Using AWS CodePipeline for secure CloudFormations deployments

AWS offers a rich set of developer tools to host code, build, and deploy your application and/or infrastructure to AWS. These include AWS CodePipeline, for continuous integration and continuous deployment orchestration; AWS CodeCommit, a fully-managed source control service; AWS CodeBuild, a fully-managed continuous integration service that compiles source code, runs tests, and produces software packages; and AWS CodeDeploy, a fully-managed deployment service that automates software deployments to a variety of compute services. These services can be used together to build powerful CI/CD pipelines on AWS.

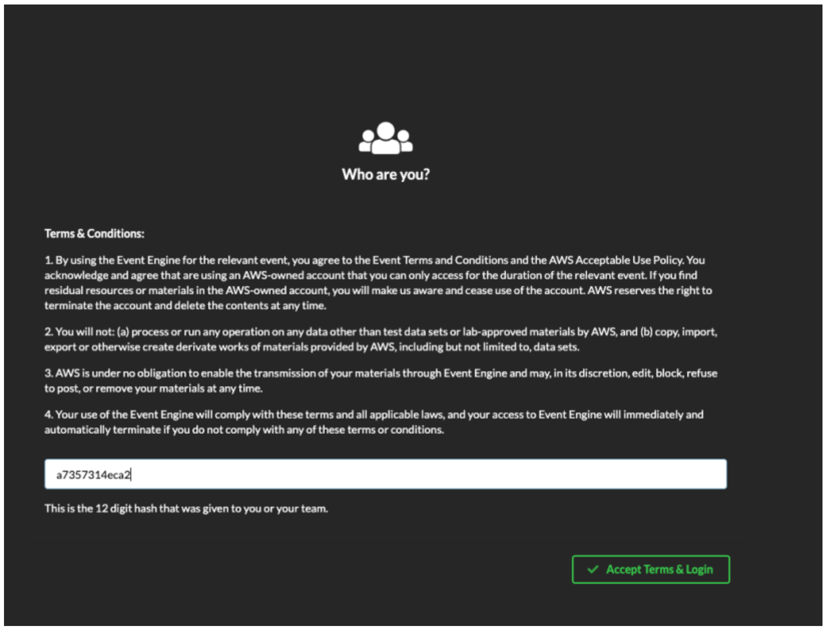
In addition, AWS developer tools allow you to leverage open source software (OSS) to build CI/CD pipelines on AWS, so you can continue to use the tools you are familiar with. For instance, you can use GitHub as the source code provider for AWS CodePipeline or run CodeBuild jobs orchestrated by Jenkins. In fact, a variety of OSS and third-party tools can be easily integrated with AWS CodeBuild, allowing you to extend the capabilities of your pipeline. As an extra benefit, you are not required to manage build servers since CodeBuild is a fully-managed service.

In this lab, we will show you how to build a serverless infrastructure deployment pipeline (i.e., no need for you to manage build servers) on AWS using AWS developer tools in conjunction with popular open source tools such as CFN-Nag, CFN-Python-Lint, and Stacker. The pipeline will run automated validation checks against CloudFormation templates and deploy the corresponding CloudFormation stacks if the templates are valid.

The companion source code for this Lab can be found in the GitHub repo aws-codepipeline-secure-cfn-lab.

Lets's get started!

1. Go to https://dashboard.eventengine.run/login and enter the hash you've received and Login



Who are you?

Terms & Conditions:

1. By using the Event Engine for the relevant event, you agree to the Event Terms and Conditions and the AWS Acceptable Use Policy. You acknowledge and agree that are using an AWS-owned account that you can only access for the duration of the relevant event. If you find residual resources or materials in the AWS-owned account, you will make us aware and cease use of the account. AWS reserves the right to terminate the account and delete the contents at any time.

2. You will not: (a) process or run any operation on any data other than test data sets or lab-approved materials by AWS, and (b) copy, import, export or otherwise create derivate works of materials provided by AWS, including but not limited to, data sets.

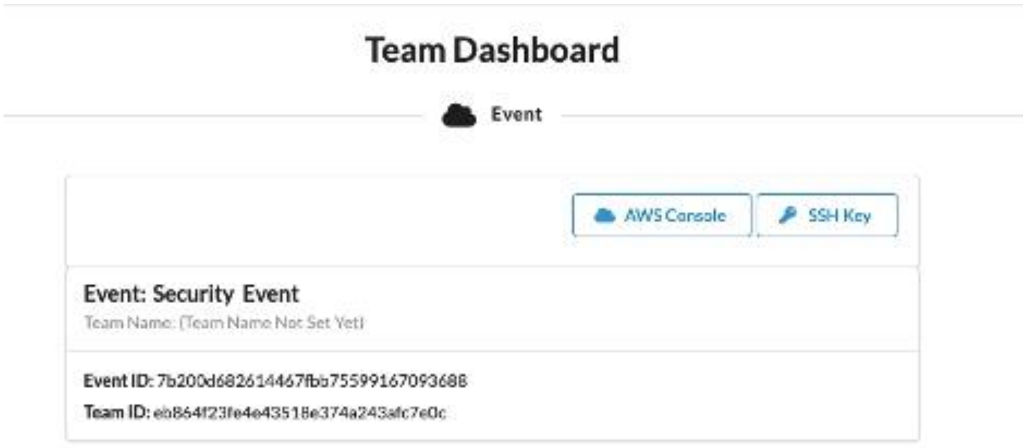
3. AWS is under no obligation to enable the transmission of your materials through Event Engine and may, in its discretion, edit, block, refuse to post, or remove your materials at any time.

4. Your use of the Event Engine will comply with these terms and all applicable laws, and your access to Event Engine will immediately and automatically terminate if you do not comply with any of these terms or conditions.

a7357314eca2

This is the 12 digit hash that was given to you or your team.

You now have access to your Team dashboard. It will provide you with a link and credentials to the AWS console



Team Dashboard

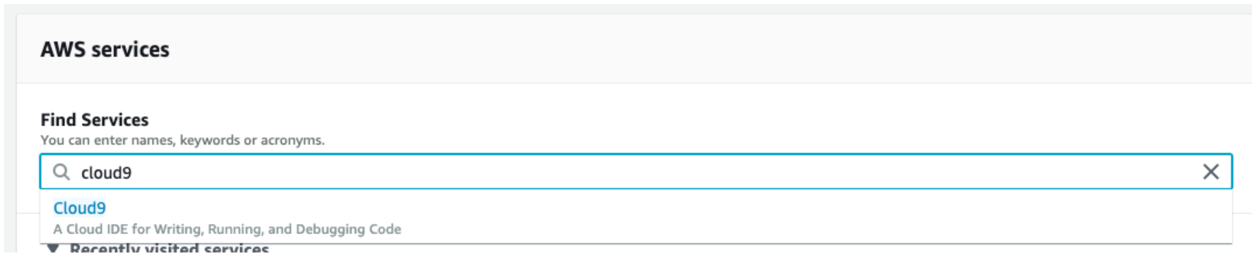
Event: Security Event

Team Name: (Team Name Not Set Yet)

Event ID: 7b200d682614467fbb75599167093688

Team ID: eb864f23fe4e43518e374a243afc7e0c

2. Next you will need a development environment. We will use AWS Cloud9 for that. Enter Cloud9 into the search bar and press enter.



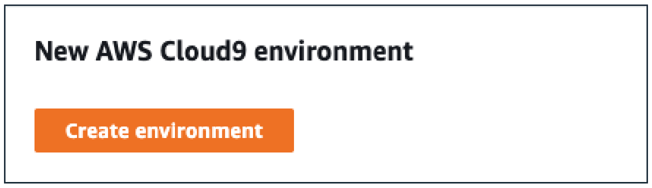
AWS services

Find Services
You can enter names, keywords or acronyms.

cloud9

Cloud9
A Cloud IDE for Writing, Running, and Debugging Code

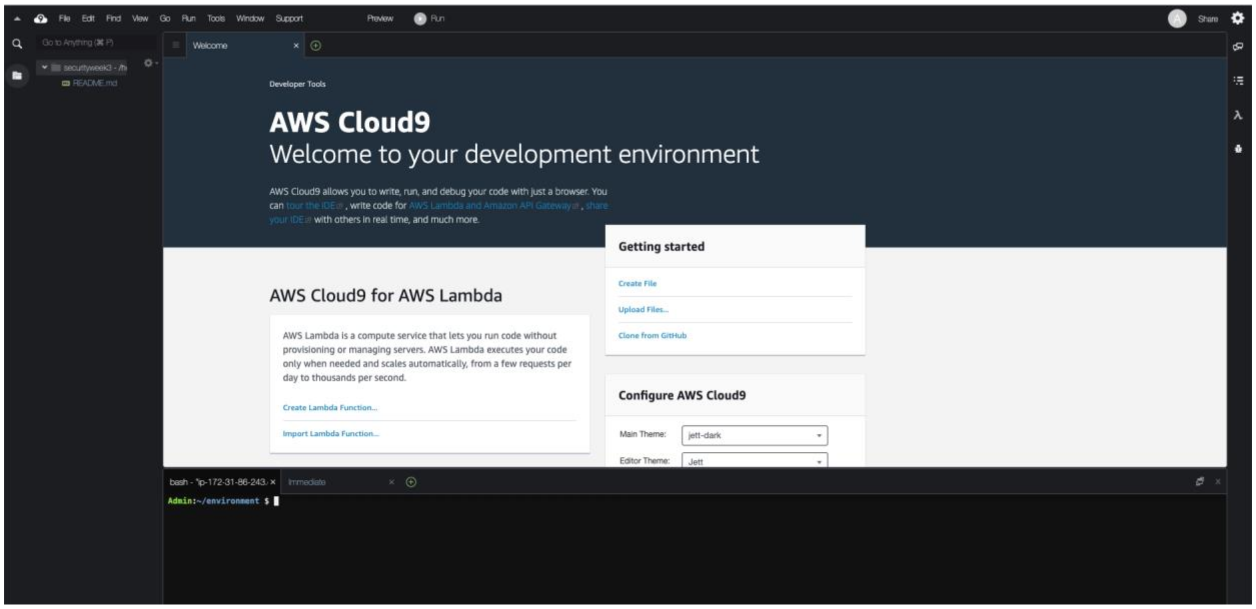
This will lead you to the Developer Tools page where you can click Create environment button.



New AWS Cloud9 environment

Enter a name for your environment and click Next step on the bottom right.

For configuration settings leave the default, choose **Instance Type** *t3.small* and for **Platform** *Amazon Linux 2*. In terms of **cost saving settings** keep the default after **30 minutes**. Then create the environment. You will see the following screen when the environment is up:



Developer Tools

AWS Cloud9
Welcome to your development environment

AWS Cloud9 allows you to write, run, and debug your code with just a browser. You can tour the IDE, write code for AWS Lambda and Amazon API Gateway, share your IDE with others in real time, and much more.

Getting started

Create File
Upload Files...
Clone from GitHub

AWS Cloud9 for AWS Lambda

AWS Lambda is a compute service that lets you run code without provisioning or managing servers. AWS Lambda executes your code only when needed and scales automatically, from a few requests per day to thousands per second.

Create Lambda Function...
Import Lambda Function...

Configure AWS Cloud9

Main Theme: jett-dark
Editor Theme: Jett

bash - "ip-172-31-86-243. Immediate

```
Admin:~/environment $
```

Your environment is now ready.

In the next few steps we will install various tooling so that you can check your templates locally on your Cloud9 environment.

Open source tools

First, let's briefly review the open source tools that will be later used to build the infrastructure pipeline discussed in this lab.

CFN-Nag

CFN-Nag is a popular open source tool developed by Stelligent and provided to the open source community to help pinpoint security problems early on in an AWS CloudFormation template. CFN-Nag looks for patterns in AWS CloudFormation templates that may indicate insecure infrastructure, for example:

- 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

To install CFN-Nag from your workstation, you'll need Ruby v.2.5 or greater installed. Assuming that you have met this prerequisite, simply run:

```
Admin:~/environment $ gem install cfn-nag
…
Done installing documentation for kwalify, cfn-model, multi_json, little-plugger, logging, netaddr, optimist, jmespath,
aws-partitions, aws-eventstream, aws-sigv4, aws-sdk-core, aws-sdk-kms, aws-sdk-s3, lightly, cfn-nag after 8 seconds
16 gems installed
```

Later in the lab when you have a CloudFormations template you can run CFN-Nag from the command line as such:

```
Admin:~/environment $ cfn_nag_scan --input-path <path to cloudformations json>
```

CFN-Lint

CFN-Python-Lint was released by AWS to the open source community as a CloudFormation linter. It validates AWS CloudFormation templates (YAML and JSON) against the AWS CloudFormation spec and does additional checks including ensuring valid values for resource properties, and best practices – for instance, if an input parameter is defined in a template but never referenced, the linter will raise a warning.

Install CFN-Python-Lint in your Cloud9 environment you can use pip3:

```
Admin:~/environment $ pip3 install cfn-lint
…
Successfully installed attrs-20.2.0 aws-sam-translator-1.27.0 boto3-1.15.17 botocore-1.18.18 cfn-lint-0.38.0
decorator-4.4.2 jmespath-0.10.0 jsonpatch-1.26 jsonpointer-2.0 jsonschema-3.2.0 junit-xml-1.9 networkx-2.5
pyrsistent-0.17.3 python-dateutil-2.8.1 pyyaml-5.3.1 s3transfer-0.3.3 six-1.15.0 urllib3-1.25.10
```

To invoke CFN-Python-Lint from the command line later in the lab you can run the following command:

```
Admin:~/environment $ cfn-lint <path to yaml template>
```

CFN-Guard

AWS CloudFormation-Guard was released by AWS to the open source community as a CLI tool that checks AWS CloudFormation templates for policy and compliance using a simple, policy-as-code, declarative syntax. CFN-guard can replace CFN-Nag. Compared to CFN-nag, cnf-guard does not require you to learn ruby or implement classes to check the things you want to validate. In this lab we will run both so you can get a comparison. CloudFormation-guard also allows you to autogenerate rules based of a CloudFormation's template.

For this lab, we have created a cfn-guard rpm which you can install directly from the Github repository.

```
Admin:~/environment $ sudo yum install -y cargo rust  && sudo rpm -ivh https://bit.ly/31CMLw3
```

To invoke cnf-guard from the command line later in the lab you can run the following command:

```
Admin:~/environment $ cfn-guard check --rule_set test.rules --template yourtemplate.yaml
```

Stacker

Stacker is an open source tool and library created by the Remind engineering team and released to the open source community. It is used to orchestrate and manage CloudFormation stacks across one or more accounts. Stacker can manage stack dependencies automatically and allow the output parameters of a given stack to be passed as input to other stacks (even for stacks deployed in different accounts!) making sure the stacks are created in the right order. It can also parallelize the deployment of non-dependent stacks reducing deployment time significantly.

Install Stacker in your Cloud9 environment by using pip3:

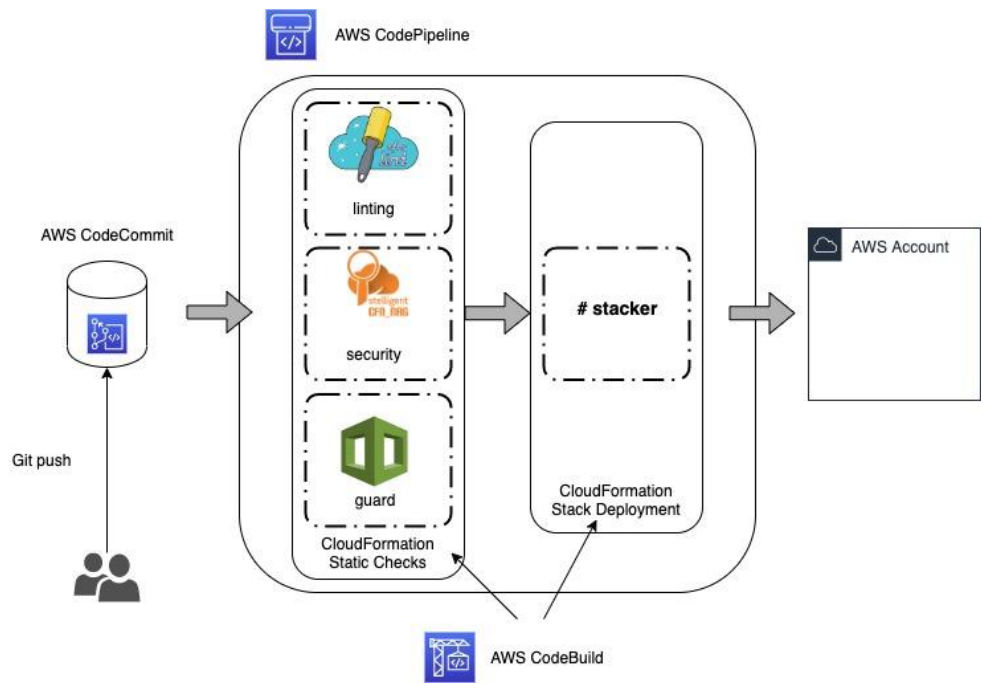
```
Admin:~/environment $ pip3 install stacker
…
Successfully installed Click-7.1.2 MarkupSafe-1.1.1 awacs-1.0.0 cfn-flip-1.2.3 ecdsa-0.16.0 formic2-1.0.3 future-0.18.2 gitdb2-2.0.6 gitpython-2.1.15 jinja2-2.11.2 more-itertools-5.0.0 pyasn1-0.4.8 python-jose-3.1.0 rsa-4.5 schematics-2.0.1 smmap-3.0.4 smmap2-3.0.1 stacker-1.7.1 troposphere-2.6.3
```

To run this command later in the lab, use the following command:

```
Admin:~/environment $ stacker build <path to cloudformation yaml or json template>
```

Stacker will either launch or update your AWS CloudFormation stacks based on their configurations. Stacker can detect if the templates or their parameters have changed. If no changes are detected to a given stack, Stacker will not update that particular stack.

Overview of the secure infrastructure pipeline



The figure above shows the infrastructure deployment pipeline we will build in this lab. It uses AWS services such as AWS CodePipeline, AWS CodeCommit, and AWS CodeBuild in conjunction with open source software such as CFN-Nag, CFN-Python-Lint, CFN-Guard and Stacker.

The pipeline works as follows: a developer creates CloudFormation templates to provision infrastructure resources on AWS (e.g., a VPC with subnets, route tables, security groups, etc). When ready, the developer pushes the templates to an AWS CodeCommit repository which will then trigger the infrastructure pipeline. OSS tools CFN-Nag, CFN -Lint and CFN-Guard will run in parallel within a CodeBuild environment to validate the templates. CFN-Nag will check the templates against well-known security vulnerabilities while CFN-Python-Lint will enforce best practices and verify compliance with the CloudFormation specification. CFN-Guard will make sure that the templates are up to your defined standard/guardrails. If the templates do not pass the verification tests, the pipeline will stop in a failure state to prevent deployment issues. If the templates pass verification, Stacker will create or update CloudFormation stacks for each template. If Stacker fails, the pipeline will stop in a failure state to indicate a deployment issue. Otherwise, the stacks will be created or updated to provision the desired AWS resources.

As you might have noticed, the infrastructure pipeline described is generic and can be used to deploy arbitrary resources on AWS across a number of AWS accounts, while enforcing verification rules against the submitted templates. For simplicity, we use Stacker to deploy to a single AWS account in this lab, but the tool really shines in the context of large cross-account deployments.

What will be deployed using the pipeline?

In this lab, we will use the infrastructure deployment pipeline to deploy three simple stacks as follows:

Stack #1: creates a private S3 bucket.

Stack #2: creates an EC2 IAM role that allows putting objects in the S3 bucket created in stack #1.

Stack #3: creates an EC2 instance and EC2 profile that assumes the IAM role created in stack #2 and uploads a simple file to the S3 bucket created in stack #1. Stack #3 depends on stacks #1 and #2.

Below is a snippet of the Stacker configuration file we use to create the three stacks we just mentioned:

```
#------------------------------------------------------------------------#
# Global variables
#------------------------------------------------------------------------#
namespace: cfnsec
stacker_execution_profile: &stacker_execution_profile stacker_execution
stacker_bucket: ""  # not using S3 buckets to store CloudFormation templates

# any unique S3 bucket - suggestion: append your account id and region to the name
s3_bucket_name: &s3_bucket_name cfnsecsamples3bucket182743192031
# cheapest EC2 instance for testing purposes
ec2_instance_type: &ec2_instance_type t2.micro

#------------------------------------------------------------------------#
# Stack Definitions (https://stacker.readthedocs.io/en/latest/config.html)
#------------------------------------------------------------------------#
stacks:
  - name: sample-s3-bucket
    profile: *stacker_execution_profile
    template_path: templates/s3-bucket-template.yaml
    variables:
      S3BucketName: *s3_bucket_name

  - name: sample-ec2-iam-role
    profile: *stacker_execution_profile
    template_path: templates/ec2-role-template.yaml
    variables:
      S3BucketName: ${output sample-s3-bucket::S3BucketName}

  - name: sample-ec2-instance
    profile: *stacker_execution_profile
    template_path: templates/ec2-template.yaml
    variables:
      InstanceType: *ec2_instance_type
      S3BucketName: ${output sample-s3-bucket::S3BucketName}
    EC2IamRoleName: ${output sample-ec2-iam-role::EC2RoleName}
```

At the top of the file we define variables. The namespace variable is reserved for Stacker and used to prefix stack names. The other variables defined use [YAML anchors]() that can be referenced in other parts of the template. For instance, the variable `ec2_instance_type` creates an anchor `ec2_instance_type` that refers to value "t2.micro". This anchor value is later assigned to input variable `InstanceType` for stack `sample-ec2-instance` (see *"InstanceType: *ec2_instance_type"* above in the snippet of the Stacker configuration file.

The **stacks session** defines our three stacks. For each stack, we indicate the name of the stack, the AWS profile (credentials) that Stacker should use to deploy the stack, the local path to the CloudFormation template, and the input parameters to the stack. Notice that we use `${output …}` to refer to stack output parameters and pass those as input parameters to other stacks. For instance, the S3BucketName output parameter from stack sample-s3-bucket is used as an input parameter by the other two stacks.

We start by provisioning the infrastructure pipeline that we introduced earlier.

3. First we need to get the source code. Clone the Git repository as shown below.

```
Admin:~/environment $ git clone https://github.com/ldomb/aws-codepipeline-secure-cfn-lab.git sample-cfnsec-pipeline
…
Cloning into 'sample-cfnsec-pipeline'...
remote: Enumerating objects: 32, done.
remote: Counting objects: 100% (32/32), done.
remote: Compressing objects: 100% (28/28), done.
remote: Total 32 (delta 7), reused 25 (delta 3), pack-reused 0
Unpacking objects: 100% (32/32), done.
```

Change into the directory you just cloned sample-cnfsec-pipeline directory and list the content:

```
Admin:~/environment $ cd sample-cfnsec-pipeline && ls -l
…
total 16
drwxrwxr-x 2 ec2-user ec2-user   24 Oct 17 11:45 cfnguard
-rw-rw-r-- 1 ec2-user ec2-user  309 Oct 17 11:45 CODE_OF_CONDUCT.md
drwxrwxr-x 2 ec2-user ec2-user   84 Oct 17 11:45 codepipeline
-rw-rw-r-- 1 ec2-user ec2-user 3306 Oct 17 11:45 CONTRIBUTING.md
-rw-rw-r-- 1 ec2-user ec2-user  932 Oct 17 11:45 LICENSE
-rw-rw-r-- 1 ec2-user ec2-user 1278 Oct 17 11:45 README.md
drwxrwxr-x 2 ec2-user ec2-user   96 Oct 17 11:45 rpms
drwxrwxr-x 2 ec2-user ec2-user   79 Oct 17 11:45 stacker
drwxrwxr-x 2 ec2-user ec2-user   92 Oct 17 11:45 templates
```

You should then see the following file structure:

- **codepipeline/:** CloudFormation templates to deploy the CI/CD pipeline including AWS CodePipeline, AWS CodeCommit, AWS CodeBuild and OSS CFN-Nag, CFN-Lint, and Stacker)
- **stacker:** Configuration files required by Stacker to perform CloudFormation stack deployments
- **stacker/buildspec.yaml:** CodeBuild buildspec that will install and invoke Stacker for CFN provisioning
- **stacker/stacker-config.yaml:** Stacker config file containing stack descriptions and input parameters
- **stacker/stacker-profiles:** AWS profiles file Stacker will use to deploy the various stacks
- **templates/*:** the sample templates for the the stacks mentioned above that will be validated and deployed by the pipeline
- **rpms/:** This is the rpm to install cnf-guard
- **cfnguard:** here you will find a sample rules template

4. Create a local git repo for the pipeline artifacts

Now that you have the original source code, you'll create a local Git repository and copy the entire content of sample-cfnsec-pipeline over.

Assuming you have cloned the provided Git repository into folder **/home/ec2-user/environment/sample-cfnsec-pipeline/** you can make a directory named /**home/ec2-user/environment/sample-pipeline-artifacts/**. Then copy the entire directory sample-cfnsec-pipeline to this directory. The contents of this directory will be pushed to the repository that we will create shortly into AWS CodeCommit.

```
Admin:~/environment $ mkdir sample-pipeline-artifacts
Admin:~/environment $ cd sample-pipeline-artifacts
Admin:~/environment/sample-pipeline-artifacts $ git init
Admin:~/environment/sample-pipeline-artifacts $ cp -r ../sample-cfnsec-pipeline/* .
Admin:~/environment/sample-pipeline-artifacts $ git add .
```

We now added all the files to the local staging. We can check that with the following command

```
Admin:~/environment/sample-pipeline-artifacts $ git status
…
git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
```

```
        new file:   CODE_OF_CONDUCT.md
        new file:   CONTRIBUTING.md
        new file:   LICENSE
        new file:   README.md
        new file:   cfnguard/test.rules
        new file:   codepipeline/codepipeline-template.yaml
        new file:   codepipeline/stacker-execution-role-template.yaml
        new file:   rpms/cfn-guard-1.0-0.amzn2.x86_64.rpm
        new file:   rpms/cfn-guard-debuginfo-1.0-0.amzn2.x86_64.rpm
        new file:   stacker/buildspec.yaml
        new file:   stacker/stacker-config.yaml
        new file:   stacker/stacker-profiles
        new file:   templates/ec2-role-template.yaml
        new file:   templates/ec2-template.yaml
        new file:   templates/s3-bucket-template.yaml
```

Before we commit anything, we need to setup our environment

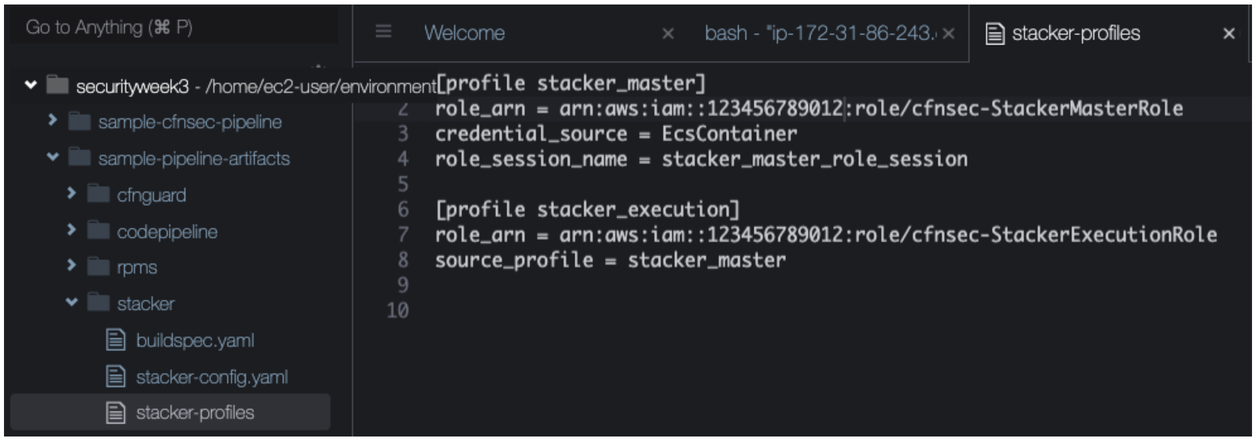
```
Admin:~/environment/sample-pipeline-artifacts $ git config --global Firstname.Lastname “John Doe”
Admin:~/environment/sample-pipeline-artifacts $ git config --global user.email john@example.com
```

Next we can commit all newly added files to our local repository:

```
Admin:~/environment/sample-pipeline-artifacts (master) $ git commit -a -m 'sample artifacts for OSS infrastructure pipeline'

[master (root-commit) af7f464] sample artifacts for OSS infrastructure pipeline
…
 15 files changed, 938 insertions(+)
 create mode 100644 CODE_OF_CONDUCT.md
 create mode 100644 CONTRIBUTING.md
 create mode 100644 LICENSE
 create mode 100644 README.md
 create mode 100644 cfnguard/test.rules
 create mode 100644 codepipeline/codepipeline-template.yaml
 create mode 100644 codepipeline/stacker-execution-role-template.yaml
 create mode 100644 rpms/cfn-guard-1.0-0.amzn2.x86_64.rpm
 create mode 100644 rpms/cfn-guard-debuginfo-1.0-0.amzn2.x86_64.rpm
 create mode 100644 stacker/buildspec.yaml
 create mode 100644 stacker/stacker-config.yaml
 create mode 100644 stacker/stacker-profiles
 create mode 100644 templates/ec2-role-template.yaml
 create mode 100644 templates/ec2-template.yaml
 create mode 100644 templates/s3-bucket-template.yaml
```

5. Edit the Stacker-Profile file

   Next, in the text editor in Cloud9 open the file *stacker/stacker-profiles* by double clicking it.



```
[profile stacker_master]
role_arn = arn:aws:iam::123456789012:role/cfnsec-StackerMasterRole
credential_source = EcsContainer
role_session_name = stacker_master_role_session

[profile stacker_execution]
role_arn = arn:aws:iam::123456789012:role/cfnsec-StackerExecutionRole
source_profile = stacker_master
```

This file specifies the AWS profiles available in the CodeBuild environment running Stacker. Each profile refers to a specific **IAM Role**. The *stacker_master* profile is used to invoke Stacker from within the AWS CodeBuild environment. Stacker assumes that IAM role to access resources in the account where the pipeline is deployed. In order to perform stack deployments, Stacker needs one or more profiles. Since we're deploying to a single target account (which in our example is the same account where the pipeline is deployed), we define a single profile named *stacker_execution*. If we had multiple target accounts, we could define multiple profiles (e.g., stacker_execution_accountA, stacker_execution_accountB, etc.)

You'll need to modify the AWS account id **123456789012** specified in **stacker/stacker-profiles** and replace that value with your own AWS account ID.

To find your own aws account ID run the following command:

```
Admin:~/environment/sample-pipeline-artifacts (master) $ aws sts get-caller-identity | grep -i account | awk {'print
$2'}
…
"546784563423",
```

This gives Stacker the target account for the deployment. After you update your account ID, save the edited file. Then run a git status to see that the file is recognized as modified by git:

```
Admin:~/environment/sample-pipeline-artifacts (master) $ git status
…
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   stacker/stacker-profiles
```

Next we can add the file to our staging area and commit it.

```
Admin:~/environment/sample-pipeline-artifacts (master) $ git add .
```

```
Admin:~/environment/sample-pipeline-artifacts (master) $ git commit -a -m 'Updated target AWS account ID in stacker-profiles'
…
[master 9f2ec01] Updated target AWS account ID in stacker-profiles
 1 file changed, 2 insertions(+), 2 deletions(-)
```

6. Edit the Stacker configuration file

In Cloud9, open the file ***stacker/stacker-config.yaml***. As discussed, this file contains parameters used by Stacker to create AWS resources for you.

Update parameter **s3_bucket_name** on **line 9** with a globally unique name **cfnsecsamples3bucket*Your_AWS_ACCOUNT_ID*** that will be created by Stacker as part of the CloudFormation stack deployment. This is just a simple private S3 bucket that will be created for you by Stacker. Also, note that the parameter *ec2_instance_type* indicates the EC2 instance type to deploy. By default, a “t2.micro” instance will be provisioned. Upon deployment, the EC2 instance will create and upload a simple file to the S3 bucket you specified. Once you have saved the file, make sure you commit it to the local repository.

```
Admin:~/environment/sample-pipeline-artifacts (master) $ git status
…
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   stacker/stacker-config.yaml
```

Add the file to the staging area and commit

```
Admin:~/environment/sample-pipeline-artifacts (master) $ git add .
Admin:~/environment/sample-pipeline-artifacts (master) $ git commit -a -m 'updated s3 bucket name to be unique'
…
[master 939bf43] updated s3 bucket name to be unique
 1 file changed, 1 insertion(+), 1 deletion(-)
```

7. Provision the pipeline

Now it’s time to deploy the infrastructure pipeline using the sample code provided. Here we create two CloudFormation pipeline stacks as follows:

Pipeline stack #1: The first stack provisions the infrastructure pipeline we introduced earlier. The stack will provision a CodeCommit repo, a CodePipeline pipeline, and three CodeBuild projects.

The first CodeBuild project has a built-in buildspec configuration that will install and use CFN-Nag to check for security vulnerabilities in CloudFormation templates. The second CodeBuild project has a built-in buildspec configuration that will install and use CFN-Python-Lint to lint CloudFormation templates. The third CodeBuild project does not have a built-in buildspec configuration. Instead, a buildspec file will be pushed to the CodeCommit repo to instruct CodeBuild how it should install and run Stacker to provision and orchestrate CloudFormation stacks.

The buildspec file can either be built into the CloudFormation template or explicitly passed by developers. The advantage of building the buildspec file into the CloudFormation template is that it can help ensure that common standards are followed by teams who deploy code through a pipeline. (If you build the buildspec file into the CloudFormation template, the buildspec file cannot be edited or modified by developers.)

Before we go any further you will have to create an s3 bucket and upload the template. The s3 bucket should be in the following format mytemplates**YourAWSAccountID**:

```
Admin:~/environment/sample-pipeline-artifacts (master) $ aws s3 mb s3://mytemplates567893452376
…
make_bucket: mytemplates567893452376
```

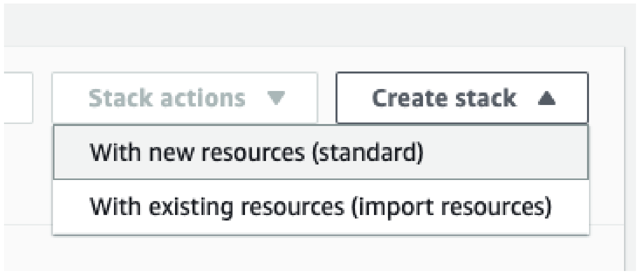
We then copy the codpipeline template codepipeline-template.yaml from the codepipeline folder into the s3 bucket

```
Admin:~/environment/sample-pipeline-artifacts (master) $ aws s3 cp codepipeline/codepipeline-template.yaml
s3://mytemplates567893452376/
```

8. Next we Provision the pipeline:

- Log in to your AWS Account
- Navigate to the CloudFormation console.
- Create a stack named **cfnsec-codepipeline-stack** that represents the infrastructure pipeline

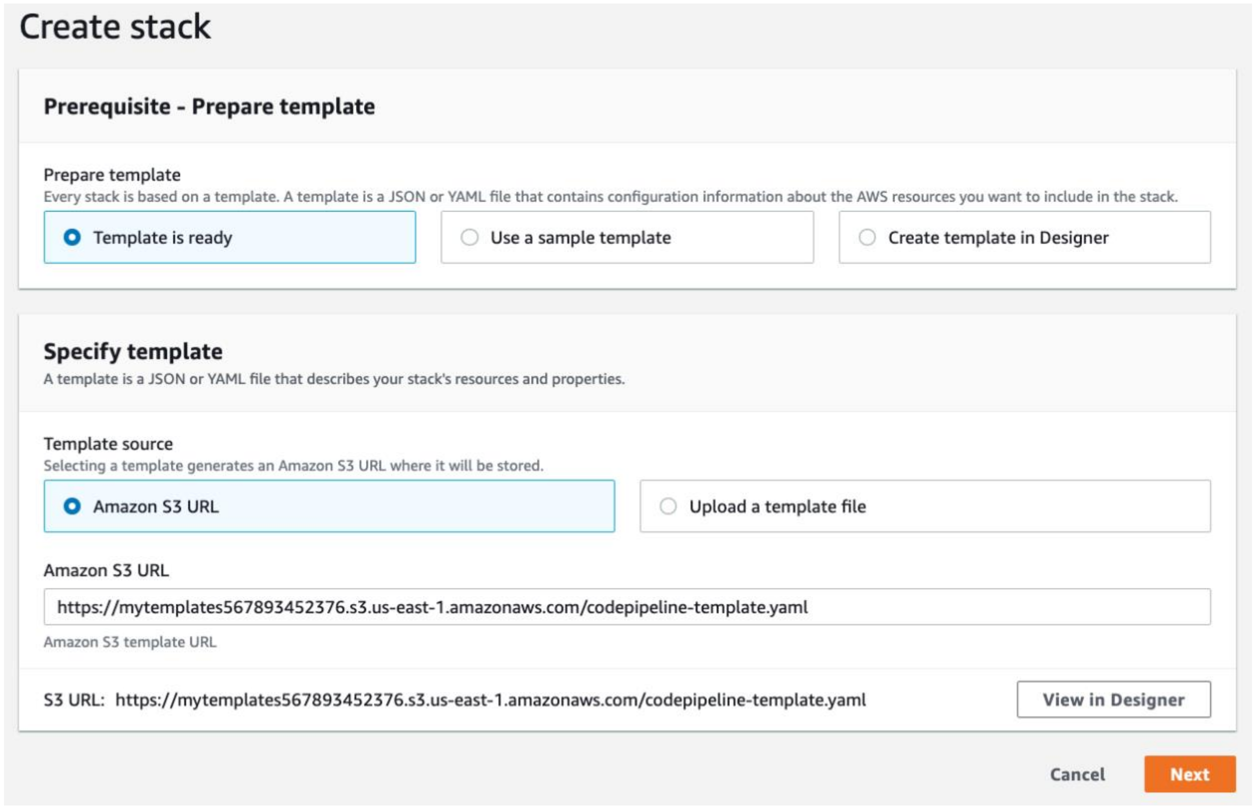
In the CloudFormation Stacks page, click the Create stack button with new resources:


Stack actions
Create stack
With new resources (standard)
With existing resources (import resources)

On the following page (Create stack), complete the following sections:

- Prerequisite – Prepare template: leave the default radio button selected, **Template is ready.**
- Specify template – select the radio button, **Amazon S3 URL** a template file. Add your created bucket as your s3 url:

  https://mytemplates***YourAmazonAccountId***.s3.us-east-1.amazonaws.com/codepipeline-template.yaml



Create stack

Prerequisite - Prepare template

Prepare template
Every stack is based on a template. A template is a JSON or YAML file that contains configuration information about the AWS resources you want to include in the stack.

[✓ Template is ready] Use a sample template / Create template in Designer

Specify template
A template is a JSON or YAML file that describes your stack's resources and properties.

Template source
Selecting a template generates an Amazon S3 URL where it will be stored.

[✓ Amazon S3 URL] Upload a template file

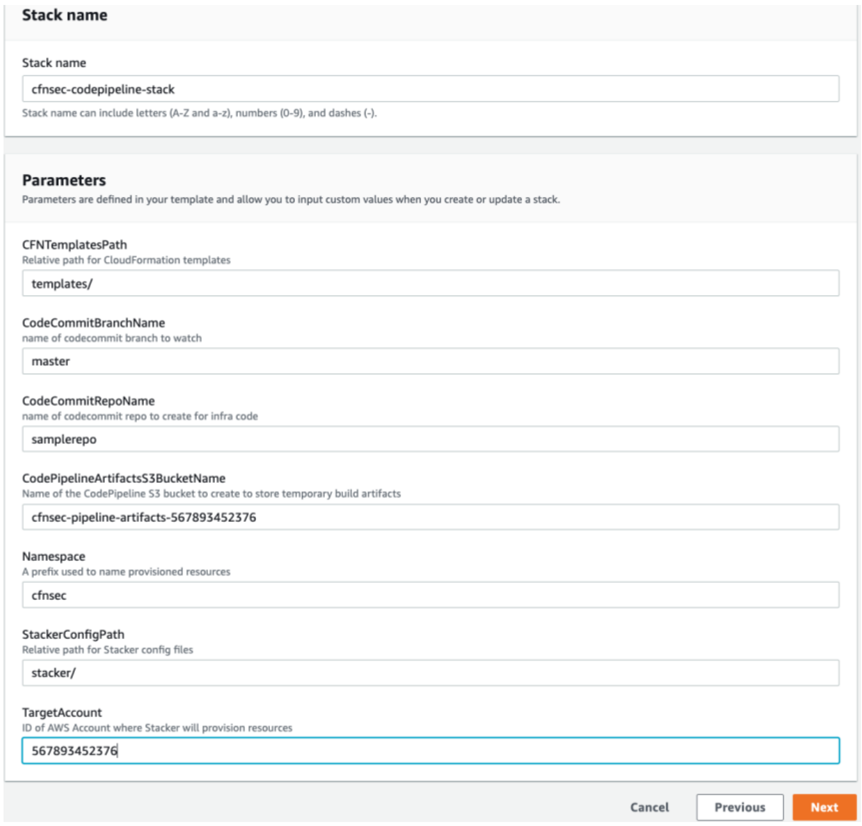
Amazon S3 URL
https://mytemplates567893452376.s3.us-east-1.amazonaws.com/codepipeline-template.yaml
Amazon S3 template URL

S3 URL: https://mytemplates567893452376.s3.us-east-1.amazonaws.com/codepipeline-template.yaml

- Click Next

On the page **Specify stack details,** complete the following sections:

- Stack name: enter the name **cfnsec-codepipeline-stack**.
- Parameters: leave the defaults or change as required (e.g., you can pick a name for you CodeCommit repository or accept the default "*samplerepo*"). Make sure to complete these two mandatory fields:
    - CodePipelineArtifactsS3BucketName: Specify a unique S3 bucket name that will be used by CodePipeline to store your build artifacts. We suggest you pick *cfnsec-pipeline-artifacts-XXXXXX* where X is a random number you choose or your AWS account id.
    - TargetAccount: Copy and paste your AWS account number in this field.


Stack name

Stack name
cfnsec-codepipeline-stack
Stack name can include letters (A-Z and a-z), numbers (0-9), and dashes (-).

Parameters
Parameters are defined in your template and allow you to input custom values when you create or update a stack.

CFNTemplatesPath
Relative path for CloudFormation templates
templates/

CodeCommitBranchName
name of codecommit branch to watch
master

CodeCommitRepoName
name of codecommit repo to create for infra code
samplerepo

CodePipelineArtifactsS3BucketName
Name of the CodePipeline S3 bucket to create to store temporary build artifacts
cfnsec-pipeline-artifacts-567893452376

Namespace
A prefix used to name provisioned resources
cfnsec

StackerConfigPath
Relative path for Stacker config files
stacker/

TargetAccount
ID of AWS Account where Stacker will provision resources
567893452376

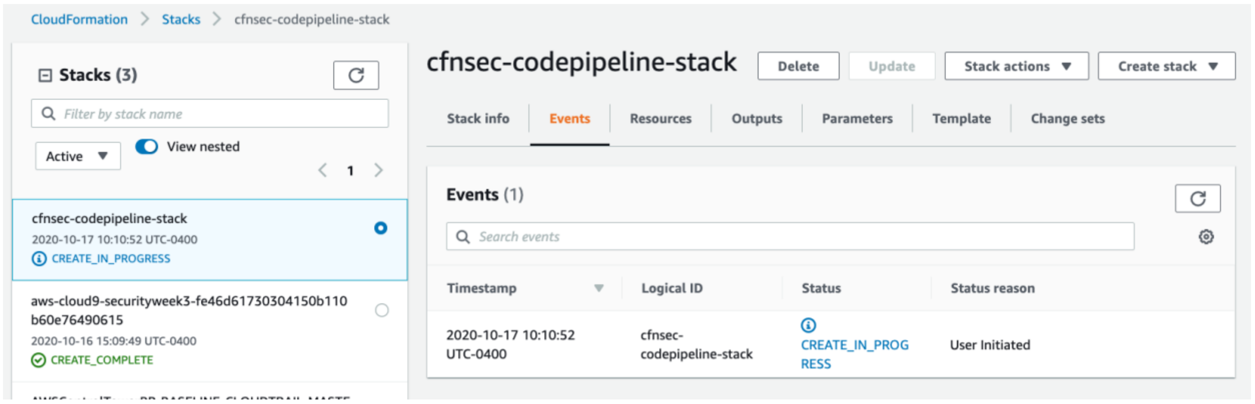
- Click the Next button.

On the page **Configure stack options**, leave the defaults in place and select Next.

On the page **Review**, take a moment to review the inputs:

- Make sure that you have the unique S3 bucket name and your correct account number entered or this project will not work.

- Once you are satisfied that all details have been correctly entered, select the check box **“I acknowledge** that AWS CloudFormation might create IAM resources with custom names.” This is required because the template will create IAM roles for the various CodeBuild projects as well as the CodePipeline pipeline.
- Select Create stack. It will take a few minutes for your first stack to be created. You should see something like this:



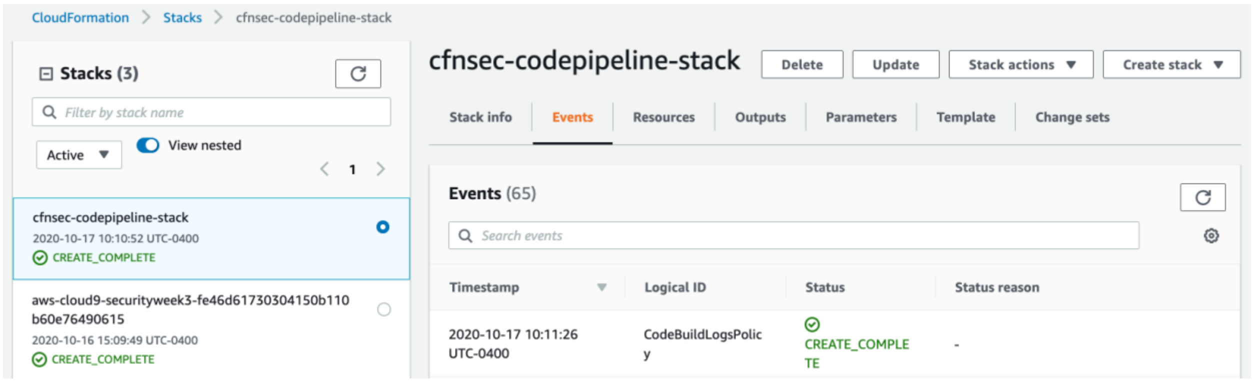
cfnsec-codepipeline-stack

Stack	
cfnsec-codepipeline-stack	2020-10-17 10:10:52 UTC-0400 CREATE_IN_PROGRESS
aws-cloud9-securityweek3-fe46d61730304150b110b60e76490615	2020-10-16 15:09:49 UTC-0400 CREATE_COMPLETE

Events (1)

Timestamp	Logical ID	Status	Status reason
2020-10-17 10:10:52 UTC-0400	cfnsec-codepipeline-stack	CREATE_IN_PROGRESS	User Initiated

Once your stack has been created successfully, you will see output similar to:



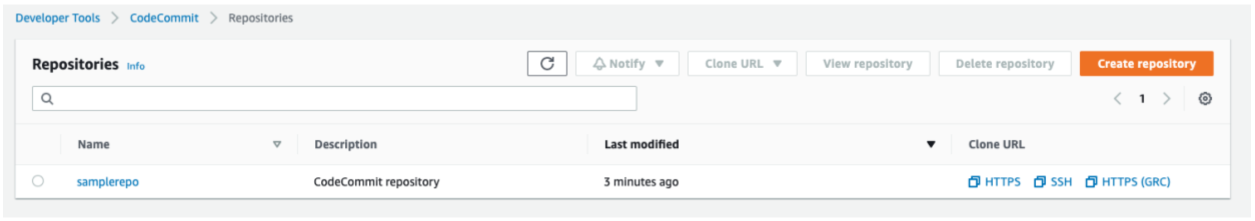
cfnsec-codepipeline-stack

Stack	
cfnsec-codepipeline-stack	2020-10-17 10:10:52 UTC-0400 CREATE_COMPLETE
aws-cloud9-securityweek3-fe46d61730304150b110b60e76490615	2020-10-16 15:09:49 UTC-0400 CREATE_COMPLETE

Events (65)

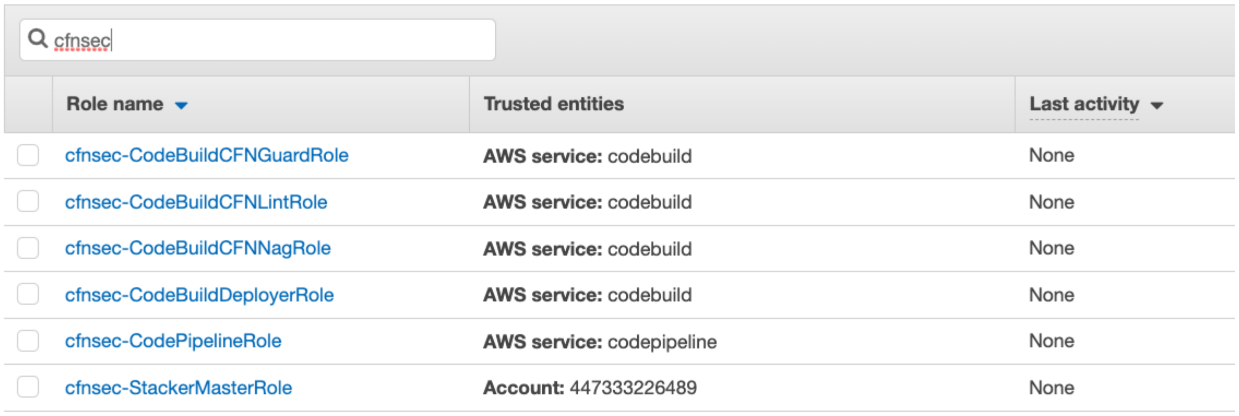
Timestamp	Logical ID	Status	Status reason
2020-10-17 10:11:26 UTC-0400	CodeBuildLogsPolicy	CREATE_COMPLETE	-

9. Before you create the second stack, navigate over to the CodeCommit Console. Select **Repositories** and you should see a sample repository created, like this (the name of the repository will be different if you have changed the default value):



Name	Description	Last modified	Clone URL
samplerepo	CodeCommit repository	3 minutes ago	HTTPS SSH HTTPS (GRC)

Next, navigate to the [IAM console]() and select **Roles**. You'll be able to see the five roles that your stack created:



cfnsec

	Role name	Trusted entities	Last activity
	cfnsec-CodeBuildCFNGuardRole	**AWS service:** codebuild	None
	cfnsec-CodeBuildCFNLintRole	**AWS service:** codebuild	None
	cfnsec-CodeBuildCFNNagRole	**AWS service:** codebuild	None
	cfnsec-CodeBuildDeployerRole	**AWS service:** codebuild	None
	cfnsec-CodePipelineRole	**AWS service:** codepipeline	None
	cfnsec-StackerMasterRole	**Account:** 447333226489	None

Let's look more closely at each of the roles and make sure that we understand the purpose of each role:

- **cfnsec-CodeBuildCFNLintRole:** This is an IAM service role created for the CFN-Python-Lint CodeBuild environment
- **cfnsec-CodeBuildCFNNagRole:** This is an IAM service role created for the CFN-Nag CodeBuild environment
- **cfnsec-CodeBuildCFNGuardRole:** This is an IAM service role created for the CFN-Guard CodeBuild environment
- **cfnsec-CodeBuildDeployerRole:** This is an IAM service role created for the Stacker CodeBuild environment
- **cfnsec-CodePipelineRole:** This in an IAM service role for CodePipeline to allow the pipeline to perform tasks such as read/write artifacts from/to the artifacts S3 bucket, to trigger the various CodeBuild environments
- **cfnsec-StackerMasterRole:** This is an IAM role used to launch Stacker, which allows Stacker to assume the cfnsec-StackerExecutionRole (to be created next) to deploy AWS resources via CloudFormation on the various target accounts. For simplicity, in our example we are using a single account, i.e., the same account where the pipeline resides.

The file **stacker/buildspec.yaml** contains the AWS CodeBuild buildspec to install and invoke Stacker for CloudFormation template provisioning. The **buildspec.yml** file defines the steps taken by AWS CodeBuild to test the AWS CloudFormation template prior to deploying it into a production environment. As mentioned before, there are two main ways in which you can specify your CodeBuild buildspec configuration. The first is to specify a buildspec inside the CloudFormation template (built-in buildspec) as we did for CFN-Nag and CFN-Python-Lint.

The code snippet below shows the CodeBuild project created via CloudFormation **codepipeline-template.yaml** for the CFN-Python-Lint CodePipeline action. Notice property `BuildSpec` below and how we used it to install (see `pip install cfn-lint`) and invoke the CFN-Python-Lint tool from within our CodeBuild Python 3.6.5 environment.

```
CFNLintCodeBuildProject:
    Type: AWS::CodeBuild::Project
    Properties:
        Name: !Sub ${Namespace}-cfn-lint-code-build-project
        Description: CodeBuild Project to validate CloudFormation templates using cnf-python-lint
        Artifacts:
          Type: CODEPIPELINE
        Environment:
            Type: LINUX_CONTAINER
            ComputeType: BUILD_GENERAL1_SMALL
            Image: aws/codebuild/python:3.6.5
            EnvironmentVariables:
              - Name: CFNTemplatesPath
                Value: !Ref CFNTemplatesPath
        ServiceRole:
          !GetAtt CFNNagCodeBuildServiceRole.Arn
        Source:
            Type: CODEPIPELINE
            BuildSpec: |
              version: 0.2
              phases:
                install:
                  commands:
                    - pip install --upgrade pip
                    - env && ls -l && python --version
                    - pip install cfn-lint
                    - cfn-lint ${CFNTemplatesPath}*.yaml

  CFNPNagCodeBuildProject:
    Type: AWS::CodeBuild::Project
    Properties:
        Name: !Sub ${Namespace}-cfn-nag-code-build-project
        Description: CodeBuild Project to validate CloudFormation templates using CFN-Nag
        Artifacts:
          Type: CODEPIPELINE
        Environment:
            Type: LINUX_CONTAINER
            ComputeType: BUILD_GENERAL1_SMALL
            # With the image below we must specify a runtime-version in the Buildspec (see below)
            Image: aws/codebuild/amazonlinux2-x86_64-standard:1.0
            EnvironmentVariables:
              - Name: CFNTemplatesPath
                Value: !Ref CFNTemplatesPath
        ServiceRole:
          !GetAtt CFNLintCodeBuildServiceRole.Arn
        Source:
            Type: CODEPIPELINE
            BuildSpec: |
              version: 0.2
              phases:
                install:
                  runtime-versions:
```

```
              ruby: 2.6
            commands:
              - env && ls -l && ruby -v
              - gem install cfn-nag
              - cfn_nag_scan -v
              - cfn_nag_scan --input-path $CFNTemplatesPath

  CFNGuardCodeBuildProject:
    Type: AWS::CodeBuild::Project
    Properties:
      Name: !Sub ${Namespace}-cfn-guard-code-build-project
      Description: CodeBuild Project to validate CloudFormation templates using cnf-guard
      Artifacts:
        Type: CODEPIPELINE
      Environment:
        Type: LINUX_CONTAINER
        ComputeType: BUILD_GENERAL1_SMALL
        # With the image below we must specify a runtime-version in the Buildspec (see below)
        Image: aws/codebuild/amazonlinux2-x86_64-standard:1.0
        EnvironmentVariables:
          - Name: CFNTemplatesPath
            Value: !Ref CFNTemplatesPath
      ServiceRole:
        !GetAtt CFNGuardCodeBuildServiceRole.Arn
      Source:
        Type: CODEPIPELINE
        BuildSpec: |
          version: 0.2
          phases:
            install:
              runtime-versions:
                ruby: 2.6
              commands:
                - rm -rf /var/cache/yum
                - yum clean all && yum makecache fast
                - yum -y install rust cargo
                - rpm -ivh $CFNTemplatesPath/cfn-guard-1.0-0.amzn2.x86_64.rpm
                - which cfn-guard
                - cfn-guard check --rule_set $CFNTemplatesPath/test.rules --template ${CFNTemplatesPath}/ec2-role-
template.yaml
                - cfn-guard check --rule_set $CFNTemplatesPath/test.rules --template ${CFNTemplatesPath}/ec2-
template.yaml
                - cfn-guard check --rule_set $CFNTemplatesPath/test.rules --template ${CFNTemplatesPath}/s3-bucket-
template.yaml
```

Using the `BuildSpec` property in CloudFormation means that developers pushing code to the pipeline cannot modify the behavior of the CodeBuild environment and how we use CFN-Python-Lint to lint Cloudformation templates.

Another way to configure a CodeBuild environment is to allow developers to create a custom **buildspec.yml** file and push it to the CodeCommit repository. This gives them more flexibility and control over the CodeBuild execution environment so they, for instance, install and run software as needed. In cases where different teams own the pipeline vs. the pipeline artifacts the pipeline builder team should plan carefully what level of customization they want to offer developers vs. what build steps are immutable.

Below is the custom **buildspec.yml** YAML file we use for the CodeBuild project that runs Stacker. Notice that we define some environment variables and run a few commands to configure AWS credentials and to install (`pip install stacker==1.7.0`) and run Stacker (`stacker build`).

```
version: 0.2

env:
  variables:
    stacker_master_profile_name: "stacker_master"
    stacker_profiles_file: "stacker-profiles"
    stacker_orchestration_file: "stacker-config.yaml"

phases:

  pre_build:
    commands:
      - pip install --upgrade pip
      - pip install stacker==1.7.0
      - env && ls -lha && python --version

  build:
    commands:
      - export AWS_CONFIG_FILE="${CODEBUILD_SRC_DIR}/${StackerConfigPath}/${stacker_profiles_file}"
      - echo "AWS_CONFIG_FILE=${AWS_CONFIG_FILE}"
      - stacker build "${CODEBUILD_SRC_DIR}/${StackerConfigPath}/${stacker_orchestration_file}" --profile
$stacker_master_profile_name --recreate-failed
```

Pipeline stack #2: The second stack we're going to create provisions the IAM role required by Stacker to deploy AWS resources in a given target AWS account. We use separate templates for the pipeline and the Stacker execution role, because the latter can now be deployed to multiple target accounts if we wish the pipeline to provision stacks into those accounts.

Since we're using a single AWS account in this post, we will create the Stacker execution role in the same account as the pipeline. The IAM policies in the role must set permissions to deploy all types of AWS resources referenced in the template. In our case, Stacker needs permissions to create an S3 bucket, an EC2 instance, IAM roles, etc. Please check the template for further details.

10. Create your second stack the same way you created the first one, named **cfnsec-stacker-execution-role-stack**, using the file **codepipeline/stacker-execution-role-template.yaml**. This second stack will create an IAM role in your account so that Stacker can deploy outputs in the same account where it lives. As stated above, while we are using a single account for this project, this approach will also work for multiple accounts.

    Upload the file to your s3 bucket:

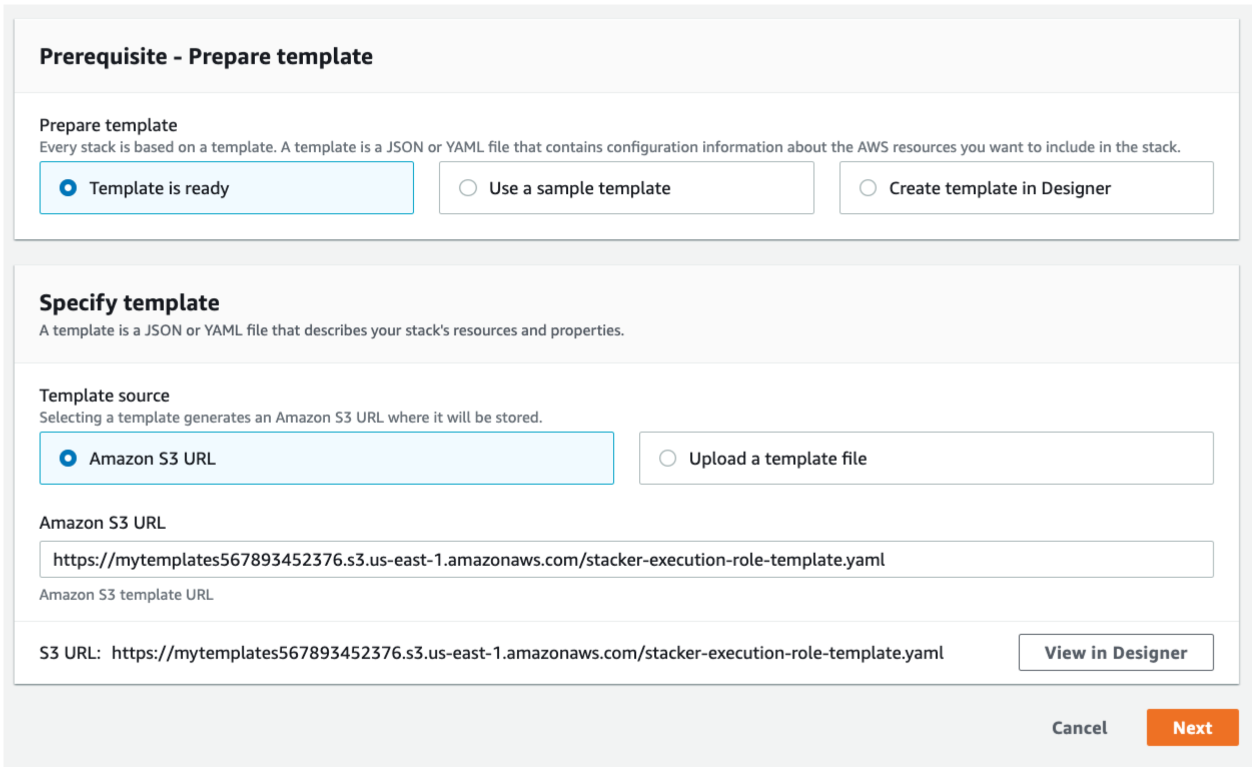
```
Admin:~/environment/sample-pipeline-artifacts (master) $ aws s3 cp codepipeline/stacker-execution-role-template.yaml s3://mytemplates567893452376/
```

On the **CloudFormation** Stacks page, select Create stack.

- On the following page (Create stack), complete the following sections:
- Prerequisite – Prepare template: leave the default radio button selected, **Template is ready**.
- Specify template – select the radio button, **Amazon S3 URL** a template file. Add your created bucket as your s3 url:

https://mytemplates***YourAmazonAccountId***.s3.us-east-1.amazonaws.com/stacker-execution-role-template.yaml

- Click Next.



Prerequisite - Prepare template

Prepare template
Every stack is based on a template. A template is a JSON or YAML file that contains configuration information about the AWS resources you want to include in the stack.

[✓ Template is ready] Use a sample template Create template in Designer

Specify template
A template is a JSON or YAML file that describes your stack's resources and properties.

Template source
Selecting a template generates an Amazon S3 URL where it will be stored.

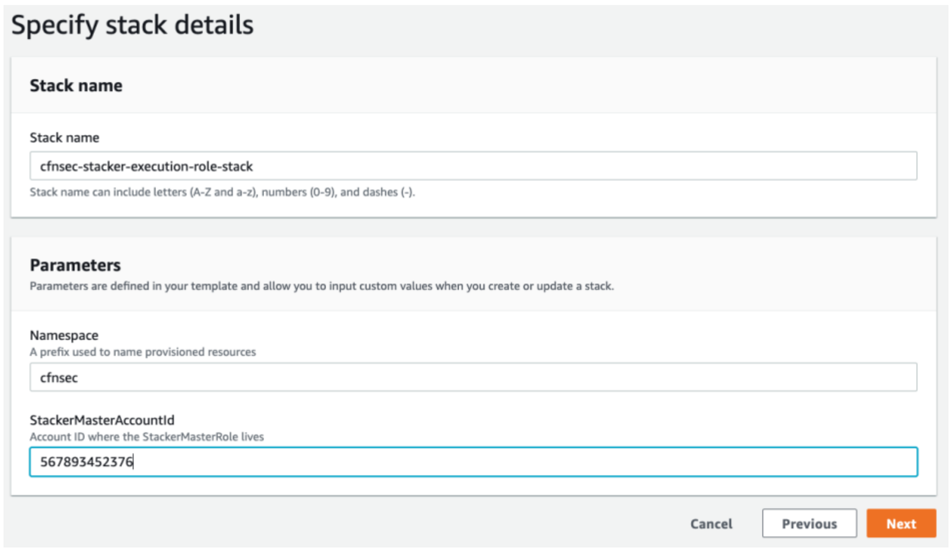
[✓ Amazon S3 URL] Upload a template file

Amazon S3 URL
https://mytemplates567893452376.s3.us-east-1.amazonaws.com/stacker-execution-role-template.yaml
Amazon S3 template URL

S3 URL: https://mytemplates567893452376.s3.us-east-1.amazonaws.com/stacker-execution-role-template.yaml

On the page **Specify stack details**, complete the following sections:

- Stack name: enter the name **cfnsec-stacker-execution-role-stack**
- Parameters: leave the defaults. However, make sure to complete these two fields:
    - Namespace: leave the default value, cfnsec, as is.
    - StackMasterAccountID: Copy and paste your AWS account number in this field.



Specify stack details

Stack name

Stack name: cfnsec-stacker-execution-role-stack

Stack name can include letters (A-Z and a-z), numbers (0-9), and dashes (-).

Parameters

Parameters are defined in your template and allow you to input custom values when you create or update a stack.

Namespace
A prefix used to name provisioned resources
cfnsec

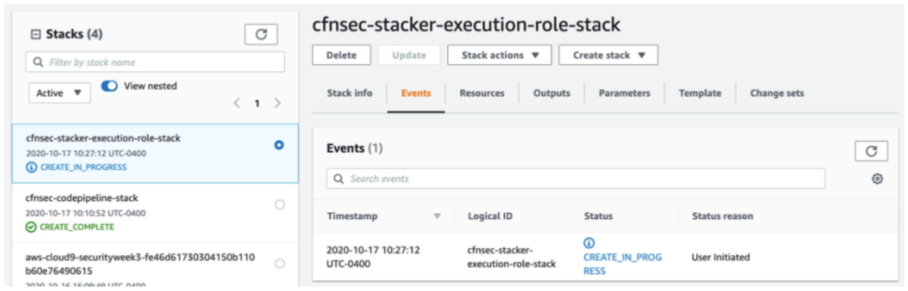
StackerMasterAccountId
Account ID where the StackerMasterRole lives
567893452376

- Select Next

On the **Configure stack options** page, leave the defaults in place and select Next

On the **Review page**, take a moment to review the inputs.

- Make sure that you have the unique S3 bucket name and your correct account number entered or this project will not work.
- At the bottom of the page, click the tick box "**I acknowledge** that AWS CloudFormation might create IAM resources with custom names."
- Click Create Stack. It will take a minute or two for your stack to be created. You should see something like this:



cfnsec-stacker-execution-role-stack
2020-10-17 10:27:12 UTC-0400
CREATE_IN_PROGRESS

cfnsec-codepipeline-stack
2020-10-17 10:10:52 UTC-0400
CREATE_COMPLETE

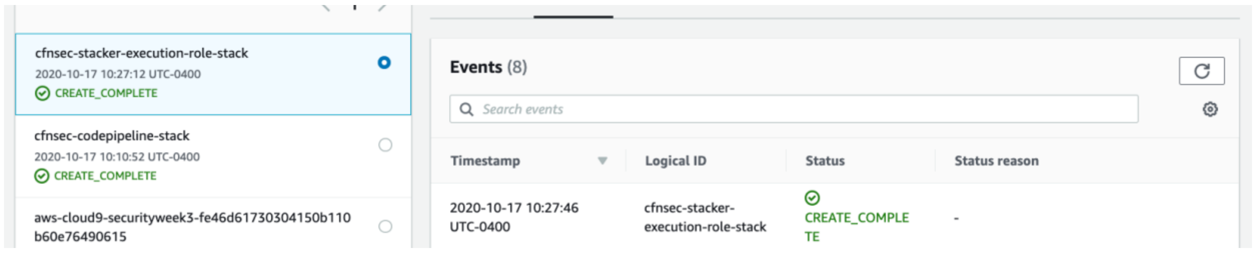
aws-cloud9-securityweek3-fe46d61730304150b110b60e76490615

cfnsec-stacker-execution-role-stack

Events (1)

Timestamp	Logical ID	Status	Status reason
2020-10-17 10:27:12 UTC-0400	cfnsec-stacker-execution-role-stack	CREATE_IN_PROGRESS	User Initiated

When your stack has been successfully built, the console screen will look like this:



cfnsec-stacker-execution-role-stack
2020-10-17 10:27:12 UTC-0400
CREATE_COMPLETE

cfnsec-codepipeline-stack
2020-10-17 10:10:52 UTC-0400
CREATE_COMPLETE

aws-cloud9-securityweek3-fe46d61730304150b110b60e76490615

Events (8)

Timestamp	Logical ID	Status	Status reason
2020-10-17 10:27:46 UTC-0400	cfnsec-stacker-execution-role-stack	CREATE_COMPLETE	-

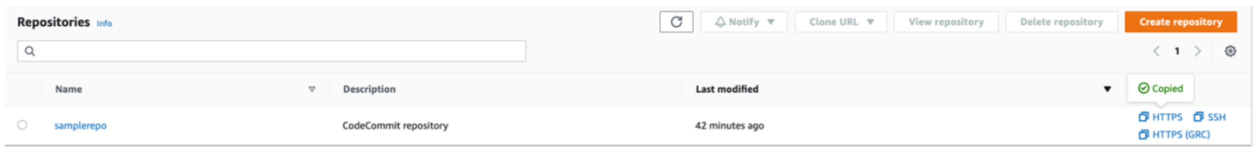
11. Using the AWS CodePipeline infrastructure pipeline

Now that the pipeline is ready and we have configured the pipeline artifacts, it's time to use the pipeline to deploy our stacks.

Prerequisites

Next we will setup our local repository sample-pipeline-artifacts to upload all the files to our sample repository. From within your AWS Account, navigate to the CodeCommit console and look for the repo named samplerepo.

- Under the heading Clone URL, click the blue text HTTPS. This action will copy the HTTPS address to your clipboard.



Name	Description	Last modified
samplerepo	CodeCommit repository	42 minutes ago

At this point, you should have a local git repository with the **stacker/** and **templates/** folders (created earlier). Add a Git remote to your CodeCommit repository like this:

```
Admin:~/environment/sample-pipeline-artifacts (master) $ git remote add origin https://git-codecommit.us-east-
1.amazonaws.com/v1/repos/samplerepo
```

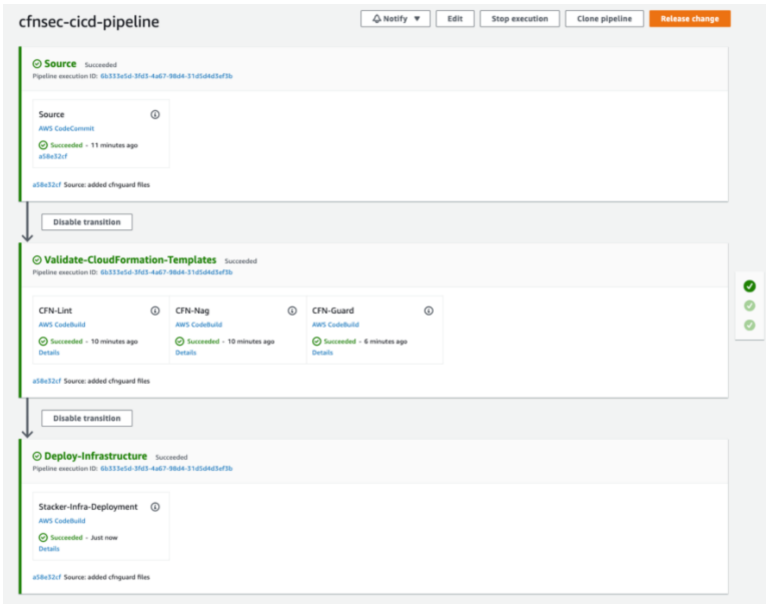
Next we need to add two more files to our templates folder so that we can run cfn-guard

```
Admin:~/environment/sample-pipeline-artifacts (master) $ cp cfnguard/test.rules templates/
Admin:~/environment/sample-pipeline-artifacts (master) $ cp rpms/cfn-guard-1.0-0.amzn2.x86_64.rpm templates/
Admin:~/environment/sample-pipeline-artifacts (master) $ git add .
Admin:~/environment/sample-pipeline-artifacts (master) $ git status
…
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   templates/cfn-guard-1.0-0.amzn2.x86_64.rpm
        new file:   templates/test.rules

Admin:~/environment/sample-pipeline-artifacts (master) $ git commit -a -m 'added cfnguard files'
[master a58e32c] added cfnguard files
 2 files changed, 9 insertions(+)
 create mode 100644 templates/cfn-guard-1.0-0.amzn2.x86_64.rpm
 create mode 100644 templates/test.rules
```

Make sure your local Git repo does not have any pending changes to be committed (git status). Push your code to the CodeCommit repository's master branch:

```
Admin:~/environment/sample-pipeline-artifacts (master) $ git push origin master:master
```

After pushing your code, navigate to the AWS CodePipeline console to verify that the pipeline was triggered successfully. Once in the CodePipeline console, select **Pipelines** then your pipeline **cfnsec-cicd-pipeline**. If all goes well, your templates will be validated and deployed and the pipeline should look like the screenshot below:



cfnsec-cicd-pipeline

Source Succeeded
Pipeline execution ID: 6b333e5d-3fd3-4a67-98d4-31d5d4d3ef3b

Source
AWS CodeCommit
Succeeded - 11 minutes ago
a58e32cf

a58e32cf Source: added cfnguard files

Validate-CloudFormation-Templates Succeeded
Pipeline execution ID: 6b333e5d-3fd3-4a67-98d4-31d5d4d3ef3b

CFN-Lint
AWS CodeBuild
Succeeded - 10 minutes ago
Details

CFN-Nag
AWS CodeBuild
Succeeded - 10 minutes ago
Details

CFN-Guard
AWS CodeBuild
Succeeded - 6 minutes ago
Details

a58e32cf Source: added cfnguard files

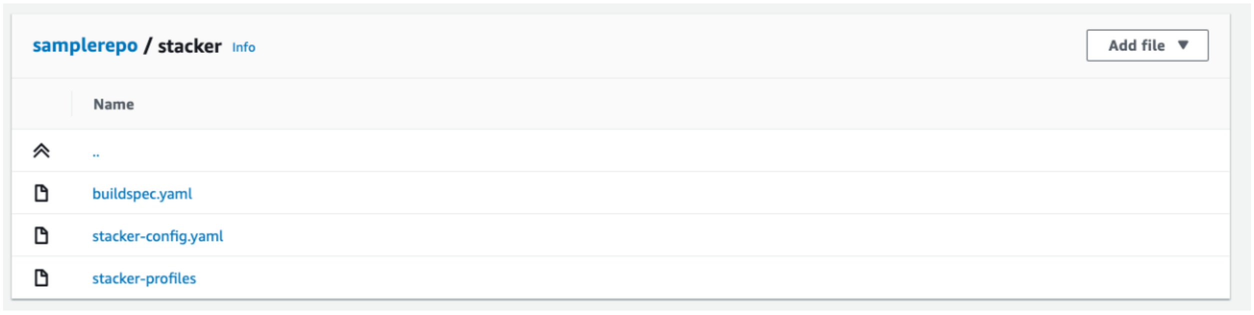
Deploy-Infrastructure Succeeded
Pipeline execution ID: 6b333e5d-3fd3-4a67-98d4-31d5d4d3ef3b

Stacker-Infra-Deployment
AWS CodeBuild
Succeeded - Just now
Details

a58e32cf Source: added cfnguard files

12. So what have we deployed exactly? Well, the three stacks we mentioned previously: an S3 bucket, an IAM role, and and EC2 instance (plus other resources, e.g., security group, EC2 profile) that assumes the IAM role and writes a file to the S3 bucket. You should have a file named 'hello.txt' in the S3 bucket your specified in **stacker/stacker-config.yaml.**

Have a look now at your sample repo in the AWS CodeCommit Console, to review what you pushed through your serverless pipeline:



samplerepo / stacker

Name
..
buildspec.yaml
stacker-config.yaml
stacker-profiles

Earlier in this post we introduced the file structure and explained the files that will be pushed out to the codepipeline and their purpose. Let's briefly recap:

- **buildspec.yaml**: This is the AWS CodeBuild buildspec. As discussed, its purpose is to install and invoke Stacker for CloudFormation resource provisioning
- **stacker-config.yaml**: This is the stacker config file containing stack descriptions and input parameters
- **stacker-profiles**: This file contains the AWS profiles that Stacker used to assume roles during deployment

13. Testing pipeline failure scenarios

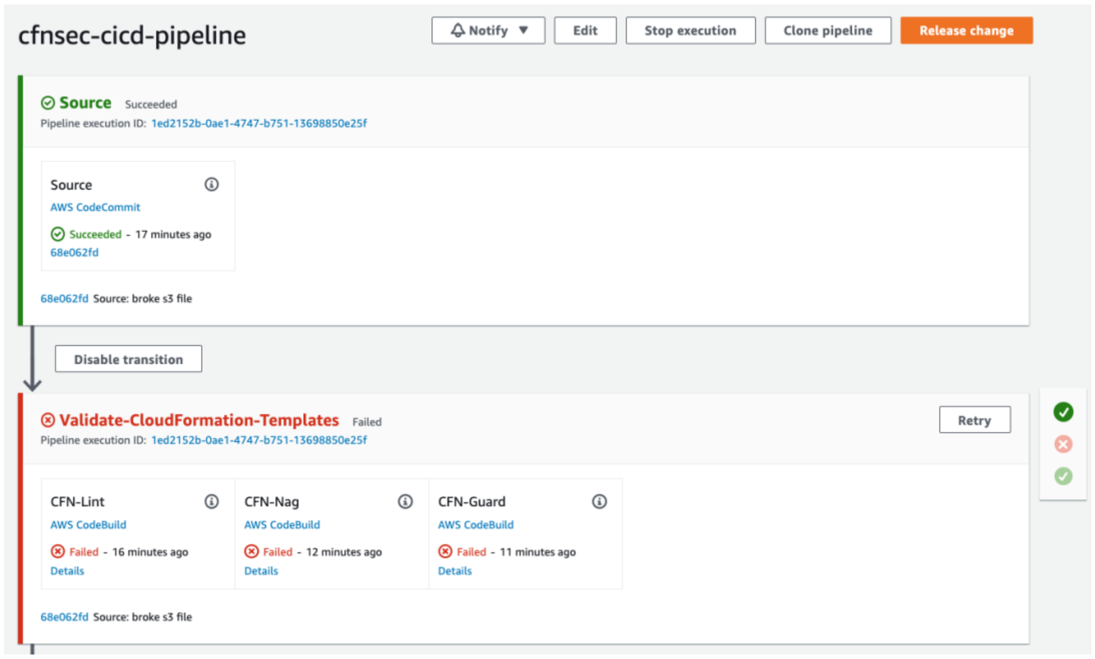
Let’s make a change to one of the templates under `templates/` to cause the pipeline template verification stage to fail. For example, specify an invalid type for the `S3BucketName` input parameter in the template `s3-bucket-template.yaml`, like this:

```
AWSTemplateFormatVersion: "2010-09-09"
Description: Creates an encrypted non-public S3 bucket to store CI/CD artifacts

Parameters:

  S3BucketName:
    Type: *InvalidType*
    Description: Unique name to assign to S3 bucket

Resources:

  Bucket:
    Type: AWS::S3::Bucket
    DeletionPolicy: Delete
```

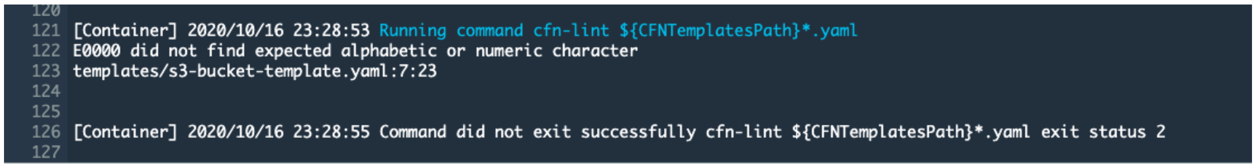
Commit and push your code to the CodeCommit repository again which will trigger the pipeline.

```
Admin:~/environment/sample-pipeline-artifacts (master) $ git add .
Admin:~/environment/sample-pipeline-artifacts/templates (master) $ git commit s3-bucket-template.yaml -m 'broke
s3 file'
…
[master 68e062f] broke s3 file
 1 file changed, 1 insertion(+), 1 deletion(-)
Admin:~/environment/sample-pipeline-artifacts/templates (master) $ git push origin master:master
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 2 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 363 bytes | 363.00 KiB/s, done.
Total 4 (delta 3), reused 0 (delta 0)
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/samplerepo
   a58e32c..68e062f  master -> master
```

Wait for the pipeline to fail as CFN-Python-Lint checks your template against CloudFormation specs, as in the screenshot below:



For details on why the pipeline failed, click on the **Details** link in the CodeBuild action for CFN- -Lint. You should see a screen similar to the one below.



```
[Container] 2020/10/16 23:28:53 Running command cfn-lint ${CFNTemplatesPath}*.yaml
E0000 did not find expected alphabetic or numeric character
templates/s3-bucket-template.yaml:7:23


[Container] 2020/10/16 23:28:55 Command did not exit successfully cfn-lint ${CFNTemplatesPath}*.yaml exit status 2
```

That's exactly what we wanted: CFN-Lint is helping us make sure that our templates are valid before letting the pipeline deploy them.

Let's revert the last commit again. Go back to your s3 file and replace the *invalidType* with String. Then run the commands below which will trigger the pipeline again and should provide a successful build.

```
Admin:~/environment/sample-pipeline-artifacts/templates (master) $ git add s3-bucket-template.yaml
Admin:~/environment/sample-pipeline-artifacts/templates (master) $ git commit s3-bucket-template.yaml -m 'reverted breaking the s3 bucket'
Admin:~/environment/sample-pipeline-artifacts/templates (master) $ git push origin master:master
…
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 2 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 367 bytes | 367.00 KiB/s, done.
Total 4 (delta 3), reused 0 (delta 0)
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/samplerepo
   47a3514..8a88bd4  master -> master
```

14. CFN-guard build failure

We will now look at what cnf-guard. Below is the test.rules files from the cfnguard folder:

```
AWS::EC2::Instance IamInstanceProfile == ECInstanceProfile
AWS::EC2::Instance ImageId == LatestAmiId
AWS::EC2::Instance InstanceType == /.nano|.micro|.small/
AWS::EC2::Instance SecurityGroups == ["InstanceSecurityGroup"]
AWS::EC2::Instance UserData == {"Fn::Base64":"#!/bin/bash -xe\nexport INSTANCE_ID=$(curl -s http://169.254.169.254/latest/meta-
data/instance-id)\necho \"Hello from EC2 instance $INSTANCE_ID\" > hello.txt\naws s3 cp hello.txt s3://${S3BucketName}\n"}
AWS::EC2::SecurityGroup GroupDescription == Enable SSH access via port 22
AWS::EC2::SecurityGroup SecurityGroupIngress == [{"CidrIp":"10.15.45.31/32","FromPort":22,"IpProtocol":"tcp","ToPort":22}]
AWS::IAM::InstanceProfile Path == /
AWS::IAM::InstanceProfile Roles == ["EC2IamRoleName"]
```

This creates guardrails which define that an InstanceType can only be nano, micro or small. You can define for example what your AMI ID should be as well as define specific security groups and much more. For example if you have a security standard defined, you can model the cfn-guard rules after it. If a developer chooses to use a larger instance then defined, cfn-guard will fail the build.

Let run through the last example of this lab.

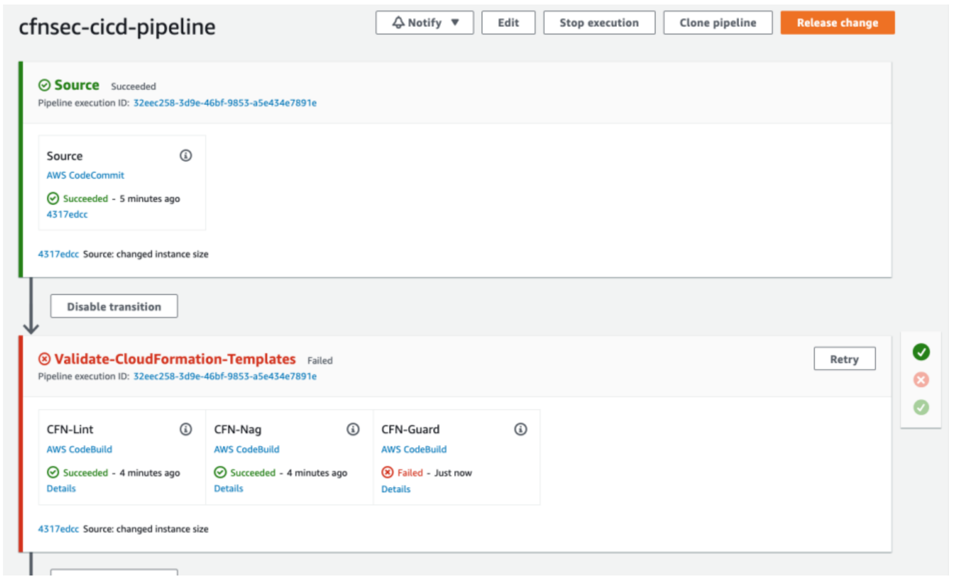
We will alter the following file ec2-template.yml which is in the templates folder. We will change the InstanceType to m5.2xlarge.

```
 EC2Instance:
   Type: AWS::EC2::Instance
   Properties:
     InstanceType: m5.2xlarge
     SecurityGroups: [!Ref 'InstanceSecurityGroup']
     ImageId: !Ref 'LatestAmiId'
     IamInstanceProfile: !Ref ECInstanceProfile
```

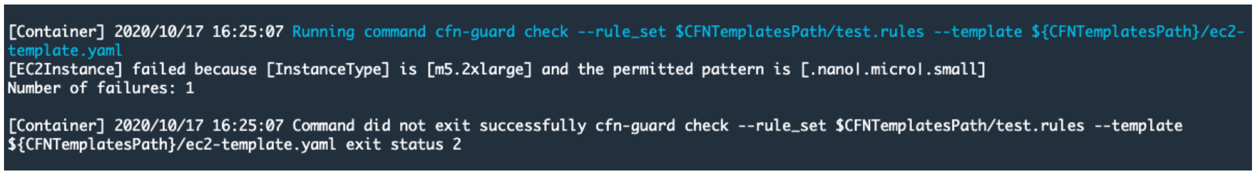
Add the file to your staging area, commit and push the file to the repository.

```
Admin:~/environment/sample-pipeline-artifacts/templates (master) $ git add .
Admin:~/environment/sample-pipeline-artifacts/templates (master) $ git commit ec2-template.yaml -m 'changed instance size'
…
[master 4317edc] changed instance size
 1 file changed, 1 insertion(+), 1 deletion(-)
git push origin master:master
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 2 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 370 bytes | 370.00 KiB/s, done.
Total 4 (delta 3), reused 0 (delta 0)
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/samplerepo
   8a88bd4..4317edc  master -> master
```

This will trigger another build. You will see that this test is passing cfn-lint as well as cfn-nag as our template is syntactically correct and we are not violating the rules of cfn-nag but we will fail cfn-guard as is does not meet our specifications.


cfnsec-cicd-pipeline
Notify
Edit
Stop execution
Clone pipeline
Release change
Source Succeeded
Pipeline execution ID: 32eec258-3d9e-46bf-9853-a5e434e7891e
Source
AWS CodeCommit
Succeeded - 5 minutes ago
4317edcc
4317edcc Source: changed instance size
Disable transition
Validate-CloudFormation-Templates Failed
Retry
Pipeline execution ID: 32eec258-3d9e-46bf-9853-a5e434e7891e
CFN-Lint
AWS CodeBuild
Succeeded - 4 minutes ago
Details
CFN-Nag
AWS CodeBuild
Succeeded - 4 minutes ago
Details
CFN-Guard
AWS CodeBuild
Failed - Just now
Details
4317edcc Source: changed instance size

If you investigate the build you will see that cfn-guard caught our change in terms of a not permitted instance size.



```
[Container] 2020/10/17 16:25:07 Running command cfn-guard check --rule_set $CFNTemplatesPath/test.rules --template ${CFNTemplatesPath}/ec2-template.yaml
[EC2Instance] failed because [InstanceType] is [m5.2xlarge] and the permitted pattern is [.nano|.micro|.small]
Number of failures: 1

[Container] 2020/10/17 16:25:07 Command did not exit successfully cfn-guard check --rule_set $CFNTemplatesPath/test.rules --template ${CFNTemplatesPath}/ec2-template.yaml exit status 2
```

Conclusion

In this lab, we showed you how to leverage popular open source tools in conjunction with AWS Code services such as CodePipeline, CodeCommit, and CodeBuild to build, validate and deploy arbitrary infrastructure stacks. We used CFN-NAG, CFN-Lint and CFN-guard to validate and guard the templates, and Stacker to perform the deployment of the CloudFormation stacks.