Which location is the terraform state file stored by default? The state file is created within the configuration directory.
Which option should we use to disable state? We cant disable state
Which format is the state file stored in by default? Json
Which of the following commands does NOT refresh the state? The terraform init command does not use the state file.
What is the name of the state file that is created by default? Terraform.tfstate

terraform init command does not create a state file. For a state file to be created, you must run terraform apply must be run at least once.
Run the terraform show command and identify the id created for the resource called speed_force.

```
iac-server $ terraform show
No state.
iac-server $
```

Now, run terraform apply in this directory.

Main.tf

```
resource "local_file" "speed_force" {
   filename = "/root/speed-force"
   content = "speed-force"
}
```

Reverse-flash.tf

```
resource "local_file" "reverse-flash" {
   filename = "/root/reverse-flash"
   content = "reverse-flash"
}
```

Riddler.tf

```
resource "local_file" "riddler" {
   filename = "/root/riddler"
   content = "riddler"
}
```

Zoom.tf

```
resource "local_file" "zoom" {
   filename = "/root/zoom"
   content = "zoom"
}
```

```
iac-server $ terraform show
# local_file.reverse-flash:
resource "local_file" "reverse-flash" {
    content              = "reverse-flash"
    content_base64sha256 = "Kace3gKIeU2ZCrzpIfitqTpaWRR2kK9h1SfG7HQX9g8="
    content_base64sha512 = "B02UG9wp6Czo23pn2gT/+YreJPaAoMf5fvjDpnymQiXsSfNSHvLfhyn4qXmNQUqborgGFq3imi6
    content_md5          = "f6cc31cf8b2e4d5868b646567f2c8edb"
    content_sha1         = "eebf1b0eee8ccea695bf7925def3d540801e16c7"
    content_sha256       = "29a71ede0288794d990abce921f8ada93a5a59147690af61d527c6ec7417f60f"
    content_sha512       = "074d941bdc29e82ce8db7a67da04fff98ade24f680a0c7f97ef8c3a67ca64225ec49f3521ef
e29a2eb0329ec66074273e"
    directory_permission = "0777"
    file_permission      = "0777"
    filename             = "/root/reverse-flash"
    id                   = "eebf1b0eee8ccea695bf7925def3d540801e16c7"
}

# local_file.riddler:
resource "local_file" "riddler" {
    content              = "riddler"
    content_base64sha256 = "ZovOaJeJQ6wk+LukJRF4IvHfbvLwrjID3HoP9Mrvo6A="
    content_base64sha512 = "ZzOhD2IhUO4zPafhb8xoi1zMpVzfiOcprBlhCY7waUIS1/zHmvtbVYqS5aQdXnTQEFjEmuvjlwq
    content_md5          = "426c70c360c4b0c5ef58e6dc535cf520"
    content_sha1         = "4a99c9eed6c660f5874cc2505558d5abf940a498"
    content_sha256       = "668bce68978943ac24f8bba425117822f1df6ef2f0ae3203dc7a0ff4caefa3a0"
    content_sha512       = "6733a10f622150ee333da7e16fcc688b5ccca55cdf88e729ac1961098ef0694212d7fcc79af
e3970a9a57d904ac3ffb4c"
    directory_permission = "0777"
    file_permission      = "0777"
    filename             = "/root/riddler"
    id                   = "4a99c9eed6c660f5874cc2505558d5abf940a498"
}

# local_file.speed_force:
resource "local_file" "speed_force" {
    content              = "speed-force"
    content_base64sha256 = "+hI5F86aVJG7nQ6K0VEOJHTIhlj5aRLnpODNbyZExtI="
```

We have just added a new configuration file called aws-infra.tf into this configuration directory and provisioned the resources.

Aws-infra.tf

```
resource "aws_instance" "dev-server" {
  instance_type = "t2.micro"
  ami       = "ami-02cff456777cd"
}
resource "aws_s3_bucket" "falshpoint"  {
  bucket = "project-flashpoint-paradox"
}
```

Inspect the terraform.tfstate file or run terraform show command.
You will notice that all the attribute details for all the resources created by this configuration is now printed on the screen!
Among them is an EC2 Instance which is created by the resource called dev-server. See if you can find out the private_ip for the instance that was created.

```
iac-server $ terraform show
# aws_instance.dev-server: (tainted)
resource "aws_instance" "dev-server" {
    ami                          = "ami-02cff456777cd"
    arn                          = "arn:aws:ec2:us-east-1::instance/i-dc1d358b978e8d9da"
    associate_public_ip_address  = true
    availability_zone            = "us-east-1a"
    ebs_optimized                = false
    get_password_data            = false
    id                           = "i-dc1d358b978e8d9da"
    instance_state               = "running"
    instance_type                = "t2.micro"
    ipv6_address_count           = 0
    ipv6_addresses               = []
    key_name                     = "None"
    monitoring                   = false
    placement_partition_number   = 0
    primary_network_interface_id = "eni-c038d465"
    private_dns                  = "ip-10-59-254-251.ec2.internal"
    private_ip                   = "10.59.254.251"
    public_dns                   = "ec2-54-214-116-211.compute-1.amazonaws.com"
    public_ip                    = "54.214.116.211"
    secondary_private_ips        = []
    security_groups              = []
    source_dest_check            = true
    subnet_id                    = "subnet-93a817c3"
    tenancy                      = "default"
    user_data_replace_on_change  = false
    vpc_security_group_ids       = []

    root_block_device {
        delete_on_termination = true
        device_name           = "/dev/sda1"
```

Terraform commands

Which command can be used to create a visual representation of our terraform resources?
terraform graph

We have created a configuration directory /root/terraform-projects/project-shazam. The configuration file inside will be used to create an RSA type private key and then a certificate signing request or a csr using this key.
However, there is an error with the configuration.
Use the terraform validate command, troubleshoot, and fix the issue.
You don't have to create the resources yet! You only need to fix the errors reported by terraform validate.

Main.tf

```
resource "local_file" "key_data" {
      filename      = "/tmp/.pki/private_key.pem"
      content = tls_private_key.private_key.private_key_pem
      file_permission =  "0400"
}
resource "tls_private_key" "private_key" {
  algorithm   = "RSA"
  dsa_bits  = 2048
  ecdsa_curve = "P384"
}
resource "tls_cert_request" "csr" {
  private_key_pem = file("/tmp/.pki/private_key.pem")
```

```
 depends_on = [ local_file.key_data ]

 subject {
  common_name  = "flexit.com"
  organization = "FlexIT Consulting Services"
 }
}
```

The argument called dsa_bits in the resource block for creating the private key is incorrect. For RSA algorithm, the correct argument to use is rsa_bits.

```
iac-server $ terraform validate

Error: Unsupported argument

  on main.tf line 8, in resource "tls_private_key" "private_key":
   8:   dsa_bits  = 2048

An argument named "dsa_bits" is not expected here. Did you mean "rsa_bits"?
```

Main.tf

```
resource "local_file" "key_data" {
      filename      = "/tmp/.pki/private_key.pem"
      content = tls_private_key.private_key.private_key_pem
      file_permission =  "0400"
}
resource "tls_private_key" "private_key" {
 algorithm   = "RSA"
 rsa_bits  = 2048
 ecdsa_curve = "P384"
}
resource "tls_cert_request" "csr" {
 private_key_pem = file("/tmp/.pki/private_key.pem")
 depends_on = [ local_file.key_data ]

 subject {
  common_name  = "flexit.com"
  organization = "FlexIT Consulting Services"
 }
}
```

Now run terraform plan and generate a configuration plan.

```
iac-server $ terraform validate
Success! The configuration is valid.

iac-server $ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.


------------------------------------------------------------------------

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # local_file.key_data will be created
  + resource "local_file" "key_data" {
      + content              = (known after apply)
      + content_base64sha256 = (known after apply)
      + content_base64sha512 = (known after apply)
      + content_md5          = (known after apply)
      + content_sha1         = (known after apply)
      + content_sha256       = (known after apply)
      + content_sha512       = (known after apply)
      + directory_permission = "0777"
      + file_permission      = "0400"
      + filename             = "/tmp/.pki/private_key.pem"
```

```
Plan: 3 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

tls_private_key.private_key: Creating...
tls_private_key.private_key: Creation complete after 1s [id=c67ca357c6dd26d979e
local_file.key_data: Creating...
local_file.key_data: Creation complete after 0s [id=b7e7c0f0d3d1fde76efd707712d

Error: Provider produced inconsistent final plan

When expanding the plan for tls_cert_request.csr to include new values learned
so far during apply, provider "registry.terraform.io/hashicorp/tls" produced
an invalid new value for .private_key_pem: inconsistent values for sensitive
attribute.

This is a bug in the provider, which should be reported in the provider's own
issue tracker.
```

The terraform apply failed in spite of our validation working! This is because the validate command only carries out a general verification of the configuration. It validated the resource block and the argument syntax but not the values the arguments expect for a specific resource!

The error in the configuration is inside the resource block for the tls_private_key type resource.
It contains the configuration that we needed for generating rsa type key..
Inspect the resource block and fix the issue.

The Algorithm used by the tls_private_key is RSA but the configuration also mentions ecdsa_curve argument.

Main.tf

```
resource "local_file" "key_data" {
      filename      = "/tmp/.pki/private_key.pem"
      content = tls_private_key.private_key.private_key_pem
      file_permission =  "0400"
}
resource "tls_private_key" "private_key" {
 algorithm   = "RSA"
 rsa_bits  = 2048
}
resource "tls_cert_request" "csr" {
 private_key_pem = file("/tmp/.pki/private_key.pem")
 depends_on = [ local_file.key_data ]

 subject {
  common_name  = "flexit.com"
  organization = "FlexIT Consulting Services"
 }
}
```

Now format the main.tf file into a canonical format.
Run the terraform fmt command in the configuration directory.

```
iac-server $ terraform fmt
main.tf
```

Now, navigate to the directory /root/terraform-projects/project-a. We have already created the resources specified in this configuration.
Fetch details from the state file and identify the value of the filename argument.

```
iac-server $ terraform show
# local_file.data:
resource "local_file" "data" {
    content              = "You've to write this code."
    content_base64sha256 = "FZy1Bn26UEIGgBj6E/3aMFPPo0z8wmDp4ljqh+WYuWQ=
    content_base64sha512 = "4pC7k3YFVhg+Cu2fo0I5b1/h1JH9HpbFLHG2MFuCEHh5
hV31i5Q=="
    content_md5          = "55403026bfc0ce8205d712ed68891251"
    content_sha1         = "fc307f99d6490d988433ce246d60eb4ca005a87c"
    content_sha256       = "159cb5067dba5042068018fa13fdda3053cfa34cfcc2
    content_sha512       = "e290bb93760556183e0aed9fa342396f5fe1d491fd1e
e6d0a924c01aaf96b631cdb62e0aad9fb92bc7e21577d62e5"
    directory_permission = "0777"
    file_permission      = "0777"
    filename             = "/root/codes"
    id                   = "fc307f99d6490d988433ce246d60eb4ca005a87c"
```

Data.tf

```
resource "local_file" "data" {
        filename = "/opt/codes"
        content = "You've to write this code."
}
```

Providers are plugins .
Which one is a valid sub-command of the terraform providers command? mirror

A new configuration directory /root/terraform-projects/provider has been created. We have already run the terraform init command.
Now check the provider plugins that have been downloaded from the command line utility (instead of inspecting the .terraform directory). After that choose the correct option.
Run the terraform providers command to see the options.

```
iac-server $ terraform providers

Providers required by configuration:
.
├── provider[registry.terraform.io/hashicorp/local]
└── provider[registry.terraform.io/hashicorp/aws]
```

Lifecycle rules :
We have a directory created called /root/terraform-projects/project-mysterio. The main.tf file already has a couple of resource blocks.

Main.tf

```
resource "local_file" "file" {
   filename = var.filename
   file_permission =  var.permission
   content = random_string.string.id

}

resource "random_string" "string" {
   length = var.length
   keepers = {
      length = var.length
   }

}
```

Variables.tf

```
variable "length" {
   default = 10

}
variable "filename" {
   default = "/root/random_text"
}
variable "content" {
   default = "This file contains a single line of data"
}
variable "permission" {
   default = 0700
}
```

Now, create these two resources that have been defined in this configuration file.

```
Plan: 2 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

random_string.string: Creating...
random_string.string: Creation complete after 0s [id=ia#mx!pJ!Y]
local_file.file: Creating...
local_file.file: Creation complete after 0s [id=ea37a63708e02d5f1b8

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
```

Which resource is created first in this case?
There is an implicit dependency defined between the resources.

We have modified the resource configuration again. Run a terraform plan now. What would happen?

```
-/+ resource "local_file" "file" {
      ~ content              = "ia#mx!pJ!Y" -> (known after apply) # forces replacement
      ~ content_base64sha256 = "l2dLaun+KnJNC1VOVhfSVJLFf1BA3QIZNra4s3A7sp4=" -> (known after a
      ~ content_base64sha512 = "I6gtyoZbqNHNypK8UDEgZm88Sln/WX95X1/r/ybF8nabLGiY+2/Pc7RPoVq9LMy
ter apply)
      ~ content_md5          = "13bb28b82d1231ab3641a173b22aba4d" -> (known after apply)
      ~ content_sha1         = "ea37a63708e02d5f1b870259756f10e87264a7c7" -> (known after apply
      ~ content_sha256       = "97674b6ae9fe2a724d0b554e5617d25492c57f5040dd021936b6b8b3703bb29
      ~ content_sha512       = "23a82dca865ba8d1cdca92bc503120666f3c4a59ff597f795f5febff26c5f27
30f6ce001631f4ca40c4d5f16f" -> (known after apply)
        directory_permission = "0777"
      ~ file_permission      = "700" -> "770" # forces replacement
        filename             = "/root/random_text"
      ~ id                   = "ea37a63708e02d5f1b870259756f10e87264a7c7" -> (known after apply
    }

  # random_string.string must be replaced
-/+ resource "random_string" "string" {
      ~ id          = "ia#mx!pJ!Y" -> (known after apply)
      ~ keepers     = { # forces replacement
          ~ "length" = "10" -> "12"
        }
      ~ length      = 10 -> 12 # forces replacement
        lower       = true
        min_lower   = 0
        min_numeric = 0
        min_special = 0
        min_upper   = 0
        number      = true
        numeric     = true
      ~ result      = "ia#mx!pJ!Y" -> (known after apply)
        special     = true
        upper       = true
    }

Plan: 2 to add, 0 to change, 2 to destroy.
```

Why is the string resource being re-created? Value for argument called keeper is changed .
Inspect the line on terraform plan that reads forces replacement

All the resources for the random provider can be recreated by using a map type argument called keepers. A change in the value will force the resource to be recreated.
This argument accepts arbitrary key/value pairs and in our example, it is set to the key called length whose value was updated from 10 to 12 in the variables.tf file.
Running a terraform apply now will destroy the current random_string resource and then create a new one with the length that is 12 characters long.

Let's change the order in which the resource called string is recreated. Update the configuration so that when applied, a new random string is created first before the old one is destroyed.
When ready, apply the changes with terraform apply

Use the lifecycle rule of create_before_destroy.
Main.tf

```
resource "local_file" "file" {
   filename = var.filename
   file_permission =  var.permission
   content = random_string.string.id

}
```

```
resource "random_string" "string" {
   length = var.length
   keepers = {
      length = var.length
   }
   lifecycle{
      create_before_destroy = true
   }
}
```

```
Plan: 2 to add, 0 to change, 2 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

local_file.file: Destroying... [id=ea37a63708e02d5f1b870259756f10e87264a7c7]
local_file.file: Destruction complete after 0s
random_string.string: Creating...
random_string.string: Creation complete after 0s [id=r-flwSX$AklK]
local_file.file: Creating...
local_file.file: Creation complete after 0s [id=75eb3bafa96896ccfc402127549d91b1f376e016]
random_string.string: Destroying... [id=ia#mx!pJ!Y]
random_string.string: Destruction complete after 0s
```

The resource block for the file resource has been updated! This will force the resource to be recreated during the next apply! But, before that, let's also add a lifecycle rule of create_before_destroy to this resource block. When ready, apply the changes with terraform apply
Important: Once the lifecycle rule has been added, only run the apply command once. We will learn why soon.

Update main.tf file adding lifecycle rule under the resource local_file.

Main.tf

```
resource "local_file" "file" {
   filename = var.filename
   file_permission =  var.permission
   content = "This is a random string - ${random_string.string.id}"
   lifecycle {
      create_before_destroy = true
   }

}

resource "random_string" "string" {
   length = var.length
   keepers = {
      length = var.length
   }
   lifecycle {
      create_before_destroy =  true
   }
}
```

```
Plan: 1 to add, 0 to change, 1 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

local_file.file: Creating...
local_file.file: Creation complete after 0s [id=0673fddfd300da44
local_file.file: Destroying... [id=75eb3bafa96896ccfc402127549d9
local_file.file: Destruction complete after 0s

Apply complete! Resources: 1 added, 0 changed, 1 destroyed.
```

Great! We have now added the lifecycle rule and forced the resources to be created first and then destroyed.
What is the id of the file resource we just created?
Run terraform show or terraform state show local_file.file to find out.

```
iac-server $ terraform state show local_file.file
# local_file.file:
resource "local_file" "file" {
    content              = "This is a random string - r-flwSX$AklK"
    content_base64sha256 = "HMVLLr4Btx4F2NYQ3DTGw3dhS2IGTkRaAC79BuPurOc="
    content_base64sha512 = "8t2L+SrjGdDvmcSnPN6axv1ug4GgvOFnR8uW+o9JdrE9nZbh+TrgAg67pG
    content_md5          = "4d3bdff4e33883ec76ec6ced26484c59"
    content_sha1         = "0673fddfd300da445e0f2dacda8ead58aa55a21e"
    content_sha256       = "1cc54b2ebe01b71e05d8d610dc34c6c377614b62064e445a002efd06e3
    content_sha512       = "f2dd8bf92ae319d0ef99c4a73cde9ac6fd6e8381a0bce16747cb96fa8f
db580c7995ec36a873a4c1"
    directory_permission = "0777"
    file_permission      = "770"
    filename             = "/root/random_text"
    id                   = "0673fddfd300da445e0f2dacda8ead58aa55a21e"
}
```

If you observe the output of the previous apply (scroll up!), you will see that the lifecycle rule we applied caused the local file to the created first and the same file to be destroyed during the recreate operation.
This goes to show that it is not always advisable to use this rule!
In this example, the filename argument for the local_file resource has to be unique which means that we cannot have two instances of the same file created at the same time!
The random_string resource on the other hand is a logical resource that is only recorded in the state and does not have such a restriction.
If you run terraform apply again, the file resource will be created as it does not exist currently.

Main.tf

```
resource "random_pet" "super_pet" {
   length = var.length
   prefix = var.prefix

}
```

Now, update the configuration so that the resource super_pet is not destroyed under any circumstances with a terraform apply command.
Add the lifecyle rule prevent_destroy in the config file.

Main.tf

```
resource "random_pet" "super_pet" {
   length = var.length
   prefix = var.prefix
   lifecycle{
      prevent_destroy = true
   }
}
```

Datasources

A data source once created, can be used to create, update, and destroy infrastructure?
A datasource can only read resource data and use that information within terraform.

A data source can be created using the data block.

A new configuration directory has been created at /root/terraform-projects/project-lexcorp. A data source block is defined in the main.tf file to read the contents of an existing file.
There is also an output variable that uses reference expression to print the file content using this data source.
However, there is something wrong!
Troubleshoot and fix the issue.
When ready, run terraform init, plan and apply to create the datasource. The configuration should print the output variable correctly.

Main.tf :

```
output "os-version" {
 value = data.local_file.os.content
}
data "local_file" "os" {
 filename = "/etc/os-release"
}
```

```
iac-server $ terraform apply
data.local_file.os: Refreshing state...

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:

Terraform will perform the following actions:

Plan: 0 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes


Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

Outputs:

os-version = NAME="Ubuntu"
VERSION="18.04.5 LTS (Bionic Beaver)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 18.04.5 LTS"
VERSION_ID="18.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=bionic
UBUNTU_CODENAME=bionic
```

Now let's practice how to work with data sources from other providers.
The next few questions will be based on the aws provider.
Although we have only predominantly worked with local and the random provider, this exercise will help you learn how to work with different data sources using the documentation.

Ebs.tf

```
data "aws_ebs_volume" "gp2_volume" {
  most_recent = true

  filter {
    name   = "volume-type"
    values = ["gp2"]
  }
}
```

Once this data source is created, how do we fetch the Volume Id for the resource that is created in AWS?
volume_id

Another file called s3.tf has now been created. It too has a data source that will be used to read data of an existing s3 bucket.
However, there is a mistake in the argument used. What is wrong here?

```
data "aws_s3_bucket" "selected" {
  bucket_name = "bucket.test.com"
}
```

The argument should be bucket not bucket_name.

Main.tf

```
resource "local_file" "name" {
    filename = "/root/user-data"
    sensitive_content = "password: S3cr3tP@ssw0rd"

}
```

How many files will be created by this configuration? 1

Now add a count argument to create 3 instances of this resource.
When ready, run terraform init, plan and apply

```
resource "local_file" "name" {
    filename = "/root/user-data"
    sensitive_content = "password: S3cr3tP@ssw0rd"
    count   = 3
}
```

The resource local_file.name is now created as a: list ; Since we used count, the resources are now created as list.

```
Plan: 3 to add, 0 to change, 0 to destroy.


Warning: Attribute Deprecated

  on main.tf line 3, in resource "local_file" "name":
   3:     sensitive_content = "password: S3cr3tP@ssw0rd"

Use the `local_sensitive_file` resource instead

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

local_file.name[1]: Creating...
local_file.name[2]: Creating...
local_file.name[0]: Creating...
local_file.name[1]: Creation complete after 0s [id=6b32344cb73c40d126d99ce62309878befca64ce]
local_file.name[2]: Creation complete after 0s [id=6b32344cb73c40d126d99ce62309878befca64ce]
local_file.name[0]: Creation complete after 0s [id=6b32344cb73c40d126d99ce62309878befca64ce]

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
```

How many files were actually created when apply was run? 1

We have now created a variables.tf file in the same configuration directory. Update the main.tf file to make use of the list type variable defined for the filename argument.
Use count to loop through all the elements of this list and do not use hard-coded values.
Use the variable called content for the argument called sensitive_content.

Variables.tf

```
variable "users" {
   type = list
}
variable "content" {
   default = "password: S3cr3tP@ssw0rd"
}
```

Main.tf

```
resource "local_file" "name" {
   filename = var.users[count.index]
   sensitive_content = var.content
   count   = length(var.users)
}
```

We have reverted back to the old configuration file and cleaned up the resources created so far.
A variable called users now has default values added to it.
What type of variable is it?

Variable.tf

```
variable "users" {
   type = list(string)
   default = [ "/root/user10", "/root/user11", "/root/user12", "/root/user10"]
}
variable "content" {
   default = "password: S3cr3tP@ssw0rd"

}
```

Main.tf

```
resource "local_file" "name" {
   filename = "/root/user-data"
   sensitive_content = "password: S3cr3tP@ssw0rd"

}
```

Can the same elements in this list be used as it is for a set instead? A set cannot have duplicates!

Let's do the same exercise as before but this time we will make use of the for_each meta argument to create the files in this configuration.
Just like before don't use any hard-coded values.
Use for_each to loop through the list type variable called users.
Use the variable called content as the value of the argument sensitive_content.
When ready, run terraform init, plan and apply.

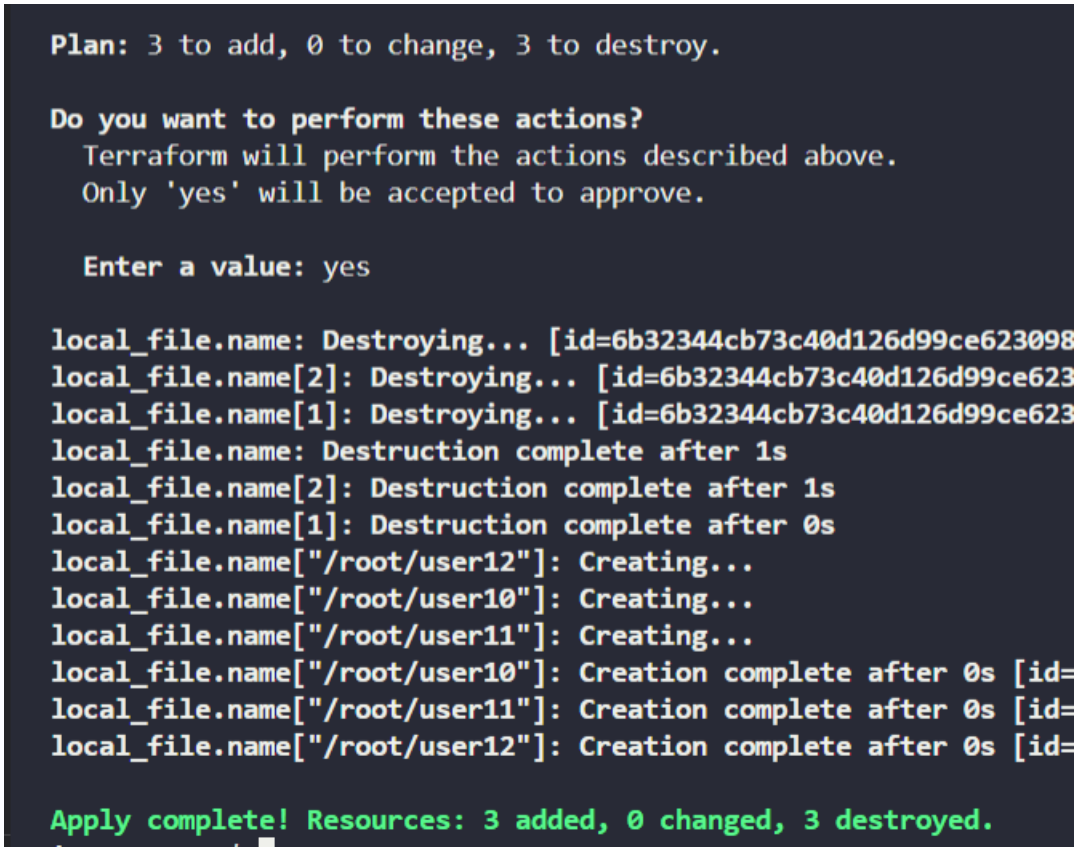
Variable.tf

```
variable "users" {
   type = list(string)
   default = [ "/root/user10", "/root/user11", "/root/user12", "/root/user10"]
}
variable "content" {
   default = "password: S3cr3tP@ssw0rd"

}
```

Main.tf

```
resource "local_file" "name" {
   filename = each.value
   for_each = toset(var.users)
   sensitive_content = var.content
}
```



The resource called name is now created as: map

The resource address with the filename - /root/user11 is now represented as: local_file.name[“/root/user11”]

Main.tf

```
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "1.2.2"
    }
  }
}

resource "local_file" "innovation" {
  filename = var.path
  content  = var.message
}
```

Variables.tf

```
variable "path" {
    default = "/root/session"
}

variable "message" {
    default = "It's time for innovative ideas.\n"
}
```

Now, change to the directory /root/terraform-projects/rotate. We have already initialized the configuration directory using the terraform init command.
Inspect the rotation.tf file and find out the correct version of the provider plugin that is downloaded.
Choose the correct version from the below options:

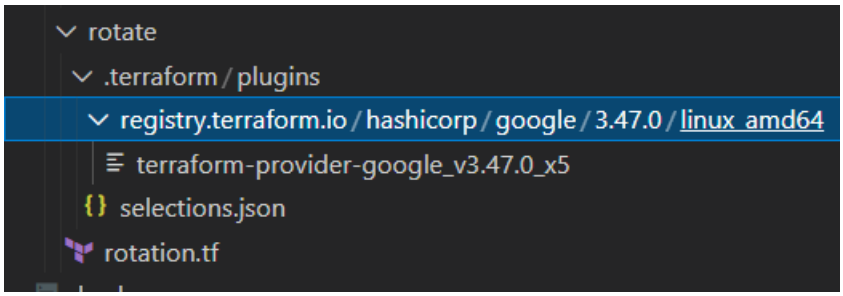
Go onto the .terraform/plugins configuration directory and inspect the correct version which has got downloaded.

Rotation.tf

```
terraform {
 required_providers {
  google = {
   source  = "hashicorp/google"
   version = "> 3.45.0, !=3.46.0, < 3.48.0"
  }
 }
}

resource "google_compute_instance" "special" {
 name         = "aone"
 machine_type = "e2-micro"
 zone         = "us-west1-c"

}
```



Which one of the below is not a valid version constraint operator?

~>

!=

<=

==

>=

== is not valid

We have been working on a project called nautilus under the configuration directory /root/terraform-projects/nautilus.
Due to a version mismatch, we don't want to download the aws provider version 3.17.0. Which version constraint can be used to achieve this?
You can try to add the below options in nautilus.tf to verify the correct syntax.
Operator "!=" excludes an exact version which we will specify. It will not download that version at all.

Nautalius.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = ""
    }
  }
}

resource "aws_ebs_volume" "soft-volume" {
  availability_zone = "us-west-2a"
  size              = 15
  tags = {
    Name = "temporary"
  }
}
```

Version = “!=3.17.0”

Now, navigate to the directory /root/terraform-projects/lexicorp where we have added the configuration files.
Inspect the file and find out which version of providers will be download.
Helm 1.2.4 && Kubernetes 1.13.2

Tecton.tf

```
terraform {
  required_providers {
    k8s = {
      source  = "hashicorp/kubernetes"
      version = "> 1.12.0, != 1.13.1, < 1.13.3 "
    }

    helm = {
      source  = "hashicorp/helm"
      version = "~> 1.2.0"
    }
  }
}
```