**Introduction**

AWS [Cloud Formation](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html) is an Infrastructure As a Code service that lets you define your resources in a JSON or YAML file called cloud formation templates, and deploy them on AWS without having to worry about setting each configuration and dependencies manually, thus saving a lot of time and money, Cloud Formation can also be included in a CI/CD pipeline allowing you to automatically update your infrastructure to the new requirements

**Problem**

In a cloud working environment if a resource isn’t working properly it can cause problems in the whole infrastructure, to solve this it must be that the Cloud Formation templates are properly tested before deploying to the production environment. There are several tools and frameworks available for this, among them being [TaskCat](https://aws.amazon.com/blogs/infrastructure-and-automation/introducing-taskcat-v0-9/#:~:text=TaskCat%20is%20an%20open%2Dsource,publish%20AWS%20CloudFormation%20reference%20architectures.), which tests the template by deploying it to several regions and compiling a report. Though this ensures that the template is properly functional, it costs a lot of money as new resources are deployed in the AWS environment. So a solution is needed to test the templates without incurring high costs

**Solution**

A CI/CD pipeline comprising of opensource tools which can perform various types of testing: linting, security testing, policy testing, unit testing, functional testing

**Phases of the proposed solution:**

Phase-1: Setting up the environment

Phase-2: Static code Analysis

* Linting using [cfn-lint](https://github.com/aws-cloudformation/cfn-lint)
* Security testing using [cfn-nag](https://github.com/stelligent/cfn_nag)
* Policy and rule testing using [cfn-guard](https://github.com/aws-cloudformation/cloudformation-guard)

Phase-3: Unit testing using [cloud-radar](https://github.com/DontShaveTheYak/cloud-radar)

Phase-4: Deploying to [localstack](https://localstack.cloud/)

Phase-5: Improving efficiency by using Github action

**Setting up the environment(Phase-1)**

The following demonstration is done on Ubuntu OS

**Pre-requisites:**

Python3.9

Git

pip

Pre-commit

Python3.9-venv

rubby

Open a terminal in the Ubuntu and write the following commands:

sudo apt install python3.9

sudo apt-get install git

sudo apt install python3-pip

sudo pip install pre-commit

sudo apt install python3.9-venv

sudo apt install ruby

**Setting up a virtual environment for python3.9**

A virtual environment is an isolated environment, such that any dependencies, packages, etc installed in this environment don’t affect the rest of the system

python3.9 -m venv atlanTest

source atlanTest/bin/activate

**Initialize as a git repository**

git init

**Setting up the directory structure**

mkdir templates

mkdir tests

mkdir tests/unit

mkdir guard\_rules

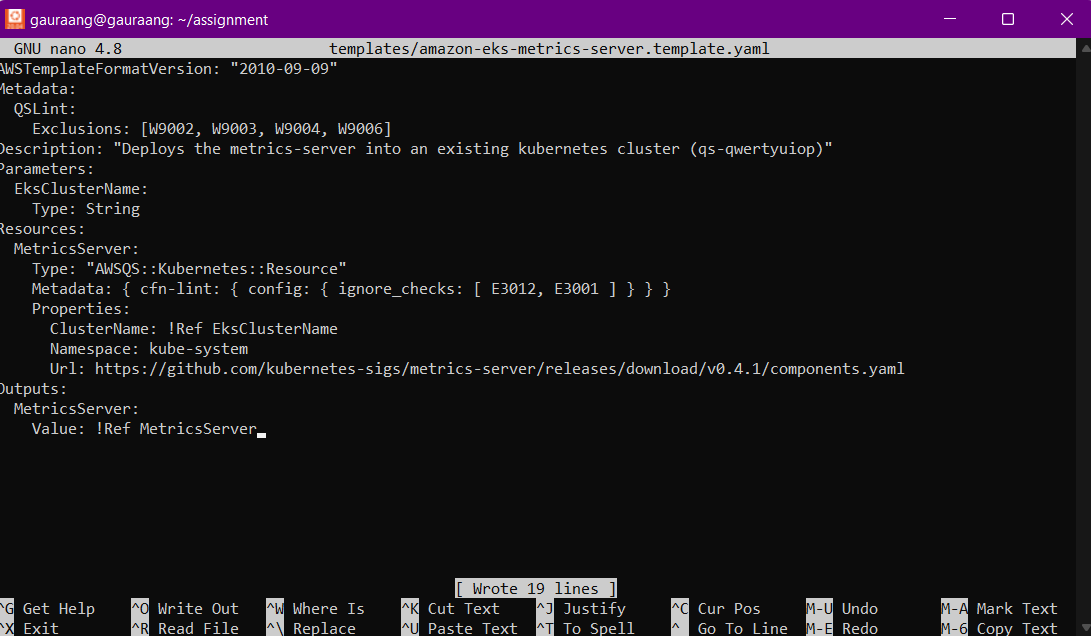
* Templates directory will store cloud formation templates
* tests/unit directory will store unit tests
* Guard\_rules will store rules for cfn-guard

Selecting a Cloud Formation template for demonstration: [amazon-eks-metrics-server.template.yaml](https://github.com/aws-quickstart/quickstart-amazon-eks/blob/main/templates/amazon-eks-metrics-server.template.yaml) and pastes its content in a file in the templates directory

nano templates/amazon-eks-metrics-server.template.yaml

(Paste the template contents)

And add E3001 to ingnore\_checks (Important otherwise cfn-lint will show errors)



**Setting up pre-commit**

Pre-commit allows you to specify various hooks, which are run automatically before you commit your code, this helps ensure that only the code that meets certain standards is committed.

For more information follow the [link](https://pre-commit.com/)

We will be storing our dependencies in the requirements.txt file.

pip freeze > requirements.txt

Create a .pre-commit-config.yaml file and add some basic pre-commit hooks to it

nano .pre-commit-config.yaml

Add the following code:

repos:

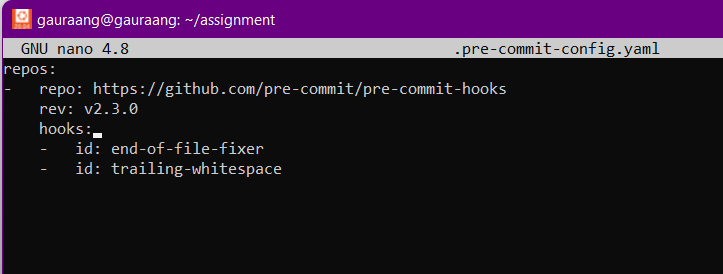
- repo: https://github.com/pre-commit/pre-commit-hooks

rev: v2.3.0

hooks:

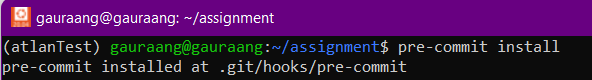
- id: end-of-file-fixer

- id: trailing-whitespace

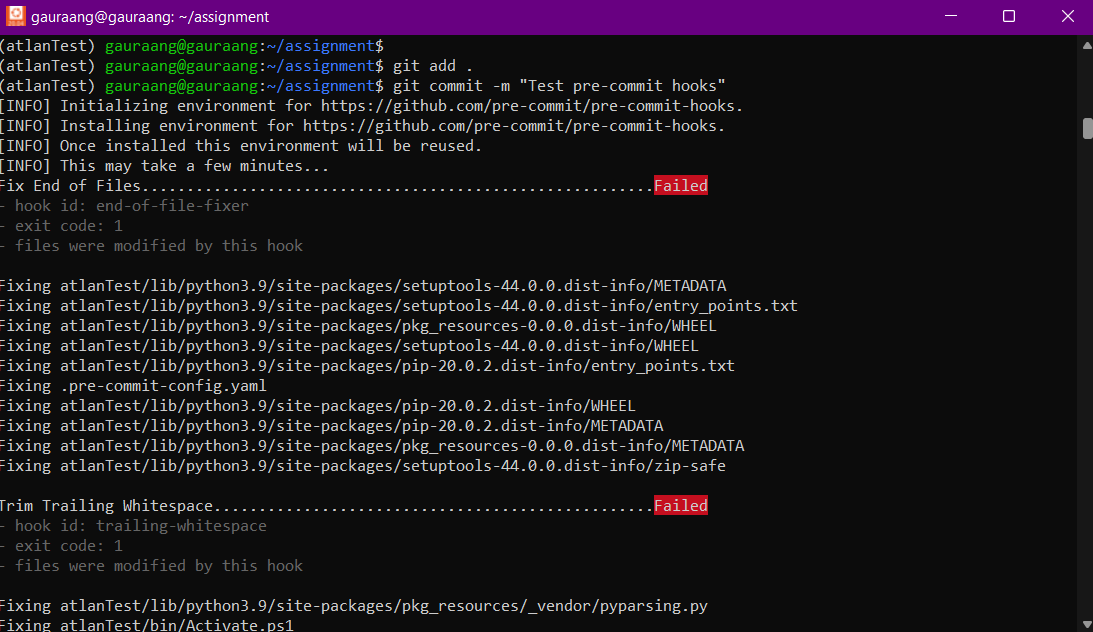


We’ll now run pre-commit using the following command:

pre-commit install

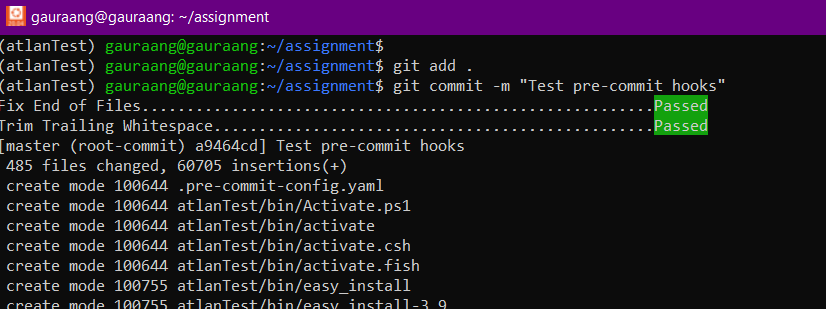


Now we will stage our files and commit them, so we can check how the pre-commit works



**Note:** Since the files were not properly formatted, hooks show the failed tag, and properly formats the files

Again staging and commit



**Static Code Analysis(Phase-2)**

It is a method of debugging by examining source code before a program is run. We will perform various types of tests under this: linting, Static Application Security test, Policy validation test

**Linting**

It is a automated process that checks your source code for programmatic and stylistic errors. Lint test can be performed on the Cloud Formation Template using [cfn-lint](https://github.com/aws-cloudformation/cfn-lint) tool. It validates AWS CloudFormation json/yaml templates against the AWS CloudFormation Resource Specification.

We will add cfn-lint as a hook in the .pre-commit-config.yaml file, this way pre-commit will automatically run cfn-lint on the souce files before a commit

Add the following code to the .pre-commit-config.yaml file

repos:

- repo: https://github.com/pre-commit/pre-commit-hooks

rev: v2.3.0

hooks:

- id: end-of-file-fixer

- id: trailing-whitespace

- repo: https://github.com/aws-cloudformation/cfn-python-lint

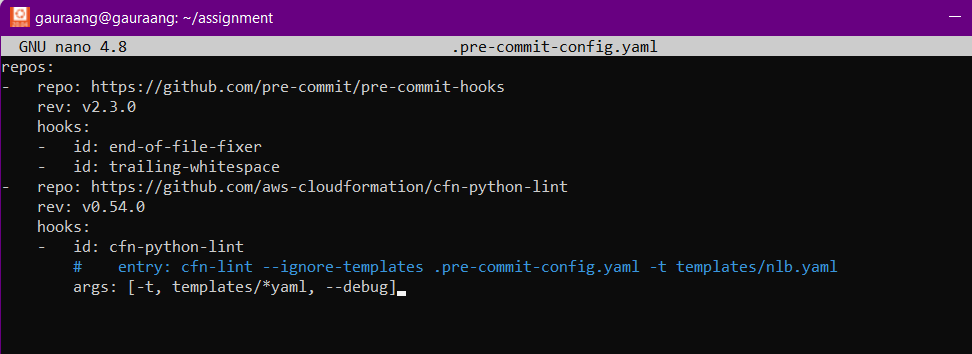
rev: v0.54.0

hooks:

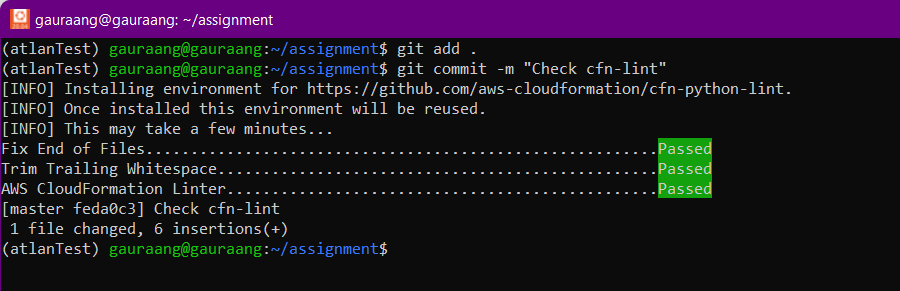
- id: cfn-python-lint

# entry: cfn-lint --ignore-templates .pre-commit-config.yaml -t templates/nlb.yaml

args: [-t, templates/\*yaml, --debug]



Check if the cfn-lint is working properly by staging and making a commit:



### **SAST (Static Application Security Test)**

SAST is a testing methodology to scan the code to locate known vulnerabilities that makes your infrastructure vulnerable to attackers

[cfn-nag](https://github.com/stelligent/cfn_nag) is a tool that looks for patterns in CloudFormation templates that may indicate insecure infrastructure. It will look for:

* IAM rules that are too permissive (wildcards)
* Security group rules that are too permissive (wildcards)
* Access logs that aren't enabled
* Encryption that isn't enabled
* Password literals

cfn-nag doesn’t provide an official hook that can be used with the pre-commit, so we will be using a mirror

Add the following code to .pre-commit-config.yaml file

- repo: https://github.com/AleksaC/mirrors-cfn-nag

rev: v0.6.13 # Use the sha / tag you want to point at

hooks:

- id: cfn-nag

name: cfn-nag

description: Wrapper around cfn\_nag\_scan that allows it to be ran on multiple files within a single hook

entry: cfn\_nag\_scan\_wrapper

language: ruby

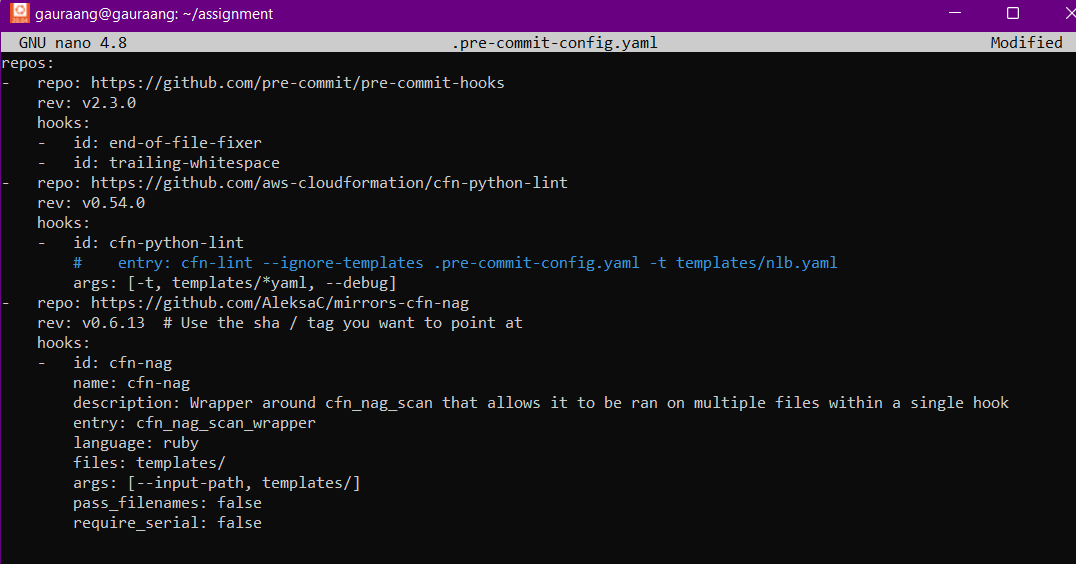
files: templates/

args: [--input-path, templates/]

pass\_filenames: false

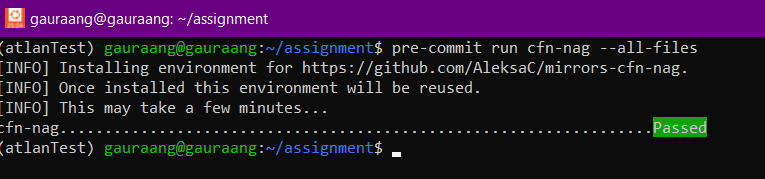
require\_serial: false

Your .pre-commit-config.yaml file will look like:



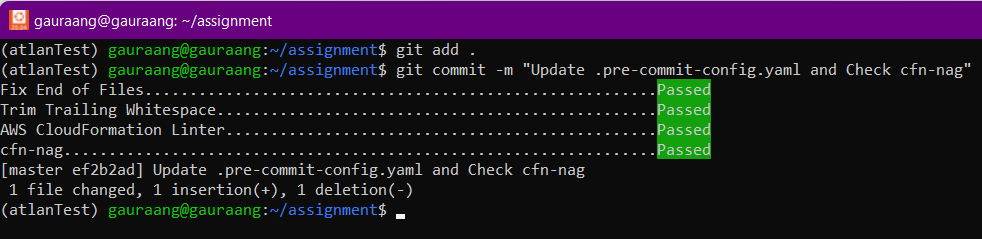
Testing the cfn-nag tool using the following command:

pre-commit run cfn-nag --all-files



Since its working properly, stage the files and make a commit to check the proper functioning

**Note:** It might show skipped in the cfn-nag, because you have already tested the files in the above command, it doesn’t for me because I changed the template slightly for demonstrating this step



**Policy validation testing**

In a cloud-based infrastructure it is essential that the resources meet certain policies this ensures the integrity, security, and reliability of the infrastructure.

For example, A organization might have the policy to ensure that all the EC2 instances are encrypted

[cfn-guard](https://github.com/aws-cloudformation/cloudformation-guard) is a tool that helps achieve this by providing ways to achieve:

* Preventative Governance and Compliance
* Detective Governance and Compliance
* Deployment Safety

It is an open-source policy-as-code evaluation tool, that allows developers to define policies and validate them against JSON or YAML infrastructure code

**Setup**

Run the following commands to install cfn-guard and add it to the path

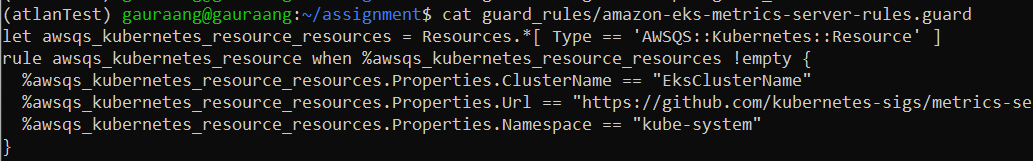
curl --proto '=https' --tlsv1.2 -sSf https://raw.githubusercontent.com/aws-cloudformation/cloudformation-guard/main/install-guard.sh | sh

PATH=${PATH}:~/.guard/bin

Generate rules for the [amazon-eks-metrics-server.template.yaml](https://github.com/aws-quickstart/quickstart-amazon-eks/blob/main/templates/amazon-eks-metrics-server.template.yaml) using the following command:

cfn-guard rulegen --output guard\_rules/amazon-eks-metrics-server-rules.guard --template templates/amazon-eks-metrics-server.template.yaml

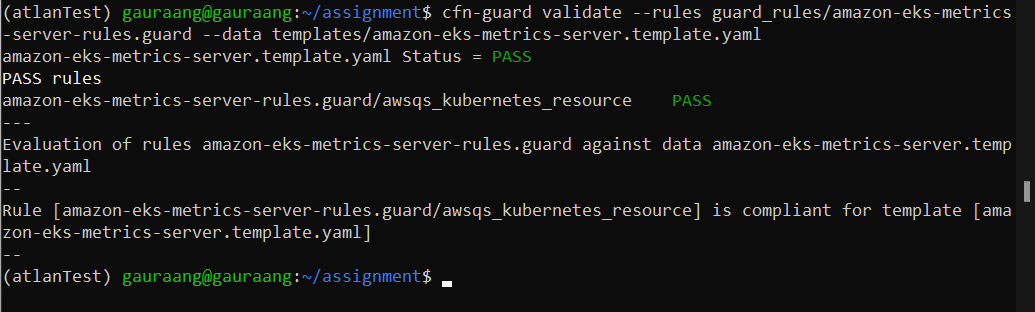
This will generate a file with the guard rules in the guard\_rules directory



**Note:** These are very basic rules, which can be further tweaked according to the organization’s requirements

Validating if our template meets the rules using the following command:

cfn-guard validate --rules guard\_rules/amazon-eks-metrics-server-rules.guard --data templates/amazon-eks-metrics-server.template.yaml



Adding the cfn-guard as a hook to the .pre-commit-config.yaml file

Add the following code:

- repo: local

hooks:

- id: cfn-guard

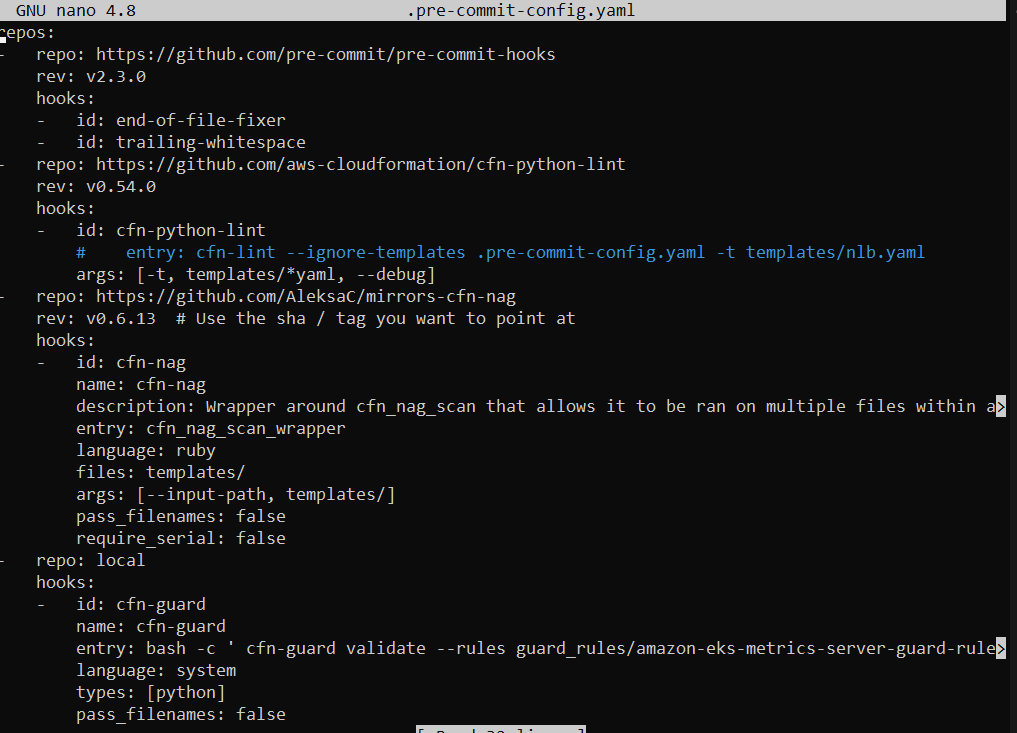
name: cfn-guard

entry: bash -c ' cfn-guard validate --rules guard\_rules/amazon-eks-metrics-server-rules.guard --data templates/amazon-eks-metrics-server.template.yaml'

language: system

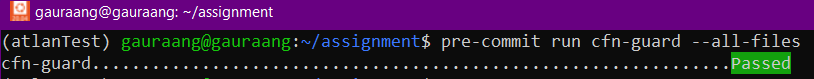
types: [python]

pass\_filenames: false

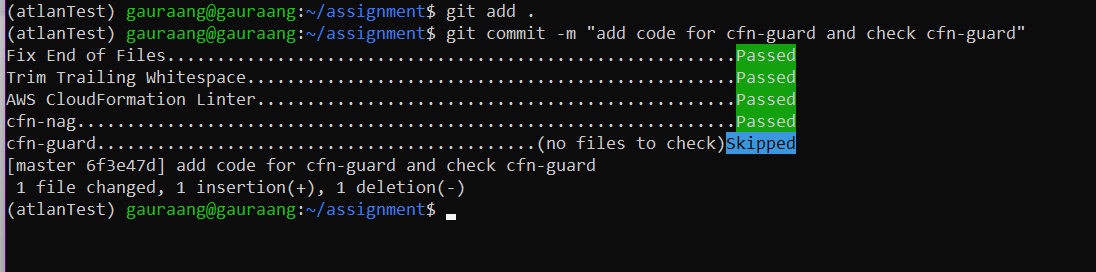


We can test cfn-guard using the following command:

pre-commit run cfn-guard --all-files



Since its working properly, stage the files and make a commit to check the proper functioning



**Note:** It shows skipped because we recently ran the hook, and no changes were made since then

**UNIT Tests(Phase-3)**

Unit testing is a testing methodology in which individual units of code are tested in isolation. For Cloud Formation Templates individual AWS resources and parameters can be seen as units and tested

[Cloud-Radar](https://github.com/DontShaveTheYak/cloud-radar) is a python module that allows testing of Cloudformation Templates/Stacks using Python. Cloud-Radar takes your template, the desired region, and some parameters then render it into the final state and passes it back, allowing you to test all the resources without worrying about AWS Credentials

cloud-radar works by reading our Cloudformation template and then rendering it the same way that the AWS Cloudformation service would. Our template consists of multiple parameters and resources that are to be created. We will be testing all of the resources that are going to be created.

**Setup**

Use the following commands to install pytest and cloud-radar

sudo apt install python-pytest

pip install cloud-radar==0.6.0

Create a test file using the following command:

nano tests/unit/test\_amazon-eks-metrics-server.py

Fixtures are functions, which will run before each test function to which it is applied. Fixtures are used to feed some data to the tests and here it will provide a template path

template\_path.resolver() will return a dictionary that has all the CloudFormation functions and conditions resolved

Add the following code:

from pathlib import Path

import pytest

from cloud\_radar.cf.unit import Template

import json

@pytest.fixture(scope='session')

def template\_path() -> Path:

base\_path = Path(\_\_file\_\_).parent

template\_path = base\_path / Path('../../templates/amazon-eks-metrics-server.template.yaml')

return template\_path.resolve()

def test\_params(template\_path: Path):

template = Template.from\_yaml(template\_path)

region = "us-west-2"

EksClusterName = "cf-testing"

# Dictionary Of Parameters

params = {"EksClusterName":EksClusterName}

# Render the template using parameters and region

result = template.render(params, region)

# Print template

print(json.dumps(result, indent=4, default=str))

# Check if Proper resources have been created

resource\_list = ["MetricsServer"]

for resource in resource\_list:

assert resource in result["Resources"]

# Test Metrics Server

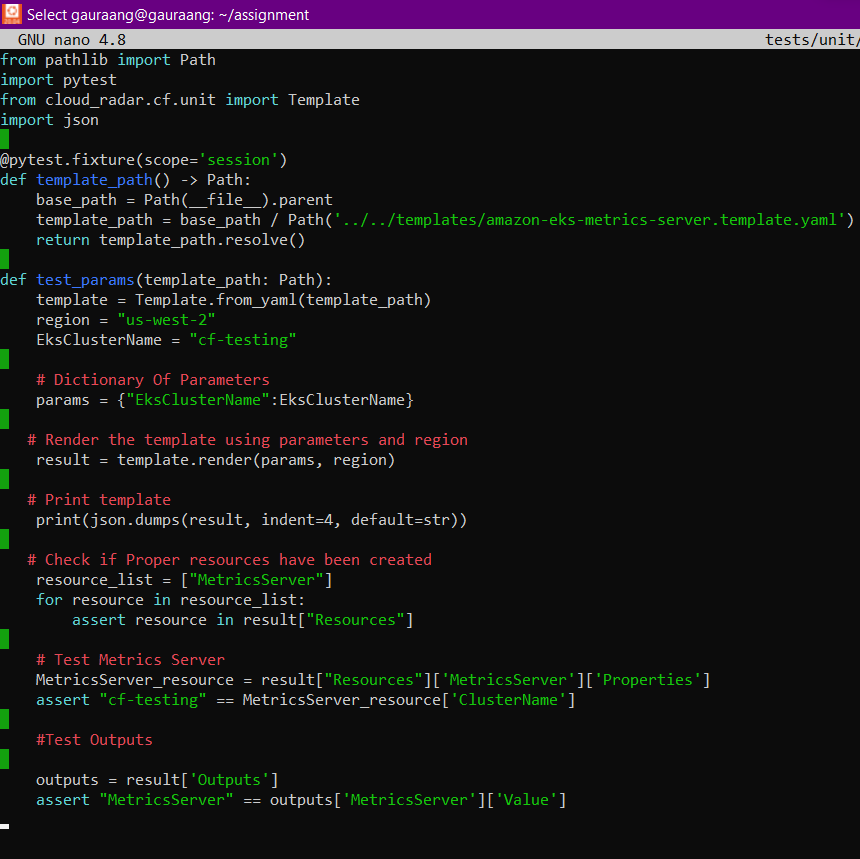
MetricsServer\_resource = result["Resources"]['MetricsServer']['Properties']

assert "cf-testing" == MetricsServer\_resource['ClusterName']

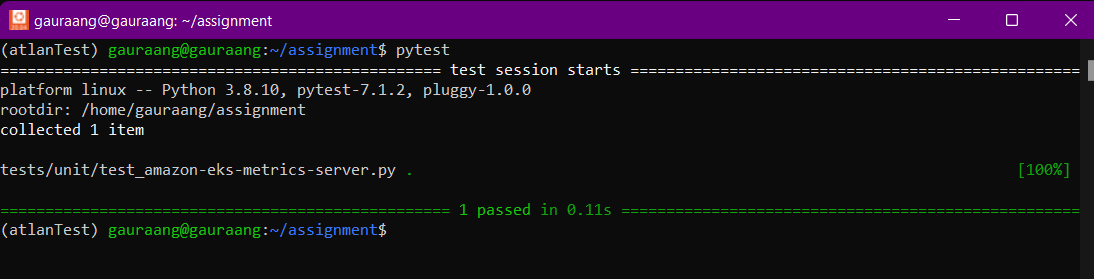
#Test Outputs

outputs = result['Outputs']

assert "MetricsServer" == outputs['MetricsServer']['Value']



Run Pytest command to check if the test is working properly



Add pytest to the .pre-commit-config.yaml file

Add the following code:

- id: pytest

name: pytest

entry: pytest

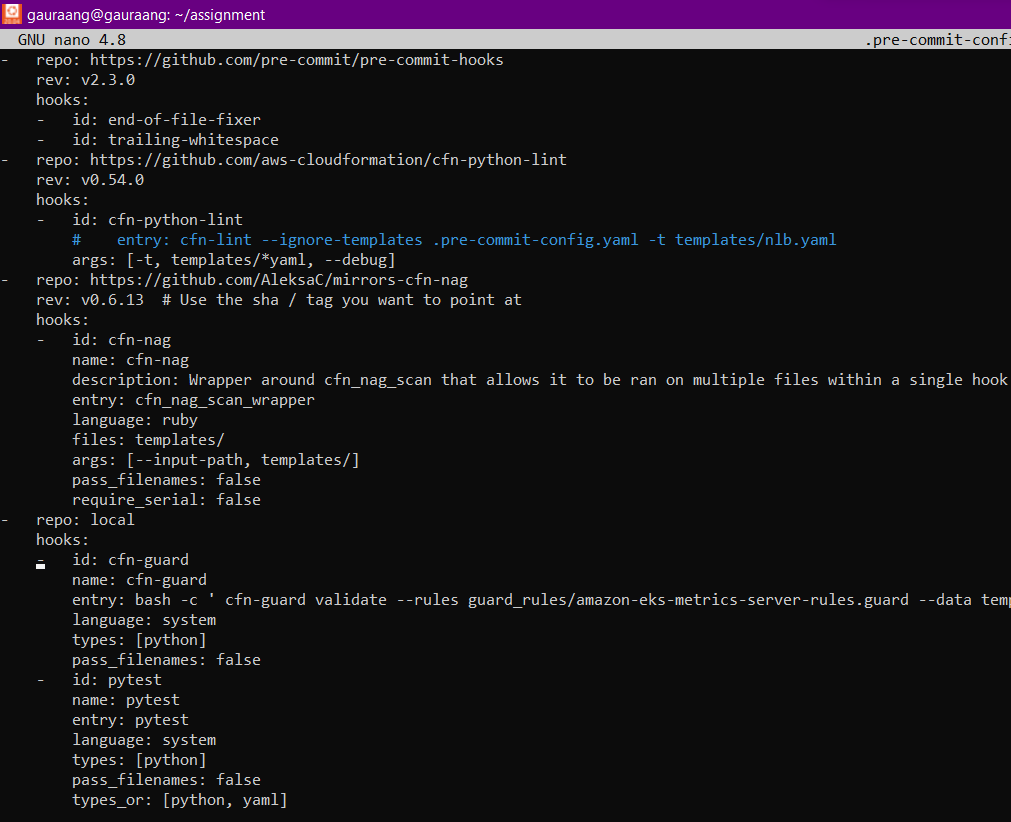
language: system

types: [python]

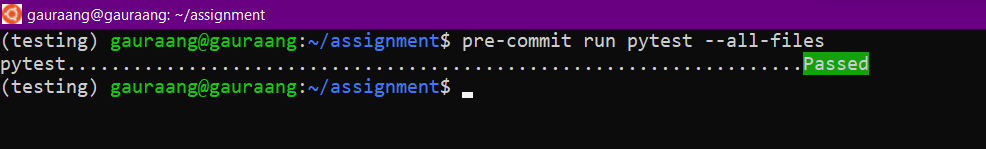
pass\_filenames: false

types\_or: [python, yaml]

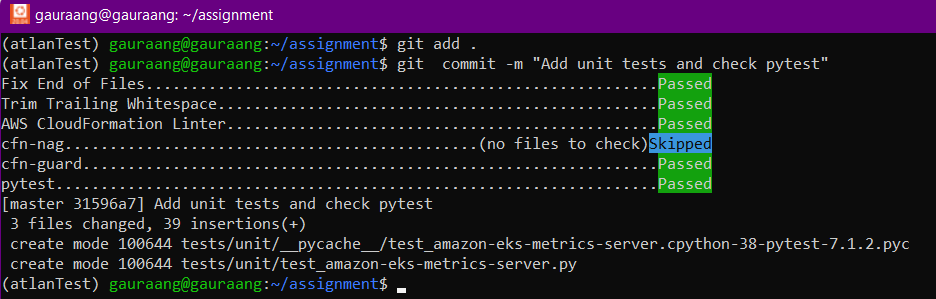
.pre-commit-config.yaml file will look like:



Run pre-commit run pytest --all-files to check if pytest is working properly



Since its working properly, stage the files and make a commit to check the proper functioning



**Deploying to localStack(Phase-4)**

[localStack](https://localstack.cloud/) provides a development environment similar to AWS on your local machine, allowing you to run and test AWS resources without spending any money. It provides the same functionality and APIs as the real AWS cloud environment.

**Setup:**

**Note:** You must have docker working on your system

Use the following commands to install localStack:

sudo python3 -m pip install localstack

sudo apt install docker.io

sudo apt install docker-compose

Install AWS cli using the following command:

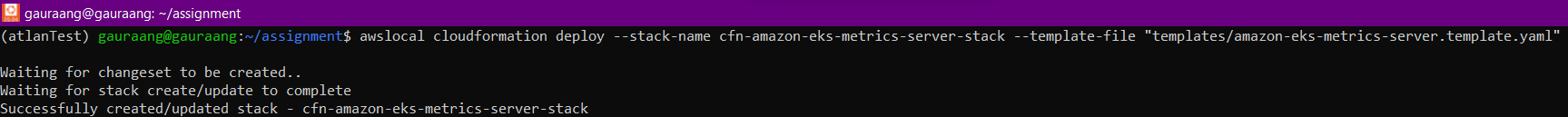
sudo pip install awscli-local[ver1]

Start localStack using the following command:

sudo nohup localstack start --host &

Create a stack of your Cloud Formation template using the following command:

awslocal cloudformation deploy --stack-name cfn-amazon-eks-metrics-server-stack --template-file "templates/amazon-eks-metrics-server.template.yaml"



Add localstack to the pre-commit

Add the following code to the .pre-commit-config.yaml file:

- id: localstack-deploy

name: localstack:deploy

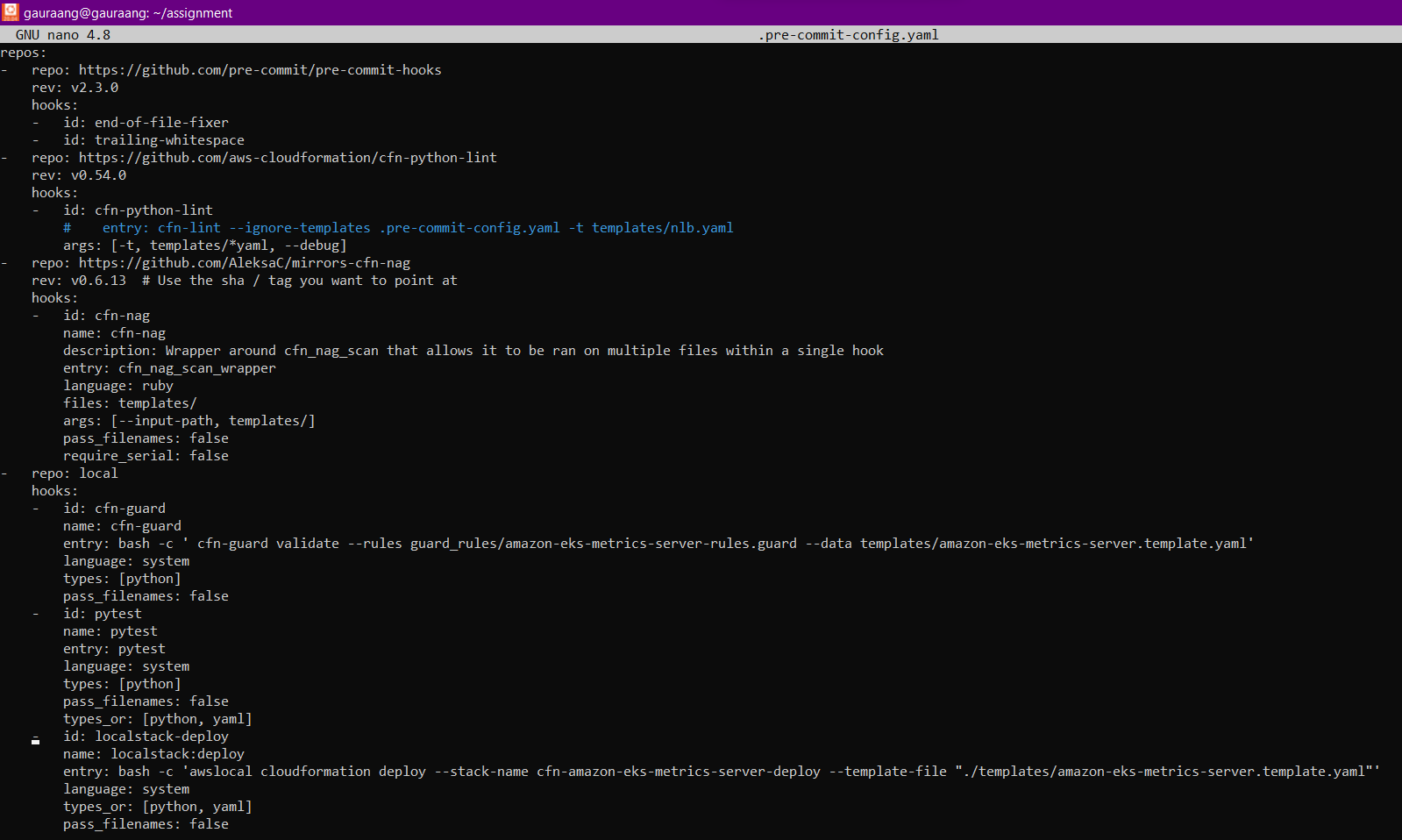
entry: bash -c 'awslocal cloudformation deploy --stack-name cfn-amazon-eks-metrics-server-deploy --template-file "./templates/amazon-eks-metrics-server.template.yaml"'

language: system

types\_or: [python, yaml]

pass\_filenames: false

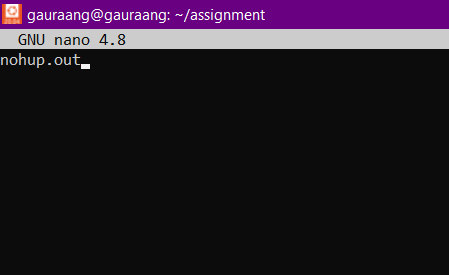
Your .pre-commit-config.yaml file will look like:



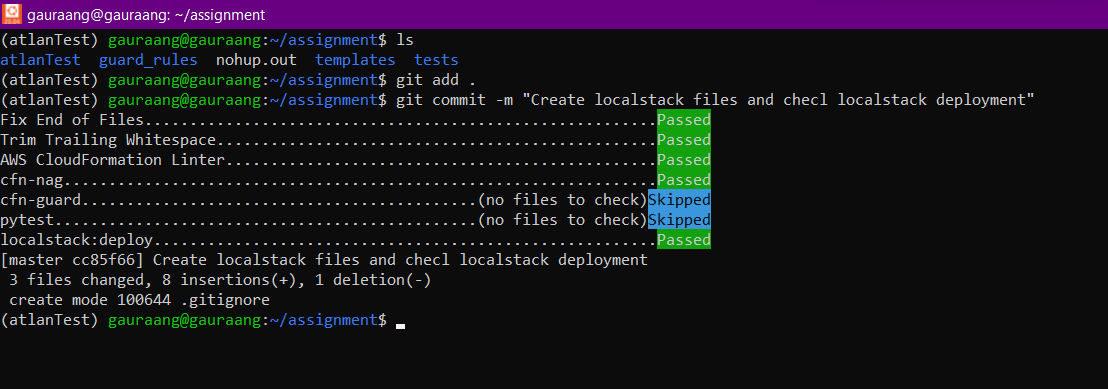
Create a .gitignore file using the following command:

nano .gitignore

And add the following text: nohup.out

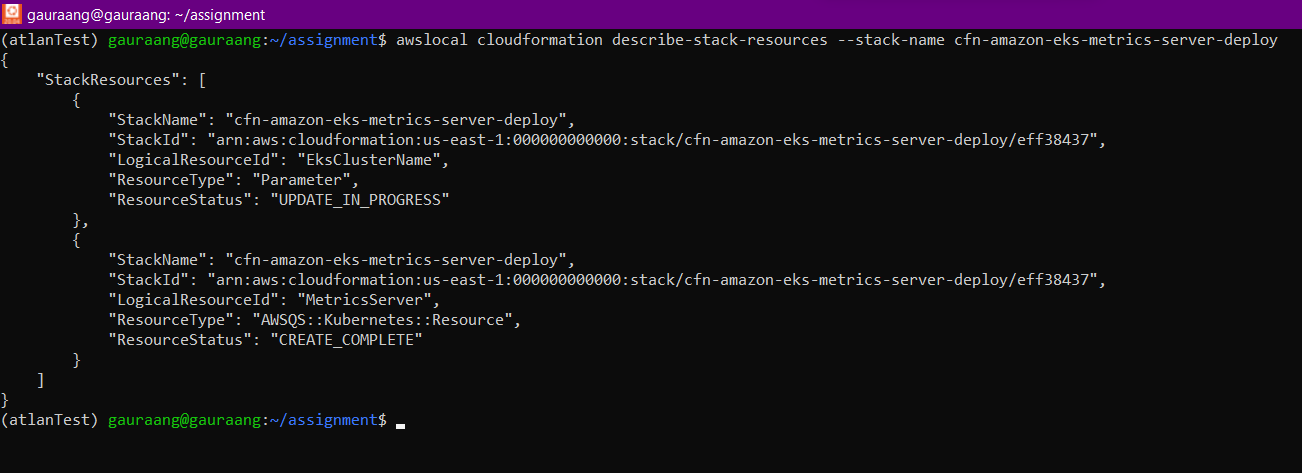


Test the pipeline by staging all files and making a commit



Check whether the localstack was deployed using the following command:

awslocal cloudformation describe-stack-resources --stack-name cfn-amazon-eks-metrics-server-deploy



**Using Github Actions to improve efficiency(Phase-5)**

**Disclaimer: In my solution to make the working more efficient I have used Github actions in the final phase, but I deleted the repository after completing the documentation, as it was mentioned not to upload on GitHub (I only uploaded the code for a very short duration)**

Performing unit testing and deployment testing before a commit can delay the process of a commit, for example, an organization’s code might have 100’s unit test, as such it would be better to perform uni tests on a separate server after the commit.

Similarly deploying resources can take a lot of time as well as computational power

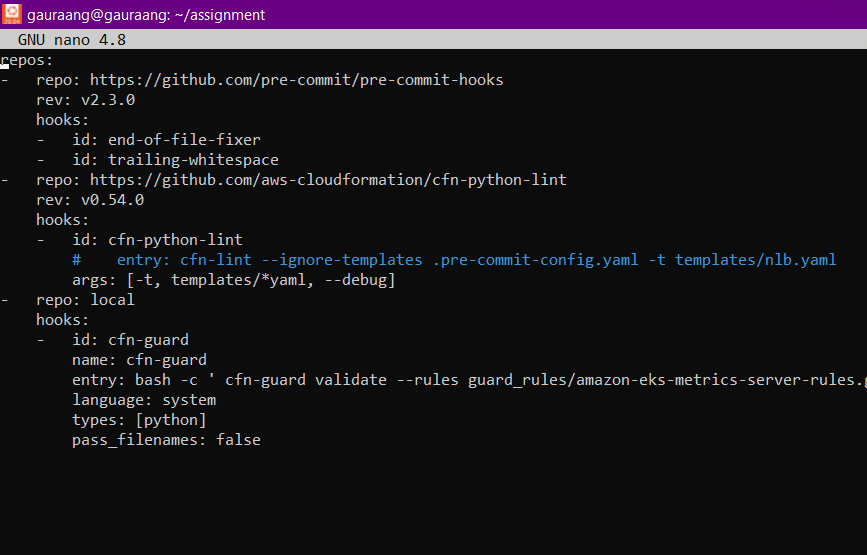
Using GitHub action, we can solve these problems:

GitHub Actions is a CI/CD tool for the GitHub flow. You can use it to integrate and deploy code changes to a third-party cloud application platform as well as test, track, and manage code changes. GitHub Actions also supports third-party CI/CD tools, the container platform Docker, and other automation platforms.

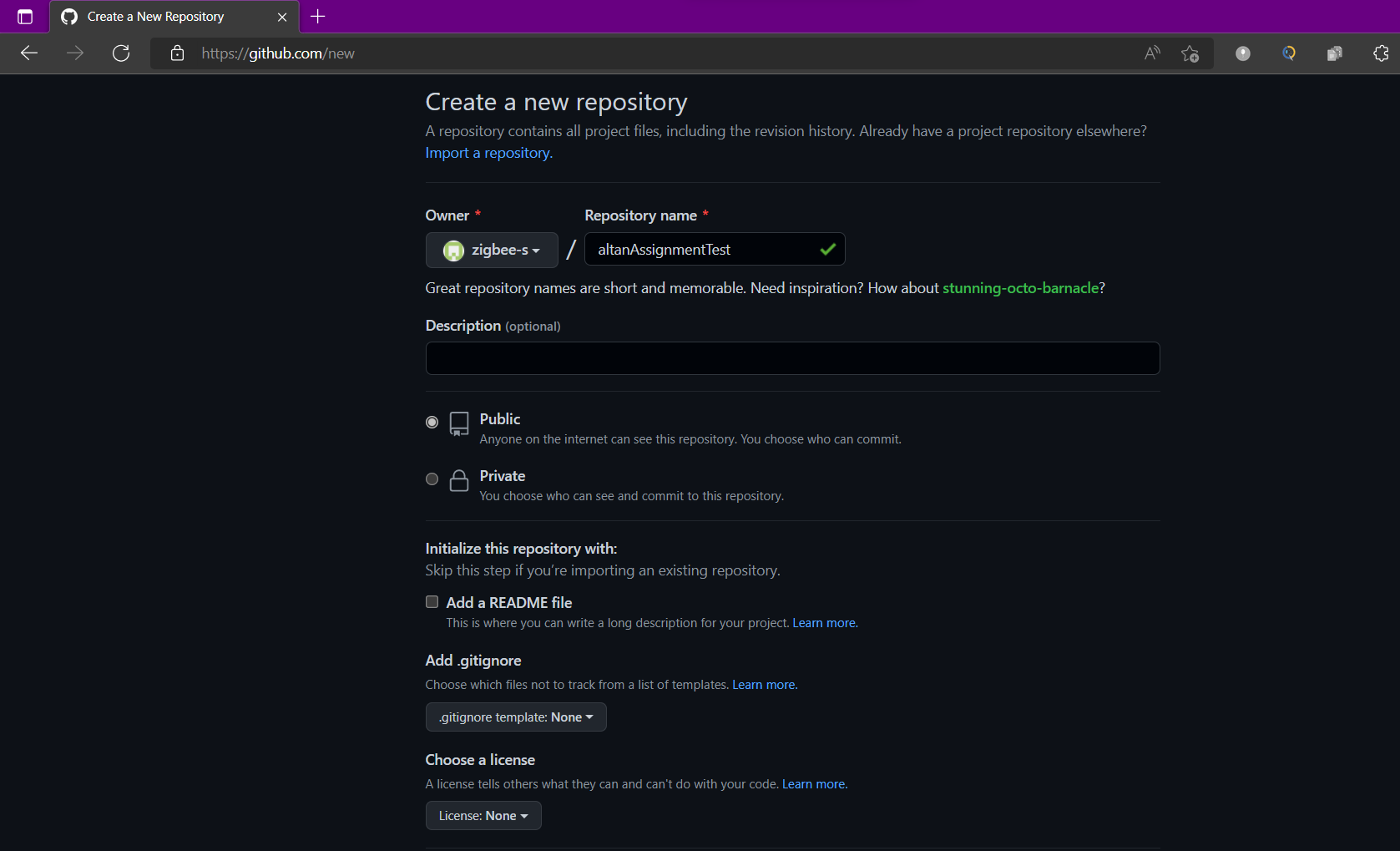
Similarly cfn-nag can also be used in github hooks

**Setup:**

Remove cfn-nag, pytest and localstack hooks from the .pre-commit-config.yaml, iit would finally look like:



Create a repository on GitHub



On the terminal run the following command:

mkdir .github

mkdir .github/workflows

nano .github/worflows/phase5.yaml

And add the following code:

name: Phase5

on:

push:

branches: [ main ]

pull\_request:

branches: [ main ]

permissions:

contents: read

jobs:

build:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- name: Set up Python 3.10

uses: actions/setup-python@v3

with:

python-version: "3.10"

- name: Setup nag

uses: stelligent/cfn\_nag@master

with:

input\_path: templates

- name: Install dependencies

run: |

python -m pip install --upgrade pip

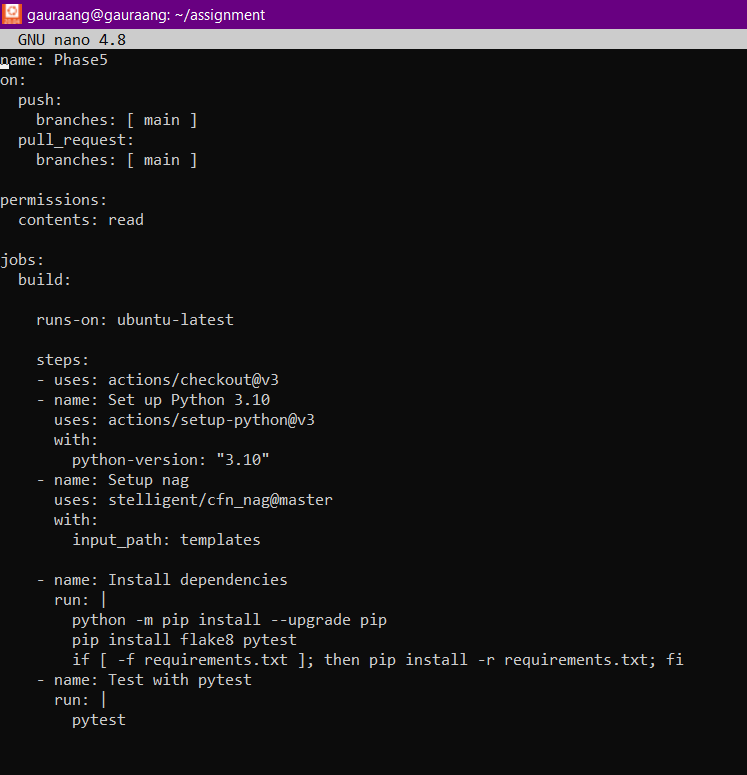
pip install flake8 pytest

if [ -f requirements.txt ]; then pip install -r requirements.txt; fi

- name: Test with pytest

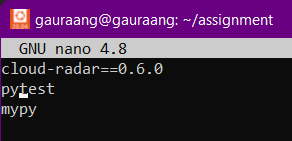
run: |

pytest



Add a requirements.txt file using the following command

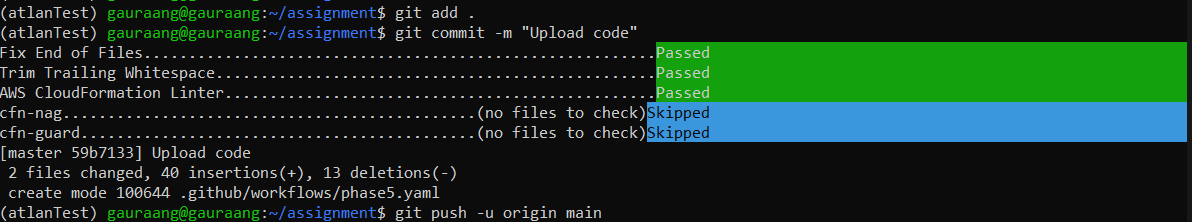
nano requirements.txt

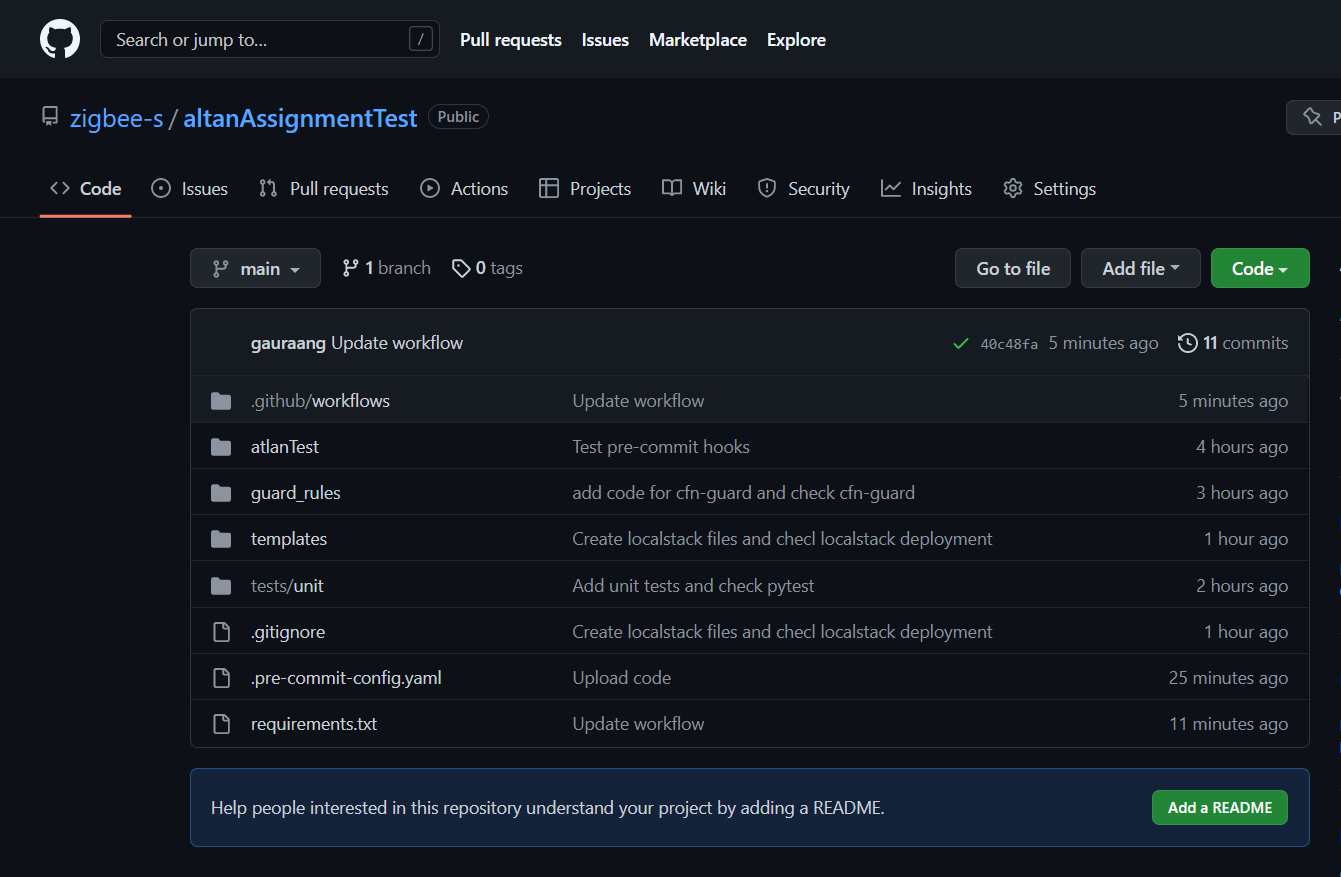


Add remote of your gitHub repository

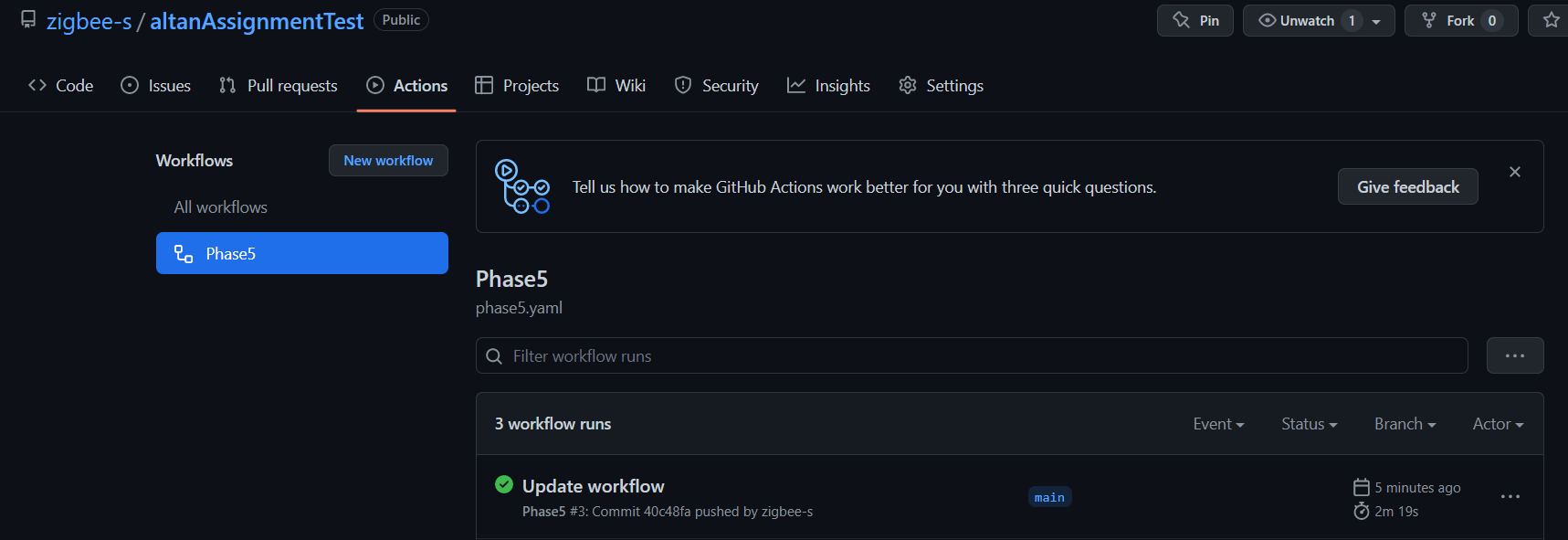
git remote add origin https://github.com/zigbee-s/altanAssignmentTest.git

Stage, commit and push the repo:

****

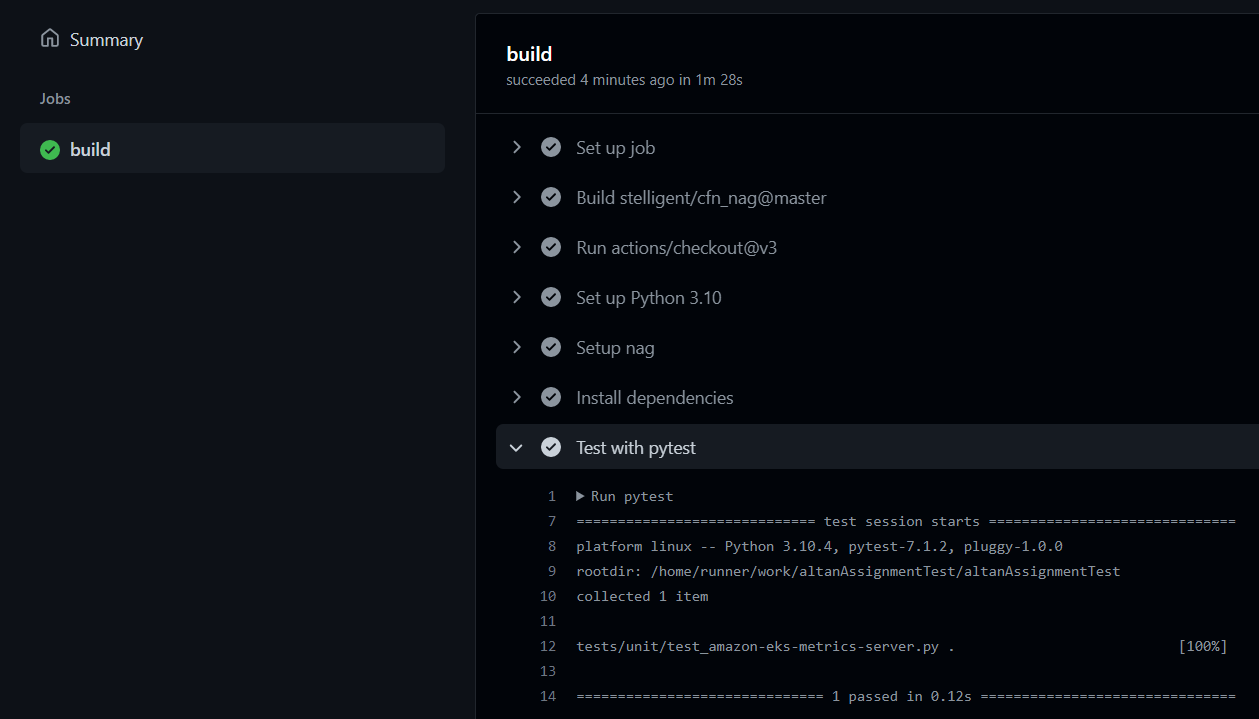


To check the workflow status got to actions > Phase5



Click on update workflow > build

Checking pytest status:



Checking nag status

