Recap

In this section, we would be recapping some of the concepts that would be used by you during today's activity:

Backend

The purpose of using terraform backend is just so that you can store your terraform state files securely somewhere (e.g. s3 bucket), which also allows you to work collaboratively with your team.

You can use the following backend configuration for the activity today(Just remember to change the key in the backend config block):

```
terraform {
  backend "s3" {
    bucket = "sctp-ce4-tfstate-bucket"
    key    = "" #Remember to change this
    region = "ap-southeast-1"  #The region of your backend bucket
  }
}
```

For more on backend:
https://developer.hashicorp.com/terraform/language/settings/backends/configuration

Output

Output values are typically used to expose some of the values that get generated as part of your terraform stack. In many cases, these values can also be passed from 1 stack to another.

For example, you can have 1 stack called a Network Stack. In this stack, lets say your VPC, subnets etc. gets created. You can then output the value of the VPC and subnets from this stack, which can then be used by another stack that is used to create your infrastructure (e.g. EC2, RDS etc.) This can be seamlessly done using something like Terraform Cloud/Enterprise, or can also be performed using CI/CD (but you would have to write a script for it).

Note: when using modules, you have to expose these values as outputs, before you can output them from the root module that is calling it.

For more on outputs: Output Values - Configuration Language | Terraform | HashiCorp Developer

Conditions

Conditions are typically used for conditionally creating resources in Terraform, which would be covered later in Activity 2.

Conditional Expressions - Configuration Language | Terraform | HashiCorp Developer

Pre-Requisites

For today's activity, Create a new github repository, with a .gitignore template for Terraform. And then clone it to your local working machine and open it in your preferred IDE (E.g. VS Code)

Activity 1

In this activity, we would be creating a EC2 in the default VPC in public subnets, based on the following conditions:

1) Create a “t2.micro” EC2 with the Amazon Linux 2023 AMI
2) Name the EC2 after your name, something like “jazeel-terraform-instance” (**Hint: Use the tags block**)
3) Associate a public ip address to your EC2 instance (**Hint: It is a boolean field. Look for it in the terraform registry under the EC2 Instance resource**)
4) Create it in the default VPC and in 1 of the public subnets
5) Do not hard code the VPC ID or Subnet IDs anywhere in the code (**Hint: Use data sources to get the default VPC ID, then try to get the public subnets within that VPC**)
6) Create an IAM Role with the “AmazonSSMManagedInstanceCore” Policy attached to it.
    a) In Terraform when you create an IAM Role, you would have to explicitly specify the trust policy on which service needs to assume the role. (In this case, being the EC2). You can refer to the example here: aws_iam_role | Resources | hashicorp/aws | Terraform | Terraform Registry
    b) Remember that in order to attach a IAM role to an Ec2 instance, you would need to create an instance profile as well. (aws_iam_instance_profile | Resources | hashicorp/aws | Terraform | Terraform Registry)
    c) Once the instance profile is created, associate it with the EC2 instance.
7) Also add the following block to your terraform code, to install httpd during the instance startup:

```
  user_data                   = <<EOF
#!/bin/bash
yum install -y httpd
systemctl enable httpd
systemctl start httpd
EOF
  user_data_replace_on_change = true # this forces instance to be
recreated upon update of user data contents
```

It is optional to create a security group, as it will automatically attach the VPC’s default security group to your instance.

Remember to run terraform destroy before proceeding to the next Activity

Activity 2

Imagine you're working in a company / project that has more than 1 environment. (non-prod and prod environment for example). Modify the code from Activity 1 to be used to fulfill the following requirements in this activity:

Requirements:

1) Both non-production and production EC2s should be accessible through Session Manager
2) Create **1 EC2** for non-production and **2 EC2** for production environment
    a) Use "ap-southeast-1" for production
    b) Use "ap-northeast-1" for non-production
3) Write as less code as possible (Look at utilizing modules)
4) Think of how you will manage the states for production and non-production environments (Single state file or separate state files?)
5) Both the non-production EC2s and production EC2s should have different names, ideally prefixed by your name + environment type
6) Output the **ARN** of your EC2 instances
7) (**Optional Challenge**): Create a SSH Key pair via Terraform. The SSH key should also be attached to the production EC2 instances. The non-production instances should not use any key pair and should be left to null. The code below is the code used to create your SSH Key pair(However, you'll need to make some very minor tweaks due to the condition specified above) (Hint: Conditional Expressions - Configuration Language | Terraform | HashiCorp Developer)

```
# Generates a secure private key and encodes it as PEM
resource "tls_private_key" "key_pair" {
  algorithm = "RSA"
  rsa_bits  = 4096
}
# Creating Key Pair
resource "aws_key_pair" "key_pair" {
  key_name   = var.key_pair_name
  public_key = tls_private_key.key_pair.public_key_openssh
}
# if running terraform locally, this would save the pemfile locally in the
directory you run terraform
resource "local_file" "ssh_key" {
  filename = "${aws_key_pair.key_pair.key_name}.pem"
  content  = tls_private_key.key_pair.private_key_pem
```

```
}
```

Important note: If you are doing the optional challenge above, remember to add your pem key name to your .gitignore file, just so that you don't end up committing your PEM key to github

So you can add the following entry to your .gitignore file

```
#Ignore all pem files from being committed to git
*.pem
```

Remember to run terraform destroy at the end of Activity 2 as well 🙂