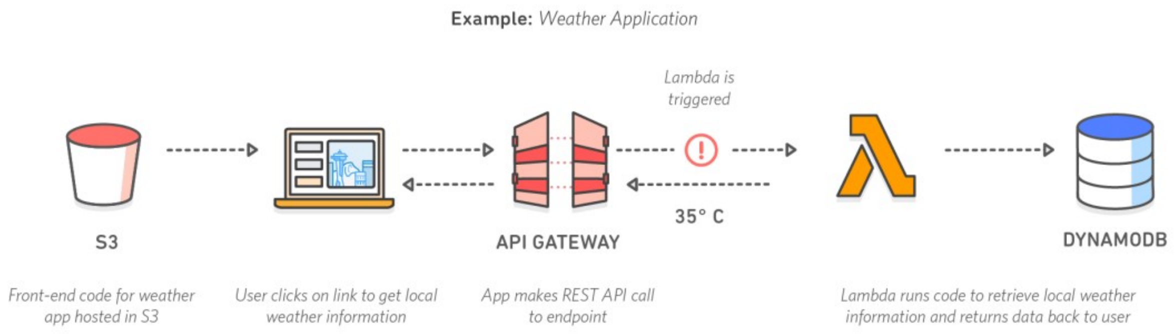

Example: Weather Application
Lambda is triggered
35° C
S3
API GATEWAY
DYNAMODB
Front-end code for weather app hosted in S3
User clicks on link to get local weather information
App makes REST API call to endpoint
Lambda runs code to retrieve local weather information and returns data back to user

Image- Another Sample

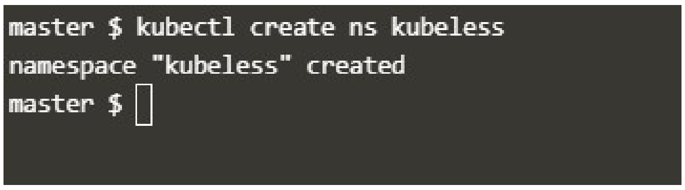
With Kubeless you can deploy functions without the need to build containers. These functions can be called via regular HTTP(S) calls or triggered by events submitted to message brokers like Kafka.

Currently Kubeless Functions have three possible types:

- HTTP triggered (function will expose an HTTP endpoint)
- Pubsub triggered (function will consume event on a specific topic; a running kafka cluster on your k8s is required)
- Schedule triggered (function will be called on a cron schedule)

Kubeless Installation

Step #1 : Create a *kubeless* namespace where you will install the controller.

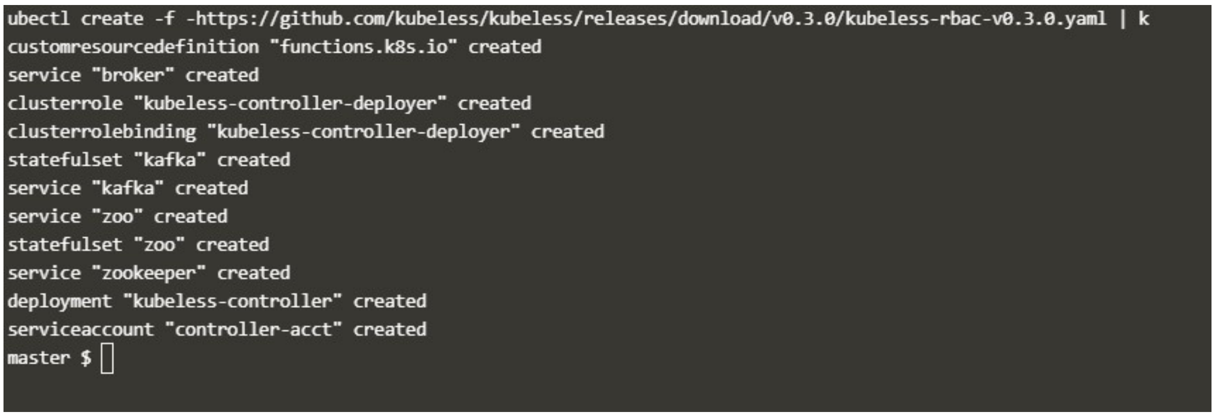


```
master $ kubectl create ns kubeless
namespace "kubeless" created
master $ 
```

Image – Create Kubeless namespace

Step #2 : Install the latest stable version with a *kubectl create* command

```
curl -sL https://github.com/kubeless/kubeless/releases/download/v0.3.0/kubeless-rbac-v0.3.0.yaml | kubectl create -f -
```

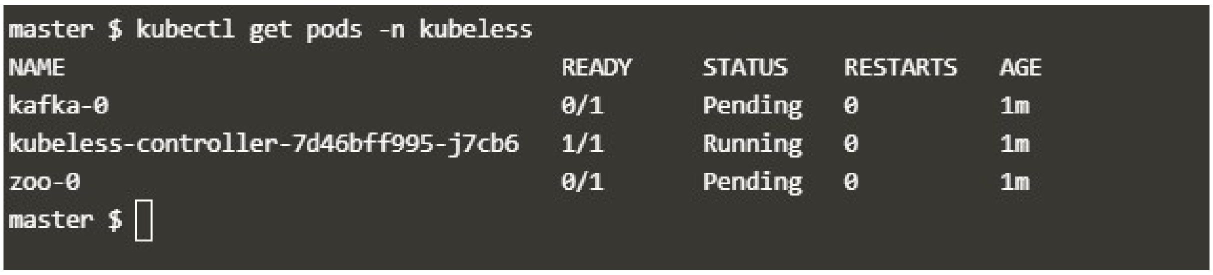


```
ubectl create -f -https://github.com/kubeless/kubeless/releases/download/v0.3.0/kubeless-rbac-v0.3.0.yaml | k
customresourcedefinition "functions.k8s.io" created
service "broker" created
clusterrole "kubeless-controller-deployer" created
clusterrolebinding "kubeless-controller-deployer" created
statefulset "kafka" created
service "kafka" created
service "zoo" created
statefulset "zoo" created
service "zookeeper" created
deployment "kubeless-controller" created
serviceaccount "controller-acct" created
master $
```

Image – Install Kubeless

You can see that few pods are being started in the *kubeless* namespace. Also if you can see that a few pods are being started in the *kubeless* namespace. The controller which will watch for function objects to be created and also two Pods to enabled PubSub function *(Kafka & Zoo pods).*

Step #3 : Check the status of the pods using get pods command



```
master $ kubectl get pods -n kubeless
NAME                                    READY    STATUS     RESTARTS   AGE
kafka-0                                 0/1      Pending    0          1m
kubeless-controller-7d46bff995-j7cb6    1/1      Running    0          1m
zoo-0                                   0/1      Pending    0          1m
master $
```

Image – check deployment status using get pods command

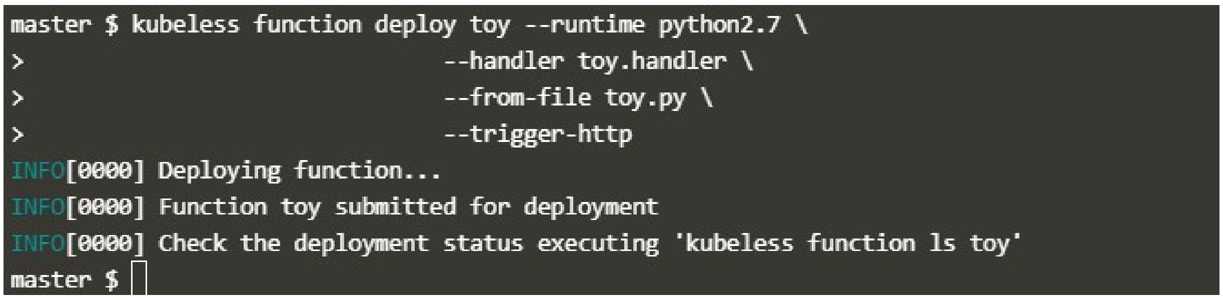
Once the controller is in *'Running'* state,we can start deploying functions.

Deploy Function

To deploy a function, we are going use the kubeless CLI. For the function that we are going to deploy,we have to specify a run time which language the function is in.

Also we need to specify the file that contains the function, how the function will get triggered *(here we are using an HTTP trigger)* and finally specify the function name as a handler.

```
kubeless function deploy toy --runtime python2.7 --handler toy.handler --from-file toy.py --trigger-http
```



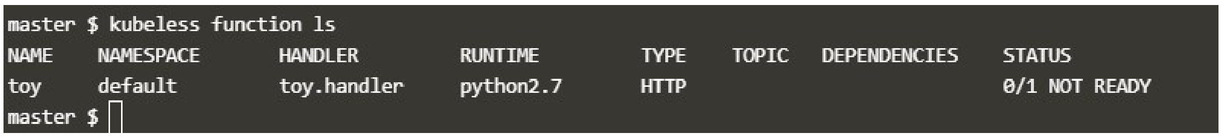
```
master $ kubeless function deploy toy --runtime python2.7 \
>                                 --handler toy.handler \
>                                 --from-file toy.py \
>                                 --trigger-http
INFO[0000] Deploying function...
INFO[0000] Function toy submitted for deployment
INFO[0000] Check the deployment status executing 'kubeless function ls toy'
master $
```

Image – Deploy kubeless function

Congrats! Now we have create new function.

We can check the list of functions with the `kubeless` CLI:

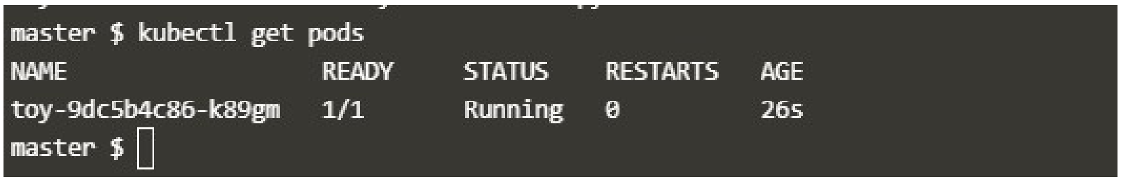
```
kubeless function ls
```



```
master $ kubeless function ls
NAME    NAMESPACE       HANDLER         RUNTIME         TYPE    TOPIC   DEPENDENCIES    STATUS
toy     default         toy.handler     python2.7       HTTP                            0/1 NOT READY
master $
```

Image – kubeless function ls command

Kubeless would have automatically created a Kubernetes deployment and service. You can check that a Pod containing your function is running:

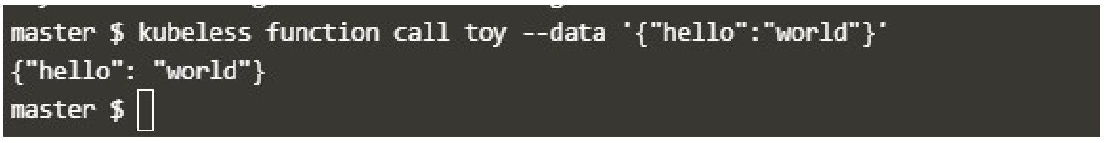


```
master $ kubectl get pods
NAME                   READY     STATUS     RESTARTS   AGE
toy-9dc5b4c86-k89gm    1/1       Running    0          26s
master $
```

Image – kubectl get pods command

Call Function via HTTP

To test the function, call the function using the kubeless CLI command:

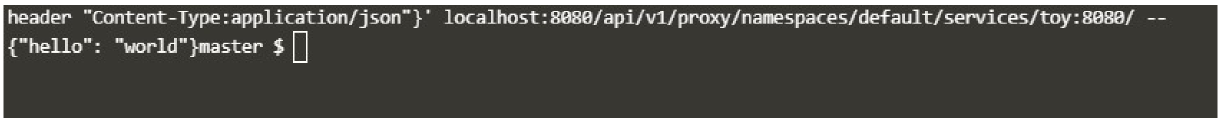


```
master $ kubeless function call toy --data '{"hello":"world"}'
{"hello": "world"}
master $
```

Image – Call kubeless function from CLI

If proxy is configured,we can call it using curl command

```
curl --data '{"hello":"world"}'
localhost:8080/api/v1/proxy/namespaces/default/services/toy:8080/ --header
"Content-Type:application/json"
```

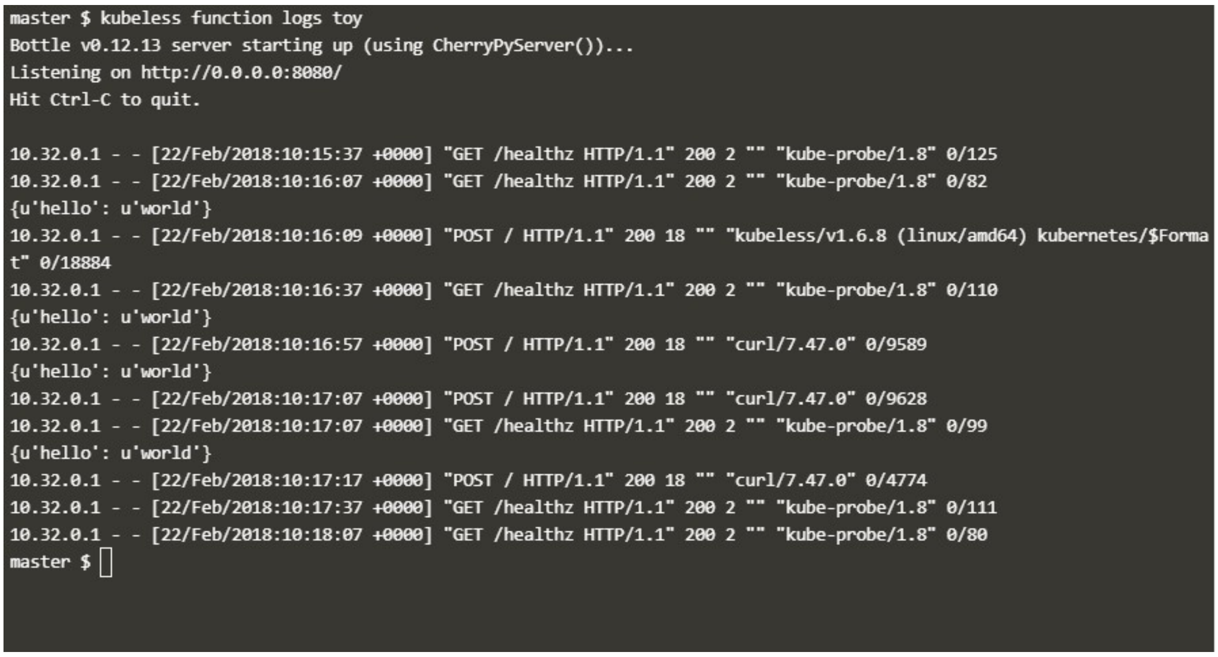


```
header "Content-Type:application/json"}' localhost:8080/api/v1/proxy/namespaces/default/services/toy:8080/ --
{"hello": "world"}master $
```

Image – Call function using curl command

For viewing the logs of the function,use logs command

```
kubeless function logs toy
```

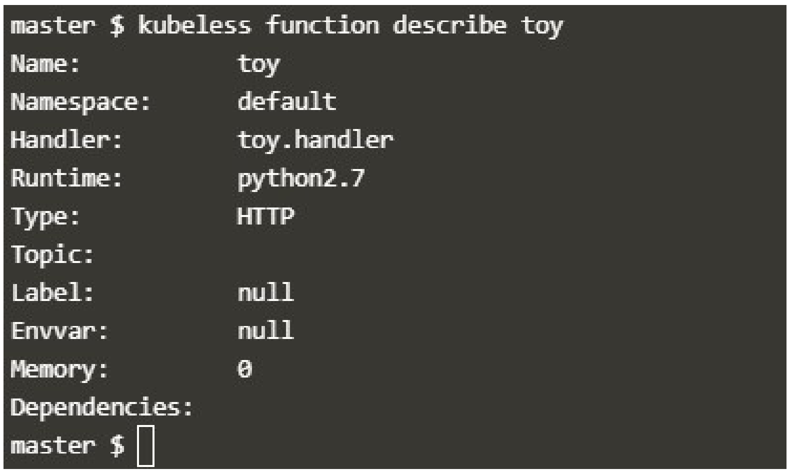


```
master $ kubeless function logs toy
Bottle v0.12.13 server starting up (using CherryPyServer())...
Listening on http://0.0.0.0:8080/
Hit Ctrl-C to quit.

10.32.0.1 - - [22/Feb/2018:10:15:37 +0000] "GET /healthz HTTP/1.1" 200 2 "" "kube-probe/1.8" 0/125
10.32.0.1 - - [22/Feb/2018:10:16:07 +0000] "GET /healthz HTTP/1.1" 200 2 "" "kube-probe/1.8" 0/82
{u'hello': u'world'}
10.32.0.1 - - [22/Feb/2018:10:16:09 +0000] "POST / HTTP/1.1" 200 18 "" "kubeless/v1.6.8 (linux/amd64) kubernetes/$Forma
t" 0/18884
10.32.0.1 - - [22/Feb/2018:10:16:37 +0000] "GET /healthz HTTP/1.1" 200 2 "" "kube-probe/1.8" 0/110
{u'hello': u'world'}
10.32.0.1 - - [22/Feb/2018:10:16:57 +0000] "POST / HTTP/1.1" 200 18 "" "curl/7.47.0" 0/9589
{u'hello': u'world'}
10.32.0.1 - - [22/Feb/2018:10:17:07 +0000] "POST / HTTP/1.1" 200 18 "" "curl/7.47.0" 0/9628
10.32.0.1 - - [22/Feb/2018:10:17:07 +0000] "GET /healthz HTTP/1.1" 200 2 "" "kube-probe/1.8" 0/99
{u'hello': u'world'}
10.32.0.1 - - [22/Feb/2018:10:17:17 +0000] "POST / HTTP/1.1" 200 18 "" "curl/7.47.0" 0/4774
10.32.0.1 - - [22/Feb/2018:10:17:37 +0000] "GET /healthz HTTP/1.1" 200 2 "" "kube-probe/1.8" 0/111
10.32.0.1 - - [22/Feb/2018:10:18:07 +0000] "GET /healthz HTTP/1.1" 200 2 "" "kube-probe/1.8" 0/80
master $
```

Image – View Function logs

To get the description of the function,use describe command like below

```
kubeless function describe toy
```



```
master $ kubeless function describe toy
Name:           toy
Namespace:      default
Handler:        toy.handler
Runtime:        python2.7
Type:           HTTP
Topic:
Label:          null
Envvar:         null
Memory:         0
Dependencies:
master $
```

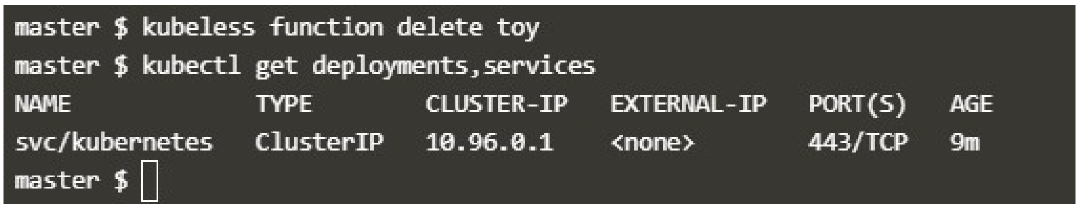
Image – Describe command to get details of function

To update a function use the kubeless function updatecommand. For example to replace the toy function which we have created with the method from the toy-udpate.py script, do:

```
kubeless function update toy --from-file toy-update.py
```

As clean up activity,we can also remove the functions,deployments we have created.

```
kubeless function delete toy
```



```
master $ kubeless function delete toy
master $ kubectl get deployments,services
NAME              TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)   AGE
svc/kubernetes    ClusterIP   10.96.0.1     <none>        443/TCP   9m
master $
```

Image – Delete command

The deployment and Kubernetes services will be removed automatically.You can use get deployments,services to check the same.

Resources

1. Kubectl cheat sheet
2. Official documentation as a reference to understand any command.
3. Take a free course on Scalable Microservices with Kubernetes.
4. If you're looking for Kubernetes examples,here it is GitHub
5. Azure CLI commands
6. Serverless Architectures