**Maven and Terraform**

You can install Maven on Ubuntu using the following steps:

### **1. Update Package Index**

sudo apt update

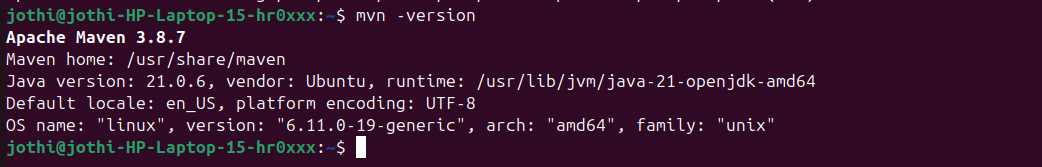
### **2. Install Maven**

sudo apt install maven -y

### **3. Verify Installation**

mvn -version

This should display the installed Maven version.



**Docker file for maven:**

# Use OpenJDK 17 base image (lightweight version)

FROM openjdk:17-jdk-slim

# Set the working directory inside the container

WORKDIR /app

# Copy the built JAR file into the container

COPY target/bookmydr-0.0.1-SNAPSHOT.jar bookmydr-0.0.1-SNAPSHOT.jar

# Expose port 8080 for the application

EXPOSE 8080

# Command to run the application

CMD ["java", "-jar", "bookmydr-0.0.1-SNAPSHOT.jar"]

### **Terraform Workflow Overview**

Terraform follows a standard **Infrastructure as Code (IaC)** workflow that includes initialization, planning, applying changes, and destroying infrastructure. Below are the main steps:

### **1. Write Configuration (.tf files)**

* Define the infrastructure in Terraform configuration files (.tf).

Example (main.tf for AWS EC2 instance):  
 hcl  
provider "aws" {

region = "us-east-1"

}

resource "aws\_instance" "example" {

ami = "ami-0c55b159cbfafe1f0"

instance\_type = "t2.micro"

}

### **2. Initialize (terraform init)**

* Initializes Terraform by downloading necessary provider plugins.

Run:  
  
terraform init

* Output:  
   nginx  
  Initializing provider plugins...

Terraform has been successfully initialized!

### **3. Plan (terraform plan)**

* Shows the execution plan (what Terraform will create/update/destroy).

Run:  
  
terraform plan

Output example:  
 arduino  
  
+ aws\_instance.example

ami: "ami-0c55b159cbfafe1f0"

instance\_type: "t2.micro"

### **4. Apply (terraform apply)**

* Applies the configuration and provisions the infrastructure.

Run:  
terraform apply -auto-approve

Output example:  
 yaml  
  
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

### **5. Verify the Infrastructure**

* Use cloud provider’s dashboard (AWS, Azure, GCP) or CLI to check the resources.

Example AWS CLI:  
aws ec2 describe-instances

### **6. Modify Infrastructure**

* Edit the .tf files to update configurations.
* Run terraform plan to preview changes.
* Apply changes with terraform apply.

### **7. Destroy (terraform destroy)**

* Deletes all managed infrastructure.

Run:

terraform destroy -auto-approve

Output example:  
 yaml  
  
Destroy complete! Resources: 1 destroyed.

### **Terraform Workflow Summary**

| **Step** | **Command** | **Purpose** |
| --- | --- | --- |
| **1. Write Configuration** | main.tf | Define resources |
| **2. Initialize** | terraform init | Initialize plugins |
| **3. Plan** | terraform plan | Preview changes |
| **4. Apply** | terraform apply | Deploy infrastructure |
| **5. Verify** | - | Check cloud provider |
| **6. Modify** | Edit .tf | Update resources |
| **7. Destroy** | terraform destroy | Remove resources |

This workflow ensures **efficient** and **repeatable** infrastructure deployment. 🚀

### **Variables in Terraform**

Terraform allows the use of **variables** to make configurations reusable and dynamic. Variables help in managing infrastructure efficiently by avoiding hardcoded values.

### **1. Defining Variables**

Variables in Terraform can be defined in a .tf file using the variable keyword.

Example (variables.tf):

hcl

CopyEdit

variable "aws\_region" {

description = "AWS region"

type = string

default = "us-east-1"

}

variable "instance\_type" {

description = "EC2 instance type"

type = string

default = "t2.micro"

}

### **2. Using Variables in Configuration**

In the main Terraform configuration file, use the var. prefix to reference variables.

Example (main.tf):

hcl

CopyEdit

provider "aws" {

region = var.aws\_region

}

resource "aws\_instance" "example" {

ami = "ami-0c55b159cbfafe1f0"

instance\_type = var.instance\_type

}

### **3. Passing Values to Variables**

There are multiple ways to provide values to Terraform variables:

#### **(a) Using a .tfvars File**

Create a terraform.tfvars file:

aws\_region = "us-west-2"

instance\_type = "t3.micro"

Then, apply Terraform:

terraform apply -var-file="terraform.tfvars"

#### **(b) Using Command-Line Flags**

Pass variables directly while applying Terraform:

terraform apply -var "aws\_region=us-west-2" -var "instance\_type=t3.micro"

#### **(c) Using Environment Variables**

Export the variables before running Terraform:

export TF\_VAR\_aws\_region="us-west-2"

export TF\_VAR\_instance\_type="t3.micro"

terraform apply

### 

### 

### 

### 

### **4. Variable Types in Terraform**

Terraform supports multiple types of variables:

| **Type** | **Example** |
| --- | --- |
| **String** | type = string |
| **Number** | type = number |
| **Boolean** | type = bool |
| **List** | type = list(string) |
| **Map** | type = map(string) |

Example for list and map:

hcl

variable "instance\_types" {

type = list(string)

default = ["t2.micro", "t3.micro", "t3.small"]

}

variable "ami\_map" {

type = map(string)

default = {

us-east-1 = "ami-0c55b159cbfafe1f0"

us-west-2 = "ami-0d5eff06f840b45e9"

}

}

Usage:

hcl

instance\_type = var.instance\_types[0]

ami = var.ami\_map["us-east-1"]

### **5. Output Variables**

To display values after applying Terraform, use output.

Example (outputs.tf):

Hcl

output "instance\_id" {

value = aws\_instance.example.id

}

Run:

terraform apply

terraform output

### **Summary**

| **Step** | **Command** | **Purpose** |
| --- | --- | --- |
| **1. Define Variables** | variables.tf | Create reusable variables |
| **2. Use Variables** | main.tf | Reference using var. |
| **3. Pass Values** | .tfvars / CLI / Env Vars | Assign values dynamically |
| **4. Output Variables** | outputs.tf | Display resource details |

Pre-Requisites