Centralized Machine Learning Service with Kubeflow

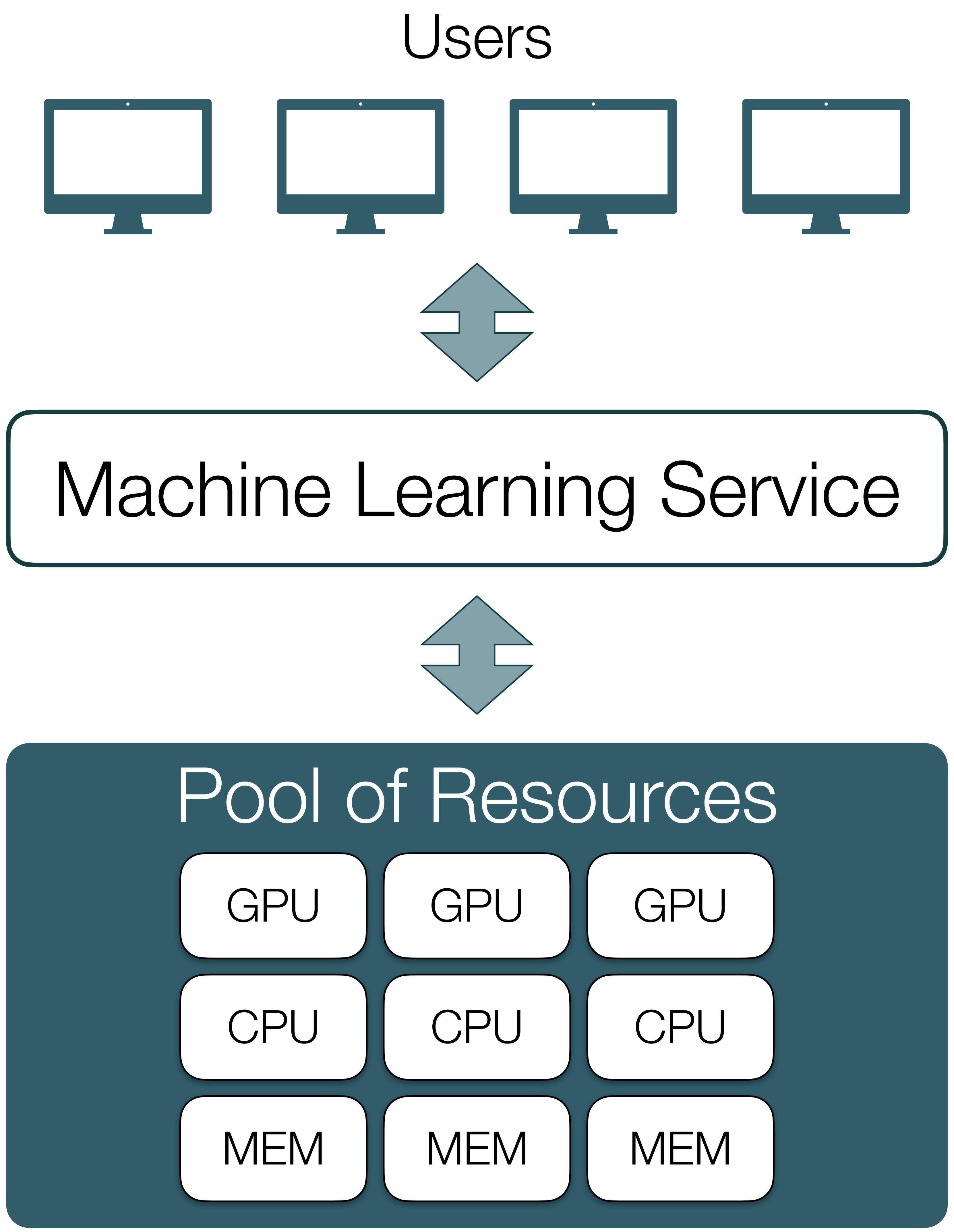
Dejan Golubovic, Ricardo Rocha

Outline

- Introduction
- Project motivation and roadmap
- Kubeflow - features, current status of development
- Demo
- Upcoming plans

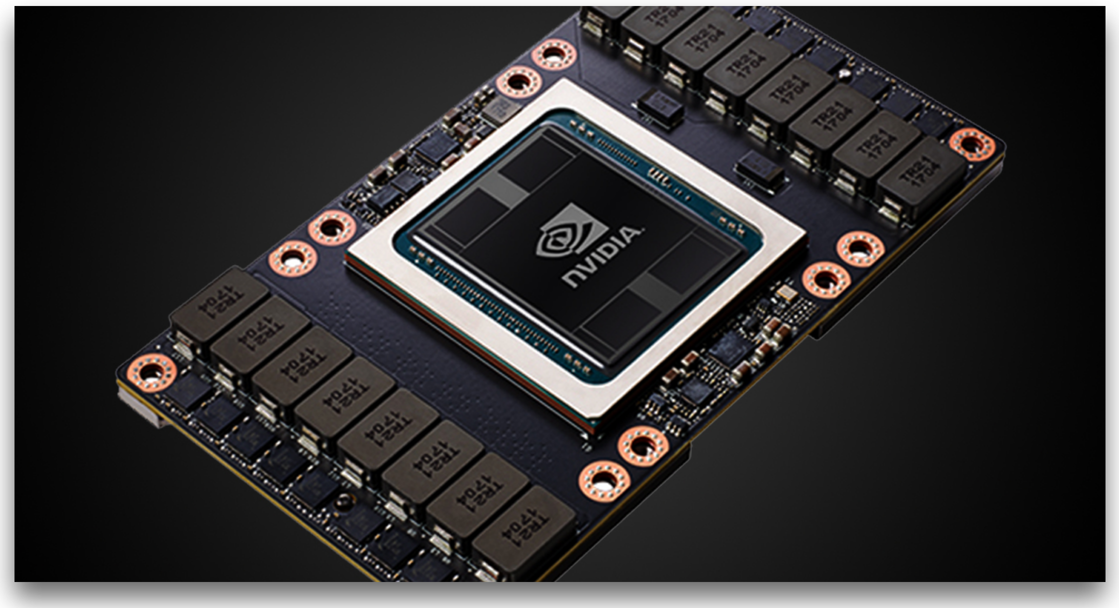
Project Motivation

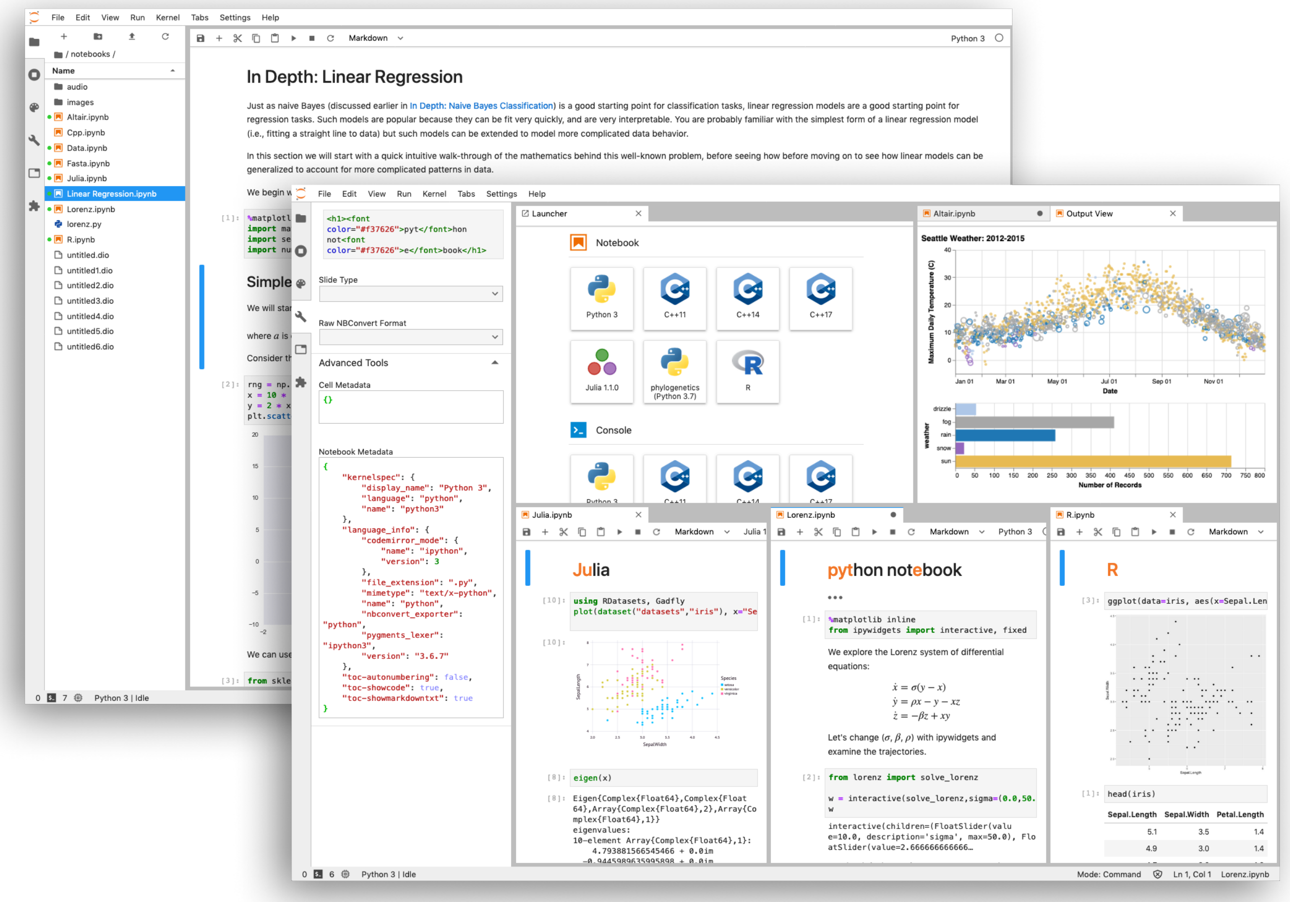
- Set-up a **centralized machine learning service**
- Offer variety of hardware resources to users
- Provide an easy-to-use web interface for ML tasks
- User advantages
  - No need to buy expensive hardware
  - Less time spent setting up infrastructure
  - More time for research

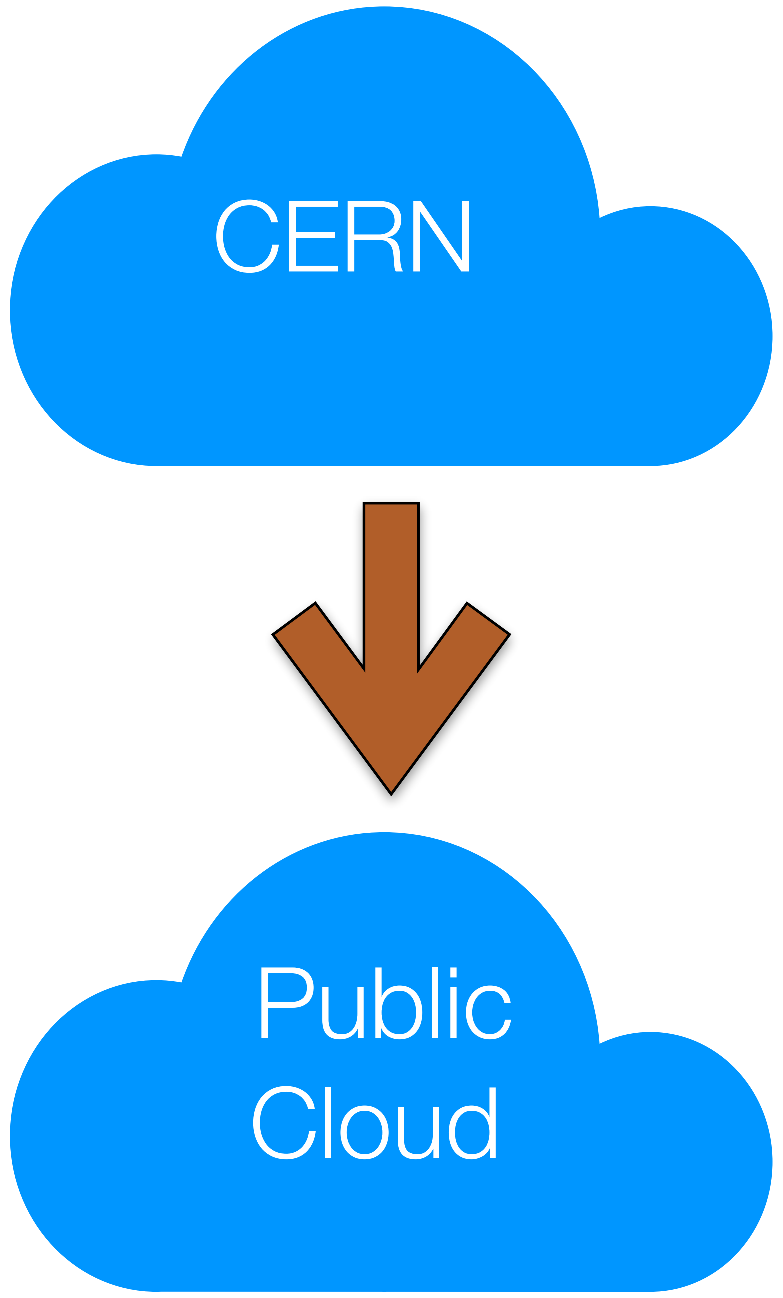


Idea

- Offer GPUs for efficient training
- User interface - notebooks, terminal, pipelines
- Scalability - possible migration to public clouds

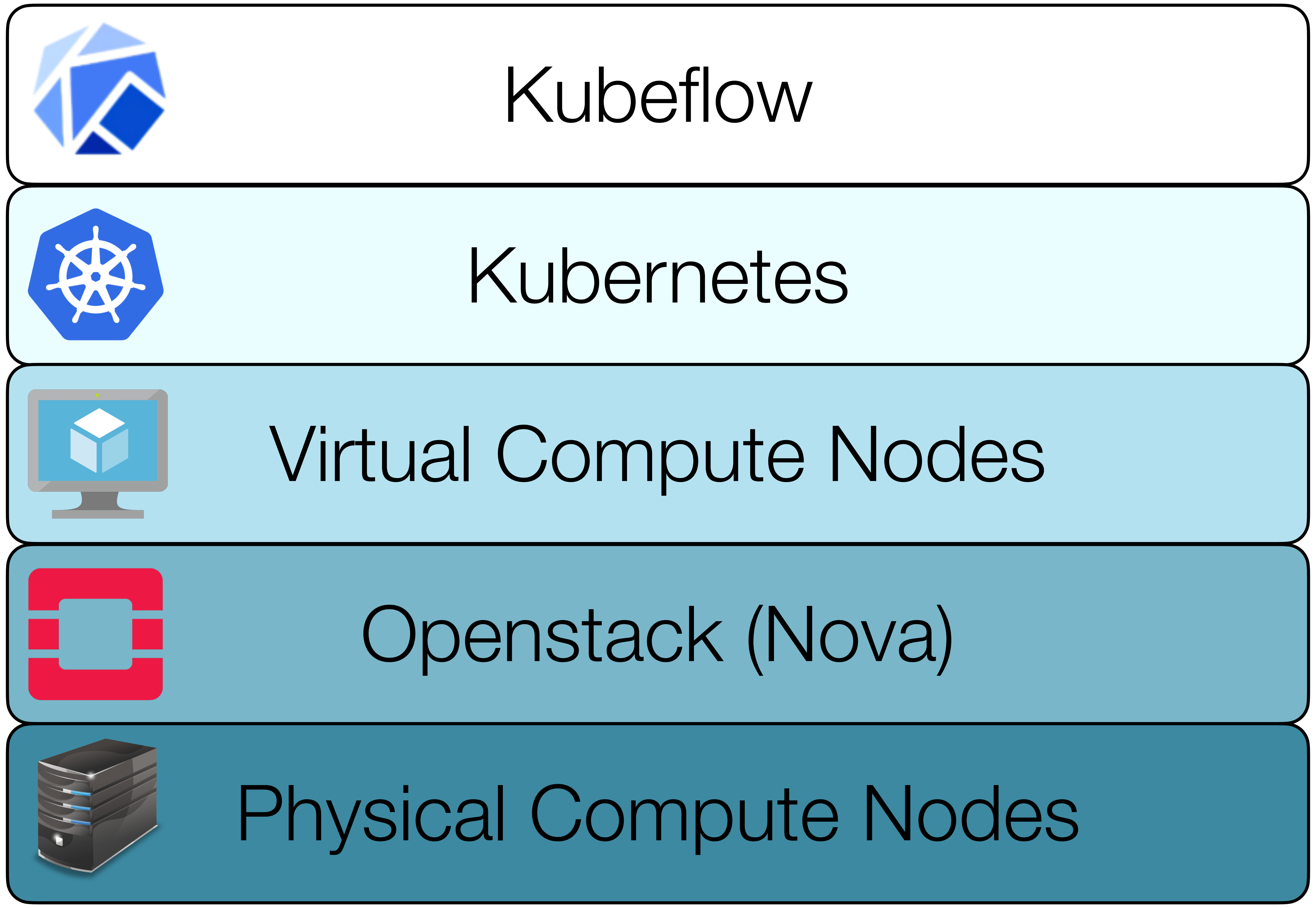




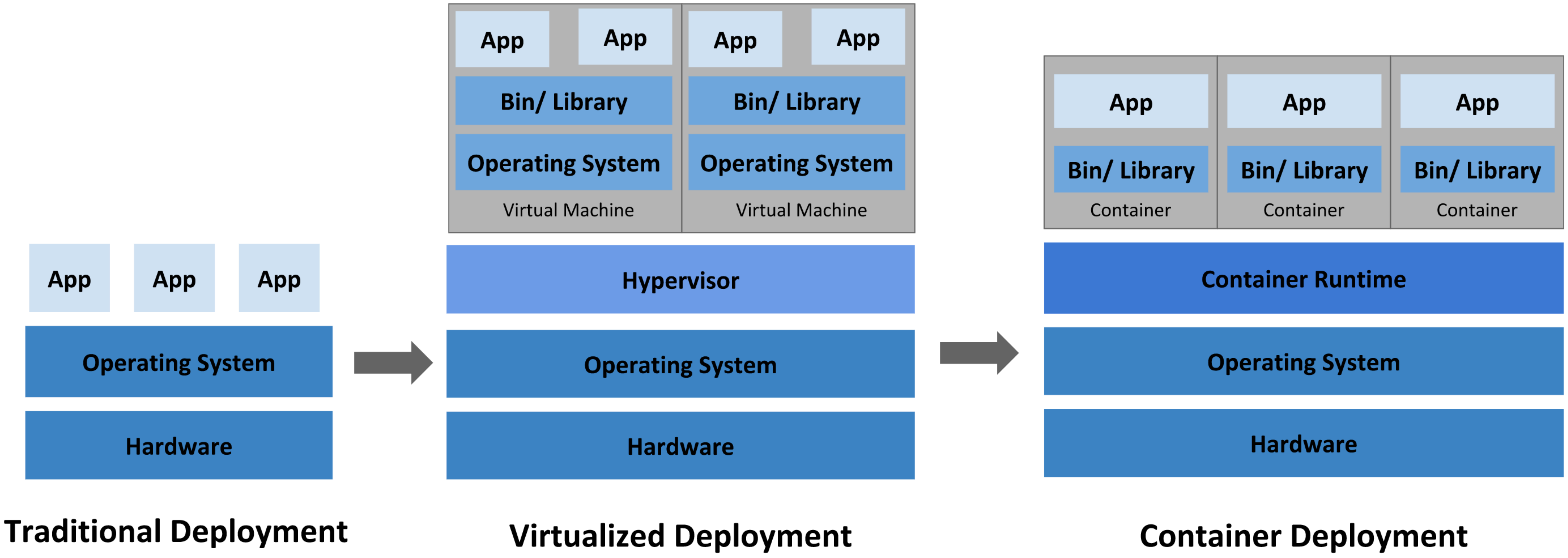


Implementation

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

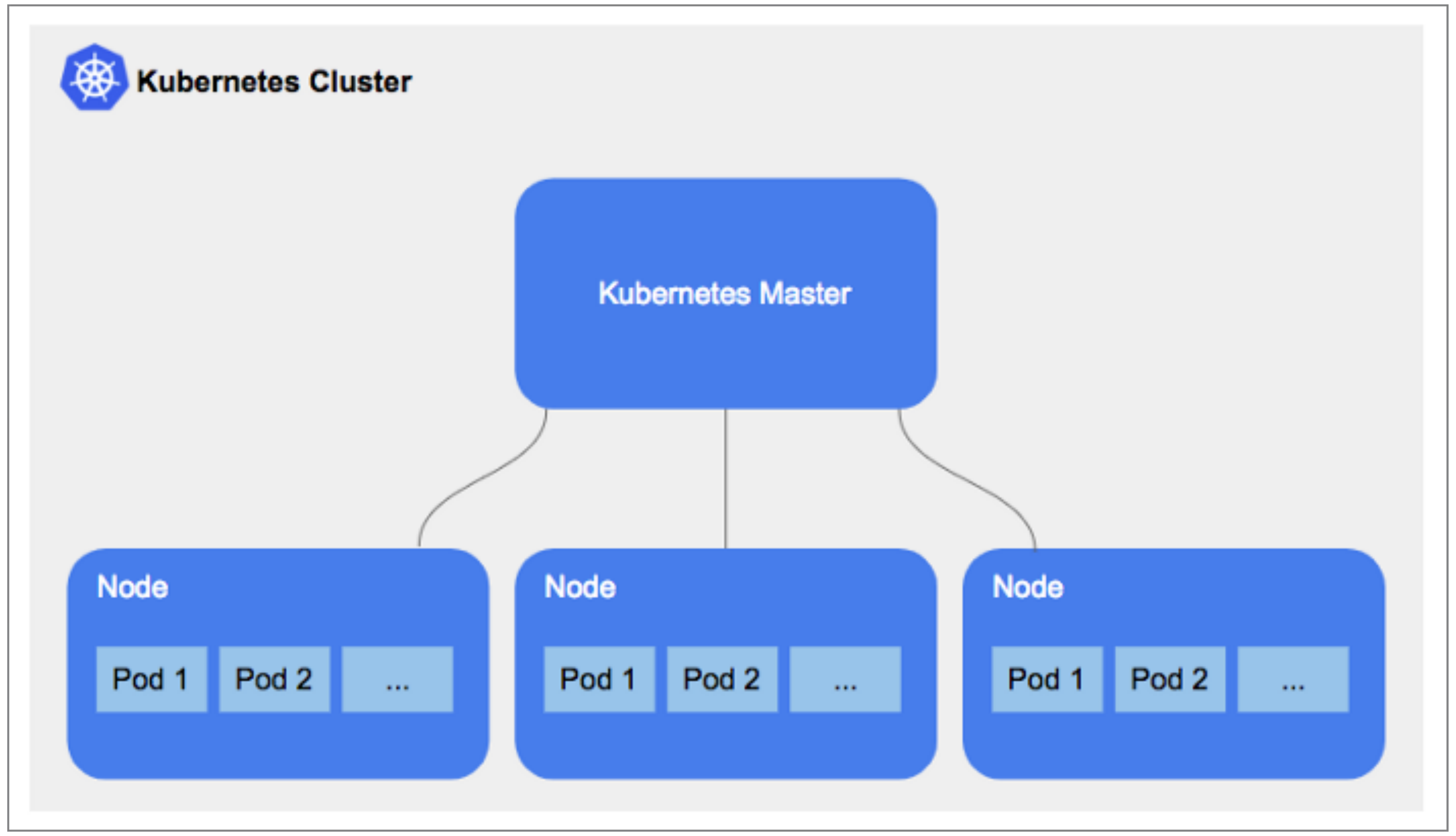


Container Evolution



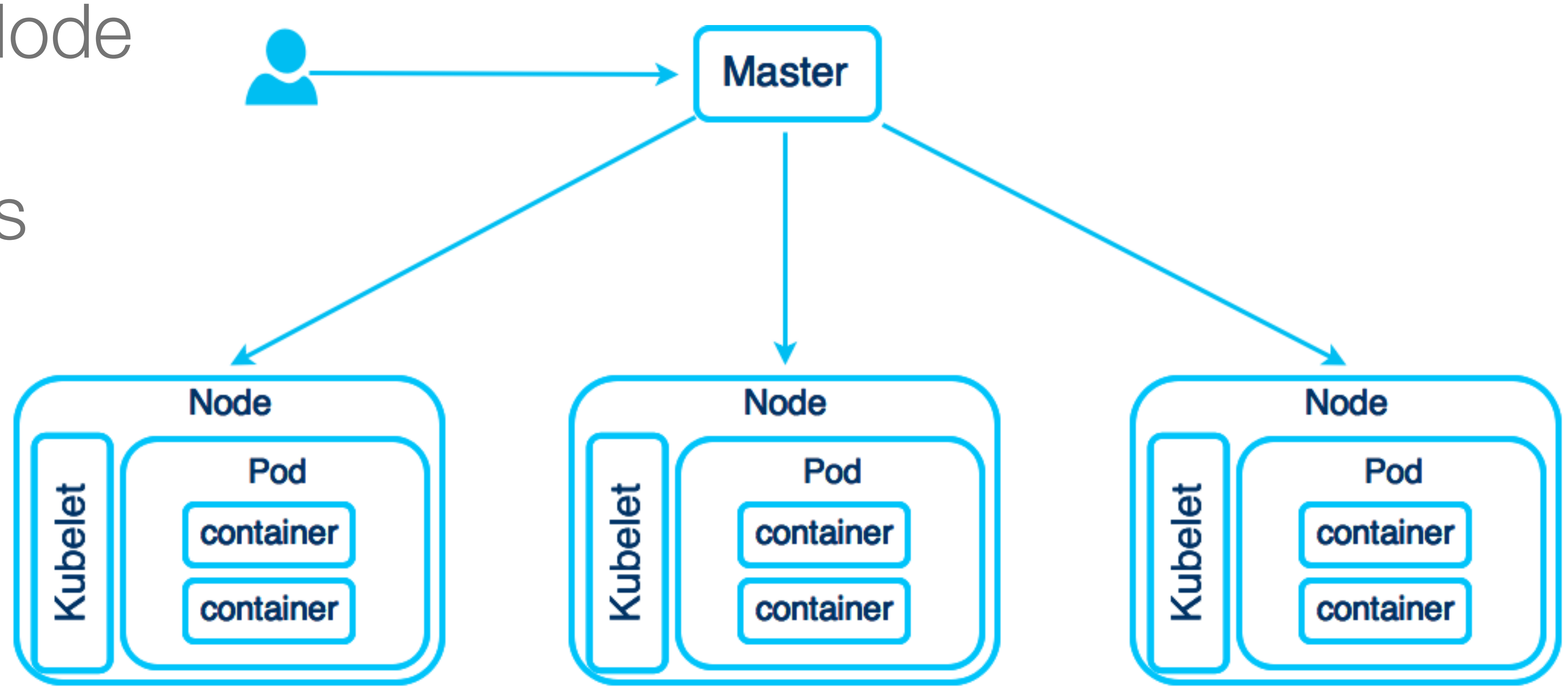
Why Kubernetes?

- **Manage containers in runtime environment**
- Restart containers automatically
- Schedule jobs
- Load balancing
- Storage orchestration
- Automated rollouts and rollbacks



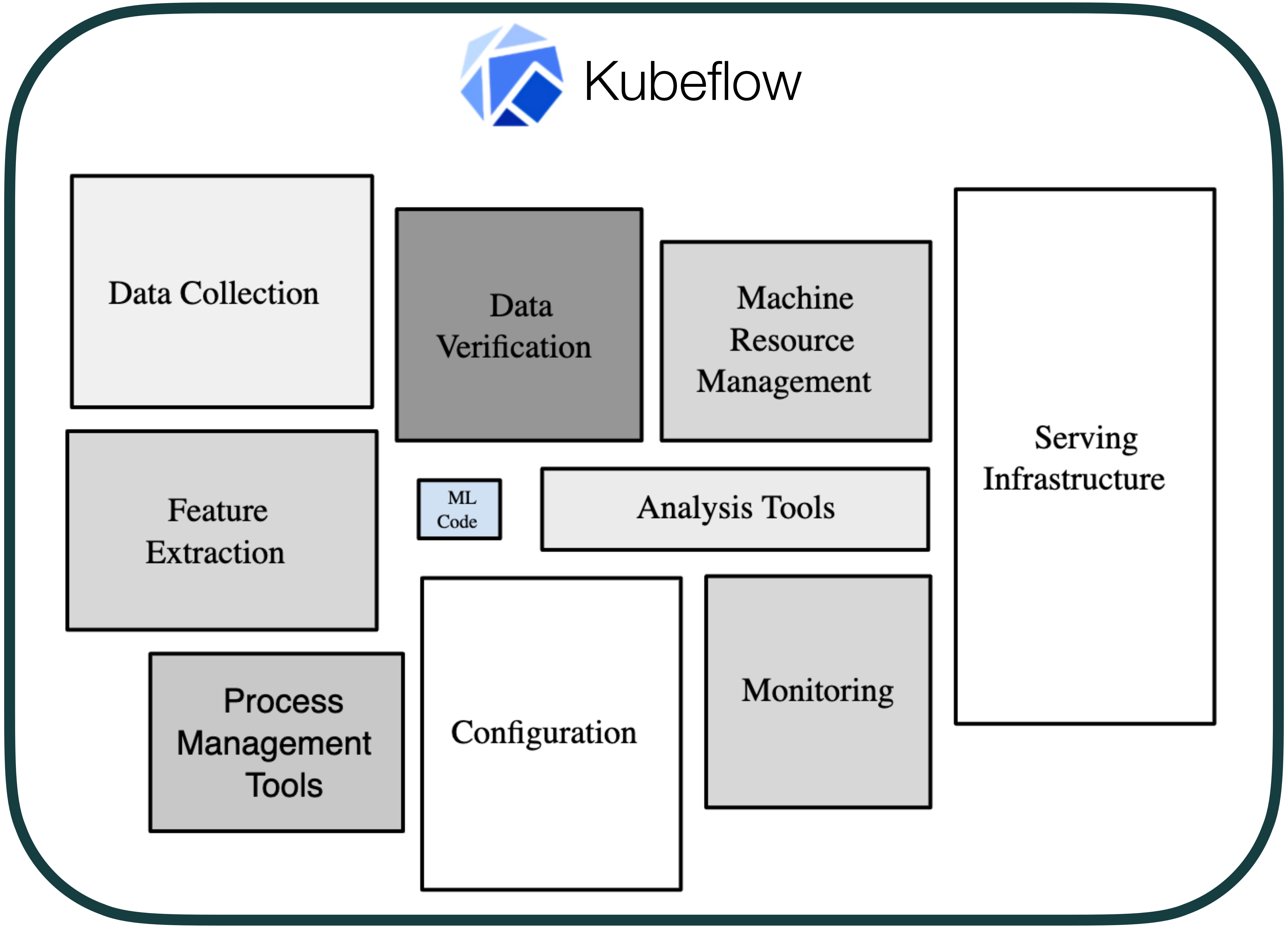
Kubernetes Architecture

- **Node** - physical or virtual machine where application code can be deployed
- **Master node** - a node which controls and manages a set of worker nodes
- **Kubelet** - primary "node agent" that runs on each node
- **Pod** - container wrapper that runs on a Node
- **Cluster** - bundle of Kubernetes resources

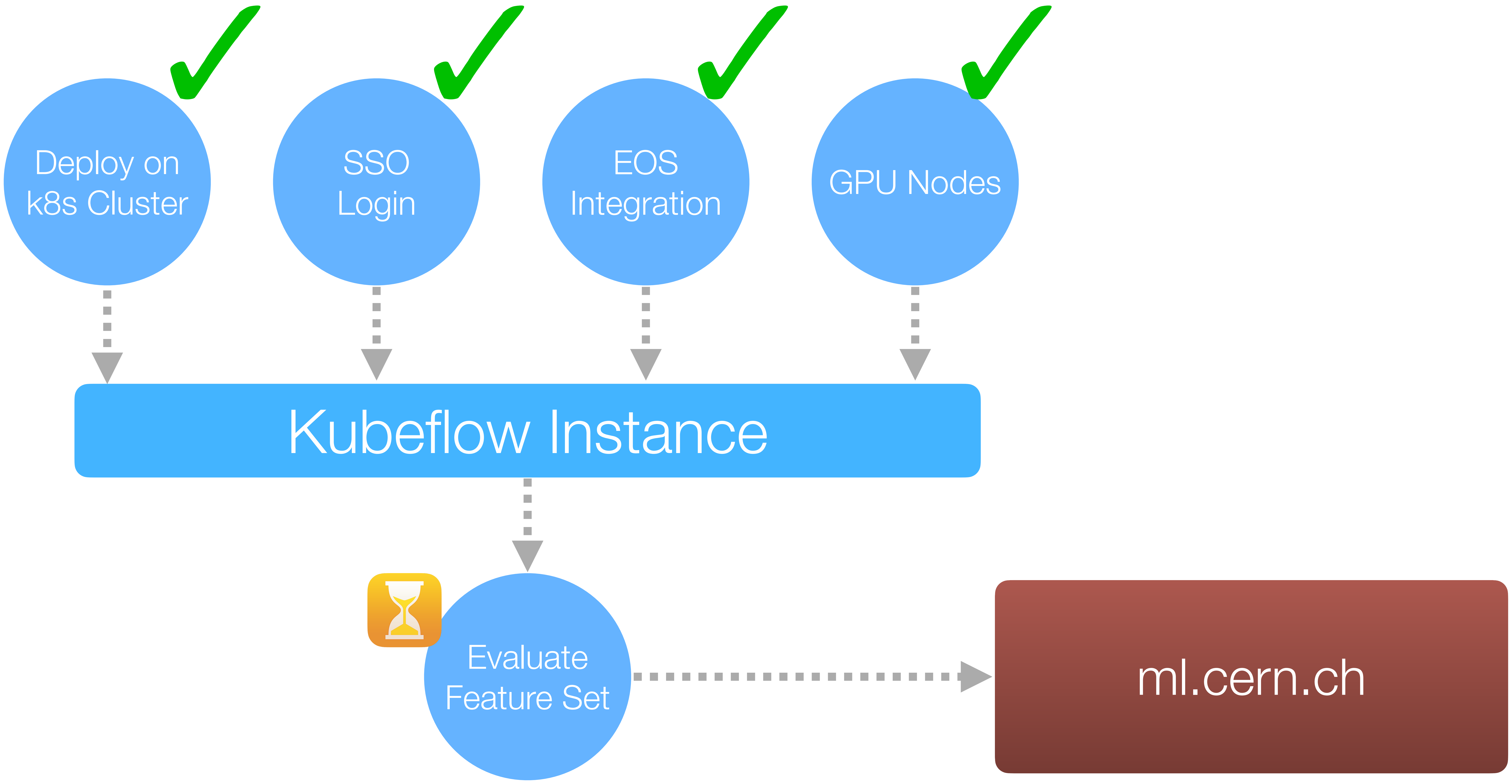


Kubeflow - Machine Learning Toolkit for Kubernetes

- ML deployments on Kubernetes made **simple, portable and scalable**
- Utilise power of Kubernetes to run **ML jobs**
- Manage ML infrastructure, platform and resource considerations
- Support for the **entire lifecycle** of ML applications
    - **Training, inference, deployment**
    - Development and production
- Open source, wide community support

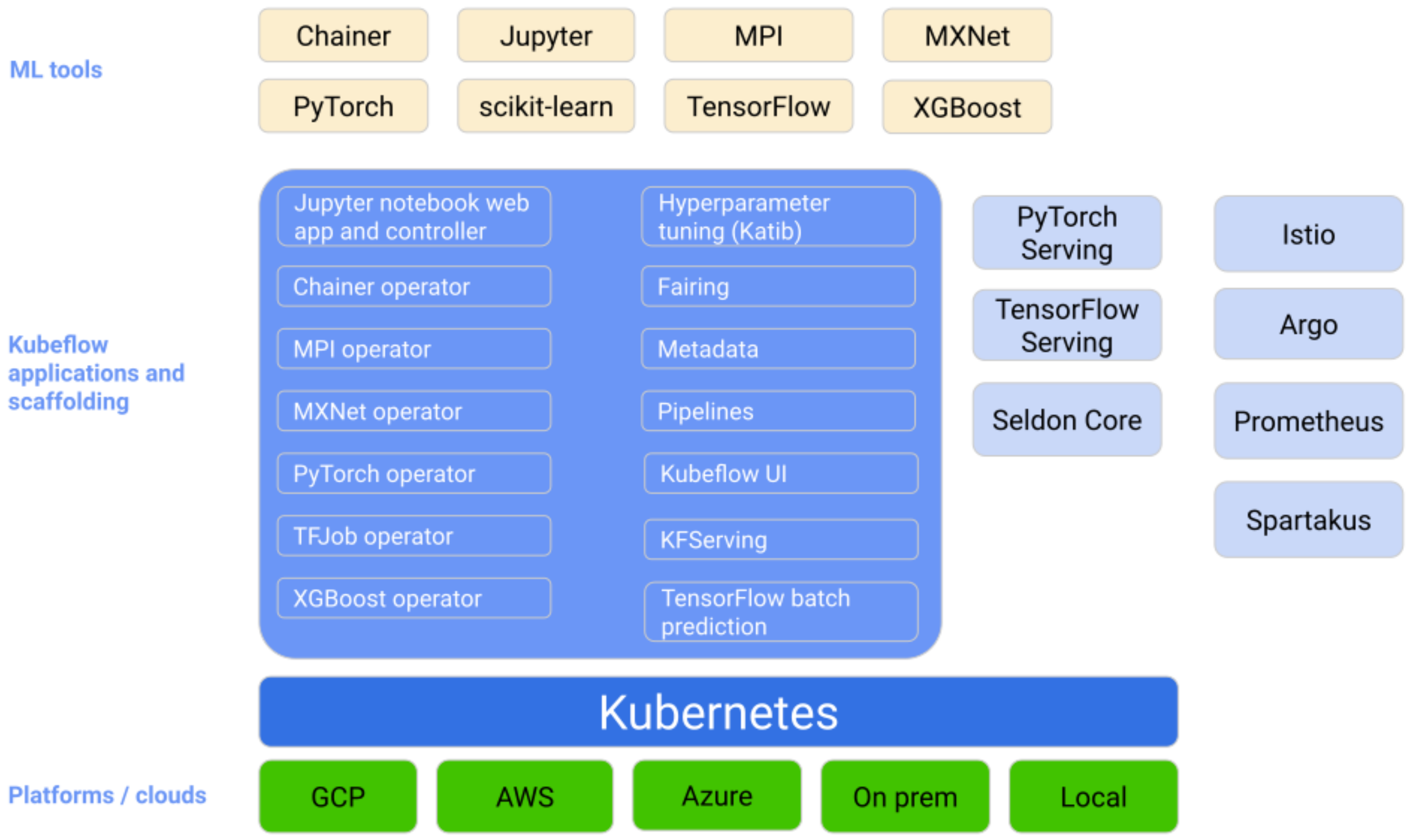

Kubeflow
Data Collection
Data Verification
Machine Resource Management
Serving Infrastructure
Feature Extraction
ML Code
Analysis Tools
Process Management Tools
Configuration
Monitoring

Project Roadmap



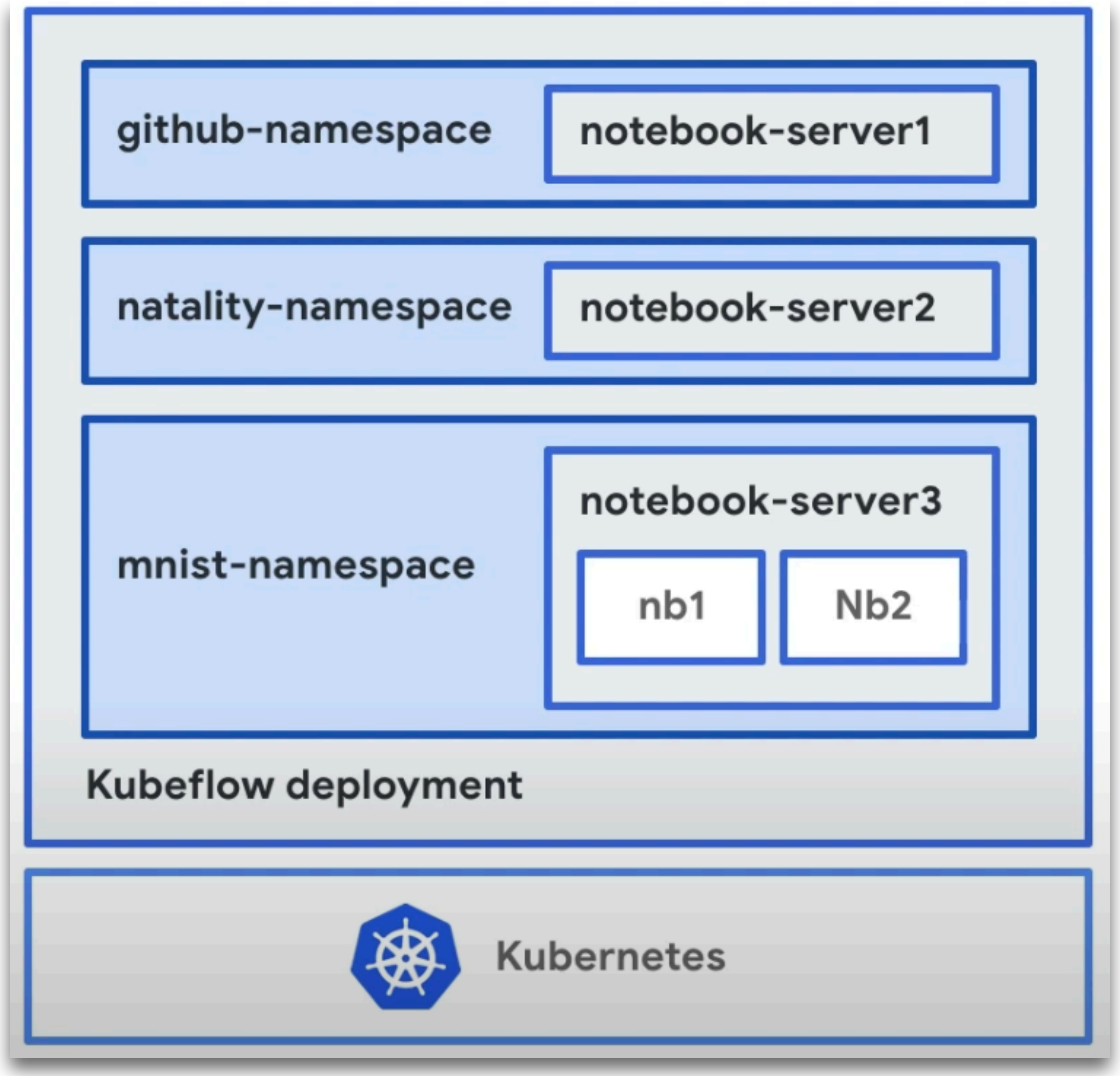
Kubeflow Components and Features

- Various Frameworks
  - Tensorflow, PyTorch, MPI
- Jupyter Notebooks
- Machine Learning Pipelines
- Katib - Hyper-parameter Optimization
- KALE - Notebooks to Pipelines or Katib
- Fairing - High level API



Jupyter Notebooks

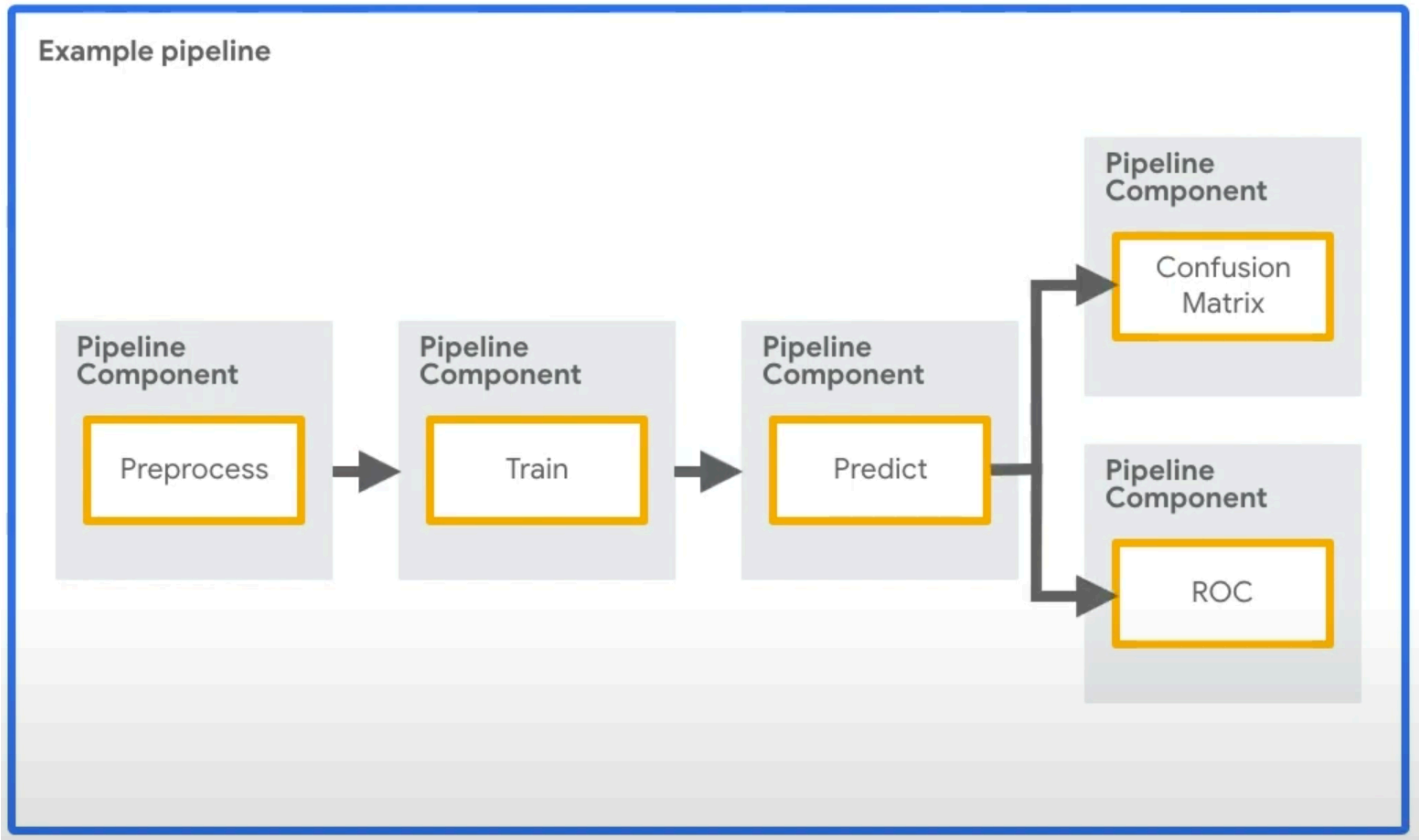
- Easiest way to start experimenting with Kubeflow
- Integration with other Kubeflow components
  - Access Kubeflow services in a cluster
- Create a Notebook server **using existing images**
  - Select resources (CPU, MEM, GPU)
- Create multiple Notebooks within one server



Machine Learning Pipelines

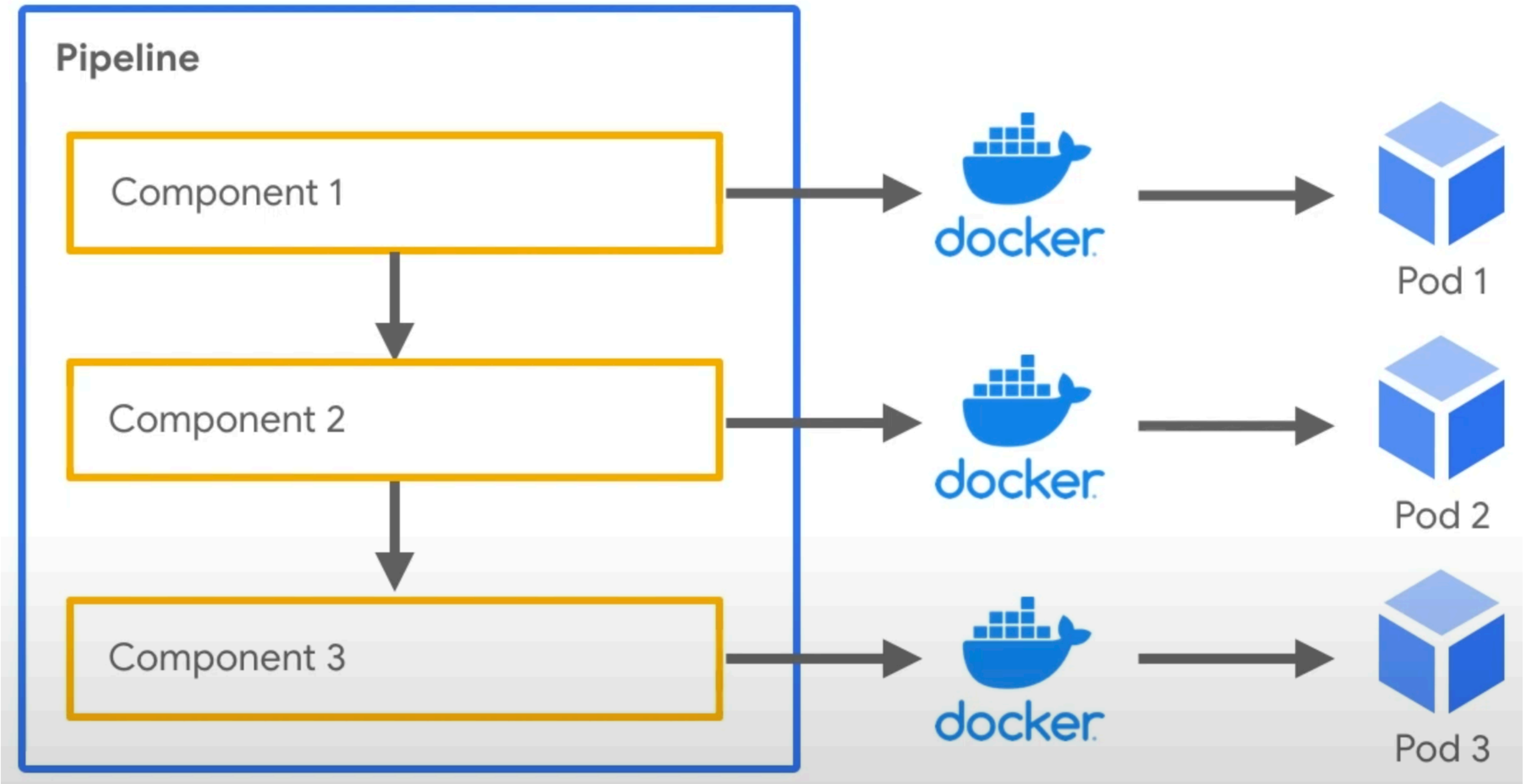
- A ***pipeline*** is a description of an **ML workflow**, including all components of a workflow and how they combine in a form of a **graph**

- A pipeline ***component*** is a self-contained set of user code, packaged as a **Docker image**, that performs **one step** in the pipeline



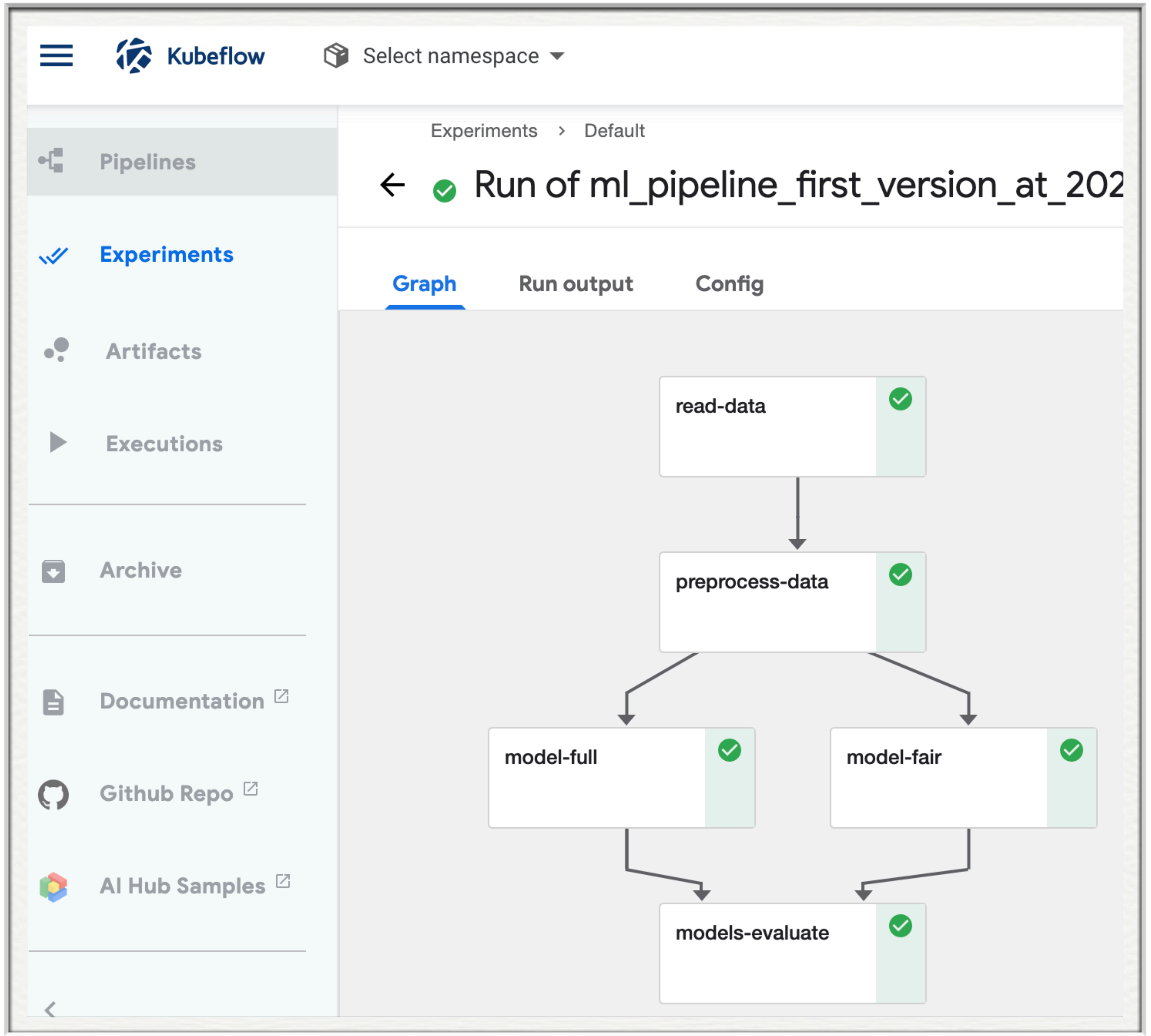
Machine Learning Pipelines

- A **user interface** (UI) 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
- Automatic scheduling of components, run in the specified order



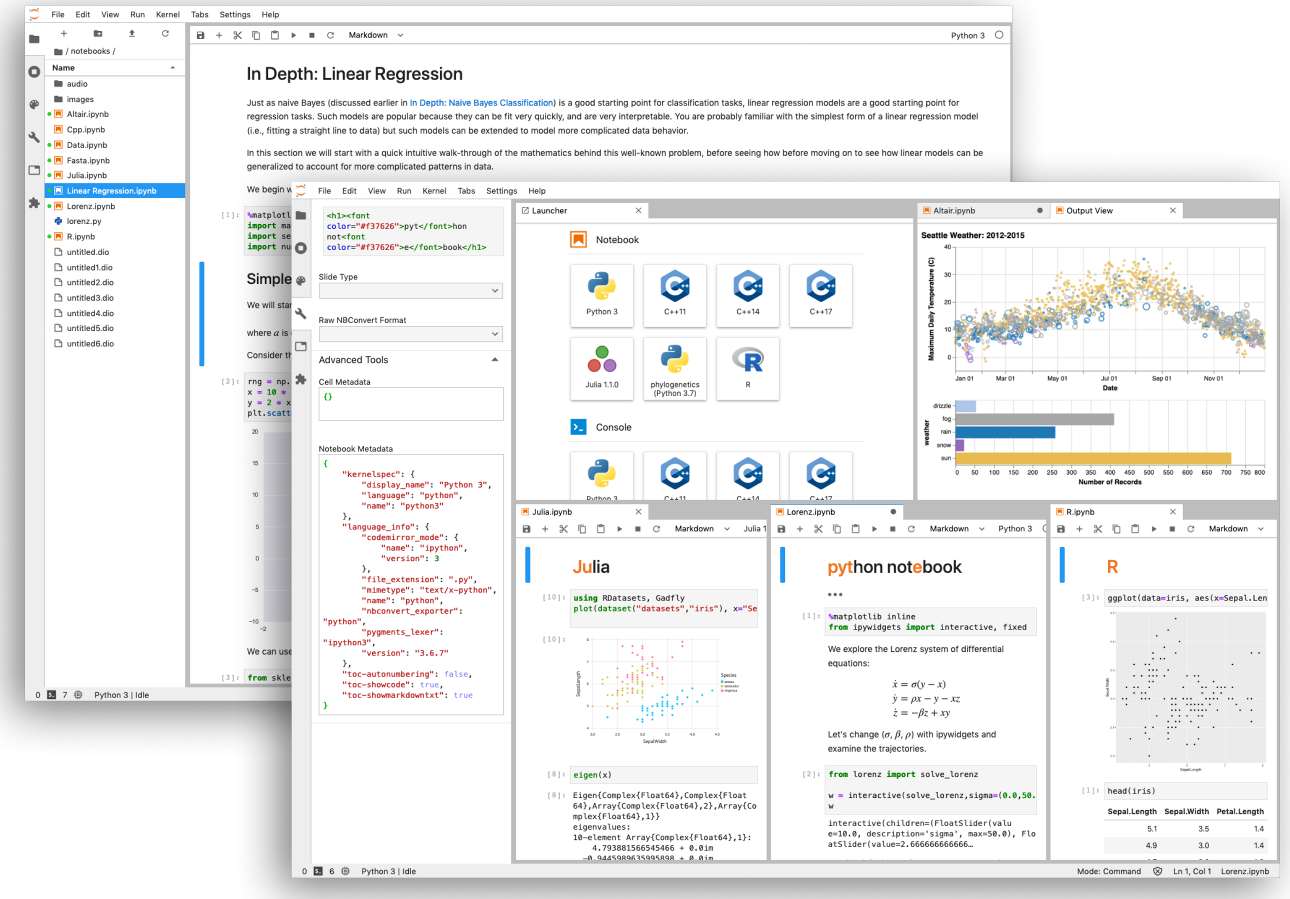
Benefits of Machine Learning Pipelines

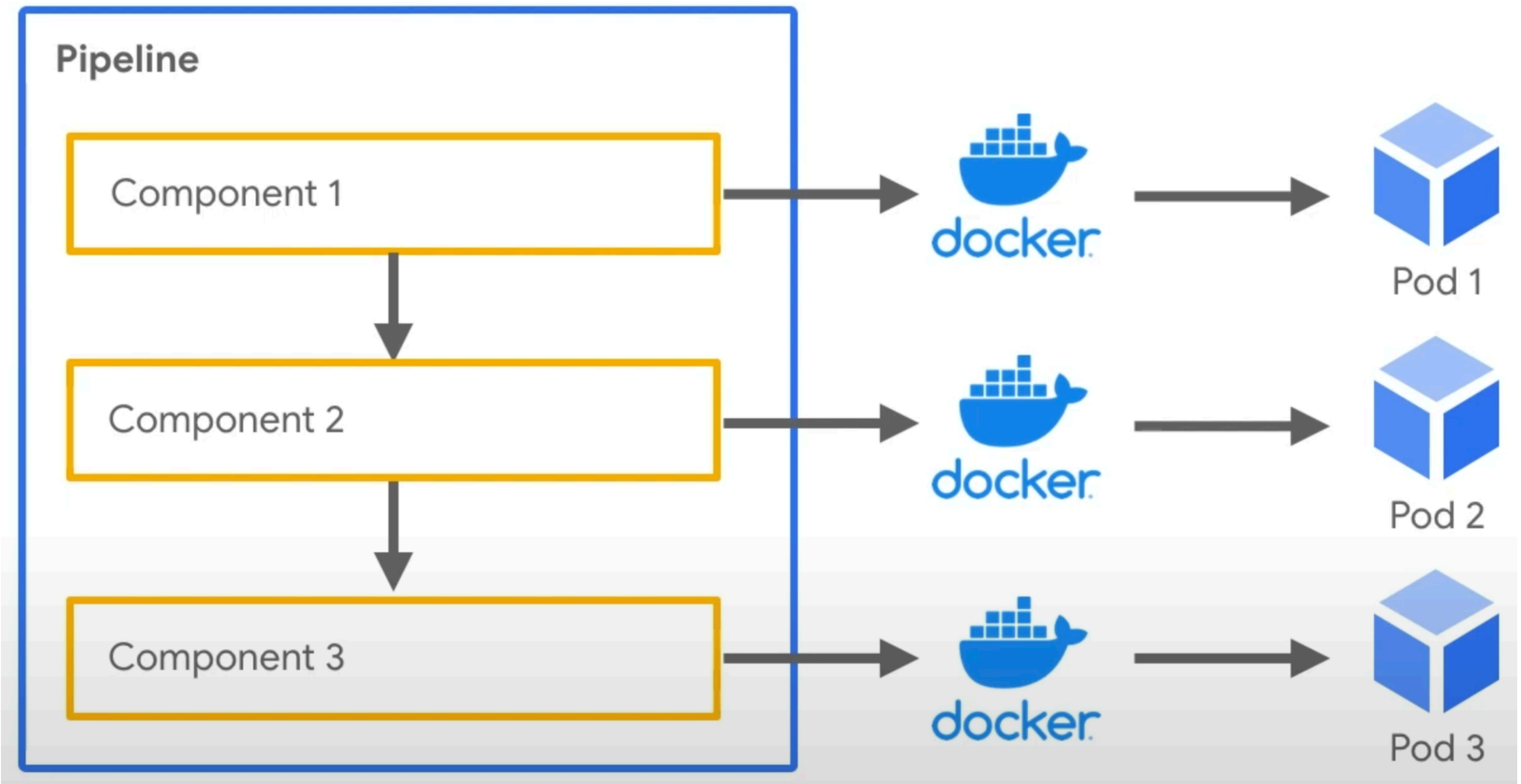
- Each step of the workflow clearly defined
- Components can be examined separately
- **Parallelisation**
- Versioning
- Non-blocking GPU access



Notebooks to Pipelines

- Kubeflow SDK, using ***kfp*** Python library
- KALE - Kubeflow Automated pipeLines Engine

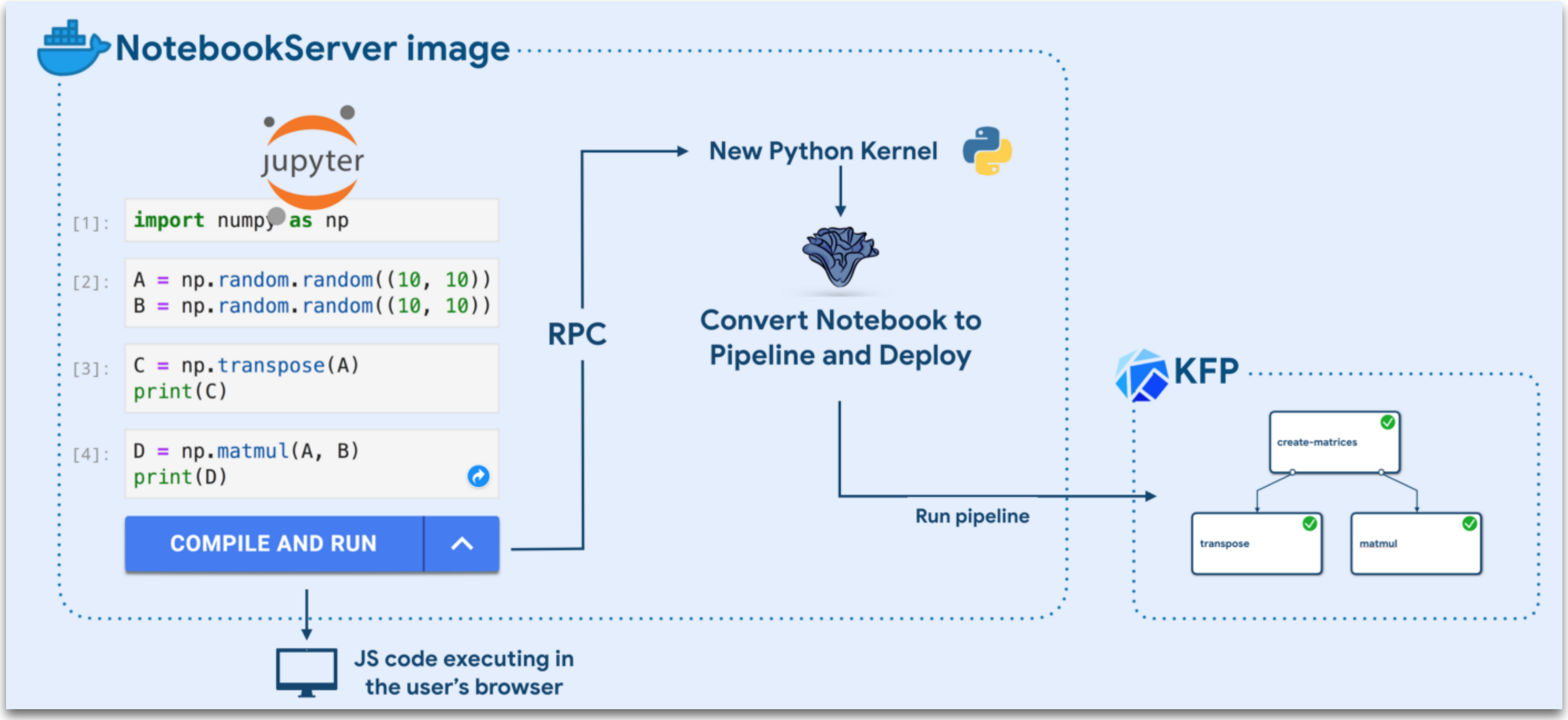




KALE - Kubeflow Automated pipeLines Engine

- 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**, part of a Docker image

KALE - Kubeflow Automated pipeLines Engine

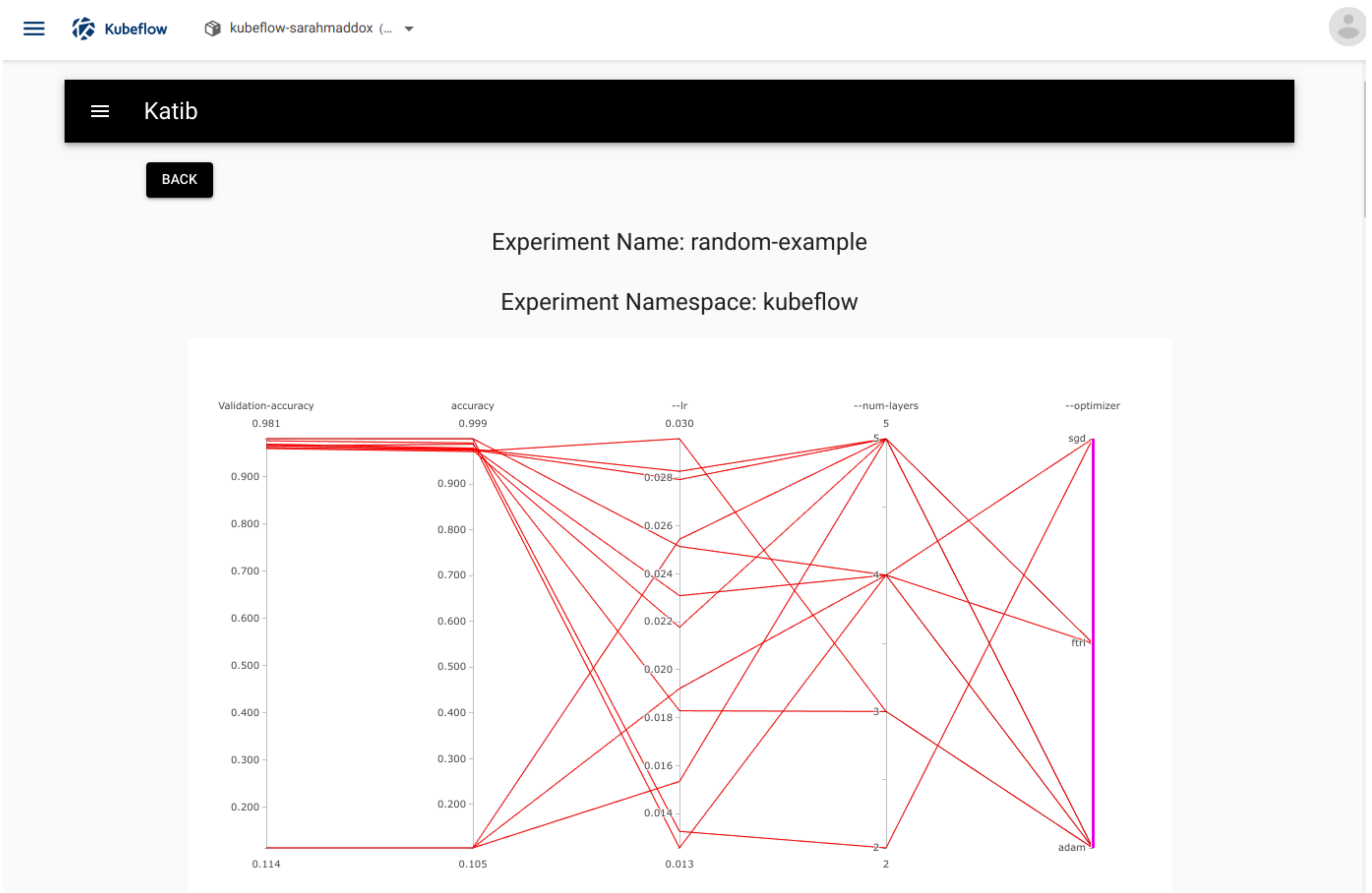


Katib

- Hyper-parameter Optimisation
- Neural Architecture Search

Katib - Hyper-parameter Optimisation

- 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, in terms of parallelisation
- Automated with Katib



TFJob - Tensorflow Distributed Training

- Split training jobs across multiple GPUs
- **TensorFlow** supports distributed training
  - Jobs are split across multiple **local GPUs**
  - https://www.tensorflow.org/guide/distributed_training
- **TFJob** - Kubernetes custom resource for distributed training
  - Jobs are split across multiple **cluster GPUs**
  - Combine TFJob with **TensorFlow** to parallelise model training
  - https://www.kubeflow.org/docs/components/training/tftraining/

TFJob Example

```yaml
apiVersion: kubeflow.org/v1
kind: TFJob
metadata:
  generateName: tfjob
  namespace: your-user-namespace
spec:
  tfReplicaSpecs:
    PS:
      replicas: 1
      restartPolicy: OnFailure
      template:
        metadata:
          annotations:
            sidecar.istio.io/inject: "false"
        spec:
          containers:
          - name: tensorflow
            image: gcr.io/your-project/your-image
            command:
              - python
              - -m
              - trainer.task
              - --batch_size=32
              - --training_steps=1000
```

```yaml
Worker:
  replicas: 3
  restartPolicy: OnFailure
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: "false"
    spec:
      containers:
      - name: tensorflow
        image: gcr.io/your-project/your-image
        command:
          - python
          - -m
          - trainer.task
          - --batch_size=32
          - --training_steps=1000
```

Kubeflow Fairing

- **Python package** for easier **training and deployment** of ML models
- Easily package ML training jobs
  - Using Kaniko, images can be built without Docker daemon
- Easily train ML models in a **hybrid cloud environment**, high level API
  - Run TFJobs from notebooks
  - Inspect the status of jobs, check logs
- Streamline the process of **deploying** a trained model
- Run jobs in **public cloud**

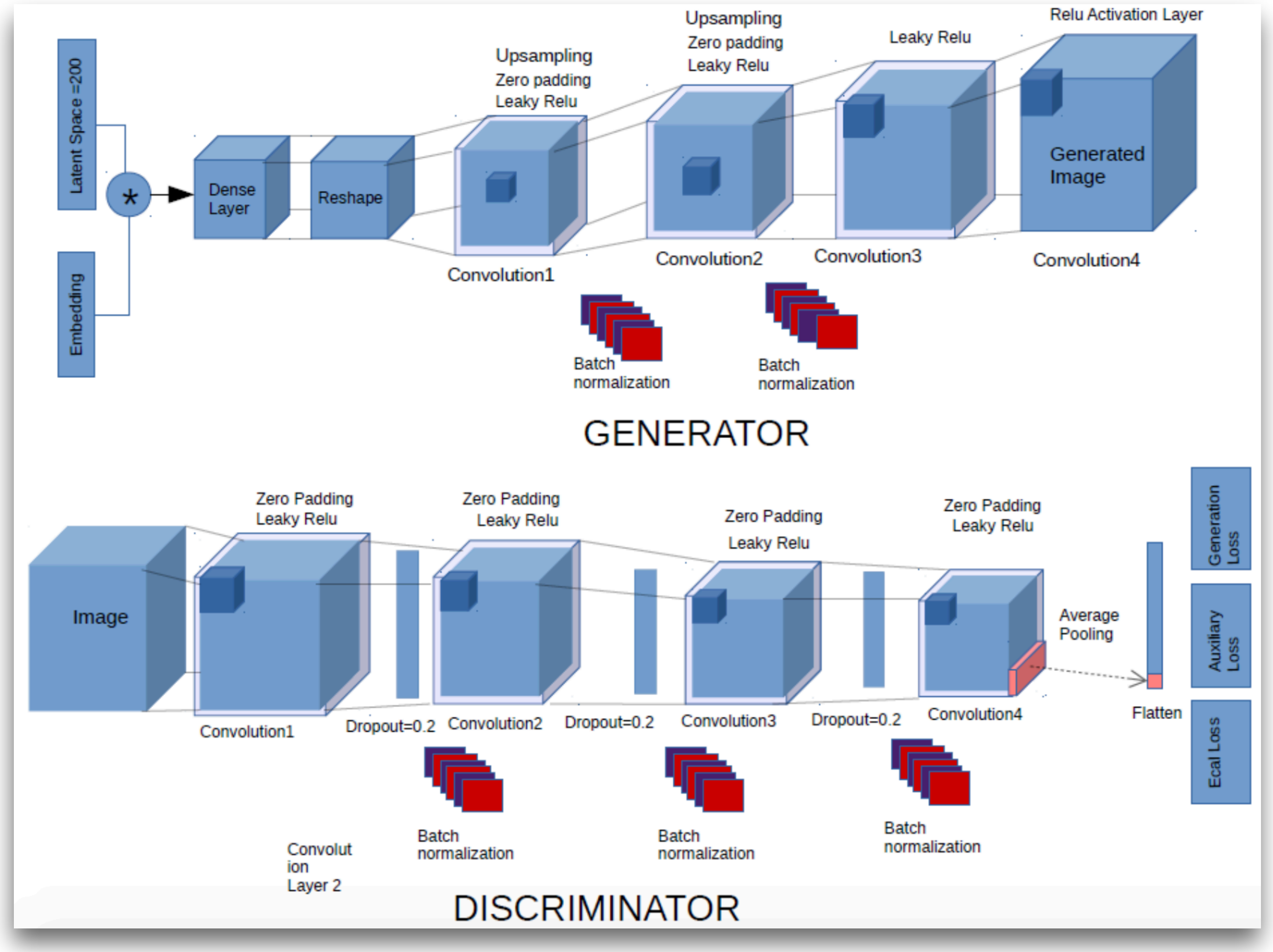
Model Serving

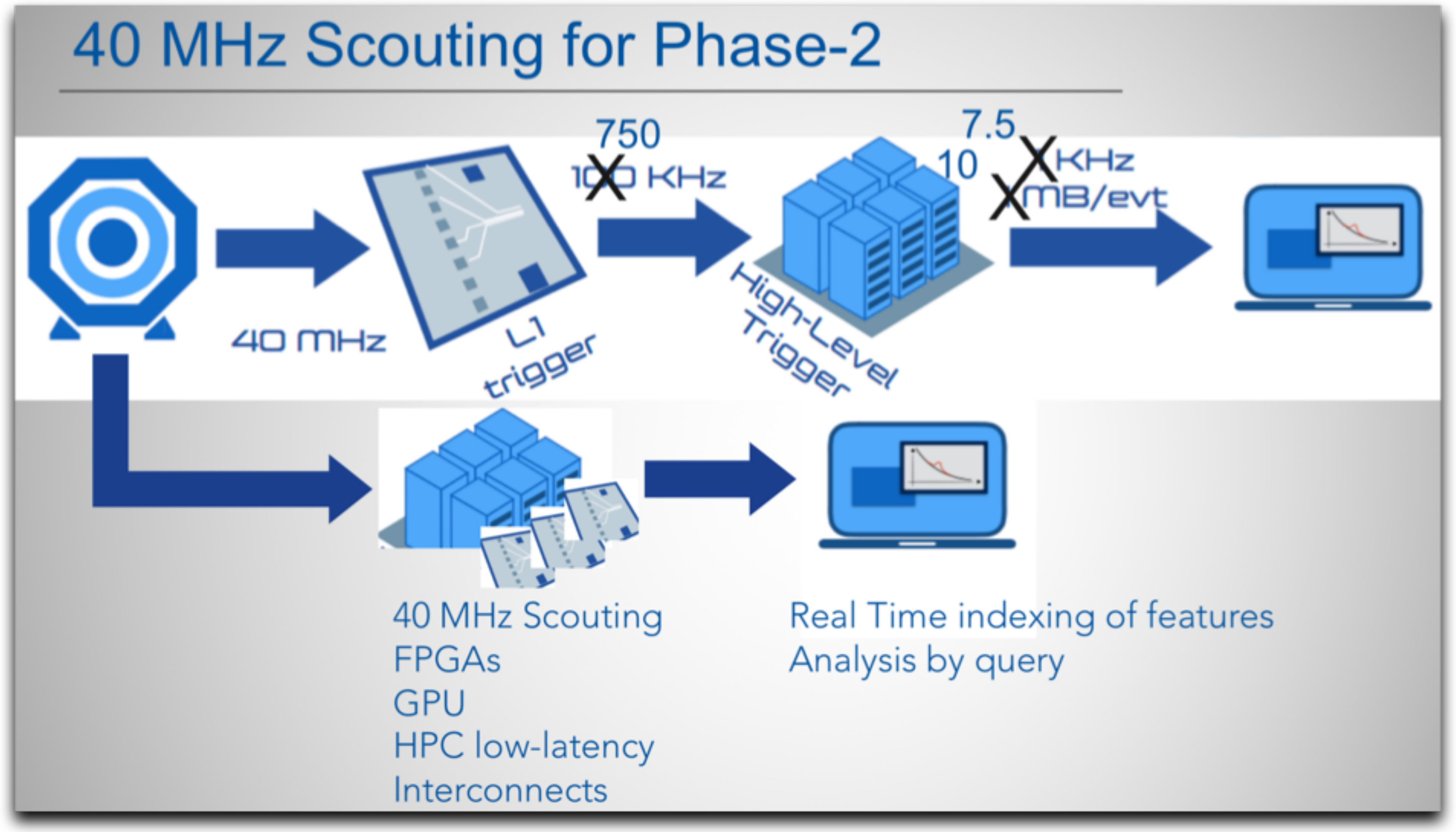
- Provided via **various tools**
  - KFServing, TensorFlow Serving, Seldon…
- Simplified with Kubeflow Fairing
- Idea - create a server as a **Kubernetes pod**
- Access server **endpoint via API**
  - *curl -v -H “Host: hostname” “http://host_ip/v1/models/mnist:predict*" *-d @./input.json*
- Current status - **not working** due to networking issues
  - Expected to be fixed by the end of October

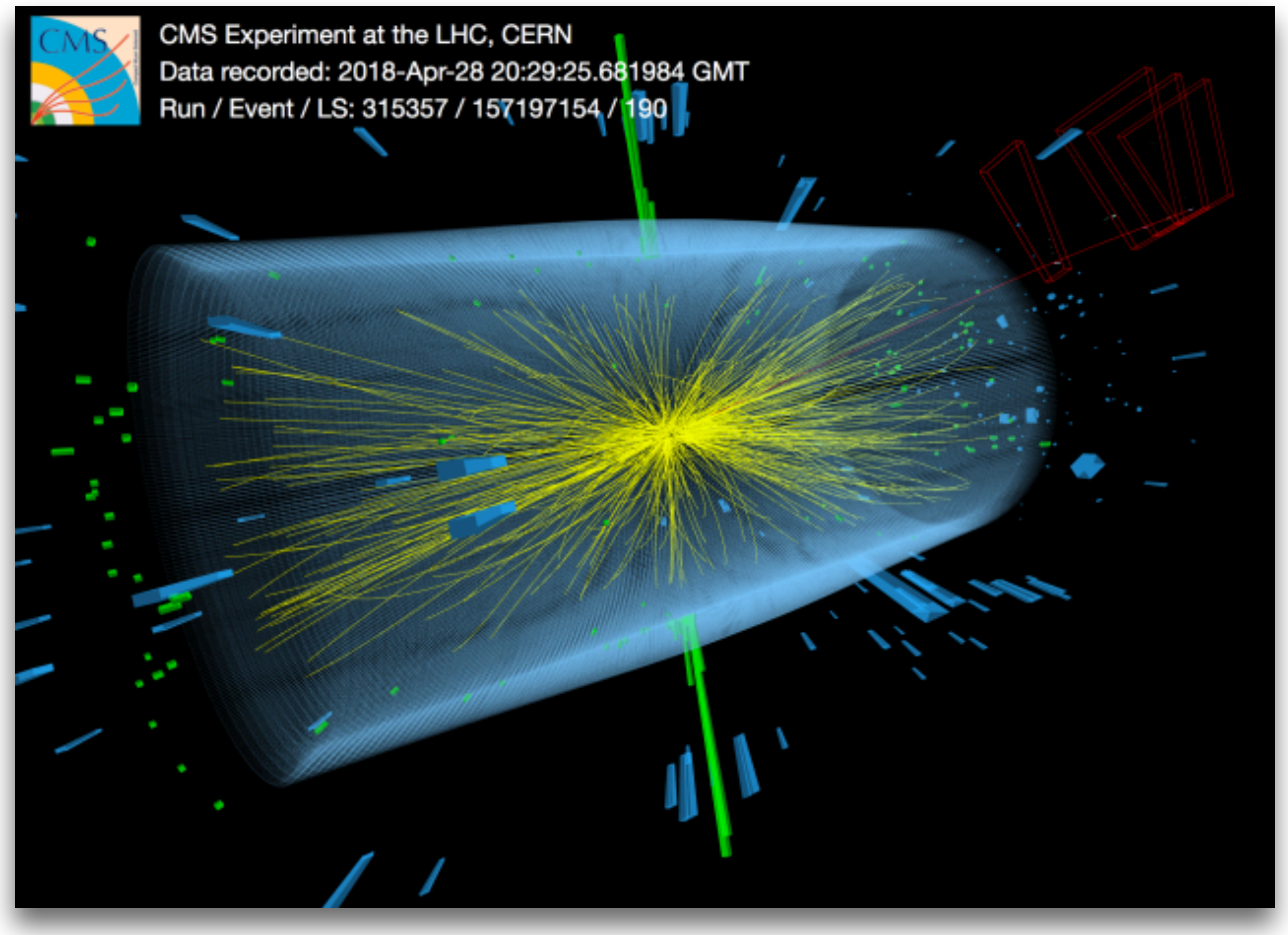
Demo

Use Cases

- Fast simulation with 3dGAN (ongoing)
- DUNE experiment, CNNs
- CMS 40MHz Scouting, MLPs







Upcoming

- Fix Kubeflow issues
    - KALE Notebook to Katib conversion
    - Model serving
    - EOS integration without *kinit*
- Integrate 64 T4 GPUs
- Obtain initial feedback from users
- Stable version of *ml.cern.ch* cluster - end of October

Thank you for the attention!

Questions?