Deploying Amazon Elastic File System (EFS) with EC2 for Scalable and Shared Storage Architecture

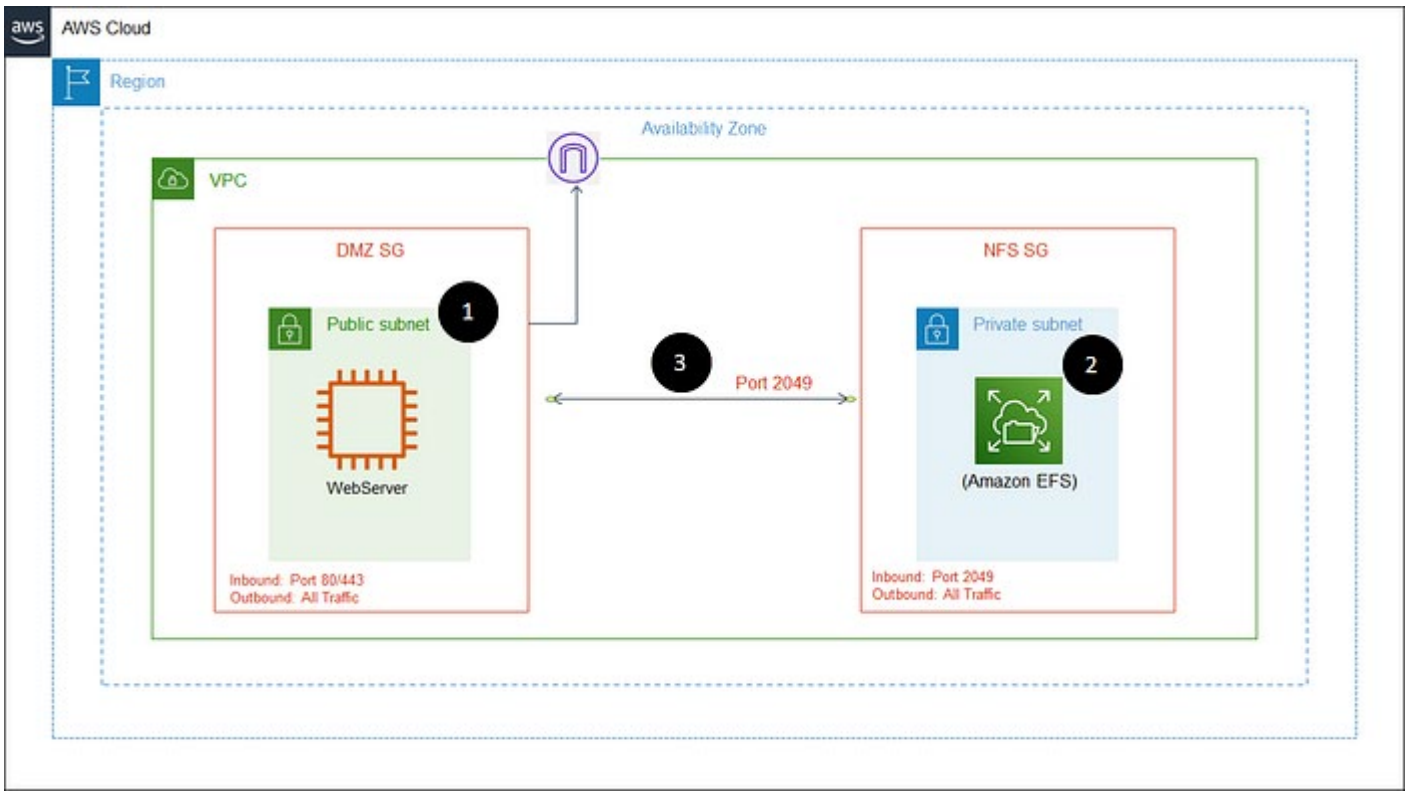
Amazon Elastic File System (Amazon EFS) is a fully managed cloud-based file storage service offered by Amazon Web Services (AWS). It is designed to provide scalable, elastic, and highly available file storage for AWS cloud-based applications and services. Amazon EFS is built on a distributed architecture that allows it to grow and shrink automatically as data storage needs change, making it an excellent choice for applications with dynamic storage requirements.

Key features of Amazon EFS include:

1. **Scalability**: Amazon EFS can automatically scale up or down based on demand, allowing you to accommodate changes in your storage requirements without having to provision or manage additional storage capacity manually.
2. **Fully Managed**: AWS takes care of all the underlying infrastructure, maintenance, and patching of Amazon EFS, allowing you to focus on using the service without worrying about server management tasks.
3. **Multi-AZ Availability**: Amazon EFS provides high availability by storing data redundantly across multiple Availability Zones (AZs) within a region. This ensures that your data remains accessible even if an entire AZ becomes unavailable.
4. **File System Sharing**: Amazon EFS supports multiple instances running simultaneously that can share access to the same file system, making it suitable for distributed applications and workflows.
5. **Security**: EFS supports AWS Identity and Access Management (IAM) for access control, allowing you to manage user permissions and secure access to your file systems.

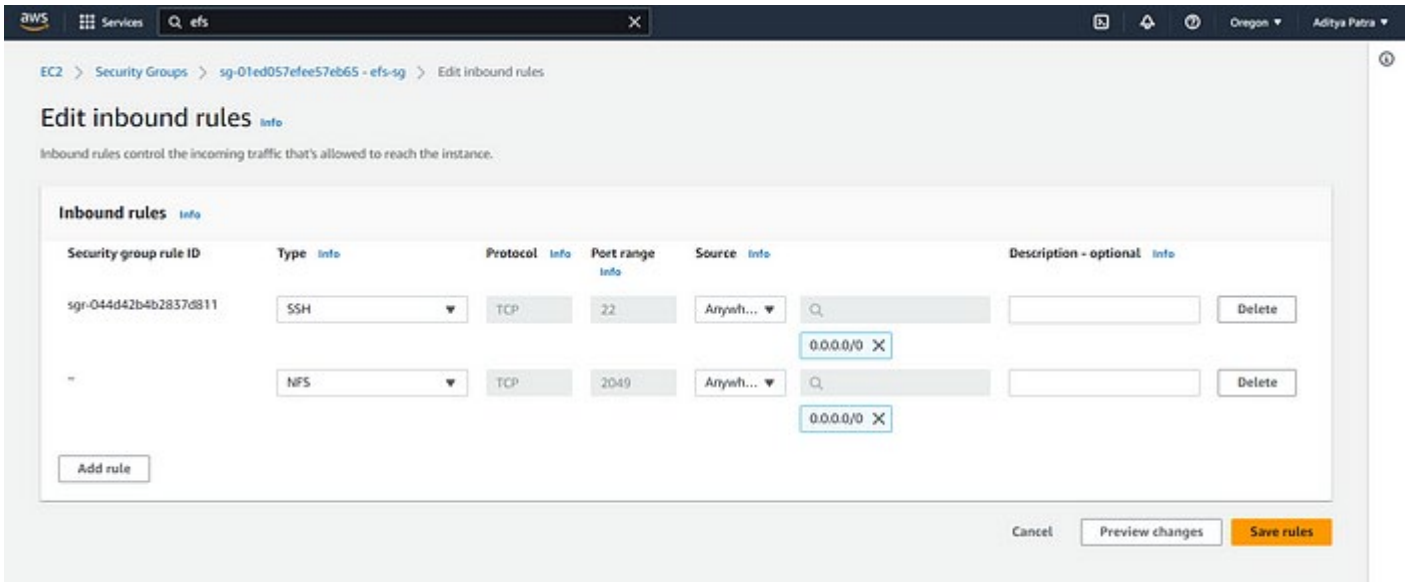
Amazon EFS is particularly well-suited for applications that require shared access to a common data source, such as content management systems, web servers, development environments, and big data processing workflows.

To enable EFS we have to add NFS which has port 2049 in the security group. NFS converts the storage into logical volume.

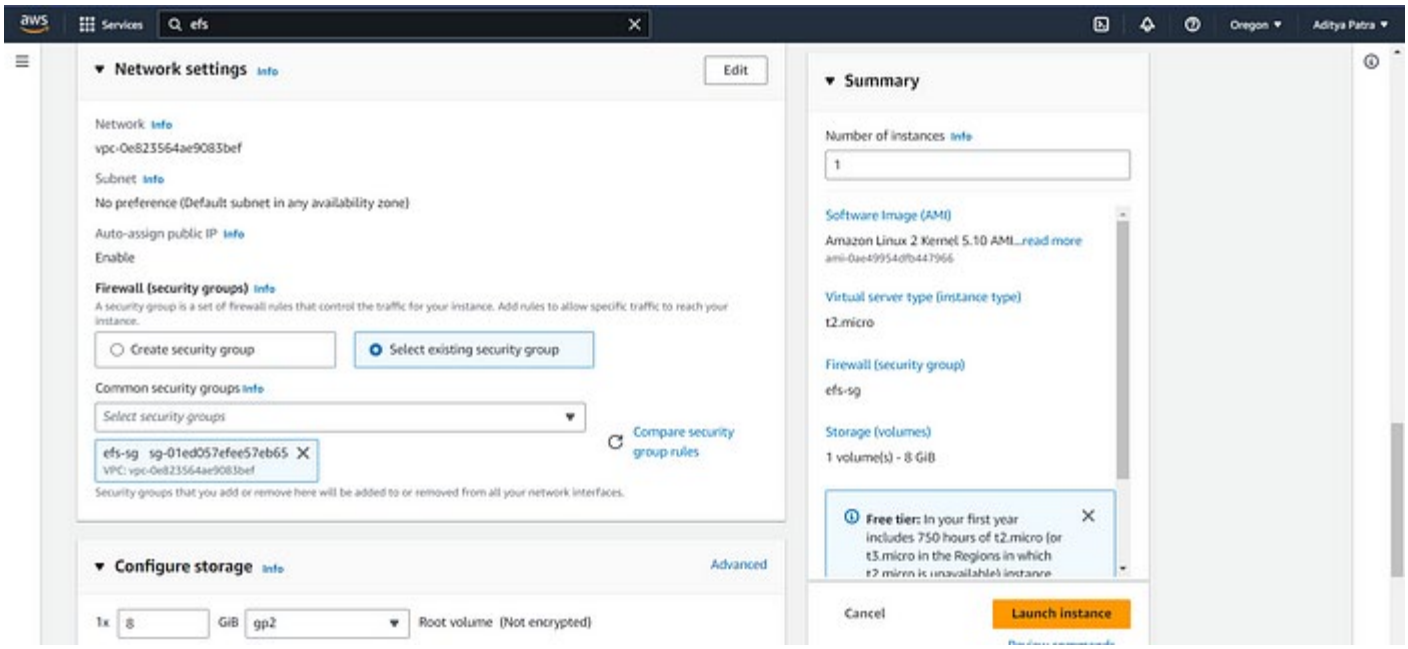


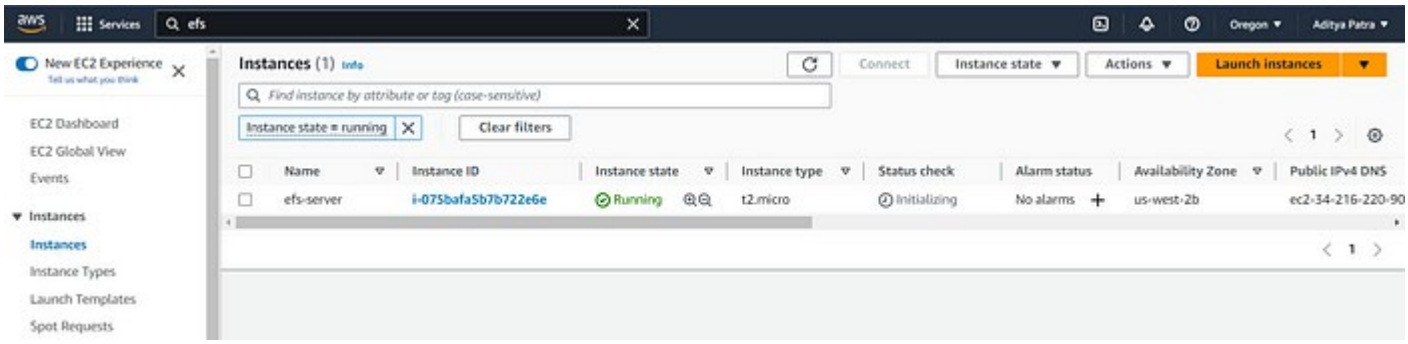
How to add EFS to your Ec2 instance?

i) Create an Security group in which we have to add **NFS(Port: 2049)** and **SSH(Port: 22)** in the inbound rule . So that we can access EFS.

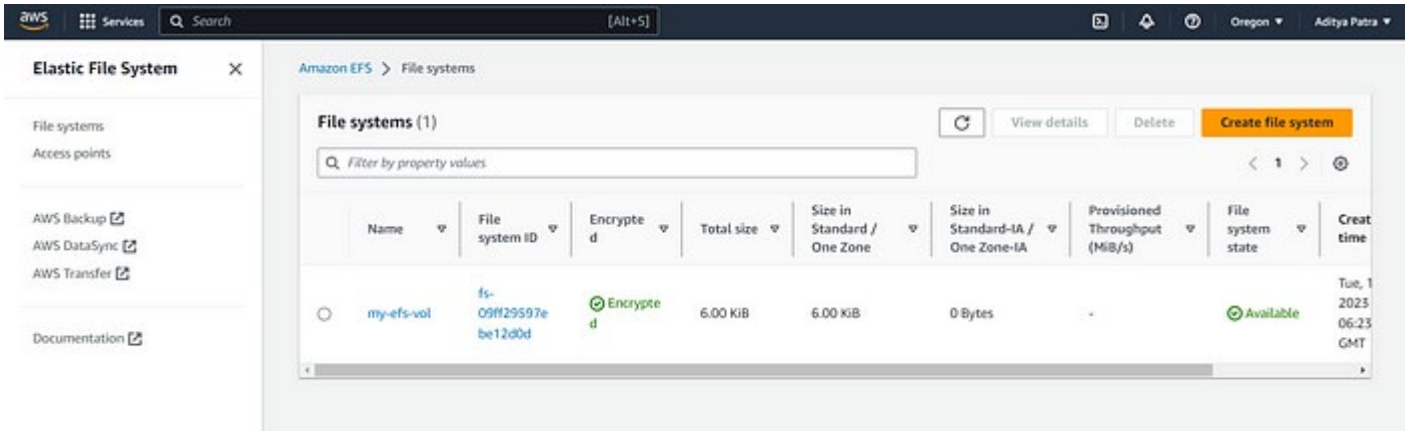


ii) Launch an Ec2 instance using the above created security group.





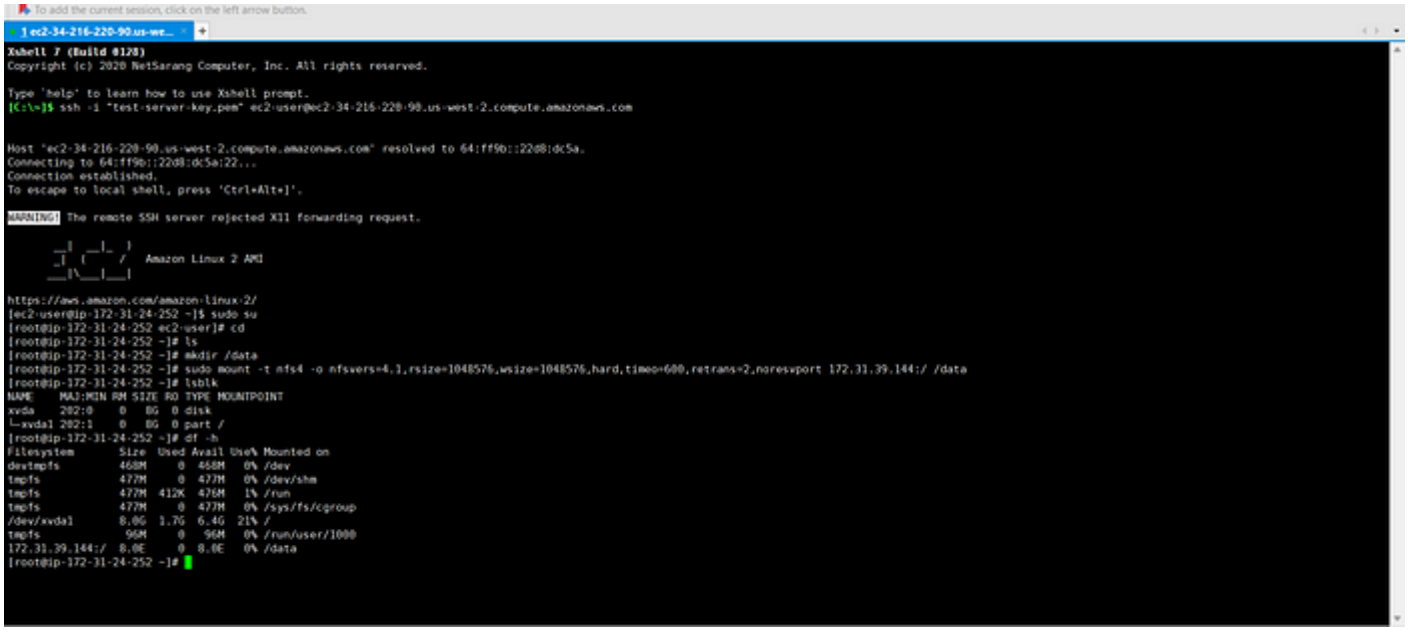
iii) Create EFS



iv) Attach this EFS to Ec2 instance through **Mount via IP**

*here we are using one-zone deployment, you can also use multi-zone deployment.

v) Copy the **NFS client** and paste it in Xshell



*here we **mount** EFS inside a directory called **data** which is created inside root

vi) Use **df -h** command to check that EFS is successfully mounted or not

Infrastructure as Code (IaC) Overview

In cloud computing, **Infrastructure as Code (IaC)** allows developers and system administrators to automatically manage, provision, and deploy computing infrastructure using code and automation rather than manual processes. AWS supports two main IaC tools:

- **AWS CloudFormation** – An AWS-native service that allows you to write YAML or JSON templates to define infrastructure.
- **Terraform** – An open-source, cloud-agnostic IaC tool developed by HashiCorp that uses its own declarative language (HCL – HashiCorp Configuration Language).

In this project, we use **both** CloudFormation and Terraform to deploy **Amazon EFS** integrated with an **EC2 instance**, demonstrating how IaC can streamline infrastructure creation, ensure repeatability, and eliminate manual errors.

Deploying EFS using CloudFormation

CloudFormation Template (YAML):

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Deploy EFS with EC2

Resources:
  MyVPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: 10.0.0.0/16

  MySubnet:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref MyVPC
      CidrBlock: 10.0.1.0/24
      AvailabilityZone: !Select [0, !GetAZs ""]

  MySecurityGroup:
```

```
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Enable SSH and NFS
      VpcId: !Ref MyVPC
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: 0.0.0.0/0
        - IpProtocol: tcp
          FromPort: 2049
          ToPort: 2049
          CidrIp: 0.0.0.0/0

MyInstance:
    Type: AWS::EC2::Instance
    Properties:
      InstanceType: t2.micro
      SubnetId: !Ref MySubnet
      ImageId: ami-0c02fb55956c7d316 # Amazon Linux 2
      SecurityGroupIds:
        - !Ref MySecurityGroup
      KeyName: your-key-pair-name
      UserData:
        Fn::Base64: !Sub |
          #!/bin/bash
```

```
            yum update -y
            yum install -y amazon-efs-utils
            mkdir /data
            mount -t efs ${MyEFS.FileSystemId}:/ /data

  MyEFS:
    Type: AWS::EFS::FileSystem
    Properties:
      PerformanceMode: generalPurpose
      Encrypted: true

  MountTarget:
    Type: AWS::EFS::MountTarget
    Properties:
      FileSystemId: !Ref MyEFS
      SubnetId: !Ref MySubnet
      SecurityGroups:
        - !Ref MySecurityGroup
```

Deploying EFS using Terraform

Terraform Code:

```
provider "aws" {
  region = "ap-south-1"
}

resource "aws_vpc" "main" {
```

```
    cidr_block = "10.0.0.0/16"
}

resource "aws_subnet" "main" {
    vpc_id          = aws_vpc.main.id
    cidr_block = "10.0.1.0/24"
}

resource "aws_security_group" "efs_sg" {
    name                = "efs_sg"
    description = "Allow SSH and NFS"
    vpc_id            = aws_vpc.main.id

    ingress {
        from_port      = 22
        to_port          = 22
        protocol        = "tcp"
        cidr_blocks = ["0.0.0.0/0"]
    }

    ingress {
        from_port      = 2049
        to_port          = 2049
        protocol        = "tcp"
        cidr_blocks = ["0.0.0.0/0"]
    }
```

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

resource "aws_instance" "ec2" {
    ami                                 = "ami-0c02fb55956c7d316"
    instance_type                   = "t2.micro"
    subnet_id                          = aws_subnet.main.id
    vpc_security_group_ids = [aws_security_group.efs_sg.id]
    key_name                           = "your-key-pair-name"

    user_data = <<-EOF
                        #!/bin/bash
                        yum update -y
                        yum install -y amazon-efs-utils
                        mkdir /data
                        mount -t efs ${aws_efs_file_system.efs.id}:/ /data
                    EOF
}

resource "aws_efs_file_system" "efs" {
```

```
  performance_mode = "generalPurpose"
  encrypted        = true
}

resource "aws_efs_mount_target" "efs_mount" {
  file_system_id  = aws_efs_file_system.efs.id
  subnet_id       = aws_subnet.main.id
  security_groups = [aws_security_group.efs_sg.id]
}
```

Commands for AWS CloudShell / EC2 :

Mount via IP:

```
sudo yum install -y amazon-efs-utils
sudo mkdir /data
sudo mount -t efs -o tls <EFS-ID>:/ /data
```

Verify:

```
df -h
```

Summary and Key Notes

- **EFS is shared, scalable, and elastic** storage.
- Always **install amazon-efs-utils** before mounting.
- For production, consider **multi-zone deployment** for high availability.
- Ensure **EFS and EC2** are in the **same VPC & AZ** for mount target compatibility.
- Automate deployment using **CloudFormation or Terraform** to ensure reproducibility and disaster recovery support.