1. What is a data source? In what scenarios for example would need to use it?

Data sources allow data to be fetched or computed for use elsewhere in Terraform configuration. Use of data sources allows a Terraform configuration to make use of information defined outside of Terraform, or defined by another separate Terraform configuration.

A data source is accessed via a special kind of resource known as a data resource, declared using a data block:

```
data "aws_ami" "example" {
  most_recent = true

  owners = ["self"]
  tags = {
    Name   = "app-server"
    Tested = "true"
  }
}
```

For example, a data source may retrieve remote state data from a Terraform Cloud workspace, configuration information from Consul, or look up a pre-existing AWS resource by filtering on its attributes and tags.

Every data source in Terraform is mapped to a provider based on longest-prefix matching. For example the aws_ami data source would map to the aws provider (if that exists).

2. What is .tfstate file and why do we need to manage it?

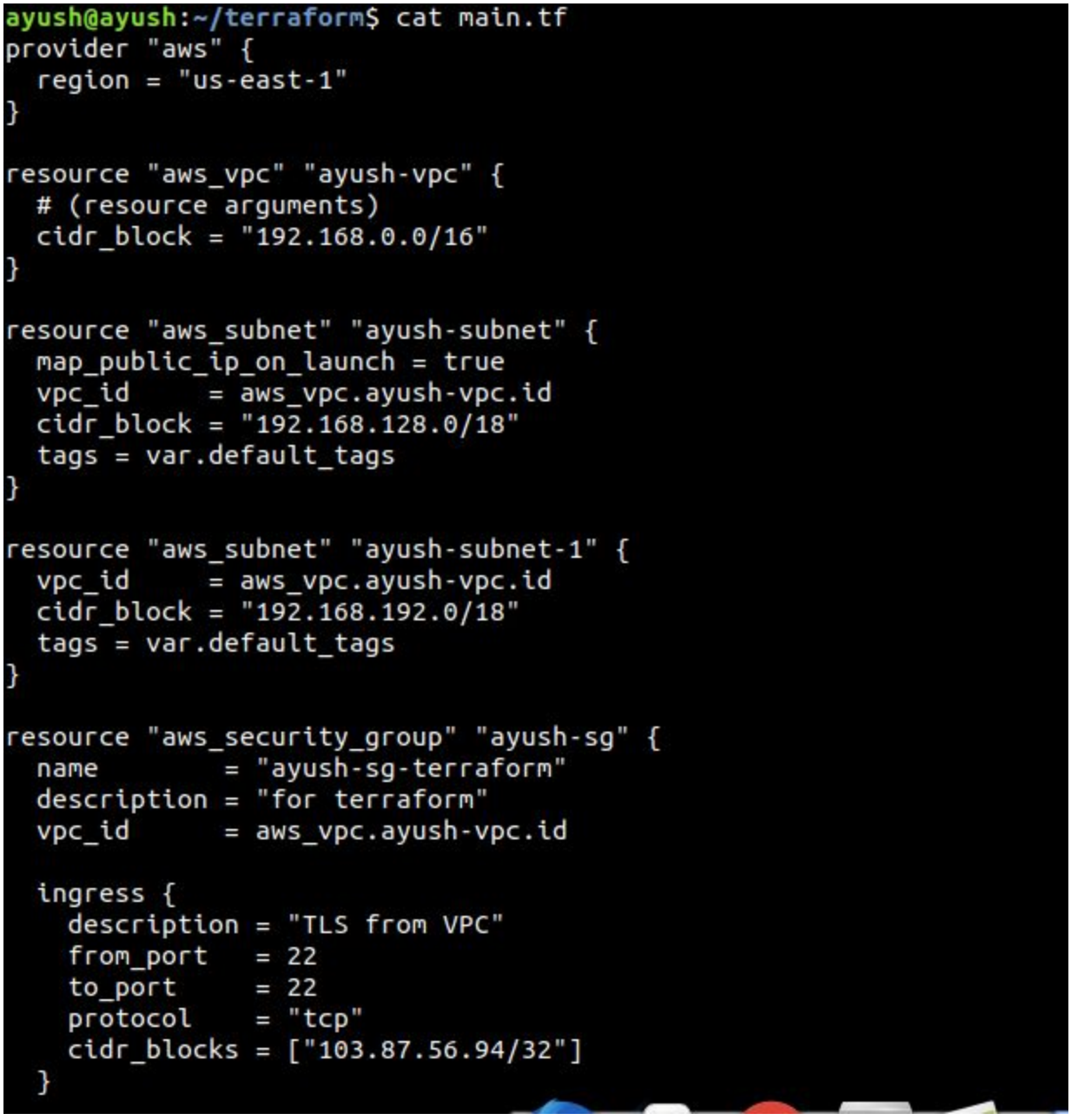
This state is used by Terraform to map real world resources to your configuration, keep track of metadata, and to improve performance for large infrastructures. This state is stored by default in a local file named "terraform. tfstate", but it can also be stored remotely, which works better in a team environment.

If you're using Terraform with AWS, Amazon S3 (Simple Storage Service), which is Amazon's managed file store, is typically your best bet as a remote
backend for the following reasons:

- It's a managed service, so you don't have to deploy and manage extra infrastructure to use it.
- It's designed for 99.999999999% durability and 99.99% availability, which means you don't have to worry too much about data loss or outages.
- It supports encryption, which reduces worries about storing sensitive data in state files. Anyone on your team who has access to that S3 bucket will be able to see the state files in an unencrypted form, so this is still a partial solution, but at least the data will be encrypted at rest (S3 supports server-side encryption using AES-256) and in transit (Terraform uses SSL to read and write data in S3).
- It supports locking via DynamoDB. More on this below.
- It supports versioning, so every revision of your state file is stored, and you can roll back to an older version if something goes wrong.
- It's inexpensive, with most Terraform usage easily fitting into the free tier.

3. Launch an EC2 instance with custom AMI for nginx, attach to the target group and create listener rule in a load balancer.
a. Use s3 backend as well as dynamodb locking
b. Create security group
c. Pass same tags as map to all resources

Ref:-
https://medium.com/@mitesh_shamra/state-management-with-terraform-9f13497e54cf

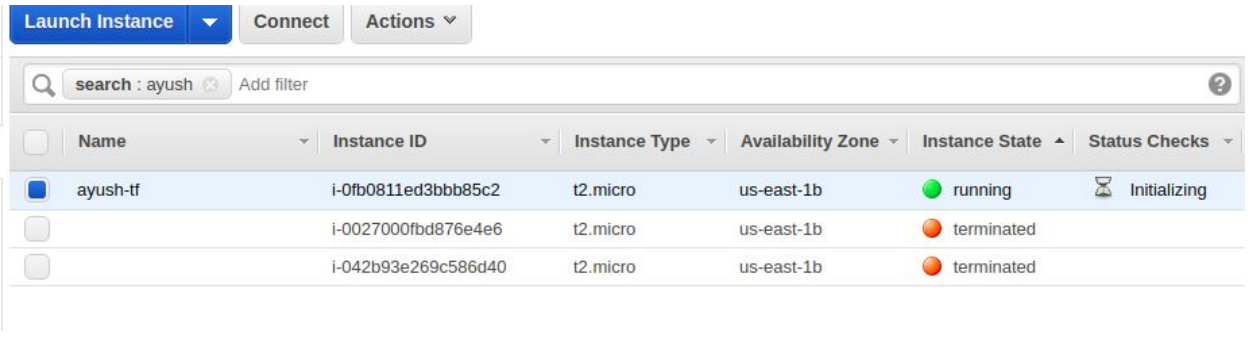


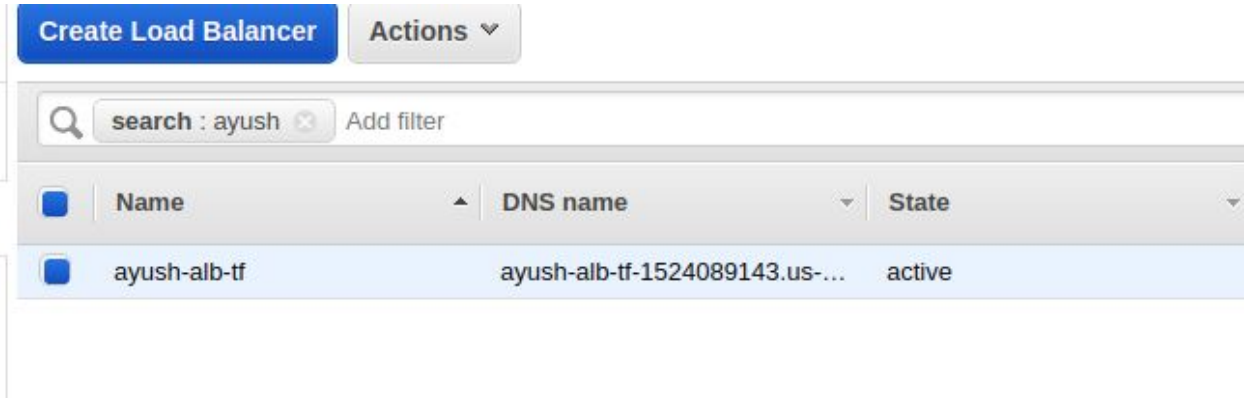
```
ayush@ayush:~/terraform$ cat main.tf
provider "aws" {
  region = "us-east-1"
}

resource "aws_vpc" "ayush-vpc" {
  # (resource arguments)
  cidr_block = "192.168.0.0/16"
}

resource "aws_subnet" "ayush-subnet" {
  map_public_ip_on_launch = true
  vpc_id     = aws_vpc.ayush-vpc.id
  cidr_block = "192.168.128.0/18"
  tags = var.default_tags
}

resource "aws_subnet" "ayush-subnet-1" {
  vpc_id     = aws_vpc.ayush-vpc.id
  cidr_block = "192.168.192.0/18"
  tags = var.default_tags
}

resource "aws_security_group" "ayush-sg" {
  name        = "ayush-sg-terraform"
  description = "for terraform"
  vpc_id      = aws_vpc.ayush-vpc.id

  ingress {
    description = "TLS from VPC"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["103.87.56.94/32"]
  }
```

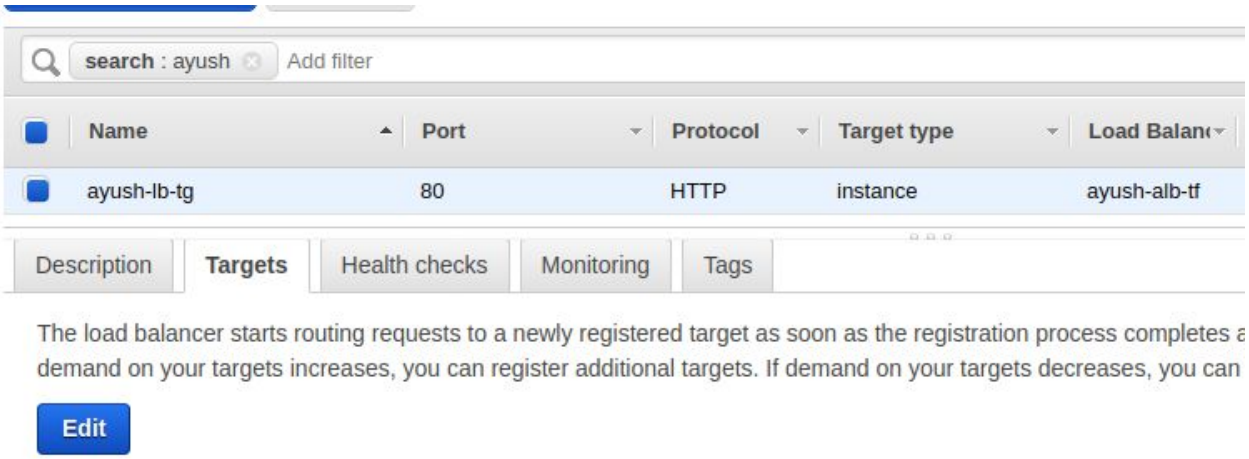
```
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    owner = "ayush"
    purpose = "ayush-sg-terraform"
    Name = "ayush-sg-terraform"
  }
}

resource "aws_security_group_rule" "example" {
  type              = "ingress"
  from_port         = 80
  to_port           = 80
  protocol          = "tcp"
  cidr_blocks       = ["0.0.0.0/0"]
  security_group_id = aws_security_group.ayush-sg.id
}

resource "aws_instance" "ayush" {
  ami           = "ami-05055726077359305"
  instance_type = "t2.micro"
  key_name = "ayush-pem"
  subnet_id = aws_subnet.ayush-subnet.id
  security_groups = [aws_security_group.ayush-sg.id]
  tags = var.default_tags
}
```

```
resource "aws_lb_target_group" "ayush-tg" {
  name     = "ayush-lb-tg"
  port     = 80
  protocol = "HTTP"
  vpc_id   = aws_vpc.ayush-vpc.id
  tags = var.default_tags
}

resource "aws_lb_target_group_attachment" "ayush-tg-attach" {
  target_group_arn = aws_lb_target_group.ayush-tg.arn
  target_id        = aws_instance.ayush.id
  port             = 80
}


resource "aws_lb" "ayush-alb" {
  name               = "ayush-alb-tf"
  internal           = false
  load_balancer_type = "application"
  security_groups    = [aws_security_group.ayush-sg.id]
  subnets            = [aws_subnet.ayush-subnet.id, aws_subnet.ayush-subnet-1.id]
  tags = var.default_tags
}

resource "aws_lb_listener" "ayush-lb-listener" {
  load_balancer_arn = aws_lb.ayush-alb.arn
  port              = "80"
  protocol          = "HTTP"

  default_action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.ayush-tg.arn
  }
}
```

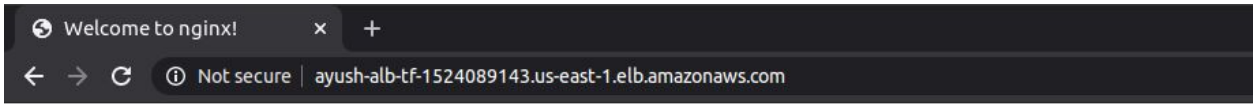
```
resource "aws_lb_listener" "ayush-lb-listener" {
  load_balancer_arn = aws_lb.ayush-alb.arn
  port              = "80"
  protocol          = "HTTP"

  default_action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.ayush-tg.arn
  }
}

terraform {
  backend "s3" {
    bucket = "ec2-ttn"
    key    = "ayush/terraform.tfstate"
    region = "us-east-1"
  }
}
```


Launch Instance
Connect
Actions
search : ayush
Add filter
Name
Instance ID
Instance Type
Availability Zone
Instance State
Status Checks
ayush-tf
i-0fb0811ed3bbb85c2
t2.micro
us-east-1b
running
Initializing
i-0027000fbd876e4e6
t2.micro
us-east-1b
terminated
i-042b93e269c586d40
t2.micro
us-east-1b
terminated


Create Load Balancer
Actions
search : ayush
Add filter
Name
DNS name
State
ayush-alb-tf
ayush-alb-tf-1524089143.us-...
active




```
ayush@ayush:~/terraform$ cat variable.tf
variable "default_tags" {
    type = map(string)
    default = {
        Name: "ayush-tf",
        owner: "ayush",
        purpose: "ayush-tf",
  }
}

ayush@ayush:~/terraform$
```



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

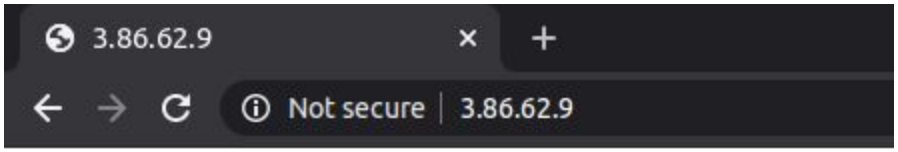
For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

4. Create a terraform ec2 module to explicitly pass IAM role policy and userdata as a file.

```
ayush@ayush:~/terraform/ques4$ cat main.tf
provider "aws" {
  region = "us-east-1"
}

module "ec2_instances" {
  source  = "terraform-aws-modules/ec2-instance/aws"
  version = "2.12.0"

  name           = "ayush-ec2"
  instance_count = 1

  associate_public_ip_address = true
  key_name                    = "ayush-pem"
  ami                         = "ami-07ebfd5b3428b6f4d"
  instance_type               = "t2.micro"
  vpc_security_group_ids      = ["sg-03105b60052b9d72c"]
  subnet_id                   = "subnet-0a5a6b106347d1b70"
  iam_instance_profile        = "ec2-s3-ayush"
  user_data = "${file("install_apache2.sh")}"
  tags = {
    owner = "ayush"
    purpose = "ayush-terraform"
  }
}
```

```
ayush@ayush:~/terraform/ques4$ cat install_apache2.sh
#! /bin/bash
sudo apt-get update
sudo apt-get install -y apache2
sudo systemctl start apache2
sudo systemctl enable apache2
echo "<h1>Deployed via Terraform</h1>" | sudo tee /var/www/html/index.html
ayush@ayush:~/terraform/ques4$
```



Deployed via Terraform

```
ubuntu@ip-192-168-112-219:~$ aws s3 ls s3://ec2-ttn
                           PRE ayush/
                           PRE trail/
2019-03-28 05:24:59        157 out.txt
ubuntu@ip-192-168-112-219:~$
```