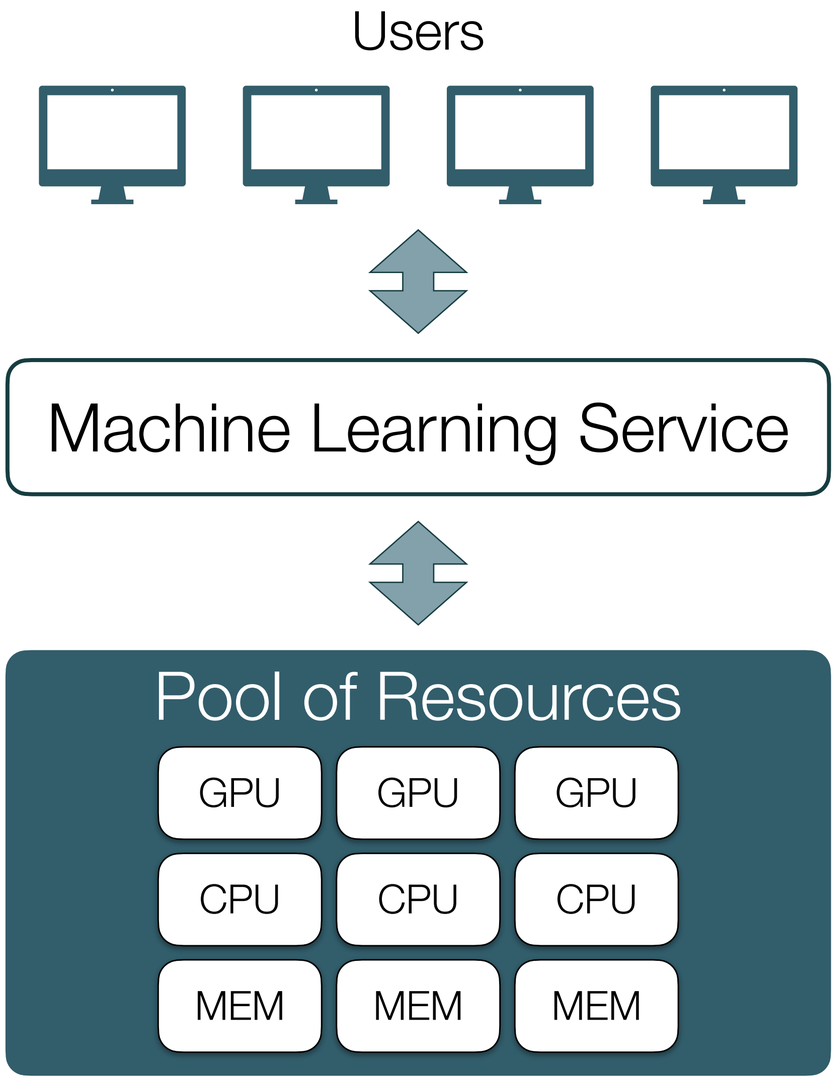
Training and Serving ML workloads with Kubeflow at CERN

Dejan Golubovic, Ricardo Rocha

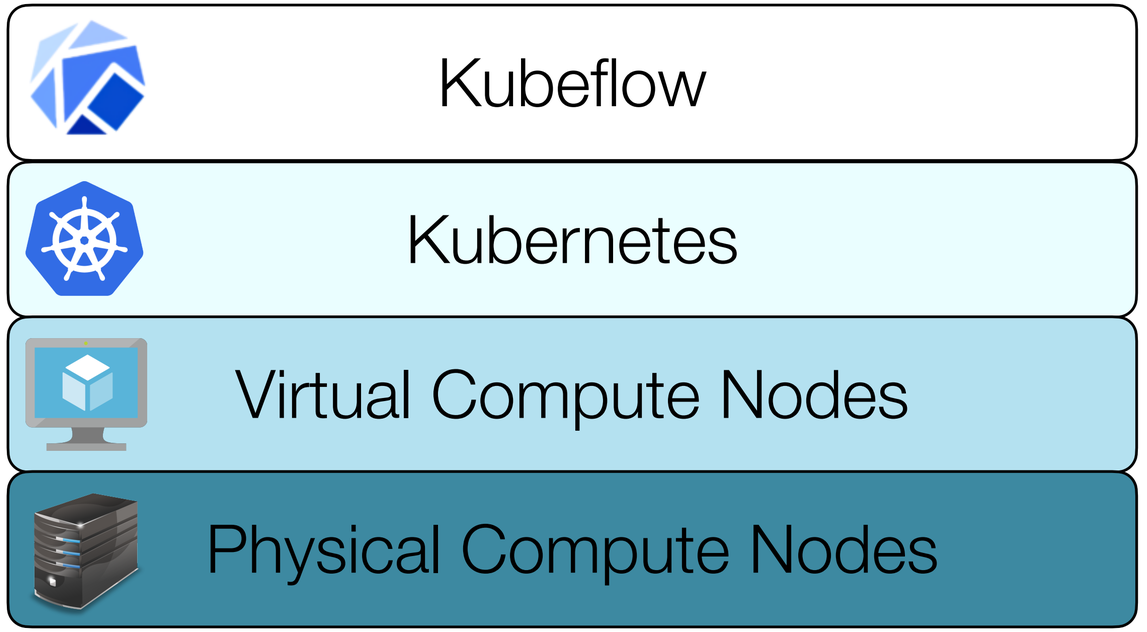
Project Motivation

- Different groups at CERN have machine learning workloads
  - Managing local infrastructure
- Idea - setup a **centralized machine learning service**
- Offer variety of hardware resources to CERN users
- Provide an easy-to-use web interface for ML tasks
- Integrate with existing identity and storage services
- Extend to public clouds when necessary



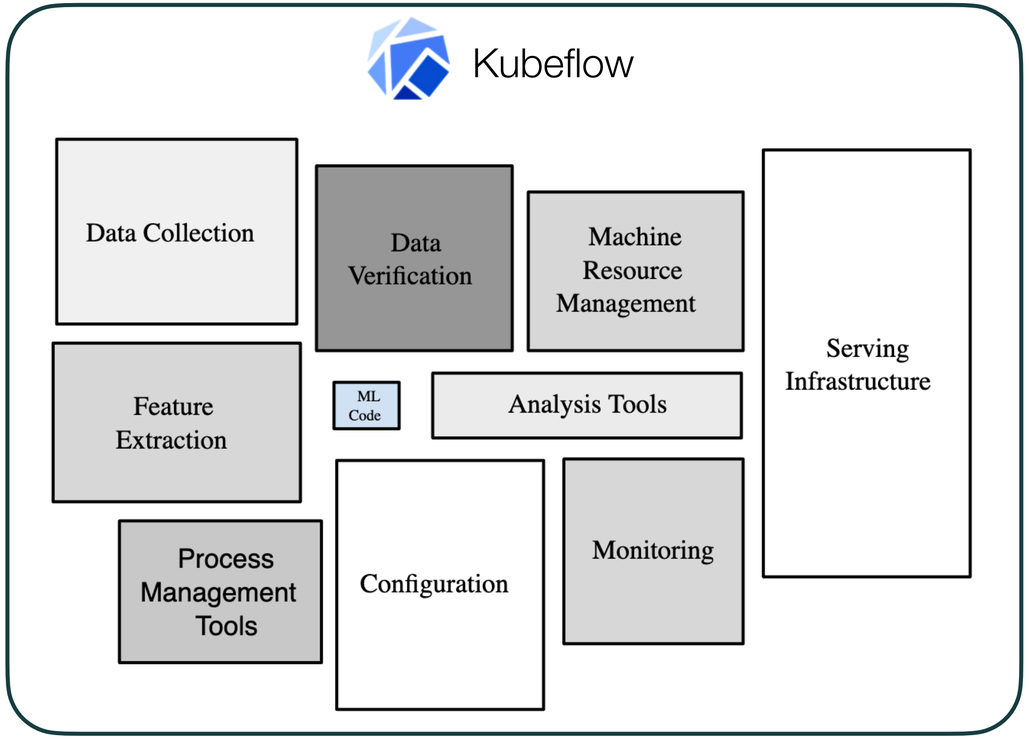
Implementation

- Layered architecture
- Expose GPUs from physical servers
- Use Openstack provided VMs
- Setup a Kubernetes cluster with Kubeflow



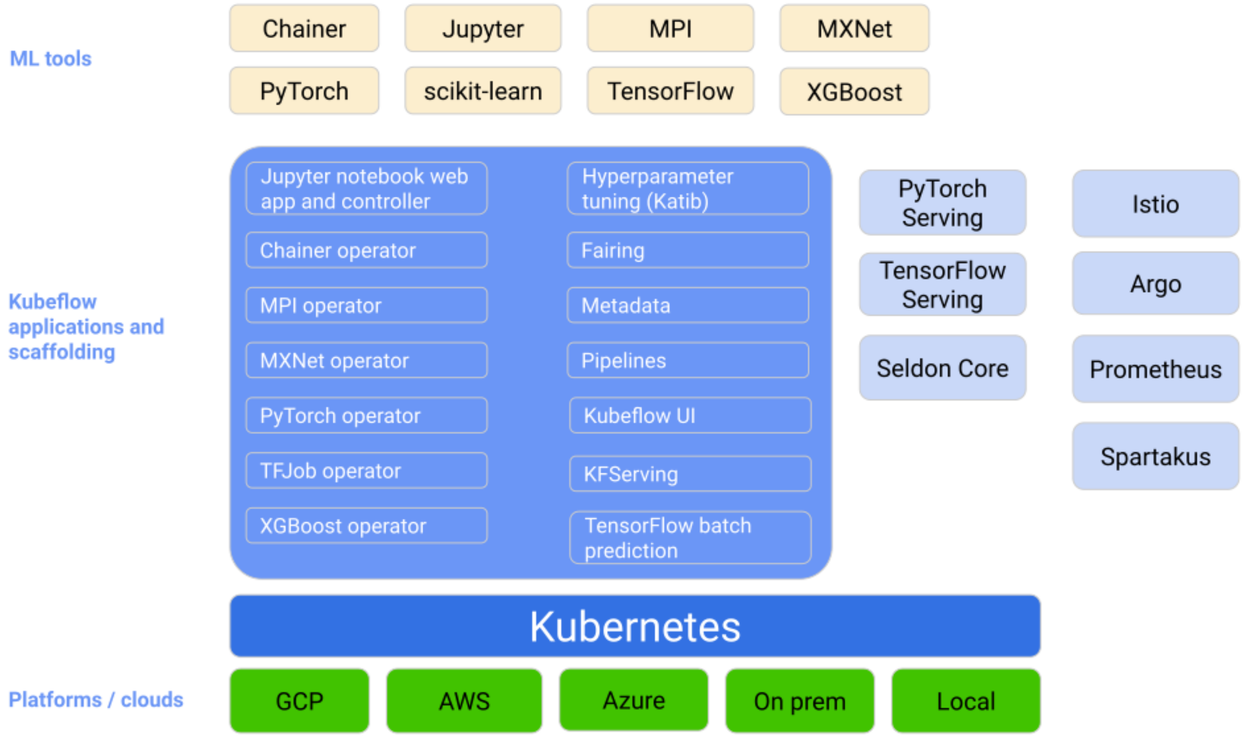
Kubeflow - Machine Learning Toolkit for Kubernetes

- ML deployments on Kubernetes made **simple, portable and scalable**
- Utilise power of Kubernetes to run **ML jobs**
- **Web UI** to interact with components and features
- Support for the **entire lifecycle** of ML applications
  - **Training, inference, deployment**
- Open source, wide community support



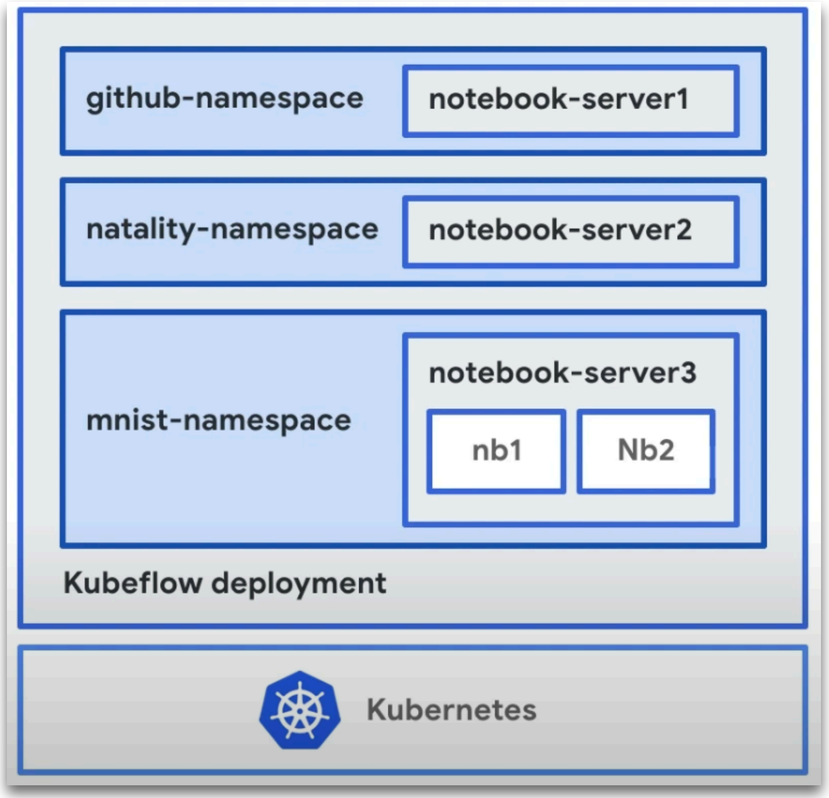
Kubeflow Components and Features

- Jupyter Notebooks
- Machine Learning Pipelines
- Katib - Hyper-parameter Optimization
- Distributed Training
- Model Serving
- Persistent Storage



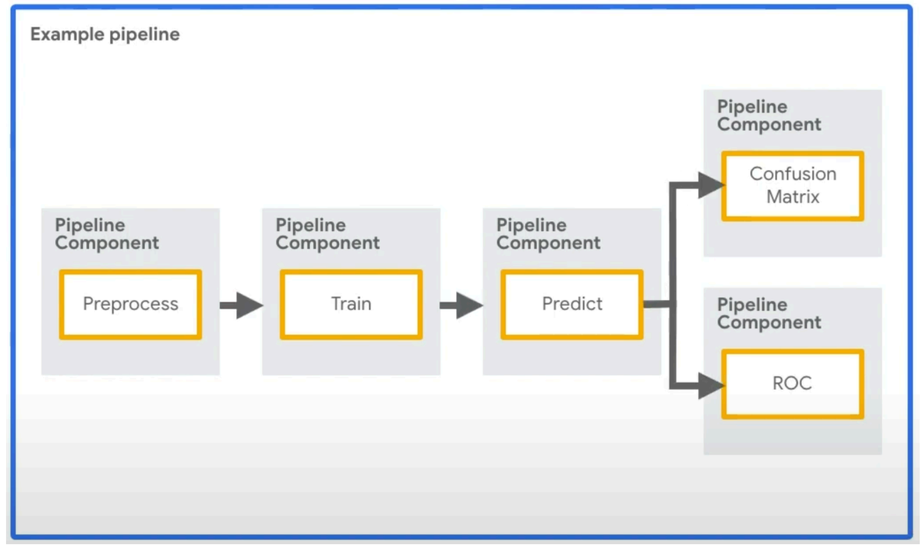
Jupyter Notebooks

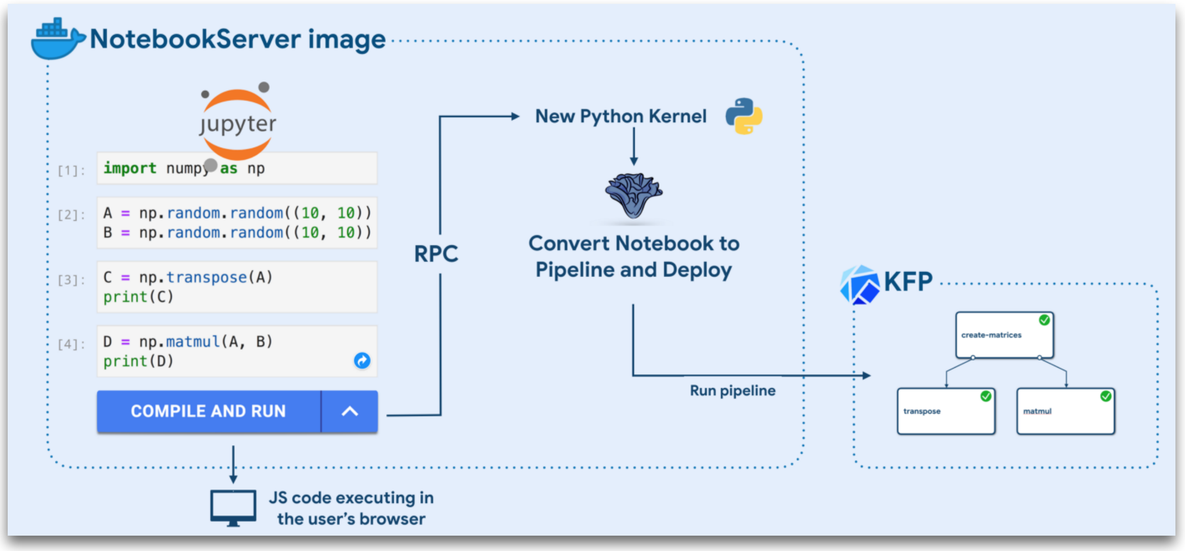
- Easiest way to start experimenting with Kubeflow
- Integration with other Kubeflow components
- Create a Notebook server **using existing images**
  - Select resources (CPU, MEM, GPU)
- Notebook server = JupyterLab environment with Terminal



Machine Learning Pipelines

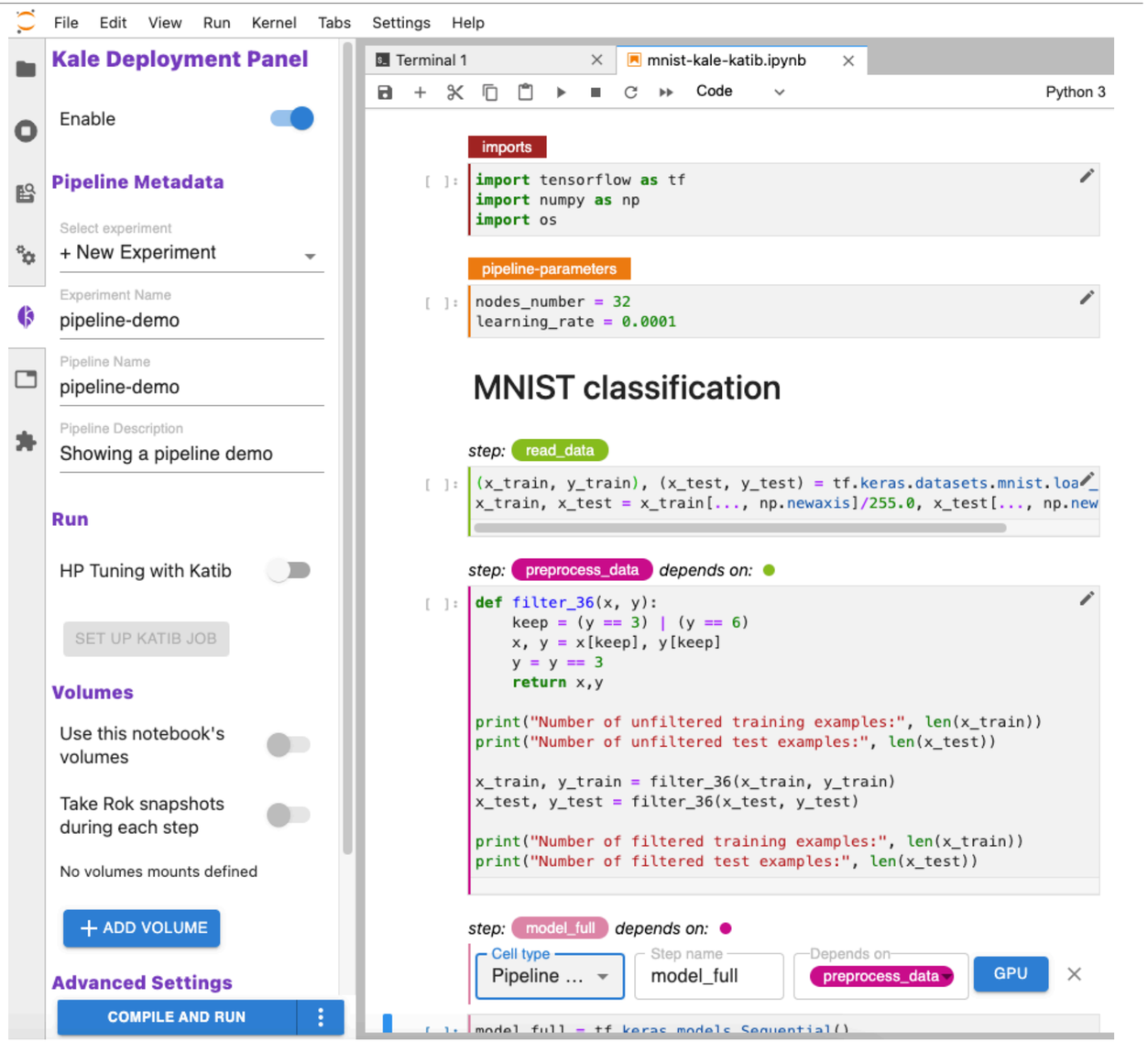
- A ***pipeline*** is a description of an **ML workflow**, including all components of a workflow in a form of a **graph**
- Kubeflow provides a **user interface** for managing and tracking experiments, jobs, and runs
- An **engine** for scheduling multi-step ML workflows
- An **SDK** for defining and manipulating pipelines and components





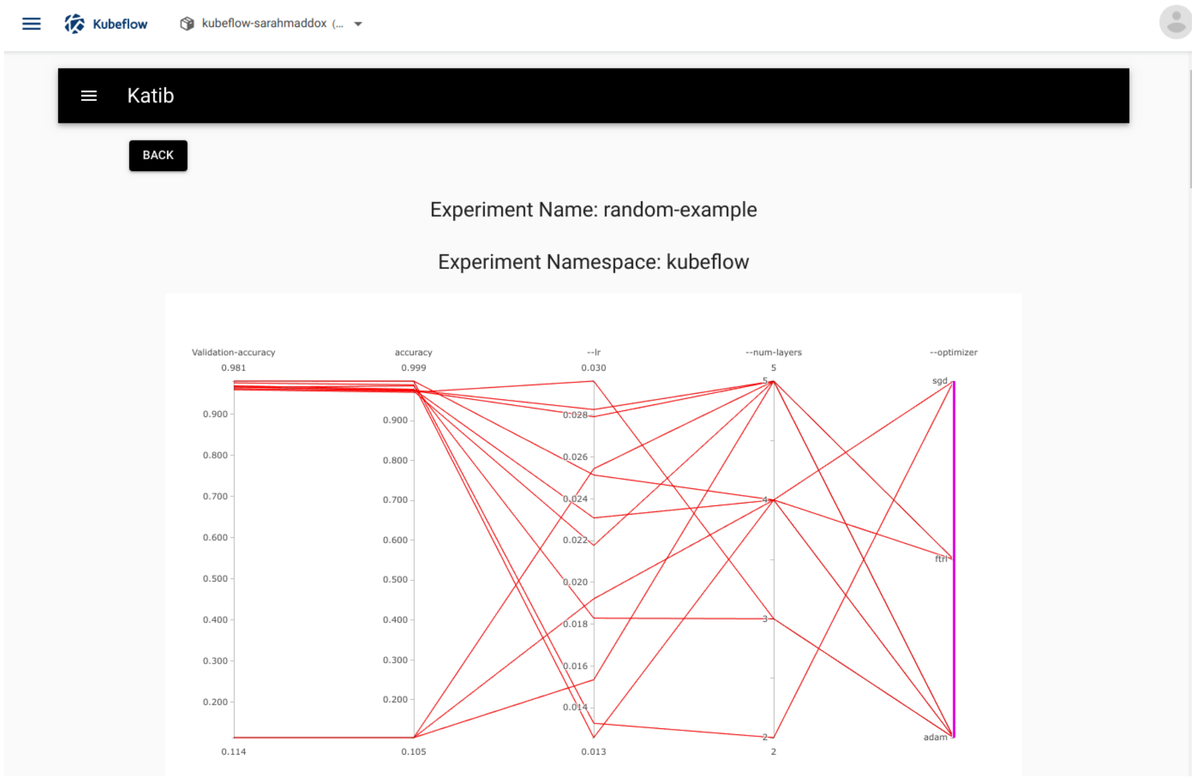
Notebooks to Pipelines - KALE

- Automated **conversion** notebooks to pipelines
- **Running** the converted pipelines, *in-place*
- No need to use Kubeflow SDK for conversion to pipelines
- Provided as a UI **Jupyter Lab official extension**

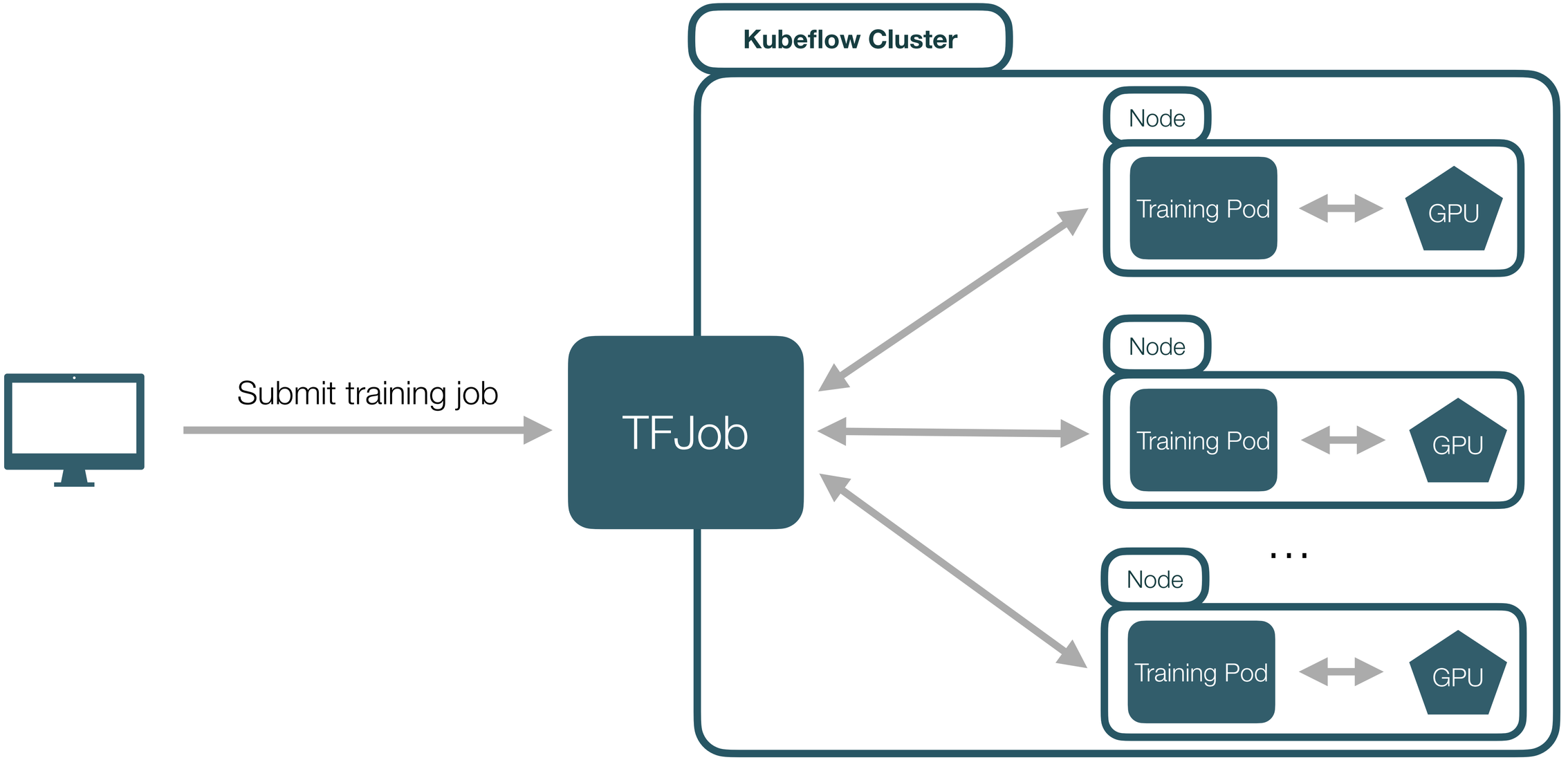


Katib - Hyper-parameter Optimization

- Finding the best set of non-trainable parameters of the models
- Usually takes a lot of effort when implemented by hand
- Made easier with pipelines
- Automated with Katib



Distributed GPU Training

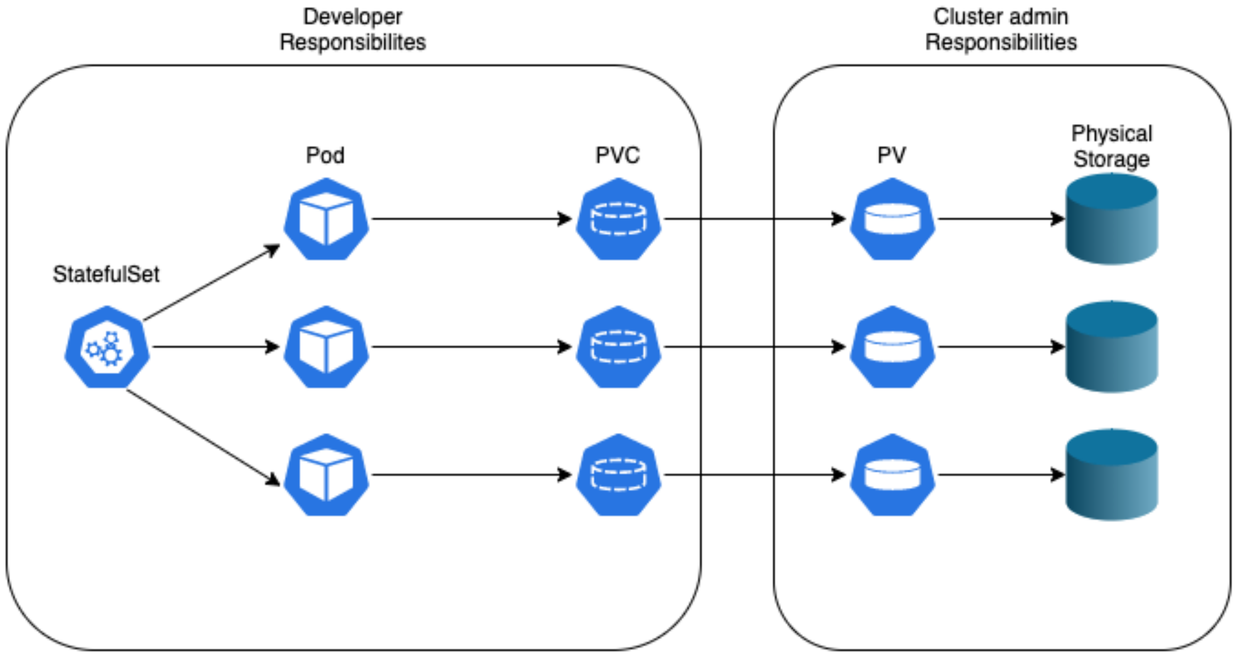


Model Serving

- Deploy a server as a **Kubernetes pod**, access server **endpoint via API**
  - *curl -v -H “Host: hostname” “http://host_ip/v1/models/mnist:predict" -d @./input.json*
- **Serverless** architecture
  - Automatic scaling per number of requests

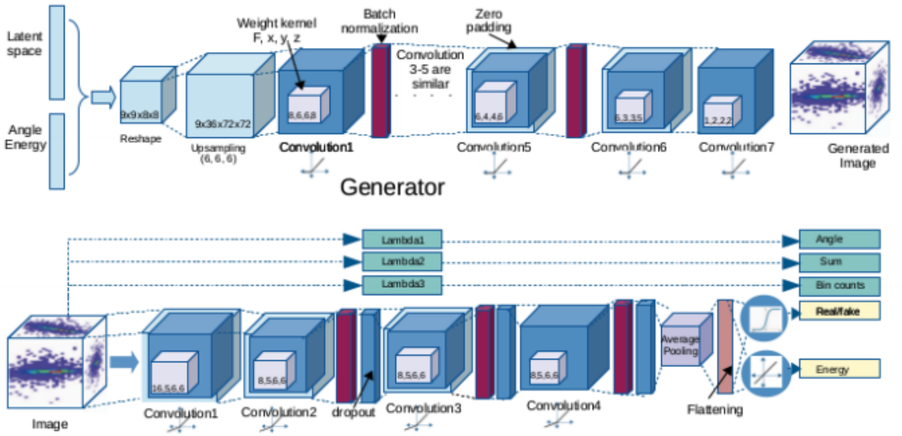
Persistent Storage

- Access to personal **EOS** folders
  - In notebooks
  - In pipelines (work in progress)
- **Object storage**
  - S3 in-cluster buckets - *minio* service
  - CERN s3.cern.ch centralized service
- Remote repositories - GitHub, GitLab



Sample Use Case - Training a 3DGAN

- 3D convolutional Generative Adversarial Network
  - Generate 51x51x25 pixels images
  - Represent energy depositions in calorimeters
  - Alternative to traditional Monte Carlo

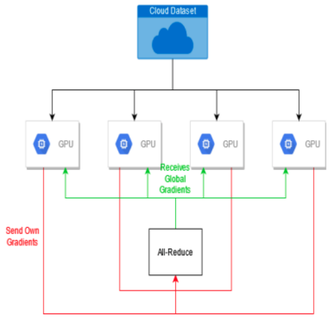


https://iopscience.iop.org/article/10.1088/1742-6596/1525/1/012064/pdf

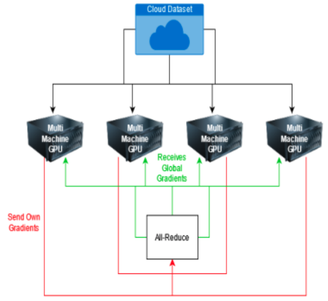
3DGAN Training Challenges

- Computationally extensive model
  - Full training of a single model using **one** GPU: ~2.5 days
- Solution - distributed training
  - Use TensorFlow distributed Strategy tf.distribute.Strategy

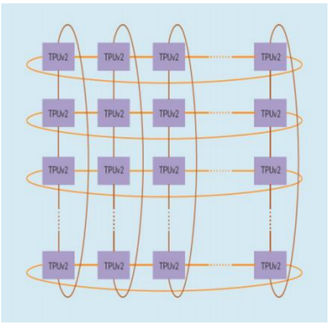
Mirrored Strategy



Multi Worker Mirrored Strategy



TPU Strategy

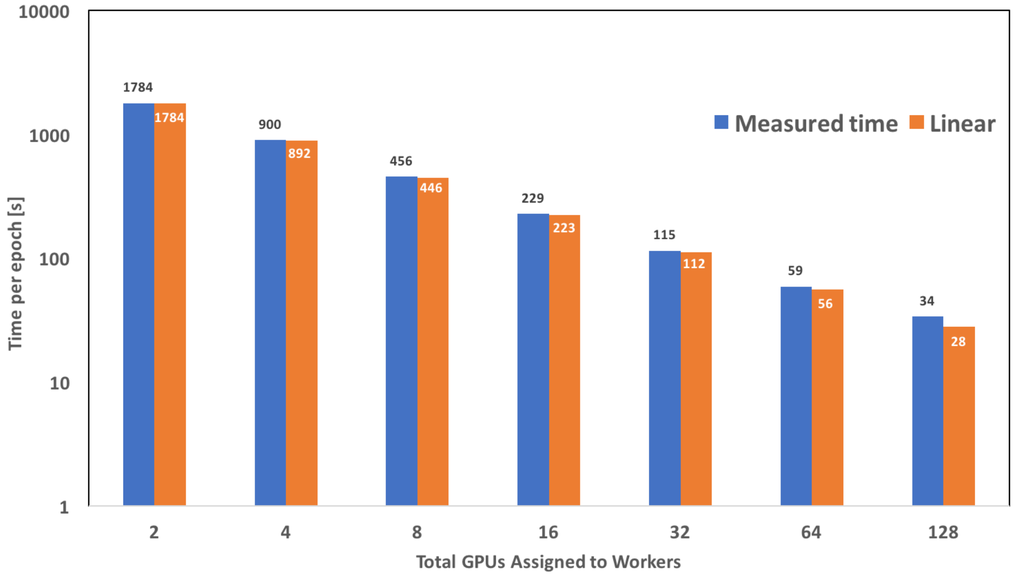


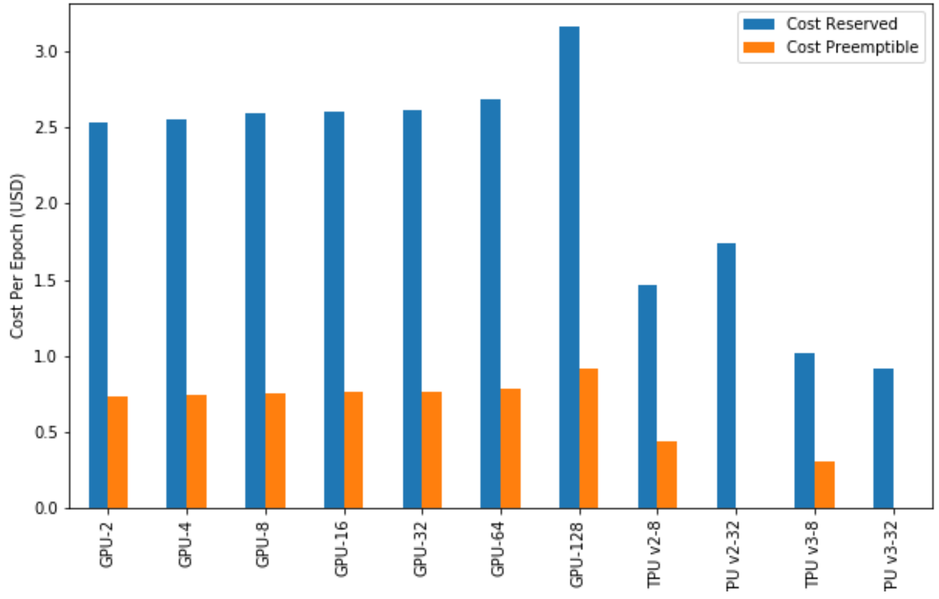
3DGAN Kubeflow Distributed Training

- Automate distributed training process
- Be able to quickly iterate over different training configurations
- Use TFJob
  - Test distributed training on a **local cluster** and on a **public cloud**
  - Rely on **128** (preemptible) Google Cloud **GPUs** for the distributed training
  - Kubeflow cluster running on GKE, deployed with same ArgoCD setup

3DGAN Kubeflow Distributed Training - Conclusions

- Performance improvement is **close to linear**
- Cost per epoch remains similar when increasing the number of GPUs
  - Training time reduced up to 52 times for 128 GPUs
- Best results achieved using preemptible **TPU v3-8**, 2.4 times cheaper than a GPU equivalent





Thank you for the attention!

Questions?