KubeFATE on VMware Cloud Foundation with VMware Tanzu

Operationalize Federated Learning Platform Efficiently and Securely

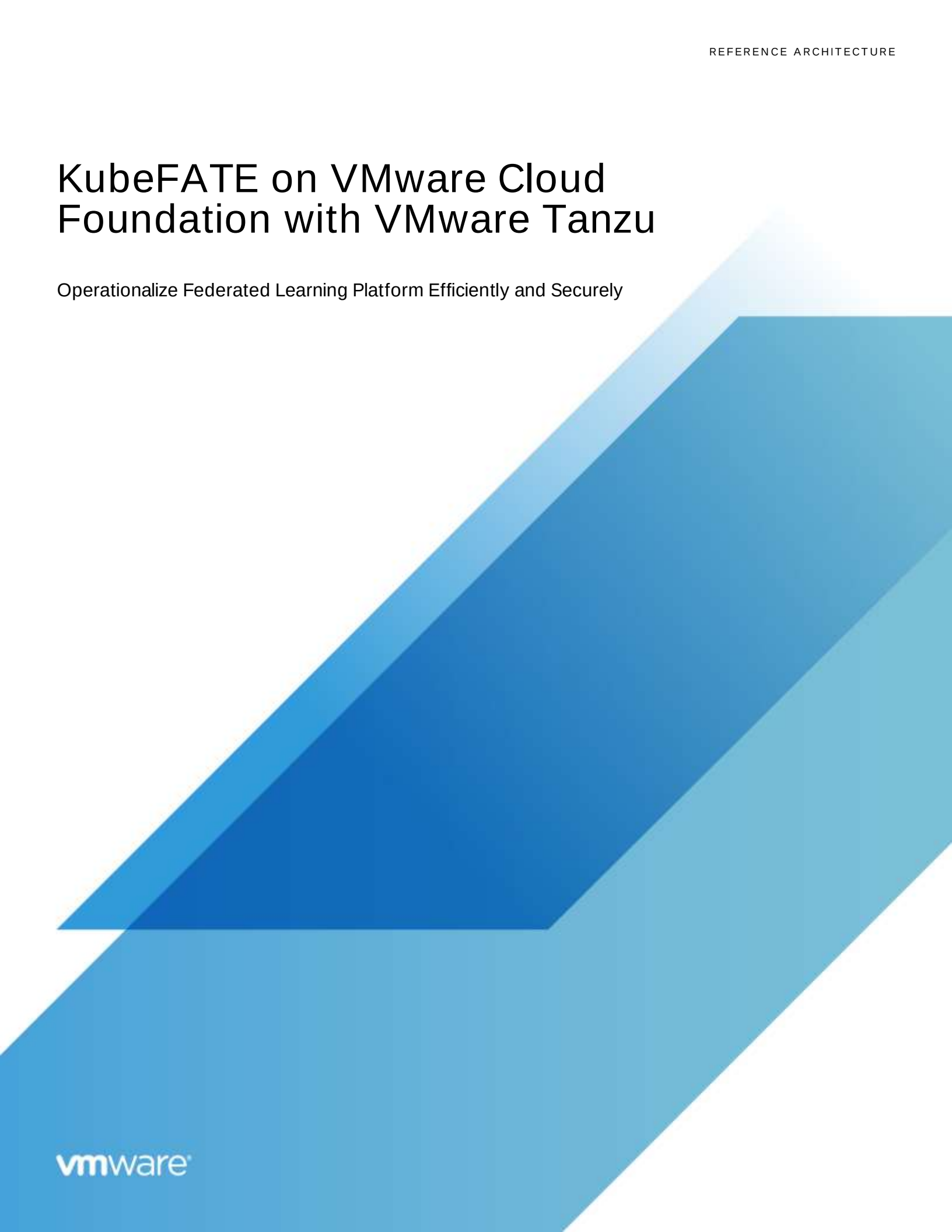


Table of contents

Business Case 3
- Business Values 3
- Key Results 4
- Audience 4

Technology Overview 4
- VMware Cloud Foundation 5
  - VMware Cloud Foundation with VMware Tanzu 5
  - VMware vSphere 5
  - VMware NSX Data Center 5
- KubeFATE 6

Solution Configuration 6
- Architecture Diagram 6
- Hardware Resources 10
- Software Resources 10
- Network Configuration 11

Solution Validation 11
- Overview 11
- Test Tools 13
- Deploying a KubeFATE Instance 13
- Leveraging an external Spark and HDFS Cluster 17
- Setting up a vSphere HA Pulsar Cluster 18
- Running a Federated Learning Workload 18
- Failure Scenarios 21

Sizing Guidelines 24
- VMware Cloud Foundation Infrastructure 24
- KubeFATE 24

Use Cases 26

Conclusion 27

Reference Architecture 28

About the Author 28

Appendix 29

Business Case

Artificial Intelligence (AI) adoptions have significant growth in more organizations and more industries. The evolving landscape of data collection challenges the traditional AI processing methods as worldwide data are increasingly in isolated islands. Likewise, data privacy and security regulations, such as General Data Protection Regulation (GDPR), impose a significant burden of compliance to collect and use this data. Federated learning has emerged as a distributed learning paradigm for collaboration between enterprises to solve isolated data problems while addressing the critical issue of data privacy and data governance. Federated learning is a state-of-the-art technology:

- Data of all parties is stored locally, ensuring that data privacy and compliance with laws and regulations.
- Multiple parties contribute data to develop a global model from which they can mutually benefit.
- All parties are of equal status.
- The modeling performance of federated learning is the same as, or (in case of user alignment or feature alignment of data) slightly different from, the modeling result achieved through aggregation of all datasets.
- Transfer learning ensures that knowledge transfer can also be achieved through the exchange of encryption parameters between data, even when users or features are not aligned. [1]

Federated learning allows multiple organizations or companies to build global common machine learning models without leaking any raw data. The data owners can share high-quality models that get exposed to a significantly wider range of data than they would on any single organization while preserving privacy and security.

FATE (Federated AI Technology Enabler) is an open-source project hosted by Linux Foundation. It provides a secure computing framework to support the federated AI ecosystem. It implements secure computation protocols based on homomorphic encryption and multi-party computation (MPC). It supports federated learning architectures and secure computation of various machine learning algorithms, including logistic regression, tree-based algorithms, deep learning, and transfer learning.

KubeFATE is designed to provision, orchestrate, operate, and manage FATE-based federated learning systems on Kubernetes in data centers or multi-cloud environments, and exploits the advantages of the cloud computing delivery model. It manages the system in the form of a Federated Learning Cluster (FLC), which includes the FATE service, all other services it depends on, as well as their configurations with cloud-native technology.

Organizations are undergoing an exceptional period of change. Driven by the relentless growth of data, the emergence of Kubernetes, and the demands of the hybrid cloud, both IT and DevOps teams face challenges that are both massive and inherently cross disciplinary.

VMware Cloud Foundation™ with VMware Tanzu™ accelerates Kubernetes infrastructure provisioning with full stack consisting of compute, storage, networking, and management. Through the automatic and reliable deployment of multiple workload domains, it increases admin productivity while reducing overall TCO to deliver a faster path to a hybrid cloud.

Deploying and operating a federated learning environment within traditional IT infrastructure can be challenging. The KubeFATE on VMware Cloud Foundation with Tanzu solution aims to address the complexity of AI solution integration with IT infrastructure to rapidly deploy federated learning systems and the underlying infrastructure with resiliency and security.

In this solution, we provide deployment procedures, design and sizing guidance, and best practices for enterprise infrastructure admins and application owners to provision and run KubeFATE on the Cloud Foundation platform.

Business Values

Here are the top 4 benefits for deploying and operating KubeFATE for federated learning on VMware Cloud Foundation with Tanzu.

- **Full stack integration**

  VMware Cloud Foundation's software stack lifecycle is automated and complete lifecycle management that greatly reduces risks and increases IT operational efficiency. Tanzu Kubernetes cluster deployment is fully integrated with the VMware vSphere® SDDC stack, including storage, networking, and authentication.

- **Consistent operations and infrastructure for hybrid and multi cloud**

[1] These bullet definitions are quoted from **1.3 Feasible Solutions to Data Privacy** in *White Paper on Federated Learning V2.0*.

This tool provides edge, private, and public cloud workload deployment options for a true hybrid cloud solution that maintains the flexibility of networking and topology. It allows enterprises to build, run, manage, connect, and protect any app on any cloud or across clouds with complete consistency of experience.

- **Security**

  FATE applies various security protocols, including homomorphic encryption, secret sharing, RSA, and Diffie-Hellman, to different algorithms to comprise requirements of security, audit, and law.

  Data of all parties are stored locally, ensuring data privacy and compliance with laws and regulations and no data leakage to the outside. All parties only interact with the intermediate results of encryption after processing local data in the process of modeling and reasoning to ensure the information security of all parties from the aspects of algorithm design, encryption algorithm strength, and communication security.

- **Automated end-to-end lifecycle management**

  KubeFATE will minimize the federated learning workload impact and downtime during the necessary patching and upgrading of the full private cloud stack using automated and self-managed services within the workload domain.

Key Results

This reference architecture is a showcase of operating and managing KubeFATE on VMware Cloud Foundation with Tanzu[2].

Key results can be summarized as follows:

- Empowers customers to securely provision and operate enterprise-grade federated learning in a production environment using VMware Cloud Foundation.
- Validates the deployment and management of KubeFATE by providing an end-to-end integration on Tanzu Kubernetes with a unified platform at any scale, on-premises, and in the cloud.
- Provides baseline performance benchmarks and sizing guidelines to achieve lower TCO.
- Identifies the steps required to ensure system resiliency and availability against various failures.

Audience

This solution is intended for IT admins and storage experts who are involved in planning, designing, deploying, and managing KubeFATE for federated learning on VMware Cloud Foundation with Tanzu. This document assumes that the reader is familiar with the concepts and operations of federated learning and VMware Cloud Foundation-related components.

Technology Overview

Solution technology components are listed below:

- VMware Cloud Foundation
  - VMware Cloud Foundation with Tanzu
  - VMware vSphere
  - VMware NSX® Data Center
- KubeFATE

---

[2] The reference architecture is validated on VMware Cloud Foundation with Tanzu, it also applies to vSphere with Tanzu.

VMware Cloud Foundation

VMware Cloud Foundation is full stack hyperconverged infrastructure solution that combines compute virtualization (VMware vSphere®), storage virtualization (VMware vSAN™), network virtualization (VMware NSX®), and cloud management and monitoring (VMware vRealize® Suite) into a single platform that can be deployed on-premises as a private cloud or multi-cloud in conjunction with VMware services that run in all major hyperscalers and VMware Cloud Providers. This documentation focuses on the private cloud use case. VMware Cloud Foundation can be deployed on-premises on a broad range of vSAN ReadyNode™ servers, on jointly engineered systems like Dell EMC VxRail, or consumed as a service in the public cloud from VMware Cloud on AWS, Azure VMware Solution, Google Cloud VMware Engine, and select VMware Cloud™ Providers. VMware Cloud Foundation bridges the traditional administrative silos in data centers, merging compute, storage, network provisioning, and cloud management to facilitate end-to-end support for application deployment.

VMware Cloud Foundation with VMware Tanzu

VMware Cloud Foundation with Tanzu is a full stack hyperconverged infrastructure solution that automates infrastructure deployment and lifecycle management of complex Kubernetes clusters alongside mission critical enterprise applications, including an embedded Kubernetes runtime environment that accelerates the development of modern applications.

VMware Cloud Foundation with Tanzu automates full-stack deployment and operation of Kubernetes clusters through integration with VMware Tanzu Kubernetes Grid™ and VMware Tanzu Mission Control. VMware Cloud Foundation with Tanzu helps to eliminate manual steps for host configuration, creating logical relationships, managing hypervisors for faster deployment of applications at scale. VMware Cloud Foundation with Tanzu provides a comprehensive hybrid cloud platform that bridges the gap between app developers and IT administrators. VMware Cloud Foundation can be deployed on-premises on a broad range of vSAN ReadyNode™ servers, on engineered systems like Dell EMC VxRail, or consumed as a service in the public cloud from VMware Cloud on AWS, Azure VMware Solution, Google Cloud VMware Engine, and select VMware Cloud™ Providers.

VMware Cloud Foundation with Tanzu is a major architectural upgrade to the industry's most advanced hybrid cloud platform. The most exciting feature added to the VMware Cloud Foundation architecture is the integration of Kubernetes directly into the vSphere Hypervisor, which delivers an entirely new set of VMware Cloud Foundation services, a new Kubernetes and RESTful API surface that empowers developers to have self-service access to Kubernetes clusters, vSphere Pods, virtual machines, persistent volumes, stateful services, and networking resources. These services include VMware Tanzu Kubernetes Grid plus infrastructure and automation services that provide the basis for the cloud infrastructure and container ecosystems to boost developer productivity. VMware Cloud Foundation with Tanzu represents a major advancement in cloud-native compute, storage, networking, and management to seamlessly support containers and VMs all within the same automated hybrid cloud infrastructure.

VMware vSphere

VMware vSphere is the next-generation infrastructure for next-generation applications, which provides a powerful, flexible, and secure foundation for business agility that accelerates the digital transformation to cloud computing and promotes success in the digital economy. VMware vSphere embeds containers and Kubernetes into vSphere, unifying them with virtual machines as first-class citizens. This enables all vSphere admins to become Kubernetes admins and easily deliver new services to their developers. VMware vSphere addresses key challenges faced by the IT admins in areas of lifecycle management, security, and performance and resiliency needed by business-critical applications, AI/ML applications, and latency-sensitive applications. With VMware vSphere, customers can run, manage, connect, and secure both traditional and cloud native applications in a common operating environment, across clouds and devices.

VMware NSX Data Center

VMware NSX® Data Center is the network virtualization and security platform that enables the virtual cloud network, a software-defined approach to networking that extends across data centers, clouds, and application frameworks. With NSX Data Center, networking and security are brought closer to the application wherever it is running, from virtual machines to containers to bare metal. Like the operational model of VMs, networks can be provisioned and managed independent of the underlying hardware. NSX Data Center reproduces the entire network model in software, enabling any network topology—from simple to complex multitier networks—to be created and provisioned in seconds. Users can create multiple virtual networks with diverse requirements, leveraging a combination of the services offered via NSX or from a broad ecosystem of third-party integrations ranging from next-generation firewalls to performance management solutions to build inherently more agile and secure environments. These services can then be extended to a variety of endpoints within and across clouds.

KubeFATE

To run a federated learning workload requires several services to cooperate, which brings some challenges to the management. The KubeFATE open source project aims to address these challenges. It helps manage and operate the Federated Learning Cluster (FLC) with cloud-native technology. Currently, it supports two types of backends to provision FLC: the docker-compose based development environment and the Kubernetes based production environment.

KubeFATE enables federated learning jobs to run across public, private, and hybrid cloud environments. See KubeFATE for more information.

Note: We assume that all parties involved in federated learning have agreed on the collaboration principles and the technology at the first stage in this KubeFATE solution.

Solution Configuration

This section introduces the resources and configurations:

- Architecture diagram
- Hardware resources
- Software resources
- Network configuration

Architecture Diagram

The following sections describe how the KubeFATE components are configured. Figure 1 shows the overall architecture of KubeFATE on VMware Cloud Foundation.

The configuration is composed of two dimensions. The first one is to provision and manage FLC with KubeFATE locally in each party. The other one is to bridge the local FLC to others of different parties.

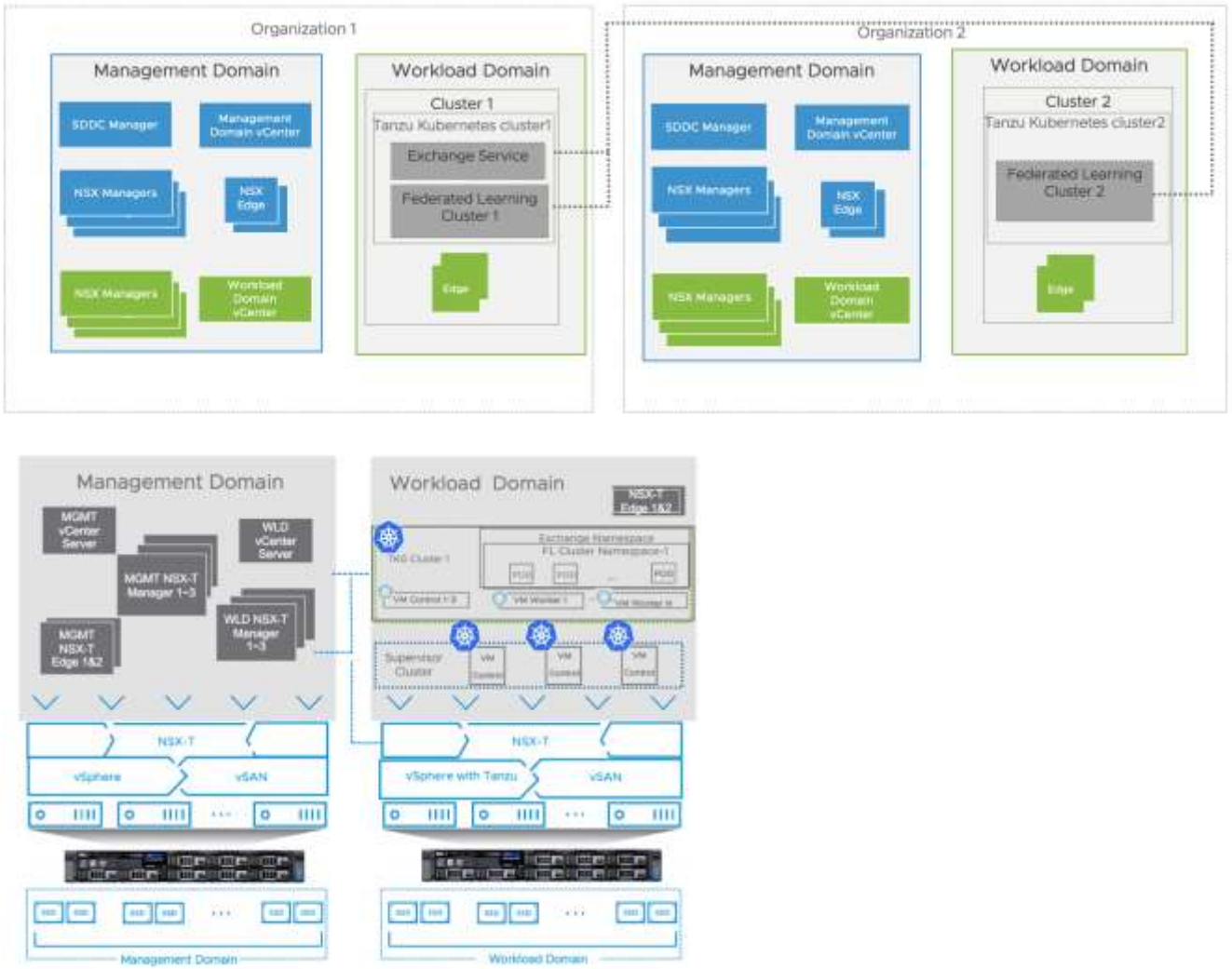


Figure 1. Architecture of KubeFATE on VMware Cloud Foundation

Figure 1 shows the architecture of a two-party KubeFATE cluster on VMware Cloud Foundation with Tanzu. In each organization, we deployed a VMware Cloud Foundation instance consisting of a management domain and a workload domain. The 4-node management domain cluster hosts multiple management virtual machines and appliances. For the workload domain, we created another 4-node cluster with workload management enabled and provisioned a Tanzu Kubernetes cluster. We deployed an FLC in a namespace in a VMware Tanzu Kubernetes cluster in each party, and we deployed the Exchange service in another namespace in one party. Similarly, we can deploy a multi-party KubeFATE cluster across geographies with more building blocks equivalent to Organization 2.

An Overview of KubeFATE Cluster

As Figure 2 shows, a KubeFATE cluster includes three components:

- A KubeFATE service to deploy/manage several Federated Learning Clusters.
- One or many FLCs to run federated learning workload. An FLC is a FATE cluster that includes the following components:
  - A FATE Flow service to schedule and manage federated learning jobs

- A MySQL database to store metadata
- An Nginx/Pulsar server to synchronize job status and data between different FLCs
- A Jupyter Notebook for users to build and run federated learning jobs
- A FATE Board to visualize the status of federated learning workload
- A Spark cluster to run the actual federated learning workload
- An HDFS cluster to store training dataset and intermediated results

For more component details, refer to the **Components Table** in Appendix.

- An optional Exchange service to manage the connection information between FLCs. The Exchange service can be deployed on Kubernetes and in a demilitarized zone **(**DMZ).

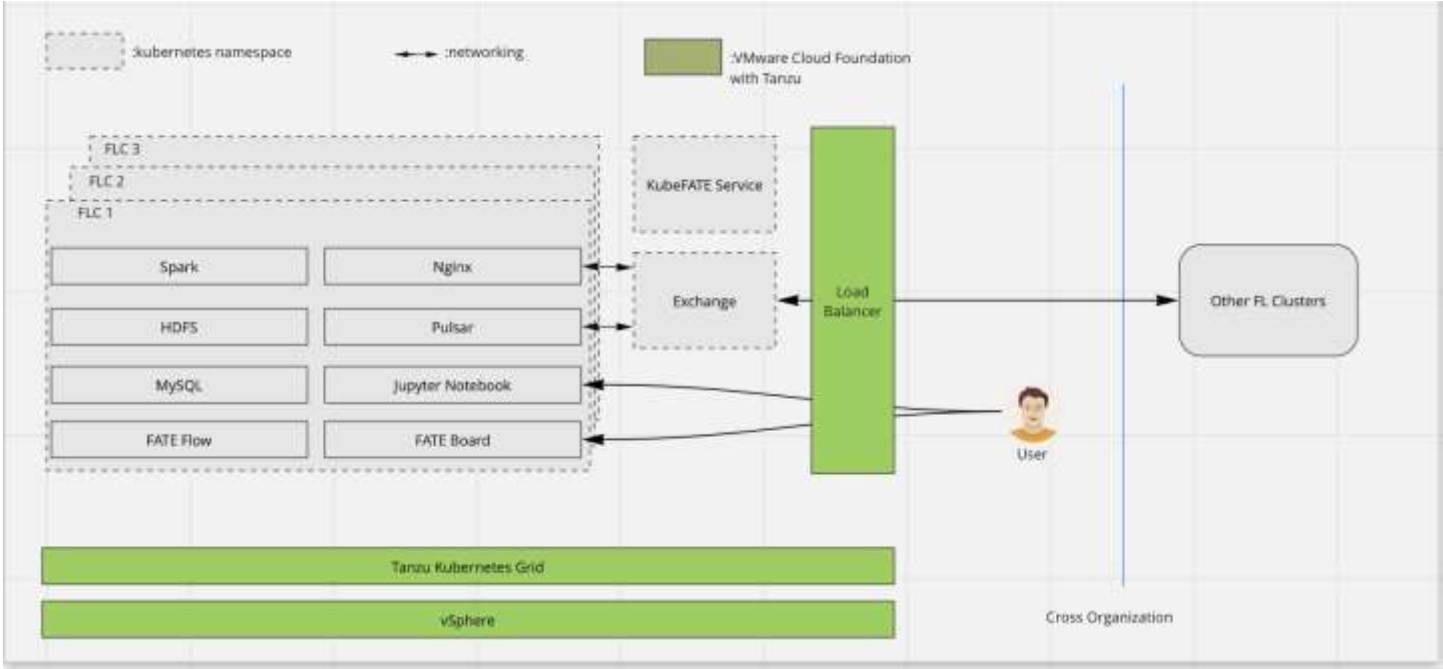


Figure 2. KubeFATE Components

Build Federation of FLCs

The typical topology of KubeFATE clusters of three parties is shown in Figure 3. Each FLC is managed by different organizations or different departments within the same organization. It is important to connect these FLCs as a federation for collaboration. Otherwise, each FLC is an isolated system.

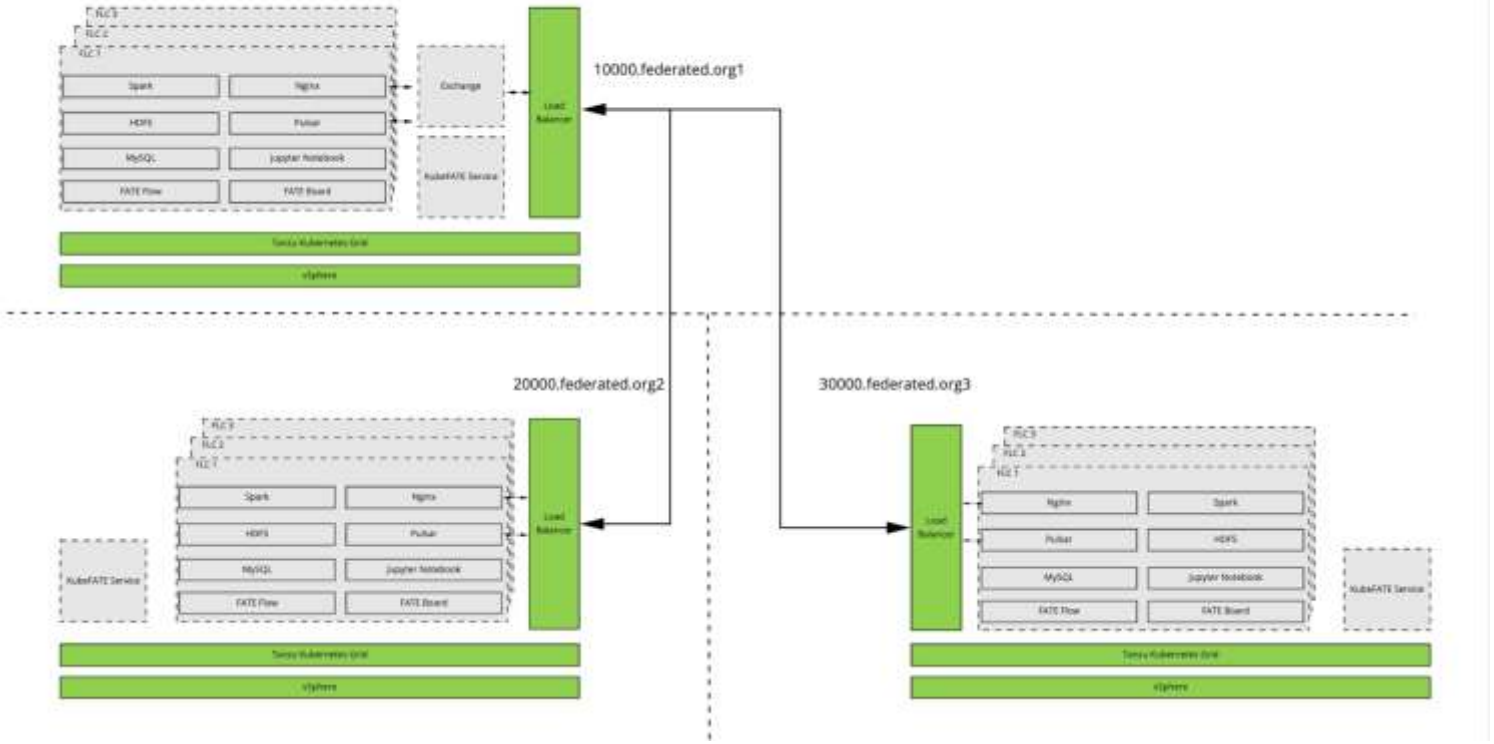


Figure 3. KubeFATE Clusters of Three Organizations

As Figure 3 shows, FLCs expose the following two services to each other:

- Nginx to transfer controlling data and job status to other clusters
- Pulsar to transfer encrypted model weights or gradients to other clusters

An Exchange service is deployed in an organization of the federation. It is used to manage the IP address and port number of the load balancer of each participant. The Exchange service is independent from the FLC.

The key steps to build federation among FLCs are as follows:

1. Deploy an exchange service
2. Connect each participant's FLC to the exchange service
3. Add IP addresses or domain names and ports of the loadbalancer of all participants in the exchange

Refer to *Deployment Guide of Multiple Parties* for more information.

Hardware Resources

KubeFATE is hardware agnostic and runs on a variety of hardware architectures. The following table provides a general description of a server of VMware Cloud Foundation workload domain used by KubeFATE.

Table 1. Hardware Configuration

PROPERTY	SPECIFICATION
CPU	KubeFATE is not CPU bound and is performant with any modern server-grade CPU, see VMware Cloud Foundation 4.2 supported *hardware*Sizing Guidelines.
RAM	KubeFATE is not memory bound and is performant with any modern speed memory.
Network adapter	Minimum 2 X10GbE NIC.
Storage adapter	Dedicated SAS/SATA Storage Controllers with JBOD support and RAID disabled.
Disks	Dedicated SAS/SATA drives.

Software Resources

Table 2 shows the software resources used in this solution.

Table 2. Software Resources

SOFTWARE	VERSION	PURPOSE
VMware Cloud Foundation	4.2	VMware Cloud Foundation provides integrated cloud infrastructure (compute, storage, networking, and security) and cloud management services to run both cloud native and traditional workloads. See *VMware Cloud Foundation* for details.
Kubernetes	1.18.15[3]	Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.
KubeFATE	1.6.1 or later	KubeFATE is designed to provision, orchestrate, operate, and manage FATE-based federated learning systems on Kubernetes in data centers or multi-cloud environments. It exploits the advantages of the cloud native delivery model.

[3] We used this version in the validation, all Tanzu Kubernetes Grid versions are supported.

Network Configuration

The Tanzu Kubernetes cluster configured with VMware NSX-T Data Center uses the software-based networks of the solution as well as an NSX Edge load balancer to provide connectivity to external services and DevOps users. The pods of the KubeFATE cluster are placed in a namespace.

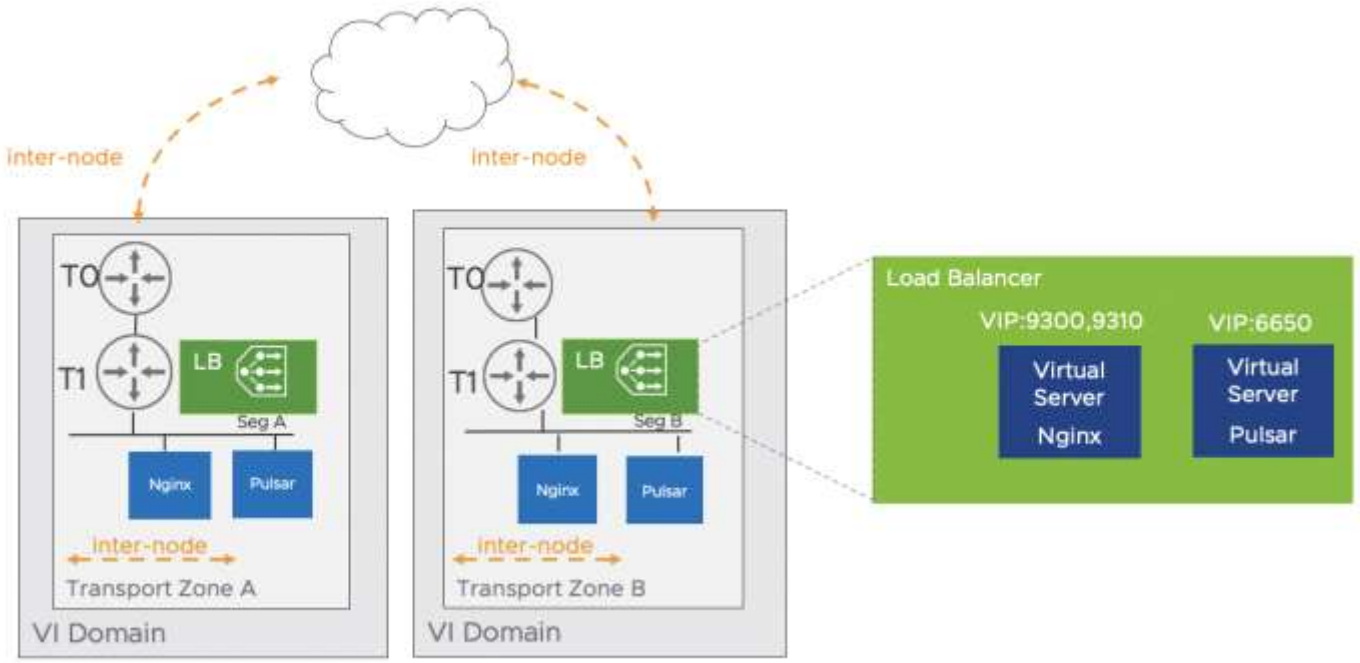


Figure 4. Network Configuration

Solution Validation

Overview

VMware Cloud Foundation with Tanzu automates the full-stack deployment and operation of Kubernetes clusters through integration with VMware Tanzu Kubernetes, makes it easy to stand up the underlying vSphere infrastructure, set up NSX and the NSX Edge Clusters.

Figure 5 shows the SDDC Manager UI to deploy the Workload Management for vSphere with Kubernetes.

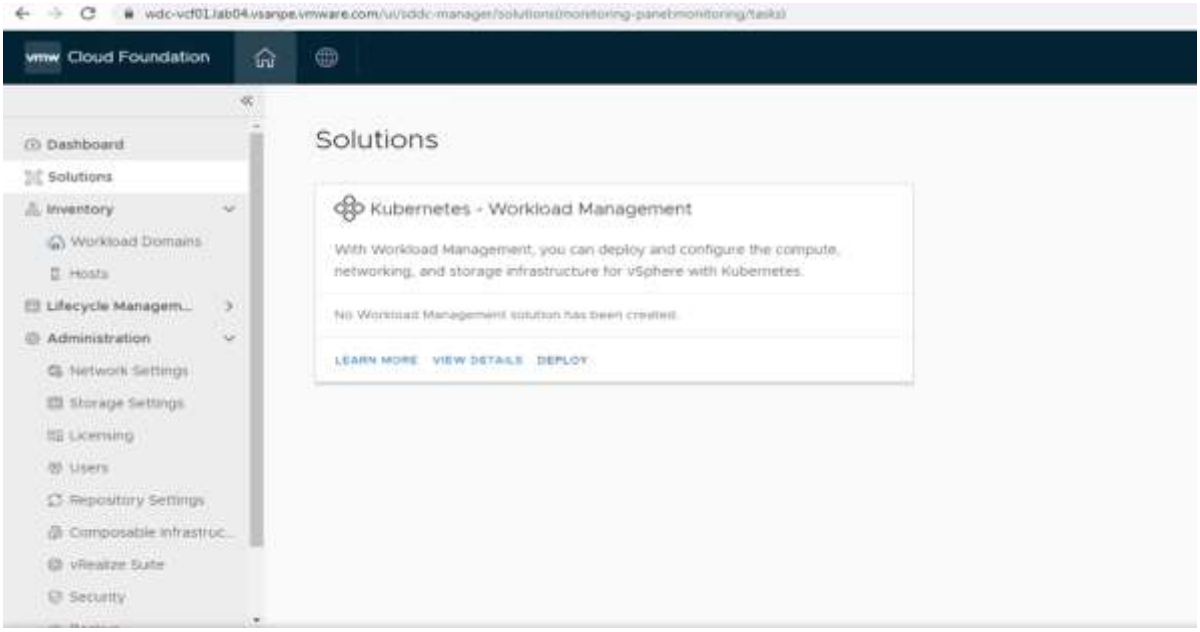


Figure 5. SDDC Manager integrated Kubernetes–Workload Management Deployment

With Workload Management enabled, we can easily enable Tanzu Kubernetes clusters.

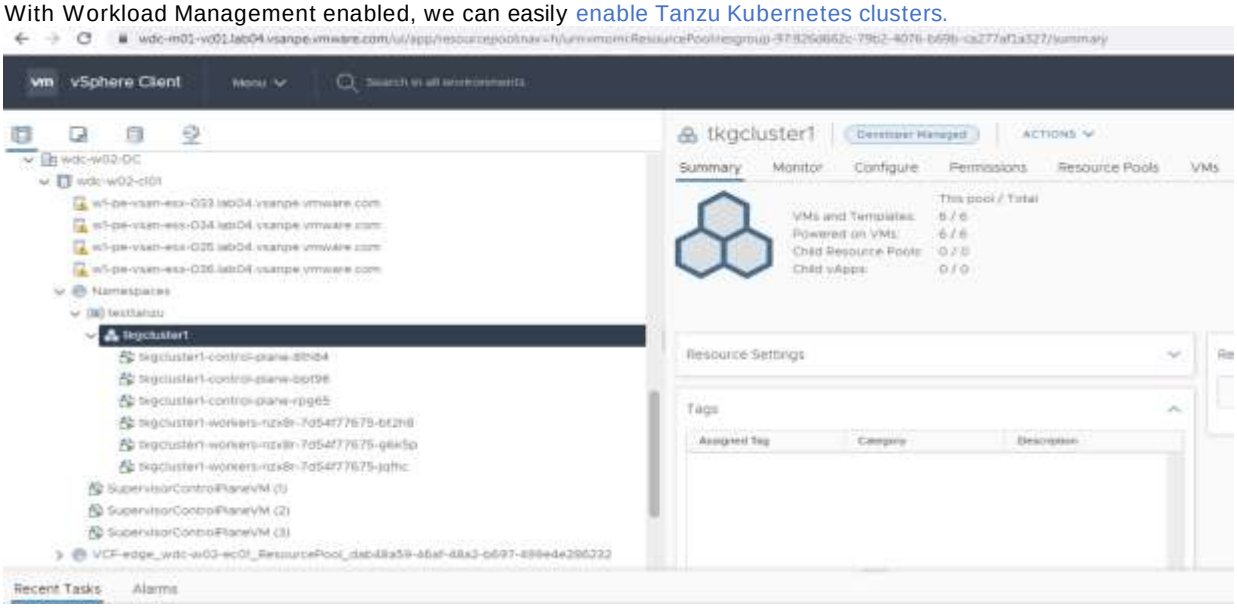


Figure 6. Tanzu Kubernetes Cluster Overview

Figure 6 shows the Tanzu Kubernetes cluster with 3 control nodes and 3 worker nodes deployed in a namespace of a vSphere Cluster, we can easily deploy large-scale Tanzu Kubernetes clusters in the same way.

The validation is a showcase of VMware Cloud Foundation with Tanzu for operating and managing KubeFATE federated learning platform in a fully integrated SDDC environment.

Key results can be summarized as follows:

- Deploying Instance: Quick guide for multi-party network configuration, KubeFATE Cluster deployment, and validation of multi-party connection.
- Running a federated learning workload: Showcases federated training workflow using integrated Jupyter Notebook, FATE Board for job management, model evaluation and prediction.
- Resilience Tests: Proves the solution resilience to guarantee the service continuity and stability of KubeFATE in failure scenarios such as disk and host failures.
- Best Practices: Provides best practices to deploy the infrastructure and sizing guidelines of CPU/memory/storage/network bandwidth planning to target a workload of a given scale.
- Use cases: Categorizes a broad range of use cases in both vertical federated learning and horizontal federated learning.

Test Tools

We used the following monitoring tools and benchmark tools in the solution testing:

Monitoring tools

- Kubernetes CLI tools
kubectl + vSphere plugin

  The Kubernetes CLI Tools download package includes two executables: the standard open source kubectl and the vSphere Plugin for kubectl. The kubectl CLI has a pluggable architecture. The vSphere Plugin for kubectl extends the commands available to kubectl so that you can connect to the Supervisor Cluster and to the Tanzu Kubernetes clusters using vCenter Single Sign-On credentials.

- Tanzu Kubernetes Cluster Status
You can *monitor the status of Tanzu Kubernetes clusters using the vSphere Client*.

- vSAN Health Check
vSAN Health Check delivers a simplified troubleshooting and monitoring experience of all things related to vSAN. Through the vSphere client, it offers multiple health checks specifically for vSAN, including cluster, hardware compatibility, data, limits, and physical disks.

Workload generation and testing tools

In our baseline benchmark testing, we verified some classic algorithms like logistic regression, gradient boosting tree, and neural network. For the neural network, we used the public *MNIST dataset*. For others, we used scripts from the FATE's open source community to generate random datasets. For more details about the testing setup, refer to this *repo*.

Deploying a KubeFATE Instance

The deployment of KubeFATE clusters usually includes two or multiple parties, where each party is deployed in a Tanzu Kubernetes cluster and managed by the KubeFATE service running in the cluster. The networking information of all parties is configured in the Exchange component, which only needs to be deployed in one party. As it is logically a standalone service, the Exchange component can even be deployed in a separate cluster. For simplicity, it is deployed alongside one of the parties in the validation example.

When provisioning a Tanzu Kubernetes cluster, it is required to configure a separate volume disk to store the container images. The recommended size of the separate disk is 32GB.

Assuming the underlying Tanzu Kubernetes clusters have been provisioned, the following steps depict how to set up KubeFATE clusters to perform federated learning tasks. Note that the following deployment uses pure HTTP as transportation protocol as an example. In real deployment cases, especially for banking or financial companies that have complicated security setup with hardware and third-party providers, we keep pure HTTP connection for the Kubernetes level. It is suggested to reach your hardware and third-party providers for boundary and transportation protection.

Step 1: Deploying the KubeFATE service

KubeFATE orchestrates the FLC of a party, managing the lifecycle of FATE clusters and the Exchange component.

One of the major prerequisites of using KubeFATE is setting up an ingress controller. The *official example* can be used as a reference. The following command is used to install an Nginx ingress controller:

```
$ helm install ingress-nginx --set controller.service.type=LoadBalancer --set podSecurityPolicy.enabled=true ingress-nginx/ingress-nginx
```

Once the command completes, take note of the external IP of the ingress controller service.

The KubeFATE project repository contains a detailed *step-by-step guide* of the complete workflow. One can follow Deploying a KubeFATE Instance to install the KubeFATE service, namely setting up the RBAC roles and creating the deployments and services on Tanzu Kubernetes cluster.

After installation, the service deployments and ingress information can be verified as below. The KubeFATE service is exposed via a Kubernetes Ingress and is associated with a domain name. In this example, the domain name is "example.com". This can be configured in the *yaml file* while installing KubeFATE. To access the service, you can configure the DNS server to point the domain to the ingress external IP, "10.159.229.67" in this example.

```
[root@localhost ~]# kubectl get deployment -n kube-fate
NAME       READY   UP-TO-DATE   AVAILABLE   AGE
kubefate   1/1     1            1           47d
mariadb    1/1     1            1           47d
[root@localhost ~]# kubectl get ingress -n kube-fate
NAME       CLASS    HOSTS         ADDRESS         PORTS   AGE
kubefate   <none>   example.com   10.159.229.67   80      47d
[root@localhost ~]# kubectl get svc
NAME                                 TYPE           CLUSTER-IP       EXTERNAL-IP     PORT(S)                      AGE
ingress-nginx-controller             LoadBalancer   10.110.119.205   10.159.229.67   80:31138/TCP,443:31557/TCP   47d
ingress-nginx-controller-admission   ClusterIP      10.100.141.56    <none>          443/TCP                      47d
```

Figure 7. KubeFATE Deployment

Many of the next steps are performed using the KubeFATE CLI command to interact with the KubeFATE service. Before proceeding, as suggested in the section of "*KubeFATE Command Line Connection*", make sure the CLI command can connect to the service without error. The DNS service of the CLI machine should resolve the domain name to the IP address of the ingress controller service. Alternatively, the machine's host file should contain an entry to map the domain name to the IP address of the ingress controller service.

Step 2: Deploying an FLC with Exchange Component

Assume this party is selected to deploy the Exchange component. Two installations are needed:

A. Deploying Exchange component
   - Create a namespace fate-exchange in the Tanzu Kubernetes cluster.
   - Prepare the certificate files as suggested in KubeFATE documents, including *generating the certificates* and *importing them into the fate-exchange namespace*.
   - Prepare a fate-exchange.yaml file containing the configuration for KubeFATE to deploy the Exchange component. The *yaml file* is customized from the *example in the KubeFATE project*.
   - Use KubeFATE CLI to install the Exchange component:

     ```
     ./kubefate cluster install -f fate-exchange.yaml
     ```

     The Exchange component does not contain the network configuration of any party. Such information will be updated once all FLCs are deployed.

Below is the information of a deployed Exchange component.

```
[root@localhost ~]# kubectl get deployment -n fate-exchange
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
nginx            1/1     1            1           93s
traffic-server   1/1     1            1           93s
[root@localhost ~]# kubectl get svc -n fate-exchange
NAME             TYPE           CLUSTER-IP       EXTERNAL-IP     PORT(S)                         AGE
nginx            LoadBalancer   10.104.112.188   10.159.229.69   9300:32176/TCP,9310:31934/TCP   100s
traffic-server   LoadBalancer   10.101.189.138   10.159.229.68   443:32315/TCP                   100s
```

Figure 8. Exchange Component

The external IP address and the port of the Exchange service will be used by the FLC deployment.

B. Deploying the FLC

- Create a namespace for this FLC in the Tanzu Kubernetes cluster. In this validation example, fate-30000 is used.
- Follow the same steps as the Exchange component, generate the corresponding certificates and import them into the cluster as Kubernetes secrets.
- Prepare a cluster-spark-pulsar.yaml file containing the configuration to install a FATE cluster. This validation uses *the example yaml file* for this party.

NOTE: There are some key considerations for better integration with the Tanzu Kubernetes cluster:

- "podSecurityPolicy" needs to be set to true as PodSecurityPolicy Admission Controller is enabled in Tanzu Kubernetes clusters.
- The service type of the "nginx" component should be LoadBalancer so that this party can communicate with the Exchange service.
- Similarly, "publicLB" of the "pulsar" component needs to be enabled.
- The "exchange" fields of the "nginx" and "pulsar" components are the external IP address and port of the related Exchange service.
- "persistence" needs to be set to true and "storageClass" should be configured for all the components to preserve key data in the event of pod restarting. Refer to the Failure Scenarios section for more detailed discussions.
- Use the KubeFATE CLI command to install the FLC by providing the yaml file:

$./kubefate cluster install -f cluster-spark-pulsar.yaml

Below is a deployed FLC of one party.

```
[root@localhost ~]# kubectl get deployment -n fate-30000
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
client         1/1     1            1           3m15s
datanode       1/1     1            1           3m15s
mysql          1/1     1            1           3m15s
namenode       1/1     1            1           3m15s
nginx          1/1     1            1           3m15s
pulsar         1/1     1            1           3m15s
python         1/1     1            1           3m15s
spark-master   1/1     1            1           3m15s
spark-worker   1/1     1            1           3m15s
```

Figure 9. FLC Deployment

The below command shows that the Nginx external IP address is 10.159.229.72, and the pulsar-public-tls external IP address is 10.159.229.73. These IP addresses will be used by the Exchange service later.

```
[root@localhost ~]# kubectl get svc -n fate-30000
NAME                TYPE           CLUSTER-IP       EXTERNAL-IP     PORT(S)                                AGE
datanode            ClusterIP      10.99.62.41      <none>          9000/TCP,9870/TCP,50070/TCP            3m21s
fateboard           ClusterIP      10.100.169.101   <none>          8080/TCP                               3m21s
fateflow            ClusterIP      10.110.45.193    <none>          9360/TCP,9380/TCP                      3m21s
fateflow-client     ClusterIP      10.98.154.51     <none>          9360/TCP,9380/TCP                      3m21s
mysql               ClusterIP      10.97.198.14     <none>          3306/TCP                               3m21s
namenode            ClusterIP      10.96.134.254    <none>          9000/TCP,9870/TCP,50070/TCP            3m21s
nginx               LoadBalancer   10.99.129.226    10.159.229.72   9300:32434/TCP,9310:32081/TCP          3m21s
notebook            ClusterIP      10.102.0.117     <none>          20000/TCP                              3m21s
pulsar              ClusterIP      10.97.254.149    <none>          6650/TCP,6651/TCP,8080/TCP,8081/TCP    3m21s
pulsar-public-tls   LoadBalancer   10.101.72.155    10.159.229.73   6651:30508/TCP                         3m21s
spark-client        ClusterIP      10.105.65.201    <none>          8080/TCP,7077/TCP,6066/TCP             3m21s
spark-master        ClusterIP      None             <none>          8080/TCP,7077/TCP,6066/TCP             3m21s
spark-worker-1      ClusterIP      None             <none>          8081/TCP                               3m21s
```

Figure 10. External IP Address of Nginx and Pulsar Service

Using KubeFATE CLI, one can list the installed components. This includes the Exchange service and the FLC, namely fate-exchange and fate-30000 in the below example.

```
[root@localhost ~]# ./kubefate cluster ls
UUID                                   NAME           NAMESPACE      REVISION   STATUS   CHART          ChartVERSION   AGE
de5bd453-fc54-4f63-a79c-f7422383a13c   fate-exchange  fate-exchange  1          Running  fate-exchange  v1.6.1         27m
5a63effa-41fd-41e9-b98d-34af4f0b06bd   fate-30000     fate-30000     1          Running  fate           v1.6.1         5m59s
```

Figure 11. KubeFATE Cluster Info

An FLC has a Jupyter Notebook service (notebook) and a Fate Board service (fateboard) that can also be accessed via the ingress controller service. In this example, the domain name "30000.notebook.example.com" and "30000.fateboard.example.com is used for the Jupyter Notebook service and the FATE Board service, respectively. The domain names can be configured in the yaml file for deploying the FLC. Similar to the KubeFATE service, the DNS service should be able to resolve these domain names to the ingress' external IP address. The following sections interact with an FLC using the notebook and fateboard service.

Step 3: Deploying more federating learning clusters without Exchange service

Repeat Step 1 and Step 2 to deploy multiple parties.

In Step 1, the KubeFATE service needs to be deployed to every party's Tanzu Kubernetes cluster and KubeFATE CLI should be used to connect to the KubeFATE service for the subsequent steps.

In Step 2, since an Exchange component is already deployed in the first party, no Exchange deployment is required for other parties so Step 2A can be skipped. The FLC name and namespace should be planned and used accordingly. For example, the FLC name of the second party can be named as "fate-40000", as used in *this example*.

Step 4: Updating each FLC's exposed IP address in the route table of the Exchange service

The Exchange route table should be updated to reflect the external IP addresses of all the FLCs. This can be done by *updating the fate-exchange.yaml file* and invoking the KubeFATE CLI command as follows:

```
$ ./kubefate cluster update -f fate-exchange.yaml
```

After updating the Exchange route table, the Exchange service needs to reload the latest configuration. Use the following commands to get the pod of the service:

```
$ kubectl get pod -n fate-exchange
```

Look for the pod whose name starts with "traffic-server-". Then run the reloaded command in the pod:

$ kubectl exec <traffic-server pod name> -n fate-exchange -- /opt/trafficserver/bin/traffic_ctl config reload

If the Exchange service has multiple pod instances, only one instance of the pod needs to run the above command.

Step 5: Accessing FATE cluster and validating the connection of multiple parties

Use the below command to get the name of the FATE Flow pod:

$ kubectl get pod -n fate-30000

Look for the pod whose name starts with "python-". Next, run the below command in the pod to invoke a toy_example script to validate the connection:

$ kubctl exec –it <python- pod name> -c python –n fate-30000 -- /bin/bash

Figure 12 shows the steps to verify the connection between two parties (party identifiers are 30000 and 40000 respectively) from the fate-30000 cluster. The script should print the generated job information, and start waiting for the job to finish:

```
[root@localhost ~]# kubectl exec -it python-b96f44cdc-cqmrb -c python -n fate-30000 -- /bin/bash
(app-root) bash-4.2# python /data/projects/fate/examples/toy_example/run_toy_example.py 30000 40000 1 -b 2
stdout:{
    "data": {
        "board_url": "http://fateboard:8080/index.html#/dashboard?job_id=202108260746463836769&role=guest&party_id=30000",
        "job_dsl_path": "/data/projects/fate/jobs/202108260746463836769/job_dsl.json",
        "job_id": "202108260746463836769",
        "job_runtime_conf_on_party_path": "/data/projects/fate/jobs/202108260746463836769/guest/job_runtime_on_party_conf.json",
        "job_runtime_conf_path": "/data/projects/fate/jobs/202108260746463836769/job_runtime_conf.json",
        "logs_directory": "/data/projects/fate/logs/202108260746463836769",
        "model_info": {
            "model_id": "guest-30000#host-40000#model",
            "model_version": "202108260746463836769"
        },
        "pipeline_dsl_path": "/data/projects/fate/jobs/202108260746463836769/pipeline_dsl.json",
        "train_runtime_conf_path": "/data/projects/fate/jobs/202108260746463836769/train_runtime_conf.json"
    },
    "jobId": "202108260746463836769",
    "retcode": 0,
    "retmsg": "success"
}


job status is running
job status is running
```

Figure 12. Run Toy-example to Verify the Connection

Typically, after a few minutes, depending on the network conditions between the parties, the script should finish with success. It verifies the network setup of the two parties is correct:

```
job status is running
job status is running
[INFO] [2021-08-26 07:46:52,957] [14898:139814899767104] - secure_add_guest.py[line:99]: begin to init parameters of secure add example guest
[INFO] [2021-08-26 07:46:52,958] [14898:139814899767104] - secure_add_guest.py[line:102]: begin to make guest data
[INFO] [2021-08-26 07:46:59,590] [14898:139814899767104] - secure_add_guest.py[line:105]: split data into two random parts
[INFO] [2021-08-26 07:47:02,621] [14898:139814899767104] - secure_add_guest.py[line:108]: share one random part data to host
[INFO] [2021-08-26 07:47:04,501] [14898:139814899767104] - secure_add_guest.py[line:111]: get share of one random part data from host
[INFO] [2021-08-26 07:47:10,160] [14898:139814899767104] - secure_add_guest.py[line:114]: begin to get sum of guest and host
[INFO] [2021-08-26 07:47:13,725] [14898:139814899767104] - secure_add_guest.py[line:117]: receive host sum from guest
[INFO] [2021-08-26 07:47:13,830] [14898:139814899767104] - secure_add_guest.py[line:124]: success to calculate secure_sum, it is 2000.0000000000005
```

Figure 13. Job Status

For detailed information, including KubeFATE CLI reference and deployment yaml configuration values, refer to https://github.com/FederatedAI/KubeFATE/tree/master/k8s-deploy

Leveraging an external Spark and HDFS Cluster

A workable FLC is heavily relied on Spark, HDFS, and Pulsar services. By default, the KubeFATE service provides the capability to provision these services; however, it does not guarantee high availability (HA) and optimization for these services.

It is highly recommended for users to deploy their Spark and HDFS clusters according to the official documents, respectively.

Once the Spark and HDFS clusters are ready, the FLC can leverage them in federated learning jobs. As there are lots of configurations involved, refer to *FATE On Spark – Leverage the external cluster* for more details.

Setting up a vSphere HA Pulsar Cluster

By default, the KubeFATE service deploys a standalone Pulsar service; however, the standalone Pulsar service does not have high availability. To set up a Pulsar service with HA, a Helm chart is provided with this white paper to deploy Pulsar service with HA on Kubernetes. For more details, refer to the pulsar deployment. See *FATE On Spark – Leverage the external cluster* about how to use the external Pulsar cluster in FLC.

Running a Federated Learning Workload

In the Deploying a KubeFATE Instance section, two FLCs with ID 30000 and 40000 were created. Next, we will run a workload on them. We choose the dataset (https://www.kaggle.com/mlg-ulb/creditcardfraud) containing transactions of credit cards in September 2013 by European cardholders to showcase federated learning workflow using Jupyter Notebook and FATE Board to monitor jobs and to visualize the result of model evaluation and prediction.

The FATE open source project provides out-of-the-box components like feature engineering and machine learning algorithms for users to compose a job. In this example, we will use the homogeneous logistic regression algorithm to train a model. For more details about the available components, refer to the *FATE repo.*

The dataset can be split into two halves: "datasets_guest" and "datasets_host". Each half of the dataset can be used by one of the two involved parties (party identifies are 30000 and 40000), respectively.

Next, we upload "datasets_guest" and "datasets_host" to the FLC 30000 and 40000 through Jupyter Notebook, respectively. After the feature engineering of the dataset is finished by each party, we can start a federated learning job from party 30000 and evaluate the output model. All scripts and commands could be found in the *repo*.

Data Preparation

Download the dataset "creditcard.csv" and follow the "readme" to split the dataset into two halves.

The following operations except the job submission are required to be performed on both FLC 30000 and 40000. However, since they are identical operations, we only show the workflow of the party 30000.

Uploading data to Jupyter Notebook

FLC has an integrated Jupyter Notebook service for editing the federated learning job. It can be accessed from the configured hostname such as 30000.notebook.example.com.

Figure 14. Uploading Data to Jupyter Notebook

Preprocessing the dataset and uploading it to the FLC

Just like traditional machine learning, before the training is running, it needs preprocessing of the dataset, for example, normalization and null value handling. After data preprocessing, the dataset needs to be uploaded to the FLC for further usage in the federated learning workload.

```
In [ ]: # simple feature engineering
df = pd.read_csv('/Examples/Pipeline/notebooks/creditcard_guest.csv')

robust_scale = RobustScaler()
df['Scaled_Amount'] = robust_scale.fit_transform(df['Amount'].values.reshape(-1,1))
df['Scaled_Time'] = robust_scale.fit_transform(df['Time'].values.reshape(-1,1))

Scaled_Amount = df['Scaled_Amount']
Scaled_Time = df['Scaled_Time']
IDs = [i + 1 for i in range(len(credit_card))]

df.drop(['Scaled_Amount','Scaled_Time'], axis=1, inplace=True)
df.insert(0, 'id', IDs)
df.insert(1, 'Scaled_Amount', Scaled_Amount)
df.insert(2, 'Scaled_Time', Scaled_Time)

df.drop(['Time','Amount'], axis=1, inplace=True)

df.to_csv('/Examples/Pipeline/notebooks/creditcard_guest_scaled.csv')
```

Figure 15. Preprocessing Dataset

As Figure 16 shows, we can upload the dataset with the *FATE's Pipeline* Python library.

```
In [ ]: guest = 10000

# spark + pulsar
backend = 2
work_mode = WorkMode.CLUSTER

partition = 4

guest_train_data = {"name": "dataset_guest", "namespace": f"experiment"}

guest_eval_data = {"name": "dataset_guest", "namespace": f"experiment"}

pipeline_upload = PipeLine().set_initiator(role="guest", party_id=guest).set_roles(guest=guest)
# add upload data info
# original csv file path
pipeline_upload.add_upload_data(file=os.path.join('/Examples/Pipeline/notebooks/creditcard_guest_scaled.csv'),
                                table_name=host_train_data["name"],
                                namespace=host_train_data["namespace"],
                                head=1, partition=partition)
```

Figure 16. Uploading Data to FLC

Job Definition and Submission

After the data is ready, a user can continue to define the training job. We can develop federated learning models conveniently with FATE's Pipeline Python library in a few lines. The sample code below shows how to construct a homogenous logistic regression job to train pre-uploaded data. The code to evaluate the output model is added. Once the job is finished, the output model is stored on FLC with a unique identifier for later usage.

```
In [ ]: # initialize pipeline
pipeline = PipeLine()

# set job initiator
pipeline.set_initiator(role="guest", party_id=guest)
# set participants information
pipeline.set_roles(guest=guest, host=host, arbiter=arbiter)

# add components to pipeline, in order of task execution
pipeline.add_component(reader_0)
pipeline.add_component(reader_1)

pipeline.add_component(dataio_0, data=Data(data=reader_0.output.data))
pipeline.add_component(dataio_1, data=Data(data=reader_1.output.data),
                       model=Model(dataio_0.output.model))

homo_lr_0 = HomoLR(name='homo_lr_0', **param)
pipeline.add_component(homo_lr_0, data=Data(train_data=dataio_0.output.data,
                                            validate_data=dataio_1.output.data))

evaluation_0 = Evaluation(name="evaluation_0", eval_type="binary")
evaluation_0.get_party_instance(role='host', party_id=host).component_param(need_run=False)
pipeline.add_component(evaluation_0, data=Data(data=homo_lr_0.output.data))

# compile pipeline once finished adding modules, this step will form conf and dsl files for running job
pipeline.compile()

# fit model
job_parameters = JobParameters(backend=backend, work_mode=work_mode)
pipeline.fit(job_parameters)
# query component summary
import json
print(json.dumps(pipeline.get_component("homo_lr_0").get_summary(), indent=4))
```

homo lr

model evaluation

train model

Figure 17. Key Definition for the Job

The FATE Board portal can be used to create and view federated training jobs. It can be accessed from a browser through URL like 30000.fateboard.example.com. In our example, a user can check the status of a job and the evaluation results.

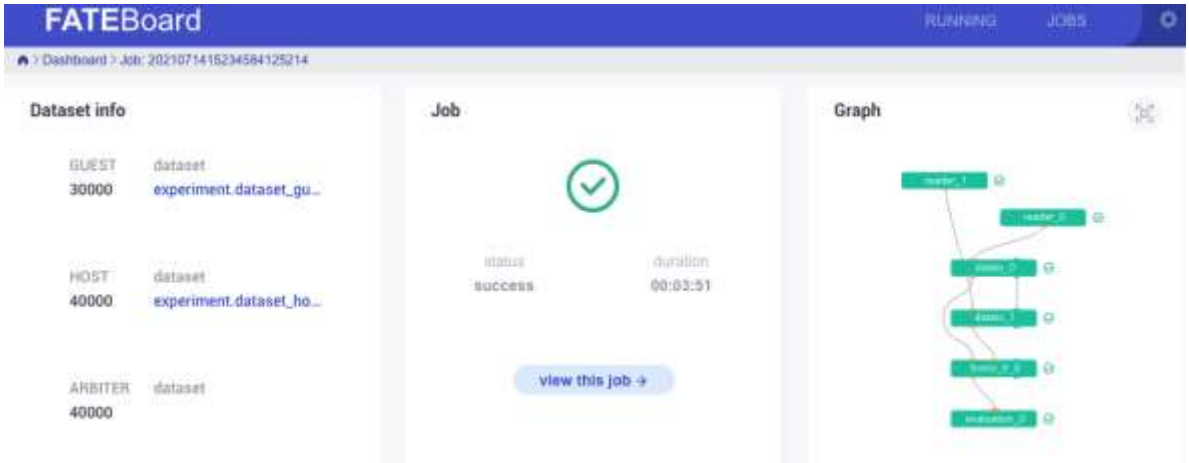


Figure 18. Job Status

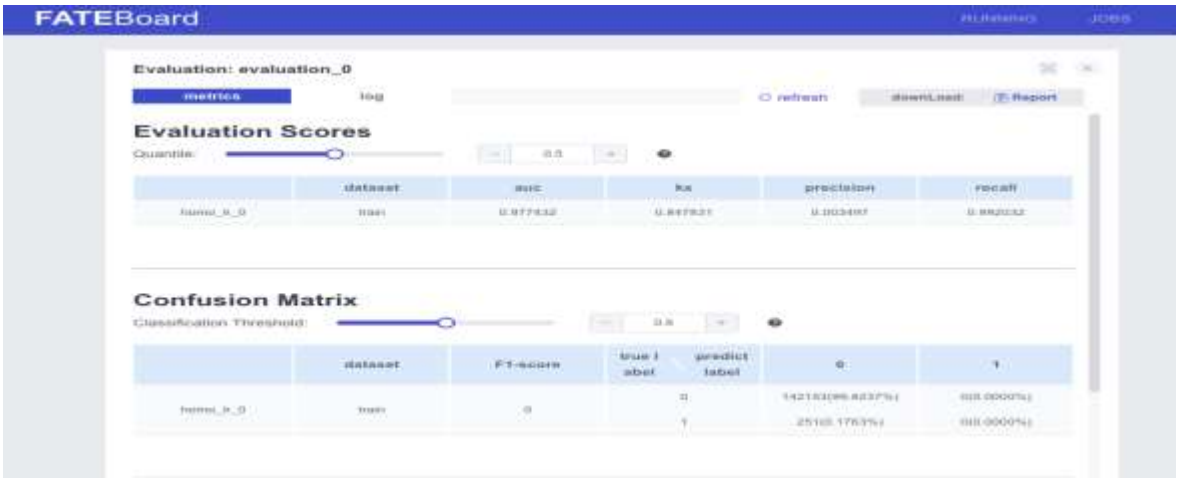


Figure 19. The Evaluation Output

Prediction

After successfully running a training job, we can define a job to perform prediction with the output model. After the job is finished, we can download the prediction result using the FATE API or from the FATE Board portal.

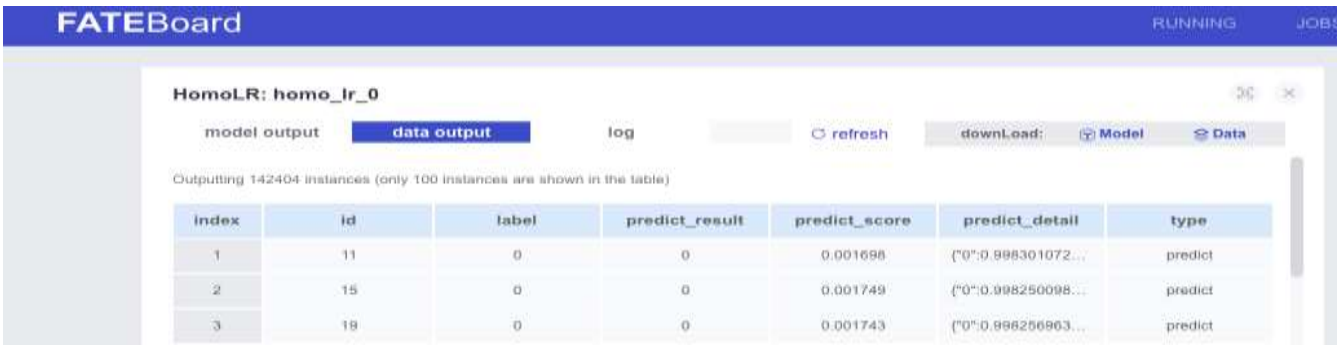


Figure 20. Prediction Result

Failure Scenarios

This section introduces the failure scenarios and the behavior of failure handling. This section includes:

- KubeFATE component failures
- Single party failures in an FL training job
- Host and VM failures
- Disk failure

KubeFATE Component Failures

Using KubeFATE, an FLC and all its components are deployed in the form of Kubernetes pods and organized as deployments and services. A component failure means its related pod failed, either due to the underlying failure of the host or VM, or the unexpected errors in the corresponding containers. The pod level resiliency provided by Tanzu Kubernetes Grid Service and Kubernetes native reconciling mechanism guarantees that when such a pod failure event occurs, it will be restarted, or a new pod will be launched to replace the failed one. It takes less than 30 seconds for an FLC pod to restart and become fully functional.

When a job is running on FLC, the resiliency of the job is validated as the components below are configured with necessary persistent volumes to store key data:

- The components in the Exchange service are all stateless and can be replicated to multiple instances. Restarting any of them will not impact the continuation of a training job.
- Since all necessary stateful data are stored in the underlying volume and preserved across pods restart or recreation, users can create new training jobs, view job history, and download trained models and data after such events without any manual intervention.
- When restarting events of FLC pods occur during ongoing training jobs, training jobs can resume in most cases. The table below lists the related pods and their impact on ongoing training jobs when the pods restart.

Table 4. Impact of Component Failures

Component Failures	Impact on the Training Job
FATE Flow	The job hung and eventually reported as a timeout. Users must re-submit the job to rerun from the beginning.
MySQL	The job continues to finish automatically.
Nginx	
Pulsar	
HDFS datanode	
Spark master	
Spark worker	
HDFS namenode	The job fails but can be resumed manually.

In the case of FATE Flow failure where the job will hang and timeout, a FATE training job timeout is 72 hours by default. This can be configured on a per-job basis by providing the "`"timeout": <value in seconds>`" setting in the job parameters field during the job submission.

To manually resume a failed training job, the FATE Board Web UI provides a retry link that can be used to continue the job from the previously failed step.

Figure 21. Retry a Failed Job

After clicking the retry link, the job will resume from the step where it failed. If the dependent components are back online, the job will finish successfully.

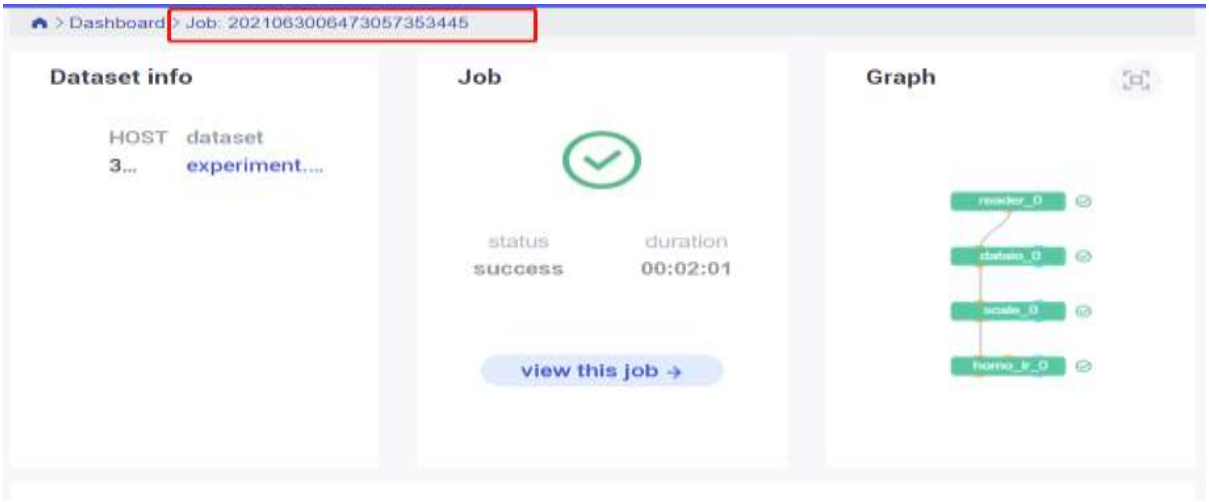


Figure 22. Status of a Retried Job

Single Party Failures in an FL Training Job

An FL training job requires multiple participants. From the perspective of a multi-party training job, several types of failures can happen to one participant. The table below lists how a training job involving multi-party ends if some error occurs at one participant.

Table 5. Single Party Failures and Expected Results

Failure type	Training job
Network disconnection	The job continues to finish if the network issue is resolved. Otherwise, the job will be reported as a timeout.
FATE Flow failure	The job cannot resume and will be reported as a timeout.
Job encountered an error in one party	The job will be reported as a failure on all other parties.

Host and VM Failures

When host failures or VM failures occur, the pods running on the host or VM will be affected. It is validated that with vSphere HA and Tanzu Kubernetes Grid Service, the impacted pods can be restarted promptly.

- In the event of a host failure, the affected VMs will be restarted on other hosts by vSphere HA.
- When a VM of Tanzu Kubernetes cluster fails, it will be restarted automatically.
  - vSphere HA will restart the VM if the VM crashes. This typically takes less than 1 minute after the event is detected.
  - Tanzu Kubernetes Grid Service can restart the VM if the VM is shut down or powered off. This can take less than 5 minutes.

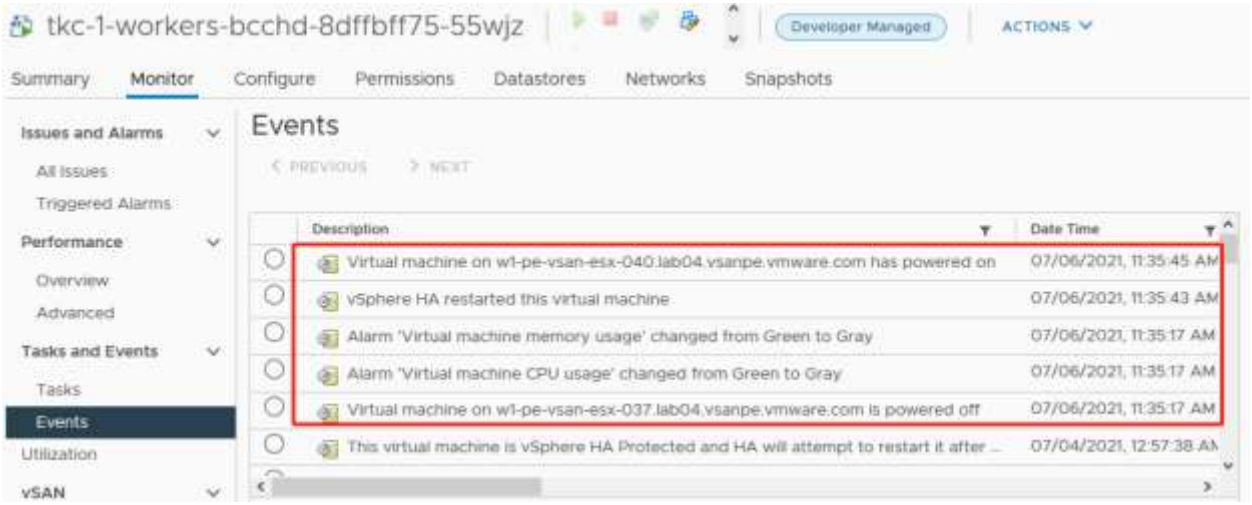


Figure 23. Tanzu Kubernetes Grid Worker's Events

NOTE: Since a separate disk is configured as one storage volume when provisioning a Tanzu Kubernetes cluster, an extra step of configuration is required to make sure the volume is properly mounted during the reboot of a VM.

- After the Tanzu Kubernetes cluster is provisioned, apply *a DaemonSet* to the Tanzu Kubernetes cluster, which would update the worker node VM with the proper mount information.
- After all the DaemonSet pods have updated the worker node, the DaemonSet can be deleted. In future, if a new worker node VM is added to the cluster, run the DaemonSet again to update the new node. The existing worker node VMs will not be updated again.
- Alternatively, this DaemonSet can be left active so it will automatically update any newly added worker node VMs.

With the above configuration, once the worker node VM restarts, either caused by a host failure or a VM failure, the impacted pods will be restarted on the same VM. For FLC pods, they typically take less than 3 minutes to be relaunched after VM restarts. During the failover, these pods will be temporarily unavailable. This means the ongoing FL training job could hang during this period, and users cannot view the job history or launch new jobs. After the pods restart, as discussed in the previous "**KubeFATE Component Failures**" part, the FLC can automatically resume working in most cases. To further achieve the minimum service interruption, as recommended in the Deploying a KubeFATE Instance section, many FLC components including Spark, HDFS, and Pulsar can be deployed separately with HA enabled.

Disk Failure

As discussed previously, KubeFATE supports configuring and provisioning persistent volumes for all the components it manages. These persistent volumes are backed by vSAN that provides storage availability for all the persistent volumes (vsan-default-storage-policy is the default storage class). It is validated that disk failure will not impact the KubeFATE cluster—training jobs can continue to run and finish without error; new jobs can be launched, and historical job data are preserved.

- In the component of FATE Flow, the logs, job metadata, and trained models are all saved in the backing persistent volume.
- For other components, KubeFATE supports storing key data for the MySQL, HDFS, and Pulsar deployments.

Sizing Guidelines

The following recommendations provide the best practices and sizing guidance to run KubeFATE on VMware Cloud Foundation.

VMware Cloud Foundation Infrastructure

Follow the general sizing guide: *Cloud Foundation Kubernetes Sizing Guide*

KubeFATE

Figure 24 shows the resource consumption type of KubeFATE components. For example, the FATE Flow service is CPU and memory intensive, we should allocate adequate CPU or memory resource to the worker node it runs on.

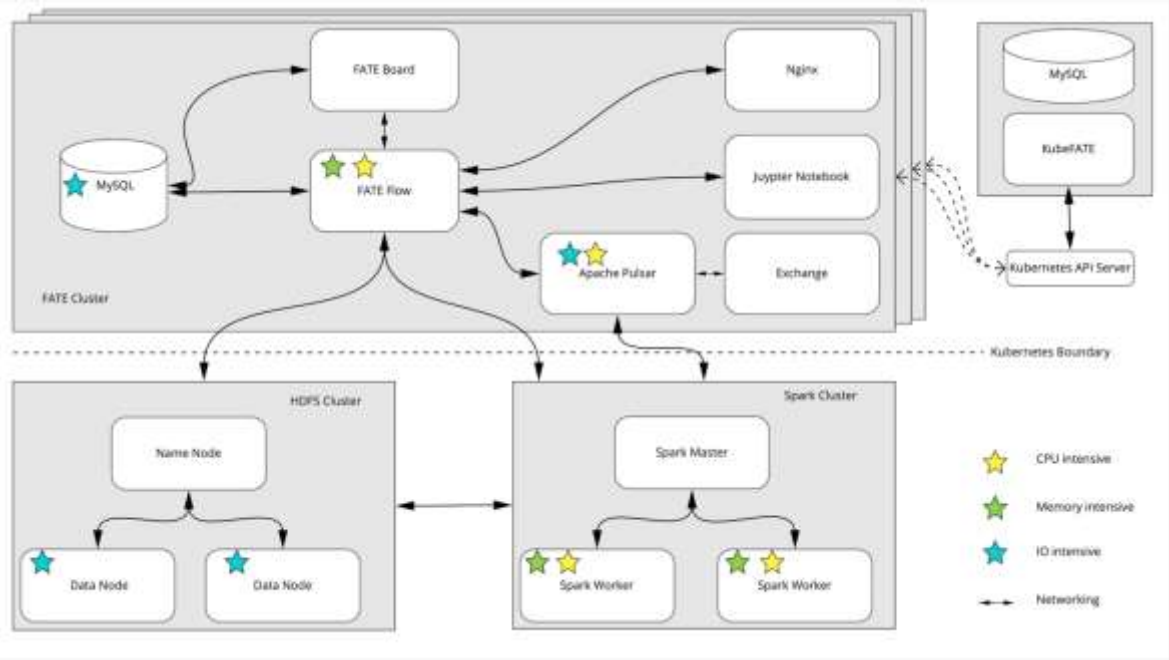


Figure 24. Service Detail of the KubeFATE Instance

The following parameters affect the overall sizing of compute and storage requirements:

Deployment with the internal Spark and HDFS service (default):

- **VM Class**: A virtual machine (VM) class is a request for resource reservations on the VM for processing power (CPU and memory). There are two class reservation types: guaranteed and best effort. The guaranteed class fully reserves its configured resources. The best effort class allows resources to be overcommitted. It is recommended that the VM class type of the Kubernetes control panel is best-effort-small or larger. For Kubernetes worker nodes, the VM class types of best-effort-2xlarge or larger are recommended. To avoid overcommitting resources, production workloads should use the guaranteed class type. Refer to *VM classes for Tanzu Kubernetes Cluster* for more details.
- **CPU**: The number of vCPUs associated per pod except for the Spark worker pod should be 4 vCPUs at minimum. For the neural network workload, more vCPU resources should be allocated to the "fateflow" pod based on workload size. At least 8 vCPUs should be allocated to the Spark worker pod. It is recommended to allocate all CPU resources of a Tanzu Kubernetes cluster worker node to the Spark worker pod, because more CPU resources will speed up the workload execution according to our baseline testing. For more details, refer to *Resource Usage of TKG Cluster.*
- **Memory**: A minimum of 8 GB memory is recommended for all pods. For the neural network workload, the "python" pod should be given more memory based on the workload size. At least 16 GB memory should be allocated to the Spark worker pod.
  Number of nodes: It is recommended to run only one Spark worker pod on a Tanzu Kubernetes Grid cluster worker node.
- **Storage size**: The persistent volume size of each pod should be 5GB at least. For the DataNode of HDFS and "python" pod, the storage size must be at least two times of the dataset size.
- **Network:** 10 Gb Ethernet or higher speed network is recommended.

Deployment without internal Spark and HDFS service:

Component Pod sizing is similar to the default deployment.

Use Cases

FL is considered as one of the most exciting technologies nowadays. Many industries and companies are beginning to incorporate FL into their work cycles. More use cases are surfaced when FL is getting more mature. These use cases can be categorized as the following two types of federated learning:

Vertical Federated Learning, also known as heterogeneous federated learning, applies to the cases that two data sets share the same sample ID space but differ in feature space. Vertical federated learning usually happens in collaboration between organizations from different industries.

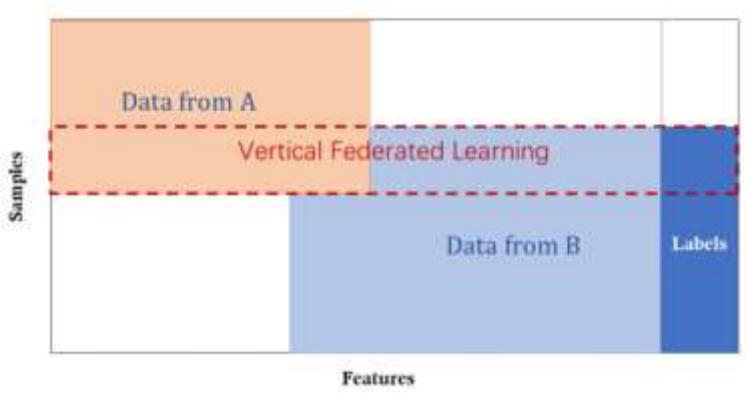


Figure 25. Vertical Federated Learning[4]

The typical use cases of vertical federated learning are:

- Auto insurance pricing: FL in the prediction of auto insurance pricing gathers the facts of consumer, vehicle, and behaviors from different data resources for modeling. The predictive pricing accuracy has been improved to over 90% while comparing to the traditional vehicle-based pricing strategy.
- Credit-risk management: The credit-risk management for small and micro enterprises (SME) is high cost and low accuracy due to the lack of enough credit records of SMEs. By using FL to establish a multisource data fusion mechanism to include transaction data, taxation, industrial and commercial data, and other SME data to assist financial institutions in obtaining data in more dimensions to enrich their feature systems. After applying the FL, there is a 12% AUC improvement compared with the traditional credit checks method with a much lower cost.
- Smart retail: To keep up with the business transformation strategies, retail enterprises can provide personalized product services in a legal and compliant manner to expand sales channels. In addition to improving the user experience, the solutions lay a foundation for precision marketing. The FL smart labor allocation system provides a complete platform that covers the entire labor allocation process and solves the problem of data asymmetry between the supply and demand of labor. In future, the FL models can further diversify into industries such as manufacturing, warehousing and logistics, import/export, and other vertical areas.

Horizontal Federated Learning, also called homogenous federated learning, is the scenario that data sets share the same feature space but are different in samples. In this case, the organizations are in the same type of industry. The data can be images, video files, audio files, and other unstructured data, which share the same features.

[4] This Vertical Federated Learning concept and diagram is quoted from the *Federated Machine Learning: Concept and Applications*.

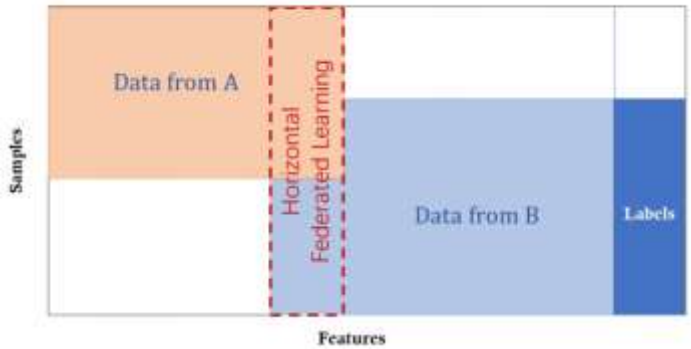


Figure 26. Horizontal Federated Learning[5]

The typical use cases of horizontal federated learning are:

- Smart security: By using FL and multi-community data to build an interconnected and intercommunicated security model, which establishes a smart security network with overlapping dimensions. FL smart security ensures data privacy while integrating user traffic and other data across multiple communities. The solution delivers all-weather monitoring of the public and secure areas, early prediction, timely detection, along with early warning and post-incident tracing to enhance community security.
- Healthcare assistant: FL smart medical care empowers the treatment in clinical diagnosis and other subfields while protecting patient privacy. The applications of federated learning for smart medical care scenarios develop high-quality medical resources shared by regions with fewer resources and improve the capacity and quality of medical services.
- Smart advertising: FL smart advertising enables the enhanced integration of multiparty data to develop user insight and targeting strategies and preserve the privacy of the individual user. The differential privacy technology in federated learning obfuscates data so that other parties can only view a generalized summary without identifying any individual in the data. FL advertising technology reduces delivery costs, improves advertiser ROI, and enables the utilization of funds for product innovation and R&D. For ad providers, it improves user click-through rate (CTR) and conversion rate and optimizes link efficiency to achieve mutually beneficial cooperation between all parties.
- Autonomous driving: Using horizontal federated learning to integrate the data from the cameras, ultrasonic sensors, radars (mmWave and LIDAR), and other devices of different vehicles accelerates the deployment of scenario data and improves the model robustness. Horizontal FL driving technologies accelerate perception training while protecting the privacy of drivers and passengers. In future, vertical FL will integrate with IoT, CVIS, 5G, and other new technologies to constitute a smart traffic ecosystem that is efficient, secure, and low-cost.

Conclusion

VMware Cloud Foundation with Tanzu automates infrastructure provisioning and scaling so that developers can focus on building and deploying apps while infrastructure teams become more strategic, maintaining centralized visibility and control of their global cloud infrastructure and operations. Developers consume cloud resources such as Kubernetes clusters, disks, and networks using familiar Kubernetes CLI and API tools, while the admins can manage systems at scale through vCenter Server. It suits the demands of modernized federated learning workloads.

KubeFATE on VMware Cloud Foundation with Tanzu simplifies the deployment and management of federated learning systems and workloads. KubeFATE enables VMware's partners and customers to provision and manage the industrial grade FL clusters on demand and to run FL workload according to their business needs.

[5] This Horizontal Federated Learning concept and diagram is quoted from the *Federated Machine Learning: Concept and Applications*.

Reference Architecture

- *VMware Cloud Foundation*
- *VMware vSphere*
- *VMware vSAN*
- *VMware NSX Data Center*
- *KubeFATE*
- *White Paper on Federated Learning V2.0*
- *Yang, Qiang & Liu, Yang & Chen, Tianjian & Tong, Yongxin. (2019). Federated Machine Learning: Concept and Applications. ACM Transactions on Intelligent Systems and Technology. 10. 1-19. 10.1145/3298981.*

About the Author

Ting Yin, Senior Solutions Architect in the Solutions Architecture team of the Cloud Infrastructure Big Group, and Fangchi Wang, Senior Member of Technical Staff in Office of the CTO, co-authored the original content of this reference architecture.

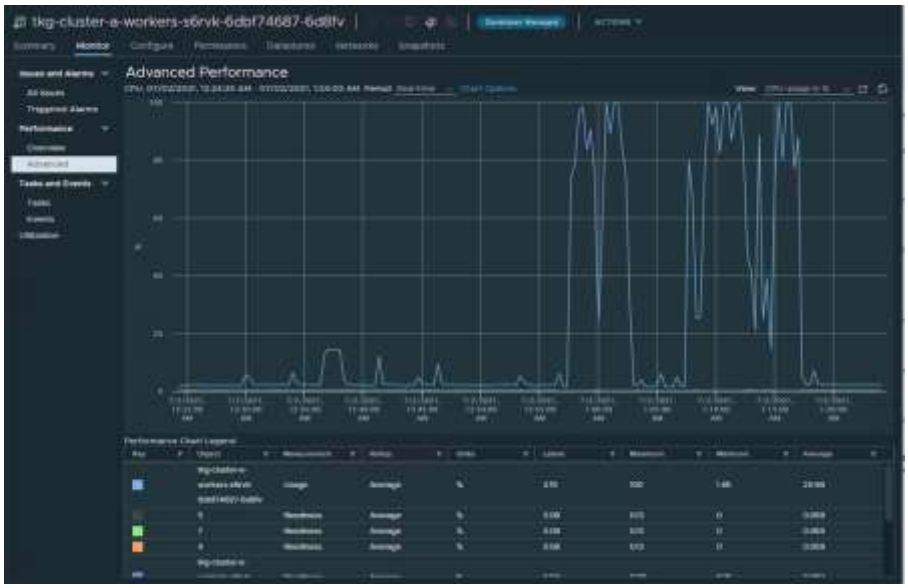
The following people also contributed to this paper:

- Layne Peng, Staff Engineer 2 in Office of the CTO in VMware
- Jiahao Chen, Member of Technical Staff in Office of the CTO in VMware

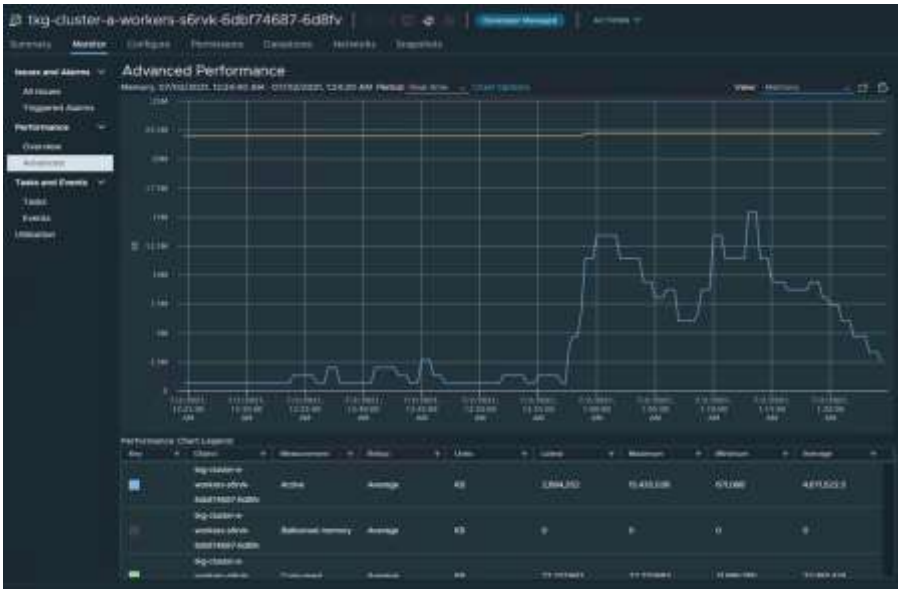
Appendix

Figures – Resource usage of Tanzu Kubernetes Grid cluster

- CPU and memory usage of a host running Spark worker

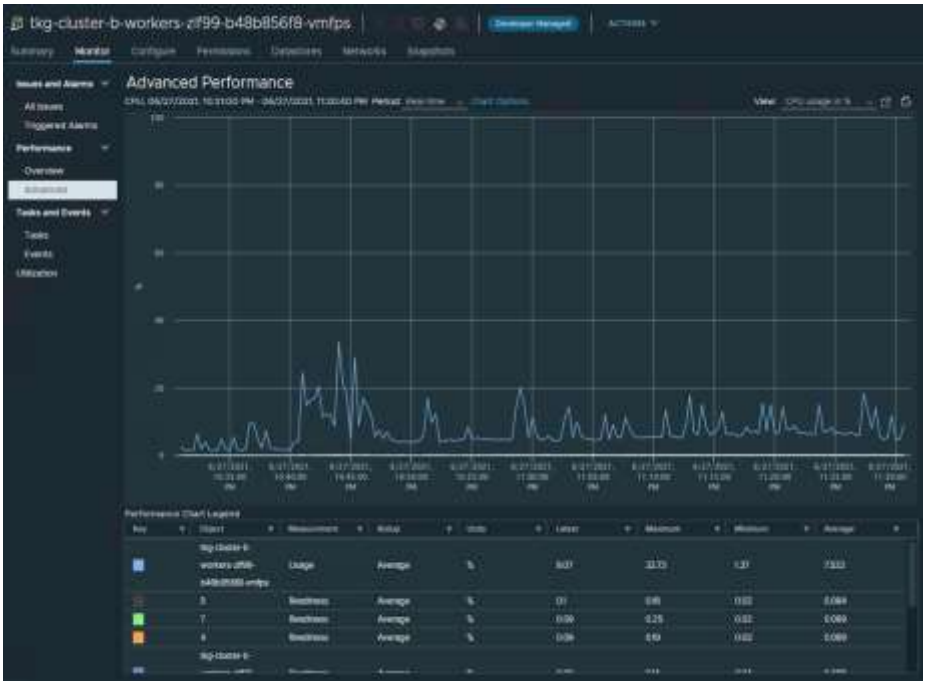


CPU usage

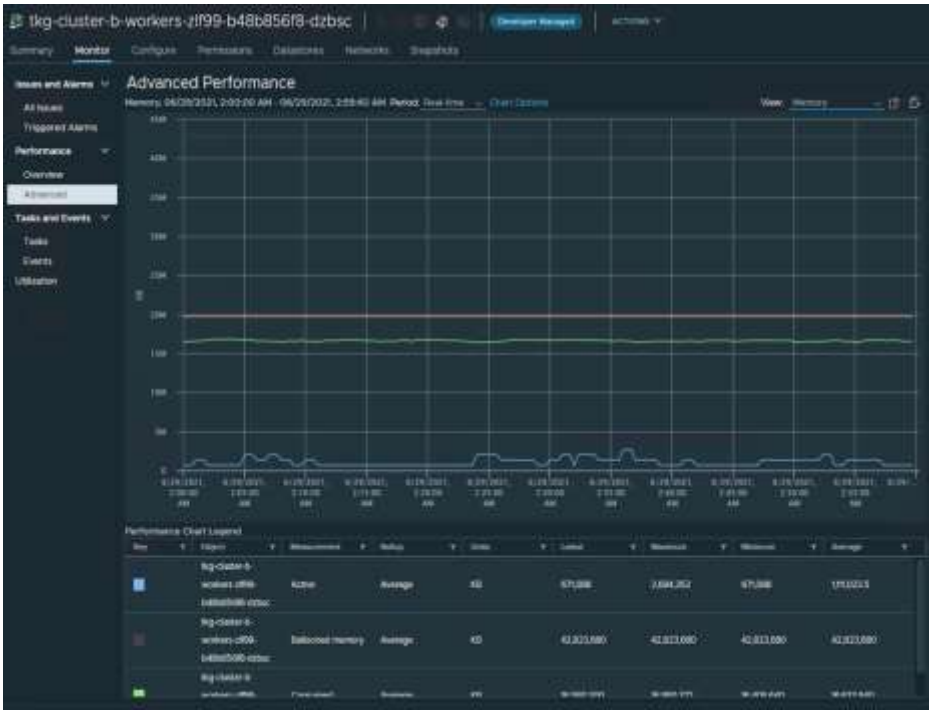


Memory usage

- CPU and Memory usage of a host running non-spark components.



CPU usage



Memory usage

Components Table

Component Name	Component's Role
FATE Flow Service	The FATE Flow service is the entry point of the FATE cluster, accepting requests from a user. It also schedules jobs and manages job status among different FATE clusters.
MySQL	In a FATE cluster, the MySQL service is used to persist metadata for jobs.
Nginx	In a FATE cluster, the Nginx service is used to transfer controlling messages between FATE clusters.
Pulsar	In a FATE cluster, the Pulsar service is used to transfer encrypted gradients, model weights between FATE clusters during training.
Juypter Notebook	The Jupyter Notebook is used to build and run jobs for a user.
FATE Board	The FATE Board is used to visualize the status of the federated learning workload.
Spark Cluster	In a FATE cluster, the Spark is used to run the computing workload of a federated learning job.
HDFS Cluster	In a FATE cluster, the HDFS stores the training dataset and intermediated results.

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 www.vmware.com.
 Item No: vmw-wp-tech-temp-word-102-proof 5/19