Feast Spark SDK

Feast Authors

Jul 08, 2022

CONTENTS:

CHAPTER

ONE

CLIENT

class feast_spark.client.**Client**(*feast_client: Client*)

Bases: object

property config: Config

property feature_store: Client

get_health_metrics(*project: str*, *table_names: List[str]*) → Dict[str, List[str]]

get_historical_features(*feature_refs: List[str]*, *entity_source: Union[DataFrame, FileSource, BigQuerySource]*, *output_location: Optional[str] = None*) → RetrievalJob

Launch a historical feature retrieval job.

Parameters

- **feature_refs** – List of feature references that will be returned for each entity. Each feature reference should have the following format: "feature_table:feature" where "feature_table" & "feature" refer to the feature and feature table names respectively.
- **entity_source** (*Union[pd.DataFrame, FileSource, BigQuerySource]*) – Source for the entity rows. If entity_source is a Panda DataFrame, the dataframe will be staged to become accessible by spark workers. If one of feature tables' source is in BigQuery - entities will be upload to BQ. Otherwise to remote file storage (derived from configured staging location). It is also assumed that the column event_timestamp is present in the dataframe, and is of type datetime without timezone information.

  The user needs to make sure that the source (or staging location, if entity_source is a Panda DataFrame) is accessible from the Spark cluster that will be used for the retrieval job.
- **output_location** – Specifies the path in a bucket to write the exported feature data files

Returns

Returns a retrieval job object that can be used to monitor retrieval progress asynchronously, and can be used to materialize the results.

Examples

```
>>> import feast
>>> import feast_spark
>>> from feast.data_format import ParquetFormat
>>> from datetime import datetime
>>> feast_client = feast.Client(core_url="localhost:6565")
>>> feature_refs = ["bookings:bookings_7d", "bookings:booking_14d"]
>>> entity_source = FileSource("event_timestamp", ParquetFormat(), "gs://some-
↪bucket/customer")
>>> feature_retrieval_job = feast_spark.Client(feast_client).get_historical_
↪features(
>>>     feature_refs, entity_source)
>>> output_file_uri = feature_retrieval_job.get_output_file_uri()
    "gs://some-bucket/output/
```

get_historical_features_df(*feature_refs: List[str]*, *entity_source: Union[FileSource, BigQuerySource]*)

Launch a historical feature retrieval job.

Parameters

- **feature_refs** – List of feature references that will be returned for each entity. Each feature reference should have the following format: "feature_table:feature" where "feature_table" & "feature" refer to the feature and feature table names respectively.
- **entity_source** (*Union[FileSource, BigQuerySource]*) – Source for the entity rows. The user needs to make sure that the source is accessible from the Spark cluster that will be used for the retrieval job.

Returns

Returns the historical feature retrieval result in the form of Spark dataframe.

Examples

```
>>> import feast
>>> import feast_spark
>>> from feast.data_format import ParquetFormat
>>> from datetime import datetime
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession.builder.getOrCreate()
>>> feast_client = feast.Client(core_url="localhost:6565")
>>> feature_refs = ["bookings:bookings_7d", "bookings:booking_14d"]
>>> entity_source = FileSource("event_timestamp", ParquetFormat, "gs://some-
↪bucket/customer")
>>> df = feast_spark.Client(feast_client).get_historical_features(
>>>     feature_refs, entity_source)
```

get_job_by_id(*job_id: str*) → SparkJob

list_jobs(*include_terminated: bool*, *project: Optional[str] = None*, *table_name: Optional[str] = None*) → List[SparkJob]

List ingestion jobs currently running in Feast.

Parameters

- **include_terminated** – Flag to include terminated jobs or not
- **project** – Optionally specify the project to use as filter when retrieving jobs
- **table_name** – Optionally specify name of feature table to use as filter when retrieving jobs

Returns
List of SparkJob ingestion jobs.

property metrics_redis: Redis

schedule_offline_to_online_ingestion(*feature_table: FeatureTable*, *ingestion_timespan: int*, *cron_schedule: str*)

Launch Scheduled Ingestion Job from Batch Source to Online Store for given feature table

Parameters

- **feature_table** – FeatureTable that will be ingested into the online store
- **ingestion_timespan** – Days of data which will be ingestion per job. The boundaries on which to filter the source are [end of day of execution date - ingestion_timespan (days) , end of day of execution date)
- **cron_schedule** – Cron schedule expression

Returns: Spark Job Proxy object

start_offline_to_online_ingestion(*feature_table: FeatureTable*, *start: datetime*, *end: datetime*) → SparkJob

Launch Ingestion Job from Batch Source to Online Store for given feature table

Parameters

- **feature_table** – FeatureTable that will be ingested into the online store
- **start** – lower datetime boundary on which to filter the source
- **end** – upper datetime boundary on which to filter the source

Returns: Spark Job Proxy object

start_stream_to_online_ingestion(*feature_table: FeatureTable*, *extra_jars: Optional[List[str]] = None*, *project: Optional[str] = None*) → SparkJob

unschedule_offline_to_online_ingestion(*feature_table: FeatureTable*, *project=None*)

feast_spark.client.**stage_entities_to_bq_with_partition**(*entity_source: DataFrame*, *project: str*, *dataset: str*) → BigQuerySource

Stores given (entity) dataframe as new table in BQ. Name of the table generated based on current time. Table will expire in 1 day. Returns BigQuerySource with reference to created table.

PYTHON MODULE INDEX

f

INDEX