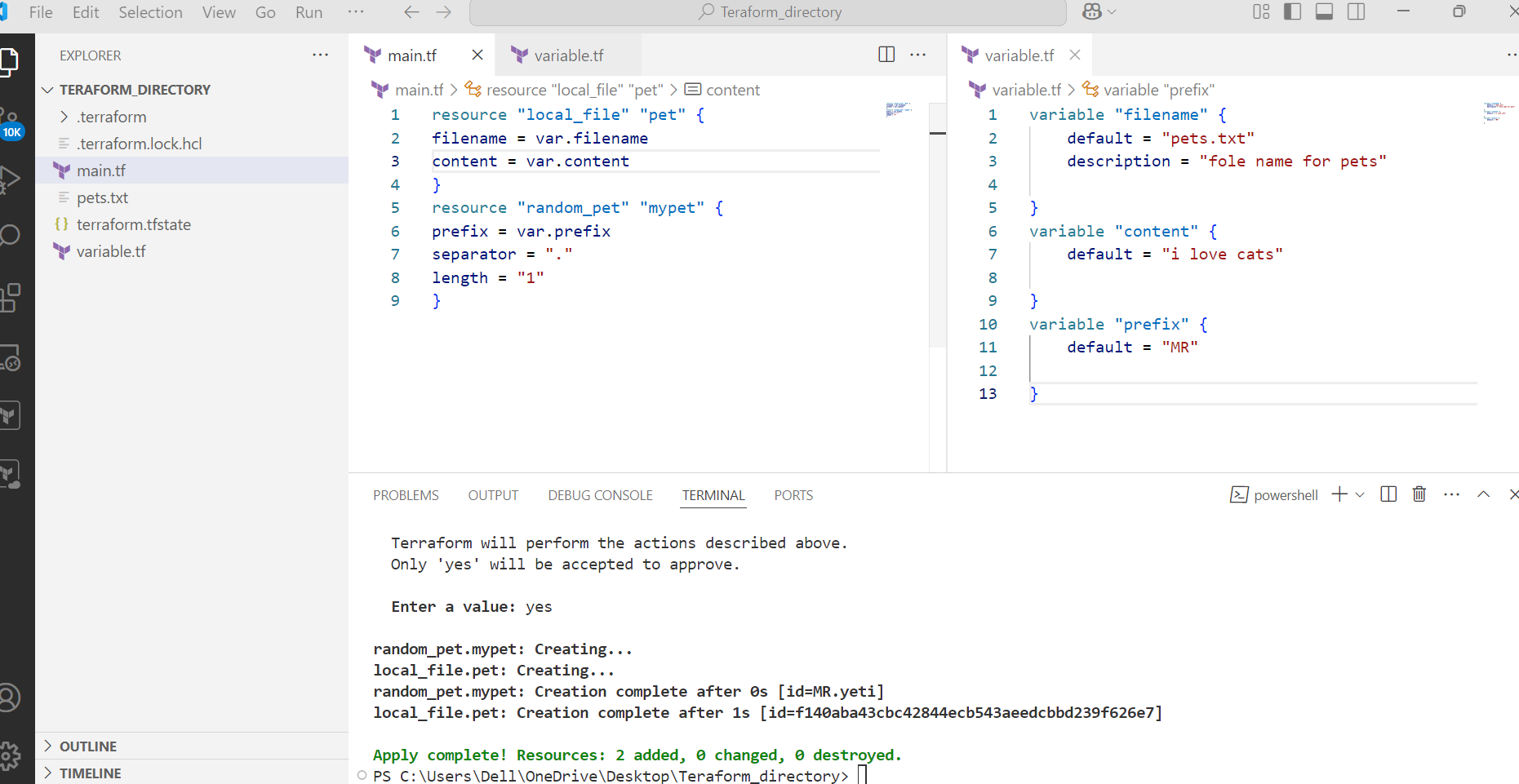
1. Watch terraform-03 video.

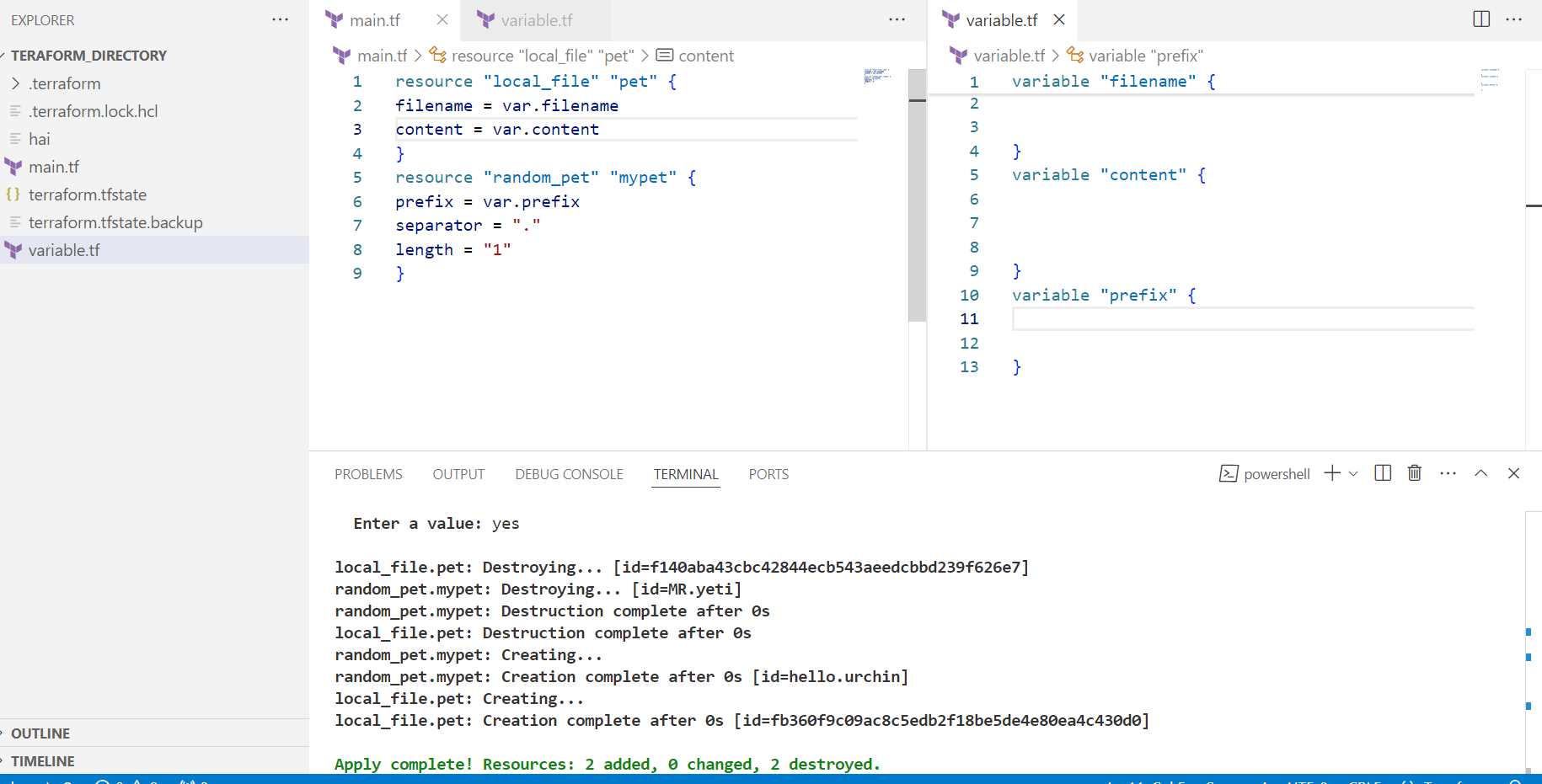
Watched

1. Execute the script shown in video.

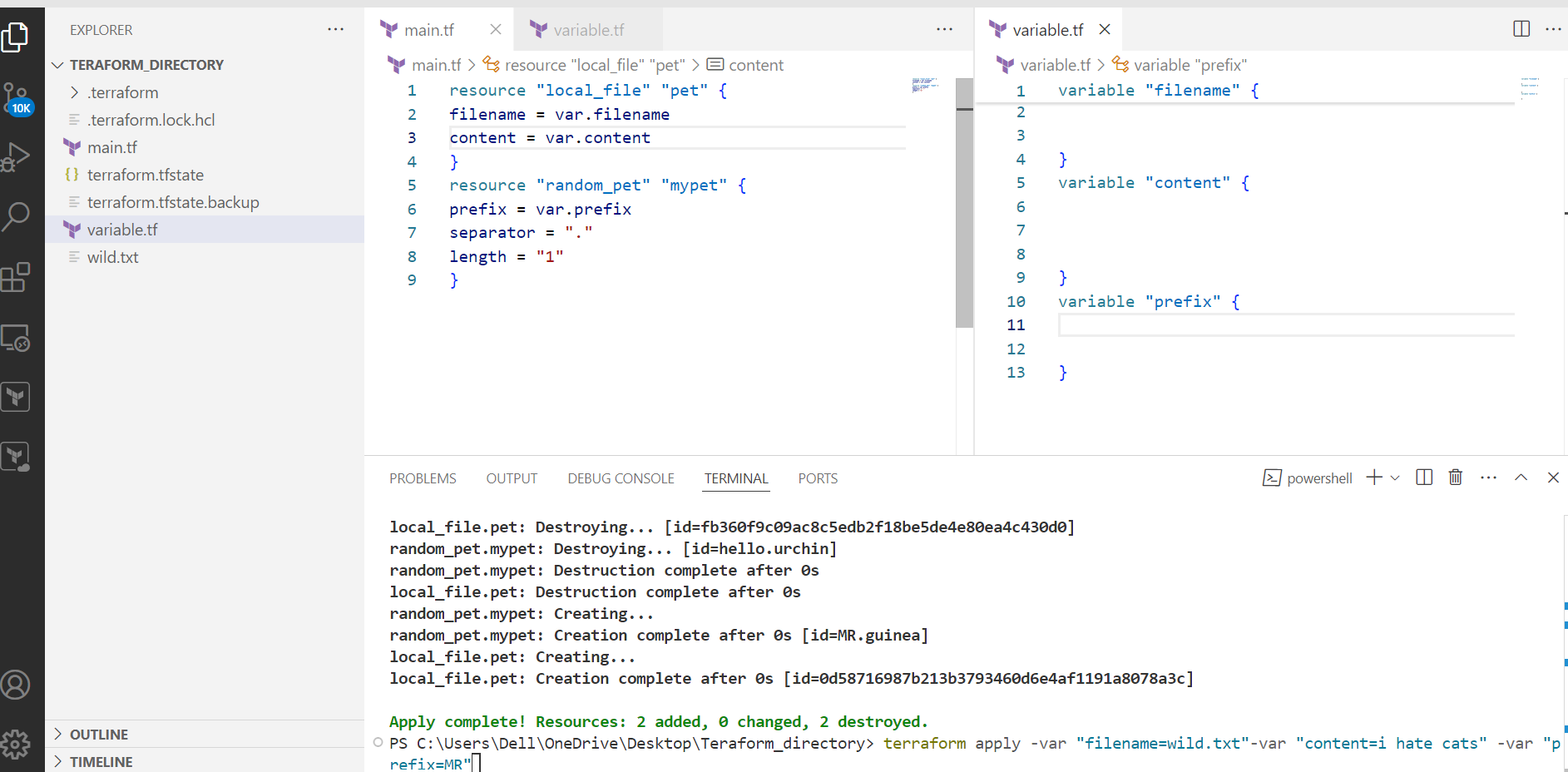
Executed enviranmental variables

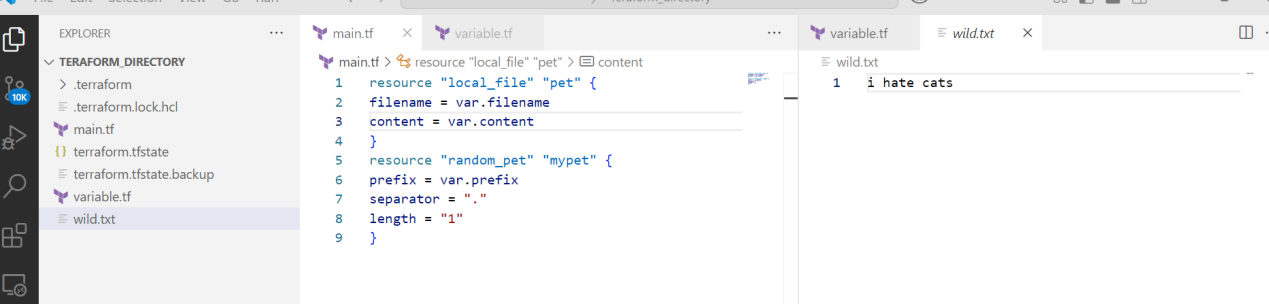


By using interactive mode

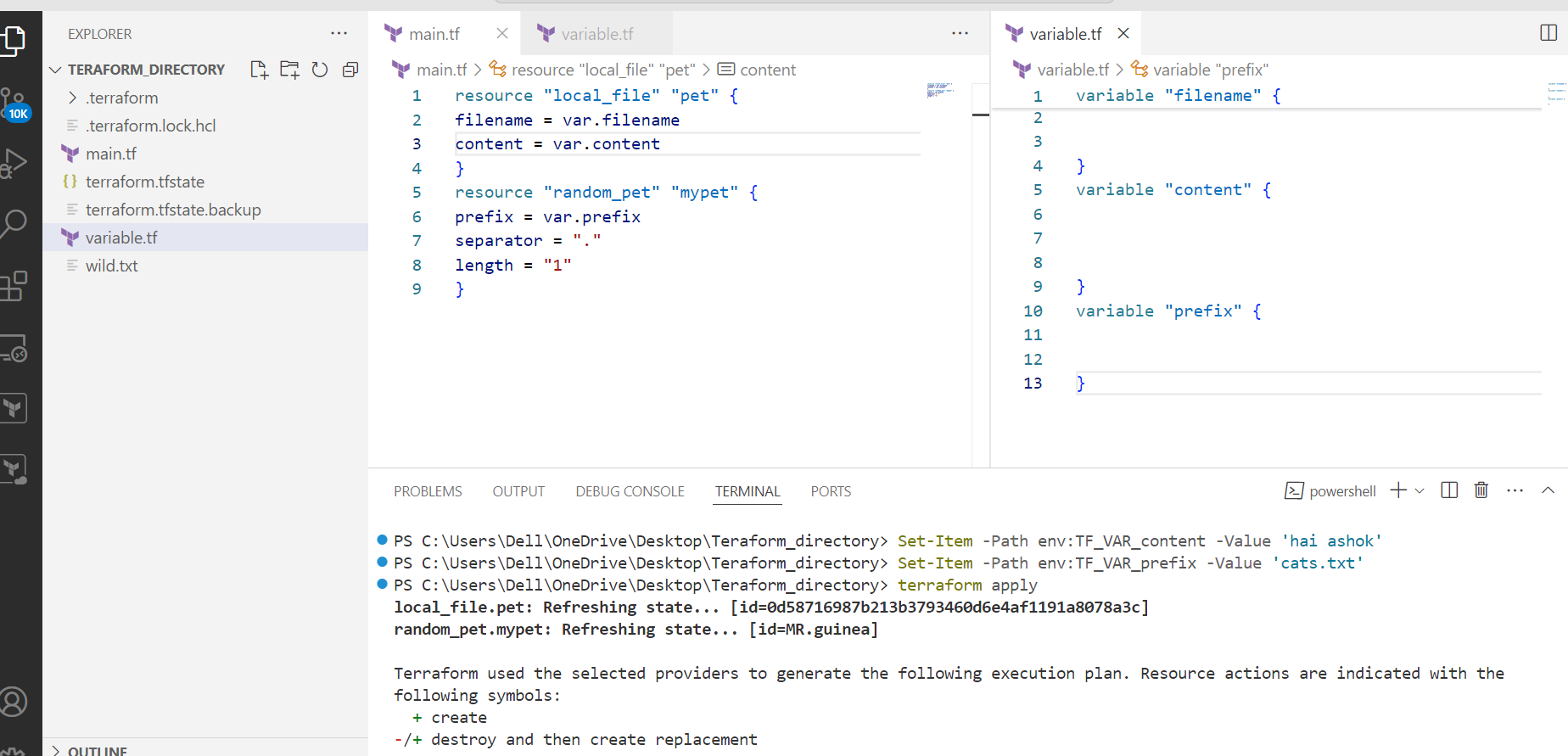


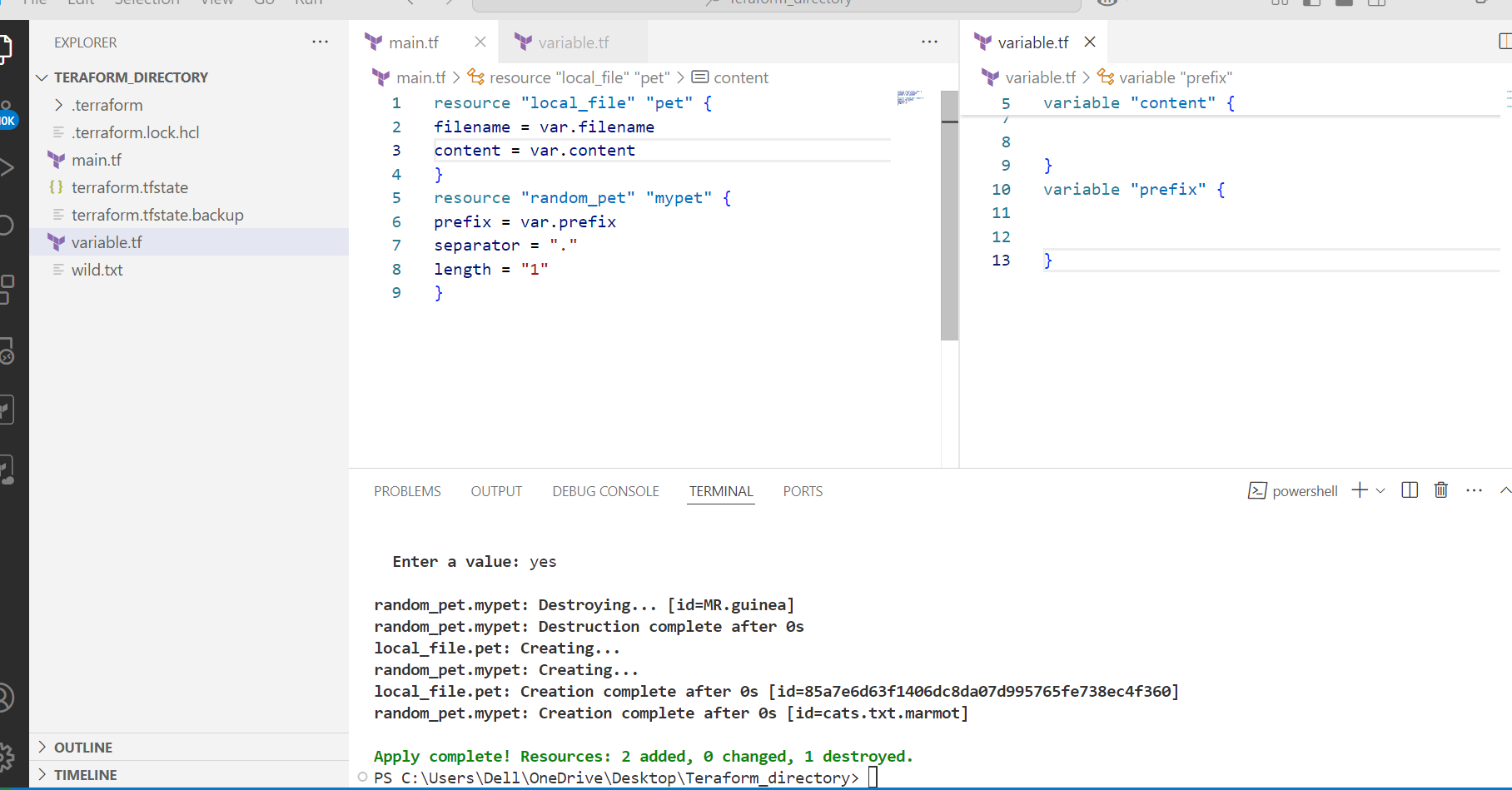
Command line flags





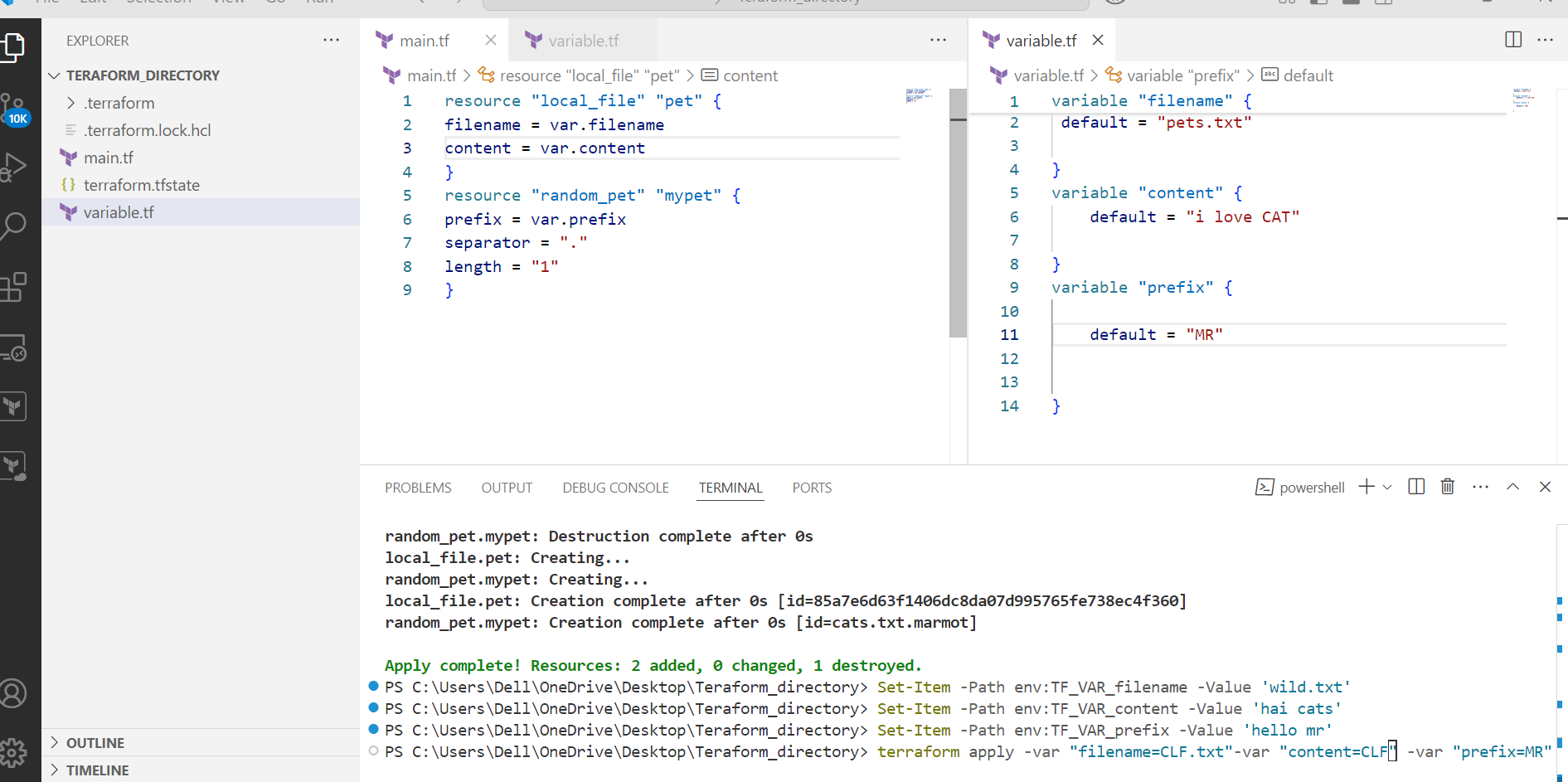
Environmental variable

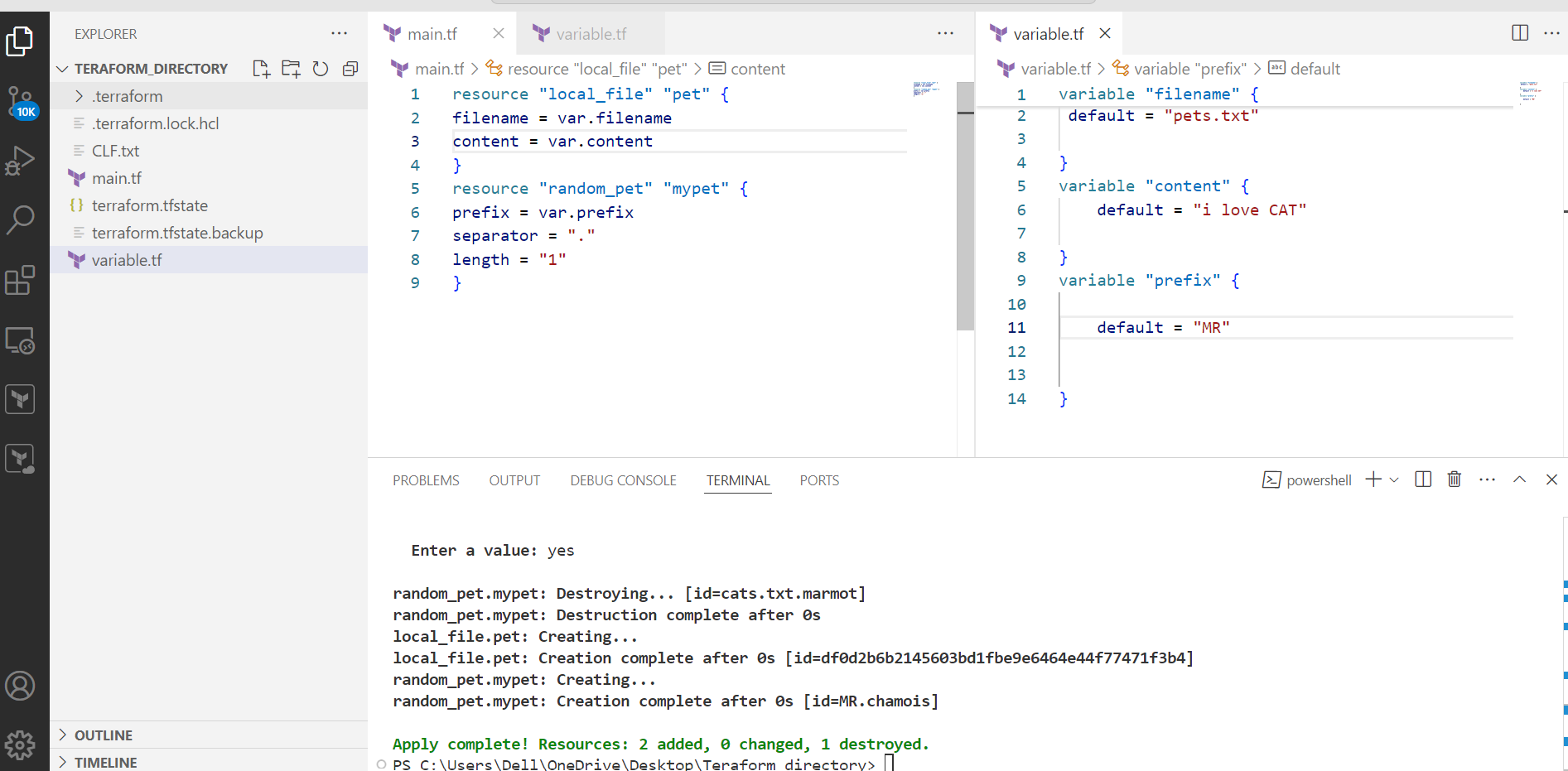


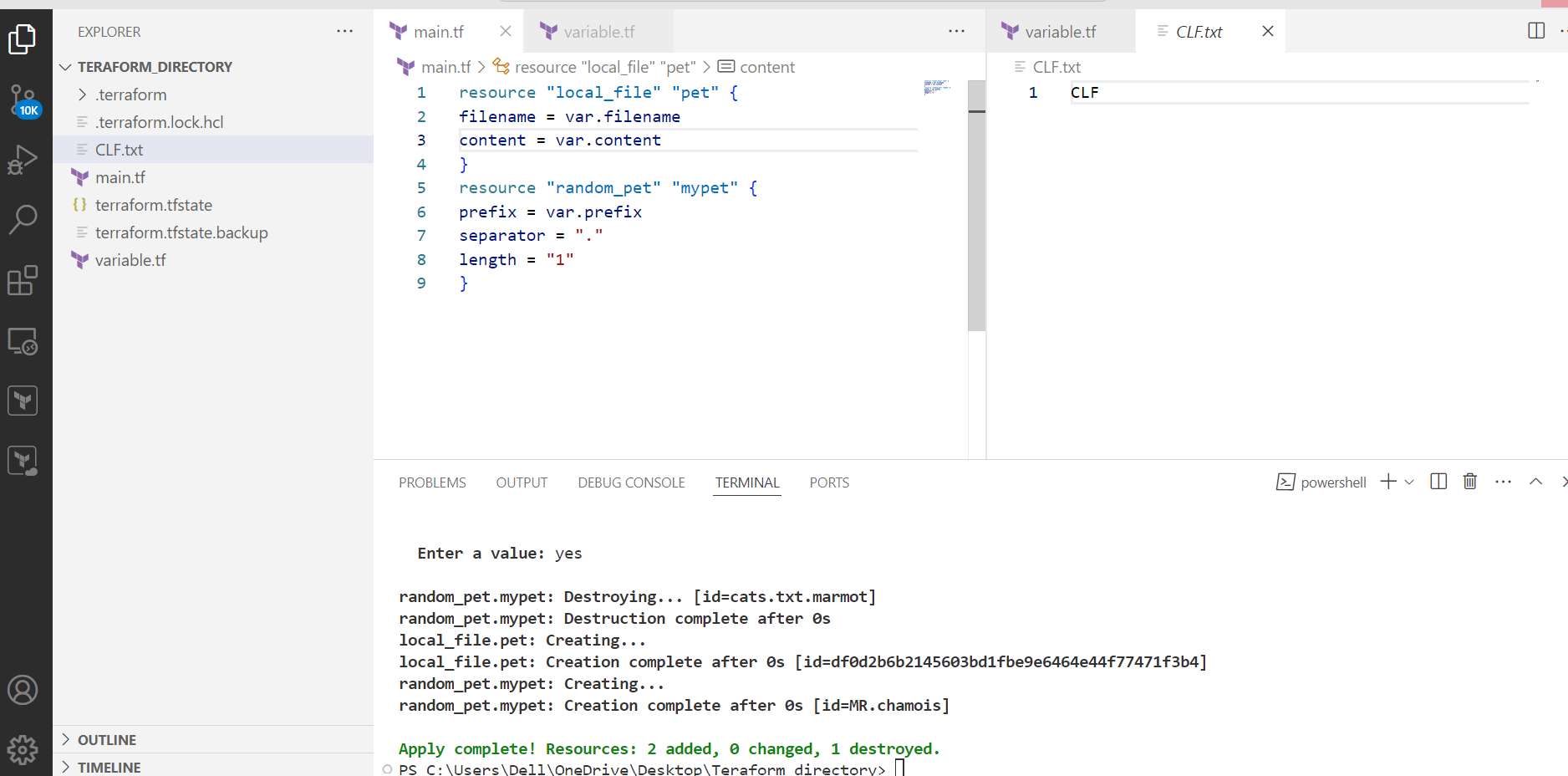


--> variable definition file: we are giving varibles from cli and in variable.tf

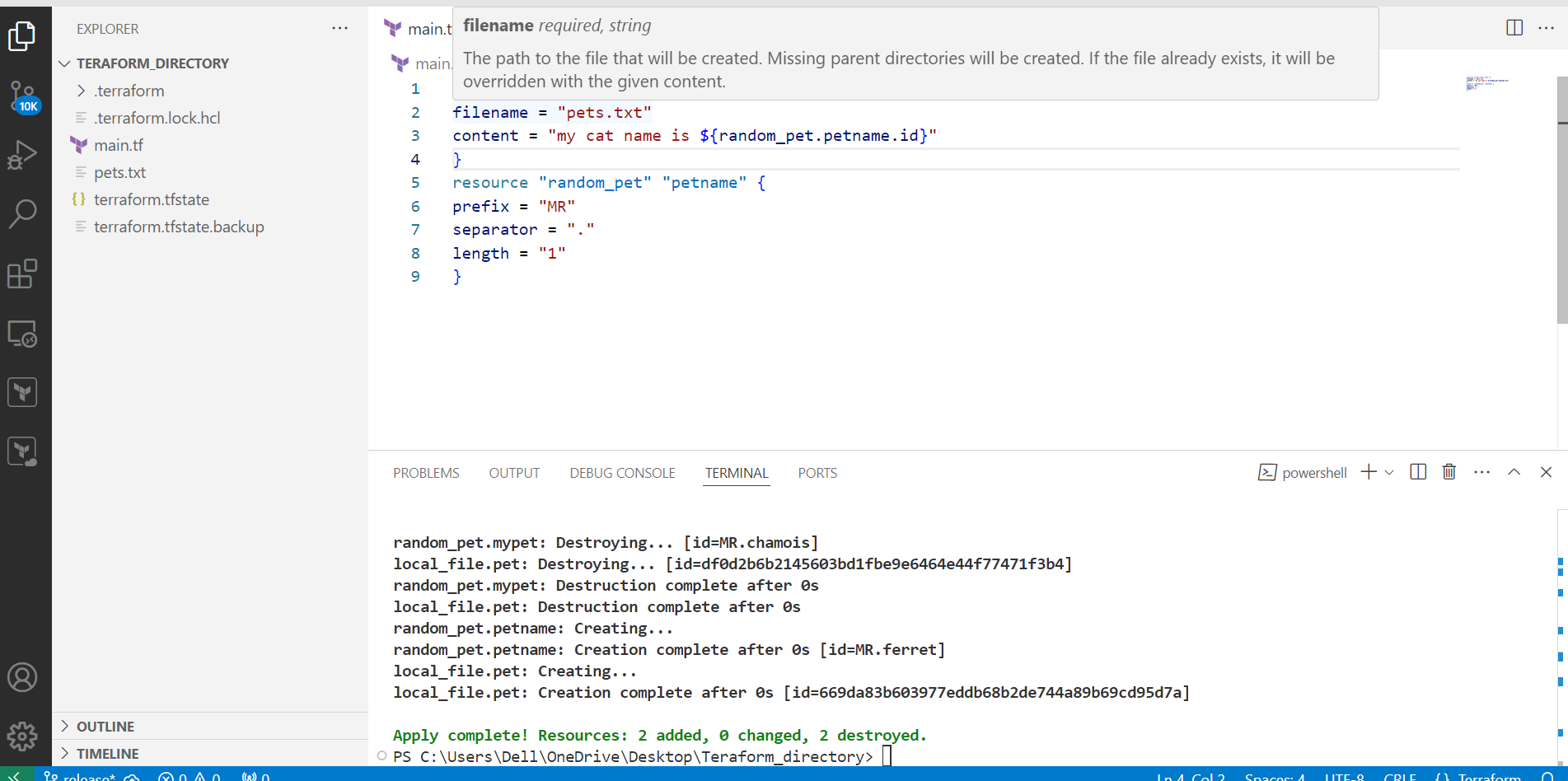
--> depending upon the prescience order it will execute



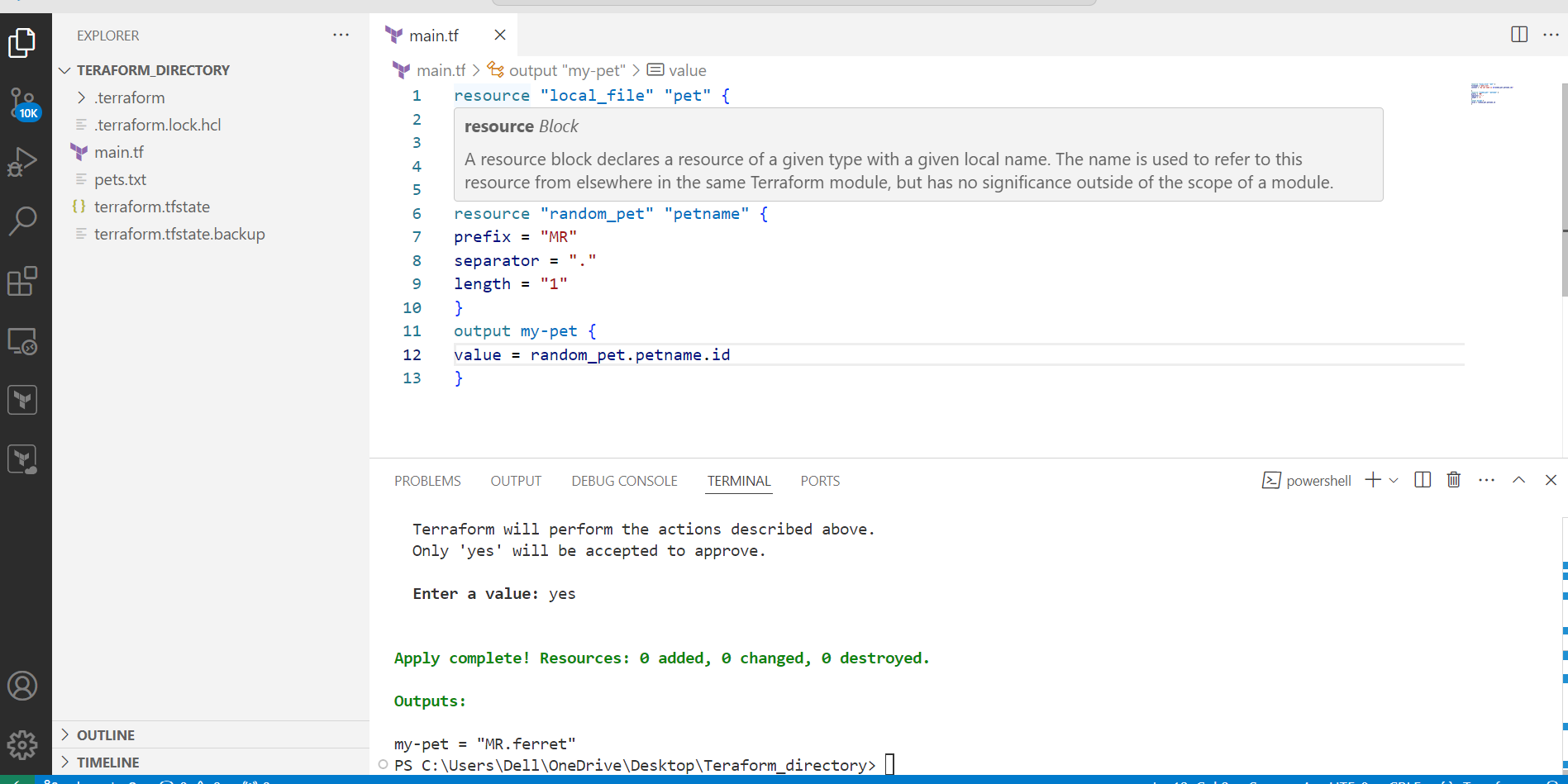




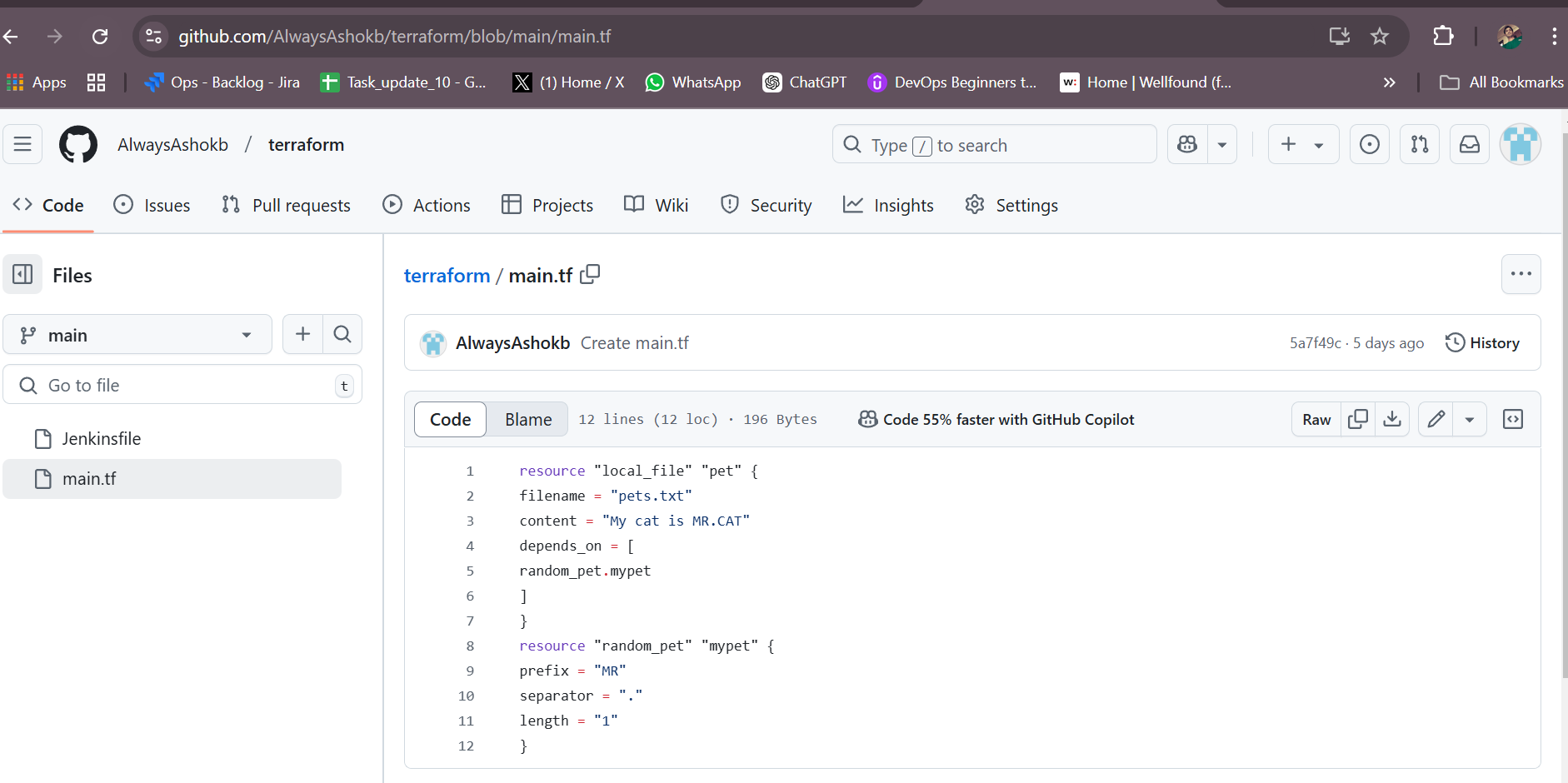
Using resouce attribute

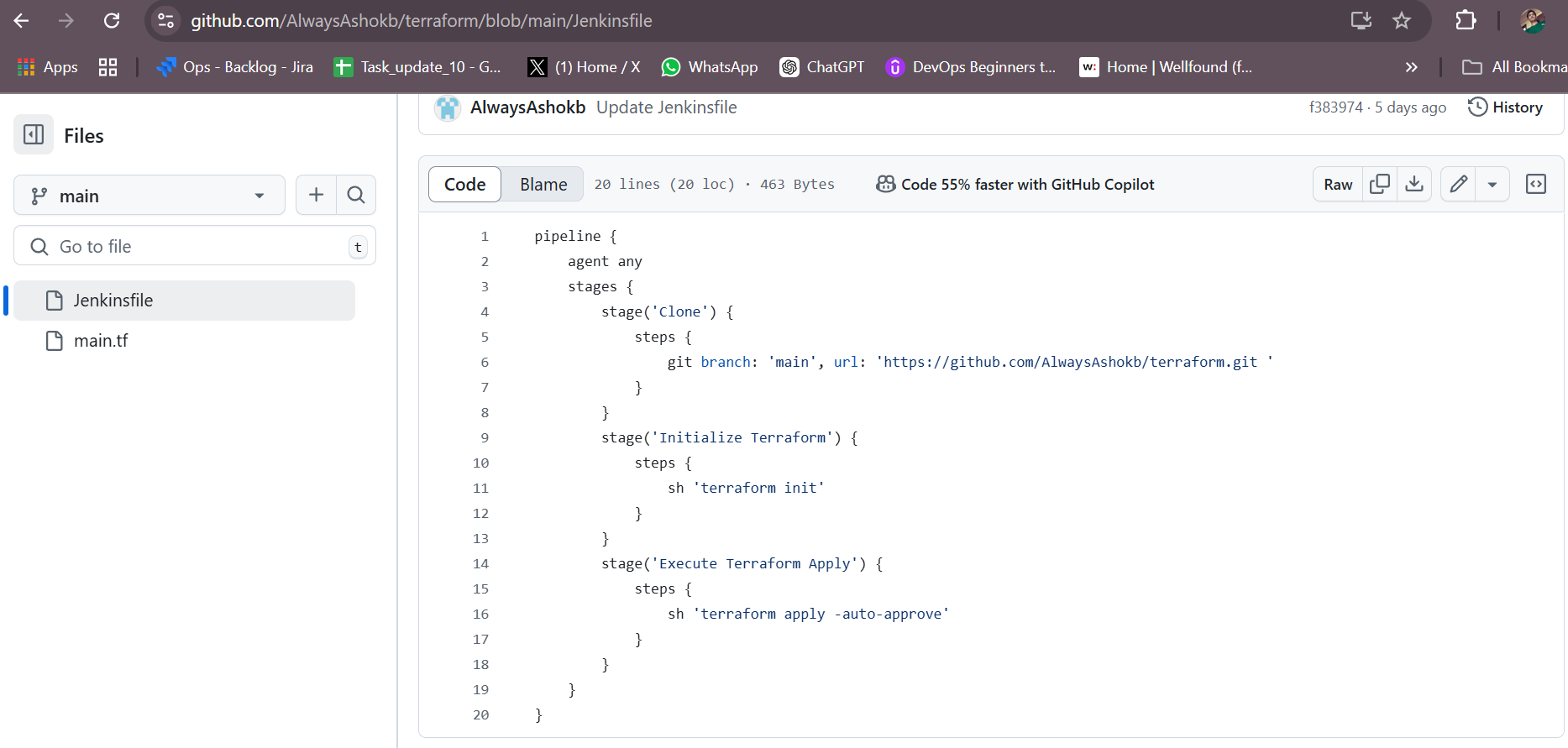


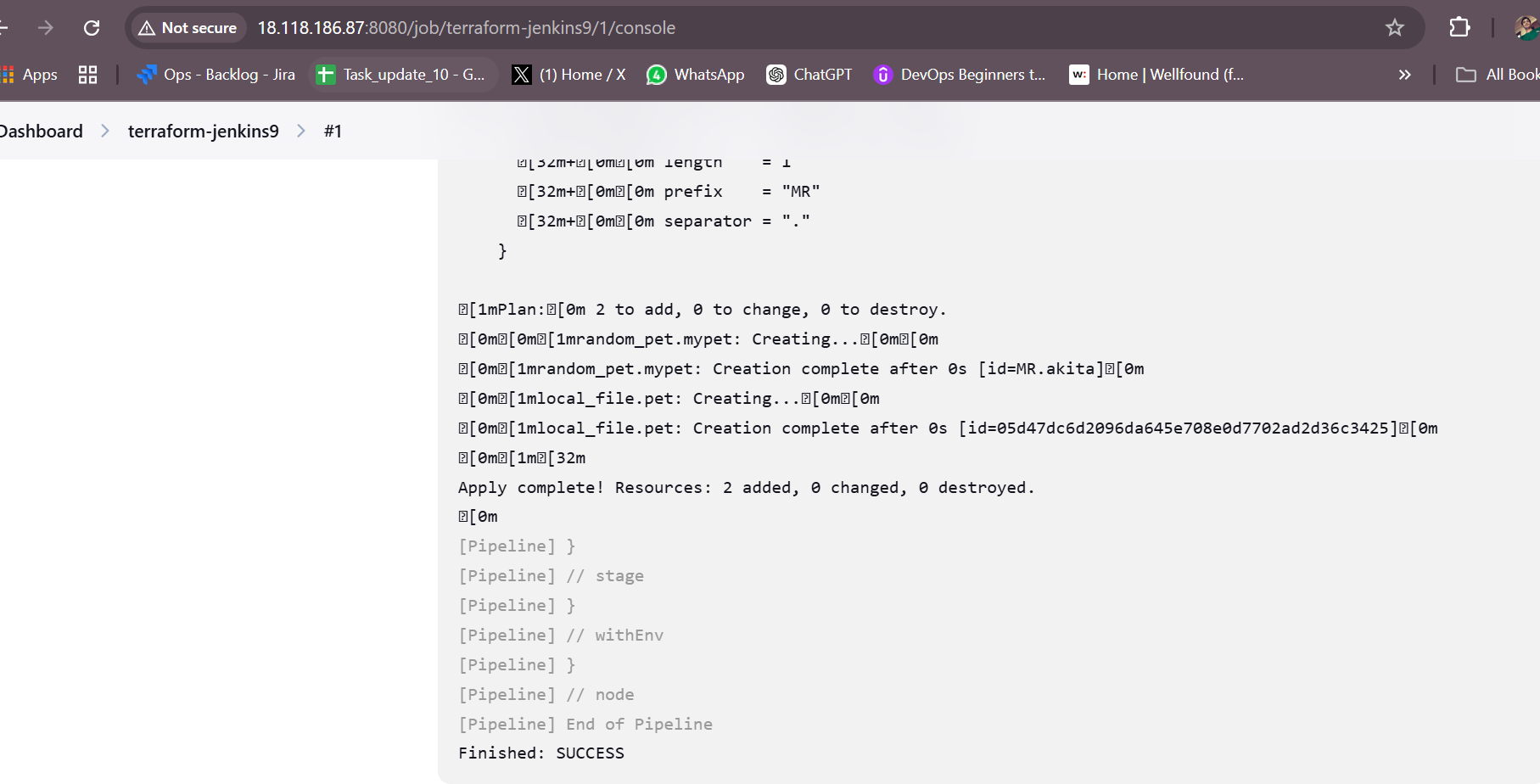
---> output variable



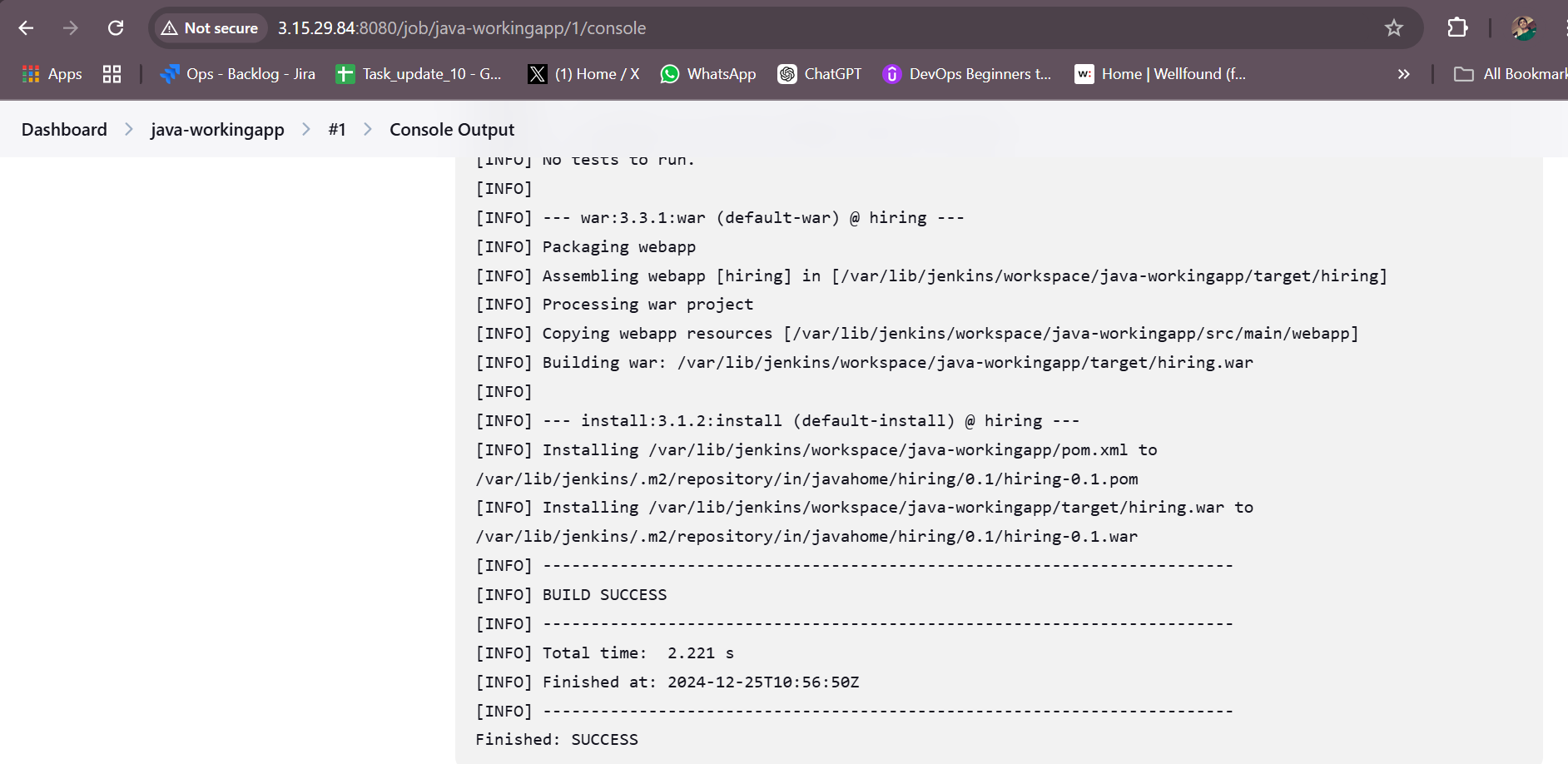
1. Intergrate terrafrom in jenkins using Terraform plugin.



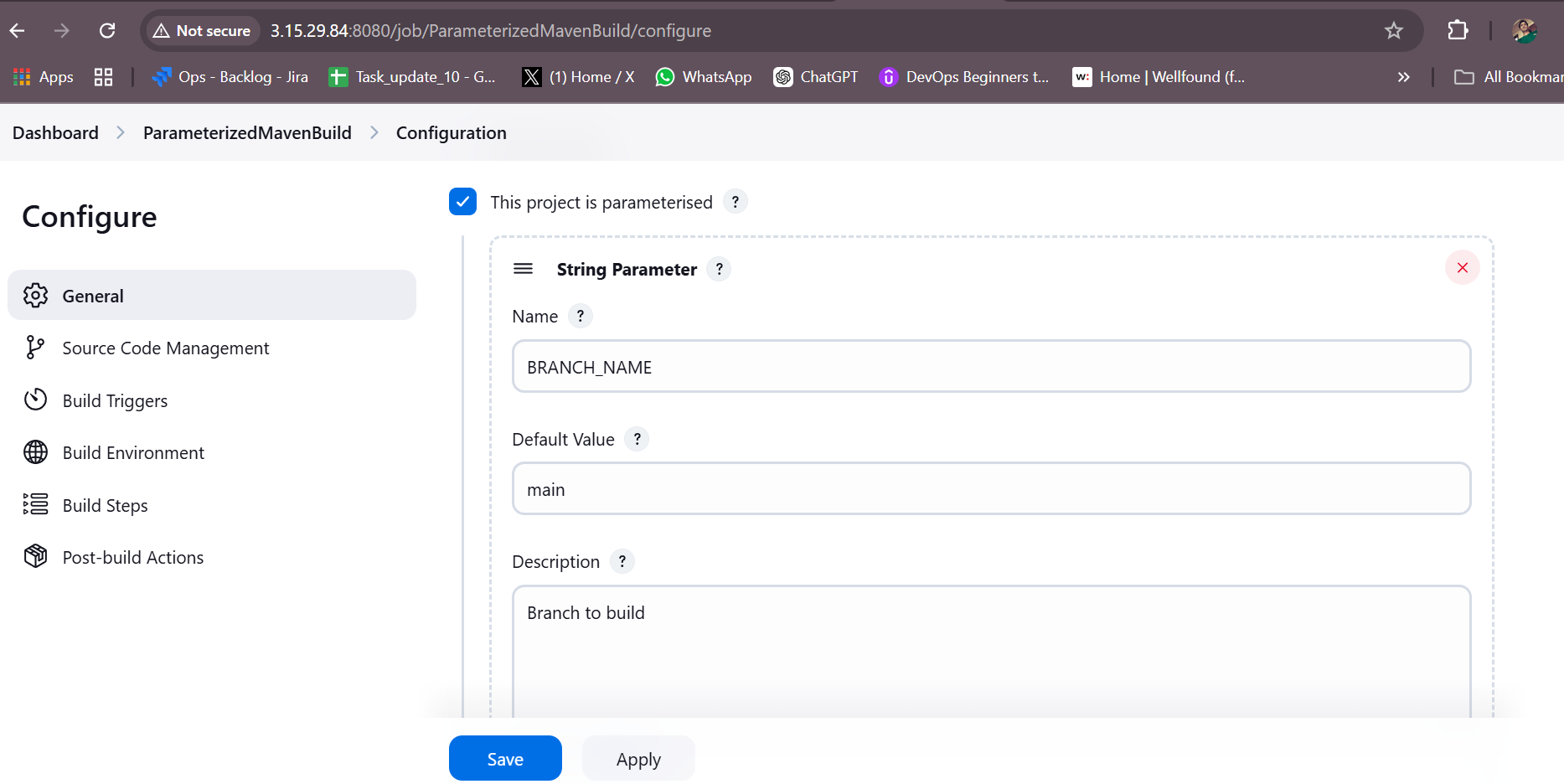


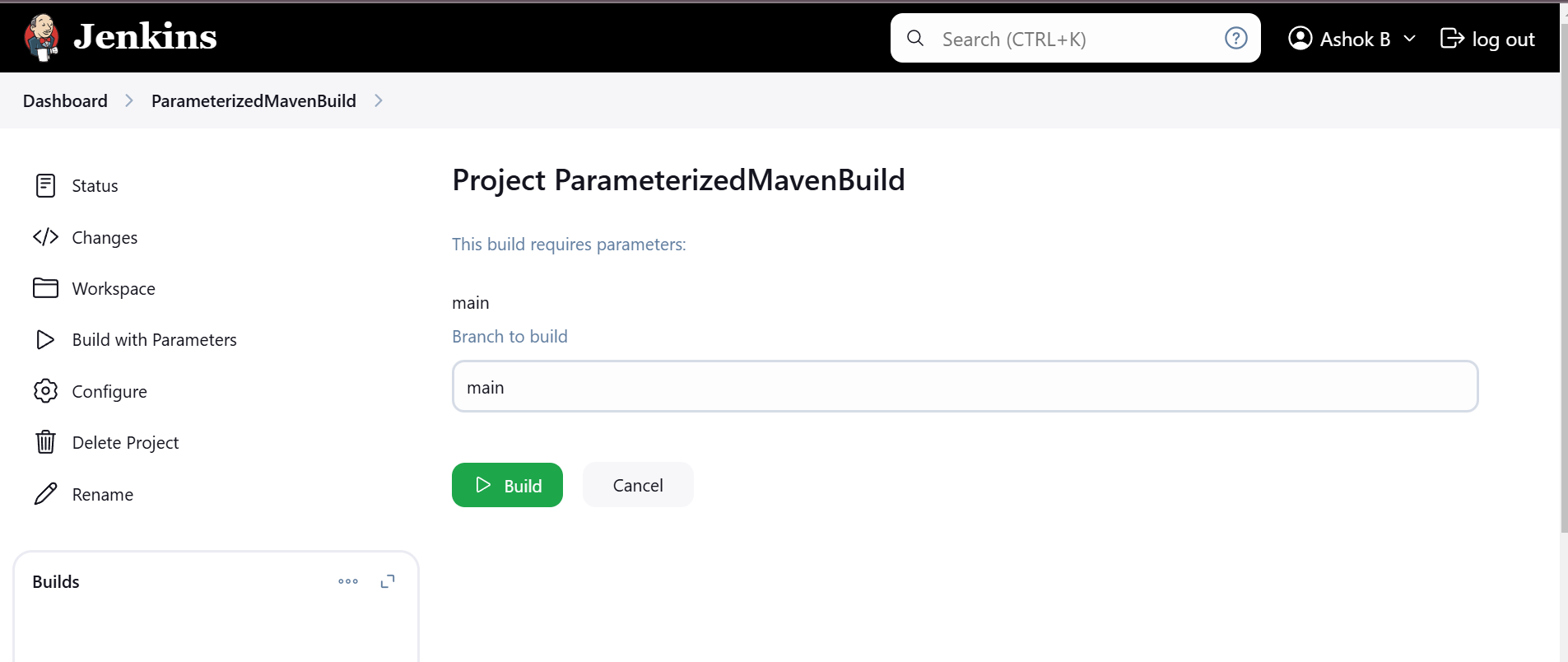


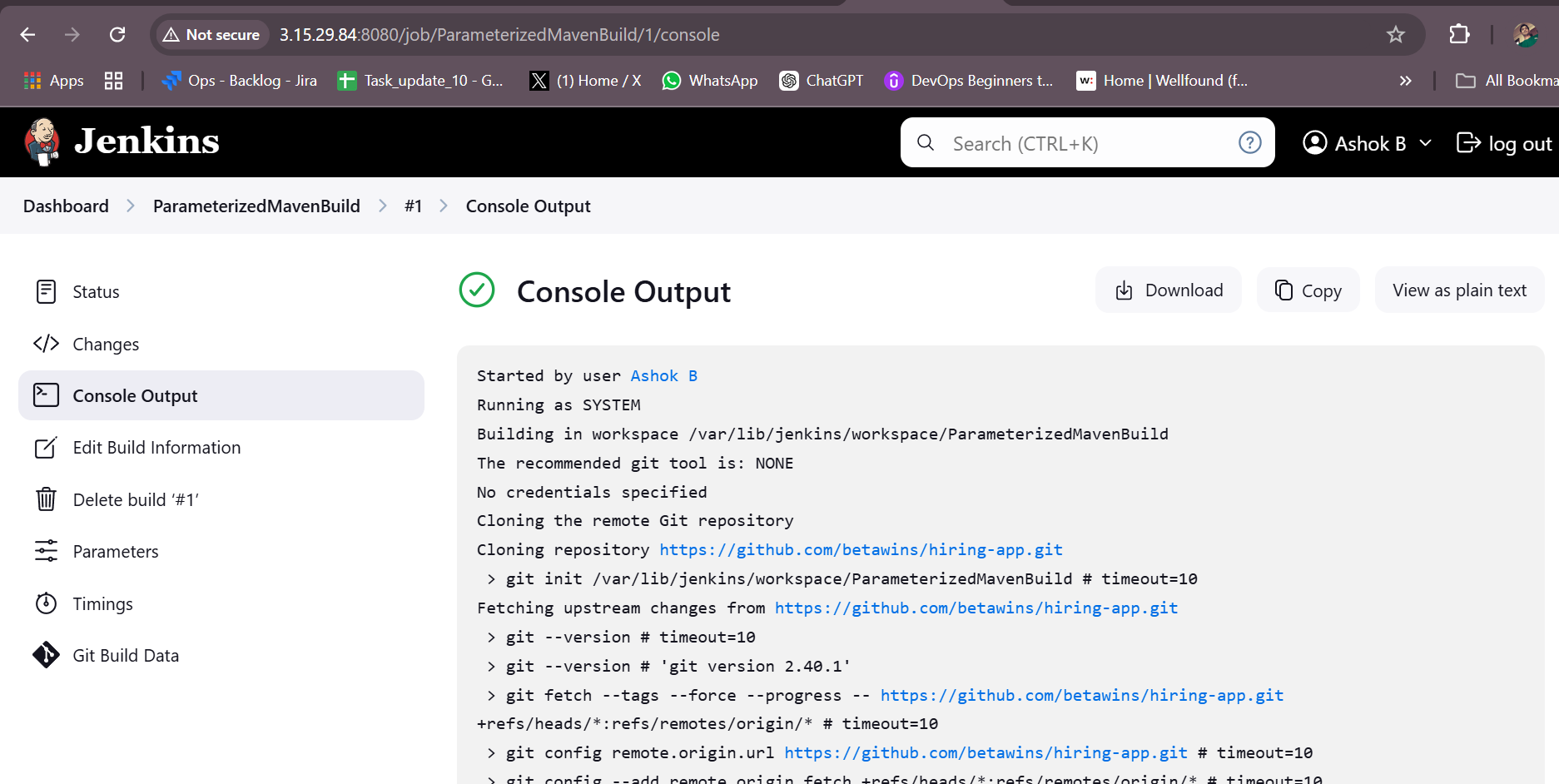
1. Create one jenkins job using MAVEN PROJECT for the below code with two stages. stage 1: Git clone stage 2: Maven Compilation Code: [https://github.com/betawins/java-Working-app.git](https://github.com/betawins/java-Working-app.git" \t "https://app.slack.com/client/T03TRQ064Q0/_blank)

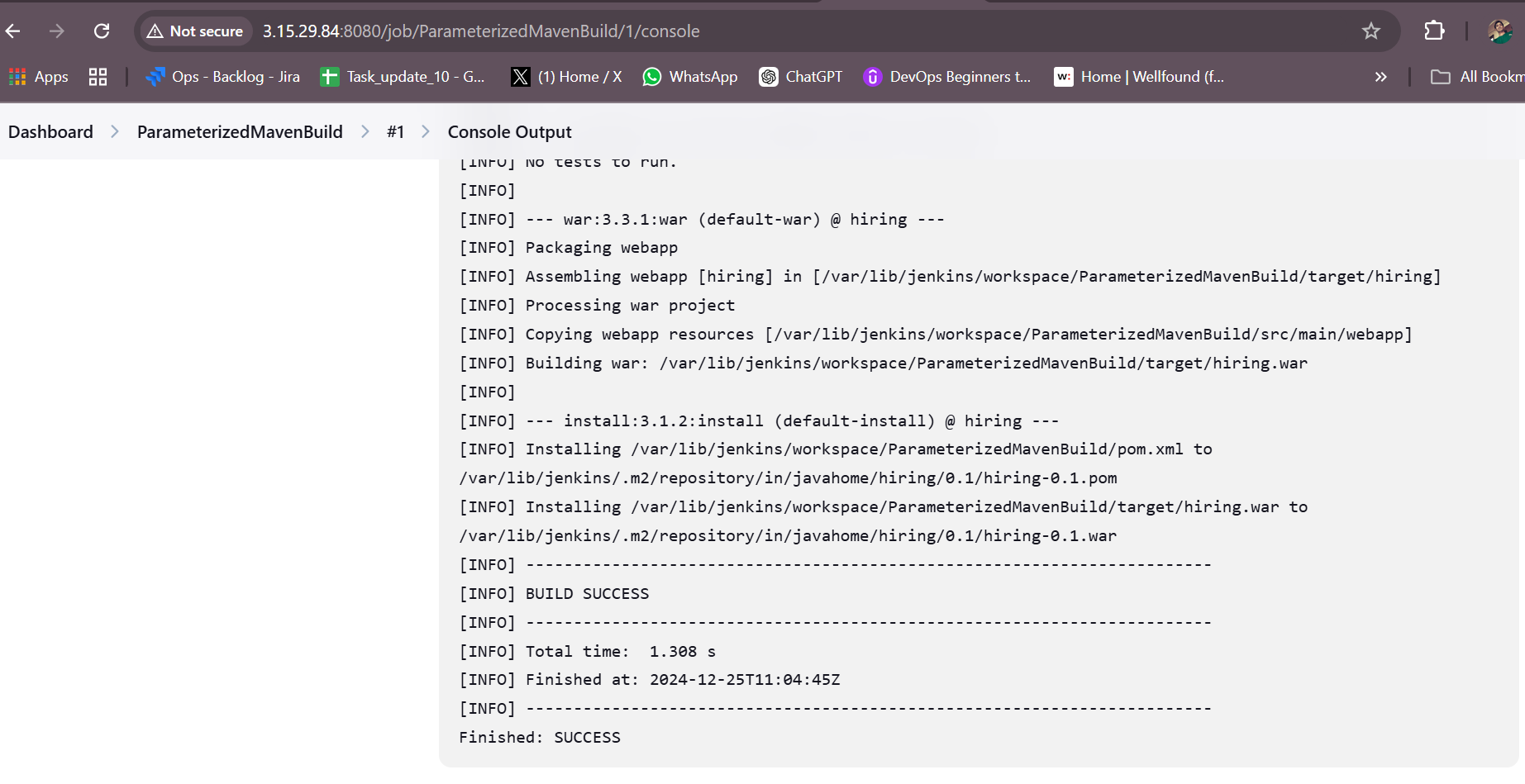


1. Use the below code and create a parameterized job in jenkins stage 1: Git clone stage 2: Maven Compilation Code: [https://github.com/betawins/java-Working-app.git](https://github.com/betawins/java-Working-app.git" \t "https://app.slack.com/client/T03TRQ064Q0/_blank)









1. What are the global varaiables in jenkins?

# Global Variables in Jenkins

Global variables in Jenkins are predefined variables and objects available in pipelines. They simplify access to Jenkins features and reduce the need for explicit configuration. Here is a categorized overview of key global variables in Jenkins:

## Pipeline-Specific Global Variables

### `env`

Represents the environment variables for the current build.

echo "Workspace: ${env.WORKSPACE}"

### `params`

Used to access pipeline parameters defined in the job configuration.

echo "Parameter value: ${params.MY\_PARAM}"

### `currentBuild`

Provides details about the current build, such as result, duration, and more.

echo "Build result: ${currentBuild.result}"

### `build` (Scripted Pipeline only)

Refers to the current build instance.

echo "Build number: ${build.number}"

### `scm`

Provides details about the source code management (SCM) configuration, such as Git or SVN.

checkout scm

## Job Information Variables

### `WORKSPACE`

Directory where the job is executed.

echo "Workspace directory: ${env.WORKSPACE}"

### `BUILD\_NUMBER`

Unique identifier for the current build.

echo "Build number: ${env.BUILD\_NUMBER}"

### `JOB\_NAME`

Name of the Jenkins job.

echo "Job name: ${env.JOB\_NAME}"

### `BUILD\_ID`

Build ID, typically the same as `BUILD\_NUMBER`.

echo "Build ID: ${env.BUILD\_ID}"

### `NODE\_NAME`

Name of the node where the build is running.

echo "Running on node: ${env.NODE\_NAME}"

## Utility Objects and Methods

### `steps`

Provides access to pipeline steps.

steps.echo("Hello, World!")

### `manager`

Available in some Scripted Pipelines to manage jobs and builds.

manager.build.displayName = "Custom Build Name"

### `docker`

Used to interact with Docker containers.

docker.image('alpine').inside {  
 sh 'echo Hello from Docker!'  
}

### `tool`

Provides access to tools configured in Jenkins, like Maven or JDK.

def mavenHome = tool name: 'Maven', type: 'maven'

### `pipeline`

Provides access to Declarative Pipeline structure.

## Predefined Environment Variables

### `GIT\_COMMIT`

Current Git commit hash.

### `GIT\_BRANCH`

Current branch being built.

### `BUILD\_URL`

URL of the build in Jenkins.

### `JOB\_URL`

URL of the job in Jenkins.

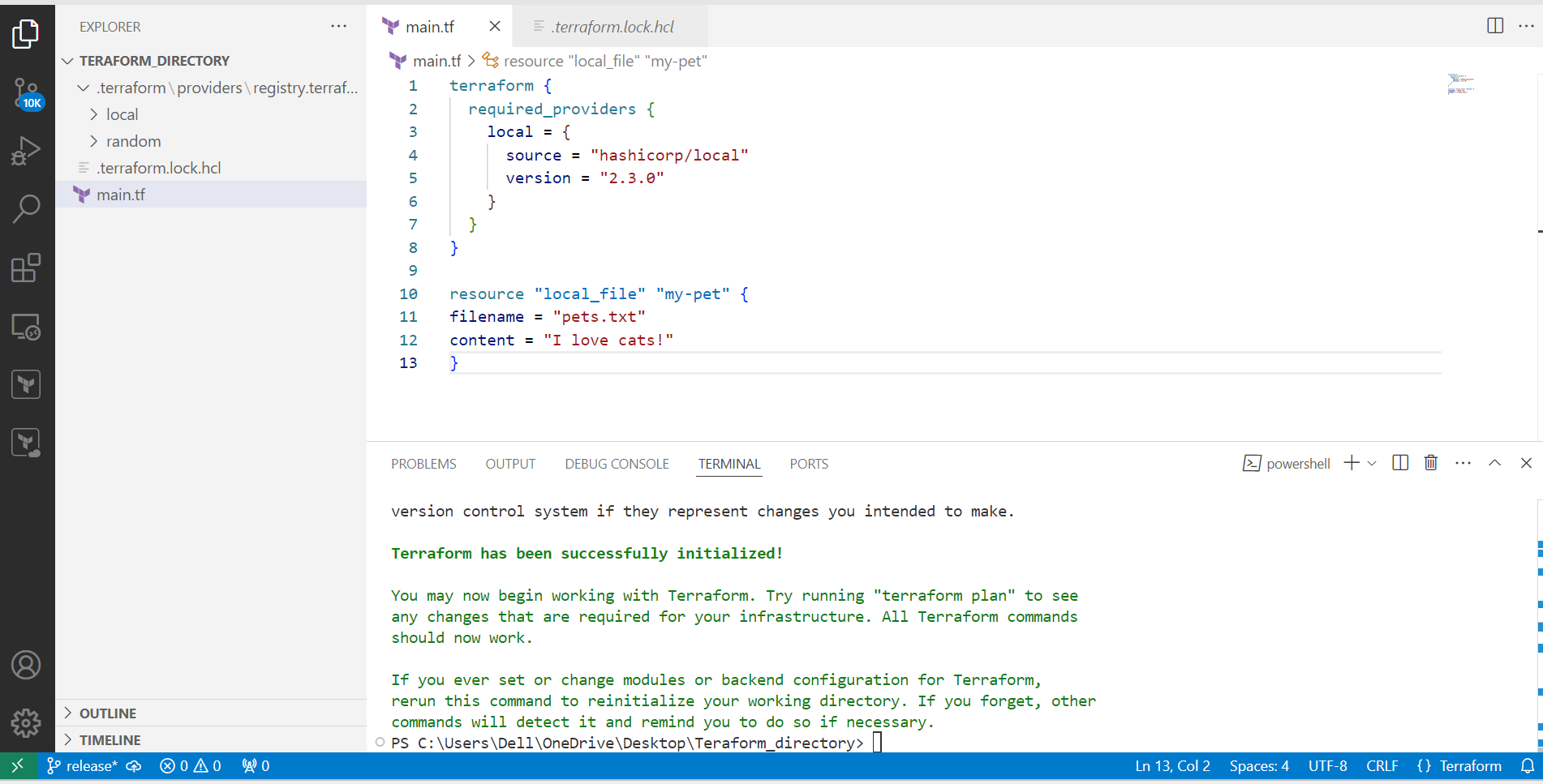
1. Watch terraform-04 video.

Watched

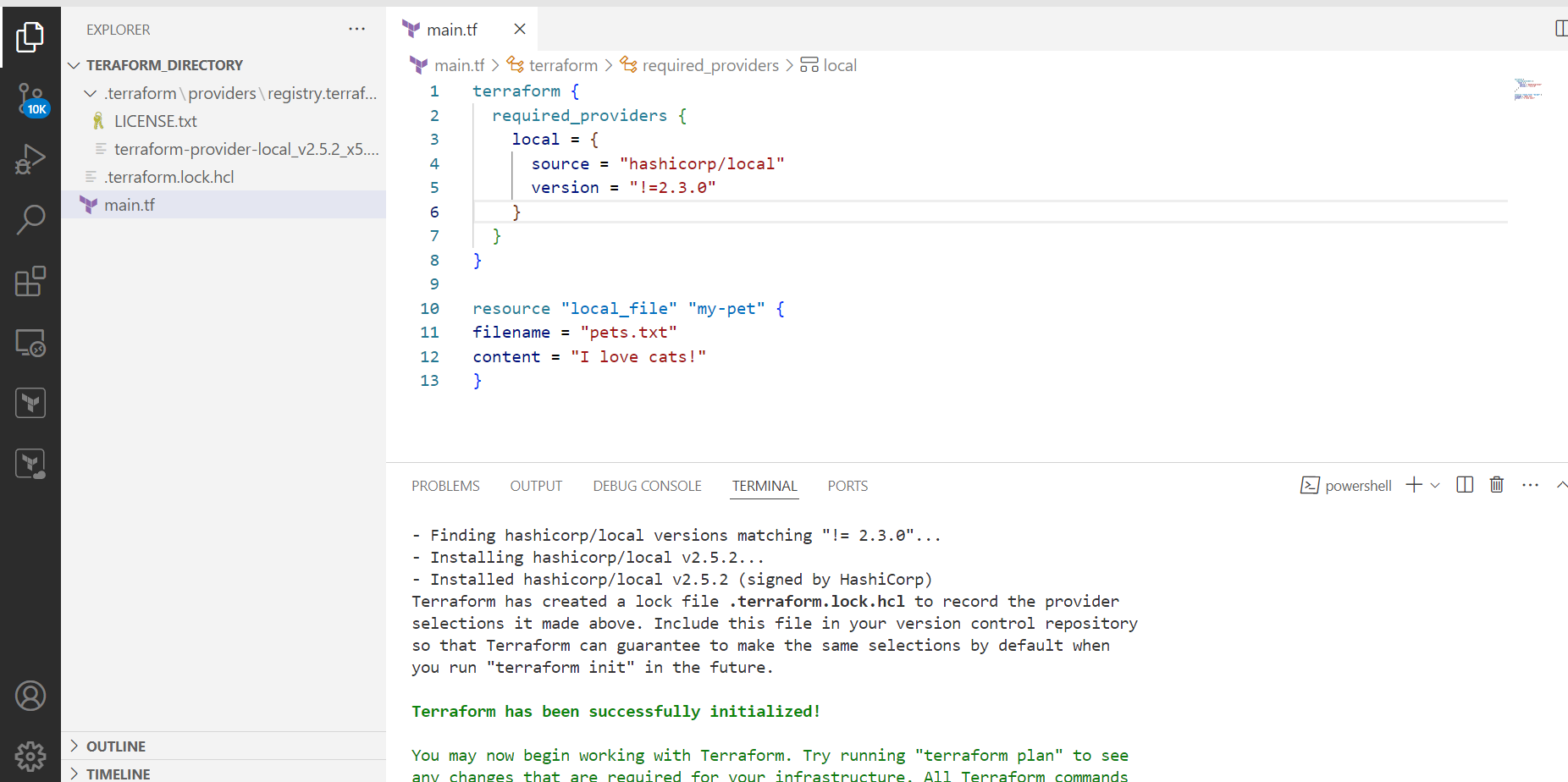
1. Execute the script shown in video.

--> terraform providers

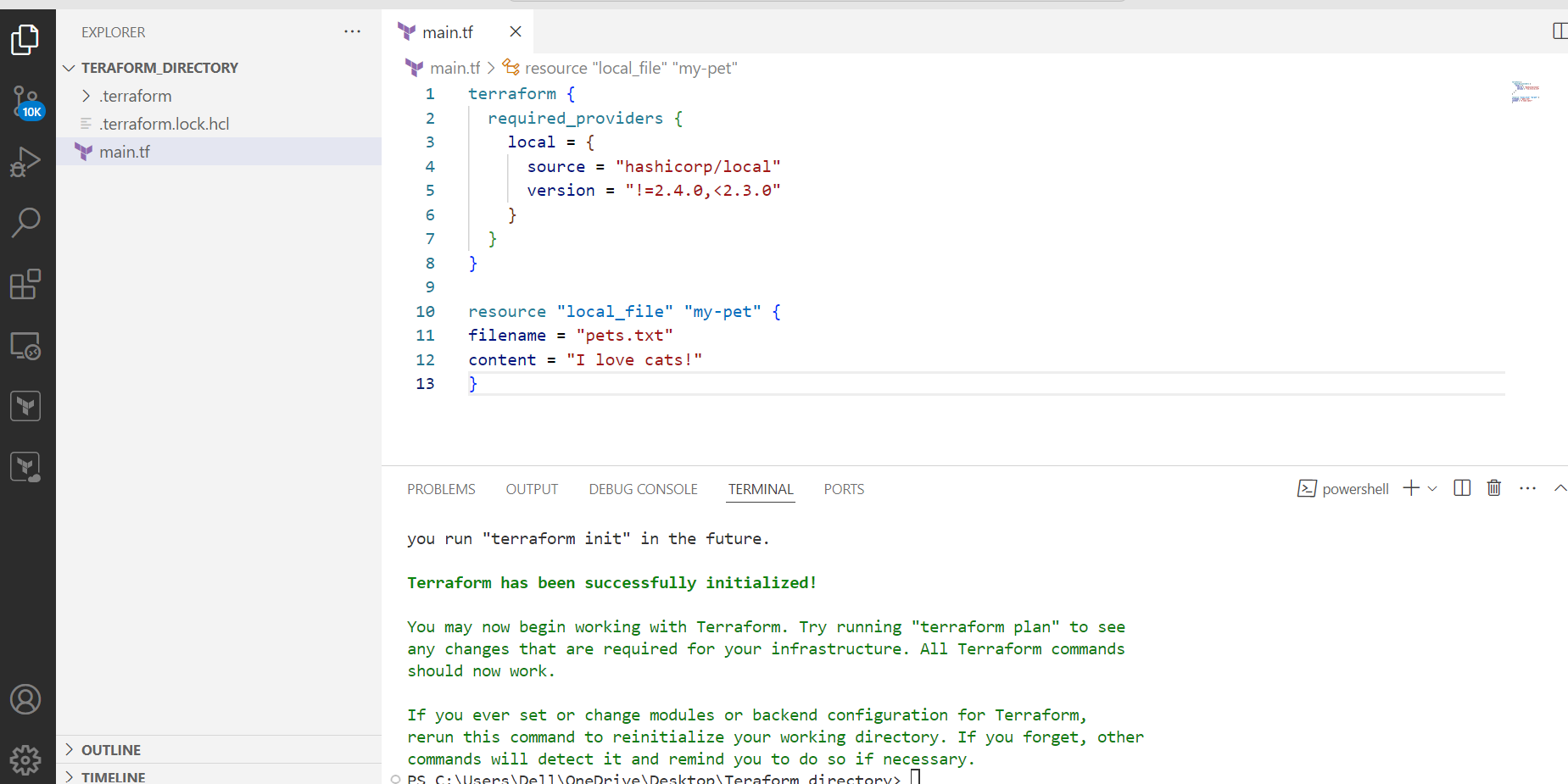
To download the exact version



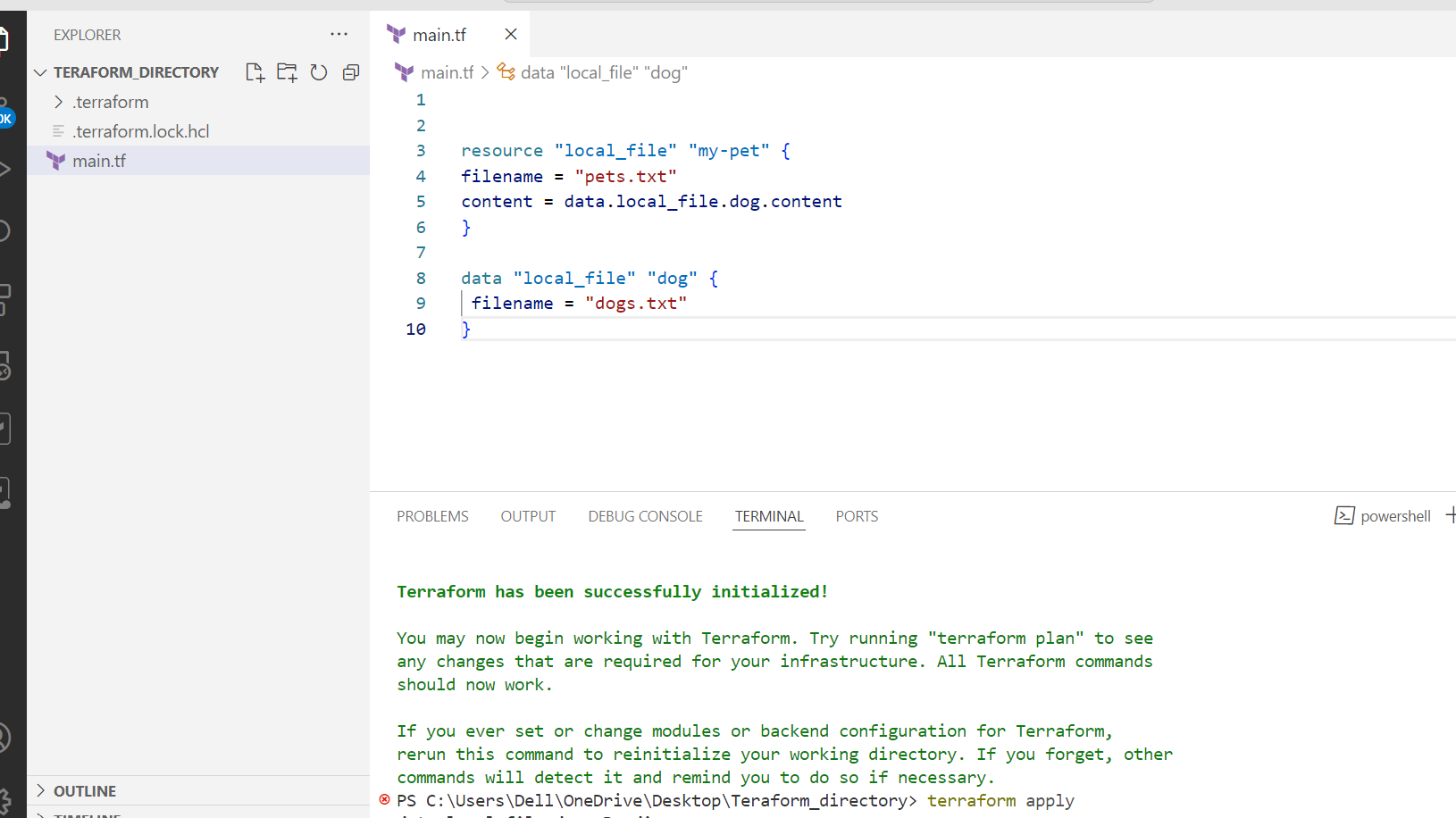
To download the will not mentioned version



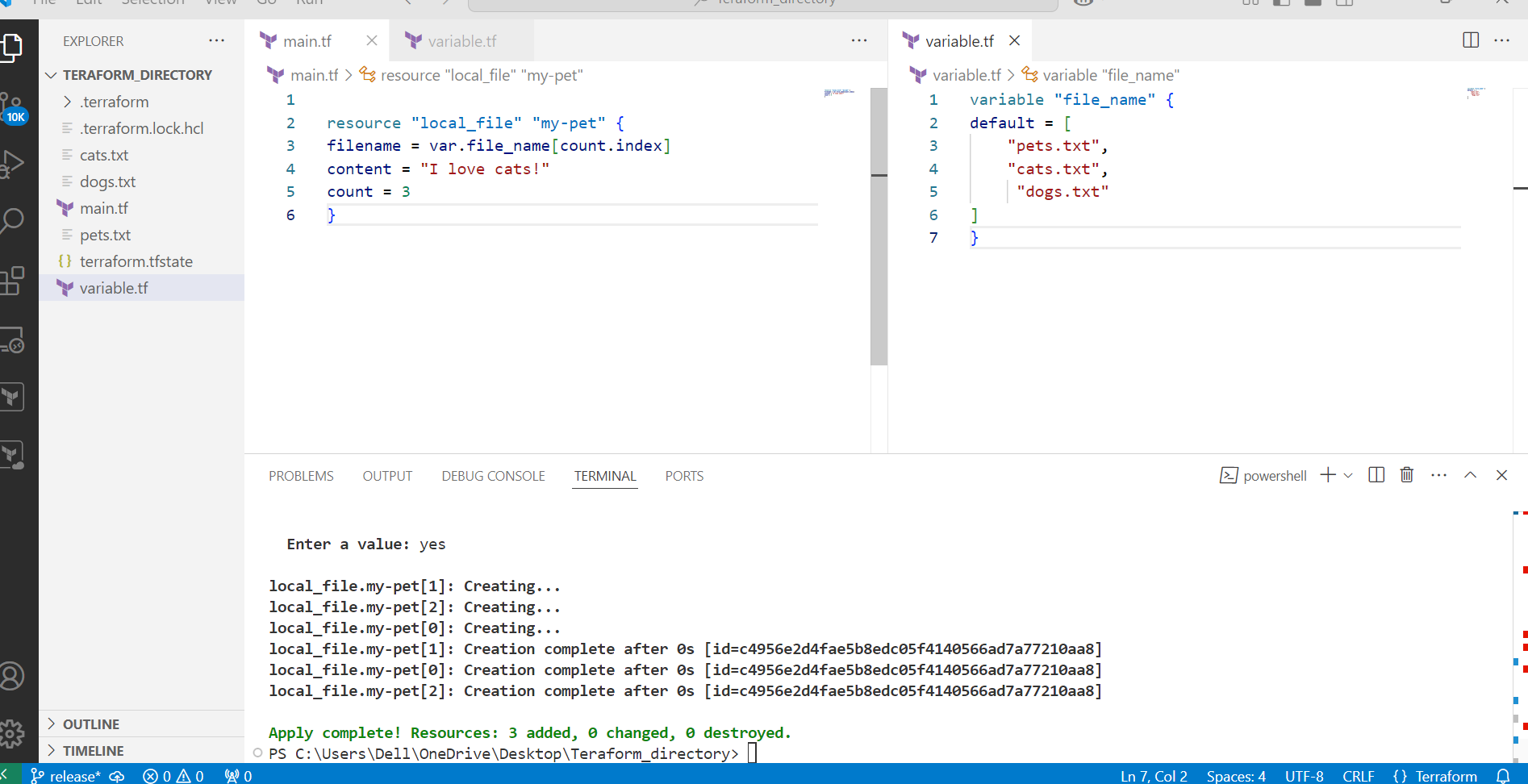
Less then the mentioned versions



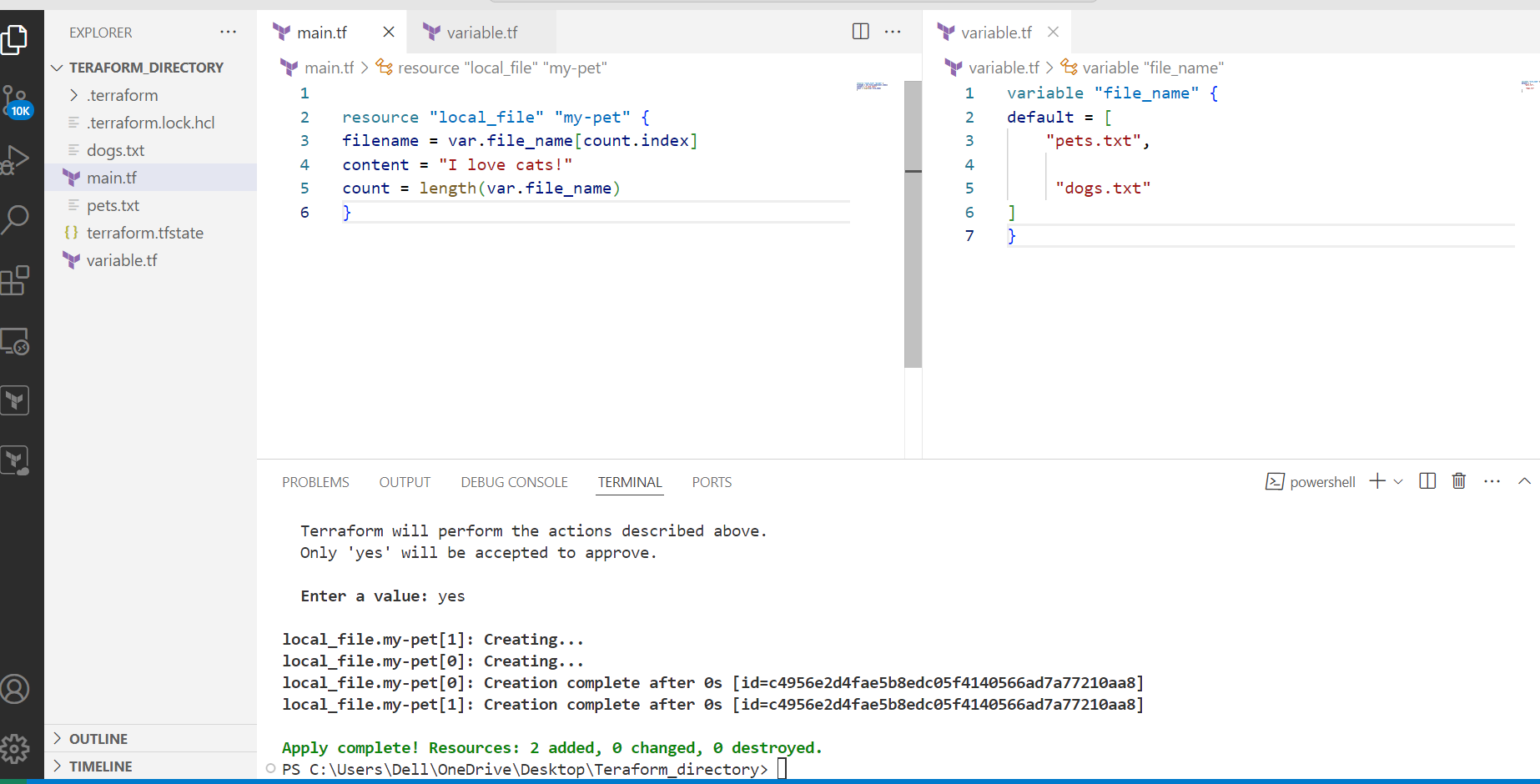
--> data source



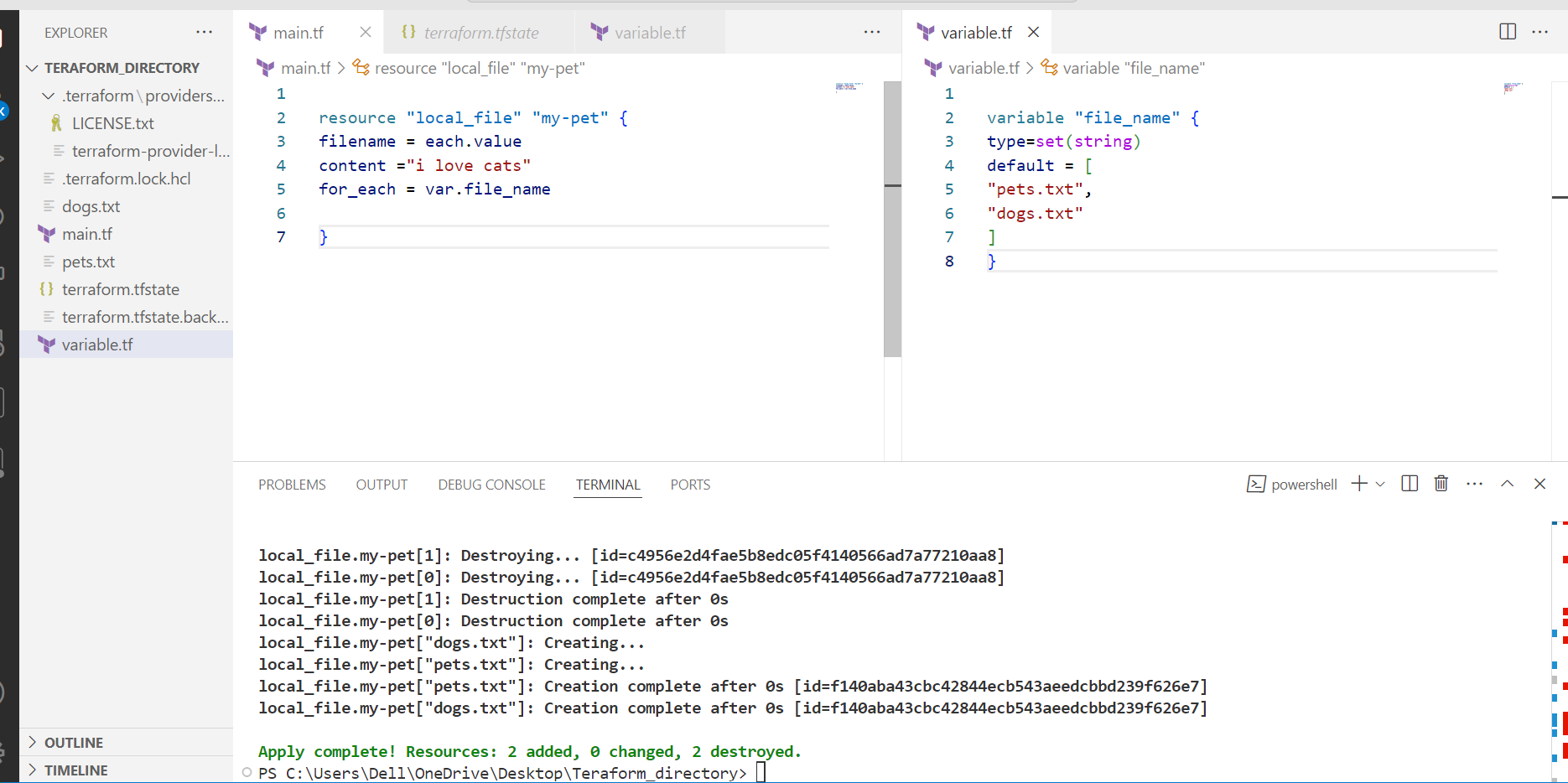
--> Count



Count= length(var.file\_name)



For\_each:



1. Create CICD pipeline for Nodejs Application. [https://github.com/betawins/Trading-UI.git](https://github.com/betawins/Trading-UI.git" \t "https://app.slack.com/client/T03TRQ064Q0/_blank)
2. Explain 10 Maven commands.

**1. mvn clean: This command cleans the project by deleting the target directory, which contains compiled classes and build artifacts. It's useful when you want to start fresh with your build.  
2. mvn compile: Compiles your project's source code (located in the src/main/java directory by default) into bytecode. The compiled classes are placed in the target/classes directory.  
3. mvn test: Runs the tests in your project. Test classes should be placed in the src/test/java directory. Maven uses testing frameworks like JUnit to execute the tests.  
4. mvn package: Packages your project into a distributable format, like a JAR or WAR file. It compiles the code, runs tests, and creates the artifact in the target directory.  
5. mvn install: Builds the project, runs tests, and installs the artifact in the local Maven repository. This makes it available for other projects on the same machine to use.  
6. mvn dependency:tree: Generates a tree-like view of your project's dependencies, showing their relationships and versions. It helps you understand the entire dependency hierarchy.  
7. mvn clean install: A common combination of commands. It cleans the project, then builds, tests, and installs the artifact.  
8. mvn clean package -DskipTests: Builds the project and packages it, skipping the test execution. This is useful when you want a quick build without running tests.  
9. mvn dependency:resolve: Resolves and downloads project dependencies without building the project. This ensures that required dependencies are available locally.  
10. mvn archetype:generate: Generates a new Maven project from a predefined template (archetype). This is helpful to quickly set up a project structure with common configurations.**