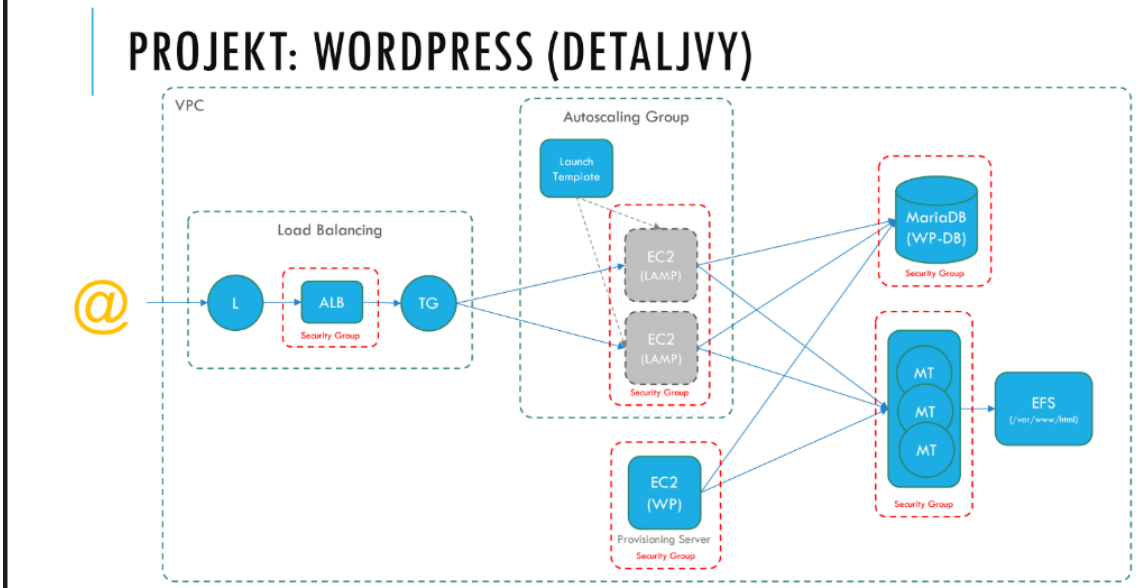
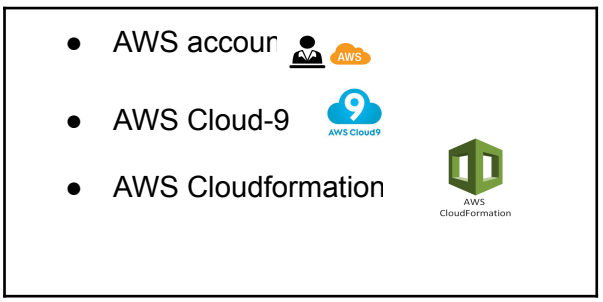
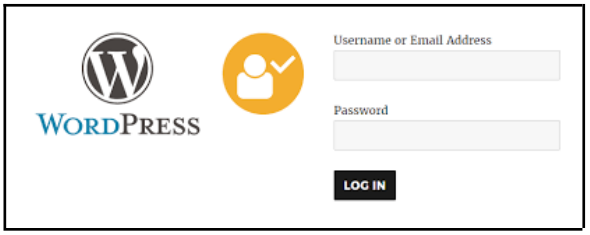
Create a robust, secure, and possibly scalable WordPress site that leverages EFS (Elastic File System) and RDS.



Things we will work with to get this:





Now prepare the things:

1. Security groups x5
2. Lamp launch template
3. Wordpress and Apache instance
4. Remote desktop (mariadb)
5. EFS
6. Load Balance
7. Auto Scaling

Step1# Login and Security groups

After login to your AWS account console , you need to create an environment by going to Cloud 9 then create a file with the name of “wordpress.yaml” or any name you want!

Now we need to create 5 Security group:

1. Load Balance
2. LAMPinstance
3. Wordpress Instance
4. RDS
5. EFS

Security Groups:

ALBSecuritySG1: Defines a security group for the Application Load Balancer allowing incoming traffic on ports 80 and 443.

MyEC2LAMPSecGroup: Security group for EC2 instances, allowing SSH from any IP, and HTTP/HTTPS from the ALBSecuritySG1 group.

MyWPSecGroup: Security group for WordPress instances, allowing SSH, HTTP, and HTTPS traffic.

MyEfsSecGroup1 and **MyEfsSecGroup:** Security groups for EFS, allowing SSH and NFS traffic.

MyRDSSecGroup: Security group for RDS, allowing MySQL traffic on port 3306.

```
---
AWSTemplateFormatVersion: "2010-09-09"

Description: >
  Here are some
  details about
  the template.

Resources:
  ALBSecuritySG1:
    Type: AWS::EC2::SecurityGroup
    Properties:
      VpcId: "vpc-0e0feebb6af022e5b"
      GroupDescription: Allow http for loadbalance
      GroupName: "ALBSecuritySG1"
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
        - IpProtocol: tcp
          FromPort: 443
          ToPort: 443
          CidrIp: 0.0.0.0/0

  MyEC2LAMPSecGroup:
```

```
  Type: AWS::EC2::SecurityGroup
  Properties:
    VpcId: "vpc-0e0feebb6af022e5b"
    GroupDescription: "Allow ssh for all and http for loadbalance"
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 22
        ToPort: 22
        CidrIp: 0.0.0.0/0
      - IpProtocol: tcp
        FromPort: 80
        ToPort: 80
        SourceSecurityGroupId: !GetAtt ALBSecuritySG1.GroupId
      - IpProtocol: tcp
        FromPort: 443
        ToPort: 443
        SourceSecurityGroupId: !GetAtt ALBSecuritySG1.GroupId

MyWPSecGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    VpcId: "vpc-0e0feebb6af022e5b"
    GroupDescription: "Allow ssh for all and http for loadbalance"
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 22
        ToPort: 22
        CidrIp: 0.0.0.0/0
      - IpProtocol: tcp
        FromPort: 80
        ToPort: 80
        CidrIp: 0.0.0.0/0
      - IpProtocol: tcp
        FromPort: 443
        ToPort: 443
        CidrIp: 0.0.0.0/0

MyEfsSecGroup1:
  Type: AWS::EC2::SecurityGroup
  Properties:
    VpcId: "vpc-0e0feebb6af022e5b"
    GroupDescription: "Allow ssh for all"
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 22
        ToPort: 22
        CidrIp: 0.0.0.0/0

MyEfsSecGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    VpcId: "vpc-0e0feebb6af022e5b"
    GroupDescription: "Allow ssh for all"
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 2049
        ToPort: 2049
        CidrIp: 0.0.0.0/0
      - IpProtocol: tcp
```

```
      FromPort: 22
      ToPort: 22
      SourceSecurityGroupId: !GetAtt MyEfsSecGroup1.GroupId

MyRDSSecGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    VpcId: "vpc-0e0feebb6af022e5b"
    GroupDescription: "Allow Rds"
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 3306
        ToPort: 3306
        CidrIp: 0.0.0.0/0
```

Step2# Instances

Instance 1# *Launch Template (MyEC2launchInstance):*

Defines a launch template for EC2 instances with user data to install necessary packages, start Apache, and mount EFS.

```
MyEC2launchInstance:
  Type: AWS::EC2::LaunchTemplate
  Properties:
    LaunchTemplateName: "MyLampLaunchTemplate"
    LaunchTemplateData:
      ImageId: "ami-06ed60ed1369448bd"
      KeyName: "amna"
      InstanceType: "t2.micro"
      SecurityGroupIds:
        - !GetAtt MyEC2LAMPSecGroup.GroupId
      UserData:
        Fn::Base64: !Sub |
          #!/bin/bash
          dnf update -y
          dnf install -y httpd wget php-fpm php-mysqli php-json php php-devel
          systemctl start httpd
          systemctl enable httpd
          dnf install -y amazon-efs-utils
          sudo mount -t efs -o tls ${MyEFS}:/ /var/www/html
          sudo mount -t efs -o tls ${MyEFS}:/ /var/www/html
  DependsOn:
    - EFSMountTarget1
    - EFSMountTarget2
    - EFSMountTarget3
```

Instance2# *EC2 Instance (MyWPProv)*:

Deploys an EC2 instance for WordPress, using the launch template and mounting EFS. Install's and configures WordPress.

```
MyWPProv:
   Type: AWS::EC2::Instance
   Properties:
      ImageId: "ami-06ed60ed1369448bd"
      KeyName: "amna"
      InstanceType: "t2.micro"
      SecurityGroupIds:
        - !GetAtt MyWPSecGroup.GroupId
      UserData:
        Fn::Base64: !Sub |
          #!/bin/bash
          dnf install -y amazon-efs-utils
          sudo mount -t efs -o tls ${MyEFS}:/ /var/www/html
          dnf install -y httpd wget php-fpm php-mysqli php-json php php-devel
          systemctl start httpd
          systemctl enable httpd
          usermod -a -G apache ec2-user
          chown -R ec2-user:apache /var/www
          chmod 2775 /var/www && find /var/www -type d -exec sudo chmod 2775 {} \;
          find /var/www -type f -exec sudo chmod 0664 {} \;
          wget https://wordpress.org/latest.tar.gz
          tar -xzvf latest.tar.gz
          cp wordpress/wp-config-sample.php wordpress/wp-config.php
          cp -r wordpress/* /var/www/html
          sed -i -e 's/database_name_here/wordpressdb/g' /var/www/html/wp-config.php
          sed -i -e 's/username_here/Admin/g' /var/www/html/wp-config.php
          sed -i -e 's/password_here/Amna1234/g' /var/www/html/wp-config.php
          sed -i -e 's/localhost/${MyDB.Endpoint.Address}/g' /var/www/html/wp-config.php
   DependsOn:
     - EFSMountTarget1
     - EFSMountTarget2
     - EFSMountTarget3
     - MyDB
```

Step3# *RDS Database (MyDB):*

Defines an RDS MySQL database instance with specified settings.

```
MyDB:
   Type: AWS::RDS::DBInstance
   Properties:
     AllocatedStorage: 20
     DBInstanceClass: db.t3.micro
     Engine: MySQL
     MasterUsername: Admin
     MasterUserPassword: Amna1234
```

```
    DBName: wordpressdb
    VPCSecurityGroups:
      - !GetAtt MyRDSSecGroup.GroupId
    MultiAZ: false
    BackupRetentionPeriod: 0
```

Step4# Elastic File System (MyEFS):

Creates an EFS file system for shared storage.

```
MyEFS:
    Type: AWS::EFS::FileSystem
    Properties:
      PerformanceMode: generalPurpose
      ThroughputMode: bursting
      Encrypted: true
      LifecyclePolicies:
        - TransitionToIA: AFTER_30_DAYS
```

Step5# EFS Mount Targets (EFSMountTarget1, EFSMountTarget2, EFSMountTarget3):

Creates mount targets for EFS in different subnets.

```
  EFSMountTarget1:
    Type: AWS::EFS::MountTarget
    Properties:
      FileSystemId: !Ref MyEFS
      SubnetId: subnet-0dba558a60b154c88
      SecurityGroups:
        - !GetAtt MyEfsSecGroup.GroupId

  EFSMountTarget2:
    Type: AWS::EFS::MountTarget
    Properties:
      FileSystemId: !Ref MyEFS
      SubnetId: subnet-00bf6b2d4eed1742a
      SecurityGroups:
        - !GetAtt MyEfsSecGroup.GroupId

  EFSMountTarget3:
    Type: AWS::EFS::MountTarget
    Properties:
      FileSystemId: !Ref MyEFS
```

```
    SubnetId: subnet-0a1456112e3ea7b35
    SecurityGroups:
      - !GetAtt MyEfsSecGroup.GroupId
```

Step6# Target Group (MyTargetGroup):

Defines an Elastic Load Balancer target group for routing traffic to EC2 instances.

```
MyTargetGroup:
   Type: AWS::ElasticLoadBalancingV2::TargetGroup
   Properties:
     Name: "MyTargetGroup"
     Port: 80
     Protocol: "HTTP"
     VpcId: "vpc-0e0feebb6af022e5b"
```

Step7# Application Load Balancer (ALB):

Creates an Application Load Balancer with specified security groups and subnets.Use the subnet while searching on aws console type”subnets” and their you can get subnets.

```
ALB:
   Type: AWS::ElasticLoadBalancingV2::LoadBalancer
   Properties:
     Name:  "ALB"
     Type: "application"
     SecurityGroups:
       - !Ref ALBSecuritySG1
     Subnets:
       - subnet-0dba558a60b154c88
       - subnet-00bf6b2d4eed1742a
       - subnet-0a1456112e3ea7b35
```

Step8# Listener (Listenerupp1):

Defines an ALB listener to forward traffic to the target group on port 80.

```
Listenerupp1:
   Type: AWS::ElasticLoadBalancingV2::Listener
   DependsOn:
    - MyTargetGroup
    - ALB
```

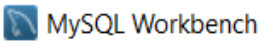
```
Properties:
  DefaultActions:
    - Type: forward
      TargetGroupArn: !Ref MyTargetGroup
  LoadBalancerArn: !Ref ALB
  Port: 80
  Protocol: HTTP
```

Step9# Auto Scaling Group (MyAutoScalingGroup):

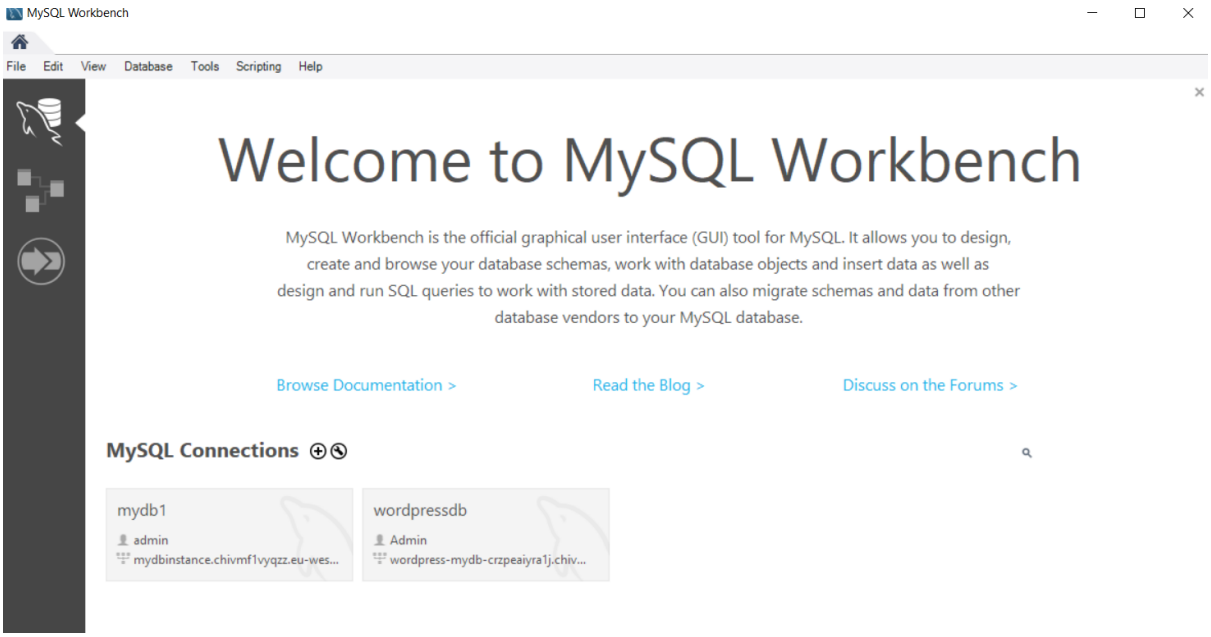
Sets up an Auto Scaling Group using the launch template to manage EC2 instances, specifying minimum, maximum, and desired capacities.

```
MyAutoScalingGroup:
  Type: "AWS::AutoScaling::AutoScalingGroup"
  Properties:
    AutoScalingGroupName: "MyAutoScalingGroup"
    LaunchTemplate:
      LaunchTemplateId: !Ref MyEC2launchInstance
      Version: !GetAtt MyEC2launchInstance.LatestVersionNumber
    MinSize: "2"
    MaxSize: "4"
    DesiredCapacity: "2"
    AvailabilityZones:
      - eu-west-1a
      - eu-west-1b
    TargetGroupARNs:
      - !Ref MyTargetGroup
```

Step10# MYSQL

Download the application some name as MYSQL workbench 

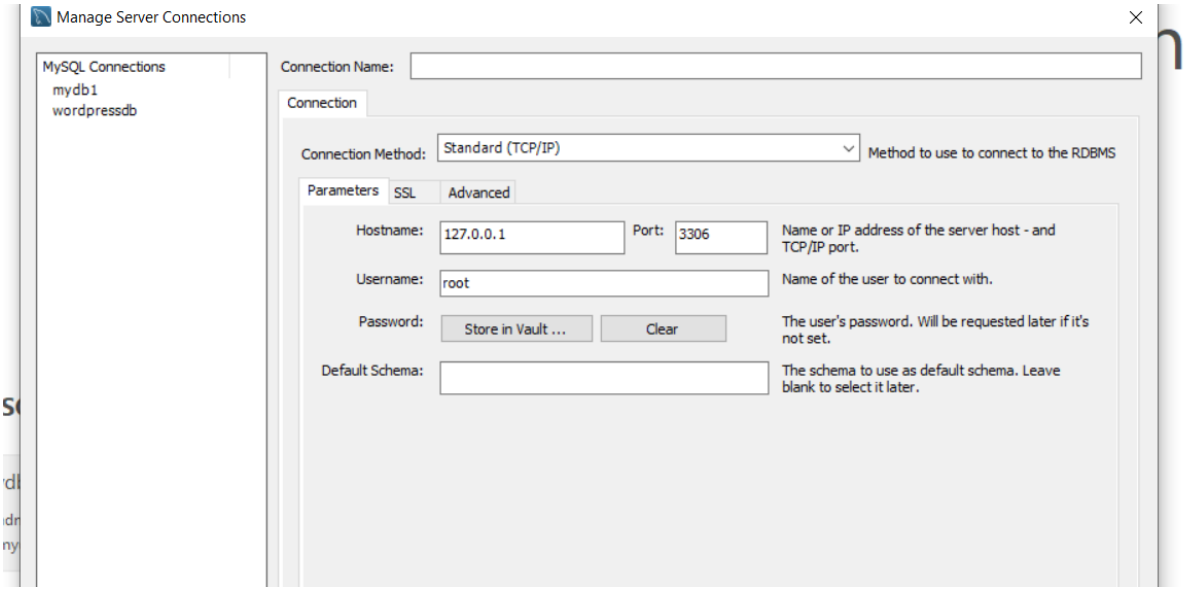
.



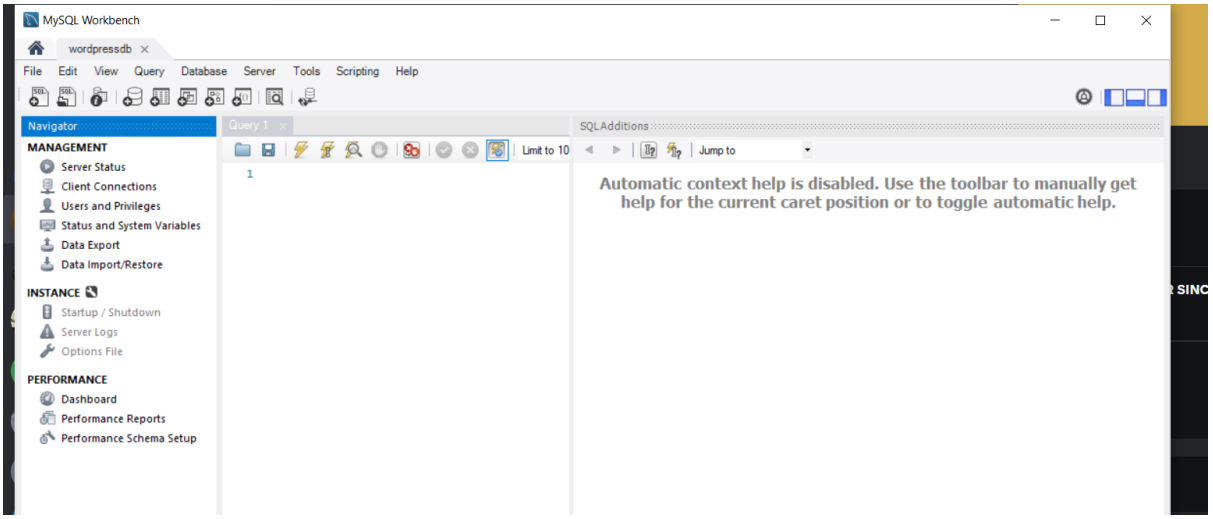
Now go to create a connection using the data you have entered in your script such as Master name and password!

Remember that Connection name,User name and Password must be the same while creating wordpress and MYsql data in script.

Use the Endpoint of RDS and paste it in Host name section to get connected.



After to connected you can see the console as :



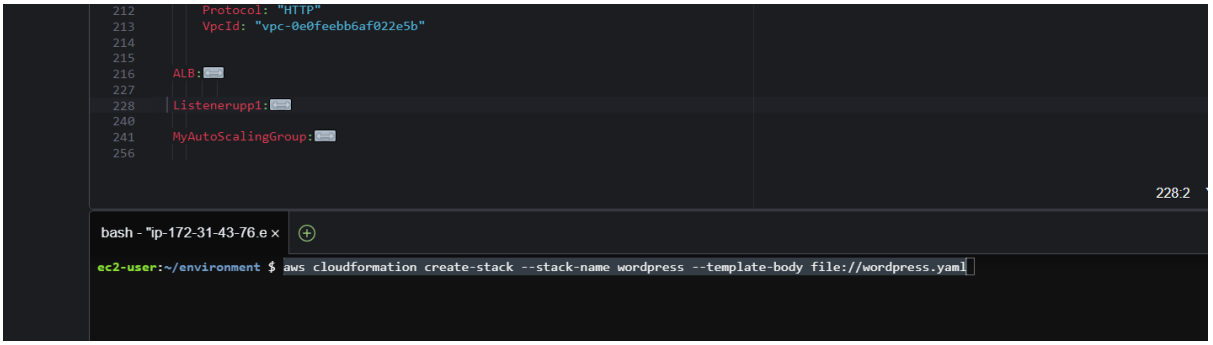
Dependencies:

MyWPProv(word press instance**)** and **MyEC2launchInstance (Lamp instance)**have dependencies on **EFSMountTarget1, EFSMountTarget2, and EFSMountTarget3** to ensure that the EFS is available before launching instances.

MyWPProv has a dependency on **MyDB(Mariadb/rds)** to ensure the database is created before launching WordPress instances.

After creating this all script in ".yaml" you can run the script in the environment by typing this.

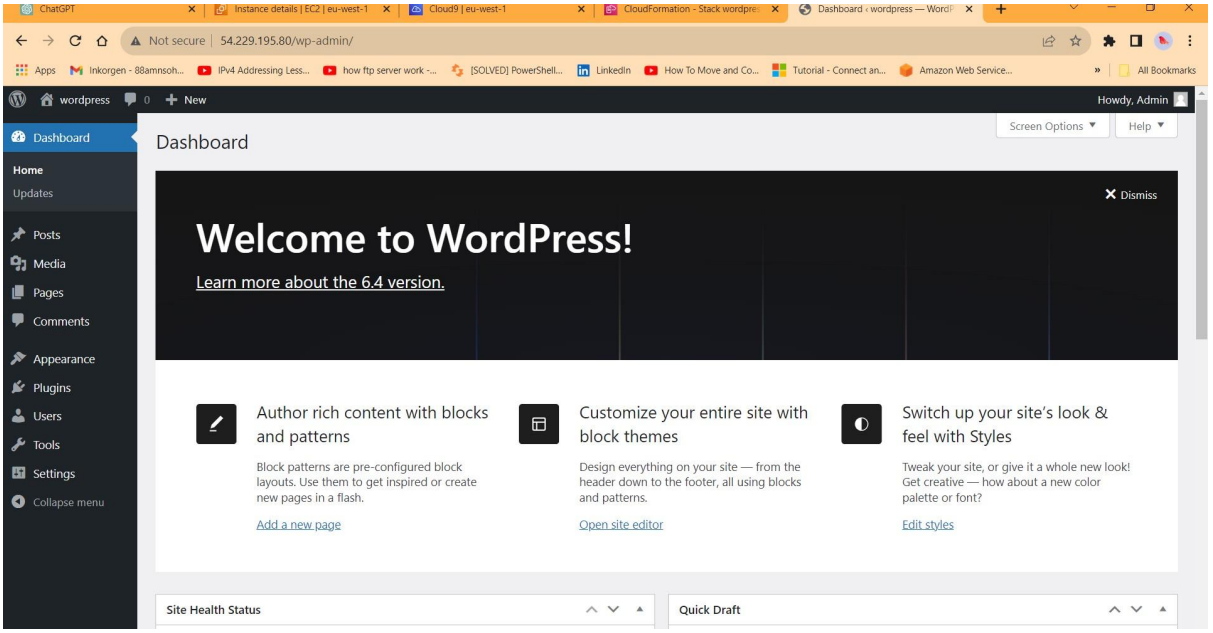
```
aws cloudformation create-stack --stack-name wordpress --template-body file://wordpress.yaml
```



Now go to Cloudformation and check the stack if it is created.

In case it is created : Then check all the instance is they are working correctly

You can go to ur instance and connect them and type “cd/var/www/html” to check efs is shared and mounted correctly ,
For Wordpress use public IP in URL if you can see the wordpress site:



In case it is not created: Then go to the Events and see where are the errors you need to fix.
Then go back to environment and delete the previous stack using this script:

```
aws cloudformation delete-stack --stack-name wordpress
```

Remember : use the name of the stack whatever you give while creating and deleting.

Helping tips:

If your role is created and you are unable to see your wordpress or EFS then connect to your created instance and run each command one by one to see the result , in case of any error command change it and find its alternative and edit it in your script in case alternative command succeeded.

SCRIPT:

```
---
AWSTemplateFormatVersion: "2010-09-09"

Description: >
  Here are some
  details about
  the template.

Resources:
  ALBSecuritySG1:
    Type: AWS::EC2::SecurityGroup
    Properties:
      VpcId: "vpc-0e0feebb6af022e5b"
      GroupDescription: Allow http for loadbalance
      GroupName: "ALBSecuritySG1"
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
        - IpProtocol: tcp
          FromPort: 443
          ToPort: 443
          CidrIp: 0.0.0.0/0

  MyEC2LAMPSecGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      VpcId: "vpc-0e0feebb6af022e5b"
      GroupDescription: "Allow ssh for all and http for loadbalance"
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: 0.0.0.0/0
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          SourceSecurityGroupId: !GetAtt ALBSecuritySG1.GroupId
        - IpProtocol: tcp
          FromPort: 443
          ToPort: 443
          SourceSecurityGroupId: !GetAtt ALBSecuritySG1.GroupId

  MyWPSecGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      VpcId: "vpc-0e0feebb6af022e5b"
      GroupDescription: "Allow ssh for all and http for loadbalance"
      SecurityGroupIngress:
```

```
      - IpProtocol: tcp
        FromPort: 22
        ToPort: 22
        CidrIp: 0.0.0.0/0
      - IpProtocol: tcp
        FromPort: 80
        ToPort: 80
        CidrIp: 0.0.0.0/0
      - IpProtocol: tcp
        FromPort: 443
        ToPort: 443
        CidrIp: 0.0.0.0/0

MyEfsSecGroup1:
  Type: AWS::EC2::SecurityGroup
  Properties:
    VpcId: "vpc-0e0feebb6af022e5b"
    GroupDescription: "Allow ssh for all"
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 22
        ToPort: 22
        CidrIp: 0.0.0.0/0

MyEfsSecGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    VpcId: "vpc-0e0feebb6af022e5b"
    GroupDescription: "Allow ssh for all"
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 2049
        ToPort: 2049
        CidrIp: 0.0.0.0/0
      - IpProtocol: tcp
        FromPort: 22
        ToPort: 22
        SourceSecurityGroupId: !GetAtt MyEfsSecGroup1.GroupId

MyRDSSecGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    VpcId: "vpc-0e0feebb6af022e5b"
    GroupDescription: "Allow Rds"
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 3306
        ToPort: 3306
        CidrIp: 0.0.0.0/0

MyEC2launchInstance:
  Type: AWS::EC2::LaunchTemplate
  Properties:
    LaunchTemplateName: "MyLampLaunchTemplate"
```

```
    LaunchTemplateData:
      ImageId: "ami-06ed60ed1369448bd"
      KeyName: "amna"
      InstanceType: "t2.micro"
      SecurityGroupIds:
        - !GetAtt MyEC2LAMPSecGroup.GroupId
      UserData:
        Fn::Base64: !Sub |
          #!/bin/bash
          dnf update -y
          dnf install -y httpd wget php-fpm php-mysqli php-json php php-devel
          systemctl start httpd
          systemctl enable httpd
          dnf install -y amazon-efs-utils
          sudo mount -t efs -o tls ${MyEFS}:/ /var/www/html
  DependsOn:
    - EFSMountTarget1
    - EFSMountTarget2
    - EFSMountTarget3

MyWPProv:
  Type: AWS::EC2::Instance
  Properties:
      ImageId: "ami-06ed60ed1369448bd"
      KeyName: "amna"
      InstanceType: "t2.micro"
      SecurityGroupIds:
        - !GetAtt MyWPSecGroup.GroupId
      UserData:
        Fn::Base64: !Sub |
          #!/bin/bash
          dnf install -y amazon-efs-utils
          sudo mount -t efs -o tls ${MyEFS}:/ /var/www/html
          dnf install -y httpd wget php-fpm php-mysqli php-json php php-devel
          systemctl start httpd
          systemctl enable httpd
          usermod -a -G apache ec2-user
          chown -R ec2-user:apache /var/www
          chmod 2775 /var/www && find /var/www -type d -exec sudo chmod 2775 {} \;
          find /var/www -type f -exec sudo chmod 0664 {} \;
          wget https://wordpress.org/latest.tar.gz
          tar -xzvf latest.tar.gz
          cp wordpress/wp-config-sample.php wordpress/wp-config.php
          cp -r wordpress/* /var/www/html
          sed -i -e 's/database_name_here/wordpressdb/g' /var/www/html/wp-config.php
          sed -i -e 's/username_here/Admin/g' /var/www/html/wp-config.php
          sed -i -e 's/password_here/Amna1234/g' /var/www/html/wp-config.php
          sed -i -e 's/localhost/${MyDB.Endpoint.Address}/g' /var/www/html/wp-config.php
  DependsOn:
    - EFSMountTarget1
    - EFSMountTarget2
    - EFSMountTarget3
    - MyDB
```

```
MyDB:
  Type: AWS::RDS::DBInstance
  Properties:
    AllocatedStorage: 20
    DBInstanceClass: db.t3.micro
    Engine: MySQL
    MasterUsername: Admin
    MasterUserPassword: Amna1234
    DBName: wordpressdb
    VPCSecurityGroups:
      - !GetAtt MyRDSSecGroup.GroupId
    MultiAZ: false
    BackupRetentionPeriod: 0

MyEFS:
  Type: AWS::EFS::FileSystem
  Properties:
    PerformanceMode: generalPurpose
    ThroughputMode: bursting
    Encrypted: true
    LifecyclePolicies:
      - TransitionToIA: AFTER_30_DAYS

EFSMountTarget1:
  Type: AWS::EFS::MountTarget
  Properties:
    FileSystemId: !Ref MyEFS
    SubnetId: subnet-0dba558a60b154c88
    SecurityGroups:
      - !GetAtt MyEfsSecGroup.GroupId

EFSMountTarget2:
  Type: AWS::EFS::MountTarget
  Properties:
    FileSystemId: !Ref MyEFS
    SubnetId: subnet-00bf6b2d4eed1742a
    SecurityGroups:
      - !GetAtt MyEfsSecGroup.GroupId

EFSMountTarget3:
  Type: AWS::EFS::MountTarget
  Properties:
    FileSystemId: !Ref MyEFS
    SubnetId: subnet-0a1456112e3ea7b35
    SecurityGroups:
      - !GetAtt MyEfsSecGroup.GroupId


MyTargetGroup:
  Type: AWS::ElasticLoadBalancingV2::TargetGroup
  Properties:
    Name: "MyTargetGroup"
    Port: 80
    Protocol: "HTTP"
```

```
    VpcId: "vpc-0e0feebb6af022e5b"


ALB:
  Type: AWS::ElasticLoadBalancingV2::LoadBalancer
  Properties:
    Name:  "ALB"
    Type: "application"
    SecurityGroups:
      - !Ref ALBSecuritySG1
    Subnets:
      - subnet-0dba558a60b154c88
      - subnet-00bf6b2d4eed1742a
      - subnet-0a1456112e3ea7b35

Listenerupp1:
 Type: AWS::ElasticLoadBalancingV2::Listener
 DependsOn:
  - MyTargetGroup
  - ALB
 Properties:
  DefaultActions:
   - Type: forward
     TargetGroupArn: !Ref MyTargetGroup
  LoadBalancerArn: !Ref ALB
  Port: 80
  Protocol: HTTP

MyAutoScalingGroup:
  Type: "AWS::AutoScaling::AutoScalingGroup"
  Properties:
    AutoScalingGroupName: "MyAutoScalingGroup"
    LaunchTemplate:
      LaunchTemplateId: !Ref MyEC2launchInstance
      Version: !GetAtt MyEC2launchInstance.LatestVersionNumber
    MinSize: "2"
    MaxSize: "4"
    DesiredCapacity: "2"
    AvailabilityZones:
      - eu-west-1a
      - eu-west-1b
    TargetGroupARNs:
      - !Ref MyTargetGroup
```