IBM Resilient

Incident Response Platform Integrations

AWS Utilities Functions V1.0.0

Release Date: July 2018

Resilient Functions simplify development of integrations by wrapping each activity into an individual workflow component. These components can be easily installed, then used and combined in Resilient workflows. The Resilient platform sends data to the function component that performs an activity then returns the results to the workflow. The results can be acted upon by scripts, rules, and workflow decision points to dynamically orchestrate the security incident response activities.

This guide describes the AWS Utilities Functions.

Overview

The AWS Utilities Functions integration package contains several useful workflow functions for common AWS services such as AWS SNS and AWS Lambda.

This document describes each utility function, how to configure it in custom workflows, and any additional customization options.

Installation

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

- Resilient platform is version 30 or later.
- You have a Resilient account to use for the integrations. This can be any account that has the permission to view and modify administrator and customization settings, and read and update incidents. You need to know the account username and password.
- You have access to the command line of the Resilient appliance, which hosts the Resilient platform; or to a separate integration server where you will deploy and run the functions code. If using a separate integration server, you must install Python version 2.7.10 or later, or version 3.6 or later, and “pip”. (The Resilient appliance is preconfigured with a suitable version of Python.)

Install the Python components

The functions package contains Python components that are called by the Resilient platform to execute the functions during your workflows. These components run in the Resilient Circuits integration framework.

The package also includes Resilient customizations that will be imported into the platform later.

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. To use the AWS Utilities function package, you must install a 3rd party package called boto3:

```
sudo pip install boto3
```

3. Run the following command to install the package:

```
sudo pip install --upgrade fn_aws_utilities-1.0.0.zip
```

Configure the Python components

The Resilient Circuits components run 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
```

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 [fn_<*fn_name*>] section, edit the settings as follows:

```
<settings>
```

Deploy customizations to the Resilient platform

The package contains function definitions that you can use in workflows, and includes example workflows and rules that show how to use these functions.

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

```
resilient-circuits customize
```

2. Respond to the prompts to deploy functions, message destinations, workflows and rules.

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.

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 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
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"
```

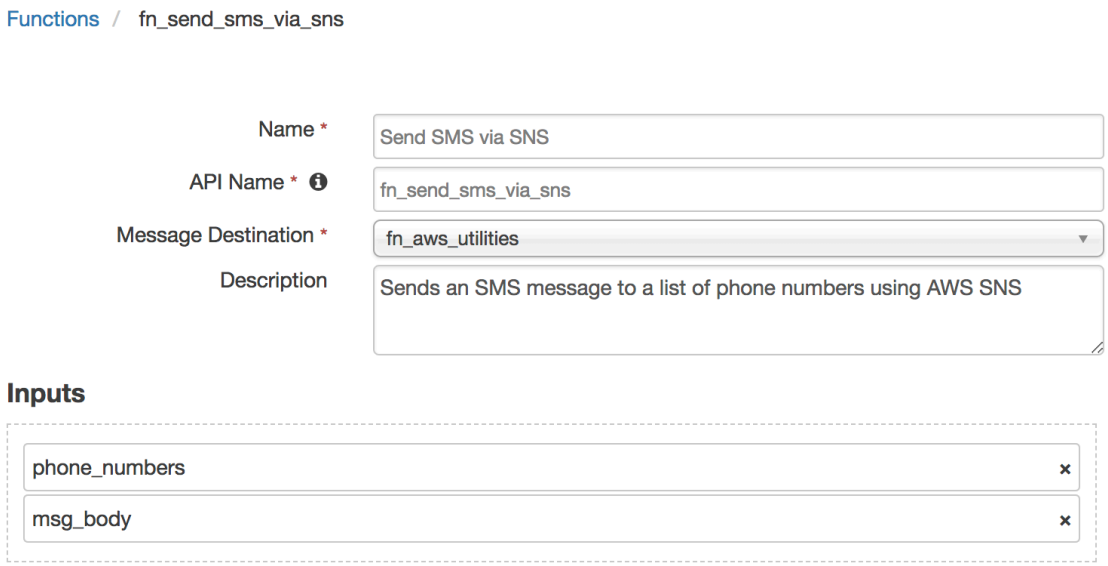
Function Descriptions

Once the function package deploys the function(s), you can view them in the Resilient platform Functions tab, as shown below. The package also includes example workflows and rules that show how the functions can be used. You can copy and modify these workflows and rules for your own needs.

Example: Invoke AWS Lambda: NodeJS Addition	Adds two numbers by executing an AWS Lambda function created in NodeJS	Incident	Example: Invoke AWS Lambda: NodeJS Addition	
Example: Invoke AWS Lambda: Python Addition	Adds two numbers by executing an AWS Lambda function created in Python	Incident	Example: Invoke AWS Lambda: Python Addition	
Example: Send SMS: Incident	Sends an SMS to members of an incident using AWS SNS	Incident	Example: Send SMS: Incident	
Example: Send SMS: Task	Sends a text message to members of a task indicating it has been created	Task	Example: Send SMS: Task	

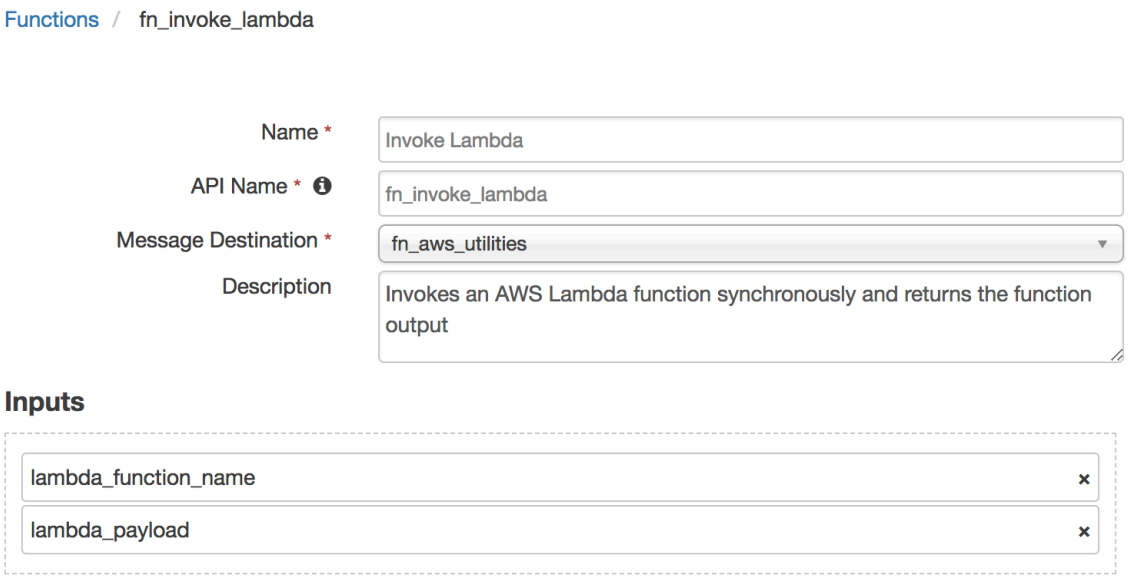
fn_send_sms_via_sns: Send SMS via SNS

This function sends a message to a list of phone numbers. Provide a list of phone_numbers as a string in csv format (i.e “19998887777, 16665554444”) and a msg_body to send to the phone numbers and the function will utilize the AWS SNS service to send a text message to each number. The function does not provide an output.



fn_invoke_lambda: Invoke Lambda

This function invokes an AWS Lambda function by it’s function name. Provide a function name and a payload to provide the function, and the fn_invoke_lambda function will return the response from the function in the “response_payload” attribute of the response object.



To utilize the two provided example workflows for this function, “Example: Invoke AWS Lambda: NodeJS Addition” and “Example: Invoke AWS Lambda: Python Addition”, you need to create two Lambda functions in your AWS function called “resilient_example_nodejs_addition” and “resilient_example_python_addition”.

For the “resilient_example_nodejs_addition” AWS Lambda function, NodeJS 6.10 or NodeJS 8.10 will work. The function code follows:

```
exports.handler = async (event) => {
    return event.x + event.y;
};
```

For the “resilient_example_python_addition” AWS Lambda function, Python 2.7 or Python 3.6 will work. The function code follows:

```
def lambda_handler(event, context):
    return event.get("x") + event.get("y")
```

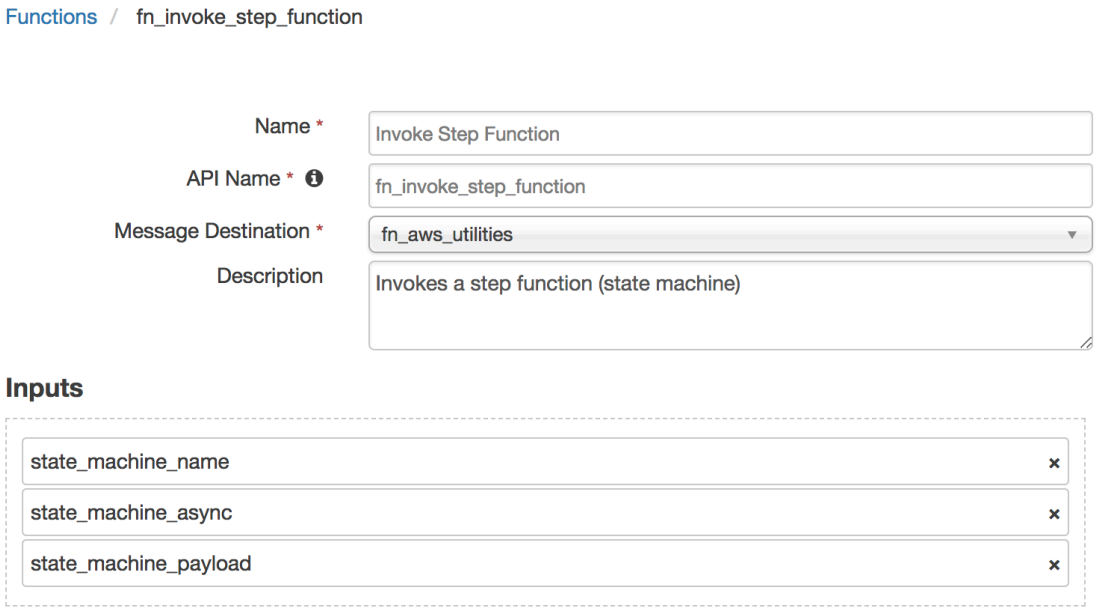
These AWS Lambda functions simply return the sum of the json input x and y.

fn_invoke_step_function: Invoke Step Function

This function invokes an AWS Step Function by it’s state machine name. Provide a state machine name and a payload to provide the function, and the fn_invoke_step_function will return the response from the function in the “output” attribute of the response object. An optional argument, “state_machine_async” exists to specify whether or not to run the function asynchronously. If the function is ran asynchronously, the output attribute may not exist, and you must use fn_get_step_function_execution to get an output. An fn_invoke_step_function result may look like this:

```
{
  'executionArn': 'string',
  'stateMachineArn': 'string',
  'name': 'string',
  'status': 'RUNNING'|'SUCCEEDED'|'FAILED'|'TIMED_OUT'|'ABORTED',
  'startDate': datetime(2015, 1, 1),
  'stopDate': datetime(2015, 1, 1),
```

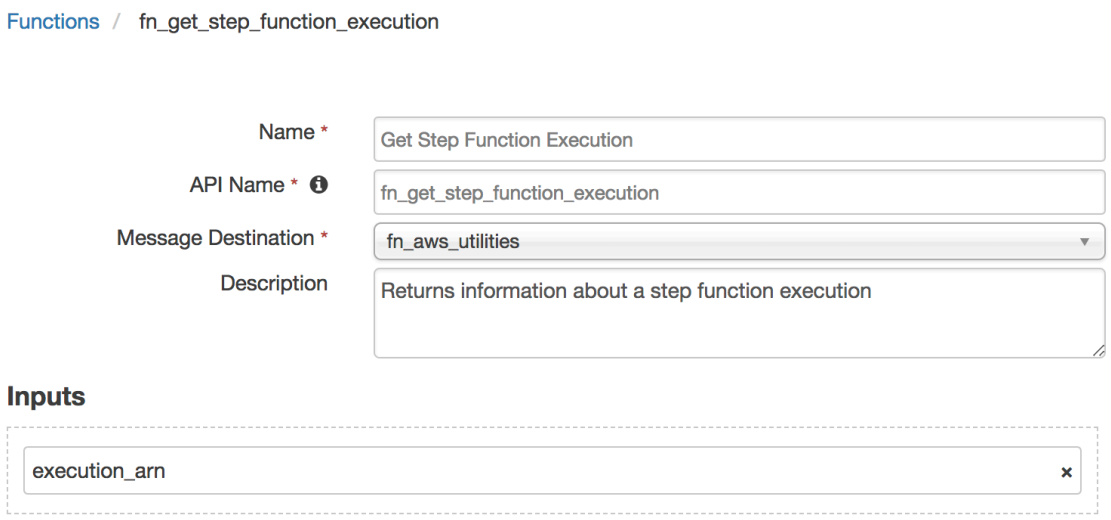
```
  'input': 'string',
  'output': 'string'
}
```



fn_get_step_function_execution: Get Step Function Execution

This function returns an output similar to that of fn_invoke_step_function. Provide an execution arn to the function, and it will return the execution state, mirroring the result of fn_invoke_step_function. An fn_get_step_function_execution result may look like this:

```
{
  'executionArn': 'string',
  'stateMachineArn': 'string',
  'name': 'string',
  'status': 'RUNNING'|'SUCCEEDED'|'FAILED'|'TIMED_OUT'|'ABORTED',
  'startDate': datetime(2015, 1, 1),
  'stopDate': datetime(2015, 1, 1),
  'input': 'string',
  'output': 'string'
}
```



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.

Support

For additional support, contact support@resilientsystems.com.

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