**Variables in Terraform**

**Variables**: Variables in Terraform parameterize configurations, enabling reusability and flexibility for managing infrastructure across multiple environments.

**Types of variables:**

* Input variables
* Output variables
* Local variables

**Input variables**:

Input variables in Terraform allow users to specify dynamic values when creating or managing their infrastructure, enhancing configurability and reusability.

* **Declaration**: Input variables are declared using the variable block, which defines their name, type, and default values.

Example:

variable "region" {

type = string

default = "us-west-1"

}

**Data Types**: Input variables can hold data of different data types, including primitive types (like string, number, and bool) and collection types (like list, map, object, and tuple). This flexibility allows for structured and organized variable definitions tailored to specific use cases.

### **Types of Input Variables:**

1. primitive types
2. complex types

**Primitive Types (or) Simple:**

* + **String** – Represents a simple text value.
  + **Number** – Represents a numeric value
  + **Boolean** – Represents a true/false value.

**String** – Represents a simple text value.

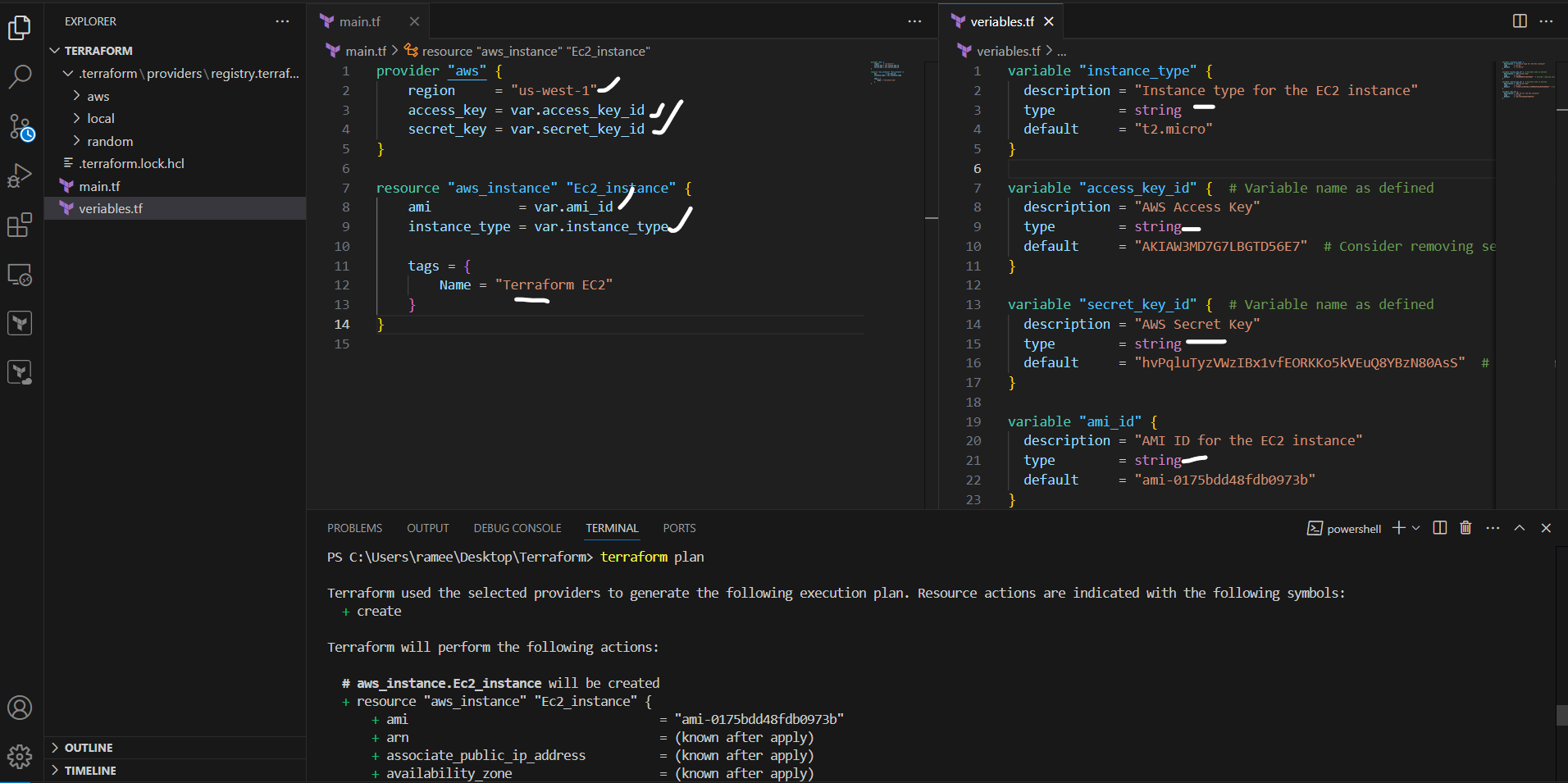
###### For defining **variable block** you need

1. description : Small or short description about the purpose of the variable
2. type : What type of variable it is going to be ex - string, bool, number ...
3. default : What would be the default value of the variable

* Here I am created main.tf file and veriables.tf.
* In the main.tf I am calling through variable’s.
* Make sure after creation files you need to save.
* In terminal perform the below commands.

CMD: terraform init

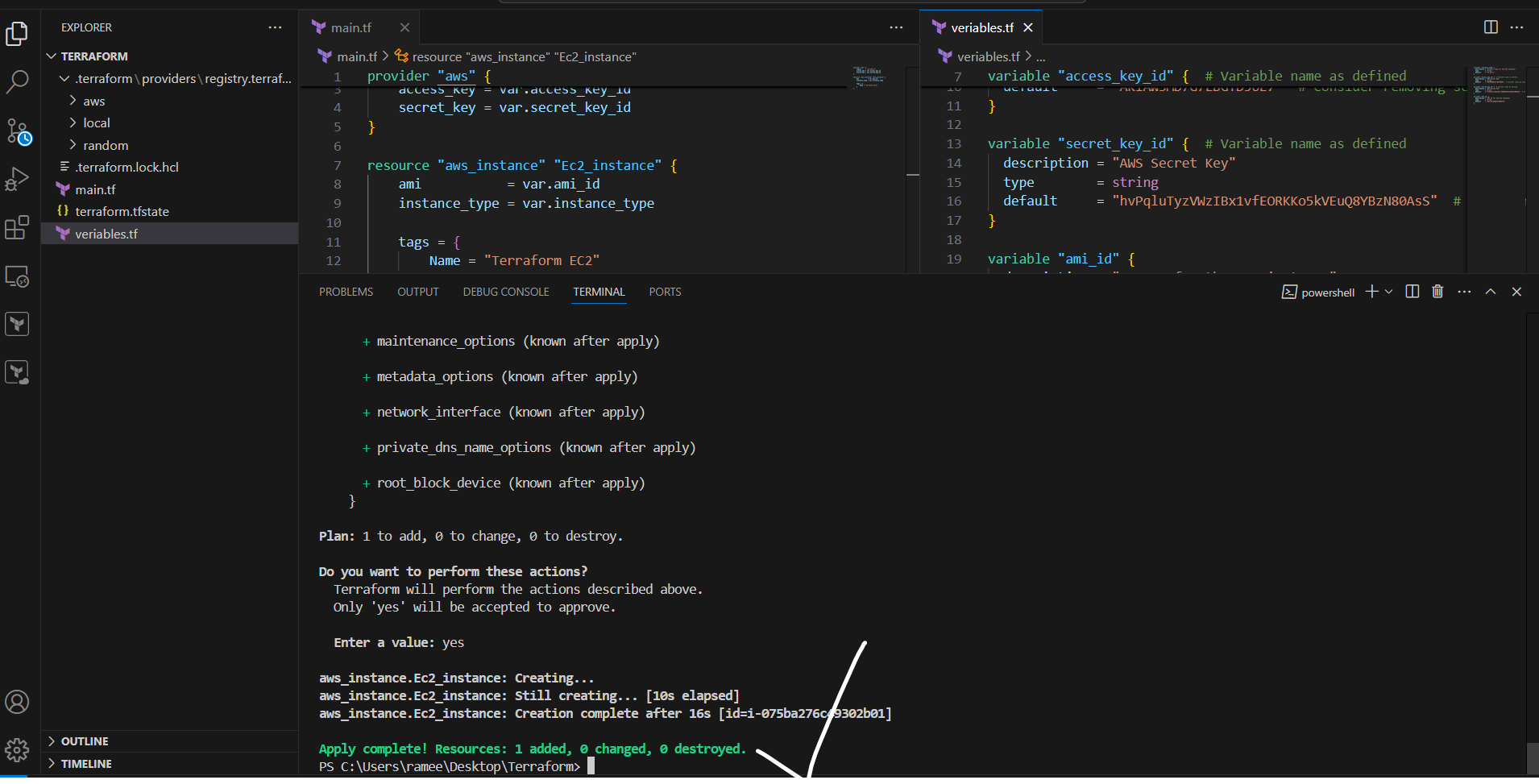
CMD: terraform plan



My Plane is successful ---

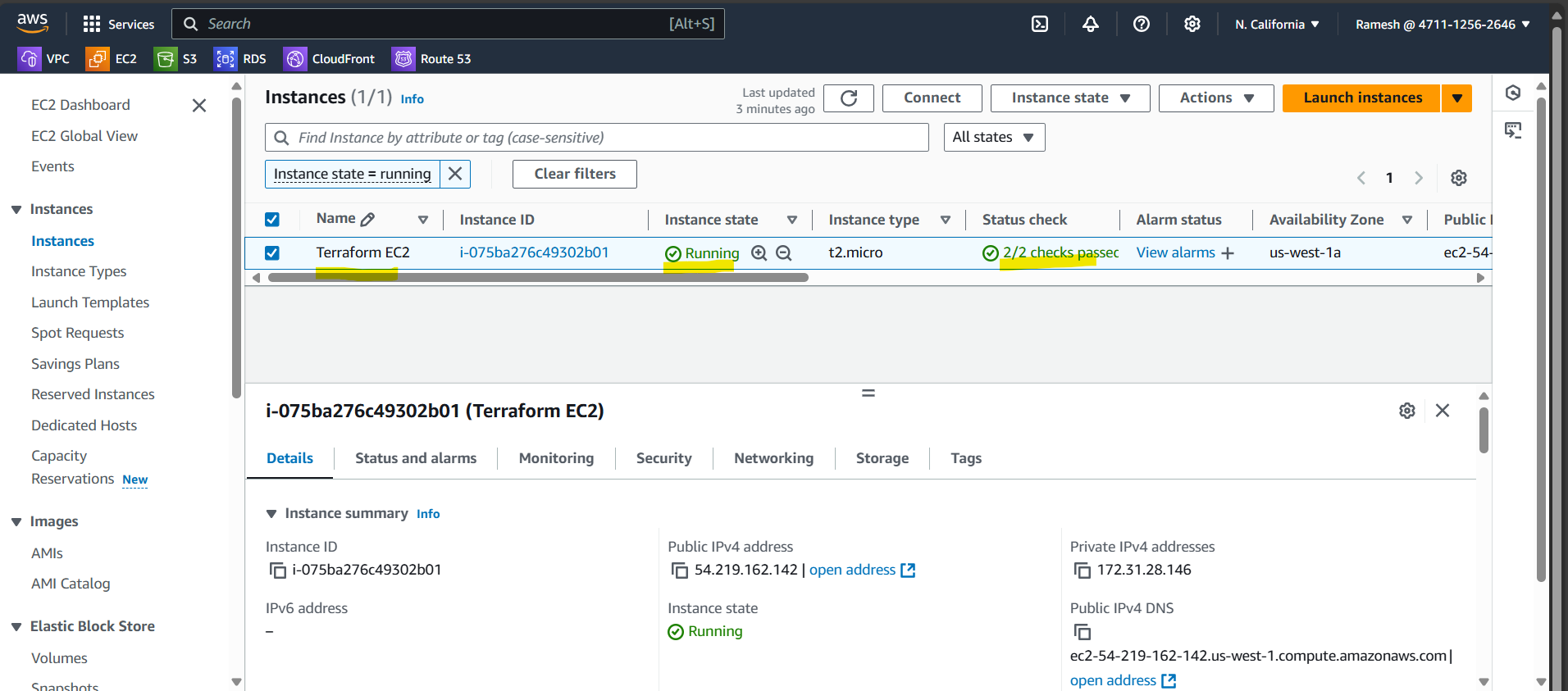
I am ready to create ec2 instance in AWS.

CMD: terraform apply ----



Now I went to the Aws console.

Now our Ec2 instance is created with same ami\_id and instance type.



**Number** – Represents a numeric value

The next variable type we are going to take is number.

For example, we are going to increase the instance\_count of the ec2\_instances.

Variables.tf ---- variable "instance\_count" {

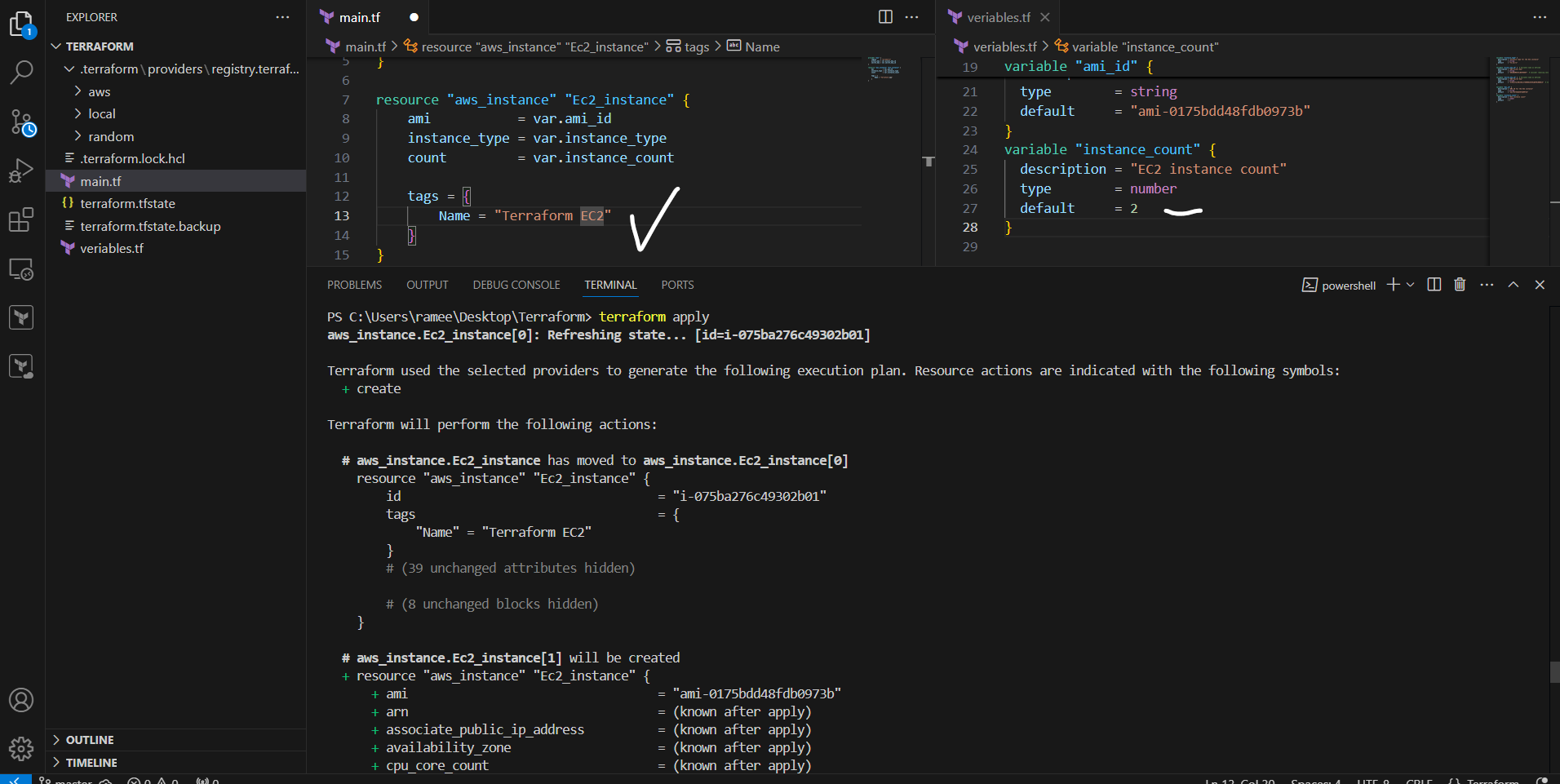
  description = "EC2 instance count"

  type        = number

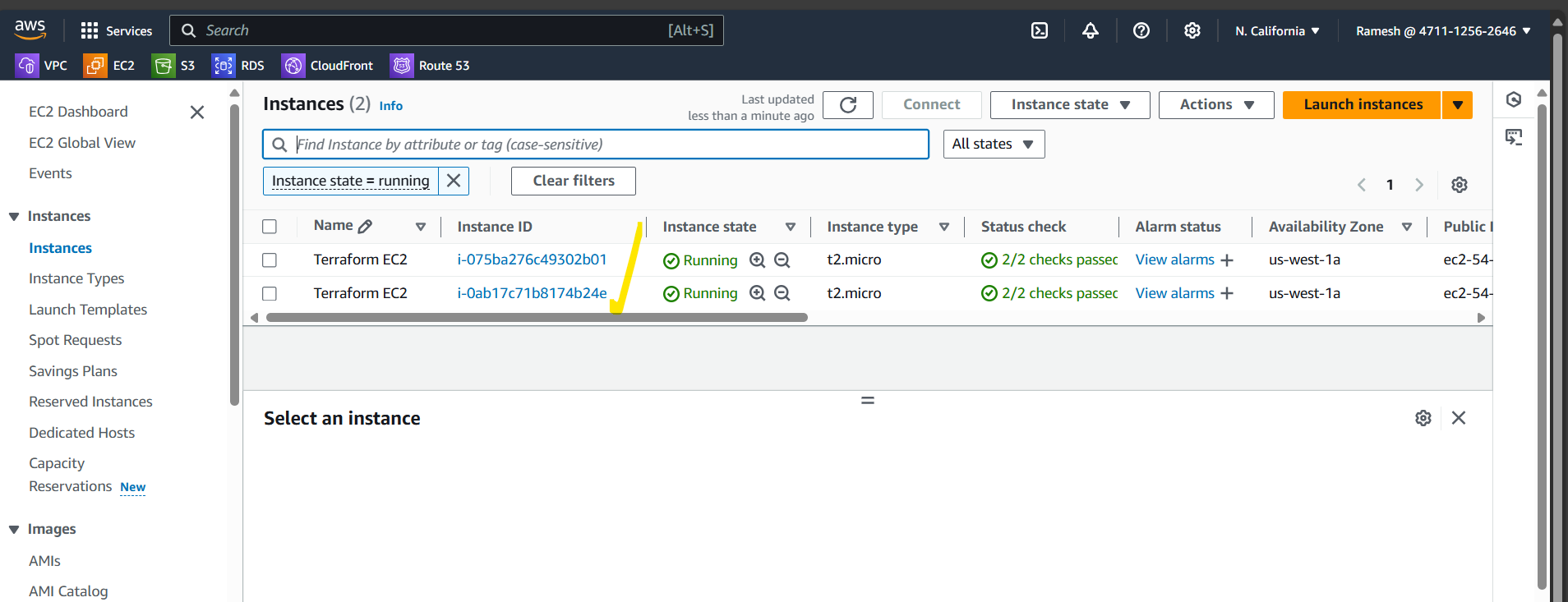
  default     = 2

}

* After Save the variable’s.tf file. Just here adding in main.tf ----count = var.instance\_count
* After this save the file’s and enter the command in terminal terraform apply.



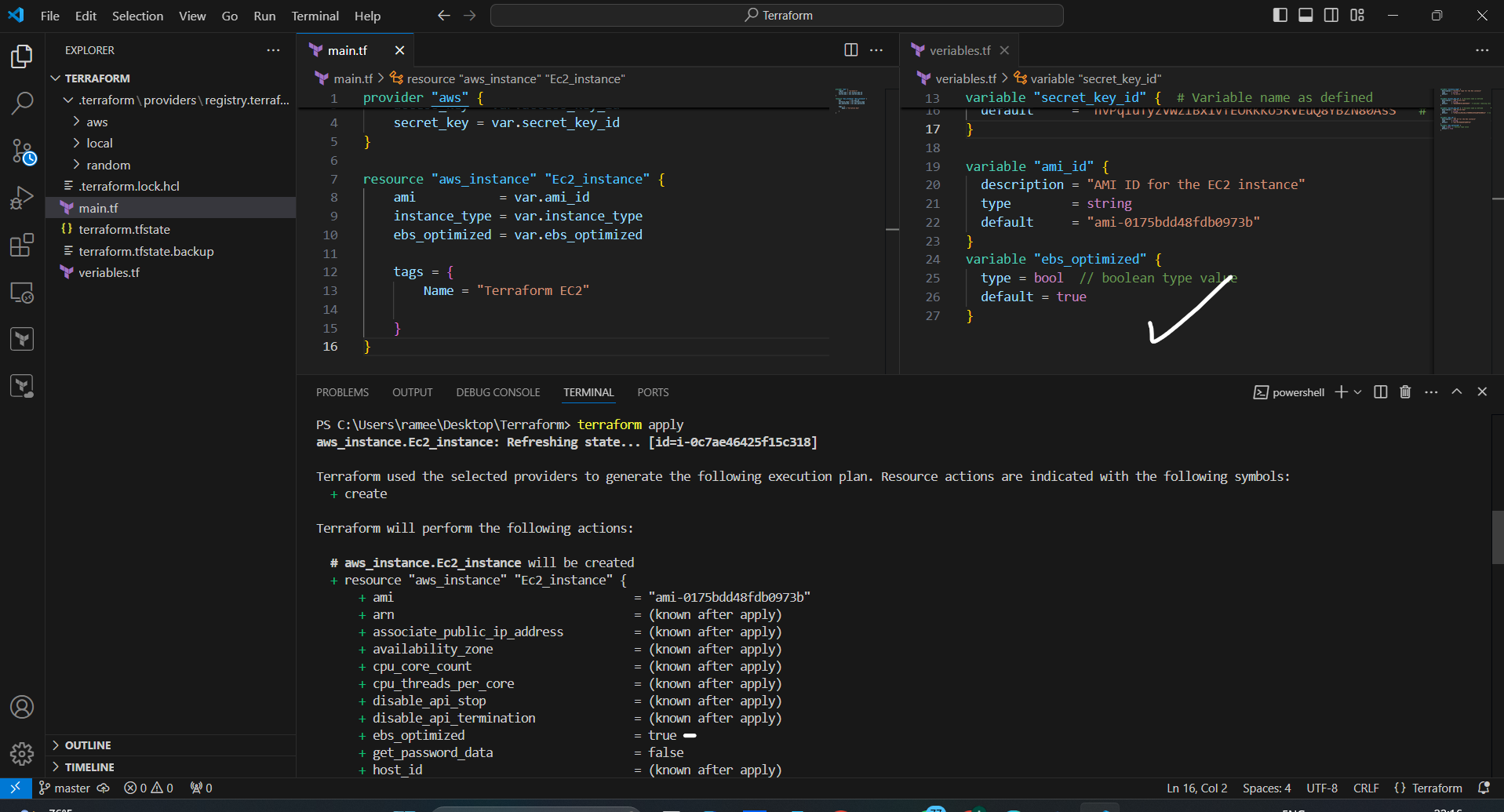
Now you will able to see the Two instances in Aws console.



**Boolean** – Represents a true/false value.

The next variable type which we are going to discuss is bool.

The bool variable can be used to set true or false values inside your terraform file.



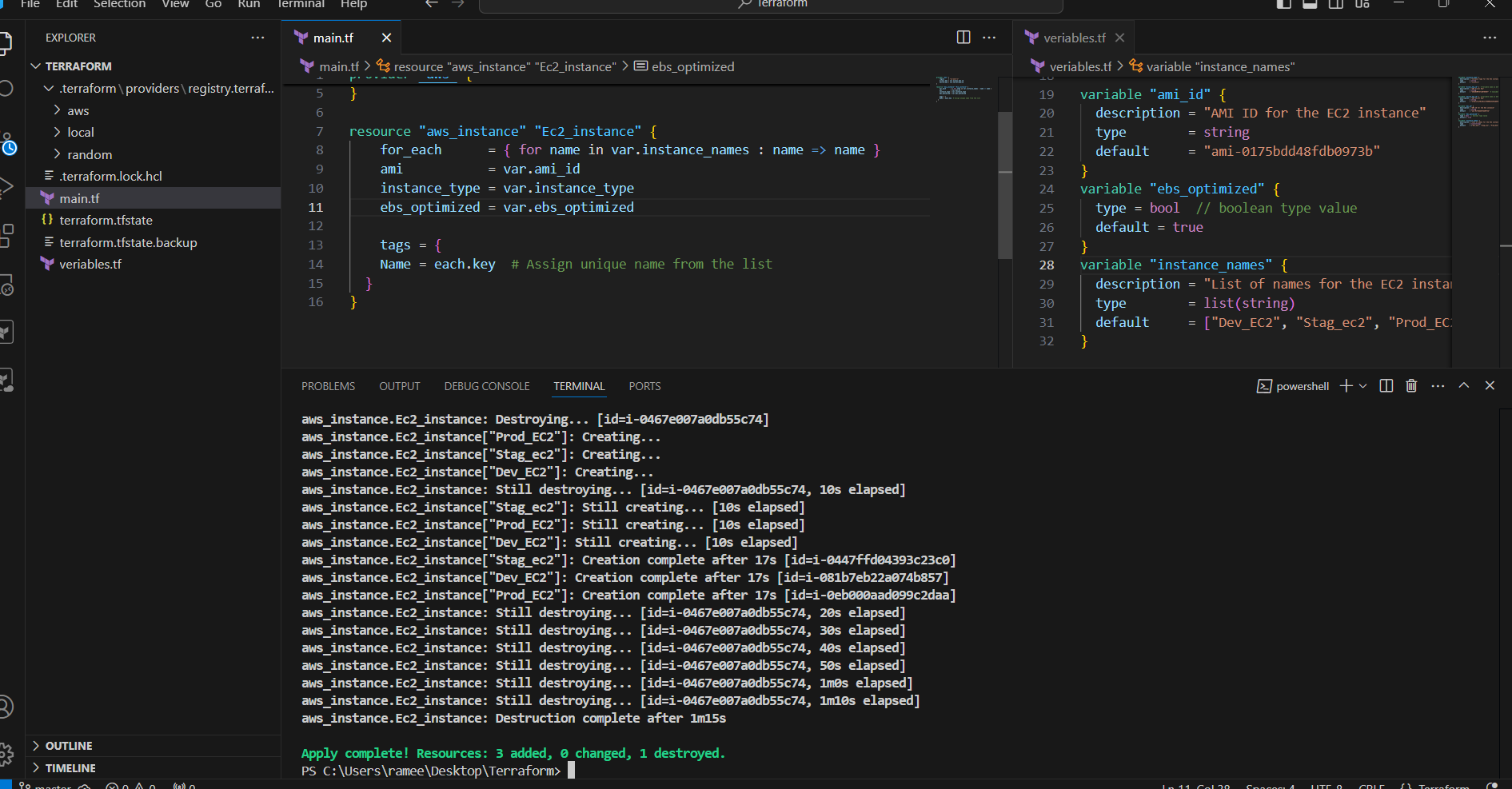
**2. Complex or collection** **types**

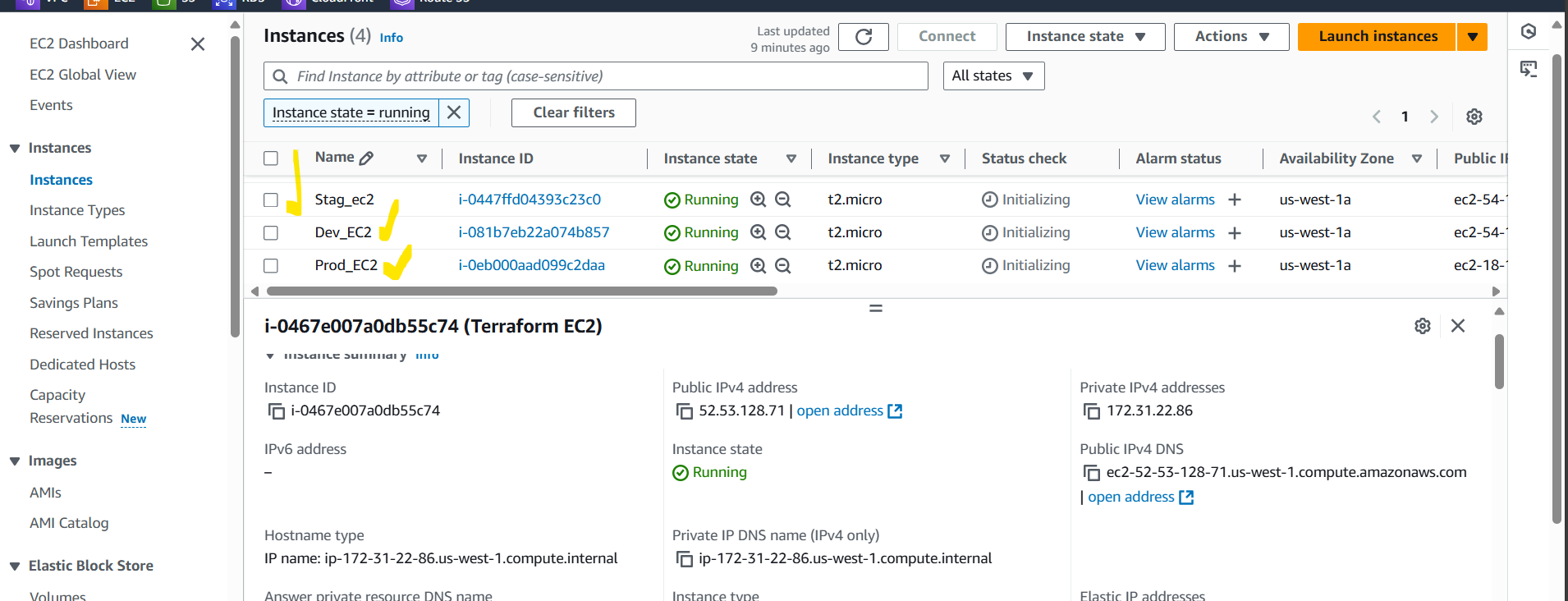
In the collection variable, it consists of -

* List
* Map
* Set

**List:** As the name suggests we are going to define a list that will contain more than one element in it.

Here using list I am created 3 instances with different names.

****

****

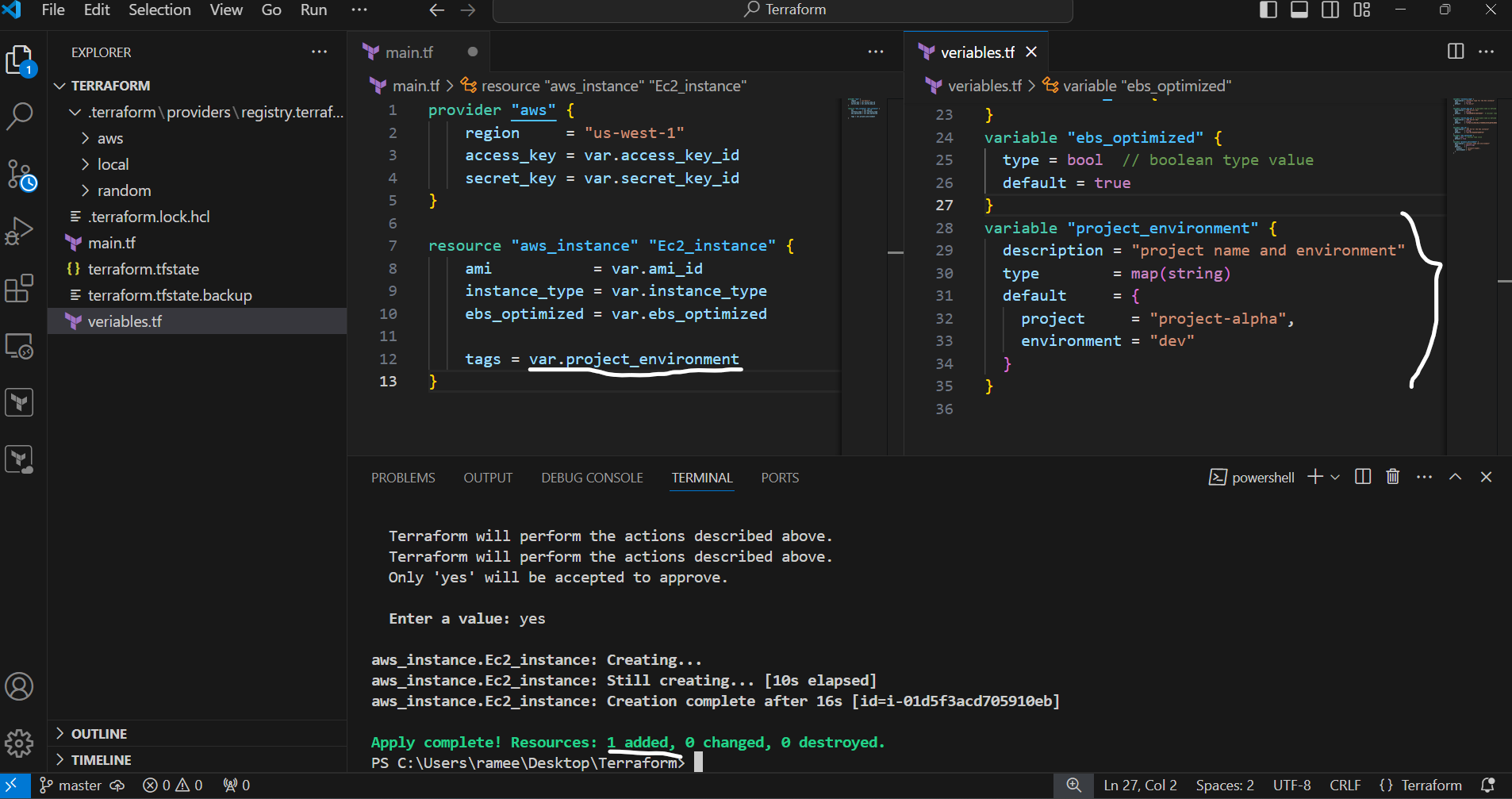
Done.

**MAP**

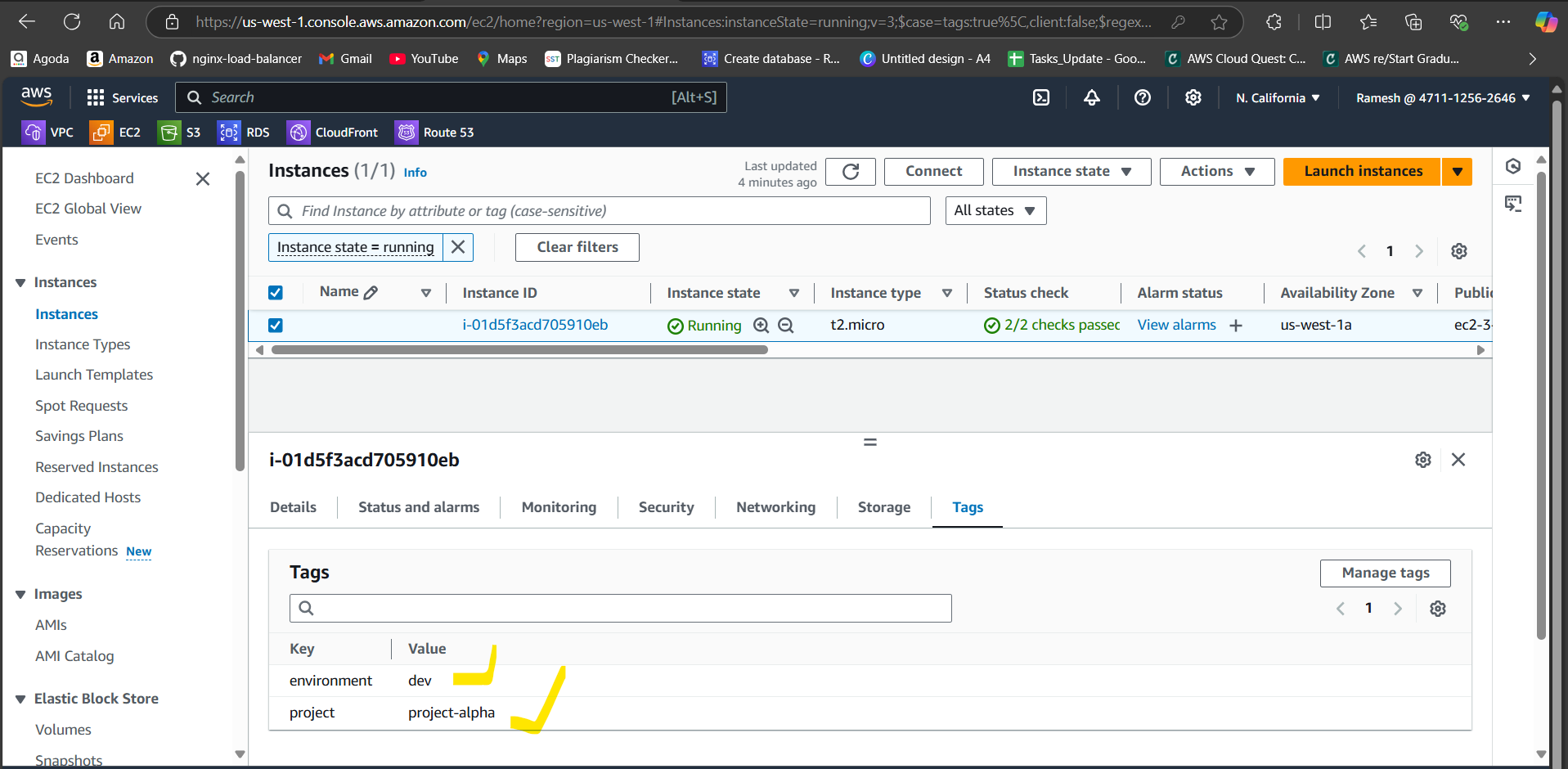
Terraform also supports the map variable type where you can define the key-valye pair.

Let's take an example where we need to define project and environment, so we can use the map variable to achieve that.

Here we created Project environment variable there I am defined project name and environment.



Here our instance is created and tags also attached to the instance.



Done.

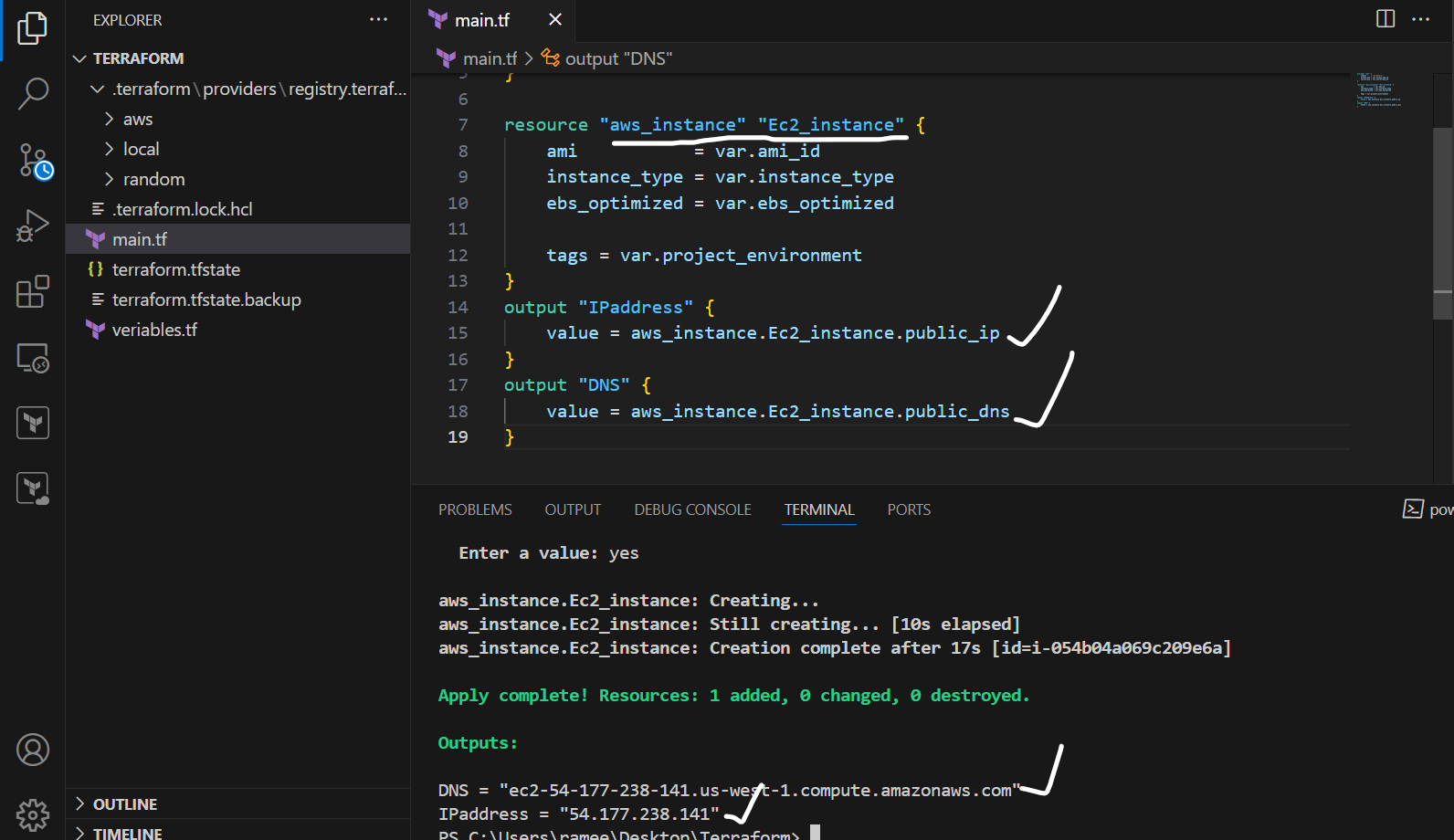
**Output variables:**

In Terraform, we use output variables to get important information like IP addresses or database credentials. After deployment, I run terraform output to see all outputs or terraform output.

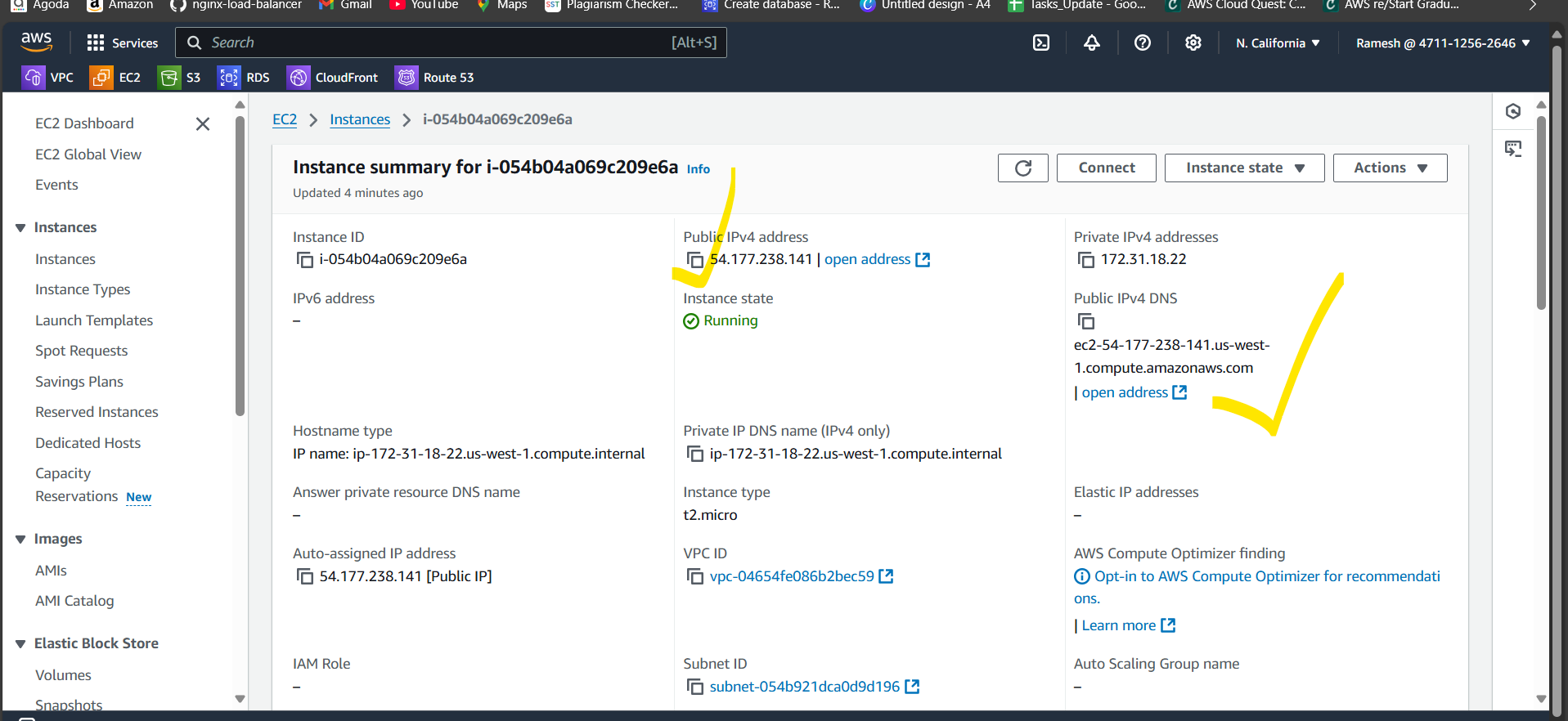
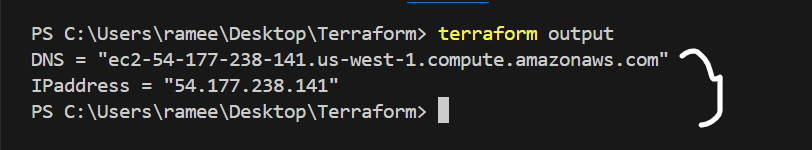
In main.tf file I am created two output block’s and name’s as IPaddress and DNS.

Value = < give here resource name.logical name.public\_ip>

Value = < give here resource name.logical name.public\_dns>

****

Apply will successful we got dns and IPaddress also once we have to cross check.

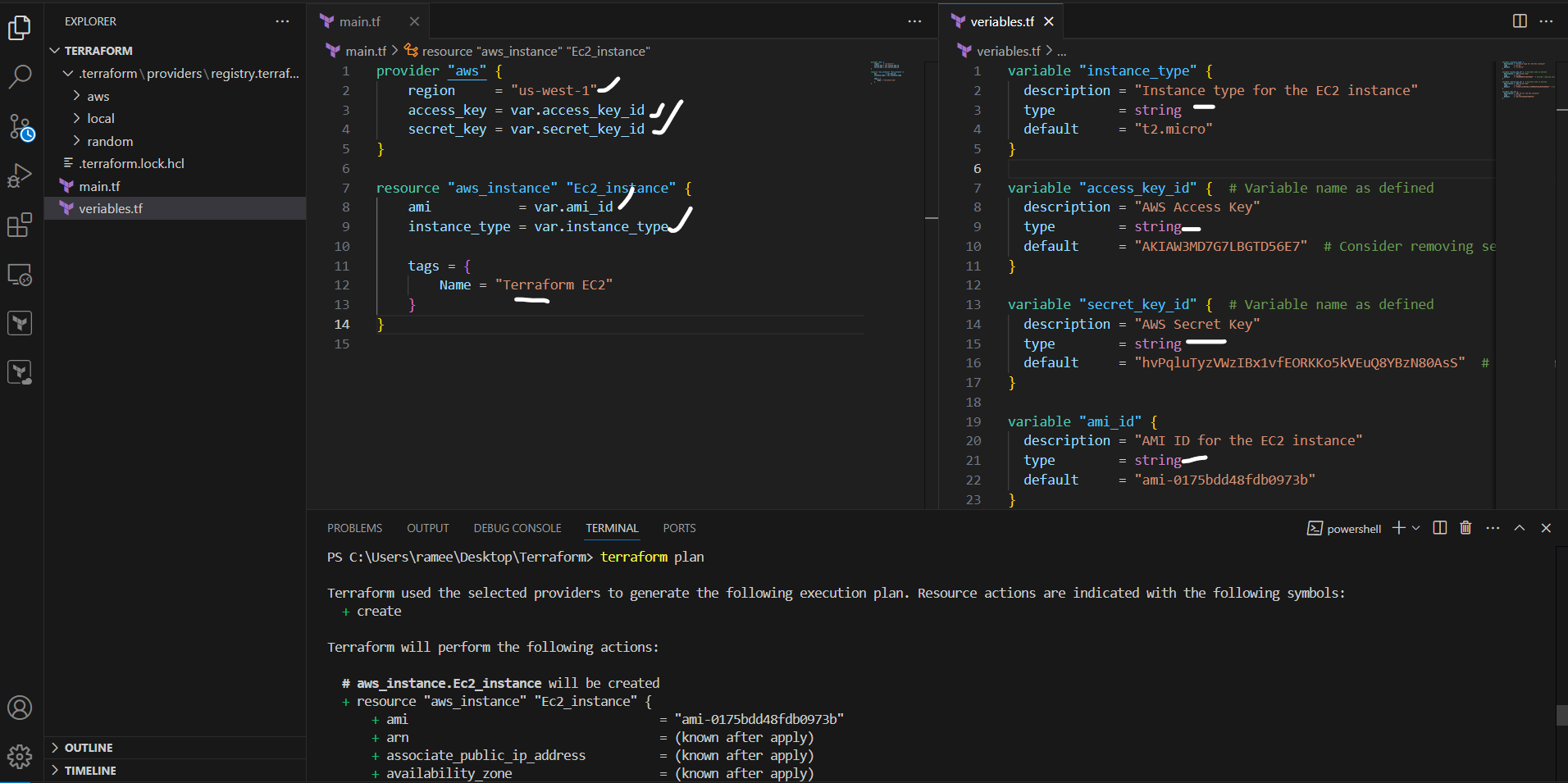
Both are same.

### **Using Variables in Terraform:**

**Using variables.tf File:**

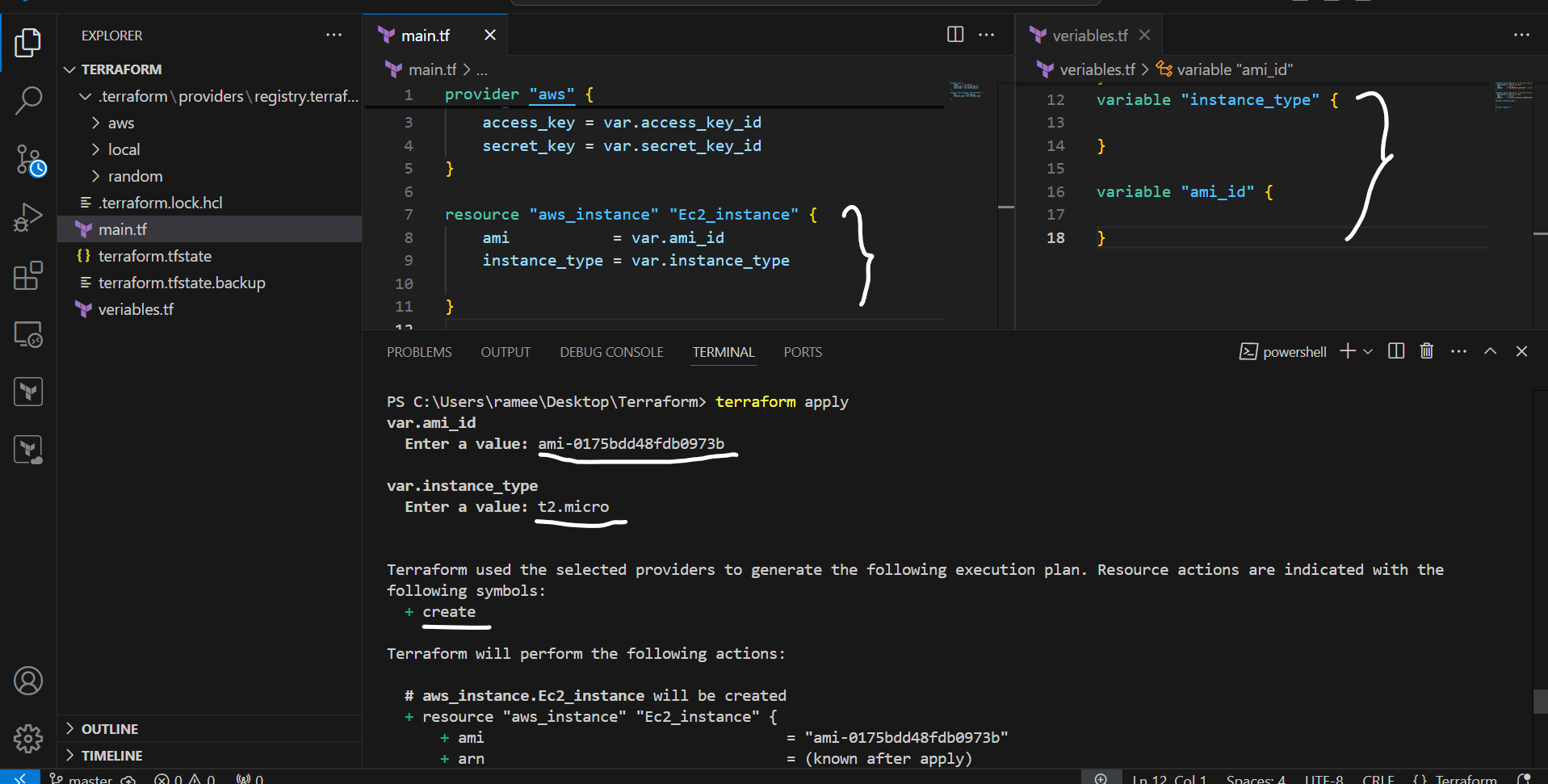
Define variables inside a .tf file to make configurations dynamic.

Here we have to declared variable and default value.



**Interactive Mode:**

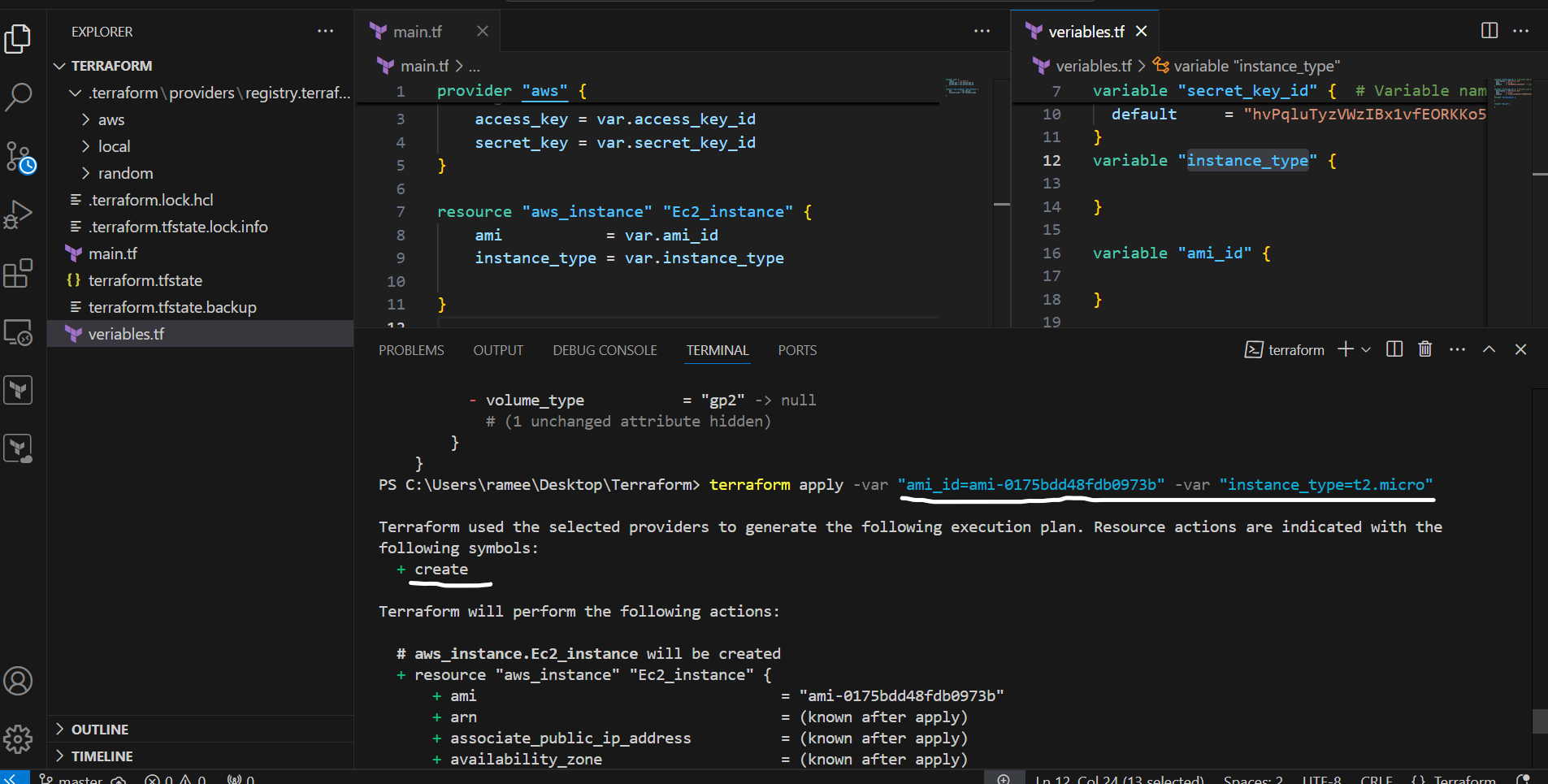
If a variable is declared without a default value, Terraform will prompt for it during execution.

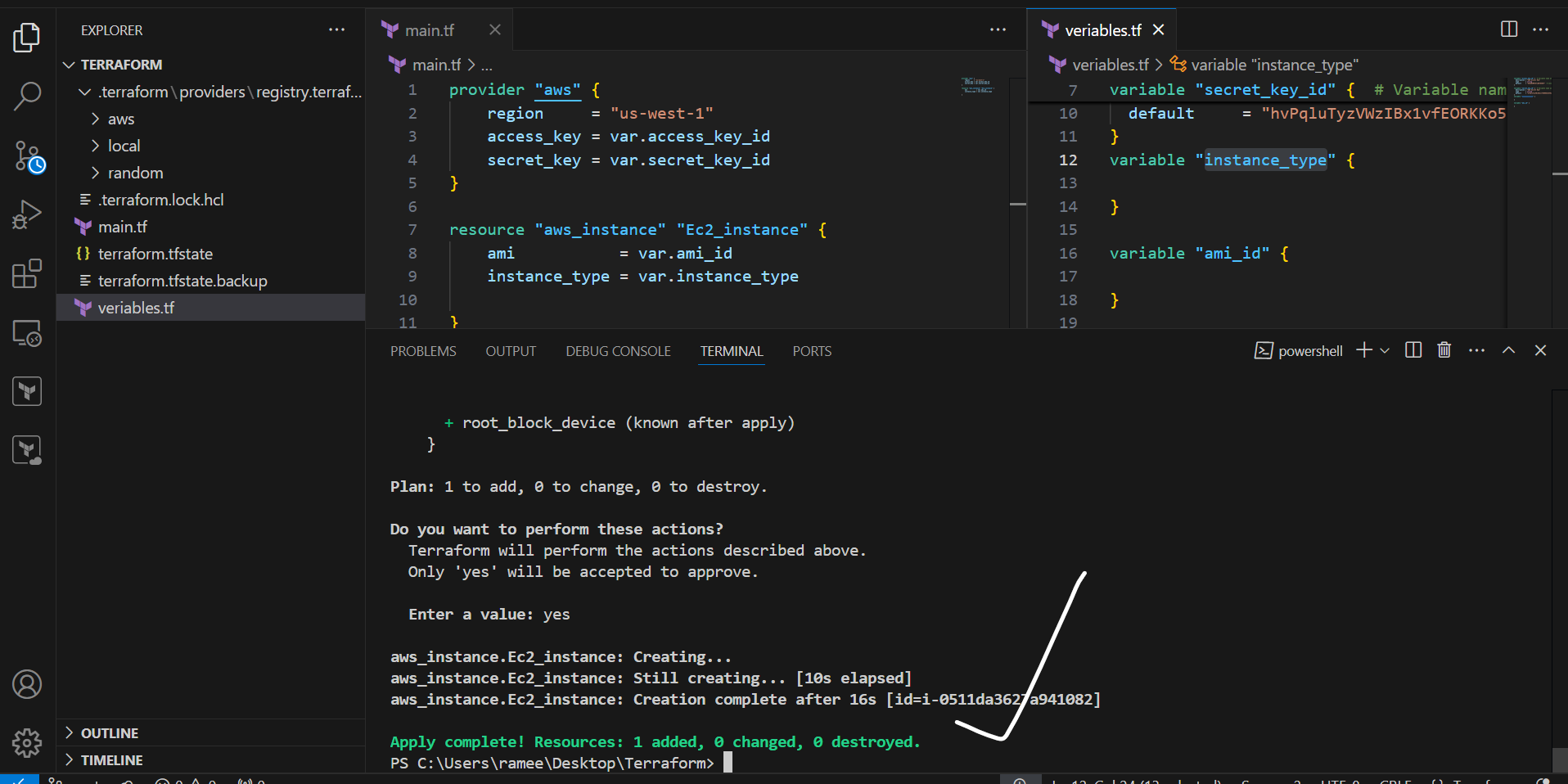


**Command-Line Flags:**

Pass variables directly via CLI using -var:

It’s taken from the CLI directly.



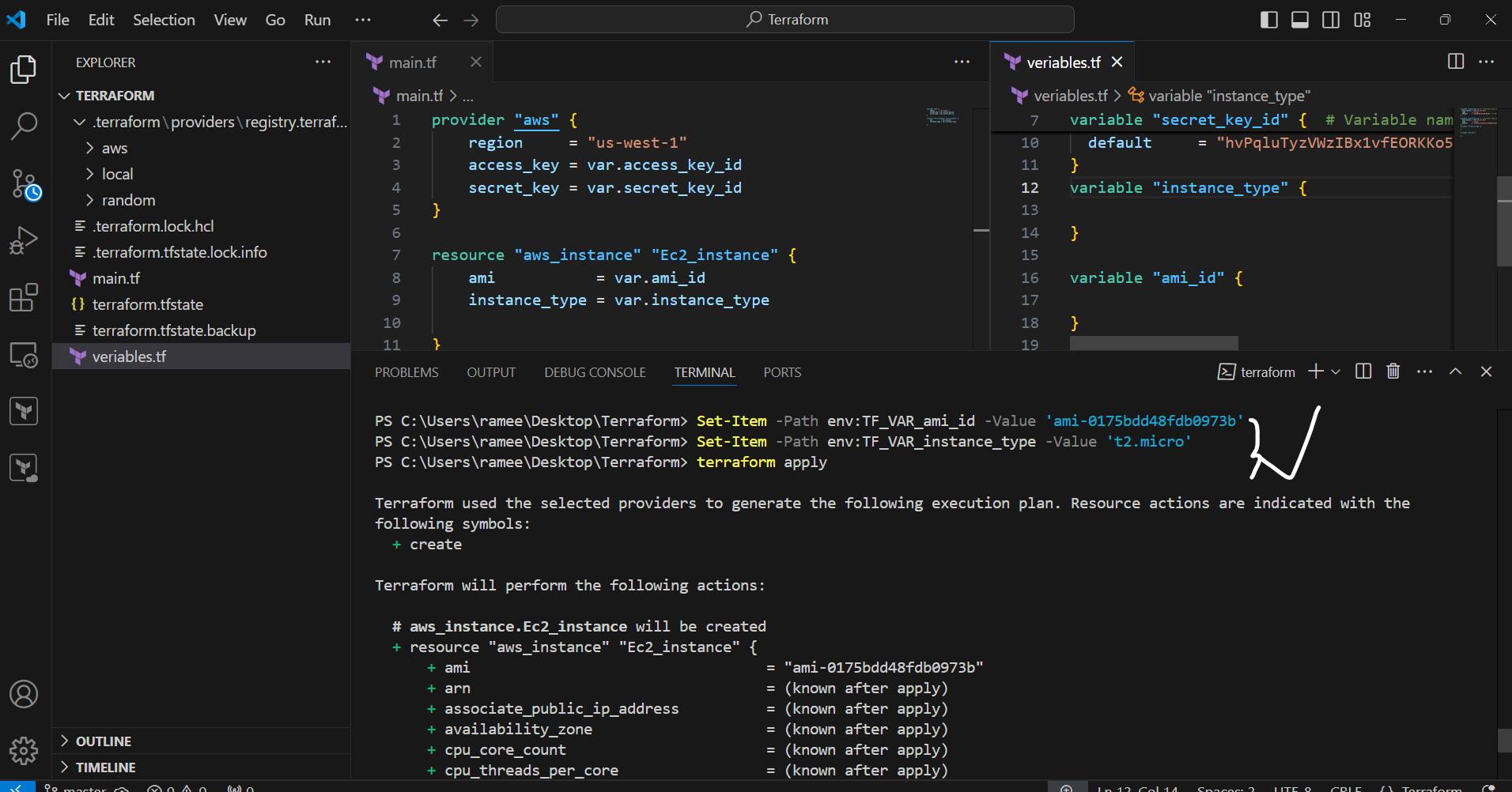


**Environment Variables:**

In Terraform, you can set variables using environment variables by adding the TF\_VAR\_ prefix. This helps keep sensitive information secure and makes it easy to change settings without modifying your code.

Set variables as environment variables using TF\_VAR\_ prefix:

* Set-Item -Path env:TF\_VAR\_ami\_id -Value 'ami-0175bdd48fdb0973b'
* Set-Item -Path env:TF\_VAR\_instance\_type -Value 't2.micro'



**Variable Definition Files:**

Files like terraform.tfvars or terraform.tfvars.json are automatically loaded.

Files ending with .auto.tfvars are also auto-loaded.

If using a custom name, specify the file with the -var-file flag:

4) Create one jenkins job using MAVEN PROJECT for the below code with two stages.

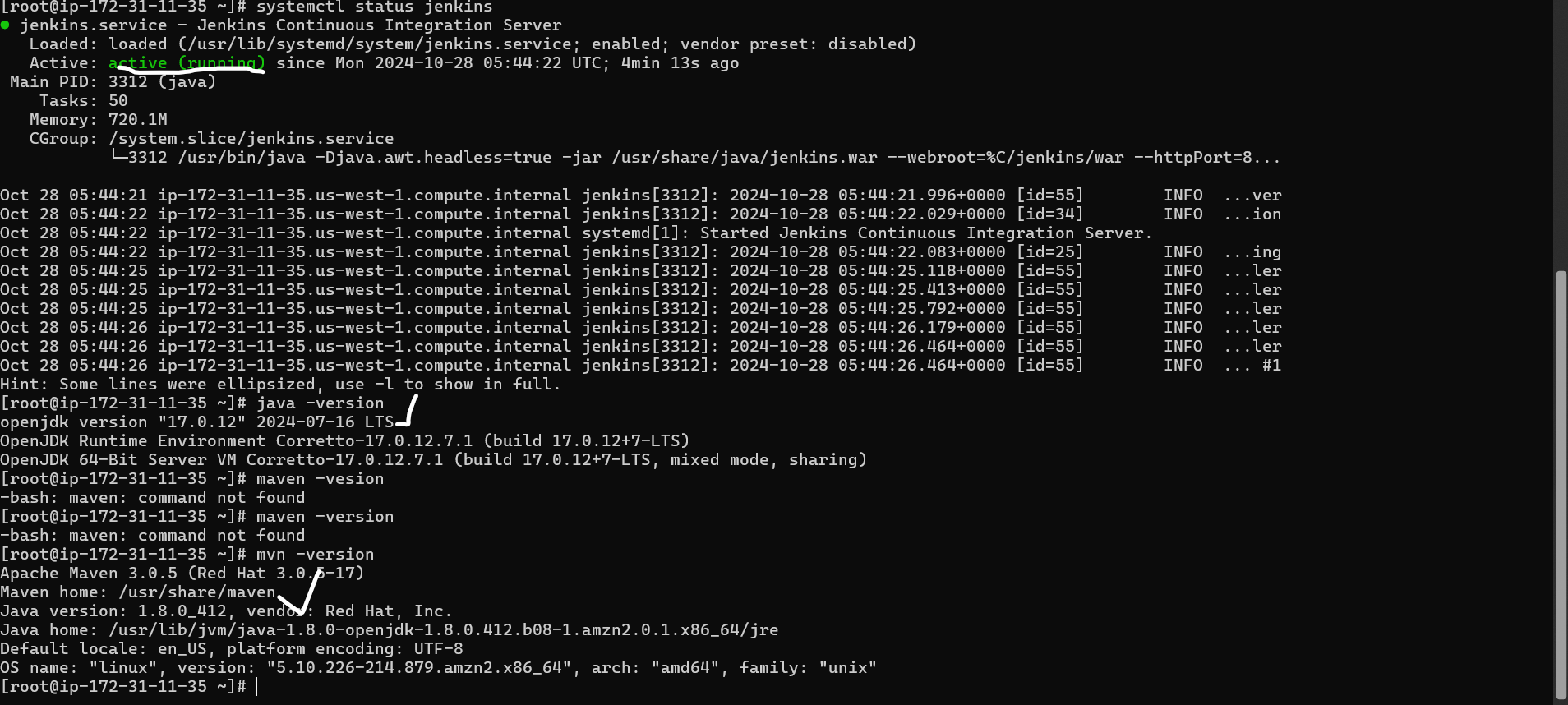
stage 1: Git clone

stage 2: Maven Compilation

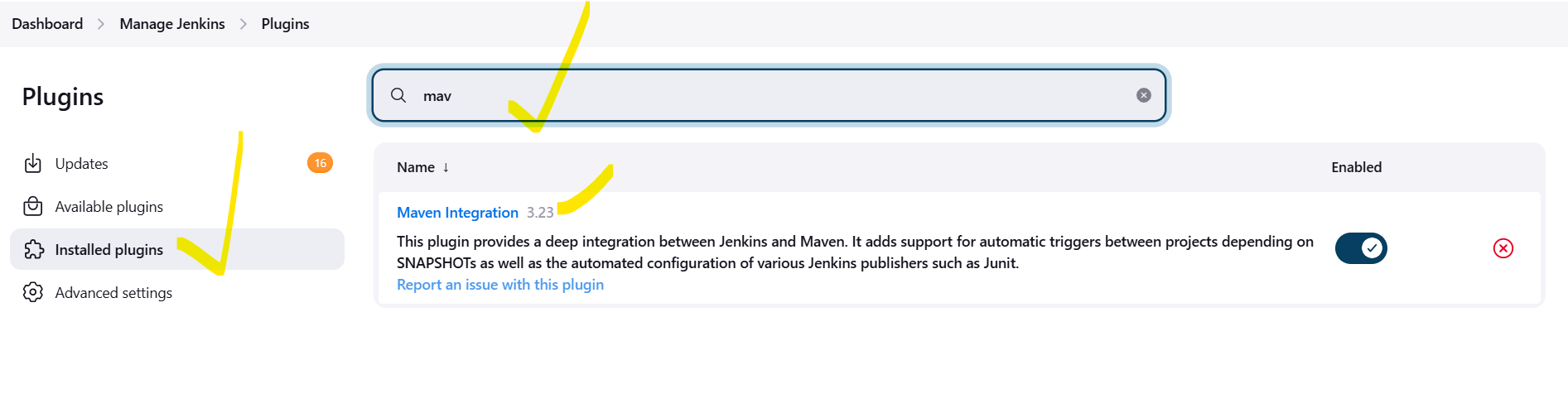
Code: <https://github.com/betawins/java-Working-app.git>

Here to do the task first we need Ec2 instance.

* First connect the sever there you need to install java. CMD: sudo yum –y install java-17\*
* After that you need to install the Jenkins.
* And also you need to install maven.



After that you need install Maven integration Plugin.



After that you need to go manage jenkins > tools configurations

Here you need to give the Maven Path -- /usr/share/maven

Machine generated alternative text:
Agoda Amazon nginx-load-balancer 
Gmail 
69 Edited 
YouTube 
Maps 
Plagiarism Checker... 
Create database - 
Untitled design - 
Tasks Update - 
Goo... 
AWS Cloud Quest C... 
AWS re/Start Gradu... 
Dashboard 
Manage Jenkins Tools 
Maven installations 
Add Maven 
Maven 
Name 
MVN HOME 
MAVEN HOME 
/usr/share/maven 
Install automatically 
Add Maven 
Apply 

* Then you need save and go to the Jenkins Dashboard.
* Just click on the New item and give the item name.
* Select the Maven Project and click on save.

Machine generated alternative text:
Dashboard 
New Item 
New Item 
Enter an item name 
Maven project 
Select an item type 
Freestyle project 
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build 
steps like archiving artifacts and sending email notifications. 
Maven project 
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration. 
Pipeline 
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as 
workflows) and/or organizing complex activities that do not easily fit in free-style job type. 
Multi-configuration project 
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, 
platform-specific builds, etc. 
OK 

* Then you see the below interface.
* Here you select git and give URL of the GitHub repo link and Branch Name.

Machine generated alternative text:
Dashboard 
Configure 
General 
Maven_project 
Configuration 
Git ? 
Repositories ? 
Repository URL ? 
https://github.com/betawins/hiring-app.git 
Credentials ? 
none 
+ Add 
Advanced v 
Add Repository 
Branches to build ? 
Branch Specifier (blank for 'any') 
*/main 
Apply 
p Source Code Management 
(9 Build Triggers 
Build Environment 
@ Pre Steps 
@ Build 
@ Post Steps 
@ Build Settings 
Post-build Actions 

* After that Scroll down
* In the pre steps under the build see option is Root POM this automatically it will selected as pom.xml

* Under the goals and option you need to give clean install to build the image.

Machine generated alternative text:
Dashboard 
Configure 
General 
Maven_project 
Configuration 
Build 
Root POM 
pom.xml 
Goals and options ? 
clean install 
Advanced v 
Post Steps 
Run only if build succeeds 
Run only if build succeeds or is unstable 
Apply 
p Source Code Management 
(9 Build Triggers 
Build Environment 
@ Pre Steps 
@ Build 
@ Post Steps 
@ Build Settings 
Post-build Actions 

After that save and build the Job.

Machine generated alternative text:
Dashboard 
Maven_project 
Console Output 
C JENKINS J Recording test results 
CINFOJ 
CINFOJ 
CINFOJ 
CINFOJ 
CINFOJ 
CINFOJ 
CINFOJ 
CINFOJ 
CINFOJ 
CINFOJ 
Ø.1.war 
CINFOJ 
CINFOJ 
CINFOJ 
CINFOJ 
CINFOJ 
CINFOJ 
CINFOJ 
- maven-war-plugin: 3.3. I:war (default-war) @ hiring - 
Packaging webapp 
Assembling webapp Lhiring) in C/var/1ib/jenkins/workspace/Maven_project/target/hiringJ 
Processing war project 
Copying webapp resources L/var/1ib/jenkins/workspace/Maven_project/src/main/webapp) 
Building war: /var/1ib/jenkins/workspace/Maven_project/target/hiring.war 
- (default-install) @ hiring - 
Installing /var/1ib/jenkins/workspace/Maven_project/target/hiring.war to /var/lib/jenkins/ .m2/repository/in/javahome/hiring/Ø.1/hiring- 
Installing /var/1ib/jenkins/workspace/Maven_project/pom.xm1 to /var/lib/jenkins/ .m2/repository/in/javahome/hiring/Ø.1/hiring-e.1.pom 
EC1m- 
EC1;32mBUILD SUCCESSELm 
EC1m- 
Total time: 25.532 s 
Finished at: 
EC1m- 
2ø24-1e-28Te7 
to finish collecting data 
/var/lib/jenkins/workspace/Maven_project/pom.xm1 to in 1/ hiring-ø.l.pom 
/var/1ib/jenkins/workspace/Maven_project/target/hiring.war to in.javahome/hiring/ø. 1/hiring-Ø.1.war 
Waiting for Jenkins 
C JENKINS J Archiving 
C JENKINS J Archiving 
channel stopped 
Finished: SUCCESS 

Our build is successful.

Machine generated alternative text:
Lroot@ip-172-31-11-35 
Lroot@ip-172-31-11-35 
Lroot@ip-172-31-11-35 
Docker+ile Jenkins+ 
Lroot@ip-172-31-11- 
Lroot@ip-172-31-11 5 
cd 
cd /var/Iib/jenkins/workspace/Maven_project/ 
Is 
e jenkins+ile—cicd pom . xml README . md 
src target 
Maven _ cd target/ 
target) # Is 
Untitled Diagram . drawio 
hiring hiring . wa 
maven—archiver surefire 
—35 target) # 

Our war file is available in this Location.

5) Use the below code and create a parameterized job in jenkins

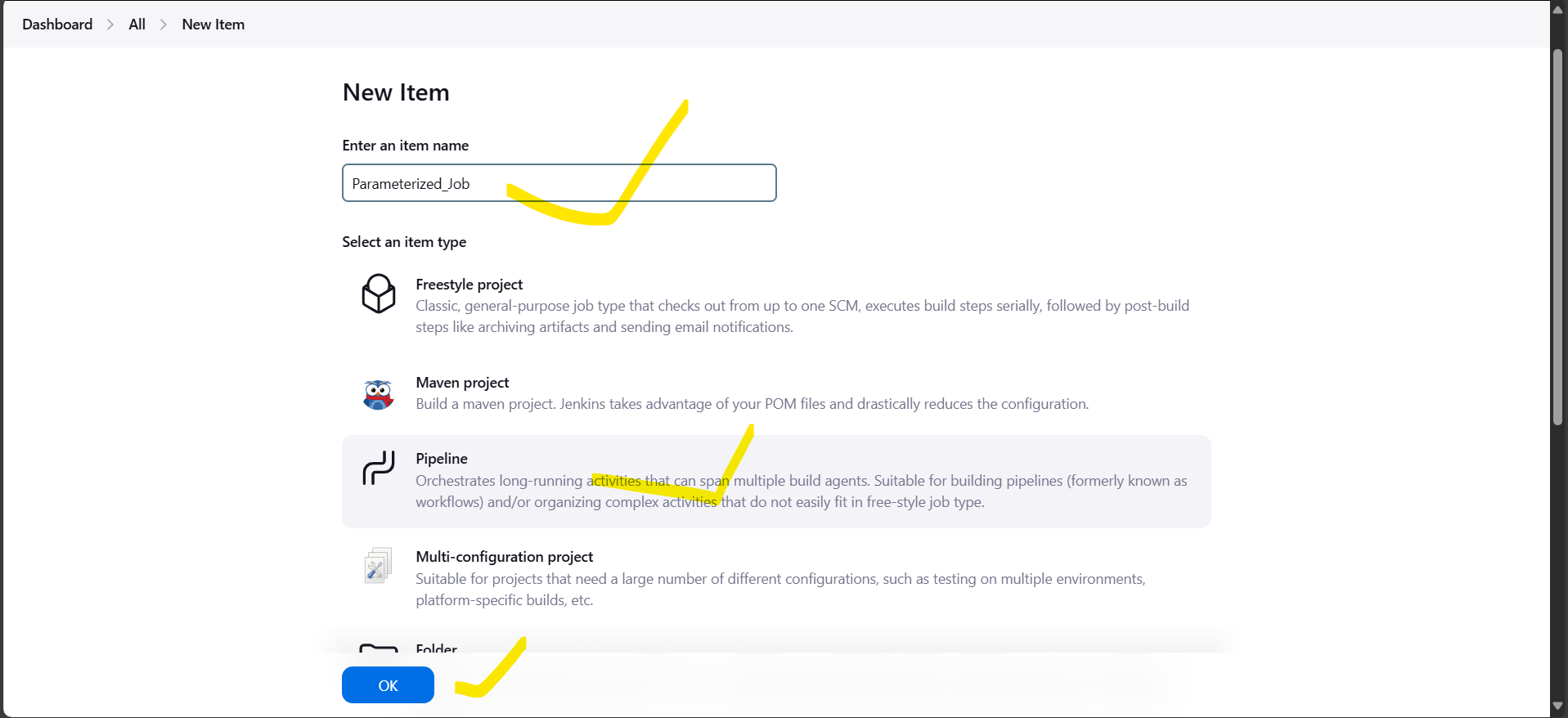
stage 1: Git clone

stage 2: Maven Compilation

Code: <https://github.com/betawins/java-Working-app.git>

Jenkins parameterized jobs let users enter specific inputs when starting a build. After the first build, you can choose different options for the next builds, like which branch to use or what type of build to run. This makes it easy to customize and test without changing the job settings.

* We already setup is there so just we want to create Job.
* Here I am selecting the pipeline and click on save.

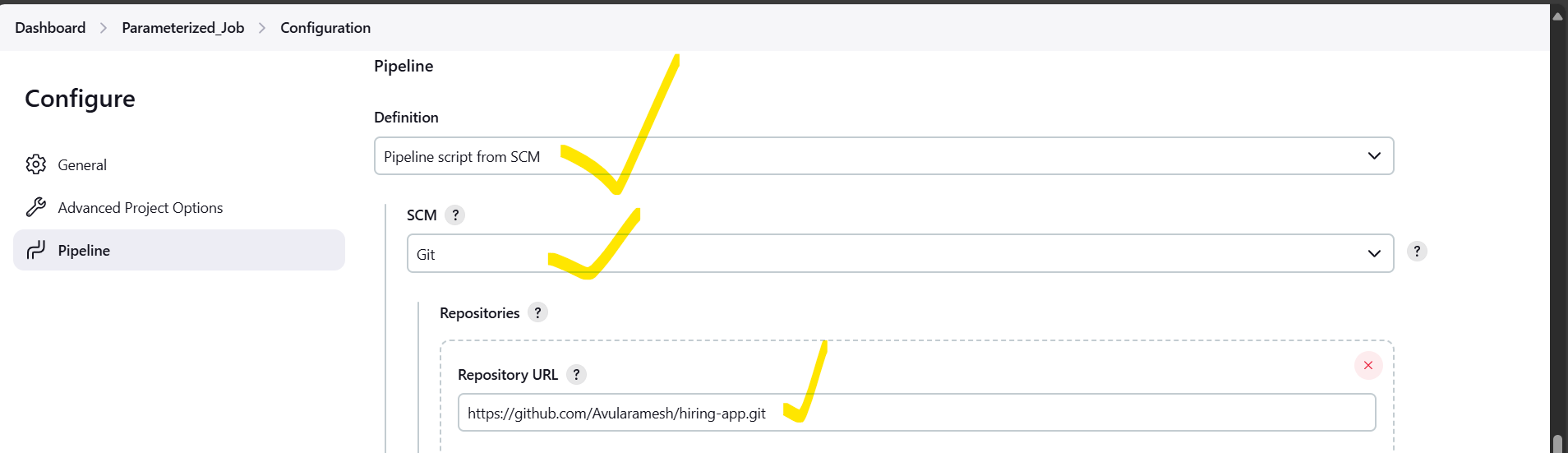


Then I went to the Pipeline.

Under the definition I am selecting --- Pipeline script from SCM

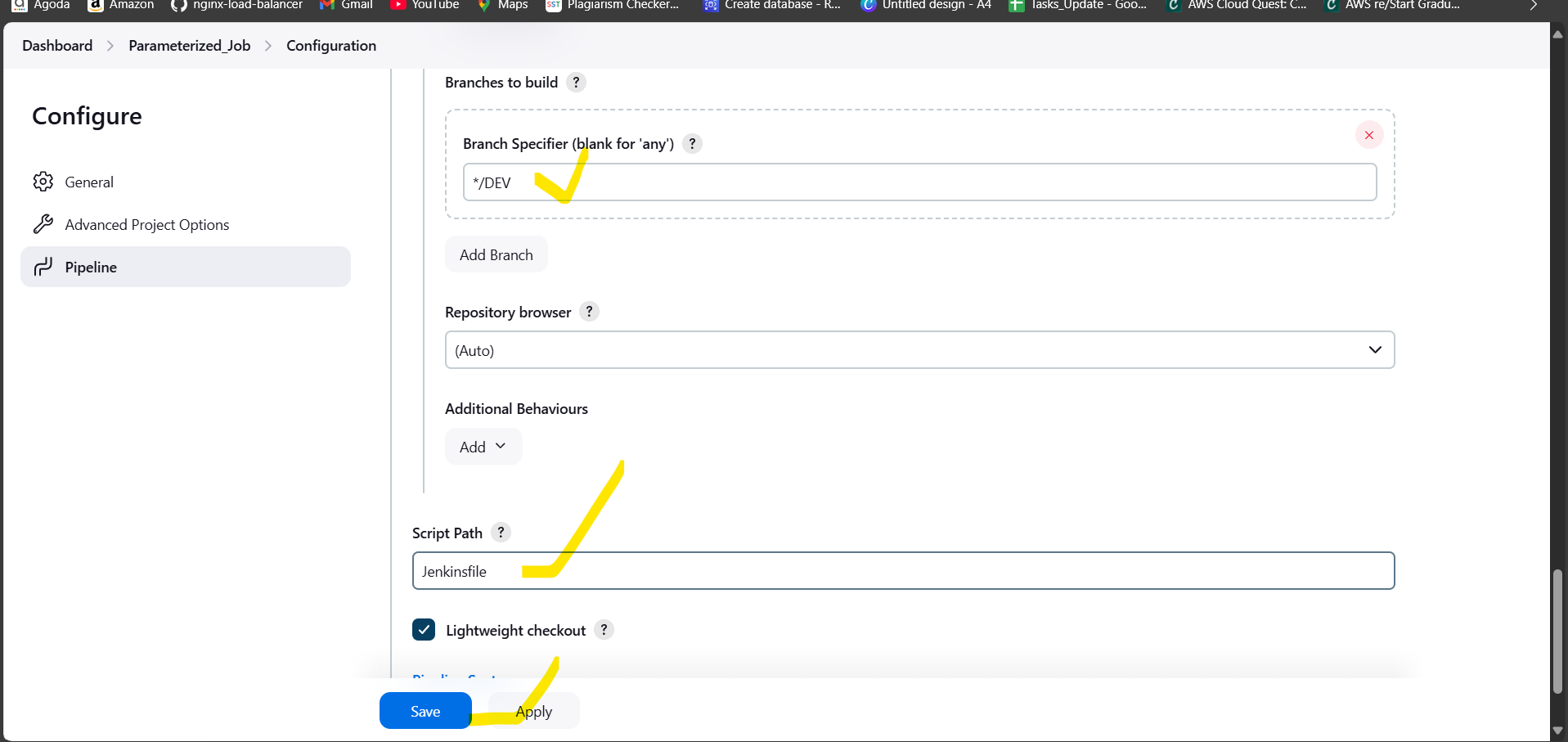
SCM ----- Git

Repositories ---- https://github.com/Avularamesh/hiring-app.git



Select Branch---

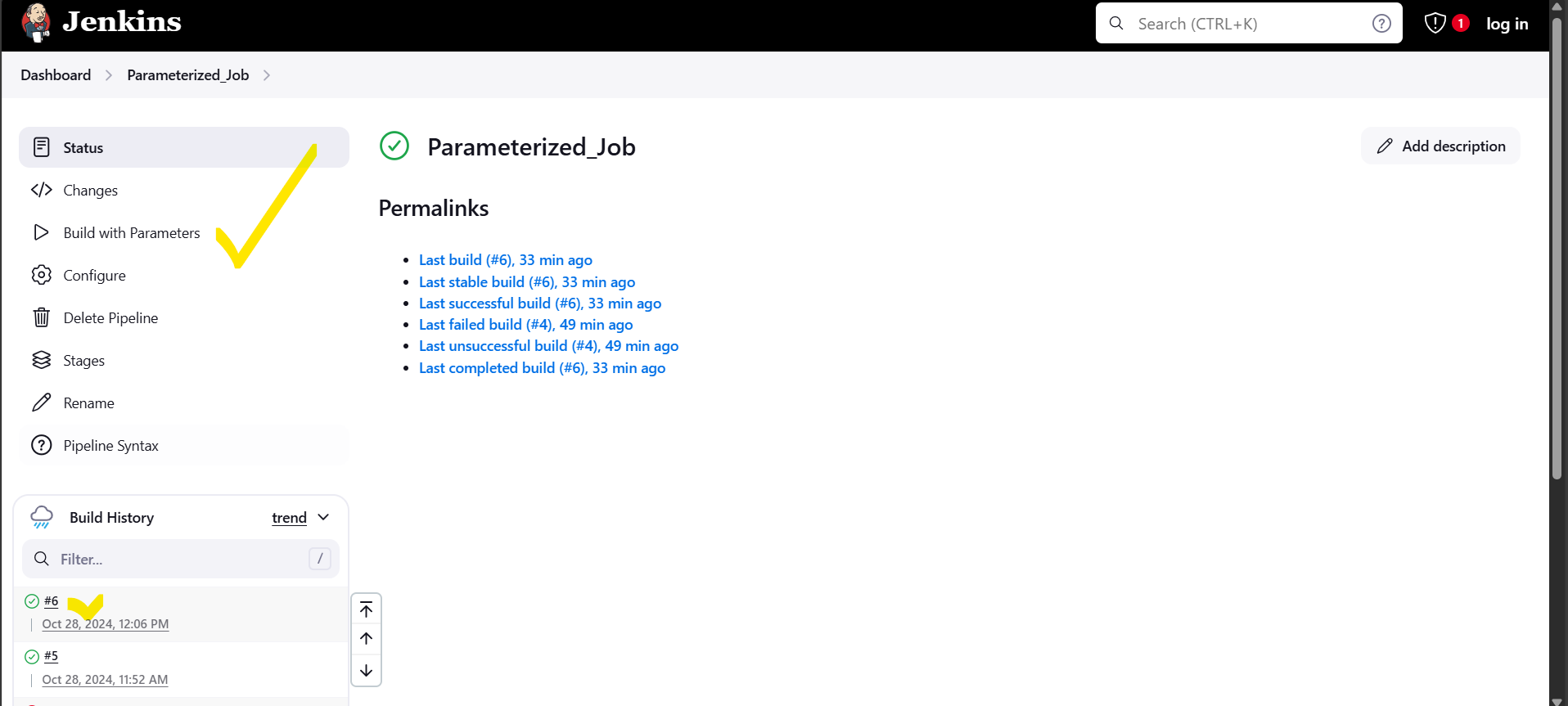
Script Path – Jenkinsfile and save the file.



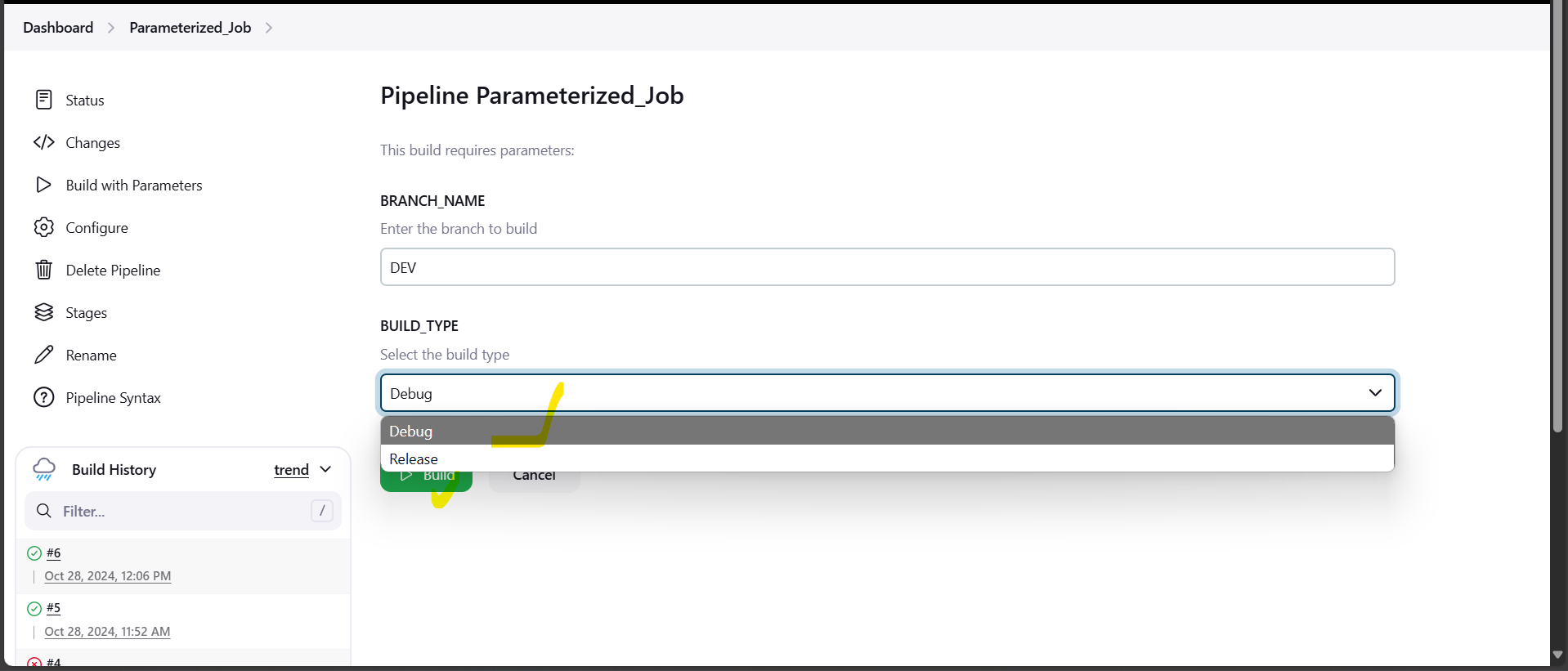
Now you have to build the Job.

After first build you see the option like build with parameters.

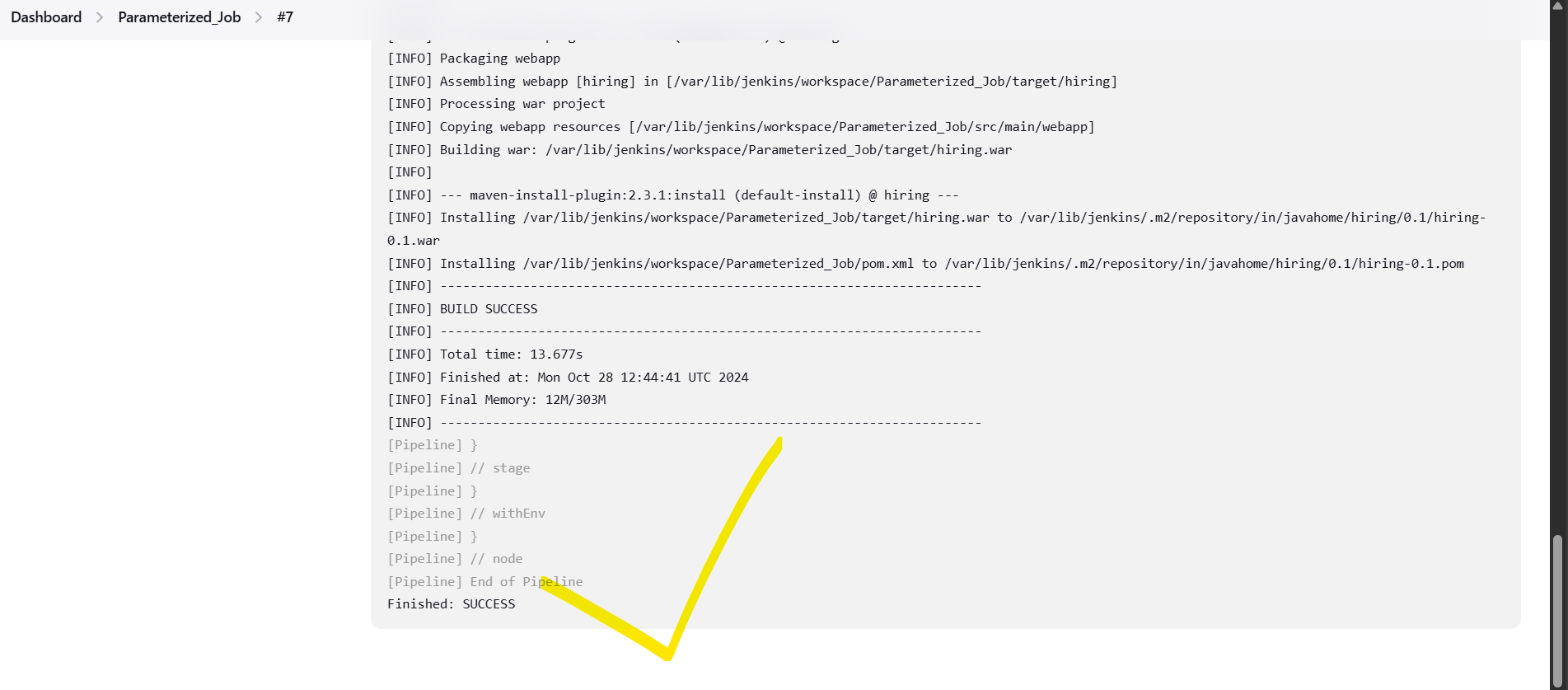
Then click on that



After that which option you want to select and build.



The build is successful.



6) What are the global varaiables in jenkins?

In Jenkins, global variables are predefined environment variables that provide context and information about the build environment and job execution. These variables are accessible during the execution of jobs and pipelines. Here are some commonly used global variables in Jenkins:

### **Common Global Variables:**

**env.BUILD\_ID**

* **Description:** Unique identifier for the current build.
* **Usage:** Can be used to track specific builds in logs or notifications.

**env.BUILD\_NUMBER**

* **Description:** The current build number for the job.
* **Usage:** Useful for naming artifacts or versioning.

**env.JOB\_NAME**

* **Description:** The name of the job being executed.
* **Usage:** Helps in identifying the job in scripts or logs.

**env.WORKSPACE**

* **Description:** The path to the workspace directory for the current job.
* **Usage:** Used to reference files or directories created during the build process.

**env.GIT\_COMMIT**

* **Description:** The commit hash of the checked-out Git commit.
* **Usage:** Useful for tracking which code version was built.

**env.GIT\_BRANCH**

* **Description:** The name of the branch being built.
* **Usage:** Helpful for differentiating builds from different branches.

**env.JOB\_URL**

* **Description:** The URL of the job on the Jenkins server.
* **Usage:** Can be included in notifications or reports for easy access to the job page.

**env.NODE\_NAME**

* **Description:** The name of the node where the build is running.
* **Usage:** Useful in multi-node Jenkins environments to identify where the build is executed.

### **Conclusion:**

Global variables in Jenkins are essential for creating dynamic and context-aware build processes. They enable users to access and utilize relevant information about the build environment and job execution, improving automation efficiency and clarity.

Precedency order

variable "instance\_type" {

  description = "Instance type for the EC2 instance"

  type        = string

  default     = "t2.micro"

}

variable "access\_key\_id" {  # Variable name as defined

  description = "AWS Access Key"

  type        = string

  default     = "AKIAW3MD7G7LBGTD56E7"  # Consider removing sensitive data

}

variable "secret\_key\_id" {  # Variable name as defined

  description = "AWS Secret Key"

  type        = string

  default     = "hvPqluTyzVWzIBx1vfEORKKo5kVEuQ8YBzN80AsS"  # Consider removing sensitive data

}

variable "ami\_id" {

  description = "AMI ID for the EC2 instance"

  type        = string

  default     = "ami-0175bdd48fdb0973b"

}

variable "ebs\_optimized" {

  type = bool  // boolean type value

  default = true

}

variable "instance\_names" {

  description = "List of names for the EC2 instances"

  type        = list(string)

  default     = ["Dev\_EC2", "Stag\_ec2", "Prod\_EC2"]

}

provider "aws" {

    region     = "us-west-1"

    access\_key = var.access\_key\_id

    secret\_key = var.secret\_key\_id

}

resource "aws\_instance" "Ec2\_instance" {

    for\_each      = { for name in var.instance\_names : name => name }

    ami           = var.ami\_id

    instance\_type = var.instance\_type

    ebs\_optimized = var.ebs\_optimized

    tags = {

    Name = each.key  # Assign unique name from the list

  }

}

variable "project\_environment" {

  description = "project name and environment"

  type        = map(string)

  default     = {

    project     = "project-alpha",

    environment = "dev"

  }

}