gRPC Function for IBM Resilient

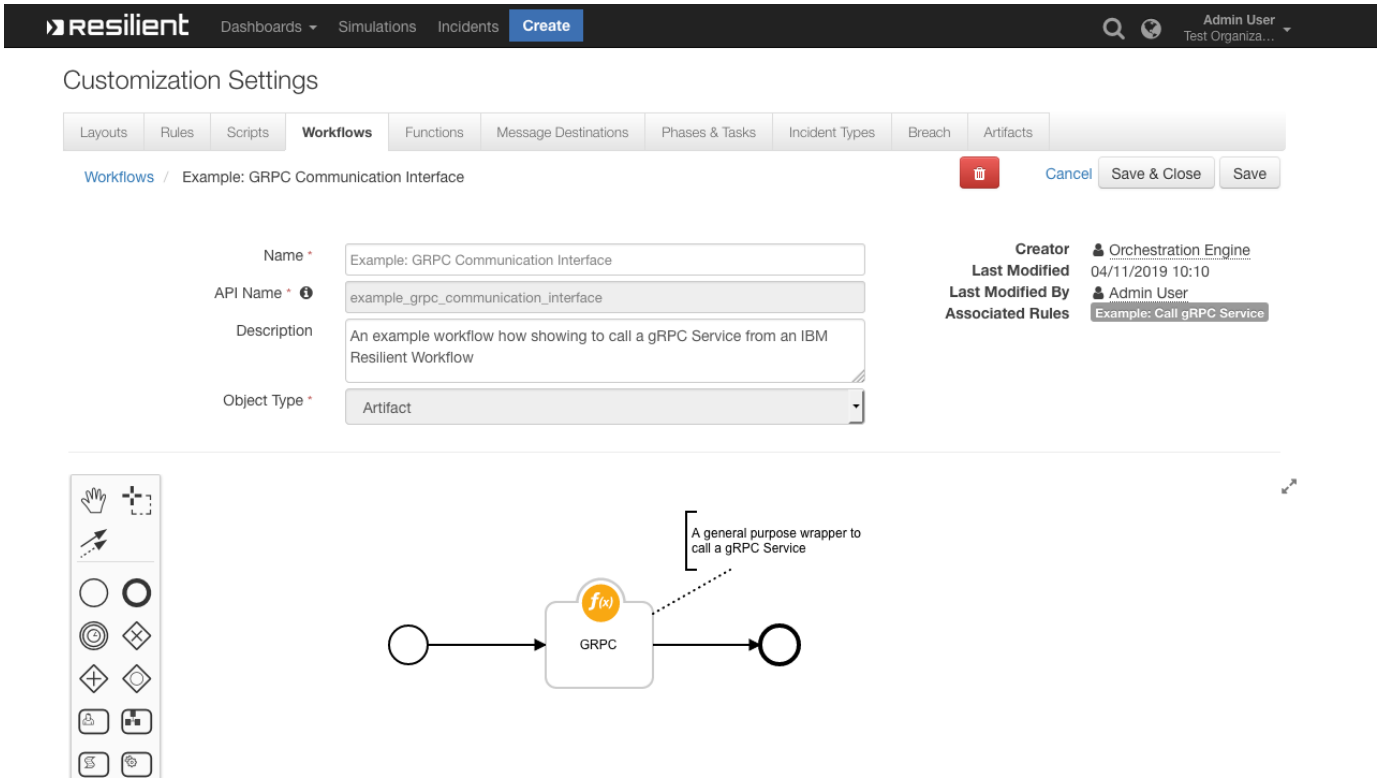
Table of Contents

- About
- Requirements
- Installation
- Function Inputs
- Function Output
- Pre-Process Script
- Post-Process Script
- Rules

About

This Function provides a general purpose wrapper that allows you to call gRPC services from within IBM Resilient

- Using gRPC you can efficiently connect services in and across data centers to help with your Incident Response
- This version supports **Unary RPCs** where the client sends a single request to the server and gets a single response back, just like a normal function call.
- See https://grpc.io/ for more information on gRPC
- We recommend following the gRPC `helloworld` example here to help get this Integration up and running.



Requirements

- Resilient Appliance >= v31.0.0
- Integrations Server running resilient_circuits >= v30.0.0
- A knowledge of Remote Procedure Calls (RPCs) and gRPC

Installation

- Download fn_grpc_interface.zip from our App Exchange
- Copy the .zip to your Integrations Server and SSH into it.
- **Unzip** the package:

```
$ unzip fn_grpc_interface-x.x.x.zip
```

- **Install** the package:

```
$ pip install fn_grpc_interface-x.x.x.tar.gz
```

- Import the **configurations** into your app.config file:

```
$ resilient-circuits config -u
```

- Import the fn-grpc-interface **customizations** into the Resilient Appliance:

```
$ resilient-circuits customize -y -l fn-grpc-interface
```

- Open the config file, scroll to the bottom and **edit your gRPC configurations**:

```
$ nano ~/.resilient/app.config
```

```
[fn_grpc_interface]
interface_dir=<<path to the parent directory of your Protocol Buffer (pb2) files>>
#<<package_name>>=<<communication_type>>, <<secure connection type>>,
<<certificate_path or google API token>>

# Note: to setup, in your interface_dir, create a sub-directory that has
# the same name as your package, and copy the interface buffer pb2 files
# into that directory.
# 'package_name' is a CSV list of length 3, where each possible value is described
in the documentation
#
# If the package_name was 'helloworld', your app.config would look like:
# [fn_grpc_interface]
# interface_dir=/home/admin/integrations/grpc_interface_files
# helloworld=unary, None, None
```

Configuration Parameters	Description	Example
`interface_dir`	The parent directory containing the gRPC client (pb2) files. These files are auto-generated from your .proto file via the grpc-tools utility.	interface_dir=/usr/local/grpc_clients/

Configuration Parameters	Description	Example
`<<package_name>>=<<communication_type>>,<<secure_connection_type>>,<<certificate_path or google_API_token>>`	*package_name:* Define one `package_name` per line. Within the `interface_dir`, create a directory with the same name as `package_name` where the client Protocol Buffer files will reside. *communication_type:* Currently we only support Unary RPCs so this value must be - `unary`. For further information, refer to https://grpc.io/docs/guides/concepts.html *secure_connection:* We currently support `SSL` or `TLS` secure connections. This value can be `SSL`, `TLS` or `None`. If SSL/TLS, ensure you provide a `certificate_path` *certificate_path/google token:* If `secure_connection` is defined, specify either a path to the certificate file or the token provided by Google	helloworld=unary,None,None

- **Save** and **Close** the app.config file.
- To **uninstall** the package:

```
$ pip uninstall fn_grpc_interface
```

Function Inputs

Input Name	Type	Required	Example	Info
`grpc_channel`	`String`	Yes	`"localhost:50051"`	The `host` and `port` of the gRPC Server
`grpc_function`	`String`	Yes	`"helloworld:SayHello(HelloRequest)"`	`<<RPC .proto file name>>:<<RPC Service Definition Method>>(<<RPC Service Definition Parameter>>)`
`grpc_function_data`	`JSON String`	Yes	`'{ "name": "Joe Bloggs" }'`	A JSON String of the Object to get passed as the `RPC Service Definition Parameter`

NOTE: the `grpc_function` is derived from the your `.proto` file like the `helloworld.proto` example:

```
syntax = "proto3";

option java_multiple_files = true;
option java_package = "io.grpc.examples.helloworld";
option java_outer_classname = "HelloWorldProto";
option objc_class_prefix = "HLW";

package helloworld;

// The greeting service definition.
service Greeter {
 // Sends a greeting
 rpc SayHello (HelloRequest) returns (HelloReply) {}
}

// The request message containing the user's name.
message HelloRequest {
 string name = 1;
}

// The response message containing the greetings
message HelloReply {
 string message = 1;
}
```

Function Output

- The gRPC Server Response is returned in `results.content`
- An attempt is made to convert the gRPC Server Response to a Python Dictionary
- Therefore `results.content` will either be a Python Dictionary or (JSON) String
- To see the full output of the Function, we recommend running `resilient-circuits` in `DEBUG` mode.
- To do this run:

```
$ resilient-circuits run --loglevel=DEBUG
```

Pre-Process Script

This example passes the `artifact.value` as a gRPC Request Parameter

```
def dict_to_json_str(d):
  """Function that converts a dictionary into a JSON string.
    Supports types: basestring, unicode, bool, int and nested dicts.
    Does not support lists.
    If the value is None, it sets it to False."""

  json_entry = u'"{0}":{1}'
  json_entry_str = u'"{0}":"{1}"'
  entries = []

  for entry in d:
    key = entry
    value = d[entry]

    if value is None:
      value = False
```

```
    if isinstance(value, list):
      helper.fail('dict_to_json_str does not support Python Lists')

    if isinstance(value, basestring):
      value = value.replace(u'"', u'\\"')
      entries.append(json_entry_str.format(unicode(key), unicode(value)))

    elif isinstance(value, unicode):
      entries.append(json_entry.format(unicode(key), unicode(value)))

    elif isinstance(value, bool):
      value = 'true' if value == True else 'false'
      entries.append(json_entry.format(key, value))

    elif isinstance(value, int):
      entries.append(json_entry.format(unicode(key), value))

    elif isinstance(value, dict):
      entries.append(json_entry.format(key, dict_to_json_str(value)))

    else:
      helper.fail('dict_to_json_str does not support this type: {0}'.format(type(value)))

  return u'{0} {1} {2}'.format(u'{', ','.join(entries), u'}')

# Define Inputs

# The gRPC Channel to use
inputs.grpc_channel = "localhost:50051"

# The gRPC Function to call
inputs.grpc_function = "helloworld:SayHello(HelloRequest)"

# The gRPC Function Request Data
inputs.grpc_function_data = dict_to_json_str({"name": artifact.value})
```

Post-Process Script

In this example we add a Note to the Incident containing the gRPC Server Response:

```
grpc_response_data = results['content']
grpc_channel = results['channel']

rich_text = helper.createRichText(u"""A gRPC Response has been received from <b>{0}</b>
<br>
                                     <b>Response:</b> {1}""".format(grpc_channel,
grpc_response_data))

incident.addNote(rich_text)
```

Rules

Rule Name	Object Type	Workflow Triggered
Example: Call gRPC Service	Artifact	Example: GRPC Communication Interface