ALIBABA CLOUD

Alibaba Cloud

Container Service for Kubernetes

Solutions

Document Version: 20220414

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	**Danger:** Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	**Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	**Notice:** If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	**Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.Big data solutions

1.1. Run Apache Spark workloads on ACK

1.1.1. Overview

Apache Spark is an open source program that is widely used to analyze workloads in scenarios such as big data and machine learning. You can use Kubernetes to run and manage resources on Apache Spark 2.3.0 and later. This topic describes the benefits of running Apache Spark on Alibaba Cloud Container Service for Kubernetes (ACK).

Benefits

Kubernetes is an open source container management system that provides features such as the release, O&M, and scaling of applications. Container Service for Kubernetes (ACK) is a high-performance and scalable containerized application management service. It allows you to manage enterprise-level containerized applications throughout the lifecycle. ACK is one of the first Kubernetes platforms that obtain the certificate of conformance from Kubernetes. ACK simplifies the deployment and scale-out operations of Kubernetes clusters and integrates Alibaba Cloud capabilities of virtualization, storage, networking, and security. Based on these capabilities, ACK provides an ideal runtime environment for Kubernetes-based containerized applications. You can run a variety of workloads on ACK, including microservices, data processing in batches, and machine learning.

The following list describes the benefits of running Apache Spark workloads on ACK:

- By packaging a Spark application and its dependencies in a container, you reap the benefits of containers. For example, you do not have to consider whether your application is compatible with the Hadoop version, and you can attach tags to your container images to control the image versions. This helps you test different versions of Apache Spark or dependencies.
- You can reuse a variety of Kubernetes add-ons, such as the add-ons for monitoring or logging. By deploying Apache Spark workloads based on the existing Kubernetes infrastructure, you can save time and a large amount of O&M costs.
- ACK allows multiple tenants to run Apache Spark jobs in an ACK cluster. You can schedule resources for each tenant by using Kubernetes namespaces and resource quotas. ACK provides a mechanism to schedule applications to nodes. This ensures that dedicated resources are used to run each Apache Spark job. The Spark driver pod uses a Kubernetes service account to create executor pods. You can use the service account that is granted a Role or ClusterRole to define fine-grained access permissions. This secures your workloads against the impact of other workloads.
- ACK deploys Spark and data-centric applications that manage the lifecycle of your data in the same cluster. You can build an end-to-end lifecycle solution by using a single orchestrator and replicate the solution in other regions or even in private clouds.

Tools

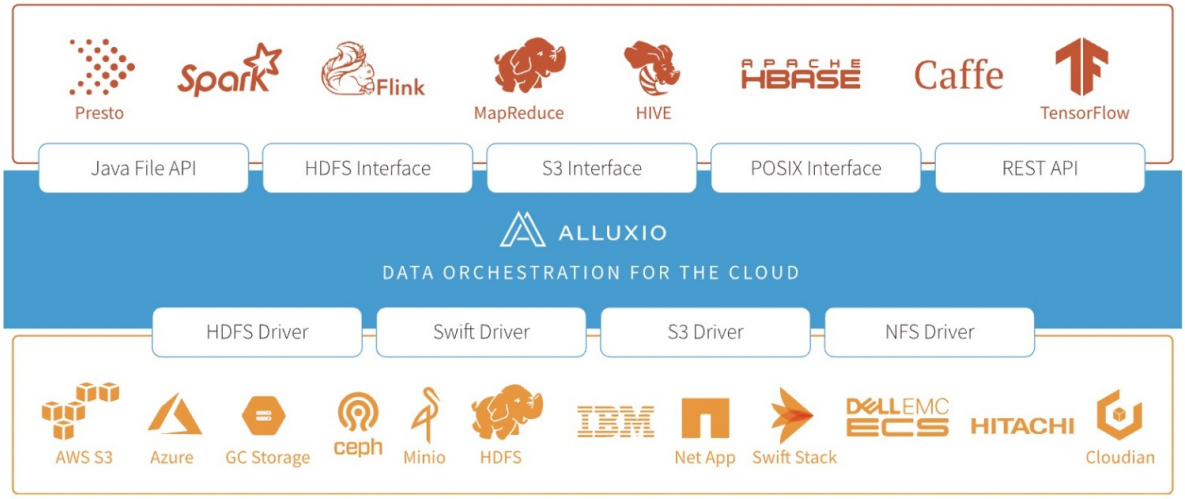
- **Spark on Kubernetes Operator**

  Spark on Kubernetes Operator allows you to run Spark workloads in the same way as you run other workloads on Kubernetes. It uses Kubernetes custom resources to specify, run, and display the running status of Spark jobs. Spark 2.3 or later is required to support resource scheduling in Kubernetes.

- **Alluxio**

Alluxio is a cloud-based open source data orchestrator that applies to data analysis and artificial intelligence applications. It migrates data from the storage layer of the system to a location that is more accessible for data-driven applications. It also allows applications to connect to multiple storage systems through a common interface. Alluxio stores application data in the memory of each node. This increases the data access speed by several orders of magnitude.

In the big data ecosystem, Alluxio is located between data-driven frameworks or applications and persistent storage systems. Data-driven frameworks or applications include Apache Spark, Presto, Tensorflow, Apache HBase, Apache Hive, and Apache Flink. Persistent storage systems include Amazon S3, Google Cloud Storage, OpenStack Swift, HDFS, GlusterFS, IBM clever, EMC ECS, Ceph, NFS, Minio, and Alibaba OSS. Alluxio unifies the data stored in these storage systems and provides a unified client API and global namespaces for the data-driven applications that are built based on these storage systems.



In the Spark on ACK big data solution, data in the persistent storage systems is cached in Alluxio. This accelerates the Spark access to the data.

- **TPC-DS Benchmark**

  The TPC Benchmark DS (TPC-DS) is a performance testing and decision support benchmark developed and maintained by a third-party community. The TPC-DS test set involves various functions, such as data collection, report generation, online queries, and data mining based on large data sets. Data in real-life scenarios is collected in the test set. Therefore, an application must have high performance to pass the tests based on the test set.

  TPC-DS contains 104 queries that cover most SQL statements in the SQL:2003 standard. TPC-DS contains 99 stress testing queries. Each of the following queries has two variants: 14, 23, 24, and 39. The `s_max` query performs a full scan and aggregates large tables.

  TPC-DS has the following features:

  - Follows the SQL:2003 standard and provides complex SQL query cases.
  - Requires analysis of a large amount of data and answers to real business questions.
  - Provides test cases that include various business models, such as analysis report models, iterative OLAP models, and data mining models.
  - Requires high I/O load and high CPU utilization in almost all test cases.

  Therefore, the Spark on ACK big data solution uses TPC-DS to evaluate the performance of Spark on ACK.

Related information

- Set up a test environment
- Write test code
- Run Spark benchmarks on ACK
- Analyze test results
- Troubleshooting

1.1.2. Set up a test environment

You can run Spark jobs in the Container Service for Kubernetes (ACK) console and use Alluxio to accelerate data processing in a distributed manner. ACK provides Spark Operator to simplify the procedure of submitting Spark jobs. ACK also provides Spark History Server to record historical data of Spark jobs. This facilitates troubleshooting. This topic describes how to set up an environment in the ACK console to run Spark jobs.

Context

To run Spark jobs in the ACK console, you must perform the following operations:

- Create an ACK cluster
- Create an OSS bucket
- Install ack-spark-operator
- Install ack-spark-history-server
- Install Alluxio

Create an ACK cluster

For more information about how to create an ACK cluster, see Create an ACK managed cluster.

> **Notice**
>
> Take note of the following information when you set the cluster parameters:
>
> - When you set the instance type of worker nodes, select **ecs.d1ne.6xlarge** in the **Big Data Network Performance Enhanced** instance family and set the number of worker nodes to 20.
> - Each worker node of the **ecs.d1ne.6xlarge** instance type has 12 HDDs. Each HDD has a storage capacity of 5 TB. Before you mount the HDDs, you must partition and format them. For more information, see Partition and format a data disk larger than 2 TiB in size.
> - After you partition and format the HDDs, mount them to the ACK cluster. You can run the `df -h` command to query the mount information of the HDDs. The Mount information figure shows an example of the command output.
> - The 12 file paths under the */mnt* directory are used in the configuration file of Alluxio. ACK provides a simplified method to mount data disks when the cluster has a large number of nodes. For more information, see Use LVM to manage local storage.

Mount information

```
/dev/vdb1       5.4T   89M  5.1T    1% /mnt/disk1
/dev/vdc1       5.4T   89M  5.1T    1% /mnt/disk2
/dev/vdd1       5.4T   89M  5.1T    1% /mnt/disk3
/dev/vde1       5.4T   89M  5.1T    1% /mnt/disk4
/dev/vdf1       5.4T   89M  5.1T    1% /mnt/disk5
/dev/vdg1       5.4T   89M  5.1T    1% /mnt/disk6
/dev/vdh1       5.4T   89M  5.1T    1% /mnt/disk7
/dev/vdi1       5.4T   89M  5.1T    1% /mnt/disk8
/dev/vdj1       5.4T   89M  5.1T    1% /mnt/disk9
/dev/vdk1       5.4T   89M  5.1T    1% /mnt/disk10
/dev/vdl1       5.4T   89M  5.1T    1% /mnt/disk11
/dev/vdm1       5.4T   89M  5.1T    1% /mnt/disk12
```

Create an OSS bucket

You must create an Object Storage Service (OSS) bucket to store data, including the test data generated by TPC-DS, test results, and test logs. In this example, the name of the OSS bucket is *cloudnativeai*. For more information about how to create an OSS bucket, see Create buckets.

Install ack-spark-operator

You can install **ack-spark-operator** and use the component to simplify the procedure of submitting Spark jobs.

1.
2.
3. On the **Marketplace** page, click the **App Catalog** tab. Find and click **ack-spark-operator**.
4. On the **ack-spark-operator** page, click **Deploy**.
5. In the **Deploy** wizard, select a cluster and namespace, and then click **Next**.
6. On the **Parameters** wizard page, set the parameters and click **OK**.

Install ack-spark-history-server

ack-spark-history-server generates logs and events for Spark jobs and provides a user interface to help you troubleshoot issues.

When you install **ack-spark-history-server**, you must specify parameters related to the OSS bucket on the **Parameters** wizard page. The OSS bucket is used to store historical data of Spark jobs. For more information about how to install **ack-spark-history-server**, see Install ack-spark-operator.

The following code block shows the parameters related to the OSS bucket:

```
oss:
  enableOSS: true
  # Please input your accessKeyId
  alibabaCloudAccessKeyId: ""
  # Please input your accessKeySecret
  alibabaCloudAccessKeySecret: ""
  # oss bucket endpoint such as oss-cn-beijing.aliyuncs.com
  alibabaCloudOSSEndpoint: ""
  # oss file path such as oss://bucket-name/path
  eventsDir: "oss://cloudnativeai/spark/spark-events"
```

Run the following command to check whether ack-spark-history-server is installed:

```
kubectl get service ack-spark-history-server -n {YOUR-NAMESPACE}
```

Install Alluxio

You must run the **Helm** command to install Alluxio in the ACK console.

1. Run the following command to download the installation file of Alluxio:

   ```
   wget http://kubeflow.oss-cn-beijing.aliyuncs.com/alluxio-0.6.8.tgz
   tar -xvf alluxio-0.6.8.tgz
   ```

2. Create and configure a file named *config.yaml* in the directory where the installation file of Alluxio is saved.

   For more information about how to configure the file, see config.yaml.

   The following code block shows the key parameters.

   - Modify the following parameters based on the information of the OSS bucket: AccessKey ID, AccessKey secret, the endpoint of the OSS bucket, and UNIX File System (UFS).

     ```
     # Site properties for all the components
     properties:
       fs.oss.accessKeyId: YOUR-ACCESS-KEY-ID
       fs.oss.accessKeySecret: YOUR-ACCESS-KEY-SECRET
       fs.oss.endpoint: oss-cn-beijing-internal.aliyuncs.com
       alluxio.master.mount.table.root.ufs: oss://cloudnativeai/
       alluxio.master.persistence.blacklist: .staging,_temporary
       alluxio.security.stale.channel.purge.interval: 365d
       alluxio.user.metrics.collection.enabled: 'true'
       alluxio.user.short.circuit.enabled: 'true'
       alluxio.user.file.write.tier.default: 1
       alluxio.user.block.size.bytes.default: 64MB #default 64MB
       alluxio.user.file.writetype.default: CACHE_THROUGH
       alluxio.user.file.metadata.load.type: ONCE
       alluxio.user.file.readtype.default: CACHE
       #alluxio.worker.allocator.class: alluxio.worker.block.allocator.MaxFreeAllocator
       alluxio.worker.allocator.class: alluxio.worker.block.allocator.RoundRobinAllocator
       alluxio.worker.file.buffer.size: 128MB
       alluxio.worker.evictor.class: alluxio.worker.block.evictor.LRUEvictor
       alluxio.job.master.client.threads: 5000
       alluxio.job.worker.threadpool.size: 300
     ```

   - In the tieredstore section, mediumtype specifies the IDs of the data disks on a worker node, and path specifies the paths where the data disks are mounted.

```
tieredstore:
  levels:
    - level: 0
      alias: HDD
      mediumtype: HDD-0,HDD-1,HDD-2,HDD-3,HDD-4,HDD-5,HDD-6,HDD-7,HDD-8,HDD-9,HDD-10,
HDD-11
      path: /mnt/disk1,/mnt/disk2,/mnt/disk3,/mnt/disk4,/mnt/disk5,/mnt/disk6,/mnt/di
sk7,/mnt/disk8,/mnt/disk9,/mnt/disk10,/mnt/disk11,/mnt/disk12
      type: hostPath
      quota: 1024G,1024G,1024G,1024G,1024G,1024G,1024G,1024G,1024G,1024G,1024G,1024G
      high: 0.95
      low: 0.7
```

3. Add the alluxio=true label to the worker nodes of the ACK cluster.

   For more information about how to add node labels, see Manage node labels.

4. Run the following **Helm** command to install Alluxio:

```
kubectl create namespace alluxio
helm install -f config.yaml -n alluxio alluxio alluxio
```

5. Run the following command to check whether Alluxio is installed:

```
kubectl get pod -n alluxio
```

6. Run the following command as the Alluxio admin to check whether the disks are mounted to worker nodes of the cluster:

```
kubectl exec -it alluxio-master-0 -n alluxio -- /bin/bash
./bin/alluxio fsadmin report capacity
```

   If disks are mounted to each worker node, Alluxio is installed.

```
bash-4.4# ./bin/alluxio fsadmin report capacity
Capacity information for all workers:
    Total Capacity: 240.00TB
        Tier: HDD  Size: 240.00TB
    Used Capacity: 647.96GB
        Tier: HDD  Size: 647.96GB
    Used Percentage: 0%
    Free Percentage: 100%

Worker Name      Last Heartbeat   Storage       HDD
192.[redacted]   0                capacity      12.00TB
                                  used          29.75GB (0%)
192.[redacted]   0                capacity      12.00TB
                                  used          34.68GB (0%)
192.[redacted]   0                capacity      12.00TB
                                  used          33.47GB (0%)
192.[redacted]   0                capacity      12.00TB
                                  used          32.86GB (0%)
192.[redacted]   0                capacity      12.00TB
                                  used          34.19GB (0%)
192.[redacted]   0                capacity      12.00TB
                                  used          29.03GB (0%)
192.[redacted]   0                capacity      12.00TB
                                  used          30.37GB (0%)
192.[redacted]   0                capacity      12.00TB
                                  used          32.65GB (0%)
192.[redacted]   0                capacity      12.00TB
                                  used          32.45GB (0%)
192.[redacted]   0                capacity      12.00TB
                                  used          32.53GB (0%)
192.[redacted]   0                capacity      12.00TB
                                  used          31.99GB (0%)
192.[redacted]   0                capacity      12.00TB
                                  used          33.07GB (0%)
192.[redacted]   0                capacity      12.00TB
                                  used          32.92GB (0%)
192.[redacted]   0                capacity      12.00TB
                                  used          30.49GB (0%)
192.[redacted]   0                capacity      12.00TB
                                  used          30.93GB (0%)
192.[redacted]   0                capacity      12.00TB
                                  used          30.32GB (0%)
192.[redacted]   0                capacity      12.00TB
                                  used          33.72GB (0%)
192.[redacted]   0                capacity      12.00TB
                                  used          33.40GB (0%)
192.[redacted]   0                capacity      12.00TB
                                  used          34.31GB (0%)
192.[redacted]   0                capacity      12.00TB
                                  used          34.85GB (0%)
```

What's next

Write test code

1.1.3. Write test code

Before you use the benchmarks to test the performance of Spark, write the code to generate test data based on TPC-DS and the code to run Spark SQL. This topic describes how to write the code and create an image.

Prerequisites

Set up a test environment

Context

The registry.cn-beijing.aliyuncs.com/yukong/ack-spark-benchmark:1.0.0 image is prepared. You can use it to simplify the testing procedure. For information about the code of the image, see benchmark-for-spark.

Before you begin

The test code is written based on two Databricks-based tools: the TPC-DS test package and tpcds-kit. tpcds-kit generates test data.

1. Run the following command to put the JAR dependencies of TPC-DS into a package:

```
git clone https://github.com/databricks/spark-sql-perf.git
sbt package
```

   The *spark-sql-perf_2.11-0.5.1-SNAPSHOT* JAR package is generated. You can use it as the dependency of the test.

2. Run the following commands to compile tpcds-kit:

```
git clone https://github.com/davies/tpcds-kit.git
yum install gcc gcc-c++ bison flex cmake ncurses-devel
cd tpcds-kit/tools
cp Makefile.suite Makefile# Replicate the Makefile.suite file and save the replica as t
he Makefile file.
make
# Verify the result of the compilation.
./dsqgen --help
```

   After you compile the code, a binary executable program is generated. In this topic, dsdgen is used to generate test data, and dsqgen is used to generate queries.

Write code

1. To generate test data, create and configure the *DataGeneration.scala* file by running the following code:

```
package com.aliyun.spark.benchmark.tpcds
import com.databricks.spark.sql.perf.tpcds.TPCDSTables
import org.apache.log4j.{ Level, LogManager}
import org.apache.spark.sql.SparkSession
import scala.util.Try
object DataGeneration {
  def main(args: Array[String]) {
    val tpcdsDataDir = args(0)
    val dsdgenDir = args(1)
    val format = Try(args(2).toString).getOrElse("parquet")
    val scaleFactor = Try(args(3).toString).getOrElse("1")
    val genPartitions = Try(args(4).toInt).getOrElse(100)
    val partitionTables = Try(args(5).toBoolean).getOrElse(false)
    val clusterByPartitionColumns = Try(args(6).toBoolean).getOrElse(false)
    val onlyWarn = Try(args(7).toBoolean).getOrElse(false)
    println(s"DATA DIR is $tpcdsDataDir")
    println(s"Tools dsdgen executable located in $dsdgenDir")
    println(s"Scale factor is $scaleFactor GB")
    val spark = SparkSession
      .builder
      .appName(s"TPCDS Generate Data $scaleFactor GB")
      .getOrCreate()
    if (onlyWarn) {
      println(s"Only WARN")
      LogManager.getLogger("org").setLevel(Level.WARN)
    }
    val tables = new TPCDSTables(spark.sqlContext,
      dsdgenDir = dsdgenDir,
      scaleFactor = scaleFactor,
      useDoubleForDecimal = false,
      useStringForDate = false)
    println(s"Generating TPCDS data")
    tables.genData(
      location = tpcdsDataDir,
      format = format,
      overwrite = true, // overwrite the data that is already there
      partitionTables = partitionTables,  // create the partitioned fact tables
      clusterByPartitionColumns = clusterByPartitionColumns, // shuffle to get partitio
ns coalesced into single files.
      filterOutNullPartitionValues = false, // true to filter out the partition with NU
LL key value
      tableFilter = "", // "" means generate all tables
      numPartitions = genPartitions) // how many dsdgen partitions to run - number of i
nput tasks.
    println(s"Data generated at $tpcdsDataDir")
    spark.stop()
  }
}
```

2. To query data, create and configure the *BenchmarkSQL.scala* file by running the following code:

```
package com.aliyun.spark.benchmark.tpcds
import com.databricks.spark.sql.perf.tpcds.{ TPCDS, TPCDSTables}
import org.apache.spark.sql.SparkSession
```

```
import org.apache.spark.sql.functions._
import org.apache.spark.sql.functions.col
import org.apache.log4j.{ Level, LogManager}
import scala.util.Try
object BenchmarkSQL {
  def main(args: Array[String]) {
    val tpcdsDataDir = args(0)
    val resultLocation = args(1)
    val dsdgenDir = args(2)
    val format = Try(args(3).toString).getOrElse("parquet")
    val scaleFactor = Try(args(4).toString).getOrElse("1")
    val iterations = args(5).toInt
    val optimizeQueries = Try(args(6).toBoolean).getOrElse(false)
    val filterQueries = Try(args(7).toString).getOrElse("")
    val onlyWarn = Try(args(8).toBoolean).getOrElse(false)
    val databaseName = "tpcds_db"
    val timeout = 24*60*60
    println(s"DATA DIR is $tpcdsDataDir")
    val spark = SparkSession
      .builder
      .appName(s"TPCDS SQL Benchmark $scaleFactor GB")
      .getOrCreate()
    if (onlyWarn) {
      println(s"Only WARN")
      LogManager.getLogger("org").setLevel(Level.WARN)
    }
    val tables = new TPCDSTables(spark.sqlContext,
      dsdgenDir = dsdgenDir,
      scaleFactor = scaleFactor,
      useDoubleForDecimal = false,
      useStringForDate = false)
    if (optimizeQueries) {
      Try {
        spark.sql(s"create database $databaseName")
      }
      tables.createExternalTables(tpcdsDataDir, format, databaseName, overwrite = true,
discoverPartitions = true)
      tables.analyzeTables(databaseName, analyzeColumns = true)
      spark.conf.set("spark.sql.cbo.enabled", "true")
    } else {
      tables.createTemporaryTables(tpcdsDataDir, format)
    }
    val tpcds = new TPCDS(spark.sqlContext)
    var query_filter : Seq[String] = Seq()
    if (! filterQueries.isEmpty) {
      println(s"Running only queries: $filterQueries")
      query_filter = filterQueries.split(",").toSeq
    }
    val filtered_queries = query_filter match {
      case Seq() => tpcds.tpcds2_4Queries
      case _ => tpcds.tpcds2_4Queries.filter(q => query_filter.contains(q.name))
    }
    // Start experiment
    val experiment = tpcds.runExperiment(
      filtered_queries
```

```
      filtered_queries,
      iterations = iterations,
      resultLocation = resultLocation,
      forkThread = true)
    experiment.waitForFinish(timeout)
    // Collect general results
    val resultPath = experiment.resultPath
    println(s"Reading result at $resultPath")
    val specificResultTable = spark.read.json(resultPath)
    specificResultTable.show()
    // Summarize results
    val result = specificResultTable
      .withColumn("result", explode(col("results")))
      .withColumn("executionSeconds", col("result.executionTime")/1000)
      .withColumn("queryName", col("result.name"))
    result.select("iteration", "queryName", "executionSeconds").show()
    println(s"Final results at $resultPath")
    val aggResults = result.groupBy("queryName").agg(
      callUDF("percentile", col("executionSeconds").cast("long"), lit(0.5)).as('medianR
untimeSeconds),
      callUDF("min", col("executionSeconds").cast("long")).as('minRuntimeSeconds),
      callUDF("max", col("executionSeconds").cast("long")).as('maxRuntimeSeconds)
    ).orderBy(col("queryName"))
    aggResults.repartition(1).write.csv(s"$resultPath/summary.csv")
    aggResults.show(105)
    spark.stop()
  }
}
```

Create an image

After you compile the test code into a JAR package, you can create a test image by combining this JAR package with other JAR dependencies of the test. You can use the following dockerfile:

```
FROM registry.cn-hangzhou.aliyuncs.com/acs/spark:ack-2.4.5-f757ab6
RUN mkdir -p /opt/spark/jars
RUN mkdir -p /tmp/tpcds-kit
COPY ./target/scala-2.11/ack-spark-benchmark-assembly-0.1.jar /opt/spark/jars/
COPY ./lib/*.jar /opt/spark/jars/
COPY ./tpcds-kit/tools.tar.gz /tmp/tpcds-kit/
RUN cd /tmp/tpcds-kit/ && tar -xzvf tools.tar.gz
```

What's next

Run Spark benchmarks on ACK

1.1.4. Run Spark benchmarks on ACK

After the test environment and test image are ready, you can run Spark benchmarks on ACK. This topic describes how to generate test data, run benchmarks based on the test data, and accelerate queries by using the Alluxio distributed cache.

Prerequisites

Write test code

Test description:

In this test, 1 TB of data is generated to test the performance of Spark on ACK in the following scenarios: Spark reads data from OSS. Spark uses the Alluxio cold cache to accelerate data access. Spark uses the Alluxio warm cache to accelerate data access.

Generate 1 TB of test data

Run the DataGeneration.scala command to generate 1 TB of data, and store the data in OSS for Spark SQL queries. Perform the following steps:

1. Use the following template to create the *tpcds-data-generator.yaml* file:

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: tpcds-data-generation
  namespace: default
spec:
  type: Scala
  image: registry.cn-beijing.aliyuncs.com/kube-ai/ack-spark-benchmark:1.0.1
  sparkVersion: 2.4.5
  mainClass: com.aliyun.spark.benchmark.tpcds.DataGeneration
  mainApplicationFile: "local:///opt/spark/jars/ack-spark-benchmark-assembly-0.1.jar"
  mode: cluster
  arguments:
    # TPC-DS data localtion
    - "oss://cloudnativeai/spark/data/tpc-ds-data/1000g"
    # Path to kit in the docker image
    - "/tmp/tpcds-kit/tools"
    # Data Format
    - "parquet"
    # Scale factor (in GB)
    - "1000"
    # Generate data num partitions
    - "100"
    # Create the partitioned fact tables
    - "false"
    # Shuffle to get partitions coalesced into single files.
    - "false"
    # Logging set to WARN
    - "true"
  hadoopConf:
    # OSS
    "fs.oss.impl": "org.apache.hadoop.fs.aliyun.oss.AliyunOSSFileSystem"
    "fs.oss.endpoint": "oss-cn-beijing-internal.aliyuncs.com"
    "fs.oss.accessKeyId": "YOUR-ACCESS-KEY-ID"
    "fs.oss.accessKeySecret": "YOUR-ACCESS-KEY-SECRET"
  sparkConf:
    "spark.kubernetes.allocation.batch.size": "100"
    "spark.sql.adaptive.enabled": "true"
    "spark.eventLog.enabled": "true"
    "spark.eventLog.dir": "oss://cloudnativeai/spark/spark-events"
  driver:
    cores: 6
    memory: "20480m"
    serviceAccount: spark
  executor:
    instances: 20
    cores: 8
    memory: "61440m"
    memoryOverhead: 2g
  restartPolicy:
    type: Never
```

2. Run the following command to generate data:

```
kubectl apply -f tpcds-data-generator.yaml
```

Run the benchmarks

Query data for three times:

1. Spark reads 1 TB of data from OSS.
2. Alluxio caches data from OSS in a distributed manner. Spark reads data from the Alluxio cold cache.
3. After you modify the OSS path where the data is stored in the preceding scenario, Spark reads data from the Alluxio warm cache.

The test results show that the Alluxio cache improves the query performance. The following content describes the procedure:

1. Run the benchmark in the scenario where Spark reads data from OSS.
   i. Use the following template to create and deploy the *tpcds-benchmark.yaml* file:

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: tpcds-benchmark
  namespace: default
spec:
  type: Scala
  mode: cluster
  image: registry.cn-beijing.aliyuncs.com/kube-ai/ack-spark-benchmark:1.0.1
  imagePullPolicy: Always
  sparkVersion: 2.4.5
  mainClass: com.aliyun.spark.benchmark.tpcds.BenchmarkSQL
  mainApplicationFile: "local:///opt/spark/jars/ack-spark-benchmark-assembly-0.1.ja
r"
  arguments:
    # TPC-DS data localtion
    - "oss://cloudnativeai/spark/data/tpc-ds-data/1000g"
    # results location
    - "oss://cloudnativeai/spark/result/tpcds-benchmark-result-1000g"
    # Path to kit in the docker image
    - "/tmp/tpcds-kit/tools"
    # Data Format
    - "parquet"
    # Scale factor (in GB)
    - "1000"
    # Number of iterations
    - "1"
    # Optimize queries
    - "false"
    # Filter queries, will run all if empty - "q70-v2.4,q82-v2.4,q64-v2.4"
    - ""
    # Logging set to WARN
    - "true"
  hostNetwork: true
  dnsPolicy: ClusterFirstWithHostNet
  restartPolicy:
    type: Never
```

```
  timeToLiveSeconds: 86400
  hadoopConf:
    # OSS
    "fs.oss.impl": "org.apache.hadoop.fs.aliyun.oss.AliyunOSSFileSystem"
    "fs.oss.endpoint": "oss-cn-beijing-internal.aliyuncs.com"
    "fs.oss.accessKeyId": "YOUR-ACCESS-KEY-ID"
    "fs.oss.accessKeySecret": "YOUR-ACCESS-KEY-SECRET"
  sparkConf:
    "spark.kubernetes.allocation.batch.size": "200"
    "spark.sql.adaptive.join.enabled": "true"
    "spark.eventLog.enabled": "true"
    "spark.eventLog.dir": "oss://cloudnativeai/spark/spark-events"
  driver:
    cores: 5
    memory: "20480m"
    labels:
      version: 2.4.5
      spark-app: spark-tpcds
      role: driver
    serviceAccount: spark
  executor:
    cores: 7
    instances: 20
    memory: "20480m"
    memoryOverhead: "8g"
    labels:
      version: 2.4.5
      role: executor
```

ii. Run the following command to run the benchmark:

```
kubectl apply -f tpcds-benchmark.yaml
```

2. Run the benchmark in the scenario where Spark reads data from the Alluxio cold cache.

i. Use the following template to create and deploy the *tpcds-benchmark-with-alluxio.yaml* file:

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: tpcds-benchmark-sql
  namespace: default
spec:
  type: Scala
  mode: cluster
  image: registry.cn-beijing.aliyuncs.com/kube-ai/ack-spark-benchmark:1.0.1
  imagePullPolicy: Always
  sparkVersion: 2.4.5
  mainClass: com.aliyun.spark.benchmark.tpcds.BenchmarkSQL
  mainApplicationFile: "local:///opt/spark/jars/ack-spark-benchmark-assembly-0.1.ja
r"
  arguments:
    # TPC-DS data localtion
    - "alluxio://alluxio-master-0.alluxio.svc.cluster.local:19998/spark/data/tpc-ds
-data/1000g"
    # results location
```

```
    - "oss://cloudnativeai/spark/result/tpcds-benchmark-result-1000g-alluxio"
    # Path to kit in the docker image
    - "/tmp/tpcds-kit/tools"
    # Data Format
    - "parquet"
    # Scale factor (in GB)
    - "1000"
    # Number of iterations
    - "1"
    # Optimize queries
    - "false"
    # Filter queries, will run all if empty - "q70-v2.4,q82-v2.4,q64-v2.4"
    - ""
    # Logging set to WARN
    - "true"
  hostNetwork: true
  dnsPolicy: ClusterFirstWithHostNet
  restartPolicy:
    type: Never
  timeToLiveSeconds: 86400
  hadoopConf:
    # OSS
    "fs.oss.impl": "org.apache.hadoop.fs.aliyun.oss.AliyunOSSFileSystem"
    "fs.oss.endpoint": "oss-cn-beijing-internal.aliyuncs.com"
    "fs.oss.accessKeyId": "YOUR-ACCESS-KEY-ID"
    "fs.oss.accessKeySecret": "YOUR-ACCESS-KEY-SECRET"
  sparkConf:
    "spark.kubernetes.allocation.batch.size": "200"
    "spark.sql.adaptive.join.enabled": "true"
    "spark.eventLog.enabled": "true"
    "spark.eventLog.dir": "oss://cloudnativeai/spark/spark-events"
  volumes:
    - name: "spark-local-dir-1"
      hostPath:
        path: "/mnt/disk1"
        type: Directory
    - name: "spark-local-dir-2"
      hostPath:
        path: "/mnt/disk2"
        type: Directory
    - name: "spark-local-dir-3"
      hostPath:
        path: "/mnt/disk3"
        type: Directory
    - name: "spark-local-dir-4"
      hostPath:
        path: "/mnt/disk4"
        type: Directory
    - name: "spark-local-dir-5"
      hostPath:
        path: "/mnt/disk5"
        type: Directory
    - name: "spark-local-dir-6"
      hostPath:
```

```
        path: "/mnt/disk6"
        type: Directory
    - name: "spark-local-dir-7"
      hostPath:
        path: "/mnt/disk7"
        type: Directory
    - name: "spark-local-dir-8"
      hostPath:
        path: "/mnt/disk8"
        type: Directory
    - name: "spark-local-dir-9"
      hostPath:
        path: "/mnt/disk9"
        type: Directory
    - name: "spark-local-dir-10"
      hostPath:
        path: "/mnt/disk10"
        type: Directory
    - name: "spark-local-dir-11"
      hostPath:
        path: "/mnt/disk11"
        type: Directory
    - name: "spark-local-dir-12"
      hostPath:
        path: "/mnt/disk12"
        type: Directory
  driver:
    cores: 5
    memory: "20480m"
    labels:
      version: 2.4.5
      spark-app: spark-tpcds
      role: driver
    serviceAccount: spark
  executor:
    cores: 7
    instances: 20
    memory: "20480m"
    memoryOverhead: "8g"
    labels:
      version: 2.4.5
      role: executor
    volumeMounts:
      - name: "spark-local-dir-1"
        mountPath: "/mnt/disk1"
      - name: "spark-local-dir-2"
        mountPath: "/mnt/disk2"
      - name: "spark-local-dir-3"
        mountPath: "/mnt/disk3"
      - name: "spark-local-dir-4"
        mountPath: "/mnt/disk4"
      - name: "spark-local-dir-5"
        mountPath: "/mnt/disk5"
      - name: "spark-local-dir-6"
```

```
        mountPath: "/mnt/disk6"
      - name: "spark-local-dir-7"
        mountPath: "/mnt/disk7"
      - name: "spark-local-dir-8"
        mountPath: "/mnt/disk8"
      - name: "spark-local-dir-9"
        mountPath: "/mnt/disk9"
      - name: "spark-local-dir-10"
        mountPath: "/mnt/disk10"
      - name: "spark-local-dir-11"
        mountPath: "/mnt/disk11"
      - name: "spark-local-dir-12"
        mountPath: "/mnt/disk12"
```

ii. Run the following command to run the benchmark:

```
kubectl apply -f tpcds-benchmark-with-alluxio.yaml
```

3. Run the benchmark in the scenario where Spark reads data from the Alluxio warm cache.

   When you use Alluxio to accelerate data access for the first time, Spark reads data from OSS and caches the data in Alluxio. In this case, the read speed is slow. Therefore, we recommend that you run more benchmarks after the preceding step. In these tests, Spark reads data from the Alluxio hot cache.

What's next

Analyze test results

1.1.5. Analyze test results

This topic compares the performance of Container Service for Kubernetes (ACK)-based Spark SQL queries on 1 TB of data before and after the Alluxio distributed caching service is used.

Prerequisites

Run Spark benchmarks on ACK

Hardware configurations

The following table lists the ACK cluster configurations.

Cluster type	Standard dedicated cluster
ECS instances	• Instance type: ecs.d1ne.6xlarge • Alibaba Cloud Linux 2.1903 • CPU: 24 cores • Memory: 96 GB • Disk size: 5,500 GB. Disk type: HDD.
Number of worker nodes	20

Software configurations

- Version
  - Apache Spark: 2.4.5
  - Alluxio: 2.3.0
- Spark configurations

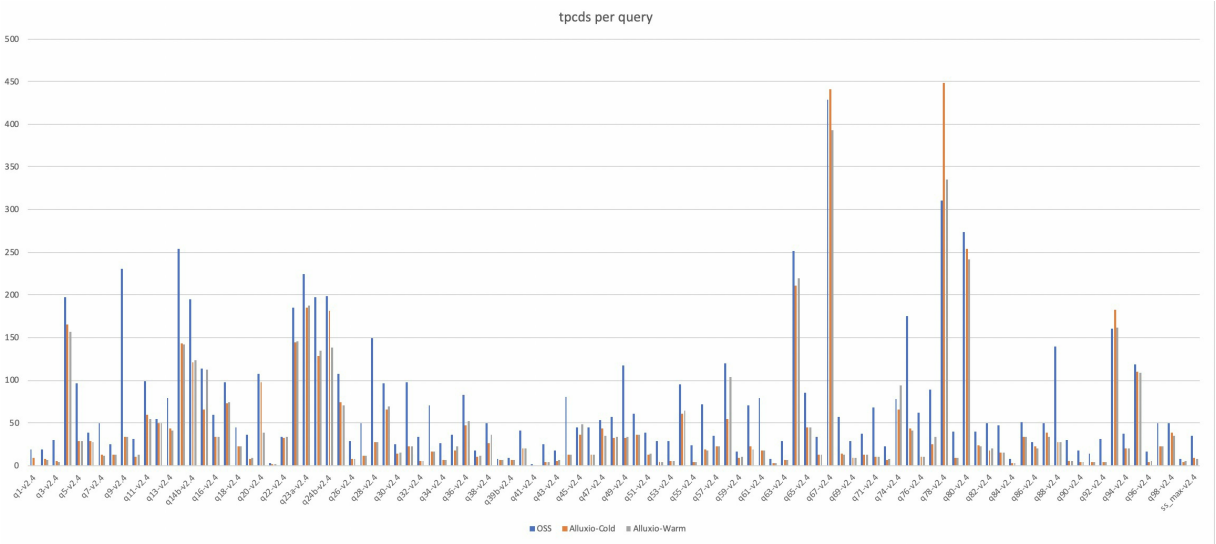
Parameter	Value
spark.driver.cores	5
spark.driver.memory (MB)	20480
spark.executor.cores	7
spark.executor.memory (MB)	20480
spark.executor.instances	20

Test results

The following table lists the amount of time consumed by the tests based on different benchmarks. The queries are performed on 1 TB of data one after another.

Benchmark	Total time consumed by 104 queries (Unit: minutes)
Spark with OSS	180
Spark with Alluxio Cold	145
Spark with Alluxio Warm	137

The following figure shows the amount of time consumed by each query.



Conclusions

The test results show that the query performance is improved after the Alluxio caching service is used. The first time Alluxio is used, the query performance is not high because Alluxio has to cache data from Object Storage Service (OSS). The query performance will be greatly improved in subsequent tests.

> **Notice** The tests analyzed in this topic use ACK-based Spark SQL queries on datasets generated by using Transaction Processing Performance Council-Decision Support (TPC-DS) to compare the performance before and after the Alluxio distributed caching service is used. Therefore, these tests are not based on the TPC benchmarks and may result in a discrepancy between these tests and tests that are based on the TPC benchmarks.

What's next

Troubleshooting

1.1.6. Troubleshooting

During the test, invalid parameters may lead to exceptions in Apache Spark jobs. This topic describes how to use the Spark UI and ACK Spark History Server to view application status and troubleshoot problems.

Prerequisites

Analyze test results

Access Spark UI

You can to view the status of each SQL job in Spark UI in real time. The following content describes the procedure:

1. Run the **kubectl get services** command.

   ```
   ➜  ~ kubectl get services
   NAME                                                  TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)             AGE
   kubernetes                                            ClusterIP   172.[redacted]  <none>        443/TCP             11d
   tpcds-query-runner-with-alluxio-1595839065930-driver-svc   ClusterIP   None           <none>        7078/TCP,7079/TCP   3m54s
   tpcds-query-runner-with-alluxio-ui-svc                ClusterIP   172.[redacted]  <none>        4040/TCP            3m54s
   ➜  ~
   ```

   In this command, **tpcds-query-runner-with-alluxio-ui-svc** is the Spark UI service.

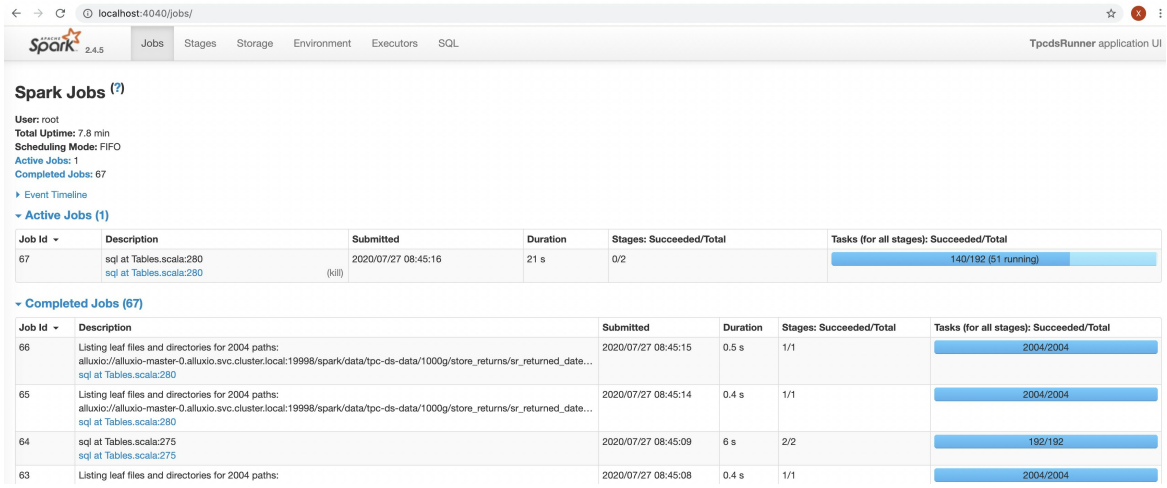
2. Run the following command to access the Spark UI from your on-premises machine:

   ```
   kubectl port-forward service/tpcds-query-runner-with-alluxio-ui-svc 4040:4040
   ```

   The following figure shows the output of the command:

   ```
   ➜  ~ kubectl port-forward service/tpcds-query-runner-with-alluxio-ui-svc 4040:4040
   Forwarding from 127.[redacted]:4040 -> 4040
   Forwarding from [::1]:4040 -> 4040
   Handling connection for 4040
   Handling connection for 4040
   Handling connection for 4040
   Handling connection for 4040
   Handling connection for 4040
   Handling connection for 4040
   ```

3. In the address bar of a browser, enter *localhost:4040* to view jobs in Spark UI.



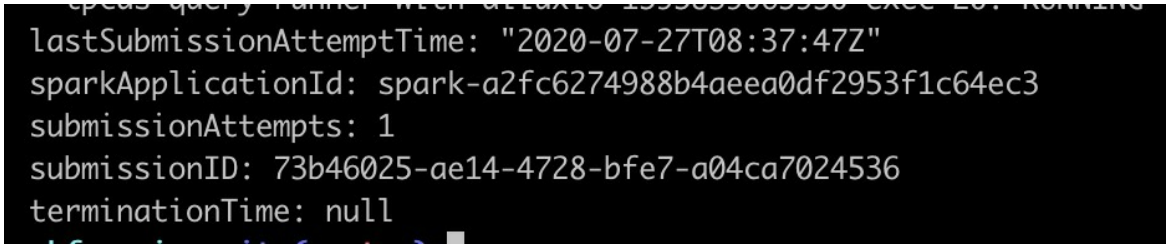
Access ACK Spark History Server

After a job is completed, you can perform the following operations to view historical data:

1. Run the following command to query the ID of the sparkapplication job:

   ```
   kubectl get sparkapplication tpcds-query-runner-with-alluxio -o yaml
   ```

   The following figure shows **sparkApplicationId**.

2. Run the following command to query the endpoint of ACK Spark History Server:

   ```
   kubectl get service ack-spark-history-server
   ```

3. In the address bar of a browser, enter the IP address (39.XX.XX.XXX) that appears in the **EXTERNAL-IP** column in the preceding figure and the port number.

   Then you can view the historical data of all Spark jobs. By using the **sparkApplicationId** returned in Step 1, you can find the job that you want to view and perform troubleshooting.

1.1.7. Use LVM to manage local storage

In big data scenarios, you can use ECS instances of the ecs.d1ne.6xlarge type to run Apache Spark jobs. Each instance has twelve 5 TB hard disk drives (HDDs). You must manually partition, format, and mount these disks. The manual partitioning, formatting, and mounting operations on disks are tedious and time-consuming if the Kubernetes cluster has a large number of nodes. This topic describes a simple way to format and mount data disks by using Logical Volume Manager (LVM).

Prerequisites

- Create an ACK managed cluster.
- Data disks on worker nodes of the cluster have not been formatted or mounted.

Deploy LVM plug-ins

Step 1: Deploy the LVM CSI plug-in by using the following template:

```
apiVersion: storage.k8s.io/v1
kind: CSIDriver
metadata:
  name: localplugin.csi.alibabacloud.com
spec:
  attachRequired: false
  podInfoOnMount: true
---
kind: DaemonSet
apiVersion: apps/v1
metadata:
  name: csi-local-plugin
  namespace: kube-system
spec:
  selector:
    matchLabels:
      app: csi-local-plugin
  template:
    metadata:
      labels:
        app: csi-local-plugin
    spec:
      tolerations:
        - operator: Exists
      serviceAccount: admin
      priorityClassName: system-node-critical
      hostNetwork: true
      hostPID: true
      containers:
        - name: driver-registrar
          image: registry.cn-hangzhou.aliyuncs.com/acs/csi-node-driver-registrar:v1.1.0
          imagePullPolicy: Always
          args:
            - "--v=5"
            - "--csi-address=/csi/csi.sock"
            - "--kubelet-registration-path=/var/lib/kubelet/csi-
plugins/localplugin.csi.alibabacloud.com/csi.sock"
          env:
            - name: KUBE_NODE_NAME
              valueFrom:
                fieldRef:
                  apiVersion: v1
                  fieldPath: spec.nodeName
          volumeMounts:
            - name: plugin-dir
              mountPath: /csi
            - name: registration-dir
              mountPath: /registration
```

```
        - name: csi-localplugin
          securityContext:
            privileged: true
            capabilities:
              add: ["SYS_ADMIN"]
            allowPrivilegeEscalation: true
          image: registry.cn-hangzhou.aliyuncs.com/acs/csi-plugin:v1.14.8.41-bce68b74-
aliyun
          imagePullPolicy: "Always"
          args :
            - "--endpoint=$(CSI_ENDPOINT)"
            - "--v=5"
            - "--nodeid=$(KUBE_NODE_NAME)"
            - "--driver=localplugin.csi.alibabacloud.com"
          env:
            - name: KUBE_NODE_NAME
              valueFrom:
                fieldRef:
                  apiVersion: v1
                  fieldPath: spec.nodeName
            - name: DRIVER_VENDOR
              value: localplugin.csi.alibabacloud.com
            - name: CSI_ENDPOINT
              value: unix://var/lib/kubelet/csi-
plugins/localplugin.csi.alibabacloud.com/csi.sock
          volumeMounts:
            - name: pods-mount-dir
              mountPath: /var/lib/kubelet
              mountPropagation: "Bidirectional"
            - mountPath: /dev
              mountPropagation: "HostToContainer"
              name: host-dev
            - mountPath: /var/log/
              name: host-log
      volumes:
        - name: plugin-dir
          hostPath:
            path: /var/lib/kubelet/csi-plugins/localplugin.csi.alibabacloud.com
            type: DirectoryOrCreate
        - name: registration-dir
          hostPath:
            path: /var/lib/kubelet/plugins_registry
            type: DirectoryOrCreate
        - name: pods-mount-dir
          hostPath:
            path: /var/lib/kubelet
            type: Directory
        - name: host-dev
          hostPath:
            path: /dev
        - name: host-log
          hostPath:
            path: /var/log/
```

```
  updateStrategy:
    rollingUpdate:
      maxUnavailable: 10%
    type: RollingUpdate
```

☐ Step 2: Deploy the LVM CSI provisioner by using the following template:

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: csi-local-provisioner
  namespace: kube-system
spec:
  selector:
    matchLabels:
      app: csi-local-provisioner
  replicas: 2
  template:
    metadata:
      labels:
        app: csi-local-provisioner
    spec:
      tolerations:
      - operator: "Exists"
      affinity:
        nodeAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 1
            preference:
              matchExpressions:
              - key: node-role.kubernetes.io/master
                operator: Exists
      priorityClassName: system-node-critical
      serviceAccount: admin
      hostNetwork: true
      containers:
        - name: external-local-provisioner
          image: registry.cn-hangzhou.aliyuncs.com/acs/csi-provisioner:v1.6.0-b6f763a43-ack
          args:
            - "--csi-address=$(ADDRESS)"
            - "--feature-gates=Topology=True"
            - "--volume-name-prefix=lvm"
            - "--strict-topology=true"
            - "--timeout=150s"
            - "--extra-create-metadata=true"
            - "--enable-leader-election=true"
            - "--leader-election-type=leases"
            - "--retry-interval-start=500ms"
            - "--v=5"
          env:
            - name: ADDRESS
              value: /socketDir/csi.sock
          imagePullPolicy: "Always"
```

```
          volumeMounts:
            - name: socket-dir
              mountPath: /socketDir
        - name: external-local-resizer
          image: registry.cn-hangzhou.aliyuncs.com/acs/csi-resizer:v0.3.0
          args:
            - "--v=5"
            - "--csi-address=$(ADDRESS)"
            - "--leader-election"
          env:
            - name: ADDRESS
              value: /socketDir/csi.sock
          imagePullPolicy: "Always"
          volumeMounts:
            - name: socket-dir
              mountPath: /socketDir/
      volumes:
        - name: socket-dir
          hostPath:
            path: /var/lib/kubelet/csi-plugins/localplugin.csi.alibabacloud.com
            type: DirectoryOrCreate
```

Step 3: Deploy Node Storage Manager by using the following template:

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: node-storage-manager
  namespace: kube-system
spec:
  selector:
    matchLabels:
      app: node-storage-manager
  template:
    metadata:
      labels:
        app: node-storage-manager
    spec:
      containers:
      - args:
        - --nodeid=$(KUBE_NODE_NAME)
        env:
        - name: KUBE_NODE_NAME
          valueFrom:
            fieldRef:
              apiVersion: v1
              fieldPath: spec.nodeName
        image: registry.cn-hangzhou.aliyuncs.com/plugins/node-storage-manager:v1.14.8-
bac4c12
        imagePullPolicy: Always
        name: node-storage-manager
        securityContext:
          allowPrivilegeEscalation: true
          capabilities:
```

```
          capabilities:
            add:
            - SYS_ADMIN
          privileged: true
        volumeMounts:
        - mountPath: /dev
          mountPropagation: HostToContainer
          name: host-dev
        - mountPath: /var/log/
          name: host-log
        - mountPath: /host/etc
          name: etc
      hostNetwork: true
      hostPID: true
      priorityClassName: system-node-critical
      restartPolicy: Always
      serviceAccount: admin
      serviceAccountName: admin
      tolerations:
      - operator: Exists
      volumes:
      - hostPath:
          path: /dev
          type: ""
        name: host-dev
      - hostPath:
          path: /var/log/
          type: ""
        name: host-log
      - hostPath:
          path: /etc
          type: ""
        name: etc
  templateGeneration: 1
  updateStrategy:
    rollingUpdate:
      maxUnavailable: 10%
    type: RollingUpdate
```

☐ Step 4: Deploy a ConfigMap by using the following template:
You can configure the blacklist and whitelist for a node in the pvConfig file. After the ConfigMap is deployed, you can use LVM on the node that you want to manage.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cm-node-storage
  namespace: kube-system
data:
  volumegroup.json: |-
    {
      "volumegroup1": {
        "nodeList": {
            "whiteList": {
                "nodeName": ["all-cluster-nodes"],
                "nodeLabel": ["diskType=ssd", "diskType=hhd"]
            },
            "blackList": {
                "nodeName": [],
                "nodeLabel": []
            }
        },
        "pvConfig": {
            "globalConfig": ["aliyun-local-disk"],
            "specialConfig": {
                "nodeName": {},
                "nodeLabel": {}
            }
        },
        "status": "In_Use"
      }
    }
```

Configure a PVC

1. Configure a StorageClass by using the following template:

```
allowVolumeExpansion: true
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-local-imm
parameters:
  fsType: ext4
  lvmType: striping
  vgName: volumegroup1
  volumeType: LVM
provisioner: localplugin.csi.alibabacloud.com
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

2. Configure a PVC by using the following template:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: lvm-pvc-im
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 12288Gi
  storageClassName: csi-local-imm
```

Configure Alluxio

Configure Alluxio by using the following template:

```
# The Alluxio Open Foundation licenses this work under the Apache License, version 2.0
# (the "License"). You may not use this work except in compliance with the License, which i
s
# available at www.apache.org/licenses/LICENSE-2.0
#
# This software is distributed on an "AS IS" basis, WITHOUT WARRANTIES OR CONDITIONS OF ANY
KIND,
# either express or implied, as more fully set forth in the License.
#
# See the NOTICE file distributed with this work for information regarding copyright owners
hip.
#
# This should not be modified in the usual case.
fullnameOverride: alluxio
## Common ##
# Docker Image
image: registry-vpc.cn-beijing.aliyuncs.com/alluxio/alluxio
imageTag: 2.3.0
imagePullPolicy: IfNotPresent
# Security Context
user: 0
group: 0
fsGroup: 0
# Site properties for all the components
properties:
  fs.oss.accessKeyId: YOUR-ACCESS-KEY-ID
  fs.oss.accessKeySecret: YOUR-ACCESS-KEY-SECRET
  fs.oss.endpoint: oss-cn-beijing-internal.aliyuncs.com
  alluxio.master.mount.table.root.ufs: oss://cloudnativeai/
  alluxio.master.persistence.blacklist: .staging,_temporary
  alluxio.security.stale.channel.purge.interval: 365d
  alluxio.user.metrics.collection.enabled: 'true'
  alluxio.user.short.circuit.enabled: 'true'
  alluxio.user.file.write.tier.default: 1
  alluxio.user.block.size.bytes.default: 64MB #default 64MB
  alluxio.user.file.writetype.default: CACHE_THROUGH
  alluxio.user.file.metadata.load.type: ONCE
  alluxio.user.file.readtype.default: CACHE
```

```
  #alluxio.worker.allocator.class: alluxio.worker.block.allocator.MaxFreeAllocator
  alluxio.worker.allocator.class: alluxio.worker.block.allocator.RoundRobinAllocator
  alluxio.worker.file.buffer.size: 128MB
  alluxio.worker.evictor.class: alluxio.worker.block.evictor.LRUEvictor
  alluxio.job.master.client.threads: 5000
  alluxio.job.worker.threadpool.size: 300
# Recommended JVM Heap options for running in Docker
# Ref: https://developers.redhat.com/blog/2017/03/14/java-inside-docker/
# These JVM options are common to all Alluxio services
# jvmOptions:
#   - "-XX:+UnlockExperimentalVMOptions"
#   - "-XX:+UseCGroupMemoryLimitForHeap"
#   - "-XX:MaxRAMFraction=2"
# Mount Persistent Volumes to all components
# mounts:
# - name: <persistentVolume claimName>
#   path: <mountPath>
# Use labels to run Alluxio on a subset of the K8s nodes
## Master ##
master:
  count: 1 # Controls the number of StatefulSets. For multiMaster mode increase this to >1.
  replicas: 1 # Controls #replicas in a StatefulSet and should not be modified in the usual
case.
  enableLivenessProbe: false
  enableReadinessProbe: false
  args: # Arguments to Docker entrypoint
    - master-only
    - --no-format
  # Properties for the master component
  properties:
  # Example: use ROCKS DB instead of Heap
  # alluxio.master.metastore: ROCKS
  # alluxio.master.metastore.dir: /metastore
  resources:
    # The default xmx is 8G
    limits:
      cpu: "4"
      memory: "8G"
    requests:
      cpu: "1"
      memory: "1G"
  ports:
    embedded: 19200
    rpc: 19998
    web: 19999
  hostPID: true
  hostNetwork: true
  # dnsPolicy will be ClusterFirstWithHostNet if hostNetwork: true
  # and ClusterFirst if hostNetwork: false
  # You can specify dnsPolicy here to override this inference
  # dnsPolicy: ClusterFirst
  # JVM options specific to the master container
  jvmOptions:
  nodeSelector:
```

```
    alluxio: 'true'
jobMaster:
  args:
    - job-master
  # Properties for the jobMaster component
  enableLivenessProbe: false
  enableReadinessProbe: false
  properties:
  resources:
    limits:
      cpu: "4"
      memory: "8G"
    requests:
      cpu: "1"
      memory: "1G"
  ports:
    embedded: 20003
    rpc: 20001
    web: 20002
  # JVM options specific to the jobMaster container
  jvmOptions:
# Alluxio supports journal type of UFS and EMBEDDED
# UFS journal with HDFS example
# journal:
#   type: "UFS"
#   folder: "hdfs://{$hostname}:{$hostport}/journal"
# EMBEDDED journal to /journal example
# journal:
#   type: "EMBEDDED"
#   folder: "/journal"
journal:
  type: "UFS" # "UFS" or "EMBEDDED"
  ufsType: "local" # Ignored if type is "EMBEDDED". "local" or "HDFS"
  folder: "/journal" # Master journal folder
  # volumeType controls the type of journal volume.
  # It can be "persistentVolumeClaim" or "emptyDir"
  volumeType: emptyDir
  size: 1Gi
  # Attributes to use when the journal is persistentVolumeClaim
  storageClass: "standard"
  accessModes:
    - ReadWriteOnce
  # Attributes to use when the journal is emptyDir
  medium: ""
  # Configuration for journal formatting job
  format:
    runFormat: false # Change to true to format journal
    job:
      activeDeadlineSeconds: 30
      ttlSecondsAfterFinished: 10
    resources:
      limits:
        cpu: "4"
        memory: "8G"
```

```
      requests:
        cpu: "1"
        memory: "1G"
# You can enable metastore to use ROCKS DB instead of Heap
# metastore:
#   volumeType: persistentVolumeClaim # Options: "persistentVolumeClaim" or "emptyDir"
#   size: 1Gi
#   mountPath: /metastore
# # Attributes to use when the metastore is persistentVolumeClaim
#   storageClass: "standard"
#   accessModes:
#    - ReadWriteOnce
# # Attributes to use when the metastore is emptyDir
#   medium: ""
## Worker ##
worker:
  args:
    - worker-only
    - --no-format
  enableLivenessProbe: false
  enableReadinessProbe: false
  # Properties for the worker component
  properties:
  resources:
    limits:
      cpu: "4"
      memory: "4G"
    requests:
      cpu: "1"
      memory: "2G"
  ports:
    rpc: 29999
    web: 30000
  hostPID: true
  hostNetwork: true
  # dnsPolicy will be ClusterFirstWithHostNet if hostNetwork: true
  # and ClusterFirst if hostNetwork: false
  # You can specify dnsPolicy here to override this inference
  # dnsPolicy: ClusterFirst
  # JVM options specific to the worker container
  jvmOptions:
  nodeSelector:
    alluxio: 'true'
jobWorker:
  args:
    - job-worker
  enableLivenessProbe: false
  enableReadinessProbe: false
  # Properties for the jobWorker component
  properties:
  resources:
    limits:
      cpu: "4"
      memory: "4G"
    requests:
```

```
    requests:
      cpu: "1"
      memory: "1G"
  ports:
    rpc: 30001
    data: 30002
    web: 30003
  # JVM options specific to the jobWorker container
  jvmOptions:
# Tiered Storage
# emptyDir example
#  - level: 0
#    alias: MEM
#    mediumtype: MEM
#    path: /dev/shm
#    type: emptyDir
#    quota: 1G
#
# hostPath example
#  - level: 0
#    alias: MEM
#    mediumtype: MEM
#    path: /dev/shm
#    type: hostPath
#    quota: 1G
#
# persistentVolumeClaim example
#  - level: 1
#    alias: SSD
#    mediumtype: SSD
#    type: persistentVolumeClaim
#    name: alluxio-ssd
#    path: /dev/ssd
#    quota: 10G
#
# multi-part mediumtype example
#  - level: 1
#    alias: SSD,HDD
#    mediumtype: SSD,HDD
#    type: persistentVolumeClaim
#    name: alluxio-ssd,alluxio-hdd
#    path: /dev/ssd,/dev/hdd
#    quota: 10G,10G
tieredstore:
  levels:
    - level: 0
      alias: HDD
      mediumtype: HDD-0
      path: /mnt/disk1
      type: persistentVolumeClaim
      name: lvm-pvc-im
      quota: 12000G
      high: 0.95
      low: 0.7
# Short circuit related properties
```

```
shortCircuit:
  enabled: true
  # The policy for short circuit can be "local" or "uuid",
  # local means the cache directory is in the same mount namespace,
  # uuid means interact with domain socket
  policy: uuid
  # volumeType controls the type of shortCircuit volume.
  # It can be "persistentVolumeClaim" or "hostPath"
  volumeType: hostPath
  size: 1Mi
  # Attributes to use if the domain socket volume is PVC
  pvcName: alluxio-worker-domain-socket
  accessModes:
    - ReadWriteOnce
  storageClass: standard
  # Attributes to use if the domain socket volume is hostPath
  hostPath: "/tmp/alluxio-domain" # The hostPath directory to use
## FUSE ##
fuse:
  image: registry-vpc.cn-beijing.aliyuncs.com/alluxio/alluxio-fuse
  imageTag: 2.3.0
  imagePullPolicy: IfNotPresent
  # Change both to true to deploy FUSE
  enabled: false
  clientEnabled: false
  # Properties for the jobWorker component
  properties:
  # Customize the MaxDirectMemorySize
  # These options are specific to the FUSE daemon
  jvmOptions:
    - "-XX:MaxDirectMemorySize=2g"
  hostNetwork: true
  hostPID: true
  dnsPolicy: ClusterFirstWithHostNet
  user: 0
  group: 0
  fsGroup: 0
  args:
    - fuse
    - --fuse-opts=allow_other
  # Mount path in the host
  mountPath: /mnt/alluxio-fuse
  resources:
    requests:
      cpu: "0.5"
      memory: "1G"
    limits:
      cpu: "4"
      memory: "4G"
  nodeSelector:
    alluxio: 'true'
##  Secrets ##
# Format: (<name>:<mount path under /secrets/>):
# secrets:
```

```
#   master: # Shared by master and jobMaster containers
#     alluxio-hdfs-config: hdfsConfig
#   worker: # Shared by worker and jobWorker containers
#     alluxio-hdfs-config: hdfsConfig
```

Notice

- You must use the preceding PVC to configure tieredstore.

```
tieredstore:
  levels:
    - level: 0
      alias: HDD
      mediumtype: HDD-0
      path: /mnt/disk1
      type: persistentVolumeClaim
      name: lvm-pvc-im
      quota: 12000G
      high: 0.95
      low: 0.7
```

- Alluxio and the PVC must be deployed in the same namespace.

Related information

- Use LVs
- Set up a test environment