VMware Kubernetes Integration with Bitfusion
(Updated September 2021)

Table of Contents

Table of Figures

1 Introduction

Modern data scientists need diverse sets of tools and infrastructure in order to deliver valuable insights to their organizations. IT infrastructure has struggled to meet the unique needs of data scientists. Because of the increasing importance of data science to business, IT infrastructure needs to evolve rapidly to meet their complex requirements. Infrastructure needs to evolve and bring agility and flexibility to make the data scientist productive through the use of modern developer platforms like Kubernetes. IT should also be able to provide access to specialized HW like GPUs and other accelerators to help data scientists be productive.

1.1 Kubernetes evolving as the leading development platform

Enterprise are shifting their focus from infrastructure to application development. It's imperative that companies increase developer productivity, shorten the path to production, and accelerate the cadence of new features and services. Easy access to resources is crucial for developer success, but there are many additional reasons that developers like Kubernetes. These include its inherent resilience, repeatability, flexibility, and visibility. Because Kubernetes is flexible rather than prescriptive, it adapts to a wide range of developer needs. A subset of these developer community are data scientists & data engineers that perform machine learning as AI takes on a significant role in today's business.

1.2 GPUs for Machine Learning

With the impending end to Moore's law, the spark that is fueling the current revolution in deep learning is having enough compute horsepower to train neural-network based models in a reasonable amount of time. The needed compute horsepower is derived largely from GPUs, which NVIDIA began optimizing for deep learning since 2012. A lot of the machine learning and AI related work relies on processing large blocks of data, which makes GPUs a good fit for ML tasks. Most of the machine learning frameworks have in built support for GPUs. There is a need to provide the capabilities needed by data scientists such as GPU access from Kubernetes environments.

1.3 vSphere 7 brings together Kubernetes and GPU sharing capabilities with Bitfusion

With the release of vSphere 7 many new features such as Kubernetes and Bitfusion were introduced. Kubernetes is the preferred platform for developers and is now fully integrated with vSphere as first class citizens along with virtual machines. vSphere Bitfusion provides the ability to share NVIDIA GPUs over the network. Modern applications quite often need access to GPUs and their massive compute capabilities for timely and efficient processing. There is a significant need for Kubernetes based developer environments to access GPUs. A combination of the VMware Kubernetes platform and vSphere Bitfusion for GPU sharing over the network can help meet the infrastructure needs of modern data scientists. This solution show cases the integration of VMware Kubernetes platforms like TKG and TKGI with vSphere Bitfusion.

2 Solution Components

2.1 VMware vSphere

VMware vSphere extends virtualization to storage and network services and adds automated, policy-based provisioning and management. As the foundation for VMware's complete SDDC platform, vSphere is the starting point for your virtualization infrastructure providing wide ranging support for the latest hardware and accelerators used for Machine Learning. VMware vSphere with Kubernetes (formerly called "Project Pacific") empowers IT Operators and Application Developers to accelerate innovation by converging Kubernetes, containers and VMs into VMware's vSphere platform. VMware has leveraged Kubernetes to rearchitect vSphere and extend its capabilities to all modern and traditional applications.

vSphere with Kubernetes innovations:

- Unite vSphere and Kubernetes by embedding Kubernetes into the control plane of vSphere, unifying control of compute, network and storage resources. Converging VMs and containers using the new native container service that creates high performing, secure and easy to consume containers
- App-focused management with App level control for applying policies, quota and role-based access to Developers. Ability apply vSphere features (HA, vMotion, DRS) at the app level and to the containers. Unified visibility in vCenter for Kubernetes clusters, containers and existing VMs
- Enable Dev & IT Ops collaboration. Developers use Kubernetes APIs to access the datacenter infrastructure (compute, storage and networking). IT operators use vSphere tools to deliver Kubernetes clusters to developers. Consistent view between Dev and Ops via Kubernetes constructs in vSphere
- Enterprises can expect to get improved economics due to the convergence of vSphere, Kubernetes and containers. Operators get control at scale by focusing on managing apps versus managing VMs. Developers & operators collaborate to gain increased velocity due to familiar tools (vSphere tools for Operators and Kubernetes service for Developers).

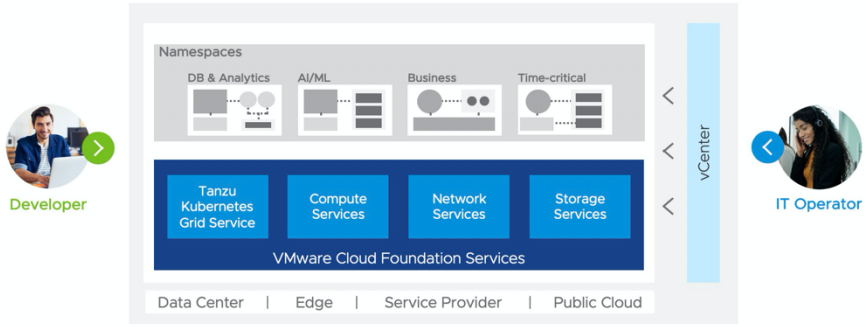

Figure 1: vSphere 7 with Kubernetes innovations

2.2 VSphere Bitfusion

VSphere Bitfusion extends the power of VMware vSphere's virtualization technology to GPUs. VSphere Bitfusion helps enterprises disaggregate the GPU compute and dynamically attach GPUs anywhere in the datacenter. Support more users in test and development phase. VSphere Bitfusion supports CUDA frameworks and demonstrated virtualization and remote attach for all hardware. GPUs are attached based on CUDA calls at run-time, maximizing utilization of GPU servers anywhere in the network.

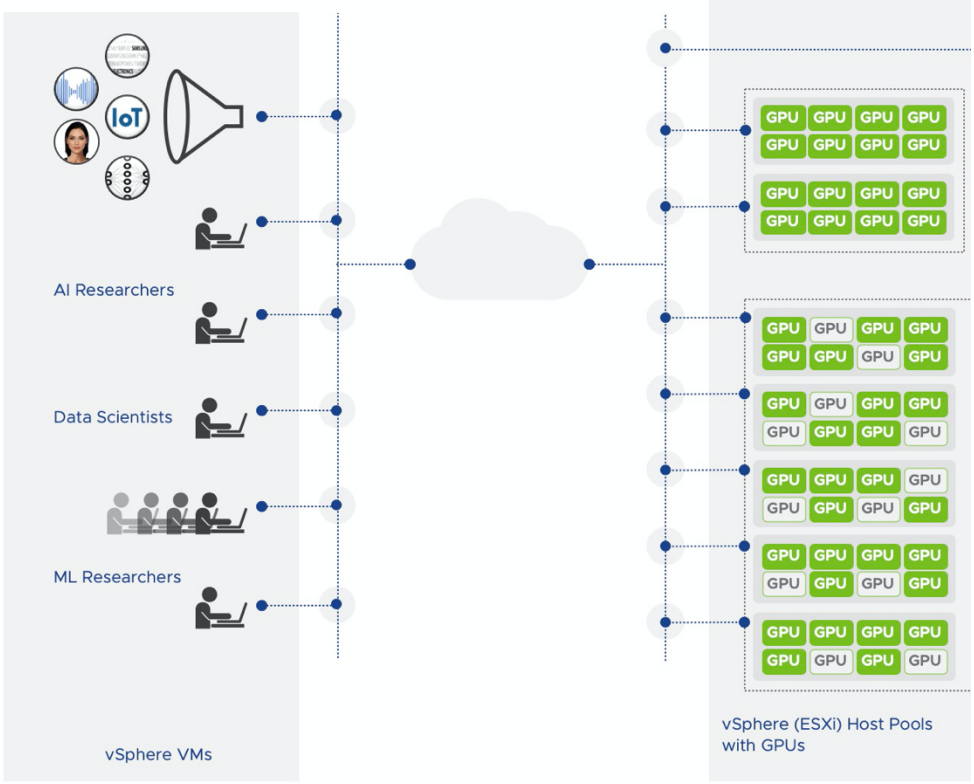

Figure 2: Remote GPU access with vSphere Bitfusion

2.3 VMware Tanzu Kubernetes Grid (TKG)

VMware Tanzu Kubernetes Grid is a container services solution that enables Kubernetes to operate in multi-cloud environments. VMware TKG simplifies the deployment and management of Kubernetes clusters with Day 1 and Day 2 operations support. VMware TKG manages container deployment from the application layer all the way to the infrastructure layer, according to the requirements VMware TKG supports high availability, autoscaling, health-checks, and self-repairing of underlying VMs and rolling upgrades for the Kubernetes clusters.

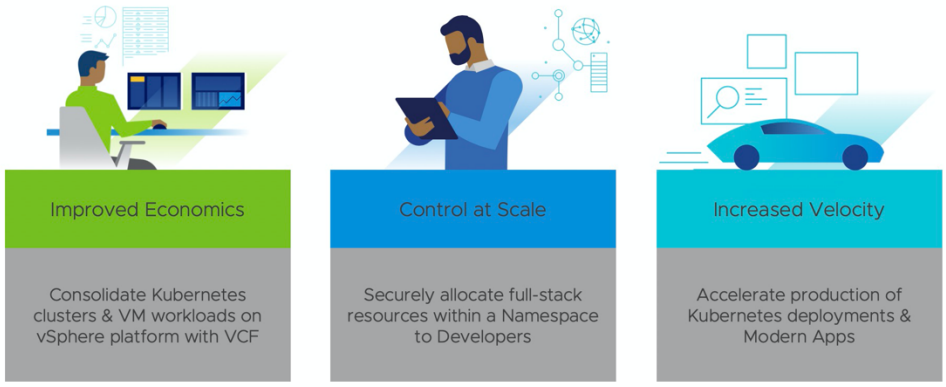

Figure 3: Tanzu Kubernetes Grid Benefits

2.4 Tanzu Kubernetes Grid Integrated (TKGI formerly VMware Enterprise PKS)

TKGI is a purpose-built container solution to operationalize Kubernetes for multi-cloud enterprises and service providers. It significantly simplifies the deployment and management of Kubernetes clusters with day 1 and day 2 operations support. With hardened production-grade capabilities, TKGI takes care of your container deployments from the application layer all the way to the infrastructure layer.

TKGI builds on Kubernetes, BOSH, VMware NSX-T, and Project Harbor to form a production-grade, highly-available container runtime that operates on vSphere and public clouds. With built-in intelligence and integration, TKGI ties all these open source and commercial modules together, delivering a simple-to-use product for customers, ensuring the customers have the most efficient Kubernetes deployment and management experience possible.

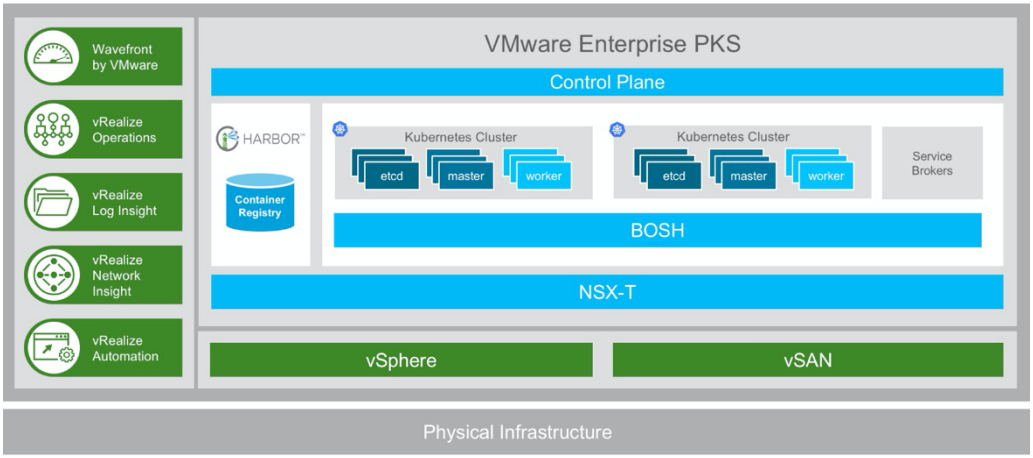

Figure 4: Tanzu Kubernetes Grid Integrated (TKGI)

2.5 VMware NSX-T

VMware NSX-T provides an agile software-defined infrastructure to build cloud-native application environments. NSX-T is focused on providing networking, security, automation, and operational simplicity for emerging application frameworks and architectures that have heterogeneous endpoint environments and technology stacks. NSX-T supports cloud-native applications, bare metal workloads, multi-hypervisor environments, public clouds, and multiple clouds.

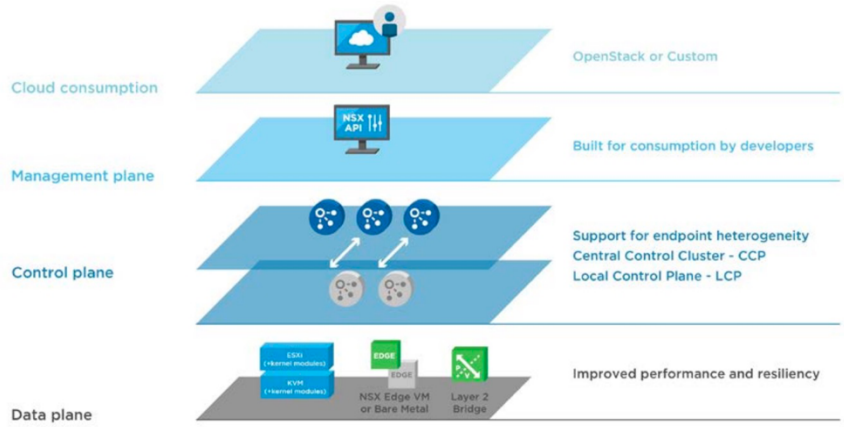

Figure 5: NSX-T Components

NSX-T is designed for management, operation, and consumption by development organizations. NSX-T Data Center allows IT and development teams to select the technologies best suited for their applications. NSX-T 3.x is fully integrated with TKG/TKGI and provides seamless networking for the Kubernetes workloads.

2.6 Bitfusion with Kubernetes integration

The Bitfusion Device Plugin provides a more convenient way to use Bitfusion in Kubernetes clusters. Users do not need to rebuild the workload image. They can simply configure the original image into POD and the Device Plugin will be responsible for importing the Bitfusion dependencies. It also supports the use of K8S native quota limiting function to limit the quota management of Bitfusion GPU resources.

3 Solution Setup

The logical schematic below shows the components of the solution as validated.

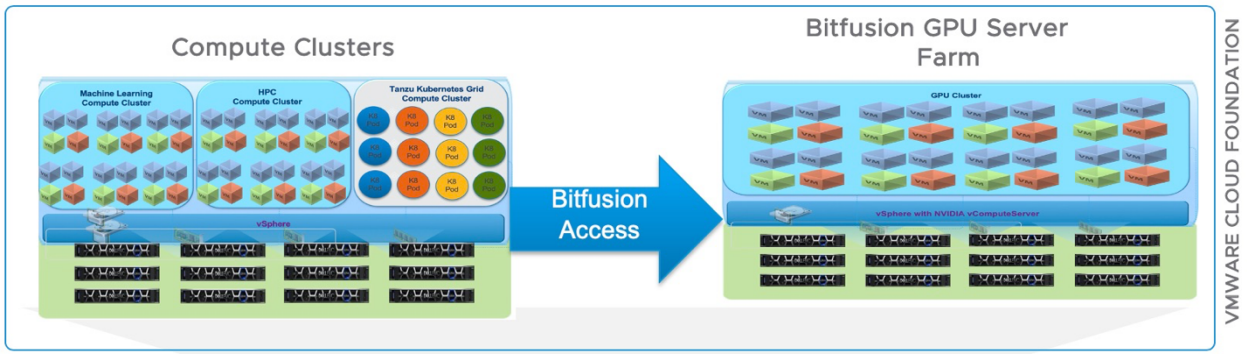

Figure 6: Logical schematic of solution

All GPU resources are consolidated in a GPU Cluster. The Bitfusion servers are sourced from this cluster where they are direct attached to the GPU in pass-through mode. Kubernetes Clusters that include different variants of VMware Kubernetes such as Supervisor, TKG Guest and TKGI represented as VMware compute clusters are connected via high-speed datacenter networking to the Bitfusion server farm. Egress network traffic from the Kubernetes clusters is routed by NSX-T to the Bitfusion GPU server farm to establish client-server connectivity to the remote GPUs. The GPU cluster and the Compute clusters are running vSphere 7 and managed by vCenter 7.

3.1 Enabling Bitfusion access:

The Bitfusion plugin is installed in the vCenter and can be used to manage and monitor Bitfusion servers and access to clients. Once Bitfusion is installed and integrated with vCenter and licensed, it appears as a plugin for vCenter. Virtual machines associated with the vCenter can now be enabled for Bitfusion Client access. For Kubernetes environments, the Bitfusion client needs to be embedded inside the container/pod. An example Linux virtual machine is used to obtain these client files. The Linux machine is enabled for Bitfusion as shown in the process below.

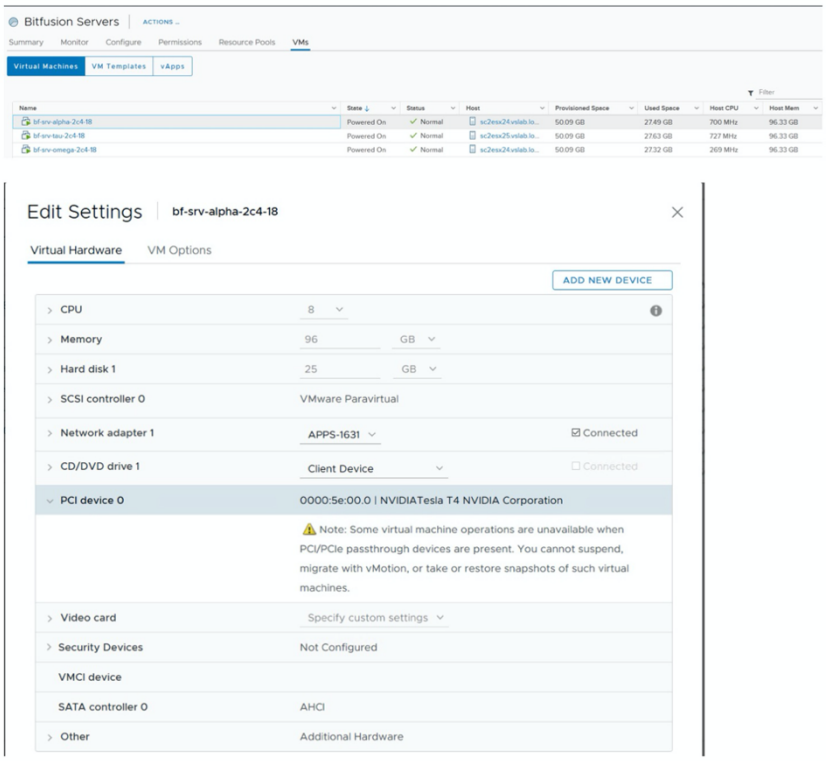

Figure 7: Bitfusion Servers with pass-through GPU

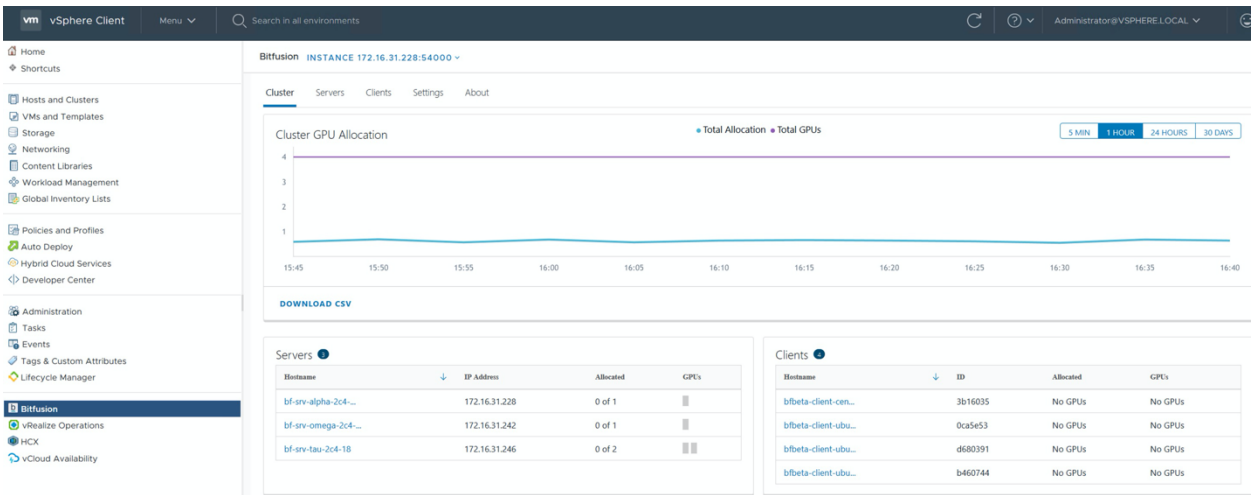

Figure 8: Bitfusion plugin console

Once enabled the files are extracted from the Linux machine and used to build the container as a Bitfusion client as shown in Appendix C. The process of enabling a virtual machine for Bitfusion client access is shown in Figure below.

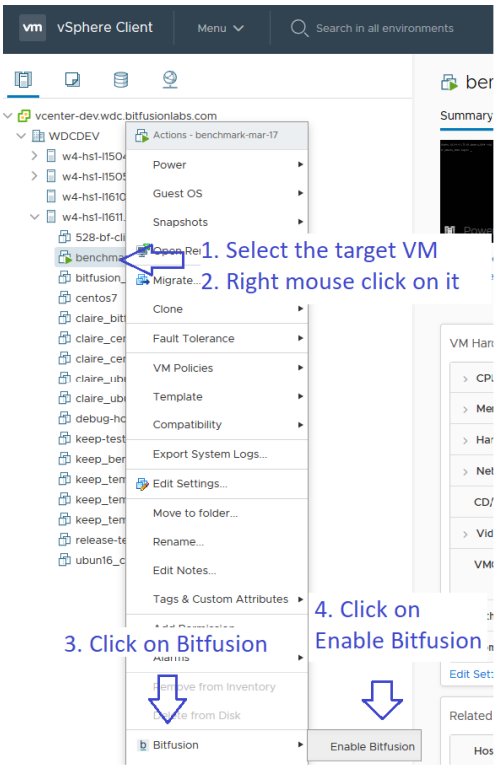

Figure 9: Enabling Bitfusion client for a virtual machine

3.2 Customizations to enable Bitfusion in containers

Bitfusion client components are installed in the container itself so that it works seamlessly after the pod is deployed. No changes are needed to the Kubernetes worker nodes.

Here are the steps to enable Bitfusion client to work inside a container.

Once you identify a client Linux machine for Bitfusion Client deployment,

- Enable Bitfusion on the client Linux machine
- Install Bitfusion client for that operating system (point to link in documentation)

Confirm that it works fine by running the command:

```
Bitfusion list_gpus
```

Now this Bitfusion client machine has the necessary authorization files for Bitfusion access.
Copy the following files to a staging directory, say, Bitfusion-files:

```
cp ~/.Bitfusion/client.yaml Bitfusion-files/
cp /etc/Bitfusion/servers.conf Bitfusion-files/
cp /etc/Bitfusion/tls/ca.crt Bitfusion-files/
```

Additionally download and copy the appropriate installer .deb or .rpm file to the same staging directory:
For example, for Ubuntu OS, we will use a .deb file as below:

```
cp Bitfusion-client-ubuntu1604_2.0.0_amd64.deb Bitfusion-files/
```

When you create a docker image for a container, you need to install Bitfusion client as well as copy the authorization files to the appropriate directories in the container image. Additionally, install open-vm-tools. In the example shown we are copying files to appropriate directories for the user root.
For this, add the following to your Docker file:

```
#-------------------------------------------------------------------
# Copy Bitfusion client related files
#-------------------------------------------------------------------
# Copy Bitfusion files
RUN mkdir -p /root/.Bitfusion
COPY ./Bitfusion-files/client.yaml /root/.Bitfusion/client.yaml
COPY ./Bitfusion-files/servers.conf /etc/Bitfusion/servers.conf
RUN mkdir -p /etc/Bitfusion/tls
COPY ./Bitfusion-files/ca.crt /etc/Bitfusion/tls/ca.crt
#-------------------------------------------------------------------
# Update package list
# Install Bitfusion. Use deb file for Ubuntu16.04
#-------------------------------------------------------------------
#  Set initial working directory
RUN mkdir -p /workspace/Bitfusion
WORKDIR /workspace/Bitfusion
# Copy Release version of Bitfusion client
COPY ./Bitfusion-files/Bitfusion-client-ubuntu1604_2.0.0_amd64.deb .
RUN apt-get update \
      && apt-get install -y ./Bitfusion-client-ubuntu1604_2.0.0_amd64.deb \
      && apt-get install -y open-vm-tools \
      && \
      rm -rf /var/lib/apt/lists/
#-------------------------------------------------------------------
```

These steps ensure that the Bitfusion components are baked into the container image itself. Appendix A and C show the docker file used to create Tensorflow and Pytorch containers with Bitfusion client embedded in it.

3.3 Kubernetes Setup:

Kubernetes Workload management is a new capability that helps manage Kubernetes workloads in vSphere 7. All Kubernetes clusters and pods are visualized and managed in vCenter using workload management. The Kubernetes clusters used in the solution were deployed and is shown below.

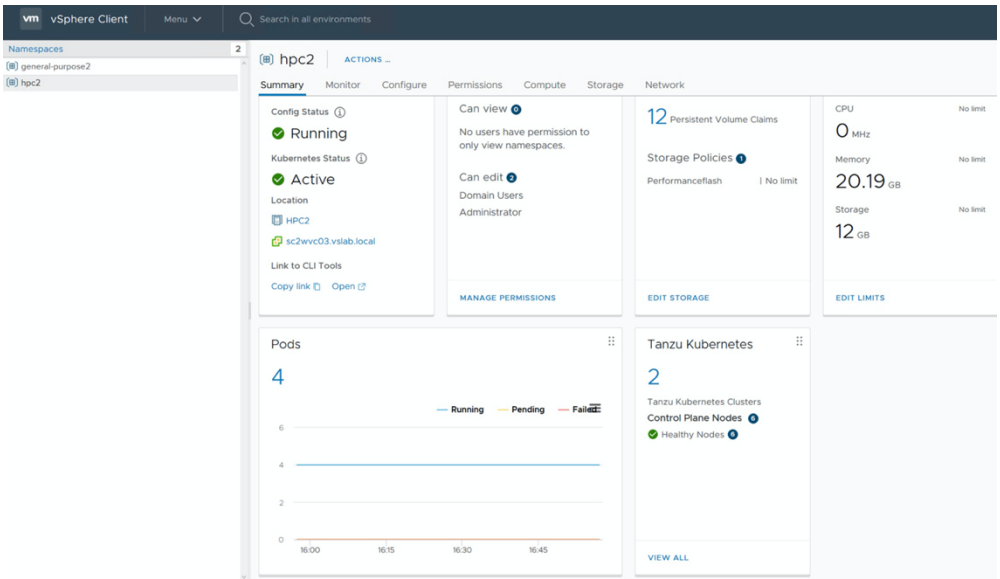

Figure 10: Kubernetes namespaces in vCenter Workload management

The hpc2 namespace represents the Kubernetes namespace running in the vSphere Cluster HPC2. All aspects of the Kubernetes namespace such as control plane nodes, TKG guest clusters, storage policies, etc. are seen. The figure below shows the representation of the namespace, its clusters including control and worker nodes in the "Hosts and Clusters" vCenter view.

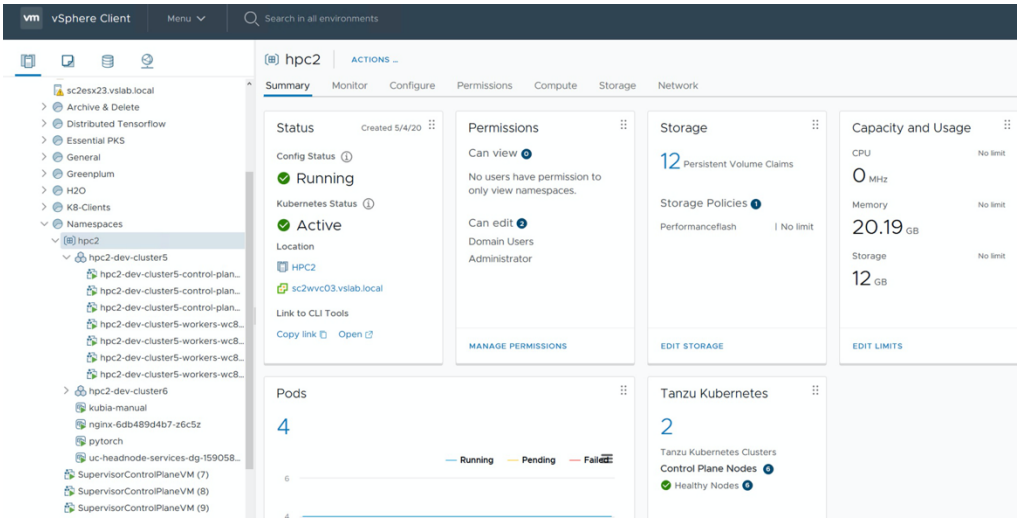

Figure 11: Kubernetes components as seen in "Hosts and Clusters" view

An example guest cluster hpc2-dev-cluster5 is shown with its control and worker nodes. The PyTorch virtual machine is a podVM running in the supervisor cluster for hpc2. The supervisor cluster control nodes are also shown at the bottom of the list. Kubernetes command line is used to login to a guest cluster and by setting the context as shown below.

```
[root@sc2k8cl4 tkg-bf]# kubectl vsphere login --server 172.30.12.1 --vsphere-username ssgash@vslab.local --insecure-skip-tls-v
erify --tanzu-kubernetes-cluster-name hpc2-dev-cluster5

Password:
WARN[0005] Tanzu Kubernetes cluster login: no namespace given, name (hpc2-dev-cluster5) may be ambiguous
Logged in successfully.

You have access to the following contexts:
   172.30.12.1
   general-purpose2
   general-purpose2-172.30.11.1
   hpc2
   hpc2-172.30.12.1
   hpc2-dev-cluster5

If the context you wish to use is not in this list, you may need to try
logging in again later, or contact your cluster administrator.

To change context, use `kubectl config use-context <workload name>`
[root@sc2k8cl4 tkg-bf]# kubectl config use-context hpc2-dev-cluster5
Switched to context "hpc2-dev-cluster5".
[root@sc2k8cl4 tkg-bf]# kubectl get nodes
NAME                                          STATUS   ROLES       AGE   VERSION
hpc2-dev-cluster5-control-plane-6mrpz         Ready    master      48d   v1.16.8+vmware.1
hpc2-dev-cluster5-control-plane-lc79b         Ready    master      48d   v1.16.8+vmware.1
hpc2-dev-cluster5-control-plane-mtfq4         Ready    master      48d   v1.16.8+vmware.1
hpc2-dev-cluster5-workers-wc8sg-656f68bc4c-6tsbp   Ready    uc-master   48d   v1.16.8+vmware.1
hpc2-dev-cluster5-workers-wc8sg-656f68bc4c-b25nr   Ready    uc-worker   48d   v1.16.8+vmware.1
hpc2-dev-cluster5-workers-wc8sg-656f68bc4c-j2qj6   Ready    uc-worker   48d   v1.16.8+vmware.1
hpc2-dev-cluster5-workers-wc8sg-656f68bc4c-jkzb7   Ready    uc-worker   48d   v1.16.8+vmware.1
[root@sc2k8cl4 tkg-bf]#
```

Figure 12: Kubernetes login and node listing for a TKG guest cluster

Similarly, one can directly login to the Supervisor cluster to run native pods that run directly on the ESXi hypervisor.

```
[root@sc2k8cl4 tkg-bf]# kubectl vsphere login --server 172.30.12.1 --vsphere-username administrator@vsphere.local --insecure-s
kip-tls-verify

Password:
Logged in successfully.

You have access to the following contexts:
   172.30.12.1
   general-purpose2
   hpc2

If the context you wish to use is not in this list, you may need to try
logging in again later, or contact your cluster administrator.

To change context, use `kubectl config use-context <workload name>`
[root@sc2k8cl4 tkg-bf]# kubectl config use-context hpc2
Switched to context "hpc2".
[root@sc2k8cl4 tkg-bf]# kubectl get nodes
NAME                                STATUS   ROLES    AGE   VERSION
420013823c5b73283d65e88e1eb00e28   Ready    master   46d   v1.16.7-2+bfe512e5ddaaaa
42002935e11918c16ca21f7e51d8d34e   Ready    master   46d   v1.16.7-2+bfe512e5ddaaaa
420093bf6ccec1eafca4a3238876504b   Ready    master   46d   v1.16.7-2+bfe512e5ddaaaa
sc2esx22                            Ready    agent    46d   v1.16.7-sph-4d52cd1
sc2esx23                            Ready    agent    46d   v1.16.7-sph-4d52cd1
[root@sc2k8cl4 tkg-bf]#
```

Figure 13: Kubernetes login and node listing for a supervisor cluster

TKGI formerly known as Enterprise PKS is the other prevalent Kubernetes platform used by customers. A TKGI environment was setup in a dedicated VMware cluster. All Enterprise PKS management components were installed followed by a Kubernetes cluster as shown below.

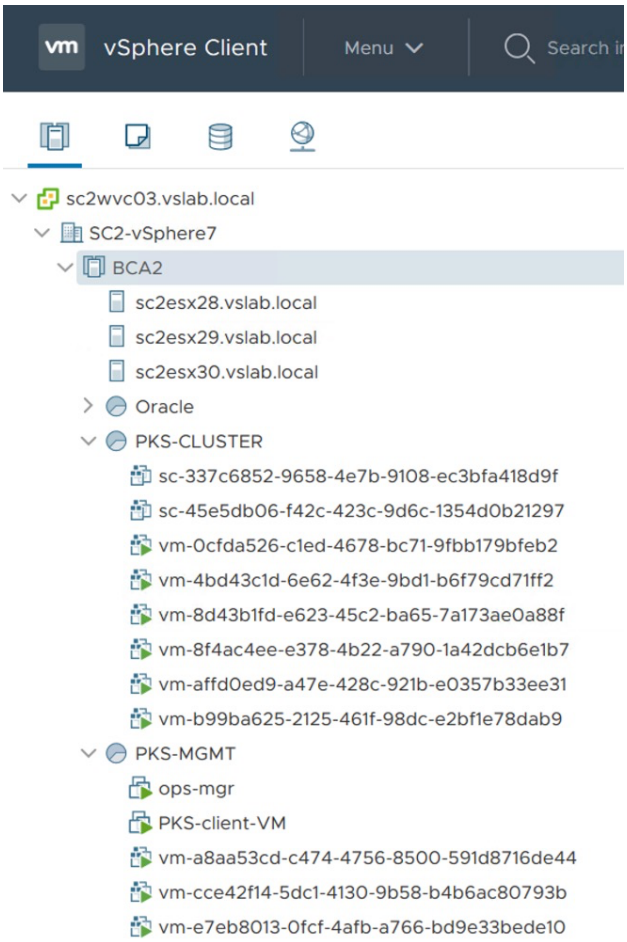

Figure 14: Kubernetes login and node listing for a supervisor cluster

Kubernetes command line is used to login to a PKS cluster as shown below. The Linux client has preloaded configs for the PKS cluster to enable the login process,

```
API Endpoint: pks.vslab.local
User: guillierf
Login successful.

root@pks-client-vm:/gpu_data/mlperf-data/tkg-bf# pks get-credentials pks-cluster-1

Fetching credentials for cluster pks-cluster-1.
Context set for cluster pks-cluster-1.

You can now switch between clusters by using:
$kubectl config use-context <cluster-name>
root@pks-client-vm:/gpu_data/mlperf-data/tkg-bf# kubectl get nodes
NAME                                   STATUS   ROLES    AGE   VERSION
7cf2398b-784b-4883-9b10-62d2c11580bb   Ready    <none>   15d   v1.17.5+vmware.1
95a46d7c-ab84-46b0-9c92-a435a4c5912a   Ready    <none>   15d   v1.17.5+vmware.1
b96d8047-0ad4-4543-99c7-312a674b98d2   Ready    <none>   15d   v1.17.5+vmware.1
root@pks-client-vm:/gpu_data/mlperf-data/tkg-bf#
```

Figure 15: Kubernetes login and node listing for TKGI cluster

3.4 Bitfusion with Kubernetes integration Setup:

Refer to the following link to install the Bitfusion Device Plugin:
https://github.com/vmware/Bitfusion-with-kubernetes-integration/tree/main/Bitfusion_device_plugin

4 Validation of the Solution:

There are three different variants of VMware based Kubernetes that were tested in this solution to confirm interoperability with Bitfusion. To validate the solution containers with TensorFlow and PyTorch with embedded Bitfusion components were used. The three different VMware Kubernetes variants were then tested by deploying these pods in their clusters and running TensorFlow and PyTorch. The customization of the pods with Bitfusion was done by adding the section shown below to the docker file. YAML files used for TensorFlow and PyTorch Kubernetes pods are shown in Appendix B & D.

```
# Copy bitfusion files
RUN mkdir -p /root/.bitfusion
COPY ./bitfusion-files/client.yaml /root/.bitfusion/client.yaml
COPY ./bitfusion-files/servers.conf /etc/bitfusion/servers.conf
RUN mkdir -p /etc/bitfusion/tls
COPY ./bitfusion-files/ca.crt /etc/bitfusion/tls/ca.crt
COPY ./bitfusion-files/nvidia-smi /workspace/bitfusion/batch-scripts/nvidia-smi


#-------------------------------------------------------------------------------
# Update package list
# Install Bitfusion. Use deb file for Ubuntu16.04
# resides in mounted directory, /pkgs
#-------------------------------------------------------------------------------
# Copy Release version of bitfusion client
COPY ./bitfusion-files/bitfusion-client-ubuntu1604_2.0.0v12nstaging-876_amd64.deb .
RUN apt-get update \
      && apt-get install -y ./bitfusion-client-ubuntu1604_2.0.0v12nstaging-876_amd64.deb \
      && apt-get install -y open-vm-tools \
      && \
      rm -rf /var/lib/apt/lists/


#-------------------------------------------------------------------------------
```

Figure 16: Embedding of Bitfusion files into the docker container

4.1 Tanzu Kubernetes Grid

Tanzu Kubernetes Grid (TKG) provides a consistent Kubernetes experience across clouds. With it, customers are able to rapidly provision and manage Kubernetes clusters in any and all locations they need Kubernetes-based workloads to run (both vSphere-based and non-vSphere-based). The goal of TKG is deliver a consistent experience with Kubernetes, irrespective of the underlying infrastructure. However, when TKG runs on vSphere, we are able to leverage all the innovations we've created with vSphere 7 with Kubernetes to offer a better experience for customers. Shown below is a successful TensorFlow run leveraging Bitfusion in a TKG cluster that was run as part of the validation.

```
[root@sc2k8cl4 tkg-bf]# kubectl config get-contexts
CURRENT   NAME                CLUSTER       AUTHINFO                                  NAMESPACE
          172.30.11.1         172.30.11.1   wcp:172.30.11.1:ssgash@vslab.local
          general-purpose2    172.30.11.1   wcp:172.30.11.1:ssgash@vslab.local        general-purpose2
*         gp2-dev-cluster3    172.30.11.4   wcp:172.30.11.4:ssgash@vslab.local
          hpc2                172.30.12.1   wcp:172.30.12.1:ssgash@vslab.local        hpc2
[root@sc2k8cl4 tkg-bf]# kubectl create -f tf_tmpdisk.yaml
pod/tf created
[root@sc2k8cl4 tkg-bf]# kubectl get pods
NAME   READY   STATUS    RESTARTS   AGE
tf     1/1     Running   0          68s
[root@sc2k8cl4 tkg-bf]# kubectl exec -it tf -- /bin/bash
root@tf:/workspace/dlbs# /workspace/dlbs/run_tensorflow.sh
```

```
 ["alexnet"]
WARNING:root:Module 'numpy' cannot be imported, some system information will not be available
WARNING:root:Module 'pandas' cannot be imported, some system information will not be available
WARNING:root:Module 'matplotlib' cannot be imported, some system information will not be available
INFO:root:Plan was built with 1 experiments
Plan was built with 1 experiments
--------------------------------------------------------------
Experimenter pid 51. Run this to gracefully terminate me:
       kill -USR1 51
I will terminate myself as soon as current benchmark finishes.
--------------------------------------------------------------
------------------------------------------------------------------------
- DLBS benchmark session summary (this is not a performance report) -
------------------------------------------------------------------------
  Start time: ................... 2020-06-17 10:57:16.907766
  End time: ..................... 2020-06-17 11:00:09.461530
  Duration (minutes): ........... 2.87589606667
  Total benchmarks in plan: ..... 1
  |--Inactive benchmarks: ....... 0
  |--Existing benchmarks: ....... 0
  |  |--Successful benchmarks: .. 0
  |  |--Failed benchmarks: ...... 0
  |--Active benchmarks: ......... 1
  |  |--Completed benchmarks: ... 1
  |  |--Successful benchmarks: .. 1
  |  |--Failed benchmarks: ...... 0
------------------------------------------------------------------------
```

Figure 17: TensorFlow run leveraging Bitfusion in a TKG cluster

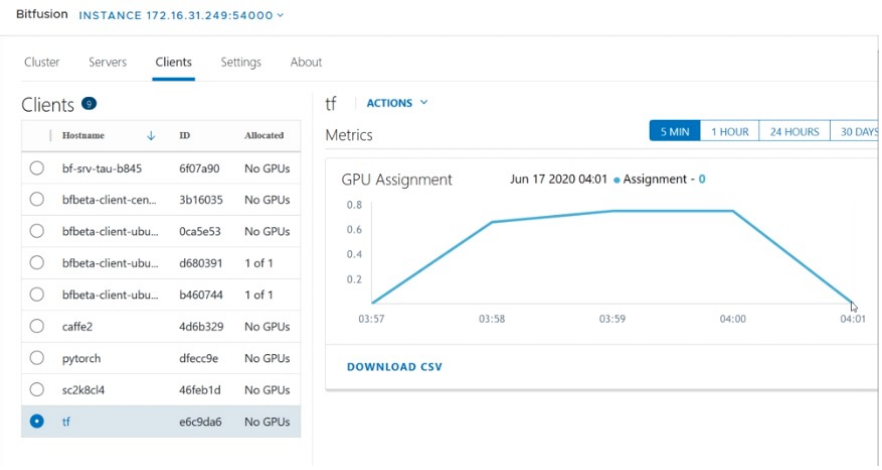

Figure 18: Bitfusion console during the TensorFlow

4.2 PodVM or vSphere native pod (Supervisor Clusters)

The **vSphere native pod (Supervisor Cluster)** in vSphere 7 combines the best of containers and virtualization by running each Kubernetes pod in its own, dynamically created VM.

```
[root@sc2k8cl4 tkg-bf]# kubectl config current-context
hpc2
[root@sc2k8cl4 tkg-bf]# kubectl get nodes
NAME                                STATUS   ROLES    AGE   VERSION
420013823c5b73283d65e88e1eb00e28    Ready    master   43d   v1.16.7-2+bfe512e5ddaaaa
42002935e11918c16ca21f7e51d8d34e    Ready    master   43d   v1.16.7-2+bfe512e5ddaaaa
420093bf6ccec1eafca4a3238876504b    Ready    master   43d   v1.16.7-2+bfe512e5ddaaaa
sc2esx22                            Ready    agent    43d   v1.16.7-sph-4d52cd1
sc2esx23                            Ready    agent    43d   v1.16.7-sph-4d52cd1
[root@sc2k8cl4 tkg-bf]# kubectl create -f pyt_tmpdisk.yaml
pod/pytorch created
[root@sc2k8cl4 tkg-bf]# kubectl get pods|grep pytorch
pytorch                                          1/1     Running        0          67s
[root@sc2k8cl4 tkg-bf]# kubectl exec -it pytorch -- /bin/bash
root@pytorch:/# /workspace/dlbs/run_pytorch.sh
 ["alexnet"]
WARNING:root:Module 'numpy' cannot be imported, some system information will not be available
WARNING:root:Module 'pandas' cannot be imported, some system information will not be available
WARNING:root:Module 'matplotlib' cannot be imported, some system information will not be available
INFO:root:Plan was built with 1 experiments
Plan was built with 1 experiments
------------------------------------------------------------
Experimenter pid 50. Run this to gracefully terminate me:
       kill -USR1 50
I will terminate myself as soon as current benchmark finishes.
------------------------------------------------------------
--------------------------------------------------------------------
- DLBS benchmark session summary (this is not a performance report) -
--------------------------------------------------------------------
  Start time: ................... 2020-06-17 09:54:04.348782
  End time: ..................... 2020-06-17 09:56:55.431550
  Duration (minutes): ........... 2.85137946667
  Total benchmarks in plan: ..... 1
  |--Inactive benchmarks: ....... 0
  |--Existing benchmarks: ....... 0
  |  |--Successful benchmarks: .. 0
  |  |--Failed benchmarks: ...... 0
  |--Active benchmarks: ......... 1
  |  |--Completed benchmarks: ... 1
  |  |--Successful benchmarks: .. 1
  |  |--Failed benchmarks: ...... 0
--------------------------------------------------------------------
```

Figure 19: PyTorch run leveraging Bitfusion in a supervisor cluster

The idea here is to leverage the isolation and security of a VM with the simplicity and configurability of a pod. vSphere Pods are also first-class entities in vSphere, so VI admins can both get full visibility into them from the vSphere Client, but can also use all their existing tooling to manage vSphere Pods just like existing VMs. Shown is a successful PyTorch run leveraging Bitfusion in a Supervisor cluster that was run as part of the validation.

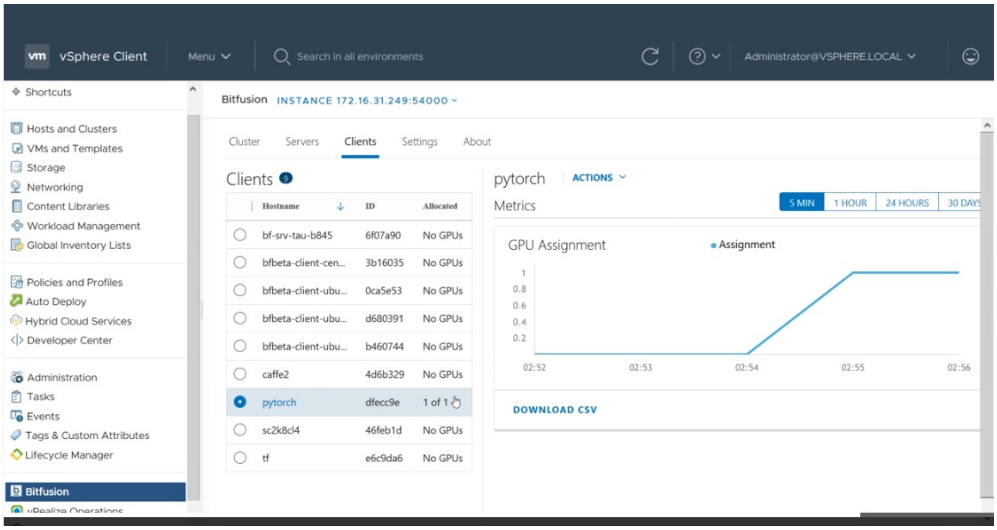

Figure 20: Bitfusion console during PyTorch run in Supervisor cluster

4.3 TKGI (formerly VMware Enterprise PKS)

```
root@pks-client-vm:/gpu_data/mlperf-data/tkg-bf# kubectl create -f tf_tmpdisk.yaml
pod/tf created
root@pks-client-vm:/gpu_data/mlperf-data/tkg-bf# kubectl get pods
NAME   READY   STATUS    RESTARTS   AGE
tf     1/1     Running   0          12s
root@pks-client-vm:/gpu_data/mlperf-data/tkg-bf# kubectl exec -it tf -- /bin/bash
root@tf:/workspace/dlbs# export TOTAL_NUM_BATCHES=1000
root@tf:/workspace/dlbs# /workspace/dlbs/run_tensorflow.sh
```

```
 ["alexnet"]
WARNING:root:Module 'numpy' cannot be imported, some system information will not be available
WARNING:root:Module 'pandas' cannot be imported, some system information will not be available
WARNING:root:Module 'matplotlib' cannot be imported, some system information will not be availab
INFO:root:Plan was built with 1 experiments
Plan was built with 1 experiments
------------------------------------------------------------
Experimenter pid 49. Run this to gracefully terminate me:
       kill -USR1 49
I will terminate myself as soon as current benchmark finishes.
------------------------------------------------------------
------------------------------------------------------------
- DLBS benchmark session summary (this is not a performance report) -
------------------------------------------------------------
  Start time: ................... 2020-06-17 11:10:30.250794
  End time: ..................... 2020-06-17 11:11:39.482146
  Duration (minutes): ........... 1.15385586667
  Total benchmarks in plan: ..... 1
  |--Inactive benchmarks: ....... 0
  |--Existing benchmarks: ....... 0
  |  |--Successful benchmarks: .. 0
  |  |--Failed benchmarks: ...... 0
  |--Active benchmarks: ......... 1
  |  |--Completed benchmarks: ... 1
  |  |--Successful benchmarks: .. 1
  |  |--Failed benchmarks: ...... 0
------------------------------------------------------------
- Your next steps: analyze performance data with bench_data.py      -
```

Figure 21: Successful TensorFlow run with TKGI leveraging NVIDIA GPUs over the network

The TensorFlow run shown below uses a remote NVIDIA GPU via Bitfusion for execution. The job executes via the network. The testing with the pods and the containers were seamless with TKGI.

4.4 Bitfusion with Kubernetes integration

Using Bitfusion GPU in Kubernetes workload

Use Bitfusion.io/gpu-amount and Bitfusion.io/gpu-percent parameters in YAML file to specify the resource the pod request.

```
#------------------------------------------------------------------------------
# yaml file to start pod
#------------------------------------------------------------------------------
apiVersion: v1
kind: Pod
metadata:
  annotations:
    auto-management/Bitfusion: "all"
    Bitfusion-client/os: "ubuntu18"
    Bitfusion-client/version: "250"
  name: bf-pkgs
  # You can specify any namespace
  namespace: tensorflow-benchmark
spec:
  containers:
    - image: nvcr.io/nvidia/tensorflow:19.07-py3
      imagePullPolicy: IfNotPresent
      name: bf-pkgs
```

```
      command: ["python
/benchmark/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py --
local_parameter_device=gpu --batch_size=32 --model=inception3"]
      resources:
        limits:
          # Request one GPU for this Pod from the Bitfusion cluster
          Bitfusion.io/gpu-amount: 1
          # 50 percent of each GPU to be consumed
          Bitfusion.io/gpu-percent: 50
      volumeMounts:
        - name: code
          mountPath: /benchmark
    volumes:
        - name: code
          # The Benchmarks used for the test came from:
https://github.com/tensorflow/benchmarks/tree/tf_benchmark_stage
          # Please make sure you have the corresponding content in
/home/benchmarks directory on your node
          hostPath:
            path: /home/benchmarks
```

```
......

TensorFlow:  1.14
Model:       inception3
Dataset:     imagenet (synthetic)
Mode:        training
SingleSess:  False
Batch size:  32 global
             32 per device
Num batches: 100
Num epochs:  0.00
Devices:     ['/gpu:0']
NUMA bind:   False
Data format: NCHW
Optimizer:   sgd
Variables:   parameter_server
==========
Generating training model
Initializing graph
Running warm up
Done warm up
Step    Img/sec total_loss
1       images/sec: 199.4 +/- 0.0 (jitter = 0.0)        7.312
10      images/sec: 196.6 +/- 2.1 (jitter = 5.7)        7.290
20      images/sec: 198.3 +/- 1.3 (jitter = 4.5)        7.351
30      images/sec: 198.4 +/- 0.9 (jitter = 3.8)        7.300
40      images/sec: 199.4 +/- 0.8 (jitter = 4.1)        7.250
50      images/sec: 199.8 +/- 0.7 (jitter = 4.6)        7.283
60      images/sec: 200.1 +/- 0.6 (jitter = 4.2)        7.301
70      images/sec: 199.8 +/- 0.6 (jitter = 4.2)        7.266
80      images/sec: 200.1 +/- 0.6 (jitter = 4.4)        7.286
90      images/sec: 199.9 +/- 0.5 (jitter = 4.4)        7.334
100     images/sec: 199.9 +/- 0.5 (jitter = 4.0)        7.380
----------------------------------------------------------------
total images/sec: 199.65
----------------------------------------------------------------

......
```

Figure 22: Successful TensorFlow run

5 Summary:

This solution has successfully validated access to Bitfusion served GPUs from all three VMware Kubernetes platforms. ML training runs with TensorFlow and PyTorch were executed successfully with GPU access over the network through Bitfusion.

This solution has effectively shown the following:

- VMware Cloud foundation with vSphere 7 has two key new features
    - Native Support for Kubernetes
    - Bitfusion which enables sharing of GPUs across the network
- This solution showcased integration between Kubernetes and Bitfusion
- vSphere Bitfusion can be used over the network from Kubernetes for machine learning

Appendix A. Dockerfile for creating PyTorch container:

```
#-----------------------------------------------------------------------------------
# IMPORTANT:
#-----------------------------------------------------------------------------------
# We are adding dlbs code to make use of the benchmark scripts that
# is available with dlbs
#
# This example Dockerfile illustrates a method to install
# additional packages on top of NVIDIA's pytorch container image.
#
# To use this Dockerfile, use the `docker build` command.
# See https://docs.docker.com/engine/reference/builder/
# for more information.
#
#-----------------------------------------------------------------------------------
FROM nvcr.io/nvidia/pytorch:19.02-py3
#-----------------------------------------------------------------------------------
# Copy Bitfusion client related files
#-----------------------------------------------------------------------------------
RUN mkdir -p /root/.Bitfusion
COPY ./Bitfusion-files/client.yaml /root/.Bitfusion/client.yaml
COPY ./Bitfusion-files/servers.conf /etc/Bitfusion/servers.conf
RUN mkdir -p /etc/Bitfusion/tls
COPY ./Bitfusion-files/ca.crt /etc/Bitfusion/tls/ca.crt
COPY ./Bitfusion-files/nvidia-smi /workspace/Bitfusion/batch-scripts/nvidia-smi
#-----------------------------------------------------------------------------------
# Update package list
# Install Bitfusion. Use deb file for Ubuntu16.04
# Install open-vm-tools
#-----------------------------------------------------------------------------------
#  Set initial working directory
RUN mkdir -p /workspace/Bitfusion/batch-scripts
WORKDIR /workspace/Bitfusion
# Copy Release version of Bitfusion client
COPY ./Bitfusion-files/Bitfusion-client-ubuntu1604_2.0.0v12nstaging-876_amd64.deb .
RUN apt-get update \
      && apt-get install -y ./Bitfusion-client-ubuntu1604_2.0.0v12nstaging-
876_amd64.deb \
      && apt-get install -y open-vm-tools \
      && \
      rm -rf /var/lib/apt/lists/
#-----------------------------------------------------------------------------------
# This is for DLB cookbook - experimenter.py needs python 2.7
# So installing it too
#-----------------------------------------------------------------------------------
RUN apt-get update && apt-get install -y --no-install-recommends \
        python2.7 \
      && \
    rm -rf /var/lib/apt/lists/
#-----------------------------------------------------------------------------------
# Copy dlbs dir
#-----------------------------------------------------------------------------------
RUN mkdir -p /workspace/dlbs
WORKDIR /workspace/dlbs
COPY ./dlbs /workspace/dlbs
#-----------------------------------------------------------------------------------
# End of Dockerfile
#-----------------------------------------------------------------------------------
```

Appendix B. PyTorch pod config yaml file

```
#-------------------------------------------------------------------------------------------
# yaml file to start pod for pytorch
#-------------------------------------------------------------------------------------------
apiVersion: v1
kind: Pod
metadata:
  name: pytorch
spec:
  containers:
  - name: bit2pyt
    # The following image id for HP DLB pytorch
    image: pmohan77/Bitfusion2:pyt_cuda_1902py3
    "imagePullPolicy": "Always"
    resources:
      limits:
        memory: "16Gi"
      requests:
        memory: "16Gi"
    #
    #The command that is run
    #command: ["/workspace/dlbs/run_pytorch.sh"]
    command: ["/bin/bash", "-ec", "while :; do echo '.'; sleep 300 ; done"]
    env:
      - name: POD_UID
        valueFrom:
          fieldRef:
            apiVersion: v1
            fieldPath: metadata.uid
      - name: OUTPUT_PATH
        value: "/tmp"
      - name: NUM_WARMUPS
        value: "20"
      - name: TOTAL_NUM_BATCHES
        value: "100"
      - name: MODEL_LIST
        #value: "alexnet resnet50 vgg16"
        value: "alexnet"
      - name: BATCH_SIZE
        value: "32"
      - name: PARTIALGPU
        # Change value to the required fraction of GPU
        #value: "0.75"
        value: "1.0"
      - name: SERVER_LIST
        value: "--server_list 172.16.31.247:56001"
  restartPolicy: Never
#-------------------------------------------------------------------------------------------
```

Appendix C. Dockerfile for creating TensorFlow container:

```
#-----------------------------------------------------------------------------
# IMPORTANT:
# We are adding dlbs code to make use of the benchmark scripts that is
# available with dlbs
# Have adapted tensorflow benchmark code to run under dlbs benchmark scripts
# This example Dockerfile illustrates a method to install
# additional packages on top of NVIDIA's tensorflow container image.
#
# To use this Dockerfile, use the `docker build` command.
# See https://docs.docker.com/engine/reference/builder/
# for more information.
#-----------------------------------------------------------------------------
FROM nvcr.io/nvidia/tensorflow:19.03-py3-
# Copy Bitfusion client related files
#-----------------------------------------------------------------------------
# Copy Bitfusion files
RUN mkdir -p /root/.Bitfusion
COPY ./Bitfusion-files/client.yaml /root/.Bitfusion/client.yaml
COPY ./Bitfusion-files/servers.conf /etc/Bitfusion/servers.conf
RUN mkdir -p /etc/Bitfusion/tls
COPY ./Bitfusion-files/ca.crt /etc/Bitfusion/tls/ca.crt
COPY ./Bitfusion-files/nvidia-smi /workspace/Bitfusion/batch-scripts/nvidia-smi
#-----------------------------------------------------------------------------
# Update package list
# Install Bitfusion. Use deb file for Ubuntu16.04
# Install open-vm-tools
#-----------------------------------------------------------------------------
#  Set initial working directory
RUN mkdir -p /workspace/Bitfusion/batch-scripts
WORKDIR /workspace/Bitfusion
# Copy Release version of Bitfusion client
COPY ./Bitfusion-files/Bitfusion-client-ubuntu1604_2.0.0v12nstaging-876_amd64.deb .
RUN apt-get update \
      && apt-get install -y ./Bitfusion-client-ubuntu1604_2.0.0v12nstaging-
876_amd64.deb \
      && apt-get install -y open-vm-tools \
      && \
      rm -rf /var/lib/apt/lists/
#-----------------------------------------------------------------------------
# This is for DLB cookbook - experimenter.py needs python 2.7
# So installing it too
#-----------------------------------------------------------------------------
RUN apt-get update && apt-get install -y --no-install-recommends \
        python2.7 \
      && \
    rm -rf /var/lib/apt/lists/
#-----------------------------------------------------------------------------
# Copy dlbs dir
# Then copy latest tensorflow benchmark code - otherwise we get errors like:
#     " ImportError: cannot import name 'nccl'"
#-----------------------------------------------------------------------------
RUN mkdir -p /workspace/dlbs
WORKDIR /workspace/dlbs
COPY ./dlbs /workspace/dlbs
WORKDIR /workspace/dlbs/python
RUN git clone https://github.com/tensorflow/benchmarks.git
RUN mv tf_cnn_benchmarks/ 1tf_cnn_benchmarks
RUN cp -R benchmarks/scripts/tf_cnn_benchmarks/ .
WORKDIR /workspace/dlbs
#-----------------------------------------------------------------------------
# End of Dockerfile
#-----------------------------------------------------------------------------
```

Appendix D. TensorFlow pod config yaml file

```
#-------------------------------------------------------------------------------------------
# yaml file to start pod for tensorflow
#-------------------------------------------------------------------------------------------
apiVersion: v1
kind: Pod
metadata:
  name: tf
spec:
  containers:
  - name: bit2tf
    # The following image id for DLB tensorflow
    image: pmohan77/Bitfusion2:tf_cuda_dlbs
    "imagePullPolicy": "Always"
    resources:
      limits:
        memory: "16Gi"
      requests:
        memory: "16Gi"
    #The command that is run
    #command: ["/workspace/dlbs/run_tensorflow.sh"]
    command: ["/bin/bash", "-ec", "while :; do echo '.'; sleep 300 ; done"]
    env:
      - name: POD_UID
        valueFrom:
          fieldRef:
            apiVersion: v1
            fieldPath: metadata.uid
      - name: OUTPUT_PATH
        value: "/tmp"
      - name: NUM_WARMUPS
        value: "20"
      - name: TOTAL_NUM_BATCHES
        value: "100"
      - name: MODEL_LIST
        #value: "alexnet resnet50 vgg16"
        value: "alexnet"
      - name: BATCH_SIZE
        value: "32"
      - name: PARTIALGPU
        # Change value to the required fraction of GPU
        #value: "0.75"
        value: "1.0"
      - name: SERVER_LIST
        value: "--server_list 172.16.31.247:56001"
  restartPolicy: Never
#-------------------------------------------------------------------------------------------
```