Kubernetes with vGPU and ML Framework (Tensorflow) Setup Guide

By:Han Wang

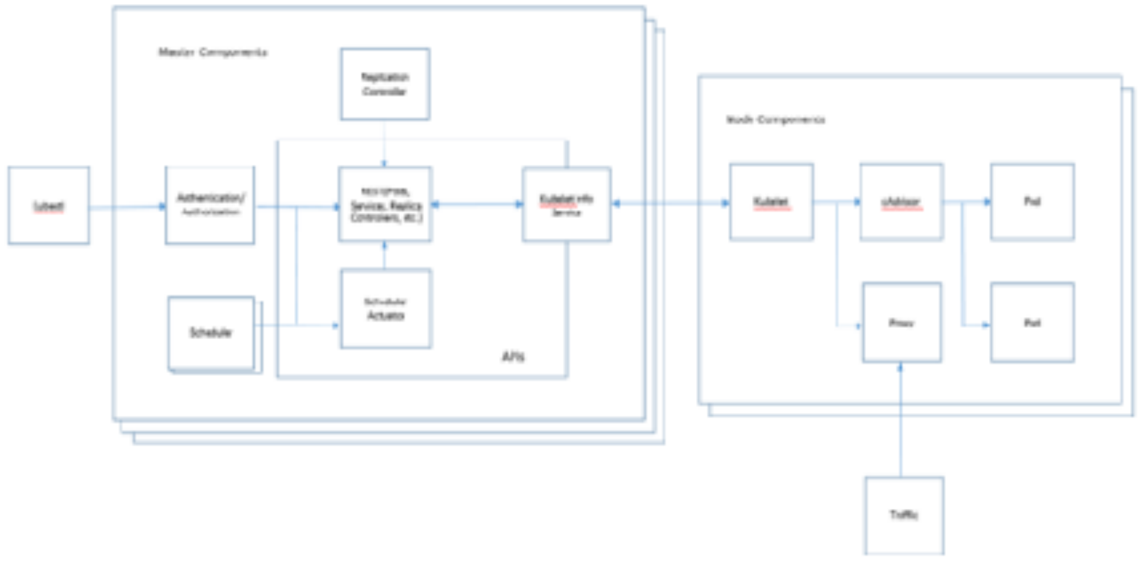
Cloud/Physical Layer:

- Cloud Provider: Google Cloud Platform
- 1 Jumphost, 3 Masters, 2 Nodes
- 4 GPUs, 2 GPUs on each node

Specifications:

Type	OS	CPU	GPU	Memory (GB)	Disk (GB)
Master	CentOS 7	2		8	100
Node	CentOS 7	8	2xNVIDIA Tesla P100	52	100
Jumphost	CentOS 7	2		8	100

Architecture Diagram:



Deploy Kubernetes Cluster

The entire building process is fully automated. On jumphost, under k8s-infrastructure directory, each playbook/role is written by ansible. The entire process is divided into three parts: pre-build, rebuild, and post-build. Pre-build contains steps that will install needed drivers, docker-ce etc. Rebuild process is actually a build process to have Kubernetes cluster deployed. Post-build process contains steps that install service accounts, GPU device plugins, machine learning framework-tensorflow etc. By running/re-running ansible playbooks, users can create/recreate the entire infrastructure platform in about 15 minutes.

Pre-requisites:

- Generate SSH key pairs and add SSH key to Google Cloud project under Metadata (project wide key) or specific instance.
- Provision masters, nodes and jumphost on Google Cloud with defined specification

Step by Step Deployment:

1. SSH to jumphost
    a. *ssh username@<jumphost-ip>*
2. Install tools

With a fresh jumphost, user will need to install needed utility tools for usage. There are a few required ones that will be used in later deployment process: python-pip, python-netaddr, git, and ansible (v2.4 or newer). Execute the following command lines to install these tools:

   a. *sudo yum install git python-pip python-netaddr git -y*
   b. *sudo pip install ansible -y*

3. Clone Kubespray repository

   Kubespray is a composition of Ansible playbooks, inventory, provisioning tools, and domain knowledge for generic OS/Kubernetes clusters configuration management tasks. We will leverage part of Kubespray to deploy Kubernetes cluster. However, we will need to create our own Ansible playbooks and inventory for tasks that Kubespray doesn't support. For example, installation of NVIDIA driver, CUDA driver, NVIDIA-Docker, NVIDIA GPU device plugin, specific Docker-CE, Docker runtime modification, customizing firewall rules, deploying Tensorflow, creating needed Service Accounts, deploying monitoring tools (Prometheus, Grafana, Alertmanager), etc.

   a. *git clone https://github.com/kubernetes-incubator/kubespray.git*

4. Create inventory and modify configuration files accordingly based on instances information, cluster setting, networking, default feature values like kube_feature_gates etc.

5. Pre-build

   Kubespray doesn't support GPU related tasks, so I wrote a few Ansible playbooks for pre-build installation so that we automate the process rather than manual deployment. At Jumphost: k8s-infrastructure/playbooks and roles, there are tasks to install NVIDIA and CUDA driver, specific Docker version (required by NVIDIA-Docker 2.0), NVIDIA-Docker 2.0, and set up NVIDIA runtime as docker default runtime rather than runc.

   Prerequisites for NVIDIA-Docker 2.0:

   a. GNU/Linux x86_64 with kernel version > 3.10
   b. Docker >= 1.12
   c. NVIDIA GPU with Architecture > Fermi (2.1)
   d. NVIDIA driver ~= 361.93

   The pre_build.sh contains a list of ansible-play with pre-build steps need to be performed:

   a. Execute NVIDIA playbook to install NVIDIA driver, CUDA drivers.
   b. Execute docker playbook to install the required docker version
   c. Execute NVIDIA-docker playbook to install NVIDIA=docker 2.0
   d. Execute docker-runtime playbook to set default runtime as nvidia in docker daemon.json

   Execute by *./pre_build.sh*

6. Build/Rebuild

   Rebuild.sh contains Ansible-playbooks tasks to deploy Kubernetes cluster, execute the following command under k8s-infrastructure directory:

   ```
   ./rebuild.sh
   ```

7. Post Build

   Post_build.sh includes Ansible-play tasks for post deployment and Tensorflow installation, GPU utilization monitoring customized exporter deployment, pushgateway deployment, etc.. Simply execute post_build.sh to get needed Ansible-playbooks executed.

   ```
   ./post_build.sh
   ```

   *The entire deployment takes about 15 minutes.

8. Test GPU consumption

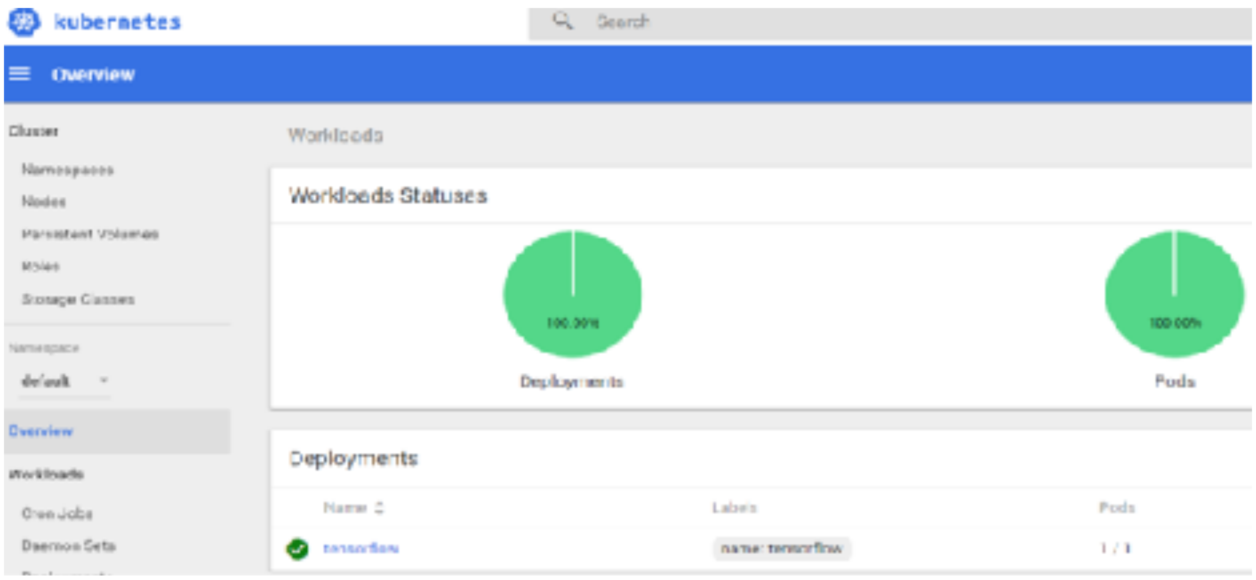
   Now Kubernetes cluster is built, with necessary components like monitoring tools, tensorflow installed. To test GPU consumption with Tensorflow, use the sample python code below:

   ```
   import tensorflow as tf
   hello = tf.constant('Hello, TensorFlow!')
   sess = tf.Session()
   print(sess.run(hello))
   a = tf.constant(20)
   b = tf.constant(50)
   print(sess.run(a + b))
   ```

Dashboards:

View Kubernetes dashboard, Prometheus dashboard, Grafana dashboard, Alert-manager dashboard, and Kube-API by following Kube proxy and port-forwarding methods:

1. Kubernetes Dashboard
    a. On local machine, install kubectl
    b. Create a kubeconfig file with needed server and admin information
    c. *./kubectl.exe --kubeconfig kubeconfig proxy*
    d. Go to localhost:8001 in browser from your local machine, you will see Kubernetes dashboard as below:



2. Prometheus Dashboard
    a. *./kubectl.exe port-forward -n monitoring prometheus-kube-prometheus-0 9090 --kubeconfig=kubeconfig*
    b. Go to localhost:9090 in browser from your local machine, then we will see the dashboard.

3. Grafana Dashboard
    a. *./kubectl.exe port-forward $(./kubectl.exe get pods --selector=app=kube-prometheus-grafana –n monitoring --output=jsonpath="{.items..metadata.name}" --kubeconfig=kubeconfig) -n monitoring 3000 --kubeconfig=kubeconfig*
    b. Go to localhost:3000 to check Grafana dashboard

4. Alertmanager Dashboard

   a. *./kubectl.exe port-forward -n monitoring alertmanager-kube-prometheus-0 9093 --kubeconfig=kubeconfig*
   b. Go to localhost:9093 on your browser to check Alert-manager dashboard

5. GPU Utilization Monitoring

   Other than monitoring the Kubernetes Cluster with monitoring listed from 1-4, we will want to monitoring GPU utilization as well.

   a. One way to monitor core GPU utilization is by NVIDIA command: *nvidia-smi* on Nodes with GPU

   b. Also, we could leverage Prometheus and Node_Exporter to monitor GPU utilization. However, in order to do so, we need to use a customized GPU_Node_Expertor. Nvidia-smi command has a switch to export data as XML, so it could be converted to a Textfile Collector. And then, I will leverage Prometheus Pushgateway and NodePort for GPU_Node_Exporter usage. To view Pushgateway with the GPU utilization, execute following command on local machine:
      i. *./kubectl.exe port-forward $(./kubectl.exe get pods --selector=app=pushgateway -n monitoring --output=jsonpath="{.items..metadata.name}" --kubeconfig=kubeconfig) -n monitoring 9091 --kubeconfig=kubeconfig*
      ii. Then check localhost:9091 with browser, we will see the dashboard

Conclusion

There are different ways to consume Kubernetes clusters, and more components should be and can be added on in a production environment, as examples to mention:

- CI/CD
  - Continuous integration and development
- Load Balancer
  - Even though Kube-proxy act as load balancer with in the Kubernetes clusters. Ideally, we will also need external and internal load balancers for efficiently distributing incoming traffic across backend servers
- Persistent Storage

    - Data, data… It is not ideal to store data in containers directly. There are many ways of leveraging storage layer for both on-premises (Ceph RDB, GlusterFS, etc.) and cloud (AWS EBS, Google Persistent Disks, etc.) options. In this proof of concept solution, we leverage Google Cloud VM with standard persistent disks attached.
- **AND MORE...**

As a proof of concept, I focused on fully automated Kubernetes cluster build and rebuild, monitoring, machine learning framework deployment, and testing GPU on Kubernetes consumption by Tensorflow.

Reference:

1. https://kubernetes.io/
2. https://github.com/kubernetes-incubator/kubespray
3. https://collabnix.com/docker-prometheus-pushgateway-for-nvidia-gpu-metrics-monitoring/
4. Nvidia p100