**AWS CodePipeline - Deliver DevSecOps**

**Deliver DevSecOps with Checkov**

**Inception**

Hello everyone, In the [previous article](https://eraki.hashnode.dev/aws-codepipeline-automate-iac-provisioning) we built a comprehensive AWS pipeline that includes multiple stages, provisioning AWS resources whereby terrafrom.

Today’s Article, will continue from there, and adjust our pipeline with a new action for scanning terrafrom plan file.

This article is part of The [Terraform + AWS series](https://mohamed-eleraky.hashnode.dev/series/terraform-aws), I use this series to publish-out AWS + Terraform Projects & Knowledge.

**Checkov Overview**

*Checkov is a static code analysis tool for scanning infrastructure as code (IaC) files for misconfigurations that may lead to security or compliance problems. Checkov includes more than 750 predefined policies to check for common misconfiguration issues. Checkov also supports the creation and contribution of*[*custom policies*](https://www.checkov.io/3.Custom%20Policies/Custom%20Policies%20Overview.html)*.*

[*what is checkov*](https://www.checkov.io/1.Welcome/What%20is%20Checkov.html#supported-iac-types)

Checkov is coming up as a Python library providing CLI commands for simplicity, Checkov have the ability to run scans on terraform files, or on the plan result itself. It’s recommended to run the scan on the plan result delivering the following characteristics:

* Plan results represent intended changes to your environment after Terraform evaluates your configurations.
* Checkov can analyze how resources will communicate after the plan is applied.
* Checkov can understand how a resource change will affect others.
* Checkov will provide more accurate results while using terraform variables.

**Practical commands examples**

Before we start configuring our pipeline, let’s practically go through checkov installation and commands. as mentioned checkov is a Python library managed by pypi and pip Python package manager. Therefore, I will install checkov in a virtual environment on my local.

**Create & Activate Python virtual env**

Copy

Copy

python3 -m venv checkov-env

source checkov-env/bin/activate

**Install checkov**

Copy

Copy

pip install checkov

**Configure & Run an inputs *folder or file***

**Configure a folder for scan**

Copy

Copy

checkov --directory /user/path/to/iac/code

**Configure a specific file for scan**

Copy

Copy

checkov --file /user/tf/example.tf

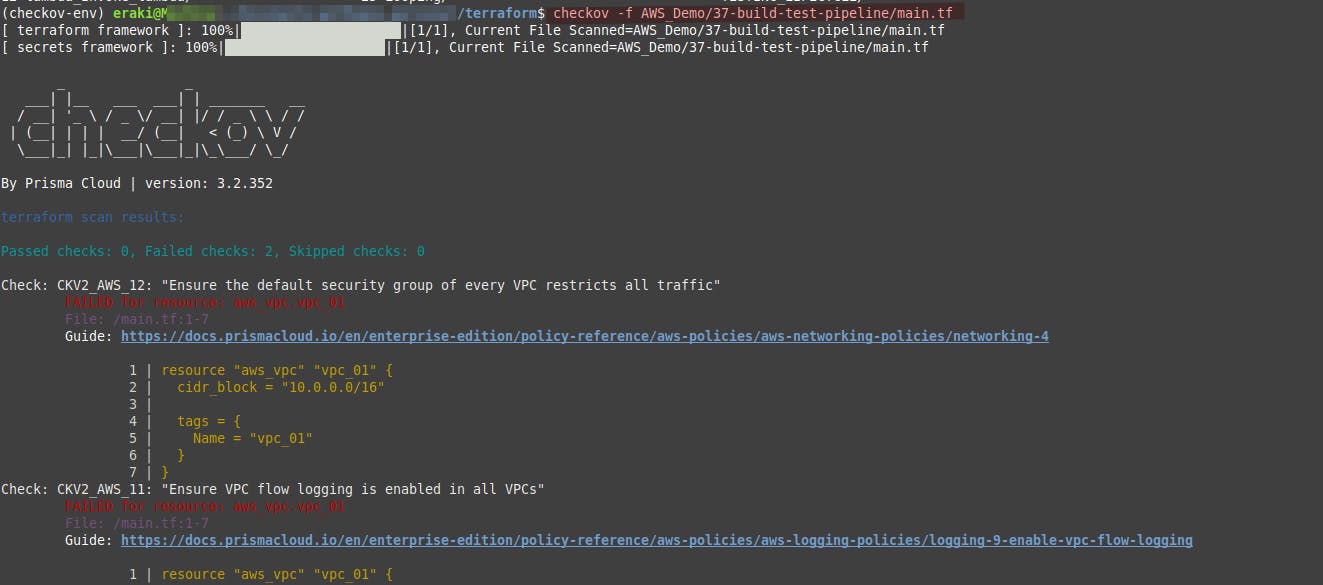
**Configure multiple files for scan**

Copy

Copy

checkov -f /user/cloudformation/example1.yml -f /user/cloudformation/example2.yml

The result of running these commands as follows:

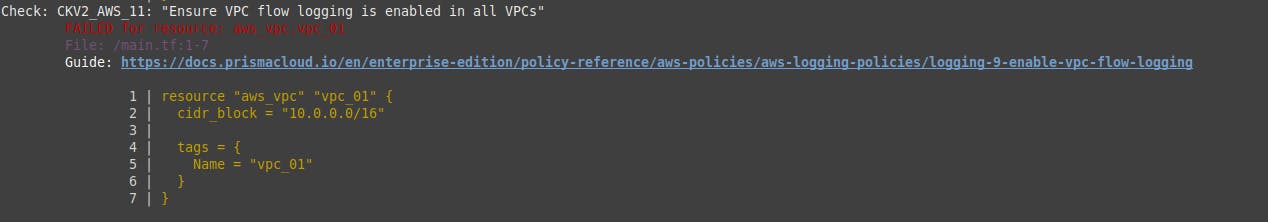


💡

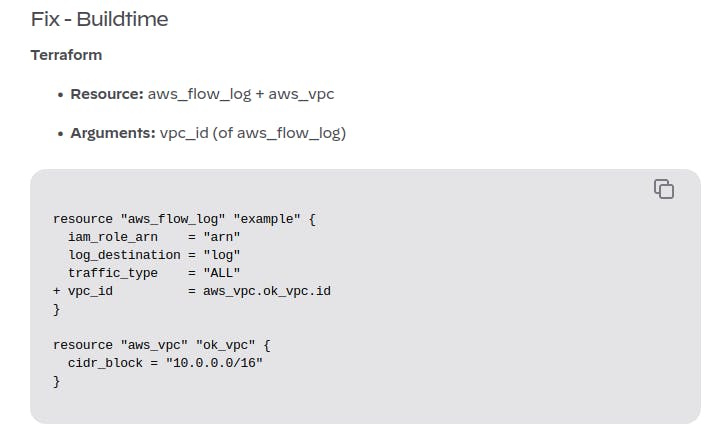
Figure out Checkov result, with each policy result including brief details and a link providing more details and how to configure.

**Checkov result discovering**

let’s take the following screenshot example



Here Checkov recommends enabling the VPC flow log, which captures all network traffic that went through this VPC and saves it in a defined S3. To discover how to fix this open the mentioned link that contains the how-to as follows:



**Configure a Terraform Plan file in JSON**

Copy

Copy

terraform init

terraform plan -out tf.plan # Extract plan in tf.plan file

terraform show -json tf.plan > tf.json # convert it to json file allows checkov read it

checkov -f tf.json

Note: The Terraform show output file tf.json will be a single line. For that reason Checkov will report all findings as line number 0.

Copy

Copy

check: CKV\_AWS\_21: "Ensure all data stored in the S3 bucket have versioning enabled"

FAILED for resource: aws\_s3\_bucket.customer

File: /tf/tf.json:0-0

Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/s3-policies/s3-16-enable-versioning

If you have installed jq, you can convert a JSON file into multiple lines with the command terraform show -json tf.plan | jq '.' > tf.json, making it easier to read the scan result, Then run the following command.

💡

Install jq using your OS package manager as: apt install jq

Copy

Copy

checkov -f tf.json

Check: CKV\_AWS\_21: "Ensure all data stored in the S3 bucket have versioning enabled"

FAILED for resource: aws\_s3\_bucket.customer

File: /tf/tf1.json:224-268

Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/s3-policies/s3-16-enable-versioning

225 | "values": {

226 | "acceleration\_status": "",

227 | "acl": "private",

228 | "arn": "arn:aws:s3:::mybucket",

💡

If you faced an issue with urllib library update is with the following: pip install --upgrade urllib3

💡

Use grep to print a specific policy result: checkov -f tfplan.json | grep CKV\_AWS\_129 -A 3

**CSV output**

Copy

Copy

checkov -d /path/to/your/code --output csv > checkov\_results.csv

checkov -f /path/to/your/file.tf --output csv > checkov\_results.csv

**Ignored checks**

Since the Terraform checks are used for both normal templates and plan files, some of those are not applicable to a plan file. They evaluate the lifecycle block, which is only relevant for the CLI and are not stored in the plan file itself.

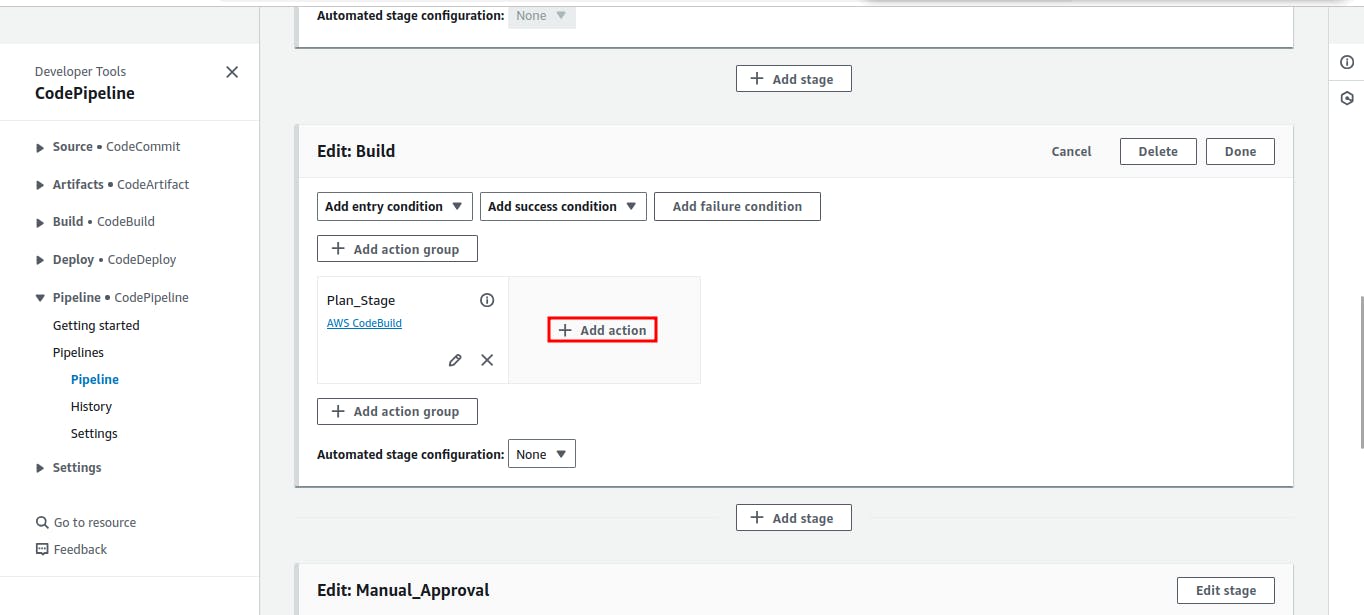
The following checks will be ignored;

* CKV\_AWS\_217
* CKV\_AWS\_233
* CKV\_AWS\_237
* CKV\_GCP\_82

**Update Pipeline Action**

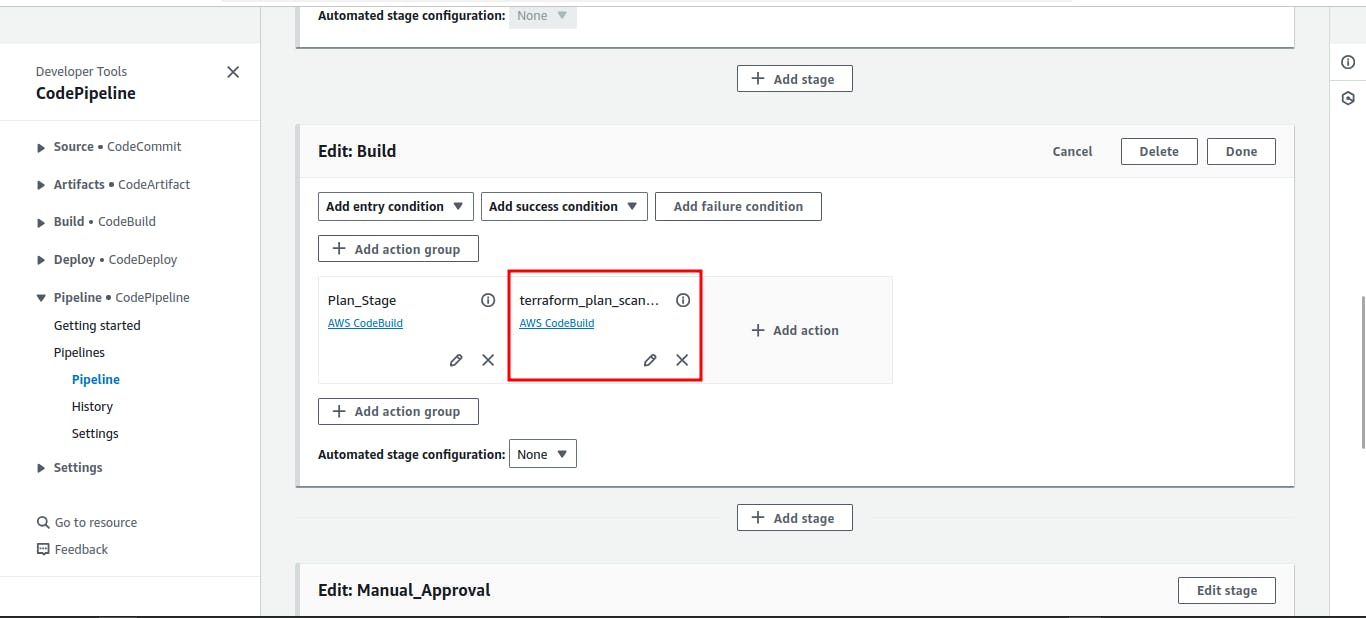
Referring to [AWS CodePipeline Automate IaC provisioning](https://eraki.hashnode.dev/aws-codepipeline-automate-iac-provisioning) Article; let’s update our existing pipeline with plan scanning step action, as follows:

* Open-up the CodePipeline service, Then navigate to “**eraki\_pipeline\_us1\_tagtrigger**“ Pipeline.
* Press **Edit** button, Scroll down to **build** stage, Then **Edit stage.**
* Press Add action to append scan action step.



* For The **Action name** type “**terraform\_plan\_scanning**“
* For Action Provider specify **AWS codeBuild.**
* For **input Artifact** specify **SourceArtifact.**
* For **Project name** specify **eraki\_codebuild\_us1\_tagtrigger.**
* For **Variables** Appen the following variable  
  Key: STAGE\_TYPE Value: SCAN

Hit **Done.**



* Then **Done** on stage level, and **Save** on pipeline level.

**Update**buildspec.yml

At the moment we did configure the stage if scan but didn’t update the buildspec.yml file that holds the commands in this action.

* Update the **install** phase with the following

Copy

Copy

echo "Installing Checkov components"

yum install python -y

pip install --no-input checkov

pip install --upgrade --no-input urllib3

yum install jq -y

* Update the **Pre-build** phase with the following

Copy

Copy

elif [ "$STAGE\_TYPE" = "SCAN" ]; then

echo "Filter on Scan action"

echo "terraform scan started on `date`"

cd "AWS\_Demo/37-build-test-pipeline";

ls -lathr;

terraform init;

terraform plan -out tfplan;

terraform show -json tfplan | jq '.' > tfplan.json;

checkov -f tfplan.json;

Therefore, the **entire** buildspec.yml file will be as follows:

Copy

Copy

version: 0.2

env:

variables:

TERRAFORM\_VERSION: "1.5.6"

#key: "value"

#

# parameter-store:

# key: "value"

#

# secrets-manager:

# key: "value"

phases:

install:

runtime-versions:

python: 3.12

on-failuer: ABORT

commands: |

echo "Installing terraform"

yum install -y wget unzip

yum clean all

tf\_version=$TERRAFORM\_VERSION

wget https://releases.hashicorp.com/terraform/"$TERRAFORM\_VERSION"/terraform\_"$TERRAFORM\_VERSION"\_linux\_amd64.zip

unzip terraform\_"$TERRAFORM\_VERSION"\_linux\_amd64.zip

chmod 775 terraform

mv terraform /usr/local/bin/

terraform --version

rm terraform\_"$TERRAFORM\_VERSION"\_linux\_amd64.zip

ls -al /usr/local/bin/terraform

echo "Installing Checkov components"

yum install python -y

pip install --no-input checkov

pip install --upgrade --no-input urllib3

yum install jq -y

pre\_build:

on-failure: ABORT

commands: |

if [ "$STAGE\_TYPE" = "plan" ]; then

echo "Filter on Plan action";

echo terraform plan started on `date`

#cd "CODEBUILD\_SRC\_DIR/AWS\_Demo/37-build-test-pipeline";

cd "AWS\_Demo/37-build-test-pipeline";

ls -lathr;

terraform init;

terraform validate;

terraform plan -out tfplan;

elif [ "$STAGE\_TYPE" = "SCAN" ]; then

echo "Filter on Scan action"

echo "terraform scan started on `date`"

cd "AWS\_Demo/37-build-test-pipeline";

ls -lathr;

terraform init;

terraform plan -out tfplan;

terraform show -json tfplan | jq '.' > tfplan.json;

checkov -f tfplan.json;

else

echo "No Plan, or Scan stages";

fi

build:

on-failure: ABORT

commands: |

if [ "$STAGE\_TYPE" = "apply" ]; then

echo "Filter on Apply stage";

echo terraform execution started on `date`;

ls -lathr;

#cd "$CODEBUILD\_SRC\_DIR/AWS\_Demo/37-build-test-pipeline";

cd "AWS\_Demo/37-build-test-pipeline";

ls -lathr;

terraform apply tfplan;

else

echo "No Apply stage";

fi

post\_build:

on-failure: CONTINUE

commands: |

echo "Fetching provisioning details"

terraform show -json tfplan > tfplan.json

yum install -y jq

echo "print out terrafrom version and json format version"

jq '.terraform\_version, .format\_version' tfplan.json

echo ""

echo "print out provider config"

jq '.configuration.provider\_config' tfplan.json

echo ""

echo "print out resource config"

jq '.configuration.root\_module.resources' tfplan.json

echo ""

echo "print out outputs"

jq '.outputs' tfplan.json

echo ""

echo "print out resource changes"

jq '.resource\_changes' tfplan.json

echo ""

echo "print out resource config"

jq '.configuration.root\_module.resources' tfplan.json

echo ""

echo "print out provider config"

jq '.configuration.provider\_config' tfplan.json

echo ""

echo "print out provider config"

jq '.configuration.provider\_config' tfplan.json

echo ""

echo "print out lock file configuration"

jq '.configuration.lock\_version' tfplan.json

💡

Make sure the buildspec.yml file exists in the main directory. check here for more info <https://github.com/Mohamed-Eleraki/terraform/blob/main/buildspec.yml>

and now we are ready to test our pipeline.

**Fire the Pipeline**

After updating your buildspec.yml file locally now it’s time to push and tag our commit, follow up:

* Save buildspec.yml file.
* Push your update as follows

Copy

Copy

git add buildspec.yml

git commit -m "pipeline/Prov Scan stg"

git push

* Tag your commit and push it, [Check previous article](https://eraki.hashnode.dev/aws-codepipeline-automate-iac-provisioning#heading-build-a-new-release) for more info

Copy

Copy

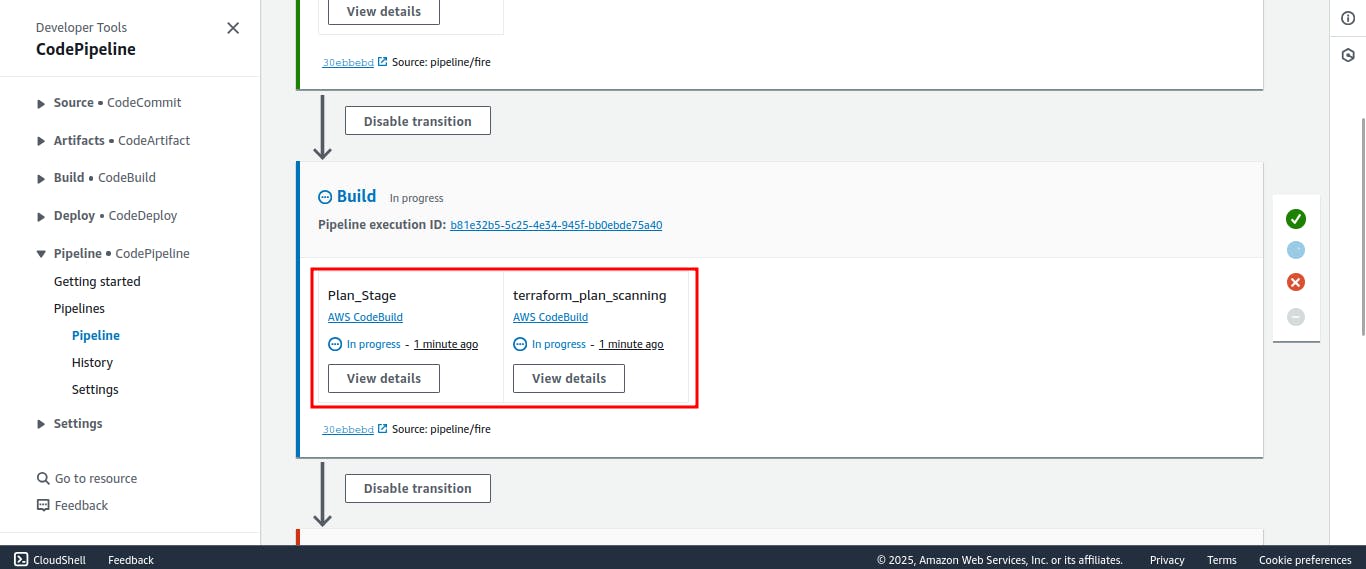
git tag <commit\_hash> release-21

git push origin release-21

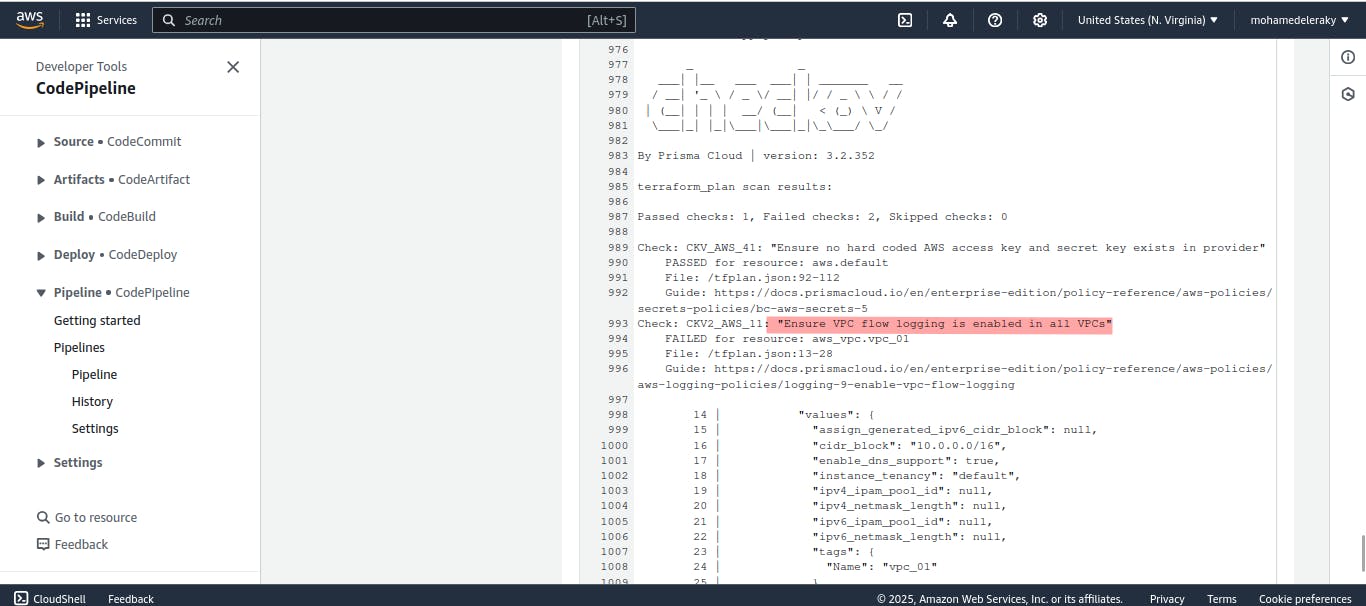
Everything up-to-date

This should fire the pipeline as it’s fired up by the release-\* tag; Go to your pipeline and it should appear results as below screenshots:

**In parallel actions**



The Scan Action will fail because we faced a security issue we needed to make our code match the checkov security recommendations; However, that means we did configure our pipeline successfully



**AWS Pipeline Notifications**

As we configured last time your pipeline should send you notifications as follows, [Check previous article](https://eraki.hashnode.dev/aws-codepipeline-automate-iac-provisioning#heading-set-up-a-cloudwatch-events-rule-to-receive-email-notifications-for-pipeline-state-changes) for more info:

