Modern Data Analytics

NetApp Solutions

NetApp
September 24, 2022

Table of Contents

NetApp Modern Data Analytics Solutions

NetApp Storage Solutions for Apache Spark

TR-4570: NetApp Storage Solutions for Apache Spark: Architecture, Use Cases, and Performance Results

Rick Huang, Karthikeyan Nagalingam, NetApp

This document focuses on the Apache Spark architecture, customer use cases, and the NetApp storage portfolio related to big data analytics and artificial intelligence (AI). It also presents various testing results using industry-standard AI, machine learning (ML), and deep learning (DL) tools against a typical Hadoop system so that you can choose the appropriate Spark solution. To begin, you need a Spark architecture, appropriate components, and two deployment modes (cluster and client).

This document also provides customer use cases to address configuration issues, and it discusses an overview of the NetApp storage portfolio relevant to big data analytics and AI, ML, and DL with Spark. We then finish with testing results derived from Spark-specific use cases and the NetApp Spark solution portfolio.

Customer challenges

This section focuses on customer challenges with big data analytics and AI/ML/DL in data growth industries such as retail, digital marketing, banking, discrete manufacturing, process manufacturing, government, and professional services.

Unpredictable performance

Traditional Hadoop deployments typically use commodity hardware. To improve performance, you must tune the network, operating system, Hadoop cluster, ecosystem components such as Spark, and hardware. Even if you tune each layer, it can be difficult to achieve desired performance levels because Hadoop is running on commodity hardware that was not designed for high performance in your environment.

Media and node failures

Even under normal conditions, commodity hardware is prone to failure. If one disk on a data node fails, the Hadoop master by default considers that node to be unhealthy. It then copies specific data from that node over the network from replicas to a healthy node. This process slows down the network packets for any Hadoop jobs. The cluster must then copy the data back again and remove the over- replicated data when the unhealthy node returns to a healthy state.

Hadoop vendor lock-in

Hadoop distributors have their own Hadoop distribution with their own versioning, which locks in the customer to those distributions. However, many customers require support for in-memory analytics that does not tie the customer to specific Hadoop distributions. They need the freedom to change distributions and still bring their analytics with them.

Lack of support for more than one language

Customers often require support for multiple languages in addition to MapReduce Java programs to run their jobs. Options such as SQL and scripts provide more flexibility for getting answers, more options for organizing and retrieving data, and faster ways of moving data into an analytics framework.

Difficulty of use

For some time, people have complained that Hadoop is difficult to use. Even though Hadoop has become simpler and more powerful with each new version, this critique has persisted. Hadoop requires that you understand Java and MapReduce programming patterns, a challenge for database administrators and people with traditional scripting skill sets.

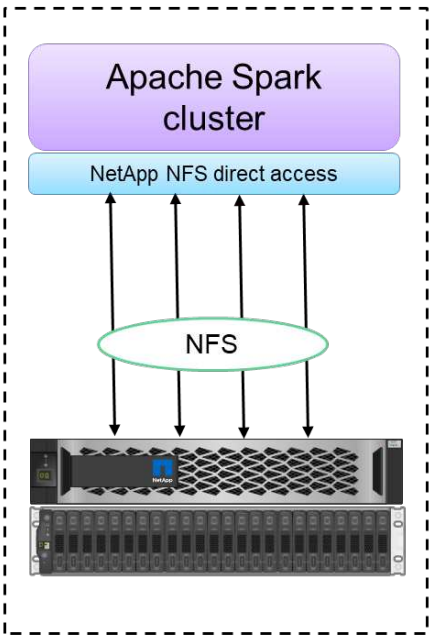
Complicated frameworks and tools

Enterprises AI teams face multiple challenges. Even with expert data science knowledge, tools and frameworks for different deployment ecosystems and applications might not translate simply from one to another. A data science platform should integrate seamlessly with corresponding big data platforms built on Spark with ease of data movement, reusable models, code out of the box, and tools that support best practices for prototyping, validating, versioning, sharing, reusing, and quickly deploying models to production.

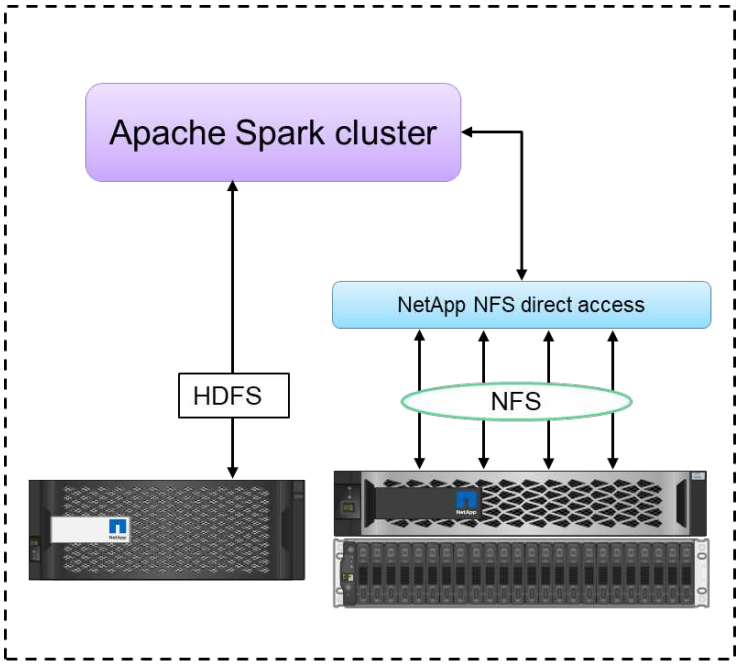
Why choose NetApp?

NetApp can improve your Spark experience in the following ways:

- NetApp NFS direct access (shown in the figure below) allows customers to run big-data-analytics jobs on their existing or new NFSv3 or NFSv4 data without moving or copying the data. It prevents multiple copies of data and eliminates the need to sync the data with a source.
- More efficient storage and less server replication. For example, the NetApp E-Series Hadoop solution requires two rather than three replicas of the data, and the FAS Hadoop solution requires a data source but no replication or copies of data. NetApp storage solutions also produce less server-to-server traffic.
- Better Hadoop job and cluster behavior during drive and node failure.
- Better data-ingest performance.



Configuration 1: NFS as primary storage



Configuration 2: HDFS and NFS in single Spark cluster

For example, in the financial and healthcare sector, the movement of data from one place to another must meet legal obligations, which is not an easy task. In this scenario, NetApp NFS direct access analyzes the financial and healthcare data from its original location. Another key benefit is that using NetApp NFS direct

access simplifies protecting Hadoop data by using native Hadoop commands and enabling data protection workflows with the rich data management portfolio from NetApp.

NetApp NFS direct access provides two kinds of deployment options for Hadoop/Spark clusters:

- By default, Hadoop or Spark clusters use the Hadoop Distributed File System (HDFS) for data storage and the default file system. NetApp NFS direct access can replace the default HDFS with NFS storage as the default file system, enabling direct analytics on NFS data.
- In another deployment option, NetApp NFS direct access supports configuring NFS as additional storage along with HDFS in a single Hadoop or Spark cluster. In this case, the customer can share data through NFS exports and access it from the same cluster along with HDFS data.

The key benefits of using NetApp NFS direct access include the following:

- Analyzing the data from its current location, which prevents the time- and performance-consuming task of moving analytics data to a Hadoop infrastructure such as HDFS.
- Reducing the number of replicas from three to one.
- Enabling users to decouple compute and storage to scale them independently.
- Providing enterprise data protection by leveraging the rich data management capabilities of ONTAP.
- Certification with the Hortonworks data platform.
- Enabling hybrid data analytics deployments.
- Reducing backup time by leveraging dynamic multithread capability.

See TR-4657: NetApp hybrid cloud data solutions - Spark and Hadoop based on customer use cases for backing up Hadoop data, backup and disaster recovery from the cloud to on-premises, enabling DevTest on existing Hadoop data, data protection and multicloud connectivity, and accelerating analytics workloads.

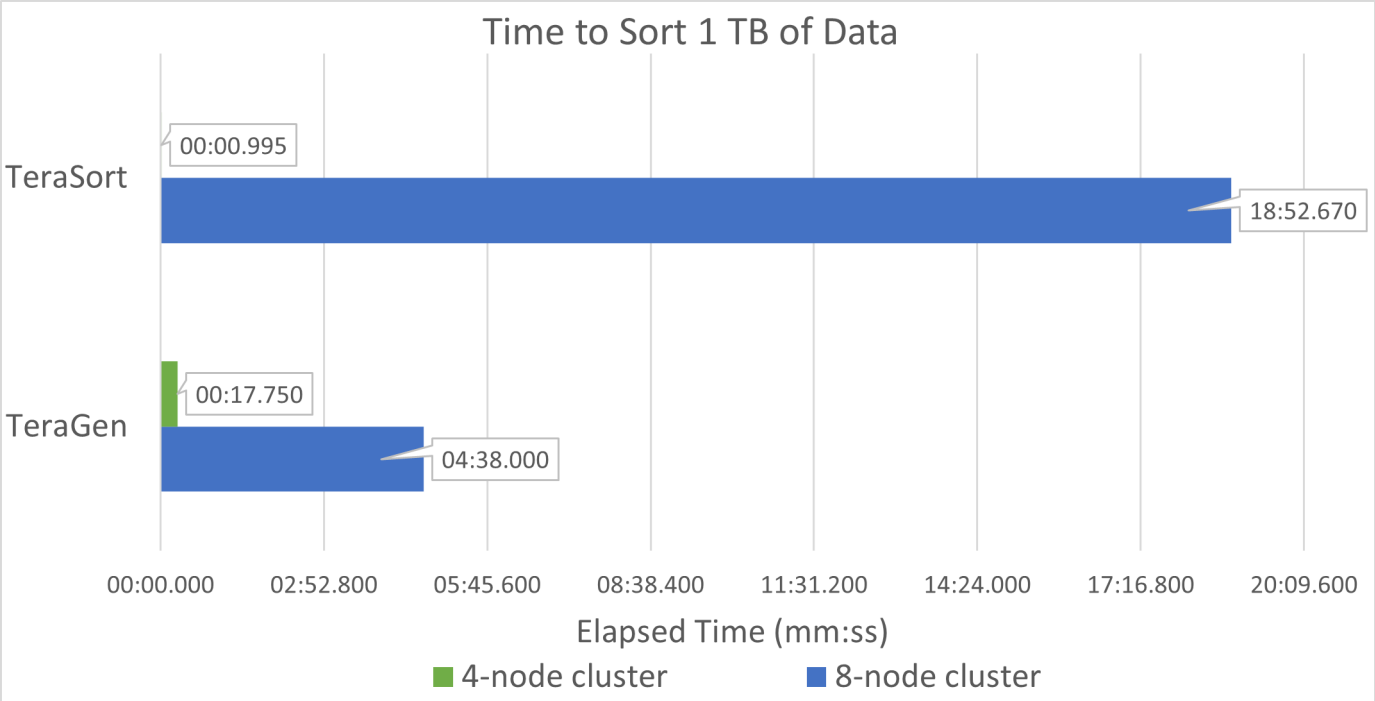
The following sections describe storage capabilities that are important for Spark customers.

Storage tiering

With Hadoop storage tiering, you can store files with different storage types in accordance with a storage policy. Storage types include `hot`, `cold`, `warm`, `all_ssd`, `one_ssd`, and `lazy_persist`.

We performed validation of Hadoop storage tiering on a NetApp AFF storage controller and an E-Series storage controller with SSD and SAS drives with different storage policies. The Spark cluster with AFF-A800 has four compute worker nodes, whereas the cluster with E-Series has eight.

The following figure shows the performance of NetApp solutions for a Hadoop SSD.



- The baseline NL-SAS configuration used eight compute nodes and 96 NL-SAS drives. This configuration generated 1TB of data in 4 minutes and 38 seconds. See TR-3969 NetApp E-Series Solution for Hadoop for details on the cluster and storage configuration.
- Using TeraGen, the SSD configuration generated 1TB of data 15.66x faster than the NL-SAS configuration. Moreover, the SSD configuration used half the number of compute nodes and half the number of disk drives (24 SSd drives in total). Based on the job completion time, it was almost twice as fast as the NL-SAS configuration.
- Using TeraSort, the SSD configuration sorted 1TB of data 1138.36 times more quickly than the NL-SAS configuration. Moreover, the SSD configuration used half the number of compute nodes and half the number of disk drives (24 SSd drives in total). Therefore, per drive, it was approximately three times faster than the NL-SAS configuration.

Performance scaling - Scale out

When you need more computation power from a Hadoop cluster in an AFF solution, you can add data nodes with an appropriate number of storage controllers. NetApp recommends starting with four data nodes per storage controller array and increasing the number to eight data nodes per storage controller, depending on workload characteristics.

AFF and FAS are perfect for in-place analytics. Based on computation requirements, you can add node managers, and non-disruptive operations allow you to add a storage controller on demand without downtime. We offer rich features with AFF and FAS, such as NVME media support, guaranteed efficiency, data reduction, QOS, predictive analytics, cloud tiering, replication, cloud deployment, and security. To help customers meet their requirements, NetApp offers features such as file system analytics, quotas, and on-box load balancing with no additional license costs. NetApp has better performance in the number of concurrent jobs, lower latency, simpler operations, and higher gigabytes per second throughput than our competitors. Furthermore, NetApp Cloud Volumes ONTAP runs on all three major cloud providers.

Performance scaling - Scale up

Scale-up features allow you to add disk drives to AFF, FAS, and E-Series systems when you need additional storage capacity. With Cloud Volumes ONTAP, scaling storage to the PB level is a combination of two factors:

tiering infrequently used data to object storage from block storage and stacking Cloud Volumes ONTAP licenses without additional compute.

Multiple protocols

NetApp systems support most protocols for Hadoop deployments, including SAS, iSCSI, FCP, InfiniBand, and NFS.

Operational and supported solutions

The Hadoop solutions described in this document are supported by NetApp. These solutions are also certified with major Hadoop distributors. For information, see the MapR site, the Hortonworks site, and the Cloudera certification and partner sites.

Next: Target audience.

Target audience

Previous: Solution overview.

The world of analytics and data science touches multiple disciplines in IT and business:

- The data scientist needs the flexibility to use their tools and libraries of choice.
- The data engineer needs to know how the data flows and where it resides.
- A DevOps engineer needs the tools to integrate new AI and ML applications into their CI and CD pipelines.
- Cloud administrators and architects must be able to set up and manage hybrid cloud resources.
- Business users want to have access to analytics, AI, ML, and DL applications.

In this technical report, we describe how NetApp AFF, E-Series, StorageGRID, NFS direct access, Apache Spark, Horovod, and Keras help each of these roles bring value to business.

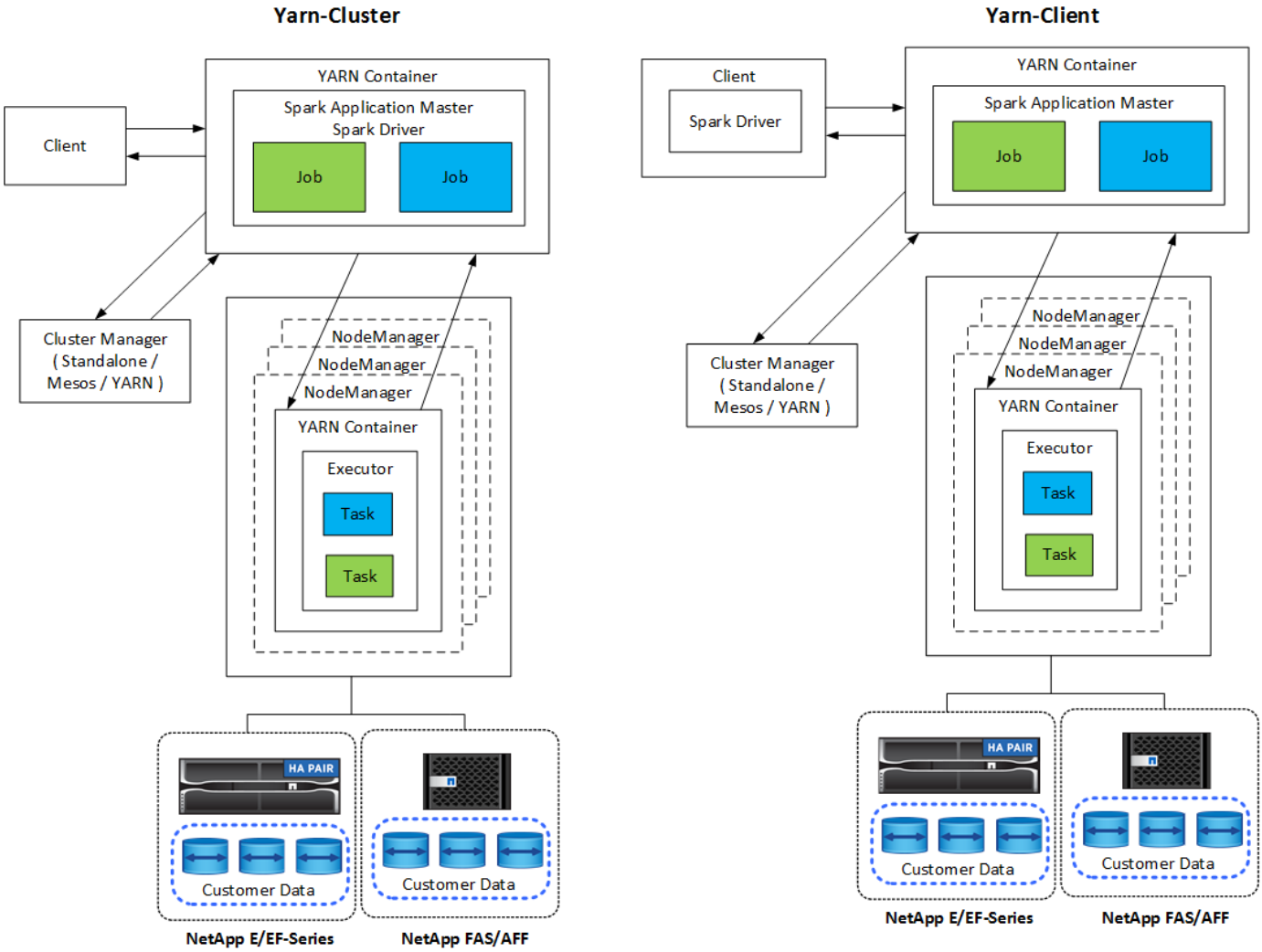
Next: Solution technology.

Solution technology

Previous: Target audience.

Apache Spark is a popular programming framework for writing Hadoop applications that works directly with the Hadoop Distributed File System (HDFS). Spark is production ready, supports processing of streaming data, and is faster than MapReduce. Spark has configurable in-memory data caching for efficient iteration, and the Spark shell is interactive for learning and exploring data. With Spark, you can create applications in Python, Scala, or Java. Spark applications consist of one or more jobs that have one or more tasks.

Every Spark application has a Spark driver. In YARN-Client mode, the driver runs on the client locally. In YARN-Cluster mode, the driver runs in the cluster on the application master. In the cluster mode, the application continues to run even if the client disconnects.



There are three cluster managers:

- **Standalone.** This manager is a part of Spark, which makes it easy to set up a cluster.
- **Apache Mesos.** This is a general cluster manager that also runs MapReduce and other applications.
- **Hadoop YARN.** This is a resource manager in Hadoop 3.

The resilient distributed dataset (RDD) is the primary component of Spark. RDD recreates the lost and missing data from data stored in memory in the cluster and stores the initial data that comes from a file or is created programmatically. RDDs are created from files, data in memory, or another RDD. Spark programming performs two operations: transformation and actions. Transformation creates a new RDD based on an existing one. Actions return a value from an RDD.

Transformations and actions also apply to Spark Datasets and DataFrames. A dataset is a distributed collection of data that provides the benefits of RDDs (strong typing, use of lambda functions) with the benefits of Spark SQL's optimized execution engine. A Dataset can be constructed from JVM objects and then manipulated using functional transformations (map, flatMap, filter, and so on.). A DataFrame is a dataset organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python. DataFrames can be constructed from a wide array of sources such as structured data files, tables in Hive/HBase, external databases on-premises or in the cloud, or existing RDDs.

Spark applications include one or more Spark jobs. Jobs run tasks in executors, and executors run in YARN containers. Each executor runs in a single container, and executors exist throughout the life of an application.

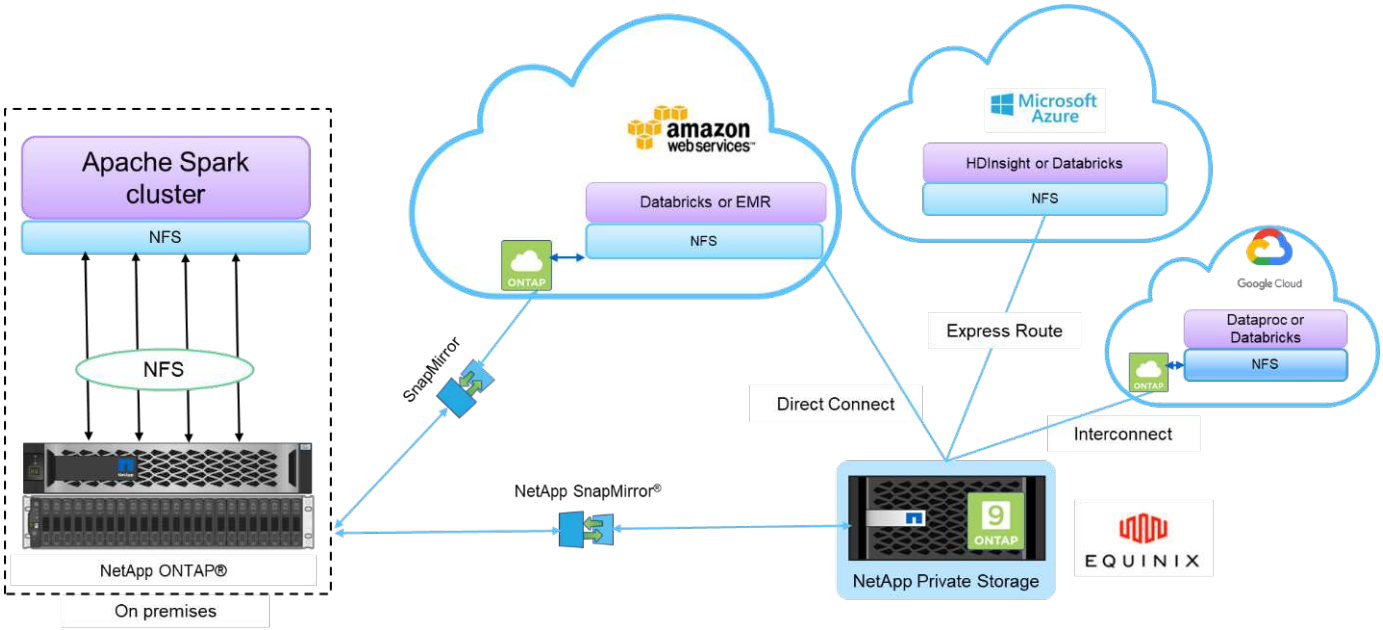
An executor is fixed after the application starts, and YARN does not resize the already allocated container. An executor can run tasks concurrently on in-memory data.

Next: NetApp Spark solutions overview.

NetApp Spark solutions overview

Previous: Solution technology.

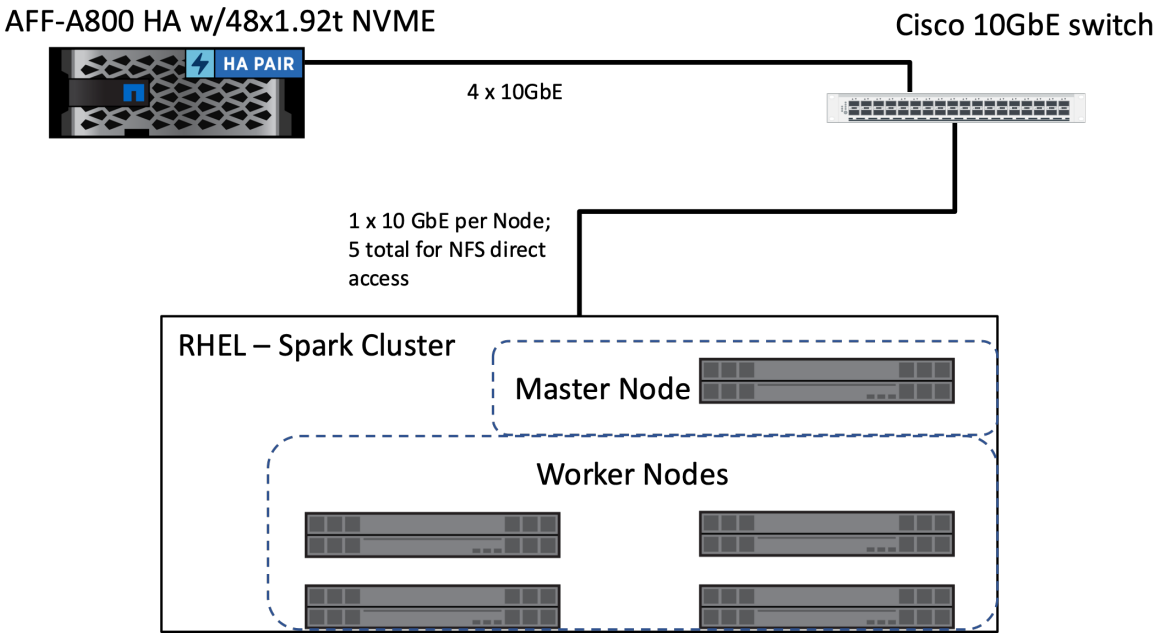
NetApp has three storage portfolios: FAS/AFF, E-Series, and Cloud Volumes ONTAP. We have validated AFF and the E-Series with ONTAP storage system for Hadoop solutions with Apache Spark. The data fabric powered by NetApp integrates data management services and applications (building blocks) for data access, control, protection, and security, as shown in the figure below.



The building blocks in the figure above include:

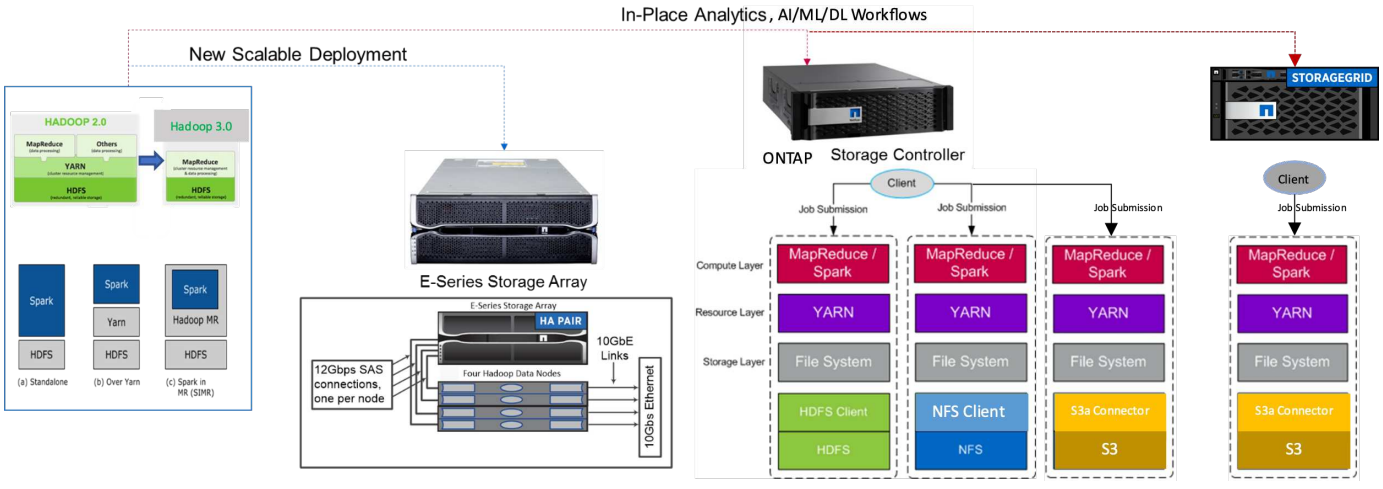
- **NetApp NFS direct access.** Provides the latest Hadoop and Spark clusters with direct access to NetApp NFS volumes without additional software or driver requirements.
- **NetApp Cloud Volumes ONTAP and Cloud Volume Services.** Software-defined connected storage based on ONTAP running in Amazon Web Services (AWS) or Azure NetApp Files (ANF) in Microsoft Azure cloud services.
- **NetApp SnapMirror technology.** Provides data protection capabilities between on-premises and ONTAP Cloud or NPS instances.
- **Cloud service providers.** These providers include AWS, Microsoft Azure, Google Cloud, and IBM Cloud.
- **PaaS.** Cloud-based analytics services such as Amazon Elastic MapReduce (EMR) and Databricks in AWS as well as Microsoft Azure HDInsight and Azure Databricks.

The following figure depicts the Spark solution with NetApp storage.

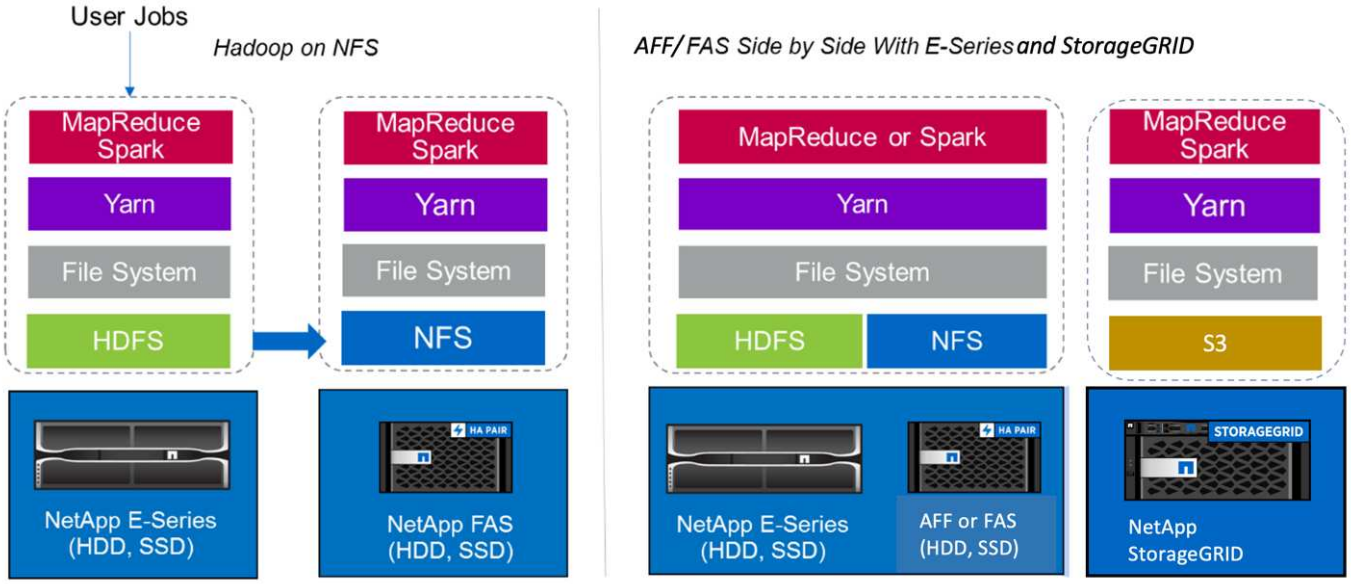


The ONTAP Spark solution uses the NetApp NFS direct access protocol for in-place analytics and AI, ML, and DL workflows using access to existing production data. Production data available to Hadoop nodes is exported to perform in-place analytical and AI, ML, and DL jobs. You can access data to process in Hadoop nodes either with NetApp NFS direct access or without it. In Spark with the standalone or `yarn` cluster manager, you can configure an NFS volume by using `file:///<target_volume`. We validated three use cases with different datasets. The details of these validations are presented in the section “Testing Results.” (xref)

The following figure depicts NetApp Apache Spark/Hadoop storage positioning.



We identified the unique features of the E-Series Spark solution, the AFF/FAS ONTAP Spark solution, and the StorageGRID Spark solution, and performed detailed validation and testing. Based upon our observations, NetApp recommends the E-Series solution for greenfield installations and new scalable deployments and the AFF/FAS solution for in-place analytics, AI, ML, and DL workloads using existing NFS data, and StorageGRID for AI, ML, and DL and modern data analytics when object storage is required.



A data lake is a storage repository for large datasets in native form that can be used for analytics, AI, ML, and DL jobs. We built a data lake repository for the E-Series, AFF/FAS, and StorageGRID SG6060 Spark solutions. The E-Series system provides HDFS access to the Hadoop Spark cluster, whereas existing production data is accessed through the NFS direct access protocol to the Hadoop cluster. For datasets that reside in object storage, NetApp StorageGRID provides S3 and S3a secure access.

Next: Use cases summary.

Use case summary

Previous: NetApp Spark solutions overview.

Streaming data

Apache Spark can process streaming data, which is used for streaming extract, transform, and load (ETL) processes; data enrichment; triggering event detection; and complex session analysis:

- **Streaming ETL.** Data is continually cleaned and aggregated before it is pushed into datastores. Netflix uses Kafka and Spark streaming to build a real-time online movie recommendation and data monitoring solution that can process billions of events per day from different data sources. Traditional ETL for batch processing is treated differently, however. This data is read first, and then it is converted into a database format before being written to the database.
- **Data enrichment.** Spark streaming enriches the live data with static data to enable more real-time data analysis. For example, online advertisers can deliver personalized, targeted ads directed by information about customer behavior.
- **Trigger event detection.** Spark streaming allows you to detect and respond quickly to unusual behavior that could indicate potentially serious problems. For example, financial institutions use triggers to detect and stop fraud transactions, and hospitals use triggers to detect dangerous health changes detected in a patient's vital signs.
- **Complex session analysis.** Spark streaming collects events such as user activity after logging in to a website or application, which are then grouped and analyzed. For example, Netflix uses this functionality to provide real-time movie recommendations.

For more streaming data configuration, Confluent Kafka verification, and performance tests, see TR-4912: Best

practice guidelines for Confluent Kafka tiered storage with NetApp.

Machine learning

The Spark integrated framework helps you run repeated queries on datasets using the machine learning library (MLlib). MLlib is used in areas such as clustering, classification, and dimensionality reduction for some common big data functions such as predictive intelligence, customer segmentation for marketing purposes, and sentiment analysis. MLlib is used in network security to conduct real-time inspections of data packets for indications of malicious activity. It helps security providers learn about new threats and stay ahead of hackers while protecting their clients in real time.

Deep learning

TensorFlow is a popular deep learning framework used across the industry. TensorFlow supports the distributed training on a CPU or GPU cluster. This distributed training allows users to run it on a large amount of data with lot of deep layers.

Until fair recently, if we wanted to use TensorFlow with Apache Spark, we needed to perform all necessary ETL for TensorFlow in PySpark and then write data to intermediate storage. That data would then be loaded onto the TensorFlow cluster for the actual training process. This workflow required the user to maintain two different clusters, one for ETL and one for distributed training of TensorFlow. Running and maintaining multiple clusters was typically tedious and time consuming.

DataFrames and RDD in earlier Spark versions were not well-suited for deep learning because random access was limited. In Spark 3.0 with project hydrogen, native support for the deep learning frameworks is added. This approach allows non-MapReduce-based scheduling on the Spark cluster.

Interactive analysis

Apache Spark is fast enough to perform exploratory queries without sampling with development languages other than Spark, including SQL, R, and Python. Spark uses visualization tools to process complex data and visualize it interactively. Spark with structured streaming performs interactive queries against live data in web analytics that enable you to run interactive queries against a web visitor’s current session.

Recommender system

Over the years, recommender systems have brought tremendous changes to our lives, as businesses and consumers have responded to dramatic changes in online shopping, online entertainment, and many other industries. Indeed, these systems are among the most evident success stories of AI in production. In many practical use cases, recommender systems are combined with conversational AI or chatbots interfaced with an NLP backend to obtain relevant information and produce useful inferences.

Today, many retailers are adopting newer business models like buying online and picking up in store, curbside pickup, self-checkout, scan-and-go, and more. These models have become prominent during the COVID-19 pandemic by making shopping safer and more convenient for consumers. AI is crucial for these growing digital trends, which are influenced by consumer behavior and vice versa. To meet the growing demands of consumers, to augment the customer experience, to improve operational efficiency, and to grow revenue, NetApp helps its enterprise customers and businesses use machine- learning and deep- learning algorithms to design faster and more accurate recommender systems.

There are several popular techniques used for providing recommendations, including collaborative filtering, content-based systems, the deep learning recommender model (DLRM), and hybrid techniques. Customers previously utilized PySpark to implement collaborative filtering for creating recommendation systems. Spark MLlib implements alternating least squares (ALS) for collaborative filtering, a very popular algorithm among enterprises before the rise of DLRM.

Natural language processing

Conversational AI, made possible by natural language processing (NLP), is the branch of AI helping computers communicate with humans. NLP is prevalent in every industry vertical and many use cases, from smart assistants and chatbots to Google search and predictive text. According to a Gartner prediction, by 2022, 70% of people will be interacting with conversational AI platforms on a daily basis. For a high-quality conversation between a human and a machine, responses must be rapid, intelligent, and natural sounding.

Customers need a large amount of data to process and train their NLP and automatic speech recognition (ASR) models. They also need to move data across the edge, core, and cloud, and they need the power to perform inference in milliseconds to establish natural communication with humans. NetApp AI and Apache Spark is an ideal combination for compute, storage, data processing, model training, fine-tuning, and deployment.

Sentiment analysis is a field of study within NLP in which positive, negative, or neutral sentiments are extracted from text. Sentiment analysis has a variety of use cases, from determining support center employee performance in conversations with callers to providing appropriate automated chatbot responses. It has also been used to predict a firm's stock price based on the interactions between firm representatives and the audience at quarterly earnings calls. Furthermore, sentiment analysis can be used to determine a customer's view on the products, services, or support provided by the brand.

We used the Spark NLP library from John Snow Labs to load pretrained pipelines and Bidirectional Encoder Representations from Transformers (BERT) models including financial news sentiment and FinBERT, performing tokenization, named entity recognition, model training, fitting and sentiment analysis at scale. Spark NLP is the only open-source NLP library in production that offers state-of-the-art transformers such as BERT, ALBERT, ELECTRA, XLNet, DistilBERT, RoBERTa, DeBERTa, XLM- RoBERTa, Longformer, ELMO, Universal Sentence Encoder, Google T5, MarianMT, and GPT2. The library works not only in Python and R, but also in the JVM ecosystem (Java, Scala, and Kotlin) at scale by extending Apache Spark natively.

Next: Major AI, ML, and DL use cases and architectures.

Major AI, ML, and DL use cases and architectures

Previous: Use cases summary.

Major AI, ML, and DL use cases and methodology can be divided into the following sections:

Spark NLP pipelines and TensorFlow distributed inferencing

The following list contains the most popular open-source NLP libraries that have been adopted by the data science community under different levels of development:

- Natural Language Toolkit (NLTK). The complete toolkit for all NLP techniques. It has been maintained since the early 2000s.
- TextBlob. An easy-to-use NLP tools Python API built on top of NLTK and Pattern.
- Stanford Core NLP. NLP services and packages in Java developed by the Stanford NLP Group.
- Gensim. Topic Modelling for Humans started off as a collection of Python scripts for the Czech Digital Mathematics Library project.
- SpaCy. End-to-end industrial NLP workflows with Python and Cython with GPU acceleration for transformers.
- Fasttext. A free, lightweight, open-source NLP library for the learning-of-word embeddings and sentence classification created by Facebook's AI Research (FAIR) lab.

Spark NLP is a single, unified solution for all NLP tasks and requirements that enables scalable, high-performance, and high-accuracy NLP-powered software for real production use cases. It leverages transfer learning and implements the latest state-of-the-art algorithms and models in research and across industries. Due to the lack of full support by Spark for the above libraries, Spark NLP was built on top of Spark ML to take advantage of Spark's general-purpose in-memory distributed data processing engine as an enterprise-grade NLP library for mission-critical production workflows. Its annotators utilize rule-based algorithms, machine learning, and TensorFlow to power deep learning implementations. This covers common NLP tasks including but not limited to tokenization, lemmatization, stemming, part-of-speech tagging, named-entity recognition, spell checking, and sentiment analysis.

Bidirectional Encoder Representations from Transformers (BERT) is a transformer-based machine learning technique for NLP. It popularized the concept of pretraining and fine tuning. The transformer architecture in BERT originated from machine translation, which models long-term dependencies better than Recurrent Neural Network (RNN)-based language models. It also introduced the Masked Language Modelling (MLM) task, where a random 15% of all tokens are masked and the model predicts them, enabling true bidirectionality.

Financial sentiment analysis is challenging due to the specialized language and lack of labeled data in that domain. FinBERT, a language model based on pretrained BERT, was domain adapted on Reuters TRC2, a financial corpus, and fine-tuned with labeled data (Financial PhraseBank) for financial sentiment classification. Researchers extracted 4, 500 sentences from news articles with financial terms. Then 16 experts and masters students with finance backgrounds labeled the sentences as positive, neutral, and negative. We built an end-to-end Spark workflow to analyze sentiment for Top-10 NASDAQ company earnings call transcripts from 2016 to 2020 using FinBERT and two other pre-trained pipelines (Sentiment Analysis for Financial News, Explain Document DL) from Spark NLP.

The underlying deep learning engine for Spark NLP is TensorFlow, an end-to-end, open-source platform for machine learning that enables easy model building, robust ML production anywhere, and powerful experimentation for research. Therefore, when executing our pipelines in Spark `yarn cluster` mode, we were essentially running distributed TensorFlow with data and model parallelization across one master and multiple worker nodes, as well as network- attached storage mounted on the cluster.

Horovod distributed training

The core Hadoop validation for MapReduce-related performance is performed with TeraGen, TeraSort, TeraValidate, and DFSIO (read and write). The TeraGen and TeraSort validation results are presented in TR-3969: NetApp Solutions for Hadoop for E-Series and in the section "Storage Tiering" (xref) for AFF.

Based upon customer requests, we consider distributed training with Spark to be one of the most important of the various use cases. In this document, we used the Hovorod on Spark to validate Spark performance with NetApp on-premises, cloud-native, and hybrid cloud solutions using NetApp All Flash FAS (AFF) storage controllers, Azure NetApp Files, and StorageGRID.

The Horovod on Spark package provides a convenient wrapper around Horovod that makes running distributed training workloads in Spark clusters simple, enabling a tight model design loop in which data processing, model training, and model evaluation are all done in Spark where training and inferencing data resides.

There are two APIs for running Horovod on Spark: a high-level Estimator API and a lower-level Run API. Although both use the same underlying mechanism to launch Horovod on Spark executors, the Estimator API abstracts the data processing, model training loop, model checkpointing, metrics collection, and distributed training. We used Horovod Spark Estimators, TensorFlow, and Keras for an end-to-end data preparation and distributed training workflow based on the Kaggle Rossmann Store Sales competition.

The script `keras_spark_horovod_rossmann_estimator.py` can be found in the section "Python scripts for each major use case." It contains three parts:

- The first part performs various data preprocessing steps over an initial set of CSV files provided by Kaggle and gathered by the community. The input data is separated into a training set with a `Validation` subset, and a testing dataset.
- The second part defines a Keras Deep Neural Network (DNN) model with logarithmic sigmoid activation function and an Adam optimizer, and it performs distributed training of the model using Horovod on Spark.
- The third part performs prediction on the testing dataset using the best model that minimizes the validation set overall mean absolute error. It then creates an output CSV file.

See the section "Machine Learning" for various runtime comparison results.

Multi-worker deep learning using Keras for CTR prediction

With the recent advances in ML platforms and applications, a lot of attention is now on learning at scale. The click-through rate (CTR) is defined as the average number of click-throughs per hundred online ad impressions (expressed as a percentage). It is widely adopted as a key metric in various industry verticals and use cases, including digital marketing, retail, e-commerce, and service providers. See our TR-4904: Distributed training in Azure - Click-Through Rate Prediction for more detail on the applications of CTR and an end-to-end Cloud AI workflow implementation with Kubernetes, distributed data ETL, and model training using Dask and CUDA ML.

In this technical report we used a variation of the Criteo Terabyte Click Logs dataset (see TR-4904) for multi-worker distributed deep learning using Keras to build a Spark workflow with Deep and Cross Network (DCN) models, comparing its performance in terms of log loss error function with a baseline Spark ML Logistic Regression model. DCN efficiently captures effective feature interactions of bounded degrees, learns highly nonlinear interactions, requires no manual feature engineering or exhaustive searching, and has low computational cost.

Data for web-scale recommender systems is mostly discrete and categorical, leading to a large and sparse feature space that is challenging for feature exploration. This has limited most large-scale systems to linear models such as logistic regression. However, identifying frequently predictive features and at the same time exploring unseen or rare cross features is the key to making good predictions. Linear models are simple, interpretable, and easy to scale, but they are limited in their expressive power.

Cross features, on the other hand, have been shown to be significant in improving the models' expressiveness. Unfortunately, it often requires manual feature engineering or exhaustive search to identify such features. Generalizing to unseen feature interactions is often difficult. Using a cross neural network like DCN avoids task-specific feature engineering by explicitly applying feature crossing in an automatic fashion. The cross network consists of multiple layers, where the highest degree of interactions is provably determined by layer depth. Each layer produces higher-order interactions based on existing ones and keeps the interactions from previous layers.

A deep neural network (DNN) has the promise to capture very complex interactions across features. However, compared to DCN, it requires nearly an order of magnitude more parameters, is unable to form cross features explicitly, and may fail to efficiently learn some types of feature interactions. The cross network is memory efficient and easy to implement. Jointly training the cross and DNN components together efficiently captures predictive feature interactions and delivers state-of-the-art performance on the Criteo CTR dataset.

A DCN model starts with an embedding and stacking layer, followed by a cross network and a deep network in parallel. These in turn are followed by a final combination layer which combines the outputs from the two networks. Your input data can be a vector with sparse and dense features. In Spark, both ml and mllib libraries contain the type `SparseVector`. It is therefore important for users to distinguish between the two and be mindful when calling their respective functions and methods. In web-scale recommender systems such as CTR prediction, the inputs are mostly categorical features, for example `'country=usa'`. Such features are often encoded as one-hot vectors, for example, `'[0,1,0, …]'`. One-hot-encoding (OHE) with `SparseVector` is useful when dealing with real-world datasets with ever-changing and growing vocabularies. We modified

examples in DeepCTR to process large vocabularies, creating embedding vectors in the embedding and stacking layer of our DCN.

The Criteo Display Ads dataset predicts the ads click-through rate. It has 13 integer features and 26 categorical features in which each category has a high cardinality. For this dataset, an improvement of 0.001 in logloss is practically significant due to the large input size. A small improvement in prediction accuracy for a large user base can potentially lead to a large increase in a company's revenue. The dataset contains 11GB of user logs from a period of 7 days, which equates to around 41 million records. We used Spark `dataFrame.randomSplit()function` to randomly split the data for training (80%), cross-validation (10%), and the remaining 10% for testing.

DCN was implemented on TensorFlow with Keras. There are four main components in implementing the model training process with DCN:

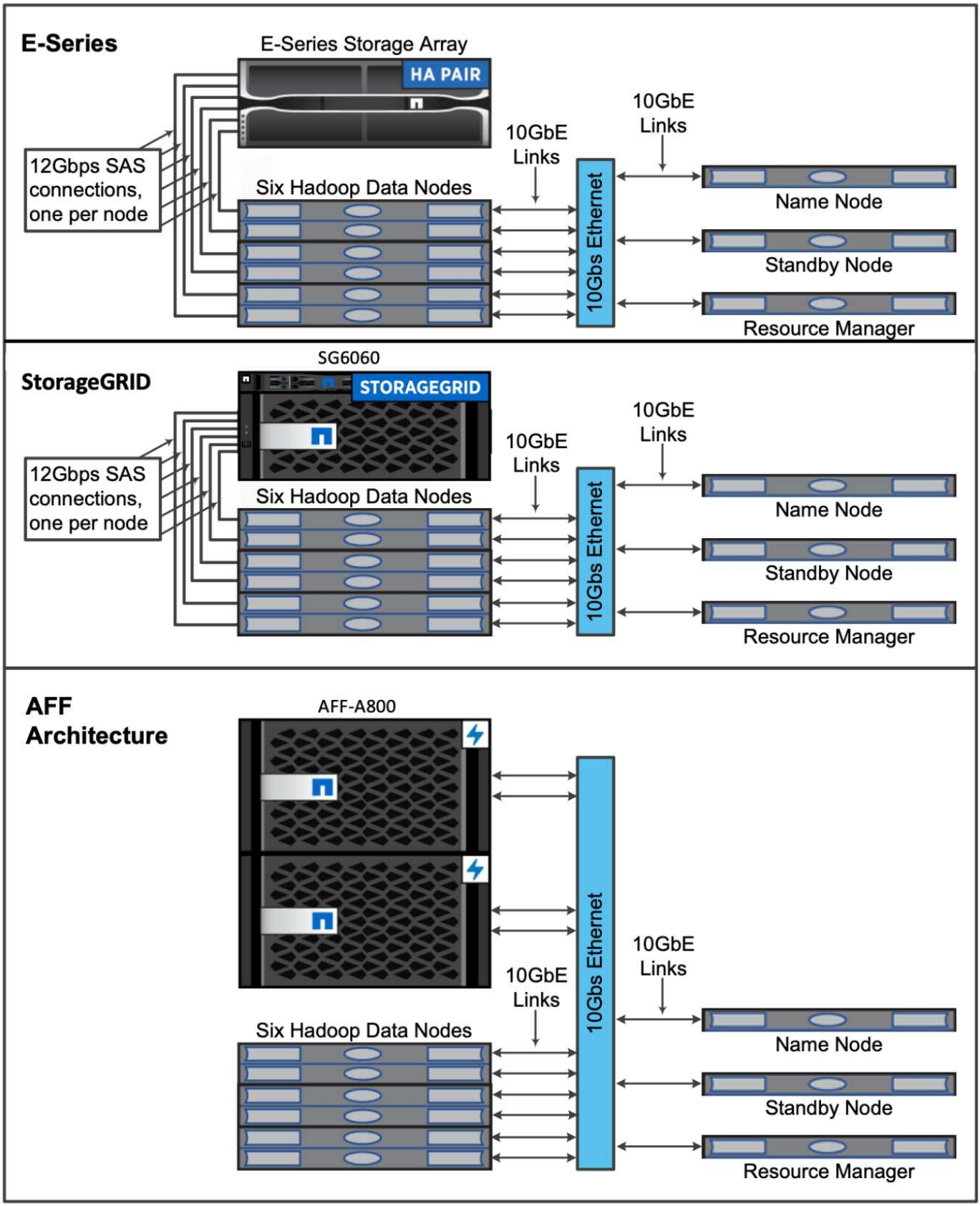
- **Data processing and embedding.** Real-valued features are normalized by applying a log transform. For categorical features, we embed the features in dense vectors of dimension 6×(category cardinality)1/4. Concatenating all embeddings results in a vector of dimension 1026.
- **Optimization.** We applied mini-batch stochastic optimization with the Adam optimizer. The batch size was set to 512. Batch normalization was applied to the deep network and the gradient clip norm was set at 100.
- **Regularization.** We used early stopping, as L2 regularization or dropout was not found to be effective.
- **Hyperparameters.** We report results based on a grid search over the number of hidden layers, the hidden layer size, the initial learning rate, and the number of cross layers. The number of hidden layers ranged from 2 to 5, with hidden layer sizes ranging from 32 to 1024. For DCN, the number of cross layers was from 1 to 6. The initial learning rate was tuned from 0.0001 to 0.001 with increments of 0.0001. All experiments applied early stopping at training step 150,000, beyond which overfitting started to occur.

In addition to DCN, we also tested other popular deep-learning models for CTR prediction, including DeepFM, xDeepFM, AutoInt, and DCN v2.

Architectures used for validation

For this validation, we used four worker nodes and one master nodes with an AFF-A800 HA pair. All cluster members were connected through 10GbE network switches.

For this NetApp Spark solution validation, we used three different storage controllers: the E5760, the E5724, and the AFF-A800. The E-Series storage controllers were connected to five data nodes with 12Gbps SAS connections. The AFF HA-pair storage controller provides exported NFS volumes through 10GbE connections to Hadoop worker nodes. The Hadoop cluster members were connected through 10GbE connections in the E-Series, AFF, and StorageGRID Hadoop solutions.



Next: Testing results.

Testing results

Previous: Major AI, ML, and DL use cases and architectures.

We used the TeraSort and TeraValidate scripts in the TeraGen benchmarking tool to measure the Spark performance validation with E5760, E5724, and AFF-A800 configurations. In addition, three major use cases were tested: Spark NLP pipelines and TensorFlow distributed training, Horovod distributed training, and multi-worker deep learning using Keras for CTR Prediction with DeepFM.

For both E-Series and StorageGRID validation, we used Hadoop replication factor 2. For AFF validation, we only used one source of data.

The following table lists the hardware configuration for the Spark performance validation.

Type	Hadoop worker nodes	Drive type	Drives per node	Storage controller
SG6060	4	SAS	12	Single high-availability (HA) pair
E5760	4	SAS	60	Single HA pair
E5724	4	SAS	24	Single HA pair
AFF800	4	SSD	6	Single HA pair

The following table lists software requirements.

Software	Version
RHEL	7.9
OpenJDK Runtime Environment	1.8.0
OpenJDK 64-Bit Server VM	25.302
Git	2.24.1
GCC/G++	11.2.1
Spark	3.2.1
PySpark	3.1.2
SparkNLP	3.4.2
TensorFlow	2.9.0
Keras	2.9.0
Horovod	0.24.3

Financial sentiment analysis

We published TR-4910: Sentiment Analysis from Customer Communications with NetApp AI, in which an end-to-end conversational AI pipeline was built using the NetApp DataOps Toolkit, AFF storage, and NVIDIA DGX System. The pipeline performs batch audio signal processing, automatic speech recognition (ASR), transfer learning, and sentiment analysis leveraging the DataOps Toolkit, NVIDIA Riva SDK, and the Tao framework. Expanding the sentiment analysis use case to the financial services industry, we built a SparkNLP workflow, loaded three BERT models for various NLP tasks, such as named entity recognition, and obtained sentence-

level sentiment for NASDAQ Top 10 companies' quarterly earnings calls.

The following script `sentiment_analysis_spark. py` uses the FinBERT model to process transcripts in HDFS and produce positive, neutral, and negative sentiment counts, as shown in the following table:

```
-bash-4.2$ time ~/anaconda3/bin/spark-submit
--packages com.johnsnowlabs.nlp:spark-nlp_2.12:3.4.3
--master yarn
--executor-memory 5g
--executor-cores 1
--num-executors 160
--conf spark.driver.extraJavaOptions="-Xss10m -XX:MaxPermSize=1024M"
--conf spark.executor.extraJavaOptions="-Xss10m -XX:MaxPermSize=512M"
/sparkusecase/tr-4570-nlp/sentiment_analysis_spark.py
hdfs:///data1/Transcripts/
> ./sentiment_analysis_hdfs.log 2>&1
real13m14.300s
user557m11.319s
sys4m47.676s
```

The following table lists the earnings-call, sentence-level sentiment analysis for NASDAQ Top 10 companies from 2016 to 2020.

Sentiment counts and percentage	All 10 Companies	AAPL	AMD	AMZN	CSCO	GOOGL	INTC	MSFT	NVDA
Positive counts	7447	1567	743	290	682	826	824	904	417
Neutral counts	64067	6856	7596	5086	6650	5914	6099	5715	6189
Negative counts	1787	253	213	84	189	97	282	202	89
Uncategorized counts	196	0	0	76	0	0	0	1	0
(total counts)	73497	8676	8552	5536	7521	6837	7205	6822	6695

In terms of percentages, most sentences spoken by the CEOs and CFOs are factual and therefore carry neutral sentiment. During an earnings call, analysts ask questions which might convey positive or negative sentiment. It is worth further investigating quantitatively how negative or positive sentiment affect stock prices on the same or next day of trading.

The following table lists the sentence-level sentiment analysis for NASDAQ Top 10 companies, expressed in

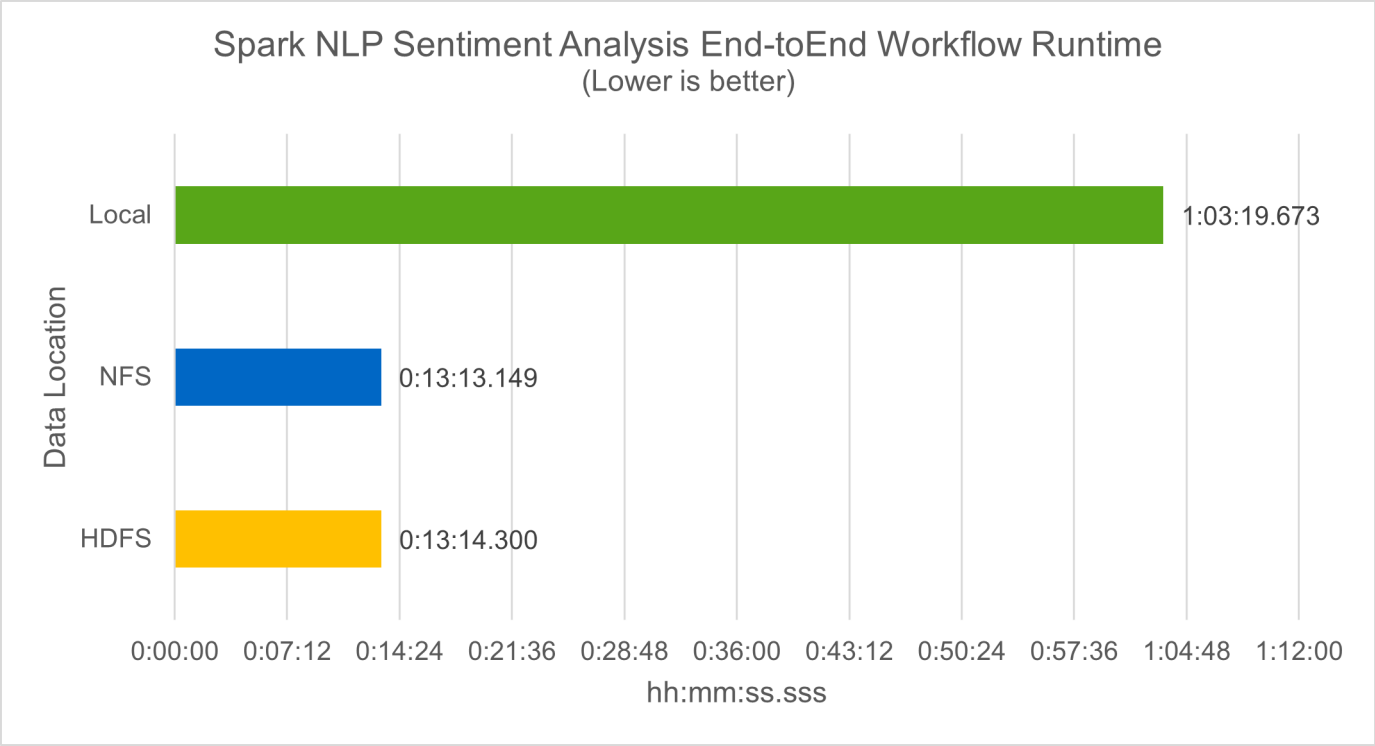
percentage.

Sentiment percentage	All 10 Companies	AAPL	AMD	AMZN	CSCO	GOOGL	INTC	MSFT	NVDA
Positive	10.13%	18.06%	8.69%	5.24%	9.07%	12.08%	11.44%	13.25%	6.23%
Neutral	87.17%	79.02%	88.82%	91.87%	88.42%	86.50%	84.65%	83.77%	92.44%
Negative	2.43%	2.92%	2.49%	1.52%	2.51%	1.42%	3.91%	2.96%	1.33%
Uncategorized	0.27%	0%	0%	1.37%	0%	0%	0%	0.01%	0%

In terms of the workflow runtime, we saw a significant 4.78x improvement from `local` mode to a distributed environment in HDFS, and a further 0.14% improvement by leveraging NFS.

```
-bash-4.2$ time ~/anaconda3/bin/spark-submit
--packages com.johnsnowlabs.nlp:spark-nlp_2.12:3.4.3
--master yarn
--executor-memory 5g
--executor-cores 1
--num-executors 160
--conf spark.driver.extraJavaOptions="-Xss10m -XX:MaxPermSize=1024M"
--conf spark.executor.extraJavaOptions="-Xss10m -XX:MaxPermSize=512M"
/sparkusecase/tr-4570-nlp/sentiment_analysis_spark.py
file:///sparkdemo/sparknlp/Transcripts/
> ./sentiment_analysis_nfs.log 2>&1
real13m13.149s
user537m50.148s
sys4m46.173s
```

As the following figure shows, data and model parallelism improved the data processing and distributed TensorFlow model inferencing speed. Data location in NFS yielded a slightly better runtime because the workflow bottleneck is the downloading of pretrained models. If we increase the transcripts dataset size, the advantage of NFS is more obvious.



Distributed training with Horovod performance

The following command produced runtime information and a log file in our Spark cluster using a single `master` node with 160 executors each with one core. The executor memory was limited to 5GB to avoid out-of-memory error. See the section "Python scripts for each major use case" for more detail regarding the data processing, model training, and model accuracy calculation in `keras_spark_horovod_rossmann_estimator.py`.

```
(base) [root@n138 horovod]# time spark-submit
--master local
--executor-memory 5g
--executor-cores 1
--num-executors 160
/sparkusecase/horovod/keras_spark_horovod_rossmann_estimator.py
--epochs 10
--data-dir file:///sparkusecase/horovod
--local-submission-csv /tmp/submission_0.csv
--local-checkpoint-file /tmp/checkpoint/
> /tmp/keras_spark_horovod_rossmann_estimator_local. log 2>&1
```

The resulting runtime with ten training epochs was as follows:

```
real43m34.608s
user12m22.057s
sys2m30.127s
```

It took more than 43 minutes to process input data, train a DNN model, calculate accuracy, and produce

TensorFlow checkpoints and a CSV file for prediction results. We limited the number of training epochs to 10, which in practice is often set to 100 to ensure satisfactory model accuracy. The training time typically scales linearly with the number of epochs.

We next used the four worker nodes available in the cluster and executed the same script in `yarn` mode with data in HDFS:

```
(base) [root@n138 horovod]# time spark-submit
--master yarn
--executor-memory 5g
--executor-cores 1 --num-executors 160
/sparkusecase/horovod/keras_spark_horovod_rossmann_estimator.py
--epochs 10
--data-dir hdfs:///user/hdfs/tr-4570/experiments/horovod
--local-submission-csv /tmp/submission_1.csv
--local-checkpoint-file /tmp/checkpoint/
> /tmp/keras_spark_horovod_rossmann_estimator_yarn.log 2>&1
```

The resulting runtime was improved as follows:

```
real8m13.728s
user7m48.421s
sys1m26.063s
```

With Horovod's model and data parallelism in Spark, we saw a 5.29x runtime speedup of `yarn` versus `local` mode with ten training epochs. This is shown in the following figure with the legends `HDFS` and `Local`. The underlying TensorFlow DNN model training can be further accelerated with GPUs if available. We plan to conduct this testing and publish results in a future technical report.

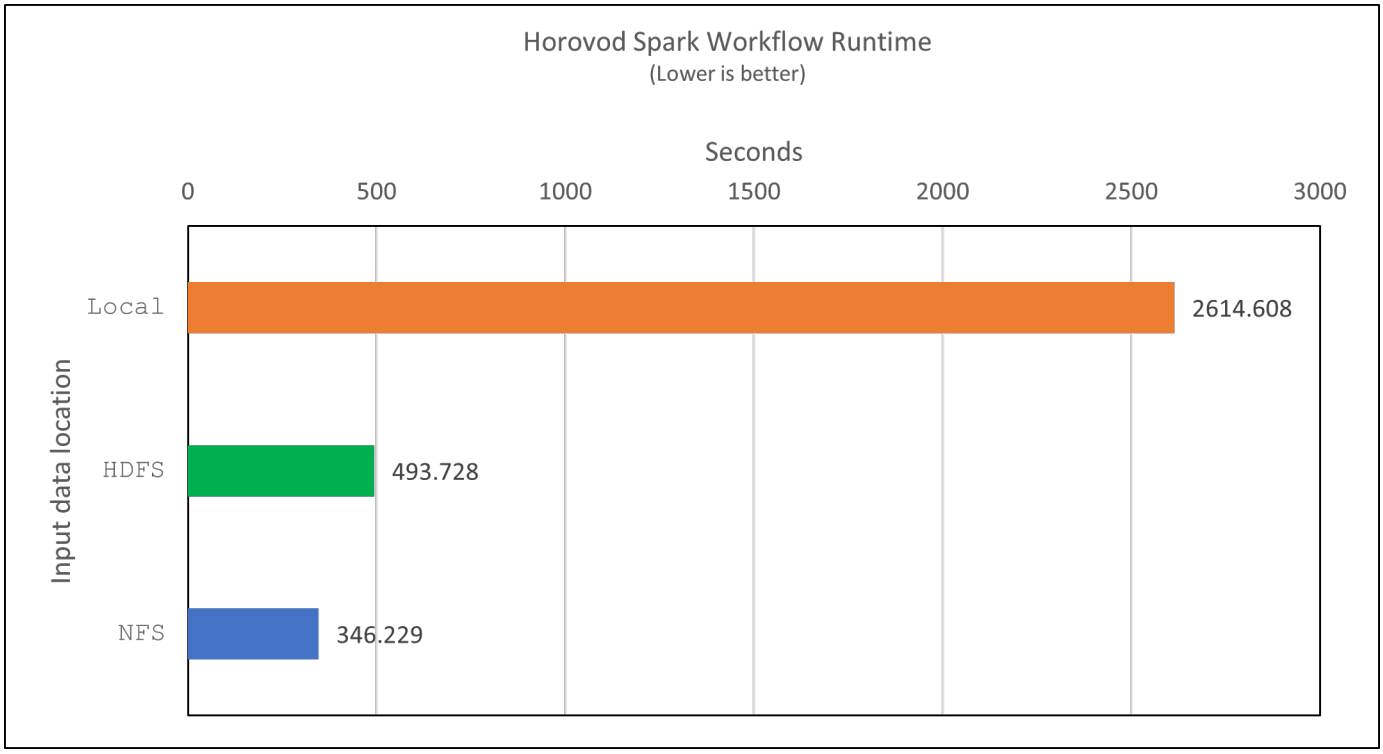
Our next test compared the runtimes with input data residing in NFS versus HDFS. The NFS volume on the AFF A800 was mounted on `/sparkdemo/horovod` across the five nodes (one master, four workers) in our Spark cluster. We ran a similar command as for previous tests, with the `--data- dir` parameter now pointing to the NFS mount:

```
(base) [root@n138 horovod]# time spark-submit
--master yarn
--executor-memory 5g
--executor-cores 1
--num-executors 160
/sparkusecase/horovod/keras_spark_horovod_rossmann_estimator.py
--epochs 10
--data-dir file:///sparkdemo/horovod
--local-submission-csv /tmp/submission_2.csv
--local-checkpoint-file /tmp/checkpoint/
> /tmp/keras_spark_horovod_rossmann_estimator_nfs.log 2>&1
```

The resulting runtime with NFS was as follows:

```
real 5m46.229s
user 5m35.693s
sys  1m5.615s
```

There was a further 1.43x speedup, as shown in the following figure. Therefore, with a NetApp all-flash storage connected to their cluster, customers enjoy the benefits of fast data transfer and distribution for Horovod Spark workflows, achieving 7.55x speedup versus running on a single node.



Deep learning models for CTR prediction performance

For recommender systems designed to maximize CTR, you must learn sophisticated feature interactions behind user behaviors that can be mathematically calculated from low order to high order. Both low-order and high-order feature interactions should be equally important for a good deep learning model without biasing towards one or the other. Deep Factorization Machine (DeepFM), a factorization machine-based neural network, combines factorization machines for recommendation and deep learning for feature learning in a new neural network architecture.

Although conventional factorization machines model pairwise feature interactions as an inner product of latent vectors between features and can theoretically capture high-order information, in practice, machine learning practitioners usually only use second- order feature interactions due to the high computation and storage complexity. Deep neural network variants like Google's Wide & Deep Models on the other hand learns sophisticated feature interactions in a hybrid network structure by combining a linear wide model and a deep model.

There are two inputs to this Wide & Deep Model, one for the underlying wide model and the other for the deep, the latter part of which still requires expert feature engineering and thus renders the technique less generalizable to other domains. Unlike the Wide & Deep Model, DeepFM can be efficiently trained with raw features without any feature engineering because its wide part and deep part share the same input and the

embedding vector.

We first processed the Criteo `train.txt` (11GB) file into a CSV file named `ctr_train.csv` stored in an NFS mount `/sparkdemo/tr-4570-data` using `run_classification_criteo_spark.py` from the section "Python scripts for each major use case." Within this script, the function `process_input_file` performs several string methods to remove tabs and insert `','` as the delimiter and `'\n'` as newline. Note that you only need to process the original `train.txt` once, so that the code block is shown as comments.

For the following testing of different DL models, we used `ctr_train.csv` as the input file. In subsequent testing runs, the input CSV file was read into a Spark DataFrame with schema containing a field of `'label'`, integer dense features `['I1', 'I2', 'I3', …, 'I13']`, and sparse features `['C1', 'C2', 'C3', …, 'C26']`. The following `spark-submit` command takes in an input CSV, trains DeepFM models with 20% split for cross validation, and picks the best model after ten training epochs to calculate prediction accuracy on the testing set:

```
(base) [root@n138 ~]# time spark-submit --master yarn --executor-memory 5g
--executor-cores 1 --num-executors 160
/sparkusecase/DeepCTR/examples/run_classification_criteo_spark.py --data
-dir file:///sparkdemo/tr-4570-data >
/tmp/run_classification_criteo_spark_local.log 2>&1
```

Note that since the data file `ctr_train.csv` is over 11GB, you must set a sufficient `spark.driver.maxResultSize` greater than the dataset size to avoid error.

```
 spark = SparkSession.builder \
    .master("yarn") \
    .appName("deep_ctr_classification") \
    .config("spark.jars.packages", "io.github.ravwojdyla:spark-schema-
utils_2.12:0.1.0") \
    .config("spark.executor.cores", "1") \
    .config('spark.executor.memory', '5gb') \
    .config('spark.executor.memoryOverhead', '1500') \
    .config('spark.driver.memoryOverhead', '1500') \
    .config("spark.sql.shuffle.partitions", "480") \
    .config("spark.sql.execution.arrow.enabled", "true") \
    .config("spark.driver.maxResultSize", "50gb") \
    .getOrCreate()
```

In the above `SparkSession.builder` configuration we also enabled Apache Arrow, which converts a Spark DataFrame into a Pandas DataFrame with the `df.toPandas()` method.

```
22/06/17 15:56:21 INFO scheduler.DAGScheduler: Job 2 finished: toPandas at
/sparkusecase/DeepCTR/examples/run_classification_criteo_spark.py:96, took
627.126487 s
Obtained Spark DF and transformed to Pandas DF using Arrow.
```

After random splitting, there are over 36M rows in the training dataset and 9M samples in the testing set:

```
Training dataset size =  36672493
Testing dataset size =  9168124
```

Because this technical report is focused on CPU testing without using any GPUs, it is imperative that you build TensorFlow with appropriate compiler flags. This step avoids invoking any GPU-accelerated libraries and takes full advantage of TensorFlow's Advanced Vector Extensions (AVX) and AVX2 instructions. These features are designed for linear algebraic computations like vectorized addition, matrix multiplications inside a feed-forward, or back-propagation DNN training. Fused Multiply Add (FMA) instruction available with AVX2 using 256-bit floating point (FP) registers is ideal for integer code and data types, resulting in up to a 2x speedup. For FP code and data types, AVX2 achieves 8% speedup over AVX.

```
2022-06-18 07:19:20.101478: I
tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary
is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the
following CPU instructions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the
appropriate compiler flags.
```

To build TensorFlow from source, NetApp recommends using Bazel. For our environment, we executed the following commands in the shell prompt to install `dnf`, `dnf-plugins`, and Bazel.

```
yum install dnf
dnf install 'dnf-command(copr)'
dnf copr enable vbatts/bazel
dnf install bazel5
```

You must enable GCC 5 or newer to use C++17 features during the build process, which is provided by RHEL with Software Collections Library (SCL). The following commands install `devtoolset` and GCC 11.2.1 on our RHEL 7.9 cluster:

```
subscription-manager repos --enable rhel-server-rhscl-7-rpms
yum install devtoolset-11-toolchain
yum install devtoolset-11-gcc-c++
yum update
scl enable devtoolset-11 bash
. /opt/rh/devtoolset-11/enable
```

Note that the last two commands enable `devtoolset-11`, which uses `/opt/rh/devtoolset-11/root/usr/bin/gcc` (GCC 11.2.1). Also, make sure your `git` version is greater than 1.8.3 (this comes with RHEL 7.9). Refer to this article for updating `git` to 2.24.1.

We assume that you have already cloned the latest TensorFlow master repo. Then create a `workspace`

directory with a `WORKSPACE` file to build TensorFlow from source with AVX, AVX2, and FMA. Run the `configure` file and specify the correct Python binary location. CUDA is disabled for our testing because we did not use a GPU. A `.bazelrc` file is generated according to your settings. Further, we edited the file and set `build --define=no_hdfs_support=false` to enable HDFS support. Refer to `.bazelrc` in the section "Python scripts for each major use case," for a complete list of settings and flags.

```
./configure
bazel build -c opt --copt=-mavx --copt=-mavx2 --copt=-mfma --copt=
-mfpmath=both -k //tensorflow/tools/pip_package:build_pip_package
```

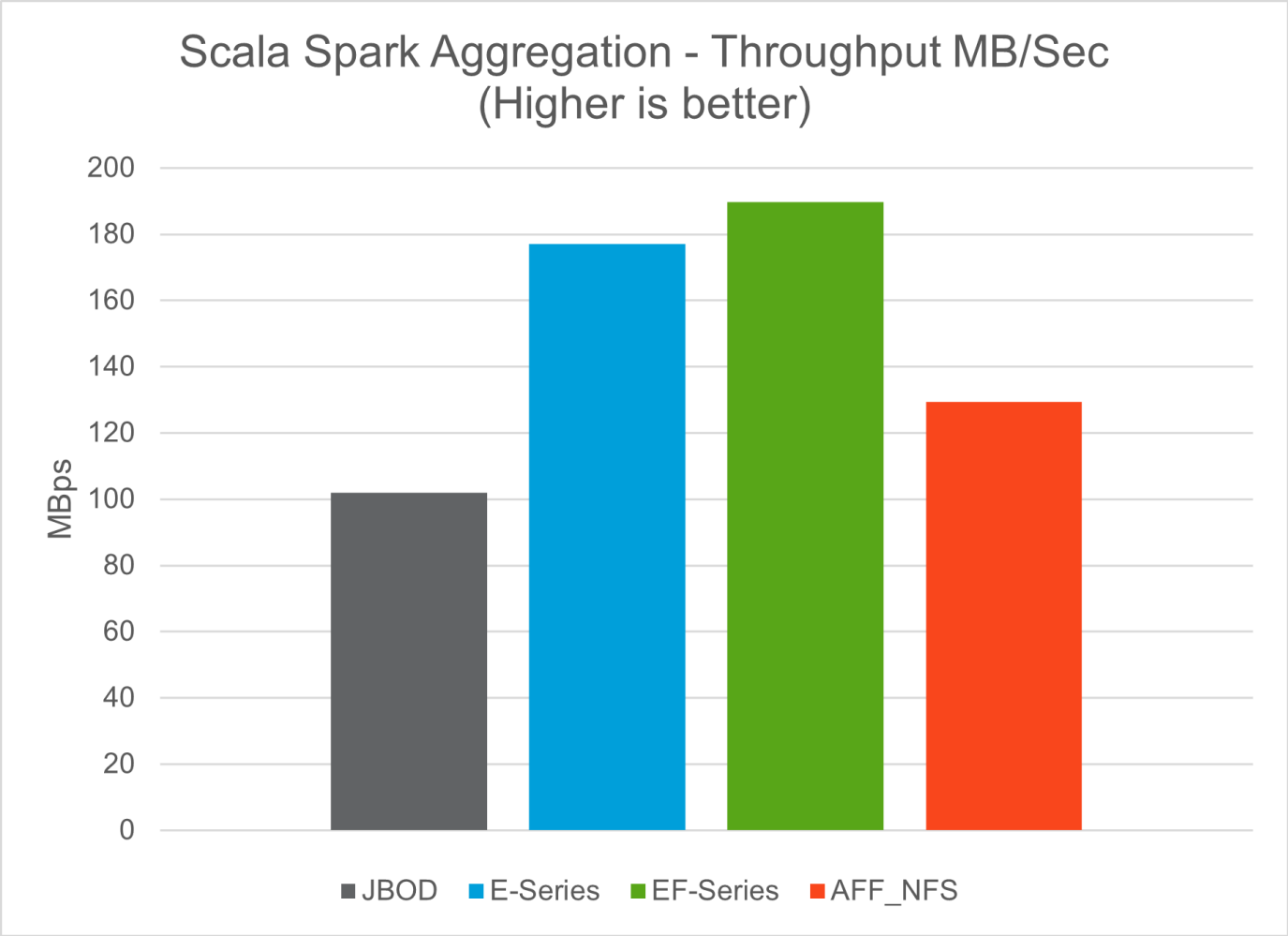
After you build TensorFlow with the correct flags, run the following script to process the Criteo Display Ads dataset, train a DeepFM model, and calculate the Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.

```
(base) [root@n138 examples]# ~/anaconda3/bin/spark-submit
--master yarn
--executor-memory 15g
--executor-cores 1
--num-executors 160
/sparkusecase/DeepCTR/examples/run_classification_criteo_spark.py
--data-dir file:///sparkdemo/tr-4570-data
> . /run_classification_criteo_spark_nfs.log 2>&1
```

After ten training epochs, we obtained the AUC score on the testing dataset:

```
Epoch 1/10
125/125 - 7s - loss: 0.4976 - binary_crossentropy: 0.4974 - val_loss:
0.4629 - val_binary_crossentropy: 0.4624
Epoch 2/10
125/125 - 1s - loss: 0.3281 - binary_crossentropy: 0.3271 - val_loss:
0.5146 - val_binary_crossentropy: 0.5130
Epoch 3/10
125/125 - 1s - loss: 0.1948 - binary_crossentropy: 0.1928 - val_loss:
0.6166 - val_binary_crossentropy: 0.6144
Epoch 4/10
125/125 - 1s - loss: 0.1408 - binary_crossentropy: 0.1383 - val_loss:
0.7261 - val_binary_crossentropy: 0.7235
Epoch 5/10
125/125 - 1s - loss: 0.1129 - binary_crossentropy: 0.1102 - val_loss:
0.7961 - val_binary_crossentropy: 0.7934
Epoch 6/10
125/125 - 1s - loss: 0.0949 - binary_crossentropy: 0.0921 - val_loss:
0.9502 - val_binary_crossentropy: 0.9474
Epoch 7/10
125/125 - 1s - loss: 0.0778 - binary_crossentropy: 0.0750 - val_loss:
1.1329 - val_binary_crossentropy: 1.1301
Epoch 8/10
125/125 - 1s - loss: 0.0651 - binary_crossentropy: 0.0622 - val_loss:
1.3794 - val_binary_crossentropy: 1.3766
Epoch 9/10
125/125 - 1s - loss: 0.0555 - binary_crossentropy: 0.0527 - val_loss:
1.6115 - val_binary_crossentropy: 1.6087
Epoch 10/10
125/125 - 1s - loss: 0.0470 - binary_crossentropy: 0.0442 - val_loss:
1.6768 - val_binary_crossentropy: 1.6740
test AUC 0.6337
```

In a manner similar to previous use cases, we compared the Spark workflow runtime with data residing in different locations. The following figure shows a comparison of the deep learning CTR prediction for a Spark workflows runtime.



Next: Hybrid cloud solution.

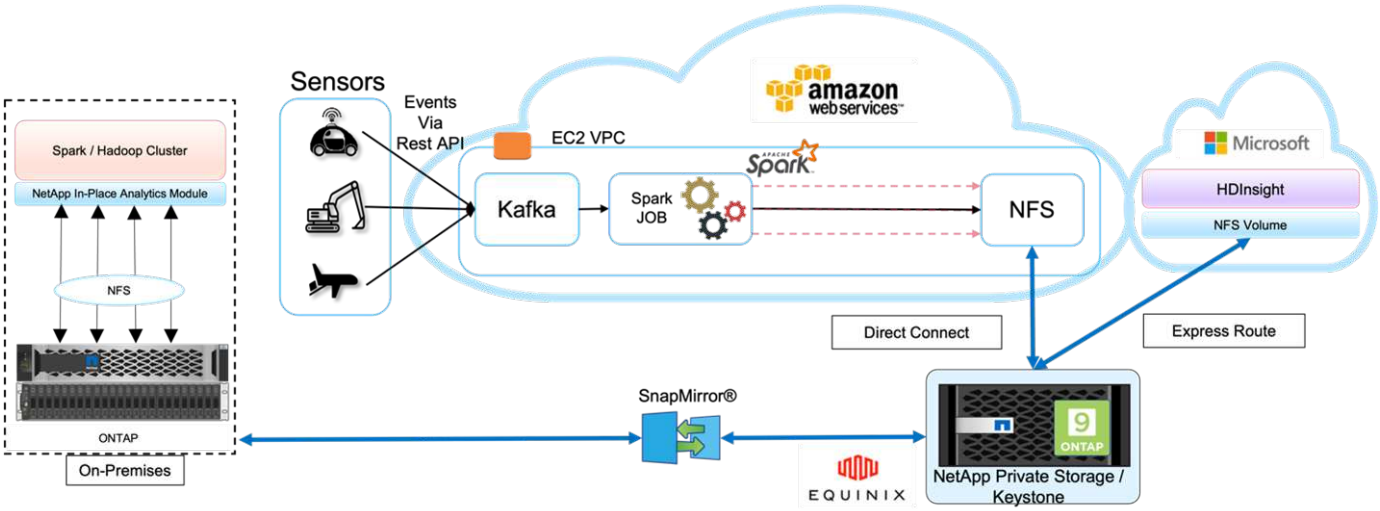
Hybrid cloud solution

Previous: Testing results.

A modern enterprise data center is a hybrid cloud that connects multiple distributed infrastructure environments through a continuous data management plane with a consistent operating model, on premises and/or in multiple public clouds. To get the most out of a hybrid cloud, you must be able to seamlessly move data between your on-premises and multi-cloud environments without the need for any data conversions or application refactoring.

Customers have indicated that they start their hybrid cloud journey either by moving secondary storage to the cloud for use cases such as data protection or by moving less business-critical workloads such as application development and DevOps to the cloud. They then move on to more critical workloads. Web and content hosting, DevOps and application development, databases, analytics, and containerized apps are among the most popular hybrid-cloud workloads. The complexity, cost, and risks of enterprise AI projects have historically hindered AI adoption from experimental stage to production.

With a NetApp hybrid-cloud solution, customers benefit from integrated security, data governance, and compliance tools with a single control panel for data and workflow management across distributed environments, while optimizing the total cost of ownership based on their consumption. The following figure is an example solution of a cloud service partner tasked with providing multi-cloud connectivity for customers' big-data-analytics data.



In this scenario, IoT data received in AWS from different sources is stored in a central location in NetApp Private Storage (NPS). The NPS storage is connected to Spark or Hadoop clusters located in AWS and Azure enabling big-data-analytics applications running in multiple clouds accessing the same data. The main requirements and challenges for this use case include the following:

- Customers want to run analytics jobs on the same data using multiple clouds.
- Data must be received from different sources such as on-premises and cloud environments through different sensors and hubs.
- The solution must be efficient and cost effective.
- The main challenge is to build a cost-effective and efficient solution that delivers hybrid analytics services between different on-premises and cloud environments.

Our data protection and multicloud connectivity solution resolves the pain point of having cloud analytics applications across multiple hyperscalers. As shown in the figure above, data from sensors is streamed and ingested into the AWS Spark cluster through Kafka. The data is stored in an NFS share residing in NPS, which is located outside of the cloud provider within an Equinix data center.

Because NetApp NPS is connected to Amazon AWS and Microsoft Azure through Direct Connect and Express Route connections respectively, customers can leverage the In-Place Analytics Module to access the data from both Amazon and AWS analytics clusters. Consequently, because both on-premises and NPS storage runs ONTAP software, SnapMirror can mirror the NPS data into the on-premises cluster, providing hybrid cloud analytics across on-premises and multiple clouds.

For the best performance, NetApp typically recommends using multiple network interfaces and direct connection or express routes to access the data from cloud instances. We have other data mover solutions including XCP and Cloud Sync to help customers build application-aware, secure, and cost-effective hybrid-cloud Spark clusters.

Next: Python scripts for each major use case.

Python scripts for each major use case

Previous: Hybrid cloud solution.

The following three Python scripts correspond to the three major use cases tested. First is `sentiment_analysis_sparknlp.py`.

```
# TR-4570 Refresh NLP testing by Rick Huang
from sys import argv
import os
import sparknlp
import pyspark.sql.functions as F
from sparknlp import Finisher
from pyspark.ml import Pipeline
from sparknlp.base import *
from sparknlp.annotator import *
from sparknlp.pretrained import PretrainedPipeline
from sparknlp import Finisher
# Start Spark Session with Spark NLP
spark = sparknlp.start()
print("Spark NLP version:")
print(sparknlp.version())
print("Apache Spark version:")
print(spark.version)
spark = sparknlp.SparkSession.builder \
     .master("yarn") \
     .appName("test_hdfs_read_write") \
     .config("spark.executor.cores", "1") \
     .config("spark.jars.packages", "com.johnsnowlabs.nlp:spark-
nlp_2.12:3.4.3")\
     .config('spark.executor.memory', '5gb') \
     .config('spark.executor.memoryOverhead','1000')\
     .config('spark.driver.memoryOverhead','1000')\
     .config("spark.sql.shuffle.partitions", "480")\
     .getOrCreate()
sc = spark.sparkContext
from pyspark.sql import SQLContext
sql = SQLContext(sc)
sqlContext = SQLContext(sc)
# Download pre-trained pipelines & sequence classifier
explain_pipeline_model = PretrainedPipeline('explain_document_dl',
lang='en').model#pipeline_sa =
PretrainedPipeline("classifierdl_bertwiki_finance_sentiment_pipeline",
lang="en")
# pipeline_finbert =
BertForSequenceClassification.loadSavedModel('/sparkusecase/bert_sequence_
classifier_finbert_en_3', spark)
sequenceClassifier = BertForSequenceClassification \
        .pretrained('bert_sequence_classifier_finbert', 'en') \
        .setInputCols(['token', 'document']) \
        .setOutputCol('class') \
        .setCaseSensitive(True) \
        .setMaxSentenceLength(512)
```

```
def process_sentence_df(data):
    # Pre-process: begin
    print("1. Begin DataFrame pre-processing...\n")
    print(f"\n\t2. Attaching DocumentAssembler Transformer to the
pipeline")
    documentAssembler = DocumentAssembler() \
        .setInputCol("text") \
        .setOutputCol("document") \
        .setCleanupMode("inplace_full")
        #.setCleanupMode("shrink", "inplace_full")
    doc_df = documentAssembler.transform(data)
    doc_df.printSchema()
    doc_df.show(truncate=50)
    # Pre-process: get rid of  blank lines
    clean_df = doc_df.withColumn("tmp", F.explode("document")) \
        .select("tmp.result").where("tmp.end !=
-1").withColumnRenamed("result", "text").dropna()
    print("[OK!] DataFrame after initial cleanup:\n")
    clean_df.printSchema()
    clean_df.show(truncate=80)
    # for FinBERT
    tokenizer = Tokenizer() \
        .setInputCols(['document']) \
        .setOutputCol('token')
    print(f"\n\t3. Attaching Tokenizer Annotator to the pipeline")
    pipeline_finbert = Pipeline(stages=[
        documentAssembler,
        tokenizer,
        sequenceClassifier
        ])
    # Use Finisher() & construct PySpark ML pipeline
    finisher = Finisher().setInputCols(["token", "lemma", "pos",
"entities"])
    print(f"\n\t4. Attaching Finisher Transformer to the pipeline")
    pipeline_ex = Pipeline() \
        .setStages([
           explain_pipeline_model,
           finisher
           ])
    print("\n\t\t\t ---- Pipeline Built Successfully ----")
    # Loading pipelines to annotate
    #result_ex_df = pipeline_ex.transform(clean_df)
    ex_model = pipeline_ex.fit(clean_df)
    annotations_finished_ex_df = ex_model.transform(clean_df)
    # result_sa_df = pipeline_sa.transform(clean_df)
    result_finbert_df = pipeline_finbert.fit(clean_df).transform(clean_df)
```

```
    print("\n\t\t\t ----Document Explain, Sentiment Analysis & FinBERT
Pipeline Fitted Successfully ----")
    # Check the result entities
    print("[OK!] Simple explain ML pipeline result:\n")
    annotations_finished_ex_df.printSchema()
    annotations_finished_ex_df.select('text',
'finished_entities').show(truncate=False)
    # Check the result sentiment from FinBERT
    print("[OK!] Sentiment Analysis FinBERT pipeline result:\n")
    result_finbert_df.printSchema()
    result_finbert_df.select('text', 'class.result').show(80, False)
    sentiment_stats(result_finbert_df)
    return
def sentiment_stats(finbert_df):
    result_df = finbert_df.select('text', 'class.result')
    sa_df = result_df.select('result')
    sa_df.groupBy('result').count().show()
    # total_lines = result_clean_df.count()
    # num_neutral = result_clean_df.where(result_clean_df.result ==
['neutral']).count()
    # num_positive = result_clean_df.where(result_clean_df.result ==
['positive']).count()
    # num_negative = result_clean_df.where(result_clean_df.result ==
['negative']).count()
    # print(f"\nRatio of neutral sentiment = {num_neutral/total_lines}")
    # print(f"Ratio of positive sentiment = {num_positive / total_lines}")
    # print(f"Ratio of negative sentiment = {num_negative /
total_lines}\n")
    return
def process_input_file(file_name):
    # Turn input file to Spark DataFrame
    print("START processing input file...")
    data_df = spark.read.text(file_name)
    data_df.show()
    # rename first column 'text' for sparknlp
    output_df = data_df.withColumnRenamed("value", "text").dropna()
    output_df.printSchema()
    return output_dfdef process_local_dir(directory):
    filelist = []
    for subdir, dirs, files in os.walk(directory):
        for filename in files:
            filepath = subdir + os.sep + filename
            print("[OK!] Will process the following files:")
            if filepath.endswith(".txt"):
                print(filepath)
                filelist.append(filepath)
```

```
    return filelist
def process_local_dir_or_file(dir_or_file):
    numfiles = 0
    if os.path.isfile(dir_or_file):
        input_df = process_input_file(dir_or_file)
        print("Obtained input_df.")
        process_sentence_df(input_df)
        print("Processed input_df")
        numfiles += 1
    else:
        filelist = process_local_dir(dir_or_file)
        for file in filelist:
            input_df = process_input_file(file)
            process_sentence_df(input_df)
            numfiles += 1
    return numfiles
def process_hdfs_dir(dir_name):
    # Turn input files to Spark DataFrame
    print("START processing input HDFS directory...")
    data_df = spark.read.option("recursiveFileLookup",
"true").text(dir_name)
    data_df.show()
    print("[DEBUG] total lines in data_df = ", data_df.count())
    # rename first column 'text' for sparknlp
    output_df = data_df.withColumnRenamed("value", "text").dropna()
    print("[DEBUG] output_df looks like: \n")
    output_df.show(40, False)
    print("[DEBUG] HDFS dir resulting data_df schema: \n")
    output_df.printSchema()
    process_sentence_df(output_df)
    print("Processed HDFS directory: ", dir_name)
    returnif __name__ == '__main__':
    try:
        if len(argv) == 2:
            print("Start processing input...\n")
    except:
        print("[ERROR] Please enter input text file or path to
process!\n")
        exit(1)
    # This is for local file, not hdfs:
    numfiles = process_local_dir_or_file(str(argv[1]))
    # For HDFS single file & directory:
    input_df = process_input_file(str(argv[1]))
    print("Obtained input_df.")
    process_sentence_df(input_df)
    print("Processed input_df")
```

```
    numfiles += 1
    # For HDFS directory of subdirectories of files:
    input_parse_list = str(argv[1]).split('/')
    print(input_parse_list)
    if input_parse_list[-2:-1] == ['Transcripts']:
        print("Start processing HDFS directory: ", str(argv[1]))
        process_hdfs_dir(str(argv[1]))
    print(f"[OK!] All done. Number of files processed = {numfiles}")
```

The second script is `keras_spark_horovod_rossmann_estimator.py`.

```
# Copyright 2022 NetApp, Inc.
# Authored by Rick Huang
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
================================================================================
====
# The below code was modified from: https://www.kaggle.com/c/rossmann-
store-sales
import argparse
import datetime
import os
import sys
from distutils.version import LooseVersion
import pyspark.sql.types as T
import pyspark.sql.functions as F
from pyspark import SparkConf, Row
from pyspark.sql import SparkSession
import tensorflow as tf
import tensorflow.keras.backend as K
from tensorflow.keras.layers import Input, Embedding, Concatenate, Dense,
Flatten, Reshape, BatchNormalization, Dropout
import horovod.spark.keras as hvd
from horovod.spark.common.backend import SparkBackend
```

```
from horovod.spark.common.store import Store
from horovod.tensorflow.keras.callbacks import BestModelCheckpoint
parser = argparse.ArgumentParser(description='Horovod Keras Spark Rossmann
Estimator Example',

formatter_class=argparse.ArgumentDefaultsHelpFormatter)
parser.add_argument('--master',
                    help='spark cluster to use for training. If set to
None, uses current default cluster. Cluster'
                         'should be set up to provide a Spark task per
multiple CPU cores, or per GPU, e.g. by'
                         'supplying `-c <NUM_GPUS>` in Spark Standalone
mode')
parser.add_argument('--num-proc', type=int,
                    help='number of worker processes for training,
default: `spark.default.parallelism`')
parser.add_argument('--learning_rate', type=float, default=0.0001,
                    help='initial learning rate')
parser.add_argument('--batch-size', type=int, default=100,
                    help='batch size')
parser.add_argument('--epochs', type=int, default=100,
                    help='number of epochs to train')
parser.add_argument('--sample-rate', type=float,
                    help='desired sampling rate. Useful to set to low
number (e.g. 0.01) to make sure that '
                         'end-to-end process works')
parser.add_argument('--data-dir', default='file://' + os.getcwd(),
                    help='location of data on local filesystem (prefixed
with file://) or on HDFS')
parser.add_argument('--local-submission-csv', default='submission.csv',
                    help='output submission predictions CSV')
parser.add_argument('--local-checkpoint-file', default='checkpoint',
                    help='model checkpoint')
parser.add_argument('--work-dir', default='/tmp',
                    help='temporary working directory to write
intermediate files (prefix with hdfs:// to use HDFS)')
if __name__ == '__main__':
    args = parser.parse_args()
    # ================ #
    # DATA PREPARATION #
    # ================ #
    print('================')
    print('Data preparation')
    print('================')
    # Create Spark session for data preparation.
    conf = SparkConf() \
```

```
        .setAppName('Keras Spark Rossmann Estimator Example') \
        .set('spark.sql.shuffle.partitions', '480') \
        .set("spark.executor.cores", "1") \
        .set('spark.executor.memory', '5gb') \
        .set('spark.executor.memoryOverhead','1000')\
        .set('spark.driver.memoryOverhead','1000')
    if args.master:
        conf.setMaster(args.master)
    elif args.num_proc:
        conf.setMaster('local[{}]'.format(args.num_proc))
    spark = SparkSession.builder.config(conf=conf).getOrCreate()
    train_csv = spark.read.csv('%s/train.csv' % args.data_dir,
header=True)
    test_csv = spark.read.csv('%s/test.csv' % args.data_dir, header=True)
    store_csv = spark.read.csv('%s/store.csv' % args.data_dir,
header=True)
    store_states_csv = spark.read.csv('%s/store_states.csv' %
args.data_dir, header=True)
    state_names_csv = spark.read.csv('%s/state_names.csv' % args.data_dir,
header=True)
    google_trend_csv = spark.read.csv('%s/googletrend.csv' %
args.data_dir, header=True)
    weather_csv = spark.read.csv('%s/weather.csv' % args.data_dir,
header=True)
    def expand_date(df):
        df = df.withColumn('Date', df.Date.cast(T.DateType()))
        return df \
            .withColumn('Year', F.year(df.Date)) \
            .withColumn('Month', F.month(df.Date)) \
            .withColumn('Week', F.weekofyear(df.Date)) \
            .withColumn('Day', F.dayofmonth(df.Date))
    def prepare_google_trend():
        # Extract week start date and state.
        google_trend_all = google_trend_csv \
            .withColumn('Date', F.regexp_extract(google_trend_csv.week,
'(.*?) -', 1)) \
            .withColumn('State', F.regexp_extract(google_trend_csv.file,
'Rossmann_DE_(.*)', 1))
        # Map state NI -> HB,NI to align with other data sources.
        google_trend_all = google_trend_all \
            .withColumn('State', F.when(google_trend_all.State == 'NI',
'HB,NI').otherwise(google_trend_all.State))
        # Expand dates.
        return expand_date(google_trend_all)
    def add_elapsed(df, cols):
        def add_elapsed_column(col, asc):
```

```
            def fn(rows):
                last_store, last_date = None, None
                for r in rows:
                    if last_store != r.Store:
                        last_store = r.Store
                        last_date = r.Date
                    if r[col]:
                        last_date = r.Date
                    fields = r.asDict().copy()
                    fields[('After' if asc else 'Before') + col] = (r.Date
- last_date).days
                    yield Row(**fields)
            return fn
        df = df.repartition(df.Store)
        for asc in [False, True]:
            sort_col = df.Date.asc() if asc else df.Date.desc()
            rdd = df.sortWithinPartitions(df.Store.asc(), sort_col).rdd
            for col in cols:
                rdd = rdd.mapPartitions(add_elapsed_column(col, asc))
            df = rdd.toDF()
        return df
    def prepare_df(df):
        num_rows = df.count()
        # Expand dates.
        df = expand_date(df)
        df = df \
            .withColumn('Open', df.Open != '0') \
            .withColumn('Promo', df.Promo != '0') \
            .withColumn('StateHoliday', df.StateHoliday != '0') \
            .withColumn('SchoolHoliday', df.SchoolHoliday != '0')
        # Merge in store information.
        store = store_csv.join(store_states_csv, 'Store')
        df = df.join(store, 'Store')
        # Merge in Google Trend information.
        google_trend_all = prepare_google_trend()
        df = df.join(google_trend_all, ['State', 'Year',
'Week']).select(df['*'], google_trend_all.trend)
        # Merge in Google Trend for whole Germany.
        google_trend_de = google_trend_all[google_trend_all.file ==
'Rossmann_DE'].withColumnRenamed('trend', 'trend_de')
        df = df.join(google_trend_de, ['Year', 'Week']).select(df['*'],
google_trend_de.trend_de)
        # Merge in weather.
        weather = weather_csv.join(state_names_csv, weather_csv.file ==
state_names_csv.StateName)
        df = df.join(weather, ['State', 'Date'])
```

```
        # Fix null values.
        df = df \
            .withColumn('CompetitionOpenSinceYear',
F.coalesce(df.CompetitionOpenSinceYear, F.lit(1900))) \
            .withColumn('CompetitionOpenSinceMonth',
F.coalesce(df.CompetitionOpenSinceMonth, F.lit(1))) \
            .withColumn('Promo2SinceYear', F.coalesce(df.Promo2SinceYear,
F.lit(1900))) \
            .withColumn('Promo2SinceWeek', F.coalesce(df.Promo2SinceWeek,
F.lit(1)))
        # Days & months competition was open, cap to 2 years.
        df = df.withColumn('CompetitionOpenSince',
                           F.to_date(F.format_string('%s-%s-15',
df.CompetitionOpenSinceYear,

df.CompetitionOpenSinceMonth)))
        df = df.withColumn('CompetitionDaysOpen',
                           F.when(df.CompetitionOpenSinceYear > 1900,
                                  F.greatest(F.lit(0), F.least(F.lit(360 *
2), F.datediff(df.Date, df.CompetitionOpenSince))))
                           .otherwise(0))
        df = df.withColumn('CompetitionMonthsOpen',
(df.CompetitionDaysOpen / 30).cast(T.IntegerType()))
        # Days & weeks of promotion, cap to 25 weeks.
        df = df.withColumn('Promo2Since',
                           F.expr('date_add(format_string("%s-01-01",
Promo2SinceYear), (cast(Promo2SinceWeek as int) - 1) * 7)'))
        df = df.withColumn('Promo2Days',
                           F.when(df.Promo2SinceYear > 1900,
                                  F.greatest(F.lit(0), F.least(F.lit(25 *
7), F.datediff(df.Date, df.Promo2Since))))
                           .otherwise(0))
        df = df.withColumn('Promo2Weeks', (df.Promo2Days /
7).cast(T.IntegerType()))
        # Check that we did not lose any rows through inner joins.
        assert num_rows == df.count(), 'lost rows in joins'
        return df
    def build_vocabulary(df, cols):
        vocab = {}
        for col in cols:
            values = [r[0] for r in df.select(col).distinct().collect()]
            col_type = type([x for x in values if x is not None][0])
            default_value = col_type()
            vocab[col] = sorted(values, key=lambda x: x or default_value)
        return vocab
    def cast_columns(df, cols):
```

```
        for col in cols:
            df = df.withColumn(col,
F.coalesce(df[col].cast(T.FloatType()), F.lit(0.0)))
        return df
    def lookup_columns(df, vocab):
        def lookup(mapping):
            def fn(v):
                return mapping.index(v)
            return F.udf(fn, returnType=T.IntegerType())
        for col, mapping in vocab.items():
            df = df.withColumn(col, lookup(mapping)(df[col]))
        return df
    if args.sample_rate:
        train_csv = train_csv.sample(withReplacement=False,
fraction=args.sample_rate)
        test_csv = test_csv.sample(withReplacement=False,
fraction=args.sample_rate)
    # Prepare data frames from CSV files.
    train_df = prepare_df(train_csv).cache()
    test_df = prepare_df(test_csv).cache()
    # Add elapsed times from holidays & promos, the data spanning training
& test datasets.
    elapsed_cols = ['Promo', 'StateHoliday', 'SchoolHoliday']
    elapsed = add_elapsed(train_df.select('Date', 'Store', *elapsed_cols)
                          .unionAll(test_df.select('Date', 'Store',
*elapsed_cols)),
                          elapsed_cols)
    # Join with elapsed times.
    train_df = train_df \
        .join(elapsed, ['Date', 'Store']) \
        .select(train_df['*'], *[prefix + col for prefix in ['Before',
'After'] for col in elapsed_cols])
    test_df = test_df \
        .join(elapsed, ['Date', 'Store']) \
        .select(test_df['*'], *[prefix + col for prefix in ['Before',
'After'] for col in elapsed_cols])
    # Filter out zero sales.
    train_df = train_df.filter(train_df.Sales > 0)
    print('====================')
    print('Prepared data frame')
    print('====================')
    train_df.show()
    categorical_cols = [
        'Store', 'State', 'DayOfWeek', 'Year', 'Month', 'Day', 'Week',
'CompetitionMonthsOpen', 'Promo2Weeks', 'StoreType',
        'Assortment', 'PromoInterval', 'CompetitionOpenSinceYear',
```

```
'Promo2SinceYear', 'Events', 'Promo',
        'StateHoliday', 'SchoolHoliday'
    ]
    continuous_cols = [
        'CompetitionDistance', 'Max_TemperatureC', 'Mean_TemperatureC',
'Min_TemperatureC', 'Max_Humidity',
        'Mean_Humidity', 'Min_Humidity', 'Max_Wind_SpeedKm_h',
'Mean_Wind_SpeedKm_h', 'CloudCover', 'trend', 'trend_de',
        'BeforePromo', 'AfterPromo', 'AfterStateHoliday',
'BeforeStateHoliday', 'BeforeSchoolHoliday', 'AfterSchoolHoliday'
    ]
    all_cols = categorical_cols + continuous_cols
    # Select features.
    train_df = train_df.select(*(all_cols + ['Sales', 'Date'])).cache()
    test_df = test_df.select(*(all_cols + ['Id', 'Date'])).cache()
    # Build vocabulary of categorical columns.
    vocab = build_vocabulary(train_df.select(*categorical_cols)

.unionAll(test_df.select(*categorical_cols)).cache(),
                              categorical_cols)
    # Cast continuous columns to float & lookup categorical columns.
    train_df = cast_columns(train_df, continuous_cols + ['Sales'])
    train_df = lookup_columns(train_df, vocab)
    test_df = cast_columns(test_df, continuous_cols)
    test_df = lookup_columns(test_df, vocab)
    # Split into training & validation.
    # Test set is in 2015, use the same period in 2014 from the training
set as a validation set.
    test_min_date = test_df.agg(F.min(test_df.Date)).collect()[0][0]
    test_max_date = test_df.agg(F.max(test_df.Date)).collect()[0][0]
    one_year = datetime.timedelta(365)
    train_df = train_df.withColumn('Validation',
                                   (train_df.Date > test_min_date -
one_year) & (train_df.Date <= test_max_date - one_year))
    # Determine max Sales number.
    max_sales = train_df.agg(F.max(train_df.Sales)).collect()[0][0]
    # Convert Sales to log domain
    train_df = train_df.withColumn('Sales', F.log(train_df.Sales))
    print('====================================')
    print('Data frame with transformed columns')
    print('====================================')
    train_df.show()
    print('===============')
    print('Data frame sizes')
    print('===============')
    train_rows = train_df.filter(~train_df.Validation).count()
```

```
    val_rows = train_df.filter(train_df.Validation).count()
    test_rows = test_df.count()
    print('Training: %d' % train_rows)
    print('Validation: %d' % val_rows)
    print('Test: %d' % test_rows)
    # ============== #
    # MODEL TRAINING #
    # ============== #
    print('==============')
    print('Model training')
    print('==============')
    def exp_rmspe(y_true, y_pred):
        """Competition evaluation metric, expects logarithic inputs."""
        pct = tf.square((tf.exp(y_true) - tf.exp(y_pred)) /
tf.exp(y_true))
        # Compute mean excluding stores with zero denominator.
        x = tf.reduce_sum(tf.where(y_true > 0.001, pct,
tf.zeros_like(pct)))
        y = tf.reduce_sum(tf.where(y_true > 0.001, tf.ones_like(pct),
tf.zeros_like(pct)))
        return tf.sqrt(x / y)
    def act_sigmoid_scaled(x):
        """Sigmoid scaled to logarithm of maximum sales scaled by 20%."""
        return tf.nn.sigmoid(x) * tf.math.log(max_sales) * 1.2
    CUSTOM_OBJECTS = {'exp_rmspe': exp_rmspe,
                      'act_sigmoid_scaled': act_sigmoid_scaled}
    # Disable GPUs when building the model to prevent memory leaks
    if LooseVersion(tf.__version__) >= LooseVersion('2.0.0'):
        # See https://github.com/tensorflow/tensorflow/issues/33168
        os.environ['CUDA_VISIBLE_DEVICES'] = '-1'
    else:

K.set_session(tf.Session(config=tf.ConfigProto(device_count={'GPU': 0})))
    # Build the model.
    inputs = {col: Input(shape=(1,), name=col) for col in all_cols}
    embeddings = [Embedding(len(vocab[col]), 10, input_length=1,
name='emb_' + col)(inputs[col])
                  for col in categorical_cols]
    continuous_bn = Concatenate()([Reshape((1, 1), name='reshape_' +
col)(inputs[col])
                                   for col in continuous_cols])
    continuous_bn = BatchNormalization()(continuous_bn)
    x = Concatenate()(embeddings + [continuous_bn])
    x = Flatten()(x)
    x = Dense(1000, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.00005))(x)
```

```
    x = Dense(1000, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.00005))(x)
    x = Dense(1000, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.00005))(x)
    x = Dense(500, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.00005))(x)
    x = Dropout(0.5)(x)
    output = Dense(1, activation=act_sigmoid_scaled)(x)
    model = tf.keras.Model([inputs[f] for f in all_cols], output)
    model.summary()
    opt = tf.keras.optimizers.Adam(lr=args.learning_rate, epsilon=1e-3)
    # Checkpoint callback to specify options for the returned Keras model
    ckpt_callback = BestModelCheckpoint(monitor='val_loss', mode='auto',
save_freq='epoch')
    # Horovod: run training.
    store = Store.create(args.work_dir)
    backend = SparkBackend(num_proc=args.num_proc,
                           stdout=sys.stdout, stderr=sys.stderr,
                           prefix_output_with_timestamp=True)
    keras_estimator = hvd.KerasEstimator(backend=backend,
                                         store=store,
                                         model=model,
                                         optimizer=opt,
                                         loss='mae',
                                         metrics=[exp_rmspe],
                                         custom_objects=CUSTOM_OBJECTS,
                                         feature_cols=all_cols,
                                         label_cols=['Sales'],
                                         validation='Validation',
                                         batch_size=args.batch_size,
                                         epochs=args.epochs,
                                         verbose=2,

checkpoint_callback=ckpt_callback)
    keras_model =
keras_estimator.fit(train_df).setOutputCols(['Sales_output'])
    history = keras_model.getHistory()
    best_val_rmspe = min(history['val_exp_rmspe'])
    print('Best RMSPE: %f' % best_val_rmspe)
    # Save the trained model.
    keras_model.save(args.local_checkpoint_file)
    print('Written checkpoint to %s' % args.local_checkpoint_file)
    # =============== #
    # FINAL PREDICTION #
    # =============== #
    print('===============')
```

```
    print('Final prediction')
    print('================')
    pred_df=keras_model.transform(test_df)
    pred_df.printSchema()
    pred_df.show(5)
    # Convert from log domain to real Sales numbers
    pred_df=pred_df.withColumn('Sales_pred', F.exp(pred_df.Sales_output))
    submission_df = pred_df.select(pred_df.Id.cast(T.IntegerType()),
pred_df.Sales_pred).toPandas()
    submission_df.sort_values(by=['Id']).to_csv(args.local_submission_csv,
index=False)
    print('Saved predictions to %s' % args.local_submission_csv)
    spark.stop()
```

The third script is `run_classification_criteo_spark.py`.

```
import tempfile, string, random, os, uuid
import argparse, datetime, sys, shutil
import csv
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import EarlyStopping
from pyspark import SparkContext
from pyspark.sql import SparkSession, SQLContext, Row, DataFrame
from pyspark.mllib import linalg as mllib_linalg
from pyspark.mllib.linalg import SparseVector as mllibSparseVector
from pyspark.mllib.linalg import VectorUDT as mllibVectorUDT
from pyspark.mllib.linalg import Vector as mllibVector, Vectors as
mllibVectors
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.classification import LogisticRegressionWithSGD
from pyspark.ml import linalg as ml_linalg
from pyspark.ml.linalg import VectorUDT as mlVectorUDT
from pyspark.ml.linalg import SparseVector as mlSparseVector
from pyspark.ml.linalg import Vector as mlVector, Vectors as mlVectors
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import OneHotEncoder
from math import log
from math import exp  # exp(-t) = e^-t
from operator import add
from pyspark.sql.functions import udf, split, lit
from pyspark.sql.functions import size, sum as sqlsum
import pyspark.sql.functions as F
import pyspark.sql.types as T
from pyspark.sql.types import ArrayType, StructType, StructField,
```

```
LongType, StringType, IntegerType, FloatType
from pyspark.sql.functions import explode, col, log, when
from collections import defaultdict
import pandas as pd
import pyspark.pandas as ps
from sklearn.metrics import log_loss, roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from deepctr.models import DeepFM
from deepctr.feature_column import SparseFeat, DenseFeat,
get_feature_names
spark = SparkSession.builder \
    .master("yarn") \
    .appName("deep_ctr_classification") \
    .config("spark.jars.packages", "io.github.ravwojdyla:spark-schema-
utils_2.12:0.1.0") \
    .config("spark.executor.cores", "1") \
    .config('spark.executor.memory', '5gb') \
    .config('spark.executor.memoryOverhead', '1500') \
    .config('spark.driver.memoryOverhead', '1500') \
    .config("spark.sql.shuffle.partitions", "480") \
    .config("spark.sql.execution.arrow.enabled", "true") \
    .config("spark.driver.maxResultSize", "50gb") \
    .getOrCreate()
# spark.conf.set("spark.sql.execution.arrow.enabled", "true") # deprecated
print("Apache Spark version:")
print(spark.version)
sc = spark.sparkContext
sqlContext = SQLContext(sc)
parser = argparse.ArgumentParser(description='Spark DCN CTR Prediction
Example',

formatter_class=argparse.ArgumentDefaultsHelpFormatter)
parser.add_argument('--data-dir', default='file://' + os.getcwd(),
                    help='location of data on local filesystem (prefixed
with file://) or on HDFS')
def process_input_file(file_name, sparse_feat, dense_feat):
    # Need this preprocessing to turn Criteo raw file into CSV:
    print("START processing input file...")
    # only convert the file ONCE
    # sample = open(file_name)
    # sample = '\n'.join([str(x.replace('\n', '').replace('\t', ',')) for
x in sample])
    # # Add header in data file and save as CSV
    # header = ','.join(str(x) for x in (['label'] + dense_feat +
sparse_feat))
```

```
    # with open('/sparkdemo/tr-4570-data/ctr_train.csv', mode='w',
encoding="utf-8") as f:
    #     f.write(header + '\n' + sample)
    #     f.close()
    # print("Raw training file processed and saved as CSV: ", f.name)
    raw_df = sqlContext.read.option("header", True).csv(file_name)
    raw_df.show(5, False)
    raw_df.printSchema()
    # convert columns I1 to I13 from string to integers
    conv_df = raw_df.select(col('label').cast("double"),
                            *(col(i).cast("float").alias(i) for i in
raw_df.columns if i in dense_feat),
                            *(col(c) for c in raw_df.columns if c in
sparse_feat))
    print("Schema of raw_df with integer columns type changed:")
    conv_df.printSchema()
    # result_pdf = conv_df.select("*").toPandas()
    tmp_df = conv_df.na.fill(0, dense_feat)
    result_df = tmp_df.na.fill('-1', sparse_feat)
    result_df.show()
    return result_df
if __name__ == "__main__":
    args = parser.parse_args()
    # Pandas read CSV
    # data = pd.read_csv('%s/criteo_sample.txt' % args.data_dir)
    # print("Obtained Pandas df.")
    dense_features = ['I' + str(i) for i in range(1, 14)]
    sparse_features = ['C' + str(i) for i in range(1, 27)]
    # Spark read CSV
    # process_input_file('%s/train.txt' % args.data_dir, sparse_features,
dense_features) # run only ONCE
    spark_df = process_input_file('%s/data.txt' % args.data_dir,
sparse_features, dense_features) # sample data
    # spark_df = process_input_file('%s/ctr_train.csv' % args.data_dir,
sparse_features, dense_features)
    print("Obtained Spark df and filled in missing features.")
    data = spark_df
    # Pandas
    #data[sparse_features] = data[sparse_features].fillna('-1', )
    #data[dense_features] = data[dense_features].fillna(0, )
    target = ['label']
    label_npa = data.select("label").toPandas().to_numpy()
    print("label numPy array has length = ", len(label_npa)) # 45,840,617
w/ 11GB dataset
    label_npa.ravel()
    label_npa.reshape(len(label_npa), )
```

```
    # 1.Label Encoding for sparse features,and do simple Transformation
for dense features
    print("Before LabelEncoder():")
    data.printSchema()  # label: float (nullable = true)
    for feat in sparse_features:
        lbe = LabelEncoder()
        tmp_pdf = data.select(feat).toPandas().to_numpy()
        tmp_ndarray = lbe.fit_transform(tmp_pdf)
        print("After LabelEncoder(), tmp_ndarray[0] =", tmp_ndarray[0])
        # print("Data tmp PDF after lbe transformation, the output ndarray
has length = ", len(tmp_ndarray)) # 45,840,617 for 11GB dataset
        tmp_ndarray.ravel()
        tmp_ndarray.reshape(len(tmp_ndarray), )
        out_ndarray = np.column_stack([label_npa, tmp_ndarray])
        pdf = pd.DataFrame(out_ndarray, columns=['label', feat])
        s_df = spark.createDataFrame(pdf)
        s_df.printSchema() # label: double (nullable = true)
        print("Before joining data df with s_df, s_df example rows:")
        s_df.show(1, False)
        data = data.drop(feat).join(s_df, 'label').drop('label')
        print("After LabelEncoder(), data df example rows:")
        data.show(1, False)
        print("Finished processing sparse_features: ", feat)
    print("Data DF after label encoding: ")
    data.show()
    data.printSchema()
    mms = MinMaxScaler(feature_range=(0, 1))
    # data[dense_features] = mms.fit_transform(data[dense_features]) # for
Pandas df
    tmp_pdf = data.select(dense_features).toPandas().to_numpy()
    tmp_ndarray = mms.fit_transform(tmp_pdf)
    tmp_ndarray.ravel()
    tmp_ndarray.reshape(len(tmp_ndarray), len(tmp_ndarray[0]))
    out_ndarray = np.column_stack([label_npa, tmp_ndarray])
    pdf = pd.DataFrame(out_ndarray, columns=['label'] + dense_features)
    s_df = spark.createDataFrame(pdf)
    s_df.printSchema()
    data.drop(*dense_features).join(s_df, 'label').drop('label')
    print("Finished processing dense_features: ", dense_features)
    print("Data DF after MinMaxScaler: ")
    data.show()

    # 2.count #unique features for each sparse field,and record dense
feature field name
    fixlen_feature_columns = [SparseFeat(feat,
vocabulary_size=data.select(feat).distinct().count() + 1, embedding_dim=4)
```

```
                              for i, feat in enumerate(sparse_features)] +
\
                             [DenseFeat(feat, 1, ) for feat in
dense_features]
    dnn_feature_columns = fixlen_feature_columns
    linear_feature_columns = fixlen_feature_columns
    feature_names = get_feature_names(linear_feature_columns +
dnn_feature_columns)
    # 3.generate input data for model
    # train, test = train_test_split(data.toPandas(), test_size=0.2,
random_state=2020) # Pandas; might hang for 11GB data
    train, test = data.randomSplit(weights=[0.8, 0.2], seed=200)
    print("Training dataset size = ", train.count())
    print("Testing dataset size = ", test.count())
    # Pandas:
    # train_model_input = {name: train[name] for name in feature_names}
    # test_model_input = {name: test[name] for name in feature_names}
    # Spark DF:
    train_model_input = {}
    test_model_input = {}
    for name in feature_names:
        if name.startswith('I'):
            tr_pdf = train.select(name).toPandas()
            train_model_input[name] = pd.to_numeric(tr_pdf[name])
            ts_pdf = test.select(name).toPandas()
            test_model_input[name] = pd.to_numeric(ts_pdf[name])
    # 4.Define Model,train,predict and evaluate
    model = DeepFM(linear_feature_columns, dnn_feature_columns,
task='binary')
    model.compile("adam", "binary_crossentropy",
                  metrics=['binary_crossentropy'], )
    lb_pdf = train.select(target).toPandas()
    history = model.fit(train_model_input,
pd.to_numeric(lb_pdf['label']).values,
                        batch_size=256, epochs=10, verbose=2,
validation_split=0.2, )
    pred_ans = model.predict(test_model_input, batch_size=256)
    print("test LogLoss",
round(log_loss(pd.to_numeric(test.select(target).toPandas()).values,
pred_ans), 4))
    print("test AUC",
round(roc_auc_score(pd.to_numeric(test.select(target).toPandas()).values,
pred_ans), 4))
```

Next: Conclusion.

Conclusion

Previous: Python scripts for each major use case.

In this document, we discuss the Apache Spark architecture, customer use cases, and the NetApp storage portfolio as it relates to big data, modern analytics, and AI, ML, and DL. In our performance validation tests based on industry-standard benchmarking tools and customer demand, the NetApp Spark solutions demonstrated superior performance relative to native Hadoop systems. A combination of the customer use cases and performance results presented in this report can help you to choose an appropriate Spark solution for your deployment.

Next: Where to find additional information.

Where to find additional information

Previous: Conclusion.

The following references were used in this TR:

- Apache Spark architecture and components

  http://spark.apache.org/docs/latest/cluster-overview.html

- Apache Spark use cases

  https://www.qubole.com/blog/big-data/apache-spark-use-cases/

- Apache challenges

  http://www.infoworld.com/article/2897287/big-data/5-reasons-to-turn-to-spark-for-big-data-analytics.html

- Spark NLP

  https://www.johnsnowlabs.com/spark-nlp/

- BERT

  https://arxiv.org/abs/1810.04805

- Deep and Cross Network for Ad Click Predictions

  https://arxiv.org/abs/1708.05123

- FlexGroup

  http://www.netapp.com/us/media/tr-4557.pdf

- Streaming ETL

  https://www.infoq.com/articles/apache-spark-streaming

- NetApp E-Series Solutions for Hadoop

  https://www.netapp.com/media/16420-tr-3969.pdf

- Sentiment Analysis from Customer Communications with NetApp AI

  https://docs.netapp.com/us-en/netapp-solutions/pdfs/sidebar/Sentiment_analysis_with_NetApp_AI.pdf

- NetApp Modern Data Analytics Solutions

  https://docs.netapp.com/us-en/netapp-solutions/data-analytics/index.html

- SnapMirror

  https://docs.netapp.com/us-en/ontap/data-protection/snapmirror-replication-concept.html

- XCP

  https://mysupport.netapp.com/documentation/docweb/index.html?productID=63942&language=en-US

- Cloud Sync

  https://cloud.netapp.com/cloud-sync-service

- DataOps Toolkit

  https://github.com/NetApp/netapp-dataops-toolkit

Big Data Analytics Data to Artificial Intelligence

TR-4732: Big data analytics data to artificial intelligence

Karthikeyan Nagalingam, NetApp

This document describes how to move big-data analytics data and HPC data to AI. AI processes NFS data through NFS exports, whereas customers often have their AI data in a big-data analytics platform, such as HDFS, Blob, or S3 storage as well as HPC platforms such as GPFS. This paper provides guidelines for moving big-data-analytics data and HPC data to AI by using NetApp XCP and NIPAM. We also discuss the business benefits of moving data from big data and HPC to AI.

Concepts and components

Big data analytics storage

Big data analytics is the major storage provider for HDFS. A customer often uses a Hadoop-compatible file system (HCFS) such as Windows Azure Blob Storage, MapR File System (MapR-FS), and S3 object storage.

General parallel file system

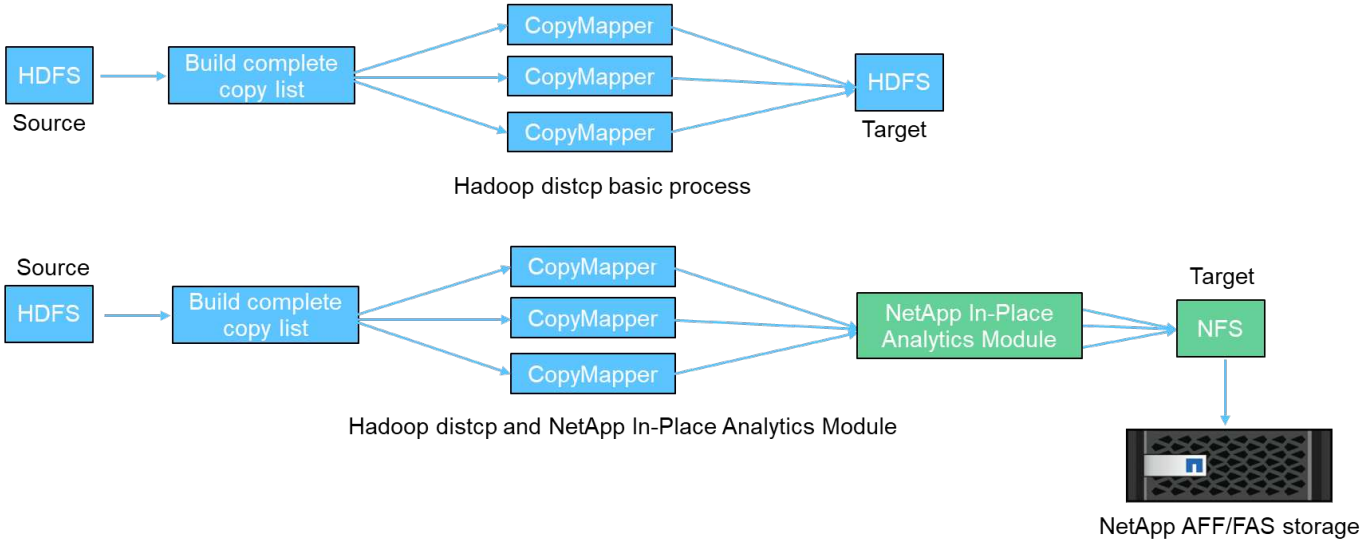
IBM's GPFS is an enterprise file system that provides an alternative to HDFS. GPFS provides flexibility for applications to decide the block size and replication layout, which provide good performance and efficiency.

NetApp In-Place Analytics Module

The NetApp In-Place Analytics Module (NIPAM) serves as a driver for Hadoop clusters to access NFS data. It has four components: a connection pool, an NFS InputStream, a file handle cache, and an NFS OutputStream. For more information, see TR-4382: NetApp In-Place Analytics Module.

Hadoop Distributed Copy

Hadoop Distributed Copy (DistCp) is a distributed copy tool used for large inter-cluster and intra-cluster coping tasks. This tool uses MapReduce for data distribution, error handling, and reporting. It expands the list of files and directories and inputs them to map tasks to copy the data from the source list. The image below shows the DistCp operation in HDFS and nonHDFS.



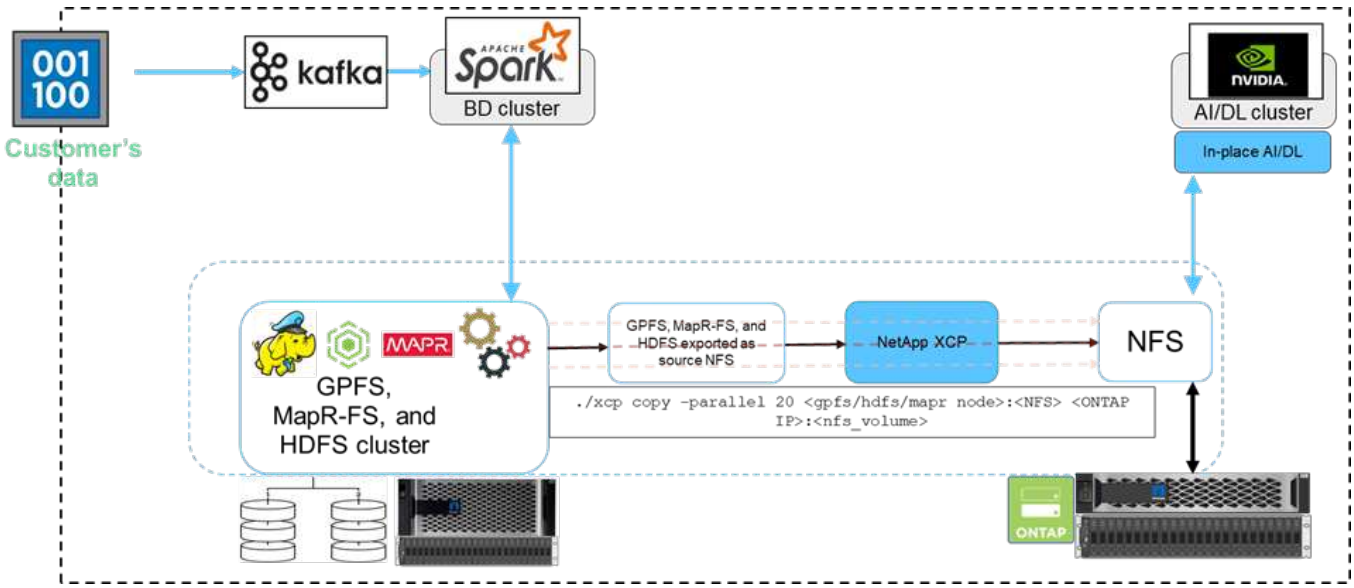
Hadoop DistCp moves data between the two HDFS systems without using an additional driver. NetApp provides the driver for non-HDFS systems. For an NFS destination, NIPAM provides the driver to copy data that Hadoop DistCp uses to communicate with NFS destinations when copying data.

NetApp Cloud Volumes Service

The NetApp Cloud Volumes Service is a cloud-native file service with extreme performance. This service helps customers accelerate their time-to-market by rapidly spinning resources up and down and using NetApp features to improve productivity and reduce staff downtime. The Cloud Volumes Service is the right alternative for disaster recovery and back up to cloud because it reduces the overall data-center footprint and consumes less native public cloud storage.

NetApp XCP

NetApp XCP is client software that enables fast and reliable any-to-NetApp and NetApp-to-NetApp data migration. This tool is designed to copy a large amount of unstructured NAS data from any NAS system to a NetApp storage controller. The XCP Migration Tool uses a multicore, multichannel I/O streaming engine that can process many requests in parallel, such as data migration, file or directory listings, and space reporting. This is the default NetApp data Migration Tool. You can use XCP to copy data from a Hadoop cluster and HPC to NetApp NFS storage. The diagram below shows data transfer from a Hadoop and HPC cluster to a NetApp NFS volume using XCP.



NetApp Cloud Sync

NetApp Cloud Sync is a hybrid data replication software-as-a-service that transfers and synchronizes NFS, S3, and CIFS data seamlessly and securely between on-premises storage and cloud storage. This software is used for data migration, archiving, collaboration, analytics, and more. After data is transferred, Cloud Sync continuously syncs the data between the source and destination. Going forward, it then transfers the delta. It also secures the data within your own network, in the cloud, or on premises. This software is based on a pay-as-you-go model, which provides a cost-effective solution and provides monitoring and reporting capabilities for your data transfer.

Next: Customer challenges.

Customer challenges

Previous: Introduction.

Customers might face the following challenges when trying to access data from big-data analytics for AI operations:

- Customer data is in a data lake repository. The data lake can contain different types of data such as structured, unstructured, semi-structured, logs, and machine-to-machine data. All these data types must be processed in AI systems.
- AI is not compatible with Hadoop file systems. A typical AI architecture is not able to directly access HDFS and HCFS data, which must be moved to an AI-understandable file system (NFS).
- Moving data lake data to AI typically requires specialized processes. The amount of data in the data lake can be very large. A customer must have an efficient, high-throughput, and cost-effective way to move data into AI systems.
- Syncing data. If a customer wants to sync data between the big-data platform and AI, sometimes the data processed through AI can be used with big data for analytical processing.

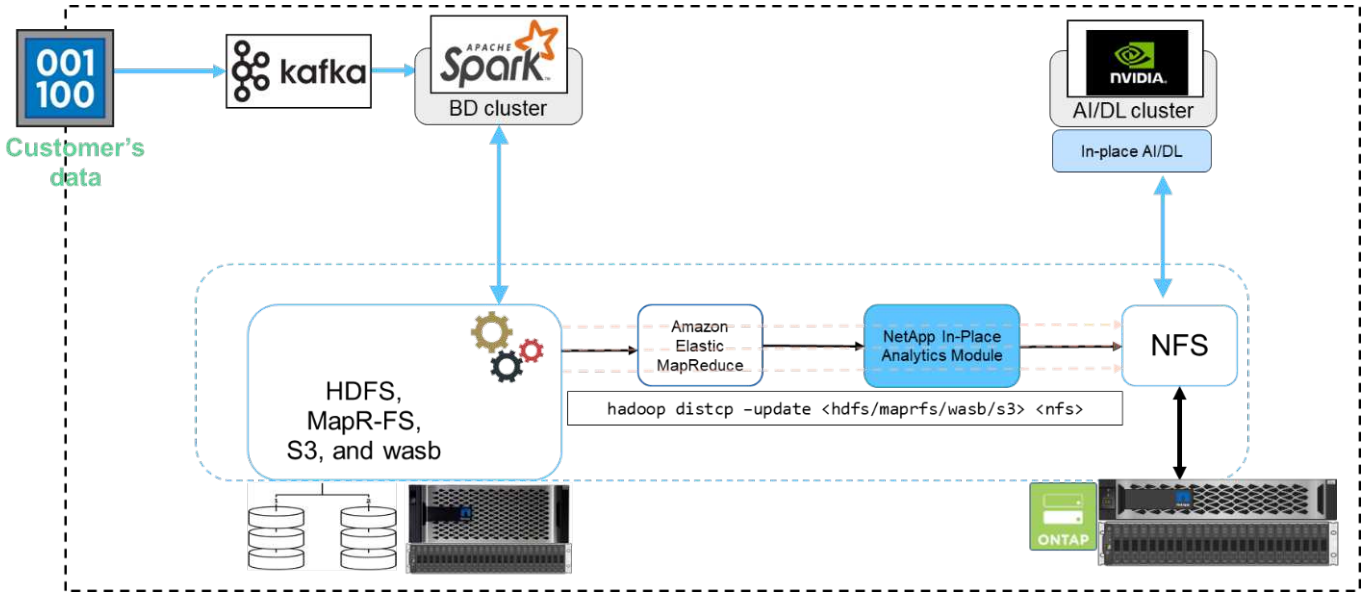
Next: Data mover solution.

Data mover solution

Previous: Customer challenges.

In a big-data cluster, data is stored in HDFS or HCFS, such as MapR-FS, the Windows Azure Storage Blob, S3, or the Google file system. We performed testing with HDFS, MapR-FS, and S3 as the source to copy data to NetApp ONTAP NFS export with the help of NIPAM by using the `hadoop distcp` command from the source.

The following diagram illustrates the typical data movement from a Spark cluster running with HDFS storage to a NetApp ONTAP NFS volume so that NVIDIA can process AI operations.



The `hadoop distcp` command uses the MapReduce program to copy the data. NIPAM works with MapReduce to act as a driver for the Hadoop cluster when copying data. NIPAM can distribute a load across multiple network interfaces for a single export. This process maximizes the network throughput by distributing the data across multiple network interfaces when you copy the data from HDFS or HCFS to NFS.

NIPAM is not supported or certified with MapR.

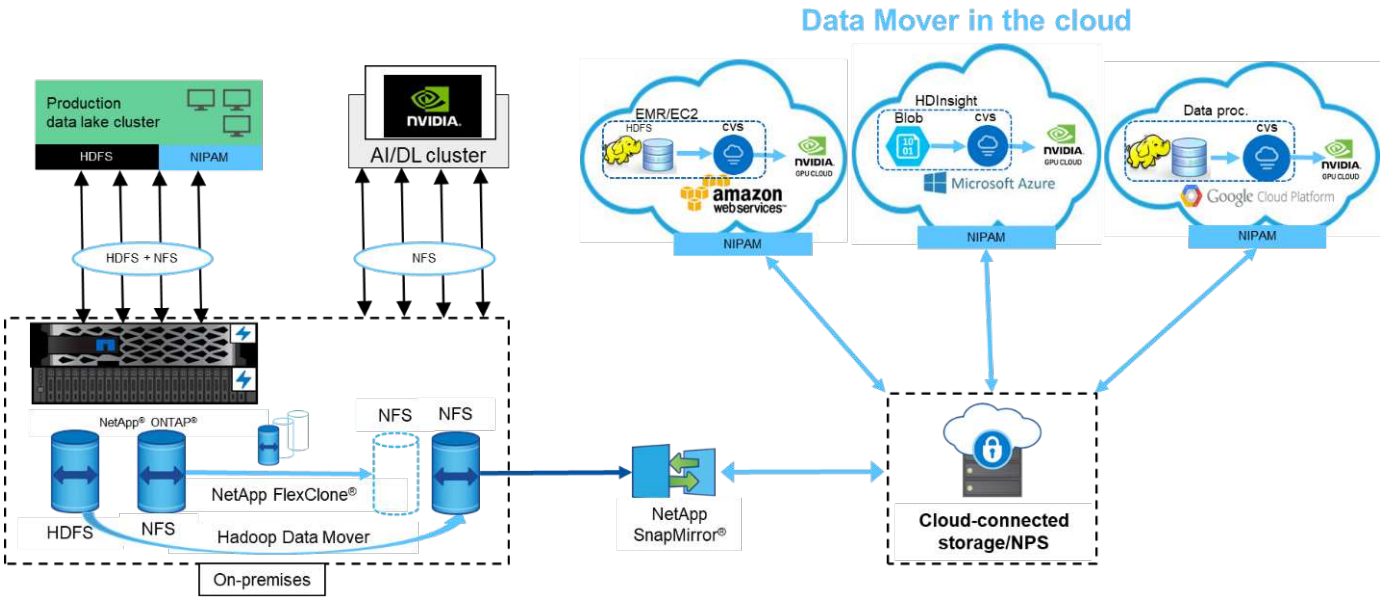
Next: Data mover solution for AI.

Data mover solution for AI

Previous: Data mover solution.

The data mover solution for AI is based on customers' needs to process Hadoop data from AI operations. NetApp moves data from HDFS to NFS by using the NIPAM. In one use case, the customer needed to move data to NFS on the premises and another customer needed to move data from the Windows Azure Storage Blob to Cloud Volumes Service in order to process the data from the GPU cloud instances in the cloud.

The following diagram illustrates the data mover solution details.



The following steps are required to build the data mover solution:

1. ONTAP SAN provides HDFS, and NAS provides the NFS volume through NIPAM to the production data lake cluster.
2. The customer's data is in HDFS and NFS. The NFS data can be production data from other applications that is used for big data analytics and AI operations.
3. NetApp FlexClone technology creates a clone of the production NFS volume and provisions it to the AI cluster on premises.
4. Data from an HDFS SAN LUN is copied into an NFS volume with NIPAM and the `hadoop distcp` command. NIPAM uses the bandwidth of multiple network interfaces to transfer data. This process reduces the data copy time so that more data can be transferred.
5. Both NFS volumes are provisioned to the AI cluster for AI operations.
6. To process on-the-premises NFS data with GPUs in the cloud, the NFS volumes are mirrored to NetApp Private Storage (NPS) with NetApp SnapMirror technology and mounted to cloud service providers for GPUs.
7. The customer wants to process data in EC2/EMR, HDInsight, or DataProc services in GPUs from cloud service providers. The Hadoop data mover moves the data from Hadoop services to the Cloud Volumes Services with NIPAM and the `hadoop distcp` command.
8. The Cloud Volumes Service data is provisioned to AI through the NFS protocol.Data that is processed through AI can be sent on an on-premises location for big data analytics in addition to the NVIDIA cluster through NIPAM, SnapMirror, and NPS.

In this scenario, the customer has large file-count data in the NAS system at a remote location that is required for AI processing on the NetApp storage controller on premises. In this scenario, it's better to use the XCP Migration Tool to migrate the data at a faster speed.

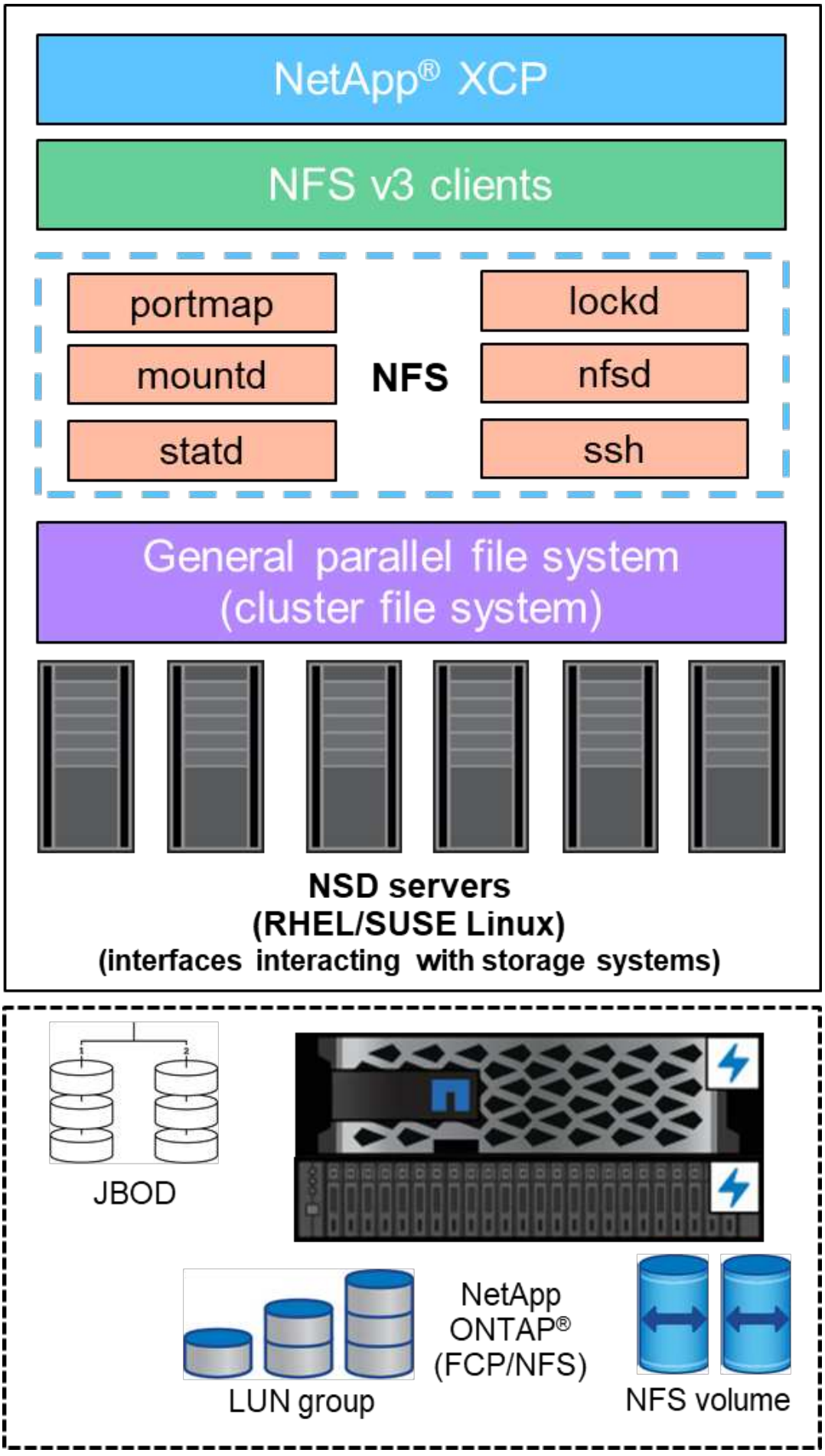
The hybrid-use-case customer can use Cloud Sync to migrate on-premises data from NFS, CIFS, and S3 data to the cloud and vice versa for AI processing by using GPUs such as those in an NVIDIA cluster. Both Cloud Sync and the XCP Migration Tool are used for the NFS data migration to NetApp ONTAP NFS.

Next: GPFS to NetApp ONTAP NFS.

GPFS to NetApp ONTAP NFS

Previous: Data mover solution for AI.

In this validation, we used four servers as Network Shared Disk (NSD) servers to provide physical disks for GPFS. GPFS is created on top of the NSD disks to export them as NFS exports so that NFS clients can access them, as shown in the figure below. We used XCP to copy the data from GPFS- exported NFS to a NetApp NFS volume.


NetApp® XCP
NFS v3 clients
portmap
mountd
statd
NFS
lockd
nfsd
ssh
General parallel file system
(cluster file system)
NSD servers
(RHEL/SUSE Linux)
(interfaces interacting with storage systems)
JBOD
LUN group
NetApp
ONTAP®
(FCP/NFS)
NFS volume

GPFS essentials

The following node types are used in GPFS:

- **Admin node.** Specifies an optional field containing a node name used by the administration commands to communicate between nodes. For example, the admin node `mastr-51.netapp.com` could pass a network check to all other nodes in the cluster.
- **Quorum node.** Determines whether a node is included in the pool of nodes from which quorum is derived. You need at least one node as a quorum node.
- **Manager Node.** Indicates whether a node is part of the node pool from which file system managers and token managers can be selected. It is a good idea to define more than one node as a manager node. How many nodes you designate as manager depends on the workload and the number of GPFS server licenses you have. If you are running large parallel jobs, you might need more manager nodes than in a four-node cluster supporting a web application.
- **NSD Server.** The server that prepares each physical disk for use with GPFS.
- **Protocol node.** The node that shares GPFS data directly through any Secure Shell (SSH) protocol with the NFS. This node requires a GPFS server license.

List of operations for GPFS, NFS, and XCP

This section provides the list of operations that create GPFS, export GPFS as an NFS export, and transfer the data by using XCP.

Create GPFS

To create GPFS, complete the following steps:

1. Download and install spectrum-scale data access for the Linux version on one of the servers.
2. Install the prerequisite package (chef for example) in all nodes and disable Security-Enhanced Linux (SELinux) in all nodes.
3. Set up the install node and add the admin node and the GPFS node to the cluster definition file.
4. Add the manager node, the quorum node, the NSD servers, and the GPFS node.
5. Add the GUI, admin, and GPFS nodes, and add an additional GUI server if required.
6. Add another GPFS node and check the list of all nodes.
7. Specify a cluster name, profile, remote shell binary, remote file copy binary, and port range to be set on all the GPFS nodes in the cluster definition file.
8. View the GPFS configuration settings and add an additional admin node.
9. Disable the data collection and upload the data package to the IBM Support Center.
10. Enable NTP and precheck the configurations before install.
11. Configure, create, and check the NSD disks.
12. Create the GPFS.
13. Mount the GPFS.
14. Verify and provide the required permissions to the GPFS.
15. Verify the GPFS read and write by running the `dd` command.

Export GPFS into NFS

To export the GPFS into NFS, complete the following steps:

1. Export GPFS as NFS through the `/etc/exports` file.
2. Install the required NFS server packages.
3. Start the NFS service.
4. List the files in the GPFS to validate the NFS client.

Configure NFS client

To configure the NFS client, complete the following steps:

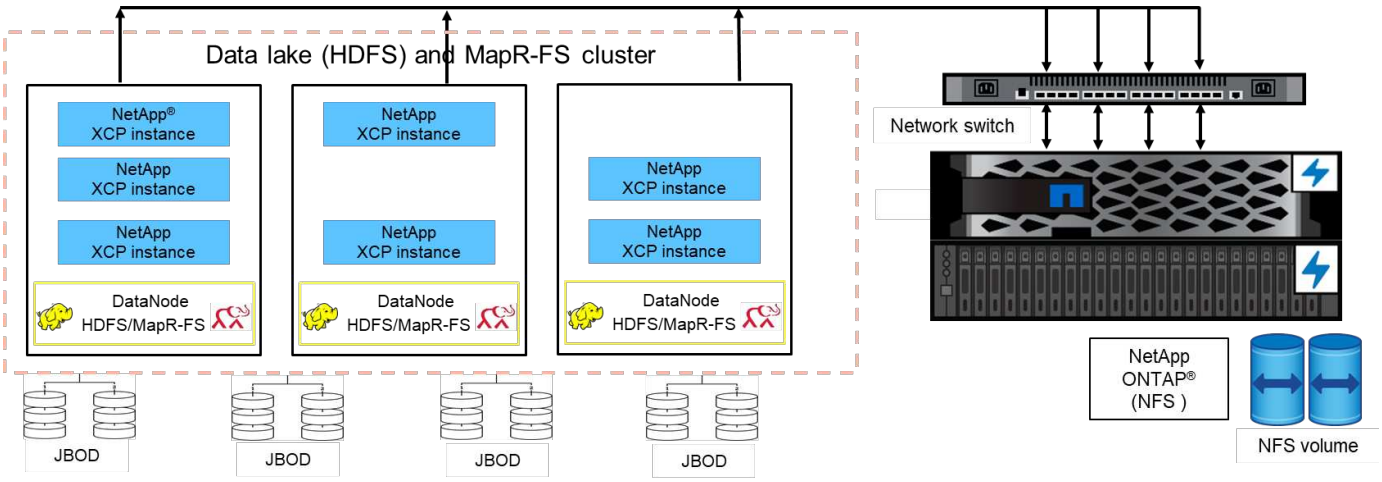
1. Export the GPFS as NFS through the `/etc/exports` file.
2. Start the NFS client services.
3. Mount the GPFS through the NFS protocol on the NFS client.
4. Validate the list of GPFS files in the NFS mounted folder.
5. Move the data from GPFS exported NFS to NetApp NFS by using XCP.
6. Validate the GPFS files on the NFS client.

Next: HDFS and MapR-FS to ONTAP NFS.

HDFS and MapR-FS to ONTAP NFS

Previous: GPFS to NetApp ONTAP NFS.

For this solution, NetApp validated the migration of data from data lake (HDFS) and MapR cluster data to ONTAP NFS. The data resided in MapR-FS and HDFS. NetApp XCP introduced a new feature that directly migrates the data from a distributed file system such as HDFS and MapR-FS to ONTAP NFS. XCP uses async threads and HDFS C API calls to communicate and transfer data from MapR- FS as well as HDFS. The below figure shows the data migration from data lake (HDFS) and MapR-FS to ONTAP NFS. With this new feature, you don't have to export the source as an NFS share.



Why are customers moving from HDFS and MapR-FS to NFS?

Most of the Hadoop distributions such as Cloudera and Hortonworks use HDFS and MapR distributions uses their own filesystem called Mapr-FS to store data. HDFS and MapR-FS data provides the valuable insights to data scientists that can be leveraged in machine learning (ML) and deep learning (DL). The data in HDFS and MapR-FS is not shared, which means it cannot be used by other applications. Customers are looking for shared data, specifically in the banking sector where customers' sensitive data is used by multiple applications. The latest version of Hadoop (3.x or later) supports NFS data source, which can be accessed without additional third-party software. With the new NetApp XCP feature, data can be moved directly from HDFS and MapR-FS to NetApp NFS in order to provide access to multiple applications

Testing was done in Amazon Web Services (AWS) to transfer the data from MapR-FS to NFS for the initial performance test with 12 MAPR nodes and 4 NFS servers.

	Quantity	Size	vCPU	Memory	Storage	Network
NFS server	4	i3en.24xlarge	96	488GiB	8x 7500 NVMe SSD	100
MapR nodes	12	I3en.12xlarge	48	384GiB	4x 7500 NVMe SSD	50

Based on initial testing, we obtained 20GBps throughput and were able to transfer 2PB per day of data.

For more information about HDFS data migration without exporting HDFS to NFS, see the "Deployment steps - NAS" section in TR-4863: TR-4863: Best-Practice Guidelines for NetApp XCP - Data Mover, File Migration, and Analytics.

Next: Business benefits.

Business benefits

Previous: HDFS and MapR-FS to ONTAP NFS.

Moving data from big data analytics to AI provides the following benefits:

- The ability to extract data from different Hadoop file systems and GPFS into a unified NFS storage system
- A Hadoop-integrated and automated way to transfer data
- A reduction in the cost of library development for moving data from Hadoop file systems
- Maximum performance by aggregated throughput of multiple network interfaces from a single source of data by using NIPAM
- Scheduled and on-demand methods to transfer data
- Storage efficiency and enterprise management capability for unified NFS data by using ONTAP data management software
- Zero cost for data movement with the Hadoop method for data transfer

Next: GPFS to NFS-Detailed steps.

GPFS to NFS-Detailed steps

Previous: Business benefits.

This section provides the detailed steps needed to configure GPFS and move data into NFS by using NetApp XCP.

Configure GPFS

1. Download and Install Spectrum Scale Data Access for Linux on one of the servers.

```
[root@mastr-51 Spectrum_Scale_Data_Access-5.0.3.1-x86_64-Linux-
install_folder]# ls
Spectrum_Scale_Data_Access-5.0.3.1-x86_64-Linux-install
[root@mastr-51 Spectrum_Scale_Data_Access-5.0.3.1-x86_64-Linux-
install_folder]# chmod +x Spectrum_Scale_Data_Access-5.0.3.1-x86_64-
Linux-install
[root@mastr-51 Spectrum_Scale_Data_Access-5.0.3.1-x86_64-Linux-
install_folder]# ./Spectrum_Scale_Data_Access-5.0.3.1-x86_64-Linux-
install --manifest
manifest
…
<contents removes to save page space>
…
```

2. Install the prerequisite package (including chef and the kernel headers) on all nodes.

```
[root@mastr-51 5.0.3.1]# for i in 51 53 136 138 140  ; do ssh
10.63.150.$i "hostname; rpm -ivh  /gpfs_install/chef* "; done
mastr-51.netapp.com
warning: /gpfs_install/chef-13.6.4-1.el7.x86_64.rpm: Header V4 DSA/SHA1
Signature, key ID 83ef826a: NOKEY
Preparing...
########################################
package chef-13.6.4-1.el7.x86_64 is already installed
mastr-53.netapp.com
warning: /gpfs_install/chef-13.6.4-1.el7.x86_64.rpm: Header V4 DSA/SHA1
Signature, key ID 83ef826a: NOKEY
Preparing...
########################################
Updating / installing...
chef-13.6.4-1.el7
########################################
Thank you for installing Chef!
workr-136.netapp.com
warning: /gpfs_install/chef-13.6.4-1.el7.x86_64.rpm: Header V4 DSA/SHA1
Signature, key ID 83ef826a: NOKEY
Preparing...
########################################
Updating / installing...
```

```
chef-13.6.4-1.el7
########################################
Thank you for installing Chef!
workr-138.netapp.com
warning: /gpfs_install/chef-13.6.4-1.el7.x86_64.rpm: Header V4 DSA/SHA1
Signature, key ID 83ef826a: NOKEY
Preparing...
########################################
Updating / installing...
chef-13.6.4-1.el7
########################################
Thank you for installing Chef!
workr-140.netapp.com
warning: /gpfs_install/chef-13.6.4-1.el7.x86_64.rpm: Header V4 DSA/SHA1
Signature, key ID 83ef826a: NOKEY
Preparing...
########################################
Updating / installing...
chef-13.6.4-1.el7
########################################
Thank you for installing Chef!
[root@mastr-51 5.0.3.1]#
[root@mastr-51 installer]# for i in 51 53 136 138 140  ; do ssh
10.63.150.$i "hostname; yumdownloader kernel-headers-3.10.0-
862.3.2.el7.x86_64 ; rpm -Uvh --oldpackage kernel-headers-3.10.0-
862.3.2.el7.x86_64.rpm"; done
mastr-51.netapp.com
Loaded plugins: priorities, product-id, subscription-manager
Preparing...
########################################
Updating / installing...
kernel-headers-3.10.0-862.3.2.el7
########################################
Cleaning up / removing...
kernel-headers-3.10.0-957.21.2.el7
########################################
mastr-53.netapp.com
Loaded plugins: product-id, subscription-manager
Preparing...
########################################
Updating / installing...
kernel-headers-3.10.0-862.3.2.el7
########################################
Cleaning up / removing...
kernel-headers-3.10.0-862.11.6.el7
########################################
```

```
workr-136.netapp.com
Loaded plugins: product-id, subscription-manager
Repository ambari-2.7.3.0 is listed more than once in the configuration
Preparing...
########################################
Updating / installing...
kernel-headers-3.10.0-862.3.2.el7
########################################
Cleaning up / removing...
kernel-headers-3.10.0-862.11.6.el7
########################################
workr-138.netapp.com
Loaded plugins: product-id, subscription-manager
Preparing...
########################################
package kernel-headers-3.10.0-862.3.2.el7.x86_64 is already installed
workr-140.netapp.com
Loaded plugins: product-id, subscription-manager
Preparing...
########################################
Updating / installing...
kernel-headers-3.10.0-862.3.2.el7
########################################
Cleaning up / removing...
kernel-headers-3.10.0-862.11.6.el7
########################################
[root@mastr-51 installer]#
```

3. Disable SELinux in all nodes.

```
[root@mastr-51 5.0.3.1]# for i in 51 53 136 138 140  ; do ssh
10.63.150.$i "hostname; sudo setenforce 0"; done
mastr-51.netapp.com
setenforce: SELinux is disabled
mastr-53.netapp.com
setenforce: SELinux is disabled
workr-136.netapp.com
setenforce: SELinux is disabled
workr-138.netapp.com
setenforce: SELinux is disabled
workr-140.netapp.com
setenforce: SELinux is disabled
[root@mastr-51 5.0.3.1]#
```

4. Set up the install node.

```
[root@mastr-51 installer]# ./spectrumscale setup -s 10.63.150.51
[ INFO  ] Installing prerequisites for install node
[ INFO  ] Existing Chef installation detected. Ensure the PATH is
configured so that chef-client and knife commands can be run.
[ INFO  ] Your control node has been configured to use the IP
10.63.150.51 to communicate with other nodes.
[ INFO  ] Port 8889 will be used for chef communication.
[ INFO  ] Port 10080 will be used for package distribution.
[ INFO  ] Install Toolkit setup type is set to Spectrum Scale (default).
If an ESS is in the cluster, run this command to set ESS mode:
./spectrumscale setup -s server_ip -st ess
[ INFO  ] SUCCESS
[ INFO  ] Tip : Designate protocol, nsd and admin nodes in your
environment to use during install:./spectrumscale -v node add <node> -p
-a -n
[root@mastr-51 installer]#
```

5. Add the admin node and the GPFS node to the cluster definition file.

```
[root@mastr-51 installer]# ./spectrumscale node add mastr-51 -a
[ INFO  ] Adding node mastr-51.netapp.com as a GPFS node.
[ INFO  ] Setting mastr-51.netapp.com as an admin node.
[ INFO  ] Configuration updated.
[ INFO  ] Tip : Designate protocol or nsd nodes in your environment to
use during install:./spectrumscale node add <node> -p -n
[root@mastr-51 installer]#
```

6. Add the manager node and the GPFS node.

```
[root@mastr-51 installer]# ./spectrumscale node add mastr-53 -m
[ INFO  ] Adding node mastr-53.netapp.com as a GPFS node.
[ INFO  ] Adding node mastr-53.netapp.com as a manager node.
[root@mastr-51 installer]#
```

7. Add the quorum node and the GPFS node.

```
[root@mastr-51 installer]# ./spectrumscale node add workr-136 -q
[ INFO  ] Adding node workr-136.netapp.com as a GPFS node.
[ INFO  ] Adding node workr-136.netapp.com as a quorum node.
[root@mastr-51 installer]#
```

8. Add the NSD servers and the GPFS node.

```
[root@mastr-51 installer]# ./spectrumscale node add workr-138 -n
[ INFO  ] Adding node workr-138.netapp.com as a GPFS node.
[ INFO  ] Adding node workr-138.netapp.com as an NSD server.
[ INFO  ] Configuration updated.
[ INFO  ] Tip :If all node designations are complete, add NSDs to your
cluster definition and define required filessytems:./spectrumscale nsd
add <device> -p <primary node> -s <secondary node> -fs <file system>
[root@mastr-51 installer]#
```

9. Add the GUI, admin, and GPFS nodes.

```
[root@mastr-51 installer]# ./spectrumscale node add workr-136 -g
[ INFO  ] Setting workr-136.netapp.com as a GUI server.
[root@mastr-51 installer]# ./spectrumscale node add workr-136 -a
[ INFO  ] Setting workr-136.netapp.com as an admin node.
[ INFO  ] Configuration updated.
[ INFO  ] Tip : Designate protocol or nsd nodes in your environment to
use during install:./spectrumscale node add <node> -p -n
[root@mastr-51 installer]#
```

10. Add another GUI server.

```
[root@mastr-51 installer]# ./spectrumscale node add mastr-53 -g
[ INFO  ] Setting mastr-53.netapp.com as a GUI server.
[root@mastr-51 installer]#
```

11. Add another GPFS node.

```
[root@mastr-51 installer]# ./spectrumscale node add workr-140
[ INFO  ] Adding node workr-140.netapp.com as a GPFS node.
[root@mastr-51 installer]#
```

12. Verify and list all nodes.

```
[root@mastr-51 installer]# ./spectrumscale node list
[ INFO  ] List of nodes in current configuration:
[ INFO  ] [Installer Node]
[ INFO  ] 10.63.150.51
[ INFO  ]
[ INFO  ] [Cluster Details]
[ INFO  ] No cluster name configured
[ INFO  ] Setup Type: Spectrum Scale
[ INFO  ]
[ INFO  ] [Extended Features]
[ INFO  ] File Audit logging     : Disabled
[ INFO  ] Watch folder           : Disabled
[ INFO  ] Management GUI         : Enabled
[ INFO  ] Performance Monitoring : Disabled
[ INFO  ] Callhome               : Enabled
[ INFO  ]
[ INFO  ] GPFS                 Admin  Quorum  Manager   NSD   Protocol
GUI   Callhome   OS   Arch
[ INFO  ] Node                  Node   Node     Node   Server   Node
Server  Server
[ INFO  ] mastr-51.netapp.com    X
rhel7  x86_64
[ INFO  ] mastr-53.netapp.com                   X
X             rhel7  x86_64
[ INFO  ] workr-136.netapp.com   X       X
X             rhel7  x86_64
[ INFO  ] workr-138.netapp.com                          X
rhel7  x86_64
[ INFO  ] workr-140.netapp.com
rhel7  x86_64
[ INFO  ]
[ INFO  ] [Export IP address]
[ INFO  ] No export IP addresses configured
[root@mastr-51 installer]#
```

13. Specify a cluster name in the cluster definition file.

```
[root@mastr-51 installer]# ./spectrumscale config gpfs -c mastr-
51.netapp.com
[ INFO  ] Setting GPFS cluster name to mastr-51.netapp.com
[root@mastr-51 installer]#
```

14. Specify the profile.

```
[root@mastr-51 installer]# ./spectrumscale config gpfs -p default
[ INFO  ] Setting GPFS profile to default
[root@mastr-51 installer]#
Profiles options: default [gpfsProtocolDefaults], random I/O
[gpfsProtocolsRandomIO], sequential I/O [gpfsProtocolDefaults], random
I/O [gpfsProtocolRandomIO]
```

15. Specify the remote shell binary to be used by GPFS; use `-r argument`.

```
[root@mastr-51 installer]# ./spectrumscale config gpfs -r /usr/bin/ssh
[ INFO  ] Setting Remote shell command to /usr/bin/ssh
[root@mastr-51 installer]#
```

16. Specify the remote file copy binary to be used by GPFS; use `-rc argument`.

```
[root@mastr-51 installer]# ./spectrumscale config gpfs -rc /usr/bin/scp
[ INFO  ] Setting Remote file copy command to /usr/bin/scp
[root@mastr-51 installer]#
```

17. Specify the port range to be set on all GPFS nodes; use `-e argument`.

```
[root@mastr-51 installer]# ./spectrumscale config gpfs -e 60000-65000
[ INFO  ] Setting GPFS Daemon communication port range to 60000-65000
[root@mastr-51 installer]#
```

18. View the GPFS config settings.

```
[root@mastr-51 installer]# ./spectrumscale config gpfs --list
[ INFO  ] Current settings are as follows:
[ INFO  ] GPFS cluster name is mastr-51.netapp.com.
[ INFO  ] GPFS profile is default.
[ INFO  ] Remote shell command is /usr/bin/ssh.
[ INFO  ] Remote file copy command is /usr/bin/scp.
[ INFO  ] GPFS Daemon communication port range is 60000-65000.
[root@mastr-51 installer]#
```

19. Add an admin node.

```
[root@mastr-51 installer]# ./spectrumscale node add 10.63.150.53 -a
[ INFO  ] Setting mastr-53.netapp.com as an admin node.
[ INFO  ] Configuration updated.
[ INFO  ] Tip : Designate protocol or nsd nodes in your environment to
use during install:./spectrumscale node add <node> -p -n
[root@mastr-51 installer]#
```

20. Disable the data collection and upload the data package to the IBM Support Center.

```
[root@mastr-51 installer]# ./spectrumscale callhome disable
[ INFO  ] Disabling the callhome.
[ INFO  ] Configuration updated.
[root@mastr-51 installer]#
```

21. Enable NTP.

```
[root@mastr-51 installer]# ./spectrumscale config ntp -e on
[root@mastr-51 installer]# ./spectrumscale config ntp -l
[ INFO  ] Current settings are as follows:
[ WARN  ] No value for Upstream NTP Servers(comma separated IP's with NO
space between multiple IPs) in clusterdefinition file.
[root@mastr-51 installer]# ./spectrumscale config ntp -s 10.63.150.51
[ WARN  ] The NTP package must already be installed and full
bidirectional access to the UDP port 123 must be allowed.
[ WARN  ] If NTP is already running on any of your nodes, NTP setup will
be skipped. To stop NTP run 'service ntpd stop'.
[ WARN  ] NTP is already on
[ INFO  ] Setting Upstream NTP Servers(comma separated IP's with NO
space between multiple IPs) to 10.63.150.51
[root@mastr-51 installer]# ./spectrumscale config ntp -e on
[ WARN  ] NTP is already on
[root@mastr-51 installer]# ./spectrumscale config ntp -l
[ INFO  ] Current settings are as follows:
[ INFO  ] Upstream NTP Servers(comma separated IP's with NO space
between multiple IPs) is 10.63.150.51.
[root@mastr-51 installer]#

[root@mastr-51 installer]# service ntpd start
Redirecting to /bin/systemctl start ntpd.service
[root@mastr-51 installer]# service ntpd status
Redirecting to /bin/systemctl status ntpd.service
● ntpd.service - Network Time Service
   Loaded: loaded (/usr/lib/systemd/system/ntpd.service; enabled; vendor
preset: disabled)
```

```
   Active: active (running) since Tue 2019-09-10 14:20:34 UTC; 1s ago
  Process: 2964 ExecStart=/usr/sbin/ntpd -u ntp:ntp $OPTIONS
(code=exited, status=0/SUCCESS)
 Main PID: 2965 (ntpd)
   CGroup: /system.slice/ntpd.service
           └─2965 /usr/sbin/ntpd -u ntp:ntp -g

Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: ntp_io: estimated max
descriptors: 1024, initial socket boundary: 16
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: Listen and drop on 0
v4wildcard 0.0.0.0 UDP 123
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: Listen and drop on 1
v6wildcard :: UDP 123
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: Listen normally on 2 lo
127.0.0.1 UDP 123
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: Listen normally on 3
enp4s0f0 10.63.150.51 UDP 123
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: Listen normally on 4 lo
::1 UDP 123
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: Listen normally on 5
enp4s0f0 fe80::219:99ff:feef:99fa UDP 123
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: Listening on routing
socket on fd #22 for interface updates
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: 0.0.0.0 c016 06 restart
Sep 10 14:20:34 mastr-51.netapp.com ntpd[2965]: 0.0.0.0 c012 02 freq_set
kernel 11.890 PPM
[root@mastr-51 installer]#
```

22. Precheck the configurations before Install.

```
[root@mastr-51 installer]# ./spectrumscale install -pr
[ INFO  ] Logging to file: /usr/lpp/mmfs/5.0.3.1/installer/logs/INSTALL-
PRECHECK-10-09-2019_14:51:43.log
[ INFO  ] Validating configuration
[ INFO  ] Performing Chef (deploy tool) checks.
[ WARN  ] NTP is already running on: mastr-51.netapp.com. The install
toolkit will no longer setup NTP.
[ INFO  ] Node(s): ['workr-138.netapp.com'] were defined as NSD node(s)
but the toolkit has not been told about any NSDs served by these node(s)
nor has the toolkit been told to create new NSDs on these node(s). The
install will continue and these nodes will be assigned server licenses.
If NSDs are desired, either add them to the toolkit with
<./spectrumscale nsd add> followed by a <./spectrumscale install> or add
them manually afterwards using mmcrnsd.
[ INFO  ] Install toolkit will not configure file audit logging as it
has been disabled.
[ INFO  ] Install toolkit will not configure watch folder as it has been
disabled.
[ INFO  ] Checking for knife bootstrap configuration...
[ INFO  ] Performing GPFS checks.
[ INFO  ] Running environment checks
[ INFO  ] Skipping license validation as no existing GPFS cluster
detected.
[ INFO  ] Checking pre-requisites for portability layer.
[ INFO  ] GPFS precheck OK
[ INFO  ] Performing Performance Monitoring checks.
[ INFO  ] Running environment checks for Performance Monitoring
[ INFO  ] Performing GUI checks.
[ INFO  ] Performing FILE AUDIT LOGGING checks.
[ INFO  ] Running environment checks for file  Audit logging
[ INFO  ] Network check from admin node workr-136.netapp.com to all
other nodes in the cluster passed
[ INFO  ] Network check from admin node mastr-51.netapp.com to all other
nodes in the cluster passed
[ INFO  ] Network check from admin node mastr-53.netapp.com to all other
nodes in the cluster passed
[ INFO  ] The install toolkit will not configure call home as it is
disabled. To enable call home, use the following CLI command:
./spectrumscale callhome enable
[ INFO  ] Pre-check successful for install.
[ INFO  ] Tip : ./spectrumscale install
[root@mastr-51 installer]#
```

23. Configure the NSD disks.

```
[root@mastr-51 cluster-test]# cat disk.1st
%nsd: device=/dev/sdf
nsd=nsd1
servers=workr-136
usage=dataAndMetadata
failureGroup=1

%nsd: device=/dev/sdf
nsd=nsd2
servers=workr-138
usage=dataAndMetadata
failureGroup=1
```

24. Create the NSD disks.

```
[root@mastr-51 cluster-test]# mmcrnsd -F disk.1st -v no
mmcrnsd: Processing disk sdf
mmcrnsd: Processing disk sdf
mmcrnsd: Propagating the cluster configuration data to all
  affected nodes.  This is an asynchronous process.
[root@mastr-51 cluster-test]#
```

25. Check the NSD disk status.

```
[root@mastr-51 cluster-test]# mmlsnsd

 File system   Disk name    NSD servers
---------------------------------------------------------------------------
---
 (free disk)   nsd1         workr-136.netapp.com
 (free disk)   nsd2         workr-138.netapp.com

[root@mastr-51 cluster-test]#
```

26. Create the GPFS.

```
[root@mastr-51 cluster-test]# mmcrfs gpfs1 -F disk.1st -B 1M -T /gpfs1

The following disks of gpfs1 will be formatted on node workr-
136.netapp.com:
    nsd1: size 3814912 MB
    nsd2: size 3814912 MB
Formatting file system ...
Disks up to size 33.12 TB can be added to storage pool system.
Creating Inode File
Creating Allocation Maps
Creating Log Files
Clearing Inode Allocation Map
Clearing Block Allocation Map
Formatting Allocation Map for storage pool system
Completed creation of file system /dev/gpfs1.
mmcrfs: Propagating the cluster configuration data to all
  affected nodes.  This is an asynchronous process.
[root@mastr-51 cluster-test]#
```

27. Mount the GPFS.

```
[root@mastr-51 cluster-test]# mmmount all -a
Tue Oct  8 18:05:34 UTC 2019: mmmount: Mounting file systems ...
[root@mastr-51 cluster-test]#
```

28. Check and provide the required permissions to the GPFS.

```
[root@mastr-51 cluster-test]# mmlsdisk gpfs1
disk         driver   sector     failure holds    holds
storage
name         type       size       group metadata data  status
availability pool
------------ -------- ------ ----------- -------- ----- -------------
------------ ------------
nsd1         nsd         512           1 Yes      Yes   ready         up
system
nsd2         nsd         512           1 Yes      Yes   ready         up
system
[root@mastr-51 cluster-test]#

[root@mastr-51 cluster-test]# for i in 51 53 136 138  ; do ssh
10.63.150.$i "hostname; chmod 777 /gpfs1" ; done;
mastr-51.netapp.com
mastr-53.netapp.com
workr-136.netapp.com
workr-138.netapp.com
[root@mastr-51 cluster-test]#
```

29. Check the GPFS read and write by running the `dd` command.

```
[root@mastr-51 cluster-test]# dd if=/dev/zero of=/gpfs1/testfile
bs=1024M count=5
5+0 records in
5+0 records out
5368709120 bytes (5.4 GB) copied, 8.3981 s, 639 MB/s
[root@mastr-51 cluster-test]# for i in 51 53 136 138  ; do ssh
10.63.150.$i "hostname; ls -ltrh /gpfs1" ; done;
mastr-51.netapp.com
total 5.0G
-rw-r--r-- 1 root root 5.0G Oct  8 18:10 testfile
mastr-53.netapp.com
total 5.0G
-rw-r--r-- 1 root root 5.0G Oct  8 18:10 testfile
workr-136.netapp.com
total 5.0G
-rw-r--r-- 1 root root 5.0G Oct  8 18:10 testfile
workr-138.netapp.com
total 5.0G
-rw-r--r-- 1 root root 5.0G Oct  8 18:10 testfile
[root@mastr-51 cluster-test]#
```

Export GPFS into NFS

To export GPFS into NFS, complete the following steps:

1. Export the GPFS as NFS through the `/etc/exports` file.

```
[root@mastr-51 gpfs1]# cat /etc/exports
/gpfs1        *(rw,fsid=745)
[root@mastr-51 gpfs1]
```

2. Install the required NFS server packages.

```
[root@mastr-51 ~]# yum install rpcbind
Loaded plugins: priorities, product-id, search-disabled-repos,
subscription-manager
Resolving Dependencies
--> Running transaction check
---> Package rpcbind.x86_64 0:0.2.0-47.el7 will be updated
---> Package rpcbind.x86_64 0:0.2.0-48.el7 will be an update
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
================================================================================
================================================================================
======================
 Package                                                 Arch
Version                                                     Repository
Size
================================================================================
================================================================================
================================================================================
======================
Updating:
 rpcbind                                                 x86_64
0.2.0-48.el7                                                rhel-7-
server-rpms                                                 60 k

Transaction Summary
================================================================================
================================================================================
================================================================================
======================
Upgrade  1 Package
```

```
Total download size: 60 k
Is this ok [y/d/N]: y
Downloading packages:
No Presto metadata available for rhel-7-server-rpms
rpcbind-0.2.0-48.el7.x86_64.rpm
|  60 kB  00:00:00
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Updating   : rpcbind-0.2.0-48.el7.x86_64
1/2
  Cleanup    : rpcbind-0.2.0-47.el7.x86_64
2/2
  Verifying  : rpcbind-0.2.0-48.el7.x86_64
1/2
  Verifying  : rpcbind-0.2.0-47.el7.x86_64
2/2

Updated:
  rpcbind.x86_64 0:0.2.0-48.el7

Complete!
[root@mastr-51 ~]#
```

3. Start the NFS service.

```
[root@mastr-51 ~]# service nfs status
Redirecting to /bin/systemctl status nfs.service
● nfs-server.service - NFS server and services
   Loaded: loaded (/usr/lib/systemd/system/nfs-server.service; disabled;
vendor preset: disabled)
  Drop-In: /run/systemd/generator/nfs-server.service.d
           └─order-with-mounts.conf
   Active: inactive (dead)
[root@mastr-51 ~]# service rpcbind start
Redirecting to /bin/systemctl start rpcbind.service
[root@mastr-51 ~]# service nfs start
Redirecting to /bin/systemctl start nfs.service
[root@mastr-51 ~]# service nfs status
Redirecting to /bin/systemctl status nfs.service
● nfs-server.service - NFS server and services
   Loaded: loaded (/usr/lib/systemd/system/nfs-server.service; disabled;
vendor preset: disabled)
  Drop-In: /run/systemd/generator/nfs-server.service.d
           └─order-with-mounts.conf
   Active: active (exited) since Wed 2019-11-06 16:34:50 UTC; 2s ago
  Process: 24402 ExecStartPost=/bin/sh -c if systemctl -q is-active
gssproxy; then systemctl reload gssproxy ; fi (code=exited,
status=0/SUCCESS)
  Process: 24383 ExecStart=/usr/sbin/rpc.nfsd $RPCNFSDARGS (code=exited,
status=0/SUCCESS)
  Process: 24379 ExecStartPre=/usr/sbin/exportfs -r (code=exited,
status=0/SUCCESS)
 Main PID: 24383 (code=exited, status=0/SUCCESS)
   CGroup: /system.slice/nfs-server.service

Nov 06 16:34:50 mastr-51.netapp.com systemd[1]: Starting NFS server and
services...
Nov 06 16:34:50 mastr-51.netapp.com systemd[1]: Started NFS server and
services.
[root@mastr-51 ~]#
```

4. List the files in GPFS to validate the NFS client.

```
[root@mastr-51 gpfs1]# df -Th
Filesystem                                      Type      Size  Used Avail
Use% Mounted on
/dev/mapper/rhel_stlrx300s6--22--irmc-root xfs        94G   55G   39G
59% /
devtmpfs                                        devtmpfs   32G     0   32G
0% /dev
tmpfs                                           tmpfs      32G     0   32G
0% /dev/shm
tmpfs                                           tmpfs      32G  3.3G   29G
11% /run
tmpfs                                           tmpfs      32G     0   32G
0% /sys/fs/cgroup
/dev/sda7                                       xfs       9.4G  210M  9.1G
3% /boot
tmpfs                                           tmpfs     6.3G     0  6.3G
0% /run/user/10065
tmpfs                                           tmpfs     6.3G     0  6.3G
0% /run/user/10068
tmpfs                                           tmpfs     6.3G     0  6.3G
0% /run/user/10069
10.63.150.213:/nc_volume3                       nfs4      380G  8.0M  380G
1% /mnt
tmpfs                                           tmpfs     6.3G     0  6.3G
0% /run/user/0
gpfs1                                           gpfs      7.3T  9.1G  7.3T
1% /gpfs1
[root@mastr-51 gpfs1]#
[root@mastr-51 ~]# cd /gpfs1
[root@mastr-51 gpfs1]# ls
catalog  ces  gpfs-ces  ha  testfile
[root@mastr-51 gpfs1]#
[root@mastr-51 ~]# cd /gpfs1
[root@mastr-51 gpfs1]# ls
ces  gpfs-ces  ha  testfile
[root@mastr-51 gpfs1]# ls -ltrha
total 5.1G
dr-xr-xr-x   2 root root 8.0K Jan  1  1970 .snapshots
-rw-r--r--   1 root root 5.0G Oct  8 18:10 testfile
dr-xr-xr-x. 30 root root 4.0K Oct  8 18:19 ..
drwxr-xr-x   2 root root 4.0K Nov  5 20:02 gpfs-ces
drwxr-xr-x   2 root root 4.0K Nov  5 20:04 ha
drwxrwxrwx   5 root root 256K Nov  5 20:04 .
drwxr-xr-x   4 root root 4.0K Nov  5 20:35 ces
[root@mastr-51 gpfs1]#
```

Configure the NFS client

To configure the NFS client, complete the following steps:

1. Install packages in the NFS client.

```
[root@hdp2 ~]# yum install nfs-utils rpcbind
Loaded plugins: product-id, search-disabled-repos, subscription-manager
HDP-2.6-GPL-repo-4
| 2.9 kB  00:00:00
HDP-2.6-repo-4
| 2.9 kB  00:00:00
HDP-3.0-GPL-repo-2
| 2.9 kB  00:00:00
HDP-3.0-repo-2
| 2.9 kB  00:00:00
HDP-3.0-repo-3
| 2.9 kB  00:00:00
HDP-3.1-repo-1
| 2.9 kB  00:00:00
HDP-3.1-repo-51
| 2.9 kB  00:00:00
HDP-UTILS-1.1.0.22-repo-1
| 2.9 kB  00:00:00
HDP-UTILS-1.1.0.22-repo-2
| 2.9 kB  00:00:00
HDP-UTILS-1.1.0.22-repo-3
| 2.9 kB  00:00:00
HDP-UTILS-1.1.0.22-repo-4
| 2.9 kB  00:00:00
HDP-UTILS-1.1.0.22-repo-51
| 2.9 kB  00:00:00
ambari-2.7.3.0
| 2.9 kB  00:00:00
epel/x86_64/metalink
|  13 kB  00:00:00
epel
| 5.3 kB  00:00:00
mysql-connectors-community
| 2.5 kB  00:00:00
mysql-tools-community
| 2.5 kB  00:00:00
mysql56-community
| 2.5 kB  00:00:00
rhel-7-server-optional-rpms
| 3.2 kB  00:00:00
rhel-7-server-rpms
```

```
| 3.5 kB  00:00:00
(1/10): mysql-connectors-community/x86_64/primary_db
|  49 kB  00:00:00
(2/10): mysql-tools-community/x86_64/primary_db
|  66 kB  00:00:00
(3/10): epel/x86_64/group_gz
|  90 kB  00:00:00
(4/10): mysql56-community/x86_64/primary_db
| 241 kB  00:00:00
(5/10): rhel-7-server-optional-rpms/7Server/x86_64/updateinfo
| 2.5 MB  00:00:00
(6/10): rhel-7-server-rpms/7Server/x86_64/updateinfo
| 3.4 MB  00:00:00
(7/10): rhel-7-server-optional-rpms/7Server/x86_64/primary_db
| 8.3 MB  00:00:00
(8/10): rhel-7-server-rpms/7Server/x86_64/primary_db
|  62 MB  00:00:01
(9/10): epel/x86_64/primary_db
| 6.9 MB  00:00:08
(10/10): epel/x86_64/updateinfo
| 1.0 MB  00:00:13
Resolving Dependencies
--> Running transaction check
---> Package nfs-utils.x86_64 1:1.3.0-0.61.el7 will be updated
---> Package nfs-utils.x86_64 1:1.3.0-0.65.el7 will be an update
---> Package rpcbind.x86_64 0:0.2.0-47.el7 will be updated
---> Package rpcbind.x86_64 0:0.2.0-48.el7 will be an update
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
================================================
 Package                  Arch                 Version
Repository                          Size
================================================================================
================================================
Updating:
 nfs-utils                x86_64               1:1.3.0-0.65.el7
rhel-7-server-rpms                412 k
 rpcbind                  x86_64               0.2.0-48.el7
rhel-7-server-rpms                 60 k

Transaction Summary
================================================================================
================================================
```

```
Upgrade  2 Packages

Total download size: 472 k
Is this ok [y/d/N]: y
Downloading packages:
No Presto metadata available for rhel-7-server-rpms
(1/2): rpcbind-0.2.0-48.el7.x86_64.rpm
|  60 kB  00:00:00
(2/2): nfs-utils-1.3.0-0.65.el7.x86_64.rpm
| 412 kB  00:00:00
--------------------------------------------------------------------------------
--------------------------------------------
Total
1.2 MB/s | 472 kB  00:00:00
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Updating   : rpcbind-0.2.0-48.el7.x86_64
1/4
service rpcbind start

  Updating   : 1:nfs-utils-1.3.0-0.65.el7.x86_64
2/4
  Cleanup    : 1:nfs-utils-1.3.0-0.61.el7.x86_64
3/4
  Cleanup    : rpcbind-0.2.0-47.el7.x86_64
4/4
  Verifying  : 1:nfs-utils-1.3.0-0.65.el7.x86_64
1/4
  Verifying  : rpcbind-0.2.0-48.el7.x86_64
2/4
  Verifying  : rpcbind-0.2.0-47.el7.x86_64
3/4
  Verifying  : 1:nfs-utils-1.3.0-0.61.el7.x86_64
4/4

Updated:
  nfs-utils.x86_64 1:1.3.0-0.65.el7
rpcbind.x86_64 0:0.2.0-48.el7

Complete!
[root@hdp2 ~]#
```

2. Start the NFS client services.

```
[root@hdp2 ~]# service rpcbind start
Redirecting to /bin/systemctl start rpcbind.service
 [root@hdp2 ~]#
```

3. Mount the GPFS through the NFS protocol on the NFS client.

```
[root@hdp2 ~]# mkdir /gpfstest
[root@hdp2 ~]# mount 10.63.150.51:/gpfs1 /gpfstest
[root@hdp2 ~]# df -h
Filesystem                              Size  Used Avail Use% Mounted on
/dev/mapper/rhel_stlrx300s6--22-root  1.1T  113G  981G  11% /
devtmpfs                                126G     0  126G   0% /dev
tmpfs                                   126G   16K  126G   1% /dev/shm
tmpfs                                   126G  510M  126G   1% /run
tmpfs                                   126G     0  126G   0%
/sys/fs/cgroup
/dev/sdd2                               197M  191M  6.6M  97% /boot
tmpfs                                    26G     0   26G   0% /run/user/0
10.63.150.213:/nc_volume2                95G  5.4G   90G   6% /mnt
10.63.150.51:/gpfs1                     7.3T  9.1G  7.3T   1% /gpfstest
[root@hdp2 ~]#
```

4. Validate the list of GPFS files in the NFS-mounted folder.

```
[root@hdp2 ~]# cd /gpfstest/
[root@hdp2 gpfstest]# ls
ces  gpfs-ces  ha  testfile
[root@hdp2 gpfstest]# ls -l
total 5242882
drwxr-xr-x 4 root root       4096 Nov  5 15:35 ces
drwxr-xr-x 2 root root       4096 Nov  5 15:02 gpfs-ces
drwxr-xr-x 2 root root       4096 Nov  5 15:04 ha
-rw-r--r-- 1 root root 5368709120 Oct  8 14:10 testfile
[root@hdp2 gpfstest]#
```

5. Move the data from the GPFS- exported NFS to the NetApp NFS by using XCP.

```
[root@hdp2 linux]# ./xcp copy -parallel 20 10.63.150.51:/gpfs1
10.63.150.213:/nc_volume2/
XCP 1.4-17914d6; (c) 2019 NetApp, Inc.; Licensed to Karthikeyan
Nagalingam [NetApp Inc] until Tue Nov  5 12:39:36 2019

xcp: WARNING: your license will expire in less than one week! You can
renew your license at https://xcp.netapp.com
xcp: open or create catalog 'xcp': Creating new catalog in
'10.63.150.51:/gpfs1/catalog'
xcp: WARNING: No index name has been specified, creating one with name:
autoname_copy_2019-11-11_12.14.07.805223
xcp: mount '10.63.150.51:/gpfs1': WARNING: This NFS server only supports
1-second timestamp granularity. This may cause sync to fail because
changes will often be undetectable.
 34 scanned, 32 copied, 32 indexed, 1 giant, 301 MiB in (59.5 MiB/s),
784 KiB out (155 KiB/s), 6s
 34 scanned, 32 copied, 32 indexed, 1 giant, 725 MiB in (84.6 MiB/s),
1.77 MiB out (206 KiB/s), 11s
 34 scanned, 32 copied, 32 indexed, 1 giant, 1.17 GiB in (94.2 MiB/s),
2.90 MiB out (229 KiB/s), 16s
 34 scanned, 32 copied, 32 indexed, 1 giant, 1.56 GiB in (79.8 MiB/s),
3.85 MiB out (194 KiB/s), 21s
 34 scanned, 32 copied, 32 indexed, 1 giant, 1.95 GiB in (78.4 MiB/s),
4.80 MiB out (191 KiB/s), 26s
 34 scanned, 32 copied, 32 indexed, 1 giant, 2.35 GiB in (80.4 MiB/s),
5.77 MiB out (196 KiB/s), 31s
 34 scanned, 32 copied, 32 indexed, 1 giant, 2.79 GiB in (89.6 MiB/s),
6.84 MiB out (218 KiB/s), 36s
 34 scanned, 32 copied, 32 indexed, 1 giant, 3.16 GiB in (75.3 MiB/s),
7.73 MiB out (183 KiB/s), 41s
 34 scanned, 32 copied, 32 indexed, 1 giant, 3.53 GiB in (75.4 MiB/s),
8.64 MiB out (183 KiB/s), 46s
 34 scanned, 32 copied, 32 indexed, 1 giant, 4.00 GiB in (94.4 MiB/s),
9.77 MiB out (230 KiB/s), 51s
 34 scanned, 32 copied, 32 indexed, 1 giant, 4.46 GiB in (94.3 MiB/s),
10.9 MiB out (229 KiB/s), 56s
 34 scanned, 32 copied, 32 indexed, 1 giant, 4.86 GiB in (80.2 MiB/s),
11.9 MiB out (195 KiB/s), 1m1s
Sending statistics...
34 scanned, 33 copied, 34 indexed, 1 giant, 5.01 GiB in (81.8 MiB/s),
12.3 MiB out (201 KiB/s), 1m2s.
[root@hdp2 linux]#
```

6. Validate the GPFS files on the NFS client.

```
[root@hdp2 mnt]# df -Th
Filesystem                              Type      Size  Used Avail Use%
Mounted on
/dev/mapper/rhel_stlrx300s6--22-root xfs       1.1T  113G  981G  11% /
devtmpfs                                devtmpfs  126G     0  126G   0%
/dev
tmpfs                                   tmpfs     126G   16K  126G   1%
/dev/shm
tmpfs                                   tmpfs     126G  518M  126G   1%
/run
tmpfs                                   tmpfs     126G     0  126G   0%
/sys/fs/cgroup
/dev/sdd2                               xfs       197M  191M  6.6M  97%
/boot
tmpfs                                   tmpfs      26G     0   26G   0%
/run/user/0
10.63.150.213:/nc_volume2               nfs4       95G  5.4G   90G   6%
/mnt
10.63.150.51:/gpfs1                     nfs4      7.3T  9.1G  7.3T   1%
/gpfstest
[root@hdp2 mnt]#
[root@hdp2 mnt]# ls -ltrha
total 128K
dr-xr-xr-x   2 root        root              4.0K Dec 31  1969
.snapshots
drwxrwxrwx   2 root        root              4.0K Feb 14  2018 data
drwxrwxrwx   3 root        root              4.0K Feb 14  2018
wcresult
drwxrwxrwx   3 root        root              4.0K Feb 14  2018
wcresult1
drwxrwxrwx   2 root        root              4.0K Feb 14  2018
wcresult2
drwxrwxrwx   2 root        root              4.0K Feb 16  2018
wcresult3
-rw-r--r--   1 root        root              2.8K Feb 20  2018
READMEdemo
drwxrwxrwx   3 root        root              4.0K Jun 28 13:38 scantg
drwxrwxrwx   3 root        root              4.0K Jun 28 13:39
scancopyFromLocal
-rw-r--r--   1 hdfs        hadoop            1.2K Jul  3 19:28 f3
-rw-r--r--   1 hdfs        hadoop            1.2K Jul  3 19:28 README
-rw-r--r--   1 hdfs        hadoop            1.2K Jul  3 19:28 f9
-rw-r--r--   1 hdfs        hadoop            1.2K Jul  3 19:28 f6
-rw-r--r--   1 hdfs        hadoop            1.2K Jul  3 19:28 f5
-rw-r--r--   1 hdfs        hadoop            1.2K Jul  3 19:30 f4
-rw-r--r--   1 hdfs        hadoop            1.2K Jul  3 19:30 f8
```

```
-rw-r--r--   1 hdfs         hadoop               1.2K Jul  3 19:30 f2
-rw-r--r--   1 hdfs         hadoop               1.2K Jul  3 19:30 f7
drwxrwxrwx   2 root         root                 4.0K Jul  9 11:14 test
drwxrwxrwx   3 root         root                 4.0K Jul 10 16:35
warehouse
drwxr-xr-x   3       10061 tester1              4.0K Jul 15 14:40 sdd1
drwxrwxrwx   3 testeruser1 hadoopkerberosgroup 4.0K Aug 20 17:00
kermkdir
-rw-r--r--   1 testeruser1 hadoopkerberosgroup    0 Aug 21 14:20 newfile
drwxrwxrwx   2 testeruser1 hadoopkerberosgroup 4.0K Aug 22 10:13
teragen1copy_3
drwxrwxrwx   2 testeruser1 hadoopkerberosgroup 4.0K Aug 22 10:33
teragen2copy_1
-rw-rwxr--   1 root         hdfs                 1.2K Sep 19 16:38 R1
drwx------   3 root         root                 4.0K Sep 20 17:28 user
-rw-r--r--   1 root         root                 5.0G Oct  8 14:10
testfile
drwxr-xr-x   2 root         root                 4.0K Nov  5 15:02 gpfs-
ces
drwxr-xr-x   2 root         root                 4.0K Nov  5 15:04 ha
drwxr-xr-x   4 root         root                 4.0K Nov  5 15:35 ces
dr-xr-xr-x. 26 root         root                 4.0K Nov  6 11:40 ..
drwxrwxrwx  21 root         root                 4.0K Nov 11 12:14 .
drwxrwxrwx   7 nobody       nobody               4.0K Nov 11 12:14 catalog
[root@hdp2 mnt]#
```

Next: MapR-FS to ONTAP NFS.

MapR-FS to ONTAP NFS

Previous: GPFS to NFS - Detailed steps.

This section provides the detailed steps needed to move MapR-FS data into ONTAP NFS by using NetApp XCP.

1. Provision three LUNs for each MapR node and give the LUNs ownership of all MapR nodes.
2. During installation, choose newly added LUNs for MapR cluster disks that are used for MapR-FS.
3. Install a MapR cluster according to the MapR 6.1 documentation.
4. Check the basic Hadoop operations using MapReduce commands such as `hadoop jar xxx`.
5. Keep customer data in MapR-FS. For example, we generated approximately a terabyte of sample data in MapR-FS by using Teragen.
6. Configure MapR-FS as NFS export.
    a. Disable the nlockmgr service on all MapR nodes.

```
root@workr-138: ~$ rpcinfo -p
   program vers proto   port  service
    100000    4   tcp    111  portmapper
    100000    3   tcp    111  portmapper
    100000    2   tcp    111  portmapper
    100000    4   udp    111  portmapper
    100000    3   udp    111  portmapper
    100000    2   udp    111  portmapper
    100003    4   tcp   2049  nfs
    100227    3   tcp   2049  nfs_acl
    100003    4   udp   2049  nfs
    100227    3   udp   2049  nfs_acl
    100021    3   udp  55270  nlockmgr
    100021    4   udp  55270  nlockmgr
    100021    3   tcp  35025  nlockmgr
    100021    4   tcp  35025  nlockmgr
    100003    3   tcp   2049  nfs
    100005    3   tcp   2049  mountd
    100005    1   tcp   2049  mountd
    100005    3   udp   2049  mountd
    100005    1   udp   2049  mountd
root@workr-138: ~$

root@workr-138: ~$ rpcinfo -d 100021 3
root@workr-138: ~$ rpcinfo -d 100021 4
```

b. Export specific folders from MapR-FS on all MapR nodes in the `/opt/mapr/conf/exports` file. Do not export the parent folder with different permissions when you export sub folders.

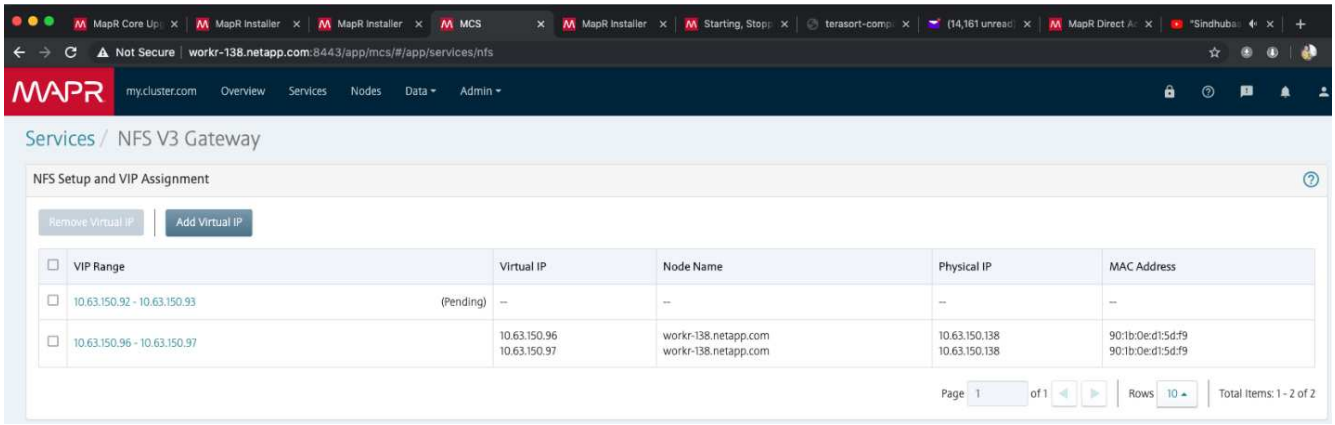
```
[mapr@workr-138 ~]$ cat /opt/mapr/conf/exports
# Sample Exports file
# for /mapr exports
# <Path> <exports_control>
#access_control -> order is specific to default
# list the hosts before specifying a default for all
#  a.b.c.d,1.2.3.4(ro) d.e.f.g(ro) (rw)
#  enforces ro for a.b.c.d & 1.2.3.4 and everybody else is rw
# special path to export clusters in mapr-clusters.conf. To disable
exporting,
# comment it out. to restrict access use the exports_control
#
#/mapr (rw)
#karthik
/mapr/my.cluster.com/tmp/testnfs /maprnfs3 (rw)
#to export only certain clusters, comment out the /mapr & uncomment.
#/mapr/clustername (rw)
#to export /mapr only to certain hosts (using exports_control)
#/mapr a.b.c.d(rw),e.f.g.h(ro)
# export /mapr/cluster1 rw to a.b.c.d & ro to e.f.g.h (denied for
others)
#/mapr/cluster1 a.b.c.d(rw),e.f.g.h(ro)
# export /mapr/cluster2 only to e.f.g.h (denied for others)
#/mapr/cluster2 e.f.g.h(rw)
# export /mapr/cluster3 rw to e.f.g.h & ro to others
#/mapr/cluster2 e.f.g.h(rw) (ro)
#to export a certain cluster, volume or a subdirectory as an alias,
#comment out  /mapr & uncomment
#/mapr/clustername         /alias1 (rw)
#/mapr/clustername/vol     /alias2 (rw)
#/mapr/clustername/vol/dir /alias3 (rw)
#only the alias will be visible/exposed to the nfs client not the
mapr path, host options as before
[mapr@workr-138 ~]$
```

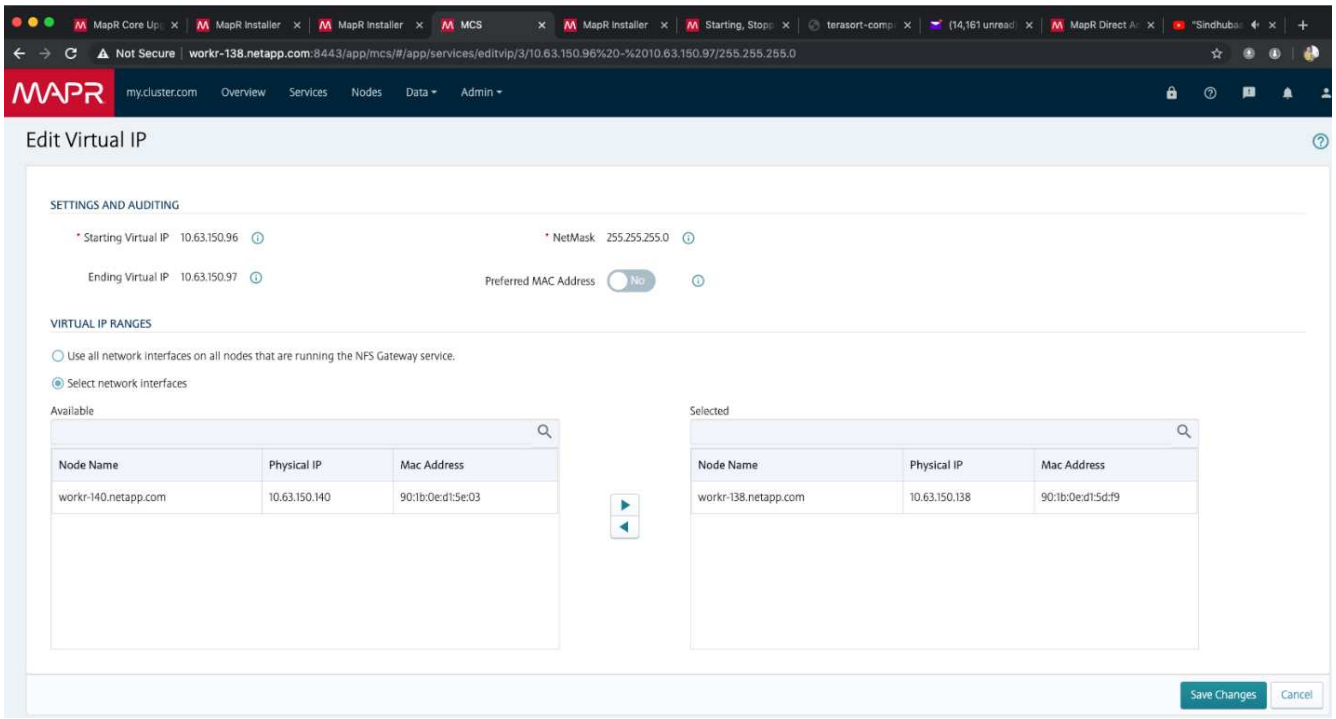
7. Refresh the MapR-FS NFS service.

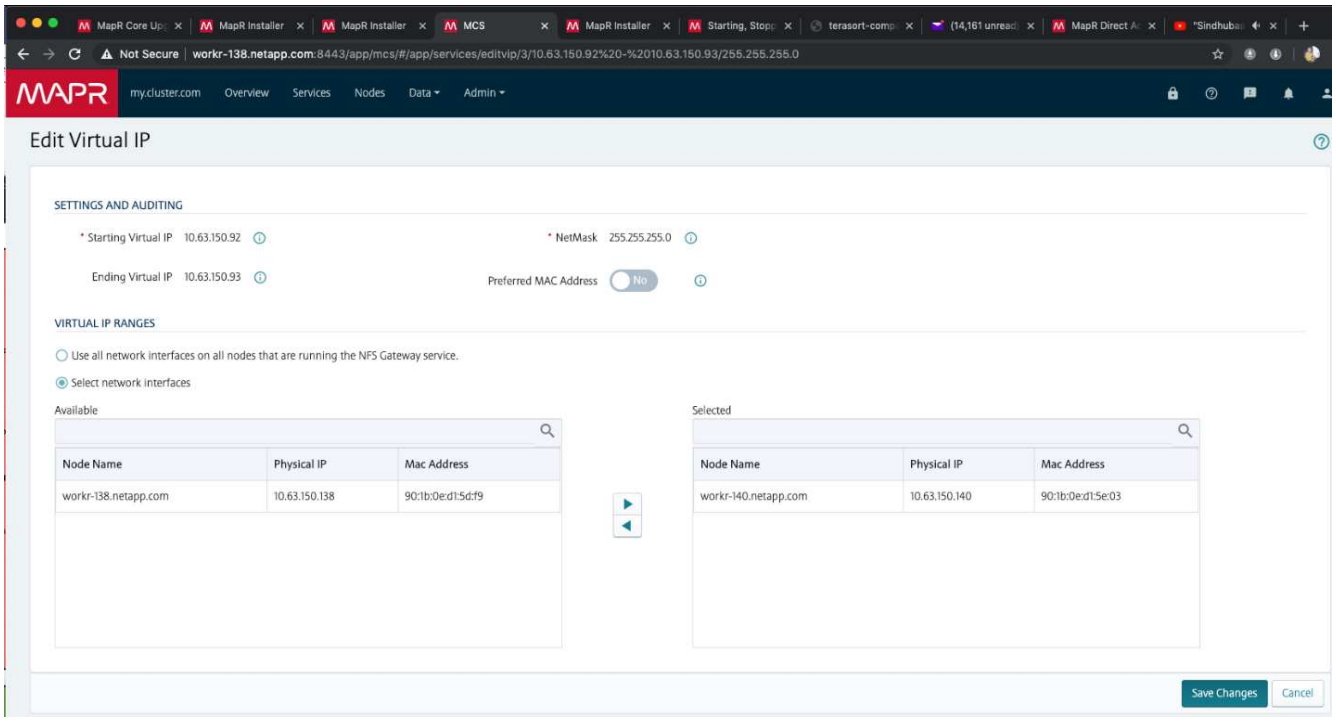
```
root@workr-138: tmp$ maprcli nfsmgmt refreshexports
ERROR (22) -  You do not have a ticket to communicate with
127.0.0.1:9998. Retry after obtaining a new ticket using maprlogin
root@workr-138: tmp$ su - mapr
[mapr@workr-138 ~]$ maprlogin password -cluster my.cluster.com
[Password for user 'mapr' at cluster 'my.cluster.com': ]
MapR credentials of user 'mapr' for cluster 'my.cluster.com' are written
to '/tmp/maprticket_5000'
[mapr@workr-138 ~]$ maprcli nfsmgmt refreshexports
```

8. Assign a virtual IP range to a specific server or a set of servers in the MapR cluster. Then the MapR cluster assigns an IP to a specific server for NFS data access. The IPs enable high availability, which means that, if a server or network with a particular IP experiences failure, the next IP from the range of IPs can be used for NFS access.

> If you would like to provide NFS access from all MapR nodes, then you can assign a set of virtual IPs to each server, and you can use the resources from each MapR node for NFS data access.







9. Check the virtual IPs assigned on each MapR node and use them for NFS data access.

```
root@workr-138: ~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
```

```
       valid_lft forever preferred_lft forever
2: ens3f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc mq state UP
group default qlen 1000
    link/ether 90:1b:0e:d1:5d:f9 brd ff:ff:ff:ff:ff:ff
    inet 10.63.150.138/24 brd 10.63.150.255 scope global noprefixroute
ens3f0
       valid_lft forever preferred_lft forever
    inet 10.63.150.96/24 scope global secondary ens3f0:~m0
       valid_lft forever preferred_lft forever
    inet 10.63.150.97/24 scope global secondary ens3f0:~m1
       valid_lft forever preferred_lft forever
    inet6 fe80::921b:eff:fed1:5df9/64 scope link
       valid_lft forever preferred_lft forever
3: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP
group default qlen 1000
    link/ether 90:1b:0e:d1:af:b4 brd ff:ff:ff:ff:ff:ff
4: ens3f1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP
group default qlen 1000
    link/ether 90:1b:0e:d1:5d:fa brd ff:ff:ff:ff:ff:ff
5: eno2: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state
DOWN group default qlen 1000
    link/ether 90:1b:0e:d1:af:b5 brd ff:ff:ff:ff:ff:ff
[root@workr-138: ~$
[root@workr-140 ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens3f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc mq state UP
group default qlen 1000
    link/ether 90:1b:0e:d1:5e:03 brd ff:ff:ff:ff:ff:ff
    inet 10.63.150.140/24 brd 10.63.150.255 scope global noprefixroute
ens3f0
       valid_lft forever preferred_lft forever
    inet 10.63.150.92/24 scope global secondary ens3f0:~m0
       valid_lft forever preferred_lft forever
    inet6 fe80::921b:eff:fed1:5e03/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP
group default qlen 1000
    link/ether 90:1b:0e:d1:af:9a brd ff:ff:ff:ff:ff:ff
4: ens3f1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP
group default qlen 1000
```

```
    link/ether 90:1b:0e:d1:5e:04 brd ff:ff:ff:ff:ff:ff
5: eno2: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state
DOWN group default qlen 1000
    link/ether 90:1b:0e:d1:af:9b brd ff:ff:ff:ff:ff:ff
[root@workr-140 ~]#
```

10. Mount the NFS- exported MapR-FS using the assigned virtual IP for checking the NFS operation. However, this step is not required for data transfer using NetApp XCP.

```
root@workr-138: tmp$ mount -v -t nfs 10.63.150.92:/maprnfs3
/tmp/testmount/
mount.nfs: timeout set for Thu Dec  5 15:31:32 2019
mount.nfs: trying text-based options
'vers=4.1,addr=10.63.150.92,clientaddr=10.63.150.138'
mount.nfs: mount(2): Protocol not supported
mount.nfs: trying text-based options
'vers=4.0,addr=10.63.150.92,clientaddr=10.63.150.138'
mount.nfs: mount(2): Protocol not supported
mount.nfs: trying text-based options 'addr=10.63.150.92'
mount.nfs: prog 100003, trying vers=3, prot=6
mount.nfs: trying 10.63.150.92 prog 100003 vers 3 prot TCP port 2049
mount.nfs: prog 100005, trying vers=3, prot=17
mount.nfs: trying 10.63.150.92 prog 100005 vers 3 prot UDP port 2049
mount.nfs: portmap query retrying: RPC: Timed out
mount.nfs: prog 100005, trying vers=3, prot=6
mount.nfs: trying 10.63.150.92 prog 100005 vers 3 prot TCP port 2049
root@workr-138: tmp$ df -h
Filesystem              Size  Used Avail Use% Mounted on
/dev/sda7                84G   48G   37G  57% /
devtmpfs                126G     0  126G   0% /dev
tmpfs                   126G     0  126G   0% /dev/shm
tmpfs                   126G   19M  126G   1% /run
tmpfs                   126G     0  126G   0% /sys/fs/cgroup
/dev/sdd1               3.7T  201G  3.5T   6% /mnt/sdd1
/dev/sda6               946M  220M  726M  24% /boot
tmpfs                    26G     0   26G   0% /run/user/5000
gpfs1                   7.3T  9.1G  7.3T   1% /gpfs1
tmpfs                    26G     0   26G   0% /run/user/0
localhost:/mapr         100G     0  100G   0% /mapr
10.63.150.92:/maprnfs3   53T  8.4G   53T   1% /tmp/testmount
root@workr-138: tmp$
```

11. Configure NetApp XCP to transfer data from the MapR-FS NFS gateway to ONTAP NFS.
    a. Configure the catalog location for XCP.

```
[root@hdp2 linux]# cat /opt/NetApp/xFiles/xcp/xcp.ini
# Sample xcp config
[xcp]
#catalog =  10.63.150.51:/gpfs1
catalog =  10.63.150.213:/nc_volume1
```

b. Copy the license file to `/opt/NetApp/xFiles/xcp/`.

```
root@workr-138: src$ cd /opt/NetApp/xFiles/xcp/
root@workr-138: xcp$ ls -ltrha
total 252K
drwxr-xr-x 3 root   root     16 Apr  4  2019 ..
-rw-r--r-- 1 root   root    105 Dec  5 19:04 xcp.ini
drwxr-xr-x 2 root   root     59 Dec  5 19:04 .
-rw-r--r-- 1 faiz89 faiz89  336 Dec  6 21:12 license
-rw-r--r-- 1 root   root    192 Dec  6 21:13 host
-rw-r--r-- 1 root   root   236K Dec 17 14:12 xcp.log
root@workr-138: xcp$
```

c. Activate XCP using the `xcp activate` command.

d. Check the source for NFS export.

```
[root@hdp2 linux]# ./xcp show 10.63.150.92
XCP 1.4-17914d6; (c) 2019 NetApp, Inc.; Licensed to Karthikeyan
Nagalingam [NetApp Inc] until Wed Feb  5 11:07:27 2020
getting pmap dump from 10.63.150.92 port 111...
getting export list from 10.63.150.92...
sending 1 mount and 4 nfs requests to 10.63.150.92...
== RPC Services ==
'10.63.150.92': TCP rpc services: MNT v1/3, NFS v3/4, NFSACL v3, NLM
v1/3/4, PMAP v2/3/4, STATUS v1
'10.63.150.92': UDP rpc services: MNT v1/3, NFS v4, NFSACL v3, NLM
v1/3/4, PMAP v2/3/4, STATUS v1
== NFS Exports ==
 Mounts  Errors  Server
      1       0  10.63.150.92
     Space    Files      Space    Files
      Free     Free       Used     Used Export
  52.3 TiB    53.7B   8.36 GiB    53.7B 10.63.150.92:/maprnfs3
== Attributes of NFS Exports ==
drwxr-xr-x --- root root 2 2 10m51s 10.63.150.92:/maprnfs3
1.77 KiB in (8.68 KiB/s), 3.16 KiB out (15.5 KiB/s), 0s.
[root@hdp2 linux]#
```

e. Transfer the data using XCP from multiple MapR nodes from multiple source IPs and multiple destination IPs (ONTAP LIFs).

```
root@workr-138: linux$ ./xcp_yatin copy --parallel 20
10.63.150.96,10.63.150.97:/maprnfs3/tg4
10.63.150.85,10.63.150.86:/datapipeline_dataset/tg4_dest
XCP 1.6-dev; (c) 2019 NetApp, Inc.; Licensed to Karthikeyan
Nagalingam [NetApp Inc] until Wed Feb  5 11:07:27 2020
xcp: WARNING: No index name has been specified, creating one with
name: autoname_copy_2019-12-06_21.14.38.652652
xcp: mount '10.63.150.96,10.63.150.97:/maprnfs3/tg4': WARNING: This
NFS server only supports 1-second timestamp granularity. This may
cause sync to fail because changes will often be undetectable.
 130 scanned, 128 giants, 3.59 GiB in (723 MiB/s), 3.60 GiB out (724
MiB/s), 5s
 130 scanned, 128 giants, 8.01 GiB in (889 MiB/s), 8.02 GiB out (890
MiB/s), 11s
 130 scanned, 128 giants, 12.6 GiB in (933 MiB/s), 12.6 GiB out (934
MiB/s), 16s
 130 scanned, 128 giants, 16.7 GiB in (830 MiB/s), 16.7 GiB out (831
MiB/s), 21s
 130 scanned, 128 giants, 21.1 GiB in (907 MiB/s), 21.1 GiB out (908
MiB/s), 26s
```

```
 130 scanned, 128 giants, 25.5 GiB in (893 MiB/s), 25.5 GiB out (894
MiB/s), 31s
 130 scanned, 128 giants, 29.6 GiB in (842 MiB/s), 29.6 GiB out (843
MiB/s), 36s
….
[root@workr-140 linux]# ./xcp_yatin copy  --parallel 20
10.63.150.92:/maprnfs3/tg4_2
10.63.150.85,10.63.150.86:/datapipeline_dataset/tg4_2_dest
XCP 1.6-dev; (c) 2019 NetApp, Inc.; Licensed to Karthikeyan
Nagalingam [NetApp Inc] until Wed Feb  5 11:07:27 2020
xcp: WARNING: No index name has been specified, creating one with
name: autoname_copy_2019-12-06_21.14.24.637773
xcp: mount '10.63.150.92:/maprnfs3/tg4_2': WARNING: This NFS server
only supports 1-second timestamp granularity. This may cause sync to
fail because changes will often be undetectable.
 130 scanned, 128 giants, 4.39 GiB in (896 MiB/s), 4.39 GiB out (897
MiB/s), 5s
 130 scanned, 128 giants, 9.94 GiB in (1.10 GiB/s), 9.96 GiB out
(1.10 GiB/s), 10s
 130 scanned, 128 giants, 15.4 GiB in (1.09 GiB/s), 15.4 GiB out
(1.09 GiB/s), 15s
 130 scanned, 128 giants, 20.1 GiB in (953 MiB/s), 20.1 GiB out (954
MiB/s), 20s
 130 scanned, 128 giants, 24.6 GiB in (928 MiB/s), 24.7 GiB out (929
MiB/s), 25s
 130 scanned, 128 giants, 29.0 GiB in (877 MiB/s), 29.0 GiB out (878
MiB/s), 31s
 130 scanned, 128 giants, 33.2 GiB in (852 MiB/s), 33.2 GiB out (853
MiB/s), 36s
 130 scanned, 128 giants, 37.8 GiB in (941 MiB/s), 37.8 GiB out (942
MiB/s), 41s
 130 scanned, 128 giants, 42.0 GiB in (860 MiB/s), 42.0 GiB out (861
MiB/s), 46s
 130 scanned, 128 giants, 46.1 GiB in (852 MiB/s), 46.2 GiB out (853
MiB/s), 51s
 130 scanned, 128 giants, 50.1 GiB in (816 MiB/s), 50.2 GiB out (817
MiB/s), 56s
 130 scanned, 128 giants, 54.1 GiB in (819 MiB/s), 54.2 GiB out (820
MiB/s), 1m1s
 130 scanned, 128 giants, 58.5 GiB in (897 MiB/s), 58.6 GiB out (898
MiB/s), 1m6s
 130 scanned, 128 giants, 62.9 GiB in (900 MiB/s), 63.0 GiB out (901
MiB/s), 1m11s
 130 scanned, 128 giants, 67.2 GiB in (876 MiB/s), 67.2 GiB out (877
MiB/s), 1m16s
```

f. Check the load distribution on the storage controller.

```
Hadoop-AFF8080::*> statistics show-periodic -interval 2 -iterations 0
-summary true -object nic_common -counter rx_bytes|tx_bytes -node
Hadoop-AFF8080-01 -instance e3b
Hadoop-AFF8080: nic_common.e3b: 12/6/2019 15:55:04
 rx_bytes tx_bytes
 -------- --------
    879MB   4.67MB
    856MB   4.46MB
    973MB   5.66MB
    986MB   5.88MB
    945MB   5.30MB
    920MB   4.92MB
    894MB   4.76MB
    902MB   4.79MB
    886MB   4.68MB
    892MB   4.78MB
    908MB   4.96MB
    905MB   4.85MB
    899MB   4.83MB
Hadoop-AFF8080::*> statistics show-periodic -interval 2 -iterations 0
-summary true -object nic_common -counter rx_bytes|tx_bytes -node
Hadoop-AFF8080-01 -instance e9b
Hadoop-AFF8080: nic_common.e9b: 12/6/2019 15:55:07
 rx_bytes tx_bytes
 -------- --------
    950MB   4.93MB
    991MB   5.84MB
    959MB   5.63MB
    914MB   5.06MB
    903MB   4.81MB
    899MB   4.73MB
    892MB   4.71MB
    890MB   4.72MB
    905MB   4.86MB
    902MB   4.90MB
```

Next: Where to find additional information.

Where to find additional information

Previous: MapR-FS to ONTAP NFS.

To learn more about the information that is described in this document, review the following documents and/or websites:

- NetApp In-Place Analytics Module Best Practices

  https://www.netapp.com/us/media/tr-4382.pdf

- NetApp FlexGroup Volume Best Practices and Implementation Guide

  https://www.netapp.com/us/media/tr-4571.pdf

- NetApp Product Documentation

  https://www.netapp.com/us/documentation/index.aspx

Version history

Version	Date	Document version history
Version 3.0	January 2022	Directly move data from HDFS and MapR-FS to NFS by using NetApp XCP.
Version 2.0	January 2020	XCP included as the default data mover. Added MapR-FS to NFS and GPFS to NFS data transfer.
Version 1.0	November 2018	Initial release.

Best practices for Confluent Kafka

TR-4912: Best practice guidelines for Confluent Kafka tiered storage with NetApp

Karthikeyan Nagalingam, Joseph Kandatilparambil, NetApp
Rankesh Kumar, Confluent

Apache Kafka is a community-distributed event-streaming platform capable of handling trillions of events a day. Initially conceived as a messaging queue, Kafka is based on an abstraction of a distributed commit log. Since it was created and open-sourced by LinkedIn in 2011, Kafka has evolved from a messages queue to a full-fledged event-streaming platform. Confluent delivers the distribution of Apache Kafka with the Confluent Platform. The Confluent Platform supplements Kafka with additional community and commercial features designed to enhance the streaming experience of both operators and developers in production at a massive scale.

This document describes the best-practice guidelines for using Confluent Tiered Storage on a NetApp's Object storage offering by providing the following content:

- Confluent verification with NetApp Object storage – NetApp StorageGRID
- Tiered storage performance tests
- Best-practice guidelines for Confluent on NetApp storage systems

Why Confluent Tiered Storage?

Confluent has become the default real-time streaming platform for many applications, especially for big data, analytics, and streaming workloads. Tiered Storage enables users to separate compute from storage in the

Confluent platform. It makes storing data more cost effective, enables you to store virtually infinite amounts of data and scale workloads up (or down) on-demand, and makes administrative tasks like data and tenant rebalancing easier. S3 compatible storage systems can take advantage of all these capabilities to democratize data with all events in one place, eliminating the need for complex data engineering. For more info on why you should use tiered storage for Kafka, check this article by Confluent.

Why NetApp StorageGRID for tiered storage?

StorageGRID is an industry-leading object storage platform by NetApp. StorageGRID is a software-defined, object-based storage solution that supports industry-standard object APIs, including the Amazon Simple Storage Service (S3) API. StorageGRID stores and manages unstructured data at scale to provide secure, durable object storage. Content is placed in the right location, at the right time, and on the right storage tier, optimizing workflows and reducing costs for globally distributed rich media.

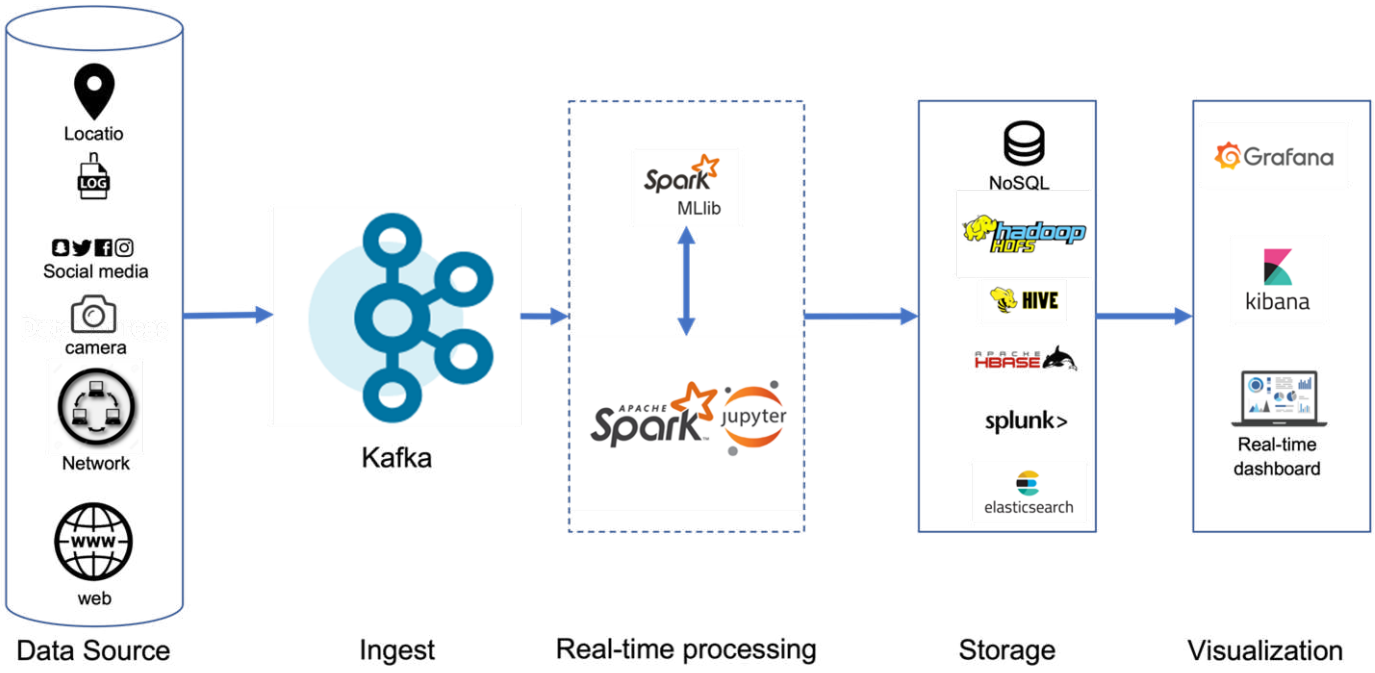
The greatest differentiator for StorageGRID is its Information Lifecycle Management (ILM) policy engine that enables policy-driven data lifecycle management. The policy engine can use metadata to manage how data is stored across its lifetime to initially optimize for performance and automatically optimize for cost and durability as data ages.

Enabling Confluent Tiered Storage

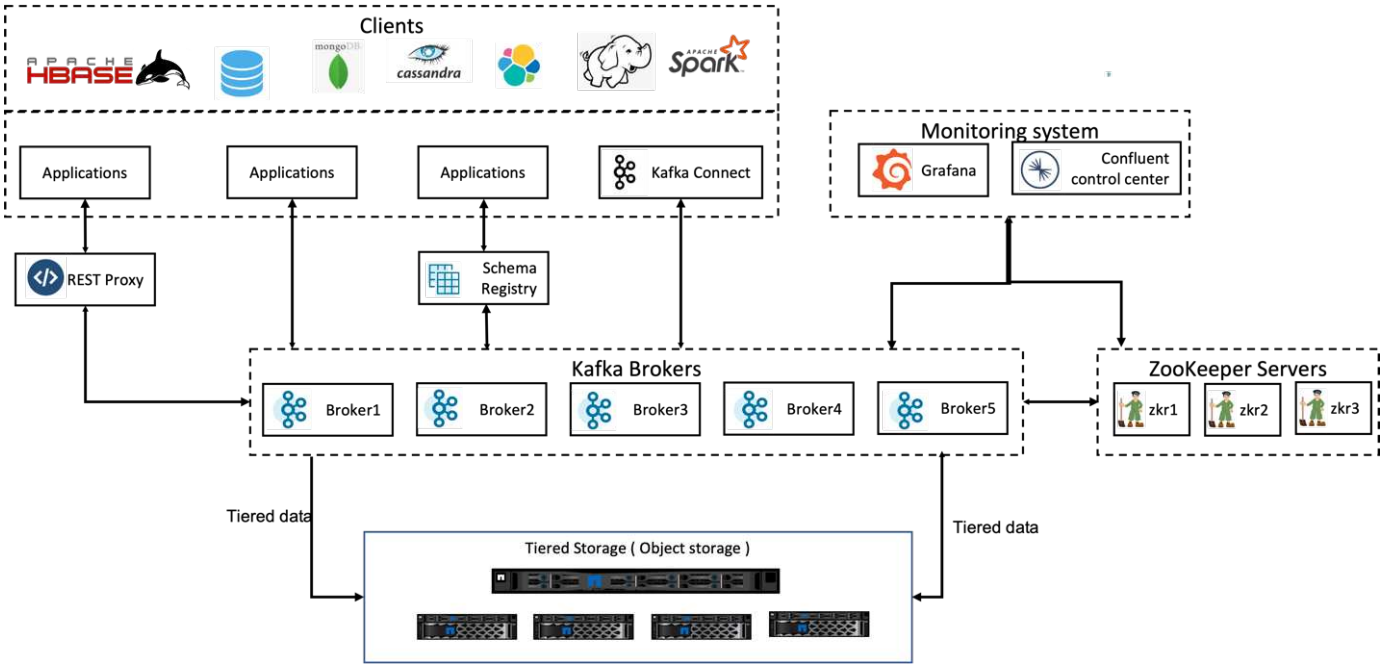
The basic idea of tiered storage is to separate the tasks of data storage from data processing. With this separation, it becomes much easier for the data storage tier and the data processing tier to scale independently.

A tiered storage solution for Confluent must contend with two factors. First, it must work around or avoid common object store consistency and availability properties, such as inconsistencies in LIST operations and occasional object unavailability. Secondly, it must correctly handle the interaction between tiered storage and Kafka's replication and fault tolerance model, including the possibility of zombie leaders continuing to tier offset ranges. NetApp Object storage provides both the consistent object availability and HA model make the tired storage available to tier offset ranges. NetApp object storage provides consistent object availability and an HA model to make the tired storage available to tier offset ranges.

With tiered storage, you can use high-performance platforms for low-latency reads and writes near the tail of your streaming data, and you can also use cheaper, scalable object stores like NetApp StorageGRID for high-throughput historical reads. We also have technical solution for Spark with netapp storage controller and details are here. The following figure shows how Kafka fits into a real-time analytics pipeline.



The following figure depicts how NetApp StorageGRID fits in as Confluent Kafka's object storage tier.



Next: Solution architecture details.

Solution architecture details

Previous: Introduction.

This section covers the hardware and software used for Confluent verification. This information is applicable to Confluent Platform deployment with NetApp storage. The following table covers the tested solution architecture and base components.

Solution components	Details
Confluent Kafka version 6.2	• Three zookeepers • Five broker servers • Five tools servers • One Grafana • One control center
Linux (ubuntu 18.04)	All servers
NetApp StorageGRID for tiered storage	• StorageGRID software • 1 x SG1000 (load balancer) • 4 x SGF6024 • 4 x 24 x 800 SSDs • S3 protocol • 4 x 100GbE (network connectivity between broker and StorageGRID instances)
15 Fujitsu PRIMERGY RX2540 servers	Each equipped with: * 2 CPUs, 16 physical cores total * Intel Xeon * 256GB physical memory * 100GbE dual port

Next: Technology overview.

Technology overview

Previous: Solution architecture details.

This section describes the technology used in this solution.

NetApp StorageGRID

NetApp StorageGRID is a high-performance, cost-effective object storage platform. By using tiered storage, most of the data on Confluent Kafka, which is stored in local storage or the SAN storage of the broker, is offloaded to the remote object store. This configuration results in significant operational improvements by reducing the time and cost to rebalance, expand, or shrink clusters or replace a failed broker. Object storage plays an important role in managing data that resides on the object store tier, which is why picking the right object storage is important.

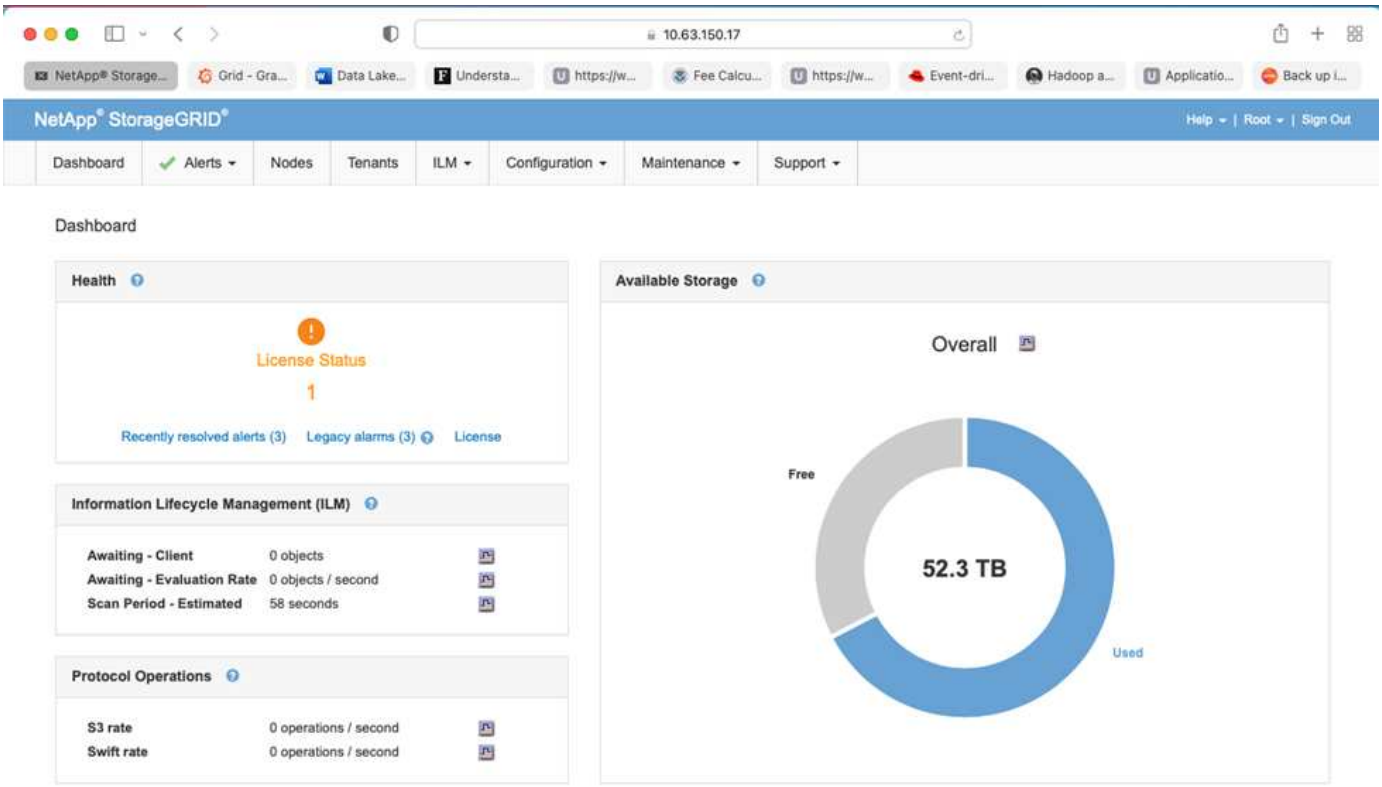
StorageGRID offers intelligent, policy-driven global data management using a distributed, node-based grid architecture. It simplifies the management of petabytes of unstructured data and billions of objects through its ubiquitous global object namespace combined with sophisticated data management features. Single-call object access extends across sites and simplifies high availability architectures while ensuring continual object access, regardless of site or infrastructure outages.

Multitenancy allows multiple unstructured cloud and enterprise data applications to be securely serviced within the same grid, increasing the ROI and use cases for NetApp StorageGRID. You can create multiple service levels with metadata-driven object lifecycle policies, optimizing durability, protection, performance, and locality

across multiple geographies. Users can adjust data management policies and monitor and apply traffic limits to realign with the data landscape nondisruptively as their requirements change in ever-changing IT environments.

Simple management with Grid Manager

The StorageGRID Grid Manager is a browser-based graphical interface that allows you to configure, manage, and monitor your StorageGRID system across globally distributed locations in a single pane of glass.



You can perform the following tasks with the StorageGRID Grid Manager interface:

- Manage globally distributed, petabyte-scale repositories of objects such as images, video, and records.
- Monitor grid nodes and services to ensure object availability.
- Manage the placement of object data over time using information lifecycle management (ILM) rules. These rules govern what happens to an object's data after it is ingested, how it is protected from loss, where object data is stored, and for how long.
- Monitor transactions, performance, and operations within the system.

Information Lifecycle Management policies

StorageGRID has flexible data management policies that include keeping replica copies of your objects and using EC (erasure coding) schemes like 2+1 and 4+2 (among others) to store your objects, depending on specific performance and data protection requirements. As workloads and requirements change over time, it's common that ILM policies must change over time as well. Modifying ILM policies is a core feature, allowing StorageGRID customers to adapt to their ever-changing environment quickly and easily. Please check the ILM policy and ILM rules setup in StorageGRID.

Performance

StorageGRID scales performance by adding more storage nodes, which can be VMs, bare metal, or purpose-built appliances like the SG5712, SG5760, SG6060, or SGF6024. In our tests, we exceeded the Apache Kafka key performance requirements with a minimum-sized, three-node grid using the SGF6024 appliance. As customers scale their Kafka cluster with additional brokers, they can add more storage nodes to increase performance and capacity.

Load balancer and endpoint configuration

Admin nodes in StorageGRID provide the Grid Manager UI (user interface) and REST API endpoint to view, configure, and manage your StorageGRID system, as well as audit logs to track system activity. To provide a highly available S3 endpoint for Confluent Kafka tiered storage, we implemented the StorageGRID load balancer, which runs as a service on admin nodes and gateway nodes. In addition, the load balancer also manages local traffic and talks to the GSLB (Global Server Load Balancing) to help with disaster recovery.

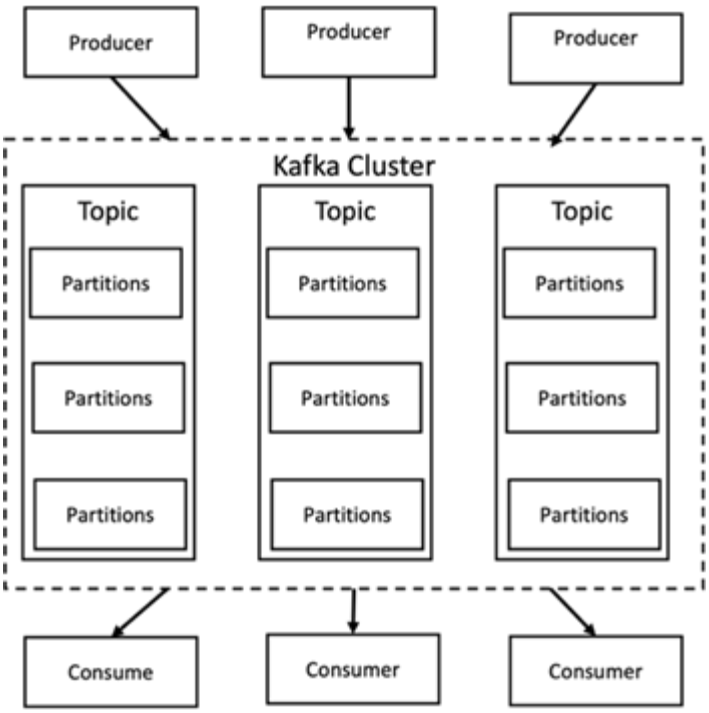
To further enhance endpoint configuration, StorageGRID provides traffic classification policies built into the admin node, lets you monitor your workload traffic, and applies various quality-of-service (QoS) limits to your workloads. Traffic classification policies are applied to endpoints on the StorageGRID Load Balancer service for gateway nodes and admin nodes. These policies can assist with traffic shaping and monitoring.

Traffic classification in StorageGRID

StorageGRID has built-in QoS functionality. Traffic classification policies can help monitor different types of S3 traffic coming from a client application. You can then create and apply policies to put limits on this traffic based on in/out bandwidth, the number of read/write concurrent requests, or the read/write request rate.

Apache Kafka

Apache Kafka is a framework implementation of a software bus using stream processing written in Java and Scala. It’s aimed to provide a unified, high-throughput, low-latency platform for handling real-time data feeds. Kafka can connect to an external system for data export and import through Kafka Connect and provides Kafka streams, a Java stream processing library. Kafka uses a binary, TCP-based protocol that is optimized for efficiency and relies on a "message set" abstraction that naturally groups messages together to reduce the overhead of the network roundtrip. This enables larger sequential disk operations, larger network packets, and contiguous memory blocks, thereby enabling Kafka to turn a bursty stream of random message writes into linear writes. The following figure depicts the basic data flow of Apache Kafka.



Kafka stores key-value messages that come from an arbitrary number of processes called producers. The data can be partitioned into different partitions within different topics. Within a partition, messages are strictly ordered by their offsets (the position of a message within a partition) and indexed and stored together with a timestamp. Other processes called consumers can read messages from partitions. For stream processing, Kafka offers the Streams API that allows writing Java applications that consume data from Kafka and write results back to Kafka. Apache Kafka also works with external stream processing systems such as Apache Apex, Apache Flink, Apache Spark, Apache Storm, and Apache NiFi.

Kafka runs on a cluster of one or more servers (called brokers), and the partitions of all topics are distributed across the cluster nodes. Additionally, partitions are replicated to multiple brokers. This architecture allows Kafka to deliver massive streams of messages in a fault-tolerant fashion and has allowed it to replace some of the conventional messaging systems like Java Message Service (JMS), Advanced Message Queuing Protocol (AMQP), and so on. Since the 0.11.0.0 release, Kafka offers transactional writes, which provide exactly once stream processing using the Streams API.

Kafka supports two types of topics: regular and compacted. Regular topics can be configured with a retention time or a space bound. If there are records that are older than the specified retention time or if the space bound is exceeded for a partition, Kafka is allowed to delete old data to free storage space. By default, topics are configured with a retention time of 7 days, but it’s also possible to store data indefinitely. For compacted topics, records don’t expire based on time or space bounds. Instead, Kafka treats later messages as updates to older message with the same key and guarantees never to delete the latest message per key. Users can delete messages entirely by writing a so-called tombstone message with the null value for a specific key.

There are five major APIs in Kafka:

- **Producer API.** Permits an application to publish streams of records.
- **Consumer API.** Permits an application to subscribe to topics and processes streams of records.
- **Connector API.** Executes the reusable producer and consumer APIs that can link the topics to the existing applications.
- **Streams API.** This API converts the input streams to output and produces the result.
- **Admin API.** Used to manage Kafka topics, brokers and other Kafka objects.

The consumer and producer APIs build on top of the Kafka messaging protocol and offer a reference implementation for Kafka consumer and producer clients in Java. The underlying messaging protocol is a binary protocol that developers can use to write their own consumer or producer clients in any programming language. This unlocks Kafka from the Java Virtual Machine (JVM) ecosystem. A list of available non-Java clients is maintained in the Apache Kafka wiki.

Apache Kafka use cases

Apache Kafka is most popular for messaging, website activity tracking, metrics, log aggregation, stream processing, event sourcing, and commit logging.

- Kafka has improved throughput, built-in partitioning, replication, and fault-tolerance, which makes it a good solution for large-scale message-processing applications.
- Kafka can rebuild a user's activities (page views, searches) in a tracking pipeline as a set of real-time publish-subscribe feeds.
- Kafka is often used for operational monitoring data. This involves aggregating statistics from distributed applications to produce centralized feeds of operational data.
- Many people use Kafka as a replacement for a log aggregation solution. Log aggregation typically collects physical log files off of servers and puts them in a central place (for example, a file server or HDFS) for processing. Kafka abstracts files details and provides a cleaner abstraction of log or event data as a stream of messages. This allows for lower-latency processing and easier support for multiple data sources and distributed data consumption.
- Many users of Kafka process data in processing pipelines consisting of multiple stages, in which raw input data is consumed from Kafka topics and then aggregated, enriched, or otherwise transformed into new topics for further consumption or follow-up processing. For example, a processing pipeline for recommending news articles might crawl article content from RSS feeds and publish it to an "articles" topic. Further processing might normalize or deduplicate this content and publish the cleansed article content to a new topic, and a final processing stage might attempt to recommend this content to users. Such processing pipelines create graphs of real-time data flows based on the individual topics.
- Event souring is a style of application design for which state changes are logged as a time-ordered sequence of records. Kafka's support for very large stored log data makes it an excellent backend for an application built in this style.
- Kafka can serve as a kind of external commit-log for a distributed system. The log helps replicate data between nodes and acts as a re-syncing mechanism for failed nodes to restore their data. The log compaction feature in Kafka helps support this use case.

Confluent

Confluent Platform is an enterprise-ready platform that completes Kafka with advanced capabilities designed to help accelerate application development and connectivity, enable transformations through stream processing, simplify enterprise operations at scale, and meet stringent architectural requirements. Built by the original creators of Apache Kafka, Confluent expands the benefits of Kafka with enterprise-grade features while removing the burden of Kafka management or monitoring. Today, over 80% of the Fortune 100 are powered by data streaming technology – and most of those use Confluent.

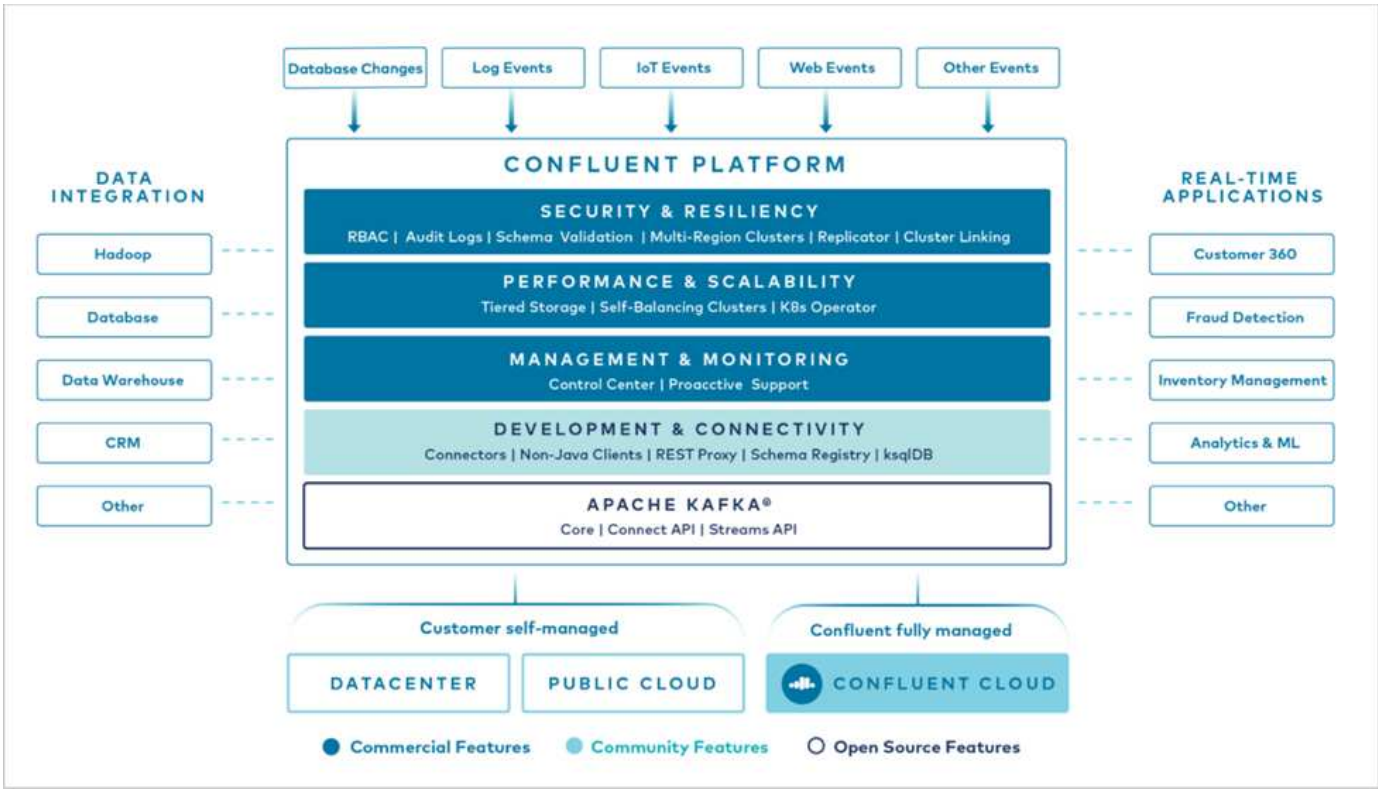
Why Confluent?

By integrating historical and real-time data into a single, central source of truth, Confluent makes it easy to build an entirely new category of modern, event-driven applications, gain a universal data pipeline, and unlock powerful new use cases with full scalability, performance, and reliability.

What is Confluent used for?

Confluent Platform lets you focus on how to derive business value from your data rather than worrying about the underlying mechanics, such as how data is being transported or integrated between disparate systems. Specifically, Confluent Platform simplifies connecting data sources to Kafka, building streaming applications, as well as securing, monitoring, and managing your Kafka infrastructure. Today, Confluent Platform is used for a wide array of use cases across numerous industries, from financial services, omnichannel retail, and autonomous cars, to fraud detection, microservices, and IoT.

The following figure shows Confluent Kafka Platform components.



Overview of Confluent's event streaming technology

At the core of Confluent Platform is Apache Kafka, the most popular open-source distributed streaming platform. The key capabilities of Kafka are as follows:

- Publish and subscribe to streams of records.
- Store streams of records in a fault tolerant way.
- Process streams of records.

Out of the box, Confluent Platform also includes Schema Registry, REST Proxy, a total of 100+ prebuilt Kafka connectors, and ksqlDB.

Overview of Confluent platform's enterprise features

- **Confluent Control Center.** A GUI-based system for managing and monitoring Kafka. It allows you to easily manage Kafka Connect and to create, edit, and manage connections to other systems.
- **Confluent for Kubernetes.** Confluent for Kubernetes is a Kubernetes operator. Kubernetes operators extend the orchestration capabilities of Kubernetes by providing the unique features and requirements for a specific platform application. For Confluent Platform, this includes greatly simplifying the deployment

process of Kafka on Kubernetes and automating typical infrastructure lifecycle tasks.

- **Confluent connectors to Kafka.** Connectors use the Kafka Connect API to connect Kafka to other systems such as databases, key-value stores, search indexes, and file systems. Confluent Hub has downloadable connectors for the most popular data sources and sinks, including fully tested and supported versions of these connectors with Confluent Platform. More details can be found here.
- **Self- balancing clusters.** Provides automated load balancing, failure detection and self-healing. It provides support for adding or decommissioning brokers as needed, with no manual tuning.
- **Confluent cluster linking.** Directly connects clusters together and mirrors topics from one cluster to another over a link bridge. Cluster linking simplifies setup of multi-datacenter, multi-cluster, and hybrid cloud deployments.
- **Confluent auto data balancer.** Monitors your cluster for the number of brokers, the size of partitions, number of partitions, and the number of leaders within the cluster. It allows you to shift data to create an even workload across your cluster, while throttling rebalance traffic to minimize the effect on production workloads while rebalancing.
- **Confluent replicator.** Makes it easier than ever to maintain multiple Kafka clusters in multiple data centers.
- **Tiered storage.** Provides options for storing large volumes of Kafka data using your favorite cloud provider, thereby reducing operational burden and cost. With tiered storage, you can keep data on cost-effective object storage and scale brokers only when you need more compute resources.
- **Confluent JMS client.** Confluent Platform includes a JMS-compatible client for Kafka. This Kafka client implements the JMS 1.1 standard API, using Kafka brokers as the backend. This is useful if you have legacy applications using JMS and you would like to replace the existing JMS message broker with Kafka.
- **Confluent MQTT proxy.** Provides a way to publish data directly to Kafka from MQTT devices and gateways without the need for a MQTT broker in the middle.
- **Confluent security plugins.** Confluent security plugins are used to add security capabilities to various Confluent Platform tools and products. Currently, there is a plugin available for the Confluent REST proxy that helps to authenticate the incoming requests and propagate the authenticated principal to requests to Kafka. This enables Confluent REST proxy clients to utilize the multitenant security features of the Kafka broker.

Next: Confluent verification.

Confluent verification

Previous: Technology overview.

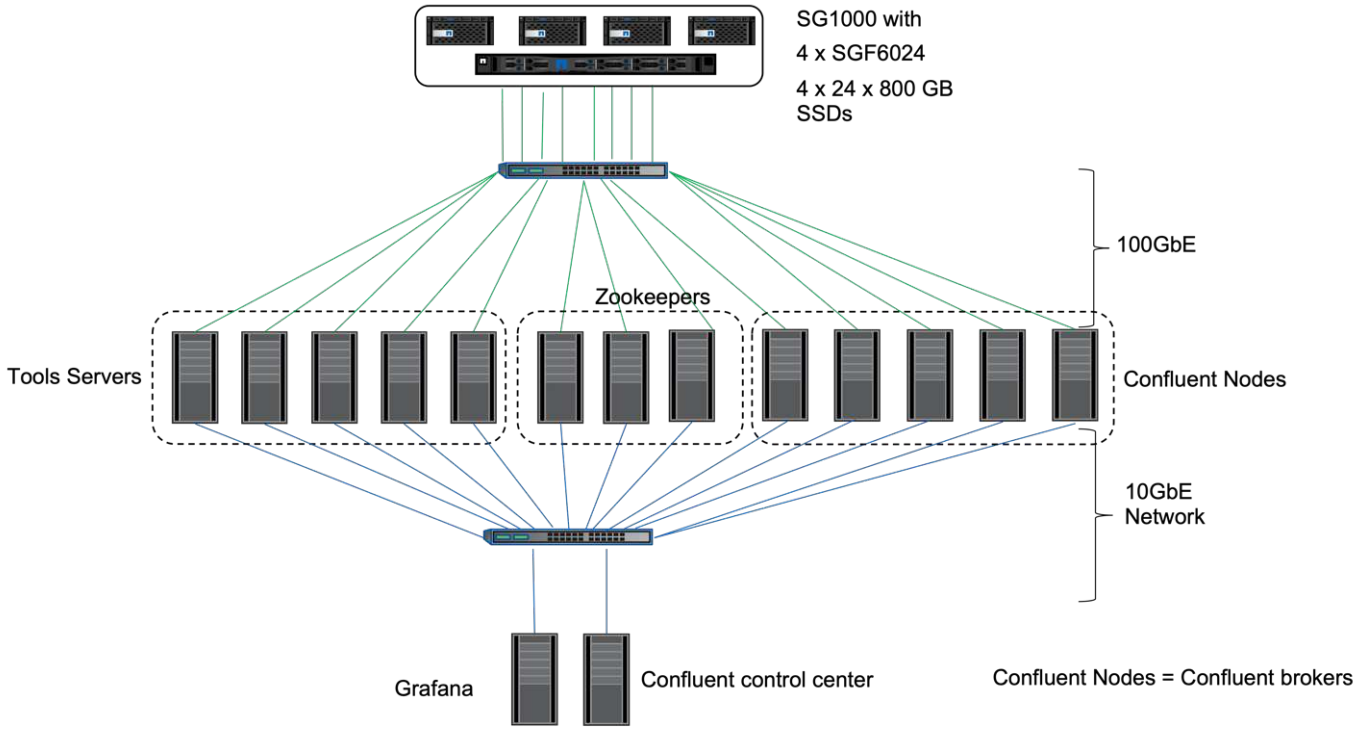
We performed verification with Confluent Platform 6.2 Tiered Storage in NetApp StorageGRID. The NetApp and Confluent teams worked on this verification together and ran the test cases required for verification.

Confluent Platform setup

We used the following setup for verification.

For verification, we used three zookeepers, five brokers, five test-script executing servers, named tools servers with 256GB RAM, and 16 CPUs. For NetApp storage, we used StorageGRID with an SG1000 load balancer with four SGF6024s. The storage and brokers were connected via 100GbE connections.

The following figure shows the network topology of configuration used for Confluent verification.



The tools servers act as application clients that send requests to Confluent nodes.

Confluent tiered storage configuration

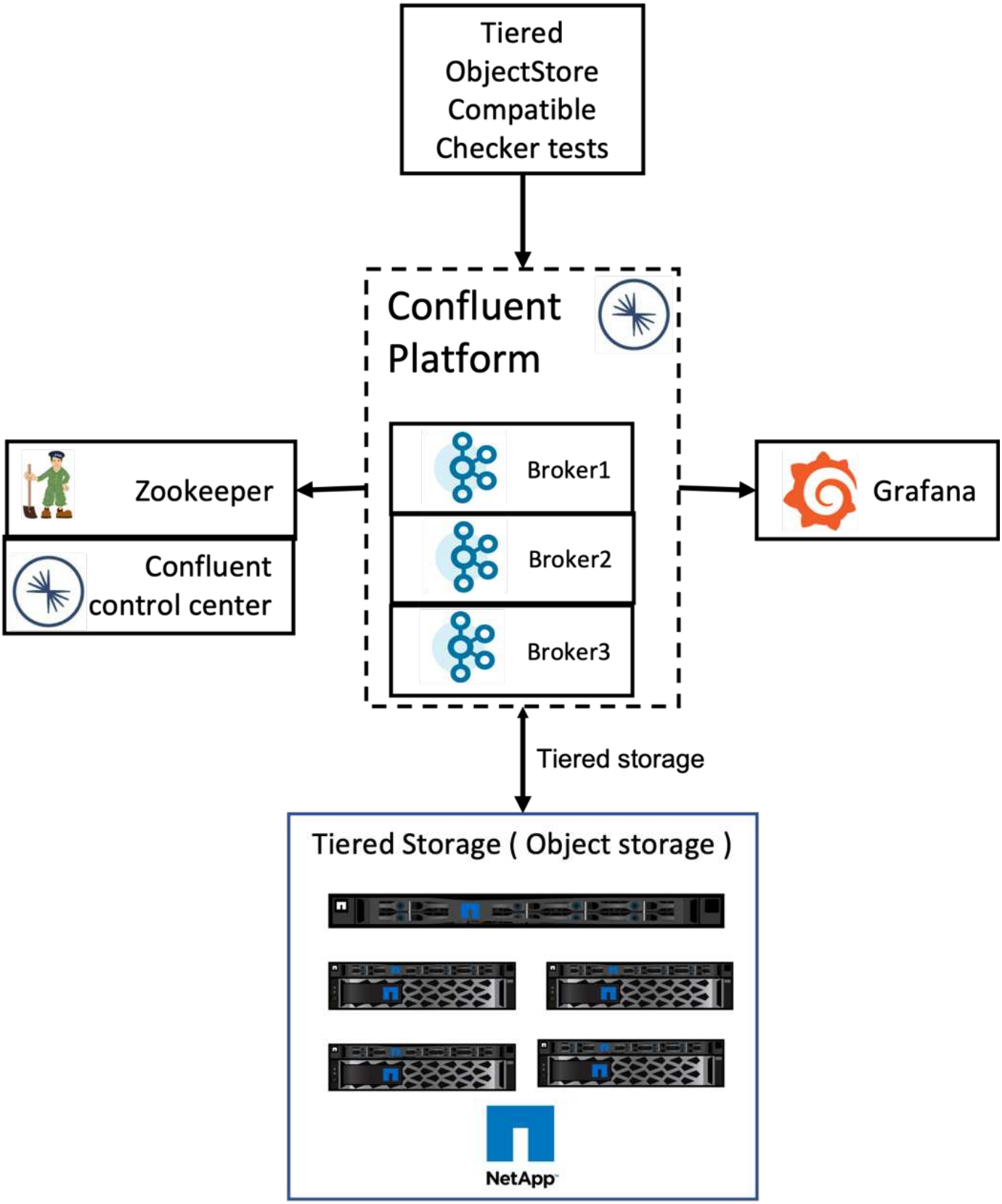
The tiered storage configuration requires the following parameters in Kafka:

```
Confluent.tier.archiver.num.threads=16
confluent.tier.fetcher.num.threads=32
confluent.tier.enable=true
confluent.tier.feature=true
confluent.tier.backend=S3
confluent.tier.s3.bucket=kafkasgdbucket1-2
confluent.tier.s3.region=us-west-2
confluent.tier.s3.cred.file.path=/data/kafka/.ssh/credentials
confluent.tier.s3.aws.endpoint.override=http://kafkasgd.rtpppe.netapp.com:
10444/
confluent.tier.s3.force.path.style.access=true
```

For verification, we used StorageGRID with the HTTP protocol, but HTTPS also works. The access key and secret key are stored in the file name provided in the `confluent.tier.s3.cred.file.path` parameter.

NetApp object storage - StorageGRID

We configured single-site configuration in StorageGRID for verfication.



Verification tests

We completed the following five test cases for the verification. These tests are executed on the Trogdor framework. The first two were functionality tests and the remaining three were performance tests.

Object store correctness test

This test determines whether all basic operations (for example, get/put/delete) on the object store API work well according to the needs of tiered storage. It is a basic test that every object store service should expect to pass ahead of the following tests. It is an assertive test that either passes or fails.

Tiering functionality correctness test

This test determines if end-to-end tiered storage functionality works well with an assertive test that either passes or fails. The test creates a test topic that by default is configured with tiering enabled and highly a reduced hotset size. It produces an event stream to the newly created test topic, it waits for the brokers to archive the segments to the object store, and it then consumes the event stream and validates that the consumed stream matches the produced stream. The number of messages produced to the event stream is configurable, which lets the user generate a sufficiently large workload according to the needs of testing. The reduced hotset size ensures that the consumer fetches outside the active segment are served only from the object store; this helps test the correctness of the object store for reads. We have performed this test with and without an object-store fault injection. We simulated node failure by stopping the service manager service in one of the nodes in StorageGRID and validating that the end-to-end functionality works with object storage.

Tier fetch benchmark

This test validated the read performance of the tiered object storage and checked the range fetch read requests under heavy load from segments generated by the benchmark. In this benchmark, Confluent developed custom clients to serve the tier fetch requests.

Produce-consume workload benchmark

This test indirectly generated write workload on the object store through the archival of segments. The read workload (segments read) was generated from object storage when consumer groups fetched the segments. This workload was generated by the test script. This test checked the performance of read and write on the object storage in parallel threads. We tested with and without object store fault injection as we did for the tiering functionality correctness test.

Retention workload benchmark

This test checked the deletion performance of an object store under a heavy topic-retention workload. The retention workload was generated using a test script that produces many messages in parallel to a test topic. The test topic was configuring with an aggressive size-based and time-based retention setting that caused the event stream to be continuously purged from the object store. The segments were then archived. This led to a large number of deletions in the object storage by the broker and collection of the performance of the object-store delete operations.

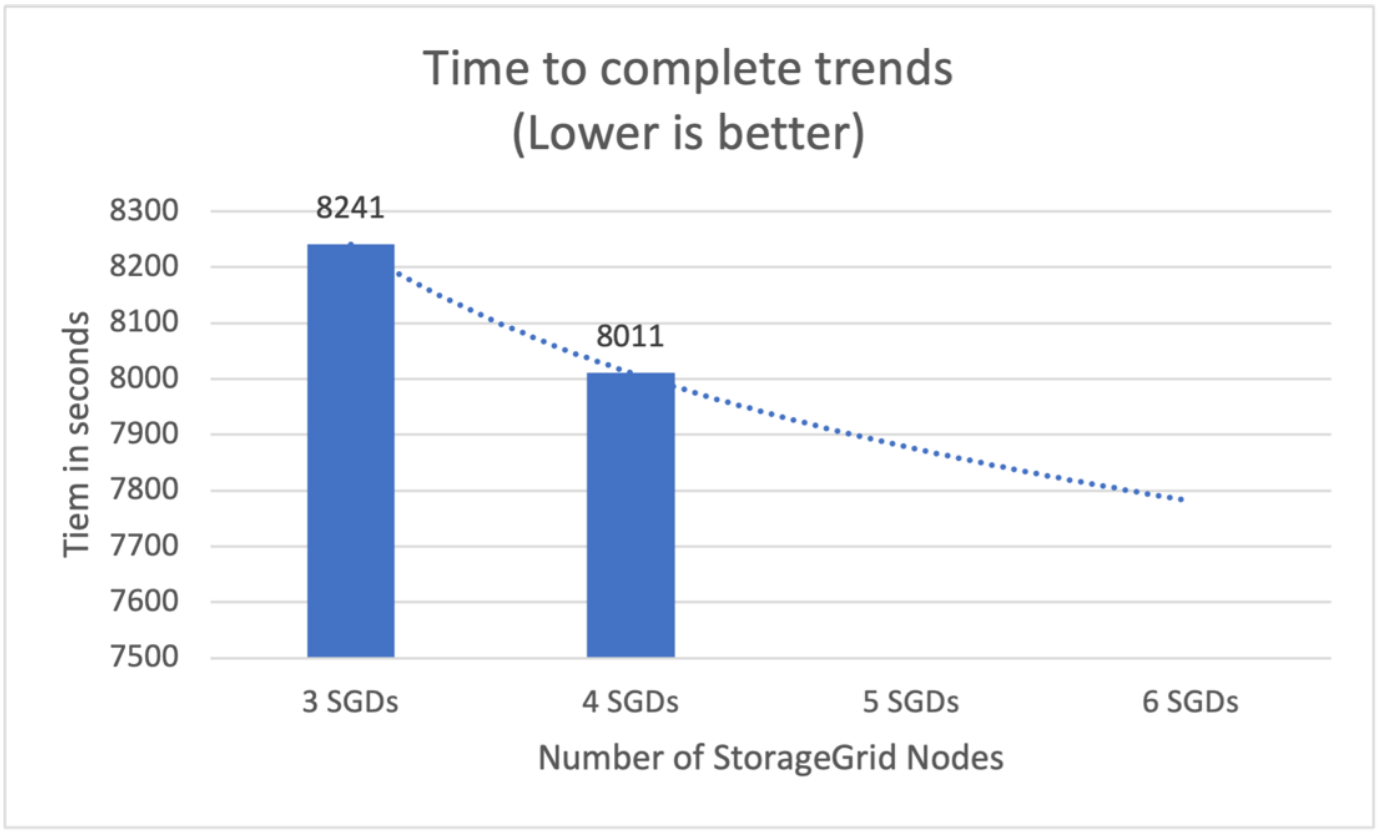
Next: Performance tests with scalability.

Performance tests with scalability

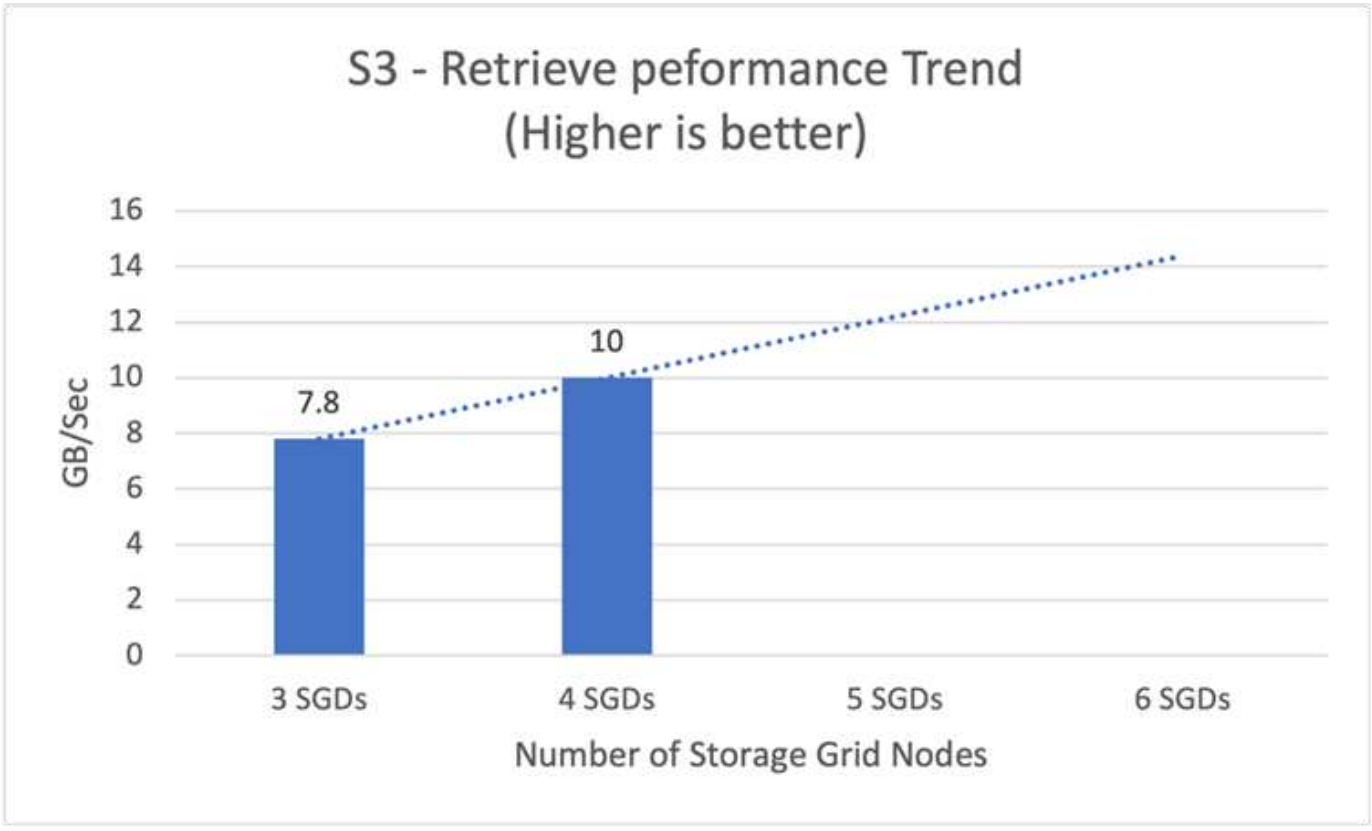
Previous: Confluent verification.

We performed the tiered storage testing with three to four nodes for producer and consumer workloads with the NetApp StorageGRID setup. According to our tests, the time to completion and the performance results were directly proportional to the number of StorageGRID nodes. The StorageGRID setup required a minimum of three nodes.

- The time to complete the produce and consumer operation decreased linearly when the number of storage nodes increased.



- The performance for the s3 retrieve operation increased linearly based on number of StorageGRID nodes. StorageGRID supports up to 200 StorgeGRID nodes.



Next: Confluent s3 connector.

Confluent s3 connector

Previous: Performance tests with scalability.

The Amazon S3 Sink connector exports data from Apache Kafka topics to S3 objects in either the Avro, JSON, or Bytes formats. The Amazon S3 sink connector periodically polls data from Kafka and in turn uploads it to S3. A partitioner is used to split the data of every Kafka partition into chunks. Each chunk of data is represented as an S3 object. The key name encodes the topic, the Kafka partition, and the start offset of this data chunk.

In this setup, we show you how to read and write topics in object storage from Kafka directly using the Kafka s3 sink connector. For this test, we used a stand-alone Confluent cluster, but this setup is applicable to a distributed cluster.

1. Download Confluent Kafka from the Confluent website.
2. Unpack the package to a folder on your server.
3. Export two variables.

```
Export CONFLUENT_HOME=/data/confluent/confluent-6.2.0
export PATH=$PATH:/data/confluent/confluent-6.2.0/bin
```

4. For a stand-alone Confluent Kafka setup, the cluster creates a temporary root folder in `/tmp`. It also creates Zookeeper, Kafka, a schema registry, connect, a ksql-server, and control-center folders and copies their respective configuration files from `$CONFLUENT_HOME`. See the following example:

```
root@stlrx2540m1-108:~# ls -ltr /tmp/confluent.406980/
total 28
drwxr-xr-x 4 root root 4096 Oct 29 19:01 zookeeper
drwxr-xr-x 4 root root 4096 Oct 29 19:37 kafka
drwxr-xr-x 4 root root 4096 Oct 29 19:40 schema-registry
drwxr-xr-x 4 root root 4096 Oct 29 19:45 kafka-rest
drwxr-xr-x 4 root root 4096 Oct 29 19:47 connect
drwxr-xr-x 4 root root 4096 Oct 29 19:48 ksql-server
drwxr-xr-x 4 root root 4096 Oct 29 19:53 control-center
root@stlrx2540m1-108:~#
```

5. Configure Zookeeper. You don’t need to change anything if you use the default parameters.

```
root@stlrx2540m1-108:~# cat
/tmp/confluent.406980/zookeeper/zookeeper.properties  | grep -iv ^#
dataDir=/tmp/confluent.406980/zookeeper/data
clientPort=2181
maxClientCnxns=0
admin.enableServer=false
tickTime=2000
initLimit=5
syncLimit=2
server.179=controlcenter:2888:3888
root@stlrx2540m1-108:~#
```

In the above configuration, we updated the `server. xxx` property. By default, you need three Zookeepers for the Kafka leader selection.

6. We created a myid file in `/tmp/confluent.406980/zookeeper/data` with a unique ID:

```
root@stlrx2540m1-108:~# cat /tmp/confluent.406980/zookeeper/data/myid
179
root@stlrx2540m1-108:~#
```

We used the last number of IP addresses for the myid file. We used default values for the Kafka, connect, control-center, Kafka, Kafka-rest, ksql-server, and schema-registry configurations.

7. Start the Kafka services.

```
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin# confluent
local services  start
The local commands are intended for a single-node development
environment only,
NOT for production usage.

Using CONFLUENT_CURRENT: /tmp/confluent.406980
ZooKeeper is [UP]
Kafka is [UP]
Schema Registry is [UP]
Kafka REST is [UP]
Connect is [UP]
ksqlDB Server is [UP]
Control Center is [UP]
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin#
```

There is a log folder for each configuration, which helps troubleshoot issues. In some instances, services take more time to start. Make sure all services are up and running.

8. Install Kafka connect using `confluent-hub`.

```
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin# ./confluent-
hub install confluentinc/kafka-connect-s3:latest
The component can be installed in any of the following Confluent
Platform installations:
  1. /data/confluent/confluent-6.2.0 (based on $CONFLUENT_HOME)
  2. /data/confluent/confluent-6.2.0 (where this tool is installed)
Choose one of these to continue the installation (1-2): 1
Do you want to install this into /data/confluent/confluent-
6.2.0/share/confluent-hub-components? (yN) y

Component's license:
Confluent Community License
http://www.confluent.io/confluent-community-license
I agree to the software license agreement (yN) y
Downloading component Kafka Connect S3 10.0.3, provided by Confluent,
Inc. from Confluent Hub and installing into /data/confluent/confluent-
6.2.0/share/confluent-hub-components
Do you want to uninstall existing version 10.0.3? (yN) y
Detected Worker's configs:
  1. Standard: /data/confluent/confluent-6.2.0/etc/kafka/connect-
distributed.properties
  2. Standard: /data/confluent/confluent-6.2.0/etc/kafka/connect-
standalone.properties
  3. Standard: /data/confluent/confluent-6.2.0/etc/schema-
registry/connect-avro-distributed.properties
  4. Standard: /data/confluent/confluent-6.2.0/etc/schema-
registry/connect-avro-standalone.properties
  5. Based on CONFLUENT_CURRENT:
/tmp/confluent.406980/connect/connect.properties
  6. Used by Connect process with PID 15904:
/tmp/confluent.406980/connect/connect.properties
Do you want to update all detected configs? (yN) y
Adding installation directory to plugin path in the following files:
  /data/confluent/confluent-6.2.0/etc/kafka/connect-
distributed.properties
  /data/confluent/confluent-6.2.0/etc/kafka/connect-
standalone.properties
  /data/confluent/confluent-6.2.0/etc/schema-registry/connect-avro-
distributed.properties
  /data/confluent/confluent-6.2.0/etc/schema-registry/connect-avro-
standalone.properties
  /tmp/confluent.406980/connect/connect.properties
  /tmp/confluent.406980/connect/connect.properties
```

```
Completed
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin#
```

You can also install a specific version by using `confluent-hub install confluentinc/kafka-connect-s3:10.0.3`.

9. By default, `confluentinc-kafka-connect-s3` is installed in `/data/confluent/confluent-6.2.0/share/confluent-hub-components/confluentinc-kafka-connect-s3`.
10. Update the plug-in path with the new `confluentinc-kafka-connect-s3`.

```
root@stlrx2540m1-108:~# cat /data/confluent/confluent-
6.2.0/etc/kafka/connect-distributed.properties | grep plugin.path
#
plugin.path=/usr/local/share/java,/usr/local/share/kafka/plugins,/opt/co
nnectors,
plugin.path=/usr/share/java,/data/zookeeper/confluent/confluent-
6.2.0/share/confluent-hub-components,/data/confluent/confluent-
6.2.0/share/confluent-hub-components,/data/confluent/confluent-
6.2.0/share/confluent-hub-components/confluentinc-kafka-connect-s3
root@stlrx2540m1-108:~#
```

11. Stop the Confluent services and restart them.

```
confluent local services  stop
confluent local services  start
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin# confluent
local services  status
The local commands are intended for a single-node development
environment only,
NOT for production usage.

Using CONFLUENT_CURRENT: /tmp/confluent.406980
Connect is [UP]
Control Center is [UP]
Kafka is [UP]
Kafka REST is [UP]
ksqlDB Server is [UP]
Schema Registry is [UP]
ZooKeeper is [UP]
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin#
```

12. Configure the access ID and secret key in the `/root/.aws/credentials` file.

```
root@stlrx2540m1-108:~# cat /root/.aws/credentials
[default]
aws_access_key_id = xxxxxxxxxxxx
aws_secret_access_key = xxxxxxxxxxxxxxxxxxxxxxxxx
root@stlrx2540m1-108:~#
```

13. Verify that the bucket is reachable.

```
root@stlrx2540m4-01:~# aws s3 –endpoint-url
http://kafkasgd.rtpppe.netapp.com:10444 ls kafkasgdbucket1-2
2021-10-29 21:04:18       1388 1
2021-10-29 21:04:20       1388 2
2021-10-29 21:04:22       1388 3
root@stlrx2540m4-01:~#
```

14. Configure the s3-sink properties file for s3 and bucket configuration.

```
root@stlrx2540m1-108:~# cat /data/confluent/confluent-
6.2.0/share/confluent-hub-components/confluentinc-kafka-connect-
s3/etc/quickstart-s3.properties | grep -v ^#
name=s3-sink
connector.class=io.confluent.connect.s3.S3SinkConnector
tasks.max=1
topics=s3_testtopic
s3.region=us-west-2
s3.bucket.name=kafkasgdbucket1-2
store.url=http://kafkasgd.rtpppe.netapp.com:10444/
s3.part.size=5242880
flush.size=3
storage.class=io.confluent.connect.s3.storage.S3Storage
format.class=io.confluent.connect.s3.format.avro.AvroFormat
partitioner.class=io.confluent.connect.storage.partitioner.DefaultPartit
ioner
schema.compatibility=NONE
root@stlrx2540m1-108:~#
```

15. Import a few records to the s3 bucket.

```
kafka-avro-console-producer --broker-list localhost:9092 --topic
s3_topic \
--property
value.schema='{"type":"record","name":"myrecord","fields":[{"name":"f1",
"type":"string"}]}'
{"f1": "value1"}
{"f1": "value2"}
{"f1": "value3"}
{"f1": "value4"}
{"f1": "value5"}
{"f1": "value6"}
{"f1": "value7"}
{"f1": "value8"}
{"f1": "value9"}
```

16. Load the s3-sink connector.

```
root@stlrx2540m1-108:~# confluent local services connect connector load
s3-sink  --config /data/confluent/confluent-6.2.0/share/confluent-hub-
components/confluentinc-kafka-connect-s3/etc/quickstart-s3.properties
The local commands are intended for a single-node development
environment only,
NOT for production usage.
https://docs.confluent.io/current/cli/index.html
{
  "name": "s3-sink",
  "config": {
    "connector.class": "io.confluent.connect.s3.S3SinkConnector",
    "flush.size": "3",
    "format.class": "io.confluent.connect.s3.format.avro.AvroFormat",
    "partitioner.class":
"io.confluent.connect.storage.partitioner.DefaultPartitioner",
    "s3.bucket.name": "kafkasgdbucket1-2",
    "s3.part.size": "5242880",
    "s3.region": "us-west-2",
    "schema.compatibility": "NONE",
    "storage.class": "io.confluent.connect.s3.storage.S3Storage",
    "store.url": "http://kafkasgd.rtpppe.netapp.com:10444/",
    "tasks.max": "1",
    "topics": "s3_testtopic",
    "name": "s3-sink"
  },
  "tasks": [],
  "type": "sink"
}
root@stlrx2540m1-108:~#
```

17. Check the s3-sink status.

```
root@stlrx2540m1-108:~# confluent local services connect connector
status s3-sink
The local commands are intended for a single-node development
environment only,
NOT for production usage.
https://docs.confluent.io/current/cli/index.html
{
  "name": "s3-sink",
  "connector": {
    "state": "RUNNING",
    "worker_id": "10.63.150.185:8083"
  },
  "tasks": [
    {
      "id": 0,
      "state": "RUNNING",
      "worker_id": "10.63.150.185:8083"
    }
  ],
  "type": "sink"
}
root@stlrx2540m1-108:~#
```

18. Check the log to make sure that s3-sink is ready to accept topics.

```
root@stlrx2540m1-108:~# confluent local services connect log
```

19. Check the topics in Kafka.

```
kafka-topics --list --bootstrap-server localhost:9092
…
connect-configs
connect-offsets
connect-statuses
default_ksql_processing_log
s3_testtopic
s3_topic
s3_topic_new
root@stlrx2540m1-108:~#
```

20. Check the objects in the s3 bucket.

```
root@stlrx2540m1-108:~# aws s3 --endpoint-url
http://kafkasgd.rtpppe.netapp.com:10444 ls --recursive kafkasgdbucket1-
2/topics/
2021-10-29 21:24:00        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000000.avro
2021-10-29 21:24:00        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000003.avro
2021-10-29 21:24:00        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000006.avro
2021-10-29 21:24:08        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000009.avro
2021-10-29 21:24:08        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000012.avro
2021-10-29 21:24:09        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000015.avro
root@stlrx2540m1-108:~#
```

21. To verify the contents, copy each file from S3 to your local filesystem by running the following command:

```
root@stlrx2540m1-108:~# aws s3 --endpoint-url
http://kafkasgd.rtpppe.netapp.com:10444 cp s3://kafkasgdbucket1-
2/topics/s3_testtopic/partition=0/s3_testtopic+0+0000000000.avro
tes.avro
download: s3://kafkasgdbucket1-
2/topics/s3_testtopic/partition=0/s3_testtopic+0+0000000000.avro to
./tes.avro
root@stlrx2540m1-108:~#
```

22. To print the records, use avro-tools-1.11.0.1.jar (available in the Apache Archives).

```
root@stlrx2540m1-108:~# java -jar /usr/src/avro-tools-1.11.0.1.jar
tojson tes.avro
21/10/30 00:20:24 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
{"f1":"value1"}
{"f1":"value2"}
{"f1":"value3"}
root@stlrx2540m1-108:~#
```

Next: Confluent self-rebalancing clusters.

Confluent Self-balancing Clusters

Previous: Kafka s3 connector.

If you have managed a Kafka cluster before, you are likely familiar with the challenges that come with manually reassigning partitions to different brokers to make sure that the workload is balanced across the cluster. For organizations with large Kafka deployments, reshuffling large amounts of data can be daunting, tedious, and risky, especially if mission-critical applications are built on top of the cluster. However, even for the smallest Kafka use cases, the process is time consuming and prone to human error.

In our lab, we tested the Confluent self-balancing clusters feature, which automates rebalancing based on cluster topology changes or uneven load. The Confluent rebalance test helps to measure the time to add a new broker when node failure or the scaling node requires rebalancing data across brokers. In classic Kafka configurations, the amount of data to be rebalanced grows as the cluster grows, but, in tiered storage, rebalancing is restricted to a small amount of data. Based on our validation, rebalancing in tiered storage takes seconds or minutes in a classic Kafka architecture and grows linearly as the cluster grows.

In self-balancing clusters, partition rebalances are fully automated to optimize Kafka's throughput, accelerate broker scaling, and reduce the operational burden of running a large cluster. At steady-state, self-balancing clusters monitor the skew of data across the brokers and continuously reassigns partitions to optimize cluster performance. When scaling the platform up or down, self-balancing clusters automatically recognize the presence of new brokers or the removal of old brokers and trigger a subsequent partition reassignment. This enables you to easily add and decommission brokers, making your Kafka clusters fundamentally more elastic. These benefits come without any need for manual intervention, complex math, or the risk of human error that partition reassignments typically entail. As a result, data rebalances are completed in far less time, and you are free to focus on higher-value event-streaming projects rather than needing to constantly supervise your clusters.

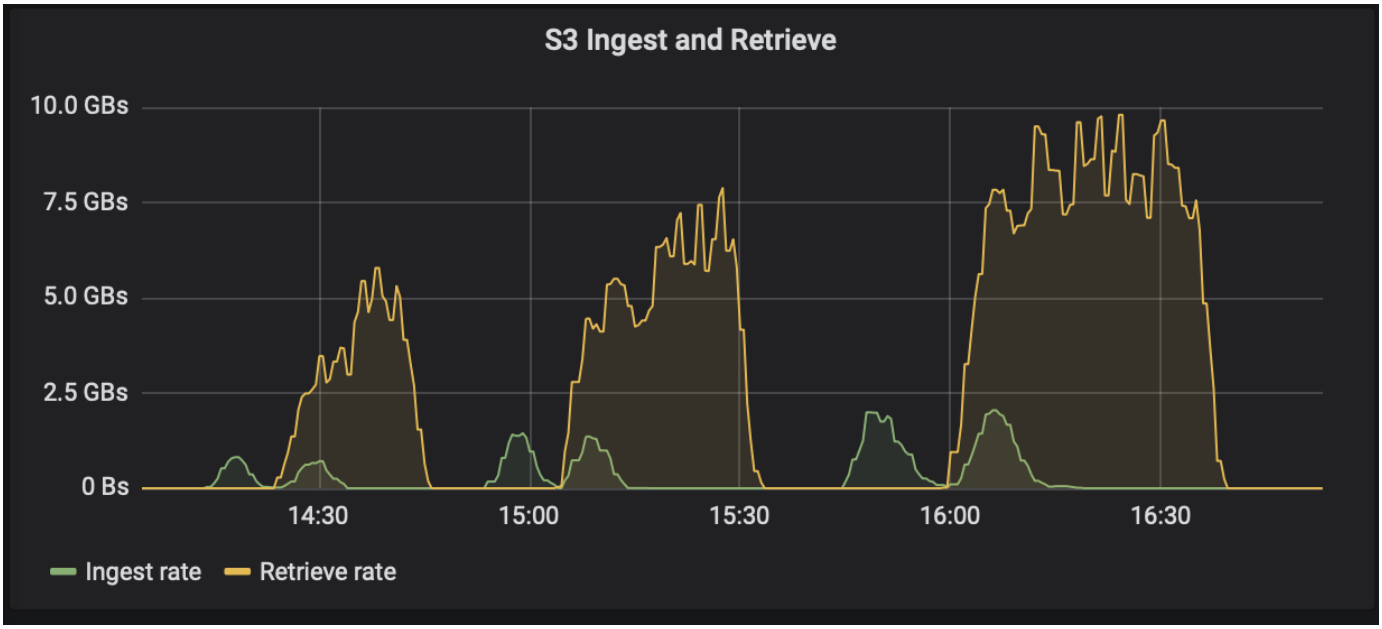
Next: Best practice guidelines.

Best practice guidelines

Previous: Confluent self-rebalancing clusters.

- Based on our validation, S3 object storage is best for Confluent to keep data.
- We can use high-throughput SAN (specifically FC) to keep the broker hot data or local disk, because, in the Confluent tiered storage configuration, the size of the data held in the brokers data directory is based on the segment size and retention time when the data is moved to object storage.
- Object stores provide better performance when segment.bytes is higher; we tested 512MB.
- In Kafka, the length of the key or value (in bytes) for each record produced to the topic is controlled by the `length.key.value` parameter. For StorageGRID, S3 object ingest and retrieve performance increased to higher values. For example, 512 bytes provided a 5.8GBps retrieve, 1024 bytes provided a 7.5GBps s3 retrieve, and 2048 bytes provided close to 10GBps.

The following figure presents the S3 object ingest and retrieve based on `length.key.value`.



- **Kafka tuning.** To improve the performance of tiered storage, you can increase TierFetcherNumThreads and TierArchiverNumThreads. As a general guideline, you want to increase TierFetcherNumThreads to match the number of physical CPU cores and increase TierArchiverNumThreads to half the number of CPU cores. For example, in server properties, if you have a machine with eight physical cores, set confluent.tier.fetcher.num.threads = 8 and confluent.tier.archiver.num.threads = 4.
- **Time interval for topic deletes.** When a topic is deleted, deletion of the log segment files in object storage does not immediately begin. Rather, there is a time interval with a default value of 3 hours before deletion of those files takes place. You can modify the configuration, confluent.tier.topic.delete.check.interval.ms, to change the value of this interval. If you delete a topic or cluster, you can also manually delete the objects in the respective bucket.
- **ACLs on tiered storage internal topics.** A recommended best practice for on-premises deployments is to enable an ACL authorizer on the internal topics used for tiered storage. Set ACL rules to limit access on this data to the broker user only. This secures the internal topics and prevents unauthorized access to tiered storage data and metadata.

```
kafka-acls --bootstrap-server localhost:9092 --command-config adminclient-
configs.conf \
--add --allow-principal User:<kafka> --operation All --topic "_confluent-
tier-state"
```

Replace the user `<kafka>` with the actual broker principal in your deployment.

For example, the command `confluent-tier-state` sets ACLs on the internal topic for tiered storage. Currently, there is only a single internal topic related to tiered storage. The example creates an ACL that provides the principal Kafka permission for all operations on the internal topic.

Next: Sizing.

Sizing

Previous: Best practice guidelines.

Kafka sizing can be performed with four configuration modes: simple, granular, reverse, and partitions.

Simple

The simple mode is appropriate for the first-time Apache Kafka users or early state use cases. For this mode, you provide requirements such as throughput MBps, read fanout, retention, and the resource utilization percentage (60% is default). You also enter the environment, such as on-premises (bare-metal, VMware, Kubernetes, or OpenStack) or cloud. Based on this information, the sizing of a Kafka cluster provides the number of servers required for the broker, the zookeeper, Apache Kafka connect workers, the schema registry, a REST Proxy, ksqlDB, and the Confluent control center.

For tiered storage, consider the granular configuration mode for sizing a Kafka cluster. Granular mode is appropriate for experienced Apache Kafka users or well-defined use cases. This section describes sizing for producers, stream processors, and consumers.

Producers

To describe the producers for Apache Kafka (for example a native client, REST proxy, or Kafka connector), provide the following information:

- **Name.** Spark.
- **Producer type.** Application or service, proxy (REST, MQTT, other), and existing database (RDBMS, NOSQL, other). You can also select “I don’t know.”
- **Average throughput.** In events per second (1,000,000 for example).
- **Peak throughput.** In events per second (4,000,000 for example).
- **Average message size.** In bytes, uncompressed (max 1MB; 1000 for example).
- **Message format.** Options include Avro, JSON, protocol buffers, binary, text, “I don’t know,” and other.
- **Replication factor.** Options are 1, 2, 3 (Confluent recommendation), 4, 5, or 6.
- **Retention time.** One day (for example). How long do you want your data to be stored in Apache Kafka? Enter -1 with any unit for an infinite time. The calculator assumes a retention time of 10 years for infinite retention.
- Select the check box for "Enable Tiered Storage to Decrease Broker Count and Allow for Infinite Storage?"
- When tiered storage is enabled, the retention fields control the hot set of data that is stored locally on the broker. The archival retention fields control how long data is stored in archival object storage.
- **Archival Storage Retention.** One year (for example). How long do you want your data to be stored in archival storage? Enter -1 with any unit for an infinite duration. The calculator assumes a retention of 10 years for infinite retention.
- **Growth Multiplier.** 1 (for example). If the value of this parameter is based on current throughput, set it to 1. To size based on additional growth, set this parameter to a growth multiplier.
- **Number of producer instances.** 10 (for example). How many producer instances will be running? This input is required to incorporate the CPU load into the sizing calculation. A blank value indicates that CPU load is not incorporated into the calculation.

Based on this example input, sizing has the following effect on producers:

- Average throughput in uncompressed bytes: 1GBps. Peak throughput in uncompressed bytes: 4GBps. Average throughput in compressed bytes: 400MBps. Peak throughput in compressed bytes: 1.6GBps. This is based on a default 60% compression rate (you can change this value).

    - Total on-broker hotset storage required: 31,104TB, including replication, compressed. Total off-broker archival storage required: 378,432TB, compressed. Use https://fusion.netapp.com for StorageGRID sizing.

Stream Processors must describe their applications or services that consume data from Apache Kafka and produce back into Apache Kafka. In most cases these are built in ksqlDB or Kafka Streams.

- **Name.** Spark streamer.
- **Processing time.** How long does this processor take to process a single message?
    - 1 ms (simple, stateless transformation) [example], 10ms (stateful in-memory operation).
    - 100ms (stateful network or disk operation), 1000ms (3rd party REST call).
    - I have benchmarked this parameter and know exactly how long it takes.
- **Output Retention.** 1 day (example). A stream processor produces its output back to Apache Kafka. How long do you want this output data to be stored in Apache Kafka? Enter -1 with any unit for an infinite duration.
- Select the check box "Enable Tiered Storage to Decrease Broker Count and Allow for Infinite Storage?"
- **Archival Storage Retention.** 1 year (for example). How long do you want your data to be stored in archival storage? Enter -1 with any unit for an infinite duration. The calculator assumes a retention of 10 years for infinite retention.
- **Output Passthrough Percentage.** 100 (for example). A stream processor produces its output back to Apache Kafka. What percentage of inbound throughput will be outputted back into Apache Kafka? For example, if inbound throughput is 20MBps and this value is 10, the output throughput will be 2MBps.
- From which applications does this read from? Select “Spark,” the name used in producer type-based sizing.
  Based on the above input, you can expect the following effects of sizing on stream processer instances and topic partition estimates:
- This stream processor application requires the following number of instances. The incoming topics likely require this many partitions as well. Contact Confluent to confirm this parameter.
    - 1,000 for average throughput with no growth multiplier
    - 4,000 for peak throughput with no growth multiplier
    - 1,000 for average throughput with a growth multiplier
    - 4,000 for peak throughput with a growth multiplier

Consumers

Describe your applications or services that consume data from Apache Kafka and do not produce back into Apache Kafka; for example, a native client or Kafka Connector.

- **Name.** Spark consumer.
- **Processing time.** How long does this consumer take to process a single message?
    - 1ms (for example, a simple and stateless task like logging)
    - 10ms (fast writes to a datastore)
    - 100ms (slow writes to a datastore)
    - 1000ms (third party REST call)
    - Some other benchmarked process of known duration.

- **Consumer type.** Application, proxy, or sink to an existing datastore (RDBMS, NoSQL, other).
- From which applications does this read from? Connect this parameter with producer and stream sizing determined previously.

Based on the above input, you must determine the sizing for consumer instances and topic partition estimates. A consumer application requires the following number of instances.

- 2,000 for average throughput, no growth multiplier
- 8,000 for peak throughput, no growth multiplier
- 2,000 for average throughput, including growth multiplier
- 8,000 for peak throughput, including growth multiplier

The incoming topics likely need this number of partitions as well. Contact Confluent to confirm.

In addition to the requirements for producers, stream processors, and consumers, you must provide the following additional requirements:

- **Rebuild time.** For example, 4 hours. If an Apache Kafka broker host fails, its data is lost, and a new host is provisioned to replace the failed host, how fast must this new host rebuild itself? Leave this parameter blank if the value is unknown.
- **Resource utilization target (percentage).** For example, 60. How utilized do you want your hosts to be during average throughput? Confluent recommends 60% utilization unless you are using Confluent self-balancing clusters, in which case utilization can be higher.

Describe your environment

- **What environment will your cluster be running in?** Amazon Web Services, Microsoft Azure, Google cloud platform, bare-metal on premises, VMware on premises, OpenStack on premises, or Kubernates on premises?
- **Host details.** Number of cores: 48 (for example), network card type (10GbE, 40GbE, 16GbE, 1GbE, or another type).
- **Storage volumes.** Host: 12 (for example). How many hard drives or SSDs are supported per host? Confluent recommends 12 hard drives per host.
- **Storage capacity/volume (in GB).** 1000 (for example). How much storage can a single volume store in gigabytes? Confluent recommends 1TB disks.
- **Storage configuration.** How are storage volumes configured? Confluent recommends RAID10 to take advantage of all Confluent features. JBOD, SAN, RAID 1, RAID 0, RAID 5, and other types are also supported.
- **Single volume throughput (MBps).** 125 (for example). How fast can a single storage volume read or write in megabytes per second? Confluent recommends standard hard drives, which typically have 125MBps throughput.
- **Memory capacity (GB).** 64 (for example).

After you have determined your environmental variables, select Size my Cluster. Based on the example parameters indicated above, we determined the following sizing for Confluent Kafka:

- **Apache Kafka.** Broker count: 22. Your cluster is storage-bound. Consider enabling tiered storage to decrease your host count and allow for infinite storage.
- **Apache ZooKeeper.** Count: 5; Apache Kafka Connect Workers: Count: 2; Schema Registry: Count: 2;

REST Proxy: Count: 2; ksqlDB: Count: 2; Confluent Control Center: Count: 1.

Use reverse mode for platform teams without a use case in mind. Use partitions mode to calculate how many partitions a single topic requires. See https://eventsizer.io for sizing based on the reverse and partitions modes.

Next: Conclusion.

Conclusion

Previous: Sizing.

This document provides best practice guidelines for using Confluent Tiered Storage with NetApp storage, including verification tests, tiered storage performance results, tuning, Confluent S3 connectors, and the self-balancing feature. Considering ILM policies, Confluent performance with multiple performance tests for verification, and industry-standard S3 APIs, NetApp StorageGRID object storage is an optimal choice for Confluent tiered storage.

Where to find additional information

To learn more about the information that is described in this document, review the following documents and/or websites:

- What is Apache Kafka

  https://www.confluent.io/what-is-apache-kafka/

- NetApp Product Documentation

  https://www.netapp.com/support-and-training/documentation/

- S3-sink parameter details

  https://docs.confluent.io/kafka-connect-s3-sink/current/configuration_options.html#s3-configuration-options

- Apache Kafka

  https://en.wikipedia.org/wiki/Apache_Kafka

- Infinite Storage in Confluent Platform

  https://www.confluent.io/blog/infinite-kafka-storage-in-confluent-platform/

- Confluent Tiered Storage - Best practices and sizing

  https://docs.confluent.io/platform/current/kafka/tiered-storage.html#best-practices-and-recommendations

- Amazon S3 sink connector for Confluent Platform

  https://docs.confluent.io/kafka-connect-s3-sink/current/overview.html

- Kafka sizing

  https://eventsizer.io

- StorageGRID sizing

https://fusion.netapp.com/

- Kafka use cases

  https://kafka.apache.org/uses

- Self-balancing Kafka clusters in confluent platform 6.0

  https://www.confluent.io/blog/self-balancing-kafka-clusters-in-confluent-platform-6-0/

  https://www.confluent.io/blog/confluent-platform-6-0-delivers-the-most-powerful-event-streaming-platform-to-date/

Version history

Version	Date	Document version history
Version 1.0	December 2021	Initial release.

NetApp hybrid cloud data solutions - Spark and Hadoop based on customer use cases

TR-4657: NetApp hybrid cloud data solutions - Spark and Hadoop based on customer use cases

Karthikeyan Nagalingam and Sathish Thyagarajan, NetApp

This document describes hybrid cloud data solutions using NetApp AFF and FAS storage systems, NetApp Cloud Volumes ONTAP, NetApp connected storage, and NetApp FlexClone technology for Spark and Hadoop. These solution architectures allow customers to choose an appropriate data protection solution for their environment. NetApp designed these solutions based on interaction with customers and their business use-cases. This document provides the following detailed information:

- Why we need data protection for Spark and Hadoop environments and customer challenges.
- The data fabric powered by NetApp vision and its building blocks and services.
- How these building blocks can be used to architect flexible data protection workflows.
- The pros and cons of several architectures based on real-world customer use cases. Each use case provides the following components:
  - Customer scenarios
  - Requirements and challenges
  - Solutions
  - Summary of the solutions

Why Hadoop data protection?

In a Hadoop and Spark environment, the following concerns must be addressed:

- **Software or human failures.** Human error in software updates while carrying out Hadoop data operations can lead to faulty behavior that can cause unexpected results from the job. In such case, we need to protect the data to avoid failures or unreasonable outcomes. For example, as the result of a poorly

executed software update to a traffic signal analysis application, a new feature that fails to properly analyze traffic signal data in the form of plain text. The software still analyzes JSON and other non- text file formats, resulting in the real-time traffic control analytics system producing prediction results that are missing data points. This situation can cause faulty outputs that might lead to accidents at the traffic signals. Data protection can address this issue by providing the capability to quickly roll back to the previous working application version.

- **Size and scale.** The size of the analytics data grows day by day due to the ever-increasing numbers of data sources and volume. Social media, mobile apps, data analytics, and cloud computing platforms are the main sources of data in the current big data market, which is increasing very rapidly, and therefore the data needs to be protected to ensure accurate data operations.
- **Hadoop's native data protection.** Hadoop has a native command to protect the data, but this command does not provide consistency of data during backup. It only supports directory-level backup. The snapshots created by Hadoop are read-only and cannot be used to reuse the backup data directly.

Data protection challenges for Hadoop and Spark customers

A common challenge for Hadoop and Spark customers is to reduce the backup time and increase backup reliability without negatively affecting performance at the production cluster during data protection.

Customers also need to minimize recovery point objective (RPO) and recovery time objective (RTO) downtime and control their on-premises and cloud-based disaster recovery sites for optimal business continuity. This control typically comes from having enterprise-level management tools.

The Hadoop and Spark environments are complicated because not only is the data volume huge and growing, but the rate this data arrives is increasing. This scenario makes it difficult to rapidly create efficient, up-to-date DevTest and QA environments from the source data. NetApp recognizes these challenges and offers the solutions presented in this paper.

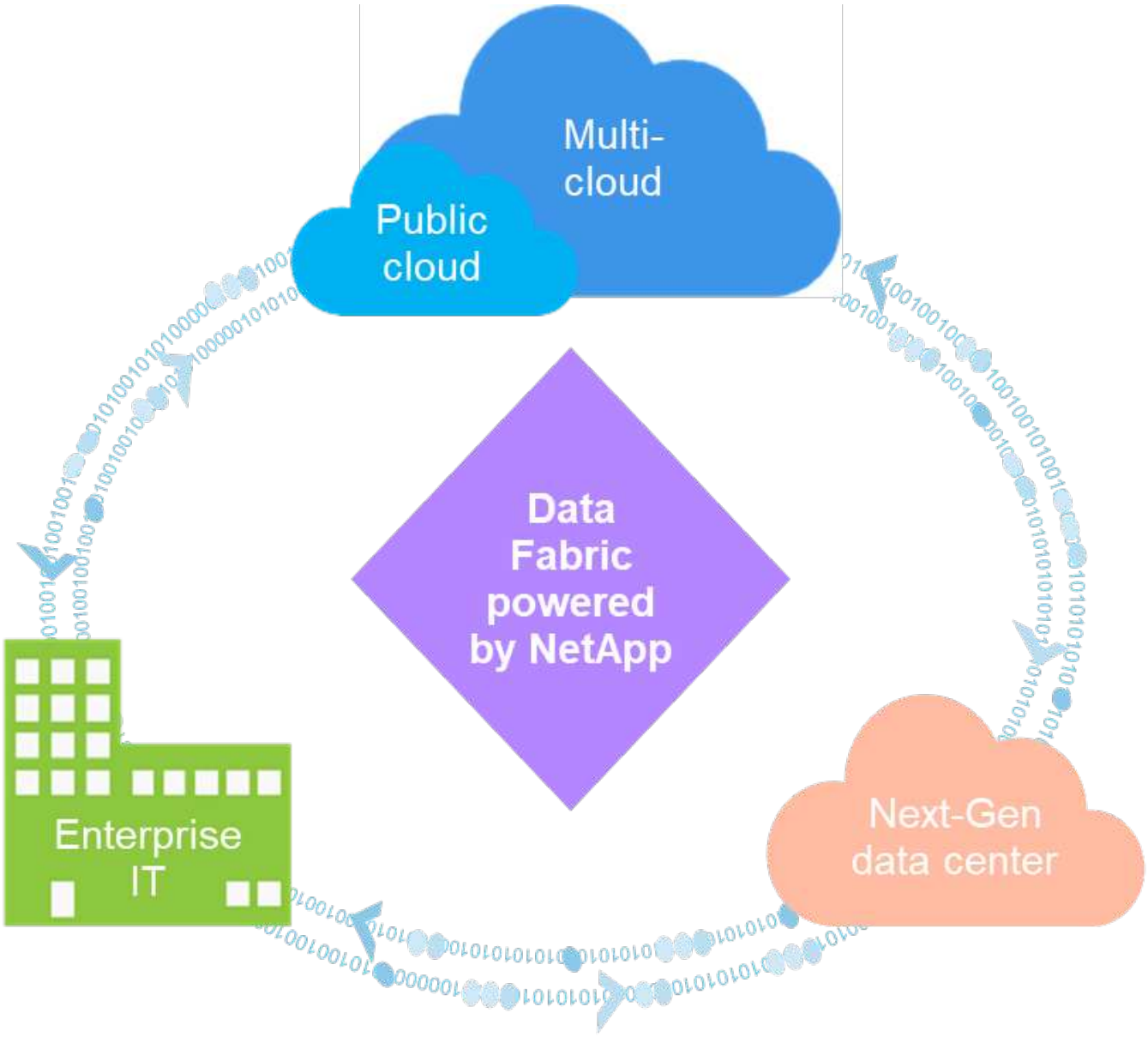
Next: Data fabric powered by NetApp for big data architecture.

Data fabric powered by NetApp for big data architecture

Previous: Solution overview.

The data fabric powered by NetApp simplifies and integrates data management across cloud and on-premises environments to accelerate digital transformation.

The data fabric powered by NetApp delivers consistent and integrated data management services and applications (building blocks) for data visibility and insights, data access and control, and data protection and security, as shown in the figure below.



Proven data fabric customer use cases

The data fabric powered by NetApp provides the following nine proven use cases for customers:

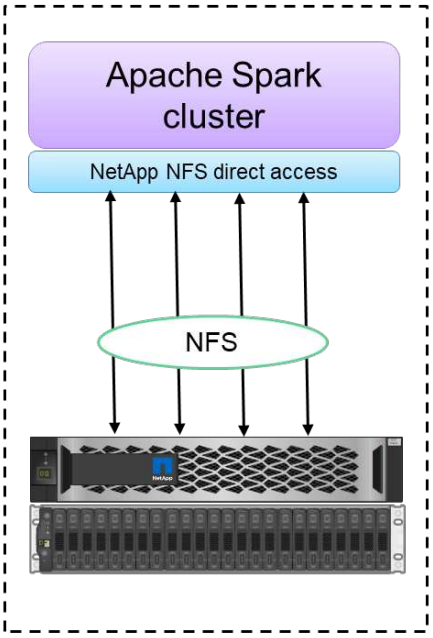
- Accelerate analytics workloads
- Accelerate DevOps transformation
- Build cloud hosting infrastructure
- Integrate cloud data services
- Protect and secure data
- Optimize unstructured data
- Gain data center efficiencies
- Deliver data insights and control
- Simplify and automate

This document covers two of the nine use cases (along with their solutions):

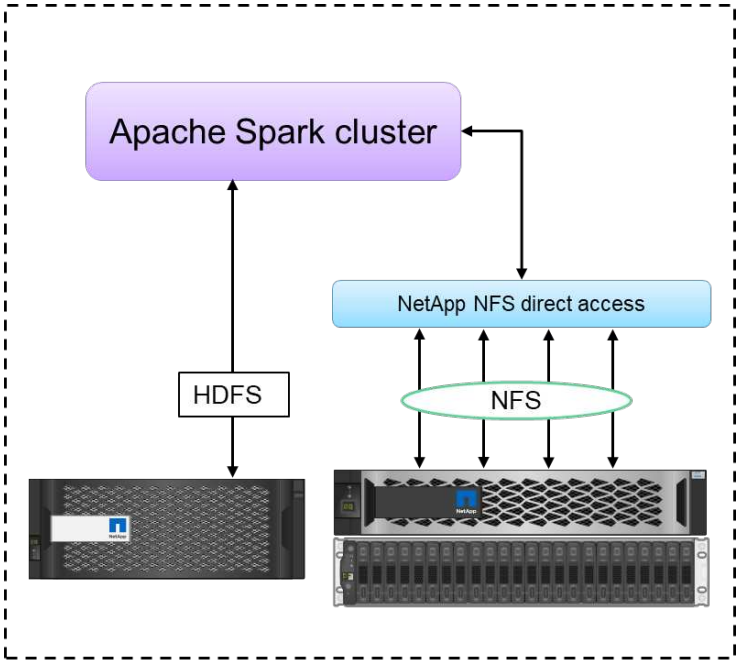
- Accelerate analytics workloads
- Protect and secure data

NetApp NFS direct access

The NetApp NFS direct access (formerly known as NetApp In-Place Analytics Module) (shown in the figure below) allows customers to run big data analytics jobs on their existing or new NFSv3 or NFSv4 data without moving or copying the data. It prevents multiple copies of data and eliminates the need to sync the data with a source. For example, in the financial sector, the movement of data from one place to another place must meet legal obligations, which is not an easy task. In this scenario, the NetApp NFS direct access analyzes the financial data from its original location. Another key benefit is that using the NetApp NFS direct access simplifies protecting Hadoop data by using native Hadoop commands and enables data protection workflows leveraging NetApp's rich data management portfolio.



Configuration 1: NFS as primary storage



Configuration 2: HDFS and NFS in single Spark cluster

The NetApp NFS direct access provides two kinds of deployment options for Hadoop/Spark clusters:

- By default, the Hadoop/Spark clusters use Hadoop Distributed File System (HDFS) for data storage and the default file system. The NetApp NFS direct access can replace the default HDFS with NFS storage as the default file system, enabling direct analytics operations on NFS data.
- In another deployment option, the NetApp NFS direct access supports configuring NFS as additional storage along with HDFS in a single Hadoop/Spark cluster. In this case, the customer can share data through NFS exports and access it from the same cluster along with HDFS data.

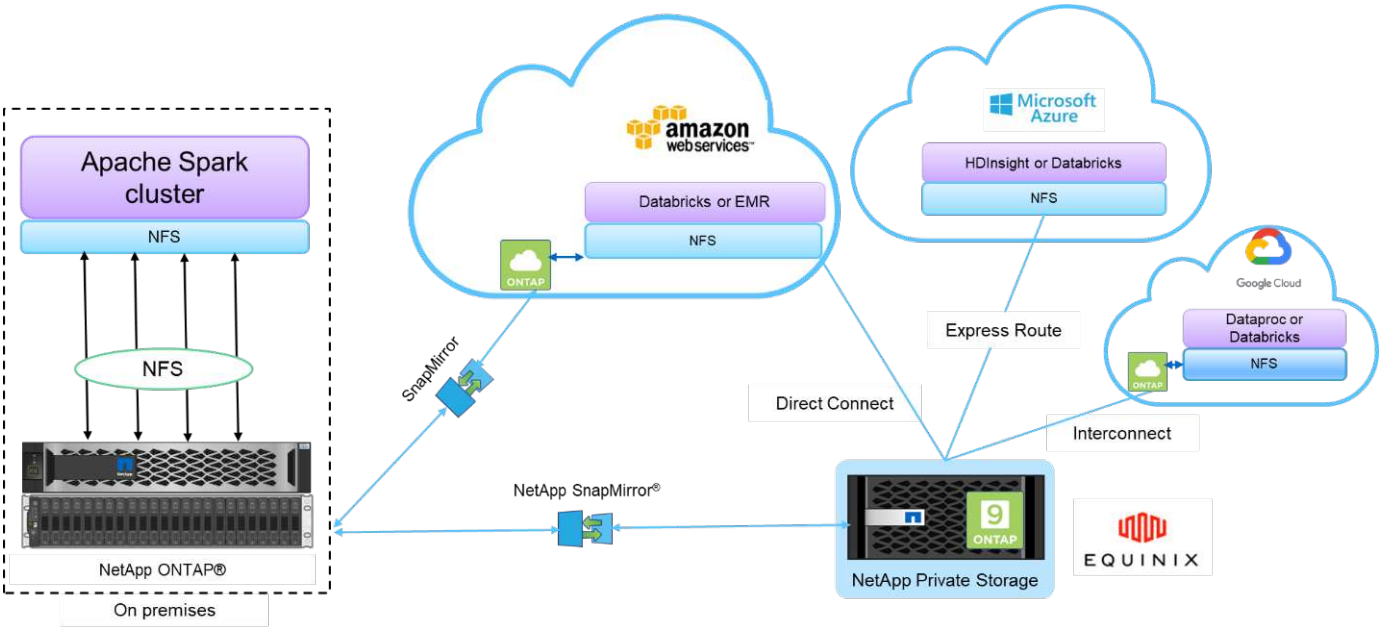
The key benefits of using the NetApp NFS direct access include:

- Analyzes the data from its current location, which prevents the time- and performance-consuming task of moving analytics data to a Hadoop infrastructure such as HDFS.
- Reduces the number of replicas from three to one.
- Enables users to decouple the compute and storage to scale them independently.
- Provides enterprise data protection by leveraging the rich data management capabilities of ONTAP.

- Is certified with the Hortonworks data platform.
- Enables hybrid data analytics deployments.
- Reduces the backup time by leveraging dynamic multithread capability.

Building blocks for big data

The data fabric powered by NetApp integrates data management services and applications (building blocks) for data access, control, protection, and security, as shown in the figure below.



The building blocks in the figure above include:

- **NetApp NFS direct access.** Provides the latest Hadoop and Spark clusters with direct access to NetApp NFS volumes without additional software or driver requirements.
- **NetApp Cloud Volumes ONTAP and Cloud Volume Services.** Software-defined connected storage based on ONTAP running in Amazon Web Services (AWS) or Azure NetApp Files (ANF) in Microsoft Azure cloud services.
- **NetApp SnapMirror technology**. Provides data protection capabilities between on-premises and ONTAP Cloud or NPS instances.
- **Cloud service providers.** These providers include AWS, Microsoft Azure, Google Cloud, and IBM Cloud.
- **PaaS.** Cloud-based analytics services such as Amazon Elastic MapReduce (EMR) and Databricks in AWS as well as Microsoft Azure HDInsight and Azure Databricks.

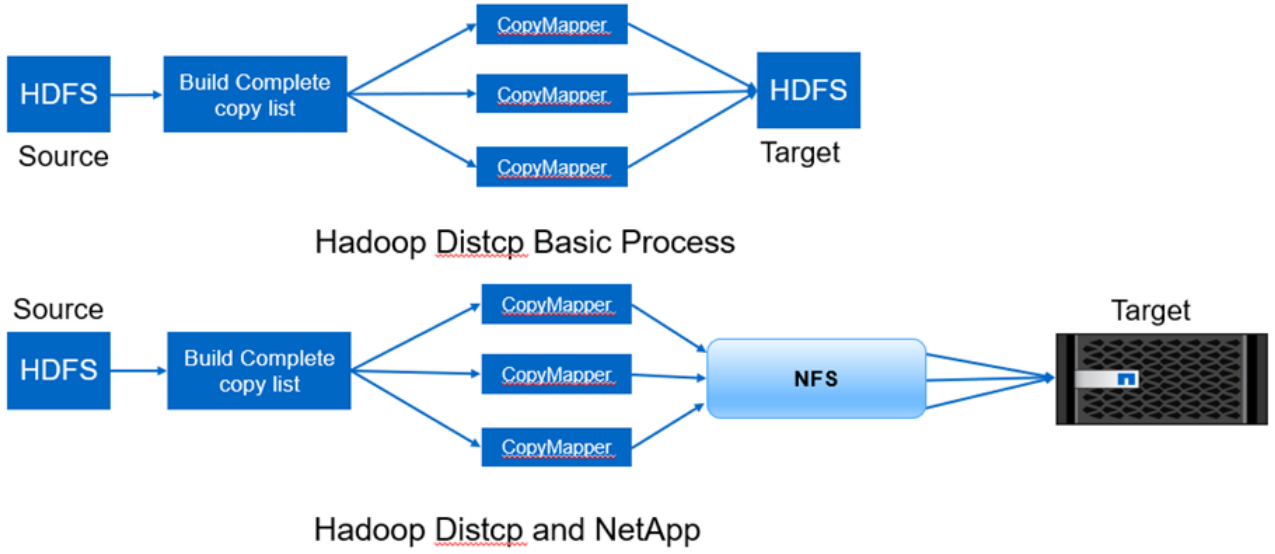
Next: Hadoop data protection and NetApp.

Hadoop data protection and NetApp

Previous: Data fabric powered by NetApp for big data architecture.

Hadoop DistCp is a native tool used for large intercluster and intracluster copying. The Hadoop DistCp basic process shown in the figure below is a typical backup workflow using Hadoop native tools such as MapReduce to copy Hadoop data from an HDFS source to a corresponding target. The NetApp NFS direct access enables customers to set NFS as the target destination for the Hadoop DistCp tool to copy the data from HDFS source

into an NFS share through MapReduce. The NetApp NFS direct access acts as an NFS driver for the DistCp tool.



Next: Overview of Hadoop data protection use cases.

Overview of Hadoop data protection use cases

Previous: Hadoop data protection and NetApp.

This section provides a high-level description of the data protection use cases, which constitute the focus of this paper. The remaining sections provide more details for each use case, such as the customer problem (scenario), requirements and challenges, and solutions.

Use case 1: Backing up Hadoop data

For this use case, the In-Place Analytics Module helped a large financial institution reduce the long backup window time from more than 24 hours to just under a few hours.

Use case 2: Backup and disaster recovery from the cloud to on-premises

By using the data fabric powered by NetApp as building blocks, a large broadcasting company was able to fulfill its requirement of backing up cloud data into its on-premise data center depending on the different modes of data transfers, such as on demand, instantaneous, or based on the Hadoop/Spark cluster load.

Use case 3: Enabling DevTest on existing Hadoop data

NetApp solutions helped an online music distributor to rapidly build multiple space-efficient Hadoop clusters in different branches to create reports and run daily DevTest tasks by using scheduled policies.

Use case 4: Data protection and multicloud connectivity

A large service provider used the data fabric powered by NetApp to provide multicloud analytics to its customers from different cloud instances.

Use case 5: Accelerate analytic workloads

One of the largest financial services and investment banks used the NetApp network-attached storage solution to reduce I/O wait time and accelerate its quantitative financial analytics platform.

Next: Use case 1 - Backing up Hadoop data.

Use case 1: Backing up Hadoop data

Previous: Overview of Hadoop data protection use cases.

Scenario

In this scenario, the customer has a large on-premises Hadoop repository and wants to back it up for disaster recovery purposes. However, the customer's current backup solution is costly and is suffering from a long backup window of more than 24 hours.

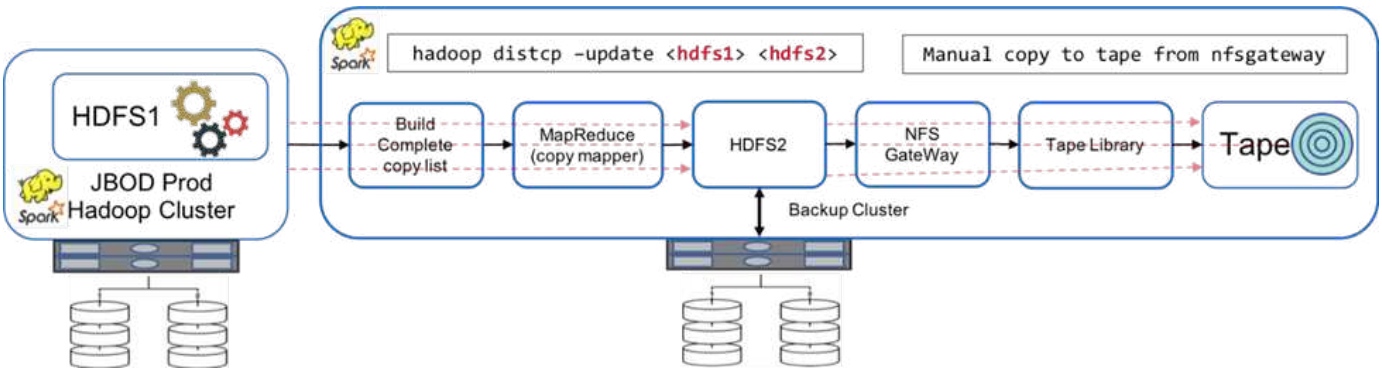
Requirements and challenges

The main requirements and challenges for this use case include:

- Software backward compatibility:
  - The proposed alternative backup solution should be compatible with the current running software versions used in the production Hadoop cluster.
- To meet the committed SLAs, the proposed alternative solution should achieve very low RPOs and RTOs.
- The backup created by the NetApp backup solution can be used in the Hadoop cluster built locally in the data center as well as the Hadoop cluster running in the disaster recovery location at the remote site.
- The proposed solution must be cost effective.
- The proposed solution must reduce the performance effect on the currently running, in-production analytics jobs during the backup times.

Customer's existing backup solution

The figure below shows the original Hadoop native backup solution.



The production data is protected to tape through the intermediate backup cluster:

- HDFS1 data is copied to HDFS2 by running the `hadoop distcp -update <hdfs1> <hdfs2>` command.
- The backup cluster acts as an NFS gateway, and the data is manually copied to tape through the Linux `cp`

command through the tape library.

The benefits of the original Hadoop native backup solution include:

- The solution is based on Hadoop native commands, which saves the user from having to learn new procedures.
- The solution leverages industry-standard architecture and hardware.

The disadvantages of the original Hadoop native backup solution include:

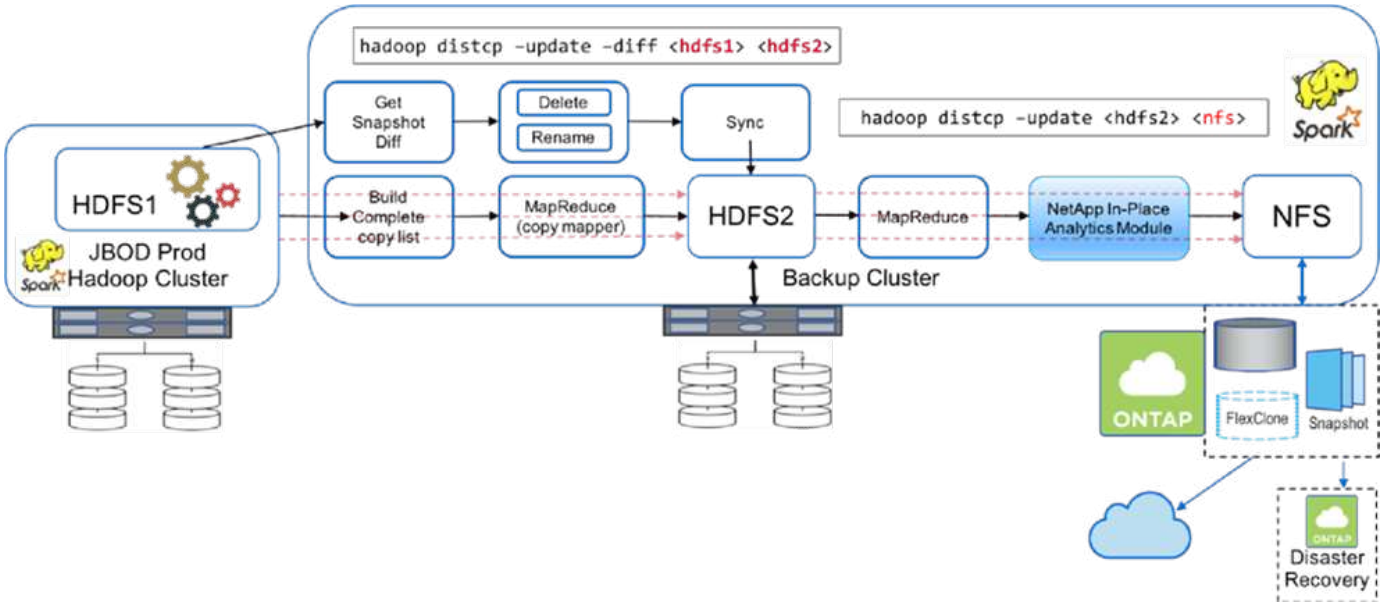
- The long backup window time exceeds 24 hours, which makes the production data vulnerable.
- Significant cluster performance degradation during backup times.
- Copying to tape is a manual process.
- The backup solution is expensive in terms of the hardware required and the human hours required for manual processes.

Backup solutions

Based on these challenges and requirements, and taking into consideration the existing backup system, three possible backup solutions were suggested. The following subsections describe each of these three different backup solutions, labeled solution A through solution C.

Solution A

Solution A adds the In-Place Analytics Module to the backup Hadoop cluster, which allows secondary backups to NetApp NFS storage systems, eliminating the tape requirement, as shown in the figure below.



The detailed tasks for solution A include:

- The production Hadoop cluster has the customer's analytics data in the HDFS that requires protection.
- The backup Hadoop cluster with HDFS acts as an intermediate location for the data. Just a bunch of disks (JBOD) provides the storage for HDFS in both the production and backup Hadoop clusters.
- Protect the Hadoop production data is protected from the production cluster HDFS to the backup cluster HDFS by running the `Hadoop distcp -update -diff <hdfs1> <hdfs2>` command.

The Hadoop snapshot is used to protect the data from production to the backup Hadoop cluster.

- The NetApp ONTAP storage controller provides an NFS exported volume, which is provisioned to the backup Hadoop cluster.
- By running the `Hadoop distcp` command leveraging MapReduce and multiple mappers, the analytics data is protected from the backup Hadoop cluster to NFS by using the In-Place Analytics Module.

  After the data is stored in NFS on the NetApp storage system, NetApp Snapshot, SnapRestore, and FlexClone technologies are used to back up, restore, and duplicate the Hadoop data as needed.

Hadoop data can be protected to the cloud as well as disaster recovery locations by using SnapMirror technology.

The benefits of solution A include:

- Hadoop production data is protected from the backup cluster.
- HDFS data is protected through NFS enabling protection to cloud and disaster recovery locations.
- Improves performance by offloading backup operations to the backup cluster.
- Eliminates manual tape operations
- Allows for enterprise management functions through NetApp tools.
- Requires minimal changes to the existing environment.
- Is a cost-effective solution.

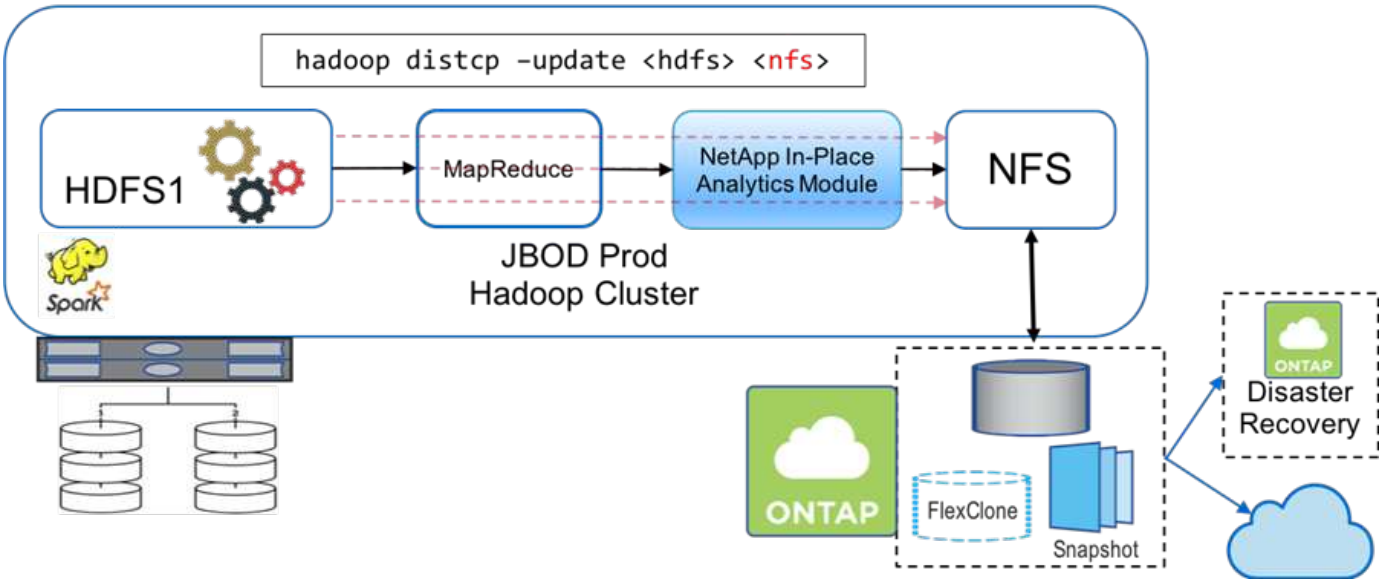
The disadvantage of this solution is that it requires a backup cluster and additional mappers to improve performance.

The customer recently deployed solution A due to its simplicity, cost, and overall performance.

In this solution, SAN disks from ONTAP can be used instead of JBOD. This option offloads the backup cluster storage load to ONTAP; however, the downside is that SAN fabric switches are required.

Solution B

Solution B adds the In-Place Analytics Module to the production Hadoop cluster, which eliminates the need for the backup Hadoop cluster, as shown in the figure below.



The detailed tasks for solution B include:

- The NetApp ONTAP storage controller provisions the NFS export to the production Hadoop cluster.

  The Hadoop native `hadoop distcp` command protects the Hadoop data from the production cluster HDFS to NFS through the In-Place Analytics Module.

- After the data is stored in NFS on the NetApp storage system, Snapshot, SnapRestore, and FlexClone technologies are used to back up, restore, and duplicate the Hadoop data as needed.

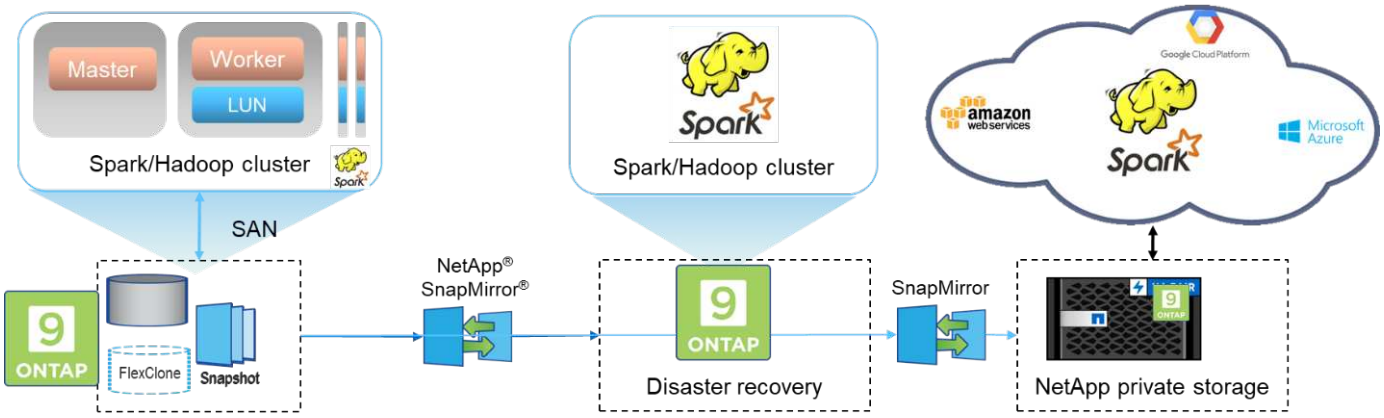
The benefits of solution B include:

- The production cluster is slightly modified for the backup solution, which simplifies implementation and reduces additional infrastructure cost.
- A backup cluster for the backup operation is not required.
- HDFS production data is protected in the conversion to NFS data.
- The solution allows for enterprise management functions through NetApp tools.

The disadvantage of this solution is that it's implemented in the production cluster, which can add additional administrator tasks in the production cluster.

Solution C

In solution C, the NetApp SAN volumes are directly provisioned to the Hadoop production cluster for HDFS storage, as shown in the figure below.



The detailed steps for solution C include:

- NetApp ONTAP SAN storage is provisioned at the production Hadoop cluster for HDFS data storage.
- NetApp Snapshot and SnapMirror technologies are used to back up the HDFS data from the production Hadoop cluster.
- There is no performance effect to production for the Hadoop/Spark cluster during the Snapshot copy backup process because the backup is at the storage layer.

> Snapshot technology provides backups that complete in seconds regardless of the size of the data.

The benefits of solution C include:

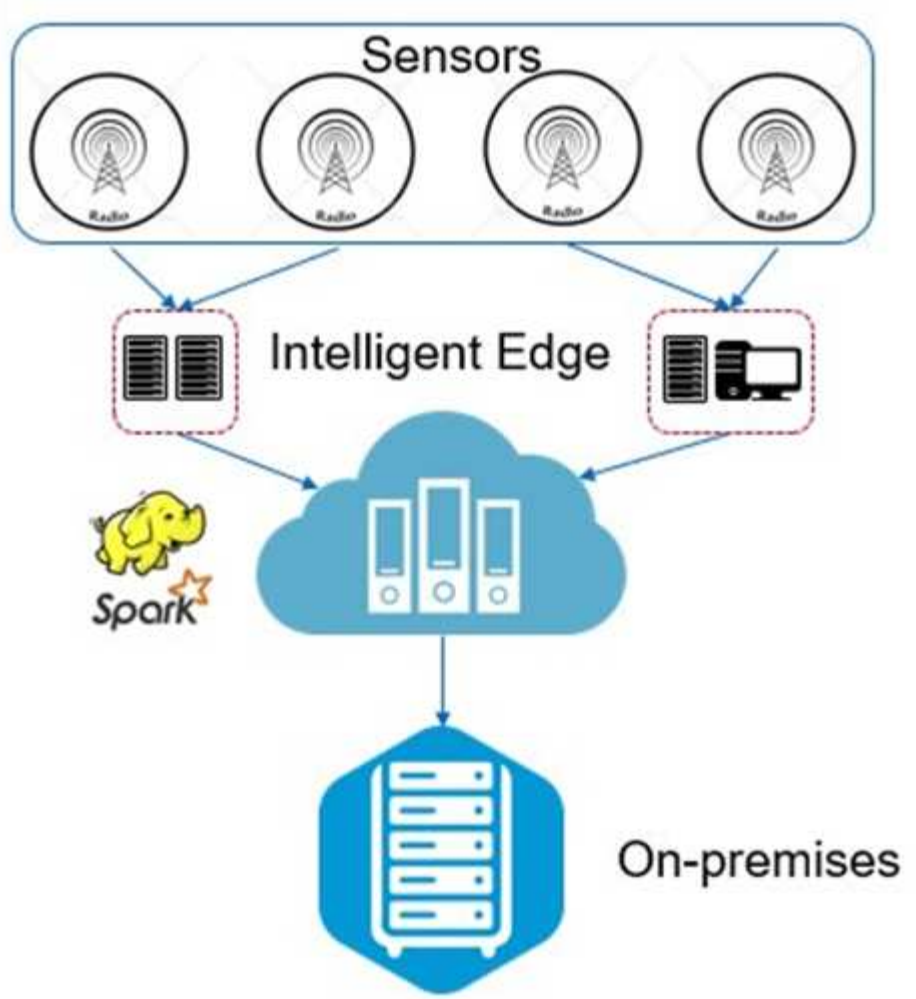
- Space-efficient backup can be created by using Snapshot technology.
- Allows for enterprise management functions through NetApp tools.

Next: Use case 2 - Backup and disaster recovery from the cloud to on-premises.

Use case 2: Backup and disaster recovery from the cloud to on-premises

Previous: Use case 1 - Backing up Hadoop data.

This use case is based on a broadcasting customer that needs to back up cloud-based analytics data to its on-premises data center, as illustrated in the figure below.



Scenario

In this scenario, the IoT sensor data is ingested into the cloud and analyzed by using an open source Apache Spark cluster within AWS. The requirement is to back up the processed data from the cloud to on-premises.

Requirements and challenges

The main requirements and challenges for this use case include:

- Enabling data protection should not cause any performance effect on the production Spark/Hadoop cluster in the cloud.
- Cloud sensor data needs to be moved and protected to on-premises in an efficient and secure way.
- Flexibility to transfer data from the cloud to on-premises under different conditions, such as on-demand, instantaneous, and during low-cluster load times.

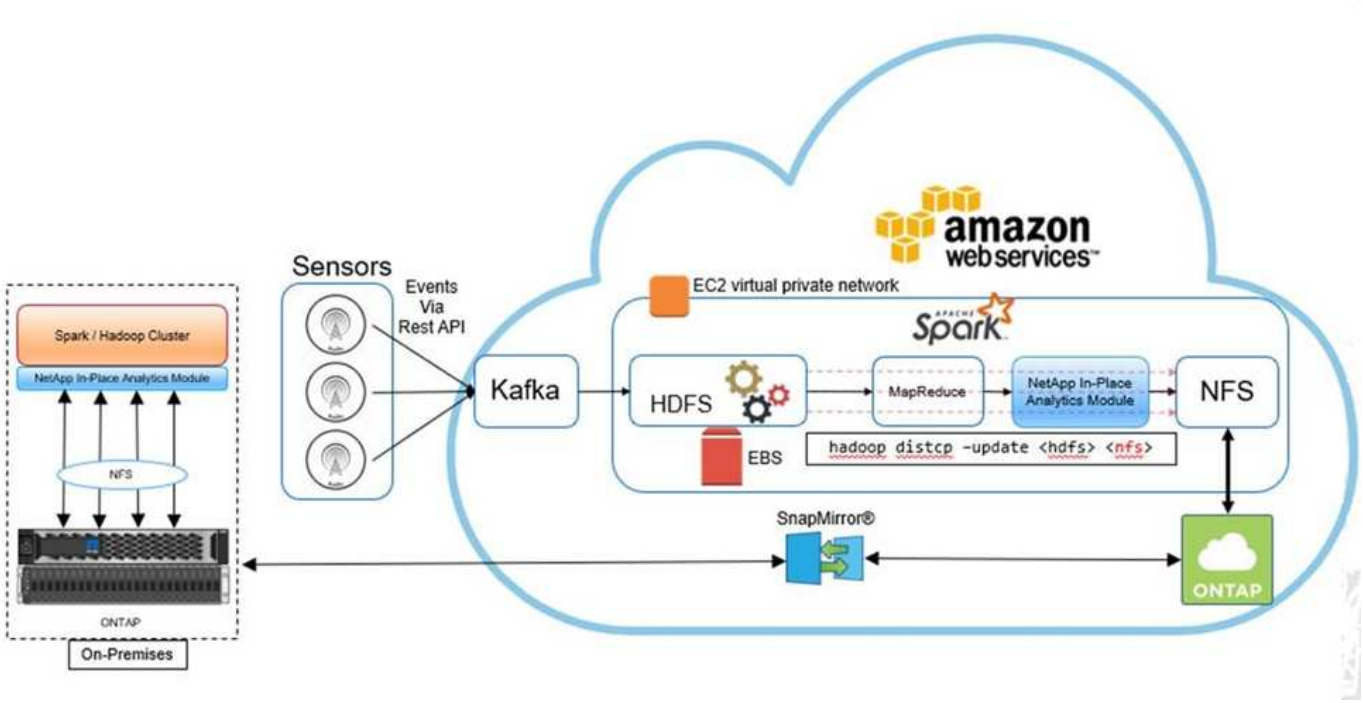
Solution

The customer uses AWS Elastic Block Store (EBS) for its Spark cluster HDFS storage to receive and ingest data from remote sensors through Kafka. Consequently, the HDFS storage acts as the source for the backup data.

To fulfill these requirements, NetApp ONTAP Cloud is deployed in AWS, and an NFS share is created to act as the backup target for the Spark/Hadoop cluster.

After the NFS share is created, the In-Place Analytics Module is leveraged to copy the data from the HDFS EBS storage into the ONTAP NFS share. After the data resides in NFS in ONTAP Cloud, SnapMirror technology can be used to mirror the data from the cloud into on-premises storage as needed in a secure and efficient way.

This image shows the backup and disaster recovery from cloud to on-premises solution.



Next: Use case 3 - Enabling DevTest on existing Hadoop data.

Use case 3: Enabling DevTest on existing Hadoop data

Previous: Use case 2 - Backup and disaster recovery from the cloud to on-premises.

In this use case, the customer's requirement is to rapidly and efficiently build new Hadoop/Spark clusters based on an existing Hadoop cluster containing a large amount of analytics data for DevTest and reporting purposes in the same data center as well as remote locations.

Scenario

In this scenario, multiple Spark/Hadoop clusters are built from a large Hadoop data lake implementation on-premises as well as at disaster recovery locations.

Requirements and challenges

The main requirements and challenges for this use case include:

- Create multiple Hadoop clusters for DevTest, QA, or any other purpose that requires access to the same production data. The challenge here is to clone a very large Hadoop cluster multiple times instantaneously and in a very space-efficient manner.
- Sync the Hadoop data to DevTest and reporting teams for operational efficiency.
- Distribute the Hadoop data by using the same credentials across production and new clusters.

- Use scheduled policies to efficiently create QA clusters without affecting the production cluster.

Solution

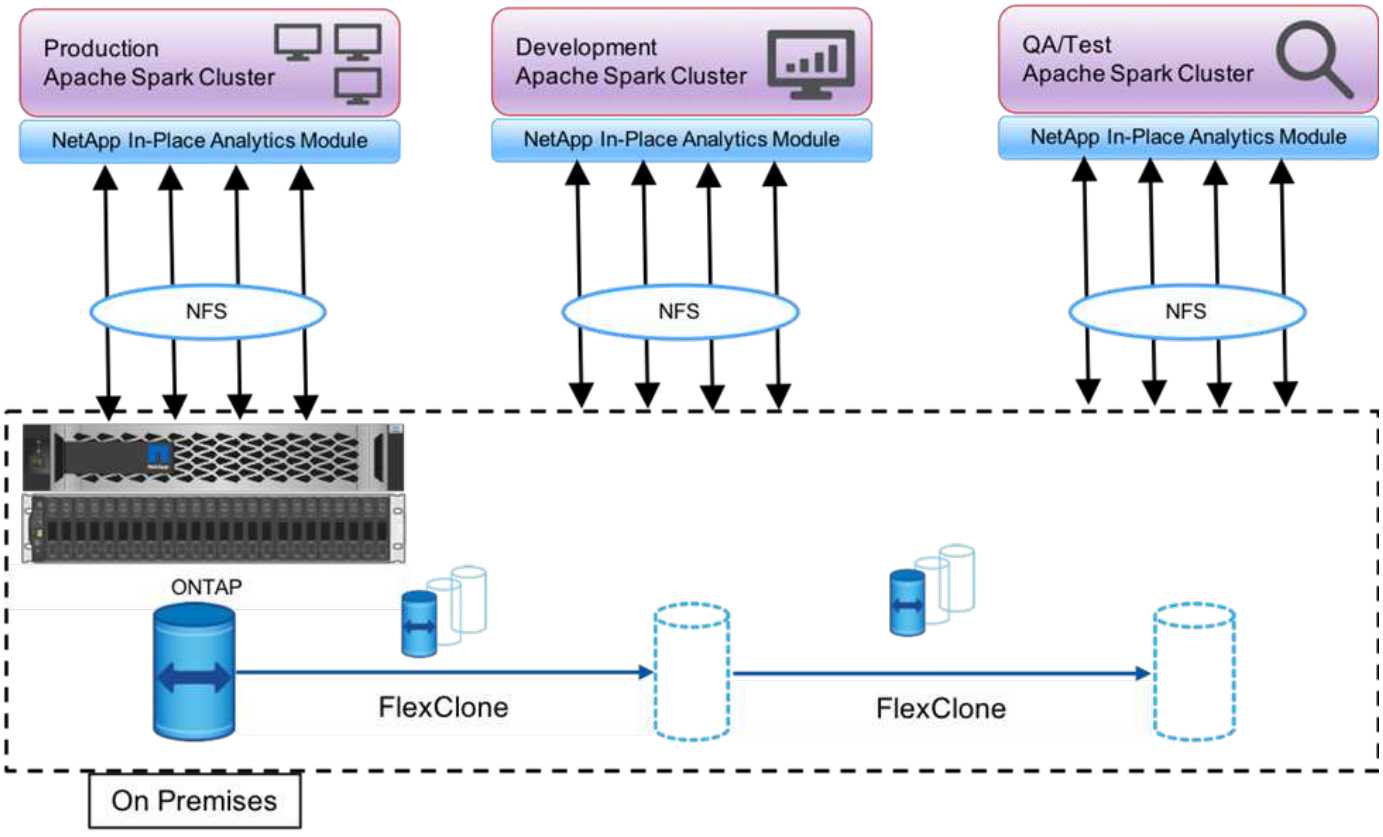
FlexClone technology is used to answer the requirements just described. FlexClone technology is the read/write copy of a Snapshot copy. It reads the data from parent Snapshot copy data and only consumes additional space for new/modified blocks. It is fast and space-efficient.

First, a Snapshot copy of the existing cluster was created by using a NetApp consistency group.

Snapshot copies within NetApp System Manager or the storage admin prompt. The consistency group Snapshot copies are application-consistent group Snapshot copies, and the FlexClone volume is created based on consistency group Snapshot copies. It is worth mentioning that a FlexClone volume inherits the parent volume's NFS export policy. After the Snapshot copy is created, a new Hadoop cluster must be installed for DevTest and reporting purposes, as shown in the figure below. The In-Place Analytics Module accesses the cloned NFS volume from the new Hadoop cluster through In-Place Analytics Module users and group authorization for the NFS data.

To have proper access, the new cluster must have the same UID and GUID for the users configured in the In-Place Analytics Module users and group configurations.

This image shows the Hadoop cluster for DevTest.



Next: Use case 4 - Data protection and multicloud connectivity.

Use case 4: Data protection and multicloud connectivity

Previous: Use case 3 - Enabling DevTest on existing Hadoop data.

This use case is relevant for a cloud service partner tasked with providing multicloud connectivity for customers' big data analytics data.

Scenario

In this scenario, IoT data received in AWS from different sources is stored in a central location in NPS. The NPS storage is connected to Spark/Hadoop clusters located in AWS and Azure enabling big data analytics applications running in multiple clouds accessing the same data.

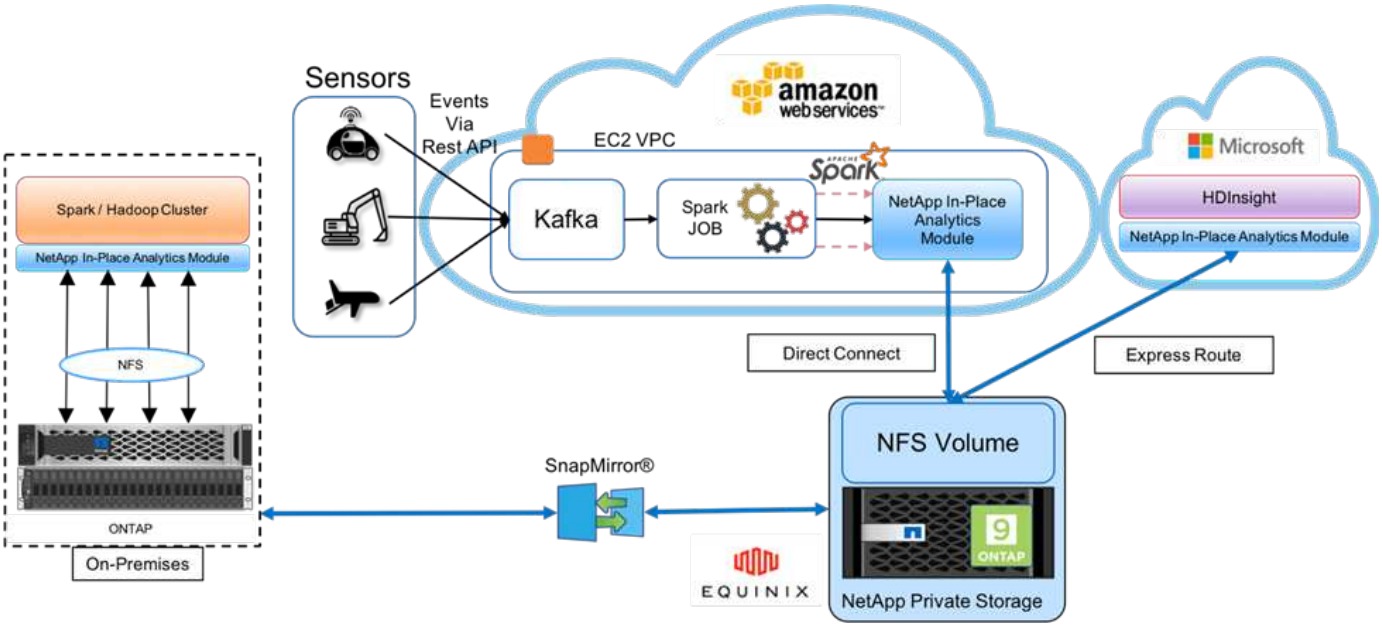
Requirements and challenges

The main requirements and challenges for this use case include:

- Customers want to run analytics jobs on the same data using multiple clouds.
- Data must be received from different sources such as on-premises and cloud through different sensors and hubs.
- The solution must be efficient and cost-effective.
- The main challenge is to build a cost-effective and efficient solution that delivers hybrid analytics services between on-premises and across different clouds.

Solution

This image illustrates the data protection and multicloud connectivity solution.



As shown in the figure above, data from sensors is streamed and ingested into the AWS Spark cluster through Kafka. The data is stored in an NFS share residing in NPS, which is located outside of the cloud provider within an Equinix data center. Because NetApp NPS is connected to Amazon AWS and Microsoft Azure through Direct Connect and Express Route connections, respectively, customers can leverage the In-Place Analytics Module to access the data from both Amazon and AWS analytics clusters. This approach solves having cloud analytics across multiple hyperscalers.

Consequently, because both on-premises and NPS storage runs ONTAP software, SnapMirror can mirror the NPS data into the on-premises cluster, providing hybrid cloud analytics across on-premises and multiple clouds.

For the best performance, NetApp typically recommends using multiple network interfaces and direct connection/express routes to access the data from cloud instances.

Next: Use case 5 - Accelerate analytic workloads.

Use case 5: Accelerate analytic workloads

Previous: Use case 4 - Data protection and multicloud connectivity.

In this scenario, a large financial services and investment bank's analytics platform was modernized using the NetApp NFS storage solution to achieve significant improvement in analyzing investment risks and derivatives for its asset management and quantitative business unit.

Scenario

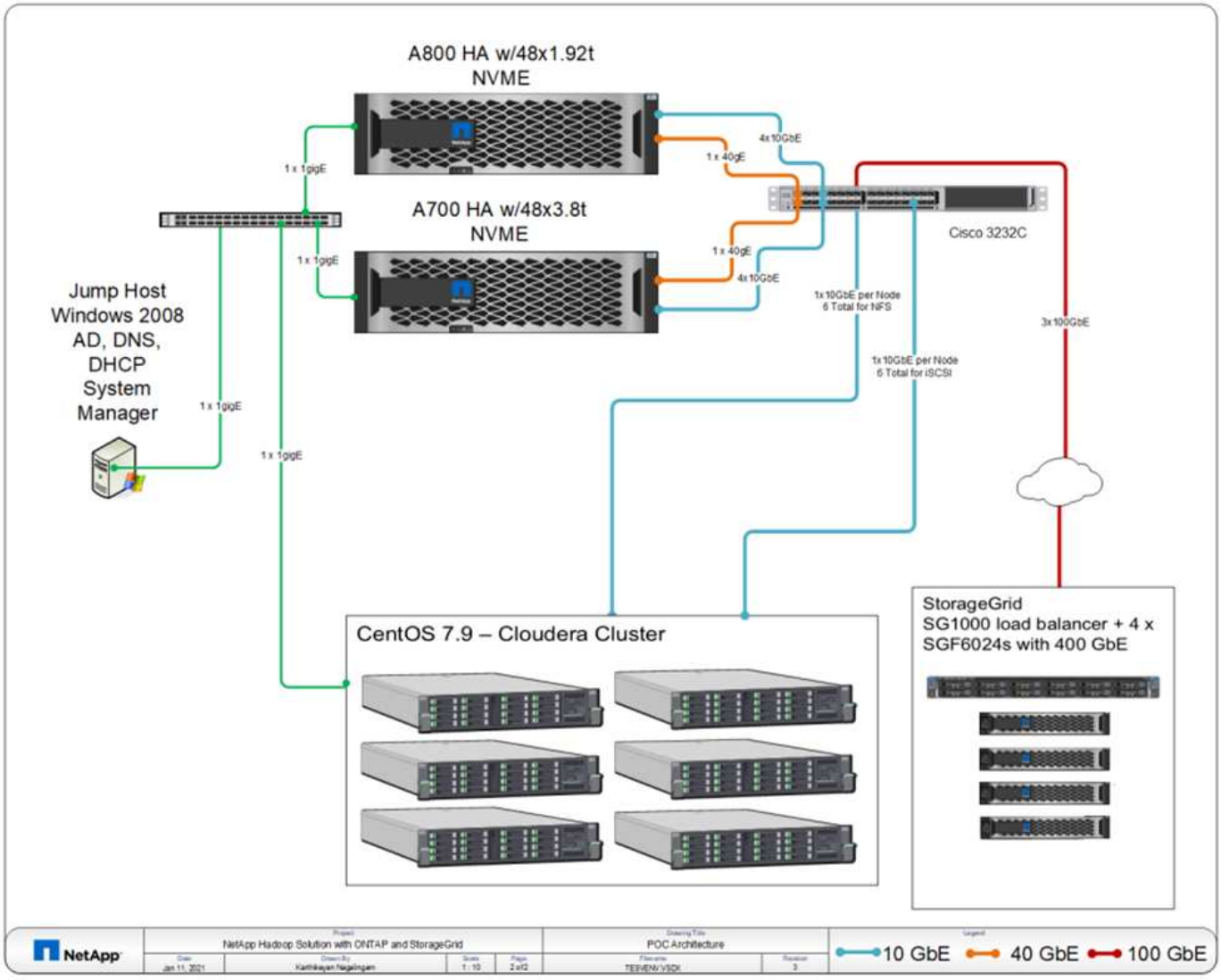
In the customer's existing environment, the Hadoop infrastructure used for the analytics platform leveraged internal storage from the Hadoop servers. Due to proprietary nature of JBOD environment, many internal customers within the organization were unable to take advantage of their Monte Carlo quantitative model, a simulation that relies on the recurring samples of real-time data. The suboptimal ability to understand the effects of uncertainty in market movements was serving unfavorably for the quantitative asset management business unit.

Requirements and challenges

The quantitative business unit at the bank wanted an efficient forecasting method to attain accurate and timely predictions. To do so, the team recognized the need to modernize the infrastructure, reduce existing I/O wait time and improve performance on the analytic applications such as Hadoop and Spark to efficiently simulate investment models, measure potential gains and analyze risks.

Solution

The customer had JBOD for their existing Spark solution. NetApp ONTAP, NetApp StorageGRID, and MinIO Gateway to NFS was then leveraged to reduce the I/O wait time for the bank's quantitative finance group that runs simulation and analysis on investment models that assess potential gains and risks. This image shows the Spark solution with NetApp storage.



As shown in figure above, AFF A800, A700 systems, and StorageGRID were deployed to access parquet files through NFS and S3 protocols in a six-node Hadoop cluster with Spark, and YARN and Hive metadata services for data analytic operations.

A direct-attached storage (DAS) solution in the customer's old environment had the disadvantage to scale compute and storage independently. With NetApp ONTAP solution for Spark, the bank's financial analytics business unit was able to decouple storage from compute and seamlessly bring infrastructure resources more effectively as needed.

By using ONTAP with NFS, the compute server CPUs were almost fully utilized for Spark SQL jobs and the I/O wait time was reduced by nearly 70%, therefore providing better compute power and performance boost to Spark workloads. Subsequently, increasing CPU utilization also enabled the customer to leverage GPUs, such as GPUDirect, for further platform modernization. Additionally, StorageGRID provides a low-cost storage option for Spark workloads and MinIO Gateway provides secure access to NFS data through the S3 protocol. For data in the cloud, NetApp recommends Cloud Volumes ONTAP, Azure NetApp Files, and NetApp Cloud Volumes Service.

Next: Conclusion.

Conclusion

Previous: Use case 5 - Accelerate analytic workloads.

This section provides a summary of the use cases and solutions provided by NetApp to fulfill various Hadoop data protection requirements. By using the data fabric powered by NetApp, customers can:

- Have the flexibility to choose the right data protection solutions by leveraging NetApp's rich data management capabilities and integration with Hadoop native workflows.
- Reduce their Hadoop cluster backup window time by almost 70%.
- Eliminate any performance effect resulting from Hadoop cluster backups.
- Provide multicloud data protection and data access from different cloud providers simultaneously to a single source of analytics data.
- Create fast and space-efficient Hadoop cluster copies by using FlexClone technology.

Where to find additional information

To learn more about the information described in this document, see the following documents and/or websites:

- NetApp Big Data Analytics Solutions

  https://www.netapp.com/us/solutions/applications/big-data-analytics/index.aspx

- Apache Spark Workload with NetApp Storage

  https://www.netapp.com/pdf.html?item=/media/26877-nva-1157-deploy.pdf

- NetApp Storage Solutions for Apache Spark

  https://www.netapp.com/media/16864-tr-4570.pdf

- Apache Hadoop on data fabric enabled by NetApp

  https://www.netapp.com/media/16877-tr-4529.pdf

- NetApp In-Place Analytics Module

  https://library.netapp.com/ecm/ecm_download_file/ECMLP2854071

Acknowledgements

- Paul Burland, Sales Rep, ANZ Victoria District Sales, NetApp
- Hoseb Dermanilian, Business Development Manager, NetApp
- Lee Dorrier, Director MPSG, NetApp
- David Thiessen, Systems Engineer, ANZ Victoria District SE, NetApp

Version history

Version	Date	Document version history
Version 1.0	January 2018	Initial release

Version	Date	Document version history
Version 2.0	October 2021	Updated with use case #5: Accelerate analytic workload