ASSIGNMENT 6

1. What is Kubeflow?

The Kubeflow project is dedicated to making deployments of machine learning (ML) workflows on Kubernetes simple, portable and scalable. Our goal is not to recreate other services, but to provide a straightforward way to deploy best-of-breed open-source systems for ML to diverse infrastructures. Anywhere you are running Kubernetes, you should be able to run Kubeflow.

Kubeflow is the machine learning toolkit for Kubernetes.

To use Kubeflow, the basic workflow is:

Download and run the Kubeflow deployment binary.

Customize the resulting configuration files.

Run the specified script to deploy your containers to your specific environment.

You can adapt the configuration to choose the platforms and services that you want to use for each stage of the ML workflow:

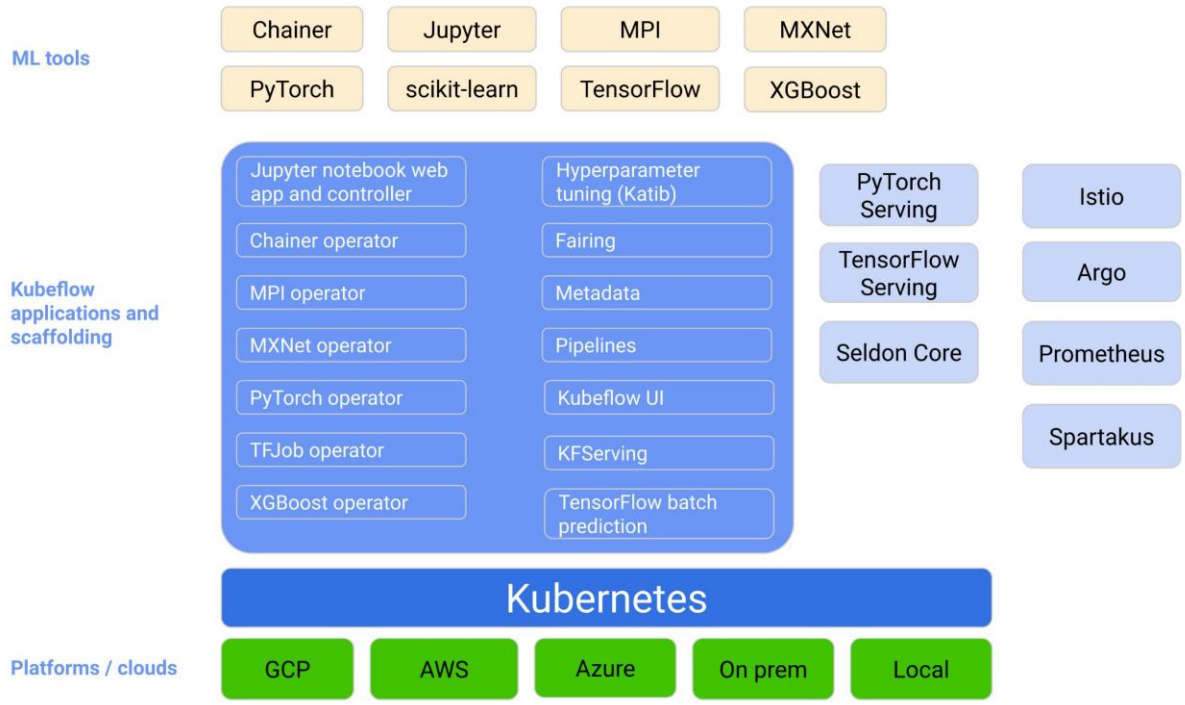
data preparation

model training,

prediction serving

service management

You can choose to deploy your Kubernetes workloads locally, on-premises, or to a cloud environment. Architecture is below



2. What is the difference between Kubeflow and Kubernetes?

Kubernetes is an open source orchestration system for Docker containers. It handles scheduling onto nodes in a compute cluster and actively manages workloads to ensure that their state matches the users declared intentions.

The Kubeflow project is dedicated to making Machine Learning on Kubernetes easy, portable and scalable by providing a straightforward way for spinning up best of breed OSS solutions.

Kubernetes helps applications adopt end-to-end automation by offering a rich library of declarative APIs for managing microservices. Kubernetes takes care of resource management, job allocation, and other operational problems that have traditionally been time-consuming. Kubeflow allows engineers to focus on writing ML algorithms instead of managing their operations.

3. What do you mean by Pod Disruption Budget?

A disruption, refers to an event when a pod needs to be killed and respawned. Disruptions are inevitable, and they would need to be handled delicately, otherwise we will have outage. Imagine when we have a service but none of the backing pods are available. Bad, right?

PodDisruptionBudget (pdb) is a useful layer of defence provided by Kubernetes to deal with this kind of issue.

If you have a kubernetes cluster currently running in production, try running

kubectl get pdb --all-namespaces

What is pdb?

pdb stands for PodDisruptionBudget. Kubernetes API introduced this resource in the initial release of version 1.0. The API remained fairly stable until version 1.21, when it was graduated and promoted to policy/v1 from policy/v1beta1. policy/v1beta1 is currently deprecated, and will be removed in version 1.25.

A pdb defines the budget of voluntary disruption. In essence, a human operator is letting the cluster aware of a minimum threshold in terms of available pods that the cluster needs to guarantee in order to ensure a baseline availability or performance. The word budget is used as in error budget, in the sense that any voluntary disruption within this budget should be acceptable.

A typical pdb looks like

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: pdb
spec:
  minAvailable: 1
  selector:
    matchLabels:
      app: nginx
```

If we take a closer look at this sample, we will notice

It selects other resources based on labels

It demands that there needs to be at least one pod running.

There are three possible fields in a pdb:

.spec.selector (required): a selector field to specify a resource on which this pdb is applied. A pdb can be applied on:

Deployment

ReplicationController

ReplicaSet

StatefulSet

Most frequently, pdb is used in conjunction with Deployment. You can also use pdb with arbitrary controllers. It is beyond the scope of this blog, though.

one .spec.minAvailable or one .spec.maxUnavailable (required. either but not together): You can define minAvailable or maxUnavailable in absolute digits (i.e. have at least two pods available / at most two pods unavailable) or in percentage (i.e. have at least 10% of pods available / at most 10% pods unavailable)

4. What do you mean by Container Orchestration?

Container orchestration automates the deployment, management, scaling, and networking of containers. Enterprises that need to deploy and manage hundreds or thousands of Linux® containers and hosts can benefit from container orchestration.

Container orchestration can be used in any environment where you use containers. It can help you to deploy the same application across different environments without needing to redesign it. And microservices in containers make it easier to orchestrate services, including storage, networking, and security.

Container orchestration tools

Kubernetes is an open source container orchestration tool that was originally developed and designed by engineers at Google.

Kubernetes orchestration allows you to build application services that span multiple containers, schedule containers across a cluster, scale those containers, and manage their health over time.

Kubernetes eliminates many of the manual processes involved in deploying and scaling containerized applications

Main components of Kubernetes:

Cluster: A control plane and one or more compute machines, or nodes.

Control plane: The collection of processes that control Kubernetes nodes. This is where all task assignments originate.

Kubelet: This service runs on nodes and reads the container manifests and ensures the defined containers are started and running.

Pod: A group of one or more containers deployed to a single node. All containers in a pod share an IP address, IPC, hostname, and other resources.

5. Why do we need Container Orchestration?

Use container orchestration to automate and manage tasks such as:

Provisioning and deployment

Configuration and scheduling

Resource allocation

Container availability

Scaling or removing containers based on balancing workloads across your infrastructure

Load balancing and traffic routing

Monitoring container health

Configuring applications based on the container in which they will run

Keeping interactions between containers secure

6. Are Kubernetes and Docker Related?

Docker is a technology for creating and running containers, while Kubernetes is a container orchestration technology.

The important thing to understand about Docker and Kubernetes is that one is a technology for defining and running containers, and the other is a container orchestration framework that represents and manages containers within a web application. Kubernetes does not make containers. Rather, it relies upon a container realization technology such as Docker to make them

Docker is a technology that is used to create and run software containers. A container is a collection of one or more processes, organized under a single name and identifier. A container is isolated from the other processes running within a computing environment, be it a physical computer or a virtual machine (VM).

On the other hand, Kubernetes is a container orchestration technology.

Kubernetes groups the containers that support a single application or microservice into a pod. A pod is exposed to the network by way of another Kubernetes abstraction called a service. In short, the network knows about Kubernetes services and a service knows about the pod(s) that has its logic. Within each pod is one or many containers that realize the logic in the given pod.

7. What do you mean by Kube-proxy?

kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept. kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.