Cloudera Streaming Analytics

Job lifecycle

Date published: 2019-12-17
Date modified: 2019-12-17

Contents

Running a Flink job

After developing your application, you must submit the job to the Flink cluster. To submit the Flink job, you need to run the Flink client in the command line with also including all the configuration and security parameters along the run command.

Before you begin

- You have deployed the Flink parcel on your CDP Private Cloud Base cluster.
- You have HDFS Gateway, Flink and YARN Gateway roles assigned to the host you are using for Flink submission. For instructions, see the Cloudera Manager documentation.
- You have established your HDFS home directory.

Procedure

The following is a working example of a word count application that reads text from a socket and counts the number of distinct words.

```
> hdfs dfs -put /opt/cloudera/parcels/FLINK/lib/flink/README.txt /tmp
> flink run --detached \
 /opt/cloudera/parcels/FLINK/lib/flink/examples/streaming/WordCount.jar \
 --input hdfs:///tmp/README.txt \
 --output hdfs:///tmp/ReadMe-Counts
> hdfs dfs -tail /tmp/ReadMe-Counts
...
(and,7)
(source,1)
(code,2)
...
```

Note:

To run a Flink job, your HDFS Home Directory has to exist. If it does not exist, you receive an error message similar to:

```
Permission denied: user=$USER_NAME, access=WRITE, inode=“/user”
```

You can set how to run your Flink job with the execution.target setting in the Flink configuration file. By default, execution.target is set to yarn-per-job, but you can change it to yarn-session. It is recommended to use per-job configuration to simple jobs, and the session configuration in case of SQL client.

Related Information

Setting up your HDFS Home directory
Simple Tutorial: Running the application from IntelliJ
Simple Tutorial: Running the application on a Cloudera cluster
Stateful Tutorial: Deploy and monitor the application

Flink CLI

You can use the Flink command line interface to operate, configure and maintain your Flink applications.

The Flink CLI works without requiring the user to always specify the YARN application ID when submitting commands to Flink jobs. Instead, the jobs are identified uniquely on the YARN cluster by their job IDs.

The following improvements are implemented for Flink CLI:

- flink list: This command lists all the jobs on the YARN cluster by default, instead of listing the jobs of a single Flink cluster.
- flink savepoint <jobId> and flink cancel <jobId>: The savepoint and cancel commands, along with the other single job commands, no longer require the -yId parameter, and work if you provide only the ID of the job.
- flink run: You do not need to specify -m yarn-cluster, as it is included in the run command by default.

Important: On secure clusters an extra -yD security.ssl.rest.enabled=true parameter is required to run flink list/cancel job commands, for example:

```
flink list -yD security.ssl.rest.enabled=true
flink stop -yD security.ssl.rest.enabled=true <jobId>
flink cancel -yD security.ssl.rest.enabled=true <jobId>
```

Checkpoint

To make your Flink application fault tolerant, you need to enable checkpointing in your application design. If an error occurs, you can reload your application with state by using the automatically saved snapshot of your data stream.

Checkpointing improves Flink fault tolerance by enabling application to recover their state and stream to the same point in time as the latest availableat a checkpoint in case of failure. Checkpointing is not enabled by default.

Before enabling this feature make sure that:

- The source is capable to replay records for a certain amount of time (such as Kafka or HDFS)
- The storage for the state is persistent (such as HDFS)

Enable checkpointing in Flink with env.enableCheckpointing(num), where env is an instance of StreamExecutionE nvironment and num refers to the checkpoint interval in milliseconds. The env.getCheckpointConfig() configuration object is used to configure the checkpointing settings.

See the Apache Flink documentation for the available settings.

The following is an example of a checkpoint configuration.

```
// Configure checkpointing if interval is set
  long cpInterval = params.getLong("checkpoint.interval.millis", TimeUnit.
MINUTES.toMillis(1));
  if (cpInterval > 0) {
   CheckpointConfig checkpointConf = env.getCheckpointConfig();
   checkpointConf.setCheckpointInterval(cpInterval);
   checkpointConf.setCheckpointingMode(CheckpointingMode.EXACTLY_ONCE);
   checkpointConf.setCheckpointTimeout(TimeUnit.HOURS.toMillis(1));
   checkpointConf.enableExternalizedCheckpoints(CheckpointConfig.External
izedCheckpointCleanup.RETAIN_ON_CANCELLATION);
   env.getConfig().setUseSnapshotCompression(true);
  }
```

Savepoint

Beside checkpointing, you are also able to create a savepoint of your executed Flink jobs. Savepoints are not automatically created, so you need to trigger them in case of upgrade or maintenance. You can also resume your applications from savepoint.

You can set the default savepoint directory in flink-conf.yaml under state.savepoints.dir property.

The following command lines can be used to maintain savepoints:

Trigger savepoint	$ bin/flink savepoint -m yarn-cluster -yid <*yarnAppID*> <*jobId*> [ta rgetDirectory]
Cancel job with savepoint	$ bin/flink cancel -m yarn-cluster -yid <*yarnAppID*> <*jobId*>
Resume from savepoint	$ bin/flink run -m yarn-cluster -s <*savepointPath*> [runArgs]
Deleting savepoint	$ bin/flink savepoint -m yarn-cluster -d <*savepointPath*>