

Artificial Intelligence using Apache Spark on Hitachi Unified Compute Platform with Hitachi Content Software for File

Reference Architecture Guide

© 2024 Hitachi Vantara LLC. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including copying and recording, or stored in a database or retrieval system for commercial purposes without the express written permission of Hitachi, Ltd., or Hitachi Vantara LLC (collectively "Hitachi"). Licensee may make copies of the Materials provided that any such copy is: (i) created as an essential step in utilization of the Software as licensed and is used in no other manner; or (ii) used for archival purposes. Licensee may not make any other copies of the Materials. "Materials" mean text, data, photographs, graphics, audio, video and documents.

Hitachi reserves the right to make changes to this Material at any time without notice and assumes no responsibility for its use. The Materials contain the most current information available at the time of publication.

Some of the features described in the Materials might not be currently available. Refer to the most recent product announcement for information about feature and product availability, or contact Hitachi Vantara LLC at https://support.hitachivantara.com/en_us/contact-us.html.

Notice: Hitachi products and services can be ordered only under the terms and conditions of the applicable Hitachi agreements. The use of Hitachi products is governed by the terms of your agreements with Hitachi Vantara LLC.

By using this software, you agree that you are responsible for:

1. Acquiring the relevant consents as may be required under local privacy laws or otherwise from authorized employees and other individuals; and
2. Verifying that your data continues to be held, retrieved, deleted, or otherwise processed in accordance with relevant laws.

Notice on Export Controls. The technical data and technology inherent in this Document may be subject to U.S. export control laws, including the U.S. Export Administration Act and its associated regulations, and may be subject to export or import regulations in other countries. Reader agrees to comply strictly with all such regulations and acknowledges that Reader has the responsibility to obtain licenses to export, re-export, or import the Document and any Compliant Products.

Hitachi and Lumada are trademarks or registered trademarks of Hitachi, Ltd., in the United States and other countries.

AIX, AS/400e, DB2, Domino, DS6000, DS8000, Enterprise Storage Server, eServer, FICON, FlashCopy, GDPS, HyperSwap, IBM, Lotus, MVS, OS/390, PowerHA, PowerPC, RS/6000, S/390, System z9, System z10, Tivoli, z/OS, z9, z10, z13, z14, z15, z16, z/VM, and z/VSE are registered trademarks or trademarks of International Business Machines Corporation.

Active Directory, ActiveX, Bing, Excel, Hyper-V, Internet Explorer, the Internet Explorer logo, Microsoft, Microsoft Edge, the Microsoft corporate logo, the Microsoft Edge logo, MS-DOS, Outlook, PowerPoint, SharePoint, Silverlight, SmartScreen, SQL Server, Visual Basic, Visual C++, Visual Studio, Windows, the Windows logo, Windows Azure, Windows PowerShell, Windows Server, the Windows start button, and Windows Vista are registered trademarks or trademarks of Microsoft Corporation. Microsoft product screen shots are reprinted with permission from Microsoft Corporation.

All other trademarks, service marks, and company names in this document or website are properties of their respective owners.

Copyright and license information for third-party and open source software used in Hitachi Vantara products can be found in the product documentation, at <https://www.hitachivantara.com/en-us/company/legal.html> or https://knowledge.hitachivantara.com/Documents/Open_Source_Software.

Feedback

Hitachi Vantara welcomes your feedback. Please share your thoughts by sending an email message to SolutionLab@HitachiVantara.com. To assist the routing of this message, use the paper number in the subject and the title of this white paper in the text.

Revision history

Changes	Date
Updated for GPUs	April 2024
Initial release	January 2024

Contents

Reference Architecture Guide.....	4
Introduction.....	7
Business use cases.....	10
Solution benefits.....	12
Solution overview.....	15
Data management.....	21
Solution architecture.....	22
Solution components.....	24
Sizing considerations.....	28
Advanced Analytics (AI/ML).....	28
Generative AI.....	32
Methodology of solution validation and testing.....	41
Solution validation.....	41
AI/GenAI with GPUs.....	42
Install GPU operators on K8S clusters.....	44
Install RAPID Accelerator for Apache Spark.....	45
Traditional Analytics without GPUs.....	46
Hitachi Content Software for File installation instructions.....	46
Attach a local Object store.....	47
Create a filesystem on Hitachi Content Software for File storage.....	47
Install the HCSF Client.....	48
Mount the Filesystem.....	49
Configure the HCSF CSI plugin.....	50
Configure a Secret for StorageClass.....	50
Create a StorageClass.....	51
Create a Persistent Volume Claim.....	52
Configure Spark Clusters.....	53
Solution testing.....	55
AI/GenAI with GPUs (using K8S).....	55
Test Spark Job with GPUs vs CPUs on the K8S Cluster.....	55
Traditional Analytics without GPUs (using OCP).....	58
Run Spark Jobs.....	58
Spark Cluster Scaling.....	59
Conclusion.....	59

Reference Architecture Guide

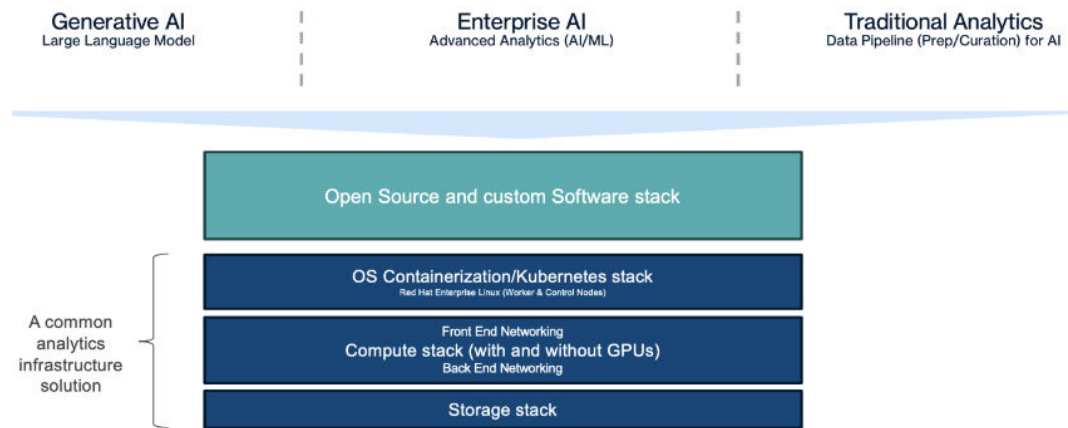
Executive summary

There is an insatiable demand for Analytics solutions using Artificial Intelligence (AI) to extract meaningful insights from the information available within and outside an organization.

Broadly categorized, these analytics solutions comprise the following:

- Advanced Analytics (AI/ML) solutions, that empower enterprises with predictive, prescriptive, and cognitive insights by running AI workloads on accelerated compute and on large amounts of data.
- Generative AI solutions, which have taken center stage recently, enable enterprises to generate original content by recognizing patterns in existing data. These types of solutions need a much larger scale of accelerated compute and extremely large amounts of data.
- Traditional Analytics solutions, available for over decades, typically involve analysis of historical data to identify trends and build reports. These types of solutions need fast compute and large amounts of data.
- Edge Analytics solutions, which have become increasingly important because of the explosion of data from IOT and edge devices, the need for real-time analysis and bandwidth constraints, and executing AI-enabled applications. These analytics solutions are smaller-scale, enterprise-grade edge infrastructure, that may connect to numerous outdoor ruggedized devices/sensors or connect to analytics running in larger data center infrastructure. These types of solutions have small footprints but require accelerated compute.

The accelerating demand for analytics solutions reflects the competitive landscape within which enterprises are striving to gain an advantage. The surging volume, ever-increasing velocity, and diverse variety of data highlight the need for analytics solutions that are robust, highly adaptable, and can operate effectively with this explosive increase in data. This reference architecture serves as an implementation guide for Advanced Analytics (AI/ML), Generative AI and Traditional Analytics solutions. It strives to provide a framework for efficient AI/ML, including Generative AI, enabling enterprises to be at the forefront of data-driven decision-making.



The reference architecture (RA) offers significant benefits to the business. The solution delivers accelerated time to value and empowers customers to execute analytics workloads at accelerated speed, facilitating quicker decision-making and insight.

The RA delivers a foundational reference implementation and has been validated to drive infrastructure operational efficiency, provides accelerated compute powered by NVIDIA GPUs with parallel high-performance Hitachi Content Software for File (HCSF) storage paired with high-capacity Hitachi Content Platform (HCP) object storage to minimize complex computational processing, and ships as a pre-integrated system for rapid installation, setup, and user access.

The reference architecture provides flexible infrastructure with seamless deployment options for on-premises and near-cloud colocation facilities, ensuring data is secured within organizational boundaries and enabling compliance with regulations such as HIPAA and GDPR. With independently scaling compute and storage, customer can start with minimal investments and rapidly evolve as demands on infrastructure grow due to market dynamics. The diverse choice of hardware and software components provides added flexibility to tailor infrastructure to specific workload requirements without being constrained to pre-defined configurations.

The integrated UCP and HCSF architecture, coupled with HCP for tiered storage, enables cost effective performance for customers by efficiently storing and rapidly accessing vast amounts of unstructured data at lower costs. This facilitates increased utilization of GPUs for demanding workloads.

Hitachi Vantara delivers consolidated enterprise support for its solutions, backed by leaders like NVIDIA and Red Hat. It provides swift issue resolution bound by SLAs, simplifying customer engagement with a single point of contact for diverse infrastructure components.

This document introduces Advanced Analytics (AI/ML), Generative AI, and Traditional Analytics, followed by how these applications are intertwined for solving a business problem for selected industries, along with business benefits. This document describes key solution components associated with the solution architecture. Sizing considerations are provided to scale the infrastructure to meet the operational requirements of the customer. And finally, the operational capabilities of the solution are validated.

Analytics Workload Type	Data Preparation and Curation	Model Training	Fine-Tuning	Retrieval Augmented Generation (RAG)	Inferencing
Advance Analytics (AI/ML)	Yes	Yes	Yes	Not applicable	Yes
Generative AI	Yes	No	Yes	Yes	Yes
Traditional Analytics	Yes	Not applicable	Not applicable	Not applicable	Not applicable
Edge Analytics	Yes	No	No	Not applicable	Yes

Hereafter in this document, Edge Analytics will be implicitly covered as a smaller scale version of the key points mentioned in this document for Advanced Analytics (AI/ML), Generative AI, and Traditional Analytics.

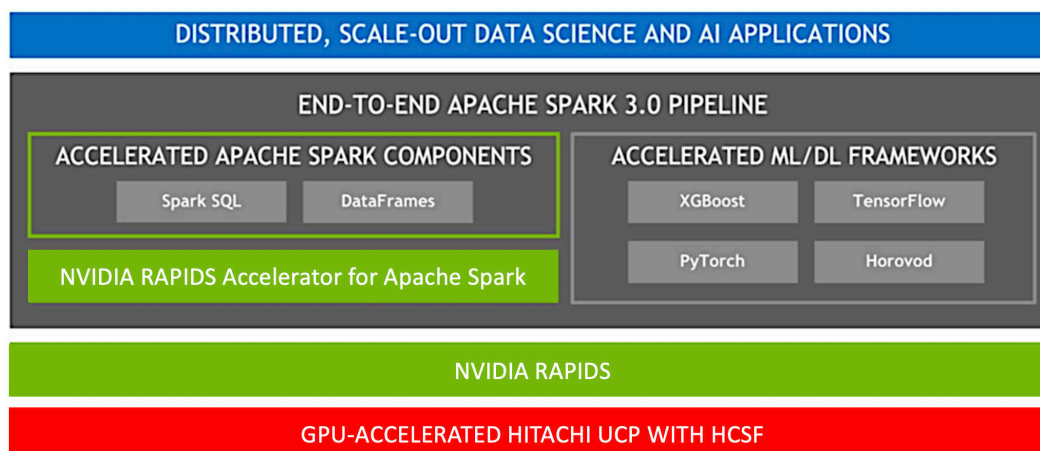
This reference architecture can be used to build an end-to-end architecture for Advanced Analytics (AI/ML) solutions using Apache Spark – a popular unified engine for large-scale analytics. It addresses key challenges for analytics solutions, such as how to put together accelerated compute with massively parallel distributed storage and how to build an independently scalable architecture.

This same architecture seamlessly extends to handle Generative AI workloads such as fine-tuning and inferencing. It is an independently scalable design, coupled with innovative computing powered with NVIDIA GPUs, massively parallel distributed storage capabilities, and high-performance networking, empowering organizations to fine-tune Generative AI models easily with petabyte-scale datasets. Furthermore, this architecture excels in executing inference workloads at very high speed.

The architecture can also manage Traditional Analytics workloads to uncover patterns in historical data. Apache Spark can curate structured data sources, employ statistical methods, and utilize SQL query and reporting tools to deliver insights for decision-making.

The example outlined in this reference architecture demonstrates how to process unstructured batch data using Apache Spark, one of the many options for Artificial Intelligence analytics. The example can be extended to process structured and semi-structured datasets for streaming and batch processing. The architecture uses popular containerization platforms – Kubernetes and Red Hat OpenShift. Using these containerized platforms, other popular distributed compute engines such as Apache Flink, Dask, Apache Arrow, or Apache Ray can be deployed.

Furthermore, using NVIDIA GPUs and optimized software, this reference architecture can deliver accelerated data analytics.



NVIDIA RAPIDS Accelerator for Apache Spark seamlessly integrate GPU acceleration into the Spark ecosystem. These RAPIDS Accelerators enhance data processing speed and scalability. This allows data engineers and data scientists to efficiently analyze large-scale datasets using GPUs. NVIDIA RAPIDS also accelerate existing popular frameworks such as Google's TensorFlow and Meta's PyTorch.

Intended audience

This paper documents a reference architecture using Apache Spark on Unified Compute Platform (UCP) with Hitachi Content Software for File (HCSF) high-performance storage. The document is intended for customers, solutions architects, services consultants, pre-sales engineers, and subject matter experts who are interested in implementing artificial intelligence, machine learning, Generative AI, and data analytics.



Note: Testing of this configuration was in a lab environment. Many factors affect production environments beyond prediction or duplication in a lab environment. Follow the recommended practice of conducting proof-of-concept testing for acceptable results in a non-production, isolated test environment that otherwise matches your production environment before implementing this solution in your production environment.

Introduction

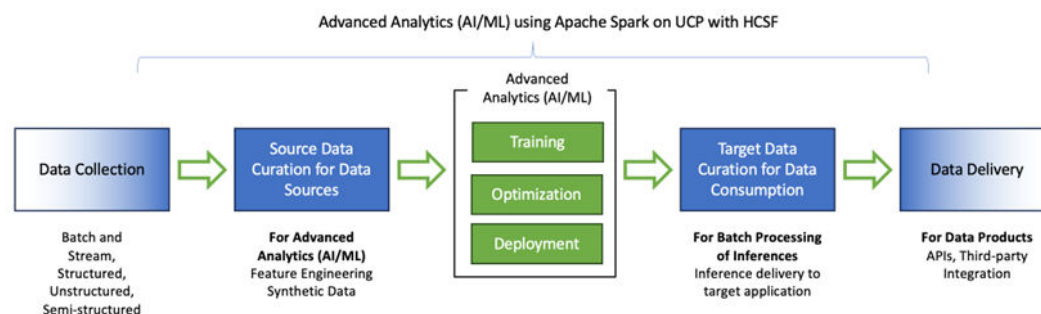
Enterprises are investing to gain a better understanding of their business by using the exponential growth and complexity in data available to them. Enterprises have a strong desire to analyze this data and create new AI-enabled applications or integrate AI as an embedded feature into their existing applications/offerings to gain competitive advantage. To analyze the data and generate insights from it, enterprises need an independently scalable infrastructure that has high-performance versatile compute, massively parallel distributed storage, and non-blocking high bandwidth networks.

Without access to such infrastructure, processing time-sensitive data becomes challenging, leading to the delivery of unusable insights that hinder informed decision-making and impede organizations' objectives and service level agreements, ultimately resulting in businesses losing their competitive advantage.

The advent of new-age algorithms, such as large language models (LLM), large diffusion models (LDM), and code generation models (CGM), now used in various AI models and analytics applications further heightens the need for high-speed data access and advanced computation using technology such as graphics processing units (GPUs). These workloads span artificial intelligence model building and deployment, Generative AI model training, fine-tuning, and inferencing, and then finally data collection, processing, and transformation workloads.

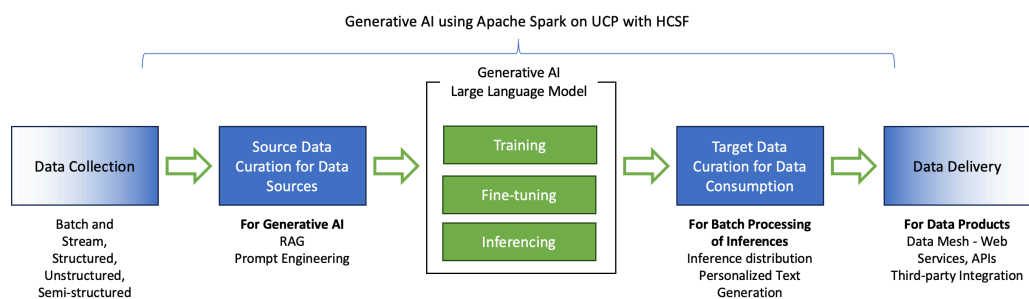
The solution described in this RA can address the following three overarching categories of analytics applications and their corresponding data pipelines:

- Advanced Analytics (AI/ML)** — Predictive analytics, prescriptive analytics and cognitive analytics are some of the common analytics applications under this category. Predictive analytics utilizes historical data and statistical algorithms to foresee events like churn predictions, preserving the data privacy by keeping sensitive data within your network. Prescriptive analytics goes further, offering optimized strategies, employing algorithms for tasks like inventory managements. Cognitive analytics integrates human-like cognitive capabilities, contextualizing predictions, and prescriptions to solve problems effectively.

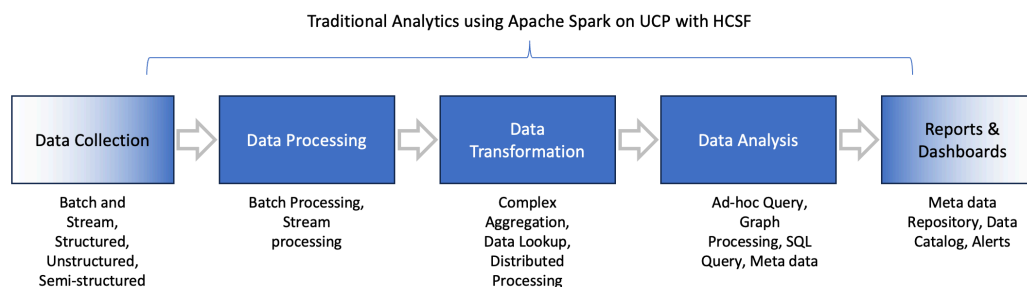


The end-to-end data pipeline as captured below involves data collection, preparation, transformation, and analysis. This pipeline typically involves several stages, each with its own set of tools, technologies, and processes.

- Generative AI** — Recent breakthroughs in algorithms, often referred to as “Generative AI,” have ushered in a new era of autonomous learning and corrective mechanisms with diverse applications across a variety of domains. These algorithms produce content that often rivals human-generated content in terms of originality, creativity, and refinement. The data pipelines to handle Generative AI involve using very advanced AI algorithms and complex models to create new content or data that resembles patterns observed in training data. The following example pipeline shows steps to collect and curate the data, train, fine-tune, and inference from the model and finally prepare the data for consumption.



- Traditional Analytics — Business intelligence and data warehousing are the two main components of traditional analytics. Business intelligence revolves around dashboards and reports while data warehousing consolidates the data from various sources for easy access and analysis. Traditional analytics pipeline involves a series of stages and processes as captured by the image below to transform raw data from the sources into valuable insights that business can consume on reports and dashboards for decision-making.

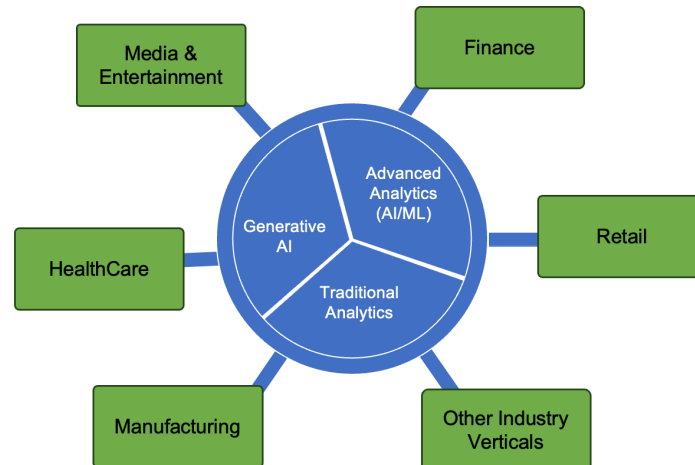


For the applications and pipelines discussed above, Apache Spark is used as a distributed compute engine. It is a flexible, general-purpose analytics framework that provides high-level APIs in Java, Scala, Python and R. It has a rich set of higher-level tools for data processing, data analysis, machine learning, graph processing and stream processing. Apache Spark seamlessly integrates with Google's TensorFlow and Meta's PyTorch deep learning frameworks. The NVIDIA RAPIDS Accelerator for Apache Spark leverages GPUs to accelerate data processing. The integration between high-performance data processing of RAPIDS cuDF library with the expansive scalability of the Spark distributed computing framework, empowers enterprises with accelerated AI processing workflows and pipelines.

Business use cases

This versatile solution is tailored for seamless integration across diverse industry verticals and various business functions, offering an array of operational, financial, and strategic advantages. By combining previously outlined high-level use cases, this solution can address a multitude of business challenges across different industry verticals.

Even though a thorough description and comprehensive inventory of all solutions are beyond the scope of this reference architecture, some industry verticals and use cases are covered to provide ideas about what is possible.



Manufacturing

In manufacturing AI and analytics have become integral for optimizing production processes and enhancing overall efficiency.

- Advanced Analytics (AI/ML) applications use historical and real-time data to identify patterns, predict equipment failures, and optimize supply chain logistics. AI automates quality control, predictive maintenance, and demand forecasting, reducing production downtime and costs while improving product quality.
- Generative AI is leveraged to create innovative designs, optimize manufacturing workflows, and generate production-ready CAD models, enabling manufacturers to stay competitive in a rapidly evolving industry.

Media and entertainment

Media companies use AI to gather insights about viewer behavior, content preferences, and advertising effectiveness. This data-driven approach allows them to optimize content recommendations, scheduling, and marketing strategies, leading to increased viewer engagement and revenue.

- Advanced Analytics (AI/ML) powers content recommendation engines that personalize user experiences, enhance content discovery, and automate tasks such as content tagging and moderation. AI algorithms also enable the analysis of vast datasets to identify trends, predict audience reactions, and improve production and distribution efficiency.
- Generative AI is employed to create compelling content. This technology enhances content creation, special effects, and storytelling possibilities, expanding the creative boundaries of the industry.

Together, these technologies enable media and entertainment companies to better understand their audiences, create more engaging and personalized content, optimize operations, and unlock new avenues for creativity and innovation in the rapidly evolving digital landscape.

Financial services

In the banking, financial services, and insurance (BFSI) sector, AI is transforming operations and customer experiences. They are used to gain insights from vast financial data, enhancing risk assessment, fraud detection, and customer segmentation.

- Advanced Analytics (AI/ML) algorithms power chatbots for customer support, algorithmic trading, and credit risk assessment, automating decision-making processes, and improving customer service.
- Generative AI helps automate document generation, streamline compliance reporting, and create personalized financial advice.

Collectively, these technologies enable BFSI institutions to optimize operations, reduce costs, and provide more tailored services to their clients.

Healthcare

AI is increasingly integrated into healthcare to analyze patient data, optimize treatment plans, and improve operational efficiency. Additional advanced usage includes robot-assisted surgery.

- Advanced Analytics (AI/ML) algorithms can process vast amounts of medical data to predict disease risks, personalize treatments, and enhance medical imaging for more accurate diagnostics.
- Generative AI is used to generate synthetic medical images for testing, assist in drug discovery, and simulate patient data to aid in medical research and training, further advancing healthcare capabilities.

Retail

In the retail industry, AI is employed to optimize inventory management, enhance customer experiences, and personalize marketing efforts.

- Advanced Analytics (AI/ML) algorithms analyze historical sales data to forecast demand, ensuring that retailers maintain the right inventory levels, minimize stockouts, and reduce excess inventory costs.
- Generative AI is used to create compelling product descriptions, design marketing materials, and even generate visual content, enabling retailers to engage customers with dynamic and appealing content.

Solution benefits

The solution described in this reference architecture offers significant benefits to the businesses. The primary benefits follow.

Accelerated time to value:

Accelerated time to value is critical for customers; they want to quickly extract value from their AI initiatives. Customers want outcomes like improved decision-making, operational efficiency, employee productivity, customer and employee experience, and automation of complex processes and workflows.

Most customers struggle when implementing an AI initiative, and unfortunately, too many fail. This AI reference architecture creates the foundation reference point and has been validated for the enterprise architecture needed to drive operational efficiency. Customers can now avoid the trial-and-error approaches that are associated with customers' lack of domain expertise.

The data pipeline for Advanced Analytics (AI/ML) and Generative AI needs significant amounts of accelerated compute, and this accelerated compute needs high-performance, high-bandwidth, non-blocking networking and storage. The solution is designed to accelerate complex computational processing, so algorithms run faster, data scientists sit less idle, and businesses get faster results.

Sizing the infrastructure is an arduous task, especially for fast-evolving technologies such as Advanced Analytics (AI/ML) and Generative AI. The sizing guidelines provided in this document enable customers to save time and quickly estimate the number of compute nodes and storage required for their use case. It also lists considerations that must be factored in while sizing the infrastructure.

The solution is delivered as an integrated system for fast onsite installation and setup, enabling quick access for data scientists, data engineers, application developers, and end users. All components of the integrated system have been pre-validated and integrated in manufacturing before shipment.

This solution empowers customers to execute AI/analytics workloads at accelerated speed, facilitating quicker decision-making and insight.

Flexible infrastructure:

This solution offers flexible deployment options, allowing deployment on-premises or at near-cloud colocation facilities. This ensures seamless workload execution while providing enhanced security, as data remains within the organization's boundaries. Customers retain full control over their data, enabling swift compliance with industry-specific regulations like HIPAA and GDPR.

This solution offers independent scaling for accelerated compute, high-performance storage, and high-capacity storage without any networking bottlenecks. Customers can start their journey at a smaller scale, with minimal investment, and incrementally expand as the infrastructure demands increase. Given the rapid evolution of the AI landscape, including the expansion to multimodal data types, the exponential growth in data, and changes in customer needs and market dynamics, the demand for expanding their infrastructure solutions is increasing accordingly. The independently scalable design empowers customers to scale as rapidly and extensively as required, adapting seamlessly to evolving demands.

The solution presents a wide array of hardware and software configuration flexibility to develop a customized solution that addresses both their technical and budget requirements. Customers have flexibility in selecting hardware components such as processors, memory, interface types and storage configurations. This is complemented by an equally wide array of flexibility in software component selection, such as Red Hat OpenShift and Kubernetes. The flexibility enables customers to design infrastructure that has appropriate CPU-to-GPU ratios that facilitate the execution of AI data prep, AI workloads, and traditional analytics simultaneously. Furthermore, customers need flexibility and choice of various GPUs, memory capacity, and drive sizes to suit specific workload requirements. For instance, higher-end GPUs are preferred for model customization, and lower-end GPUs are optimal for data processing workloads. The diverse range of component choices enables customers to adapt infrastructure with an optimal component mix for their specific workloads without being constrained to a specific combination cost-effectively.

Cost effective performance:

This solution offers flexible deployment options, allowing deployment on-premises or at near-cloud colocation facilities. This ensures seamless workload execution while providing enhanced security, as data remains within the organization's boundaries. Customers retain full control over their data, enabling swift compliance with industry-specific regulations like HIPAA and GDPR.

This solution offers independent scaling for accelerated compute, high-performance storage, and high-capacity storage without any networking bottlenecks. Customers can start their journey at a smaller scale, with minimal investment, and incrementally expand as the infrastructure demands increase. Given the rapid evolution of the AI landscape, including the expansion to multimodal data types, the exponential growth in data, and changes in customer needs and market dynamics, the demand for expanding their infrastructure solutions is increasing accordingly. The independently scalable design empowers customers to scale as rapidly and extensively as required, adapting seamlessly to evolving demands.

The solution presents a wide array of hardware and software configuration flexibility to develop a customized solution that addresses both their technical and budget requirements. Customers have flexibility in selecting hardware components such as processors, memory, interface types and storage configurations. This is complemented by an equally wide array of flexibility in software component selection, such as Red Hat OpenShift and Kubernetes. The flexibility enables customers to design infrastructure that has appropriate CPU-to-GPU ratios that facilitate the execution of AI data prep, AI workloads, and traditional analytics simultaneously. Furthermore, customers need flexibility and choice of various GPUs, memory capacity, and drive sizes to suit specific workload requirements. For instance, higher-end GPUs are preferred for model customization, and lower-end GPUs are optimal for data processing workloads. The diverse range of component choices enables customers to adapt infrastructure with an optimal component mix for their specific workloads without being constrained to a specific combination cost-effectively.

Consolidated enterprise support:

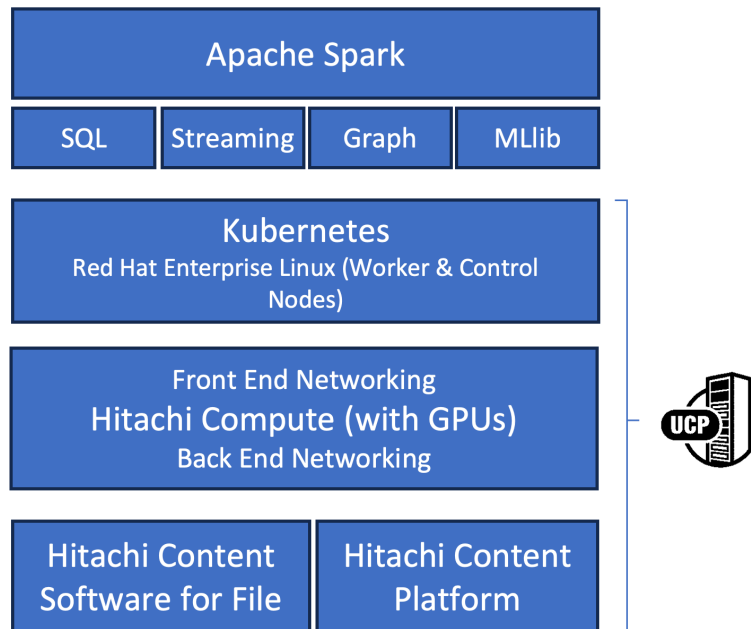
Hitachi Vantara delivers a consolidated support experience for the AI/Analytics infrastructure defined in this RA. The landscape of components used for AI/Analytics workloads is extensive and sourced from multiple vendors, making managing warranties and support agreements for these diverse components a daunting task for customers. Often, different departments within organizations engage in separate contracts with component vendors, leading to suboptimal support experience. Hitachi's compute platform is certified with industry leaders like NVIDIA and Red Hat. With the Hitachi unified support offering, customers can expect expedited SLA-bound issue resolution.

Solution overview

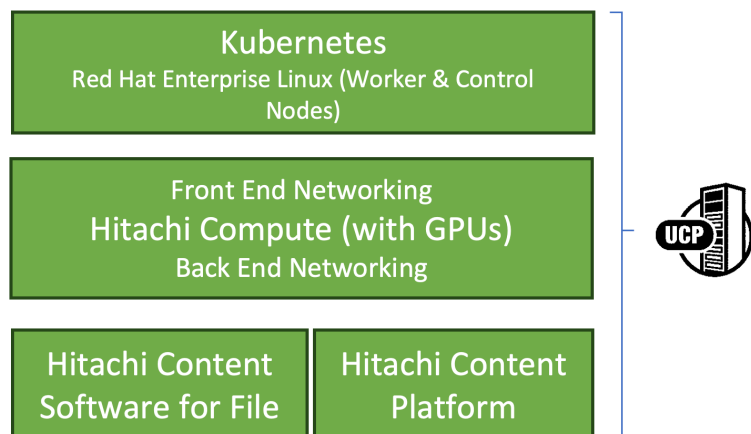
The UCP with HCSF solution reference architecture is designed for Advanced Analytics (AI/ML), Generative AI, and Traditional Analytics applications. Scaling high-performance storage and compute independently is a fundamental requirement to address the needs of the diverse use cases.

The solution is available in three options:

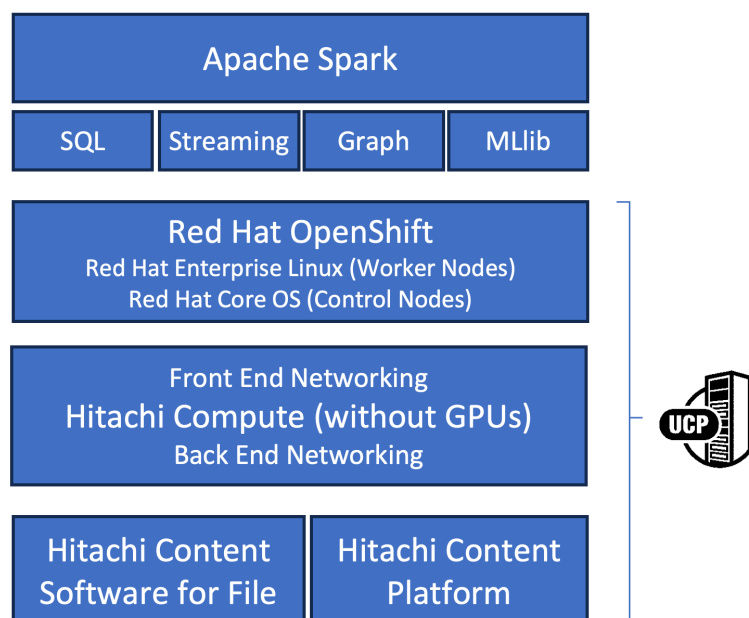
- Kubernetes running on UCP with HCSF with GPUs – Open-source Kubernetes deployed with control and worker node running Red Hat Enterprise Linux. The container stack is validated with Apache Spark.



- Kubernetes running on UCP with HCSF with GPUs – The same stack can also be leveraged for Generative AI workloads. Customers can use underlying containerized infrastructure for generative AI workloads such as fine-tuning and inferencing.



- OpenShift running on UCP with HCSF – Red Hat OpenShift deployed with control nodes running Red Hat CoreOS and the worker nodes running Red Hat Enterprise Linux operating system. For this option, the stack is validated with Apache Spark as well.



Hitachi Compute and Networking

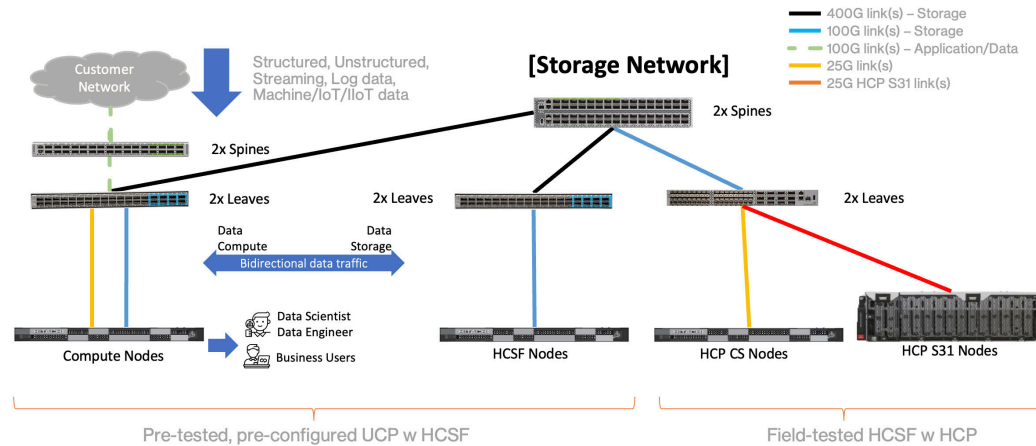
Hitachi Vantara UCP is an advanced infrastructure solution designed to streamline and optimize data center operations. It integrates compute, storage, and networking components into a unified system, simplifying the management and deployment of IT resources. UCP offers scalability and flexibility, allowing organizations to easily adapt to changing workloads and business needs. With its robust features and capabilities, UCP is a reliable choice for businesses seeking efficient and agile data center solutions.

On this infrastructure, workloads such as traditional analytics applications that require access to large amounts of data can be run. These traditional analytics applications need faster CPU with higher core count processors to process huge amounts of data, which is provided by Hitachi Advanced Server compute.

The compute is powered by the Hitachi Advanced Server DS220 G2 and DS120 G2 servers, which are powered by Intel 3rd generation Xeon Scalable processor family. DS220 G2 has a form factor of 2U and DS120 G2 has a form factor of 1U. The total memory capacity for both advanced servers is up to 4TB with RDIMM.

The connectivity for the front-end network is 25 (leaf)/100 (spine) GbE and the connectivity between the compute nodes and HCSF storage nodes is 100 (leaf)/400 (spine) GbE. This enables high-performance compute and high-performance storage connectivity. It is designed as a non-blocking network which helps organizations scale compute and storage independently. With this design, the solution's maximum number of worker nodes can be up to 77 with up to 154 GPUs, and HCSF storage nodes can scale to 24. The leaf and spine switch arrangement is designed to be optimal for the subscription ratio.

The following diagram represents a simplified version of the network diagram connecting compute and storage.



The following table outlines the scale that can be achieved for this solution:

Maximum Racks	Combination of 6
Maximum HCP Racks	4
HCSF Nodes / Rack	24-28
Maximum Compute Nodes / Rack	16*
Compute Data NIC speed	25GbE
Compute Storage NIC speed	100GbE**

* Based on port availability @ leaf switch,

** Shared bandwidth between compute and storage (performance & capacity)



Note:

1. GPUs are supported only with Red Hat Enterprise Linux with Kubernetes Container Platform.
2. GPUs are not supported with the OpenShift Container Platform with Red Hat CoreOS on Control Nodes and Red Hat Enterprise Linux on Worker Nodes. There is a limitation in the NVIDIA GPU driver and in the HCSF CSI driver.

It is possible to configure the solution beyond the scale described in the above table, based on your requirements. Contact technical sales and product management team members for more details.

The hot, warm, and cold storage is provided by a combination of Hitachi Content Software for File and Hitachi Content Platform.

Hitachi Content Software for File

Hitachi Content Software for File (HCSF) is a scalable distributed filesystem that can be deployed rapidly, easily, and affordably. HCSF helps customers gain insights faster by simplifying the process of building and supporting an end-to-end solution and simplify the process of architecting and scaling a solution that is flexible enough to meet a wide range of business requirements.

HCSF is a high-performance storage solution for analytics workloads. It delivers the speed of a high-performance distributed file system with the capacity and hybrid cloud capabilities of an object store. The unique integration of file and object storage offers a tightly coupled, single solution for an appliance-like experience designed for ultra-high performance and capacity applications.

The basic HCSF deployment model involves the creation of a shareable file system to be used by the application servers. The filesystem driver enables each application server to access the HCSF storage system as if it is a local drive, presenting the HCSF storage system as a locally attached filesystem device that is shared among multiple application servers.

Hitachi Content Software for File and Hitachi Content Platform Anywhere along with the Hitachi Content Platform for cloud scale solutions have been tightly integrated to provide a highly available, highly scalable, high-performance, enterprise file and object storage solution that is designed to handle large amounts of data.

The HCSF backend storage nodes are configured as a cluster which, together with the HCSF client software installed on the application servers, forms one large shareable, distributed, and scalable file-accessible storage system.

The basic HCSF deployment model involves the creation of a shareable file system to be used by the application servers. The filesystem driver enables each application server to access the HCSF storage system as if it is a local drive, presenting the HCSF storage system as a locally attached filesystem device that is shared among multiple application servers.




Hitachi Content Software for File and Hitachi Content Platform Anywhere along with the Hitachi Content Platform for cloud scale solutions have been tightly integrated to provide a highly available, highly scalable, high-performance, enterprise file and object storage solution that is designed to handle large amounts of data.

The HCSF backend storage nodes are configured as a cluster which, together with the HCSF client software installed on the application servers, forms one large shareable, distributed, and scalable file-accessible storage system:

- Scalable: The performance of an HCSF is linear and depends on the size of the cluster. Consequently, a certain amount of performance will be received for a cluster of size x , while doubling the size of the cluster to $2x$ will deliver double the performance. This applies to both data and metadata.
- Distributed: An HCSF system is formed as a cluster of multiple backend storage nodes, providing services concurrently.
- Shareable: All clients can share the same filesystems, so any file written by any client is immediately available to any client reading the data. The HCSF system is a strongly consistent, POSIX-compliant system.
- Sustainable: HCSF offers lower energy consumption and reduces the resulting carbon emissions by cutting data pipeline idle time, extending the usable life of the hardware. It also maximizes GPU utilization so that accelerated compute can complete jobs quicker.

HCSF allows customers to design and deploy a leading-edge file system, gain insights faster by simplifying the process of building and supporting an end-to-end solution, and simplify the process of architecting and scaling a solution that is flexible enough to meet a wide range of business requirements.

Based on the scale of analytics problems, there are three HCSF configuration options.

Cluster Type	Entry HCSF cluster	General Purpose HCSF cluster	Performance HCSF cluster
			
Purpose	Designed to be a full cluster unit, good entry point, or development base for starting analytics development.	Designed for dense capacity requirement with IO optimization.	Designed for maximum performance with throughput optimization.

All of these HCSF options can be included with UCP with HCSF solutions to deliver an optimal combination of compute and storage for your analytics requirements.

Hitachi Content Platform

Hitachi Content Platform (HCP) is a secure, simple, and intelligent web-scale object storage platform that delivers superior scale, performance, security, efficiency, and interoperability. It allows any organization to deliver unique, feature rich, private, hybrid, multi-cloud, or public cloud storage services at a cost comparable to public cloud. The rich feature set and extensive ecosystem surrounding the platform allow organizations to improve efficiencies and optimize costs. They can choose to move data to on-premises storage tiers, off-site to a choice of public cloud providers, or to a combination of both.

Red Hat OpenShift

Red Hat OpenShift enables enterprises to accelerate application development with built-in support for continuous integration and continuous delivery. It has better integration with various public cloud service providers so enterprises can scale infrastructure seamlessly when burst capacity is required. It also has validated storage and network plugins that fast-track standing up of solutions infrastructure.

Red Hat OpenShift does this by being a leading containerization and Kubernetes-based platform that enables organizations to efficiently deploy, manage, and scale containerized applications. It provides a robust set of tools for developers to build, test, and deploy applications across hybrid cloud environments, fostering agility and flexibility.

With its comprehensive security features and built-in automation, OpenShift ensures the safe and reliable operation of containerized workloads, making it a popular choice for enterprises seeking to modernize their IT infrastructure. As an open-source solution, it also benefits from a vibrant community and ecosystem of add-ons, making it a versatile and powerful platform for container orchestration and application deployment.

Apache Spark

Apache Spark is a multi-language engine for executing a variety of workloads such as data ingestion and integration, data science and machine learning workloads on multi-node clusters.

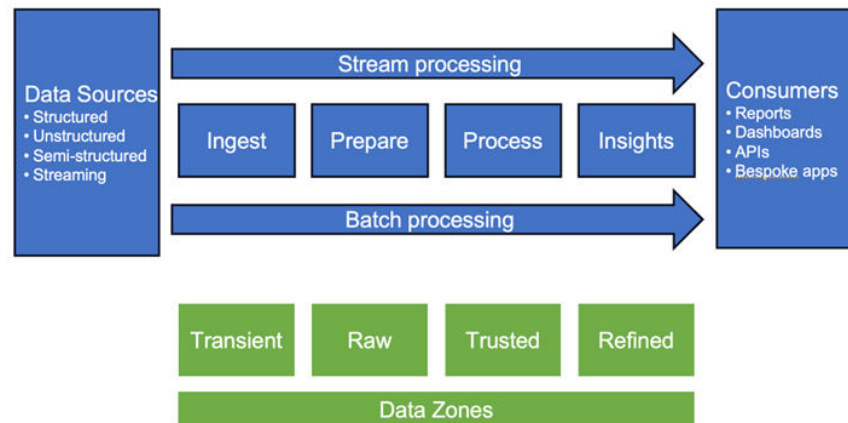
It has four key components:

- SQL and Data Frames for traditional batch data processing
- Streaming for data stream processing

- Graph for graph connected databases
- MLlib for machine learning libraries

Data management

The following illustration shows a micro-service based end-to-end Advanced Analytics (AI/ML), Generative AI, and Traditional Analytics data flow. This is one of the most used approaches in the industry to process workloads.



Data sources

Data sources in data pipelines for Advanced Analytics (AI/ML), Generative AI, and Traditional Analytics encompass a wide array of structured and unstructured data, including databases, cloud storage, logs, APIs, and external feeds. These pipelines play a critical role in collecting, cleaning, and transforming this data into usable formats, enabling organizations to extract insights, train machine learning models, and feed data to Generative AI algorithms. The quality and diversity of data sources are key factors in the success of these pipelines, influencing the accuracy and effectiveness of AI and Generative AI algorithms and applications.

Processing stages

In a data pipeline for Advanced Analytics (AI/ML), Generative AI, and Traditional Analytics, the processing stages typically include data ingestion, transformation, and analysis. For AI pipelines, data preprocessing, model training, and evaluation are essential steps. In the context of Generative AI, the pipeline involves data input, model training for generative tasks, and the generation and evaluation of creative outputs, such as images, text, or music. For Traditional Analytics, the pipeline involves data lookup and simple statistical calculations.

Data zones

Data zones, including trusted, refined, and raw data repositories, play a critical role in Advanced Analytics (AI/ML), Generative AI, and Traditional Analytics workflows. Trusted data zones provide a secure and verified source of data, ensuring the reliability and accuracy of generated insights. Refined data zones offer curated and transformed data, optimizing it for analytics and machine learning processes.

Raw data zones store unprocessed data that is vital for Generative AI applications, enabling model training and innovation by providing access to diverse and unfiltered data sources. These data zones collectively support data-driven decision-making and foster innovation across various domains.

Consumers

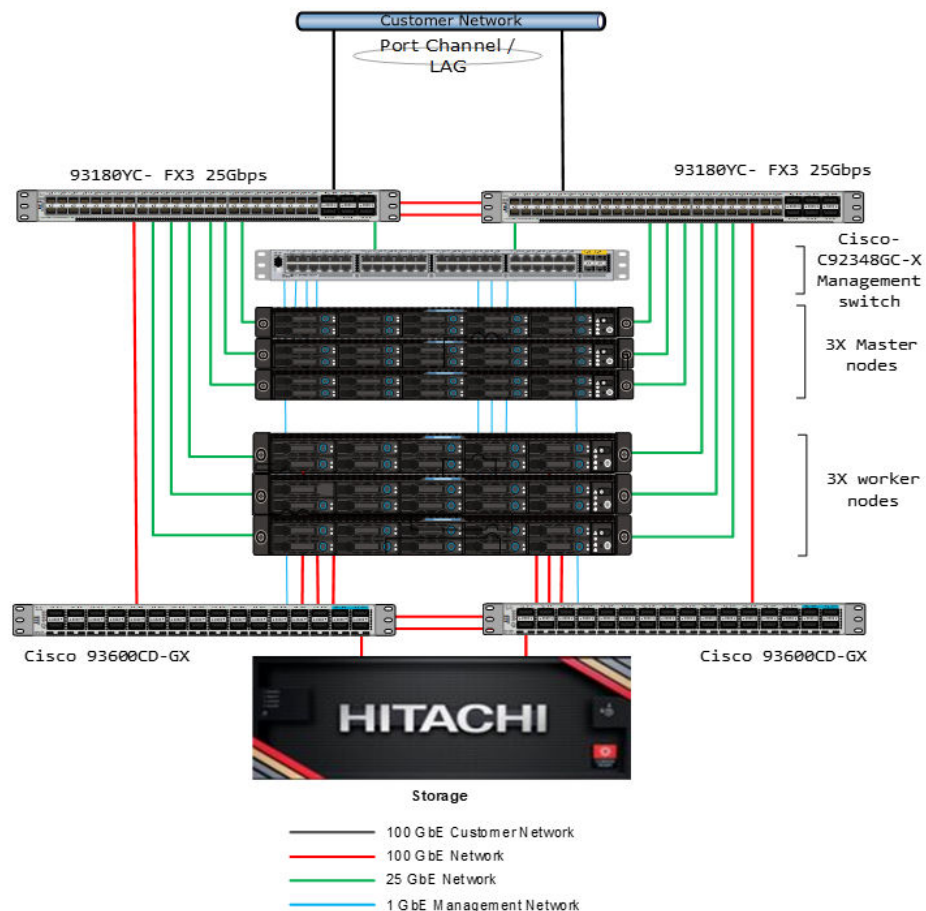
Consumers of Advanced Analytics (AI/ML), Generative AI, and Traditional Analytics leverage BI apps, APIs, and mobile devices to access actionable insights. These tools empower consumers to make data-driven decisions, harnessing the power of artificial intelligence and machine learning for predictive analysis and automation. Additionally, Generative AI adds a creative dimension, enabling the generation of content, art, and innovations, further expanding the horizons of technology-driven possibilities.

Solution architecture

This UCP reference architecture demonstrates a configuration to run containerized Apache Spark on Kubernetes and Red Hat OpenShift Container Platform (OCP) using Hitachi Advanced Servers and Hitachi Content Software for File.

Physical deployment architecture

The following illustration shows the solution architecture.



The architecture includes the following:

- Fully redundant hardware for both OCP worker nodes and master nodes are considered.
 - 3 × DS220 G2 servers as worker nodes
 - 3 × DS120 G2 servers as master nodes
- A dedicated 25 Gb uplink network is used for communication between the nodes.
- Worker nodes are installed with 100 Gb ConnectX-6 DX/VPI for Hitachi Content Software for File storage connectivity.
- Cisco Nexus 9336 100 Gbps switches are used for storage network.
- Cisco Nexus 93180YC-FX3 25 Gbps switches are used for master and worker nodes.
- Cisco C92348GC-X 1 Gbps switch for the management network.



Note: The network layout in this reference architecture was used for testing purposes only. For a production environment and complete configuration of Content Software for File, see the Hitachi Content Software for File documentation at <https://docs.hitachivantara.com/r/en-us/content-software-for-file/4.0.x/mk-hcsf000>.

For AI workloads with GPUs, we used the following configuration:

- Fully redundant hardware for both Kubernetes control nodes and worker nodes.
 - 3 × DS220 G2 servers as worker nodes
 - 3 × DS120 G2 servers as master nodes
- NVIDIA A100 40 GB GPU installed in one of the worker nodes.
- A 25 Gb network is for communication between the control and worker nodes.
- A 25 Gb uplink network to communicate from the cluster to the customer (corporate) network. The developers (customers) will access the compute (that is, control and worker) and storage via this link.
- Worker nodes are installed with 100 Gb ConnectX-6 DX/VPI to connect to Hitachi Content Software for File storage.
- Cisco Nexus 93600CD-GX100 Gbps switches are used for storage network.
- Cisco Nexus 93180YC-FX3 25 Gbps switches are used for master and worker nodes.
- Cisco C92348GC-X1 Gbps switch for the management network.



Note: At the time of writing this document, connectivity to HCSF is supported on GPU-enabled worker nodes with RHEL installed. Connectivity to HCSF is not supported from the worker nodes with RHCOS installed.

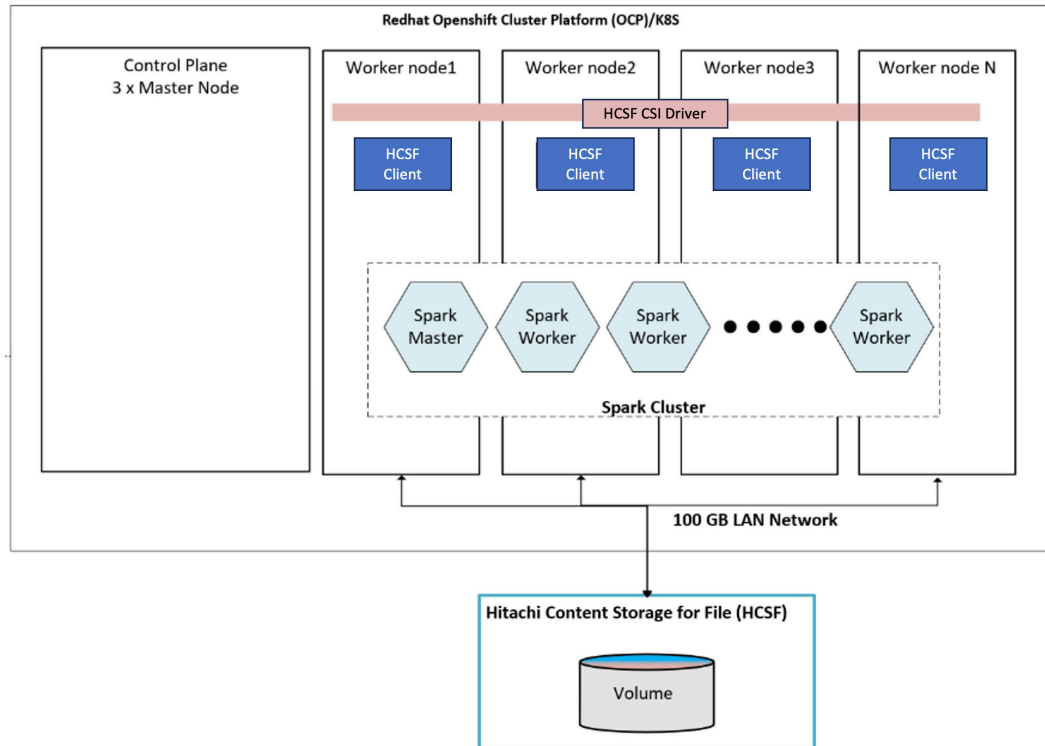
For more information see <https://docs.weka.io/install/prerequisites-and-compatibility>.

Logical deployment architecture

The following illustration shows the logical architecture.



Note: The logical architecture diagram highlights how to scale the cluster for OCP and Spark worker nodes.



The following configuration was used for testing (without GPUs):

- OCP Cluster with RHEL OS 3 × control plane nodes and 3 × worker nodes

The following configuration was used for testing (with GPUs):

- K8S Cluster with RHEL OS on 3× control nodes and RHEL OS on 3× worker nodes

Solution components

These are the key hardware and software components used for this reference architecture.

Hardware components

Vendor	Hardware	Detail Description	Version	Quantity	Role
Hitachi Vantara	Hitachi Advanced Server DS220 G2	Processors: <ul style="list-style-type: none"> ▪ 8368 (38 core, 270W, 2.4GHz) ▪ 8380 (40 core, 270W, 2.3GHz) Memory: <ul style="list-style-type: none"> ▪ 512 GB with 16 × 32 GB DDR4 RDIMM 1024 GB with one of the following: <ul style="list-style-type: none"> ▪ 32 × 32 GB DDR4 RDIMM ▪ 16 × 64 GB DDR4 RDIMM 2048 GB with one of the following: <ul style="list-style-type: none"> ▪ 32 × 64 GB DDR4 RDIMM ▪ 16 × 128 GB DDR4 RDIMM 4096 GB with 32 × 128 GB DDR4 RDIMM LSI MegaRAID 12G SAS/PCIe Secure SAS39xx for local storage other than boot.	Firmware: 3.62.06 BIOS: S5XH3A17.H00	3	Worker nodes
	Hitachi Advanced Server DS120 G2	<ul style="list-style-type: none"> ▪ 2 × Intel(R) Xeon(R) Silver 4210 CPU @ 2.20GHz, 10 Cores, 56 Logical Processors 	Firmware: 4.70.06 BIOS: S5BH3B25.H00	3	Master nodes

Vendor	Hardware	Detail Description	Version	Quantity	Role
		<ul style="list-style-type: none"> ▪ 2 × Intel(R) Xeon(R) Silver 4210 CPU @ 2.20GHz, 10 Cores, 56 Logical Processors ▪ 382 GB (32 GB × 12) DIMM DDR4 Synchronous Registered (Buffered) 2666 MHz ▪ 2 × 128 GB SATADOM for boot 			
NVIDIA	Mellanox Technologies MT2892 Family [ConnectX-6 Dx]	Dual-Port 100 GbE/Single-Port 200 GbE Smart NIC	Firmware: 22.35.1012	6	Hitachi Content Software for File storage network
NVIDIA	Mellanox Technologies MT2892 Family [ConnectX-6 Lx]	Dual-Port 25 GbE/Single-Port 50 GbE Smart NIC	Firmware: 26.35.1012	6	Inter-node connectivity
Cisco	Cisco Nexus 93180YC-E/FX3	48 × 1/10/25 Gbps fiber ports and 6 × 40/100 Gbps QSFP28 ports	BIOS version: 07.65 NXOS version: 9.3	2	Inter-node communication ToR switch for compute
Cisco	Cisco Nexus 933600CD-GX	28 × 100/40-Gbps Quad Small Form-Factor Pluggable (QSFP28) and 8 × 400/100-Gbps QSFP-DD ports	NXOS version: 9.3.5	2	ToR switch for Content Software for File storage

Vendor	Hardware	Detail Description	Version	Quantity	Role
Cisco	Cisco-C92348GC-X	1 Gbps management switch	NXOS version: 9.3(3)	1	Management Switch



Note: An existing HCSF cluster was used for this testing.

Software components

For the OCP Cluster Setup:

Software	Version	Function
Compute Nodes		
Red Hat OpenShift Container Platform (OCP)	4.10	Container Orchestration Platform
Red Hat Enterprise Linux	8.4	Operating System on OCP worker nodes
Red Hat CoreOS (RHCOS)	4.10	Operating System on OCP control plane nodes
HCSF Agent	4.0.1	HCSF Agent
Apache Spark	3.3.1	Application

For the K8S Cluster setup:

Software	Version	Function
Compute Nodes		
Kubernetes Cluster (K8S)	1.27.1	Container Orchestration Platform
Red Hat Enterprise Linux	8.4	Operating System on K8S control and worker nodes
Red Hat Core OS (RHCOS)	4.0.1	HCSF Agent
Apache Spark	3.3.1	Application

Sizing considerations

Sizing guidelines and recommendations discussed here are based on the in-house infrastructure components and source data characteristics. These guidelines indicate the parameters to be considered when determining the compute/network/storage infrastructure necessary for an organization's requirements.

Component features such as faster processors and memory continue to improve, and data characteristics vary across organizations. Therefore, the sales team and professional services teams should be engaged when determining the size and configuration needed to meet specific use case requirements.



Note: The suitability of these sizing recommendations might vary for different environments. Data characteristics, workload variations, and tuning parameters can influence actual performance. Therefore, we recommend conducting thorough testing and tuning for each unique use case to ensure optimal Generative AI Model performance on a given cluster.

For more information or assistance sizing your infrastructure, contact your sales team.

This section discusses sizing for the following two topics:

Advanced Analytics (AI/ML)

Several steps are involved while running advanced AI algorithms to glean insight from the data. Data sizing is a critical step to ensure AI algorithms and associated data curation are run successfully and fault-free. Understanding the complexity of the algorithm and the volume, variety, and velocity of data is essential to provide a robust infrastructure for Apache Spark so that it can scale effectively and extract insights. While much is dependent on the customer use case, the sizing considerations discussion of one realistic situation is explained as an example.

Generative AI

This is a rapidly evolving field with advances in hardware and software technology reducing the time required to train and fine-tune the model and draw inferences. For sizing consideration, a Large Language Model (LLM), OpenLLaMA with 7 billion (7B) parameters is used. The details cover fine-tuning and inferencing only. Additional considerations are discussed as well.

Advanced Analytics (AI/ML)

A typical data warehouse contains 100s of terabytes of data. It is arranged in various data marts with different schemas better suited for AI algorithms and data analysis. Having a data mart with about 25 terabytes of data is common. In this scenario, the data warehouse has 25 terabytes of data with the expectation to process this data in under an hour.

When sizing the infrastructure, we start with a small cluster. We then determine how many AI algorithms this cluster can run or how much data this cluster can process. After we have this estimate, it can be used as a multiplier to estimate the sizing of the infrastructure to run the required AI algorithms or to process the required amount of data.

The alternative is to start with an individual Spark AI algorithm using MLlib, or process and work your way up to estimate the sizing of the infrastructure. In this reference architecture, we started with an individual Spark process because it is much more granular and can be generalized for estimating the required infrastructure for other data processing engines such as Apache Ray or Apache Arrow.

NVIDIA RAPIDS Accelerator for Apache Spark can leverage NVIDIA GPUs to accelerate Advanced Analytics (AI/ML) data curation tasks such as data preparation and data transformation. By offloading data from SQL computational tasks to GPUs, enterprises can benefit from faster execution time, reduced infrastructure costs, and quick time to value. While RAPIDS Accelerator for Apache Spark is not discussed for sizing, integrating this accelerator will certainly benefit enterprises even further. More information about NVIDIA RAPIDS Accelerator for Apache Spark is available, including the ability to analyze existing Spark logs to provide an estimate of the acceleration resulting from GPUs. The tool analyzes Spark events generated from CPU based Spark applications to help quantify the expected acceleration of migrating a Spark application to GPU. The analyzes both CPU or GPU generated event logs and generates information which can be used for debugging and profiling Spark applications.

Calculation details

1. Each Spark executor (for AI algorithms or to process a certain amount of data) consumes about 4 CPU cores – 3 cores for data processing and an additional core for OS and execution management.
2. Each Spark executor consumes 24 GB of RAM.
3. Let us assume here that it takes about 10 minutes to process 25 GB of data per core for an individual Spark executor to process reasonably complex data.



Note: The definition of “reasonably complex” data will vary between organizations. The numbers given are empirical and will change.

4. A Spark executor allocated with 4 cores and 24 GB RAM can process up to 75 GB of data in 10 minutes (3 cores × 25 GB/core).
5. Let us allocate 10% of the RAM and Cores to the OS and cluster management. In that case, a single node with 500 GB RAM and 80 CPU cores will have 450 GB usable RAM and 72 CPU cores for data processing.

Therefore, each individual machine can process approximately 8 TB of data per hour. The calculation for this is as follows:

Spark Executor = the lower of 18 (450 GB/25 GB) or 18 (72 cores/4 cores) = 18

Data processed = (# Spark Executors) * (75 GB/10 min) * (60 min/1 hr) = 8.1 TB

With a few additional calculations, to run AI algorithms or to process 24 TB of “reasonably complex” data in under an hour, three two-socket systems with 40 core CPUs per socket and 512 GB RAM are required.



Note: Keep in mind that the suitability of these sizing recommendations may vary for different customer environments. Factors such as data characteristics, workload variations, and tuning parameters can influence actual performance. Therefore, we recommend conducting thorough testing and tuning for each unique use case to ensure optimal Spark cluster performance.

General sizing considerations

To manage the volume, velocity, variety, and complexity of the data you intend to run AI algorithms on or to analyze and process for an advanced analytic infrastructure, the considerations listed in this section are crucial. Proper sizing helps prevent performance bottlenecks, reduces operational costs, and ensures efficient data processing.

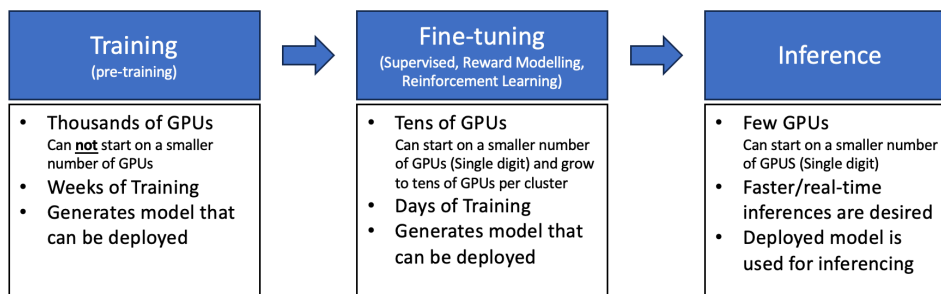
Several key factors to consider when sizing your advanced analytic infrastructure include:

- Data volume
 - Estimate the amount of data you will be working with, both currently and in the foreseeable future. This includes historical data and anticipated growth rates.
 - Consider factors such as data retention policies and compliance requirements because these can impact storage needs.
- Data velocity
 - Determine the rate at which data is ingested into your system. This will help you size components such as data ingestion pipelines and real-time processing clusters.
 - Be prepared for spikes in data velocity because some applications might experience bursts of incoming data.
- Data variety
 - Analyze the types and formats of data you will be working with, including structured, semi-structured, and unstructured data.
 - Choose appropriate tools and technologies for processing different data formats and size your infrastructure accordingly.
- Data complexity
 - Assess the complexity of your analytics tasks. Some algorithms and models might require more computational resources than others.
 - Consider the need for data pre-processing, feature engineering, and data transformation steps.
- Data processing workloads
 - Understand the specific analytics workloads your infrastructure will support. This could include batch processing, real-time streaming, machine learning, or graph analytics.
 - Size your infrastructure components (clusters, servers, memory, and CPU) to manage these workloads efficiently.
- Scalability
 - Design your infrastructure to be scalable both horizontally and vertically. This allows you to add resources as your data and processing needs grow.
 - Consider technologies such as containerization and orchestration (for example, Kubernetes) to enable easier scalability.
- Network bandwidth
 - Ensure that your network infrastructure can manage the data transfer rates required for data movement within your analytics ecosystem.
 - Minimize network latency to improve overall system performance.
- Data storage
 - Select the right data storage solutions (high performance, HDFS, distributed databases, or data lakes) based on data volume and access patterns.
 - Estimate the growth rate of your data and plan for storage expansion.

Generative AI

Generative models represent a category of AI designed to generate meaningful content by leveraging extensive datasets encompassing text, images, audio, and code. This class of models supports diverse applications, including personalized user experiences and the creation of innovative art. Despite the array of applications, the foundational algorithm at the heart of these models typically involves some form of neural network. These neural networks are trained on substantial datasets, enabling them to learn intricate patterns and produce coherent, contextually relevant outputs.

The typical life cycle of generative models is captured in the following illustration:

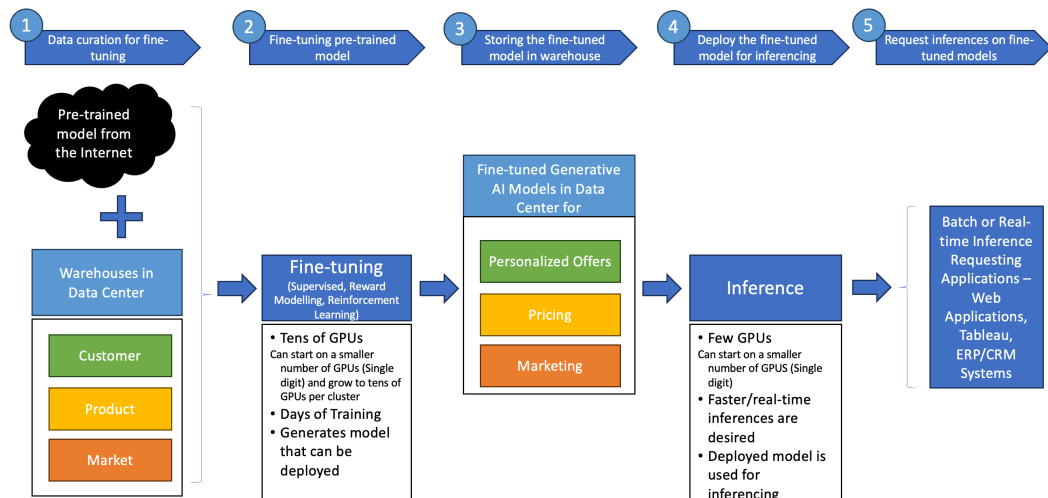


Training: Using a variety of data sources available on the internet, such as Wikipedia and Internet Archives, the models are trained on thousands of GPUs over weeks to ensure the model can communicate as effectively as humans in widely spoken languages such as English and Mandarin.

Fine-Tuning: Using the relevant data sources for a given use case, the trained model is fine-tuned, further adapting it to perform and excel at a specific task or function. Fine-tuning tailors the model's outputs to your specific needs, leading to higher quality and accuracy. This can be achieved with a smaller number of compute nodes with GPUs.

Inference: After training or fine-tuning, the models can be deployed in production for inferencing either at the edge or in the data center. The infrastructure required can start with a smaller number of compute nodes with GPUs and scale to many nodes when inferencing many jobs in parallel.

To understand how fine-tuning and inferencing processes interface with each other and enterprise applications, refer to the following illustration:



The following steps outline the process of fine-tuning the model and running inference on fine-tuned models:

1. **Data curation** – After the appropriate model is obtained/downloaded, internal data is curated so that the downloaded pre-trained model can be fine-tuned for desired business goal. For example, images/videos are converted to a certain resolution, data is converted to a format which can be digested by the model.
2. **Fine-tuning** – The downloaded pre-trained model is fine-tuned in this process for specific business goals. For example, for a personalized offering with the corresponding discount, customer product data is fed to the pre-trained model.
3. **Storing fine-tuned models** – Business specific fine-tuned models are stored in the data warehouse and are prepared for deployment on the inferencing infrastructure.
4. **Inferencing** – An appropriate version of the fine-tuned model for a specific business goal (for example, a pricing discount) that is deployed on the inferencing infrastructure.
5. **Request for inferencing** – Enterprise applications deployed in production, such as a web site of a large retailer, reporting systems of a specific department, or an ERP/CRM system of the enterprise, start requesting inferences for a particular set of questions to the fine-tuned model deployed for inferencing.

The following are infrastructure sizing guidelines for fine-tuning and inferencing.


Fine-tuning

There are several methods of fine-tuning a trained model. In this section, a simple supervised fine-tuning method for the pretrained model OpenLLaMA 7B is described. The parameters to consider are the vocabulary (consists of a sequence of words rather than individual words), learning rate (determines the step size), batch size (number of training examples to be given to model in each step), sequence length, and epochs. For understanding purposes, consider an epoch as running the model once with all the training data required to fine-tune the model.

Each of the parameters has an influence on the computing resources required. For example, the vocabulary size is directly proportional to the size and the speed of processing (CPU) and memory (CPU RAM). Higher numbers of epochs result in better accuracy, up to a certain number of epochs. A higher number of epochs require faster GPUs. Larger models require larger GPU RAM.

To run the workloads for Generative AI and related data curation, higher clock speeds and a larger number of physical cores for the CPUs are recommended.

The following table outlines the calculations for the number of cores required for fine-tuning.

CPU Physical Cores	Usage
6	NVIDIA recommends 6 CPU cores per GPU installed on the server
20	<p>Fine-tuning jobs also involve data curation processes, as explained in the previous section.</p> <p>In 2 hours, 20 cores can typically process about 4 TB of data.</p> <ol style="list-style-type: none"> 1. Assuming that each Spark job is configured with 4 CPU cores and 24 GB main memory, three CPU cores are used for processing and 1 CPU core is used for management and scheduling. The memory used for management and scheduling will be insignificant. 2. Assume that a single core can process 25 GB of data in 10 minutes. 3. In an hour 1 Spark job can process 25 GB * 3 Cores * (60/10) minutes = 450 GB of data. 4. 1 Spark job uses 4 cores. Therefore, 20 cores can process approximately 2 TB of data in one hour. <p> Note: Assume 2 hours for data curation. Based on your enterprise SLA scale this parameter up or down.</p>
4	For OS and management, 4 CPU codes are recommended
30	Total cores

Per GPU installed in the server, a physical CPU with 30 cores and a higher CPU clock frequency is recommended for fine-tuning.

- **Main Memory (CPU):** Main memory of 2 to 3 times that of the GPU RAM is recommended for efficient execution of Generative AI workloads. It is crucial that the model fits in GPU RAM as much as possible. For example, if $2 \times$ A100 80 GB GPUs are installed in the server, the total GPU memory in the system will be 160 GB. Therefore, the recommended main memory should be 2 to 3 times the GPU RAM or between 320 GB to 480 GB. The minimum main memory is twice the size of the GPU RAM.
- **GPU and GPU High Bandwidth Memory (HBM) RAM:** It is strongly recommended that the GPUs selected fit the maximum portion of the model in the GPU HBM RAM. For example, for the OpenLLaMA 7B model, a GPU with at least 14 GB of HBM RAM is recommended, considering each node in the neural net holds parameters at half-precision (16 bits), which is common. GPUs with a higher number of Tensor Cores and a higher number of floating-point operations (FLOPS) will perform better for fine-tuning or running inferences.
- **Type of GPUs:** For efficient utilization of GPUs, partitioning capabilities such as Multi-Instance GPU (MIG) are desirable. For example, consider using a MIG-enabled NVIDIA A100 80 GB GPU with multiple partitions of 20 GB to fine-tune the OpenLLaMA 7B model. Other parameters to consider are GPUs with greater HBM and FLOPS. The MIG capability is especially useful for a large number of smaller jobs. For instance, AI inference workloads that typically do not demand all the performance of a single GPU can benefit from occupying a slice of GPU RAM and computational capability while leaving the remaining GPU resources for other workloads.
- **Internal Storage:** Because of a variety of factors, the data processed on CPUs and GPUs is spilled over from the CPU RAM and GPU HBA RAM to storage. Internal storage of at least 2.5 times the sum of main memory + GPU HBA RAM is recommended. For example, if there is 512 GB of main memory and $2 \times$ A100 80 GB GPUs per worker node, then 2 TB high performance NVMe internal storage is recommended per worker node.
- **External Storage:** External storage is required to store the data required for fine-tuning. We need to store the LLM model and the data for the product, customer, and market in the data warehouse.

The storage required for the LLM model: OpenLLaMA 7 billion model is trained on 1 trillion tokens (as of July 2023). Given this information, it is safe to assume that approximately 1% or 10 billion tokens will be used for fine-tuning. Further assuming the average token size to be around 80 bytes, the storage requirements can be estimated as follows:

Storage = Number of Tokens \times Average Token Size

For 10 billion tokens with an average of 80 bytes, approximately 800 GB is needed.

- The assumption above leans towards the higher end. It is possible to fine tune the model effectively even with a significantly smaller dataset.
- A token size of 80 bytes consists of 10 words with an average of 8 characters, including spaces and punctuation. The reader can vary this parameter as needed.

A Data mart of 10s of terabytes is common, and there are several such data marts in each data warehouse. About 30% of this data is stored on hot storage while the rest is stored on warm/cold data storage.

Use the following table to decide how much storage is needed:

Data mart	Hot data (HCSF)	Warm/cold data (HCP)	Total
Models (pre-trained, fine-tuned, and ready for deployment) *	4 TB	16 TB	20 TB
Customer	30 TB	70 TB	100 TB
Product	30 TB	70 TB	100 TB
Market	30 TB	70 TB	100 TB

* Expecting 4 to 5 teams working on this at a given point in time, and having 4 to 5 versions of each model, the storage can be up to 20 TB (5 teams * 5 Models * 800 GB/model with 1 active model).

In this example, we need at least 94 GB of hot storage (HCSF), and we also need 226 GB of warm/cold storage (HCP).

A containerized environment is preferred for Generative AI workloads. Containers are lightweight and run consistently across different environments. Containers provide a sufficient level of isolation yet can share data easily. Other benefits of a containerized environment include developer-friendly version control, the ability to deploy microservices-based architectures, resource efficiency, and orchestration.

Now putting this all together for fine-tuning the OpenLLaMA 7B model, the following configuration is recommended:

1. Three control nodes with two CPUs (2GHz, 28C) and 256 GB RAM or better per node.

Why? – Three control nodes are recommended for the production environment.

2. Four worker nodes with two CPUs (2GHz, 32C) and 512 GB RAM or better per node.

Why? – A minimum of two environments are required to fine-tune. One for actual development, and another for QA/acceptance testing. For each environment, two worker nodes are typically provisioned, therefore a total of four worker nodes is needed.

3. Each worker node is equipped with two A100 80 GB GPUs.

Why? – The OpenLLaMA 7B model needs 20 GB of GPU slice to run. For concurrent execution, this model is deployed on two slices, so two GPUs are needed. Also, multiple teams are expected to work in parallel requiring more than 80 GB of GPU RAM.

With two A100s with 160 GB of RAM, up to four teams can simultaneously fine-tune their models. Optionally, while fine-tuning is happening, other GPU slices can be used for data processing (AI/ML related) and other model training.

In general GPUs with greater memory and higher FLOPS are recommended for fine-tuning.

4. Each worker node is equipped with a total of 2 TB of fast local storage such as NVMe.

Why? – Each node has 512 GB of main memory and 160 GB of GPU HBM RAM. For spillover storage, 2.5 times the sum of total main memory and GPU HBM RAM is suggested, therefore 2 TB of local storage is recommended.

5. External parallel distributed and capacity storage (HCSF + HCP).

Why? – To fine-tune the downloaded pre-trained model, storage is needed for the model and associated parameters, plus the enterprise business-specific vocabulary. In this example, we need at least 94 GB of hot storage (HCSF), and we also need 226 GB of warm/cold storage (HCP).

6. Each node is equipped with a single port 100 GbE network adapter for the storage network and a single port 25 GbE network adapter for the front-end network.

Why? – For distributed parallel data processing engines such as Spark and Dask, and storage such as HCSF, there is a sizeable amount of traffic on the network. Faster 100 GbE NICs help to move the data faster between compute and storage. The front-end network can work with a 25 GbE network, as high-volume, high-speed data is restricted on the storage network. For added redundancy, dual port network adapters can be considered.

Inferencing

Compared to fine-tuning, inferencing has fewer considerations for model deployment. In production, GPU speed (FLOPS), GPU memory, and the desired connections are the key drivers for the AI model's inference performance. The inferencing workloads involve extracting inferences from the AI model loaded in GPU memory. The inference workloads submit the requests, either in batch or real-time mode to the deployed AI model to extract inferences. Therefore, less storage is required for batch inferencing workloads when compared to fine-tuning workloads.

If the inferencing workloads are real-time, the storage requirement is reduced further. The other key consideration for inferencing is how many connections the infrastructure can sustain when an inference on the given data is being requested. This value is a function of CPU and main memory.

To run AI inferencing on the edge or in remote locations, a single node with a single mid-range CPU, and with a single GPU is sufficient. A single AI model can be deployed on a GPU for inferencing. Edge data processing focuses on preparing job submissions to the AI model. Therefore, nominal storage is recommended to hold data to be submitted and inferences to be received from the AI model. With real-time inferencing, storage requirements are reduced even further.

The following is a summary of a configuration that can serve edge data inferencing requirements:

1. Single node with single mid-range CPU (2GHz, 16C).

Why? – 6 cores for GPU coordination, 4 cores for OS and management, and 4 cores for inference request and response management. A total of 14 cores are required.

2. RAM: 192 GB (Main CPU memory – 12 × 16 GB, or 6 × 32 GB).

Why? – Main memory 2 to 3 times that of GPU RAM is recommended for CPU memory. Using the L40S GPU with 48 GB RAM, three times 48 GB is 144 GB. For maximum memory performance, the memory DIMM size should be selected to populate all memory channels available to the CPU.

3. GPU: L40S (48 GB GPU memory).

Why? – The OpenLLaMA 7B model can be deployed in 20 GB GPU RAM. NVIDIA's L4 GPU with 24 GB can fit the OpenLLaMA 7B model; however, for most used AI operations using FP16, the L40S is three times faster than L4 (L40S – 733TFLOPS vs. L4 – 242TFLOPS). If lower performance without any future expansion possibility is acceptable, then L4 can be used.

4. Storage: Internal 800 GB (2 × 400 GB NVMe). In the case of a RAID configuration, the number of disks required will be higher.

Why? – Each node has 192 GB of main memory and 48 GB of GPU HBA RAM. For spillover storage, 2.5 times the sum of total main memory and GPU HBA RAM is suggested, and therefore 800 GB ((192+48) * 2.5 = 600 GB) of local storage is recommended. If the data to be submitted and inferences extracted are of significant size, then additional storage space may need to be provisioned.

For remote office, branch office (ROBO), and data center deployments, enterprises seek additional inferencing capacity with resiliency. For such a scenario, a three-node cluster running control and worker nodes is recommended. The containerized cluster can run in a bare metal or virtualized environment. Depending on the type and number of GPUs installed, the number of AI models that can be deployed for inferencing can vary.

For example, if three L40S GPUs are installed on this cluster, then three models can run inferencing workloads on this cluster concurrently. With increased CPU capacity, heavier data preparation workloads can be run in batch or/and real-time. Due to heavier data preparation workloads, slightly increased storage is recommended.

The following summarizes the configuration per node:

1. Three node cluster with single mid-range CPU (2GHz, 16C):

- a. For single or no GPU node: single CPU (2GHz, 16C).
- b. For dual GPU nodes: dual CPU (2GHz, 16C)

Notes:

- A total of 14 cores are required. 6 CPU cores for GPU coordination, 4 cores for OS and management, and 4 cores for inference request and response management.
- Proportional to data processing complexity, CPU speed and core count can be scaled up or down.
- As the inferencing workload increases, control nodes and worker nodes can be decoupled, and their count can be scaled up. For a higher inference workload, 3 control nodes and worker nodes proportional to AI models to be deployed for inferencing are recommended.

2. RAM:

- a. For single GPU nodes: 192 GB (Main CPU memory – 12 × 16GB, or 6 × 32 GB).
- b. For dual GPU nodes: 384 GB (Main CPU memory – 24 × 16 GB, or 12 × 32 GB)

3. GPU: L40S (48 GB GPU memory):

Note: The L40S does not have the multi-instance GPU capability, therefore a single model can be deployed. For a higher number of inferencing AI models per GPU, select a GPU with MIG capability, such as A100.

4. Storage:

- a. For single GPU node: Internal 800 GB (2 × 400 GB NVMe)
- b. For dual GPU nodes: Internal 1.6 TB (2 × 800 GB NVMe)

Note: In the case of RAID configuration, the number of disks required will be higher.

Why? – Each single GPU node has 192 GB of main memory and 48 GB of GPU HBA RAM. For spillover storage, 2.5 times the sum of total main memory and GPU HBA RAM is suggested, therefore 800 GB $((192+48) * 2.5 = 600 \text{ GB})$ of local storage is recommended. A similar calculation is applicable for dual GPU nodes.

General sizing considerations for Generative AI models

Infrastructure Sizing Considerations for Generative AI Models							
Compute	Memory (GPU & CPU)	Storage	Network	Scalability	Cost	Monitoring	Observability
<ul style="list-style-type: none"> • Model Size • Training vs Inference • Batch Size 	<ul style="list-style-type: none"> • Model Parameters • Data Size 	<ul style="list-style-type: none"> • Performance Data Storage • Capacity Data Storage • Model Checkpoints • Output Storage 	<ul style="list-style-type: none"> • High-Throughput Network • Low Latency Network 	<ul style="list-style-type: none"> • Burst Capacity • Seasonality & Duration • Containerization 	<ul style="list-style-type: none"> • Rightsizing • On-premise & Cloud mix 	<ul style="list-style-type: none"> • Performance • Model Performance 	<ul style="list-style-type: none"> • Model Tracking • Alerting • Auditing • Logging

Provisioning infrastructure for generative AI models involves consideration of the following parameters to ensure optimal performance, scalability, and efficiency.

- Model architecture and size

Understand the architecture and size of the generative AI model. Larger models with more parameters typically require more computational resources.

- Compute power (CPU/GPU/TPU)

Choose the appropriate processing units based on the model's requirements. Generative AI models often benefit significantly from powerful GPUs or TPUs for faster training and inference.

- Memory (RAM)

Ensure sufficient RAM to accommodate the model, especially for larger models and datasets. Inadequate memory can lead to performance bottlenecks.

- Storage

Assess storage requirements for model parameters, datasets, and any intermediate results. Fast storage, such as SSDs, can improve data access speeds during training.

- Data pipeline and input/output throughput

Optimize data pipelines to efficiently feed data to the model during training. Consider the throughput required for reading and writing data.

- Parallelism and distributed training

Determine if the model supports parallel training or distributed training across multiple devices or nodes. Provision the infrastructure accordingly to leverage parallel processing capabilities.

- Scalability

Plan for scalability, especially if the workload is expected to grow. Consider whether the infrastructure can easily scale horizontally or vertically based on demand.

- Network bandwidth

Ensure sufficient network bandwidth to manage the data transfer between components, especially in distributed training scenarios.

- Batch size

Adjust the infrastructure based on the preferred batch size during training. Larger batch sizes may require more memory and computational power.

- Precision (Floating-point format)

Consider the precision requirements of the model (that is, 16-bit or 32-bit floating-point). Lower precision can reduce memory requirements but may affect model accuracy.

- Framework and software dependencies

Ensure compatibility with the deep learning framework used for building the generative AI model. Install and configure any required software dependencies.

- Monitoring and logging

Implement monitoring and logging solutions to track resource utilization, model performance, and potential issues.

- Security considerations

Implement security measures to protect sensitive data and ensure secure access to the infrastructure.

- Cost considerations

Evaluate the cost implications of the chosen infrastructure configuration. Optimize for cost-effectiveness while meeting performance requirements.

- Lifecycle management

Plan for model deployment, versioning, and updates. Consider the infrastructure requirements for hosting and serving the generative AI model in a production environment.

The specific requirements for your implementation can vary based on the nature of the generative AI model, the dataset, and the desired performance characteristics. Regularly monitor and reassess infrastructure needs as models evolve or datasets change.

Methodology of solution validation and testing

The solution is validated and tested for:

1. AI/GenAI with GPUs using Kubernetes on UCP with HCSF
2. Traditional Analytics without GPUs using Red Hat OpenShift (OCP) on UCP with HCSF

GPUs are supported only with Red Hat Enterprise Linux using Kubernetes container orchestration system. GPUs are not supported with the OpenShift container orchestration system on control nodes and Red Hat Enterprise Linux on the worker nodes. This is a limitation in the NVIDIA GPU driver and in the HCSF CSI driver. Because of this the solution was validated with combinations as stated above.

Post solution validation, solution is tested using Spark. The algorithm used is K-Means for GPUs using NVIDIA RAPIDS Accelerators for Spark. For testing the solution without GPUs, Spark in-built examples are used. How the job is scaled with Red Hat OpenShift is also tested.

Solution validation



Note: Installation and configuration of Hitachi Content Software for File storage, Kubernetes, and OCP is beyond the scope of this reference architecture.

This section contains details of how to install Kubernetes and Red Hat OpenShift container orchestration systems on the infrastructure. The steps for Kubernetes are marked with K8S, and steps for Red Hat OpenShift are marked as OCP at the beginning of each section. There is some commonality while installing these container orchestration systems, and they are marked as common before the section.

Generative AI and Enterprise AI test cases were executed with GPUs, while Traditional Analytics use cases were executed without GPUs.

AI/GenAI with GPUs

For information about Kubernetes see the Kubernetes documentation at <https://kubernetes.io/docs/home/>.

Procedure

1. Install and configure haproxy.
2. Install and configure keepalived.
3. Prepare Kubernetes nodes for the traffic control utility package, and then install the traffic control utility package.

```
$ sudo# dnf install -y iproute-tc
```

4. Allow firewall rules for K8S.
5. Install the CRI-O container runtime.

We need to enable two crucial kernel modules – `overlay` and `br_netfilter` modules.

To achieve this, we need to configure the prerequisites as follows:

First, create a modules configuration file for Kubernetes.

```
$ sudo# vi /etc/modules-load.d/k8s.conf
```

Add these lines and save the changes.

```
overlay
br_netfilter
```

Then load both modules using the `modprobe` command.

```
$ sudo# modprobe overlay
$ sudo# modprobe br_netfilter
```

Next, configure the required `sysctl` parameters as follows.

```
$ sudo# vi /etc/sysctl.d/k8s.conf
```

Add the following lines:

```
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
```

Save the changes and exit. To confirm the changes have been applied, run the following command:

```
$ sudo# systemctl --system
```

To install CRI-O, set the \$VERSION environment variable to match your CRI-O version. For instance, to install CRI-O version 1.26 set the \$VERSION as shown:

```
$ #export VERSION=1.26
```

Next, run the following commands:

```
$ sudo# curl -L -o /etc/yum.repos.d/devel:kubic:libcontainers:stable.repo
https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/
CentOS_8/devel:kubic:libcontainers:stable.repo
$ sudo# curl -L -o /etc/yum.repos.d/devel:kubic:libcontainers:stable:cri-
o:$VERSION.repo https://download.opensuse.org/repositories/
devel:kubic:libcontainers:stable:cri-o:$VERSION/CentOS_8/
devel:kubic:libcontainers:stable:cri-o:$VERSION.repo
```

Then use the DNF package manager to install CRI-O:

```
$ sudo# dnf install cri-o
```

Next, enable CRI-O on boot time and start it:

```
$ sudo# systemctl enable cri-o
$ sudo# systemctl start cri-o
```

6. Install Kubernetes Packages.

```
$ sudo# vi /etc/yum.repos.d/kubernetes.repo
```

And add the following lines.

```
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://
packages.cloud.google.com/yum/doc/rpm-package-key.gpg
exclude=kubelet kubeadm kubectl
```

Save the changes and exit. It will look like the following screenshot.

```
[root@lab-master-1 opt]# cat /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
exclude=kubelet kubeadm kubectl
```

Save the changes and exit. Finally, install K8S package as follows.

```
$ sudo# dnf install -y kubelet-1.27.1 kubeadm-1.27.1 kubectl-1.27.1 --
disableexcludes=kubernetes
```

Once installed, be sure to enable and start the Kubelet service.

```
$ sudo# systemctl enable kubelet
$ sudo# systemctl start kubelet
```

7. Post this configure K8S based on your enterprise standards.
8. Install CalicoAntrea Pod Network Add-on.

```
$ #kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/
v3.25.0/manifests/calico.yamlantrea-io/antrea/main/build/yamls/antrea.yml
```

To verify that the pods have started, run the following command.

```
$ #kubectl get pods -n kube-system
$ #kubectl get nodes
$ #kubectl get pods --all-namespaces-A
```

9. Post this, execute the steps to join the appropriate number of nodes in your cluster. For the cluster in the lab, the `kubectl get nodes` command is as follows.

```
[root@lab-master-2 ~]# kubectl get nodes
NAME             STATUS    ROLES    AGE   VERSION
lab-master-1    Ready    control-plane   24d   v1.27.1
lab-master-2    Ready    control-plane   5d23h v1.27.1
lab-master-3    Ready    control-plane   5d23h v1.27.1
lab-worker-1    Ready    <none>          6d20h v1.27.1
lab-worker-2    Ready    <none>          24d   v1.27.1
lab-worker-4    Ready    <none>          18d   v1.27.1
[root@lab-master-2 ~]#
```



Note: Your Kubernetes environment could be different based on your cluster setup and needs. Refer to official Kubernetes documentation to build a cluster.

Install GPU operators on K8S clusters

The NVIDIA GPU Operator uses the operator framework within Kubernetes to automate the management of all NVIDIA software components needed to provision GPUs. These components include the NVIDIA drivers (to enable CUDA), Kubernetes device plugin for GPUs, the NVIDIA Container Toolkit, automatic node labelling using GFD and DCGM-based monitoring, and others.

Procedure

1. Use the following procedure to install the GPU Operator on Kubernetes clusters.

```
helm repo add nvidia https://helm.ngc.nvidia.com/nvidia
helm repo update
```

2. Run the following command to install GPU Operators using a RedHat toolkit version.

```
helm install --wait --generate-name \
  -n gpu-operator --create-namespace \
  nvidia/gpu-operator \
  --set toolkit-version=1.14.3-ubi8
```

After successful installation of GPU Operator, pods will look like the following.

```
[root@lab-master-1 opt]# kubectl get po -n gpu-operator
NAME                                READY   STATUS    RESTARTS   AGE
gpu-feature-discovery-c7fns         1/1    Running   0           7d21h
gpu-feature-discovery-jwzr5         1/1    Running   0           7d16h
gpu-feature-discovery-psqlh         1/1    Running   0           7d16h
gpu-operator-1702816134-node-feature-discovery-gc-67cb584cswbsp 1/1    Running   2           23d
gpu-operator-1702816134-node-feature-discovery-master-667fzjmjd 1/1    Running   0           4d16h
gpu-operator-1702816134-node-feature-discovery-worker-7n7pk 1/1    Running   2 (5d23h ago) 7d16h
gpu-operator-1702816134-node-feature-discovery-worker-8dhzw 1/1    Running   0           6d19h
gpu-operator-1702816134-node-feature-discovery-worker-cn2mh 1/1    Running   1 (5d23h ago) 6d19h
gpu-operator-1702816134-node-feature-discovery-worker-dcxrp 1/1    Running   3 (5d23h ago) 19d
gpu-operator-1702816134-node-feature-discovery-worker-f72x7 1/1    Running   10 (5d23h ago) 23d
gpu-operator-1702816134-node-feature-discovery-worker-q9wms 1/1    Running   5 (5d23h ago) 23d
gpu-operator-86b987bffc-lj7rs       1/1    Running   1 (2d17h ago) 4d16h
nvidia-container-toolkit-daemonset-4t49p 1/1    Running   0           7d21h
nvidia-container-toolkit-daemonset-fj656 1/1    Running   0           7d16h
nvidia-container-toolkit-daemonset-plwcc 1/1    Running   0           7d16h
nvidia-cuda-validator-dfp9k         0/1    Completed 0           7d21h
nvidia-cuda-validator-lw42c         0/1    Completed 0           7d16h
nvidia-cuda-validator-plhmq         0/1    Completed 0           7d16h
nvidia-dcgm-exporter-6ghbg          1/1    Running   0           7d16h
nvidia-dcgm-exporter-j24z6          1/1    Running   0           7d21h
nvidia-dcgm-exporter-n9tvz          1/1    Running   0           7d16h
nvidia-device-plugin-daemonset-2l8cn 1/1    Running   0           7d16h
nvidia-device-plugin-daemonset-5nrwm 1/1    Running   0           7d16h
nvidia-device-plugin-daemonset-kj9s8 1/1    Running   0           7d21h
nvidia-driver-daemonset-6zm57       1/1    Running   1           19d
nvidia-driver-daemonset-85gxs       1/1    Running   0           7d16h
nvidia-driver-daemonset-nlhm5       1/1    Running   2           23d
nvidia-mig-manager-6n68x            1/1    Running   0           7d21h
nvidia-mig-manager-c6d4t            1/1    Running   0           7d16h
nvidia-mig-manager-zrzcs5           1/1    Running   0           7d16h
nvidia-operator-validator-2tvm9      1/1    Running   0           7d16h
nvidia-operator-validator-krrm6      1/1    Running   0           7d21h
nvidia-operator-validator-w6ltk      1/1    Running   0           7d16h
[root@lab-master-1 opt]#
```

See <https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/23.9.1/getting-started.html#installation-on-red-hat-enterprise-linux> for more information.

Install RAPID Accelerator for Apache Spark

Kubernetes requires a Docker image to run Spark. Everything needed is in the Docker image - Spark, the RAPIDS Accelerator for Spark jars, and the discovery script.

Before you begin

Complete these steps on a client machine that has access to the Kubernetes cluster.

Procedure

1. Download Apache Spark bundle into a local directory and extract the bundle as a directory named `spark`.

2. Upload `rapids-4-spark_<version>.jar` and `getGpusResources.sh` in the `spark` directory.
3. Download the RAPIDS Accelerator for Spark jars and the GPU discovery script.
4. Download the sample `Dockerfile.cuda` in the `spark` directory.

The sample `Dockerfile.CUDA` will copy the `spark` directory's several sub-directories into `/opt/spark/` along with the RAPIDS Accelerator jars and `getGpusResources.sh` into `/opt/sparkRapidsPlugin` inside the Docker image.

Modify the `Dockerfile` to copy your application into the Docker image. Currently the directory in the local machine should look like the following.

```
[root@lab-master-1 opt]# ls
Dockerfile  getGpusResources.sh  rapids-4-spark_2.12-23.12.0.jar  spark
[root@lab-master-1 opt]#
```

5. Build the Docker image with a proper repository name and tag, and then push it to the repository.

```
export IMAGE_NAME=yourprivaterepository/yyy:tag
podman build . -f Dockerfile -t $IMAGE_NAME
podman push $IMAGE_NAME
```



Note: Because the base Operating system is RedHat Linux 8, podman has been used for image build.

See <https://docs.nvidia.com/spark-rapids/user-guide/23.12.1/getting-started/kubernetes.html#getting-started-kubernetes> for more information.

Traditional Analytics without GPUs

For more information about Red Hat OpenShift see the Red Hat documentation at <https://docs.openshift.com/>.

Procedure

1. Follow cluster configuration instructions as outlined on OCP documentation for your cluster.
2. Consider using Kubernetes Operators or Helm charts to simplify the deployment of your Spark applications.



Note: If you are using any other application then follow Kubernetes Operators or Helm charts for those applications.

Hitachi Content Software for File installation instructions

For more information about Hitachi Content Software for File (HCSF) see <https://docs.hitachivantara.com/r/en-us/content-software-for-file/4.0.x/mk-hcsf000/about-the-content-software-for-file-system>.

The following high-level activities need to be completed to validate this solution after you already have Kubernetes or OpenShift platform for compute nodes and HCSF storage nodes configured.

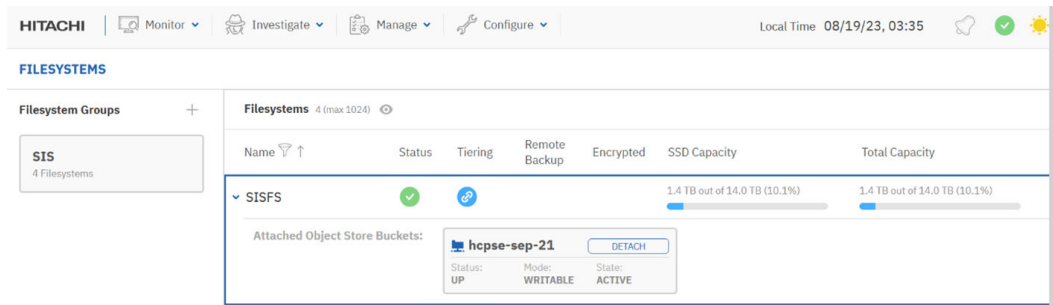
Attach a local Object store

Two local object store buckets can be attached to a filesystem, but only one of the buckets is writable. A local object store bucket is used for both tiering and snapshots. When attaching a new object store bucket to an already tiered filesystem, the existing object-store bucket becomes read-only, and the new object store bucket is read/write.

Multiple object stores allow a range of use cases, including migration to different object stores, object store capacity scaling, and increasing the total tiering capacity of filesystems.

When attaching an object store bucket to a non-tiered filesystem, the filesystem becomes tiered.

For more details see <https://docs.hitachivantara.com/r/en-us/content-software-for-file/4.0.x/mk-hcsf000>.



Create a filesystem on Hitachi Content Software for File storage

Use this procedure to create a filesystem in Hitachi Content Software for File (HCSF).

Procedure

1. From the main menu, select **Manage > Filesystems**.
2. Select **+Create**.
3. In the **Create Filesystem** dialog, set the following:
 - **Name:** Enter a meaningful name for the filesystem.
 - **Group:** Select the filesystem group that fits your filesystem.
 - **Capacity:** Enter the storage size to provision or select **Use All** to provision all the free capacity.

- Optional: If **Thin Provision** is required, select the toggle button, and set the minimum and the maximum capacity of the SSD to use for thin provisioning.

- Optional: If Encryption is required and your HCSF system is deployed with a key management system (KMS), select the toggle button.
- Select **Save**.

Install the HCSF Client

To use the HCSF filesystems from a client, call the `mount` command. The `mount` command automatically installs the software version, and there is no need to join the client to the cluster.

Procedure

- To mount a filesystem using this method, install the HCSF agent from one of the backend instances and then mount the filesystem.

```
# Agent Installation (one time)
$ curl http://Backend-1:14000/dist/v1/install | sh
# Creating a mount point (one time)
$ mkdir -p /mnt/weka
# Mounting a filesystem
$ mount -t wekafs -o net=eth0 backend-1/my_fs /mnt/weka
```



Note: Eight backend IP addresses are available (192.168.1.1 to 192.168.1.8), and worker nodes can use any one of these backend IP addresses to access the storage.

- Run the following command on worker nodes to install the HCSF agent.

```
root@lab-worker-3 ~]# curl http://192.168.1.1:14000/dist/v1/install | sh % Total
% Received % Xferd Average Speed Time Time Time Current Dload Upload Total Spent
Left Speed 100 1421 100 1421 0 0 1387k 0 ---:---:-- --:---:-- --: 1387k
Downloading WekaIO CLI 4.0.1.36-hcsf % Total % Received % Xferd Average Speed
Time Time Time Current Dload Upload Total Spent Left Speed 100 86.8M 100 86.8M 0
```



```
0 1240M 0 --:---:-- --:---:-- --:---:-- 1240M Installing... Installing agent of
version 4.0.1.36-hcsf Waiting for agent service to be ready Installation
finished successfully WekaIO CLI 4.0.1.36-hcsf is now installed
```

Next steps

Repeat these steps for other worker nodes.

Mount the Filesystem

Use the following procedure to mount the filesystem.

Procedure

1. See <https://docs.weka.io/v/4.0/fs/mounting-filesystems>.

```
[root@lab-worker-3 ~]# mount -t wekafs -o num_cores=8 -o net=ens99f1 192.168.1.1/
SISFS /mnt/weka
Mounting 192.168.1.1/SISFS on /mnt/weka
Basing mount on container client
Downloading [1/17] http://192.168.1.1:14000/dist/v1/image/weka-container-2.3.0-
8939ec123986ee716f117c7254ede4ea.squashfs
Downloading [2/17] http://192.168.1.1:14000/dist/v1/image/container-s3-weka-
release-261683bd60a19fadfabe3c126c2365ed.squashfs
Downloading [3/17] http://192.168.1.1:14000/dist/v1/image/dependencies-1.0.0-
336aa94d40e1e5223f8da59ddca4a035.squashfs
Downloading [4/17] http://192.168.1.1:14000/dist/v1/image/weka-ganesha-
12b55f846beaad35e6131fb498f53b7d.squashfs
Downloading [5/17] http://192.168.1.1:14000/dist/v1/image/driver-uo-pci-generic-
1.0.0-d644841c998c88e4fc66529e4484dbb6.squashfs
Downloading [6/17] http://192.168.1.1:14000/dist/v1/image/dashboard-
9e7bf5de9ef5dd65a948ca3a5c35ca7a.squashfs
Downloading [7/17] http://192.168.1.1:14000/dist/v1/image/container-ganesha-weka-
3.5.0.8-57d471a087cd656535867407094510f3.squashfs
Downloading [8/17] http://192.168.1.1:14000/dist/v1/image/weka-driver-1.0.0-
d6037cd081b2d40aff6ccb8e3616ac2d.squashfs
Downloading [9/17] http://192.168.1.1:14000/dist/v1/image/weka-s3-
a5681a134e0af48b86d2458a89f9da87.squashfs
Downloading [10/17] http://192.168.1.1:14000/dist/v1/image/ofed-
1b295470b56ec067af7340f2cca7e27a.squashfs
Downloading [11/17] http://192.168.1.1:14000/dist/v1/image/weka-driver-igb-uo-
4.0.0-45a2501c82e695e4254356bc75e94383.squashfs
Downloading [12/17] http://192.168.1.1:14000/dist/v1/image/container-samba-weka-
4.7.12.1-c5471da5154clad418a3ab06e9dfefbd.squashfs
Downloading [13/17] http://192.168.1.1:14000/dist/v1/image/weka-hostside-
5edfa1328eef57248b68bf658c3a1c5e.squashfs
Downloading [14/17] http://192.168.1.1:14000/dist/v1/image/ui-1.0.0-
5bc747765d326e61c3488285822f459.squashfs
Downloading [15/17] http://192.168.1.1:14000/dist/v1/image/weka-node-
6c6442127191b70a499aed9270bd8644.squashfs
Downloading [16/17] http://192.168.1.1:14000/dist/v1/image/api-
```

```

c39357c20a93420e9fa61058d8c71664.squashfs
Downloading [17/17] http://192.168.1.1:14000/dist/v1/image/weka-samba-
50c4972aaf8bf5c562263359b5d5d910.squashfs
Finished getting version 4.0.1.36-hcsf
Creating Weka container 'client' in version 4.0.1.36-hcsf
Applying resources
Starting container 'client'
The NetworkManager service is running, we aren't allowed to stop it, but we were
told to run anyway, so trying
Waiting for container 'client' to join cluster
Container "client" is ready (pid = 387226)
Calling the mount command
Mount completed successfully

```

2. Validate the filesystem using `df -Th`.

```

[root@lab-worker-3 ~]# df -Th | egrep -i weka
/dev/loop0          xfs          2.0G   47M   2.0G   3% /opt/weka/logs
tmpfs              tmpfs       472G   12K   472G   1% /opt/weka/data/agent/
tmpfss/cleanup
tmpfs              tmpfs       472G   2.0G   471G   1% /opt/weka/data/agent/
tmpfss/client-persistent-tmpfs
tmpfs              tmpfs       472G   4.0K   472G   1% /opt/weka/data/agent/
tmpfss/cleanup_before_stop_and_delete
SISFS              wekafs      13T    1.3T   12T    11% /mnt/weka`

```

Configure the HCSF CSI plugin

Procedure

1. Add the `csi-wekafs` repository.

```
helm repo add csi-wekafs https://weka.github.io/csi-wekafs
```

2. Install the HCSF CSI plugin.

```
helm install csi-wekafs csi-wekafs/csi-wekafspugin --namespace csi-wekafs --
create-namespace [--set selinuxSupport=<off | mixed | enforced>]
```

Configure a Secret for StorageClass

Procedure

1. Create a secret data file.

```

apiVersion: v1
kind: Secret
metadata:
  name: csi-wekafs-api-secret
  namespace: csi-wekafs

```

```

type: Opaque
data:
  # A username to connect to the cluster API (base64-encoded)
  username: YWRtaW4=
  # A password to connect to the cluster API (base64-encoded)
  password: YWRtaW4=
  # An organization to connect to (default Root, base64-encoded)
  organization: Um9vdA==
  # A comma-separated list of cluster management endpoints. Format: <IP:port>
  (base64-encoded)
  # It is recommended to configure at least 2 management endpoints (cluster
  backend nodes), or a load-balancer if used
  # e.g. 172.31.15.113:14000,172.31.12.91:14000
  endpoints:
  MTcyLjMxLjQxLjU0OjE0MDAwLDE3Mi4zMS40Ny4xNTI6MTQwMDAsMTcyLjMxLjM4LjI1MDoxNDAwMCwxN
  zIuMzEuNDcuMTU1OjE0MDAwLDE3Mi4zMS4zMy45MToxNDAwMCwxNzIuMzEuMzguMTU1OjE0MDAwCg==
  # protocol to use for API connection (may be either http or https, base64-
  encoded)
  scheme: aHR0cA==
  # for multiple clusters setup, set a specific container name (base64-encoded)
  localContainerName: ""

```

Secrets can be created with the following command:

```
#echo -n "<string>" | base64 to decoded secret string for values
```

2. Apply the secret data and validate that it is created successfully.

```

# apply the secret .yaml file
$ kubectl apply -f csi-wekafs-api-secret.yaml

# Check the secret was successfully created
$ kubectl get secret csi-wekafs-api-secret -n csi-wekafs

```

NAME	TYPE	DATA	AGE
csi-wekafs-api-secret	Opaque	5	7m

Create a StorageClass

Create Storage Class using the following yaml file.

Use the secret file that has already been created.

```

apiVersion: v1
items:
- allowVolumeExpansion: true
  apiVersion: storage.k8s.io/v1
  kind: StorageClass
  metadata:
  parameters:
    capacityEnforcement: HARD
    csi.storage.k8s.io/controller-expand-secret-name: csi-wekafs-api-secret

```

```

csi.storage.k8s.io/controller-expand-secret-namespace: csi-wekafs
csi.storage.k8s.io/controller-publish-secret-name: csi-wekafs-api-secret
csi.storage.k8s.io/controller-publish-secret-namespace: csi-wekafs
csi.storage.k8s.io/node-publish-secret-name: csi-wekafs-api-secret
csi.storage.k8s.io/node-publish-secret-namespace: csi-wekafs
csi.storage.k8s.io/node-stage-secret-name: csi-wekafs-api-secret
csi.storage.k8s.io/node-stage-secret-namespace: csi-wekafs
csi.storage.k8s.io/provisioner-secret-name: csi-wekafs-api-secret
csi.storage.k8s.io/provisioner-secret-namespace: csi-wekafs
filesystemName: SISFS
volumeType: dir/v1
provisioner: csi.weka.io
reclaimPolicy: Delete
volumeBindingMode: Immediate
- allowVolumeExpansion: true
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  parameters:
    capacityEnforcement: HARD
    csi.storage.k8s.io/controller-expand-secret-name: csi-wekafs-api-secret
    csi.storage.k8s.io/controller-expand-secret-namespace: csi-wekafs
    csi.storage.k8s.io/controller-publish-secret-name: csi-wekafs-api-secret
    csi.storage.k8s.io/controller-publish-secret-namespace: csi-wekafs
    csi.storage.k8s.io/node-publish-secret-name: csi-wekafs-api-secret
    csi.storage.k8s.io/node-publish-secret-namespace: csi-wekafs
    csi.storage.k8s.io/node-stage-secret-name: csi-wekafs-api-secret
    csi.storage.k8s.io/node-stage-secret-namespace: csi-wekafs
    csi.storage.k8s.io/provisioner-secret-name: csi-wekafs-api-secret
    csi.storage.k8s.io/provisioner-secret-namespace: csi-wekafs
    filesystemName: SISFS
    volumeType: dir/v1
  provisioner: csi.weka.io
  reclaimPolicy: Delete
  volumeBindingMode: Immediate
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""

```

Create a Persistent Volume Claim

Procedure

1. Create a PVC yaml file.

```

apiVersion: v1
kind: PersistentVolumeClaim metadata:
annotations:
pv.kubernetes.io/bind-completed: "yes" pv.kubernetes.io/bound-by-controller:

```

```
"yes" volume.beta.kubernetes.io/storage-provisioner: csi.weka.io
volume.kubernetes.io/storage-provisioner: csi.weka.io
finalizers:
-   kubernetes.io/pvc-protection
-   provisioner.storage.kubernetes.io/cloning-protection name: pvc-wekafs-dir-
dynamic
namespace: default resourceVersion: "77704317"
spec:
accessModes:
-   ReadWriteMany resources:
requests: storage: 1Ti
storageClassName: storageclass-wekafs-dir-api volumeMode: Filesystem
volumeName: pvc-8fd6ed4e-26d4-4362-89c1-bb30e0cfad82 status:
accessModes:
-   ReadWriteMany capacity:
storage: 1Ti phase: Bound
```

2. Verify PVC details.

```
$ kubectl get pv
```

Configure Spark Clusters

Set up an OpenShift cluster with appropriate resources (CPU and memory) for running Spark applications. Consider using Kubernetes Operators or Helm charts to simplify the deployment of your Spark clusters. We have deployed a spark cluster with a Helm chart using a bitnami image.

Procedure

1. Use the following syntax to install Spark.

```
helm install my-release oci://registry-1.docker.io/bitnamicharts/spark
```

The following is an example `values.yaml` file:

```
---
worker:
  extraVolumes:
    - name: spark-data
      persistentVolumeClaim:
        claimName: spark-data
  extraVolumeMounts:
    - name: spark-data
      mountPath: '/sparkdata'
```

In this solution, the Content Software for File storage filesystem is only mounted in worker nodes, so the following values are updated.

```
[root@ocp-admin-ws hitachi]# cat values.yaml
---
worker:
  extraVolumes:
    - name: pvc-8fd6ed4e-26d4-4362-89c1-bb30e0cfad82
      persistentVolumeClaim:
        claimName: pvc-wekafs-dir-dynamic
  extraVolumeMounts:
    - name: pvc-8fd6ed4e-26d4-4362-89c1-bb30e0cfad82
      mountPath: '/sparkdata'
[root@ocp-admin-ws hitachi]#
```

For example:

```
[root@ocp-admin-ws hitachi# helm install my-test-release-3
oci://registry-1.docker.io/bitnamicharts/spark --values values.yaml
3 pods created
```

```
[root@ocp-admin-ws hitachi]# oc get pods
NAME                                READY   STATUS    RESTARTS   AGE
ndfs-operator-86d7cbbc68-jsqtp      1/1     Running   1           58d
my-test-release-2-spark-master-0    1/1     Running   0           4d19h
my-test-release-2-spark-master-1    1/1     Running   0           4d19h
my-test-release-2-spark-master-2    1/1     Running   0           4d19h
my-test-release-2-spark-worker-0    0/1     Pending   0           3d19h
my-test-release-3-spark-master-0    1/1     Running   0           22m
my-test-release-3-spark-worker-0    1/1     Running   0           22m
my-test-release-3-spark-worker-1    1/1     Running   0           21m
[root@ocp-admin-ws hitachi]#
```

2. Verify that the PV is mounted under /sparkdata.

```
[root@lab-worker-2 hitachi]# oc exec -it my-test-release-3-spark-worker-0 -
- /bin/bash
1001@my-test-release-3-spark-worker-0:/opt/bitnami/spark$ df -Th
Filesystem      Type      Size  Used Avail Use% Mounted on
overlay         overlay   270G   81G  190G  30% /
tmpfs           tmpfs     64M    0   64M   0% /dev
tmpfs           tmpfs    252G    0  252G   0% /sys/fs/cgroup
shm            tmpfs     64M    0   64M   0% /dev/shm
tmpfs           tmpfs    252G   83M  252G   1% /etc/passwd
SISFS          wekafs    1.0T   16K  1.0T   1% /sparkdata
```

```
1001@my-test-release-3-spark-worker-0:/opt/bitnami/spark$ df -Th
Filesystem      Type      Size  Used Avail Use% Mounted on
overlay         overlay   270G   81G  190G  30% /
tmpfs          tmpfs     64M    0   64M   0% /dev
tmpfs          tmpfs    252G    0  252G   0% /sys/fs/cgroup
shm            tmpfs     64M    0   64M   0% /dev/shm
tmpfs          tmpfs    252G   83M  252G   1% /etc/passwd
SISFS          wekafs    1.0T   16K   1.0T   1% /sparkdata
/dev/mapper/rhel-root xfs       270G   81G  190G  30% /etc/hosts
tmpfs          tmpfs    503G   20K  503G   1% /run/secrets/kubernetes.io/serviceaccount
tmpfs          tmpfs    252G    0  252G   0% /proc/acpi
tmpfs          tmpfs    252G    0  252G   0% /proc/scsi
tmpfs          tmpfs    252G    0  252G   0% /sys/firmware
1001@my-test-release-3-spark-worker-0:/opt/bitnami/spark$ cd /sparkdata/
1001@my-test-release-3-spark-worker-0:/sparkdata$ ls -la
total 4
drwxrws--- 1 root 1001  0 Jul  6 16:21 .
dr-xr-xr-x 1 root root  67 Jul 10 11:45 ..
-rw-rw-r-- 1 root 1001 672 Jul  6 16:25 temp.txt
1001@my-test-release-3-spark-worker-0:/sparkdata$
```

```
/dev/mapper/rhel-root xfs      270G   81G  190G  30% /etc/hosts
tmpfs                 tmpfs   503G   20K  503G   1% /run/secrets/kubernetes.io/
serviceaccount
tmpfs                 tmpfs   252G    0  252G   0% /proc/acpi
tmpfs                 tmpfs   252G    0  252G   0% /proc/scsi
tmpfs                 tmpfs   252G    0  252G   0% /sys/firmware
1001@my-test-release-3-spark-worker-0:/opt/bitnami/spark$ cd /sparkdata/
1001@my-test-release-3-spark-worker-0:/data$ ls -la
total 4
drwxrws--- 1 root 1001  0 Jul  6 16:21 .
dr-xr-xr-x 1 root root  67 Jul 10 11:45 ..
-rw-rw-r-- 1 root 1001 672 Jul  6 16:25 temp.txt
1001@my-test-release-3-spark-worker-0:/sparkdata$
```

Solution testing

The following test cases were performed on this solution.

AI/GenAI with GPUs (using K8S)

Test Spark Job with GPUs vs CPUs on the K8S Cluster

Spark offers various methods for submitting and executing your data processing tasks, catering to different environments and user preferences. Here we followed the classic way to launch Spark applications, Spark-Submit.

The following is an example Spark job using GPUs in cluster mode. This Spark job uses the *kmeans* clustering algorithm. This Spark job uses Kubernetes functionality called Persistent Volume for storing large datasets. Persistent volume has been created using Hitachi Content Software for File (HCSF). See [Solution validation \(on page 41\)](#) for more information on HCSF configuration. Use the following Spark job as an example for your environment. To tune the Spark submit job, optimize values for `spark.executor.cores`, `spark.task.resource.gpu.amount`, `spark.driver.memory`, and `spark.executor.memory`.

Make sure service accounts are created with sufficient permission. If not, use the following procedure to create a service account.

```
kubectl create serviceaccount apache-spark-driver -n spark
kubectl create clusterrolebinding spark-role --clusterrole=edit --
serviceaccount=spark:apache-spark-driver --namespace=spark
```

To support the Spark job that follows, a Persistent Volume Claim (PVC) named `pvc-wekafs-dir-1` has been created on HCSF. This PVC is used for dataset storage. HCSF demonstrates superior performance over locally attached disks, particularly with larger datasets. See [Create a Persistent Volume Claim \(on page 52\)](#) for more information.

See the *Rapids Accelerator for Apache Spark Tuning Guide* at <https://docs.nvidia.com/spark-rapids/user-guide/23.10/tuning-guide.html> for more information.

The following example job is given as a reference. Change the parameters according to your data size and dimensions.

The following Spark job is used to run kmeans using GPUs.

```
spark-submit --master k8s://https://172.21.1.15:8443 \
--deploy-mode cluster --name kmeans --conf
spark.kubernetes.authenticate.driver.serviceAccountName=apache-spark-driver \
--conf spark.kubernetes.namespace=spark --conf spark.executor.instances=3 --conf
spark.kubernetes.container.image=$IMAGE_NAME \
--conf spark.kubernetes.executor.volumes.persistentVolumeClaim.pvc-wekafs-dir-
1.options.claimName=pvc-wekafs-dir-1 \
--conf spark.kubernetes.driver.volumes.persistentVolumeClaim.pvc-wekafs-dir-
1.options.claimName=pvc-wekafs-dir-1 \
--conf spark.kubernetes.executor.volumes.persistentVolumeClaim.pvc-wekafs-dir-
1.mount.path=/opt/spark/work-dir \
--conf spark.kubernetes.driver.volumes.persistentVolumeClaim.pvc-wekafs-dir-
1.mount.path=/opt/spark/work-dir \
--conf spark.kubernetes.executor.volumes.persistentVolumeClaim.pvc-wekafs-dir-
1.mount.readOnly=false \
--conf spark.kubernetes.driver.volumes.persistentVolumeClaim.pvc-wekafs-dir-
1.mount.readOnly=false \
--conf spark.kubernetes.driver.pod.name=kmeans-driver \
--conf spark.kubernetes.pyspark.pythonVersion=3 \
--conf spark.kubernetes.memoryOverheadFactor=0.2 \
--conf spark.executor.memoryOverhead=8g \
--conf spark.executor.cores=32 \
--conf spark.task.cpus=1 \
--conf spark.driver.memory=40g \
--conf spark.executor.memory=40g \
--conf spark.plugins=com.nvidia.spark.SQLPlugin \
--conf spark.rapids.sql.incompatibleOps.enabled=true \
--conf spark.rapids.sql.variableFloatAgg.enabled=true \
--conf spark.executor.resource.gpu.discoveryScript=/opt/sparkRapidsPlugin/
getGpusResources.sh \
--conf spark.executor.resource.gpu.vendor=nvidia.com \
--conf spark.executor.resource.gpu.amount=1 \
```



```
--conf spark.rapids.sql.concurrentGpuTasks=2 \
--conf spark.executor.extraClassPath=/opt/sparkRapidsPlugin/rapids-4-spark_2.12-23.12.0.jar \
--conf spark.driver.extraClassPath=/opt/sparkRapidsPlugin/rapids-4-spark_2.12-23.12.0.jar \
--conf spark.task.resource.gpu.amount=0.03125 \
--conf spark-webhook-enabled=true local:///opt/spark/examples/src/main/python/kmeans.py /opt/spark/work-dir/agaricus.txt.train 2 .01
```

The following Spark job is used to run kmeans using CPUs.

```
spark-submit --master k8s://https://172.21.1.15:8443 \
--deploy-mode cluster --name kmeans --conf
spark.kubernetes.authenticate.driver.serviceAccountName=apache-spark-driver \
--conf spark.kubernetes.namespace=spark --conf spark.executor.instances=50 --conf
spark.kubernetes.container.image=$IMAGE_NAME \
--conf spark.kubernetes.executor.volumes.persistentVolumeClaim.pvc-wekafs-dir-1.options.claimName=pvc-wekafs-dir-1 \
--conf spark.kubernetes.driver.volumes.persistentVolumeClaim.pvc-wekafs-dir-1.options.claimName=pvc-wekafs-dir-1 \
--conf spark.kubernetes.executor.volumes.persistentVolumeClaim.pvc-wekafs-dir-1.mount.path=/opt/spark/work-dir \
--conf spark.kubernetes.driver.volumes.persistentVolumeClaim.pvc-wekafs-dir-1.mount.path=/opt/spark/work-dir \
--conf spark.kubernetes.executor.volumes.persistentVolumeClaim.pvc-wekafs-dir-1.mount.readOnly=false \
--conf spark.kubernetes.driver.volumes.persistentVolumeClaim.pvc-wekafs-dir-1.mount.readOnly=false \
--conf spark.kubernetes.driver.pod.name=kmeans-driver \
--conf spark.kubernetes.pyspark.pythonVersion=3 \
--conf spark.driver.memory=40g \
--conf spark.executor.memory=40g \
--conf spark-webhook-enabled=true local:///opt/spark/examples/src/main/python/kmeans.py /opt/spark/work-dir/agaricus.txt.train 2 .01
```



Note: RAPIDS Accelerator for Spark only accelerates Spark SQL (for example, for preprocessing and I/O from various file formats like parquet) and not the machine learning computation part. For the latter, try using the `spark-rapids-ml` library from NVIDIA. See <https://github.com/NVIDIA/spark-rapids-ml> for more information.

While using the `Spark-Rapids-ML` library, consider the following suggestions:

1. Set `spark.task.resource.gpu.amount=1` in the `spark submit` command. Setting its value less than 1 is not compatible with `spark-rapids-ml` which requires one training task per GPU.
2. Set `spark.rapids.ml.uvm.enabled=true`. This allows oversubscribing GPU memory for transient data copying.
3. To optimize data loading to GPUs, use the GPU-optimized python workers that are

enabled by setting the following parameters.

```
spark.python.daemon.module=rapids.daemon
spark.rapids.sql.python.gpu.enabled=true
spark.executorEnv.PYTHONPATH=${rapids_jar}
```

4. For high dimensional data consider changing the value for `spark.sql.execution.arrow.maxRecordsPerBatch`. For example, if your data consists of 2 million rows and three thousand rows, consider keeping its value at 100000. According to data dimension, increase or decrease its value.

Traditional Analytics without GPUs (using OCP)

Run Spark Jobs

Run a sample Spark job on the local filesystem

In our test setup, a local filesystem was configured in an SSD disk with xfs filesystem. The performance can vary depending on the type of filesystem and type of hard drives. A 1 TB NVMe SSD disk was used for the performance test.

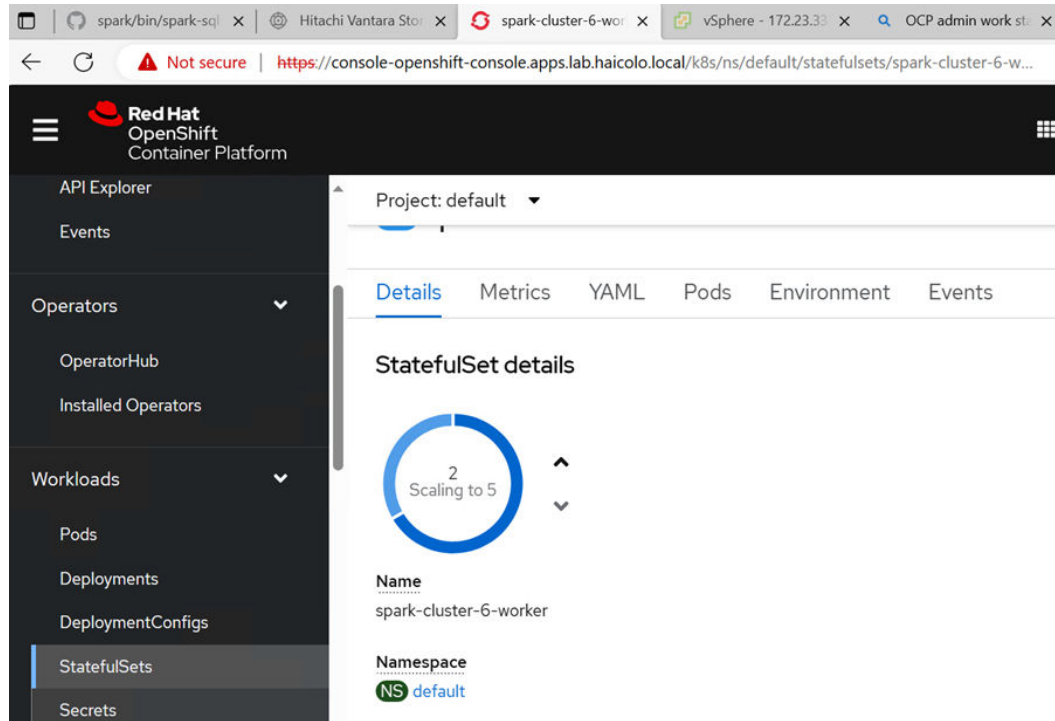
```
#oc exec --stdin --tty spark-cluster-worker-0 -- spark-submit
/opt/bitnami/spark/examples/src/main/python/wordcount.py /tmp/AirlineReviews10.csv
```

Run the same Spark job on the Content Software for File filesystem

```
#oc exec --stdin --tty spark-cluster-worker-0 -- spark-submit
/opt/bitnami/spark/examples/src/main/python/wordcount.py /spark-data/
AirlineReviews10.csv
```

Spark Cluster Scaling

The auto-scaling capabilities of Spark on OpenShift were tested. It was validated that the cluster could scale up or down based on the workload demand.



Conclusion

The Artificial Intelligence using Apache Spark on UCP with HCSF solution is designed to run Advanced Analytics (AI/ML), Generative AI, and Traditional Analytics applications and associated workloads. This innovative solution empowers organizations to efficiently develop and deploy applications tailored to their specific needs. It allows organizations to process their domain-specific data in a manner that is highly relevant and uniquely customized to their individual requirements.

The solution provides a rapidly deployable infrastructure design with a scalable, modular, high-performance, and non-blocking architecture, ideal for diverse analytical workloads. This validated and tested solution accelerates time-to-value, minimizes uncertainty, and mitigates risks through its proven design, ensuring a seamless and efficient analytics experience.

Apache Spark is a game-changer for Advanced Analytics (AI/ML) and Generative AI application development. With its unique capabilities such as SQL processing, built-in machine learning library MLlib, and integration with popular AI frameworks such as TensorFlow and PyTorch, it can seamlessly execute Advanced Analytics (AI/ML) algorithms and assist data curation for Generative AI workloads. Apache Spark stands out for its capacity to efficiently handle large-scale data processing using in-memory computation. Its flexibility to seamlessly integrate with various data sources and various cluster managers positions it as the go-to solution for organizations in pursuit of scalable and efficient solutions. Furthermore, NVIDIA's RAPIDS accelerator for Apache Spark combines the power of GPUs to accelerate the scale of the Sparks' distributed computing, to enable highly efficient data processing.

This reference architecture provides guidelines and a validated design for analytics workload. The topics discussed include the following:

- Various types of workloads – traditional, advanced analytics, and self-learning.
- Business use cases – how the workloads are put together to serve various industry verticals such as manufacturing, financial services, and healthcare.
- The impact of validated architecture on businesses.
- A description of primary solution components, and the purpose of the component in the whole stack.
- The solution architecture, which outlines how the components are put together to validate the architecture.
- An example of sizing considerations by considering a typically observed workload in the field.
- A detailed description of solution validation, including execution of test cases, and a comparison of results with and without HCSF.

With this reference architecture, Hitachi Vantara enables organizations to deliver analytics solutions built on world-class compute, network, and storage infrastructure. This solution delivers great time to value and is efficient to operate. The UCP with HCSF solution running containerized Apache Spark makes it easy for organizations to deliver meaningful insights from their data.

Hitachi Vantara

Corporate Headquarters
2535 Augustine Drive
Santa Clara, CA 95054 USA



HitachiVantara.com/contact