

# IBM Security QRadar SOAR: Building Custom Integrations with the App Host

---



# IBM Security

**Bo Bleckel & Shane Curtin, App Engineers, IBM Security QRadar SOAR**

See the live version of this document at <https://github.com/ibmresilient/resilient-reference/blob/MSU2022/MSU2022/Lab%20Guides/Part%20I:%20Building%20Custom%20Integrations%20With%20the%20App%20Host/README.md>

---

## Contents

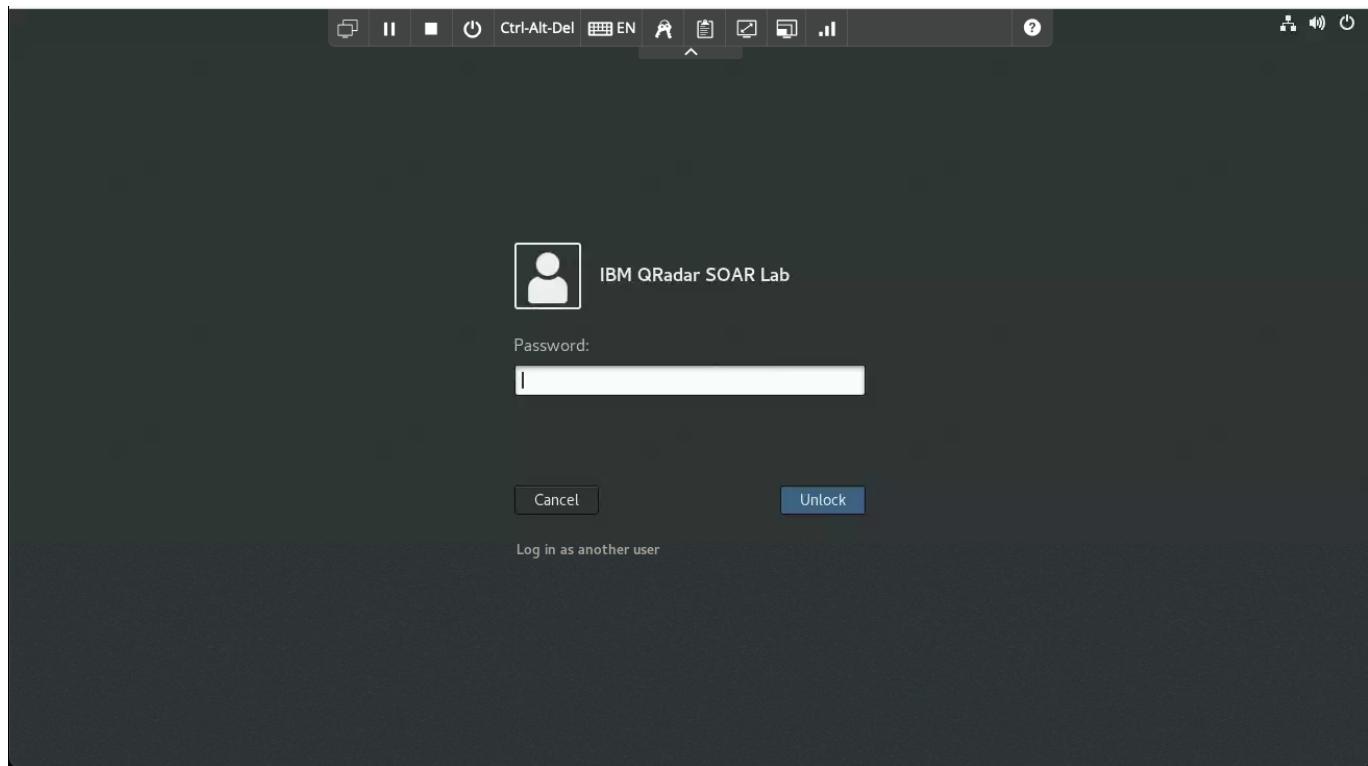
- Step 0: *Sign up for SkyTap Account*
  - Step 1: *Login to Virtual Environment*
  - Step 2: *VS Code IDE Setup*
  - Step 3: *resilient-circuits Configuration*
  - Step 4: *Create Message Destination in SOAR*
  - Step 5: *Create Functions for fn\_my\_ldap*
  - Step 6: *Create a custom Datatable*
  - Step 7: *Create the Function Package in Code*
  - Step 8: *Fill in the Code for the Function and Utility Files*
  - Step 9: *Create a Playbook to Test*
  - Step 10: *Start resilient-circuits Server*
  - Step 11: *Test*
  - Step 12: *Debug resilient-circuits Server*
  - Step 13: *Validate*
  - Step 14: *Package*
  - Step 15: *Push to local registry*
  - Step 16: *Install with App Host*
- 

## Step 0: *Sign up for SkyTap Account*

- Go to [https://ibm.biz/MSU22SkyTap + \\_<your\\_number>](https://ibm.biz/MSU22SkyTap + _<your_number>)

- Example: I'll use [https://ibm.biz/MSU22SkyTap\\_01](https://ibm.biz/MSU22SkyTap_01)
  - Sign in with email and password provided by course provider
- 

## Step 1: Login to Virtual Environment



- Login as **resadmin** with the password shared by your course leader.
- Open **FireFox** and log into **IBM Security QRadar SOAR**.
- Open a new Terminal and restart the **ldap** and **registry** podman containers:
  - List all podman processes:

```
sudo podman ps -a
```

- From the output of the above command, start those containers:

```
sudo podman start <CONTAINER_ID_1> <CONTAINER_ID_2>
```

The screenshot shows a terminal window titled 'resadmin@host-1:~'. The window displays the output of the command 'sudo podman ps -a'. Two specific lines of output are highlighted with red boxes:

```
[resadmin@host-1 ~]$ sudo podman ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
 NAMES
5b2de41bde75 docker.io/rroemhild/test-openldap:latest -h ldapi:/// ldap... 41 hours ago Exited (0) 28 seconds ago 0.0.0.0:10389->10389/tcp
cp_vigorous villani
b53b135b4977 docker.io/library/registry:2 /etc/docker/regis... 41 hours ago Exited (2) 28 seconds ago 0.0.0.0:5000->5000/tcp
registry
[resadmin@host-1 ~]$ sudo podman start 5b2de41bde75 b53b135b4977
5b2de41bde75bba771040f9c91c5abf2419f9b394f232cd40f298d33e30fd646
b53b135b4977d9bf4ec8d15837607de587c2a00b1993e08fda8bae5ad01dc720
[resadmin@host-1 ~]$
```

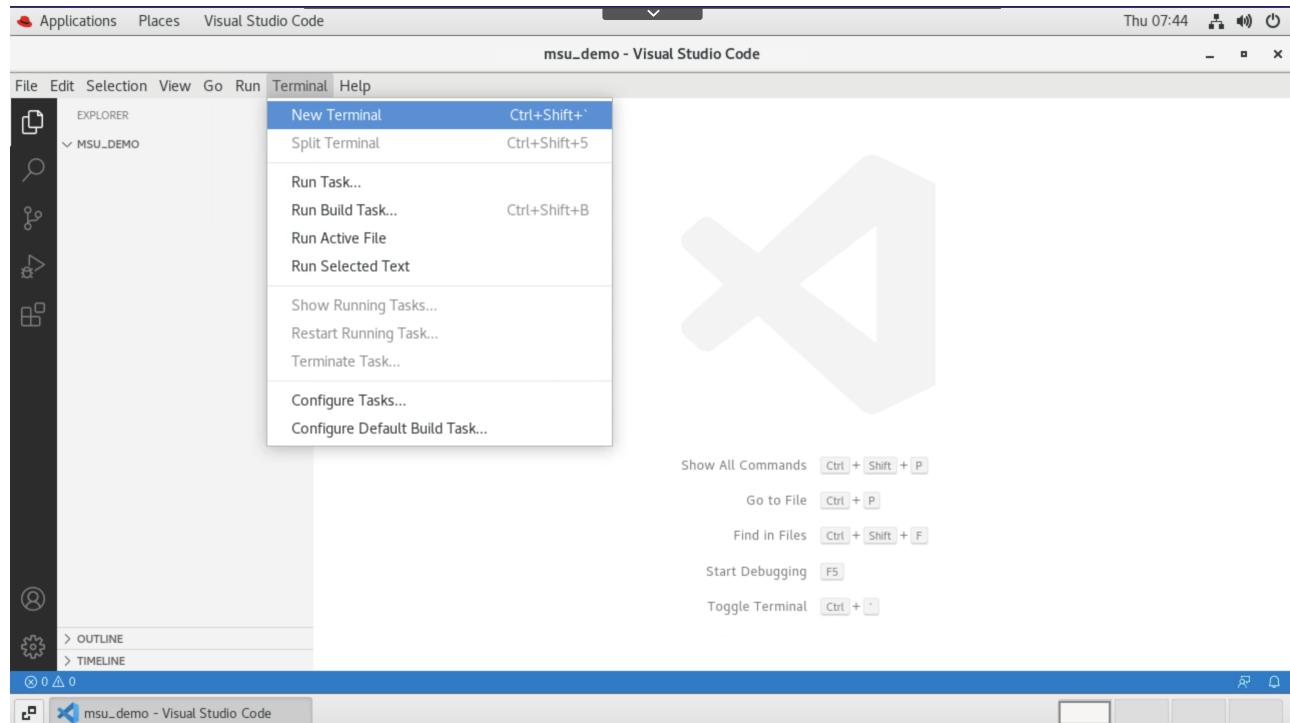
More info on the LDAP container used in this lab can be found [here](#)

## Step 2: VS Code IDE Setup

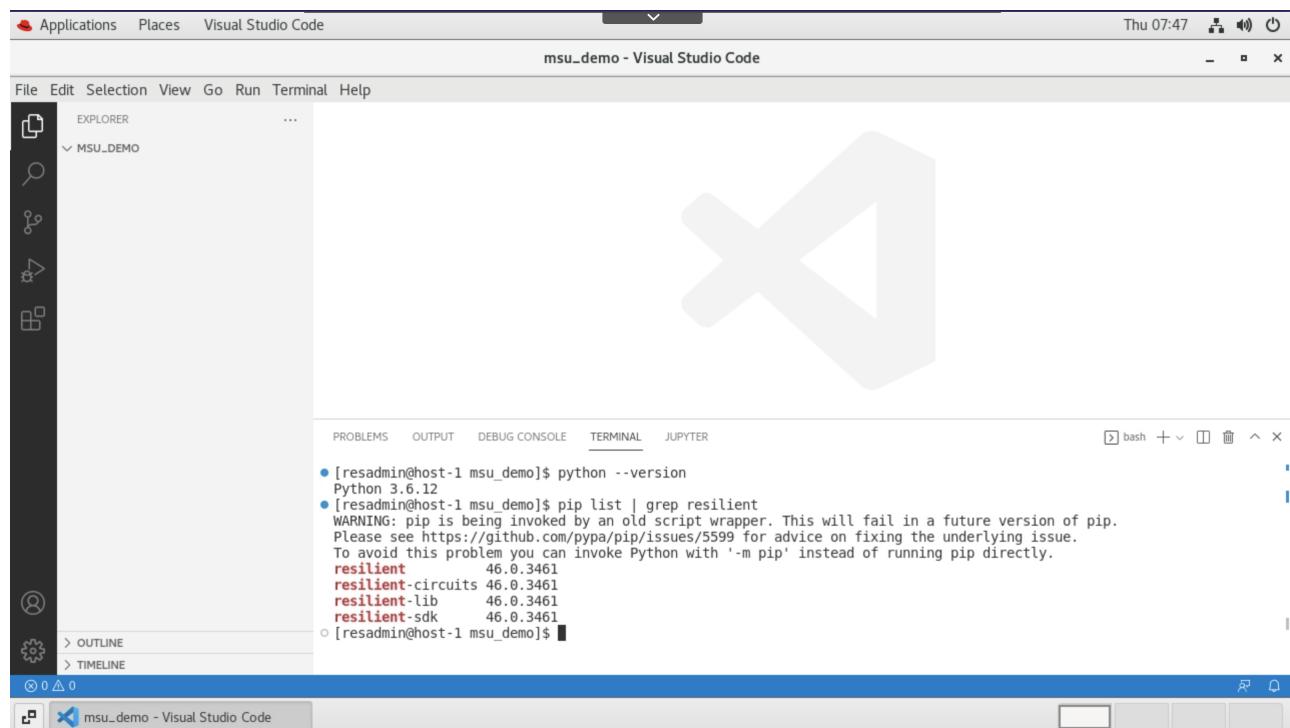
- Open VS Code



- Create a new terminal



- Ensure that the python version is 3.6.x and that pip has **resilient-circuits** and **resilient-sdk** installed



*Note that you can use a later version of Python. As of publication of this lab, IBM Security QRadar SOAR officially supports 3.6 and 3.9 for app development.*

If you don't see the required resilient packages or the versions are out of date, run **pip install -U resilient-circuits resilient-sdk** to install/upgrade the packages. Note that **resilient** and **resilient-lib** are packaged with **resilient-circuits** and don't need to be explicitly installed.

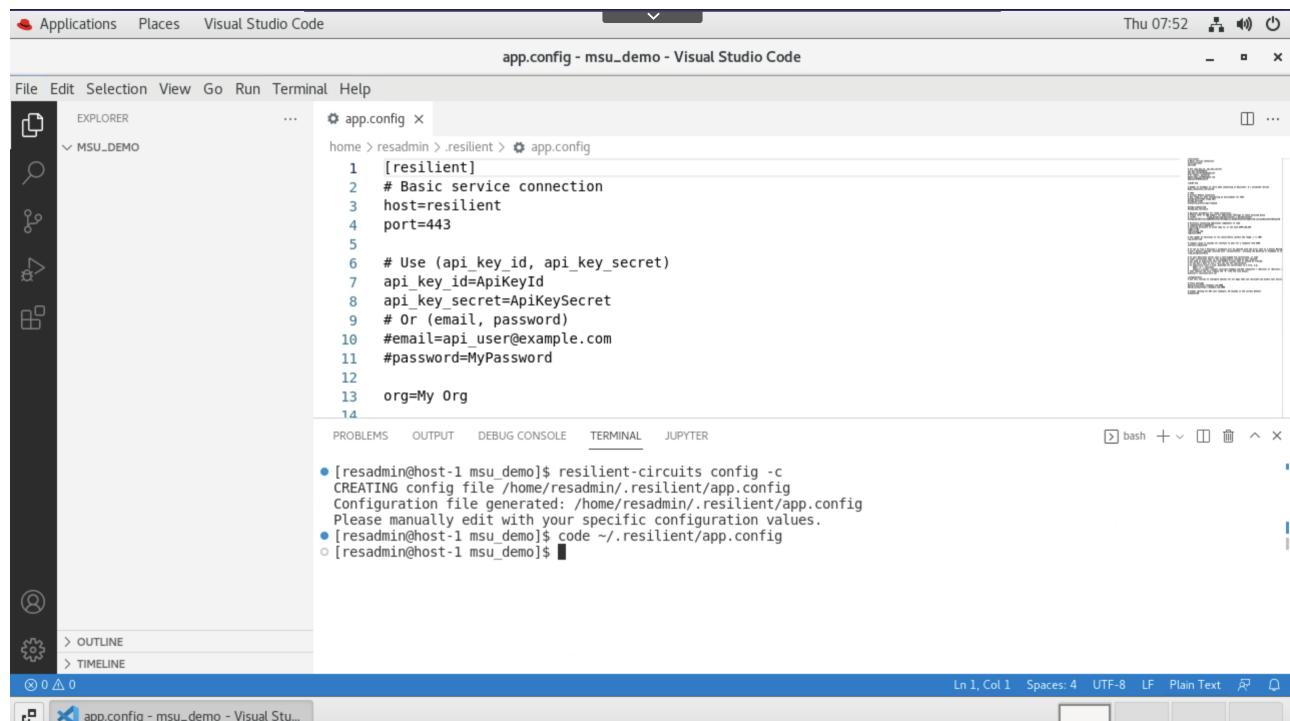
## Step 3: *resilient-circuits* Configuration

- In your terminal, run the following command to create a local *app.config* file:

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

And open the file in your IDE by running

```
code ~/.resilient/app.config
```



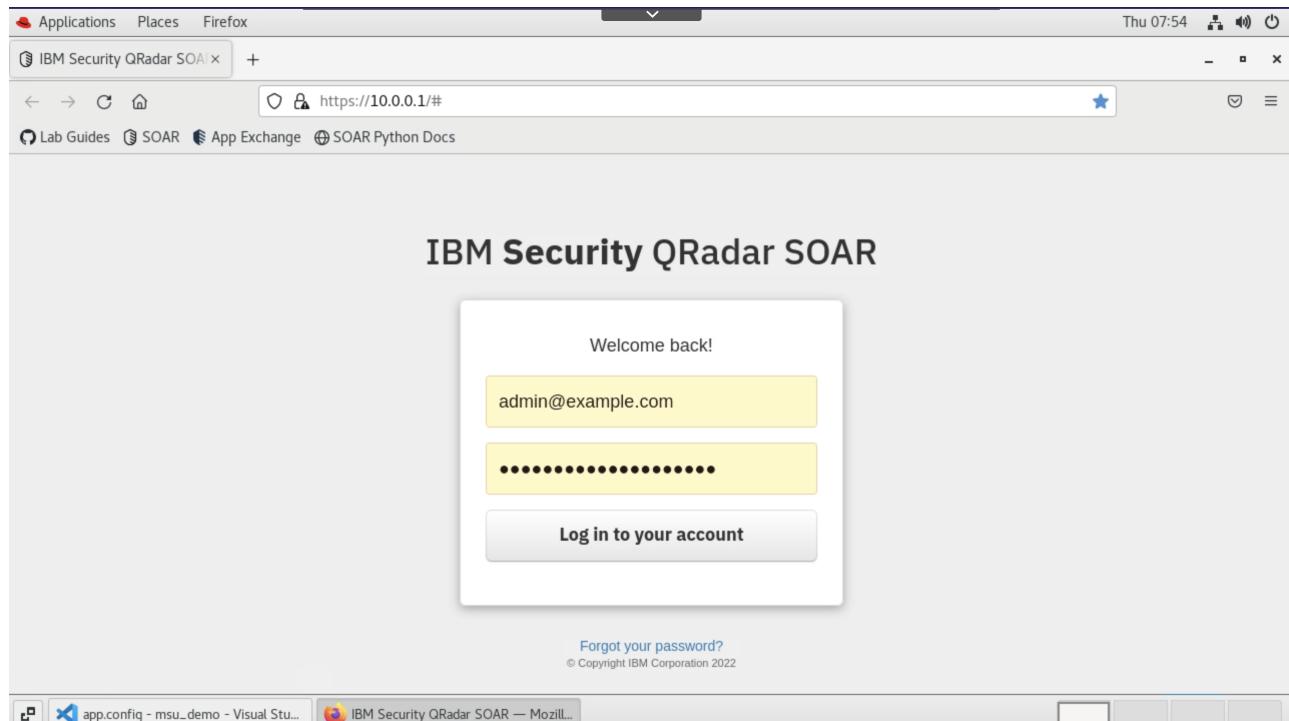
The screenshot shows the Visual Studio Code interface with the file `app.config` open. The code content is as follows:

```
1 [resilient]
2 # Basic service connection
3 host=resilient
4 port=443
5
6 # Use (api_key_id, api_key_secret)
7 api_key_id=ApiKeyId
8 api_key_secret=ApiKeySecret
9 # Or (email, password)
10 #email=api user@example.com
11 #password=MyPassword
12
13 org=My Org
14
```

Below the code editor, the terminal pane shows the command being run and its output:

```
[resadmin@host-1 msu_demo]$ resilient-circuits config -c
CREATING config file /home/resadmin/.resilient/app.config
Configuration file generated: /home/resadmin/.resilient/app.config
Please manually edit with your specific configuration values.
[resadmin@host-1 msu_demo]$ code ~/.resilient/app.config
[resadmin@host-1 msu_demo]$
```

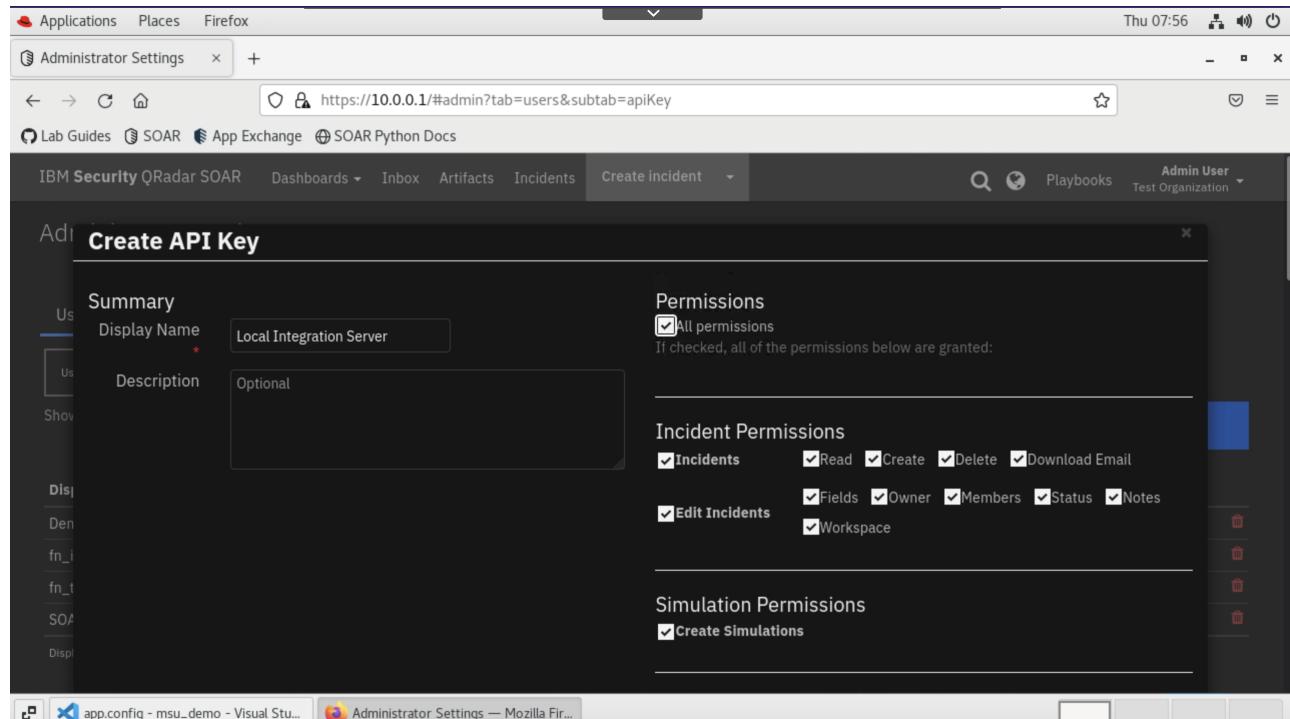
- Switch to Firefox and navigate to your SOAR host, in this case, 10.0.0.1 or through the bookmark in the bookmark menu. Log in using the saved admin credentials.



- Navigate to the **API Keys** section of the **Users** tab in **Administrator Settings** and create a new API Key.

Display Name	Last Renewal Date	Last Modified Date	Last Modified by	Create Date	Creator	Secret Expiry Date	Locked
Demo EDR App	09/06/2022 13:50	Admin User	09/06/2022 13:50	Admin User	09/06/2022 13:50	09/06/2023 13:50	false
fn_ioc_parser_v2	09/06/2022 13:43	Admin User	09/06/2022 13:43	Admin User	09/06/2022 13:43	09/06/2023 13:43	false
fn_task_utils	09/06/2022 13:58	Admin User	09/06/2022 13:58	Admin User	09/06/2022 13:58	09/06/2023 13:58	false
SOAR LDAP Utilities	09/06/2022 14:00	Admin User	09/06/2022 14:00	Admin User	09/06/2022 14:00	09/06/2023 14:00	false

- Give the API Key the name "Local Integration Server" and give it all permissions. Click Create.



- IMPORTANT: Before clicking away from the screen that displays the credentials, copy them to your clipboard. (If you accidentally navigate away from this screen that's ok. You can click into the key and simply regenerate the credentials.)
- Switch back to VS Code. Here we'll fill in the required values for our `app.config` file, including the copied API Key info, the Organization (in this case **Test Organization**) and the host information.

```

[resilient]
# Basic service connection
host=10.0.0.1
port=443

# Use (api key id, api key secret)
api_key_id=4b2543f1-fb4e-49f7-a4b0-f55a6655b612
api_key_secret=Bv0qey40fr4Ibk3ZrJHR9BnZD0myjk1zF6GnyWq9wJo

# Or (email, password)
#email=api_user@example.com
#password=MyPassword

org=Test Organization

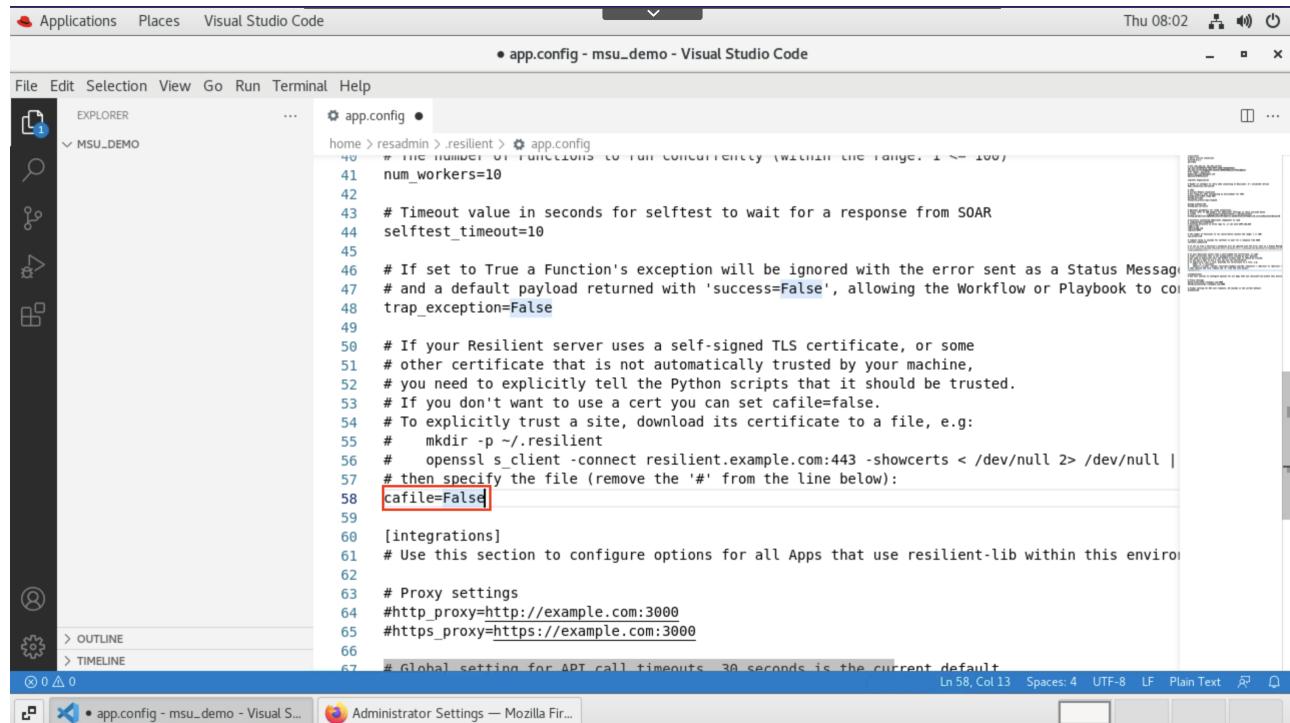
# Number of attempts to retry when connecting to Resilient. 0 = unlimited retries
#max_connection_retries=10

# CP4S
# Actions Module connection
# Use stomp_url when configuring an environment for CP4S
#stomp_host=<CP4S stomp URL>
#stomp_port=443
#resource_prefix=/api/respond

#stomp_timeout=120
#stomp_max_retries=3

```

Scroll down to find the `cafile` setting and set to `False`:



```

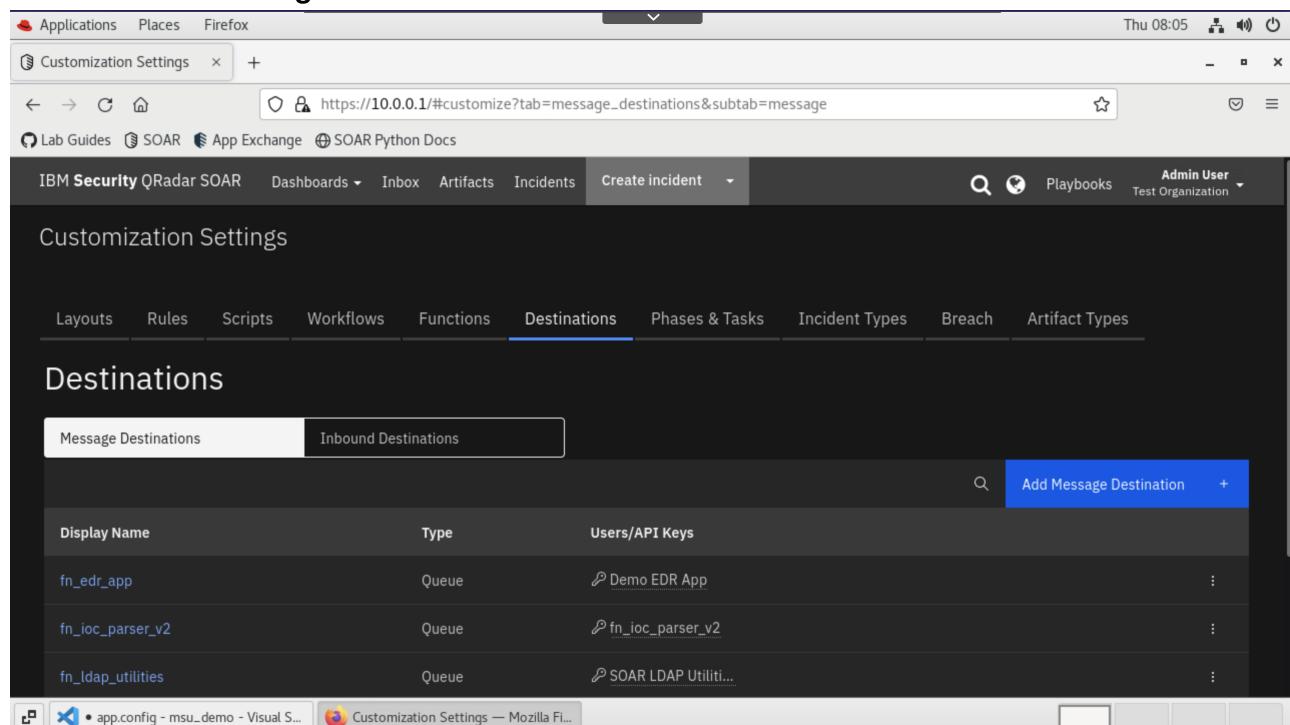
    Applications Places Visual Studio Code
    Thu 08:02 ⌂ ⌊ ⌋ ⌈ ⌉ ⌍
    • app.config - msu_demo - Visual Studio Code
    File Edit Selection View Go Run Terminal Help
    EXPLORER ... ⚙ app.config ●
    home > resadmin > .resilient > ⚙ app.config
    40 # THE NUMBER OF FUNCTIONS TO RUN CONCURRENTLY (WITHIN THE RANGE: 1 ~ 100)
    41 num_workers=10
    42
    43 # Timeout value in seconds for selftest to wait for a response from SOAR
    44 selftest_timeout=10
    45
    46 # If set to True a Function's exception will be ignored with the error sent as a Status Message
    47 # and a default payload returned with 'success=False', allowing the Workflow or Playbook to continue
    48 trap_exception=False
    49
    50 # If your Resilient server uses a self-signed TLS certificate, or some
    51 # other certificate that is not automatically trusted by your machine,
    52 # you need to explicitly tell the Python scripts that it should be trusted.
    53 # If you don't want to use a cert you can set cafile=false.
    54 # To explicitly trust a site, download its certificate to a file, e.g:
    55 #   mkdir -p ~/.resilient
    56 #   openssl s_client -connect resilient.example.com:443 -showcerts < /dev/null 2> /dev/null |
    57 # then specify the file (remove the '#' from the line below):
    58 cafile=False
    59
    60 [integrations]
    61 # Use this section to configure options for all Apps that use resilient-lib within this environment
    62
    63 # Proxy settings
    64 #http_proxy=http://example.com:3000
    65 #https_proxy=https://example.com:3000
    66
    67 # Global setting for APT call timeouts. 30 seconds is the current default

```

Ln 58, Col 13 Spaces: 4 UTF-8 LF Plain Text ⌂ ⌋ ⌈ ⌉ ⌍

## Step 4: Create Message Destination in SOAR

- Switch back to the SOAR platform and navigate in the UI to the **Destinations** tab found within **Customization Settings**.



The screenshot shows the IBM Security QRadar SOAR platform interface. The top navigation bar includes 'IBM Security QRadar SOAR', 'Dashboards', 'Inbox', 'Artifacts', 'Incidents', 'Create incident', 'Playbooks', and 'Admin User Test Organization'. Below this is the 'Customization Settings' header with tabs for 'Destinations' (which is selected), 'Layouts', 'Rules', 'Scripts', 'Workflows', 'Functions', 'Phases & Tasks', 'Incident Types', 'Breach', and 'Artifact Types'. The main content area is titled 'Destinations' and contains two tabs: 'Message Destinations' (selected) and 'Inbound Destinations'. A search bar and a blue 'Add Message Destination' button are at the top of the list. The table lists three message destinations:

Display Name	Type	Users/API Keys
fn_edr_app	Queue	Demo EDR App
fn_ioc_parser_v2	Queue	fn_ioc_parser_v2
fn_ldap_utilities	Queue	SOAR LDAP Utilities

- Create a new **Message Destination** of type **Queue** with display name and API name `fn_my_ldap`. Add the API Key generated earlier to the list of allowed keys for this message destination. Click **Add**.

The screenshot shows the 'Customization Settings' section of the IBM Security QRadar SOAR interface. On the left, there's a sidebar with 'Destinations' and a list of 'Message Destinations' including 'fn\_edr\_app', 'fn\_ioc\_parser\_v2', 'fn\_ldap\_utilities', and 'fn\_task\_utils'. A modal window titled 'Add Message Destination' is open in the center. It has fields for 'Display Name' (set to 'fn\_md\_ldap') and 'API Name' (also set to 'fn\_md\_ldap'). A toggle switch for 'Expect Acknowledgement' is set to 'Yes'. Below these are dropdowns for 'Users/API Keys (optional)' and 'Artifact Types', both currently set to 'Breach'. At the bottom of the modal are 'Cancel' and 'Add' buttons, with 'Add' being highlighted in blue.

## Step 5: Create Functions for *fn\_my\_ldap*

- Navigate to the **Functions** tab found with **Customization Settings**. Create a new Function called **My LDAP: Search**:

The screenshot shows the 'Customization Settings' interface with the 'Functions' tab selected. A new function is being created with the following details:

- Name:** My LDAP: Search
- API Name:** my\_ldap\_search
- Message Destination:** fn\_my\_ldap
- Description:** A description of the Function.

The 'Inputs' section is currently empty, indicated by the placeholder "Drag fields here". The 'Input Fields' section shows a single input field named "edr\_device\_id". Action buttons at the top right include "Cancel", "Save & Close" (highlighted in blue), and "Save".

- Create the input field `my_ldap_search_filter` and add it to the function:

The screenshot shows two main windows from the IBM Security QRadar SOAR platform.

**Top Window: Create Input Field**

- Type of field:** Text
- API name:** my\_ldap\_search\_filter
- Requirement:** Always
- Placeholder value (Optional):** A placeholder value
- Buttons:** Cancel, Create

**Bottom Window: Functions / New**

- Name:** My LDAP: Search
- API Name:** my\_ldap\_search
- Message Destination:** fn\_my\_ldap
- Description:** A description of the Function.
- Inputs:** my\_ldap\_search\_filter (highlighted with a red box)
- Input Fields:** my\_ldap, my\_ldap\_search\_filter (highlighted with a red box)
- Buttons:** Cancel, Save & Close, Save

- Save & Close the function.
- Repeat this process for a function called `My LDAP: Toggle Access` with inputs `my_ldap_user_dn` (Text, Required), `my_ldap_enable` (Boolean, Optional) and `my_ldap_disable`

(Boolean, Optional):

The screenshot shows the 'Functions' tab selected in the navigation bar. A new function is being created with the following details:

- Name:** My LDAP: Toggle Access
- API Name:** my\_ldap\_toggle\_access
- Message Destination:** fn\_my\_ldap
- Description:** A description of the Function.
- Inputs:** my\_ldap\_user\_dn, my\_ldap\_enable, my\_ldap\_disable
- Associated Workflows:** Function is not currently referenced by any workflow.

At the bottom right, the 'Save & Close' button is highlighted in blue.

- **Save & Close** the function.

## Step 6: Create a custom Datatable

- Navigate to the **Layouts** tab found with **Customization Settings**.
- Select **Incident Tabs** and scroll down and select any of the specific tabs. We chose **Artifacts** as that is where we'll place the custom datatable.

The screenshot shows the 'Layouts' tab selected in the navigation bar. In the 'Incident Tabs' section, the 'Tasks' tab is selected. Other tabs include Details, Breach, Notes, Members, and News Feed. At the bottom right, the 'Save' button is highlighted in blue.

The screenshot shows the IBM Security QRadar SOAR web interface. The left sidebar has 'Artifacts' selected. The right panel shows various data tables like 'California Health Risk Assessment' and 'City'. A red box highlights the 'Add Table' button in the 'Data Tables' section.

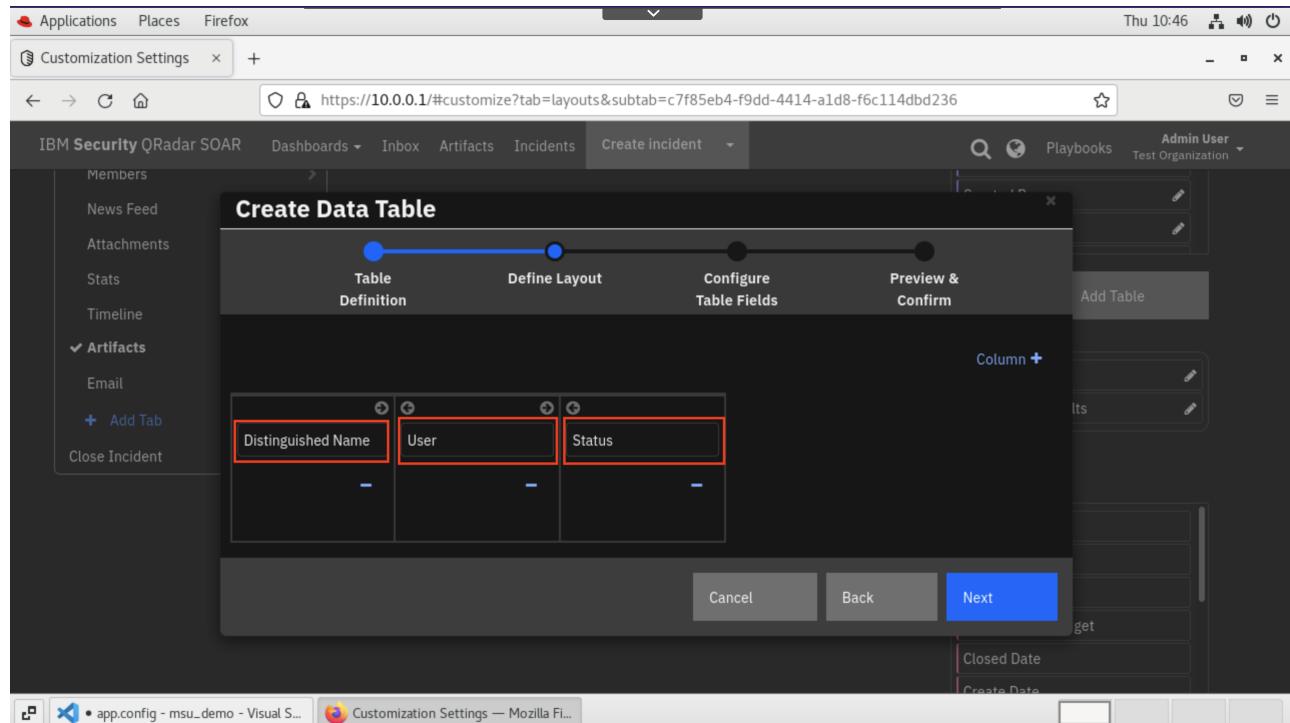
Click **Add Table**.

- Name the table **My LDAP: Results** and give it an API name **my\_ldap\_dt**.

The screenshot shows the 'Create Data Table' wizard. Step 1: Table Definition. It asks for the table label ('My LDAP: Results') and API access name ('my\_ldap\_dt'). A red box highlights the 'Next' button.

Click **Next**.

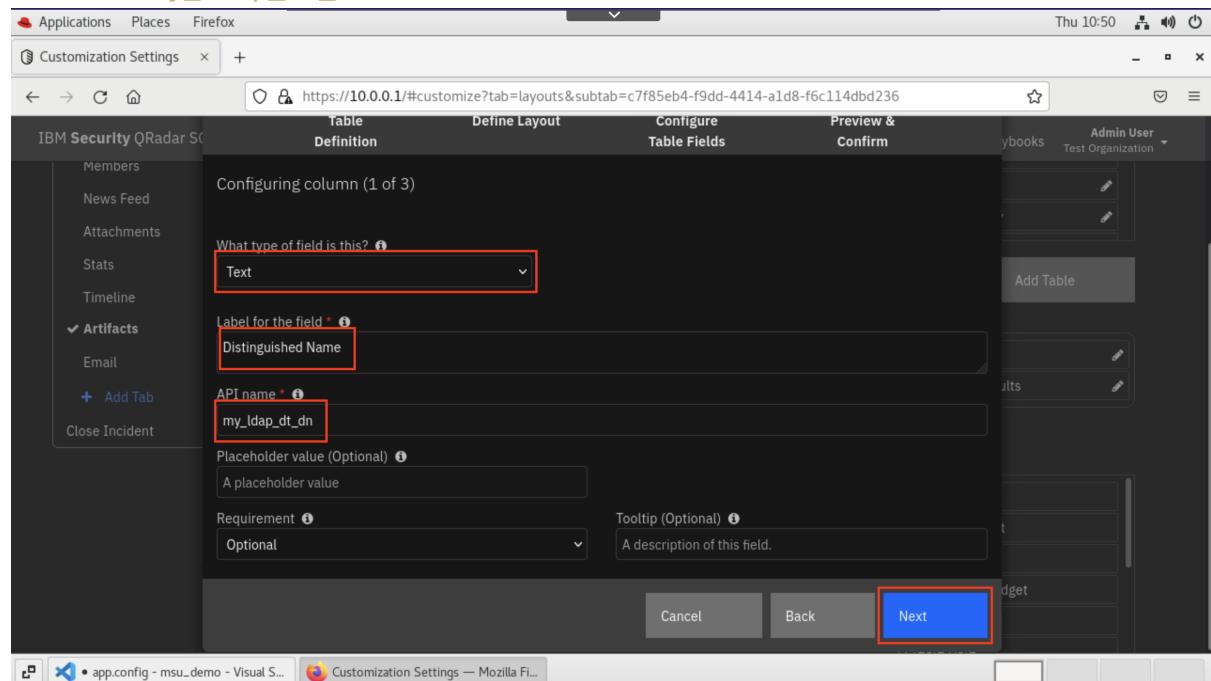
- Create three columns: **Distinguished Name, User, Status**:



Click **Next**. This will now walk you through each column's details.

- Fill in the following details for the "Distinguished Name" column:

- Type: **Text**
- Label: "Distinguished Name"
- API Name: **my\_ldap\_dt\_dn**



Click **Next**.

- Fill in the following details for the "User" column:

- Type: **Text**
- Label: "User"

- API Name: **my\_ldap\_dt\_user**

The screenshot shows the 'Customization Settings' interface in a Firefox browser. The URL is <https://10.0.0.1/#customize?tab=layouts&subtab=c7f85eb4-f9dd-4414-a1d8-f6c114dbd236>. The main panel is titled 'Configuring column (2 of 3)'. It shows the following fields:
 

- 'What type of field is this?' dropdown set to 'Text'.
- 'Label for the field' input field containing 'User'.
- 'API name' input field containing 'my\_ldap\_dt\_user'.
- 'Placeholder value (Optional)' input field containing 'A placeholder value'.
- 'Requirement' dropdown set to 'Optional'.
- 'Tooltip (Optional)' input field containing 'A description of this field'.

 At the bottom right are 'Cancel', 'Back', and 'Next' buttons, with 'Next' highlighted in a red box.

**Click Next.**

- Fill in the following details for the "Status" column:

- Type: **Text Area**
- Label: "Status"
- API Name: **my\_ldap\_dt\_status**
- Check "Add rich Text controls"

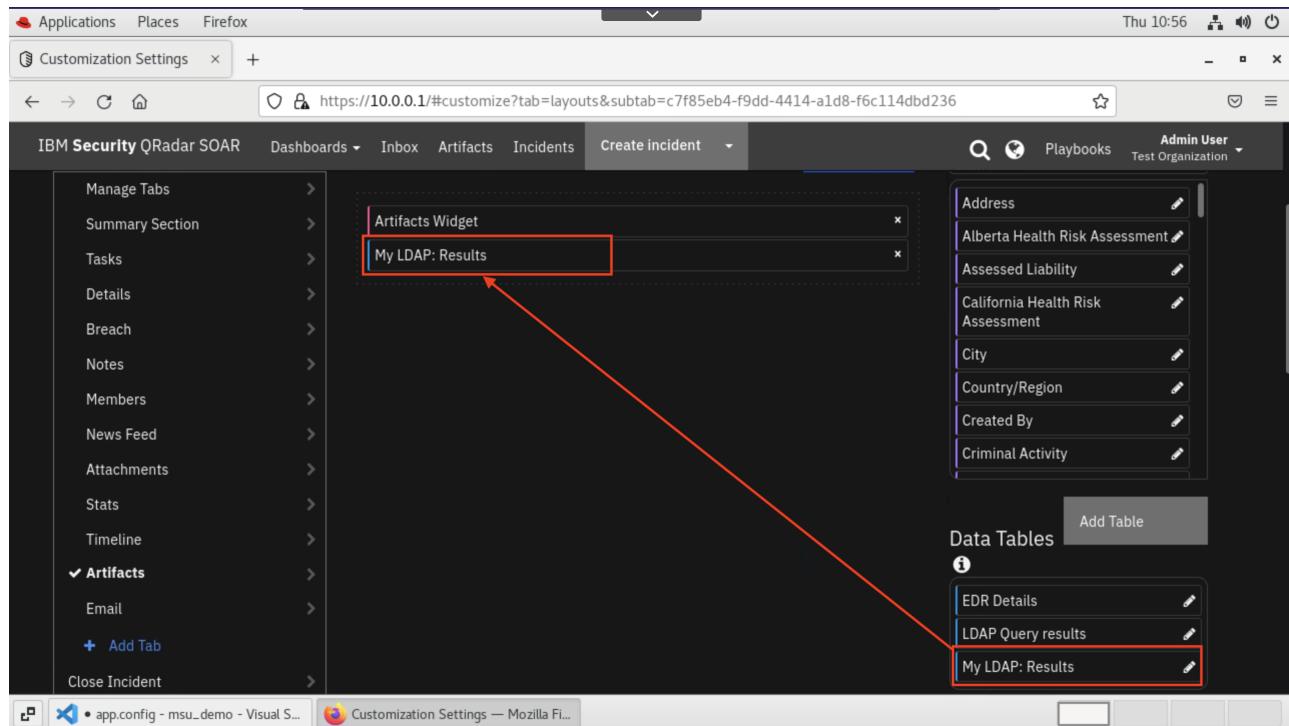
The screenshot shows the 'Customization Settings' interface in a Firefox browser. The URL is <https://10.0.0.1/#customize?tab=layouts&subtab=c7f85eb4-f9dd-4414-a1d8-f6c114dbd236>. The main panel is titled 'Configuring column (3 of 3)'. It shows the following fields:
 

- 'What type of field is this?' dropdown set to 'Text Area'.
- 'Label for the field' input field containing 'Status'.
- 'API name' input field containing 'my\_ldap\_dt\_status'.
- 'Placeholder value (Optional)' input field containing 'A placeholder value'.
- 'Requirement' dropdown set to 'Optional'.
- 'Tooltip (Optional)' input field containing 'A description of this field'.
- A checked checkbox labeled 'Add rich Text controls'.

 At the bottom right are 'Cancel', 'Back', and 'Next' buttons, with 'Next' highlighted in a red box.

**Click Next.**

- Click **Create**.
- Click and drag the newly created table to the Artifacts tab section so it will display along side the rest of the Artifacts on an Incident.

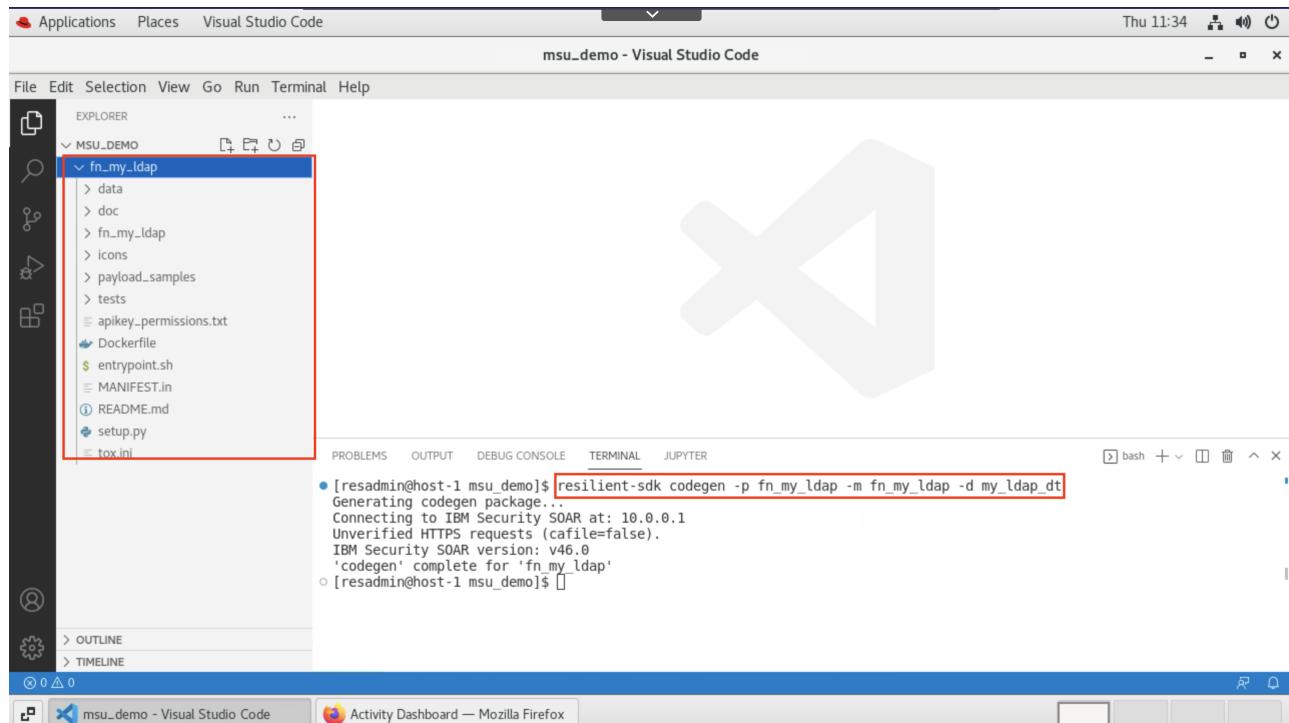


Click **Save**.

## Step 7: Create the Function Package in Code

- Switch back to VS Code. In the terminal, navigate to the directory where you want to create and save your app's package files.
- Run the `codegen` command and pull in your message destination, functions, and datatable.

```
resilient-sdk codegen -p fn_my_ldap -m fn_my_ldap -d my_ldap_dt
```



Note: we did not need to specifically mention the functions, as they are included with the message destination that they are associated with.

Note: this did necessitate that we complete step 3 where the app.config values were filled in.

## Step 8: Fill in the Code for the Function and Utility Files

- Copy in the values from the linked files below into their corresponding files in your package:

► fn\_my\_ldap/components/funct\_my\_ldap\_search.py

```
# -*- coding: utf-8 -*-

"""AppFunction implementation"""

from ldap3 import SAFE_SYNC, Connection, Server
from resilient_circuits import AppFunctionComponent, app_function,
FunctionResult
from resilient_lib import IntegrationError, validate_fields

PACKAGE_NAME = "fn_my_ldap"
FN_NAME = "my_ldap_search"

class FunctionComponent(AppFunctionComponent):
    """Component that implements function 'my_ldap_search'"""

    def __init__(self, opts):
        super(FunctionComponent, self).__init__(opts, PACKAGE_NAME)

    @app_function(FN_NAME)
    def _app_function(self, fn_inputs):
        """
        Function: None
        Inputs:
            - fn_inputs.my_ldap_search_filter
        """

        yield self.status_message("Starting App Function: '{0}'".format(FN_NAME))

        # validate app.configs
        validate_fields(["server", "port", "admin_user",
"admin_password", "search_base"], self.app_configs)

        # validate inputs
        validate_fields(["my_ldap_search_filter"], fn_inputs)

        # once fields from app.config and fn_inputs have been
        validated,
        # set them to local variables for easier use
```

```

server_address = self.app_configs.server
server_port = int(self.app_configs.port)
admin_user = self.app_configs.admin_user
admin_secret = self.app_configs.admin_password

search_filter = fn_inputs.my_ldap_search_filter
search_base = self.app_configs.search_base

# Instantiate `Server` object and establish connection
server = Server(server_address, port=server_port)
connection = Connection(server, admin_user, admin_secret,
client_strategy=SAFE_SYNC, auto_bind=True)

# Call out to search LDAP (can ignore second and fourth return
values)
status, _, response, _ = connection.search(search_base,
search_filter)

# For demo purposes, we'll only return the list of
distinguished names that were found.
# For a more robust solution, one could take other values
returned from the search functions.
# NOTE: any byte values (i.e. "raw_dn" should be filtered out)
response = [r.get("dn") for r in response]
results = {"result_list": response}

yield self.status_message("Finished running App Function:
'{0}'".format(FN_NAME))

yield FunctionResult(results, success=status)

```

► fn\_my\_ldap/components/funct\_my\_ldap\_toggle\_access.py

```

# -*- coding: utf-8 -*-

"""AppFunction implementation"""

from ldap3 import Server, Connection, SAFE_SYNC, MODIFY_REPLACE
from resilient_circuits import AppFunctionComponent, app_function,
FunctionResult
from resilient_lib import IntegrationError, validate_fields

PACKAGE_NAME = "fn_my_ldap"
FN_NAME = "my_ldap_toggle_access"

LDAP_FIELD_DESCRIPTION = "description"
LDAP_STATUS_ENABLED = "ENABLED"
LDAP_STATUS_DISABLED = "DISABLED"

class FunctionComponent(AppFunctionComponent):

```

```
"""Component that implements function
'my_ldap_toggle_user_access'"""

def __init__(self, opts):
    super(FunctionComponent, self).__init__(opts, PACKAGE_NAME)

@app_function(FN_NAME)
def _app_function(self, fn_inputs):
    """
    Function: None
    Inputs:
        - fn_inputs.my_ldap_disable
        - fn_inputs.my_ldap_enable
        - fn_inputs.my_ldap_user_dn
    """

    yield self.status_message("Starting App Function:
'{0}'".format(FN_NAME))

    # validate app.configs
    validate_fields(["server", "port", "admin_user",
"admin_password", "search_base"], self.app_configs)

    validate_fields(["my_ldap_user_dn"], fn_inputs)

    enable = getattr(fn_inputs, "my_ldap_enable", None)
    disable = getattr(fn_inputs, "my_ldap_disable", None)

    # Check that neither both are set nor neither are set.
    if enable and disable:
        raise IntegrationError("Cannot set both enable and disable
to True. Pick one.")
    elif not enable and not disable:
        raise IntegrationError("Must set one of enable and disable
to True.")

    if disable:
        change = LDAP_STATUS_DISABLED
    else:
        change = LDAP_STATUS_ENABLED

    # once fields from app.config and fn_inputs have been
    validated,
        # set them to local variables for easier use
    server_address = self.app_configs.server
    server_port = int(self.app_configs.port)
    admin_user = self.app_configs.admin_user
    admin_secret = self.app_configs.admin_password

    # Instantiate `Server` object and establish connection
    server = Server(server_address, port=server_port)
    connection = Connection(server, admin_user, admin_secret,
client_strategy=SAFE_SYNC, auto_bind=True)
```

```

        # make the modification
        connection.modify(fn_inputs.my_ldap_user_dn,
{LDAP_FIELD_DESCRIPTION: [(MODIFY_REPLACE, [change])]})

    results = {
        # return the status as "DISABLED" or "ENABLED"
        "status": change if change == LDAP_STATUS_DISABLED else
LDAP_STATUS_ENABLED
    }

    yield self.status_message("Finished running App Function:
'{0}'".format(FN_NAME))

    yield FunctionResult(results)

```

► fn\_my\_ldap/util/config.py

```

# -*- coding: utf-8 -*-

"""Generate a default configuration-file section for fn_my_ldap"""

def config_section_data():
    """
    Produce add the default configuration section to app.config,
    for fn_my_ldap when called by `resilient-circuits config [-c|-u]`"
    """

    config_data = u"""\n[fn_my_ldap]\nserver=<ldap_server_here>\nport=<ldap_port_here>\nadmin_user=<admin_user_dn>\nadmin_password=<admin_user_secret>\nsearch_base=ou=people,dc=planetexpress,dc=com\n"""

    return config_data

```

► fn\_my\_ldap/util/selftest.py

```

# -*- coding: utf-8 -*-

"""
Function implementation test.
Usage:
    resilient-circuits selftest -l fn-my-ldap
    resilient-circuits selftest --print-env -l fn-my-ldap

Return examples:

```

```
    return {
        "state": "success",
        "reason": "Successful connection to third party endpoint"
    }

    return {
        "state": "failure",
        "reason": "Failed to connect to third party endpoint"
    }
....
```

```
import logging

from ldap3 import Server, Connection, SAFE_SYNC
from resilient_lib import validate_fields

log = logging.getLogger(__name__)
log.setLevel(logging.INFO)
log.addHandler(logging.StreamHandler())


def selftest_function(opts):
    ....
    Placeholder for selftest function. An example use would be to test
    package api connectivity.
    Suggested return values are be unimplemented, success, or failure.
    ....
    app_configs = opts.get("fn_my_ldap", {})

    try:
        validate_fields(["server", "port", "admin_user",
"admin_password", "search_base"], app_configs)

        server = Server(app_configs.get("server"),
port=int(app_configs.get("port")))
        connection = Connection(server, app_configs.get("admin_user"),
app_configs.get("admin_password"), client_strategy=SAFE_SYNC,
auto_bind=True)

        return {
            "state": "success",
            "reason": None
        }
    except Exception as e:
        return {
            "state": "failure",
            "reason": str(e)
        }
```

► fn\_my\_ldap/setup.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from setuptools import setup, find_packages
import glob
import ntpath

def get_module_name(module_path):
    """
    Return the module name of the module path
    """
    return ntpath.split(module_path)[1].split(".")[0]

def snake_to_camel(word):
    """
    Convert a word from snake_case to CamelCase
    """
    return ''.join(x.capitalize() or '_' for x in word.split('_'))

setup(
    name="fn_my_ldap",
    display_name="LDAP Functions for SOAR",
    version="1.0.0",
    license="MIT",
    author="IBM SOAR",
    author_email="",
    url="ibm.com",
    description="App to Search, and enable/disable users in LDAP system",
    long_description="""App provides functionality to search for users, enable users, and disable users in a LDAP system.""",
    install_requires=[
        "resilient-circuits>=46.0.0",
        "ldap3"
    ],
    python_requires='>=3.6',
    packages=find_packages(),
    include_package_data=True,
    platforms="any",
    classifiers=[
        "Programming Language :: Python",
    ],
    entry_points={
        "resilient.circuits.components": [
            # When setup.py is executed, loop through the .py files in the components directory and create the entry points.
            "{}FunctionComponent = fn_my_ldap.components.{}".format(snake_to_camel(get_module_name(filename)),
            get_module_name(filename)) for filename in
            glob.glob("./fn_my_ldap/components/[a-zA-Z]*.py")]
    }
)
```

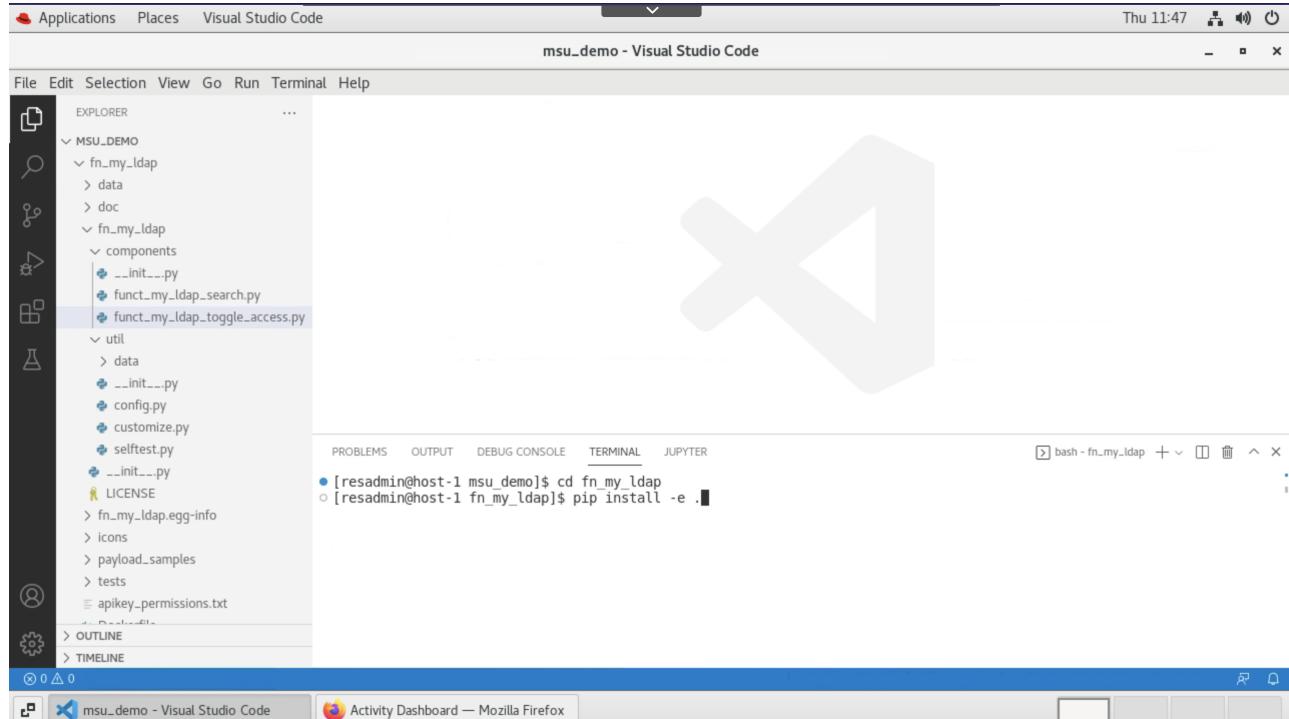
```

        ]
        ,
        "resilient.circuits.configsection": ["gen_config =
fn_my_ldap.util.config:config_section_data"],
        "resilient.circuits.customize": ["customize =
fn_my_ldap.util.customize:customization_data"],
        "resilient.circuits.selftest": ["selftest =
fn_my_ldap.util.selftest:selftest_function"]
    }
)

```

- Pip install the app in editable mode:

```
cd fn_my_ldap
pip install -e .
```



Note: this does *not* have to be done each time a change is made in the code as the package was installed in **editable** mode, however, it *does need* to be redone each time a change is made in the *setup.py* file; all other files' changes will be reflected automatically.

- Create a config section and populate it with data:

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

Then open the app.config file in VS Code

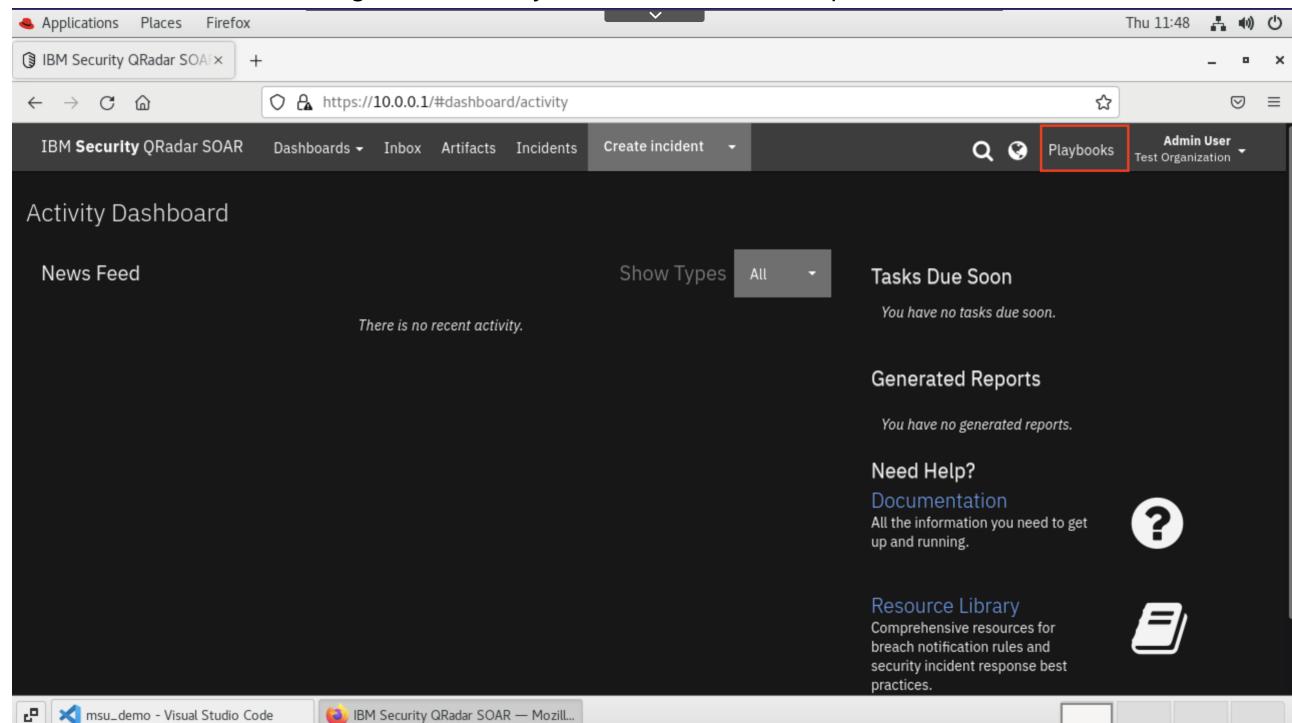
```
code ~/.resilient/app.config
```

Fill in the following values for the [fn\_my\_ldap] section:

```
[fn_my_ldap]
server=10.0.0.1
port=10389
admin_user=cn=admin,dc=planetexpress,dc=com
admin_password=GoodNewsEveryone
search_base=ou=people,dc=planetexpress,dc=com
```

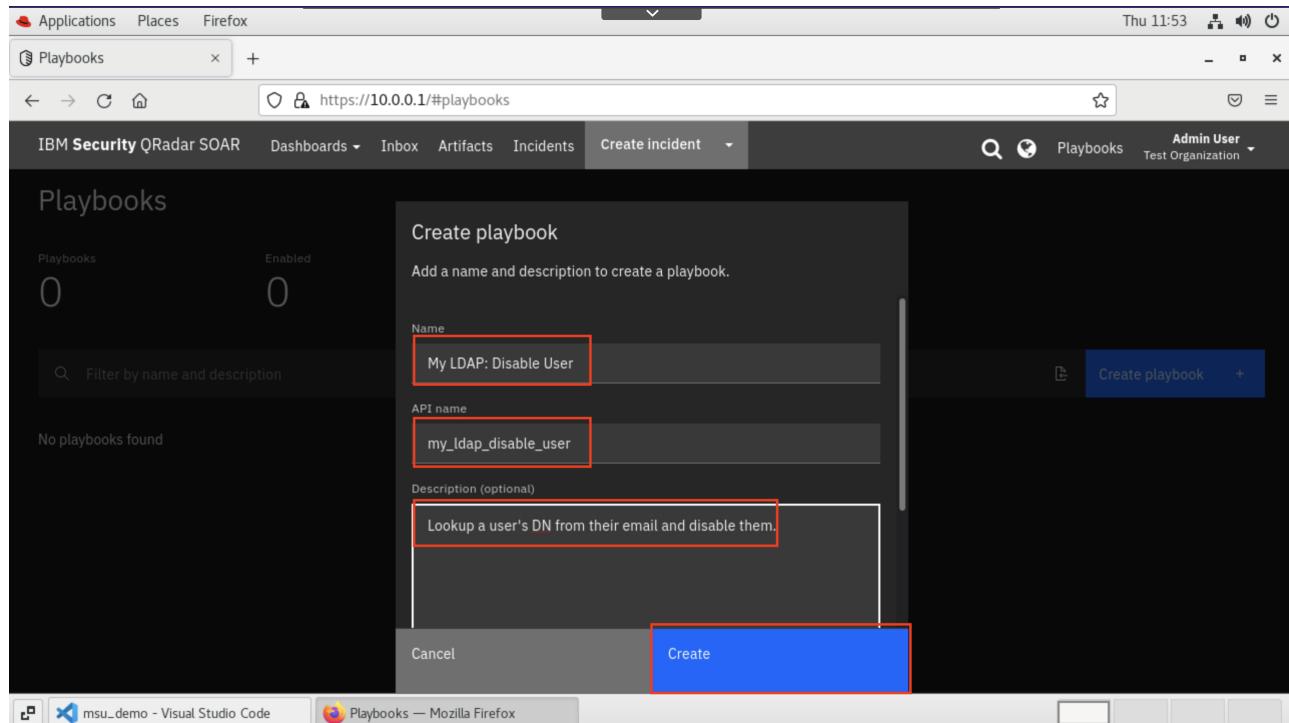
## Step 9: Create a Playbook to Test

- Switch back to SOAR. Navigate to the Playbooks interface of the platform:

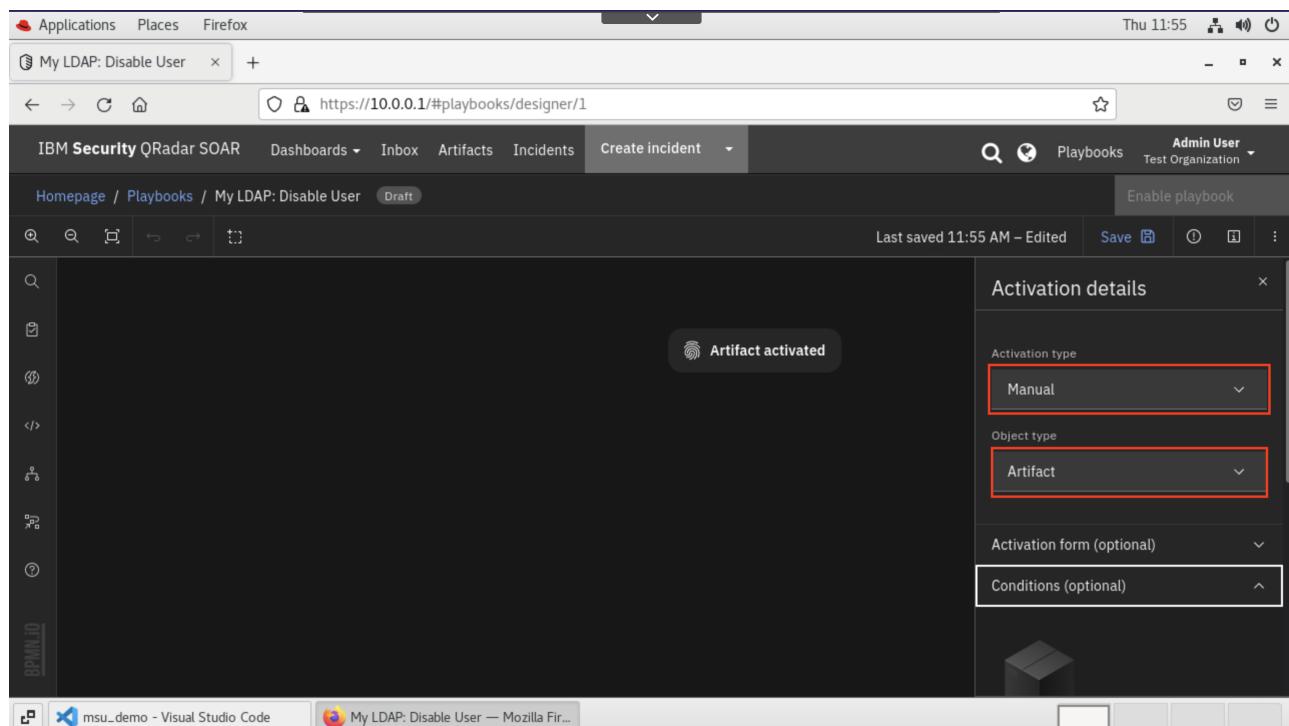


- Click **Create Playbook**.

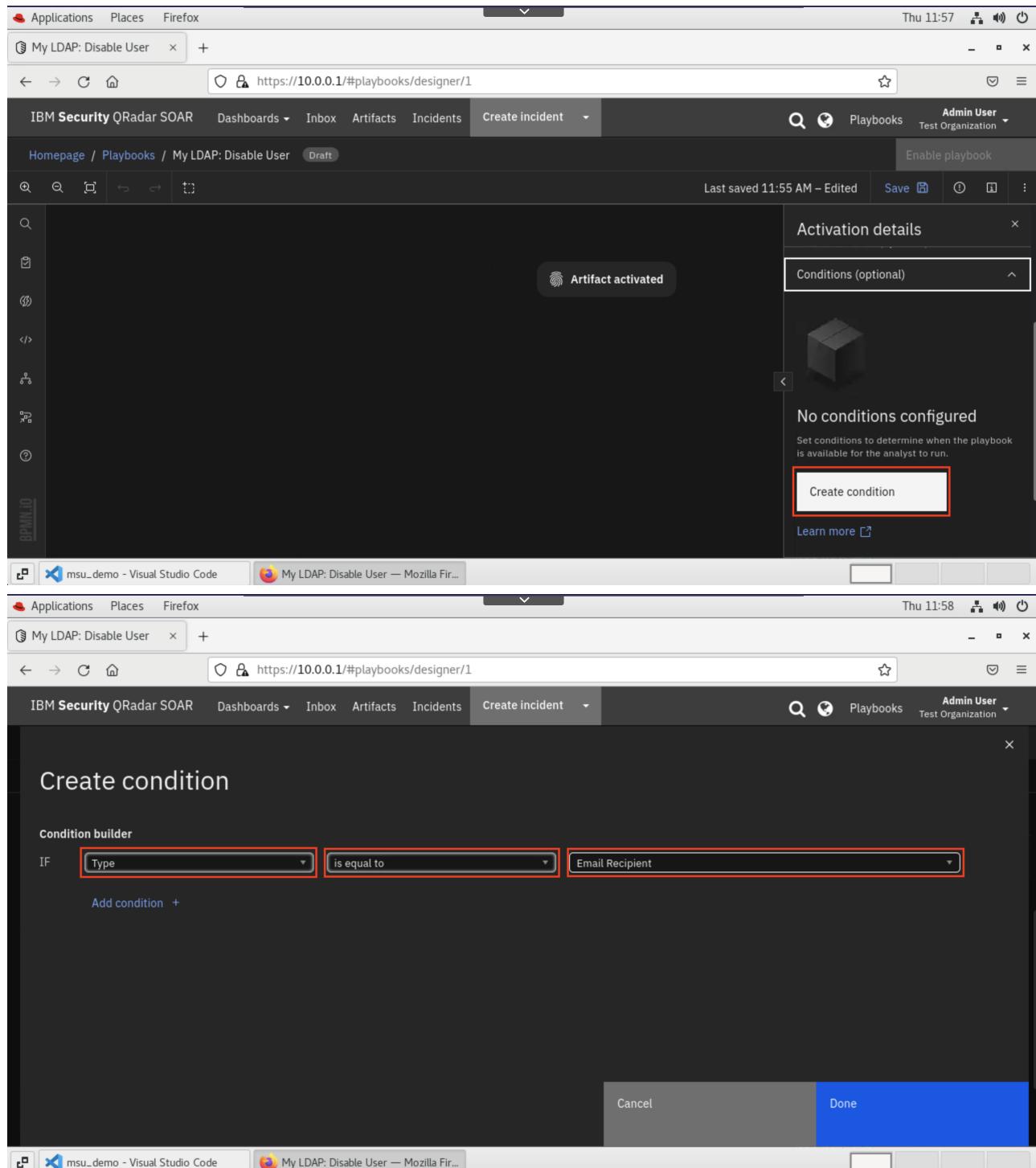
- Call it "My LDAP: Disable User":



- Click **Create**.
- Set the **Activation Type** to **Manual** and the **Object** to **Artifact** as we'll be manually running this on email artifacts.

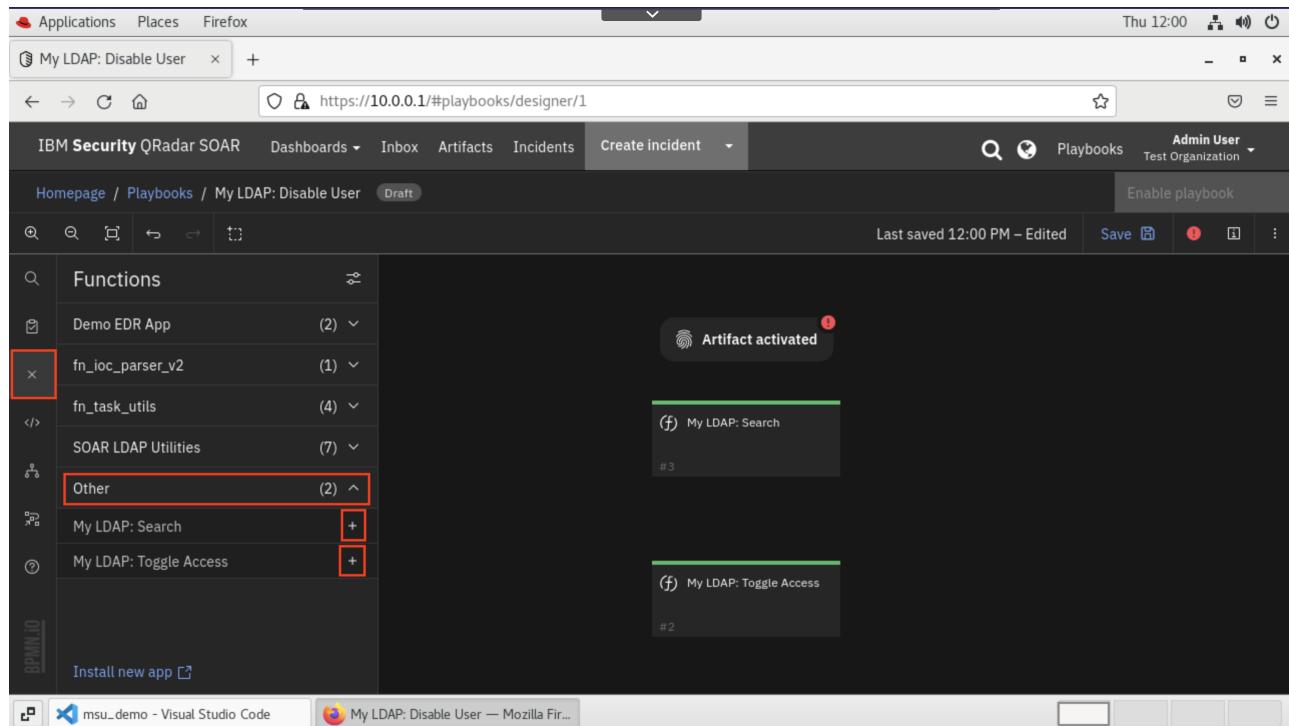


- Create an activation condition so that this only activates on **Email Recipient** artifacts. This could be changed to suit your needs. For demo purposes, we'll stick with just the one condition.



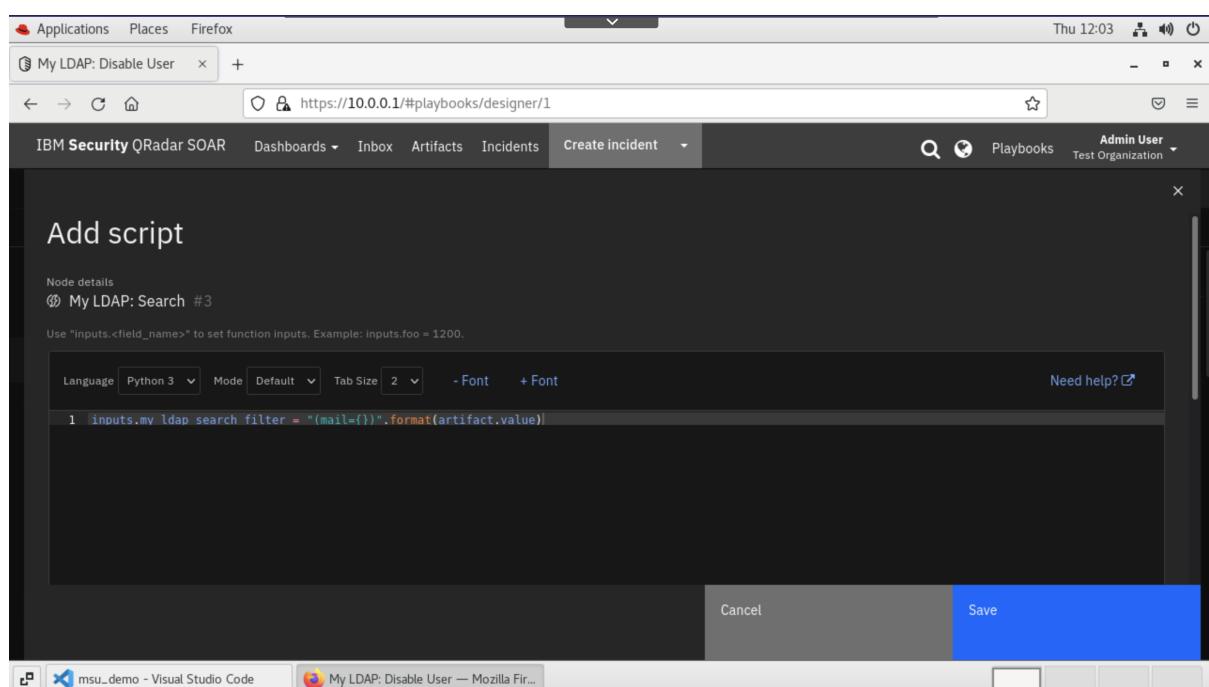
Click **Done**.

- Expand the Function panel on the left. Under **Other**, add both the functions we made previously to the canvas:



- Now we'll fill out the details for each function.
  - For the Search function, add the following input script:

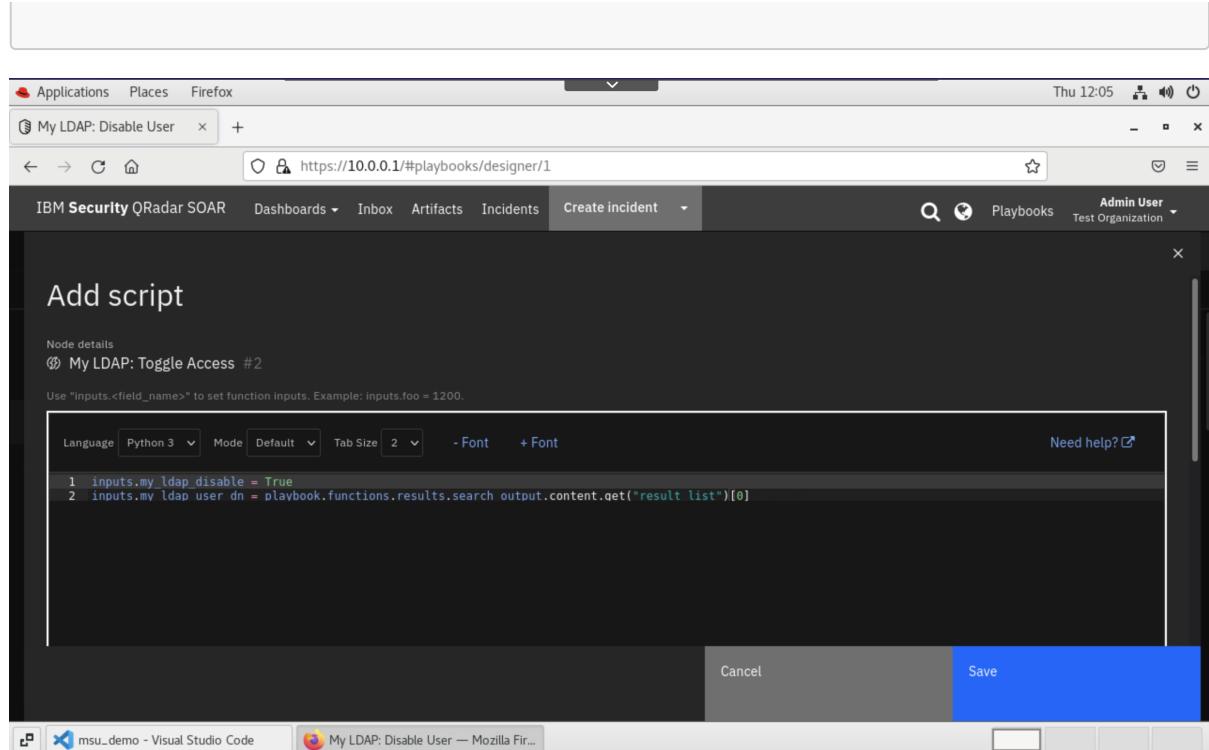
```
inputs.my_ldap_search_filter = "(mail={})".format(artifact.value)
```



Click **Save**.

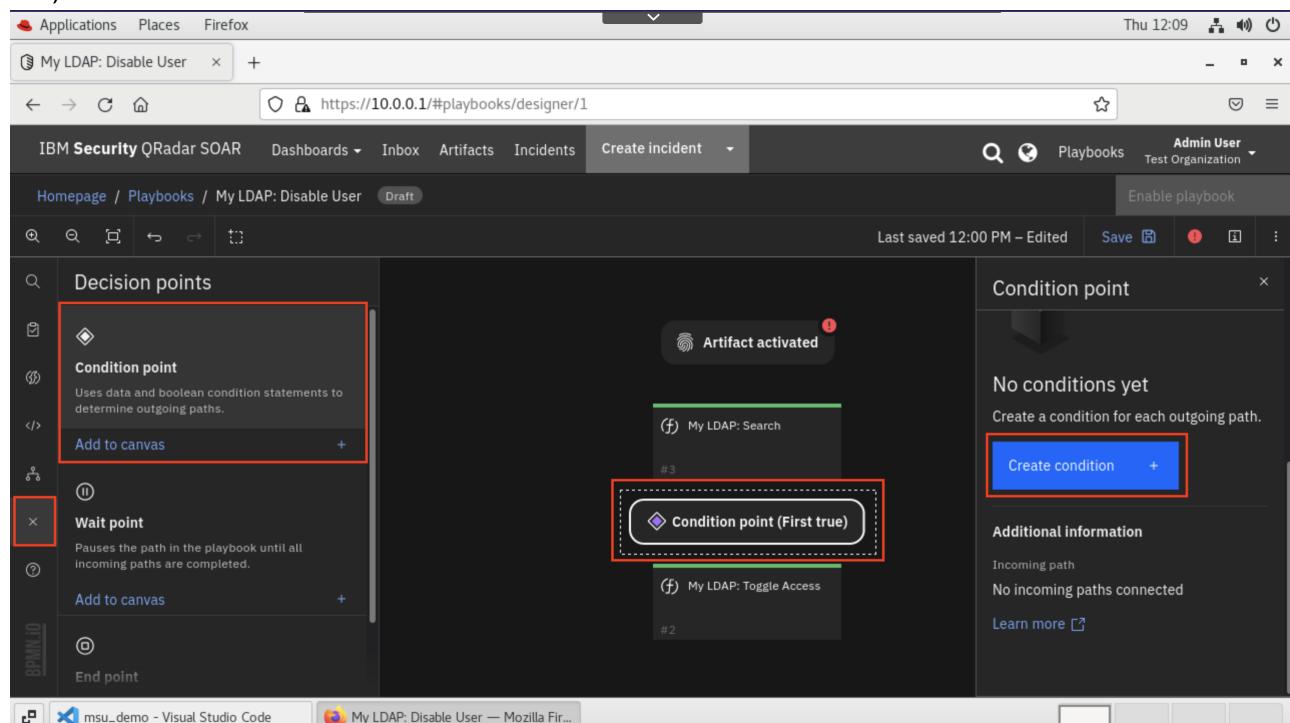
- Name the output of the Search function **search\_output**.
- For the Toggle function, add the following input script:

```
inputs.my_ldap_disable = True
inputs.my_ldap_user_dn =
playbook.functions.results.search_output.content.get("result_list"
") [0]
```



**Click Save.**

- Name the output of the Toggle function **toggle\_output**.
- Now we need to build the flow of the playbook. There needs to be a decision point where based on the output of the first function, we run the second function. If there was no Distinguished Name found in the search function, we don't want to run the toggle function. To accomplish this we will add a Condition Point to the canvas (which can be found by expanding the Decision Points panel on the left).



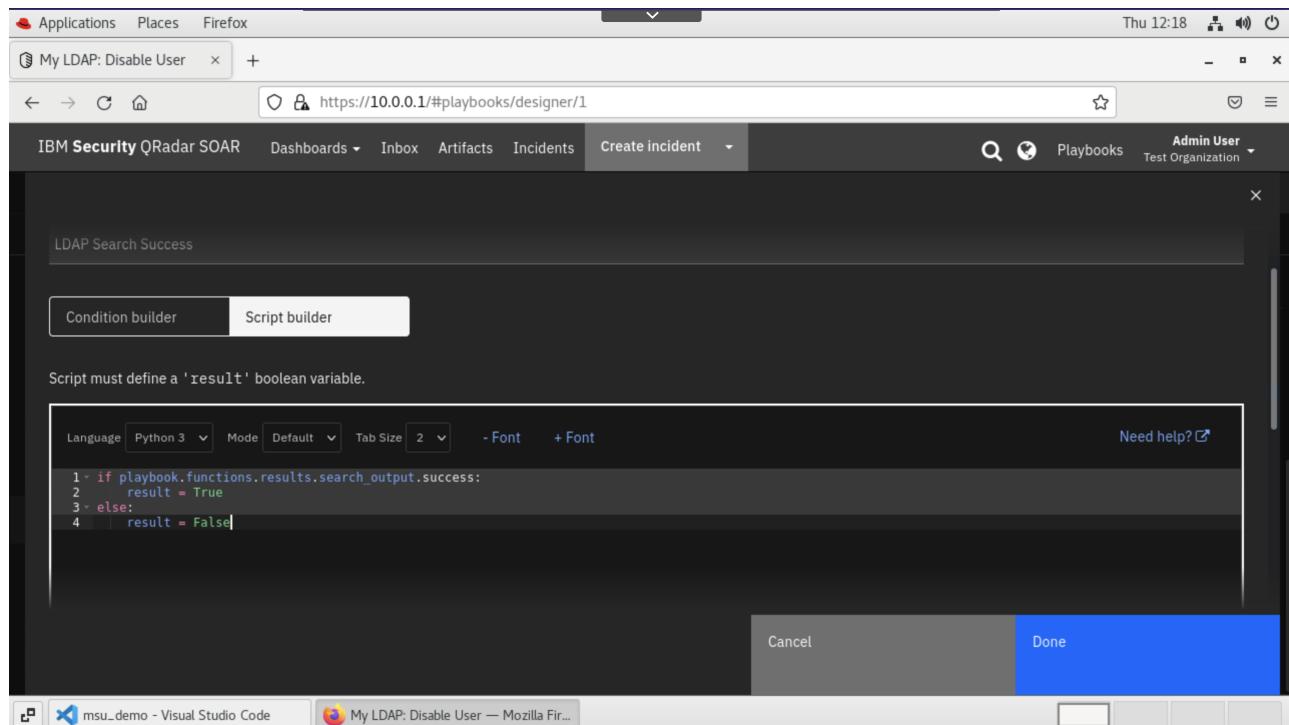
- Click **Create Condition**. Name the condition "LDAP Search Success" and switch to the **Script Builder** condition option. Enter the following script as the condition:

```

if playbook.functions.results.search_output.success:
    result = True

```

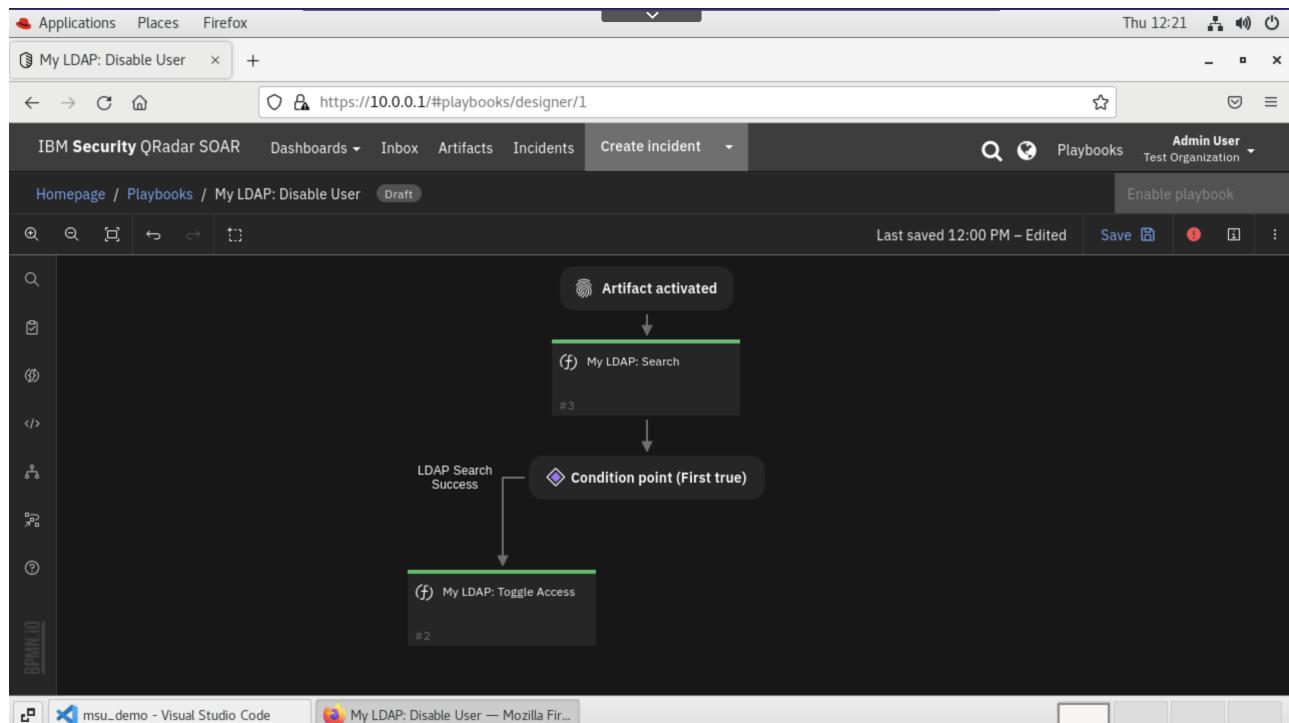
```
else:
    result = False
```



Click **Done**.

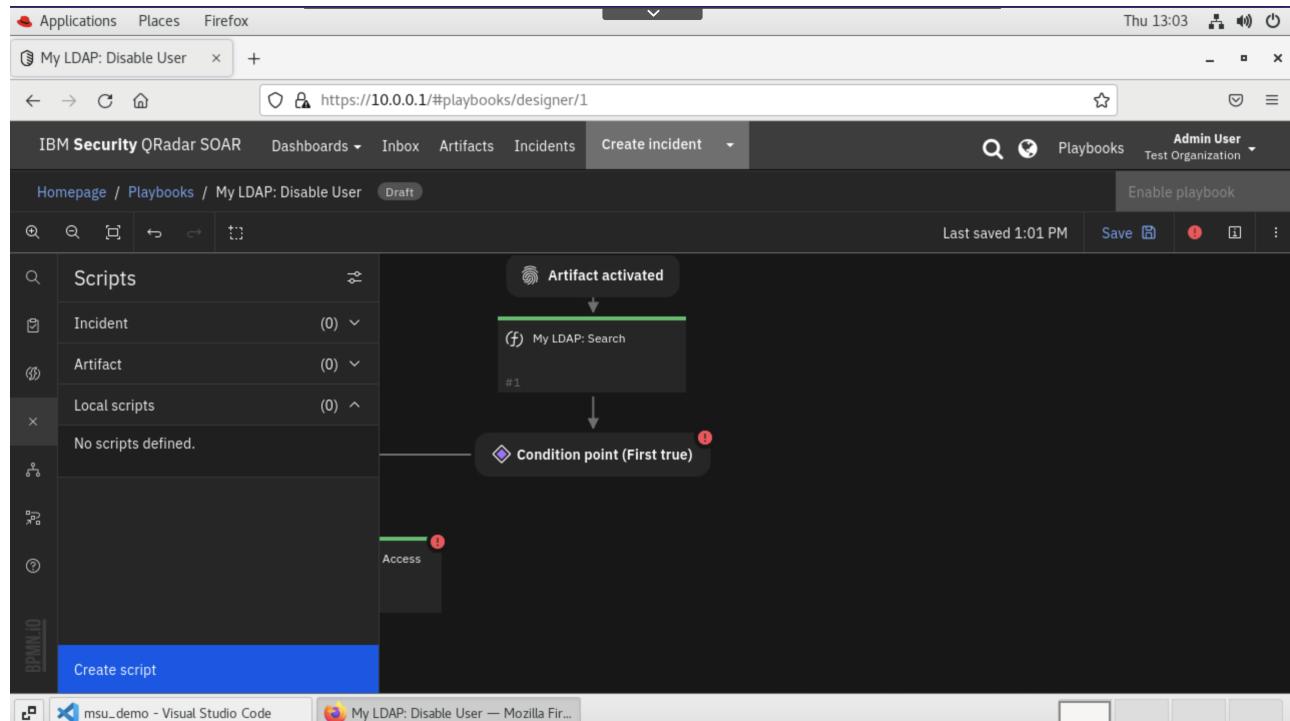
**Note:** **result** must be defined for all branches of the condition to properly function.

- Now we must connect the paths of the playbook. To do that, click the vertex you wish to connect *from*, and drag from one of the blue dots to a blue dot on the vertex you wish to connect *to*. Repeat until it looks like this:



- Now we must deal with two things: how to end the playbook in our two branches that we've created.

- So we'll create two local scripts to handle the two possible options.



- Local Script #1: **My LDAP: Toggle Success**

```
results = playbook.functions.results.toggle_output

if results.success:
    row = incident.addRow("my_ldap_dt")

    row.my_ldap_dt_dn = results.inputs.get("my_ldap_user_dn")
    row.my_ldap_dt_user = artifact.value
    row.my_ldap_dt_status = '<div'
    style="color:#e60000">' + results.content.get("status") + '</div>'
```

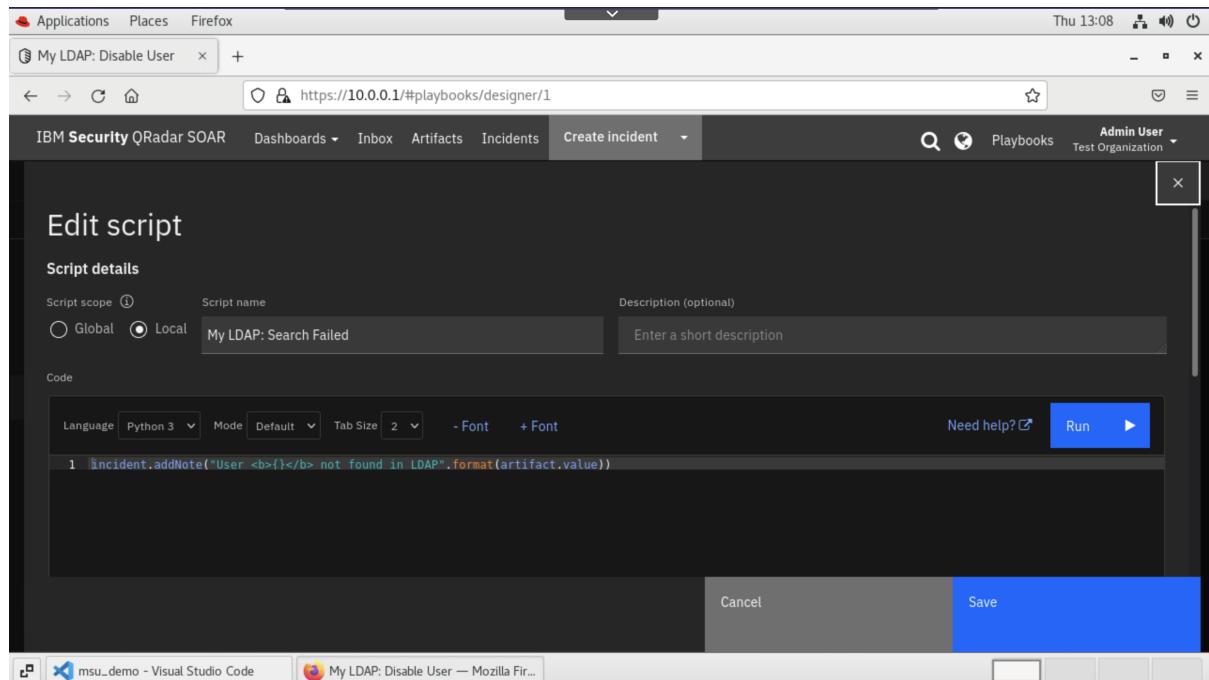
The screenshot shows the "Edit script" dialog for the "My LDAP: Toggle Success" script. The "Script details" section shows "Script scope: Local" and "Script name: My LDAP: Toggle Success". The "Code" section contains the Python script code:

```
1 results = playbook.functions.results.toggle_output
2
3 if results.success:
4     row = incident.addRow("my_ldap_dt")
5
6     row.my_ldap_dt_dn = results.inputs.get("my_ldap_user_dn")
7     row.my_ldap_dt_user = artifact.value
8     row.my_ldap_dt_status = '<div style="color:#e60000">' + results.content.get("status") + '</div>'
```

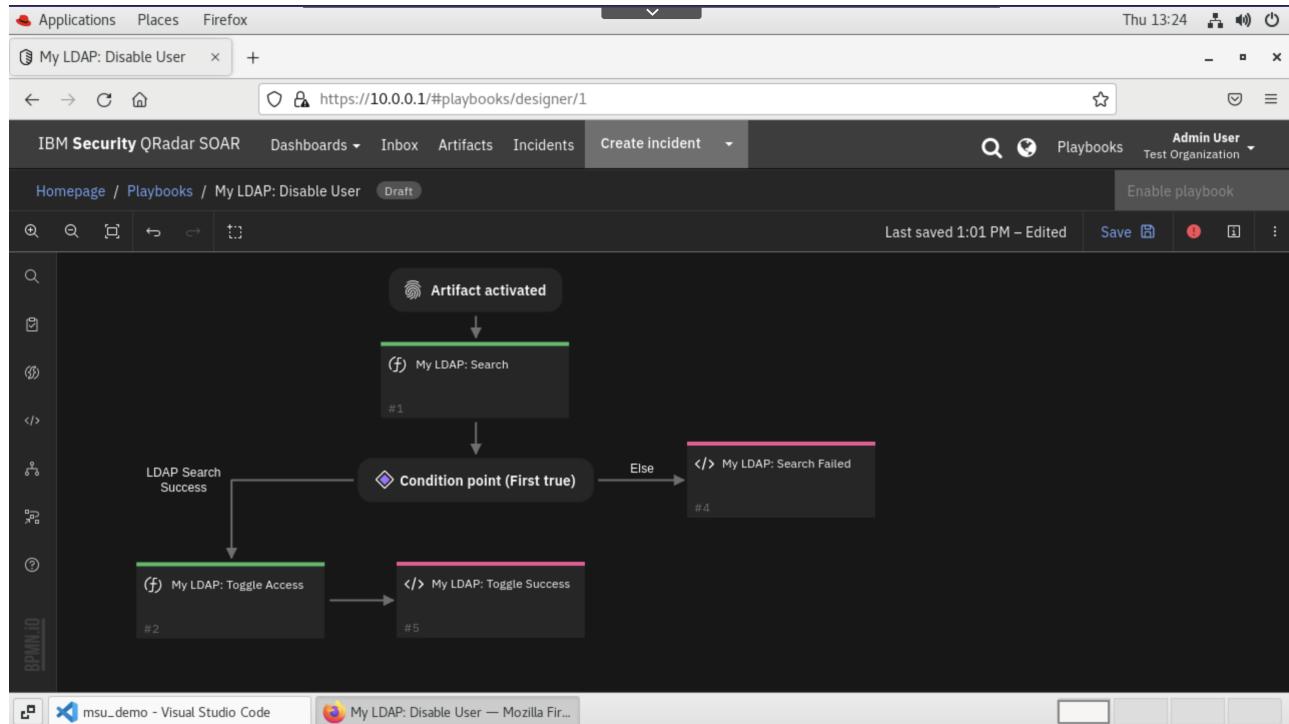
The "Run" button is highlighted in the top right corner of the code editor.

- Local Script #2: **My LDAP: Search Failed**

```
incident.addNote("User <b>{ }</b> not found in
LDAP".format(artifact.value))
```

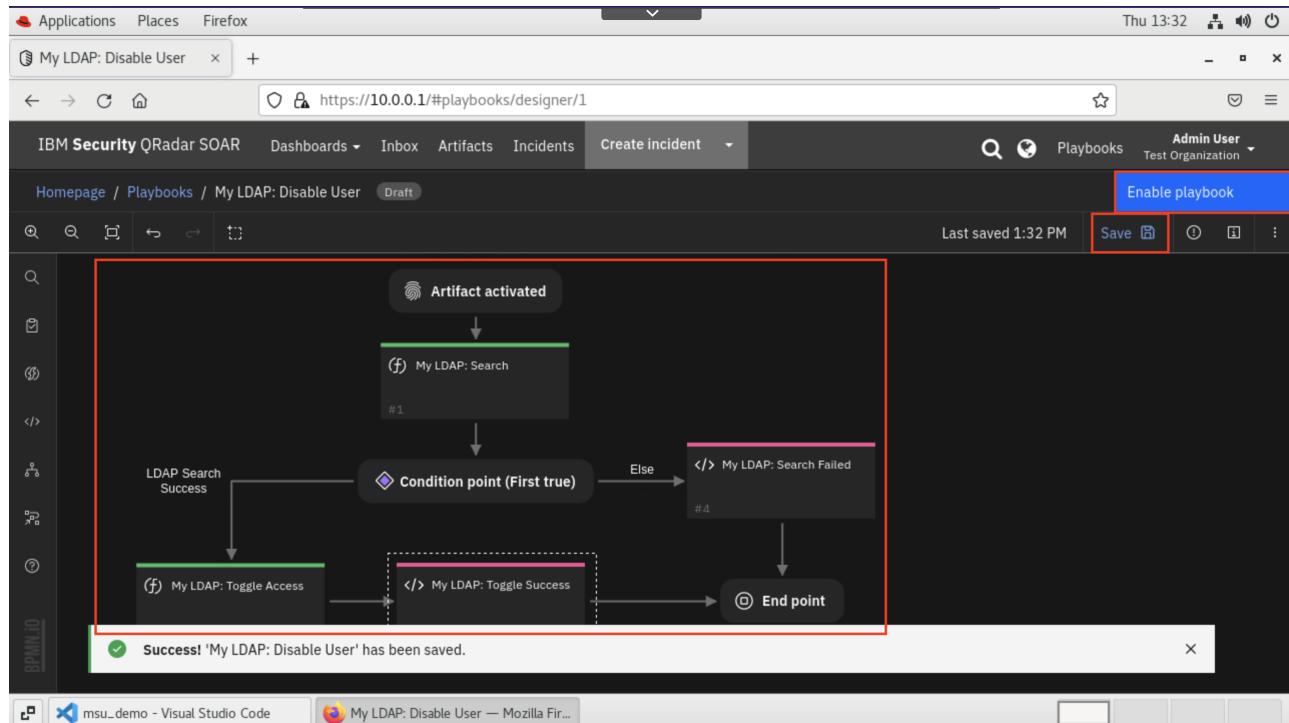


- Add the two scripts to the canvas and attach to the rest of the playbook as shown:



- Finally, add an endpoint (found in the decision points panel) and connect the two scripts to it.

- Click Save and Enable Playbook.



## Step 10: Start *resilient-circuits* Server

- Switch back to VS Code and the terminal. In the terminal, we'll want to run `codegen` to reload and bring in the playbook (find the API Name in the info section of the Playbook) and any other changes we made in the UI.

```
resilient-sdk codegen -p . --reload -pb my_ldap_disable_user
```

The screenshot shows the Visual Studio Code interface with a terminal window open. The terminal shows the command `resilient-sdk codegen -p . --reload -pb my_ldap_disable_user` being run. The output of the command is displayed, including the generation of a codegen package, connecting to the IBM Security SOAR server, and the completion of the codegen for the `fn_my_ldap` component. The terminal window is titled "msu\_demo - Visual Studio Code".

Note: you should be generous in running `codegen --reload` as much as you want. It will never hurt, and it can be painful if you forget to reload any UI changes you've made.

- Now we'll start our `resilient-circuits` server to allow the SOAR platform to pick up our code for the functions that we've written:

`resilient-circuits run`

The screenshot shows a Linux terminal window within Visual Studio Code. The command `[resadmin@host-1 fn_my_ldap]$ resilient-circuits run` has been entered and is running. A large gray X is overlaid on the terminal window, indicating it is unresponsive or failing.

```
[resadmin@host-1 fn_my_ldap]$ resilient-circuits run
```

Keep any eye out for "resilient-circuits has started successfully and is now running..."

The screenshot shows the same terminal window after the command has completed. The output now includes the message "resilient-circuits has started successfully and is now running...". This message is highlighted with a red rectangle.

```
2022-09-08 13:39:43,501 INFO [actions_component] STOMP attempting to connect
2022-09-08 13:39:43,502 INFO [stomp_component] Connect to Stomp...
2022-09-08 13:39:43,503 INFO [client] Connecting to 10.0.0.1:65001 ...
2022-09-08 13:39:43,597 INFO [client] Connection established
2022-09-08 13:39:43,779 INFO [client] Connected to stomp broker [session=ID:host-1-36126-1662640637932-4:5, version=1.2]
2022-09-08 13:39:43,780 INFO [stomp_component] Connected to failover:(ssl://10.0.0.1:65001)?maxReconnectAttempts=3,startupMaxReconnectAttempts=3
2022-09-08 13:39:43,781 INFO [stomp_component] Client HB: 0 Server HB: 15000
2022-09-08 13:39:43,781 INFO [stomp_component] No Client heartbeats will be sent
2022-09-08 13:39:43,781 INFO [stomp_component] Requested heartbeats from server.
2022-09-08 13:39:43,784 INFO [actions_component] STOMP connected
2022-09-08 13:39:43,887 INFO [actions_component] resilient-circuits has started successfully and is now running...
2022-09-08 13:39:43,887 INFO [actions_component] Subscribe to message destination 'fn_my_ldap'
2022-09-08 13:39:43,888 INFO [stomp_component] Subscribe to message destination actions.201.fn_my_ldap
```

Note: this will only work properly if you already used `pip` to install the app as at the end of step 8.

## Step 11: Test

- Return to the UI and exit out of the playbook. Create a new incident with default values and add an Email Recipient artifact to the incident with value `hubert@planetexpress.com`.

The screenshot shows the IBM Security QRadar SOAR web interface. A modal window titled "Add Artifact" is open, showing the "Type" dropdown set to "Email Recipient". The "Value" field contains the email address "hubert@planetexpress.com", which is highlighted with a red box. The "Create" button at the bottom right of the modal is also highlighted with a red box. In the background, the main dashboard shows an incident titled "Test My LDAP" with a status of "Initial". The "Summary" section on the right provides details about the incident, including its ID (2096), phase (Initial), severity (Low), and date created (09/08/2022 13:43). The "People" section shows it was created by an Admin User.

- Select the action **My LDAP: Disable User** from the list of available actions for the artifact.

The screenshot shows the IBM Security QRadar SOAR interface. An artifact table is displayed, showing one item with the value "hubert@planetexpress.com". The "Actions" column for this item has a dropdown menu open, with the option "My LDAP: Disable User" highlighted with a red box. The "People" section on the right shows the artifact was created by an Admin User and has an owner listed as Admin User. The "Related Incidents" section indicates "No related incidents".

- This will kick off the playbook and two functions. If you switch back to VS Code, you should see the activity in the logs in `resilient-circuits`:

```

2022-09-08 14:00:16,031 INFO [client] Connected to stomp broker [session=ID:host-1-36126-1662640637932-4:7, version=1.2]
2022-09-08 14:00:16,032 INFO [stomp_component] Connected to failover:(ssl://10.0.0.1:65001)?maxReconnectAttempts=3,startupMaxReconnectAttempts=3
2022-09-08 14:00:16,137 INFO [stomp_component] Client HB: 0 Server HB: 15000
2022-09-08 14:00:16,032 INFO [stomp_component] No Client heartbeats will be sent
2022-09-08 14:00:16,032 INFO [stomp_component] Requested heartbeats from server.
2022-09-08 14:00:16,034 INFO [actions_component] STOMP connected.
2022-09-08 14:00:16,136 INFO [actions_component] resilient-circuits has started successfully and is now running...
2022-09-08 14:00:16,137 INFO [actions_component] Subscribe to message destination actions.201.fn_my_ldap
2022-09-08 14:00:16,137 INFO [stomp_component] Subscribe to message destination actions.201.fn_my_ldap
2022-09-08 14:00:36,994 INFO [actions_component] Event: <my_ldap_search[] (id=15, workflow=playbook_bfd1a07c_713d_48f3_8d1e_f9eac6bd721a, user=admin@example.com) 2022-09-08 21:00:36.691000> Channel: functions.my_ldap_search
2022-09-08 14:00:37,197 INFO [decorators] [my_ldap_search] Validated function inputs
2022-09-08 14:00:37,199 INFO [decorators] [my_ldap_search] StatusMessage: Starting App Function: 'my_ldap_search'
2022-09-08 14:00:37,237 INFO [decorators] [my_ldap_search] StatusMessage: Finished running App Function: 'my_ldap_search'
2022-09-08 14:00:37,238 INFO [decorators] [my_ldap_search] Returning results
2022-09-08 14:00:39,665 INFO [decorators] Event: <my_ldap_toggle_access[] (id=16, workflow=playbook_bfd1a07c_713d_48f3_8d1e_f9eac6bd721a, user=4b2543f1-fb4e-49f7-aabb-f55a6655b612) 2022-09-08 21:00:39.389000> Channel: functions.my_ldap_toggle_access
2022-09-08 14:00:39,869 INFO [decorators] [my_ldap_toggle_access] Validated function inputs
2022-09-08 14:00:39,870 INFO [decorators] [my_ldap_toggle_access] StatusMessage: Starting App Function: 'my_ldap_toggle_access'
2022-09-08 14:00:39,950 INFO [decorators] [my_ldap_toggle_access] StatusMessage: Finished running App Function: 'my_ldap_toggle_access'
2022-09-08 14:00:39,951 INFO [decorators] [my_ldap_toggle_access] Returning results

```

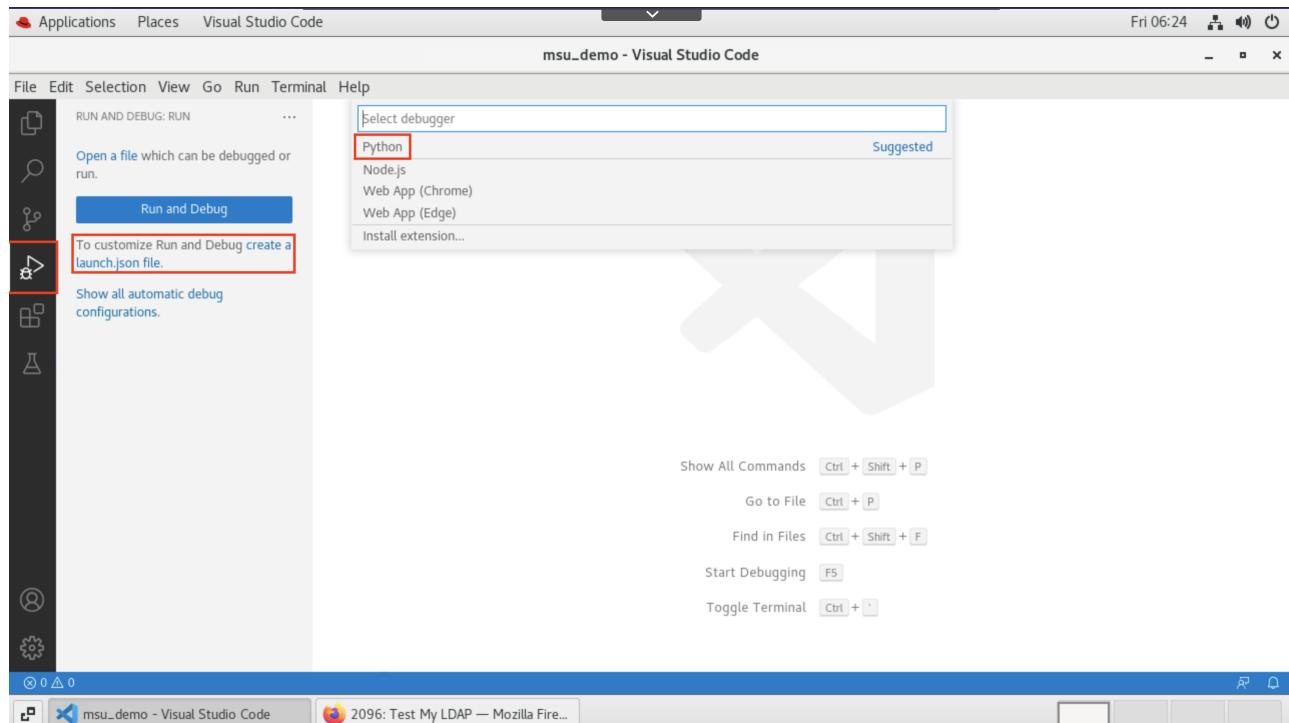
- Refresh the artifacts tab, and you should see the following in the datatable:

Distinguished Name	User	Status
cn=Hubert J. Farnsworth,ou=people,dc=planetexpress,dc=com	hubert@planetexpress.com	DISABLED

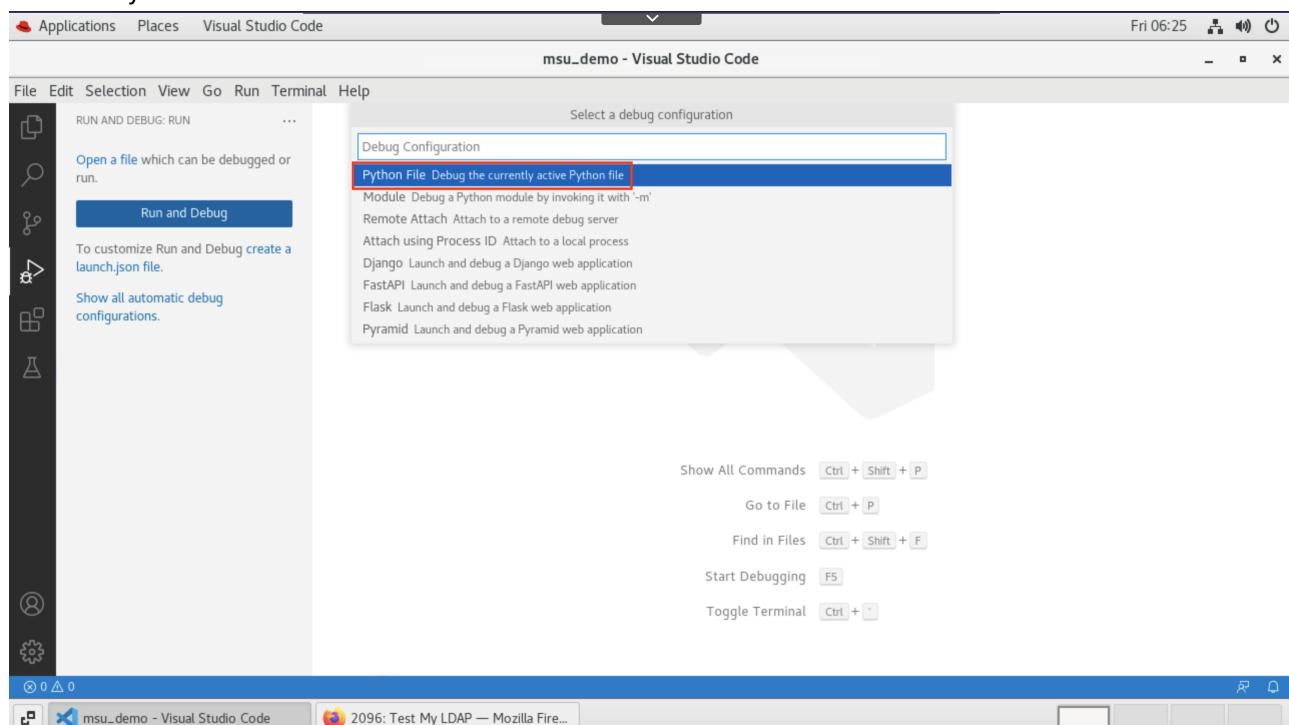
- To test the other path where a note is posted when the email is not found, enter a new artifact with a random value for the email recipient and run the playbook again. Notice that nothing is added to the datatable and a note is added to the incident.

## Step 12: Debug **resilient-circuits** Server

- Stop **resilient-circuits** (CTRL + C).
- On the left of the VS Code panel, select "Run + Debug". Then click "create a launch.json file" and select "Python" from the list of debuggers:



- Select "Python File":



- In the newly created `launch.json` file, enter the details as seen in the screenshot below:

```

{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Python: resilient-circuits",
      "type": "python",
      "request": "launch",
      "program": "/home/resadmin/.local/lib/python3.6/site-packages/resilient_circuits/bin/resilient_circuits_cmd.py",
      "args": ["run", "--loglevel=DEBUG"],
      "console": "integratedTerminal",
      "justMyCode": true
    }
  ]
}
  
```

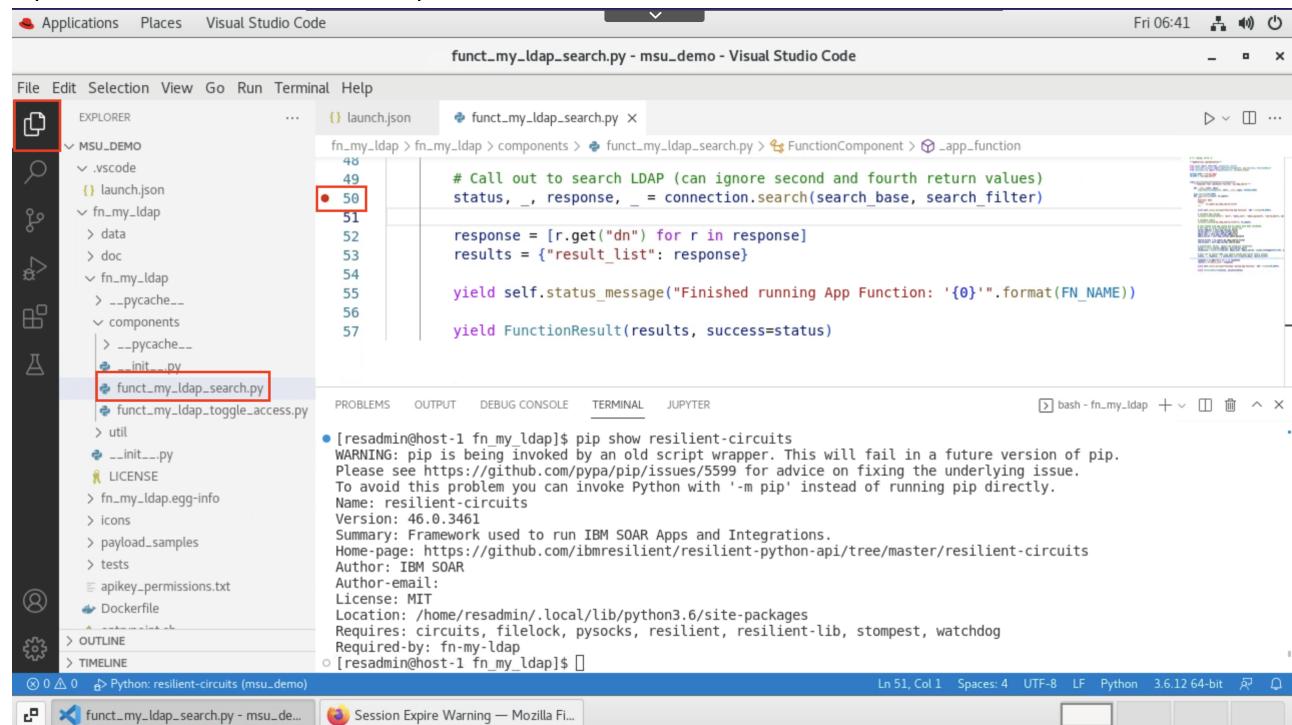
The program path is found by running `pip show resilient-circuits` and taking the location value and appendeding `resilient_circuits/bin/resilient_circuits_cmd.py`.

Be sure to add the `args` section to the launch file.

- In the Debug pannel, click the Play button:

This launches the program as defined in the `launch.json` file and starts circuits with loglevel DEBUG in a new interactive terminal.

- Open a function and set a breakpoint:



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under 'MSU\_DEMO'. A red box highlights the 'Breakpoints' icon (a small square with a dot) in the sidebar.
- Code Editor:** Displays the file 'func\_my\_ldap\_search.py'. Line 50 is highlighted with a red box and has a breakpoint marker (a red circle with a white dot).
- Terminal:** Shows a pip command being run: '[resadmin@host-1 fn\_my\_ldap]\$ pip show resilient-circuits'. It includes a warning about pip being invoked by an old script wrapper.
- Status Bar:** Shows the current file is 'func\_my\_ldap\_search.py - msu\_demo - Visual Studio Code', the line is 'Ln 51, Col 1', and the terminal mode is 'Python 3.6.12 64-bit'.

- From the UI, run the playbook and notice that the breakpoint is hit. You can now step around in the function and see session variables.
- Feel free to play around with this, it can be a very powerful functionality.

## Step 13: Validate

- The SDK has the capability to validate the work that you've done. It will statically check as well as dynamically check your code for potential missing values, security vulnerabilities, and more. The validate tool can also run selftest and verify your local configuration details.
- We'll run the validate tool for our app (static validation only by passing the `--validate` flag):

```
resilient-sdk validate -p . --validate
```

- This should fail as we haven't had the time to fill out all of the details.

```

msu_demo - Visual Studio Code
File Edit Selection View Go Run Terminal Help
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
package files FAIL
-----
Validating payload samples
-----
CRITICAL 'output_json_example.json' and 'output_json_schema.json' for 'my_ldap_search' empty
Fill in values manually or by using "'resilient-sdk codegen -p /home/resadmin/msu_demo/fn_my_ldap --gather-results'"
CRITICAL 'output_json_example.json' and 'output_json_schema.json' for 'my_ldap_toggle_access' empty
Fill in values manually or by using "'resilient-sdk codegen -p /home/resadmin/msu_demo/fn_my_ldap --gather-results'"
payload samples FAIL
-----
Validation Results
-----
Critical Issues: 4
Warnings: 1
Validations Passed: 20
See the detailed report at /home/resadmin/msu_demo/fn_my_ldap/dist/validate_report.md
[resadmin@host-1 fn_my_ldap]$ █
0 0 Python: resilient-circuits (msu_demo)
msu_demo - Visual Studio Code IBM Security QRadar SOAR — Mozilla Firefox resadmin@host-1:~
```

- When developing your own custom apps, if you don't have a passing validation report, your submission to our App Exchange will be automatically rejected.
- The other options available for validate are
  - tests, --pylint, --bandit, --selftest**
  - If you run the SDK in verbose mode, you will see the output of the passing validations:

```
resilient-sdk -v validate -p .
```

- Run **resilient-sdk validate -h** for more details.
- The validation tool outputs the report of the run to the terminal, but also includes the report in a markdown format in a file located at **fn\_my\_ldap/dist/validate\_report.md**

## Step 14: Package

- Next we'll package the app for distribution.
- Before packaging, make sure to reload the package just in case there have been changes in the UI that haven't been captured yet:

```
resilient-sdk codegen --reload -p .
```

- Run the package command, skipping payload samples and specifying a custom repository name (this is optional; the default value is **ibmresilient**):

```
resilient-sdk package -p . --no-samples --repository-name my_msu_repo
```

The screenshot shows a Visual Studio Code interface with a terminal window open. The terminal window displays the following command and its output:

```
[resadmin@host-1 fn_my_ldap]$ resilient-sdk codegen -p . --reload
'codegen --reload' started for 'fn_my_ldap'
Generating codegen package...
Connecting to IBM Security SOAR at: 10.0.0.1
Unverified HTTPS requests (cafile=false).
IBM Security SOAR version: v46.0
'codegen' complete for 'fn_my_ldap'

NOTE: Ensure the MANIFEST.in file includes line:
recursive-inlude fn_my_ldap/util *

'codegen --reload' complete for 'fn_my_ldap'
● [resadmin@host-1 fn_my_ldap]$ resilient-sdk package -p . --no-samples --repository-name "my_msu_repo"
Build Distribution starting

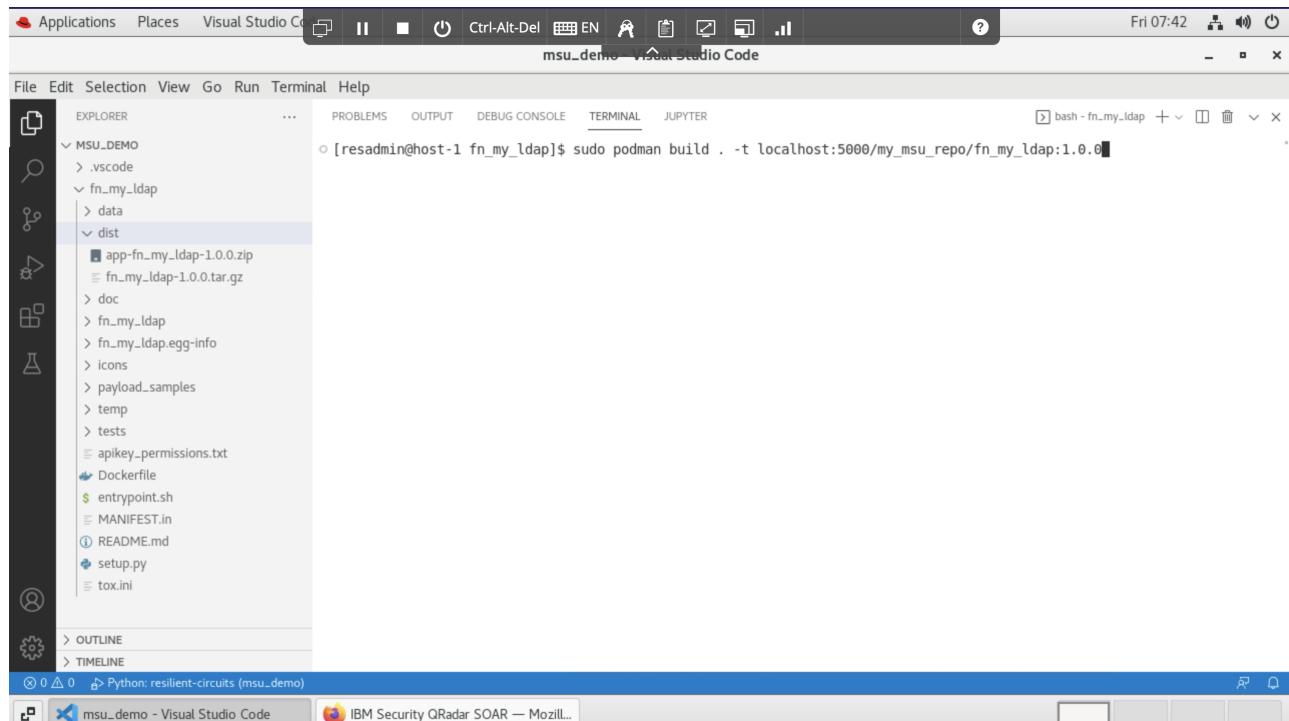
/opt/rh/rh-python36/root/usr/lib64/python3.6/distutils/dist.py:261: UserWarning: Unknown distribution option: 'display_name'
  warnings.warn(msg)
running sdist
running egg_info
writing fn_my_ldap.egg-info/PKG-INFO
writing dependency_links to fn_my_ldap.egg-info/dependency_links.txt
writing entry points to fn_my_ldap.egg-info/entry_points.txt
writing requirements to fn_my_ldap.egg-info/requirements.txt
writing top-level names to fn_my_ldap.egg-info/top_level.txt
listing git files failed - pretending there aren't any
reading manifest file 'fn_my_ldap.egg-info/SOURCES.txt'
reading manifest template 'MANIFEST.in'
warning: no files found matching 'doc/*.md'
writing manifest file 'fn_my_ldap.egg-info/SOURCES.txt'
running check
warning: check: missing meta-data: if 'author' supplied, 'author_email' must be supplied too
```

- Notice now that a **dist** directory has been created with a file called **fn\_my\_ldap/dist/app-fn\_my\_ldap-1.0.0.zip**. This is what we will use to install the app in step 15.

## Step 15: Push to local registry

- Before we can install the app with App Host, we need to ensure that the container image exists in a place where our App Host can pull it down.
- Back in step 1, we started up the local registry running in podman. We'll use that to host our container image. Other public and private repositories are available and can be used as long as the App Host machine can reach them!
- Use podman (docker would work, it just isn't available on this specific system) to build and tag the image for our repository

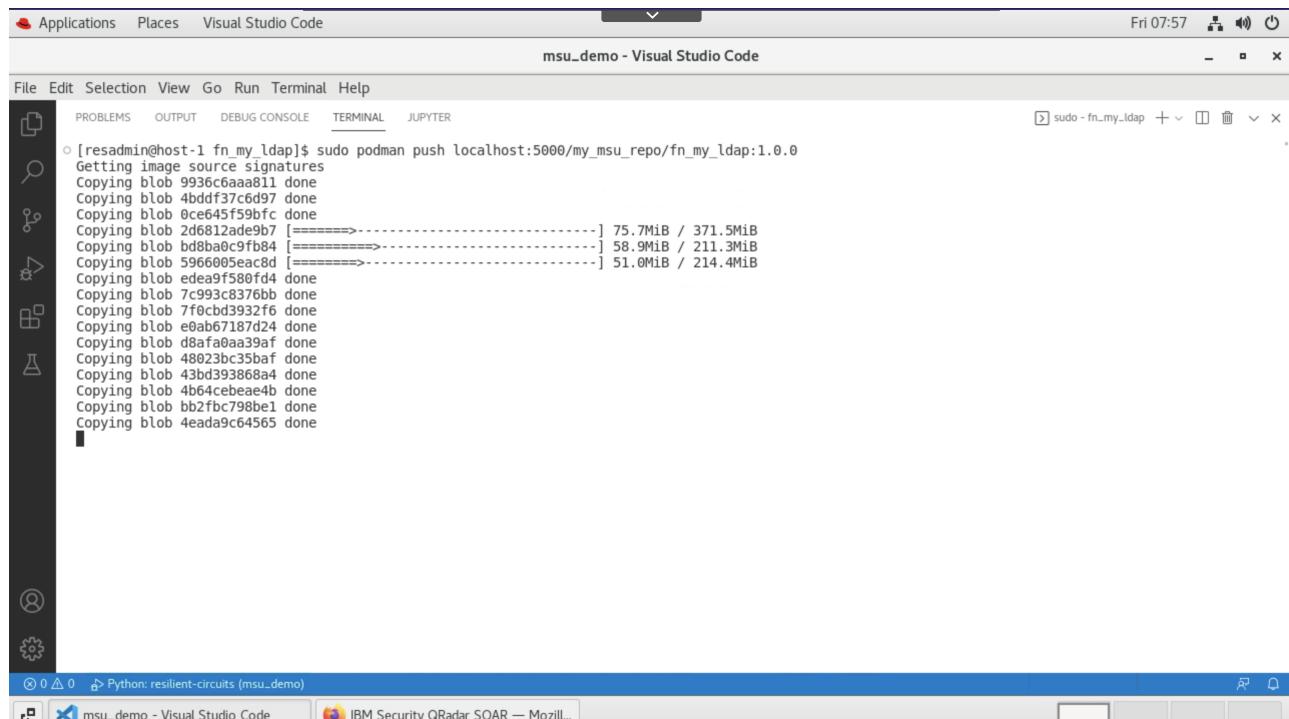
```
sudo podman build . -t localhost:5000/my_msu_repo/fn_my_ldap:1.0.0
```



Note: the image repository name must match the value given in the `resilient-sdk package` command. In this case, we've used `my_msu_repo` but any value is ok as long as it matches the `--repository-name` flag value in the `package` command. The default if no value is given to `package` is `ibmresilient`.

- Push the newly build container to the local registry:

```
sudo podman push localhost:5000/my_msu_repo/fn_my_ldap:1.0.0
```



## Step 16: Install with App Host

- Switch back to the SOAR UI. Navigate to the **Apps** tab of the **Administrator Settings**.

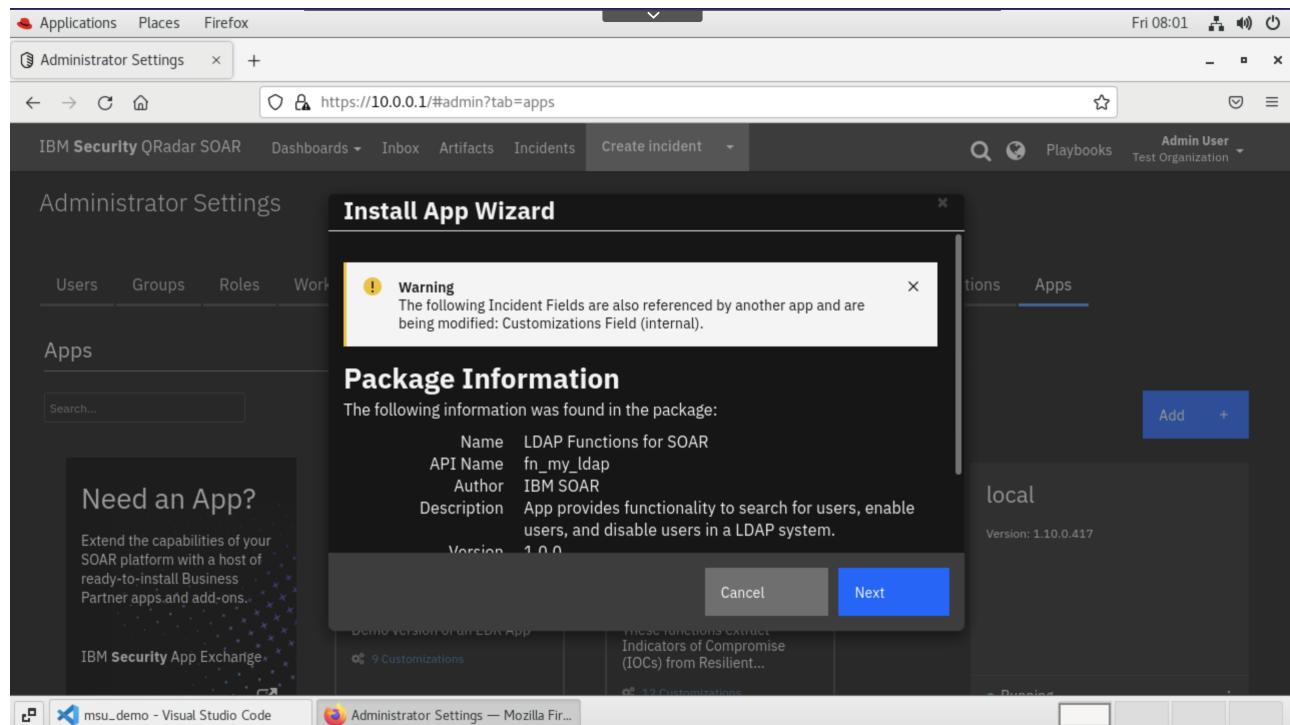
The screenshot shows the IBM Security QRadar SOAR Administrator Settings interface. The top navigation bar includes links for Applications, Places, Firefox, and the current tab, Administrator Settings. The URL in the address bar is https://10.0.0.1/#admin?tab=apps. The main content area is titled "Administrator Settings" and has a sub-navigation bar with tabs: Users, Groups, Roles, Workspaces, Timeframes, Network, Organization, Threat Sources, Notifications, and Apps. The Apps tab is currently selected. Below this, there's a search bar labeled "Search..." and a row of buttons: "Install" (blue), "Add" (white), and two others partially visible. A sidebar on the left says "Need an App?" and encourages extending capabilities with Business Partner apps and add-ons from the IBM Security App Exchange. Several app cards are listed, including "Demo EDR App" (IBM SOAR, Version: 1.0.0, Demo version of an EDR App, 9 Customizations), "fn\_ioc\_parser\_v2" (Resilient Labs, Version: 1.0.2, These functions extract Indicators of Compromise (IOCs) from Resilient..., 12 Customizations), and "local" (Version: 1.10.0.417). At the bottom of the page, there are browser tabs for "msu\_demo - Visual Studio Code" and "Administrator Settings — Mozilla Fir...".

- Install your app by clicking **Install** and finding the **fn\_my\_ldap/dist/app-fn\_my\_ldap-1.0.0.zip** file on your local machine.

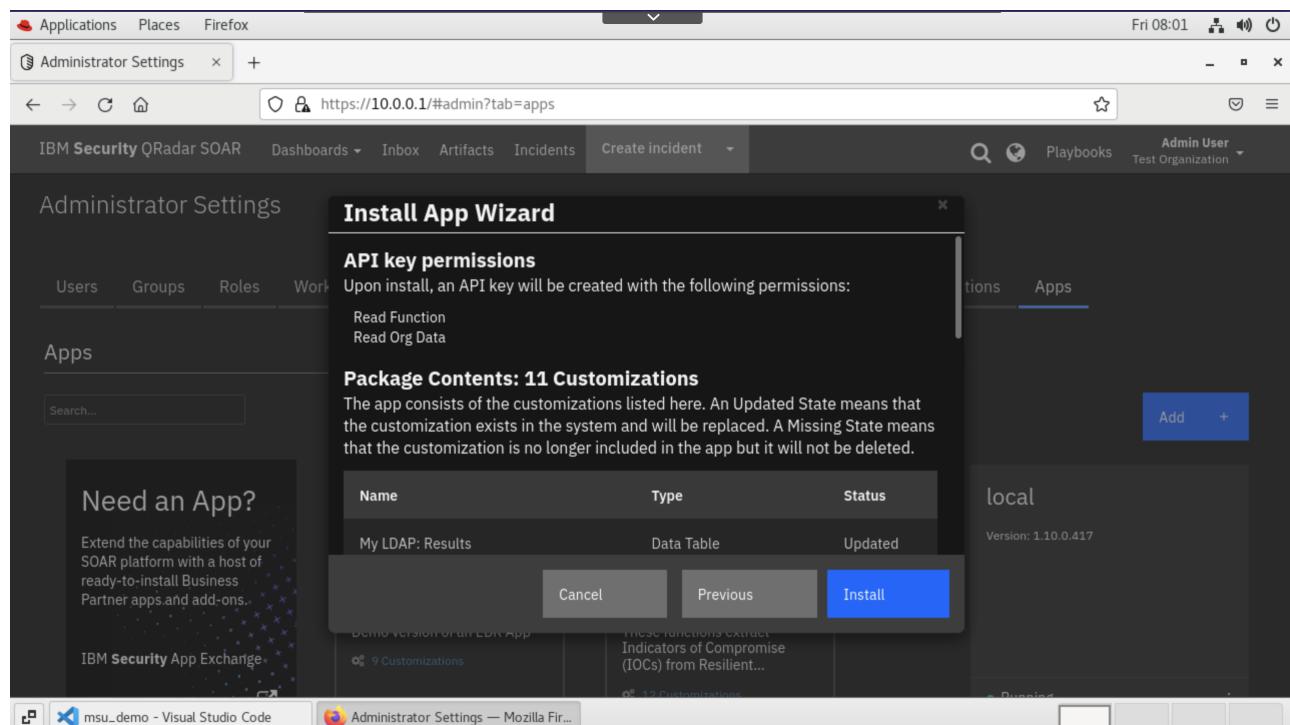
The screenshot shows a "File Upload" dialog box. On the left is a sidebar with a tree view of locations: Recent, Home, Documents, Downloads, Music, Pictures, Videos, Floppy Disk, and Other Locations. The "Recent" location is selected. The main area is titled "File Upload" and contains a file list table with columns for Name, Size, and Modified. Two files are listed: "app-fn\_my\_ldap-1.0.0.zip" (217.5 kB, 06:45) and "fn\_my\_ldap-1.0.0.tar.gz" (163.0 kB, 06:45). At the bottom right of the dialog box are buttons for "All Files" and "Open". The status bar at the bottom shows browser tabs for "msu\_demo - Visual Studio Code" and "Administrator Settings — Mozilla Fir...".

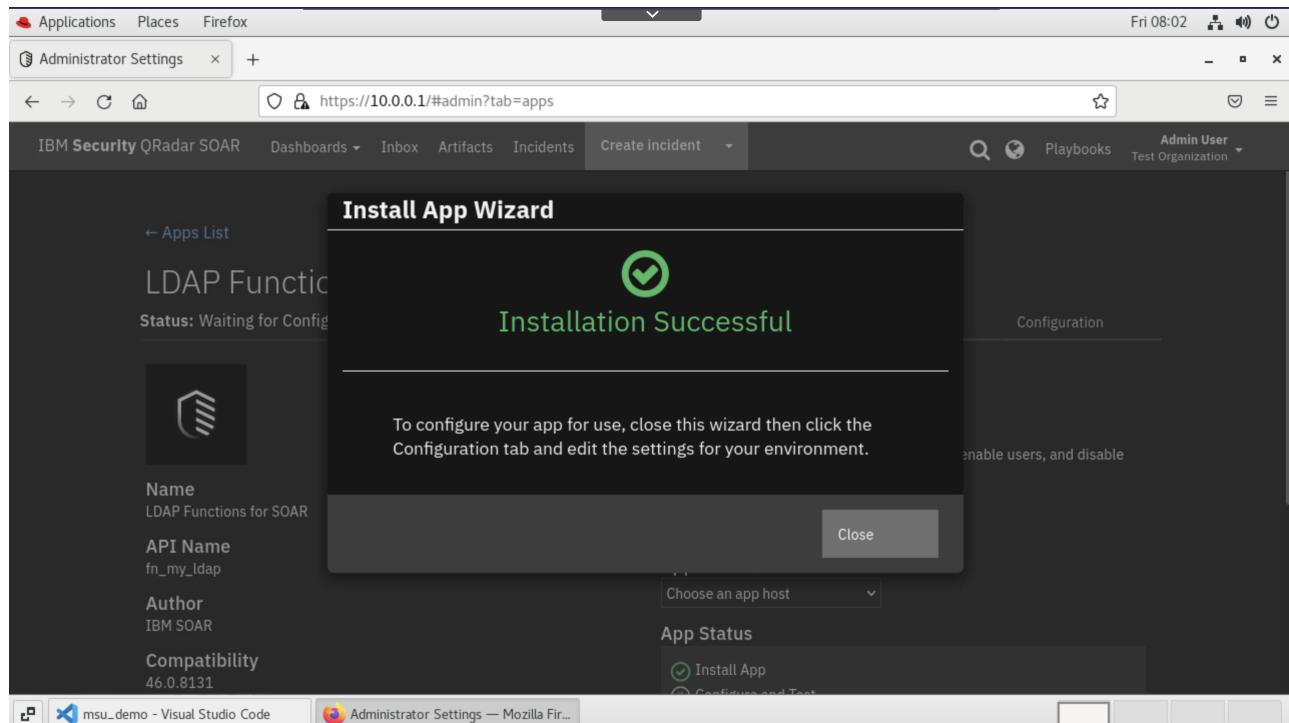
Click **Open**, followed by **Upload File**.

- Click Next:



- Click Install:





- Navigate to the **app.config** file in the **Configuration Tab** of the app:

File Name	File Location	File Type	Created At	Last Modified
app.config	/etc/rescircuits	Initialization	09/09/2022 08:01	09/09/2022 08:01
cert.cer	/etc/rescircuits	Plain Text	09/09/2022 08:01	09/09/2022 08:01

- Enter the same details that we entered locally. Once entered, select the app host that points to the local repo and click **Test Configuration**:

The screenshot shows the IBM Security QRadar SOAR interface. At the top, there's a navigation bar with tabs like Applications, Places, Firefox, Administrator Settings, Dashboards, Inbox, Artifacts, Incidents, Create incident, and Admin User Test Organization. Below the navigation bar is a code editor window displaying a configuration file:

```
2 - [fn_my_ldap]
3 server=10.0.0.1
4 port=10389
5 admin_user=cn=admin,dc=planetexpress,dc=com
6 admin_password=GoodNewsEveryone
7 search_base=ou=people,dc=planetexpress,dc=com
8
9 - [resilient]
10 api_key_id = 595c046d-7359-4f02-84c9-fccdb2141c97
11 api_key_secret = $API_KEY_SECRET
12 cafile = false
13 host = 10.0.0.1
14 port = 443
15 org = Test Organization
16
17
```

Below the code editor, there are two input fields: "App Host" (containing "local") and "Test Configuration". The "Test Configuration" button is highlighted with a red box. In the bottom right corner of the main window, there's a yellow notification bar stating "The last test was successful." with a close button.

- Scroll up and click **Save and Push Changes**:

The screenshot shows the 'Configuration' tab for the 'LDAP Functions / app.config' section. The 'File Name' field contains 'app.config'. The 'File Path' field contains '/etc/rescircuits'. The 'File Annotations' field contains the configuration snippet: [fn\_my\_ldap] server=<ldap\_server\_here>. The 'Save and Push Changes' button is highlighted with a red box.

- Navigate back to the **Details** tab for the app, scroll down, and click **Deploy**:

The screenshot shows the 'Details' tab for the 'LDAP Functions for SOAR' app. The 'Name' field is 'LDAP Functions for SOAR'. The 'API Name' field is 'fn\_my\_ldap'. The 'Author' field is 'IBM SOAR'. The 'Compatibility' field is '46.0.8131'. In the 'App Status' section, the 'Deploy to App Host' step is shown with a yellow warning icon. The 'Deploy' button is highlighted with a red box.

- Once the app is deployed, you can now run its functions and playbooks.

Administrator Settings

IBM Security QRadar SOAR

Installed 09/09/2022 by Admin User

**Description**  
App provides functionality to search for users, enable users, and disable users in a LDAP system.

**Version**  
1.0.0

**App Host** local

**App Status**

- Install App
- Configure and Test
- Deploy to App Host
- Ready For Use!

Uninstall Undeploy Restart Download Logs Upgrade

- To view the logs from app host, open a new terminal (in a production environment you'd have to ssh in to the App Host machine). Run the following `kubectl` command to get the pods that are deployed:

```
sudo kubectl get po -L app.kubernetes.io/instance -A
```

Note the container ID and namespace ID for the pod of your app.

```
File Edit View Search Terminal Help
[resadmin@host-1 ~]$ sudo kubectl get po -L app.kubernetes.io/instance -A
NAMESPACE     NAME                               READY   STATUS    RESTARTS   AGE
INSTANCE
kube-system   metrics-server-7cd5fcb6b7-m2dpt   1/1     Terminating   2 (6d19h ago)   6d20h
kube-system   local-path-provisioner-6c79684f77-b2nxt  1/1     Terminating   3 (6d19h ago)   6d20h
kube-system   coredns-d76bd69b-zq8wl            1/1     Terminating   2 (6d19h ago)   6d20h
kube-system   local-path-provisioner-6c79684f77-c2dxl  1/1     Running     7 (27h ago)    3d
83dc79bc-417d-4a69-baa8-ec2946817af5  4ef32c72-5cce-4875-a33d-63bc70091889-b575778d4-zqwj2  1/1     Running     0          27h
fn_task_utils 83dc79bc-417d-4a69-baa8-ec2946817af5  bb9f77ea-b6fe-4536-a6a9-8cd44eb99737-5d9b7b586d-j8n6g  1/1     Running     0          27h
fn_edr_app    kube-system   coredns-d76bd69b-sk92s            1/1     Running     4 (27h ago)    3d
83dc79bc-417d-4a69-baa8-ec2946817af5  850dddeec-72c3-4f22-a819-3c3b1edfae62-c8955cf68-jvmx6  1/1     Running     0          27h
fn_ioc_parser_v2 192281ea-d1c4-4bba-86ca-cbb0baf6f4a8  deployment-synchronizer-dd987448f-9qxp6           1/1     Running     5 (27h ago)    2d17h
192281ea-d1c4-4bba-86ca-cbb0baf6f4a8  192281ea-d1c4-4bba-86ca-cbb0baf6f4a8           2247690a-573b-47c4-a855-c4f4cc347d18-ff89f4b9d-wt5vh  1/1     Running     0          27h
fn_ldap_utilities 83dc79bc-417d-4a69-baa8-ec2946817af5  metrics-server-7cd5fcb6b7-9sbqw           1/1     Running     7 (27h ago)    3d
192281ea-d1c4-4bba-86ca-cbb0baf6f4a8  192281ea-d1c4-4bba-86ca-cbb0baf6f4a8           deployment-operator-96bd867f6-5v5st           1/1     Running     5 (27h ago)    2d17h
83dc79bc-417d-4a69-baa8-ec2946817af5  83dc79bc-417d-4a69-baa8-ec2946817af5           deployment-operator-67bc9c64f7-6qsn5           1/1     Running     7 (27h ago)    2d19h
83dc79bc-417d-4a69-baa8-ec2946817af5  83dc79bc-417d-4a69-baa8-ec2946817af5           deployment-synchronizer-7459985bb6-5tqtg          1/1     Running     10 (27h ago)   2d19h
83dc79bc-417d-4a69-baa8-ec2946817af5  192281ea-d1c4-4bba-86ca-cbb0baf6f4a8           f3bdad2f-0d87-40b8-a655-57089c50d71d-598457f447-q6p9x  1/1     Running     0          28m
fn_my_ldap
```

- Launch the logs in `follow` mode:

```
sudo kubectl logs -f <POD_ID> -n <NAMESPACE_ID>
```

