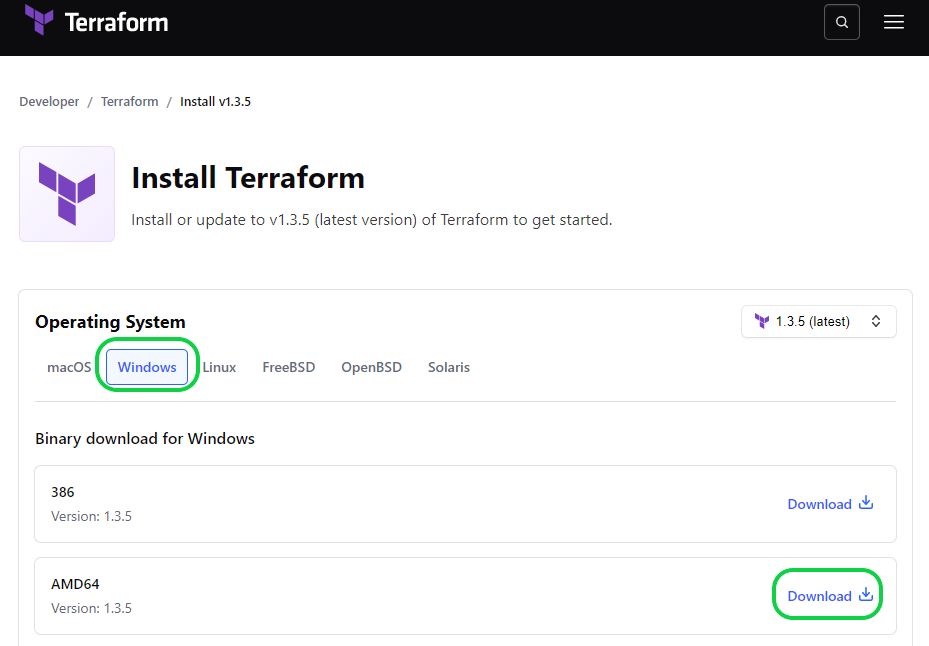
**Terraform**

1. [Download and install Terraform on Windows x64 and Amazon Linux.](#Task_1)
2. [Creating “terraform” user IAM in AWS console with Admin access.](#Task_2)
3. [Creating infrasrtucture.](#Task_3)
4. [Project initialization and terraform plan.](#Task_4)
5. [Terraform apply and creating infrastructure](#Task_5).
6. [Simple Terraform destroy](#Task_6).
7. [Bootstrapping container instances with Amazon EC2 user data](#Task_7).
8. [Terraform implicit dependencies example](#Task_8).
9. [Terraform explicit dependencies example](#Task_9).
10. [Terraform variables](#Task_10).
11. [Terraform Output](#Task_11).
12. [Terraform Modules](#Task_12).

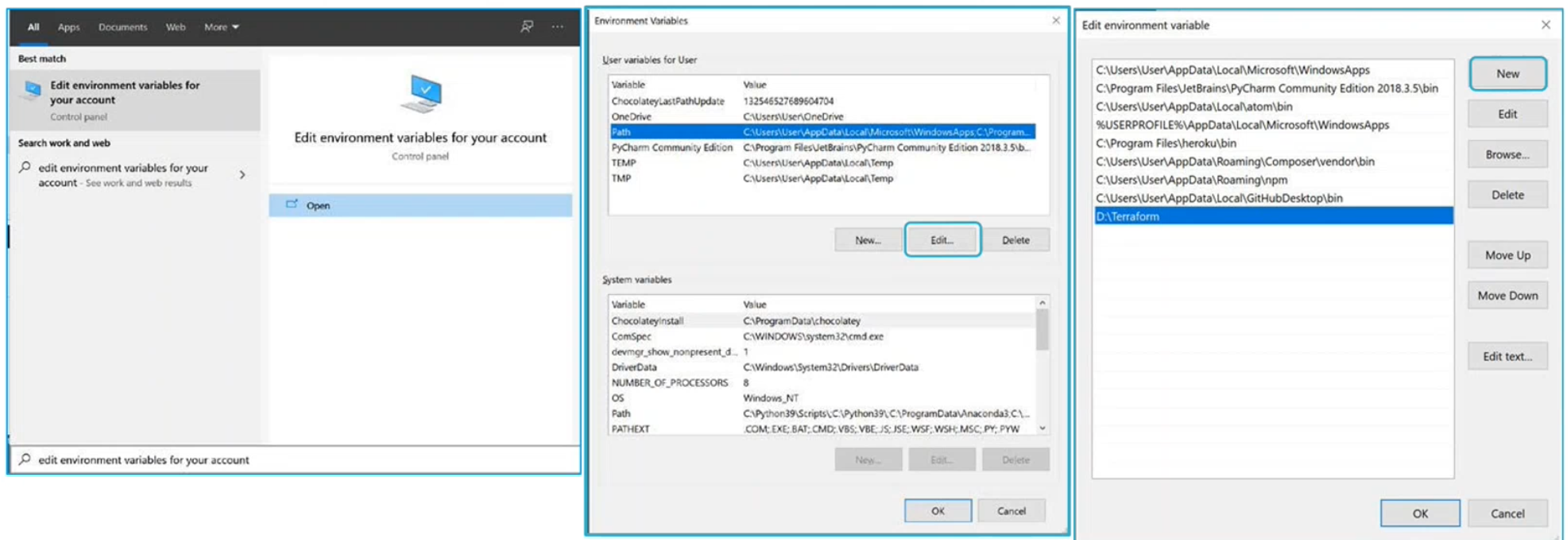
1. [Download and install Terraform](https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli) on [Windows x64](https://developer.hashicorp.com/terraform/downloads) and Amazon Linux.

* **Windows**

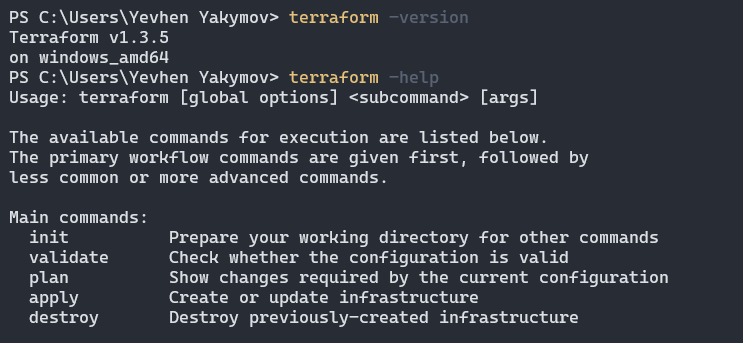
1. Download newest version Terraform 1.3.5 from [website](https://developer.hashicorp.com/terraform/downloads) and unzip archive



1. Added path where are you save terraform.exe to system PATH

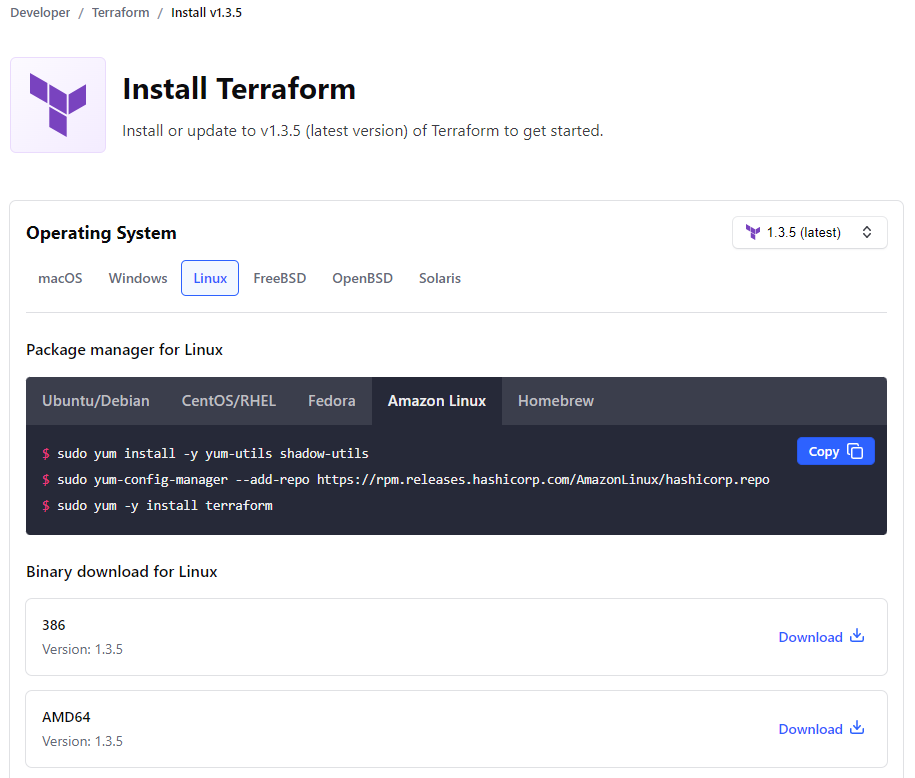


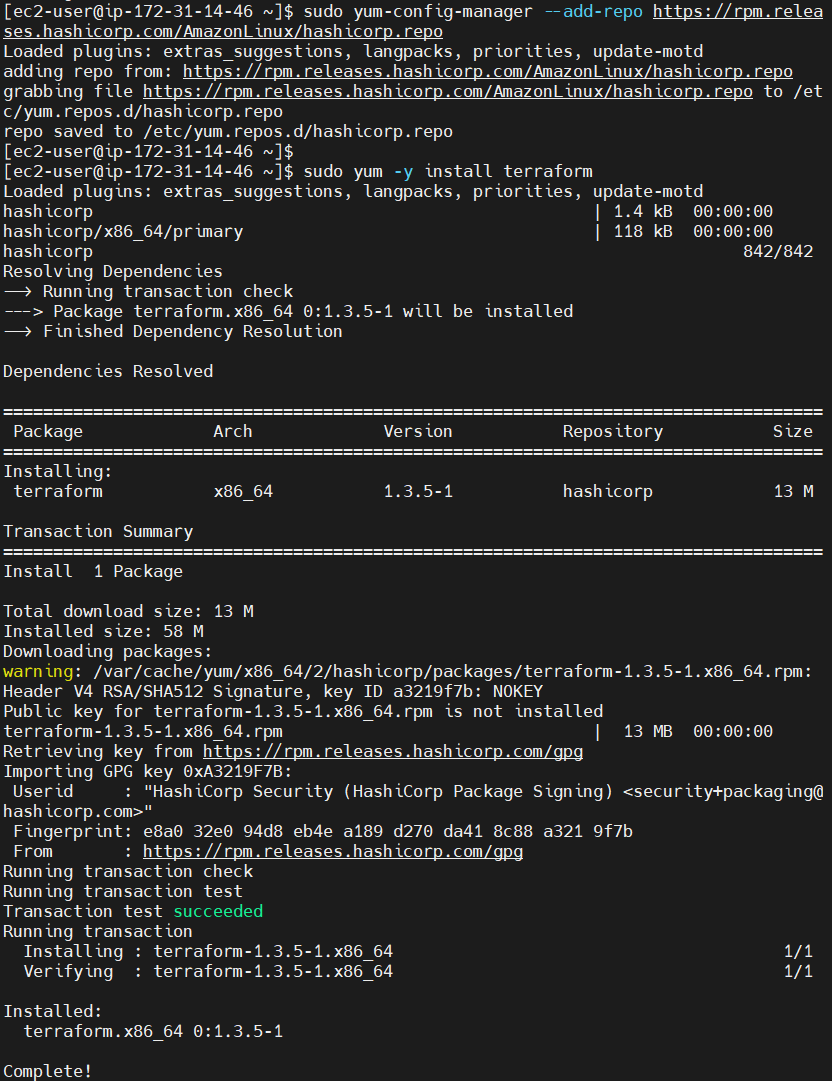
1. Verify the instalation

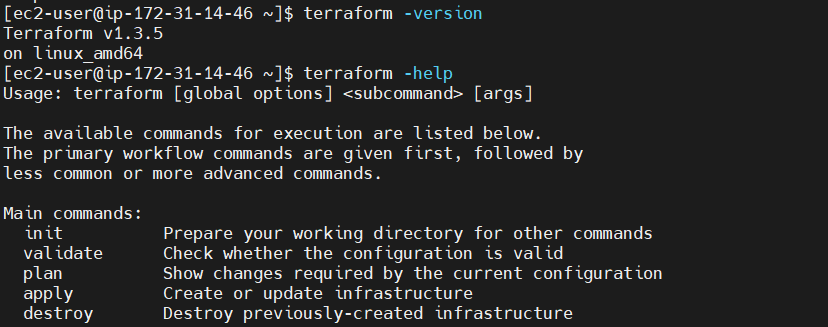


* **Amazon Linux**

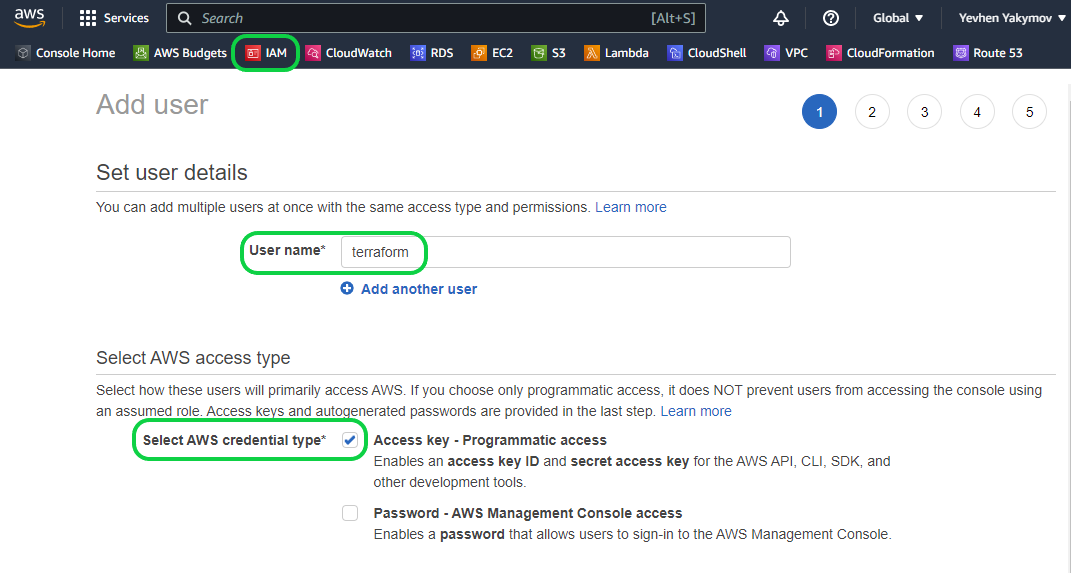
1. Download and install (package manager or binary download it is up to you)

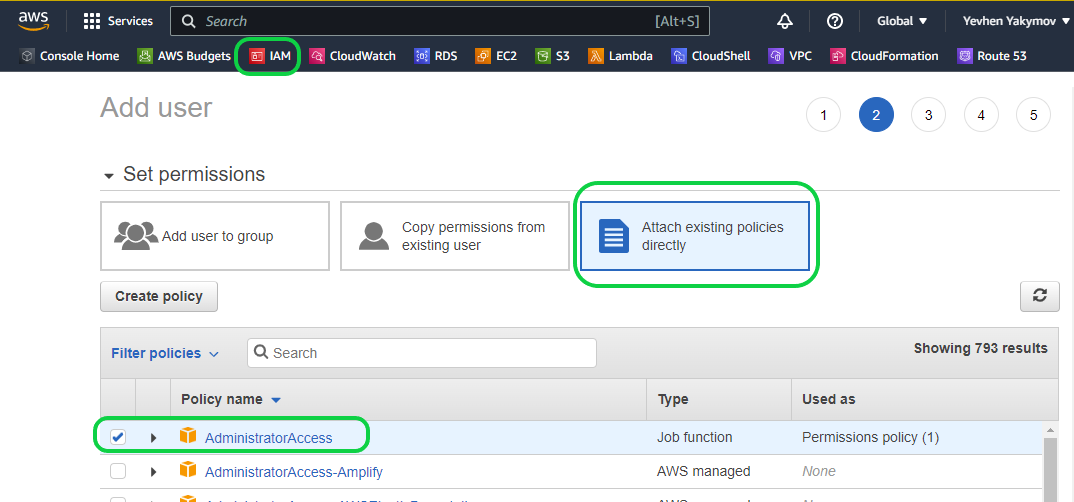


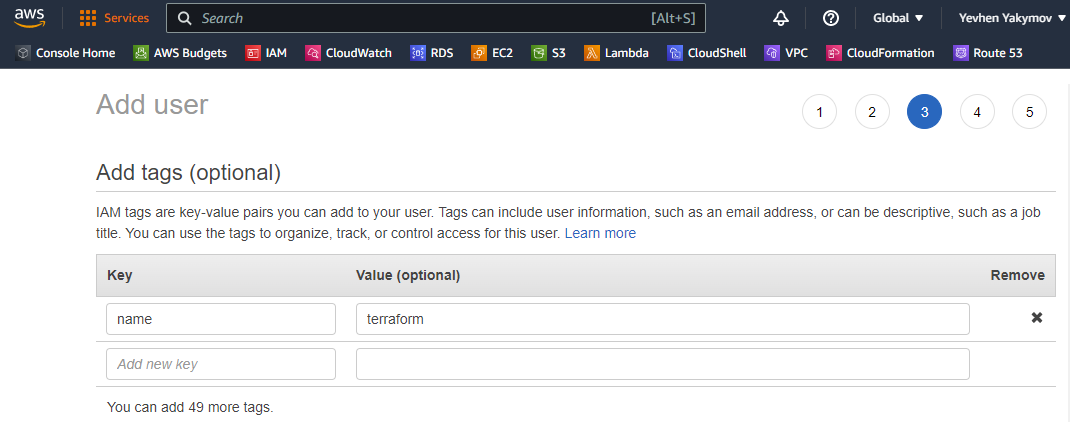


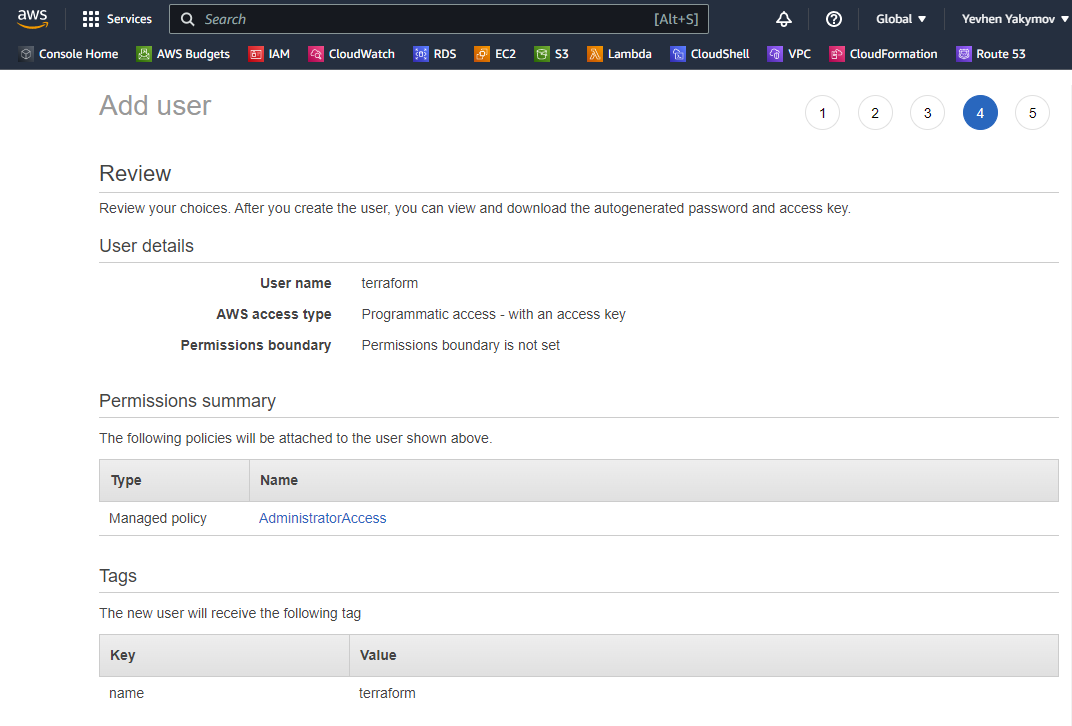


1. Creating “terraform” user IAM in AWS console with Admin access

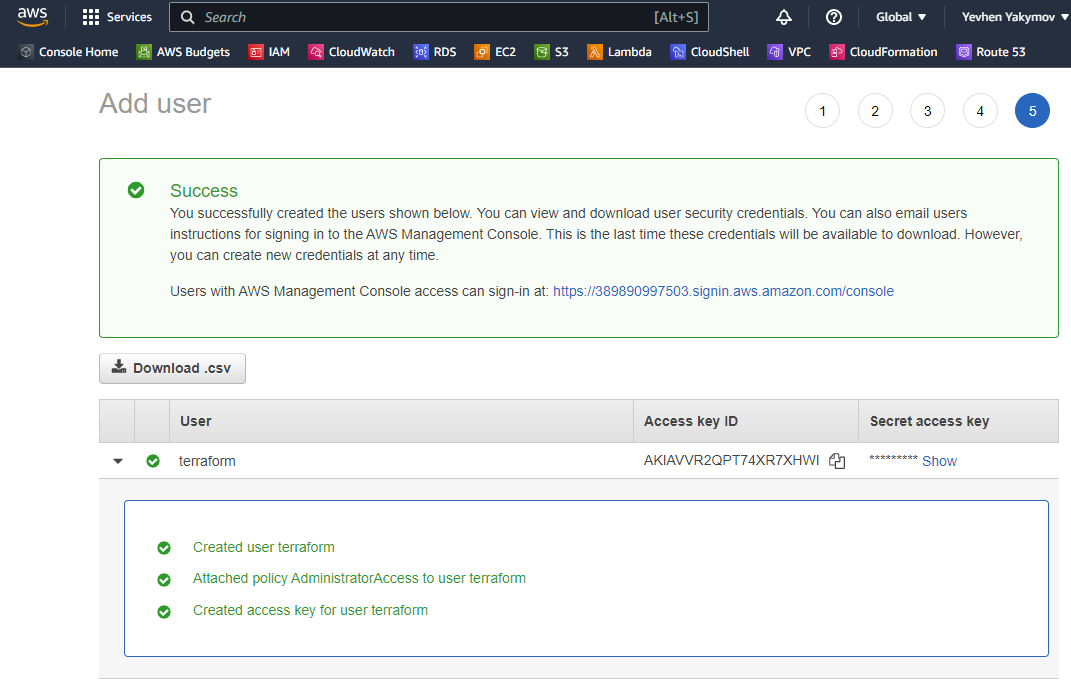






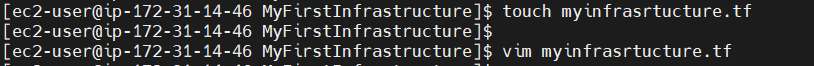


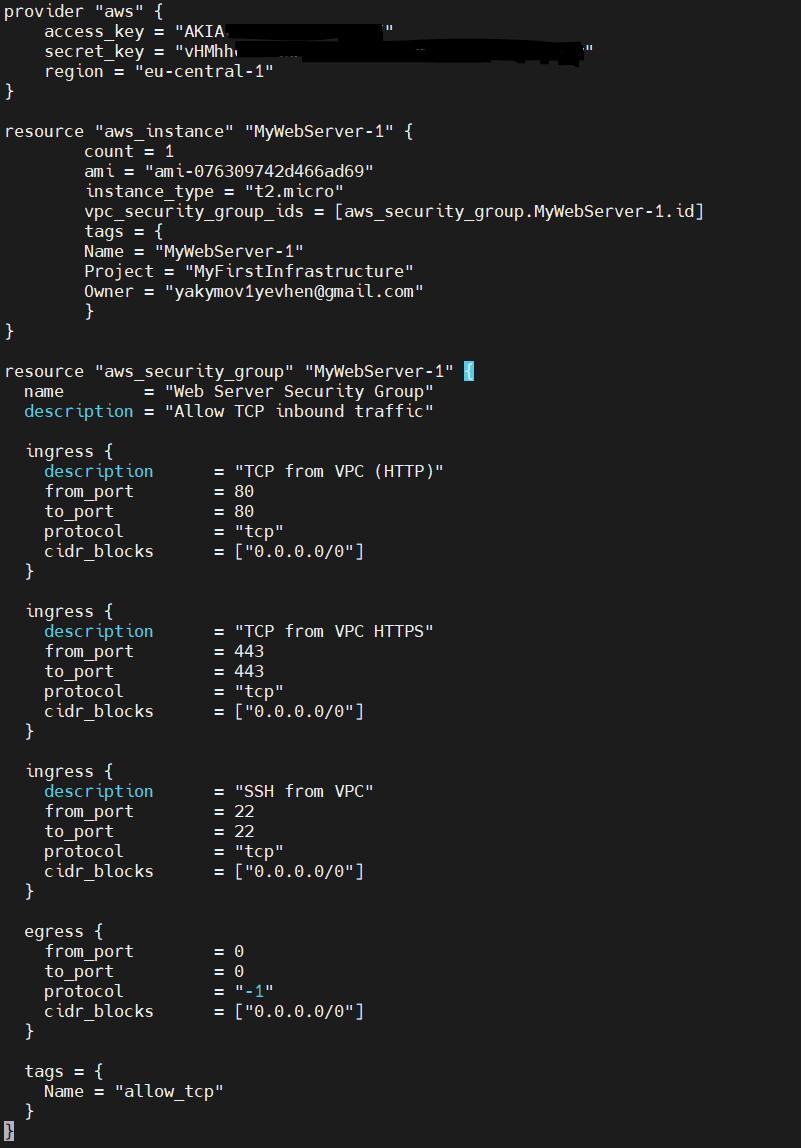
Save **Access key ID** and **Secret access key**. This credentials should be used in **“provider“** section of \*.tf file(**access\_key** and **secret\_key** variable)



1. Creating infrasrtucture.

Create **myinfrasrtucture.tf** file with a description of the infrastructure to be deployed.





**You have different ways to grant credentials, besides directly specifying access\_key and secret\_key it in the file \*.tf.**

**AWS Credentials:**

1. export before apply:

***bash***

*export AWS\_ACCES\_KEY\_ID="XXXXXXXXXXX"*

*export AWS\_SECRET\_ACCES\_KEY="XXXXXXXXXXX"*

*export AWS\_DEFAULT\_REGION="your-region"*

1. specify in the provider's section of **\*.tf** file:

***Go***

*provider "aws" {*

*access\_key="XXXXXXXXXXX"*

*secret\_key="XXXXXXXXXXX"*

*region="your-region"*

*}*

1. in the credentials file:

~/path/to/file/credentials

***Text***

[default/IAM user]

*aws\_access\_key\_id=XXXXXXXXXXX*

*aws\_secret\_access\_key=XXXXXXXXXXX*

*~/path/to/file/config*

***Text***

[default/IAM user]

*region=us-west-2*

***Go***

*provider "aws" {*

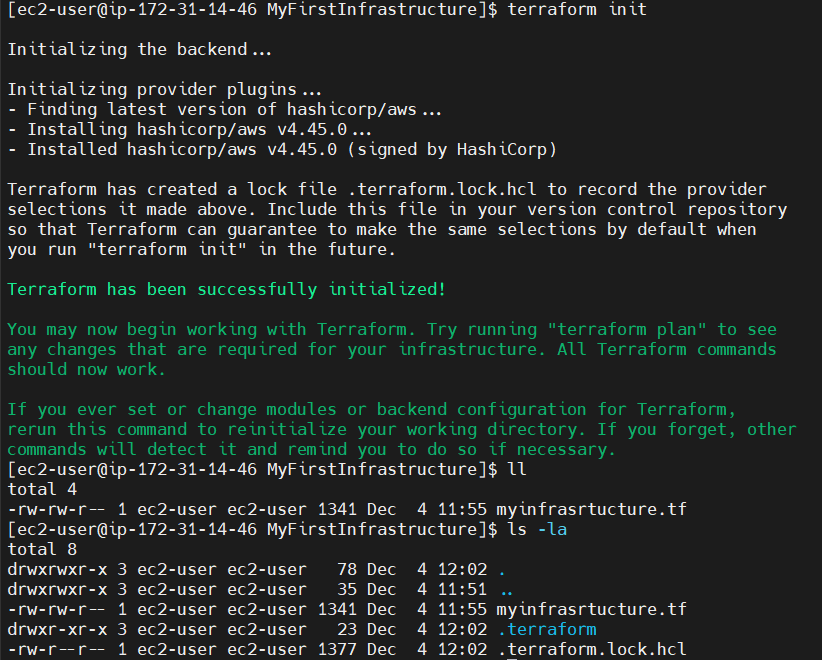
*shared\_credentials\_files = ["~/path/to/file/credentials"]*

*shared\_config\_files = ["~/path/to/file/config"]*

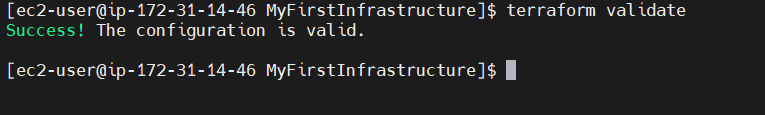
*}*

1. Project initialization and terraform plan

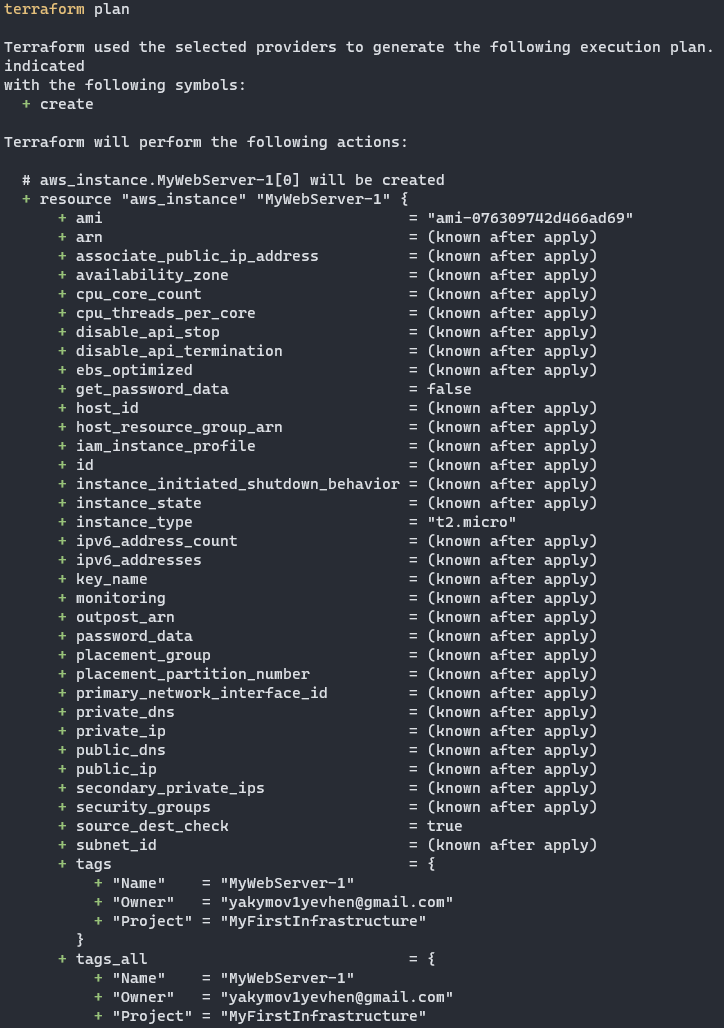
**terraform init** – prepare your working directory for other commands(download providers)

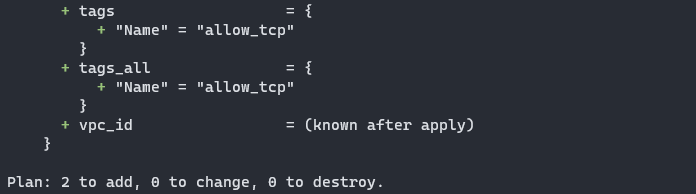


**terraform validate** – check whether the configuration is valid



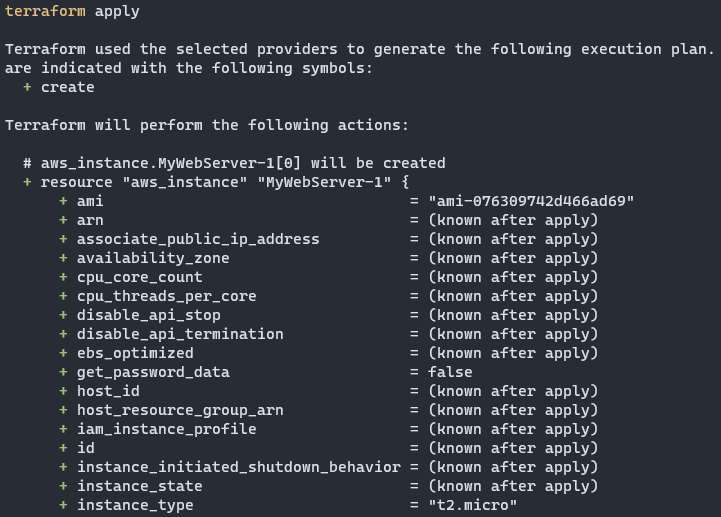
**terraform plan** – show changes required by the current configuration

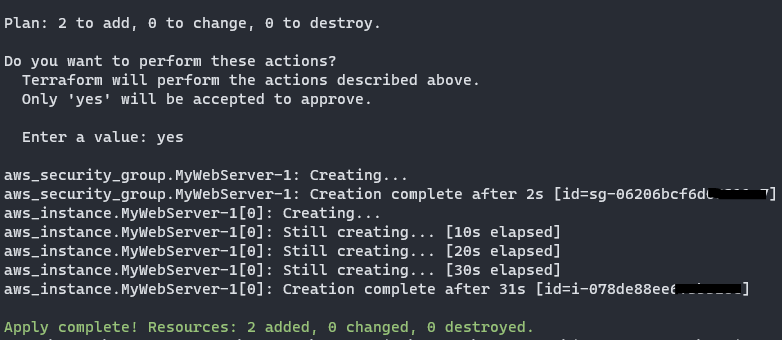


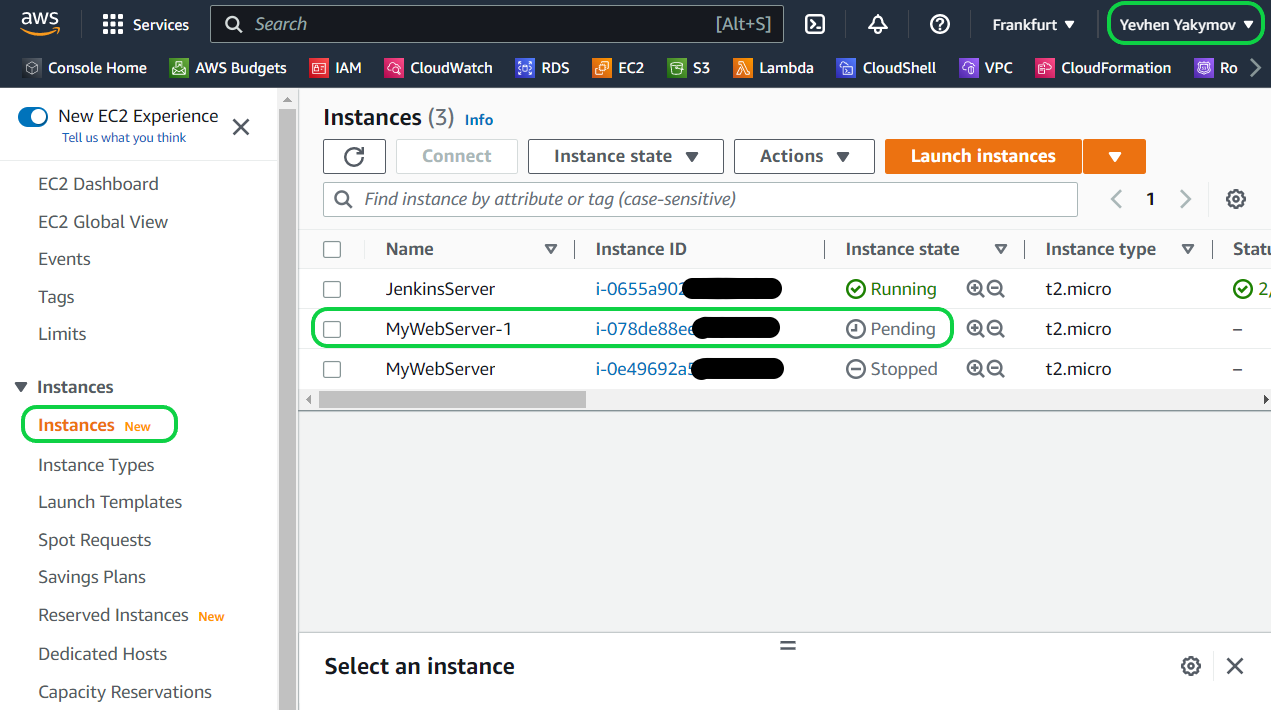


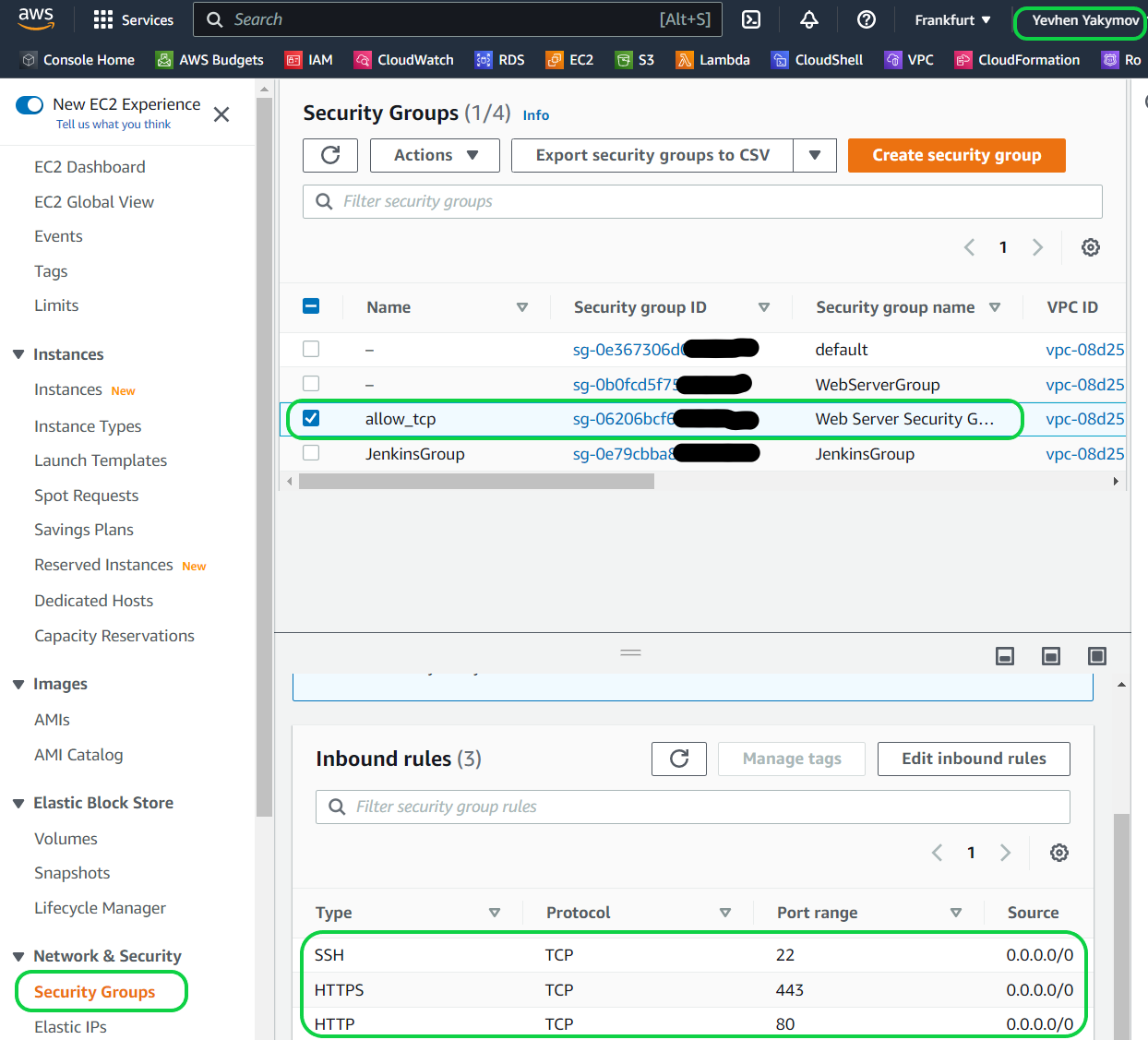
1. Terraform apply and creating infrastructure

**terraform apply** – create or update infrastructure



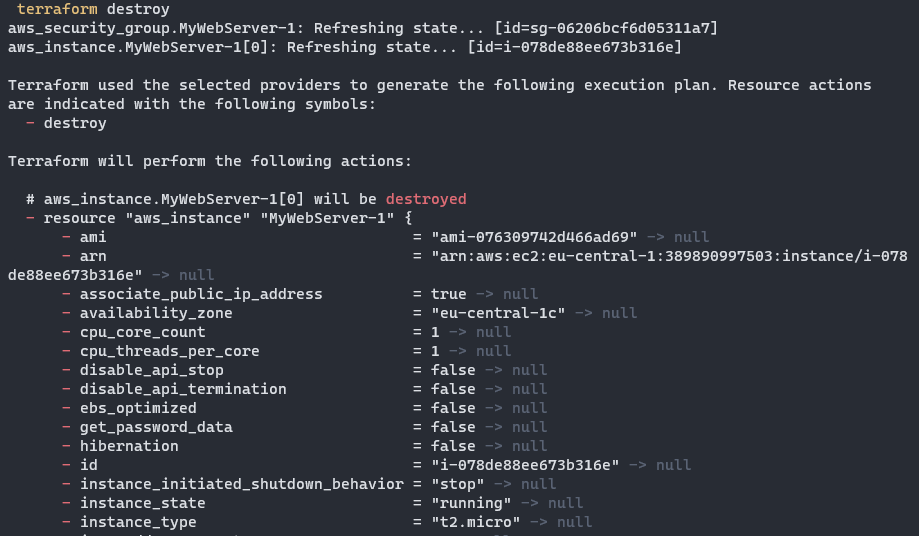


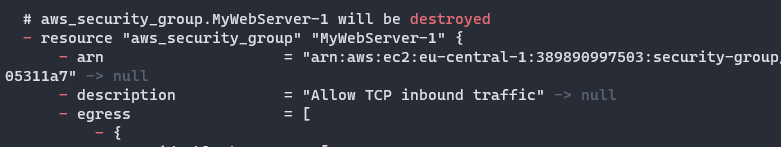


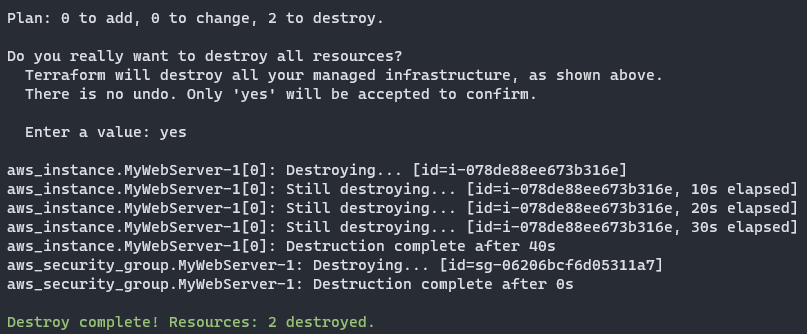


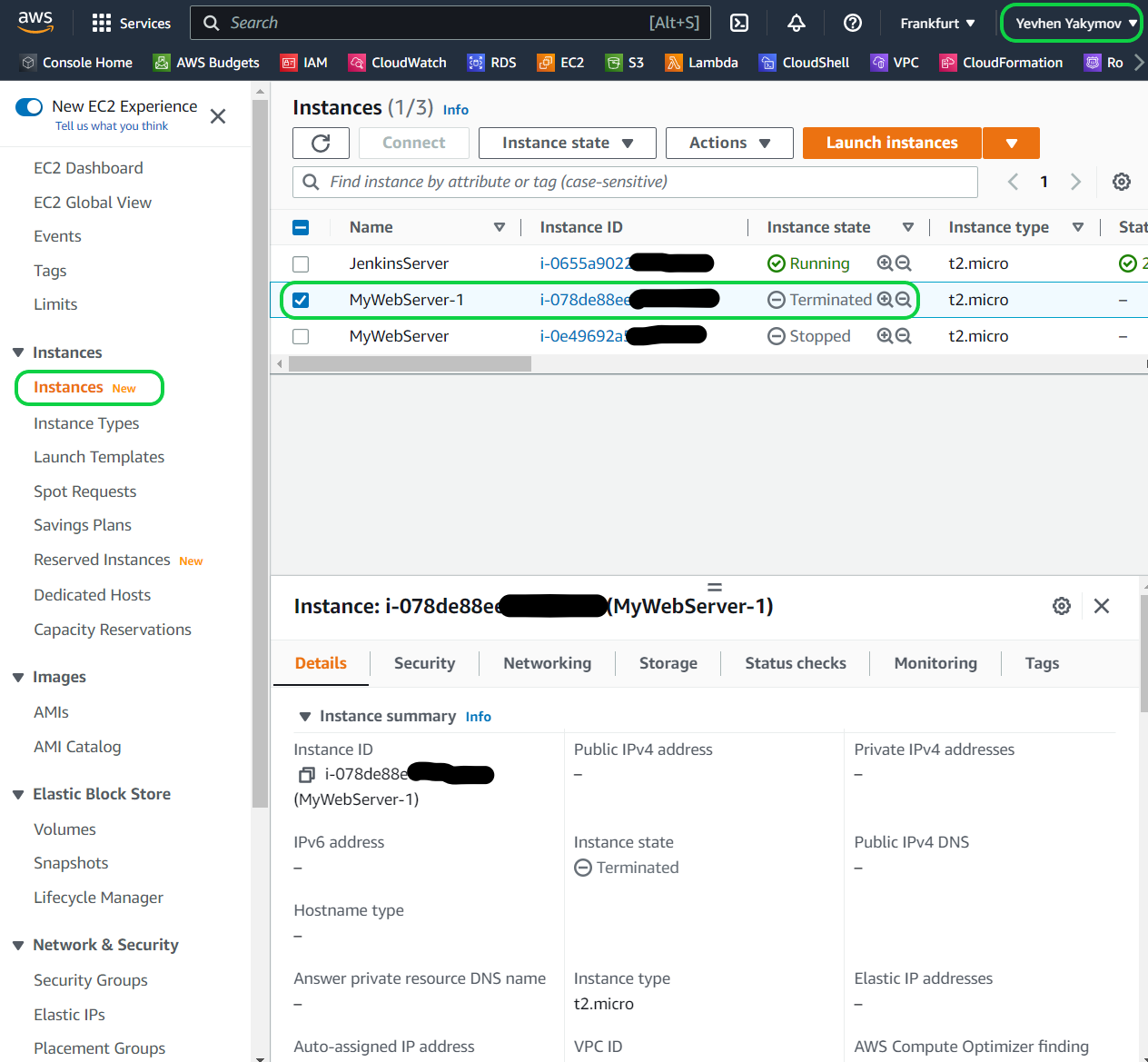
1. Terraform destroy.

**terraform destroy** – destroy previously-created infrastructure



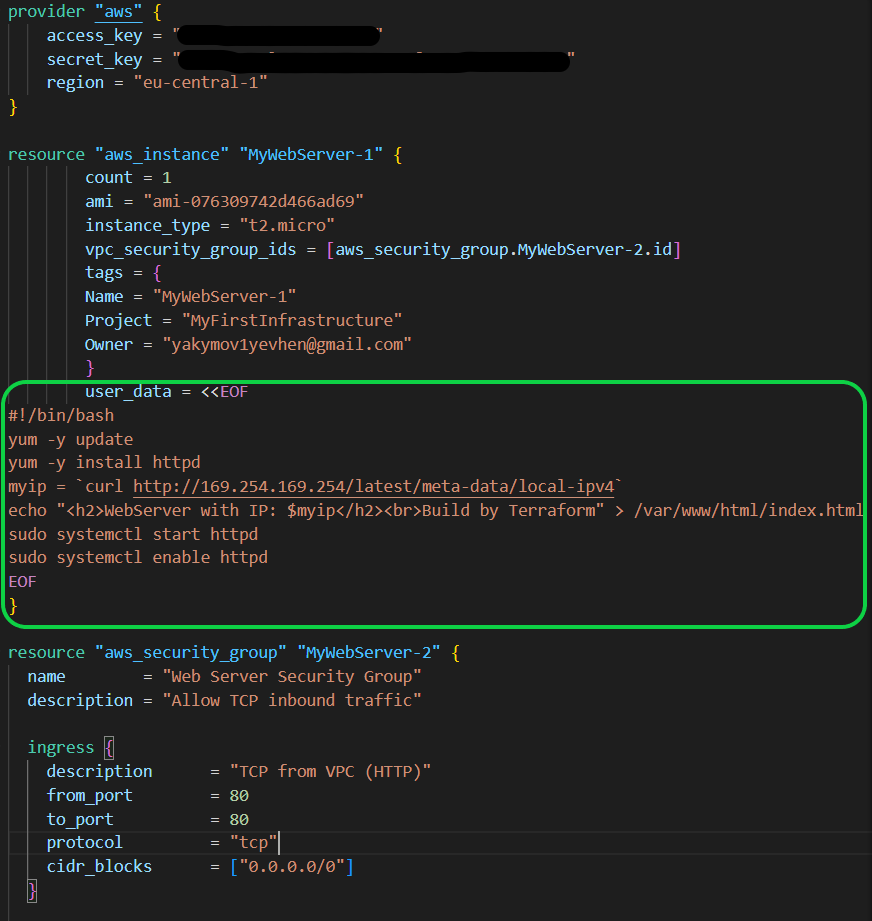






1. [Bootstrapping container instances](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/bootstrap_container_instance.html) with Amazon EC2 user data.

You can add in **\*.tf** file **user\_data** for the section where aws instance is discribed. **user\_data** is done via nested Heredoc in the **resource / user\_data section**.



You can also make a shell script and add it to user\_data.

**user\_data = file("script.sh")**

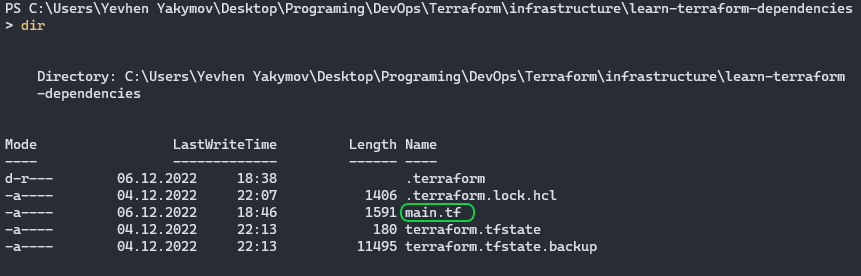
**It should remembered that the main purpose of Terraform is not a configure infrastructure, but creating infrastructure. So use user\_data in specific cases.**

1. Terraform implicit dependencies example.

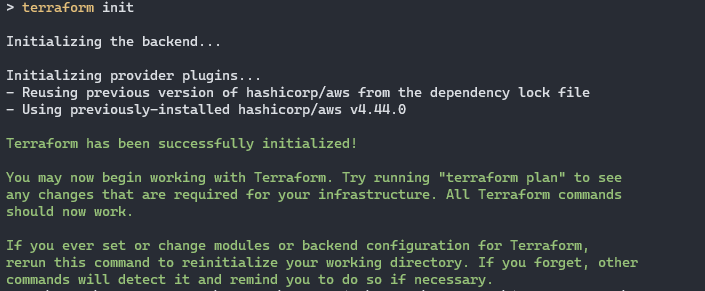
Most common source of dependencies is an implicit dependency between two resources.

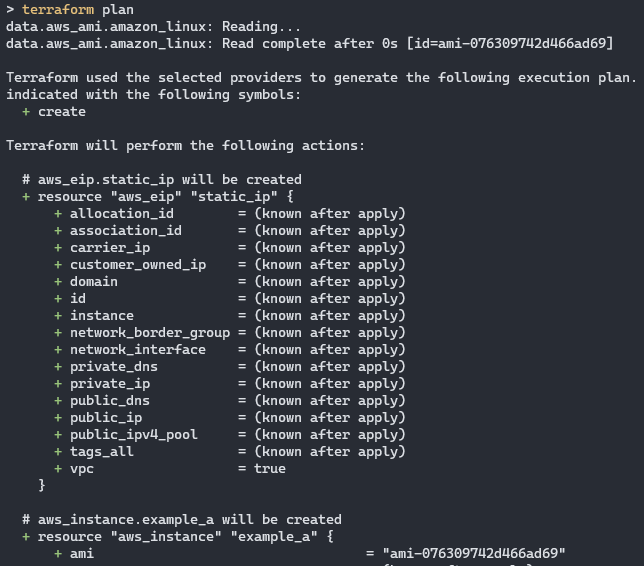
Create a directory named **learn-terraform-dependencies** and write configuration into **main.tf**

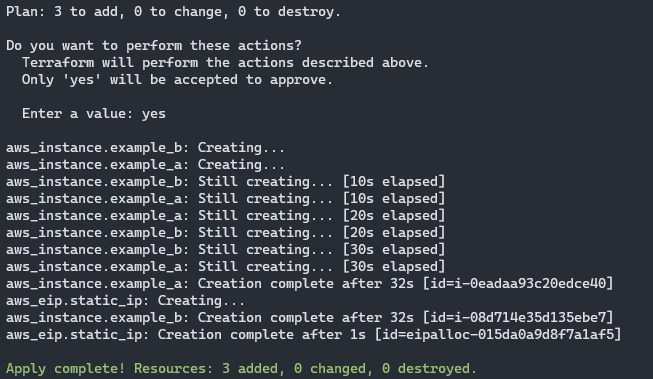
The **aws\_eip** resource type allocate and associates an **Elastic IP** to **EC2** instance. Instance must exist before the **Elastic IP** can be created and attached, Terraform musr ensure that **aws\_instance.example\_a** was created before it creates **aws\_eip.static\_ip.** Meanwhile, **aws\_instance.example\_b** can be created in parallel to the other resources.

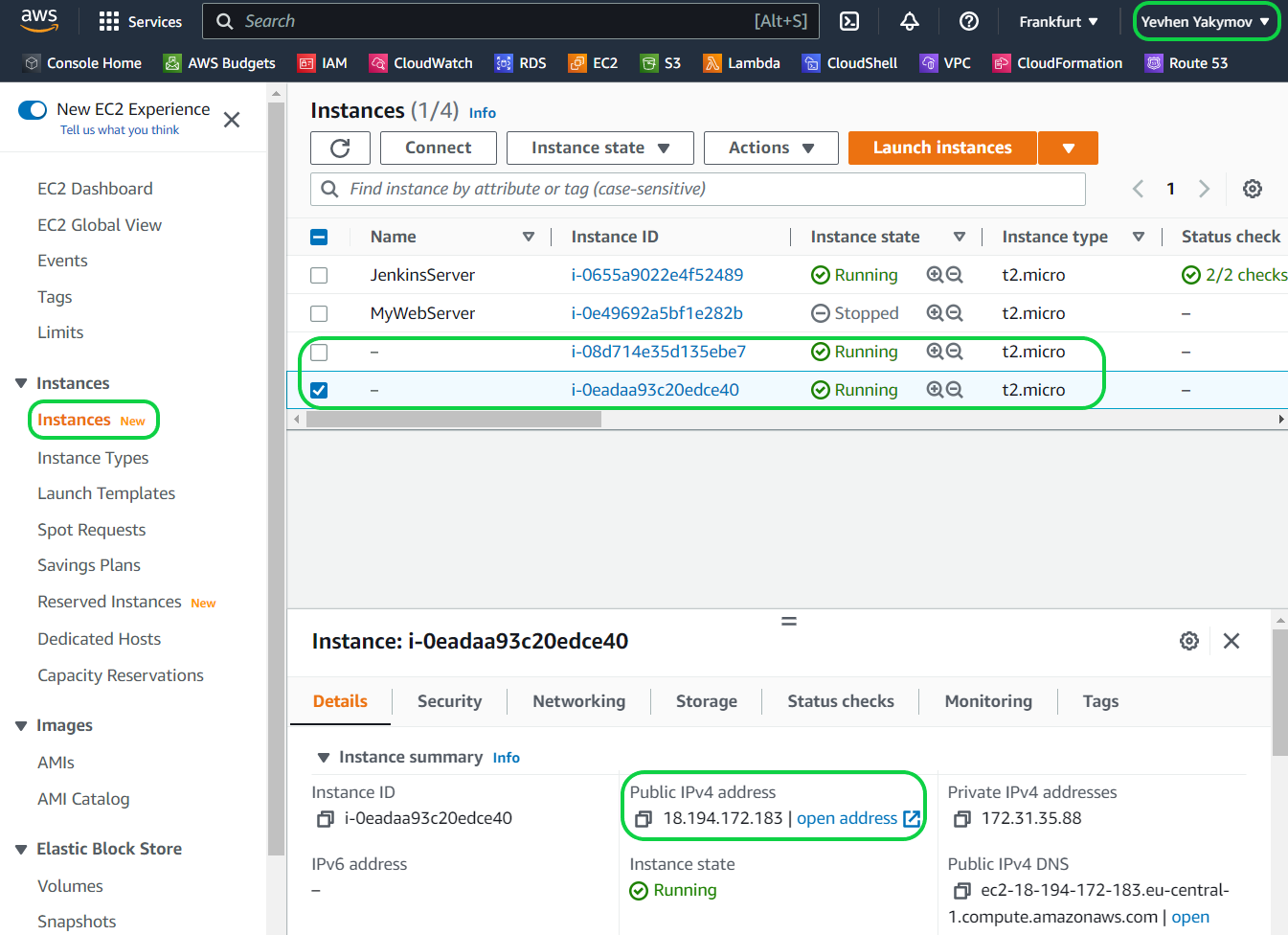
****

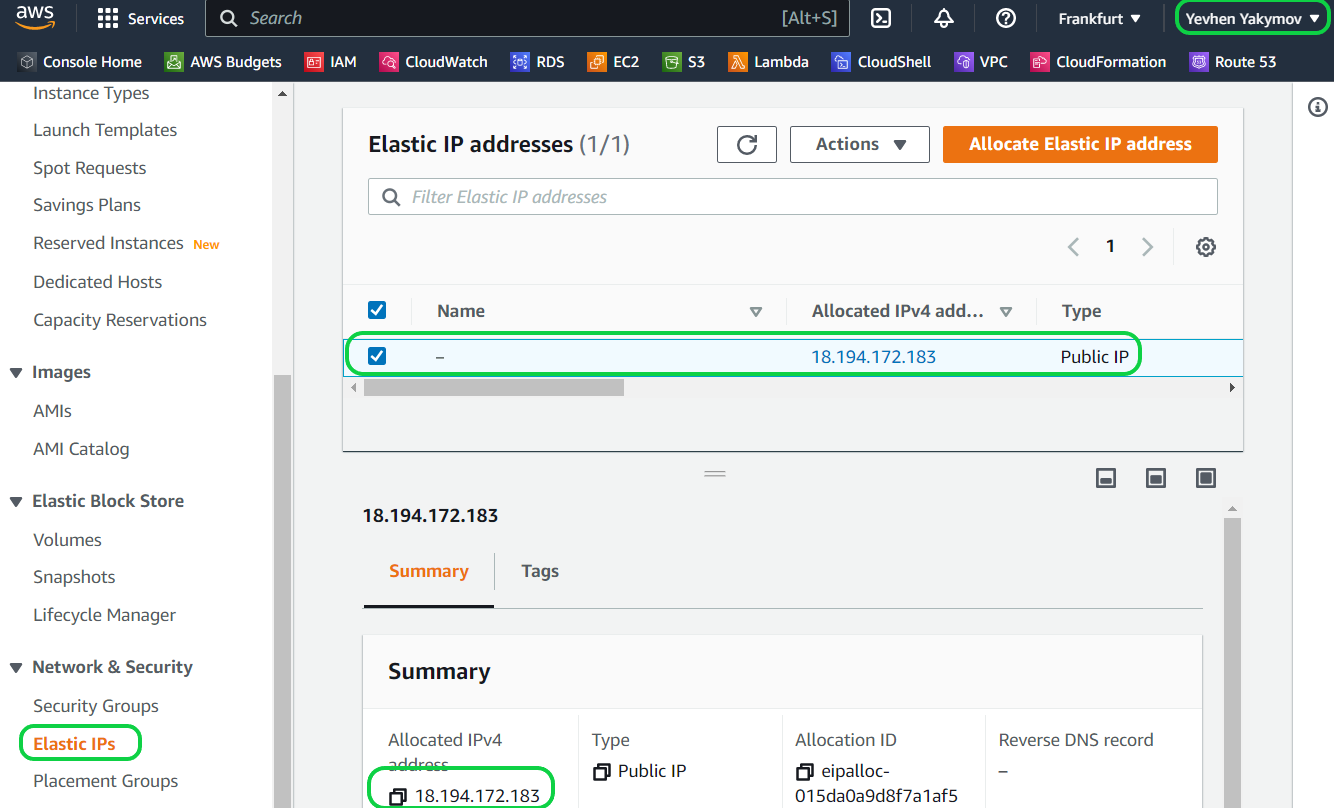


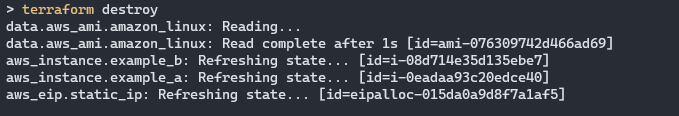


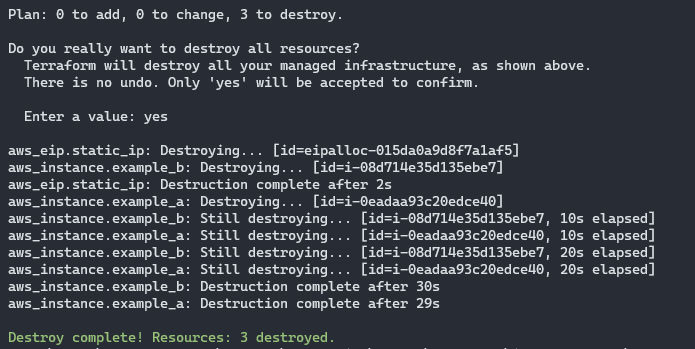




****

****





1. Terraform explicit dependencies example.

The **depends\_on** argument is acepted by any resource or module block and accepts a list of resources to create explicit dependencies for.

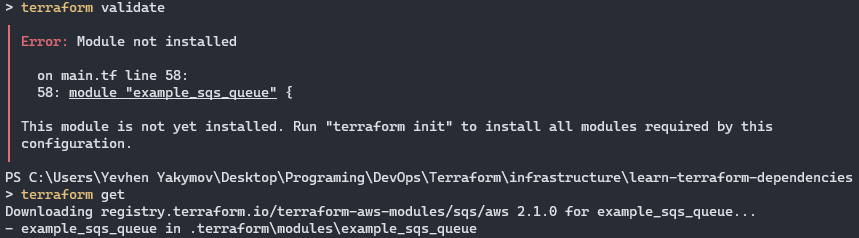
Assume we have an app running on your **EC2 instance** that expects to use a specific **Amazon S3 bucket**. This dependency is configured inside the application, and not visible to Terraform. You can use **depends\_on** to explicity declare the dependency. You can also specify multiple resources in the **depends\_on** argument, and Terraform will wait until all of them have been created before creating the target resources.

Add configuration into the file **main.tf.** First must be created **Amazon S3 bucket** than **EC2 instance** and at the end **module SQS.**

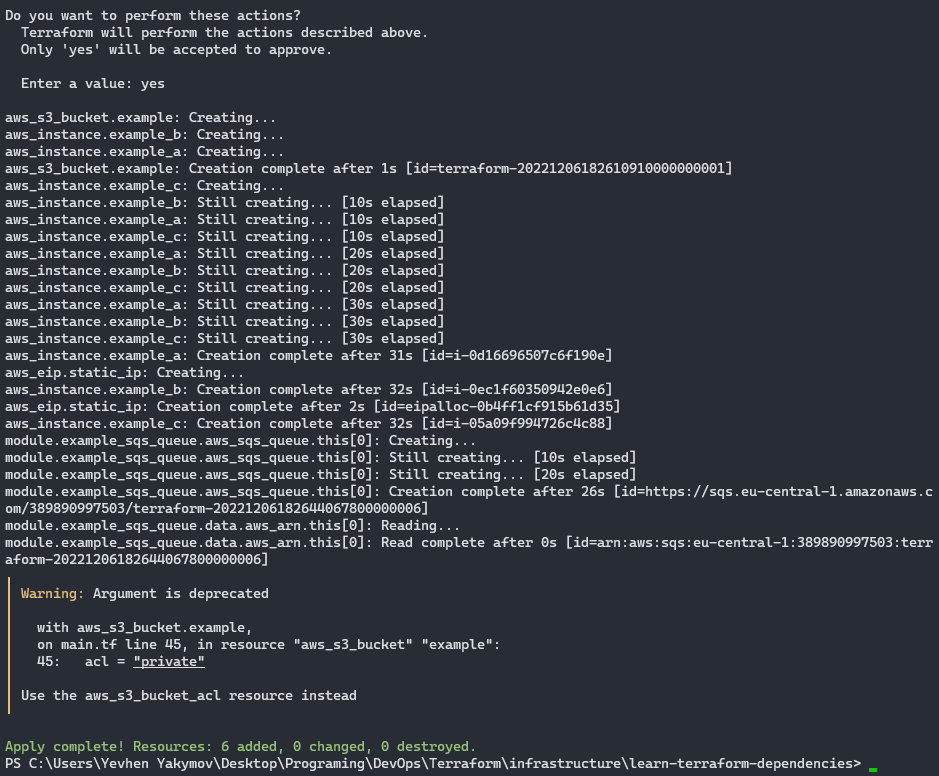


This configuration includes a reference to a new module, **terraform-aws-modules/sqs/aws.** Modules must be installed before Terraform can use them.

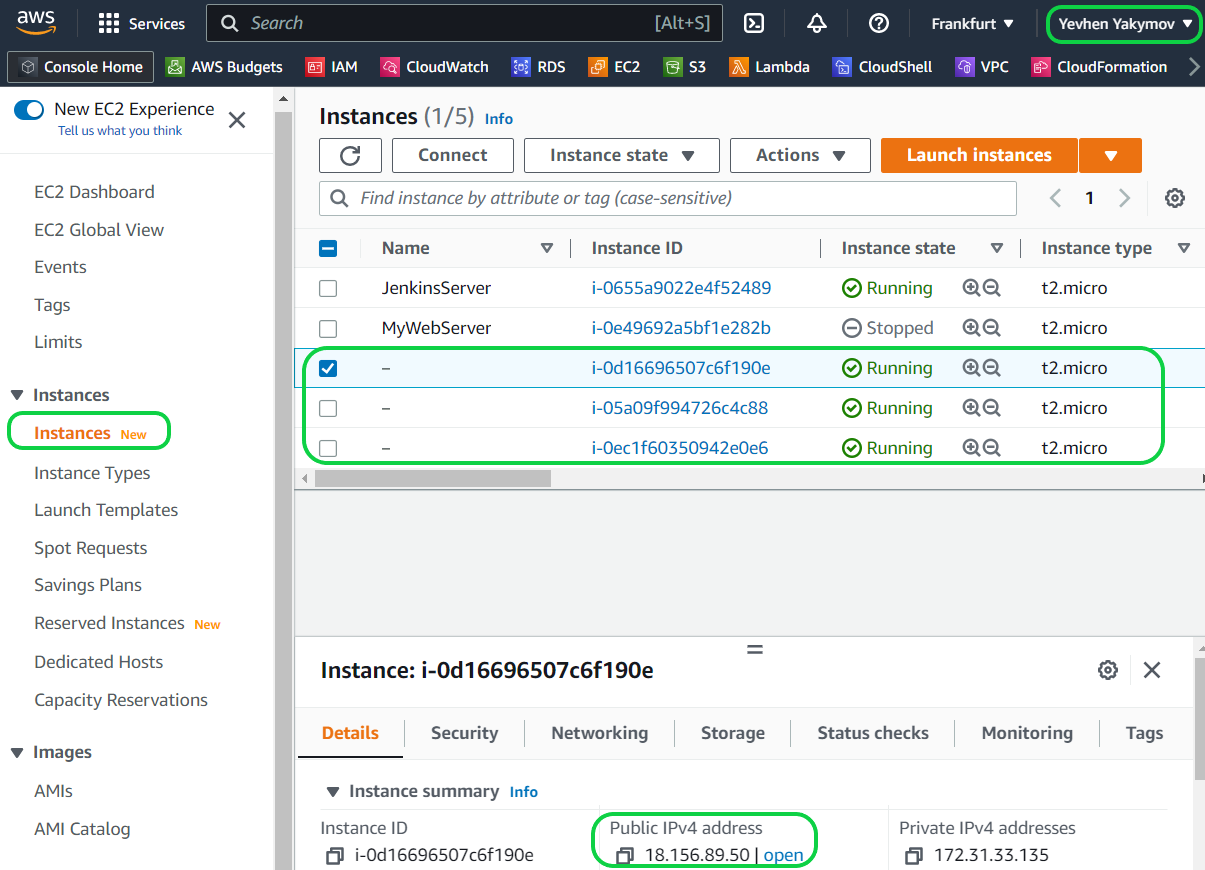
Run **terraform get** to install the module

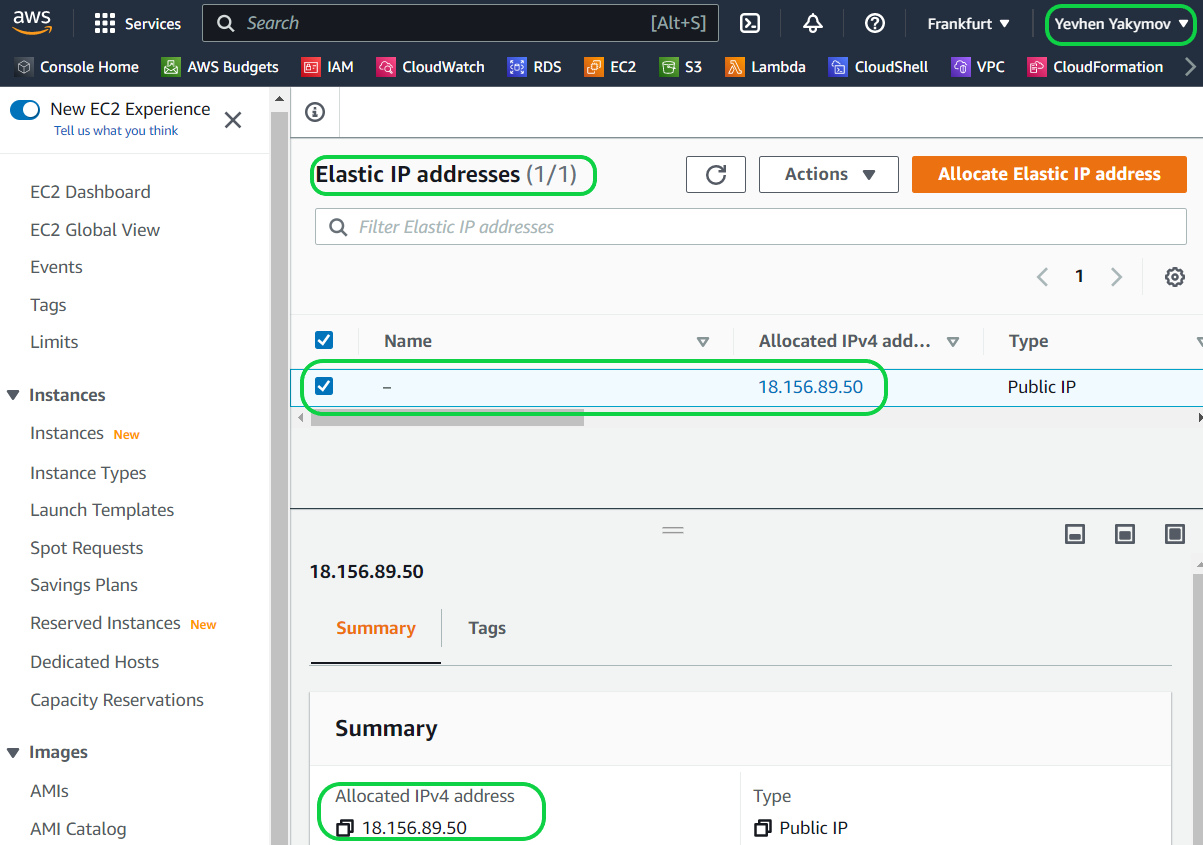


Run **terraform apply**

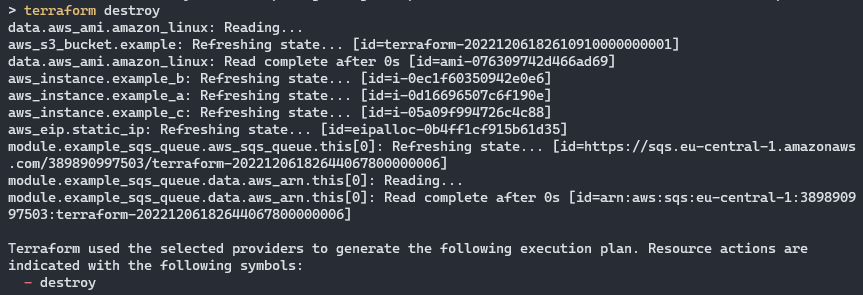


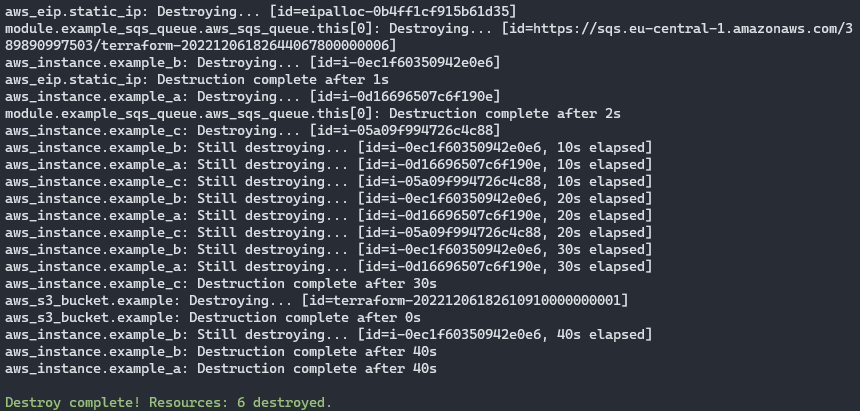
Since both the instance “**example\_c**” and **SQS Queue** are dependent upon the **S3 bucket**, Terraform waits until the bucket is created to begin creating the other two resources.





Run **terraform destroy**





Both implicit and explicit dependencies affect the order in which resources are destroyed as well as created.

1. Terraform Variables.

According our goal, you should have a directory named **learn-terraform-aws-variables** with the following configuration in a file **main.tf.**

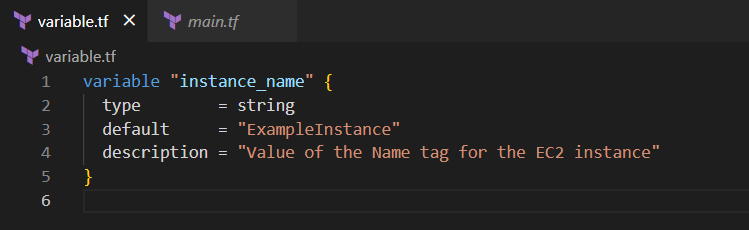


Set the instance name with a variable.

The configuration includes a number of “hard-coded” values. Terraform variebles allow you to write configuration that is flexible and easier to re-use.

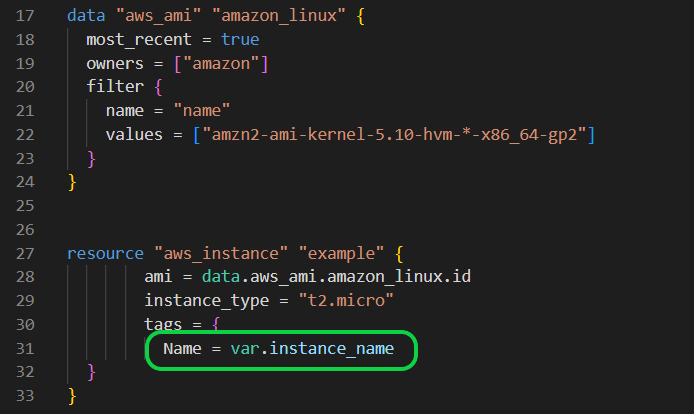
Add a variable to define the instance name.

Creating a new file called **variables.tf** with a block defining a new **instance\_name** variable.



Note: Terraform loads all files in the current directory ending in .**tf** , so you can name your configuration files however you choose.

In **main.tf,** update the **aws\_instance** resource block to use the new variable.



Apply the configuration. Respond to the confirmation promt with a “yes”.

Another one way to set the variable is put in command line with option **-var:**



1. Terraform Output.

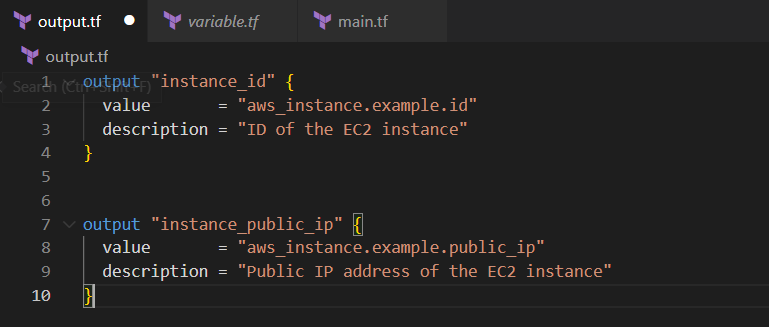
**Output EC2 instance configuration**

According our goal, you should have a directory named **learn-terraform-aws-outputs** with the following configuration in a file **main.tf.**

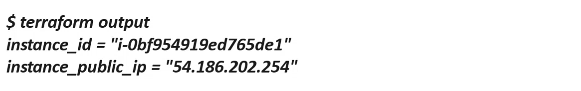


Create a file called **output.tf**  in your **learn-terraform-aws-outputs** directory.

Add outputs to the new file for your EC2 inastance’s ID and IP address.



Terraform prints output values to the screen when you apply your configuration. Query the **outputs** with the **terraform output** command.



You can use Terraform **outputs** to connect your Terraform projects with other parts of your infrastructure, or with other Terrafirm projects.

1. Terraform Modules.

As you manage your infrastructure with Terraform, you will create increasingly complex configurations. There is no limit to the complexity of a single Terraform configuration file or directory, so it is possible to continue writing and updating your configurationfiles in a single directory. However, if you do, you may encounter one or more problems:

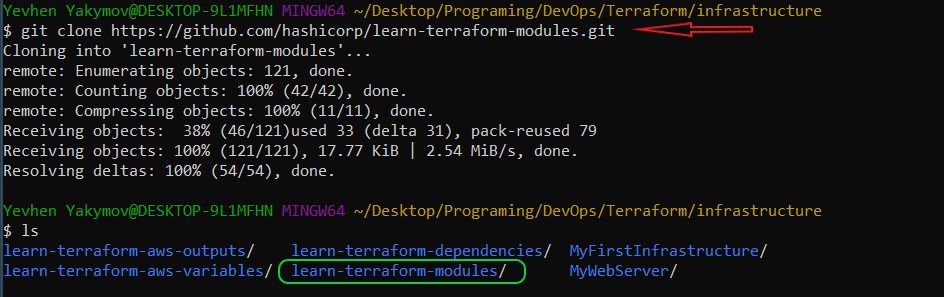
* Understanding and navigating the configuration files will become increasingly difficult
* Updating the configuration will become more risky, as an update to one section may cause unintended consequences to other parts of your configuration
* There will be an increasing amount of duplication of similar blocks of configuration, for instance when configuration separete dev/staging/production environments, which will cause an incresing burdenwhen updating those parts of your configuration
* You may wish to share parts of your configuration between projects and tems, and will quickly find that cutting and pasting blocks of configuration between projects is error prone and hard to maintain.

So main goal of creating and using Terraform modules is to simplify your current workflow.

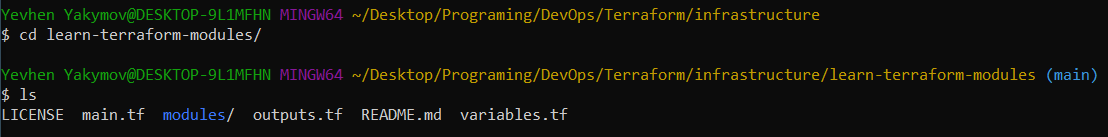
**Create Terraform configuration**

For example, you will use **modules** to create an example AWS envirinment using a VPC and two EC2 instances. You can create it by manualy building the directory structure and files using the following commands to clone this GitHub repo.

Clone the GitHub repository.



Change into that directory in your terminal



Check out the ec2-instances tag into a local branch





Configuration in a file **main.tf.**

