Feast Documentation

Feast Authors

Oct 25, 2021

CONTENTS

1 Feature Store **1**

2 Config **9**

3 Data Source **11**

4 Entity **13**

5 Feature View **15**

6 On Demand Feature View **17**

7 Feature **19**

8 Feature Service **21**

9 Registry **23**

10 Provider **27**
- 10.1 Passthrough Provider . . . 28
- 10.2 Local Provider . . . 29
- 10.3 GCP Provider . . . 30
- 10.4 AWS Provider . . . 30

11 Offline Store **31**
- 11.1 File Offline Store . . . 31
- 11.2 BigQuery Offline Store . . . 32
- 11.3 Redshift Offline Store . . . 34

12 Online Store **37**
- 12.1 Sqlite Online Store . . . 38
- 12.2 Datastore Online Store . . . 39
- 12.3 DynamoDB Online Store . . . 41
- 12.4 Redis Online Store . . . 42

Python Module Index **45**

Index **47**

CHAPTER

ONE

FEATURE STORE

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

Bases: object

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

Parameters

- **repo_path** (*optional*) – Path to a *feature_store.yaml* used to configure the feature store.
- **config** (*optional*) – Configuration object used to configure the feature store.

apply(*objects: Union[feast.entity.Entity, feast.feature_view.FeatureView, feast.on_demand_feature_view.OnDemandFeatureView, feast.request_feature_view.RequestFeatureView, feast.feature_service.FeatureService, List[Union[feast.feature_view.FeatureView, feast.on_demand_feature_view.OnDemandFeatureView, feast.request_feature_view.RequestFeatureView, feast.entity.Entity, feast.feature_service.FeatureService]]]*, *commit: 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 registry has been updated, the apply method will update related infrastructure (e.g., create tables in an online store) in order to reflect these new definitions. 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.
- **commit** – whether to commit changes to the registry

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

Examples

Register an Entity and a FeatureView.

```
>>> from feast import FeatureStore, Entity, FeatureView, Feature, ValueType,␣
↪FileSource, RepoConfig
>>> from datetime import timedelta
>>> fs = FeatureStore(repo_path="feature_repo")
>>> driver = Entity(name="driver_id", value_type=ValueType.INT64, description=
↪"driver id")
>>> driver_hourly_stats = FileSource(
```

(continues on next page)

(continued from previous page)

```
...     path="feature_repo/data/driver_stats.parquet",
...     event_timestamp_column="event_timestamp",
...     created_timestamp_column="created",
... )
>>> driver_hourly_stats_view = FeatureView(
...     name="driver_hourly_stats",
...     entities=["driver_id"],
...     ttl=timedelta(seconds=86400 * 1),
...     batch_source=driver_hourly_stats,
... )
>>> fs.apply([driver_hourly_stats_view, driver]) # register entity and feature␣
↪view
```

config: **feast.repo_config.RepoConfig**

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.

ensure_request_data_values_exist(*needed_request_data: Set[str]*, *needed_request_fv_features: Set[str]*, *request_data_features: Dict[str, List[Any]]*)

get_entity(*name: str*) → feast.entity.Entity
Retrieves an entity.

Parameters **name** – Name of entity.

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*) → *feast.feature_service.FeatureService*
Retrieves a feature service.

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

Returns The specified feature service.

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

get_feature_view(*name: str*) → feast.feature_view.FeatureView
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_historical_features(*entity_df: Union[pandas.core.frame.DataFrame, str]*, *features: Optional[Union[List[str], feast.feature_service.FeatureService]] = None*, *feature_refs: Optional[List[str]] = None*, *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** – A list of features, that should be retrieved from the offline store. Either a list of string feature references can be provided or a FeatureService object. Feature references are of the format "feature_view:feature", e.g., "customer_fv:daily_transactions".
- **full_feature_names** – A boolean that provides the option to add the feature view prefixes to the feature names, changing them from the format "feature" to "feature_view__feature" (e.g., "daily_transactions" changes to "customer_fv__daily_transactions"). By default, this value is set to False.

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="feature_repo")
>>> entity_df = pd.DataFrame.from_dict(
...     {
...         "driver_id": [1001, 1002],
...         "event_timestamp": [
...             datetime(2021, 4, 12, 10, 59, 42),
...             datetime(2021, 4, 12, 8, 12, 10),
...         ],
...     }
... )
>>> retrieval_job = fs.get_historical_features(
...     entity_df=entity_df,
...     features=[
```

(continues on next page)

(continued from previous page)

```
...         "driver_hourly_stats:conv_rate",
...         "driver_hourly_stats:acc_rate",
...         "driver_hourly_stats:avg_daily_trips",
...     ],
... )
>>> feature_data = retrieval_job.to_df()
```

get_needed_request_data(*grouped_odfv_refs: List[Tuple[*feast.on_demand_feature_view.OnDemandFeatureView*, List[*str*]]], grouped_request_fv_refs: List[Tuple[feast.request_feature_view.RequestFeatureView, List[*str*]]]*) → Tuple[Set[str], Set[str]]

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]], feature_refs: Optional[List[str]] = None, 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** – List of feature references that will be returned for each entity. Each feature reference should have the following format: "feature_table:feature" where "feature_table" & "feature" refer to the feature and feature table names respectively. Only the feature name is required.
- **entity_rows** – A list of dictionaries where each key-value is an entity-name, entity-value pair.

Returns OnlineResponse containing the feature data in records.

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

Examples

Materialize all features into the online store over the interval from 3 hours ago to 10 minutes ago, and then retrieve these online features.

```
>>> from feast import FeatureStore, RepoConfig
>>> fs = FeatureStore(repo_path="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()
```

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() → 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.

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="feature_repo")
>>> fs.materialize(
...     start_date=datetime.utcnow() - timedelta(hours=3), end_date=datetime.
↪utcnow() - timedelta(minutes=10)
... )
Materializing...

...
```

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="feature_repo")
>>> fs.materialize_incremental(end_date=datetime.utcnow() -␣
↪timedelta(minutes=5))
Materializing...

...
```

property project: str

Gets the project of this feature store.

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_feature() 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.registry.Registry
Gets the registry of this feature store.

repo_path: pathlib.Path

serve(*port: int*) → 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.

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

version() → str
Returns the version of the current Feast SDK/CLI.

CHAPTER

TWO

CONFIG

class feast.repo_config.**FeastConfigBaseModel**
Feast Pydantic Configuration Class

exception feast.repo_config.**FeastConfigError**(*error_message*, *config_path*)

class feast.repo_config.**RegistryConfig**(*, *registry_store_type: pydantic.types.StrictStr = None*, *path: pydantic.types.StrictStr*, *cache_ttl_seconds: pydantic.types.StrictInt = 600*, ***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

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*, *online_store: Any = None*, *offline_store: Any = None*, *feature_server: Any = None*, *flags: Any = None*, *repo_path: pathlib.Path = None*, ***data: Any*)
Repo config. Typically loaded from *feature_store.yaml*

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

Type FeatureServerConfig

flags: Any
Feature flags for experimental features (optional)

Type Flags

offline_store: Any
Offline store configuration (optional depending on provider)

Type OfflineStoreConfig

online_store: Any
Online store configuration (optional depending on provider)

Type OnlineStoreConfig

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

CHAPTER

THREE

DATA SOURCE

class feast.data_source.**DataSource**(*event_timestamp_column: Optional[str] = None*, *created_timestamp_column: Optional[str] = None*, *field_mapping: Optional[Dict[str, str]] = None*, *date_partition_column: Optional[str] = None*)

DataSource that can be used to source features.

Parameters

- **event_timestamp_column** (*optional*) – Event timestamp column 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.
- **date_partition_column** (*optional*) – Timestamp column used for partitioning.

property created_timestamp_column: str

Returns the created timestamp column of this data source.

property date_partition_column: str

Returns the date partition column of this data source.

property event_timestamp_column: str

Returns the event timestamp column of this data source.

property field_mapping: Dict[str, str]

Returns the field mapping of this data source.

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

Converts data source config in FeatureTable 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 an 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.

class feast.data_source.**RequestDataSource**(*name: str*, *schema: Dict[str, feast.value_type.ValueType]*)
RequestDataSource that can be used to provide input features for on demand transforms

Parameters

- **name** – Name of the request data source
- **schema** – Schema mapping from the input feature name to a ValueType

static from_proto(*data_source: feast.core.DataSource_pb2.DataSource*)
Converts data source config in FeatureTable 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.

property name: str
Returns the name of this data source

property schema: Dict[str, feast.value_type.ValueType]
Returns the schema for this request 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 an 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.

class feast.data_source.**SourceType**(*value*)
DataSource value type. Used to define source types in DataSource.

CHAPTER

FOUR

ENTITY

class feast.entity.**Entity**(*name: str*, *value_type: feast.value_type.ValueType = ValueType.UNKNOWN*, *description: str = ''*, *join_key: Optional[str] = None*, *labels: Optional[Dict[str, str]] = None*)

Represents a collection of entities and associated metadata.

Parameters

- **name** – Name of the entity.
- **value_type** (*optional*) – The type of the entity, such as string or float.
- **description** (*optional*) – Additional information to describe the entity.
- **join_key** (*optional*) – A property that uniquely identifies different entities within the collection. Used as a key for joining entities with their associated features. If not specified, defaults to the name of the entity.
- **labels** (*optional*) – User-defined metadata in dictionary form.

property created_timestamp: Optional[datetime.datetime]

Gets the created_timestamp of this entity.

property description: str

Gets the description of this entity.

classmethod from_dict(*entity_dict*)

Creates an entity from a dict.

Parameters **entity_dict** – A dict representation of an entity.

Returns An EntityV2 object based on the entity dict.

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 EntityV2 object based on the entity protobuf.

classmethod from_yaml(*yml: str*)

Creates an entity from a YAML string body or a file path.

Parameters **yml** – Either a file path containing a yaml file or a YAML string.

Returns An EntityV2 object based on the YAML file.

is_valid()

Validates the state of this entity locally.

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

property join_key: str
Gets the join key of this entity.

property labels: Dict[str, str]
Gets the labels of this entity.

property last_updated_timestamp: Optional[datetime.datetime]
Gets the last_updated_timestamp of this entity.

property name: str
Gets the name of this entity.

to_dict() → Dict
Converts entity to dict.

Returns Dictionary object representation of entity.

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

Returns An EntityV2Proto protobuf.

to_spec_proto() → feast.core.Entity_pb2.EntitySpecV2
Converts an EntityV2 object to its protobuf representation. Used when passing EntitySpecV2 object to Feast request.

Returns An EntitySpecV2 protobuf.

to_yaml()
Converts a entity to a YAML string.

Returns An entity string returned in YAML format.

property value_type: feast.value_type.ValueType
Gets the type of this entity.

CHAPTER

FIVE

FEATURE VIEW

class feast.feature_view.**FeatureView**(*name: str*, *entities: List[str]*, *ttl: Union[google.protobuf.duration_pb2.Duration, datetime.timedelta]*, *input: Optional[feast.data_source.DataSource] = None*, *batch_source: Optional[feast.data_source.DataSource] = None*, *stream_source: Optional[feast.data_source.DataSource] = None*, *features: Optional[List[feast.feature.Feature]] = None*, *tags: Optional[Dict[str, str]] = None*, *online: bool = True*)

A FeatureView defines a logical grouping of serveable features.

Parameters

- **name** – Name of the group of features.
- **entities** – The entities to which this group of features is associated.
- **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.
- **input** – The source of data where this group of features is stored.
- **batch_source** (*optional*) – The batch source of data where this group of features is stored.
- **stream_source** (*optional*) – The stream source of data where this group of features is stored.
- **features** (*optional*) – The set of features defined as part of this FeatureView.
- **tags** (*optional*) – A dictionary of key-value pairs used for organizing FeatureViews.

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.

infer_features_from_batch_source(*config: feast.repo_config.RepoConfig*)

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

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

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

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]*)
Sets the join_key_map by returning a copy of this feature view with that field set. 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.

Returns A copy of this FeatureView with the join_key_map replaced with the 'join_key_map' input.

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"}

),

],

)

with_name(*name: str*)
Renames this feature view by returning a copy of this feature view with an alias set for the feature view name. This rename operation is only used as part of query operations and will not modify the underlying FeatureView.

Parameters **name** – Name to assign to the FeatureView copy.

Returns A copy of this FeatureView with the name replaced with the 'name' input.

with_projection(*feature_view_projection: feast.feature_view_projection.FeatureViewProjection*)
Sets the feature view projection by returning a copy of this feature view with its projection set to the given projection. A projection is an object that stores the modifications to a feature view that is used during query operations.

Parameters **feature_view_projection** – The FeatureViewProjection object to link to this OnDemandFeatureView.

Returns A copy of this FeatureView with its projection replaced with the 'feature_view_projection' argument.

CHAPTER

SIX

ON DEMAND FEATURE VIEW

class feast.on_demand_feature_view.**OnDemandFeatureView**(*name: str*, *features: List[feast.feature.Feature]*, *inputs: Dict[str, Union[feast.feature_view.FeatureView, feast.data_source.RequestDataSource]]*, *udf: method*)

[Experimental] An OnDemandFeatureView defines on demand transformations on existing feature view values and request data.

Parameters

- **name** – Name of the group of features.
- **features** – Output schema of transformation with feature names
- **inputs** – The input feature views passed into the transform.
- **udf** – User defined transformation function that takes as input pandas dataframes

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.

feast.on_demand_feature_view.**on_demand_feature_view**(*features: List[feast.feature.Feature]*, *inputs: Dict[str, feast.feature_view.FeatureView]*)

Declare an on-demand feature view

Parameters

- **features** – Output schema with feature names
- **inputs** – The inputs passed into the transform.

Returns An On Demand Feature View.

CHAPTER

SEVEN

FEATURE

class feast.feature.**Feature**(*name: str*, *dtype: feast.value_type.ValueType*, *labels: Optional[Dict[str, str]] = None*)

A Feature represents a class of serveable feature.

Parameters

- **name** – Name of the feature.
- **dtype** – The type of the feature, such as string or float.
- **labels** (*optional*) – User-defined metadata in dictionary form.

property **dtype: feast.value_type.ValueType**

Gets the data type of this feature.

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

Parameters **feature_proto** – FeatureSpecV2 protobuf object

Returns Feature object

property **labels: Dict[str, str]**

Gets the labels of this feature.

property **name**

Gets the name of this feature.

to_proto() → feast.core.Feature_pb2.FeatureSpecV2

Converts Feature object to its Protocol Buffer representation.

Returns A FeatureSpecProto protobuf.

class feast.feature.**FeatureRef**(*name: str*, *feature_table: str*)

Feature Reference represents a reference to a specific feature.

classmethod **from_proto**(*proto: feast.serving.ServingService_pb2.FeatureReferenceV2*)

Construct a feature reference from the given FeatureReference proto

Parameters **proto** – Protobuf FeatureReference to construct from

Returns FeatureRef that refers to the given feature

classmethod **from_str**(*feature_ref_str: str*)

Parse the given string feature reference into FeatureRef model String feature reference should be in the format feature_table:feature. Where "feature_table" and "name" are the feature_table name and feature name respectively.

Parameters **feature_ref_str** – String representation of the feature reference

Returns FeatureRef that refers to the given feature

`to_proto()` → feast.serving.ServingService_pb2.FeatureReferenceV2
Convert and return this feature table reference to protobuf.

Returns Protobuf respresentation of this feature table reference.

CHAPTER

EIGHT

FEATURE SERVICE

class feast.feature_service.**FeatureService**(*name: str*, *features: List[Union[feast.feature_table.FeatureTable, feast.feature_view.FeatureView, feast.on_demand_feature_view.OnDemandFeatureView]]*, *tags: Optional[Dict[str, str]] = None*, *description: Optional[str] = None*)

A feature service is a logical grouping of features for retrieval (training or serving). The features grouped by a feature service may come from any number of feature views.

Parameters

- **name** – Unique name of the feature service.
- **features** – A list of Features that are grouped as part of this FeatureService. The list may contain Feature Views, Feature Tables, or a subset of either.
- **tags** (*optional*) – A dictionary of key-value pairs used for organizing Feature Services.

static 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.

to_proto() → feast.core.FeatureService_pb2.FeatureService

Converts a FeatureService to its protobuf representation.

Returns A FeatureServiceProto protobuf.

CHAPTER

NINE

REGISTRY

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

Registry: A registry allows for the management and persistence of feature definitions and related metadata.

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_table(*feature_table: feast.feature_table.FeatureTable*, *project: str*, *commit: bool = True*)

Registers a single feature table with Feast

Parameters

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

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

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

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

Parameters

- **name** – Name of feature table
- **project** – Feast project that this feature table 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

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

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

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

get_feature_table(*name: str*, *project: str*) → feast.feature_table.FeatureTable
Retrieves a feature table.

Parameters

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

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

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

Parameters

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

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

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 belongs to

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

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_tables(*project: str*) → List[feast.feature_table.FeatureTable]
Retrieve a list of feature tables from the registry

Parameters **project** – Filter feature tables based on project name

Returns List of feature tables

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

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

Returns List of on demand feature views

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 feature views

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

teardown()
Tears down (removes) the registry.

CHAPTER

TEN

PROVIDER

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

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

abstract online_read(*config: feast.repo_config.RepoConfig*, *table: Union[feast.feature_table.FeatureTable, 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 given an Entity Key. This is a low level interface, not expected to be used by the users directly.

Returns Data is returned as a list, one item per entity key. Each item in the list is a tuple of event_ts for the row, and the feature data as a dict from feature names to values. Values are returned as Value proto message.

abstract online_write_batch(*config: feast.repo_config.RepoConfig*, *table: Union[feast.feature_table.FeatureTable, 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 the online store. This is a low level interface, not expected to be used by the users directly.

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

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **table** – Feast FeatureTable
- **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.

abstract teardown_infra(*project: str*, *tables: Sequence[Union[feast.feature_table.FeatureTable, feast.feature_view.FeatureView]]*, *entities: Sequence[feast.entity.Entity]*)
Tear down all cloud resources for a repo.

Parameters

- **project** – Feast project to which tables belong
- **tables** – Tables that are declared in the feature repo.
- **entities** – Entities that are declared in the feature repo.

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

Reconcile cloud resources with the objects declared in the feature repo.

Parameters

- **project** – Project to which tables belong
- **tables_to_delete** – Tables that were deleted from the feature repo, so provider needs to clean up the corresponding cloud resources.
- **tables_to_keep** – Tables that are still in the feature repo. Depending on implementation, provider may or may not need to update the corresponding resources.
- **entities_to_delete** – Entities that were deleted from the feature repo, so provider needs to clean up the corresponding cloud resources.
- **entities_to_keep** – Entities that are still in the feature repo. Depending on implementation, provider may or may not need to update the corresponding resources.
- **partial** – if true, then tables_to_delete and tables_to_keep are *not* exhaustive lists. There may be other tables that are not touched by this update.

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.

online_read(*config: feast.repo_config.RepoConfig*, *table: Union[feast.feature_table.FeatureTable, 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 given an Entity Key. This is a low level interface, not expected to be used by the users directly.

Returns Data is returned as a list, one item per entity key. Each item in the list is a tuple of event_ts for the row, and the feature data as a dict from feature names to values. Values are returned as Value proto message.

online_write_batch(*config: feast.repo_config.RepoConfig*, *table: Union[feast.feature_table.FeatureTable, 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 the online store. This is a low level interface, not expected to be used by the users directly.

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

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **table** – Feast FeatureTable
- **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_infra(*project: str*, *tables: Sequence[Union[feast.feature_table.FeatureTable, feast.feature_view.FeatureView]]*, *entities: Sequence[feast.entity.Entity]*) → None

Tear down all cloud resources for a repo.

Parameters

- **project** – Feast project to which tables belong
- **tables** – Tables that are declared in the feature repo.
- **entities** – Entities that are declared in the feature repo.

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

Reconcile cloud resources with the objects declared in the feature repo.

Parameters

- **project** – Project to which tables belong
- **tables_to_delete** – Tables that were deleted from the feature repo, so provider needs to clean up the corresponding cloud resources.
- **tables_to_keep** – Tables that are still in the feature repo. Depending on implementation, provider may or may not need to update the corresponding resources.
- **entities_to_delete** – Entities that were deleted from the feature repo, so provider needs to clean up the corresponding cloud resources.
- **entities_to_keep** – Entities that are still in the feature repo. Depending on implementation, provider may or may not need to update the corresponding resources.
- **partial** – if true, then tables_to_delete and tables_to_keep are *not* exhaustive lists. There may be other tables that are not touched by this update.

10.2 Local Provider

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

This class only exists for backwards compatibility.

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[Union[feast.feature_table.FeatureTable, feast.feature_view.FeatureView]]*, *entities: Sequence[feast.entity.Entity]*) → None

Tear down all cloud resources for a repo.

Parameters

- **project** – Feast project to which tables belong
- **tables** – Tables that are declared in the feature repo.
- **entities** – Entities that are declared in the feature repo.

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

Reconcile cloud resources with the objects declared in the feature repo.

Parameters

- **project** – Project to which tables belong
- **tables_to_delete** – Tables that were deleted from the feature repo, so provider needs to clean up the corresponding cloud resources.
- **tables_to_keep** – Tables that are still in the feature repo. Depending on implementation, provider may or may not need to update the corresponding resources.
- **entities_to_delete** – Entities that were deleted from the feature repo, so provider needs to clean up the corresponding cloud resources.
- **entities_to_keep** – Entities that are still in the feature repo. Depending on implementation, provider may or may not need to update the corresponding resources.
- **partial** – if true, then tables_to_delete and tables_to_keep are *not* exhaustive lists. There may be other tables that are not touched by this update.

CHAPTER

ELEVEN

OFFLINE STORE

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

OfflineStore is an object used for all interaction between Feast and the service used for offline storage of features.

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]*, *event_timestamp_column: str*, *created_timestamp_column: Optional[str]*, *start_date: datetime.datetime*, *end_date: datetime.datetime*) → feast.infra.offline_stores.offline_store.RetrievalJob

Note that join_key_columns, feature_name_columns, event_timestamp_column, 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.

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

RetrievalJob is used to manage the execution of a historical feature retrieval

to_arrow() → pyarrow.lib.Table

Return dataset as pyarrow Table synchronously

to_df() → pandas.core.frame.DataFrame

Return dataset as Pandas DataFrame synchronously including on demand transforms

11.1 File Offline Store

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

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]*, *event_timestamp_column: str*, *created_timestamp_column: Optional[str]*, *start_date: datetime.datetime*, *end_date: datetime.datetime*) → feast.infra.offline_stores.offline_store.RetrievalJob

Note that join_key_columns, feature_name_columns, event_timestamp_column, 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.

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

Offline store config for local (file-based) store

type: typing_extensions.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

11.2 BigQuery Offline Store

class feast.infra.offline_stores.bigquery.**BigQueryOfflineStore**

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]*, *event_timestamp_column: str*, *created_timestamp_column: Optional[str]*, *start_date: datetime.datetime*, *end_date: datetime.datetime*) → feast.infra.offline_stores.offline_store.RetrievalJob

Note that join_key_columns, feature_name_columns, event_timestamp_column, 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.

class feast.infra.offline_stores.bigquery.**BigQueryOfflineStoreConfig**(*, *type: typing_extensions.Literal[bigquery] = 'bigquery'*, *dataset: pydantic.types.StrictStr = 'feast'*, *project_id: pydantic.types.StrictStr = None*, *location: pydantic.types.StrictStr = None*)

Offline store config for GCP BigQuery

dataset: pydantic.types.StrictStr
(optional) BigQuery Dataset name for temporary tables

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: typing_extensions.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

to_bigquery(*job_config: Optional[google.cloud.bigquery.job.query.QueryJobConfig] = None*, *timeout: int = 1800*, *retry_cadence: int = 10*) → Optional[str]

Triggers the execution of a historical feature retrieval query and exports the results to a BigQuery table. Runs for a maximum amount of time specified by the timeout parameter (defaulting to 30 minutes).

Parameters

- **job_config** – An optional bigquery.QueryJobConfig to specify options like destination table, dry run, etc.
- **timeout** – An optional number of seconds for setting the time limit of the QueryJob.
- **retry_cadence** – An optional number of seconds for setting how long the job should checked for completion.

Returns Returns the destination table name or returns None if job_config.dry_run is True.

to_sql() → str

Returns the SQL query that will be executed in BigQuery to build the historical feature table.

feast.infra.offline_stores.bigquery.**block_until_done**(*client: google.cloud.bigquery.client.Client*, *bq_job: Union[google.cloud.bigquery.job.query.QueryJob, google.cloud.bigquery.job.load.LoadJob]*, *timeout: int = 1800*, *retry_cadence: float = 1*)

Waits for bq_job to finish running, up to a maximum amount of time specified by the timeout parameter (defaulting to 30 minutes).

Parameters

- **client** – A bigquery.client.Client to monitor the bq_job.
- **bq_job** – The bigquery.job.QueryJob that blocks until done runnning.
- **timeout** – An optional number of seconds for setting the time limit of the job.
- **retry_cadence** – An optional number of seconds for setting how long the job should checked for completion.

Raises

- **BigQueryJobStillRunning exception if the function has blocked longer than 30 minutes.** –
- **BigQueryJobCancelled exception to signify when that the job has been cancelled (i.e. from timeout or KeyboardInterrupt)** –

11.3 Redshift Offline Store

class feast.infra.offline_stores.redshift.**RedshiftOfflineStore**

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]*, *event_timestamp_column: str*, *created_timestamp_column: Optional[str]*, *start_date: datetime.datetime*, *end_date: datetime.datetime*) → feast.infra.offline_stores.offline_store.RetrievalJob

Note that join_key_columns, feature_name_columns, event_timestamp_column, 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.

class feast.infra.offline_stores.redshift.**RedshiftOfflineStoreConfig**(*, *type: typing_extensions.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: typing_extensions.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.OnDemar

to_redshift(*table_name: str*) → None
Save dataset as a new Redshift table

to_s3() → str
Export dataset to S3 in Parquet format and return path

CHAPTER

TWELVE

ONLINE STORE

class feast.infra.online_stores.online_store.**OnlineStore**

OnlineStore is an object used for all interaction between Feast and the service used for online storage of features.

abstract online_read(*config: feast.repo_config.RepoConfig*, *table: Union[feast.feature_table.FeatureTable, 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 given an Entity Key. This is a low level interface, not expected to be used by the users directly.

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **table** – Feast FeatureTable or FeatureView
- **entity_keys** – a list of entity keys that should be read from the FeatureStore.
- **requested_features** – (Optional) A subset of the features that should be read from the FeatureStore.

Returns Data is returned as a list, one item per entity key. Each item in the list is a tuple of event_ts for the row, and the feature data as a dict from feature names to values. Values are returned as Value proto message.

abstract online_write_batch(*config: feast.repo_config.RepoConfig*, *table: Union[feast.feature_table.FeatureTable, 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 the online store. This is a low level interface, not expected to be used by the users directly.

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

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **table** – Feast FeatureTable or 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*) –

12.1 Sqlite Online Store

class feast.infra.online_stores.sqlite.**SqliteOnlineStore**

OnlineStore is an object used for all interaction between Feast and the service used for offline storage of features.

online_read(*config: feast.repo_config.RepoConfig*, *table: Union[feast.feature_table.FeatureTable, 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 given an Entity Key. This is a low level interface, not expected to be used by the users directly.

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **table** – Feast FeatureTable or FeatureView
- **entity_keys** – a list of entity keys that should be read from the FeatureStore.
- **requested_features** – (Optional) A subset of the features that should be read from the FeatureStore.

Returns Data is returned as a list, one item per entity key. Each item in the list is a tuple of event_ts for the row, and the feature data as a dict from feature names to values. Values are returned as Value proto message.

online_write_batch(*config: feast.repo_config.RepoConfig*, *table: Union[feast.feature_table.FeatureTable, 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 the online store. This is a low level interface, not expected to be used by the users directly.

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

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **table** – Feast FeatureTable or 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*) –

class feast.infra.online_stores.sqlite.**SqliteOnlineStoreConfig**(*, *type: typing_extensions.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: typing_extensions.Literal[sqlite, *feast.infra.online_stores.sqlite.SqliteOnlineStore*]
Online store type selector

12.2 Datastore Online Store

class feast.infra.online_stores.datastore.**DatastoreOnlineStore**

OnlineStore is an object used for all interaction between Feast and the service used for offline storage of features.

online_read(*config: feast.repo_config.RepoConfig*, *table: Union[feast.feature_table.FeatureTable, 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 given an Entity Key. This is a low level interface, not expected to be used by the users directly.

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **table** – Feast FeatureTable or FeatureView
- **entity_keys** – a list of entity keys that should be read from the FeatureStore.
- **requested_features** – (Optional) A subset of the features that should be read from the FeatureStore.

Returns Data is returned as a list, one item per entity key. Each item in the list is a tuple of event_ts for the row, and the feature data as a dict from feature names to values. Values are returned as Value proto message.

online_write_batch(*config: feast.repo_config.RepoConfig*, *table: Union[feast.feature_table.FeatureTable, 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 the online store. This is a low level interface, not expected to be used by the users directly.

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

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **table** – Feast FeatureTable or 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[Union[feast.feature_table.FeatureTable, feast.feature_view.FeatureView]]*, *entities: Sequence[feast.entity.Entity]*)
There's currently no teardown done for Datastore.

update(*config: feast.repo_config.RepoConfig*, *tables_to_delete: Sequence[Union[feast.feature_table.FeatureTable, feast.feature_view.FeatureView]]*, *tables_to_keep: Sequence[Union[feast.feature_table.FeatureTable, feast.feature_view.FeatureView]]*, *entities_to_delete: Sequence[feast.entity.Entity]*, *entities_to_keep: Sequence[feast.entity.Entity]*, *partial: bool*)

class feast.infra.online_stores.datastore.**DatastoreOnlineStoreConfig**(*, *type: typing_extensions.Literal[datastore] = 'datastore'*, *project_id: pydantic.types.StrictStr = None*, *namespace: pydantic.types.StrictStr = None*, *write_concurrency: pydantic.types.PositiveInt = 40*, *write_batch_size: 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: typing_extensions.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**

Online feature store for AWS DynamoDB.

online_read(*config: feast.repo_config.RepoConfig*, *table: Union[feast.feature_table.FeatureTable, 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 given an Entity Key. This is a low level interface, not expected to be used by the users directly.

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **table** – Feast FeatureTable or FeatureView
- **entity_keys** – a list of entity keys that should be read from the FeatureStore.
- **requested_features** – (Optional) A subset of the features that should be read from the FeatureStore.

Returns Data is returned as a list, one item per entity key. Each item in the list is a tuple of event_ts for the row, and the feature data as a dict from feature names to values. Values are returned as Value proto message.

online_write_batch(*config: feast.repo_config.RepoConfig*, *table: Union[feast.feature_table.FeatureTable, 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 the online store. This is a low level interface, not expected to be used by the users directly.

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

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **table** – Feast FeatureTable or 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*) –

class feast.infra.online_stores.dynamodb.**DynamoDBOnlineStoreConfig**(*, *type: typing_extensions.Literal[dynamodb] = 'dynamodb'*, *region: pydantic.types.StrictStr*)

Online store config for DynamoDB store

region: pydantic.types.StrictStr
AWS Region Name

type: typing_extensions.Literal[dynamodb]
Online store type selector

12.4 Redis Online Store

class feast.infra.online_stores.redis.**RedisOnlineStore**

online_read(*config: feast.repo_config.RepoConfig*, *table: Union[feast.feature_table.FeatureTable, 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 given an Entity Key. This is a low level interface, not expected to be used by the users directly.

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **table** – Feast FeatureTable or FeatureView
- **entity_keys** – a list of entity keys that should be read from the FeatureStore.
- **requested_features** – (Optional) A subset of the features that should be read from the FeatureStore.

Returns Data is returned as a list, one item per entity key. Each item in the list is a tuple of event_ts for the row, and the feature data as a dict from feature names to values. Values are returned as Value proto message.

online_write_batch(*config: feast.repo_config.RepoConfig*, *table: Union[feast.feature_table.FeatureTable, 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 the online store. This is a low level interface, not expected to be used by the users directly.

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

Parameters

- **config** – The RepoConfig for the current FeatureStore.
- **table** – Feast FeatureTable or 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[Union[feast.feature_table.FeatureTable, 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[Union[feast.feature_table.FeatureTable, feast.feature_view.FeatureView]]*, *tables_to_keep: Sequence[Union[feast.feature_table.FeatureTable, feast.feature_view.FeatureView]]*, *entities_to_delete: Sequence[feast.entity.Entity]*, *entities_to_keep: Sequence[feast.entity.Entity]*, *partial: bool*)
We delete the keys in redis for tables/views being removed.

class feast.infra.online_stores.redis.**RedisOnlineStoreConfig**(*, *type: typing_extensions.Literal[redis] = 'redis'*, *redis_type: feast.infra.online_stores.redis.RedisType = RedisType.redis*, *connection_string: pydantic.types.StrictStr = 'localhost:6379'*)
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

redis_type: ***feast.infra.online_stores.redis.RedisType***
redis or redis_cluster

Type Redis type

type: typing_extensions.Literal[redis]
Online store type selector

class feast.infra.online_stores.redis.**RedisType**(*value*)
An enumeration.

PYTHON MODULE INDEX

f

feast.feature_service, 21
feast.infra.aws, 30
feast.infra.gcp, 30
feast.infra.local, 29
feast.infra.offline_stores.redshift, 34
feast.infra.online_stores.datastore, 39
feast.infra.online_stores.dynamodb, 41
feast.infra.online_stores.online_store, 37
feast.infra.online_stores.redis, 42
feast.infra.online_stores.sqlite, 38
feast.infra.passthrough_provider, 28
feast.infra.provider, 27
feast.on_demand_feature_view, 17

INDEX

A

apply() (*feast.feature_store.FeatureStore method*), 1
apply_entity() (*feast.registry.Registry method*), 23
apply_feature_service() (*feast.registry.Registry method*), 23
apply_feature_table() (*feast.registry.Registry method*), 23
apply_feature_view() (*feast.registry.Registry method*), 23
apply_materialization() (*feast.registry.Registry method*), 23
AwsProvider (*class in feast.infra.aws*), 30

B

BigQueryOfflineStore (*class in feast.infra.offline_stores.bigquery*), 32
BigQueryOfflineStoreConfig (*class in feast.infra.offline_stores.bigquery*), 32
BigQueryRetrievalJob (*class in feast.infra.offline_stores.bigquery*), 32
block_until_done() (*in module feast.infra.offline_stores.bigquery*), 33

C

cache_ttl_seconds (*feast.repo_config.RegistryConfig attribute*), 9
cluster_id (*feast.infra.offline_stores.redshift.RedshiftOfflineStoreConfig attribute*), 34
commit() (*feast.registry.Registry method*), 24
config (*feast.feature_store.FeatureStore attribute*), 2
connection_string (*feast.infra.online_stores.redis.RedisOnlineStoreConfig attribute*), 43
created_timestamp (*feast.entity.Entity property*), 13
created_timestamp_column (*feast.data_source.DataSource property*), 11

D

database (*feast.infra.offline_stores.redshift.RedshiftOfflineStoreConfig attribute*), 34
dataset (*feast.infra.offline_stores.bigquery.BigQueryOfflineStoreConfig attribute*), 32
DataSource (*class in feast.data_source*), 11
DatastoreOnlineStore (*class in feast.infra.online_stores.datastore*), 39
DatastoreOnlineStoreConfig (*class in feast.infra.online_stores.datastore*), 40
date_partition_column (*feast.data_source.DataSource property*), 11
delete_feature_service() (*feast.feature_store.FeatureStore method*), 2
delete_feature_service() (*feast.registry.Registry method*), 24
delete_feature_table() (*feast.registry.Registry method*), 24
delete_feature_view() (*feast.feature_store.FeatureStore method*), 2
delete_feature_view() (*feast.registry.Registry method*), 24
description (*feast.entity.Entity property*), 13
dtype (*feast.feature.Feature property*), 19
DynamoDBOnlineStore (*class in feast.infra.online_stores.dynamodb*), 41
DynamoDBOnlineStoreConfig (*class in feast.infra.online_stores.dynamodb*), 41

E

ensure_request_data_values_exist() (*feast.feature_store.FeatureStore method*), 2
ensure_valid() (*feast.feature_view.FeatureView method*), 15
Entity (*class in feast.entity*), 13
event_timestamp_column (*feast.data_source.DataSource property*), 11

F

feast.data_source
    module, 11
feast.entity

module, 13
feast.feature
module, 19
feast.feature_service
module, 21
feast.feature_store
module, 1
feast.feature_view
module, 15
feast.infra.aws
module, 30
feast.infra.gcp
module, 30
feast.infra.local
module, 29
feast.infra.offline_stores.bigquery
module, 32
feast.infra.offline_stores.file
module, 31
feast.infra.offline_stores.offline_store
module, 31
feast.infra.offline_stores.redshift
module, 34
feast.infra.online_stores.datastore
module, 39
feast.infra.online_stores.dynamodb
module, 41
feast.infra.online_stores.online_store
module, 37
feast.infra.online_stores.redis
module, 42
feast.infra.online_stores.sqlite
module, 38
feast.infra.passthrough_provider
module, 28
feast.infra.provider
module, 27
feast.on_demand_feature_view
module, 17
feast.registry
module, 23
feast.repo_config
module, 9
FeastConfigBaseModel (*class in feast.repo_config*), 9
FeastConfigError, 9
Feature (*class in feast.feature*), 19
feature_server (*feast.repo_config.RepoConfig attribute*), 9
FeatureRef (*class in feast.feature*), 19
FeatureService (*class in feast.feature_service*), 21
FeatureStore (*class in feast.feature_store*), 1
FeatureView (*class in feast.feature_view*), 15
field_mapping (*feast.data_source.DataSource property*), 11
FileOfflineStore (*class in feast.infra.offline_stores.file*), 31
FileOfflineStoreConfig (*class in feast.infra.offline_stores.file*), 31
FileRetrievalJob (*class in feast.infra.offline_stores.file*), 32
flags (*feast.repo_config.RepoConfig attribute*), 9
from_dict() (*feast.entity.Entity class method*), 13
from_proto() (*feast.data_source.DataSource static method*), 11
from_proto() (*feast.data_source.RequestDataSource static method*), 12
from_proto() (*feast.entity.Entity class method*), 13
from_proto() (*feast.feature.Feature class method*), 19
from_proto() (*feast.feature.FeatureRef class method*), 19
from_proto() (*feast.feature_service.FeatureService static method*), 21
from_proto() (*feast.feature_view.FeatureView class method*), 15
from_proto() (*feast.on_demand_feature_view.OnDemandFeatureView class method*), 17
from_str() (*feast.feature.FeatureRef class method*), 19
from_yaml() (*feast.entity.Entity class method*), 13

G

GcpProvider (*class in feast.infra.gcp*), 30
get_entity() (*feast.feature_store.FeatureStore method*), 2
get_entity() (*feast.registry.Registry method*), 24
get_feature_server_endpoint() (*feast.feature_store.FeatureStore method*), 2
get_feature_server_endpoint() (*feast.infra.aws.AwsProvider method*), 30
get_feature_server_endpoint() (*feast.infra.provider.Provider method*), 27
get_feature_service() (*feast.feature_store.FeatureStore method*), 2
get_feature_service() (*feast.registry.Registry method*), 24
get_feature_table() (*feast.registry.Registry method*), 25
get_feature_view() (*feast.feature_store.FeatureStore method*), 2
get_feature_view() (*feast.registry.Registry method*), 25
get_historical_features() (*feast.feature_store.FeatureStore method*), 2
get_needed_request_data() (*feast.feature_store.FeatureStore method*), 4

get_on_demand_feature_view() (*feast.feature_store.FeatureStore method*), 4
get_on_demand_feature_view() (*feast.registry.Registry method*), 25
get_online_features() (*feast.feature_store.FeatureStore method*), 4
get_table_column_names_and_types() (*feast.data_source.DataSource method*), 11
get_table_column_names_and_types() (*feast.data_source.RequestDataSource method*), 12
get_table_query_string() (*feast.data_source.DataSource method*), 11

I

iam_role (*feast.infra.offline_stores.redshift.RedshiftOfflineStoreConfig attribute*), 34
infer_features() (*feast.on_demand_feature_view.OnDemandFeatureView method*), 17
infer_features_from_batch_source() (*feast.feature_view.FeatureView method*), 15
is_valid() (*feast.entity.Entity method*), 13

J

join_key (*feast.entity.Entity property*), 13

L

labels (*feast.entity.Entity property*), 14
labels (*feast.feature.Feature property*), 19
last_updated_timestamp (*feast.entity.Entity property*), 14
list_entities() (*feast.feature_store.FeatureStore method*), 5
list_entities() (*feast.registry.Registry method*), 25
list_feature_services() (*feast.feature_store.FeatureStore method*), 5
list_feature_services() (*feast.registry.Registry method*), 25
list_feature_tables() (*feast.registry.Registry method*), 25
list_feature_views() (*feast.feature_store.FeatureStore method*), 5
list_feature_views() (*feast.registry.Registry method*), 25
list_on_demand_feature_views() (*feast.feature_store.FeatureStore method*), 5
list_on_demand_feature_views() (*feast.registry.Registry method*), 26
list_request_feature_views() (*feast.feature_store.FeatureStore method*), 5
list_request_feature_views() (*feast.registry.Registry method*), 26
LocalProvider (*class in feast.infra.local*), 29
location (*feast.infra.offline_stores.bigquery.BigQueryOfflineStoreConfig attribute*), 32

M

materialize() (*feast.feature_store.FeatureStore method*), 5
materialize_incremental() (*feast.feature_store.FeatureStore method*), 6
module
    feast.data_source, 11
    feast.entity, 13
    feast.feature, 19
    feast.feature_service, 21
    feast.feature_store, 1
    feast.feature_view, 15
    feast.infra.aws, 30
    feast.infra.gcp, 30
    feast.infra.local, 29
    feast.infra.offline_stores.bigquery, 32
    feast.infra.offline_stores.file, 31
    feast.infra.offline_stores.offline_store, 31
    feast.infra.offline_stores.redshift, 34
    feast.infra.online_stores.datastore, 39
    feast.infra.online_stores.dynamodb, 41
    feast.infra.online_stores.online_store, 37
    feast.infra.online_stores.redis, 42
    feast.infra.online_stores.sqlite, 38
    feast.infra.passthrough_provider, 28
    feast.infra.provider, 27
    feast.on_demand_feature_view, 17
    feast.registry, 23
    feast.repo_config, 9
most_recent_end_time (*feast.feature_view.FeatureView property*), 15

N

name (*feast.data_source.RequestDataSource property*), 12
name (*feast.entity.Entity property*), 14
name (*feast.feature.Feature property*), 19
namespace (*feast.infra.online_stores.datastore.DatastoreOnlineStoreConfig attribute*), 40

O

offline_store (*feast.repo_config.RepoConfig attribute*), 9
OfflineStore (*class in feast.infra.offline_stores.offline_store*), 31
on_demand_feature_view() (*in module feast.on_demand_feature_view*), 17
OnDemandFeatureView (*class in feast.on_demand_feature_view*), 17
online_read() (*feast.infra.online_stores.datastore.DatastoreOnlineStore method*), 39
online_read() (*feast.infra.online_stores.dynamodb.DynamoDBOnlineStore method*), 41
online_read() (*feast.infra.online_stores.online_store.OnlineStore method*), 37
online_read() (*feast.infra.online_stores.redis.RedisOnlineStore method*), 42
online_read() (*feast.infra.online_stores.sqlite.SqliteOnlineStore method*), 38
online_read() (*feast.infra.passthrough_provider.PassthroughProvider method*), 28
online_read() (*feast.infra.provider.Provider method*), 27
online_store (*feast.repo_config.RepoConfig attribute*), 10
online_write_batch() (*feast.infra.online_stores.datastore.DatastoreOnlineStore method*), 39
online_write_batch() (*feast.infra.online_stores.dynamodb.DynamoDBOnlineStore method*), 41
online_write_batch() (*feast.infra.online_stores.online_store.OnlineStore method*), 37
online_write_batch() (*feast.infra.online_stores.redis.RedisOnlineStore method*), 42
online_write_batch() (*feast.infra.online_stores.sqlite.SqliteOnlineStore method*), 38
online_write_batch() (*feast.infra.passthrough_provider.PassthroughProvider method*), 28
online_write_batch() (*feast.infra.provider.Provider method*), 27
OnlineStore (*class in feast.infra.online_stores.online_store*), 37

P

PassthroughProvider (*class in feast.infra.passthrough_provider*), 28
path (*feast.infra.online_stores.sqlite.SqliteOnlineStoreConfig attribute*), 39
path (*feast.repo_config.RegistryConfig attribute*), 9
project (*feast.feature_store.FeatureStore property*), 6
project (*feast.repo_config.RepoConfig attribute*), 10
project_id (*feast.infra.offline_stores.bigquery.BigQueryOfflineStoreConfi attribute*), 32
project_id (*feast.infra.online_stores.datastore.DatastoreOnlineStoreConf attribute*), 40
Provider (*class in feast.infra.provider*), 27
provider (*feast.repo_config.RepoConfig attribute*), 10
pull_latest_from_table_or_query() (*feast.infra.offline_stores.bigquery.BigQueryOfflineStore static method*), 32
pull_latest_from_table_or_query() (*feast.infra.offline_stores.file.FileOfflineStore static method*), 31
pull_latest_from_table_or_query() (*feast.infra.offline_stores.offline_store.OfflineStore static method*), 31
pull_latest_from_table_or_query() (*feast.infra.offline_stores.redshift.RedshiftOfflineStore static method*), 34

R

redis_type (*feast.infra.online_stores.redis.RedisOnlineStoreConfig attribute*), 43
RedisOnlineStore (*class in feast.infra.online_stores.redis*), 42
RedisOnlineStoreConfig (*class in feast.infra.online_stores.redis*), 43
RedisType (*class in feast.infra.online_stores.redis*), 43
RedshiftOfflineStore (*class in feast.infra.offline_stores.redshift*), 34
RedshiftOfflineStoreConfig (*class in feast.infra.offline_stores.redshift*), 34
RedshiftRetrievalJob (*class in feast.infra.offline_stores.redshift*), 34
refresh() (*feast.registry.Registry method*), 26
refresh_registry() (*feast.feature_store.FeatureStore method*), 6
region (*feast.infra.offline_stores.redshift.RedshiftOfflineStoreConfig attribute*), 34
region (*feast.infra.online_stores.dynamodb.DynamoDBOnlineStoreConfig attribute*), 41
Registry (*class in feast.registry*), 23
registry (*feast.feature_store.FeatureStore property*), 7
registry (*feast.repo_config.RepoConfig attribute*), 10
registry_store_type (*feast.repo_config.RegistryConfig attribute*), 9
RegistryConfig (*class in feast.repo_config*), 9
repo_path (*feast.feature_store.FeatureStore attribute*), 7
RepoConfig (*class in feast.repo_config*), 9
RequestDataSource (*class in feast.data_source*), 12
RetrievalJob (*class in feast.infra.offline_stores.offline_store*), 31

S

s3_staging_location (*feast.infra.offline_stores.redshift.RedshiftOfflineStoreConfig attribute*), 34
schema (*feast.data_source.RequestDataSource property*), 12
serve() (*feast.feature_store.FeatureStore method*), 7
serve_transformations() (*feast.feature_store.FeatureStore method*), 7
source_datatype_to_feast_value_type() (*feast.data_source.DataSource static method*), 11
source_datatype_to_feast_value_type() (*feast.data_source.RequestDataSource static method*), 12
SourceType (*class in feast.data_source*), 12
SqliteOnlineStore (*class in feast.infra.online_stores.sqlite*), 38
SqliteOnlineStoreConfig (*class in feast.infra.online_stores.sqlite*), 39

T

teardown() (*feast.feature_store.FeatureStore method*), 7
teardown() (*feast.infra.online_stores.datastore.DatastoreOnlineStore method*), 40
teardown() (*feast.infra.online_stores.redis.RedisOnlineStore method*), 43
teardown() (*feast.registry.Registry method*), 26
teardown_infra() (*feast.infra.aws.AwsProvider method*), 30
teardown_infra() (*feast.infra.passthrough_provider.PassthroughProvider method*), 29
teardown_infra() (*feast.infra.provider.Provider method*), 27
to_arrow() (*feast.infra.offline_stores.offline_store.RetrievalJob method*), 31
to_bigquery() (*feast.infra.offline_stores.bigquery.BigQueryRetrievalJob method*), 33
to_df() (*feast.infra.offline_stores.offline_store.RetrievalJob method*), 31
to_dict() (*feast.entity.Entity method*), 14
to_proto() (*feast.data_source.DataSource method*), 12
to_proto() (*feast.data_source.RequestDataSource method*), 12
to_proto() (*feast.entity.Entity method*), 14
to_proto() (*feast.feature.Feature method*), 19
to_proto() (*feast.feature.FeatureRef method*), 20
to_proto() (*feast.feature_service.FeatureService method*), 21
to_proto() (*feast.feature_view.FeatureView method*), 16
to_proto() (*feast.on_demand_feature_view.OnDemandFeatureView method*), 17
to_redshift() (*feast.infra.offline_stores.redshift.RedshiftRetrievalJob method*), 35
to_s3() (*feast.infra.offline_stores.redshift.RedshiftRetrievalJob method*), 35
to_spec_proto() (*feast.entity.Entity method*), 14
to_sql() (*feast.infra.offline_stores.bigquery.BigQueryRetrievalJob method*), 33
to_yaml() (*feast.entity.Entity method*), 14
type (*feast.infra.offline_stores.bigquery.BigQueryOfflineStoreConfig attribute*), 32
type (*feast.infra.offline_stores.file.FileOfflineStoreConfig attribute*), 32
type (*feast.infra.offline_stores.redshift.RedshiftOfflineStoreConfig attribute*), 34
type (*feast.infra.online_stores.datastore.DatastoreOnlineStoreConfig attribute*), 40
type (*feast.infra.online_stores.dynamodb.DynamoDBOnlineStoreConfig attribute*), 42
type (*feast.infra.online_stores.redis.RedisOnlineStoreConfig attribute*), 43
type (*feast.infra.online_stores.sqlite.SqliteOnlineStoreConfig attribute*), 39

U

update() (*feast.infra.online_stores.datastore.DatastoreOnlineStore method*), 40
update() (*feast.infra.online_stores.redis.RedisOnlineStore method*), 43
update_infra() (*feast.infra.aws.AwsProvider method*), 30
update_infra() (*feast.infra.passthrough_provider.PassthroughProvider method*), 29
update_infra() (*feast.infra.provider.Provider method*), 28
user (*feast.infra.offline_stores.redshift.RedshiftOfflineStoreConfig attribute*), 34

V

validate() (*feast.data_source.DataSource method*), 12
validate() (*feast.data_source.RequestDataSource method*), 12
value_type (*feast.entity.Entity property*), 14
version() (*feast.feature_store.FeatureStore method*), 7

W

with_join_key_map() (*feast.feature_view.FeatureView method*), 16
with_name() (*feast.feature_view.FeatureView method*), 16
with_projection() (*feast.feature_view.FeatureView method*), 16
write_batch_size (*feast.infra.online_stores.datastore.DatastoreOnlineStore attribute*), 40

write_concurrency (*feast.infra.online_stores.datastore.DatastoreOnlineStoreConfig attribute*), 40