IBM Resilient

resilient

Incident Response Platform

DATA FEED INTEGRATION GUIDE v1.0

Resilient Incident Response Platform Data Feed Integration Guide

Version	Publication	Notes
1.0	June 2019	Initial release.

Table of Contents

1. Overview

This guide describes the Resilient Data Feed capability. This functionality allows a Resilient customer to maintain "replica" data from a Resilient platform for access by other tools, such as Business Intelligence solutions. The replications and updates are performed in near real-time. The data written represents the current state of incident data, notes, artifacts, data tables, and so on.

1.1. Architecture

This extension allows you to perform queries on Resilient data without having to access the Resilient database directly. You can then run business intelligence queries from other tools.

The data can be written to any or all of the following destinations:

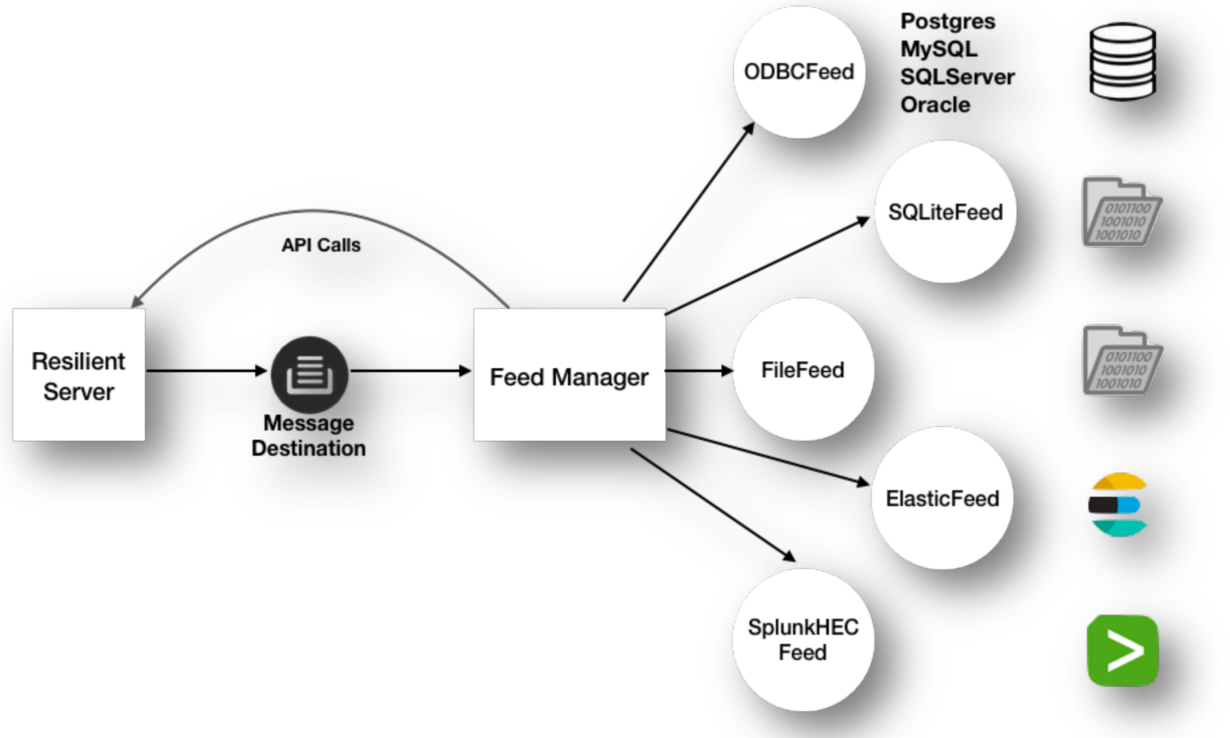
- ODBC databases:
  - PostgreSQL
  - MySQL (MariaDB)
  - Microsoft SQLServer
  - Oracle
- SQLite file/database
- ElasticSearch
- Splunk ES using the HTTP Event Collector
- Local directory (one file per object)

The content is consistent with the Resilient type/field semantic model and includes custom fields. For SQL destinations, the table name is the same as the type name (such as incident, task, and artifact) and the column names are the same as the field "programmatic name".

Data tables are supported (the programmatic name will be the same as the DB table name).

Newly added fields and data tables are dynamically added to the existing destination database.

The architecture allows other destinations to be created with relative ease. The following diagram shows the overall data flow from the Resilient platform to the existing feeds.



1.2. Initial Population

When you run the Data Feed extension against a Resilient platform that has pre-existing data, it can optionally read all that platform’s data and populate your feeds. After the initial data population, the extension performs all updates by listening on a message destination (queue) from which integration logic is called to update the data source.

1.3. Useful Tools

Most "business intelligence" tools allow you to query data from SQL databases. Here are some tools that you can use to run queries and reports against this data:

- Tableau
- Microsoft Power BI
- IBM Cognos
- Grafana
- Interactive SQL, for example:

```
-- Number of artifacts for each incident
select i.id, i.name, count(*) num_artifacts
from incident i join artifact a on i.id = a.inc_id
group by i.id, i.name;
```

2. Prerequisites

Before installing, verify that your environment meets the following prerequisites:

- Resilient platform is version 31 or later.
- You have access to a Resilient integration server. An *integration server* is the system that you use to deploy integration packages to the Resilient platform. See the Resilient Integration Server Guide (PDF) for more information.
- The following environments have been tested and are recommended:
  - PostgreSQL 9.6 or higher
  - MySQL 5.7 (MariaDB 10.3) or higher
  - Microsoft SQL Server 2017
  - Oracle 12c (Python 3.6 or greater required)
  - Splunk ES 7.1
  - ElasticSearch 7.0

3. Installation

The integration package contains Python components that are called by the Resilient platform. These components run in the Resilient Circuits integration framework. The package also includes Resilient customizations that will be imported into the platform later.

You perform these installation procedures at the Resilient integration server.

3.1. Install the Python components

Complete the following steps to install the Python components:

1. Ensure that the environment is up-to-date, as follows:

```
sudo pip install --upgrade pip
sudo pip install --upgrade setuptools
sudo pip install --upgrade resilient-circuits
```

2. Run the following commands to install the package:

```
unzip rc-data-feed-1.0.0.tar.gz.zip

[sudo] pip install --upgrade rc-data-feed-1.0.0.tar.gz
```

3.2. Configure the Python components

The Resilient Circuits process runs as an unprivileged user, typically named integration. If you do not already have an integration user configured on your appliance, create it now.

Complete the following steps to configure and run the integration:

1. Using sudo, switch to the integration user, as follows:

```
sudo su - integration
```

2. Use one of the following commands to create or update the resilient-circuits configuration file. Use `–c` for new environments or `–u` for existing environments.

```
resilient-circuits config -c
```

or

```
resilient-circuits config –u [-l rc-data-feed]
```

3. Edit the resilient-circuits configuration file, as follows:
   a. In the [resilient] section, ensure that you provide all the information required to connect to the Resilient platform.
   b. In the [feeds] section, define the feeds you intend to use and create separate sections for each feed. For example:

```
[feeds]
feed_names=my_postgresql_feed,my_file_feed,my_sqlite_feed
reload=True
# feed_data is the default queue that will be listened to
queue=feed_data

[my_postgresql_feed]
class=ODBCFeed
odbc_connect=Driver={PostgreSQL};Server=localhost;Port=5432;Database=feed
sql_dialect=PostgreSQL96Dialect
uid=mypguser
pwd=mypassword

[my_file_feed]
class=FileFeed
directory=/var/resilient/feed_data

[my_sqlite_feed]
class=SQLiteFeed
file_name=/var/resilient/feed.sqlite3
```

   See the Configuration section for details.

3.3. Deploy customizations to the Resilient platform

The package contains rules required for the flow of incident data changes to the data feeds.

1. Use the following command to deploy these customizations to the Resilient platform:

```
resilient-circuits customize [-l rc-data-feed]
```

2. Respond to the prompts to deploy the message destination and rules.

3.4. Run the integration framework

To test the integration package before running it in a production environment, you must run the integration manually with the following command:

```
resilient-circuits run
```

The resilient-circuits command starts, loads its components, and continues to run until interrupted. If it stops immediately with an error message, check your configuration values and retry.

3.5. Configure Resilient Circuits for restart

For normal operation, Resilient Circuits must run continuously. The recommend way to do this is to configure it to automatically run at startup. On a Red Hat appliance, this is done using a systemd unit file such as the one below. You may need to change the paths to your working directory and the app.config file.

1. The unit file must be named `resilient_circuits.service` To create the file, enter the following command:

   ```
   sudo vi /etc/systemd/system/resilient_circuits.service
   ```

2. Add the following contents to the file and change as necessary:

   ```
   [Unit]
   Description=Resilient-Circuits Service
   After=resilient.service
   Requires=resilient.service

   [Service]
   Type=simple
   User=integration
   WorkingDirectory=/home/integration
   ExecStart=/usr/local/bin/resilient-circuits run
   Restart=always
   TimeoutSec=10
   Environment=APP_CONFIG_FILE=/home/integration/.resilient/app.config
   Environment=APP_LOCK_FILE=/home/integration/.resilient/resilient_circuits.
   lock

   [Install]
   WantedBy=multi-user.target
   ```

3. Ensure that the service unit file is correctly permissioned, as follows:

   ```
   sudo chmod 664 /etc/systemd/system/resilient_circuits.service
   ```

4. Use the systemctl command to manually start, stop, restart and return status on the service:

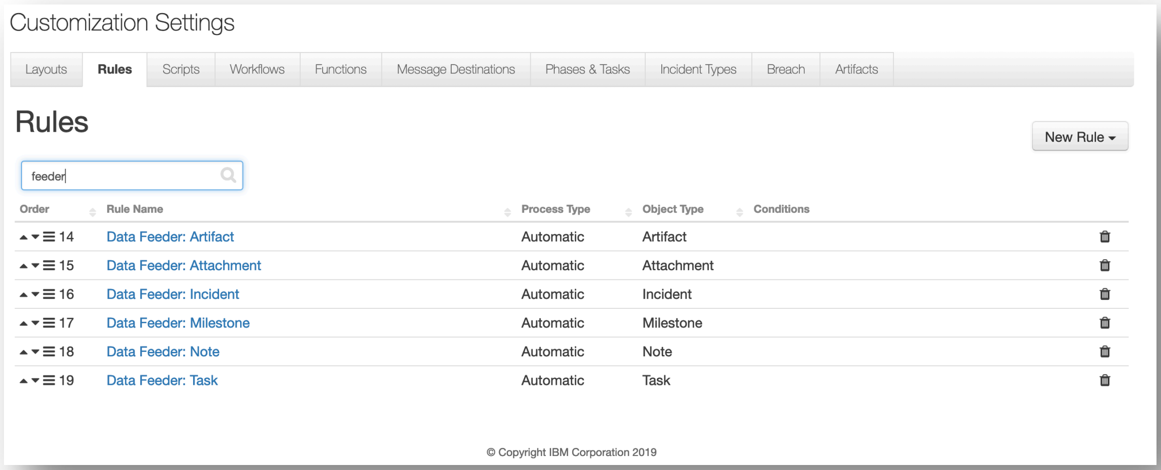
   ```
   sudo systemctl resilient_circuits [start|stop|restart|status]
   ```

You can view log files for systemd and the resilient-circuits service using the journalctl command, as follows:

```
sudo journalctl -u resilient_circuits --since "2 hours ago"
```

3.6. Confirm deployment

Once the package deployment is complete, you can view them in the Resilient platform Rules tab, as shown below.

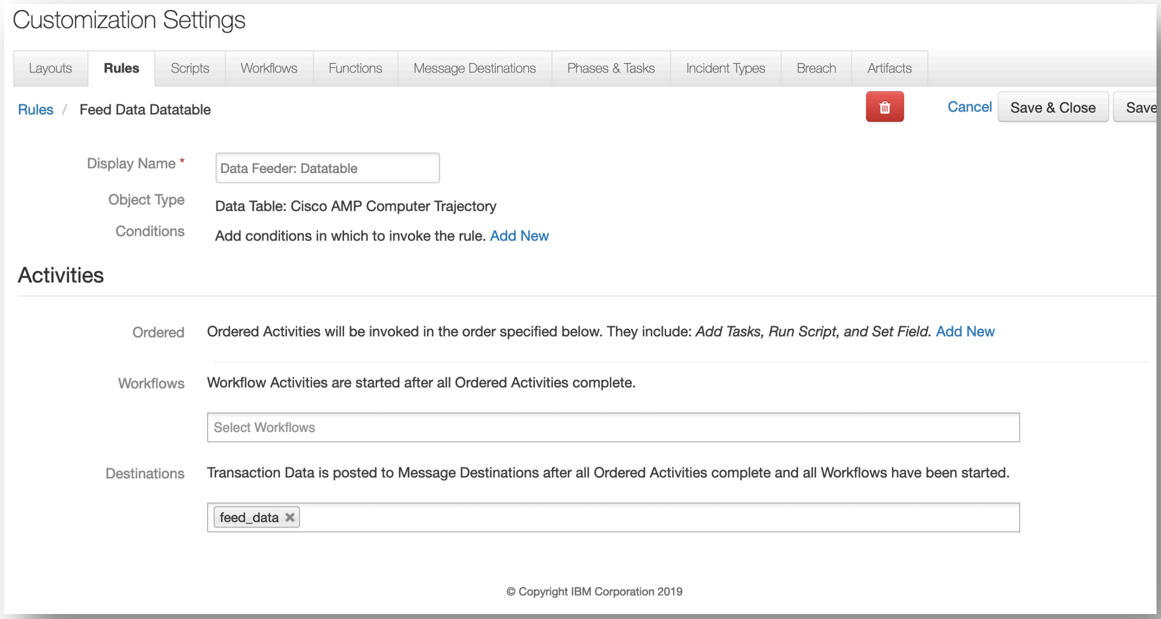


Order	Rule Name	Process Type	Object Type	Conditions
14	Data Feeder: Artifact	Automatic	Artifact	
15	Data Feeder: Attachment	Automatic	Attachment	
16	Data Feeder: Incident	Automatic	Incident	
17	Data Feeder: Milestone	Automatic	Milestone	
18	Data Feeder: Note	Automatic	Note	
19	Data Feeder: Task	Automatic	Task	

© Copyright IBM Corporation 2019

3.6.1. Configuration of Rules for data tables

Data tables are also supported by the Data Feeder. Because rules are written to specific data tables, they require a Resilient administrator to create them manually. The following screenshot shows a rule which references a data table and it triggers for any row change (insert, update, delete). Follow this procedure for all the data tables you need to include in the Data Feeder.



Display Name *	Data Feeder: Datatable
Object Type	Data Table: Cisco AMP Computer Trajectory
Conditions	Add conditions in which to invoke the rule. Add New

Activities

Ordered	Ordered Activities will be invoked in the order specified below. They include: *Add Tasks, Run Script, and Set Field.* Add New
Workflows	Workflow Activities are started after all Ordered Activities complete. Select Workflows
Destinations	Transaction Data is posted to Message Destinations after all Ordered Activities complete and all Workflows have been started. feed_data

© Copyright IBM Corporation 2019

4. Configuration

There are two aspects to configuration of the Data Feed extension: Resilient Server configuration and app.config configuration.

4.1. Resilient Circuits config file

The configuration goes in the standard app.config file. There is a "feeds" section that contains a "feed_names" key, which is a comma separated list of feeds that you want to enable. The names you use represent the names of the subsequent sections that contain the actual feed configurations. Each name must be unique. For example:

```
[feeds]
feed_names=my_postgresql_feed,my_file_feed,my_sqlite_feed
reload=True
reload_query_api_method=False
# feed_data is the default queue that will be listened to
queue=feed_data

[my_postgresql_feed]
class=ODBCFeed
odbc_connect=Driver={PostgreSQL};Server=localhost;Port=5432;Database=feed
sql_dialect=PostgreSQL96Dialect
uid=mypguser
pwd=mypassword

[my_file_feed]
class=FileFeed
directory=/var/resilient/feed_data

[my_sqlite_feed]
class=SQLiteFeed
file_name=/var/resilient/feed.sqlite3
```

In this example `feed_names` references three feeds. The comma separated names refer to additional sections of configuration settings.

This structure allows you to create as many feeds as you need, regardless of the "class". For example, you can have three different ODBC feeds as long as they all have unique names.

You can perform an initial data population by setting `reload=True` in the app.config [feeds] section and then restarting Resilient Circuits. Be aware that every time Resilient Circuits starts with reload=True, the entire set of Resilient incidents, notes, artifacts, and so on are refreshed in your feeds.

Two methods exist for synchronizing incidents and related data when `reload=True` is set. The default method uses a search API call which is the most efficient method. For very large incident lists, this method may fail with ElasticSearch errors. When this occurs, use `reload_query_api_method=True` to use an alternative (and slower method) of performing the initial synchronization.

The following sections explain the configuration options for each of the feed classes.

4.2. ODBCFeed class

The ODBCFeed class is probably the most flexible and useful of the feeds. It allows you to write all the incoming data to an ODBC database.

The following configuration items are supported:

Key	Values	Description
class	ODBCFeed	Indicates that the section is for an ODBCFeed.
odbc_connect	ODBC connect string	Example for PostgreSQL: `Driver={PostgreSQL};Server=localhost;Port=5432;Database=feed`
sql_dialect	PostgreSQL96Dialect, MariaDBDialect, SQLServerDialect, OracleDialect	Name of the SQL dialect.
uid	DB user name	Specify the database user name in this property and not in the connect string. Most DBs support the uid in the connect string but you should specify in this property instead.
pwd	DB password	Specify the database user's password in this property and not in the connect string. Most DBs support the pwd in the connect string but you should specify it in this property instead. You can use the standard Resilient Circuits mechanism for secure password storage.

When using a data feed database, IBM Resilient strongly recommends that you create and maintain the database on system separate from the Resilient platform, where queries cannot impact your running Resilient instance. Allowing access to the Resilient platform for a database instance can also compromise security of the platform itself.

4.2.1. Additional connection strings

The following table lists additional database connection strings for the other supported databases.

Database	Connection Strings
MariaDB	`Driver={MariaDB ODBC 3.0 Driver};Server=127.0.0.1;Port=3306;DB=<yourDB>;connectTimeout=0`[1]
Oracle	`Driver={Oracle 12c ODBC driver};DBQ=ORCLCDB`
SQLServer	`DRIVER={ODBC Driver 17 for SQL Server};SERVER=127.0.0.1;PORT=1443;DATABASE=<yourDB>;`

[1] See section **Error! Reference source not found.** for more information needed to setup mySql a nd MariaDB

Your naming of the database drivers (Ex. `{MariaDB ODBC 3.0 Driver})` may vary and is specified in your `odbcinst.ini` file.

Oracle has the further requirement of specifying the connection string references in a TNSNAMES.ORA file. Setting up the Oracle client environment will include the following environment variables (and may include others):

```
export LD_LIBRARY_PATH=/path/to/oracle/libraries/
export TNS_ADMIN=/path/to/tnsnames/
```

4.2.2. Database Field Length Considerations

Each database type has limits to the size of data stored. The following table describes the limits for each database.

Database	Field	Limit
Postgres	text	1GB
MySQL/MariaDB	text	4GB
MS SQLServer	varchar(max)	2GB
MS SQLServer	varchar(xx)	4000
Oracle	nvarchar2(xx)	2000

Some databases support blobs which can be supported specific cases. Blobs are presently unsupported.

4.2.3. Additional considerations

Some databases have reserve words which cannot be used in tables (such as *date* and *size*). If a Resilient custom field is found to be in a database reserve list, the name (for example, the column name) is altered to include a trailing '_'.

4.3. FileFeed class

The FileFeed class allows you to write all the incoming data to a directory on your file system (one file per object). The structure of the file names is either:

```
incident_{inc_id}.json (for incident objects)
```

or (for all other types of objects):

```
incident_{inc_id}_{type}_{obj_id}.json
```

In this context, "type" can be:

- task
- artifact
- milestone
- note
- data table programmatic name

The following configuration items are supported:

Key	Values	Description
class	FileFeed	Indicates that the section is for a FileFeed.
directory	Path for a directory on local system (required)	Location where the files are to be written.

4.4. SQLiteFeed class

The SQLiteFeed class allows you to write all the incoming data to a SQLite DB file. SQLite is very useful for testing and in cases where you want to have the data stored in a single file that you can easily share. Some tools may natively support SQLite as well.

SQLite supports CSV formatting, so you can easily export the data from the SQLite file into a CSV file, which can then be imported into another tool, such as Excel, for further analysis.

The following configuration items are supported:

Key	Values	Description
class	SQLiteFeed	Indicates that the section is for an SQLite.
file_name	Path for a local file on the system where the SQLite DB resides.	This is created if it does not exist. If it does exist, it must be an SQLite database.

4.5. ElasticFeed

This class allows you to write all incoming data to ElasticSearch. The data representation within Elastic is referenced by index, document type (incident, note, task, artifact, etc.) and document_id (incident_id, note_id, task_id, etc.).

The following configuration items are supported:

Key	Values	Description
class	ElasticFeed	Indicates that the section is for an ElasticSearch.
url	Ex. https://eliastic.yourorg.com	URL of Elastic server. Port is specified in it's own parameter.
port	Ex. 9200	Default is 9200
index	Ex. resilient	if using multiple organizations, consider indexes such as resilient<org_ID>
auth_user auth_password		User and password to authenticate to ElasticSearch.
cafile	True | False	Specify 'false' to bypass certification authentication

4.5.1. Considerations

- ElasticSearch allows for the updating to and deleting of individual documents. No data duplication occurs. A recently deleted custom datatable column may also not update until circuits is re-run or until the datatable is edited in the UI. Consult section 7.2 for datatable limitations in general.

4.6. SplunkHECFeed

The SplunkHECFeed class utilizes the Splunk HTTP Event Collector for data import. This is convenient as the data from Resilient is readily converted to JSON which can be natively consumed by Slunk.

Key	Values	Description
class	SplunkHECFeed	Indicates that the section is for Splunk ES.
token	Ex. 81e2d4bb-c008-49ac-a7a7-c0bf408c999	API token defined for the HEC data input.
host	Ex. splunk.yourorg.com	
port	Ex. 8088	The default is 8088
index	Ex data_feeder	The name of the index already defined.
event_host	Ex. myorg.com	Optional host name to record as the source of the events.
event_source=resilient	Ex. resilient	Optional source name of the events. Specifying a value improves searching
event_source_type		Optional source_type if one value is used for all events. If unspecified, each object type (incident, task, note, etc.) is used as the source_type
use_ssl	True | False	Indicate if connections to the HEC uses encryption (https)

4.6.1. Considerations

Enable the HTTP Event Collector within Splunk ES before using this data feed.

Splunk events are immutable. Resilient object changes are represented as new events. No event deletion is possible.

Be aware that when using `reload=True`, all Resilient records will be duplicated in Splunk each time resilient-circuits is re-started,

5. Test

A simple test can be performed by setting up a data feed with the `feed_names=file_feed` datastore and the parameter `reload=False`. Once configured and resilient-circuits is running, perform an update to an incident and confirm that the resilient-circuits logs contain an update

```
2019-04-04 21:16:02,165 INFO [file_feed] Inserting/updating incident; id = 2783460
```

The resulting file (`incident_2783460.json`) will contain a json formatted representation of the incident data similar to the following

```
{
  "addr": null,
  "alberta_health_risk_assessment": null,
  "hard_liability": 0,
  "city": null,
  "country": "United States",
  "creator_id": "able baker (a@example.com)",
  "crimestatus_id": "Unknown",
  "data_encrypted": null,
  "data_format": null,
  "end_date": "2019-03-14T13:06:48.550000",
  "create_date": "2019-02-13T18:38:55",
  "discovered_date": "2019-02-13T18:38:40",
  "start_date": null,
  "exposure_dept_id": null,
  "description": "<div class=\"rte\"><div>new description new again --
updated</div></div>",
  "employee_involved": null,
  …
}
```

Now repeat the same test with your production datastore(s). Data is not synchronized until new Resilient objects are created or existing ones are updated.

6. Preparation Checklist

In order to successfully use this extension, ensure you have built out your environments outlined in the sections above, and reviewed the requirements below:

1. Identified all the objects you will perform analytics on, including data tables. Data tables will require additional rules added for synchronization to your datastore.
2. Configured all aspects of your datastore
    a. For odbc datastores, install and test the appropriate drivers and configuration settings.
    b. Build out the database to use and the account has the correct permissions for data access.
3. Added the datastore configuration data to your app.config file, including the feed(s) to use.
4. Confirmed that your BI tool has access to the datastore environment.

5. Reviewed your use of the `reload` app.config setting as this will have potential performance consequences when resilient-circuits is restarted.
6. Completed the tests defined in section 5.

7. Configuration and Known Issues

There are a number of issues to be aware of when using the Data Feed

7.1. All Feeds

- When deleting a Resilient incident, that record will be deleted from the replica datastore. However, all the incident's associated tasks, notes, artifacts, datatables, etc. are not removed.
- For sizing purposes, here are some guidelines for the size of each object type. Use this information for estimation of your file system and database requirements.

Incident	~ 4000 characters
Tasks	2000-5000 characters
Notes	1000-3000 characters
Artifacts	< 500 characters
Attachments	Up to 20mb
Data tables	200-2000 characters per row

- Incident HIPAA fields when directly edited are not synchronized in near-time. Any additional change to an incident will then also update the HIPAA fields.
- Due to the structure of the internal Resilient database, some fields written to a data feeder datastore will always be blank or zero. Below is a short list of those fields:
    - Task - attachment_count, notes_count, at_id
- Moving an attachment from a task to the Incident will remove that attachment from the datastore. There is no workaround at present.

7.2. ODBC Databases

- All databases require their own set of drivers to be installed. This package does not install the libraries necessary as those operations are specific to the database used, Once the necessary driver(s) and installed, this information is captured in the `odbcinst.ini` file (for instance, found here: `/usr/local/etc/odbcinst.inifile`) similar to the following:

```
[PostresSQL Driver]
Description=ODBC for PostgreSQL
Driver=/usr/local/lib/psqlodbcw.so

[MariaDB ODBC 3.0 Driver]
Description=MariaDB Connector/ODBC v.3.0
Driver=/usr/local/Cellar/mariadb-connector-odbc/3.0.2/lib/libmaodbc.dylib
```

```
[ODBC Driver 17 for SQL Server]
Description=Microsoft ODBC Driver 17 for SQL Server
Driver=/usr/local/lib/libmsodbcsql.17.dylib
UsageCount=1

[Oracle 12c ODBC driver]
Description     = Oracle ODBC driver for Oracle 12c
Driver          = /usr/local/lib/libsqora.so.12.1
Setup           =
FileUsage       =
CPTimeout       =
CPReuse         =
```

- All data in Resilient is encoded as UTF-8. When creating the initial database, ensure that a similar encoding is specified.
- When creating the database user account which will access the defined database, provide the necessary permissions to allow full access to create tables, alter tables by adding columns, and full capability to insert, update and delete records.
  These are the permissions granted when creating a user to reference a SQLServer database:

  ```
  CREATE LOGIN res_db WITH PASSWORD = '*****';
  CREATE USER res_db FOR LOGIN res_db;
  GRANT UPDATE,INSERT,DELETE,SELECT,ALTER,CREATE TABLE TO res_db;
  ```

- Deleting a custom field in Resilient will not remove it from the Data Feed datastore.
- A custom field changed between Text and TextArea will have no effect in the Data Feed process.
- Python 2.7 with pyodbc 4.0.X will lose the millisecond precision on datetime fields. If this value is important to your environment, use Python 3.6.X or greater.
- A custom field (or custom database column) deleted in Resilient and recreated as a different data type (for instance, a text field recreated as number) will break the feed process. It's recommended one also delete the existing datastore table column so that it can be automatically recreated when resilient-circuits is restarted.
- Renaming a custom datatable field's api_name will create a new column in the database. The original field's api_name column will remain.
- No foreign keys are created between tables. However, each table contains the `inc_id` column which can be used to link tables together. Below is a sample SQL query linking an incident with its tasks.

  ```
  SELECT * FROM incident
  INNER JOIN task
  ON incident.inc_id = task.inc_id
  WHERE incident.inc_id=2095;
  ```

7.3. Datetime Fields and Timezones

All Resilient datetime fields retain their data in the UTC timezone. When persisting this data to a datastore, the ISO date format is used such as: `2019-04-18T19:07:42+00:00`.

Some databases, such as Postgres, convert query results with datetime fields into the timezone of the server system. To avoid this conversion, sql statements such `SET TIMEZONE='UTC'` should be used prior to any queries run.

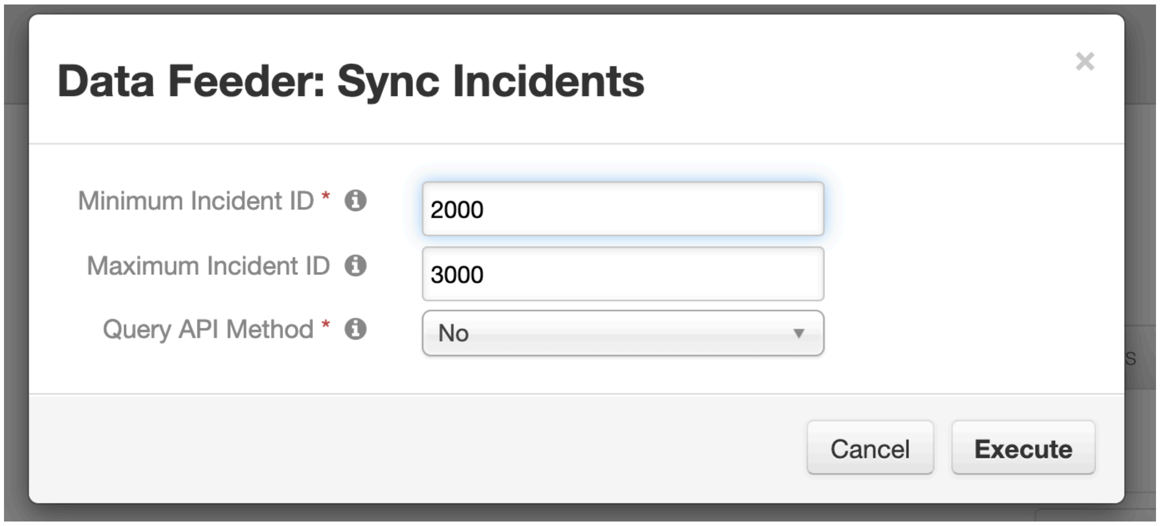
For MySql and MariaDB, `sql_mode` needs to be set without `STRICT_TRANS_TABLE`. See the database documentation on this setting: https://mariadb.com/kb/en/library/server-system-variables/#sql_mode.

8. Functions

The Data Feed solution also includes a rule, workflow and function which can be run within the Resilient UI to synchronize Incidents and their associated Tasks, Notes, Artifacts, etc. From the Actions menu within an incident, the `Data Feeder: Sync Incidents` rule allows one to specify a range of incidents to synchronize and which method of synchronization to use. It's suggested to initially use `Query API Method: No` for performance optimization.

This function is intended for the synchronization of a small number of incidents and is not intended to replace the `reload=True` app.config setting. Continue to use `reload=True` to initially load all existing incidents and their data.

Note: Only versions of Resilient >= 31 support this capability.



Data Feeder: Sync Incidents

Minimum Incident ID *	2000
Maximum Incident ID	3000
Query API Method *	No

Figure 1 Rule Activity Fields Example

9. Troubleshoot

There are several ways to verify the successful operation of a Resilient extension.

- Resilient Action Status

  When viewing an incident, use the Actions menu to view Action Status. By default, pending and errors are displayed. Modify the filter for actions to also show Completed actions. Clicking on an action displays additional information on the progress made or what error occurred.

- Resilient Scripting Log

  A separate log file is available to review scripting errors. This is useful when issues occur in the pre-processing or post-processing scripts. The default location for this log file is: `/var/log/resilient-scripting/resilient-scripting.log`.

- Resilient Logs

  By default, Resilient logs are retained at `/usr/share/co3/logs`. The `client.log` may contain additional information regarding the execution of functions.

- Resilient-Circuits

  The log is controlled in the `.resilient/app.config` file under the section `[resilient]` and the property `logdir`. The default file name is `app.log`. Each function will create progress information. Failures will show up as errors and may contain python trace statements.

9.1. reload=True issues

This app.config setting will synchronize all Resilient incidents and their related tasks, notes, incidents when resilient-circuits is started. Depending on the number of incidents, this process can take up to several hours.

It's possible to run into errors associated with an API call used (search_ex) when synchronizing incident tasks, notes, artifacts, etc. When this situation occurs, an error in the ElasticSearch log where appear as:

```
22:22:06.801 [http-bio-443-exec-16757] INFO
c.c.web.rest.Co3ExceptionMapperBase - Mapping exception to REST

com.co3.domain.exceptions.Co3IllegalArgumentException: There are one or more
invalid characters in the search query.

   at com.co3.search.ElasticSearchClient$ClientHelper.execute
(ElasticSearchClient.java:215)
```

If this should occur, edit the `app.config` file and change the `reload_query_api_method` parameter to True.
`reload_query_api_method=True`

This change will use an alternative method to synchronize all data. It should be noted that this method will take longer to complete the reload process.

10. Support

For additional support, contact support@resilientsystems.com.

Including relevant information from the log files will help us resolve your issue.

11. Modifications

Several code modifications are possible. This section gives you the necessary hints for how to consider making changes. In all cases, make sure you have a backup of the original code.

11.1. Adding new datastores

If you're interested in adding a completely new datastore, start by adding its name to the list defined in `components/feed_ingest.py` and adding an import statement for the Destination class

```
from rc_data_feed.lib.feed import FeedContext

from rc_data_feed.lib.my_feed import MyFeedDestination

…

class FeedComponent(ResilientComponent):
    """This component handles initial population of a feed and ongoing
    modifications from the associated queue."""

    DATATABLE_TYPE_ID = 8
    INCIDENT_TYPE_ID = 0
    INC_PAGE_SIZE = 500
    SEARCH_PAGE_SIZE = 50
    AVAILABLE_CLASSES = {
        "ODBCFeed": ODBCFeedDestination,
        "FileFeed": FileFeedDestination,
        "ElasticFeed": ElasticFeedDestination,
        "SQLiteFeed": SqliteFeedDestination,
        "SplunkHECFeed": SplunkHECFeedDestination,

        "MyFeed": MyFeedDestination
    }
```

Your new class will reside in the `lib/` directory and the class will inherent from `FeedDestinationBase` class.

Each new class will override the `FeedDestinationBase` `send_data()` function. This function defines the steps needed to convert the Resilient json data into the format necessary for your datastore. The helper routine `context.type_info.flatten()` can be used to flatten the Resilient json data to a simple key/value pair json representation which many datastores can natively consume.

The `send_data()` function can further support delete logic as well as insert/update logic by testing the boolean variable `context.is_deleted`.

11.2. SQL Dialects

Each SQL database dialect is defined in `lib/sql_dialect.py`. Here you'll find the existing dialect classes such as SqliteDialect, PostgresSQL96Dialect, MySqlDialect, SqlServerDialect and OracleDialect. Each of these dialects are referenced in your app.config file using the `sql_dialect=` parameter

```
[postgres_feed]
class=ODBCFeed
odbc_connect=Driver={PostresSQL
Driver};Server=127.0.0.1;DB=res_test;Port=5432;connectTimeout=0
sql_dialect=PostgreSQL96Dialect
```

```
uid=res_test
pwd=res_test
```

To add a new dialect, specify a new class inherited from the `ODBCDialectBase` base class.

Each dialect needs to define the logic for the operations below. These operations are defined in the `SqlDialect` base class. These functions build the SQL statements executed on by the standard sequence logic defined in `components/feed_ingest._populate_incidents()` and `_populate_others()`.

```
get_upsert()

get_delete()

get_create_table_if_not_exists()

get_add_column_to_table()
```

There are additional functions specified in `ODBCDialectBase` which may be overwritten:

```
get_parameters()

configure_connection()

get_column_type()
```

The best of advice is to copy an existing dialect and modify the existing functions to match the requirements of your database.

11.2.1. Modify Dialect Encoding

Each SQL dialect modifies the database connection with the encoding used to read and write data. The `configure_connection()` function makes these changes and in all cases, sets the encoding to UTF-8. In some cases, the format of the encoding parameters may change between python environments and the following logic can be used to account for those different environments:

```
def configure_connection(self, connection):
    connection.setdecoding(pyodbc.SQL_WCHAR, encoding='utf-8')  # pylint:
disable=c-extension-no-member
    if sys.version_info.major == 2: # to set encoding on python 2
        connection.setencoding(str, encoding='utf-8')
        connection.setencoding(unicode, encoding='utf-8')
    else: # an issue and try encoding without specifying fromtype
        connection.setencoding(encoding='utf-8')
```

11.2.2. Modify data type mapping

Each SQL dialect contains a mapping table to convert Resilient data types to your dialect's datastore. See the function `get_column_type()` for how mapping is presently done.

```
def get_column_type(self, input_type):  # pylint: disable=no-self-use
    """
    Gets the DB column type for the specified Resilient 'input type'

    :param input_type: The Resilient input type value (e.g. datepicker,
        boolean, number, text, text_area, etc.)

    :returns The DB type to use for this dialect.
    """
    type_dict = dict(
        number='BIGINT',
        datepicker='DATE',
        datetimepicker='TIMESTAMP',
        boolean='BOOLEAN'
    )

    if input_type in type_dict:
        return type_dict[input_type]

    return 'TEXT'
```

Text data types can be challenging for some databases based on the data length limits. In those cases, use a constant to define the limit and truncate your data using that value. Be aware that character limits do not account for Unicode characters which are double-byte. This means that a data limit of 64k characters actually translates to 32k Unicode characters.

```
MAX_MARIADB_TEXT = 32000


def get_parameters(self, parameter_names, parameters):
    # Need to get a list that contains all the values in the same order as
parameter_names.
    bind_parameters = list()

    for name in parameter_names:
        bind_parameters.append(parameters[name][:MAX_MARIADB_TEXT] if
isinstance(parameters[name], string_types) else parameters[name])
```

11.2.3. Modifying dialect reserved words

Each SQL Dialect contains reserved words which cannot be used in database table and column definitions. To ensure all data fields in Resilient can be stored, a list of reserve words is maintained. When a conflict is found the resulting table or column name has an underscore added to the name. From time to time, new database releases add to this reserved word list. Just add those words to an existing list:

```
RESERVE_LIST = ['all', 'analyse', 'analyze', 'and', 'any', 'array', 'as',
                'asc', 'asymmetric', 'both', 'case', 'cast',
                'check', 'collate', 'column', 'constraint', 'create',
                'current_date', 'current_role', 'current_time',
                'current_timestamp', 'current_user', 'default',

                …
                ]
```

11.3.Datastore testing

When adding any new datastore or modifying an existing one, it's good practice to create or exercise test cases to ensure the data is properly stored. See the existing tests in the `tests/` folder for ways to ensure all data types are stored and retrieved correctly. These tests are run using `pytest`:

```
pytest -v tests/test_postressql.py

=================================================================
test session starts
==================================================================
collected 7 items


tests/test_postressql.py::test_get_parameters PASSED
tests/test_postressql.py::test_createtable PASSED
tests/test_postressql.py::test_insert_row PASSED
tests/test_postressql.py::test_update_row PASSED
tests/test_postressql.py::test_delete_row PASSED
tests/test_postressql.py::test_altertable PASSED
tests/test_postressql.py::test_droptable PASSED
```

12. Appendix: Schemas

This sample represents the schema for a Postgres DB built-in objects (incident, artifact, attachment, note, milestone and task). Your schema will vary when custom fields are added as well as datatables added. This schema presents a complete representation of the fields stored within Resilient.

This list was generated via the following sql command

```
SELECT columns.table_name, column_name, data_type FROM
information_schema.tables
inner join INFORMATION_SCHEMA.COLUMNS
on columns.table_name = tables.table_name
where tables.table_catalog='res_test' and tables.table_schema='public'
and tables.table_name in ('incident', 'artifact', 'note', 'task',
'milestone', 'attachment')
order by columns.table_name, column_name;
```

Table Name	Column Name	Data Type
artifact	description	text
artifact	hits	text
artifact	id	integer
artifact	inc_id	integer
artifact	relating	boolean
artifact	type	text
artifact	value	text

Table Name	Column Name	Data Type
attachment	content_type	text

attachment	created	timestamp without time zone
attachment	creator_id	text
attachment	id	integer
attachment	inc_id	integer
attachment	name	text
attachment	size	Bigint
Attachment	task_id	number

Table Name	Column Name	Data Type
incident	addr	text
incident	alberta_health_risk_assessment	boolean
incident	city	text
incident	confirmed	boolean
incident	country	text
incident	create_date	timestamp without time zone
incident	creator_id	text
incident	crimestatus_id	text
incident	data_compromised	boolean
incident	data_contained	boolean
incident	data_encrypted	boolean
incident	data_format	text
incident	data_source_ids	text
incident	description	text
incident	discovered_date	timestamp without time zone
incident	due_date	timestamp without time zone
incident	employee_involved	boolean
incident	end_date	timestamp without time zone

incident	exposure_dept_id	text
incident	exposure_individual_name	text
incident	exposure_type_id	text
incident	exposure_vendor_id	text
incident	gdpr_breach_circumstances	text
incident	gdpr_breach_type	text
incident	gdpr_breach_type_comment	text
incident	gdpr_consequences	text
incident	gdpr_consequences_comment	text
incident	gdpr_final_assessment	text
incident	gdpr_final_assessment_comment	text
incident	gdpr_harm_risk	text
incident	gdpr_identification	text
incident	gdpr_identification_comment	text
incident	gdpr_lawful_data_processing_categories	text
incident	gdpr_personal_data	text
incident	gdpr_personal_data_comment	text
incident	gdpr_subsequent_notification	boolean
incident	hard_liability	bigint
incident	harmstatus_id	text
incident	hipaa_acquired_comment	text
incident	hipaa_additional_misuse_comment	text
incident	hipaa_additional_misuse	boolean
incident	hipaa_adverse_comment	text
incident	hipaa_breach_comment	text
incident	hipaa_breach	boolean
incident	hipaa_adverse	boolean
incident	hipaa_acquired	boolean

incident	hipaa_misused_comment	text
incident	hipaa_misused	boolean
incident	id	integer
incident	impact_likely	boolean
incident	inc_id	integer
incident	inc_training	boolean
incident	incident_type_ids	text
incident	jurisdiction_name	text
incident	members	text
incident	name	text
incident	negative_pr_likely	boolean
incident	nist_attack_vectors	text
incident	owner_id	text
incident	org_id	number
incident	phase_id	text
incident	plan_status	text
incident	reporter	text
incident	resolution_id	text
incident	resolution_summary	text
incident	severity_code	text
incident	start_date	timestamp without time zone
incident	state	text
incident	workspace	text
incident	zip	text

Table Name	Column Name	Data Type
milestone	date	timestamp without time zone
milestone	description	text

milestone	id	integer
milestone	inc_id	integer
milestone	title	text

Table Name	Column Name	Data Type
note	create_date	timestamp without time zone
note	id	integer
note	inc_id	integer
note	mentioned_users	text
Note	task_id	number
note	text	text
note	user_id	text

Table Name	Column Name	Data Type
task	active	boolean
task	at_id	bigint
task	attachments_count	bigint
task	cat_name	text
task	category_id	text
task	closed_date	timestamp without time zone
task	custom	boolean
task	description	text
task	due_date	timestamp without time zone
task	id	integer
task	inc_id	integer
task	inc_name	text
task	init_date	timestamp without time zone

task	instr_text	text
task	instructions	text
task	members	text
task	name	text
task	notes_count	bigint
task	owner_id	text
task	phase_id	text
task	private	boolean
task	required	boolean
task	status	text