

Introduction to Kubeflow on Kubernetes Engine

1. Introduction



Kubeflow

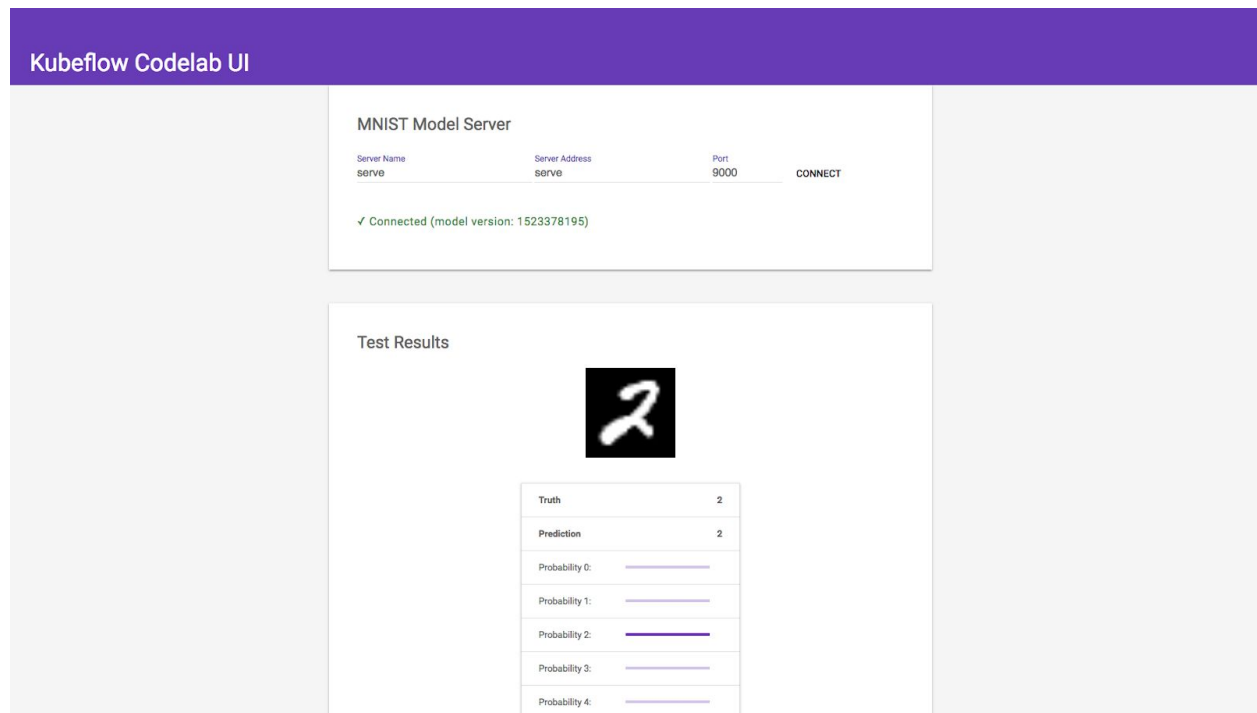
As datasets continue to expand and models become more complex, distributing machine learning (ML) workloads across multiple nodes is becoming more attractive. Unfortunately, breaking up and distributing a workload can add both computational overhead, and a great deal more complexity to the system. Data scientists should be able to focus on ML problems, not DevOps.

Fortunately, distributed workloads are becoming easier to manage, thanks to [Kubernetes](#). Kubernetes is a mature, production ready platform that gives developers a simple API to deploy programs to a cluster of machines as if they were a single piece of hardware. Using Kubernetes, computational resources can be added or removed as desired, and the same cluster can be used to both train and serve ML models.

This course will serve as an introduction to [Kubeflow](#), an open-source project which aims to make running ML workloads on Kubernetes simple, portable and scalable. Kubeflow adds some resources to your cluster to assist with a variety of tasks, including training and serving models and running [Jupyter Notebooks](#). It also extends the Kubernetes API by adding new [Custom Resource Definitions \(CRDs\)](#) to your cluster, so machine learning workloads can be

treated as first-class citizens on Kubernetes. For this course, it is expected that you are familiar with the basics of the Kubernetes cluster orchestration system.

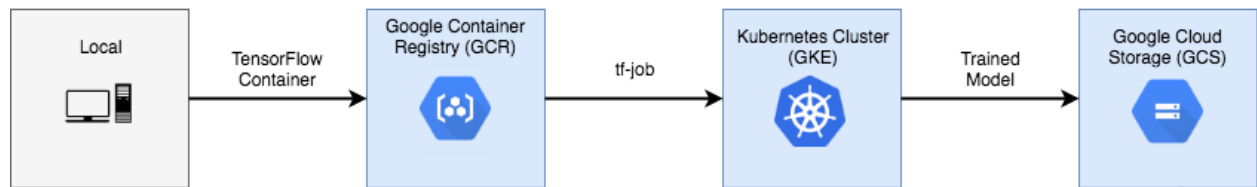
What You'll Build



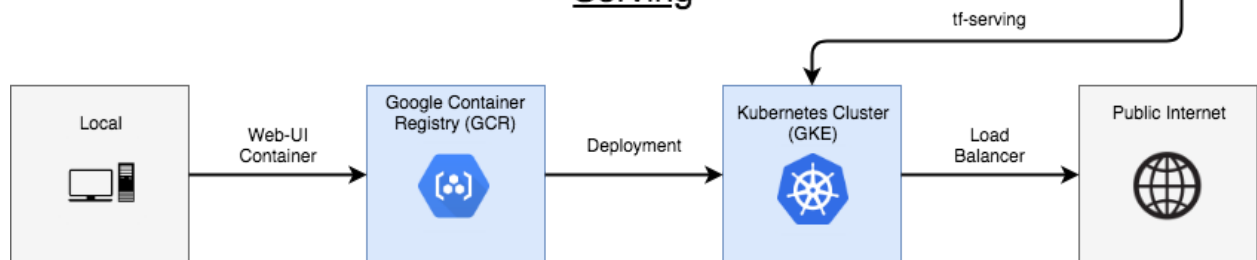
This course will describe how to train and serve a TensorFlow model, and then how to deploy a web interface to allow users to interact with the model over the public internet. You will build a classic handwritten digit recognizer using the MNIST dataset.

The purpose of this course is to get a brief overview of how to interact with Kubeflow. To keep things simple, the model we'll deploy will use CPU-only distributed training. Kubeflow's [documentation](#) has more information when you are ready to explore further.

Training



Serving



What You'll Learn

- How to build a training image using your TensorFlow model code
- How to set up and run a distributed training job using [TFJob](#).
- How to serve the resulting model using [TensorFlow Serving](#).
- How to deploy a web app that uses the model.

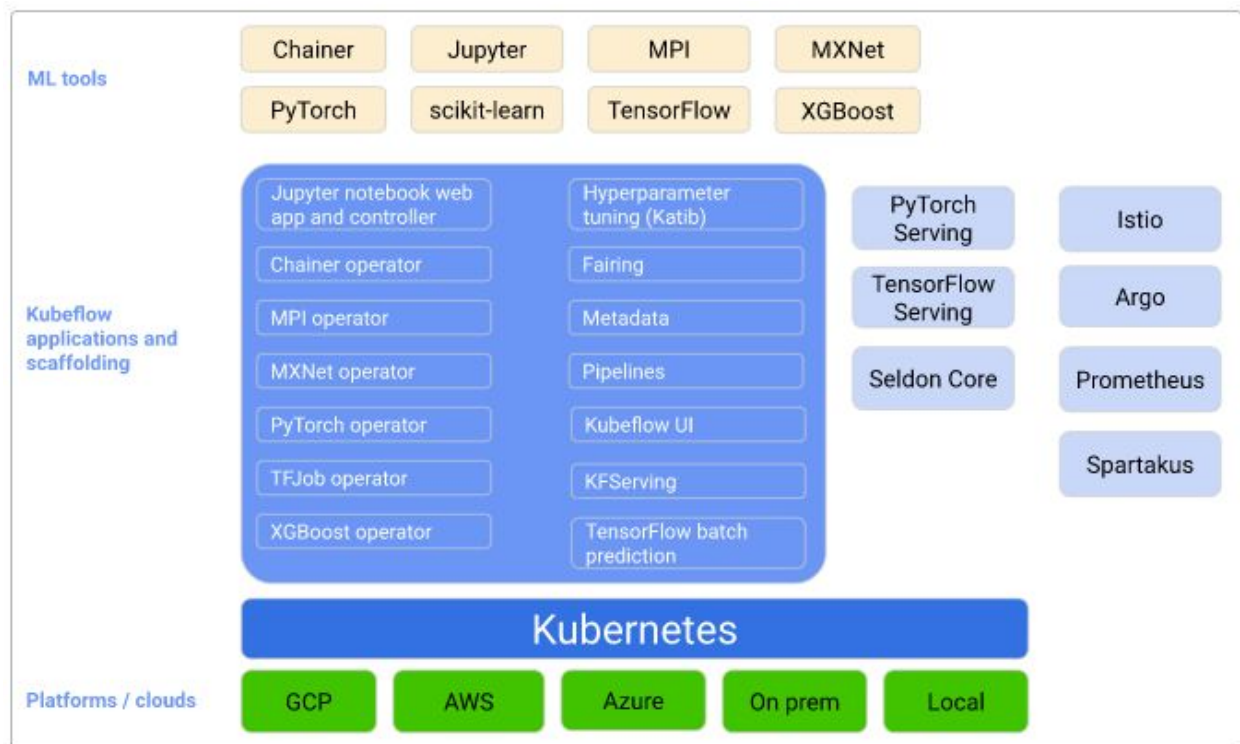
What You'll Need

- Install [minikube](#) to your device
- Access to the Google Cloud Shell, available in the [Google Cloud Console](#)
- If you'd prefer to run the codelab from your local machine, you'll need to have [gcloud](#), [kubectl](#), and [docker](#) installed.

2. Setting Up the Kubeflow Environment

Kubeflow is a platform for data scientists who want to build and experiment with ML pipelines. Kubeflow is also for ML engineers and operational teams who want to deploy ML systems to various environments for development, testing, and production-level serving.

Kubeflow is *the ML toolkit for Kubernetes*. The following diagram shows Kubeflow as a platform for arranging the components of your ML system on top of Kubernetes:



Kubeflow builds on [Kubernetes](#) as a system for deploying, scaling, and managing complex systems.

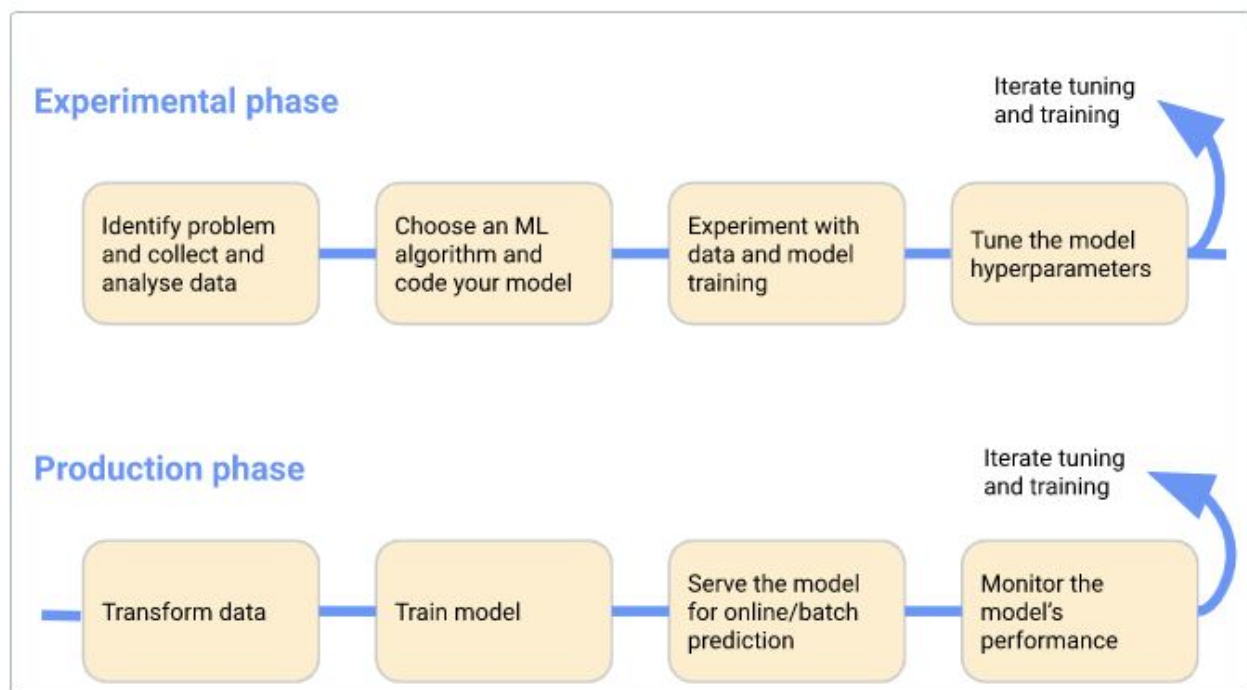
Using the Kubeflow configuration interfaces, you can specify the ML tools required for your workflow. Then you can deploy the workflow to various clouds, local, and on-premises platforms for experimentation and for production use.

Introducing the ML workflow

When you develop and deploy an ML system, the ML workflow typically consists of several stages. Developing an ML system is an iterative process. You need to evaluate the output of

various stages of the ML workflow, and apply changes to the model and parameters when necessary to ensure the model keeps producing the results you need.

For the sake of simplicity, the following diagram shows the workflow stages in sequence. The arrow at the end of the workflow points back into the flow to indicate the iterative nature of the process:



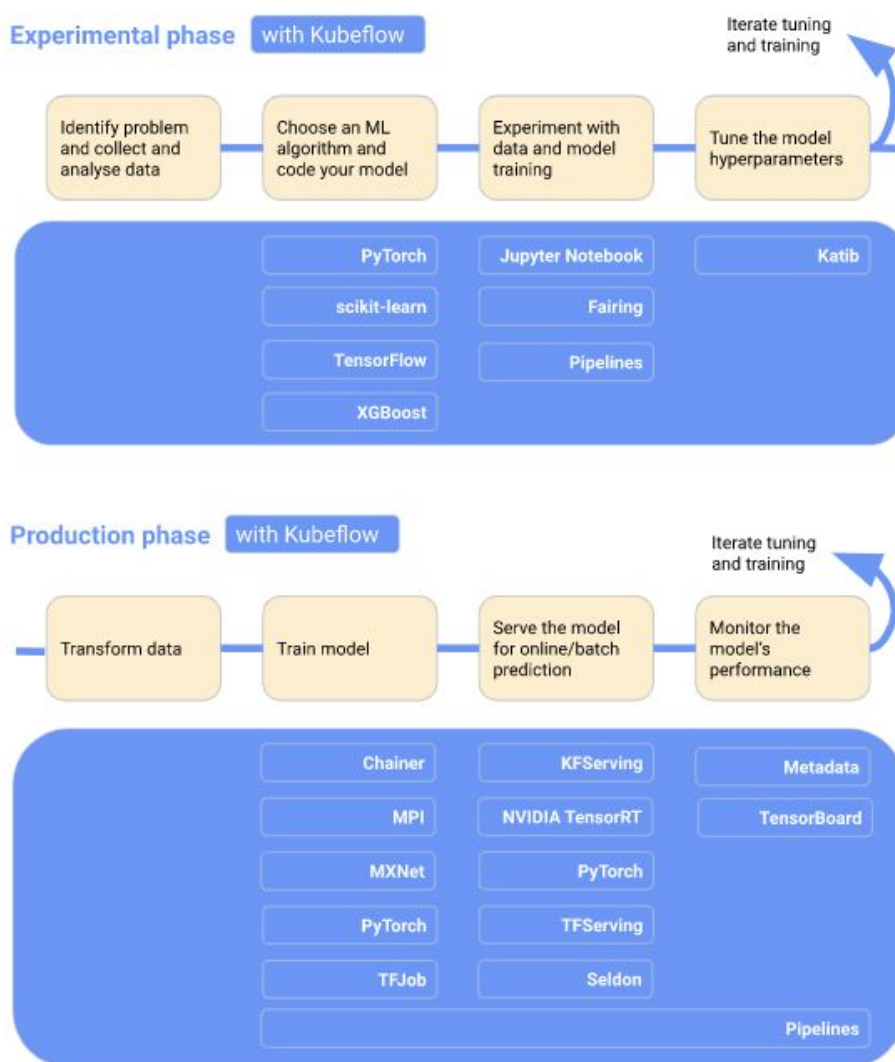
Looking at the stages in more detail:

- In the experimental phase, you develop your model based on initial assumptions, and test and update the model iteratively to produce the results you're looking for:
 - Identify the problem you want the ML system to solve.
 - Collect and analyze the data you need to train your ML model.
 - Choose an ML framework and algorithm, and code the initial version of your model.
 - Experiment with the data and with training your model.
 - Tune the model hyperparameters to ensure the most efficient processing and the most accurate results possible.
- In the production phase, you deploy a system that performs the following processes:
 - Transform the data into the format that your training system needs. To ensure that your model behaves consistently during training and prediction, the transformation process must be the same in the experimental and production phases.

- Train the ML model.
- Serve the model for online prediction or for running in batch mode.
- Monitor the model's performance, and feed the results into your processes for tuning or retraining the model.

Kubeflow components in the ML workflow

The next diagram adds Kubeflow to the workflow, showing which Kubeflow components are useful at each stage:



To learn more, read the following guides to the Kubeflow components:

- Kubeflow includes services for spawning and managing [Jupyter notebooks](#). Use notebooks for interactive data science and experimenting with ML workflows.
- [Kubeflow Pipelines](#) is a platform for building, deploying, and managing multi-step ML workflows based on Docker containers.
- Kubeflow offers several [components](#) that you can use to build your ML training, hyperparameter tuning, and serving workloads across multiple platforms.

Installing Kubeflow

It's important that you have some knowledge of the following systems and tools:

- [Kubernetes](#)
- [Kustomize](#)

There are various ways to install Kubeflow. Choose one of the following options to suit your environment (public cloud, existing Kubernetes cluster, or a single-node cluster which you can use on a desktop or server or in the cloud).

Installing Kubeflow on Minikube

Recall that Minikube runs a simple, single-node Kubernetes cluster inside a virtual machine (VM).

- To use kubeflow, first you will need to install a hypervisor, if you do not already have one installed. Install [Virtual Box](#) or [VMware Fusion](#) for Mac OS X or KVM for Linux and CentOS
- To check if virtualization is supported on Windows 8 and above, run the following command on your Windows terminal or command prompt.

```
systeminfo
```

If you see the following output, virtualization is supported on Windows.

```
Hyper-V Requirements: VM Monitor Mode Extensions: Yes
```

```
Virtualization Enabled In Firmware: Yes
```

```
Second Level Address Translation: Yes
```

```
Data Execution Prevention Available: Yes
```

If you see the following output, your system already has a Hypervisor installed and you can skip the next step.

Hyper-V Requirements: A hypervisor has been detected. Features required for Hyper-V will not be displayed.

Install Kubectl

GCloud SDK

```
$ gcloud components install kubectl
```

Mac OS X

```
$ brew install kubectl
```

Ubuntu

```
$ sudo apt-get update && sudo apt-get install -y apt-transport-https
```

```
$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
```

```
$ sudo touch /etc/apt/sources.list.d/kubernetes.list
```

```
$ echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a  
/etc/apt/sources.list.d/kubernetes.list
```

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

```
$ sudo apt-get install -y kubectl
```

CentOS

```
$ cat <<EOF > /etc/yum.repos.d/kubernetes.repo
```

```
[kubernetes]
```

```
name=Kubernetes
```

```
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
```

```
enabled=1
```

```
gpgcheck=1
```

```
repo_gpgcheck=1
```

```
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg  
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
```

```
EOF
```



```
$ sudo yum install -y kubectl
```

Install & Start Minikube

Please see [detailed instructions](#) for Minikube installation. For quick setup instructions follow along below.

Mac OS X

```
$ brew cask install minikube
```

Another way of doing this is by following the commands below:

```
$ curl -Lo minikube https://storage.googleapis.com/minikube/releases/v0.28.0/minikube-darwin-amd64
```

```
$ chmod +x minikube
```

```
$ sudo mv minikube /usr/local/bin/
```

Ubuntu or CentOS

```
$ curl -Lo minikube https://storage.googleapis.com/minikube/releases/v0.28.0/minikube-linux-amd64
```

```
$ chmod +x minikube
```

```
$ sudo mv minikube /usr/local/bin/
```

Start your minikube cluster

```
$ minikube start --cpus 4 --memory 8096 --disk-size=40g
```

Windows

For **Windows**, **install** VirtualBox or Hyper-V first. **Minikube** is distributed in binary form: GitHub Repo . Download the **minikube**-installer.exe file, and execute the installer. This should automatically add **minikube**.exe to your path with an uninstaller available as well.

Notes:

1. These are the minimum recommended settings on the VM created by minikube for kubeflow deployment. You are free to adjust them **higher** based on your host machine capabilities and workload requirements.
2. Using certain hypervisors might require you to set -vm-driver option [specifying the driver](#) you want to use.

In case, you have the default minikube VM already created (following detailed installation instructions), please use the following to update the VM.

```
$ minikube stop
```

```
$ minikube delete
```

```
$ minikube start --cpus 4 --memory 8096 --disk-size=40g
```

Installing Kubeflow on a public cloud

Choose the Kubeflow deployment guide for your chosen cloud:

- To use Kubeflow on Google Cloud Platform (GCP) and Kubernetes Engine (GKE), follow the [GCP deployment guide](#).
- To use Kubeflow on Amazon Web Services (AWS), follow the [AWS deployment guide](#).
- To use Kubeflow on Microsoft Azure Kubernetes Service (AKS), follow the [AKS deployment guide](#).
- To use Kubeflow on IBM Cloud (IKS), follow the [IKS deployment guide](#).
- To use Kubeflow on OpenShift, follow the [OpenShift deployment guide](#).

Installing Kubeflow on an existing Kubernetes cluster

You can use the following options to run Kubeflow on a single-node Kubernetes cluster, which you can use on a desktop or server or in the cloud.

Choose the guide for your operating system or environment:

- To use Kubeflow on Linux, follow the [Linux deployment guide](#).
- To use Kubeflow on MacOS, follow the [MacOS deployment guide](#).
- To use Kubeflow on Windows, follow the [Windows deployment guide](#).
- To use MiniKF (mini Kubeflow) on Google Cloud Platform, follow the guide to [MiniKF on GCP](#).

Configuration quick reference

Below is a matrix of the platforms where you can deploy Kubeflow and the corresponding manifest files that specify the default configuration for each platform. The matrix shows the same manifests as the installation guides. The matrix is therefore an alternative way of accessing the information in the [installation section above](#).

Deployment platform	Manifest	Deployment guide
Existing Kubernetes cluster using a standard Kubeflow installation	kfctl_k8s_istio.v1.0.2.yaml	Docs
Existing Kubernetes cluster using Dex for authentication	kfctl_istio_dex.v1.0.2.yaml	Docs
Amazon Web Services (AWS) using the standard setup	kfctl_aws.v1.0.2.yaml	Docs
Amazon Web Services (AWS) with authentication	kfctl_aws_cognito.v1.0.2.yaml	Docs
Microsoft Azure	kfctl_k8s_istio.v1.0.2.yaml	Docs
Google Cloud Platform (GCP) with Cloud Identity-Aware Proxy (Cloud IAP)	kfctl_gcp_iap.v1.0.2.yaml	Docs
IBM Cloud (IKS)	kfctl_ibm.v1.0.2.yaml	Docs
OpenShift	kfctl_openshift.yaml	Docs

Installing command line tools

The following information is useful if you need or prefer to use command line tools for deploying and managing Kubeflow:

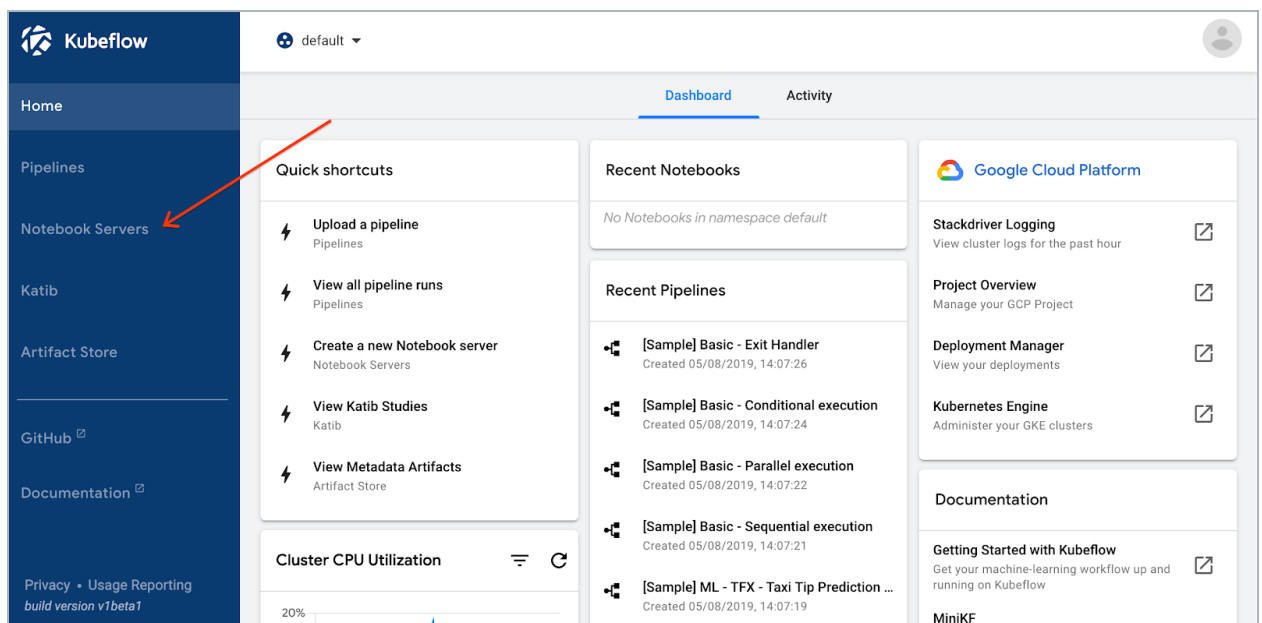
- Download the kfctl binary from the [Kubeflow releases page](#).
- Follow the kubectl installation and setup instructions from the [Kubernetes documentation](#). As described in the Kubernetes documentation, your kubectl version

must be within one minor version of the Kubernetes version that you use in your Kubeflow cluster.

- Follow the Kustomize installation and setup instructions from the guide to [kustomize in Kubeflow](#). Make sure that you have the minimum required version of kustomize: 2.0.3 or later.

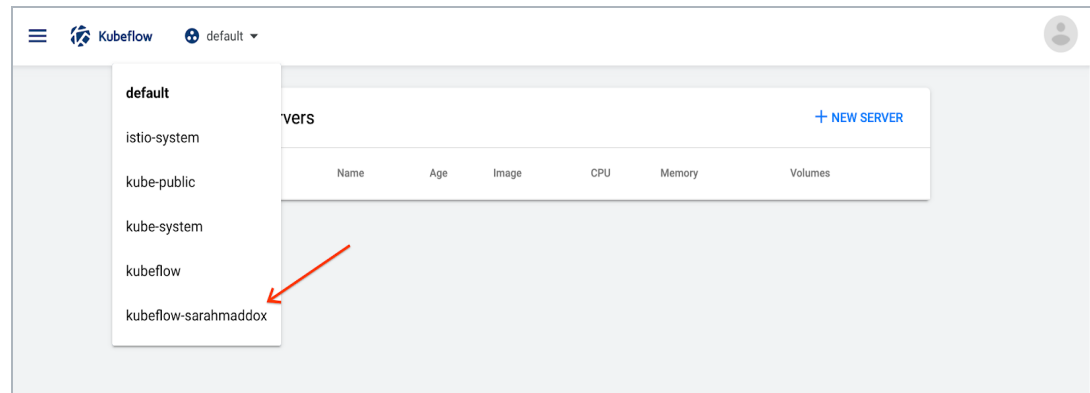
Create a Jupyter notebook server and add a notebook

1. Click **Notebook Servers** in the left-hand panel of the Kubeflow UI to access the Jupyter notebook services deployed with Kubeflow:

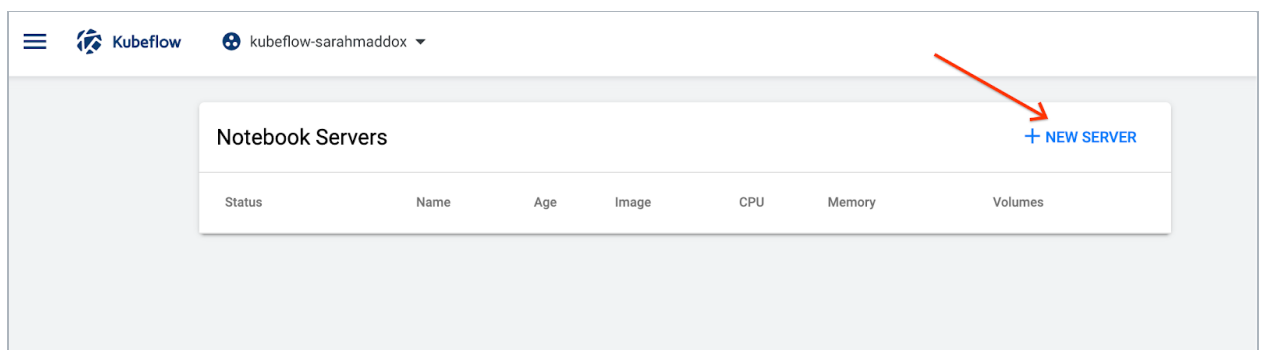


2. Sign in:
 - On GCP, sign in using your Google Account. (If you have already logged in to your Google Account you may not need to log in again.)
 - On all other platforms, sign in using any username and password.
3. Select a namespace:
 - Click the namespace dropdown to see the list of available namespaces.

- Choose the namespace that corresponds to your Kubeflow profile. (See the page on [multi-user isolation](#) for more information about namespaces.)



4. Click **NEW SERVER** on the **Notebook Servers** page



You should see a page for entering details of your new server. Here is a partial screenshot of the page.

The screenshot shows the configuration page for a new server. It includes the following sections:

- Name**: A section for specifying the name and namespace. It contains two input fields: 'Name' and 'Namespace'. The 'Namespace' field is pre-filled with 'kubeflow-sarahmaddox'.
- Image**: A section for specifying the Docker image. It includes a description: 'A starter Jupyter Docker Image with a baseline deployment and typical ML packages.' Below this is a checkbox for 'Custom Image'. If checked, there would be an 'Image' input field. Currently, a dropdown menu shows the selected image: 'gcr.io/kubeflow-images-public/tensorflow-1.13.1-notebook-cpu:v0.5.0'.
- CPU / RAM**: A section for specifying the total amount of CPU and RAM reserved by the server. It includes a description: 'Specify the total amount of CPU and RAM reserved by your Notebook Server. For CPU-intensive workloads, you can choose more than 1 CPU (e.g. 1.5).'

5. Enter a **name** of your choice for the notebook server. The name can include letters and numbers, but no spaces. For example, **my-first-notebook**.
6. KubeFlow automatically updates the value in the **namespace** field to be the same as the namespace that you selected in a previous step. This ensures that the new notebook server is in a namespace that you can access.
7. Select a Docker **image** for the baseline deployment of your notebook server. You can specify a custom image or choose from a range of standard images:
 - **Custom image:** If you select the custom option, you must specify a Docker image in the form **registry/image:tag**. For guidelines on creating a Docker image for your notebook, see the guide to [creating a custom Jupyter image](#).
 - **Standard image:** Click the **Image** dropdown menu to see the list of available images. The standard Docker images include typical machine learning (ML) packages that you can use within your Jupyter notebooks on this notebook server. Click one of the images to select it.
8. The image names indicate the following choices:
 - A TensorFlow version (for example, tensorflow-1.15.2).
 - cpu or gpu, depending on whether you want to train your model on a CPU or a GPU.

If you choose a GPU image, make sure that you have GPUs available in your KubeFlow cluster. Run the following command to check if there are any GPUs available:

```
kubectl get nodes
```

```
"-o=custom-columns=NAME:.metadata.name,GPU:.status.allocatable.nvidia\.com/gpu"
```

- If you have GPUs available, you can schedule your server on a GPU node by specifying a number of GPUs to be attached to the server under the **GPUs** section at the bottom of the form.
 - KubeFlow version (for example, 1.0.0).
9. *Hint:* If you're not sure which image you need, choose a *standard* image running TensorFlow on a CPU. You need a TensorFlow image to run the example code in the [experiment section](#) below.
 10. Specify the total amount of **CPU** that your notebook server should reserve. The default is 0.5. For CPU-intensive jobs, you can choose more than one CPU (for example, 1.5).
 11. Specify the total amount of **memory** (RAM) that your notebook server should reserve. The default is 1.0Gi.
 12. Specify a **workspace volume** to hold your personal workspace for this notebook server. KubeFlow provisions a [Kubernetes persistent volume \(PV\)](#) for your workspace volume. The PV ensures that you can retain data even if you destroy your notebook server.

- The default is to create a new volume for your workspace with the following configuration:
 - Name: The volume name is synced with the name of the notebook server, and has the form workspace-<server-name>. When you start typing the notebook server name, the volume name appears. You can edit the volume name, but if you later edit the notebook server name, the volume name changes to match the notebook server name.
 - Size: 10Gi
 - Access mode: ReadWriteOnce. This setting means that the volume can be mounted as read-write by a single node. See the [Kubernetes documentation](#) for more details about access modes.
 - Mount point: /home/jovyan
- Alternatively, you can point the notebook server at an existing volume by specifying the name of the existing volume.

13. *(Optional)* Specify one or more **data volumes** if you want to store and access data from the notebooks on this notebook server. You can add new volumes or specify existing volumes. Kubeflow provisions a [Kubernetes persistent volume \(PV\)](#) for each of your data volumes.

14. *(Optional)* Specify one or more additional **configurations** as a list of PodDefault labels. To make use of this option, you must create a [PodDefault manifest](#). In the PodDefault manifest, you can specify configurations including volumes, secrets, and environment variables. Kubeflow matches the labels in the '**configurations**' field against the properties specified in the PodDefault manifest. Kubeflow then injects these configurations into all the notebook Pods on this notebook server.

For example, enter the label addgcpsecret in the **configurations** field to match to a PodDefault manifest containing the following configuration:

matchLabels:

addgcpsecret: "true"

For in depth information on PodDefault usage, see the [admission-webhook README](#)

15. *(Optional)* Schedule one or more **GPUs** for your notebook server, as discussed in the section on [specifying your Docker image](#).

You can find more details about scheduling GPUs in the [Kubernetes documentation](#).

16. *(Optional)* Change the setting for **enable shared memory**. The default is that shared memory is enabled. Some libraries like PyTorch use shared memory for multiprocessing. Currently there is no implementation in Kubernetes to activate shared memory. As a workaround, Kubeflow creates an empty directory at /dev/shm.

17. Click **LAUNCH**. You should see an entry for your new notebook server on the **Notebook Servers** page, with a spinning indicator in the **Status** column. It can take a few minutes to set up the notebook server.

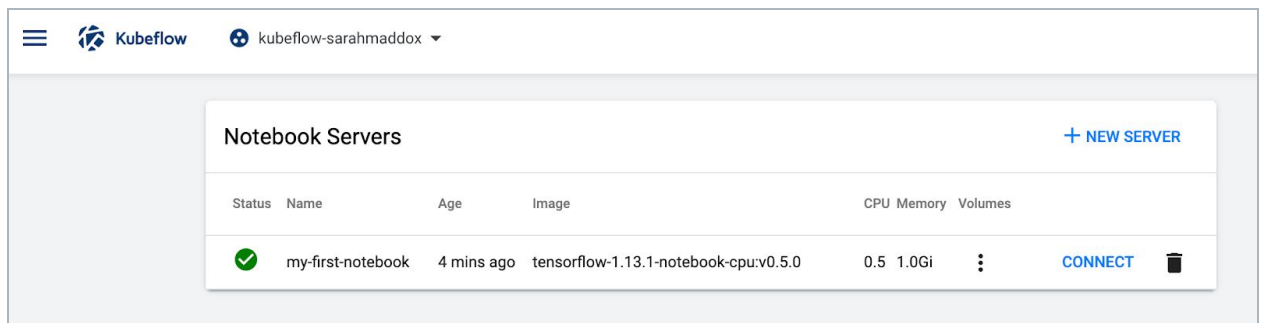
You can check the status of your Pod by hovering your mouse cursor over the icon in the **Status** column next to the entry for your notebook server. For example, if th/e image is downloading then the status spinner has a tooltip that says ContainerCreating. Alternatively, you can check the Pod status by entering the following command:

```
kubectl -n <NAMESPACE> describe pods jupyter-<USERNAME>
```

Where <NAMESPACE> is the namespace you specified earlier (default kubeflow) and <USERNAME> is the name you used to log in. **A note for GCP users:** If you have IAP turned on, the Pod has a different name. For example, if you signed in as USER@DOMAIN.EXT the Pod has a name of the following form:

```
jupyter-accounts-2egoogle-2ecom-3USER-4ODOMAIN-2eEXT
```

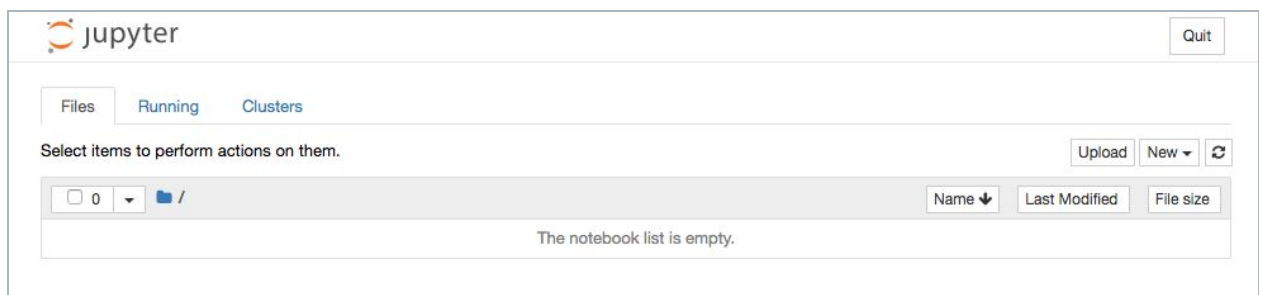
18. When the notebook server provisioning is complete, you should see an entry for your server on the **Notebook Servers** page, with a check mark in the **Status** column:



Status	Name	Age	Image	CPU	Memory	Volumes	
✓	my-first-notebook	4 mins ago	tensorflow-1.13.1-notebook-cpu:v0.5.0	0.5	1.0Gi		CONNECT

19. Click **CONNECT** to start the notebook server.

20. When the notebook server is running, you should see the Jupyter dashboard interface. If you requested a new workspace, the dashboard should be empty of notebooks:



21. Click **Upload** to upload an existing notebook, or click **New** to create an empty notebook. You can read about using notebooks in the [Jupyter document](#)

Further Readings

- https://docs.google.com/presentation/d/1xgE1y6ltKYpk2jhTq_EqlUu6o3ohU0g2WJ4goaJOD_A/edit#slide=id.g89c30b96c7_0_413
- <https://github.com/IBM/KubeflowDojo/>
- https://www.youtube.com/watch?v=qfhb_RyOwAl
- <https://www.youtube.com/watch?v=H0LopPvAJtE>
- https://www.youtube.com/watch?v=AYleNtXLT_k