Hewlett Packard Enterprise

HPE Reference Architecture for HPE Ezmeral ML Ops on Kubernetes

Providing DevOps Speed and Agility for Machine Learning

CONTENTS

EXECUTIVE SUMMARY

Enterprises across all industries are embarking on a hybrid cloud journey for the development and deployment of their data-driven analytics and AI/ML applications. The continuous integration/continuous deployment (CI/CD) workflows, collectively referred to as DevOps, have become ubiquitous for all software development today. On the machine learning front, data scientists still spend a significant amount of time and effort moving projects from development to production. Model versioning is still manual, making it hard to update models in production. Code sharing is manual; data is copied onto local storage leading to the variability of results between environments. There is also a lack of standardization on the tools and frameworks used, which makes it tedious and time-consuming to deploy models across all environments.

HPE Ezmeral ML Ops includes the same capabilities and functionality as the HPE Ezmeral Runtime while also providing DevOps-like agility to enterprise machine learning. With HPE Ezmeral ML Ops, enterprises can implement CI/CD workflows and standardize their ML pipelines. The HPE Ezmeral ML Ops software platform supports every stage of the machine learning lifecycle — from supporting sandbox experimentation with the choice of ML/DL frameworks and integrating with model and code repositories to deploying and tracking models in production.

HPE Ezmeral ML Ops gives data scientists and developers the ability to quickly and easily build and train machine learning models. It allows data scientists to manage and track models built on any platform and deploy them into a scalable and secure production environment. Using HPE Ezmeral ML Ops, data scientists can spin up containerized environments for distributed data processing, Machine Learning (ML), or Deep Learning (DL) in minutes rather than weeks. It provides data science teams the flexibility to run their ML/DL workloads either on-premises, in multiple public clouds, or a hybrid model and respond to dynamic business requirements in a variety of use cases.

With HPE Ezmeral ML Ops, Hewlett Packard Enterprise is making it easier for organizations to deliver a flexible and secure multitenant architecture, with the agility, flexibility, and performance needed to address today's evolving workload and application requirements. Its deployment on HPE hardware can be done using pre-tested and optimized HPE Apollo building blocks on-premises, as well as in hybrid IT architectures and a multi-cloud model.

SOLUTION OVERVIEW

HPE Ezmeral Runtime is a unified container software platform that is built on open-source Kubernetes and designed for both cloud-native applications and non-cloud-native applications running on any infrastructure either on-premises, in multiple public clouds, in a hybrid model, or at the edge. With HPE Ezmeral Runtime, container deployment and operations can be simplified at scale. HPE Ezmeral Runtime best practices and automation can help streamline operations and improve SLAs. Hewlett Packard Enterprise delivers highly automated playbooks for Day 0 deployments combined with best practices and configuration automation to set up container HA, backup/restore, security validation, and monitoring to minimize manual overheads. HPE Ezmeral Container Platform includes KubeDirector—an open-source Kubernetes-based controller that can be used to deploy non-cloud-native apps. The HPE Ezmeral Runtime provides an App Store of curated, prebuilt images for a wide range of applications including machine learning (ML), analytics, IoT/edge, and CI/CD.

In this Reference Architecture, we discuss HPE Ezmeral ML Ops on Kubernetes and its components including the HPE Ezmeral Runtime. This is a workload-optimized platform to serve the needs of DevOps teams, CI/CD workflow integration, application modernization, and hybrid cloud solutions for the enterprise. This solution provides a cloud-like experience to customers from edge to core to cloud.

The HPE Apollo fit-for-purpose built server, storage, and networking hardware is the foundation for an infrastructure that provides both rapid deployment and scaling while delivering the highest levels of performance, quality, and availability. This solution also showcases how to modernize a legacy application using KubeDirector.

The combination of HPE Ezmeral Runtime (with pre-integrated HPE Ezmeral Data Fabric), and HPE Apollo servers deliver a composable architecture that rapidly deploys containers supporting the latest application frameworks. Ultimately, this results in faster digital transformation for the business. With services from HPE Pointnext and HPE GreenLake, the customer decides whether to purchase hardware upfront or move to a pay-as-you-go consumption model.

Figure 1 shows the architecture of the HPE Ezmeral Runtime.

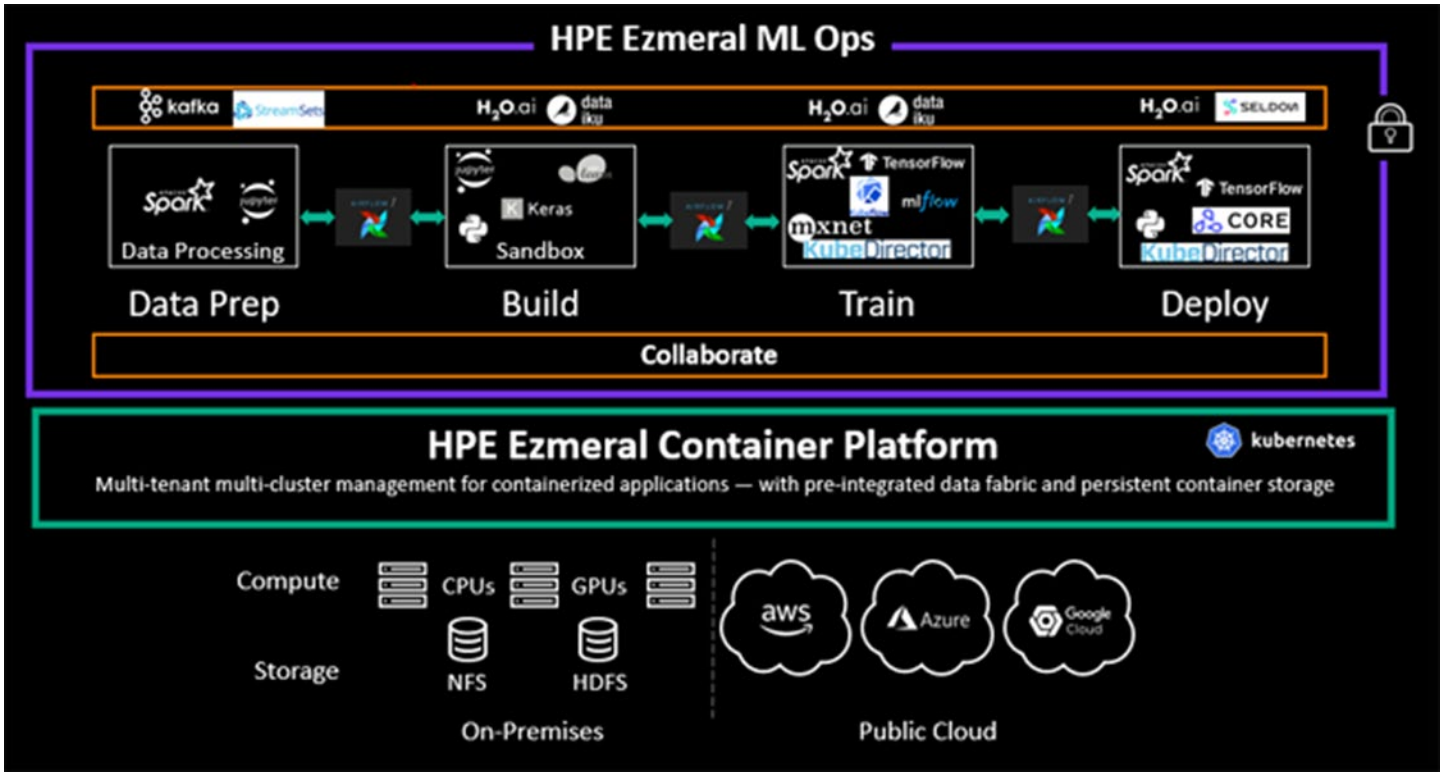


FIGURE 1. HPE Ezmeral ML Ops Architecture

HPE Ezmeral ML Ops brings the power of Kubernetes pods and Docker containers to the entire machine learning lifecycle to allow customers to build, train, deploy, and monitor machine learning (ML) and deep learning (DL) models. It supports sandbox development (notebooks), distributed training, and the deployment and monitoring of trained models in production. Project repository, source control, and model registry features allow seamless collaboration.

Some of the specific features supplied at each stage of the machine learning lifecycle include:

Table 1 shows the steps of the ML Lifecycle.

TABLE 1. ML Lifecycle

Steps	Description	Users
Data Preparation	• Select data • Preprocess the data by formatting, cleaning, and sampling it • Transform data by scaling, decomposing, and aggregating it • Use for fast data ingest and movement	Data Analyst
Model Building	• Containerized sandbox environments • Choice of ML/DL tools, interface, and frameworks • Secure access to shared data • Data Scientists can now quickly spin up environments using their preferred data science tools	Data Scientist, App Developers
Model Training	• Containerized, distributed ML/DL environment • Auto-scaling capabilities • Prepackaged images for Python, Spark, and TensorFlow	Data Scientist, Data Engineer, DevOps Engineer
Model Deployment	• Support multiple runtime engines for handling scoring logic (e.g. Python, R, etc.) • Can deploy distributed ML/DL environments such as TensorFlow, Caffe2, H2O, BigDL, and SparkMLlib • REST endpoints with token-based authorization	Data Scientist, DevOps Engineer

Steps	Description	Users
	• Autoscaling and load balancing • Integration with the model registry allows data scientists to track model versions and seamlessly update models	

The following are some additional features of ML Lifecycle in HPE Ezmeral ML Ops:

- **Model Monitoring**
  - Track, measure, and report model performance
  - Save and inspect inputs and outputs for each scoring request
  - Third-party integrations track accuracy and interpretability
- **Collaboration**
  - CI/CD workflows with code, model, and project repositories
  - Integration with GitHub and Bitbucket for project/code repository
  - Storing multiple models (multiple versions with metadata) for various runtime engines
  - A/B or Canary testing to validate the model
  - NFS based project repository that eases collaboration
- **Security and control**
  - Secure multi-tenancy with integration to enterprise authentication mechanisms
  - Multitenancy and data isolation to ensure logical separation between each project, group, or department within the organization
  - Enterprise security and authentication mechanisms such as LDAP, Active Directory, and Kerberos
  - Share the same infrastructure and access the same data sources for AI/ML and Big Data analytics workloads
- **Hybrid deployment**
  - On-premises, public cloud, or hybrid
  - Run on-premises on any infrastructure (including in multiple data centers)
  - Supports multiple public clouds (Amazon® Web Services, Google® Cloud Platform, or Microsoft® Azure)
  - Provides a hybrid model for effective utilization of resources and lower operating costs

In addition to the new features and business benefits delivered through the HPE Ezmeral ML Ops Software, the underlying functionality delivered previously via the HPE Ezmeral Container Platform will continue to be part of the overall HPE Ezmeral Runtime framework and integrated with the new HPE Ezmeral Machine Learning Ops Software.

This Reference Architecture describes our solution testing performed in February 2021.

Document purpose: This Reference Architecture provides an overview of the deployment of HPE Ezmeral Runtime on servers – HPE Apollo 6500, HPE Apollo 2000. Also, it provides deployment steps of Kubeflow for pipeline management and HPE Ezmeral Data Fabric (formerly 'MapR') based Spark Operator.

SOLUTION COMPONENTS

HPE Ezmeral Runtime

HPE Ezmeral Runtime installs as a software layer between the underlying server infrastructure and the Big Data distribution, AI/ML libraries, and applications. The use of containers is completely transparent, and HPE Ezmeral Runtime customers benefit from greater agility and bare-metal performance due to the lightweight nature of containers. They can leverage the flexibility of containers to simplify the development of DevOps, CI/CD pipelines, and applications across hybrid cloud deployments.

Key features

- **Multi-cluster Kubernetes management:** Fast, easy deployment, management and monitoring of multiple clusters with either out-of-the-box or default configuration for networking, load balancing, and storage. This permits the user to run and manage different versions of Kubernetes simultaneously, and seamlessly supports the in-place upgrades.
- **Enterprise-ready persistent container storage**: Fully managed, integrated, scale-out, and edge-ready persistent storage with the HPE Ezmeral Data Fabric. This Data Fabric, along with DataTap and FS Mount functionality provides connectivity to data without copying the data locally.
- **100% open-source Cloud Native Computing Foundation (CNCF) Kubernetes**: With innovations such as KubeDirector—an open-source Kubernetes-based controller to deploy non-cloud-native, stateful apps. HPE Ezmeral Runtime is a CNCF-certified Kubernetes distribution.
- **One-click provisioning**: Pre-packaged App Store with curated, prebuilt images for a wide range of applications including machine learning (ML), analytics, IoT/edge, CI/CD, and other modern apps. The pre-bundled contents of the apps include Helm Charts, Operators, YAML configuration files, and KubeDirector scripts.
- **Simplified installation and upgrade workflows**: This includes installation on bare metal, Virtual Machines, and cloud instances.
- **Flexible multi-cluster, multitenant control plane**: Deploy multiple open-source K8s clusters and manage cloud K8s clusters (Example: GKE, EKS) from an HPE Ezmeral Runtime control plane, without vendor lock-in or modification to native K8s.
- **KubeDirector**: The first and only K8s custom controller that deploys non-cloud native, monolithic distributed stateful applications (Example: CDH, HDP, Confluent, Bring your own app).
- **Streamlined access to K8s clusters and services for end-users**: Gateway hosts isolate the HPE Ezmeral Runtime control plane and K8s hosts from the user network. This uniquely provides load balancing to multi-master K8s cluster(s) and routes to K8s services exposed via Node Ports and Ingress Controllers.
- **Bare-metal performance**: HPE Ezmeral Runtime provides storage I/O optimizations to deliver data to applications without the penalties commonly associated with virtualization or containerization. The compute cores and RAM in each host are pooled and then partitioned into virtual resource groups based on tenant requirements.
- **Self-Service Environments**: Users can get up and run quickly with HPE Ezmeral Runtime Elastic Plane functionality. New containerized environments are provisioned on-demand with just a few mouse clicks—whether they're transient for development and testing, or long-running for a production workload. Data scientists and analysts can now quickly respond to dynamic business requirements for a variety of use cases ranging from deep learning with AI Frameworks like TensorFlow to analytical SQL workloads running on Hadoop. Flexibility for Tools of Choice: The HPE Ezmeral Runtime offers pre-integrated container images, including many of the most common AI and Big Data tools, ready-to-run versions of major Hadoop distributions, such as Cloudera (CDH), Hortonworks (HDP), and MapR (CDP). Also includes recent versions of Spark standalone as well as Kafka and Cassandra.
- **Compute and Storage Separation:** HPE Ezmeral Runtime disconnects analytical processing from data storage, giving users the ability to independently scale compute and storage based on the needs of the workloads.
- **Data Access from Any Storage:** With HPE Ezmeral Runtime DataTap capability, users can access data from any shared storage system (including HDFS as well as NFS) or cloud storage (e.g. Amazon S3). It is unnecessary need to make multiple copies of data or move data before running an analysis. Sensitive data can stay in a secure storage system with enterprise-grade data governance without the costs and risks of creating and maintaining multiple copies or moving large-scale data.

HPE Ezmeral Runtime goes beyond Hadoop and Spark support by leveraging the inherent infrastructure portability and flexibility of containers to support distributed AI for both ML and DL use cases. The separation of compute and storage for Big Data and ML/DL workloads is one of the key concepts behind this flexibility. Organizations can deploy multiple containerized compute clusters for different workflows (e.g. Spark, Kafka, or TensorFlow) while sharing access to a common data lake. This also enables hybrid and multi-cloud HPE Ezmeral Runtime deployments, with the ability to mix and match on- and/or off-premises compute and storage resources to suit each workload. Furthermore, compute resources can be quickly and easily scaled and optimized independently of data storage, thereby increasing flexibility and improving resource utilization while eliminating data duplication and reducing cost.

Kubernetes features on HPE Ezmeral ML Ops

Data engineers, ML architects, and others can spin up containerized Kubernetes environments on scalable compute clusters with their choice of machine learning tools and frameworks for Big Data, AI, and/or ML use cases. Some of the key features of Kubernetes on HPE Ezmeral ML Ops include:

- **Software installation**: either on physical or virtual hosts located in a hybrid environment.
- **Storage medium**: a pre-integrated persistent container storage system known as the HPE Ezmeral Data Fabric.
- **DataTaps and FS Mounts**: access to existing data sources, with no need to copy data back and forth. See DataTaps and FS Mounts.
- **Multitenant, multi-cluster management**: use open-source Kubernetes orchestration to run a variety of databases, analytics, AI/ML, CI/CD pipeline, and other applications.
- **Big Data Kubernetes tenants**: HPE Ezmeral Runtime can deploy applications with KubeDirector, or onboard Kubectl deployed applications, from the built-in Kubernetes Applications screen.
- **KubeDirector**: KubeDirector custom resource comes pre-installed with the HPE Ezmeral Runtime. The set of applications that can be automatically launched into a cluster is found by accessing a Kubernetes tenant and then clicking the Applications tab. See the Applications article at this link.

Kubernetes architecture with HPE Ezmeral Runtime

This diagram depicts the physical Kubernetes cluster architecture within the HPE Ezmeral Runtime. For details on Kubernetes physical architecture, see the following link.

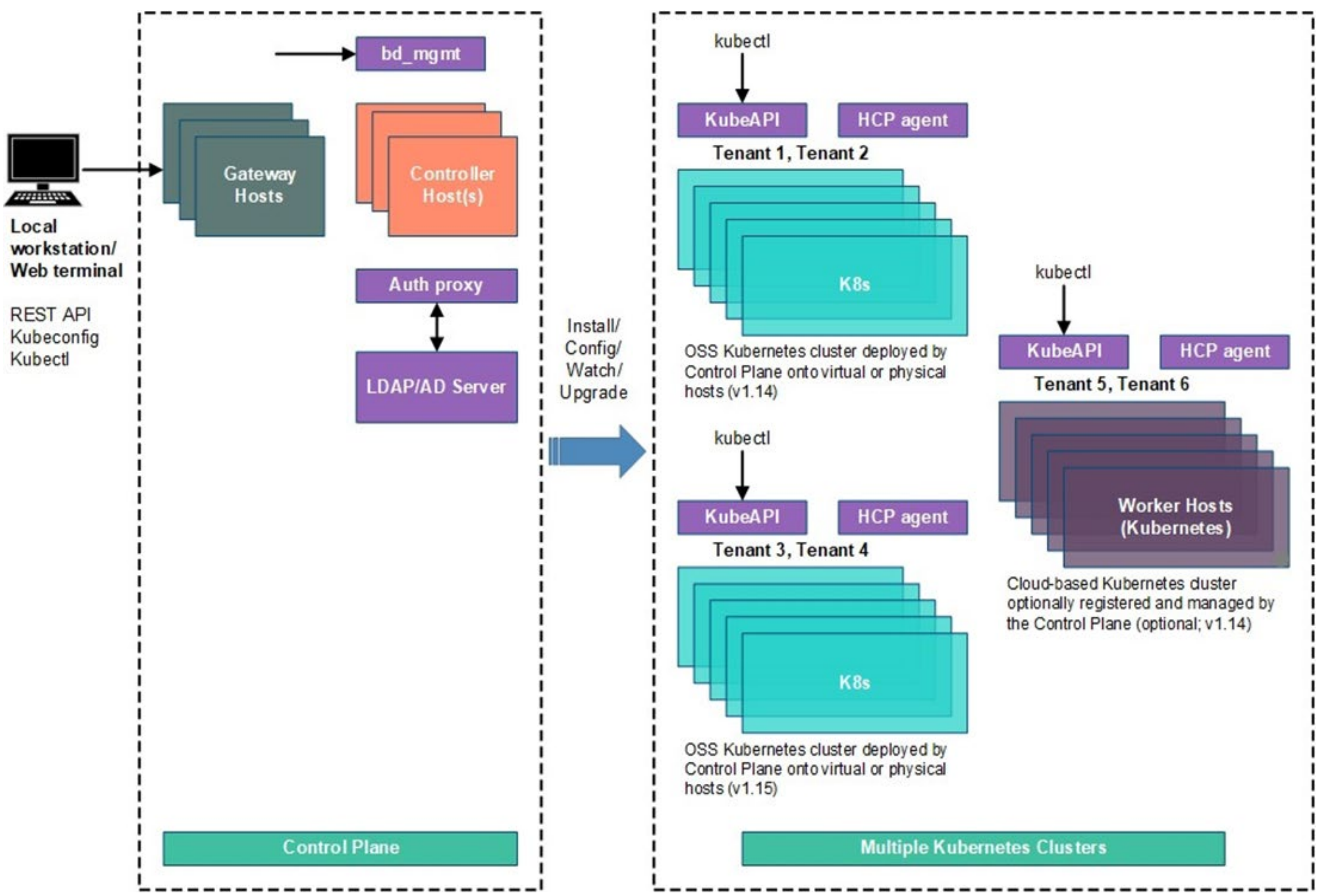


FIGURE 2. Kubernetes physical architecture in HPE Ezmeral Runtime

HPE Ezmeral ML Ops

With the HPE Ezmeral ML Ops solution, data science teams involved in the ML/DL model lifecycle can benefit from the industry's most comprehensive operationalization and lifecycle management solution for enterprise AI.

Figure 3 shows these features in the ML/DL lifecycle causal relationship. For further details, see HPE Ezmeral Container Platform 5.3 Documentation.

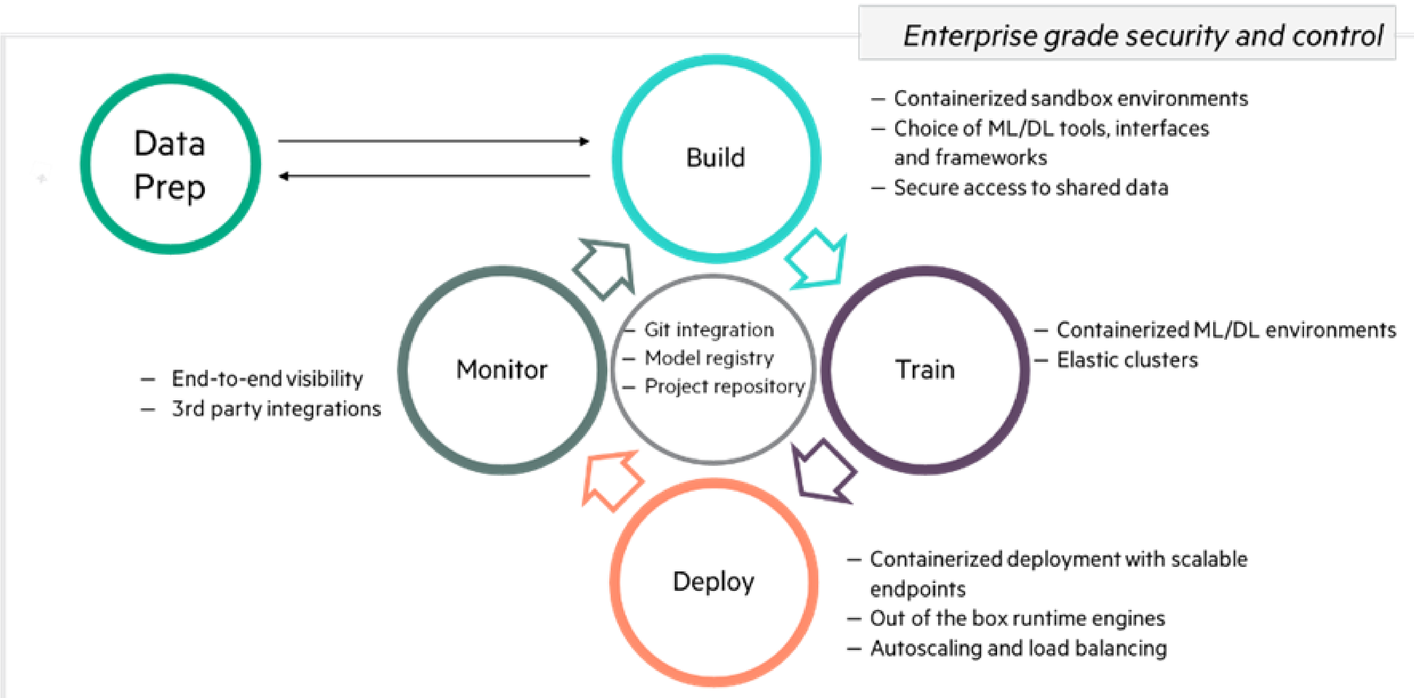


FIGURE 3. ML/DL lifecycle

HPE Ezmeral ML Ops delivers the following AI/ML features in addition to the HPE Ezmeral Runtime functionality:

- Leverage the power of containers to create a complex machine learning and deep learning stacks that include distributed TensorFlow, Apache Spark on Yarn with Kerberos, H2O, and Python ML and DL toolkits.
- Spin-up distributed, scalable, machine learning, and deep learning training environments on-premises, public cloud, or in a hybrid model.
- Support for a variety of programming languages and open-source tools designed to support even the most complex ML pipelines. For example, start with data pre-processing in Spark with Scala, followed by model development with TensorFlow on GPUs, and finally model deployment on CPUs with TensorFlow runtime.
- The model registry stores models and versions created within HPE Ezmeral ML Ops, as well as those created using different tools/platforms.
- Improves the reliability and reproducibility of machine learning projects in a shared project repository (GitHub).
- Enables the deployment of models in production with secure, scalable, highly available endpoint deployment with out-of-the-box auto-scaling, and load balancing.
- Enables out-of-the-box application images to be rapidly deployed in containerized environments – sandbox, distributed training, or serving (inferencing).
- Enables the creation of custom application images with any combination of tools, library packages, and frameworks.

For additional information, see HPE Ezmeral ML Ops.

Kubeflow for Pipeline Management

Kubeflow is an open-source project designed to make machine learning workflows on Kubernetes simple, portable, and scalable. Kubeflow is sponsored by Google and inspired by TensorFlow Extended, or TFX, the company's internal machine learning platform. Originally intended to simply allow TensorFlow users to run training jobs on their Kubernetes clusters, the project now integrates a broad set of tools in support of many steps in an end-to-end machine learning process.

Kubeflow includes components for:

- Launching Jupyter Notebooks
- Building ML Pipelines
- Training Models
- Tracking Experiment Metadata
- Hyperparameter Tuning
- Serving Models
- Monitoring
- Continuous integration and deployment for ML

Spark within HPE Ezmeral Runtime

Spark is a data processing framework that can rapidly perform processing tasks on massive data sets and can also allocate data processing tasks across multiple nodes. Spark can work in stand-alone mode or together with other distributed computing tools. These features are the key to Big Data and ML/DL, which require significant computing power to crunch through large data stores. It also helps the developers by getting rid of some of the programming burdens with an easy-to-use API that abstracts much of the monotonous work of distributed computing and big data processing.

Spark has become one of the key big data distributed processing frameworks in the world. It can be deployed in a variety of ways and provides native bindings for the Java, Scala, Python, and R programming languages. In addition, it supports SQL, streaming data, machine learning, and graph processing.

Another Spark advantage is increased performance using its in-memory data engine. It can run tasks up to several orders of magnitude faster than MapReduce in certain situations. Furthermore, it offers the developer-friendly Spark API which hides much of the complexity of a distributed processing engine behind simple method calls.

The tutorial in the Spark Operator with K8s section describes how to set up and execute a Spark framework within the HPE Ezmeral Runtime which allows users to run Spark workloads on Kubernetes clusters. A user may instantiate the Spark framework instance on a dynamic cluster using the Spark operator. The Spark operator is a Kubernetes custom resource that is installed in a tenant namespace to support the on-demand deployment of Spark Executor pods. These pods are deleted once job execution completes.

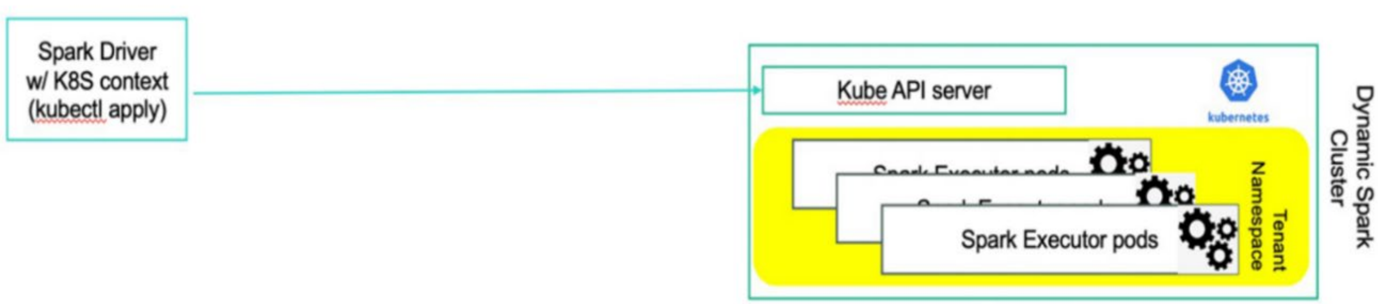


FIGURE 4. Spark Operator on Kubernetes

BEST PRACTICES

Deploying the HPE Ezmeral Runtime on the HPE Elastic Platform for Analytics (EPA) platform provides great flexibility in deploying workloads and managing resource growth, by decoupling storage from compute. This section is intended to provide high-level guidance and best practices for deploying HPE Ezmeral Runtime, and HPE Ezmeral ML Ops solution with the HPE Elastic Platform for Analytics.

HPE EPA Platform configuration for HPE Ezmeral Runtime

HPE Ezmeral Runtime uses four distinct host types, as shown in Table 2, with the recommended HPE EPA server model.

TABLE 2. HPE Ezmeral Runtime Host Types (Intel)

Host Type	HPE Server Model
Primary Controller, Shadow Controller, Arbiter	HPE Apollo 2000 Needs 3 x XL170r for controller nodes with HA
Kubernetes– Master/Worker	HPE Apollo 2000 (4 x HPE ProLiant XL170r) Recommend minimum 3 x HPE ProLiant XL170r for Kubernetes Worker/Compute nodes
Kubernetes – Compute with GPUs	• HPE Apollo 2000 with up to 2 x HPE ProLiant XL190r each with 1 or 2 GPUs • HPE Apollo 6500 with HPE ProLiant XL270d with 4 or 8 GPUs • HPE ProLiant DL380 with up to 4 GPUs
Kubernetes - Gateway	HPE Apollo 2000 Recommend 2 x HPE ProLiant XL170r

TABLE 3. HPE Container Platform Host Types (AMD)

Host Type	HPE Server Model
Primary Controller, Shadow Controller, Arbiter	HPE Apollo 2000 Gen10 Plus Needs 3 x HPE XL225n Gen10 Plus for controller nodes with HA
Kubernetes– Master/Worker	4 x HPE ProLiant DL325 Gen10 Plus Recommend minimum 3 x HPE ProLiant DL385 Gen10 Plus for Kubernetes Worker/Compute nodes
Kubernetes – Compute with GPUs	HPE Apollo 6500 (4 x HPE ProLiant DL675d Gen10 Plus)
Kubernetes - Gateway	HPE Apollo 2000 Gen10 Plus Recommend 2 x HPE XL225n Gen10 Plus

HPE Apollo 2000 Gen10 Plus Compute Servers

The HPE Apollo 2000 Gen10 Plus System is a shared infrastructure chassis with flexible support for up to four (4) HPE ProLiant XL225n Gen10 Plus servers (AMD) or up to four (4) HPE ProLiant XL220n Gen10 Plus servers (Intel®) or two (2) XL290n Gen10 Plus servers (Intel), helping increase rack space density. Server nodes can be serviced without impacting the operation of other nodes in the same chassis for increased server up-time. It delivers the flexibility to tailor the system to the precise needs of demanding high-performance computing (HPC) workloads with the right compute, flexible I/O, and storage options. The system can be deployed with a single server, leaving room to scale as customers' needs grow, bringing the power of supercomputing to data centers of any size. It is ideal for HPC applications in industry verticals like manufacturing, oil and gas, life sciences, and financial services.

- **HPE ProLiant XL170r Gen10 Server:** For compute-intensive workloads, HPE ProLiant XL170r delivers four servers in a single 2U chassis. Each HPE ProLiant XL170r server is serviced individually without impacting the operation of other servers sharing the same chassis to provide increased server uptime.
- **HPE ProLiant XL225n Gen10 Plus Server:** 1U Node Configure-to-order Server supports the full stack of 2nd generation AMD® EPYC™ 7000 Series processors.

For more information, see HPE Apollo 2000 servers.

HPE Apollo 6500 Gen10 Plus GPU System

Built for the exascale era, the HPE Apollo 6500 Gen10 Plus System accelerates performance with NVIDIA® HGX A100 Tensor Core GPUs and AMD Instinct™ MI100 with Infinity Fabric™ accelerators to take on some of the most complex HPC and AI workloads. This purpose-built platform provides enhanced performance with premier graphics processing units (GPU), fast GPU interconnect, high-bandwidth fabric, and configurable

GPU topology, providing rock-solid reliability, availability, and serviceability (RAS). Configure with single or dual processor options for a better balance of processor cores, memory, and I/O. Improve system flexibility with support for 4, 8, 10, or 16 GPUs and a broad selection of operating systems and options, all within a customized design to reduce costs, improve reliability, and provide leading serviceability.

- **HPE ProLiant XL675d Gen10 Plus**: It is a dual-processor system for the NVIDIA HGX A100 8-GPU or AMD Instinct with 8 to 10 double-wide or 16 single-wide PCIe accelerators.

For more detailed information, see HPE Apollo 6500 Gen10 Plus system.

HPE and NVIDIA GPUs

HPE ProLiant servers offer NVIDIA accelerators for high-performance computation for deep learning, high-performance computing (HPC) workloads, or graphics. The NVIDIA accelerators for HPE ProLiant servers seamlessly integrate GPU computing with select HPE server families. Designed for power-efficient, high-performance supercomputing, NVIDIA accelerators deliver dramatically higher application acceleration than a CPU-only approach for a range of deep learning, scientific, and commercial applications. The thousands of NVIDIA CUDA® cores of each accelerator allow it to divide large computing or graphics tasks into thousands of smaller tasks that can be run concurrently, thus enabling much faster simulations and improved graphics fidelity for extremely demanding 3D models.

For more detailed information, see HPE AI and deep learning.

HPE Intelligent System Tuning (IST)

Available in HPE ProLiant Gen10 servers, HPE Intelligent System Tuning is a new set of revolutionary capabilities that deliver higher levels of performance, agility, and control to the server environment. With these groundbreaking new features, we can:

- Dynamically tune the servers' performance to match the needs of each workload
- Drive real cost savings
- Radically improve server performance

HPE ProLiant Gen10 Servers offer a UEFI configuration option to help customers tune their BIOS settings by using the known workload-based tuning profiles developed by the HPE performance engineering team. The default BIOS settings on HPE servers provide a balance between performance and power efficiency. For workloads running on the HPE ProLiant XL190r Gen10 and HPE Apollo 6500 servers with GPUs, the recommended workload profile is Graphic Processing which disables power management and virtualization to optimize the bandwidth between I/O and memory.

For more information about how to tune an HPE ProLiant Gen10 server using the workload profiles, refer to the UEFI workload-based Performance Tuning Guide for HPE ProLiant Gen10 servers.

TABLE 4. HPE Ezmeral Runtime host CPU and memory recommendations (Intel)

Host Type	Deployment Size	Memory	Processor
Ezmeral - Controller, Shadow Controller, Arbiter	Starter	192 GB	2 x Intel® Xeon® Gold 5215 - 10C 2.5 GHz
	Medium	384-768 GB	2 x Intel® Xeon® Gold 6242 - 16C 2.8 GHz
K8s – Master/Compute	Starter	192 GB	2 x Intel® Xeon® Gold 5215 - 10C 2.5 GHz
	Medium	384-768 GB	2 x Intel® Xeon® Gold 6226 - 12C 2.7 GHz
K8s – GPU	Starter	384 GB	2 x Intel® Xeon® Gold 5215 - 10C 2.5 GHz
	Medium	384-768 GB	2 x Intel® Xeon® Gold 6226 - 12C 2.7 GHz
	Large	384-768 GB	2 x Intel® Xeon® Gold 6242 - 16C 2.8 GHz
K8s – Gateway	All	192 GB	2 x Intel® Xeon® Gold 5215 - 10C 2.5 GHz

TABLE 5 HPE Ezmeral Runtime host CPU and memory recommendations (AMD)

Host Type	Deployment Size	Memory	Processor
	Starter	192 GB	AMD EPYC 7742 - 64C 2.25GHz
Ezmeral - Controller, Shadow Controller, Arbiter	Medium	384-768 GB	AMD EPYC 7742 - 64C 2.25GHz

Host Type	Deployment Size	Memory	Processor
K8s – Master/Compute	Starter	192 GB	AMD EPYC 7262 - 8C 3.2GHz
	Medium	384-768 GB	AMD EPYC 7262 - 8C 3.2GHz
K8s – GPU	Starter	384 GB	AMD EPYC 7542 - 32C 2.9GHz
	Medium	384-768 GB	AMD EPYC 7542 - 32C 2.9GHz
	Large	384-768 GB	AMD EPYC 7542 - 32C 2.9GHz
K8s – Gateway	All	192 GB	AMD EPYC 7742 - 64C 2.25GHz

To assist in sizing an HPE Ezmeral Runtime cluster, Hewlett Packard Enterprise has developed a sizing tool.

NOTE
HPE Apollo Gen10 Plus systems support a variety of flexible memory configurations. But for optimal performance, it is recommended to balance the total memory capacity across all installed processors and make use of all six memory channels per CPU with up to two DIMM slots per channel.

Storage

A typical Kubernetes environment may have pods frequently coming and going. Large Kubernetes environments, such as a public cloud, may handle pools of systems where new hosts are added to support pod and cluster placement. In the HPE Ezmeral Runtime, a Data Fabric cluster is a Kubernetes Custom Resource that functions as a storage cluster providing access to PVCs, tenant storage, shares, and other storage needs. In a Data Fabric cluster:

- The hosts (called nodes) commit considerable disk resources that may include NVMe and enterprise-class SSDs
- The Data Fabric cluster can be deployed on a small number of nodes
- Unlike a typical k8s environment, pods are not deleted frequently
- The Data Fabric cluster must account for host resource profiles to guarantee core pod availability

HPE Ezmeral Runtime includes native support for HPE Ezmeral Data Fabric. This automates many manual steps and allows the creation of Data Fabric clusters like that used for creating Compute Kubernetes clusters (see Creating a New Data Fabric Cluster and Creating a New Kubernetes Cluster). Each Data Fabric cluster resides on nodes. See Kubernetes Worker Installation Overview and Kubernetes Data Fabric Node Installation Overview.

Ephemeral Storage (Node Storage)

Ephemeral Storage is built from the local storage in each host and is used for the disk volumes that back the local storage for each virtual node. Using SEDs (Self-Encrypting Drives) will ensure that any data written to node storage is encrypted on write and decrypted on read by the OS. A tenant can optionally be assigned a quota for how much storage the nodes in that tenant can consume.

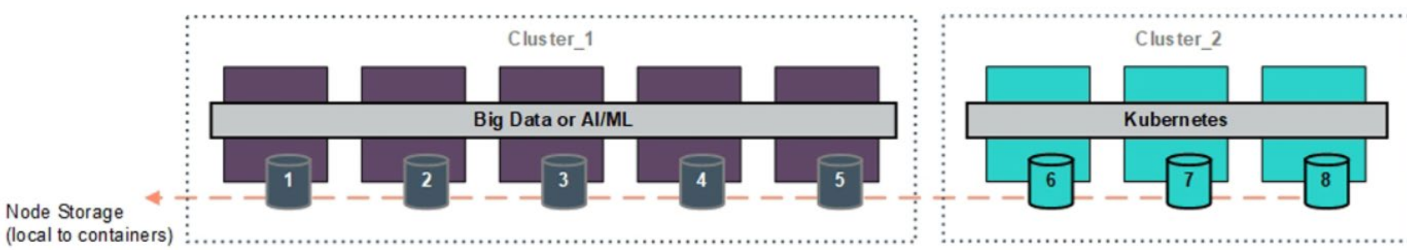


FIGURE 5. Storage

Virtual nodes/containers running on public cloud VMs (such as AWS EC2) utilize storage within the instance (such as AWS Elastic Block Storage, or EBS) as node storage.

Persistent Storage

Deploying a persistent data fabric is supported on the local disks within the hosts. This local storage can serve as either HDFS storage or as persistent volumes for Kubernetes clusters. Persistent volumes for Kubernetes stateful clusters are seamlessly available either from the native persistent data fabric or Nimble Storage using the storage interface driver (CSI) that is deployed during cluster creation.

Tenant Storage for Local Data Access

HPE Ezmeral Runtime can deploy a Data Fabric (MapR) file system on local disks for Tenant Storage within the servers running the K8s services. The DataTap interface then surfaces the physical locations of the Tenant Storage data blocks to the containers that make up the virtual cluster. This allows the Big Data task scheduling software running within the containers to route Big Data tasks to the containers running on the physical servers where copies of the required data blocks reside. This behavior mimics bare-metal Big Data deployments, thereby preserving the performance advantages of data locality without losing the flexibility and agility of a container-based virtualized compute platform. It also allows Big Data datasets to persist beyond the lifespan of a given Big Data cluster.

Operating system storage

For all host types, the recommended storage for the operating system is two 960 GB SSDs in a RAID 1 configuration.

HPE Ezmeral Runtime storage recommendations

Table 4 lists the recommended minimum storage configuration for each host type.

TABLE 5. Storage recommendations for HPE ProLiant XL170r, HPE ProLiant XL190r, and HPE ProLiant XL270d

Host Type	K8s – Storage Type	Storage Recommendation
Ezmeral – Controller, Shadow Controller, Arbiter	OS Ephemeral Storage Local Data Fabric	2 x 960 GB SSD configured as RAID 1 3 x 6.4 TB mixed-use SSD. 0-1 x 2 TB SATA 7.2 SFF HDD.
K8s – Master/ Compute	OS Ephemeral Storage Local Data Fabric	2x 960 GB SSD configured as RAID 1 3 x 6.4 TB mixed-use SSD. 1 x 2 TB SATA 7.2 SFF HDD.
K8s – Gateway	OS	2 x 960 GB SSD configured as RAID 1

NOTE

The HPE ProLiant XL170r and HPE ProLiant XL190r have six drive bays. Two are used for the OS drives and four are available for node storage and local tenant storage. For additional high availability, it is recommended that the node storage disks be configured as RAID 5. Choose the appropriate size and number of disks based on node storage and local tenant storage space requirements.

Networking

HPE Ezmeral Runtime operates on two networks, as shown in Figure 6 below.

The two networks are laid out as follows:

- Network for the Controller, Worker, and Gateway hosts: This network must be both routable and part of the organization’s network that is managed by that organization’s IT department.

- Network for virtual nodes (containers): HPE Ezmeral Runtime creates and manages this network, which can be either public (routable) or private (non-routable). For Kubernetes, Canal is used as the Container Network (CN) Network Provider. The container network is typically private (non-routable) instead of public (routable). See detailed description of Public (Routable) Virtual Node Network and Private (Non-Routable) Virtual Node Network.

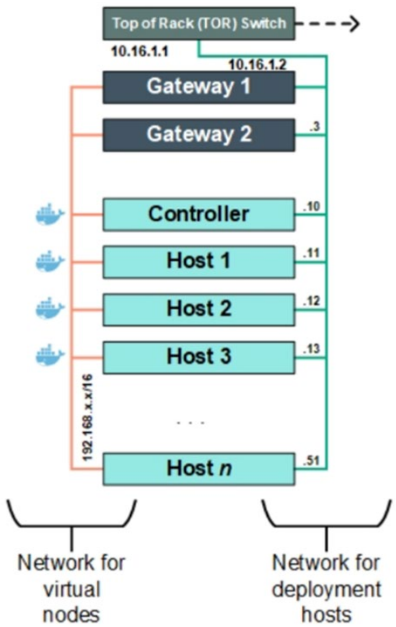


FIGURE 6. Networking layout

We recommend deploying 25GB Ethernet adapters on all the hosts. Table 6 shows the recommended networking hardware for the clusters.

TABLE 6. HPE Ezmeral Runtime networking recommendations

Host Type	Network Recommendation
All host types	HPE Eth 10/25GB 2P 640FLR-SFP28 Ethernet Adapter

KUBEFLOW WITH HPE EZMERAL ML OPS

Kubeflow conceptual overview

HPE Ezmeral Machine Learning Ops (HPE Ezmeral ML Ops) 5.3 upgraded to Kubeflow version 1.2.

Kubeflow is an open-source platform that makes the deployment of machine learning workflows in Kubernetes simple, portable, and scalable. Kubeflow deploys a suite of machine learning (ML) applications to Kubernetes that multiple users can securely access.

Kubeflow uses the Istio Gateway and LDAP authentication with the Dex authentication service to authenticate and authorize user access. Each user is assigned a profile based on a Kubernetes namespace. The profile provides an isolated view of Kubeflow for each user.

A Kubernetes administrator can install Kubeflow in environments where the computer network can access the internet, as well as in air-gapped environments where the network is isolated from outside networks.

At a high level, the execution of a pipeline proceeds as follows:

- Python SDK: Language use in the creation of components or pipelines using the Kubeflow Pipelines.
- DSL Compiler: Convert the pipeline's Python code into a static configuration (YAML).
- Pipeline Service: Call during the creation of a pipeline running from the static configuration (YAML).

- Kubernetes Resources: Allocate when Pipeline Service calls the Kubernetes API server to create the necessary resources (CRDs) to run the pipeline.
- Orchestration Controllers: A set of orchestration controllers execute the containers needed to complete the pipeline. An example controller is the Argo Workflow (demonstrated below) controller, which orchestrates task-driven workflows.
- Artifact Storage: The Pods store two kinds of data:
  - Metadata: Experiments, jobs, pipeline runs, and single scalar metrics. Metric data is aggregated to sort and filter. Kubeflow Pipelines stores the metadata in a MySQL database.
  - Artifacts: Pipeline packages, views, and large-scale metrics (time series). Use large-scale metrics to debug a pipeline run or investigate an individual run's performance. Kubeflow Pipelines stores the artifacts in an artifact store like a Minio server or Cloud Storage.
- The MySQL database and the Minio server are both backed by the Kubernetes Persistent Volume subsystem.
- Persistence Agent and ML Metadata: The Pipeline Persistence Agent (PPA) watches the Kubernetes resources created by the Pipeline Service and persists the state of these resources in the ML Metadata Service. The PPA records the set of containers along with their inputs and outputs.
- Pipeline Web Server: The Pipeline web server gathers data from various services to display relevant views: the list of pipelines currently running, the history of pipeline execution, the list of data artifacts, debugging information about individual pipeline runs, execution status of the individual pipeline runs.

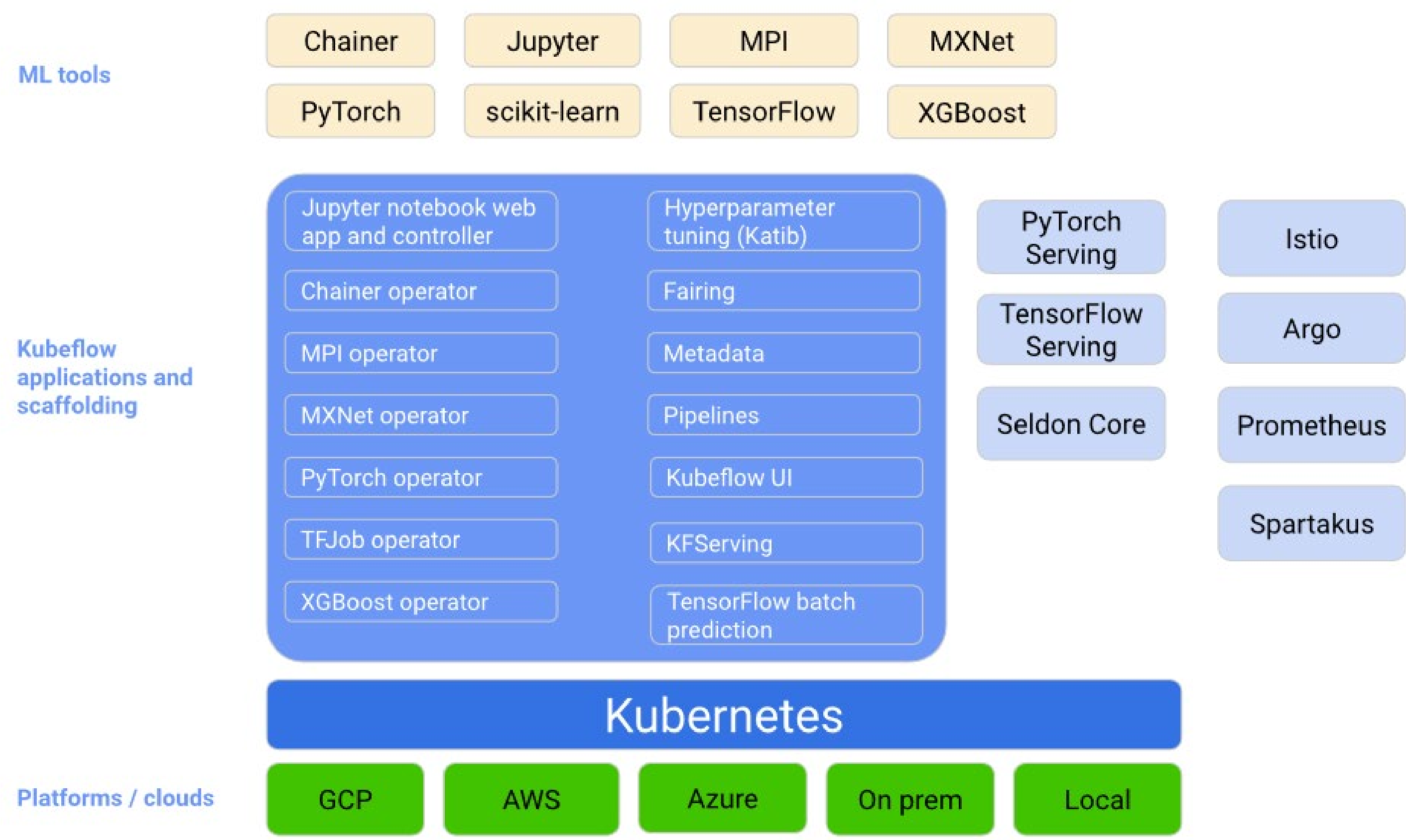


FIGURE 7. Kubeflow is a platform for components of ML systems on Kubernetes

Katib hyperparameter tuning

Katib is a Kubernetes-based system for hyperparameter tuning and neural architecture search. Katib supports many ML frameworks, including TensorFlow, MXNet, PyTorch, XGBoost, and others. It can be used to submit experiments and monitor results.

Argo Workflow

Argo Workflows is an open-source container-native workflow engine for orchestrating parallel jobs on Kubernetes. Argo Workflows is implemented as a Kubernetes CRD (Custom Resource Definition). Argo Workflows can run on any Kubernetes cluster within the HPE Ezmeral Runtime and orchestrate highly parallel jobs on Kubernetes.

Istio Prometheus

Prometheus is an open-source monitoring system and time series database. Prometheus can be used with Istio to record metrics that track the health of Istio and applications within the service mesh. Metrics can be visualized metrics using tools like Grafana and Kiali. The HPE Ezmeral Runtime supports the deployment of Prometheus

HPE EZMERAL ML OPS USE CASES

Much like pre-DevOps software development, data science organizations still spend a significant amount of time and effort moving projects from development to production. Model version control and code sharing are manual, and there is a lack of standardization on tools and frameworks thus making it tedious and time-consuming to productize machine learning models.

HPE Ezmeral Machine Learning Ops (HPE Ezmeral ML Ops) extends the capabilities of the platform and brings DevOps-like agility to enterprise machine learning. With the HPE Ezmeral ML Ops, enterprises can implement DevOps processes to standardize their ML workflows.

HPE Ezmeral ML Ops provides data science teams with a platform for their end-to-end data science needs with the flexibility to run their machine learning or deep learning (DL) workloads on-premises, in multiple public clouds, or in a hybrid model and respond to dynamic business requirements in a variety of use cases.

To complete the use case using the default datasets, 100GB of space will be required.

NOTE

The examples and documentation provided in this section are meant to supplement, not replace the HPE Ezmeral Runtime manuals.

Software components

This section describes the software versions utilized in the solution as well as notes any special installation or configuration requirements.

Table 7 lists the specific versions of the software used in this solution.

TABLE 7. Software Versions

Component	Versions
HPE Ezmeral Runtime	5.3.1
CentOS	CentOS Linux release 7.7.1908 Kernel 3.10.0-1062.el7.x86_64
389 Directory Server	1.15.1-37

Active Directory/LDAP: HPE Ezmeral ML Ops requires Active Directory (AD)/LDAP user authentication and supports the member of the attribute. The use cases use the following AD/LDAP users which can be substituted with existing LDAP users.

TABLE 8. AD/LDAP users

Projects	User	Member Of	Role
ML Ops/Kubeflow/Spark	Fraudadmin	Fraud	Admin
ML Ops/Kubeflow/Spark	fraudanalyst	Fraud	Member
ML Ops/Kubeflow/Spark	imageadmin	Image	Admin
ML Ops/Kubeflow/Spark	imageanalyst	Image	Member

Hardware components

The cluster was configured with AD/LDAP authentication, and platform high availability as shown in Figure 8.

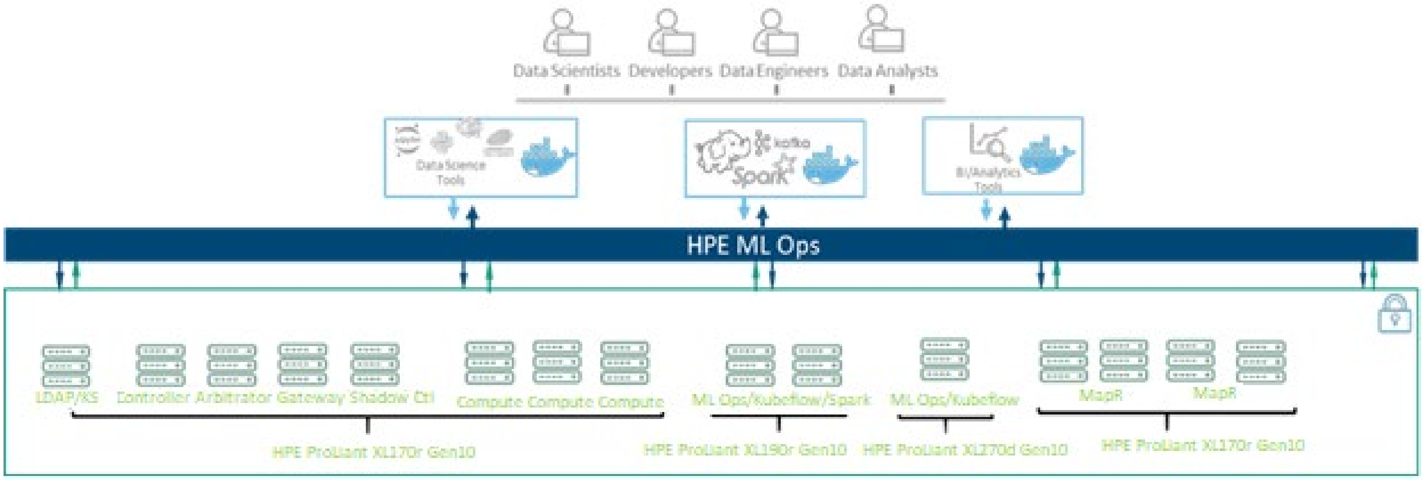


FIGURE 8. HPE Ezmeral ML Ops cluster

A complete list of the hardware components and service configuration is listed in Table 9. This is an example of how an HPE Ezmeral Runtime cluster could be deployed. The number of servers and types of servers will vary based on the workload.

TABLE 9. Hardware details

Qty	Host Service	Server Type	GPU	CPU Cores	Memory	OS Storage	Node Storage	Tenant Storage	K8s/EPIC Containers
		HPE Ezmeral Runtime Cluster							
3	Ezmeral Controller	HPE ProLiant XL225n Gen10 Plus	0	8	128 GB	1.2 TB	1.2 TB	None	N
1	Ezmeral Gateway	HPE ProLiant XL225n Gen10 Plus	0	8	128 GB	1.2 TB	None	None	N
1	K8s Master/ Compute	HPE ProLiant DL325 Gen10 Plus	0	32	256 GB	1.2 TB	1.2 TB	None	Y
2	K8s Compute GPU	HPE ProLiant XL675d Gen10 Plus	4	2 x 32	256 GB	1.2 TB	1.2 TB	None	Y
1	K8s Compute	HPE ProLiant DL385 Gen10 Plus	0	2 x 32	256 GB	1.2 TB	1.2 TB	None	Y

Qty	Host Service	Server Type	CPU Cores	Memory	OS Storage	NFS Storage
	Supporting Services					
1	MIT Kerberos LDAP 389 Directory Server NTP Time Service DNS Name Server NFS Storage	HPE ProLiant DL360 Gen10	24	196 GB	1.2 TB	10 TB

Use case NYC taxi rides

The use case provided demonstrates how HPE ML Ops provides an end-to-end solution for the complete lifecycle to build, train, deploy, and monitor ML and DL models in multitenant enterprise environments. The use case demonstrates the NYC taxi ride prediction using TensorFlow.

The dataset contains a sample of approximately 375,000 NYC taxi rides from January-June 2019. Pickup and drop-off locations are specified as location ID numbers.

Sample data can be found at https://github.com/bluedatainc/solutions/tree/master/MLOps/examples/NYCTaxi/Taxi Datasets.

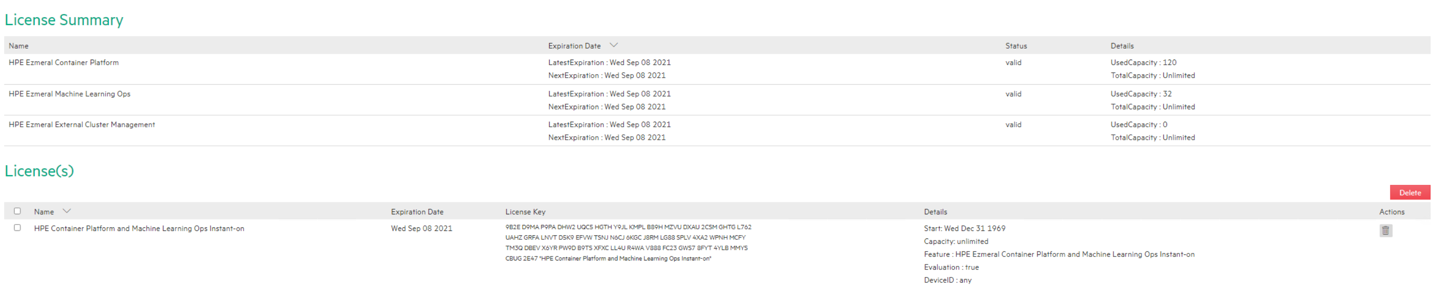
NOTE
The examples and documentation provided in this section are meant to supplement, not replace the manuals.

The AI/ML workflow allows the user to build, train, and deploy a model and then send API requests to that model to make predictions. This workflow consists of three high-level steps that must be performed by users with different roles in the following order:

- Kubernetes Administrator
- LDAP/AD Administrator (For Jupyter Notebook KDapp use)
- Project Administrator
- Project Member (Data Scientist)

Persona: Kubernetes Administrator

[redacted]Verify that HPE Ezmeral Runtime is licensed for at least the number of CPU cores that will be used for the new Kubernetes cluster. HPE Ezmeral ML Ops requires a separate license for each of the CPU cores that will be used in AI/ML projects. Open the system settings screen, and then select the License tab to verify the number of CPU cores licensed for HPE Ezmeral ML Ops as shown in Figure 9.



License Summary

Name	Expiration Date	Status	Details
HPE Ezmeral Container Platform	LatestExpiration : Wed Sep 08 2021 NextExpiration : Wed Sep 08 2021	valid	UsedCapacity : 120 TotalCapacity : Unlimited
HPE Ezmeral Machine Learning Ops	LatestExpiration : Wed Sep 08 2021 NextExpiration : Wed Sep 08 2021	valid	UsedCapacity : 32 TotalCapacity : Unlimited
HPE Ezmeral External Cluster Management	LatestExpiration : Wed Sep 08 2021 NextExpiration : Wed Sep 08 2021	valid	UsedCapacity : 0 TotalCapacity : Unlimited

License(s)

Delete

Name	Expiration Date	License Key	Details	Actions
HPE Container Platform and Machine Learning Ops Instant-on	Wed Sep 08 2021	9B2E D9MA P9PA DHW2 UQC5 HGTH Y9JL KMPL B89H MZVU DXAU 2CSM GHTG L762 UAHZ GRFA LNVT D5K9 EFVW TSNJ N6CJ 6KGC J8RM LG88 SPLV 4XA2 WPNH MCFY TM3Q DBEV X6YR PW9D B9TS XFXC LL4U R4WA V888 FC23 GWS7 8FYT 4YLB MMYS CBUG 2E47 "HPE Container Platform and Machine Learning Ops Instant-on"	Start: Wed Dec 31 1969 Capacity: unlimited Feature : HPE Ezmeral Container Platform and Machine Learning Ops Instant-on Evaluation : true DeviceID : any	

FIGURE 9. License Summary

[redacted]Configure LDAP/AD authentication, If required.

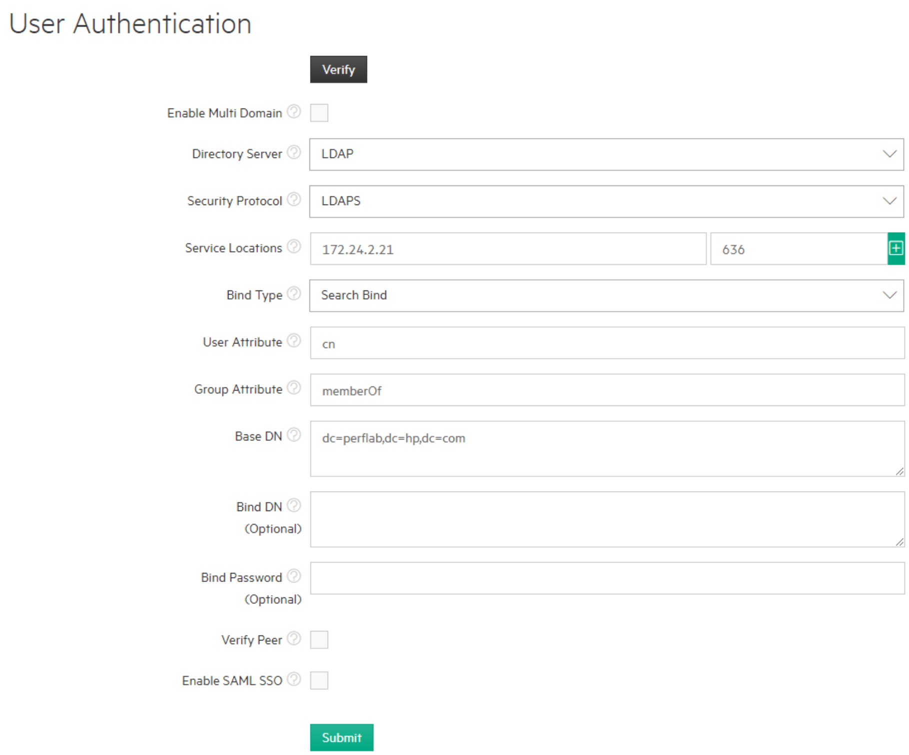


FIGURE 10. User Authentication

[redacted]Create a Kubernetes cluster. Be sure to provide LDAP server information in Step 2: Authentication screen; LDAP must be configured to run HPE Ezmeral ML Ops in a Kubernetes cluster.

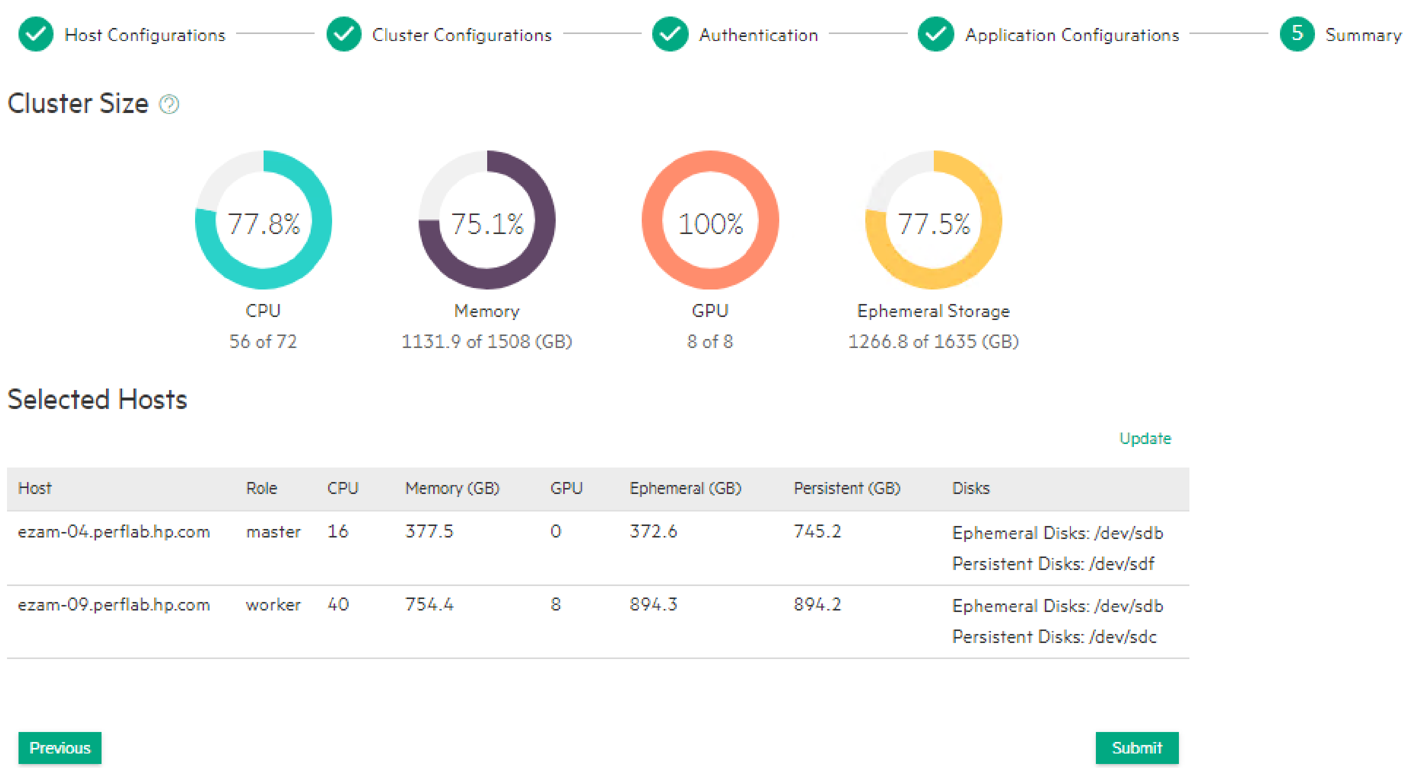


FIGURE 11. Kubernetes cluster creation

[redacted]Assign at least one user to be a Kubernetes Administrator for the Kubernetes cluster just created.

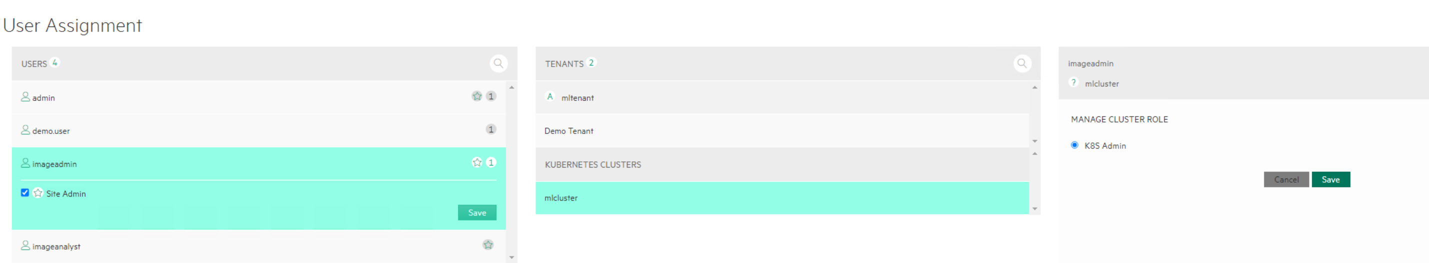


FIGURE 12. Admin Assignment to K8s cluster

[redacted]Execute the following commands on the Kubernetes cluster master hosts to create LDAP/AD secret labels before creating tenants:

```
kubectl config set-context --current --namespace=hpecp
kubectl label secrets hpecp-ext-auth-secret "kubernetes.hpe.com/resource-tenant-visibility"="True"
```

```
[root@ezam-04 ~]# kubectl config set-context --current --namespace=hpecp
Context "kubernetes-admin@k8s-6" modified.
[root@ezam-04 ~]# kubectl label secrets hpecp-ext-auth-secret "kubernetes.hpe.com/resource-tenant-visibility"="True"
error: 'kubernetes.hpe.com/resource-tenant-visibility' already has a value (true), and --overwrite is false
[root@ezam-04 ~]#
```

[redacted]Create a new Kubernetes AI/ML project, being sure to:

a. Check the AI/ML Project checkbox.

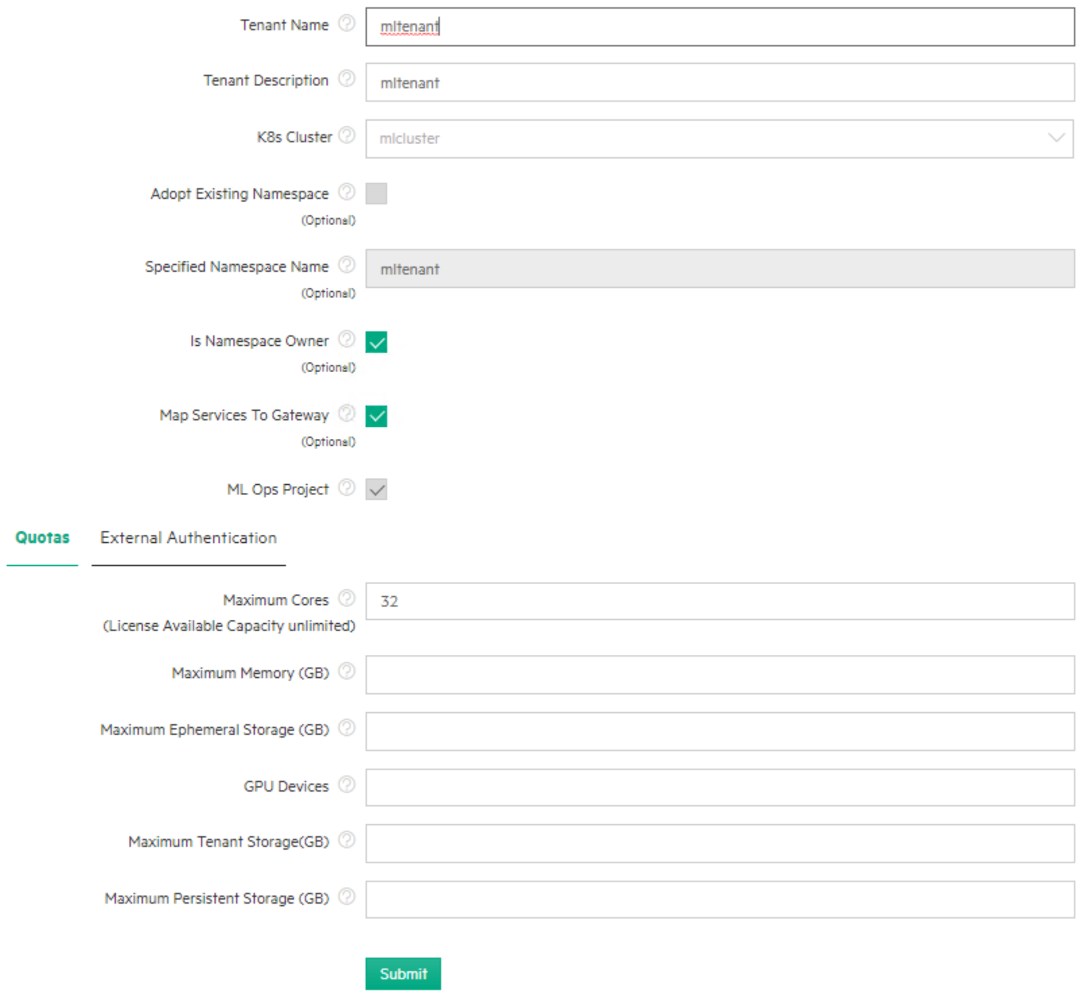


FIGURE 13. Tenant Creation

b. Enter the external LDAP/AD user group in the External Authentication tab.

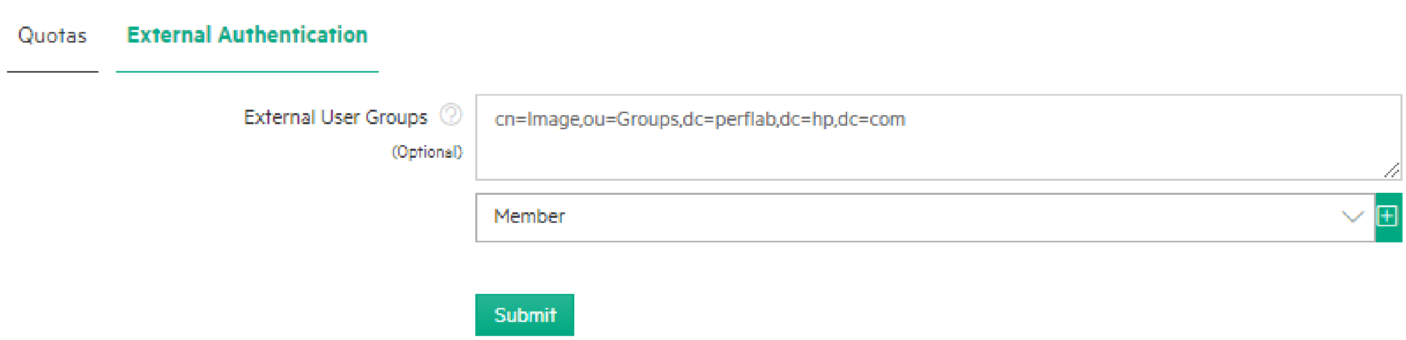


FIGURE 14. External Authentication

[redacted] Assign at least one user to be a Kubernetes Project Administrator for the newly created project.

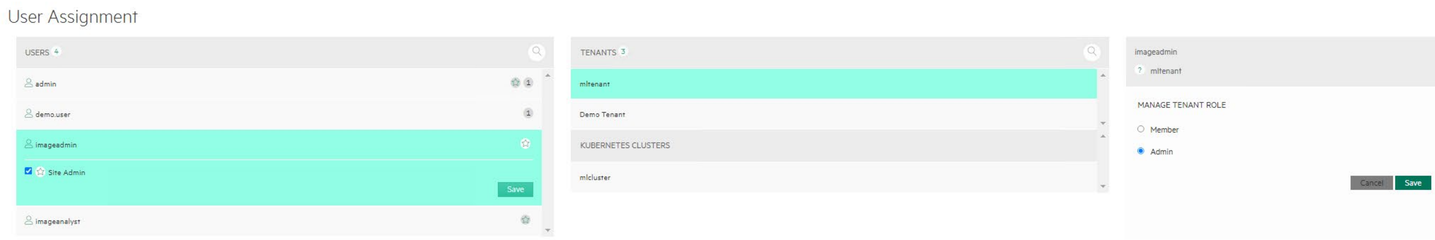


FIGURE 15. Admin Assignment to the tenant

LDAP/AD Administrator (For Jupyter Notebook KDapp use)

If the environment will include the ability to use the Jupyter Notebook KubeDirector application (KDapp), LDAP server group settings must be changed to include all members of the group. For details, see HPE Ezmeral Container Platform 5.3 Documentation.

Persona: Kubernetes project administrator

[redacted] Assign at least one user to the new project.

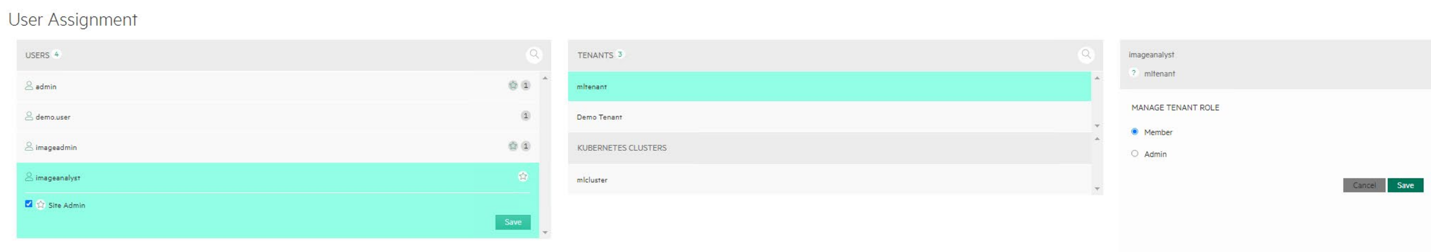


FIGURE 16. Member assignment to the tenant

NOTE

All AI/ML project users (Project Members and Project Administrators) must be LDAP/AD users. They cannot be authenticated using local authentication.

[redacted]Configure one or more global source control configurations.

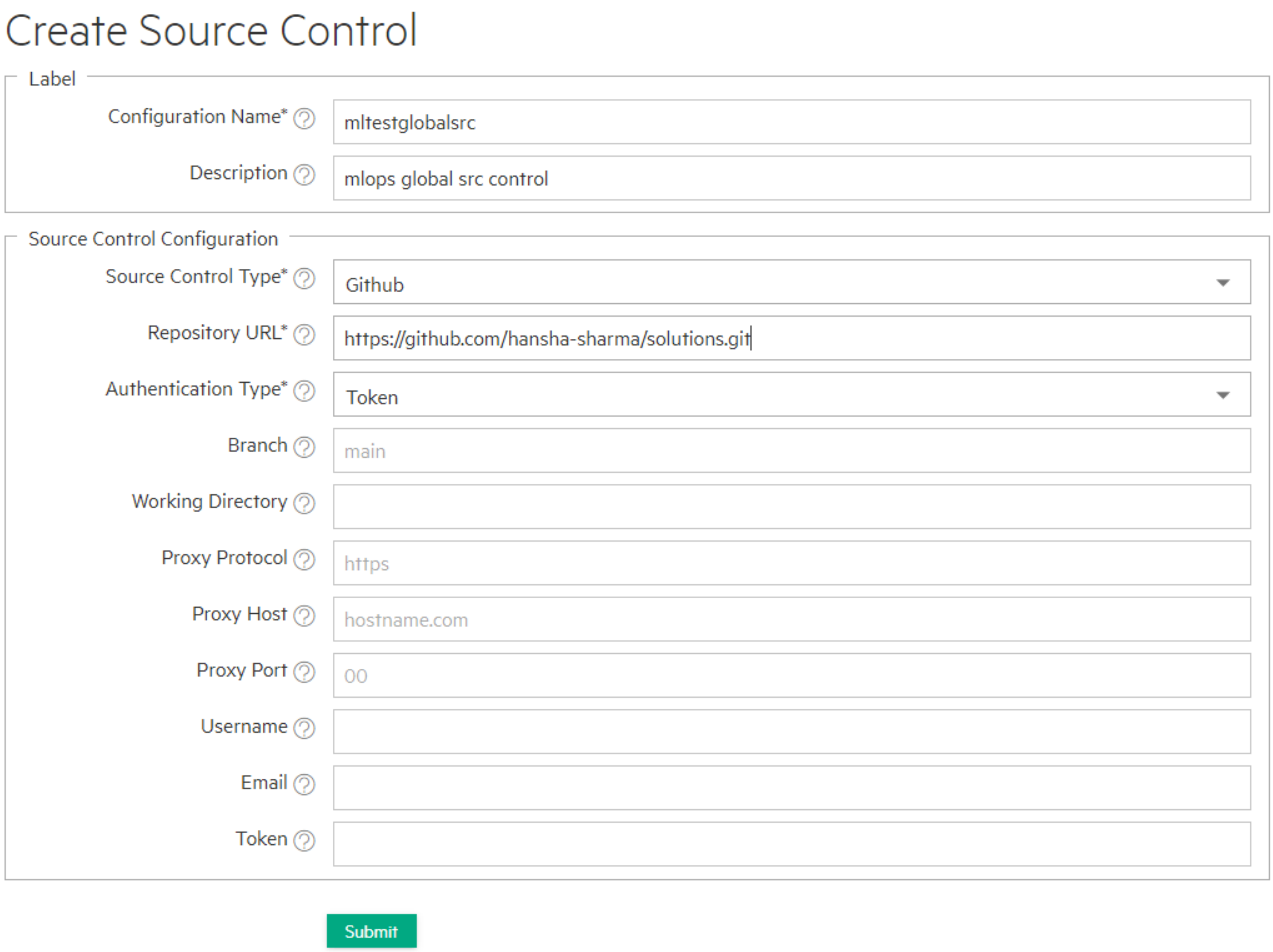


FIGURE 17. Global Source Control Configurations

Persona: Kubernetes project member (Data Scientist)

[redacted]Log out of the web interface, and then log back in as the user that was created or assigned in step 1 of the Kubernetes Administrator workflow described above.

[redacted]Configure at least one individual source control repository. Be sure to copy the name of the secret created for this source control.

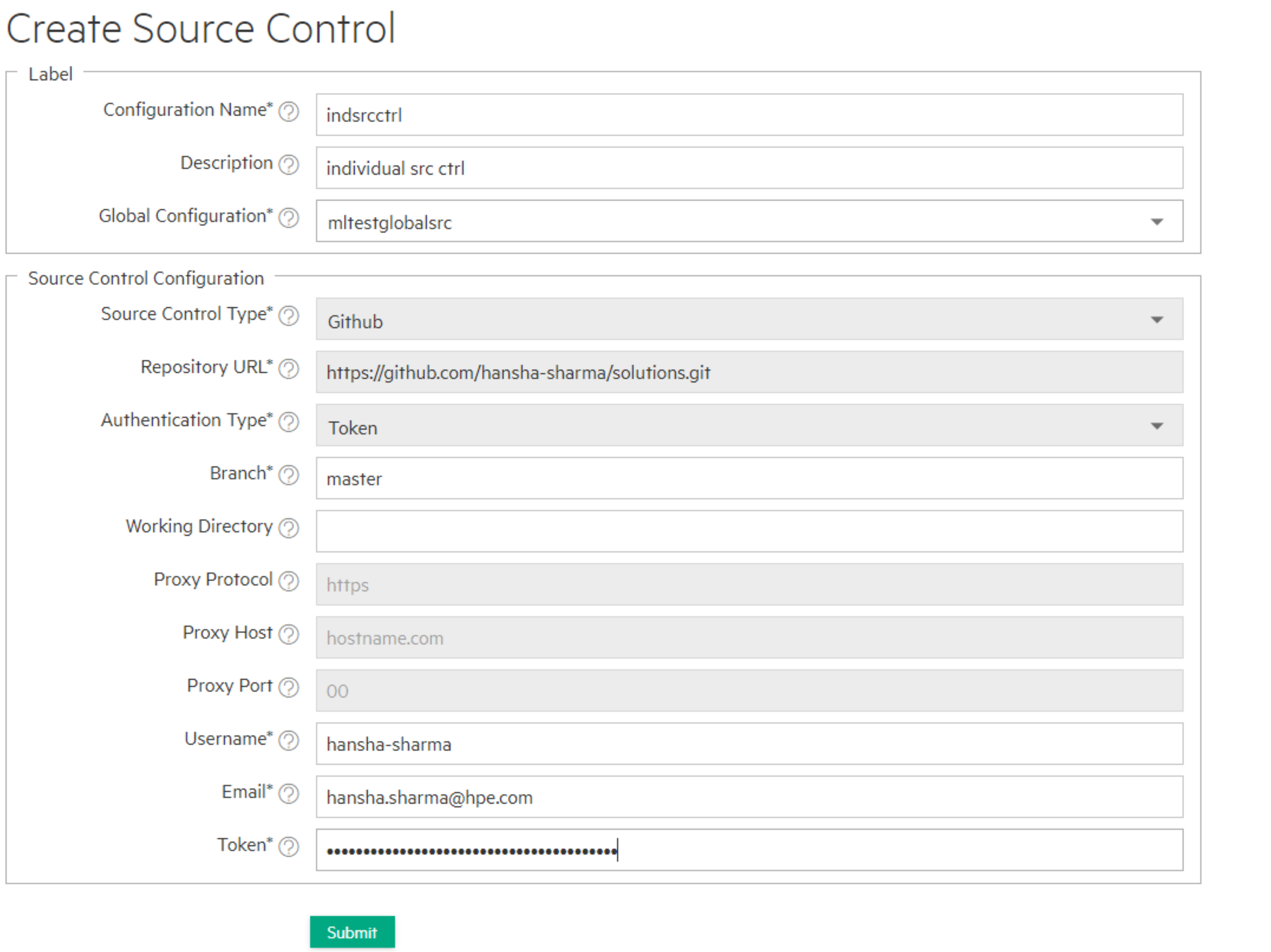


FIGURE 18. Individual Source Control Configuration

[redacted]Access the Kubernetes Training screen, and then onboard the necessary training applications.

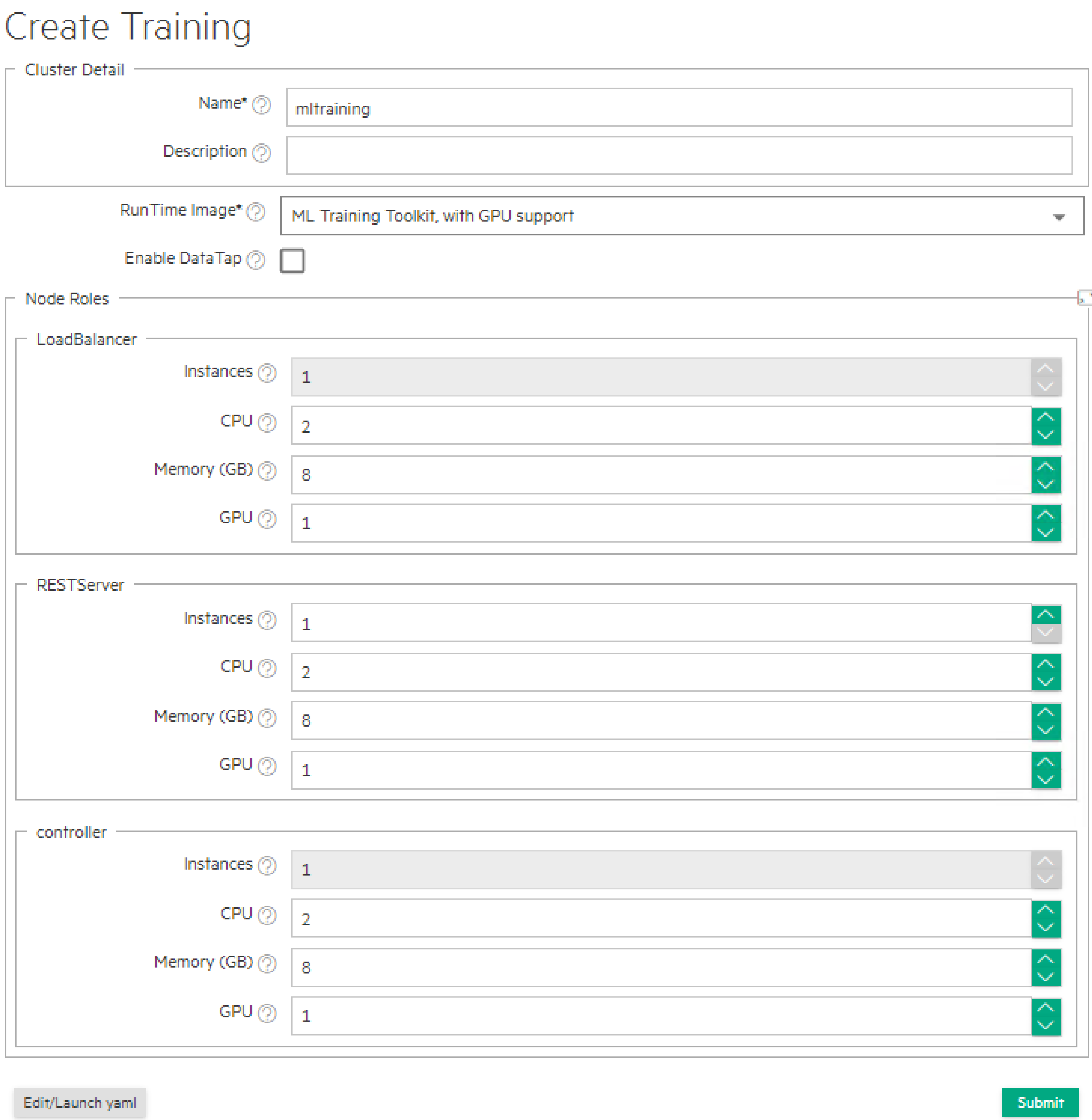


FIGURE 19. Training Cluster Creation

[redacted]Access the Kubernetes Notebooks screen, and then launch the notebook application. Also, attach the training cluster.

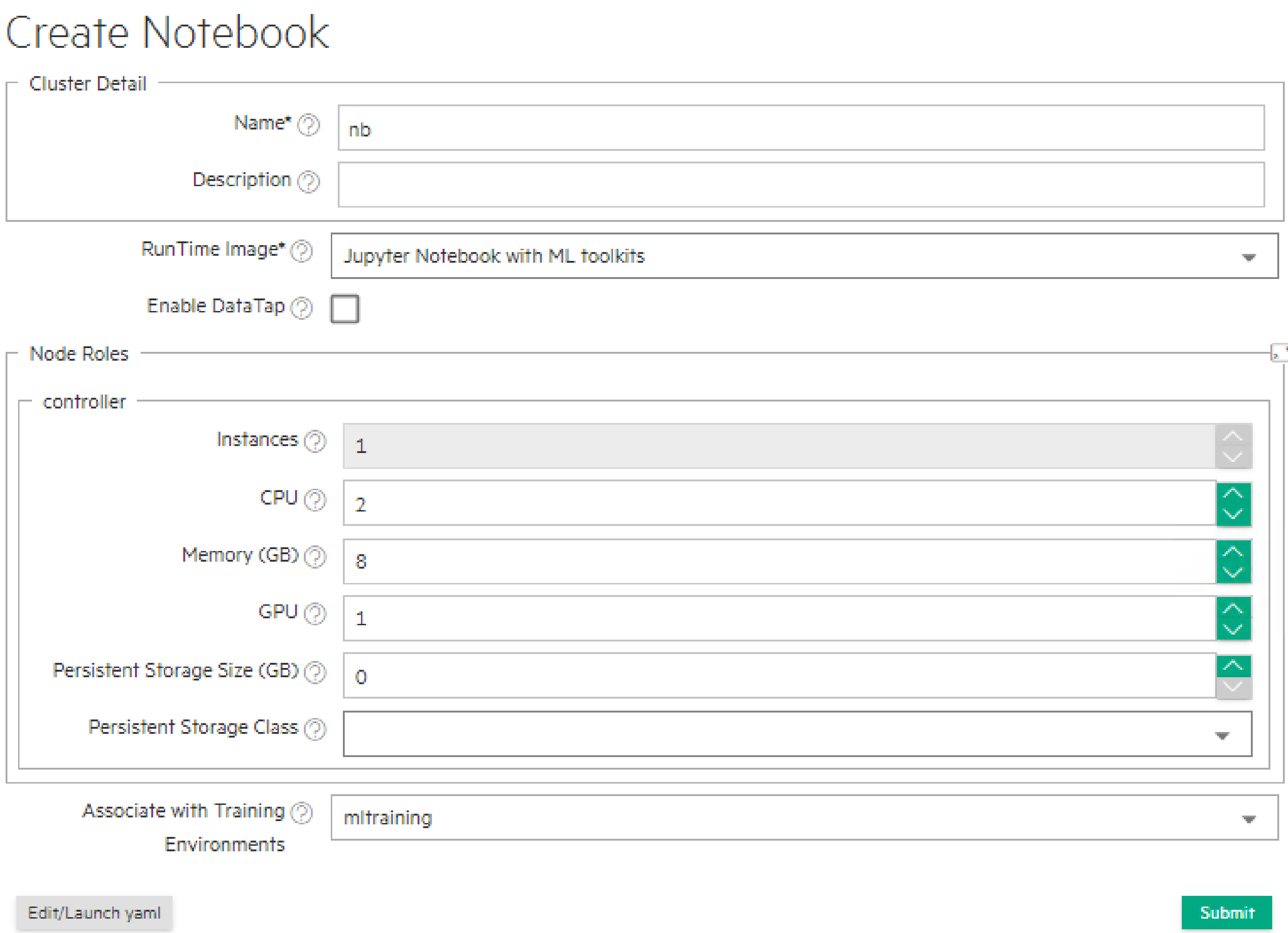


FIGURE 20. Notebook Cluster Creation

[redacted]Once the notebook status appears as configured, it is possible to view the notebook applications and access the application endpoints.

[redacted]Select the Jupyterlab endpoint from the Notebook Endpoints tab.

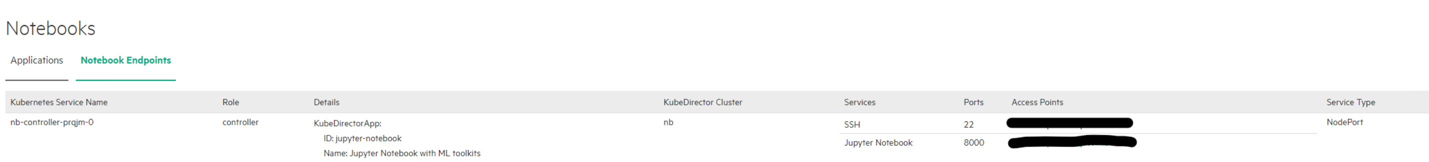


FIGURE 21. Accessing Jupyterlab

[redacted] Log into Jupyterlab using AD/LDAP credentials, and then launch a Python 3 notebook.

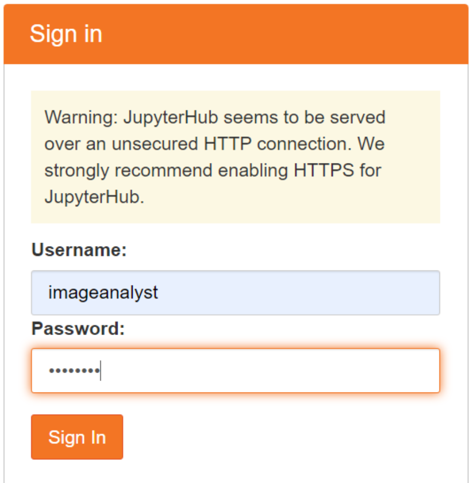


FIGURE 22. Sign in to Jupyterlab

[redacted] Enter the '%kubeRefresh' magic in the Python cell and provide the logged-in member's password when prompted. Kubectl commands can now be entered from the notebook.

```
[6]: %kubeRefresh
      ........
     kubeconfig set for user imageanalyst
```

[redacted] Training dataset to prepare the model. See the HPE Ezmeral Container Platform 5.3 documentation for Notebook Magic Functions.

```
#%%mltraining
print("Importing libraries")
import pandas as pd
import numpy as np
from scipy import stats
import math
import os
import datetime
import xgboost as xgb
import pickle

import matplotlib.pyplot as plt

# Start time
print("Start time: ", datetime.datetime.now())

# Project repo path function
def ProjectRepo(path):
    ProjectRepo = "/bd-fs-mnt/project_repo"
    return str(ProjectRepo + '/' + path)


print("Reading in data")
# Reading in dataset table
dbName = "pqyellowtaxi"
df = pd.read_csv(ProjectRepo('/data/demodata.csv'))

# Reading in latitude/longitude coordinate lookup table
```

```
lookupDbName = "pqlookup"
dflook = pd.read_csv(ProjectRepo('/data/lookup-ipyheader.csv'))
print("Done reading in data")


# merging dataset and lookup tables on latitudes/coordinates
df = pd.merge(df, dflook[[lookupDbName + '.location_i', lookupDbName + '.long', lookupDbName + '.lat']],
how='left', left_on=dbName + '.pulocationid', right_on=lookupDbName + '.location_i')
df.rename(columns = {(lookupDbName + '.long'):(dbName + '.startstationlongitude')}, inplace = True)
df.rename(columns = {(lookupDbName + '.lat'):(dbName + '.startstationlatitude')}, inplace = True)
df = pd.merge(df, dflook[[lookupDbName + '.location_i', lookupDbName + '.long', lookupDbName + '.lat']],
how='left', left_on=dbName + '.dolocationid', right_on=lookupDbName + '.location_i')
df.rename(columns = {(lookupDbName + '.long'):(dbName + '.endstationlongitude')}, inplace = True)
df.rename(columns = {(lookupDbName + '.lat'):(dbName + '.endstationlatitude')}, inplace = True)


def fullName(colName):
  return dbName + '.' + colName

# convert string to datetime
df[fullName('tpep_pickup_datetime')] = pd.to_datetime(df[fullName('tpep_pickup_datetime')])
df[fullName('tpep_dropoff_datetime')] = pd.to_datetime(df[fullName('tpep_dropoff_datetime')])
df[fullName('duration')] = (df[fullName("tpep_dropoff_datetime")] -
df[fullName("tpep_pickup_datetime")]).dt.total_seconds()

# feature engineering
df[fullName("weekday")] = (df[fullName('tpep_pickup_datetime')].dt.dayofweek < 5).astype(float)
df[fullName("hour")] = df[fullName('tpep_pickup_datetime')].dt.hour
df[fullName("work")] = (df[fullName('weekday')] == 1) & (df[fullName("hour")] >= 8) & (df[fullName("hour")] < 18)
df[fullName("month")] = df[fullName('tpep_pickup_datetime')].dt.month
# convert month to a categorical feature using one-hot encoding
df = pd.get_dummies(df, columns=[fullName("month")])

# Filter dataset to rides under 3 hours and under 150 miles to remove outliers
df = df[df[fullName('duration')] > 20]
df = df[df[fullName('duration')] < 10800]
df = df[df[fullName('trip_distance')] > 0]
df = df[df[fullName('trip_distance')] < 150]

# drop null rows
df = df.dropna(how='any',axis=0)

# select columns to be used as features
cols = [fullName('work'), fullName('startstationlatitude'), fullName('startstationlongitude'),
fullName('endstationlatitude'), fullName('endstationlongitude'), fullName('trip_distance'), fullName('weekday'),
fullName('hour')]
cols.extend([fullName('month_' + str(x)) for x in range(1, 7)])
cols.append(fullName('duration'))
dataset = df[cols]


X = dataset.iloc[:, 0:(len(cols) - 1)].values
y = dataset.iloc[:, (len(cols) - 1)].values
X = X.copy()
y = y.copy()
del dataset
del df

print("Done cleaning data")
```

```
print("Training...")

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)


from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

xgbr = xgb.XGBRegressor(objective ='reg:squarederror', colsample_bytree = 1, subsample = 1, learning_rate = 0.15,
booster = "gbtree", max_depth = 15, eta = 0.5, eval_metric = "rmse", tree_method='gpu_hist', gpu_id=0)
print("num train elements: " + str(len(X_train)))
print("Train start time: ", datetime.datetime.now())
xgbr.fit(X_train, y_train)
print("Train end time: ", datetime.datetime.now())
y_pred = xgbr.predict(X_test)
y_pred = y_pred.clip(min=0)


from sklearn import metrics
from sklearn.metrics import mean_squared_log_error
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('Root Mean Squared Log Error:', np.sqrt(mean_squared_log_error( y_test, y_pred)))
print()


print("Saving model")
pickle.dump(xgbr, open( ProjectRepo('models/') + "XGB.pickle.dat", "wb"))


# from xgboost import plot_importance
# plot_importance(xgbr, max_num_features=10) # top 10 most important features
# plt.show()

# Finish time
print("End time: ", datetime.datetime.now())
```

The output looks like this:

```
Importing libraries
Start time:  2021-08-24 16:50:31.977260
Reading in data
Done reading in data
Done cleaning data
Training...
num train elements: 264325
Train start time:  2021-08-24 16:50:35.330399
Train end time:  2021-08-24 16:51:22.312780
Mean Absolute Error: 179.24821523937294
Mean Squared Error: 83067.06130576036
Root Mean Squared Error: 288.21356891333267
Root Mean Squared Log Error: 0.3093430779701812

Saving model
End time:  2021-08-24 16:51:30.038040
```

[redacted] Access the Model Management screen, and then click the Register New Model button to open the Register New Model screen.

[redacted] Register the serialized model, being sure to include the Model Version and Path to Model Repo.

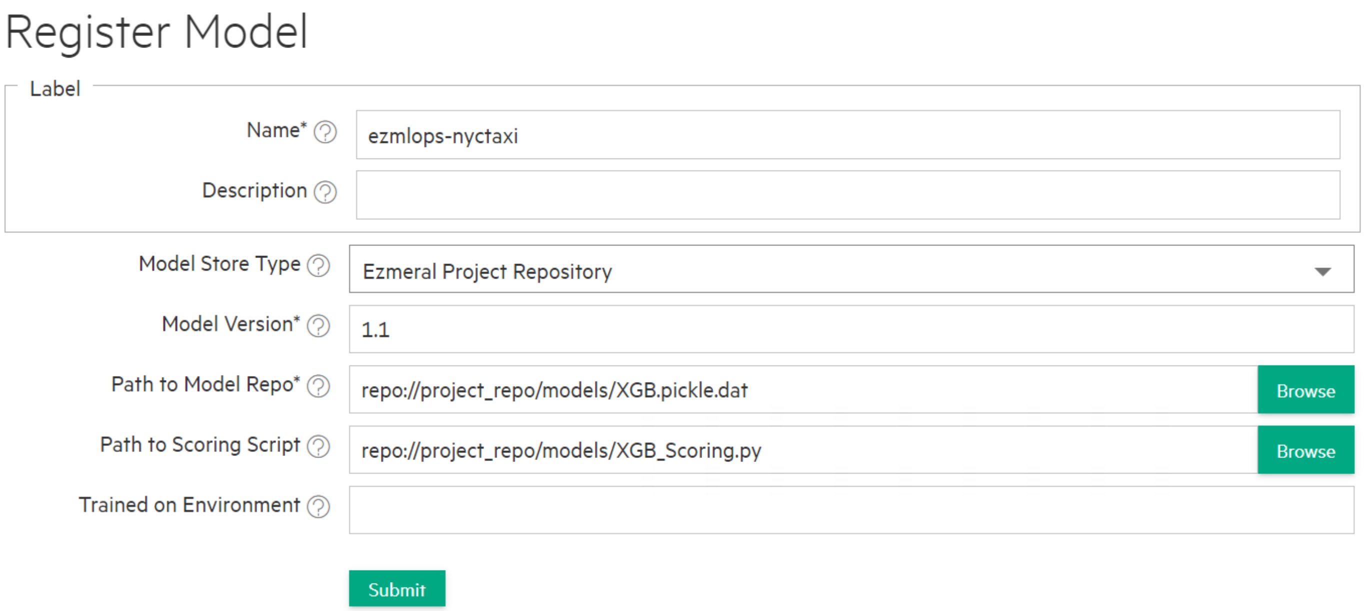


FIGURE 23. Registering Model

[redacted] Access the Kubernetes Model Serving screen and then onboard the necessary training applications.

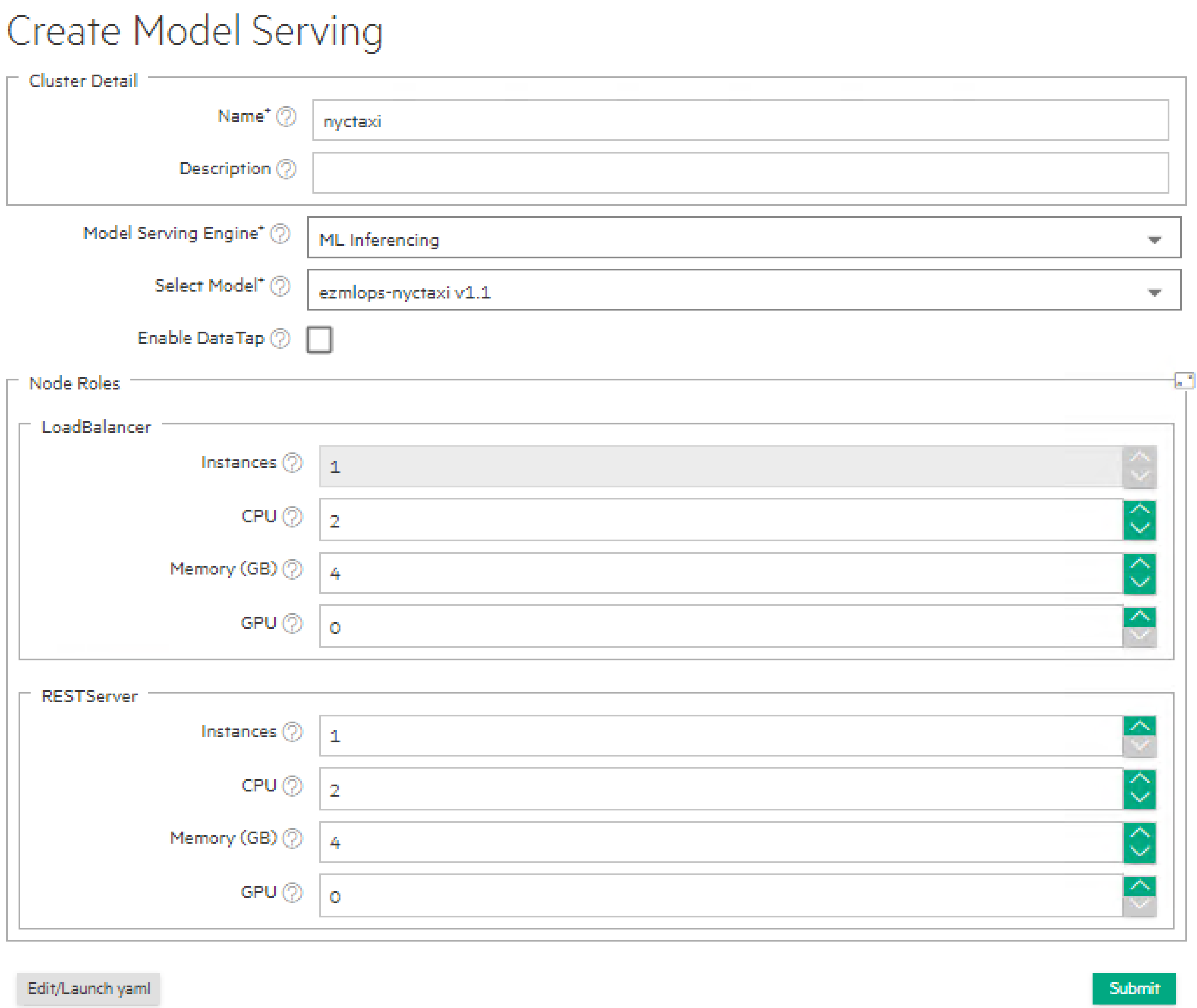


FIGURE 24. Launching Deployment Cluster

Now use the deployed model. To make predictions:

[redacted]Create a Postman API call that is formulated as follows:

a. Prefix: Either http:// or https://, as appropriate.

b. Body: The access point from the ModelServingLoadbalancer of the Load Balancer role in the Ezmeral Serving Endpoints tab.

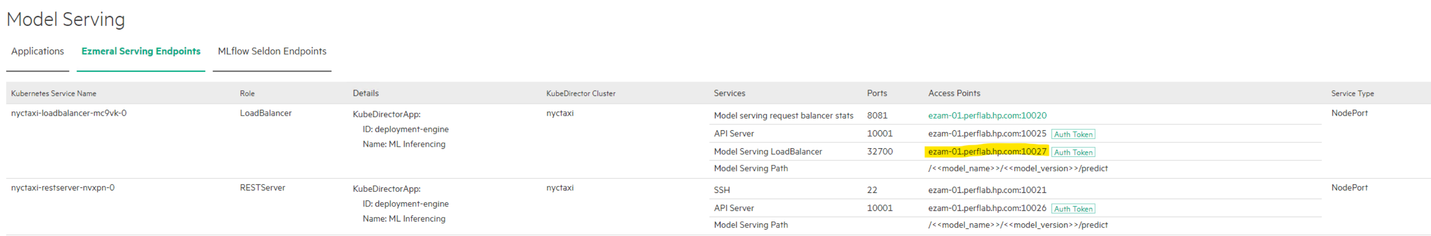


FIGURE 25. Deployment Endpoints

c. Suffix: Registered model name and version number, in the format/model_name_registered/version_number.

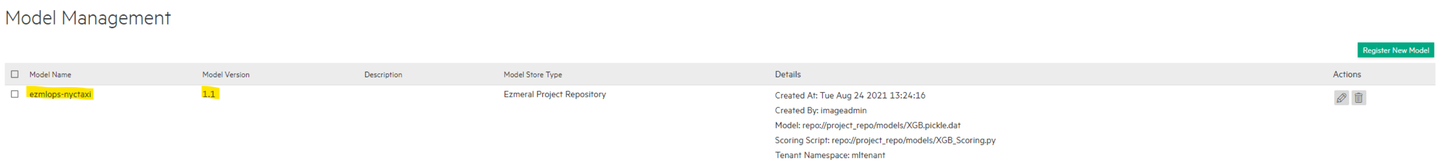


FIGURE 26 Model Management

d. Ending: /predict.

[redacted]Verify that the finished API call looks similar to the example below:

```
m24wn02.bluedata:10038/ezmlops-nyctaxi/1.1/predict
```

[redacted]In the Deployment Endpoints tab, click the Copy Auth Token link for the Load Balancer role.

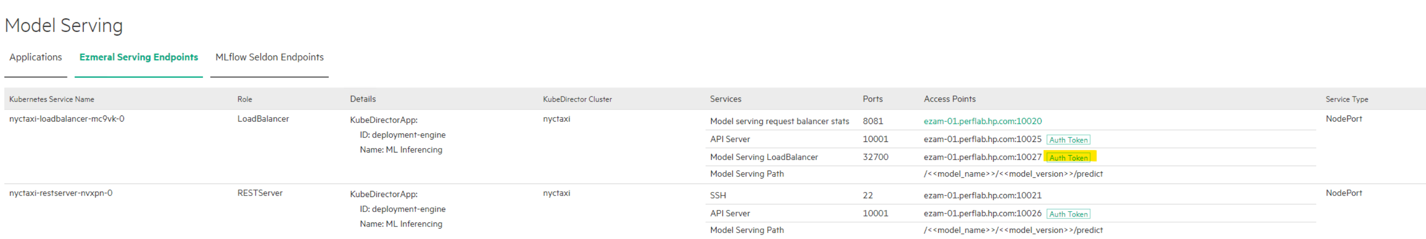


FIGURE 27. Auth Token

[redacted]Launch Postman, and then enter the following information:

a. Request URL: The URL created in Step 2.

b. X-AUTH-TOKEN: Auth token copied in Step 3.

[redacted]Use the raw body data to send queries.

```
{
    "use_scoring": true,
    "scoring_args": {
        "work": 0,
        "start_latitude": 40.57689727,
        "start_longitude": -73.99047356,
```

```
        "end_latitude": 40.72058154,
        "end_longitude": -73.99740673,
        "distance": 8,
        "weekday": 1,
        "hour": 9,
        "month_1": 0,
        "month_2": 1,
        "month_3": 0,
        "month_4": 0,
        "month_5": 0,
        "month_6": 0
    }
}
```

[redacted]Click **Send** to get a prediction.

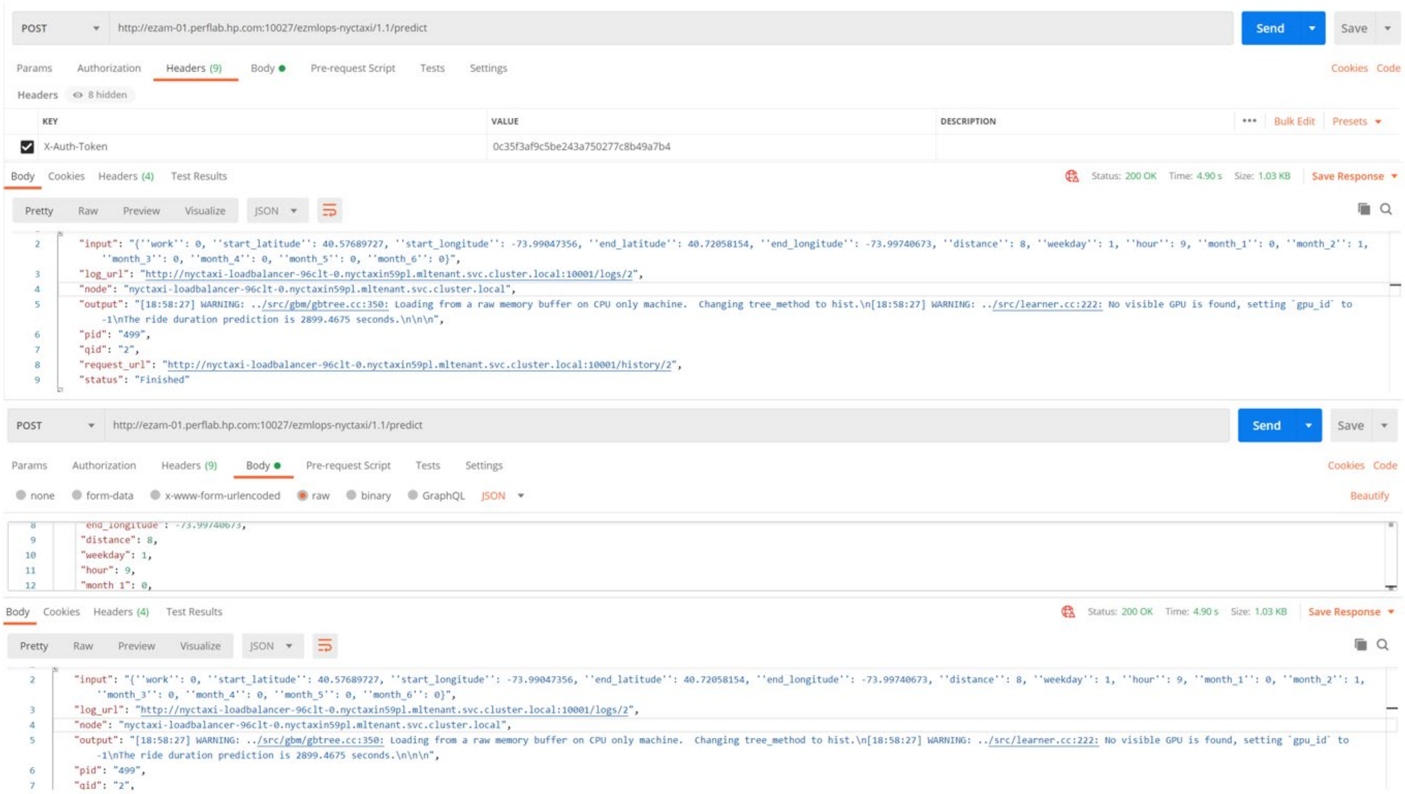


FIGURE 28. Taxi Ride Prediction

See the HPE Ezmeral Container Platform 5.3 documentation for Getting started with AI and ML in Kubernetes for more details.

Use case on Pima Indian's diabetes prediction

This section explains model building, model registry, and model serving for Diabetes prediction using end-to-end ML Ops components on HPE Ezmeral Runtime.

- Dataset: https://www.kaggle.com/uciml/pima-indians-diabetes-database
- Attempting to classify whether or not a patient has diabetes based on some diagnostic measurements
- All patients in the dataset are females at least 21 years old of Pima Indian heritage

Follow the below step-by-step procedure to implement the use case on HPE Ezmeral Runtime.

[redacted]Create an ML Ops tenant.

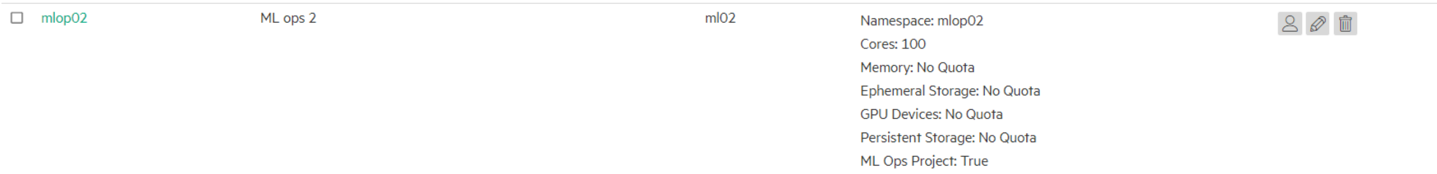


FIGURE 29. ML Ops Tenant

[redacted]Create a training cluster inside the ML Ops tenant.

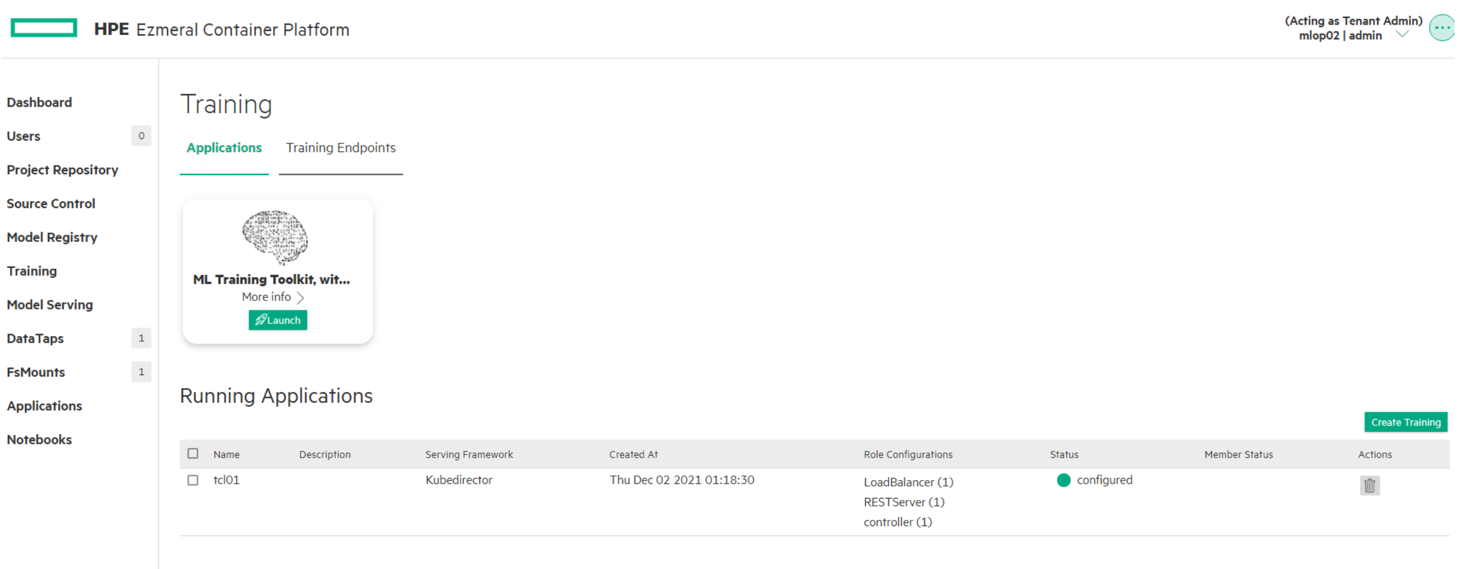


FIGURE 30. Training Cluster

[redacted]Create a notebook and attach a training cluster to it.

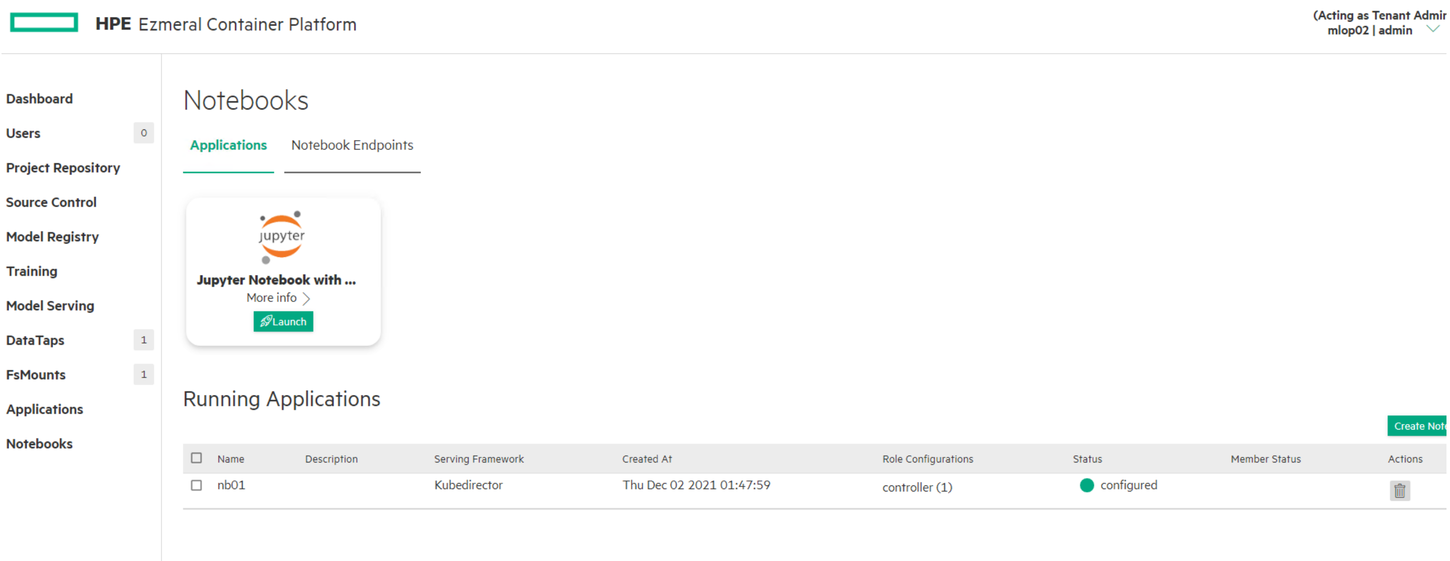


FIGURE 31. Creating notebook

[redacted]Upload dataset to project Repository at project_repo/data/pima_Indians/ and scoring script at project_repo/code/Tensorflow/Diabetes_Scoring.py (can be found in the notebook default examples directory at examples/tensorflow/diabetes_prediction/ Diabetes_Scoring.py).

Also, create a new directory "Diabetes_Prediction" under project_repo/models.

The project_repo structure is as shown below.

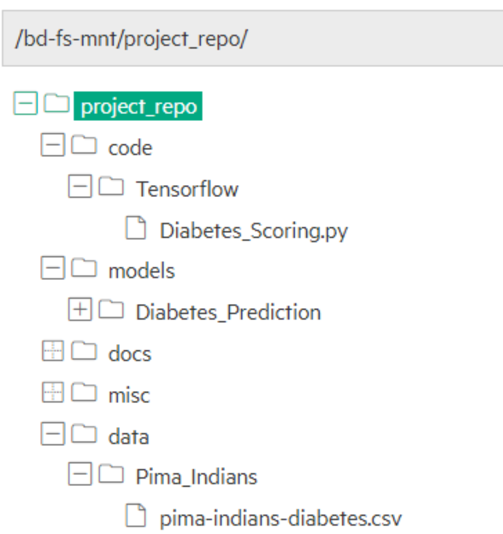


FIGURE 32. Project repo

[redacted]Login to notebook and open the training cluster notebook at /examples/tensorflow/diabetes_prediction/Diabetes_Prediction-k8s.ipynb.

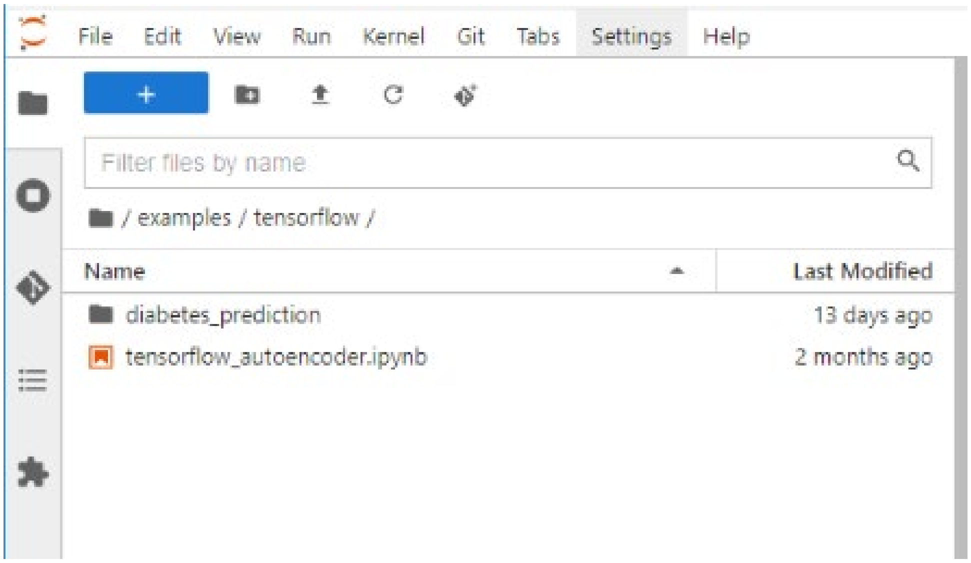


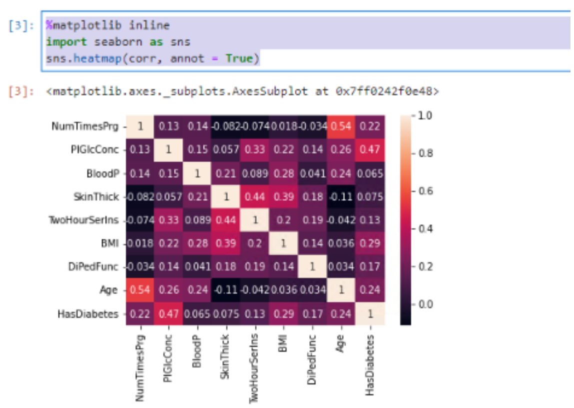
FIGURE 33. Training Cluster notebook

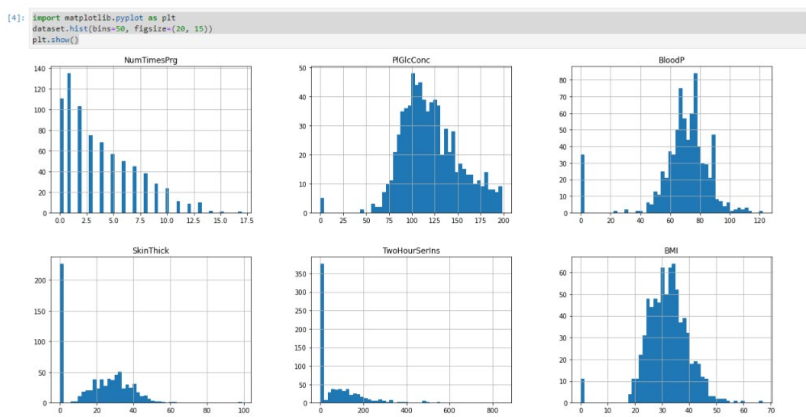
[redacted]Setting up the environment and Data Preprocessing.

```
import numpy
import os
import pandas as pd
import tensorflow as tf

## Set the project repo
def ProjectRepo(path):
    ProjectRepo = '/bd-fs-mnt/project_repo/'#os.popen('bdvcli --get cluster.project_repo').read().rstrip()
#    print(ProjectRepo)
    return ProjectRepo + '/' + path
print(ProjectRepo('data/Pima_Indians/pima-indians-diabetes.csv'))
## Load the dataset
dataset = pd.read_csv(ProjectRepo('data/Pima_Indians/pima-indians-diabetes.csv'), delimiter=",")
dataset.columns = [
    "NumTimesPrg", "PlGlcConc", "BloodP",
    "SkinThick", "TwoHourSerIns", "BMI",
    "DiPedFunc", "Age", "HasDiabetes"]
```

[redacted]Visualizing correlation of variables with a heatmap and plotting histogram.

```
%matplotlib inline
import seaborn as sns
sns.heatmap(corr, annot = True)
import matplotlib.pyplot as plt
dataset.hist(bins=50, figsize=(20, 15))
plt.show()
```





Model development (Part 1)

[redacted]Attempting the first model with XGB.

```
# First XGBoost model for Pima Indians dataset
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pickle

# load data
dataset = loadtxt(ProjectRepo('data/Pima_Indians/pima-indians-diabetes.csv'), delimiter=",",skiprows=1)

# split data into X and y
X_train = dataset[:,0:8]
y_train = dataset[:,8]

# fit model no training data
model = XGBClassifier()
model.fit(X_train, y_train)

#
print(model.get_xgb_params())
```

```
[10:02:23] WARNING: ../src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error'
to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
{'objective': 'binary:logistic', 'base_score': 0.5, 'booster': 'gbtree', 'colsample_bylevel': 1, 'colsample_bynode': 1, 'colsample_bytree': 1, 'gamma': 0, 'gpu_id': -1,
'interaction_constraints': '', 'learning_rate': 0.300000012, 'max_delta_step': 0, 'max_depth': 6, 'min_child_weight': 1, 'monotone_constraints': '()', 'n_jobs': 64, 'nu
m_parallel_tree': 1, 'random_state': 0, 'reg_alpha': 0, 'reg_lambda': 1, 'scale_pos_weight': 1, 'subsample': 1, 'tree_method': 'exact', 'validate_parameters': 1, 'verbo
sity': None}
```

```
[10]: %attachments

Training Cluster    ML Engine
-----------------   -----------
tcl01               python
```

Model development (Part 2)

[redacted]The second model uses Keras with a remote training cluster.

a. Save model and prepare for TensorFlow Serving

```
%%tcl01

import numpy
import os
import pandas as pd
import tensorflow as tf
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense

with tf.device("/device:CPU:0"):

    ## Set the project repo
    def ProjectRepo(path):
        ProjectRepo = '/bd-fs-mnt/project_repo/'#os.popen('bdvcli --get cluster.project_repo').read().rstrip()
        print(ProjectRepo)
        return ProjectRepo + '/' + path

## Load the dataset
    print("Loading data")
    dataset = loadtxt(ProjectRepo('data/Pima_Indians/pima-indians-diabetes.csv'), delimiter=",",skiprows=1)
    dataset.shape

# Split into input (X) and output (y) variables
    X = dataset[:,0:8]
    y = dataset[:,8]

# Define the keras model
    print("Building model")
    model = Sequential()
    model.add(Dense(12, input_dim=8, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

# Compile the keras model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fit the keras model on the dataset
    print("Training model")
    model.fit(X, y, epochs=150, batch_size=10, verbose= 0)

# Evaluate the keras model
    _, accuracy = model.evaluate(X, y, verbose=0)
    print('Accuracy: %.2f' % (accuracy*100))

# Make class predictions with the model
    predictions = model.predict_classes(X)

# Summarize the first 3 cases
    for i in range(3):
        print('%s => %d (expected %d)' % (X[i].tolist(), predictions[i], y[i]))

# Save model weights and architecture together
    print("Saving model")
    model.save(ProjectRepo('models/Diabetes_Prediction/db_remote.h5'))

# Evaluate the keras model
    _, accuracy = model.evaluate(X, y, verbose=0)g
    print('Accuracy: %.2f' % (accuracy*100))

# Make class predictions with the model
```

```
    predictions = model.predict_classes(X)

# Summarize the first 5 cases
    for i in range(5):
        print('%s => %d (expected %d)' % (X[i].tolist(), predictions[i], y[i]))

# Prepare TF Serving
    print("Preparing for TF Serving")
    MODEL_VERSION = 1
    tf.keras.backend.set_learning_phase(0)
    model = tf.keras.models.load_model(ProjectRepo('models/Diabetes_Prediction/db_remote.h5'))
    export_path = ProjectRepo('models/Diabetes_Prediction/' + str(MODEL_VERSION))
    tf.keras.models.save_model(model, export_path)

# Summarize model.
    model.summary()
    print("Done")
```

```
# Summarize model.
    model.summary()
    print("Done")
```

```
History URL: http://tcl01-restserver-c6fgl-0.tcl01ms2q9.mlop02.svc.cluster.local:10001/history/4
```

[redacted]See the logs and Job Status as **Finished**.

```
[24]: %logs --url  http://tcl01-restserver-c6fgl-0.tcl01ms2q9.mlop02.svc.cluster.local:10001/history/4

Job Status: Finished
Loading data
/bd-fs-mnt/project_repo/
Building model
Training model
Accuracy: 75.52
[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0] => 1 (expected 1)
[1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0] => 0 (expected 0)
[8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0] => 1 (expected 1)
Saving model
/bd-fs-mnt/project_repo/
Accuracy: 75.52
[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0] => 1 (expected 1)
[1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0] => 0 (expected 0)
[8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0] => 1 (expected 1)
[1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0] => 0 (expected 0)
[0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0] => 1 (expected 1)
Preparing for TF Serving
/bd-fs-mnt/project_repo/
/bd-fs-mnt/project_repo/
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 12)                108
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 104
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 9
=================================================================
Total params: 221
Trainable params: 221
Non-trainable params: 0
_________________________________________________________________
Done
```

FIGURE 34. Job Status Logs

[redacted]For the model registry, go to tenant UI and register the model.

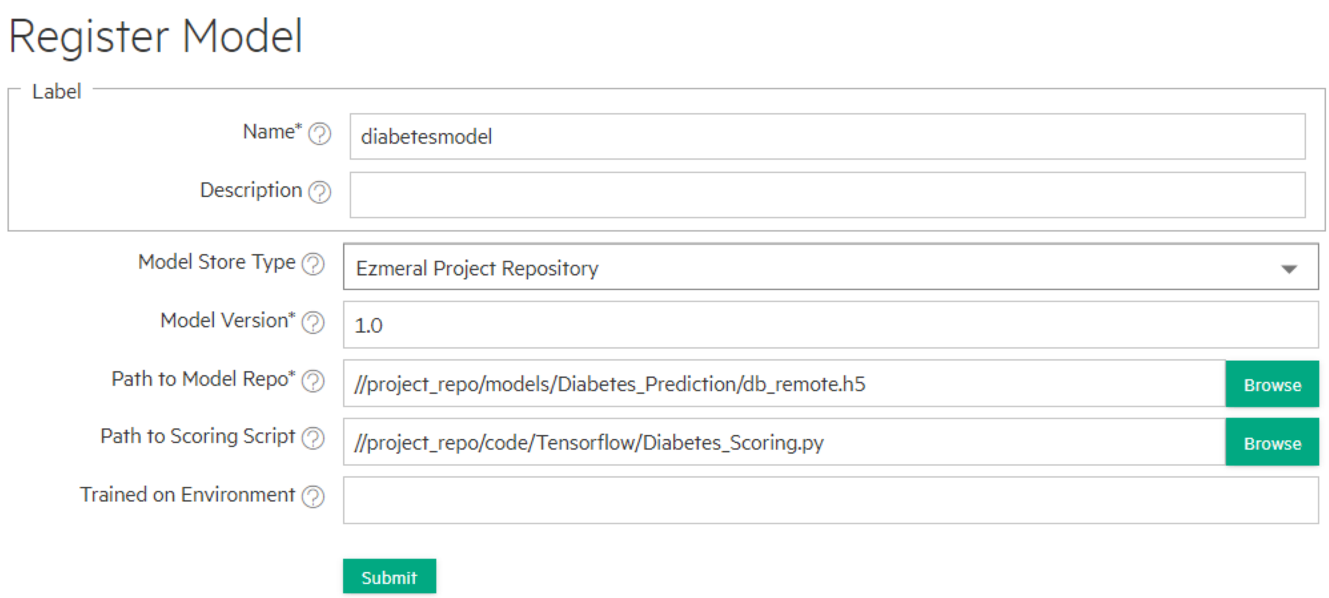


FIGURE 35. Model Registry

[redacted]Create Model Serving.

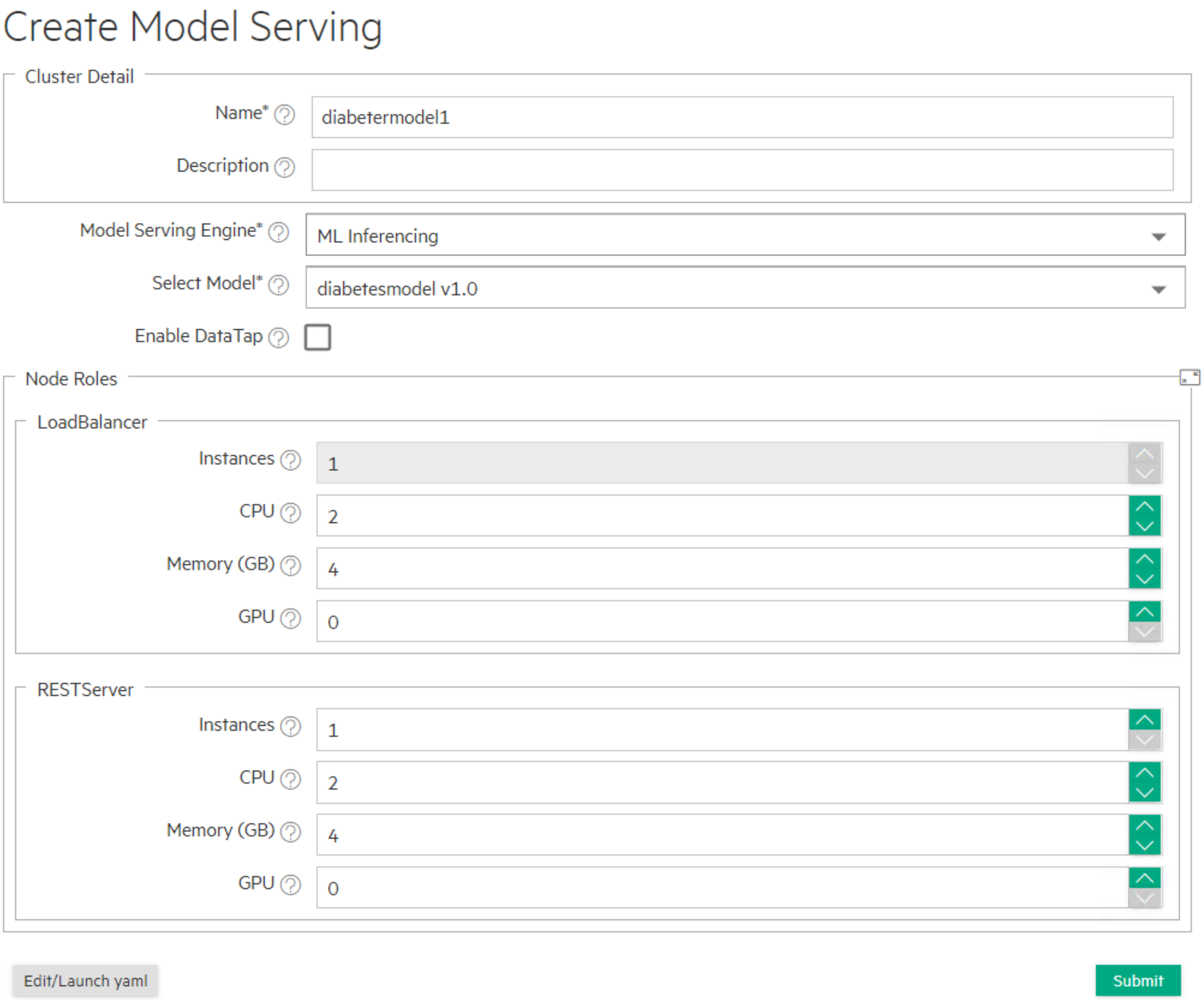


FIGURE 36. Creating Model Serving

[redacted]Once Model Serving is done, obtain the endpoint and Auth token from the Model Serving section.

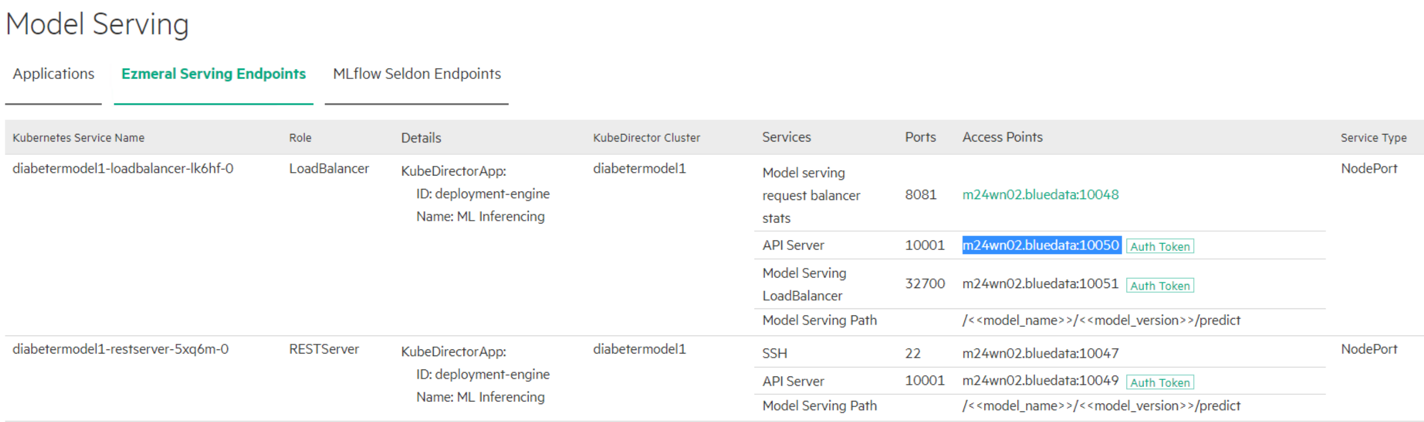


FIGURE 37. Model Serving

[redacted]Prediction can be done by making a POST request via curl or postman to the Model Serving endpoint.

```
curl -X POST -H "Content-Type:application/json" -H "X-Auth-Token:<Auth Token>" -d '{"use_scoring": true,
"scoring_args": {"NumPreg":1.0, "Glucose": 85.0, "BloodPressure": 66.0, "SkinThick": 29.0, "Insulin": 0.0, "BMI":
26.6, "DiabetesPedFunc": 0.351, "Age": 35.0} }' http://<model-serving-
Loadbalancer:port>/<modelname>/<version>/predict

For example : curl -X POST -H "Content-Type:application/json" -H "X-Auth-Token:fb3f6521015978c2f5d58530a42ce720"
-d '{"use_scoring": true, "scoring_args": {"NumPreg":1.0, "Glucose": 85.0, "BloodPressure": 66.0, "SkinThick":
29.0, "Insulin": 0.0, "BMI": 26.6, "DiabetesPedFunc": 0.351, "Age": 35.0} }'
http://m24wn02.bluedata:10050/diabetesmodel/1.0/predict
```

[redacted]This concludes our testing for ML Ops using the Training cluster and KubeDirector Notebook.

EZMERAL ML OPS - IN ACTION WITH USE CASE (SPARK OPERATOR)

Spark operator with K8s

Spark is the cluster computing framework for big data processing. It can also be used for distributed data processing on different machines. It can be used for batch, stream, and interactive data processing. Various file systems can be loaded in Spark. The Spark API is available in Scala, Python, Java, and R. Spark provides libraries such as Spark SQL (for structured data processing), MLlib (for scalable and easy ML), GraphX (for iterative graph computation within a single system), and Spark Streaming (to process real-time data from various sources).

The Kubernetes Operator for Apache Spark included in the HPE Ezmeral Runtime makes running Spark jobs easy. The Spark operator is a custom Kubernetes resource that is configured in a tenant namespace to allow on-demand cluster deployment with Spark Executor pods. For managing Spark jobs, it uses declarative specifications. It dynamically creates the specified number of driver and executor pods during the execution and then deletes the executor pods and leaves the driver pod in the completed state when the job finishes successfully. The driver pod does not consume any Kubernetes resources in this state, and the logs can be viewed to see execution details or results.

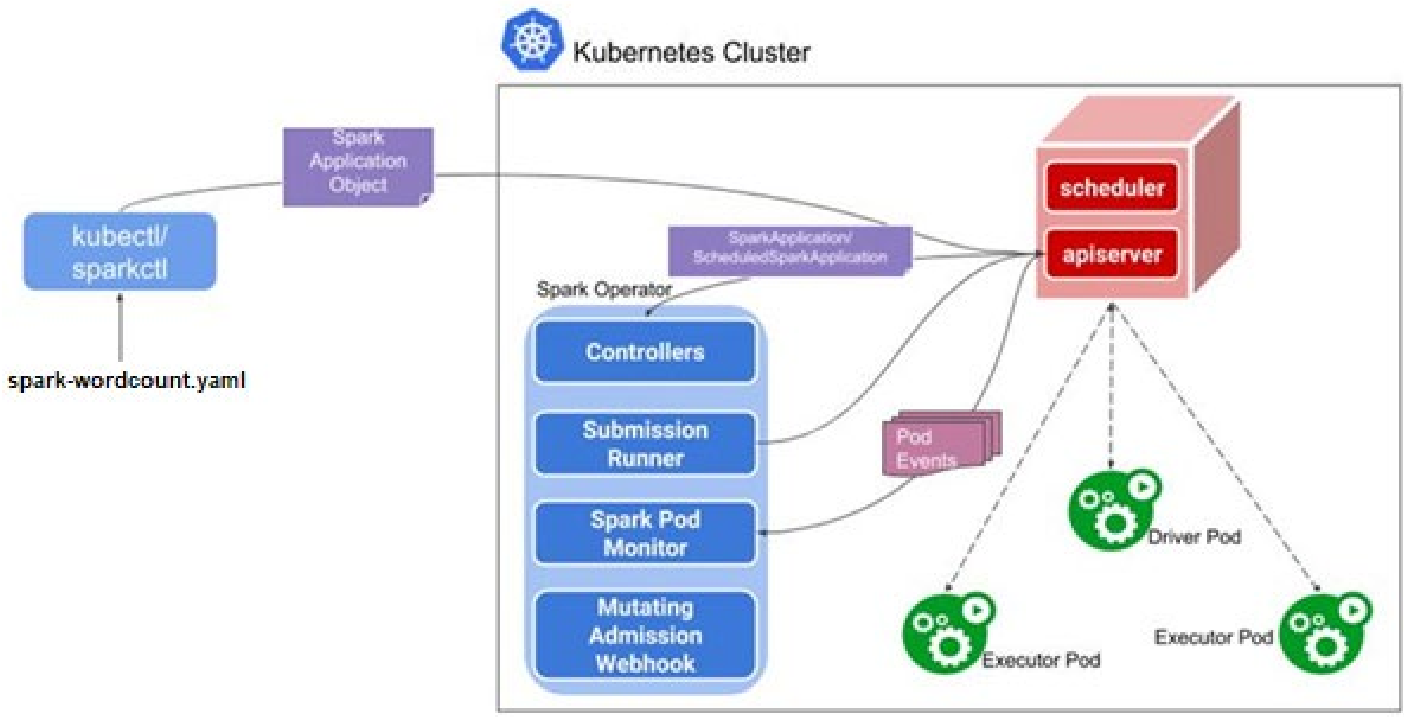


FIGURE 38. Spark operator architecture on K8s

Prerequisites

- HPE Ezmeral Runtime
- Platform administrators can access the web interface
- Root access to the controller host

System requirements

- For supported Kubernetes versions, see Kubernetes Version Requirements.
- For issues and workarounds, see Issues and Workarounds.
- The following resources must be available to install Kubeflow: h
- Minimum number of nodes for compute cluster: 2 (1 primary, 1 secondary)
- Minimum core and memory resources required:
  - CPU Cores: 36
  - Memory (GB): 160

AD/LDAP authentication requirements

- The Kubernetes cluster where Kubeflow will be installed must have AD/LDAP user authentication configured. The AD/LDP user authentication configuration is posted as a secret in the cluster.
- For information about setting AD/LDP user authentication configuration, see Authentication, in Creating a New Kubernetes Cluster.

Preparing the environment

To prepare the environment:

[redacted]Log in to the web interface as a Kubernetes Administrator.

[redacted]Create a Kubernetes cluster, being sure that this cluster meets or exceeds the prerequisites described above.

[redacted]Check the 'Enable Spark Operator'.

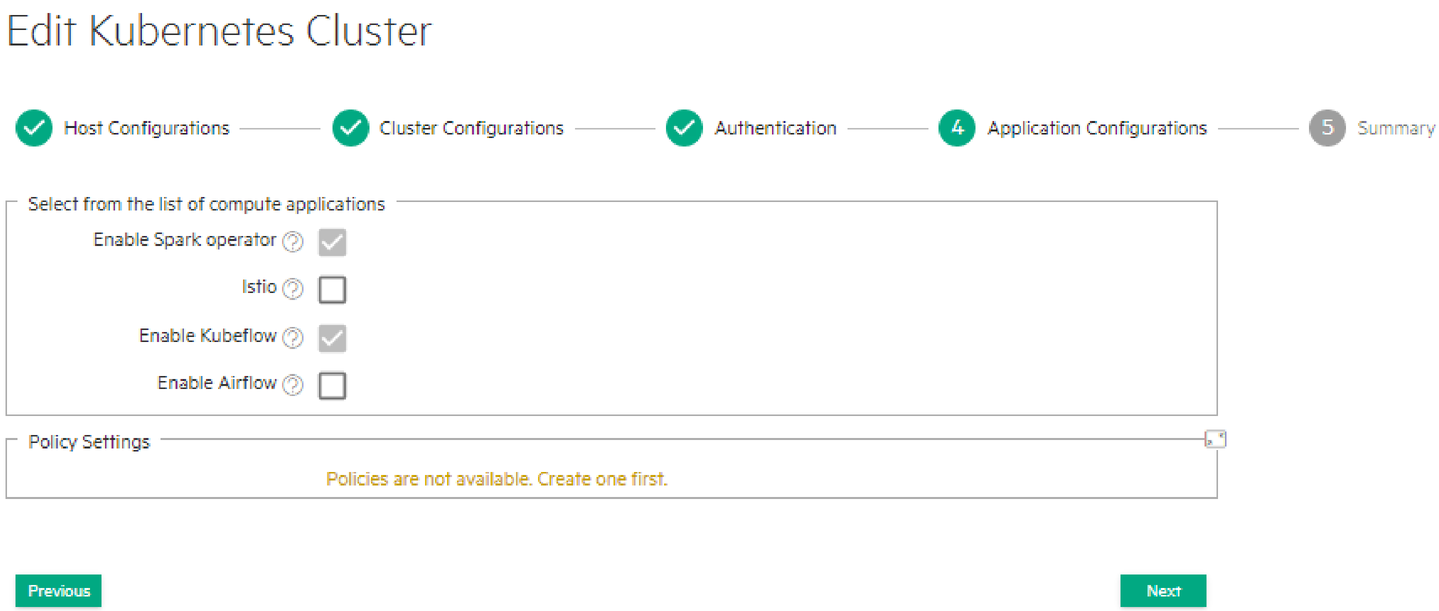


FIGURE 39. Application configurations

[redacted]This bootstraps a Spark operator and creates the following:

a. Integrated HPE Ezmeral Data Fabric Container Storage Interface (CSI) and associated service accounts.

b. Spark namespaces, service accounts, cluster roles, and role bindings, Spark applications Custom Resource Definition (CRD), Spark operator, and compute templates for Spark.

c. Auto Ticket Generator Service, roles, and role bindings.

[redacted]Create a new Kubernetes tenant. Do not assign any quotas when creating this tenant. See Figure 13 for tenant creation.

[redacted]Assign a Kubernetes Cluster Administrator user to the cluster created in Step 2. See Figure 12 for assigning users to the cluster.

[redacted]Assign a Tenant Administrator user to the tenant created in Step 4. See Figure 15 for admin assignment to the tenant.

[redacted]Log out of the web interface when this process is complete.

[redacted]HPE Ezmeral Runtime automatically creates the following:

a. Spark operator namespace.

b. User secrets for the tenant members (necessary for running the Spark workload with the Spark operator).

c. The Role-Based Access Control (RBAC) for Spark resources is also configured for the AD/LDAP tenant members.

Spark operator use case

Following are the steps to run a Spark job using the Kubernetes Web Terminal as a local user.

[redacted]Log in to the web interface as the Site Administrator, and add/assign Tenant users. See Figure 16 for members assigned to the tenant.

[redacted]In the FS Mounts screen, click the TenantShare link in the Name column of the table to open the FS Mount Browser screen. This screen functions identically to the DataTap browser screen.

[redacted]Create the data subdirectory in the TenantShare filesystem mount, and then either create a text file or download this example as wordcount.txt.

[redacted]Upload the wordcount.txt file to the data subdirectory. In HPE Ezmeral Runtime, the location to this subdirectory is /hcp/tenant-<tenant_id>/fsmount/data/wordcount.txt.

```
$/hcp/tenant-14/fsmount/data/wordcount.txt
```

[redacted]Copy the following text, and then save it as `spark-wc.yaml`.

```
apiVersion: "sparkoperator.hpe.com/v1beta2"
kind: SparkApplication
metadata:
  name: spark-wordcount-secure
  namespace: mltenant
spec:
  #sparkConf:
    # Note: If you are executing the application as a K8 user that MapR can verify,
    #       you do not need to specify a spark.mapr.user.secret
    #spark.mapr.user.secret: spark-user-secret
    # Note: You do not need to specify a spark.eventLog.dir
    #       it will be auto-generated with the pattern "maprfs:///apps/spark/<namespace>"
    #spark.eventLog.dir: "maprfs:///apps/spark/sampletenant"
  type: Java
  sparkVersion: 2.4.4
  mode: cluster
  image: gcr.io/mapr-252711/spark-2.4.4:202009090453C
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.JavaWordCount
  mainApplicationFile: "local:///opt/mapr/spark/spark-2.4.4/examples/jars/spark-examples_2.11-2.4.4.6-mapr-
630.jar"
  restartPolicy:
    type: Never
  arguments:
  - maprfs:///hcp/tenant-14/fsmount/data/wordcount.txt
  imagePullSecrets:
  - imagepull
  driver:
    cores: 1
    coreLimit: "1000m"
    memory: "512m"
    labels:
      version: 2.4.4
    # Note: You do not need to specify a serviceAccount
    #       it will be auto-generated referencing the pre-existing "hpe-<namespace>"
    #serviceAccount: hpe-sampletenant
  executor:
    cores: 1
    coreLimit: "1000m"
    instances: 2
    memory: "512m"
    labels:
      version: 2.4.4
```

[redacted]Use the Kubernetes Web Terminal to edit `spark-wordcount.yaml` by updating the namespace and input file name and path for wordcount.txt.

[redacted]Execute the Spark wordcount job by executing the following command:

```
$ kubectl apply -f /bd-fs-mnt/TenantShare/apps/spark-wordcount.yaml -n mltenant
```

```
k8suser@kdss-5zd6r-0:/bd-fs-mnt/TenantShare/apps/spark$ kubectl apply -f spark-wordcount.yaml -n mltenant
sparkapplication.sparkoperator.hpe.com/spark-wordcount-secure created
```

Check the pods running within the tenant namespace by executing the following command:

`$ kubectl get pods -n mltenant`

```
k8suser@kdss-5zd6r-0:/bd-fs-mnt/TenantShare/apps/spark$ kubectl get pods -n mltenant
NAME                              READY   STATUS      RESTARTS   AGE
nbone-controller-t9vtj-0          1/1     Running     0          4h13m
nyctaxi-loadbalancer-mc9vk-0      1/1     Running     0          3h50m
nyctaxi-restserver-nvxpn-0        1/1     Running     0          3h50m
spark-pi-secure-driver            0/1     Completed   0          36m
spark-wordcount-secure-driver     0/1     Completed   0          97s
sparkhs-5c5bb8c6b9-4wqsz          1/1     Running     4          25h
tenantcli-0                       1/1     Running     0          25h
```

[redacted] Check the job status of the job by executing the following command:

`$ kubectl logs spark-wordcount-secure-driver -follow`

```
programs: 2
watch: 1
Wikipedia: 1
were: 3
length: 5
and: 23
subgenre: 1
being: 1
measures: 1
21/08/24 22:14:58 INFO server.AbstractConnector: Stopped Spark@5dbf5634{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
21/08/24 22:14:58 INFO ui.SparkUI: Stopped Spark web UI at http://spark-wordcount-secure-1629843256056-driver-svc.mltenant.svc:4040
21/08/24 22:14:58 INFO k8s.KubernetesClusterSchedulerBackend: Shutting down all executors
21/08/24 22:14:58 INFO k8s.KubernetesClusterSchedulerBackend$KubernetesDriverEndpoint: Asking each executor to shut down
21/08/24 22:14:58 WARN k8s.ExecutorPodsWatchSnapshotSource: Kubernetes client has been closed (this is expected if the application is shutting down.)
21/08/24 22:14:58 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
21/08/24 22:14:58 INFO memory.MemoryStore: MemoryStore cleared
21/08/24 22:14:58 INFO storage.BlockManager: BlockManager stopped
21/08/24 22:14:58 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
21/08/24 22:14:58 INFO scheduler.OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
21/08/24 22:14:58 INFO spark.SparkContext: Successfully stopped SparkContext
```

[redacted] Validate that the job has been completed by checking the status of the pod:

`$ kubectl get pods -n mltenant`

Model training from KubeDirector Notebook using Spark with Livy

Following are the steps to run a Spark job with Livy operator REST API from KubeDirector Notebook in the Kubernetes by AD/LDAP user.

[redacted] Log in to the web interface as the Site Administrator, add/assign Tenant users. See Figure 16 for members assigned to the tenant.

[redacted] In the DataTaps screen, click the TenantStorage link in the Name column of the table to open the DataTap Browser screen.

[redacted] Create the data subdirectory in the TenantStorage filesystem, and then upload or download it as `train.csv`. Refer to this example train.csv.

[redacted]Upload the train.csv file to the data subdirectory. In HPE Ezmeral Runtime, the location to this subdirectory is /hcp/tenant-<tenant_id>/data/train.csv.

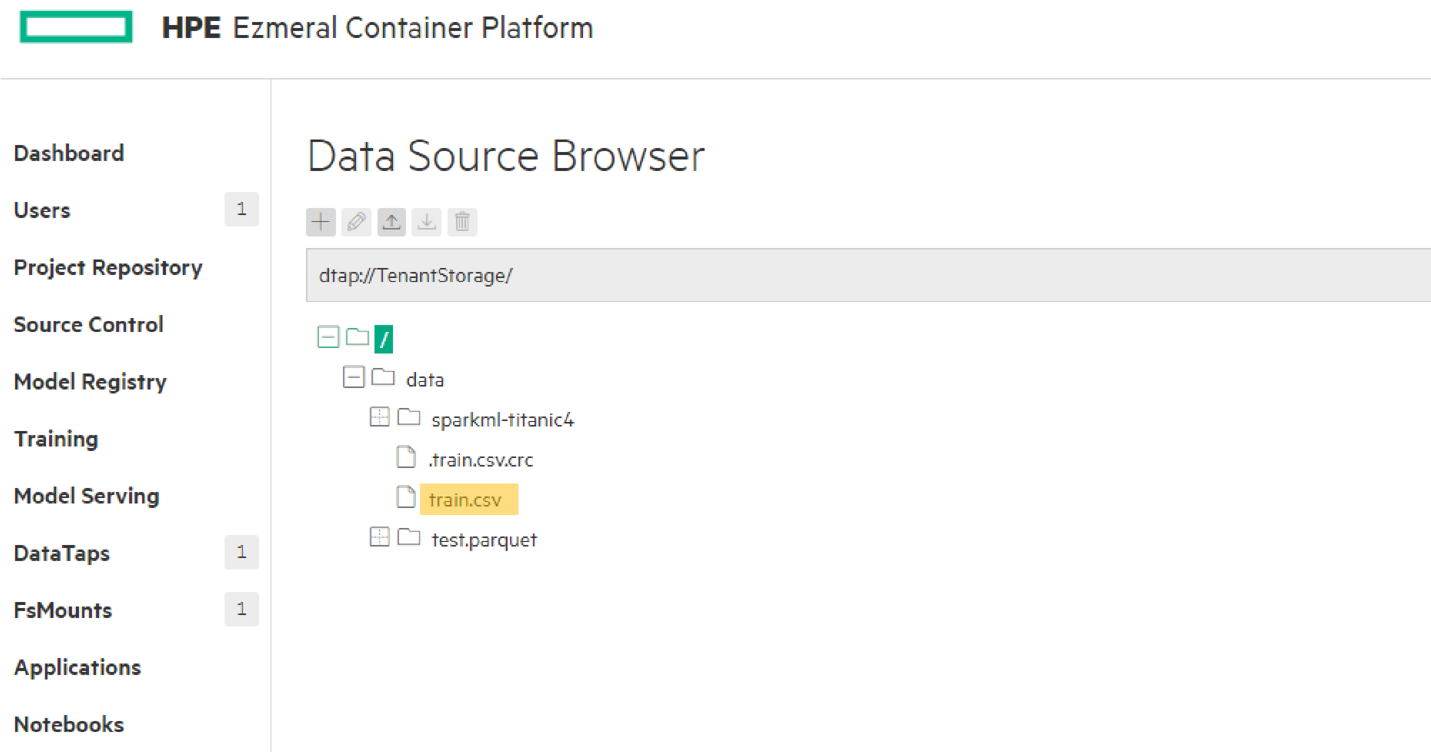


FIGURE 40. DataTaps

[redacted]Fetch Livy endpoint.

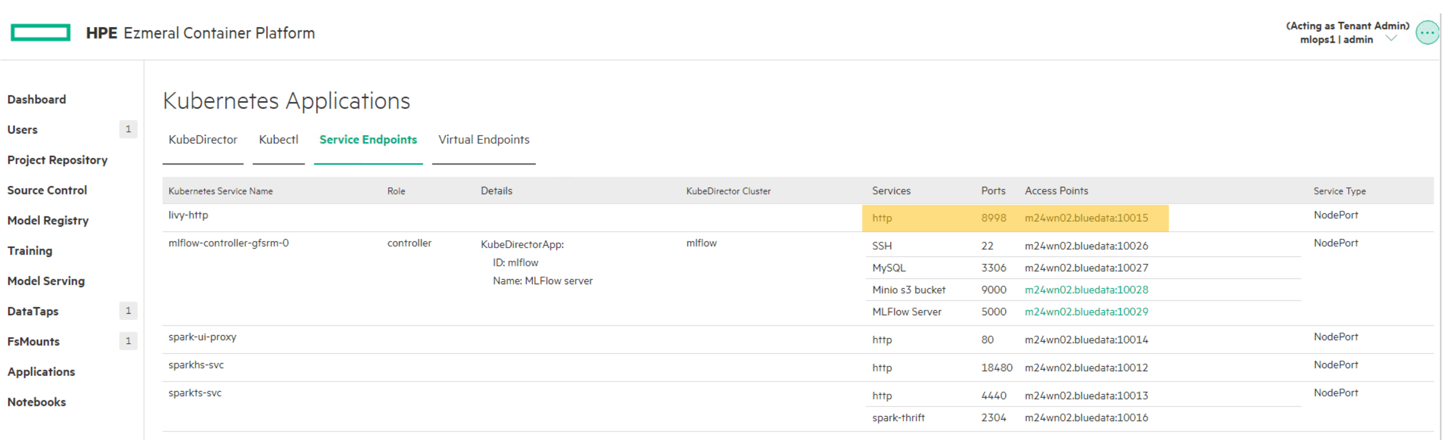


FIGURE 41. Service Endpoints

[redacted]Launch KubeDirector Notebook.

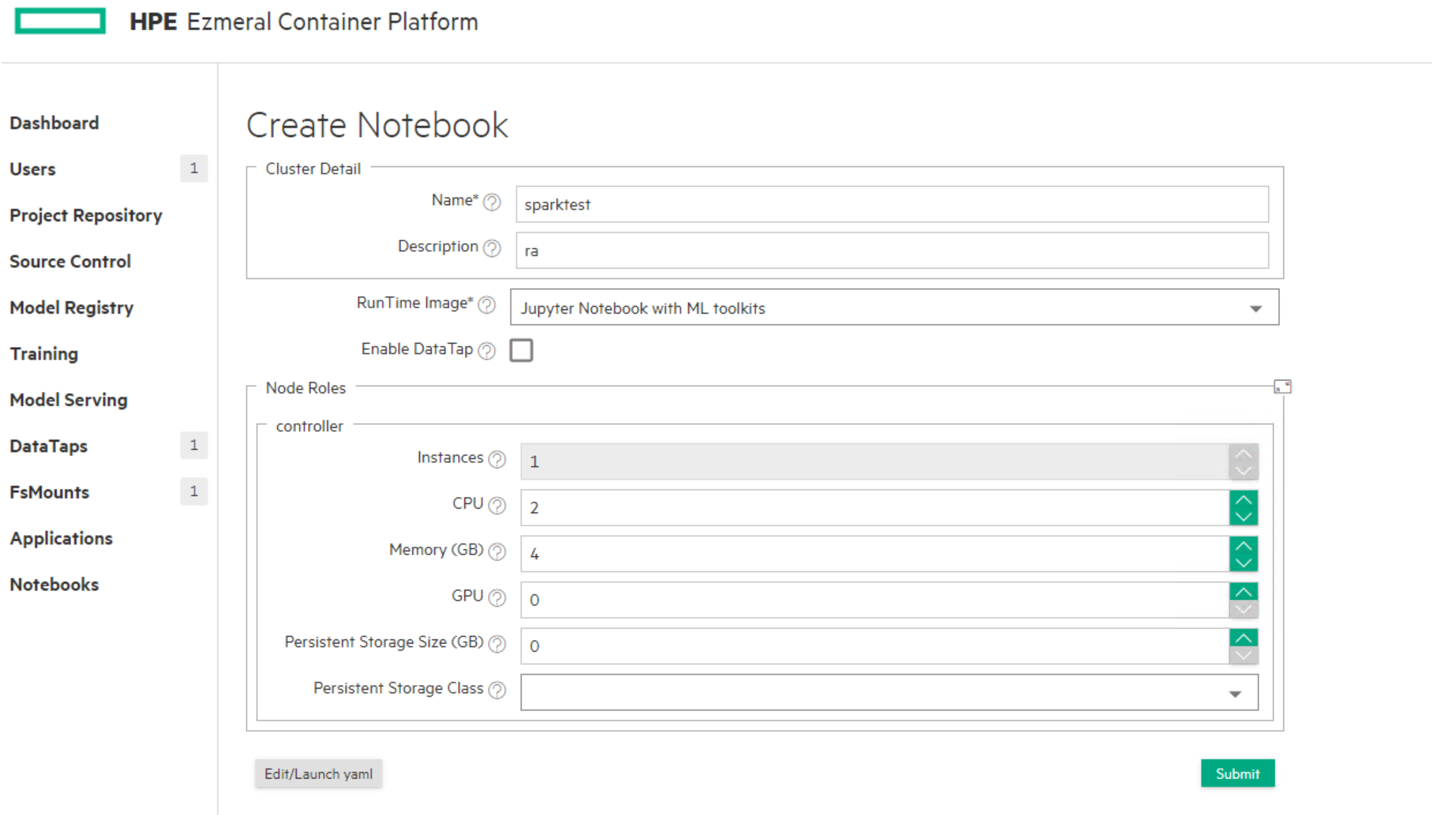


FIGURE 42. Creating Notebook

[redacted]Access JupyterHub Notebook using the service endpoint.

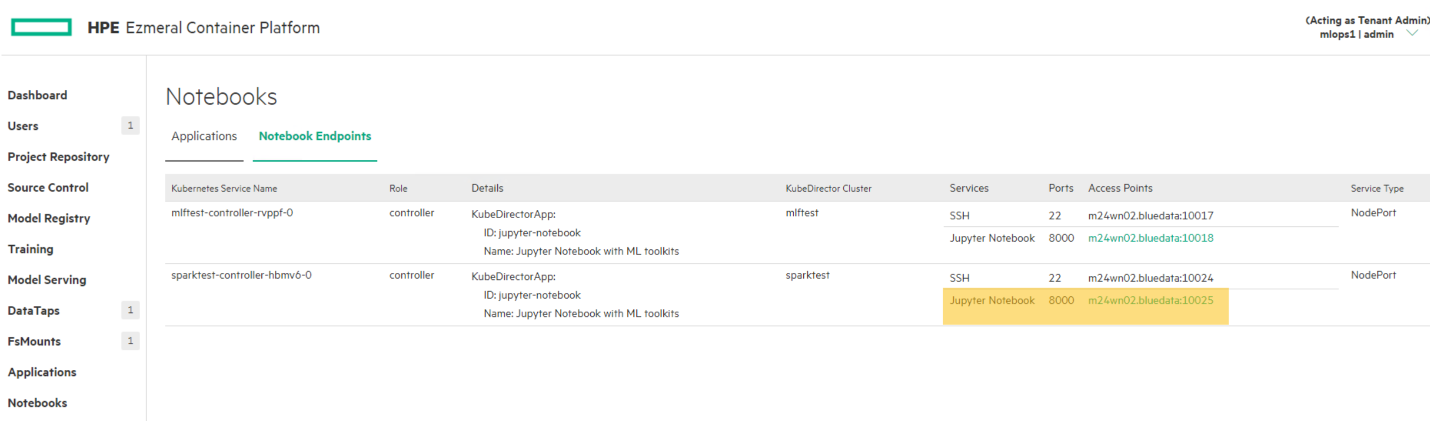


FIGURE 43. Notebook Endpoints

[redacted] Launch PySpark kernel to configure and create Livy session.

Configure the Spark Livy Session Context

Modify Spark parameters for the session In our examples we are including the DTAP Libraries and we specify and image that we want to use for our livy sessions using the spark cluster.

```
[1]: %%configure -f

{"driverCores": 1, "executorCores":2,
"conf":{"spark.kubernetes.container.image":"gcr.io/mapr-252711/spark-2.4.7:202106220630P141", "spark.hadoop.fs.dtap.impl":"com.bluedata.hadoop.bdfs.Bdfs", "spark.hadoop.fs.AbstractFileSystem.dtap.impl":"co
```

Current session configs: {'driverCores': 1, 'executorCores': 2, 'conf': {'spark.kubernetes.container.image': 'gcr.io/mapr-252711/spark-2.4.7:202106220630P141', 'spark.hadoop.fs.dtap.impl': 'com.bluedata.hadoop.bdfs.Bdfs', 'spark.hadoop.fs.AbstractFileSystem.dtap.impl': 'com.bluedata.hadoop.bdfs.BdAbstractFS', 'spark.hadoop.fs.dtap.impl.disable.cache': 'false', 'spark.driver.extraClassPath': 'local:///opt/bdfs/bluedata-dtap.jar', 'spark.executor.extraClassPath': 'local:///opt/bdfs/bluedata-dtap.jar', 'spark.kubernetes.driver.label.hpecp.hpe.com/dtap': 'hadoop2', 'spark.kubernetes.executor.label.hpecp.hpe.com/dtap': 'hadoop2'}, 'kind': 'pyspark'}
No active sessions.

Create a Spark Context

Getting a Spark Context can take a few seconds as the livy containers are started in the background. The Startup Time depends on the speed between the k8s nodes and the image repository

```
[*]: print("hi")

Enter Livy endpoint (e.g., http://<internal-livy-session-url>:<port>) : https://172.30.226.52:10015
Enter your password: ••••••••
```

Create a Spark Context

Getting a Spark Context can take a few seconds as the livy containers are started in the background. The Startup Time depends on the speed between the k8s nodes and the image repository

```
[2]: print("hi")

Enter Livy endpoint (e.g., http://<internal-livy-session-url>:<port>) : https://172.30.226.52:10015
Enter your password: ········
WARNING: Restart the kernel if you entered something wrong or if the kernel fails.
Starting Spark application
```

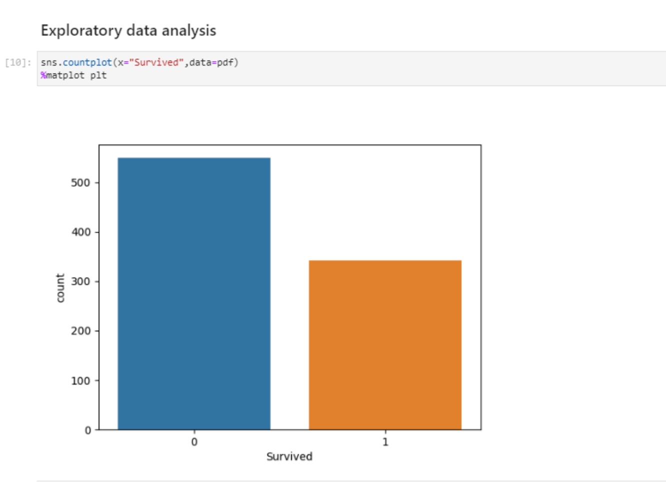
Spark Session ID	Kind	State	Spark UI	Driver log	User	Current session?
6	pyspark	idle			imageadmin	✔

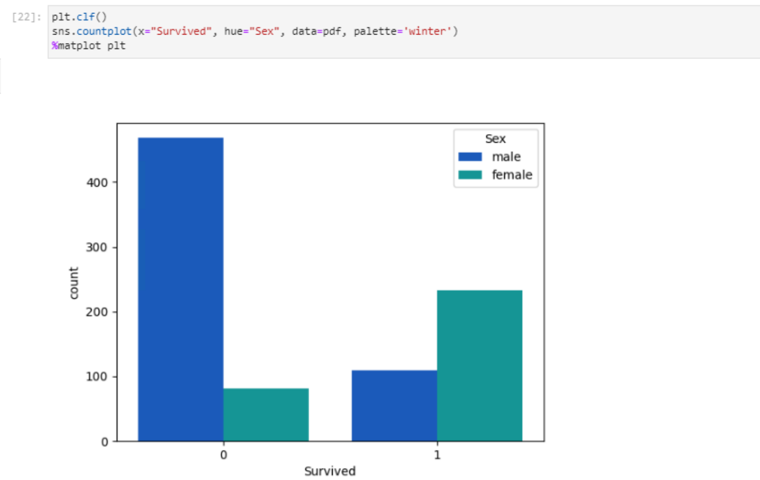
```
SparkSession available as 'spark'.
hi
```

[redacted] Copy this code to Notebook and execute it to read data from DataTap to Spark Data Frame.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import isnull, when, count, col
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

#spark = SparkSession.builder.appName("titanic").getOrCreate()

# read locally
sdf = spark.read.format("csv").option('header', 'true').load("dtap://TenantStorage/data/train.csv") # data from
https://www.kaggle.com/c/titanic/data

# or read from mapr file system
# titanic_sdf = (spark.read.format("csv").option('header', 'true').load("maprfs:///exthcp/tenant-
5/fsmount/repo/data/train.csv"))

#Converting Spark DataFrame To Pandas Datafame for exploratory data analysis
pdf = sdf.toPandas()
```

[redacted]Explore Data analysis.





[redacted]Transform and Train model.

```
data = sdf.select(col('Survived').cast('float'), col('Pclass').cast('float'),
                         col('Sex'), col('Age').cast('float'),
                         col('Fare').cast('float'), col('Embarked'))

data.select([count(when(isnull(c), c)).alias(c) for c in data.columns]).show()

data = dataset.replace('?', None).dropna(how='any')

data = StringIndexer(
            inputCol='Sex',
            outputCol='Gender',
            handleInvalid='keep').fit(data).transform(data)
data = StringIndexer(
            inputCol='Embarked',
            outputCol='Boarded',
            handleInvalid='keep').fit(data).transform(data)

data = data.drop('Sex')
data = data.drop('Embarked')
data.show()

required_features = ['Pclass', 'Age', 'Fare', 'Gender', 'Boarded']
assembler = VectorAssembler(inputCols=required_features, outputCol='features')
transformed_data = assembler.transform(data)
(training_data, test_data) = transformed_data.randomSplit([0.8,0.2])
rf = RandomForestClassifier(labelCol='Survived',
                            featuresCol='features',
                            maxDepth=5)
model = rf.fit(training_data)
```

```
+--------+------+---+---+----+--------+
|Survived|Pclass|Sex|Age|Fare|Embarked|
+--------+------+---+---+----+--------+
|       0|     0|  0|177|   0|       2|
+--------+------+---+---+----+--------+

+--------+------+----+-------+------+-------+
|Survived|Pclass| Age|   Fare|Gender|Boarded|
+--------+------+----+-------+------+-------+
|     0.0|   3.0|22.0|   7.25|   0.0|    0.0|
|     1.0|   1.0|38.0|71.2833|   1.0|    1.0|
|     1.0|   3.0|26.0|  7.925|   1.0|    0.0|
|     1.0|   1.0|35.0|   53.1|   1.0|    0.0|
|     0.0|   3.0|35.0|   8.05|   0.0|    0.0|
|     0.0|   1.0|54.0|51.8625|   0.0|    0.0|
|     0.0|   3.0| 2.0| 21.075|   0.0|    0.0|
|     1.0|   3.0|27.0|11.1333|   1.0|    0.0|
|     1.0|   2.0|14.0|30.0708|   1.0|    1.0|
|     1.0|   3.0| 4.0|   16.7|   1.0|    0.0|
|     1.0|   1.0|58.0|  26.55|   1.0|    0.0|
|     0.0|   3.0|20.0|   8.05|   0.0|    0.0|
|     0.0|   3.0|39.0| 31.275|   0.0|    0.0|
|     0.0|   3.0|14.0| 7.8542|   1.0|    0.0|
|     1.0|   2.0|55.0|   16.0|   1.0|    0.0|
|     0.0|   3.0| 2.0| 29.125|   0.0|    2.0|
|     0.0|   3.0|31.0|   18.0|   1.0|    0.0|
|     0.0|   2.0|35.0|   26.0|   0.0|    0.0|
|     1.0|   2.0|34.0|   13.0|   0.0|    0.0|
|     1.0|   3.0|15.0| 8.0292|   1.0|    2.0|
+--------+------+----+-------+------+-------+
only showing top 20 rows
```

[redacted]Validate and save model.

```
predictions = model.transform(test_data)
evaluator = MulticlassClassificationEvaluator(
               labelCol='Survived',
               predictionCol='prediction',
               metricName='accuracy')
accuracy = evaluator.evaluate(predictions)
print('==========' * 10)
print('Test Accuracy = ', accuracy)
print('==========' * 10)
# saving locally
model_location = "maprfs:///hcp/tenant-8/dco/data/sparkml-titanic4"

# saving on mapr file system
# model_location = "maprfs:///exthcp/tenant-5/fsmount/repo/models/sparkml-titanic"

model.save(model_location)
print("Model Save to location : {}".format(model_location))
```

```
=========================================================================================================
Test Accuracy =  0.7755102040816326
=========================================================================================================
```

Saving the Model

```
[17]: # saving locally
model_location = "maprfs:///hcp/tenant-8/dco/data/sparkml-titanic4"

# saving on mapr file system
# model_location = "maprfs:///exthcp/tenant-5/fsmount/repo/models/sparkml-titanic"

model.save(model_location)
print("Model Save to location : {}".format(model_location))
```

```
Model Save to location : maprfs:///hcp/tenant-8/dco/data/sparkml-titanic4
```

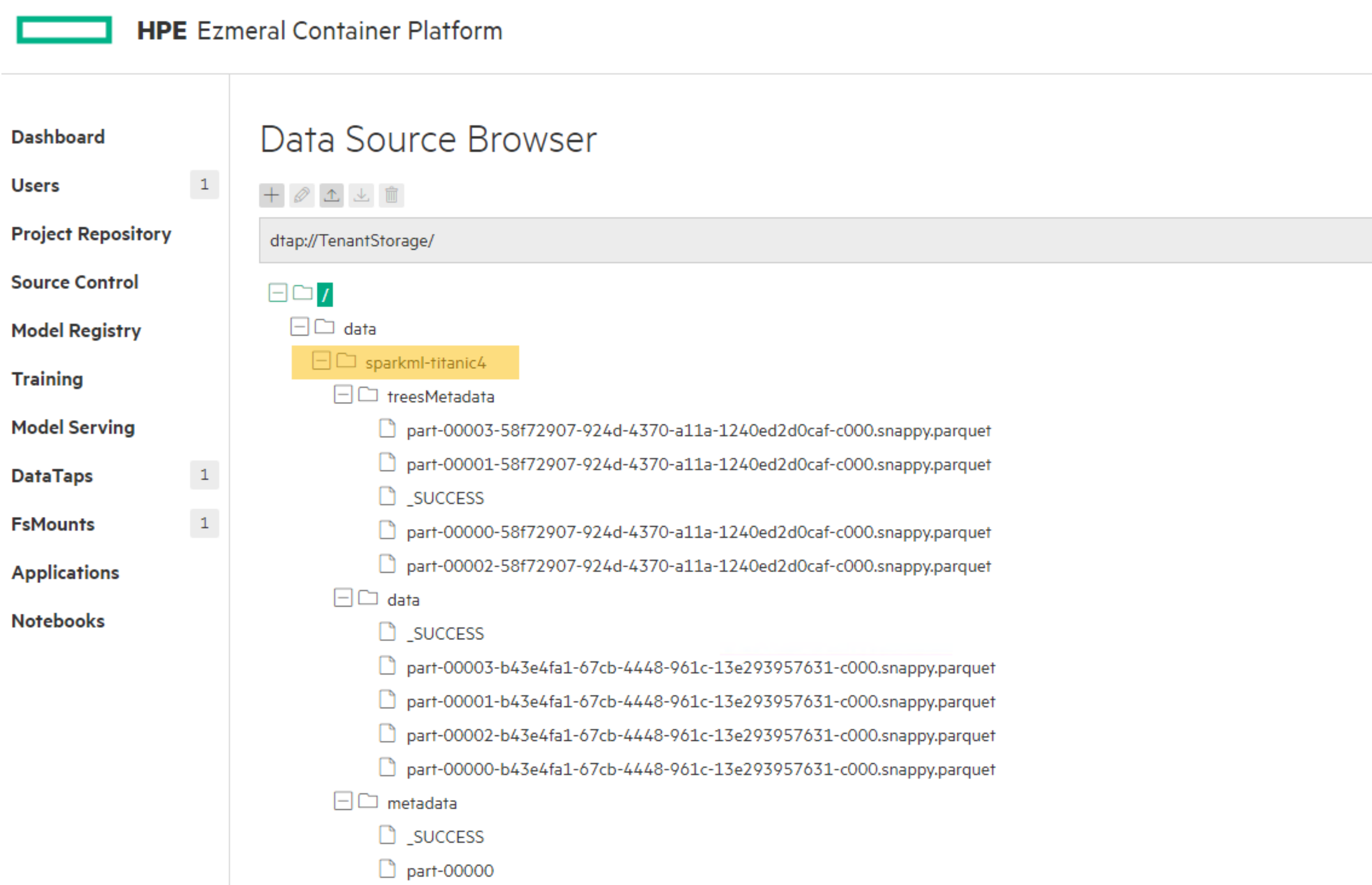


FIGURE 44. Data Source Browser

EZMERAL ML OPS – EXPERIMENT TRACKING WITH MLFLOW

The use case contains dataset preprocessing, model training and evaluation, model tuning via MLflow tracking, and finding the best-trained model.

Goal: Predict rented_bikes (count per hour) based on weather and time information.

Prerequisite
Dataset: Bike Sharing Dataset

Use case workflow

Following are the steps to implement the use case and run the experiment into the MLflow model server.

[redacted]Initialize the web terminal.

[redacted]After login to HPE Ezmeral Container Platform 5.3 and in MLOps Tenant, initialize the web terminal.

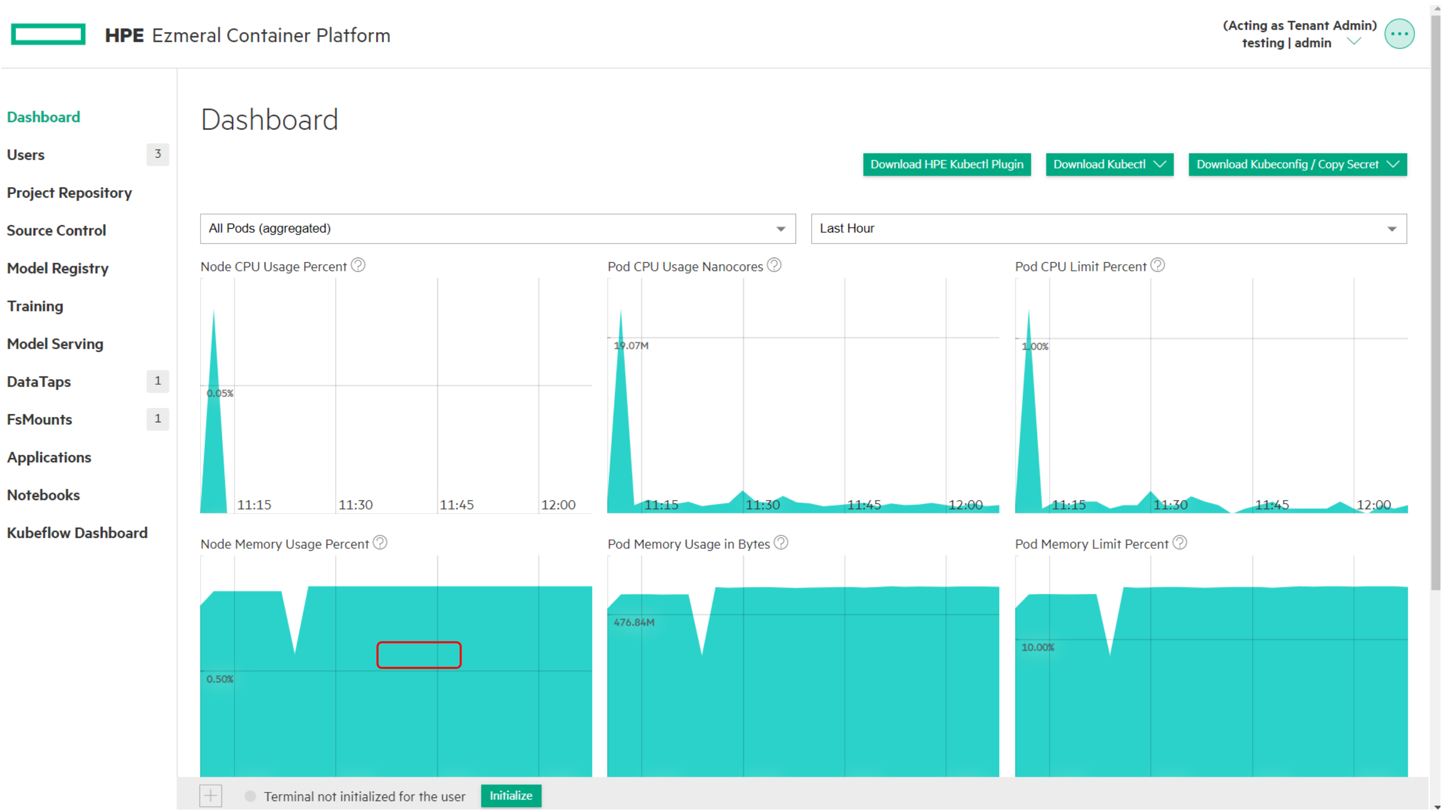
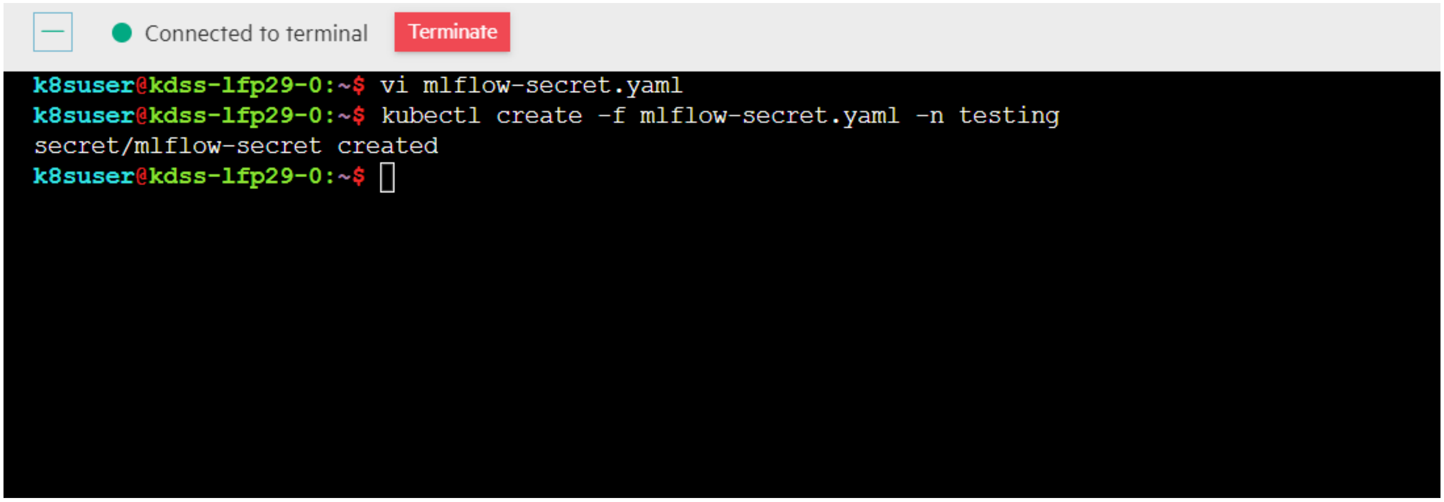


FIGURE 45. Dashboard

[redacted]Create Secret. Once the web terminal is connected, create a file `mlflow-secret.yaml`.

```
apiVersion: v1
kind: Secret
type: Opaque
metadata:
  name: mlflow-secret
  labels:
    kubedirector.hpe.com/secretType: mlflow
data:
  AWS_ACCESS_KEY_ID: YWRtaW4= #admin
  AWS_SECRET_ACCESS_KEY: YWRtaW4xMjM= #admin123
  MLFLOW_ARTIFACT_ROOT: czM6Ly9tbGZsb3c= #s3://mlflow
```

```
$ kubectl create -f mlflow-secret.yaml -n testing
```



[redacted]Launch MLflow Server. Go to the Application Tab and launch MLflow Server.

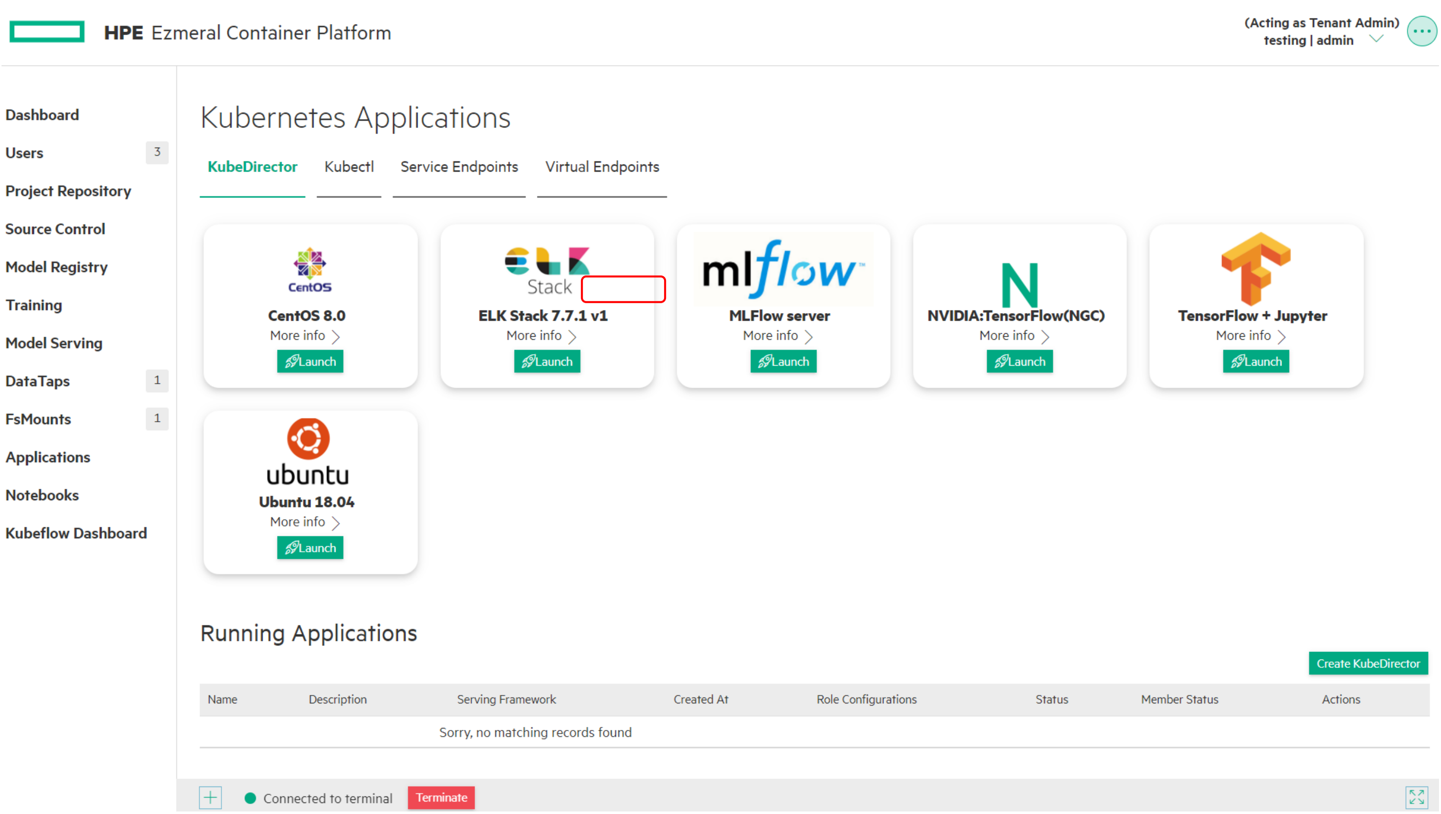


FIGURE 46. Kubernetes Applications

[redacted]We need to attach the secret created in step 3 in YAML of MLflow Server before launching it.

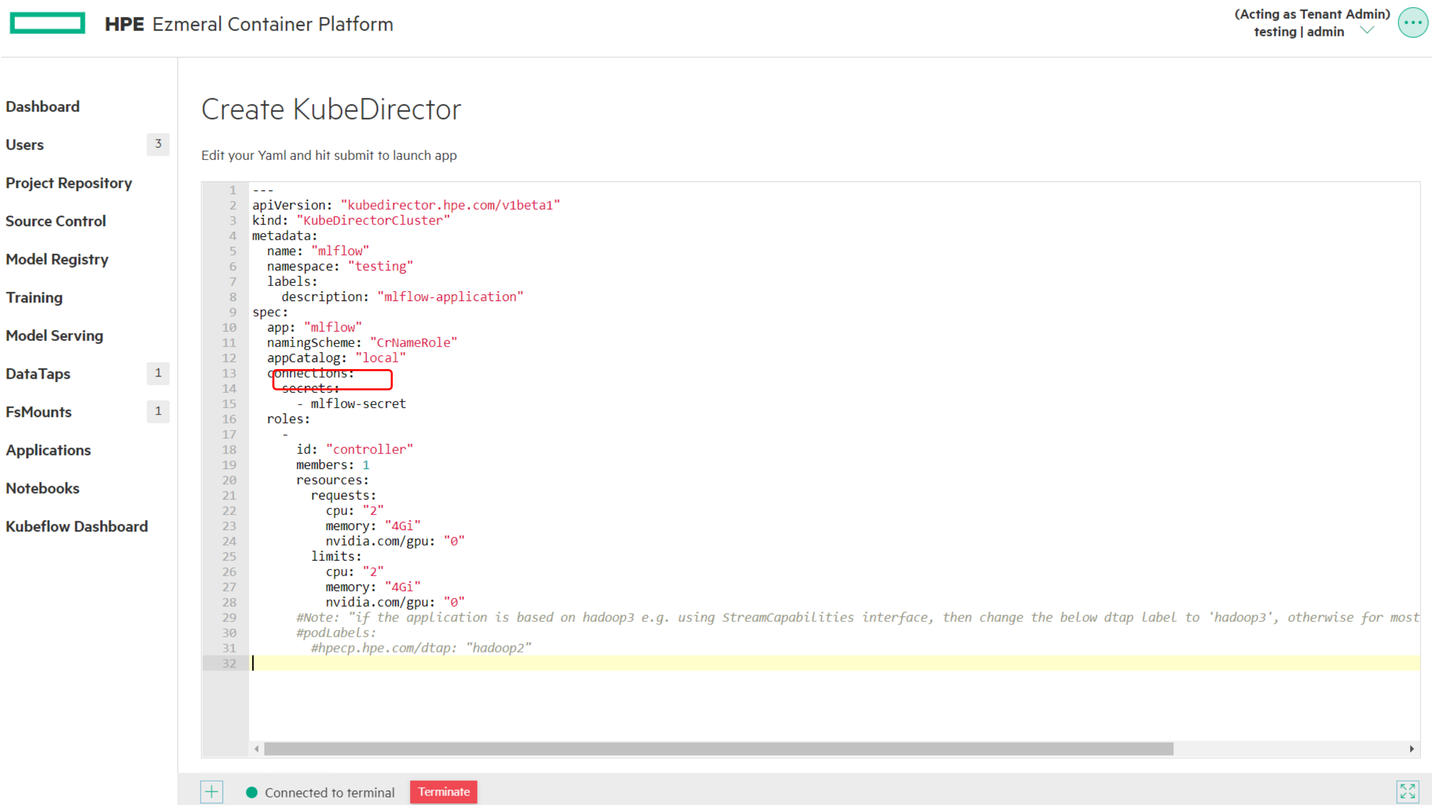


FIGURE 47. Create/edit KubeDirector YAML

[redacted]Accessing Minio and MLflow Server.

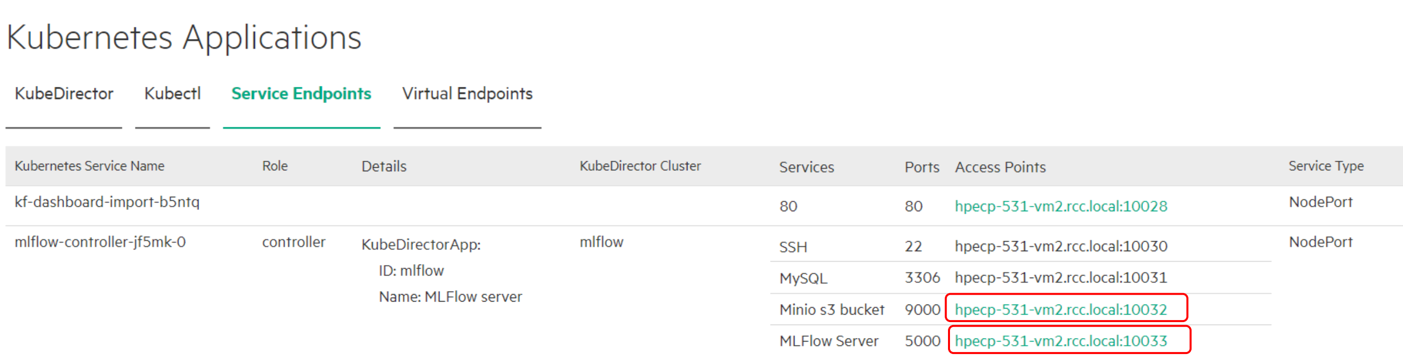


FIGURE 48. Service Endpoints

[redacted]On the MinIO Browser page, create a bucket name `mlflow` which is configured in `mlflow-secret.yaml`.

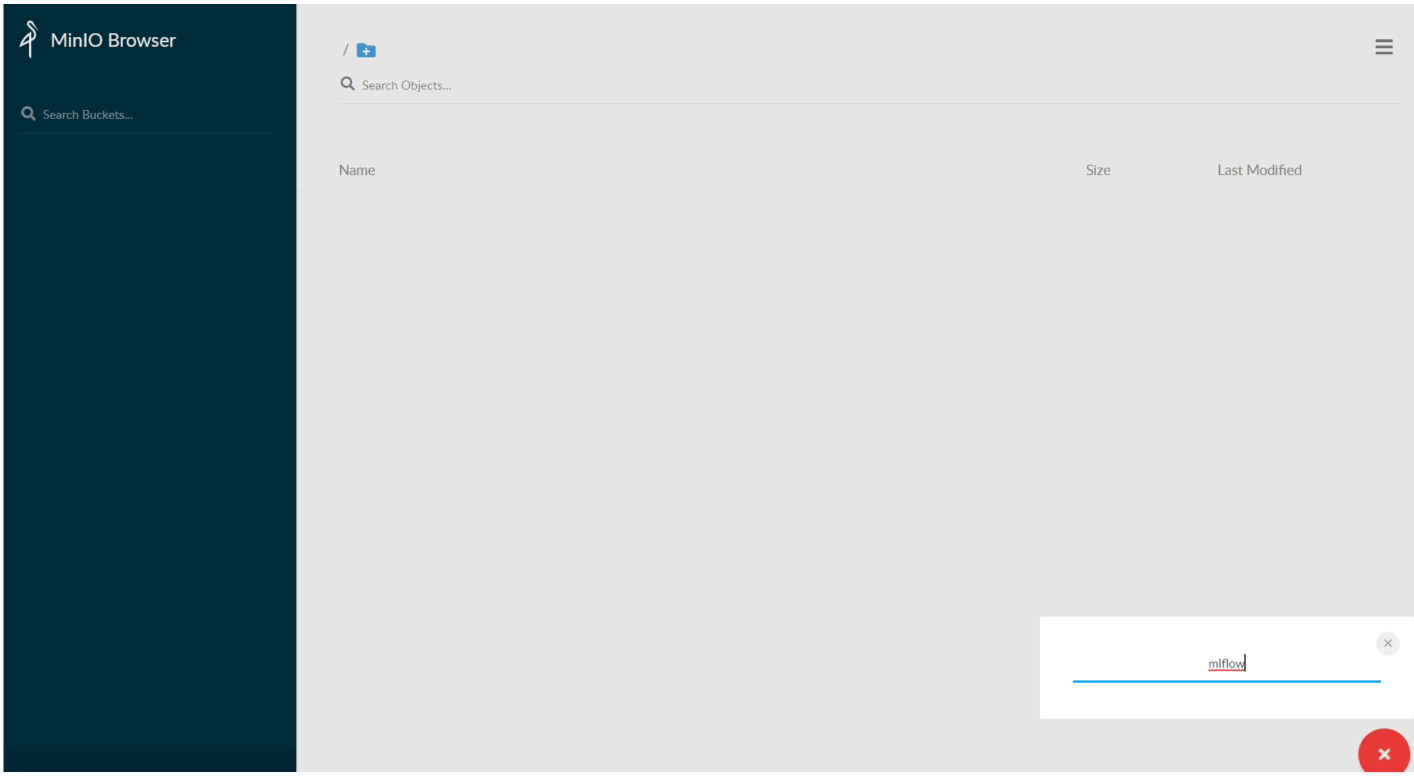


FIGURE 49. MinIO Browser

MLflow Server

[redacted]Open the MLflow Server window.

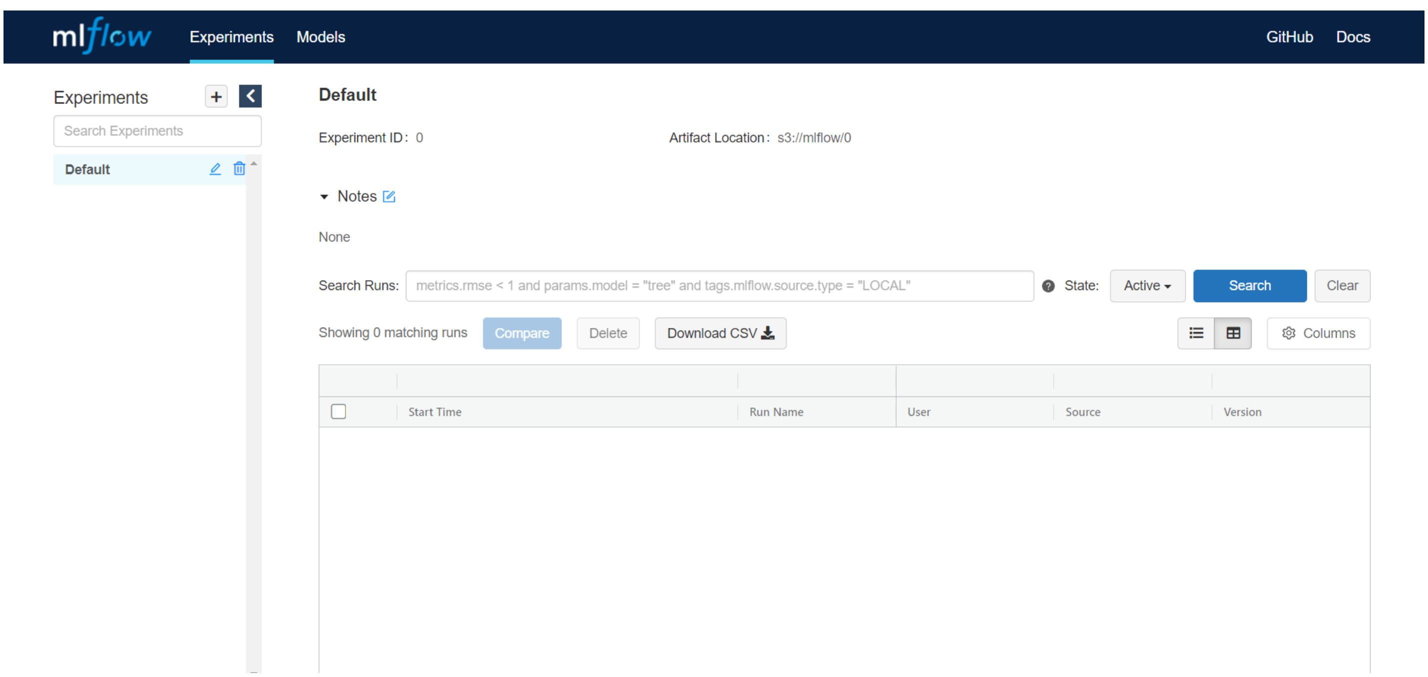


FIGURE 50. MLflow Server

[redacted] Deploy KubeDirector Notebook. Go to the Notebook Tab and launch Jupyter Notebook.

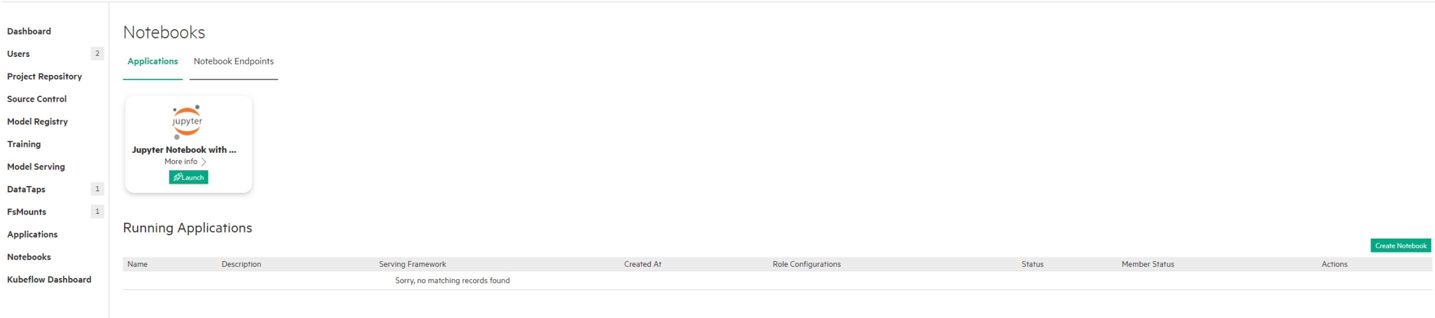


FIGURE 51. Launch Jupyter Notebook

[redacted] Attach 'clusters' and 'secret' in Launch YAML of the Jupyter Notebook.

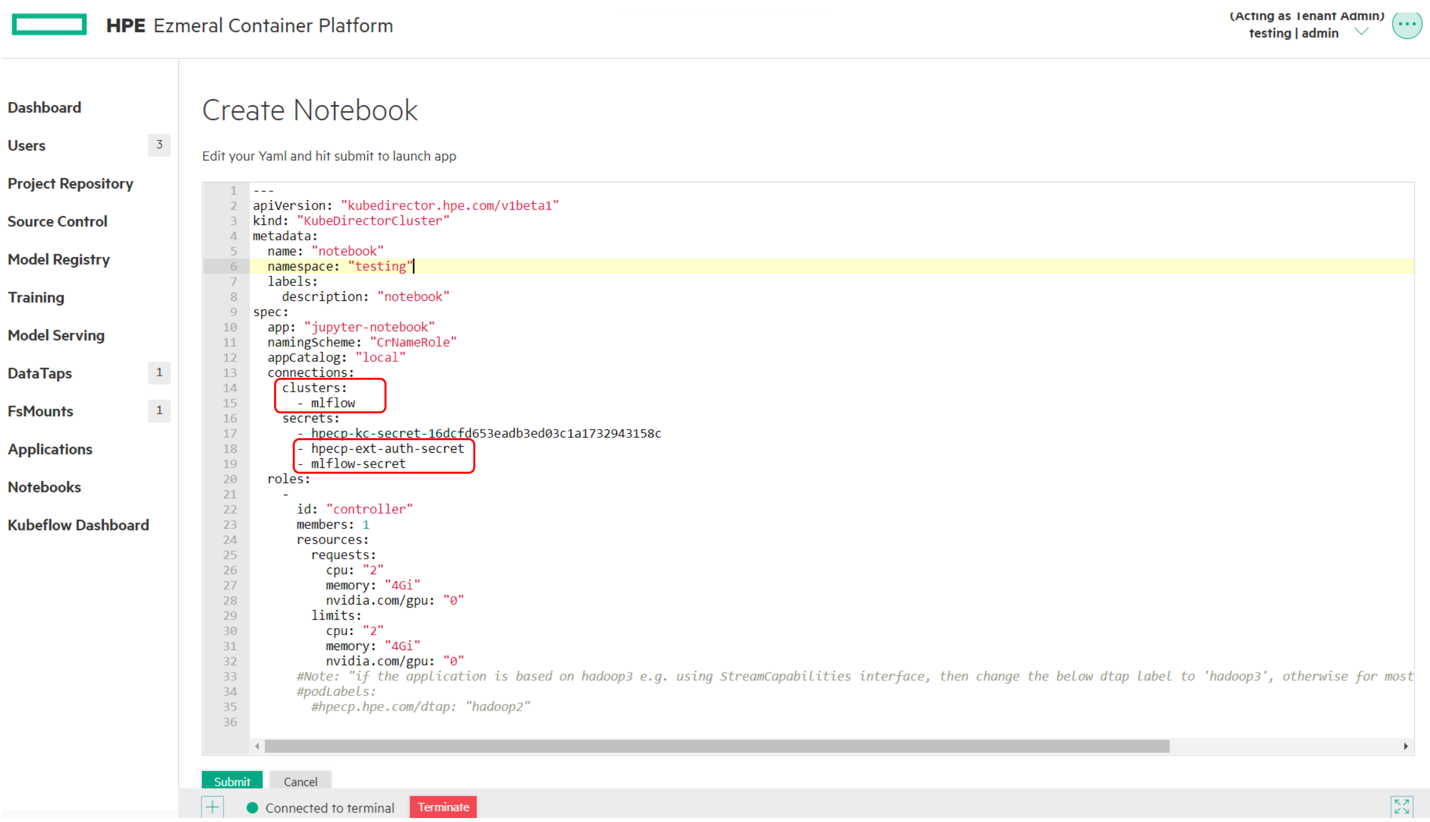
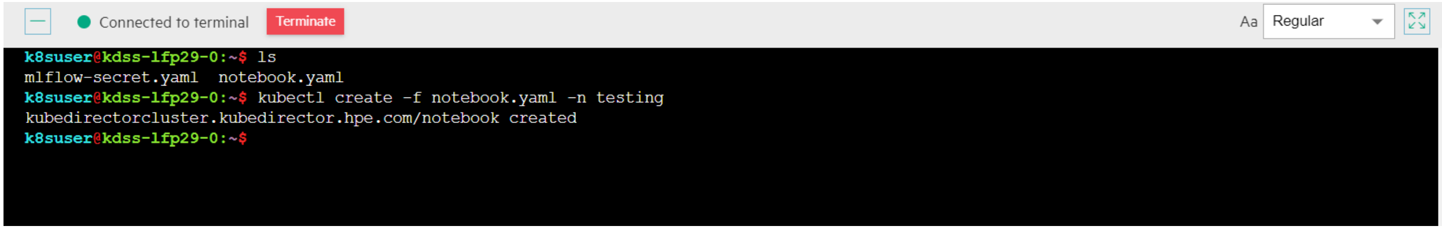


FIGURE 52. Edit Launch yaml

In Figure 52, **mlflow** in clusters and **mlflow-secret** in secrets are attached.

- **mlflow** is the application name which deployed
- **mlflow-secret** is the secret created

[redacted]Launching Notebook with MLflow cluster attached from HPE Ezmeral UI has known issue, so notebook to be created from the terminal. Copy the above YAML file and create a file at the terminal and apply using kubectl, as shown below (workaround).



[redacted]Access the Notebook from the Notebook Endpoints tab.

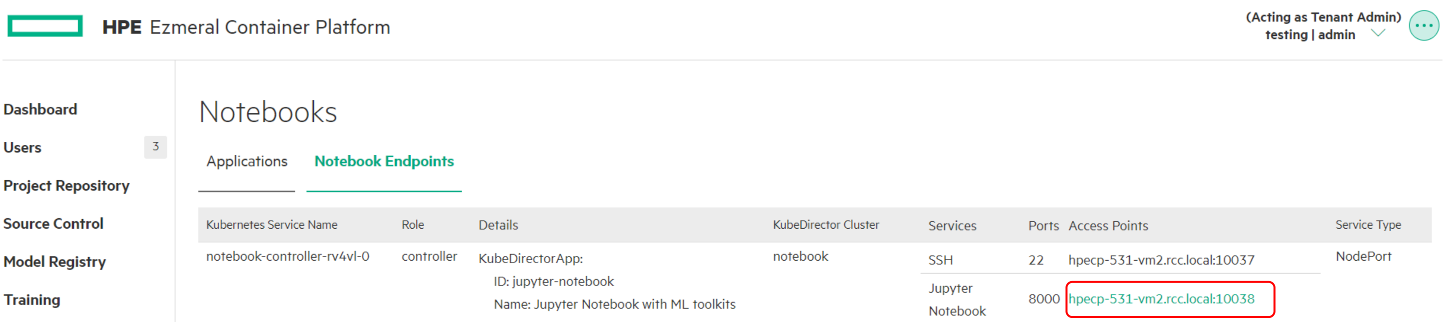


FIGURE 53. Notebook Endpoints

[redacted]Upload Notebook File and Dataset to run the experiment.

[redacted]Login to the Jupyterhub.

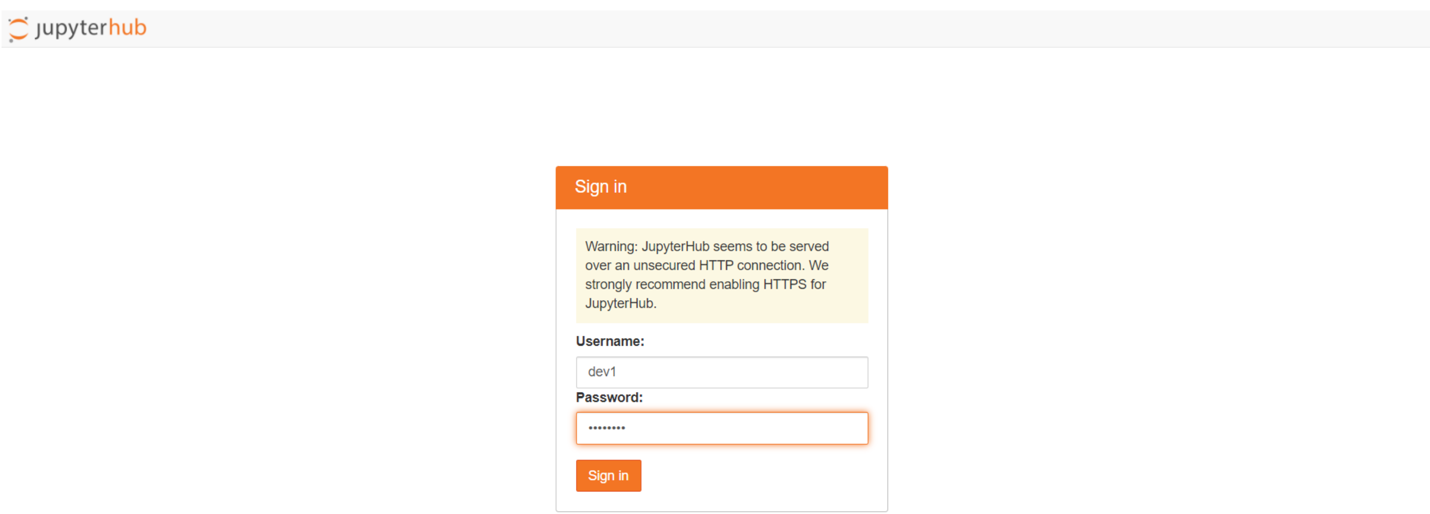


FIGURE 54. Login to Jupyterhub

[redacted]After login to Jupyter Notebook upload the care directory Bike-Sharing-MLFlow-UseCase and upload Bike-Sharing-MLFlow.ipynb, day.csv, and hour.csv from https://github.com/SANDataHPE/MLFlow-Examples/tree/main/Bike-Sharing-MLFlow-Example.

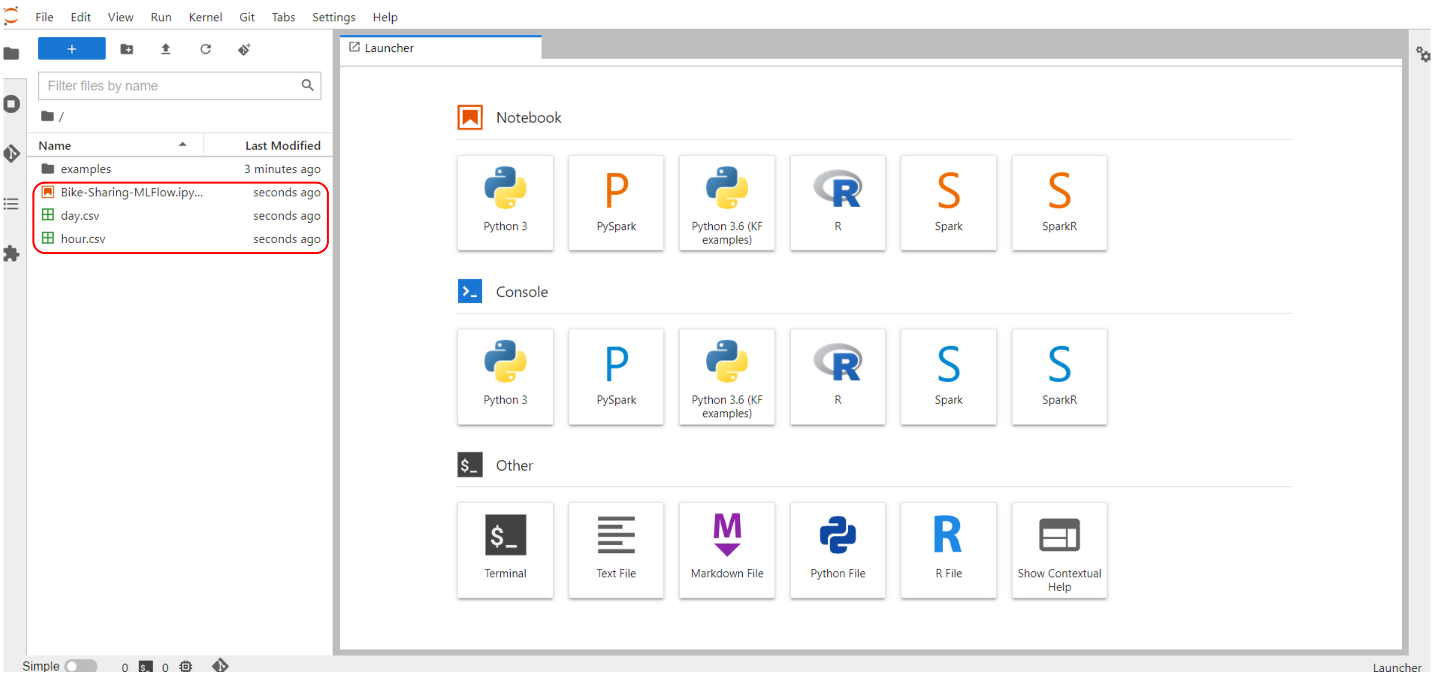
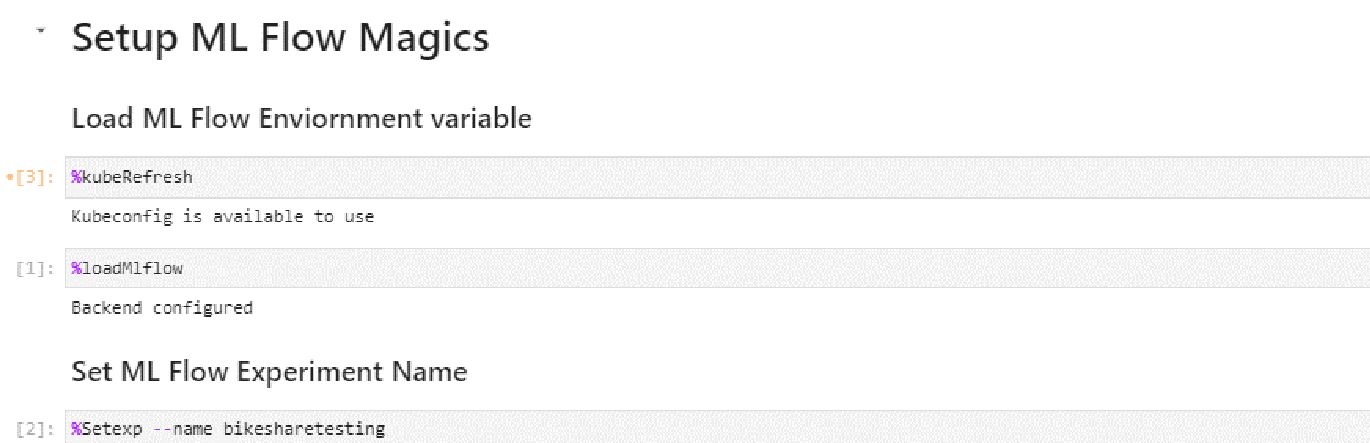


FIGURE 55. Jupyter Notebook

[redacted]Setup 'ML Flow' magics.

a. Add the user (which we've logged in to notebook with) to the platform and give it member access to the tenant.

b. Run `%kubeRefresh` in one of the cells.



c. Load MLflow server and the secret to our notebook. Run `%loadmlflow`.

[redacted]Execute cell in Notebook file.

a. Each notebook cell contains a comment about the task which we are performing in each cell. Read the markdown before executing the cell. Follow the instruction written in the notebook before executing each cell.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import mlflow
import mlflow.sklearn
from mlflow import log_metric, log_param, log_artifact

from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.inspection import permutation_importance
from mlflow.models.signature import infer_signature
from sklearn import tree

from pydotplus import graph_from_dot_data
import graphviz
from IPython.display import Image

import itertools

plt.style.use(”fivethirtyeight”)
pd.plotting.register_matplotlib_converters()

import warnings
warnings.filterwarnings(’ignore’)
bike_sharing = pd.read_csv(”hour.csv”)
bike_sharing
# remove unused columns
bike_sharing.drop(columns=[”instant”, ”dteday”, ”registered”, ”casual”], inplace=True)

# use better names
bike_sharing.rename(
    columns={
        ”yr”: ”year”,
        ”mnth”: ”month”,
        ”hr”: ”hour_of_day”,
        ”holiday”: ”is_holiday”,
        ”workingday”: ”is_workingday”,
        ”weathersit”: ”weather_situation”,
        ”temp”: ”temperature”,
        ”atemp”: ”feels_like_temperature”,
        ”hum”: ”humidity”,
        ”cnt”: ”rented_bikes”,
    },
    inplace=True,
)

# show samples
bike_sharing
hour_of_day_agg = bike_sharing.groupby([”hour_of_day”])[”rented_bikes”].sum()

hour_of_day_agg.plot(
    kind=”line”,
    title=”Total rented bikes by hour of day”,
    xticks=hour_of_day_agg.index,
```

```
    figsize=(15, 10),
)
# Split the dataset randomly into 70% for training and 30% for testing.
X = bike_sharing.drop("rented_bikes", axis=1)
y = bike_sharing.rented_bikes
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, test_size=0.3, random_state=42)

print(f"Training samples: {X_train.size}")
print(f"Test samples: {X_test.size}")
def rmse(y, y_pred):
    return np.sqrt(mean_squared_error(y, y_pred))


def rmse_score(y, y_pred):
    score = rmse(y, y_pred)
    print("RMSE score: {:.4f}".format(score))
    return score
def rmsle_cv(model, X_train, y_train):
    kf = KFold(n_splits=3, shuffle=True, random_state=42).get_n_splits(X_train.values)
    # Evaluate a score by cross-validation
    rmse = np.sqrt(-cross_val_score(model, X_train.values, y_train, scoring="neg_mean_squared_error", cv=kf))
    return rmse


def rmse_cv_score(model, X_train, y_train):
    score = rmsle_cv(model, X_train, y_train)
    print("Cross-Validation RMSE score: {:.4f} (std = {:.4f})".format(score.mean(), score.std()))
    return score
def model_feature_importance(model):
    feature_importance = pd.DataFrame(
        model.feature_importances_,
        index=X_train.columns,
        columns=["Importance"],
    )

    # sort by importance
    feature_importance.sort_values(by="Importance", ascending=False, inplace=True)

    # plot
    plt.figure(figsize=(12, 8))
    sns.barplot(
        data=feature_importance.reset_index(),
        y="index",
        x="Importance",
    ).set_title("Feature Importance")
    # save image
    plt.savefig("model_artifacts/feature_importance.png", bbox_inches='tight')

def model_permutation_importance(model):
    p_importance = permutation_importance(model, X_test, y_test, random_state=42, n_jobs=-1)

    # sort by importance
    sorted_idx = p_importance.importances_mean.argsort()[::-1]
    p_importance = pd.DataFrame(
        data=p_importance.importances[sorted_idx].T,
        columns=X_train.columns[sorted_idx]
    )

    # plot
    plt.figure(figsize=(12, 8))
    sns.barplot(
```

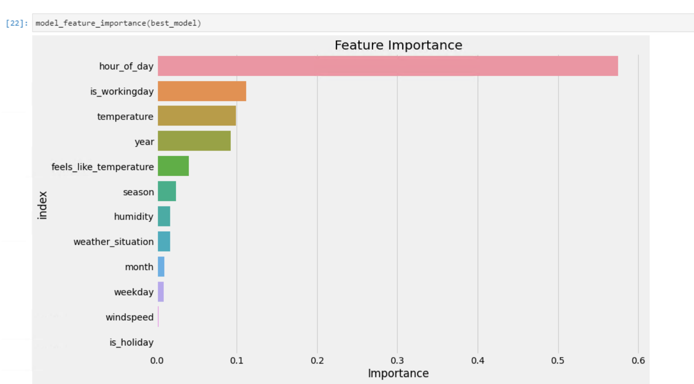
```
        data=p_importance,
        orient="h"
    ).set_title("Permutation Importance")

    # save image
    plt.savefig("model_artifacts/permutation_importance.png", bbox_inches="tight")

def model_tree_visualization(model):
    # generate visualization
    tree_dot_data = tree.export_graphviz(
        decision_tree=model.estimators_[0, 0],  # Get the first tree,
        label="all",
        feature_names=X_train.columns,
        filled=True,
        rounded=True,
        proportion=True,
        impurity=False,
        precision=1,
    )

    # save image
    graph_from_dot_data(tree_dot_data).write_png("model_artifacts/Decision_Tree_Visualization.png")

    # show tree
    return graphviz.Source(tree_dot_data)
# Track params and metrics
def log_mlflow_run(model, signature):
    # Auto-logging for scikit-learn estimators
    # mlflow.sklearn.autolog()

    # log estimator_name name
    name = model.__class__.__name__
    mlflow.set_tag("estimator_name", name)

    # log input features
    mlflow.set_tag("features", str(X_train.columns.values.tolist()))

    # Log tracked parameters only
    mlflow.log_params({key: model.get_params()[key] for key in parameters})

    mlflow.log_metrics({
        'RMSE_CV': score_cv.mean(),
        'RMSE': score,
    })

    # log training loss
    for s in model.train_score_:
        mlflow.log_metric("Train Loss", s)

    # Save model to artifacts
    mlflow.sklearn.log_model(model, "model", signature=signature)

    # log charts
    mlflow.log_artifacts("model_artifacts")

    # misc
    # Log all model parameters
    mlflow.log_params(model.get_params())
    mlflow.log_param("Training size", X_test.size)
    mlflow.log_param("Test size", y_test.size)
# GBRT (Gradient Boosted Regression Tree) scikit-learn implementation
```

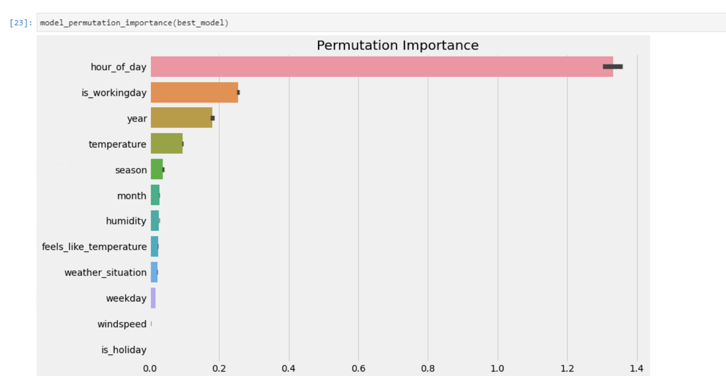
```
model_class = GradientBoostingRegressor
parameters = {
    "learning_rate": [0.1, 0.05, 0.01],
    "max_depth": [4, 5, 6],
    # "verbose": True,
}
# generate parameters combinations
params_keys = parameters.keys()
params_values = [
    parameters[key] if isinstance(parameters[key], list) else [parameters[key]]
    for key in params_keys
]
runs_parameters = [
    dict(zip(params_keys, combination)) for combination in itertools.product(*params_values)
]

# training loop
for i, run_parameters in enumerate(runs_parameters):
    print(f"Run {i}: {run_parameters}")

    # mlflow: stop active runs if any
    if mlflow.active_run():
        mlflow.end_run()
    # mlflow:track run
    mlflow.start_run(run_name=f"Run {i}")

    # create model instance
    model = model_class(**run_parameters)

    # train
    model.fit(X_train, y_train)

    # get evaluations scores
    score = rmse_score(y_test, model.predict(X_test))
    score_cv = rmse_cv_score(model, X_train, y_train)

    # generate charts
    model_feature_importance(model)
    plt.close()
    model_permutation_importance(model)
    plt.close()
    model_tree_visualization(model)

    # get model signature
    signature = infer_signature(model_input=X_train, model_output=model.predict(X_train))

    # mlflow: log metrics
    log_mlflow_run(model, signature)

    # mlflow: end tracking
    mlflow.end_run()
    print("")

best_run_df = mlflow.search_runs(order_by=['metrics.RMSE_CV ASC'], max_results=1)
if len(best_run_df.index) == 0:
    raise Exception(f"Found no runs for experiment '{experiment_name}'")

best_run = mlflow.get_run(best_run_df.at[0, 'run_id'])
best_model_uri = f"{best_run.info.artifact_uri}/model"
best_model = mlflow.sklearn.load_model(best_model_uri)
# print best run info
```

```
print("Best run info:")
print(f"Run id: {best_run.info.run_id}")
print(f"Run parameters: {best_run.data.params}")
print("Run score: RMSE_CV = {:.4f}".format(best_run.data.metrics['RMSE_CV']))
print(f"Run model URI: {best_model_uri}")
model_feature_importance(best_model)
model_permutation_importance(best_model)
model_tree_visualization(best_model)
test_predictions = X_test.copy()
# real output (rented_bikes) from test dataset
test_predictions["rented_bikes"] = y_test

# add "predicted_rented_bikes" from test dataset
test_predictions["predicted_rented_bikes"] = best_model.predict(X_test).astype(int)

# show results
test_predictions
# plot truth vs prediction values
test_predictions.plot(
    kind="scatter",
    x="rented_bikes",
    y="predicted_rented_bikes",
    title="Rented bikes vs predicted rented bikes",
    figsize=(15, 15),
)
```

NOTE
We need to install all the required packages for the use case as shown below, if already installed this step can be skipped.

[redacted]Visualize the best model.





[redacted]Model Artifact in MLflow Server.

a. For the model artifact, we can access the MLflow Server.

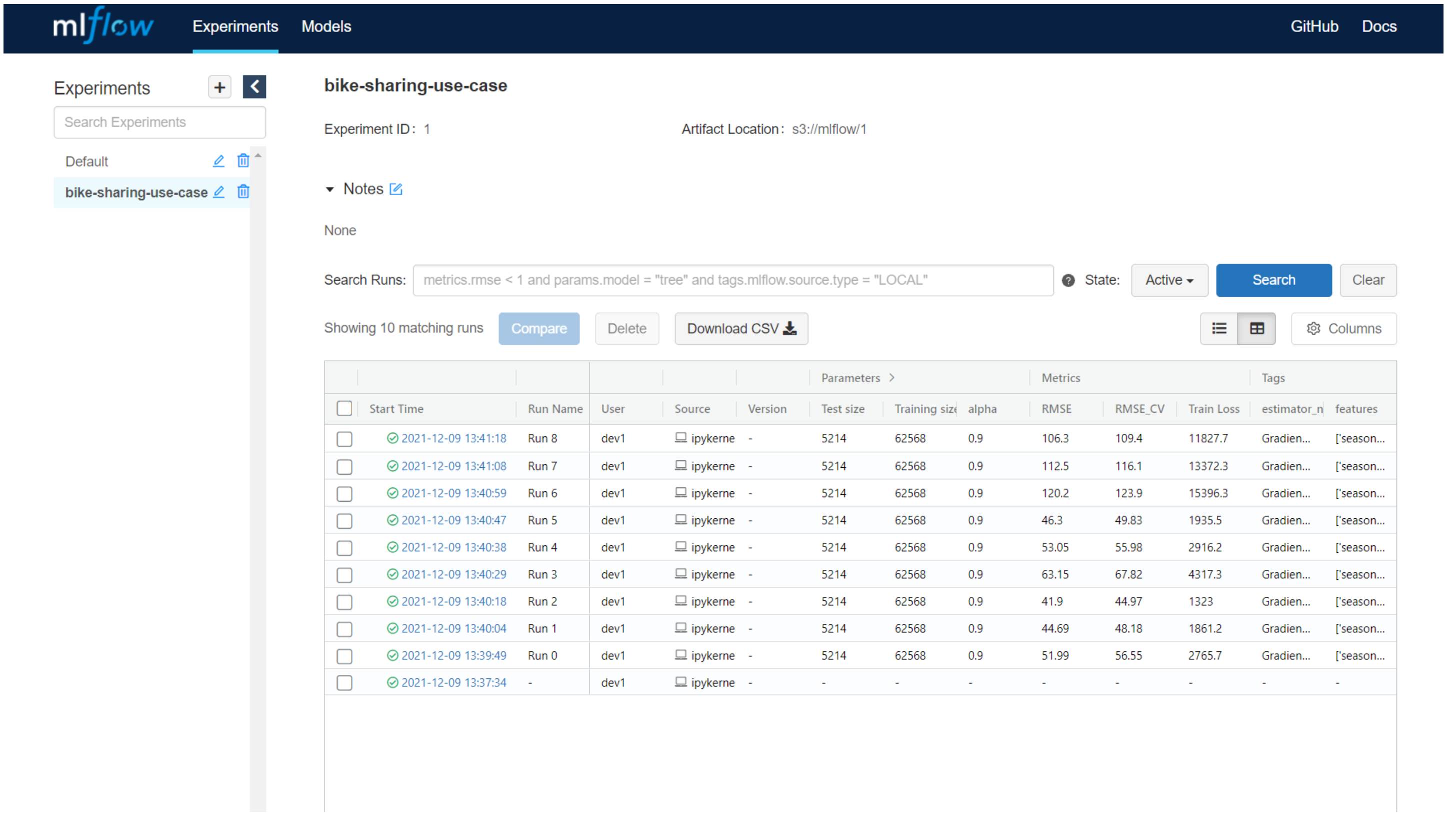


FIGURE 56. MLflow Experiments

MONITORING

There are different levels of monitoring with HPE Ezmeral ML Ops.

- At the platform level, the HPE Ezmeral Runtime provides dashboards that allow users to monitor resource utilization
- At the application level such as Kubeflow, containerized monitoring services such as Istio Prometheus are provided

Kubernetes administrator

Kubernetes users who have access to the site admin tenant can view the platform administrator dashboard which presents a high-level overview of the Kubernetes activity. Figure 57 shows the Usage tab displays usage information on a per-tenant basis. Refer to the HPE Ezmeral Container Platform 5.3 documentation, Dashboard – Kubernetes Administrator for more details.

Beginning with HPE Ezmeral Runtime 5.3, Dashboard views show additional GPU usage for Tesla-class or Quadro-class GPU families. For site administrators, the **Dashboard → Usage** tab shows the GPU devices used system-wide, while for Cluster Admin the stats would show usage cluster-wide. Finally, for tenant members, the resource usage statistics provide pod-specific GPU device information.

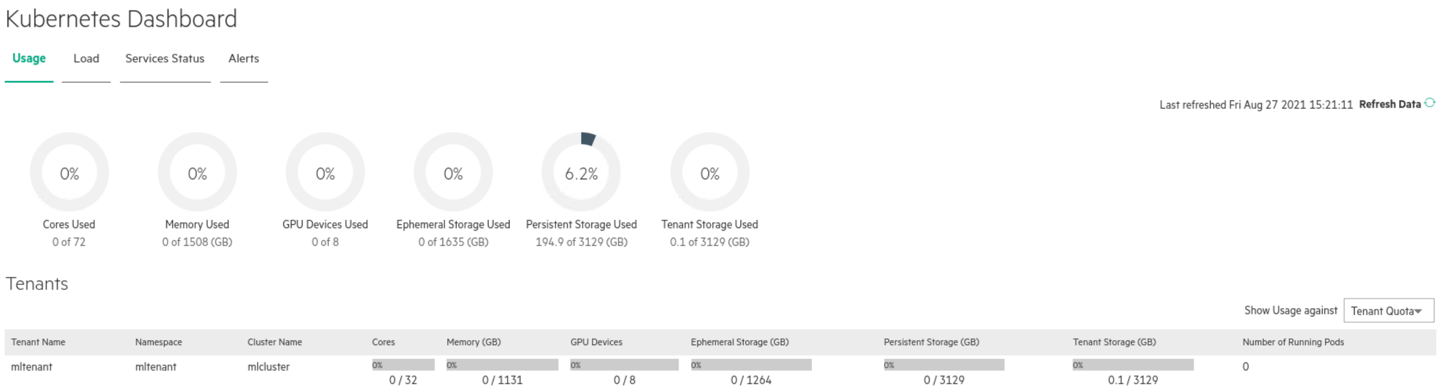


FIGURE 57. Platform administrator usage dashboard

Figure 59 shows the Load tab displays load statistics for the on-premises CPU, GPU, memory, and network resources within the K8s platform.

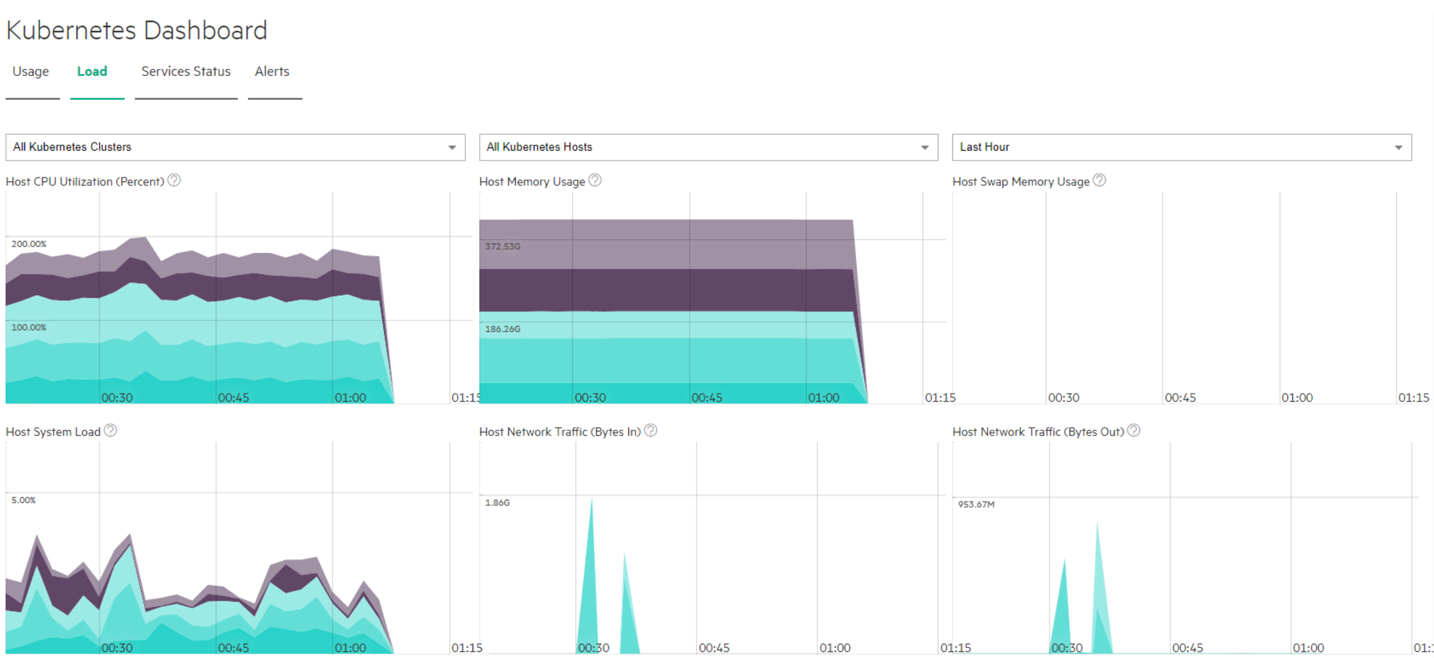


FIGURE 58. Platform administrator load dashboard

Kubernetes tenant/project administrator

HPE Ezmeral Runtime users who are logged into a Kubernetes tenant/project with the tenant/project administrator role can access the Kubernetes tenant/project administrator Dashboard. A tenant/project admin has access to the main menu and can view login details, alerts, etc. For more details, see the Toolbar & Main Menu - Kubernetes Tenant Admin page.

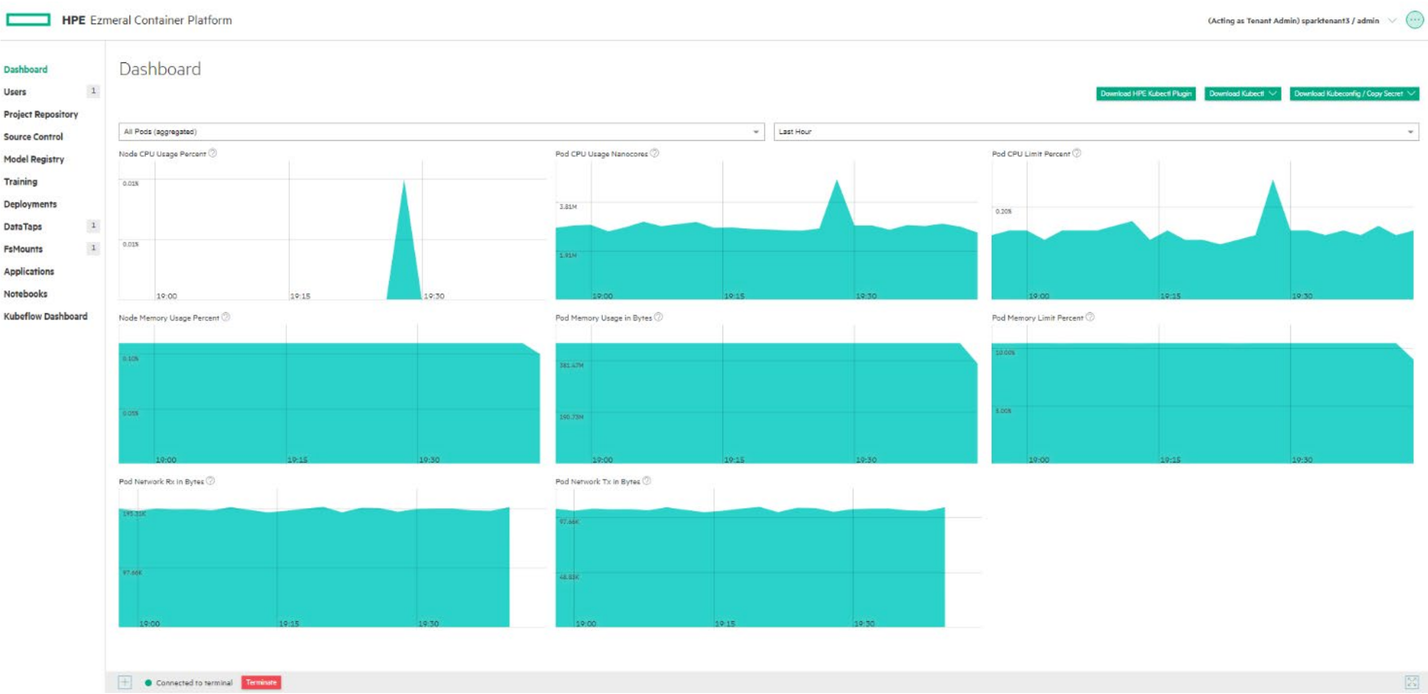


FIGURE 59. Tenant/project admin Dashboard

Istio and Prometheus

Prometheus is used for monitoring and Istio is used for managing network communication in the form of a service mesh.

Istio Prometheus use case

[redacted]Create tunnel for 9090 port via SSH.

```
$ ssh -L 9090:localhost:9090 root@<master node>
```

```
k8suser@kdss-p9tw8-0:~$
k8suser@kdss-p9tw8-0:~$ ssh -L 9090:localhost:9090 root@172.24.2.4
The authenticity of host '172.24.2.4 (172.24.2.4)' can't be established.
ECDSA key fingerprint is SHA256:1W8mZtT7R8FzqUr2yLpHRpl7Vm6UilbYeiYZeUBCy2k.
ECDSA key fingerprint is MD5:3d:36:80:3f:58:2c:c0:f3:5b:f3:65:db:e7:4f:a6:8b.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '172.24.2.4' (ECDSA) to the list of known hosts.
root@172.24.2.4's password:
bind: Cannot assign requested address
Last login: Mon Aug 23 15:36:53 2021 from ilomxq231019r.perflab.hp.com
[root@ezam-04 ~]# kubectl port-forward svc/prometheus -n istio-system 9090:9090
Forwarding from 127.0.0.1:9090 -> 9090
Forwarding from [::1]:9090 -> 9090
```

Enable port forwarding.

```
$ kubectl port-forward svc/prometheus -n istio-system 9090:9090
```

On the master node, start up the Firefox, and go to `$ http://127.0.0.1:9090/`.

Select **Status** → **Targets**, and then verify that all targets have been discovered and their statuses are being monitored.

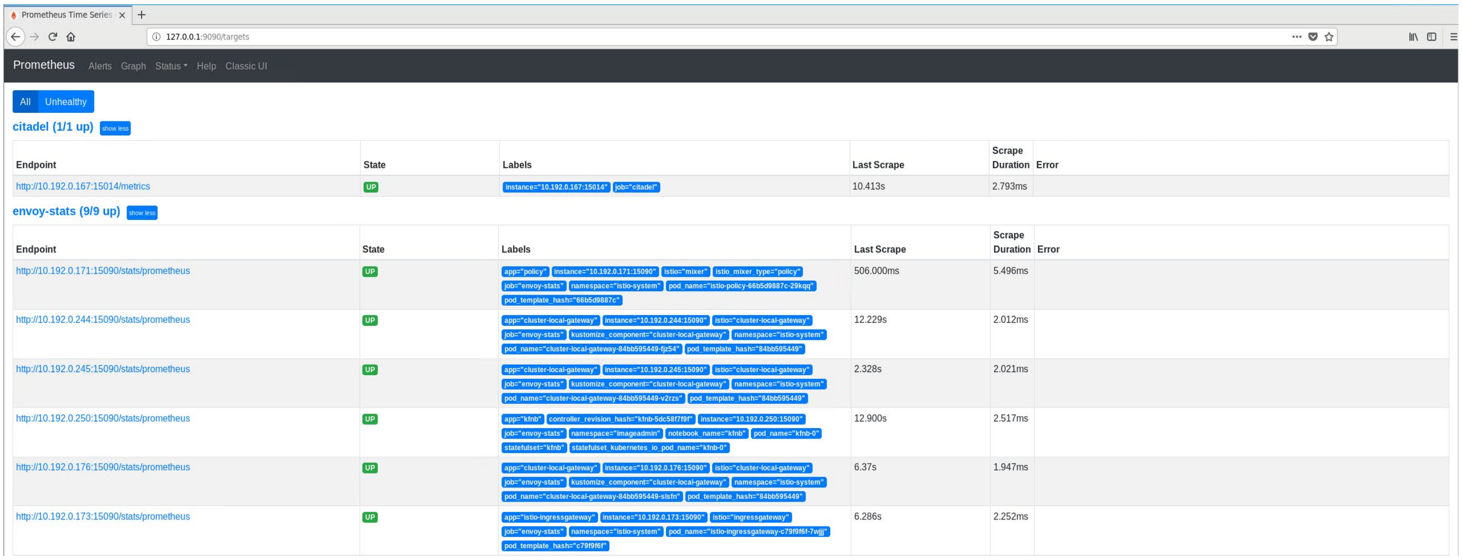


FIGURE 60. Targets

Select **Status** → **Service Discovery**, and then verify that all services have been discovered and their statuses are being monitored.

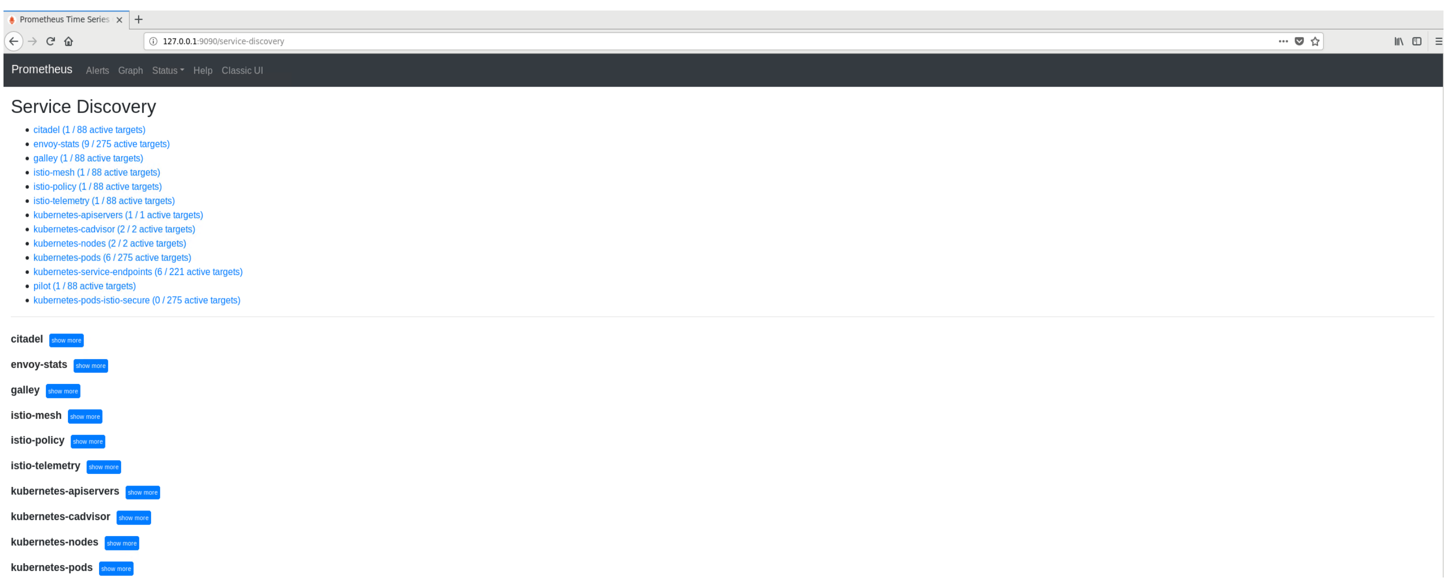


FIGURE 61. Service Discovery

[redacted] Navigate to Graph.

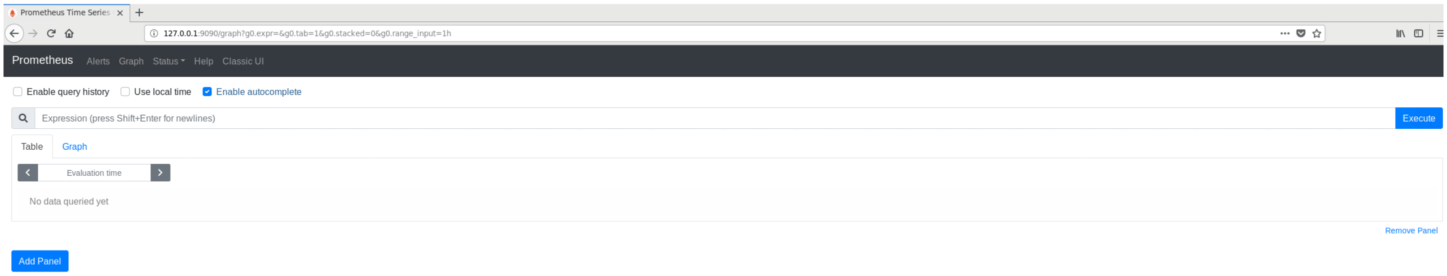


FIGURE 62. Graph

[redacted] Choose any command you want and select it.

[redacted] Click the Execute button and observe the Graph tab.

SUMMARY

Enterprises across all industries are embarking on a hybrid cloud journey for the development and deployment of their data-driven analytics and AI/ML applications. The continuous integration/continuous deployment (CI/CD) workflows, collectively referred to as DevOps, have become ubiquitous for software development today. On the machine learning front, data scientists still spend a significant amount of time and effort moving projects from development to production. Model version control is still largely manual, making it hard to update models in production. Code sharing is manual; data copied onto local storage leads to the variability of results between environments. There is a lack of standardization tools and frameworks, which makes it tedious and time-consuming to ensure the accuracy of predictions across all environments.

HPE Ezmeral Machine Learning Ops Software (HPE Ezmeral ML Ops) includes the capabilities and functionality of the HPE Ezmeral Runtime while also providing DevOps like agility to enterprise machine learning. With HPE Ezmeral ML Ops, enterprises can implement CI/CD workflows and standardize their ML pipelines. The HPE Ezmeral ML Ops software platform supports every stage of the machine learning lifecycle — supporting sandbox experimentation with a choice of ML/DL frameworks, integrations with model and code repositories, to deploying and tracking models in production.

HPE Ezmeral ML Ops gives data scientists and developers the ability to quickly and easily build and train machine learning models. HPE Ezmeral ML Ops allows data scientists to manage and track models built on any platform and deploy them into a scalable and secure production environment. Using HPE Ezmeral ML Ops, data scientists can spin-up containerized environments for distributed data processing, Machine Learning (ML), or Deep Learning (DL) in minutes rather than weeks. HPE Ezmeral ML Ops provides data science teams the flexibility to run their ML/DL workloads either on-premises, in multiple public clouds, or in a hybrid model and respond to dynamic business requirements in a variety of use cases.

With HPE Ezmeral ML Ops Software, Hewlett Packard Enterprise is making it easy for organizations to deliver a flexible and secure multitenant architecture, with the agility, flexibility, and performance needed to address evolving workload and application requirements. HPE Ezmeral ML Ops is deployed using pre-tested and optimized HPE Apollo building blocks on-premises, as well as in hybrid IT architectures and multi-cloud models.

Companies are driving digital transformation and investing in innovation to remain competitive. They are looking to deploy modern apps faster and simplify the production environment in a hybrid cloud architecture. They may have a mandate to move their application portfolio to the cloud or containers. Many organizations are still struggling to achieve these goals due to a lack of time and expertise. This Reference Architecture showcases the “lift-and-shift” application modernization use case which allows organizations to accelerate time-to-value by building a workable infrastructure the first time and every time.

With the HPE Ezmeral Runtime, enterprises now have a unified Kubernetes-based software solution for DevOps, CI/CD workflow, application modernization across hybrid cloud architecture, streamlining deployment, and operation with consistent orchestration and management. The platform acts as the control plane for container management and provides persistent container storage across multiple versions of open-source Kubernetes for container orchestration. The solution delivers a simpler, more scalable approach to modernizing applications. This is achieved using a scalable, code-driven container solution that, once assembled, can be configured within hours. This eliminates the complexities associated with implementing a K8s container platform across an enterprise data center and provides the automation of hardware and software configuration to quickly provision and deploy a containerized environment at scale.

The solution provides customers with greater efficiency, higher utilization, and bare-metal performance by "collapsing the stack" and eliminating the need for virtualization. Developers have secured on-demand access to their environment. They can develop apps and release code faster, with the portability of containers to build once and deploy anywhere. IT teams can manage multiple Kubernetes clusters with multitenant container isolation and data access, for any workload, from edge to core to cloud. The benefits of containers, beyond cloud-native microservices-architected stateless applications, can be extended by providing the ability to containerize monolithic stateful analytic applications with persistent data.

The combination of HPE Ezmeral Runtime paired with HPE Apollo compute and HPE Nimble Storage delivers a composable architecture that can rapidly deploy modern containers supporting the new application framework. Ultimately, this results in faster digital transformation for businesses by helping organizations drastically increase the velocity of application development and accelerate innovation. This Reference Architecture provides an overview of an enterprise-grade solution that helps organizations increase agility, simplify operations, and deliver a cloud-like experience while offering a compelling return on investment.

APPENDIX A: KUBEFLOW AND TESTS OF USE CASES

For Kubeflow installation, see the HPE Ezmeral Container Platform 5.2 documentation Kubeflow installation page. For HPE Ezmeral Runtime and uninstallation, see Uninstalling Kubeflow page.

We have used kubeflow_tutorials.zip, for Kubeflow use case testing which contains sample files for all of the included Kubeflow tutorials. The testing was done in a non-air-gapped environment.

Figure A1 shows Kubeflow architecture.

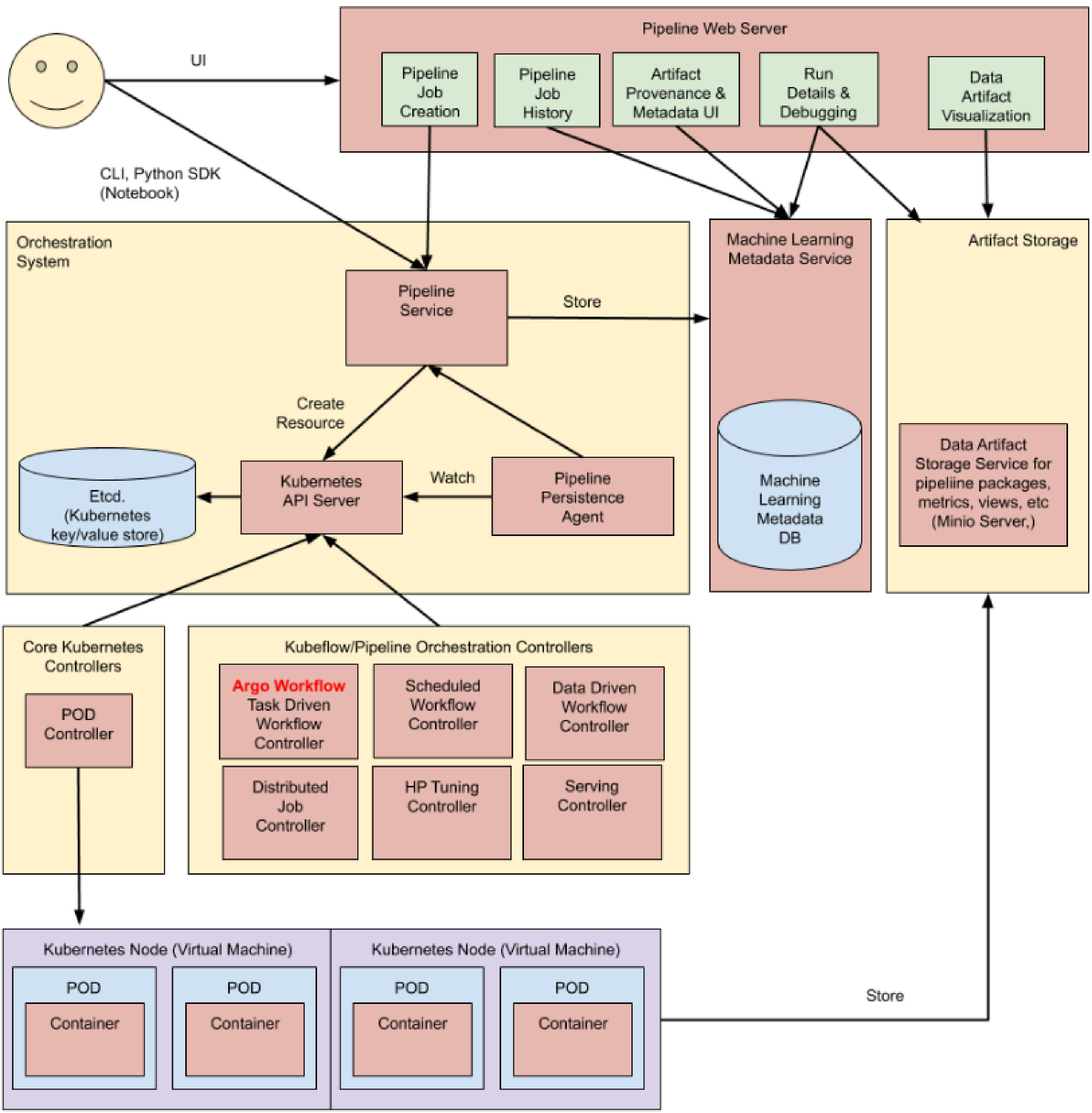


FIGURE A1. Kubeflow architectural diagram

KubeFlow components

The following table lists the components that Kubeflow can deploy.

TABLE A1. Kubeflow Components (Kubeflow Operator version 1.2)

Component	Version
Argo	2.3.0
Dex	2.22.0
Istio	1.3.1
Grafana	6.0.2
Jupyter Web Application	1.0.0
Katib	v1beta1
Kiali	1.4.0
Kfserving	0.3.0
Kubeflow Dashboard	1.0.0
ML Metadata	0.21.1
Notebook Controller	kf-ecp-5.3.0
Pipelines	1.0.4
PyTorch	1.0.0
Seldon	1.4.0
Spartakus	1.1.0
TensorFlow	1.0.0

Kubeflow components use cases GitHub issue summarization - Training with Jupyter

The steps followed in this Tutorial: GitHub Issue Summarization can be found in the official BlueData Documentation.

To begin with the Kubeflow examples:

[redacted]Log in to HPE Ezmeral Runtime.

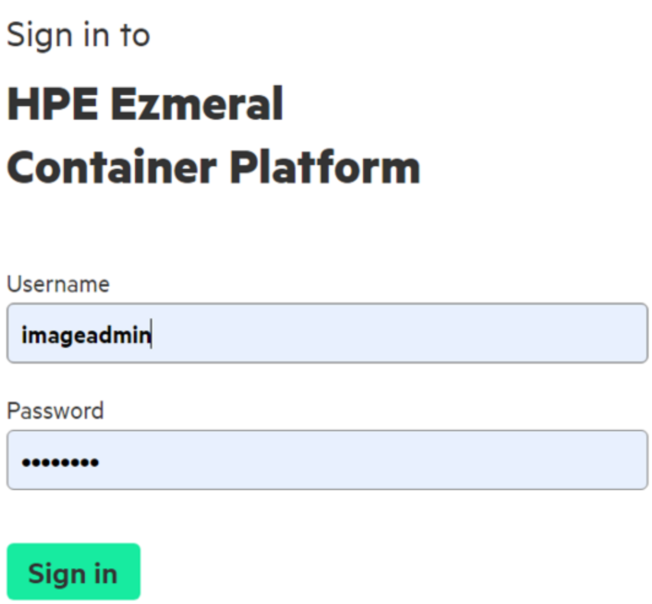


FIGURE A2. HPE Ezmeral Logging Page

[redacted]Select the Tenant and navigate to Kubeflow Dashboard.

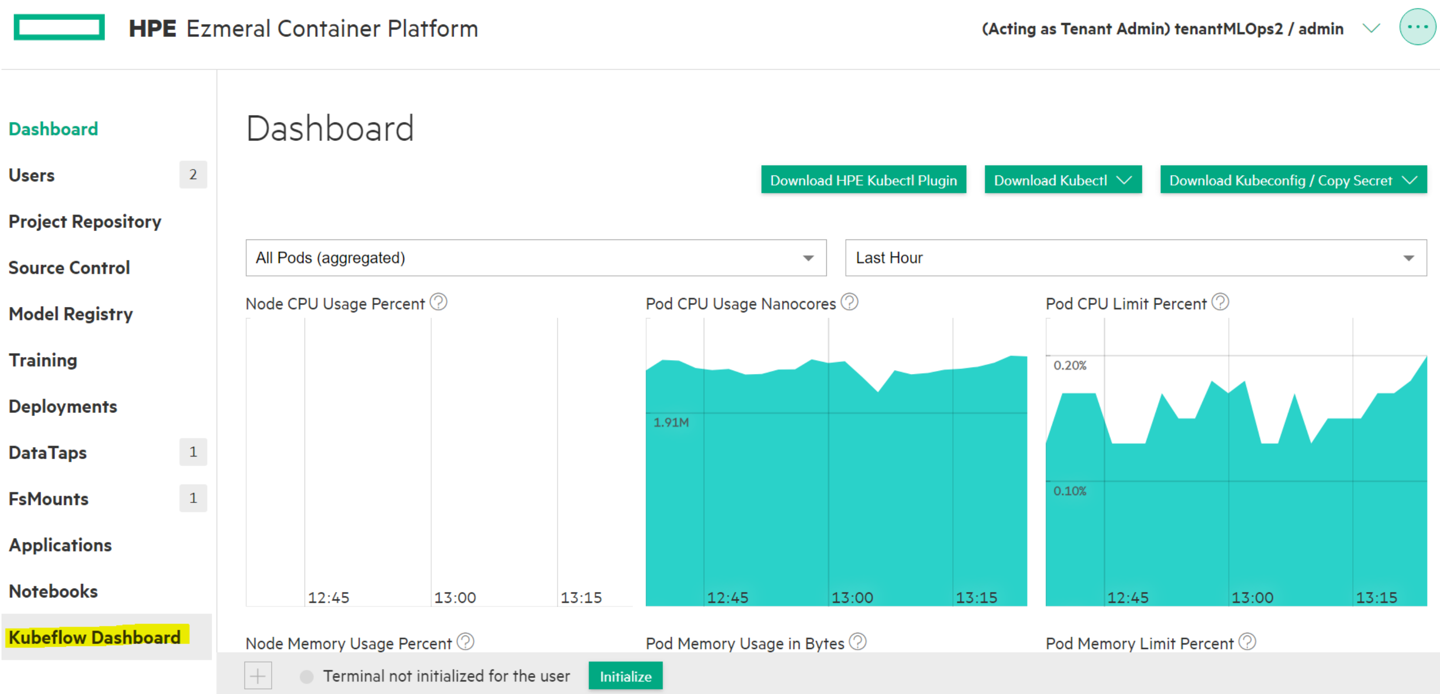


FIGURE A3. Navigating Kubeflow Dashboard

[redacted]A list of Notebook Servers in Kubeflow Dashboard appears.

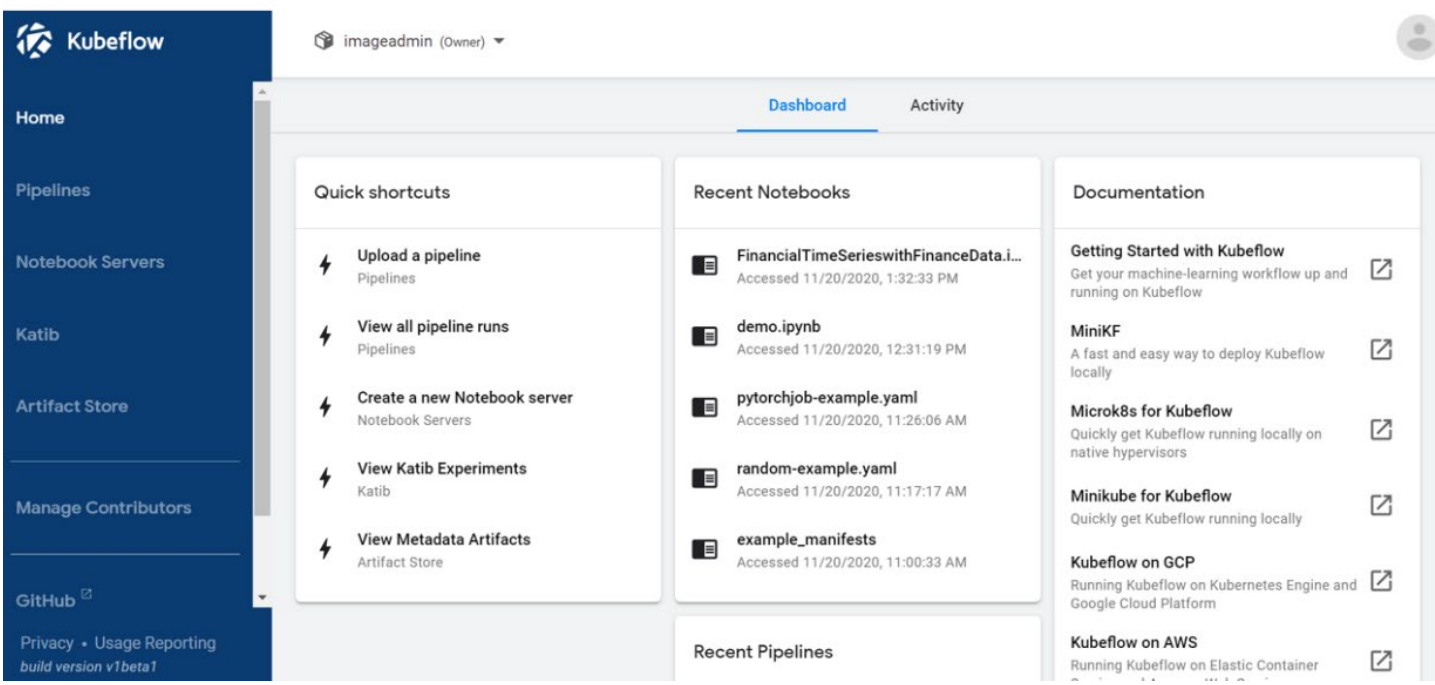


FIGURE A4. NB Servers in Kubeflow Dashboard [redacted] Select Notebook Servers and create one with the following instructions:

a. In the Data Volume section, select ReadWriteMany.
b. Change the Mount Path to a shorter name, such as /data.
c. Leave the Workspace Volume as-is.
d. Spawn the notebook by clicking Launch at the end of the page.

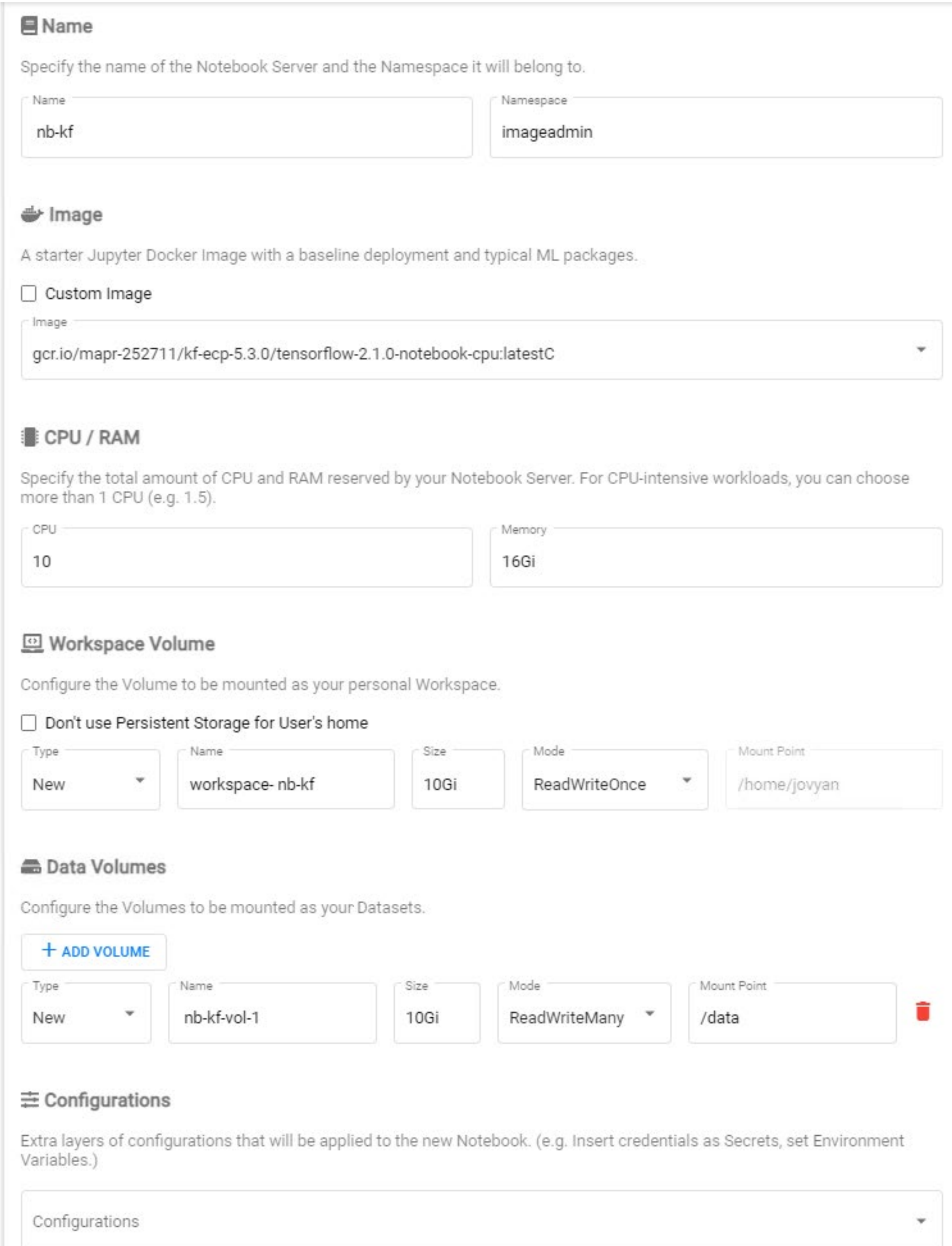


FIGURE A5. NB Servers

e. Connect to the Notebook Server created.

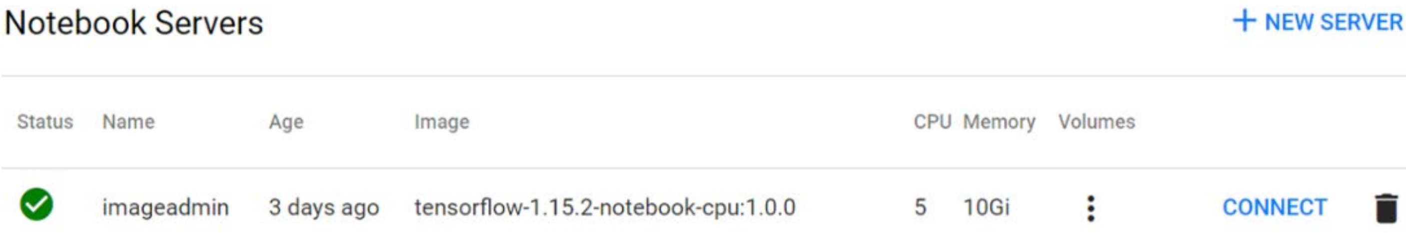


FIGURE A6. NB Servers in Ready State

[redacted]Open a new terminal from the Jupyter Hub and follow the steps below:

a. Download the kubeflow_tutorials.zip file, containing sample files for all of the included Kubeflow tutorials.

```
$ pwd
/home/jovyan

$ wget kubeflow_tutorials.zip
```

b. In a non-air-gapped environment only, execute the following commands to create the mapr-image-pull secrets and patch the Notebook. This must be done before pulling the Jupyter Notebook image for this tutorial.

```
$ kubectl apply -f imagepull-secrets.yaml
secret/mapr-imagepull-secrets created

$ kubectl patch serviceaccount default-editor -p '{"imagePullSecrets": [{"name": "mapr-imagepull-secrets"}]}'
serviceaccount/default-editor patched
```

c. Connect to the Notebook, then open a new terminal, and then clone the Kubeflow examples repo:

```
$ git clone https://github.com/mapr/kubeflow-examples.git
```

d. Return to the Jupyter folder list, and then open the file:

```
$ kubeflow-examples/github_issue_summarization/notebooks/Training.ipynb
```

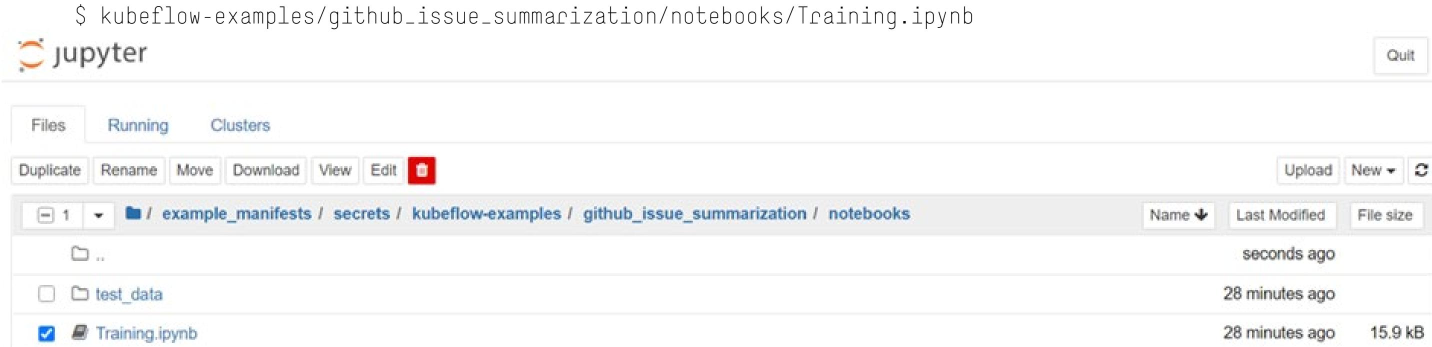


FIGURE A7. Training Notebook

e. In the Set path for the data_dir cell, change the data dir to:

```
%env DATA_DIR=/data
```

```
In [ ]: # Set path for data dir
        %env DATA_DIR=/data
```

f. In the pre-process data for the Deep Learning cell, comment out the magic function:

```
%%time
```

```
In [ ]: #%%time
        # Clean, tokenize, and apply padding / truncating such that each document length = 70
        #  also, retain only the top 8,000 words in the vocabulary and set the remaining words
        #  to 1 which will become common index for rare words
        body_pp = processor(keep_n=8000, padding_maxlen=70)
        train_body_vecs = body_pp.fit_transform(train_body_raw)
```

g. To execute each of the cells in the Notebook, either:

I. Click the Rerun button to run all the steps.

II. Click the Run button to manually run each step one at a time.

h. After training completes, use the Notebook terminal to copy the files to the MapR file system:

```
$ cd kubeflow-examples/github_issue_summarization/notebooks/
$ cp *.h5 /data/
$ cp *.dpkl /data/
```

```
$ cd kubeflow-examples/github_issue_summarization/notebooks/
$ ls
body_pp.dpkl              seq2seq_model_tutorial.h5  train_title_vecs.npy
Dockerfile                seq2seq_utils.py           tutorial_seq2seq.epoch01-val7.19677.hdf5
Dockerfile.estimator      test_data                  tutorial_seq2seq.epoch02-val6.43952.hdf5
environment_seldon_rest   title_pp.dpkl              tutorial_seq2seq.epoch03-val5.95483.hdf5
github-issues-data        train_body_vecs.npy        tutorial_seq2seq.epoch04-val5.61074.hdf5
IssueSummarization.py     trainer.py                 tutorial_seq2seq.epoch05-val5.37961.hdf5
Makefile                  Training.ipynb             tutorial_seq2seq.epoch06-val5.19002.hdf5
__pycache__               train.py                   tutorial_seq2seq.epoch07-val5.10789.hdf5
requirements.txt          train_test.py              tutorial_seq2seq.log
$ cp *.h5 /data/
$ cp *.dpkl /data/
```

GitHub issue summarization – Serving with Seldon

Download the kubeflow_tutorials.zip file, which contains sample files for all of the included Kubeflow tutorials.

This tutorial uses the following image `idzikovsky/sandbox:seldon-issuesum`.

[redacted] Apply the deployment by executing the following command.

```
$ kubectl apply -f seldon-issue-sum-deployment.yaml
Seldondeployment.machinelearning.seldon.io/issue-summarization created
```

[redacted] Verify that the Seldon deployment was created.

```
$ kubectl get sdep
NAME          AGE
issue-summarization   2m54s
```

[redacted]Verify that the pods are running.

```
$ kubectl get pods | grep classifier
issue-summarization-example-0-classifier-59fbc99c8-9zt6      2/2     Running 0       7m8s
```

[redacted]Connect to the Notebook and upload the file `seldon-request.py` to send a sample request to the server model.

[redacted]In the Notebook terminal, install the following Python dependencies.

```
$ pip install requests lxml --user
```

[redacted]Execute seldon-request.py with the following options.

```
$ python seldon-request.py http://istio-ingressgateway.istio-system.svc.cluster.local:80 imageadmin hp123456
imageadmin issue-summarization
{"data":{"names":["t:0"],"ndarray":[["add support for for"]]},"meta":{}}
```

[redacted]To delete the deployment, execute the following commands.

```
$ kubectl get sdep
NAME         AGE
issue-summarization  18m

$ kubectl delete sdep issue-summarization
Seldondeployment.machinelearning.seldon.io "issue-summarization" deleted
```

Training with TensorFlow (Financial series)

Before beginning this tutorial, download the kubeflow_tutorials.zip file, which contains sample files for all of the included Kubeflow tutorials.

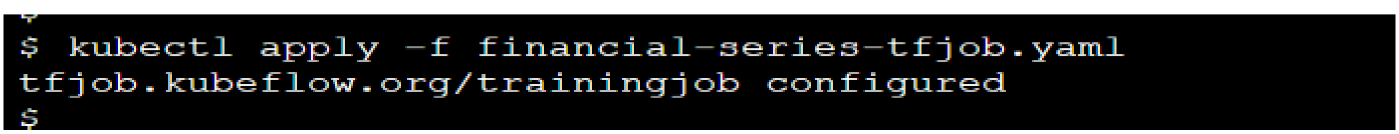
Step 1: Mount the MapRFS Directory

To mount the MapRFS directory:

[redacted]Obtain `pvc-tf-training-fin-series.yaml` from the zip file mentioned above for the Persistent Volume Claim (PVC).

[redacted]Apply the .yaml file to create the PVC:

```
$ kubectl apply -f pvc-tf-training-fin-series.yaml
```



[redacted] Verify that the PVC was created and is in the bound state:

```
$ kubectl get pvc
```

```
$ kubectl get pvc
NAME                    STATUS   VOLUME                                        CAPACITY   ACCESS MODES   STORAGECLASS
 AGE
data                    Bound    mapr-pv-e3686806-4252-404d-8f9a-0b00f85b7013   10Gi       RWX            hcp-mapr-cluster
 8d
imageadmin-vol-1        Bound    mapr-pv-d1c80ef1-1f69-46fe-a42c-d1fe93055d39   10Gi       RWX            hcp-mapr-cluster
 12d
kfnotebook1-vol-1       Bound    mapr-pv-a03cd16f-815a-4745-b1e7-14f87760c549   10Gi       RWO            hcp-mapr-cluster
 14d
pvcpy                   Bound    mapr-pv-9002ba6a-13e3-4cd7-8622-25430d381ed2   5Gi        RWX            hcp-mapr-cluster
 12d
pvctf                   Bound    mapr-pv-2feaa310-19f0-48ff-94c3-10e2a9b1a498   5Gi        RWX            hcp-mapr-cluster
 14d
voval-vol-1             Bound    mapr-pv-76d19390-61ce-4a1c-b8c6-b148bc092963   10Gi       RWO            hcp-mapr-cluster
 15d
```

Step 2: Exploration phase

To complete the exploration phase:

[redacted] Open the Kubeflow web interface.

[redacted] Follow the procedure based on the example described here (link opens an external website in a new browser tab/window) to spawn the Notebook. Do not create a Data Volume; simply select New Workspace and leave the remaining options set to their default values.

[redacted] Connect to the Notebook server, and then open a terminal.

[redacted] Execute the following command:

```
$ curl
https://raw.githubusercontent.com/kubeflow/examples/master/financial_time_series/Financial%20Time%20Series%20w
ith%20Finance%20Data.ipynb --output FinancialTimeSerieswithFinanceData.ipynb
```

```
$ curl https://raw.githubusercontent.com/kubeflow/examples/master/financial_time_series/Financial%20Time%20Series%20with%20F
inance%20Data.ipynb --output FinancialTimeSerieswithFinanceData.ipynb
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 47939  100 47939    0     0   178k      0 --:--:-- --:--:-- --:--:--  178k
$ 
```

[redacted] Open the uploaded Notebook.

FIGURE A8. Financial Time Series with Finance Data Notebook

[redacted] Walkthrough the Notebook step by step to better understand the problem and suggested solution(s).

Step 3: Training phase

To complete the training phase:

[redacted]If required, use the already-pushed image nskopiuk/sandbox:tensorflowimage-finseries, else skip to step 3 if not.

[redacted]Build and push the image.

```
$ git clone https://github.com/mapr/private-kubeflow-examples.git cd private-kubeflow-
examples/financial_time_series/tensorflow_model export TRAIN_PATH= <your-docker-
username>/sandbox:tensorflowimage-finseries docker build -t $TRAIN_PATH . docker push $TRAIN_PATH
```

[redacted]Obtain the secrets file user-gcp-sa from the zip file mentioned above (required for this example).

```
$ kubectl create secret generic user-gcp-sa --from-file user-gcp-sa.json
```

[redacted]Apply the TF-training job using the .yaml file from the zip file mentioned above.

```
$ kubectl apply -f financial-series-tfjob.yaml
```

```
$ kubectl apply -f financial-series-tfjob.yaml
tfjob.kubeflow.org/trainingjob configured
$
```

[redacted]Verify that user-gcp-sa is mounted.

```
$ kubectl exec -it trainingjob-ps-0 -- /bin/sh
# ls /auth/
user-gcp-sa.json
```

[redacted]Verify that the TF-job trainingjob was successfully created.

```
$ kubectl get tfjobs
```

```
$ kubectl get tfjobs
NAME          STATE       AGE
trainingjob   Succeeded   8d
$
```

[redacted]Verify that pods were created, run, and completed.

```
$ kubectl get pods | grep trainingjob
```

```
$ kubectl get pods | grep trainingjob
trainingjob-ps-0                      0/1     Completed   0          8d
trainingjob-worker-0                  0/1     Completed   0          8d
$
```

[redacted]Check the logs to see the training job description.

```
$ kubectl logs trainingjob-ps-0 tensorflow
```

```
$ kubectl logs trainingjob-ps-0 tensorflow
INFO:root:getting the ML model...
INFO:root:getting the data...
INFO:root:generating training data...
INFO:root:defining the training objective...
2020-11-23 22:03:46.400338: I tensorflow/core/platform/cpu_feature_guard.cc:140] Your CPU supports instructions that this Te
nsorFlow binary was not compiled to use: SSE4.1 SSE4.2 AVX AVX2 AVX512F FMA
INFO:root:training the model...
INFO:root:training took 19.19 sec
INFO:root:validating model on test set...
INFO:root:Exporting model for tensorflow-serving...
INFO:tensorflow:Assets added to graph.
INFO:tensorflow:Assets added to graph.
INFO:tensorflow:No assets to write.
INFO:tensorflow:No assets to write.
INFO:tensorflow:SavedModel written to: b'model/1/saved_model.pb'
INFO:tensorflow:SavedModel written to: b'model/1/saved_model.pb'
INFO:root:copy files to /data/model/1
5000 0.5607639
10000 0.5755208
15000 0.5946181
20000 0.6145833
25000 0.6302083
30000 0.6449653
Precision =  0.9142857142857143
Recall =  0.2222222222222222
F1 Score =  0.35754189944134074
Accuracy =  0.6006944444444444
$
```

Serving a TensorFlow model with KFServing (Financial series)

Before beginning this tutorial, download the kubeflow_tutorials.zip file, which contains sample files for all of the included Kubeflow tutorials.

Step 1: Create and apply the YAML file

To complete the tutorial:

[redacted]Obtain the serving YAML file from the zip file mentioned above.

[redacted]Apply the file.

```
$ kubectl apply -f financial-series-serving.yaml
```

```
$ kubectl apply -f financial-series-serving.yaml
inferenceservice.serving.kubeflow.org/finance-sample unchanged
$
```

[redacted]Verify that the inference service, revision, and relevant pods have been created.

```
KSVC:
$ kubectl get ksvc
```

```
$ kubectl get ksvc
NAME                                URL                                                                 LATESTCREATED
                 LATESTREADY                            READY   REASON
finance-sample-predictor-default   http://finance-sample-predictor-default.imageadmin.example.com   finance-sample-predictor
-default-mhjwg   finance-sample-predictor-default-mhjwg   True
$
```

```
Revision:
$ kubectl get revision
```

```
$ kubectl get revision
NAME                                   CONFIG NAME                        K8S SERVICE NAME                        GENERAT
ION    READY   REASON
finance-sample-predictor-default-mhjwg   finance-sample-predictor-default   finance-sample-predictor-default-mhjwg   1
      True
$
```

```
Inference Services:
$ kubectl get inferenceservices
```

```
$ kubectl get inferenceservices
NAME             URL                                                                    READY   DEFAULT TRAFFIC   CANARY TR
AFFIC   AGE
finance-sample   http://finance-sample.imageadmin.example.com/v1/models/finance-sample   True    100
        5d16h
```

```
Pods:
$ kubectl get pods | grep finance-sample
```

```
$ kubectl get pods | grep finance-sample
finance-sample-predictor-default-mhjwg-deployment-7549f689ftktm   2/2     Running     0          5d16h
$
```

[redacted]Verify that virtual services have been created.

```
$ kubectl get virtualservices | grep finance-sample
```

```
$ kubectl get virtualservices | grep finance-sample
finance-sample                        [kubeflow-gateway.kubeflow knative-serving/cluster-local-gateway]   [finance-sample.
imageadmin.example.com finance-sample.imageadmin.svc.cluster.local]
                                                                          5d16h
finance-sample-predictor-default        [knative-serving/cluster-local-gateway kubeflow/kubeflow-gateway]   [finance-sample-
predictor-default.imageadmin finance-sample-predictor-default.imageadmin.example.com finance-sample-predictor-default.imagea
dmin.svc finance-sample-predictor-default.imageadmin.svc.cluster.local]   5d16h
finance-sample-predictor-default-mesh    [mesh]                                                           [finance-sample-
predictor-default.imageadmin finance-sample-predictor-default.imageadmin.svc finance-sample-predictor-default.imageadmin.svc
.cluster.local]                                                           5d16h
```

Step 2: Perform inferences against the served model

To send a request to the model:

[redacted]Obtain kfserving-request.py from the zip file mentioned above.

[redacted]Install the following Python dependencies.

```
$ pip install requests lxml -user
```

[redacted]Launch kfserving-request.py with the following options.

```
python kfserving-request.py <base_url> <login> <password> <profile_name>
For example:
$ python kfsrving-request.py http://cv-hcp-lb1.qa.lab:10046 imageadmin 12341234 imageadmin
```

[redacted]To send requests from the Jupyter Notebook terminal, use the ingressgateway address (istio-ingressgateway.svc.cluster.local). For example:

```
$ python kfserving-request.py http://istio-ingressgateway.istio-system.svc.cluster.local:80 imageadmin
12341234 imageadmin
```

The output will be similar to the following:
200
```
{'predictions': [{'model-version': '1', 'prediction': 0}]}
```

Training a PyTorch model (PyTorch MNIST)

Download the kubeflow_tutorials.zip file, which contains sample files for all of the included Kubeflow tutorials, if desired.

1. Obtain `vim pytorch-mnist-ddp-cpu.yaml` from the zip file mentioned above.

```
$ vim pytorch-mnist-ddp-cpu.yaml
```

[redacted] Create the PyTorch Job.

```
$ kubectl apply -f pytorch-mnist-ddp-cpu.yaml
pytorchjob.kubeflow.org/pytorch-mnist-ddp-cpu created
```

[redacted] Verify that the PyTorch job was created.

```
$ kubectl get pytorchjobs
NAME          STATE    AGE
pytorch-mnist-ddp-cpu  Succeeded  108s
```

[redacted] Verify that relevant pods have been created.

```
$ kubectl get pods -l pytorch-job-name=pytorch-mnist-ddp-cpu
NAME               READY  STATUS   RESTARTS  AGE
pytorch-mnist-ddp-cpu-master-0  0/1   Completed  0     10m
pytorch-mnist-ddp-cpu-worker-0  0/1   Completed  0     10m
pytorch-mnist-ddp-cpu-worker-1  0/1   Completed  0     10m
pytorch-mnist-ddp-cpu-worker-2  0/1   Completed  0     10m
```

[redacted] Inspect the logs to observe PyTorch training progress:

```
$ PODNAME=$(kubectl get pods -l pytorch-job-name=pytorch-mnist-ddp-cpu,pytorch-replica-type=master,pytorch-
replica-index=0 -o name)
kubectl logs -f ${PODNAME}$
```

[redacted] Verify the PyTorch Job pod statuses, and wait until all pods show the status **Completed**.

```
$ kubectl get pods -l pytorch-job-name=pytorch-mnist-ddp-cpu
```

[redacted] Check the logs again to verify that the output contains the following information.

```
$ kubectl logs -f ${PODNAME}$
Using CUDA
Using distributed PyTorch with gloo backend
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Processing...
Done!
Train Epoch: 1 [58880/60000 (98%)]   loss=0.2060
Train Epoch: 1 [59520/60000 (99%)]   loss=0.0644

accuracy=0.9664
```

[redacted] Use the `describe` command to check PyTorch job status.

```
$ kubectl describe pytorchjobs pytorch-mnist-ddp-cpu
```

The output should look similar to the following:

```
Type     Reason                   Age    From              Message
----     ------                   ----   ----              -------
Normal   SuccessfulCreatePod      16m    pytorch-operator  Created pod: pytorch-mnist-ddp-cpu-worker-0
Normal   SuccessfulCreatePod      16m    pytorch-operator  Created pod: pytorch-mnist-ddp-cpu-worker-1
Normal   SuccessfulCreatePod      16m    pytorch-operator  Created pod: pytorch-mnist-ddp-cpu-worker-2
Normal   SuccessfulCreatePod      16m    pytorch-operator  Created pod: pytorch-mnist-ddp-cpu-master-0
Normal   SuccessfulCreateService  16m    pytorch-operator  Created service: pytorch-mnist-ddp-cpu-master-0
Normal   PyTorchJobSucceeded      5m57s  pytorch-operator  PyTorchJob pytorch-mnist-ddp-cpu is successfully completed.
```

Additional information

After training is complete, the files will be located in the MapR file system.

```
$ ssh root@<controller_node_ip_address>
bdmapr --root bash
hadoop fs -ls <volumePath>/pytorch/model
```

Result:

Found 1 items

```
-rw-r--r--   3 root    88548 2020-07-15 09:36 /mapr-csi/k8s-10--siaitqgucc/pytorch/model/model_cpu.dat
```

Sample Pipeline in the pipelines interface

Refer to an example from the Kubeflow documentation (The link opens an external website in a new browser tab/window).

[redacted]Open the Kubeflow dashboard (see Accessing the Kubeflow Dashboard), and then select Pipelines.

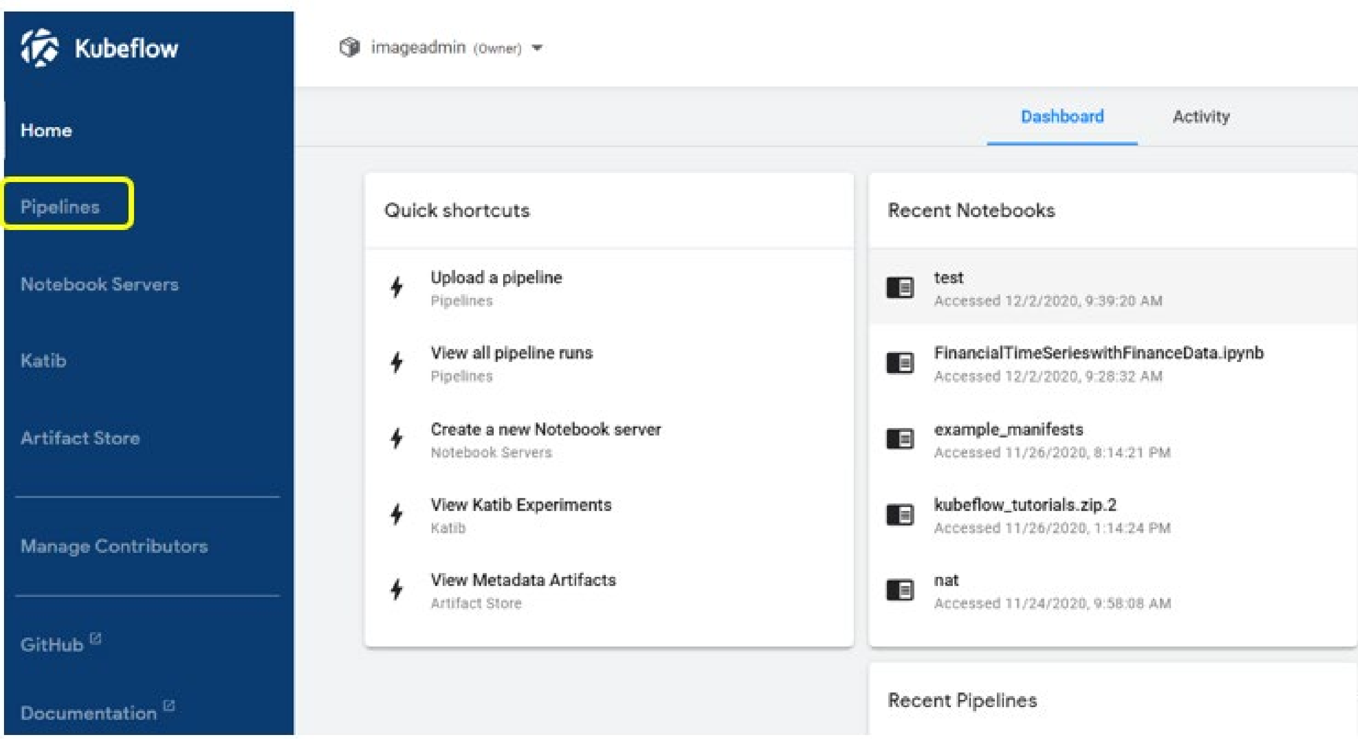


FIGURE A9. Kubeflow Pipeline

[redacted]Click the sample name [Tutorial] DSL- Control Structures.

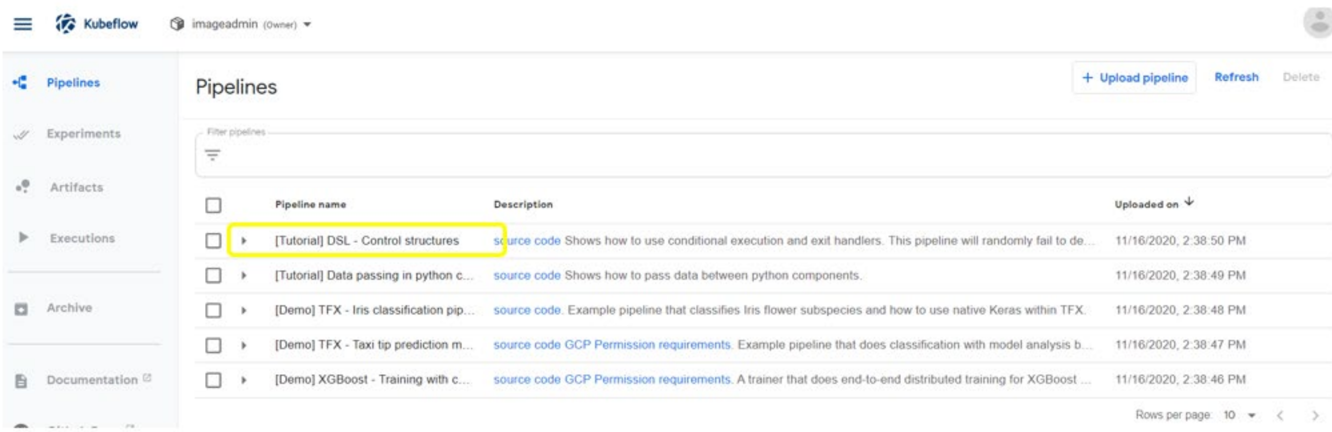


FIGURE A10. Kubeflow Pipeline Dashboard

[redacted]Click **Experiments**, and then follow the on-screen prompts.

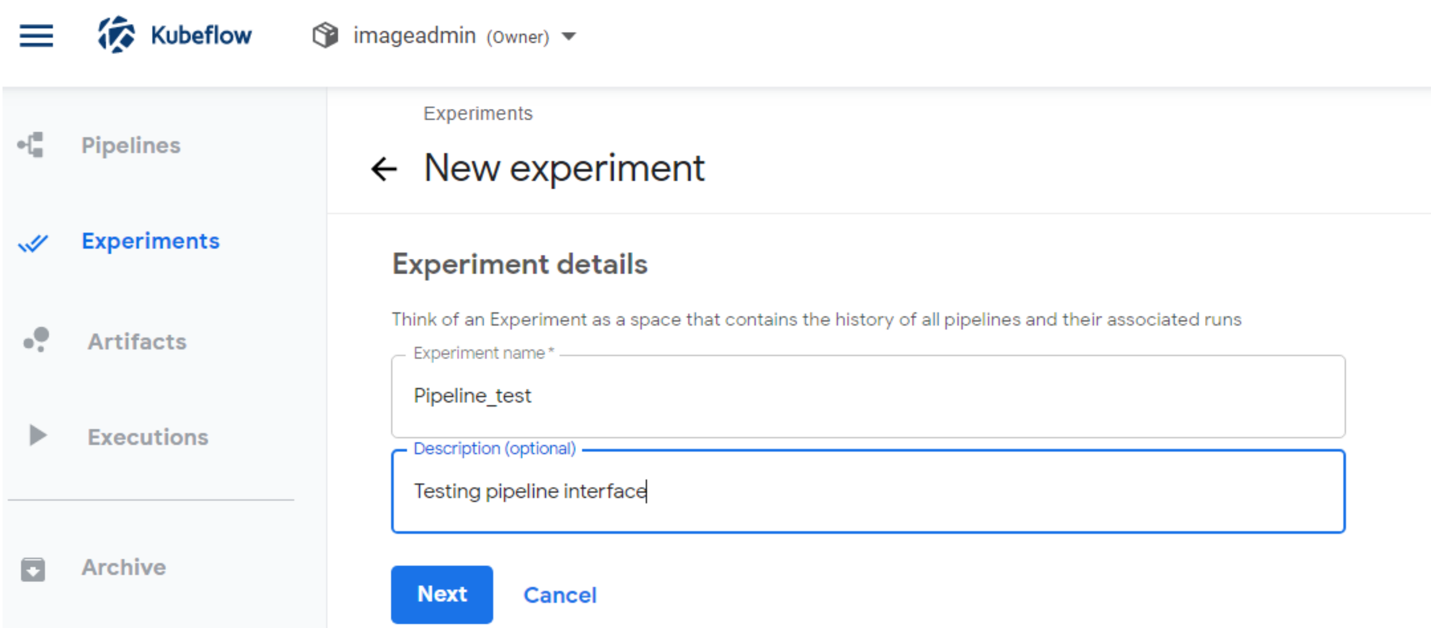


FIGURE A11. Create new Experiment

[redacted]Enter the Run details and click **Start**.

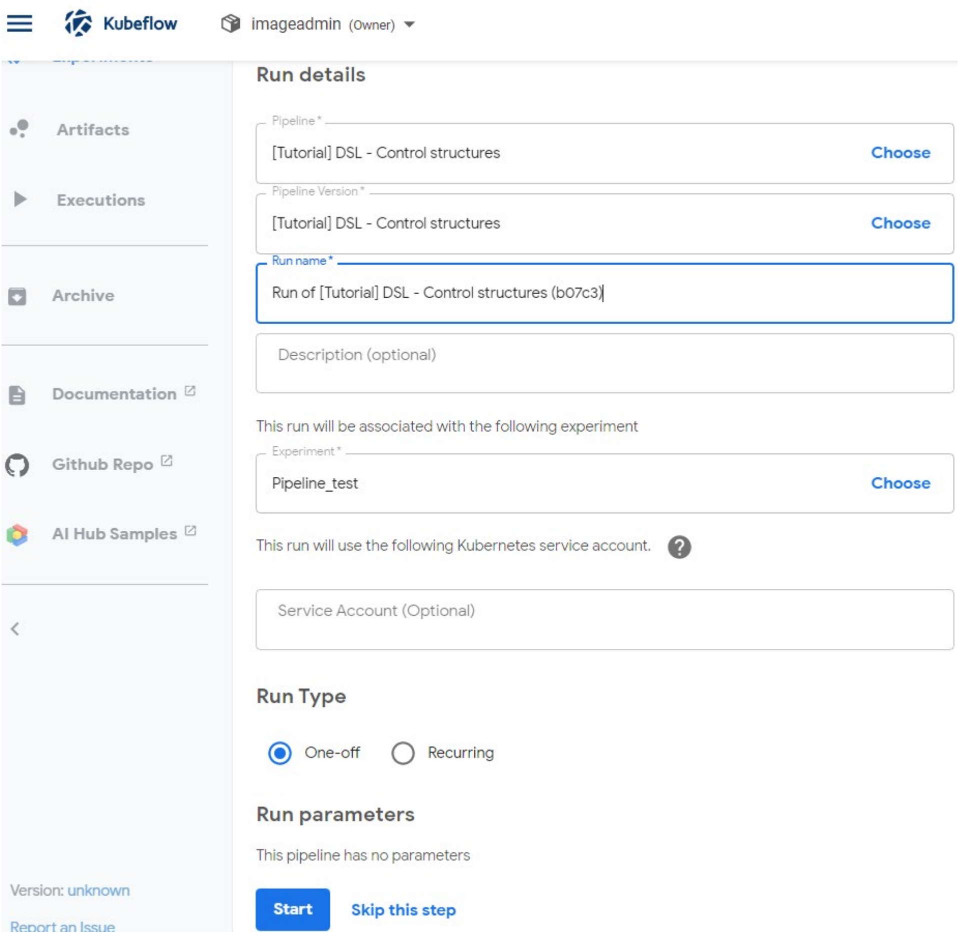


FIGURE A12. Create Run

[redacted]Select the run that was created in step 4 on the Experiments dashboard.

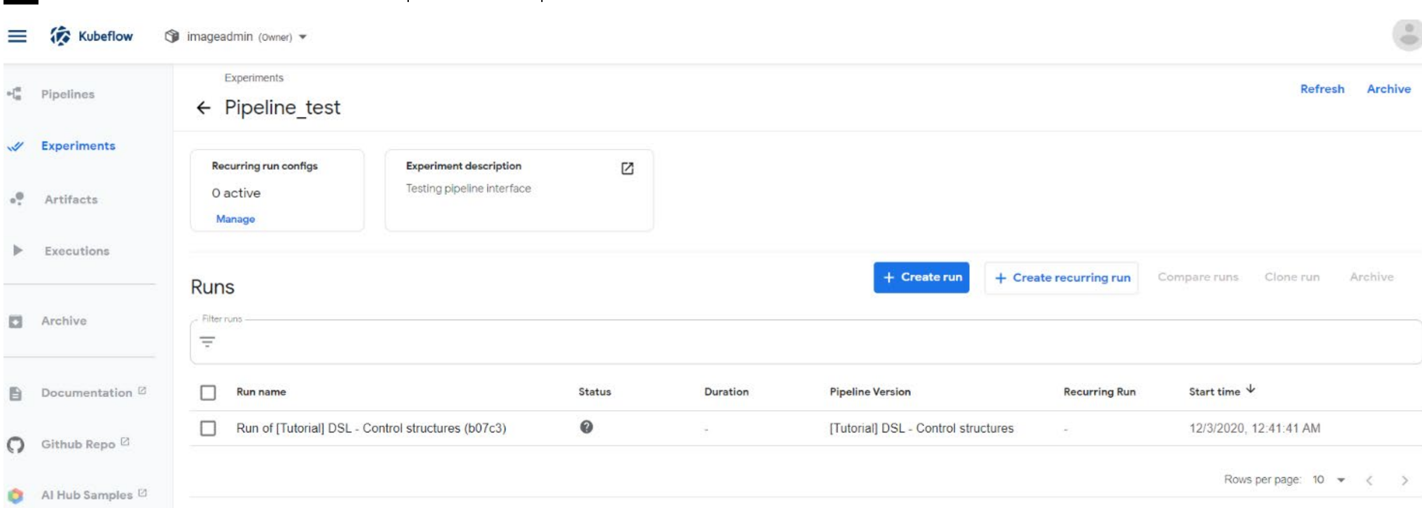


FIGURE A13. Select Run on Experiments Dashboard [redacted] Explore the graph and other aspects of the run by clicking the graph components and other interface elements.

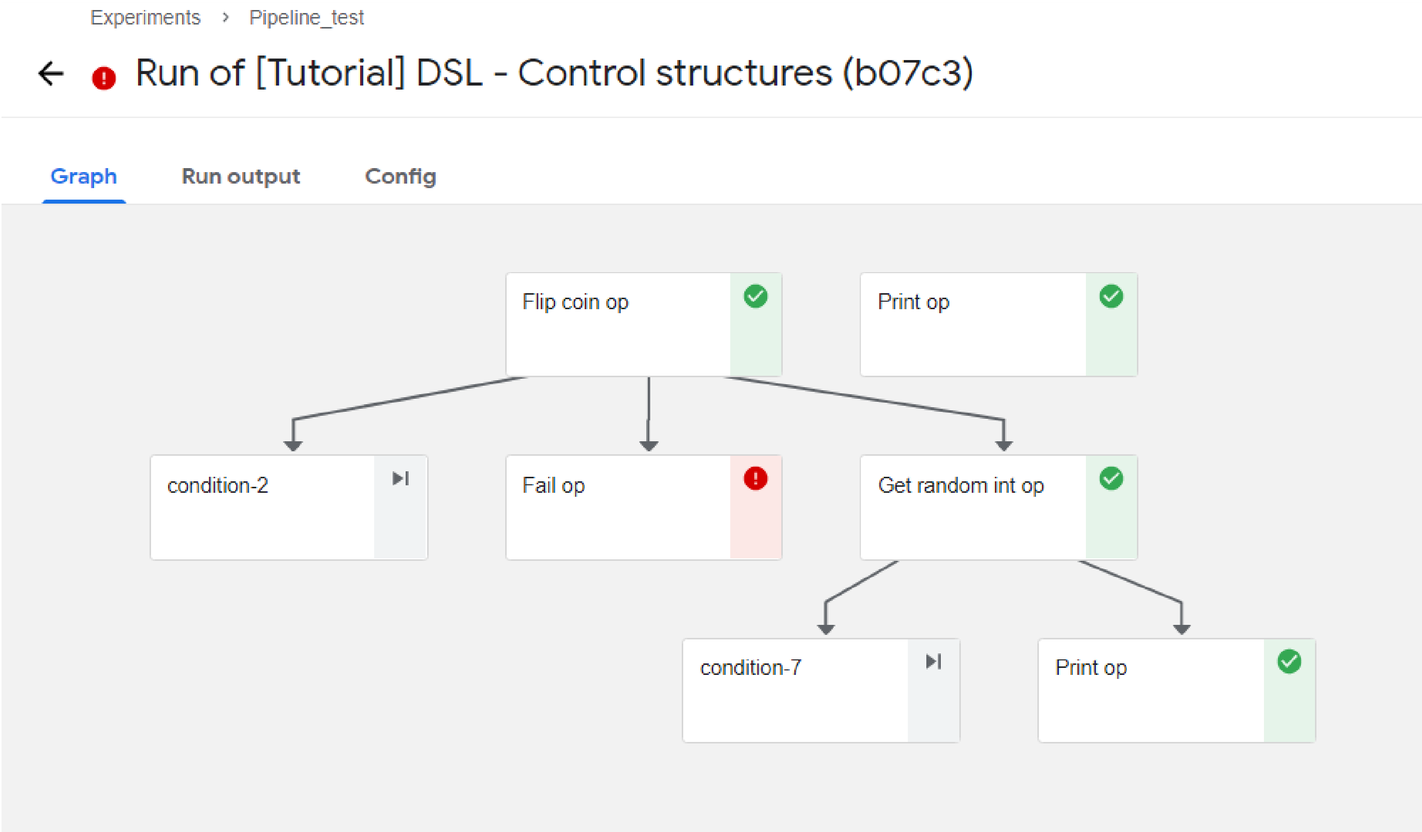


FIGURE A14. Run Graph

Running a pipeline in Jupyter Notebook

Before beginning this tutorial, download the kubeflow_tutorials.zip file, which contains sample files for all of the included Kubeflow tutorials.

[redacted] Create a Jupyter Notebook.

[redacted] Connect to the Notebook, and then click **New** Terminal.

[redacted] Clone the kubeflow/pipelines repo:

```
$ git clone https://github.com/kubeflow/pipelines.git
```

[redacted]Return to theFiles tab, and then open the Notebook:

```
$ pipelines/sam ples/core/lightweight_component.ipynb
```

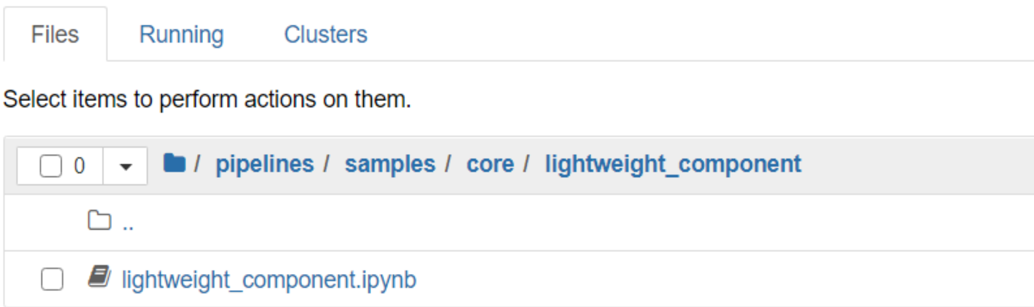


FIGURE A15. Lightweight Component Notebook

[redacted]Execute each cell in the Notebook until it is finished.

[redacted]Follow the Notebook link to the created experiment in the Pipelines interface.

[redacted]Follow the Notebook link to the created run in the Pipelines interface.

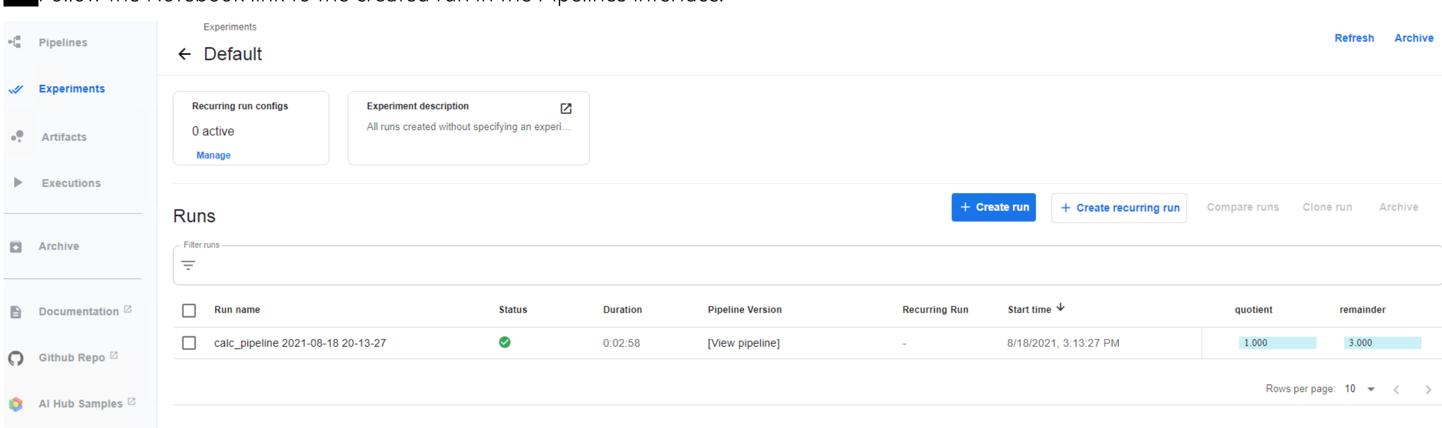


FIGURE A16. Calc Pipeline Run

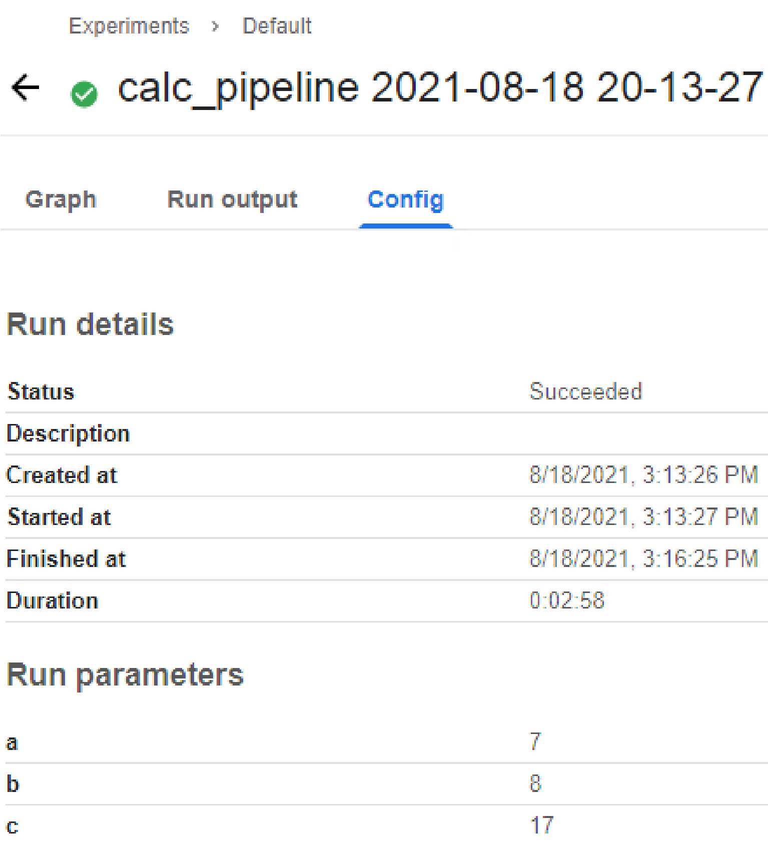


FIGURE A17. Run Details

Here are some general pipeline viewing steps from the Pipelines Dashboard interface:

[redacted]Open the **Experiments** page in the Pipelines dashboard.

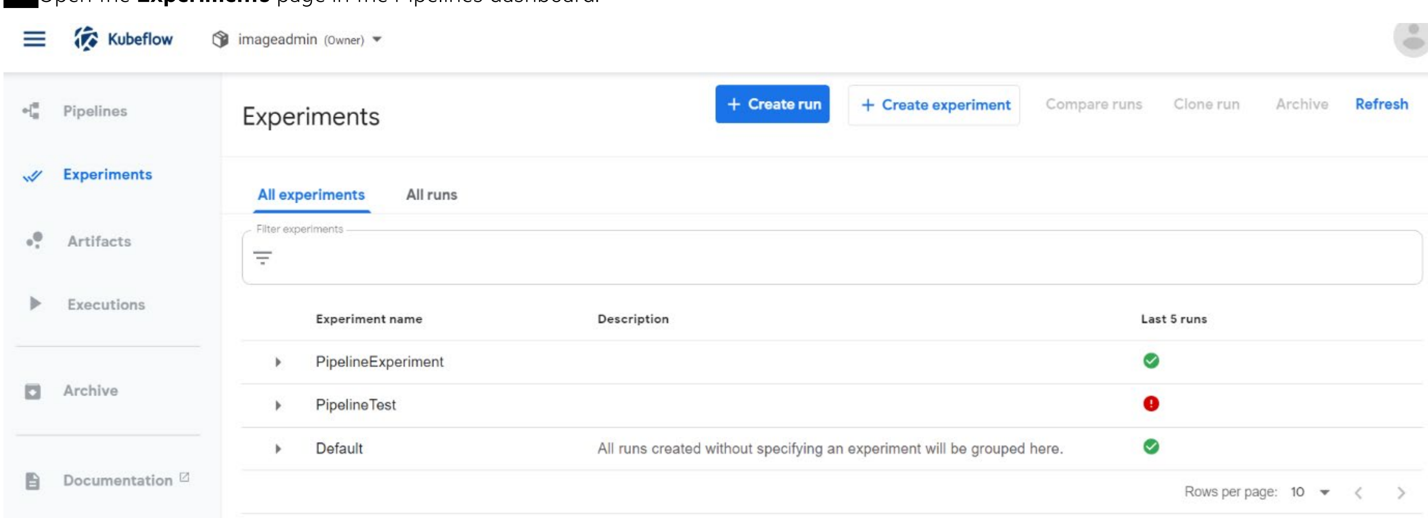


FIGURE A18. Experiments Dashboard

- In the **All experiments** tab, expand the Default group, and then view the pipeline graph and details per step by clicking the appropriate (view pipeline) link.
- In the **All runs** tab, click the name of the run to view the **Graph**, **Run output**, and **Config** tabs.

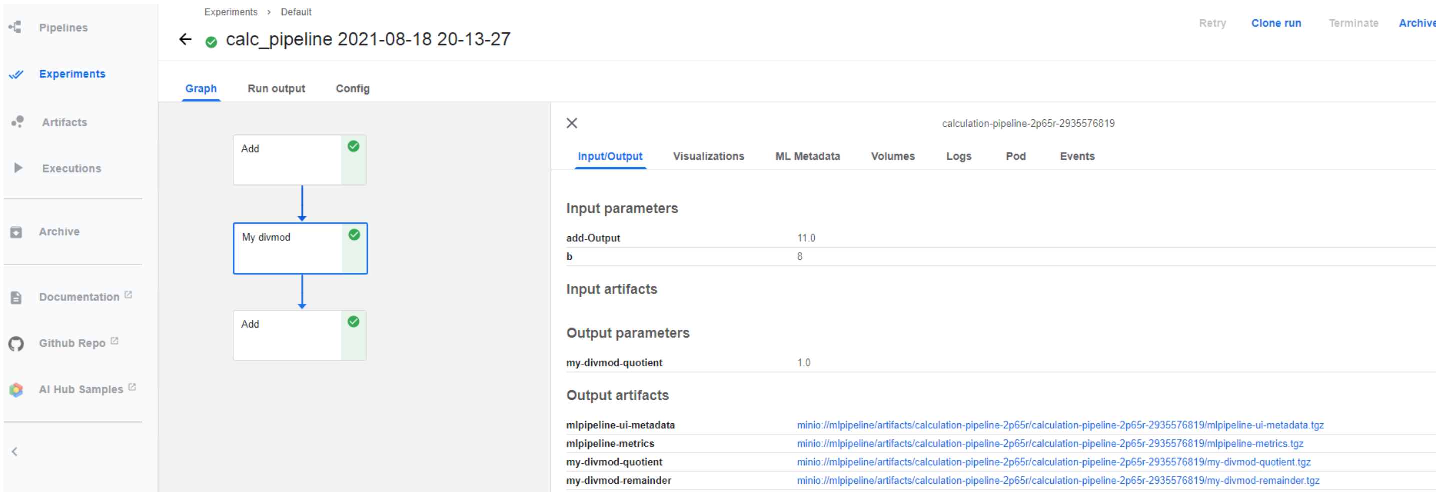


FIGURE A19. Pipeline Run Graph

Katib Hyperparameter Tuning

Example 1: TensorFlow

- Download the Kubeflow tutorials zip file which contains sample files for all of the included Kubeflow tutorials.
- Edit the tensorflow-example.yaml to put the following on the pod template.

```
metadata:
 annotations:
  sidecar.istio.io/inject: "false"
```

- Deploy the example.

```
$ kubectl apply -f tensorflow-example.yaml
```

- Open the Kubeflow Dashboard, and then select Katib.

[redacted] Click the left menu button, and then go to **HP** → **Monitor**.

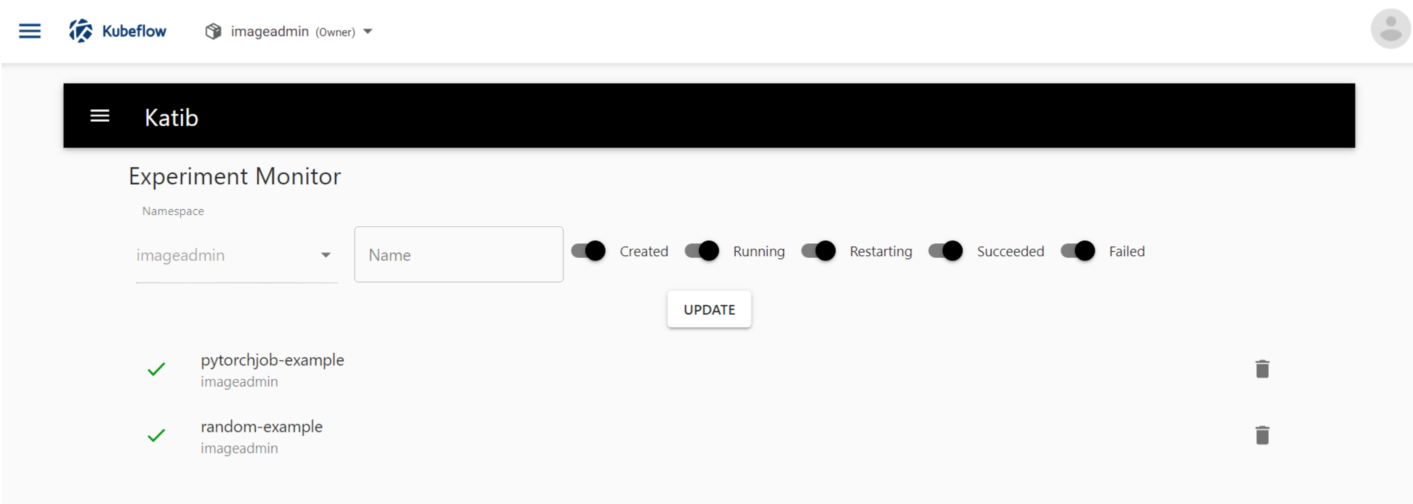
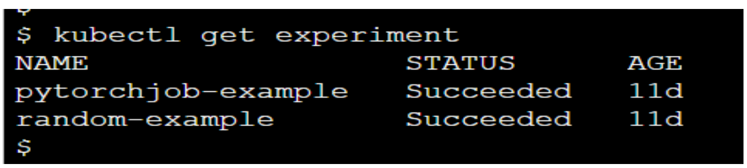


FIGURE A20. Katib HP Monitor

[redacted] Click the experiment name, and then observe the running trials.

[redacted] Check the experiment status:

```
$ kubectl get experiment
```



```
$ kubectl get experiment
NAME                 STATUS      AGE
pytorchjob-example   Succeeded   11d
random-example       Succeeded   11d
$
```

[redacted]Check the experiment trials.

```
$ kubectl get trial
```

```
$
$ kubectl get trial
NAME                           TYPE        STATUS   AGE
pytorchjob-example-2flzmzb9    Succeeded   True     11d
pytorchjob-example-4qhbxrl9    Succeeded   True     11d
pytorchjob-example-55gj2kpx    Succeeded   True     11d
pytorchjob-example-55w2j4x6    Succeeded   True     11d
pytorchjob-example-58f9qtwc    Succeeded   True     11d
pytorchjob-example-5gwd5srs    Succeeded   True     11d
pytorchjob-example-7xsvgqgz    Succeeded   True     11d
pytorchjob-example-dqtmjp8l    Succeeded   True     11d
pytorchjob-example-hkjg2hzm    Succeeded   True     11d
pytorchjob-example-vtd47rnv    Succeeded   True     11d
pytorchjob-example-zxlzvb6j    Succeeded   True     11d
pytorchjob-example-zzsglc9r    Succeeded   True     11d
random-example-25gr9zqw        Succeeded   True     11d
random-example-8cmpkkk9        Succeeded   True     11d
random-example-c5h242cr        Succeeded   True     12d
random-example-d9lsc7jl        Succeeded   True     11d
random-example-dmwcldxb        Succeeded   True     11d
random-example-gmb2ncgq        Succeeded   True     11d
random-example-kdp6pqjp        Succeeded   True     12d
random-example-lptj9mkv        Succeeded   True     11d
random-example-lpv84kvs        Succeeded   True     12d
random-example-m9kzdnmw        Succeeded   True     11d
random-example-nkkrz4nr        Succeeded   True     11d
random-example-sp6hhs8s        Succeeded   True     11d
$
```

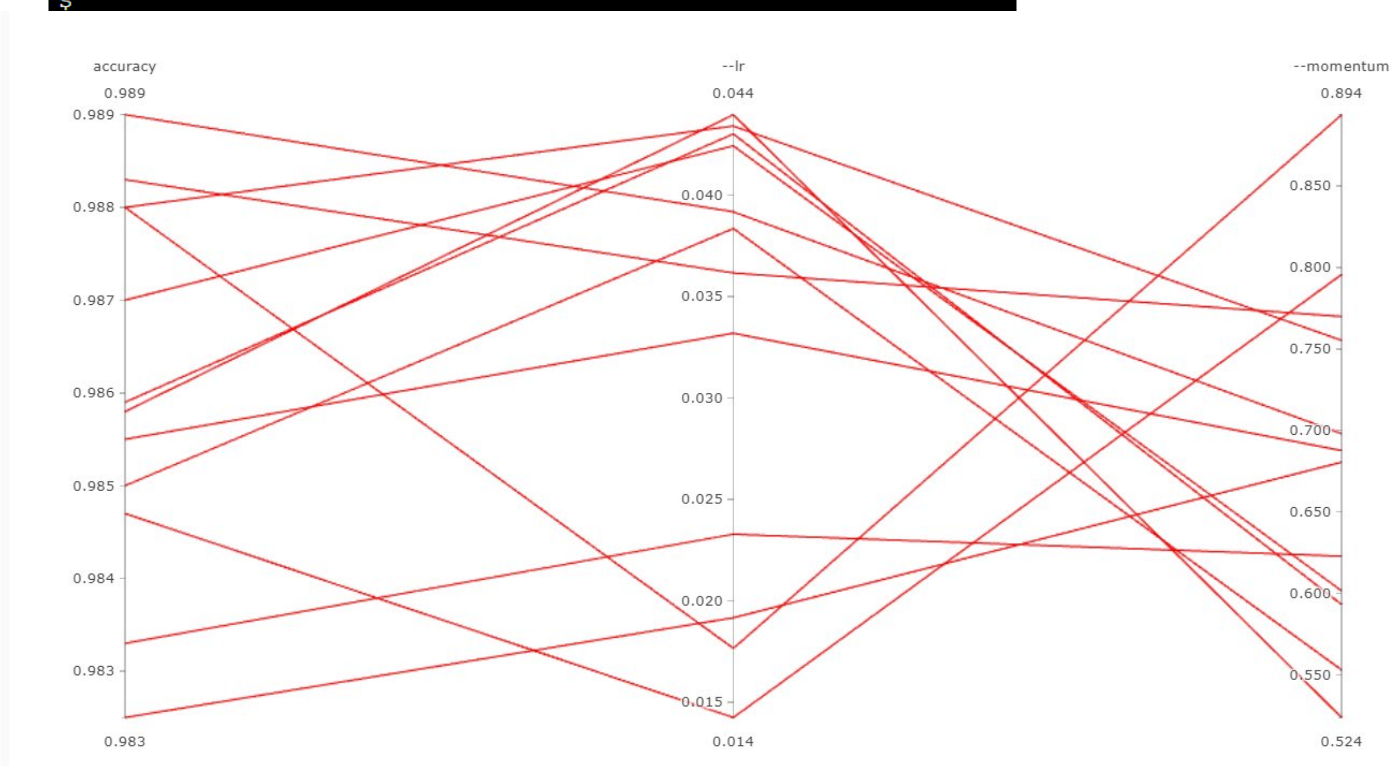


FIGURE A21. Katib HP Tuning

Example 2: Random algorithm

The following hyperparameters can be tuned:

```
--lr - learning rate
--num-layers - Number of layers in the neural networks
--optimizer
```

To launch an experiment using the random algorithm example:

- Download the kubeflow-tutorials.zip (link opens an external website in a new browser tab/window).
- Edit `random-example.yaml` to put the following on the pod template.

```
metadata:
 annotations:
  sidecar.istio.io/inject: "false"
```

- Deploy the example:

```
$kubectl apply -f random-example.yaml
```

This example embeds the hyperparameters as arguments. Hyperparameters can be embedded in other ways (e.g. by using environment variables) by using the template defined in the TrialTemplate, GoTemplate, and RawTemplate section of the yaml file. The template uses the Go template format (link opens an external website in a new browser tab/window).

This example randomly generates the following hyperparameters:

```
--lr - Learning rate (type: double).
--num-layers - Number of layers in the neural network (type: integer).
--optimizer - Optimizer (type: categorical).
```

[redacted]Check the experiment status:

```
$ kubectl describe experiment random-example
```

```
$ kubectl describe experiment random-example
Name:         random-example
Namespace:    imageadmin
Labels:       controller-tools.k8s.io=1.0
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"kubeflow.org/v1alpha3","kind":"Experiment","metadata":{"annotations":{},"labels":{"controller
-tools.k8s.io":"1.0"},"name":"...
API Version:  kubeflow.org/v1alpha3
Kind:         Experiment
Metadata:
  Creation Timestamp:  2020-11-20T19:17:36Z
  Finalizers:
    update-prometheus-metrics
  Generation:        1
  Resource Version:  3303458
  Self Link:         /apis/kubeflow.org/v1alpha3/namespaces/imageadmin/experiments/random-example
  UID:               819d66c8-a977-46e9-8bfb-daf61c7a4e38
Spec:
  Algorithm:
    Algorithm Name:        random
  Max Failed Trial Count:  3
  Max Trial Count:         12
  Metrics Collector Spec:
    Collector:
      Kind:  StdOut
  Objective:
    Additional Metric Names:
      Train-accuracy
    Goal:                   0.99
    Objective Metric Name:  Validation-accuracy
    Type:                   maximize
  Parallel Trial Count:     3
  Parameters:
    Feasible Space:
      Max:           0.03
      Min:           0.01
    Name:            --lr
    Parameter Type:  double
    Feasible Space:
      Max:           5
      Min:           2
    Name:            --num-layers
    Parameter Type:  int
    Feasible Space:
      List:
        sgd
        adam
        ftrl
    Name:            --optimizer
    Parameter Type:  categorical
  Resume Policy:     LongRunning
  Trial Template:
    Go Template:
      Raw Template:  apiVersion: batch/v1
kind: Job
metadata:
  name: {{.Trial}}
  namespace: {{.NameSpace}}
spec:
  template:
    spec:
      containers:
      - name: {{.Trial}}
        image: docker.io/kubeflowkatib/mxnet-mnist
        command:
```

Example 3: PyTorch

1. Download the Kubeflow tutorials zip file which contains sample files for all of the included Kubeflow tutorials.

2. Edit the YAML to point to put the following on the pod template.

```
metadata:D
 annotations:
  sidecar.istio.io/inject: "false"
```

3. Deploy the example.

```
$ kubectl apply -f pytorch-example.yaml
```

[redacted]Go to the Katib page.

[redacted]Click the Menu button, and then select HP → Monitor.

[redacted]Click the **experiment** name, and then observe the trials running.

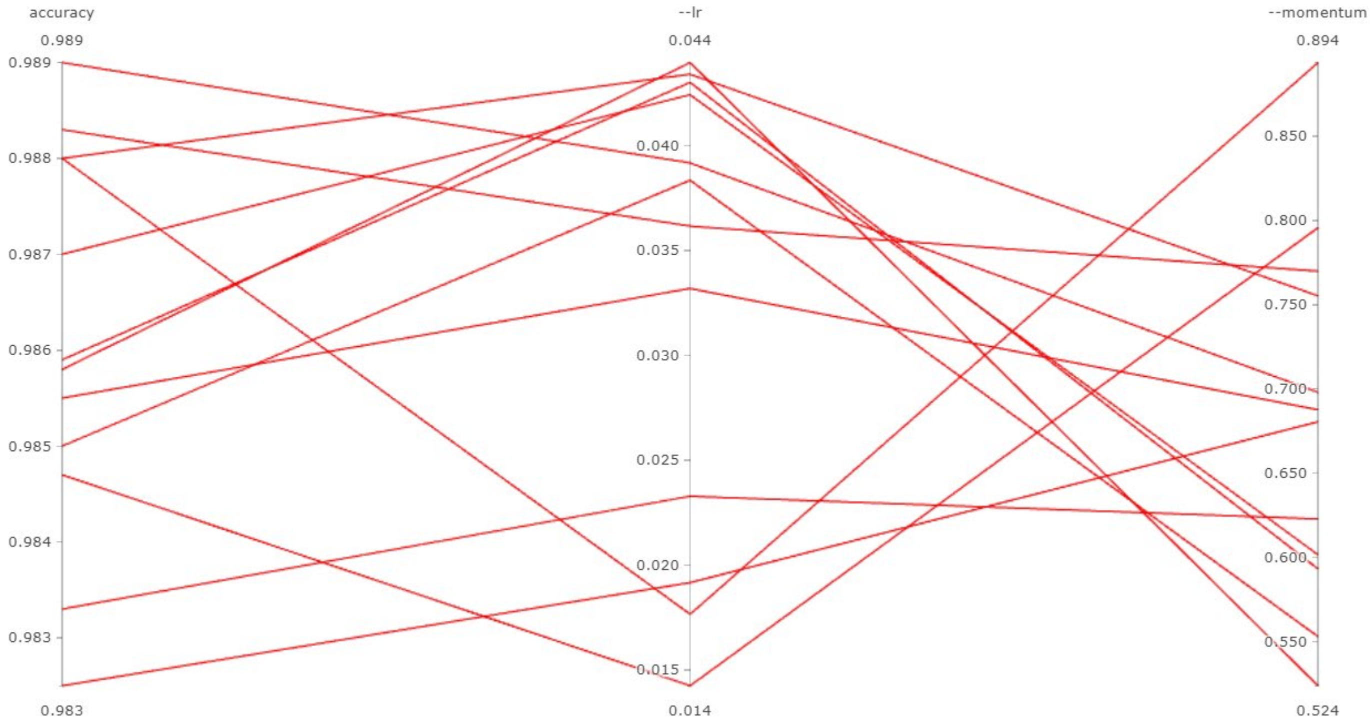


FIGURE A22. Pytorch Job Experiment

[redacted]Check the experiment status.

```
$ kubectl get experiment
```

[redacted]Use the following command to check the trials of the experiment.

```
$ kubectl get trial
```

Sample Katib commands

To check experiment results via thekub ectl CLI.

- List experiments

```
$ kubectl get experiment
```

- Check experiment result

```
$ kubectl get experiment random-example -o yaml
```

- List trials

```
$kubectl get trials
```

- Check trial detail

```
$ kubectl get trials random-experiment-24lgqghm -o yaml
```

To check the status using the interface:

[redacted] Go to the Katib page.

[redacted] Click the Menu button, and then select **HP** → **Monitor**.

[redacted] Click the experiment name and observe the built experiment graph after all the trials have succeeded.

Argo workflows

Download the kubeflow_tutorials.zip file, which contains sample files for all of the included Kubeflow tutorials.

This article provides the following two examples:

- Simple workflow
- Parallel execution workflow

Simple workflow

To complete the simple workflow:

[redacted] Create and apply the Argo workflow in the profile namespace.

[redacted] Obtain `argo-hello-world.yaml` from the zip file mentioned above.

```
$ kubectl apply -f argo-hello-world.yaml
workflow.argoproj.io/hello-world created
```

[redacted] Verify that the workflow was created:

```
$ kubectl get wf
NAME           AGE
hello-world       41m
parallelism-nested-dag  12d
```

[redacted] Verify that the related `hello-world` pod was created and is running.

```
$ kubectl get pods hello-world
NAME     READY  STATUS    RESTARTS  AGE
hello-world  0/2   Completed  0     49m
```

[redacted] Open the Argo interface by navigating to:

```
http://<kubeflow_url>/argo/
http://ezam-01.perflab.hp.com:10053/?ns=imageadmin/argo/
```

[redacted] To remove the workflow:

```
$ kubectl delete wf hello-world
workflow.argoproj.io "hello-world" deleted
```

Parallel execution workflow

To complete the parallel execution workflow:

[redacted] Create and apply the nested Argo workflow in the profile namespace.

[redacted] Obtain `argo-parallel-nested.yaml` from the zip file mentioned above.

```
$ kubectl apply -f argo-parallel-nested.yaml
workflow.argoproj.io/parallelism-nested-dag configured
```

[redacted] Verify that pods were created, as per the template (Observe the `.yaml`fil the template).

[redacted]Open the Argo interface by navigating to:

```
http://<kubeflow_url>/argo/
http://ezam-01.perflab.hp.com:10053/?ns=imageadmin/argo/
```

ML metadata

Before beginning this tutorial, download the Kubeflow tutorials zip file which contains sample files for all of the included Kubeflow tutorials.

[redacted]Create a Jupyter notebook server with any of the default images.

[redacted]Connect to the created notebook server and upload the following notebook: demo-ml.ipynb.

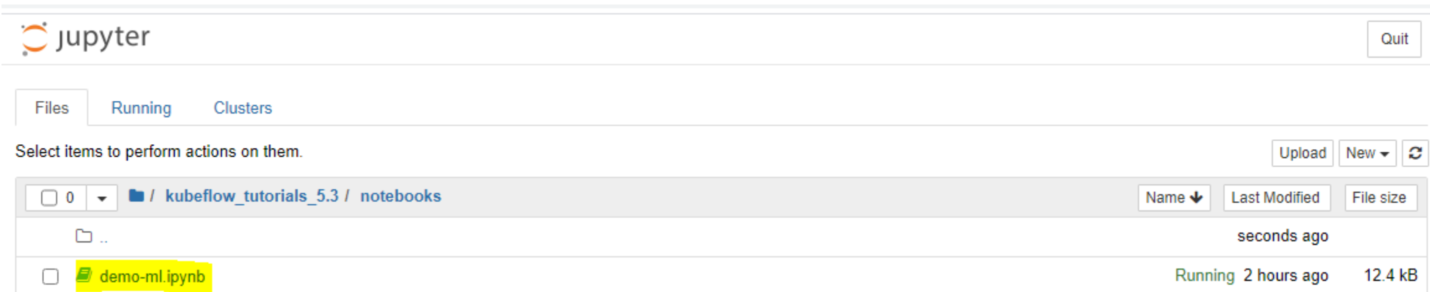


FIGURE 63. Demo-ml

[redacted]Run the notebook step by step, and observe the result on the **Pipelines** → **Artifacts** page in the Kubeflow UI.

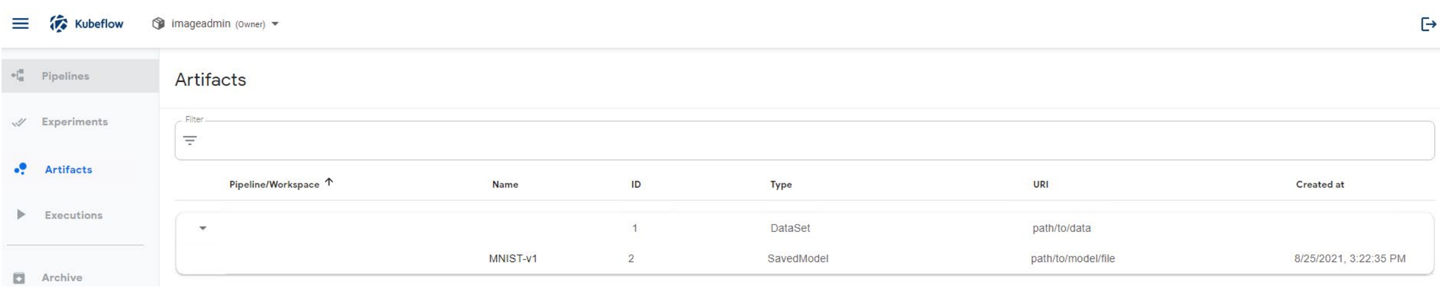


FIGURE A23. Artifacts

[redacted]Click the name of each item to view detailed information. Click the Execution tab on the left and see the details.

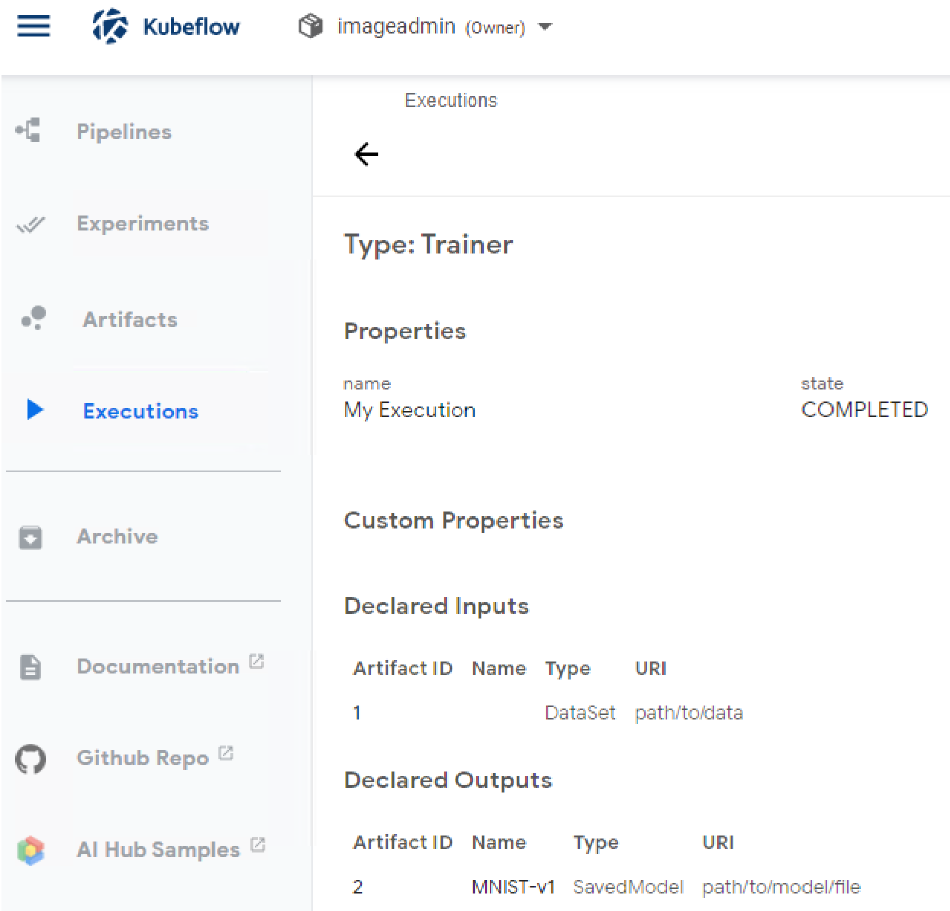


FIGURE A24. Executions

APPENDIX B: HPE ML OPS KDAPP

Centos/Ubuntu

In the Applications screen, launch Centos/Ubuntu app with the required resources screen, launch Centos/Ubuntu app with the required resources.

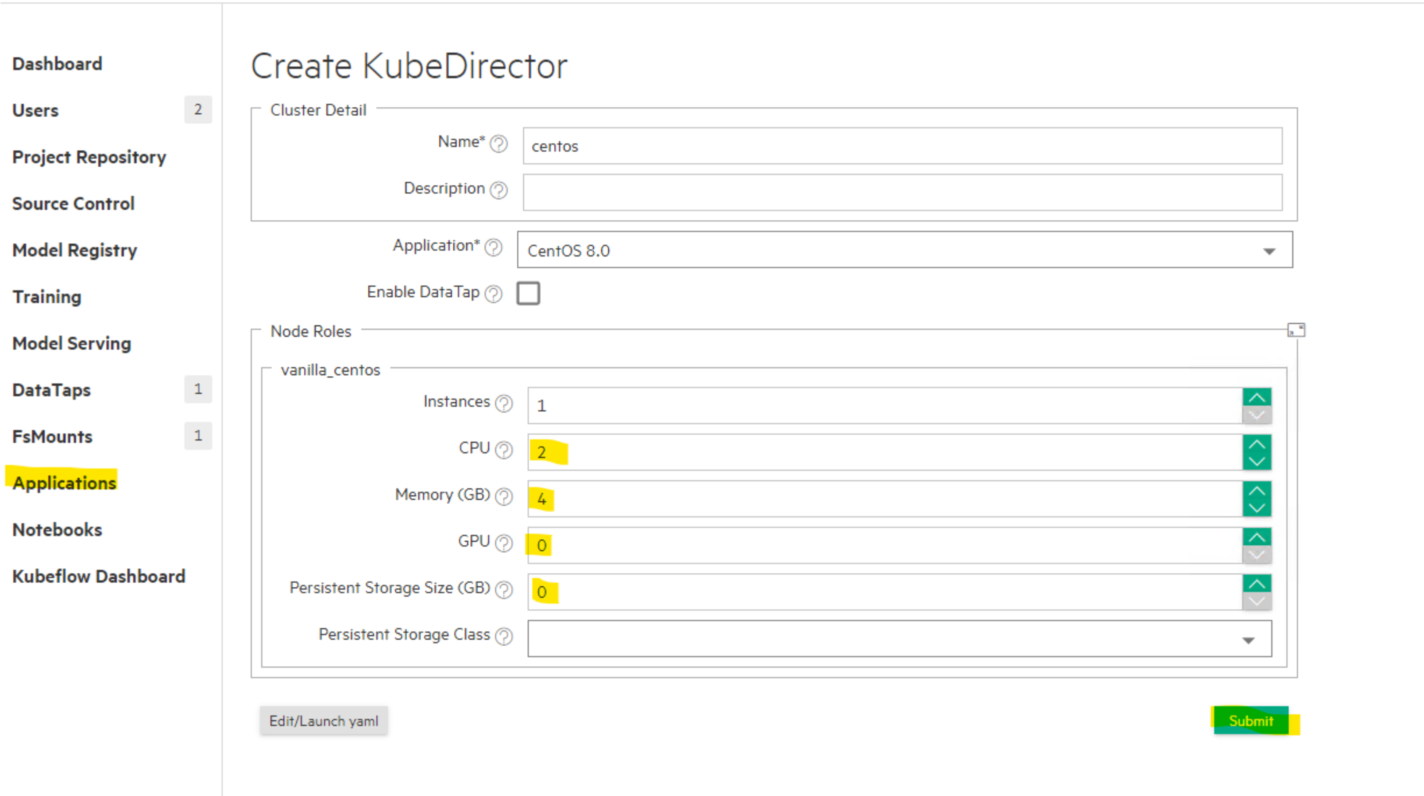


FIGURE B1. Launching KDAPP CentOS

[redacted] ssh-keygen in webterm and copy `id_rsa.pub`.

[redacted] Exec into the centos pod and run `ssh-keygen`.

```
k8suser@kdss-4dzrd-0:~$
k8suser@kdss-4dzrd-0:~$ kubectl exec -it centos-vanilla-centos-znsk6-0 -- bash
[root@centos-vanilla-centos-znsk6-0 /]# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa): c
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in c.
Your public key has been saved in c.pub.
The key fingerprint is:
SHA256:n/3GTa5DbTCvFSpfxDUKxCKz9J9VbbPA+1iq27EWDTY root@centos-vanilla-centos-znsk6-0
The key's randomart image is:
+---[RSA 3072]----+
|          o..   .|
|       + . o o ++|
|      . = . . *.=|
|       . .   Eo* |
|        S . + @=.|
|         . * =.+*|
|          o =+oB |
|           ..+B o|
|           .o+oo |
+----[SHA256]-----+
[root@centos-vanilla-centos-znsk6-0 /]#
```

[redacted] Vi /root/.ssh/authorized_keys, paste the `id_rsa.pub` that was copied, and exit.

```
[root@centos-vanilla-centos-zptfk-0 /]# vi root/.ssh/authorized_keys
[root@centos-vanilla-centos-zptfk-0 /]# cat root/.ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDuu/ZrMrbvlv8LYbfN10tX6utdhMzJIFRAw65cavGIYLyKIU5pKzp4pAWGTnkj1v0zJ6Txb/5uATCD5uBRZjs5jQcIrm2tKK3ZqO/hK+laorQ1AN0DY5F7XLWnpU6CK6hiCmGVnLy1Sai/Mv/0T5d5uWeER9oq1KnMTu6uCnjvYNd4pznOmMsXGStH9w
SB6w/zpBXpRjn2iBlnYGqfjwhyKg9iYFM5+8mzsTcMr8z0jOn0U2LYrtJKOy9EA5r3QBS1rEVmRQjkr/Ljkl5ST4f7wFEbK1PDc/Etc+TLe/RBVnIlIkpH/s6S7tH3gT6nSGOgFIIPI9mT4xfYROJEK5 k8suser@kdss-bwkz8-0
[root@centos-vanilla-centos-zptfk-0 /]#
```

[redacted] ssh to the Access Points.

Kubernetes Applications

KubeDirector | Kubectl | **Service Endpoints** | Virtual Endpoints

Kubernetes Service Name	Role	Details	KubeDirector Cluster	Services	Ports	Access Points	Service Type
centos-vanilla-centos-zptfk-0	vanilla_centos	KubeDirectorApp: ID: centos8x Name: CentOS 8.0	centos	SSH	22	ezam-01.perflab.hp.com:10026	NodePort
kf-dashboard-import-l6n8v				80	80	ezam-01.perflab.hp.com:10019	NodePort
livy-http				http	8998	ezam-01.perflab.hp.com:10023	NodePort
spark-ui-proxy				http	80	ezam-01.perflab.hp.com:10022	NodePort
sparkhs-svc				http	18480	ezam-01.perflab.hp.com:10020	NodePort
sparkts-svc				http	4440	ezam-01.perflab.hp.com:10021	NodePort
				spark-thrift	2304	ezam-01.perflab.hp.com:10024	

FIGURE B2. CentOS Access Points

```
k8suser@kdss-bwkz8-0:~$
k8suser@kdss-bwkz8-0:~$ ssh root@ezam-01.perflab.hp.com -p 10026
The authenticity of host '[ezam-01.perflab.hp.com]:10026 ([172.24.2.1]:10026)' can't be established.
ECDSA key fingerprint is SHA256:DKJi3QsAESBEAHwMOk41V4TJG1WakTK4c9kkFmsH8Ik.
ECDSA key fingerprint is MD5:c8:b0:18:40:f9:e4:6f:31:d0:2d:50:4d:64:48:90:91.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[ezam-01.perflab.hp.com]:10026,[172.24.2.1]:10026' (ECDSA) to the list of known hosts.
[root@centos-vanilla-centos-zptfk-0 ~]# ls
anaconda-ks.cfg  anaconda-post.log  original-ks.cfg
[root@centos-vanilla-centos-zptfk-0 ~]# pwd
/root
[root@centos-vanilla-centos-zptfk-0 ~]#
```

MLflow

MLflow is an open-source platform to manage the machine learning lifecycle, including experimentation, reproducibility, deployment, and a central model registry. For MLflow integration in the HPE Ezmeral Runtime details, see MLflow for Model Management. See MLflow Configuration and Deployment for the process to execute one run of MLflow from training to deployment. The user can clone the MLflow notebook used in this tutorial from https://github.com/pcao11/mlflow.git.

[redacted] Generate and apply the MLflow Secret (See the HPE Ezmeral Container Platform 5.3 Documentation for the details).

```
k8suser@kdss-r9bk8-0:~$ vi mlflow-sc.yaml
k8suser@kdss-r9bk8-0:~$ cat mlflow-sc.yaml
apiVersion: v1
kind: Secret
stringData:
  MLFLOW_ARTIFACT_ROOT: s3://hanshabucket #s3://mlflow
  AWS_ACCESS_KEY_ID: AKIAW2IT4GWHVDDWS25E #myusername
  AWS_SECRET_ACCESS_KEY: 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 #mypassword
  #MLFLOW_BACKEND_STORE: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx=
                        #mysql://mlflowusr:Password^12@192.0.2.0:3306/mlflowdb
  MLFLOW_S3_ENDPOINT_URL: https://s3.us-east-2.amazonaws.com #https://s3.us-east-2.amazonaws.com
  #MLFLOW_S3_ENDPOINT_URL: https://hanshabucket.s3.us-east-2.amazonaws.com #https://s3.us-east-2.amazonaws.com
#xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx==
                           #http://myserver.example.com:10007
  #AWS_DEFAULT_REGION: us-east-2
metadata:
  name: mlflow-sc
  labels:
    kubedirector.hpe.com/secretType: mlflow
type: Opaque
      #podLabels:
        #hpecp.hpe.com/dtap: "hadoop2"

k8suser@kdss-r9bk8-0:~$ kubectl apply -f mlflow-sc.yaml
secret/mlflow-sc created
k8suser@kdss-r9bk8-0:~$
```

[redacted] Create an MLflow App Instance by attaching the mlflow secret (See the HPE Ezmeral Container Platform 5.3 Documentation for the details).

Create KubeDirector

Edit your Yaml and hit submit to launch app

```
---
apiVersion: "kubedirector.hpe.com/v1beta1"
kind: "KubeDirectorCluster"
metadata:
  name: "mlflow"
  namespace: "amdtenant"
  labels:
    description: ""
spec:
  app: "mlflow"
  namingScheme: "CrNameRole"
  appCatalog: "local"
  connections:
    secrets:
    - mlflow-sc
  roles:
    -
      id: "controller"
      members: 1
      resources:
        requests:
          cpu: "2"
          memory: "4Gi"
          nvidia.com/gpu: "0"
        limits:
          cpu: "2"
          memory: "4Gi"
          nvidia.com/gpu: "0"
      #Note: "if the application is based on hadoop3 e.g. using StreamCapabilities interface, then change the below dtap label to 'hadoop3', otherwise for most applications use the default 'hadoop2'"
      #podLabels:
        #hpecp.hpe.com/dtap: "hadoop2"
```

Submit Cancel

FIGURE B3. Launching MLflow App Instance [redacted]Create a Training Application Instance. See the HPE Ezmeral Container Platform 5.3 Documentation for details (optional).

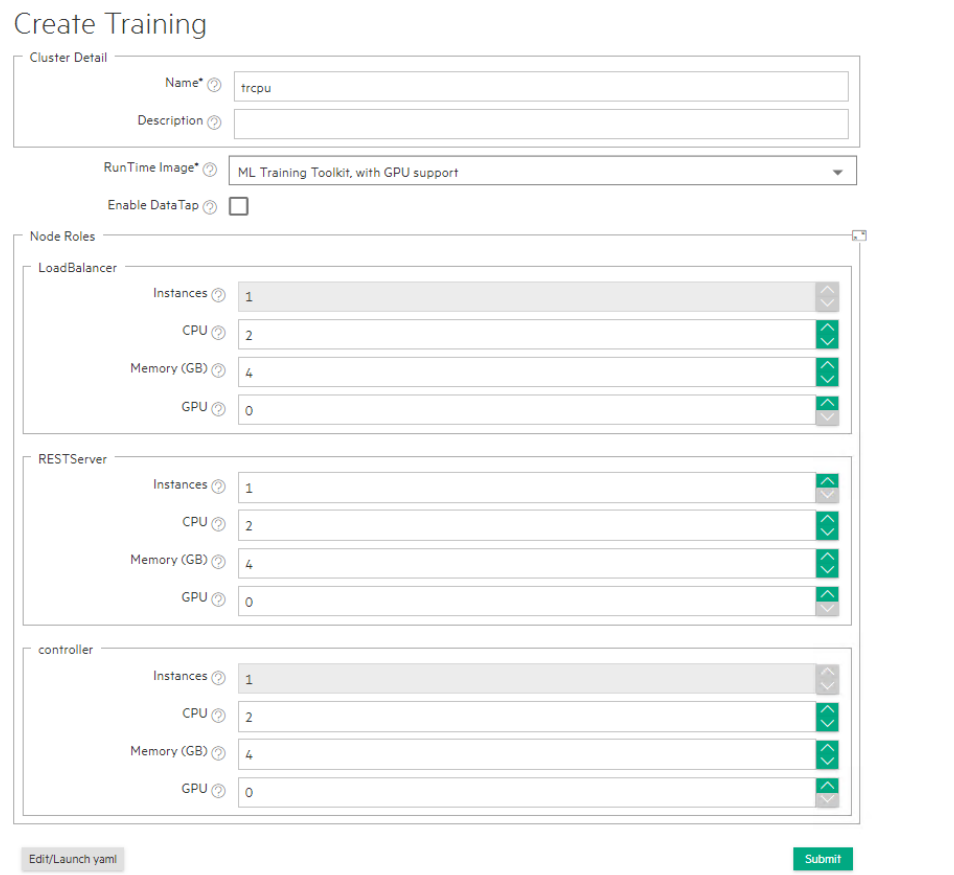


FIGURE B4. Create Training Cluster

[redacted]Launch and configure the Notebook (See the HPE Ezmeral Container Platform 5.3 Documentation for more details).

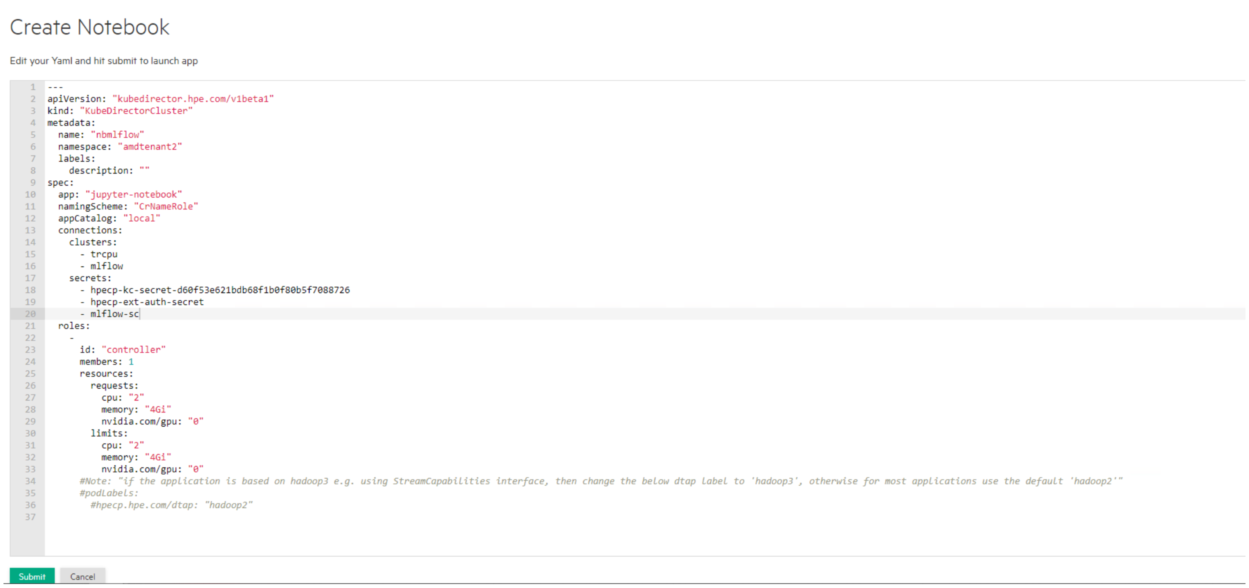


FIGURE B5. Create MLflow Notebook

[redacted]Enable kubectl to run MLflow backend (mandatory). Set the experiment name, Train, and track models.

Set your password (mandatory)

```
[1]: PASSWORD = "hp123456" # use your password
```

Enable kubectl to run MLflow backend (mandatory)

```
[2]: %kubeRefresh --pwd $PASSWORD
```

```
kubeconfig set for user imageadmin
```

```
[3]: # This magic sets the environmental variables required for mlflow in backend.
%loadMlflow
```

```
Backend configured
```

Set your experiment name

```
[4]: # Magic function '%Setexp' replaces the two lines below.
#mlflow.set_experiment('demoexp')
#mlflow.set_tag('mlflow.user','chris')
%Setexp --name demoexp
```

[redacted]Observe experiments, runs, metrics, parameters, dependency, and trained models in the MLflow UI.

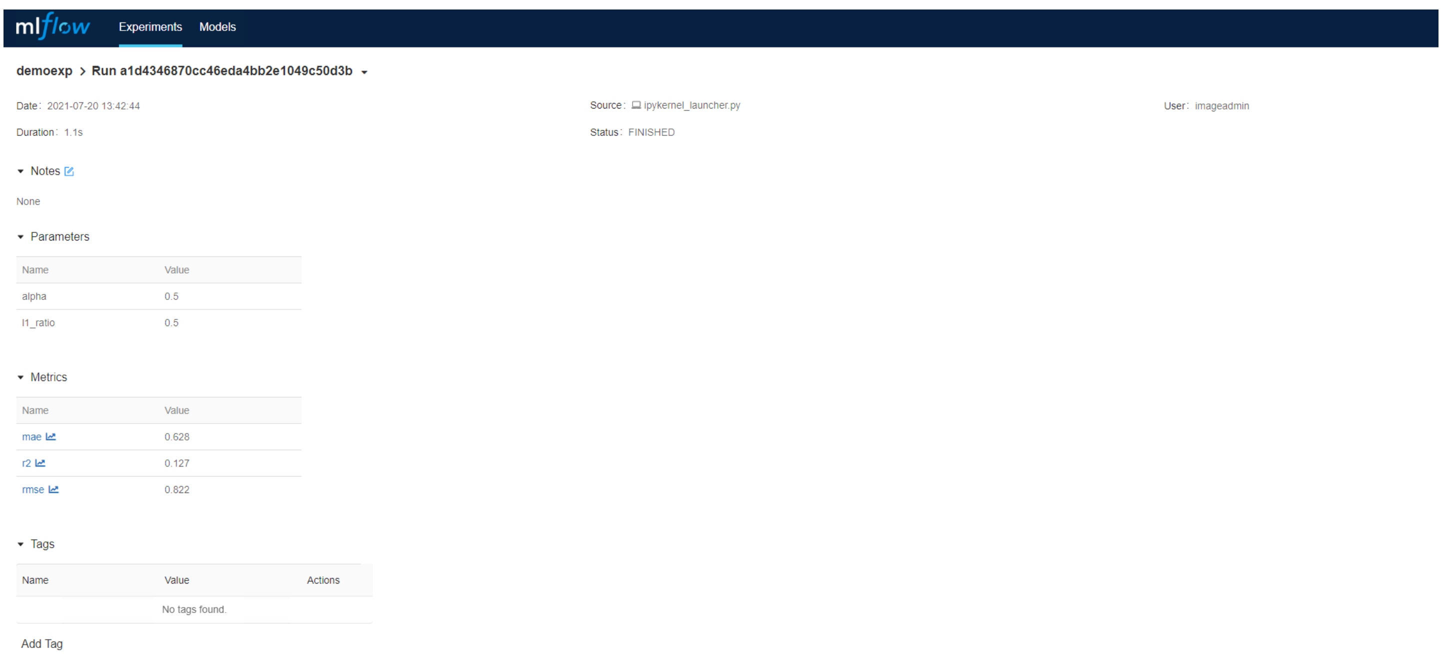


FIGURE B6. MLflow Experiment, Run

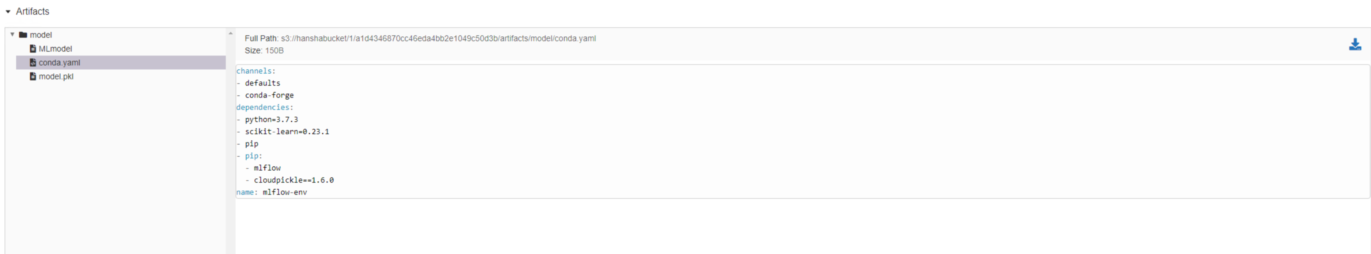


FIGURE B7. Dependency and Trained Model

[redacted]Register Model for MLflow (See the HPE Ezmeral Container Platform 5.3 Documentation for details).

FIGURE B8. Register Model

[redacted]Create an MLflow Model Serving (See the HPE Ezmeral Container Platform 5.3 Documentation for details).

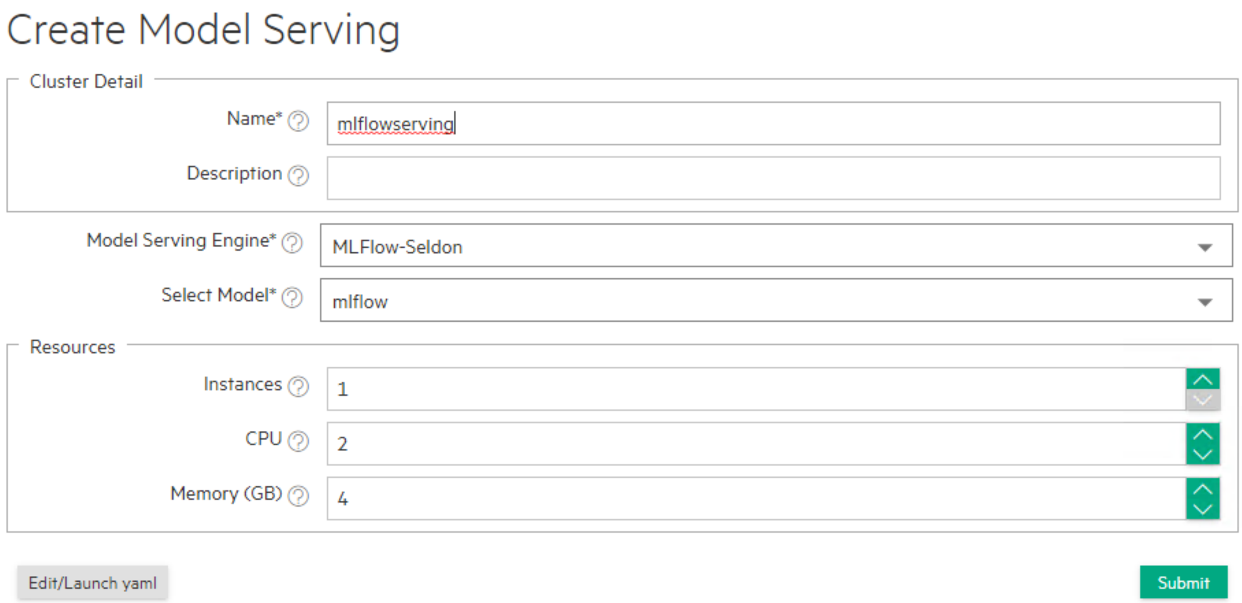
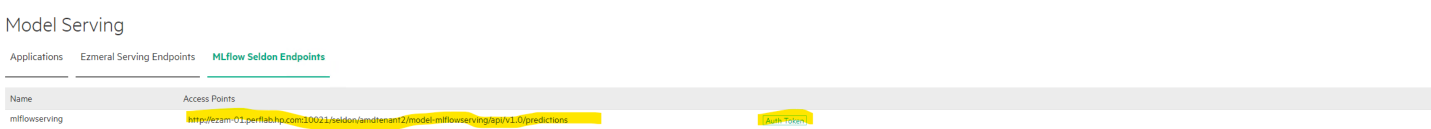


FIGURE B8. MLflow Model Serving

[redacted]Copy Access Points and Auth Token.



[redacted]Making Prediction Calls (See the HPE Ezmeral Container Platform 5.3 Documentation for details).

```
[root@ezam-09 ~]#
[root@ezam-09 ~]#
[root@ezam-09 ~]# curl -k --cookie "authservice_session=MTYyNjgwNjg4M3xOd3dBTkVWQlVFRklVRVpEV0U4MFZVZFdRVTlZUzBSS1JqUTNSMWRFUWsxTVRGQlNRVk5VVGtOUlNVcERWRUkzUXpOVFdFVlJWMEU9fJaVakxGdSnp3ZbquQcOYLUXhvzZG-STuceSp_wzMRFT" -X POST -H 'Content-Type: application/json' -d '{"data":{"names": [
"fixed acidity","volatile acidity","citric acid","residual sugar","chlorides","free sulfur dioxide","total sulfur dioxide","density","pH","sulphates","alcohol"], "ndarray":[[7,0.27,0.36,20.7,0.045,45,170,1.001,3,0.45,8.8]]}}' http://ezam-01.perflab.hp.com:10021/seldon/amdtenant2/model
-mlflowserving/api/v1.0/predictions
{"data":{"names":[],"ndarray":[5.655099099229192]},"meta":{}}
[root@ezam-09 ~]#
```

NVIDIA: TensorFlow (NGC)

[redacted]Create an NVIDIA TensorFlow (NGC) App Instance.

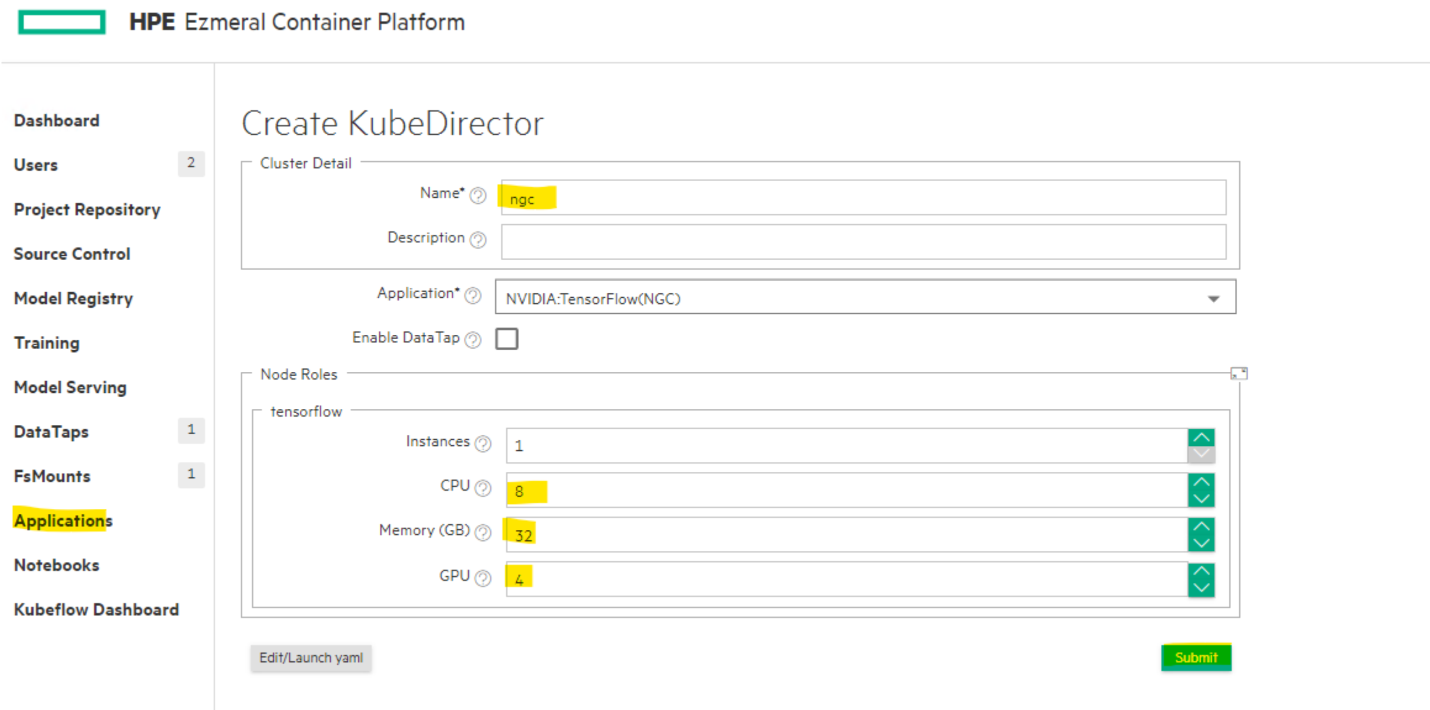


FIGURE B9. Launching NVIDIA TensorFlow (NGC) App Instance

[redacted]Login inside the pod to run TensorFlow jobs with GPU.

```
k8suser@kdss-bwkz8-0:~$
k8suser@kdss-bwkz8-0:~$ kubectl exec -it ngc-tensorflow-qvz94-0 -- bash
root@ngc-tensorflow-qvz94-0:/workspace# vi tftest.py
root@ngc-tensorflow-qvz94-0:/workspace# python3 tftest.py
2021-08-16 18:28:24.644481: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library libcudart.so.11.0
WARNING:tensorflow:Deprecation warnings have been disabled. Set TF_ENABLE_DEPRECATION_WARNINGS=1 to re-enable them.
2021-08-16 18:28:28.447813: I tensorflow/core/platform/profile_utils/cpu_utils.cc:94] CPU Frequency: 2500000000 Hz
```

TensorFlow + Jupyter

[redacted]Create a TensorFlow + Jupyter App Instance.

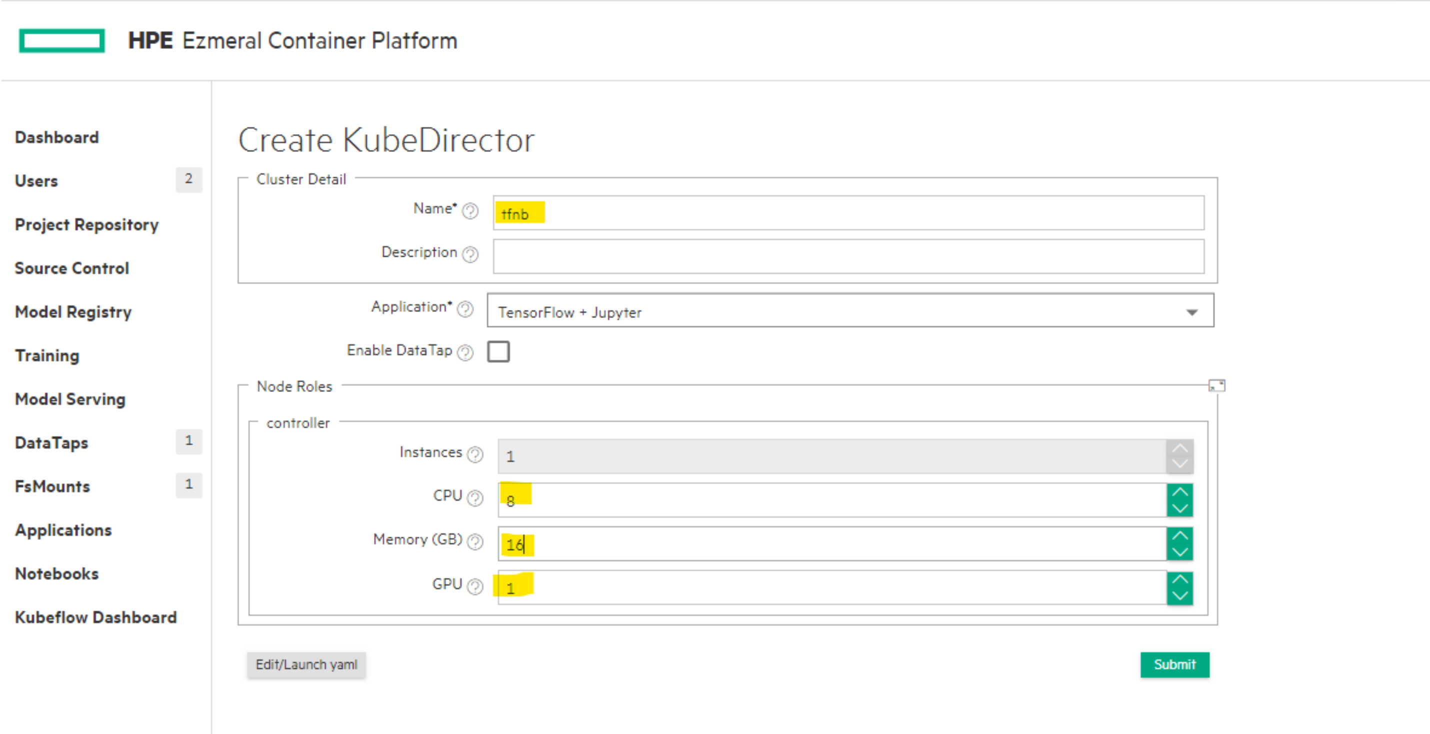


FIGURE B10. Launching TensorFlow + Jupyter App Instance

[redacted]Click the **access points** of the created App Instance.

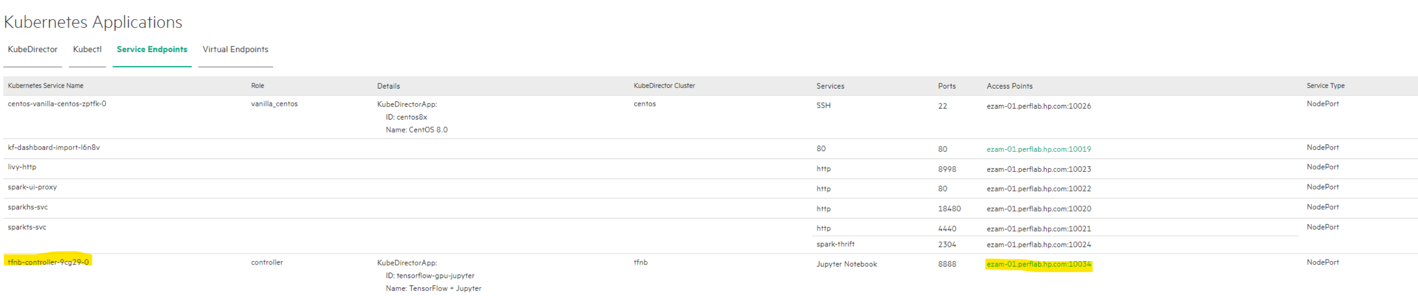


FIGURE B11. Access Points of TensorFlow + Jupyter [redacted] Exec into the TensorFlow +Jupyter pod and run `jupyter notebook list` and copy the token.

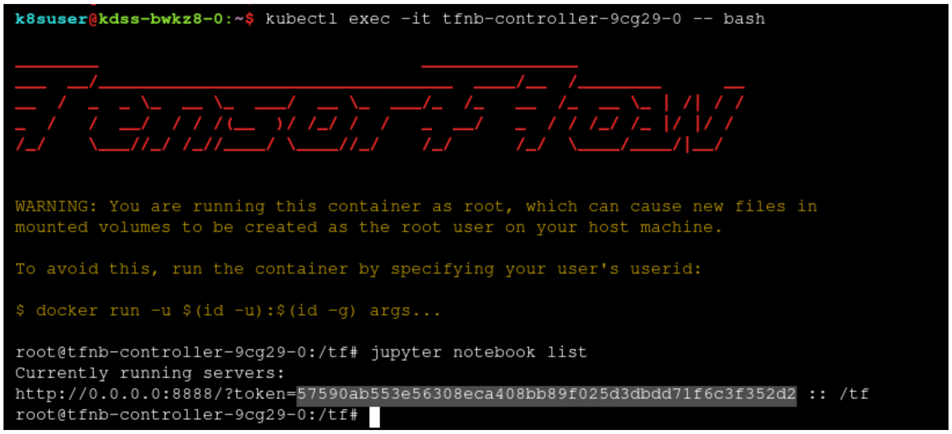


FIGURE B12. Copy the token

[redacted] Paste the copied token and login to TensorFlow + Jupyter notebook.

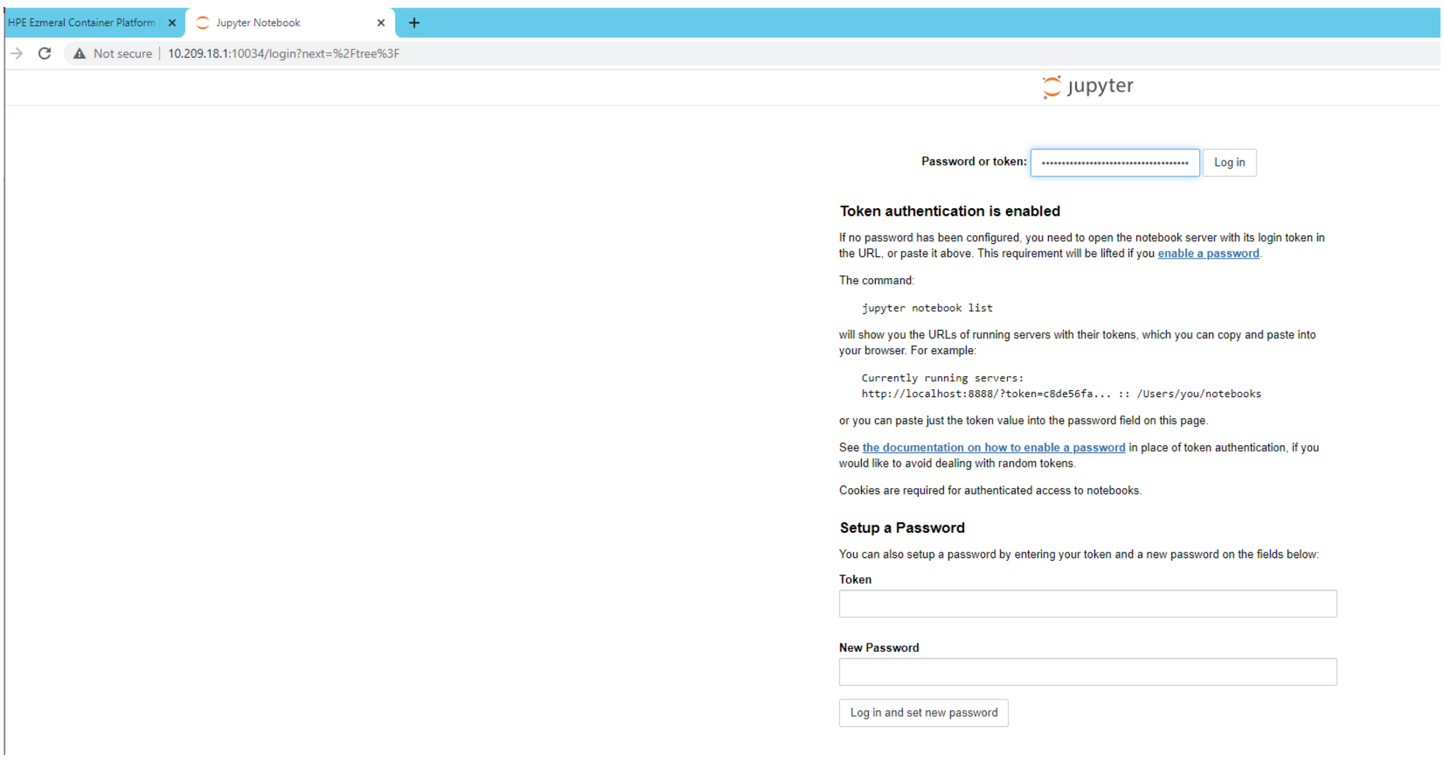


FIGURE B13. Login to TensorFlow + Jupyter Notebook [redacted]Now launch the notebook and run TensorFlow with GPU

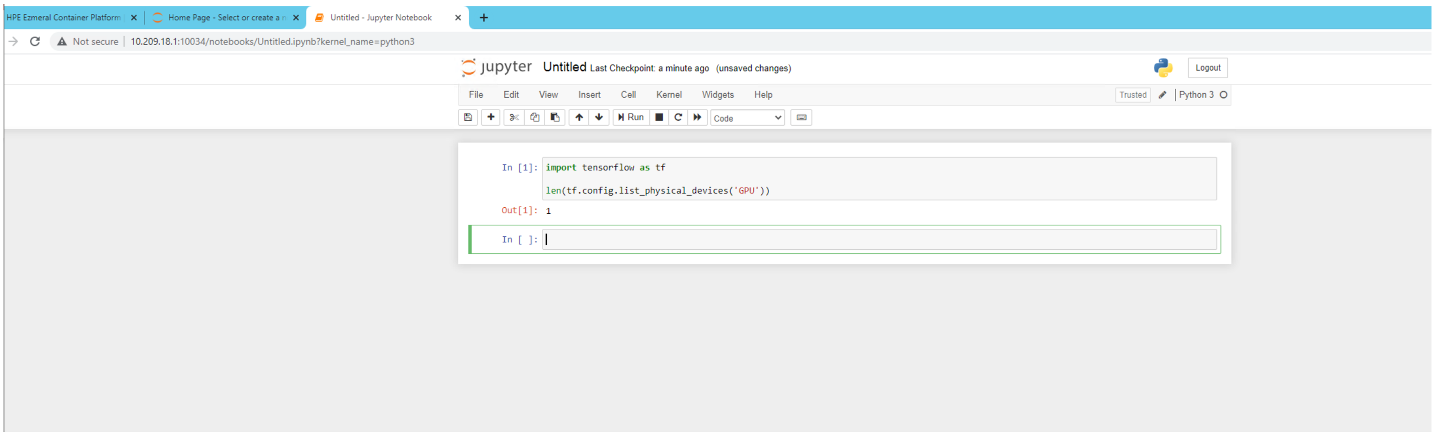


FIGURE B14. Python3 Notebook

APPENDIX C: INSTALL AND CONFIGURE HPE EZMERAL RUNTIME

Follow the steps as outlined to install and configure the HPE Ezmeral Runtime. This section assumes that all the prerequisites mentioned in the earlier sections were followed. See the Standard Installation procedure.

RESOURCES AND ADDITIONAL LINKS

HPE Reference Architectures, hpe.com/info/ra

HPE Servers, hpe.com/servers

HPE Storage, hpe.com/storage

HPE Networking, hpe.com/networking

HPE Technology Consulting Services, hpe.com/us/en/services/consulting.html

HPE Ezmeral Machine Learning Ops, https://buy.hpe.com/us/en/enterprise-solutions/artificial-intelligence-analytics/artificial-intelligence-analytics/artificial-intelligence-analytics/hpe-ezmeral-machine-learning-ops/p/1011947349

Operationalization for the Machine Learning Lifecycle live demonstration, https://hpedemoportal.api.ext.hpe.com/DemoPortal/api/DocContent/GetDocByToken/26858daf-c14b-49f0-ae8f-60d3e7423e66

HPE Ezmeral ML Ops, https://assets.ext.hpe.com/is/content/hpedam/documents/a50000000-0999/a50000137/a50000137enw.pdf

Kubeflow Introduction, https://www.kubeflow.org/docs/pipelines/overview/pipelines-overview/

https://v0-5.kubeflow.org/docs/use-cases/

Broadcom AIOps, https://www.broadcom.com/sw-tech-blogs/aiops-blog/what-is-prometheus#:~:text=Prometheus%20is%20a%20time-series%20streaming%20data%20tool.%20It,make%20that%20data%20available%20for%20processing%20and%20analysis

HPE Ezmeral Container Platform 5.3 Documentation, https://docs.containerplatform.hpe.com/53/index.html

To help us improve our documents, please provide feedback at hpe.com/contact/feedback.

a50006296enw, version 1.0, May 2023