Building cloud native Machine learning based applications

Mr. Bhushan Hiralal Patil
Mr. Mohan Nikam
Mr. Arunvel A
M.Tech, Department of Computing Science and Engineering,
Sandip University, Nashik
CDAC ACTS - PUNE

***Abstract-* Cloud computing has an emerged as animportant relevence to improve resource utilization, efficiency, flexibility, However, cloud platforms cause performance degradations due to their virtualization layer and may not be appropriate for the requirements of high performance applications, such as machine learning. The Research tackles the problem of improving network performance in container based cloud instances to create a viable alternative to run network intensive machine learning applications.**

Our approach consists of deploying container based machine learning based apps via LXC (Linux Container) cloud instances to increase the available bandwidth & performance. so to evaluate the efficiency of this approach and the overhead added by the container-based cloud environment, we ran a set of experiments to measure throughput, latency, bandwidth utilization, and completion times. The outcomes proves that this approach adds minimum overhead in cloud environment as well as increases throughput and reduces latency.

Containers are a lightweight virtualization solution to replace virtual machines for deploying cloud applications as they are less resource and time consuming. Easy deployment of applications on containers is done using kubernetes which helps user to analyze applications with various network topologies improving transfer rate of large data sets which are required for machine learning models.

I. INTRODUCTION

Today, cloud computing is rapidly developing and large scale application. As a kind of network-based computing, cloud computing can provide shared software and hardware resource to users by using network, which realizes the reasonable distribution of information and resource. Meanwhile, cloud computing brings along the changes of the traditional data center, which produces a new generation of data center: cloud data center. By building computing resources, storage resources and network resources into dynamic virtual resource pool using virtualization technology But now cloud data center faces with many problems such as low resource utilization, application and platform can not be decoupled, application runtime environment limitations are strong, and operational staff control decrease, which limit the development of it.

Virtual machines (VMs) are an infrastructure as a service (IaaS) focusing on hardware virtualization whose techniques have been used at the infrastructure layer. To achieve sharing and elasticity of resources, the cloud makes use of such virtualization techniques for administering scheduling, provisioning and security. However, VMs suffer from slow start up time and exhibit lower densities even when full. Applications occupy entire host operating system (OS) in VMs and there are also several limitations on storage and Containers offer as a lightweight virtualization solution to deploying applications across various domains and sectors. They allow infrastructure and platform to be shared in a secure and portable manner, along with application packaging and management. They facilitate faster deployment of applications and have faster start up times. They are less resource and time consuming and can be scaled up or down providing higher density levels than full VMs. They facilitate easier and portable across infrastructure deployment of applications in an interoperable way. Using containers will accelerate agile application development of distributed applications providing an additional layer of protection by isolating applications and the host, without using incremental resources. They also allow easy updates to applications. Fig. 1 shows the difference between traditional hypervisor and container-based architectures

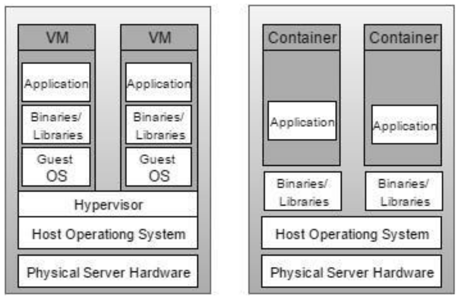


Fig.1 hypervisor architecture and a container-based architecture

Container technology is a virtualization technology at the operating system level, it can provide a separate file system, network and process context space for each running server without modifying the host operating environment. So that, each service can arbitrarily change the contents of the operating system files, configuration of network and user in their own virtual running space, without damage to real host system files. Although an instance of the operating system is shared between containers, they run independently and without interference. Besides, do not relaying on the integrity of the operating system makes container more lightweight and fast start than virtual machine, which meets the needs of cloud data center. Cloud computing infrastructures support rapid resource provisioning, which in turn is a suitable option to deploy a container environment. Primarily due to the advances of cloud platforms, there are possibilities to develop new business models based on the pay-per-use billing structure. Also, there are cloud computing technologies (specifically those focused on network and storage) that provide lower performance degradation. However, the use of cloud environments for processing distributed applications has traditionally been avoided when it requires high bandwidth and throughput as well as low latency. Traditional cloud providers, which use virtualization technologies that are not optimized for the execution of distributed applications, add significant overheads. Our main challenge in this paper is to improve network performance in the container-based cloud environment for applications. We chose to increase the network performance by using the IEEE 802.3ad link aggregation standard. It gives and uses a standard method to combine multiple physical links that can be used as a single logical link. The standard is a layer 2 control protocol that can be used to automatically detect, configure, and manage a single logical link with multiple physical links between two adjacent enabled devices. Thus, link aggregation provides higher availability and capacity while network performance improvements are obtained using existing hardware (IEEE 802.3ad requires support in a network switch). Also, our goal is to target a high-performance cloud environment by deploying container-based instances, where applications may run. We chose container-based technologies to enable multiple isolated Linux systems to run on a single host through Namespaces (providing isolated user environments in the form of containers) and cgroups (providing resource management and accounting). LXC (Linux Container) technology is an important part of this cloud infrastructure, because it is a free software that provides a powerful set of user space tools and utilities to manage Linux containers

II. RELATED WORK

In this section, we provide a brief background about machines placement, Docker containers, Docker Container Networking. We also described related work reported in the literature with regards to performance of Docker containers when compared to the performance of VMs. ALSO we should focus on container network models. There are two mainstream models called Container Network Model (CNM) and Container Network Interface (CNI).

A. Virtual machine placement:

Nowadays, cloud computing has been widely used.The Virtual Machines (VMs) are created on servers in cloud computing. With the rapid development of Cloud computing technologies, the management of Cloud resources has become crucial to Cloud Service Providers (CSPs). CSPs provide on demand services to the end users through provisioning Virtual Machines (VMs) in their available Physical Machines (PM)s. In order to reduce the Total Cost of Ownership(TCO), CSPs consolidate multiple VMs in each of the available PMs. In this context, significant number of research work have already been done which indicate the benefit of running multiple VMs in a PM. The VM scheduling on servers for energy saving in the cloud computing has been studied. The Virtual Machine Scheduling Algorithm (VSA) is proposed to schedule VMs in cluster environments. However, it is not effective and has high time complexity. The VM scheduling on servers for energy saving inthe cloud computing has been studied. The Virtual Machine Scheduling Algorithm (VSA) is we propose a new scheduling method called Energy-aware Virtual Machine Placement (EVP)method to schedule VMs that can reduce power consumption complexity. Power consumption of a PM depends on the number of VMs deployed, reserved resources of VMs, and the state ofthe processes running within the VMs. There are significant number of works that try to minimize the the overall power consumption of a data-center by reducing the number of active PMs. They classify this problem of mapping the VMs to minimum number of PMs satisfying the constraints of physical resources and the requirements of the VMs as the VM Placement Problem

B. Docker Container:

Docker containers are lightweight, highly portable and scalable. Such features become attractive to develop containerized services (or microservices). In the literature, container-based services are reported to always outperform VM-based services. For example, it was shown in that containers outperform virtual machines in terms of execution time, latency, throughput, power consumption, CPU utilization and memory usage. However, according to Amazon AWS documentation. it was reported that Docker containers are deployed on top of virtual machines (VMs), and not on bare-metal hardware. This contradicts the common practice of container deployment which is widely adopted in the literature of deploying container on bare-metal hardware. This was the primary motivation for our research work in which we aim to investigate and assess the performance difference resulting from deploying services using VMs (virtual machines) and using Docker containers

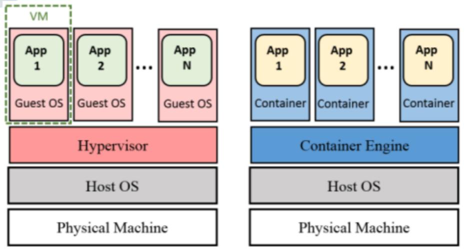


Fig. 2 Common approach of hosting of VMs vs. Containers

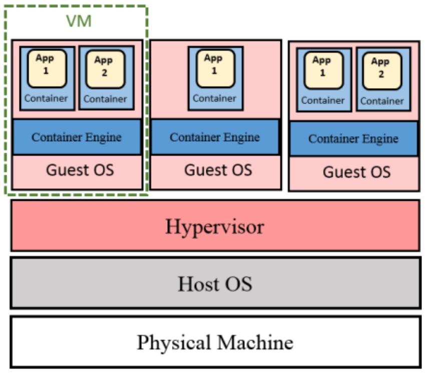
The Services deployed using containers are expected to take less execution time thus resulting in less latency over virtual machines as in. In addition, containers were shown to introduce lower power consumption over virtualized technologies.

C. Docker Container Networking:

As an important part of container technology, container network mainly solves the problem of communication between containers in the case of ensure container isolation. Since container is a new technology, container network is not yet mature, and many container network solutions were proposed to solve the problems of container network. Each solution has its own characteristics, and how to choose appropriate solution according to different scenarios is not only very important for the efficiency of communication between containers but also of great significance in improving cloud data center performance. In order to provide experiences and references for choosing and deploying appropriate container network, we introduce three kinds of mainstream container network solutions (Flannel, Docker Swarm Overlay, Calico), and design experiments to measure and evaluate the performance of them

III. LITERATURE REVIEW

This section describes the primary related works aimed at improving the network performance and deploying Container Based instances or Services. Our first scenario consists of evaluating network performance and then second scenario consist deploying Container based ML services The Authors Wattanasomboon, P., & Somchit, Y. (2015, October)[1] proposes the method called EVP(Energy-Aware virtual machine placement) to method to schedule VMs that can reduce power consumption. We also formulate power consumption model to evaluate the performance of the EVP method. Finally, we evaluate the EVP method by simulation. On The basis of experimental results show that the EVP method has better performance. There are many researches of VM scheduling for energy saving. Befor using EVP method for our approach, we have to check performance of our VMs and containers and choose which platform is better to to deploy the ML(machine learning)Services.the Salah, Tasneem, et al.[5] stated the performace comparison between VMs and Containers and gave theirs results as Container always outperforms the VM in terms of the parameters like Cpu Utilization, latency, Throughput, Memory usage, Execution Time etc. So Services deployed using containers are expected to take less execution time thus resulting in less latency over virtual machines as in



Fig,3 Hosting of Containers in Amazon cloud

Furthermore, according to Amazon , Docker containers are deployed on top of virtual machine, which is against the common practice of deployment as depicted in Fig.A&B and Fig.3 shows the container deployment approach adopted by Amazon cloud.

IV. PROPOSED METHODOLOGY

This approach involves the accessing of services that are stored in containers and checking the maintenance of computing nodes. These services are designed to perform two types of tasks: first accessing the all configuration, dependancies of machine learning services and second is deploying container based ML services in cloud Enviornment.

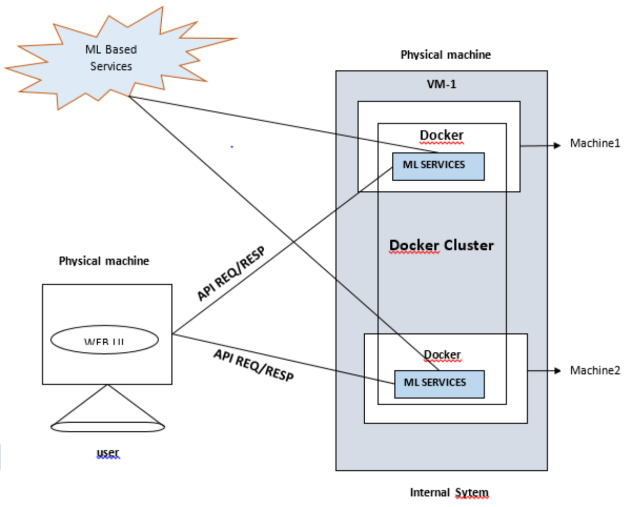


Fig.4 Proposed Architecture

The above Figure shows proposed architecture ,in that we make the use of VMs over PMs and Configured all the necessary packeges related to Docker. after docker installed, we create the cluster of docker running the command - docker Swarm init on one of machine and then token is generated for adding or joing workers/slaves to master,so we used that token for forming a docker cluster. so this is how cluster is created. once cluster is formed, we are trying to build containers which contains the set of ML services.so on top of docker ,we deploy container-based ML services on cloud using deployment technology (eg: using kubernetes). for deployment we served container-based ML services as a API using Keras and Flask.We then put that application inside of a Docker container, uploaded the container to Docker Hub, and deployed it with Kubernetes.this is how the workflow and implementation of proposed methodology is done So the machine learning services or applications that I'm builded and deployed is nothing but the object detections in a images. It finds out object in the given image that where is the object in images and what is ratio of object and maps it. The ML services is built on top of python 3.6 (docker) and has these packages built into it. The packeges are:

- Tensorflow — 1.7
- Tensorflow Object detection
- Keras
- Scipy
- Matplotlib
- Numpy
- Pandas
- Sklearn
- Open CV
- Scikit Image
- Pillow
- Jupyter Notebook

V. IMPLEMENTATION

Following steps are listed and certain tools are used for developing this project.The Tools That are required to build and deploy container based ML services are follows

A. Docker Cluster:

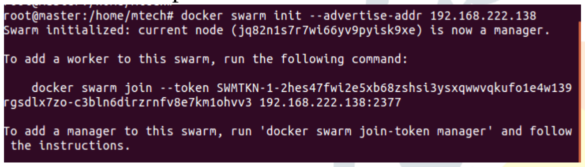
Docker is a tool designed to make it easier to create, deploy, and run applications by using containers.In a way, Docker is a bit like a virtual machine. But unlike a virtual machine, rather than creating a whole virtual operating system, Docker allows applications to use the same Linux kernel as the system that they're running on and only requires applications be shipped with things not already running on the host computer. This gives a significant performance boost and reduces the size of the application.So, to gain boost in performance we create the cluster of docker. crating cluster of docker is done by using Docker swarm. Docker Swarm is a clustering and scheduling tool for Docker containers. With Swarm, IT administrators and developers can establish and manage a cluster of Docker nodes as a single virtual system.you will also need to install the Docker engine on both nodes. By default, Docker is not available in the Ubuntu 16.04 repository. Requirements:

- Hypervisor
- Docker engine
- Docker swarm

First, you will need to initialize the cluster with the IP address, so your node acts as a Manager node. On the Manager Node, run the following command for advertising IP address:

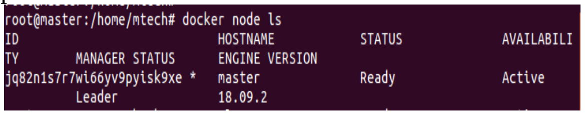
~docker swarm init --advertise-addr 192.168.222.138

You should see output like this:



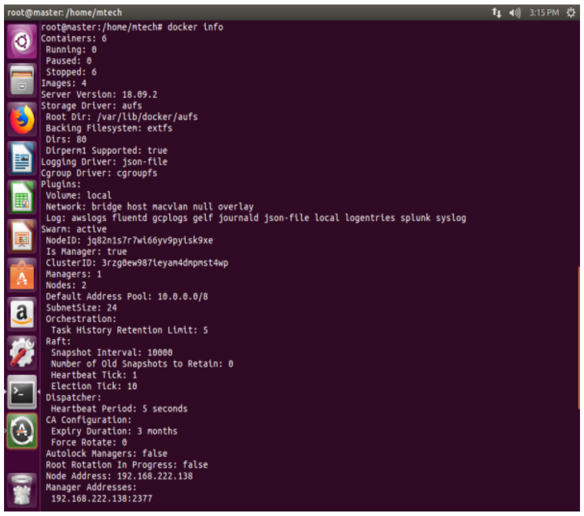
```
root@master:/home/mtech# docker swarm init --advertise-addr 192.168.222.138
Swarm initialized: current node (jq82n1s7r7wi66yv9pyisk9xe) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-2hes47fwi2e5xb68zshsi3ysxqwwvqkufo1e4w139
rgsdlx7zo-c3bln6dirzrnfv8e7km1ohvv3 192.168.222.138:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow
 the instructions.
```

The token shown in the above output will be used to add worker nodes to the cluster in next step. The Docker Engine joins the swarm depending on the join-token you provide to the docker swarm join command. The node only uses the token at join time. If you subsequently rotate the token, it doesn't affect existing swarm nodes. Now, check the status of the Manager Node with the following command: docker node ls If everything is fine, you should see the following output:



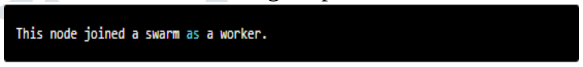
```
root@master:/home/mtech# docker node ls
ID                          HOSTNAME            STATUS              AVAILABILI
TY        MANAGER STATUS    ENGINE VERSION
jq82n1s7r7wi66yv9pyisk9xe *  master             Ready               Active
          Leader            18.09.2
```

You can also check the status of the Docker Swarm Cluster: code>*docker info*



```
root@master:/home/mtech# docker info
Containers: 6
 Running: 0
 Paused: 0
 Stopped: 6
Images: 4
Server Version: 18.09.2
Storage Driver: aufs
 Root Dir: /var/lib/docker/aufs
 Backing Filesystem: extfs
 Dirs: 80
 Dirperm1 Supported: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
 Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
Swarm: active
 NodeID: jq82n1s7r7wi66yv9pyisk9xe
 Is Manager: true
 ClusterID: 3rzg0ew987ieyam4dmpmst4wp
 Managers: 1
 Nodes: 2
 Default Address Pool: 10.0.0.0/8
 SubnetSize: 24
 Orchestration:
  Task History Retention Limit: 5
 Raft:
  Snapshot Interval: 10000
  Number of Old Snapshots to Retain: 0
  Heartbeat Tick: 1
  Election Tick: 10
 Dispatcher:
  Heartbeat Period: 5 seconds
 CA Configuration:
  Expiry Duration: 3 months
  Force Rotate: 0
 Autolock Managers: false
 Root Rotation In Progress: false
 Node Address: 192.168.222.138
 Manager Addresses:
  192.168.222.138:2377
```

Manager Node (master) is now configured properly, it's time to add Worker Node(slave) to the Swarm Cluster. First, copy the output of the "swarm init" command from the previous step, then paste that output on the Worker Node to join the Swarm Cluster: *docker swarm join --token SWMTKN-1-5p5f6p6tv1cmjzq9ntx3zmck9kpgt355qq0uaqoj2ple629dl458 80qso8jio78djpx5mzbqcfu192.168.222.138:2377*
You should see the following output:

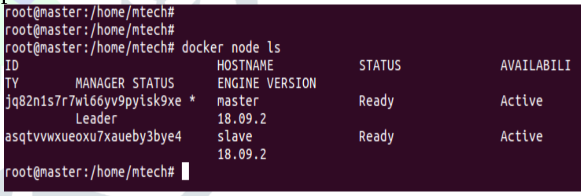


```
This node joined a swarm as a worker.
```

Now, on the Manager Node(master), run the following command to list the Worker Node(slave):

~docker node ls

You should see the Worker Node(slave) in the following output



```
root@master:/home/mtech#
root@master:/home/mtech#
root@master:/home/mtech# docker node ls
ID                          HOSTNAME            STATUS              AVAILABILI
TY        MANAGER STATUS    ENGINE VERSION
jq82n1s7r7wi66yv9pyisk9xe *  master             Ready               Active
          Leader            18.09.2
asqtvvwxueoxu7xaueby3bye4    slave              Ready               Active
                            18.09.2

root@master:/home/mtech# 
```

Fig A. Docker Cluster

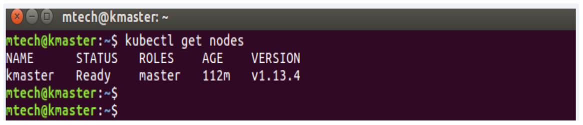
Docker Swarm Cluster is now up and running, it's time to launch the web service inside Docker Swarm Mode. On the Manager Node, run the commands to deploy webservice

B. Configuring Kubernetes:

Kubernetes (commonly stylized as k8s) is an open-source container orchestration system for automating application deployment, scaling, and management. It was originally designed by Google, and is now maintained by the Cloud Native Computing Foundation. It aims to provide a platform for automating deployment, scaling, and operations of application containers across clusters of hosts. It works with a range of container tools, including Docker. Many cloud services offer a Kubernetes-based platform or infrastructure as a service (PaaS or IaaS) on which Kubernetes can be deployed as a platform-providing service.

C. Installing Kubernetes:

After Docker, we install kubernetes on top of docker to deploy Containerized applications,kuberenetes is configured is shown by the readiness state.the below images shown that

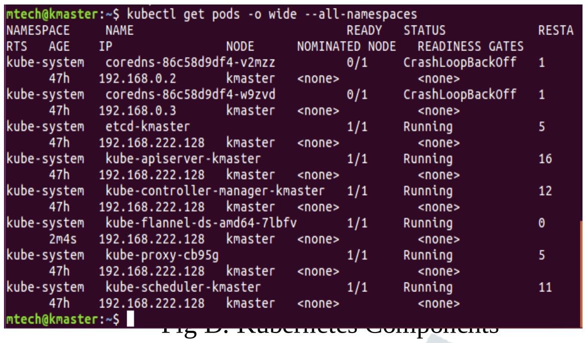


```
mtech@kmaster: ~
mtech@kmaster:~$ kubectl get nodes
NAME      STATUS   ROLES    AGE    VERSION
kmaster   Ready    master   112m   v1.13.4
mtech@kmaster:~$
mtech@kmaster:~$
```

Fig C. kubernetes installation

D. Kubernetes Components:

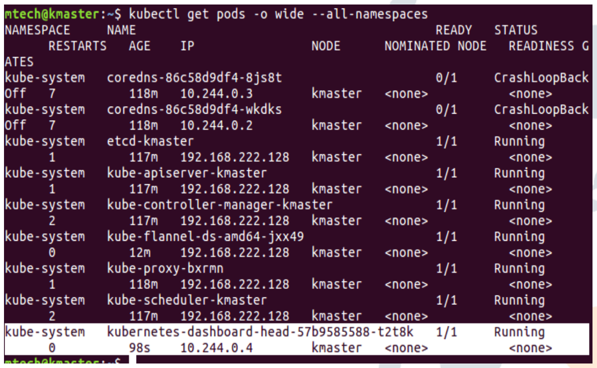
Below image shows that kubernetes components configured via command.



```
mtech@kmaster:~$ kubectl get pods -o wide --all-namespaces
NAMESPACE     NAME                                  READY   STATUS             RESTA
RTS   AGE    IP                NODE      NOMINATED NODE   READINESS GATES
kube-system   coredns-86c58d9df4-v2mzz              0/1     CrashLoopBackOff   1
      47h    192.168.0.2       kmaster   <none>             <none>
kube-system   coredns-86c58d9df4-w9zvd              0/1     CrashLoopBackOff   1
      47h    192.168.0.3       kmaster   <none>             <none>
kube-system   etcd-kmaster                          1/1     Running            5
      47h    192.168.222.128   kmaster   <none>             <none>
kube-system   kube-apiserver-kmaster                1/1     Running            16
      47h    192.168.222.128   kmaster   <none>             <none>
kube-system   kube-controller-manager-kmaster       1/1     Running            12
      47h    192.168.222.128   kmaster   <none>             <none>
kube-system   kube-flannel-ds-amd64-7lbfv           1/1     Running            0
      2m4s   192.168.222.128   kmaster   <none>             <none>
kube-system   kube-proxy-cb95g                      1/1     Running            5
      47h    192.168.222.128   kmaster   <none>             <none>
kube-system   kube-scheduler-kmaster                1/1     Running            11
      47h    192.168.222.128   kmaster   <none>             <none>
mtech@kmaster:~$
```

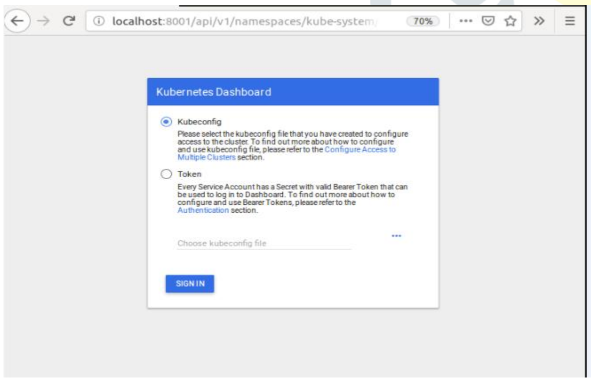
Fig D. Kubernetes Components

E. Dashboard Creation:



```
mtech@kmaster:~$ kubectl get pods -o wide --all-namespaces
NAMESPACE     NAME                                          READY   STATUS
      RESTARTS   AGE    IP                NODE      NOMINATED NODE   READINESS G
ATES
kube-system   coredns-86c58d9df4-8js8t                      0/1     CrashLoopBack
Off   7          118m   10.244.0.3        kmaster   <none>             <none>
kube-system   coredns-86c58d9df4-wkdks                      0/1     CrashLoopBack
Off   7          118m   10.244.0.2        kmaster   <none>             <none>
kube-system   etcd-kmaster                                  1/1     Running
      1          117m   192.168.222.128   kmaster   <none>             <none>
kube-system   kube-apiserver-kmaster                        1/1     Running
      1          117m   192.168.222.128   kmaster   <none>             <none>
kube-system   kube-controller-manager-kmaster               1/1     Running
      2          117m   192.168.222.128   kmaster   <none>             <none>
kube-system   kube-flannel-ds-amd64-jxx49                   1/1     Running
      0          12m    192.168.222.128   kmaster   <none>             <none>
kube-system   kube-proxy-bxrmn                              1/1     Running
      1          118m   192.168.222.128   kmaster   <none>             <none>
kube-system   kube-scheduler-kmaster                        1/1     Running
      2          117m   192.168.222.128   kmaster   <none>             <none>
kube-system   kubernetes-dashboard-head-57b9585588-t2t8k    1/1     Running
      0          98s    10.244.0.4        kmaster   <none>             <none>
```

After Creating dashboard it shown as below.



localhost:8001/api/v1/namespaces/kube-system

Kubernetes Dashboard

(●) Kubeconfig

Please select the kubeconfig file that you have created to configure access to the cluster. To find out more about how to configure and use kubeconfig file, please refer to the Configure Access to Multiple Clusters section.

( ) Token

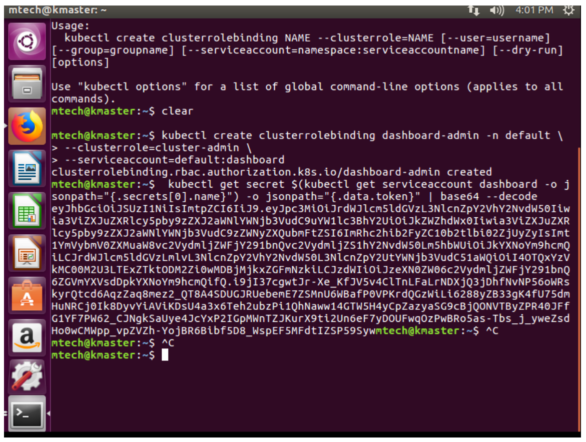
Every Service Account has a Secret with valid Bearer Token that can be used to log in to Dashboard. To find out more about how to configure and use Bearer Tokens, please refer to the Authentication section.

Choose kubeconfig file

Fig E. Dashboard Creation

F. Token Generation:

we can get access to dashboard by generating token, for that we used token generation command and execute it and used it to get access to dashboard.The token generation by command is shown below:

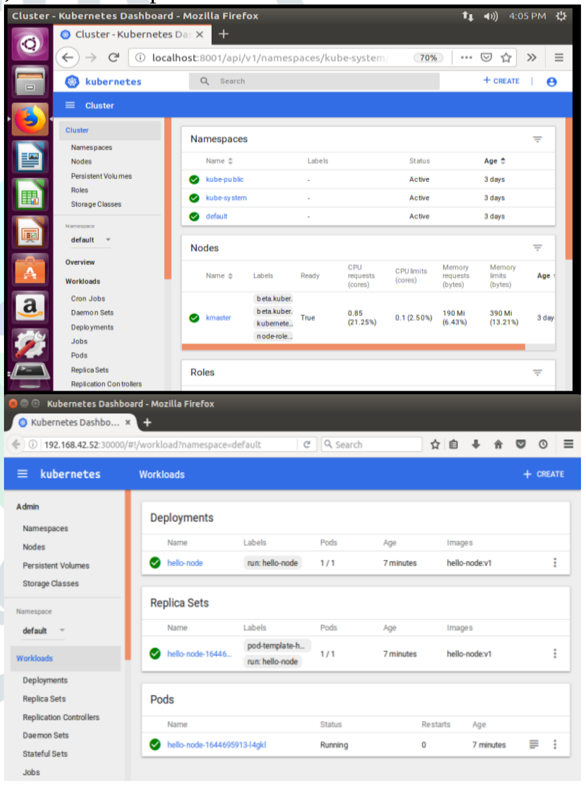


```
mtech@kmaster: ~
Usage:
  kubectl create clusterrolebinding NAME --clusterrole=NAME [--user=username]
[--group=groupname] [--serviceaccount=namespace:serviceaccountname] [--dry-run]
[options]

Use "kubectl options" for a list of global command-line options (applies to all
commands).
mtech@kmaster:~$ clear

mtech@kmaster:~$ kubectl create clusterrolebinding dashboard-admin -n default \
> --clusterrole=cluster-admin \
> --serviceaccount=default:dashboard
clusterrolebinding.rbac.authorization.k8s.io/dashboard-admin created
mtech@kmaster:~$  kubectl get secret $(kubectl get serviceaccount dashboard -o j
sonpath="{.secrets[0].name}") -o jsonpath="{.data.token}" | base64 --decode
eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiw
ia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJkZWZhdWx0Iiwia3ViZXJuZXR
lcy5pby9zZXJ2aWNlYWNjb3VudC9zZWNyZXQubmFtZSI6ImRhc2hib2FyZC10b2tlbi02ZjUyZyIsImt
1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW50Lm5hbWUiOiJkYXNoYm9hcmQ
iLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC51aWQiOiI4OTQxYzV
kMC00M2U3LTExZTktODM2Zi0wMDBjMjkxZGFmNzkiLCJzdWIiOiJzeXN0ZW06c2VydmljZWFjY291bnQ
6ZGVmYXVsdDpkYXNoYm9hcmQifQ.i9jI37cgwtJr-Xe_KfJV5v4ClTnLFaLrNDXjQ3jDhfNvNP56oWRs
kyrQtcd6AqzZaq8mez2_QT8A4SDUGJRUebemE7ZSMnU6WBafP0VPKrdQGzWiLi6288yZB33gK4fU75dm
HuNRCj0Ik8DyvYiAViKDsU4a3x6Teh2ubzPi1QhNaww14GTW5H4yCpZazyaSG9cBjQONVTByZPR40JFf
G1YF7PW62_CJNgkSaUye4JcYxP2IGpMWnTZJKurX9ti2Un6eF7yDOUFwqOzPwBRo5as-Tbs_j_yweZsd
Ho0wCMWpp_vpZVZh-YojBR6Bibf5D8_WspEF5MFdtIZSP59Sywmtech@kmaster:~$ ^C
mtech@kmaster:~$ ^C
mtech@kmaster:~$
```

Fig F. Token Generation

G. Kubernetes Cluster:

After getting access to dashboard we have create cluster by joining nodes to master.so we have to deploy container based ML services using kubernetes. The cluster of kubernetes is shown below and there deployements ,workloads replica sets are also shown below.



Cluster - Kubernetes Dashboard - Mozilla Firefox

localhost:8001/api/v1/namespaces/kube-system/

Namespaces

Name	Labels	Status	Age
✓ kube-public	-	Active	3 days
✓ kube-system	-	Active	3 days
✓ default	-	Active	3 days

Nodes

Name	Labels	Ready	CPU requests (cores)	CPU limits (cores)	Memory requests (bytes)	Memory limits (bytes)	Age
✓ kmaster	beta.kuber. beta.kuber. kubernete... node-role...	True	0.85 (21.25%)	0.1 (2.50%)	190 Mi (6.43%)	390 Mi (13.21%)	3 day

Roles

Kubernetes Dashboard - Mozilla Firefox

192.168.42.52:30000/#!/workload?namespace=default

Deployments

Name	Labels	Pods	Age	Images
✓ hello-node	run: hello-node	1 / 1	7 minutes	hello-node:v1

Replica Sets

Name	Labels	Pods	Age	Images
✓ hello-node-16446...	pod-template-h... run: hello-node	1 / 1	7 minutes	hello-node:v1

Pods

Name	Status	Restarts	Age
✓ hello-node-1644695913-l4gkl	Running	0	7 minutes

Fig G. Kubernetes Cluster

Once an applications is deployed ,we can see the how in the given image the object is detected and the outcome or results shows clearly the two given images in which we see the objects are detected.we can see those objects are also detected that we can see clearly by our eyes. So you can imagine how much powerful the machine learning algorithms and how efficient it is. By using kubernetes you can see cpu usage, memory usage of deployed applicaton on kubernetes dashboard

H. Machine Learning Services:

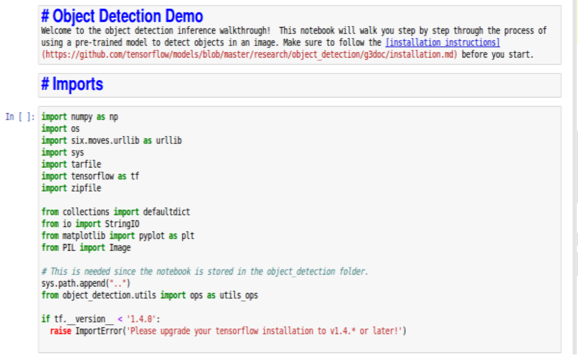
Here, the machine learning services are nothing but the applications that we builded and deployed. there are many kind of services we deployed using docker and kubernetes, we just need to packaged the source code/scripts inside container and deployed it using one the deployement technology. But it is not that easy job, we also require lots of libraries , configuration files and dependancies. One most important things we need to install is machine learning packages, if we want to build and deploy the machine learning services or applications. Following are the list of packages that are necessary for any machine learning application and we also need to configure these packages for our application which is machine learning based. The packeges are:

• Tensorflow Object detection
• Keras
• Scipy
• Matplotlib
• Numpy
• Pandas
• Sklearn
• Open CV
• Scikit Image
• Pillow
• Jupyter Notebook

The machine learning based services which is being build is nothing but the object detection in image. It means, build an machine learning application which detects the object in given image and gives accurate measurement related to object or entity. It gives you ratio about object means it scales where is your object and at how big in size. So I'm currently working on doing some changes like i said above. So here's below is the code or snippet about my application

a. Imports:

Once the packages for application is installed and configured then its time to import them



Object Detection Demo
Welcome to the object detection inference walkthrough! This notebook will walk you step by step through the process of using a pre-trained model to detect objects in an image. Make sure to follow the [installation instructions](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/installation.md) before you start.

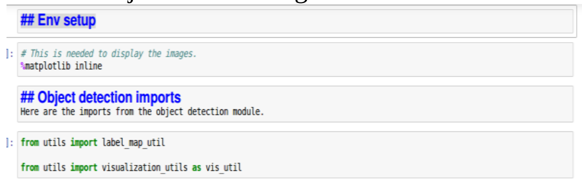
Imports

```
In [ ]: import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile

from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image

# This is needed since the notebook is stored in the object_detection folder.
sys.path.append("..")
from object_detection.utils import ops as utils_ops

if tf.__version__ < '1.4.0':
  raise ImportError('Please upgrade your tensorflow installation to v1.4.* or later!')
```

Fig a. Imports

b. Environment Setup:

Enviornmental setup is important part of application To display an image where the result located.So we used a library called matplotlib to display an images and plot the detected object on that images.



Env setup

```
]: # This is needed to display the images.
%matplotlib inline
```

Object detection imports
Here are the imports from the object detection module.

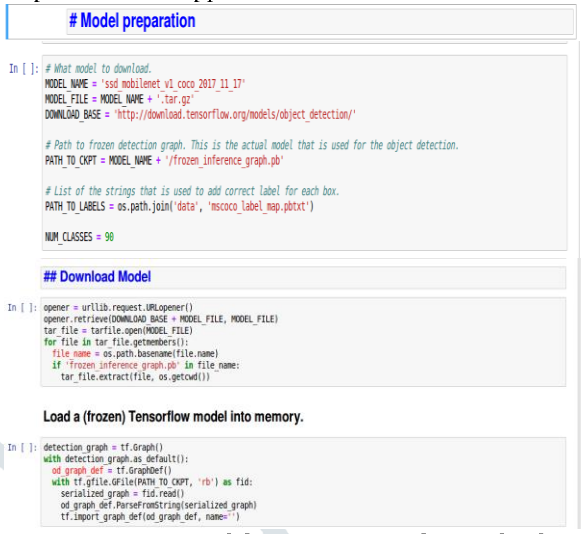
```
]: from utils import label_map_util

from utils import visualization_utils as vis_util
```

Fig b. Enviornment Setup

Enviroment setup is important step because once your foundation is strong then you Have much more opportunites to prepare and train your model effectively accurate.

c. Model Preparation and Download

Next step is to prepare the model which is being builded and deployed after that we download that model for future improvement in application



Model preparation

```
In [ ]: # What model to download.
MODEL_NAME = 'ssd_mobilenet_v1_coco_2017_11_17'
MODEL_FILE = MODEL_NAME + '.tar.gz'
DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/'

# Path to frozen detection graph. This is the actual model that is used for the object detection.
PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'

# List of the strings that is used to add correct label for each box.
PATH_TO_LABELS = os.path.join('data', 'mscoco_label_map.pbtxt')

NUM_CLASSES = 90
```

Download Model

```
In [ ]: opener = urllib.request.URLopener()
opener.retrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)
tar_file = tarfile.open(MODEL_FILE)
for file in tar_file.getmembers():
  file_name = os.path.basename(file.name)
  if 'frozen_inference_graph.pb' in file_name:
    tar_file.extract(file, os.getcwd())
```

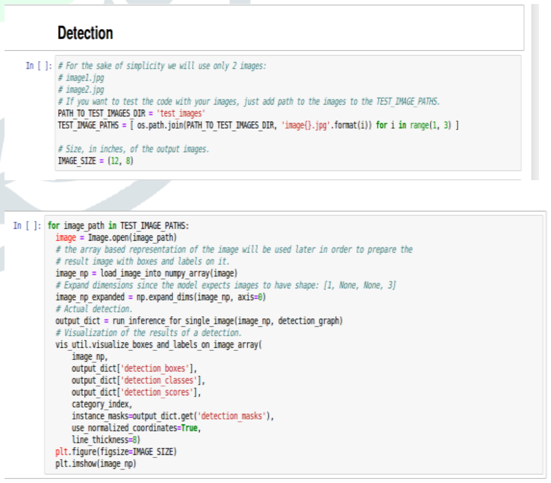
Load a (frozen) Tensorflow model into memory.

```
In [ ]: detection_graph = tf.Graph()
with detection_graph.as_default():
  od_graph_def = tf.GraphDef()
  with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
    serialized_graph = fid.read()
    od_graph_def.ParseFromString(serialized_graph)
    tf.import_graph_def(od_graph_def, name='')
```

Fig c. Model Preparation and Download

d. Detection:

Finally, the last part come i.e to detect object in image that's our aim of application or machine learning based service. We have using two images for simplicity to test out application to check either it is working correctly or not. We added the path of images that we gonna used for testing purpose to check whether application is correctly able to detect an object on given images or not So that's our final step or final script which is the detection script to detects the object in a given images. It finds or detects the object in images that we specifically used two images that we seen in result of application.



Detection

```
In [ ]: # For the sake of simplicity we will use only 2 images:
# image1.jpg
# image2.jpg
# If you want to test the code with your images, just add path to the images to the TEST_IMAGE_PATHS.
PATH_TO_TEST_IMAGES_DIR = 'test_images'
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR, 'image{}.jpg'.format(i)) for i in range(1, 3) ]

# Size, in inches, of the output images.
IMAGE_SIZE = (12, 8)
```

```
In [ ]: for image_path in TEST_IMAGE_PATHS:
  image = Image.open(image_path)
  # the array based representation of the image will be used later in order to prepare the
  # result image with boxes and labels on it.
  image_np = load_image_into_numpy_array(image)
  # Expand dimensions since the model expects images to have shape: [1, None, None, 3]
  image_np_expanded = np.expand_dims(image_np, axis=0)
  # Actual detection.
  output_dict = run_inference_for_single_image(image_np, detection_graph)
  # Visualization of the results of a detection.
  vis_util.visualize_boxes_and_labels_on_image_array(
      image_np,
      output_dict['detection_boxes'],
      output_dict['detection_classes'],
      output_dict['detection_scores'],
      category_index,
      instance_masks=output_dict.get('detection_masks'),
      use_normalized_coordinates=True,
      line_thickness=8)
  plt.figure(figsize=IMAGE_SIZE)
  plt.imshow(image_np)
```

Fig d. Detection

e. Results

As you can see below the outcome of our application that detect the object in a given image.. The results generated by the application is shown below in two images. First result shows the detection of being and birds. The second images shows the detection of dog.

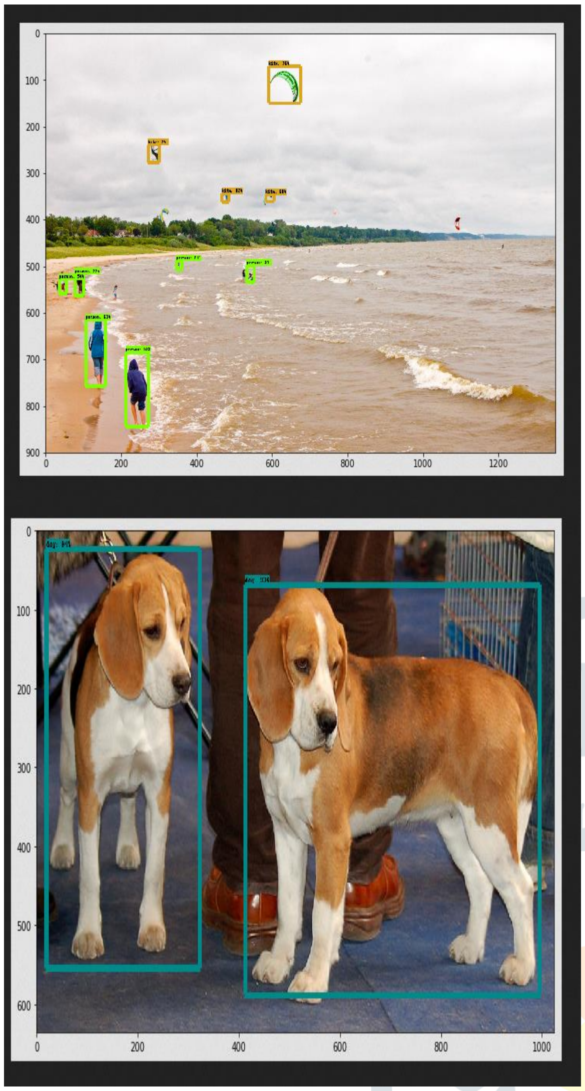


Fig e. Results

VI. CONCLSUTION & FUTURE SCOPE STUDY

This project concludes the developing a machine learning application is common and might be easy and we call such applications as a user-specific application but bringing a machine learning application by converting its whole scenario in cloud enviornment is something different. so in our project, we achieve this feet by using new cloud technologies. The purpose of this project is to develop an application using emerging trending technologies. the tools and technologies which are used in this project are rarely know to people because this are the upcoming technologies that i have used in project. so people are not much aware about this technologies and i build the project in such technologies might be great and nice experience for me. The future step might be doing the performance comparison of deployed services using Prometheus & ISTIO and also improve their performance as its our primary concern..we trying to the monitor performance of containers based ml services by using the prometheus which is a open-source monitoring and altering toolkit.

REFERENCES

[1] Ernst &Young."Virtual machine placement method for energy saving in cloud computing." Information Technology and Electrical Engineering (ICITEE), 2015 7th International Conference on. IEEE, 2015.

[2] Su, Nan, et al. "Research on virtual machine placement in the cloud based on improved simulated annealing algorithm." World Automation Congress (WAC), 2016. IEEE, 2016.

[3] Karmakar, Kamalesh, SunirmalKhatua, and Rajib K. Das. "Efficient virtual machine placement in cloud environment." Advances in Computing, Communications and Informatics (ICACCI), 2017 International Conference on. IEEE, 2017.

[4] Yang, Song, et al. "Reliable virtual machine placement and routing in clouds." arXiv preprint arXiv:1701.06005 (2017).

[5] Salah, Tasneem, et al. "Performance comparison between container-based and VM-based services." Innovations in Clouds, Internet and Networks (ICIN), 2017 20th Conference on. IEEE, 2017.

[6] Rista, Cassiano, et al. "Improving the Network Performance of a Container-Based Cloud Environment for Distributed Systems." High Performance Computing & Simulation (HPCS), 2017 International Conference on. IEEE, 2017.

[7] Xu, Pengfei, Shaohuai Shi, and Xiaowen Chu. "Performance Evaluation of Deep Learning Tools in Docker Containers." Big Data Computing and Communications (BIGCOM), 2017 3rd International Conference on. IEEE, 2017.

[8] Zeng, Hao, et al. "Measurement and Evaluation for Docker Container Networking." Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2017 International Conference on. IEEE, 2017

[9] Mao, Ying, et al. "Draps: Dynamic and resource-aware placement scheme for docker containers in a heterogeneous cluster." Performance Computing and Communications Conference (IPCCC), 2017 IEEE 36th International. IEEE, 2017

[10] Model, Víctor, et al. "Modelling performance & resource management in kubernetes." 2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC). IEEE, 2016.

[11] , Víctor, et al. "Adaptive application scheduling under interference in kubernetes." Utility and Cloud Computing (UCC), 2016 IEEE/ACM 9th International Conference on. IEEE, 2016

[12] Tsai, Pei-Hsuan, et al. "Distributed analytics in fog computing platforms using Tensorflow and Kubernetes." Network Operations and Management Symposium (APNOMS), 2017 19th Asia-Pacific. IEEE, 2017.

[13] Chang, Chia-Chen, et al. "A Kubernetes-Based Monitoring Platform for Dynamic Cloud Resource Provisioning." GLOBECOM 2017-2017 IEEE Global Communications Conference. IEEE, 2017. [14] Xie, Xiao-Lan, Peng Wang, and Qi Wang. "The performance analysis of Docker and rkt based on Kubernetes." 2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD). IEEE, 2017.

[15] Rovnyagin, Mikhail M., et al. "Using the ML-based architecture for adaptive containerized system creation." Young Researchers in Electrical and Electronic Engineering (EIConRus), 2018 IEEE Conference of Russian. IEEE, 2018.