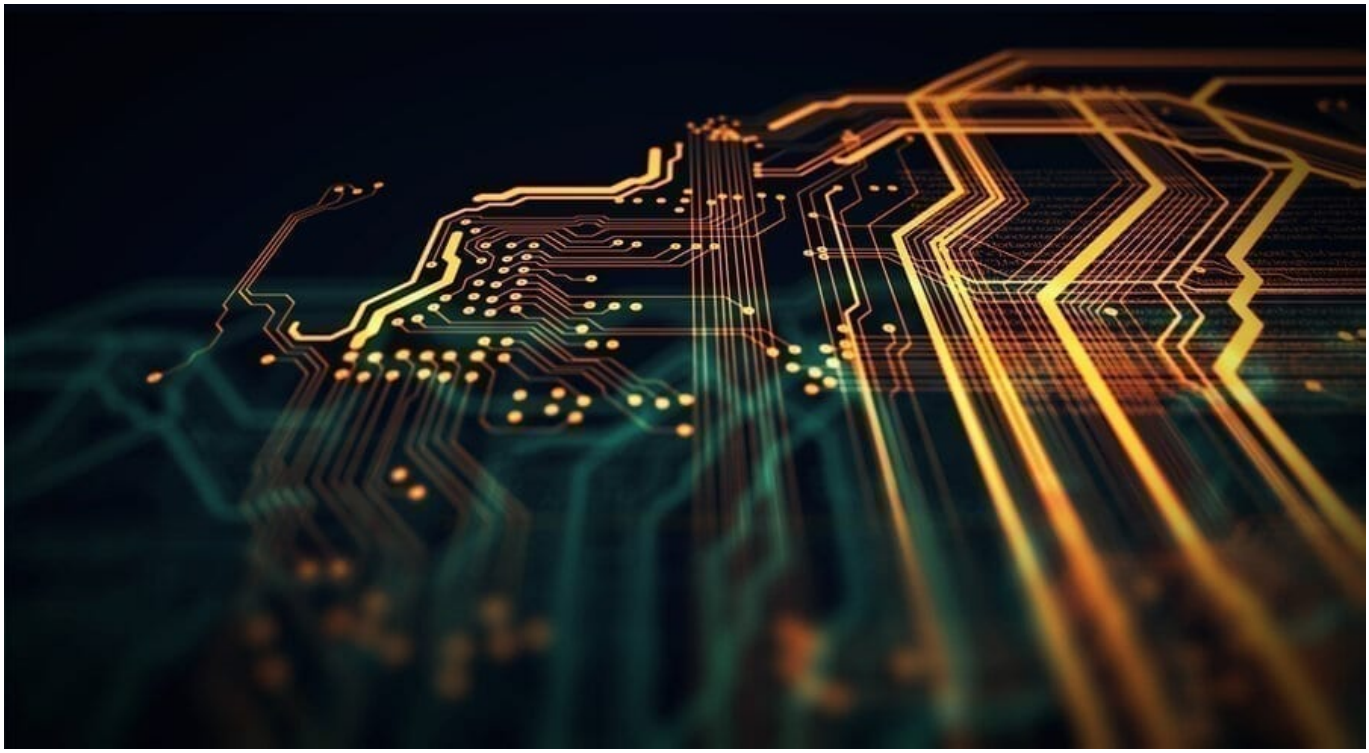
Jupyter-as-a-Service on Flash: Sewing Up Data Pipelines on Kubernetes



Modern businesses are building smarter applications powered by massive amounts of data generated at scale. Smart applications are designed to be productive by consuming information from different data sources. Designing and developing the classic Continuous Integration (CI)/Continuous Delivery (CD) pipeline on Kubernetes is an ongoing DevOps process on-premises and in the cloud to develop these smart applications.

Most modern and innovative businesses extend the CI/CD processes into analytics and Machine Learning (ML) pipelines. Application developers work in conjunction with Data Engineers who publish different data sources and services, and Data Scientists who federate the transformed data to prototype, identify patterns, and develop algorithms to train models. The assembly line coalesces the different data sources for better decision making to fuel company growth.

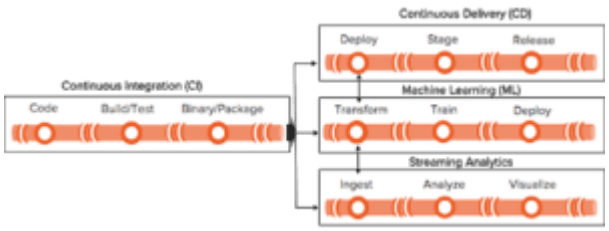


Figure 1: Assembly pipeline with connected data pipelines

This blog will be focusing on the ML pipeline, platform, and data challenges that pose with newer Data Scientists who have limited control and access to expensive Graphics Processing units (GPUs) and other platform resources during their early exploration phases. Data Scientists commonly use an open-source tool called Jupyter notebook that helps them create, share text, code, write documents, use functions, plot graphs, and more. Jupyter notebook is primarily used during the prototyping phase where every individual data scientist can write and execute code in their private work area.

In this blog, I will analyze why and how Data Scientists can functionally and operationally perform better with Kubernetes on FlashBlade using PSO in a robust and reliable manner.

Machine Learning (ML) Pipeline

ML pipeline is not a separate workload that runs as a separate batch job but is the heart of the assembly line with over-laying functionalities with other data pipelines. Machine learning in production is a component of intelligent applications that learn from data and improves functionality as people continue to use them. The data pipeline is required to gather data from different sources to transform, federate it in order to continuously train, and service models to support any key application functionality.

The workflow shown below highlights the sequence of stages that may be part of single or multiple data pipelines that feed into each other at various stages. After the data is transformed during the preparation phase, Data Scientists write and use techniques to train and re-train models and test them before deploying them into production.

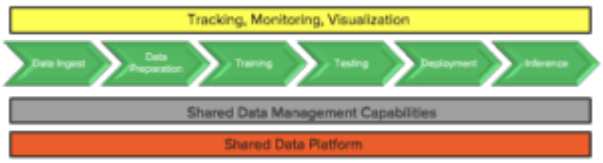


Figure 2: ML data pipeline and layers

Challenges

Traditionally, a Data Scientist would perform all data pipeline activities on a laptop, posing compelling infrastructure, process, and security challenges.

1. GPUs are expensive and are not readily available to new Data Scientists who struggle to set up a

quick-start environment where they use CPUs to perform exploration before going mainstream. The other choice may be to configure a small GPU environment in the public cloud, but there may be cost repercussions while scaling the environment.

2. Personal computers and laptops run out of cores and storage very quickly as the prototyping and exploration process intensifies. File access and IO operations are not distributed and parallel.
3. Jupyter notebooks are designed to share and collaborate among other members of the team. Members of the team may get different results due to limitations from build dependencies for bespoke environments or the notebook would crash when shared among themselves.
4. Reproducing similar test environments iteratively requires the desired infrastructure knowledge and resources that may not be available on demand.
5. Adding more compute and memory for applications to scale requires a greater number of servers where additional storage may not be required in the environment. The inability to scale compute and storage independently increases cost and high manageability of server sprawl.

Jupyter-as-a-Service Validated Architecture

Kubernetes distribution from Pivotal (Pivotal Container Service – PKS) is the popular choice of container orchestrator used for application development pipelines in modern times. Applications and tools that are part of any data pipeline, as illustrated in Figure 2, have evolved to be more stateful where data needs to be persistent and reused in a more pertinent and predictable manner. Figure 3 shows a validated Jupyter-as-a-Service architecture on FlashBlade for Data Scientists who that can consume infrastructure on demand from PKS cluster. Data Scientists can use Central Processing Units (CPUs) at scale for their initial testing.

There are more than half a dozen Kubernetes storage class provisioners available in the market that provisions persistent storage for stateful applications. However, Pure Storage Orchestrator™ (PSO) from Pure Storage® stands out among others on the merits of smart provisioning, elastic scaling and transparent recovery from failures.

```
root@sc2k8cl1:~# kubectl get sc
NAME          PROVISIONER        AGE
pure          pure-provisioner   28d
pure-block    pure-provisioner   28d
pure-file     pure-provisioner   28d
```

PSO as a first-class Kubernetes citizen that provisions persistent storage on-demand for stateful applications on FlashArray™ (blocks) and FlashBlade™ (files) from Pure Storage.

Different application workloads that uses files, objects, logs, streams etc. feed into pipelines that provide data-driven decisions use using a standard data platform like FlashBlade. FlashBlade possesses the true characteristics of a Data Hub that allows different workloads and datasets to co-exist on the same data platform and to share and collaborate different data pipelines.

Stateful applications and tools running in the Kubernetes cluster greatly benefits from predictable performance that is linearly scalable, distributed and has parallel file-system access from FlashBlade. The combination of PSO and FlashBlade complements the application requirements from the Kubernetes cluster through scalable performance and resiliency.

Building ML and other connected pipelines on Kubernetes cluster allows the Data Scientists to provision, scale-out and deploy applications rapidly. PSO and FlashBlade not only provisions persistent storage for stateful applications but also removes the complexity of managing disaggregated data from different production pipelines. Applications and data can scale independently of each other where applications can allocate compute, memory and persistent storage based on the needs of the workload.

Multiple versions of notebook image(s) with the required dependencies can be tested and built to use and share among the members of the team. Data Scientists can now perform declarative deployments of the image for training, testing, and deploying into production. While Kubernetes provides a distributed form of application deployment, FlashBlade provides a distributed and parallel access to the data for faster workload processing. A standard notebook image provides repeatability while training a model and can run multiple instances in parallel.

Technology

The following snippet of the Jupyterhub v0.9.6 (helm chart v0.8.2) values.yaml file uses a “k8s-singleuser-sample” notebook image for every user trying to launch Jupyter notebook. The DockerFile for this image can be customized to any environment. The StorageClass is set to “pure-file” where 10Gi of space is provisioned on-demand by PSO on FlashBlade every time a user logs in for the first time using their credentials.

```
singleuser:
  extraTolerations: []
  nodeSelector: {}
  extraNodeAffinity:
    required: []
    preferred: []
  extraPodAffinity:
    required: []
    preferred: []
  extraPodAntiAffinity:
    required: []
    preferred: []
  networkTools:
    image:
      name: jupyterhub/k8s-network-tools
      tag: '0.8.2'
  cloudMetadata:
    enabled: false
    ip: 169.254.169.254
  networkPolicy:
    enabled: false
    egress:
    # Required egress is handled by other rules so it's safe to modify this
      - to:
          - ipBlock:
              cidr: 0.0.0.0/0
              except:
                - 169.254.169.254/32
  events: true
  extraAnnotations: {}
  extraLabels:
    hub.jupyter.org/network-access-hub: 'true'
```

```
extraEnv: {}
lifecycleHooks:
initContainers: []
extraContainers: []
uid: 1000
fsGid: 100
serviceAccountName:
storage:
  type: dynamic
  extraLabels: {}
  extraVolumes: []
  extraVolumeMounts: []
  static:
    pvcName:
    subPath: '{username}'
  capacity: 10Gi
  homeMountPath: /home/jovyan
  dynamic:
    storageClass: pure-file
    pvcNameTemplate: claim-{username}{servername}
    volumeNameTemplate: volume-{username}{servername}
    storageAccessModes: [ReadWriteOnce]
image:
  name: jupyterhub/k8s-singleuser-sample
  tag: '0.8.2'
  pullPolicy: IfNotPresent
```

Kubernetes v1.11 and later support Metallb v0.8.2 as the load balancer on bare metal and VMs to route data packets for on-premise cluster. Metallb adds great value as a layer 4 (at the TCP layer) load balancer for Jupyterhub to declare as a Kubernetes service for end-users. Setting up a layer 2 routing with a pool of IP addresses prior to running Jupyterhub chart allows the public-proxy to expose the Jupyterhub service as an IP address for the end-user to launch their own Jupyter notebook.

Watch the following demo recording for more details.

When a file is loaded by Jupyter, it is primarily stored in a variable and the corresponding python process reserves the memory to store the content, as long as the notebook is running. Jupyter sometimes does not release the memory even after the access to the data stops leading to a memory leak and pressure situation. Shutting down the notebook or logging out of the notebook releases the memory and flushes the buffers to the persistent storage for the data to be reused in the future.

Once the user logs out from an idle timeout or by manually shutting the notebook, the pods are recycled to free up the compute leaving behind Persistent Volume Claims (PVC) and its corresponding Persistent Volumes (PV). When the user logs back into the notebook, a new pod spins up using the existing PVC, thus optimizing the compute resources. This allows more freedom for Data Scientists to reuse the data for iterative testing as they make progress through the respective pipelines.

Data Scientists can repeat the stages rapidly in the ML pipeline from raw data to clean data, use different vectors, train models, and evaluate and deploy in production. Distributed application can be deployed and released into production on Kubernetes cluster using Blue/Green deployments where network traffic can be restricted to a new model before opening it up to everyone.

Data Scientists can move away from the ask-wait-get process of engaging with opening tickets, waiting for an infrastructure engineer to respond, and then finally getting the data pipelines on-boarded. Kubernetes natively supports application components along with compute and data-intensive ML pipelines on FlashBlade that makes it simpler to manage and deploy intelligent applications. Jupyterhub integration on FlashBlade using PSO provides more autonomy to the Data Scientists to consume infrastructure on demand.

Jupyter-as-a-Service on Kubernetes and CPUs allows Data Scientists to qualify the preliminary exploration and validation of the data-intensive ML pipelines before they can promote themselves to more specialized hardware like DGXs and T4s. The subsequent series of blogs will highlight some key benefits of running ML pipelines on Kubernetes and DGX on FlashBlade.