IBM Resilient

Incident Response Platform Integrations
Cloud Foundry Function V1.0.0
Release Date: August 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 Platform 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 Cloud Foundry Function.

Overview

This function interfaces with Cloud Foundry platform to allow the user to manage deployed applications, their instances, and deploy new applications. Managing the applications includes starting/stopping, updating, restaging, deleting, and getting various types of information about it. This package implements these actions in 3 functions, 3 example workflows and 3 example rules.

This package is a wrapper around Cloud Foundry's API with possibilities to be adjusted for different platform providers. On the time of release the latest stable version is: https://apidocs.cloudfoundry.org/3.1.0/.

Installation

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

- Resilient platform is version 30 or later.

- You have the necessary credentials to access your provider's Cloud Foundry API

- 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. Run the following command to install the package:

```
sudo pip install --upgrade fn_cloud_foundry-1.0.0.tar.gz
```

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_cloud_foundry] section, edit the settings as follows:

```
[fn_cloud_foundry]

#Base url endpoint of your CF platform

#For example, for IBM's BlueMix it is: https://api.ng.bluemix.net/

cf_api_base=https://api.ng.bluemix.net/

#Enter only what's required by your authenticator.

#For example, the default BlueMixCF authenticator only requires apikey.

cf_api_apikey=

#Enter username and password if needed for access to DockerHub for Create
Application function

cf_api_username=

cf_api_password=
```

4. Current package is configured by default to work with BlueMix Cloud Foundry platform, but it can be connected to any platform that exposes API.

a. Create a new **Authenticator** class, that inherits from **AuthenticatorBase** located in utils/authentication. It needs to implement 2 methods, auth – to authenticate, and get_headers – to return headers that need to be added to the http requests in order for it to be authenticated as a dictionary.

b. In the code for the functions, the default authenticator will need to be replaced by the one you've created.

c. Build the package with the new code, by running **python setup.py sdist**, and install the newly built package.

Deploy customizations to the Resilient platform

The package contains 3 functions that can be used in workflows, as well as 3 example workflows and 3 rules that demonstrate how to call the 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 <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 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 Cloud Foundry package is installed, you can view them in the Resilient platform Functions tab, as shown below. This package also includes 3 example workflows to show how to use the functions and 3 rules to invoke the workflows. You can edit and modify the workflows and rules to suit your needs.

Name	Description	
Cloud Foundry: Application Command	Performs a specified action on the chosen Cloud Foundry applications. Currently available actions: start, stop, restage, info, delete, and instances.	🗑
Cloud Foundry: Create an App	Create and deploy a cloud foundry application from the specified parameters/docker files. additional_parameters accept any additional parameters that CF API accepts in JSON format.	🗑
Cloud Foundry: Instance Command	Performs an action on the specified instances of an application. Currently available: delete. Multiple instances can be provided, but only one application name should be given.	🗑

Function fn_cloud_foundry_manage_applications: Cloud Foundry: Application Command

Functions / fn_cloud_foundry_manage_applications

Name *	Cloud Foundry: Application Command
API Name * ⓘ	fn_cloud_foundry_manage_applications
Message Destination *	cloud_foundry ▾
Description	Performs a specified action on the chosen Cloud Foundry applications. Currently available actions: start, stop, restage, info, delete, and instances.

Inputs

fn_cloud_foundry_applications   ✕

fn_cloud_foundry_action   ✕

fn_cloud_foundry_additional_parameters_json   ✕

This function performs one of the chosen actions on the specified Cloud Foundry applications.

It takes in 3 inputs:

- fn_cloud_foundry_applications: **Required** – name or comma-separated names of the application(s) that the action should be applied to.

- fn_cloud_foundry_action: **Required** – select field with the list of all possible actions to be performed on the application. They include start, stop, restage, delete, instances, and info.

- fn_cloud_foundry_additional_parameters_json: **Optional** – here for the purposes of allowing the actions to be added and extended in the future. Accepts valid JSON, which will be passed to the method executing chosen actions. See the Cloud Foundry API documentation for possible JSON parameters. https://apidocs.cloudfoundry.org/

Listed below, is the format of the results of the function. The results indicate whether the action was successfully applied to each of the applications specified, as well as other data relevant to the action itself, per application.

```
{
    "application-1": {
        "success":true,
        "other_data":  "...",
        "_keys": ["success","other_data"]
    },
    "application-2": {
        "success":false,
        "details ": "...",
        "_keys": ["success","details"]
    },
    "_keys": ["application-1", "application-2"]
}
OR, if the action is incorrect:
{
  "success":false,
  "details":  "...",
  "_keys": ["success","details"]
}
```
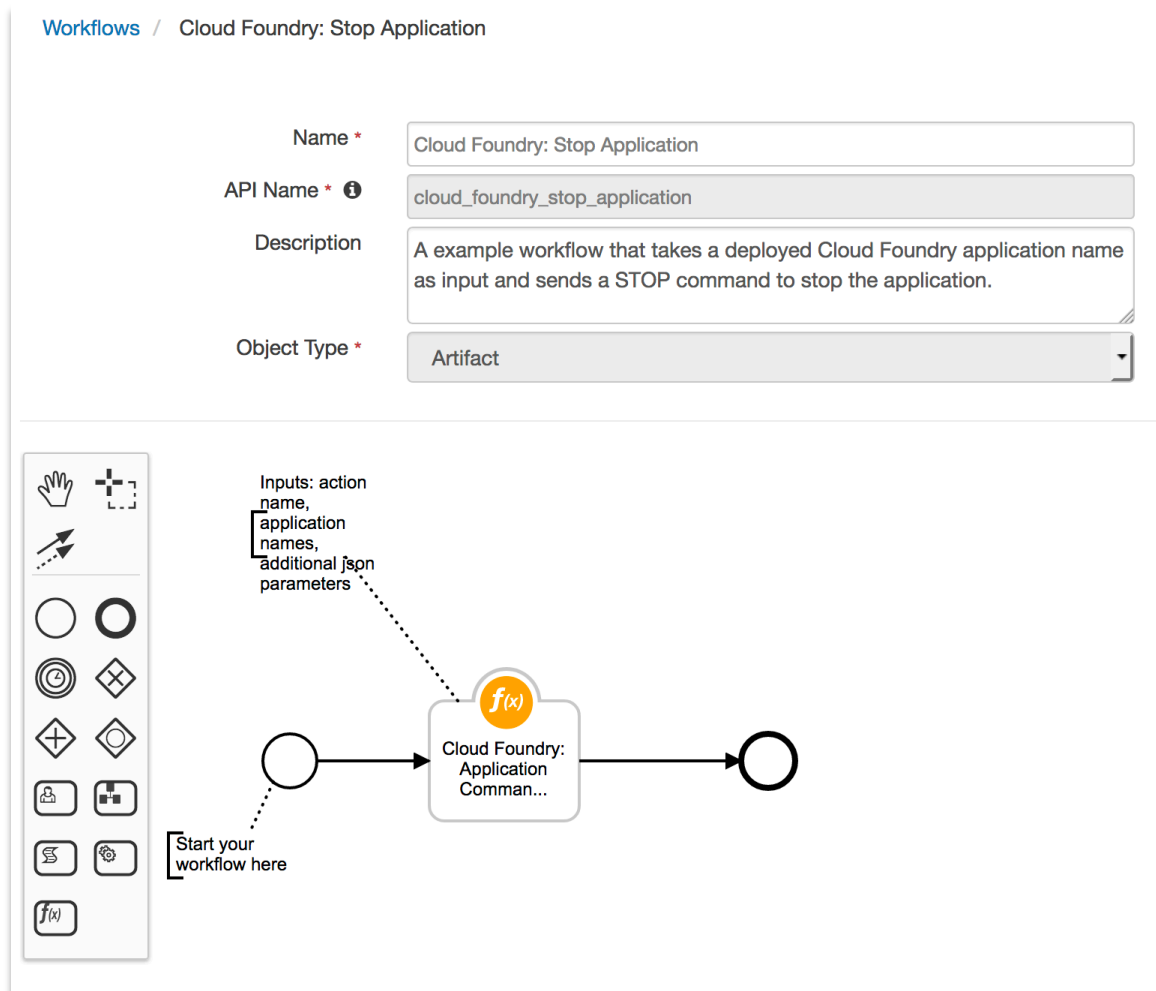
"_keys" store the list of keys for each level of a dictionary to provide convenience in post-processing.

Workflow: Cloud Foundry: Stop Application

Below is a screen shot of an example workflow that calls the fn_cloud_foundry_manage_applications function. The Cloud Foundry: Stop Application workflow specifies the STOP action command be executed on the deployed application.

Workflows / Cloud Foundry: Stop Application

Name *   Cloud Foundry: Stop Application

API Name *   ⓘ   cloud_foundry_stop_application

Description   A example workflow that takes a deployed Cloud Foundry application name as input and sends a STOP command to stop the application.

Object Type *   Artifact

Inputs: action name, application names, additional json parameters

Cloud Foundry: Application Comman...

Start your workflow here

The Cloud Foundry: Stop Application workflow pre-processor script below gets the deployed application name from the artifact value. In this example, a Service type artifact was created in Resilient with the artifact value set to the name of the Cloud Foundry deployed application.
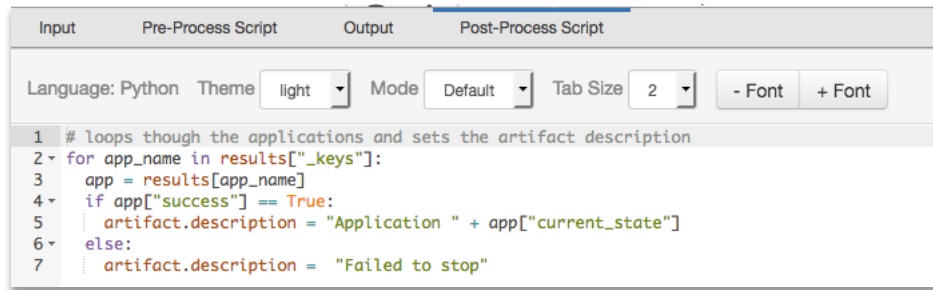
Input	Pre-Process Script	Output	Post-Process Script

Language: Python   Theme [light ▾]   Mode [Default ▾]   Tab Size [2 ▾]   [- Font] [+ Font]

```
1  inputs.fn_cloud_foundry_applications=artifact.value
```
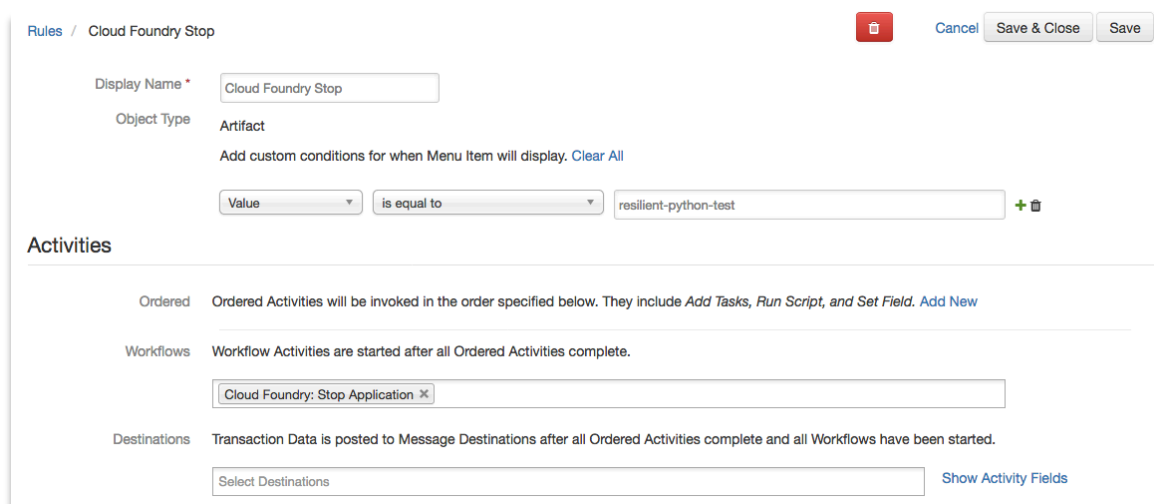
The Cloud Foundry: Stop Application workflow post-processor script below loops through the applications and sets the artifact description to indicate the result of the workflow: whether the application was successfully stopped or not.

| Input | Pre-Process Script | Output | Post-Process Script |

Language: Python Theme [light ▾] Mode [Default ▾] Tab Size [2 ▾] [- Font] [+ Font]

```
1  # loops though the applications and sets the artifact description
2▾ for app_name in results["_keys"]:
3    app = results[app_name]
4▾   if app["success"] == True:
5      artifact.description = "Application " + app["current_state"]
6▾   else:
7      artifact.description =  "Failed to stop"
```

Rule: Cloud Foundry Stop

The following screen shot shows the example Cloud Foundry Stop rule menu item which will appear in the Action Menu of an artifact when it's value is set to "resilient-python-test" (the Cloud Foundry deployed application). When the rule menu item is activated, the Cloud Foundry Stop Application workflow is invoked on the deployed "resilient-python-test" application. Modify or create your own rules to act on your deployed application.

Rules / Cloud Foundry Stop   [🗑]   Cancel [Save & Close] [Save]

Display Name *   [Cloud Foundry Stop   ]
Object Type   Artifact
                  Add custom conditions for when Menu Item will display. Clear All

                  [Value   ▾] [is equal to   ▾] [resilient-python-test   ] + 🗑

Activities

Ordered   Ordered Activities will be invoked in the order specified below. They include *Add Tasks, Run Script, and Set Field.* Add New

Workflows   Workflow Activities are started after all Ordered Activities complete.
              [Cloud Foundry: Stop Application ✕   ]

Destinations Transaction Data is posted to Message Destinations after all Ordered Activities complete and all Workflows have been started.

              [Select Destinations   ]   Show Activity Fields

Function fn_cloud_foundry_create_app: Cloud Foundry: Create an App

Functions / fn_cloud_foundry_manage_applications

Name *	Cloud Foundry: Application Command
API Name * ⓘ	fn_cloud_foundry_manage_applications
Message Destination *	cloud_foundry ▼
Description	Performs a specified action on the chosen Cloud Foundry applications. Currently available actions: start, stop, restage, info, delete, and instances.

Inputs

fn_cloud_foundry_applications	✕
fn_cloud_foundry_action	✕
fn_cloud_foundry_additional_parameters_json	✕

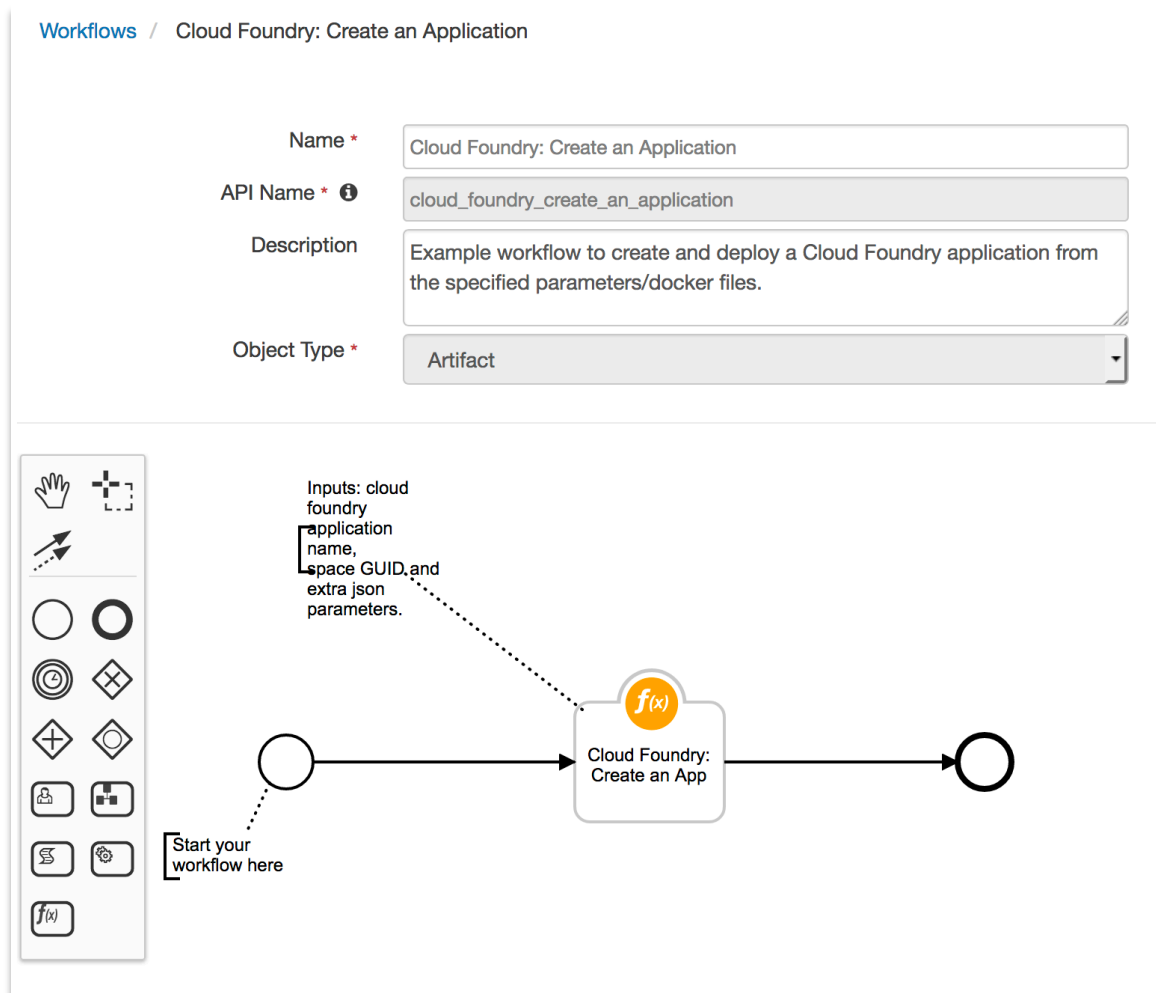The fn_cloud_foundry_create_app function creates a new application on the Cloud Foundry platform.

This function has 3 inputs:

- fn_cloud_foundry_applications: **Required** – single name of the application or docker application to be created.

- fn_cloud_foundry_space_guid: **Required** – GUID of the space in which the app should be created.

- fn_cloud_foundry_additional_parameters_json: **Optional** –Accepts valid JSON, which will be passed to the method executing chosen actions. These additional parameters can be used to specify the fields needed for the creation of a **Docker** application, as stated in the API. See the Cloud Foundry API documentation for possible JSON parameters. https://apidocs.cloudfoundry.org/

Return value will contain the information about the app, as provided in the API as well as "success" field to determine whether the app was, in fact, created.
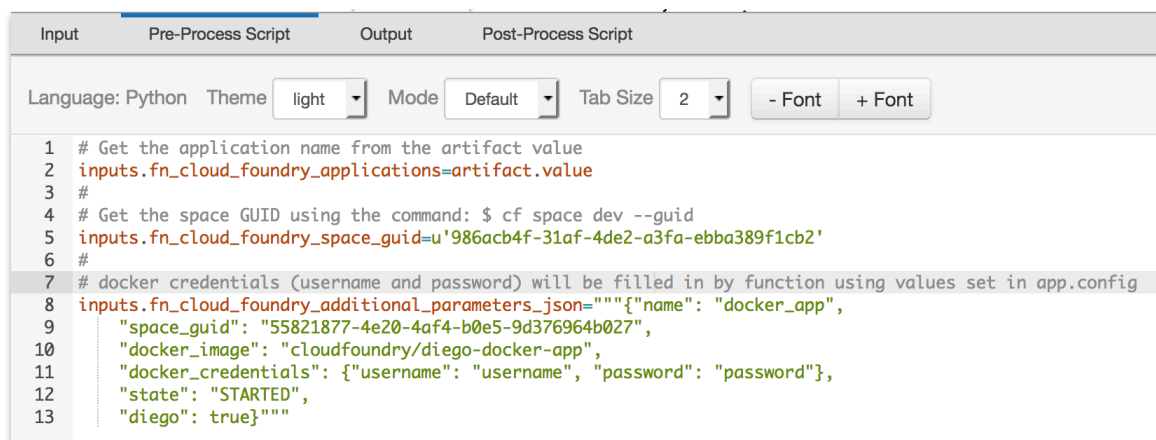
Workflow: Cloud Foundry: Create an Application

Below is a screen shot of an example workflow that calls the fn_cloud_foundry_create_app function. The Cloud Foundry: Create Application workflow specifies the application or Docker image to be created.

Workflows / Cloud Foundry: Create an Application

Name *   Cloud Foundry: Create an Application

API Name * ℹ   cloud_foundry_create_an_application

Description   Example workflow to create and deploy a Cloud Foundry application from the specified parameters/docker files.

Object Type *   Artifact

Inputs: cloud foundry application name, space GUID and extra json parameters.

Cloud Foundry: Create an App

Start your workflow here

Below is a screenshot of the pre-processor script for the example Cloud Foundry Create Application workflow. The application name is retrieved from the artifact value, which is an artifact of type "Service". The space GUID can be found using the Cloud Foundry command line interface command:
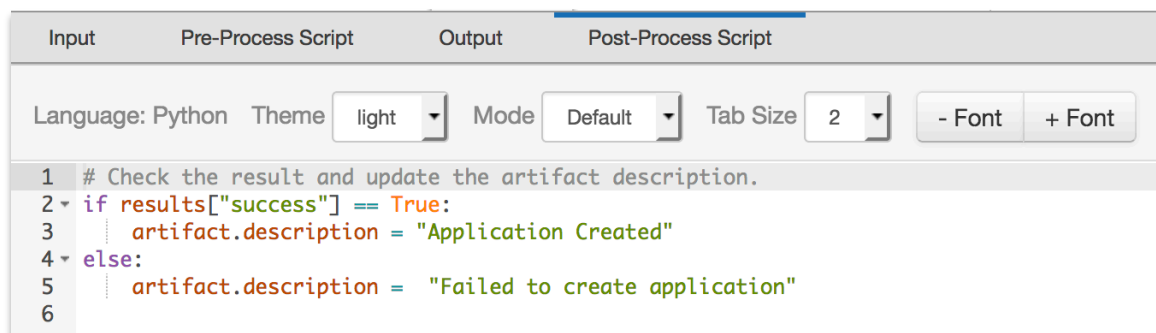
```
$ cf space dev --guid
```

Note the additional JSON parameters that are used to create the application which is a Docker image. See the Cloud Foundry API documentation for possible JSON parameters. https://apidocs.cloudfoundry.org/ "Creating a Docker App" or "Creating an App" link.

Input	Pre-Process Script	Output	Post-Process Script

Language: Python   Theme [light ▾]   Mode [Default ▾]   Tab Size [2 ▾]   [- Font] [+ Font]

```python
1   # Get the application name from the artifact value
2   inputs.fn_cloud_foundry_applications=artifact.value
3   #
4   # Get the space GUID using the command: $ cf space dev --guid
5   inputs.fn_cloud_foundry_space_guid=u'986acb4f-31af-4de2-a3fa-ebba389f1cb2'
6   #
7   # docker credentials (username and password) will be filled in by function using values set in app.config
8   inputs.fn_cloud_foundry_additional_parameters_json="""{"name": "docker_app",
9       "space_guid": "55821877-4e20-4af4-b0e5-9d376964b027",
10      "docker_image": "cloudfoundry/diego-docker-app",
11      "docker_credentials": {"username": "username", "password": "password"},
12      "state": "STARTED",
13      "diego": true}"""
```

Below is a screenshot of the post-processor script used in the Cloud Foundry Create Application workflow:

Input	Pre-Process Script	Output	Post-Process Script

Language: Python   Theme [light ▾]   Mode [Default ▾]   Tab Size [2 ▾]   [- Font] [+ Font]

```python
1   # Check the result and update the artifact description.
2   if results["success"] == True:
3       artifact.description = "Application Created"
4   else:
5       artifact.description =  "Failed to create application"
6
```

The artifact description is updated to indicate whether the application was successfully created or not.

Rule: Cloud Foundry: Create Application

The following screen shot shows an example Cloud Foundry: Create Application menu item rule which will appear in the Action Menu of an artifact when the artifact value is set to "docker_app". When the rule menu item is activated, the Cloud Foundry Create Application workflow is invoked on the deployed "docker_app" application. Modify or create your own rules to create and deploy your application.

Rules / Example: Cloud Foundry Create Application   Cancel   Save & Close

Display Name *   ple: Cloud Foundry Create Applicatic
Object Type   Artifact
                      Add custom conditions for when Menu Item will display. Clear All

                      Value   ▾   is equal to   ▾   docker_app   ✚ 🗑

Activities

Ordered   Ordered Activities will be invoked in the order specified below. They include *Add Tasks, Run Script, and Set Field.* Add New

Workflows   Workflow Activities are started after all Ordered Activities complete.

              Cloud Foundry: Create an Application ✕

Destinations Transaction Data is posted to Message Destinations after all Ordered Activities complete and all Workflows have been started.

              Select Destinations   Show Activity Fields

Function fn_cloud_foundry_instance_command: Cloud Foundry: Instance Command

Function fn_cloud_foundry_instance_command executes commands on the instances of a specified application. Currently only supports deleting an instance.

It takes in 4 inputs:

- fn_cloud_foundry_instances: **Required** – name or comma-separated names of the instances(s) that the actions should be applied to.

- fn_cloud_foundry_applications: **Required** – name of the single application whose instances should be affected by the action.

- fn_cloud_foundry_instance_action: **Required** – select field with the list of all possible actions to be performed on the instance. Currently, only delete.

- fn_cloud_foundry_additional_parameters_json: **Optional** – here for the purposes of allowing the actions to be added and extended in the future. Accepts valid JSON, which will be passed to the method executing chosen actions. See the Cloud Foundry API documentation for possible JSON parameters. https://apidocs.cloudfoundry.org/

Listed below, is the format of the results of the function. The results indicate whether the action was successfully applied to each of the instance of the application specified, as well as other data relevant to the action itself, per instance.

```
{
    "application-1": {
        "0":{
            "success": true,
            "other_data": "...",
            "_keys":["success", "other_data"]
        },
        "1": {
            "success":false,
            "details": "...",
            "_keys": ["success", "details"]
        },
        "_keys": ["0", "1"]
    },
    "_keys": ["application-1"]
}
```

"_keys" store the list of keys for each level of a dictionary to provide convenience in post-processing.

Workflow: Cloud Foundry: Instance Command

Below is a screen shot of an example workflow that calls the
fn_cloud_foundry_instance_command function. The Cloud Foundry: Instance Command
workflow specifies the application and a command to apply to each of the specified instances of
the application.

Workflows / Cloud Foundry: Instance Command

Name *   Cloud Foundry: Instance Command

API Name * ⓘ   cloud_foundry_instance_command
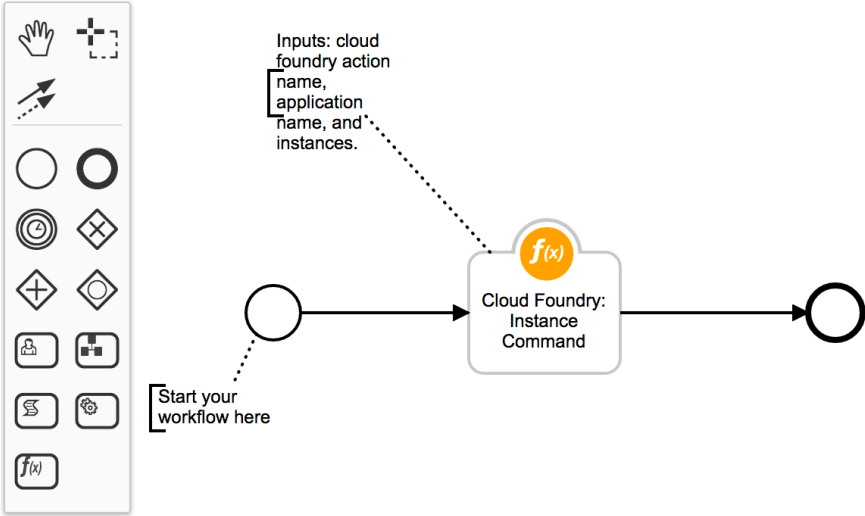
Description   This example workflow performs an action on the specified instances of a
                  Cloud Foundry application. Available commands: DELETE.

Object Type *   Artifact

Inputs: cloud
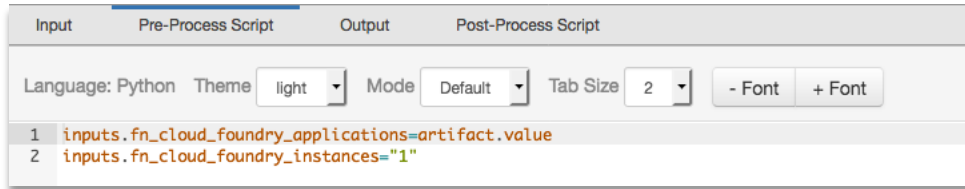foundry action
name,
application
name, and
instances.

Cloud Foundry:
Instance
Command

Start your
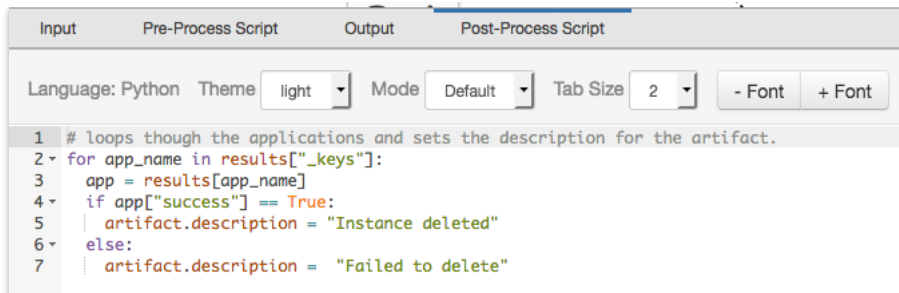workflow here

Below is a screenshot of the pre-processor script for the example Cloud Foundry Instance Command workflow. The application name is retrieved from the artifact value, which is an artifact of type "Service". The Cloud Foundry instances are a comma separated list of the application instances on which the command will be applied. In this workflow DELETE is the only command implemented.

Input	Pre-Process Script	Output	Post-Process Script

Language: Python Theme light ▾ Mode Default ▾ Tab Size 2 ▾ - Font + Font

```
1  inputs.fn_cloud_foundry_applications=artifact.value
2  inputs.fn_cloud_foundry_instances="1"
```

Below is a screenshot of the post-processor script used in the Cloud Foundry Instance Command workflow:

Input	Pre-Process Script	Output	Post-Process Script

Language: Python Theme light ▾ Mode Default ▾ Tab Size 2 ▾ - Font + Font

```
1  # loops though the applications and sets the description for the artifact.
2  for app_name in results["_keys"]:
3    app = results[app_name]
4    if app["success"] == True:
5      artifact.description = "Instance deleted"
6    else:
7      artifact.description =  "Failed to delete"
```

The artifact description is updated to indicate whether the command was successfully executed on each application instance.

Rule: Example: Cloud Foundry Instance Command

The following screen shot shows an Example: Cloud Foundry Instance Command menu item rule which will appear in the Action Menu of an artifact when the artifact value is set to "resilient-python-test". When the rule menu item is activated, the Cloud Foundry Instance Command workflow is invoked on the deployed "resilient-python-test" application. Modify or create your own rules to act on instances of your Clound Foundry application.

Rules / Example: Cloud Foundry Instance Command

Cancel Save & Close

Display Name * Example: Cloud Foundry Instance C

Object Type Artifact

Add custom conditions for when Menu Item will display. Clear All

Value ▼ is equal to ▼ resilient-python-test ➕ 🗑

Activities

Ordered Ordered Activities will be invoked in the order specified below. They include *Add Tasks, Run Script, and Set Field.* Add New

Workflows Workflow Activities are started after all Ordered Activities complete.

Cloud Foundry: Instance Command ✕

Destinations Transaction Data is posted to Message Destinations after all Ordered Activities complete and all Workflows have been started.

Select Destinations Show Activity Fields

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.