TensorFlow Serving

|  |  |
| --- | --- |
| 制作小组 | BIG DATA GROUP |
| 制作日期 | 2018.11.02 |

Contents

[TensorFlow Serving 3](#_Toc529201579)

[1.About TensorFlow Serving 3](#_Toc529201580)

[2.TensorFlow Serving Install 6](#_Toc529201581)

[2.1.Install Docker: 6](#_Toc529201582)

[2.2.1.Get Docker 6](#_Toc529201583)

[2.2.2.Install docker 6](#_Toc529201584)

[2.2.3.Restart computer 6](#_Toc529201585)

[2.2.4. Change Daemon: 6](#_Toc529201586)

[2.2.5.Check if installed alright 7](#_Toc529201587)

[2.2.Install TensorFlow serving. 7](#_Toc529201588)

[2.2.1.Pulling a serving image 7](#_Toc529201589)

[2.2.2.Clone build script 8](#_Toc529201590)

[2.2.3.Train example model 8](#_Toc529201591)

[2.2.4.Serving models using docker 8](#_Toc529201592)

[2.2.5.Querying the model 9](#_Toc529201593)

[2.2.6.Models file for Docker 9](#_Toc529201594)

[2.2.7.TensorFlow serving version 9](#_Toc529201595)

[3.Kubernetes for distributed TensorFlow Serving 10](#_Toc529201596)

[3.1.About Kubernetes 10](#_Toc529201597)

[3.1.1.Overview 10](#_Toc529201598)

[3.1.2.Basic 10](#_Toc529201599)

[3.2.Deploy model in Kubernetes 11](#_Toc529201600)

[3.2.1.Create a container cluster 11](#_Toc529201601)

[3.2.2.Upload docker image 11](#_Toc529201602)

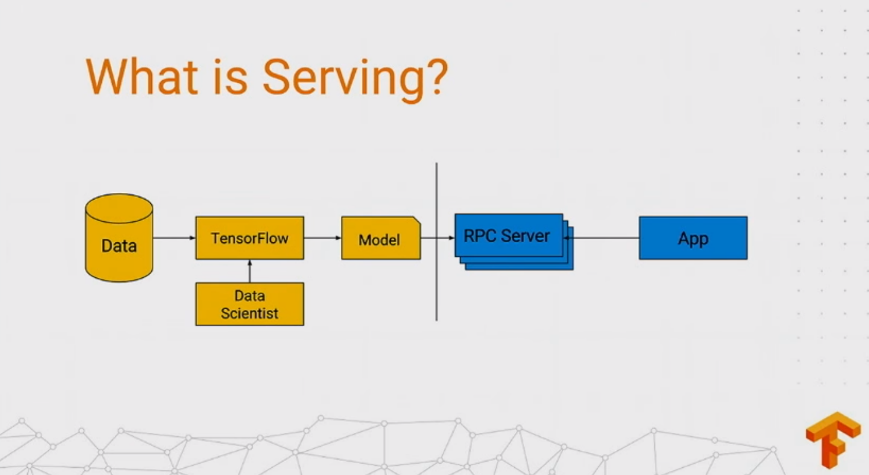
[3.2.3.Create Deployment and service 11](#_Toc529201603)

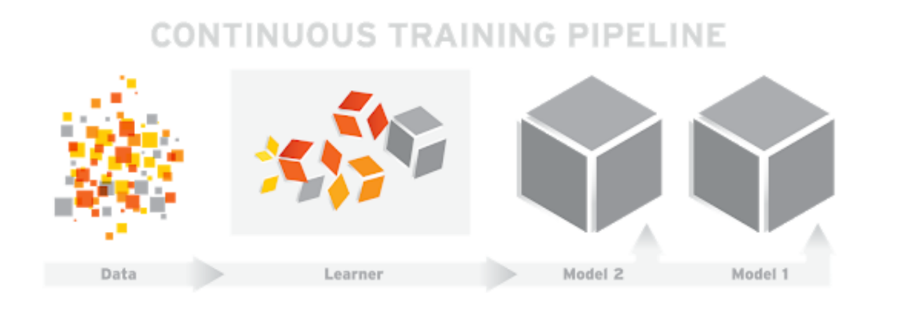
[3.2.4.Query the model 12](#_Toc529201604)

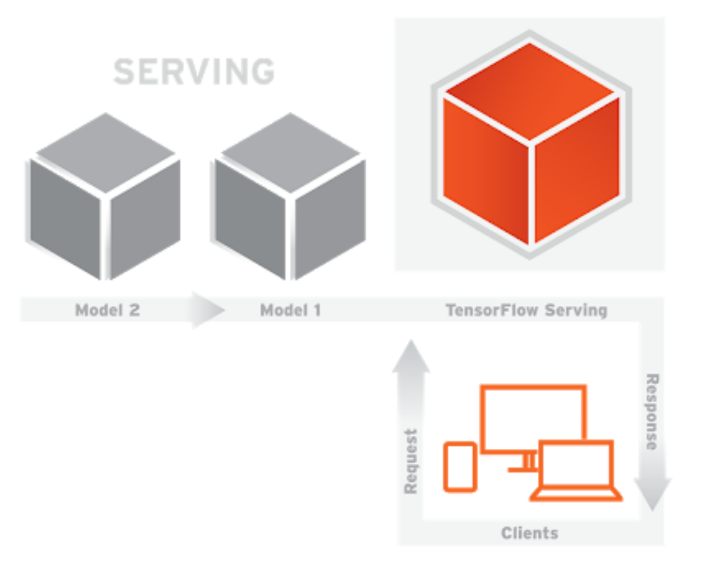
# TensorFlow Serving

# 1.About TensorFlow Serving

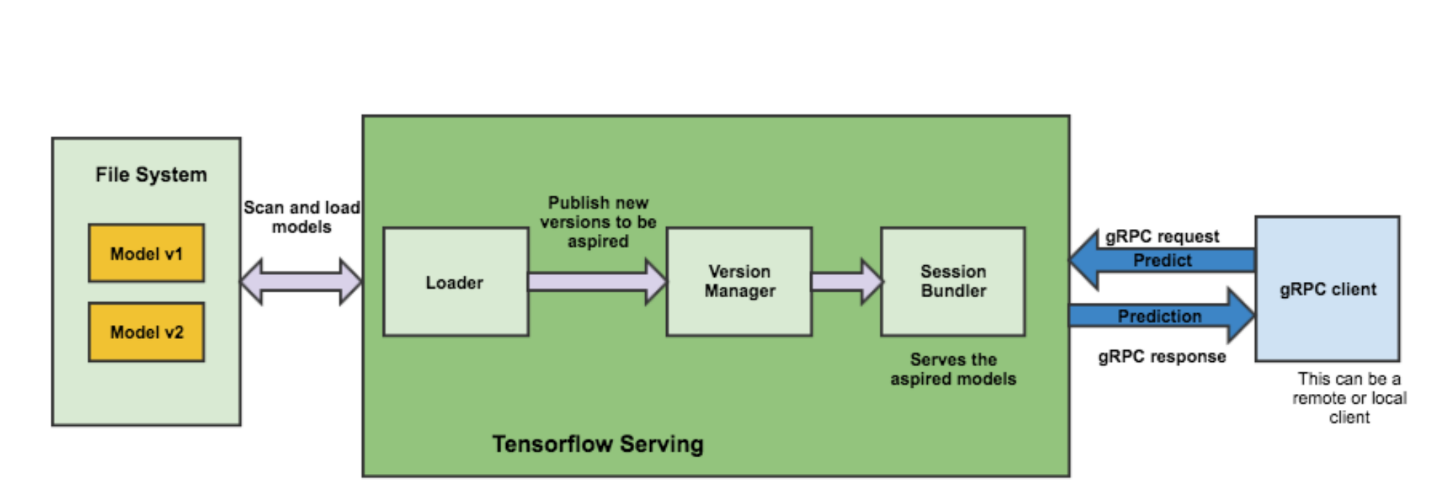
TensorFlow Serving is a **flexible, high-performance** **serving system** for machine learning models, designed for **production environments**. TensorFlow Serving makes it easy to deploy new algorithms and experiments, while keeping the same server architecture and APIs. TensorFlow Serving provides out-of-the-box integration with TensorFlow models, but can be easily extended to serve other types of models and data.



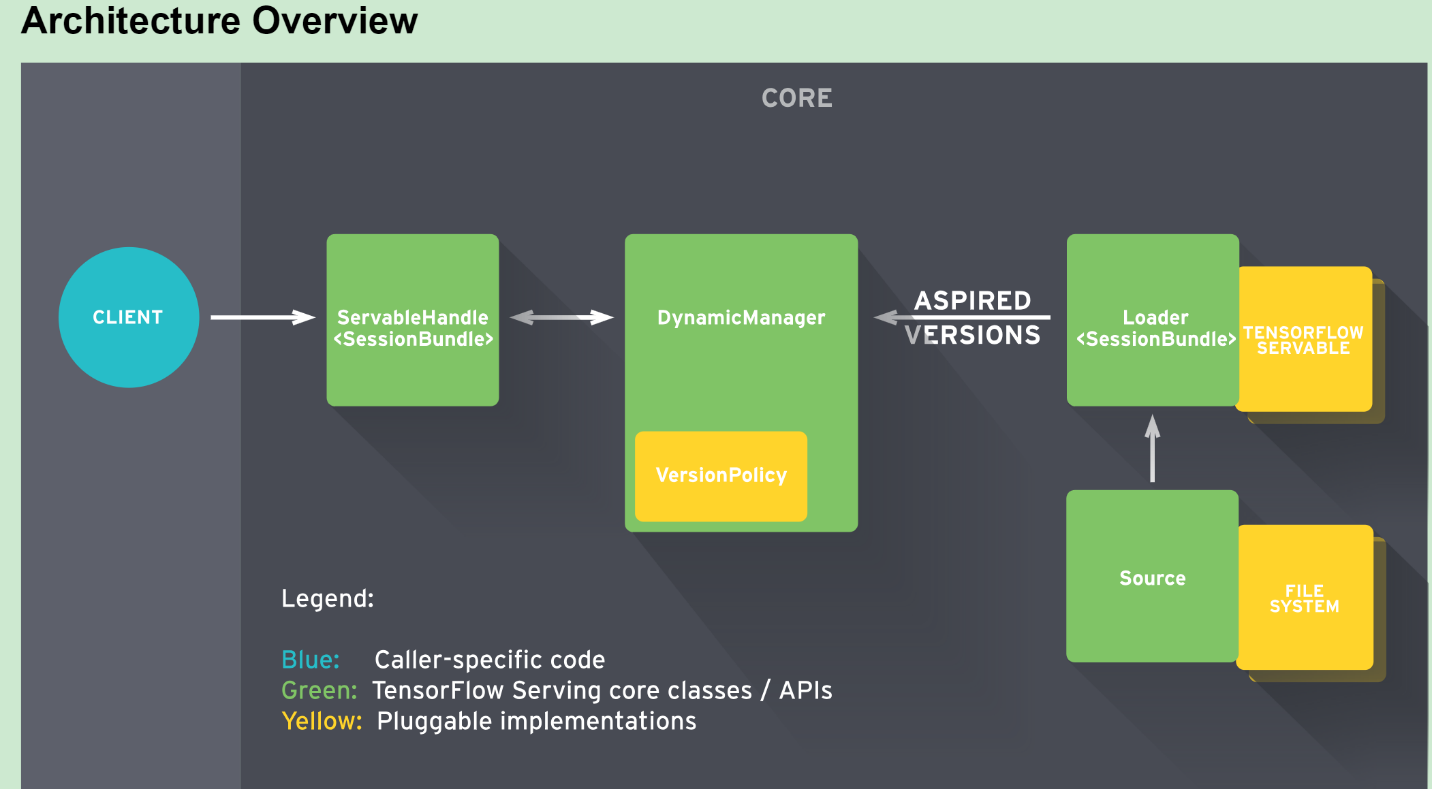


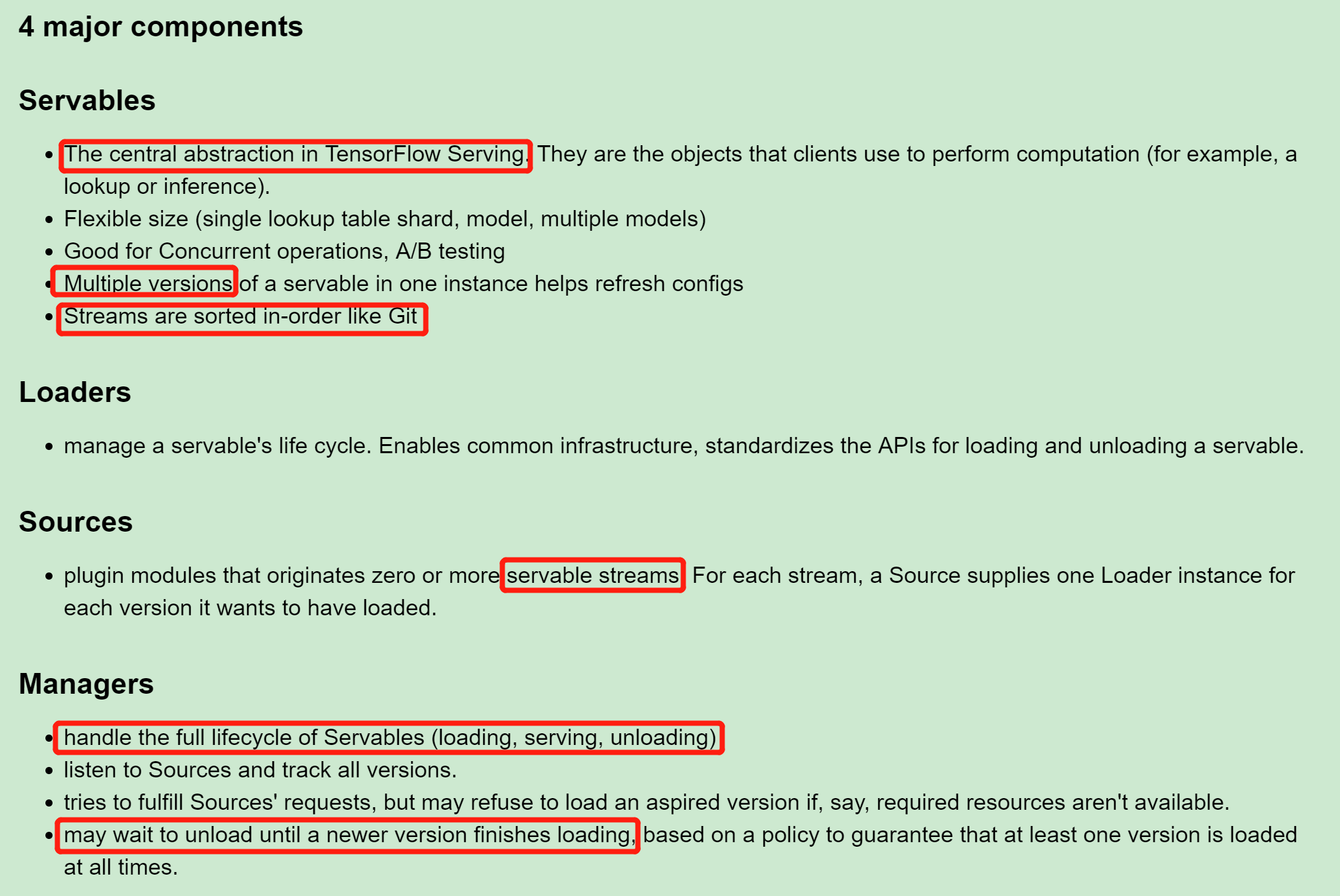


Procedure of TensorFlow Serving:



TensorFlow serving architecture:





TensorFlow Serving is written in C++ and it supports **Linux**. TensorFlow Serving introduces minimal overhead. In our benchmarks we recoded ~**100,000 queries per second (QPS)** **per core** on a **16 vCPU** Intel Xeon E5 2.6 GHz machine, excluding gRPC and the TensorFlow inference processing time.

# 2.TensorFlow Serving Install

## 2.1.Install Docker:

### 2.2.1.Get Docker

Download docker for windows form Docker official website: <https://store.docker.com/editions/community/docker-ce-desktop-windows>

Noted(Have to sign in for docker, get **community** **Edition**)

### 2.2.2.Install docker

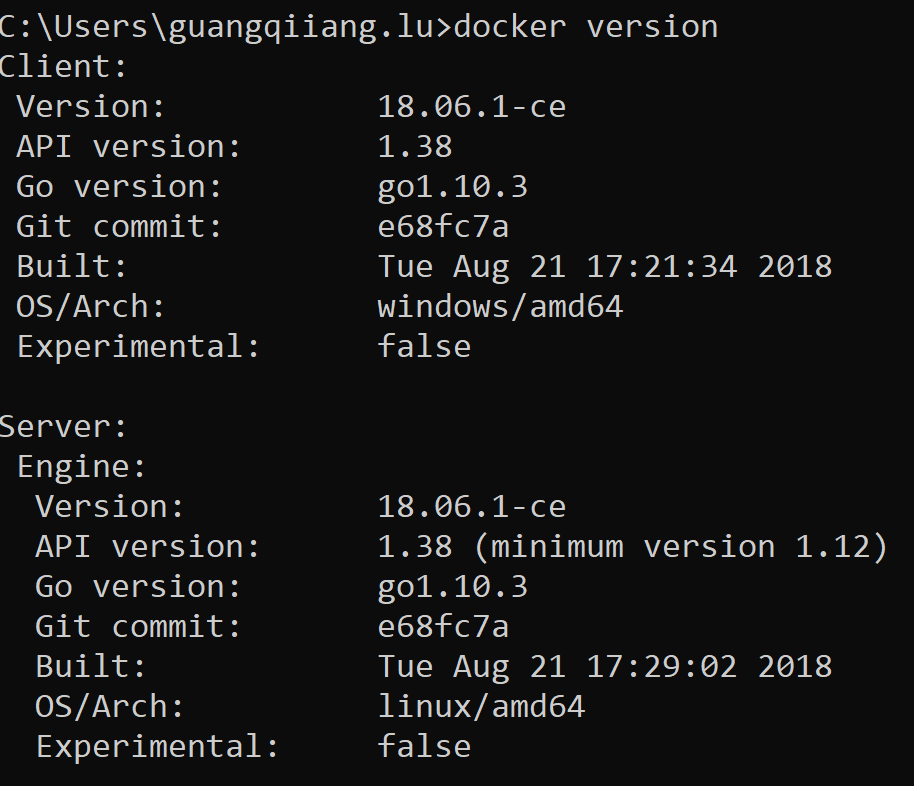
Then just install the ….exe. After install is finished, have to **log out** for windows user.

### 2.2.3.Restart computer

After **restart** the windows, double click Docker icon and you have to login in: user\_name and password. If you have seen docker is **running**, then open a terminal, in terminal:

**docker version**

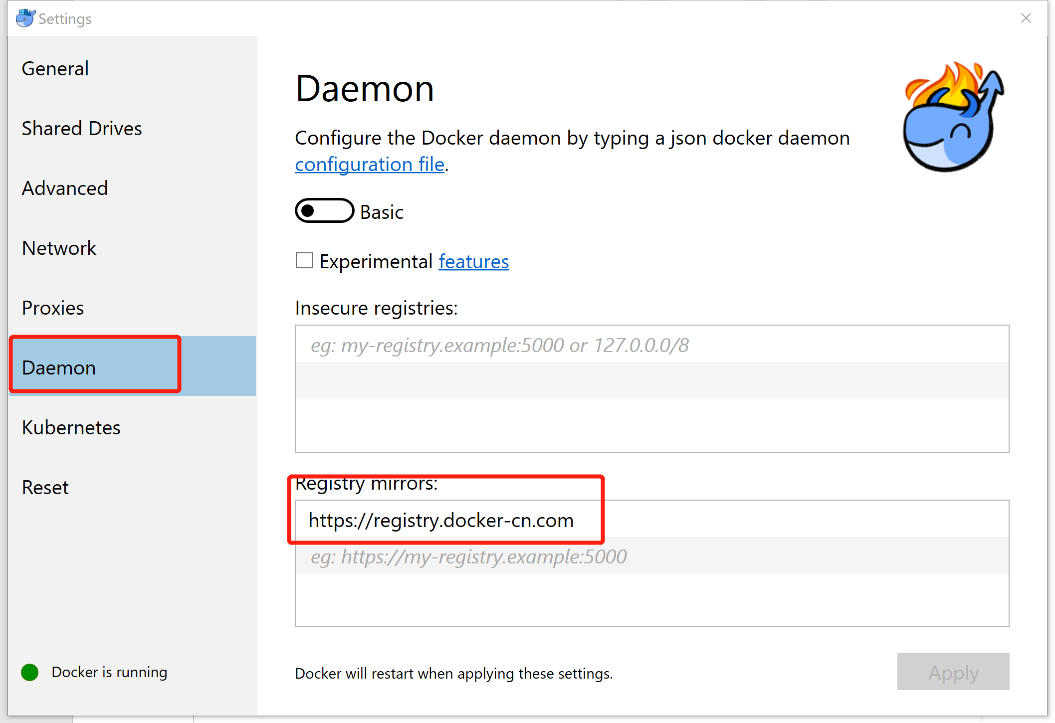
As you can see:



### 2.2.4. Change Daemon:

For running docker icon right bellow, right click the running docker icon, click **Settings**

Change Daemon with this: [https://registry.docker-cn.com](https://registry.docker-cn.com/) ,apply, then docker will restart.

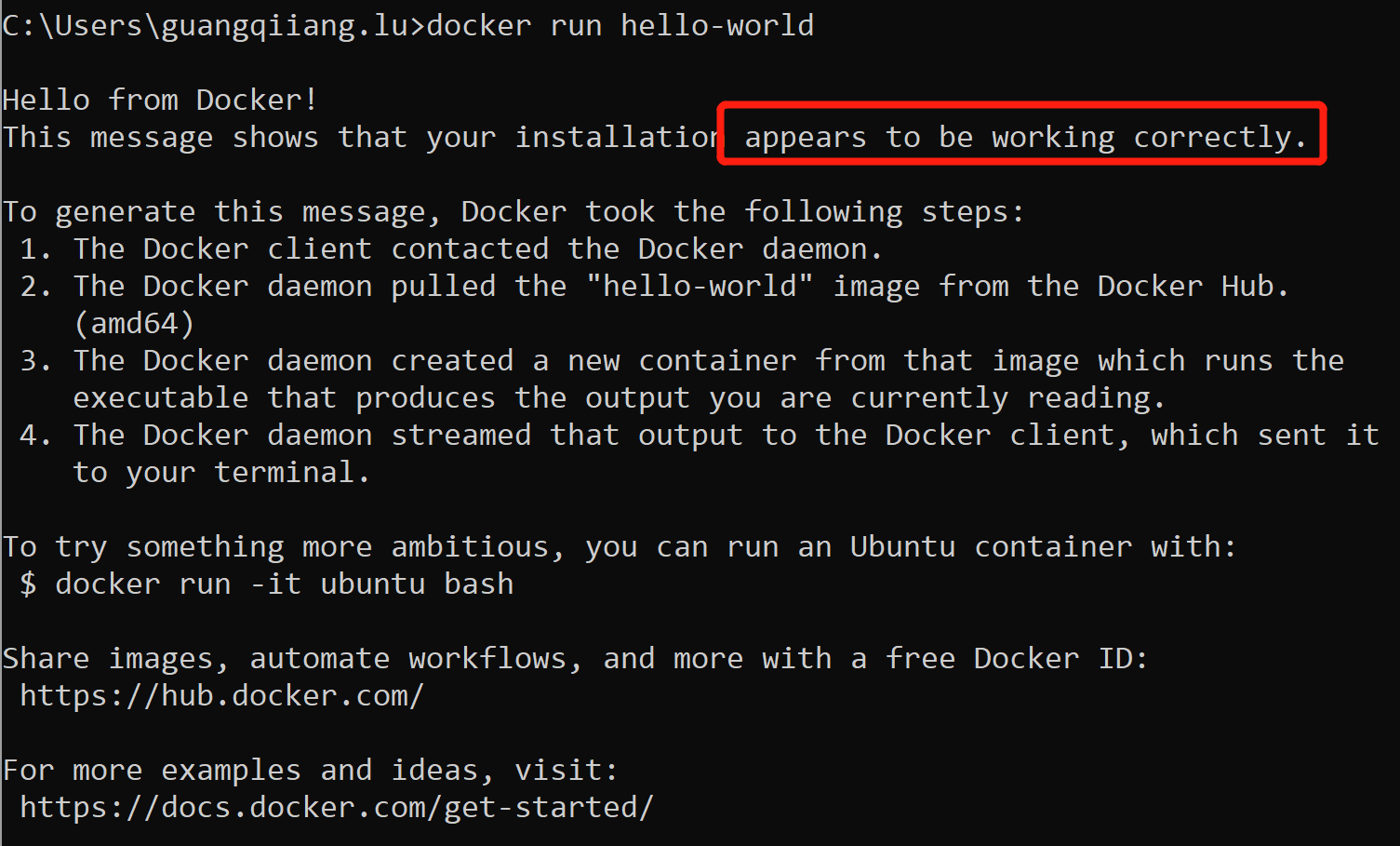


### 2.2.5.Check if installed alright

Open a terminal, code this:

**docker run hello-world**

Result as bellow:

****

## 2.2.Install TensorFlow serving.

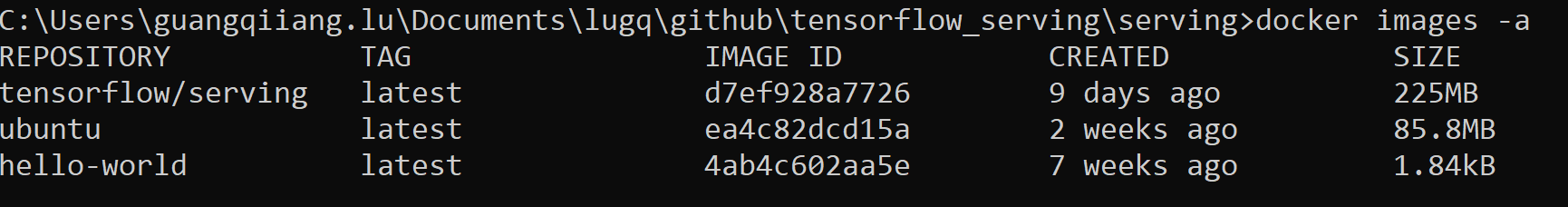
### 2.2.1.Pulling a serving image

Pull the latest TensorFlow serving image using this code:

**docker pull tensorflow/serving**

Then use this commend to find that we have finished installing this image:

**docker images -a**

****

### 2.2.2.Clone build script

Make a directory called: **tensorflow\_serving** in **path**(C:\Users\guangqiiang.lu\Documents\lugq\github), then use git clone the repository:

**git clone** [**https://github.com/tensorflow/serving.git**](https://github.com/tensorflow/serving.git)

**cd serving**

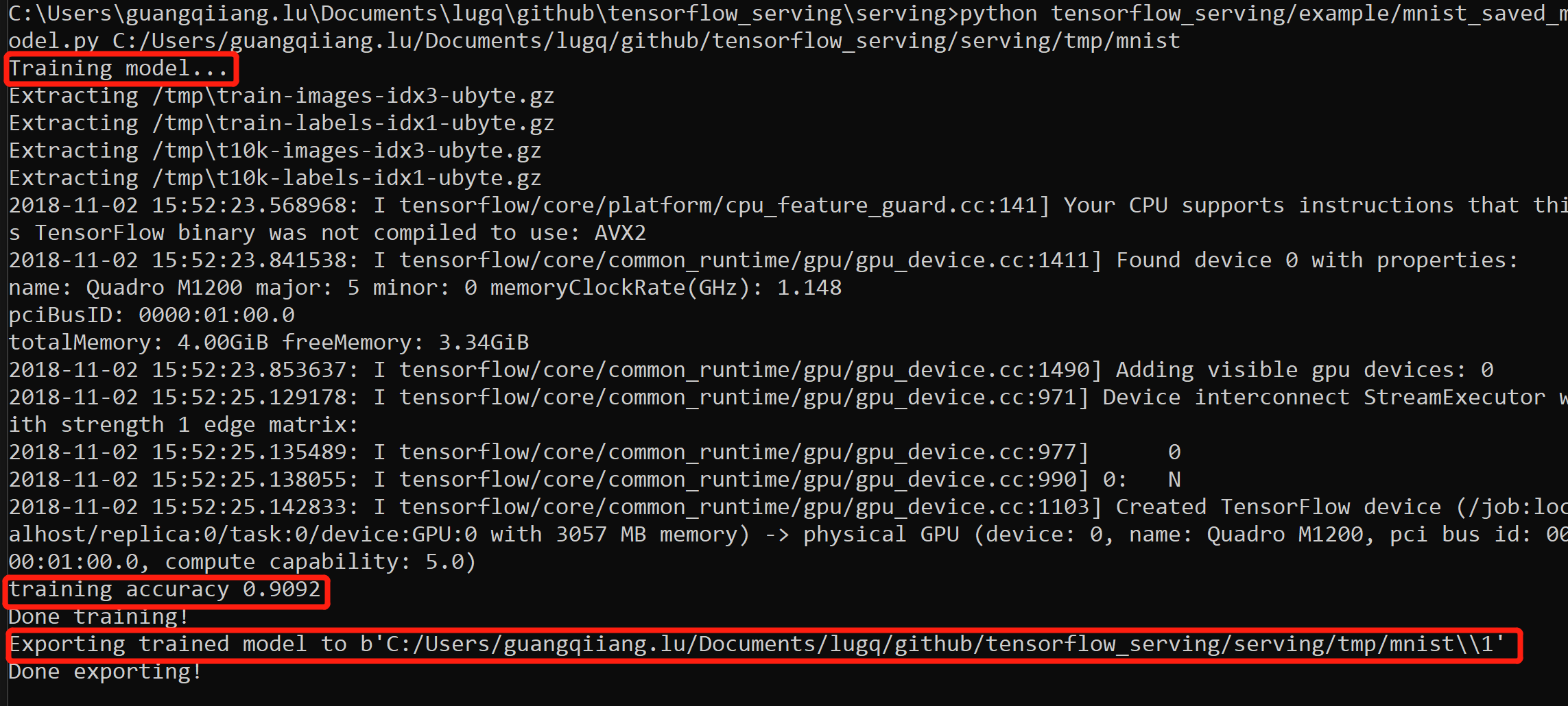
Noted(If git is not installed, maybe install the git first, just download and install)

### 2.2.3.Train example model

At the same terminal, make a directory called: **tmp/mnist,** then run this code in terminal:

**python tensorflow\_serving/example/mnist\_saved\_model.py C:/Users/guangqiiang.lu/Documents/lugq/github/tensorflow\_serving/serving/tmp/mnist**

Result:

****

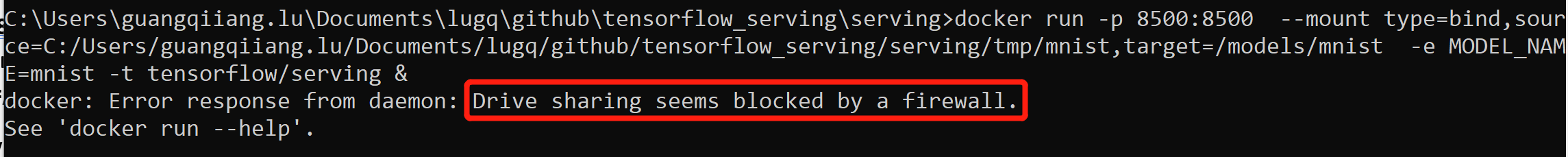
### 2.2.4.Serving models using docker

Now that we have already trained models and save it to disk, then we can serve it using Docker, code this in terminal:

**docker run -p 8500:8500 --mount type=bind,source=C:/Users/guangqiiang.lu/Documents/lugq/github/tensorflow\_serving/serving/tmp/mnist,target=/models/mnist -e MODEL\_NAME=mnist -t tensorflow/serving &**

Then Docker will ask you whether or not to share document, **yes.** After that, you will need to sign the **password** of computer.

But for now, because of firewall, we can’t serve the example model.



### 2.2.5.Querying the model

Because of windows firewall problem, I can’t get prediction for querying the model. Here is official website for querying the model example and result:

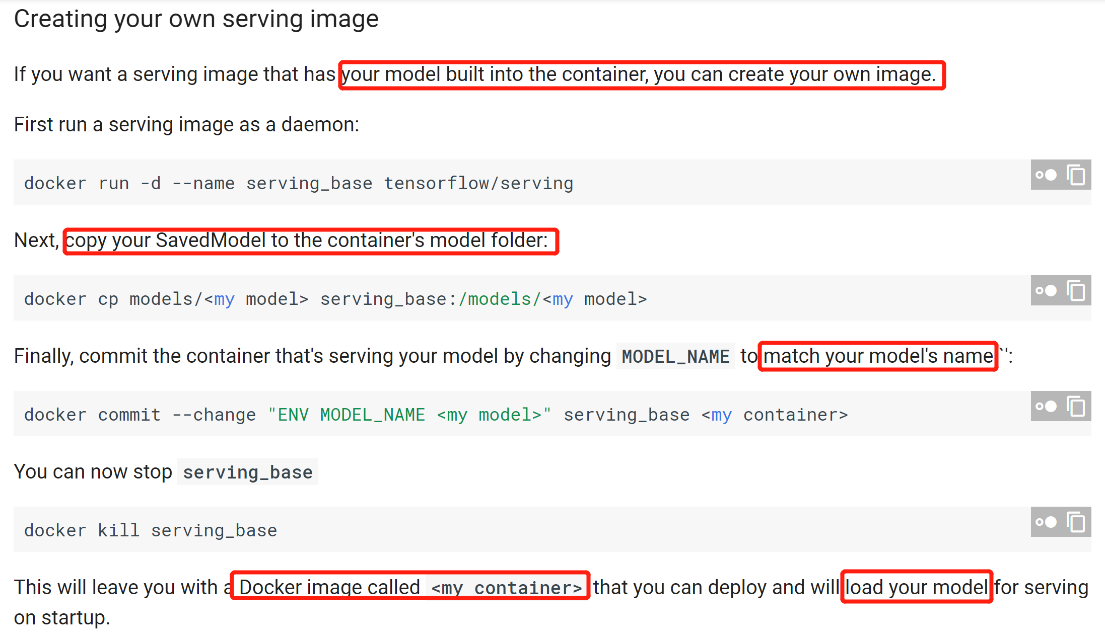
**curl -d '{"instances": [1.0, 2.0, 5.0]}' -X POST** [**http://localhost:8501/v1/models/half\_plus\_two:predict**](http://localhost:8501/v1/models/half_plus_two:predict)

result:

**{ "predictions": [2.5, 3.0, 4.5] }**

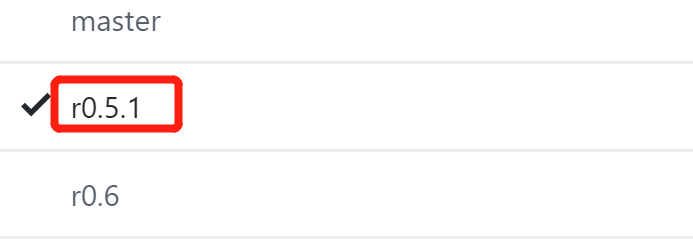
### 2.2.6.Models file for Docker

For using docker to serve models, already trained model file can be stored in **local for container of docker or just server client local,** as previous example. But how about make my **own** **images** that **contains model file**?



## 2.2.7.TensorFlow serving version

According to TensorFlow serving Github repository information, we can include that TensorFlow serving begins as **TensorFlow Version:0.5.1**



# 3.Kubernetes for distributed TensorFlow Serving

## 3.1.About Kubernetes

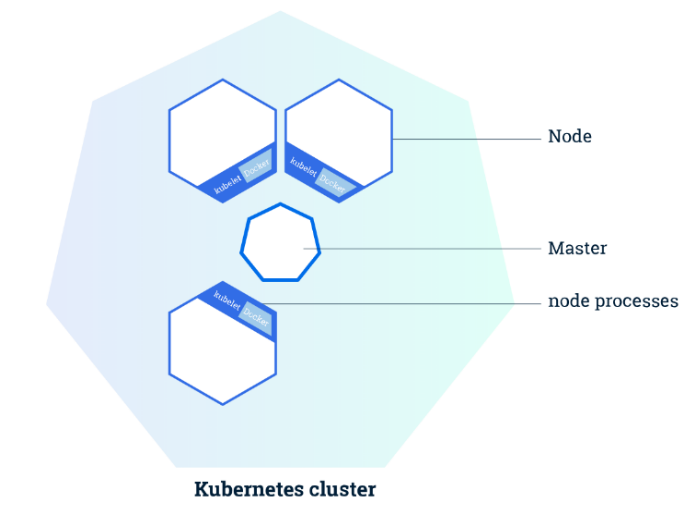
### 3.1.1.Overview

With modern web services, users expect applications to be available 24/7, and developers expect to deploy new versions of those applications several times a day. Containerization helps package software to serve these goals, enabling applications to be **released and updated** in an **easy and fast way** without downtime. Kubernetes helps you make sure those **containerized applications** run where and when you want, and helps them find the resources and tools they need to work. Kubernetes is a **production-ready, open source platform** designed with Google's accumulated experience in container orchestration, combined with best-of-breed ideas from the community.

### 3.1.2.Basic

**Kubernetes Clusters**

Kubernetes **coordinates** a highly **available cluster** of computers that are connected to work as a **single unit**. The abstractions in Kubernetes allow you to deploy containerized applications to a cluster **without** tying them **specifically** to **individual machines**. To make use of this new model of deployment, applications need to be packaged in a way that decouples them from individual hosts: they need to be containerized. **Containerized applications** are more **flexible and available** than in past deployment models, where applications were installed directly onto specific machines as packages deeply integrated into the host. Kubernetes **automates** the **distribution and scheduling of application containers** across a cluster in a **more efficient way**. Kubernetes is an open-source platform and is **production-ready.**



The **Master** is responsible for **managing** the cluster.

A **node** is a **VM** or a **physical computer** that serves as a **worker machine** in a Kubernetes cluster.  A Kubernetes cluster that handles production traffic should have a **minimum of three nodes**. **The nodes communicate with the master using the Kubernetes API.**

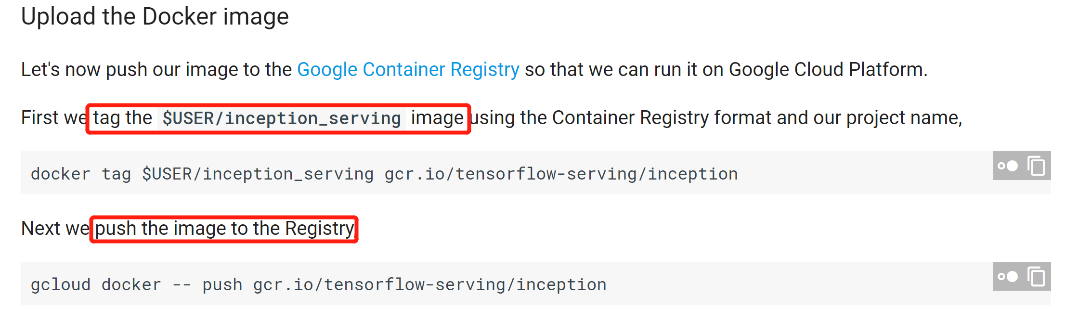
## 3.2.Deploy model in Kubernetes

### 3.2.1.Create a container cluster

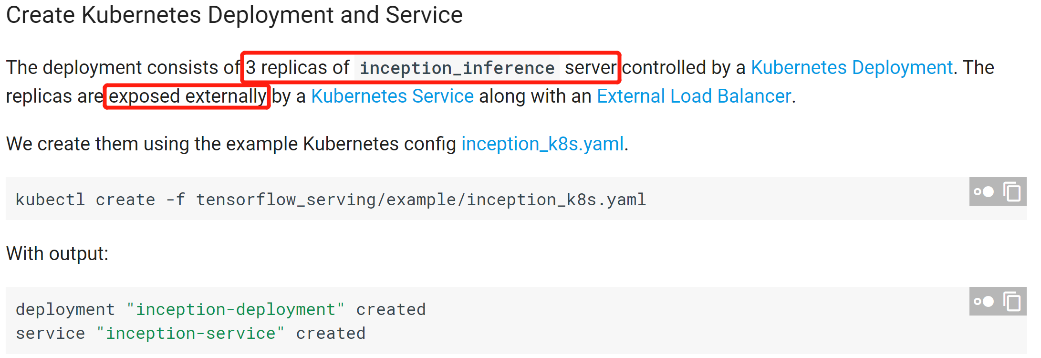
Here is using google cloud Kubernetes example:



### 3.2.2.Upload docker image



### 3.2.3.Create Deployment and service





### 3.2.4.Query the model

$ tools/bazel\_in\_docker.sh \

bazel-bin/tensorflow\_serving/example/inception\_client \

--server=104.155.184.157:8500 --image=local/path/to/my\_cat\_image.jpg

**Result** is a Json file:

outputs {

**key: "classes"**

value {

dtype: DT\_STRING

tensor\_shape {

dim {

size: 1

}

dim {

size: 5

}

}

**string\_val: "tiger cat"**

string\_val: "Egyptian cat"

string\_val: "tabby, tabby cat"

string\_val: "lynx, catamount"

string\_val: "Cardigan, Cardigan Welsh corgi"

}

}

outputs {

**key: "scores"**

value {

dtype: DT\_FLOAT

tensor\_shape {

dim {

size: 1

}

dim {

size: 5

}

}

**float\_val: 9.5486907959**

float\_val: 8.52025032043

float\_val: 8.05995368958

float\_val: 4.30645561218

float\_val: 3.93207240105

}

}