Kubernetes Essential

Practical Guide to Learn Kubernetes

Prepared by
Jakir Mahemood Patel, Software Engineer
Josphat Mutai, Cloud Infrastructure Engineer

1st Edition

LINOXIDE
Linux Freedom Thoughts

CONTENTS

1. Introduction to Kubernetes 00

2. Key definitions and concepts 00

1. What is Kubernetes? 00
2. What therefore is Containerization 00
3. What is a docker container? 00
4. How Kubernetes differs from Docker project? 00
5. What is orchestration? 00
6. Features of orchestration 00
7. Key Features of Kubernetes 00
8. Work Units of Kubernetes 00
9. Components of Kubernetes 00

3. Kubernetes Concepts 00

1. Pods 00
2 Controllers 00

4. Deploying Kubernetes Manually 00

1. Install Docker Engine on Ubuntu 00
2 Installing Kubernetes on Ubuntu 00
3 Installing etcd 2.0 on Ubuntu 00
4 Installing Addons 00

5. Downloading Kubernetes Docker Images 00

1 Setting up Kubernetes Cluster 00
2 Dockerizing the App 00
3 Writing Kubernetes Manifest Files for Sample App 00
4 Understanding Kubectl Utility 00

5 Launching and Running Container pods with Kubernetes 00
6 Kubernetes - App Deployment Flow 00
7 Kubernetes – Auto scaling 00
8 Destroying Kubernetes Cluster and Pods 00
6. Deploying Kubernetes with Ansible 00
7. Provisioning Storage in Kubernetes 00
1 Kubernetes Persistent Volumes 00
2 Requesting storage 00
3 Using Claim as a Volume 00
4 Kubernetes and NFS 00
5 Kubernetes and iSCSI 00
8. Troubleshooting Kubernetes and Systemd Services 00
1 Kubernetes Troubleshooting Commands 00
2 Networking Constraints 00
3 Inspecting and Debugging Kubernetes 00
4 Querying the State of Kubernetes 00
5 Checking Kubernetesyaml or json Files 00
6 Deleting Kubernetes Components 00
9. Kubernetes Maintenance 00
1 Monitoring Kubernetes Cluster 00
2 Managing Kubernetes with Dashboard 00
3 Logging Kubernetes Cluster 00
4 Upgrading Kubernetes 00

1. INTRODUCTION TO KUBERNETES

This chapter will give a brief overview of containers and Kubernetes and how the two technologies play a key role in shifting to DevOps methodologies and CI/CD (continuous integration/continuous deployment) strategies to accelerate innovation and service delivery.

2. KEY DEFINITIONS AND CONCEPTS

WHAT IS KUBERNETES?

In order to fully understand Kubernetes and its significance in the Information Technology World, a brief look at recent history will be quite beneficial as you will find out.

Virtualization has come a long way in Information Technology which began when there was need to share the resources of a computer among many users. As it is known, Computer resources can be pretty expensive and hence there is a need to utilize whatever you have to the optimum instead of investing in another expensive venture. In the 1960's and early 1970's IBM embarked on a journey with the objective of finding ways that will make it possible to share computer resources in a robust fashion. The breakthrough was the concept of virtualization that made computing capability costs to plunge remarkably in such proportions that it made organizations and individual entities to use computer resources devoid of owning one. Virtualization has made it possible to improve the utilization of resources and more importantly a reduction in costs.

With the constant development of technology, virtualization has not been left behind in the growth cycle. With more innovative solutions on the rise, containerization in the field of technology is the current standard that is improving efficiency and resource utilization.

WHAT THEREFORE IS CONTAINERIZATION

When cargo is being shipped from one country to another across the ocean, they are normally placed in different containers for easy management. Instead of piling up every product in one huge container, shoes and clothing for instance are placed in different containers without

either interfering with each other. The same is applied in computing Containerization. Creating a container is basically putting everything you need for your application to work be it libraries, operating system or any other technology. What has been created can be replicated and will work in any environment which saves time and makes it easy for other processes to continue without re-installing the same components of the container every time you spin a virtual machine. It is a type of a strategy in virtualization that has come about as an alternative to the native or initial hypervisor-based virtualization. Containerization involves creating separate containers at the operating system level which makes it possible to share libraries, file systems and other important components hence saving a lot of space compared to native virtualization where each virtual machine had its own components in isolation. There are few containerization technologies that offer containerization tools and API such as Docker Engine, Rkt, LXC, OpenVZ, runC and LXD. After understanding the key concepts, Kubernetes can thus be easily defined. Below are few similarities and differences between these container technologies. Kubernetes is an active open source project founded by Google to assist system developers/administrators orchestrate and manage containers in different kind of environments such as virtual, physical, and cloud infrastructure. Currently, Kubernetes project is hosted by Cloud Native Computing Foundation (CNCF).

WHAT IS A DOCKER CONTAINER?

A docker container is a lightweight software package that includes everything needed to run it, including its own minimal operating system, run-time resources, and dependencies. Docker ecosystem lies at the heart of the mass adoption and the excitement witnessed in the container space. To spin a Specific Docker container, they are developed out of images designed to provide a specific capability, for instance a database such as MariaDB, a base operating system or even a web server such as Apache. These images of Docker are made from file systems that are layered so that they are capable of sharing common files. Sharing of common files adds the advantage of reducing the usage of disk space and speeding up image download

As compared to virtual machines, containers are more resource-efficient because they do not require hypervisors. In addition, containers have less memory footprint and can help organizations avoid high costs and hassles associated with server sprawl.

HOW KUBERNETES DIFFERS FROM DOCKER PROJECT?

Docker project aims at defining a container format, building and managing individual containers

WHAT IS ORCHESTRATION?

In order to implement certain applications, many containers need to be spinned and managed. In order to optimize this process, the deployment of these containers can be automated. This is especially beneficial if there is a growth in the number of hosts. This automation process is called orchestration.

FEATURES OF ORCHESTRATION

Preparing and equipping hosts

Instantiating a set of desired containers

Maintaining failed containers for example through rescheduling them

Merging containers together through interfaces that have been agreed upon

Exposing services to machines outside of the cluster

Docker has several orchestration tolls such as Kubernetes, Docker Machine and Docker swam among others. Kubernetes is one of the most feature-rich orchestration tools and is widely used.

After building the container image you want with Docker, you can use Kubernetes or others to automate deployment on one or more compute nodes in the cluster. In Kubernetes, interconnections between a set of containers are managed by defining Kubernetes services. As demand for individual containers increases or decreases, Kubernetes can start more or stop some container pods as needed using its feature called replication controller.

Kubernetes gives you a capability to easily add new features to your application, manage system resources and ship your applications from development to production effortlessly. It has a flexible plugin-architecture and provides a convenient pathway to a hybrid cloud implementation. To conclude this section, many organizations favor the Kubernetes

framework because it is highly portable and provides a smooth migration path for legacy applications. Although containers will never be and are not designed to be the single solution to all enterprise workloads, they are a smart way to accelerate development, deployment, and scaling of cloud-native workloads with the help of tools like Kubernetes.

KEY FEATURES OF KUBERNETES

- **Extensibility**

This is the ability of a tool to allow an extension of its capacity/capabilities without serious infrastructure changes. Users can freely extend and add services. This means users can easily add their own features such as security updates, conduct server hardening or other custom features.

- **Portability**

In its broadest sense, this means, the ability of an application to be moved from one machine to the other. This means package can run anywhere. Additionally, you could be running your application on google cloud computer and later along the way get interested in using IBM watson services or you use a cluster of raspberry PI in your backyard. The application-centric nature of Kubernetes allows you to package your app once and enjoy seamless migration from one platform to the other.

- **Self-healing**

Kubernetes offers application resilience through operations it initiates such as auto start, useful when an app crash, auto-replication of containers and scales automatically depending on traffic. Through service discovery, Kubernetes can learn the health of application process by evaluating the main process and exit codes among others. Kubernetes healing property allows it to respond effectively.

- **Load balancing**

Kubernetes optimizes the tasks on demand by making them available and avoids undue strain on the resources. In the context of Kubernetes, we have two types of Load balancers – Internal and external load balancer. The creation of a load balancer is asynchronous process, information about provisioned load balancer is published in the Service's status. loadBalancer.

Traffic coming from the external load balancer is directed at the backend pods. In most cases, external load balancer is created with user-specified load balancer IP address. If no IP address is specified, an ephemeral IP will be assigned to the load balancer.

- **Automated deployment and even replication of containers**

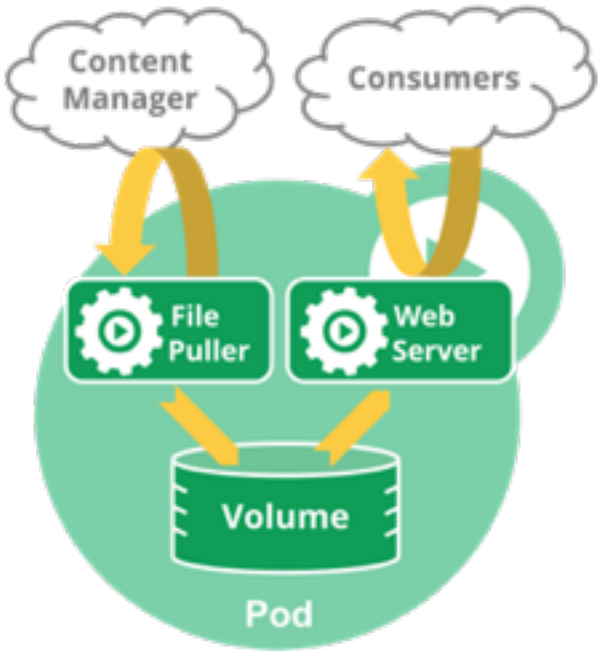
WORK UNITS OF KUBERNETES/

- **Cluster**

These are the nodes or the collection of virtual machines or bare-metal servers which provide the resources that Kubernetes uses to run applications.

- **Pods**

Pods are the smallest units of Kubernetes. A pod can be a single or a group of containers that work together. Generally, pods are relatively tightly coupled. A canonical example is pulling and serving some files as shown in the picture below.



It doesn't make sense to pull the files if you're not serving them and it doesn't make sense to serve them if you haven't pulled them.

Application containers in a pod are in an isolated environment with resource constraints. They all share network space, volumes, cgroups and Linux namespaces. All containers within a pod share an IP address and port space, hence they can find each other via 127.0.0.1 (localhost). They can as well communicate with each other via standard inter-process communications, e.g. SystemV semaphores/POSIX shared memory. Since they are co-located, they are always scheduled together.

When pods are being created, they are assigned a unique ID (UID), and scheduled to run on nodes until they are terminated or deleted. If a node dies, pods that were scheduled to that node are deleted after a timeout period.

- **Labels**

These are key/value pairs attached to objects like pods. When containers need to be managed as a group, they are given tags called labels.
This can allow them to be exposed to the outside to offer services. A replication controller defined next gives the same label to all containers developed from its templates. Labels make it easy for administration and management of services.

Labels are attached to objects at creation time and can be modified at any time. Each set of key/value must be unique for a given object. Unlike names and UIDs, labels do not provide uniqueness, hence many objects can carry the same label(s).

The client or user identifies a set of objects using a label selector. The label selector can be defined as the core grouping primitive in Kubernetes.

Note: Within a namespace, there should be no overlap of the label collectors that belong to two controllers.

- **Services**

A service is an abstraction that defines a logical set of pods and access policy. Services include load balancers services for other containers. Pods performing a similar function are grouped together and represent one entity. If a certain process or application needs a service, a single access point grants it a scalable backend which can be easily replicated making

it optimum and fast. Service can be defined as an abstraction on top of a number of pods.

A Kubernetes service deployment has, at least, two parts. A replication controller and a service. The replication controller defines how many instances should be running, the Container image to use, and a name to identify the service. Additional options can be utilized for configuration and discovery.

- **Replication Controller**

A Replication Controller ensures that a specified number of pod replicas are running at any one time. It defines pods that are to be scaled horizontally. Pods that have been completely defined are provided as templates which are then added with what the new replication should have. It is the responsibility of Replication controller to make sure that a pod or a homogeneous set of pods is always up and available.

Replication Controller supervises multiple pods across multiple nodes. Pods are automatically replaced, deleted or terminated if they fail. As an example, pods are re-created on a node after disruptive maintenance such as a kernel upgrade. If the number of pods is too few, Replication Controller starts more pods. If there are too many pods, extra pods are terminated.

COMPONENTS OF KUBERNETES

The diagram on the next page gives a representation of the components discussed above.

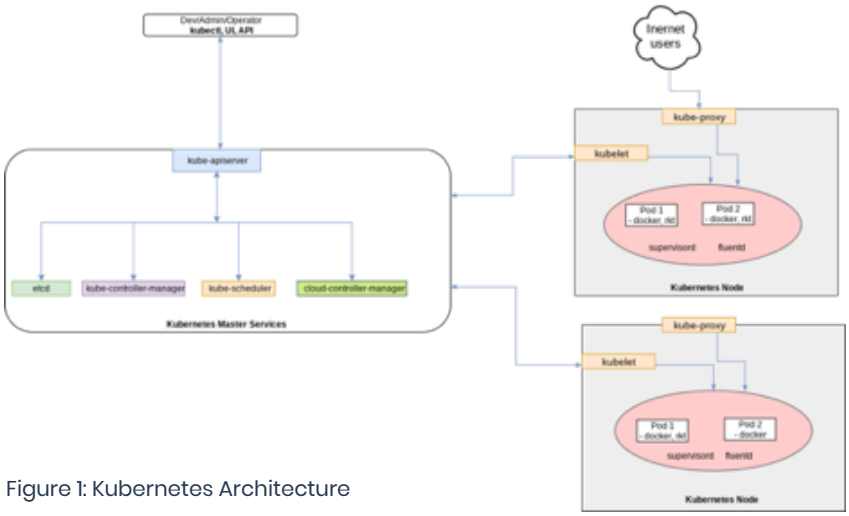


Figure 1: Kubernetes Architecture

Each of these Kubernetes components and how they work is covered in the next table. Note that it's broken into two parts – Kubernetes Master and Kubernetes Node. For Kubernetes Node to function in coordination with master services, there exist control plane within the Master Node.

Kubernetes Master Component	Function of each component
kubectl	• This is a command line interface which enables you to run commands against Kubernetes cluster(s) • Each command that you run with kubectl performs an operation on one or more resources. • Examples of resource types are jobs, nodes, pods, services, endpoints e.t.c
Etcd	• This is a highly available distributed key-value store that's used to store shared configurations and for service discovery • Kubernetes uses etcd to store its API objects and as an interface to register watchers on specific nodes for reliable • For high availability and durability in production environments, you need to run etcd as a multi-node cluster • It is recommended to run etcd as a cluster of odd members (n/2)+1, where n is the number of nodes. For any odd-sized cluster, adding one node will always increase the number of nodes • An etcd cluster needs a majority of nodes, a quorum, to agree on updates to the cluster state. necessary for quorum • You can also run etcd cluster in front of a load balancer.

kube-apiserver	• This service provides an API for orchestrating Kubernetes cluster • It provides the frontend to the shared state of the cluster and services all REST operations against the cluster.
kube-controller-manager	• The kube-controller-manager service regulates the state of Kubernetes cluster. • It does the watching of cluster's shared state through the using apiserver component/service. • It ensures cluster is operating within the desired state
kube-scheduler	• Kube-scheduler does the management of pods lifecycle. This includes deployment of configured pods, deletion and termination. • Is also gathers resources information from all cluster Nodes. • It works closely with controller manager to deploy Kubernetes objects in the cluster depending on the resources available..
Cloud-controller-manager	• This was introduced in 1.6 release version of Kubernetes • It is the future of integrating Kubernetes with any cloud • This is to enable cloud providers develop their own features independently from the core Kubernetes cycles. • The cloud-controller manager provided in the core Kubernetes utilize same cloud libraries as kube-controller-manager • As of v1.8, the cloud controller manager can implement service controller, route controller, node controller and PersistentVolumeLabelAdmission Controller.

Table 1: Kubernetes Master Services

Kubernetes Node Component	Role
Kube-proxy	• This service acts as a network proxy and does load balancing of service running on a single worker node. • Kube-proxy usually runs on each node in the cluster. • It watches the master for Service and Endpoints addition/removal and does load balancing through simple UDP, TCP stream forwarding and round-robin across a set of backend services without the clients knowing anything about Kubernetes or Services or Pods. • Kube-proxy is also responsible for implementing a form of virtual IP for services of types other than ExternalName.
Kubelet	• This is the primary node agent running on each node in the cluster • It gets the configuration of a pod in YAML/ JSON format from the apiserver and ensure that the containers described in those configurations are running and in healthy state. • It doesn't manage containers which were created outside Kubernetes
Supervisord	• Supervisor is a client/server system that allows its users to monitor and control a number of processes on UNIX-like operating systems. • In Kubernetes, supervisord make sure container engine and kubelet are always in running state.
Container Engine – Rkt, docker, e.t.c	• These runs the configured pods on worker nodes • It does downloading of container images and acts as runtime environment for containers

Table 2: Kubernetes Node Components

Kubernetes Concepts

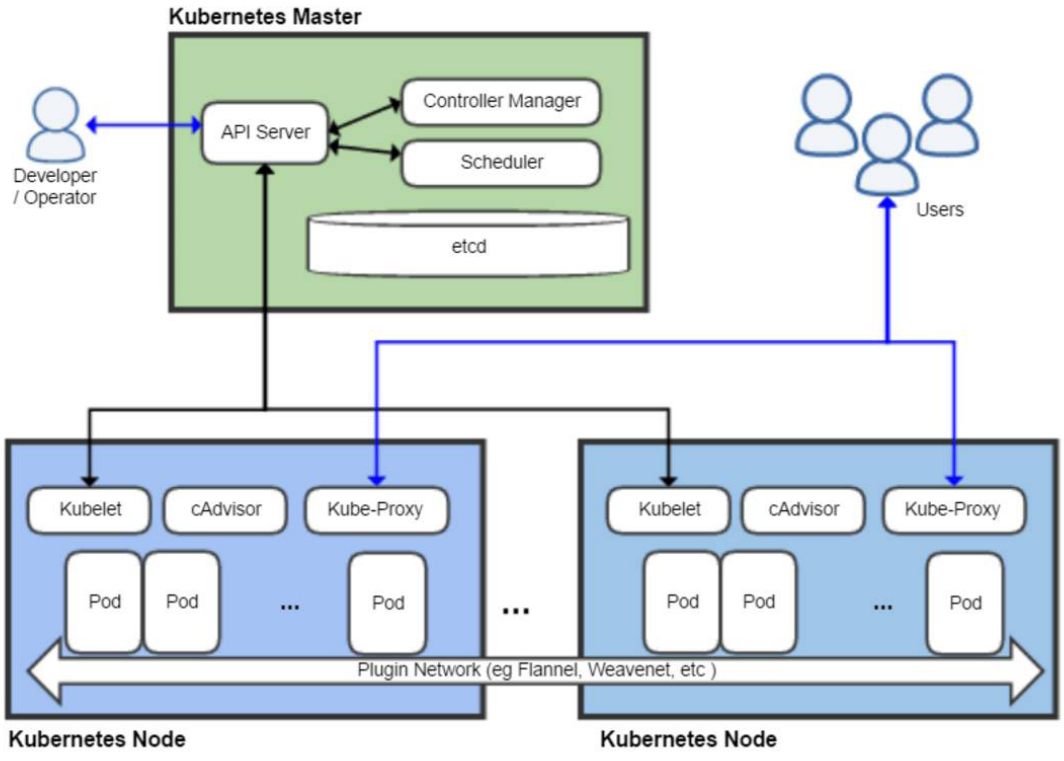
- Pods
- Controllers

Deploying Kubernetes Manually

- Install Docker Engine on Ubuntu
- Installing Kubernetes on Ubuntu
- Installing etcd 2.0 on Ubuntu
- Installing Addons
- Downloading Kubernetes Docker Images

Kubernetes Concepts

To fully understand Kubernetes operations, you'll need a good foundation on the basics of pods and controllers. We'll refer to the diagram below while explaining these concepts.



Pods

In Kubernetes, a Pod is the smallest deployable object. It is the smallest building unit representing a running process on your cluster. A Pod can run a single container or multiple containers that need to run together.

A Pod can also be defined as a group of containers that share resources like file systems, kernel namespaces, and an IP address.

A pod encapsulates the following pieces

- Application container; single or many containers
- A unique network IP address; each pod has an IP address
- Storage resources; All containers in a pod share same storage
- Options governing how containers should run

A single instance of an application is Pod. This instance of an application can be run on a single container or on a small number of containers that share resources and are tightly coupled. Pods support a number of container runtime environments though docker is the most common in Kubernetes.

There are two models of running pods in Kubernetes:

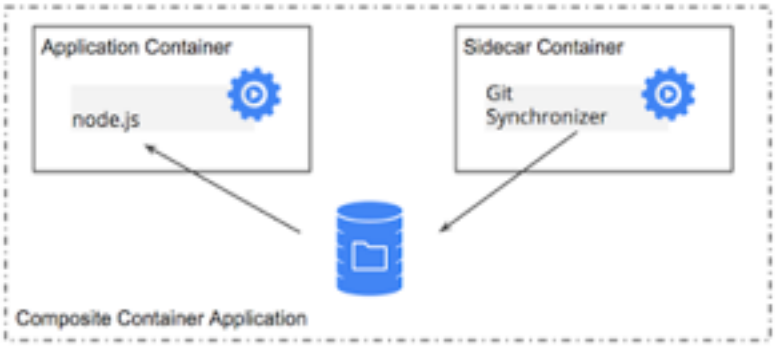
- **One container per pod** - This is the most common model used in Kubernetes. In this case, a pod is a wrapper around a single container. Kubernetes then manage pods instead of directly interacting with individual containers.

- **Multiple containers per pod**: In this model, a pod encapsulates an application that runs on a multiple co-located containers that share resources and are tightly coupled. These co-located containers might form one container that serves files from a shared volume to the public while one container tracks and updates changes of these files.

When talking about pods in Kubernetes, there are different types of containers that you need to know:

Sidecar containers

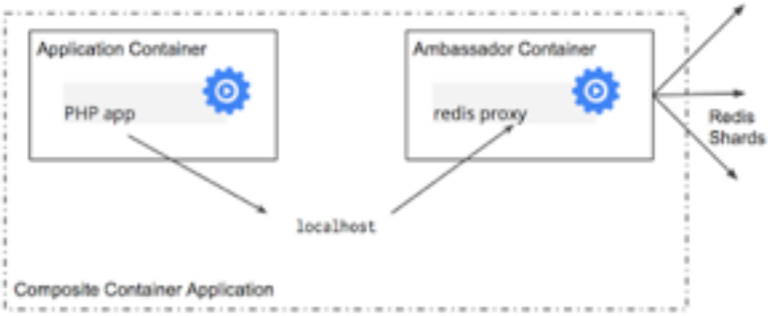
- These are containers which assist the main container. They take main container better in its functionalities.



From this diagram, the sidebar container does pulling of updates from git and application controller then serve these files on application server.

Ambassador containers:

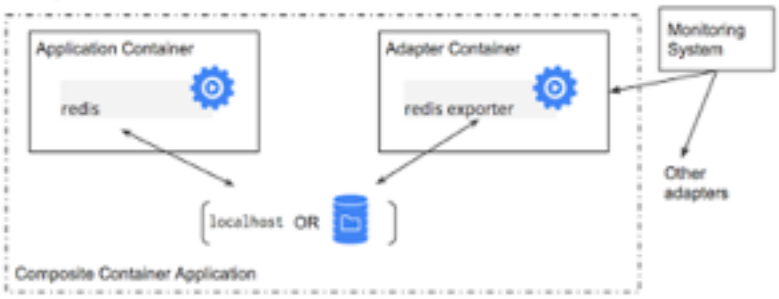
- These are containers which proxy a local connection to the world.



As shown above, the ambassador container runs Redis proxy service. This connects to application container via localhost, but the proxy make the application accessible from outside.

Adapter containers:

- The main work of this is to standardize and normalize output.



How Pods manage multiple Containers

By design, Pods support multiple cooperating containers which work as a single unit of service. The containers in a pod are automatically co-located and co-scheduled on the same virtual machine or physical node in the cluster. All containers in the same pod communicate with each other and can share resources and dependencies. They coordinate on when and how they get terminated. There are two kinds of shared resources provided by Pods for constituent containers, these are:

Storage

An administrator will configure a set of shared volumes that are dedicated to a pod. This shared volume is not bound to a container for it to have persistent data storage. All containers in a Pod access shared volumes and all data in those volumes. In this way, if one container in a Pod is destroyed and another one created, the new container will have access to the shared data store and resume operation on that data.

Networking

In Kubernetes, each Pod is assigned a unique IP address. All containers in the same Pod will share network namespace- IP address and ports. Containers on a Pod are able to communicate with each other through the localhost. When containers in a Pod need to reach the outside, they have to coordinate how they use shared network resources.

Pods creation

A controller can make your task easier by creating and managing multiple Pods for you using a Pod Template you provide. Kubernetes controller also manage replication, rollout and self-healing features. If a Node fails, the Controller can schedule the creation of identical Pod on a different Node.

Pods Lifetime

Pods are mostly managed using a Controller and they are ephemeral entities. When a Pod is created, it is scheduled to run on a Node in the cluster. The Pod will keep running in that specific Node until the parent process is terminated – end of lifecycle, then the Pod object is deleted

Pods by themselves do not self-heal. A pod is deleted if a Node it was running on fails or if the scheduling operation itself fails. If a Pod is evicted for lack of resources, it will be removed as well.

Controllers

The major controller component is ReplicationController which work to ensure that a specified number of pod replicas are running at any one time. It makes sure that a pod or a homogeneous set of pods is always up and available.

When there are too many pods, the ReplicationController will terminate the extra pods. If the number of pods is too low, ReplicationController will start more pods. It does this using application-provided metrics like CPU utilization. Note that Horizontal Pod Autoscaling does not apply to objects that can't be scaled, for example, DaemonSet. Advantage of having pods maintained by ReplicationController is that if a pod fails for any reason, it will automatically create another pod to replace failing one. For this reason, it is always recommended to use ReplicationController even if you have only one pod.

ReplicationController can supervise multiple pods across multiple nodes. Have a look at the below ReplicationController configuration which runs three copies of the caddy web server.

```
$ cat replication.yml

apiVersion: v1
kind: ReplicationController
metadata:
  name: caddy
spec:
  replicas: 4
  selector:
    app: caddy
  template:
    metadata:
      name: caddy
      labels:
        app: caddy
    spec:
      containers:
      - name: caddy
        image: caddy
        ports:
        - containerPort: 80
```

From above code snippet, you can see that we have specified that four copies of caddy web server be created. Container image to be used is caddy and port exposed on the container is 80

Create replicas by running the command:

```
$ kubectl create -f ./replication.yaml

replicationcontroller "caddy" created
```

Give it some seconds to pull image and create container, then check for status:

```
$ kubectl describe replicationcontrollers/caddy

---
Pods Status:    4 Running / 0 Waiting / 0 Succeeded / 0 Failed
```

If you would like to list all the pods that belong to the ReplicationController in a machine readable form, run:

```
$ pods=`kubectl get pods --selector=app=nginx\
 --output=jsonpath={.items..metadata.name}`

$ echo $pods
```

Rescheduling

ReplicationController makes it easy to do rescheduling. You can just change the value of replicas and redeploy, the specified number of replicas will be scheduled for creation accordingly. ReplicationController always ensure that the specified number of pods exists, even in the event of node failure or pod termination

Scaling

You can easily scale the number of replicas up and down using auto-scaling control agent or through manual process. The only change required is on the number of replicas. Please note that Horizontal Pod Autoscaling does not apply to objects that can't be scaled, for example, DaemonSet.

Rolling updates

ReplicatioController facilitates rolling updates to a service by replacing pods one-by-one. To achieve this, you use a Deployment which in turn uses a ReplicaSet. Replication controller is able to update one pod at a time to avoid any service downtime. The command used is kubectl rolling-update. It is always recommended to use Deployment which is a higher-level controller that automates rolling updates of applications declaratively.

4. Deploying Kubernetes Manually

In this section, we're going to cover steps used to install Kubernetes on CentOS and Ubuntu Linux base operating systems. There are many ways to deploy Kubernetes, one of them is manual deployment, and second method is automated deployment with configuration management tools like Ansible, Puppet or Chef.

Manual deployment of Kubernetes includes building different components of Kubernetes one by one to create a working Kubernetes cluster.

In our Lab, we'll setup one Kubernetes master and two Kubernetes nodes. This Lab is done on VirtualBox. These three virtual machines will be created using vagrant. Vagrant is a software applications available on Windows, Linux and Mac which allows you to easily build and maintain portable virtual software development environments.

Prerequisites:

1. Install VirtualBox
2. Install Vagrant
3. Spin up three VMs

Install VirtualBox:

VirtualBox installation varies depending on your base operating system. Refer to official documentation for your specific OS. If you're using Ubuntu 16.x, use the following commands to get latest version of VirtualBox.

```
# echo deb http://download.virtualbox.org/virtualbox/debian yakkety contrib > /etc/apt/sources.list.d/virtualbox.list
# wget -q https://www.virtualbox.org/download/oracle_vbox_2016.asc -O- | sudo apt-key add -
# wget -q https://www.virtualbox.org/download/oracle_vbox.asc -O- | sudo apt-key add -
$ sudo apt-get update
$ sudo apt-get install virtualbox
```

Install Vagrant

Vagrant is an open-source software product for building and maintaining portable virtual software development environments, e.g. for VirtualBox. It makes creation of Virtual Machines easier. If you don't already have Vagrant, install using the following commands:

```
$ sudo apt-get update
$ sudo apt-get install vagrant
```

After successfully installing Vagrant, We can proceed to create three VMs needed for this Lab. Initialize vagrant environment using below commands:

```
$ mkdir kubernetes_lab
$ cd kubernetes_lab
$ vim Vagrantfile
```

If you don't have ssh key, you can generate using command below:

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/jmutai/.ssh/id_rsa): id_rsa
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in id_rsa.
Your public key has been saved in id_rsa.pub.
The key fingerprint is:
SHA256:8F2ObfrwvIa4/rn3oCHjnx5FgEsxVH/MJP1pf17mgt4 jmutai@
dev.jmtai.com
The key's randomart image is:
+---[RSA 2048]----+
|      .++o ...  |
|       o. o =.  |
|     .. .  + +..|
|      o.. * . o.|
|       S o = . .|
|          +    =|
|       o.=... +o|
|      ..o.@+o. o|
|      .+=O=*oE. |
+----[SHA256]-----+
```

By default generated ssh keys will be located under $HOME/.ssh directory.

Now that you have ssh keys that we'll use to ssh to the VMs, it is time to write Vagrantfile used to automatically bring the three VMs up. Vagrantfile uses ruby programming language syntax to define parameters. Below is a sample Vagrantfile contents used for this Lab.

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
# All Vagrant configuration is done below. The "2" in Vagrant.
configure
```

```
# configures the configuration version (we support older styles for
# backwards compatibility). Please don't change it unless you know
what
# you're doing.
Vagrant.configure("2") do |config|
 config.vm.box = "ubuntu/xenial64"
 config.vm.define "kubernetes-master"  do |web|
  web.vm.network "public_network", ip: "192.168.60.2"
  web.vm.hostname="kubernetes-master"
 end
 config.vm.define "kubernetes-node-01" do |web|
  web.vm.network "public_network", ip: "192.168.60.3"
  web.vm.hostname="kubernetes-node-01"
 end
 config.vm.define "kubernetes-node-02" do |web|
  web.vm.network "public_network", ip: "192.168.60.4"
  web.vm.hostname="kubernetes-node-02"
 end
end
```

Once you have the file saved as Vagrantfile. Now create virtual machines using from the Vagrantfile. Note that you need to be on same directory as Vagrantfile before running the command shown below:

```
$ vagrant up
Bringing machine 'kubernetes-master' up with 'virtualbox' provider...
Bringing machine 'kubernetes-node-01' up with 'virtualbox'
provider...
Bringing machine 'kubernetes-node-02' up with 'virtualbox'
provider...
==> kubernetes-master: Importing base box 'ubuntu/xenial64'...
==> kubernetes-master: Matching MAC address for NAT networking...
==> kubernetes-master: Checking if box 'ubuntu/xenial64' is up to
date...
```

```
==> kubernetes-master: Setting the name of the VM: kubernetes_lab_
kubernetes-master_1509819157682_272
==> kubernetes-master: Clearing any previously set network
interfaces...
==> kubernetes-master: Preparing network interfaces based on
configuration...
    kubernetes-master: Adapter 1: nat
    kubernetes-master: Adapter 2: bridged
==> kubernetes-master: Forwarding ports...
    kubernetes-master: 22 (guest) => 2222 (host) (adapter 1)
==> kubernetes-master: Running 'pre-boot' VM customizations...
==> kubernetes-master: Booting VM...
==> kubernetes-master: Waiting for machine to boot. This may take a
few minutes...
    kubernetes-master: SSH address: 127.0.0.1:2222
    kubernetes-master: SSH username: ubuntu
    kubernetes-master: SSH auth method: password
    kubernetes-master:
    kubernetes-master: Inserting generated public key within guest...
    kubernetes-master: Removing insecure key from the guest if it's
present...
    kubernetes-master: Key inserted! Disconnecting and reconnecting
using new SSH key...
==> kubernetes-master: Machine booted and ready!
==> kubernetes-master: Checking for guest additions in VM...
    kubernetes-master: The guest additions on this VM do not match
the installed version of
    kubernetes-master: VirtualBox! In most cases this is fine, but in
rare cases it can
    kubernetes-master: prevent things such as shared folders from
working properly. If you see
    kubernetes-master: shared folder errors, please make sure the
guest additions within the
    kubernetes-master: virtual machine match the version of
VirtualBox you have installed on
```

```
    kubernetes-master: your host and reload your VM.
    kubernetes-master:
    kubernetes-master: Guest Additions Version: 5.0.40
    kubernetes-master: VirtualBox Version: 5.2
==> kubernetes-master: Setting hostname...
==> kubernetes-master: Configuring and enabling network
interfaces...
==> kubernetes-master: Mounting shared folders...
    kubernetes-master: /vagrant => /home/jmutai/kubernetes_lab
==> kubernetes-node-01: Importing base box 'ubuntu/xenial64'...
==> kubernetes-node-01: Matching MAC address for NAT
networking...
==> kubernetes-node-01: Checking if box 'ubuntu/xenial64' is up to
date...
==> kubernetes-node-01: Setting the name of the VM: kubernetes_
lab_kubernetes-node-01_1509819210676_63689
==> kubernetes-node-01: Fixed port collision for 22 => 2222. Now on
port 2200.
==> kubernetes-node-01: Clearing any previously set network
interfaces...
==> kubernetes-node-01: Preparing network interfaces based on
configuration...
    kubernetes-node-01: Adapter 1: nat
    kubernetes-node-01: Adapter 2: bridged
==> kubernetes-node-01: Forwarding ports...
    kubernetes-node-01: 22 (guest) => 2200 (host) (adapter 1)
==> kubernetes-node-01: Running 'pre-boot' VM customizations...
==> kubernetes-node-01: Booting VM...
==> kubernetes-node-01: Waiting for machine to boot. This may take
a few minutes...
    kubernetes-node-01: SSH address: 127.0.0.1:2200
    kubernetes-node-01: SSH username: ubuntu
    kubernetes-node-01: SSH auth method: password
    kubernetes-node-01: Warning: Connection reset. Retrying...
    kubernetes-node-01: Warning: Authentication failure. Retrying...
```

```
    kubernetes-node-01:
    kubernetes-node-01: Inserting generated public key within guest...
    kubernetes-node-01: Removing insecure key from the guest if it's
present...
    kubernetes-node-01: Key inserted! Disconnecting and reconnecting
using new SSH key...
==> kubernetes-node-01: Machine booted and ready!
==> kubernetes-node-01: Checking for guest additions in VM...
    kubernetes-node-01: The guest additions on this VM do not match
the installed version of
    kubernetes-node-01: VirtualBox! In most cases this is fine, but in
rare cases it can
    kubernetes-node-01: prevent things such as shared folders from
working properly. If you see
    kubernetes-node-01: shared folder errors, please make sure the
guest additions within the
    kubernetes-node-01: virtual machine match the version of
VirtualBox you have installed on
    kubernetes-node-01: your host and reload your VM.
    kubernetes-node-01:
    kubernetes-node-01: Guest Additions Version: 5.0.40
    kubernetes-node-01: VirtualBox Version: 5.2
==> kubernetes-node-01: Setting hostname...
==> kubernetes-node-01: Configuring and enabling network
interfaces...
==> kubernetes-node-01: Mounting shared folders...
    kubernetes-node-01: /vagrant => /home/jmutai/kubernetes_lab
==> kubernetes-node-02: Importing base box 'ubuntu/xenial64'...
==> kubernetes-node-02: Matching MAC address for NAT
networking...
==> kubernetes-node-02: Checking if box 'ubuntu/xenial64' is up to
date...
==> kubernetes-node-02: Setting the name of the VM: kubernetes_
lab_kubernetes-node-02_1509819267475_56994
==> kubernetes-node-02: Fixed port collision for 22 => 2222. Now on
port 2201.
```

```
==> kubernetes-node-02: Clearing any previously set network interfaces...
==> kubernetes-node-02: Preparing network interfaces based on configuration...
    kubernetes-node-02: Adapter 1: nat
    kubernetes-node-02: Adapter 2: bridged
==> kubernetes-node-02: Forwarding ports...
    kubernetes-node-02: 22 (guest) => 2201 (host) (adapter 1)
==> kubernetes-node-02: Running 'pre-boot' VM customizations...
==> kubernetes-node-02: Booting VM...
==> kubernetes-node-02: Waiting for machine to boot. This may take a few minutes...
    kubernetes-node-02: SSH address: 127.0.0.1:2201
    kubernetes-node-02: SSH username: ubuntu
    kubernetes-node-02: SSH auth method: password
    kubernetes-node-02:
    kubernetes-node-02: Inserting generated public key within guest...
    kubernetes-node-02: Removing insecure key from the guest if it's present...
    kubernetes-node-02: Key inserted! Disconnecting and reconnecting using new SSH key...
==> kubernetes-node-02: Machine booted and ready!
==> kubernetes-node-02: Checking for guest additions in VM...
    kubernetes-node-02: The guest additions on this VM do not match the installed version of
    kubernetes-node-02: VirtualBox! In most cases this is fine, but in rare cases it can
    kubernetes-node-02: prevent things such as shared folders from working properly. If you see
    kubernetes-node-02: shared folder errors, please make sure the guest additions within the
    kubernetes-node-02: virtual machine match the version of VirtualBox you have installed on
    kubernetes-node-02: your host and reload your VM.
    kubernetes-node-02:
    kubernetes-node-02: Guest Additions Version: 5.0.40
```

```
    kubernetes-node-02: VirtualBox Version: 5.2
==> kubernetes-node-02: Setting hostname...
==> kubernetes-node-02: Configuring and enabling network
interfaces...
==> kubernetes-node-02: Mounting shared folders...
    kubernetes-node-02: /vagrant => /home/jmutai/kubernetes_lab
```

The command above will download Ubuntu Xenial vagrant image and create three Virtual Machiness with specified names - kubernetes-master, kubernetes-node-01 and kubernetes-node-02. All these VMs will be on the same subnet 192.168.60.0/24.

Confirm that the VMs were successfully created:

```
$ vagrant status

Current machine states:

kubernetes-master       running (virtualbox)
kubernetes-node-01      running (virtualbox)
kubernetes-node-02      running (virtualbox)

This environment represents multiple VMs. The VMs are all listed
above with their current state. For more information about a specific
VM, run `vagrant status NAME`.
```

Now ssh to the Kubernetes master node and update apt cache, then do system upgrade:

```
$ vagrant ssh kubernetes-master
sudo su -
apt-get update
apt-get upgrade && apt-get dist-upgrade
```

Do the same on both Kubernetes nodes - Perform system update and upgrade.

```
$ vagrant ssh kubernetes-node-01
sudo su -
apt-get update && apt-get upgrade && apt-get dist-upgrade

$ vagrant ssh kubernetes-node-02
sudo su -
apt-get update && apt-get upgrade && apt-get dist-upgrade
```

Now that Kubernetes master node is ready, let's proceed to install docker engine community edition on this VM.

Install Docker Engine on Ubuntu

Docker is a platform for developers and system administrators to develop, ship, and run applications. One of the key components of Docker is docker engine which is a lightweight and powerful open source containerization technology combined with a workflow for building and containerizing your applications. Kubernetes depends on docker engine to run containers. Though other container runtimes like rkt and lxc can be used, the most mature and popular is docker.

Docker engine can be easily installed on Ubuntu from official apt repositories provided by Docker Inc. Follow steps below to setup Docker repository and install docker engine on Ubuntu host.

1. Update the apt package index:

```
$ sudo apt-get update
```

2. Install packages to allow apt to use a repository over HTTPS:

```
$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    software-properties-common
```

3. Add Docker's official GPG key:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo
apt-key add -
```

4. Set up the stable repository on Ubuntu 16.x

```
$ echo deb [arch=amd64] https://download.docker.com/linux/ubuntu
$(lsb_release -cs) Stable > /etc/apt/sources.list.d/docker.list
```

5. Update the apt package index and install Docker CE.

```
sudo apt-get update && apt-get install docker-ce
```

Another way to install docker is by using script provided at get.docker.com. The scripts attempt to detect your Linux distribution and version and configure your package management system for you. You only need to be root or use sudo to run this script. To install Docker with this script use the commands shown below:

```
$ curl -fsSL get.docker.com -o get-docker.sh
$ sudo sh get-docker.sh
```

If you would like to use Docker as a non-root user, you should now consider adding your user to the "docker" group with something like:

```
$ sudo usermod -aG docker vagrant
```

If you are using different username, remember to replace vagrant with that name. Check version of docker installed using the command below:

```
$ docker version
Client:
 Version:      17.10.0-ce
 API version:  1.33
 Go version:   go1.8.3
 Git commit:   f4ffd25
 Built:        Tue Oct 17 19:04:16 2017
 OS/Arch:      linux/amd64

Server:
 Version:      17.10.0-ce
 API version:  1.33 (minimum version 1.12)
 Go version:   go1.8.3
 Git commit:   f4ffd25
 Built:        Tue Oct 17 19:02:56 2017
 OS/Arch:      linux/amd64
 Experimental: false
```

Now that docker is installed, we can proceed to install Kubernetes on the master node.

Install Docker Engine on Ubuntu

This section provides instructions for installing Kubernetes and setting up a Kubernetes cluster. Kubernetes can run on various platforms; Virtual Machine on cloud, Laptop, Virtual machine on VirtualBox, to bare metal servers. The effort required to setup Kubernetes cluster varies depending on the need for setting up the cluster. Test environment can be done using minikube while for production environment, a number of customizations might be required, for this manual setup will work fine.

Install dependencies:

```
apt-get install -y transport-https
```

Add key for new repository:

```
echo deb http://apt.kubernetes.io/ kubernetes-$(lsb_release -cs) main
> /etc/apt/sources.list.d/kubernetes.list
```

Update apt index:

```
echo deb http://apt.kubernetes.io/ kubernetes-$(lsb_release -cs) main
> /etc/apt/sources.list.d/kubernetes.list
```

Install kubelet, kubeadm kubectl and kubernetes-cni:

```
apt-get install -y kubelet kubeadm kubectl kubernetes-cni
```

Short description of installed packages:

Kubelet: This is the core component of Kubernetes. It is a primary "node agent" that runs on each node and does things like starting pods and containers.

Kubeadm: Used to easily bootstrap a secure Kubernetes cluster.

Kubectl: This is a command line interface for running commands against Kubernetes clusters. This is only used on the master.

Kubernetes-cni: This enables cni network on your machine. CNI stands for Container Networking Interface which is a spec that defines how network drivers should interact with Kubernetes

Once you have all the master components installed, the next step is initialize cluster on the master node. The master is the machine where the "control plane" components run, including etcd (the cluster database) and the API server (which the kubectl CLI communicates with). All of these components run in pods started by kubelet.

Initialize master cluster

On your master node, initialize cluster by running the command:

```
kubeadm init
```

This will download and install the cluster database and "control plane" components. This may take several minutes depending on your internet connection speed. The output from this command will give you the exact command you need to join the nodes to the master, take note of this command:

```
...
Your Kubernetes master has initialized successfully!

To start using your cluster, you need to run (as a regular user):
```

```
  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options
listed at:
  http://kubernetes.io/docs/admin/addons/

You can now join any number of machines by running the following
on each node
as root:

  kubeadm join --token 453a5e.3d2fedeee757cb02 10.0.2.15:6443 --dis-
covery-token-ca-cert-hash sha256:536f72f8c6d48d711358fd7f19cec-
c0b903824a90bf8b3470ef1f7bd34f94892
```

Before you join a cluster, run the commands:

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

On the two Kubernetes nodes, run the following command to join them to the cluster:

```
$ kubeadm join --token 453a5e.3d2fedeee757cb02 10.0.2.15:6443 --dis-
covery-token-ca-cert-hash sha256:536f72f8c6d48d711358fd7f19cec-
c0b903824a90bf8b3470ef1f7bd34f94892
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Remember to replace token id and --discovery-token-ca-cert-hash values with the ones you got from the output of 'kubeadm init' command.

Installing etcd 2.0 on Ubuntu

Etcd 2.0 is a Distributed reliable key-value store for the most critical data of a distributed system. It focuses mainly on being Secure, Simple, Fast and Reliable. It is written in Go programming language and Raft consensus algorithm that manages a highly-available replication log,

There are two ways to install etcd on Ubuntu – One being building it form source, next being using pre-built binary available for download. If you're interested in getting the latest release, consider building it from source.

To build it from source, you'll need installed git and golang, then run:

```
$ git clone https://github.com/coreos/etcd.git
$ cd etcd
$ ./build
$ ./bin/etcd
```

Installing Addons

Here I'll show a number of plugins that you can install to extend Kubernetes functionalities. This list is not exclusive, feel free to add what you feel might help.

Deploy Flannel Pod Network

Flannel is an overlay network provider that can be used with Kubernetes. Let's configure Flannel pod network by first installing in on the master node. Follow steps provided here to get flannel network plugin up and running.

```
sudo kubectl apply -f  https://raw.githubusercontent.com/coreos/
flannel/master/Documentation/kube-flannel.yml
sudo kubectl apply -f https://raw.githubusercontent.com/coreos/
flannel/master/Documentation/kube-flannel-rbac.yml
$ ./build
$ ./bin/etcd
```

For further reading about Flannel, refer to https://github.com/coreos/flannel

Deploy CoreDNS

CoreDNS is a flexible and extensible DNS server written in Go which you can use on Kubernetes setup to manage Pod DNS records. It chains plugins which manage DNS functions such as Kubernetes service discovery, Prometheus metrics or rewriting queries. CoreDNS is able to integrate with Kubernetes via the Kubernetes plugin, or directly with etcd plugin. Mostly you'll either server zone

To deploy CoreDNS on Kubernetes do:

1. Install Golang and git:

```
sudo apt-get install golang git
```

2. Verify Installation

```
# go version
go version go1.6.2 linux/amd64
```

3. Set GOPATH environment variable.

```
mkdir ~/go
export GOPATH=$HOME/go
export PATH=$GOPATH/bin:$PATH
```

4. Compile CoreDNS application using go

```
go get github.com/coredns/coredns
```

Another way to install CoreDNS is from a binary of latest release. To check latest release visit > https://github.com/coredns/coredns/releases

Then download latest version, uncompress and copy binary to /usr/local/bin/

```
apt-get install wget
wget https://github.com/coredns/coredns/releases/download/v0.9.10/coredns_0.9.10_linux_amd64.tgz
tar zxvf coredns_0.9.10_linux_amd64.tgz
cp coredns /usr/local/bin/
```

Test that the binary is copied and working by checking coredns version:

```
# /usr/local/bin/coredns -version
CoreDNS-0.9.10
linux/amd64, go1.9.1, d272e525
```

For more information about CoreDNS, visit official project page on github:

Deploying Kubernetes Dashboard

Kubernetes Dashboard is a general purpose, web-based UI for managing and configuring Kubernetes clusters. This Dashboard aims at making applications running in the cluster easier to manage and troubleshoot. It provides for a secure setup and by default it has minimal set of privileges with access only through https.

To deploy Dashboard execute following command on master node command line interface:

```
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/recommended/kubernetes-dashboard.yaml
```

Before you can start using Kubernetes Dashboard, you'll need to configure the proxy server using kubectl command. This will set url for you which you can use to access the dashboard. Run command below to get proxy configured:

```
$ kubectl proxy
```

After successful execution of this command, dashboard is now accessible on http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/

To learn more about using Kubernetes dashboard, visit Kubernetes Wiki

Getting Kubernetes Images

Now that you have ready environment, let's look at how to download docker images for use with Kubernetes. A docker image is an inert, immutable file that's essentially a snapshot of a container. This contains OS utilities and basic tools required to run an application. In this example, I'll show you how to download busybox docker image:

```
# docker pull busybox
Using default tag: latest
latest: Pulling from library/busybox
0ffadd58f2a6: Pull complete
Digest:  sha256:bbc3a03235220b170ba48a157dd097dd1379299370e1ed-
99ce976df0355d24f0
Status: Downloaded newer image for busybox:latest
```

Confirm that the image has been downloaded:

```
# docker images
REPOSITORY          TAG       IMAGE ID       CREATED        SIZE
ubuntu              xenial    dd6f76d9cc90   12 days ago    122MB
busybox             latest    6ad733544a63   12 days ago    1.13MB
passenger-ruby24    latest    c3f873600e95   5 months ago   640MB
phusion/passenger-ruby24   latest    c3f873600e95   5 months ago   640MB
```

As you can see above, the image has been downloaded successfully. We'll use this image in the next section.

Testing:

Now that we have ready Kubernetes cluster, let's create a simple pod on this cluster. As an example, consider below simple Pod template manifest for a Pod with a container to print a message. Pod configuration file is defined using YAML syntax:

```
$ cat pod.yaml

apiVersion: v1
kind: Pod
metadata:
  name: testapp-pod
  labels:
    app: testapp
spec:
  containers:
  - name: testapp-container
    image: busybox
    command: ['sh', '-c', 'echo This is Kubernetes!! && sleep 3600']
```

Options used:

apiVersion: v1 - Using API version 1 of Kubernetes
kind: Pod - This specify that we want to create a Pod. Other values include Deployment, Job, Service, and so on
metadata: - Here we have to specify the name of the Pod, as well and the label used to identify the pod to Kubernetes
Spec: - Here we are specifying the actual objects that make up the pod. The spec property includes storage volumes, containers and other information that Kubernetes require, as well as properties like whether to restart a container in case it fails e.t.c. In this case, we have minimal definition:

A container name (testapp-container), the image on which it is based (busybox), and the command that will execute on the container upon creation.

Tell Kubernetes to create its contents:

```
$ kubectl create -f pod.yaml
```

Confirm creation by asking for a name of pods:

```
$ kubectl get pods
NAME          READY    STATUS              RESTARTS    AGE
testapp-pod   0/1      ContainerCreating   0           6s
```

After a few seconds, you should see the containers running:

```
$ kubectl get pods
NAME          READY    STATUS              RESTARTS    AGE
testapp-pod   1/1      Running             0           6s
```

Setting up Kubernetes Cluster:

There are many solutions available for setting up Kubernetes cluster for the different environment. To get started with the Kubernetes, Minikube is one of the most preferred options.

Before starting with Minikube lets look into other options:

In independent solutions, minikube and kubeadm are the best options to bootstrap Kubernetes cluster. Both options work well with the development environment. In hosted solutions Google Kubernetes Engine, Azure Container Service, and IBM Cloud Container Service are options. It is the less time-consuming process to launch Kubernetes cluster with hosted solutions.

Kubespray and KOPS(Kubernetes Operations) are community supported tools for bootstrapping the cluster. With Kubespray it is possible to setup multi-master Kubernetes cluster on on-premise and cloud platforms.

Minikube is a tool written in Golang to set up the cluster locally on the machine. It will require virtualization to be enabled for Operating System. Minikube supports Container Network Interface (CNI Plugins), Domain Name System, Kubernetes Dashboard, Ingress for load balancing, Config Maps and Secrets and Container runtime which can be docker or rkt.

If Linux is used then, it will require Virtual Machine Driver to run minikube. Following are steps to install Minikube on Linux:

(Note: Following steps are preferred for Ubuntu 16.04)

Install a Hypervisor:

Check if CPU supports hardware virtualization: (If value > 0 then it supports)

```
$ egrep -c '(vmx|svm)' /proc/cpuinfo
```

Download VirtualBox:

```
$ curl -LO http://download.virtualbox.org/virtualbox/5.2.0/virtualbox-5.2_5.2.0-118431~Ubuntu~xenial_amd64.deb
```

Install using Debian package manager:

```
$ sudodpkg -i virtualbox-5.2_5.2.0-118431-Ubuntu-xenial_amd64.deb
```

For latest version please check https://www.virtualbox.org/ .

Install Kubectl:

Kubectl is the command line utility which interacts with API Server of the Kubernetes.

Download Kubectl Stable Binary:

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

Make binary executable:

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

Move binary to the system path:

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

Check if Kubectl is configured or not:

```
$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
  run-container  Run a particular image on the cluster
  set           Set specific features on objects
...
```

Install Minikube:

```
$curl -Lo minikube https://storage.googleapis.com/minikube/releases/v0.22.3/minikube-linux-amd64 &&chmod +x minikube&&sudo mv minikube /usr/local/bin/
```

For latest version please checkhttps://github.com/kubernetes/minikube/releases.

Verify Installation:

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

Usage:
minikube [command]

Available Commands:
addons          Modify minikube'skubernetesaddons
 completion      Outputs minikube shell completion for the given shell (bash)
config          Modify minikubeconfig
 dashboard       Opens/displays the kubernetes dashboard URL for your local cluster
 delete          Deletes a local kubernetes cluster
docker-env      Sets up dockerenv variables; similar to '$(docker-machine env)'
 get-k8s-versions Gets the list of available kubernetes versions available for minikube
ip              Retrieves the IP address of the running cluster
```

The message indicates that Kubernetes cluster is started with Minikube. Docker should be running on the host machine. Minikube will use default container engine (docker here) to run the app.

Create Kubernetes Cluster through Minikube:

Run following command:

```
$minikube start
```

Verify the Kubernetes Cluster Started:

```
$minikube start
Starting local Kubernetes v1.7.5 cluster...
Starting VM...
Getting VM IP address...
Moving files into cluster...
Setting up certs...
Connecting to cluster...
Setting up kubeconfig...
Starting cluster components...
Kubectl is now configured to use the cluster.
```

If you got this return message then, you successfully started MinikubeKubernetes Cluster locally. Kubernetes API server will be accessed with Kubectl utility.

To verify run following command:

```
$kubectlconfig get-clusters
```

It will list the clusters. You should get the result like:

```
$kubectlconfig get-clusters
NAME
minikube
```

Here Kubectl is successfully configured and Kubernetes cluster will be running on local machine.

Dockerizing the App:

Containerization is the most important aspect to bundle all the dependencies together we called it as a container. The container can be an application container, web container or a database container. The advantage of the containerization is to run app anywhere regardless the dependencies because container bundles it with Images. Container engine is used for communication between containers and underlying kernel of Operating System. Docker is the most popular container engine. There are other container engines like rkt, lxc.

Dockerfile consists the set of instruction to dockerize the application. It bundles all the dependencies.

Here we will use sample NodeJs application. The structure of the basic app will be:

```
app/
-- server.js
-- package.json
```

server.js:

```
const express = require('express');

const app = express();

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

app.listen(3000, function () {
console.log('Example app listening on port 3000!');
});
```

Package.json:

```
{
  "name": "hello-world",
  "version": "1.0.0",
  "description": "Hello world app taken from: https://expressjs.com/en/
starter/hello-world.html",
  "main": "server.js",
  "scripts": {
    "test": "",
    "start": ""
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/borderguru/hello-world.git"
  },
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/borderguru/hello-world/issues"
  },
  "homepage": "https://github.com/borderguru/hello-world#readme",
  "dependencies": {
    "chai": "^4.1.2",
    "express": "^4.15.3",
    "mocha": "^4.0.1",
    "request": "^2.83.0"
  }
}
```

To Dockerize the app, we need to create the Dockerfile:

Dockerfile:

```
# The official image of the node
FROMnode:boron
# ARG Version to define app version
ARG VERSION=1.0.0

# Create app directory inside image
WORKDIR /usr/src/app

# Set required environment variables
ENV NODE_ENV production

# Install app dependencies by copying package.json to image
COPY package.json .

# For npm@5 or later, copy package-lock.json as well
# COPY package.json package-lock.json ./

RUNnpm install

# Bundle app source
COPY . .
# It will start the app on 3000 port of the container
EXPOSE3000
CMD [ "npm", "start" ]
```

This is well-defined dockerfile. Make sure docker should be pre-installed on the machine. If it is not installed then, install via official documentation of the docker.

The Dockerfile consists set of commands which includes making work directory where necessary files will be copied. Installing dependencies using npm. Note the base image node:boron is an official image from NodeJS and it consists stable npm version. After copying all files and installing dependencies exposing 3000 port of the container and the first command will run npm start.

This is sample Dockerfile but it is not limited to just given commands. For more in-depth information please check official guide of the Docker for creating Dockerfile.

To create the images from the Dockerfiledockercli is used. Now the app structure is:

```
app/
--- server.js
--- package.json
--- Dockerfile
```

Run the following command:

```
$ sudodocker build -t helloworld:1.0 .
```

It will build the image and tag it with helloworld:1.0 here 1.0 is the version of the image. If nothing is specified then, the latest version will be chosen.

This will download all dependencies to run the app. After a successful build, check the images.

sudodocker images				
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
helloworld	1.0	c812ebca7a95	About a minute ago	678
node	boron	c0cea7b613ca	11 days ago	661 MB

To run the container docker run command is used. It's necessary to bind the node port to container port.

```
sudodocker run -it -p 3000:3000 helloworld:1.0
```

Here its binding port 3000 for container port 3000.

Check if app is running:

```
$curl localhost:3000
Hello World!
```

Tag the image and push to dockerhub:

$sudodocker images			
REPOSITORY SIZE	TAG	IMAGE ID	CREATED
helloworld	1.0	c812ebca7a95	3 hours ago
678 MB			
node	boron	c0cea7b613ca	11 days ago
661 MB			
$sudodocker tag helloworld:1.0 helloworld:latest			
REPOSITORY SIZE	TAG	IMAGE ID	CREATED
helloworld	1.0	c812ebca7a95	3 hours ago
678 MB			
helloworld	1.0	c812ebca7a95	3 hours ago
678 MB			
helloworld	1.0	c812ebca7a95	3 hours ago
678 MB			
node	boron	c0cea7b613ca	11 days ago
661 MB			

Tagging image to latest to indicate this will be the most recent version of the image. Dockerhub is the central registry for storing docker images. However, there are many other registries available like JFROG, Quay and Amazon ECR.

Login to dockerhub:
(Notice: If you don't know dockerhub then please visithttps://hub.docker.com and create account)

```
$ sudodocker login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username (kubejack): kubejack
Password:
Login Succeeded
```

```
$sudodocker tag helloworld:latestkubejack/helloworld:latest
$sudodocker push kubejack/helloworld:latest
The push refers to a repository [docker.io/kubejack/helloworld]
50c9a7dd83b6: Pushed
94df5aea4989: Pushed
e7158264ab54: Pushed
54e9e4999177: Pushed
7c966bc7c94e: Mounted from library/node
c3031c131aac: Mounted from library/node
7df250eb9e5c: Mounted from library/node
e9591a63b7e4: Mounted from library/node
d752a0310ee4: Mounted from library/node
db64edce4b5b: Mounted from library/node
d5d60fc34309: Mounted from library/node
c01c63c6823d: Mounted from library/node
latest: digest: sha256:952ff7e89547e157630be120126a3e1d8717d45e0d-f72a1f49901c2bcde74250 size: 2838
```

This will be used in Kubernetes manifests. The app is successfully containerized now and images are pushed to the dockerhub.

Writing Kubernetes Manifest Files for Sample App:

For now, the sample app is containerized but to run the app over Kubernetes it will require the Kubernetes Objects.

The following are the Kubernetes objects:

Pod :Kubernetes deployment unit. It wraps single or multiple containers. Multiple containers will share network and storage namespace.

Replication Controller / Replica Sets: Replica Sets is next version of the Replication controller. There is more efficiency while working with Replica Sets using Selectors and Labels. This object will instruct replication controller to maintain the number of replicas. Replication Controller / Replica Sets layered on top of pods.

Deployment: Deployment is layered on top of Replica Sets / Replication Controller for automated rollouts and rollbacks. It's the most important use case while in production we deploy the different version of the application.

DaemonSets : Daemon sets are used for running the pods on each Kubernetes node. This usually used for monitoring and logging.

Services: Services are the backbone of the Kubernetes service discovery mechanism. It's the internal service discover implemented by the Kubernetes. Service act as the proxy for replicas of the pods and redirect the traffic to the appropriate pod. That means it acts as the load balancer for the pods too.

Here we will be using deployment and service object to create kubernetes manifests. For now, create Kubernetes directory.

```
mkdirkubernetes
```

Create deployment.yml and service.yml files

```
cd kubernetes
touch deployment.yml
touch service.yml
```

Now the structure of the directory will be:

```
app/
--- server.js
--- package.json
--- Dockerfile
kubernetes/
--- deployment.yml
--- service.yml
```

Kubernetes manifests are plain yaml files which will define the desired state of the cluster.

deployment.yml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: hello-world
  labels:
    app: hello-world
ver: v1
spec:
  replicas: 10
  selector:
matchLabels:
      app: hello-world
ver: v1
```

```
  template:
    metadata:
      labels:
        app: hello-world
ver: v1
    spec:
      containers:
      - name: hello-world
        image: kubejack/helloworld:latest
imagePullPolicy: Always
        ports:
        - containerPort: 3000
```

Let's understand the deployments.

Deployment Object:

deployment.yml:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: hello-world
  labels:
    app: hello-world
ver: v1
```

Kubernetes API Server is responsible for all tasks related to Kubernetes. Here the deployment is the object of the Kubernetes. apiVersion indicates which API it is using. apiVersion consists two different type of version. It consists the extensions and apps.

Kind indicates the type of kubernetes object it can be pod, deployment, service etc. Metadata is the naming convention for the kubernetes object. So this deployment will be known by name: hello-world.

Labels are important for service discovery. Labels are attached to the kubernetes object. And all operations on API server are with Labels.

Replica Set Object Specification:

deployment.yml

```
spec:
  replicas: 10
  selector:
matchLabels:
     app: hello-world
ver: v1
 template:
   metadata:
     labels:
       app: hello-world
ver: v1
```

We discussed previously that deployment is layered above the Replication Controller / ReplicaSets. Here it is replica set.ReplicaSet Object is the specification for the Replica Sets. It indicates numbers of replicas. That means there should be 10 replicas of Replica Sets object. It will select the deployment by a selector which will use matchLabelsto match with pods.

ReplicaSets will create pod-template-hash and recognize pods with Pod labels. These are given in template. That means Labels are used to group the pods. And selectors have used to group right pods according to a label.

Pod Object Specification:

deployment.yml

```
    spec:
      containers:
      - name: hello-world
        image: kubejack/helloworld:latest
imagePullPolicy: Always
        ports:
        - containerPort: 3000
```

The Pod Object Specification wraps one or multiple containers. It's always the best practice to wrap one container within one pod. Multiple container's per pod will create the more complexity in terms of network and storage namespace.

The Pod specification will consist the container specification. It means this specification will give information about image name, container name, container ports to be exposed and what will be pull policy for the image.

imagePullPolicy indicates when the image should get pulled and can be "Always" to make sure updated image is pulled each time.

To group pods and load balance between them Kubernetes internal service discovery mechanism is used.

service.yml

```
apiVersion: v1
kind: Service
metadata:
  name: hello-world-svc
  labels:
    name: hello-world-svc
spec:
  ports:
    - port: 80
targetPort: 3000
      protocol: TCP
  selector:
    app: hello-world
ver: v1
```

Services are important to proxy the replicas of ReplicaSets. In above service.yml file apiVersion is used which indicates the Kubernetes API version to be used. Here its v1. It was not same in deployment because v1 is not supporting deployment object. Again the labels and metadata for the services are used to uniquely identify the service within Kubernetes cluster.

In the spec, it's possible to expose the port of the node. The exposed port will be mapped to the container port. Here targetPort is the port of the container which is mapped with node port 80.TCP and UDP protocols are supported.

This service is not exposed publicly. To expose it with cloud provider type: Loadbalanceris used.

The most important is Selector. It will select the group of pods which will match all labels. Even if any pod consist more than, specified labels then it should match. It is not the same case with the selector, if there is any label missed in selector but not in the pod then service will just ignore it.
Labels and Selectors are good ways to maintain version and to rollback and rollout the updates.

Understanding the Kubectl Utility

In previous sections, we have discussed the master components. The API Server plays the important role of the master.

Everything can be done using API Server only. There are different options for calling REST API's, the User interface of the Kubernetes and Kubectl.

Kubectl is the command line utility to interact with Kubernetes cluster. We created the Kubernetes Cluster through Minikube. Kubectlconfig is used to configure Kubectl for Kubernetes Cluster. To verify the Kubectl is connected to API Server:

```
$ cat ~/.kube/config
apiVersion: v1
clusters:
- cluster:
    certificate-authority: /home/ut/.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/ut/.minikube/client.crt
    client-key: /home/ut/.minikube/client.key
```

Kubectl is authenticated to interact with Kubernetes Cluster. Kubectl can manage everything for Kubernetes Cluster.

Kubectl can,
Create the Kubernetes Object:

```
$kubectl create -f deployment.yml
```

Apply the changes to the Kubernetes Object:

```
$kubectl apply -f deployment.yml
```

Get the deployments

```
$kubectl get deployments
```

Get the pods:

```
$kubectl get pods
```

Get the namespaces

```
$kubectl get namespaces
```

Get the replica sets:

```
$kubectl get rs
```

Describe the Kubernetes Object:

```
$kubectl describe deployment hello-world
```

Delete the Kubernetes Object

```
$kubectl delete -f deployment.yml
```

To get the all pods from namespace kube-system

```
$kubectl get po --namespace kube-system
```

All these are few examples of the Kubectl utility. But there are more advanced use cases like scaling with kubectl. All these are imperative methods.

Imperative methods are good at the time of troubleshooting or in the development environment. It's always the best practice to have declarative methods for creating YAML files.

To create the kubernetes object Kubectl cli will be used for the communication with API Server.

Create deployment object:

```
$kubectl create -f deployment.yml
```

Create service object:

```
$kubectl create -f service.yml
```

It will create the 10 Pods, 1 ReplicaSet and 1 Deployment object in the default namespace.

Get deployment:

```
$kubectl get deploy
NAME          DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
hello-world   10        10        10           10          1h
```

Get replica set:

```
kubectl get rs
NAME                    DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
hello-world-493621601   10        10        10           10          1h
```

493621601 is the hash value added over deployment

Get pods:

```
$kubectl get po
NAME                      READY  STATUS   RESTARTS  AGE
hello-world-493621601     1/1    Running  0         1h
hello-world-493621601-2dq3c 1/1  Running  0         1h
hello-world-493621601-8jnjw 1/1  Running  0         1h
hello-world-493621601-g5sfl 1/1  Running  0         1h
hello-world-493621601-j3jjt 1/1  Running  0         1h
hello-world-493621601-rb7gr 1/1  Running  0         1h
hello-world-493621601-s5qp9 1/1  Running  0         1h
hello-world-493621601-v086d 1/1  Running  0         1h
hello-world-493621601-xkhhs 1/1  Running  0         1h
hello-world-493621601-ztp32 1/1  Running  0         1h
```

Again the more hash values are added over the replica set. The status indicates all pods are in running state. That means desired state of the cluster is met.

Get the service object and describe it:

```
$kubectl get svc
NAME          CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
hello-world-svc 10.0.0.131  <pending>     80:30912/TCP 1h
kubernetes    10.0.0.1     <none>        443/TCP    2h
```

Describe service:

```
$kubectl describe svc hello-world-svc
Name:                   hello-world-svc
Namespace:              default
Labels:                 name=hello-world-svc
Annotations:            kubectl.kubernetes.io/last-applied-
```

```
configuration={"apiVersion":"v1","kind":"Service","metadata":{"anno
tations":{},"labels":{"name":"hello-world-svc"},"name":"hello-world-
svc","namespace":"default"},"s...
Selector:          app=hello-world,ver=v1
Type:                    LoadBalancer
IP:                10.0.0.131
Port:              <unset>  80/TCP
NodePort:          <unset>  30912/TCP
Endpoints:               172.17.0.10:3000,172.17.0.12:3000,172.17.0.2:300
0 + 7
more...
Session Affinity:  None
Events:                  <none>
```

Endpoints are unique IP's of the Pods. Services group all Pods with the IP's. Kubernetes Services plays the role of Proxy.

Exposing the Service locally through Minikube:

To access the app it's necessary to get the IP and port from the Minikube.

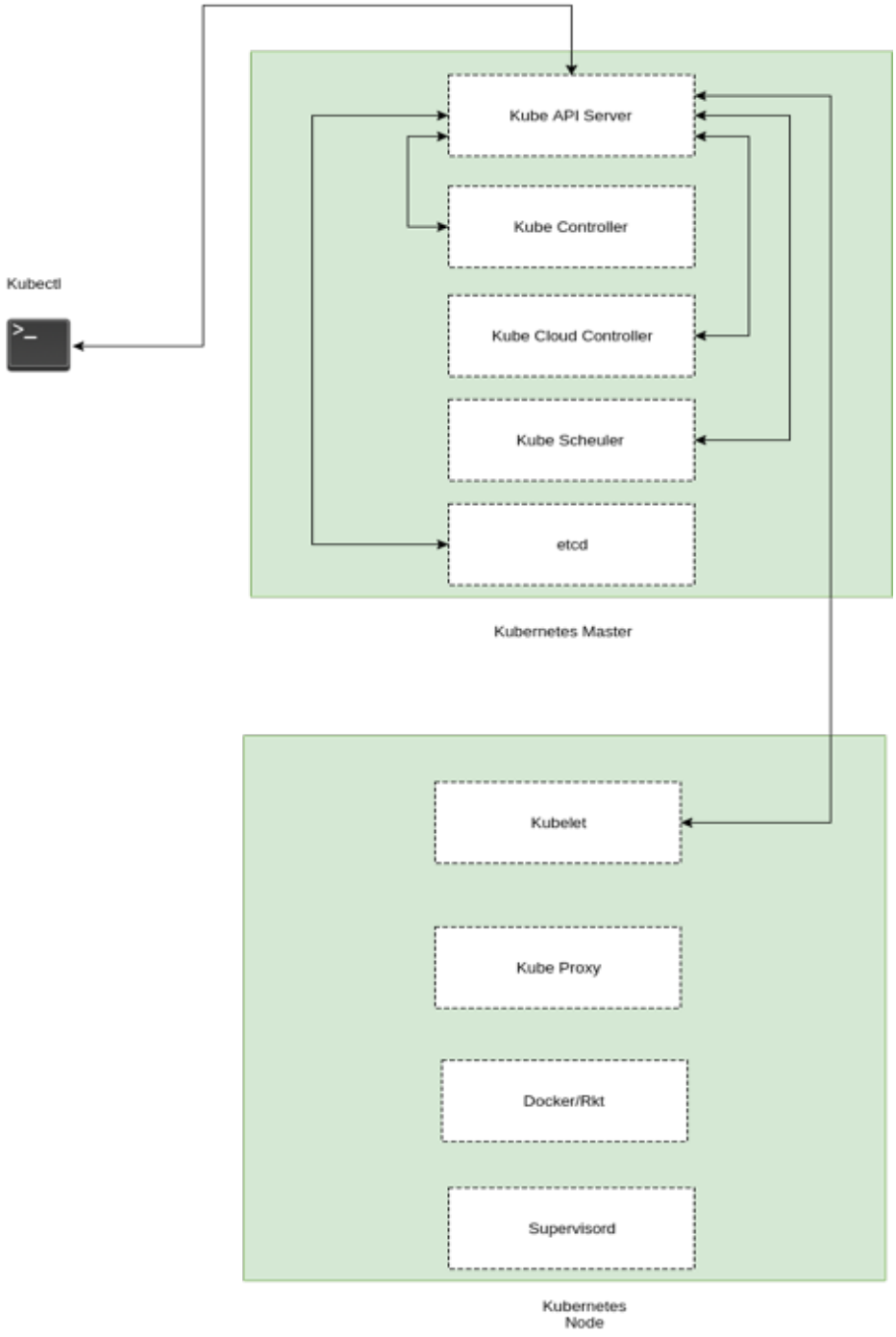
```
$minikube service hello-world-svc
```

It will open the app into default browser.

Understanding the Kubernetes App Flow:

To understand the process in the background when Kubectl creates objects let's look into the master-node communication of Kubernetes.

The following diagram indicates the communication between kubectl, master components, and node components. Here API Server is the responsible component of communication. All the information on Node is associated with kubelet.



When

```
$kubectl create -f deployment.yml
```

Kubectl create command is used for creating Kubernetes object with specification file. Here the two-sided arrows indicate the bi-directional communication between components. Kubelet will inform the Node information to API Server which will be stored in the distributed key-value pair(etcd).The important task of scheduling is done by Kube-scheduler which uses the advanced algorithm to schedule the Kubernetes object on right Node with the help of kube controller manager.

The API Server communicates with Kubelet. If the connection is broken then that node is treated as unhealthy. It will help and maintain the desired state of the cluster by load balancing and autoscaling.

Kubernetes Auto Scaling:

It's possible to scale up and scale down the replicas of the Kubernetes deployment object. To scale up by 20 replicas.

Scaling up:
deployment.yml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: hello-world
  labels:
    app: hello-world
ver: v1
spec:
  replicas: 20
  selector:
matchLabels:
      app: hello-world
ver: v1
  template:
    metadata:
      labels:
        app: hello-world
ver: v1
    spec:
      containers:
      - name: hello-world
        image: kubejack/helloworld:latest
imagePullPolicy: Always
        ports:
        - containerPort: 3000
```

Update the deployment with,

```
$kubectl apply -f deployment.yml
Warning: kubectl apply should be used on resource created by either kubectl create --save-config or kubectl apply
deployment "hello-world" configured
```

```
$kubectl get pods
NAME                           READY     STATUS    RESTARTS   AGE
hello-world-493621601-1g76s    1/1       Running   0          2h
hello-world-493621601-2dq3c    1/1       Running   0          2h
hello-world-493621601-2qdg3    1/1       Running   0          38s
hello-world-493621601-8jnjw    1/1       Running   0          2h
hello-world-493621601-cwm1j    1/1       Running   0          2h
hello-world-493621601-g5sfl    1/1       Running   0          2h
hello-world-493621601-j3jjt    1/1       Running   0          38s
hello-world-493621601-j3rrj    1/1       Running   0          38s
hello-world-493621601-kgj7z    1/1       Running   0          38s
hello-world-493621601-lqvk4    1/1       Running   0          38s
hello-world-493621601-mrktj    1/1       Running   0          38s
hello-world-493621601-nfd5d    1/1       Running   0          38s
```

```
hello-world-493621601-pjbdn   1/1   Running   0   38s
hello-world-493621601-q6xlg   1/1   Running   0   38s
hello-world-493621601-rb7gr   1/1   Running   0   2h
hello-world-493621601-s5qp9   1/1   Running   0   2hs
hello-world-493621601-shl7v   1/1   Running   0   38s
hello-world-493621601-v086d   1/1   Running   0   2hs
hello-world-493621601-xkhhs   1/1   Running   0   2hs
hello-world-493621601-ztp32   1/1   Running   0   2hs
```

It will scale up the replicas to the 20.

Scaling down:

To scale down the application change replicas to the 5, deployment.yml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: hello-world
  labels:
    app: hello-world
ver: v1
spec:
  replicas: 5
  selector:
matchLabels:
```

```
      app: hello-world
  ver: v1
   template:
     metadata:
       labels:
         app: hello-world
  ver: v1
     spec:
       containers:
       - name: hello-world
         image: kubejack/helloworld:latest
  imagePullPolicy: Always
         ports:
         - containerPort: 3000
```

Update the deployment:

```
kubectl apply -f deployment.yml
deployment "hello-world" configured
$kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
hello-world-493621601-1g76s	1/1	Running	0	2h
hello-world-493621601-2dq3c	1/1	Terminating	0	2h
hello-world-493621601-2qdg3	1/1	Terminating	0	2m
hello-world-493621601-8jnjw	1/1	Terminating	0	2h
hello-world-493621601-cwm1j	1/1	Terminating	0	2m

hello-world-493621601-g5sfl	1/1	Terminating	0	2h
hello-world-493621601-j3rrj	1/1	Terminating	0	2h
hello-world-493621601-kgj7z	1/1	Terminating	0	2m
hello-world-493621601-lqvk4	1/1	Terminating	0	2m
hello-world-493621601-mrktj	1/1	Terminating	0	2m
hello-world-493621601-nfd5d	1/1	Terminating	0	2m
hello-world-493621601-pjbdn	1/1	Terminating	0	2m
hello-world-493621601-q6xlg	1/1	Terminating	0	2m
hello-world-493621601-rb7gr	1/1	Running	0	2h
hello-world-493621601-s5qp9	1/1	Running	0	2h
hello-world-493621601-shl7v	1/1	Terminating	0	2m
hello-world-493621601-v086d	1/1	Terminating	0	2h
hello-world-493621601-xkhhs	1/1	Terminating	0	2h
hello-world-493621601-ztp32	1/1	Running	0	2h

It will scale down the Kubernetes deployment.

Destroying Kubernetes Cluster and Pods:

Delete Kubernetes Objects:

To destroy the Kubernetes Application:

```
$kubectl delete -f deployment.yml
deployment "hello-world" deleted
$kubectl delete -f service.yml
service "hello-world-svc" deleted
```

It will delete the Kubernetes Pods, Deployment, ReplicaSets and Services which was created.

Stopping Minikube and Deleting cluster:

```
$minikube stop
Stopping local Kubernetes cluster...
Machine stopped.
```

```
$minikube delete
Deleting local Kubernetes cluster...
Machine deleted.
```

6. Deploying Kubernetes with Ansible

Ansible is the configuration management tool used for the deploying the different types of the infrastructure. It is possible to deploy the Kubernetes with Ansible. Kubespray is a tool used for deploying the Kubernetes cluster with Ansible.

Kubespray can deploy the cluster on AWS, GCE, Azure, OpenStack or Bare Metal. It makes sure the higher availability cluster and composable support with container networking. Kubernetes cluster can be deployed over popular Linux distributions. Even If any CI tests are available then it's possible to run over Kubernetes Cluster created by Kubespray.

Before using Kubespray it's necessary to have following requirements in place:

1. Ansible v2.4 (or newer) and python-netaddr is installed on the machine that will run Ansible commands
2. Jinja 2.9 (or newer) is required to run the Ansible Playbooks
3. The target servers must have access to the Internet in order to pull docker images.
4. The target servers are configured to allow IPv4 forwarding.
5. Your ssh key must be copied to all the servers part of your inventory.
6. The firewalls are not managed, you'll need to implement your own rules the way you used to. in order to avoid any issue during deployment, you should disable your firewall.

Kubespray provides the Terraform and kubespray-cli to provision the environment.

Pre- Requisites for Kubespray

Ansible v2.3 (or newer)
Execute below commands to install the latest ansible on RPM based distributions.

```
$ sudo yum install epel-release
$ sudo yum install ansible
```

Execute below commands to install latest ansible on debian based distributions.

```
$ sudo apt-get update
$ sudo apt-get install software-properties-common
$ sudo apt-add-repository ppa:ansible/ansible
$ sudo apt-get update
$ sudo apt-get install ansible
```

Jinja 2.9 (or newer)

Execute below commands to install Jinja 2.9 or upgrade existing Jinja to version 2.9

```
$ easy_install pip
$ pip2 install jinja2 --upgrade
```

Allow IPv4 forwarding

You can check IPv4 forwarding is enabled or disabled by executing below command.

```
$ sudo sysctl net.ipv4.ip_forward
```

If the value is 0 then, IPv4 forwarding is disabled. Execute below command to enable it.

```
$ sudo sysctl -w net.ipv4.ip_forward=1
```

Your machine ssh key must be copied to all the servers part of your inventory.The firewalls should not be managed and The target servers must have access to the Internet.
For SSH Authentication between machines. On the source machine,

Step 1: Generate Keys

```
$ ssh-keygen -t rsa

Generating public/private rsa key pair.
Enter file in which to save the key (/home/user4/.ssh/id_rsa):
Created directory '/home/user4/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user4/.ssh/id_rsa.
Your public key has been saved in /home/user4/.ssh/id_rsa.pub.
The key fingerprint is:
ad:1e:14:a5:cd:77:25:29:9f:75:ee:4f:a4:8f:f5:65 user4@server1
The key's randomart image is:
+--[ RSA 2048]----+
|         .   ...|
|        =  . .oo|
|       o o .o.+.|
|        o . .o o|
|       S .    + |
|      . .    . E|
|       o      *+|
|      . .    . +|
|       .       |
+-----------------+
```

Note that your key pair is id_rsa and id_rsa.pub files in shown directories. Your id_rsa is the private key which will reside on the source machine. id_rsa.pub is the public key which resides on the destination machine. When SSH attempt is made from source to destination, protocol checks these both keys from source and destination. If they match then the connection will be established without asking password.

Step 2: Copy Keys

Now, we need to copy id_rsa.pub key on the destination machine. It should be copied to a home directory of the intended user in the destination server. It should reside under ~/.ssh/ (i.e. home directory/.ssh/) and with name authorized_keys. You can copy the file using shell or any other file transfer program.

If you are trying from source machine using ssh then use below commands:

```
$ ssh user4@10.10.4.12 "mkdir ~/.ssh"
The authenticity of host '10.10.4.12 (10.10.4.12)' can't be established.
RSA key fingerprint is 08:6c:51:09:9f:4c:69:34:84:ef:08:af:68:df:5e:24.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.10.4.12' (RSA) to the list of known
hosts.
user4@10.10.4.12's password:

$ cat .ssh/id_rsa.pub | ssh user4@10.10.4.12 'cat >> .ssh/authorized_
keys'
user4@10.10.4.12's password:

$ ssh user4@10.10.4.12 "chmod 700 .ssh; chmod 640 .ssh/authorized_
keys"
user4@10.10.4.12's password:
```

Here, the first command creates .ssh directory on the destination machine. Second command copies id_rs.pub file's content to destination machine under file ~/.ssh/authorized_keys and last command set proper permissions.

Repeat the copying steps for each Node of the Kubernetes Cluster.

Step 3: Test the Connection

```
$ ssh user4@10.10.4.12
Last login: Tue Oct  6 21:59:00 2015 from 10.10.4.11
[user4@server2 ~]$
```

Kubespray CLI installation

You can also use kubespray without CLI by directory cloning its git repository. We will use it using CLI. Execute below step to install kubespray.

```
$ pip2 install kubespray
```

You can check the version of kubespray after successful completion of installation

```
$ kubespray -v
```

Inventory File setup

Create new inventory file at ~/.kubespray/inventory/inventory.cfg and Add the contents as shown below.

```
$ vi ~/.kubespray/inventory/inventory.cfg

machine-01 ansible_ssh_host=192.168.0.144 http_proxy=http://
genproxy:8080
machine-02 ansible_ssh_host=192.168.0.145 http_proxy=http://
genproxy:8080
machine-03 ansible_ssh_host=192.168.0.146 http_proxy=http://
genproxy:8080
```

```
[kube-master]
machine-01
machine-02

[etcd]
machine-01
machine-02
machine-03

[kube-node]
machine-02
machine-03

[k8s-cluster:children]
kube-node
kube-master
```

Here the 3 Nodes of the server with the proxy.Let's start the cluster deployment.

Here Inventory is the term specifically related to Ansible. Ansible works with different systems in the infrastructure. Ansible communicates over ssh using the configuration file called as Inventory. Inventory is a configuration file with the .cfg extension which lists the information about the systems. Look following information about the Kubernetes Nodes is stored inside inventory.

Kubernetes Cluster Deployment Using Kubespray

Before, starting actual Deployment, Let's see what will be going behind the scenes and how painful manual installation task is executed smoothly. Kubespray will install kubernetes-api-server, etcd (key-value store), controller, Scheduler will be installed on master machines and kubelet, kube-proxy and Docker (or rkt) will be installed on node machines. These all components will be installed and configured by ansible roles in kubespray. All, We need to do is to execute one command.

To start deployment of the kubernetes cluster, execute following commands.

```
$ kubespray deploy
```

Based on the number of master and minions, It will take time to deploy the complete cluster. At the end of execution, you will get output something like shown below. If there are no failed task, Your deployment is successful.

```
PLAY RECAP *********************************************************
192.168.0.144   :ok=278   changed=89    unreachable=0 failed=0
192.168.0.145   :ok=287   changed=103   unreachable=0 failed=0
192.168.0.146   :ok=246   changed=78    unreachable=0 failed=0
localhost       :ok=3     changed=1     unreachable=0 failed=0
```

To check that Everything went good and deployment was successful, you can login to master node and get all the worker node.

```
$ kubectl get nodes
machine-02          Ready          4m
machine-03          Ready          4m
```

List pods in all namespaces by executing below command.

```
$ kubectl get pods --all-namespaces
NAMESPACE     NAME                                    READY   STATUS    RESTARTS   AGE
kube-system   dnsmasq-7yk3n                           1/1     Running   0          5m
kube-system   dnsmasq-5vfh0j                          1/1     Running   0          5m
kube-system   flannel-machine-02                      2/2     Running   0          4m
kube-system   flannel-machine-03                      2/2     Running   0          4m
kube-system   kube-apiserver-machine-01               1/1     Running   0          5m
kube-system   kube-controller-manager-machine-01      1/1     Running   0          5m
kube-system   kube-proxy-machine-02                   1/1     Running   0          4m
kube-system   kube-proxy-machine-03                   1/1     Running   0          4m
kube-system   kube-scheduler-machine-02               1/1     Running   0          5m
kube-system   kubedns-p8mk7                           3/3     Running   0          4m
kube-system   nginx-proxy-machine-02                  1/1     Running   0          2m
kube-system   nginx-proxy-machine-03                  1/1     Running   0          2m
```

This will install kubernetes using Ansible.

7. Provisioning Storage in Kubernetes

Kubernetes Persistent Volumes

- Requesting storage
- Using Claim as a Volume
- Kubernetes and NFS
- Kubernetes and iSCSI

Provisioning Storage with Kubernetes

Storage plays the important role of storing the data. Kubernetes consists pods which are ephemeral. The important aspect of the having persistent storage is to maintain the state of the application. The most useful method is to have persistent storage attached with containers. The state of the database will be maintained with the Persistent Storage. Kubernetes provides the PersistentVolume object.

Kubernetes currently supports the following plugins:

GCEPersistentDisk
AWSElasticBlockStore
AzureFile
AzureDisk
FC (Fibre Channel)
FlexVolume
Flocker
NFS
iSCSI
RBD (Ceph Block Device)
CephFS
Cinder (OpenStack block storage)
Glusterfs
VsphereVolume
Quobyte Volumes
VMware Photon
Portworx Volumes
ScaleIO Volumes
StorageOS

Kubernetes Persistent Volumes

Kubernetes provides PersistentVolume API for users and administrators. It abstracts the details about how the storage is provided for the other Kubernetes objects. There are two new API are introduced to do so:

1. PersistentVolume (PV)
2. PersistentVolumeClaim (PVC)

A PersistentVolume (PV) is a piece of the storage from the cluster. This storage could be provisioned by the administrator. This API object is responsible for implementation of the storage like NFS, iSCSI or cloud-specific storage. The PersistentVolume is cluster resource similar to the node.

A PersistentVolumeClaim (PVC) is a request made by a user for storage that means PersistentVolume. If we correlate it to the pod, pod consumes the node resources. Similarly, the PersistentVolumeClaim consumes PV resources. Claims can request the specific size and access modes from PersistentVolume.

A StorageClass gives the abstraction for the PersistentVolume. The implementation details are hidden with StorageClass. It also provides the quality of service levels or backup policies with PersistentVolume. It is described as “Class” of the storage.

Lifecycle of the Volume

The interaction between the Persistent Volume and Persistent Volume Claim consists the lifecycle.

This lifecycle consists several phases:
1. Provisioning
2. Binding
3. Using
4. Reclaiming

1. Provisioning :

There are two types of the Persistent Volume provisioning methods. Those

are Static and Dynamic.In the static method, the administrator creates the PV's. But if PV's does not match with users PVC (Persistent Volume Claim) then the dynamic method is used. The cluster will try to generate the PV dynamically for the PVC.

2. Binding:

A control loop on the master watches the PVC's and binds matched PV with PVC. If no PV found matching PVC then the PVC will remain unbounded.

3. Using:

Pods use the Claim as the Volume. Once the PV matches with required PVC, the cluster inspects the claim to find the bound volume and mounts the volume for a pod.

4.Reclaiming

The reclaim policy decides what to do with the PersistentVolume once it has been released. Currently, volumes can be Retained, Recycled or Deleted.

Retain policy consists the reclamation of the resource. If the Persistent Volume Claim is deleted and Persistent Volume is released, still Persistent Volume cant be used for other PVC. The reason behind it that Persistent Volume may contain the data. The administrator can manually reclaim the volume:

a. Delete the PersistentVolume.
b. Manually clean up the data.
c. Manually delete the storage assets.

In **Recycling** the volume again available for new claim. The Recycling performs the basic scrub on the volume if it is supported.

The **Deleting** will delete the PersistentVolume object from the Kubernetes Cluster. It will also delete the associated external infrastructure Volumes like AWS EBS, GCE etc.

Requesting Storage

To request the storage that means PersistentVolume (PV) the PersistentVolumeClaim is used. Here PersistentVolumeClaim is API provided for the user to request PersistentVolume. Similar to Pod, PersistentVolumeClain (PVC) also consists the specification in YAML to request the resource.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: myclaim
spec:
 accessModes:
  - ReadWriteOnce
 resources:
  requests:
   storage: 8Gi
 storageClassName: slow
 selector:
  matchLabels:
   release: "stable"
  matchExpressions:
   - {key: environment, operator: In, values: [dev]}
```

Kind represents the Kubernetes Object. PersistentVolumeClain (PVC) is available with v1 apiVersion.

Metadata is the key-value pair representing the metadata for **PersistentVolumeClaim** object. The specification consists the access modes which is ReadWriteOnce. The access mode is important while accessing the PV resource with given access mode. resources consist the requesting of the storage in Gi. The **storageClassName** represents the classes in the PersistentVolume.

The **selector** can consist of two fields:

1. **matchLabels** - the volume must have a label with this value
2. **matchExpressions** - a list of requirements made by specifying key, list of values, and operator that relates the key and values. Valid operators include In, NotIn, Exists, and DoesNotExist.

Access Modes for Persistent Storage

A Persistent Volume can be mounted on the host. The access modes play the important role for PV's. Its possible to handle the Persistent Volume with different Access Modes.
The access modes are:

- **ReadWriteOnce** – the volume can be mounted as read-write by a single node
- **ReadOnlyMany** – the volume can be mounted read-only by many nodes
- **ReadWriteMany** – the volume can be mounted as read-write by many nodes

In the CLI, the access modes are abbreviated to:

- **RWO** - ReadWriteOnce
- **ROX** - ReadOnlyMany
- **RWX** - ReadWriteMany
-

Using Claim As Volume

Pods are ephemeral and require the storage. Pod uses the claim for the volume. The claim must be in the namespace of the Pod. PersistentVolume is used for the pod which is backing the claim. The volume is mounted to the host and then into the Pod.

```
kind: Pod
apiVersion: v1
metadata:
 name: production-pv
spec:
 containers:
  - name: frontend
    image: dockerfile/nginx
    volumeMounts:
    - mountPath: "/var/www/html"
      name: pv
 volumes:
  - name: pv
    persistentVolumeClaim:
      claimName: storage-pv
```

This is the example of sample pod where volumes and persistentVolumeClaim is used with the claimName myclaim. This is the way pod will use the persistent volume. That means claim as a volume.

Kubernetes and NFS

Kubernetes PersistentVolume can use external volume for storage. nfs volume allows mounting NFS (Network file system) to be mounted into the pod. NFS can be mounted with multiple writers simultaneously. NFS server can be used for provisioning the volume.

Below is the example of NFS volume as Persistent Volume and Persistent Volume Claim:

PersistentVolume:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs
spec:
  capacity:
    storage: 1Mi
  accessModes:
    - ReadWriteMany
  nfs:
    # FIXME: use the right IP
    server: 10.244.1.4
    path: "/"
```

The server is pointing over the NFS server.

PersistentVolumeClaim:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ""
  resources:
    requests:
      storage: 1Mi
```

Kubernetes and iSCSI

SCSI and IP (iSCSI) can be mounted with Kubernetes using an iscsi volume. When the pod dies, the volume preserves the data and the volume is merely unmounted. The feature of iSCSI is that it can be mounted read-only by multiple consumers simultaneously. iSCSI does not allow write with simultaneous users.

The simple example of the iSCSI volume:

```
---
apiVersion: v1
kind: Pod
metadata:
  name: iscsipd
spec:
  containers:
  - name: iscsipd-rw
    image: kubernetes/pause
    volumeMounts:
    - mountPath: "/mnt/iscsipd"
      name: iscsipd-rw
  volumes:
  - name: iscsipd-rw
    iscsi:
      targetPortal: 10.0.2.15:3260
      portals: ['10.0.2.16:3260', '10.0.2.17:3260']
      iqn: iqn.2001-04.com.example:storage.kube.sys1.xyz
      lun: 0
      fsType: ext4
      readOnly: true
```

8. Troubleshooting Kubernetes and Systemd Services

- Kubernetes Troubleshooting Commands
- Networking Constraints
- Inspecting and Debugging Kubernetes
- Querying the State of Kubernetes
- Checking Kubernetes yaml or json Files
- Deleting Kubernetes Components

Kubernetes Troubleshooting Commands

Kubectl utility is used for communicating with cluster through command line environment. It is necessary to debug the Kubernetes services running on master and nodes. Also, the Kubernetes Cluster itself to ensure higher availability of the application. Here few are Kubernetes troubleshooting commands which will help you with troubleshooting Kubernetes.

Getting Information about Pods:

```
$ kubectl get po --namespace kube-system
NAME                                READY     STATUS    RESTARTS   AGE
kube-addon-manager-minikube         1/1       Running   0          35m
kube-dns-910330662-dp7xt            3/3       Running   0          35m
kubernetes-dashboard-1pc46          2/2       Running   0          35m
```

It will give the status of each pod. Also mentioning the namespace is always the best practice. If you don't mention the namespace, then it will list the pods from the default namespace.

To get the detailed information about the pods:

```
$ kubectl describe pods kube-dns-910330662-dp7xt --namespace
kube-system
Name: kube-dns-910330662-dp7xt
Namespace: kube-system
Node: minikube/192.168.99.100
Start Time:     Sun, 12 Nov 2017 17:15:19 +0530
Labels:         k8s-app=kube-dns
                pod-template-hash=910330662
Annotations:    kubernetes.io/created-by={“kind”:”SerializedReferenc
e”,”apiVersion”:”v1”,”reference”:{“kind”:”ReplicaSet”,”namespace”:”ku
be-system”,”name”:”kube-dns-910330662”,”uid”:”f5509e26-c79e-11e7-
a6a9-080027646…
                scheduler.alpha.kubernetes.io/critical-pod=
Status:         Running
IP:         172.17.0.3
Created By:     ReplicaSet/kube-dns-910330662
Controlled By:  ReplicaSet/kube-dns-910330662
Containers
      kubedns:
            Container ID:
docker://76de2b5156ec2a4e2a7f385d700eb96c91874137495b9d715ddf-
5cd4c819452b
Image:          gcr.io/google_containers/k8s-dns-kube-dns-
amd64:1.14.4
Image ID:       docker://sha256:a8e00546bcf3fc9ae1f33302c16a6d4c-
717d0a47a444581b5bcabc4757bcd79c
Ports:          10053/UDP, 10053/TCP, 10055/TCP
Args:
   --domain=cluster.local.
   --dns-port=10053
   --config-map=kube-dns
   --v=2
State:          Running
```

```
 Started:          Sun, 12 Nov 2017 17:15:20 +0530
Ready:             True
Restart Count:  0
   Limits:
memory:  170Mi
Requests:
cpu:  100m
     memory:     70Mi
Liveness:  http-get http://:10054/healthcheck/kubedns delay=60s
timeout=5s period=10s #success=1 #failure=5
Readiness:       http-get http://:8081/readiness delay=3s timeout=5s
period=10s #success=1 #failure=3
   Environment:
     PROMETHEUS_PORT:  10055
   Mounts:
     /kube-dns-config from kube-dns-config (rw)
     /var/run/secrets/kubernetes.io/serviceaccount from default-token-
vdlgs (ro)
  dnsmasq:
   Container ID:
        docker://8dbf3c31a8074e566875e04b66055b1a96dcb4f192acb-
c1a8a083e789bf39a79
Image:             gcr.io/google_containers/k8s-dns-dnsmasq-nanny-
amd64:1.14.4
Image ID:
docker://sha256:f7f45b9cb733af946532240cf7e6cde1278b687cd7094cf-
043b768c800cfdafd
Ports:             53/UDP, 53/TCP
Args:
     -v=2
     -logtostderr
     -configDir=/etc/k8s/dns/dnsmasq-nanny
     -restartDnsmasq=true
```

```
    --
    -k
    --cache-size=1000
    --log-facility=-
    --server=/cluster.local/127.0.0.1#10053
    --server=/in-addr.arpa/127.0.0.1#10053
    --server=/ip6.arpa/127.0.0.1#10053
State:          Running
Started:        Sun, 12 Nov 2017 17:15:20 +0530
Ready:          True
Restart Count:  0
Requests:
cpu:        150m
    memory:         20Mi
Liveness:       http-get http://:10054/healthcheck/dnsmasq delay=60s
timeout=5s period=10s #success=1 #failure=5
   Environment:     <none>
   Mounts:
    /etc/k8s/dns/dnsmasq-nanny from kube-dns-config (rw)
    /var/run/secrets/kubernetes.io/serviceaccount from default-token-
vdlgs (ro)
 sidecar:
   Container ID:
docker:// sha256:38bac66034a6217abfd44b4a8a763b1a4c-
973045cae2763f2cc857baa5c9a872
   Image:          gcr.io/google_containers/k8s-dns-sidecar-
amd64:1.14.4
Image ID:
       docker://sha256:38bac66034a6217abfd44b4a8a763b1a4c-
973045cae2763f2cc857baa5c9a872
Port:       10054/TCP
   Args:
    --v=2
    --logtostderr
```

```
    --
probe=kubedns,127.0.0.1:10053,kubernetes.default.svc.cluster.
local.,5,A
    --
probe=dnsmasq,127.0.0.1:53,kubernetes.default.svc.cluster.local.,5,A
State:          Running
Started:        Sun, 12 Nov 2017 17:15:19 +0530
Ready:          True
Restart Count:  0
Requests:
    cpu:        10m
    memory:         20Mi
  Liveness:         http-get http://:10054/metrics delay=60s
timeout=5s period=10s #success=1 #failure=5
Environment:  <none>
  Mounts:
    /var/run/secrets/kubernetes.io/serviceaccount from default-token-
vdlgs (ro)
Conditions:
 Type           Status
 Initialized    True
 Ready    True
 PodScheduled       True
Volumes:
kube-dns-config:
  Type:    ConfigMap (a volume populated by a ConfigMap)
Name:      kube-dns
Optional:  true
 default-token-vdlgs:
  Type:    Secret (a volume populated by a Secret)
  SecretName:  default-token-vdlgs
  Optional:    false
QoS Class:Burstable
```

```
Node-Selectors:      <none>
Tolerations:     CriticalAddonsOnly
Events:
FirstSeen  LastSeen   Count      From                    SubObjectPath
Type        Reason                 Message
  ---------    --------       -----   ----                 -------------
        --------       ------                 -------
40m         40m         1      default-scheduler
Normal            Scheduled              Successfully assigned kube-dns-
910330662-dp7xt to minikube
40m         40m         1      kubelet, minikube
Normal            SuccessfulMountVolume  MountVolume.SetUp
succeeded for volume "kube-dns-config"
40m         40m         1      kubelet, minikube
Normal            SuccessfulMountVolume  MountVolume.SetUp
succeeded for volume "default-token-vdlgs"
40m         40m         1      kubelet, minikube    spec.
containers{sidecar}
Normal            Pulled                 Container image "gcr.io/
google_containers/k8s-dns-sidecar-amd64:1.14.4" already present on
machine
40m         40m         1      kubelet, minikube    spec.
containers{sidecar}
Normal            Created                Created container
  40m             40m         1      kubelet, minikube    spec.
containers{sidecar}
Normal            Started                Started container
40m         40m         1      kubelet, minikube    spec.
containers{kubedns}
Normal            Pulled                 Container image "gcr.io/google_
containers/k8s-dns-kube-dns-amd64:1.14.4" already present on
machine
```

```
40m         40m         1       kubelet, minikube    spec.
containers{kubedns}
Normal          Created                 Created container
40m         40m         1       kubelet, minikube    spec.
containers{kubedns}
Normal          Started                 Started container


 40m            40m         1       kubelet, minikube    spec.
containers{dnsmasq}
Normal          Pulled                  Container image
"gcr.io/google_containers/k8s-dns-dnsmasq-nanny-amd64:1.14.4"
already present on machine
40m         40m         1       kubelet, minikube    spec.
containers{dnsmasq}
Normal          Created                 Created container
 40m            40m         1       kubelet, minikube    spec.
containers{dnsmasq}
Normal          Started                 Started container
```

Check if Kubernetes nodes are ready or not:

```
$ kubectl get nodes
NAME            STATUS          AGE         VERSION
minikube        Ready           43m         v1.7.5
```

If some node fails, then the the STATUS will be not ready.

It's possible to get the information about Kubernetes Objects using kubectl get

$ kubectl get
You must specify the type of resource to get. Valid resource types include:

* all
* certificatesigningrequests (aka 'csr')
* clusterrolebindings
* clusterroles
* clusters (valid only for federation apiservers)
* componentstatuses (aka 'cs')
* configmaps (aka 'cm')
* controllerrevisions
* cronjobs
* daemonsets (aka 'ds')
* deployments (aka 'deploy')
* endpoints (aka 'ep')
* events (aka 'ev')
* horizontalpodautoscalers (aka 'hpa')
* ingresses (aka 'ing')
* jobs
* limitranges (aka 'limits')
* namespaces (aka 'ns')
* networkpolicies (aka 'netpol')
* nodes (aka 'no')
* persistentvolumeclaims (aka 'pvc')
* persistentvolumes (aka 'pv')
* poddisruptionbudgets (aka 'pdb')
* podpreset
* pods (aka 'po')
* podsecuritypolicies (aka 'psp')
* podtemplates

```
* replicasets (aka 'rs')
* replicationcontrollers (aka 'rc')
* resourcequotas (aka 'quota')
* rolebindings
* roles
* secrets
* serviceaccounts (aka 'sa')
* services (aka 'svc')
* statefulsets
* storageclasses
* thirdpartyresources
```

Additionally you can specify the namespace to get the Kubernetes objects from specific namespace. In commands (aka 'shortform') shows the short name for the commands. I.e. kubectl get pods and kubectl get po will return same.

If the pod is crashing / unhealthy:

```
$ kubectl logs ${POD_NAME} ${CONTAINER_NAME}
```

To run the commands inside the container:

```
$ kubectl exec ${POD_NAME} -c ${CONTAINER_NAME} -- ${CMD}
${ARG1} ${ARG2} ... ${ARGN}
```

For example:

```
$ kubectl exec cassandra -- cat /var/log/cassandra/system.log
```

Check if your cluster process is running with ps:

```
$ ps -ef | grep kube
ut       6534  6237 36 17:10 ?        00:25:38 /usr/lib/virtualbox/
VBoxHeadless --comment minikube --startvm c0baba8e-2f4b-4c9c-
83d3-bd83c9787fde --vrde config
```

In this case minikube process is running.

Also if the container engine is docker,checking the docker containers on each node and master running necessary services will help troubleshooting the cluster.

```
$ docker ps
CONTAINER ID       IMAGE                                          COMMAND
CREATED            STATUS           PORTS            NAMES57ce7b913026
gcr.io/google_containers/heapster_grafana:v2.6.0-2          "/bin/sh -c /
run.sh"     28 seconds ago      Up 27 se
conds                    k8s_grafana.eb42e400_influxdb-grafana-qkps6_
kube-system_f421d2cf-c7a8-11e7-909a-0242ac11005e_130df7e5278222
b93e3c       kubernetes/heapster_influxdb:v0.6                    "influxd
--config ..."   29 seconds ago      Up 27 seconds                    k8s_
influxdb.78c83de7_influxdb-grafana-qkps6_kube-system_f421d2cf-
c7a8-11e7-909a-0242ac11005e_9a71b614
5b430ca4827c       gcr.io/google_containers/pause-amd64:3.0
"/pause"              29 seconds ago      Up 28 seconds                    k8s_
POD.d8dbe16c_influxdb-grafana-qkps6_kube-system_f421d2cf-c7a8-
11e7-909a-0242ac11005e_6364783d72155910afc7       gcr.io/google_
containers/pause-amd64:3.0                  "/pause"              29 seconds
ago      Up 28 seconds                    k8s_POD.d8dbe16c_heapster-gfrzl_
kube-system_f3f8a7da-c7a8-11e7-909a-0242ac11005e_f63e1e28
e18ecf4006e0       gcr.io/google_containers/kubernetes-dashboard-
amd64:v1.5.1   "/dashboard --port..."   29 seconds ago      Up 28 seconds
k8s_kubernetes-dashboard.5374e7ba_kubernetes-dashboard-znmh3_
kube-system_f3cd40bc-c7a8-11e7-909a-0242ac11005e_61857f68
```

```
d08cad36b80f       gcr.io/google_containers/pause-amd64:3.0
"/pause"            29 seconds ago      Up 28 seconds                   k8s_
POD.d8dbe16c_kubernetes-dashboard-znmh3_kube-system_
f3cd40bc-c7a8-11e7-909a-0242ac11005e_01475695d02d0db563c7
gcr.io/google-containers/kube-addon-manager-amd64:v2        "/opt/
kube-addons.sh"    31 seconds ago      Up 30 se
conds                   k8s_kube-addon-manager.67196b82_kube-addon-
manager-host01_kube-system_389bfd83002ddc85e82b52309ae3a3c2_
af0dcba7d1db6bebbefb       gcr.io/google_containers/pause-amd64:3.0
"/pause"            31 seconds ago      Up 30 se
conds                   k8s_POD.d8dbe16c_kube-addon-manager-
host01_kube-system_389bfd83002ddc85e82b52309ae3a3c2_
a9942f3c107da6b3c71a       gcr.io/google_containers/localkube-
amd64:v1.5.2        "/localkube start ..."   37 seconds ago      Up 36 se
conds                   minikube
```

This command will return the all containers which are running on **Nodes and Masters.**

Networking Constraints

Kubernetes uses the service object to expose the application to the outside world. The service act as the proxy for the Kubernetes set of pods. To the kubernize application, it is necessary to keep in mind that the application will support Network Address Translation (NAT) and does not requires forward and reverse proxy.

Inspecting and Debugging Kubernetes

To inspect and debug the Kubernetes it's recommended to have the good understanding of the high-level architecture of the Kubernetes. Kubernetes architecture consists masters and nodes. Each master is having master-components. Also for the nodes, node-components are in-place.

Make sure cluster is up and running on all nodes are ready.

```
$ kubectl get nodes
NAME       STATUS   AGE      VERSION
minikube   Ready    1h       v1.7.5
```

If the status is ready for all Nodes that means the all necessary system processes are running on the Kubernetes Nodes.
Check the necessary pods running inside Kubernetes master and Kubernetes Nodes.

Run following command on every Node and Master:

```
$ kubectl get po -o wide --namespace kube-system
NAME                                    READY   STATUS    RESTARTS   AGE   IP              NODE
kube-addon-manager-minikube             1/1     Running   0          1h    192.168.99.100  minikube
kube-dns-910330662-dp7xt                3/3     Running   0          1h    172.17.0.3      minikube
kubernetes-dashboard-1pc46              1/1     Running   0          1h    172.17.0.2      minikube
```

The result of above command will vary on the type of installation. In this case the cluster is running through minikube.

Alternatively its possible to check containers using docker if the docker is container engine used within Kubernetes.

Use:

```
$ docker ps
```

It will also help to determine the master-level components running on the master. Note that following services are important on masters and nodes:

Master services: Services include: kube-controller-manager, kube-scheduler, flanneld, etcd, and kube-apiserver. The flanneld service is optional and it is possible to run the etcd services on another system.

Node services: Services include: docker,kube-proxy, kubelet, flanneld. The flanneld service is optional.

Flanneld is optional service and depends on the container networking it may be different.

If any of the service is failed to launch on the Kubernetes then check the setup (YAML) file of the service (usually at /etc/kubernetes) and try to re-start the service.

Checking the system logs through journalctl:
If there is a problem of starting specific services then the best way to debug it to check the systemd journalctl log.

Example:

```
# journalctl -l -u kubelet
# journalctl -l -u kube-apiserver
```

Querying the State of Kubernetes

Kubectl command line tool is used for getting the state of the Kubernetes. Kubectl configured with the API server. API server is the key component of the master. Kubectl utility is used to get the all information about the Kubernetes Objects. It includes Pods, Deployments, ReplicationControllers, PersistentStorage etc.

To get the information:
kubectl get command is used.

Following is the example of the few objects retrieved through the Kubectl command line.

```
$ kubectl get pods --namespace kube-system
NAME                                READY     STATUS    RESTARTS   AGE
heapster-q377d                      0/1       Error     2          22s
influxdb-grafana-x4h8c              2/2       Running   0          22s
kube-addon-manager-host01           1/1       Running   0          24s
kubernetes-dashboard-4vnrc          1/1       Running   0          22s

$ kubectl get services --namespace kube-system
NAME                    TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S          AGE
heapster                ClusterIP   10.0.0.90      <none>        80/TCP          33s
kubernetes-dashboard    NodePort    10.0.0.115     <none>        80:30000/TCP    33s
monitoring-influxdb     ClusterIP   10.0.0.166     <none>        8083/TCP,8086/TCP   32s
```

```
$ kubectl get rc --namespace kube-system
NAME                    DESIRED   CURRENT   READY     AGE
heapster                1         1         0         44s
influxdb-grafana        1         1         1         44s
kubernetes-dash-board   1         1         1         44s
$ kubectl get deployment --namespace kube-system
No resources found.
```

This is just the querying for getting the information of the Kubernetes objects. To describe the specific object kubectl describe command is used. Let's describe the information about the pod.

```
$ kubectl describe pods influxdb-grafana-x4h8c --namespace kube-system
Name:                   influxdb-grafana-x4h8c
Namespace:              kube-system
Node:                   host01/172.17.0.101
Start Time:             Mon, 13 Nov 2017 14:11:51 +0000
Labels:                 kubernetes.io/cluster-service=true
                        name=influxGrafanaAnnotations:
kubernetes.io/created-by={"kind":"Serialized Reference","apiVersion":"v1","reference":{"kind":"ReplicationController", "namespace":"kube-system","name":"influxdb-grafana","uid":"97e17ca5-c87c-11e7-ac86-...

Status:                 Running
IP:                     172.17.0.101
```

```
Created By:          ReplicationController/influxdb-grafana
Controlled By:       ReplicationController/influxdb-grafana
Containers:
influxdb:
Container ID:
docker://d43465cc2b130d736b83f465666c652a71d05a2a169eb72f2369c-
3d96723726c
Image:               kubernetes/heapster_influxdb:v0.6
  Image ID:          docker-pullable://kubernetes/heap-
                     ster_influxdb@sha256:70b34b65def36fd-
                     0f54af570d5e72ac41c3c82e086dace9e6a-
                     977bab7750c147
Port:                <none>
State:               Running
Started:             Mon, 13 Nov 2017 14:11:52 +0000
  Ready:             True
Restart Count:       0
Environment:         <none>
Mounts:              /data from influxdb-storage (rw)
                        /var/run/secrets/kubernetes.io/
                     serviceaccount from default-token-3rdpl (ro)
grafana:
  Container ID:      docker://9808d5b4815bd87e55de203df5ce2b-
                     7503d7a4a9b6499b466ad70f86f5123047
Image:               gcr.io/google_containers/heapster_
                     grafana:v2.6.0-2
Image ID:            docker-pullable://gcr.io/google_contain-
                     ers/heapster_grafana@sha256:208c98b-
                     77d4e18ad7759c0958bf87d467a3243bf75b-
                     76f1240a577002e9de277
  Port:              <none>
State:               Running
Started:             Mon, 13 Nov 2017 14:11:52 +0000
```

Ready:	True
Restart Count:	0
Environment:	
INFLUXDB_SERVICE_URL:	http://localhost:8086
GF_AUTH_BASIC_ENABLED:	false
GF_AUTH_ANONYMOUS_ENABLED:	: true
GF_AUTH_ANONYMOUS_ORG_ROLE:	Admin
GF_SERVER_ROOT_URL:	/
Mounts:	/var from grafana-storage (rw)
	/var/run/secrets/kubernetes.io/ serviceaccount from default-token-3rdpl (ro)
Conditions:	
Type	Status
Initialized	True
Ready	True
PodScheduled	True
Volumes:	
influxdb-storage:	
Type:	: EmptyDir (a temporary directory that shares a pod's lifetime)
Medium:	
grafana-storage:	
Type:	EmptyDir (a temporary directory that shares a pod's lifetime)
Medium:	
default-token-3rdpl:	
Type:	Secret (a volume populated by a Secret)

```
SecretName:                 default-token-3rdpl
Optional:                   false
QoS Class:                  BestEffort
Node-Selectors:             <none>
Tolerations:                <none>
Events:
```

Type	Reason	Age	From	Message
Normal	Scheduled	3m	default-scheduler	Successfully assigned influxdb-grafana-x4h8c to host01
Normal	Pulled	3m	kubelet, host01	Container image "kubernetes/heapster_influxdb:v0.6" already present on machine
Normal	Created	3m	kubelet, host01	Created container with docker id d43465cc2b13; Security:[seccomp=unconfined]
Normal	Started	3m	kubelet, host01	Started container with docker id d43465cc2b13
Normal	Pulled	3m	kubelet, host01	Container image "gcr.io/google_containers/heapster_grafana:v2.6.0-2" already present on machine
Normal	Created	3m	kubelet, host01	Created container with docker id 9808d5b4815b; Security:[seccomp=unconfined]
Normal	Started	3m	kubelet, host01	Started container with docker id 9808d5b4815b

This consists the in depth level of information about the Pod.

Checking Kubernetes yaml or json Files

Kubernetes supports the declarative method of the specification of the Kubernetes object. The declarative approach is one of the recommended approaches to define the specifications. The specification file can be in YAML or JSON format.

The first thing to do with the declarative approach is to validate the YAML or JSON file which contains the specifications. Online validators are available like http://www.yamllint.com/ and https://jsonlint.com/ . This will remove the syntax and parsing error from the specification file.

Checking error with kubectl create command. kubectl create command is used for the creating the Kubernetes object with the specification file.

deployment.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: hello-world
  labels:
    app: hello-world
    ver: v1
spec:
  replicas: 20
  selector:
    matchLabels:
      app: hello-world
      ver: v1
  template:
    metadata:
      labels:
        app: hello-world
```

```
      ver: v1
  spec:
    containers:
    - name: hello-world
      image: kubejack/helloworld:latest
      imagePullPolicy: Always
      ports:
      - containerPort: 3000

kubectl create -f deployment.yaml
```

It will throw an error if there is any syntax error or API error.

Deleting Kubernetes Components

With Kubectl it is possible to delete the components of Kubernetes. If declarative definition is used then the similar definition can be used for deletion.

deployment.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: hello-world
  labels:
    app: hello-world
    ver: v1
spec:
  replicas: 20
  selector:
    matchLabels:
```

```
    app: hello-world
    ver: v1
  template:
   metadata:
    labels:
     app: hello-world
     ver: v1
   spec:
    containers:
    - name: hello-world
     image: kubejack/helloworld:latest
     imagePullPolicy: Always
     ports:
     - containerPort: 3000

kubectl delete -f deployment.yaml
```

kubectl delete command will be used for the deleting the kubernetes components. In imperative method there are different way of deleting the kubernetes components.

```
# Delete pods and services with same names "baz" and "foo"
kubectl delete pod,service baz foo

# Delete pods and services with label name=myLabel.
kubectl delete pods,services -l name=myLabel

# Delete a pod with UID 1234-56-7890-234234-456456.
kubectl delete pod 1234-56-7890-234234-456456

# Delete all pods
kubectl delete pods --all
```

Similarly it will work with all other object of Kubernetes.

9. Kubernetes Maintenance

- Monitoring Kubernetes Cluster
- Managing Kubernetes with Dashboard
- Logging Kubernetes Cluster
- Upgrading Kubernetes

Monitoring Kubernetes Cluster

For reliable applications, it is required to have in place monitoring of the Kubernetes Cluster. It helps to determine availability, scalability, and reliability of the application deployed over Kubernetes Cluster.

Heapster aggregator is used for monitoring and event the logs. Heapster stores the information in storage backend. Currently, it supports Google Cloud Monitoring and InfluxDB as the storage backends. Heapster runs as a pod in the Kubernetes Cluster. It communicates with each node of the Kubernetes Cluster. Kubelet agent is responsible to provide the monitoring information to Heapster. Kubelet itself collects the data from cAdvisor.

cAdvisor:

cAdvisor is an open source container usage and performance analysis agent. In Kubernetes cAdvisor is included into Kubelet binary.cAdvisor auto-discovers all containers. It collects the information of CPU, memory, network and file system usage statistics.

Kubelet:

Kubelet bridge the gap between Kubernetes Master and Kubernetes Node. It manages the Pods and Containers on each machine.

InfluxDB and Grafana are used for storing the data and visualizing it. Google cloud monitoring provides the hosted solution for monitoring Kubernetes Cluster. Heapster can be set up to send the metrics to Google Cloud monitoring.

Let's check the Monitoring Kubernetes Cluster created with Minikube:

```
$ minikube addons list
- addon-manager: enabled
- dashboard: enabled
- kube-dns: enabled
- default-storageclass: enabled
- heapster: disabled
- ingress: disabled
- registry: disabled
- registry-creds: disabled
```

Enable the addon:

```
$ minikube addons enable heapster
```

To open the web interface:

```
$ minikube addons open heapster
```

The result will be displayed on the grafana.

In Minikube addons are helping for monitoring but it's also possible to add heapster as Kubernetes deployment. This will be the manual installation of heapster, grafana and influxdb.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: heapster
  namespace: kube-system
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: heapster
  namespace: kube-system
spec:
  replicas: 1
  template:
    metadata:
      labels:
        task: monitoring
        k8s-app: heapster
    spec:
      serviceAccountName: heapster
      containers:
      - name: heapster
        image: gcr.io/google_containers/heapster-amd64:v1.4.0
        imagePullPolicy: IfNotPresent
        command:
        - /heapster
        - --source=kubernetes:https://kubernetes.default
        - --sink=influxdb:http://monitoring-influxdb.kube-system.
svc:8086
---
apiVersion: v1
kind: Service
```

```
metadata:
  labels:
    task: monitoring
    # For use as a Cluster add-on (https://github.com/kubernetes/kubernetes/tree/master/cluster/addons)
    # If you are NOT using this as an addon, you should comment out this line.
    kubernetes.io/cluster-service: 'true'
    kubernetes.io/name: Heapster
  name: heapster
  namespace: kube-system
spec:
  ports:
  - port: 80
    targetPort: 8082
  selector:
    k8s-app: heapster
```

You can get the latest version of the heapster at https://github.com/kubernetes/heapster/ .

Using Kubectl:

```
$ kubectl create -f heapster.yaml
```

For grafana, use grafana.yaml:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: monitoring-grafana
  namespace: kube-system
spec:
  replicas: 1
  template:
    metadata:
      labels:
        task: monitoring
        k8s-app: grafana
    spec:
      containers:
      - name: grafana
        image: gcr.io/google_containers/heapster-grafana-amd64:v4.4.3
        ports:
        - containerPort: 3000
          protocol: TCP
        volumeMounts:
        - mountPath: /etc/ssl/certs
          name: ca-certificates
          readOnly: true
        - mountPath: /var
          name: grafana-storage
        env:
        - name: INFLUXDB_HOST
          value: monitoring-influxdb
        - name: GF_SERVER_HTTP_PORT
          value: "3000"
          # The following env variables are required to make Grafana ac-
cessible via
```

```
        # the kubernetes api-server proxy. On production clusters, we recommend
        # removing these env variables, setup auth for grafana, and expose the grafana
        # service using a LoadBalancer or a public IP.
       - name: GF_AUTH_BASIC_ENABLED
         value: "false"
       - name: GF_AUTH_ANONYMOUS_ENABLED
         value: "true"
       - name: GF_AUTH_ANONYMOUS_ORG_ROLE
         value: Admin
       - name: GF_SERVER_ROOT_URL
         # If you're only using the API Server proxy, set this value instead:
         # value: /api/v1/namespaces/kube-system/services/monitoring-grafana/proxy
         value: /
     volumes:
     - name: ca-certificates
       hostPath:
         path: /etc/ssl/certs
     - name: grafana-storage
       emptyDir: {}
---
apiVersion: v1
kind: Service
metadata:
  labels:
    # For use as a Cluster add-on (https://github.com/kubernetes/kubernetes/tree/master/cluster/addons)
    # If you are NOT using this as an addon, you should comment out this line.
    kubernetes.io/cluster-service: 'true'
    kubernetes.io/name: monitoring-grafana
  name: monitoring-grafana
```

```
  namespace: kube-system
spec:
  # In a production setup, we recommend accessing Grafana through an external Loadbalancer
  # or through a public IP.
  # type: LoadBalancer
  # You could also use NodePort to expose the service at a randomly-generated port
  # type: NodePort
  ports:
  - port: 80
    targetPort: 3000
  selector:
    k8s-app: grafana
```

You can get the latest version of grafana.yaml at https://github.com/kubernetes/heapster/blob/master/deploy/kube-config/influxdb/grafana.yaml .

Using kubectl :

```
$ kubectl create -f grafana.yaml
```

If influxdb is the storage backend, then use following YAML for deploying influxdb in Kubernetes Cluster:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: monitoring-influxdb
  namespace: kube-system
```

```
spec:
  replicas: 1
  template:
    metadata:
      labels:
        task: monitoring
        k8s-app: influxdb
    spec:
      containers:
      - name: influxdb
        image: gcr.io/google_containers/heapster-influxdb-amd64:v1.3.3
        volumeMounts:
        - mountPath: /data
          name: influxdb-storage
      volumes:
      - name: influxdb-storage
        emptyDir: {}
---
apiVersion: v1
kind: Service
metadata:
  labels:
    task: monitoring
    # For use as a Cluster add-on (https://github.com/kubernetes/kubernetes/tree/master/cluster/addons)
    # If you are NOT using this as an addon, you should comment out this line.
    kubernetes.io/cluster-service: 'true'
    kubernetes.io/name: monitoring-influxdb
  name: monitoring-influxdb
  namespace: kube-system
spec:
  ports:
```

```
  - port: 8086
    targetPort: 8086
  selector:
    k8s-app: influxdb
```

You can get the latest version of influxdb.yaml at https://github.com/kubernetes/heapster/blob/master/deploy/kube-config/influxdb/influxdb.yaml

Using Kubectl:

```
$ kubectl create -f influxdb.yaml
```

To access grafana dashboard with the manual setup, describe the grafana service and check endpoint of the service.

To describe the service using Kubectl use following command:

```
$ kubectl describe svc monitoring-grafana --namespace kube-system
Name:                   monitoring-grafana
Namespace:              kube-system
Labels:                 addonmanager.kubernetes.io/mode=Reconcile
                        kubernetes.io/minikube-addons=heapster
                        kubernetes.io/minikube-addons-endpoint=heapster
                        kubernetes.io/name=monitoring-grafana
Annotations:            kubectl.kubernetes.io/last-applied-configura-
                        tion={"apiVersion":"v1","kind":"Service","meta-
                        data":{"annotations":{},"labels":{"addonmanager.
                        kubernetes.io/mode":"Reconcile","kubernetes.io/
                        minikube-addons...
Selector:               addonmanager.kubernetes.io/mode=Recon-
                        cile,name=influxGrafana
Type:                   NodePort
```

```
IP:                10.0.0.62
Port:              <unset>  80/TCP
NodePort:          <unset>  30943/TCP
Endpoints:         172.17.0.9:3000
Session Affinity:  None
Events:            <none>
```

Prometheus and Data Dog are also good tools for monitoring the Kubernetes Cluster.

Managing Kubernetes with Dashboard

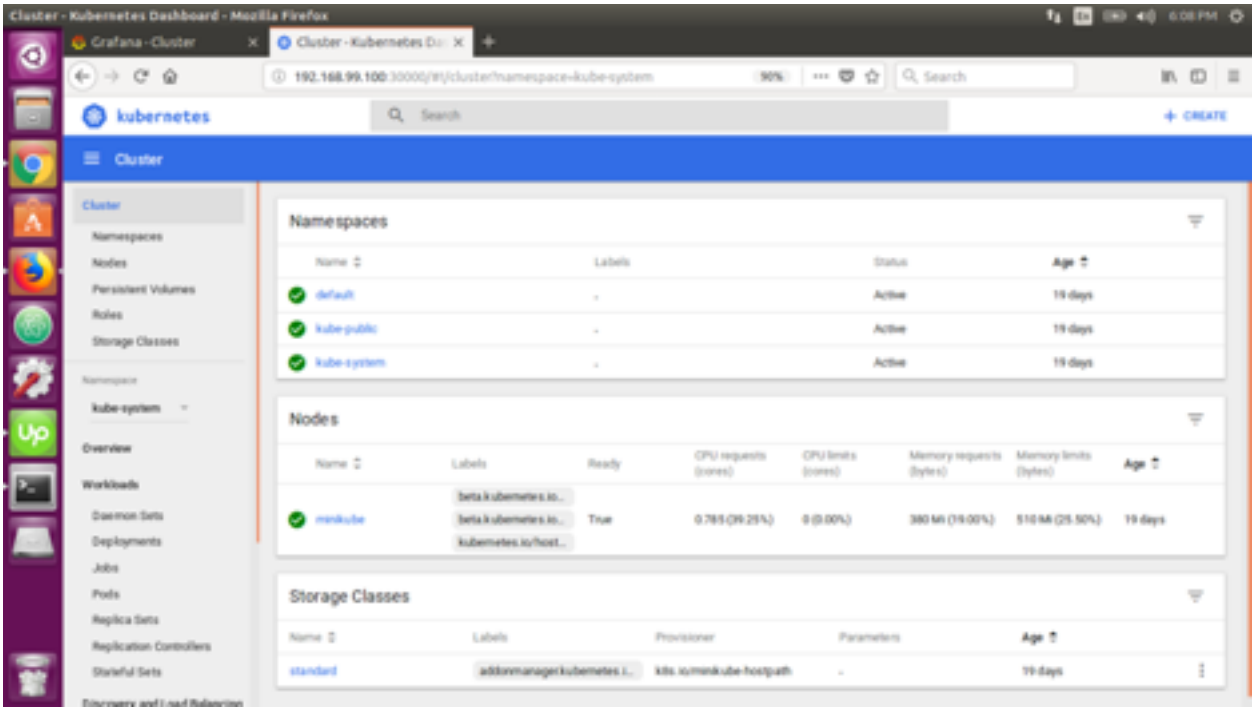
Kubernetes provides the web interface to manage the Kubernetes Cluster. With the Role Based Access Policy, it becomes simpler and easier to manage Kubernetes Cluster with Dashboard. For Minikube it gives the addon. That means Minikube provides addons for monitoring, dashboard and for managing the cluster.

To enable the addon of the dashboard on minikube:

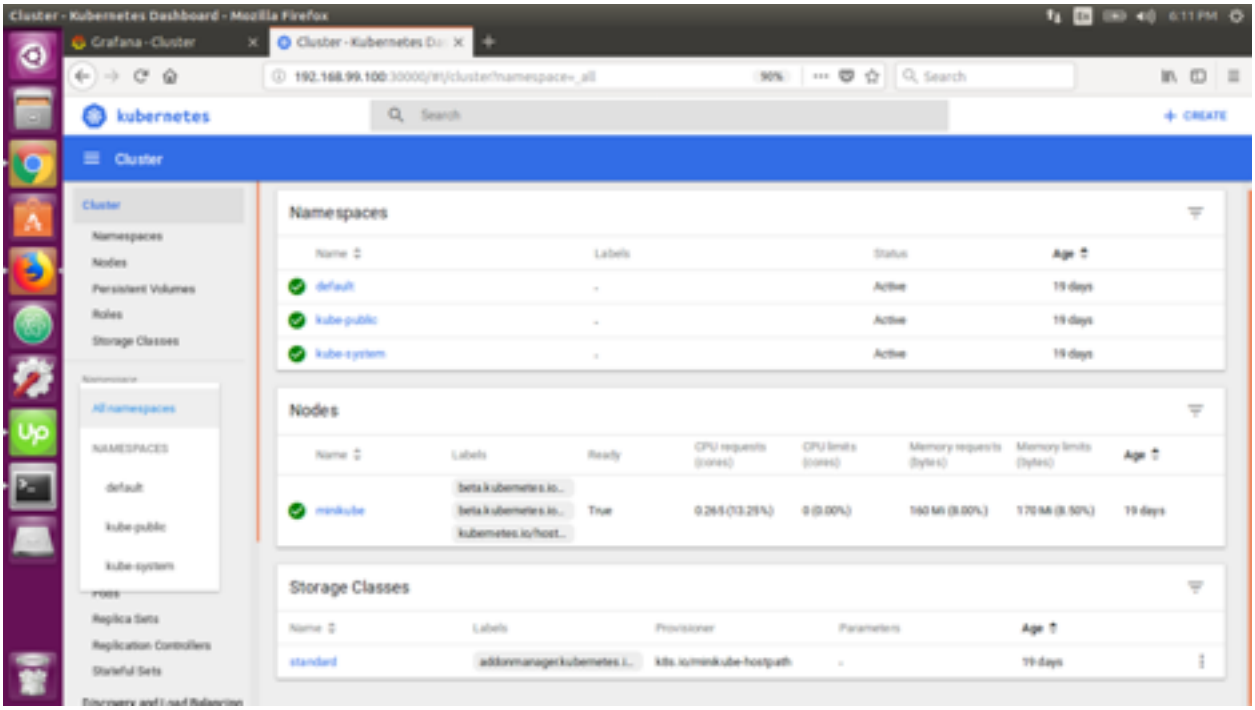
```
$ minikube addons enable dashboard
dashboard was successfully enabled
$ minikube addons open dashboard
Opening kubernetes service kube-system/kubernetes-dashboard in
default browser...
```

With Minikube, it will open the dashboard in the browser. The dashboard consists the Cluster infromation, Namespace information and information about the objects associated with cluster like workloads etc.The dashboard consists the three different informative sections including workloads, discovery and load balancing, config and storage.

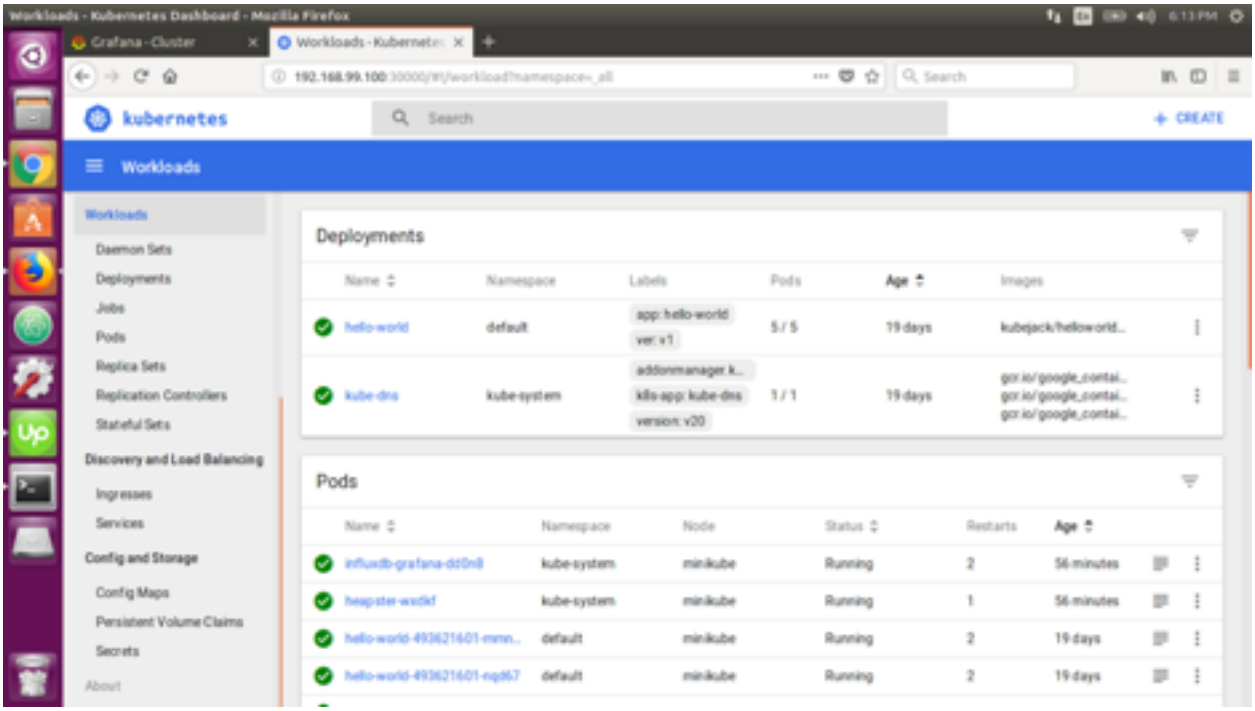
In the cluster tab, Kubernetes dashboard shows the information about the Namespaces, Nodes, Persistent Volumes and Storage Classes.



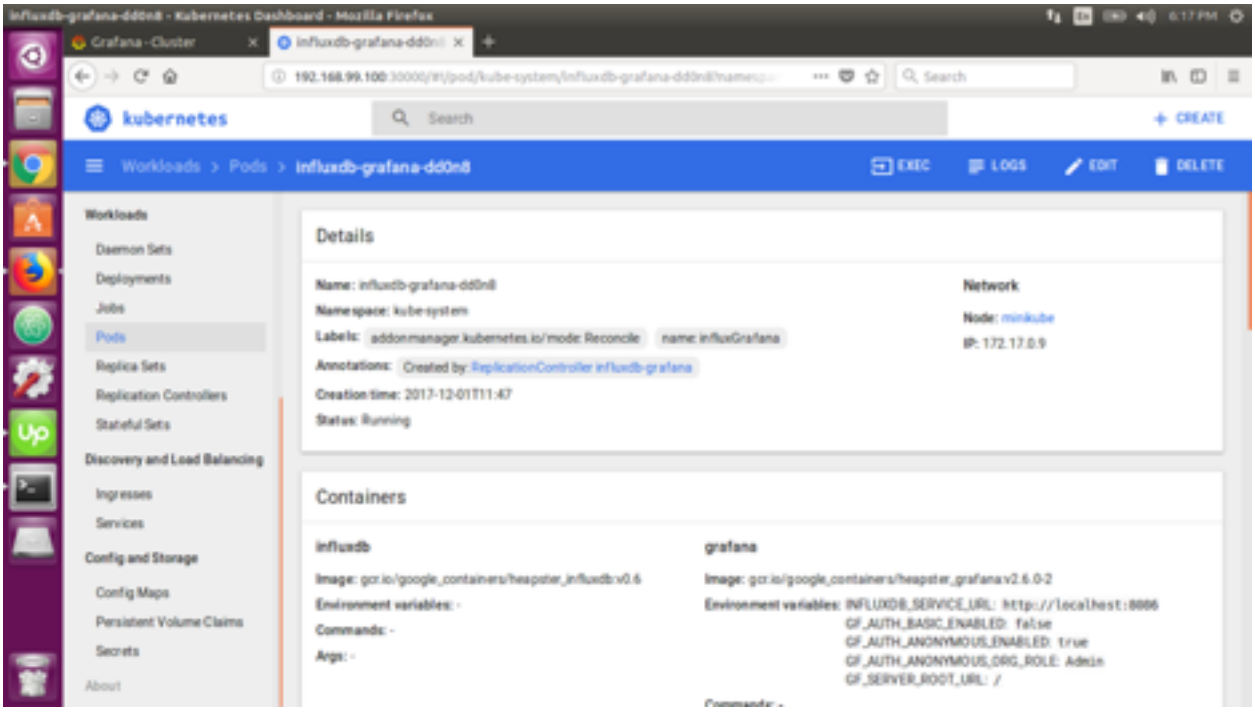
It gives the brief information about the cluster. In namespaces, it shows the all available namespaces in Kubernetes Cluster. It's possible to select all namespaces or specific namespace.



Depending on the namespace selected previously, further tabs show in depth information of associated Kubernetes object within selected namespace. It consists the workloads of Kubernetes which includes Deployments, Replica Sets, Replication Controller, Daemon Sets, Jobs, Pods and Stateful Sets. Each of this workload is important to run an application over the Kubernetes Cluster.

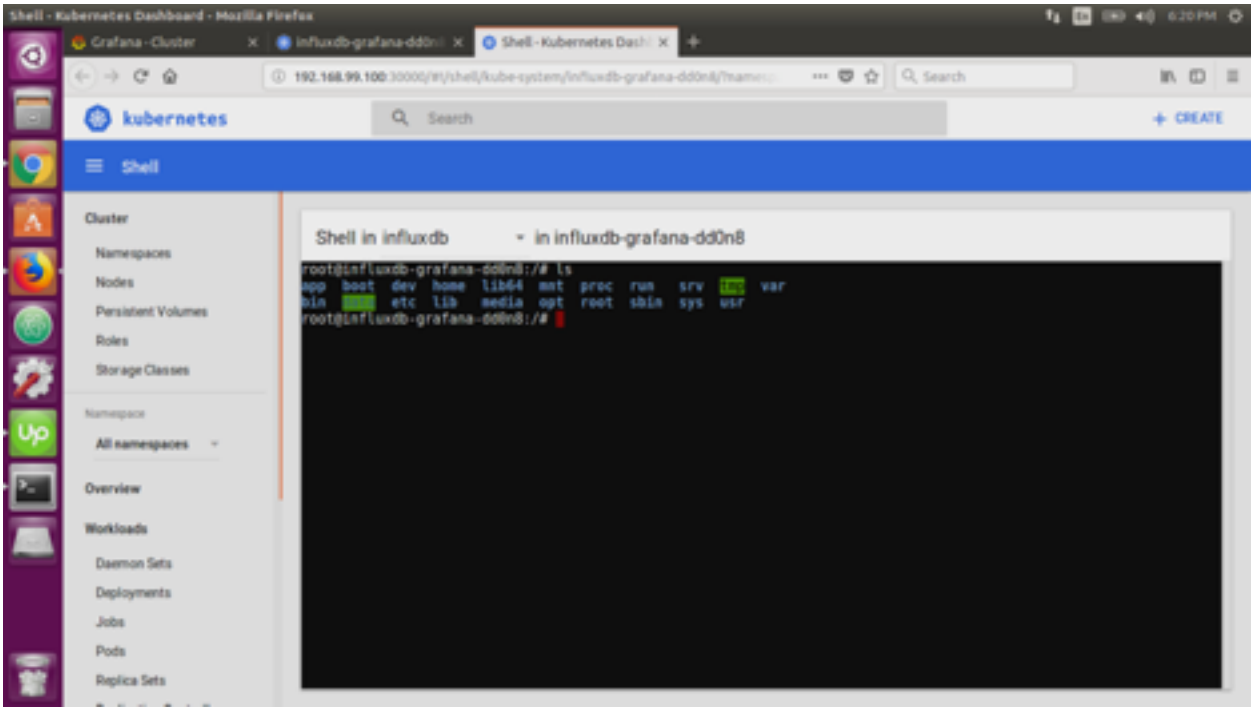


The Dashboard also gives the information of each workload. Following screenshot describes the Kubernetes Pod in detail.

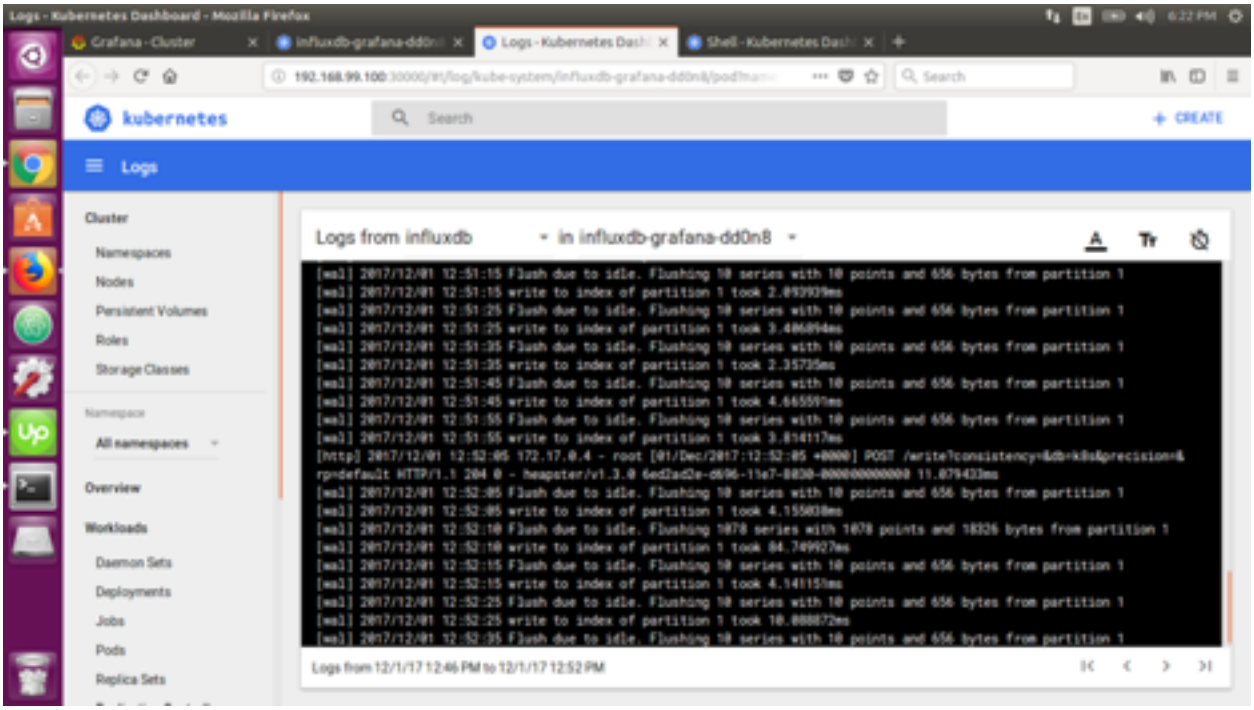


But the dashboard is just not limited to information, it's even possible to execute in the pod, get the logs of the pod, edit the pod and delete the pod.

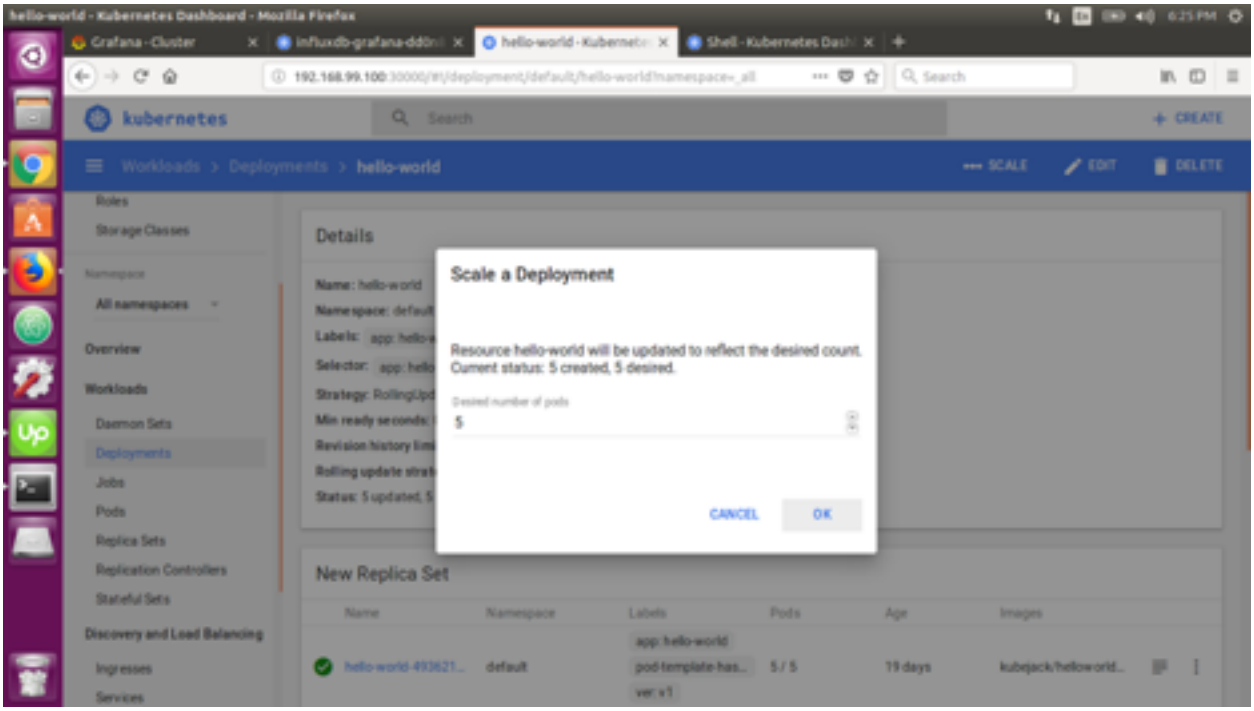
Exec inside the influxdb-grafana pod:



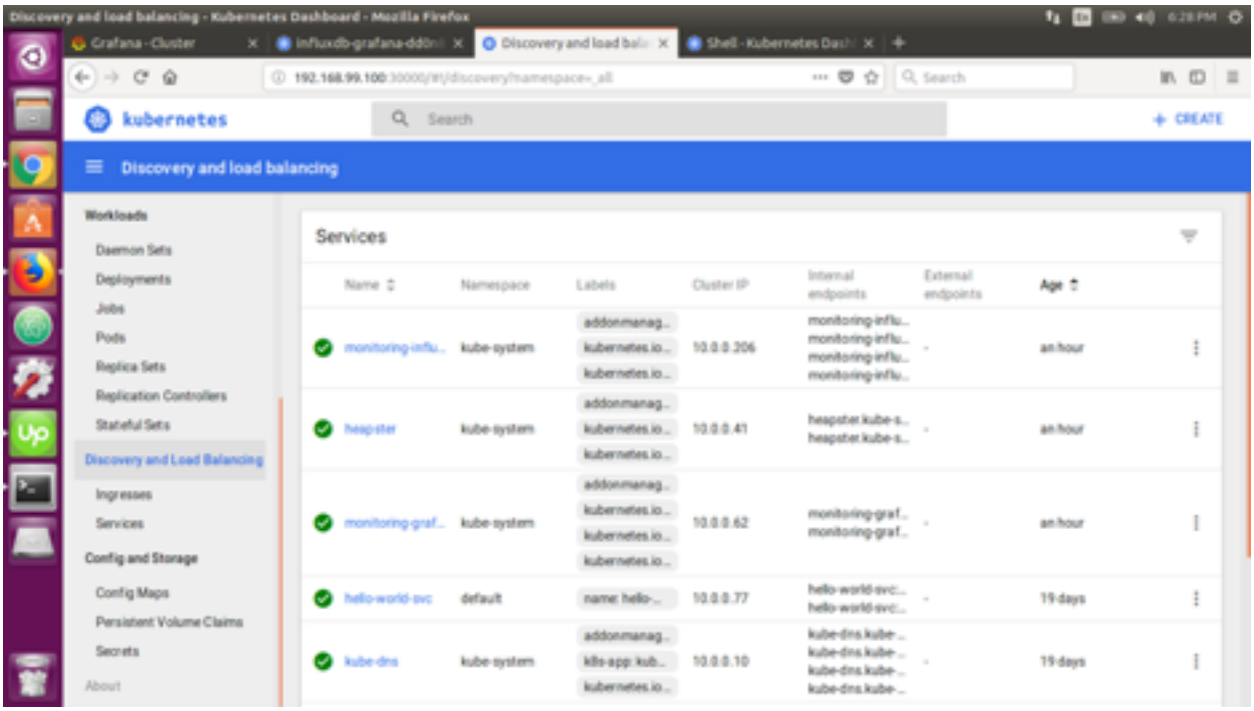
Logs of the influxdb-grafana pod:



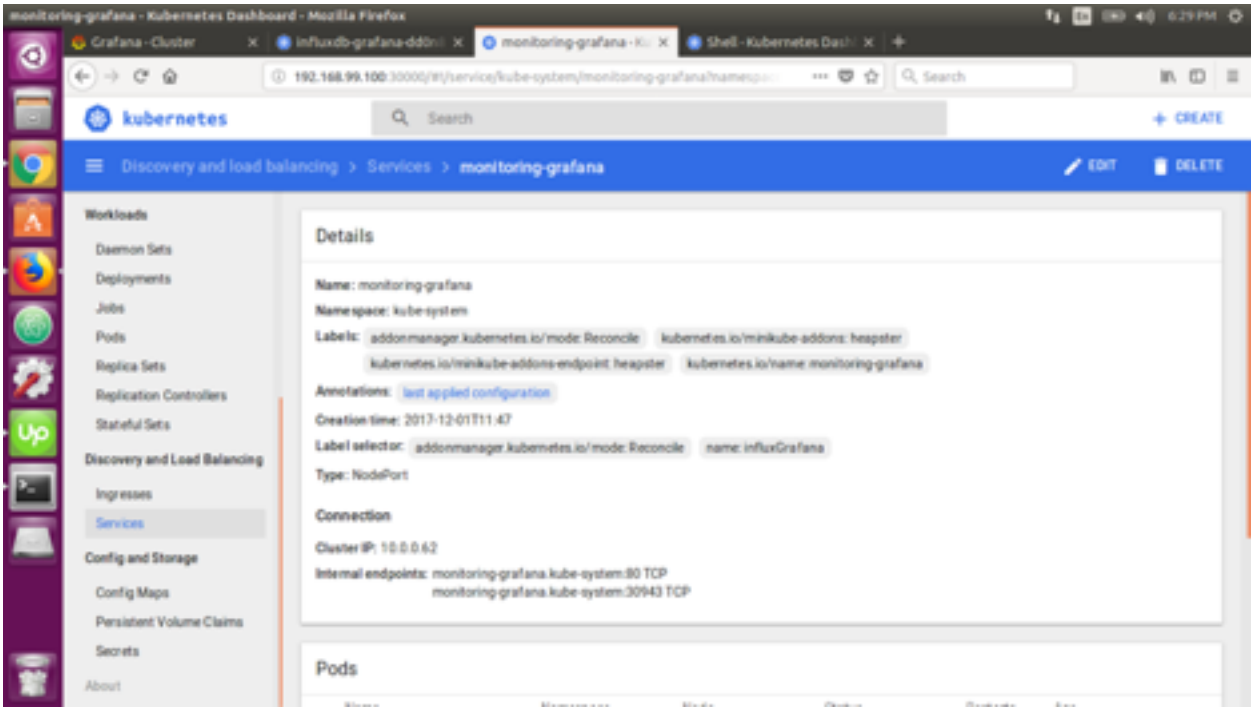
This is only for Pod. For other objects like deployments, it's possible to scale, edit and delete the deployment.



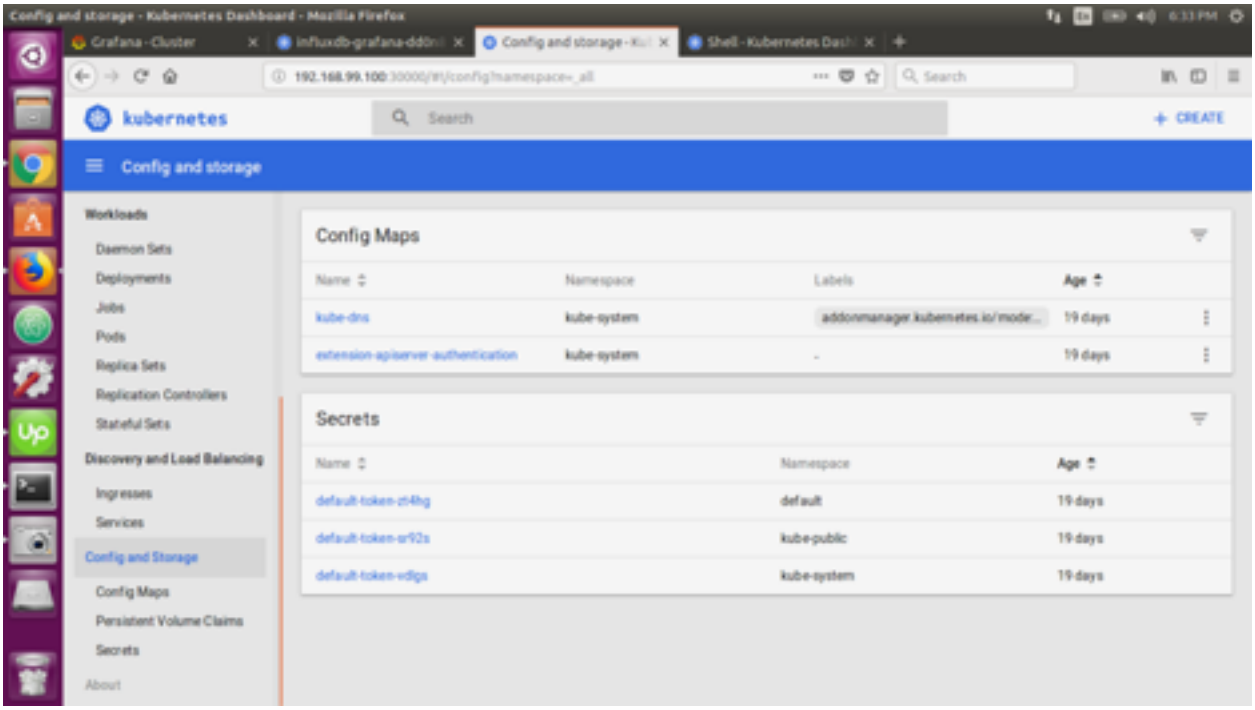
Discovery and Load Balancing consists the information about the Ingresses and Services.



The dashboard also provides the detail information about Ingresses and Services. It's possible to edit and delete the ingress or service through the dashboard.



Config and Storage consist the information about the Config Maps, Persistent Volume Claims and Secrets.



Also, it's possible to directly deploy the containerized application through web UI. The dashboard will require App name, Container Image, Number of Pod and Service inputs. Service can be internal or external service. The YAML file with required specifications can also be used for the creating the Kubernetes object.

In following example, Minikube addon is used for the Kubernetes dashboard.For manual installation following YAML should be used:

```
Kubectl create -f https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/recommended/kubernetes-dashboard.yaml
```

To access the Web UI,

Using Kubectl Proxy:

```
kubectl proxy
```

Using Kubernetes Master API Server:
https://<kubernetes-master>/ui

If the username and password are configured and unknown to you then use,

```
kubectl config view
```

Kubernetes dashboard is a flexible and reliable way to manage the Kubernetes Cluster.

Logging Kubernetes Cluster

Application and system level logs are useful to understand the problem with the system. It helps with troubleshooting and finding the root cause of the problem. Like application and system level logs containerized application also requires logs to be recorded and stored somewhere. The most standard method used for the logging is to write it to standard output and standard error streams. When the logs are recorded with separate storage then the mechanism is called as Cluster Level Logging.

Basic Logging with Kubernetes:

In the most basic logging it's possible to write the logs to the standard output using the Pod specification.

For Example:

```
apiVersion: v1
kind: Pod
metadata:
  name: counter
spec:
  containers:
  - name: count
    image: busybox
    args: [/bin/sh, -c,
          'i=0; while true; do echo "$i: $(date)"; i=$((i+1)); sleep 1; done']
```

The logs will be recorded with standard output:

```
$ kubectl create -f log-example-pod.yaml
pod "counter" created
$ kubectl get po
NAME                                READY     STATUS    RESTARTS   AGE
counter                             1/1       Running   0          8s
hello-world-493621601-1ffrp         1/1       Running   0          19d
hello-world-493621601-mmnzw         1/1       Running   0          19d
hello-world-493621601-nqd67         1/1       Running   0          19d
 hello-world-493621601-qkfcx        1/1       Running   0          19d
$ kubectl logs counter
0: Fri Dec  1 16:37:36 UTC 2017
1: Fri Dec  1 16:37:37 UTC 2017
2: Fri Dec  1 16:37:38 UTC 2017
3: Fri Dec  1 16:37:39 UTC 2017
4: Fri Dec  1 16:37:40 UTC 2017
5: Fri Dec  1 16:37:41 UTC 2017
6: Fri Dec  1 16:37:42 UTC 2017
7: Fri Dec  1 16:37:43 UTC 2017
8: Fri Dec  1 16:37:44 UTC 2017
```

Node level logging with Kubernetes:

The containerize application writes logs to stdout and stderr. Logging driver is the responsible for the writing log to the file in JSON format. In case of docker engine, docker logging driver is responsible for writing the log.

The most important part of Node level logging is log rotation with the Kubernetes. With the help of log rotation, it ensures the logs will not consume all storage space of the nodes.

Cluster Level Logging with Kubernetes:

Kubernetes does not provide the native cluster level logging. But the cluster level logging is possible with following approaches:

1. Run the agent on each node for log collection
2. Run the side container which will be responsible for log collection
3. Directly store the logs of the application into the storage

The most used and recommended method is using the node agent for log collection and storing the logs in log storage.

Stackdriver or Elasticsearch is used for the logging with Kubernetes. However, there are other solutions available like logz.io, sematext logging etc. Fluentd is used with custom configuration along with Stackdriver and Elasticsearch. Fluentd acts as the node agent.

For the Kubernetes Cluster created through minikube Giantswarm provides the solution. The solution consists ELK (Elasticsearch , logstash and Kibana) stack logging with minikube. However, it is possible to deploy all these components manually with manifests.

Start the Minikube:

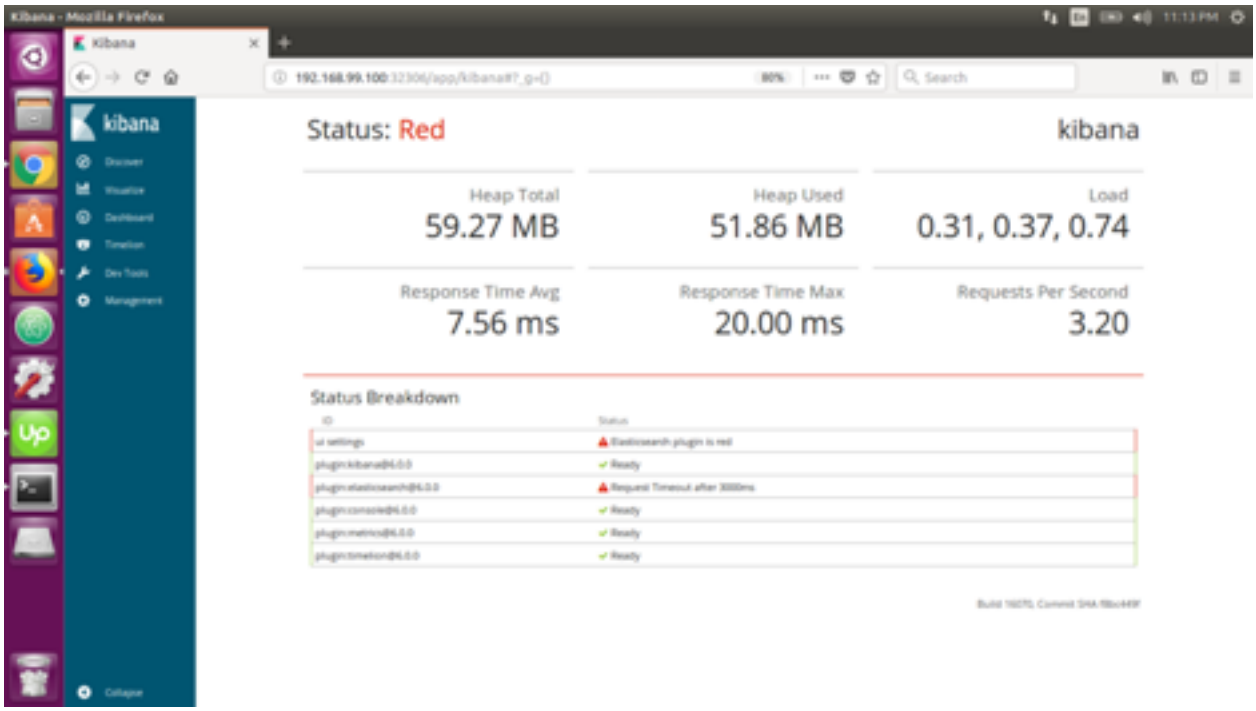
```
minikube start --memory 4096
```

Download all manifests and start Kibana:

```
kubectl apply \
  --filename https://raw.githubusercontent.com/giantswarm/kubernetes-elastic-stack/master/manifests-all.yaml

minikube service kibana
```

On the Kibana Dashboard :



The logging will be enabled and you can check it through Kibana dashboard. If you are using Google Kubernetes Engine, then stackdriver is a default logging option for GKE.

Upgrading Kubernetes

Upgrading the Kubernetes cluster is completely dependent on the platform. Here the platform is the type of installation you have followed for installing the Kubernetes Cluster. The solutions consist Google Cloud Engine, Google Kubernetes Engine, KOPS (Kubernetes Operations), CoreOS tectonic and Kubespray.

Apart from this, there are multiple solutions available including independent solution, hosted solution, and cloud-based solutions.

Upgrading Google Compute Engine Clusters:

If the cluster is created with cluster/gce/upgrade.sh script. To upgrade the master for specific version:

```
cluster/gce/upgrade.sh -M v1.0.2
```

Upgrade the entire cluster to the recent stable version:

```
cluster/gce/upgrade.sh release/stable
```

Upgrading Google Kubernetes Engine Clusters:

Google Kubernetes Engine automatically updates the master components. Example: kube-API server, kube-scheduler. It is also responsible for upgrading the operating system and other master components.

Upgrade Cluster of the Kubespray:

Kubespray provides ansible based the upgrade-cluster role. It consists the following YAML file.

Executing the following role will upgrade the components of the Kubernetes cluster.

upgrade-cluster.yml

```
---
- hosts: localhost
  gather_facts: False
  roles:
    - { role: kubespray-defaults}
    - { role: bastion-ssh-config, tags: ["localhost", "bastion"]}

- hosts: k8s-cluster:etcd:calico-rr
  any_errors_fatal: "{{ any_errors_fatal | default(true) }}"
```

```
  gather_facts: false
  vars:
    # Need to disable pipelining for bootstrap-os as some systems have
requiretty in sudoers set, which makes pipelining
    # fail. bootstrap-os fixes this on these systems, so in later plays it
can be enabled.
    ansible_ssh_pipelining: false
  roles:
    - { role: kubespray-defaults}
    - { role: bootstrap-os, tags: bootstrap-os}

- hosts: k8s-cluster:etcd:calico-rr
  any_errors_fatal: "{{ any_errors_fatal | default(true) }}"
  vars:
    ansible_ssh_pipelining: true
  gather_facts: true

- hosts: k8s-cluster:etcd:calico-rr
  any_errors_fatal: "{{ any_errors_fatal | default(true) }}"
  serial: "{{ serial | default('20%') }}"
  roles:
    - { role: kubespray-defaults}
    - { role: kubernetes/preinstall, tags: preinstall }
    - { role: docker, tags: docker }
    - role: rkt
      tags: rkt
      when: "'rkt' in [etcd_deployment_type, kubelet_deployment_type,
vault_deployment_type]"
    - { role: download, tags: download, skip_downloads: false }

- hosts: etcd:k8s-cluster:vault
  any_errors_fatal: "{{ any_errors_fatal | default(true) }}"
  roles:
```

```
    - { role: kubespray-defaults, when: "cert_management == 'vault'" }
    - { role: vault, tags: vault, vault_bootstrap: true, when: "cert_man-
agement == 'vault'" }

- hosts: etcd
  any_errors_fatal: "{{ any_errors_fatal | default(true) }}"
  roles:
    - { role: kubespray-defaults}
    - { role: etcd, tags: etcd, etcd_cluster_setup: true }

- hosts: k8s-cluster
  any_errors_fatal: "{{ any_errors_fatal | default(true) }}"
  roles:
    - { role: kubespray-defaults}
    - { role: etcd, tags: etcd, etcd_cluster_setup: false }

- hosts: etcd:k8s-cluster:vault
  any_errors_fatal: "{{ any_errors_fatal | default(true) }}"
  roles:
    - { role: kubespray-defaults, when: "cert_management == 'vault'"}
    - { role: vault, tags: vault, when: "cert_management == 'vault'"}

#Handle upgrades to master components first to maintain backwards
compat.
- hosts: kube-master
  any_errors_fatal: "{{ any_errors_fatal | default(true) }}"
  serial: 1
  roles:
    - { role: kubespray-defaults}
    - { role: upgrade/pre-upgrade, tags: pre-upgrade }
    - { role: kubernetes/node, tags: node }
    - { role: kubernetes/master, tags: master }
    - { role: kubernetes/client, tags: client }
    - { role: kubernetes-apps/cluster_roles, tags: cluster-roles }
```

```
    - { role: network_plugin, tags: network }
    - { role: upgrade/post-upgrade, tags: post-upgrade }

#Finally handle worker upgrades, based on given batch size
- hosts: kube-node:!kube-master
  any_errors_fatal: "{{ any_errors_fatal | default(true) }}"
  serial: "{{ serial | default('20%') }}"
  roles:
    - { role: kubespray-defaults}
    - { role: upgrade/pre-upgrade, tags: pre-upgrade }
    - { role: kubernetes/node, tags: node }
    - { role: network_plugin, tags: network }
    - { role: upgrade/post-upgrade, tags: post-upgrade }
    - { role: kubernetes/kubeadm, tags: kubeadm, when: "kubeadm_en-
abled" }
    - { role: kubespray-defaults}

- hosts: kube-master[0]
  any_errors_fatal: true
  roles:
    - { role: kubespray-defaults}
    - { role: kubernetes-apps/rotate_tokens, tags: rotate_tokens, when:
"secret_changed|default(false)" }

- hosts: kube-master
  any_errors_fatal: true
  roles:
    - { role: kubespray-defaults}
    - { role: kubernetes-apps/network_plugin, tags: network }
    - { role: kubernetes-apps/policy_controller, tags: policy-controller }
    - { role: kubernetes/client, tags: client }

- hosts: calico-rr
```

```
  any_errors_fatal: "{{ any_errors_fatal | default(true) }}"
  roles:
    - { role: kubespray-defaults}
    - { role: network_plugin/calico/rr, tags: network }

- hosts: k8s-cluster
  any_errors_fatal: "{{ any_errors_fatal | default(true) }}"
  roles:
    - { role: kubespray-defaults}
    - { role: dnsmasq, when: "dns_mode == 'dnsmasq_kubedns'", tags:
dnsmasq }
    - { role: kubernetes/preinstall, when: "dns_mode != 'none' and re-
solvconf_mode == 'host_resolvconf'", tags: resolvconf }

- hosts: kube-master[0]
  any_errors_fatal: "{{ any_errors_fatal | default(true) }}"
  roles:
    - { role: kubespray-defaults}
    - { role: kubernetes-apps, tags: apps }
```

It will upgrade the Kubernetes Kubespray installation of the Kubernetes Cluster.

9. Kubernetes Maintenance

- Monitoring Kubernetes Cluster
- Managing Kubernetes with Dashboard
- Logging Kubernetes Cluster
- Upgrading Kubernetes

Monitoring Kubernetes Cluster

For reliable applications, it is required to have in place monitoring of the Kubernetes Cluster. It helps to determine availability, scalability, and reliability of the application deployed over Kubernetes Cluster.

Heapster aggregator is used for monitoring and event the logs. Heapster stores the information in storage backend. Currently, it supports Google Cloud Monitoring and InfluxDB as the storage backends. Heapster runs as a pod in the Kubernetes Cluster. It communicates with each node of the Kubernetes Cluster. Kubelet agent is responsible to provide the monitoring information to Heapster. Kubelet itself collects the data from cAdvisor.

cAdvisor:
cAdvisor is an open source container usage and performance analysis agent. In Kubernetes cAdvisor is included into Kubelet binary.cAdvisor auto-discovers all containers. It collects the information of CPU, memory, network and file system usage statistics.

Kubelet:
Kubelet bridge the gap between Kubernetes Master and Kubernetes Node. It manages the Pods and Containers on each machine.

InfluxDB and Grafana are used for storing the data and visualizing it. Google cloud monitoring provides the hosted solution for monitoring Kubernetes Cluster. Heapster can be set up to send the metrics to Google Cloud monitoring.

Let's check the Monitoring Kubernetes Cluster created with Minikube:

Kubernetes Cluster created locally by minikube supports addons.

```
$ minikube addons list
- addon-manager: enabled
- dashboard: enabled
- kube-dns: enabled
- default-storageclass: enabled
- heapster: disabled
- ingress: disabled
- registry: disabled
- registry-creds: disabled
```

Enable the addon:

```
$ minikube addons enable heapster
```

To open the web interface:

```
$ minikube addons open heapster
```

The result will be displayed on the grafana.

In Minikube addons are helping for monitoring but it's also possible to add heapster as Kubernetes deployment. This will be the manual installation of heapster, grafana and influxdb.

Following is the heapster.yaml :

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: heapster
  namespace: kube-system
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: heapster
  namespace: kube-system
spec:
  replicas: 1
  template:
    metadata:
```

```
    labels:
      task: monitoring
      k8s-app: heapster
  spec:
    serviceAccountName: heapster
    containers:
    - name: heapster
      image: gcr.io/google_containers/heapster-amd64:v1.4.0
      imagePullPolicy: IfNotPresent
      command:
      - /heapster
      - --source=kubernetes:https://kubernetes.default
      - --sink=influxdb:http://monitoring-influxdb.kube-system.
svc:8086
---
apiVersion: v1
kind: Service
metadata:
 labels:
   task: monitoring
   # For use as a Cluster add-on (https://github.com/kubernetes/
kubernetes/tree/master/cluster/addons)
   # If you are NOT using this as an addon, you should comment out
this line.
   kubernetes.io/cluster-service: 'true'
   kubernetes.io/name: Heapster
 name: heapster
 namespace: kube-system
spec:
 ports:
 - port: 80
   targetPort: 8082
 selector:
   k8s-app: heapster
```

You can get the latest version of the heapster at https://github.com/kubernetes/heapster/ .

Using Kubectl:

```
$ kubectl create -f heapster.yaml
```

For grafana, use grafana.yaml:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: monitoring-grafana
  namespace: kube-system
spec:
  replicas: 1
  template:
    metadata:
      labels:
        task: monitoring
        k8s-app: grafana
    spec:
      containers:
      - name: grafana
        image: gcr.io/google_containers/heapster-grafana-amd64:v4.4.3
        ports:
        - containerPort: 3000
          protocol: TCP
        volumeMounts:
        - mountPath: /etc/ssl/certs
          name: ca-certificates
          readOnly: true
```

```
        - mountPath: /var
          name: grafana-storage
        env:
        - name: INFLUXDB_HOST
          value: monitoring-influxdb
        - name: GF_SERVER_HTTP_PORT
          value: "3000"
          # The following env variables are required to make Grafana
accessible via
          # the kubernetes api-server proxy. On production clusters, we
recommend
          # removing these env variables, setup auth for grafana, and
expose the grafana
          # service using a LoadBalancer or a public IP.
        - name: GF_AUTH_BASIC_ENABLED
          value: "false"
        - name: GF_AUTH_ANONYMOUS_ENABLED
          value: "true"
        - name: GF_AUTH_ANONYMOUS_ORG_ROLE
          value: Admin
        - name: GF_SERVER_ROOT_URL
          # If you're only using the API Server proxy, set this value instead:
          # value: /api/v1/namespaces/kube-system/services/monitoring-
grafana/proxy
          value: /
      volumes:
      - name: ca-certificates
        hostPath:
          path: /etc/ssl/certs
      - name: grafana-storage
        emptyDir: {}
---
apiVersion: v1
```

```
kind: Service
metadata:
  labels:
    # For use as a Cluster add-on (https://github.com/kubernetes/
kubernetes/tree/master/cluster/addons)
    # If you are NOT using this as an addon, you should comment out
this line.
    kubernetes.io/cluster-service: 'true'
    kubernetes.io/name: monitoring-grafana
  name: monitoring-grafana
  namespace: kube-system
spec:
  # In a production setup, we recommend accessing Grafana through
an external Loadbalancer
  # or through a public IP.
  # type: LoadBalancer
  # You could also use NodePort to expose the service at a randomly-
generated port
  # type: NodePort
  ports:
  - port: 80
    targetPort: 3000
  selector:
    k8s-app: grafana
```

You can get the latest version of grafana.yaml at https://github.com/kubernetes/heapster/blob/master/deploy/kube-config/influxdb/grafana.yaml .

Using kubectl :

```
$ kubectl create -f grafana.yaml
```

If influxdb is the storage backend, then use following YAML for deploying influxdb in Kubernetes Cluster:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: monitoring-influxdb
  namespace: kube-system
spec:
  replicas: 1
  template:
    metadata:
      labels:
        task: monitoring
        k8s-app: influxdb
    spec:
      containers:
      - name: influxdb
        image: gcr.io/google_containers/heapster-influxdb-amd64:v1.3.3
        volumeMounts:
        - mountPath: /data
          name: influxdb-storage
      volumes:
      - name: influxdb-storage
        emptyDir: {}
---
apiVersion: v1
kind: Service
metadata:
  labels:
    task: monitoring
    # For use as a Cluster add-on (https://github.com/kubernetes/
kubernetes/tree/master/cluster/addons)
```

```
    # If you are NOT using this as an addon, you should comment out
this line.
    kubernetes.io/cluster-service: 'true'
    kubernetes.io/name: monitoring-influxdb
  name: monitoring-influxdb
  namespace: kube-system
spec:
  ports:
  - port: 8086
    targetPort: 8086
  selector:
    k8s-app: influxdb
```

You can get the latest version of influxdb.yaml at https://github.com/kubernetes/heapster/blob/master/deploy/kube-config/influxdb/influxdb.yaml

Using Kubectl:

```
$ kubectl create -f influxdb.yaml
```

To access grafana dashboard with the manual setup, describe the grafana service and check endpoint of the service.

To describe the service using Kubectl use following command:

```
  minikube-addons...
  Selector:
  addonmanager.kubernetes.io/mode=Recon-
  cile,name=influxGrafana
  Type:                   NodePort
  IP:                     10.0.0.62
  Port:                   <unset>                 80/TCP
```

```
NodePort:          <unset>          30943/TCP
Endpoints:         172.17.0.9:3000
Session Affinity:
Events:            <none>
```

Prometheus and Data Dog are also good tools for monitoring the Kubernetes Cluster.

Managing Kubernetes with Dashboard

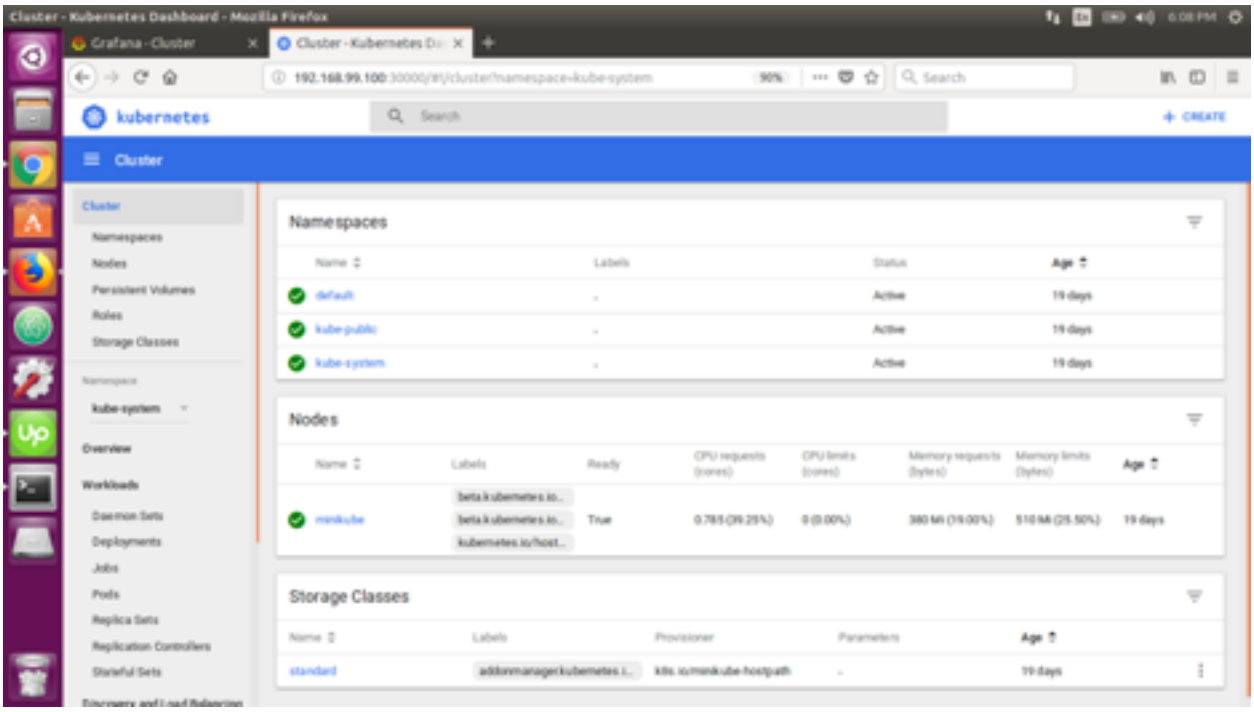
Kubernetes provides the web interface to manage the Kubernetes Cluster. With the Role Based Access Policy, it becomes simpler and easier to manage Kubernetes Cluster with Dashboard. For Minikube it gives the addon. That means Minikube provides addons for monitoring, dashboard and for managing the cluster.

To enable the addon of the dashboard on minikube:

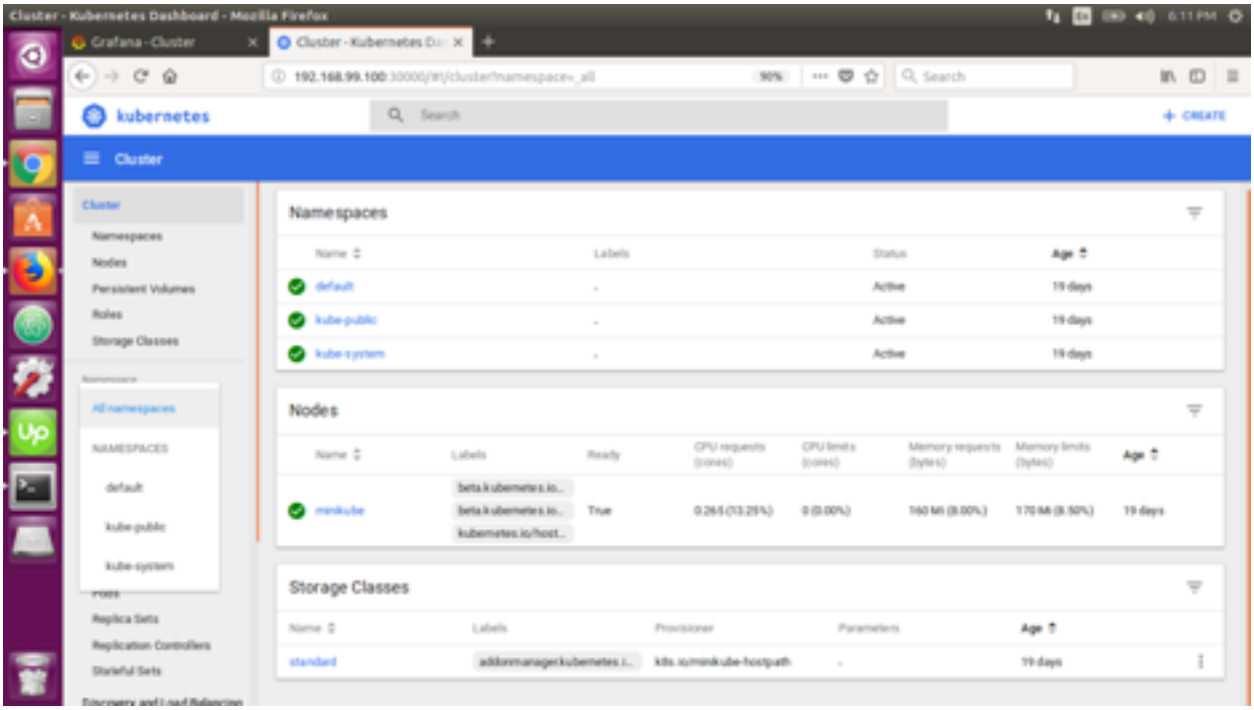
```
$ minikube addons enable dashboard
dashboard was successfully enabled
$ minikube addons open dashboard
Opening kubernetes service kube-system/kubernetes-dashboard in
default browser...
```

With Minikube, it will open the dashboard in the browser. The dashboard consists the Cluster infromation, Namespace information and information about the objects associated with cluster like workloads etc.The dashboard consists the three different informative sections including workloads, discovery and load balancing, config and storage.

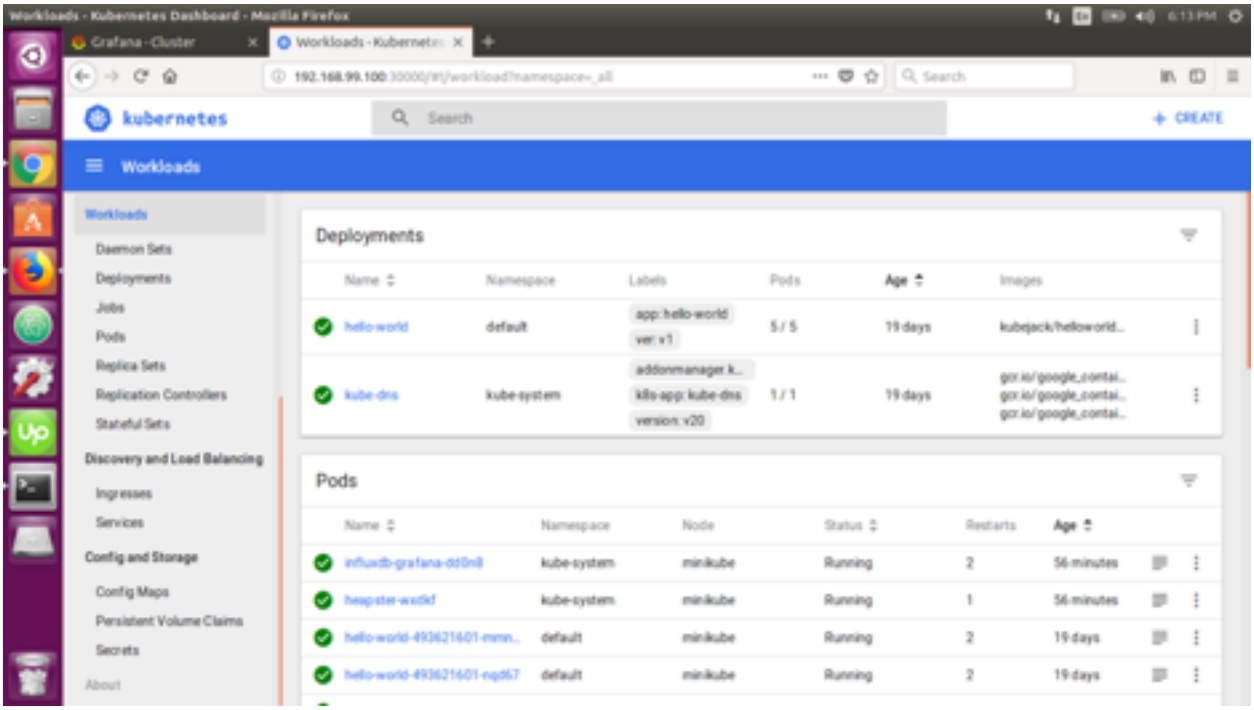
In the cluster tab, Kubernetes dashboard shows the information about the Namespaces, Nodes, Persistent Volumes and Storage Classes.



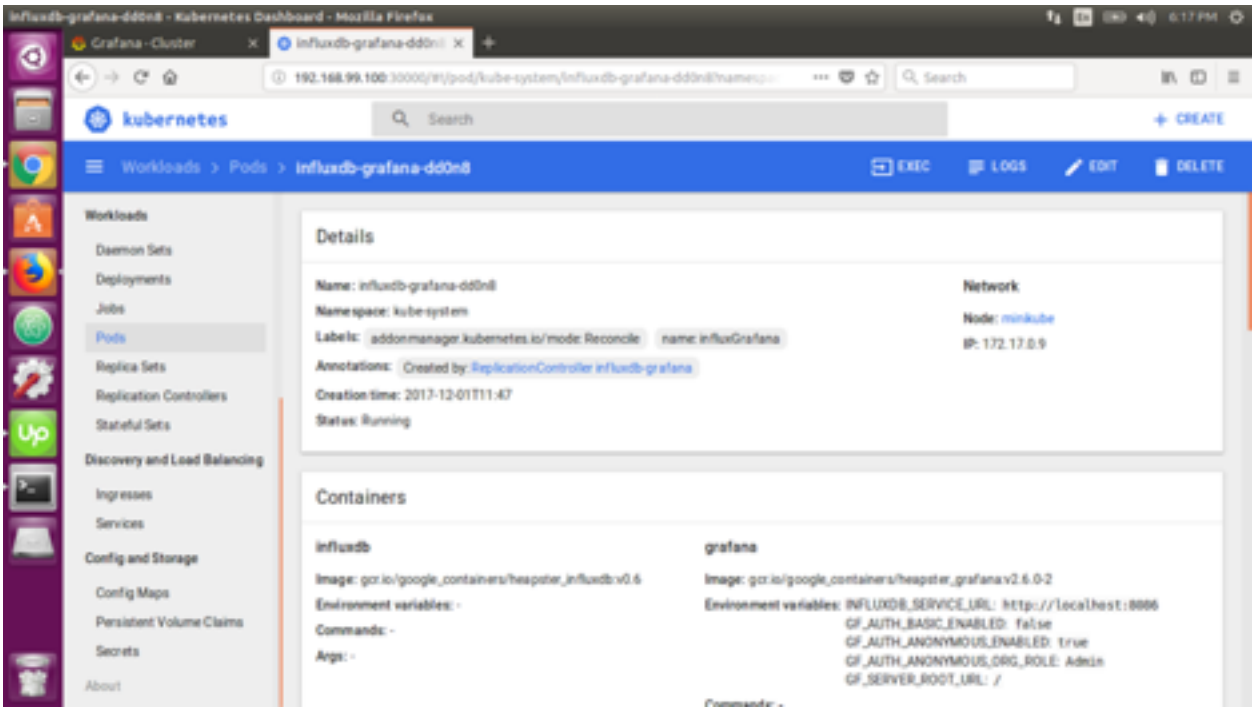
It gives the brief information about the cluster. In namespaces, it shows the all available namespaces in Kubernetes Cluster. It's possible to select all namespaces or specific namespace



Depending on the namespace selected previously, further tabs show in depth information of associated Kubernetes object within selected namespace. It consists the workloads of Kubernetes which includes Deployments, Replica Sets, Replication Controller, Daemon Sets, Jobs, Pods and Stateful Sets. Each of this workload is important to run an application over the Kubernetes Cluster.

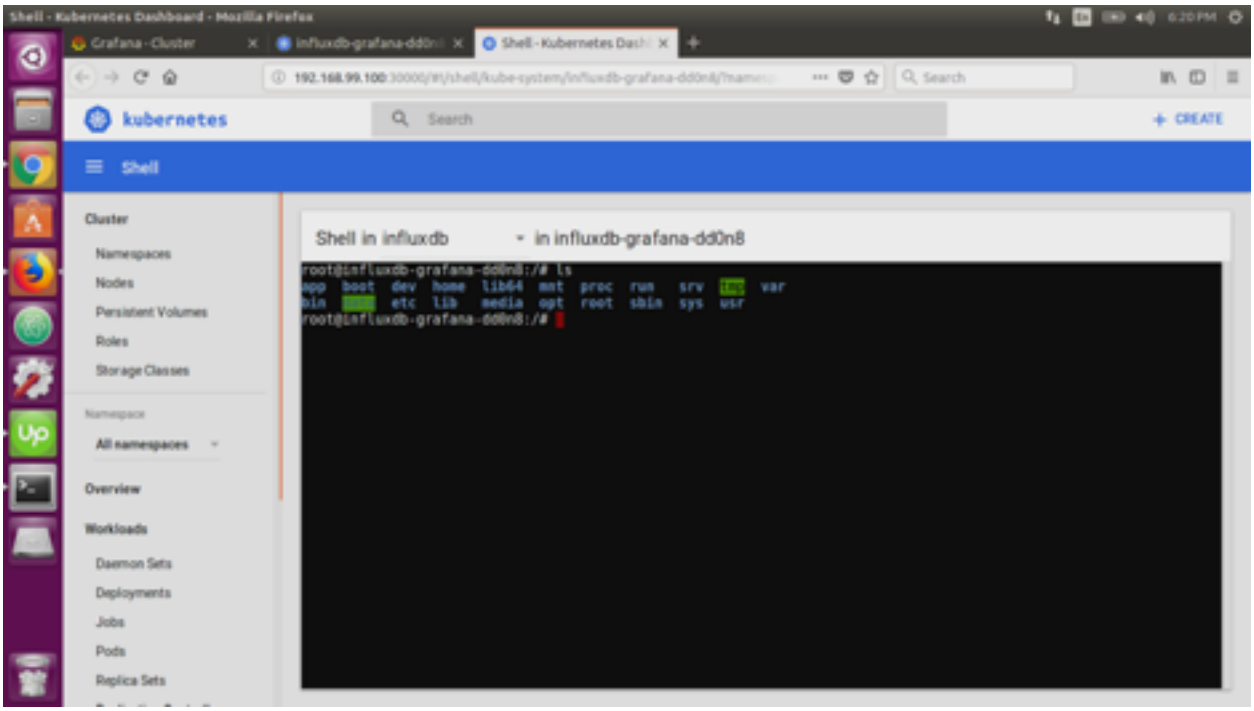


The Dashboard also gives the information of each workload. Following screenshot describes the Kubernetes Pod in detail.

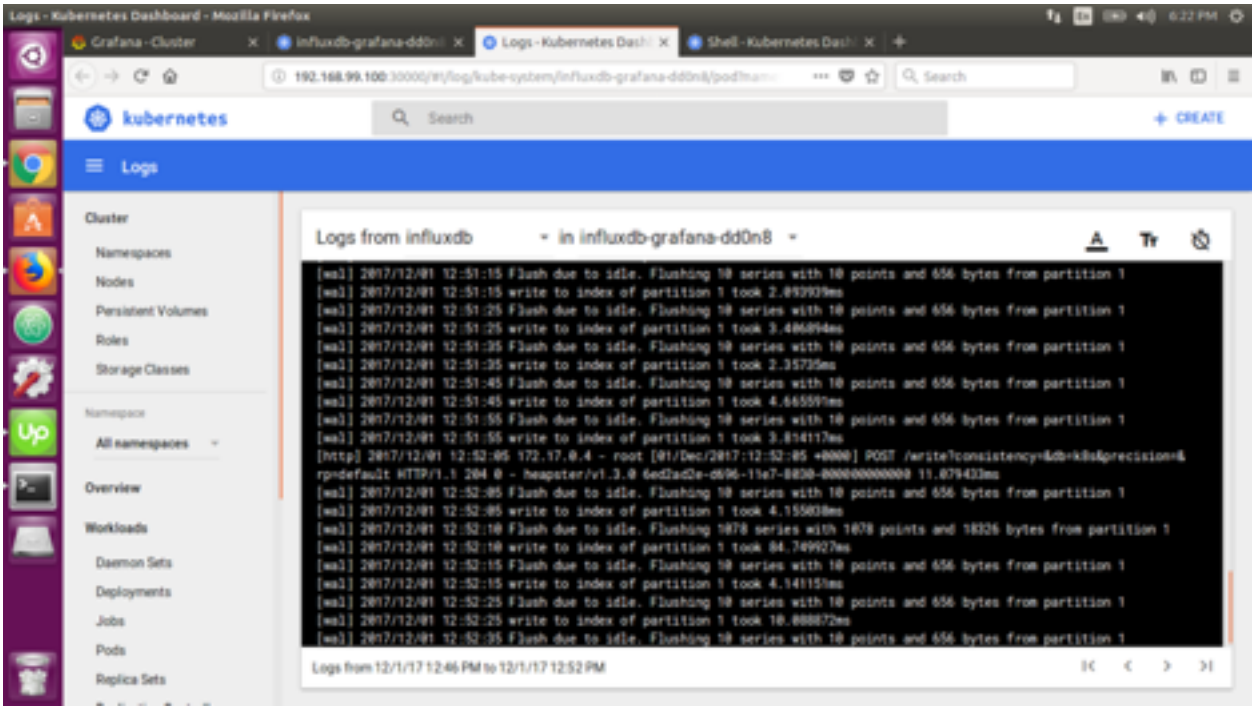


But the dashboard is just not limited to information, it's even possible to execute in the pod, get the logs of the pod, edit the pod and delete the pod.

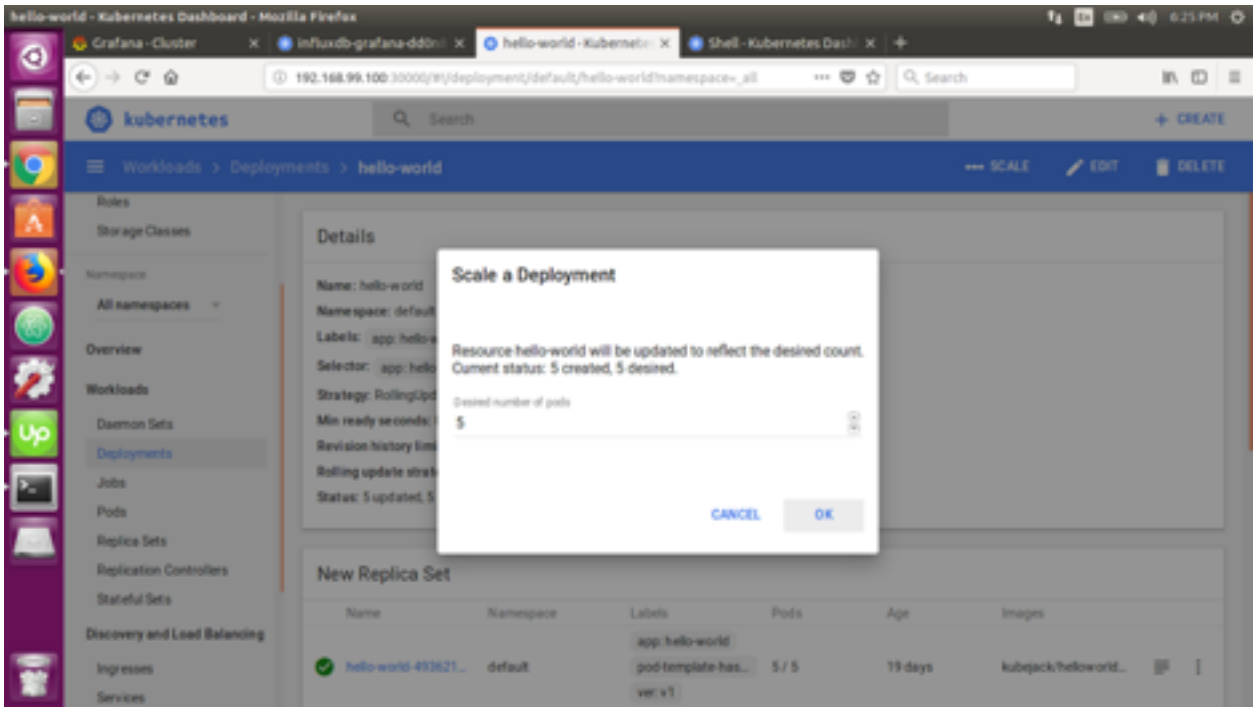
Exec inside the influxdb-grafana pod:



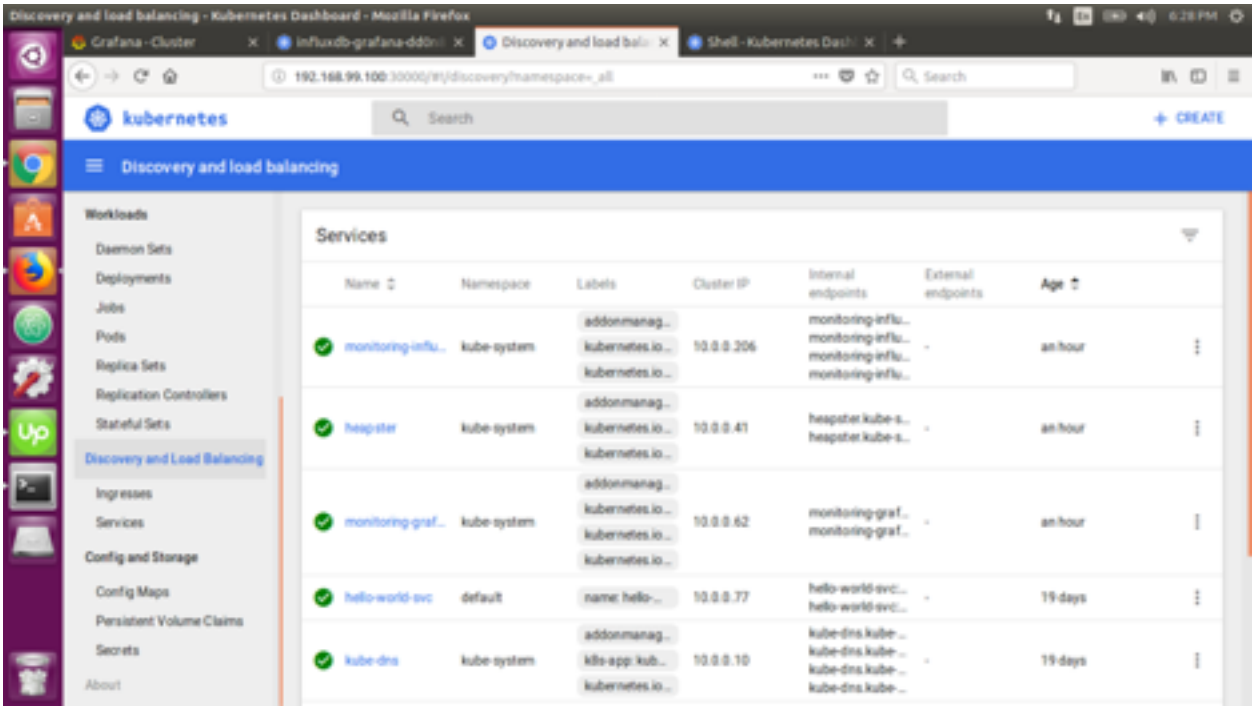
Logs of the influxdb-grafana pod:



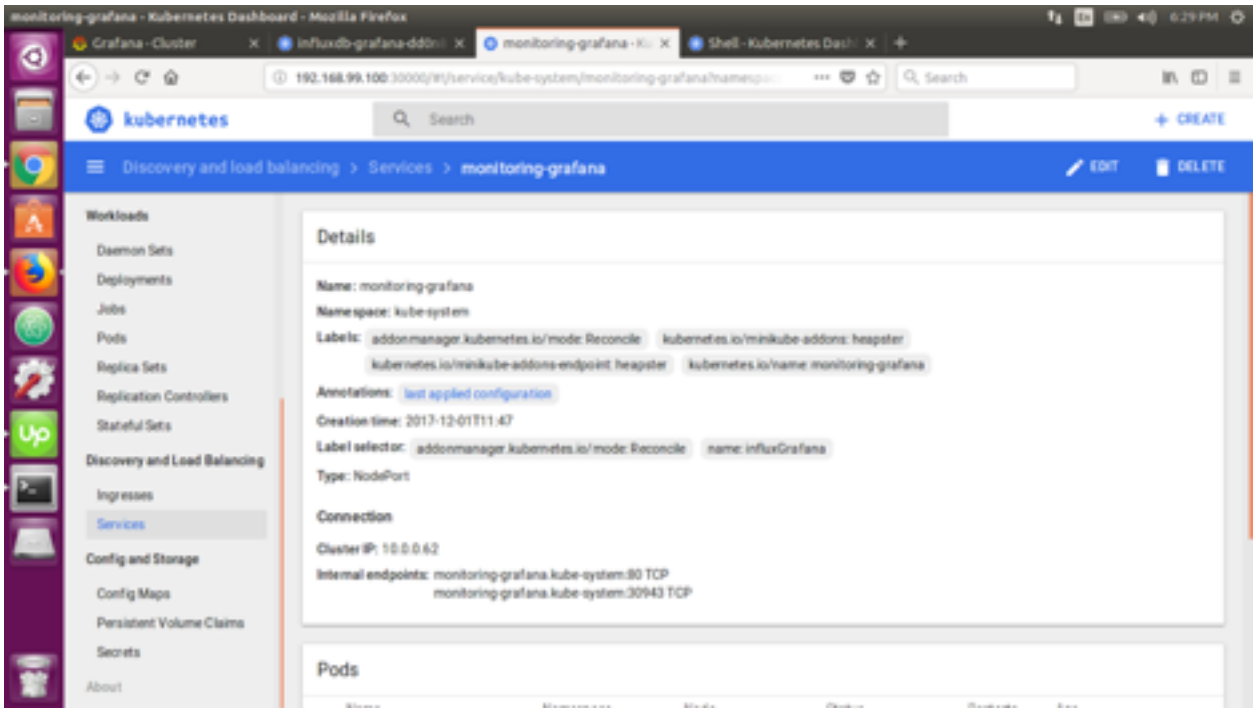
This is only for Pod. For other objects like deployments, it's possible to scale, edit and delete the deployment.



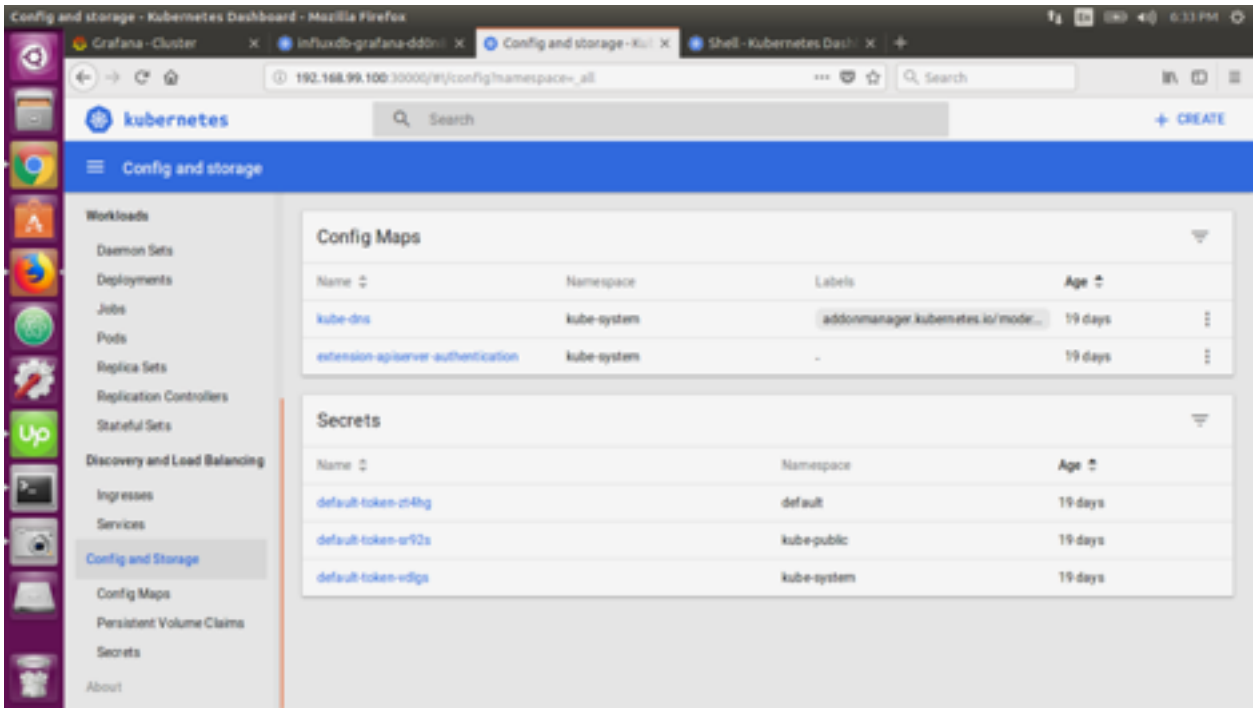
Discovery and Load Balancing consists the information about the Ingresses and Services.



The dashboard also provides the detail information about Ingresses and Services. It's possible to edit and delete the ingress or service through the dashboard.



Config and Storage consist the information about the Config Maps, Persistent Volume Claims and Secrets.



Also, it's possible to directly deploy the containerized application through web UI. The dashboard will require App name, Container Image, Number of Pod and Service inputs. Service can be internal or external service. The YAML file with required specifications can also be used for the creating the Kubernetes object.

In following example, Minikube addon is used for the Kubernetes dashboard.For manual installation following YAML should be used:

```
Kubectl create -f https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/recommended/kubernetes-dashboard.yaml
```

To access the Web UI,

Using Kubectl Proxy:

```
kubectl proxy
```

Using Kubernetes Master API Server:
https://<kubernetes-master>/ui

If the username and password are configured and unknown to you then use,

```
kubectl config view
```

Kubernetes dashboard is a flexible and reliable way to manage the Kubernetes Cluster.

Logging Kubernetes Cluster

Application and system level logs are useful to understand the problem with the system. It helps with troubleshooting and finding the root cause of the problem. Like application and system level logs containerized application also requires logs to be recorded and stored somewhere. The most standard method used for the logging is to write it to standard output and standard error streams. When the logs are recorded with separate storage then the mechanism is called as Cluster Level Logging.

Basic Logging with Kubernetes:

In the most basic logging it's possible to write the logs to the standard output using the Pod specification.

For Example:

```
apiVersion: v1
kind: Pod
metadata:
  name: counter
spec:
  containers:
  - name: count
    image: busybox
    args: [/bin/sh, -c,
          'i=0; while true; do echo "$i: $(date)"; i=$((i+1)); sleep 1; done']
```

The logs will be recorded with standard output:

```
$ kubectl create -f log-example-pod.yaml
pod "counter" created
$ kubectl get po
```

```
NAME                                READY   STATUS    RESTARTS   AGE
counter                             1/1     Running   0          8s
hello-world-493621601-1ffrp         1/1     Running   3          19d
hello-world-493621601-mmnzw         1/1     Running   3          19d
hello-world-493621601-nqd67         1/1     Running   3          19d
hello-world-493621601-qkfcx         1/1     Running   3          19d
hello-world-493621601-xbf6s         1/1     Running   3          19d
$ kubectl logs counter
0: Fri Dec  1 16:37:36 UTC 2017
1: Fri Dec  1 16:37:37 UTC 2017
2: Fri Dec  1 16:37:38 UTC 2017
3: Fri Dec  1 16:37:39 UTC 2017
4: Fri Dec  1 16:37:40 UTC 2017
5: Fri Dec  1 16:37:41 UTC 2017
6: Fri Dec  1 16:37:42 UTC 2017
7: Fri Dec  1 16:37:43 UTC 2017
8: Fri Dec  1 16:37:44 UTC 2017
```

Node level logging with Kubernetes:

The containerize application writes logs to stdout and stderr. Logging driver is the responsible for the writing log to the file in JSON format. In case of docker engine, docker logging driver is responsible for writing the log.

The most important part of Node level logging is log rotation with the Kubernetes. With the help of log rotation, it ensures the logs will not consume all storage space of the nodes.

Cluster Level Logging with Kubernetes:

Kubernetes does not provide the native cluster level logging. But the cluster level logging is possible with following approaches:

1. Run the agent on each node for log collection
2. Run the side container which will be responsible for log collection
3. Directly store the logs of the application into the storage

The most used and recommended method is using the node agent for log collection and storing the logs in log storage.

Stackdriver or Elasticsearch is used for the logging with Kubernetes. However, there are other solutions available like logz.io, sematext logging etc. Fluentd is used with custom configuration along with Stackdriver and Elasticsearch. Fluentd acts as the node agent.

For the Kubernetes Cluster created through minikube Giantswarm provides the solution. The solution consists ELK (Elasticsearch , logstash and Kibana) stack logging with minikube. However, it is possible to deploy all these components manually with manifests.

Start the Minikube:

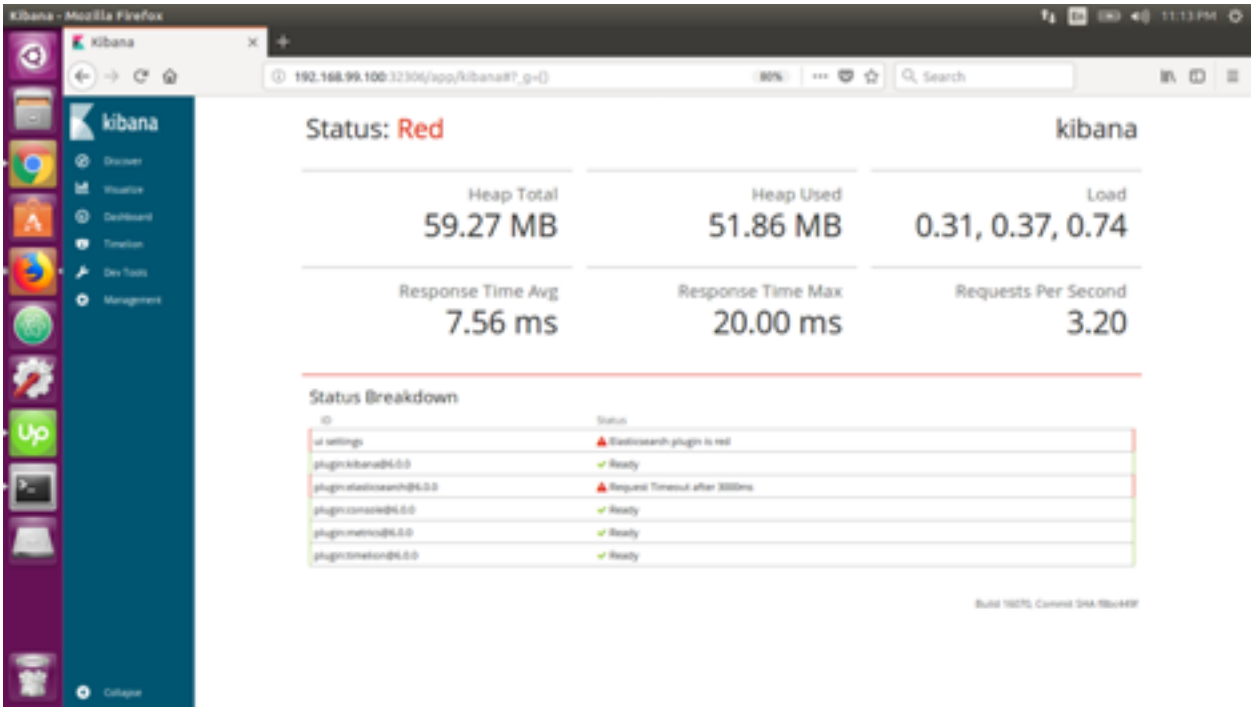
```
minikube start --memory 4096
```

Download all manifests and start Kibana:

```
kubectl apply \
  --filename https://raw.githubusercontent.com/giantswarm/
kubernetes-elastic-stack/master/manifests-all.yaml

minikube service kibana
```

On the Kibana Dashboard :



The logging will be enabled and you can check it through Kibana dashboard. If you are using Google Kubernetes Engine, then stackdriver is a default logging option for GKE.

Upgrading Kubernetes

Upgrading the Kubernetes cluster is completely dependent on the platform. Here the platform is the type of installation you have followed for installing the Kubernetes Cluster. The solutions consist Google Cloud Engine, Google Kubernetes Engine, KOPS (Kubernetes Operations), CoreOS tectonic and Kubespray.

Apart from this, there are multiple solutions available including independent solution, hosted solution, and cloud-based solutions.

Upgrading Google Compute Engine Clusters:

If the cluster is created with cluster/gce/upgrade.sh script. To upgrade the master for specific version:

```
cluster/gce/upgrade.sh -M v1.0.2
```

Upgrade the entire cluster to the recent stable version:

```
cluster/gce/upgrade.sh release/stable
```

Upgrading Google Kubernetes Engine Clusters:

Google Kubernetes Engine automatically updates the master components.Example: kube-API server, kube-scheduler. It is also responsible for upgrading the operating system and other master components.

Upgrade Cluster of the Kubespray:

Kubespray provides ansible based the upgrade-cluster role. It consists the following YAML file.
Executing the following role will upgrade the components of the Kubernetes cluster.
upgrade-cluster.yml

```
---
- hosts: localhost
  gather_facts: False
  roles:
    - { role: kubespray-defaults}
    - { role: bastion-ssh-config, tags: ["localhost", "bastion"]}

- hosts: k8s-cluster:etcd:calico-rr
  any_errors_fatal: "{{ any_errors_fatal | default(true) }}"
  gather_facts: false
  vars:
    # Need to disable pipelining for bootstrap-os as some systems have
requiretty in sudoers set, which makes pipelining
    # fail. bootstrap-os fixes this on these systems, so in later plays it
can be enabled.
    ansible_ssh_pipelining: false
  roles:
    - { role: kubespray-defaults}
    - { role: bootstrap-os, tags: bootstrap-os}

- hosts: k8s-cluster:etcd:calico-rr
  any_errors_fatal: "{{ any_errors_fatal | default(true) }}"
  vars:
    ansible_ssh_pipelining: true
  gather_facts: true

- hosts: k8s-cluster:etcd:calico-rr
  any_errors_fatal: "{{ any_errors_fatal | default(true) }}"
  serial: "{{ serial | default('20%') }}"
  roles:
    - { role: kubespray-defaults}
    - { role: kubernetes/preinstall, tags: preinstall }
    - { role: docker, tags: docker }
```

```
    - role: rkt
      tags: rkt
      when: "'rkt' in [etcd_deployment_type, kubelet_deployment_type,
vault_deployment_type]"
    - { role: download, tags: download, skip_downloads: false }

- hosts: etcd:k8s-cluster:vault
  any_errors_fatal: "{{ any_errors_fatal | default(true) }}"
  roles:
    - { role: kubespray-defaults, when: "cert_management == 'vault'" }
    - { role: vault, tags: vault, vault_bootstrap: true, when: "cert_
management == 'vault'" }

- hosts: etcd
  any_errors_fatal: "{{ any_errors_fatal | default(true) }}"
  roles:
    - { role: kubespray-defaults}
    - { role: etcd, tags: etcd, etcd_cluster_setup: true }

- hosts: k8s-cluster
  any_errors_fatal: "{{ any_errors_fatal | default(true) }}"
  roles:
    - { role: kubespray-defaults}
    - { role: etcd, tags: etcd, etcd_cluster_setup: false }

- hosts: etcd:k8s-cluster:vault
  any_errors_fatal: "{{ any_errors_fatal | default(true) }}"
  roles:
    - { role: kubespray-defaults, when: "cert_management == 'vault'"}
    - { role: vault, tags: vault, when: "cert_management == 'vault'"}

#Handle upgrades to master components first to maintain backwards
compat.
- hosts: kube-master
```

```
  any_errors_fatal: "{{ any_errors_fatal | default(true) }}"
  serial: 1
  roles:
    - { role: kubespray-defaults}
    - { role: upgrade/pre-upgrade, tags: pre-upgrade }
    - { role: kubernetes/node, tags: node }
    - { role: kubernetes/master, tags: master }
    - { role: kubernetes/client, tags: client }
    - { role: kubernetes-apps/cluster_roles, tags: cluster-roles }
    - { role: network_plugin, tags: network }
    - { role: upgrade/post-upgrade, tags: post-upgrade }

#Finally handle worker upgrades, based on given batch size
- hosts: kube-node:!kube-master
  any_errors_fatal: "{{ any_errors_fatal | default(true) }}"
  serial: "{{ serial | default('20%') }}"
  roles:
    - { role: kubespray-defaults}
    - { role: upgrade/pre-upgrade, tags: pre-upgrade }
    - { role: kubernetes/node, tags: node }
    - { role: network_plugin, tags: network }
    - { role: upgrade/post-upgrade, tags: post-upgrade }
    - { role: kubernetes/kubeadm, tags: kubeadm, when: "kubeadm_
enabled" }
    - { role: kubespray-defaults}

- hosts: kube-master[0]
  any_errors_fatal: true
  roles:
    - { role: kubespray-defaults}
    - { role: kubernetes-apps/rotate_tokens, tags: rotate_tokens, when:
"secret_changed|default(false)" }

- hosts: kube-master
```

```
  any_errors_fatal: true
  roles:
    - { role: kubespray-defaults}
    - { role: kubernetes-apps/network_plugin, tags: network }
    - { role: kubernetes-apps/policy_controller, tags: policy-controller }
    - { role: kubernetes/client, tags: client }

- hosts: calico-rr
  any_errors_fatal: "{{ any_errors_fatal | default(true) }}"
  roles:
    - { role: kubespray-defaults}
    - { role: network_plugin/calico/rr, tags: network }

- hosts: k8s-cluster
  any_errors_fatal: "{{ any_errors_fatal | default(true) }}"
  roles:
    - { role: kubespray-defaults}
    - { role: dnsmasq, when: "dns_mode == 'dnsmasq_kubedns'", tags:
dnsmasq }
    - { role: kubernetes/preinstall, when: "dns_mode != 'none' and
resolvconf_mode == 'host_resolvconf'", tags: resolvconf }

- hosts: kube-master[0]
  any_errors_fatal: "{{ any_errors_fatal | default(true) }}"
  roles:
    - { role: kubespray-defaults}
    - { role: kubernetes-apps, tags: apps }
```

It will upgrade the Kubernetes Kubespray installation of the Kubernetes Cluster.