IBM Resilient

resilient

Incident Response Platform

Resilient Incident Response Platform Data Feed Integration Guide

Version	Publication	Notes
1.0	June 2019	Initial release.
1.1	July 2019	Kafka support
1.2	September 2019	Added file-feed template to app.config
2.0	November 2019	Break out different datasource solutions to a plugin capability
2.1.0	March 2021	Support for attachment conent

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. A plugin framework is available with plugins available for:

- 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)
- Kafka
- Resilient

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.
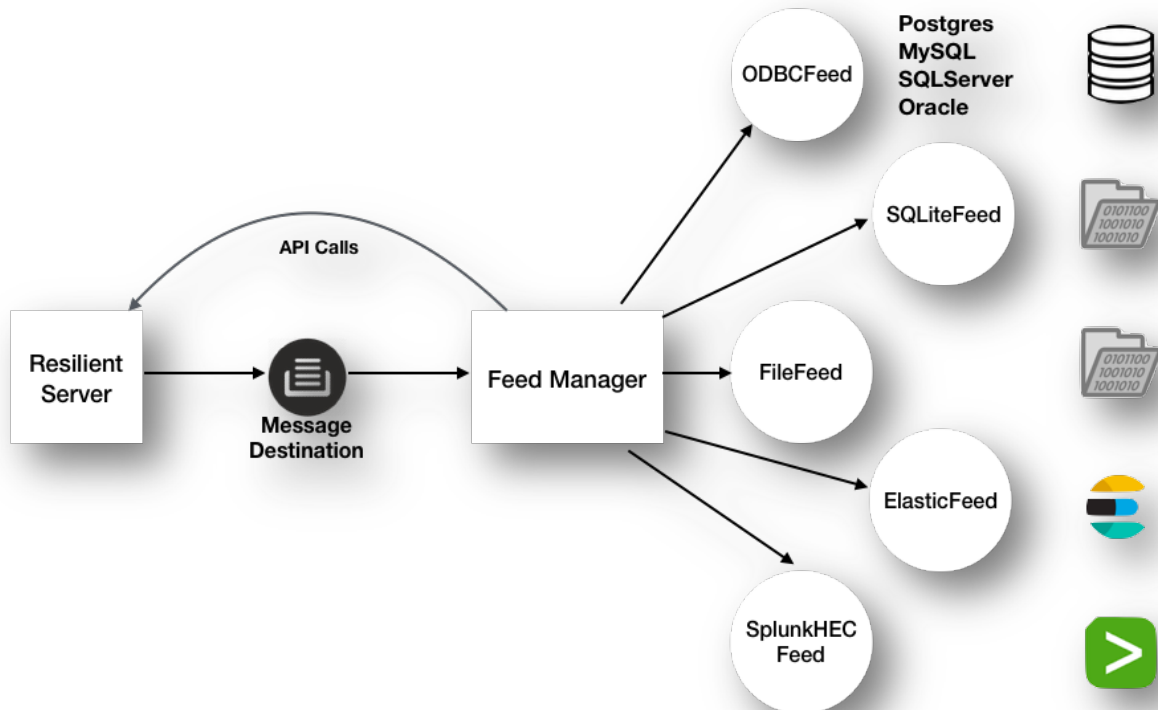
1.2. Initial Population

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 plugin environments have been tested and are recommended. These datasources are available as separate python packages which can be installed and made available to the data feed integration through references in your `app.config` file.

  - 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
  - Kafka
  - Resilient

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-<version>.zip
   ```

   ```
   [sudo] pip install --upgrade rc_data_feed-<version>.tar.gz
   ```

3. Plugins will have their own procedures for installation and library requirements. Refer to each plugin's README file for any special considerations to support.

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
# set to true if attachment data should be part of payload send to plugins
include_attachment_data = false

[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, two separate plugins have also been installed for sql databases (rc-data-feed-plugin-odbcfeed) and file system storage (rc-data-feed-plugin-filefeed).

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 <u>continuously</u>. 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.

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.

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
# set to true if attachment data should be part of payload send to plugins
include_attachment_data = false


[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. Two plugins have also been installed for sql databases (rc-data-feed-plugin-odbcfeed) and file system storage (rc-data-feed-plugin-filefeed).

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 section 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.

5. Test

A simple test can be performed by setting up a data feed with the `feed_names=file_feed` plugin and 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 plugin(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. Install the necessary plugins and 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. Add 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[1]	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:

  o 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. 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`.

See the README document for rc-data-feed-plugin-odbcfeed for specific database considerations

---
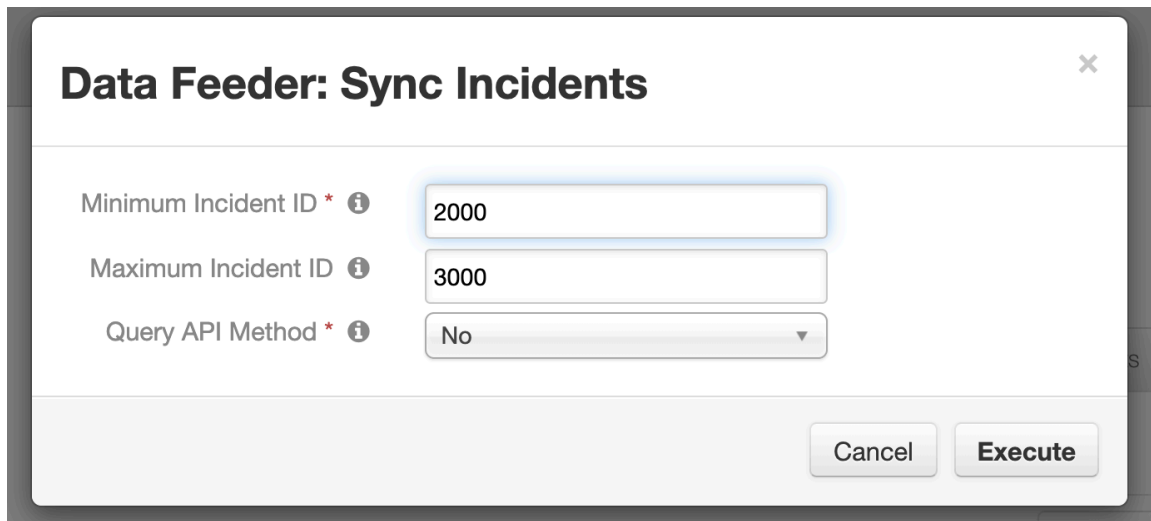
[1] Size based on including file content (see app.config setting: include_attachment_data)

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.

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.
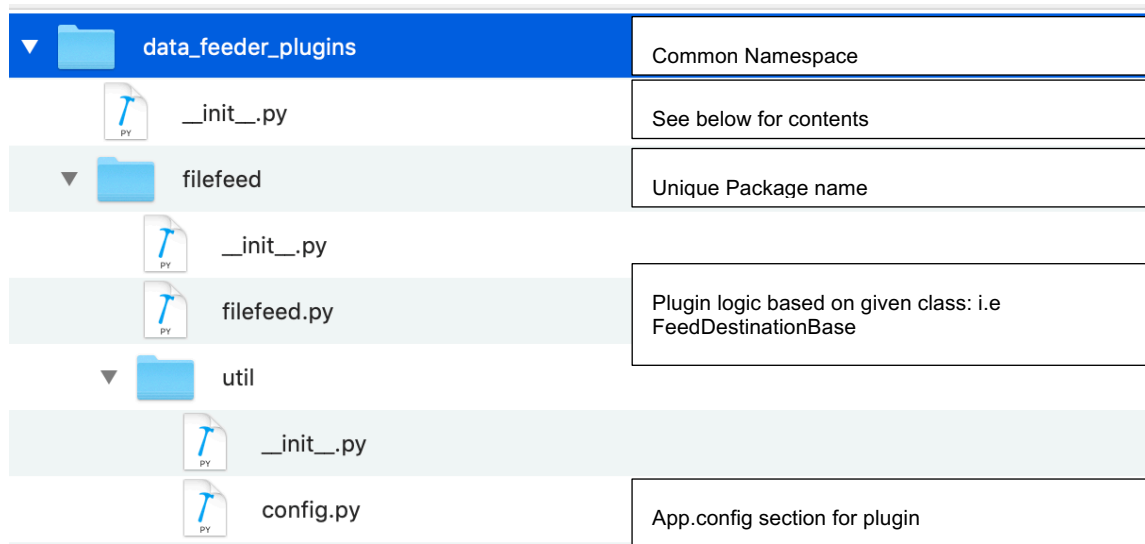
10. Support

For additional support, contact support@resilientsystems.com.

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

11. Modifications

Each datastore is defined as its own python library using a common namespace `data-feeder-plugins`. This common namespace allows for the dispatching of multiple plugins without conflicting with the other installed plugins. This diagram identifies key aspects of a plugin namespace:

▼ 📁 data_feeder_plugins	Common Namespace
📄 __init__.py	See below for contents
▼ 📁 filefeed	Unique Package name
📄 __init__.py	
📄 filefeed.py	Plugin logic based on given class: i.e FeedDestinationBase
▼ 📁 util	
📄 __init__.py	
📄 config.py	App.config section for plugin

For the namespace __init__.py file, the contents need to be:

```
from pkgutil import extend_path

__path__ = extend_path(__path__, __name__)
```

The FeedDestinationBase class can be extended to accommodate for different data structures. rc-data-feed-plugin-odbcfeed is one example of how to extend this class.

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
Attachment	content	blob[2]

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

---

[2] Only available when app.config setting: include_attachment_data=true

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