

1.19.2

Best Practices

Date published: 2023-06-22

Date modified: 2023-06-22

CLOUDERA

Legal Notice

© Cloudera Inc. 2024. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Recommendations for scaling Cloudera Data Engineering deployments.....	4
Apache Airflow scaling and tuning considerations.....	5
Cloudera Data Engineering performance tuning.....	6
Best practices for building Apache Spark applications.....	6

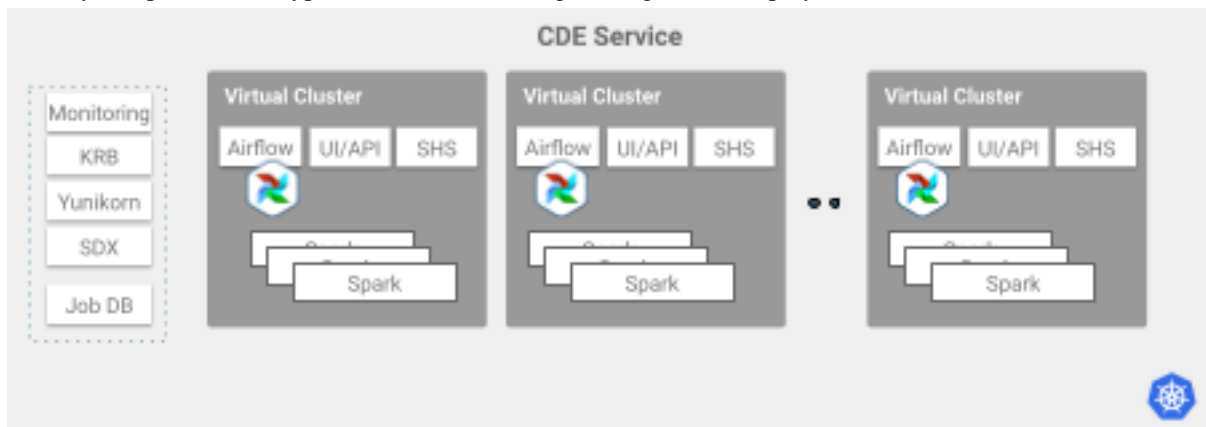
Recommendations for scaling Cloudera Data Engineering deployments

Your business might experience a sudden increase or drop in demand due to which your Cloudera Data Engineering deployment needs to autoscale. You can scale your Cloudera Data Engineering deployment by either adding new instances of a Cloudera Data Engineering service or Virtual Cluster, or by adding additional resources to the existing ones.

You can scale your Cloudera Data Engineering deployment:

- Vertically - More resources are provisioned within the same instance of a Cloudera Data Engineering service or Virtual Cluster.
- Horizontally - New instances of Cloudera Data Engineering service or Virtual Cluster are provisioned.

The key components of a typical Cloudera Data Engineering service deployment are:



Virtual Clusters provide an isolated autoscaling compute capacity to run Spark and Airflow jobs. You can use Virtual Clusters to isolate individual teams or lines of business by using user-based Access Control Lists (ACLs).

Guidelines for scaling Virtual Clusters

- Each Virtual Cluster requires infrastructure capacity to run various services such as Airflow, API server, and Spark-History-Server (SHS).

Recommendation: Do not scale horizontally beyond 50 Virtual Clusters within the same Cloudera Data Engineering service.

- Virtual Clusters can actively run hundreds of parallel jobs. In certain scenarios, it might be required to simultaneously submit multiple jobs as per the schedule or due to a burst in demand. In these scenarios, the API server enforces guardrails and limits the number of simultaneous Spark job submissions to 60. Once the jobs move from submission to running state, you can submit more jobs.



Note: The Airflow jobs are not impacted by the submission limit of 60 simultaneous jobs.

Recommendation: Distribute simultaneous submission of jobs over time or horizontally scale across multiple Virtual Clusters.

Job submission rate guardrails

When jobs are submitted to the Cloudera Data Engineering API server of a particular Virtual Cluster, it takes time and resources, known as preparing and starting states, before they begin running. This process is called the job submission intake process. To ensure proper resourcing and handling of incoming jobs, guardrails have been set up. By default, the guardrail, or limit is set to 60 simultaneous job submissions. Simultaneous incoming Spark job

submissions that exceed 60 return a 429 error message to the client. The example error message is: Failed to submit. Too many requests.

Recommendation:

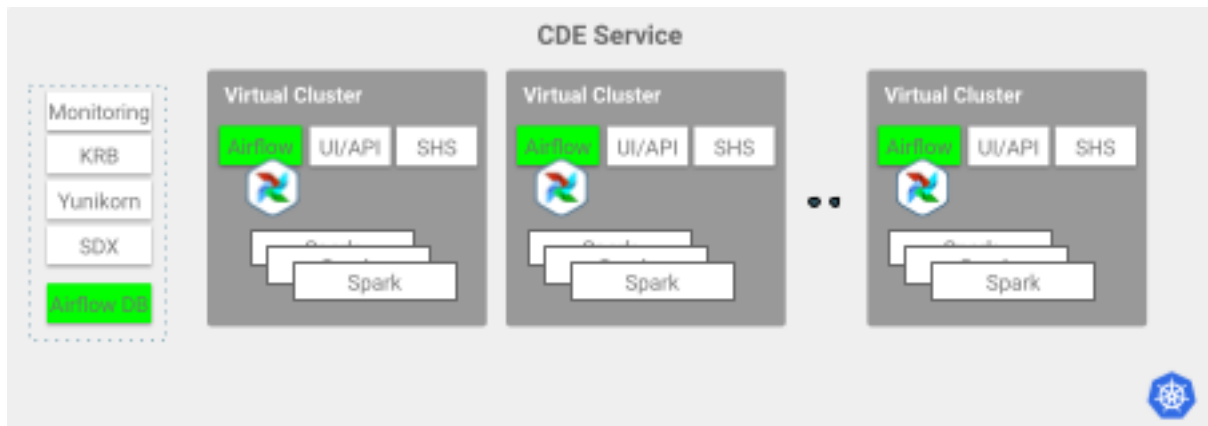
- Incorporate CDE Clients error handling for this error. The CDE CLI exit code is 77. The CDE CLI receives an HTTP 429 response code for the request to the Runtime API Server's REST API. For more information on exit codes, see Cloudera Data Engineering CLI exit codes.
- If needed, increase or decrease the guardrail limit during the Virtual Cluster creation by setting the "limiting.simultaneousJobSubmissions" configuration.

Apache Airflow scaling and tuning considerations

There are certain limitations related to the deployment architecture, and guidelines for scaling and tuning of the deployment, that you must consider while creating or running Airflow jobs (DAGs).

CDE deployment architecture

If you use Airflow to schedule and/or develop multi-step pipelines, you must consider the following deployment limitations to scale your environment.



These guidelines describe when to scale horizontally, given total loaded DAGs and/or number of parallel (concurrent) tasks. These apply to all cloud providers.

CDE service guidelines

The number of Airflow jobs that can run at the same time is limited by the number of parallel tasks that are triggered by the associated DAGs. The number of tasks can reach up to 250-300 when running in parallel per CDE service.

Recommendation: Create more CDE services to increase Airflow tasks concurrency beyond the limits noted above.

Virtual Cluster guidelines

- Each Virtual Cluster has a maximum parallel task execution limit of 250. This applies to both Amazon Web Services (AWS) and Azure.



Note: It is recommended that you do not exceed 300 parallel tasks across all Virtual Clusters.

- For AWS: The recommended maximum number of Airflow jobs loaded per virtual cluster is 1500. As the number of jobs increases, the job page loading time experiences latency along with Airflow job creation time.

- For Azure: The recommended maximum number of Airflow jobs loaded per virtual cluster is 300 to 500. Higher capacity can be achieved but will require manual configuration changes.

Recommendation: Create more Virtual Clusters to load additional Airflow DAGs.

- The suggested number of Airflow jobs created in parallel is three or less. For jobs that are created in parallel, specify a DAG and/or a task_group for each Airflow operator.

Recommendation: Distribute the creation of Airflow jobs so that you do not create more than three jobs at the same time.

Cloudera Data Engineering performance tuning

Use local SSD

Using default instances without local SSD does impact performance. Instances with local SSDs generally provide faster performance, but you need to ensure you choose instance types that have large enough local storage for intermediate results.

Avoid instance types with low compute profiles

Make sure there is reasonable headroom (at least 20%) between the per executor memory and core usage and what the instance profile provides.

Disable Spark Analysis

By default, CDE collects metrics around CPU, memory, and I/O. This can have a negative impact on performance - especially for long running jobs that take several hours to complete. Profiling metrics are useful in development and testing, but an unnecessary overhead in a production setting.

You can turn this feature off using the Spark Analysis toggle on the job Configuration tab.

Use Spark 3 instead of Spark 2

Spark 3 introduced performance improvements over Spark 2, therefore Cloudera recommends using it whenever possible.

Best practices for building Apache Spark applications

Follow these best practices when building Apache Spark Scala and Java applications:

- Refrain from using withColumn in chain, loop, and calling it multiple times in a single query. Doing so may cause performance issues. To avoid issues, use select() with multiple columns at once. See the Apache Spark API reference linked below for more information.
- Compile your applications against the same version of Spark that you are running.
- Build a single assembly JAR ("Uber" JAR) that includes all dependencies. In Maven, add the Maven assembly plug-in to build a JAR containing all dependencies:

```
<plugin>
  <artifactId>maven-assembly-plugin</artifactId>
  <configuration>
    <descriptorRefs>
      <descriptorRef>jar-with-dependencies</descriptorRef>
    </descriptorRefs>
  </configuration>
</executions>
```

```
<execution>
  <id>make-assembly</id>
  <phase>package</phase>
  <goals>
    <goal>single</goal>
  </goals>
</execution>
</executions>
</plugin>
```

This plug-in manages the merge procedure for all available JAR files during the build. Exclude Spark, Hadoop, and Kafka classes from the assembly JAR, because they are already available on the cluster and contained in the runtime classpath. In Maven, specify Spark, Hadoop, and Kafka dependencies with scope provided. For example:

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-core_2.11</artifactId>
  <version>2.4.0.7.0.0.0</version>
  <scope>provided</scope>
</dependency>
```