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

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*

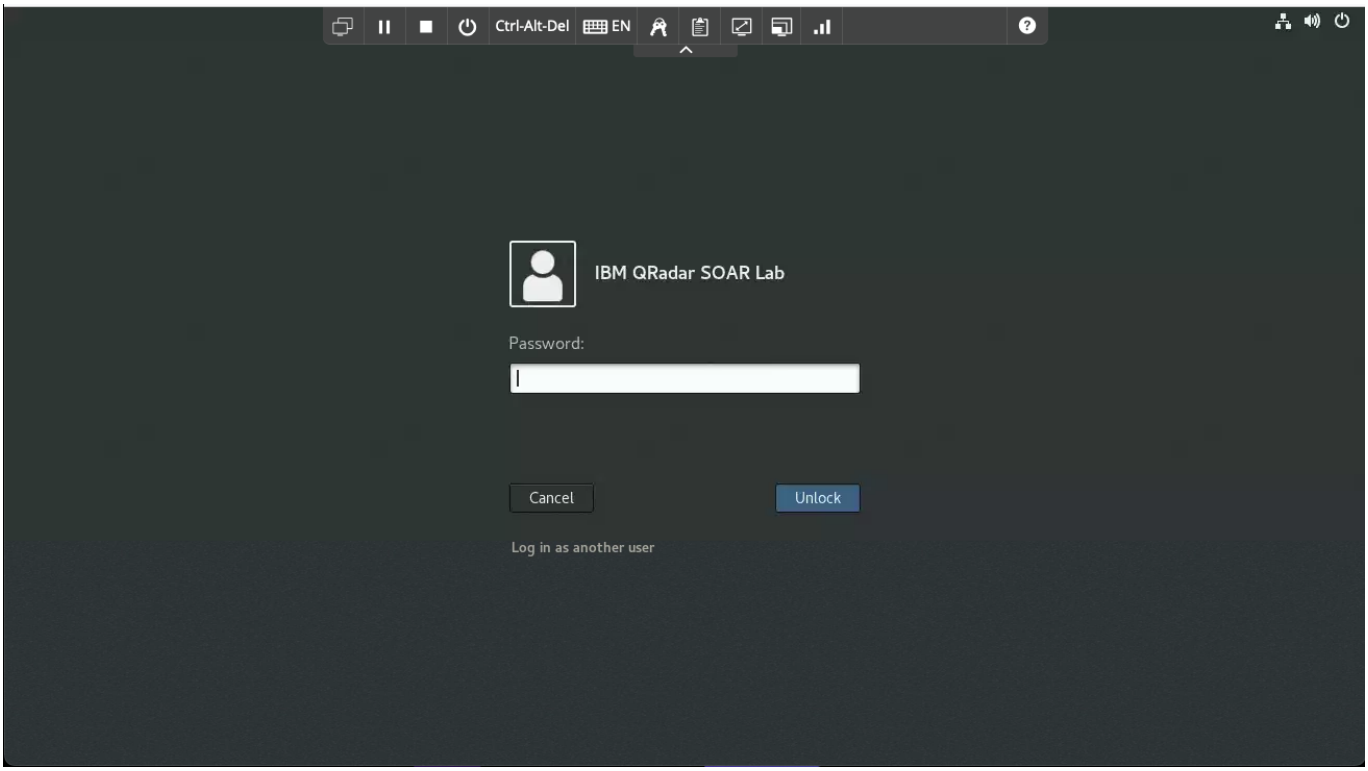
Step 0: *Sign up for SkyTap Account*

- Go to https://ibm.biz/MSU22SkyTap + `_<your_number>`

    - Example: I'll use https://ibm.biz/MSU22SkyTap_01
- Sign in with email and password provided by course provider

---

Step 1: *Login to Virtual Environment*

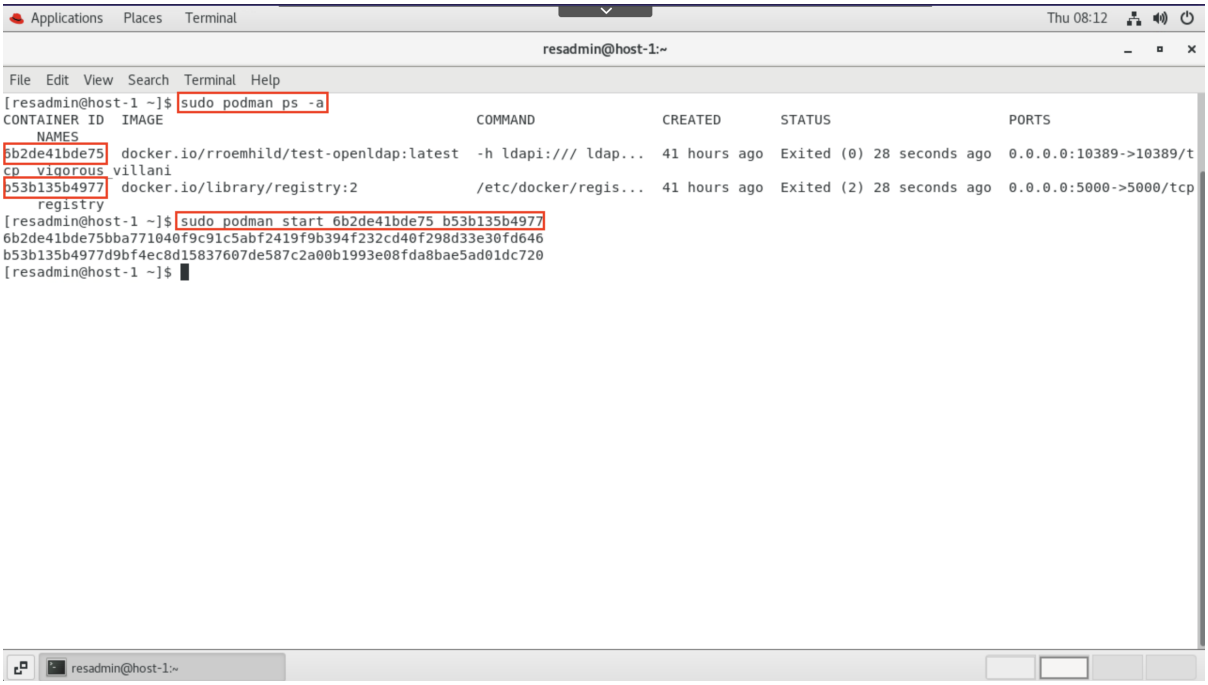


- Login as `resadmin` with the password shared by your course leader.
- Open **FireFox** and log into **IBM Security QRadar SOAR**.
- Open a new Terminal and restart the `ldap` and `registry` podman conatiners:
    - List all podman processes:

```
sudo podman ps -a
```

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

```
sudo podamn start <CONATINER_ID_1> <CONTAINER_ID_2>
```



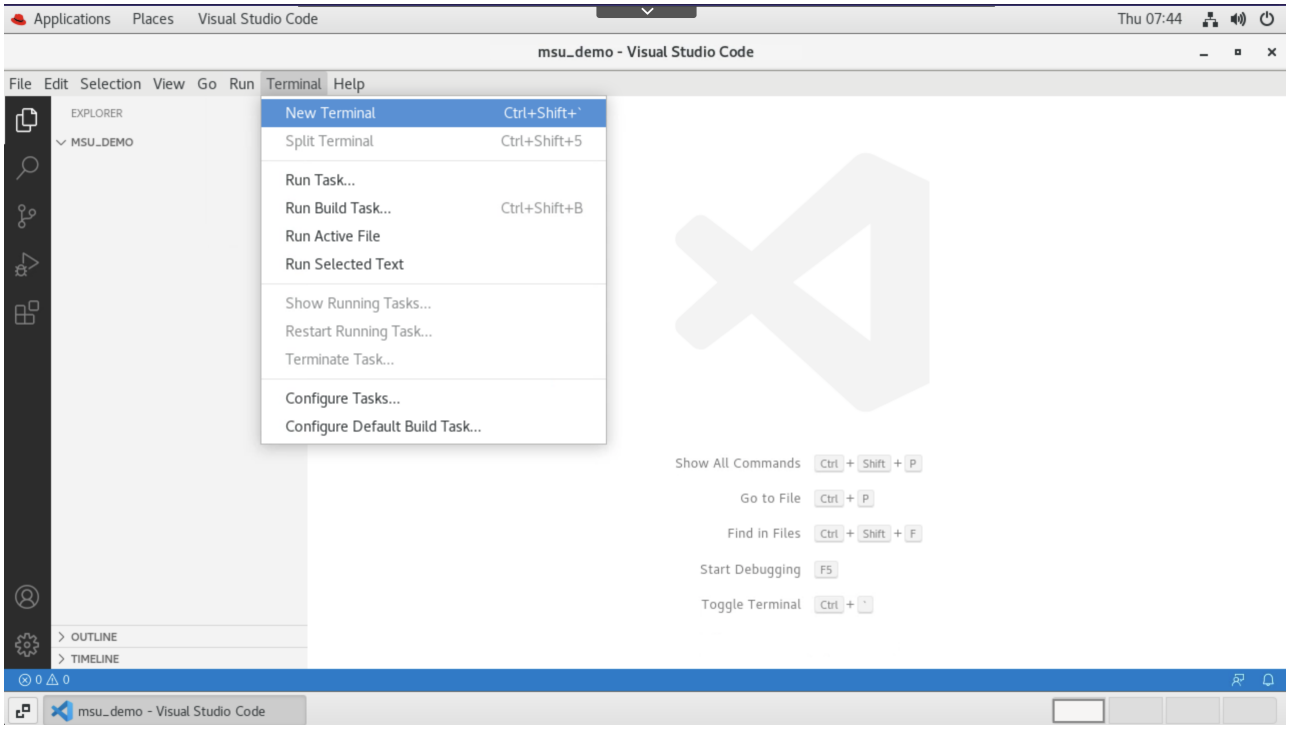
> More info on the LDAP container used in this lab can be found here

---

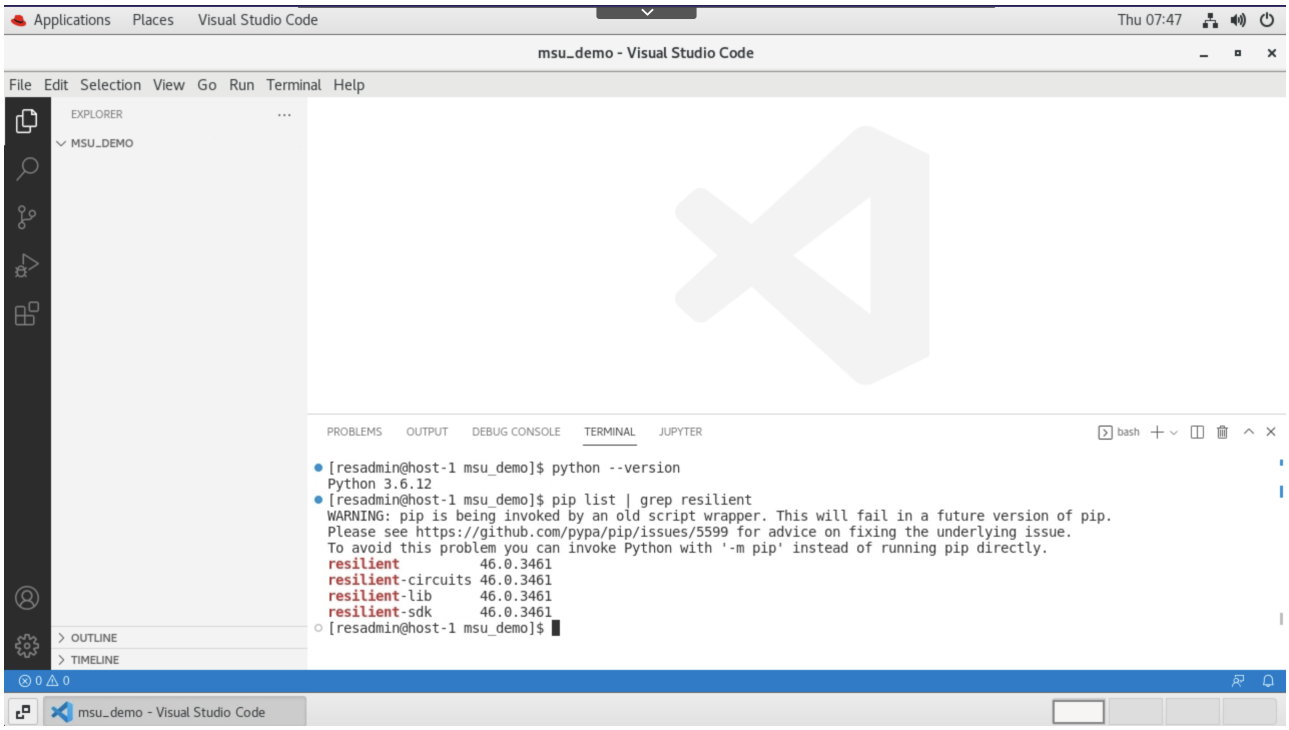
Step 2: *VS Code IDE Setup*

- 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.

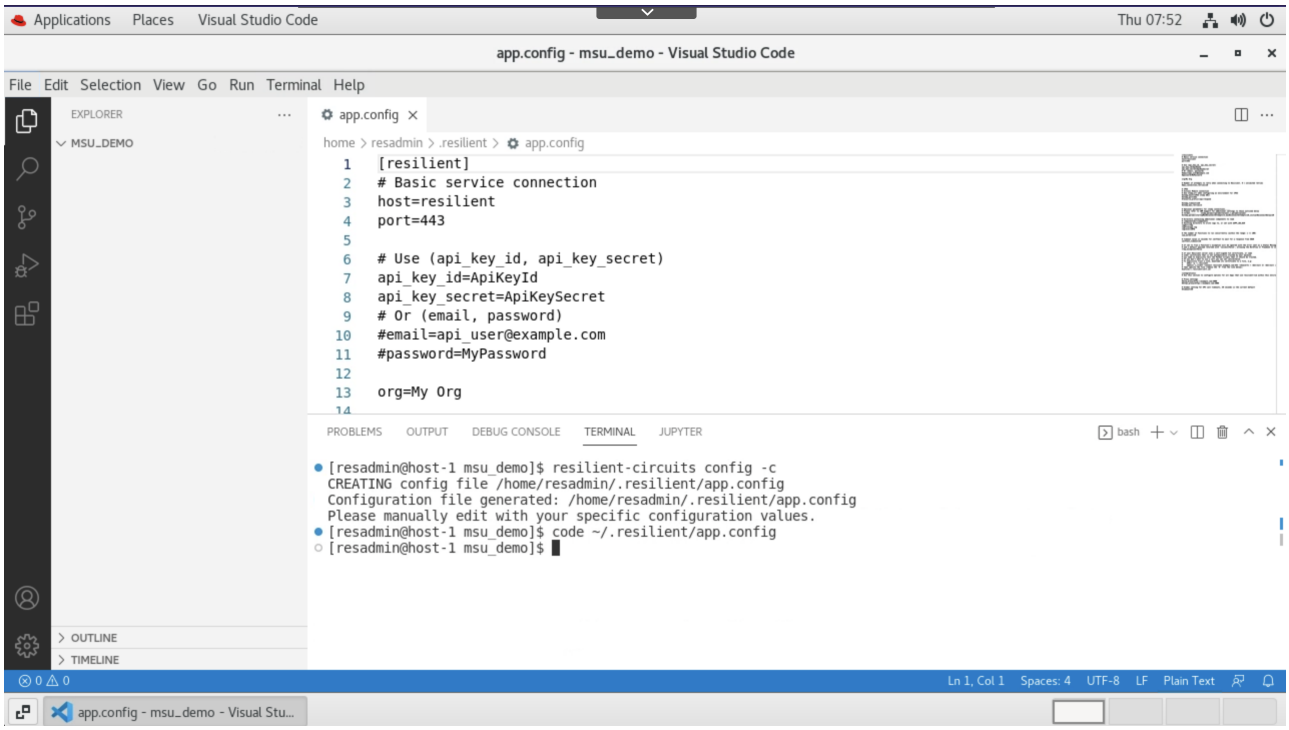
Step 3: `resilient-circuits` *Configuration*

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

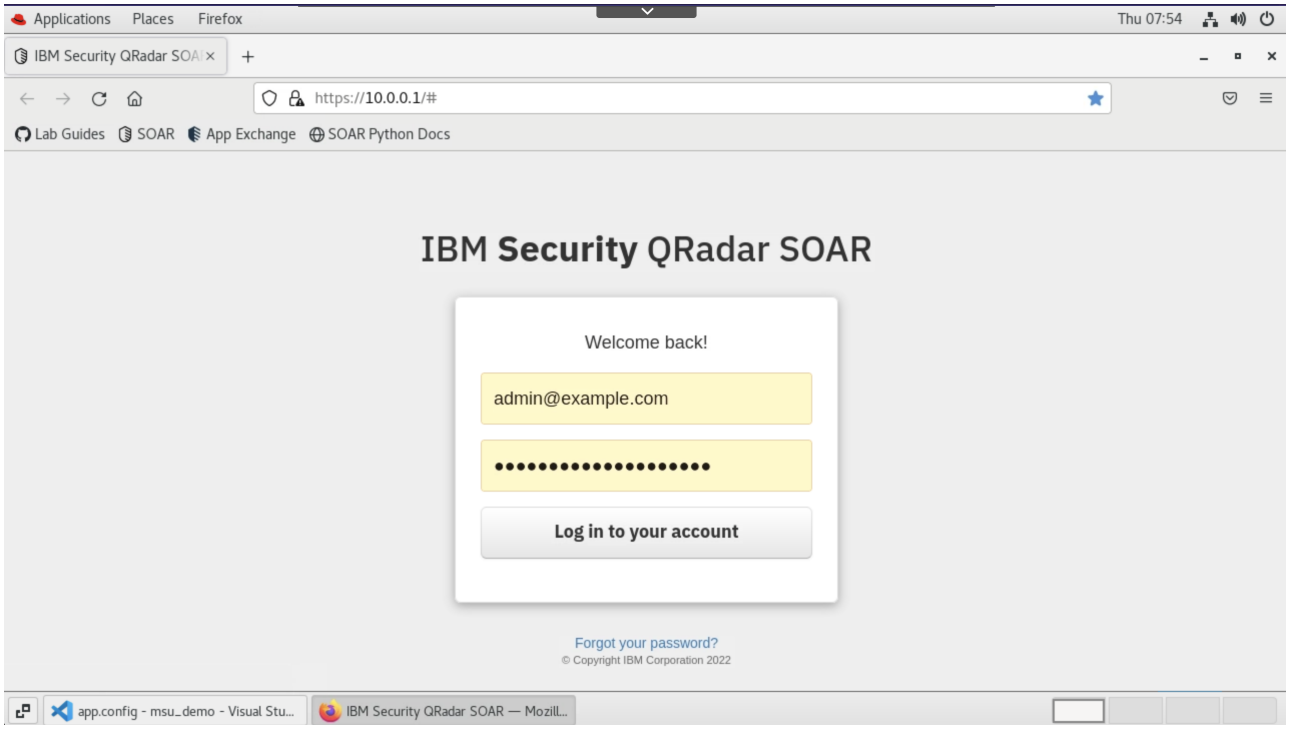
```
resilient-circuits config -c
```

And open the file in your IDE by running

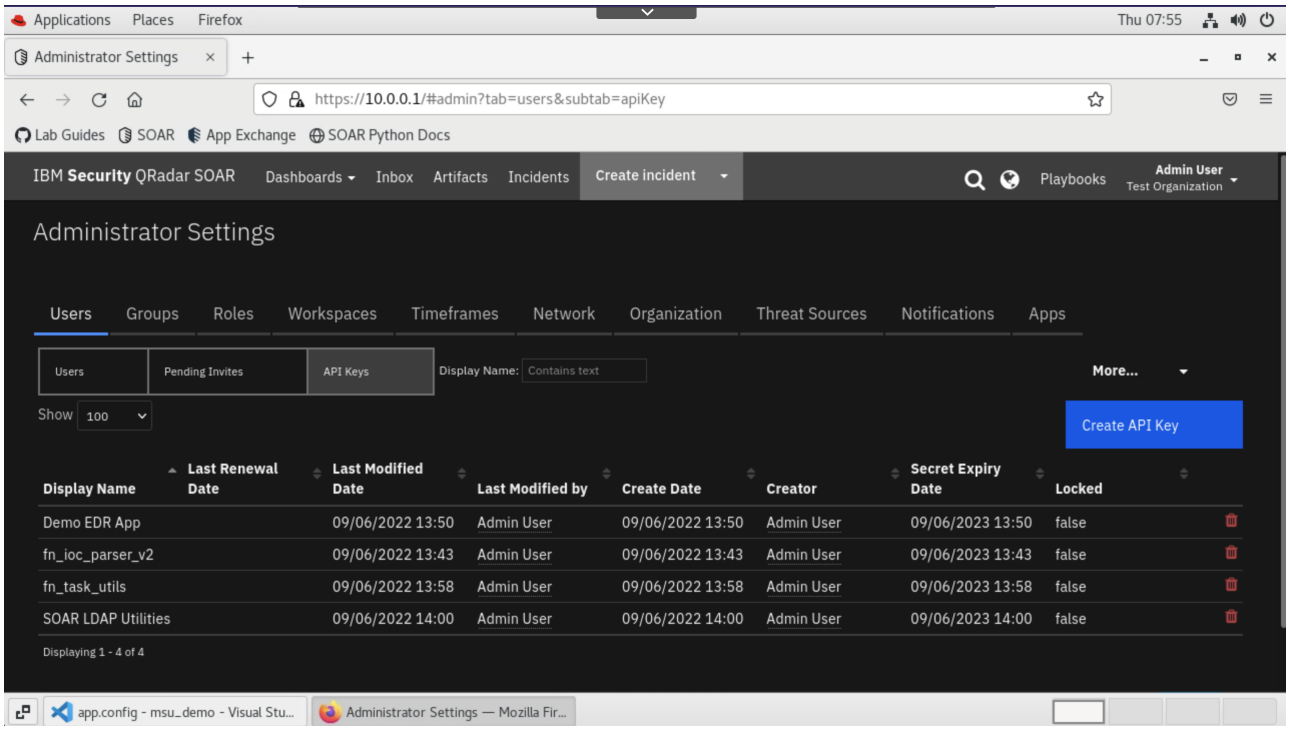
```
code ~/.resilient/app.config
```



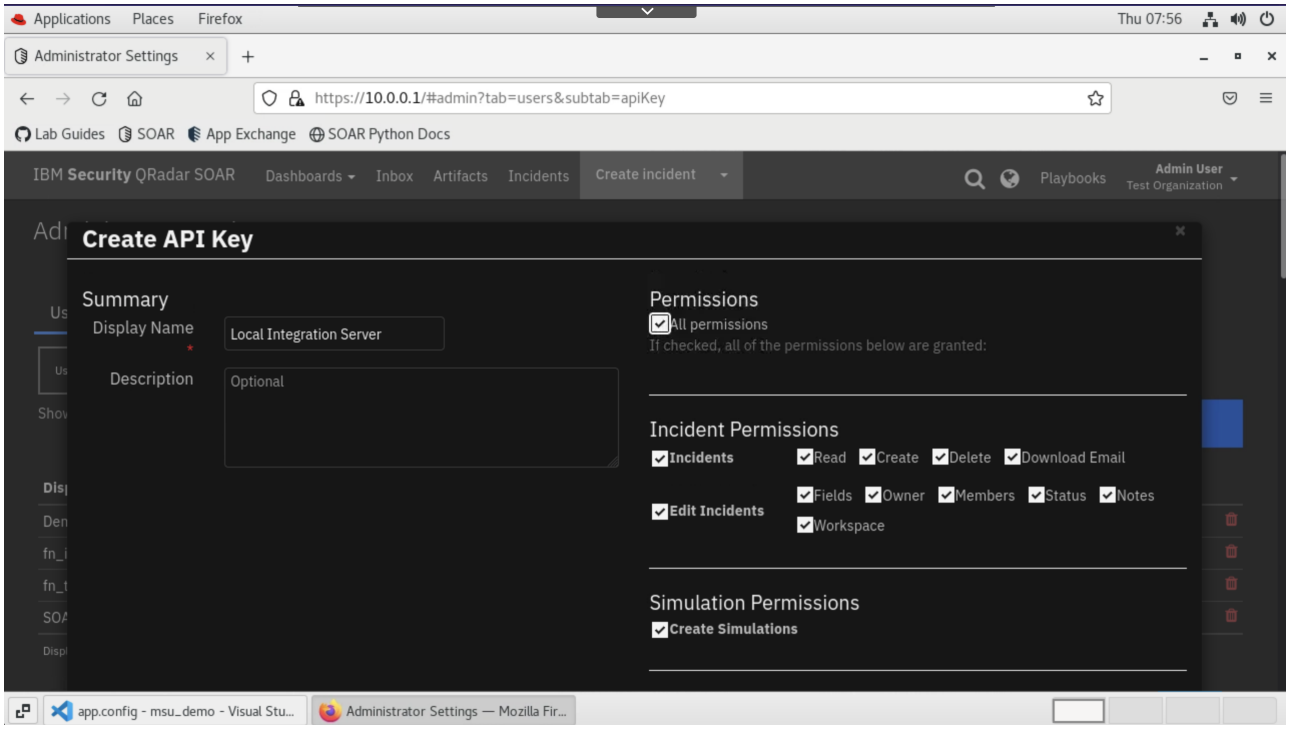
- Switch to Firefox and nagvigate to your SOAR host, in this case, 10.0.0.1 or through the bookmark in the bookmark menu. Log in using the saved admin credentials.



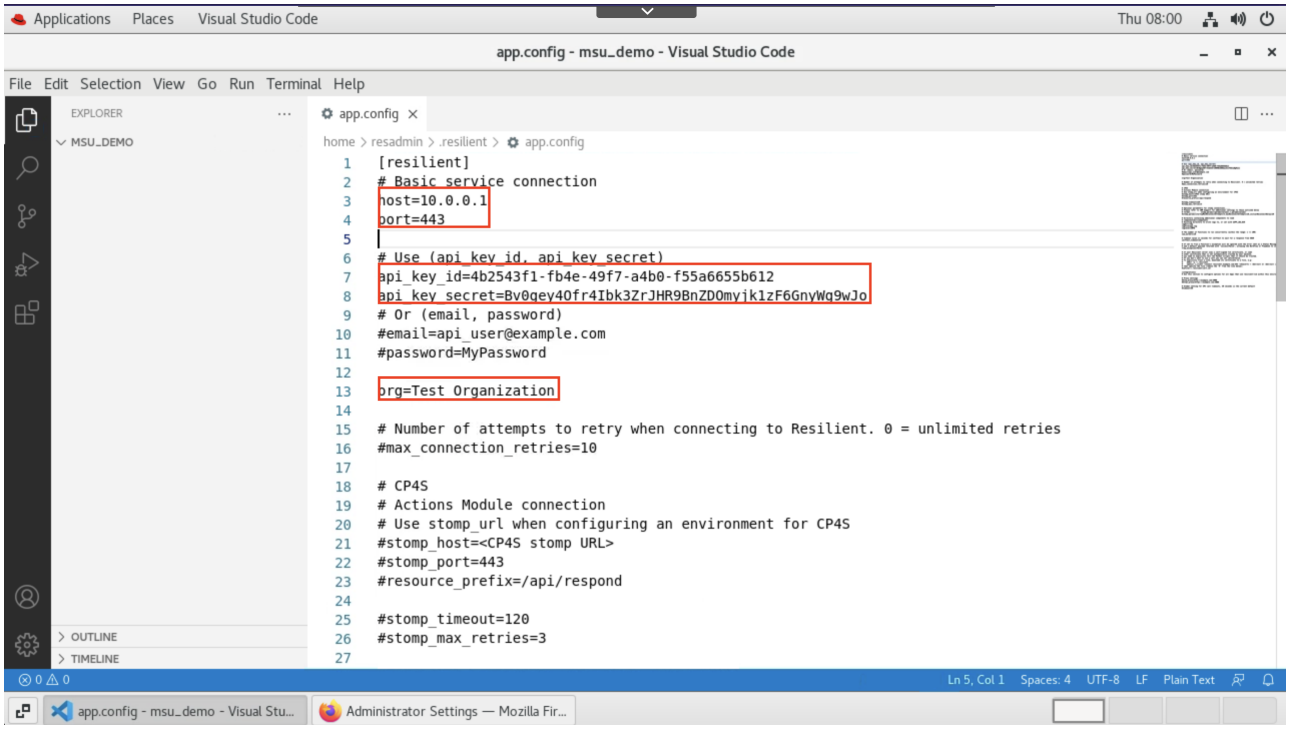
- Navigate to the **API Keys** section of the **Users** tab in **Administrator Settings** and create a new API Key.



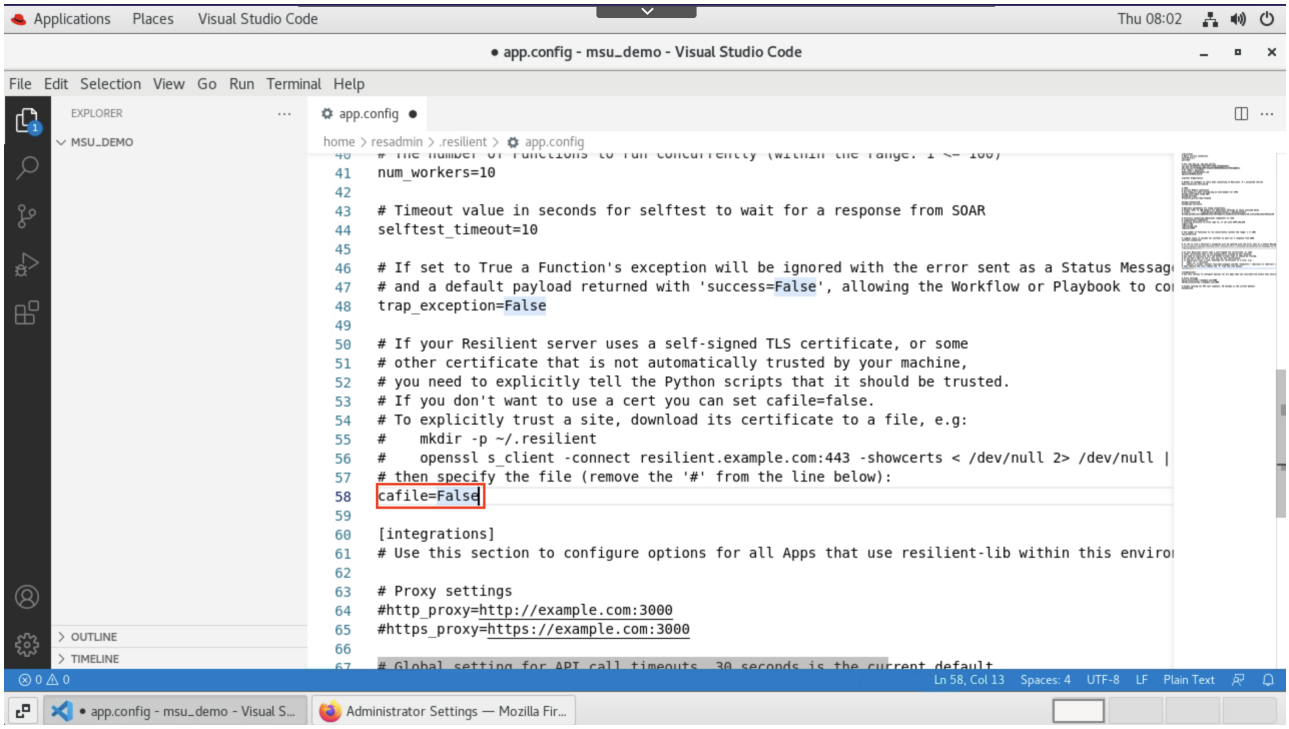
- Give the API Key the name "Local Integration Server" and give it all permissions. Click Create.



- IMPORTANT: Before clicking away from the screen that displays the crednetials, 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.

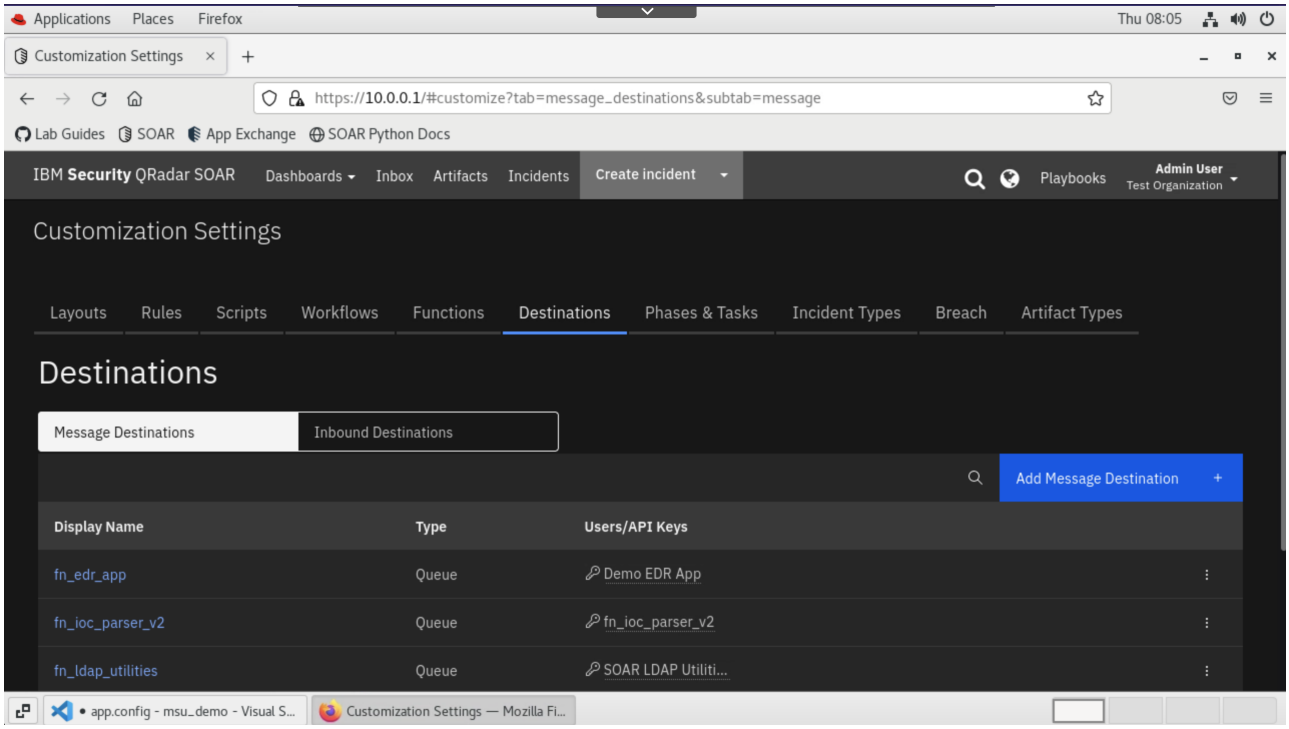


Scroll down to find the `cafile` setting and set to `False`:

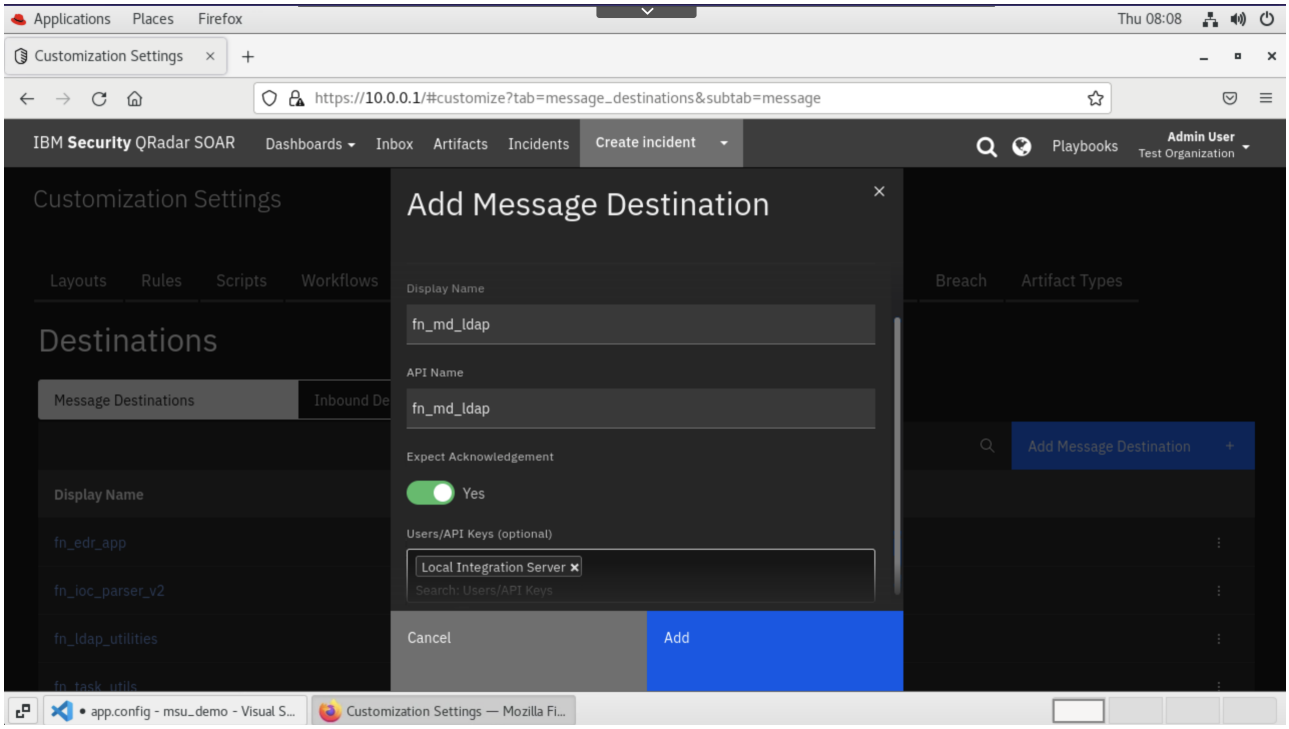


Step 4: *Create Message Destination in SOAR*

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

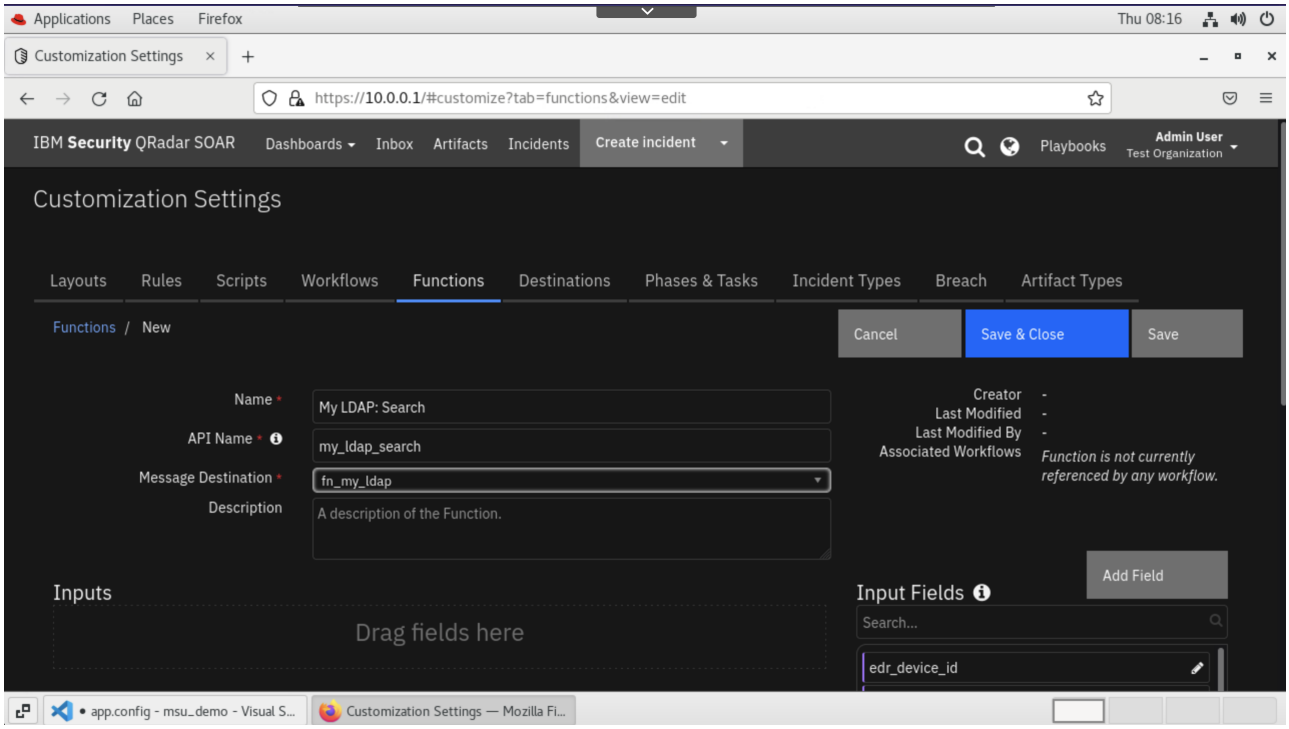


- Create a new **Message Destination** of type **Queue** with display name and API name `fn_my_ldap`. Add the API Key generated earlier to the list of allowed keys for this message destination. Click **Add**.

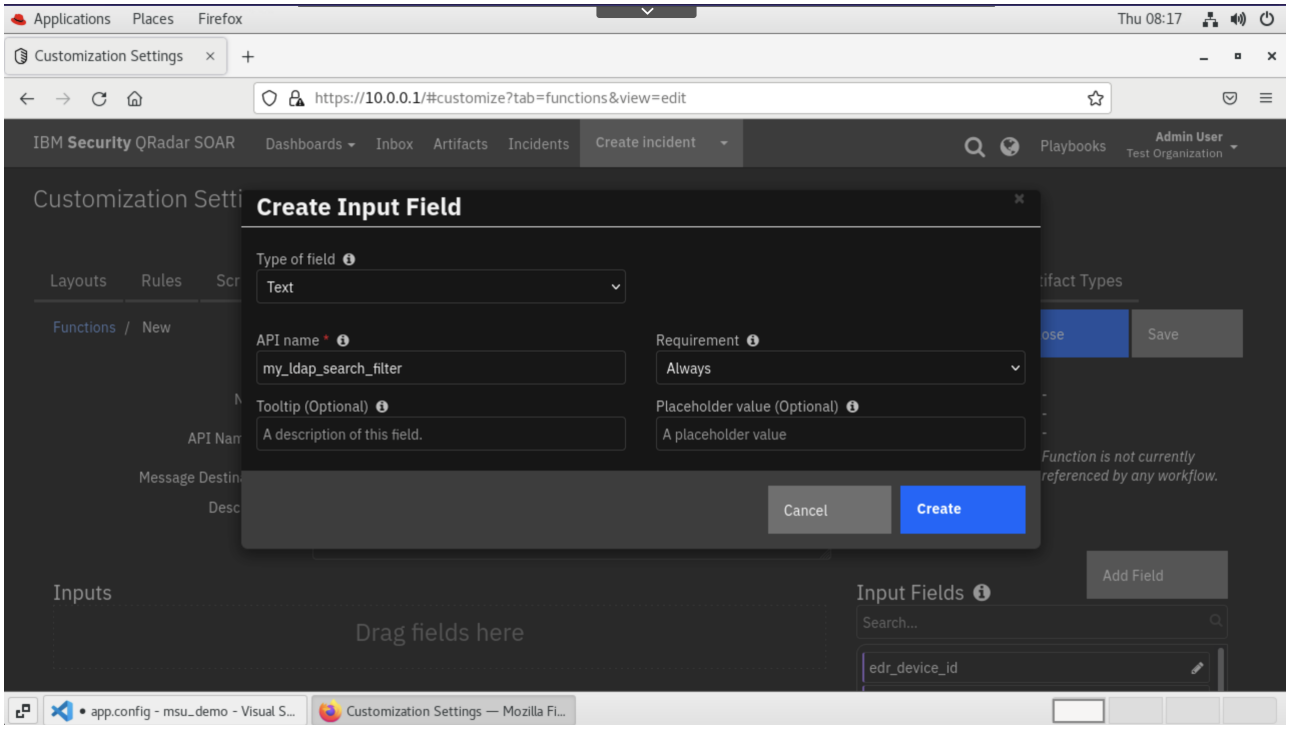


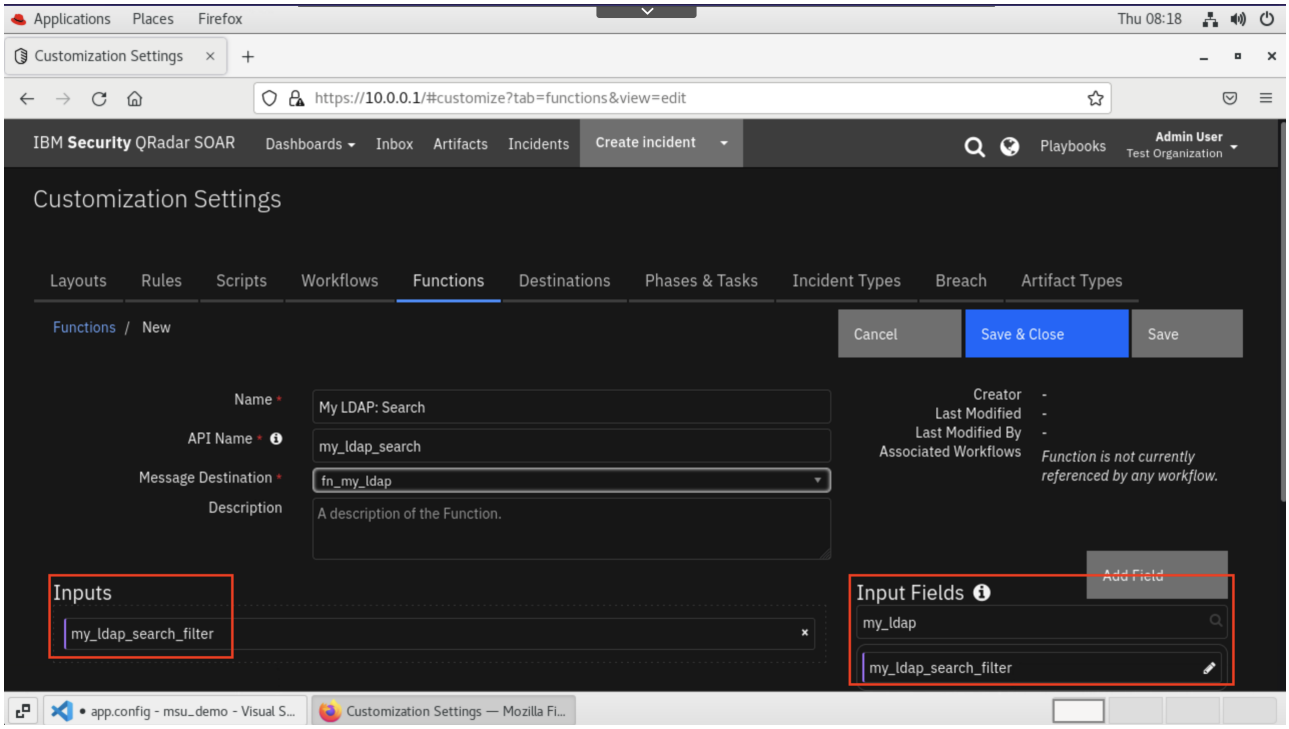
Step 5: *Create Functions for `fn_my_ldap`*

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



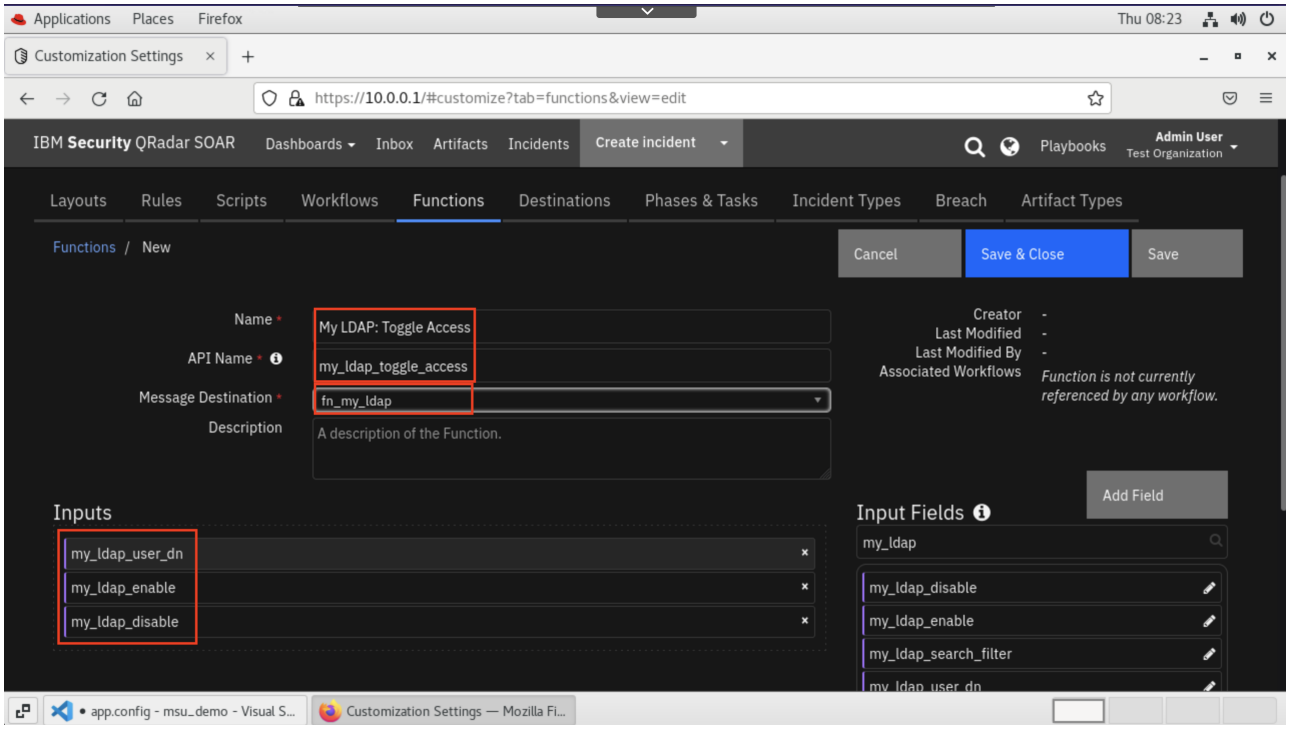
- Create the input field `my_ldap_search_filter` and add it to the function:





- **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`

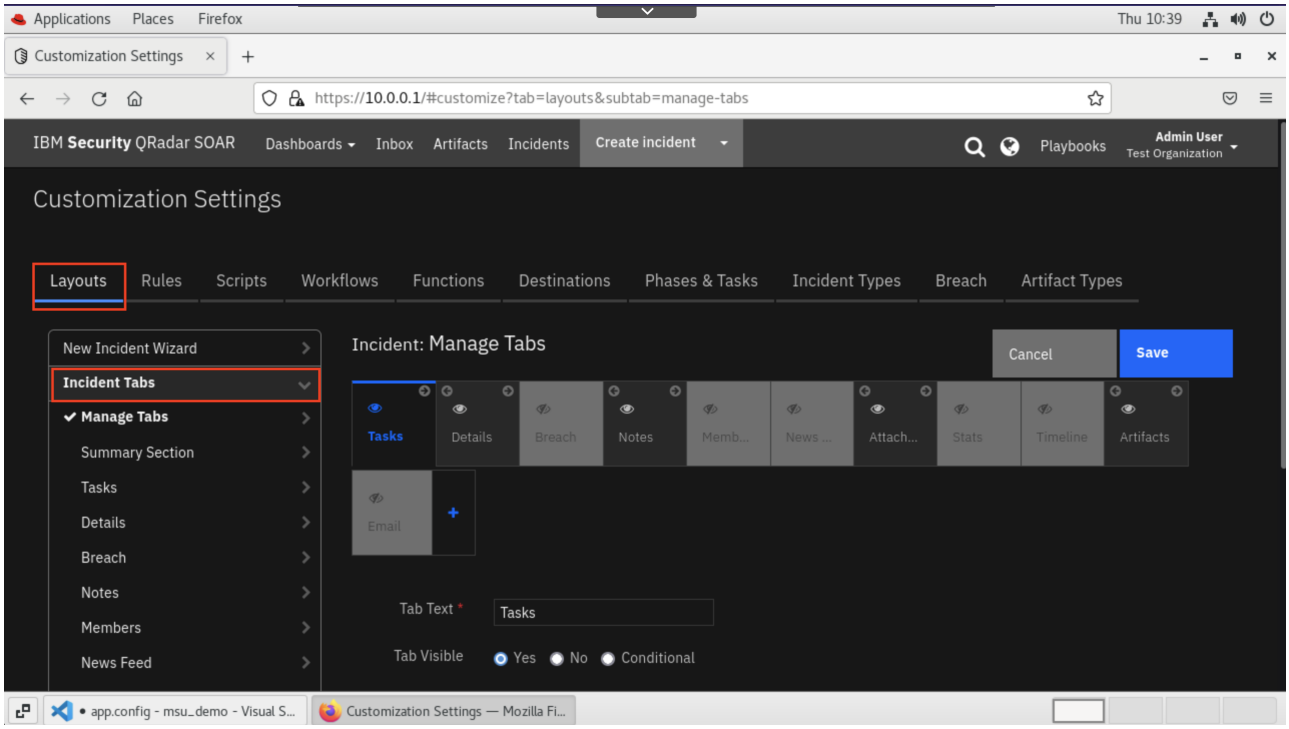
(Boolean, Optional):

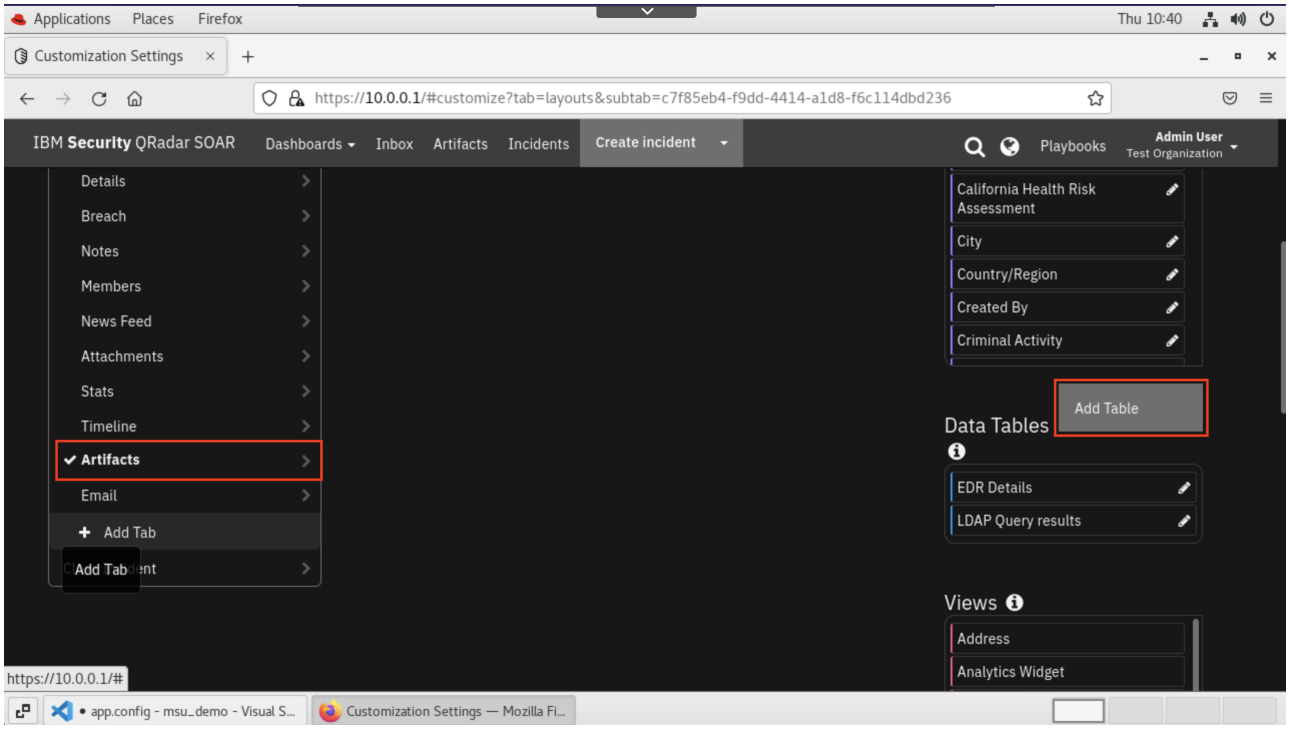


- **Save & Close** the function.

Step 6: *Create a custom Datatable*

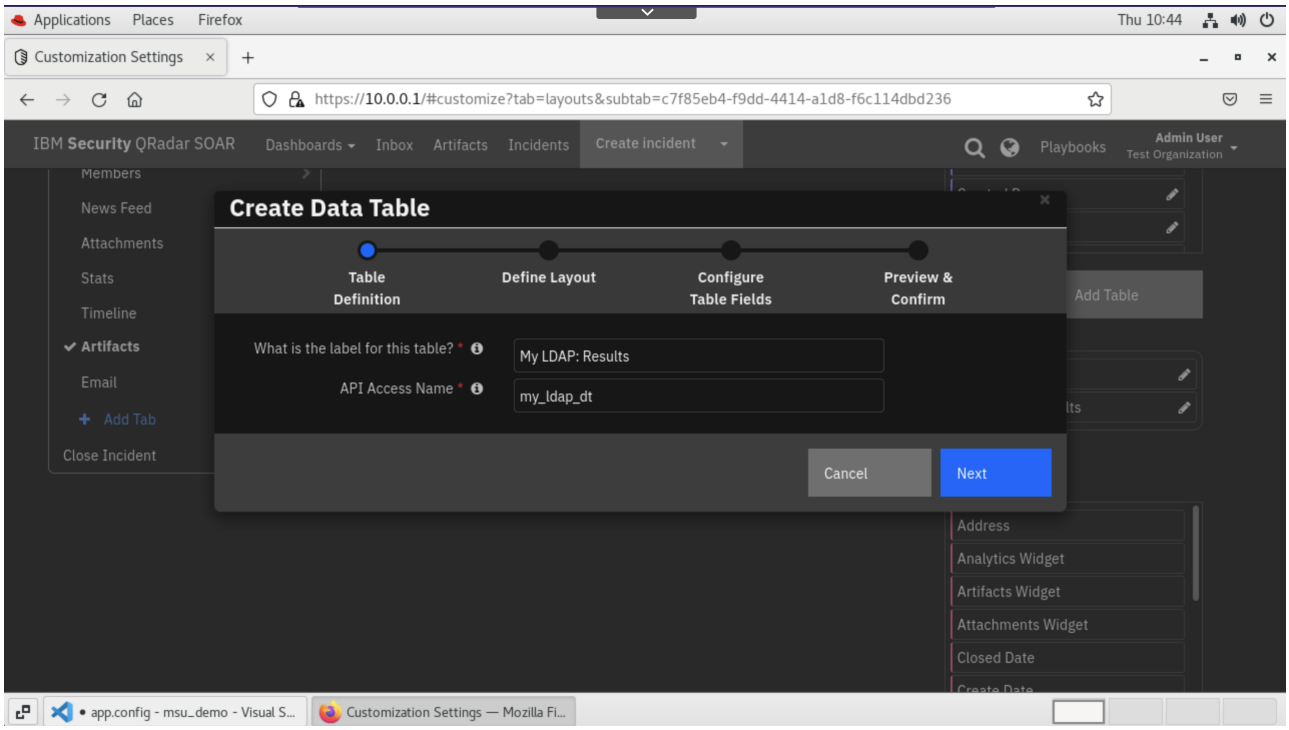
- Navigate to the **Layouts** tab found with **Customization Settings**.
- Select **Incident Tabs** and scroll down and select any of the specific tabs. We chose **Artifacts** as that is where we'll place the custom datatable.





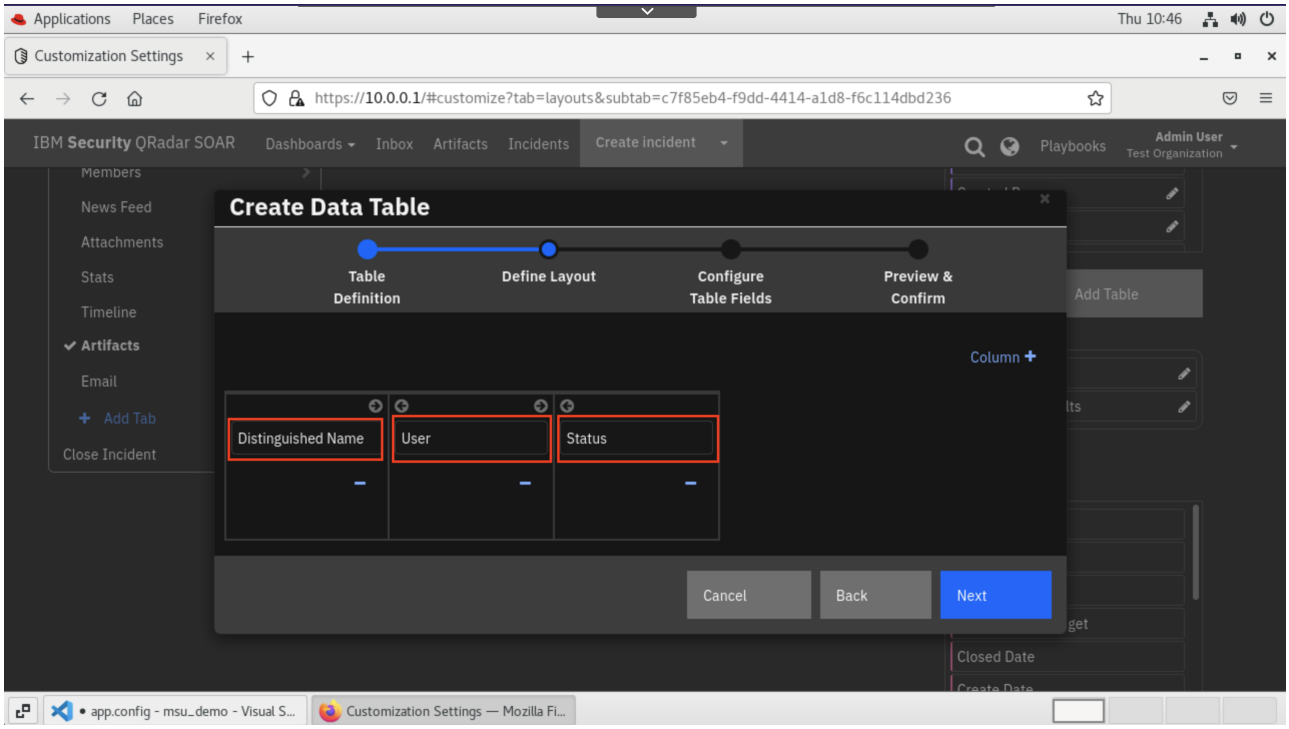
Click **Add Table**.

- Name the table **My LDAP: Results** and give it an API name `my_ldap_dt`.



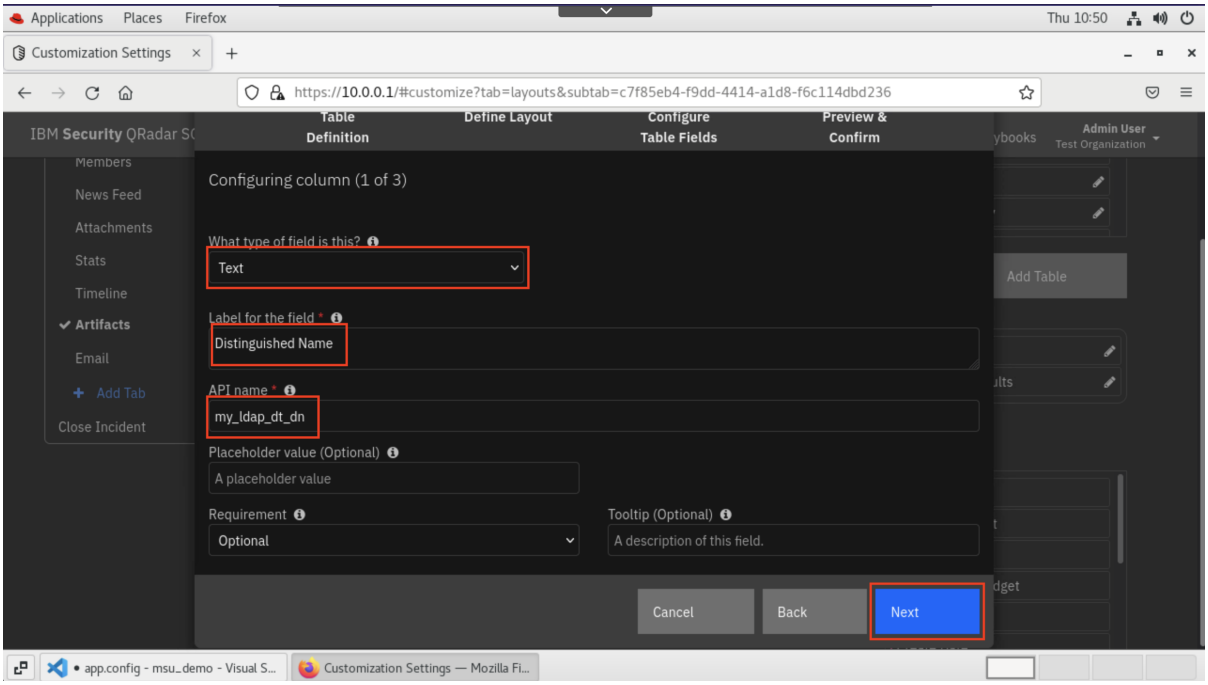
Click **Next**.

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



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

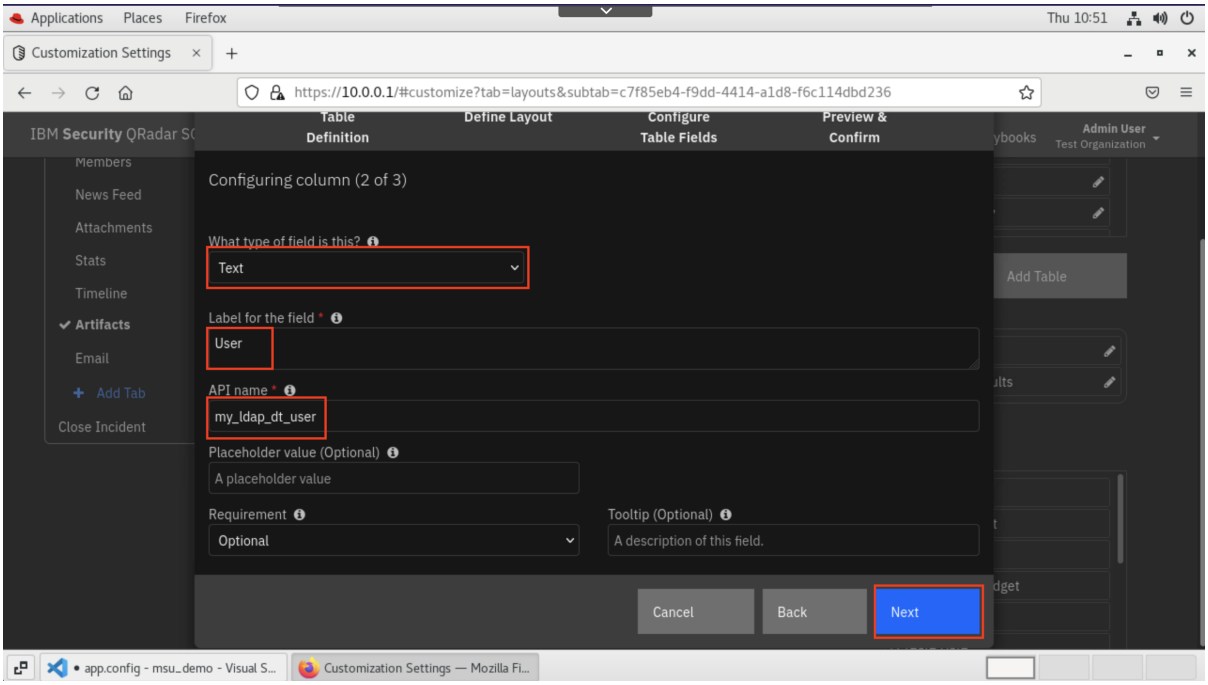
- Fill in the following details for the "Distinguished Name" column:
  - Type: `Text`
  - Label: "Distinguished Name"
  - API Name: `my_ldap_dt_dn`



  Click **Next**.

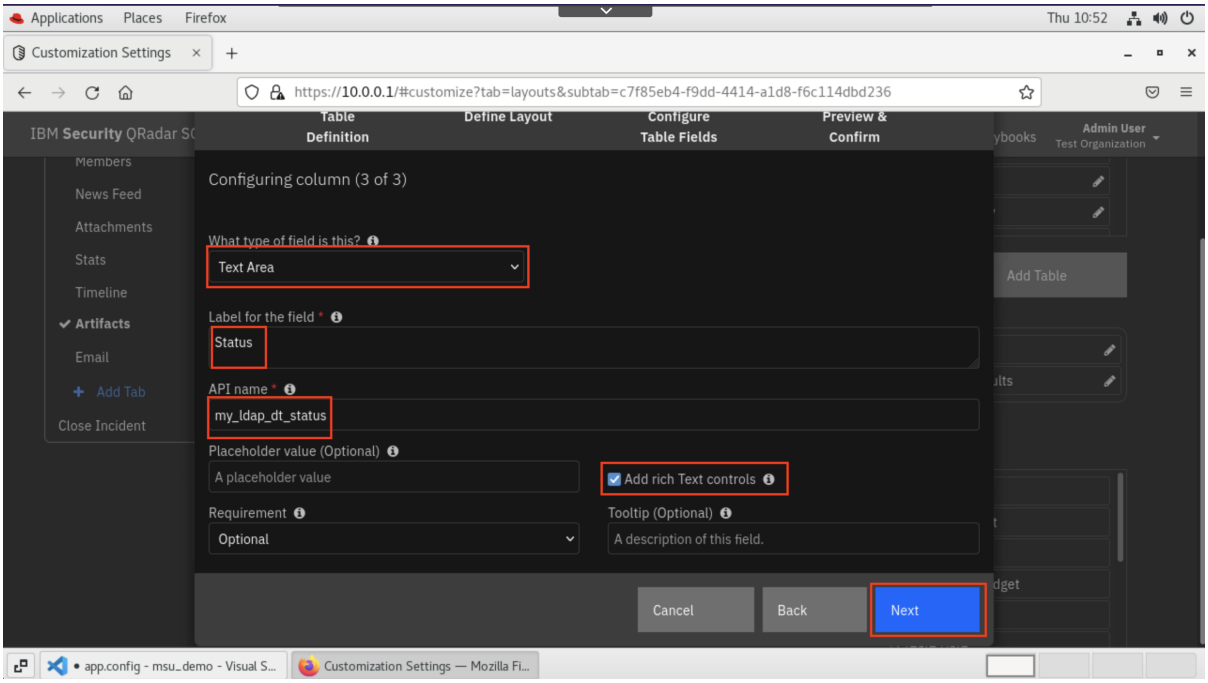
- Fill in the following details for the "User" column:
  - Type: `Text`
  - Label: "User"

- API Name: `my_ldap_dt_user`



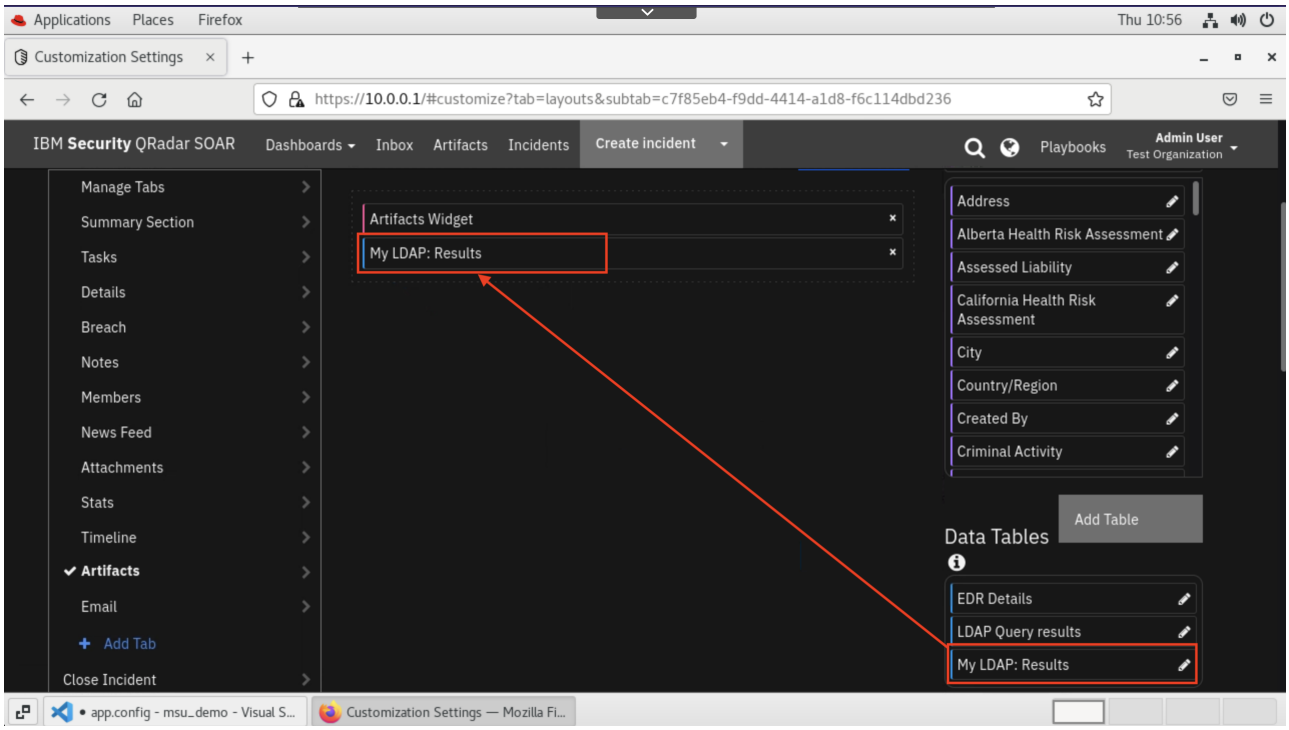
Click **Next**.

- 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"



Click **Next**.

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

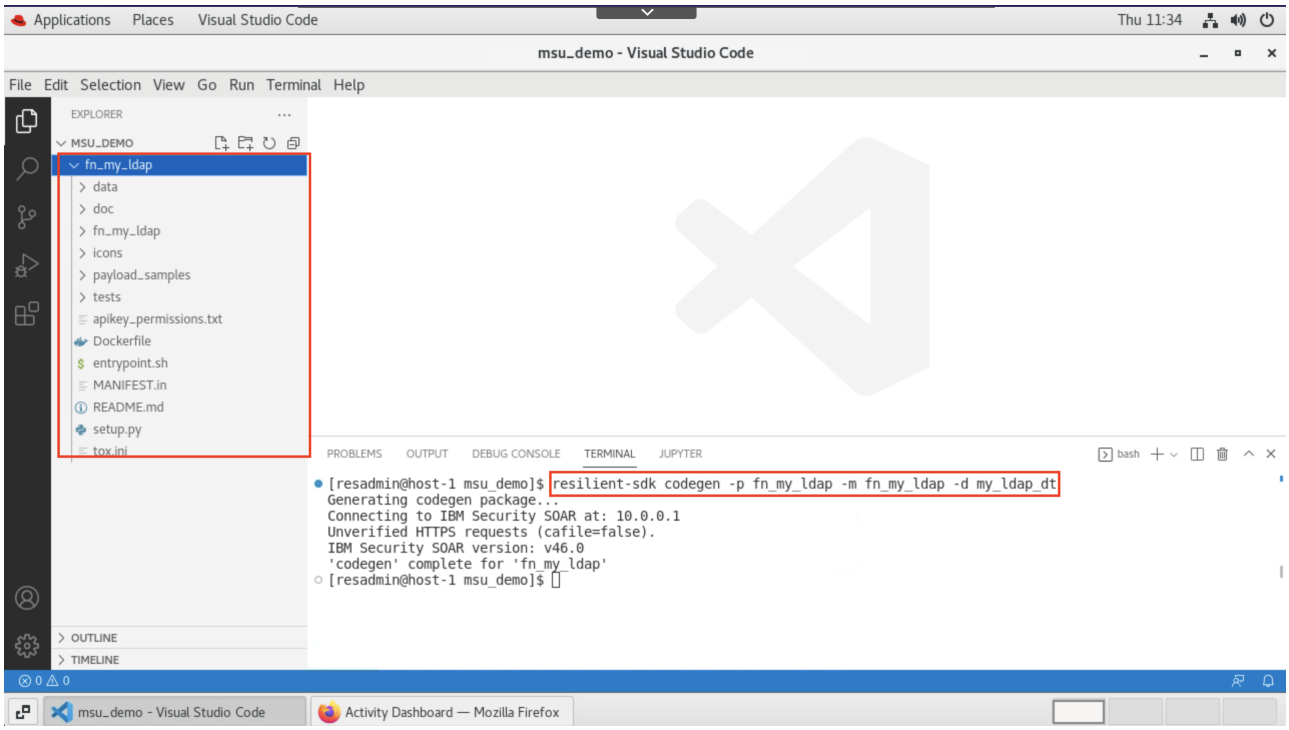


Click **Save**.

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"""[fn_my_ldap]
server=<ldap_server_here>
port=<ldap_port_here>
admin_user=<admin_user_dn>
admin_password=<admin_user_secret>
search_base=ou=people,dc=planetexpress,dc=com
"""

    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)
            }
```

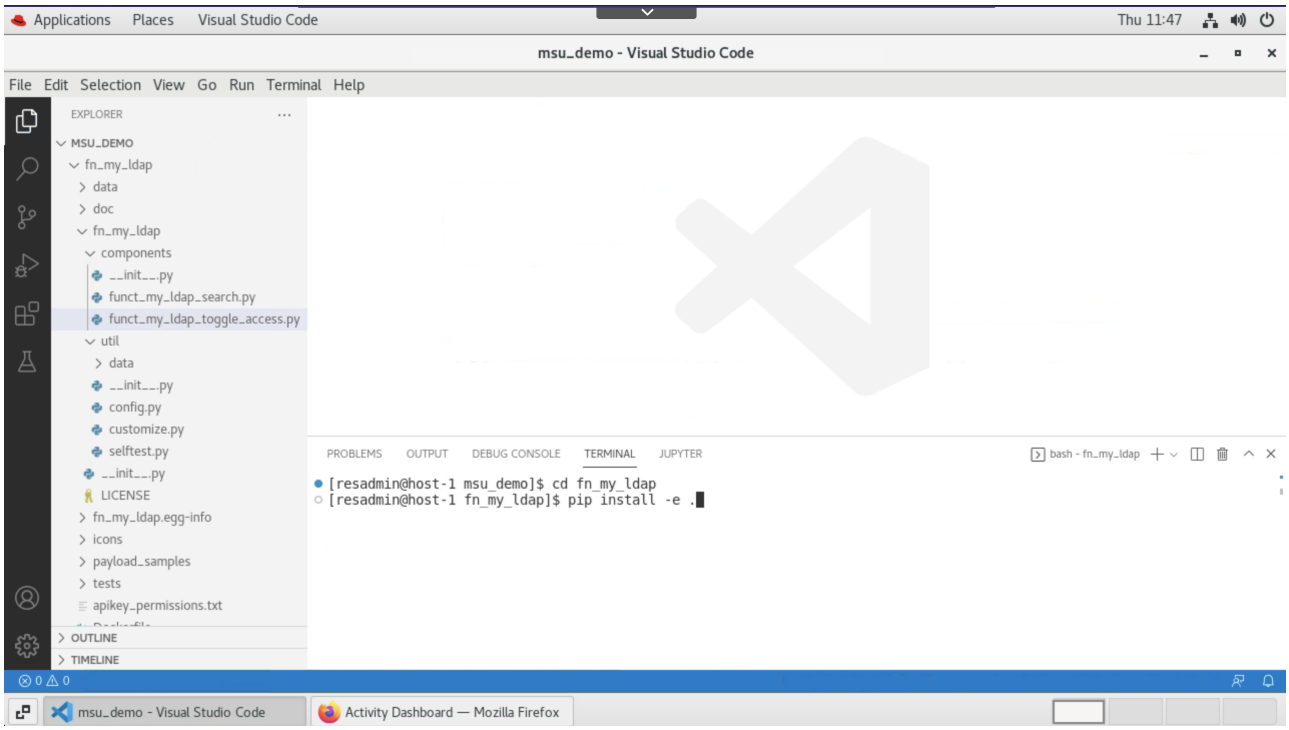
▶ fn_my_ldap/setup.py

```
#!/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.
{}:FunctionComponent".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

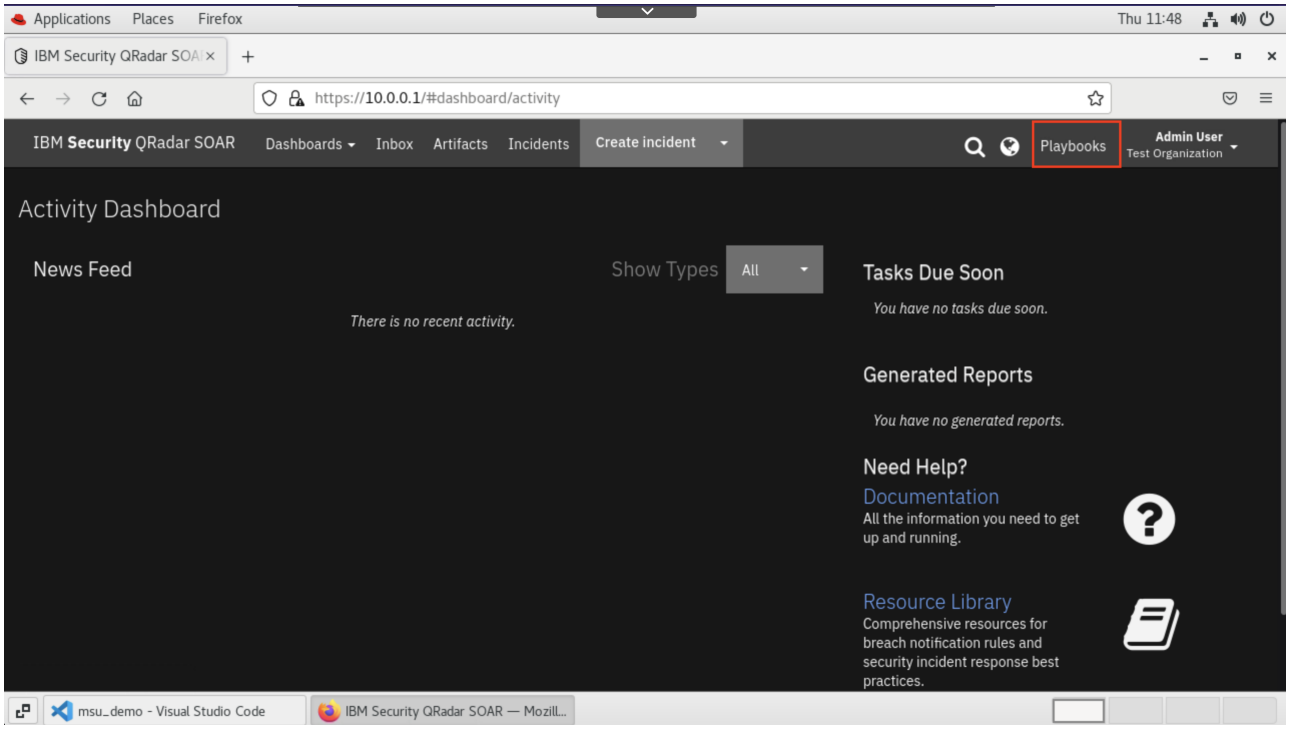
```
code ~/.resilient/app.config
```

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
```

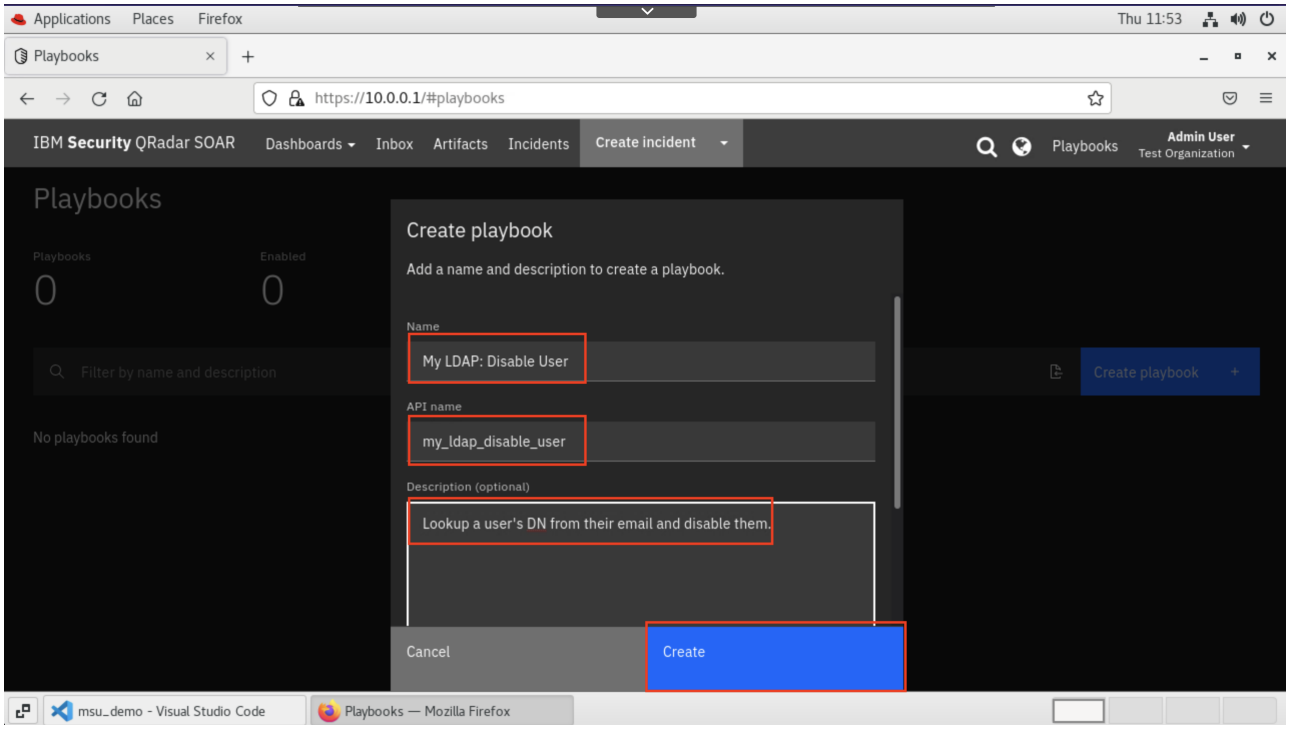
Step 9: *Create a Playbook to Test*

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

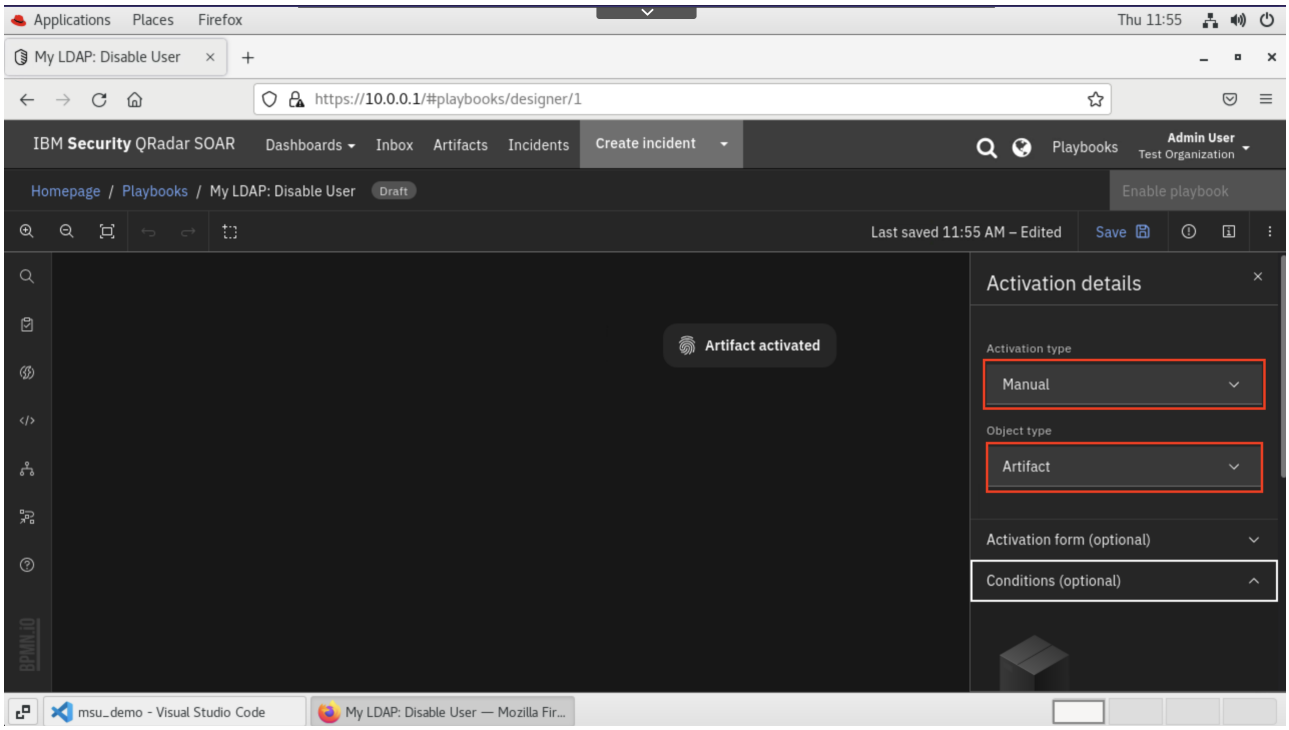


- Click **Create Playbook**.

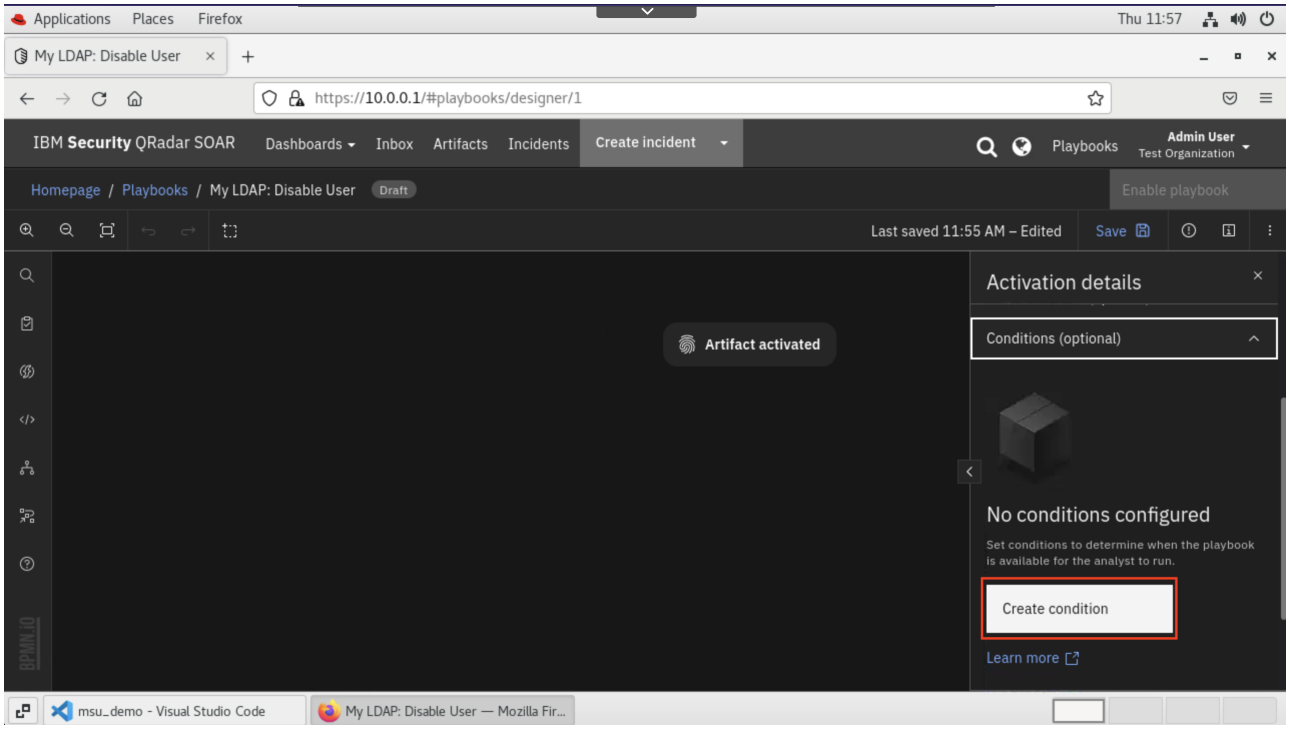
- Call it "My LDAP: Disable User":

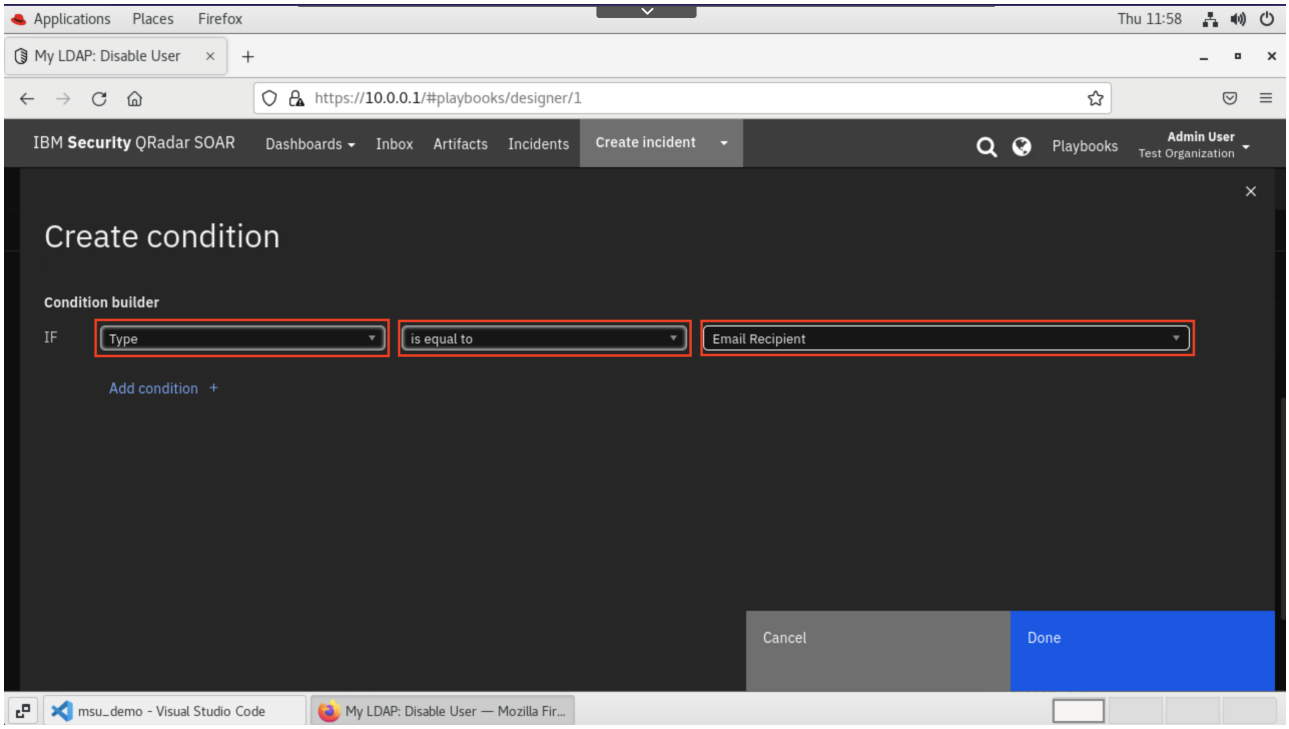


- Click **Create**.
- Set the **Activation Type** to **Manual** and the **Object** to **Artifact** as we'll be manually running this on email artifacts.



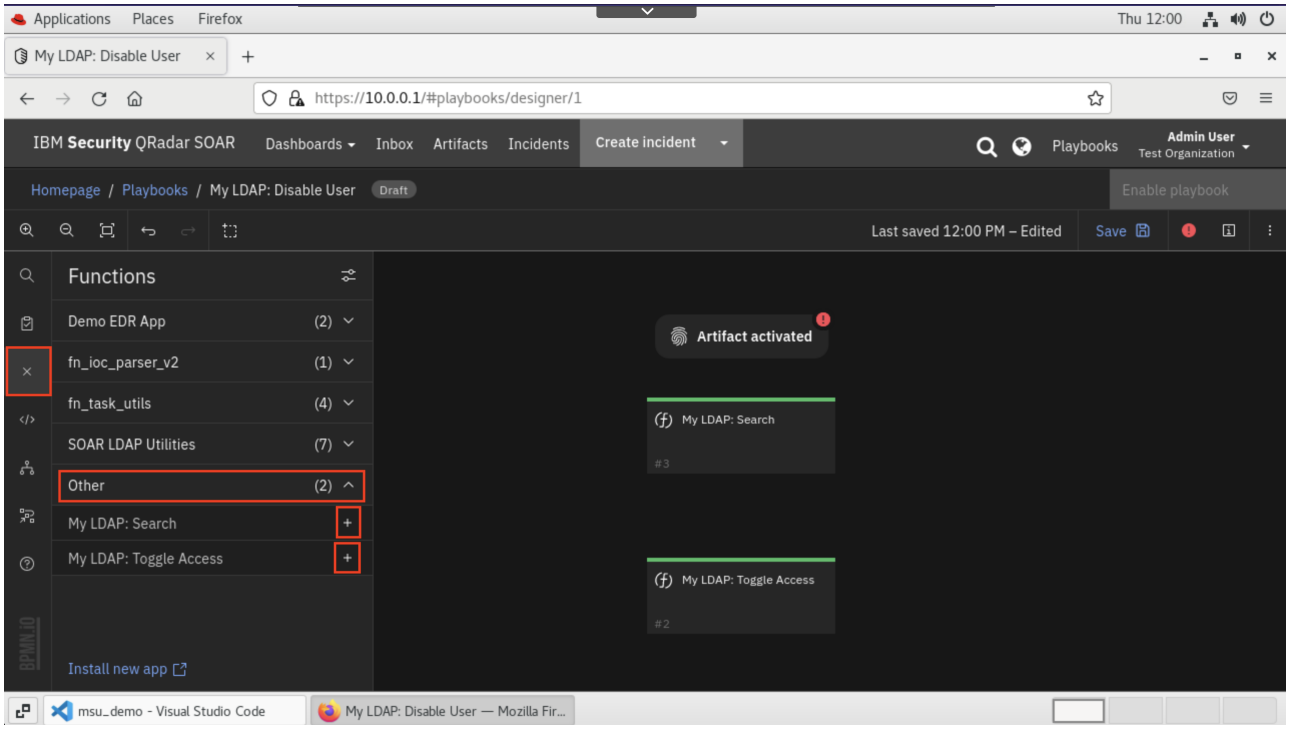
- Create an activation condition so that this only activates on `Email Recipient` artifacts. This could be changed to suit your needs. For demo purposes, we'll stick with just the one condition.





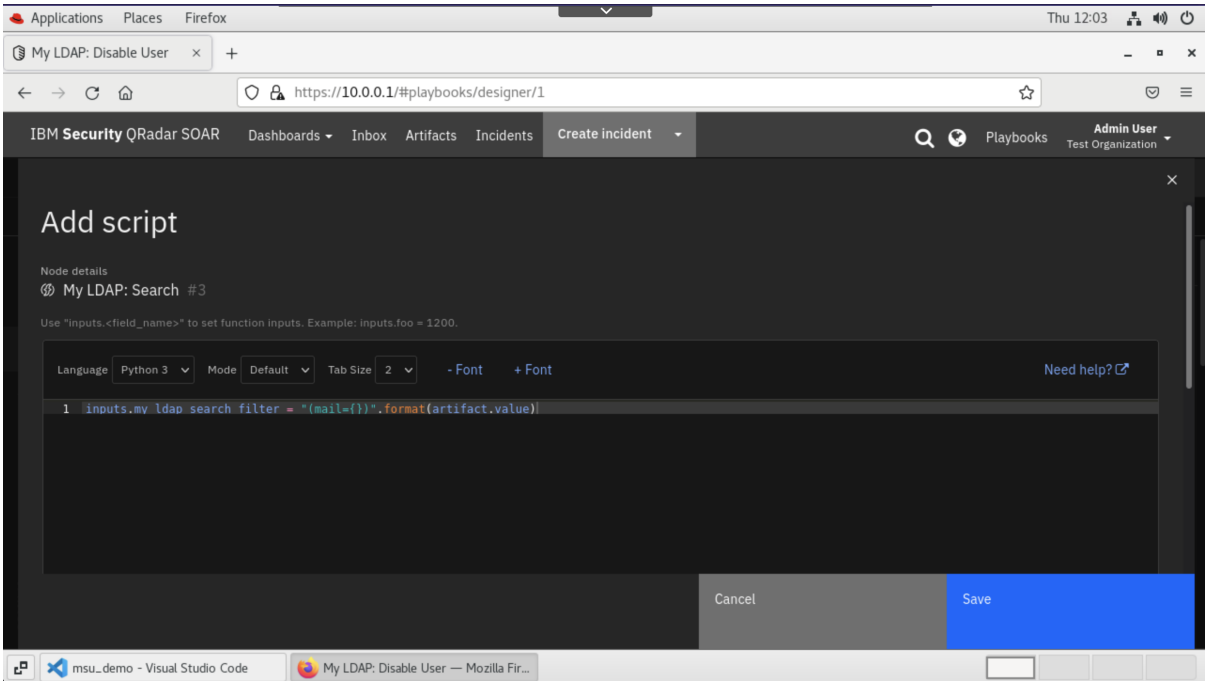
Click **Done**.

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



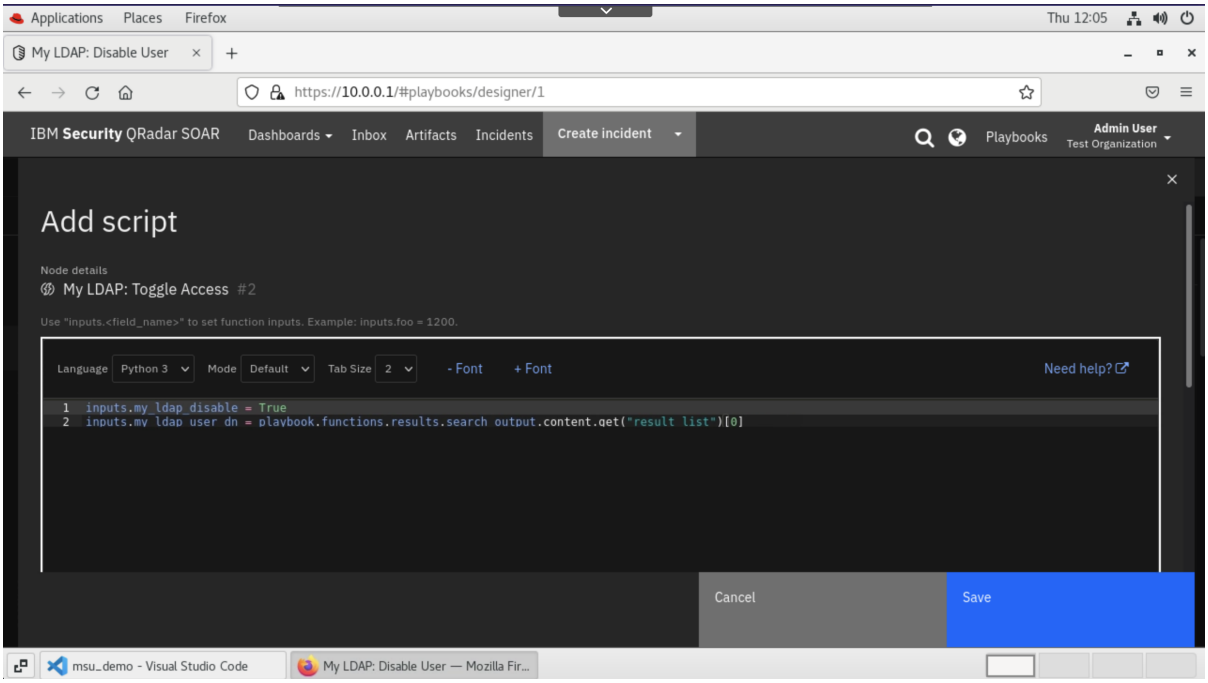
- Now we'll fill out the details for each function.
  - For the Search function, add the following input script:

```
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:

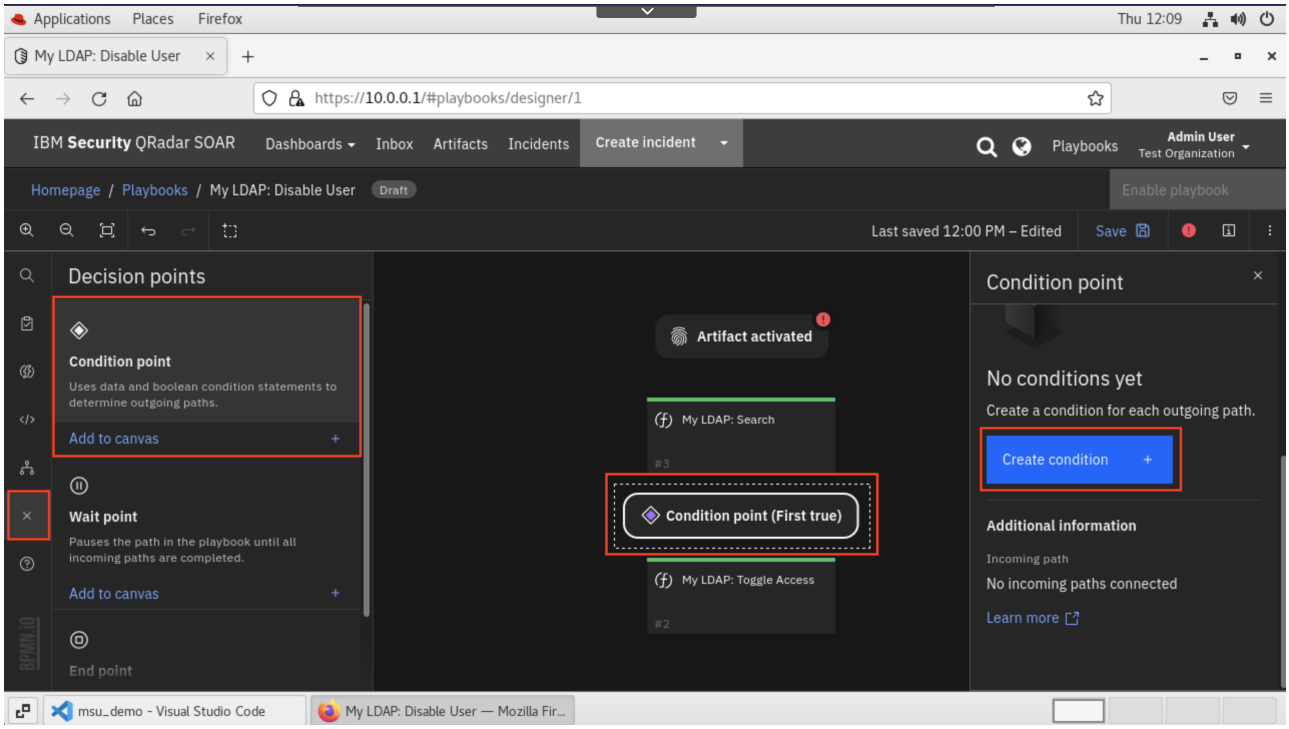
```
inputs.my_ldap_disable = True
inputs.my_ldap_user_dn =
playbook.functions.results.search_output.content.get("result_list
")[0]
```



Click **Save**.

- Name the output of the Toggle function `toggle_output`.

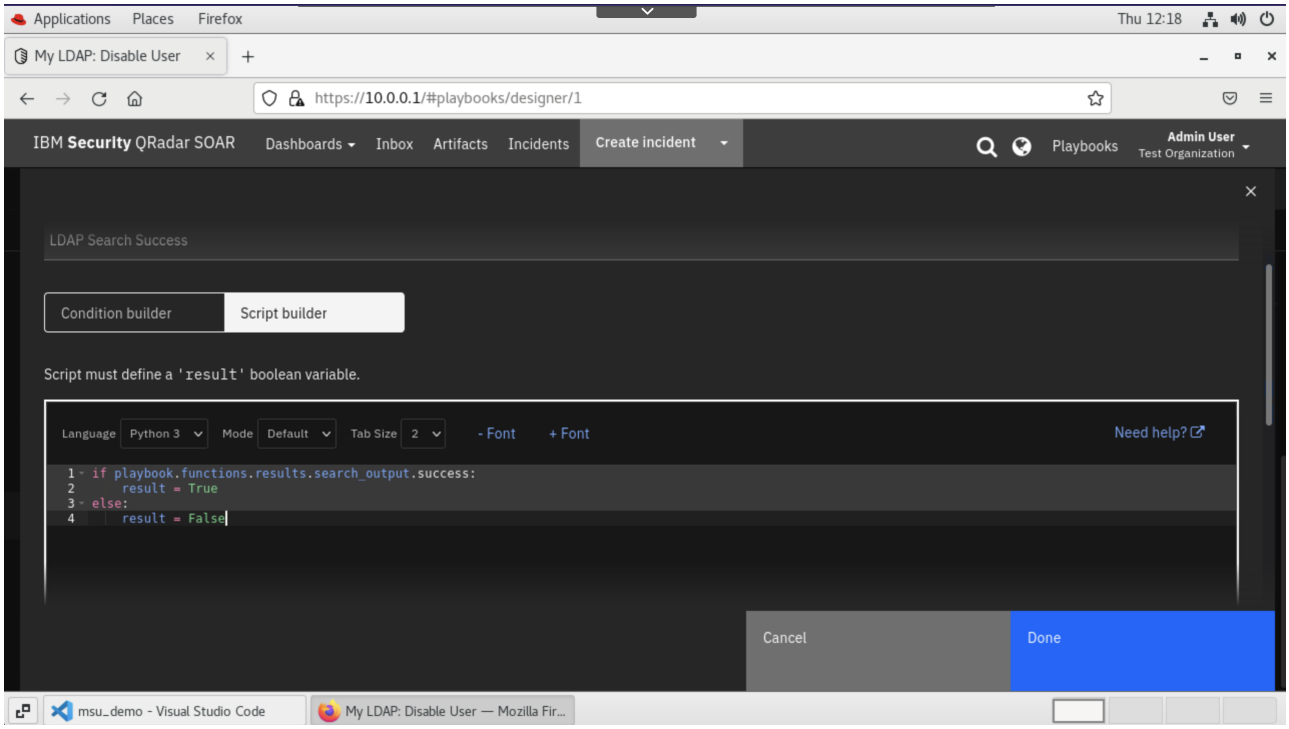
- Now we need to build the flow of the playbook. There needs to be a decision point where based on the output of the first function, we run the second function. If there was no Distinguished Name found in the search function, we don't want to run the toggle function. To accomplish this we will add a 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
```

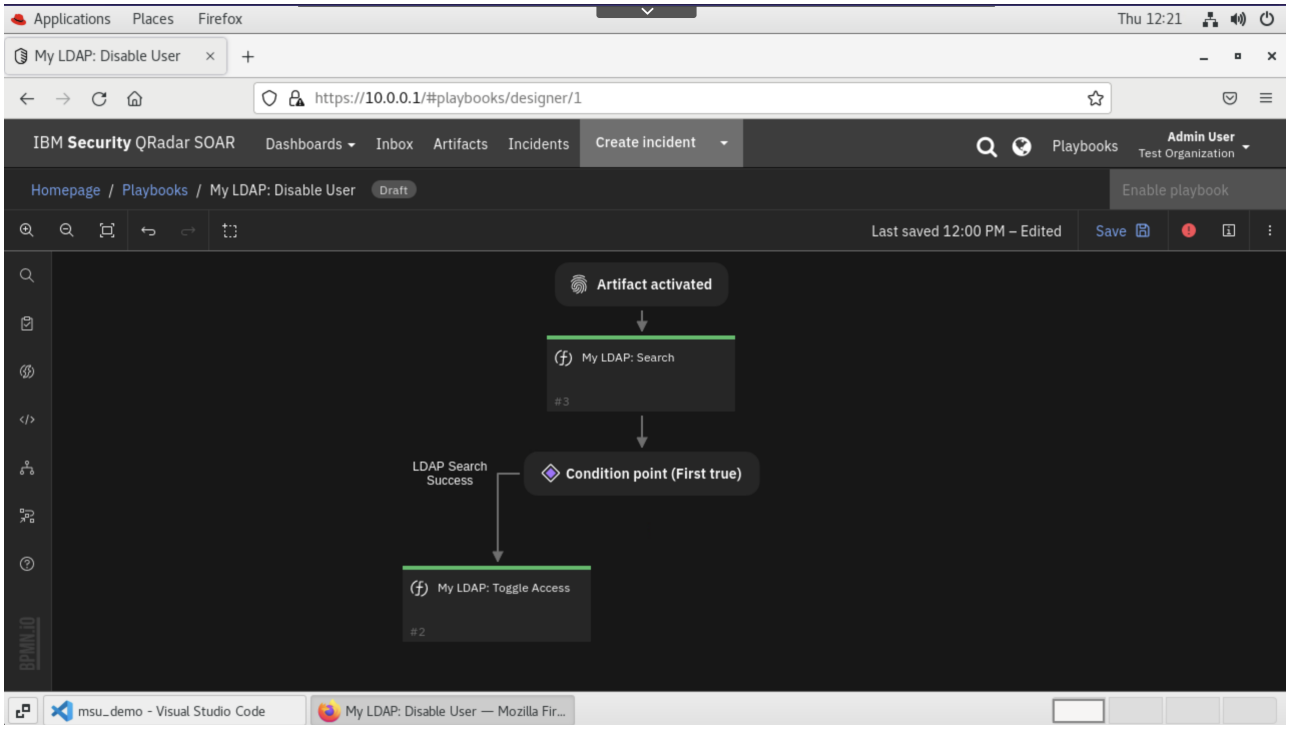
```
else:
    result = False
```



Click **Done**.

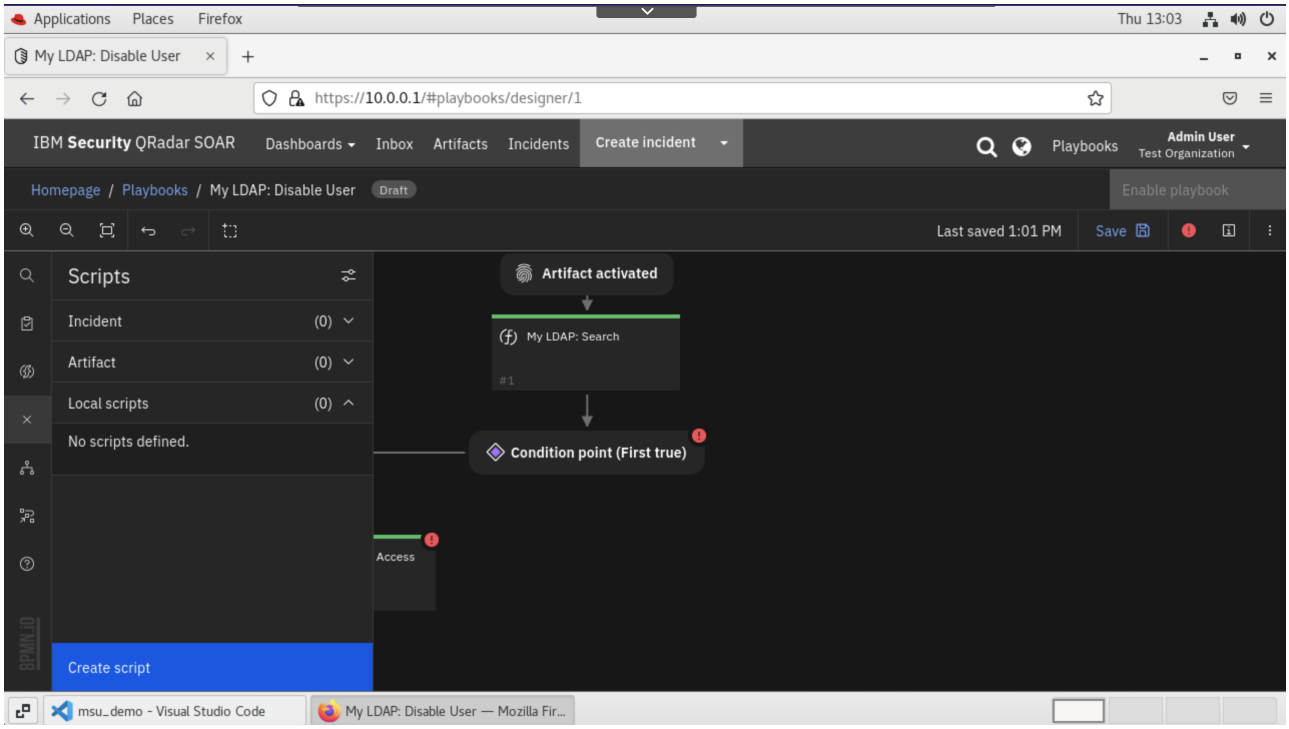
> Note: `result` must be defined for all branches of the condition to properly function.

- 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:



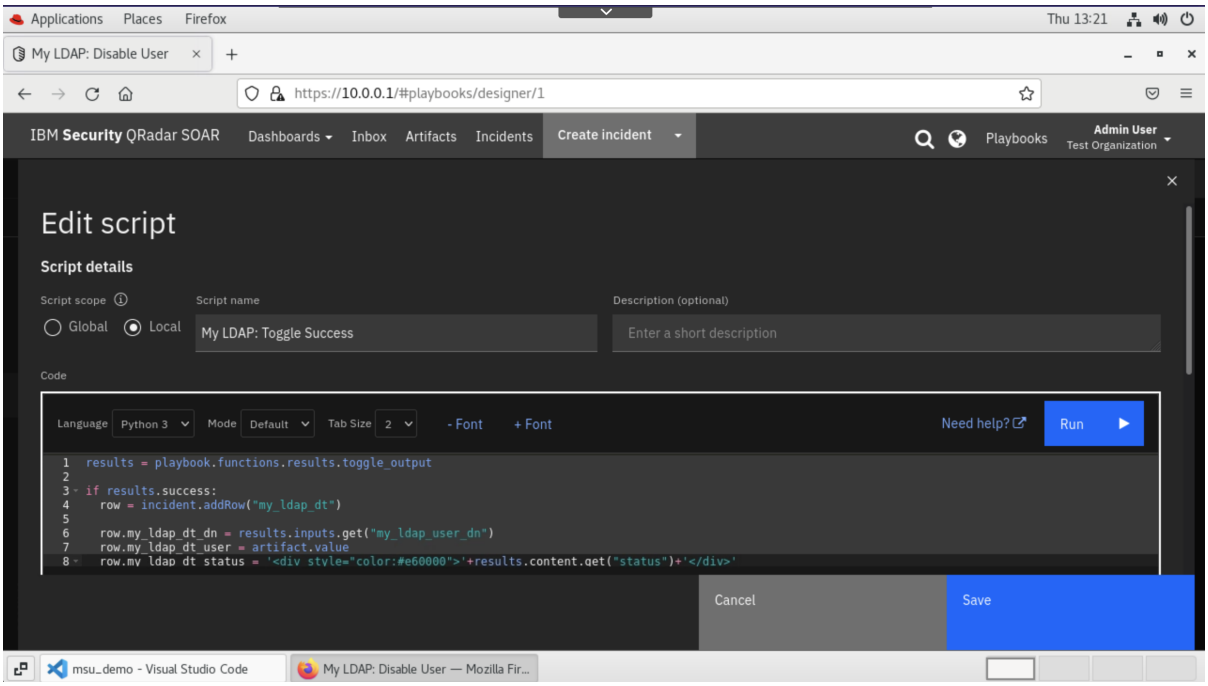
- Now we must deal with two things: how to end the playbook in our two branches that we've created.

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



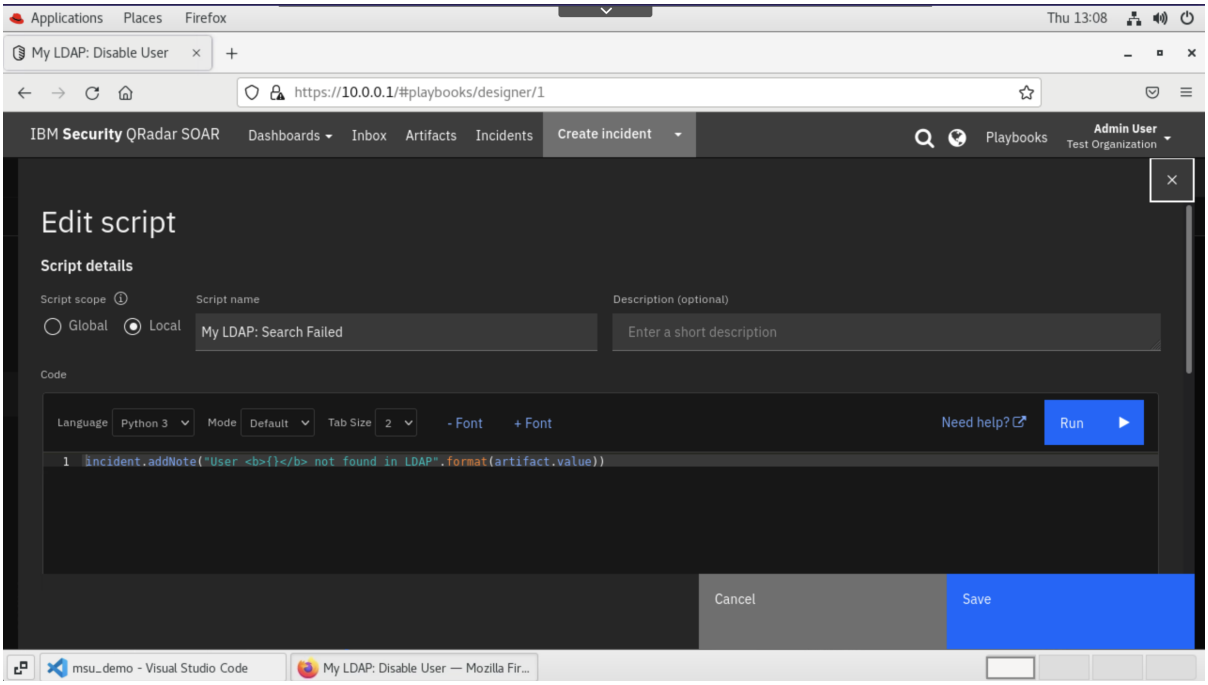
  - Local Script #1: **My LDAP: Toggle Success**

```
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>'
```

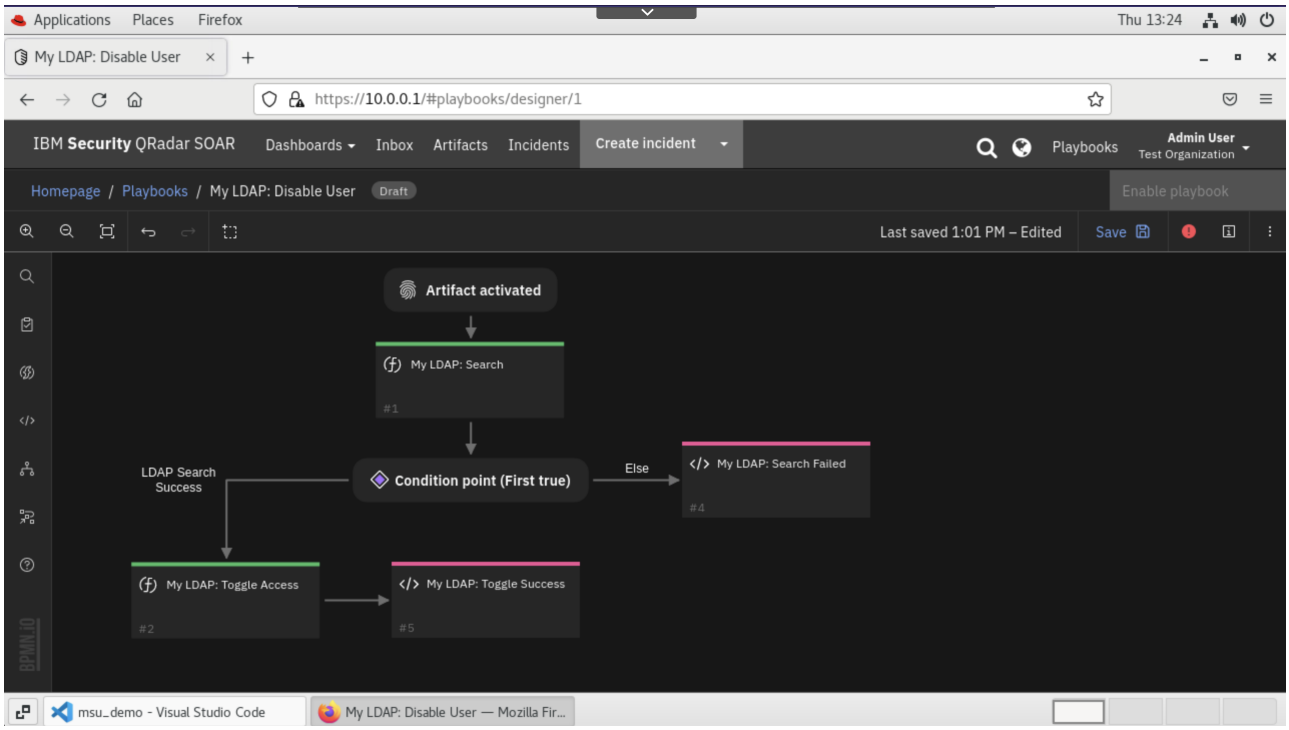


  - Local Script #2: **My LDAP: Search Failed**

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

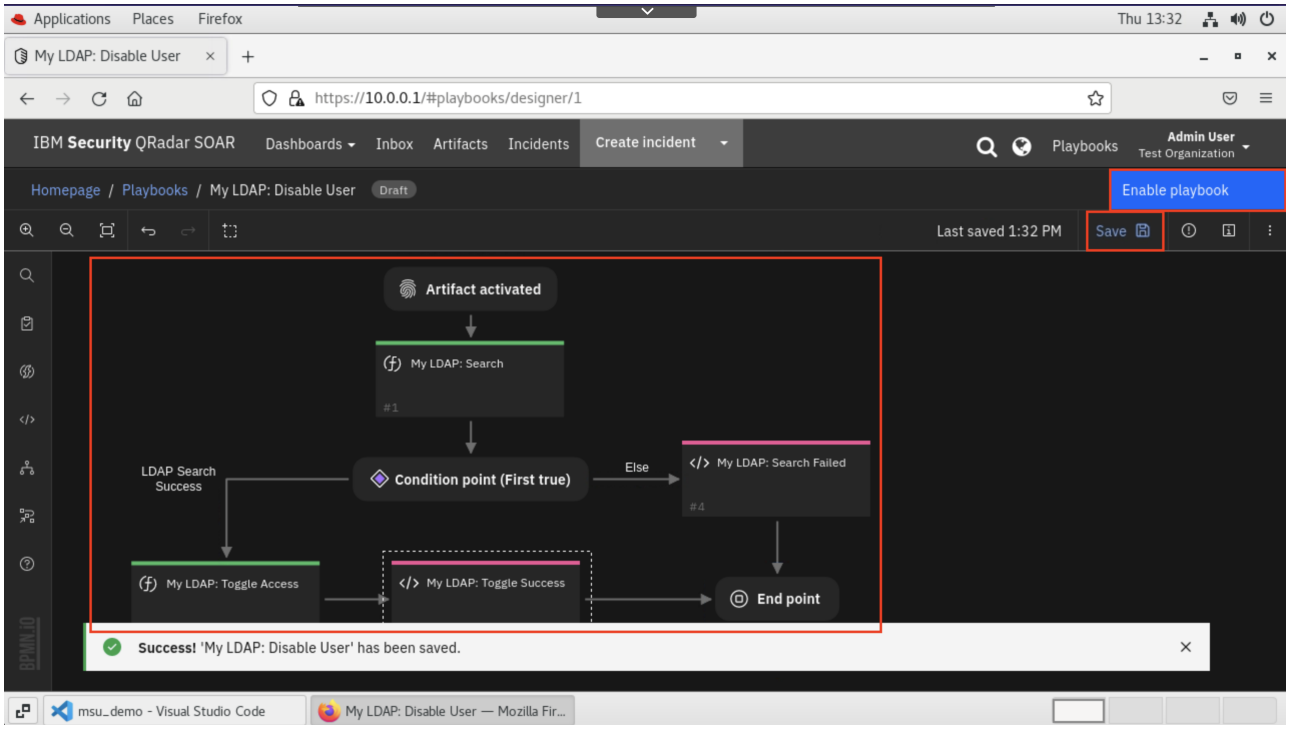


- Add the two scripts to the canvas and attach to the rest of the playbook as shown:



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

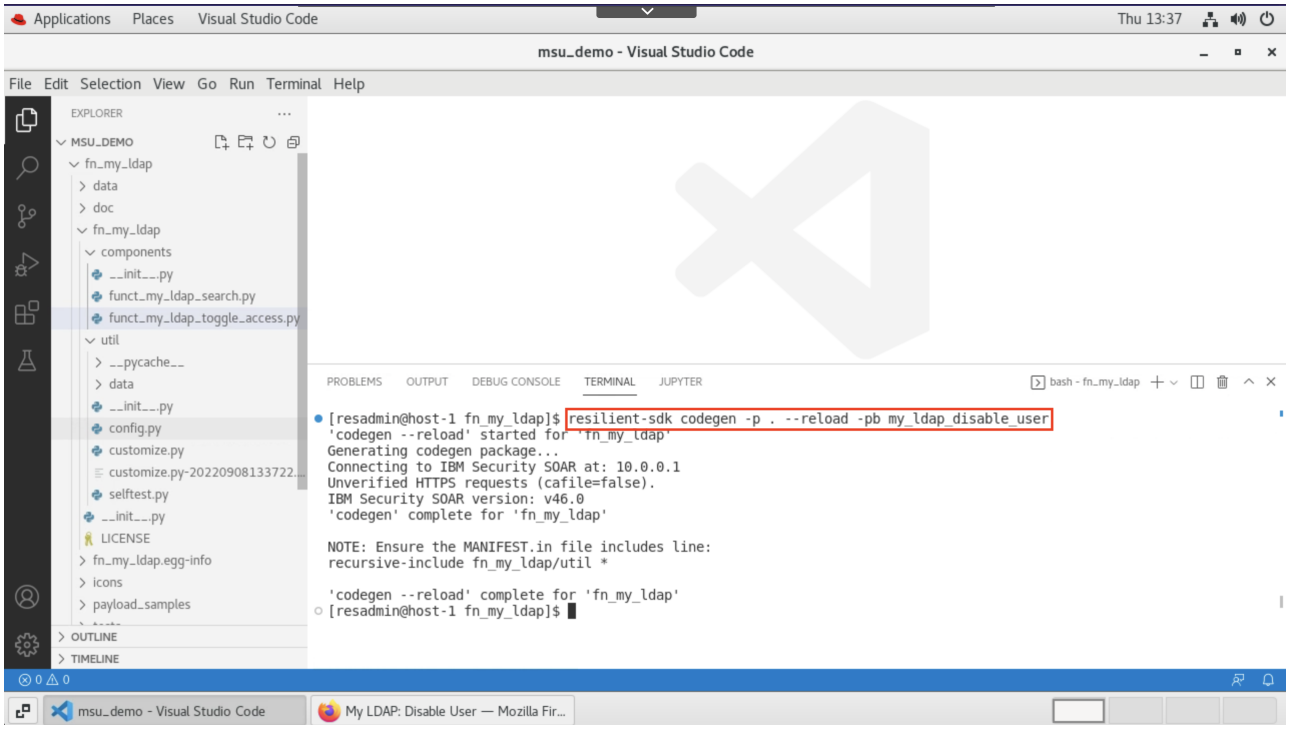
- Click **Save** and **Enable Playbook**.



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.

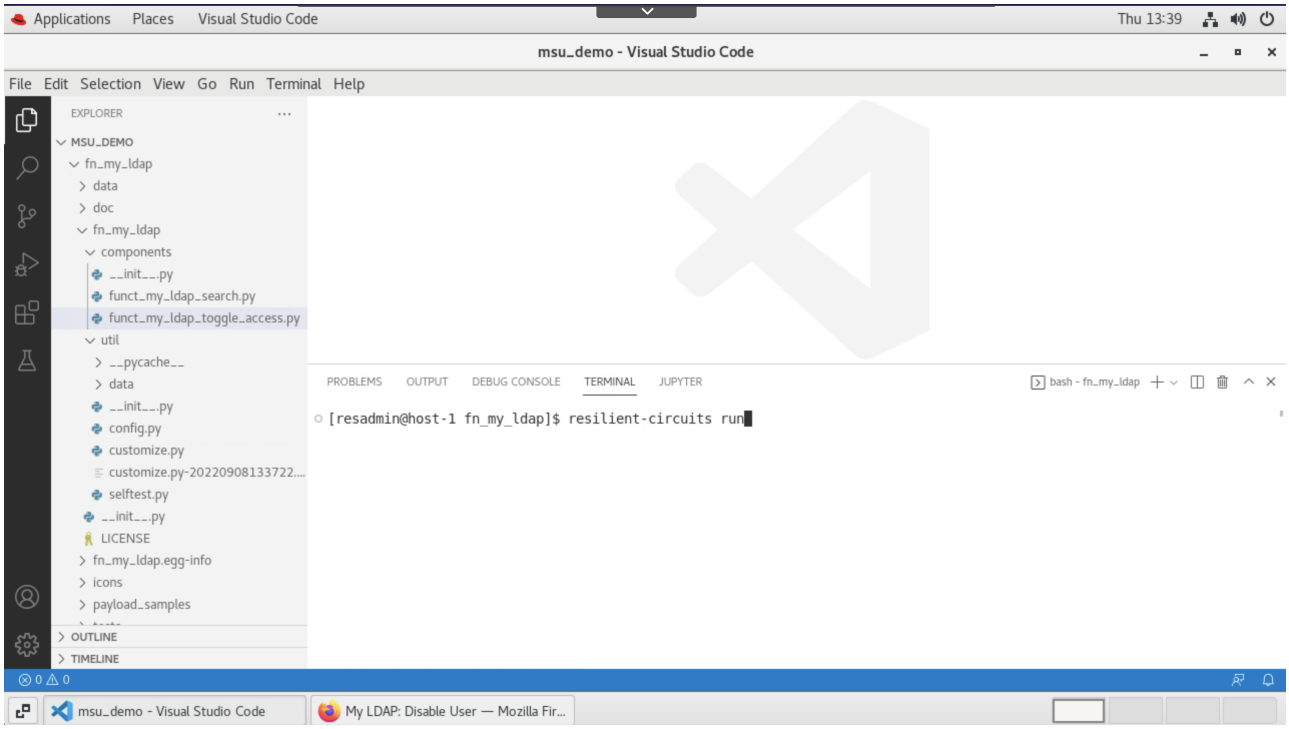
```
resilient-sdk codegen -p . --reload -pb my_ldap_disable_user
```



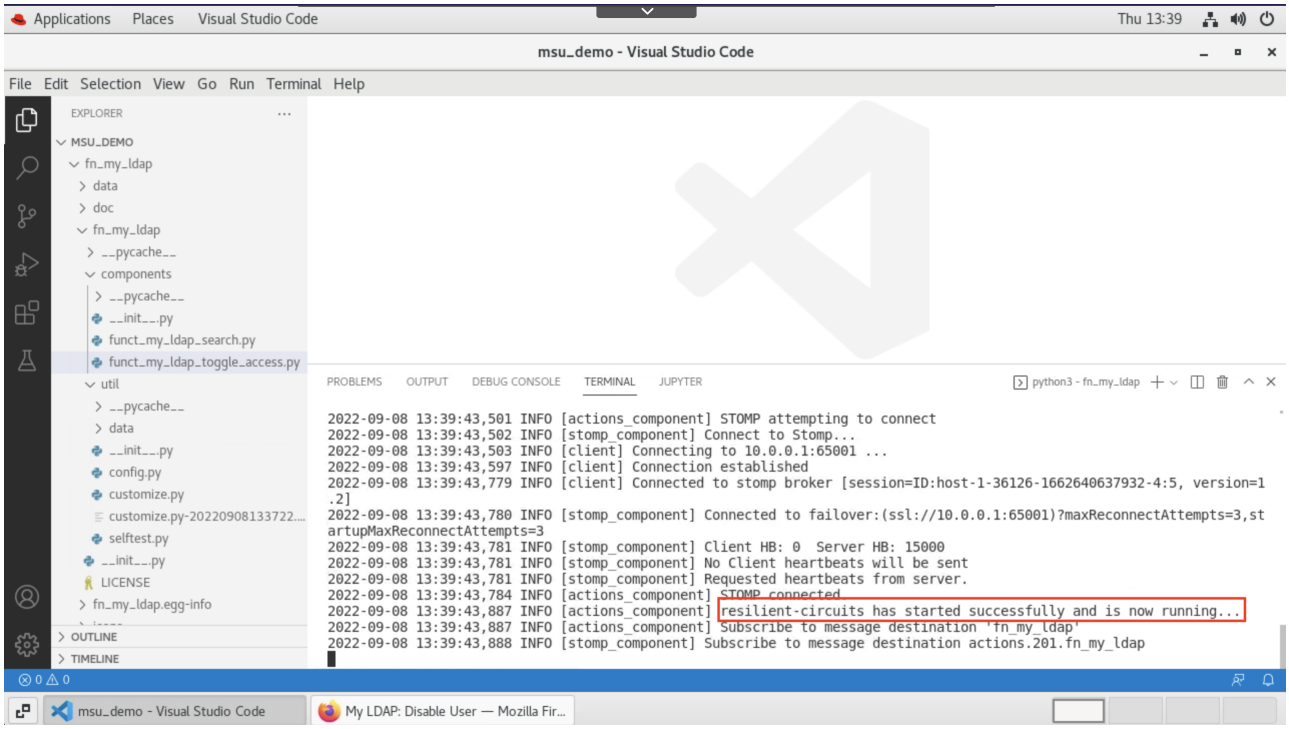
> Note: you should be generous in running `codegen --reload` as much as you want. It will never hurt, and it can be painful if you forget to reload any UI changes you've made.

- 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
```



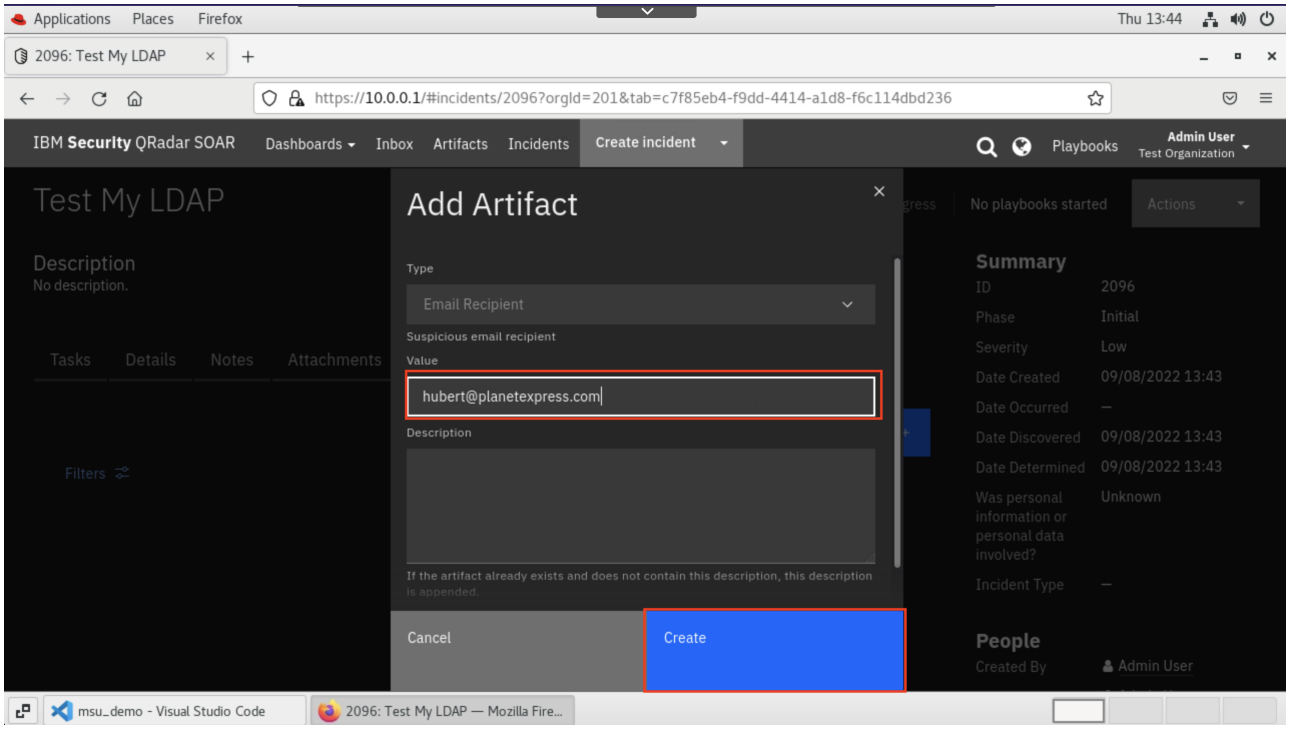
Keep any eye out for *"resilient-circuits has started successfully and is now running..."*



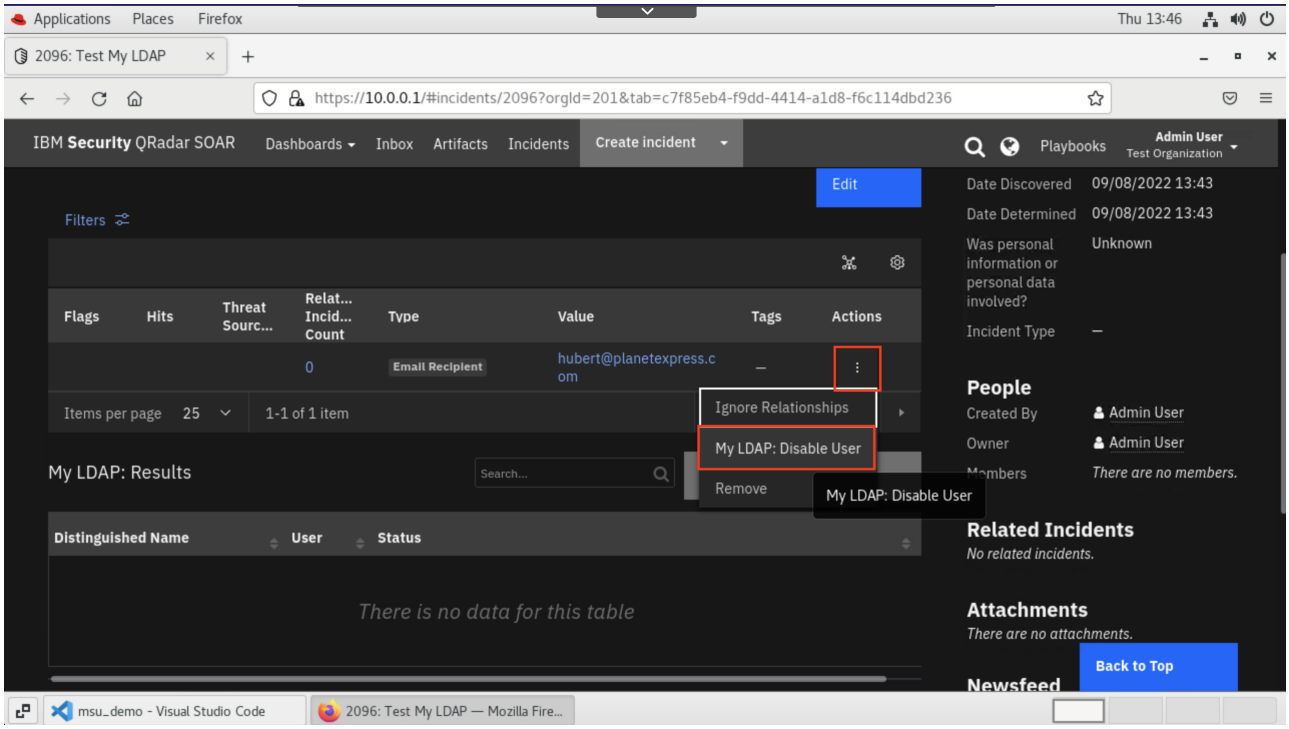
> Note: this will only work properly if you already used `pip` to install the app as at the end of step 8.

Step 11: *Test*

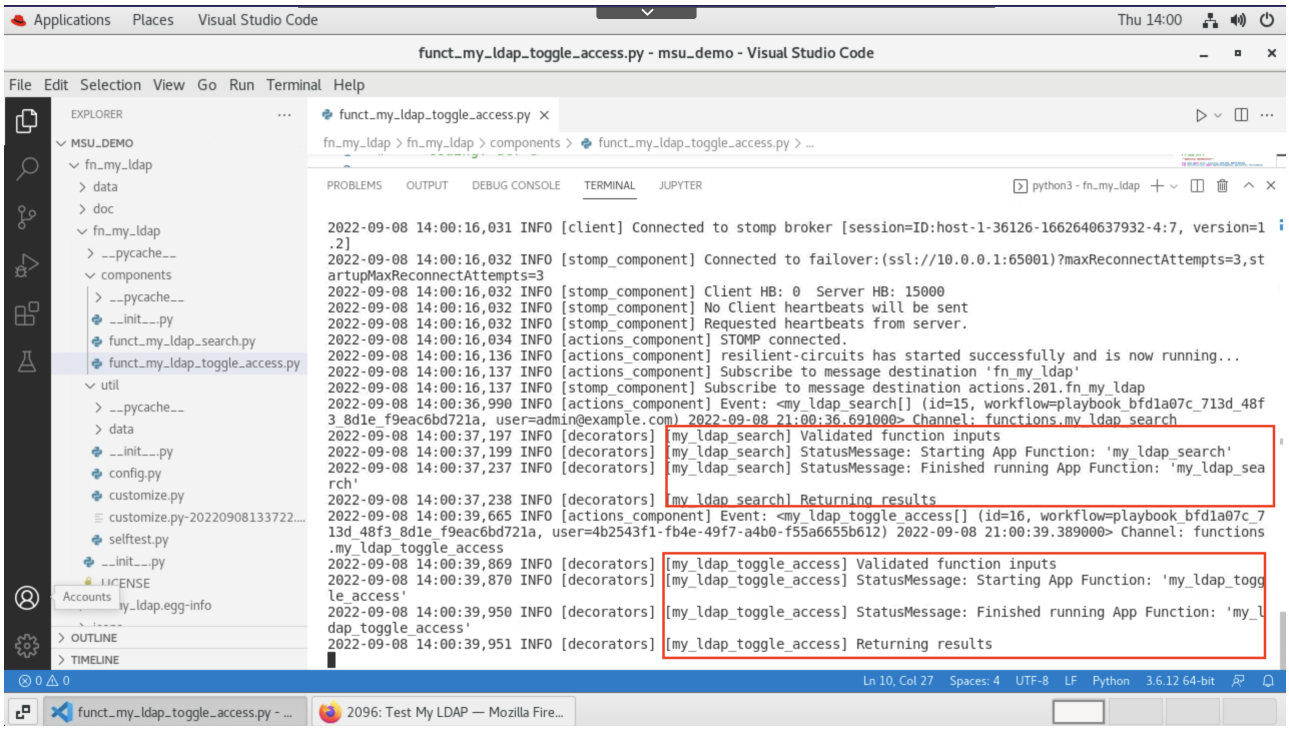
- Return to the UI and exit out of the playbook. Create a new incident with default values and add an Email Recipient artifact to the incident with value `hubert@planetexpress.com`.



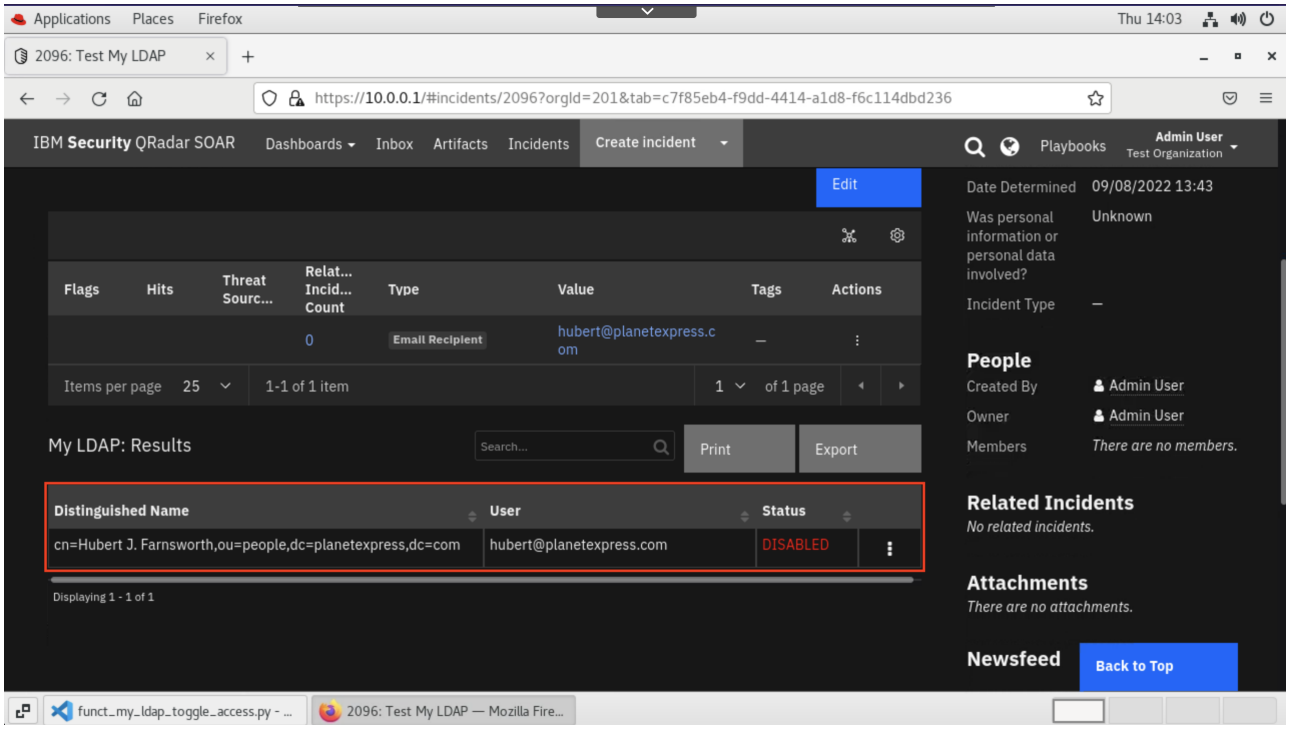
- Select the action **My LDAP: Disable User** from the list of available actions for the artifact.



- 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`:



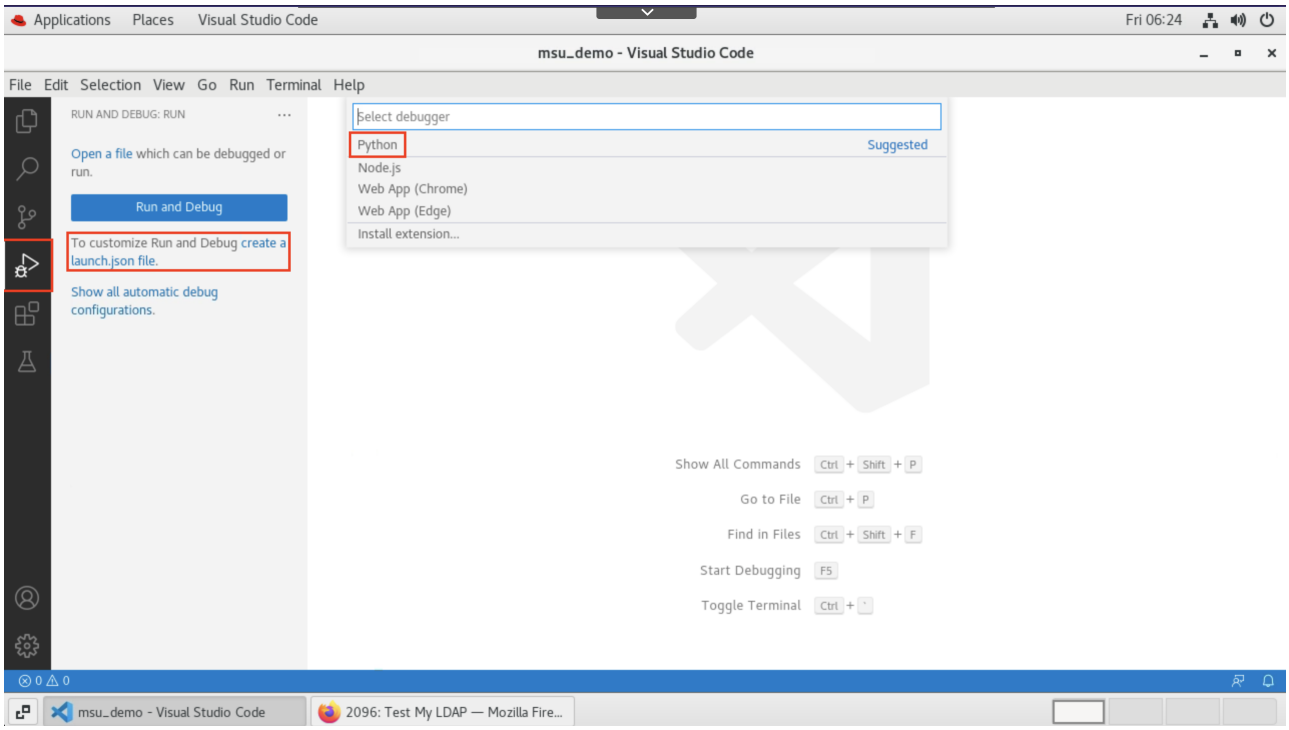
- Refresh the artifacts tab, and you should see the following in the datatable:



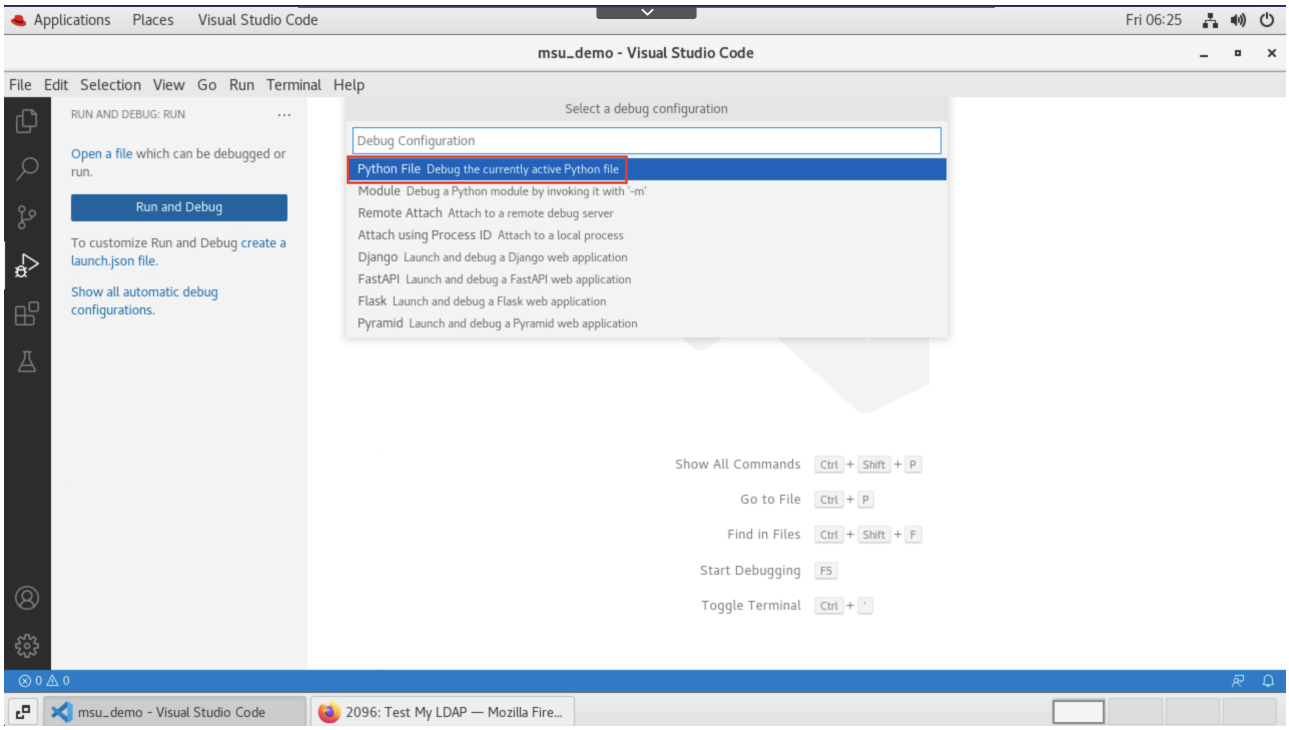
- To test the other path where a note is posted when the email is not found, enter a new artifact with a random value for the email recipient and run the playbook again. Notice that nothing is added to the datatable and a note is added to the incident.

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

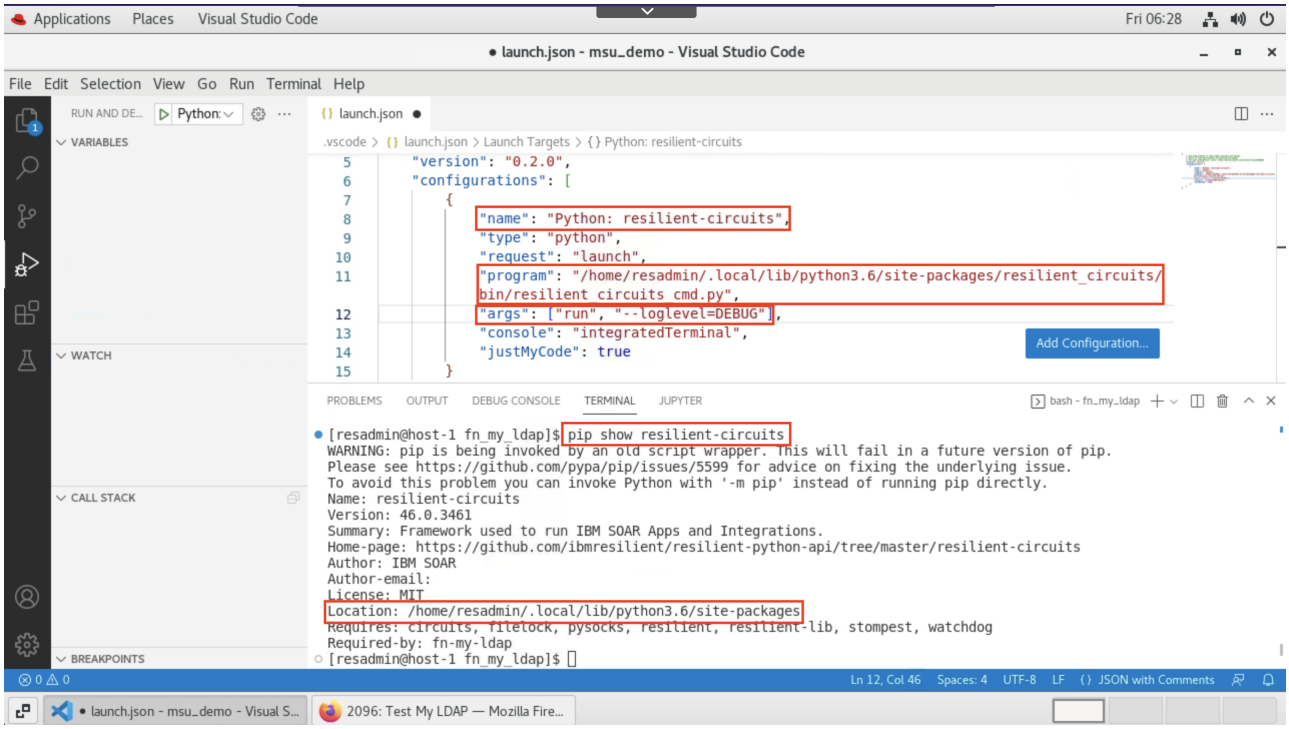
- Stop `resilient-circuits` (`CTRL + C`).
- On the left of the VS Code panel, select "Run + Debug". Then click "create a launch.json file" and select "Python" from the list of debuggers:



- Select "Python File":



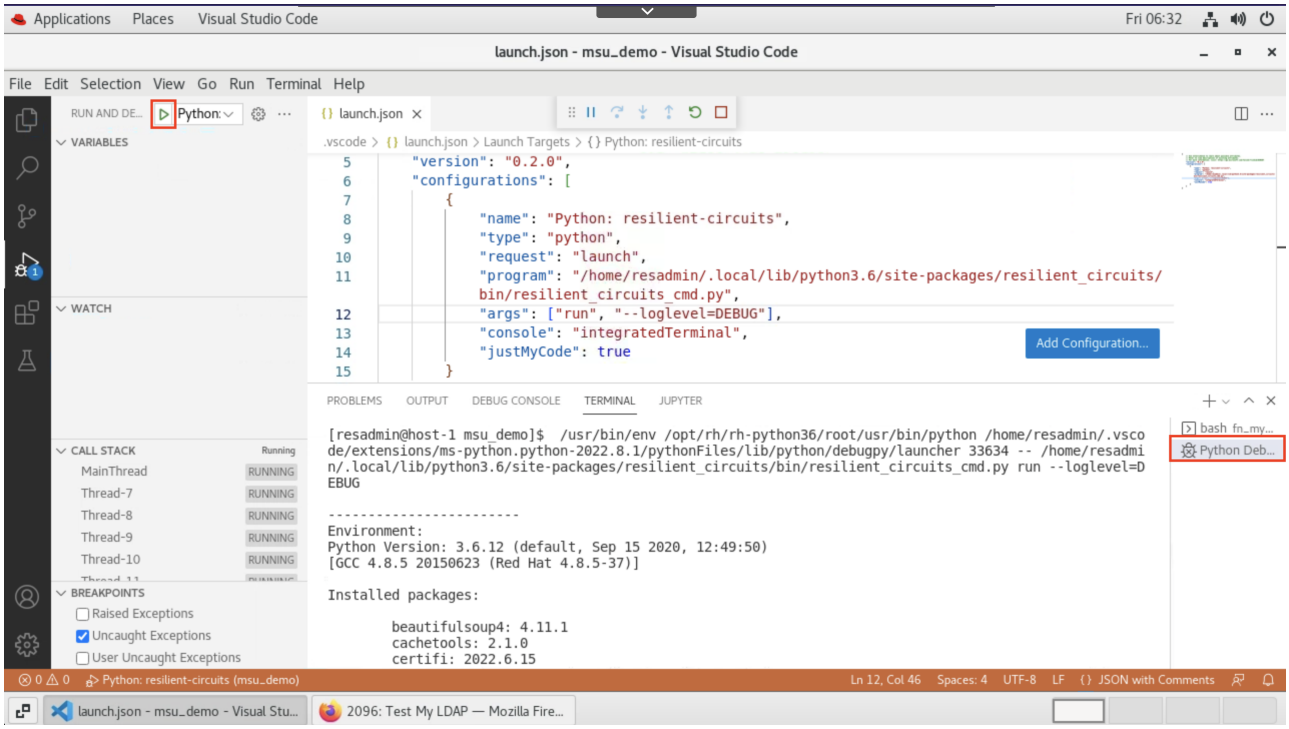
- In the newly created `launch.json` file, enter the details as seen in the screenshot below:



> 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`.

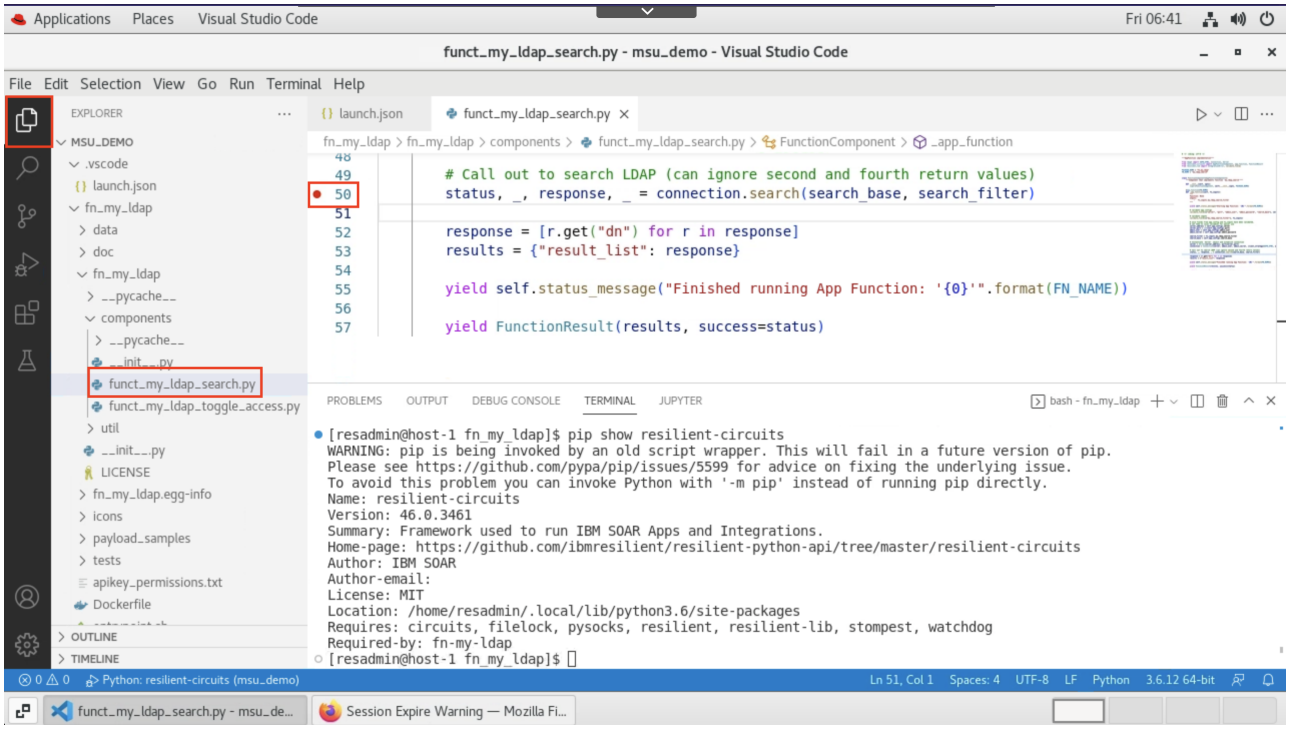
> Be sure to add the `args` section to the launch file.

- 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:



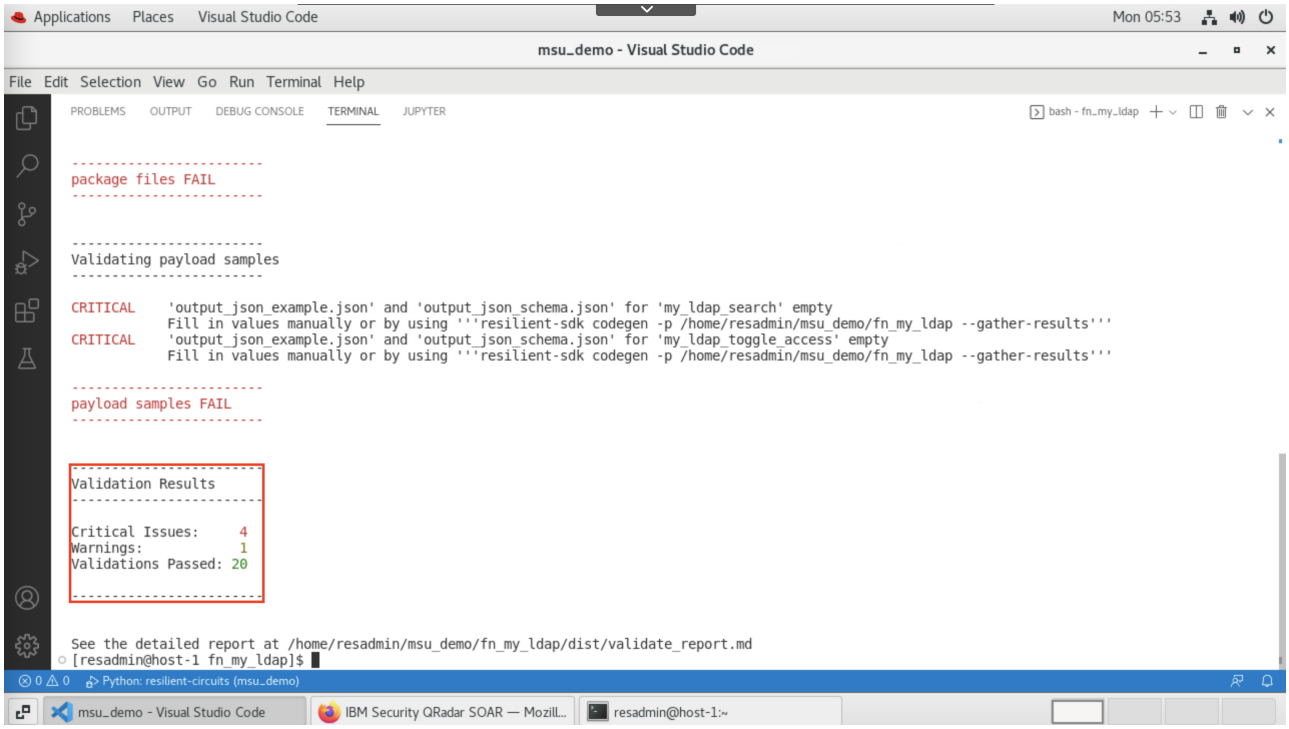
- From the UI, run the playbook and notice that the breakpoint is hit. You can now step around in the function and see session variables.
- Feel free to play around with this, it can be a very powerful functionality.

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.



- 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`

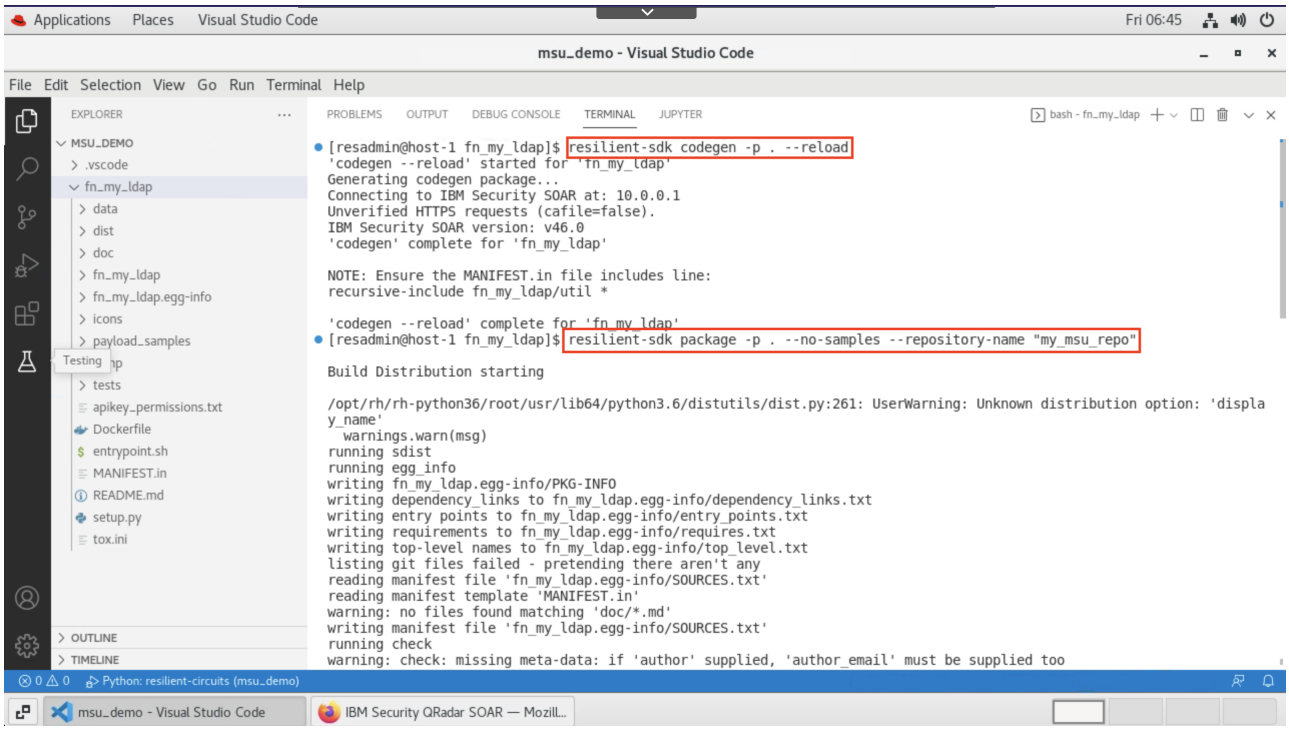
Step 14: *Package*

- 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
```

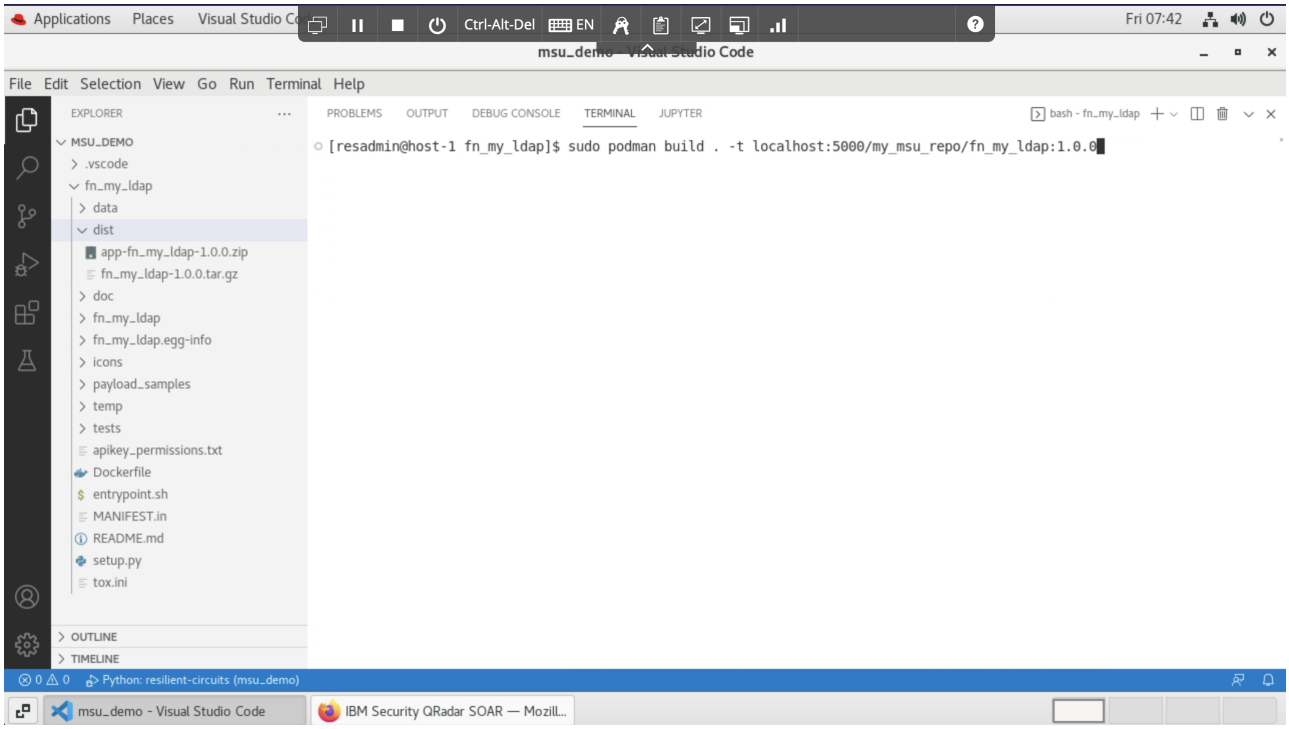


- Notice now that a `dist` directory has been created with a file called `fn_my_ldap/dist/app-fn_my_ldap-1.0.0.zip`. This is what we will use to install the app in step 15.

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

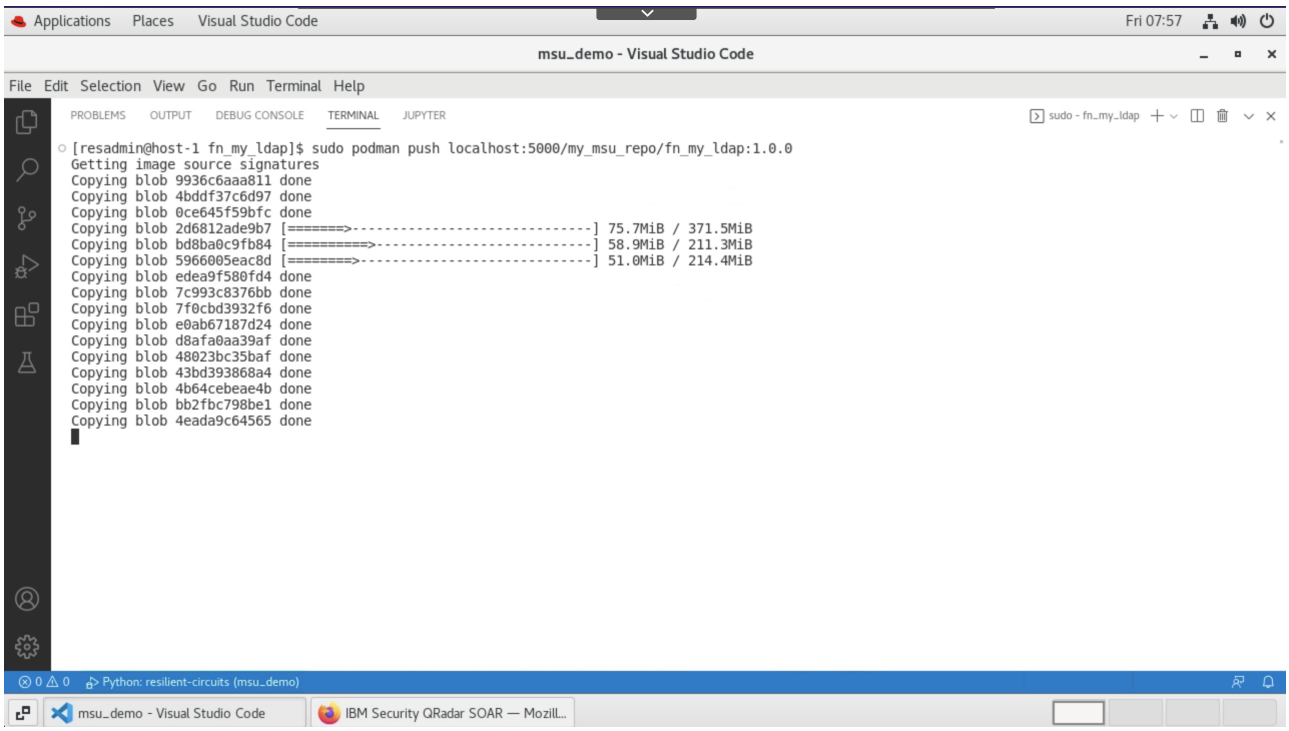
```
sudo podman build . -t localhost:5000/my_msu_repo/fn_my_ldap:1.0.0
```



> 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`.

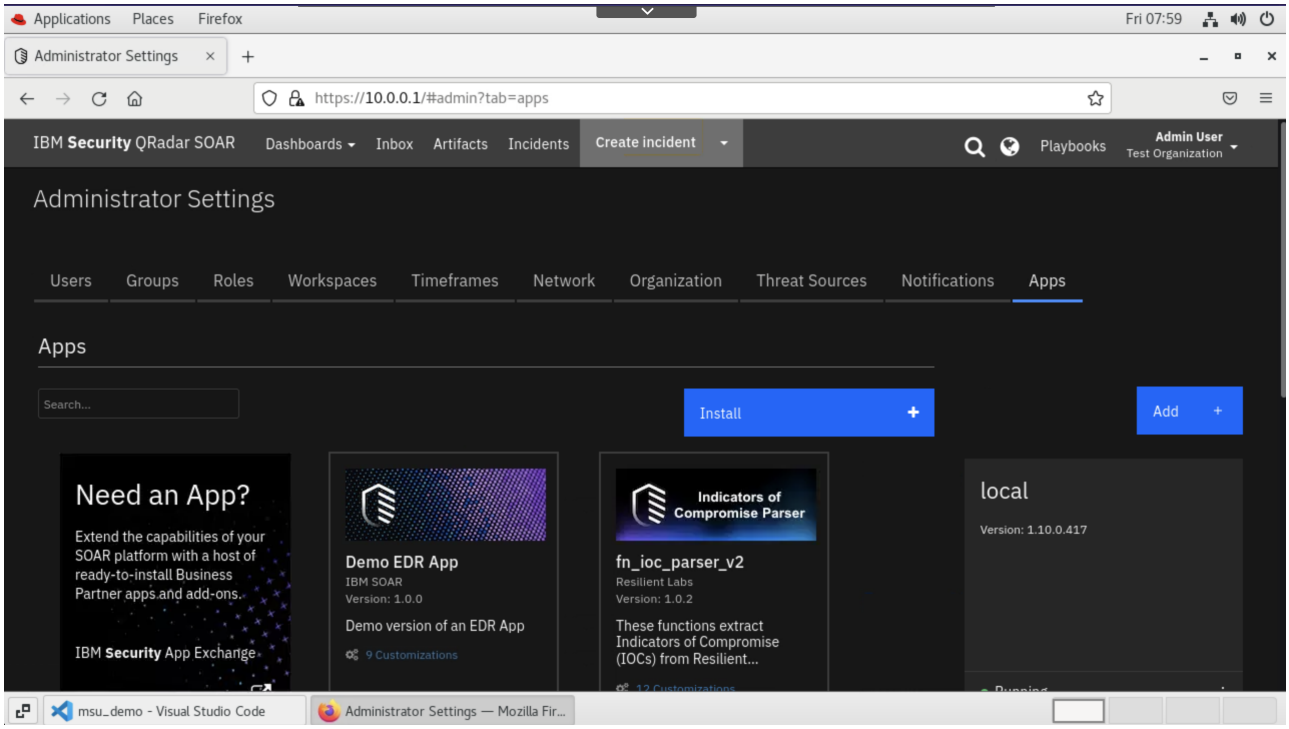
- Push the newly build container to the local registry:

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

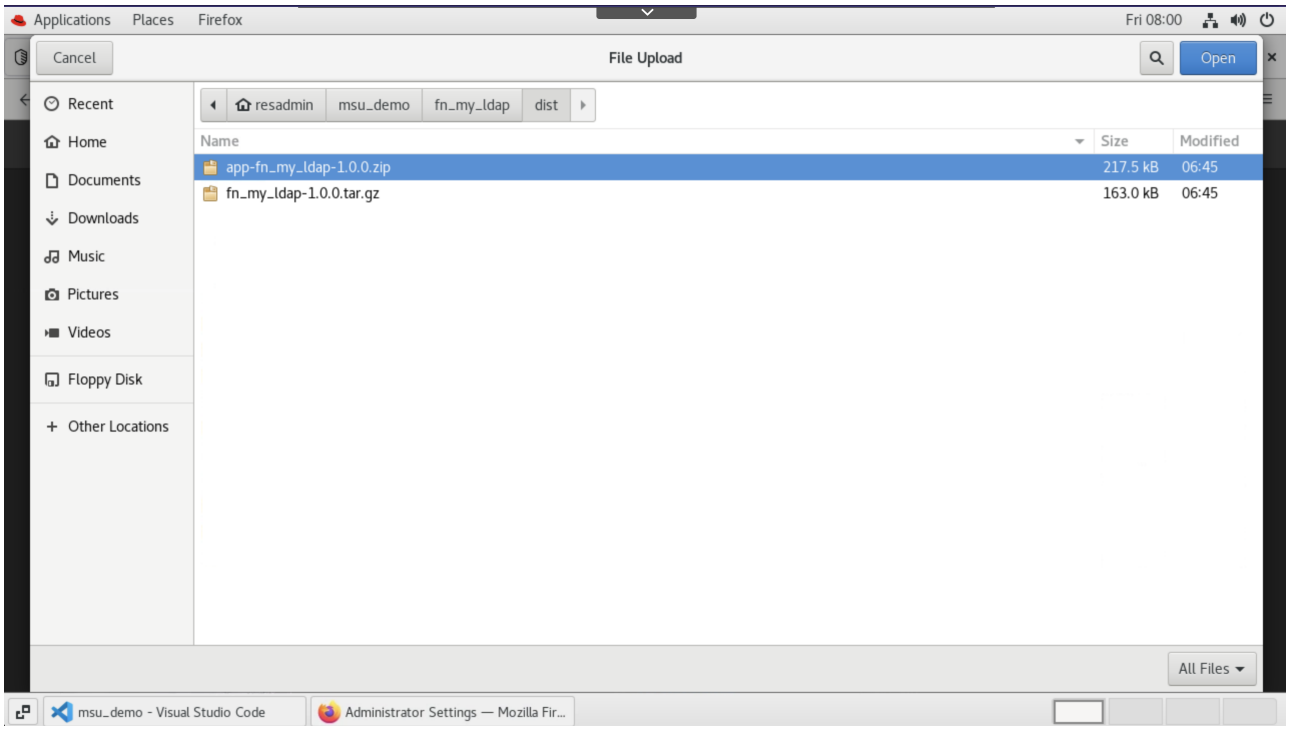


Step 16: *Install with App Host*

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

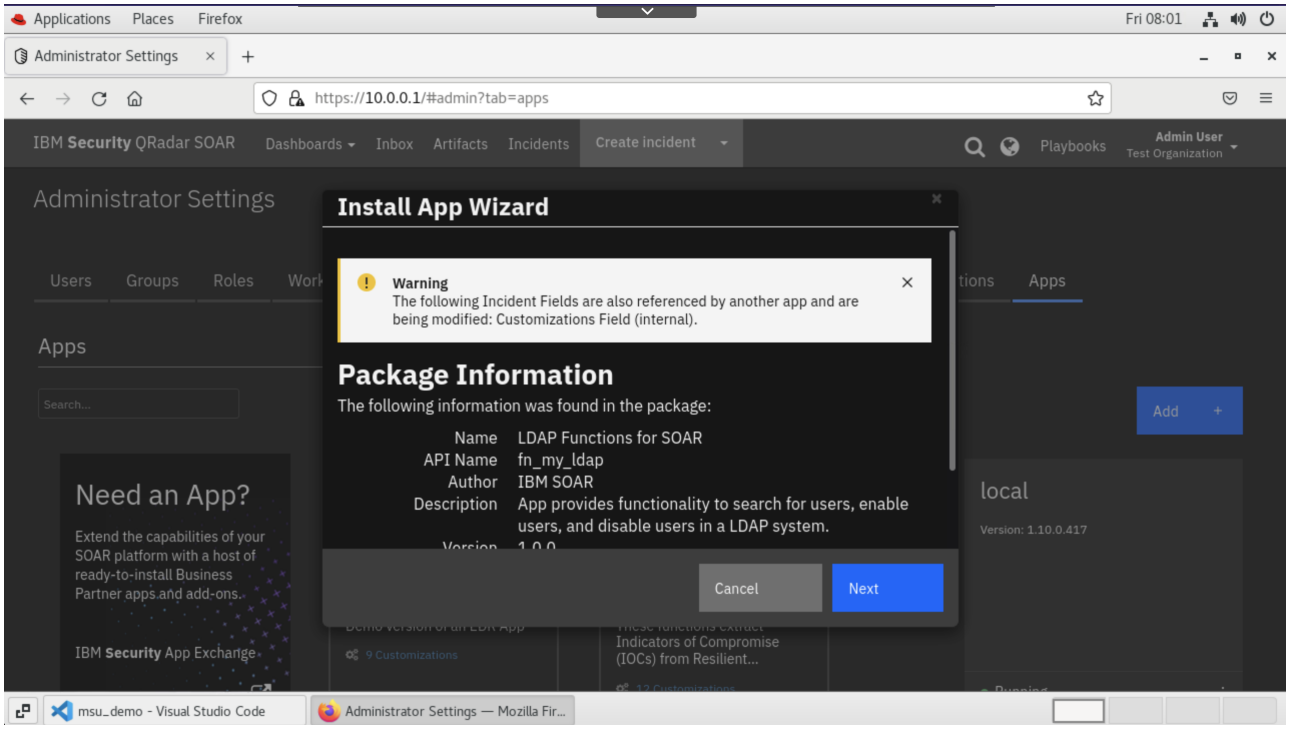


- Install your app by clicking **Install** and finding the `fn_my_ldap/dist/app-fn_my_ldap-1.0.0.zip` file on your local machine.

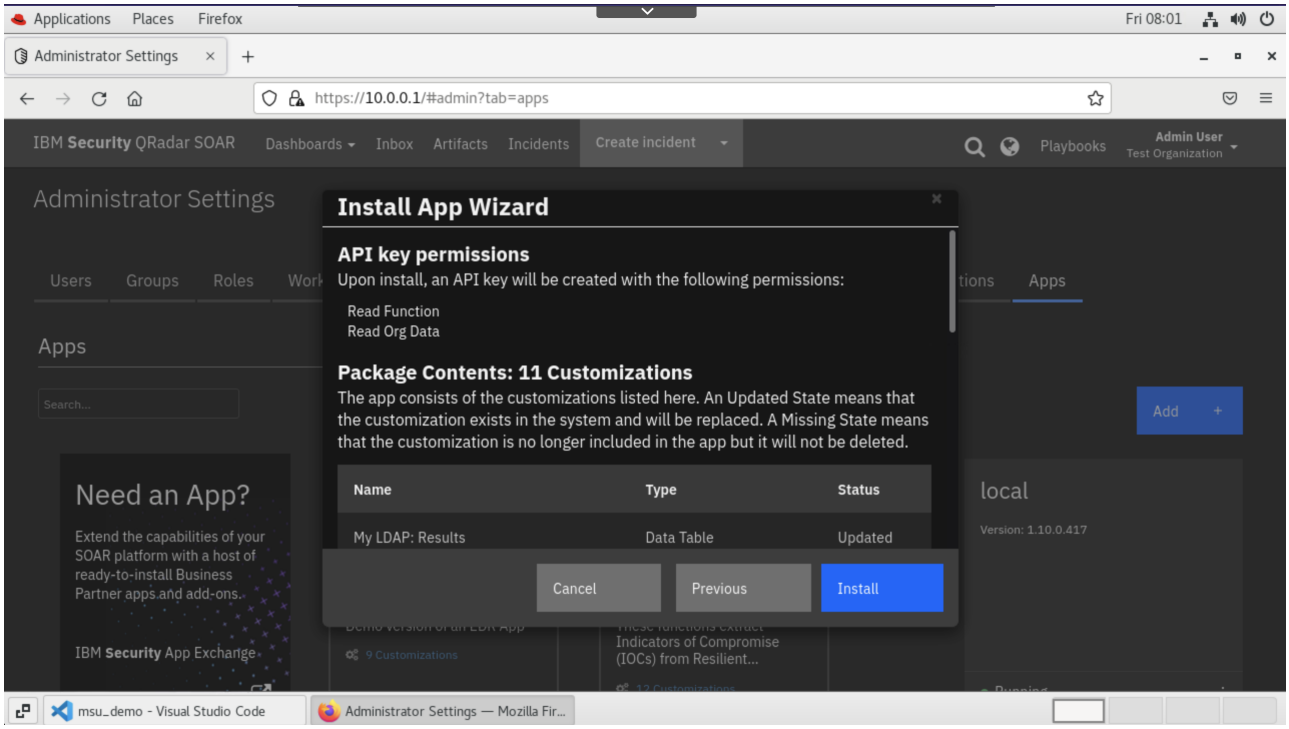


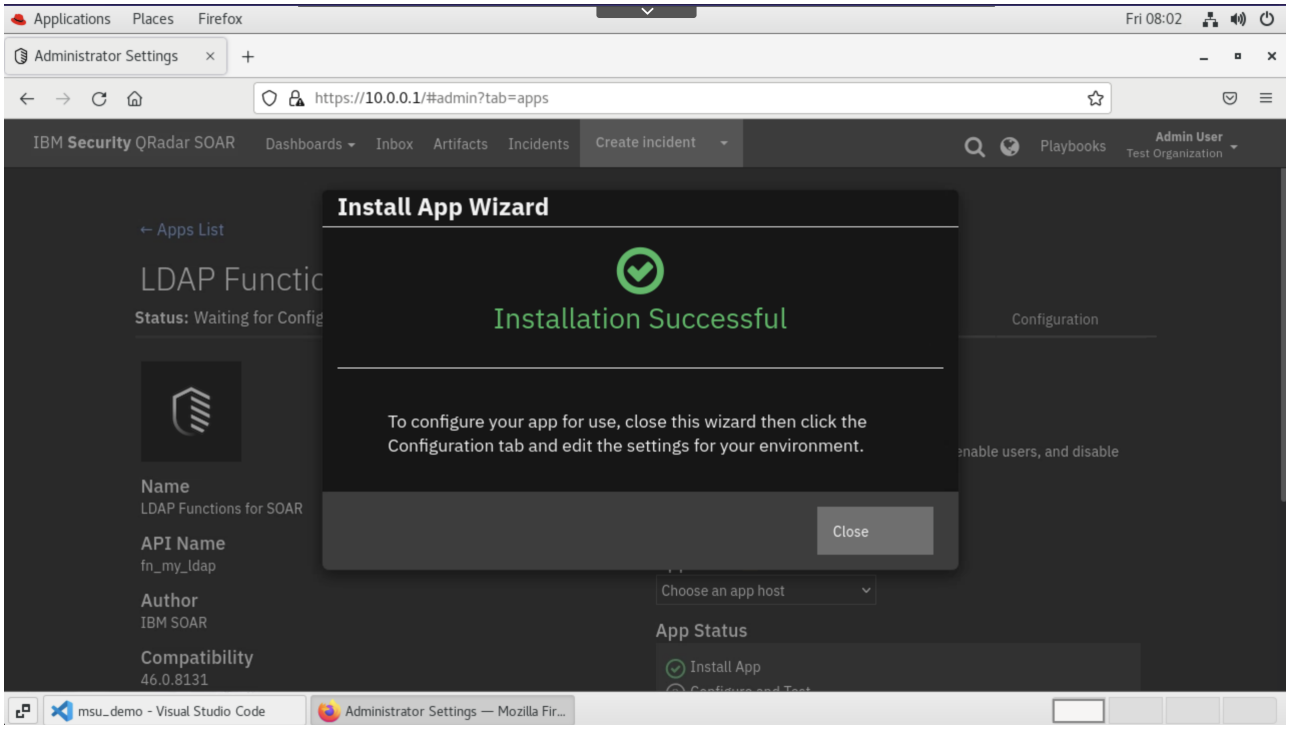
Click **Open**, followed by **Upload File**.

- Click **Next**:

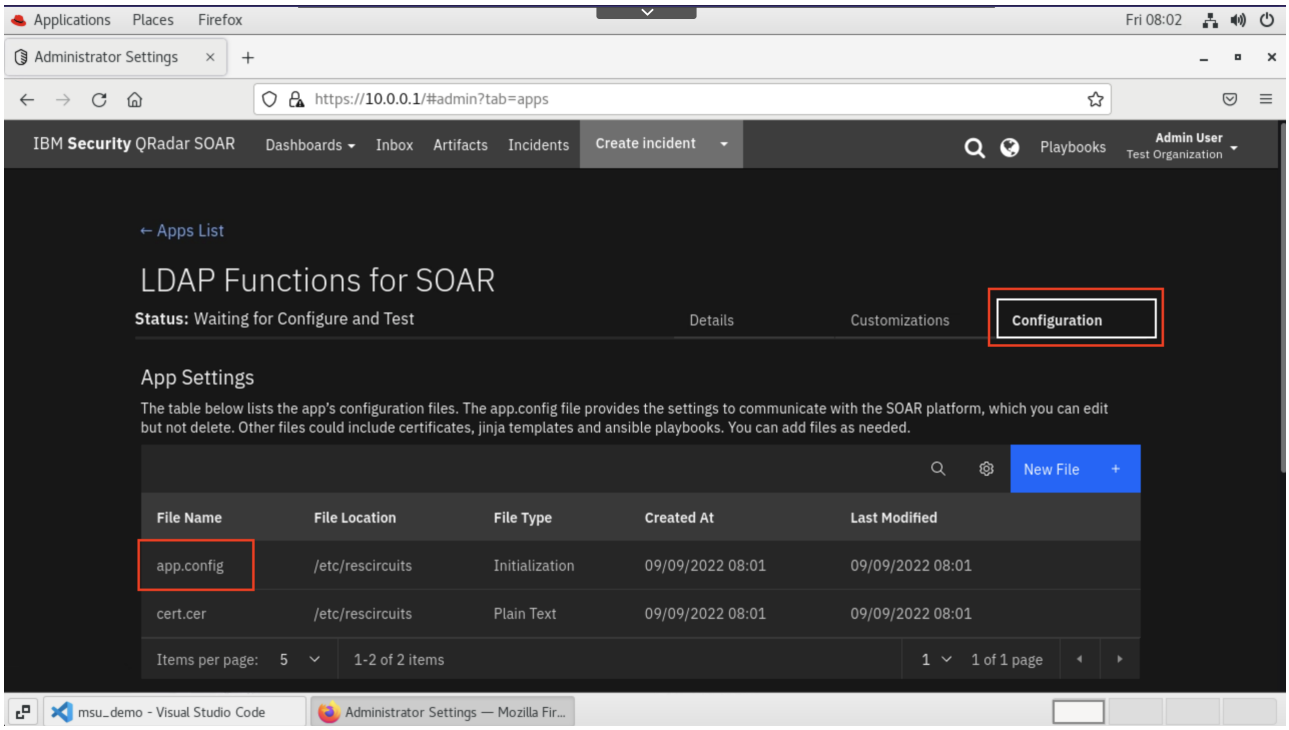


- Click **Install**:

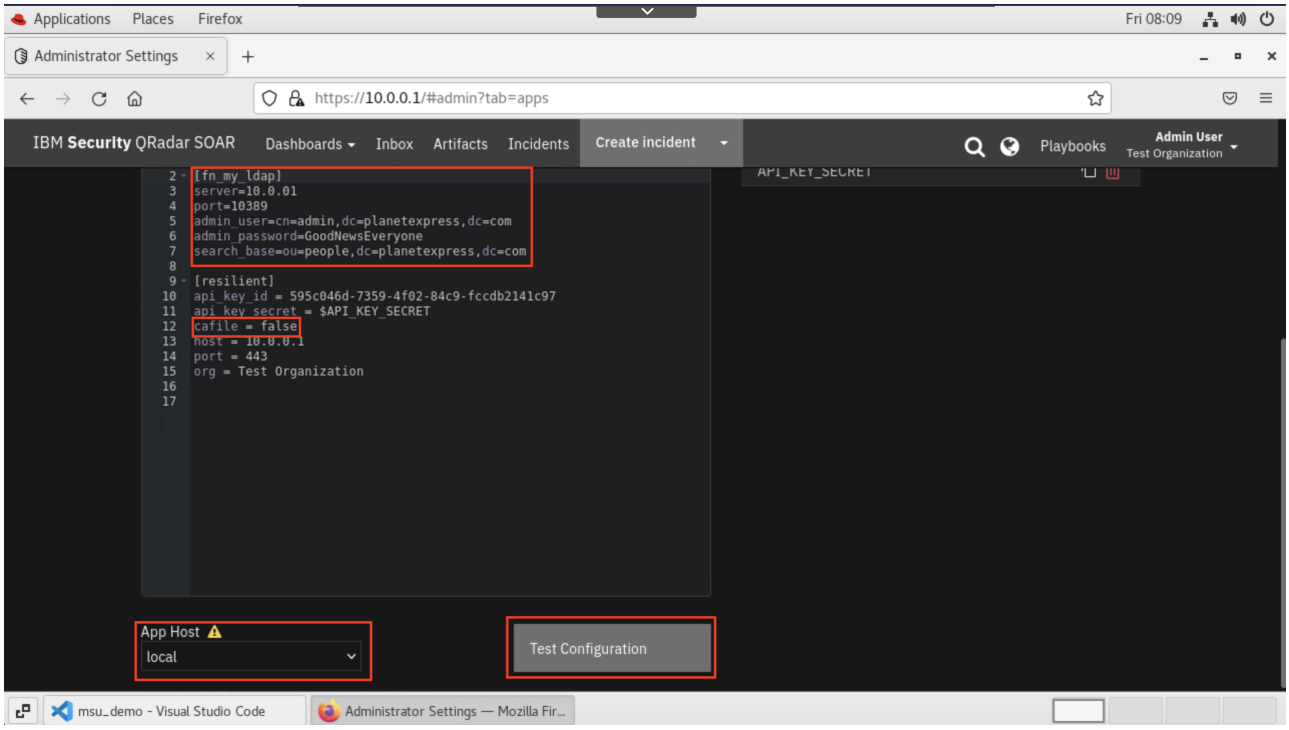


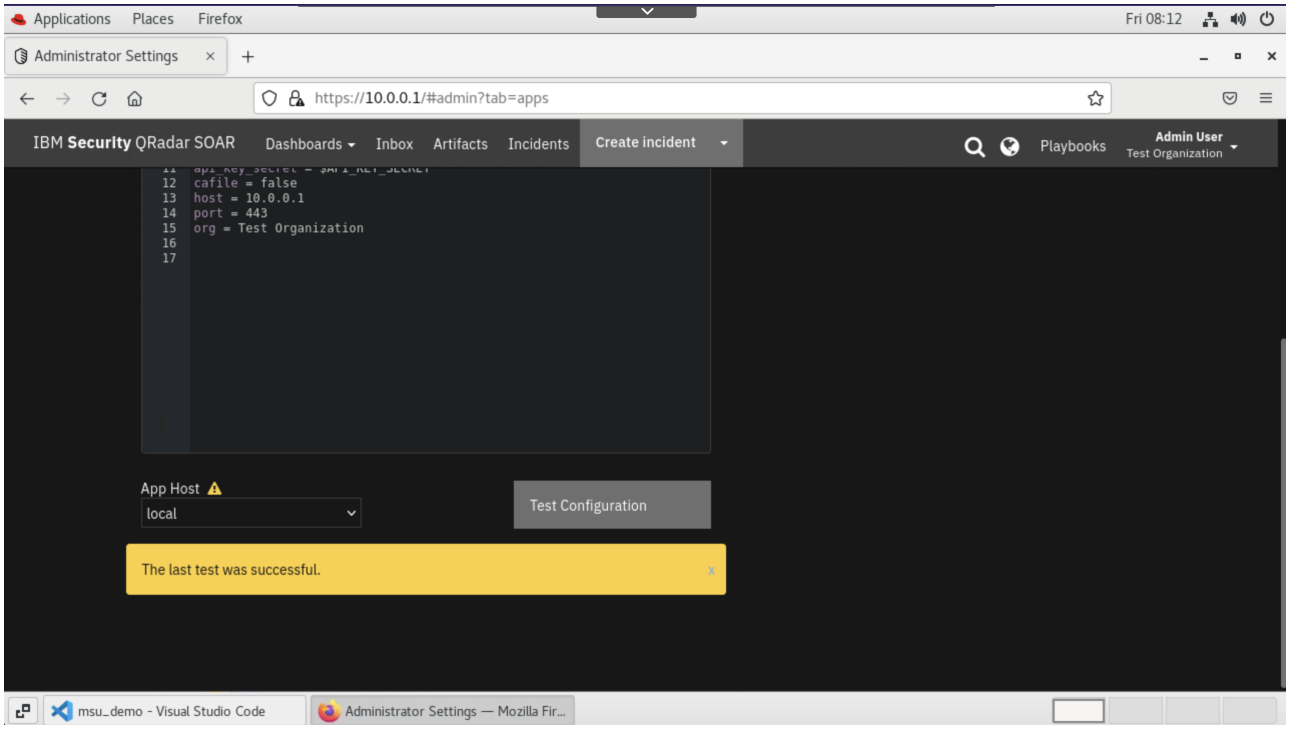


- Navigate to the `app.config` file in the **Configuration Tab** of the app:

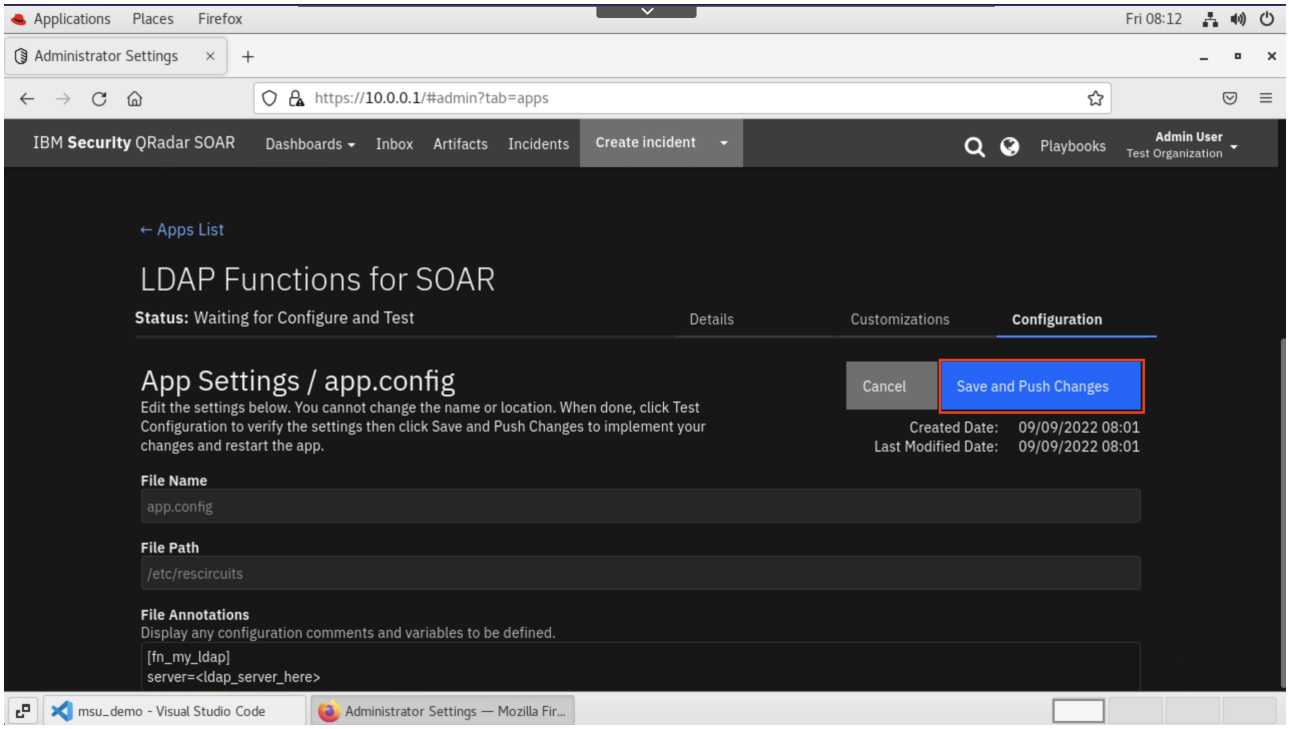


- Enter the same details that we entered locally. Once entered, select the app host that points to the local repo and click **Test Configuration**:

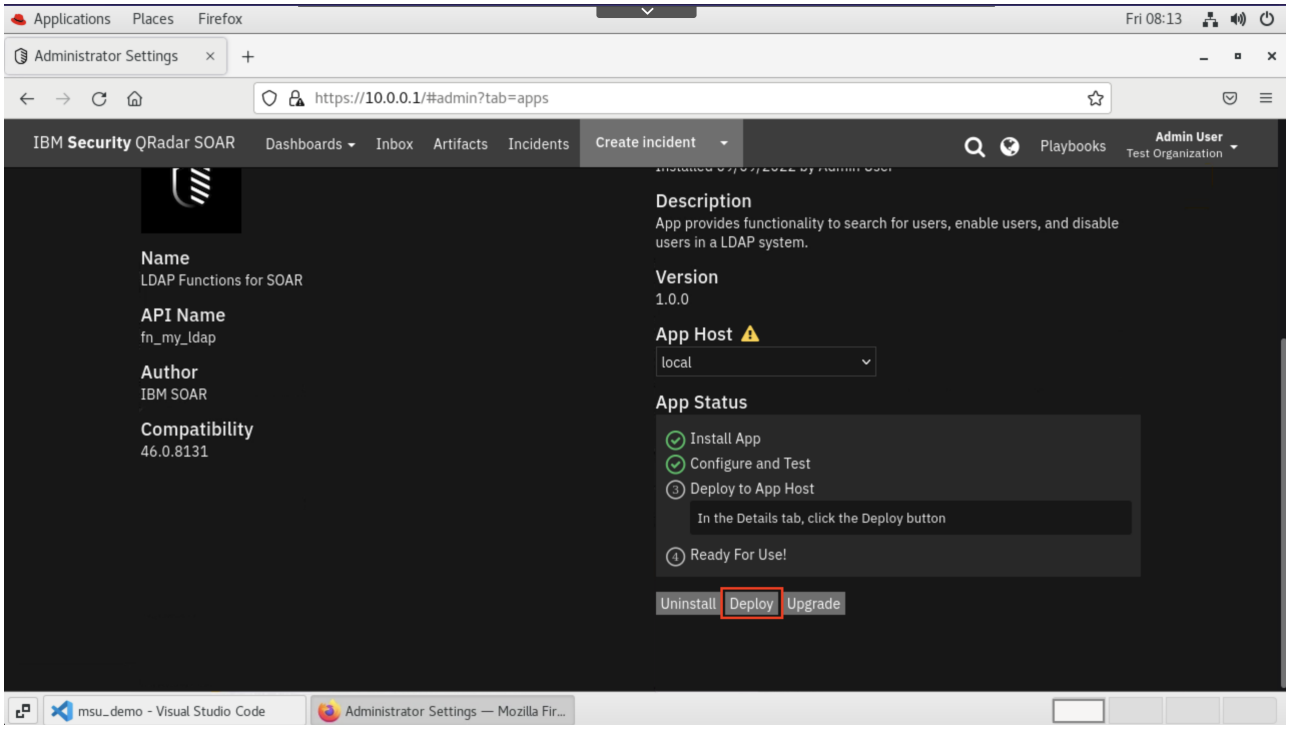

Applications Places Firefox
Fri 08:09
Administrator Settings
https://10.0.0.1/#admin?tab=apps
IBM Security QRadar SOAR
Dashboards
Inbox
Artifacts
Incidents
Create incident
Playbooks
Admin User
Test Organization
[fn_my_ldap]
server=10.0.01
port=10389
admin_user=cn=admin,dc=planetexpress,dc=com
admin_password=GoodNewsEveryone
search_base=ou=people,dc=planetexpress,dc=com
[resilient]
api_key_id = 595c046d-7359-4f02-84c9-fccdb2141c97
api_key_secret = $API_KEY_SECRET
cafile = false
host = 10.0.0.1
port = 443
org = Test Organization
App Host
local
Test Configuration
msu_demo - Visual Studio Code
Administrator Settings — Mozilla Fir...


Applications Places Firefox
Fri 08:12
Administrator Settings
https://10.0.0.1/#admin?tab=apps
IBM Security QRadar SOAR
Dashboards
Inbox
Artifacts
Incidents
Create incident
Playbooks
Admin User
Test Organization
cafile = false
host = 10.0.0.1
port = 443
org = Test Organization
App Host
local
Test Configuration
The last test was successful.
msu_demo - Visual Studio Code
Administrator Settings — Mozilla Fir...

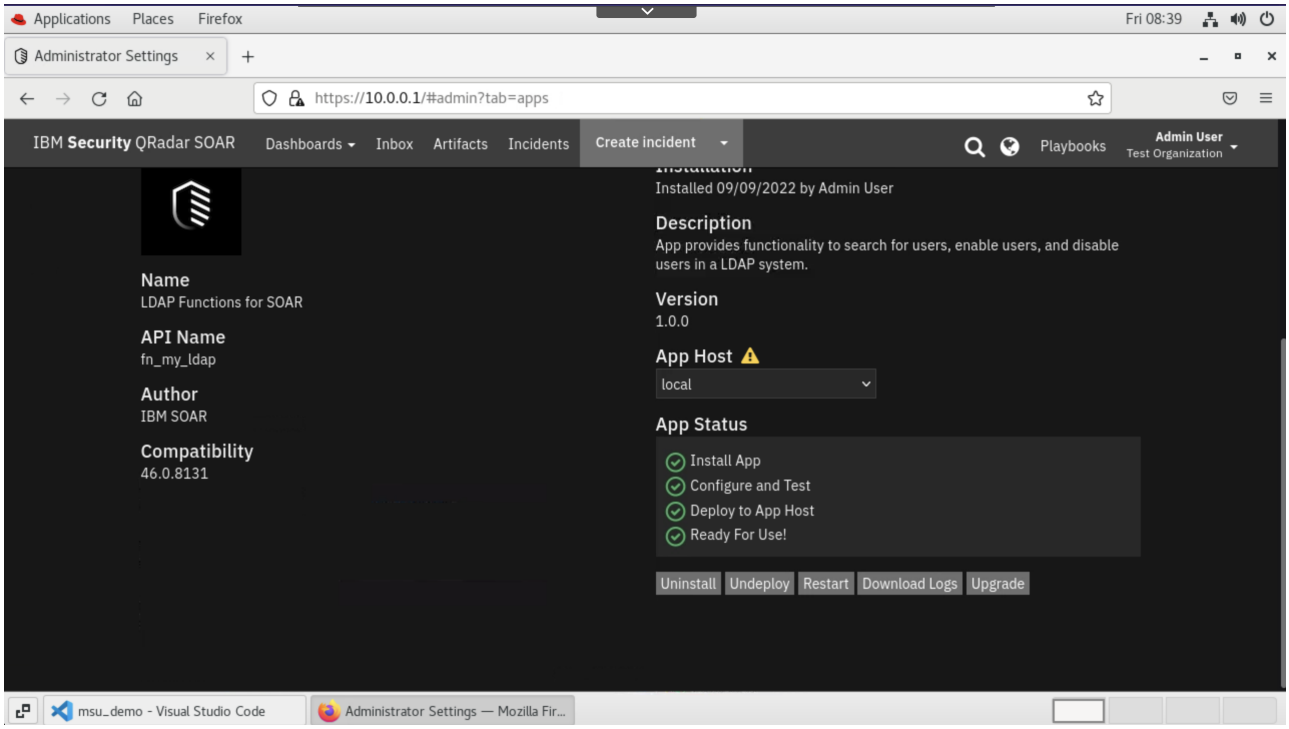
- Scroll up and click **Save and Push Changes**:



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



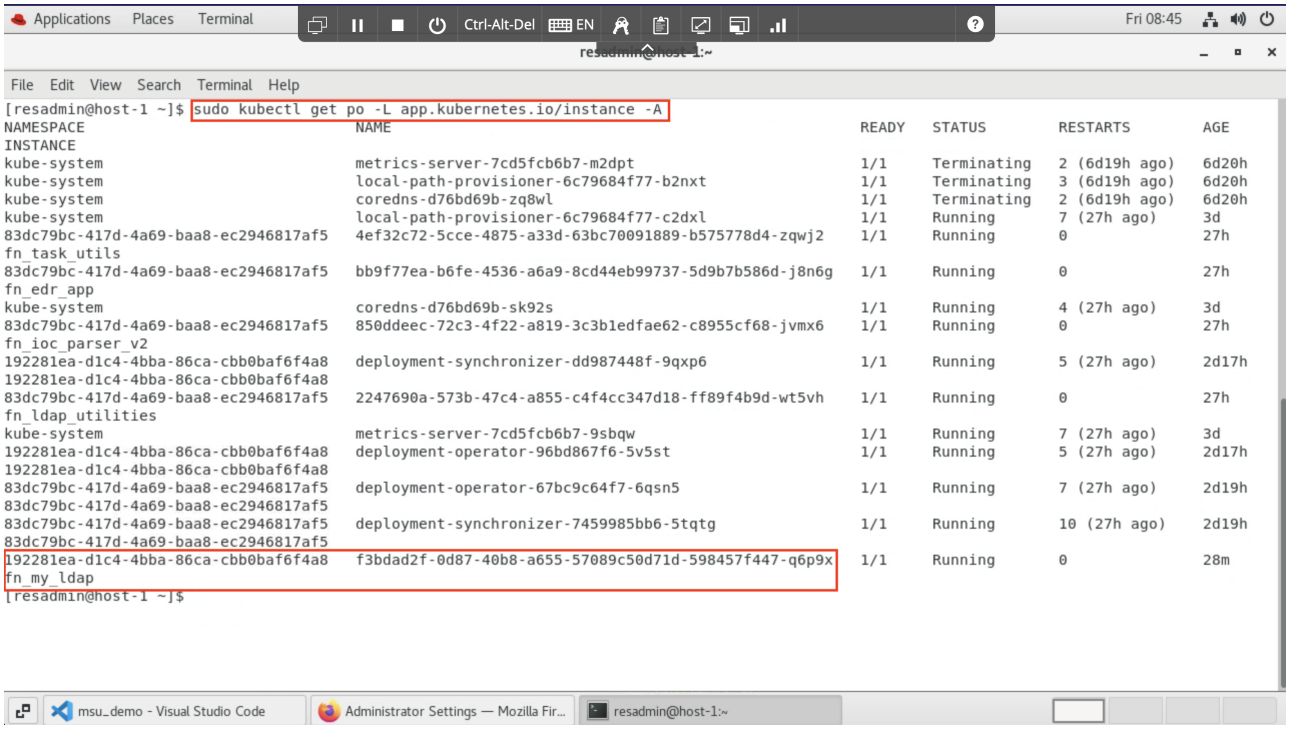
- Once the app is deployed, you can now run it's functions and playbooks.



- 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:

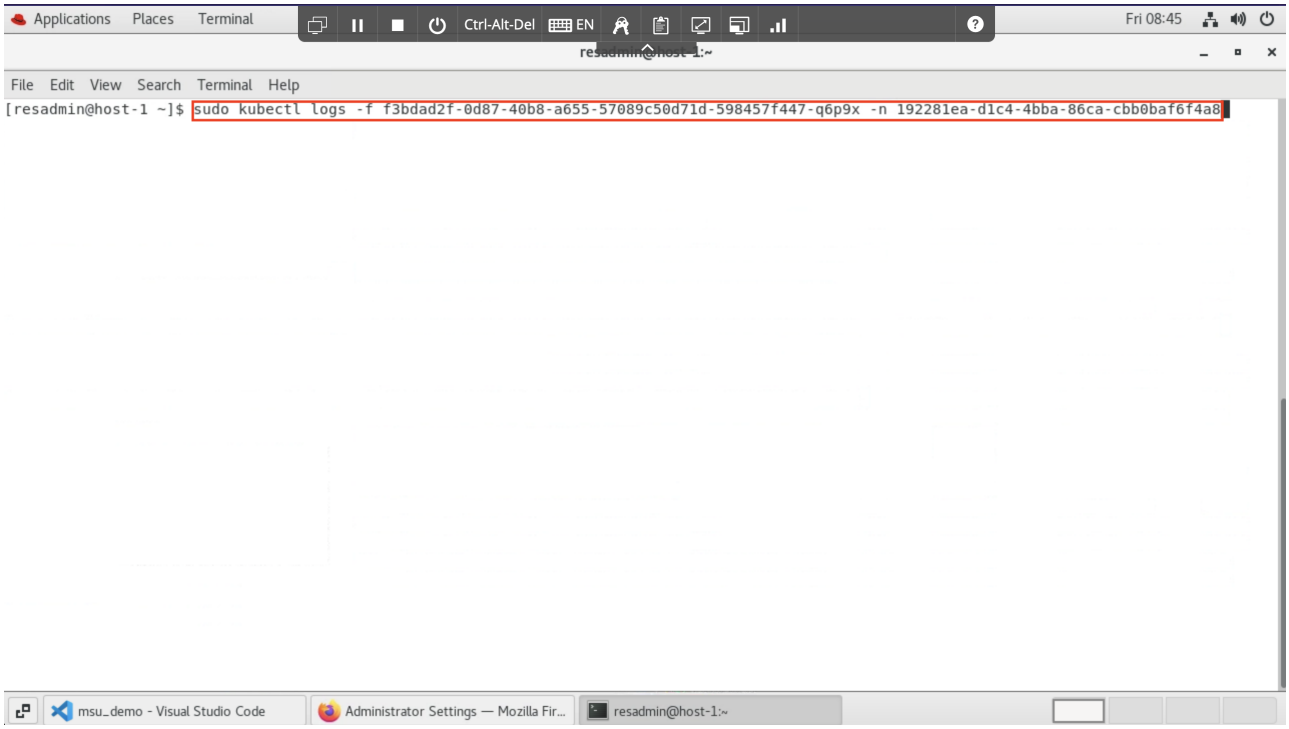
```
sudo kubectl get po -L app.kubernetes.io/instance -A
```

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



- Launch the logs in `follow` mode:

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


Applications Places Terminal
Ctrl-Alt-Del EN
Fri 08:45
File Edit View Search Terminal Help
[resadmin@host-1 ~]$ sudo kubectl logs -f f3bdad2f-0d87-40b8-a655-57089c50d71d-598457f447-q6p9x -n 192281ea-d1c4-4bba-86ca-cbb0baf6f4a8
msu_demo - Visual Studio Code
Administrator Settings — Mozilla Fir...
resadmin@host-1:~