Lab 7: Capstone Lab - Build an AWS multi-tier architecture

Lab overview

You are tasked with applying your new knowledge to solve several architectural challenges within a specific business case. First, you are given a list of requirements related to the design. Then, you perform a series of actions to deploy and configure the services needed to meet the requirements.

During this capstone lab, you have access to the following:

- Downloadable AWS CloudFormation templates and other lab files
- Task scenarios, descriptions, and requirements
- Optional step-by-step instructions, if needed

The task scenarios provide relevant background and help you understand how the requirements solve a real-world business problem. Use the templates and the requirements list to complete all of the tasks in the capstone. Now that you are familiar with concepts and services, this lab solidifies your knowledge through practice. In the real world, you encounter problems that are not well-defined or sequenced logically. By the end of this capstone, you should have a better understanding of how you can apply architectural best practices to real-world problems.

Objectives

After completing this lab, you should be able to do the following:

- Deploy a virtual network spread across multiple Availability Zones in a Region using a CloudFormation template.
- Deploy a highly available and fully managed relational database across those Availability Zones (AZ) using Amazon Relational Database Service (Amazon RDS).
- Use Amazon Elastic File System (Amazon EFS) to provision a shared storage layer across multiple Availability Zones for the application tier, powered by Network File System (NFS).
- Create a group of web servers that automatically scales in response to load variations to complete your application tier.

Lab scenario

Example Corp. creates marketing campaigns for small-sized to medium-sized businesses. They recently hired you to work with the engineering teams to build out a proof of concept for their business. To date, they have hosted their client-facing application in an on-premises data center, but they recently decided to move their operations to the cloud in an effort to save money and transform their business with a cloud-first approach. Some members of their team have cloud experience and recommended the AWS Cloud services to build their solution.

In addition, they decided to redesign their web portal. Customers use the portal to access their accounts, create marketing plans, and run data analysis on their marketing campaigns. They would like to have a working prototype in two weeks. You must design an architecture to support this application. Your solution must be fast, durable, scalable, and more cost-effective than their existing on-premises infrastructure.

The following image shows the final architecture of the designed solution:

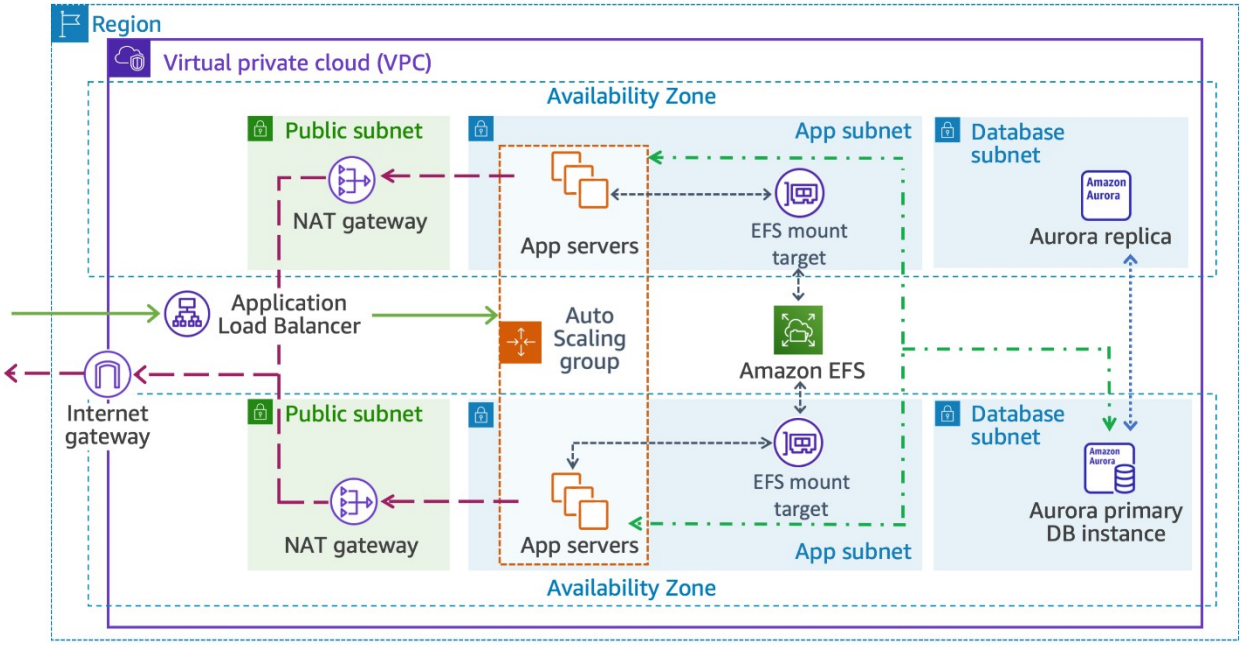


Image description: Preceding image depicts the entire architecture which will be created by the end of this lab. We have Amazon VPC to facilitate public and private networking using Public Subnets and Private Subnets. To ensure we are able to accept incoming internet traffic and allow outgoing traffic to the internet, we have Internet Gateway and NAT Gateways. In-order to facilitate high-availability (HA) for our architecture we have Application Load Balancer (ALB) and Auto-Scaling Group (ASG). There is also a fully scalable and elastic file server using Amazon EFS which can work across multiple availability zones (AZ)s. For database we are using

Amazon Aurora which works across multiple availability zones (AZ)s with primary in one availability zone (AZ) and a read-replica in another availability zone (AZ).

Task 1: Review and run a pre-configured CloudFormation template

In this task, you design and deploy a highly available and scalable deployment of the WordPress application stack. You use a CloudFormation template to deploy the networking resources required to support the application.

Task 1: Scenario

The company wants to deploy a new web application on WordPress on their existing web hosting account. The first step is to gather the requirements for the architecture design. To help you get started, the network team gave you a list of requirements to ensure there are no issues with the existing landscape. This includes the Classless Inter-Domain Routing (CIDR) range for the Amazon Virtual Private Cloud (Amazon VPC). Using this address range, build a VPC with two public subnets, two private application subnets, and two private database subnets. Make sure you attach an internet gateway to the VPC with a NAT gateway for routing traffic. They also requested two Elastic IP addresses. Ensure that you associate the following:

- The public subnet with the internet gateway
- The private application subnet and private database subnet with the NAT gateway

The network team also provided the route tables needed with their subnet associations.

Your cloud engineer has transferred the information request to a CloudFormation template and set up the security groups and outputs needed for future deployments. Please review the CloudFormation template with your cloud engineer, deploy the template, and check back with the network team to validate that the build meets all of their requirements.

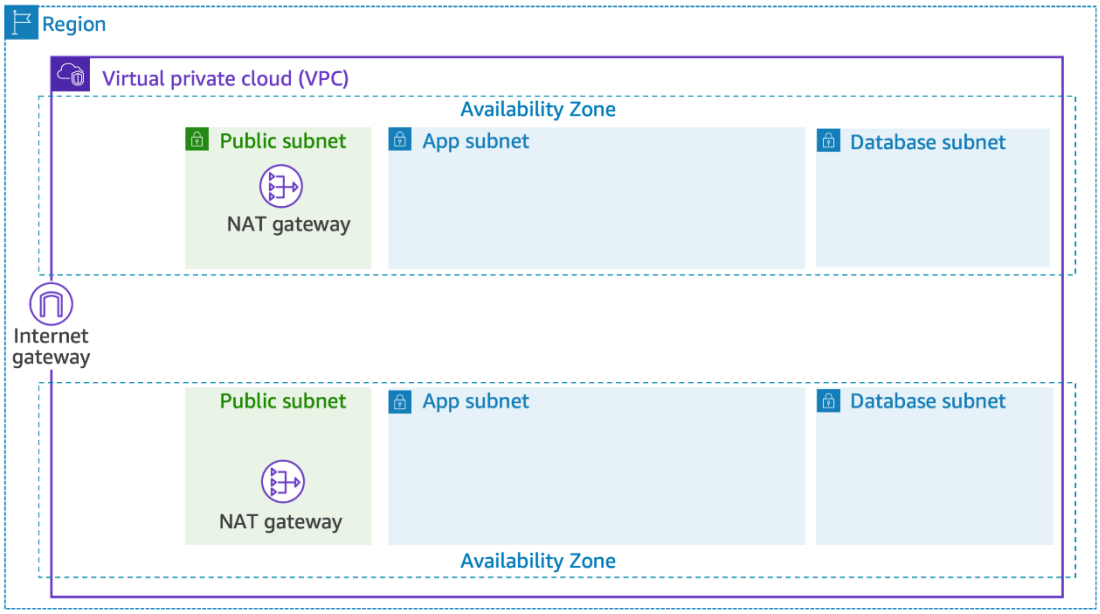
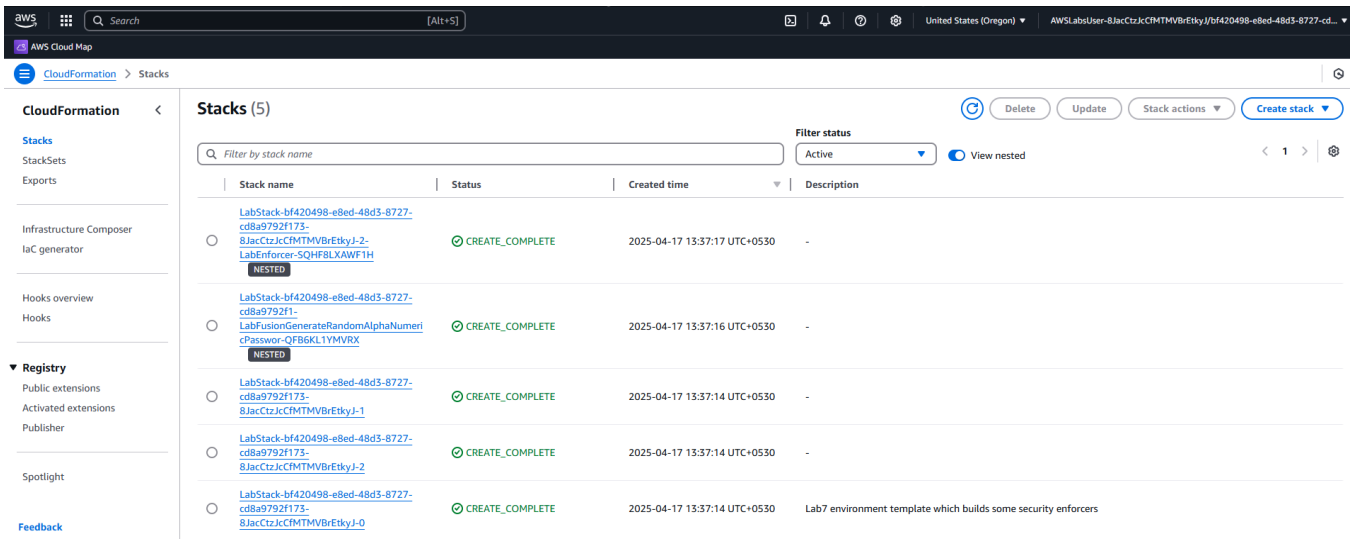


Image description: Preceding image depicts the entire architecture which will be created by the end of this lab. We have Amazon VPC to facilitate public and private networking using Public Subnets and Private Subnets. To ensure we are able to accept incoming internet traffic and allow outgoing traffic to the internet, we have Internet Gateway and NAT Gateways.

Task 1 instructions: Review and run a pre-configured CloudFormation template

Task 1.1: Navigate to the CloudFormation console

- At the top of the AWS Management Console, in the search box, search for and choose `CloudFormation`.



Task 1.2: Obtain and review the CloudFormation template

- Open the context (right-click) menu on this Task1.yaml link, and choose the option to save the CloudFormation template to your computer.
- Open the downloaded file in a text editor (not a word processor, preferably in Visual Studio Code).
- Review the CloudFormation template.
- Predict what resources are created by this template.

The `Task1.yaml` file is an AWS CloudFormation template that automatically sets up a complete virtual network environment in the AWS cloud. It contains all the instructions needed to create a Virtual Private Cloud (VPC) with both public and private subnets, allowing you to organize and secure your cloud resources. This includes components like an internet gateway for public internet access, NAT gateways for private subnet access to the internet, route tables to manage traffic flow, and security groups that act like firewalls to control access. In total, it sets up six subnets (two each for public, application, and database layers), elastic IP addresses, and connectivity rules—making it a solid foundation for deploying multi-tier applications securely in the cloud.

The `Task1.yaml` file consists of the following:

```yaml
AWSTemplateFormatVersion: 2010-09-09 # Specifies the version of the AWS
CloudFormation template syntax.

Description: Lab7 Task 1 template which builds VPC, supporting
resources, a basic networking structure, and some Security groups for
use in later tasks. # A description of the template.

Parameters: # Defines input parameters that can be specified when the
stack is created or updated.
  VPCCIDR: # Parameter for the VPC CIDR block.
    Description: CIDR Block for VPC # Description of the VPCCIDR
parameter.
    Type: String # Specifies the data type of the parameter.
    Default: 10.0.0.0/16 # Default value for the VPCCIDR parameter.
    AllowedValues: # Restricts the values that can be specified for
this parameter.
      - 10.0.0.0/16 # Allowed value for the VPCCIDR parameter.

  PublicSubnet1Param: # Parameter for the Public Subnet 1 CIDR block.
    Description: Public Subnet 1 # Description of the
PublicSubnet1Param parameter.
    Type: String # Specifies the data type of the parameter.
    Default: 10.0.0.0/24 # Default value for the PublicSubnet1Param
parameter.
```

```
    AllowedValues: # Restricts the values that can be specified for
this parameter.
      - 10.0.0.0/24 # Allowed value for the PublicSubnet1Param
parameter.

  PublicSubnet2Param: # Parameter for the Public Subnet 2 CIDR block.
    Description: Public Subnet 2 # Description of the
PublicSubnet2Param parameter.
    Type: String # Specifies the data type of the parameter.
    Default: 10.0.1.0/24 # Default value for the PublicSubnet2Param
parameter.
    AllowedValues: # Restricts the values that can be specified for
this parameter.
      - 10.0.1.0/24 # Allowed value for the PublicSubnet2Param
parameter.

  AppSubnet1Param: # Parameter for the App Subnet 1 CIDR block.
    Description: App Subnet 1 # Description of the AppSubnet1Param
parameter.
    Type: String # Specifies the data type of the parameter.
    Default: 10.0.2.0/24 # Default value for the AppSubnet1Param
parameter.
    AllowedValues: # Restricts the values that can be specified for
this parameter.
      - 10.0.2.0/24 # Allowed value for the AppSubnet1Param parameter.

  AppSubnet2Param: # Parameter for the App Subnet 2 CIDR block.
    Description: App Subnet 2 # Description of the AppSubnet2Param
parameter.
    Type: String # Specifies the data type of the parameter.
    Default: 10.0.3.0/24 # Default value for the AppSubnet2Param
parameter.
    AllowedValues: # Restricts the values that can be specified for
this parameter.
      - 10.0.3.0/24 # Allowed value for the AppSubnet2Param parameter.

  DatabaseSubnet1Param: # Parameter for the Private Subnet 1 CIDR block
(intended for databases).
    Description: Private Subnet 1 # Description of the
DatabaseSubnet1Param parameter.
    Type: String # Specifies the data type of the parameter.
    Default: 10.0.4.0/24 # Default value for the DatabaseSubnet1Param
parameter.
    AllowedValues: # Restricts the values that can be specified for
this parameter.
      - 10.0.4.0/24 # Allowed value for the DatabaseSubnet1Param
parameter.
```

```
  DatabaseSubnet2Param: # Parameter for the Private Subnet 2 CIDR block
(intended for databases).
    Description: Private Subnet 2 # Description of the
DatabaseSubnet2Param parameter.
    Type: String # Specifies the data type of the parameter.
    Default: 10.0.5.0/24 # Default value for the DatabaseSubnet2Param
parameter.
    AllowedValues: # Restricts the values that can be specified for
this parameter.
      - 10.0.5.0/24 # Allowed value for the DatabaseSubnet2Param
parameter.

Resources: # Defines the AWS resources that will be created or updated
by the template.
############
# VPC and Network Structure
############
  LabVPC: # Logical ID of the VPC resource.
    Type: 'AWS::EC2::VPC' # AWS resource type for creating a VPC.
    Properties: # Properties that define the VPC.
      CidrBlock: !Ref VPCCIDR # The IPv4 CIDR block for the VPC,
referencing the VPCCIDR parameter.
      EnableDnsSupport: True # Indicates whether DNS resolution is
supported for the VPC.
      EnableDnsHostnames: True # Indicates whether instances launched
in the VPC get public DNS hostnames.
      InstanceTenancy: 'default' # The tenancy options for instances
launched in the VPC. 'default' means instances run on shared hardware.
      Tags: # Tags to apply to the VPC.
        - Key: Name # Key of the tag.
          Value: LabVPC # Value of the tag.

  LabInternetGateway: # Logical ID of the Internet Gateway resource.
    Type: 'AWS::EC2::InternetGateway' # AWS resource type for creating
an Internet Gateway.

  AttachGateway: # Logical ID of the VPC Gateway Attachment resource.
    Type: 'AWS::EC2::VPCGatewayAttachment' # AWS resource type for
attaching the Internet Gateway to the VPC.
    Properties: # Properties that define the VPC Gateway Attachment.
      VpcId: !Ref LabVPC # The ID of the VPC to attach to, referencing
the LabVPC resource.
      InternetGatewayId: !Ref LabInternetGateway # The ID of the
Internet Gateway to attach, referencing the LabInternetGateway
resource.

#NATs
  NATGateway1: # Logical ID of the first NAT Gateway resource.
```

```
    Type: AWS::EC2::NatGateway # AWS resource type for creating a NAT
Gateway.
    Properties: # Properties that define the NAT Gateway.
      AllocationId: !GetAtt ElasticIPAddress1.AllocationId # The
allocation ID of the Elastic IP address to associate with the NAT
Gateway.
      SubnetId: !Ref PublicSubnet1 # The ID of the public subnet in
which to create the NAT Gateway.
      Tags: # Tags to apply to the NAT Gateway.
        - Key: Name # Key of the tag.
          Value: NATGateway1 # Value of the tag.

  ElasticIPAddress1: # Logical ID of the first Elastic IP Address
resource.
    Type: AWS::EC2::EIP # AWS resource type for creating an Elastic IP
address.
    Properties: # Properties that define the Elastic IP address.
      Domain: vpc # Specifies that the EIP is for use in a VPC.

  NATGateway2: # Logical ID of the second NAT Gateway resource.
    Type: AWS::EC2::NatGateway # AWS resource type for creating a NAT
Gateway.
    Properties: # Properties that define the NAT Gateway.
      AllocationId: !GetAtt ElasticIPAddress2.AllocationId # The
allocation ID of the Elastic IP address to associate with the NAT
Gateway.
      SubnetId: !Ref PublicSubnet2 # The ID of the public subnet in
which to create the NAT Gateway.
      Tags: # Tags to apply to the NAT Gateway.
        - Key: Name # Key of the tag.
          Value: NATGateway2 # Value of the tag.

  ElasticIPAddress2: # Logical ID of the second Elastic IP Address
resource.
    Type: AWS::EC2::EIP # AWS resource type for creating an Elastic IP
address.
    Properties: # Properties that define the Elastic IP address.
      Domain: vpc # Specifies that the EIP is for use in a VPC.

#Subnets
  PublicSubnet1: # Logical ID of the first Public Subnet resource.
    Type: 'AWS::EC2::Subnet' # AWS resource type for creating a Subnet.
    Properties: # Properties that define the Subnet.
      VpcId: !Ref LabVPC # The ID of the VPC in which to create the
subnet, referencing the LabVPC resource.
      CidrBlock: !Ref PublicSubnet1Param # The IPv4 CIDR block for the
subnet, referencing the PublicSubnet1Param parameter.
```

```
      MapPublicIpOnLaunch: True # Indicates whether instances launched
in this subnet receive a public IPv4 address.
      AvailabilityZone: !Select # Selects an Availability Zone from a
list.
        - '0' # Index of the Availability Zone to select (the first
one).
        - !GetAZs '' # Intrinsic function to get a list of Availability
Zones in the current region.
      Tags: # Tags to apply to the Subnet.
        - Key: Name # Key of the tag.
          Value: PublicSubnet1 # Value of the tag.

  PublicSubnet2: # Logical ID of the second Public Subnet resource.
    Type: 'AWS::EC2::Subnet' # AWS resource type for creating a Subnet.
    Properties: # Properties that define the Subnet.
      VpcId: !Ref LabVPC # The ID of the VPC in which to create the
subnet, referencing the LabVPC resource.
      CidrBlock: !Ref PublicSubnet2Param # The IPv4 CIDR block for the
subnet, referencing the PublicSubnet2Param parameter.
      MapPublicIpOnLaunch: True # Indicates whether instances launched
in this subnet receive a public IPv4 address.
      AvailabilityZone: !Select # Selects an Availability Zone from a
list.
        - '1' # Index of the Availability Zone to select (the second
one).
        - !GetAZs '' # Intrinsic function to get a list of Availability
Zones in the current region.
      Tags: # Tags to apply to the Subnet.
        - Key: Name # Key of the tag.
          Value: PublicSubnet2 # Value of the tag.

  AppSubnet1: # Logical ID of the first Application Subnet resource.
    Type: 'AWS::EC2::Subnet' # AWS resource type for creating a Subnet.
    Properties: # Properties that define the Subnet.
      VpcId: !Ref LabVPC # The ID of the VPC in which to create the
subnet, referencing the LabVPC resource.
      CidrBlock: !Ref AppSubnet1Param # The IPv4 CIDR block for the
subnet, referencing the AppSubnet1Param parameter.
      MapPublicIpOnLaunch: False # Indicates whether instances launched
in this subnet receive a public IPv4 address (set to false for private
subnets).
      AvailabilityZone: !Select # Selects an Availability Zone from a
list.
        - '0' # Index of the Availability Zone to select (the first
one).
        - !GetAZs '' # Intrinsic function to get a list of Availability
Zones in the current region.
      Tags: # Tags to apply to the Subnet.
```

```
        - Key: Name # Key of the tag.
          Value: AppSubnet1 # Value of the tag.

  AppSubnet2: # Logical ID of the second Application Subnet resource.
    Type: 'AWS::EC2::Subnet' # AWS resource type for creating a Subnet.
    Properties: # Properties that define the Subnet.
      VpcId: !Ref LabVPC # The ID of the VPC in which to create the
subnet, referencing the LabVPC resource.
      CidrBlock: !Ref AppSubnet2Param # The IPv4 CIDR block for the
subnet, referencing the AppSubnet2Param parameter.
      MapPublicIpOnLaunch: False # Indicates whether instances launched
in this subnet receive a public IPv4 address (set to false for private
subnets).
      AvailabilityZone: !Select # Selects an Availability Zone from a
list.
        - '1' # Index of the Availability Zone to select (the second
one).
        - !GetAZs '' # Intrinsic function to get a list of Availability
Zones in the current region.
      Tags: # Tags to apply to the Subnet.
        - Key: Name # Key of the tag.
          Value: AppSubnet2 # Value of the tag.

  DatabaseSubnet1: # Logical ID of the first Database Subnet resource.
    Type: 'AWS::EC2::Subnet' # AWS resource type for creating a Subnet.
    Properties: # Properties that define the Subnet.
      VpcId: !Ref LabVPC # The ID of the VPC in which to create the
subnet, referencing the LabVPC resource.
      CidrBlock: !Ref DatabaseSubnet1Param # The IPv4 CIDR block for
the subnet, referencing the DatabaseSubnet1Param parameter.
      MapPublicIpOnLaunch: False # Indicates whether instances launched
in this subnet receive a public IPv4 address (set to false for private
subnets).
      AvailabilityZone: !Select # Selects an Availability Zone from a
list.
        - '0' # Index of the Availability Zone to select (the first
one).
        - !GetAZs '' # Intrinsic function to get a list of Availability
Zones in the current region.
      Tags: # Tags to apply to the Subnet.
        - Key: Name # Key of the tag.
          Value: DatabaseSubnet1 # Value of the tag.

  DatabaseSubnet2: # Logical ID of the second Database Subnet resource.
    Type: 'AWS::EC2::Subnet' # AWS resource type for creating a Subnet.
    Properties: # Properties that define the Subnet.
      VpcId: !Ref LabVPC # The ID of the VPC in which to create the
subnet, referencing the LabVPC resource.
```

```
      CidrBlock: !Ref DatabaseSubnet2Param # The IPv4 CIDR block for
the subnet, referencing the DatabaseSubnet2Param parameter.
      MapPublicIpOnLaunch: False # Indicates whether instances launched
in this subnet receive a public IPv4 address (set to false for private
subnets).
      AvailabilityZone: !Select # Selects an Availability Zone from a
list.
        - '1' # Index of the Availability Zone to select (the second
one).
        - !GetAZs '' # Intrinsic function to get a list of Availability
Zones in the current region.
      Tags: # Tags to apply to the Subnet.
        - Key: Name # Key of the tag.
          Value: DatabaseSubnet2 # Value of the tag.

#Routing
#Route Tables
  PublicRouteTable: # Logical ID of the Public Route Table resource.
    Type: 'AWS::EC2::RouteTable' # AWS resource type for creating a
Route Table.
    Properties: # Properties that define the Route Table.
      VpcId: !Ref LabVPC # The ID of the VPC with which the route table
is associated, referencing the LabVPC resource.
      Tags: # Tags to apply to the Route Table.
        - Key: Name # Key of the tag.
          Value: PublicRouteTable # Value of the tag.

  PrivateRouteTableAZ1: # Logical ID of the Private Route Table for
Availability Zone 1.
    Type: 'AWS::EC2::RouteTable' # AWS resource type for creating a
Route Table.
    Properties: # Properties that define the Route Table.
      VpcId: !Ref LabVPC # The ID of the VPC with which the route table
is associated, referencing the LabVPC resource.
      Tags: # Tags to apply to the Route Table.
        - Key: Name # Key of the tag.
          Value: PrivateRouteTableAZ1 # Value of the tag.

  PrivateRouteTableAZ2: # Logical ID of the Private Route Table for
Availability Zone 2.
    Type: 'AWS::EC2::RouteTable' # AWS resource type for creating a
Route Table.
    Properties: # Properties that define the Route Table.
      VpcId: !Ref LabVPC # The ID of the VPC with which the route table
is associated, referencing the LabVPC resource.
      Tags: # Tags to apply to the Route Table.
        - Key: Name # Key of the tag.
          Value: PrivateRouteTableAZ2 # Value of the tag.
```

```
#Routes
  PublicRoute: # Logical ID of the Public Route resource.
    Type: 'AWS::EC2::Route' # AWS resource type for creating a Route.
    Properties: # Properties that define the Route.
      RouteTableId: !Ref PublicRouteTable # The ID of the route table
to associate with the route, referencing the PublicRouteTable resource.
      DestinationCidrBlock: 0.0.0.0/0 # The destination IPv4 CIDR block
for the route (all traffic).
      GatewayId: !Ref LabInternetGateway # The ID of an internet
gateway or virtual private gateway for the route, referencing the
LabInternetGateway resource.

  PrivateRouteAZ1: # Logical ID of the Private Route for Availability
Zone 1.
    Type: 'AWS::EC2::Route' # AWS resource type for creating a Route.
    Properties: # Properties that define the Route.
      RouteTableId: !Ref PrivateRouteTableAZ1 # The ID of the route
table to associate with the route, referencing the PrivateRouteTableAZ1
resource.
      DestinationCidrBlock: 0.0.0.0/0 # The destination IPv4 CIDR block
for the route (all traffic).
      NatGatewayId: !Ref NATGateway1 # The ID of a NAT gateway for the
route, referencing the NATGateway1 resource.

  PrivateRouteAZ2: # Logical ID of the Private Route for Availability
Zone 2.
    Type: 'AWS::EC2::Route' # AWS resource type for creating a Route.
    Properties: # Properties that define the Route.
      RouteTableId: !Ref PrivateRouteTableAZ2 # The ID of the route
table to associate with the route, referencing the PrivateRouteTableAZ2
resource.
      DestinationCidrBlock: 0.0.0.0/0 # The destination IPv4 CIDR block
for the route (all traffic).
      NatGatewayId: !Ref NATGateway2 # The ID of a NAT gateway for the
route, referencing the NATGateway2 resource.
#Subnet Associations
  PublicSubnet1RouteTableAssociation: # Logical ID for associating
PublicSubnet1 with the PublicRouteTable.
    Type: 'AWS::EC2::SubnetRouteTableAssociation' # AWS resource type
for associating a subnet with a route table.
    Properties: # Properties that define the subnet route table
association.
      SubnetId: !Ref PublicSubnet1 # The ID of the subnet to associate,
referencing the PublicSubnet1 resource.
      RouteTableId: !Ref PublicRouteTable # The ID of the route table
to associate with the subnet, referencing the PublicRouteTable
resource.
```

```
  PublicSubnet2RouteTableAssociation: # Logical ID for associating
PublicSubnet2 with the PublicRouteTable.
    Type: 'AWS::EC2::SubnetRouteTableAssociation' # AWS resource type
for associating a subnet with a route table.
    Properties: # Properties that define the subnet route table
association.
      SubnetId: !Ref PublicSubnet2 # The ID of the subnet to associate,
referencing the PublicSubnet2 resource.
      RouteTableId: !Ref PublicRouteTable # The ID of the route table
to associate with the subnet, referencing the PublicRouteTable
resource.

  AppSubnet1RouteTableAssociation: # Logical ID for associating
AppSubnet1 with the PrivateRouteTableAZ1.
    Type: 'AWS::EC2::SubnetRouteTableAssociation' # AWS resource type
for associating a subnet with a route table.
    Properties: # Properties that define the subnet route table
association.
      SubnetId: !Ref AppSubnet1 # The ID of the subnet to associate,
referencing the AppSubnet1 resource.
      RouteTableId: !Ref PrivateRouteTableAZ1 # The ID of the route
table to associate with the subnet, referencing the
PrivateRouteTableAZ1 resource.

  AppSubnet2RouteTableAssociation: # Logical ID for associating
AppSubnet2 with the PrivateRouteTableAZ2.
    Type: 'AWS::EC2::SubnetRouteTableAssociation' # AWS resource type
for associating a subnet with a route table.
    Properties: # Properties that define the subnet route table
association.
      SubnetId: !Ref AppSubnet2 # The ID of the subnet to associate,
referencing the AppSubnet2 resource.
      RouteTableId: !Ref PrivateRouteTableAZ2 # The ID of the route
table to associate with the subnet, referencing the
PrivateRouteTableAZ2 resource.

  DatabaseSubnet1RouteTableAssociation: # Logical ID for associating
DatabaseSubnet1 with the PrivateRouteTableAZ1.
    Type: 'AWS::EC2::SubnetRouteTableAssociation' # AWS resource type
for associating a subnet with a route table.
    Properties: # Properties that define the subnet route table
association.
      SubnetId: !Ref DatabaseSubnet1 # The ID of the subnet to
associate, referencing the DatabaseSubnet1 resource.
      RouteTableId: !Ref PrivateRouteTableAZ1 # The ID of the route
table to associate with the subnet, referencing the
PrivateRouteTableAZ1 resource.
```

```
  DatabaseSubnet2RouteTableAssociation: # Logical ID for associating
DatabaseSubnet2 with the PrivateRouteTableAZ2.
    Type: 'AWS::EC2::SubnetRouteTableAssociation' # AWS resource type
for associating a subnet with a route table.
    Properties: # Properties that define the subnet route table
association.
      SubnetId: !Ref DatabaseSubnet2 # The ID of the subnet to
associate, referencing the DatabaseSubnet2 resource.
      RouteTableId: !Ref PrivateRouteTableAZ2 # The ID of the route
table to associate with the subnet, referencing the
PrivateRouteTableAZ2 resource.

############
# Security Groups
############
  AppInstanceSecurityGroup: # Logical ID for the Application Instance
Security Group.
    Type: 'AWS::EC2::SecurityGroup' # AWS resource type for creating a
Security Group.
    Properties: # Properties that define the Security Group.
      GroupDescription: Security Group allowing HTTP traffic for lab
instances # A description of the security group.
      VpcId: !Ref LabVPC # The ID of the VPC to associate the security
group with, referencing the LabVPC resource.
      Tags: # Tags to apply to the Security Group.
        - Key: Name # Key of the tag.
          Value: AppInstanceSecurityGroup # Value of the tag.
      SecurityGroupIngress: # Defines the inbound rules for the
security group.
        - IpProtocol: tcp # Allows TCP traffic.
          FromPort: 80 # Allows traffic on port 80 (HTTP).
          ToPort: 80 # Allows traffic on port 80 (HTTP).
          CidrIp: 0.0.0.0/0 # Allows traffic from any IP address.

  RDSSecurityGroup: # Logical ID for the RDS Security Group.
    Type: 'AWS::EC2::SecurityGroup' # AWS resource type for creating a
Security Group.
    Properties: # Properties that define the Security Group.
      GroupDescription: Security Group allowing RDS instances to have
internet traffic # A description of the security group.
      VpcId: !Ref LabVPC # The ID of the VPC to associate the security
group with, referencing the LabVPC resource.
      Tags: # Tags to apply to the Security Group.
        - Key: Name # Key of the tag.
          Value: RDSSecurityGroup # Value of the tag.

  EFSMountTargetSecurityGroup: # Logical ID for the EFS Mount Target
Security Group.
```

```
    Type: 'AWS::EC2::SecurityGroup' # AWS resource type for creating a
Security Group.
    Properties: # Properties that define the Security Group.
      GroupDescription: Security Group allowing traffic between EFS
Mount Targets and Amazon EC2 instances # A description of the security
group.
      VpcId: !Ref LabVPC # The ID of the VPC to associate the security
group with, referencing the LabVPC resource.
      Tags: # Tags to apply to the Security Group.
        - Key: Name # Key of the tag.
          Value: EFSMountTargetSecurityGroup # Value of the tag.
      SecurityGroupIngress: # Defines the inbound rules for the
security group.
        - IpProtocol: tcp # Allows TCP traffic.
          SourceSecurityGroupId: !Ref AppInstanceSecurityGroup # Allows
traffic from the AppInstanceSecurityGroup.
          FromPort: 80 # Allows traffic on port 80.
          ToPort: 80 # Allows traffic on port 80.

Outputs: # Defines values that are returned when the stack is created.

  Region: # Logical ID of the Region output.
    Description: "Lab Region" # Description of the output value.
    Value: !Ref AWS::Region # Intrinsic function to get the AWS Region
in which the stack is being created.

  DatabaseSubnet1CIDR: # Logical ID of the DatabaseSubnet1CIDR output.
    Description: "CIDR block for the DB Subnet in AZ a" # Description
of the output value.
    Value: !Ref DatabaseSubnet1Param # References the
DatabaseSubnet1Param parameter value.

  DatabaseSubnet2CIDR: # Logical ID of the DatabaseSubnet2CIDR output.
    Description: "CIDR block for the DB Subnet in AZ b" # Description
of the output value.
    Value: !Ref DatabaseSubnet2Param # References the
DatabaseSubnet2Param parameter value.

  DatabaseSubnet1ID: # Logical ID of the DatabaseSubnet1ID output.
    Description: "The Subnet ID for the DB Subnet in AZ a" #
Description of the output value.
    Value: !Ref DatabaseSubnet1 # References the physical ID of the
DatabaseSubnet1 resource.
    Export: # Defines the output as an export value for cross-stack
references.
      Name: "DatabaseSubnet1ID" # The name used to export this output
value.
```

```
  DatabaseSubnet2ID: # Logical ID of the DatabaseSubnet2ID output.
    Description: "The Subnet ID for the DB Subnet in AZ b" #
Description of the output value.
    Value: !Ref DatabaseSubnet2 # References the physical ID of the
DatabaseSubnet2 resource.
    Export: # Defines the output as an export value for cross-stack
references.
      Name: "DatabaseSubnet2ID" # The name used to export this output
value.

  AppInstanceSecurityGroupID: # Logical ID of the
AppInstanceSecurityGroupID output.
    Description: "The Security Group ID for the Lab Instance Security
Group" # Description of the output value.
    Value: !Ref AppInstanceSecurityGroup # References the physical ID
of the AppInstanceSecurityGroup resource.
    Export: # Defines the output as an export value for cross-stack
references.
      Name: "AppInstanceSecurityGroupID" # The name used to export this
output value.

  EFSMountTargetSecurityGroupID: # Logical ID of the
EFSMountTargetSecurityGroupID output.
    Description: "The Security Group ID for the Lab EFS Mount Target" #
Description of the output value.
    Value: !Ref EFSMountTargetSecurityGroup # References the physical
ID of the EFSMountTargetSecurityGroup resource.
    Export: # Defines the output as an export value for cross-stack
references.
      Name: "EFSMountTargetSecurityGroupID" # The name used to export
this output value.

  RDSSecurityGroupID: # Logical ID of the RDSSecurityGroupID output.
    Description: "The Security Group ID for the Lab RDS cluster" #
Description of the output value.
    Value: !Ref RDSSecurityGroup # References the physical ID of the
RDSSecurityGroup resource.
    Export: # Defines the output as an export value for cross-stack
references.
      Name: "RDSSecurityGroupID" # The name used to export this output
value.

  VPCID: # Logical ID of the VPCID output.
    Description: "The VPC ID for the lab" # Description of the output
value.
    Value: !Ref LabVPC # References the physical ID of the LabVPC
resource.
```

```
    Export: # Defines the output as an export value for cross-stack
references.
      Name: "VPCID" # The name used to export this output value.
```

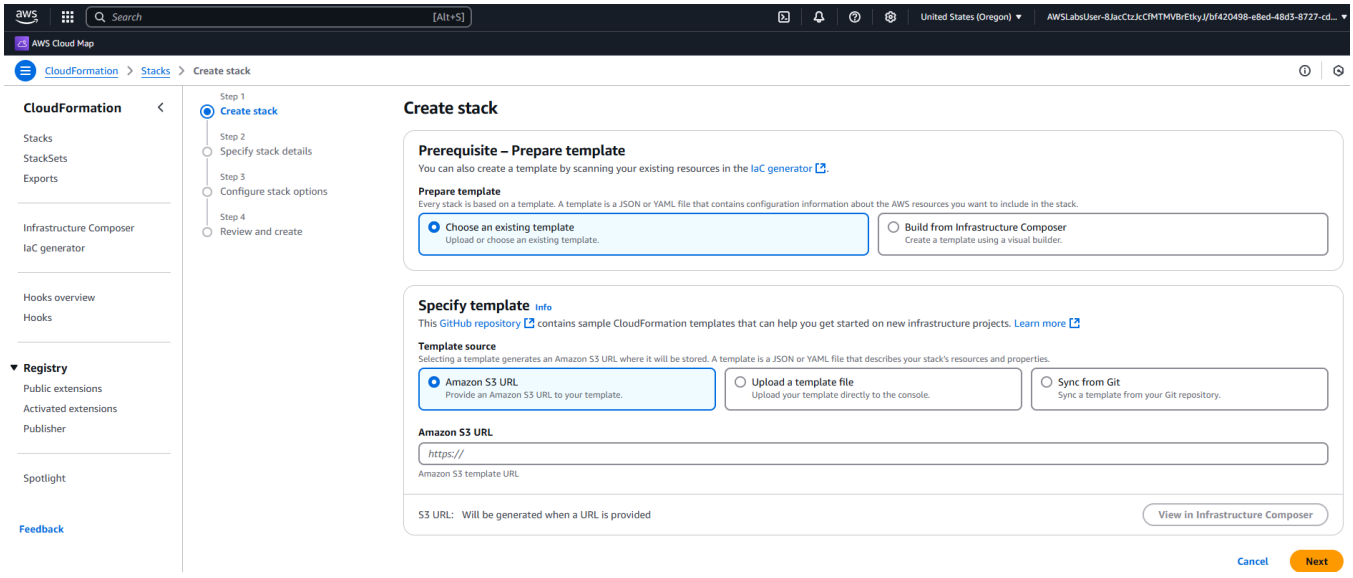
Task 1.3: Create the CloudFormation stack

- Choose **Create stack**.

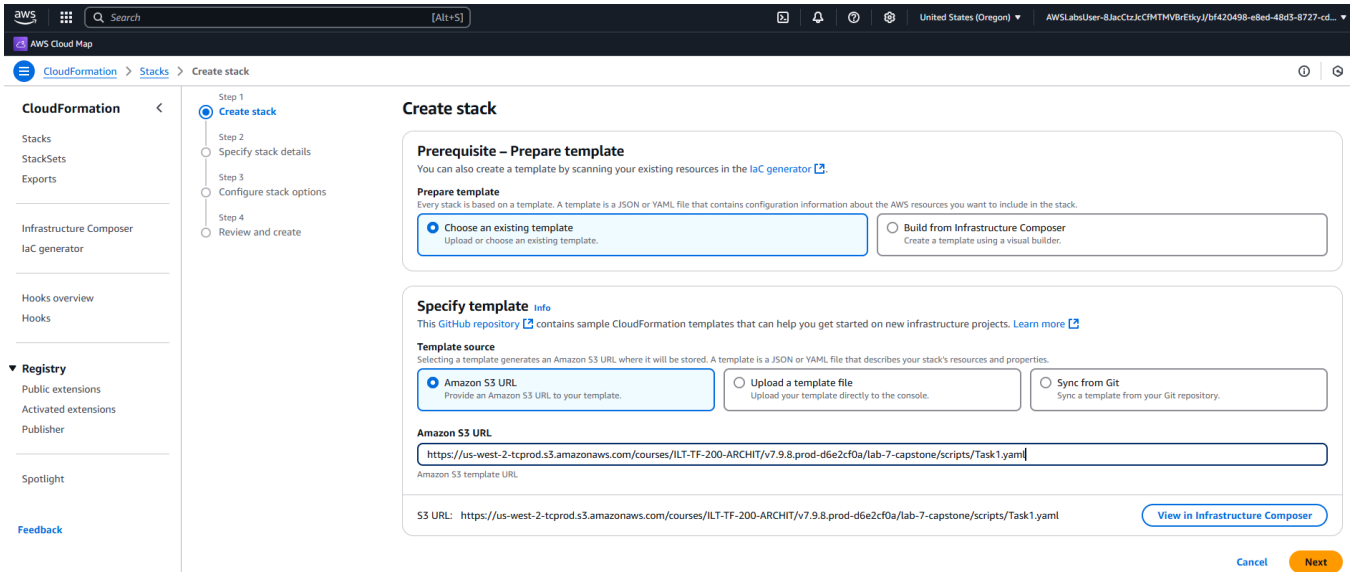
Note: If the console starts you on the Stacks page instead of the Amazon CloudFormation landing page, then you can get to the Create stack page in two steps.

  - Choose the **Create stack** dropdown menu.
  - Choose **With new resources (standard)**.

The **Create stack** page is displayed.

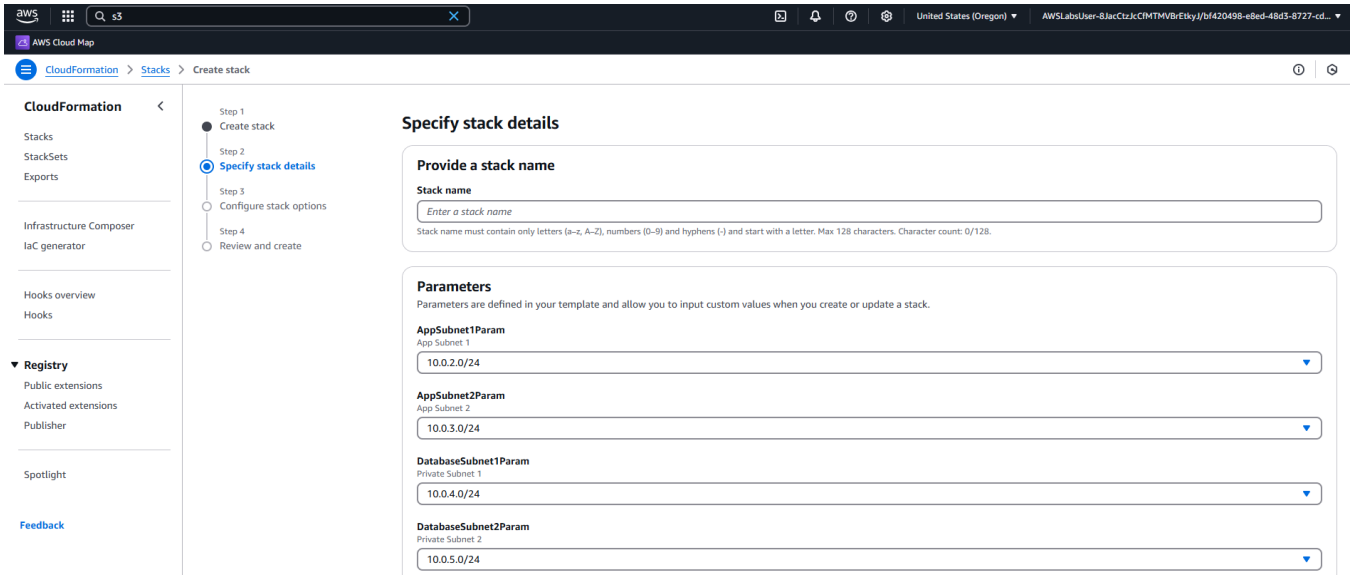


- Configure the following:
  - Select Choose an existing template.
  - Select Amazon S3 URL.
  - Copy the **Task1TemplateUrl** value (https://us-west-2-tcprod.s3.amazonaws.com/courses/ILT-TF-200-ARCHIT/v7.9.8.prod-d6e2cf0a/lab-7-capstone/scripts/Task1.yaml) and paste it in the **Amazon S3 URL** text box.

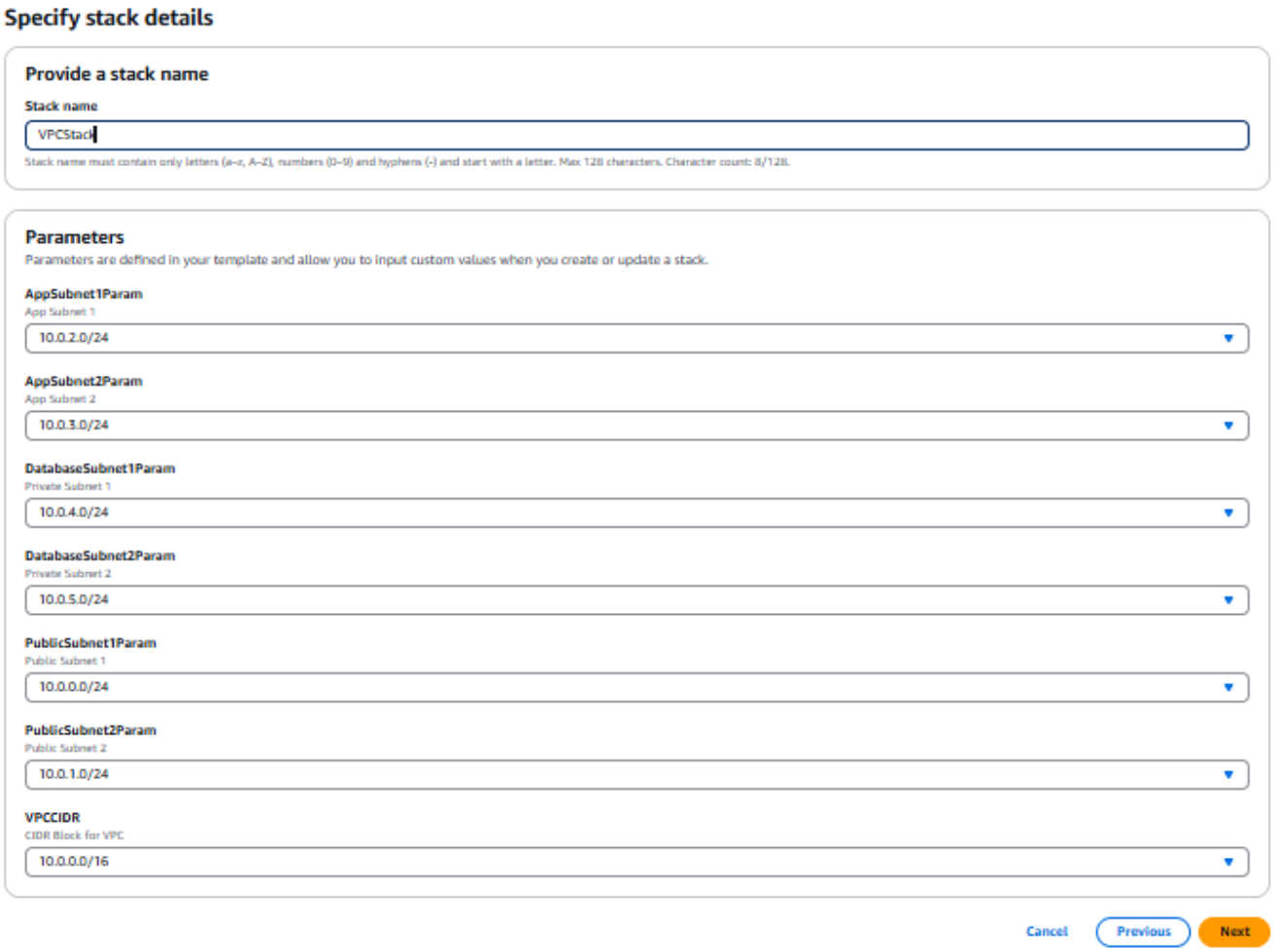


- Choose **Next**.

The **Specify stack details** page is displayed.

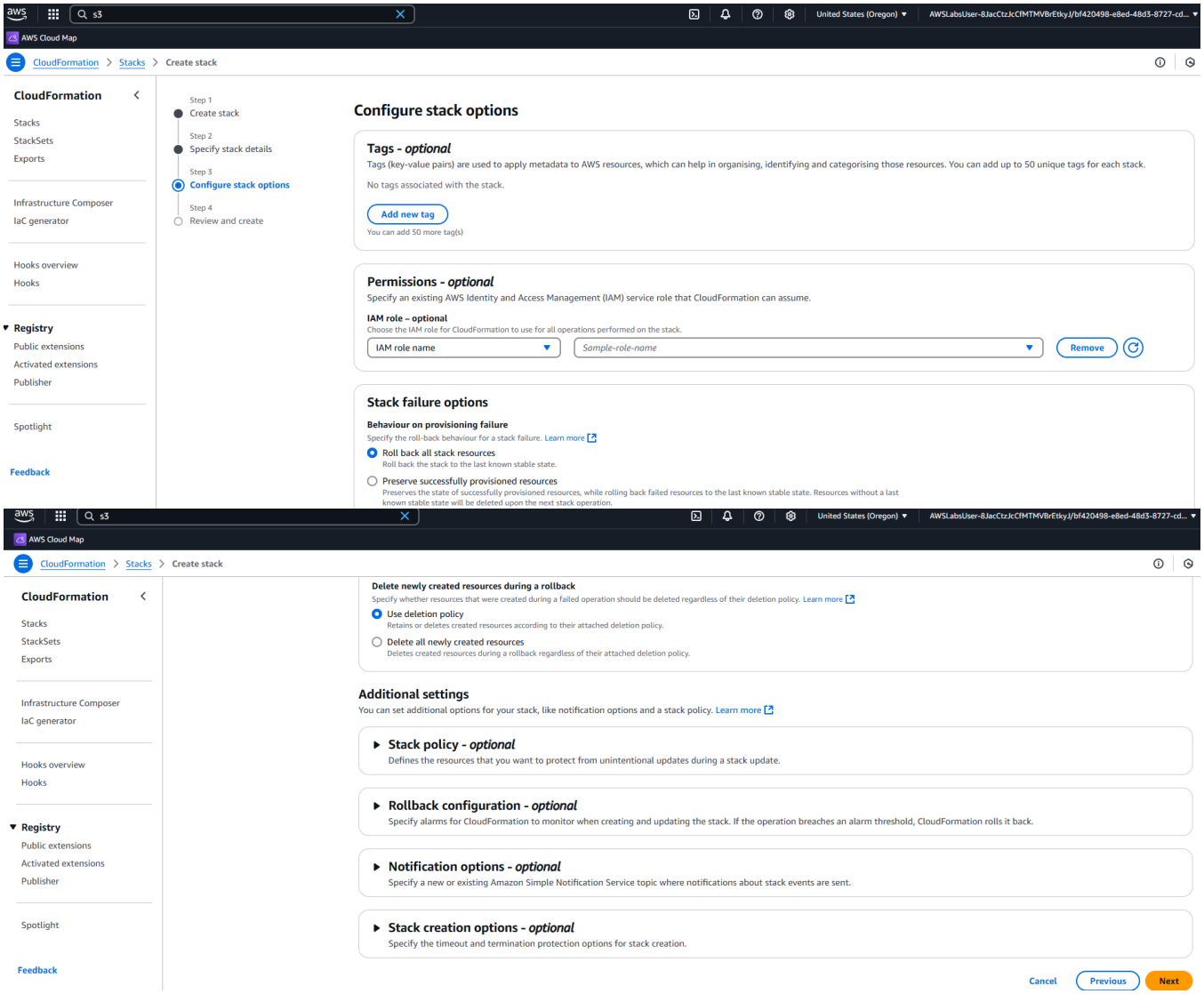


- **Stack name**: Enter `VPCStack`.
- **Parameters**: Keep the default values.



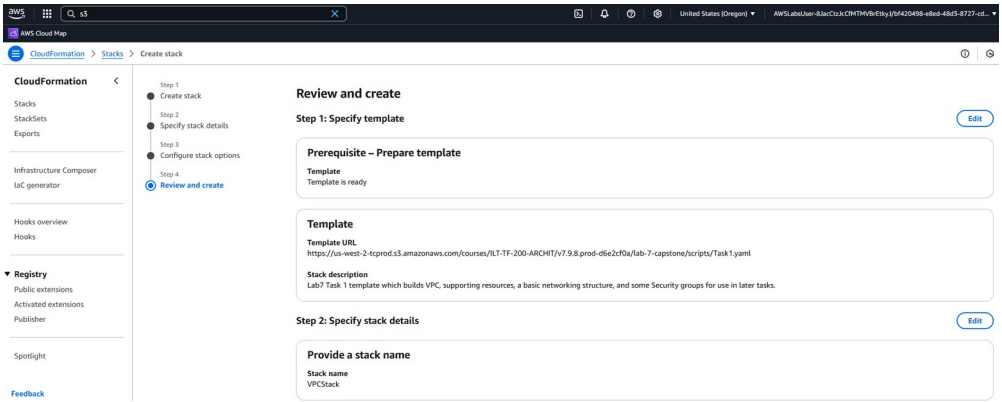
- Choose **Next**.

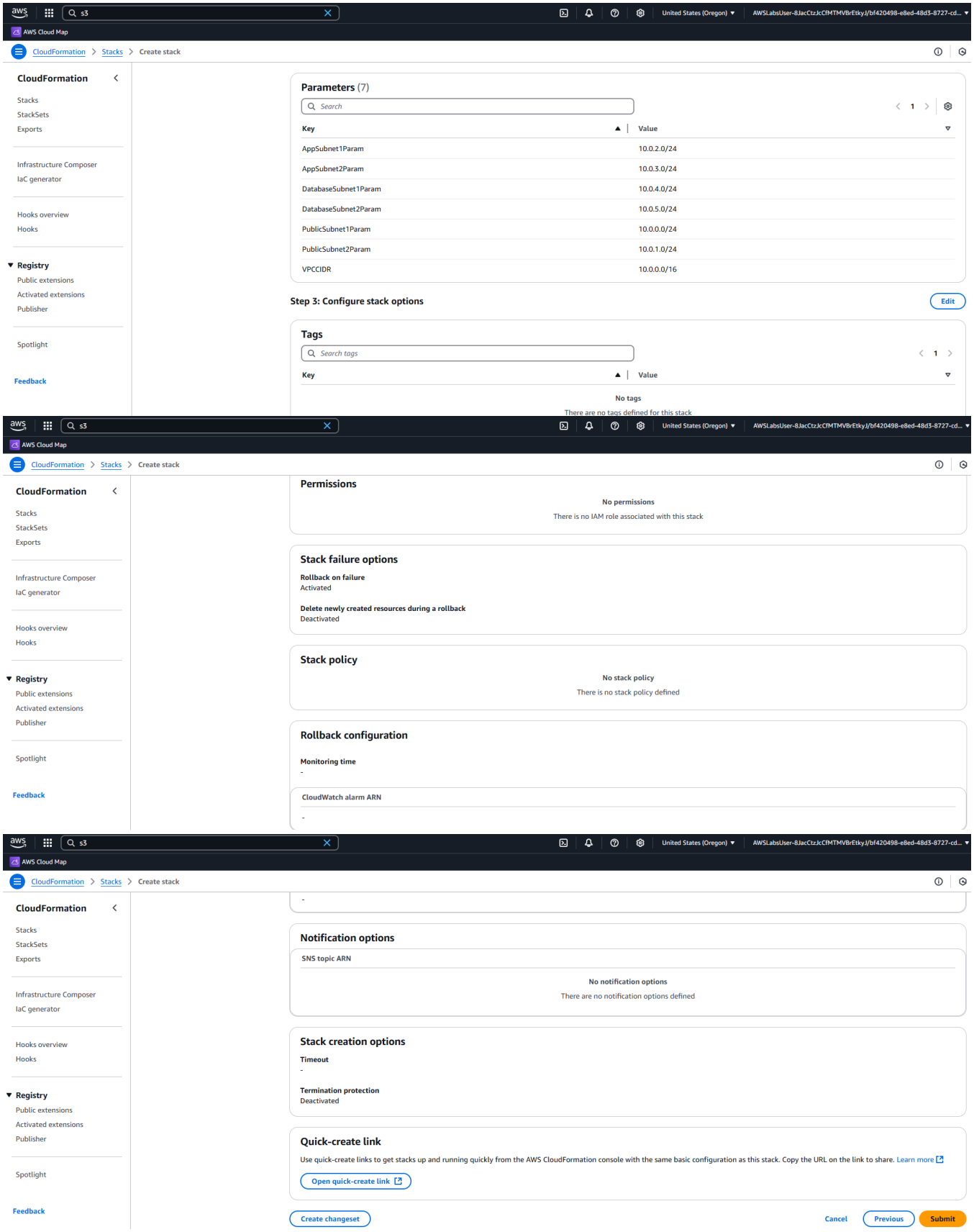
The **Configure stack options** page is displayed. You can use this page to specify additional parameters. You can browse the page, but leave settings at the default values.



- Choose **Next**.

The **Review and create** page is displayed. This page is a summary of all settings.

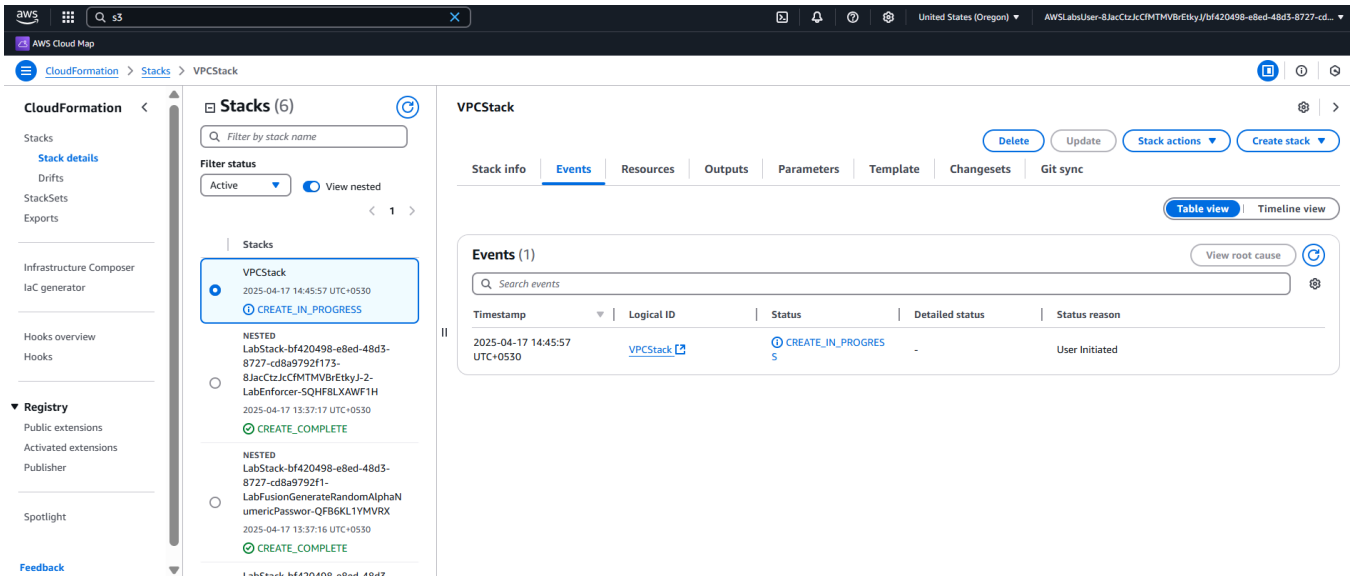




- At the bottom of the page, choose **Submit**.

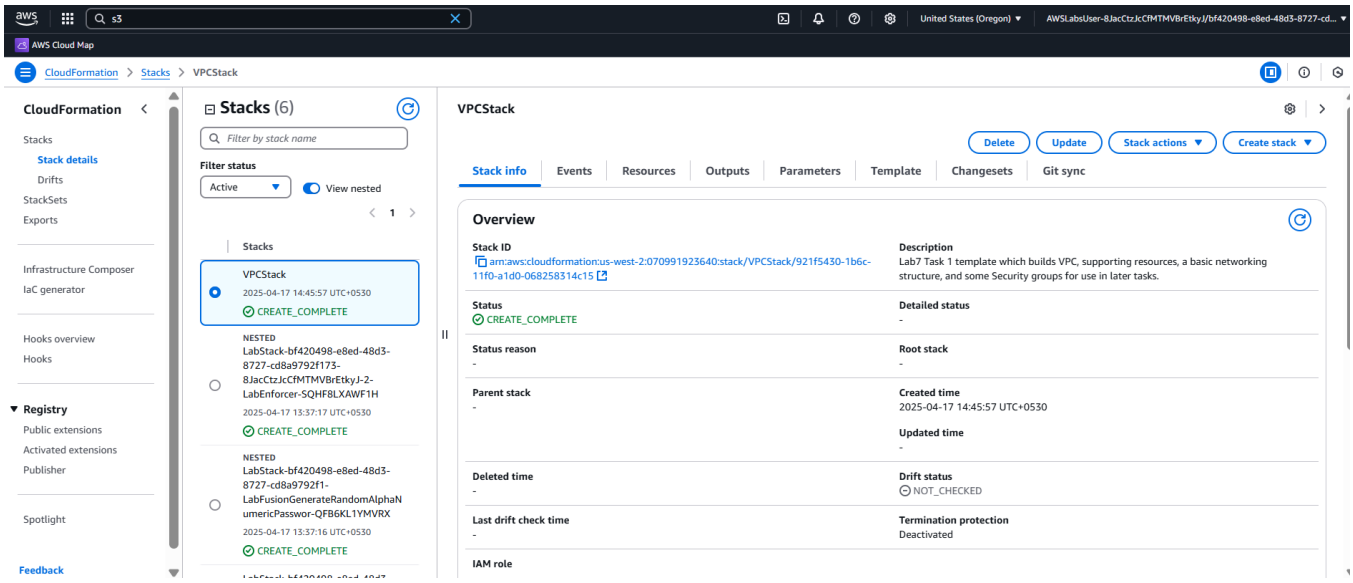
The **stack details** page is displayed.

The stack enters the CREATE_IN_PROGRESS status.



- Choose the **Stack info** tab.
- Occasionally choose the **Overview** refresh .
- Wait for the status to change to CREATE_COMPLETE.

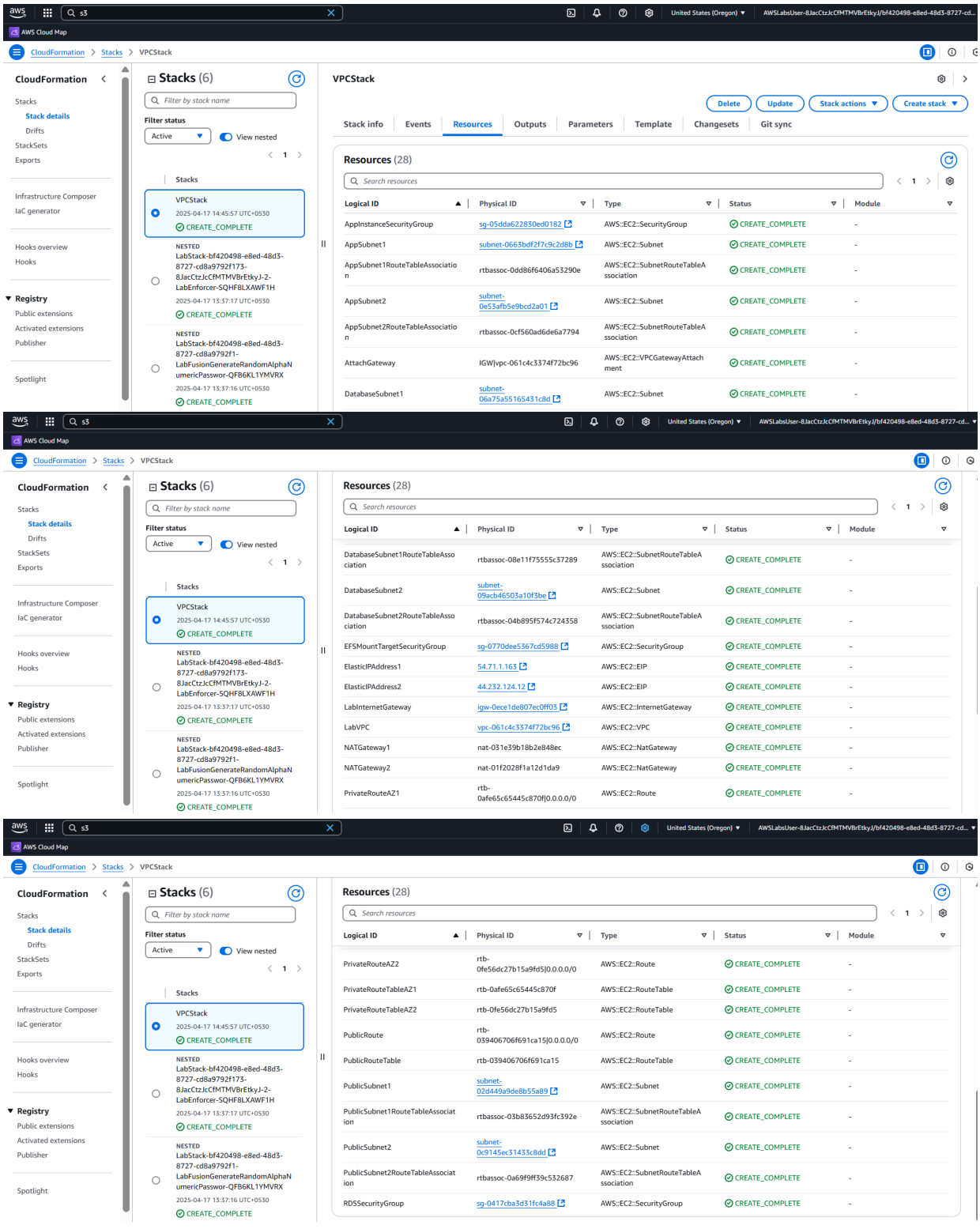
Note: This stack can take up to 5 minutes to deploy the resources.



Task 1.4: View created resources from the console

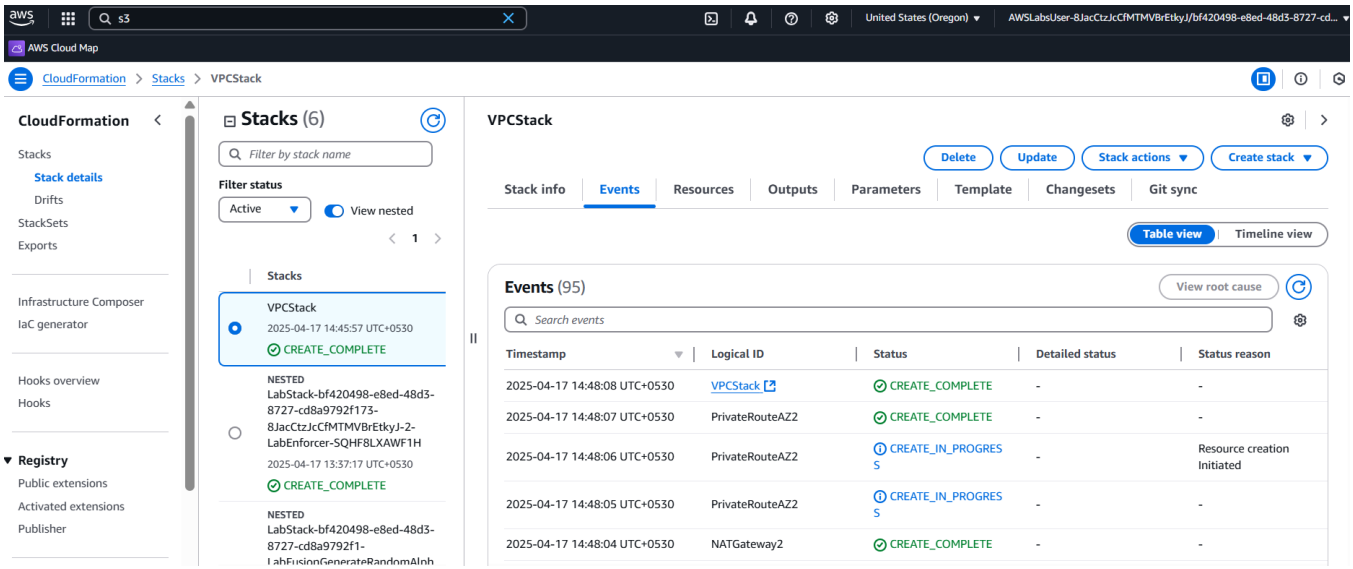
- Choose the **Resources** tab.

The list shows the resources that are being created. CloudFormation determines the optimal order for resources to be created, such as creating the VPC before the subnet.



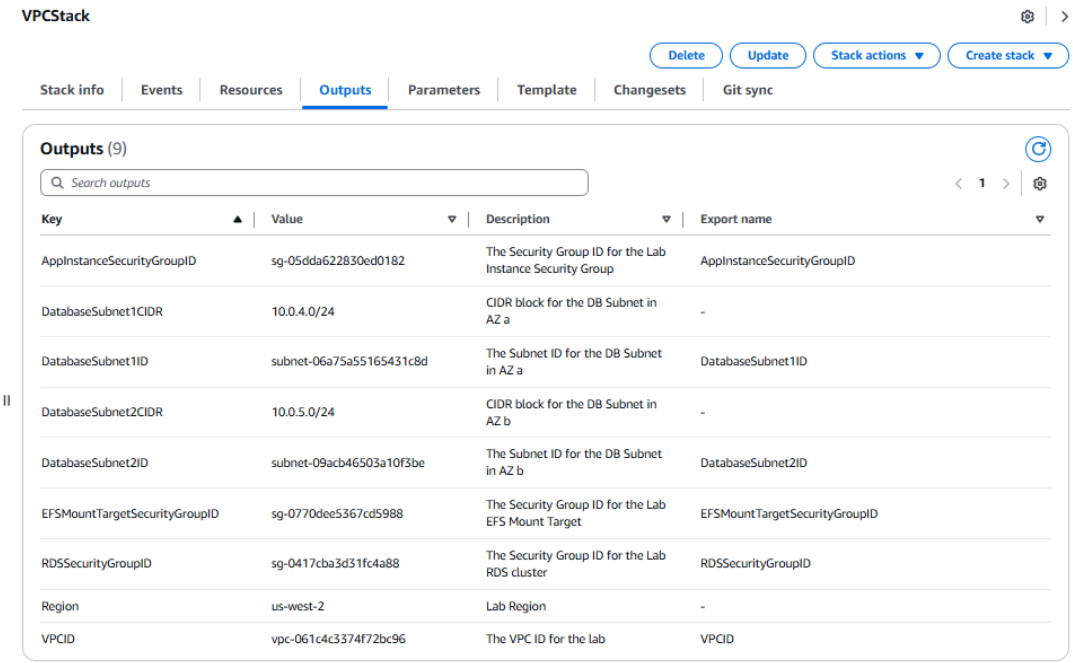
- Review the resources that were deployed in the stack.
- Choose the **Events** tab and scroll through the list.

The list shows (in reverse order) the activities performed by CloudFormation, such as starting to create a resource and then completing the resource creation. Any errors encountered during the creation of the stack are listed in this tab.



- Choose the **Outputs** tab.

Review the key-value pairs in the Outputs section. These values might be useful in later lab tasks.



You have learned to configure the stack and created all of the resources using the provided CloudFormation template.

Task 2: Create an Amazon RDS database

In this task, you create an Amazon Aurora DB instance and the subnet group to support it.

Task 2: Scenario

The network team verified the previous deployment requirements and everything looks good. The next phase of the project is to establish a secure backend database to give the database administrator (DBA) access for any migration operations before go-live.

The current database requires a lot of administrative overhead and the business has agreed to move to a managed database service. They want their architecture to be highly available, so you recommend that they set up a Multi-AZ RDS Aurora database layer. After reviewing the proposed design, the DBA has outlined the database requirements. Because of previous performance issues, the database does not require encryption, and you agree that this is the best choice.

The existing monitoring solution polls data every 10 minutes. The engineering team doesn't have room in the budget for additional features; therefore, enhanced monitoring is not required.

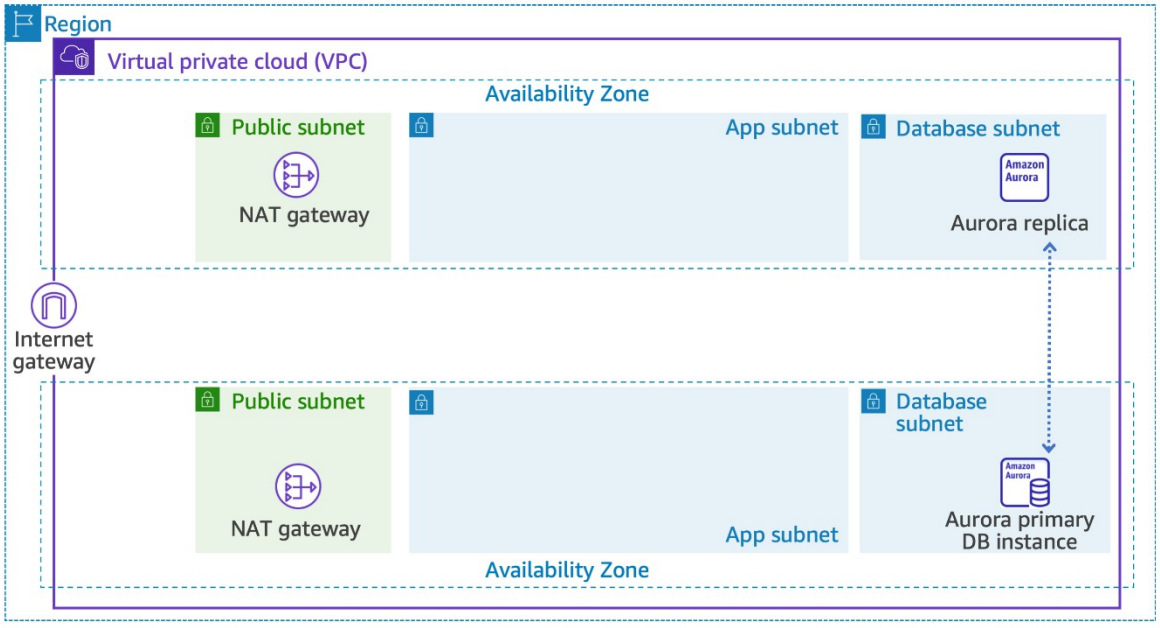


Image description: Preceding image depicts the entire architecture which will be created by the end of this lab. We have Amazon VPC to facilitate public and private networking using Public Subnets and Private Subnets. To ensure we are able to accept incoming internet traffic and allow outgoing traffic to the internet, we have Internet Gateway and NAT Gateways. For database we are using Amazon Aurora

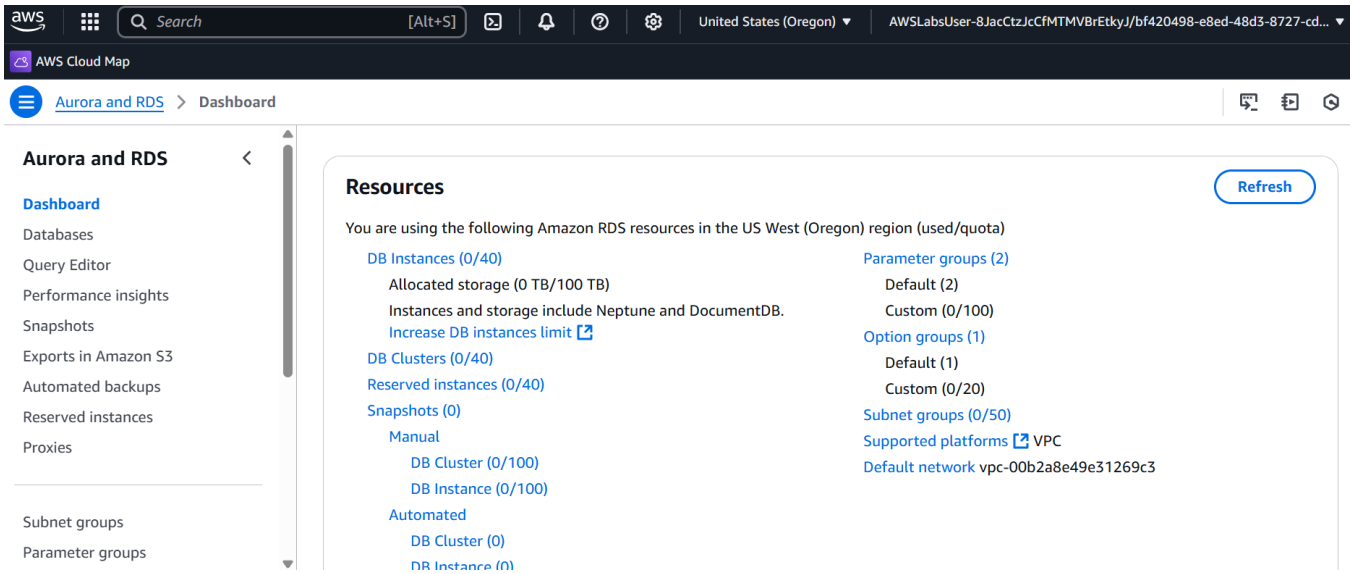
which works across multiple availability zones (AZ)s with primary in one availability zone (AZ) and a read-replica in another availability zone (AZ).

Task 2 instructions: Create an Amazon RDS database

Task 2.1: Navigate to the Amazon RDS console

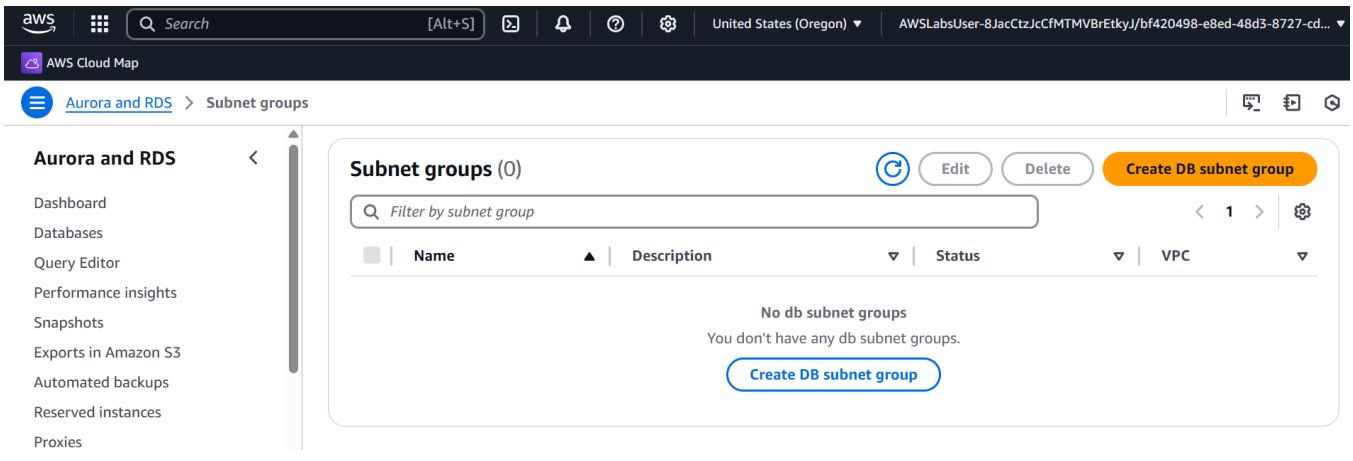
- At the top of the AWS Management Console, in the search box, search for and choose `RDS`.

The **Amazon RDS console** page is displayed.



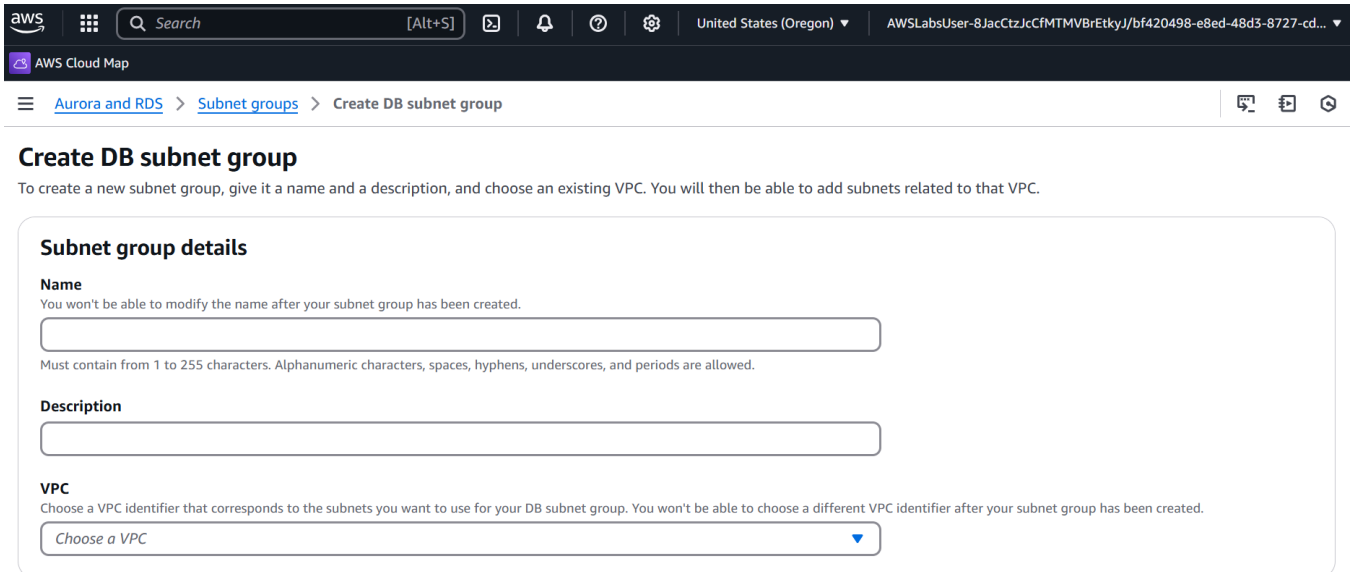
Task 2.2: Create a new DB subnet group

- In the left navigation pane, choose **Subnet groups**.

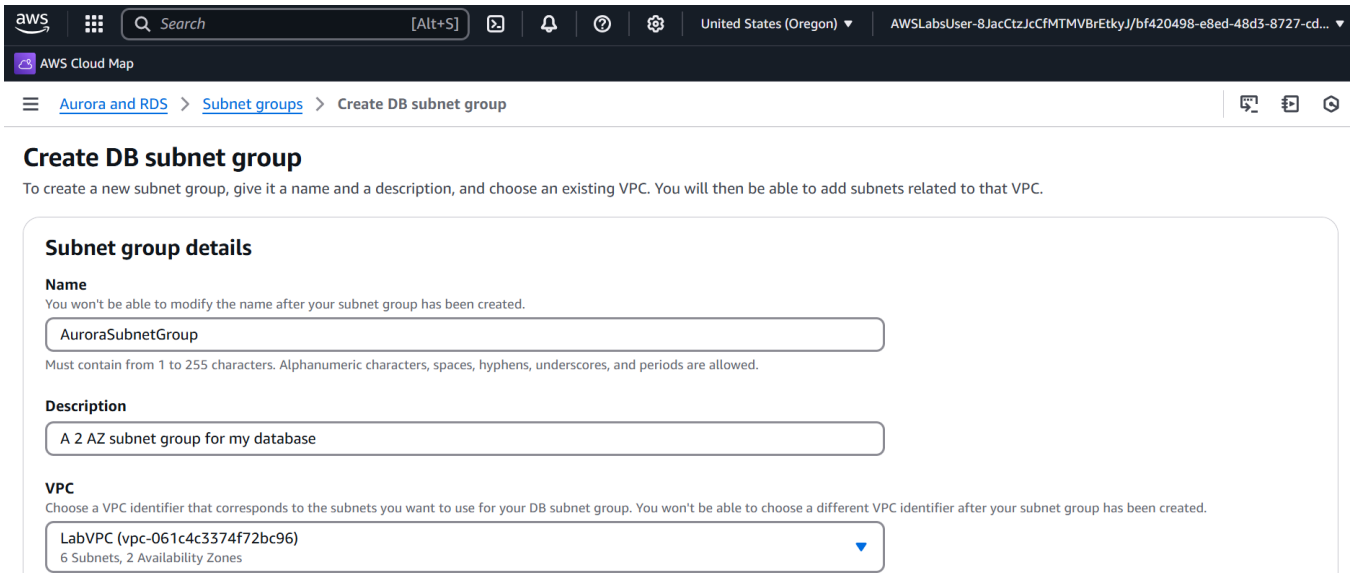


- Choose **Create DB subnet group**.

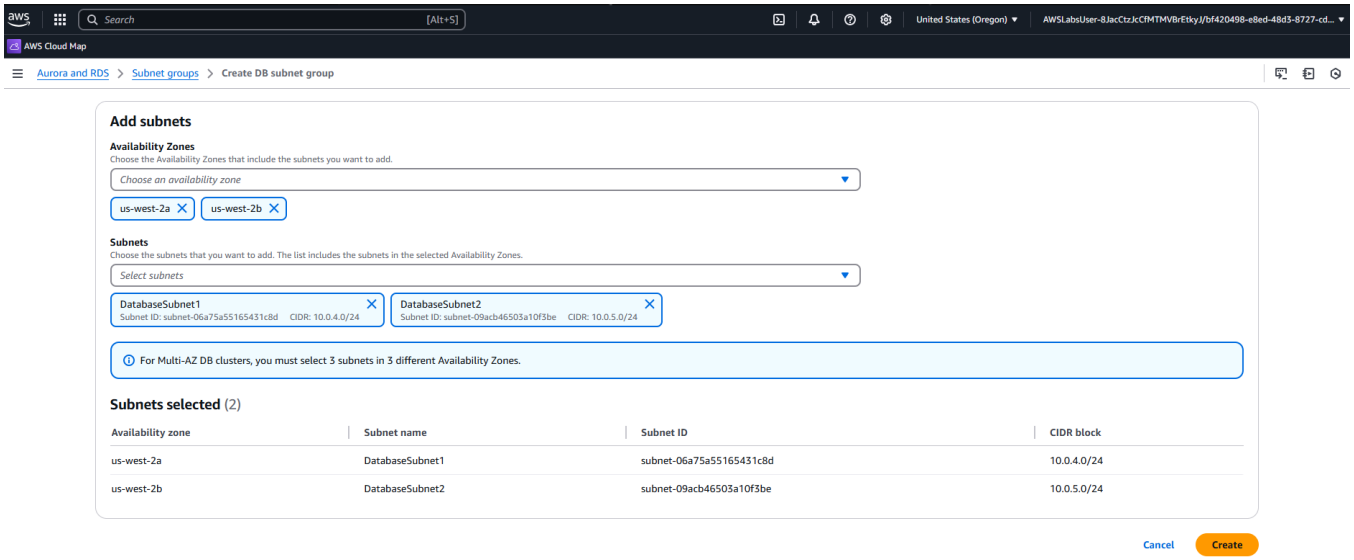
The **Create DB subnet group** page is displayed.



- In the **Subnet group details** section, configure the following:
  - **Name**: Enter `AuroraSubnetGroup`.
  - **Description**: Enter `A 2 AZ subnet group for my database`.
  - **VPC**: Select **LabVPC** from the dropdown menu.

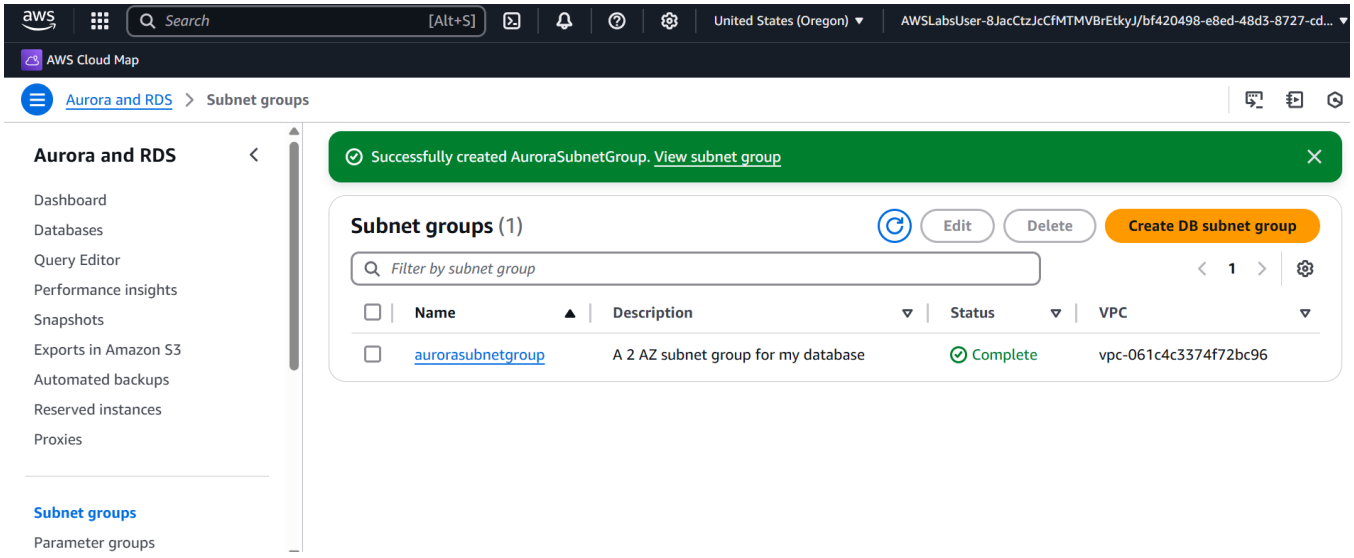


- In the **Add subnets** section, configure the following:
  - From the **Availability Zones** dropdown menu:
    - Select the Availability Zone ending in **a**.
    - Select the Availability Zone ending in **b**.
  - From the **Subnets** dropdown menu:
    - Select the subnet with the CIDR block 10.0.4.0/24 from the Availability Zone ending in **a**.
    - Select the subnet with the CIDR block 10.0.5.0/24 from the Availability Zone ending in **b**.



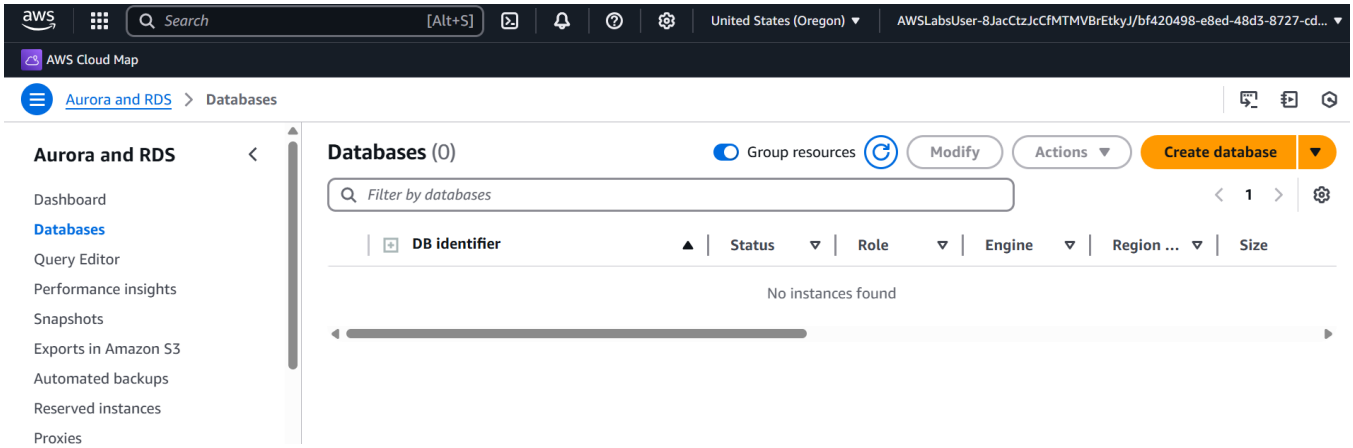
- Choose **Create**.

A Successfully created AuroraSubnetGroup. View subnet group message is displayed on top of the screen.



Task 2.3: Create a new Amazon Aurora database

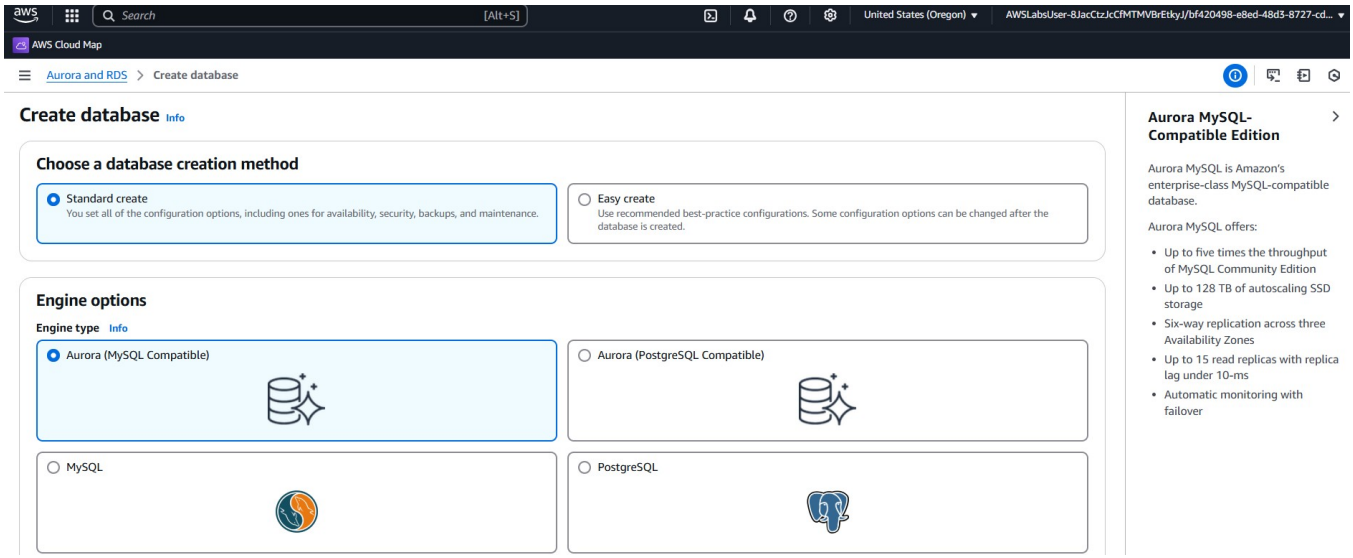
- In the left navigation pane, choose **Databases**.



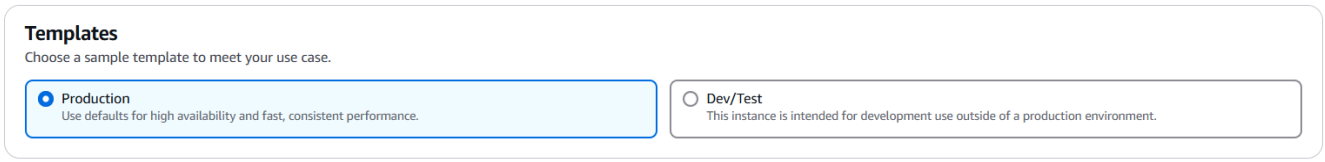
- Choose **Create database**.

The **Create database** page is displayed.

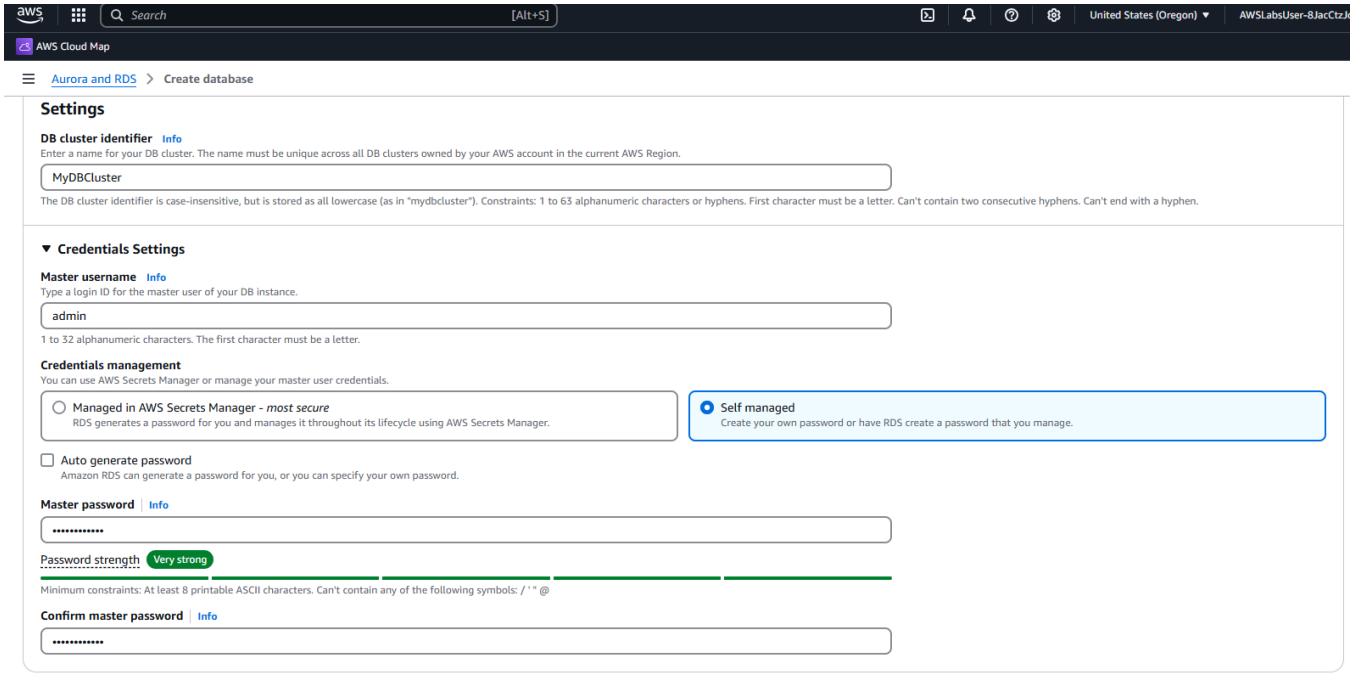
- In the **Choose a database creation method** section, select **Standard create**.
- In the **Engine options** section, configure the following:
  - In **Engine type**, select **Aurora(MySQL Compatible)**.



- In the **Templates** section, select **Production**.

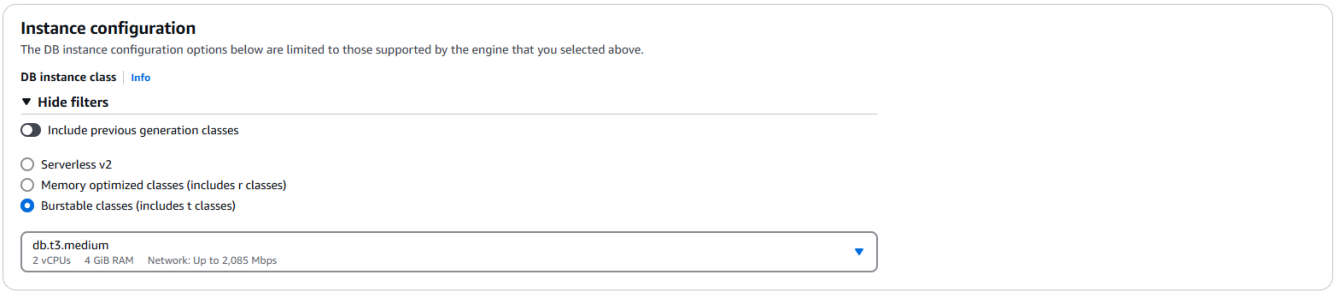


- In the **Settings** section, configure the following:
    - **DB cluster identifier**: Enter `MyDBCluster`.
    - **Master username**: Enter `admin`.
    - **Credentials management:** Select **Self managed**
    - **Master password**: Paste the **LabPassword** value(1QaDM4T5Cxny)
    - **Confirm master password**: Paste the **LabPassword** (1QaDM4T5Cxny).

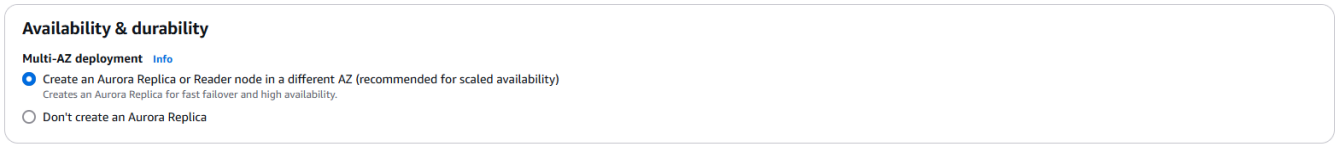


Note: Remember your credentials.

- In the **Instance configuration** section, configure the following:
    - **DB instance class**: Select **Burstable classes (includes t classes)**.
    - From the **instance type** dropdown menu, choose **db.t3.medium**.

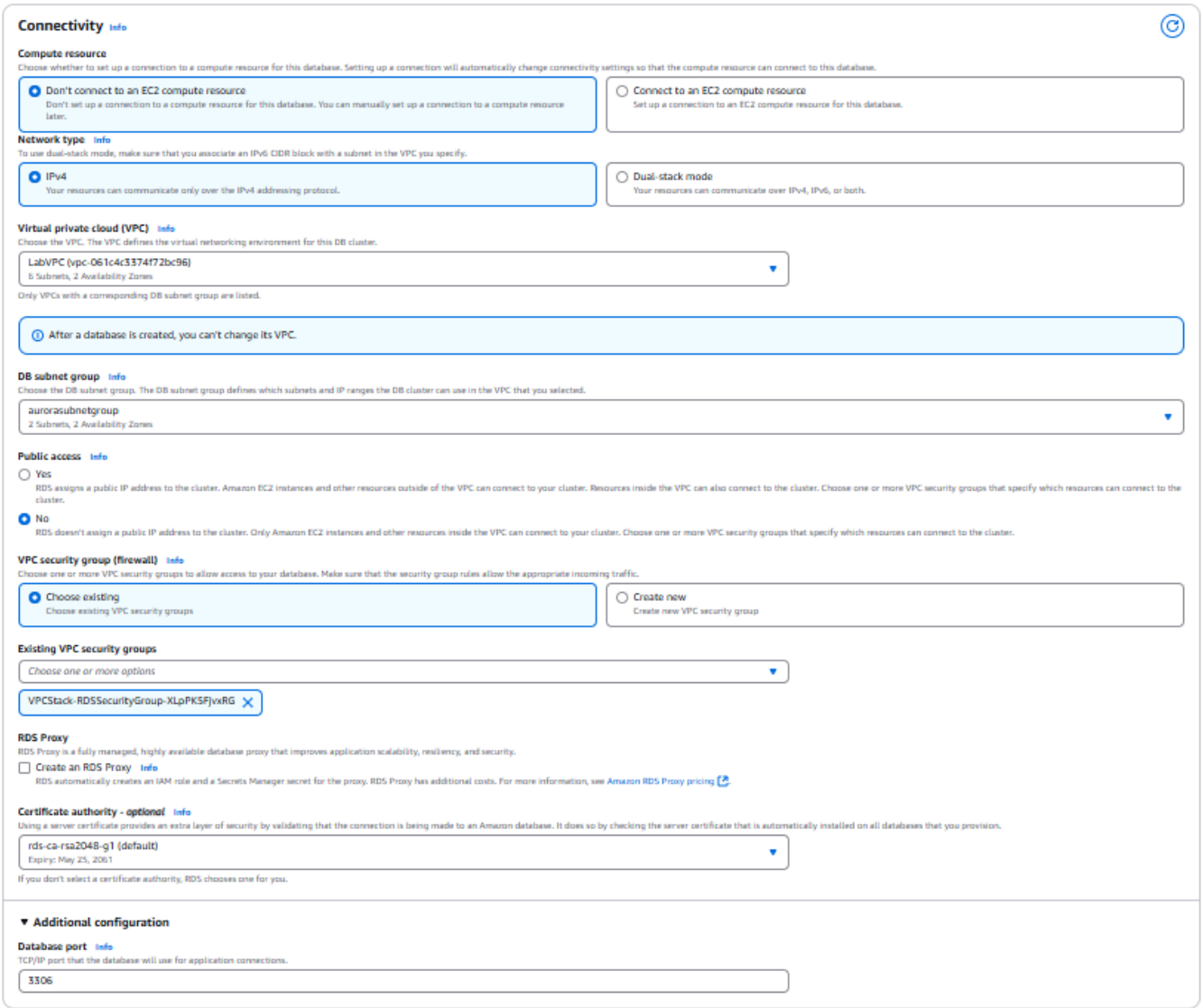


- In the **Availability & durability** section, for **Multi-AZ deployment**, select **Create an Aurora Replica or Reader node in a different AZ (recommended for scaled availability)**.

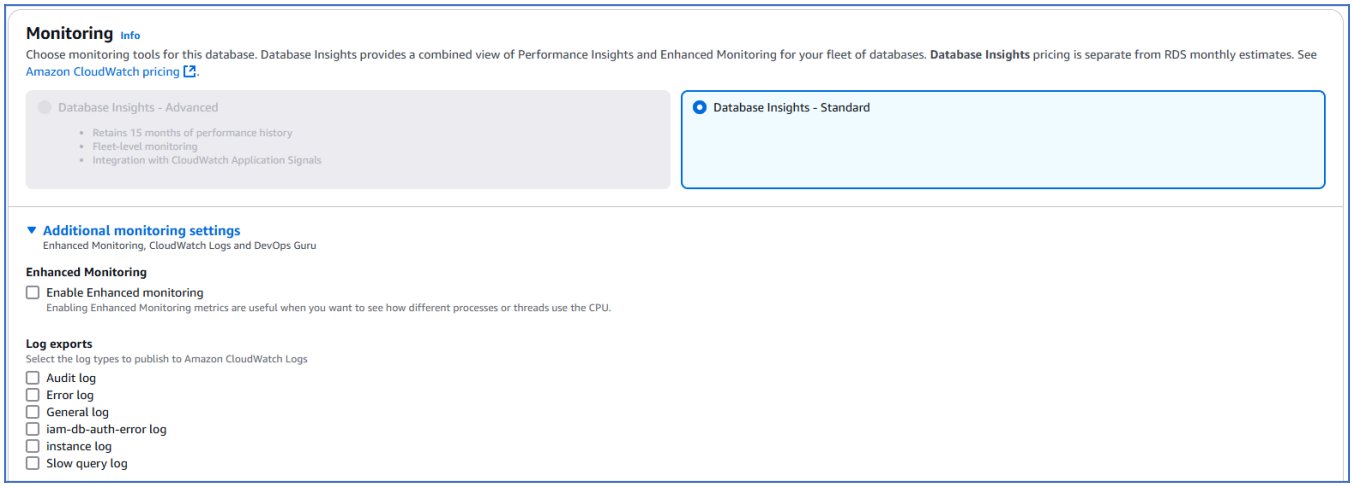


- In the **Connectivity** section, configure the following:

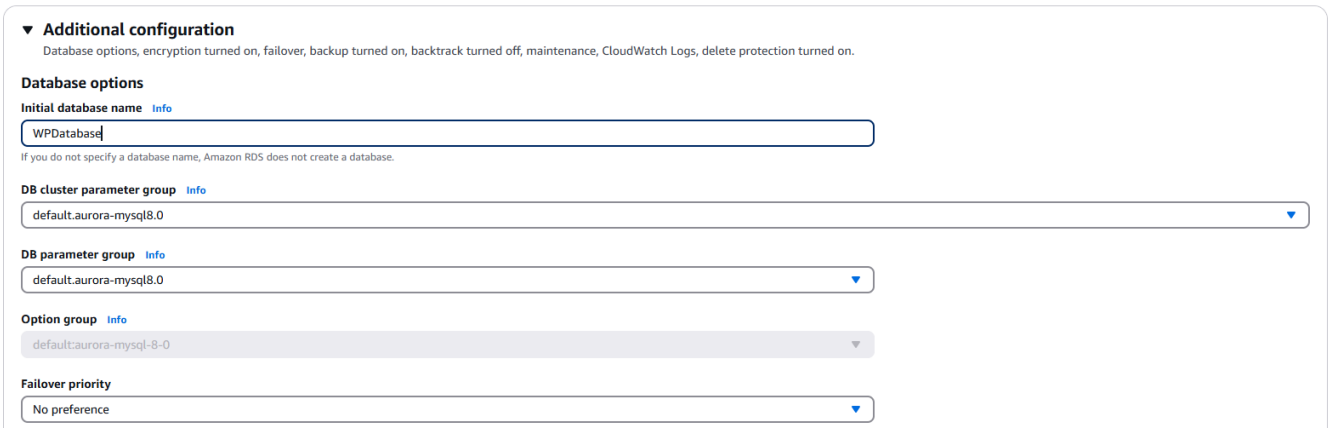
- **Virtual private cloud (VPC)**: Select **LabVPC** from the dropdown menu.
- **DB subnet group**: Select **aurorasubnetgroup** from the dropdown menu.
- **Public access**: Select **No**.
- **VPC security group (firewall)**: Select **Choose existing**.
- **Existing VPC security groups**:
  - Select only the **xxxxx-RDSSecurityGroup-xxxxx** group from the dropdown menu.
  - To remove the default security group, choose the **X**.
- Expand the **Additional configuration** section and configure the following:
  - **Database port**: Leave the configuration at the default value.



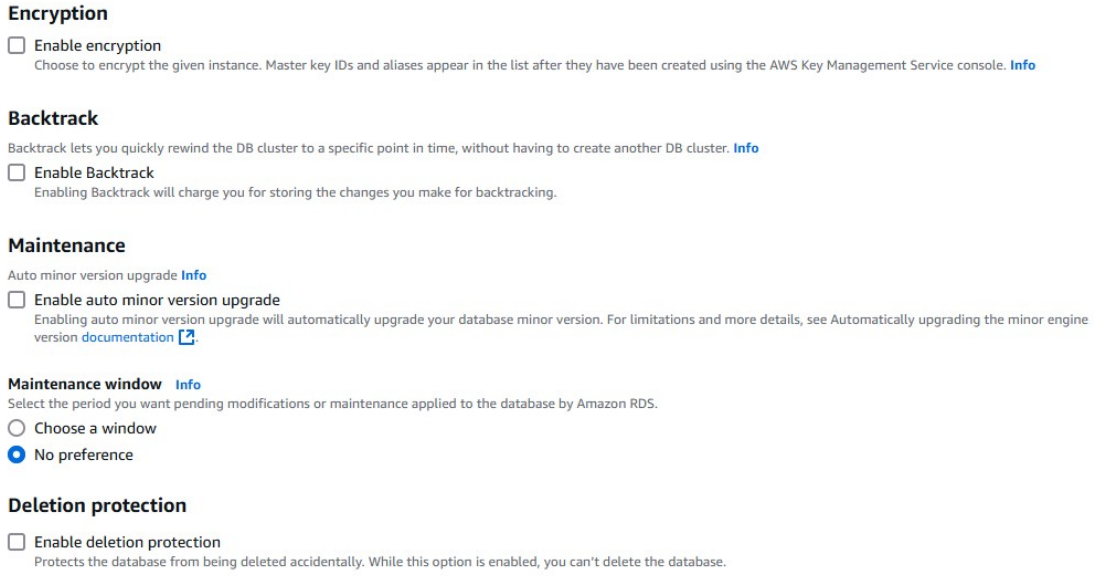
- In the **Monitoring** section, clear **Enable Enhanced monitoring**.



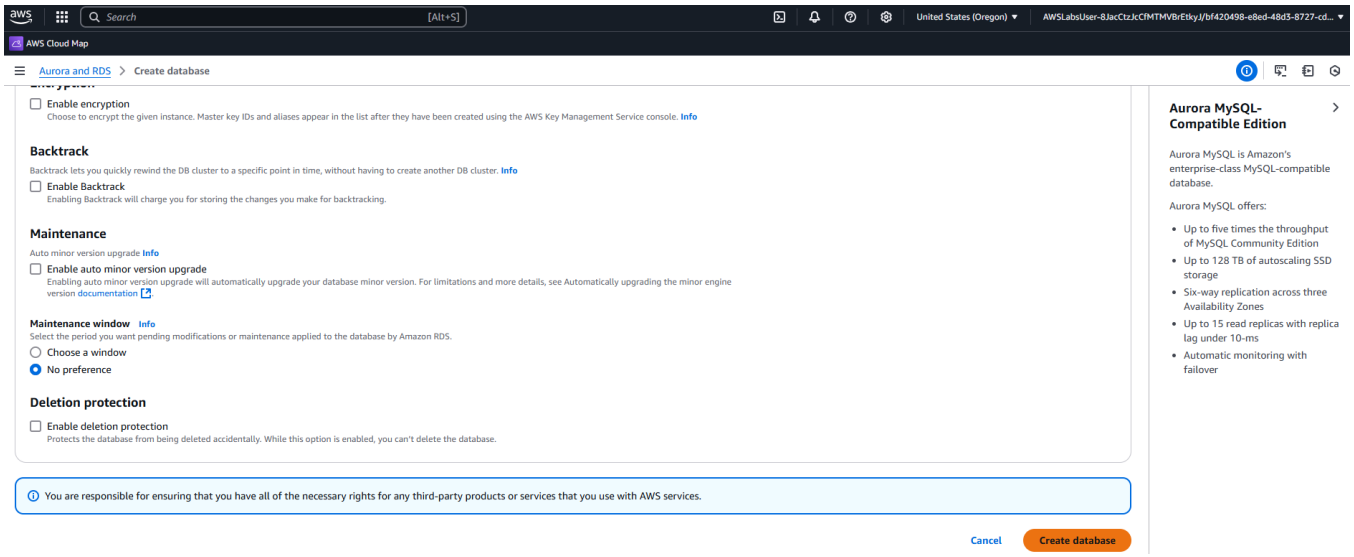
- Scroll to the bottom of the page and expand the main **Additional configuration** section.
    - In the **Database options** section, configure the following:
        - **Initial database name**: Enter `WPDatabase`.



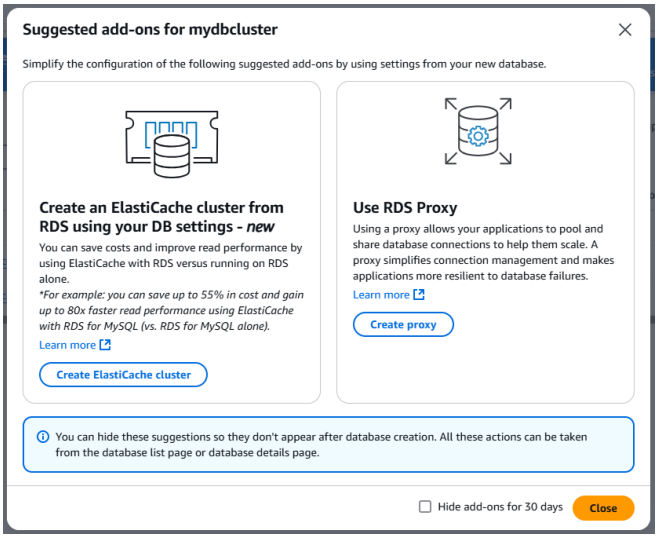
    - In the **Encryption** section, clear **Enable encryption**.
    - In the **Maintenance** section, clear **Enable auto minor version upgrade**.
    - In the **Deletion protection** section, clear **Enable deletion protection**.



- Scroll to the bottom of the screen and choose **Create database**.

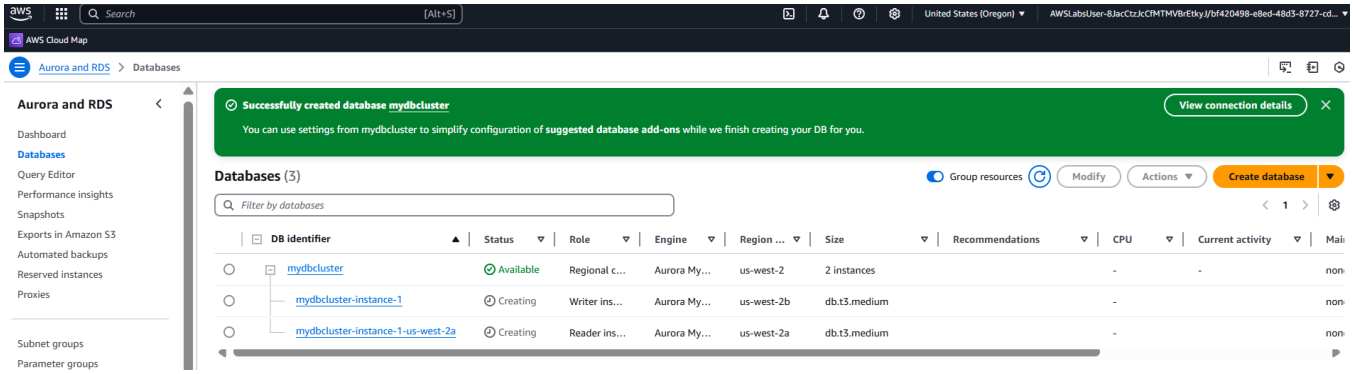


- On the **Suggested add-ons for mydbcluster** pop-up window, choose **Close**.

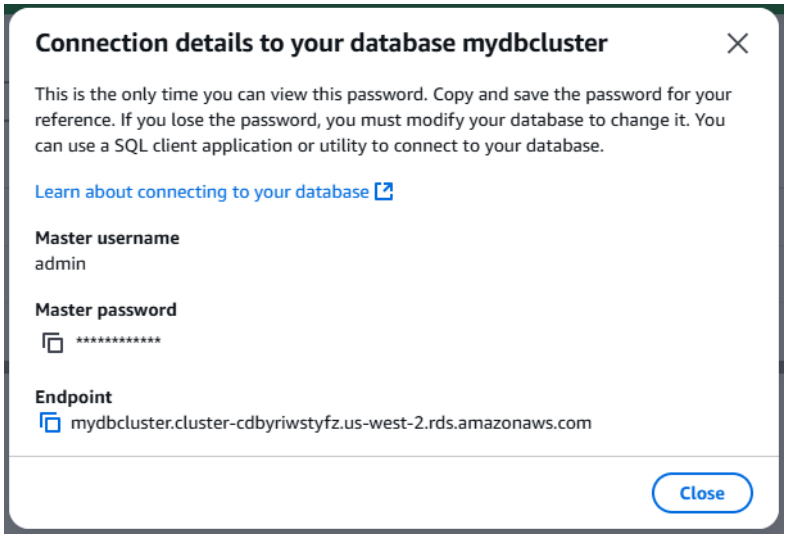


Note: Your Aurora MySQL DB cluster is in the process of launching. The cluster you configured consists of two instances, each in a different Availability Zone. The Amazon Aurora DB cluster can take up to 5 minutes to launch. Wait for the mydbcluster status to change to *Available*. You do not have to wait for the availability of the instances to continue.

A Successfully created database mydbcluster message is displayed on top of the screen.



- Choose **View connection details** displayed on the success message border to save the connection details of your **mydbcluster** database to a text editor. Once the connection details are saved on a notepad, choose **Close**.

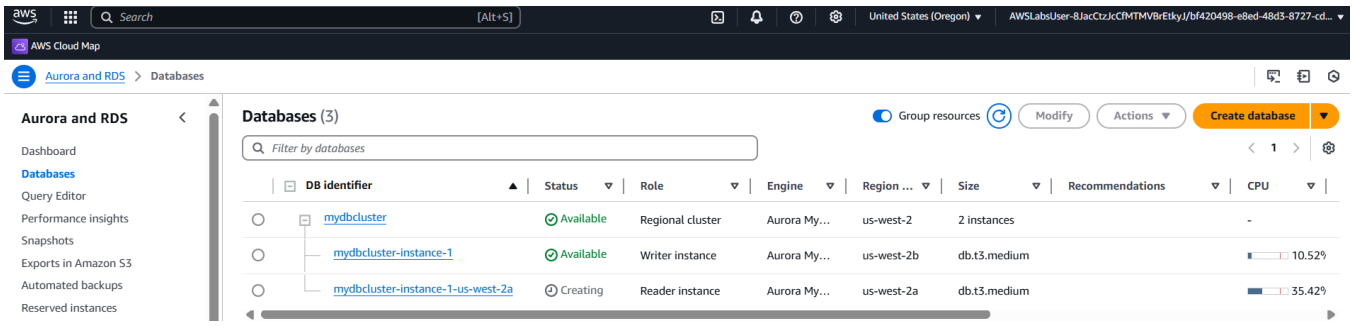


Much of this information can also be found in the next task.

Note If you notice the error "Failed to turn on enhanced monitoring for database mydbcluster because of missing permissions" you can safely ignore the error.

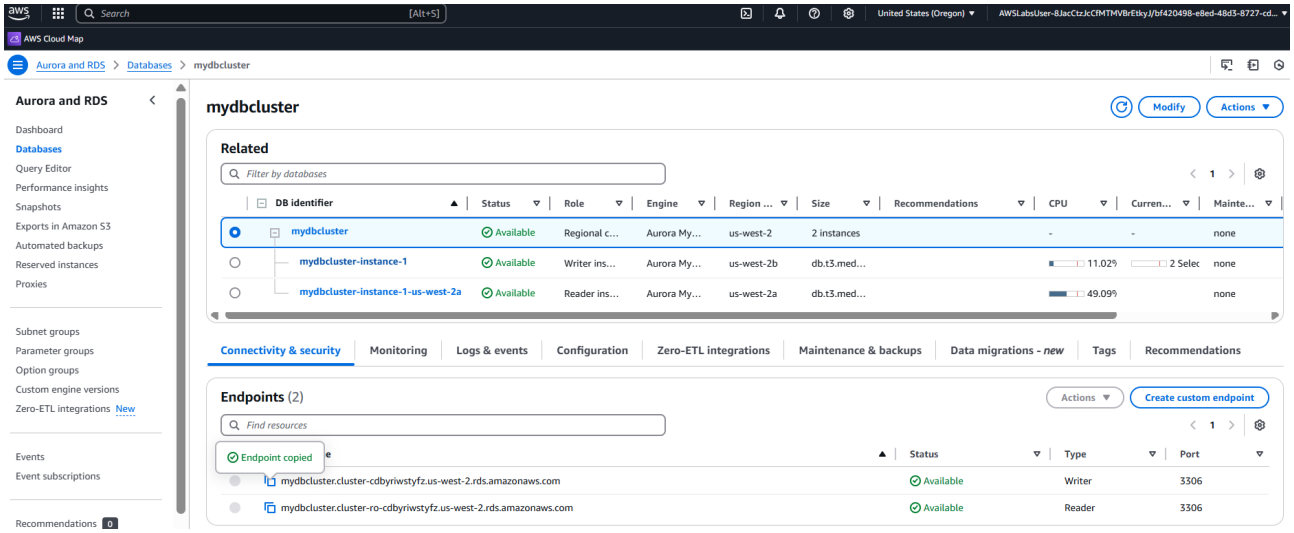
Task 2.4: Copy database metadata

- In the left navigation pane, choose **Databases**.

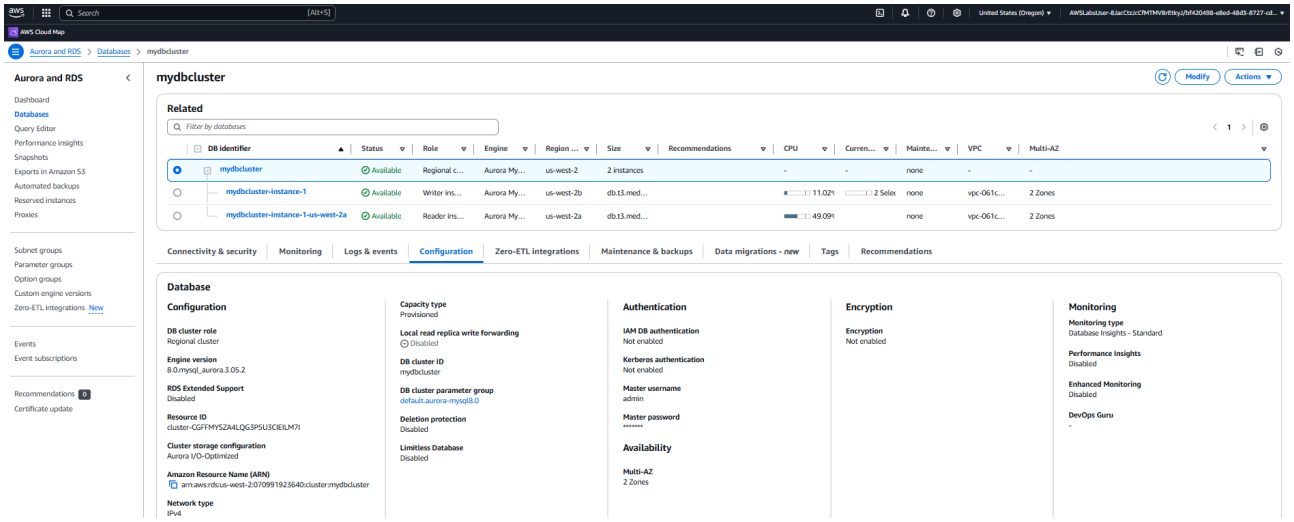


- Choose the **mydbcluster** link.

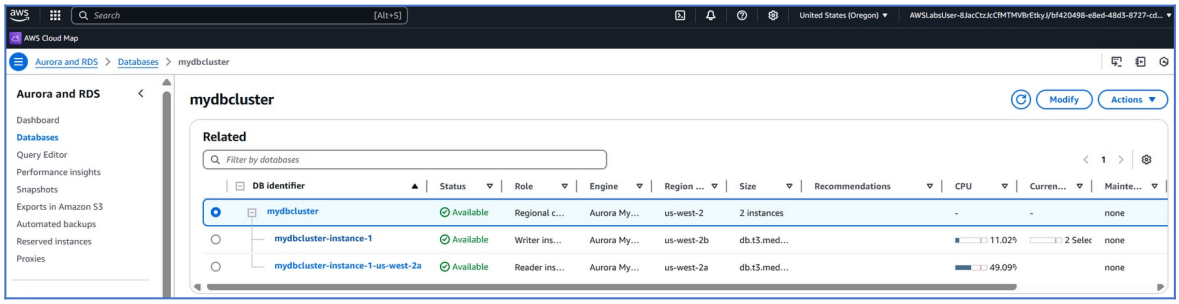
- Choose the **Connectivity & security** tab.
- Copy the **endpoint** value for the **Writer** instance to a text editor.



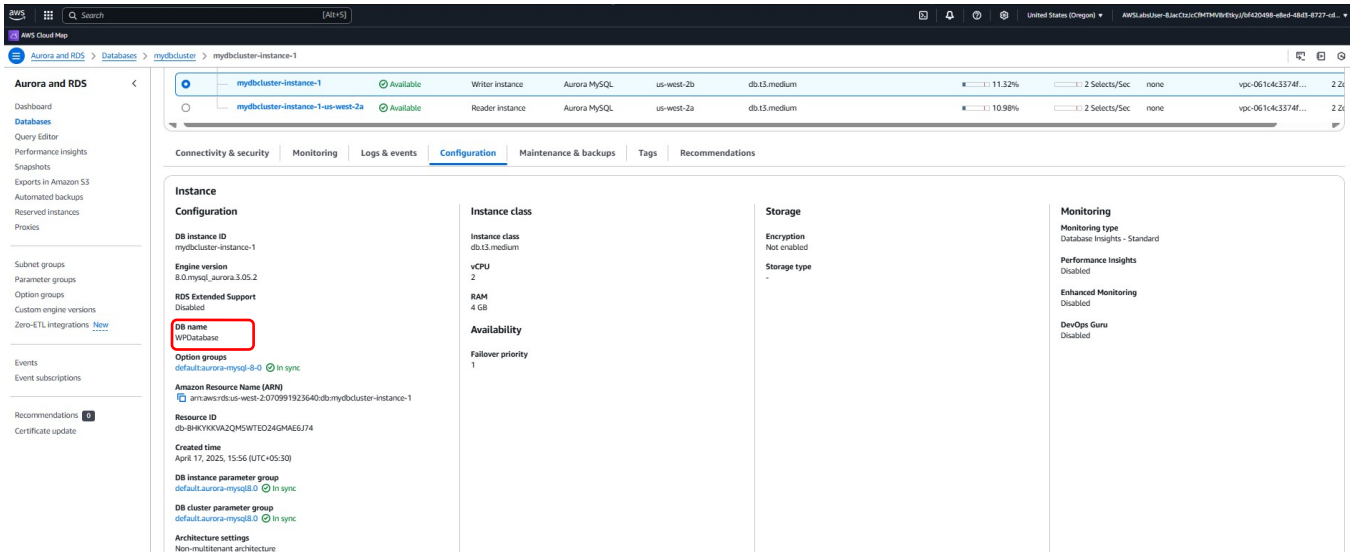
- Choose the **Configuration** tab.



- Copy the **Master username** value to a text editor.
- For **Master password**: Use the **LabPassword** value (1QaDM4T5Cxny).
- In the left navigation pane, choose **Databases**.



- Choose the **mydbcluster-instance-1** writer instance link.
- Choose the **Configuration** tab.



- Copy the **DB name** value to a text editor.

You have created the Amazon Aurora database.

Task 3: Create an Amazon EFS file system

In this task, you create and configure a custom Amazon EFS file system.

Task 3: Scenario

Example Corp. is having issues with the lead time on ordering new hardware. This is slowing down their ability to onboard new customers and expand the business. The SysOps team has a request for a storage solution built for the cloud. They need to be able to confirm that the backup policies and encryption settings meet their internal and regulatory compliance requirements. Managing time, cost, and compliance gives Example Corp. a competitive advantage.

You recommend Amazon EFS (Amazon Elastic File System (EFS) is a **fully managed, scalable, elastic, and serverless network file system** provided by Amazon Web Services (AWS). It's designed to be easy to use and provides shared file storage for Linux, macOS, and Windows-based workloads in the AWS Cloud and on-premises.) to the business as a simple, serverless, set-and-forget, elastic file system. With Amazon EFS, they can share data securely, without provisioning or managing storage.

Your task is to create an Amazon EFS file system that meets the SysOps team's requirements.

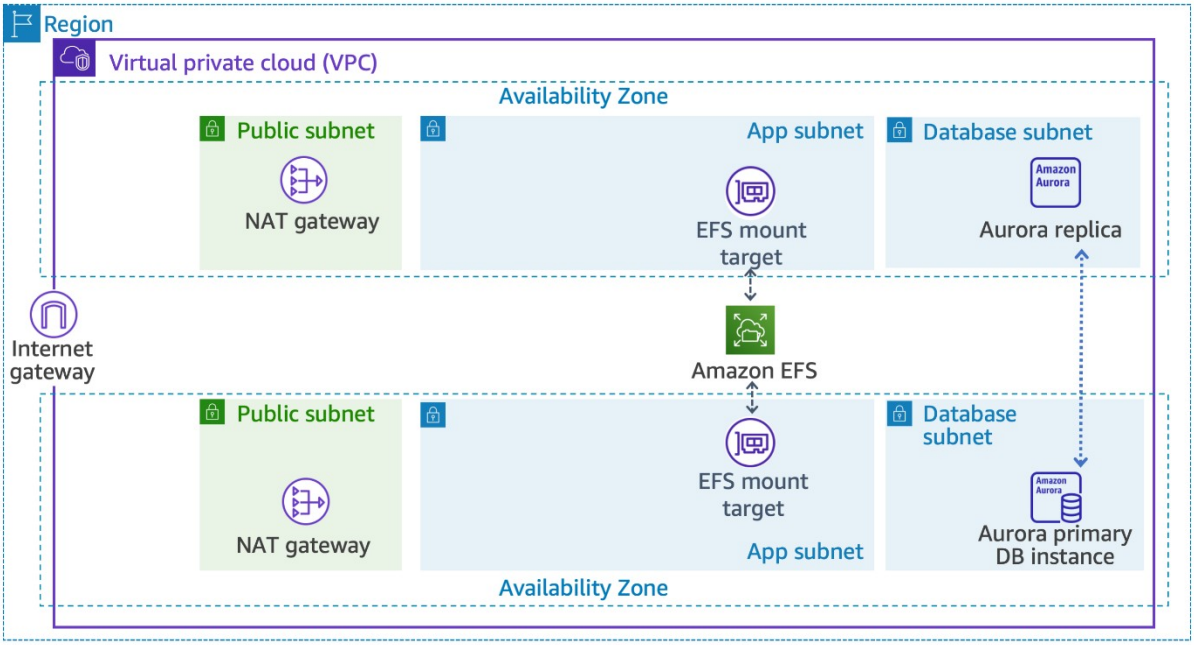


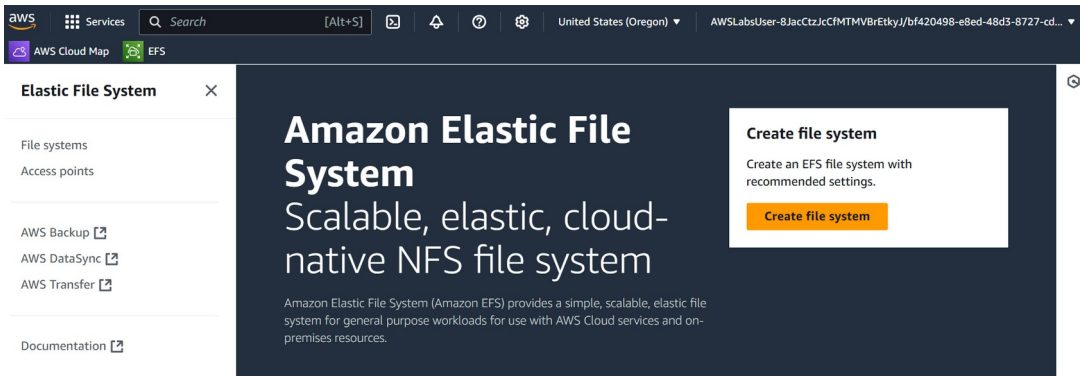
Image description: Preceding image depicts the entire architecture which will be created by the end of this lab. We have Amazon VPC to facilitate public and private networking using Public Subnets and Private Subnets. To ensure we are able to accept incoming internet traffic and allow outgoing traffic to the internet, we have Internet Gateway and NAT Gateways. There is also a fully scalable and elastic file server using Amazon EFS which can work across multiple availability zones (AZ)s. For database we are using Amazon Aurora which works across multiple availability zones (AZ)s with primary in one availability zone (AZ) and a read-replica in another availability zone (AZ).

Task 3 instructions: Creating an Amazon EFS file system

Task 3.1: Navigate to the Amazon EFS console

- At the top of the AWS Management Console, in the search box, search for and choose EFS.

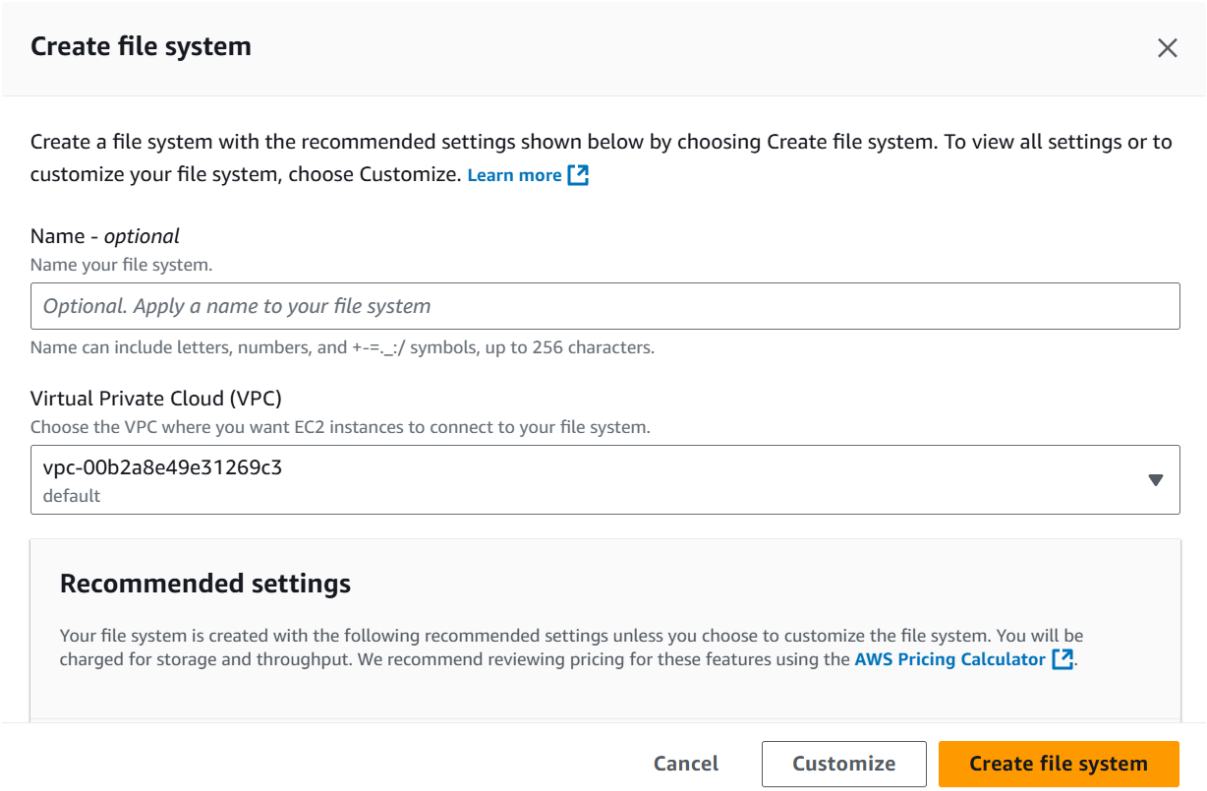
The **Amazon Elastic File System console** page is displayed.



Task 3.2: Create a new file system

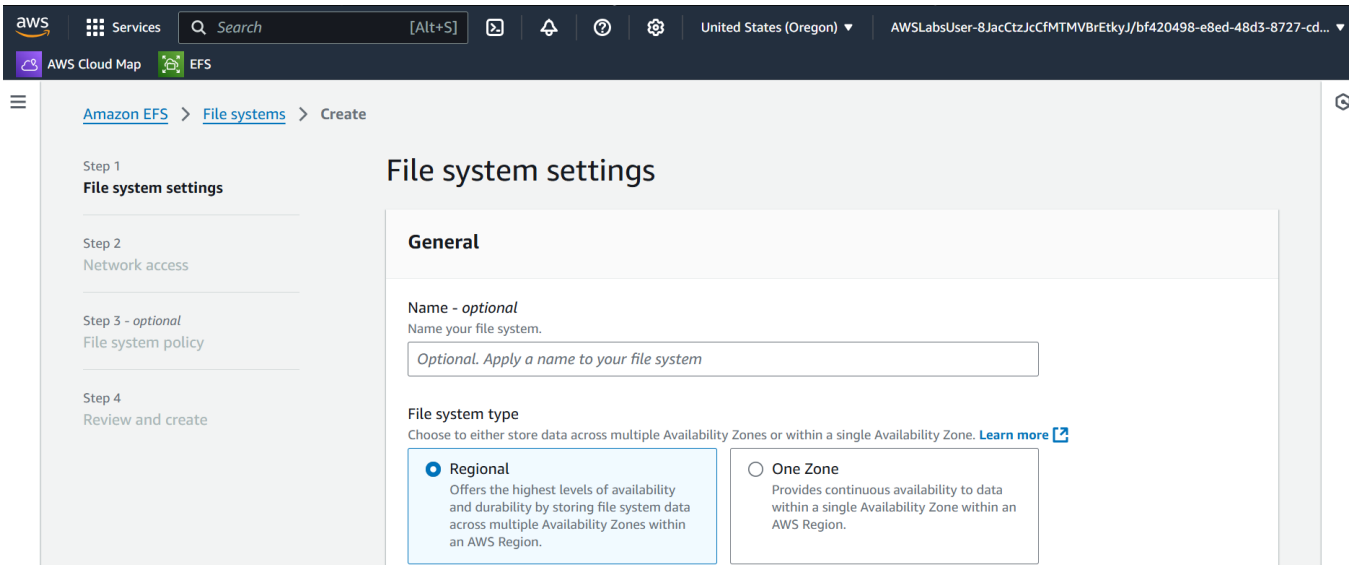
- Choose **Create file system**.

The **Create file system** page is displayed.



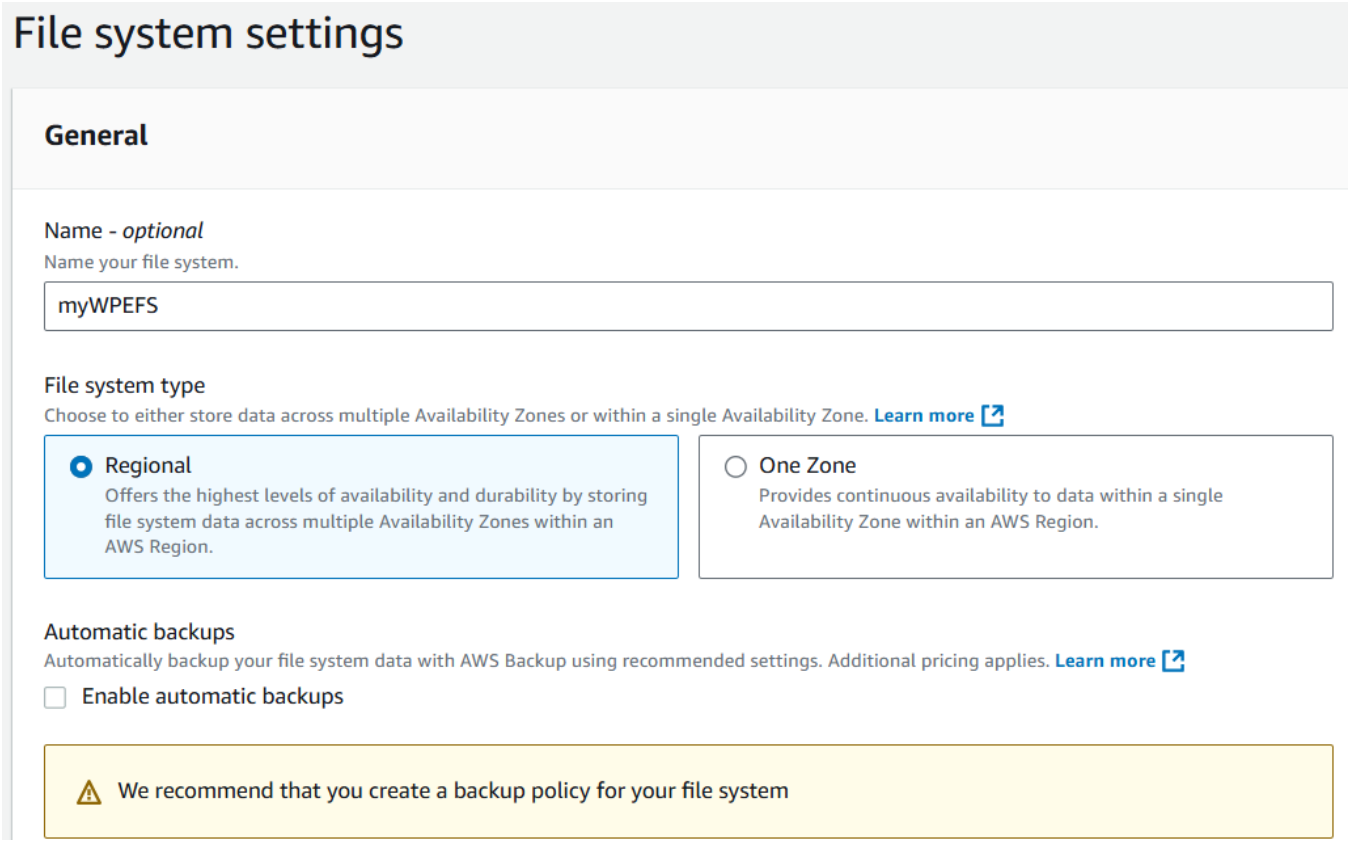
- Choose **Customize**.

The **File system settings** page is displayed.

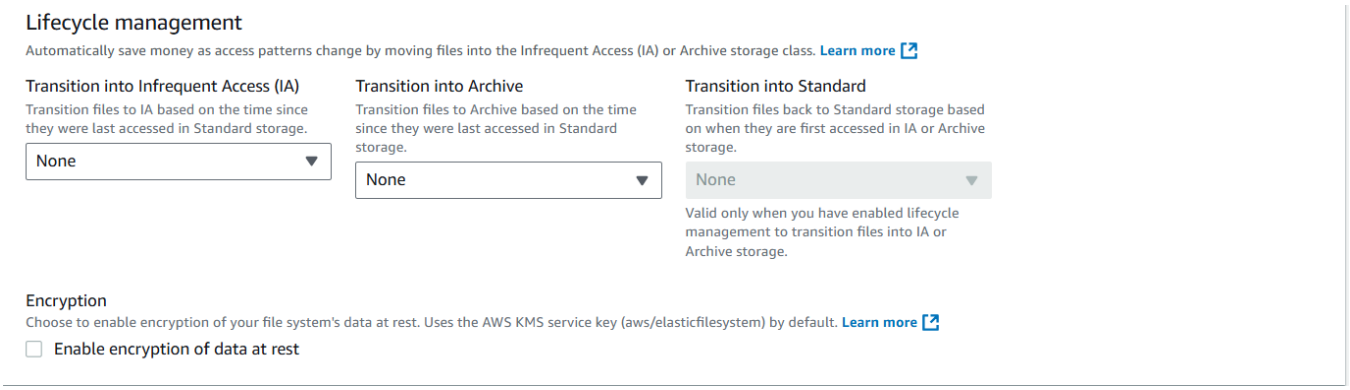


- In the **General** section, configure the following:

- **Name - *optional***: Enter `myWPEFS`.
- Clear **Enable automatic backups**.

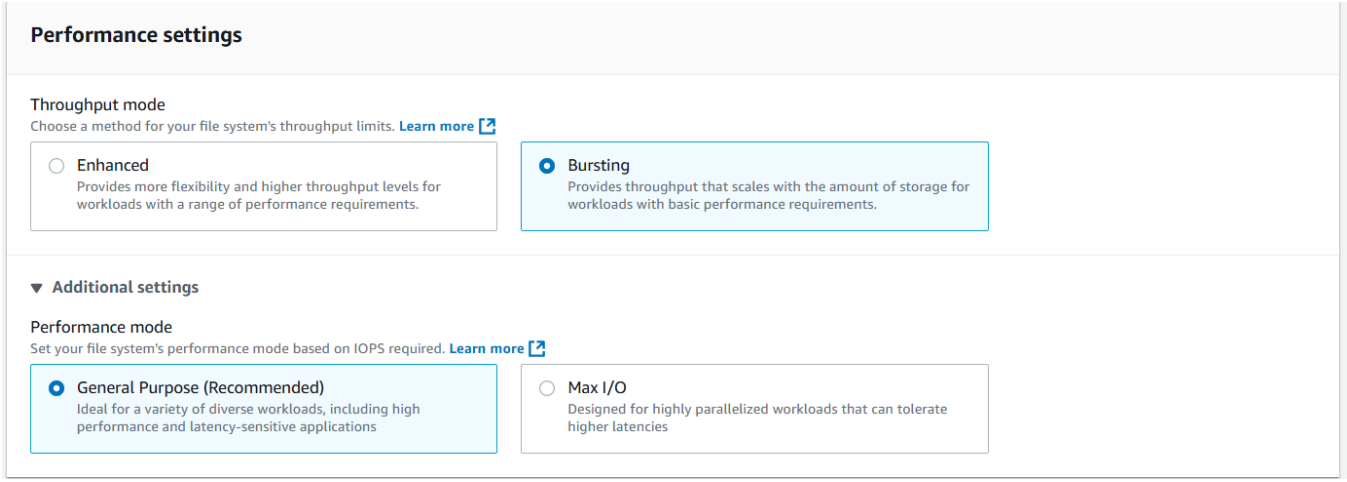


- Under **Lifecycle Management** set **Transition into Infrequent Access (IA)** as **None**
- Under **Lifecycle Management** set **Transition into Archive** as **None**
- Clear **Enable encryption of data at rest**.

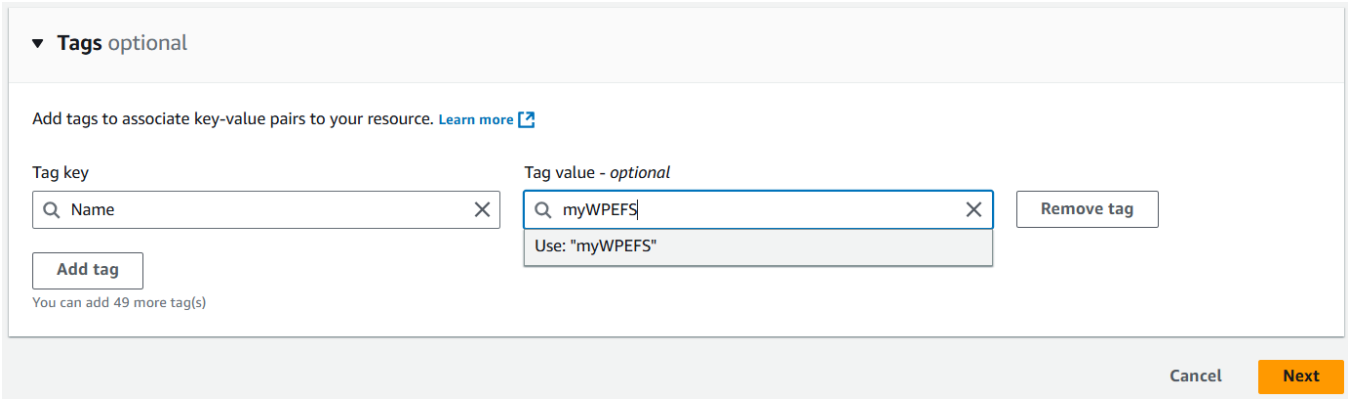


Performance Settings

- Set **Throughput mode** to **Bursting**
- Expand the **Additional settings** section, configure **Performance mode** to **General Purpose (Recommended)**

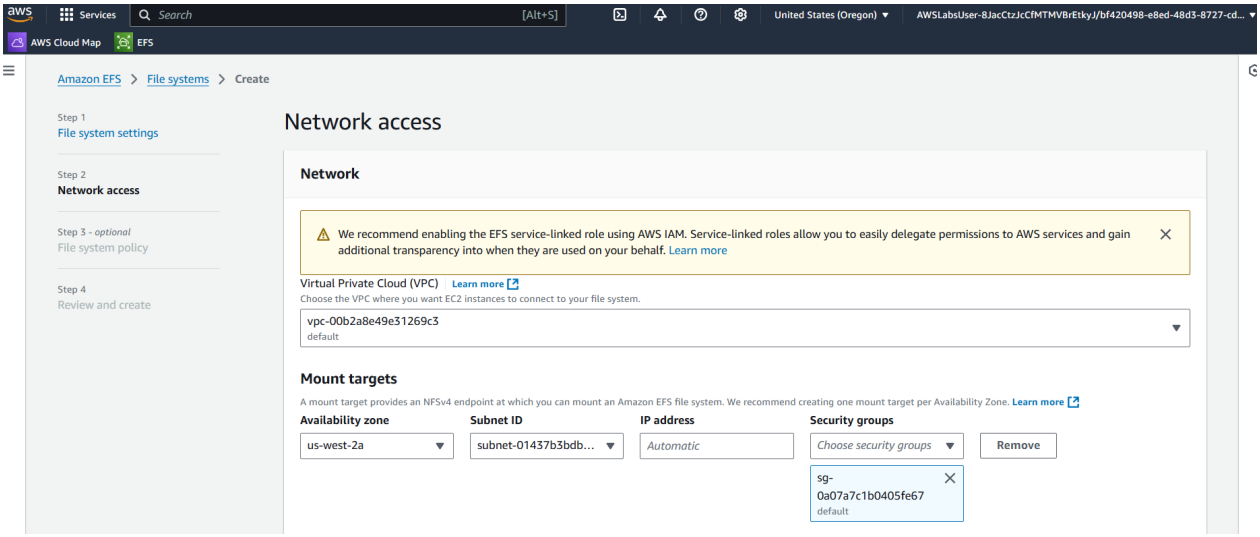


- Expand the **Tags - optional** section, configure the following:
  - **Tag key**: Enter `Name`.
  - **Tag value – *optional***: Enter `myWPEFS`.
  - Leave all other settings at their default value.



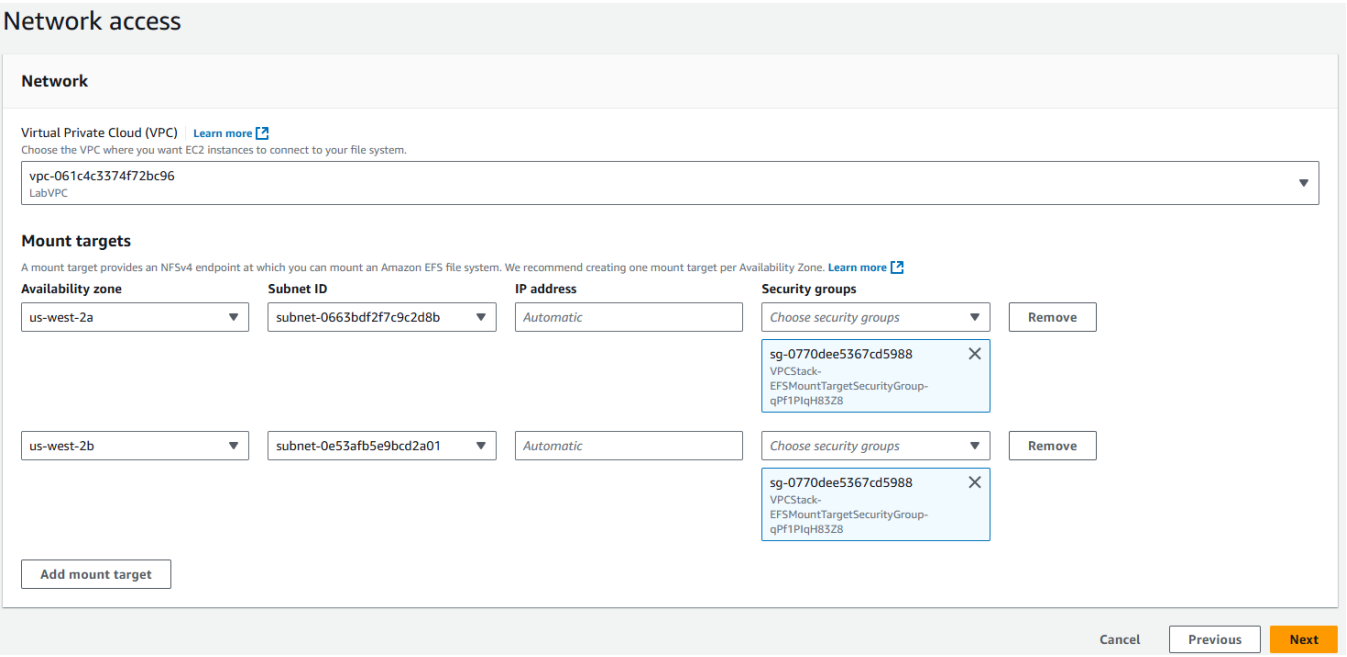
- Choose **Next**.

The **Network access** page is displayed.



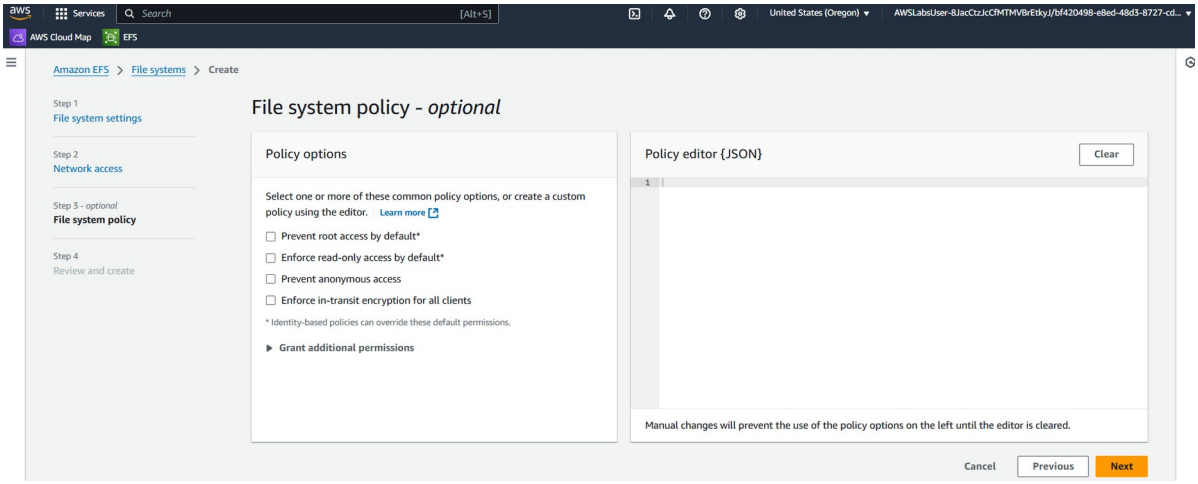
- From the **Virtual Private Cloud (VPC)** dropdown menu, select `LabVPC`.

- For **Mount targets**, configure the following:
    - **Availability zone**: Select the **Availability Zone ending in "a"** from the dropdown menu.
    - **Subnet ID**: Select **AppSubnet1** from the dropdown menu.
    - **Security groups**: Select **EFSMountTargetSecurityGroup** from the dropdown menu.
    - To remove the **default** Security group, choose the **X**.
    - **Availability zone**: Select **Availability Zone ending in "b"** from the dropdown menu.
    - **Subnet ID**: Select **AppSubnet2** from the dropdown menu.
    - **Security groups**: Select **EFSMountTargetSecurityGroup** from the dropdown menu.
    - To remove the **default** Security group, choose the **X**.



- Choose **Next**.

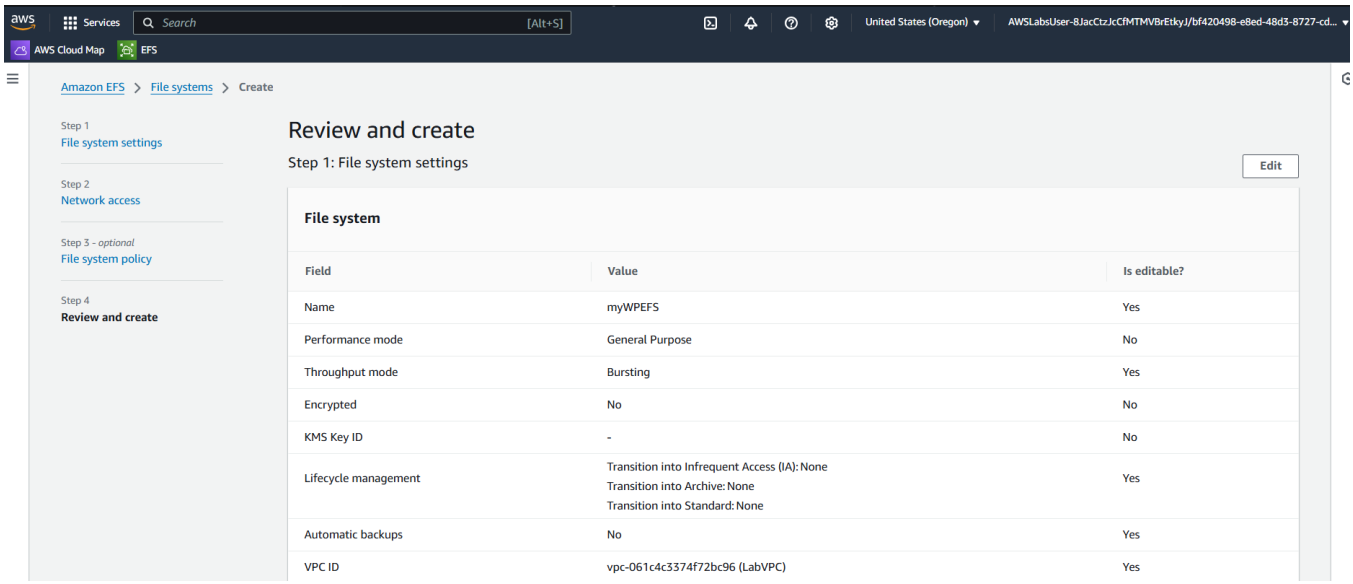
The **File system policy – *optional*** page is displayed.



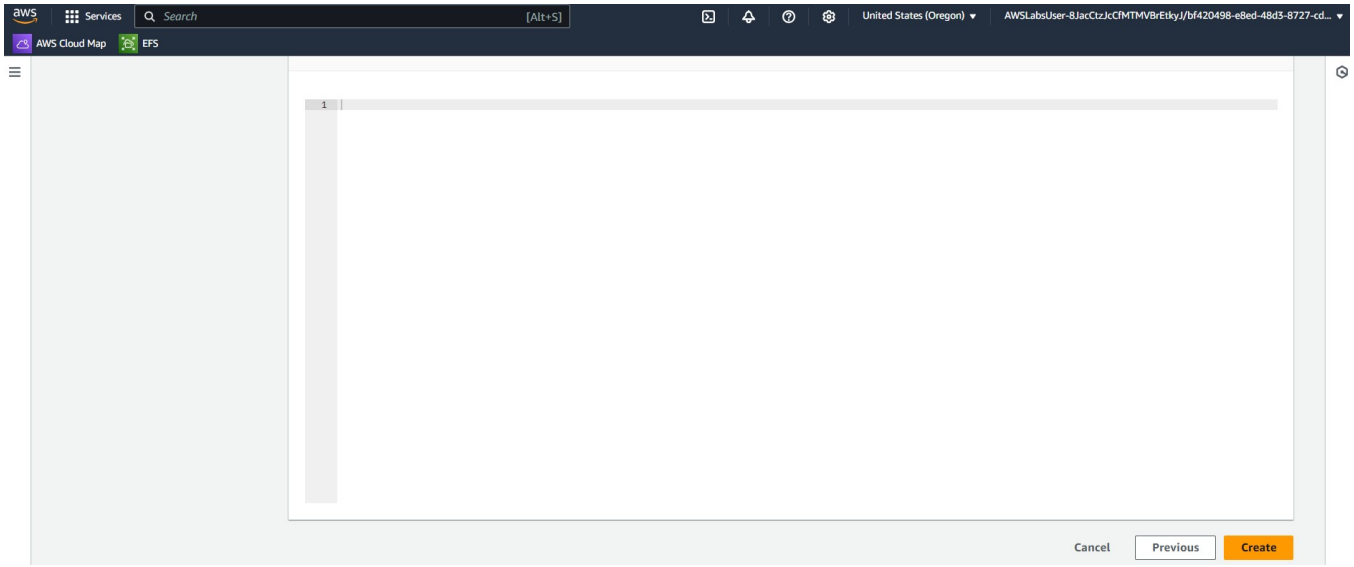
Note: Configuring this page is not necessary in this lab.

- Choose **Next**.

The **Review and create** page is displayed.

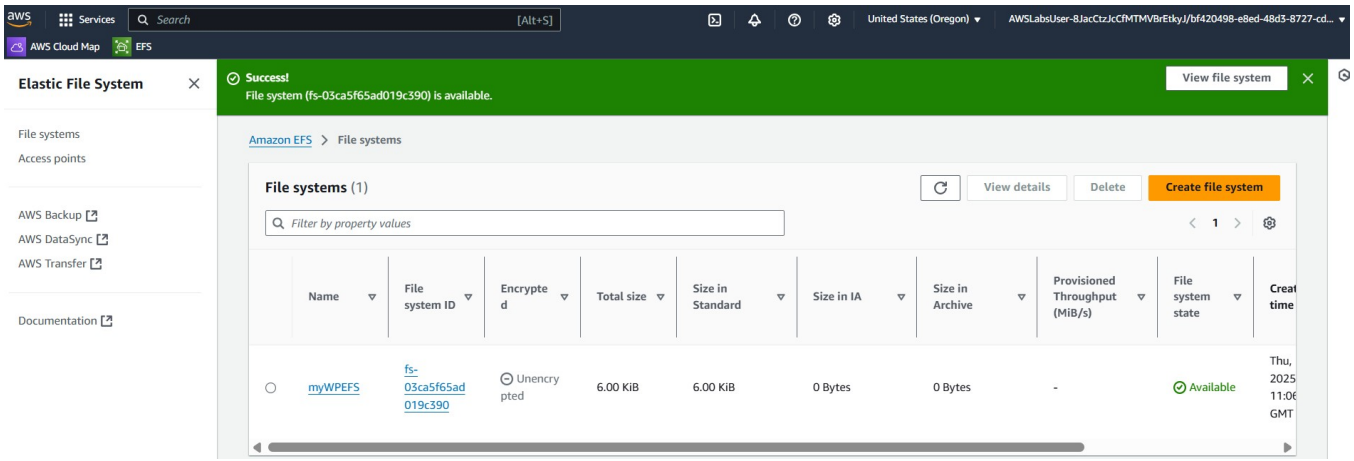


- Scroll to the bottom of the page, and choose **Create**.



A **Success!** File system (fs-xxxxxxx) is available. message is displayed on top of the screen.

The file system state displays as *Available* after several minutes.



Task 3.3: Copy Amazon EFS metadata

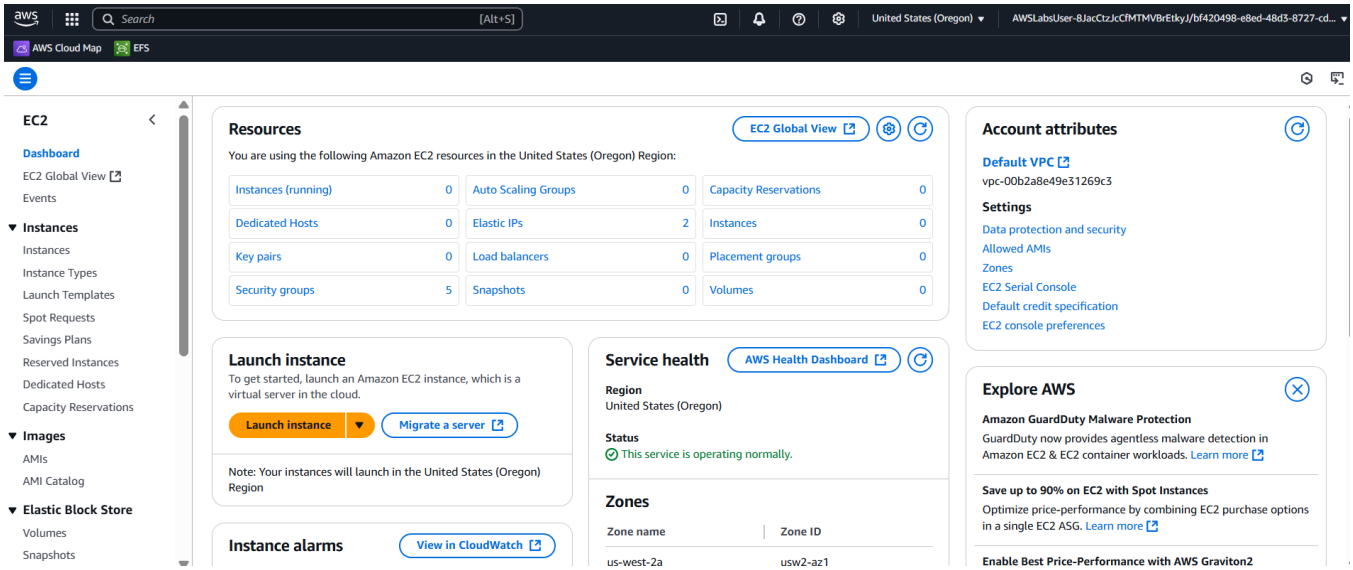
- In the left navigation pane, select **File systems**.
- Copy the **File system ID** generated for *myWPEFS* to a text editor. It has a format like *fs-a1234567*.

Congratulations, you have created the Amazon EFS.

Task 4 instructions: Create an Application Load Balancer

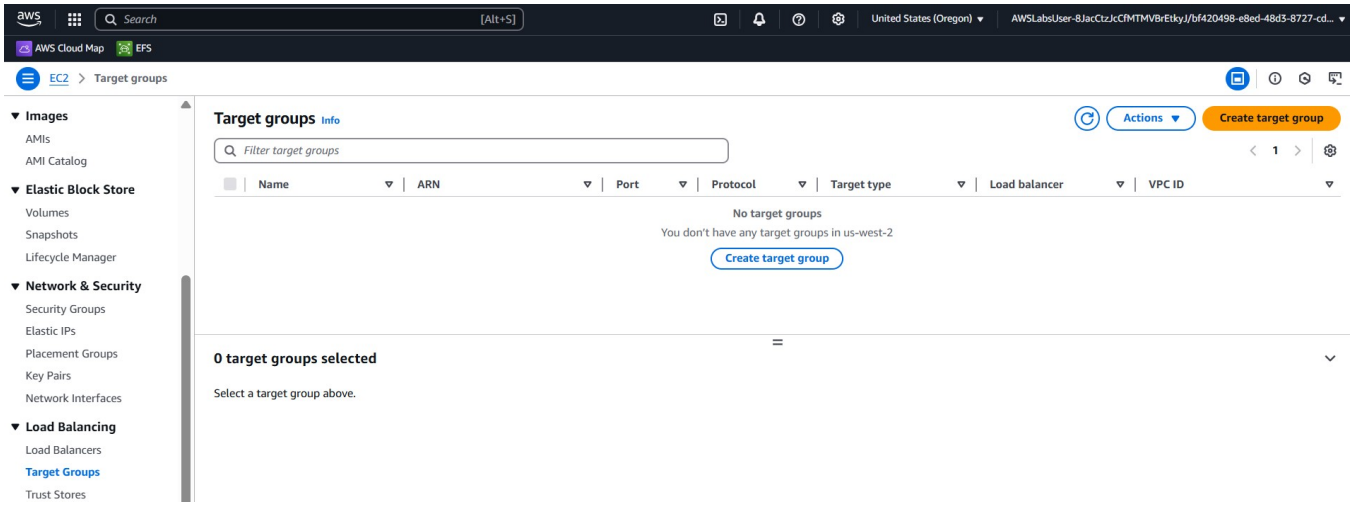
Task 4.1: Navigate to the Amazon EC2 console

- At the top of the AWS Management Console, in the search box, search for and choose `EC2`.



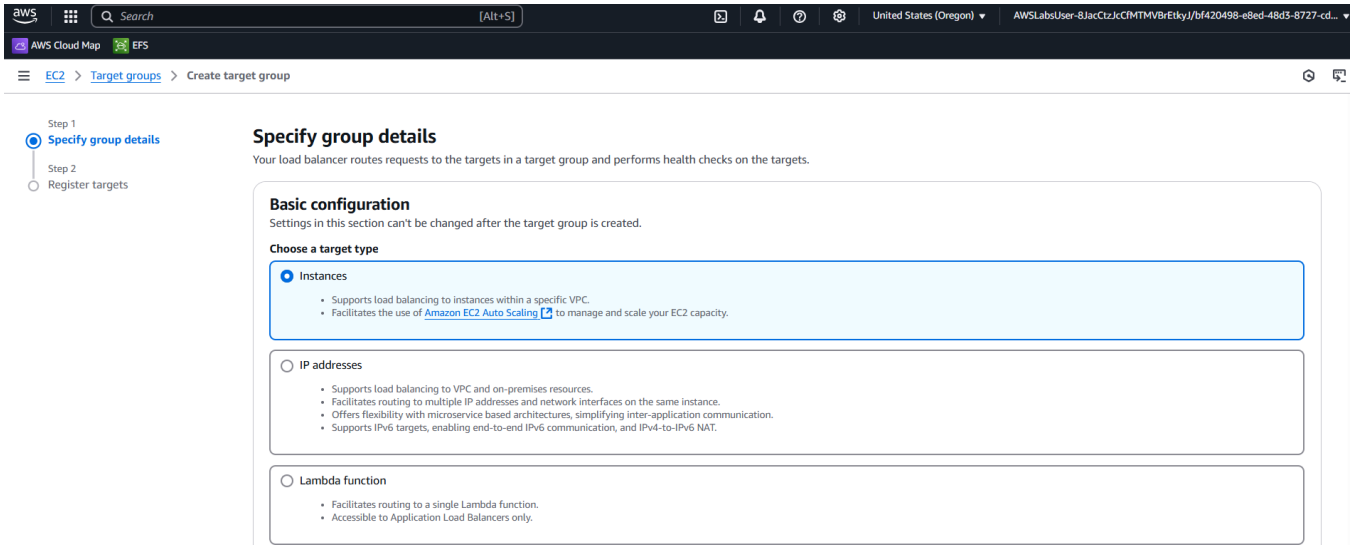
Task 4.2: Create a Target group

- In the left navigation pane, select **Target Group**.

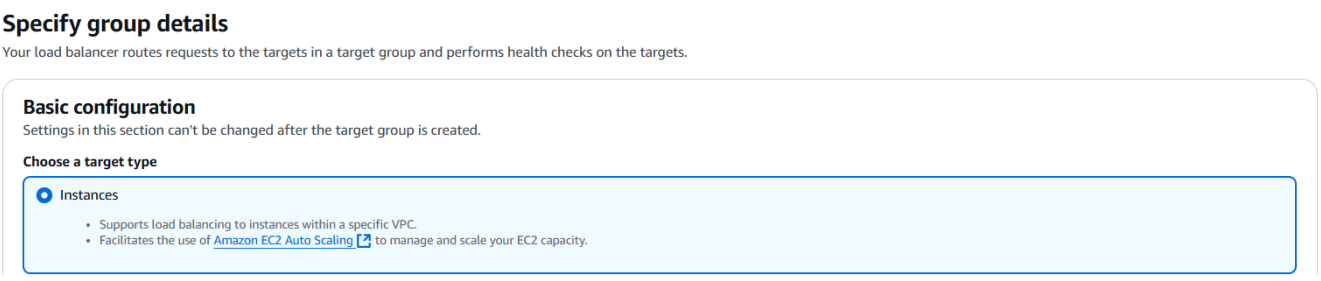


- Choose **Create target group**.

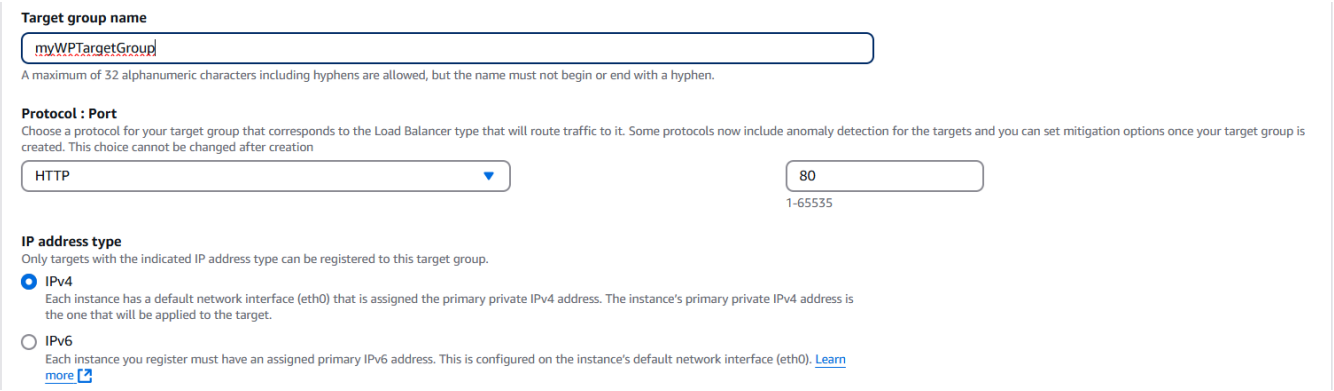
The **Specify group details** page is displayed.



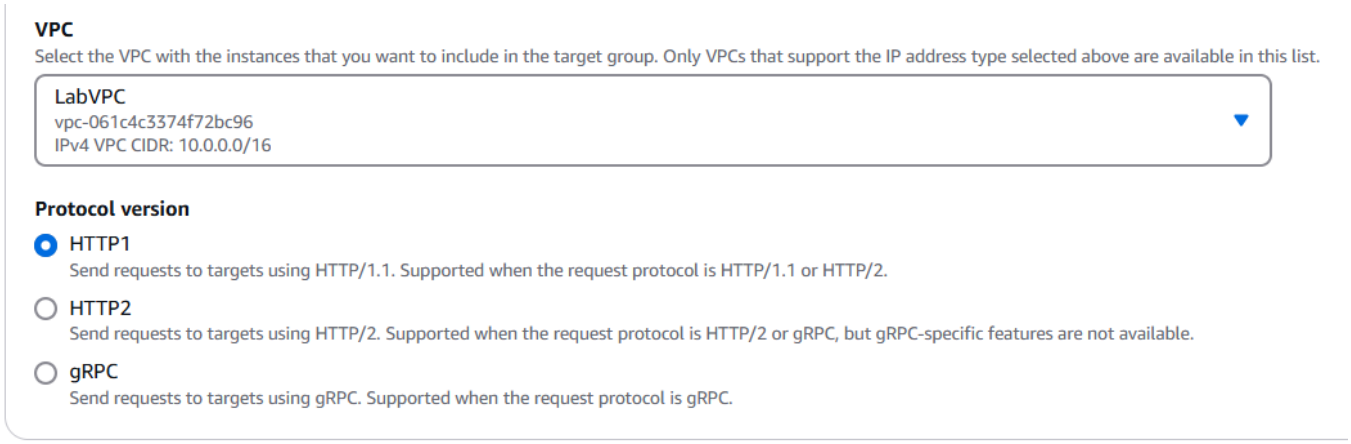
- In the **Basic configuration** section, configure the following:
  - For **Choose a target type**: Select **Instances**.



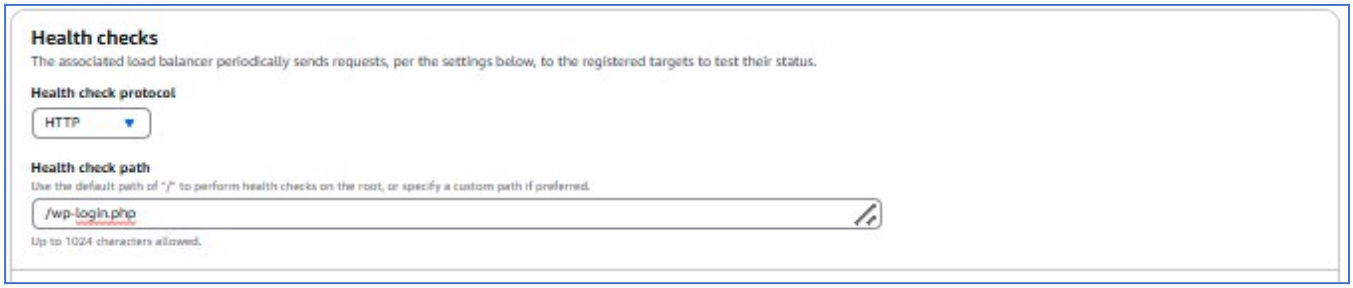
  - For **Target group name**: Enter `myWPTargetGroup`.



  - For **VPC**: Select **LabVPC** from the dropdown menu.

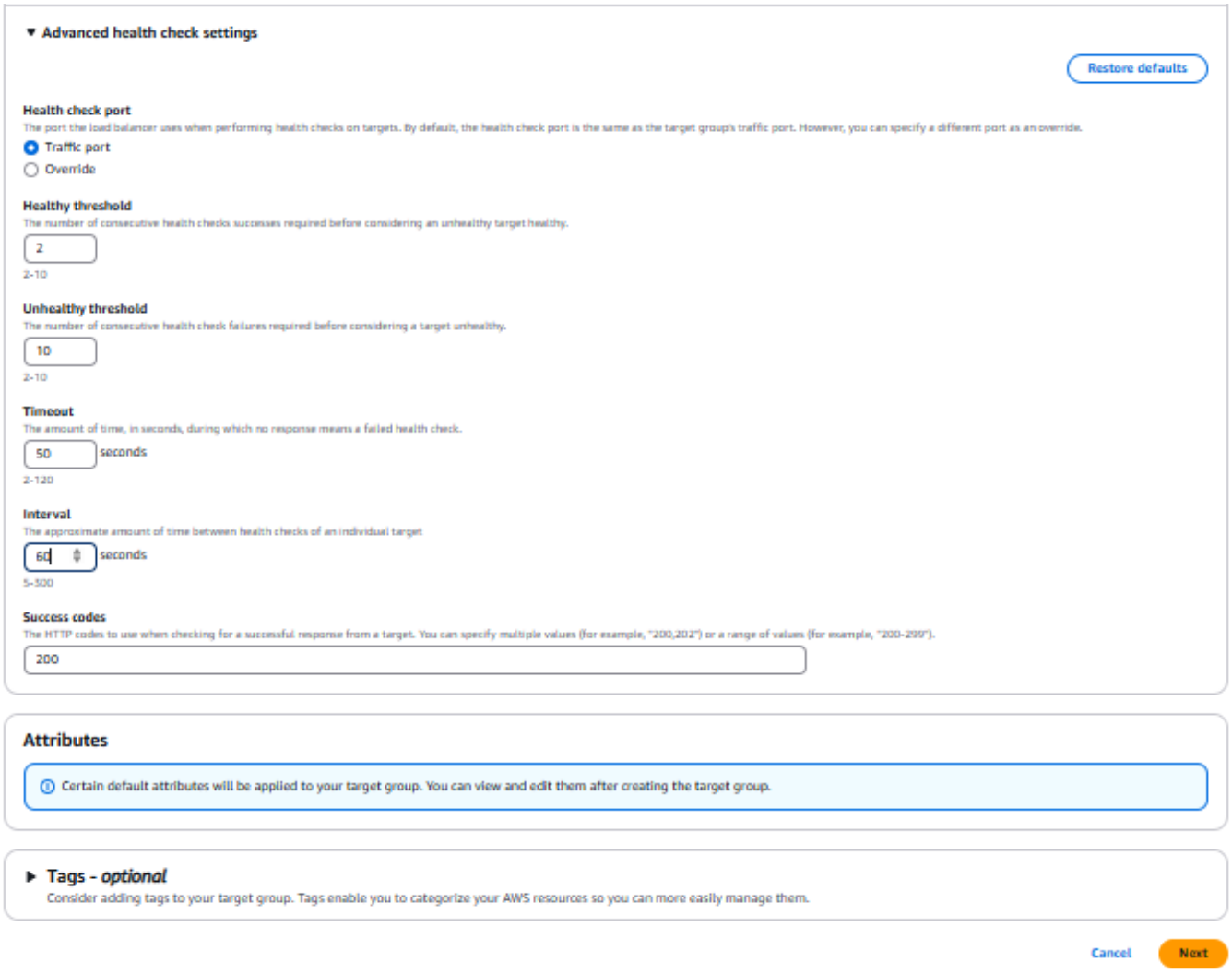


- In the **Health checks** section, configure the following:
  - For **Health check path**: Enter `/wp-login.php` .



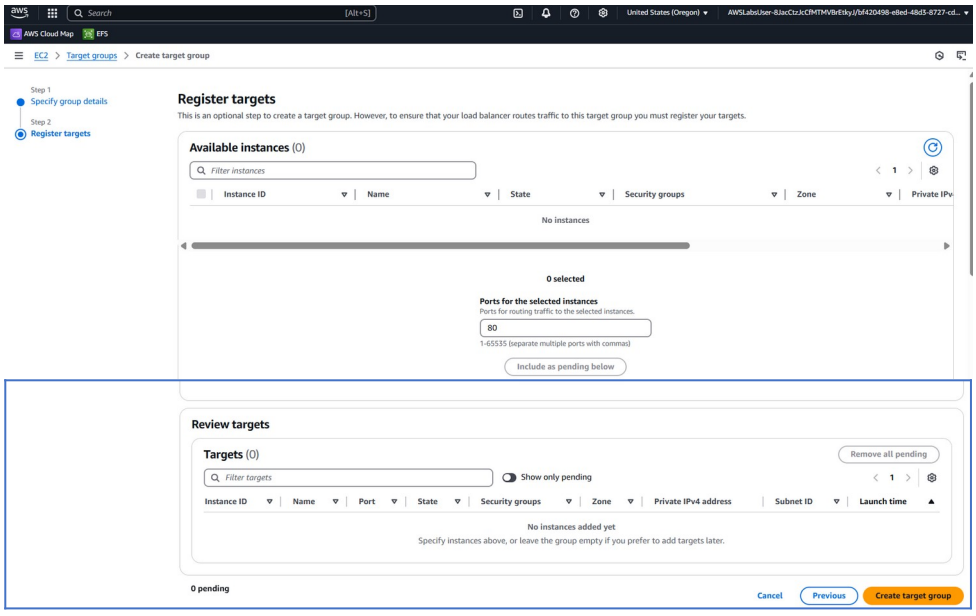
  - Expand the **Advanced health check settings** section and configure the following:
    - **Healthy threshold**: Enter `2`.
    - **Unhealthy threshold**: Enter `10`.
    - **Timeout**: Enter `50`.
    - **Interval**: Enter `60`.

Leave the remaining settings on the page at their default values.



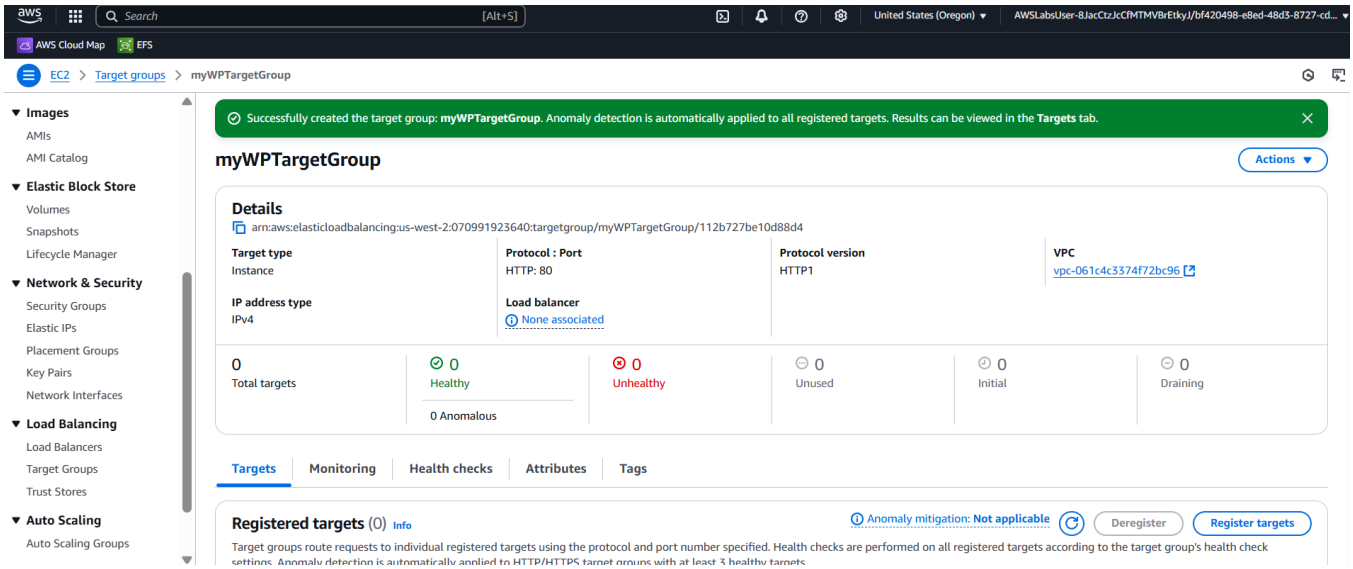
- Choose **Next**.

The **Register targets** page is displayed. There are no targets to register currently.



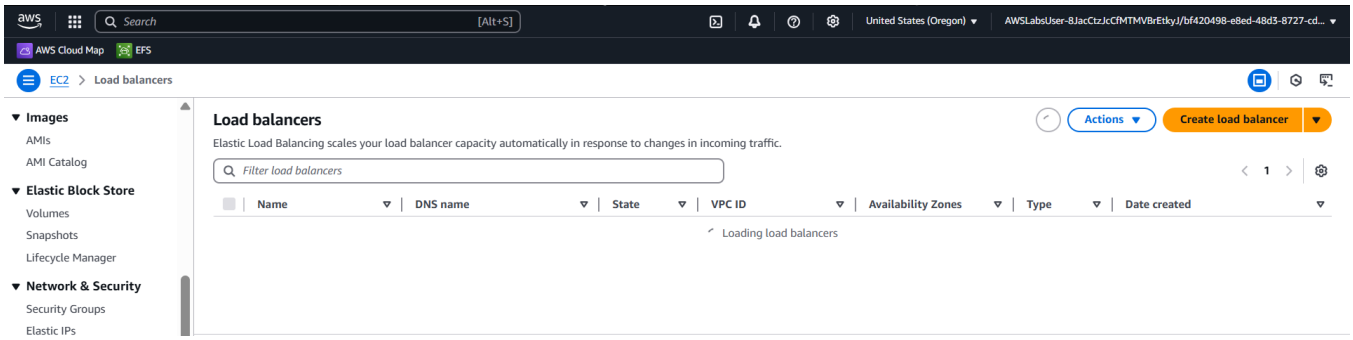
- Scroll to the bottom of the page and choose **Create target group**.

A **Successfully created target group: myWPTargetGroup** message is displayed on top of the screen.



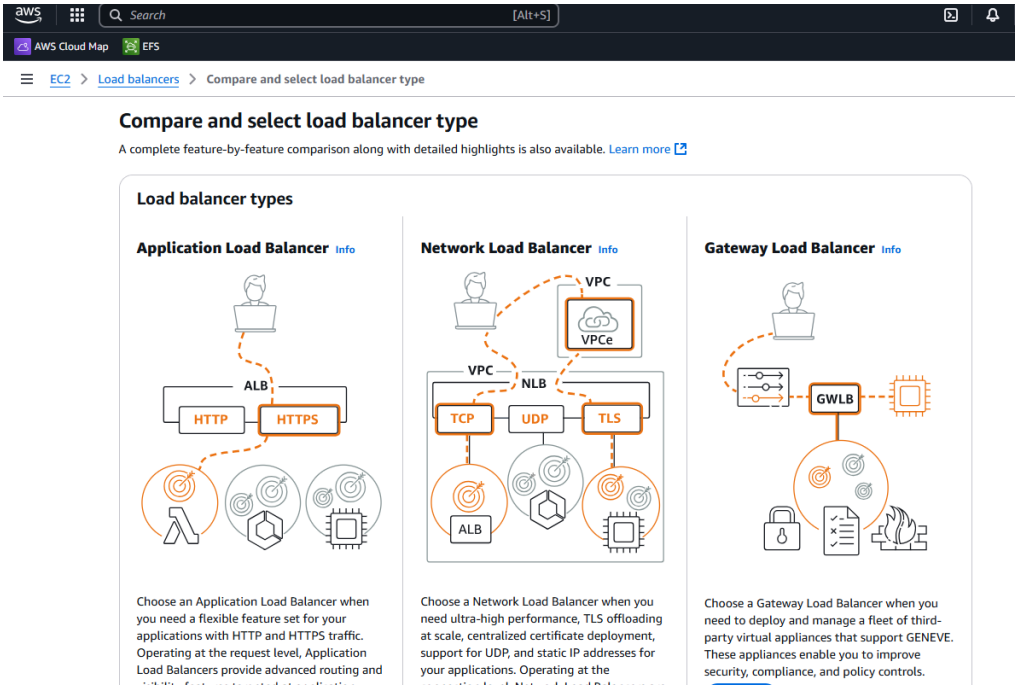
Task 4.3: Create an Application Load Balancer

- In the left navigation pane, choose **Load Balancers**.



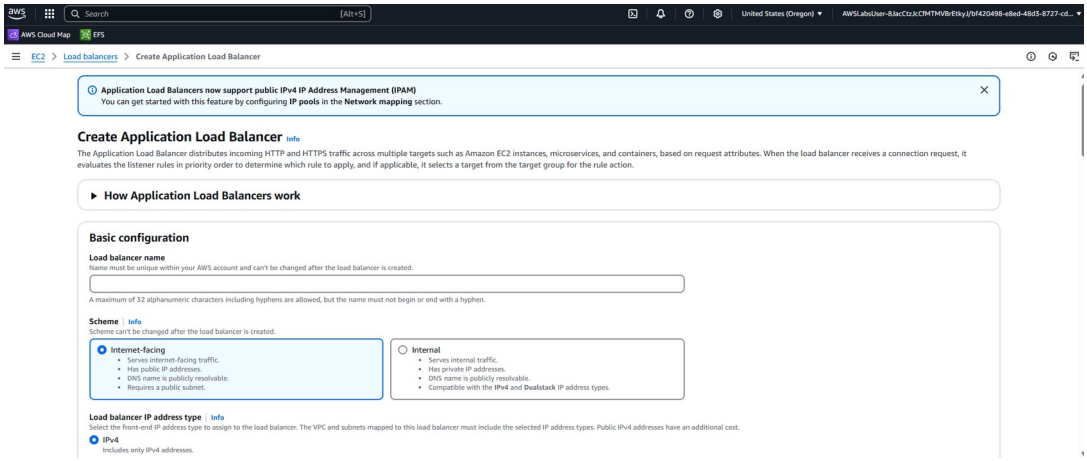
- Choose **Create load balancer**.

The **Compare and select load balancer type** page is displayed.

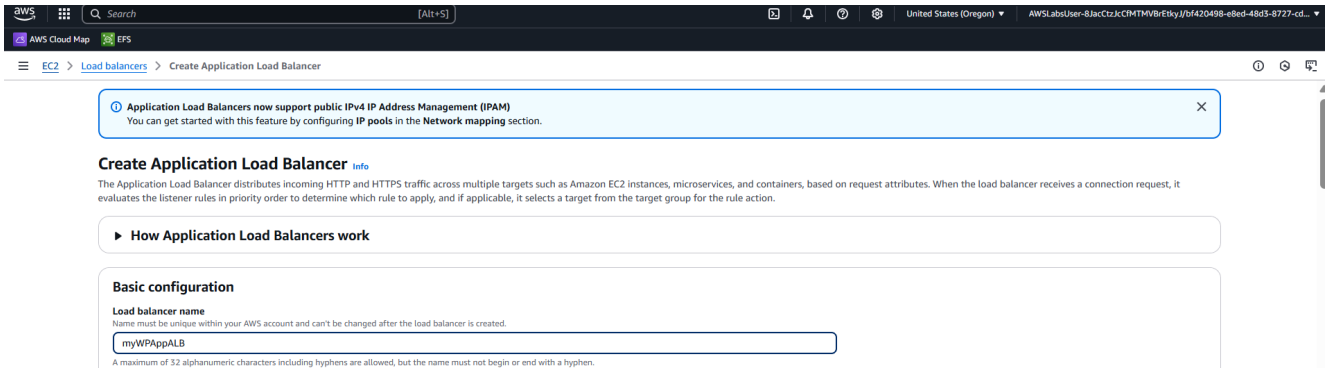


- In the **Load balancer types** section, for **Application Load Balancer**, choose **Create**.

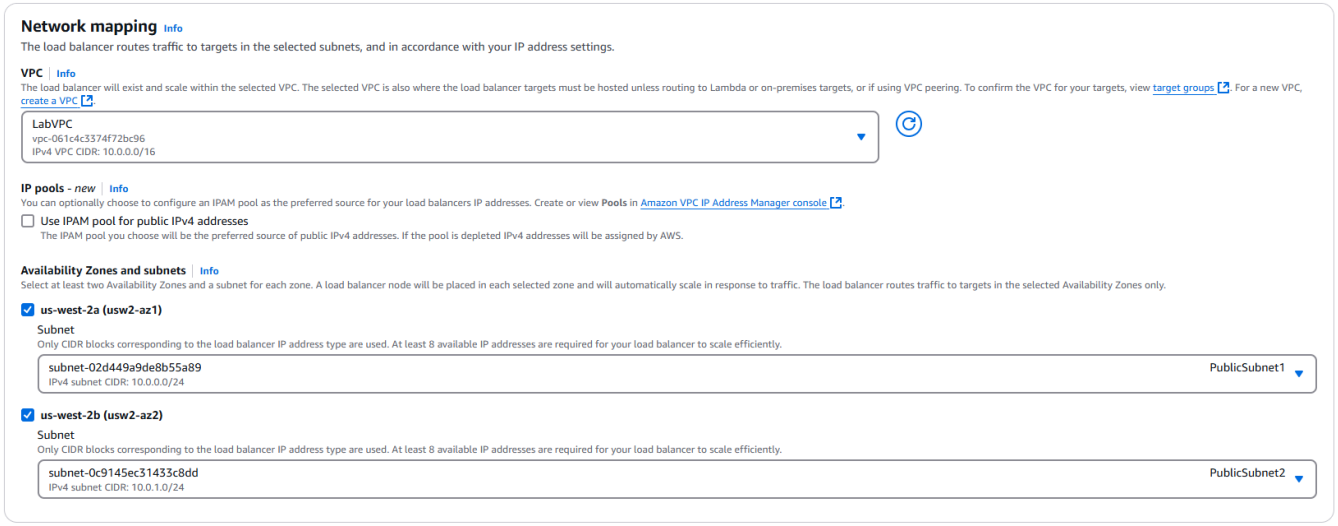
The **Create Application Load Balancer** page is displayed.



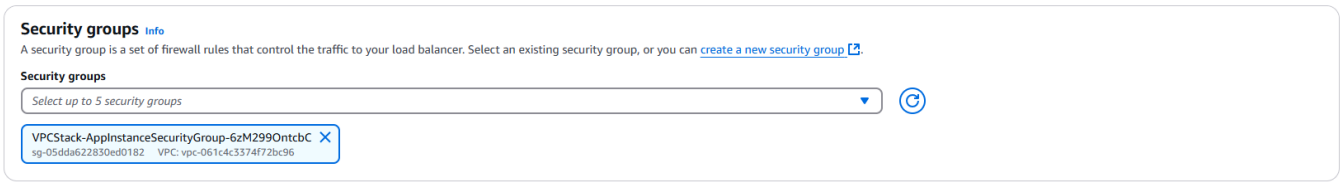
- In the **Basic configuration** section, configure the following:
  - For **Load balancer name**: Enter `myWPAppALB`.



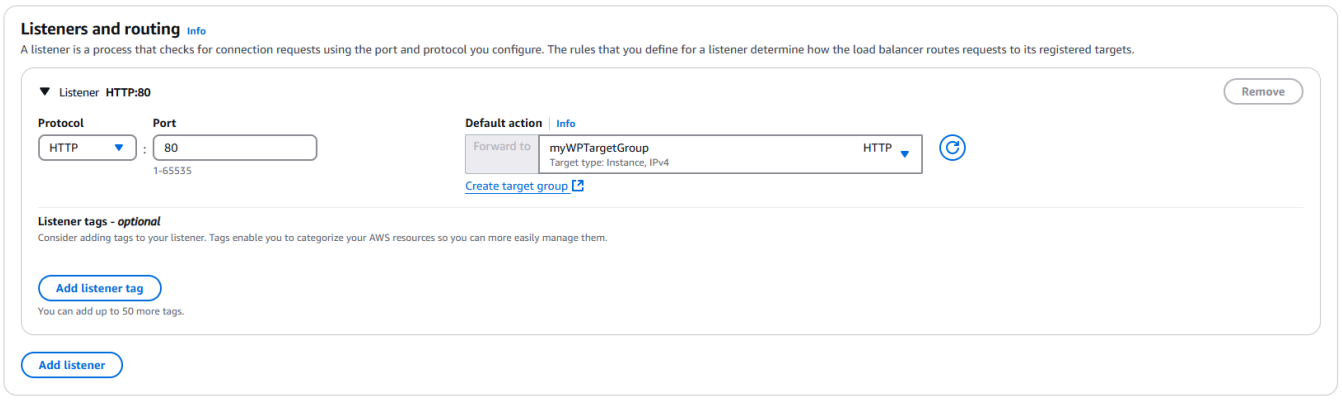
- In the **Network mapping** section, configure the following:
  - **VPC**: Select `LabVPC` from the dropdown menu.
  - **Mappings**:
    - Select the first Availability Zone listed, and select `PublicSubnet1` from the Subnet dropdown menu.
    - Select the second Availability Zone listed, and select `PublicSubnet2` from the Subnet dropdown menu.



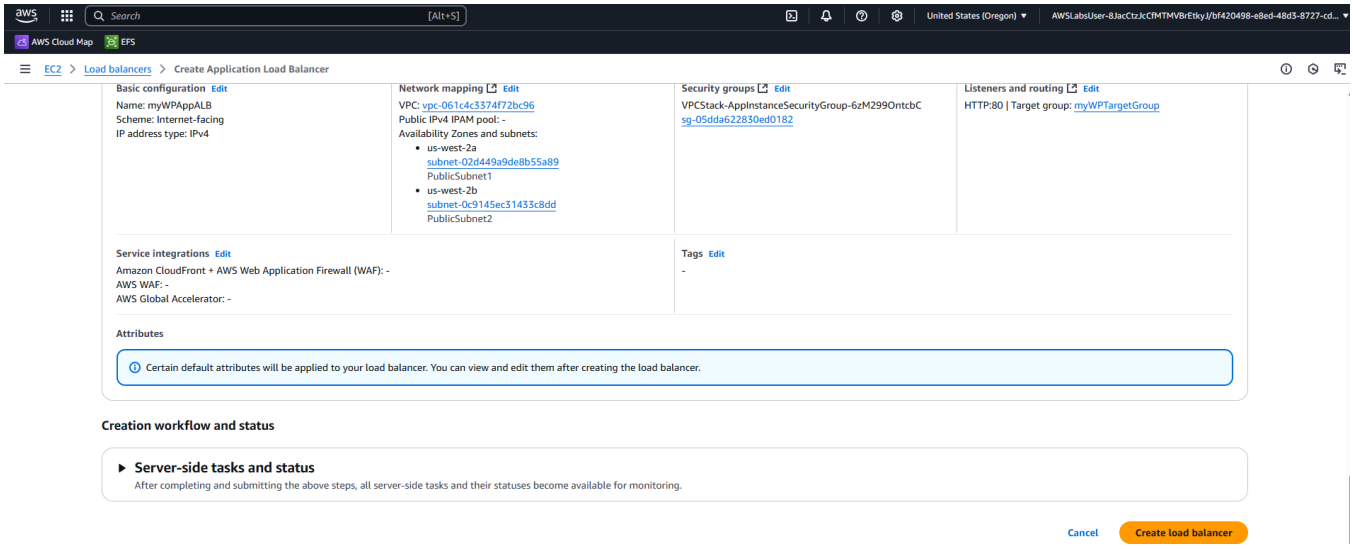
- In the **Security groups** section, configure the following:
  - From the **Security groups** dropdown menu, select **AppInstanceSecurityGroup**.
  - To remove the **default** security group, choose the **X**.



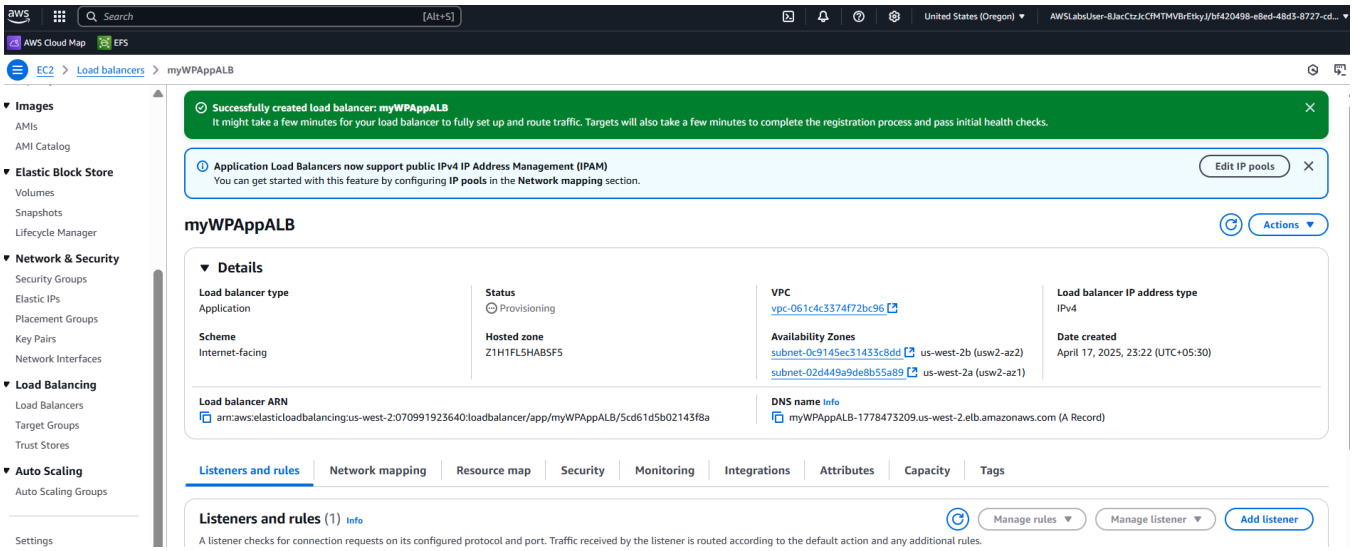
- In the **Listeners and routing** section, configure the following:
  - For **Listener HTTP:80**: from the **Default action** dropdown menu, select **myWPTargetGroup**.



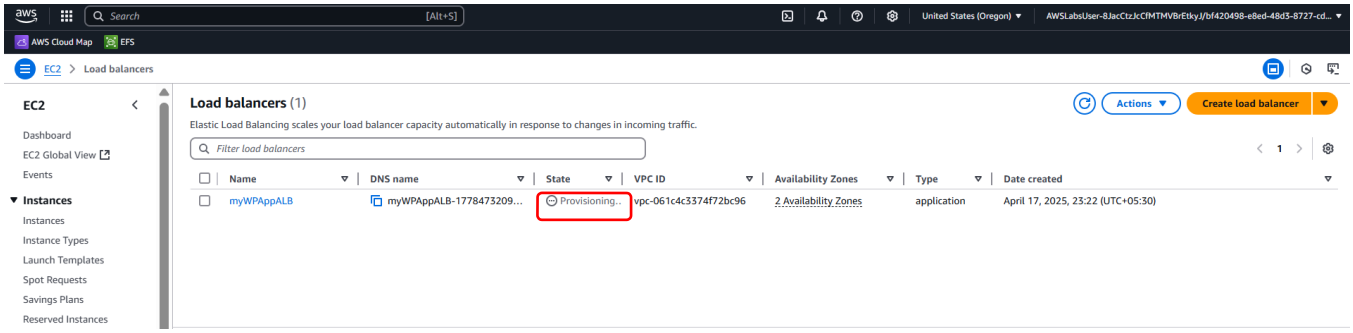
- Scroll to the bottom of the page and choose **Create load balancer**.

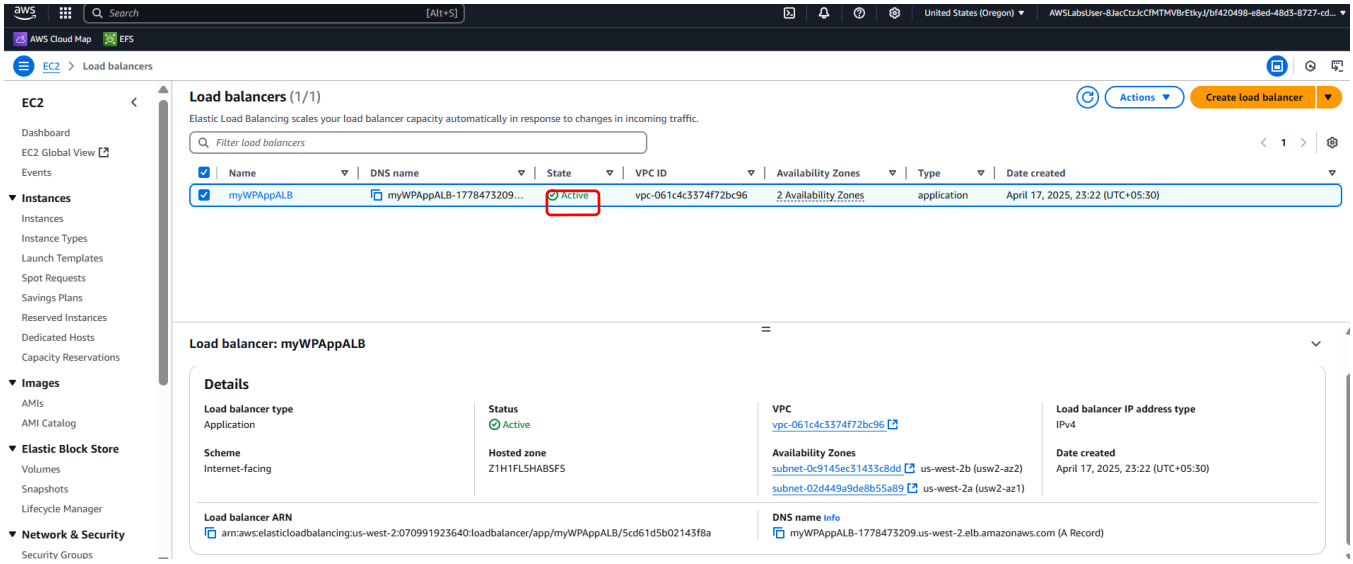


A **Successfully created load balancer: myWPAppALB** message is displayed on top of the screen.

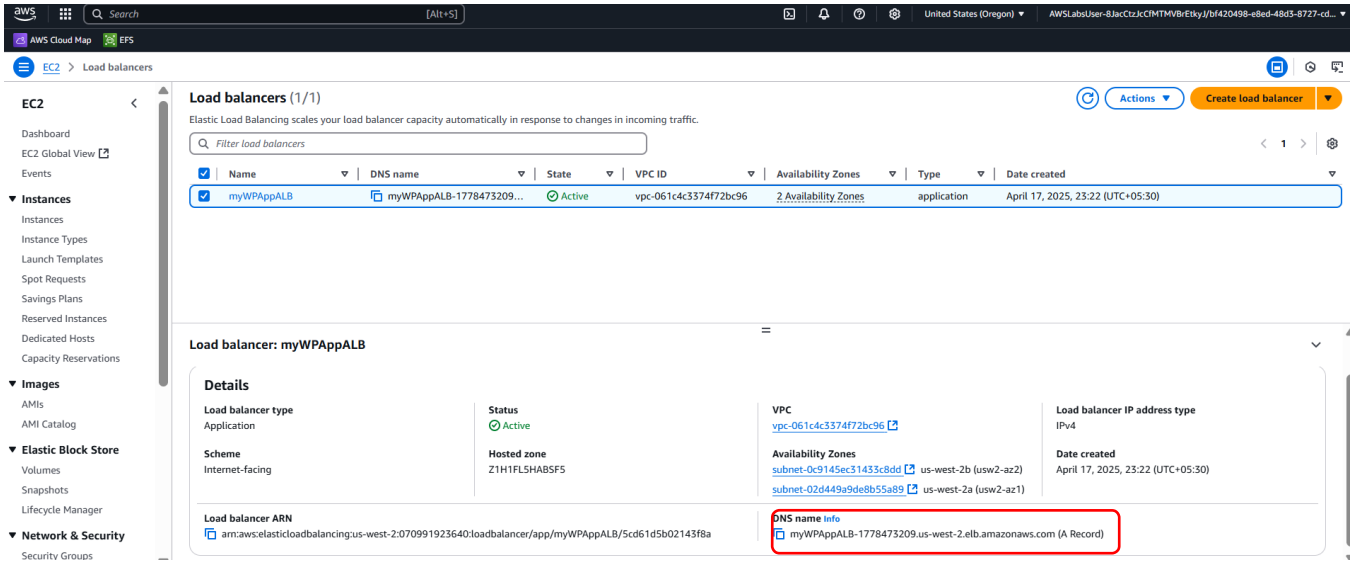


The load balancer is in the *Provisioning* state for a few minutes and then changes to *Active*.





- Copy the **myWPAppALB** load balancer **DNS name** to a text editor.



You have created the target group and an Application Load Balancer.

Task 5: Create a launch template using CloudFormation

In this task, you use a CloudFormation template to deploy the WordPress user data within an Amazon Elastic Compute Cloud (Amazon EC2) Auto Scaling launch template. The template includes the Amazon EFS mount points and the Aurora configurations.

Task 5: Scenario

Up to this point, you created the underlying network resources, the database, an Amazon EFS file system, and an Application Load Balancer. It's time to put it all together. Example Corp. already has a WordPress account, so the cloud engineer uses the settings and configuration from their existing environment to create a

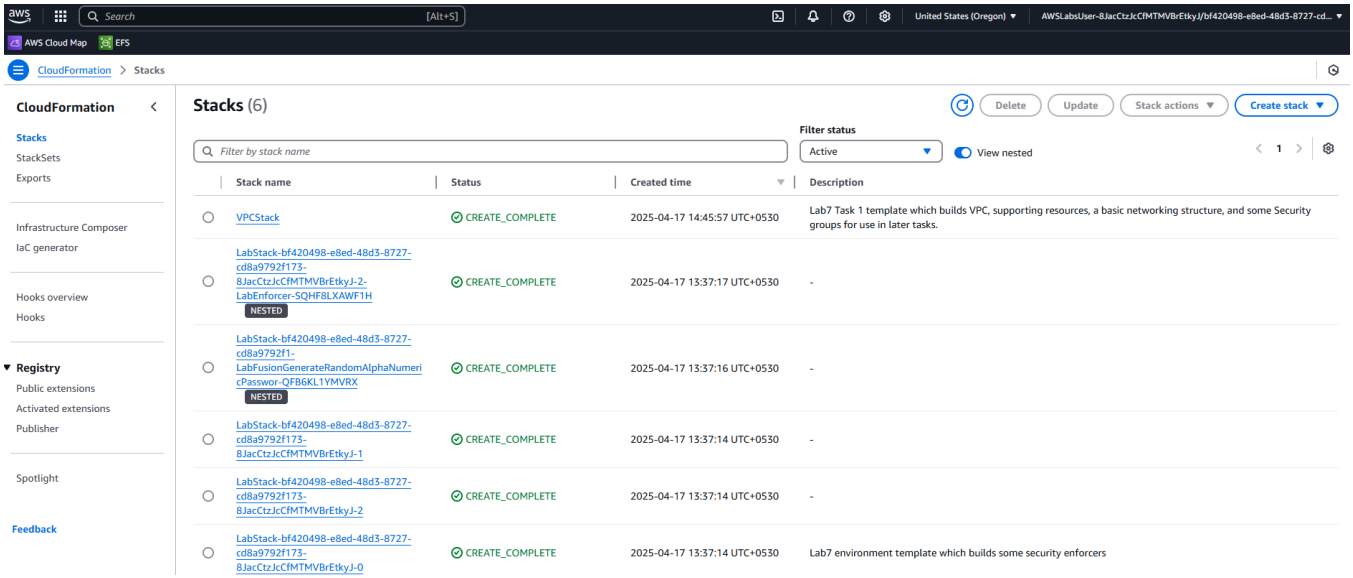
CloudFormation template. This includes all of the new resources you provisioned to set up a launch template.

Review the CloudFormation template and create all of the necessary resources. Pay special attention to the user data script. Check for any errors in the deployment and address those, if needed. After that is completed, you can create the application servers to support traffic to the new environment.

Task 5 instructions: Creating a launch template using CloudFormation

Task 5.1: Navigate to the CloudFormation console

- At the top of the AWS Management Console, in the search box, search for and choose `CloudFormation`.



Task 5.2: Obtain and review the CloudFormation template

- Open the context (right-click) menu on this Task5.yaml link, and choose the option to save the CloudFormation template to your computer.
- Open the downloaded file in a text editor (not a word processor, preferably in Visual Studio Code).
- Review the CloudFormation template.
- Predict what resources are created by this template.

The `Task5.yaml` file is an AWS CloudFormation template designed to automate the deployment of WordPress EC2 servers. It defines a stack that includes parameters for configuring the database, WordPress admin credentials, EC2 instance type, and networking components like the Application Load Balancer and Elastic File System. The template provisions key resources such as a security group for WordPress web servers, security group rules for EFS and RDS access, and a launch template that installs and configures a WordPress environment on Amazon Linux 2023. This setup

includes installing Apache, PHP, and necessary extensions, mounting an EFS volume, downloading WordPress, setting up configuration files, and enabling Opcache. The result is a ready-to-use WordPress instance hosted on AWS infrastructure.

The `Task5.yaml` file consists of the following:

```
AWSTemplateFormatVersion: 2010-09-09 # Specifies the version of the AWS
CloudFormation template.

Description: Stack to create a launch configuration for wordpress EC2
servers. # A description of the CloudFormation stack.

Metadata: # Optional section for adding metadata to the template.
  AWS::CloudFormation::Interface: # Specifies the user interface for
CloudFormation parameters in the AWS console.
    ParameterGroups: # Defines logical groupings of parameters for
better user experience.
      - Label: # Label for the first parameter group.
          default: Database Parameters # The default label for this
group.
        Parameters: # List of parameter names belonging to this group.
          - DatabaseName
          - DatabaseHostName
          - DatabaseUsername
          - DatabasePassword
      - Label: # Label for the second parameter group.
          default: WordPress Parameters # The default label for this
group.
        Parameters: # List of parameter names belonging to this group.
          - Username
          - Password
          - Email
      - Label: # Label for the third parameter group.
          default: Other Parameters # The default label for this group.
        Parameters: # List of parameter names belonging to this group.
          - EC2ServerInstanceType
    ParameterLabels: # Provides user-friendly labels for individual
parameters.
      DatabaseName:
        default: DB name # Default label for the DatabaseName
parameter.
      DatabaseHostName:
        default: Database endpoint # Default label for the
DatabaseHostName parameter.
      DatabaseUsername:
        default: Database User Name # Default label for the
DatabaseUsername parameter.
```

```
      DatabasePassword:
        default: Database Password # Default label for the
DatabasePassword parameter.
      EC2ServerInstanceType:
        default: Instance Type # Default label for the
EC2ServerInstanceType parameter.
      Username:
        default: WordPress admin username # Default label for the
Username parameter.
      Password:
        default: WordPress admin password # Default label for the
Password parameter.
      Email:
        default: WordPress admin email address # Default label for the
Email parameter.

Parameters: # Defines the input parameters for the CloudFormation
stack.

  LatestAL2023AmiId: # Parameter to store the latest Amazon Linux 2023
AMI ID.
    Type: AWS::SSM::Parameter::Value<AWS::EC2::Image::Id> # Specifies
that the parameter value is fetched from AWS Systems Manager Parameter
Store as an EC2 Image ID.
    Default: /aws/service/ami-amazon-linux-latest/al2023-ami-kernel-
default-x86_64 # The default SSM parameter path to get the latest
AL2023 AMI.

  DatabaseName: # Parameter for the WordPress database name.
    AllowedPattern: ^([a-zA-Z0-9]*)$ # Regular expression to allow only
alphanumeric characters.
    Description: The Amazon RDS database name. # Description of the
parameter.
    Type: String # Specifies the parameter data type as a string.
    Default: WPDatabase # The default value for the database name.

  DatabaseHostName: # Parameter for the Amazon RDS database endpoint.
    Description: Amazon RDS DB Endpoint # Description of the parameter.
    Type: String # Specifies the parameter data type as a string.

  DatabaseUsername: # Parameter for the Amazon RDS username.
    AllowedPattern: ^([a-zA-Z0-9]*)$ # Regular expression to allow only
alphanumeric characters.
    Description: The Amazon RDS username. # Description of the
parameter.
    ConstraintDescription: Must contain only alphanumeric characters
(minimum 8; maximum 16). # User-friendly message for the allowed
pattern.
```

```
    MaxLength: 16 # Maximum length of the username.
    MinLength: 5 # Minimum length of the username.
    Type: String # Specifies the parameter data type as a string.
    Default: admin # The default value for the database username.

  DatabasePassword: # Parameter for the Amazon RDS password.
    Description: The Amazon RDS password. # Description of the
parameter.
    MaxLength: 41 # Maximum length of the password.
    MinLength: 6 # Minimum length of the password.
    NoEcho: true # Specifies that the password value should not be
displayed in the AWS console.
    Type: String # Specifies the parameter data type as a string.

  Username: # Parameter for the WordPress admin username.
    AllowedPattern: ^([a-zA-Z0-9]*)$ # Regular expression to allow only
alphanumeric characters.
    Description: WordPress admin username. # Description of the
parameter.
    ConstraintDescription: Must contain only alphanumeric characters
(minimum 8; maximum 16). # User-friendly message for the allowed
pattern.
    MaxLength: 16 # Maximum length of the username.
    MinLength: 5 # Minimum length of the username.
    Type: String # Specifies the parameter data type as a string.
    Default: wpadmin # The default value for the WordPress admin
username.

  Password: # Parameter for the WordPress admin password.
    Description: WordPress admin password. # Description of the
parameter.
    MaxLength: 41 # Maximum length of the password.
    MinLength: 6 # Minimum length of the password.
    NoEcho: true # Specifies that the password value should not be
displayed in the AWS console.
    Type: String # Specifies the parameter data type as a string.

  Email: # Parameter for the WordPress admin email address.
    Type: String # Specifies the parameter data type as a string.
    Description: Email address for WordPress notifications #
Description of the parameter.

  EC2ServerInstanceType: # Parameter for the EC2 instance type for the
WordPress servers.
    Description: Amazon EC2 Instance Type # Description of the
parameter.
    Type: String # Specifies the parameter data type as a string.
    Default: t3.medium # The default EC2 instance type.
```

```
    AllowedValues: # List of allowed values for the instance type.
      - t3.small
      - t3.medium

  ALBDnsName: # Parameter for the DNS name of the Application Load
Balancer.
    Description: Application Load Balancer DNS. # Description of the
parameter.
    Type: String # Specifies the parameter data type as a string.

  WPElasticFileSystemID: # Parameter for the ID of the Elastic File
System.
    Description: Elastic File System ID. # Description of the
parameter.
    Type: String # Specifies the parameter data type as a string.

Resources: # Defines the AWS resources that will be created by the
stack.

  WebTierSecurityGroup: # Defines a security group for the web
instances.
    Type: AWS::EC2::SecurityGroup # Specifies the resource type as an
EC2 security group.
    Properties: # Defines the properties of the security group.
      GroupDescription: Security group for web instances # A
description for the security group.
      GroupName: Wordpress Servers Security Group # The name of the
security group.
      SecurityGroupIngress: # Defines the inbound rules for the
security group.
        - IpProtocol: tcp # Allows TCP traffic.
          FromPort: 80 # Allows traffic on port 80 (HTTP).
          ToPort: 80 # Allows traffic on port 80 (HTTP).
          SourceSecurityGroupId: !ImportValue
AppInstanceSecurityGroupID # Allows traffic from the security group of
the application load balancer (imported from another stack).
      VpcId: # Specifies the VPC ID for the security group.
        !ImportValue VPCID # Imports the VPC ID from another stack.

  EFSSecurityGroupInboundRule: # Defines an inbound rule for the EFS
security group.
    Type: AWS::EC2::SecurityGroupIngress # Specifies the resource type
as an EC2 security group inbound rule.
    Properties: # Defines the properties of the inbound rule.
      IpProtocol: tcp # Allows TCP traffic.
      FromPort: 2049 # Allows traffic on port 2049 (NFS).
      ToPort: 2049 # Allows traffic on port 2049 (NFS).
```

```
      SourceSecurityGroupId: !Ref WebTierSecurityGroup # Allows traffic
from the web tier security group defined in this stack.
      GroupId: !ImportValue EFSMountTargetSecurityGroupID # Specifies
the ID of the EFS mount target security group (imported from another
stack).

  RDSSecurityGroupInboundRule: # Defines an inbound rule for the RDS
security group.
    Type: AWS::EC2::SecurityGroupIngress # Specifies the resource type
as an EC2 security group inbound rule.
    Properties: # Defines the properties of the inbound rule.
      IpProtocol: tcp # Allows TCP traffic.
      FromPort: 3306 # Allows traffic on port 3306 (MySQL/MariaDB).
      ToPort: 3306 # Allows traffic on port 3306 (MySQL/MariaDB).
      SourceSecurityGroupId: !Ref WebTierSecurityGroup # Allows traffic
from the web tier security group defined in this stack.
      GroupId: !ImportValue RDSSecurityGroupID # Specifies the ID of
the RDS security group (imported from another stack).
# Lab Launch Template

  LabLaunchTemplate: # Defines an EC2 launch template named
'LabLaunchTemplate'.
    Type: AWS::EC2::LaunchTemplate # Specifies the resource type as an
EC2 Launch Template.
    Properties: # Defines the properties of the launch template.
      LaunchTemplateName: LabLaunchTemplate # Sets the name of the
launch template.
      LaunchTemplateData: # Defines the configuration data used to
launch EC2 instances.
        UserData: # Specifies the user data script to be executed on
instance launch.
          "Fn::Base64": # Encodes the following multi-line string in
Base64 format.
            !Sub
            - | # Substitutes variables within the following multi-line
string.
              #!/bin/bash -xe # Shebang line to execute the script with
error checking (exit on error).
              DB_NAME=${DatabaseName} # Sets a shell variable DB_NAME
to the value of the CloudFormation parameter DatabaseName.
              DB_HOSTNAME=${DatabaseHostName} # Sets a shell variable
DB_HOSTNAME to the value of the CloudFormation parameter
DatabaseHostName.
              DB_USERNAME="${DatabaseUsername}" # Sets a shell variable
DB_USERNAME to the value of the CloudFormation parameter
DatabaseUsername.
```

```
              DB_PASSWORD="${DatabasePassword}" # Sets a shell variable
DB_PASSWORD to the value of the CloudFormation parameter
DatabasePassword.
              WP_ADMIN=${Username} # Sets a shell variable WP_ADMIN to
the value of the CloudFormation parameter Username.
              WP_PASSWORD="${Password}" # Sets a shell variable
WP_PASSWORD to the value of the CloudFormation parameter Password.
              WP_EMAIL=${Email} # Sets a shell variable WP_EMAIL to the
value of the CloudFormation parameter Email.
              LB_HOSTNAME=${ALBDnsName} # Sets a shell variable
LB_HOSTNAME to the value of the CloudFormation parameter ALBDnsName.

              #package installation
              dnf update -y # Updates all installed packages on the
Amazon Linux system without prompting for confirmation.
              dnf install httpd gcc-c++ php wget php-
{pear,devel,fpm,mysqli,cgi,common,curl,mbstring,gd,mysqlnd,gettext,bcma
th,json,xml,fpm,intl,zip,devel,opcache} -y # Installs the Apache web
server (httpd), C++ compiler (gcc-c++), PHP, wget, and various
essential PHP extensions without prompting for confirmation.

              systemctl enable nfs-server.service # Enables the Network
File System (NFS) server service to start automatically on boot.
              systemctl start nfs-server.service # Starts the NFS
server service immediately.

              mkdir -p /var/www/wordpress # Creates the directory
/var/www/wordpress if it doesn't exist, including any necessary parent
directories.
              mount -t nfs4 -o
nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2 $
{ElasticFileSystem}.efs.${AWS::Region}.amazonaws.com:/
/var/www/wordpress # Mounts the specified Elastic File System volume to
the /var/www/wordpress directory using NFS version 4.1 with specific
mount options.

              ## create site config
              cat <<EOF >/etc/httpd/conf.d/wordpress.conf # Creates a
new Apache configuration file named wordpress.conf in the
/etc/httpd/conf.d/ directory.
              ServerName 127.0.0.1:80 # Sets the server name for the
virtual host to localhost on port 80.
              DocumentRoot /var/www/wordpress # Sets the document root
for the virtual host to the /var/www/wordpress directory.
              <Directory /var/www/wordpress> # Defines the directory
directives for the /var/www/wordpress directory.
                Options Indexes FollowSymLinks # Allows directory
indexing and following of symbolic links within the directory.
```

```
                AllowOverride All # Allows .htaccess files to override
server configurations within the directory.
                Require all granted # Grants access to all clients for
this directory.
              </Directory>
              EOF

              pecl install igbinary # Installs the igbinary PHP
extension, which provides a faster PHP serializer.
              cd /tmp # Changes the current working directory to /tmp.

              ## install WordPress and WP CLI
              curl -o /bin/wp https://raw.githubusercontent.com/wp-
cli/builds/gh-pages/phar/wp-cli.phar # Downloads the WordPress Command-
Line Interface (WP-CLI) as an executable phar file and saves it as
/bin/wp.
              chmod +x /bin/wp # Makes the /bin/wp file executable.
              wget -P /tmp/ https://wordpress.org/latest.tar.gz #
Downloads the latest WordPress archive to the /tmp/ directory.
              tar -vxzf latest.tar.gz -C /var/www/ # Extracts the
contents of the WordPress archive to the /var/www/ directory.
              cp /var/www/wordpress/wp-config-sample.php
/var/www/wordpress/wp-config.php # Copies the sample WordPress
configuration file to the active configuration file.
              cd /var/www/wordpress/ # Changes the current working
directory to the WordPress installation directory.
              sed -i 's/database_name_here/'"$DB_NAME"'/' wp-config.php
# Replaces the placeholder 'database_name_here' in wp-config.php with
the actual database name.
              sed -i 's/username_here/'"$DB_USERNAME"'/' wp-config.php
# Replaces the placeholder 'username_here' in wp-config.php with the
database username.
              sed -i 's/password_here/'"$DB_PASSWORD"'/' wp-config.php
# Replaces the placeholder 'password_here' in wp-config.php with the
database password.
              sed -i 's/localhost/'"$DB_HOSTNAME"'/' wp-config.php #
Replaces 'localhost' in wp-config.php with the actual database
hostname.

              # install WordPress if not installed
              # use public alb host name if wp domain name was empty
              if ! $(wp core is-installed --allow-root); then # Checks
if WordPress is already installed using WP-CLI.
                wp core install --url="http://$LB_HOSTNAME" --
title='Wordpress on AWS' --admin_user="$WP_ADMIN" --
admin_password="$WP_PASSWORD" --admin_email="$WP_EMAIL" --allow-root #
Installs WordPress using WP-CLI with the provided database and admin
credentials.
```

```
                wp plugin install w3-total-cache --allow-root #
Installs the W3 Total Cache plugin using WP-CLI.
                chown -R apache:apache /var/www/wordpress # Recursively
changes the ownership of the /var/www/wordpress directory and its
contents to the apache user and group.
                chmod u+wrx /var/www/wordpress/wp-content/* # Adds
read, write, and execute permissions for the owner to all files and
directories within the wp-content directory.
                if [ ! -f /var/www/wordpress/opcache-instanceid.php ];
then # Checks if the opcache-instanceid.php file does not exist.
                  wget -P /var/www/wordpress/
https://s3.amazonaws.com/aws-refarch/wordpress/latest/bits/opcache-
instanceid.php # Downloads the opcache-instanceid.php script to the
WordPress directory.
                fi
              fi
              RESULT=$? # Captures the exit code of the last executed
command.
              echo $RESULT # Prints the exit code of the last command.
              if [ $RESULT -eq 0 ]; then # Checks if the exit code of
the last command was 0 (success).
                touch /var/www/wordpress/wordpress.initialized #
Creates an empty file named wordpress.initialized in the WordPress
directory to indicate successful initialization.
              else # If the last command failed.
                touch /var/www/wordpress/wordpress.failed # Creates an
empty file named wordpress.failed in the WordPress directory to
indicate a failed initialization.
              fi

              ## install opcache
              # create hidden opcache directory locally & change owner
to apache

              mkdir -p /var/www/.opcache # Creates a hidden directory
named .opcache within /var/www/.
              # enable opcache in /etc/php-7.0.d/10-opcache.ini
              sed -i
's/;opcache.file_cache=.*/opcache.file_cache=\/var\/www\/.opcache/'
/etc/php.d/10-opcache.ini # Modifies the PHP-FPM opcache configuration
file to set the opcache file cache directory.
              sed -i
's/opcache.memory_consumption=.*/opcache.memory_consumption=512/'
/etc/php.d/10-opcache.ini # Modifies the PHP-FPM opcache configuration
file to set the opcache memory consumption to 512MB.
              # download opcache-instance.php to verify opcache status
              if [ ! -f /var/www/wordpress/opcache-instanceid.php ];
then # Checks if the opcache-instanceid.php file does not exist.
```

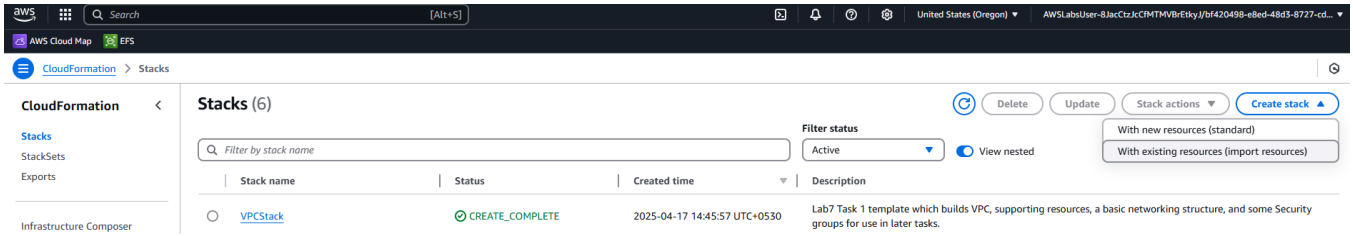
```
                wget -P /var/www/wordpress/
https://s3.amazonaws.com/aws-refarch/wordpress/latest/bits/opcache-
instanceid.php # Downloads the opcache-instanceid.php script to the
WordPress directory.
              fi

              # Turn on web server
              systemctl enable httpd # Enables the Apache web server
service to start automatically on boot.
              systemctl start httpd # Starts the Apache web server
service immediately.
            - ElasticFileSystem: !Ref WPElasticFileSystemID #
Substitutes the value of the WPElasticFileSystemID parameter.
              ALBDNSName: !Ref ALBDnsName # Substitutes the value of
the ALBDnsName parameter.
        ImageId: !Ref LatestAL2023AmiId # Specifies the AMI ID to be
used for launching instances, referencing the LatestAL2023AmiId
parameter.
        SecurityGroupIds: # Specifies the security groups to associate
with the launched EC2 instances.
          - !Ref WebTierSecurityGroup # References the
WebTierSecurityGroup resource defined in this template.
        InstanceType: !Ref EC2ServerInstanceType # Specifies the EC2
instance type to be used, referencing the EC2ServerInstanceType
parameter.

Outputs: # Defines the output values of the CloudFormation stack.
  WPLaunchTemplate: # Defines an output named WPLaunchTemplate.
    Description: 'Launch Template for WordPress' # A description for
the output.
    Value: # The value of the output.
      !Ref LabLaunchTemplate # The physical ID of the LabLaunchTemplate
resource.
```

Task 5.3: Create the CloudFormation stack

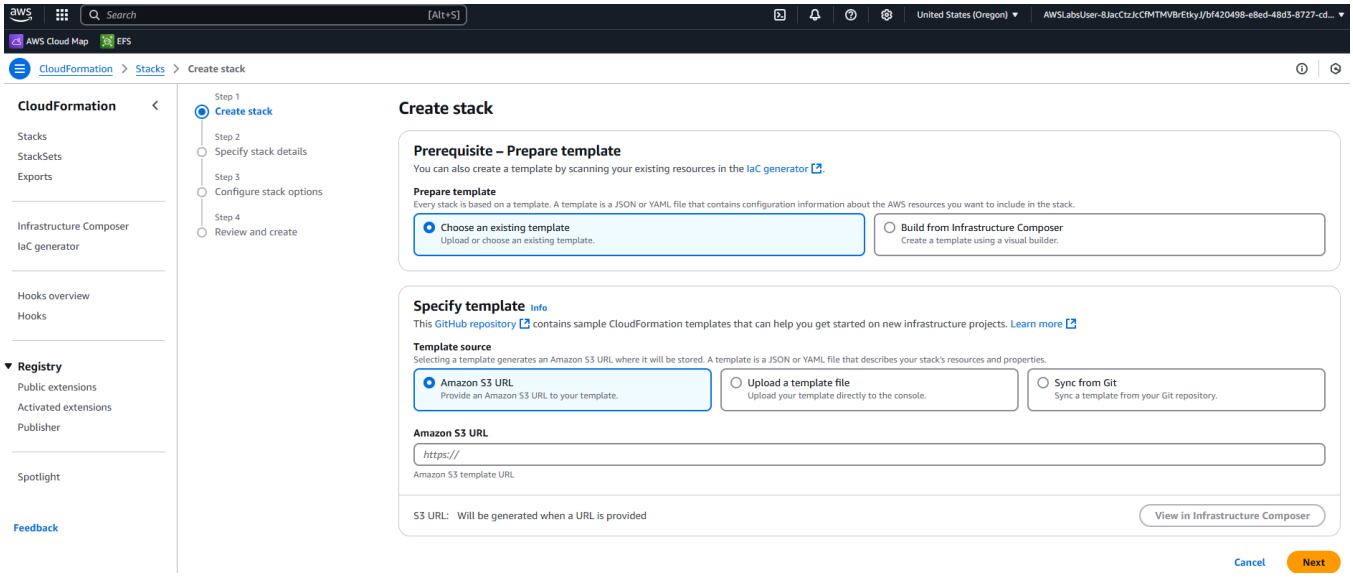
- Choose **Create stack**.

Note: If the console starts you on the Stacks page instead of the Amazon CloudFormation landing page, then you can get to the Create stack page in two steps.

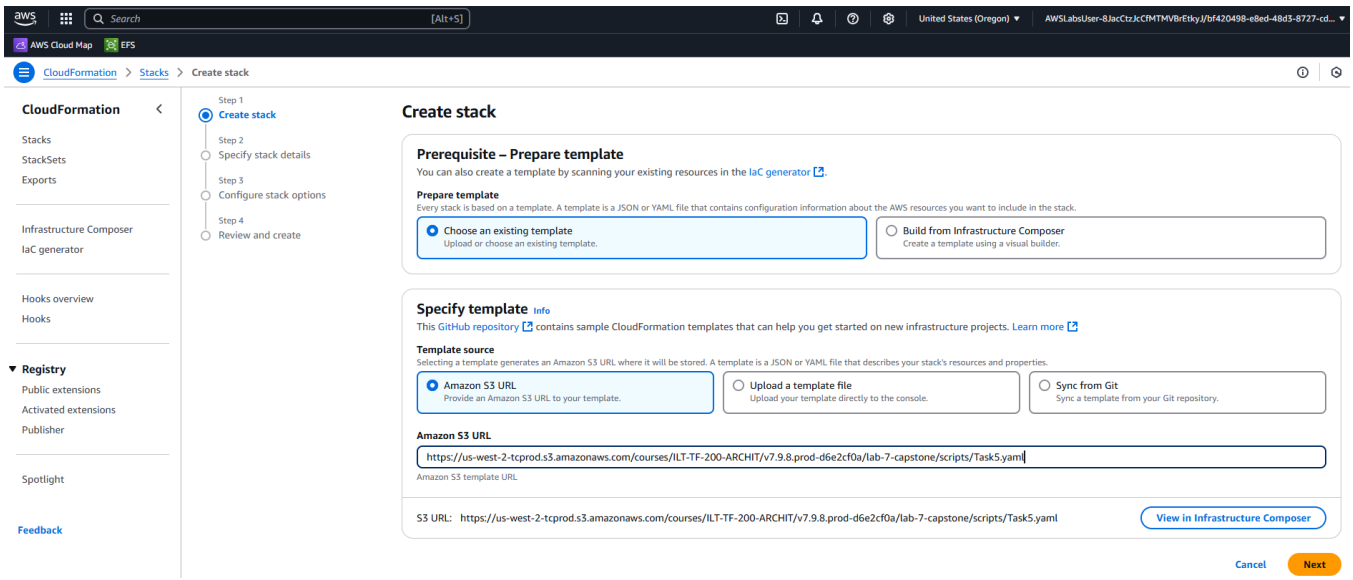
- Choose the **Create stack** dropdown menu.
- Choose **With new resources (standard)**.



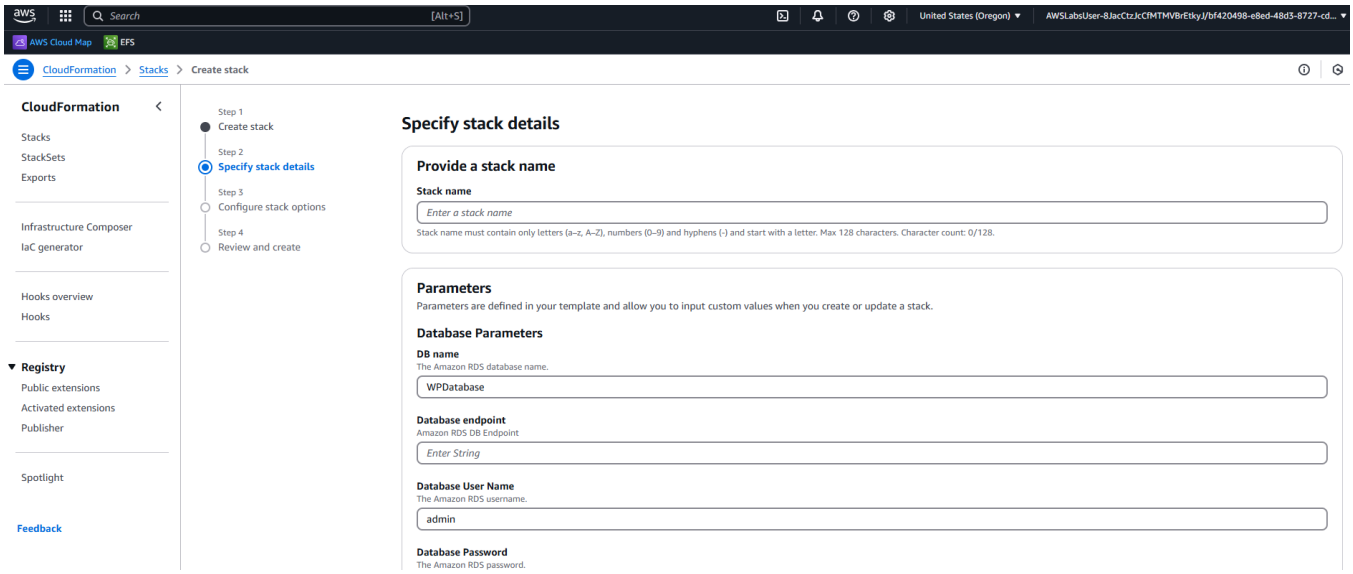
The **Create stack** page is displayed.



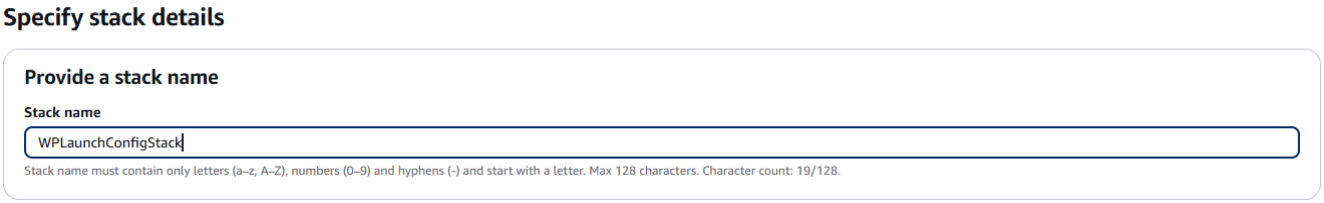
- Configure the following:
  - Select Choose an existing template.
  - Select Amazon S3 URL.
  - Copy the **Task5TemplateUrl** value **(**https://us-west-2-tcprod.s3.amazonaws.com/courses/ILT-TF-200-ARCHIT/v7.9.8.prod-d6e2cf0a/lab-7-capstone/scripts/Task5.yaml**)** and paste it in the **Amazon S3 URL** text box.
  - Choose **Next**.



The **Specify stack details** page is displayed.



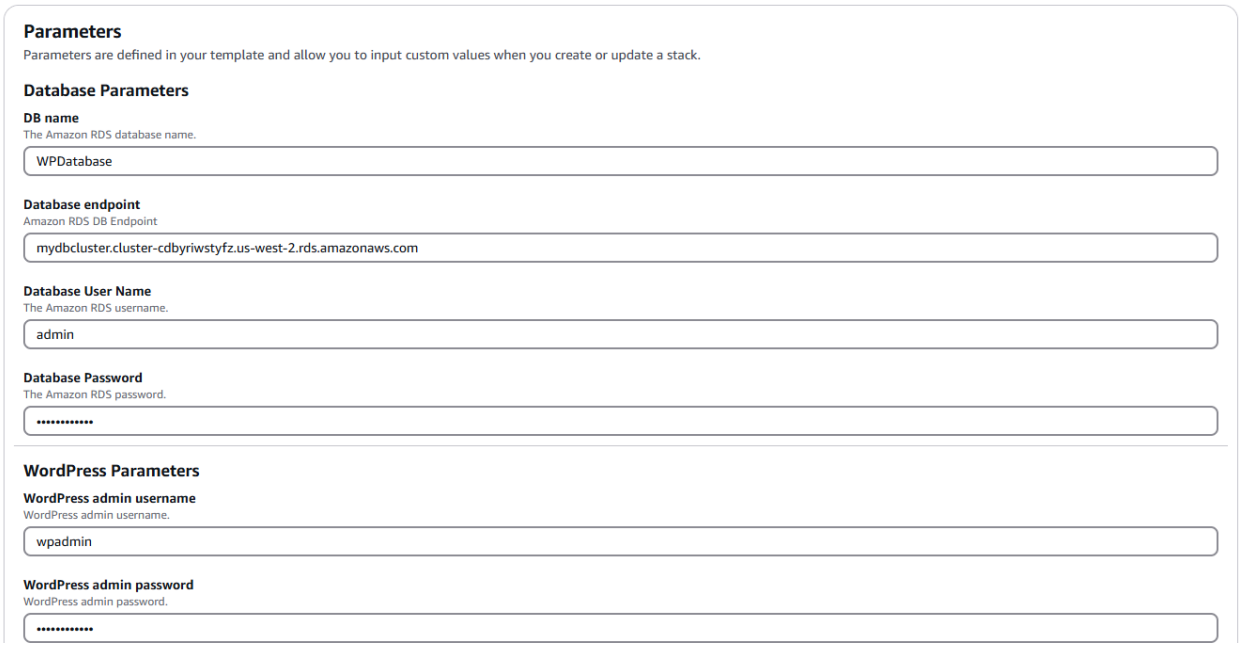
- Set the **Stack name** as `WPLaunchConfigStack` .



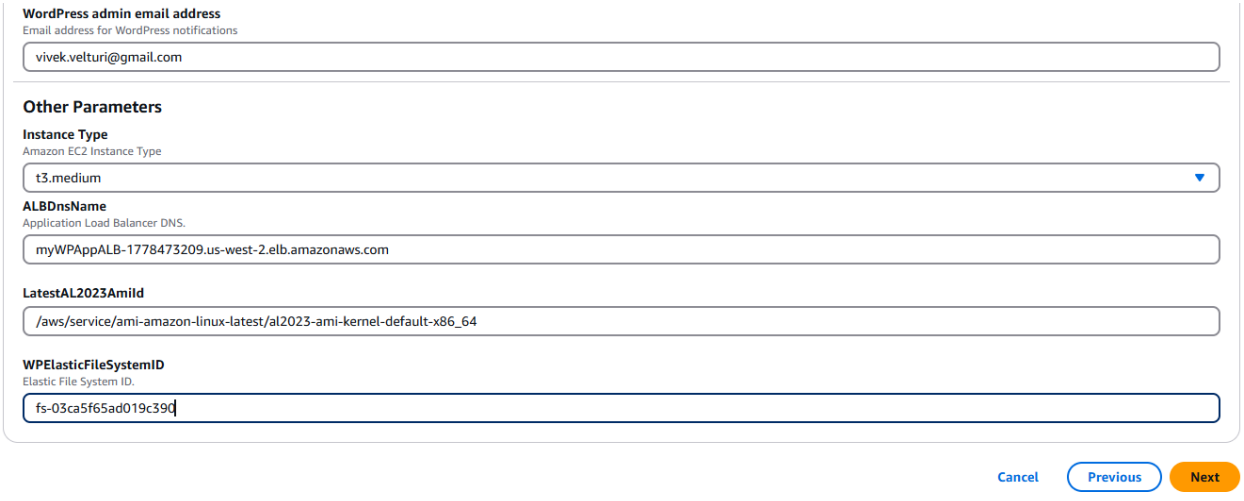
- Configure the following **parameters**:
    - **DB name**: Paste the **initial database name** you copied in Task 2 (WPDatabase)

      Note: Make sure that you paste the *initial database name*, not the cluster name.

    - **Database endpoint**: Paste the **writer endpoint** you copied in Task 2. (mydbcluster.cluster-cdbyriwstyfz.us-west-2.rds.amazonaws.com)
    - **Database User Name**: Paste the **Master username** you copied in Task 2. (admin)
    - **Database Password**: Paste the **Master password** you copied in Task 2. (1QaDM4T5Cxny)
    - **WordPress admin username**: Defaults to `wpadmin`.
    - **WordPress admin password**: Paste the **LabPassword** value (1QaDM4T5Cxny)

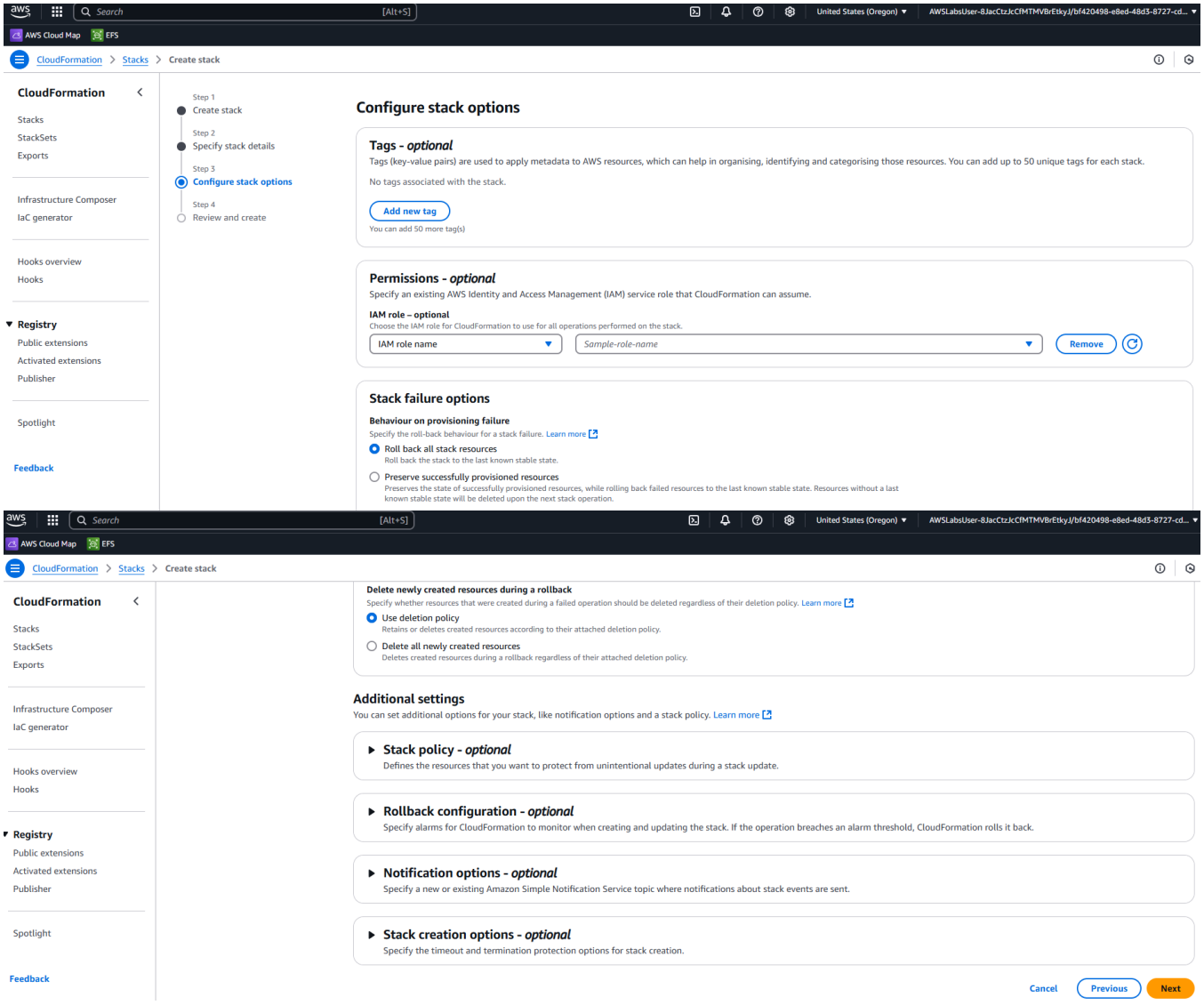


- **WordPress admin email address**: Input a valid email address. (email of your choice)
- **Instance Type**: Leave the default value of **t3.medium**.
- **ALBDnsName**: Paste the **DNS name** value you copied in Task 4. (myWPAppALB-1778473209.us-west-2.elb.amazonaws.com)
- **LatestAL2023AmiId**: Leave the default value.
- **WPElasticFileSystemID**: Paste the **File system ID** value you copied in Task 3. (fs-03ca5f65ad019c390)



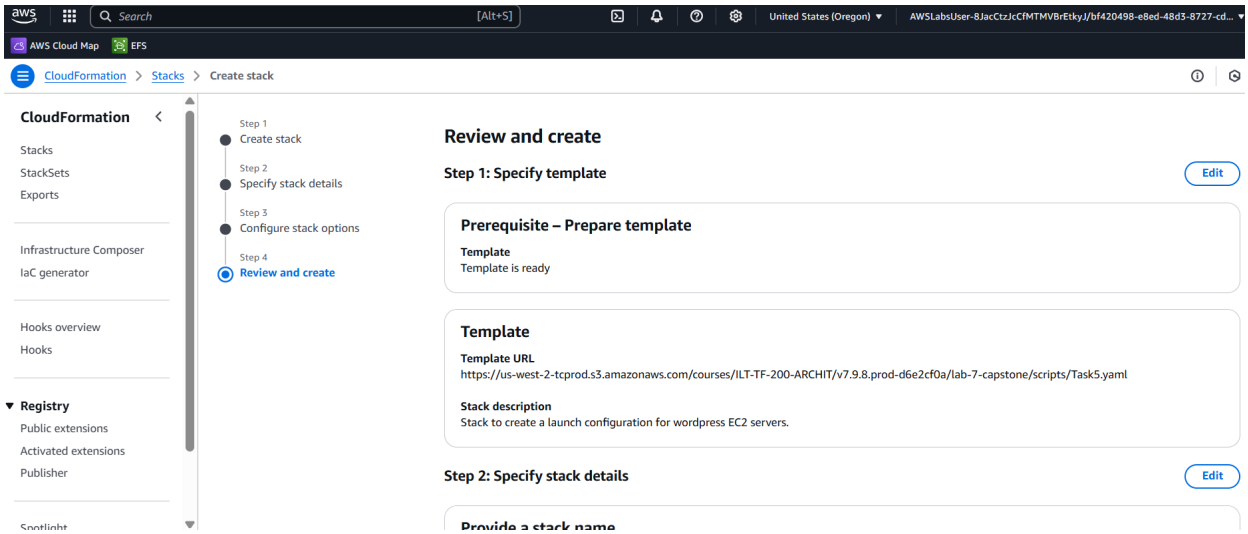
- Choose **Next**.

The **Configure stack options** page is displayed. You can use this page to specify additional parameters. You can browse the page, but leave settings at their default values.

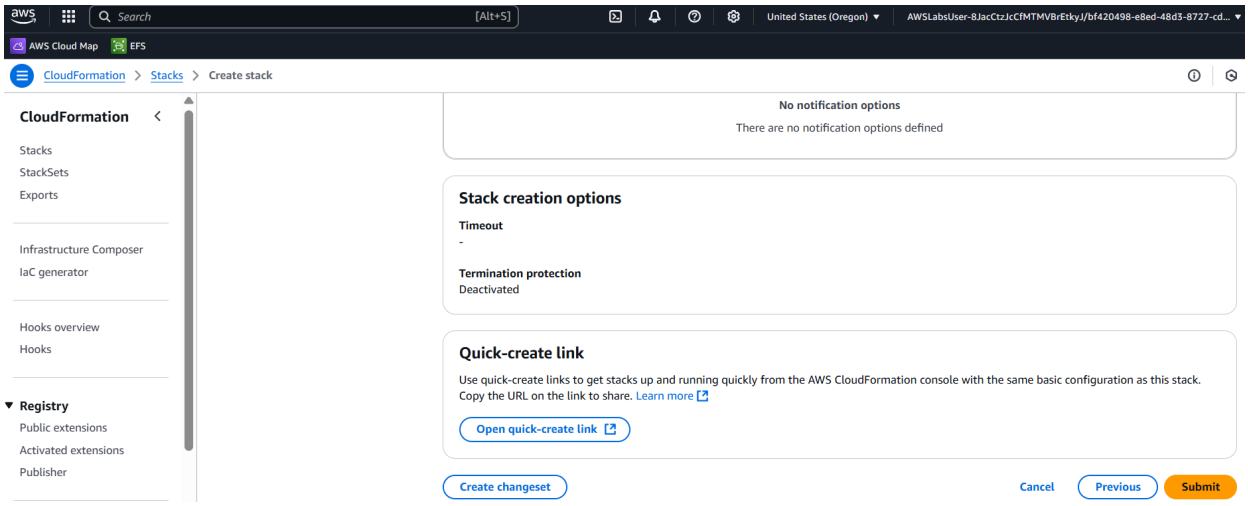


- Choose **Next**.

The **Review and create** page is displayed. This page is a summary of all settings.

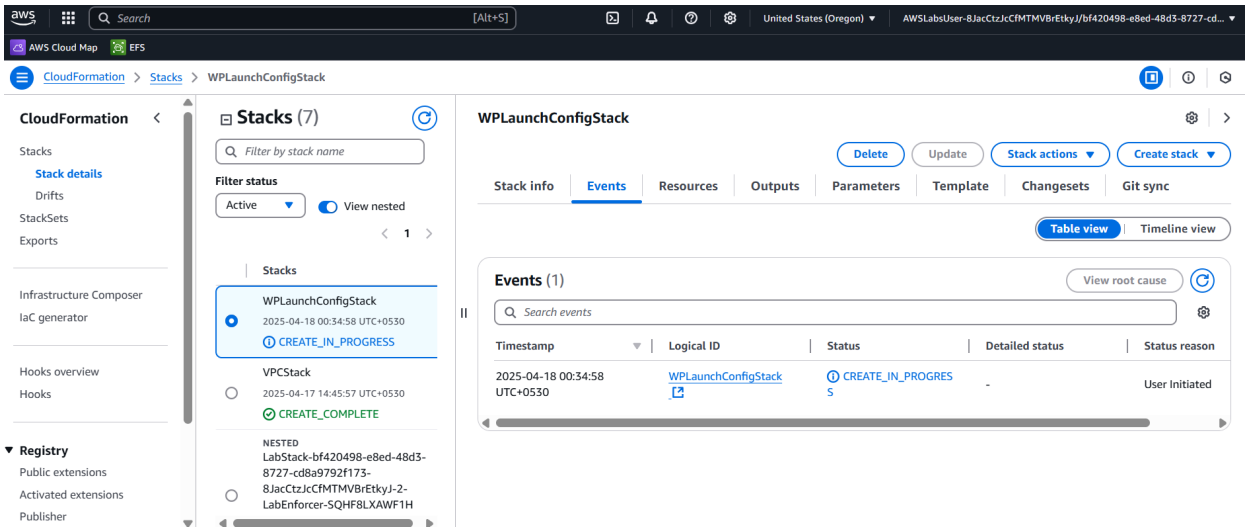


- Scroll to the bottom of the page and choose **Submit**.

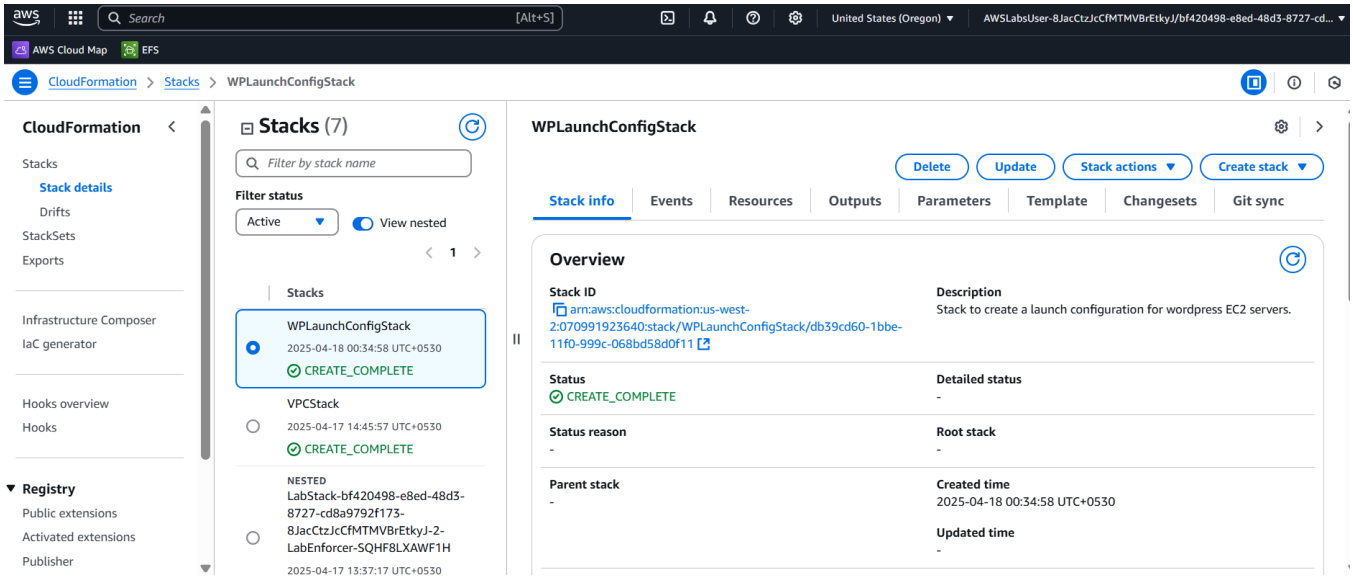


The **stack details** page is displayed.

The stack enters the CREATE_IN_PROGRESS status.



- Choose the **Stack info** tab.
- Occasionally choose the **Overview** refresh .
- Wait for the stack status to change to CREATE_COMPLETE.

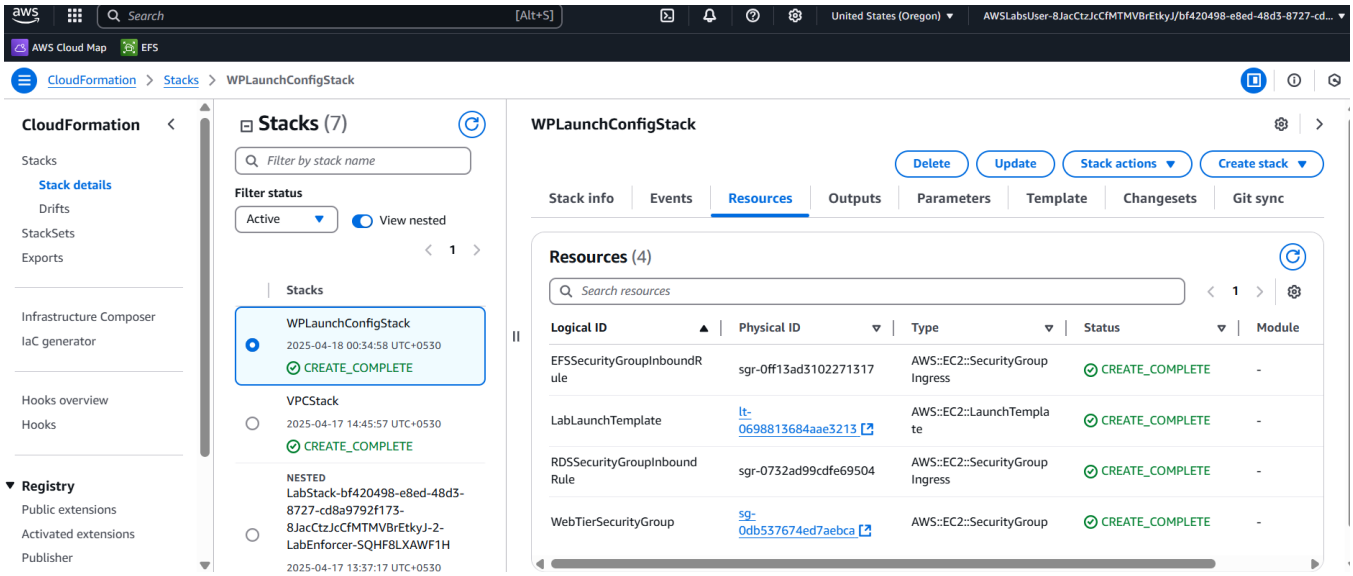


Note: This stack can take up to 5 minutes to deploy the resources.

Task 5.4: View created resources from the console

- Choose the **Resources** tab.

The list shows the resources that are created.



You have created the stack using the provided CloudFormation template.

Task 6: Create the application servers by configuring an Auto Scaling group and a scaling policy

In this task, you create the WordPress application servers by configuring an Auto Scaling group and a scaling policy.

Task 6: Scenario

Now that your template has been deployed, it's time to create the application servers and auto scaling mechanism. In the previous task, you chose to implement auto scaling for the app servers to meet the scaling requirements of the project plan.

Create an Auto Scaling group and scaling policy. Verify that the instance status is healthy, and test the load balancer availability. Hand the environment over to the engineering teams to validate full functionality when the unit test is complete, and ask them for feedback.

When the team is satisfied, ask them to migrate the WordPress site to the AWS environment and test the app functionality. A common practice is to introduce examples of failure. If the app is working as intended, you would introduce some examples of failure, for example, delete an app server or roll back the database to a recent backup. This step is not a part of this lab.

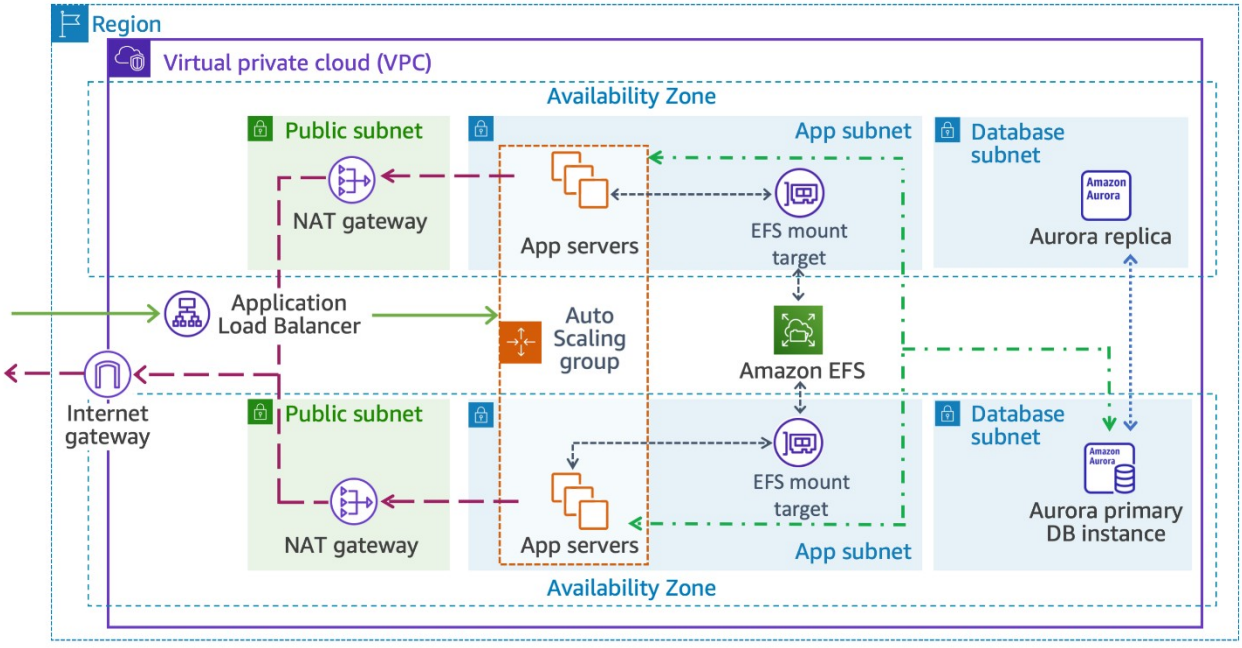


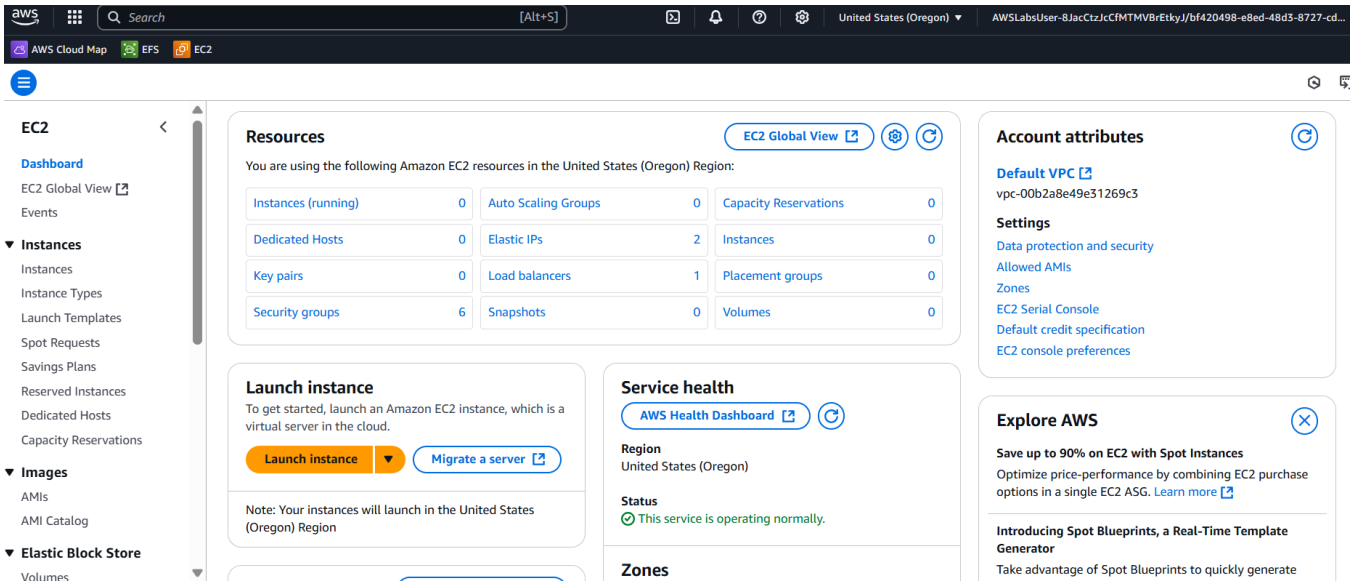
Image description: Preceding image depicts the entire architecture which will be created by the end of this lab. We have Amazon VPC to facilitate public and private networking using Public Subnets and Private Subnets. To ensure we are able to accept incoming internet traffic and allow outgoing traffic to the internet, we have

Internet Gateway and NAT Gateways. In-order to facilitate high-availability (HA) for our architecture we have Application Load Balancer (ALB) and Auto-Scaling Group (ASG). There is also a fully scalable and elastic file server using Amazon EFS which can work across multiple availability zones (AZ)s. For database we are using Amazon Aurora which works across multiple availability zones (AZ)s with primary in one availability zone (AZ) and a read-replica in another availability zone (AZ).

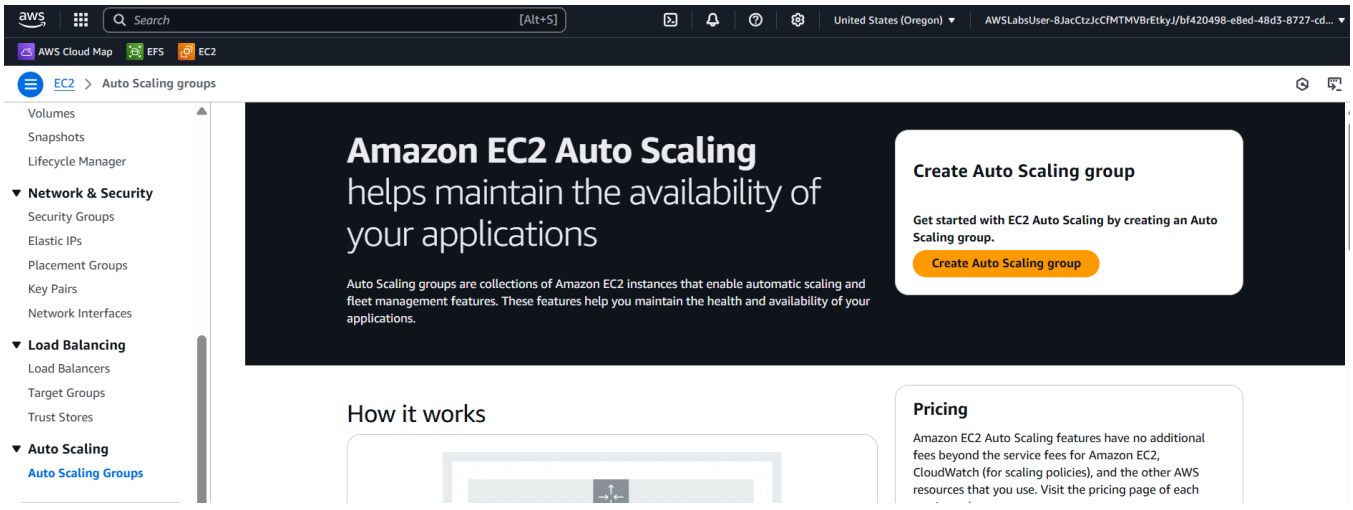
Task 6 instructions: Create the application servers by configuring an Auto Scaling group and a scaling policy

Task 6.1: Create an Auto Scaling group

- At the top of the AWS Management Console, in the search box, search for and choose EC2.

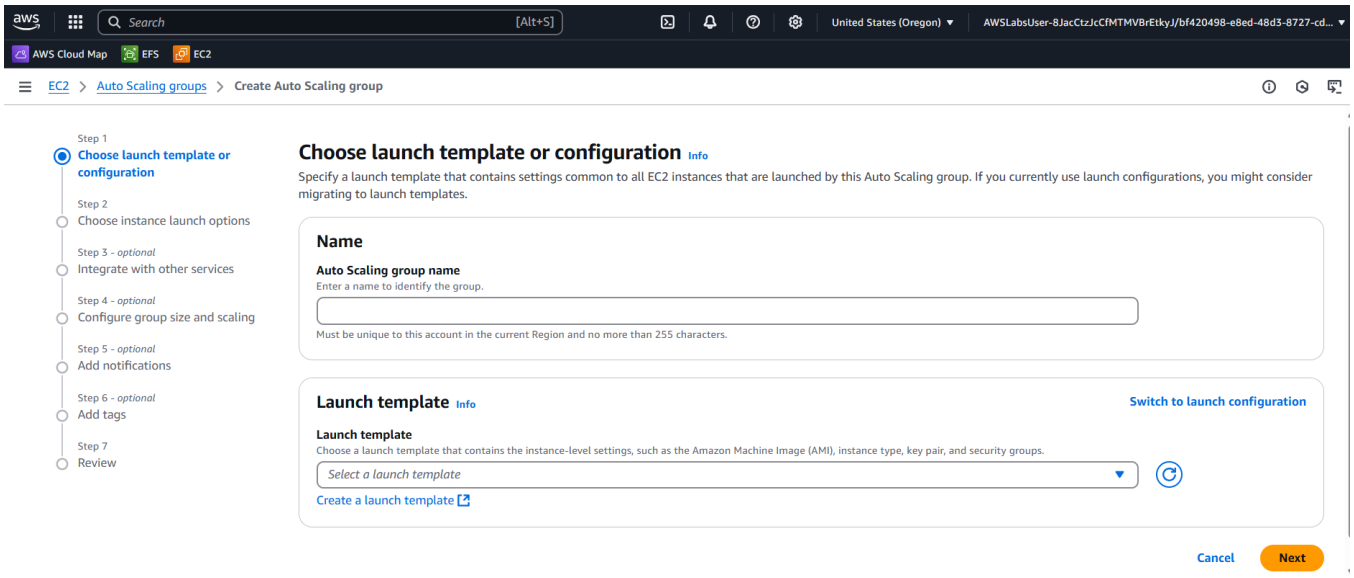


- In the left navigation pane, under the **Auto Scaling** section, choose **Auto Scaling Groups**.

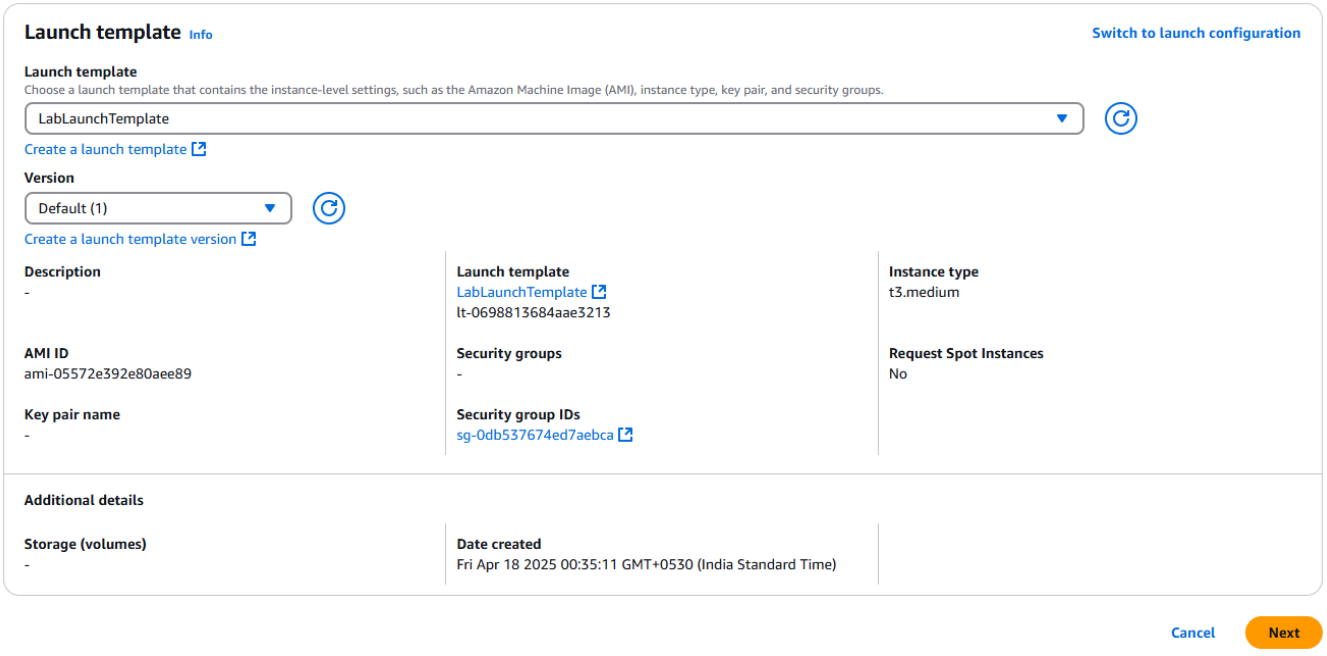


- Choose **Create Auto Scaling group**.

The **Choose launch template or configuration** page is displayed.

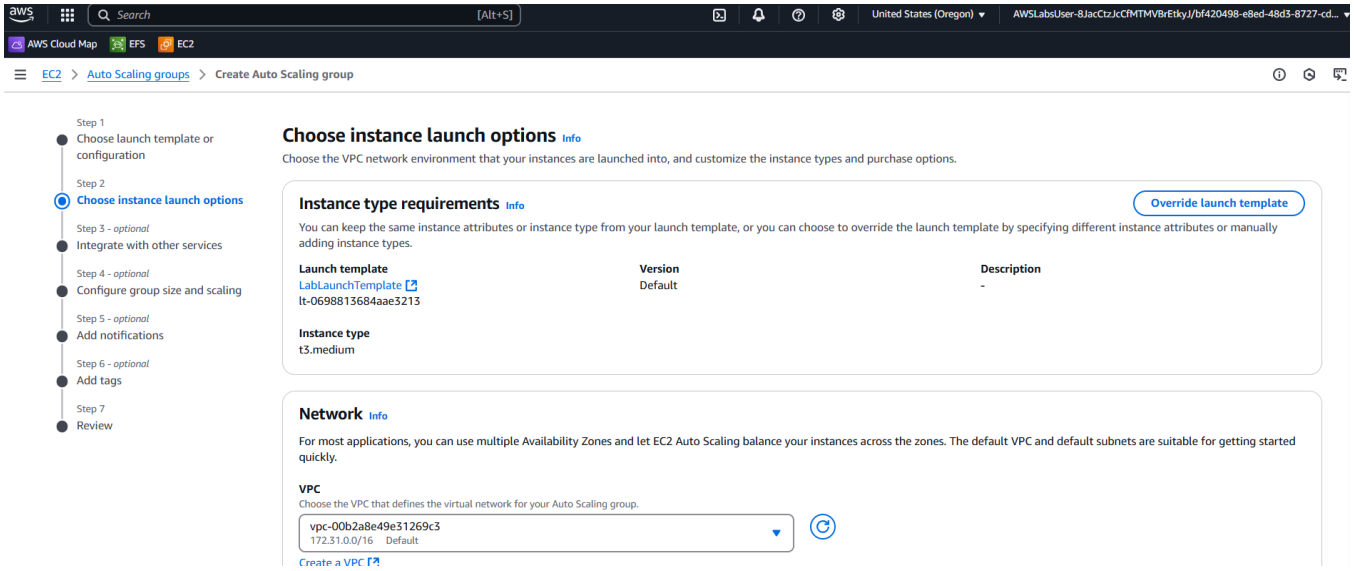


- Configure the following:
  - **Auto Scaling group name**: Enter `WP-ASG`.
  - **Launch template**: Select the launch template that you created in Task 5. (LabLaunchTemplate)

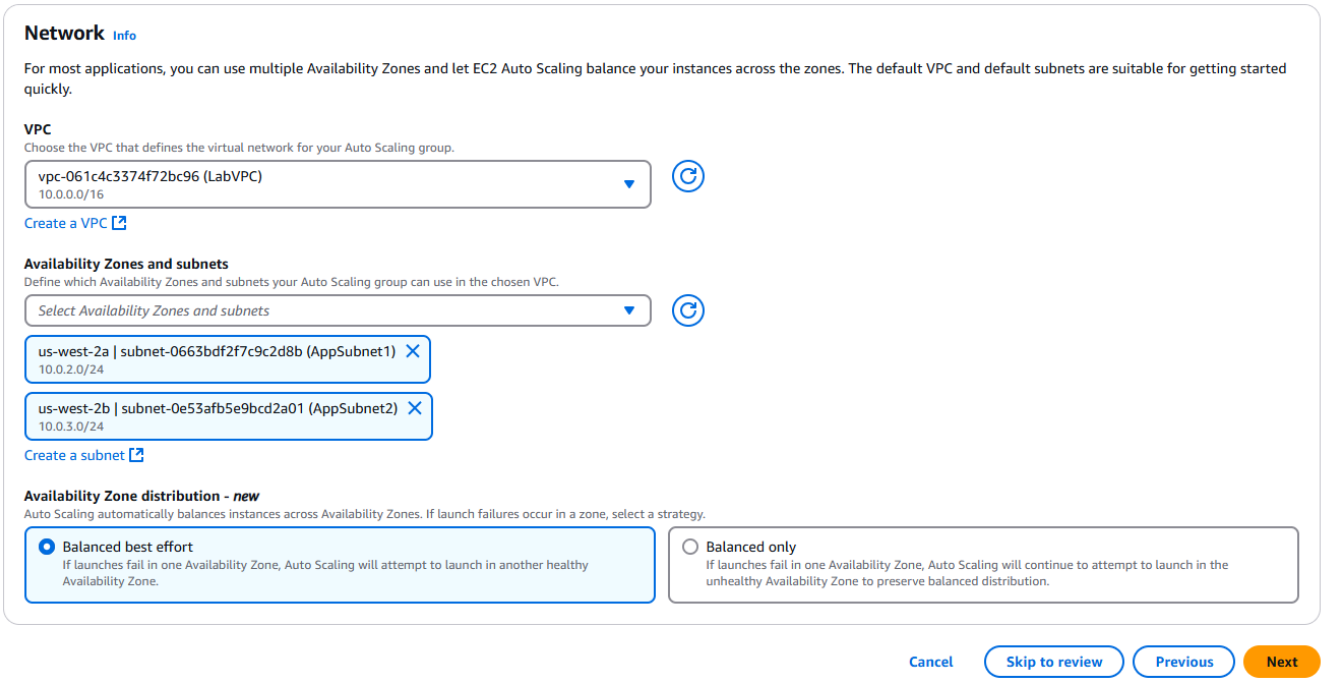


- Choose **Next**.

The **Choose instance launch options** page is displayed.

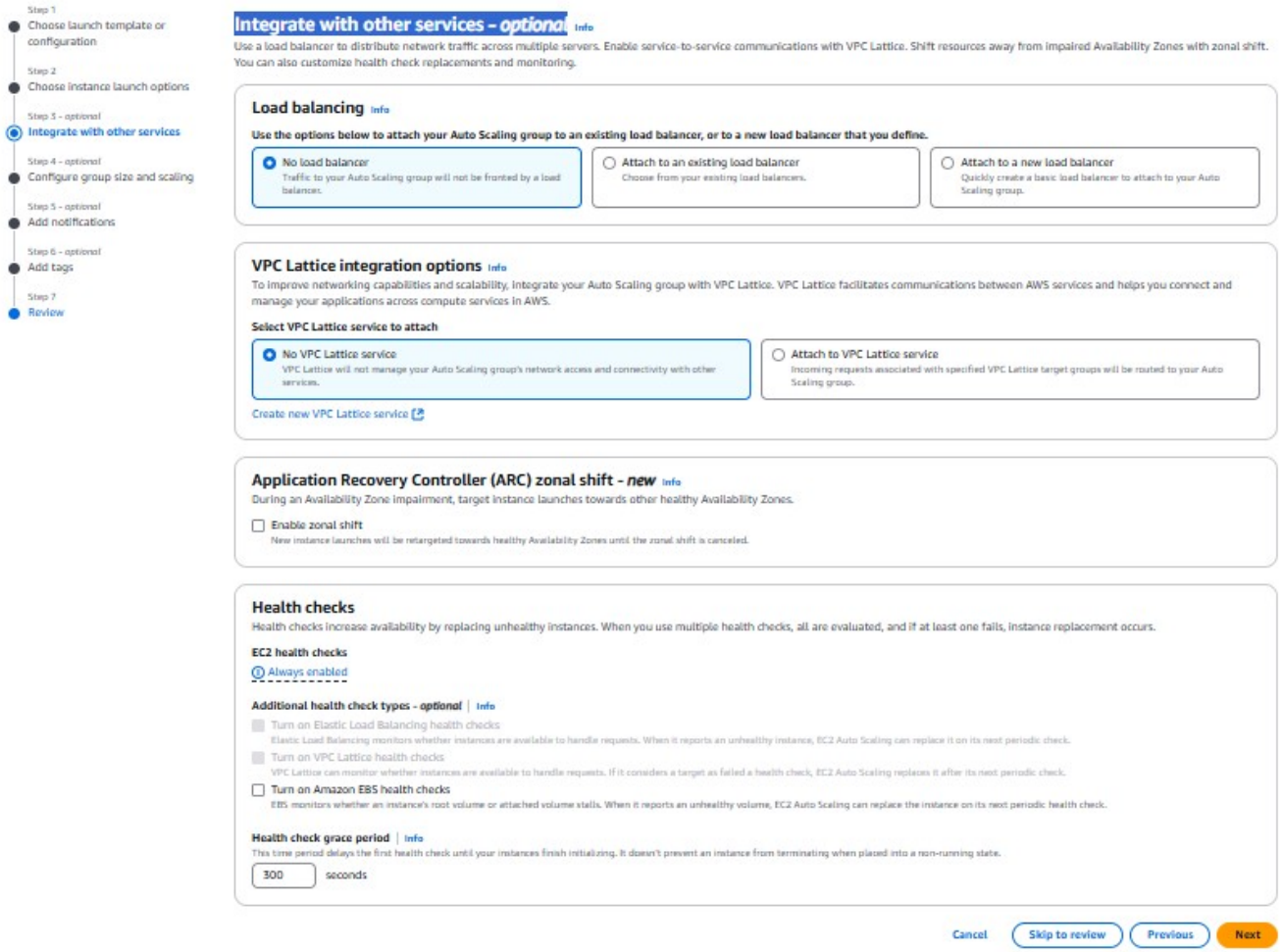


- In the **Network** section, configure the following:
  - **VPC**: Select **LabVPC** from the dropdown menu.
  - **Availability Zones and subnets**: Select **AppSubnet1** and **AppSubnet2** from the dropdown menu.

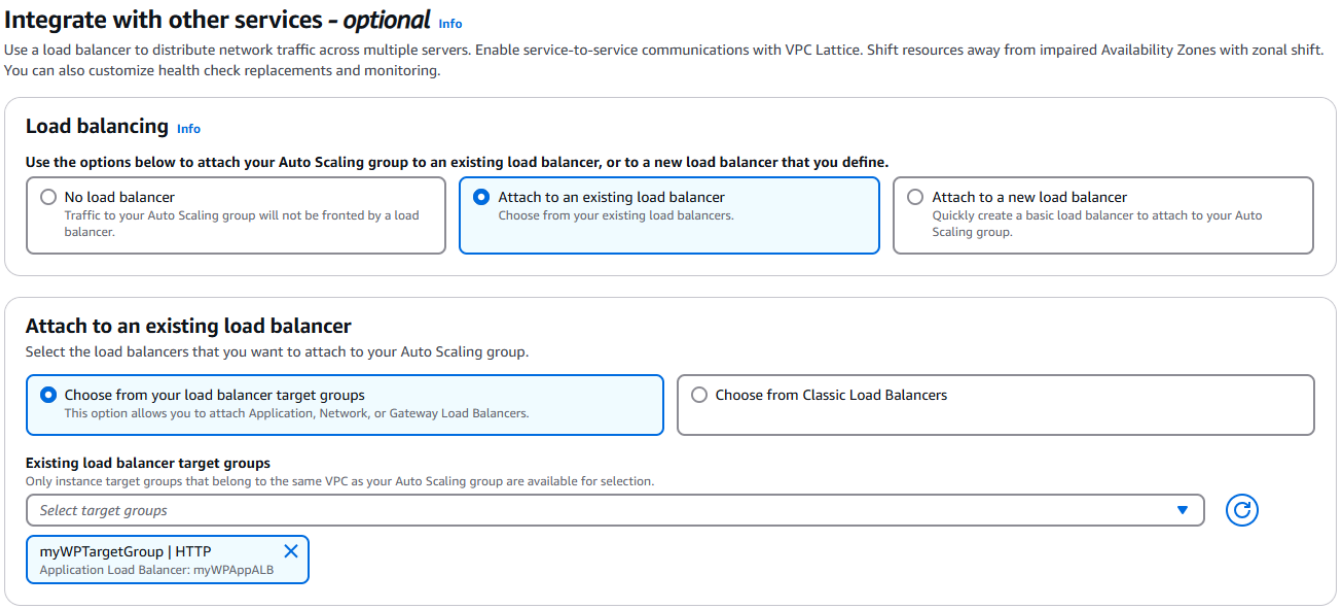


- Choose **Next**.

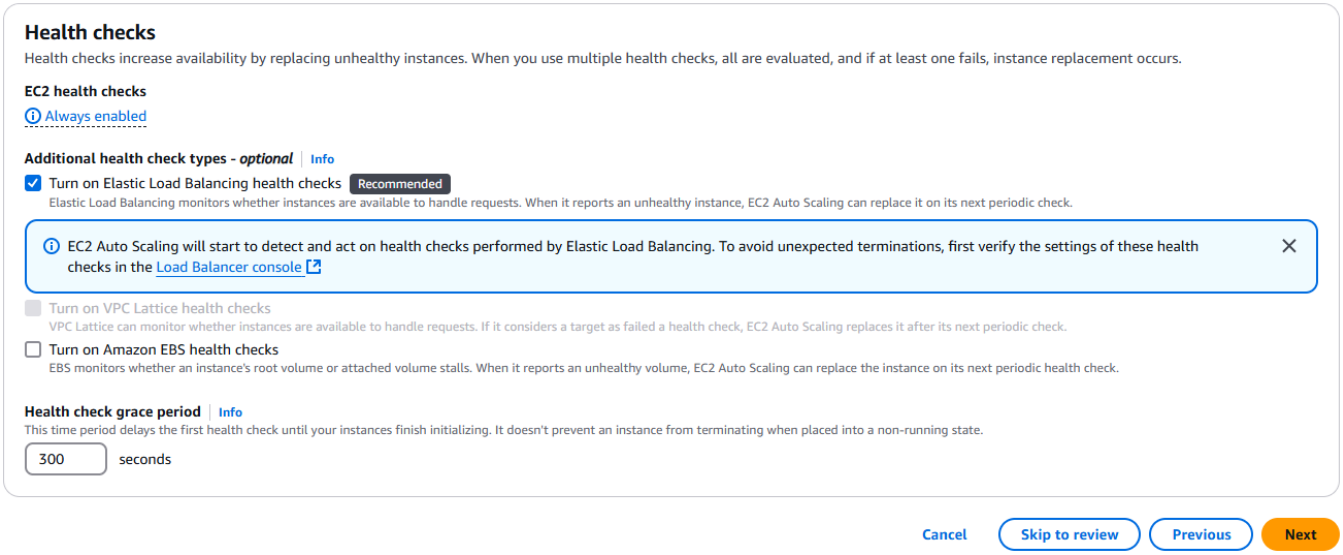
The ***Integrate with other services - optional*** page is displayed.



- On the **Integrate with other services - *optional*** page, configure the following:
  - Select **Attach to an existing load balancer**.
  - Select **Choose from your load balancer target groups**.
  - From the **Existing load balancer target groups** dropdown menu, select **myWPTargetGroup | HTTP**.

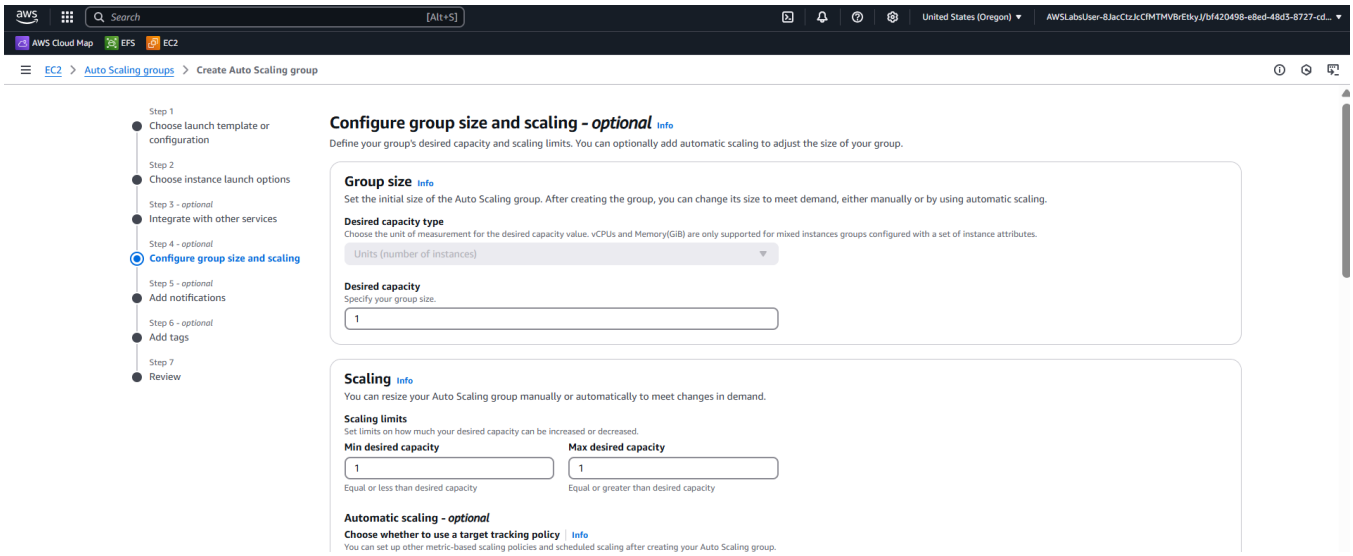


- For **Additional health check types - *optional***: Select **Turn on Elastic Load Balancing health checks**.
- **Health check grace period**: Leave at the default value of 300 or more.



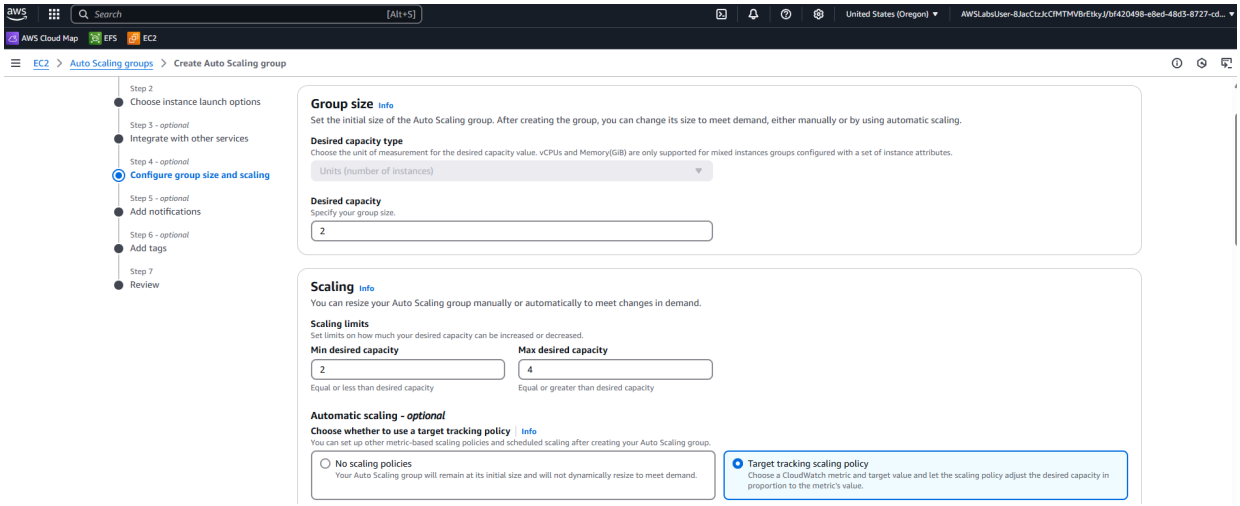
- Choose **Next**.

The **Configure group size and scaling - *optional*** page is displayed.

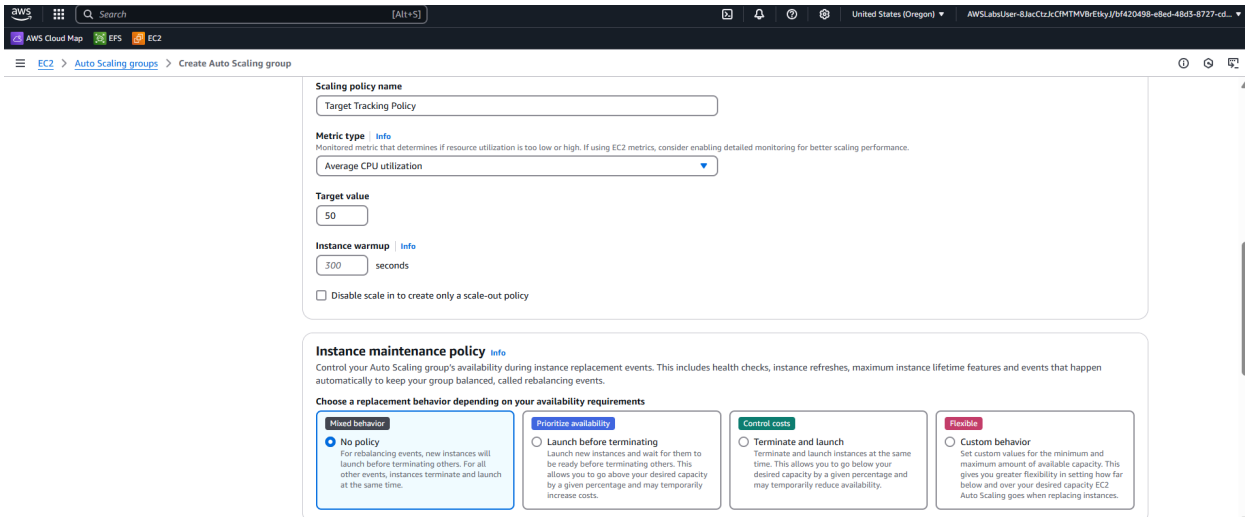


- On the **Configure group size and scaling - *optional*** page, configure the following:
  - In the **Group size** section:
    - **Desired capacity:** Enter 2 .
  - In the **Scaling** section:
    - **Min desired capacity:** Enter 2
    - **Max desired capacity:** Enter 4

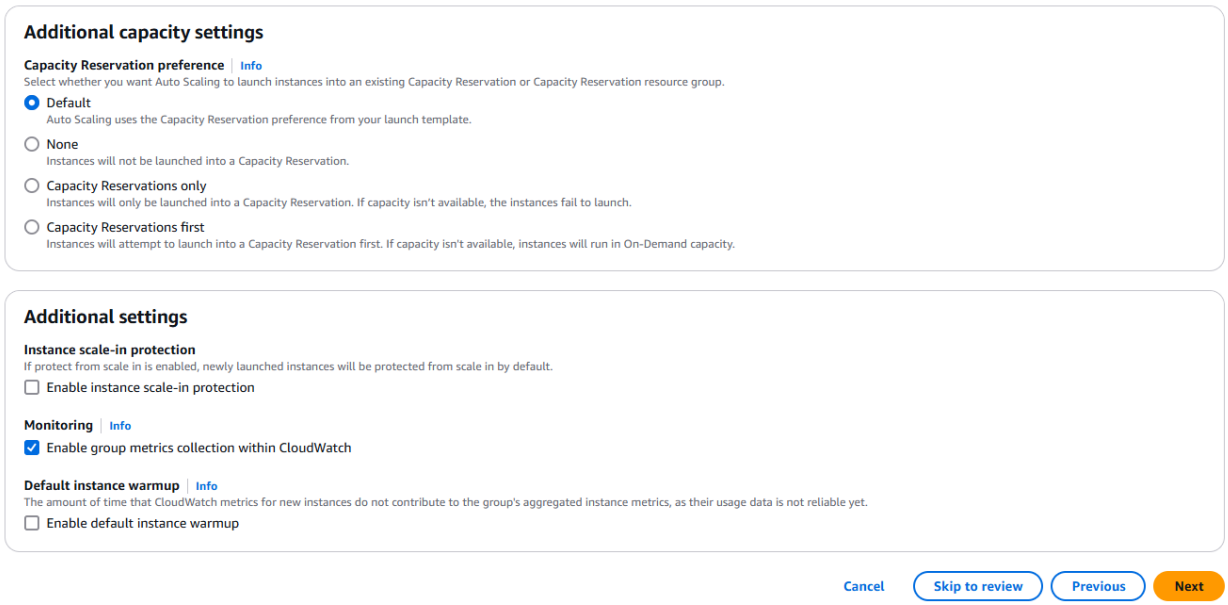
- In the **Automatic scaling - *optional*** section, configure the following:
  - Select **Target tracking scaling policy**.



The remaining settings on this section can be left at their default values.

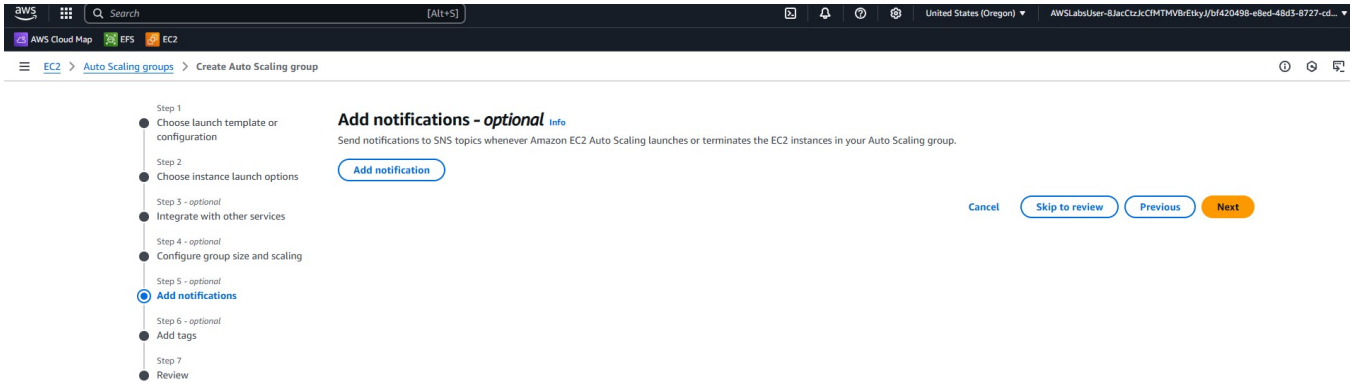


- **Monitoring**: Select **Enable group metrics collection within CloudWatch**.



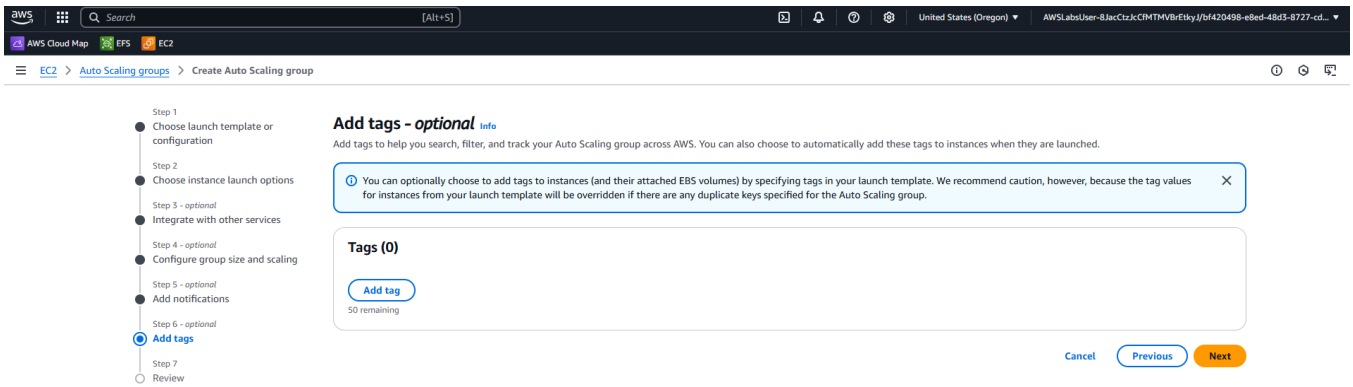
- Choose **Next**.

The **Add notifications - *optional*** page is displayed.

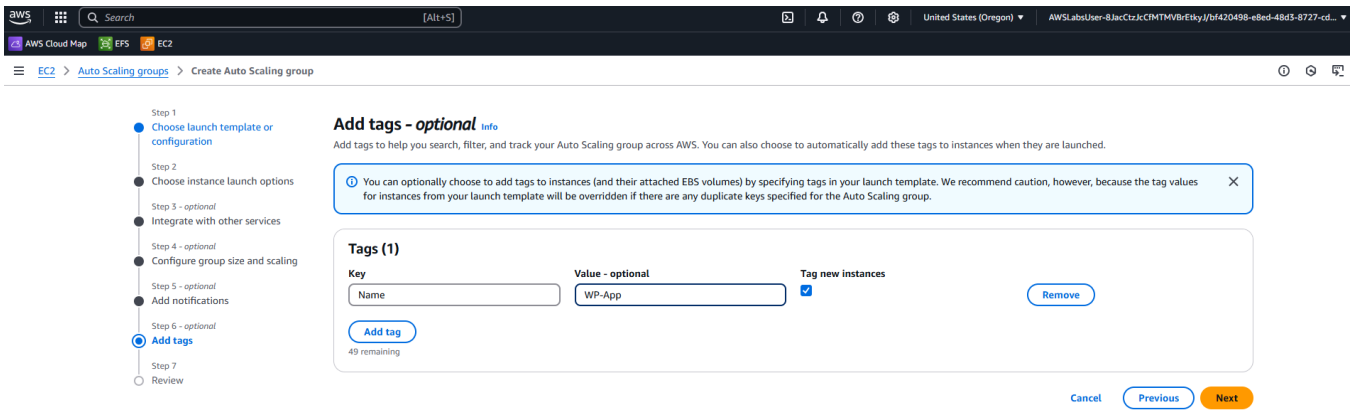


- Choose **Next**.

The **Add tags - *optional*** page is displayed.

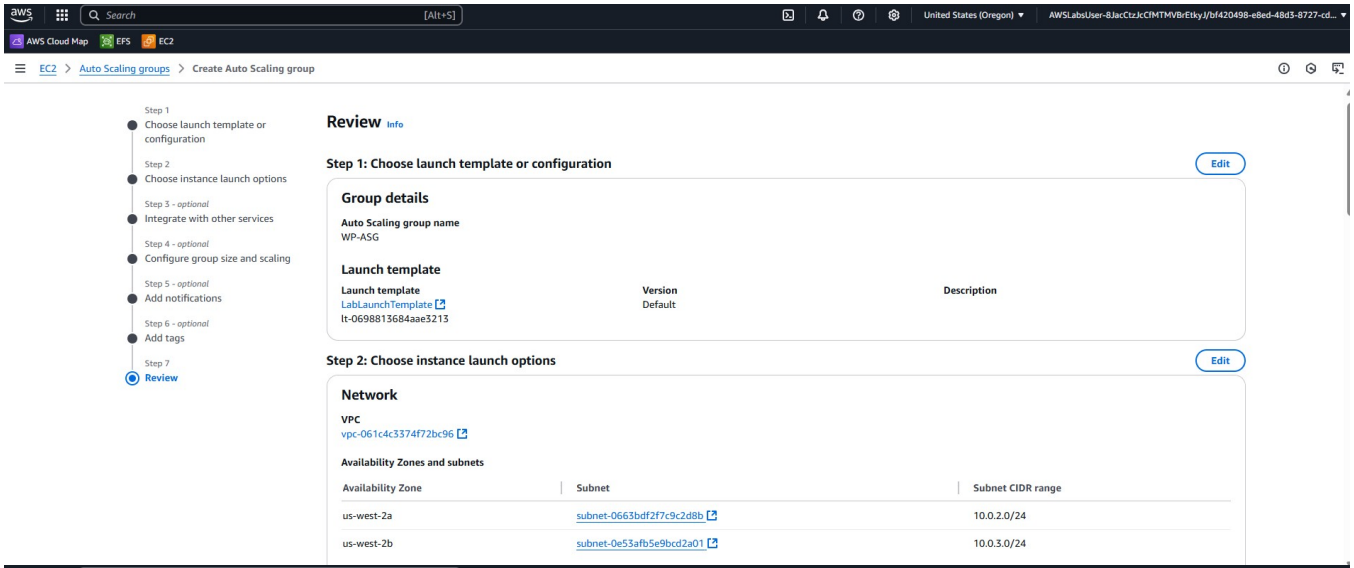


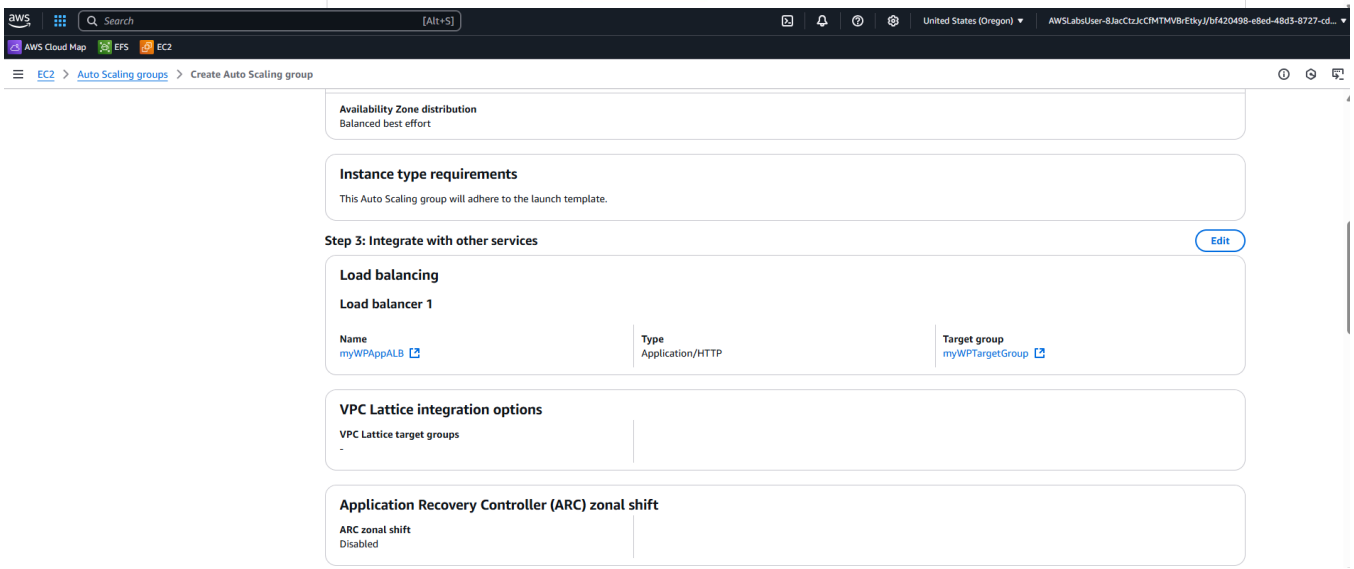
- Choose **Add tag** and then configure the following:
  - **Key**: Enter `Name`.
  - **Value - optional**: Enter `WP-App`.

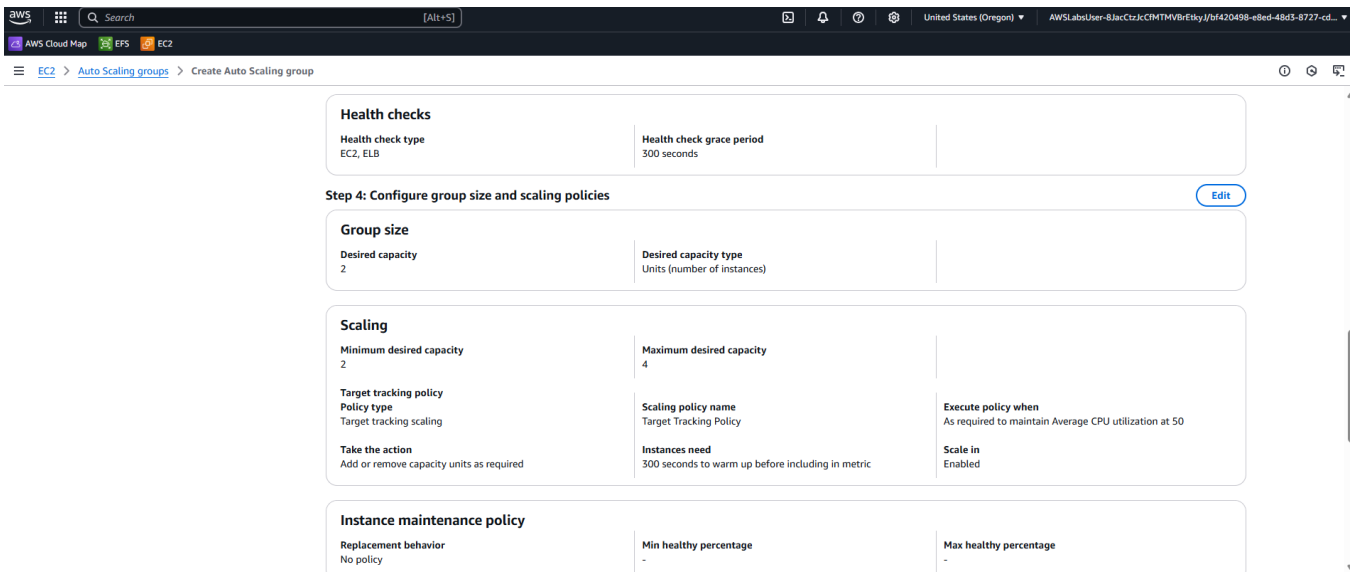


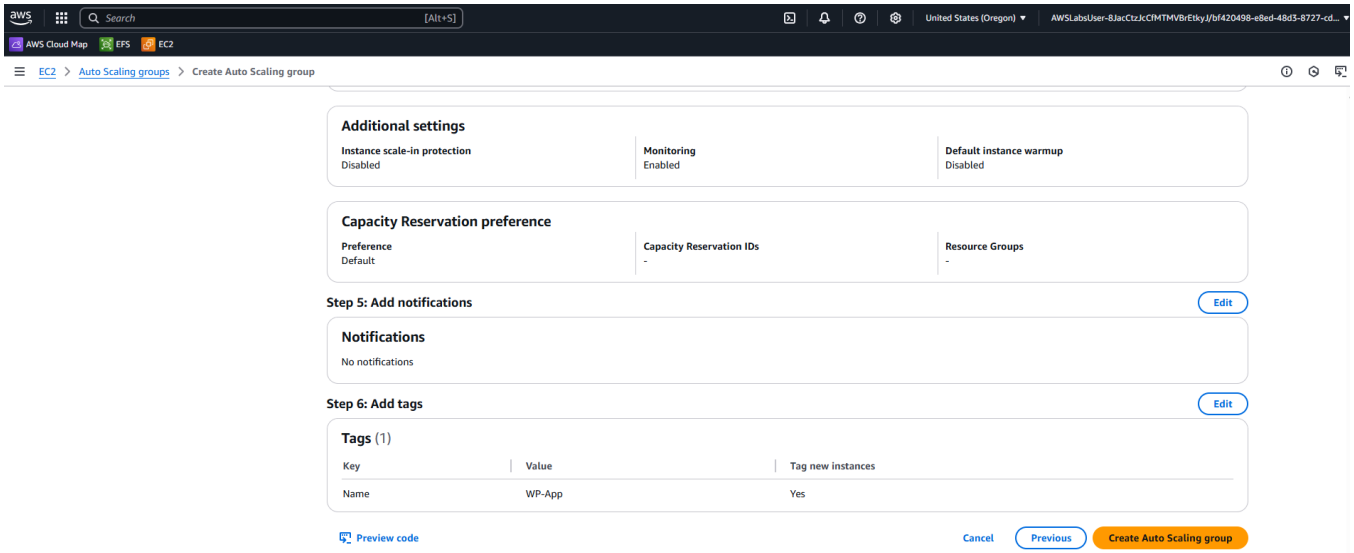
- Choose **Next**.

The **Review** page is displayed.


Search [Alt+S]
United States (Oregon)
AWS Cloud Map
EFS
EC2
EC2 > Auto Scaling groups > Create Auto Scaling group
Step 1
Choose launch template or configuration
Step 2
Choose instance launch options
Step 3 - optional
Integrate with other services
Step 4 - optional
Configure group size and scaling
Step 5 - optional
Add notifications
Step 6 - optional
Add tags
Step 7
Review
Review Info
Step 1: Choose launch template or configuration
Edit
Group details
Auto Scaling group name
WP-ASG
Launch template
Launch template
LabLaunchTemplate
lt-0698813684aae3213
Version
Default
Description
Step 2: Choose instance launch options
Edit
Network
VPC
vpc-061c4c3374f72bc96
Availability Zones and subnets
Availability Zone
Subnet
Subnet CIDR range
us-west-2a
subnet-0663bdf2f7c9c2d8b
10.0.2.0/24
us-west-2b
subnet-0e53afb5e9bcd2a01
10.0.3.0/24

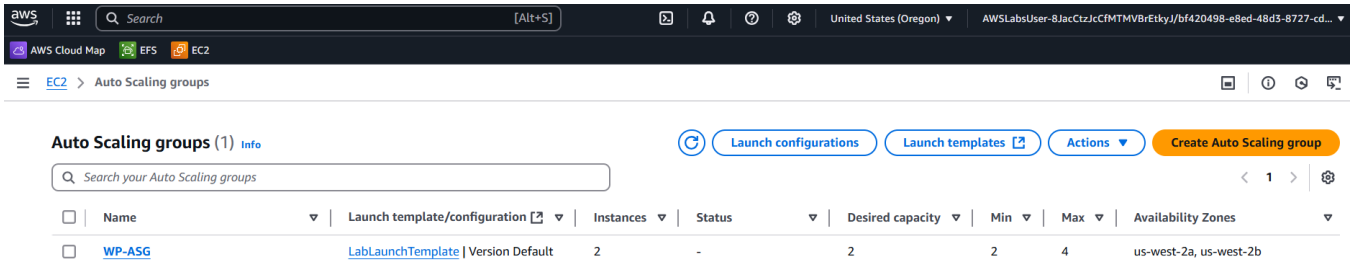

Search [Alt+S]
United States (Oregon)
AWS Cloud Map
EFS
EC2
EC2 > Auto Scaling groups > Create Auto Scaling group
Availability Zone distribution
Balanced best effort
Instance type requirements
This Auto Scaling group will adhere to the launch template.
Step 3: Integrate with other services
Edit
Load balancing
Load balancer 1
Name
myWPAppALB
Type
Application/HTTP
Target group
myWPTargetGroup
VPC Lattice integration options
VPC Lattice target groups
-
Application Recovery Controller (ARC) zonal shift
ARC zonal shift
Disabled


Search [Alt+S]
United States (Oregon)
AWS Cloud Map
EFS
EC2
EC2 > Auto Scaling groups > Create Auto Scaling group
Health checks
Health check type
EC2, ELB
Health check grace period
300 seconds
Step 4: Configure group size and scaling policies
Edit
Group size
Desired capacity
2
Desired capacity type
Units (number of instances)
Scaling
Minimum desired capacity
2
Maximum desired capacity
4
Target tracking policy
Policy type
Target tracking scaling
Scaling policy name
Target Tracking Policy
Execute policy when
As required to maintain Average CPU utilization at 50
Take the action
Add or remove capacity units as required
Instances need
300 seconds to warm up before including in metric
Scale in
Enabled
Instance maintenance policy
Replacement behavior
No policy
Min healthy percentage
-
Max healthy percentage
-



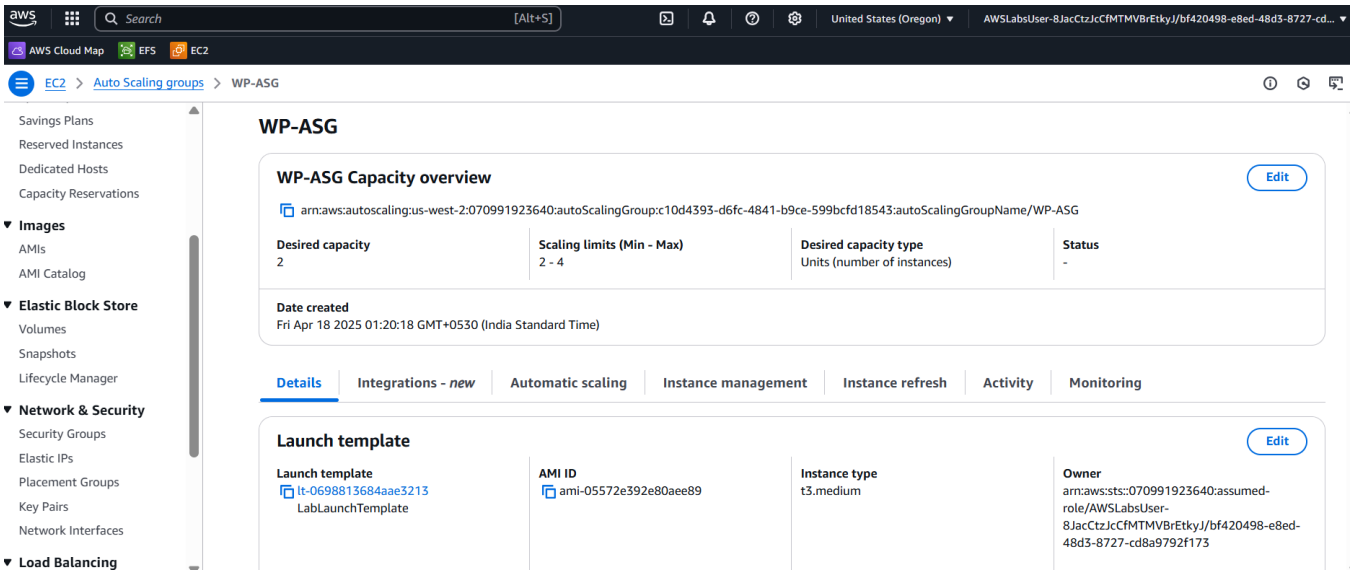
- Review the Auto Scaling group configuration for accuracy, and then at the bottom of the page, choose **Create Auto Scaling group**.

The **Auto Scaling groups** page is displayed.



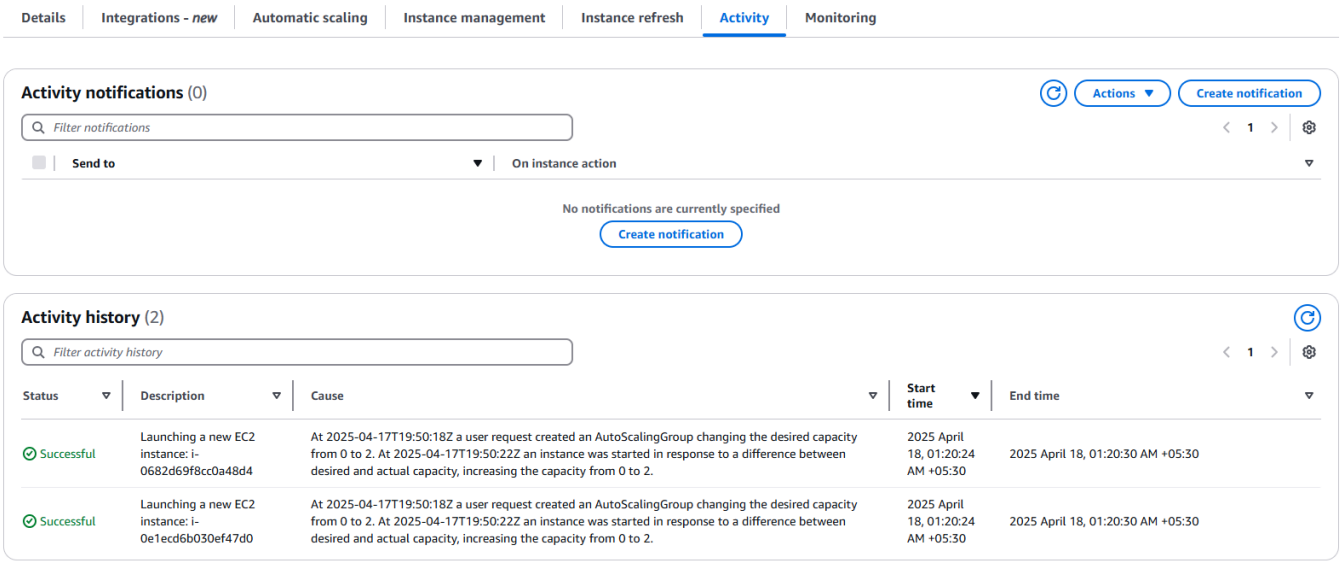
Now that you have created your Auto Scaling group, you can verify that the group has launched your EC2 instances.

- Choose the Auto Scaling group **WP-ASG** link.



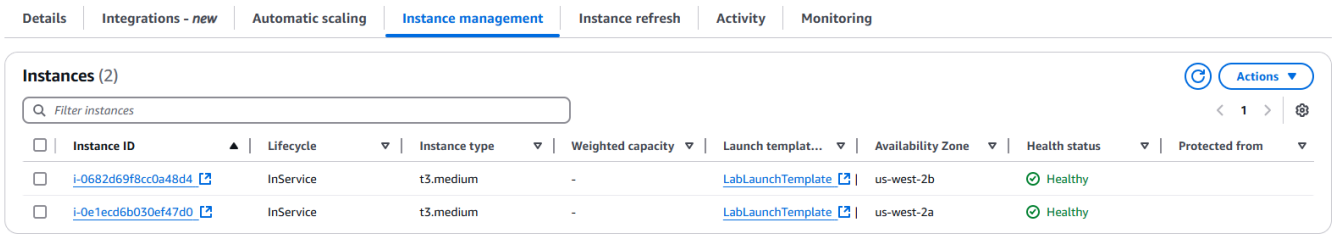
- Choose the **Activity** tab.

The **Activity history** section maintains a record of events that have occurred in your Auto Scaling group. The Status column contains the current status of your instances. When your instances are launching, the status column shows *PreInService*. The status changes to *Successful* after an instance is launched.



- Choose the **Instance management** tab.

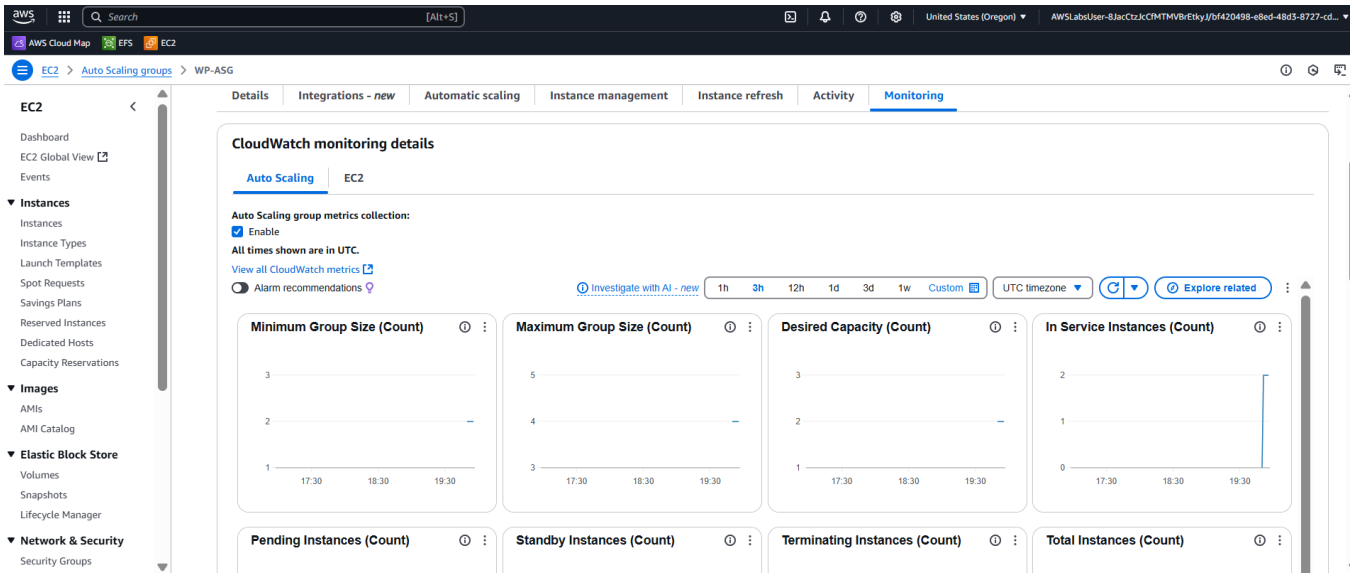
Your Auto Scaling group has launched two Amazon EC2 instances and they are in the *InService* lifecycle state. The Health Status column shows the result of the Amazon EC2 instance health check on your instances.

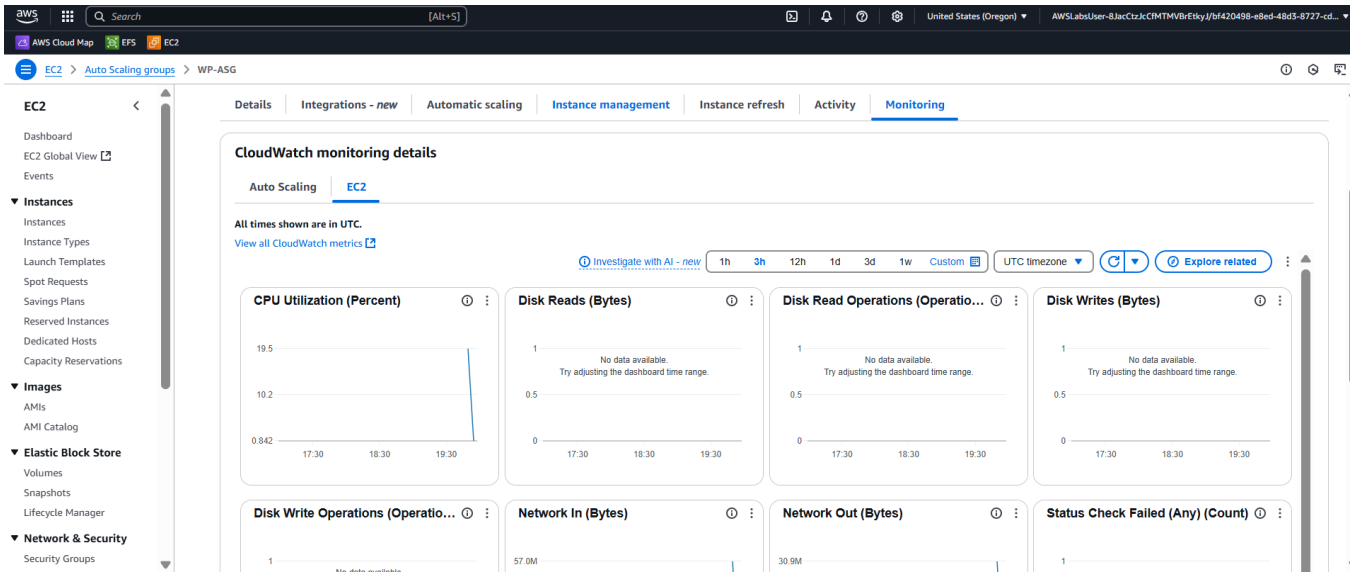


If your instances have not reached the *InService* state yet, you need to wait a few minutes. You can choose the refresh button to retrieve the current lifecycle state of your instances.

- Choose the **Monitoring** tab. Here, you can review monitoring-related information for your Auto Scaling group.

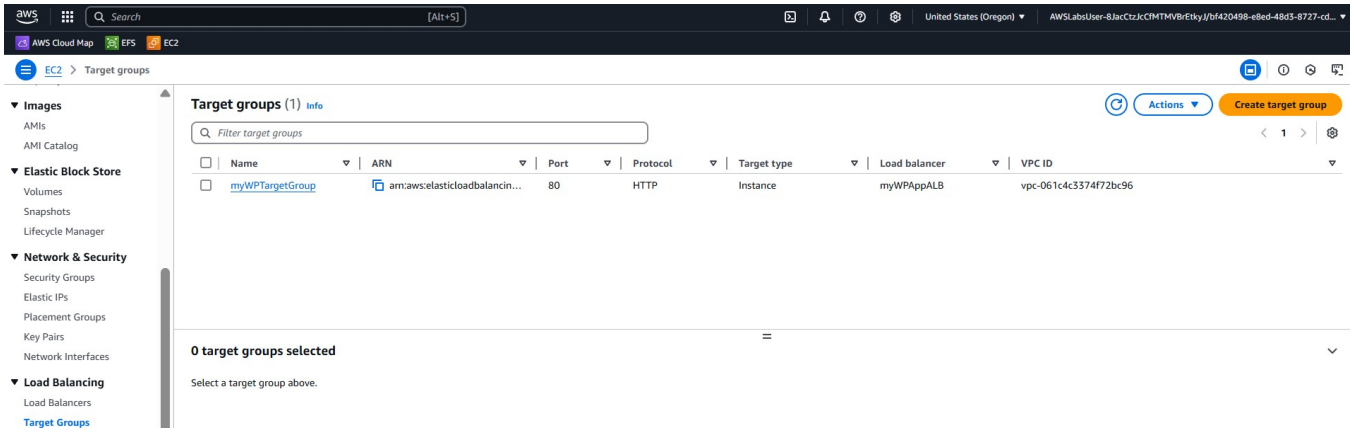
This page provides information about activity in your Auto Scaling group, as well as the usage and health status of your instances. The **Auto Scaling** tab displays Amazon CloudWatch metrics about your Auto Scaling group, while the **EC2** tab displays metrics for the Amazon EC2 instances managed by the Auto Scaling group.



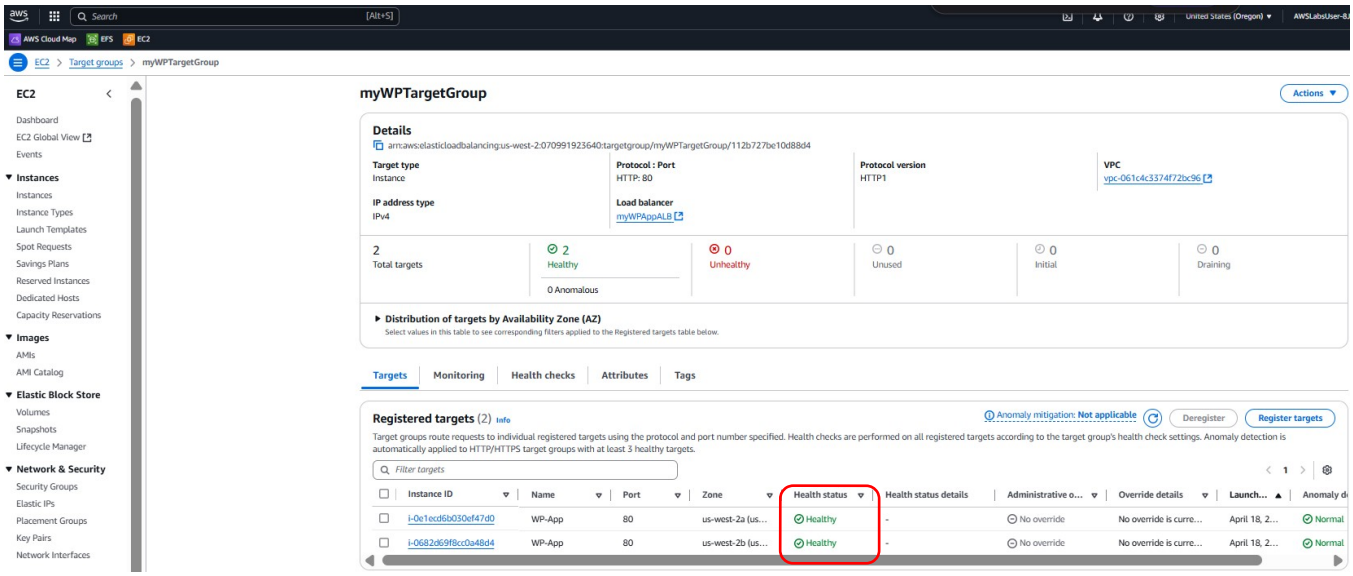


Task 6.2: Verify the target groups are healthy

- In the left navigation pane on the left, choose **Target Groups**.



- Choose the **myWPTargetGroup** link.

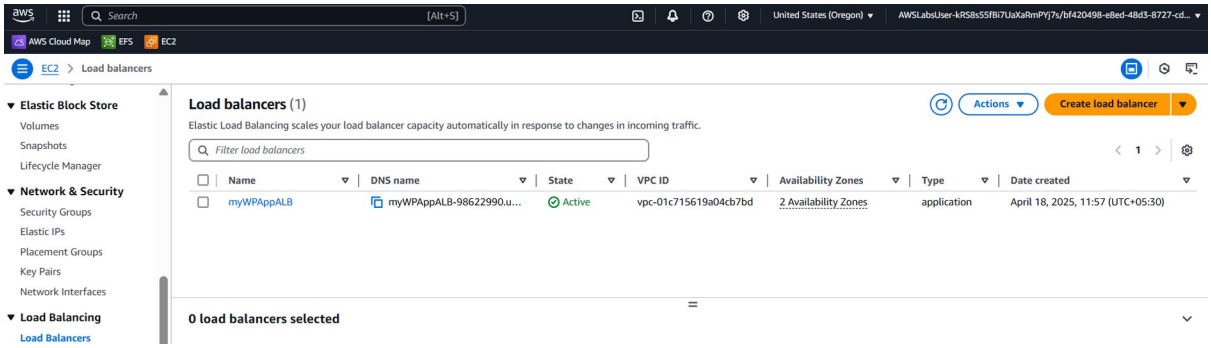


- In the **Targets** tab, wait until the instance Health status is displayed as *healthy*.

Note: It can take up to 5 minutes for the health checks to show as healthy. Wait for the **Health status** to display *healthy* before continuing.

Task 6.3: Log in to the WordPress web application

- In the left navigation pane, choose **Load Balancers**.



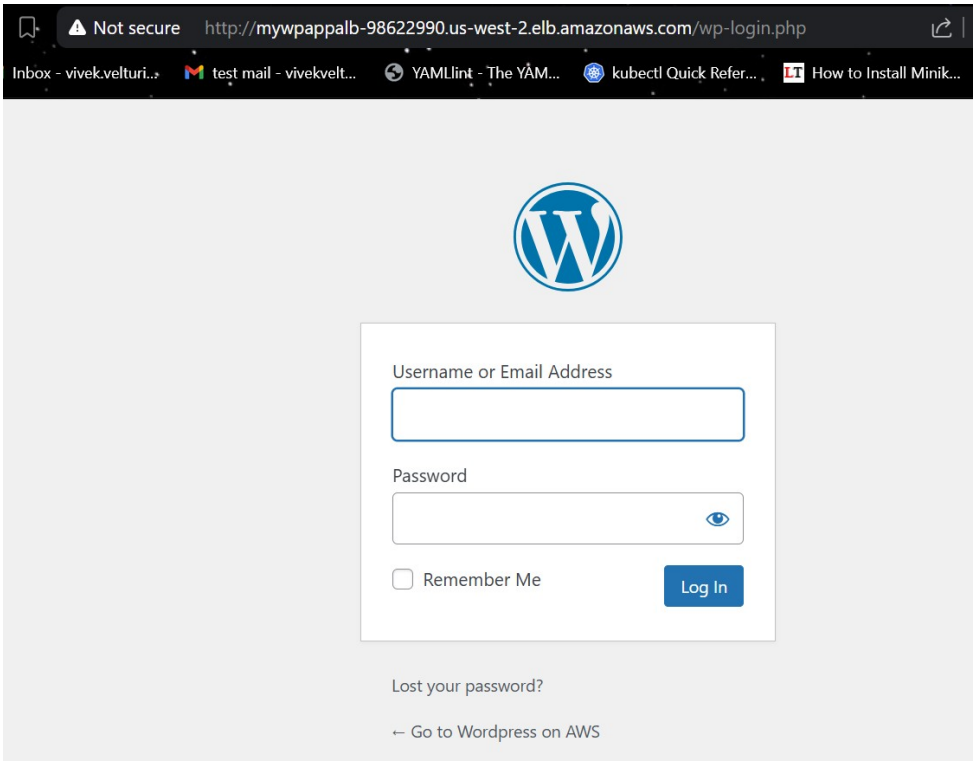
- Copy the **myWPAppALB** load balancer **DNS name** (myWPAppALB-98622990.us-west-2.elb.amazonaws.com) to a text editor and append the value `/wp-login.php` to the end of the DNS name to complete your WordPress application URL.

Expected output: Example of a completed WordPress application URL:

myWPAppELB-4e009e86b4f704cc.elb.us-west-2.amazonaws.com/wp-login.php

- Paste the WordPress application URL value into a new browser tab.

The **WordPress login** page is displayed.



- Enter the following:
  - Username or Email Address: Enter `wpadmin` .
  - Password: Paste the **LabPassword** value(1QaDM4T5Cxny) .

- Choose the **Log in** button.

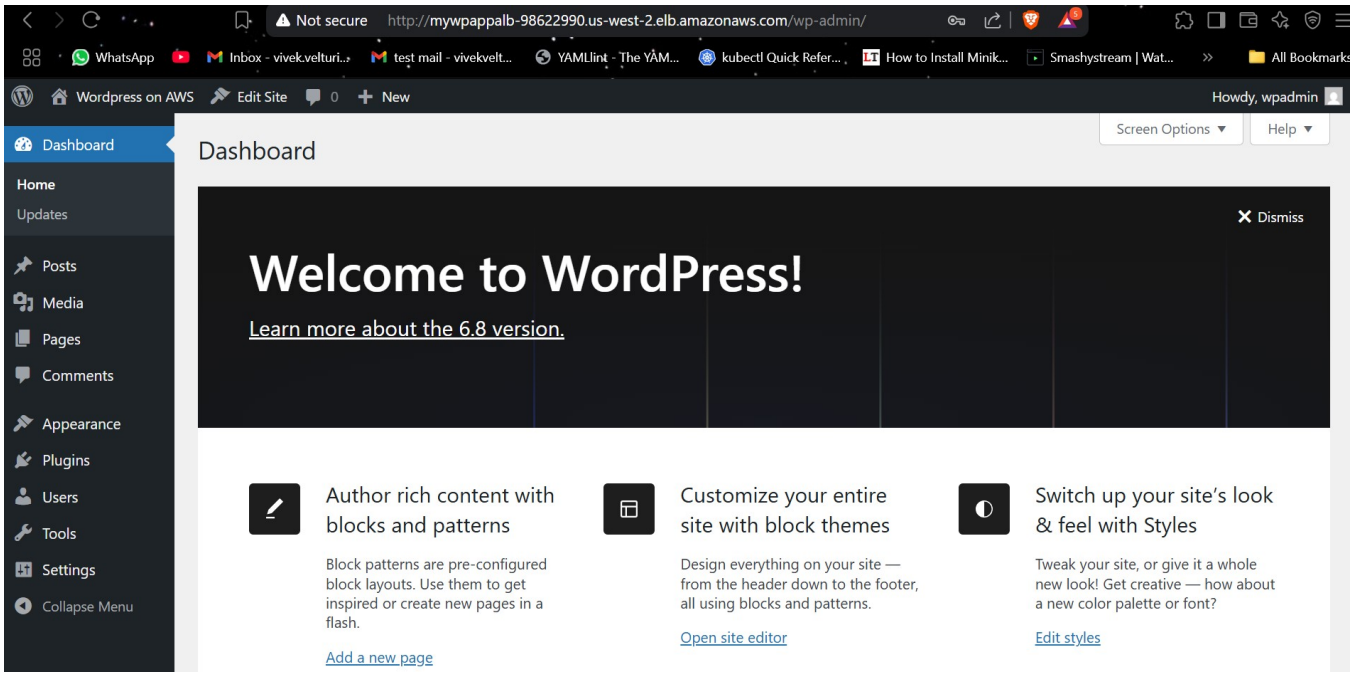


Image description: Preceding image depicts the final state when user navigates to the /wp-login.php page and inputs the login credentials.

You have created the AWS Auto Scaling group and successfully launched the WordPress application.

Conclusion

You now have successfully:

- Deployed a virtual network spread across multiple Availability Zones in a Region using a CloudFormation template.
- Deployed a highly available and fully managed relational database across those Availability Zones using Amazon RDS.
- Used Amazon EFS to provision a shared storage layer across multiple Availability Zones for the application tier, powered by NFS.
- Created a group of web servers that automatically scales in response to load variations to complete your application tier.