IBM Security

IBM

IBM Resilient SOAR Platform
Integration Server Guide
V34

Date: August 2019

IBM

Resilient SOAR Platform
Integration Server Guide

Platform Version	Publication	Notes
34.0	August 2019	Initial publication.

Contents

Chapter 1. Introduction

This guide provides the procedures to create and maintain a Resilient *integration server*, which is the system that you use to deploy extensions to the Resilient Security, Orchestration and Response (SOAR) Platform. This guide also provides the deployment procedures.

You can also use the Resilient integration server to write your own functions and custom actions, or implement custom threat services as described in the *Function Developer's Guide*, *Custom Action Developer's Guide* and *Custom Threat Service Guide*, respectively.

NOTE: As of V32.3, the name of the Resilient Incident Response Platform changed to Resilient Security, Orchestration and Response (SOAR) Platform. This guide supports all versions of the Resilient platform from V31.0 to V33.*x*.

Extensions and integrations

A Resilient *extension* is a software package that extends the functionality of the Resilient platform. It can contain one or more Resilient components and external code components. An external code component can access and return external data, interact or integrate with other security systems, or be a utility that performs a specific action.

There are various types of extensions, including functions and custom actions. The difference between custom actions and functions is where they send their results. A custom action can populate a custom field or data table within the Resilient platform. A function returns the results to the workflow that invoked the function, which allows the workflow to act upon the results.

An extension is also called an integration when its purpose is to communicate with another system.

IBM Resilient has a number of extensions in the form of functions and custom actions that you can download, customize and deploy. See "Downloading and installing a package" on page 27 for details about accessing these extensions.

Supported extensions

The following types of extensions require a Resilient integration server:

- Python-based, such as Resilient functions and some custom actions. The Resilient platform can be in an on-premises or cloud configuration.
- Java-based custom actions. The Resilient platform can be in an on-premises or cloud configuration.
- Custom threat services. The Resilient platform must be on-premises.

There are also extensions in the form of plug-ins, such as for Splunk and QRadar, which escalate incidents to the Resilient platform using the REST APIs. You do not need a Resilient integration server for these plug-ins.

If you have a package in a .res format, this means it contain components, such as scripts, workflows, and custom fields. You import these components into your Resilient platform using the import feature described in the *System Administrator Guide*.

Chapter 2. Architecture

The Resilient integration server must be able to access the Resilient platform and the hosts of the third-party applications that are integrated with the Resilient platform. The Resilient platform can reside in your own environment (known as an on-premises configuration) or within the IBM cloud (also known as a SaaS configuration).

More specifically, the integration server communicates to a specific organization within the Resilient platform. A *Resilient organization* is a self-contained area within the Resilient platform for managing incidents.

Standard Deployment

In a standard configuration, there is typically a single Resilient organization for all incidents. There may be multiple organizations for separate business divisions, as well as one organization for development and test and another for production. However, each organization is managed separately and users within a Resilient organization cannot view another organization.

The following network diagram is an example of a Resilient integration server used for python-based extensions and a Resilient platform in a cloud configuration. Resilient users access the platform from their web browsers.
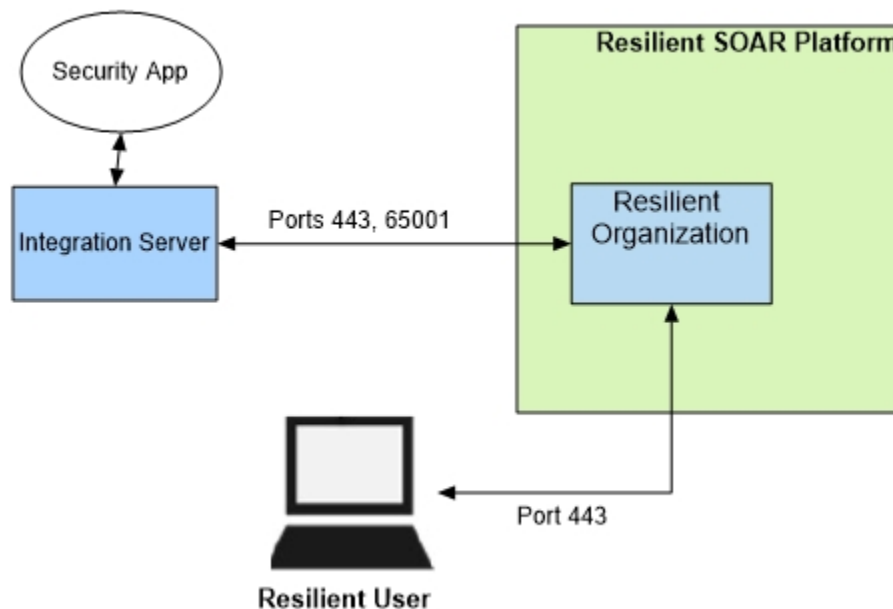
Resilient platform with MSSP add-on

The Resilient Managed Security Service Provider (MSSP) add-on is an optional feature that allows you to manage multiple Resilient child organizations from a single configuration organization. Each child organization can be assigned to a different group, division, or company to meet their incident response requirements.

If deploying extensions to a Resilient platform with the MSSP add-on, you need to deploy the extension to the configuration organization using the Resilient Circuits customize command, as you would in the standard deployment. The difference is that you do not deploy an extension to any child organization;

instead, the system administrator pushes the extension to the child organizations. You then create a separate integration server for each child organization to run the extension in the child organization.

You still need to install the extension package using the normal pip install command on each integration server whose Resilient child organization uses that package. And you need to configure the app.config file on each integration server.
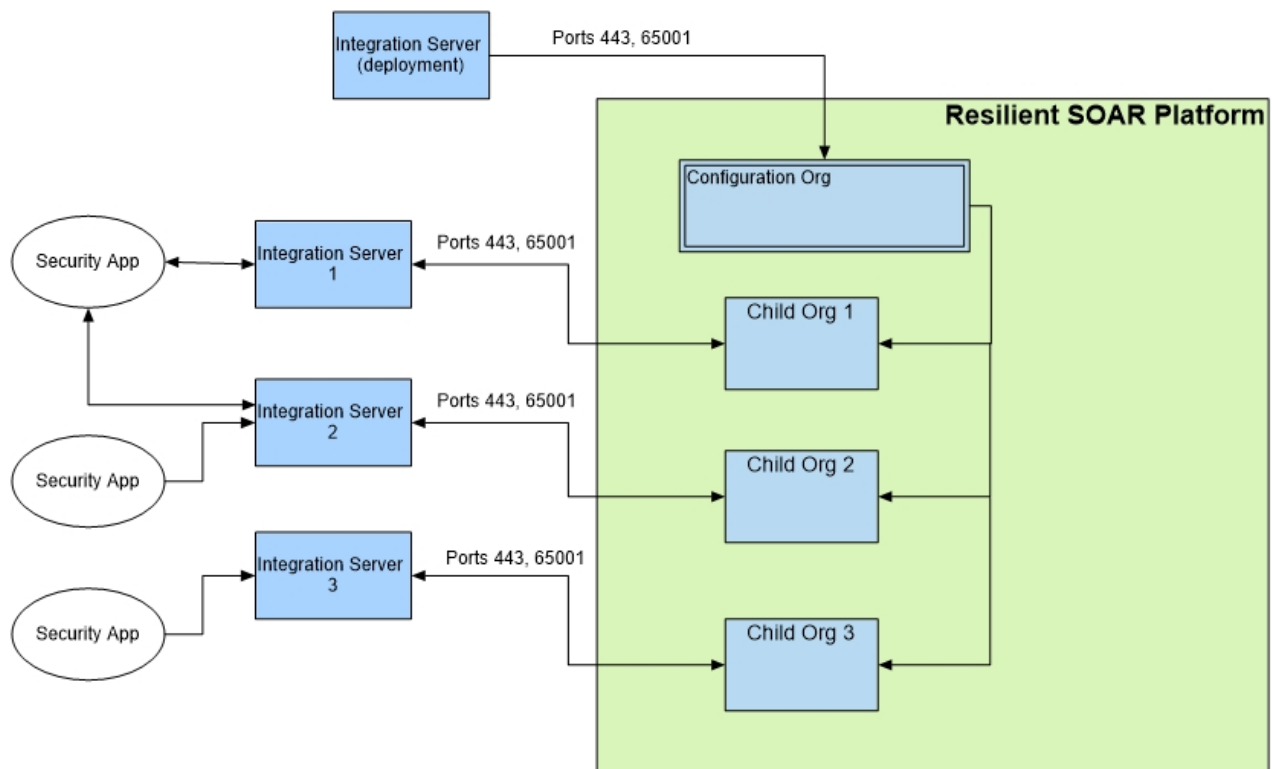
On-premises Resilient platform

If you have a Resilient platform in your environment, the Resilient integration server can be the same system as the one hosting the Resilient platform; however, IBM Resilient recommends that you keep them on separate systems for the following reasons:

• Security. Although all apps from the Community App Exchange are code scanned and tested, it is a security measure to not allow apps access to the command line of the Resilient platform host.

• Performance. Although the Resilient extensions are resource efficient, running them on the Resilient host could have impact on the available resources, especially if your extensions are heavily using in-memory processing.

• Access. The administrator of the Resilient integration server maintains, installs, and configures extensions. This administrator does not necessarily require access to the Resilient platform.

Chapter 3. Prerequisites

The following sections describe the system and network requirements. Specific extensions may have additional requirements, as described in the extension's documentation.

Resilient integration server

The following lists the requirements for the system hosting the Resilient integration server.

- Operating system:
  - Red Hat Enterprise Linux or Centos Linux 7.4.
  - Windows (Windows Enterprise Server 2016 or later is recommended)

    If using a Windows server, you also need to install the pywin32 library, which should be downloaded from here. Do not use the pypi/pip version of pywin32. Installation of the wrong version of the pywin32 library will likely result in a Resilient service that installs successfully but is unable to start.

- For Python-based extensions, Python 2.7.*x* (where *x* is 9 or later), or 3.6.*x* (where *x* is 4 or later). Make sure the Python installation includes pip. On a Linux system, you can verify the version of Python using the following command:

```
python --version
```

  NOTE: If using a keyring backend, the backend may require a later version of Python, as discussed in Choosing a keyring backend (optional).

- Minimum of 5GB free disk space.
- Minimum of 8 GB RAM.
- Dedicated operating system account for running extensions. The Resilient Circuits components run as an unprivileged user, typically named integration.
- Text editor, such as nano, to edit the configuration file.

The resources required by the Resilient integration server are variable due to the requirements of the extensions installed. Some extensions that operate on files in memory may have additional memory requirements. Extensions that perform decryption tasks may need more CPU. Therefore, you may need to increase those resources.

Resilient platform

The following lists the prerequisites for the Resilient platform:

- Version 31 or later. Some extensions may require a later version.
- Licensed for Resilient Action Module. If using custom threat services, the platform must also be licensed for the Resilient Security Module. You can verify this by running the following command on the Resilient platform:

```
resutil license
```

- Dedicated Resilient account to use as the API user specifically for the Resilient integration server. The Resilient system administrator provides the account and its credentials. The administrator provides one of the following accounts.
  - User account with user name (in the form of an email address) and password. With most extensions, the account must have the permission to view and edit incidents, and view and modify administrator

and customization settings. If the Resilient organization has Two Factor Authentication enabled, the account is configured as excluded.

– API key account with ID and secret, only if using V33 or later of the Resilient platform and Resilient Circuits. This is a more secure account. With most extensions, the account must have the permission to view and edit incidents, and view and modify administrator and customization settings. The API key account is not available with the Resilient MSSP add-on deployment.

For more information about Resilient accounts, refer to the *System Administrator Guide*.

IBM Resilient recommends that you use a Resilient platform in a test environment to create the function, message destination, rules, workflows and other components needed for your extension. This can be a separate Resilient platform or a Resilient organization within your platform dedicated for testing. Once tested, you can deploy the extension into any Resilient platform that is at the same or later version as your test platform.

Network configuration

If the Resilient platform is beyond a firewall, such as in a cloud configuration, configure the firewall to allow the Resilient integration server access to the following ports:

• 443. Required for the extension to connect to Resilient data using the REST API.

• 65000. Only if supporting Java-based extensions. The Resilient integration server communicates with port 65000 of the platform using ActiveMQ OpenWire.

• 65001. Only if supporting Python-based extensions. All Resilient functions are Python-based. The Resilient integration server communicates with port 65001 of the platform using the STOMP messaging protocol.

• 9000 (default). Only if supporting custom threat feeds, which is not applicable for a Resilient platform in a cloud configuration. The platform must be in your environment. The Resilient platform communicates with the Resilient integration server over HTTPS on a port that you choose (by default on port 9000).

Port 9000 is an "inbound-only" connection to the Resilient integration server, where the other connections are "outbound-only" from the integration server to the Resilient platform.

Chapter 4. Installation on a Linux system

To create a Resilient integration server on a Linux system, install the Resilient Circuits framework which includes its dependent modules.

Downloading Resilient Circuits (offline only)

Use this procedure only if your Resilient integration server does not have access to the Internet. Otherwise, use the pip install procedure in the next section.

If your Resilient integration server is not connected to the Internet, you need to download the Resilient Circuits package from IBM Resilient GitHub.

The following is an example transfer procedure for a Linux system.

```
mkdir ~/package_name_offline
pip download package_name -d "~/some_package_offline"
tar -cvfz package_name_offline.tar package_name_offline
```

Transfer the package to the Resilient integration server then perform the following:

```
tar -xzvf package_name.tar
cd some_directory
pip install package_name-x.x.x-py2.py3-x-x.whl -f ./ --no-index
```

Installing Resilient Circuits

Install and configure the Resilient Circuits framework as follows:

1. Use SSH to access the command line interface on the Resilient integration server.

2. If you do not have an OS user for the service, create one now. To create an OS user called integration on RHEL Linux, use the following command:

   ```
   sudo adduser integration --home /home/integration
   ```

3. Install Resilient Circuits using the following command. This command also installs its dependent modules.

   ```
   pip install --upgrade resilient-circuits
   ```

4. Verify that the Python modules, resilient and resilient-circuits, are installed.

   ```
   pip list
   ```

5. Auto-generate the app.config file as follows. This creates a directory, .resilient, in your home directory with a file in it called app.config, which is the default and preferred option. The Resilient Circuits configurations are maintained in the app.config file.

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

   The output of the command shows the directory where it installed the config file. By default, this directory is:

   ```
   /home/integration/.resilient/app.config
   ```

If you require the configuration file to be in a different location or have a different name, you need to store the full path to the environment variable, APP_CONFIG_FILE.

```
resilient-circuits config -c /path/to/<filename>.config
```

6. Open the app.config file in your text editor. If using nano, the command would be:

```
nano /home/integration/.resilient/app.config
```

7. Replace the contents with the following settings. Your actual path names may be different.

```
[resilient]
host=localhost
port=443
email=resilient_account@example.com
password=ResilientPassword
org=Dev
# componentsdir=/home/resadmin/.resilient/components
logdir=/home/resadmin/.resilient
logfile=app.log
loglevel=INFO
```

For authentication, determine if you are using a Resilient user account or API key account then enter the actual email and password, or api_key_id and api_key_secret, but not both. See Editing the configuration file for a details.

Use the actual Resilient organization name for the org name.

See Editing the configuration file for a detailed description of all the app.config settings, especially cafile if your Resilient platform does not have a valid certificate.

8. Save the file.

9. Test your installation by running the following command:

```
resilient-circuits run
```

Resilient Circuits starts, loads its components, and continues to run until interrupted. If it stops immediately with an error message, check your configuration values and retry.

You are ready to download and deploy extension packages.

Configuring 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 RHEL system, this is done using a systemd unit files to define services. The configuration file defines the following properties:

• OS user account to use.
• Directory from where it should run.
• Any required environment variables.
• Command to run the integrations, such as resilient-circuits run.
• Dependencies.

You may need to change the paths to your working directory and app.config.

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
```

```
[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
```

NOTE: If you are installing Resilient Circuits on the same system as the Resilient platform (not recommended), you need to add the following lines in the [Unit] section after Description:

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

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

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

4. Reload and enable the new service:

```
sudo systemctl daemon-reload
sudo systemctl enable resilient_circuits.service
```

You can use the systemctl command to manually start, stop, restart and return status on the service:

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

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"
```

Chapter 5. Installing on a Windows system

To create a Resilient integration server on a Windows system, install the Resilient Circuits framework which includes its dependent modules.

Choose the procedure that best fits your environment. The online procedure is for a Windows system that has access to the Internet. The offline procedure is for a system that does not.

Installing Resilient Circuits online

Install and configure the Resilient Circuits framework as follows:

1. Install Resilient Circuits on your system as follows. This command may vary slightly depending on your version of Python.

   ```
   python –m pip install resilient-circuits
   ```

2. When the installation is done, use the following command to list the packages that were installed:

   ```
   pip freeze
   ```

   The following is an example of the output:

   ```
   cachetools==2.1.0
   certifi==2019.6.16
   chardet==3.0.4
   circuits==3.2
   entrypoints==0.3
   filelock==3.0.12
   idna==2.8
   Jinja2==2.10.1
   keyring==19.0.2
   MarkupSafe==1.1.1
   PySocks==1.7.0
   pytz==2019.1
   pywin32==224
   pywin32-ctypes==0.2.0
   requests==2.22.0
   requests-mock==1.6.0
   requests-toolbelt==0.9.1
   resilient==32.0.186
   resilient-circuits==32.0.186
   setuptools-scm==3.3.3
   six==1.12.0
   stompest==2.3.0
   urllib3==1.25.3
   ```

3. Create the app.config file:

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

4. Open the app.config file in your text editor and replace the contents with the following settings. Your actual path names may be different.

   ```
   [resilient]
   host=localhost
   port=443
   email=resilient_account@example.com
   password=ResilientPassword
   org=Dev
   # componentsdir=c:\Users\Administrator\.resilient\components
   logdir c:\Users\Administrator\.resilient
   logfile=app.log
   loglevel=INFO
   ```

For authentication, determine if you are using a Resilient user account or API key account then enter the actual email and password, or api_key_id and api_key_secret, but not both. See Editing the configuration file for details.

Use the actual Resilient organization name for the org name.

See Editing the configuration file for a detailed description of all the app.config settings, especially cafile if your Resilient platform does not have a valid certificate.

NOTE: If you edit the file with Notepad, please ensure that you save it as type All Files to avoid a new extension being added to the filename, and use UTF-8 encoding.

5. Start Resilient Circuits:

```
resilient-circuits run
```

6. Install Resilient Circuits service:

```
resilient-circuits.exe service install
```

7. Once installed, you can update the service to start up automatically and run as a user account.
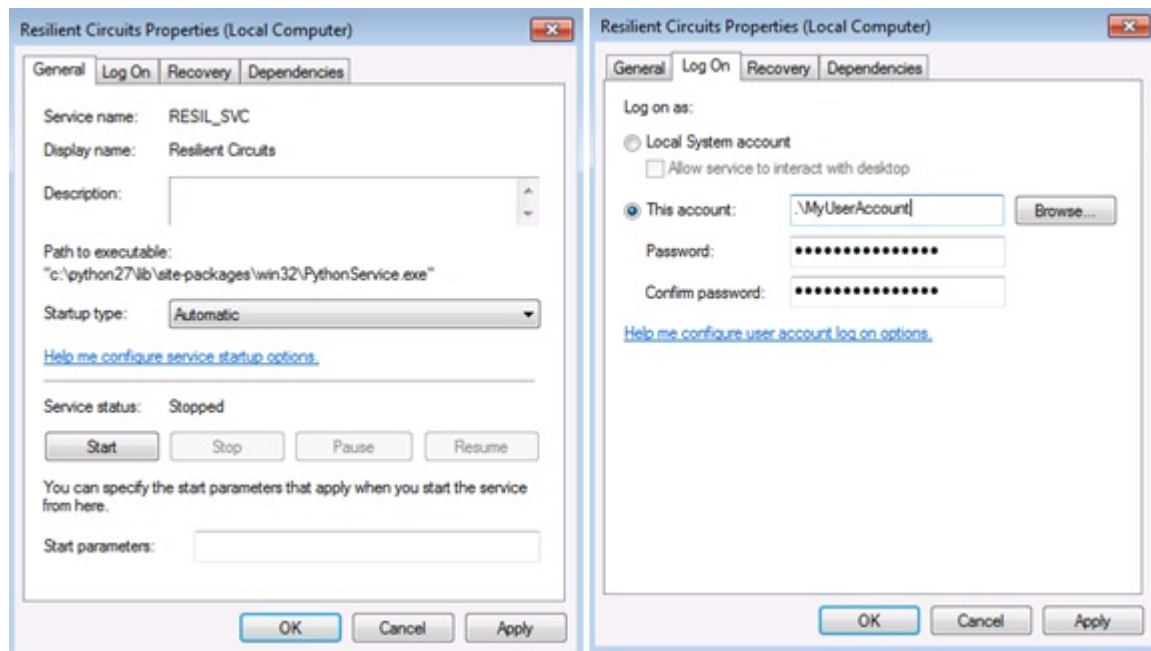
It is recommended that you log in as whichever user account the service will run as to generate the config file and confirm that the extension runs successfully with "resilient-circuits.exe run" before starting the service.

Commands to start, stop, and restart the service are provided as well.

```
resilient-circuits.exe service start
resilient-circuits.exe service stop
resilient-circuits.exe service restart
```

NOTE: To run Resilient Circuits commands on a Windows system, use resilient-circuits.exe. For example, "resilient-circuits.exe run" rather than "resilient-circuits run".

Installing Resilient Circuits offline

Use this procedure only if your Resilient integration server does not have access to the Internet.

NOTE: If you need to manually download the files using a web browser, you can use a combination of the requirements.txt and downloading the packages from PyPi and IBM Resilient GitHub. First, download the Resilient packages from GitHub or PyPi and then reference the requires.txt to find out what you then need to download. Then use PyPi to install the packages with the version specified for the correct operating system.

Perform the following procedure to download the Resilient Circuits package.

1. Make sure the Windows server you use to download Resilient Circuits has Python and the pywin32 library as described in Chapter 3, "Prerequisites," on page 5.

2. If the Windows server has the same operating system and version as the integration server, run the following command to download the required packages.

   ```
   pip download
   ```

   The **pip download** command is detailed in https://pip.pypa.io/en/stable/reference/pip_download/.

3. If the Windows server does not have the same operating system and version as the integration server, use the platform option to enter the platform used by the integration server.

   ```
   pip download --platform <platform>
   ```

4. Create a directory to hold the packages and wheel (whl) files then change into that directory. For example:

   ```
   cd c:\temp_dir
   pip download resilient-circuits -d .
   ```

5. Check the contents of the directory. For example:

   ```
   dir
    Volume in drive C has no label.
    Volume Serial Number is E650-0A06

   Directory of c:\temp_dir

   16/07/2019 17:08 <DIR> .
   16/07/2019 17:08 <DIR> ..
   16/07/2019 17:07 23,000 argparse-1.4.0-py2.py3-none-any.whl
   16/07/2019 17:07 12,047 cachetools-2.1.0-py2.py3-none-any.whl
   16/07/2019 17:07 157,119 certifi-2019.6.16-py2.py3-none-any.whl
   16/07/2019 17:07 133,356 chardet-3.0.4-py2.py3-none-any.whl
   16/07/2019 17:07 174,265 circuits-3.2.tar.gz
   16/07/2019 17:08 11,100 entrypoints-0.3-py2.py3-none-any.whl
   16/07/2019 17:07 7,576 filelock-3.0.12-py3-none-any.whl
   16/07/2019 17:07 58,594 idna-2.8-py2.py3-none-any.whl
   16/07/2019 17:07 124,883 Jinja2-2.10.1-py2.py3-none-any.whl
   16/07/2019 17:08 33,937 keyring-19.0.2-py2.py3-none-any.whl
   16/07/2019 17:07 16,575 MarkupSafe-1.1.1-cp36-cp36m-win_amd64.whl
   16/07/2019 17:07 16,506 PySocks-1.7.0-py3-none-any.whl
   16/07/2019 17:07 510,910 pytz-2019.1-py2.py3-none-any.whl
   16/07/2019 17:08 28,532 pywin32_ctypes-0.2.0-py2.py3-none-any.whl
   16/07/2019 17:07 57,952 requests-2.22.0-py2.py3-none-any.whl
   16/07/2019 17:07 22,916 requests_mock-1.6.0-py2.py3-none-any.whl
   16/07/2019 17:07 54,314 requests_toolbelt-0.9.1-py2.py3-none-any.whl
   16/07/2019 17:07 28,900 resilient-32.0.186.tar.gz
   16/07/2019 17:07 268,161 resilient_circuits-32.0.186.tar.gz
   16/07/2019 17:07 10,586 six-1.12.0-py2.py3-none-any.whl
   16/07/2019 17:07 38,103 stompest-2.3.0.tar.gz
   16/07/2019 17:07 150,942 urllib3-1.25.3-py2.py3-none-any.whl
    22 File(s) 1,940,274 bytes
    2 Dir(s) 20,978,794,496 bytes free
   ```

   In the example, you can see the wheel files listed with the versions. Many of these files are named "none-any.whl" which means they are applicable to all operating systems, but one is "win_amd64.whl," which is version-specific.

6. Download the setuptools. For example:

   ```
   cd c:\temp_dirpip download setuptools -d .
   pip download setuptools_scm -d .
   dir
   ```

```
   Volume in drive C has no label.
   Volume Serial Number is E650-0A06

 Directory of c:\temp_dir

16/07/2019  17:25    <DIR>             .
 16/07/2019  17:25    <DIR>             ..
 16/07/2019  17:24            575,966 setuptools-41.0.1-py2.py3-none-any.whl
 16/07/2019  17:25             23,752 setuptools_scm-3.3.3-py2.py3-none-any.whl
                2 File(s)        599,718 bytes
                2 Dir(s)  20,977,459,200 bytes free
```

7. Copy these files to the Integration Server system.

8. At the integration server, change directory to where the files are installed and run the following
   commands to install the wheel files. Note that there are dependencies so some wheels need to be
   installed before others. For example:

```
pip install setuptools_scm-3.3.3-py2.py3-none-any.whl -f . --no-index
pip install setuptools-41.0.1-py2.py3-none-any.whl -f . --no-index

pip install --no-index -f . stompest-2.3.0.tar.gz
pip install --no-index -f . certifi-2019.6.16-py2.py3-none-any.whl
pip install --no-index -f . urllib3-1.25.3-py2.py3-none-any.whl
pip install --no-index -f . idna-2.8-py2.py3-none-any.whl
pip install --no-index -f . chardet-3.0.4-py2.py3-none-any.whl
pip install --no-index -f . requests-2.22.0-py2.py3-none-any.whl
pip install --no-index -f . circuits-3.2.tar.gz
pip install --no-index -f . pytz-2019.1-py2.py3-none-any.whl
pip install --no-index -f . MarkupSafe-1.1.1-cp36-cp36m-win_amd64.whl
pip install --no-index -f . Jinja2-2.10.1-py2.py3-none-any.whl
pip install --no-index -f . PySocks-1.7.0-py3-none-any.whl
pip install --no-index -f . filelock-3.0.12-py3-none-any.whl
pip install --no-index -f . argparse-1.4.0-py2.py3-none-any.whl
pip install --no-index -f . requests_toolbelt-0.9.1-py2.py3-none-any.whl
pip install --no-index -f . six-1.12.0-py2.py3-none-any.whl
pip install --no-index -f . requests_mock-1.6.0-py2.py3-none-any.whl
pip install --no-index -f . cachetools-2.1.0-py2.py3-none-any.whl
pip install --no-index -f . entrypoints-0.3-py2.py3-none-any.whl
pip install --no-index -f . pywin32_ctypes-0.2.0-py2.py3-none-any.whl
pip install --no-index -f . keyring-19.0.2-py2.py3-none-any.whl
pip install --no-index -f . resilient-32.0.186.tar.gz
pip install --no-index -f . resilient_circuits-32.0.186.tar.gz
```

9. After you have downloaded all the wheels and packages, use the following command to check the
   requires.txt.

```
cd c:\temp_dir
pip download resilient-circuits -d .
```

10. Open any of the tar.gz files, for example, resilient_circuits-32.0.186.tar.gz and change directory to
    resilient_circuits.egg-info. In that directory, open the requires.txt file to verify that you have all the
    required packages. For example:

```
stompest>=2.3.0
requests>=2.6.0
circuits
pytz
jinja2>=2.10.0
pysocks
filelock>=2.0.5
resilient>=29.0
```

If you need to download the required packages, you could use a command such as the following:

```
pip download -r resilient_circuits.egg-info\requires.txt -d c:\temp_dir
```

This command could also be used to install any functions.

Chapter 6. Using encrypted keys (optional)

For security, the Resilient integration server includes a utility to add all of the keyring-based values from your app.config file to your system's compatible keyring backend, such as a KeyStore system. The utility uses the Python Keyring library to store passwords. For information on Python Keyring library, see https://pypi.org/project/keyring/.

To use the utility, see "Resilient keyring utility" on page 15.

To assist in choosing a compatible and secure keyring backend, the following sections in this guide provide recommendations and basic configuration information for these keyring backends:

- Free Desktop Secret Service. Requires the RHEL or CentOS operating system, and Python 3.6.4 or later.
- keyrings.cryptfile.cryptfile.CryptFileKeyring. Requires the RHEL or CentOS operating system, and Python 3.6.4 or later.
- Windows Credential Locker. Requires the Windows operating system, and can run on Python 2.7.9 or 3.6.4, or later.

NOTE: The keyrings.alt backend is the least secure backend and is not recommended for use in production systems. Credentials are stored in a plain text file. keyrings.alt backend does not have a password to access the keyring storage. For information, see https://pypi.org/project/keyrings.alt/.

If using an integration server on a Linux platform, you can use the following command to view any keyring backends installed on your system. For example:

```
keyring --list-backends
keyrings.cryptfile.file.EncryptedKeyring (priority: 0.6)
keyrings.cryptfile.cryptfile.CryptFileKeyring (priority: 2.5)
keyring.backends.chainer.ChainerBackend (priority: 10)
keyrings.cryptfile.file.PlaintextKeyring (priority: 0.5)
keyring.backends.fail.Keyring (priority: 0)
```

In this example, keyrings.cryptfile.cryptfile.CryptFileKeyring is listed as the backend with highest priority (2.5) after ChainerBackend, which determines which backend to invoke.

Resilient keyring utility

The Resilient integration server includes a utility to add all of the keyring-based values from your app.config file to your system's compatible keyring backend, such as a KeyStore system. Once you have created the keys in your app.config file, run res-keyring and you are prompted to create the secure values to store.

```
res-keyring
Configuration file: /Users/kexample/.resilient/app.config
Secrets are stored with 'keyring.backends.SecretService'
[resilient] password: <not set>
Enter new value (or <ENTER> to leave unchanged):
```

You can also use the utility to store passwords or other authentication values used by an extension package.

To pull environmental values into the app.config file, enter ^<*key*>. For example:

```
[resilient]
api_key_secret =^resilient_secret
```

To pull environment values into the app.config file, enter $<*key*>. For example:

```
[resilient]
```

```
api_key_secret =$resilient_secret
```

Free Desktop Secret Service

On a RHEL or Centos platform, the most secure backend is Free Desktop Secret Service run in a Python 3 environment.

Use the following procedure to use the Secret Service backend without an X11 server available.

1. Install the GNOME Keyring daemon:

   ```
   sudo yum install gnome-keyring
   ```

2. Start a D-Bus session:

   ```
   dbus-run-session -- sh
   sh-4.2$ gnome-keyring-daemon --unlock
   Passw0rd
   <CTRL-d>
   SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
   ```

3. At the prompt, start the GNOME keyring daemon.

   ```
   sh-4.2$ gnome-keyring-daemon --unlock
   ```

4. Enter your password for **ResilientKeyringPassword** followed by the enter key then Ctrl-d.

   ```
   ResilientKeyringPassword
   <CTRL-d>
   SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
   ```

   If gnome-keyring-daemon --unlock is run and there is no keyring, the GNOME keyring daemon creates a keyring with the provided password. The GNOME keyring information is stored in the following directory:

   ```
   ~/.local/share/keyrings/
   ```

5. List the keyring backends to verify that Secret Service is installed. For example:

   ```
   sh-4.2$ keyring --list-backends
   keyring.backends.SecretService.Keyring (priority: 5)
   keyring.backends.chainer.ChainerBackend (priority: 10)
   keyring.backends.fail.Keyring (priority: 0)
   ```

   In this example, the Secret Service backend is listed as the backend with highest priority after ChainerBackend which determines which backend to invoke.

6. Once the GNOME keyring is created, run the res-keyring utility to store the app.config credentials in the keystore, as described in "Resilient keyring utility" on page 15.

7. When done, run Resilient Circuits on the command line in the D-bus session, which allows the integration server to access the credentials from the GNOME keyring.

   The GNOME keyring is unlocked while in the D-bus session, so you are not prompted for a keyring password when executing "resilient-circuits run" on the command line.

Once you have setup the GNOME keyring using res-keyring on the command line, you can run the integration server as a system service. However, you need to edit the service to make sure that the system can pass the keyring password to the systemd service for unlocking.

1. Edit service file /etc/systemd/system/resilient_circuits.service as follows to use the Secret Service backend:

   ```
   Description=Resilient-Circuits Service
   [Service]
   Type=simple
   User=integrations
   ```

```
WorkingDirectory=/home/integrations/resilient-circuits-py3
Environment=RES_KEYRING_PASSWORD=ResilientKeyringPassword
ExecStart=/usr/bin/res-circuits.sh
Restart=always
TimeoutSec=10
Environment=APP_CONFIG_FILE=/home/integrations/.resilient/app.config
Environment=APP_LOCK_FILE=/home/integrations/.resilient/resilient_circuits.lock

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

NOTE: The /etc/systemd/system/resilient_circuits.service file is owned by root. Only those users with privileges can view the password in the file.

2. Add the environment variable containing the GNOME keyring password:

```
Environment=RES_KEYRING_PASSWORD=ResilientKeyringPassword
```

3. Look for the ExecStart line:

```
ExecStart=/usr/local/bin/resilient-circuits run
```

4. Edit it to start Resilient Circuits from a shell script:

```
ExecStart=/usr/bin/resilient-circuits.sh
```

The following resilient-circuits.sh script starts a dbus session and calls another script to unlock the keyring and start Resilient Circuits:

```
#!/bin/bash
dbus-run-session -- "/usr/bin/resilient-circuits-unlock-keyring.sh"
```

The /usr/bin/resilient-circuits-unlock-keyring.sh is as follows:

```
#!/bin/bash
echo $RES_KEYRING_PASSWORD | gnome-keyring-daemon --unlock
resilient-circuits run
```

Keyrings cryptfile

The keyrings.cryptfile backend is a project in PyPi that is mainly targeted on a sufficiently secure storage for plain text passwords (keyring) in a simple portable file. For more information, see https://pypi.org/project/keyrings.cryptfile/.

Use the following procedure to use the keyrings.cryptfile backend:

1. Install the keyrings.cryptfile backend.

```
pip install keyrings.cryptfile
```

2. List the keyring backends to verify that keyrings.cryptfile is installed. For example:

```
keyring --list-backends
keyrings.cryptfile.file.EncryptedKeyring (priority: 0.6)
keyrings.cryptfile.cryptfile.CryptFileKeyring (priority: 2.5)
keyring.backends.chainer.ChainerBackend (priority: 10)
keyrings.cryptfile.file.PlaintextKeyring (priority: 0.5)
keyring.backends.fail.Keyring (priority: 0)
```

3. Run the res-keyring utility to store the app.config credentials in the keystore, as described in "Resilient keyring utility" on page 15.

The newly created cryptfile keyring is stored in the following location:

```
~/.local/share/python_keyring/cryptfile_pass.cfg
```

When you run resilient-circuits on the command line, you are prompted for the keyrings.cryptfile keyring password:

```
$ resilient-circuits run
Please enter password for encrypted keyring:
```

After you enter the keyrings.cryptfile password, Resilient Circuits accesses the credentials from the keyrings.cryptfile keystore.

Once you have setup the keyring using res-keyring on the command line, you can run the integration server as a system service. However, you need to edit the service to make sure that the system can pass the keyring password to the systemd service for unlocking.

1. Edit service file /etc/systemd/system/resilient_circuits.service as follows to use the keyrings.cryptfile backend:

```
Description=Resilient-Circuits Service
[Service]
Type=simple
User=integrations
WorkingDirectory=/home/integrations/resilient-circuits-py3
Environment=RES_KEYRING_PASSWORD=ResilientKeyringPassword
ExecStart=/usr/bin/res-circuits.sh
Restart=always
TimeoutSec=10
Environment=APP_CONFIG_FILE=/home/integrations/.resilient/app.config
Environment=APP_LOCK_FILE=/home/integrations/.resilient/resilient_circuits.lock

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

NOTE: The /etc/systemd/system/resilient_circuits.service file is owned by root. Only those users with privileges can view the password in the file.

2. Add the environment variable containing the keyring password:

```
Environment=RES_KEYRING_PASSWORD=ResilientKeyringPassword
```

3. Look for the ExecStart line:

```
ExecStart=/usr/local/bin/resilient-circuits run
```

4. Edit it to start Resilient Circuits from a shell script:

```
ExecStart=/usr/bin/res-circuits.sh
```

The /usr/bin/res-circuits-unlock-keyring.sh is as follows:

```
#!/bin/bash
echo $RES_KEYRING_PASSWORD | resilient-circuits run
```

Windows Credential Locker

When running the integration server on Windows, the Windows Credential Locker backend is the recommended most secure backend. You do not need to pip install the Windows Credential Locker backend, it is there by default when Python Keyring is installed with Resilient Circuits.

For more information on the Windows Credential Locker, see:

```
https://docs.microsoft.com/en-us/windows/uwp/security/credential-locker
```

After installing Resilient Circuits, you can check that the Windows Credential Locker is in use:

```
keyring --list-backends
keyring.backends.chainer.ChainerBackend (priority: 0)
keyring.backends.Windows.WinVaultKeyring (priority: 5)
```

```
keyring.backends.fail.Keyring (priority: 0)
```

Then run res-keyring utility to store the app.config credentials in the Windows keystore:

```
res-keyring
Configuration file: C:\Users\integrations\.resilient\app.config
Secrets are stored with 'keyring.backends.Windows'
Please enter password for encrypted keyring:
[resilient] password: ^res_password
  Enter new value (or <ENTER> to leave unchanged): Passw0rd
  Confirm new value: Passw0rd
Value set.
Done.
```

To run the integration server as a service on Windows, see Chapter 5, "Installing on a Windows system," on page 11. Note that the user account that is used to run Resilient Circuits as a service should be the same account from which res-keyring utility is executed to store the credentials.

Chapter 7. Configuration

Once installed, you need to edit the configuration file, which defines essential configuration settings for all Resilient Circuits components running on the system. There is a [resilient] section in the configuration file that controls how the core Resilient Circuits and Resilient packages access the Resilient platform.

As you install extensions on the Resilient integration server, each extension creates its own section within the configuration file, as described later in this guide.

NOTE: If the APP_CONFIG_FILE environment variable is not set, Resilient Circuits looks for a file called "app.config" in the local directory where the run command is launched.

Editing the configuration file

Open the configuration file in the text-editor of your choice and update the [resilient] section with your Resilient platform host name/IP, credentials and the absolute path to the logs directory you created. The following table describes all the required and optional values that can be included in this section.

NOTE: If on a Windows system and you edit the file with Notepad, please ensure that you save it as type **All Files** to avoid a new extension being added to the filename, and use UTF-8 encoding.

Parameter	Description
logfile	Name of rotating logfile that is written to logdir. Default is app.log.
logdir	Path to directory to write log files. If not specified, program checks environment variable DEFAULT_LOG_DIR for path. If that is not set, then defaults to a directory called "log" located wherever Resilient Circuits is launched.
log_level	Level of log messages written to stdout and the logfile. Levels are: CRITICAL, ERROR, WARN, INFO (default), and DEBUG.
host	Required. IP or hostname for the Resilient platform.
org	Name of the Resilient organization. Required only if the user account is used with more than one Resilient organization.
email	Resilient user account for authenticating to the Resilient platform. It is recommended that this account is dedicated to extensions. This is required unless using a Resilient API key account.
password	Password for the Resilient user account.
api_key_id	Resilient API key account for authenticating to the Resilient platform. It is only available with V33 or later of the Resilient platform and Resilient Circuits. The ID is a long string, which is provided by the Resilient system administrator. This is required unless using a Resilient user account. Not valid for integration servers connected to a Resilient platform with the MSSP add-on.
api_key_secret	Secret for the Resilient API key account. It is also provided by the Resilient system administrator. The secret must be entered in app.config.

Parameter	Description
no_prompt_password	If set to False (default) and the "password" value is missing from this config file, the user is prompted for a password. If set to True, the user is not prompted.
stomp_port	Port number for STOMP. Default is 65001.
componentsdir	Path to directory containing additional Python modules. Resilient Circuits can load custom components from this directory. Typically this option is only used by extension developers.
noload	Optional. Comma-separated list of: • Installed components that should not be loaded. • Module names in the componentsdir that should not be loaded. Example: my_module, my_other_module, InstalledComponentX
proxy_host	IP or Host for Proxy to use for STOMP connection. By default, no proxy is used.
proxy_port	Port number for Proxy to use for STOMP connection. By default, no proxy is used.
proxy_user	Username for authentication to Proxy to use for STOMP connection. If a proxy_host is specified and no proxy_user specified, then assumed no authentication is required.
proxy_password	Password for authentication to Proxy to use for STOMP connection. Used in conjunction with proxy_user.
cafile	Path and file name of the PEM file to use as the list of trusted Certificate Authorities for SSL verification when the Resilient platform is using untrusted self-signed certificates. If not using a trusted certificate, **cafile** must be set to False. If there is a PEM file, use a second instance of **cafile** to set to True or False. If set to False, certificate verification is not performed and the PEM file is used. If set to True (default), allow only trusted certs.

NOTE: If you enter values for the Resilient user account fields and the API Key account fields, the API key account is used by default.

Monitoring the config file for changes

You can configure Resilient Circuits to monitor the app.config file for changes. When it detects a change has been saved, it updates its connection to the Resilient platform and notifies all components of the change. To enable this option, install the "watchdog" package.

```
pip install watchdog
```

Now you can run with:

```
resilient-circuits run –r
```

Without the –r option, changes to the app.config file have no impact on a running instance of Resilient Circuits. Note that not all components currently handle the reload event and may continue using the previous configuration until Resilient Circuits is restarted.

Configuring servers in an MSSP deployment

When the Resilient platform has the MSSP add-on, there is a single configuration organization and multiple child organizations. You deploy all extension packages to the configuration organization. An administrator then pushes to all the child organizations the extensions and any additional Resilient components, such as functions, rules, workflows, custom fields, data tables, any layout changes needed for the integrations.

You need to configure one integration server for the configuration organization and one integration server for each child organization, as follows:

- Install and configure one integration server to connect to the Resilient configuration organization. This is the integration server that you use to deploy every extension package used by any child organization. Specifically, this is the only integration server where you will use the Resilient Circuits customize command.
- Install and configure one integration server for each child organization. If you have five child organization then you need five integration servers, assuming that each child organization is running an extension. This includes configuring the integration server's app.config file to point to the correct Resilient organization and its dedicated Resilient account.
- Install an extension package on the integration server for the configuration organization.
- Install an extension package on each integration server whose child organization will be running that extension.

Once you deploy an extension from the integration server to the configuration organization, the Resilient system administrator pushes the integration to all child organizations. You then use the integration server for each child organization to run and test the extension.

NOTE: If you deploy an extension to a child organization or change any Resilient component, such as a custom field, rule, workflow or function, your changes will be lost the next time the administrator performs a configuration push. Only make changes to the components in the configuration organization.

Updating your environment

When Resilient Circuits is updated, you can make sure you have the latest tools and Resilient Circuits framework by entering the following commands:

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

Configuring multiple circuits (optional)

By default, you have a single circuit framework where every Resilient extension package you install uses the same configuration file.

You may need to run multiple circuits for your environment. Reasons could include:

- Isolation. You may wish to keep one or more extensions separate for troubleshooting and logging purposes. With a single circuit, all Resilient extensions reference the same app.config file and log to a single log file.
- Ability to set up development and production environments on the same integration server.
- Performance. Upon startup, Resilient Circuits loads each extension one by one. It would be quicker to have each extension loaded by its own circuit, more importantly on its own system process.
- MSSP. If you have a Resilient platform with the MSSP add-on, you may wish to have a single integration server that can run extensions on multiple organizations.

A common method to create multiple circuits on a single integration server is to use virtualenv to create isolated Python environments. Each virtual environment includes its own Python runtime, pip, and libraries, including the Resilient and Resilient Circuits libraries. Also, each environment needs to specify an alternate location for the "resilient_circuits_lockfile" file in the "APP_LOCK_FILE" environment variable to prevent multiple copies of the application from running at once. The lock file config option cannot be specified in the app.config file.

The following procedure is an example of using virtualenv to create isolated Python environments.

1. Install virtualevn:

   ```
   pip install virtualenv
   ```

2. Create a development environment:

   ```
   mkdir ~/dev
   ```

3. Create a virtualenv project:

   ```
   cd ~/dev
   virtualenv dev
   cd bin
   source ./activate
   ```

4. Install all the python libraries. To install Resilient Circuits:

   ```
   pip install resilient-circuits
   ```

5. Add the following lines into the app.py to set the lock file variable. In the following example, it is located at ~/dev/dev/lib/python2.7/site-packages/resilient_circuits/app.py.

   ```
   os.environ["APP_CONFIG_FILE"] = "/root/.resilient/app.config.dev"
   os.environ["APP_LOCK_FILE"] = "/root/.resilient/resilient_circuits_lockfile_dev"
   os.environ["APP_LOG_DIR"] = "/root/.resilient/app.log.dev"

   APP_LOG_DIR = os.environ.get("APP_LOG_DIR", "logs")

   application = None
   logging_initialized = False

   class AppArgumentParser(keyring_arguments.ArgumentParser):
   ```

6. Optionally, you can specify the log file location and app config file location.
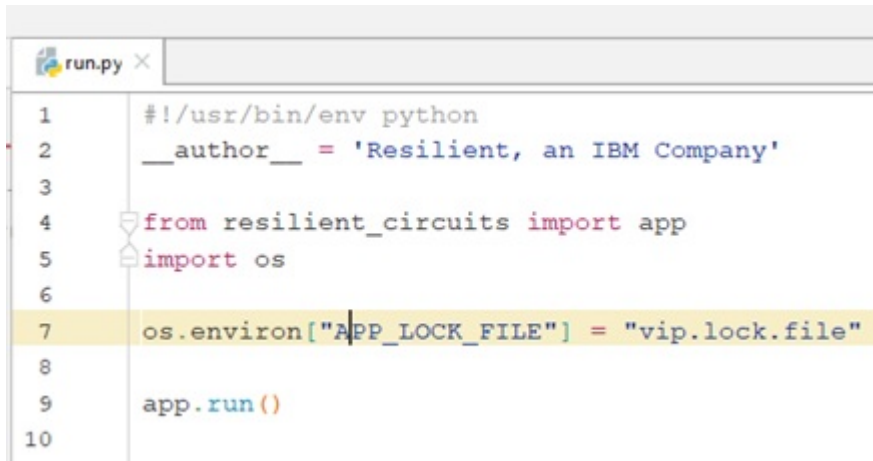
Using the previous example, you would run Resilient Circuits from: ~/dev/dev/lib/python2.7. And the pip you use is located at ~/dev/dev/bin/pip.

To run Resilient Circuits, you add the path to the configuration file. For example:

```
resilient-circuits config -c /path/to/<filename>.config
```

Alternately, instead of modifying app.py to configure the lock file as described in step 5, you can perform the following:

1. Create a file called run.py in each virtualenv folder. For example:

```python
#!/usr/bin/env python
__author__ = 'Resilient, an IBM Company'

from resilient_circuits import app
import os

os.environ["APP_LOCK_FILE"] = "vip.lock.file"

app.run()
```

2. Specify the lock file in the script. You can also specify the log_file or app config file in the script; otherwise, the run.py uses any file named run.py from the same folder of the run.py script, and generates a log file in the same folder as well.

3. Instead of using the resilient-circuits run command, you start integration by invoking "python run.py". The python has to be the python runtime from your virtualenv.

Chapter 8. Extension packages

Once Resilient Circuits is installed and running, you can install extension packages on your server.

Whenever you install a new package, it creates a new section in the app.config file. You need to update that section in accordance with the instructions that came with the extension package. Afterwards, you can deploy it to your Resilient platform for customization and testing. Also see the document provided with the extension for details on how to customize and use the extension once deployed.

IMPORTANT: If you are using the Resilient platform with the MSSP add-on, you must install the extension package on the integration server that deploys to configuration organization, and on each integration server that is connected to a child organization that runs that extension.

Downloading and installing a package

You can download functions and other extension packages from the IBM Resilient Community or IBM X-Force App Exchange.

Once downloaded, perform the following to install and configure the package on your Resilient integration server. Typically, the packages are in a tar.gz format.

NOTE: If you are not using the integration user account, you need to run the commands using sudo.

1. Use SSH to connect to your Resilient integration server.
2. Go to the folder where the package is located.
3. Unzip the file.
4. Install the package using the following command:

   ```
   pip install -–user <package_name>-x.x.x.tar.gz
   ```

   The --user ensures that the libraries are installed in the home directory of the user you are logged in as (integration) and avoids conflicts with other Python libraries in the system directory.

5. Verify that the component is installed using the following command.

   ```
   resilient-circuits list
   ```

6. After installing the package, run the following command to update app.config. This command adds a new section with default values in the app.config file.

   ```
   resilient-circuits config –u
   ```

   Alternately, you can specify which packages to update app.config.

   ```
   resilient-circuits config –u –l <package1> <package2>
   ```

   If using an alternate file location for your app.config file, you need to specify it when you update.

   ```
   resilient-circuits config –u /path/to/app.config
   ```

7. Follow the instructions in the component's documentation file to edit the app.config file. Depending on the requirements of the extension, you may need to modify the default values to fit your environment, such as credentials to a 3rd party system.

Upgrading a package

Use the same procedure as installing a package, except use the –U option in the install command:

```
pip install –U -–user <package_name>-x.x.x.tar.gz
```

Deploying and testing

Use the following command to deploy the components in a package to the Resilient platform:

```
resilient-circuits customize
```

Alternatively, you can install as specific package as follows:

```
resilient-circuits customize –l <package_name>
```

You are prompted deploy the components, such as functions, message destinations, workflows and rules. You can use the optional parameter, -y, in the command line to not be prompted.

Optionally, if your package has multiple functions, you can specify which functions to deploy as follows.

```
resilient-circuits customize –l <function_name1> <function_name2>
```

IMPORTANT: If you are using the Resilient platform with the MSSP add-on, you must use the integration server that deploys to the configuration organization when deploying using the Resilient Circuits customize command. The system administrator then pushes to integration to the appropriate child organizations. You then use the integration server connected to the child organization to run and test the integration. If more than one child organization uses the integration, you must test each one individually using the organization's integration server.

Chapter 9. Troubleshooting

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

- 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.

Chapter 10. Support

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

For additional support, contact support@resilientsystems.com.