Configuring and Managing the Streaming Server

The Greenplum Streaming Server (GPSS) manages communication and data transfer between a client (for example, the Pivotal Greenplum-Informatica Connector) and Greenplum Database. You must configure and start a GPSS instance before you use the service to load data into Greenplum Database.

Topics in this section include:

* [**Prerequisites**](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/instcfgmgt.html#prereq)

* **[Registering the GPSS Extension](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/instcfgmgt.html" \l "register)**

* **[Configuring the Greenplum Streaming Server](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/instcfgmgt.html" \l "configure)**

* **[Running the Greenplum Streaming Server](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/instcfgmgt.html" \l "run)**
* [**Managing GPSS Log Files**](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/instcfgmgt.html#logfiles)

* **[Shadowing the Greenplum Database Password](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/instcfgmgt.html" \l "shadowpass)**

Prerequisites

The Greenplum Streaming Server gpss and gpsscli command line utilities are automatically installed with Greenplum Database version 5.16 and later.

Before you start a GPSS server instance, ensure that you:

* Install and start a compatible Greenplum Database version.
* Can identify the hostname of your master node.
* Can identify the port on which your Greenplum Database master server process is running, if it is not running on the default port (5432).
* Select one or more GPSS host machines that have connectivity to:
  + The GPSS client host systems.
  + The Greenplum Database master and all segment hosts.

If you are using the gpsscli client utility, ensure that you run the command on a host that has connectivity to:

* The client data source host systems. For example, for a Kafka data source, you must have connectivity to each broker host in the Kafka cluster.
* The Greenplum Database master and all segment hosts.

Registering the GPSS Extension

The Greenplum Database and the Greenplum Streaming Server download packages install the GPSS extension. This extension must be registered in each database in which Greenplum users use GPSS to write data to Greenplum tables.

GPSS automatically registers its extension in a database the first time a Greenplum superuser or the database owner initiates a load job. You must manually register the extension in a database if non-privileged Greenplum users will be the first or only users of GPSS in that database.

Perform the following procedure as a Greenplum Database superuser or the database owner to manually register the GPSS extension:

1. Open a new terminal window, log in to the Greenplum Database master host as the gpadmin administrative user, and set up the Greenplum environment. For example:
2. $ ssh gpadmin@gpmaster

gpadmin@gpmaster$ . /usr/local/greenplum-db/greenplum\_path.sh

1. Start the psql subsystem, connecting to a database in which you want to register the GPSS formatter function. For example:

gpmaster$ psql -d testdb

1. Enter the following command to register the extension:

testdb=# CREATE EXTENSION gpss;

1. Perform steps 2 and 3 for each database in which the Greenplum Streaming Server will write client data.

Configuring the Greenplum Streaming Server

You configure an invocation of the Greenplum Streaming Server via a JSON-formatted configuration file. This configuration file includes properties that identify the listen address of the GPSS service as well as the gpfdist service host, bind address, and port number. You can specify encryption options in the file, can configure a password shadow encode/decode key, and can aso configure whether GPSS reuses external tables.

The contents of a sample GPSS JSON configuration file named gpsscfg1.json follow:

{

"ListenAddress": {

"Host": "",

"Port": 5019

},

"Gpfdist": {

"Host": "",

"Port": 8319,

"ReuseTables": false,

"BindAddress": "127.0.0.1"

},

"Shadow": {

"Key": "a\_very\_secret\_key"

}

}

Refer to the **[gpss.json](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/ref/gpss-json.html" \l "topic1)** reference page for detailed information about the GPSS configuration file format and the configuration properties that the utility supports.

Note: If your Kafka or Greenplum Database clusters are using Kerberos authentication or SSL encryption, see [**Configuring the Streaming Server for Encryption and Authentication**](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/kerb-ssl-config.html#topic1).

Running the Greenplum Streaming Server

You use the gpss utility to start an instance of the Greenplum Streaming Server on the local host. When you run the command, you provide the name of the configuration file that defines the properties of the GPSS and gpfdist service instances. You can also specify the name of a directory to which gpss writes log files. For example, to start a GPSS instance specifying a log directory named gpsslogs relative to the current working directory:

$ gpss gpsscfg1.json --log-dir ./gpsslogs

The default mode of operation for gpss is to wait for, and then consume, job requests and data from a client. When run in this mode, gpss waits indefinitely. You can interrupt and exit the command with Control-c. You may also choose to run gpss in the background (&). In both cases, gpss writes log and status messages to stdout.

Note: gpss keeps track of the loading progress of client jobs in memory. When you stop a GPSS server instance, you lose all registered jobs. You must re-submit any previously-submitted jobs that you require after you restart the GPSS instance. gpss will resume a job from the last load offset.

Refer to the **[gpss](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/ref/gpss.html" \l "topic1)** reference page for additional information about this command.

Managing GPSS Log Files

If you specify the -l or --log-dir option when you start gpss or run a gpsscli subcommand, GPSS writes log messages to a file in the directory that you specify. If you do not provide this option, GPSS writes log messages to a file in the $HOME/gpAdminLogs directory.

GPSS writes server log messages to a file with the following naming format, where *date* identifies the date that the log file was created. This date reflects the date that you started the gpss server instance, or the date that the log was rotated for that server instance (see [**Rotating the GPSS Server Log File**](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/instcfgmgt.html#logfiles__log_rotate) below):

gpss\_*date*.log

GPSS writes client log messages to a file with this naming format, where *date* identifies the date that you ran the command:

gpsscli\_*date*.log

For example, gpss\_20181228.log and gpsscli\_20181228.log.

After GPSS creates a log file, it appends all server and client log messages written on *date* to the respective file.

Rotating the GPSS Server Log File

If the log file for a gpss server instance grows too large, you may choose to archive the current log and start fresh with an empty log file.

To prompt GPSS to rotate the *server* log file, you must:

1. Rename the existing log file. For example:

gpadmin@gpmaster$ mv *logdir*/gpss\_*date*.log *logdir*/gpss\_*date*.log.1

1. Send the SIGUSR2 signal to the gpss server process. You can obtain the process id of a GPSS instance by running the ps command. For example:
2. gpadmin@gpmaster$ ps -ef | grep gpss

gpadmin@gpmaster$ kill -SIGUSR2 *gpss\_pid*

Note: There may be more than one gpss server process running on the system. Be sure to send the signal to the desired process.

When it receives the signal, GPSS emits a log message that identifies the time at which it reset the log file. For example:

... -[INFO]:-Set gpss log file rotate at 20190911:20:59:36.093

Integrating with logrotate

You can configure and manage GPSS server log file rotation with the Linux **[logrotate](https://linux.die.net/man/8/logrotate" \t "_blank)** utility.

This sample logrotate configuration rotates and compresses the log file of each gpss server instance running on the system weekly or when the file reaches 10MB in size. It operates on log files that are written to the default location:

/home/gpadmin/gpAdminLogs/gpss\_\*.log {

rotate 5

weekly

size 10M

postrotate

pkill -SIGUSR2 gpss

endscript

compress

}

If this configuration is specified in a file named gpss\_rotate.conf residing in the current working directory, you integrate with the Linux logrotate system with the following command:

$ logrotate -s status -d gpss\_rotate.conf

You may choose to create a cron job to run this command daily.

Shadowing the Greenplum Database Password

When you use GPSS to load data into Greenplum Databse, you specify the Greenplum user/role password in the PASSWORD: property setting of a YAML-format load configuration file; see **[gpsscli.yaml](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/ref/gpsscli-yaml.html" \l "topic1)**.

You specify the Greenplum password in clear text. If your security requirements do not permit this, you can configure GPSS to encode and decode a shadow password string that the GPSS client and server use when communicating the Greenplum password.

Note: GPSS supports shadowing the Greenplum password only on load jobs that you submit and manage with the gpsscli subcommands. GPSS does not support shadowed passwords on load jobs that you submit with gpkafka load.

When you use this GPSS feature:

1. (Optional) You configure a Shadow:Key in the **[gpss.json](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/ref/gpss-json.html" \l "topic1)** configuration file that you specify when you start the GPSS instance. For example:
2. ...
3. },
4. "Shadow": {
5. "Key": "a\_very\_secret\_key"
6. }

...

1. You run the **[gpsscli shadow](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/ref/gpsscli-shadow.html" \l "topic1)** command on the ETL system to interactively generate the shadowed password. For example:
2. $ gpsscli shadow --config gpss.json
3. please input your password
4. changemeCHANGEMEchangeme

"SHADOW:ERTBKXDWLAJHUF5UOGJY34QTXIBNYP4ULTWVHIUZIF4UYFPRIJVA"

You can automate this step using a command similar to the following:

$ echo changemeCHANGEMEchangeme | gpsscli shadow --config gpss.json | tail -1

"SHADOW:ERTBKXDWLAJHUF5UOGJY34QTXIBNYP4ULTWVHIUZIF4UYFPRIJVA"

If you do not specify the --config gpss.json option, or this configuration file does not include a Shadow:Key setting, GPSS uses its default key to generate the shadow password string.

1. You specify the shadow password string returned by gpsscli shadow in the PASSWORD: property setting of a **[gpsscli.yaml](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/ref/gpsscli-yaml.html" \l "topic1)** load configuration file. For example:
2. DATABASE: testdb
3. USER: testuser
4. PASSWORD: "SHADOW:ERTBKXDWLAJHUF5UOGJY34QTXIBNYP4ULTWVHIUZIF4UYFPRIJVA"

...

Always quote the complete shadow password string.

1. You provide the load configuration file as an option to **[gpsscli submit](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/ref/gpsscli-submit.html" \l "topic1)** or **[gpsscli load](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/ref/gpsscli-load.html" \l "topic1)** when you submit the job.
2. The GPSS instance servicing the job uses its Shadow:Key, or the default key, to decode the shadowed password string specified in PASSWORD:, and connects with Greenplum Database.

Loading Kafka Data into Greenplum

You will perform the following tasks when you use the Greenplum-Kafka Integration to load Kafka data into a Greenplum Database table:

1. Ensure that you meet the **[Prerequisites](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/kafka/loading.html" \l "prereq)**.
2. Register the Greenplum Streaming Server (GPSS) extension as described in the [**Greenplum Streaming Server**](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/instcfgmgt.html#register) documentation.
3. [**Identify the format of the Kafka data**](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/kafka/loading.html#dataformat).
4. (Optional) [**Register custom data formatters**](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/kafka/loading.html#regcustom).
5. [**Construct the load configuration file**](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/kafka/loading.html#cfgfile).
6. [**Create the target Greenplum Database table**](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/kafka/loading.html#createtbl).
7. Assign Greenplum Database role permissions to the table, if required, as described in the **[Greenplum Streaming Server](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/loading-gpss.html" \l "assignperm)** documentation.

1. **[Run the gpkafka load command](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/kafka/loading.html" \l "runcmd)** to load the Kafka data into Greenplum Database.
2. [**Check the progress of the load operation**](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/kafka/loading.html#check).
3. Check for load errors as described in the **[Greenplum Streaming Server](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/loading-gpss.html" \l "checkerror)** documentation. (Note that the naming format for gpkafka log files is gpkafka\_*date*.log.)

Prerequisites

The Greenplum-Kafka Integration is installed when you install Greenplum Database. Before using the gpkafka utilities to load Kafka data to Greenplum Database, ensure that you:

* Meet the Prerequisites documented for the **[Greenplum Streaming Server](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/instcfgmgt.html" \l "prereq)**.
* Have access to a running Kafka cluster with ZooKeeper, and that you can identify the hostname(s) and port number(s) of the Kafka broker(s) serving the data.
* Can identify the Kafka topic of interest.
* Can run the command on a host that has connectivity to:
  + Each Kafka broker host in the Kafka cluster.
  + The Greenplum Database master and all segment hosts.

About Supported Kafka Message Data Formats

The Greenplum-Kafka Integration supports Kafka message key and value data in the following formats:

| **Format** | **Description** |
| --- | --- |
| avro | Avro-format data. gpkafka supports:   * Loading Kafka message key or value data from a single-object encoded Avro file. * Using the Avro schema of a Kafka message key and/or value registered in a Confluent Schema Registry to load Avro-format key and/or value data.   In both cases, gpkafka reads Avro data from Kafka only as a single JSON-type column.  gpkafka supports libz-, lzma- and snappy-compressed Avro data from Kafka. |
| binary | Binary format data. gpkafka reads binary data from Kafka only as a single bytea-type column. |
| csv | Comma-delimited text format data. |
| custom | Data of a custom format, parsed by a custom formatter. |
| delimited | Text data separated by a configurable delimiter. |
| json | JSON- or JSONB-format data. gpkafka reads JSON data from Kafka only as a single column.  Note: GPSS supports JSONB-format data only when loading to Greenplum 6. |

To write Kafka message data into a Greenplum Database table, you must identify the data format in the load configuration file.

Avro

Specify the avro format when your Kafka message data is a single-object encoded Avro file or you are using the Confluent Schema Registry to load Avro message key and/or value data. gpkafka reads Avro data from Kafka and loads it into a single JSON-type column. You must define a mapping if you want gpkafka to write the data into specific columns in the target Greenplum Database table.

Binary

Use the binary format when your Kafka message data is a stream of bytes. gpkafka reads binary data from Kafka and loads it into a single bytea-type column.

CSV

Use the csv format when your Kafka message data is comma-delimited text and conforms to [**RFC 4180**](https://tools.ietf.org/html/rfc4180). The message content may not contain line ending characters (CR and LF).

Data in csv format may appear in Kafka messages as follows:

"1313131","12","backorder","1313.13"

"3535353","11","shipped","761.35"

"7979797","11","partial","18.72"

Custom

The Greenplum-Kafka Integration provides a custom data formatter plug-in framework for Kafka messages using user-defined functions. The type of Kafka message data supported by a custom formatter is formatter-specific. For example, a custom formatter may support compressed or complex data.

Delimited Text

The Greenplum-Kafka Integration supports loading Kafka data delimited by one or more characters that you specify. Use the delimited format for such data. The delimiter may be a multi-byte value and up to 32 bytes in length. You cannot specify a quote or an escape character in the delimiter.

Sample data using a pipe ('|') delimiter character follows:

1313131|12|backorder|1313.13

3535353|11|shipped|761.35

7979797|11|partial|18.72

JSON

Specify the json format when your Kafka message data is in JSON or JSONB format. gpkafka reads JSON data from Kafka only as a single column. You must define a mapping if you want gpkafka to write the data into specific columns in the target Greenplum Database table.

Note: GPSS supports JSONB-format data only when loading to Greenplum 6.

Sample JSON message data:

{ "cust\_id": 1313131, "month": 12, "amount\_paid":1313.13 }

{ "cust\_id": 3535353, "month": 11, "amount\_paid":761.35 }

{ "cust\_id": 7979797, "month": 11, "amount\_paid":18.82 }

Registering a Custom Formatter

A custom data formatter for Kafka messages is a user-defined function. If you are using a custom formatter, you must create and register the formatter function in each database in which you will use it to write Kafka data to Greenplum tables.

Constructing the gpkafka.yaml Configuration File

You configure a data load operation from Kafka to Greenplum Database via a YAML-formatted configuration file. This configuration file includes parameters that identify the source Kafka data and information about the Greenplum Database connection and target table, as well as error and commit thresholds for the operation.

The Greenplum-Kafka Integration supports two versions of the YAML configuration file: VERSION: 1 and VERSION: 2. Version 2 of the configuration file format supports all features of Version 1 of the configuration file, and introduces support for loading both the Kafka message key and value to Greenplum.

Refer to the **[gpkafka.yaml](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/kafka/gpkafka-yaml.html" \l "topic1)** reference page for Version 1 configuration file contents and syntax. Refer to the [**gpkafka-v2.yaml**](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/kafka/gpkafka-yaml-v2.html#topic1) reference page for Version 2 configuration file format and the configuration parameters that this version supports.

Contents of a sample gpkafka Version 2 YAML configuration file named loadcfg2.yaml follows:

DATABASE: ops

USER: gpadmin

PASSWORD: changeme

HOST: mdw-1

PORT: 5432

VERSION: 2

KAFKA:

INPUT:

SOURCE:

BROKERS: kbrokerhost1:9092

TOPIC: customer\_expenses2

VALUE:

COLUMNS:

- NAME: c1

           TYPE: json

FORMAT: avro

AVRO\_OPTION:

SCHEMA\_REGISTRY\_ADDR: http://localhost:8081

KEY:

COLUMNS:

- NAME: key

TYPE: json

FORMAT: avro

AVRO\_OPTION:

SCHEMA\_REGISTRY\_ADDR: http://localhost:8081

FILTER: (c1->>'month')::int = 11

     ERROR\_LIMIT: 25

OUTPUT:

SCHEMA: payables

TABLE: expenses2

MAPPING:

- NAME: customer\_id

EXPRESSION: (c1->>'cust\_id')::int

- NAME: newcust

EXPRESSION: ((c1->>'cust\_id')::int > 5000000)::boolean

- NAME: expenses

EXPRESSION: (c1->>'expenses')::decimal

- NAME: tax\_due

EXPRESSION: ((c1->>'expenses')::decimal \* .075)::decimal

METADATA:

SCHEMA: gpkafka\_internal

COMMIT:

MINIMAL\_INTERVAL: 2000

Greenplum Database Options

You identify the Greenplum Database connection options via the DATABASE, USER, PASSWORD, HOST, and PORT parameters.

The VERSION parameter identifies the version of the gpkafka YAML configuration file. The default version is Version 1.

KAFKA:INPUT Options

Specify the Kafka brokers and topic of interest using the SOURCE block. *You must create the Kafka topic prior to loading data.*

When you provide a VALUE block, you must specify the COLUMNS and FORMAT parameters. The VALUE:COLUMNS block includes the name and type of each data element in the Kafka message. The default source-to-target data mapping behaviour of gpkafka is to match a column name as defined in COLUMNS:NAME with a column name in the target Greenplum Database OUTPUT:TABLE:

* You must identify the Kafka data elements in the order in which they appear in the Kafka message.
* You may specify NAME: \_\_IGNORED\_\_ to omit a Kafka message value data element from the load operation.
* You must provide the same name for each non-ignored Kafka data element and its associated Greenplum Database table column.
* You must specify an equivalent data type for each non-ignored Kafka data element and its associated Greenplum Database table column.

The VALUE:FORMAT keyword identifies the format of the Kafka message value. gpkafka supports comma-delimited text format (csv ) and data that is separated by a configurable delimiter (delimited). gpkafka also supports binary (binary), JSON/JSONB (json), custom (custom), and Avro (avro) format value data.

When you provide a KEY block, you must specify the COLUMNS and FORMAT parameters. The KEY:COLUMNS block includes the name and type of each element of the Kafka message key, and is subject to the same restrictions as identified for VALUE:COLUMNS above. The KEY:FORMAT keyword identifies the format of the Kafka message key. gpkafka supports avro, binary, csv, custom, delimited, and json format key data.

The FILTER parameter identifies a filter to apply to the Kafka input messages before the data is loaded into Greenplum Database. If the filter evaluates to true, gpkafka loads the message. The message is dropped if the filter evaluates to false. The filter string must be a valid SQL conditional expression and may reference one or more KEY or VALUE column names.

The ERROR\_LIMIT parameter identifies the number of errors or the error percentage threshold after which gpkafka should exit the load operation.

KAFKA:OUTPUT Options

You identify the target Greenplum Database schema name and table name via the KAFKA:OUTPUT: SCHEMA and TABLE parameters. *You must pre-create the Greenplum Database table before you attempt to load Kafka data.*

The default load mode is to insert Kafka data into the Greenplum Database table. gpkafka also supports updating and merging Kafka message data into a Greenplum table. You specify the load MODE, the MATCH\_COLUMNS and UPDATE\_COLUMNS, and any UPDATE\_CONDITIONs that must be met to merge or update the data. In MERGE MODE, you can also specify ORDER\_COLUMNS to filter out duplicates and a DELETE\_CONDITION.

You can override the default mapping of the INPUT VALUE:COLUMNS and KEY:COLUMNS by specifying a MAPPING block in which you identify the association between a specific column in the target Greenplum Database table and a Kafka message value or key data element. You can also map a Greenplum Database table column to a value expression.

Note: When you specify a MAPPING block, ensure that you provide entries for all Kafka data elements of interest - gpkafka does not automatically match column names when you provide a MAPPING.

About the Merge Load Mode

MERGE mode is similar to an UPSERT operation; gpkafka may insert new rows in the database, or may update an existing database row that satisfies match and update conditions. gpkafka deletes rows in MERGE mode when the data satisfies an optional DELETE\_CONDITION that you specify.

gpkafka stages a merge operation in a temporary table, generating the SQL to populate the temp table from the set of OUTPUT configuration properties that you provide.

gpkafka uses the following algorithm for MERGE mode processing:

1. Create a temporary table like the target table.
2. Generate the SQL to insert the source data into the temporary table.
   1. Add the MAPPINGS.
   2. Add the FILTER.
   3. Use MATCH\_COLUMNS and ORDER\_COLUMNS to filter out duplicates.
3. Update the target table from rows in the temporary table that satisfy MATCH\_COLUMNS, UPDATE\_COLUMNS, and UPDATE\_CONDITION.
4. Insert non-matching rows into the target table.
5. Delete rows in the target table that satisfy MATCH\_COLUMNS and the DELETE\_CONDITION.
6. Truncate the temporary table.

Other Options

The KAFKA:METADATA:SCHEMA parameter specifies the name of the Greenplum Database schema in which gpkafka creates external and history tables.

gpkafka commits Kafka data to the Greenplum Database table at the row and/or time intervals that you specify in the KAFKA:COMMIT: MAX\_ROW and/or MINIMAL\_INTERVAL parameters. You must specify at least one of these parameters.

gpkafka reads data from Kafka with the batch size and at the time interval that you specify in the KAFKA:POLL: BATCHSIZE and TIMEOUT parameters.

Note: The POLL properties are deprecated and will be removed in a future release.

You can configure gpkafka to execute a task (user-defined function or SQL commands) after GPSS reads a configurable number of batches from Kafka. Use the KAFKA:TASK: POST\_BATCH\_SQL and BATCH\_INTERVAL configuration parameters to specify the task and the batch interval.

Specify a KAFKA:PROPERTIES block to set Kafka consumer configuration properties. gpkafka sends the property names and values to Kafka when it instantiates a consumer for the load operation.

About KEYs, VALUEs, and FORMATs

You can specify any data format in the Version 2 configuration file KEY:FORMAT and VALUE:FORMAT parameters, with some restrictions. The Greenplum-Kafka Integration supports the following KEY:FORMAT and VALUE:FORMAT combinations:

| **KEY:FORMAT** | **VALUE:FORMAT** | **Description** |
| --- | --- | --- |
| any | none (VALUE block omitted) | gpkafka loads only the Kafka message key data, subject to any MAPPING that you specify, to Greenplum Database. |
| none (KEY block omitted) | any | Equivalent to gpkafka configuration file Version 1. gpkafka ignores the Kafka message key and loads only the Kafka message value data, subject to any MAPPING that you specify, to Greenplum Database. |
| csv | any | Not permitted. |
| any | csv | Not permitted. |
| avro, binary, delimited, or json | avro, binary, delimited, or json | Any combination is permitted. gpkafka loads both the Kafka message key and value data, subject to any MAPPING that you specify, to Greenplum Database. |

About Transforming and Mapping Kafka Input Data

You can define a MAPPING between the Kafka input data (VALUE:COLUMNS and KEY:COLUMNS) and the columns in the target Greenplum Database table. Defining a mapping may be useful when you have a multi-field input column (such as a JSON-type column), and you want to assign individual components of the input field to specific columns in the target table.

You might also use a MAPPING to assign a value expression to a target table column. The expression must be one that you could specify in the SELECT list of a query, and can include a constant value, a column reference, an operator invocation, a built-in or user-defined function call, and so forth.

If you choose to map more than one input column in an expression, you can can create a user-defined function to parse and transform the input column and return the columns of interest.

For example, suppose a Kafka producer emits the following JSON messages to a topic:

{ "customer\_id": 1313131, "some\_intfield": 12 }

{ "customer\_id": 77, "some\_intfield": 7 }

{ "customer\_id": 1234, "some\_intfield": 56 }

You could define a user-defined function, udf\_parse\_json(), to parse the data as follows:

=> CREATE OR REPLACE FUNCTION udf\_parse\_json(value json)

RETURNS TABLE (x int, y text)

LANGUAGE plpgsql AS $$

BEGIN

RETURN query

SELECT ((value->>'customer\_id')::int), ((value->>'some\_intfield')::text);

END $$;

This function returns the two fields in each JSON record, casting the fields to integer and text, respectively.

An example MAPPING for the topic data in a JSON-type KAFKA:INPUT:COLUMNS named jdata follows:

MAPPING:

cust\_id: (jdata->>'customer\_id')

field2: ((jdata->>'some\_intfield') \* .075)::decimal

j1, j2: (udf\_parse\_json(jdata)).\*

The Greenplum Database table definition for this example scenario is:

=> CREATE TABLE t1map( cust\_id int, field2 decimal(7,2), j1 int, j2 text );

Creating the Greenplum Table

You must pre-create the Greenplum table before you load Kafka data into Greenplum Database. You use the KAFKA:OUTPUT: SCHEMA and TABLE load configuration file parameters to identify the schema and table names.

The target Greenplum table definition must include each column that gpkafka will load into the table. The table definition may include additional columns; gpkafka ignores these columns, and loads no data into them.

The name and data type that you specify for a column of the target Greenplum Database table must match the name and data type of the related, non-ignored Kafka message element. If you have defined a column mapping, the name of the Greenplum Database column must match the target column name that you specified for the mapping, and the type must match the target column type or expression that you define.

The CREATE TABLE command for the target Greenplum Database table receiving the Kafka topic data defined in the loadcfg2.yaml file presented in the [**Constructing the gpkafka.yaml Configuration File**](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/kafka/loading.html#cfgfile) section follows:

testdb=# CREATE TABLE payables.expenses2( customer\_id int8, newcust bool,

expenses decimal(9,2), tax\_due decimal(7,2) );

Running the gpkafka load Command

Note: gpkafka load is a wrapper around the Greenplum Streaming Server (GPSS) gpss and gpsscli utilities. Starting in Greenplum Streaming Server version 1.3.2, gpkafka load no longer launches a gpss server instance, but rather calls the backend server code directly.

When you run gpkafka load, the command submits, starts, and stops a GPSS job on your behalf.

Pivotal recommends that you migrate to using the GPSS utilities directly.

You run the gpkafka load command to load Kafka data to Greenplum. When you run the command, you provide the name of the configuration file that defines the parameters of the load operation. For example:

$ gpkafka load loadcfg2.yaml

The default mode of operation for gpkafka load is to read all pending messages and then to wait for, and then consume, new Kafka messages. When running in this mode, gpkafka load waits indefinitely; you can interrupt and exit the command with Control-c.

To run the command in batch mode, you provide the --quit-at-eof option. In this mode, gpkafka load exits when there are no new messages in the Kafka stream.

gpkafka load resumes a subsequent data load operation specifying the same Kafka topic and target Greenplum Database table names from the last recorded offset.

Refer to the **[gpkafka load](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/kafka/gpkafka-load.html" \l "topic1)** reference page for additional information about this command.

Note: gpkafka cannot detect the addition of a new Kafka partition while a load operation is in progress. You must stop, and then restart the load operation to read Kafka messages published to the new partition.

Configuring the gpfdist Server Instance

The gpkafka load command uses the gpfdist or gpfdists protocol to load data into Greenplum. You can configure the protocol used for the load request by providing the --config *gpfdistconfig.json* option to the command, where *gpfdistconfig.json* identifies a GPSS configuration file that specifies gpfdist configuration in a Gpfdist protocol block. Refer to [**Configuring the Greenplum Streaming Server**](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/instcfgmgt.html#configure) in the Greenplum Streaming Server documentation for detailed information about the file format and properties supported.

Note: gpkafka load reads the configuration specified in the Gpfdist protocol block of the *gpfdistconfig.json* file; it ignores the GPSS configuration specified in the ListenAddress block of the file.

Or, you may choose to provide gpfdist host or port configuration settings on the gpkafka load command line by specifying the --gpfdist-host *hostaddr* or --gpfdist-port *portnum* options to the command. Any options that you specify on the command line override settings provided in the *gpfdistconfig.json* file.

About Kafka Offsets, Message Retention, and Loading

Kafka maintains a partitioned log for each topic, assigning each record/message within a partition a unique sequential id number. This id is referred to as an *offset*. Kafka retains, for each gpkafka load invocation specifying the same Kafka topic and Greenplum Database table names, the last offset within the log consumed by the load operation. The Greenplum-Kafka Integration also records this offset value.

Kafka persists a message for a configurable retention time period and/or log size, after which it purges messages from the log. Kafka topics or messages can also be purged on demand. This may result in an offset mismatch between Kafka and the Greenplum-Kafka Integration.

gpkafka load returns an error if its recorded offset for the Kafka topic and Greenplum Database table combination is behind that of the current earliest Kafka message offset for the topic, or when the earliest and latest offsets do not match.

When you receive one of these messages, you can choose to:

* Resume the load operation from the earliest available message published to the topic by specifying the --force‑reset‑earliest option to gpkafka load:

$ gpkafka load --force-reset-earliest loadcfg2.yaml

* Load only new messages published to the Kafka topic, by specifying the ‑‑force‑reset‑latest option with the command:

$ gpkafka load --force-reset-latest loadcfg2.yaml

* Load messages published since a specific timestamp (milliseconds since epoch), by specifying the --force‑reset‑timestamp option to gpkafka load. To determine the create time epoch timestamp for a Kafka message, run the Kafka console consumer on the topic specifying the --property print.timestamp=true option, and review the output. You can also use a converter such as **[EpocConverter](https://www.epochconverter.com/" \t "_blank)** to convert a human-readable date to epoch time.

$ gpkafka load --force-reset-timestamp 1571066212000 loadcfg2.yaml

Note: Specifying the --force‑reset‑<xxx> options when loading data may result in missing or duplicate messages. Use of these options outside of the offset mismatch scenario is discouraged.

Checking the Progress of a Load Operation

Note: gpkafka history is deprecated and will be removed in a future release.

You can check the commit history of a load operation with the gpkafka history command. When you run gpkafka history, you provide the name of the configuration file that defined the load operation of interest. For example:

$ gpkafka history loadcfg2.yaml

Sample command output:

PartitionID StartTime EndTime BeginOffset EndOffset

0 2018-07-13T16:19:11Z 2018-07-13T16:19:11Z 0 9

When you run gpkafka history without any options, it displays the latest commit. To view the complete commit history of a load operation, run the command with the --show-commit-history all argument.

Refer to the **[gpkafka history](https://greenplum.docs.pivotal.io/streaming-server/1-3-6/kafka/gpkafka-history.html" \l "topic1)** reference page for additional information about this command.

# Best Practices

This topic presents best practices to follow when you use the Greenplum Streaming Server Kafka Integration.

## Choosing a Commit Threshold

gpkafka supports two mechanisms to control how and when it commits data to Greenplum Database: a time period or a number of rows. You specify one or both of MINIMAL\_INTERVAL or MAX\_ROW in the Kafka load configuration file.

For best results, try various settings of MINIMAL\_INTERVAL to determine what value works best in your environment.

When message flow is heavy, GPSS may receive and buffer many messages during the MINIMAL\_INTERVAL time period. In this situation, also providing a MAX\_ROW setting may mitigate any high memory usage scenarios.