Feast Documentation

Feast Authors

Sep 30, 2022

CONTENTS

CHAPTER

ONE

FEATURE STORE

class feast.feature_store.**FeatureStore**(*repo_path: Optional[str] = None*, *config: Optional[feast.repo_config.RepoConfig] = None*, *fs_yaml_file: Optional[pathlib.Path] = None*)

A FeatureStore object is used to define, create, and retrieve features.

config
The config for the feature store.

Type *feast.repo_config.RepoConfig*

repo_path
The path to the feature repo.

Type pathlib.Path

_registry
The registry for the feature store.

Type *feast.infra.registry.base_registry.BaseRegistry*

_provider
The provider for the feature store.

Type *feast.infra.provider.Provider*

_go_server
The (optional) Go feature server for the feature store.

Type Optional[EmbeddedOnlineFeatureServer]

apply(*objects: Union[feast.data_source.DataSource, feast.entity.Entity, feast.feature_view.FeatureView, feast.on_demand_feature_view.OnDemandFeatureView, feast.request_feature_view.RequestFeatureView, feast.batch_feature_view.BatchFeatureView, feast.stream_feature_view.StreamFeatureView, feast.feature_service.FeatureService, feast.saved_dataset.ValidationReference, List[Union[feast.feature_view.FeatureView, feast.on_demand_feature_view.OnDemandFeatureView, feast.request_feature_view.RequestFeatureView, feast.batch_feature_view.BatchFeatureView, feast.stream_feature_view.StreamFeatureView, feast.entity.Entity, feast.feature_service.FeatureService, feast.data_source.DataSource, feast.saved_dataset.ValidationReference]]]*, *objects_to_delete: Optional[List[Union[feast.feature_view.FeatureView, feast.on_demand_feature_view.OnDemandFeatureView, feast.request_feature_view.RequestFeatureView, feast.batch_feature_view.BatchFeatureView, feast.stream_feature_view.StreamFeatureView, feast.entity.Entity, feast.feature_service.FeatureService, feast.data_source.DataSource, feast.saved_dataset.ValidationReference]]] = None*, *partial: bool = True*)

Register objects to metadata store and update related infrastructure.

The apply method registers one or more definitions (e.g., Entity, FeatureView) and registers or updates these objects in the Feast registry. Once the apply method has updated the infrastructure (e.g., create tables in an online store), it will commit the updated registry. All operations are idempotent, meaning they can safely be rerun.

Parameters

- **objects** – A single object, or a list of objects that should be registered with the Feature Store.
- **objects_to_delete** – A list of objects to be deleted from the registry and removed from the provider's infrastructure. This deletion will only be performed if partial is set to False.
- **partial** – If True, apply will only handle the specified objects; if False, apply will also delete all the objects in objects_to_delete, and tear down any associated cloud resources.

Raises ValueError – The 'objects' parameter could not be parsed properly.

Examples

Register an Entity and a FeatureView.

```
>>> from feast import FeatureStore, Entity, FeatureView, Feature, FileSource,␣
↪RepoConfig
>>> from datetime import timedelta
>>> fs = FeatureStore(repo_path="project/feature_repo")
>>> driver = Entity(name="driver_id", description="driver id")
>>> driver_hourly_stats = FileSource(
...     path="project/feature_repo/data/driver_stats.parquet",
...     timestamp_field="event_timestamp",
...     created_timestamp_column="created",
... )
>>> driver_hourly_stats_view = FeatureView(
...     name="driver_hourly_stats",
...     entities=[driver],
...     ttl=timedelta(seconds=86400 * 1),
...     source=driver_hourly_stats,
... )
>>> fs.apply([driver_hourly_stats_view, driver]) # register entity and feature␣
↪view
```

create_saved_dataset(*from_: feast.infra.offline_stores.offline_store.RetrievalJob*, *name: str*, *storage: feast.saved_dataset.SavedDatasetStorage*, *tags: Optional[Dict[str, str]] = None*, *feature_service: Optional[feast.feature_service.FeatureService] = None*, *allow_overwrite: bool = False*) → feast.saved_dataset.SavedDataset

Execute provided retrieval job and persist its outcome in given storage. Storage type (eg, BigQuery or Redshift) must be the same as globally configured offline store. After data successfully persisted saved dataset object with dataset metadata is committed to the registry. Name for the saved dataset should be unique within project, since it's possible to overwrite previously stored dataset with the same name.

Parameters

- **from** – The retrieval job whose result should be persisted.
- **name** – The name of the saved dataset.
- **storage** – The saved dataset storage object indicating where the result should be persisted.

- **tags** (*optional*) – A dictionary of key-value pairs to store arbitrary metadata.
- **feature_service** (*optional*) – The feature service that should be associated with this saved dataset.
- **allow_overwrite** (*optional*) – If True, the persisted result can overwrite an existing table or file.

Returns SavedDataset object with attached RetrievalJob

Raises **ValueError if given retrieval job doesn't have metadata** –

delete_feature_service(*name: str*)
Deletes a feature service.

Parameters **name** – Name of feature service.

Raises **FeatureServiceNotFoundException** – The feature view could not be found.

delete_feature_view(*name: str*)
Deletes a feature view.

Parameters **name** – Name of feature view.

Raises **FeatureViewNotFoundException** – The feature view could not be found.

get_data_source(*name: str*) → *feast.data_source.DataSource*
Retrieves the list of data sources from the registry.

Parameters **name** – Name of the data source.

Returns The specified data source.

Raises **DataSourceObjectNotFoundException** – The data source could not be found.

get_entity(*name: str*, *allow_registry_cache: bool = False*) → *feast.entity.Entity*
Retrieves an entity.

Parameters

- **name** – Name of entity.
- **allow_registry_cache** – (Optional) Whether to allow returning this entity from a cached registry

Returns The specified entity.

Raises **EntityNotFoundException** – The entity could not be found.

get_feature_server_endpoint() → Optional[str]
Returns endpoint for the feature server, if it exists.

get_feature_service(*name: str*, *allow_cache: bool = False*) → *feast.feature_service.FeatureService*
Retrieves a feature service.

Parameters

- **name** – Name of feature service.
- **allow_cache** – Whether to allow returning feature services from a cached registry.

Returns The specified feature service.

Raises **FeatureServiceNotFoundException** – The feature service could not be found.

get_feature_view(*name: str*, *allow_registry_cache: bool = False*) → *feast.feature_view.FeatureView*
Retrieves a feature view.

Parameters

- **name** – Name of feature view.
- **allow_registry_cache** – (Optional) Whether to allow returning this entity from a cached registry

Returns The specified feature view.

Raises **FeatureViewNotFoundException** – The feature view could not be found.

get_historical_features(*entity_df: Union[pandas.core.frame.DataFrame, str]*, *features: Union[List[str]*, feast.feature_service.FeatureService*]*, *full_feature_names: bool = False*) → *feast.infra.offline_stores.offline_store.RetrievalJob*

Enrich an entity dataframe with historical feature values for either training or batch scoring.

This method joins historical feature data from one or more feature views to an entity dataframe by using a time travel join.

Each feature view is joined to the entity dataframe using all entities configured for the respective feature view. All configured entities must be available in the entity dataframe. Therefore, the entity dataframe must contain all entities found in all feature views, but the individual feature views can have different entities.

Time travel is based on the configured TTL for each feature view. A shorter TTL will limit the amount of scanning that will be done in order to find feature data for a specific entity key. Setting a short TTL may result in null values being returned.

Parameters

- **entity_df** (*Union[pd.DataFrame, str]*) – An entity dataframe is a collection of rows containing all entity columns (e.g., customer_id, driver_id) on which features need to be joined, as well as a event_timestamp column used to ensure point-in-time correctness. Either a Pandas DataFrame can be provided or a string SQL query. The query must be of a format supported by the configured offline store (e.g., BigQuery)
- **features** – The list of features that should be retrieved from the offline store. These features can be specified either as a list of string feature references or as a feature service. String feature references must have format "feature_view:feature", e.g. "customer_fv:daily_transactions".
- **full_feature_names** – If True, feature names will be prefixed with the corresponding feature view name, changing them from the format "feature" to "feature_view__feature" (e.g. "daily_transactions" changes to "customer_fv__daily_transactions").

Returns RetrievalJob which can be used to materialize the results.

Raises **ValueError** – Both or neither of features and feature_refs are specified.

Examples

Retrieve historical features from a local offline store.

```
>>> from feast import FeatureStore, RepoConfig
>>> import pandas as pd
>>> fs = FeatureStore(repo_path="project/feature_repo")
>>> entity_df = pd.DataFrame.from_dict(
...     {
...         "driver_id": [1001, 1002],
...         "event_timestamp": [
...             datetime(2021, 4, 12, 10, 59, 42),
```

(continues on next page)

(continued from previous page)

```
...             datetime(2021, 4, 12, 8, 12, 10),
...         ],
...     }
... )
>>> retrieval_job = fs.get_historical_features(
...     entity_df=entity_df,
...     features=[
...         "driver_hourly_stats:conv_rate",
...         "driver_hourly_stats:acc_rate",
...         "driver_hourly_stats:avg_daily_trips",
...     ],
... )
>>> feature_data = retrieval_job.to_df()
```

get_on_demand_feature_view(*name: str*) → *feast.on_demand_feature_view.OnDemandFeatureView*

Retrieves a feature view.

Parameters **name** – Name of feature view.

Returns The specified feature view.

Raises **FeatureViewNotFoundException** – The feature view could not be found.

get_online_features(*features: Union[List[str], feast.feature_service.FeatureService]*, *entity_rows: List[Dict[str, Any]]*, *full_feature_names: bool = False*) → feast.online_response.OnlineResponse

Retrieves the latest online feature data.

Note: This method will download the full feature registry the first time it is run. If you are using a remote registry like GCS or S3 then that may take a few seconds. The registry remains cached up to a TTL duration (which can be set to infinity). If the cached registry is stale (more time than the TTL has passed), then a new registry will be downloaded synchronously by this method. This download may introduce latency to online feature retrieval. In order to avoid synchronous downloads, please call refresh_registry() prior to the TTL being reached. Remember it is possible to set the cache TTL to infinity (cache forever).

Parameters

- **features** – The list of features that should be retrieved from the online store. These features can be specified either as a list of string feature references or as a feature service. String feature references must have format "feature_view:feature", e.g. "customer_fv:daily_transactions".
- **entity_rows** – A list of dictionaries where each key-value is an entity-name, entity-value pair.
- **full_feature_names** – If True, feature names will be prefixed with the corresponding feature view name, changing them from the format "feature" to "feature_view__feature" (e.g. "daily_transactions" changes to "customer_fv__daily_transactions").

Returns OnlineResponse containing the feature data in records.

Raises **Exception** – No entity with the specified name exists.

Examples

Retrieve online features from an online store.

```
>>> from feast import FeatureStore, RepoConfig
>>> fs = FeatureStore(repo_path="project/feature_repo")
>>> online_response = fs.get_online_features(
...     features=[
...         "driver_hourly_stats:conv_rate",
...         "driver_hourly_stats:acc_rate",
...         "driver_hourly_stats:avg_daily_trips",
...     ],
...     entity_rows=[{"driver_id": 1001}, {"driver_id": 1002}, {"driver_id":␣
↪1003}, {"driver_id": 1004}],
... )
>>> online_response_dict = online_response.to_dict()
```

get_saved_dataset(*name: str*) → feast.saved_dataset.SavedDataset
Find a saved dataset in the registry by provided name and create a retrieval job to pull whole dataset from storage (offline store).

If dataset couldn't be found by provided name SavedDatasetNotFound exception will be raised.

Data will be retrieved from globally configured offline store.

Returns SavedDataset with RetrievalJob attached

Raises **SavedDatasetNotFound** –

get_stream_feature_view(*name: str*, *allow_registry_cache: bool = False*) → *feast.stream_feature_view.StreamFeatureView*
Retrieves a stream feature view.

Parameters

- **name** – Name of stream feature view.
- **allow_registry_cache** – (Optional) Whether to allow returning this entity from a cached registry

Returns The specified stream feature view.

Raises **FeatureViewNotFoundException** – The feature view could not be found.

get_validation_reference(*name: str*, *allow_cache: bool = False*) → feast.saved_dataset.ValidationReference
Retrieves a validation reference.

Raises **ValidationReferenceNotFoundException** – The validation reference could not be found.

list_data_sources(*allow_cache: bool = False*) → List[*feast.data_source.DataSource*]
Retrieves the list of data sources from the registry.

Parameters **allow_cache** – Whether to allow returning data sources from a cached registry.

Returns A list of data sources.

list_entities(*allow_cache: bool = False*) → List[*feast.entity.Entity*]
Retrieves the list of entities from the registry.

Parameters **allow_cache** – Whether to allow returning entities from a cached registry.

Returns A list of entities.

list_feature_services() → List[*feast.feature_service.FeatureService*]
Retrieves the list of feature services from the registry.

Returns A list of feature services.

list_feature_views(*allow_cache: bool = False*) → List[*feast.feature_view.FeatureView*]
Retrieves the list of feature views from the registry.

Parameters **allow_cache** – Whether to allow returning entities from a cached registry.

Returns A list of feature views.

list_on_demand_feature_views(*allow_cache: bool = False*) → List[*feast.on_demand_feature_view.OnDemandFeatureView*]
Retrieves the list of on demand feature views from the registry.

Returns A list of on demand feature views.

list_request_feature_views(*allow_cache: bool = False*) → List[feast.request_feature_view.RequestFeatureView]
Retrieves the list of feature views from the registry.

Parameters **allow_cache** – Whether to allow returning entities from a cached registry.

Returns A list of feature views.

list_stream_feature_views(*allow_cache: bool = False*) → List[*feast.stream_feature_view.StreamFeatureView*]
Retrieves the list of stream feature views from the registry.

Returns A list of stream feature views.

materialize(*start_date: datetime.datetime*, *end_date: datetime.datetime*, *feature_views: Optional[List[str]] = None*) → None
Materialize data from the offline store into the online store.

This method loads feature data in the specified interval from either the specified feature views, or all feature views if none are specified, into the online store where it is available for online serving.

Parameters

- **start_date** (*datetime*) – Start date for time range of data to materialize into the online store
- **end_date** (*datetime*) – End date for time range of data to materialize into the online store
- **feature_views** (*List[str]*) – Optional list of feature view names. If selected, will only run materialization for the specified feature views.

Examples

Materialize all features into the online store over the interval from 3 hours ago to 10 minutes ago. >>> from feast import FeatureStore, RepoConfig >>> from datetime import datetime, timedelta >>> fs = FeatureStore(repo_path="project/feature_repo") >>> fs.materialize(... start_date=datetime.utcnow() - timedelta(hours=3), end_date=datetime.utcnow() - timedelta(minutes=10) ...) Materializing... <BLANKLINE> ...

materialize_incremental(*end_date: datetime.datetime*, *feature_views: Optional[List[str]] = None*) → None
Materialize incremental new data from the offline store into the online store.

This method loads incremental new feature data up to the specified end time from either the specified feature views, or all feature views if none are specified, into the online store where it is available for online serving. The start time of the interval materialized is either the most recent end time of a prior materialization or (now - ttl) if no such prior materialization exists.

Parameters

- **end_date** (*datetime*) – End date for time range of data to materialize into the online store
- **feature_views** (*List[str]*) – Optional list of feature view names. If selected, will only run materialization for the specified feature views.

Raises **Exception** – A feature view being materialized does not have a TTL set.

Examples

Materialize all features into the online store up to 5 minutes ago.

```
>>> from feast import FeatureStore, RepoConfig
>>> from datetime import datetime, timedelta
>>> fs = FeatureStore(repo_path="project/feature_repo")
>>> fs.materialize_incremental(end_date=datetime.utcnow() -␣
↪timedelta(minutes=5))
Materializing...

...
```

plan(*desired_repo_contents: feast.repo_contents.RepoContents*) → Tuple[feast.diff.registry_diff.RegistryDiff, feast.diff.infra_diff.InfraDiff, feast.infra.infra_object.Infra]

Dry-run registering objects to metadata store.

The plan method dry-runs registering one or more definitions (e.g., Entity, FeatureView), and produces a list of all the changes the that would be introduced in the feature repo. The changes computed by the plan command are for informational purposes, and are not actually applied to the registry.

Parameters **desired_repo_contents** – The desired repo state.

Raises **ValueError** – The 'objects' parameter could not be parsed properly.

Examples

Generate a plan adding an Entity and a FeatureView.

```
>>> from feast import FeatureStore, Entity, FeatureView, Feature, FileSource,␣
↪RepoConfig
>>> from feast.feature_store import RepoContents
>>> from datetime import timedelta
>>> fs = FeatureStore(repo_path="project/feature_repo")
>>> driver = Entity(name="driver_id", description="driver id")
>>> driver_hourly_stats = FileSource(
...     path="project/feature_repo/data/driver_stats.parquet",
...     timestamp_field="event_timestamp",
...     created_timestamp_column="created",
... )
>>> driver_hourly_stats_view = FeatureView(
...     name="driver_hourly_stats",
```

(continues on next page)

(continued from previous page)

```
...     entities=[driver],
...     ttl=timedelta(seconds=86400 * 1),
...     source=driver_hourly_stats,
... )
>>> registry_diff, infra_diff, new_infra = fs.plan(RepoContents(
...     data_sources=[driver_hourly_stats],
...     feature_views=[driver_hourly_stats_view],
...     on_demand_feature_views=list(),
...     stream_feature_views=list(),
...     request_feature_views=list(),
...     entities=[driver],
...     feature_services=list())) # register entity and feature view
```

property project: *str*
Gets the project of this feature store.

push(*push_source_name: str*, *df: pandas.core.frame.DataFrame*, *allow_registry_cache: bool = True*, *to: feast.data_source.PushMode = PushMode.ONLINE*)
Push features to a push source. This updates all the feature views that have the push source as stream source.

Parameters

- **push_source_name** – The name of the push source we want to push data to.
- **df** – The data being pushed.
- **allow_registry_cache** – Whether to allow cached versions of the registry.
- **to** – Whether to push to online or offline store. Defaults to online store only.

refresh_registry()
Fetches and caches a copy of the feature registry in memory.

Explicitly calling this method allows for direct control of the state of the registry cache. Every time this method is called the complete registry state will be retrieved from the remote registry store backend (e.g., GCS, S3), and the cache timer will be reset. If refresh_registry() is run before get_online_features() is called, then get_online_features() will use the cached registry instead of retrieving (and caching) the registry itself.

Additionally, the TTL for the registry cache can be set to infinity (by setting it to 0), which means that refresh_registry() will become the only way to update the cached registry. If the TTL is set to a value greater than 0, then once the cache becomes stale (more time than the TTL has passed), a new cache will be downloaded synchronously, which may increase latencies if the triggering method is get_online_features().

property registry: *feast.infra.registry.base_registry.BaseRegistry*
Gets the registry of this feature store.

serve(*host: str*, *port: int*, *type_: str*, *no_access_log: bool*, *no_feature_log: bool*) → None
Start the feature consumption server locally on a given port.

serve_transformations(*port: int*) → None
Start the feature transformation server locally on a given port.

serve_ui(*host: str*, *port: int*, *get_registry_dump: Callable*, *registry_ttl_sec: int*) → None
Start the UI server locally

teardown()
Tears down all local and cloud resources for the feature store.

validate_logged_features(*source:* feast.feature_service.FeatureService, *start:* *datetime.datetime*, *end:* *datetime.datetime*, *reference:* *feast.saved_dataset.ValidationReference*, *throw_exception:* *bool* = *True*, *cache_profile:* *bool* = *True*) → Optional[feast.dqm.errors.ValidationFailed]

Load logged features from an offline store and validate them against provided validation reference.

Parameters

- **source** – Logs source object (currently only feature services are supported)
- **start** – lower bound for loading logged features
- **end** – upper bound for loading logged features
- **reference** – validation reference
- **throw_exception** – throw exception or return it as a result
- **cache_profile** – store cached profile in Feast registry

Returns Throw or return (depends on parameter) ValidationFailed exception if validation was not successful or None if successful.

version() → str

Returns the version of the current Feast SDK/CLI.

write_logged_features(*logs:* *Union[pyarrow.lib.Table,* *pathlib.Path]*, *source:* feast.feature_service.FeatureService)

Write logs produced by a source (currently only feature service is supported as a source) to an offline store.

Parameters

- **logs** – Arrow Table or path to parquet dataset directory on disk
- **source** – Object that produces logs

write_to_offline_store(*feature_view_name:* *str*, *df:* *pandas.core.frame.DataFrame*, *allow_registry_cache:* *bool* = *True*, *reorder_columns:* *bool* = *True*)

Persists the dataframe directly into the batch data source for the given feature view.

Fails if the dataframe columns do not match the columns of the batch data source. Optionally reorders the columns of the dataframe to match.

write_to_online_store(*feature_view_name:* *str*, *df:* *pandas.core.frame.DataFrame*, *allow_registry_cache:* *bool* = *True*)

Persists a dataframe to the online store.

Parameters

- **feature_view_name** – The feature view to which the dataframe corresponds.
- **df** – The dataframe to be persisted.
- **allow_registry_cache** (*optional*) – Whether to allow retrieving feature views from a cached registry.

CHAPTER

TWO

CONFIG

class feast.repo_config.**RepoConfig**(*, *registry: Union[pydantic.types.StrictStr, feast.repo_config.RegistryConfig] = 'data/registry.db'*, *project: pydantic.types.StrictStr*, *provider: pydantic.types.StrictStr*, *feature_server: Optional[Any] = None*, *flags: Any = None*, *repo_path: Optional[pathlib.Path] = None*, *go_feature_serving: Optional[bool] = False*, *go_feature_retrieval: Optional[bool] = False*, *entity_key_serialization_version: pydantic.types.StrictInt = 1*, *coerce_tz_aware: Optional[bool] = True*, ***data: Any*)

Repo config. Typically loaded from *feature_store.yaml*

coerce_tz_aware: Optional[bool]
If True, coerces entity_df timestamp columns to be timezone aware (to UTC by default).

entity_key_serialization_version: pydantic.types.StrictInt
This version is used to control what serialization scheme is used when writing data to the online store. A value <= 1 uses the serialization scheme used by feast up to Feast 0.22. A value of 2 uses a newer serialization scheme, supported as of Feast 0.23. The main difference between the two scheme is that the serialization scheme v1 stored *long* values as `int`s, which would result in errors trying to serialize a range of values. v2 fixes this error, but v1 is kept around to ensure backwards compatibility - specifically the ability to read feature values for entities that have already been written into the online store.

Type Entity key serialization version

feature_server: Optional[Any]
Feature server configuration (optional depending on provider)

Type FeatureServerConfig

flags: Any
Feature flags for experimental features

Type Flags (deprecated field)

go_feature_retrieval: Optional[bool]
If True, use the embedded Go code to retrieve features instead of the Python SDK.

go_feature_serving: Optional[bool]
If True, use the Go feature server instead of the Python feature server.

project: pydantic.types.StrictStr
Feast project id. This can be any alphanumeric string up to 16 characters. You can have multiple independent feature repositories deployed to the same cloud provider account, as long as they have different project ids.

Type str

provider: pydantic.types.StrictStr
local or gcp or aws

Type str

registry: Union[pydantic.types.StrictStr, *feast.repo_config.RegistryConfig*]
Path to metadata store. Can be a local path, or remote object storage path, e.g. a GCS URI

Type str

class feast.repo_config.**RegistryConfig**(*, *registry_type: pydantic.types.StrictStr = 'file'*, *registry_store_type: Optional[pydantic.types.StrictStr] = None*, *path: pydantic.types.StrictStr*, *cache_ttl_seconds: pydantic.types.StrictInt = 600*, *s3_additional_kwargs: Optional[Dict[str, str]] = None*, ***extra_data: Any*)
Metadata Store Configuration. Configuration that relates to reading from and writing to the Feast registry.

cache_ttl_seconds: pydantic.types.StrictInt
The cache TTL is the amount of time registry state will be cached in memory. If this TTL is exceeded then the registry will be refreshed when any feature store method asks for access to registry state. The TTL can be set to infinity by setting TTL to 0 seconds, which means the cache will only be loaded once and will never expire. Users can manually refresh the cache by calling feature_store.refresh_registry()

Type int

path: pydantic.types.StrictStr
Path to metadata store. Can be a local path, or remote object storage path, e.g. a GCS URI

Type str

registry_store_type: Optional[pydantic.types.StrictStr]
Provider name or a class name that implements RegistryStore.

Type str

registry_type: pydantic.types.StrictStr
Provider name or a class name that implements RegistryStore. If specified, registry_store_type should be redundant.

Type str

s3_additional_kwargs: Optional[Dict[str, str]]
Extra arguments to pass to boto3 when writing the registry file to S3.

Type Dict[str, str]

CHAPTER

THREE

DATA SOURCE

class feast.data_source.**DataSource**(*, *name: str*, *timestamp_field: Optional[str] = None*, *created_timestamp_column: Optional[str] = None*, *field_mapping: Optional[Dict[str, str]] = None*, *description: Optional[str] = ''*, *tags: Optional[Dict[str, str]] = None*, *owner: Optional[str] = ''*, *date_partition_column: Optional[str] = None*)

DataSource that can be used to source features.

Parameters

- **name** – Name of data source, which should be unique within a project
- **timestamp_field** (*optional*) – Event timestamp field used for point-in-time joins of feature values.
- **created_timestamp_column** (*optional*) – Timestamp column indicating when the row was created, used for deduplicating rows.
- **field_mapping** (*optional*) – A dictionary mapping of column names in this data source to feature names in a feature table or view. Only used for feature columns, not entity or timestamp columns.
- **description** (*optional*) –
- **tags** (*optional*) – A dictionary of key-value pairs to store arbitrary metadata.
- **owner** (*optional*) – The owner of the data source, typically the email of the primary maintainer.
- **timestamp_field** – Event timestamp field used for point in time joins of feature values.
- **date_partition_column** (*optional*) – Timestamp column used for partitioning. Not supported by all offline stores.

abstract static from_proto(*data_source: feast.core.DataSource_pb2.DataSource*) → Any

Converts data source config in protobuf spec to a DataSource class object.

Parameters **data_source** – A protobuf representation of a DataSource.

Returns A DataSource class object.

Raises **ValueError** – The type of DataSource could not be identified.

get_table_column_names_and_types(*config:* feast.repo_config.RepoConfig) → Iterable[Tuple[str, str]]

Returns the list of column names and raw column types.

Parameters **config** – Configuration object used to configure a feature store.

get_table_query_string() → str

Returns a string that can directly be used to reference this table in SQL.

abstract static source_datatype_to_feast_value_type() → Callable[[str], feast.value_type.ValueType]
Returns the callable method that returns Feast type given the raw column type.

abstract to_proto() → feast.core.DataSource_pb2.DataSource
Converts a DataSourceProto object to its protobuf representation.

validate(*config:* feast.repo_config.RepoConfig)
Validates the underlying data source.

Parameters **config** – Configuration object used to configure a feature store.

3.1 File Source

class feast.infra.offline_stores.file_source.**FileSource**(*, *path: str*, *name: Optional[str] = ''*, *event_timestamp_column: Optional[str] = ''*, *file_format: Optional[feast.data_format.FileFormat] = None*, *created_timestamp_column: Optional[str] = ''*, *field_mapping: Optional[Dict[str, str]] = None*, *s3_endpoint_override: Optional[str] = None*, *description: Optional[str] = ''*, *tags: Optional[Dict[str, str]] = None*, *owner: Optional[str] = ''*, *timestamp_field: Optional[str] = ''*)

property file_format: Optional[feast.data_format.FileFormat]
Returns the file format of this file data source.

static from_proto(*data_source: feast.core.DataSource_pb2.DataSource*)
Converts data source config in protobuf spec to a DataSource class object.

Parameters **data_source** – A protobuf representation of a DataSource.

Returns A DataSource class object.

Raises **ValueError** – The type of DataSource could not be identified.

get_table_column_names_and_types(*config:* feast.repo_config.RepoConfig) → Iterable[Tuple[str, str]]
Returns the list of column names and raw column types.

Parameters **config** – Configuration object used to configure a feature store.

get_table_query_string() → str
Returns a string that can directly be used to reference this table in SQL.

property path: str
Returns the path of this file data source.

property s3_endpoint_override: Optional[str]
Returns the s3 endpoint override of this file data source.

static source_datatype_to_feast_value_type() → Callable[[str], feast.value_type.ValueType]
Returns the callable method that returns Feast type given the raw column type.

to_proto() → feast.core.DataSource_pb2.DataSource
Converts a DataSourceProto object to its protobuf representation.

validate(*config:* feast.repo_config.RepoConfig)
Validates the underlying data source.

Parameters **config** – Configuration object used to configure a feature store.

3.2 Snowflake Source

class feast.infra.offline_stores.snowflake_source.**SnowflakeSource**(*, *name: Optional[str] = None*, *timestamp_field: Optional[str] = ''*, *database: Optional[str] = None*, *warehouse: Optional[str] = None*, *schema: Optional[str] = None*, *table: Optional[str] = None*, *query: Optional[str] = None*, *created_timestamp_column: Optional[str] = ''*, *field_mapping: Optional[Dict[str, str]] = None*, *description: Optional[str] = ''*, *tags: Optional[Dict[str, str]] = None*, *owner: Optional[str] = ''*)

property database
Returns the database of this snowflake source.

static from_proto(*data_source: feast.core.DataSource_pb2.DataSource*)
Creates a SnowflakeSource from a protobuf representation of a SnowflakeSource.

Parameters **data_source** – A protobuf representation of a SnowflakeSource

Returns A SnowflakeSource object based on the data_source protobuf.

get_table_column_names_and_types(*config:* feast.repo_config.RepoConfig) → Iterable[Tuple[str, str]]
Returns a mapping of column names to types for this snowflake source.

Parameters **config** – A RepoConfig describing the feature repo

get_table_query_string() → str
Returns a string that can directly be used to reference this table in SQL.

property query
Returns the snowflake options of this snowflake source.

property schema
Returns the schema of this snowflake source.

static source_datatype_to_feast_value_type() → Callable[[str], feast.value_type.ValueType]
Returns the callable method that returns Feast type given the raw column type.

property table
Returns the table of this snowflake source.

to_proto() → feast.core.DataSource_pb2.DataSource
Converts a SnowflakeSource object to its protobuf representation.

Returns A DataSourceProto object.

validate(*config:* feast.repo_config.RepoConfig)
Validates the underlying data source.

Parameters **config** – Configuration object used to configure a feature store.

property warehouse
Returns the warehouse of this snowflake source.

3.3 BigQuery Source

class feast.infra.offline_stores.bigquery_source.**BigQuerySource**(*, *name: Optional[str] = None*, *timestamp_field: Optional[str] = None*, *table: Optional[str] = None*, *created_timestamp_column: Optional[str] = ''*, *field_mapping: Optional[Dict[str, str]] = None*, *query: Optional[str] = None*, *description: Optional[str] = ''*, *tags: Optional[Dict[str, str]] = None*, *owner: Optional[str] = ''*)

static from_proto(*data_source: feast.core.DataSource_pb2.DataSource*)
Converts data source config in protobuf spec to a DataSource class object.

Parameters **data_source** – A protobuf representation of a DataSource.

Returns A DataSource class object.

Raises **ValueError** – The type of DataSource could not be identified.

get_table_column_names_and_types(*config:* feast.repo_config.RepoConfig) → Iterable[Tuple[str, str]]
Returns the list of column names and raw column types.

Parameters **config** – Configuration object used to configure a feature store.

get_table_query_string() → str
Returns a string that can directly be used to reference this table in SQL

static source_datatype_to_feast_value_type() → Callable[[str], feast.value_type.ValueType]
Returns the callable method that returns Feast type given the raw column type.

to_proto() → feast.core.DataSource_pb2.DataSource
Converts a DataSourceProto object to its protobuf representation.

validate(*config:* feast.repo_config.RepoConfig)
Validates the underlying data source.

Parameters **config** – Configuration object used to configure a feature store.

3.4 Redshift Source

class feast.infra.offline_stores.redshift_source.**RedshiftSource**(*, *name: Optional[str] = None*, *timestamp_field: Optional[str] = ''*, *table: Optional[str] = None*, *schema: Optional[str] = None*, *created_timestamp_column: Optional[str] = ''*, *field_mapping: Optional[Dict[str, str]] = None*, *query: Optional[str] = None*, *description: Optional[str] = ''*, *tags: Optional[Dict[str, str]] = None*, *owner: Optional[str] = ''*, *database: Optional[str] = ''*)

property database
Returns the Redshift database of this Redshift source.

static from_proto(*data_source: feast.core.DataSource_pb2.DataSource*)
Creates a RedshiftSource from a protobuf representation of a RedshiftSource.

Parameters **data_source** – A protobuf representation of a RedshiftSource

Returns A RedshiftSource object based on the data_source protobuf.

get_table_column_names_and_types(*config:* feast.repo_config.RepoConfig) → Iterable[Tuple[str, str]]
Returns a mapping of column names to types for this Redshift source.

Parameters **config** – A RepoConfig describing the feature repo

get_table_query_string() → str
Returns a string that can directly be used to reference this table in SQL.

property query
Returns the Redshift query of this Redshift source.

property schema
Returns the schema of this Redshift source.

static source_datatype_to_feast_value_type() → Callable[[str], feast.value_type.ValueType]
Returns the callable method that returns Feast type given the raw column type.

property table
Returns the table of this Redshift source.

to_proto() → feast.core.DataSource_pb2.DataSource
Converts a RedshiftSource object to its protobuf representation.

Returns A DataSourceProto object.

validate(*config:* feast.repo_config.RepoConfig)
Validates the underlying data source.

Parameters **config** – Configuration object used to configure a feature store.

3.5 Spark Source

class feast.infra.offline_stores.contrib.spark_offline_store.spark_source.**SparkSource**(*, *name: Optional[str] = None*, *table: Optional[str] = None*, *query: Optional[str] = None*, *path: Optional[str] = None*, *file_format: Optional[str] = None*, *event_timestamp_colu Optional[str] = None*, *created_timestamp_colum Optional[str] = None*, *field_mapping: Optional[Dict[str, str]] = None*, *description: Optional[str] = ''*, *tags: Optional[Dict[str, str]] = None*, *owner:*

property file_format
Returns the file format of this feature data source.

static from_proto(*data_source: feast.core.DataSource_pb2.DataSource*) → Any
Converts data source config in protobuf spec to a DataSource class object.

Parameters **data_source** – A protobuf representation of a DataSource.

Returns A DataSource class object.

Raises **ValueError** – The type of DataSource could not be identified.

get_table_column_names_and_types(*config:* feast.repo_config.RepoConfig) → Iterable[Tuple[str, str]]
Returns the list of column names and raw column types.

Parameters **config** – Configuration object used to configure a feature store.

get_table_query_string() → str
Returns a string that can directly be used to reference this table in SQL

property path
Returns the path of the spark data source file.

property query
Returns the query of this feature data source

static source_datatype_to_feast_value_type() → Callable[[str], feast.value_type.ValueType]
Returns the callable method that returns Feast type given the raw column type.

property table
Returns the table of this feature data source

to_proto() → feast.core.DataSource_pb2.DataSource
Converts a DataSourceProto object to its protobuf representation.

validate(*config:* feast.repo_config.RepoConfig)
Validates the underlying data source.

Parameters **config** – Configuration object used to configure a feature store.

3.6 Trino Source

3.7 PostgreSQL Source

class feast.infra.offline_stores.contrib.postgres_offline_store.postgres_source.**PostgreSQLSource**(*name: Optional[st = None, query: Optional[st = None, table: Optional[st = None, timestamp_fie Optional[st = '', created_tim Optional[st = '', field_ma Optional[D str]] = None, description: Optional[st = '', tags: Optional[D str]] = None, owner: Optional[st = ''*)

static from_proto(*data_source: feast.core.DataSource_pb2.DataSource*)
Converts data source config in protobuf spec to a DataSource class object.

Parameters **data_source** – A protobuf representation of a DataSource.

Returns A DataSource class object.

Raises **ValueError** – The type of DataSource could not be identified.

get_table_column_names_and_types(*config:* feast.repo_config.RepoConfig) → Iterable[Tuple[str, str]]
Returns the list of column names and raw column types.

Parameters **config** – Configuration object used to configure a feature store.

get_table_query_string() → str
Returns a string that can directly be used to reference this table in SQL.

static source_datatype_to_feast_value_type() → Callable[[str], feast.value_type.ValueType]
Returns the callable method that returns Feast type given the raw column type.

to_proto() → feast.core.DataSource_pb2.DataSource
Converts a DataSourceProto object to its protobuf representation.

validate(*config:* feast.repo_config.RepoConfig)
Validates the underlying data source.

Parameters **config** – Configuration object used to configure a feature store.

3.8 Request Source

class feast.data_source.**RequestSource**(*, *name: str*, *schema: List[*feast.field.Field*]*, *description: Optional[str] = ''*, *tags: Optional[Dict[str, str]] = None*, *owner: Optional[str] = ''*)
RequestSource that can be used to provide input features for on demand transforms

name
Name of the request data source

Type str

schema
Schema mapping from the input feature name to a ValueType

Type List[*feast.field.Field*]

description
A human-readable description.

Type str

tags
A dictionary of key-value pairs to store arbitrary metadata.

Type Dict[str, str]

owner
The owner of the request data source, typically the email of the primary maintainer.

Type str

static from_proto(*data_source: feast.core.DataSource_pb2.DataSource*)
Converts data source config in protobuf spec to a DataSource class object.

Parameters **data_source** – A protobuf representation of a DataSource.

Returns A DataSource class object.

Raises **ValueError** – The type of DataSource could not be identified.

get_table_column_names_and_types(*config:* feast.repo_config.RepoConfig) → Iterable[Tuple[str, str]]
Returns the list of column names and raw column types.

Parameters **config** – Configuration object used to configure a feature store.

get_table_query_string() → str
Returns a string that can directly be used to reference this table in SQL.

static source_datatype_to_feast_value_type() → Callable[[str], feast.value_type.ValueType]
Returns the callable method that returns Feast type given the raw column type.

to_proto() → feast.core.DataSource_pb2.DataSource
Converts a DataSourceProto object to its protobuf representation.

validate(*config:* feast.repo_config.RepoConfig)
Validates the underlying data source.

Parameters **config** – Configuration object used to configure a feature store.

3.9 Push Source

class feast.data_source.**PushSource**(*, *name: str*, *batch_source:* feast.data_source.DataSource, *description: Optional[str] = ''*, *tags: Optional[Dict[str, str]] = None*, *owner: Optional[str] = ''*)
A source that can be used to ingest features on request

static from_proto(*data_source: feast.core.DataSource_pb2.DataSource*)
Converts data source config in protobuf spec to a DataSource class object.

Parameters **data_source** – A protobuf representation of a DataSource.

Returns A DataSource class object.

Raises **ValueError** – The type of DataSource could not be identified.

get_table_column_names_and_types(*config:* feast.repo_config.RepoConfig) → Iterable[Tuple[str, str]]
Returns the list of column names and raw column types.

Parameters **config** – Configuration object used to configure a feature store.

get_table_query_string() → str
Returns a string that can directly be used to reference this table in SQL.

static source_datatype_to_feast_value_type() → Callable[[str], feast.value_type.ValueType]
Returns the callable method that returns Feast type given the raw column type.

to_proto() → feast.core.DataSource_pb2.DataSource
Converts a DataSourceProto object to its protobuf representation.

validate(*config:* feast.repo_config.RepoConfig)
Validates the underlying data source.

Parameters **config** – Configuration object used to configure a feature store.

3.10 Kafka Source

class feast.data_source.**KafkaSource**(*, *name: str*, *timestamp_field: str*, *message_format: feast.data_format.StreamFormat*, *bootstrap_servers: Optional[str] = None*, *kafka_bootstrap_servers: Optional[str] = None*, *topic: Optional[str] = None*, *created_timestamp_column: Optional[str] = ''*, *field_mapping: Optional[Dict[str, str]] = None*, *description: Optional[str] = ''*, *tags: Optional[Dict[str, str]] = None*, *owner: Optional[str] = ''*, *batch_source: Optional[feast.data_source.DataSource] = None*, *watermark_delay_threshold: Optional[datetime.timedelta] = None*)

static from_proto(*data_source: feast.core.DataSource_pb2.DataSource*)
Converts data source config in protobuf spec to a DataSource class object.

Parameters **data_source** – A protobuf representation of a DataSource.

Returns A DataSource class object.

Raises **ValueError** – The type of DataSource could not be identified.

get_table_column_names_and_types(*config: feast.repo_config.RepoConfig*) → Iterable[Tuple[str, str]]
Returns the list of column names and raw column types.

Parameters **config** – Configuration object used to configure a feature store.

get_table_query_string() → str
Returns a string that can directly be used to reference this table in SQL.

static source_datatype_to_feast_value_type() → Callable[[str], feast.value_type.ValueType]
Returns the callable method that returns Feast type given the raw column type.

to_proto() → feast.core.DataSource_pb2.DataSource
Converts a DataSourceProto object to its protobuf representation.

validate(*config: feast.repo_config.RepoConfig*)
Validates the underlying data source.

Parameters **config** – Configuration object used to configure a feature store.

3.11 Kinesis Source

class feast.data_source.**KinesisSource**(*, *name: str*, *record_format: feast.data_format.StreamFormat*, *region: str*, *stream_name: str*, *timestamp_field: Optional[str] = ''*, *created_timestamp_column: Optional[str] = ''*, *field_mapping: Optional[Dict[str, str]] = None*, *description: Optional[str] = ''*, *tags: Optional[Dict[str, str]] = None*, *owner: Optional[str] = ''*, *batch_source: Optional[feast.data_source.DataSource] = None*)

static from_proto(*data_source: feast.core.DataSource_pb2.DataSource*)
Converts data source config in protobuf spec to a DataSource class object.

Parameters **data_source** – A protobuf representation of a DataSource.

Returns A DataSource class object.

Raises **ValueError** – The type of DataSource could not be identified.

get_table_column_names_and_types(*config:* feast.repo_config.RepoConfig) → Iterable[Tuple[str, str]]
Returns the list of column names and raw column types.

Parameters **config** – Configuration object used to configure a feature store.

get_table_query_string() → str
Returns a string that can directly be used to reference this table in SQL.

static source_datatype_to_feast_value_type() → Callable[[str], feast.value_type.ValueType]
Returns the callable method that returns Feast type given the raw column type.

to_proto() → feast.core.DataSource_pb2.DataSource
Converts a DataSourceProto object to its protobuf representation.

validate(*config:* feast.repo_config.RepoConfig)
Validates the underlying data source.

Parameters **config** – Configuration object used to configure a feature store.

CHAPTER

FOUR

ENTITY

class feast.entity.**Entity**(*, *name: str*, *join_keys: Optional[List[str]] = None*, *value_type: Optional[feast.value_type.ValueType] = None*, *description: str = ''*, *tags: Optional[Dict[str, str]] = None*, *owner: str = ''*)

An entity defines a collection of entities for which features can be defined. An entity can also contain associated metadata.

name
The unique name of the entity.

Type str

value_type
The type of the entity, such as string or float.

Type feast.value_type.ValueType

join_key
A property that uniquely identifies different entities within the collection. The join_key property is typically used for joining entities with their associated features. If not specified, defaults to the name.

Type str

description
A human-readable description.

Type str

tags
A dictionary of key-value pairs to store arbitrary metadata.

Type Dict[str, str]

owner
The owner of the entity, typically the email of the primary maintainer.

Type str

created_timestamp
The time when the entity was created.

Type Optional[datetime.datetime]

last_updated_timestamp
The time when the entity was last updated.

Type Optional[datetime.datetime]

classmethod from_proto(*entity_proto: feast.core.Entity_pb2.Entity*)
Creates an entity from a protobuf representation of an entity.

Parameters **entity_proto** – A protobuf representation of an entity.

Returns An Entity object based on the entity protobuf.

is_valid()
Validates the state of this entity locally.

Raises **ValueError** – The entity does not have a name or does not have a type.

to_proto() → feast.core.Entity_pb2.Entity
Converts an entity object to its protobuf representation.

Returns An EntityProto protobuf.

CHAPTER

FIVE

FEATURE VIEW

class feast.base_feature_view.**BaseFeatureView**(*, *name: str*, *features: Optional[List[*feast.field.Field*]] = None*, *description: str = ''*, *tags: Optional[Dict[str, str]] = None*, *owner: str = ''*)

A BaseFeatureView defines a logical group of features.

name
The unique name of the base feature view.

Type str

features
The list of features defined as part of this base feature view.

Type List[*feast.field.Field*]

description
A human-readable description.

Type str

tags
A dictionary of key-value pairs to store arbitrary metadata.

Type Dict[str, str]

owner
The owner of the base feature view, typically the email of the primary maintainer.

Type str

projection
The feature view projection storing modifications to be applied to this base feature view at retrieval time.

Type feast.feature_view_projection.FeatureViewProjection

created_timestamp
The time when the base feature view was created.

Type Optional[datetime.datetime]

last_updated_timestamp
The time when the base feature view was last updated.

Type Optional[datetime.datetime]

ensure_valid()
Validates the state of this feature view locally.

Raises `ValueError` – The feature view is invalid.

set_projection(*feature_view_projection: feast.feature_view_projection.FeatureViewProjection*) → None
Sets the feature view projection of this base feature view to the given projection.

Parameters **feature_view_projection** – The feature view projection to be set.

Raises **ValueError** – The name or features of the projection do not match.

with_name(*name: str*)
Returns a renamed copy of this base feature view. This renamed copy should only be used for query operations and will not modify the underlying base feature view.

Parameters **name** – The name to assign to the copy.

with_projection(*feature_view_projection: feast.feature_view_projection.FeatureViewProjection*)
Returns a copy of this base feature view with the feature view projection set to the given projection.

Parameters **feature_view_projection** – The feature view projection to assign to the copy.

Raises **ValueError** – The name or features of the projection do not match.

5.1 Feature View

class feast.feature_view.**FeatureView**(*, *name: str*, *source:* feast.data_source.DataSource, *schema: Optional[List[*feast.field.Field*]] = None*, *entities: List[*feast.entity.Entity*] = None*, *ttl: Optional[datetime.timedelta] = datetime.timedelta(0)*, *online: bool = True*, *description: str = ''*, *tags: Optional[Dict[str, str]] = None*, *owner: str = ''*)
A FeatureView defines a logical group of features.

name
The unique name of the feature view.

Type str

entities
The list of names of entities that this feature view is associated with.

Type List[str]

ttl
The amount of time this group of features lives. A ttl of 0 indicates that this group of features lives forever. Note that large ttl's or a ttl of 0 can result in extremely computationally intensive queries.

Type Optional[datetime.timedelta]

batch_source
The batch source of data where this group of features is stored. This is optional ONLY if a push source is specified as the stream_source, since push sources contain their own batch sources.

Type *feast.data_source.DataSource*

stream_source
The stream source of data where this group of features is stored.

Type Optional[*feast.data_source.DataSource*]

schema
The schema of the feature view, including feature, timestamp, and entity columns. If not specified, can be inferred from the underlying data source.

Type List[*feast.field.Field*]

entity_columns
The list of entity columns contained in the schema. If not specified, can be inferred from the underlying data source.

Type List[*feast.field.Field*]

features
The list of feature columns contained in the schema. If not specified, can be inferred from the underlying data source.

Type List[*feast.field.Field*]

online
A boolean indicating whether online retrieval is enabled for this feature view.

Type bool

description
A human-readable description.

Type str

tags
A dictionary of key-value pairs to store arbitrary metadata.

Type Dict[str, str]

owner
The owner of the feature view, typically the email of the primary maintainer.

Type str

ensure_valid()
Validates the state of this feature view locally.

Raises **ValueError** – The feature view does not have a name or does not have entities.

classmethod from_proto(*feature_view_proto: feast.core.FeatureView_pb2.FeatureView*)
Creates a feature view from a protobuf representation of a feature view.

Parameters **feature_view_proto** – A protobuf representation of a feature view.

Returns A FeatureViewProto object based on the feature view protobuf.

property join_keys: List[str]
Returns a list of all the join keys.

property most_recent_end_time: Optional[datetime.datetime]
Retrieves the latest time up to which the feature view has been materialized.

Returns The latest time, or None if the feature view has not been materialized.

to_proto() → feast.core.FeatureView_pb2.FeatureView
Converts a feature view object to its protobuf representation.

Returns A FeatureViewProto protobuf.

with_join_key_map(*join_key_map: Dict[str, str]*)
Returns a copy of this feature view with the join key map set to the given map. This join_key mapping operation is only used as part of query operations and will not modify the underlying FeatureView.

Parameters **join_key_map** – A map of join keys in which the left is the join_key that corresponds with the feature data and the right corresponds with the entity data.

Examples

Join a location feature data table to both the origin column and destination column of the entity data.

temperatures_feature_service = FeatureService(name="temperatures", features=[

location_stats_feature_view .with_name("origin_stats") .with_join_key_map(

{"location_id": "origin_id"}

),

location_stats_feature_view .with_name("destination_stats") .with_join_key_map(

{"location_id": "destination_id"}

),

],

)

5.2 On Demand Feature View

class feast.on_demand_feature_view.**OnDemandFeatureView**(*, *name: str*, *schema: List[*feast.field.Field*]*, *sources: List[Union[*feast.feature_view.FeatureView*,* feast.data_source.RequestSource*, feast.feature_view_projection.FeatureViewProjection]]*, *udf: function*, *udf_string: str = ''*, *description: str = ''*, *tags: Optional[Dict[str, str]] = None*, *owner: str = ''*)

[Experimental] An OnDemandFeatureView defines a logical group of features that are generated by applying a transformation on a set of input sources, such as feature views and request data sources.

name

The unique name of the on demand feature view.

Type str

features

The list of features in the output of the on demand feature view.

Type List[*feast.field.Field*]

source_feature_view_projections

A map from input source names to actual input sources with type FeatureViewProjection.

Type Dict[str, feast.feature_view_projection.FeatureViewProjection]

source_request_sources

A map from input source names to the actual input sources with type RequestSource.

Type Dict[str, *feast.data_source.RequestSource*]

udf

The user defined transformation function, which must take pandas dataframes as inputs.

Type function

description

A human-readable description.

Type str

tags
A dictionary of key-value pairs to store arbitrary metadata.

Type Dict[str, str]

owner
The owner of the on demand feature view, typically the email of the primary maintainer.

Type str

classmethod from_proto(*on_demand_feature_view_proto: feast.core.OnDemandFeatureView_pb2.OnDemandFeatureView*)
Creates an on demand feature view from a protobuf representation.

Parameters **on_demand_feature_view_proto** – A protobuf representation of an on-demand feature view.

Returns A OnDemandFeatureView object based on the on-demand feature view protobuf.

infer_features()
Infers the set of features associated to this feature view from the input source.

Raises **RegistryInferenceFailure** – The set of features could not be inferred.

to_proto() → feast.core.OnDemandFeatureView_pb2.OnDemandFeatureView
Converts an on demand feature view object to its protobuf representation.

Returns A OnDemandFeatureViewProto protobuf.

5.3 Batch Feature View

class feast.batch_feature_view.**BatchFeatureView**(*, *name: str*, *source:* feast.data_source.DataSource, *entities: Optional[Union[List[*feast.entity.Entity*], List[str]]] = None*, *ttl: Optional[datetime.timedelta] = None*, *tags: Optional[Dict[str, str]] = None*, *online: bool = True*, *description: str = ''*, *owner: str = ''*, *schema: Optional[List[*feast.field.Field*]] = None*)
A batch feature view defines a logical group of features that has only a batch data source.

name
The unique name of the batch feature view.

Type str

entities
List of entities or entity join keys.

Type List[str]

ttl
The amount of time this group of features lives. A ttl of 0 indicates that this group of features lives forever. Note that large ttl's or a ttl of 0 can result in extremely computationally intensive queries.

Type Optional[datetime.timedelta]

schema
The schema of the feature view, including feature, timestamp, and entity columns. If not specified, can be inferred from the underlying data source.

Type List[*feast.field.Field*]

source
The batch source of data where this group of features is stored.

Type *feast.data_source.DataSource*

online
A boolean indicating whether online retrieval is enabled for this feature view.

Type bool

description
A human-readable description.

Type str

tags
A dictionary of key-value pairs to store arbitrary metadata.

Type Dict[str, str]

owner
The owner of the batch feature view, typically the email of the primary maintainer.

Type str

5.4 Stream Feature View

class feast.stream_feature_view.**StreamFeatureView**(*, *name: str*, *source:* feast.data_source.DataSource, *entities: Optional[Union[List[*feast.entity.Entity*], List[str]]] = None*, *ttl: datetime.timedelta = datetime.timedelta(0)*, *tags: Optional[Dict[str, str]] = None*, *online: Optional[bool] = True*, *description: Optional[str] = ''*, *owner: Optional[str] = ''*, *schema: Optional[List[*feast.field.Field*]] = None*, *aggregations: Optional[List[feast.aggregation.Aggregation]] = None*, *mode: Optional[str] = 'spark'*, *timestamp_field: Optional[str] = ''*, *udf: Optional[function] = None*)

A stream feature view defines a logical group of features that has both a stream data source and a batch data source.

name
The unique name of the stream feature view.

Type str

entities
List of entities or entity join keys.

Type List[str]

ttl
The amount of time this group of features lives. A ttl of 0 indicates that this group of features lives forever. Note that large ttl's or a ttl of 0 can result in extremely computationally intensive queries.

Type Optional[datetime.timedelta]

schema
The schema of the feature view, including feature, timestamp, and entity columns. If not specified, can be inferred from the underlying data source.

Type List[*feast.field.Field*]

source
The stream source of data where this group of features is stored.

Type *feast.data_source.DataSource*

aggregations
List of aggregations registered with the stream feature view.

Type List[feast.aggregation.Aggregation]

mode
The mode of execution.

Type str

timestamp_field
Must be specified if aggregations are specified. Defines the timestamp column on which to aggregate windows.

Type str

online
A boolean indicating whether online retrieval is enabled for this feature view.

Type bool

description
A human-readable description.

Type str

tags
A dictionary of key-value pairs to store arbitrary metadata.

Type Dict[str, str]

owner
The owner of the stream feature view, typically the email of the primary maintainer.

Type str

udf
The user defined transformation function. This transformation function should have all of the corresponding imports imported within the function.

Type Optional[function]

classmethod from_proto(*sfv_proto*)
Creates a feature view from a protobuf representation of a feature view.

Parameters **feature_view_proto** – A protobuf representation of a feature view.

Returns A FeatureViewProto object based on the feature view protobuf.

to_proto()
Converts a feature view object to its protobuf representation.

Returns A FeatureViewProto protobuf.

CHAPTER

SIX

FIELD

class feast.field.**Field**(*, *name: str*, *dtype: Union[feast.types.ComplexFeastType, feast.types.PrimitiveFeastType]*, *tags: Optional[Dict[str, str]] = None*)

A Field represents a set of values with the same structure.

name

The name of the field.

Type str

dtype

The type of the field, such as string or float.

Type Union[feast.types.ComplexFeastType, feast.types.PrimitiveFeastType]

tags

User-defined metadata in dictionary form.

Type Dict[str, str]

classmethod from_feature(*feature: feast.feature.Feature*)

Creates a Field object from a Feature object.

Parameters **feature** – Feature object to convert.

classmethod from_proto(*field_proto: feast.core.Feature_pb2.FeatureSpecV2*)

Creates a Field object from a protobuf representation.

Parameters **field_proto** – FieldProto protobuf object

to_proto() → feast.core.Feature_pb2.FeatureSpecV2

Converts a Field object to its protobuf representation.

CHAPTER

SEVEN

FEATURE SERVICE

class feast.feature_service.**FeatureService**(*, *name: str*, *features: List[Union[feast.feature_view.FeatureView, feast.on_demand_feature_view.OnDemandFeatureView]]*, *tags: Dict[str, str] = None*, *description: str = ''*, *owner: str = ''*, *logging_config: Optional[feast.feature_logging.LoggingConfig] = None*)

A feature service defines a logical group of features from one or more feature views. This group of features can be retrieved together during training or serving.

name
The unique name of the feature service.

Type str

feature_view_projections
A list containing feature views and feature view projections, representing the features in the feature service.

Type List[feast.feature_view_projection.FeatureViewProjection]

description
A human-readable description.

Type str

tags
A dictionary of key-value pairs to store arbitrary metadata.

Type Dict[str, str]

owner
The owner of the feature service, typically the email of the primary maintainer.

Type str

created_timestamp
The time when the feature service was created.

Type Optional[datetime.datetime]

last_updated_timestamp
The time when the feature service was last updated.

Type Optional[datetime.datetime]

classmethod from_proto(*feature_service_proto: feast.core.FeatureService_pb2.FeatureService*)
Converts a FeatureServiceProto to a FeatureService object.

Parameters **feature_service_proto** – A protobuf representation of a FeatureService.

infer_features(*fvs_to_update: Dict[str, feast.feature_view.FeatureView]*)
Infers the features for the projections of this feature service, and updates this feature service in place.

This method is necessary since feature services may rely on feature views which require feature inference.

Parameters **fvs_to_update** – A mapping of feature view names to corresponding feature views that contains all the feature views necessary to run inference.

to_proto() → feast.core.FeatureService_pb2.FeatureService
Converts a feature service to its protobuf representation.

Returns A FeatureServiceProto protobuf.

CHAPTER

EIGHT

REGISTRY

class feast.infra.registry.base_registry.**BaseRegistry**

The interface that Feast uses to apply, list, retrieve, and delete Feast objects (e.g. entities, feature views, and data sources).

abstract apply_data_source(*data_source:* feast.data_source.DataSource, *project: str*, *commit: bool = True*)

Registers a single data source with Feast

Parameters

- **data_source** – A data source that will be registered
- **project** – Feast project that this data source belongs to
- **commit** – Whether to immediately commit to the registry

abstract apply_entity(*entity:* feast.entity.Entity, *project: str*, *commit: bool = True*)

Registers a single entity with Feast

Parameters

- **entity** – Entity that will be registered
- **project** – Feast project that this entity belongs to
- **commit** – Whether the change should be persisted immediately

abstract apply_feature_service(*feature_service:* feast.feature_service.FeatureService, *project: str*, *commit: bool = True*)

Registers a single feature service with Feast

Parameters

- **feature_service** – A feature service that will be registered
- **project** – Feast project that this entity belongs to

abstract apply_feature_view(*feature_view:* feast.base_feature_view.BaseFeatureView, *project: str*, *commit: bool = True*)

Registers a single feature view with Feast

Parameters

- **feature_view** – Feature view that will be registered
- **project** – Feast project that this feature view belongs to
- **commit** – Whether the change should be persisted immediately

abstract apply_materialization(*feature_view: feast.feature_view.FeatureView*, *project: str*, *start_date: datetime.datetime*, *end_date: datetime.datetime*, *commit: bool = True*)

Updates materialization intervals tracked for a single feature view in Feast

Parameters

- **feature_view** – Feature view that will be updated with an additional materialization interval tracked
- **project** – Feast project that this feature view belongs to
- **start_date** (*datetime*) – Start date of the materialization interval to track
- **end_date** (*datetime*) – End date of the materialization interval to track
- **commit** – Whether the change should be persisted immediately

abstract apply_saved_dataset(*saved_dataset: feast.saved_dataset.SavedDataset*, *project: str*, *commit: bool = True*)

Stores a saved dataset metadata with Feast

Parameters

- **saved_dataset** – SavedDataset that will be added / updated to registry
- **project** – Feast project that this dataset belongs to
- **commit** – Whether the change should be persisted immediately

abstract apply_validation_reference(*validation_reference: feast.saved_dataset.ValidationReference*, *project: str*, *commit: bool = True*)

Persist a validation reference

Parameters

- **validation_reference** – ValidationReference that will be added / updated to registry
- **project** – Feast project that this dataset belongs to
- **commit** – Whether the change should be persisted immediately

abstract commit()

Commits the state of the registry cache to the remote registry store.

abstract delete_data_source(*name: str*, *project: str*, *commit: bool = True*)

Deletes a data source or raises an exception if not found.

Parameters

- **name** – Name of data source
- **project** – Feast project that this data source belongs to
- **commit** – Whether the change should be persisted immediately

abstract delete_entity(*name: str*, *project: str*, *commit: bool = True*)

Deletes an entity or raises an exception if not found.

Parameters

- **name** – Name of entity
- **project** – Feast project that this entity belongs to
- **commit** – Whether the change should be persisted immediately

abstract delete_feature_service(*name: str*, *project: str*, *commit: bool = True*)
Deletes a feature service or raises an exception if not found.

Parameters

- **name** – Name of feature service
- **project** – Feast project that this feature service belongs to
- **commit** – Whether the change should be persisted immediately

abstract delete_feature_view(*name: str*, *project: str*, *commit: bool = True*)
Deletes a feature view or raises an exception if not found.

Parameters

- **name** – Name of feature view
- **project** – Feast project that this feature view belongs to
- **commit** – Whether the change should be persisted immediately

delete_saved_dataset(*name: str*, *project: str*, *allow_cache: bool = False*)
Delete a saved dataset.

Parameters

- **name** – Name of dataset
- **project** – Feast project that this dataset belongs to
- **allow_cache** – Whether to allow returning this dataset from a cached registry

Returns Returns either the specified SavedDataset, or raises an exception if none is found

abstract delete_validation_reference(*name: str*, *project: str*, *commit: bool = True*)
Deletes a validation reference or raises an exception if not found.

Parameters

- **name** – Name of validation reference
- **project** – Feast project that this object belongs to
- **commit** – Whether the change should be persisted immediately

abstract get_data_source(*name: str*, *project: str*, *allow_cache: bool = False*) → *feast.data_source.DataSource*
Retrieves a data source.

Parameters

- **name** – Name of data source
- **project** – Feast project that this data source belongs to
- **allow_cache** – Whether to allow returning this data source from a cached registry

Returns Returns either the specified data source, or raises an exception if none is found

abstract get_entity(*name: str*, *project: str*, *allow_cache: bool = False*) → *feast.entity.Entity*
Retrieves an entity.

Parameters

- **name** – Name of entity
- **project** – Feast project that this entity belongs to

- **allow_cache** – Whether to allow returning this entity from a cached registry

Returns Returns either the specified entity, or raises an exception if none is found

abstract get_feature_service(*name: str*, *project: str*, *allow_cache: bool = False*) → *feast.feature_service.FeatureService*

Retrieves a feature service.

Parameters

- **name** – Name of feature service
- **project** – Feast project that this feature service belongs to
- **allow_cache** – Whether to allow returning this feature service from a cached registry

Returns Returns either the specified feature service, or raises an exception if none is found

abstract get_feature_view(*name: str*, *project: str*, *allow_cache: bool = False*) → *feast.feature_view.FeatureView*

Retrieves a feature view.

Parameters

- **name** – Name of feature view
- **project** – Feast project that this feature view belongs to
- **allow_cache** – Allow returning feature view from the cached registry

Returns Returns either the specified feature view, or raises an exception if none is found

abstract get_infra(*project: str*, *allow_cache: bool = False*) → feast.infra.infra_object.Infra

Retrieves the stored Infra object.

Parameters

- **project** – Feast project that the Infra object refers to
- **allow_cache** – Whether to allow returning this entity from a cached registry

Returns The stored Infra object.

abstract get_on_demand_feature_view(*name: str*, *project: str*, *allow_cache: bool = False*) → *feast.on_demand_feature_view.OnDemandFeatureView*

Retrieves an on demand feature view.

Parameters

- **name** – Name of on demand feature view
- **project** – Feast project that this on demand feature view belongs to
- **allow_cache** – Whether to allow returning this on demand feature view from a cached registry

Returns Returns either the specified on demand feature view, or raises an exception if none is found

abstract get_request_feature_view(*name: str*, *project: str*) → feast.request_feature_view.RequestFeatureView

Retrieves a request feature view.

Parameters

- **name** – Name of request feature view
- **project** – Feast project that this feature view belongs to

- **allow_cache** – Allow returning feature view from the cached registry

Returns Returns either the specified feature view, or raises an exception if none is found

abstract get_saved_dataset(*name: str*, *project: str*, *allow_cache: bool = False*) → feast.saved_dataset.SavedDataset

Retrieves a saved dataset.

Parameters

- **name** – Name of dataset
- **project** – Feast project that this dataset belongs to
- **allow_cache** – Whether to allow returning this dataset from a cached registry

Returns Returns either the specified SavedDataset, or raises an exception if none is found

abstract get_stream_feature_view(*name: str*, *project: str*, *allow_cache: bool = False*)

Retrieves a stream feature view.

Parameters

- **name** – Name of stream feature view
- **project** – Feast project that this feature view belongs to
- **allow_cache** – Allow returning feature view from the cached registry

Returns Returns either the specified feature view, or raises an exception if none is found

abstract get_validation_reference(*name: str*, *project: str*, *allow_cache: bool = False*) → feast.saved_dataset.ValidationReference

Retrieves a validation reference.

Parameters

- **name** – Name of dataset
- **project** – Feast project that this dataset belongs to
- **allow_cache** – Whether to allow returning this dataset from a cached registry

Returns Returns either the specified ValidationReference, or raises an exception if none is found

abstract list_data_sources(*project: str*, *allow_cache: bool = False*) → List[*feast.data_source.DataSource*]

Retrieve a list of data sources from the registry

Parameters

- **project** – Filter data source based on project name
- **allow_cache** – Whether to allow returning data sources from a cached registry

Returns List of data sources

abstract list_entities(*project: str*, *allow_cache: bool = False*) → List[*feast.entity.Entity*]

Retrieve a list of entities from the registry

Parameters

- **allow_cache** – Whether to allow returning entities from a cached registry
- **project** – Filter entities based on project name

Returns List of entities

abstract list_feature_services(*project: str*, *allow_cache: bool = False*) → List[*feast.feature_service.FeatureService*]

Retrieve a list of feature services from the registry

Parameters

- **allow_cache** – Whether to allow returning entities from a cached registry
- **project** – Filter entities based on project name

Returns List of feature services

abstract list_feature_views(*project: str*, *allow_cache: bool = False*) → List[*feast.feature_view.FeatureView*]

Retrieve a list of feature views from the registry

Parameters

- **allow_cache** – Allow returning feature views from the cached registry
- **project** – Filter feature views based on project name

Returns List of feature views

abstract list_on_demand_feature_views(*project: str*, *allow_cache: bool = False*) → List[*feast.on_demand_feature_view.OnDemandFeatureView*]

Retrieve a list of on demand feature views from the registry

Parameters

- **project** – Filter on demand feature views based on project name
- **allow_cache** – Whether to allow returning on demand feature views from a cached registry

Returns List of on demand feature views

list_project_metadata(*project: str*, *allow_cache: bool = False*) → List[feast.project_metadata.ProjectMetadata]

Retrieves project metadata

Parameters

- **project** – Filter metadata based on project name
- **allow_cache** – Allow returning feature views from the cached registry

Returns List of project metadata

abstract list_request_feature_views(*project: str*, *allow_cache: bool = False*) → List[feast.request_feature_view.RequestFeatureView]

Retrieve a list of request feature views from the registry

Parameters

- **allow_cache** – Allow returning feature views from the cached registry
- **project** – Filter feature views based on project name

Returns List of request feature views

abstract list_saved_datasets(*project: str*, *allow_cache: bool = False*) → List[feast.saved_dataset.SavedDataset]

Retrieves a list of all saved datasets in specified project

Parameters

- **project** – Feast project
- **allow_cache** – Whether to allow returning this dataset from a cached registry

Returns Returns the list of SavedDatasets

abstract list_stream_feature_views(*project: str*, *allow_cache: bool = False*) → List[*feast.stream_feature_view.StreamFeatureView*]

Retrieve a list of stream feature views from the registry

Parameters

- **project** – Filter stream feature views based on project name
- **allow_cache** – Whether to allow returning stream feature views from a cached registry

Returns List of stream feature views

list_validation_references(*project: str*, *allow_cache: bool = False*) → List[feast.saved_dataset.ValidationReference]

Retrieve a list of validation references from the registry

Parameters

- **allow_cache** – Allow returning feature views from the cached registry
- **project** – Filter feature views based on project name

Returns List of request feature views

abstract proto() → feast.core.Registry_pb2.Registry

Retrieves a proto version of the registry.

Returns The registry proto object.

abstract refresh(*project: Optional[str]*)

Refreshes the state of the registry cache by fetching the registry state from the remote registry store.

to_dict(*project: str*) → Dict[str, List[Any]]

Returns a dictionary representation of the registry contents for the specified project.

For each list in the dictionary, the elements are sorted by name, so this method can be used to compare two registries.

Parameters **project** – Feast project to convert to a dict

abstract update_infra(*infra: feast.infra.infra_object.Infra*, *project: str*, *commit: bool = True*)

Updates the stored Infra object.

Parameters

- **infra** – The new Infra object to be stored.
- **project** – Feast project that the Infra object refers to
- **commit** – Whether the change should be persisted immediately

8.1 Registry

class feast.infra.registry.registry.**Registry**(*registry_config: Optional[feast.repo_config.RegistryConfig]*, *repo_path: Optional[pathlib.Path]*)

apply_data_source(*data_source:* feast.data_source.DataSource, *project: str*, *commit: bool = True*)
Registers a single data source with Feast

Parameters

- **data_source** – A data source that will be registered
- **project** – Feast project that this data source belongs to
- **commit** – Whether to immediately commit to the registry

apply_entity(*entity:* feast.entity.Entity, *project: str*, *commit: bool = True*)
Registers a single entity with Feast

Parameters

- **entity** – Entity that will be registered
- **project** – Feast project that this entity belongs to
- **commit** – Whether the change should be persisted immediately

apply_feature_service(*feature_service:* feast.feature_service.FeatureService, *project: str*, *commit: bool = True*)
Registers a single feature service with Feast

Parameters

- **feature_service** – A feature service that will be registered
- **project** – Feast project that this entity belongs to

apply_feature_view(*feature_view:* feast.base_feature_view.BaseFeatureView, *project: str*, *commit: bool = True*)
Registers a single feature view with Feast

Parameters

- **feature_view** – Feature view that will be registered
- **project** – Feast project that this feature view belongs to
- **commit** – Whether the change should be persisted immediately

apply_materialization(*feature_view:* feast.feature_view.FeatureView, *project: str*, *start_date: datetime.datetime*, *end_date: datetime.datetime*, *commit: bool = True*)
Updates materialization intervals tracked for a single feature view in Feast

Parameters

- **feature_view** – Feature view that will be updated with an additional materialization interval tracked
- **project** – Feast project that this feature view belongs to
- **start_date** (*datetime*) – Start date of the materialization interval to track
- **end_date** (*datetime*) – End date of the materialization interval to track

- **commit** – Whether the change should be persisted immediately

apply_saved_dataset(*saved_dataset: feast.saved_dataset.SavedDataset*, *project: str*, *commit: bool = True*)
Stores a saved dataset metadata with Feast

Parameters

- **saved_dataset** – SavedDataset that will be added / updated to registry
- **project** – Feast project that this dataset belongs to
- **commit** – Whether the change should be persisted immediately

apply_validation_reference(*validation_reference: feast.saved_dataset.ValidationReference*, *project: str*, *commit: bool = True*)
Persist a validation reference

Parameters

- **validation_reference** – ValidationReference that will be added / updated to registry
- **project** – Feast project that this dataset belongs to
- **commit** – Whether the change should be persisted immediately

commit()
Commits the state of the registry cache to the remote registry store.

delete_data_source(*name: str*, *project: str*, *commit: bool = True*)
Deletes a data source or raises an exception if not found.

Parameters

- **name** – Name of data source
- **project** – Feast project that this data source belongs to
- **commit** – Whether the change should be persisted immediately

delete_entity(*name: str*, *project: str*, *commit: bool = True*)
Deletes an entity or raises an exception if not found.

Parameters

- **name** – Name of entity
- **project** – Feast project that this entity belongs to
- **commit** – Whether the change should be persisted immediately

delete_feature_service(*name: str*, *project: str*, *commit: bool = True*)
Deletes a feature service or raises an exception if not found.

Parameters

- **name** – Name of feature service
- **project** – Feast project that this feature service belongs to
- **commit** – Whether the change should be persisted immediately

delete_feature_view(*name: str*, *project: str*, *commit: bool = True*)
Deletes a feature view or raises an exception if not found.

Parameters

- **name** – Name of feature view
- **project** – Feast project that this feature view belongs to

- **commit** – Whether the change should be persisted immediately

delete_validation_reference(*name: str*, *project: str*, *commit: bool = True*)
Deletes a validation reference or raises an exception if not found.

Parameters

- **name** – Name of validation reference
- **project** – Feast project that this object belongs to
- **commit** – Whether the change should be persisted immediately

get_data_source(*name: str*, *project: str*, *allow_cache: bool = False*) → *feast.data_source.DataSource*
Retrieves a data source.

Parameters

- **name** – Name of data source
- **project** – Feast project that this data source belongs to
- **allow_cache** – Whether to allow returning this data source from a cached registry

Returns Returns either the specified data source, or raises an exception if none is found

get_entity(*name: str*, *project: str*, *allow_cache: bool = False*) → *feast.entity.Entity*
Retrieves an entity.

Parameters

- **name** – Name of entity
- **project** – Feast project that this entity belongs to
- **allow_cache** – Whether to allow returning this entity from a cached registry

Returns Returns either the specified entity, or raises an exception if none is found

get_feature_service(*name: str*, *project: str*, *allow_cache: bool = False*) → *feast.feature_service.FeatureService*
Retrieves a feature service.

Parameters

- **name** – Name of feature service
- **project** – Feast project that this feature service belongs to
- **allow_cache** – Whether to allow returning this feature service from a cached registry

Returns Returns either the specified feature service, or raises an exception if none is found

get_feature_view(*name: str*, *project: str*, *allow_cache: bool = False*) → *feast.feature_view.FeatureView*
Retrieves a feature view.

Parameters

- **name** – Name of feature view
- **project** – Feast project that this feature view belongs to
- **allow_cache** – Allow returning feature view from the cached registry

Returns Returns either the specified feature view, or raises an exception if none is found

get_infra(*project: str*, *allow_cache: bool = False*) → feast.infra.infra_object.Infra
Retrieves the stored Infra object.

Parameters

- **project** – Feast project that the Infra object refers to
- **allow_cache** – Whether to allow returning this entity from a cached registry

Returns The stored Infra object.

get_on_demand_feature_view(*name: str*, *project: str*, *allow_cache: bool = False*) → *feast.on_demand_feature_view.OnDemandFeatureView*

Retrieves an on demand feature view.

Parameters

- **name** – Name of on demand feature view
- **project** – Feast project that this on demand feature view belongs to
- **allow_cache** – Whether to allow returning this on demand feature view from a cached registry

Returns Returns either the specified on demand feature view, or raises an exception if none is found

get_request_feature_view(*name: str*, *project: str*)

Retrieves a request feature view.

Parameters

- **name** – Name of request feature view
- **project** – Feast project that this feature view belongs to
- **allow_cache** – Allow returning feature view from the cached registry

Returns Returns either the specified feature view, or raises an exception if none is found

get_saved_dataset(*name: str*, *project: str*, *allow_cache: bool = False*) → feast.saved_dataset.SavedDataset

Retrieves a saved dataset.

Parameters

- **name** – Name of dataset
- **project** – Feast project that this dataset belongs to
- **allow_cache** – Whether to allow returning this dataset from a cached registry

Returns Returns either the specified SavedDataset, or raises an exception if none is found

get_stream_feature_view(*name: str*, *project: str*, *allow_cache: bool = False*) → *feast.stream_feature_view.StreamFeatureView*

Retrieves a stream feature view.

Parameters

- **name** – Name of stream feature view
- **project** – Feast project that this feature view belongs to
- **allow_cache** – Allow returning feature view from the cached registry

Returns Returns either the specified feature view, or raises an exception if none is found

get_validation_reference(*name: str*, *project: str*, *allow_cache: bool = False*) → feast.saved_dataset.ValidationReference

Retrieves a validation reference.

Parameters

- **name** – Name of dataset
- **project** – Feast project that this dataset belongs to
- **allow_cache** – Whether to allow returning this dataset from a cached registry

Returns Returns either the specified ValidationReference, or raises an exception if none is found

list_data_sources(*project: str*, *allow_cache: bool = False*) → List[*feast.data_source.DataSource*]
Retrieve a list of data sources from the registry

Parameters

- **project** – Filter data source based on project name
- **allow_cache** – Whether to allow returning data sources from a cached registry

Returns List of data sources

list_entities(*project: str*, *allow_cache: bool = False*) → List[*feast.entity.Entity*]
Retrieve a list of entities from the registry

Parameters

- **allow_cache** – Whether to allow returning entities from a cached registry
- **project** – Filter entities based on project name

Returns List of entities

list_feature_services(*project: str*, *allow_cache: bool = False*) → List[*feast.feature_service.FeatureService*]
Retrieve a list of feature services from the registry

Parameters

- **allow_cache** – Whether to allow returning entities from a cached registry
- **project** – Filter entities based on project name

Returns List of feature services

list_feature_views(*project: str*, *allow_cache: bool = False*) → List[*feast.feature_view.FeatureView*]
Retrieve a list of feature views from the registry

Parameters

- **allow_cache** – Allow returning feature views from the cached registry
- **project** – Filter feature views based on project name

Returns List of feature views

list_on_demand_feature_views(*project: str*, *allow_cache: bool = False*) → List[*feast.on_demand_feature_view.OnDemandFeatureView*]
Retrieve a list of on demand feature views from the registry

Parameters

- **project** – Filter on demand feature views based on project name
- **allow_cache** – Whether to allow returning on demand feature views from a cached registry

Returns List of on demand feature views

list_project_metadata(*project: str*, *allow_cache: bool = False*) → List[feast.project_metadata.ProjectMetadata]
Retrieves project metadata

Parameters

- **project** – Filter metadata based on project name
- **allow_cache** – Allow returning feature views from the cached registry

Returns List of project metadata

list_request_feature_views(*project: str*, *allow_cache: bool = False*) → List[feast.request_feature_view.RequestFeatureView]
Retrieve a list of request feature views from the registry

Parameters

- **allow_cache** – Allow returning feature views from the cached registry
- **project** – Filter feature views based on project name

Returns List of request feature views

list_saved_datasets(*project: str*, *allow_cache: bool = False*) → List[feast.saved_dataset.SavedDataset]
Retrieves a list of all saved datasets in specified project

Parameters

- **project** – Feast project
- **allow_cache** – Whether to allow returning this dataset from a cached registry

Returns Returns the list of SavedDatasets

list_stream_feature_views(*project: str*, *allow_cache: bool = False*) → List[*feast.stream_feature_view.StreamFeatureView*]
Retrieve a list of stream feature views from the registry

Parameters

- **project** – Filter stream feature views based on project name
- **allow_cache** – Whether to allow returning stream feature views from a cached registry

Returns List of stream feature views

proto() → feast.core.Registry_pb2.Registry
Retrieves a proto version of the registry.

Returns The registry proto object.

refresh(*project: Optional[str]*)
Refreshes the state of the registry cache by fetching the registry state from the remote registry store.

teardown()
Tears down (removes) the registry.

update_infra(*infra: feast.infra.infra_object.Infra*, *project: str*, *commit: bool = True*)
Updates the stored Infra object.

Parameters

- **infra** – The new Infra object to be stored.
- **project** – Feast project that the Infra object refers to
- **commit** – Whether the change should be persisted immediately

8.2 SQL Registry

class feast.infra.registry.sql.**SqlRegistry**(*registry_config: Optional[feast.repo_config.RegistryConfig]*, *repo_path: Optional[pathlib.Path]*)

apply_data_source(*data_source: feast.data_source.DataSource*, *project: str*, *commit: bool = True*)
Registers a single data source with Feast

Parameters

- **data_source** – A data source that will be registered
- **project** – Feast project that this data source belongs to
- **commit** – Whether to immediately commit to the registry

apply_entity(*entity: feast.entity.Entity*, *project: str*, *commit: bool = True*)
Registers a single entity with Feast

Parameters

- **entity** – Entity that will be registered
- **project** – Feast project that this entity belongs to
- **commit** – Whether the change should be persisted immediately

apply_feature_service(*feature_service: feast.feature_service.FeatureService*, *project: str*, *commit: bool = True*)
Registers a single feature service with Feast

Parameters

- **feature_service** – A feature service that will be registered
- **project** – Feast project that this entity belongs to

apply_feature_view(*feature_view: feast.base_feature_view.BaseFeatureView*, *project: str*, *commit: bool = True*)
Registers a single feature view with Feast

Parameters

- **feature_view** – Feature view that will be registered
- **project** – Feast project that this feature view belongs to
- **commit** – Whether the change should be persisted immediately

apply_materialization(*feature_view: feast.feature_view.FeatureView*, *project: str*, *start_date: datetime.datetime*, *end_date: datetime.datetime*, *commit: bool = True*)
Updates materialization intervals tracked for a single feature view in Feast

Parameters

- **feature_view** – Feature view that will be updated with an additional materialization interval tracked
- **project** – Feast project that this feature view belongs to
- **start_date** (*datetime*) – Start date of the materialization interval to track
- **end_date** (*datetime*) – End date of the materialization interval to track

- **commit** – Whether the change should be persisted immediately

apply_saved_dataset(*saved_dataset: feast.saved_dataset.SavedDataset*, *project: str*, *commit: bool = True*)
Stores a saved dataset metadata with Feast

Parameters

- **saved_dataset** – SavedDataset that will be added / updated to registry
- **project** – Feast project that this dataset belongs to
- **commit** – Whether the change should be persisted immediately

apply_validation_reference(*validation_reference: feast.saved_dataset.ValidationReference*, *project: str*, *commit: bool = True*)
Persist a validation reference

Parameters

- **validation_reference** – ValidationReference that will be added / updated to registry
- **project** – Feast project that this dataset belongs to
- **commit** – Whether the change should be persisted immediately

commit()
Commits the state of the registry cache to the remote registry store.

delete_data_source(*name: str*, *project: str*, *commit: bool = True*)
Deletes a data source or raises an exception if not found.

Parameters

- **name** – Name of data source
- **project** – Feast project that this data source belongs to
- **commit** – Whether the change should be persisted immediately

delete_entity(*name: str*, *project: str*, *commit: bool = True*)
Deletes an entity or raises an exception if not found.

Parameters

- **name** – Name of entity
- **project** – Feast project that this entity belongs to
- **commit** – Whether the change should be persisted immediately

delete_feature_service(*name: str*, *project: str*, *commit: bool = True*)
Deletes a feature service or raises an exception if not found.

Parameters

- **name** – Name of feature service
- **project** – Feast project that this feature service belongs to
- **commit** – Whether the change should be persisted immediately

delete_feature_view(*name: str*, *project: str*, *commit: bool = True*)
Deletes a feature view or raises an exception if not found.

Parameters

- **name** – Name of feature view
- **project** – Feast project that this feature view belongs to

- **commit** – Whether the change should be persisted immediately

delete_validation_reference(*name: str*, *project: str*, *commit: bool = True*)
Deletes a validation reference or raises an exception if not found.

Parameters

- **name** – Name of validation reference
- **project** – Feast project that this object belongs to
- **commit** – Whether the change should be persisted immediately

get_data_source(*name: str*, *project: str*, *allow_cache: bool = False*) → *feast.data_source.DataSource*
Retrieves a data source.

Parameters

- **name** – Name of data source
- **project** – Feast project that this data source belongs to
- **allow_cache** – Whether to allow returning this data source from a cached registry

Returns Returns either the specified data source, or raises an exception if none is found

get_entity(*name: str*, *project: str*, *allow_cache: bool = False*) → *feast.entity.Entity*
Retrieves an entity.

Parameters

- **name** – Name of entity
- **project** – Feast project that this entity belongs to
- **allow_cache** – Whether to allow returning this entity from a cached registry

Returns Returns either the specified entity, or raises an exception if none is found

get_feature_service(*name: str*, *project: str*, *allow_cache: bool = False*) → *feast.feature_service.FeatureService*
Retrieves a feature service.

Parameters

- **name** – Name of feature service
- **project** – Feast project that this feature service belongs to
- **allow_cache** – Whether to allow returning this feature service from a cached registry

Returns Returns either the specified feature service, or raises an exception if none is found

get_feature_view(*name: str*, *project: str*, *allow_cache: bool = False*) → *feast.feature_view.FeatureView*
Retrieves a feature view.

Parameters

- **name** – Name of feature view
- **project** – Feast project that this feature view belongs to
- **allow_cache** – Allow returning feature view from the cached registry

Returns Returns either the specified feature view, or raises an exception if none is found

get_infra(*project: str*, *allow_cache: bool = False*) → feast.infra.infra_object.Infra
Retrieves the stored Infra object.

Parameters

- **project** – Feast project that the Infra object refers to
- **allow_cache** – Whether to allow returning this entity from a cached registry

Returns The stored Infra object.

get_on_demand_feature_view(*name: str*, *project: str*, *allow_cache: bool = False*) → *feast.on_demand_feature_view.OnDemandFeatureView*

Retrieves an on demand feature view.

Parameters

- **name** – Name of on demand feature view
- **project** – Feast project that this on demand feature view belongs to
- **allow_cache** – Whether to allow returning this on demand feature view from a cached registry

Returns Returns either the specified on demand feature view, or raises an exception if none is found

get_request_feature_view(*name: str*, *project: str*)

Retrieves a request feature view.

Parameters

- **name** – Name of request feature view
- **project** – Feast project that this feature view belongs to
- **allow_cache** – Allow returning feature view from the cached registry

Returns Returns either the specified feature view, or raises an exception if none is found

get_saved_dataset(*name: str*, *project: str*, *allow_cache: bool = False*) → feast.saved_dataset.SavedDataset

Retrieves a saved dataset.

Parameters

- **name** – Name of dataset
- **project** – Feast project that this dataset belongs to
- **allow_cache** – Whether to allow returning this dataset from a cached registry

Returns Returns either the specified SavedDataset, or raises an exception if none is found

get_stream_feature_view(*name: str*, *project: str*, *allow_cache: bool = False*)

Retrieves a stream feature view.

Parameters

- **name** – Name of stream feature view
- **project** – Feast project that this feature view belongs to
- **allow_cache** – Allow returning feature view from the cached registry

Returns Returns either the specified feature view, or raises an exception if none is found

get_validation_reference(*name: str*, *project: str*, *allow_cache: bool = False*) → feast.saved_dataset.ValidationReference

Retrieves a validation reference.

Parameters

- **name** – Name of dataset
- **project** – Feast project that this dataset belongs to
- **allow_cache** – Whether to allow returning this dataset from a cached registry

Returns Returns either the specified ValidationReference, or raises an exception if none is found

list_data_sources(*project: str*, *allow_cache: bool = False*) → List[*feast.data_source.DataSource*]
Retrieve a list of data sources from the registry

Parameters

- **project** – Filter data source based on project name
- **allow_cache** – Whether to allow returning data sources from a cached registry

Returns List of data sources

list_entities(*project: str*, *allow_cache: bool = False*) → List[*feast.entity.Entity*]
Retrieve a list of entities from the registry

Parameters

- **allow_cache** – Whether to allow returning entities from a cached registry
- **project** – Filter entities based on project name

Returns List of entities

list_feature_services(*project: str*, *allow_cache: bool = False*) → List[*feast.feature_service.FeatureService*]
Retrieve a list of feature services from the registry

Parameters

- **allow_cache** – Whether to allow returning entities from a cached registry
- **project** – Filter entities based on project name

Returns List of feature services

list_feature_views(*project: str*, *allow_cache: bool = False*) → List[*feast.feature_view.FeatureView*]
Retrieve a list of feature views from the registry

Parameters

- **allow_cache** – Allow returning feature views from the cached registry
- **project** – Filter feature views based on project name

Returns List of feature views

list_on_demand_feature_views(*project: str*, *allow_cache: bool = False*) → List[*feast.on_demand_feature_view.OnDemandFeatureView*]
Retrieve a list of on demand feature views from the registry

Parameters

- **project** – Filter on demand feature views based on project name
- **allow_cache** – Whether to allow returning on demand feature views from a cached registry

Returns List of on demand feature views

list_project_metadata(*project: str*, *allow_cache: bool = False*) → List[feast.project_metadata.ProjectMetadata]
Retrieves project metadata

Parameters

- **project** – Filter metadata based on project name
- **allow_cache** – Allow returning feature views from the cached registry

Returns List of project metadata

list_request_feature_views(*project: str*, *allow_cache: bool = False*) → List[feast.request_feature_view.RequestFeatureView]
Retrieve a list of request feature views from the registry

Parameters

- **allow_cache** – Allow returning feature views from the cached registry
- **project** – Filter feature views based on project name

Returns List of request feature views

list_saved_datasets(*project: str*, *allow_cache: bool = False*) → List[feast.saved_dataset.SavedDataset]
Retrieves a list of all saved datasets in specified project

Parameters

- **project** – Feast project
- **allow_cache** – Whether to allow returning this dataset from a cached registry

Returns Returns the list of SavedDatasets

list_stream_feature_views(*project: str*, *allow_cache: bool = False*) → List[*feast.stream_feature_view.StreamFeatureView*]
Retrieve a list of stream feature views from the registry

Parameters

- **project** – Filter stream feature views based on project name
- **allow_cache** – Whether to allow returning stream feature views from a cached registry

Returns List of stream feature views

proto() → feast.core.Registry_pb2.Registry
Retrieves a proto version of the registry.

Returns The registry proto object.

refresh(*project: Optional[str]*)
Refreshes the state of the registry cache by fetching the registry state from the remote registry store.

update_infra(*infra: feast.infra.infra_object.Infra*, *project: str*, *commit: bool = True*)
Updates the stored Infra object.

Parameters

- **infra** – The new Infra object to be stored.
- **project** – Feast project that the Infra object refers to
- **commit** – Whether the change should be persisted immediately

CHAPTER

NINE

REGISTRY STORE

class feast.infra.registry.registry_store.**RegistryStore**
A registry store is a storage backend for the Feast registry.

abstract get_registry_proto() → feast.core.Registry_pb2.Registry
Retrieves the registry proto from the registry path. If there is no file at that path, raises a FileNotFoundError.

Returns Returns either the registry proto stored at the registry path, or an empty registry proto.

abstract teardown()
Tear down the registry.

abstract update_registry_proto(*registry_proto: feast.core.Registry_pb2.Registry*)
Overwrites the current registry proto with the proto passed in. This method writes to the registry path.

Parameters **registry_proto** – the new RegistryProto

9.1 File Registry Store

class feast.infra.registry.file.**FileRegistryStore**(*registry_config:* feast.repo_config.RegistryConfig, *repo_path:* *pathlib.Path*)

get_registry_proto()
Retrieves the registry proto from the registry path. If there is no file at that path, raises a FileNotFoundError.

Returns Returns either the registry proto stored at the registry path, or an empty registry proto.

teardown()
Tear down the registry.

update_registry_proto(*registry_proto: feast.core.Registry_pb2.Registry*)
Overwrites the current registry proto with the proto passed in. This method writes to the registry path.

Parameters **registry_proto** – the new RegistryProto

9.2 GCS Registry Store

class feast.infra.registry.gcs.**GCSRegistryStore**(*registry_config:* feast.repo_config.RegistryConfig, *repo_path:* *pathlib.Path*)

get_registry_proto()
Retrieves the registry proto from the registry path. If there is no file at that path, raises a FileNotFoundError.

Returns Returns either the registry proto stored at the registry path, or an empty registry proto.

teardown()
Tear down the registry.

update_registry_proto(*registry_proto: feast.core.Registry_pb2.Registry*)
Overwrites the current registry proto with the proto passed in. This method writes to the registry path.

Parameters **registry_proto** – the new RegistryProto

9.3 S3 Registry Store

class feast.infra.registry.s3.**S3RegistryStore**(*registry_config:* feast.repo_config.RegistryConfig, *repo_path:* *pathlib.Path*)

get_registry_proto()
Retrieves the registry proto from the registry path. If there is no file at that path, raises a FileNotFoundError.

Returns Returns either the registry proto stored at the registry path, or an empty registry proto.

teardown()
Tear down the registry.

update_registry_proto(*registry_proto: feast.core.Registry_pb2.Registry*)
Overwrites the current registry proto with the proto passed in. This method writes to the registry path.

Parameters **registry_proto** – the new RegistryProto

9.4 PostgreSQL Registry Store

class feast.infra.registry.contrib.postgres.postgres_registry_store.**PostgreSQLRegistryStore**(*config: feast.infra.regi reg- istry_path: str*)

get_registry_proto() → feast.core.Registry_pb2.Registry
Retrieves the registry proto from the registry path. If there is no file at that path, raises a FileNotFoundError.

Returns Returns either the registry proto stored at the registry path, or an empty registry proto.

teardown()
Tear down the registry.

update_registry_proto(*registry_proto: feast.core.Registry_pb2.Registry*)
Overwrites the current registry proto with the proto passed in. This method writes to the registry path.

Parameters `registry_proto` – the new RegistryProto

CHAPTER

TEN

PROVIDER

class feast.infra.provider.**Provider**(*config:* feast.repo_config.RepoConfig)

A provider defines an implementation of a feature store object. It orchestrates the various components of a feature store, such as the offline store, online store, and materialization engine. It is configured through a RepoConfig object.

get_feature_server_endpoint() → Optional[str]

Returns endpoint for the feature server, if it exists.

abstract get_historical_features(*config:* feast.repo_config.RepoConfig, *feature_views: List[*feast.feature_view.FeatureView*]*, *feature_refs: List[str]*, *entity_df: Union[pandas.core.frame.DataFrame, str]*, *registry:* feast.infra.registry.base_registry.BaseRegistry, *project: str*, *full_feature_names: bool*) → *feast.infra.offline_stores.offline_store.RetrievalJob*

Retrieves the point-in-time correct historical feature values for the specified entity rows.

Parameters

- **config** – The config for the current feature store.
- **feature_views** – A list containing all feature views that are referenced in the entity rows.
- **feature_refs** – The features to be retrieved.
- **entity_df** – A collection of rows containing all entity columns on which features need to be joined, as well as the timestamp column used for point-in-time joins. Either a pandas dataframe can be provided or a SQL query.
- **registry** – The registry for the current feature store.
- **project** – Feast project to which the feature views belong.
- **full_feature_names** – If True, feature names will be prefixed with the corresponding feature view name, changing them from the format "feature" to "feature_view__feature" (e.g. "daily_transactions" changes to "customer_fv__daily_transactions").

Returns A RetrievalJob that can be executed to get the features.

ingest_df(*feature_view:* feast.feature_view.FeatureView, *df: pandas.core.frame.DataFrame*)

Persists a dataframe to the online store.

Parameters

- **feature_view** – The feature view to which the dataframe corresponds.
- **df** – The dataframe to be persisted.

ingest_df_to_offline_store(*feature_view:* feast.feature_view.FeatureView, *df: pyarrow.lib.Table*)

Persists a dataframe to the offline store.

Parameters

- **feature_view** – The feature view to which the dataframe corresponds.
- **df** – The dataframe to be persisted.

abstract materialize_single_feature_view(*config: feast.repo_config.RepoConfig*, *feature_view: feast.feature_view.FeatureView*, *start_date: datetime.datetime*, *end_date: datetime.datetime*, *registry: feast.infra.registry.base_registry.BaseRegistry*, *project: str*, *tqdm_builder: Callable[[int], tqdm.std.tqdm]*) → None

Writes latest feature values in the specified time range to the online store.

Parameters

- **config** – The config for the current feature store.
- **feature_view** – The feature view to materialize.
- **start_date** – The start of the time range.
- **end_date** – The end of the time range.
- **registry** – The registry for the current feature store.
- **project** – Feast project to which the objects belong.
- **tqdm_builder** – A function to monitor the progress of materialization.

abstract online_read(*config: feast.repo_config.RepoConfig*, *table: feast.feature_view.FeatureView*, *entity_keys: List[feast.types.EntityKey_pb2.EntityKey]*, *requested_features: Optional[List[str]] = None*) → List[Tuple[Optional[datetime.datetime], Optional[Dict[str, feast.types.Value_pb2.Value]]]]

Reads features values for the given entity keys.

Parameters

- **config** – The config for the current feature store.
- **table** – The feature view whose feature values should be read.
- **entity_keys** – The list of entity keys for which feature values should be read.
- **requested_features** – The list of features that should be read.

Returns A list of the same length as entity_keys. Each item in the list is a tuple where the first item is the event timestamp for the row, and the second item is a dict mapping feature names to values, which are returned in proto format.

abstract online_write_batch(*config: feast.repo_config.RepoConfig*, *table: feast.feature_view.FeatureView*, *data: List[Tuple[feast.types.EntityKey_pb2.EntityKey, Dict[str, feast.types.Value_pb2.Value], datetime.datetime, Optional[datetime.datetime]]]*, *progress: Optional[Callable[[int], Any]]*) → None

Writes a batch of feature rows to the online store.

If a tz-naive timestamp is passed to this method, it is assumed to be UTC.

Parameters

- **config** – The config for the current feature store.
- **table** – Feature view to which these feature rows correspond.

- **data** – A list of quadruplets containing feature data. Each quadruplet contains an entity key, a dict containing feature values, an event timestamp for the row, and the created timestamp for the row if it exists.
- **progress** – Function to be called once a batch of rows is written to the online store, used to show progress.

plan_infra(*config:* feast.repo_config.RepoConfig, *desired_registry_proto:* *feast.core.Registry_pb2.Registry*) → feast.infra.infra_object.Infra

Returns the Infra required to support the desired registry.

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **desired_registry_proto** – The desired registry, in proto form.

abstract retrieve_feature_service_logs(*feature_service:* feast.feature_service.FeatureService, *start_date:* *datetime.datetime*, *end_date:* *datetime.datetime*, *config:* feast.repo_config.RepoConfig, *registry:* feast.infra.registry.base_registry.BaseRegistry) → *feast.infra.offline_stores.offline_store.RetrievalJob*

Reads logged features for the specified time window.

Parameters

- **feature_service** – The feature service whose logs should be retrieved.
- **start_date** – The start of the window.
- **end_date** – The end of the window.
- **config** – The config for the current feature store.
- **registry** – The registry for the current feature store.

Returns A RetrievalJob that can be executed to get the feature service logs.

abstract retrieve_saved_dataset(*config:* feast.repo_config.RepoConfig, *dataset:* *feast.saved_dataset.SavedDataset*) → *feast.infra.offline_stores.offline_store.RetrievalJob*

Reads a saved dataset.

Parameters

- **config** – The config for the current feature store.
- **dataset** – A SavedDataset object containing all parameters necessary for retrieving the dataset.

Returns A RetrievalJob that can be executed to get the saved dataset.

abstract teardown_infra(*project: str*, *tables: Sequence[*feast.feature_view.FeatureView*]*, *entities:* *Sequence[*feast.entity.Entity*]*)

Tears down all cloud resources for the specified set of Feast objects.

Parameters

- **project** – Feast project to which the objects belong.
- **tables** – Feature views whose corresponding infrastructure should be deleted.
- **entities** – Entities whose corresponding infrastructure should be deleted.

abstract update_infra(*project: str*, *tables_to_delete: Sequence[feast.feature_view.FeatureView]*, *tables_to_keep: Sequence[feast.feature_view.FeatureView]*, *entities_to_delete: Sequence[feast.entity.Entity]*, *entities_to_keep: Sequence[feast.entity.Entity]*, *partial: bool*)

Reconciles cloud resources with the specified set of Feast objects.

Parameters

- **project** – Feast project to which the objects belong.
- **tables_to_delete** – Feature views whose corresponding infrastructure should be deleted.
- **tables_to_keep** – Feature views whose corresponding infrastructure should not be deleted, and may need to be updated.
- **entities_to_delete** – Entities whose corresponding infrastructure should be deleted.
- **entities_to_keep** – Entities whose corresponding infrastructure should not be deleted, and may need to be updated.
- **partial** – If true, tables_to_delete and tables_to_keep are not exhaustive lists, so infrastructure corresponding to other feature views should be not be touched.

abstract write_feature_service_logs(*feature_service: feast.feature_service.FeatureService*, *logs: Union[pyarrow.lib.Table, pathlib.Path]*, *config: feast.repo_config.RepoConfig*, *registry: feast.infra.registry.base_registry.BaseRegistry*)

Writes features and entities logged by a feature server to the offline store.

The schema of the logs table is inferred from the specified feature service. Only feature services with configured logging are accepted.

Parameters

- **feature_service** – The feature service to be logged.
- **logs** – The logs, either as an arrow table or as a path to a parquet directory.
- **config** – The config for the current feature store.
- **registry** – The registry for the current feature store.

10.1 Passthrough Provider

class feast.infra.passthrough_provider.**PassthroughProvider**(*config: feast.repo_config.RepoConfig*)

The passthrough provider delegates all operations to the underlying online and offline stores.

get_historical_features(*config: feast.repo_config.RepoConfig*, *feature_views: List[feast.feature_view.FeatureView]*, *feature_refs: List[str]*, *entity_df: Union[pandas.core.frame.DataFrame, str]*, *registry: feast.infra.registry.base_registry.BaseRegistry*, *project: str*, *full_feature_names: bool*) → *feast.infra.offline_stores.offline_store.RetrievalJob*

Retrieves the point-in-time correct historical feature values for the specified entity rows.

Parameters

- **config** – The config for the current feature store.
- **feature_views** – A list containing all feature views that are referenced in the entity rows.
- **feature_refs** – The features to be retrieved.

- **entity_df** – A collection of rows containing all entity columns on which features need to be joined, as well as the timestamp column used for point-in-time joins. Either a pandas dataframe can be provided or a SQL query.
- **registry** – The registry for the current feature store.
- **project** – Feast project to which the feature views belong.
- **full_feature_names** – If True, feature names will be prefixed with the corresponding feature view name, changing them from the format "feature" to "feature_view__feature" (e.g. "daily_transactions" changes to "customer_fv__daily_transactions").

Returns A RetrievalJob that can be executed to get the features.

ingest_df(*feature_view: feast.feature_view.FeatureView*, *df: pandas.core.frame.DataFrame*)
Persists a dataframe to the online store.

Parameters

- **feature_view** – The feature view to which the dataframe corresponds.
- **df** – The dataframe to be persisted.

ingest_df_to_offline_store(*feature_view: feast.feature_view.FeatureView*, *table: pyarrow.lib.Table*)
Persists a dataframe to the offline store.

Parameters

- **feature_view** – The feature view to which the dataframe corresponds.
- **df** – The dataframe to be persisted.

materialize_single_feature_view(*config: feast.repo_config.RepoConfig*, *feature_view: feast.feature_view.FeatureView*, *start_date: datetime.datetime*, *end_date: datetime.datetime*, *registry: feast.infra.registry.base_registry.BaseRegistry*, *project: str*, *tqdm_builder: Callable[[int], tqdm.std.tqdm]*) → None
Writes latest feature values in the specified time range to the online store.

Parameters

- **config** – The config for the current feature store.
- **feature_view** – The feature view to materialize.
- **start_date** – The start of the time range.
- **end_date** – The end of the time range.
- **registry** – The registry for the current feature store.
- **project** – Feast project to which the objects belong.
- **tqdm_builder** – A function to monitor the progress of materialization.

online_read(*config: feast.repo_config.RepoConfig*, *table: feast.feature_view.FeatureView*, *entity_keys: List[feast.types.EntityKey_pb2.EntityKey]*, *requested_features: List[str] = None*) → List
Reads features values for the given entity keys.

Parameters

- **config** – The config for the current feature store.
- **table** – The feature view whose feature values should be read.
- **entity_keys** – The list of entity keys for which feature values should be read.

- **requested_features** – The list of features that should be read.

Returns A list of the same length as entity_keys. Each item in the list is a tuple where the first item is the event timestamp for the row, and the second item is a dict mapping feature names to values, which are returned in proto format.

online_write_batch(*config: feast.repo_config.RepoConfig*, *table: feast.feature_view.FeatureView*, *data: List[Tuple[feast.types.EntityKey_pb2.EntityKey, Dict[str, feast.types.Value_pb2.Value], datetime.datetime, Optional[datetime.datetime]]]*, *progress: Optional[Callable[[int], Any]]*) → None

Writes a batch of feature rows to the online store.

If a tz-naive timestamp is passed to this method, it is assumed to be UTC.

Parameters

- **config** – The config for the current feature store.
- **table** – Feature view to which these feature rows correspond.
- **data** – A list of quadruplets containing feature data. Each quadruplet contains an entity key, a dict containing feature values, an event timestamp for the row, and the created timestamp for the row if it exists.
- **progress** – Function to be called once a batch of rows is written to the online store, used to show progress.

retrieve_feature_service_logs(*feature_service: feast.feature_service.FeatureService*, *start_date: datetime.datetime*, *end_date: datetime.datetime*, *config: feast.repo_config.RepoConfig*, *registry: feast.infra.registry.base_registry.BaseRegistry*) → *feast.infra.offline_stores.offline_store.RetrievalJob*

Reads logged features for the specified time window.

Parameters

- **feature_service** – The feature service whose logs should be retrieved.
- **start_date** – The start of the window.
- **end_date** – The end of the window.
- **config** – The config for the current feature store.
- **registry** – The registry for the current feature store.

Returns A RetrievalJob that can be executed to get the feature service logs.

retrieve_saved_dataset(*config: feast.repo_config.RepoConfig*, *dataset: feast.saved_dataset.SavedDataset*) → *feast.infra.offline_stores.offline_store.RetrievalJob*

Reads a saved dataset.

Parameters

- **config** – The config for the current feature store.
- **dataset** – A SavedDataset object containing all parameters necessary for retrieving the dataset.

Returns A RetrievalJob that can be executed to get the saved dataset.

teardown_infra(*project: str*, *tables: Sequence[feast.feature_view.FeatureView]*, *entities: Sequence[feast.entity.Entity]*) → None

Tears down all cloud resources for the specified set of Feast objects.

Parameters

- **project** – Feast project to which the objects belong.
- **tables** – Feature views whose corresponding infrastructure should be deleted.
- **entities** – Entities whose corresponding infrastructure should be deleted.

update_infra(*project: str*, *tables_to_delete: Sequence[feast.feature_view.FeatureView]*, *tables_to_keep: Sequence[feast.feature_view.FeatureView]*, *entities_to_delete: Sequence[feast.entity.Entity]*, *entities_to_keep: Sequence[feast.entity.Entity]*, *partial: bool*)

Reconciles cloud resources with the specified set of Feast objects.

Parameters

- **project** – Feast project to which the objects belong.
- **tables_to_delete** – Feature views whose corresponding infrastructure should be deleted.
- **tables_to_keep** – Feature views whose corresponding infrastructure should not be deleted, and may need to be updated.
- **entities_to_delete** – Entities whose corresponding infrastructure should be deleted.
- **entities_to_keep** – Entities whose corresponding infrastructure should not be deleted, and may need to be updated.
- **partial** – If true, tables_to_delete and tables_to_keep are not exhaustive lists, so infrastructure corresponding to other feature views should be not be touched.

write_feature_service_logs(*feature_service: feast.feature_service.FeatureService*, *logs: Union[pyarrow.lib.Table, str]*, *config: feast.repo_config.RepoConfig*, *registry: feast.infra.registry.base_registry.BaseRegistry*)

Writes features and entities logged by a feature server to the offline store.

The schema of the logs table is inferred from the specified feature service. Only feature services with configured logging are accepted.

Parameters

- **feature_service** – The feature service to be logged.
- **logs** – The logs, either as an arrow table or as a path to a parquet directory.
- **config** – The config for the current feature store.
- **registry** – The registry for the current feature store.

10.2 Local Provider

class feast.infra.local.**LocalProvider**(*config: feast.repo_config.RepoConfig*)

This class only exists for backwards compatibility.

plan_infra(*config: feast.repo_config.RepoConfig*, *desired_registry_proto: feast.core.Registry_pb2.Registry*) → feast.infra.infra_object.Infra

Returns the Infra required to support the desired registry.

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **desired_registry_proto** – The desired registry, in proto form.

10.3 GCP Provider

class feast.infra.gcp.**GcpProvider**(*config:* feast.repo_config.RepoConfig)

This class only exists for backwards compatibility.

10.4 AWS Provider

class feast.infra.aws.**AwsProvider**(*config:* feast.repo_config.RepoConfig)

get_feature_server_endpoint() → Optional[str]

Returns endpoint for the feature server, if it exists.

teardown_infra(*project: str*, *tables: Sequence[*feast.feature_view.FeatureView*]*, *entities: Sequence[*feast.entity.Entity*]*) → None

Tears down all cloud resources for the specified set of Feast objects.

Parameters

- **project** – Feast project to which the objects belong.
- **tables** – Feature views whose corresponding infrastructure should be deleted.
- **entities** – Entities whose corresponding infrastructure should be deleted.

update_infra(*project: str*, *tables_to_delete: Sequence[*feast.feature_view.FeatureView*]*, *tables_to_keep: Sequence[*feast.feature_view.FeatureView*]*, *entities_to_delete: Sequence[*feast.entity.Entity*]*, *entities_to_keep: Sequence[*feast.entity.Entity*]*, *partial: bool*)

Reconciles cloud resources with the specified set of Feast objects.

Parameters

- **project** – Feast project to which the objects belong.
- **tables_to_delete** – Feature views whose corresponding infrastructure should be deleted.
- **tables_to_keep** – Feature views whose corresponding infrastructure should not be deleted, and may need to be updated.
- **entities_to_delete** – Entities whose corresponding infrastructure should be deleted.
- **entities_to_keep** – Entities whose corresponding infrastructure should not be deleted, and may need to be updated.
- **partial** – If true, tables_to_delete and tables_to_keep are not exhaustive lists, so infrastructure corresponding to other feature views should be not be touched.

CHAPTER

ELEVEN

OFFLINE STORE

class feast.infra.offline_stores.offline_store.**OfflineStore**

An offline store defines the interface that Feast uses to interact with the storage and compute system that handles offline features.

Each offline store implementation is designed to work only with the corresponding data source. For example, the SnowflakeOfflineStore can handle SnowflakeSources but not FileSources.

abstract static get_historical_features(*config:* feast.repo_config.RepoConfig, *feature_views: List[*feast.feature_view.FeatureView*]*, *feature_refs: List[str]*, *entity_df: Union[pandas.core.frame.DataFrame, str]*, *registry:* feast.infra.registry.base_registry.BaseRegistry, *project: str*, *full_feature_names: bool = False*) → *feast.infra.offline_stores.offline_store.RetrievalJob*

Retrieves the point-in-time correct historical feature values for the specified entity rows.

Parameters

- **config** – The config for the current feature store.
- **feature_views** – A list containing all feature views that are referenced in the entity rows.
- **feature_refs** – The features to be retrieved.
- **entity_df** – A collection of rows containing all entity columns on which features need to be joined, as well as the timestamp column used for point-in-time joins. Either a pandas dataframe can be provided or a SQL query.
- **registry** – The registry for the current feature store.
- **project** – Feast project to which the feature views belong.
- **full_feature_names** – If True, feature names will be prefixed with the corresponding feature view name, changing them from the format "feature" to "feature_view__feature" (e.g. "daily_transactions" changes to "customer_fv__daily_transactions").

Returns A RetrievalJob that can be executed to get the features.

static offline_write_batch(*config:* feast.repo_config.RepoConfig, *feature_view:* feast.feature_view.FeatureView, *table: pyarrow.lib.Table*, *progress: Optional[Callable[[int], Any]]*)

Writes the specified arrow table to the data source underlying the specified feature view.

Parameters

- **config** – The config for the current feature store.
- **feature_view** – The feature view whose batch source should be written.

- **table** – The arrow table to write.
- **progress** – Function to be called once a portion of the data has been written, used to show progress.

abstract static pull_all_from_table_or_query(*config: feast.repo_config.RepoConfig*, *data_source: feast.data_source.DataSource*, *join_key_columns: List[str]*, *feature_name_columns: List[str]*, *timestamp_field: str*, *start_date: datetime.datetime*, *end_date: datetime.datetime*) → *feast.infra.offline_stores.offline_store.RetrievalJob*

Extracts all the entity rows (i.e. the combination of join key columns, feature columns, and timestamp columns) from the specified data source that lie within the specified time range.

All of the column names should refer to columns that exist in the data source. In particular, any mapping of column names must have already happened.

Parameters

- **config** – The config for the current feature store.
- **data_source** – The data source from which the entity rows will be extracted.
- **join_key_columns** – The columns of the join keys.
- **feature_name_columns** – The columns of the features.
- **timestamp_field** – The timestamp column.
- **start_date** – The start of the time range.
- **end_date** – The end of the time range.

Returns A RetrievalJob that can be executed to get the entity rows.

abstract static pull_latest_from_table_or_query(*config: feast.repo_config.RepoConfig*, *data_source: feast.data_source.DataSource*, *join_key_columns: List[str]*, *feature_name_columns: List[str]*, *timestamp_field: str*, *created_timestamp_column: Optional[str]*, *start_date: datetime.datetime*, *end_date: datetime.datetime*) → *feast.infra.offline_stores.offline_store.RetrievalJob*

Extracts the latest entity rows (i.e. the combination of join key columns, feature columns, and timestamp columns) from the specified data source that lie within the specified time range.

All of the column names should refer to columns that exist in the data source. In particular, any mapping of column names must have already happened.

Parameters

- **config** – The config for the current feature store.
- **data_source** – The data source from which the entity rows will be extracted.
- **join_key_columns** – The columns of the join keys.
- **feature_name_columns** – The columns of the features.
- **timestamp_field** – The timestamp column, used to determine which rows are the most recent.

- **created_timestamp_column** – The column indicating when the row was created, used to break ties.
- **start_date** – The start of the time range.
- **end_date** – The end of the time range.

Returns A RetrievalJob that can be executed to get the entity rows.

static write_logged_features(*config:* feast.repo_config.RepoConfig, *data: Union[pyarrow.lib.Table, pathlib.Path]*, *source: feast.feature_logging.LoggingSource*, *logging_config: feast.feature_logging.LoggingConfig*, *registry:* feast.infra.registry.base_registry.BaseRegistry)

Writes logged features to a specified destination in the offline store.

If the specified destination exists, data will be appended; otherwise, the destination will be created and data will be added. Thus this function can be called repeatedly with the same destination to flush logs in chunks.

Parameters

- **config** – The config for the current feature store.
- **data** – An arrow table or a path to parquet directory that contains the logs to write.
- **source** – The logging source that provides a schema and some additional metadata.
- **logging_config** – A LoggingConfig object that determines where the logs will be written.
- **registry** – The registry for the current feature store.

class feast.infra.offline_stores.offline_store.**RetrievalJob**

A RetrievalJob manages the execution of a query to retrieve data from the offline store.

abstract property full_feature_names: bool

Returns True if full feature names should be applied to the results of the query.

abstract property metadata:
Optional[feast.infra.offline_stores.offline_store.RetrievalMetadata]

Returns metadata about the retrieval job.

abstract property on_demand_feature_views:
List[*feast.on_demand_feature_view.OnDemandFeatureView*]

Returns a list containing all the on demand feature views to be handled.

abstract persist(*storage: feast.saved_dataset.SavedDatasetStorage*, *allow_overwrite:* bool = *False*)

Synchronously executes the underlying query and persists the result in the same offline store at the specified destination.

Parameters

- **storage** – The saved dataset storage object specifying where the result should be persisted.
- **allow_overwrite** – If True, a pre-existing location (e.g. table or file) can be overwritten. Currently not all individual offline store implementations make use of this parameter.

supports_remote_storage_export() → bool

Returns True if the RetrievalJob supports *to_remote_storage*.

to_arrow(*validation_reference: Optional[ValidationReference]* = *None*) → pyarrow.lib.Table

Synchronously executes the underlying query and returns the result as an arrow table.

On demand transformations will be executed. If a validation reference is provided, the dataframe will be validated.

Parameters **validation_reference** (*optional*) – The validation to apply against the retrieved dataframe.

to_df(*validation_reference: Optional[ValidationReference] = None*) → pandas.core.frame.DataFrame
Synchronously executes the underlying query and returns the result as a pandas dataframe.

On demand transformations will be executed. If a validation reference is provided, the dataframe will be validated.

Parameters **validation_reference** (*optional*) – The validation to apply against the retrieved dataframe.

to_remote_storage() → List[str]
Synchronously executes the underlying query and exports the results to remote storage (e.g. S3 or GCS).

Implementations of this method should export the results as multiple parquet files, each file sized appropriately depending on how much data is being returned by the retrieval job.

Returns A list of parquet file paths in remote storage.

to_sql() → str
Return RetrievalJob generated SQL statement if applicable.

11.1 File Offline Store

class feast.infra.offline_stores.file.**FileOfflineStore**

static get_historical_features(*config:* feast.repo_config.RepoConfig, *feature_views: List[*feast.feature_view.FeatureView*]*, *feature_refs: List[str]*, *entity_df: Union[pandas.core.frame.DataFrame, str]*, *registry:* feast.infra.registry.base_registry.BaseRegistry, *project: str*, *full_feature_names: bool = False*) → *feast.infra.offline_stores.offline_store.RetrievalJob*
Retrieves the point-in-time correct historical feature values for the specified entity rows.

Parameters

- **config** – The config for the current feature store.
- **feature_views** – A list containing all feature views that are referenced in the entity rows.
- **feature_refs** – The features to be retrieved.
- **entity_df** – A collection of rows containing all entity columns on which features need to be joined, as well as the timestamp column used for point-in-time joins. Either a pandas dataframe can be provided or a SQL query.
- **registry** – The registry for the current feature store.
- **project** – Feast project to which the feature views belong.
- **full_feature_names** – If True, feature names will be prefixed with the corresponding feature view name, changing them from the format "feature" to "feature_view__feature" (e.g. "daily_transactions" changes to "customer_fv__daily_transactions").

Returns A RetrievalJob that can be executed to get the features.

static offline_write_batch(*config: feast.repo_config.RepoConfig*, *feature_view: feast.feature_view.FeatureView*, *table: pyarrow.lib.Table*, *progress: Optional[Callable[[int], Any]]*)

Writes the specified arrow table to the data source underlying the specified feature view.

Parameters

- **config** – The config for the current feature store.
- **feature_view** – The feature view whose batch source should be written.
- **table** – The arrow table to write.
- **progress** – Function to be called once a portion of the data has been written, used to show progress.

static pull_all_from_table_or_query(*config: feast.repo_config.RepoConfig*, *data_source: feast.data_source.DataSource*, *join_key_columns: List[str]*, *feature_name_columns: List[str]*, *timestamp_field: str*, *start_date: datetime.datetime*, *end_date: datetime.datetime*) → *feast.infra.offline_stores.offline_store.RetrievalJob*

Extracts all the entity rows (i.e. the combination of join key columns, feature columns, and timestamp columns) from the specified data source that lie within the specified time range.

All of the column names should refer to columns that exist in the data source. In particular, any mapping of column names must have already happened.

Parameters

- **config** – The config for the current feature store.
- **data_source** – The data source from which the entity rows will be extracted.
- **join_key_columns** – The columns of the join keys.
- **feature_name_columns** – The columns of the features.
- **timestamp_field** – The timestamp column.
- **start_date** – The start of the time range.
- **end_date** – The end of the time range.

Returns A RetrievalJob that can be executed to get the entity rows.

static pull_latest_from_table_or_query(*config: feast.repo_config.RepoConfig*, *data_source: feast.data_source.DataSource*, *join_key_columns: List[str]*, *feature_name_columns: List[str]*, *timestamp_field: str*, *created_timestamp_column: Optional[str]*, *start_date: datetime.datetime*, *end_date: datetime.datetime*) → *feast.infra.offline_stores.offline_store.RetrievalJob*

Extracts the latest entity rows (i.e. the combination of join key columns, feature columns, and timestamp columns) from the specified data source that lie within the specified time range.

All of the column names should refer to columns that exist in the data source. In particular, any mapping of column names must have already happened.

Parameters

- **config** – The config for the current feature store.
- **data_source** – The data source from which the entity rows will be extracted.
- **join_key_columns** – The columns of the join keys.

- **feature_name_columns** – The columns of the features.
- **timestamp_field** – The timestamp column, used to determine which rows are the most recent.
- **created_timestamp_column** – The column indicating when the row was created, used to break ties.
- **start_date** – The start of the time range.
- **end_date** – The end of the time range.

Returns A RetrievalJob that can be executed to get the entity rows.

static write_logged_features(*config:* feast.repo_config.RepoConfig, *data: Union[pyarrow.lib.Table, pathlib.Path]*, *source: feast.feature_logging.LoggingSource*, *logging_config: feast.feature_logging.LoggingConfig*, *registry:* feast.infra.registry.base_registry.BaseRegistry)

Writes logged features to a specified destination in the offline store.

If the specified destination exists, data will be appended; otherwise, the destination will be created and data will be added. Thus this function can be called repeatedly with the same destination to flush logs in chunks.

Parameters

- **config** – The config for the current feature store.
- **data** – An arrow table or a path to parquet directory that contains the logs to write.
- **source** – The logging source that provides a schema and some additional metadata.
- **logging_config** – A LoggingConfig object that determines where the logs will be written.
- **registry** – The registry for the current feature store.

class feast.infra.offline_stores.file.**FileOfflineStoreConfig**(*, *type: Literal['file'] = 'file'*)

Offline store config for local (file-based) store

type: Literal['file']

Offline store type selector

class feast.infra.offline_stores.file.**FileRetrievalJob**(*evaluation_function: Callable*, *full_feature_names: bool*, *on_demand_feature_views: Optional[List[*feast.on_demand_feature_view.OnDemandFeatureV *= None*, *metadata: Optional[feast.infra.offline_stores.offline_store.RetrievalMetadata = None*)

property full_feature_names: bool

Returns True if full feature names should be applied to the results of the query.

property metadata:
Optional[feast.infra.offline_stores.offline_store.RetrievalMetadata]

Returns metadata about the retrieval job.

property on_demand_feature_views:
List[*feast.on_demand_feature_view.OnDemandFeatureView*]

Returns a list containing all the on demand feature views to be handled.

persist(*storage: feast.saved_dataset.SavedDatasetStorage*, *allow_overwrite: bool = False*)
Synchronously executes the underlying query and persists the result in the same offline store at the specified destination.

Parameters

- **storage** – The saved dataset storage object specifying where the result should be persisted.
- **allow_overwrite** – If True, a pre-existing location (e.g. table or file) can be overwritten. Currently not all individual offline store implementations make use of this parameter.

supports_remote_storage_export() → bool
Returns True if the RetrievalJob supports *to_remote_storage*.

11.2 Snowflake Offline Store

class feast.infra.offline_stores.snowflake.**SnowflakeOfflineStore**

static get_historical_features(*config: feast.repo_config.RepoConfig*, *feature_views: List[feast.feature_view.FeatureView]*, *feature_refs: List[str]*, *entity_df: Union[pandas.core.frame.DataFrame, str]*, *registry: feast.infra.registry.base_registry.BaseRegistry*, *project: str*, *full_feature_names: bool = False*) → *feast.infra.offline_stores.offline_store.RetrievalJob*
Retrieves the point-in-time correct historical feature values for the specified entity rows.

Parameters

- **config** – The config for the current feature store.
- **feature_views** – A list containing all feature views that are referenced in the entity rows.
- **feature_refs** – The features to be retrieved.
- **entity_df** – A collection of rows containing all entity columns on which features need to be joined, as well as the timestamp column used for point-in-time joins. Either a pandas dataframe can be provided or a SQL query.
- **registry** – The registry for the current feature store.
- **project** – Feast project to which the feature views belong.
- **full_feature_names** – If True, feature names will be prefixed with the corresponding feature view name, changing them from the format "feature" to "feature_view__feature" (e.g. "daily_transactions" changes to "customer_fv__daily_transactions").

Returns A RetrievalJob that can be executed to get the features.

static offline_write_batch(*config: feast.repo_config.RepoConfig*, *feature_view: feast.feature_view.FeatureView*, *table: pyarrow.lib.Table*, *progress: Optional[Callable[[int], Any]]*)
Writes the specified arrow table to the data source underlying the specified feature view.

Parameters

- **config** – The config for the current feature store.
- **feature_view** – The feature view whose batch source should be written.
- **table** – The arrow table to write.

- **progress** – Function to be called once a portion of the data has been written, used to show progress.

static pull_all_from_table_or_query(*config: feast.repo_config.RepoConfig*, *data_source: feast.data_source.DataSource*, *join_key_columns: List[str]*, *feature_name_columns: List[str]*, *timestamp_field: str*, *start_date: datetime.datetime*, *end_date: datetime.datetime*) → *feast.infra.offline_stores.offline_store.RetrievalJob*

Extracts all the entity rows (i.e. the combination of join key columns, feature columns, and timestamp columns) from the specified data source that lie within the specified time range.

All of the column names should refer to columns that exist in the data source. In particular, any mapping of column names must have already happened.

Parameters

- **config** – The config for the current feature store.
- **data_source** – The data source from which the entity rows will be extracted.
- **join_key_columns** – The columns of the join keys.
- **feature_name_columns** – The columns of the features.
- **timestamp_field** – The timestamp column.
- **start_date** – The start of the time range.
- **end_date** – The end of the time range.

Returns A RetrievalJob that can be executed to get the entity rows.

static pull_latest_from_table_or_query(*config: feast.repo_config.RepoConfig*, *data_source: feast.data_source.DataSource*, *join_key_columns: List[str]*, *feature_name_columns: List[str]*, *timestamp_field: str*, *created_timestamp_column: Optional[str]*, *start_date: datetime.datetime*, *end_date: datetime.datetime*) → *feast.infra.offline_stores.offline_store.RetrievalJob*

Extracts the latest entity rows (i.e. the combination of join key columns, feature columns, and timestamp columns) from the specified data source that lie within the specified time range.

All of the column names should refer to columns that exist in the data source. In particular, any mapping of column names must have already happened.

Parameters

- **config** – The config for the current feature store.
- **data_source** – The data source from which the entity rows will be extracted.
- **join_key_columns** – The columns of the join keys.
- **feature_name_columns** – The columns of the features.
- **timestamp_field** – The timestamp column, used to determine which rows are the most recent.
- **created_timestamp_column** – The column indicating when the row was created, used to break ties.
- **start_date** – The start of the time range.
- **end_date** – The end of the time range.

Returns A RetrievalJob that can be executed to get the entity rows.

static write_logged_features(*config: feast.repo_config.RepoConfig*, *data: Union[pyarrow.lib.Table, pathlib.Path]*, *source: feast.feature_logging.LoggingSource*, *logging_config: feast.feature_logging.LoggingConfig*, *registry: feast.infra.registry.base_registry.BaseRegistry*)

Writes logged features to a specified destination in the offline store.

If the specified destination exists, data will be appended; otherwise, the destination will be created and data will be added. Thus this function can be called repeatedly with the same destination to flush logs in chunks.

Parameters

- **config** – The config for the current feature store.
- **data** – An arrow table or a path to parquet directory that contains the logs to write.
- **source** – The logging source that provides a schema and some additional metadata.
- **logging_config** – A LoggingConfig object that determines where the logs will be written.
- **registry** – The registry for the current feature store.

class feast.infra.offline_stores.snowflake.**SnowflakeOfflineStoreConfig**(*, *type: Literal['snowflake.offline']* = *'snowflake.offline'*, *config_path: Optional[str]* = *'/home/docs/.snowsql/config'*, *account: Optional[str]* = *None*, *user: Optional[str]* = *None*, *password: Optional[str]* = *None*, *role: Optional[str]* = *None*, *warehouse: Optional[str]* = *None*, *authenticator: Optional[str]* = *None*, *database: pydantic.types.StrictStr*, *schema: Optional[str]* = *'PUBLIC'*, *storage_integration_name: Optional[str]* = *None*, *blob_export_location: Optional[str]* = *None*)

Offline store config for Snowflake

account: Optional[str]
Snowflake deployment identifier – drop .snowflakecomputing.com

authenticator: Optional[str]
Snowflake authenticator name

blob_export_location: Optional[str]
Location (in S3, Google storage or Azure storage) where data is offloaded

config_path: Optional[str]
Snowflake config path – absolute path required (Cant use ~)

database: pydantic.types.StrictStr
Snowflake database name

password: Optional[str]
Snowflake password

role: Optional[str]
Snowflake role name

schema_: Optional[str]
Snowflake schema name

storage_integration_name: Optional[str]
Storage integration name in snowflake

type: Literal['snowflake.offline']
Offline store type selector

user: Optional[str]
Snowflake user name

warehouse: Optional[str]
Snowflake warehouse name

class feast.infra.offline_stores.snowflake.**SnowflakeRetrievalJob**(*query: Union[str, Callable[[], AbstractContextManager[str]]], snowflake_conn: snowflake.connector.connection.SnowflakeConnec config: feast.repo_config.RepoConfig, full_feature_names: bool, on_demand_feature_views: Optional[List[feast.on_demand_feature_view.OnDem = None, metadata: Optional[feast.infra.offline_stores.offline_store.Retrie = None*)

property full_feature_names: bool
Returns True if full feature names should be applied to the results of the query.

property metadata:
Optional[feast.infra.offline_stores.offline_store.RetrievalMetadata]
Returns metadata about the retrieval job.

property on_demand_feature_views:
List[*feast.on_demand_feature_view.OnDemandFeatureView*]
Returns a list containing all the on demand feature views to be handled.

persist(*storage: feast.saved_dataset.SavedDatasetStorage*, *allow_overwrite: bool = False*)
Synchronously executes the underlying query and persists the result in the same offline store at the specified destination.

Parameters

- **storage** – The saved dataset storage object specifying where the result should be persisted.
- **allow_overwrite** – If True, a pre-existing location (e.g. table or file) can be overwritten. Currently not all individual offline store implementations make use of this parameter.

supports_remote_storage_export() → bool
Returns True if the RetrievalJob supports *to_remote_storage*.

to_remote_storage() → List[str]
Synchronously executes the underlying query and exports the results to remote storage (e.g. S3 or GCS).

Implementations of this method should export the results as multiple parquet files, each file sized appropriately depending on how much data is being returned by the retrieval job.

Returns A list of parquet file paths in remote storage.

to_snowflake(*table_name: str*, *temporary=False*) → None
Save dataset as a new Snowflake table

to_sql() → str
Returns the SQL query that will be executed in Snowflake to build the historical feature table.

11.3 BigQuery Offline Store

class feast.infra.offline_stores.bigquery.**BigQueryOfflineStore**

static get_historical_features(*config: feast.repo_config.RepoConfig*, *feature_views: List[feast.feature_view.FeatureView]*, *feature_refs: List[str]*, *entity_df: Union[pandas.core.frame.DataFrame, str]*, *registry: feast.infra.registry.base_registry.BaseRegistry*, *project: str*, *full_feature_names: bool = False*) → *feast.infra.offline_stores.offline_store.RetrievalJob*
Retrieves the point-in-time correct historical feature values for the specified entity rows.

Parameters

- **config** – The config for the current feature store.
- **feature_views** – A list containing all feature views that are referenced in the entity rows.
- **feature_refs** – The features to be retrieved.
- **entity_df** – A collection of rows containing all entity columns on which features need to be joined, as well as the timestamp column used for point-in-time joins. Either a pandas dataframe can be provided or a SQL query.
- **registry** – The registry for the current feature store.
- **project** – Feast project to which the feature views belong.
- **full_feature_names** – If True, feature names will be prefixed with the corresponding feature view name, changing them from the format "feature" to "feature_view__feature" (e.g. "daily_transactions" changes to "customer_fv__daily_transactions").

Returns A RetrievalJob that can be executed to get the features.

static offline_write_batch(*config: feast.repo_config.RepoConfig*, *feature_view: feast.feature_view.FeatureView*, *table: pyarrow.lib.Table*, *progress: Optional[Callable[[int], Any]]*)
Writes the specified arrow table to the data source underlying the specified feature view.

Parameters

- **config** – The config for the current feature store.
- **feature_view** – The feature view whose batch source should be written.
- **table** – The arrow table to write.

- **progress** – Function to be called once a portion of the data has been written, used to show progress.

static pull_all_from_table_or_query(*config: feast.repo_config.RepoConfig*, *data_source: feast.data_source.DataSource*, *join_key_columns: List[str]*, *feature_name_columns: List[str]*, *timestamp_field: str*, *start_date: datetime.datetime*, *end_date: datetime.datetime*) → *feast.infra.offline_stores.offline_store.RetrievalJob*

Extracts all the entity rows (i.e. the combination of join key columns, feature columns, and timestamp columns) from the specified data source that lie within the specified time range.

All of the column names should refer to columns that exist in the data source. In particular, any mapping of column names must have already happened.

Parameters

- **config** – The config for the current feature store.
- **data_source** – The data source from which the entity rows will be extracted.
- **join_key_columns** – The columns of the join keys.
- **feature_name_columns** – The columns of the features.
- **timestamp_field** – The timestamp column.
- **start_date** – The start of the time range.
- **end_date** – The end of the time range.

Returns A RetrievalJob that can be executed to get the entity rows.

static pull_latest_from_table_or_query(*config: feast.repo_config.RepoConfig*, *data_source: feast.data_source.DataSource*, *join_key_columns: List[str]*, *feature_name_columns: List[str]*, *timestamp_field: str*, *created_timestamp_column: Optional[str]*, *start_date: datetime.datetime*, *end_date: datetime.datetime*) → *feast.infra.offline_stores.offline_store.RetrievalJob*

Extracts the latest entity rows (i.e. the combination of join key columns, feature columns, and timestamp columns) from the specified data source that lie within the specified time range.

All of the column names should refer to columns that exist in the data source. In particular, any mapping of column names must have already happened.

Parameters

- **config** – The config for the current feature store.
- **data_source** – The data source from which the entity rows will be extracted.
- **join_key_columns** – The columns of the join keys.
- **feature_name_columns** – The columns of the features.
- **timestamp_field** – The timestamp column, used to determine which rows are the most recent.
- **created_timestamp_column** – The column indicating when the row was created, used to break ties.
- **start_date** – The start of the time range.
- **end_date** – The end of the time range.

Returns A RetrievalJob that can be executed to get the entity rows.

static write_logged_features(*config: feast.repo_config.RepoConfig*, *data: Union[pyarrow.lib.Table, pathlib.Path]*, *source: feast.feature_logging.LoggingSource*, *logging_config: feast.feature_logging.LoggingConfig*, *registry: feast.infra.registry.base_registry.BaseRegistry*)

Writes logged features to a specified destination in the offline store.

If the specified destination exists, data will be appended; otherwise, the destination will be created and data will be added. Thus this function can be called repeatedly with the same destination to flush logs in chunks.

Parameters

- **config** – The config for the current feature store.
- **data** – An arrow table or a path to parquet directory that contains the logs to write.
- **source** – The logging source that provides a schema and some additional metadata.
- **logging_config** – A LoggingConfig object that determines where the logs will be written.
- **registry** – The registry for the current feature store.

class feast.infra.offline_stores.bigquery.**BigQueryOfflineStoreConfig**(*, *type: Literal['bigquery'] = 'bigquery'*, *dataset: pydantic.types.StrictStr = 'feast'*, *project_id: Optional[pydantic.types.StrictStr] = None*, *location: Optional[pydantic.types.StrictStr] = None*, *gcs_staging_location: Optional[str] = None*)

Offline store config for GCP BigQuery

dataset: pydantic.types.StrictStr
(optional) BigQuery Dataset name for temporary tables

gcs_staging_location: Optional[str]
(optional) GCS location used for offloading BigQuery results as parquet files.

location: Optional[pydantic.types.StrictStr]
(optional) GCP location name used for the BigQuery offline store. Examples of location names include `US`, `EU`, `us-central1`, `us-west4`. If a location is not specified, the location defaults to the `US` multi-regional location. For more information on BigQuery data locations see: https://cloud.google.com/bigquery/docs/locations

project_id: Optional[pydantic.types.StrictStr]
(optional) GCP project name used for the BigQuery offline store

type: Literal['bigquery']
Offline store type selector

class feast.infra.offline_stores.bigquery.**BigQueryRetrievalJob**(*query: Union[str, Callable[[], AbstractContextManager[str]]]*, *client: google.cloud.bigquery.client.Client*, *config: feast.repo_config.RepoConfig*, *full_feature_names: bool*, *on_demand_feature_views: Optional[List[feast.on_demand_feature_view.OnDeman = None*, *metadata: Optional[feast.infra.offline_stores.offline_store.Retrieva = None*)

property full_feature_names: bool
Returns True if full feature names should be applied to the results of the query.

property metadata: Optional[feast.infra.offline_stores.offline_store.RetrievalMetadata]
Returns metadata about the retrieval job.

property on_demand_feature_views: List[*feast.on_demand_feature_view.OnDemandFeatureView*]
Returns a list containing all the on demand feature views to be handled.

persist(*storage: feast.saved_dataset.SavedDatasetStorage*, *allow_overwrite: bool = False*)
Synchronously executes the underlying query and persists the result in the same offline store at the specified destination.

Parameters

- **storage** – The saved dataset storage object specifying where the result should be persisted.
- **allow_overwrite** – If True, a pre-existing location (e.g. table or file) can be overwritten. Currently not all individual offline store implementations make use of this parameter.

supports_remote_storage_export() → bool
Returns True if the RetrievalJob supports *to_remote_storage*.

to_bigquery(*job_config: Optional[google.cloud.bigquery.job.query.QueryJobConfig] = None*, *timeout: int = 1800*, *retry_cadence: int = 10*) → str
Synchronously executes the underlying query and exports the result to a BigQuery table. The underlying BigQuery job runs for a limited amount of time (the default is 30 minutes).

Parameters

- **job_config** (*optional*) – A bigquery.QueryJobConfig to specify options like the destination table, dry run, etc.
- **timeout** (*optional*) – The time limit of the BigQuery job in seconds. Defaults to 30 minutes.
- **retry_cadence** (*optional*) – The number of seconds for setting how long the job should checked for completion.

Returns Returns the destination table name or None if job_config.dry_run is True.

to_remote_storage() → List[str]
Synchronously executes the underlying query and exports the results to remote storage (e.g. S3 or GCS).

Implementations of this method should export the results as multiple parquet files, each file sized appropriately depending on how much data is being returned by the retrieval job.

Returns A list of parquet file paths in remote storage.

to_sql() → str
Returns the underlying SQL query.

11.4 Redshift Offline Store

class feast.infra.offline_stores.redshift.**RedshiftOfflineStore**

static get_historical_features(*config:* feast.repo_config.RepoConfig, *feature_views:* *List[*feast.feature_view.FeatureView*]*, *feature_refs: List[str]*, *entity_df: Union[pandas.core.frame.DataFrame, str]*, *registry:* feast.infra.registry.base_registry.BaseRegistry, *project: str*, *full_feature_names: bool = False*) → *feast.infra.offline_stores.offline_store.RetrievalJob*
Retrieves the point-in-time correct historical feature values for the specified entity rows.

Parameters

- **config** – The config for the current feature store.
- **feature_views** – A list containing all feature views that are referenced in the entity rows.
- **feature_refs** – The features to be retrieved.
- **entity_df** – A collection of rows containing all entity columns on which features need to be joined, as well as the timestamp column used for point-in-time joins. Either a pandas dataframe can be provided or a SQL query.
- **registry** – The registry for the current feature store.
- **project** – Feast project to which the feature views belong.
- **full_feature_names** – If True, feature names will be prefixed with the corresponding feature view name, changing them from the format "feature" to "feature_view__feature" (e.g. "daily_transactions" changes to "customer_fv__daily_transactions").

Returns A RetrievalJob that can be executed to get the features.

static offline_write_batch(*config:* feast.repo_config.RepoConfig, *feature_view:* feast.feature_view.FeatureView, *table: pyarrow.lib.Table*, *progress: Optional[Callable[[int], Any]]*)
Writes the specified arrow table to the data source underlying the specified feature view.

Parameters

- **config** – The config for the current feature store.
- **feature_view** – The feature view whose batch source should be written.
- **table** – The arrow table to write.
- **progress** – Function to be called once a portion of the data has been written, used to show progress.

static pull_all_from_table_or_query(*config: feast.repo_config.RepoConfig*, *data_source: feast.data_source.DataSource*, *join_key_columns: List[str]*, *feature_name_columns: List[str]*, *timestamp_field: str*, *start_date: datetime.datetime*, *end_date: datetime.datetime*) → *feast.infra.offline_stores.offline_store.RetrievalJob*

Extracts all the entity rows (i.e. the combination of join key columns, feature columns, and timestamp columns) from the specified data source that lie within the specified time range.

All of the column names should refer to columns that exist in the data source. In particular, any mapping of column names must have already happened.

Parameters

- **config** – The config for the current feature store.
- **data_source** – The data source from which the entity rows will be extracted.
- **join_key_columns** – The columns of the join keys.
- **feature_name_columns** – The columns of the features.
- **timestamp_field** – The timestamp column.
- **start_date** – The start of the time range.
- **end_date** – The end of the time range.

Returns A RetrievalJob that can be executed to get the entity rows.

static pull_latest_from_table_or_query(*config: feast.repo_config.RepoConfig*, *data_source: feast.data_source.DataSource*, *join_key_columns: List[str]*, *feature_name_columns: List[str]*, *timestamp_field: str*, *created_timestamp_column: Optional[str]*, *start_date: datetime.datetime*, *end_date: datetime.datetime*) → *feast.infra.offline_stores.offline_store.RetrievalJob*

Extracts the latest entity rows (i.e. the combination of join key columns, feature columns, and timestamp columns) from the specified data source that lie within the specified time range.

All of the column names should refer to columns that exist in the data source. In particular, any mapping of column names must have already happened.

Parameters

- **config** – The config for the current feature store.
- **data_source** – The data source from which the entity rows will be extracted.
- **join_key_columns** – The columns of the join keys.
- **feature_name_columns** – The columns of the features.
- **timestamp_field** – The timestamp column, used to determine which rows are the most recent.
- **created_timestamp_column** – The column indicating when the row was created, used to break ties.
- **start_date** – The start of the time range.
- **end_date** – The end of the time range.

Returns A RetrievalJob that can be executed to get the entity rows.

static write_logged_features(*config: feast.repo_config.RepoConfig*, *data: Union[pyarrow.lib.Table, pathlib.Path]*, *source: feast.feature_logging.LoggingSource*, *logging_config: feast.feature_logging.LoggingConfig*, *registry: feast.infra.registry.base_registry.BaseRegistry*)

Writes logged features to a specified destination in the offline store.

If the specified destination exists, data will be appended; otherwise, the destination will be created and data will be added. Thus this function can be called repeatedly with the same destination to flush logs in chunks.

Parameters

- **config** – The config for the current feature store.
- **data** – An arrow table or a path to parquet directory that contains the logs to write.
- **source** – The logging source that provides a schema and some additional metadata.
- **logging_config** – A LoggingConfig object that determines where the logs will be written.
- **registry** – The registry for the current feature store.

class feast.infra.offline_stores.redshift.**RedshiftOfflineStoreConfig**(*, *type: Literal['redshift'] = 'redshift'*, *cluster_id: pydantic.types.StrictStr*, *region: pydantic.types.StrictStr*, *user: pydantic.types.StrictStr*, *database: pydantic.types.StrictStr*, *s3_staging_location: pydantic.types.StrictStr*, *iam_role: pydantic.types.StrictStr*)

Offline store config for AWS Redshift

cluster_id: pydantic.types.StrictStr
Redshift cluster identifier

database: pydantic.types.StrictStr
Redshift database name

iam_role: pydantic.types.StrictStr
IAM Role for Redshift, granting it access to S3

region: pydantic.types.StrictStr
Redshift cluster's AWS region

s3_staging_location: pydantic.types.StrictStr
S3 path for importing & exporting data to Redshift

type: Literal['redshift']
Offline store type selector

user: pydantic.types.StrictStr
Redshift user name

class feast.infra.offline_stores.redshift.**RedshiftRetrievalJob**(*query: Union[str, Callable[[], AbstractContextManager[str]]]*, *redshift_client*, *s3_resource*, *config: feast.repo_config.RepoConfig*, *full_feature_names: bool*, *on_demand_feature_views: Optional[List[feast.on_demand_feature_view.OnDeman* = *None*, *metadata: Optional[feast.infra.offline_stores.offline_store.Retrieva* = *None*)

property full_feature_names: bool
Returns True if full feature names should be applied to the results of the query.

property metadata: Optional[feast.infra.offline_stores.offline_store.RetrievalMetadata]
Returns metadata about the retrieval job.

property on_demand_feature_views: List[*feast.on_demand_feature_view.OnDemandFeatureView*]
Returns a list containing all the on demand feature views to be handled.

persist(*storage: feast.saved_dataset.SavedDatasetStorage*, *allow_overwrite: bool* = *False*)
Synchronously executes the underlying query and persists the result in the same offline store at the specified destination.

Parameters

- **storage** – The saved dataset storage object specifying where the result should be persisted.
- **allow_overwrite** – If True, a pre-existing location (e.g. table or file) can be overwritten. Currently not all individual offline store implementations make use of this parameter.

supports_remote_storage_export() → bool
Returns True if the RetrievalJob supports *to_remote_storage*.

to_redshift(*table_name: str*) → None
Save dataset as a new Redshift table

to_remote_storage() → List[str]
Synchronously executes the underlying query and exports the results to remote storage (e.g. S3 or GCS).

Implementations of this method should export the results as multiple parquet files, each file sized appropriately depending on how much data is being returned by the retrieval job.

Returns A list of parquet file paths in remote storage.

to_s3() → str
Export dataset to S3 in Parquet format and return path

11.5 Spark Offline Store

class feast.infra.offline_stores.contrib.spark_offline_store.spark.**SparkOfflineStore**

static **get_historical_features**(*config:* feast.repo_config.RepoConfig, *feature_views: List[*feast.feature_view.FeatureView*]*, *feature_refs: List[str]*, *entity_df: Union[pandas.core.frame.DataFrame, str]*, *registry:* feast.infra.registry.registry.Registry, *project: str*, *full_feature_names: bool = False*) → *feast.infra.offline_stores.offline_store.RetrievalJob*

Retrieves the point-in-time correct historical feature values for the specified entity rows.

Parameters

- **config** – The config for the current feature store.
- **feature_views** – A list containing all feature views that are referenced in the entity rows.
- **feature_refs** – The features to be retrieved.
- **entity_df** – A collection of rows containing all entity columns on which features need to be joined, as well as the timestamp column used for point-in-time joins. Either a pandas dataframe can be provided or a SQL query.
- **registry** – The registry for the current feature store.
- **project** – Feast project to which the feature views belong.
- **full_feature_names** – If True, feature names will be prefixed with the corresponding feature view name, changing them from the format "feature" to "feature_view__feature" (e.g. "daily_transactions" changes to "customer_fv__daily_transactions").

Returns A RetrievalJob that can be executed to get the features.

static **offline_write_batch**(*config:* feast.repo_config.RepoConfig, *feature_view:* feast.feature_view.FeatureView, *table: pyarrow.lib.Table*, *progress: Optional[Callable[[int], Any]]*)

Writes the specified arrow table to the data source underlying the specified feature view.

Parameters

- **config** – The config for the current feature store.
- **feature_view** – The feature view whose batch source should be written.
- **table** – The arrow table to write.
- **progress** – Function to be called once a portion of the data has been written, used to show progress.

static **pull_all_from_table_or_query**(*config:* feast.repo_config.RepoConfig, *data_source:* feast.data_source.DataSource, *join_key_columns: List[str]*, *feature_name_columns: List[str]*, *timestamp_field: str*, *start_date: datetime.datetime*, *end_date: datetime.datetime*) → *feast.infra.offline_stores.offline_store.RetrievalJob*

Note that join_key_columns, feature_name_columns, timestamp_field, and created_timestamp_column have all already been mapped to column names of the source table and those column names are the values passed into this function.

static pull_latest_from_table_or_query(*config: feast.repo_config.RepoConfig*, *data_source: feast.data_source.DataSource*, *join_key_columns: List[str]*, *feature_name_columns: List[str]*, *timestamp_field: str*, *created_timestamp_column: Optional[str]*, *start_date: datetime.datetime*, *end_date: datetime.datetime*) → *feast.infra.offline_stores.offline_store.RetrievalJob*

Extracts the latest entity rows (i.e. the combination of join key columns, feature columns, and timestamp columns) from the specified data source that lie within the specified time range.

All of the column names should refer to columns that exist in the data source. In particular, any mapping of column names must have already happened.

Parameters

- **config** – The config for the current feature store.
- **data_source** – The data source from which the entity rows will be extracted.
- **join_key_columns** – The columns of the join keys.
- **feature_name_columns** – The columns of the features.
- **timestamp_field** – The timestamp column, used to determine which rows are the most recent.
- **created_timestamp_column** – The column indicating when the row was created, used to break ties.
- **start_date** – The start of the time range.
- **end_date** – The end of the time range.

Returns A RetrievalJob that can be executed to get the entity rows.

class feast.infra.offline_stores.contrib.spark_offline_store.spark.**SparkOfflineStoreConfig**(*, *type: pydantic.types.StrictSt = 'spark'*, *spark_conf: Optional[Dict[str, str]] = None*, *staging_location: Optional[pydantic. = None*, *region: Optional[pydantic. = None*)

region: Optional[pydantic.types.StrictStr]
AWS Region if applicable for s3-based staging locations

spark_conf: Optional[Dict[str, str]]
Configuration overlay for the spark session

staging_location: Optional[pydantic.types.StrictStr]
Remote path for batch materialization jobs

type: pydantic.types.StrictStr
Offline store type selector

class feast.infra.offline_stores.contrib.spark_offline_store.spark.**SparkRetrievalJob**(*spark_session: pyspark.sql.session.SparkSession*, *query: str*, *full_feature_names: bool*, *config: feast.repo_config.RepoConfig*, *on_demand_feature_views: Optional[List[feast.on_demand_feature_view.OnDemandFeatureView]] = None*, *metadata: Optional[feast.infra.offline_stores.offline_store.RetrievalMetadata] = None*)

property full_feature_names: bool
Returns True if full feature names should be applied to the results of the query.

property metadata:
Optional[feast.infra.offline_stores.offline_store.RetrievalMetadata]
Return metadata information about retrieval. Should be available even before materializing the dataset itself.

property on_demand_feature_views:
List[*feast.on_demand_feature_view.OnDemandFeatureView*]
Returns a list containing all the on demand feature views to be handled.

persist(*storage: feast.saved_dataset.SavedDatasetStorage*, *allow_overwrite: bool = False*)
Run the retrieval and persist the results in the same offline store used for read. Please note the persisting is done only within the scope of the spark session.

supports_remote_storage_export() → bool
Returns True if the RetrievalJob supports *to_remote_storage*.

to_remote_storage() → List[str]
Currently only works for local and s3-based staging locations

11.6 Trino Offline Store

11.7 PostgreSQL Offline Store

class feast.infra.offline_stores.contrib.postgres_offline_store.postgres.**PostgreSQLOfflineStore**

static get_historical_features(*config:* feast.repo_config.RepoConfig, *feature_views:* *List[*feast.feature_view.FeatureView*]*, *feature_refs: List[str]*, *entity_df: Union[pandas.core.frame.DataFrame, str]*, *registry:* feast.infra.registry.registry.Registry, *project: str*, *full_feature_names:* *bool = False*) → *feast.infra.offline_stores.offline_store.RetrievalJob*

Retrieves the point-in-time correct historical feature values for the specified entity rows.

Parameters

- **config** – The config for the current feature store.
- **feature_views** – A list containing all feature views that are referenced in the entity rows.
- **feature_refs** – The features to be retrieved.
- **entity_df** – A collection of rows containing all entity columns on which features need to be joined, as well as the timestamp column used for point-in-time joins. Either a pandas dataframe can be provided or a SQL query.
- **registry** – The registry for the current feature store.
- **project** – Feast project to which the feature views belong.
- **full_feature_names** – If True, feature names will be prefixed with the corresponding feature view name, changing them from the format "feature" to "feature_view__feature" (e.g. "daily_transactions" changes to "customer_fv__daily_transactions").

Returns A RetrievalJob that can be executed to get the features.

static pull_all_from_table_or_query(*config:* feast.repo_config.RepoConfig, *data_source:* feast.data_source.DataSource, *join_key_columns: List[str]*, *feature_name_columns: List[str]*, *timestamp_field: str*, *start_date: datetime.datetime*, *end_date: datetime.datetime*) → *feast.infra.offline_stores.offline_store.RetrievalJob*

Extracts all the entity rows (i.e. the combination of join key columns, feature columns, and timestamp columns) from the specified data source that lie within the specified time range.

All of the column names should refer to columns that exist in the data source. In particular, any mapping of column names must have already happened.

Parameters

- **config** – The config for the current feature store.
- **data_source** – The data source from which the entity rows will be extracted.
- **join_key_columns** – The columns of the join keys.
- **feature_name_columns** – The columns of the features.
- **timestamp_field** – The timestamp column.
- **start_date** – The start of the time range.
- **end_date** – The end of the time range.

Returns A RetrievalJob that can be executed to get the entity rows.

static pull_latest_from_table_or_query(*config: feast.repo_config.RepoConfig*, *data_source: feast.data_source.DataSource*, *join_key_columns: List[str]*, *feature_name_columns: List[str]*, *timestamp_field: str*, *created_timestamp_column: Optional[str]*, *start_date: datetime.datetime*, *end_date: datetime.datetime*) → *feast.infra.offline_stores.offline_store.RetrievalJob*

Extracts the latest entity rows (i.e. the combination of join key columns, feature columns, and timestamp columns) from the specified data source that lie within the specified time range.

All of the column names should refer to columns that exist in the data source. In particular, any mapping of column names must have already happened.

Parameters

- **config** – The config for the current feature store.
- **data_source** – The data source from which the entity rows will be extracted.
- **join_key_columns** – The columns of the join keys.
- **feature_name_columns** – The columns of the features.
- **timestamp_field** – The timestamp column, used to determine which rows are the most recent.
- **created_timestamp_column** – The column indicating when the row was created, used to break ties.
- **start_date** – The start of the time range.
- **end_date** – The end of the time range.

Returns A RetrievalJob that can be executed to get the entity rows.

class feast.infra.offline_stores.contrib.postgres_offline_store.postgres.**PostgreSQLOfflineStoreConfig**(*,

class feast.infra.offline_stores.contrib.postgres_offline_store.postgres.**PostgreSQLRetrievalJob**(*query: Union[str, Callable[[], AbstractContextManager[str]]]*, *config: feast.repo_config.RepoConfig*, *full_feature_names: bool*, *on_demand_feature_views: Optional[List[feast.on_demand_feature_view.OnDemandFeatureView]]*, *metadata: Optional[feast.infra.offline_stores.offline_store.RetrievalMetadata] = None*)

property full_feature_names: bool
Returns True if full feature names should be applied to the results of the query.

property metadata: Optional[feast.infra.offline_stores.offline_store.RetrievalMetadata]
Returns metadata about the retrieval job.

property on_demand_feature_views: List[*feast.on_demand_feature_view.OnDemandFeatureView*]
Returns a list containing all the on demand feature views to be handled.

persist(*storage: feast.saved_dataset.SavedDatasetStorage*, *allow_overwrite: bool = False*)
Synchronously executes the underlying query and persists the result in the same offline store at the specified destination.

Parameters

- **storage** – The saved dataset storage object specifying where the result should be persisted.
- **allow_overwrite** – If True, a pre-existing location (e.g. table or file) can be overwritten. Currently not all individual offline store implementations make use of this parameter.

to_sql() → str
Return RetrievalJob generated SQL statement if applicable.

CHAPTER

TWELVE

ONLINE STORE

class feast.infra.online_stores.online_store.**OnlineStore**

The interface that Feast uses to interact with the storage system that handles online features.

abstract online_read(*config:* feast.repo_config.RepoConfig, *table:* feast.feature_view.FeatureView, *entity_keys: List[feast.types.EntityKey_pb2.EntityKey]*, *requested_features: Optional[List[str]] = None*) → List[Tuple[Optional[datetime.datetime], Optional[Dict[str, feast.types.Value_pb2.Value]]]]

Reads features values for the given entity keys.

Parameters

- **config** – The config for the current feature store.
- **table** – The feature view whose feature values should be read.
- **entity_keys** – The list of entity keys for which feature values should be read.
- **requested_features** – The list of features that should be read.

Returns A list of the same length as entity_keys. Each item in the list is a tuple where the first item is the event timestamp for the row, and the second item is a dict mapping feature names to values, which are returned in proto format.

abstract online_write_batch(*config:* feast.repo_config.RepoConfig, *table:* feast.feature_view.FeatureView, *data: List[Tuple[feast.types.EntityKey_pb2.EntityKey, Dict[str, feast.types.Value_pb2.Value], datetime.datetime, Optional[datetime.datetime]]]*, *progress: Optional[Callable[[int], Any]]*) → None

Writes a batch of feature rows to the online store.

If a tz-naive timestamp is passed to this method, it is assumed to be UTC.

Parameters

- **config** – The config for the current feature store.
- **table** – Feature view to which these feature rows correspond.
- **data** – A list of quadruplets containing feature data. Each quadruplet contains an entity key, a dict containing feature values, an event timestamp for the row, and the created timestamp for the row if it exists.
- **progress** – Function to be called once a batch of rows is written to the online store, used to show progress.

plan(*config: feast.repo_config.RepoConfig*, *desired_registry_proto: feast.core.Registry_pb2.Registry*) → List[feast.infra.infra_object.InfraObject]

Returns the set of InfraObjects required to support the desired registry.

Parameters

- **config** – The config for the current feature store.
- **desired_registry_proto** – The desired registry, in proto form.

abstract teardown(*config: feast.repo_config.RepoConfig*, *tables: Sequence[feast.feature_view.FeatureView]*, *entities: Sequence[feast.entity.Entity]*)

Tears down all cloud resources for the specified set of Feast objects.

Parameters

- **config** – The config for the current feature store.
- **tables** – Feature views whose corresponding infrastructure should be deleted.
- **entities** – Entities whose corresponding infrastructure should be deleted.

abstract update(*config: feast.repo_config.RepoConfig*, *tables_to_delete: Sequence[feast.feature_view.FeatureView]*, *tables_to_keep: Sequence[feast.feature_view.FeatureView]*, *entities_to_delete: Sequence[feast.entity.Entity]*, *entities_to_keep: Sequence[feast.entity.Entity]*, *partial: bool*)

Reconciles cloud resources with the specified set of Feast objects.

Parameters

- **config** – The config for the current feature store.
- **tables_to_delete** – Feature views whose corresponding infrastructure should be deleted.
- **tables_to_keep** – Feature views whose corresponding infrastructure should not be deleted, and may need to be updated.
- **entities_to_delete** – Entities whose corresponding infrastructure should be deleted.
- **entities_to_keep** – Entities whose corresponding infrastructure should not be deleted, and may need to be updated.
- **partial** – If true, tables_to_delete and tables_to_keep are not exhaustive lists, so infrastructure corresponding to other feature views should be not be touched.

12.1 Sqlite Online Store

class feast.infra.online_stores.sqlite.**SqliteOnlineStore**

SQLite implementation of the online store interface. Not recommended for production usage.

_conn

SQLite connection.

Type Optional[sqlite3.Connection]

online_read(*config: feast.repo_config.RepoConfig*, *table: feast.feature_view.FeatureView*, *entity_keys: List[feast.types.EntityKey_pb2.EntityKey]*, *requested_features: Optional[List[str]] = None*) → List[Tuple[Optional[datetime.datetime], Optional[Dict[str, feast.types.Value_pb2.Value]]]]

Reads features values for the given entity keys.

Parameters

- **config** – The config for the current feature store.
- **table** – The feature view whose feature values should be read.
- **entity_keys** – The list of entity keys for which feature values should be read.
- **requested_features** – The list of features that should be read.

Returns A list of the same length as entity_keys. Each item in the list is a tuple where the first item is the event timestamp for the row, and the second item is a dict mapping feature names to values, which are returned in proto format.

online_write_batch(*config:* feast.repo_config.RepoConfig, *table:* feast.feature_view.FeatureView, *data: List[Tuple[feast.types.EntityKey_pb2.EntityKey, Dict[str, feast.types.Value_pb2.Value], datetime.datetime, Optional[datetime.datetime]]]*, *progress: Optional[Callable[[int], Any]]*) → None

Writes a batch of feature rows to the online store.

If a tz-naive timestamp is passed to this method, it is assumed to be UTC.

Parameters

- **config** – The config for the current feature store.
- **table** – Feature view to which these feature rows correspond.
- **data** – A list of quadruplets containing feature data. Each quadruplet contains an entity key, a dict containing feature values, an event timestamp for the row, and the created timestamp for the row if it exists.
- **progress** – Function to be called once a batch of rows is written to the online store, used to show progress.

plan(*config:* feast.repo_config.RepoConfig, *desired_registry_proto: feast.core.Registry_pb2.Registry*) → List[feast.infra.infra_object.InfraObject]

Returns the set of InfraObjects required to support the desired registry.

Parameters

- **config** – The config for the current feature store.
- **desired_registry_proto** – The desired registry, in proto form.

teardown(*config:* feast.repo_config.RepoConfig, *tables: Sequence[*feast.feature_view.FeatureView*]*, *entities: Sequence[*feast.entity.Entity*]*)

Tears down all cloud resources for the specified set of Feast objects.

Parameters

- **config** – The config for the current feature store.
- **tables** – Feature views whose corresponding infrastructure should be deleted.
- **entities** – Entities whose corresponding infrastructure should be deleted.

update(*config:* feast.repo_config.RepoConfig, *tables_to_delete: Sequence[*feast.feature_view.FeatureView*]*, *tables_to_keep: Sequence[*feast.feature_view.FeatureView*]*, *entities_to_delete: Sequence[*feast.entity.Entity*]*, *entities_to_keep: Sequence[*feast.entity.Entity*]*, *partial: bool*)

Reconciles cloud resources with the specified set of Feast objects.

Parameters

- **config** – The config for the current feature store.

- **tables_to_delete** – Feature views whose corresponding infrastructure should be deleted.
- **tables_to_keep** – Feature views whose corresponding infrastructure should not be deleted, and may need to be updated.
- **entities_to_delete** – Entities whose corresponding infrastructure should be deleted.
- **entities_to_keep** – Entities whose corresponding infrastructure should not be deleted, and may need to be updated.
- **partial** – If true, tables_to_delete and tables_to_keep are not exhaustive lists, so infrastructure corresponding to other feature views should be not be touched.

class feast.infra.online_stores.sqlite.**SqliteOnlineStoreConfig**(*, *type: Literal['sqlite', 'feast.infra.online_stores.sqlite.SqliteOnlineStore'] = 'sqlite'*, *path: pydantic.types.StrictStr = 'data/online.db'*)

Online store config for local (SQLite-based) store

path: pydantic.types.StrictStr
(optional) Path to sqlite db

type: Literal['sqlite', 'feast.infra.online_stores.sqlite.SqliteOnlineStore']
Online store type selector

12.2 Datastore Online Store

class feast.infra.online_stores.datastore.**DatastoreOnlineStore**

Google Cloud Datastore implementation of the online store interface.

See https://github.com/feast-dev/feast/blob/master/docs/specs/online_store_format.md#google-datastore-online-store-format for more details about the data model for this implementation.

_client
Datastore connection.

Type Optional[google.cloud.datastore.client.Client]

online_read(*config:* feast.repo_config.RepoConfig, *table:* feast.feature_view.FeatureView, *entity_keys: List[feast.types.EntityKey_pb2.EntityKey]*, *requested_features: Optional[List[str]] = None*) → List[Tuple[Optional[datetime.datetime], Optional[Dict[str, feast.types.Value_pb2.Value]]]]

Reads features values for the given entity keys.

Parameters

- **config** – The config for the current feature store.
- **table** – The feature view whose feature values should be read.
- **entity_keys** – The list of entity keys for which feature values should be read.
- **requested_features** – The list of features that should be read.

Returns A list of the same length as entity_keys. Each item in the list is a tuple where the first item is the event timestamp for the row, and the second item is a dict mapping feature names to values, which are returned in proto format.

online_write_batch(*config:* feast.repo_config.RepoConfig, *table:* feast.feature_view.FeatureView, *data: List[Tuple[feast.types.EntityKey_pb2.EntityKey, Dict[str, feast.types.Value_pb2.Value], datetime.datetime, Optional[datetime.datetime]]]*, *progress: Optional[Callable[[int], Any]]*) → None

Writes a batch of feature rows to the online store.

If a tz-naive timestamp is passed to this method, it is assumed to be UTC.

Parameters

- **config** – The config for the current feature store.
- **table** – Feature view to which these feature rows correspond.
- **data** – A list of quadruplets containing feature data. Each quadruplet contains an entity key, a dict containing feature values, an event timestamp for the row, and the created timestamp for the row if it exists.
- **progress** – Function to be called once a batch of rows is written to the online store, used to show progress.

teardown(*config:* feast.repo_config.RepoConfig, *tables: Sequence[*feast.feature_view.FeatureView*]*, *entities: Sequence[*feast.entity.Entity*]*)

Tears down all cloud resources for the specified set of Feast objects.

Parameters

- **config** – The config for the current feature store.
- **tables** – Feature views whose corresponding infrastructure should be deleted.
- **entities** – Entities whose corresponding infrastructure should be deleted.

update(*config:* feast.repo_config.RepoConfig, *tables_to_delete: Sequence[*feast.feature_view.FeatureView*]*, *tables_to_keep: Sequence[*feast.feature_view.FeatureView*]*, *entities_to_delete: Sequence[*feast.entity.Entity*]*, *entities_to_keep: Sequence[*feast.entity.Entity*]*, *partial: bool*)

Reconciles cloud resources with the specified set of Feast objects.

Parameters

- **config** – The config for the current feature store.
- **tables_to_delete** – Feature views whose corresponding infrastructure should be deleted.
- **tables_to_keep** – Feature views whose corresponding infrastructure should not be deleted, and may need to be updated.
- **entities_to_delete** – Entities whose corresponding infrastructure should be deleted.
- **entities_to_keep** – Entities whose corresponding infrastructure should not be deleted, and may need to be updated.
- **partial** – If true, tables_to_delete and tables_to_keep are not exhaustive lists, so infrastructure corresponding to other feature views should be not be touched.

class feast.infra.online_stores.datastore.**DatastoreOnlineStoreConfig**(*, *type: Literal['datastore'] = 'datastore'*, *project_id: Optional[pydantic.types.StrictStr] = None*, *namespace: Optional[pydantic.types.StrictStr] = None*, *write_concurrency: Optional[pydantic.types.PositiveInt] = 40*, *write_batch_size: Optional[pydantic.types.PositiveInt] = 50*)

Online store config for GCP Datastore

namespace: Optional[pydantic.types.StrictStr]
(optional) Datastore namespace

project_id: Optional[pydantic.types.StrictStr]
(optional) GCP Project Id

type: Literal['datastore']
Online store type selector

write_batch_size: Optional[pydantic.types.PositiveInt]
(optional) Amount of feature rows per batch being written into Datastore

write_concurrency: Optional[pydantic.types.PositiveInt]
(optional) Amount of threads to use when writing batches of feature rows into Datastore

12.3 DynamoDB Online Store

class feast.infra.online_stores.dynamodb.**DynamoDBOnlineStore**

AWS DynamoDB implementation of the online store interface.

_dynamodb_client
Boto3 DynamoDB client.

_dynamodb_resource
Boto3 DynamoDB resource.

online_read(*config:* feast.repo_config.RepoConfig, *table:* feast.feature_view.FeatureView, *entity_keys: List[feast.types.EntityKey_pb2.EntityKey]*, *requested_features: Optional[List[str]] = None*) → List[Tuple[Optional[datetime.datetime], Optional[Dict[str, feast.types.Value_pb2.Value]]]]

Retrieve feature values from the online DynamoDB store.

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **table** – Feast FeatureView.
- **entity_keys** – a list of entity keys that should be read from the FeatureStore.

online_write_batch(*config:* feast.repo_config.RepoConfig, *table:* feast.feature_view.FeatureView, *data: List[Tuple[feast.types.EntityKey_pb2.EntityKey, Dict[str, feast.types.Value_pb2.Value], datetime.datetime, Optional[datetime.datetime]]]*, *progress: Optional[Callable[[int], Any]]*) → None

Write a batch of feature rows to online DynamoDB store.

Note: This method applies a `batch_writer` to automatically handle any unprocessed items and resend them as needed, this is useful if you're loading a lot of data at a time.

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **table** – Feast FeatureView.
- **data** – a list of quadruplets containing Feature data. Each quadruplet contains an Entity Key,
- **values** (*a dict containing feature*) –
- **row** (*an event timestamp for the*) –
- **and** –
- **exists.** (*the created timestamp for the row if it*) –
- **progress** – Optional function to be called once every mini-batch of rows is written to
- **progress.** (*the online store. Can be used to display*) –

teardown(*config:* feast.repo_config.RepoConfig, *tables: Sequence[*feast.feature_view.FeatureView*]*, *entities: Sequence[*feast.entity.Entity*]*)

Delete tables from the DynamoDB Online Store.

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **tables** – Tables to delete from the feature repo.

update(*config:* feast.repo_config.RepoConfig, *tables_to_delete: Sequence[*feast.feature_view.FeatureView*]*, *tables_to_keep: Sequence[*feast.feature_view.FeatureView*]*, *entities_to_delete: Sequence[*feast.entity.Entity*]*, *entities_to_keep: Sequence[*feast.entity.Entity*]*, *partial: bool*)

Update tables from the DynamoDB Online Store.

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **tables_to_delete** – Tables to delete from the DynamoDB Online Store.
- **tables_to_keep** – Tables to keep in the DynamoDB Online Store.

class feast.infra.online_stores.dynamodb.**DynamoDBOnlineStoreConfig**(*, *type: Literal['dynamodb'] = 'dynamodb'*, *batch_size: int = 40*, *endpoint_url: Optional[str] = None*, *region: pydantic.types.StrictStr*, *table_name_template: pydantic.types.StrictStr = '{project}.{table_name}'*, *consistent_reads: pydantic.types.StrictBool = False*)

Online store config for DynamoDB store

batch_size: int
Number of items to retrieve in a DynamoDB BatchGetItem call.

consistent_reads: pydantic.types.StrictBool
Whether to read from Dynamodb by forcing consistent reads

endpoint_url: Optional[str]
8000

Type DynamoDB local development endpoint Url, i.e. http

Type //localhost

region: pydantic.types.StrictStr
AWS Region Name

table_name_template: pydantic.types.StrictStr
DynamoDB table name template

type: Literal['dynamodb']
Online store type selector

12.4 Redis Online Store

class feast.infra.online_stores.redis.**RedisOnlineStore**
Redis implementation of the online store interface.

See https://github.com/feast-dev/feast/blob/master/docs/specs/online_store_format.md#redis-online-store-format for more details about the data model for this implementation.

_client
Redis connection.

Type Optional[Union[redis.client.Redis, redis.cluster.RedisCluster]]

online_read(*config:* feast.repo_config.RepoConfig, *table:* feast.feature_view.FeatureView, *entity_keys: List[feast.types.EntityKey_pb2.EntityKey]*, *requested_features: Optional[List[str]] = None*) → List[Tuple[Optional[datetime.datetime], Optional[Dict[str, feast.types.Value_pb2.Value]]]]
Reads features values for the given entity keys.

Parameters

- **config** – The config for the current feature store.
- **table** – The feature view whose feature values should be read.

- **entity_keys** – The list of entity keys for which feature values should be read.
- **requested_features** – The list of features that should be read.

Returns A list of the same length as entity_keys. Each item in the list is a tuple where the first item is the event timestamp for the row, and the second item is a dict mapping feature names to values, which are returned in proto format.

online_write_batch(*config:* feast.repo_config.RepoConfig, *table:* feast.feature_view.FeatureView, *data: List[Tuple[feast.types.EntityKey_pb2.EntityKey, Dict[str, feast.types.Value_pb2.Value], datetime.datetime, Optional[datetime.datetime]]]*, *progress: Optional[Callable[[int], Any]]*) → None

Writes a batch of feature rows to the online store.

If a tz-naive timestamp is passed to this method, it is assumed to be UTC.

Parameters

- **config** – The config for the current feature store.
- **table** – Feature view to which these feature rows correspond.
- **data** – A list of quadruplets containing feature data. Each quadruplet contains an entity key, a dict containing feature values, an event timestamp for the row, and the created timestamp for the row if it exists.
- **progress** – Function to be called once a batch of rows is written to the online store, used to show progress.

teardown(*config:* feast.repo_config.RepoConfig, *tables: Sequence[*feast.feature_view.FeatureView*]*, *entities: Sequence[*feast.entity.Entity*]*)

We delete the keys in redis for tables/views being removed.

update(*config:* feast.repo_config.RepoConfig, *tables_to_delete: Sequence[*feast.feature_view.FeatureView*]*, *tables_to_keep: Sequence[*feast.feature_view.FeatureView*]*, *entities_to_delete: Sequence[*feast.entity.Entity*]*, *entities_to_keep: Sequence[*feast.entity.Entity*]*, *partial: bool*)

Look for join_keys (list of entities) that are not in use anymore (usually this happens when the last feature view that was using specific compound key is deleted) and remove all features attached to this "join_keys".

class feast.infra.online_stores.redis.**RedisOnlineStoreConfig**(*, *type: Literal['redis'] = 'redis'*, *redis_type: feast.infra.online_stores.redis.RedisType = RedisType.redis*, *connection_string: pydantic.types.StrictStr = 'localhost:6379'*, *key_ttl_seconds: Optional[int] = None*)

Online store config for Redis store

connection_string: pydantic.types.StrictStr

Connection string containing the host, port, and configuration parameters for Redis format: host:port,parameter1,parameter2 eg. redis:6379,db=0

key_ttl_seconds: Optional[int]

(Optional) redis key bin ttl (in seconds) for expiring entities

redis_type: feast.infra.online_stores.redis.RedisType

redis or redis_cluster

Type Redis type

type: Literal['redis']
Online store type selector

12.5 Snowflake Online Store

class feast.infra.online_stores.snowflake.**SnowflakeOnlineStore**

online_read(*config:* feast.repo_config.RepoConfig, *table:* feast.feature_view.FeatureView, *entity_keys: List[feast.types.EntityKey_pb2.EntityKey]*, *requested_features: List[str]*) → List[Tuple[Optional[datetime.datetime], Optional[Dict[str, feast.types.Value_pb2.Value]]]]
Reads features values for the given entity keys.

Parameters

- **config** – The config for the current feature store.
- **table** – The feature view whose feature values should be read.
- **entity_keys** – The list of entity keys for which feature values should be read.
- **requested_features** – The list of features that should be read.

Returns A list of the same length as entity_keys. Each item in the list is a tuple where the first item is the event timestamp for the row, and the second item is a dict mapping feature names to values, which are returned in proto format.

online_write_batch(*config:* feast.repo_config.RepoConfig, *table:* feast.feature_view.FeatureView, *data: List[Tuple[feast.types.EntityKey_pb2.EntityKey, Dict[str, feast.types.Value_pb2.Value], datetime.datetime, Optional[datetime.datetime]]]*, *progress: Optional[Callable[[int], Any]]*) → None
Writes a batch of feature rows to the online store.

If a tz-naive timestamp is passed to this method, it is assumed to be UTC.

Parameters

- **config** – The config for the current feature store.
- **table** – Feature view to which these feature rows correspond.
- **data** – A list of quadruplets containing feature data. Each quadruplet contains an entity key, a dict containing feature values, an event timestamp for the row, and the created timestamp for the row if it exists.
- **progress** – Function to be called once a batch of rows is written to the online store, used to show progress.

teardown(*config:* feast.repo_config.RepoConfig, *tables: Sequence[*feast.feature_view.FeatureView*]*, *entities: Sequence[*feast.entity.Entity*]*)
Tears down all cloud resources for the specified set of Feast objects.

Parameters

- **config** – The config for the current feature store.
- **tables** – Feature views whose corresponding infrastructure should be deleted.
- **entities** – Entities whose corresponding infrastructure should be deleted.

update(*config:* feast.repo_config.RepoConfig, *tables_to_delete: Sequence[*feast.feature_view.FeatureView*]*, *tables_to_keep: Sequence[*feast.feature_view.FeatureView*]*, *entities_to_delete: Sequence[*feast.entity.Entity*]*, *entities_to_keep: Sequence[*feast.entity.Entity*]*, *partial: bool*)

Reconciles cloud resources with the specified set of Feast objects.

Parameters

- **config** – The config for the current feature store.
- **tables_to_delete** – Feature views whose corresponding infrastructure should be deleted.
- **tables_to_keep** – Feature views whose corresponding infrastructure should not be deleted, and may need to be updated.
- **entities_to_delete** – Entities whose corresponding infrastructure should be deleted.
- **entities_to_keep** – Entities whose corresponding infrastructure should not be deleted, and may need to be updated.
- **partial** – If true, tables_to_delete and tables_to_keep are not exhaustive lists, so infrastructure corresponding to other feature views should be not be touched.

class feast.infra.online_stores.snowflake.**SnowflakeOnlineStoreConfig**(*, *type: Literal['snowflake.online'] = 'snowflake.online'*, *config_path: Optional[str] = '/home/docs/.snowsql/config'*, *account: Optional[str] = None*, *user: Optional[str] = None*, *password: Optional[str] = None*, *role: Optional[str] = None*, *warehouse: Optional[str] = None*, *authenticator: Optional[str] = None*, *database: pydantic.types.StrictStr*, *schema: Optional[str] = 'PUBLIC'*)

Online store config for Snowflake

account: Optional[str]
Snowflake deployment identifier – drop .snowflakecomputing.com

authenticator: Optional[str]
Snowflake authenticator name

config_path: Optional[str]
Snowflake config path – absolute path required (Can't use ~)

database: pydantic.types.StrictStr
Snowflake database name

password: Optional[str]
Snowflake password

role: Optional[str]
Snowflake role name

schema_: Optional[str]
Snowflake schema name

type: Literal['snowflake.online']
Online store type selector

user: Optional[str]
Snowflake user name

warehouse: Optional[str]
Snowflake warehouse name

12.6 PostgreSQL Online Store

class feast.infra.online_stores.contrib.postgres.**PostgreSQLOnlineStore**

online_read(*config:* feast.repo_config.RepoConfig, *table:* feast.feature_view.FeatureView, *entity_keys: List[feast.types.EntityKey_pb2.EntityKey]*, *requested_features: Optional[List[str]] = None*) → List[Tuple[Optional[datetime.datetime], Optional[Dict[str, feast.types.Value_pb2.Value]]]]
Reads features values for the given entity keys.

Parameters

- **config** – The config for the current feature store.
- **table** – The feature view whose feature values should be read.
- **entity_keys** – The list of entity keys for which feature values should be read.
- **requested_features** – The list of features that should be read.

Returns A list of the same length as entity_keys. Each item in the list is a tuple where the first item is the event timestamp for the row, and the second item is a dict mapping feature names to values, which are returned in proto format.

online_write_batch(*config:* feast.repo_config.RepoConfig, *table:* feast.feature_view.FeatureView, *data: List[Tuple[feast.types.EntityKey_pb2.EntityKey, Dict[str, feast.types.Value_pb2.Value], datetime.datetime, Optional[datetime.datetime]]]*, *progress: Optional[Callable[[int], Any]]*) → None
Writes a batch of feature rows to the online store.

If a tz-naive timestamp is passed to this method, it is assumed to be UTC.

Parameters

- **config** – The config for the current feature store.
- **table** – Feature view to which these feature rows correspond.
- **data** – A list of quadruplets containing feature data. Each quadruplet contains an entity key, a dict containing feature values, an event timestamp for the row, and the created timestamp for the row if it exists.
- **progress** – Function to be called once a batch of rows is written to the online store, used to show progress.

teardown(*config:* feast.repo_config.RepoConfig, *tables: Sequence[feast.feature_view.FeatureView]*, *entities: Sequence[feast.entity.Entity]*)
Tears down all cloud resources for the specified set of Feast objects.

Parameters

- **config** – The config for the current feature store.
- **tables** – Feature views whose corresponding infrastructure should be deleted.
- **entities** – Entities whose corresponding infrastructure should be deleted.

update(*config: feast.repo_config.RepoConfig*, *tables_to_delete: Sequence[feast.feature_view.FeatureView]*, *tables_to_keep: Sequence[feast.feature_view.FeatureView]*, *entities_to_delete: Sequence[feast.entity.Entity]*, *entities_to_keep: Sequence[feast.entity.Entity]*, *partial: bool*)

Reconciles cloud resources with the specified set of Feast objects.

Parameters

- **config** – The config for the current feature store.
- **tables_to_delete** – Feature views whose corresponding infrastructure should be deleted.
- **tables_to_keep** – Feature views whose corresponding infrastructure should not be deleted, and may need to be updated.
- **entities_to_delete** – Entities whose corresponding infrastructure should be deleted.
- **entities_to_keep** – Entities whose corresponding infrastructure should not be deleted, and may need to be updated.
- **partial** – If true, tables_to_delete and tables_to_keep are not exhaustive lists, so infrastructure corresponding to other feature views should be not be touched.

class feast.infra.online_stores.contrib.postgres.**PostgreSQLOnlineStoreConfig**(*, *host: pydantic.types.StrictStr*, *port: int = 5432*, *database: pydantic.types.StrictStr*, *db_schema: pydantic.types.StrictStr = 'public'*, *user: pydantic.types.StrictStr*, *password: pydantic.types.StrictStr*, *sslmode: Optional[pydantic.types.StrictStr] = None*, *sslkey_path: Optional[pydantic.types.StrictStr] = None*, *sslcert_path: Optional[pydantic.types.StrictStr] = None*, *sslrootcert_path: Optional[pydantic.types.StrictStr] = None*, *type: Literal['postgres'] = 'postgres'*)

12.7 HBase Online Store

class feast.infra.online_stores.contrib.hbase_online_store.hbase.**HbaseOnlineStore**

Online feature store for Hbase.

_conn

Happybase Connection to connect to hbase thrift server.

Type happybase.connection.Connection

online_read(*config:* feast.repo_config.RepoConfig, *table:* feast.feature_view.FeatureView, *entity_keys: List[feast.types.EntityKey_pb2.EntityKey]*, *requested_features: Optional[List[str]] = None*) → List[Tuple[Optional[datetime.datetime], Optional[Dict[str, feast.types.Value_pb2.Value]]]]

Retrieve feature values from the Hbase online store.

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **table** – Feast FeatureView.
- **entity_keys** – a list of entity keys that should be read from the FeatureStore.

- **requested_features** – a list of requested feature names.

online_write_batch(*config: feast.repo_config.RepoConfig*, *table: feast.feature_view.FeatureView*, *data: List[Tuple[feast.types.EntityKey_pb2.EntityKey, Dict[str, feast.types.Value_pb2.Value], datetime.datetime, Optional[datetime.datetime]]]*, *progress: Optional[Callable[[int], Any]]*) → None

Write a batch of feature rows to Hbase online store.

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **table** – Feast FeatureView.
- **data** – a list of quadruplets containing Feature data. Each quadruplet contains an Entity Key,
- **values** (*a dict containing feature*) –
- **row** (*an event timestamp for the*) –
- **and** –
- **exists.** (*the created timestamp for the row if it*) –
- **progress** – Optional function to be called once every mini-batch of rows is written to
- **progress.** (*the online store. Can be used to display*) –

teardown(*config: feast.repo_config.RepoConfig*, *tables: Sequence[feast.feature_view.FeatureView]*, *entities: Sequence[feast.entity.Entity]*)

Delete tables from the Hbase Online Store.

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **tables** – Tables to delete from the feature repo.

update(*config: feast.repo_config.RepoConfig*, *tables_to_delete: Sequence[feast.feature_view.FeatureView]*, *tables_to_keep: Sequence[feast.feature_view.FeatureView]*, *entities_to_delete: Sequence[feast.entity.Entity]*, *entities_to_keep: Sequence[feast.entity.Entity]*, *partial: bool*)

Update tables from the Hbase Online Store.

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **tables_to_delete** – Tables to delete from the Hbase Online Store.
- **tables_to_keep** – Tables to keep in the Hbase Online Store.

class feast.infra.online_stores.contrib.hbase_online_store.hbase.**HbaseOnlineStoreConfig**(*, *type: Literal['hbase'] = 'hbase'*, *host: str*, *port: str*)

Online store config for Hbase store

host: str
Hostname of Hbase Thrift server

port: str
Port in which Hbase Thrift server is running

type: Literal['hbase']
Online store type selector

12.8 Cassandra Online Store

class feast.infra.online_stores.contrib.cassandra_online_store.cassandra_online_store.**CassandraOnlineStore**
Cassandra/Astra DB online store implementation for Feast.

_cluster
Cassandra cluster to connect to.

Type cassandra.cluster.Cluster

_session
(DataStax Cassandra drivers) session object to issue commands.

Type cassandra.cluster.Session

_keyspace
Cassandra keyspace all tables live in.

Type str

_prepared_statements
cache of statements prepared by the driver.

Type Dict[str, cassandra.query.PreparedStatement]

online_read(*config:* feast.repo_config.RepoConfig, *table:* feast.feature_view.FeatureView, *entity_keys: List[feast.types.EntityKey_pb2.EntityKey]*, *requested_features: Optional[List[str]] = None*) → List[Tuple[Optional[datetime.datetime], Optional[Dict[str, feast.types.Value_pb2.Value]]]]
Read feature values pertaining to the requested entities from the online store.

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **table** – Feast FeatureView.
- **entity_keys** – a list of entity keys that should be read from the FeatureStore.

online_write_batch(*config:* feast.repo_config.RepoConfig, *table:* feast.feature_view.FeatureView, *data: List[Tuple[feast.types.EntityKey_pb2.EntityKey, Dict[str, feast.types.Value_pb2.Value], datetime.datetime, Optional[datetime.datetime]]]*, *progress: Optional[Callable[[int], Any]]*) → None
Write a batch of features of several entities to the database.

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **table** – Feast FeatureView.

- **data** – a list of quadruplets containing Feature data. Each quadruplet contains an Entity Key, a dict containing feature values, an event timestamp for the row, and the created timestamp for the row if it exists.
- **progress** – Optional function to be called once every mini-batch of rows is written to the online store. Can be used to display progress.

teardown(*config: feast.repo_config.RepoConfig*, *tables: Sequence[feast.feature_view.FeatureView]*, *entities: Sequence[feast.entity.Entity]*)
Delete tables from the database.

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **tables** – Tables to delete from the feature repo.

update(*config: feast.repo_config.RepoConfig*, *tables_to_delete: Sequence[feast.feature_view.FeatureView]*, *tables_to_keep: Sequence[feast.feature_view.FeatureView]*, *entities_to_delete: Sequence[feast.entity.Entity]*, *entities_to_keep: Sequence[feast.entity.Entity]*, *partial: bool*)
Update schema on DB, by creating and destroying tables accordingly.

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **tables_to_delete** – Tables to delete from the Online Store.
- **tables_to_keep** – Tables to keep in the Online Store.

class feast.infra.online_stores.contrib.cassandra_online_store.cassandra_online_store.CassandraOnlineSt

Configuration for the Cassandra/Astra DB online store.

Exactly one of *hosts* and *secure_bundle_path* must be provided; depending on which one, the connection will be to a regular Cassandra or an Astra DB instance (respectively).

If connecting to Astra DB, authentication must be provided with username and password being the Client ID and Client Secret of the database token.

class CassandraLoadBalancingPolicy(*, *load_balancing_policy: pydantic.types.StrictStr*, *local_dc: pydantic.types.StrictStr = 'datacenter1'*)

Configuration block related to the Cluster's load-balancing policy.

load_balancing_policy: pydantic.types.StrictStr

A stringy description of the load balancing policy to instantiate the cluster with. Supported values: "DCAwareRoundRobinPolicy" "TokenAwarePolicy(DCAwareRoundRobinPolicy)"

local_dc: pydantic.types.StrictStr

The local datacenter, usually necessary to create the policy.

hosts: Optional[List[pydantic.types.StrictStr]]

List of host addresses to reach the cluster.

keyspace: pydantic.types.StrictStr

Target Cassandra keyspace where all tables will be.

load_balancing: Optional[*feast.infra.online_stores.contrib.cassandra_online_store.cassandra_online_store.CassandraOnlineStoreConfig.CassandraLoadBalancingPolicy*]

it will be wrapped into an execution profile if present.

Type Details on the load-balancing policy

password: Optional[pydantic.types.StrictStr]

Password for DB auth, possibly Astra DB token Client Secret.

port: Optional[pydantic.types.StrictInt]

Port number for connecting to the cluster (optional).

protocol_version: Optional[pydantic.types.StrictInt]

Explicit specification of the CQL protocol version used.

secure_bundle_path: Optional[pydantic.types.StrictStr]

Path to the secure connect bundle (for Astra DB; replaces hosts).

type: Literal['cassandra']

Online store type selector.

username: Optional[pydantic.types.StrictStr]

Username for DB auth, possibly Astra DB token Client ID.

CHAPTER

THIRTEEN

BATCH MATERIALIZATION ENGINE

class feast.infra.materialization.batch_materialization_engine.**BatchMaterializationEngine**(*, *repo_config:* feast.repo_config *of-* *fline_store:* feast.infra.offline_ *on-* *line_store:* feast.infra.online_ ***kwargs*)

The interface that Feast uses to control the compute system that handles batch materialization.

abstract materialize(*registry:* feast.infra.registry.base_registry.BaseRegistry, *tasks:* *List[*feast.infra.materialization.batch_materialization_engine.MaterializationTask*]*) → List[*feast.infra.materialization.batch_materialization_engine.MaterializationJob*]

Materialize data from the offline store to the online store for this feature repo.

Parameters

- **registry** – The registry for the current feature store.
- **tasks** – A list of individual materialization tasks.

Returns A list of materialization jobs representing each task.

abstract teardown_infra(*project: str*, *fvs: Sequence[Union[*feast.batch_feature_view.BatchFeatureView, feast.stream_feature_view.StreamFeatureView, feast.feature_view.FeatureView*]]*, *entities: Sequence[*feast.entity.Entity*]*)

Tears down all cloud resources used by the materialization engine for the specified set of Feast objects.

Parameters

- **project** – Feast project to which the objects belong.
- **fvs** – Feature views whose corresponding infrastructure should be deleted.
- **entities** – Entities whose corresponding infrastructure should be deleted.

abstract update(*project: str*, *views_to_delete: Sequence[Union[feast.batch_feature_view.BatchFeatureView, feast.stream_feature_view.StreamFeatureView, feast.feature_view.FeatureView]]*, *views_to_keep: Sequence[Union[feast.batch_feature_view.BatchFeatureView, feast.stream_feature_view.StreamFeatureView, feast.feature_view.FeatureView]]*, *entities_to_delete: Sequence[feast.entity.Entity]*, *entities_to_keep: Sequence[feast.entity.Entity]*)

Prepares cloud resources required for batch materialization for the specified set of Feast objects.

Parameters

- **project** – Feast project to which the objects belong.
- **views_to_delete** – Feature views whose corresponding infrastructure should be deleted.
- **views_to_keep** – Feature views whose corresponding infrastructure should not be deleted, and may need to be updated.
- **entities_to_delete** – Entities whose corresponding infrastructure should be deleted.
- **entities_to_keep** – Entities whose corresponding infrastructure should not be deleted, and may need to be updated.

class feast.infra.materialization.batch_materialization_engine.**MaterializationJob**

A MaterializationJob represents an ongoing or executed process that materializes data as per the definition of a materialization task.

class feast.infra.materialization.batch_materialization_engine.**MaterializationTask**(*project: str*, *feature_view: Union[feast.batch_feature_view.BatchFeatureView, feast.stream_feature_view.StreamFeatureView, feast.feature_view.FeatureView]*, *start_time: datetime.datetime*, *end_time: datetime.datetime*, *tqdm_builder: Callable[[int], tqdm.std.tqdm]*)

A MaterializationTask represents a unit of data that needs to be materialized from an offline store to an online store.

13.1 Local Engine

class feast.infra.materialization.local_engine.**LocalMaterializationEngine**(*, *repo_config: feast.repo_config.RepoConfig*, *offline_store: feast.infra.offline_stores.offline_store.O*, *online_store: feast.infra.online_stores.online_store.O*, ***kwargs*)

materialize(*registry*, *tasks: List[feast.infra.materialization.batch_materialization_engine.MaterializationTask]*) → List[*feast.infra.materialization.batch_materialization_engine.MaterializationJob*]

Materialize data from the offline store to the online store for this feature repo.

Parameters

- **registry** – The registry for the current feature store.
- **tasks** – A list of individual materialization tasks.

Returns A list of materialization jobs representing each task.

teardown_infra(*project: str*, *fvs: Sequence[Union[feast.batch_feature_view.BatchFeatureView, feast.stream_feature_view.StreamFeatureView, feast.feature_view.FeatureView]]*, *entities: Sequence[feast.entity.Entity]*)

Tears down all cloud resources used by the materialization engine for the specified set of Feast objects.

Parameters

- **project** – Feast project to which the objects belong.
- **fvs** – Feature views whose corresponding infrastructure should be deleted.
- **entities** – Entities whose corresponding infrastructure should be deleted.

update(*project: str*, *views_to_delete: Sequence[Union[feast.batch_feature_view.BatchFeatureView, feast.stream_feature_view.StreamFeatureView, feast.feature_view.FeatureView]]*, *views_to_keep: Sequence[Union[feast.batch_feature_view.BatchFeatureView, feast.stream_feature_view.StreamFeatureView, feast.feature_view.FeatureView]]*, *entities_to_delete: Sequence[feast.entity.Entity]*, *entities_to_keep: Sequence[feast.entity.Entity]*)

Prepares cloud resources required for batch materialization for the specified set of Feast objects.

Parameters

- **project** – Feast project to which the objects belong.
- **views_to_delete** – Feature views whose corresponding infrastructure should be deleted.
- **views_to_keep** – Feature views whose corresponding infrastructure should not be deleted, and may need to be updated.
- **entities_to_delete** – Entities whose corresponding infrastructure should be deleted.
- **entities_to_keep** – Entities whose corresponding infrastructure should not be deleted, and may need to be updated.

class feast.infra.materialization.local_engine.**LocalMaterializationEngineConfig**(*, *type: Literal['local'] = 'local'*)

Batch Materialization Engine config for local in-process engine

type: Literal['local']

Type selector

class feast.infra.materialization.local_engine.**LocalMaterializationJob**(*job_id: str*, *status: feast.infra.materialization.batch_materiali*
error: Union[BaseException, NoneType] = None)

13.2 Bytewax Engine

class feast.infra.materialization.contrib.bytewax.bytewax_materialization_engine.**BytewaxMaterializationE**

materialize(*registry:* feast.infra.registry.base_registry.BaseRegistry, *tasks:* *List[*feast.infra.materialization.batch_materialization_engine.MaterializationTask*]*) → List[*feast.infra.materialization.batch_materialization_engine.MaterializationJob*]

Materialize data from the offline store to the online store for this feature repo.

Parameters

- **registry** – The registry for the current feature store.
- **tasks** – A list of individual materialization tasks.

Returns A list of materialization jobs representing each task.

teardown_infra(*project: str*, *fvs: Sequence[Union[*feast.batch_feature_view.BatchFeatureView, feast.stream_feature_view.StreamFeatureView, feast.feature_view.FeatureView*]]*, *entities: Sequence[*feast.entity.Entity*]*)

This method ensures that any infrastructure or resources set up by ``update()``are torn down.

update(*project: str*, *views_to_delete: Sequence[Union[*feast.batch_feature_view.BatchFeatureView, feast.stream_feature_view.StreamFeatureView, feast.feature_view.FeatureView*]]*, *views_to_keep: Sequence[Union[*feast.batch_feature_view.BatchFeatureView, feast.stream_feature_view.StreamFeatureView, feast.feature_view.FeatureView*]]*, *entities_to_delete: Sequence[*feast.entity.Entity*]*, *entities_to_keep: Sequence[*feast.entity.Entity*]*)

This method ensures that any necessary infrastructure or resources needed by the engine are set up ahead of materialization.

class feast.infra.materialization.contrib.bytewax.bytewax_materialization_engine.BytewaxMaterializationE

Batch Materialization Engine config for Bytewax

env: List[dict]
(optional) A list of environment variables to set in the created Kubernetes pods. These environment variables can be used to reference Kubernetes secrets.

image: pydantic.types.StrictStr
(optional) The container image to use when running the materialization job.

namespace: pydantic.types.StrictStr
(optional) The namespace in Kubernetes to use when creating services, configuration maps and jobs.

type: Literal['bytewax']
Materialization type selector

class feast.infra.materialization.contrib.bytewax.bytewax_materialization_job.BytewaxMaterializationJob

13.3 Snowflake Engine

class feast.infra.materialization.snowflake_engine.**SnowflakeMaterializationEngine**(*, *repo_config:* feast.repo_config.RepoConf *offline_store:* feast.infra.offline_stores.offl *online_store:* feast.infra.online_stores.onl ***kwargs*)

materialize(*registry*, *tasks:* *List[*feast.infra.materialization.batch_materialization_engine.MaterializationTask*]*) → List[*feast.infra.materialization.batch_materialization_engine.MaterializationJob*]

Materialize data from the offline store to the online store for this feature repo.

Parameters

- **registry** – The registry for the current feature store.
- **tasks** – A list of individual materialization tasks.

Returns A list of materialization jobs representing each task.

teardown_infra(*project: str*, *fvs: Sequence[Union[*feast.batch_feature_view.BatchFeatureView*,* feast.stream_feature_view.StreamFeatureView*,* feast.feature_view.FeatureView*]]*, *entities: Sequence[*feast.entity.Entity*]*)

Tears down all cloud resources used by the materialization engine for the specified set of Feast objects.

Parameters

- **project** – Feast project to which the objects belong.
- **fvs** – Feature views whose corresponding infrastructure should be deleted.
- **entities** – Entities whose corresponding infrastructure should be deleted.

update(*project: str*, *views_to_delete: Sequence[Union[*feast.batch_feature_view.BatchFeatureView*,* feast.stream_feature_view.StreamFeatureView*,* feast.feature_view.FeatureView*]]*, *views_to_keep: Sequence[Union[*feast.batch_feature_view.BatchFeatureView*,* feast.stream_feature_view.StreamFeatureView*,* feast.feature_view.FeatureView*]]*, *entities_to_delete: Sequence[*feast.entity.Entity*]*, *entities_to_keep: Sequence[*feast.entity.Entity*]*)

Prepares cloud resources required for batch materialization for the specified set of Feast objects.

Parameters

- **project** – Feast project to which the objects belong.
- **views_to_delete** – Feature views whose corresponding infrastructure should be deleted.
- **views_to_keep** – Feature views whose corresponding infrastructure should not be deleted, and may need to be updated.
- **entities_to_delete** – Entities whose corresponding infrastructure should be deleted.
- **entities_to_keep** – Entities whose corresponding infrastructure should not be deleted, and may need to be updated.

class feast.infra.materialization.snowflake_engine.**SnowflakeMaterializationEngineConfig**(*, *type: Literal['snowflake.engi* = *'snowflake.engine'*, *config_path: Optional[str]* = *'/home/docs/.snowsq* *account: Optional[str]* = *None*, *user: Optional[str]* = *None*, *password: Optional[str]* = *None*, *role: Optional[str]* = *None*, *warehouse: Optional[str]* = *None*, *authenticator: Optional[str]* = *None*, *database: pydantic.types.StrictStr*, *schema: Optional[str]* = *'PUBLIC'*)

Batch Materialization Engine config for Snowflake Snowpark Python UDFs

account: Optional[str]
Snowflake deployment identifier – drop .snowflakecomputing.com

authenticator: Optional[str]
Snowflake authenticator name

config_path: Optional[str]
Snowflake config path – absolute path required (Cant use ~)

database: pydantic.types.StrictStr
Snowflake database name

password: Optional[str]
Snowflake password

role: Optional[str]
Snowflake role name

schema_: Optional[str]
Snowflake schema name

type: Literal['snowflake.engine']
Type selector

user: Optional[str]
Snowflake user name

warehouse: Optional[str]
Snowflake warehouse name

class feast.infra.materialization.snowflake_engine.**SnowflakeMaterializationJob**(*job_id: str*, *status: feast.infra.materialization.batch*, *error: Union[BaseException, NoneType] = None*)

13.4 (Alpha) AWS Lambda Engine

class feast.infra.materialization.aws_lambda.lambda_engine.**LambdaMaterializationEngine**(*, *repo_config: feast.repo_config.Rep*, *offline_store: feast.infra.offline_sto*, *online_store: feast.infra.online_sto*, ***kwargs*)

WARNING: This engine should be considered "Alpha" functionality.

materialize(*registry*, *tasks: List[feast.infra.materialization.batch_materialization_engine.MaterializationTask]*) → List[*feast.infra.materialization.batch_materialization_engine.MaterializationJob*]
Materialize data from the offline store to the online store for this feature repo.

Parameters

- **registry** – The registry for the current feature store.
- **tasks** – A list of individual materialization tasks.

Returns A list of materialization jobs representing each task.

teardown_infra(*project: str*, *fvs: Sequence[Union[*feast.batch_feature_view.BatchFeatureView*,* feast.stream_feature_view.StreamFeatureView*,* feast.feature_view.FeatureView*]]*, *entities: Sequence[*feast.entity.Entity*]*)

Tears down all cloud resources used by the materialization engine for the specified set of Feast objects.

Parameters

- **project** – Feast project to which the objects belong.
- **fvs** – Feature views whose corresponding infrastructure should be deleted.
- **entities** – Entities whose corresponding infrastructure should be deleted.

update(*project: str*, *views_to_delete: Sequence[Union[*feast.batch_feature_view.BatchFeatureView*,* feast.stream_feature_view.StreamFeatureView*,* feast.feature_view.FeatureView*]]*, *views_to_keep: Sequence[Union[*feast.batch_feature_view.BatchFeatureView*,* feast.stream_feature_view.StreamFeatureView*,* feast.feature_view.FeatureView*]]*, *entities_to_delete: Sequence[*feast.entity.Entity*]*, *entities_to_keep: Sequence[*feast.entity.Entity*]*)

Prepares cloud resources required for batch materialization for the specified set of Feast objects.

Parameters

- **project** – Feast project to which the objects belong.
- **views_to_delete** – Feature views whose corresponding infrastructure should be deleted.
- **views_to_keep** – Feature views whose corresponding infrastructure should not be deleted, and may need to be updated.
- **entities_to_delete** – Entities whose corresponding infrastructure should be deleted.
- **entities_to_keep** – Entities whose corresponding infrastructure should not be deleted, and may need to be updated.

class feast.infra.materialization.aws_lambda.lambda_engine.**LambdaMaterializationEngineConfig**(*, *type: Literal['lambda'] = 'lambda'*, *materialization_image: pydantic.types.Stric*, *lambda_role: pydantic.types.Stric*

Batch Materialization Engine config for lambda based engine

lambda_role: pydantic.types.StrictStr
Role that should be used by the materialization lambda

materialization_image: pydantic.types.StrictStr
The URI of a container image in the Amazon ECR registry, which should be used for materialization.

type: Literal['lambda']
Type selector

class feast.infra.materialization.aws_lambda.lambda_engine.**LambdaMaterializationJob**(*job_id: str*, *status: feast.infra.materializatio*

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W