NVIDIA Riva on Red Hat OpenShift with Dell PowerFlex

Automatic speech recognition and inference with NVIDIA GPU on PowerFlex

October 2023

H19645

White Paper

Abstract

This whitepaper describes how to integrate and configure NVIDIA A100 GPUs running with Red Hat OpenShift on Dell PowerFlex software-defined infrastructure. The paper also shows the performance of Riva speech services using GPUs running on PowerFlex nodes.

Dell Technologies Solutions

PowerFlex Engineering
Validated

Contents

Chapter 1 Executive summary

This chapter presents the following topics:

Overview

Deep learning (DL), machine learning (ML), and artificial intelligence (AI) technologies have enabled success in many fields, such as computer vision, natural language processing (NLP), recommendation engines, and autonomous driving. These technologies use a model to learn from the existing data and generate results based on the new data. The success of these technologies is due to a combination of improved algorithms, access to larger datasets, and the increased computational power of both the CPU and GPU.

Speech recognition has recently been developed to deliver significant values in areas of automotive, customer services, and smart home device. To achieve high level of accuracy in a speech recognition can be challenging. This paper describes a solution that achieves automated speech recognition and inference with NVIDIA GPUs on Red Hat OpenShift along with Dell PowerFlex software-defined infrastructure.

NVIDIA Riva, part of the NVIDIA AI platform, is a GPU-accelerated speech AI SDK for building and deploying fully customizable, real-time AI pipelines that deliver world-class accuracy in all clouds, on premises, at the edge, and on embedded devices.

This paper also describes the integration and performance of NVIDIA Riva services with PowerFlex two-layer architecture. The laboratory set up for this environment includes PowerFlex, Dell PowerEdge servers, Dell switches, Red Hat OpenShift, and NVIDIA AI Enterprise.

Why PowerFlex for AI/ML

PowerFlex Benefits

Some of the unique capabilities of PowerFlex which helps in addressing the AI/ML application challenges are:

Enhanced performance

- Consistent I/O performance experience for transactions consisting of streaming, ingestion, analysis, and reporting.
- Scale out architecture with resources aggregation eliminates performance bottleneck.

Absolute flexibility and scalability requirements

- Complete flexibility to scale compute and storage and enable a **Pay as you grow** model.
- Extensive support for GPUs and flexible configurations.

High ingest and indexing requirements

- 100 GbE support to address intense throughput requirements.
- Scale out architecture for compute, throughput, and I/O to meet the ingest and indexing requirements.

Audience

This document is intended for sales engineers, field consultants, IT administrators, customers, and anyone else who is interested in configuring and deploying NVIDIA GPU on Dell PowerFlex infrastructure.

Readers are also expected to have an understanding and working knowledge of AI/ML, NVIDIA GPUs, NVIDIA Riva, PowerFlex, Red Hat OpenShift and the PowerFlex family.

Terminology

The following table provides the definitions for some of the terms that are used in this document.

Table 1. Terminology

Term	Definition
AZ	Availability Zones
BOSS	Boot Optimized Storage Solution
CO	Compute Only
ITOM	IT Operations Management
LCM	Life Cycle Management
PD	Protection Domain
SDC	Storage Data Client for PowerFlex
SDS	Storage Data Server for PowerFlex
SO	Storage Only
TOR	Top of Rack
VLAN	Virtual Local Area Network
ASR	Automatic Speech Recognition
TTS	Text To Speech
NLP	Natural Language Processing

We value your feedback

Dell Technologies and the authors of this document welcome your feedback on the solution and the solution documentation. Contact the Dell Technologies Solutions team by email.

Author: Kailas Goliwadekar

Contributor: Shashikiran Chidambara

Content editor: Ripa Bhagawati

Chapter 2 Product overview

This chapter presents the following topics:

PowerFlex

PowerFlex family

PowerFlex software-defined infrastructure enables broad consolidation across the data center, encompassing almost any type of workload and architecture. The software-defined architecture offers automation and programmability of the complete infrastructure and provides scalability, performance, and resiliency to enable effortless adherence to stringent workload Service Level Agreements (SLAs).

The PowerFlex family provides a foundation that combines compute and high-performance storage resources in a managed unified fabric. PowerFlex comes in flexible deployment options (rack, appliance, or custom nodes and in the public cloud) that enables independent (two-layer), HCI (single-layer), or mixed architectures. PowerFlex is ideal for high-performance applications and databases, building an agile private/hybrid cloud, or consolidating resources in heterogeneous environments.

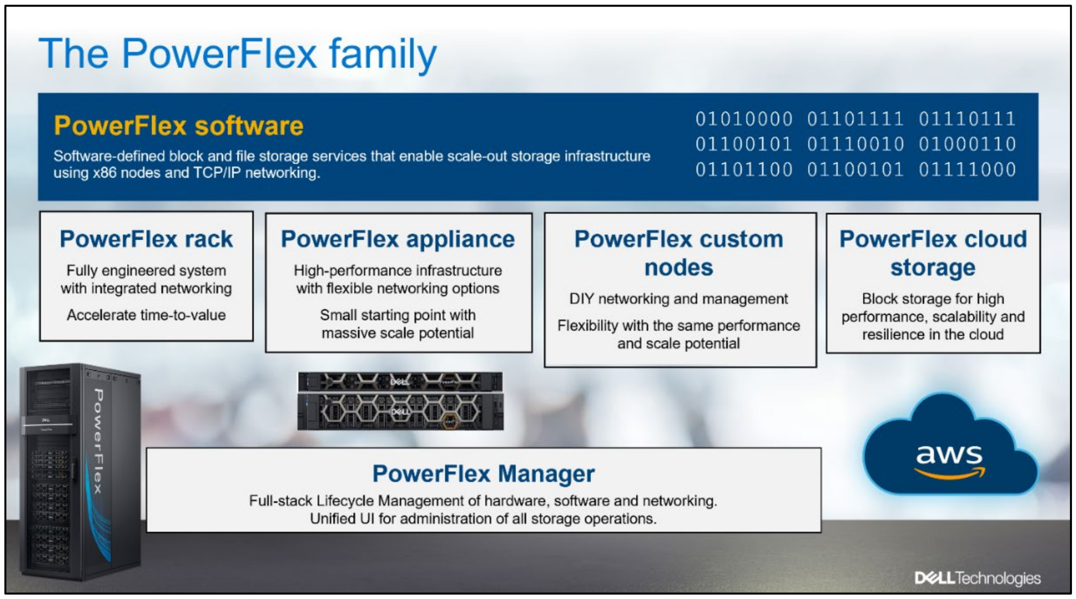

The PowerFlex family
PowerFlex software
Software-defined block and file storage services that enable scale-out storage infrastructure using x86 nodes and TCP/IP networking.
01010000 01101111 01110111
01100101 01110010 01000110
01101100 01100101 01111000
PowerFlex rack
Fully engineered system with integrated networking
Accelerate time-to-value
PowerFlex appliance
High-performance infrastructure with flexible networking options
Small starting point with massive scale potential
PowerFlex custom nodes
DIY networking and management
Flexibility with the same performance and scale potential
PowerFlex cloud storage
Block storage for high performance, scalability and resilience in the cloud
PowerFlex Manager
Full-stack Lifecycle Management of hardware, software and networking.
Unified UI for administration of all storage operations.
aws
DELL Technologies

Figure 1. PowerFlex family

PowerFlex software components

Software is the key differentiation in the PowerFlex offering. PowerFlex software components provide software-defined storage services and also help simplify infrastructure management and orchestration. This software enables comprehensive IT Operational Management (ITOM) and Life Cycle Management (LCM) capabilities that span compute and storage infrastructure, from BIOS and Firmware to nodes, software, and networking.

PowerFlex

PowerFlex is the software foundation of PowerFlex software-defined infrastructure. It is a scale-out block and file storage service that is designed to deliver flexibility, elasticity, and simplicity with predictable high performance and resiliency at scale.

PowerFlex Manager

PowerFlex Manager is the software component in PowerFlex family that enables ITOM automation and LCM capabilities for PowerFlex systems. Starting with PowerFlex 4.0, the unified PowerFlex Manager brings together three separate components that are used in previous releases: PowerFlex Manager, the core PowerFlex UI, and the PowerFlex gateway. The new PowerFlex UI runs in Kubernetes and embraces a modern development framework.

PowerFlex File Services

PowerFlex File Controllers, also known as File Nodes, are physical nodes that enable PowerFlex software-defined File Services. They host the Network-Attached Storage (NAS) Servers, which in turn host the tenant namespaces and file systems, mapping PowerFlex volumes to the file systems presented by the NAS Servers. All major protocols are supported, such as NFS, SMB/CIFS, FTP, and NDMP.

PowerFlex deployment architectures

PowerFlex software-defined infrastructure excels in deployment flexibility. PowerFlex can be deployed in a two-layer (independent compute and storage layers), single-layer (Hyperconverged Infrastructure, or HCI), or a mixture of the two architectures (Mixed).

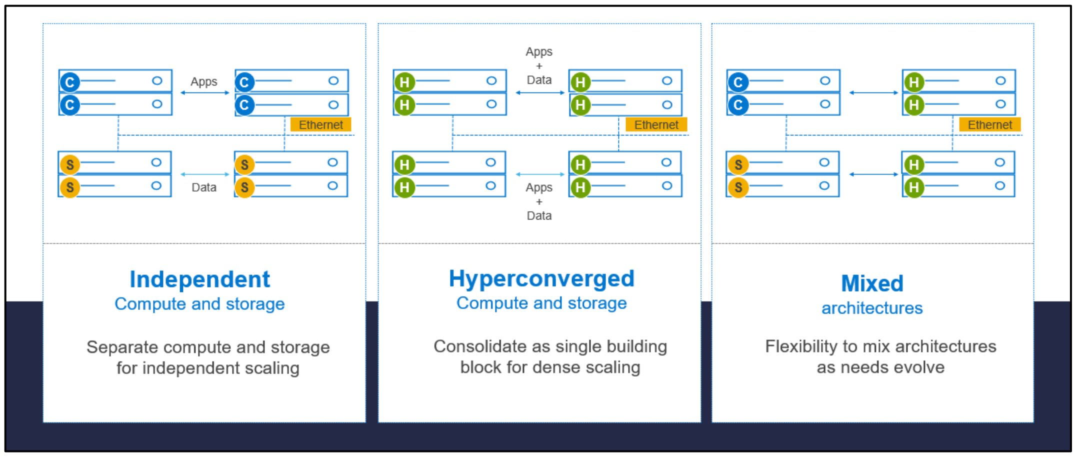

Apps
Ethernet
Data
Apps + Data
Ethernet
Apps + Data
Ethernet
Independent
Compute and storage
Separate compute and storage for independent scaling
Hyperconverged
Compute and storage
Consolidate as single building block for dense scaling
Mixed
architectures
Flexibility to mix architectures as needs evolve

Figure 2. PowerFlex deployment architectures

Independent architecture

In an independent architecture, or two-layer architecture, some nodes provide storage capacity for applications data while other separate and independent nodes provide compute resources for applications and workloads. Compute and storage resources can be scaled independently by adding nodes to the cluster while it remains active. This separation of compute and storage resources can help minimize software licensing costs in certain situations. This architecture is ideal for high-performance databases and application workloads.

Hyperconverged architecture

In an HCI architecture, each node in the cluster contributes storage and compute resources simultaneously to the applications and workloads. This architecture allows you to scale your infrastructure uniformly with building blocks that add both storage and compute resources. This architecture is appropriate for data center and workload consolidation.

Mixed architecture

In a mixed architecture, we have a combination of both the HCI and Independent architectures. As shown in Figure 2 some storage-only nodes, compute only nodes, and HCI nodes that are part of the same PowerFlex cluster. This is a desirable architecture when working with an existing compute infrastructure and adding high-performance software-defined infrastructure. This is also a starting point for a two-layer deployment design when external workloads are migrated to PowerFlex.

PowerFlex consumption options

PowerFlex rack

PowerFlex rack is a software-defined infrastructure platform that delivers flexibility, elasticity, and simplicity with predictable performance and resiliency at scale by combining compute as well as high-performance storage resources in a managed unified network. This rack-based engineered system, with integrated networking, enables customers to achieve the scalability and management requirements of a modern data center.

PowerFlex appliance

PowerFlex appliance is a PowerEdge server which has been configured as a node in a software-defined infrastructure deployment that runs PowerFlex software components. This offering allows customers the flexibility and savings to bring their own compatible networking.

PowerFlex custom nodes

PowerFlex Custom Nodes are validated server building-blocks that are configured for use with PowerFlex. Custom nodes are available with thousands of configuration options and are available for customers who to build their own environments.

PowerFlex on AWS

PowerFlex software can be deployed in the public cloud and is available in the Amazon Marketplace as PowerFlex cloud storage. PowerFlex on AWS offers the same on-premises benefits of high-performance, linear scalability, and high resilience as in the cloud. PowerFlex also adds cloud-specific benefits, such as large volume sizes, extreme performance based on NVMe drives, and predictable scalability. With PowerFlex on AWS, you can also get higher multi-Availability-Zone (multi-AZ) resiliency when PowerFlex Fault sets are distributed across multiple AWS Availability Zones.

Red Hat OpenShift

OpenShift Assisted Installer

OpenShift Assisted Installer provides a simple way to deploy OpenShift on on-premises nodes from the Red Hat Hybrid Cloud Console. The installer is a SaaS-based managed service to install OpenShift clusters on bare metal (although it is possible to use Assisted Installer for a wide variety of platforms). Before installing a cluster, you do not require a dedicated bootstrap node, and you do not need to preinstall Red Hat CoreOS on the nodes. Cluster deployment parameters are populated within the web UI and nodes are booted with a downloaded ISO image to complete the cluster deployment.

For more information about assisted installer, see OpenShift Assisted Installer blog.

NVIDIA A100 Tensor Core GPU

Overview

The NVIDIA A100 Tensor Core GPU delivers unprecedented acceleration at every scale to power the highest-performing elastic data centers for AI, data analytics, and HPC. This GPU uses the NVIDIA Ampere Architecture. The third-generation A100 provides higher performance than the prior generation and can be partitioned into seven GPU instances to dynamically adjust to the shifting demands.

The Tensor Core technology that is in the Ampere architecture brings dramatic performance gains to AI workloads. The A100 GPU can achieve high acceleration for inference workloads. This technology provides a significant advantage for the data scientist and the organization. IT professionals also benefit from reduced operational complexity by using a single technology that is easy to onboard and manage across use cases.

The A100 GPU is a dual-slot 10.5 inches PCI Express (PCIe) Gen4 card that is based on the NVIDIA Ampere architecture. It uses a passive heat sink for cooling. The A100 PCIe based GPU supports double precision (FP64), single precision (FP32), and half precision (FP16) compute tasks. It also supports unified virtual memory, and a page migration engine. The A100 GPU is available in 40 GB and 80 GB memory versions.

For more information, see NVIDIA A100 Tensor Core GPU documentation.

NVIDIA Riva

Overview

NVIDIA Riva is a GPU-accelerated SDK for building speech AI applications that can be customized to deliver real-time performance. Riva offers pretrained speech models in NVIDIA NGC that can be fine-tuned on a custom dataset.

Models can be exported, optimized, and deployed as a speech service on premises or in the cloud with a single command using Helm charts. Riva's high-performance inference is powered by NVIDIA TensorRT optimizations and served using the NVIDIA Triton Inference Server, which are both part of the NVIDIA AI platform.

Riva services are available for low-latency streaming, and high-throughput offline use cases. Riva is fully containerized and can scale to hundreds and thousands of parallel streams. For more information about Riva, see NVIDIA Riva. To learn more about purchasing Riva, contact NVIDIA Sales.

Chapter 3 Solution design

This chapter presents the following topics:

Overview

This chapter describes the logical architecture of Riva speech services on NVIDIA A100 GPUs with two-layer PowerFlex architecture on an OpenShift cluster. In this solution, the OpenShift cluster is installed on three master and four worker nodes (bare metal nodes). This solution can also be deployed on PowerFlex hyperconverged architecture.

Logical architecture

The overall deployment consists of the following stages:

- PowerFlex Storage Only (SO) node deployment is done with the help of PowerFlex Manager which automates the whole process. The SO nodes are deployed with the PowerFlex embedded operating system. Each SO node is populated with ten 894 GB SAS SSD drives. For more information, see PowerFlex Node specifications.
- Next, an OpenShift cluster is installed on three bare metal master nodes and four bare metal worker nodes with the help of Red Hat OpenShift Assisted Installer.
- Next, CSI 2.5 is installed so that the PowerFlex storage becomes available to all the OpenShift containers. The CSI installation also automates the SDC deployment on the Red Hat worker nodes.
- This completes the PowerFlex two-layer deployment and the OpenShift cluster deployment. Once OpenShift cluster is installed, then NVIDIA Riva can be installed as a container to carry out speech recognition and inference.

The detailed steps on how to install GPU operator along with Riva are mentioned in the GPU Operator installation chapter.

The following figure shows the overall logical architecture of this solution.

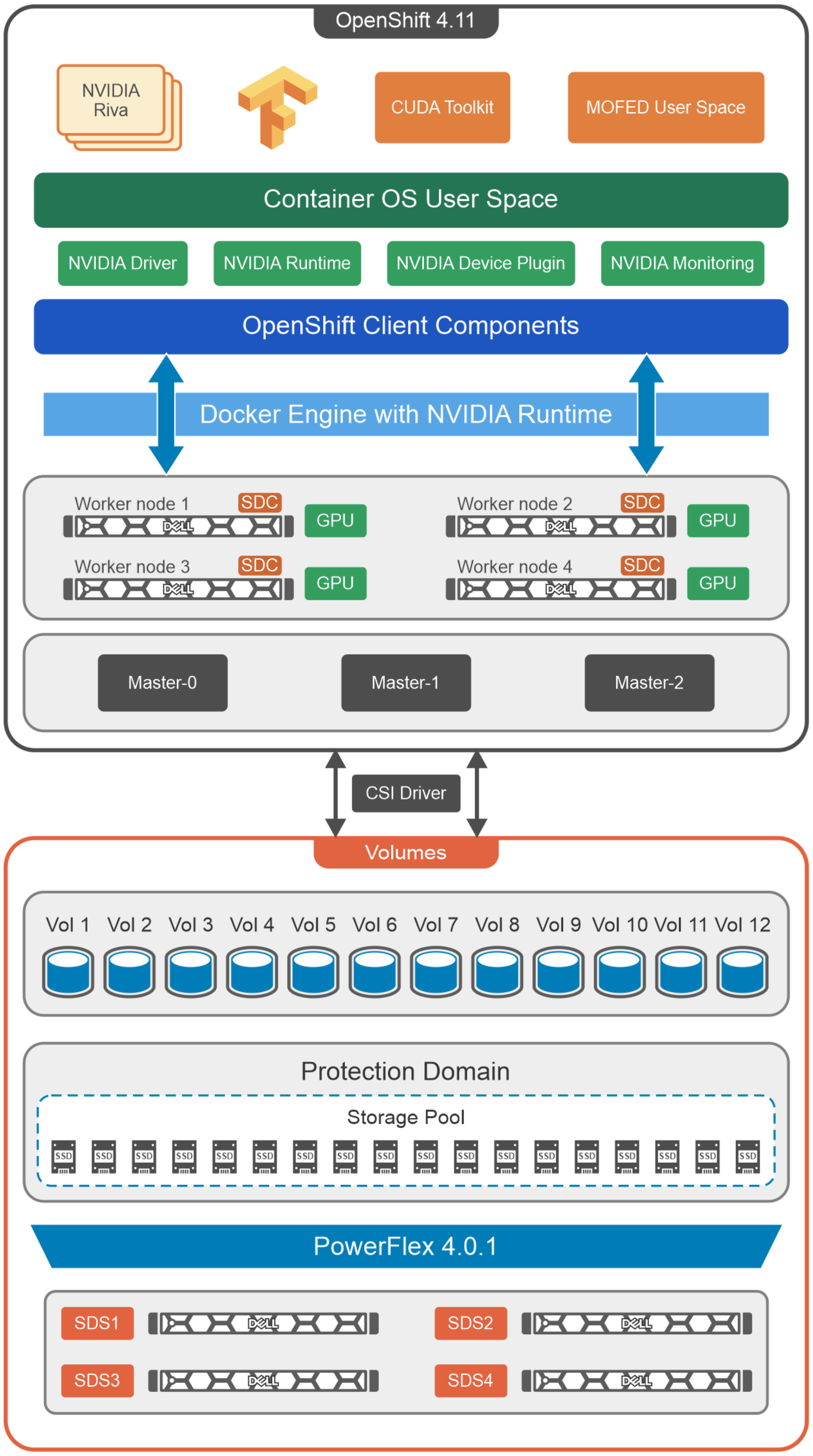
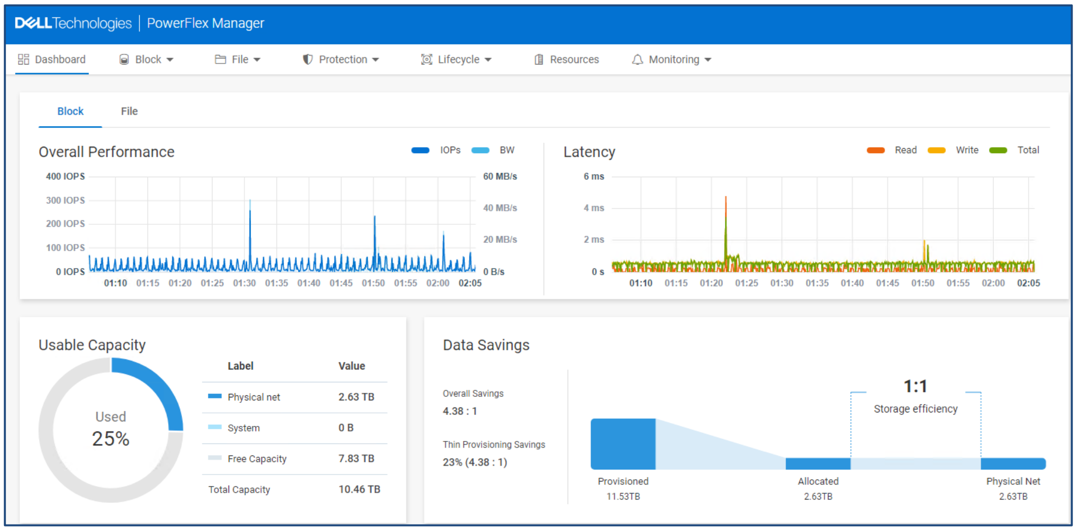


Figure 3. Logical architecture of the solution

Next, a single PowerFlex Protection Domain (PD) is created that consists of four SDSs. One Storage Pool is created from the physical storage devices within the PD. A PowerFlex volume of 2 TB is created out from this storage pool.

The following figures show a sample PowerFlex dashboard and information about the resources:

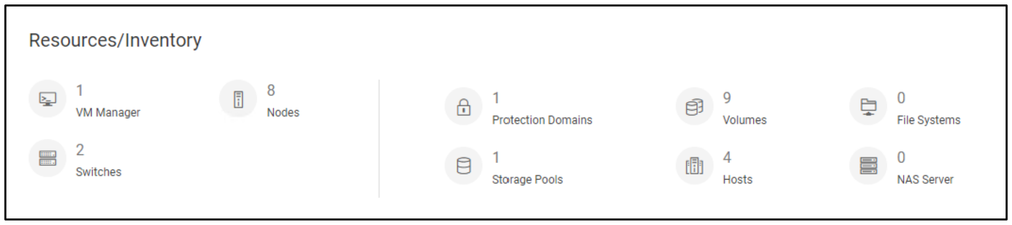


Label	Value
Physical net	2.63 TB
System	0 B
Free Capacity	7.83 TB
Total Capacity	10.46 TB

Overall Savings
4.38 : 1

Thin Provisioning Savings
23% (4.38 : 1)

Figure 4. PowerFlex 4.0.1 dashboard


Resources/Inventory

1	VM Manager
8	Nodes
2	Switches
1	Protection Domains
9	Volumes
0	File Systems
1	Storage Pools
4	Hosts
0	NAS Server

Figure 5. PowerFlex 4.0 Resources/Inventory information

Network architecture

PowerFlex Manager automates the entire switch configurations along with bond configuration for PowerFlex SO nodes. For PowerFlex Compute Only (CO) nodes, bonding is done on master and worker nodes by applying YAML files. A sample YAML file is available for reference in Appendix.

The following diagram shows the network architecture of this solution.

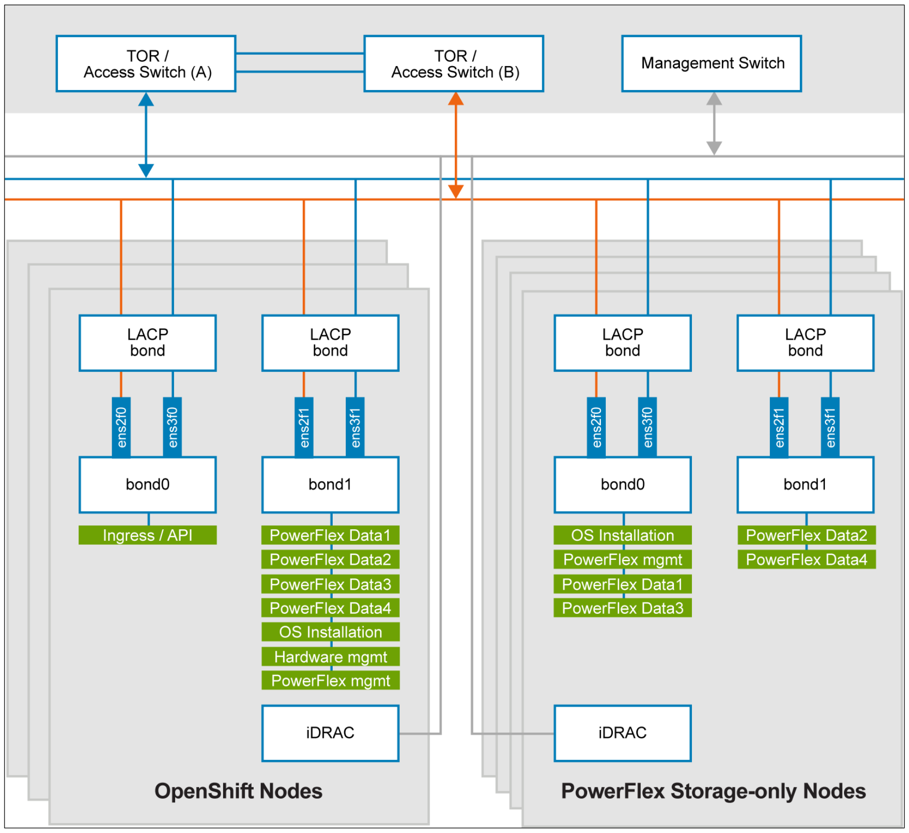


Figure 6. Network design of OpenShift nodes and PowerFlex SO nodes

At the physical layer, two supported Top of Rack (TOR) switches are used for redundancy and load-balancing purposes. A peer link is configured on both the TOR switches. VLANs are created to separate different traffic types on the network bonds.

The following table shows the different networks that are configured for this solution. For more information, see PowerFlex node VLAN setup:

Table 2. Network details

Network type	VLAN ID
Ingress/API	105
Operating System Installation	104
Hardware Management	101
PowerFlex Management	150
PowerFlex Data 1	152
PowerFlex Data 2	153
PowerFlex Data 3	154
PowerFlex Data 4	155

Chapter 4 Riva integration with PowerFlex

This chapter presents the following topics:

Overview

Red Hat OpenShift Container Platform includes enhancements to Kubernetes that enable users to configure and use GPU resources for accelerating workloads such as deep learning.

The NVIDIA GPU Operator uses the operator framework in Kubernetes to automate the management of all NVIDIA software components that are needed to provision GPU.

The OpenShift container platform uses storage from PowerFlex with the help of CSI. Initially, volumes are created at PowerFlex and the GPU operator along with Riva operator uses the PowerFlex storage to store the audio and text files.

Before deploying the `gpu-operator`, ensure that your environment has:

- A working OpenShift cluster up and running with a GPU worker node
- Access to the OpenShift cluster as a cluster-admin to perform the necessary steps
- OpenShift CLI (oc) installed

GPU operator installation

To install the gpu-operator on OpenShift, perform the following:

- Install the Node Feature Discovery (NFD) operator
- Install the GPU operator

Node Feature Discovery operator

The Node Feature Discovery (NFD) operator manages the detection of hardware features and configuration in a OpenShift Container Platform cluster by labeling the nodes with hardware-specific information.

To install the NFD operator:

1. In the OpenShift Container Platform web console, click **Operators > OperatorHub** and search for **Node Feature Discovery.**
2. Choose **Node Feature Discovery** from the list of available Operators, and then click **Install**.
3. On the Install Operator page, select the default namespace to be populated on the cluster (in this case it is `openshift-nfd`), and then click **Install**.
4. Create an instance of NodeFeatureDiscovery by clicking **Create Instance** after the NFD operator is installed.

After installation, check that the node feature discovery pods are running as follows:

```
[root@ocp411-admin Riva]# oc get pods -n openshift-nfd
NAME                                      READY   STATUS
RESTARTS   AGE
nfd-controller-manager-745467c897-2zk99   2/2     Running   0
11d
```

```
nfd-master-69lkh                                 1/1     Running   0
11d
nfd-master-7vksg                                 1/1     Running   0
11d
nfd-master-htn6d                                 1/1     Running   0
11d
nfd-worker-26ld2                                 1/1     Running   0
11d
nfd-worker-t2gx8                                 1/1     Running   0
11d
nfd-worker-whv4z                                 1/1     Running   0
11d
nfd-worker-xzs6w                                 1/1     Running   0
11d
```

NVIDIA GPU operator

After installing the NFD operator and creating a NodeFeatureDiscovery instance, you can install the NVIDIA GPU operator. The NVIDIA GPU operator makes the underlying GPUs of a compute node available to containerized workloads. When the NVIDIA GPU operator is installed, create an instance of ClusterPolicy.

To install the NVIDIA GPU operator:

1. In the side menu of the OpenShift Container Platform web console, select **Operators** > **OperatorHub** > **All Projects**.
2. Select **Operators** > **OperatorHub**, and search for the NVIDIA GPU operator.
3. Select the NVIDIA GPU operator and click **Install**.
4. In the following screen click **Install**.
5. Create the cluster policy using the web console.
   a. In the OpenShift Container Platform web console, from the side menu, select **Operators > Installed Operators**, and click **NVIDIA GPU** operator.
   b. Select the ClusterPolicy tab, then click **Create ClusterPolicy**.

      The platform assigns the default name as gpu-cluster-policy.
   c. Click **Create** to create the cluster policy.

After the installation, verify that the gpu operator is running:

```
[root@ocp411-admin Riva]# oc get pods -n nvidia-gpu-operator
NAME                                                       READY
STATUS      RESTARTS
console-plugin-nvidia-gpu-7dc9cfb5df-767nz                 1/1
Running     0
gpu-feature-discovery-8dpvj                                1/1
Running     0
gpu-feature-discovery-bp7z8                                1/1
Running     0
```

```
gpu-feature-discovery-f8nxr                             1/1
Running     0
gpu-feature-discovery-xqk47                             1/1
Running     0
gpu-operator-6cb9b76c7d-9zr55                           1/1
Running     0
nvidia-container-toolkit-daemonset-57cl6                1/1
Running     0
nvidia-container-toolkit-daemonset-9qr6g                1/1
Running     0
nvidia-container-toolkit-daemonset-dw7dp                1/1
Running     0
nvidia-container-toolkit-daemonset-rzpqq                1/1
Running     0
nvidia-cuda-validator-2fgfw                             0/1
Completed   0
nvidia-cuda-validator-8kphn                             0/1
Completed   0
nvidia-cuda-validator-j4zxn                             0/1
Completed   0
nvidia-cuda-validator-pzsj9                             0/1
Completed   0
nvidia-dcgm-47mj6                                       1/1
Running     0
nvidia-dcgm-exporter-6fg5c                              1/1
Running     0
nvidia-dcgm-exporter-p565m                              1/1
Running     0
nvidia-dcgm-exporter-psj7l                              1/1
Running     0
nvidia-dcgm-exporter-zjlnz                              1/1
Running     0
nvidia-dcgm-g4scr                                       1/1
Running     0
nvidia-dcgm-kw8cm                                       1/1
Running     0
nvidia-dcgm-v7vx9                                       1/1
Running     0
nvidia-device-plugin-daemonset-6xq9f                    1/1
Running     0
nvidia-device-plugin-daemonset-dbn9b                    1/1
Running     0
nvidia-device-plugin-daemonset-gps6x                    1/1
Running     0
nvidia-device-plugin-daemonset-gtqvw                    1/1
Running     0
nvidia-device-plugin-validator-44jnv                    0/1
Completed   0
nvidia-device-plugin-validator-87sqc                    0/1
Completed   0
```

```
nvidia-device-plugin-validator-c8q96                    0/1
Completed   0
nvidia-device-plugin-validator-hxcfx                    0/1
Completed   0
nvidia-driver-daemonset-411.86.202304190130-0-cf89x   2/2
Running     0
nvidia-driver-daemonset-411.86.202304190130-0-prgwg   2/2
Running     0
nvidia-driver-daemonset-411.86.202304190130-0-xl64j   2/2
Running     0
nvidia-driver-daemonset-411.86.202304190130-0-zcfnj   2/2
Running     0
nvidia-mig-manager-2fjlp                                1/1
Running     0
nvidia-mig-manager-2mrxp                                1/1
Running     0
nvidia-mig-manager-9khxv                                1/1
Running     0
nvidia-mig-manager-w4w59                                1/1
Running     0
nvidia-node-status-exporter-ms9wj                       1/1
Running     0
nvidia-node-status-exporter-ph56c                       1/1
Running     0
nvidia-node-status-exporter-v2q7w                       1/1
Running     0
nvidia-node-status-exporter-x54g7                       1/1
Running     0
nvidia-operator-validator-5zhpj                         1/1
Running     0
nvidia-operator-validator-9cdwx                         1/1
Running     0
nvidia-operator-validator-x87zb                         1/1
Running     0
nvidia-operator-validator-xklxj                         1/1
Running     0
```

Riva operator installation

To deploy the Riva API, perform the following steps on the OpenShift cluster.

1. Copy NGC API key from the NGC portal. To generate a key, follow the steps that are provided in the NVIDIA NGC User Guide.
2. On the workstation, run the following commands.

```
export NGC_CLI_API_KEY=<your NGC API key>
export VERSION_TAG="2.11.0"
helm fetch
https://helm.ngc.nvidia.com/nvidia/riva/charts/riva-api-
${VERSION_TAG}.tgz --username='$oauthtoken' --
```

```
password=$NGC_CLI_API_KEY tar -xvzf riva-api-
${VERSION_TAG}.tgz
```

3. After the riva-api folder is extracted, go to the `values.yaml` file in the folder and set asr,nlp, and tts to true or false as needed. In the context of this paper, all these values of asr,nlp and tts are set to true.
4. Change the PersistentVolumeClaim section in the `values.yaml` to point it to `vxflexos-xfs`. The storage class `vxflexos-xfs` is used by the Riva operator to provision the storage for the container. This storage contains all the dataset that includes audio files, and text files.

```
PersistentVolumeClaim:
  usePVC: true
  storageClassName: vxflexos-xfs
  storageAccessMode:
 storageSize: 1Ti
 artifactClaimName: riva-artifact-pvc
 workdirClaimName: riva-workdir-pvc
```

5. Change the `service.type` from `LoadBalancer` to `ClusterIP`.

   This change directly exposes the service only to other services within the cluster.
6. Enable the OpenShift cluster to run containers needing NVIDIA GPUs using the NVIDIA device plug-in.

```
helm repo add nvdp https://nvidia.github.io/k8s-device-
plugin
helm repo update
helm install --generate-name --set failOnInitError=false
nvdp/nvidia-device-plugin
```

7. Validate that the Nvidia device plug-in is running successfully on the cluster:

```
[root@ocp411-admin Riva]# oc get pods
NAME                                    READY   STATUS
RESTARTS   AGE
nvidia-device-plugin-1683609115-2rh8w   1/1     Running   0
3d6h
nvidia-device-plugin-1683609115-9x9tg   1/1     Running   0
3d6h
nvidia-device-plugin-1683609115-gm642   1/1     Running   0
3d6h
nvidia-device-plugin-1683609115-gpmtm   1/1     Running   0
3d6h
```

8. Install the Riva Helm Chart. You can explicitly override variables from the `values.yaml` file, such as the `riva.speechServices.[asr,nlp,tts]` settings.

```
helm install riva-api riva-api/ \
    --set ngcCredentials.password=`echo -n $NGC_CLI_API_KEY
| base64 -w0` \
    --set modelRepoGenerator.modelDeployKey=`echo -n
tlt_encode | base64 -w0` \
    --set riva.speechServices.asr=true \
    --set riva.speechServices.nlp=true \
    --set riva.speechServices.tts=true
```

9. The Helm chart runs two containers in order: A `riva-model-init` container that downloads and deploys the models, followed by a `riva-speech-api` container to start the speech service API.

   Note: Depending on the number of models, the initial model deployment could take an hour or more.

10. To monitor the deployment, use `kubectl` to describe the riva-api pod and to watch the container logs:

```
export pod=`kubectl get pods | cut -d " " -f 1 | grep riva-
api`
kubectl describe pod $pod
kubectl logs -f $pod -c riva-model-init
kubectl logs -f $pod -c riva-speech-api
```

11. With the Riva service is running, the cluster needs a mechanism to route requests into Riva. So, deploy the open-source Traefik edge router by running:

```
helm repo add traefik https://helm.traefik.io/traefik
helm repo update
helm fetch traefik/traefik
tar -zxvf traefik-*.tgz
```

12. Modify the `traefik/values.yaml` file to change `service.type` from `LoadBalancer` to `ClusterIP`. This change exposes the service on a cluster-internal IP.
13. Deploy the modified traefik Helm chart:

```
helm install traefik traefik/
```

14. An IngressRoute enables the Traefik load balancer to recognize incoming requests and distribute them across multiple riva-api services. When traefik Helm chart above is deployed, Kubernetes automatically creates a local DNS entry for that service named as `traefik.default.svc.cluster.local`. The IngressRoute definition in the following steps matches these DNS entries and directs requests to the riva-api service.
15. Create the following `riva-ingress.yaml` file:

```
apiVersion: traefik.containo.us/v1alpha1
kind: IngressRoute
metadata:
  name: riva-ingressroute
spec:
  entryPoints:
    - web
  routes:
    - match: "Host(`traefik.default.svc.cluster.local`)"
      kind: Rule
      services:
        - name: riva-api
          port: 50051
          scheme: h2c
```

16. Deploy the IngressRoute by running:

```
Kubectl apply -f riva-ingress.yaml
```

The Riva service is now able to serve gRPC requests from within the cluster at the address `traefik.default.svc.cluster.local.`

Chapter 5 Validation and testing

This chapter presents the following topics:

Overview

This chapter provides a detailed description and summary of the tests that the PowerFlex engineering team performed to validate the NVIDIA Riva services on OpenShift running in a PowerFlex environment. To measure the performance, the team recorded latency and throughput numbers for Riva ASR service and Riva TTS.

Test methodology

Speech recognition in Riva is a GPU-accelerated compute pipeline with optimized performance and accuracy. Riva supports offline/batch and streaming recognition modes.

Automatic Speech Recognition (ASR) takes an audio stream or audio buffer as input and returns one or more text transcripts, along with additional optional metadata.

The text-to-speech (TTS) pipeline that is implemented for the Riva TTS service is based on a two-stage pipeline. Riva first generates a mel-spectrogram using the first model, and then generates speech using the second model. This pipeline forms a TTS system that enables you to synthesize natural sounding speech from raw transcripts without any additional information such as patterns or rhythms of speech.

For this paper, the PowerFlex engineering team chose the most common use cases of Riva ASR and Riva TTS along with basic performance tests were chosen to demonstrate that the PowerFlex family is well suited for NVIDIA A100 GPUs on Red Hat OpenShift environment.

For Riva ASR, they considered the following use cases:

- Conversion of an audio file to a text file.
- Performance tests for Riva ASR using the LibriSpeech dataset. LibriSpeech is a corpus of approximately 1000 hours of 16 kHz read English speech. The LibriSpeech datasets are available at the Open SLR website.

For Riva TTS, they considered the following use cases:

- Conversion of a text to an audio file.
- Performance tests for Riva TTS using the standard text file.

Test setup

After the Riva operator is installed along with the Riva Trafeik edge router, Riva provides a container with a set of prebuilt sample clients to test the Riva services.

To deploy the Riva client:

1. Create the `client-deployment.yaml` file that defines the deployment and contains the following code:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: riva-client
```

```
      labels:
        app: "rivaasrclient"
    spec:
      replicas: 1
      selector:
        matchLabels:
          app: "rivaasrclient"
      template:
        metadata:
          labels:
            app: "rivaasrclient"
        spec:
          imagePullSecrets:
          - name: imagepullsecret
          containers:
          - name: riva-client
            image: "nvcr.io/nvidia/riva/riva-speech-
    client:2.11.0"
            command: ["/bin/bash"]
            args: ["-c", "while true; do sleep 5; done"]
```

2. Deploy the Riva client service:

```
kubectl apply -f client-deployment.yaml
export cpod=`kubectl get pods | cut -d " " -f 1 | grep riva-
client`
kubectl exec --stdin --tty $cpod /bin/bash
```

3. Validate that the Riva client service is deployed by confirming that the riva-client pod is running successfully:

```
[root@ocp411-admin Riva]# oc get all
NAME                                            READY   STATUS
RESTARTS   AGE
pod/nvidia-device-plugin-1683609115-2rh8w   1/1     Running   0
3d7h
pod/nvidia-device-plugin-1683609115-9x9tg   1/1     Running   0
3d7h
pod/nvidia-device-plugin-1683609115-gm642   1/1     Running   0
3d7h
pod/nvidia-device-plugin-1683609115-gpmtm   1/1     Running   0
3d7h
pod/riva-client-668dd7594b-cr68q            1/1     Running   0
2d7h
pod/riva-riva-api-7d5b75687b-4t6kn          1/1     Running   0
2d6h
pod/traefik-6fbf57555d-xw82v                1/1     Running   0
2d8h
```

```
NAME                        TYPE           CLUSTER-IP
EXTERNAL-IP                                PORT(S)
AGE
service/kubernetes          ClusterIP      172.30.0.1       <none>
443/TCP                                    4d6h
service/openshift           ExternalName   <none>
kubernetes.default.svc.cluster.local       <none>
4d6h
service/riva-riva-api       ClusterIP      172.30.6.226     <none>
8000/TCP,8001/TCP,8002/TCP,50051/TCP       2d6h
service/traefik             ClusterIP      172.30.246.217   <none>
80/TCP,443/TCP                             2d8h

NAME                                               DESIRED
CURRENT    READY    UP-TO-DATE    AVAILABLE    NODE SELECTOR    AGE
daemonset.apps/nvidia-device-plugin-1683609115    4          4
4        4             4            <none>          3d7h

NAME                              READY    UP-TO-DATE    AVAILABLE
AGE
deployment.apps/riva-client       1/1      1             1
2d7h
deployment.apps/riva-riva-api     1/1      1             1
2d6h
deployment.apps/traefik           1/1      1             1
2d8h

NAME                                          DESIRED    CURRENT
READY    AGE
replicaset.apps/riva-client-668dd7594b        1          1          1
2d7h
replicaset.apps/riva-riva-api-7d5b75687b      1          1          1
2d6h
replicaset.apps/traefik-6fbf57555d            1          1          1
2d8h
```

The Riva client pod is now ready for use to test Riva ASR and Riva TTS services.

Test validation

The Riva services are validated by running the Riva ASR and Riva TTS tests.

Riva ASR workload

To run the Riva ASR test, start an interactive session with Riva client container, and run the following command. This command takes in the sample .wav file and converts it into text. The same .wav file is already present in the Riva client container.

```
[root@ocp411-admin Riva]# kubectl exec --stdin --tty $cpod
/bin/bash
root@riva-client-668dd7594b-cr68q:/opt/riva#
riva_streaming_asr_client \
>    --audio_file=wav/en-US_sample.wav \
>    --automatic_punctuation=true \
>    --riva_uri=traefik.default.svc.cluster.local:80
```

Riva ASR service converts the .wav file into text as follows:

```
I0512 13:00:14.664886 47228 riva_streaming_asr_client.cc:150]
Using Insecure Server Credentials
Loading eval dataset...
filename: /opt/riva/wav/en-US_sample.wav
Done loading 1 files
what
what
what is
what is
what is
what is now
what is natural
what is natural
what is natural language
what is natural language
what is natural language
what is natural language
what is natural language Processing
what is natural language Processing
what is natural language Processing
what is natural language Processing
what is natural language Processing
what is language Processing
what is language Processing
What is Natural Language Processing?
------------------------------------------------------------
File: /opt/riva/wav/en-US_sample.wav

Final transcripts:
0 : What is Natural Language Processing?

Timestamps:
Word                                    Start (ms)      End (ms)

What                                    840             880
is                                      1160            1200
Natural                                 1800            2080
Language                                2200            2520
Processing?                             2720            3200

Audio processed: 4 sec.
------------------------------------------------------------
```

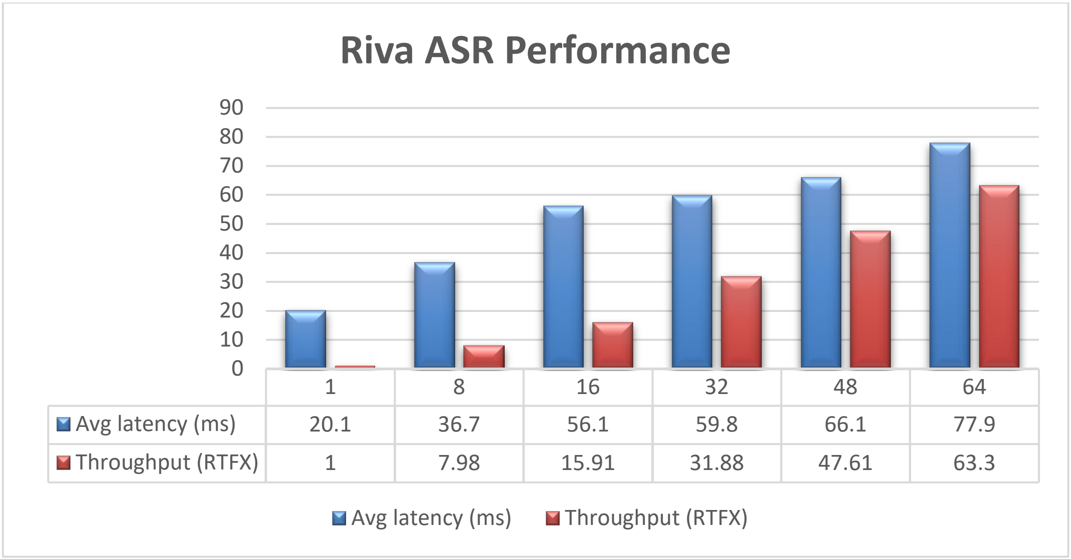
```
Not printing latency statistics because the client is run without
the --simulate_realtime option and/or the number of requests sent
is not equal to number of requests received. To get latency
statistics, run with --simulate_realtime and set the --
chunk_duration_ms to be the same as the server chunk duration
Run time: 0.1486 sec.
Total audio processed: 4.152 sec.
Throughput: 27.9407 RTFX
```

Next, performance testing of Riva ASR service (low latency) is carried out. The Riva streaming client `riva_streaming_asr_client`, which is provided in the Riva image, is used with the `simulate_realtime` flag to simulate the transcription from a microphone, where each stream was doing three iterations over a sample audio file (1272-135031-0000.wav) from the LibriSpeech dev-clean dataset. The Librispeech datasets are available from Open SLR.

The command used to measure validation is:

```
riva_streaming_asr_client \
   --chunk_duration_ms=<chunk_duration> \
   --simulate_realtime=true \
   --automatic_punctuation=true \
   --num_parallel_requests=<num_streams> \
   --word_time_offsets=true \
   --print_transcripts=false \
   --interim_results=false \
   --num_iterations=<3*num_streams> \
   --audio_file=1272-135031-0000.wav \
   --output_filename=/tmp/output.json
```

The preceding command is run for various streams starting with 1. Then, for each run, the stream is changed to 8,16, 32, 48, and 64 as shown in the following graph. During the performance test, the average latency is calculated along with the throughput.

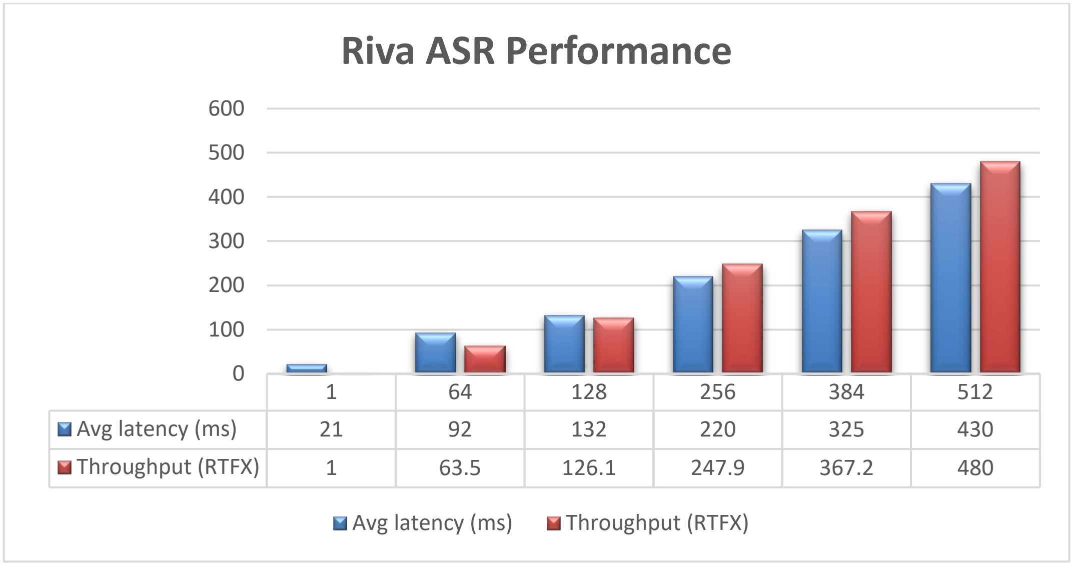


	1	8	16	32	48	64
🟦 Avg latency (ms)	20.1	36.7	56.1	59.8	66.1	77.9
🟥 Throughput (RTFX)	1	7.98	15.91	31.88	47.61	63.3

Figure 7. Riva ASR performance (low latency)

To measure the performance of the GPU on the PowerFlex CO node, the NVIDIA GPU administration dashboard is enabled. The OpenShift Console NVIDIA GPU plug-in is a dedicated administration dashboard for visualizing the NVIDIA GPU usage in the OpenShift Container Platform (OCP) console.

To get the streaming and high throughput numbers for Riva ASR, the chunk size (ms) is kept at 900 and the maximum effective number of streams without language model (greedy generation) is set to 750. For each run, the stream is changed to 1,64, 128, 256, 384, and 512 as shown in the following graph. During the performance test, the average latency is calculated along with the throughput.



	1	64	128	256	384	512
🟦 Avg latency (ms)	21	92	132	220	325	430
🟥 Throughput (RTFX)	1	63.5	126.1	247.9	367.2	480

Figure 8. Riva ASR performance (high throughput)

Note: During the test of every stream, the GPU utilization was about 48 percent. No PowerFlex storage bottlenecks were seen anywhere during the testing.

Riva TTS workload

To run the Riva ASR test, start an interactive session with Riva client container, and run the following command:

```
root@riva-client-668dd7594b-cr68q:/opt/riva#
riva_tts_client --text="PowerFlex software-defined infrastructure
enables broad consolidation across the data center, encompassing
almost any type of workload and architecture."
--audio_file =/opt/riva/wav/output.wav --voice_name=English-
US.Male-1  --riva_uri=traefik.default.svc.cluster.local:80
```

Riva TTS service converts the text into a .wav file which is published at `/opt/riva/wav/output.wav.`

```
I0525 04:23:41.519158 265654 riva_tts_client.cc:99] Using Insecure
Server Credentials
Request time: 0.261046 s
```

```
Got 776704 bytes back from server
```

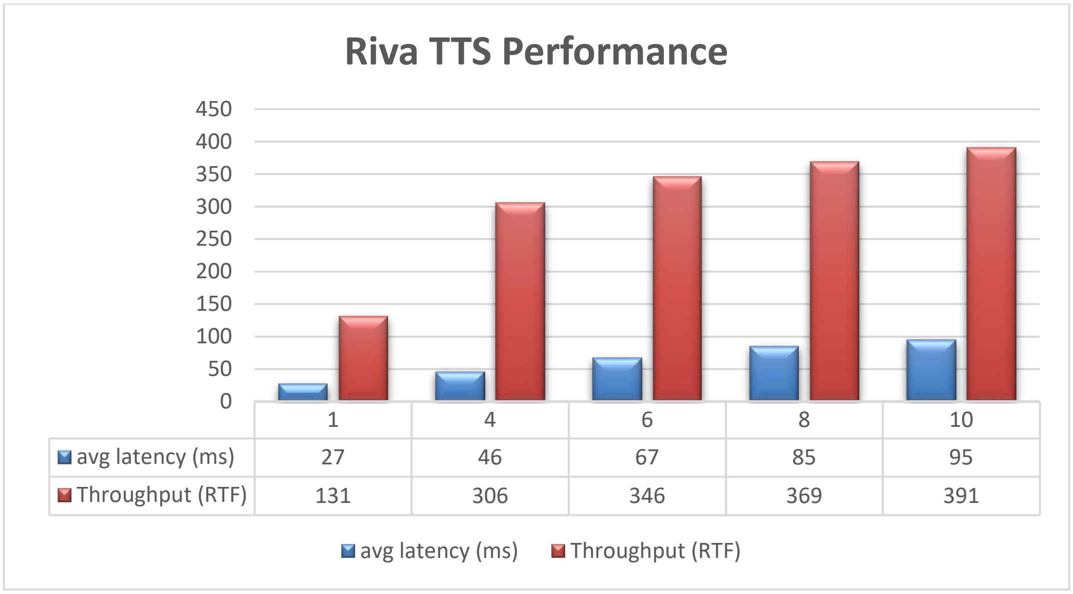
The testing team carried out performance testing on the Riva TTS service. The Riva TTS performance client `riva_tts_perf_client`, which is provided in the Riva image, measures the performance.

The following command is used to measure validation:

```
riva_tts_perf_client \
    --num_parallel_requests=<num_streams> \
    --voice_name=English-US.Female-1 \
    --num_iterations=<20*num_streams> \
    --online=true \
    --text_file=$test_file \
    --write_output_audio=false
```

Where `test_file` is a path to the `ljs_audio_text_test_filelist_small.txt` file.

The preceding command is run for various streams starting with 1. Then, for each run, the stream is changed to 4, 6, 8, and 10, as shown in the following graph. During the performance test, the average latency is calculated along with the throughput.



	1	4	6	8	10
🟦 avg latency (ms)	27	46	67	85	95
🟥 Throughput (RTF)	131	306	346	369	391

Figure 9. Riva TTS performance

To measure the performance of GPU on the PowerFlex CO node, the [NVIDIA GPU administration]() dashboard is enabled. The OpenShift Console NVIDIA GPU plug-in is a dedicated administration dashboard for NVIDIA GPU usage visualization in the OpenShift Container Platform (OCP) console.

Note: During the test for every stream, the GPU utilization observed was approximately 82 percent. No PowerFlex storage bottlenecks were seen anywhere during the testing.

Chapter 6 Conclusion

This chapter presents the following topics:

Summary

Dell Technologies and NVIDIA have worked together on numerous advancements in computation, data storage options, and high-speed networking. This document describes how to efficiently deploy and test NVIDIA Riva services with Red Hat OpenShift on a two-layer PowerFlex system efficiently.

The solution also shows the validation of NVIDIA ASR and NVIDIA TTS services running on different input streams. Throughout the tests, no bottlenecks were seen with the PowerFlex storage, as the text results show.

Chapter 7 References

This chapter presents the following topics:

Dell Technologies documentation

The following Dell Technologies documentation provides additional information. Access to these documents depends on your login credentials. If you do not have access to a document, contact your Dell Technologies representative.

- PowerFlex Software documentation
- Dell PowerFlex
- Virtualizing GPUs for AI with VMware and NVIDIA Based on Dell Infrastructure
  - White paper
  - Design Guide
  - Implementation Guide

Red Hat OpenShift documentation

The following Red Hat documentation provides additional information:

- Red Hat OpenShift Assisted Installer
- Red Hat OpenShift on PowerFlex

NVIDIA documentation

The following NVIDIA documentation provides additional information:

- Riva Overview
- NVIDIA GPU Operator
- Audio Transcription and Intelligent Virtual Assistants

Appendix A Configuration details

This appendix presents the following topics:

PowerFlex compute-only nodes

The following table shows the hardware and configuration details of the PowerFlex compute-only nodes:

Table 3. PowerFlex compute-only nodes

Hardware	Configuration
CPU cores	2 x Intel(R) Xeon(R) Gold 6150 @ 2.70 GHz with 18 cores per socket
Memory	6 x 64 GB
NIC	NIC Slot 2: Mellanox ConnectX-4 LX 25 GbE SFP Adapter 2 Ports NIC Slot 3: Mellanox ConnectX-4 LX 25 GbE SFP Adapter 2 Ports
Storage	BOSS-S1: 2 x 223 GB SATA SSD 8 x 894 GB SAS SSD
Operating system	RHCOS 8.7
GPU	A100 40 GB, each at slot 7.8

PowerFlex storage-only nodes

The following table shows the hardware and configuration details of the PowerFlex storage-only nodes:

Table 4. PowerFlex storage-only nodes

Hardware	Configuration
CPU cores	2 x Intel(R) Xeon(R) Gold 6248 CPU @ 2.50 GHz
Memory	12 x 64 GB
NIC	NIC Slot 2: Mellanox ConnectX-4 LX 25 GbE SFP Adapter 2 Ports NIC Slot 3: Mellanox ConnectX-4 LX 25 GbE SFP Adapter 2 Ports
Storage	BOSS-S1: 2 x 223 GB SATA SSD 8 x 894 GB SAS SSD
PowerFlex	4.0.1

Red Hat OpenShift master nodes

The following table shows the hardware and configuration details of the VM:

Table 5. Master node configuration

Hardware	Configuration
Master Nodes	3
CPU cores	2 x Intel(R) Xeon(R) Gold 6150 @ 2.70 GHz with 18 cores per socket
Memory	6 x 64 GB
Storage	BOSS-S1: 2 x 223 GB SATA SSD 8 x 894 GB SAS SSD

Network bond files

The following code extract is from a sample bond file that is used for the master and worker node of OpenShift.

```
[root@ocp411-admin Scripts]# cat master0-bond1-vlans.yaml
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: master0-bond1-vlans
spec:
  nodeSelector:
    kubernetes.io/hostname: "node1.ocp411.gpuflex.com"
  desiredState:
    interfaces:
    - name: ens2f1
      description: Ethernet ens2f1
      type: ethernet
      state: up
      ipv4:
        enabled: false
      mtu: 9000
    - name: ens3f1
      description: Ethernet eno2
      type: ethernet
      state: up
      ipv4:
        enabled: false
      mtu: 9000
    - name: bond1
      description: Bond using ports ens2f1 and ens3f1
      type: bond
      state: up
      ipv4:
```

```
      enabled: false
    link-aggregation:
      mode: 802.3ad
      options:
        miimon: "100"
      port:
        - ens2f1
        - ens3f1
    mtu: 9000
  - name: bond1.150
    description: PowerFlex Mgmt VLAN150 using bond1
    type: vlan
    state: up
    ipv4:
      address:
        - ip: 192.168.150.114
          prefix-length: 24
      enabled: true
    vlan:
      base-iface: bond1
      id: 150
  - name: bond1.152
    description: PowerFlex Data network using bond1
    type: vlan
    state: up
    ipv4:
      address:
        - ip: 192.168.152.114
          prefix-length: 24
      enabled: true
    vlan:
      base-iface: bond1
      id: 152
    mtu: 9000
  - name: bond1.154
    description: PowerFlex Data B VLAN152 using bond1
    type: vlan
    state: up
    ipv4:
      address:
        - ip: 192.168.154.114
          prefix-length: 24
      enabled: true
    vlan:
      base-iface: bond1
      id: 154
    mtu: 9000
```