AWS CloudFormation notes

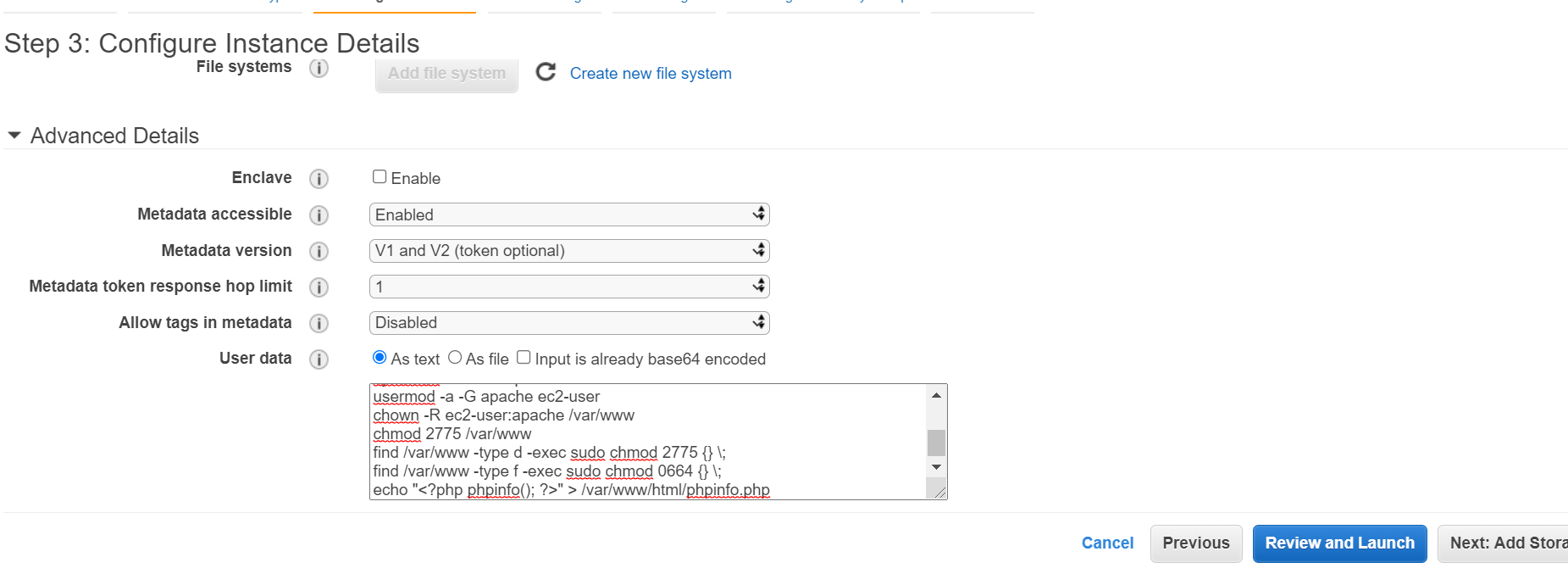
**EC2 User data:**

Ec2 user data is set of commands we mentioned while provisioning the EC2 instance so that while instance get crated this script will auto run so that what we want to configure it get configured automatically right after the EC2 up and running.

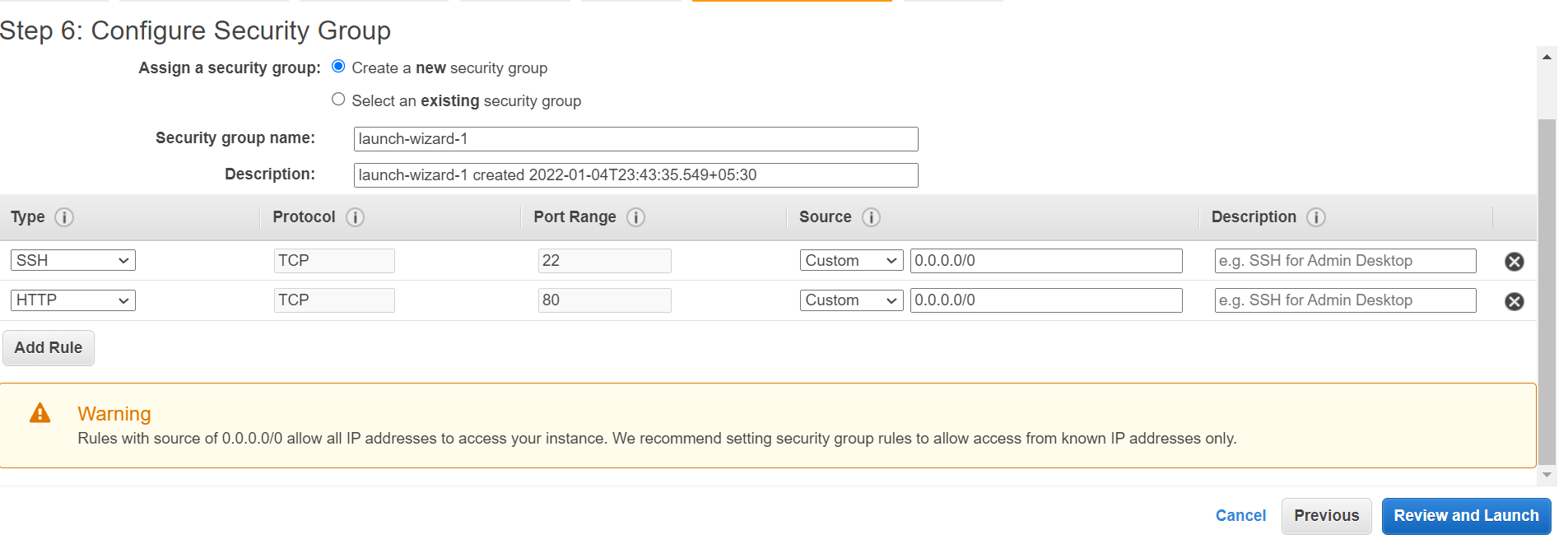
Task : Create a EC2 and Setup a Web server (Using User data)

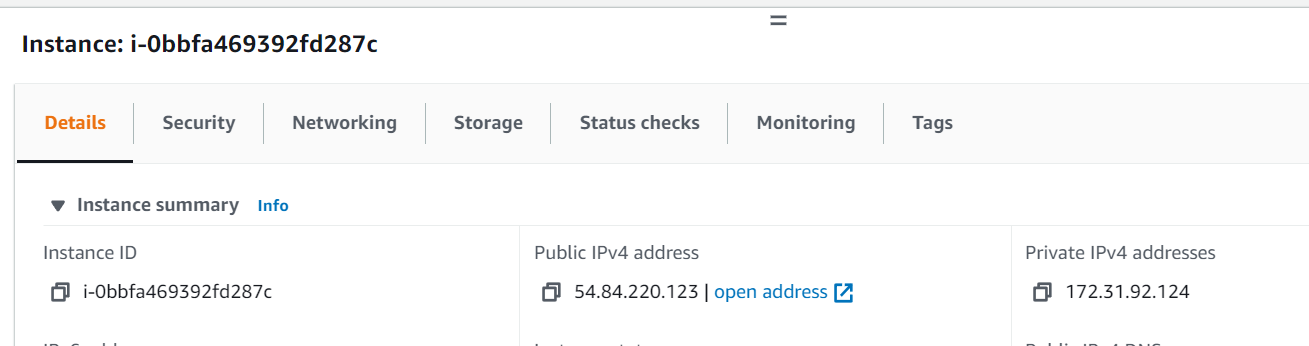
Here we are going to create an EC2 and configure it as a web server using EC2 user data approach manually

Create an EC2 from Console use 0-ec2-user-data.sh userdata

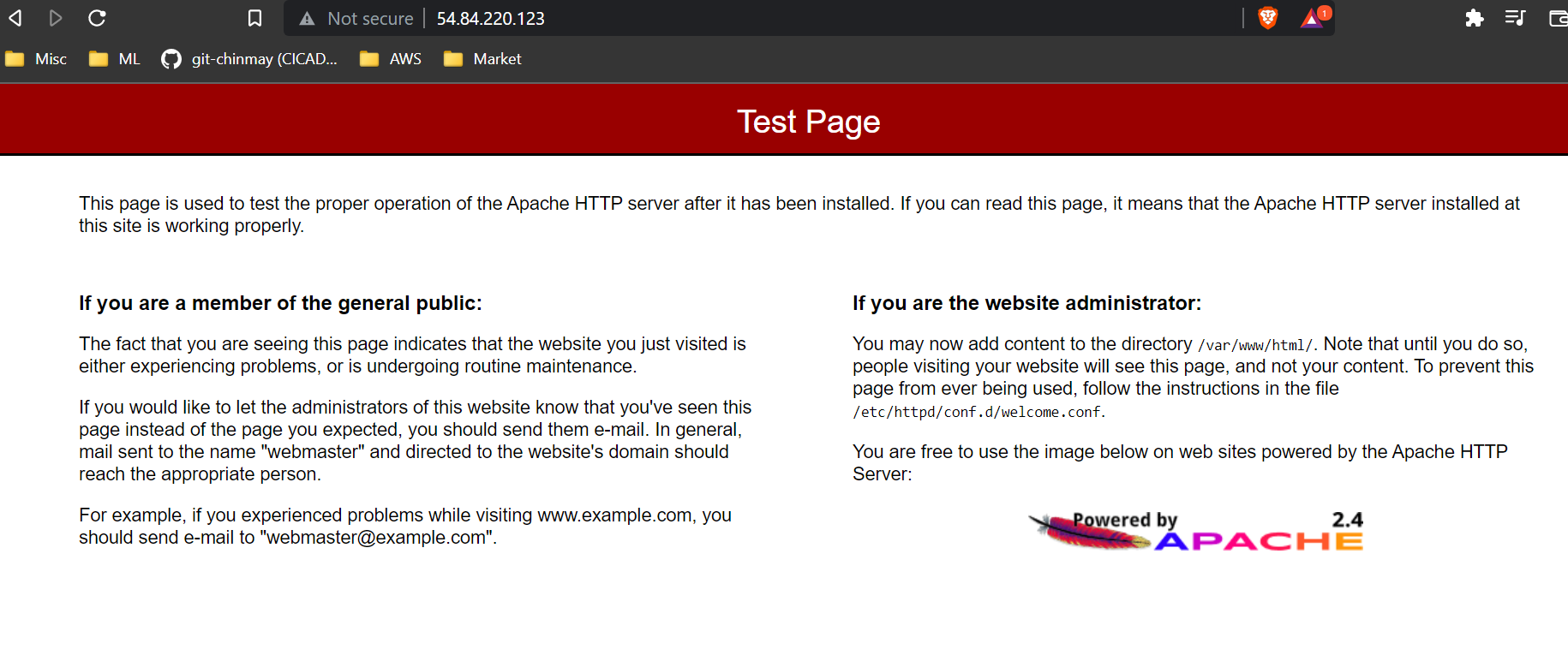


Configure security

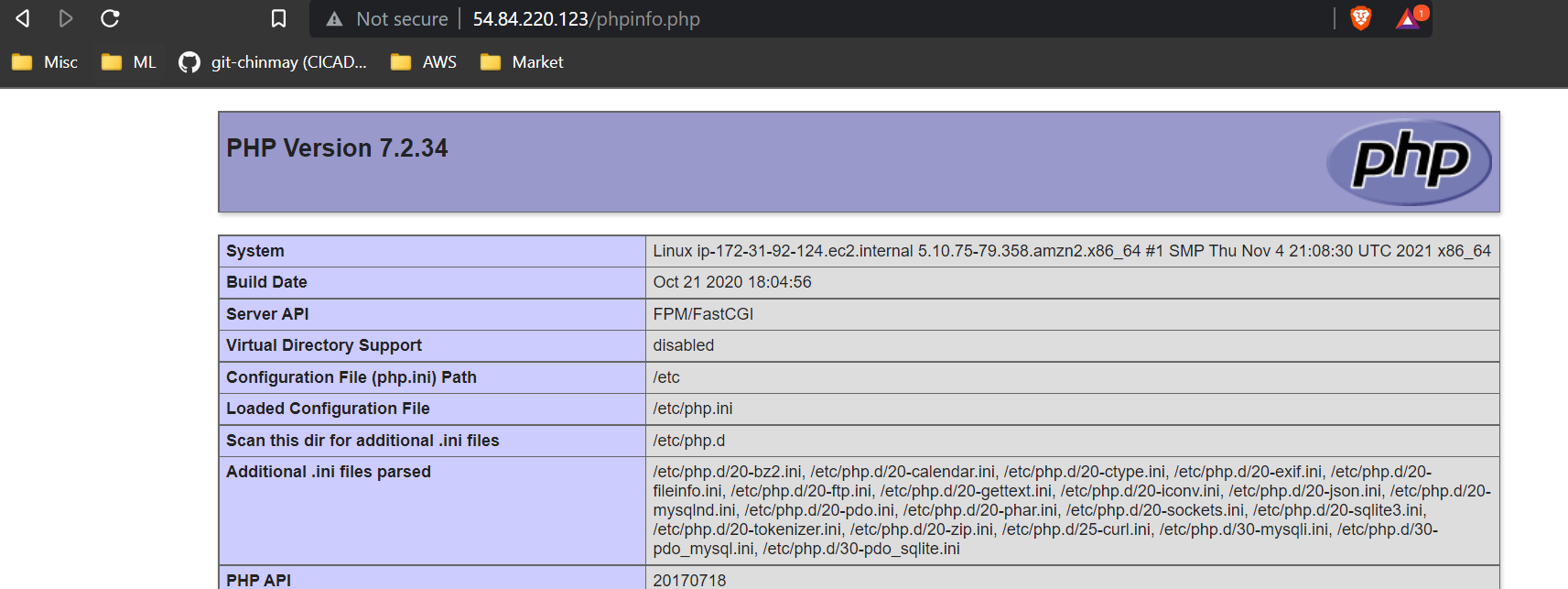
Open the public IP in browser. It will take time to get configured so have some patients (Copy the IP, don’t use the open address option)



Webpage is ready.

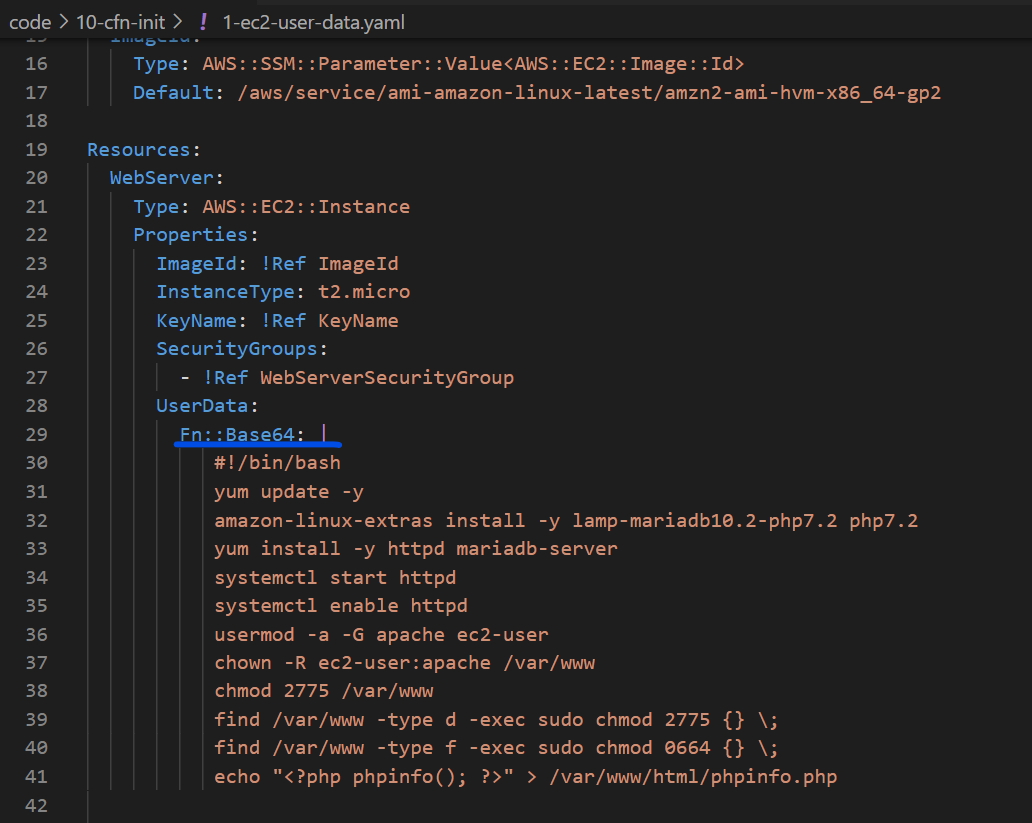


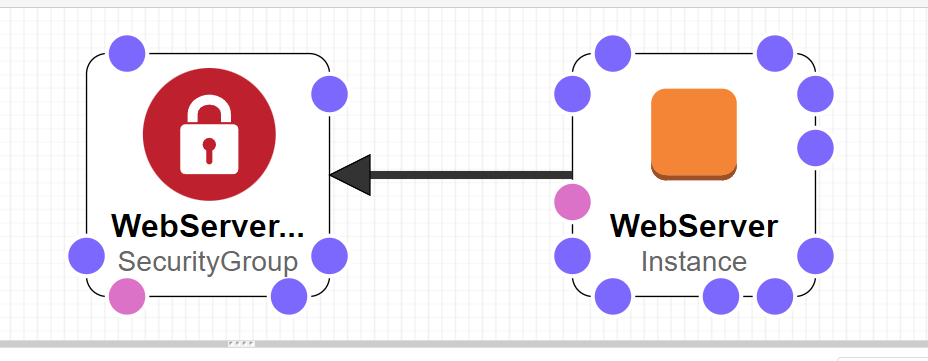
Php also running



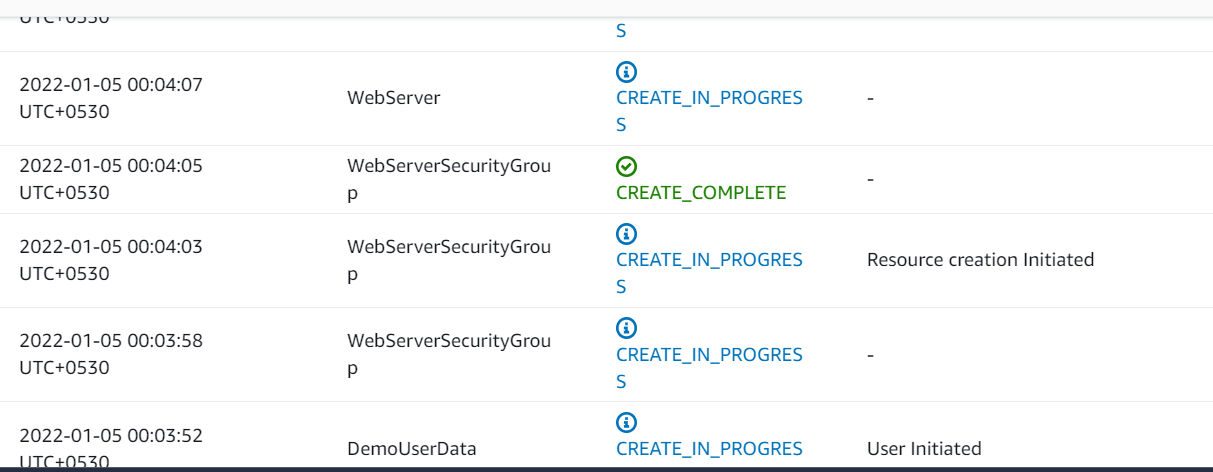
We can perform all above steps using CloudFormation Template

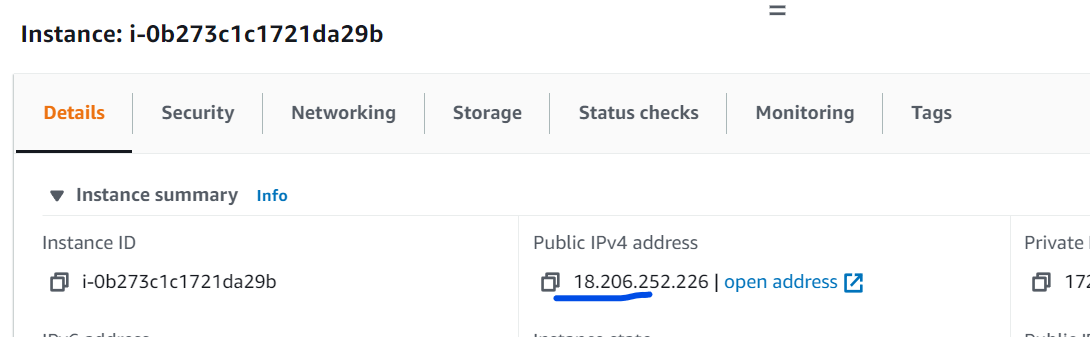
Important function is to notice use of Fn::Base64 to declare the userdata script

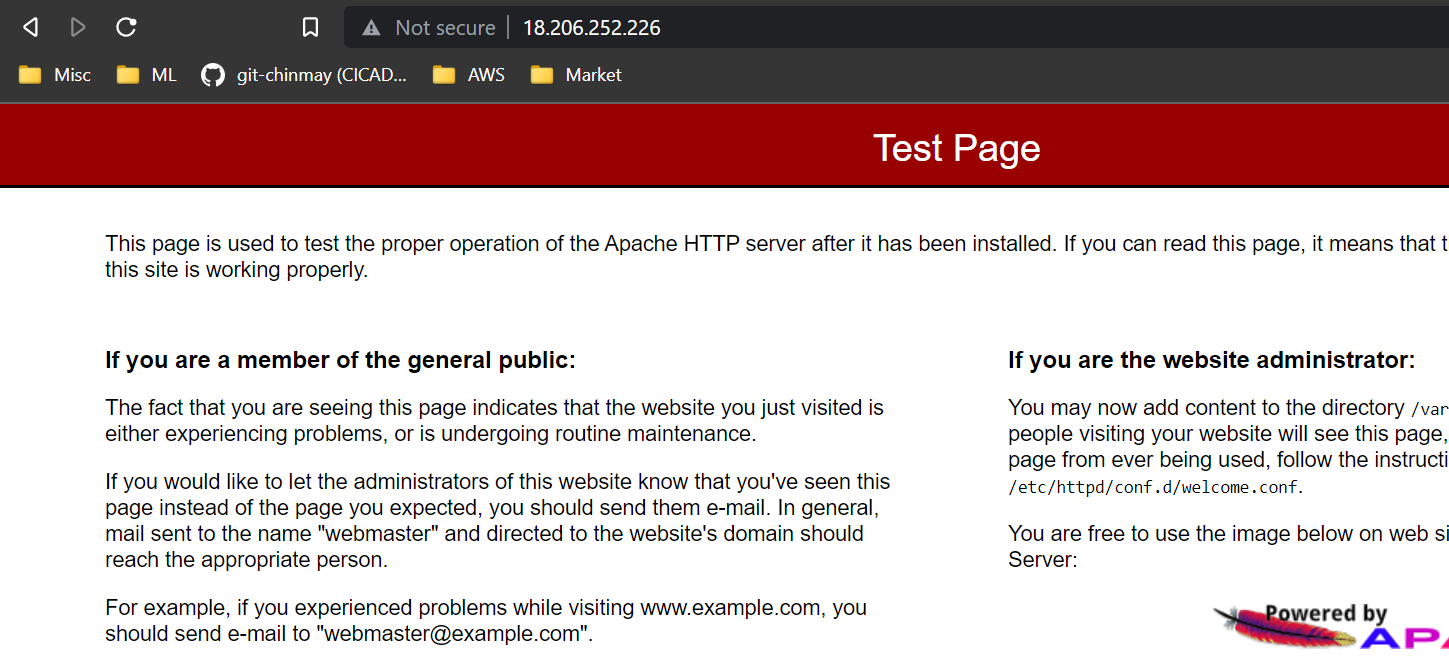




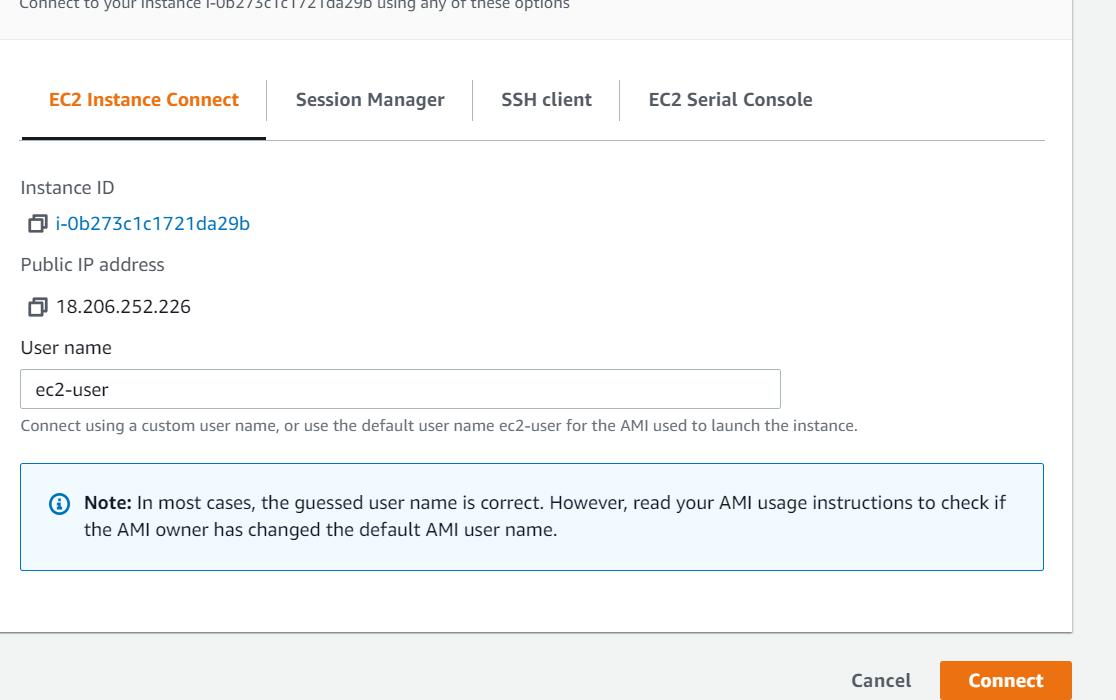
Create the stack

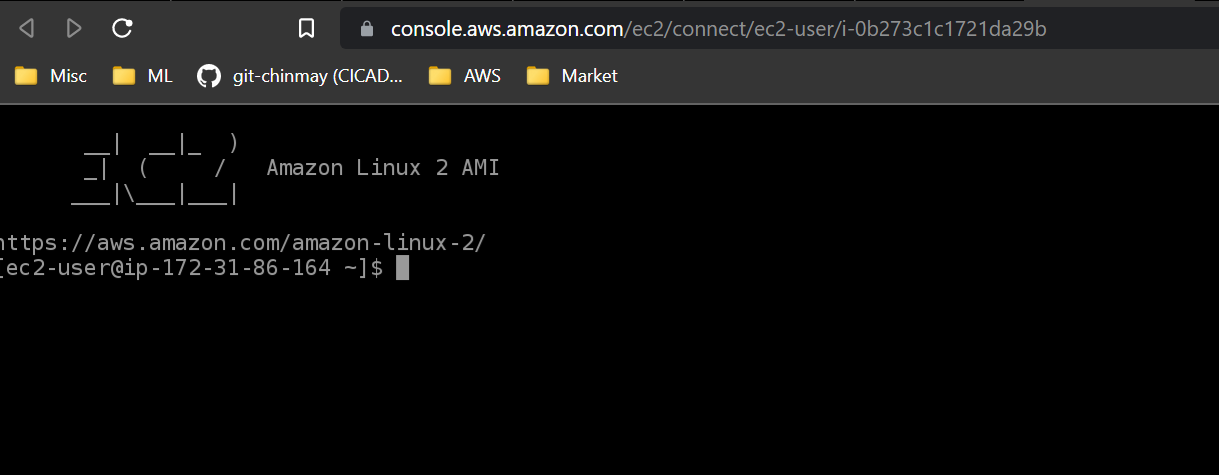




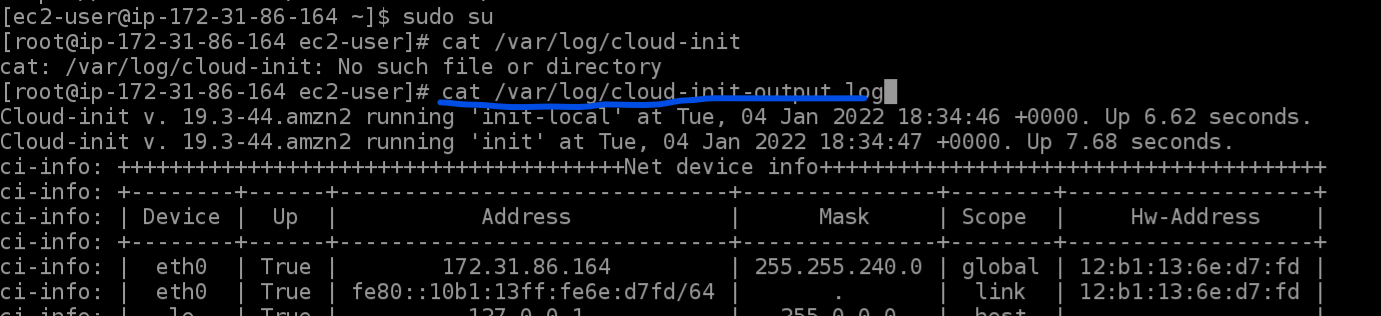


If we connect to the EC2





If we want to see what are the commands runs we can see inside the EC2 logs



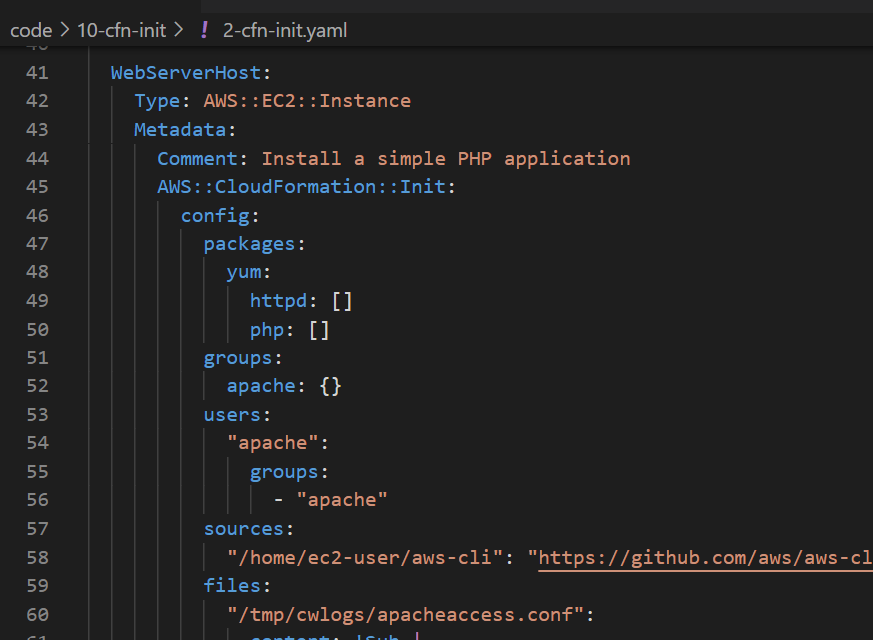
**CloudFormation helper Script:**

There are 4 python scripts come default in Amazon AMI Linux 2, we can also install them using yum on non Amazon Linux2 machines.

* cfn-init : Used to retrieve and interpret the resource metadata, installing packages, creating files and starting services.
* cfn-signal : wrapper script used by resources to talk back to CloudFormation for example to say resource creating was successful or not
* Cfn-get-metadata : A wrapper script to retrieve all metadata of a resource or path
* Cfn-hup : A daemon to check the updates to metadata and run custom hooks if foun any update.

**Cfn-init :**

Init structure is like below



Packages: used to download and install packages

Groups: Define user groups

Users: Define users and which group they belong to

Sources : Downlaod or archive files inside the Ec2

Files: Created files inside the EC2 using inline or can be pulled from a URL

Commands : run a seies of commands

Services : Run series of services using sysvinit

**AWS::CloudFormation::Authentication:**

Used to specify authentication credentials for files and sources in AWS::CloudFormation::Init

Two types:

* Basic : When the source is a URL
* S3 : When the source is an S3 bucket

**Function Fn::Sub : (!Sub)**

* Substitute function. Substitute a variable from a text.
* We can use it combing with Reference and Pseudo variables.
* String must contain ${<variable name>}

**Services:**

* Launch a bunch of services at EC2 instance launch.
* 
* It ensures services are started when file changed or package are updated.

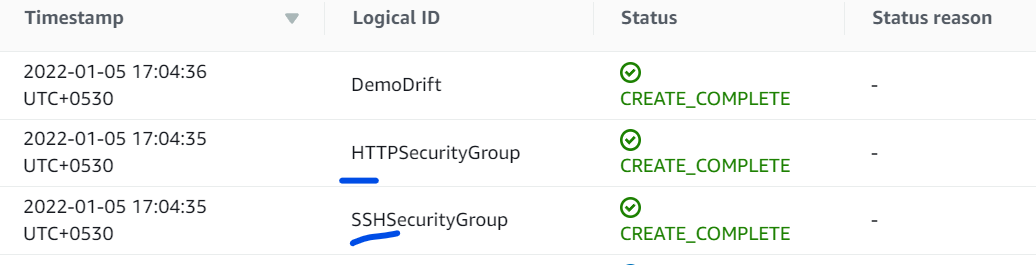
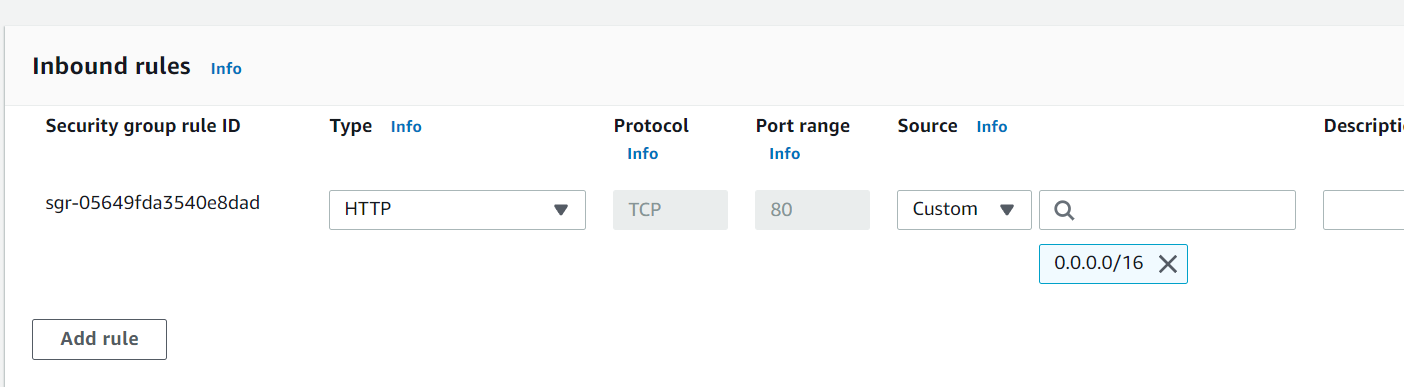
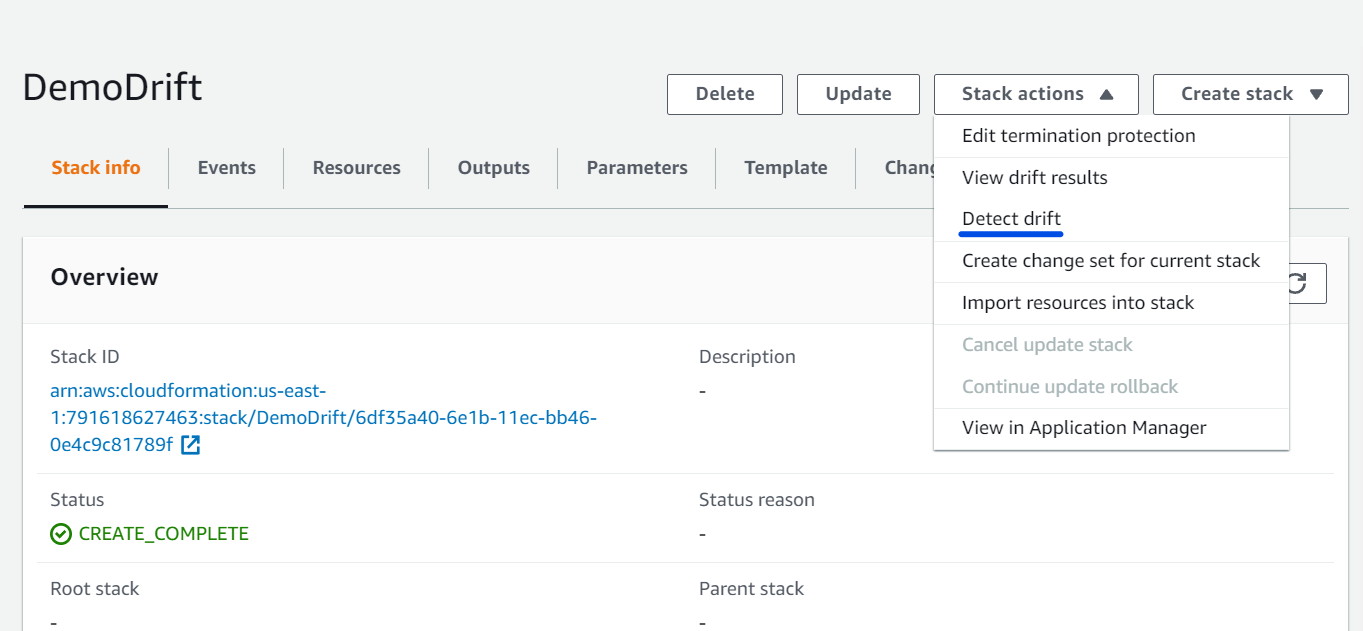
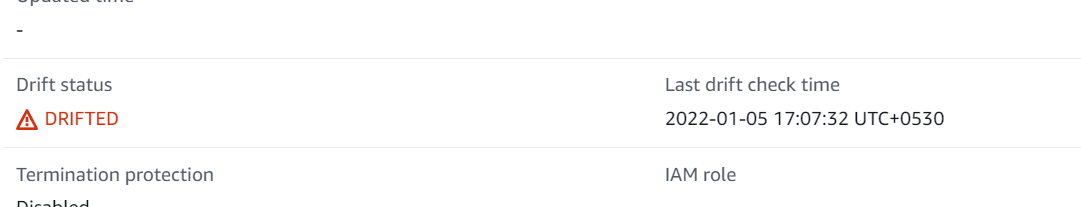
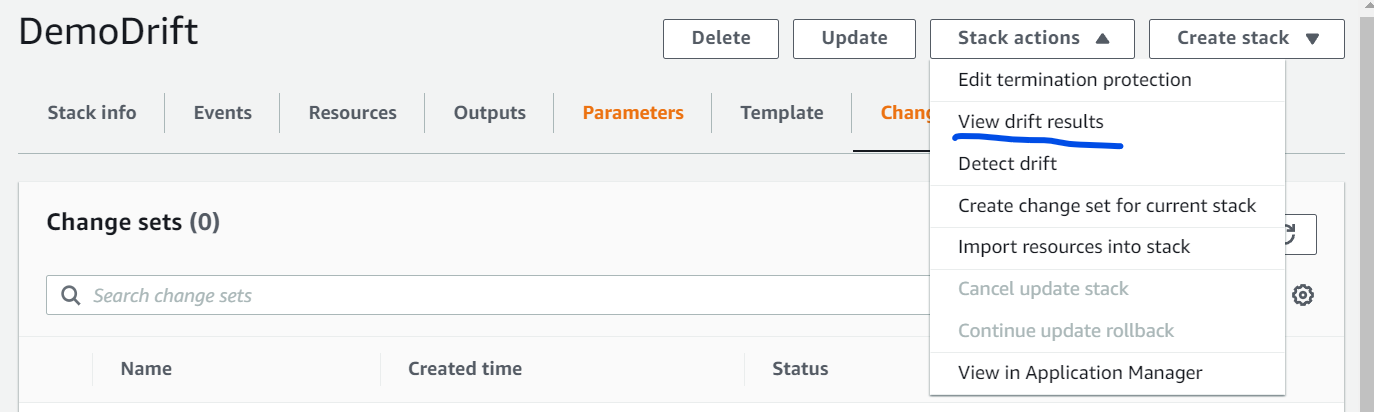
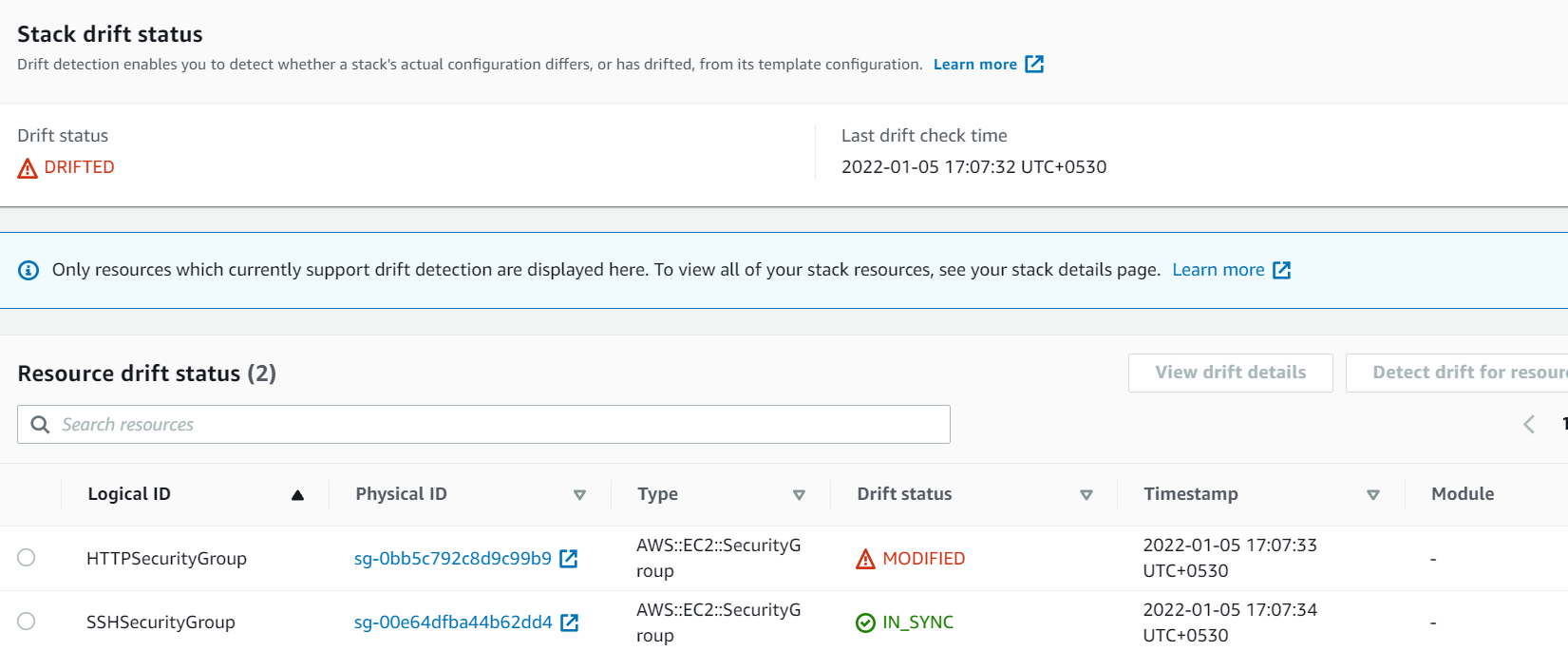
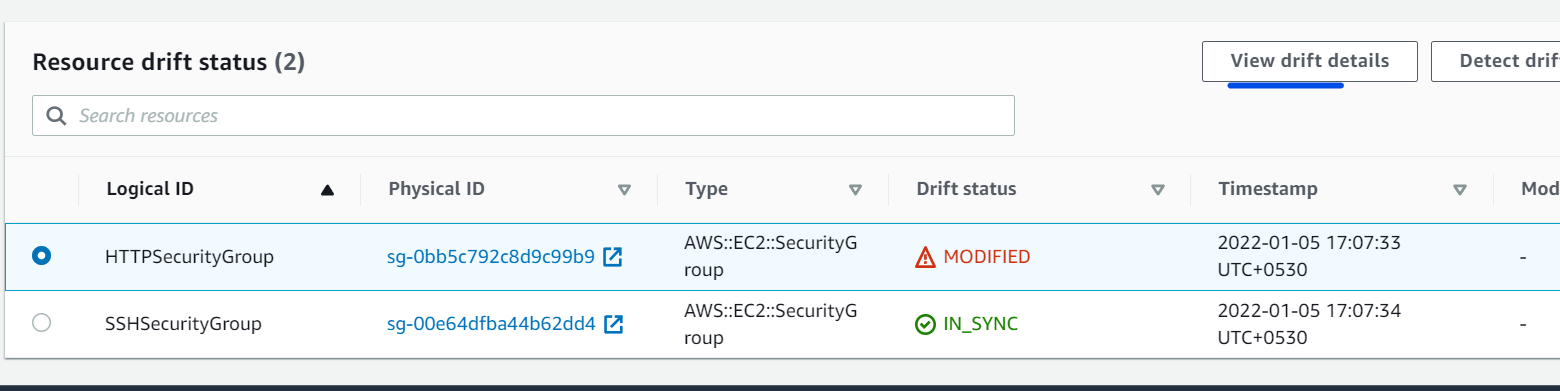
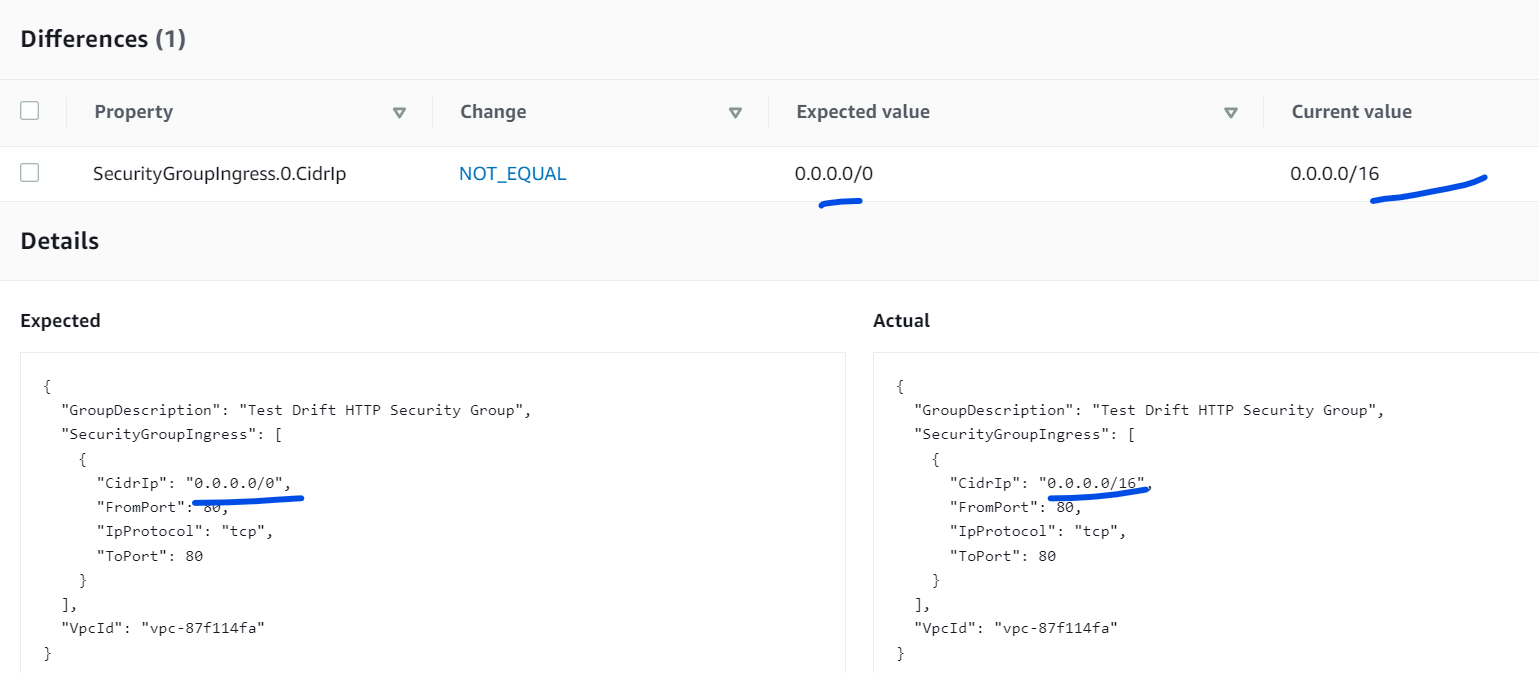
**User Data vs CloudFormation::Init vs Helper Scripts**

* EC2 User data is an imperative way to provision/bootstrap the EC2 instance using Shell syntax
* AWS::CloudFormation::Init is a declarative way to provision/bootstrap the EC2 instance using YAML or JSON syntax
* AWS::CloudFormation::Init is useless if it’s NOT triggered by a script within the EC2 User Data
* Triggering AWS::CloudFormation::Init inside EC2 User Data is done by using cfn-init or cfn-hup

**CloudFormation Drift:**

* By using template, we are able to build the infract structure but nothing prevents individual users to modify the configurations.
* CloudFormation Drift compares the entire stack or individual resources in it and shows if any difference is there with respect to original template.

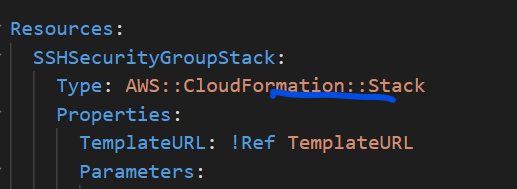
**Task: Create a stack and modify some configuration and detect the changes using Drift**

* Create the Stack using drift-security-group.yml template
* 
* Now modify one of the Security Group
* Modified the inbound rule
* 
* Now run the Drift
* 
* 
* 
* 
* 
* 

**Nested Stacks:**

Stacks that are part of other Stack.

As soon as ou saw below type for a resource understood that that it’s a nested stack



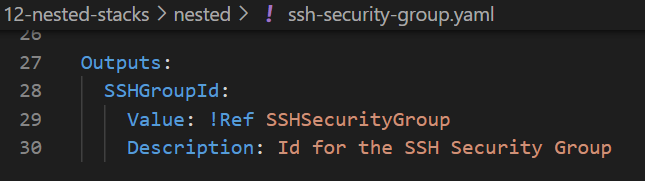
**Task: Nested stack demo**

We will have two stacks

* Stack-1 = Will create a Security group
* Stack-2 = Will create an EC2 instance

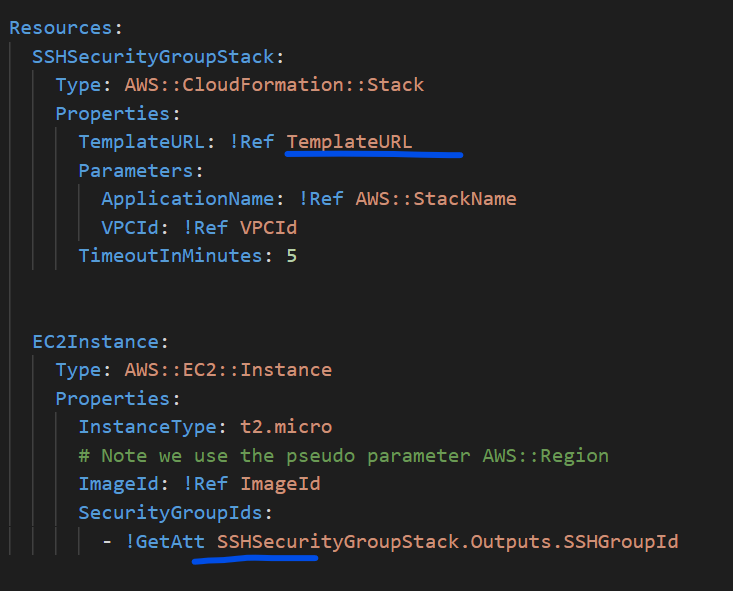
But we will use stack-1 for assigning the Security group to Stack-2 EC2 creations. We will basically add Output to expose the SG ids and will use it in Stack-2.

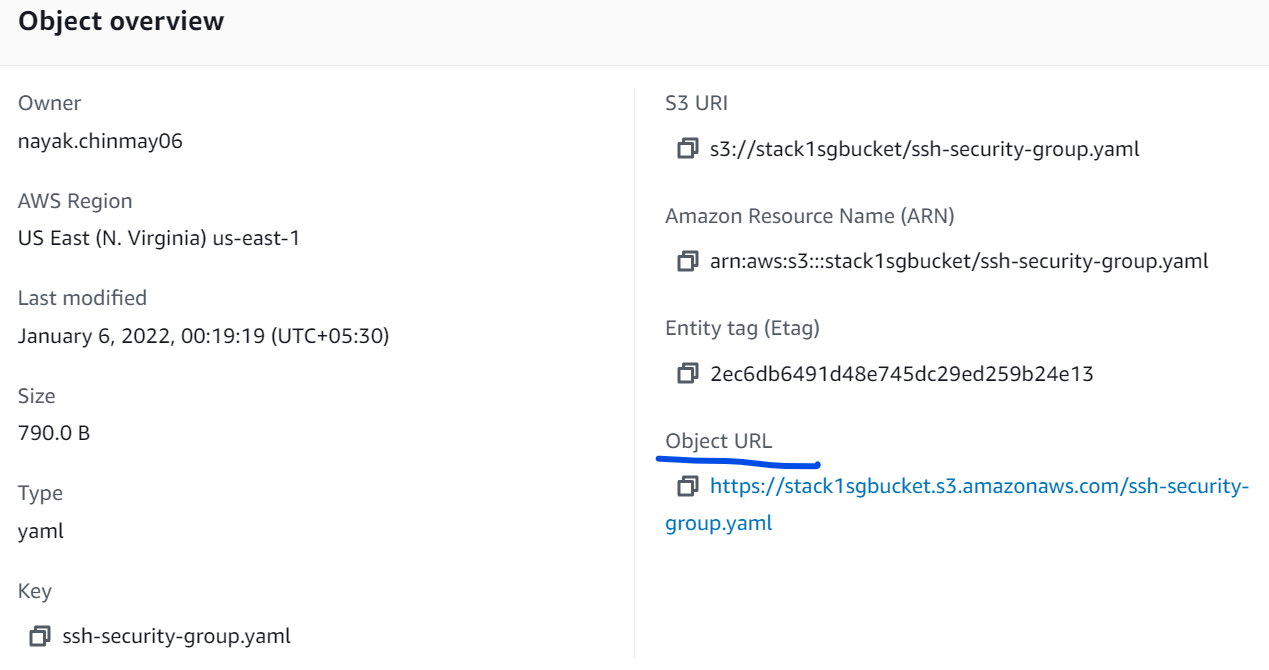
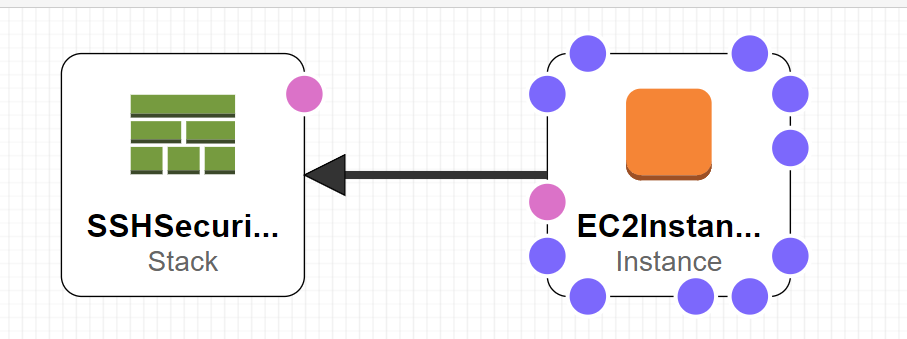
Stack-1



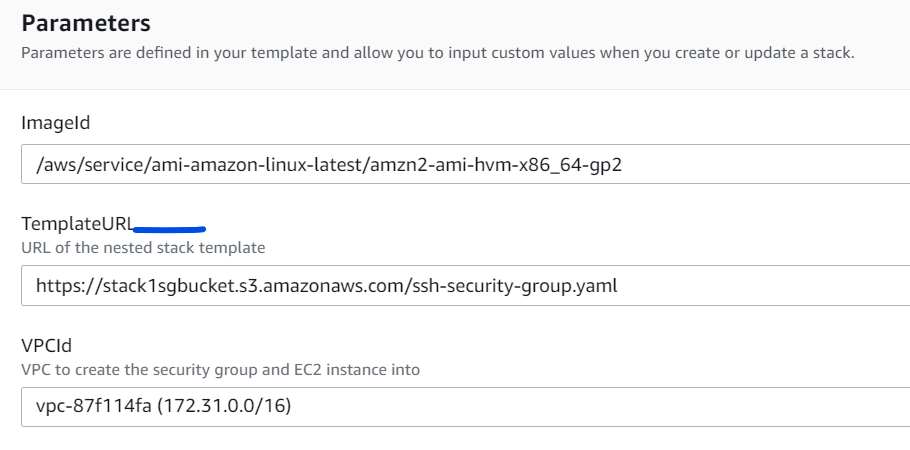
Stack-2

* Template URL we will get from Stack-1 properties
* Securitygroupids we are fetching using the GetAtr function.

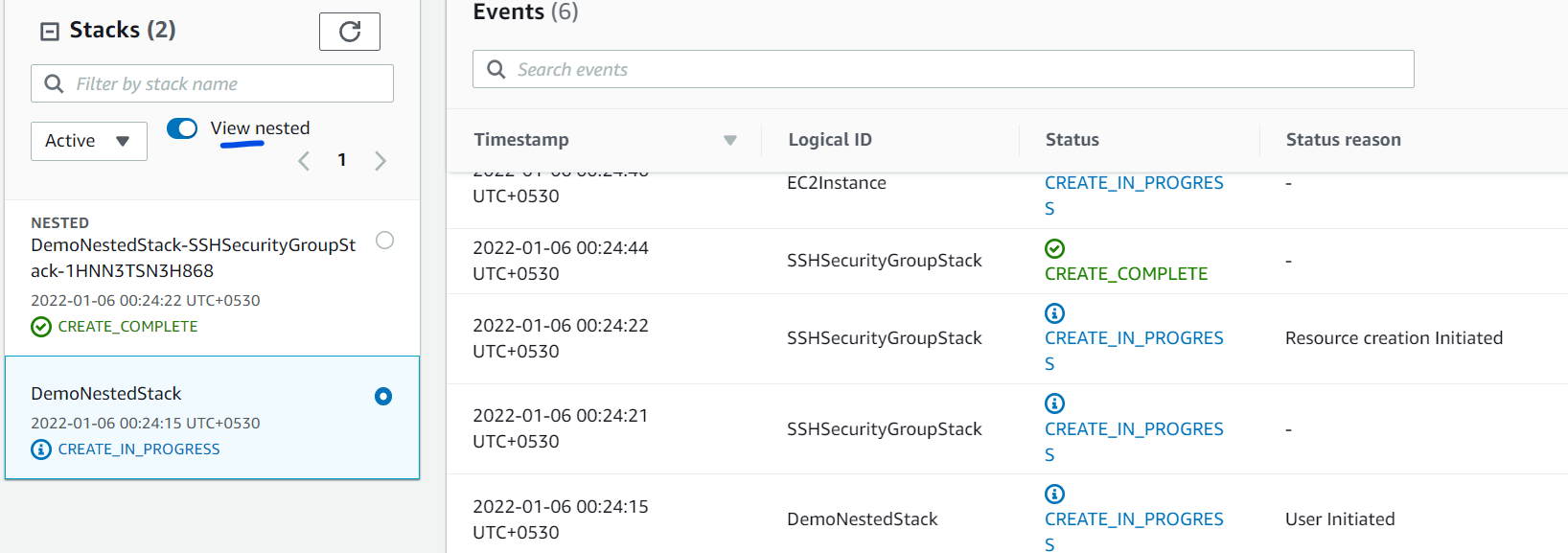


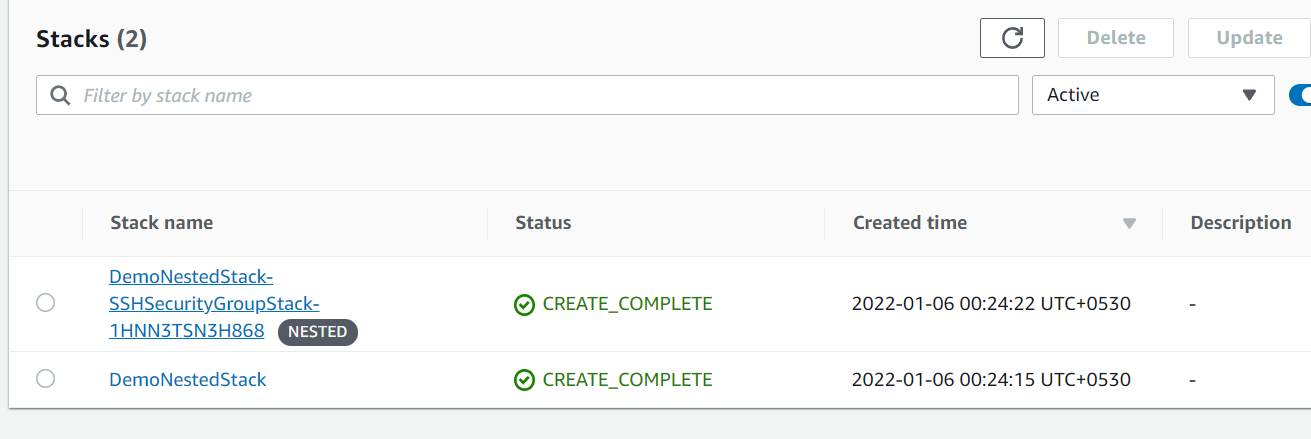
* First put the Stack-1 template into a S3 bucket so tat we get a template URL
* Copy the object URI
* 
* <https://stack1sgbucket.s3.amazonaws.com/ssh-security-group.yaml>
* Now create the Stck-2 on CloudFormation
* 

Fill the template url parameter



We will observe a change in left side

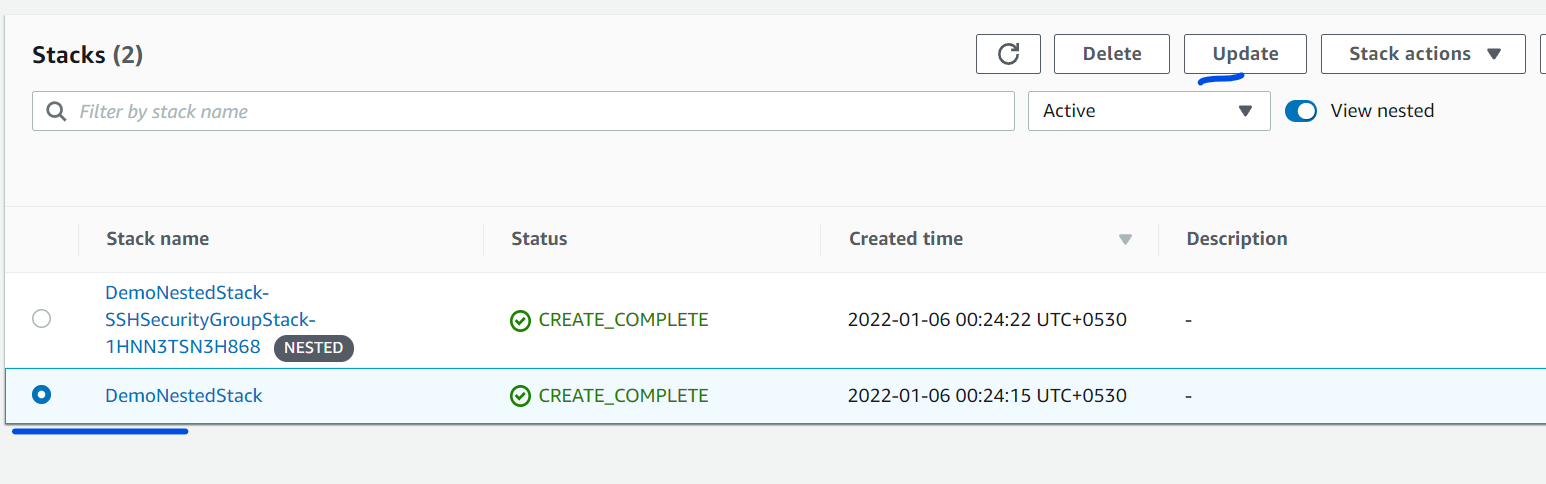




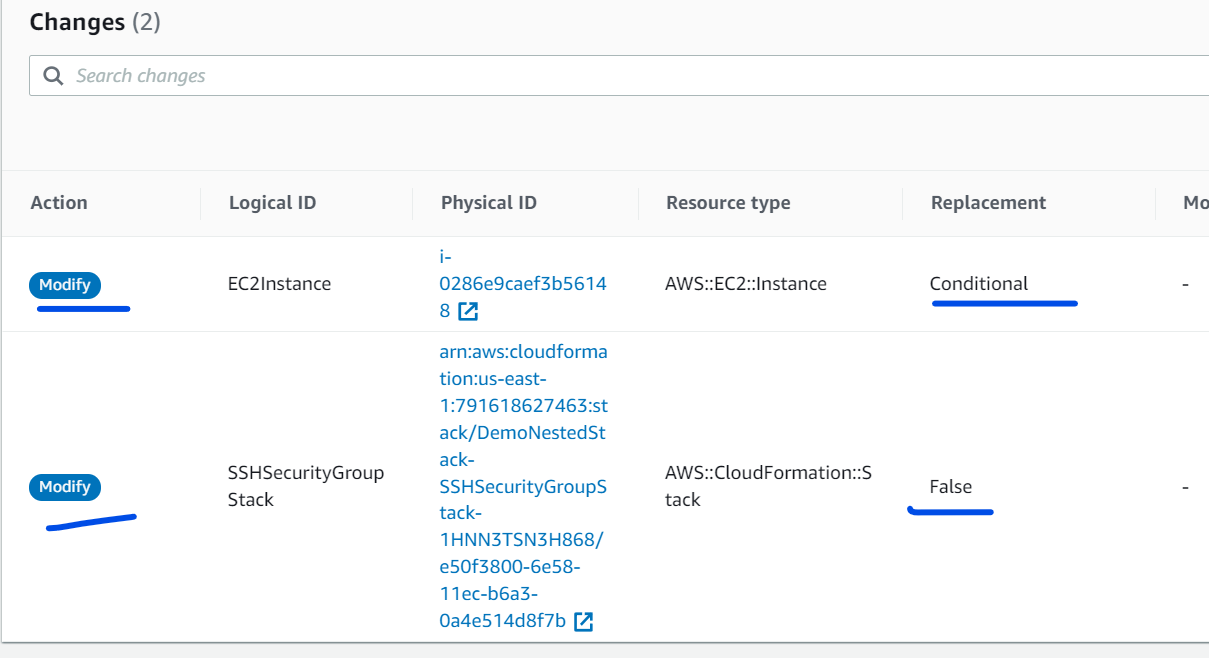
If you update the child template then we have to redo the same steps

Upload the updated template to S3

Come to the CloudFormation and update only the Root/Parent Stack

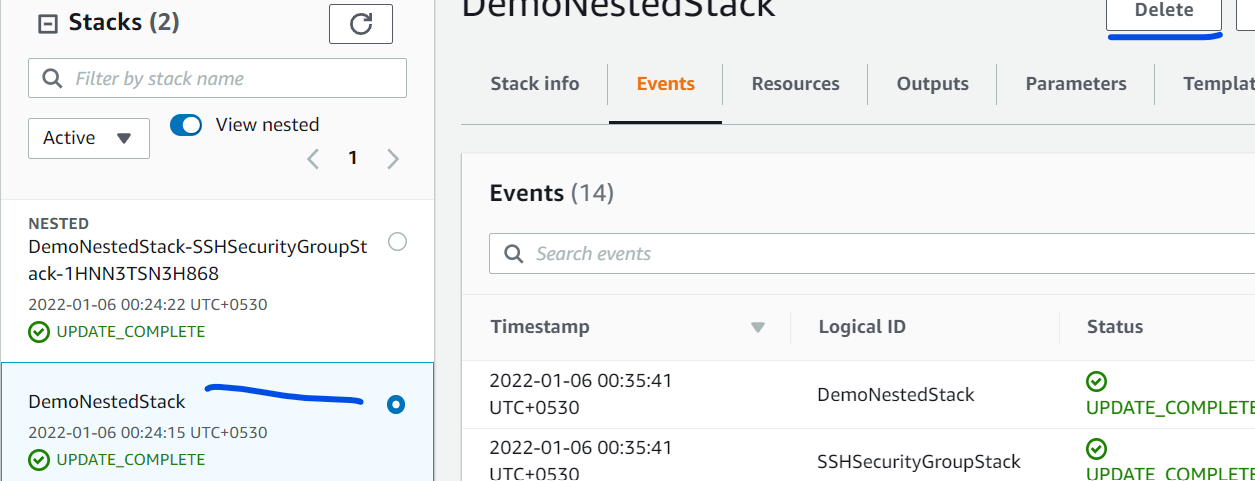


Although we are not changing anything on root template still its showing modified bcz it may be affected due to change in child template.



Deleting a Nested Stack:

Never ever delete the child/nested stack. Always delete the Root/Parent stack.



Exported Stack Output Values vs. Using Nested Stacks:

* If you have a central resource that is shared between many different other stacks, use Exported Stack Output Values
* If you need other stacks to be updated right away if a central resource is updated, use Exported Stack Output Values
* If the resources can be dedicated to one stack only and must be re-usable pieces of code, use Nested Stacks
* Note that you will need to update each Root stack manually in case of Nested stack updated