PROJECT INTRO

Data science workflows on Kubernetes with Kubeflow Pipelines

Kubeflow Pipelines are a great way to build portable, scalable machine learning workflows. It is one part of a larger Kubeflow ecosystem which aims to reduce the complexity and time involved with training and deploying machine learning models at scale.

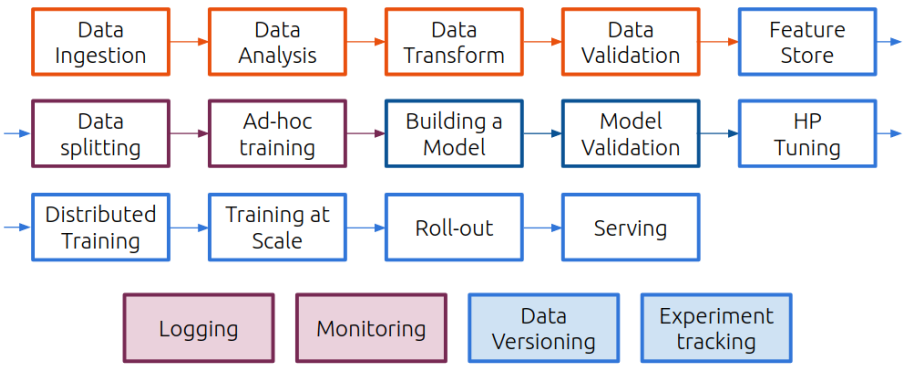
Kubeflow

Why use Kubeflow?

A machine learning workflow can involve many steps and keeping all these steps in a set of notebooks or scripts is hard to maintain, share, and collaborate on, which leads to large amounts of “Hidden Technical Debt in Machine Learning Systems”.

In addition, it is typical that these steps are run on different systems. In an initial phase of experimentation, a data scientist will work at a developer workstation or an on-prem training rig, training at scale will typically happen in a cloud environment (private, hybrid, or public), while inference and distributed training often happens at the Edge.

Containers provide the right encapsulation, avoiding the need for debugging every time a developer changes the execution environment, and Kubernetes brings scheduling and orchestration of containers.



However, managing ML workloads on top of Kubernetes is still a lot of specialized operations work which we don't want to add to the data scientist's role. Kubeflow bridges this gap between AI workloads and Kubernetes, making MLOps more manageable.

What are Kubeflow Pipelines?

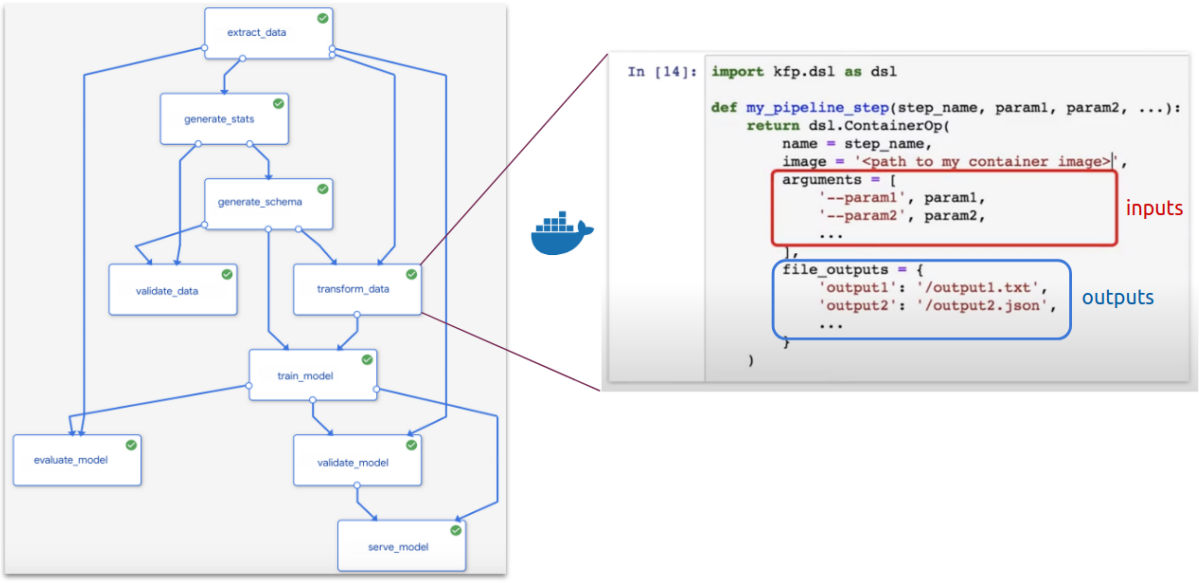
Kubeflow pipelines are one of the most important features of Kubeflow and promise to make your AI experiments reproducible, composable, i.e. made of interchangeable components, scalable and easily shareable.

The Kubeflow pipelines service has the following goals:

- End to end orchestration: enabling and simplifying the orchestration of end to end machine learning pipelines
- Easy experimentation: making it easy for you to try numerous ideas and techniques, and manage your various trials/experiments.
- Easy re-use: enabling you to re-use components and pipelines to quickly cobble together end to end solutions, without having to re-build each time.

A pipeline is a codified representation of a machine learning workflow, analogous to the sequence of steps described in the first image, which includes components of the workflow and their respective dependencies. More specifically, a pipeline is a directed acyclic graph (DAG) with a containerized process on each node, which runs on top of argo.

Each pipeline component, represented as a block, is a self-contained piece of code, packaged

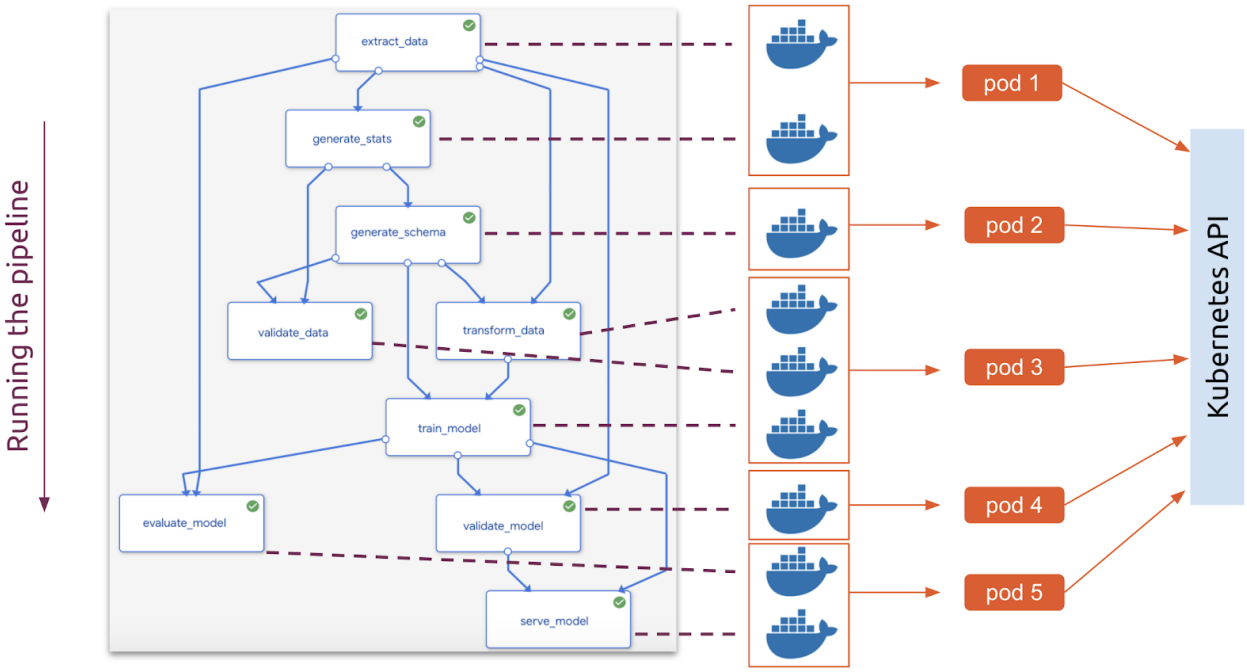


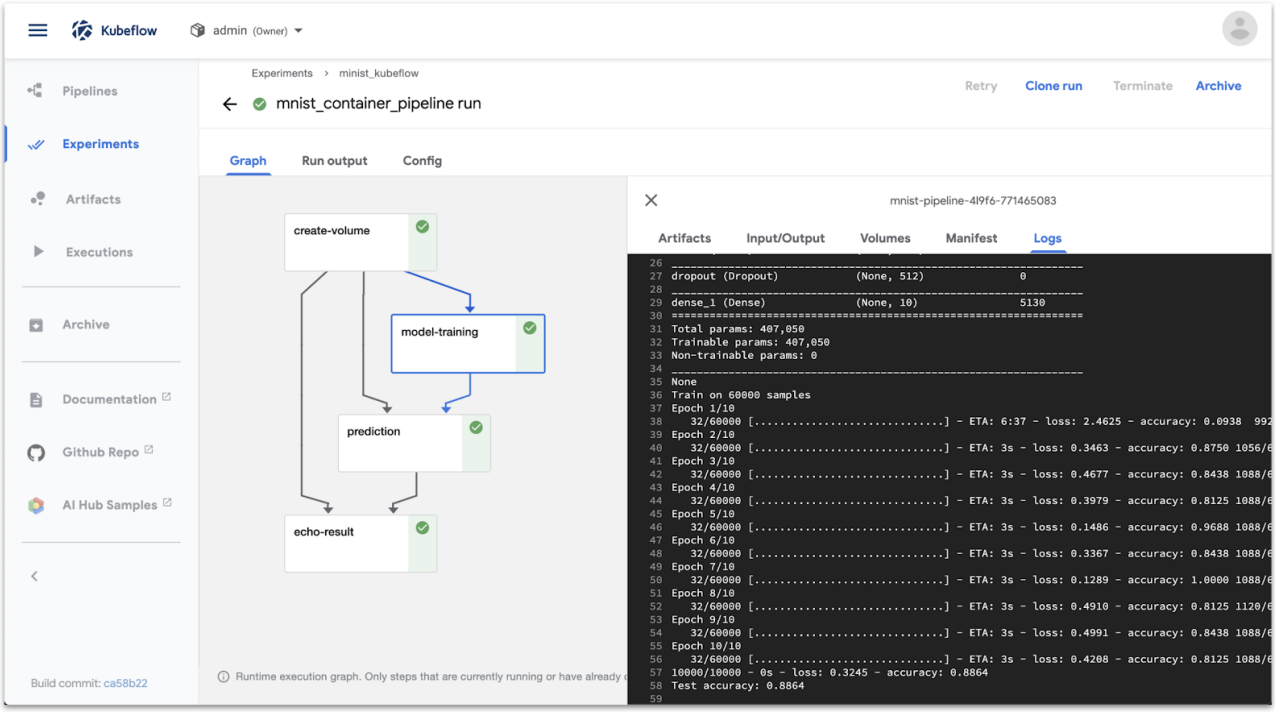
as a Docker image. It contains inputs (arguments) and outputs and performs one step in the pipeline. In the example pipeline, above, the transform_data step requires arguments that are produced as an output of the extract_data and of the generate_schema steps, and its outputs are dependencies for train_model.

Your ML code is wrapped into components, where you can:

1. Specify parameters – which become available to edit in the dashboard and configurable for every run.
2. Attach persistent volumes – without adding persistent volumes, we would lose all the data if our notebook was terminated for any reason.
3. Specify artifacts to be generated – graphs, tables, selected images, models – which end up conveniently stored on the Artifact Store, inside the Kubeflow dashboard.

Finally, when you run the pipeline, each container will now be executed throughout the cluster, according to Kubernetes scheduling, taking dependencies into consideration.





This containerized architecture makes it simple to reuse, share, or swap out components as your workflow changes, which tends to happen.

After running the pipeline, you are able to explore the results on the pipelines UI on the Kubeflow dashboard, debug, tweak parameters, and create more “runs”.

PROJECT

Using Kubeflow for ML CI/CD

The aim of the project is the exploration, training and serving of a machine learning model for either Classification or Time Series Data forecasting by leveraging Kubeflow's main components. The containerized APP to be developed should run on a Kubernetes cluster. A Kubernetes cluster consists of a set of worker machines, called nodes, that run containerized applications. Every cluster has at least one worker node. The worker node(s) host the Pods that are the components of the application workload. The control plane manages the worker nodes and the Pods in the cluster. A node may be a virtual or physical machine, depending on the cluster. In production environments, the control plane usually runs across multiple computers and a cluster usually runs multiple nodes, providing fault-tolerance and high availability.

There are three primary goals for your project:

- Demonstrate an End-to-End kubeflow example
- Present the ML model
- Build a web app able to perform the forecasting supposing that the APP should serve a total number of 50.000 customers and, during the serving time, we can have a maximum of 5000 service requests at the same time.

To this end, you should perform the following tasks:

A) Create a Kubernetes cluster with 1 control node and 1 or more worker nodes. Discuss how to set the cluster (virtual or physical machine(s)).
B) Installing Kubeflow on the Kubernetes cluster
C) Setup the Kubeflow cluster
D) Deploy Kubeflow on the cluster
E) Train a ML model on the cluster
F) Serve the model
G) Query the model via your local machine

H) Automate the steps 1/ preprocess, 2/ train and 3/ model deployment through a kubeflow pipeline.
I) Save the final model, build a web app able to perform the prevision and deploy it on Kubernetes.

Finally, after watching the webinar "Protecting Apps from Hacks in Kubernetes with NGINX", in your final report discuss how to protect the WEB APP developed by cyber-attacks.