Developing a Destination Connector

A Destination is responsible for writingRecord to third party systems.

You need to implement the functions required by Destination and provide your own implementations. Information about individual functions are listed below.

destination.go

This file provides the main functionality of your Destination Connector.

- Destination
- Struct: Every Destination implementation needs to include anUnimplementedDestination
- to satisfy the interface. This allows us to potentially change the interface in the future while remaining backward compatible with existing Destination implementations. This struct can be modified to add additional fields that can be accessed throughout the lifecycle of the Connector.
- type
- Destination
- struct
- {
- sdk
- .
- UnimplementedDestination
- config DestinationConfig
- }
- NewDestination()
- : A constructor function for your Destination struct. Note that this is the same function that should be set as the value ofConnector.NewDestination
- . The constructor should be used to wrap your Destination in the default middleware. You can add additional middleware, but unless you have a very good reason, you should always include the default middleware.
- func
- NewDestination
- (
-)
- sdk
- .
- Destination
- {
- // Create Destination and wrap it in the default middleware.
- return
- sdk
- .
- DestinationWithMiddleware
- (
- &
- Destination
- {
- }
- ,
- sdk
- .
- DefaultDestinationMiddleware
- (
-)
- ...
-)
- }
- Parameters()
- : A map of named Parameters that describe how to configure the connector. This map is typically generated usingparamgen
- .
- func
- (
- d
- *

- Destination
-)
- Parameters
- (
-)
- map
- [
- string
-]
- sdk
- .
- Parameter
- {
- return
- d
- .
- config
- .
- Parameters
- (
-)
- }
- Configure()
- : Validates and stores configuration data for the connector. Any complex validation logic should be implemented here.
- func
- (
- d
- *
- Destination
-)
- Configure
- (
- ctx context
- .
- Context
- ,
- cfg
- map
- [
- string
-]
- string
-)
- error
- {
- err
- :=
- sdk
- .
- Util
- .
- ParseConfig
- (
- cfg
- ,
- &
- d
- .
- config
-)
- if
- err
- !=
- nil
- {
- return
- fmt
- .

- Errorf
- (
- "invalid config: %w"
- ,
- err
-)
- }
- // custom validations here
- return
- nil
- }
- Open()
- : Prepares the connector to start producing records based on the last known successful position. If needed, the connector should open connections in this function.
- func
- (
- d
- *
- Destination
-)
- Open
- (
- ctx context
- .
- Context
-)
- error
- {
- // Retrieve the directory path from the config
- directoryPath
- :=
- d
- .
- config
- .
- Directory
- // Check if the directory exists
- if
- _
- ,
- err
- :=
- os
- .
- Stat
- (
- directoryPath
-)
- ;
- os
- .
- IsNotExist
- (
- err
-)
- {
- // Create the directory if it doesn't exist
- err
- :=
- os
- .
- MkdirAll
- (
- directoryPath
- ,
- 0755
-)
- if

- err
- !=
- nil
- {
- return
- fmt
- .
- Errorf
- (
- "failed to create directory '%s': %w"
- ,
- directoryPath
- ,
- err
-)
- }
- }
- else
- if
- err
- !=
- nil
- {
- // Return any error other than the directory not existing
- return
- fmt
- .
- Errorf
- (
- "error checking directory '%s': %w"
- ,
- directoryPath
- ,
- err
-)
- }
- // The directory exists (or was just created), so we can proceed
- return
- nil
- }
- Write()
- : Writes len(records) from a slice ofsdk.Record
- objects received from the Conduit pipeline to the destination right away without caching. It should return the number of records written from the slice and any error encountered that caused the write to stop early.
- func
- (
- d
- *
- Destination
-)
- Write
- (
- ctx context
- .
- Context
- ,
- recs
- [
-]
- sdk
- .
- Record
-)
- (
- int
- ,
- error
-)

- {
- outputDir
- :=
- d
- .
- config
- .
- Directory
- for
- i
- ,
- r
- :=
- range
- recs
- {
- fileName
- ,
- ok
- :=
- r
- .
- Key
- .
- (
- sdk
- .
- RawData
-)
- if
- !
- ok
- ||
- len
- (
- fileName
-)
- ==
- 0
- {
- return
- i
- ,
- fmt
- .
- Errorf
- (
- "record key is invalid or not provided, record index: %v"
- ,
- i
-)
- }
- filePath
- :=
- filepath
- .
- Join
- (
- outputDir
- ,
- string
- (
- fileName
-)
-)
- if
- err
- :=

- d
- .
- writeToFile
- (
- filePath
- ,
- r
- .
- Payload
- .
- After
- .
- Bytes
- (
-)
-)
- ;
- err
- !=
- nil
- {
- return
- i
- ,
- fmt
- .
- Errorf
- (
- "failed to write record to file '%s', record index: %v, error: %w"
- ,
- filePath
- ,
- i
- ,
- err
-)
- }
- sdk
- .
- Logger
- (
- ctx
-)
- .
- Info
- (
-)
- .
- Msgf
- (
- "Wrote file %s to directory %s\n"
- ,
- string
- (
- fileName
-)
- ,
- outputDir
-)
- }
- return
- len
- (
- recs
-)
- ,
- nil
- }

- Ack()
- : Ack signals to the implementation that the record with the supplied position was successfully processed.
- func
- (
- d
- *
- Destination
-)
- Ack
- (
- ctx context
- .
- Context
- ,
- position sdk
- .
- Position
-)
- error
- {
- sdk
- .
- Logger
- (
- ctx
-)
- .
- Debug
- (
-)
- .
- Msg
- (
- "Record successfully processed"
-)
- return
- nil
- }
- Teardown()
- : Teardown signals to the connector that there will be no more calls to any other function. Any connections that were created in theOpen()
- function should be closed here.
- func
- (
- d
- *
- Destination
-)
- Teardown
- (
- ctx context
- .
- Context
-)
- error
- {
- return
- nil
- } Edit this page Previous Developing a Source Connector Next Using a Custom Connector