AWS Systems Manager – Parameter Store

**Title**

AWS Systems Manager – Secure Configuration Management using Parameter Store

**What is this and why we need it**

In real-world DevOps and production environments, applications require configuration values such as database URLs, environment names, and API keys. Storing these values directly in code or configuration files is insecure and difficult to manage across environments.

AWS Systems Manager Parameter Store allows us to securely store, manage, and retrieve configuration data and secrets as parameters. It integrates with AWS services like EC2, Lambda, and ECS, enabling centralized configuration management with encryption, versioning, and access control.

**Objective**

To securely store application configuration values in AWS Systems Manager Parameter Store and retrieve them from an EC2 instance using IAM permissions.

**Prerequisites**

* AWS account
* IAM user with Systems Manager and EC2 permissions
* One EC2 instance (Amazon Linux)
* EC2 instance must have **SSM Agent installed** (default in Amazon Linux)
* IAM role attached to EC2 with:
  + AmazonSSMManagedInstanceCore
  + AmazonSSMReadOnlyAccess

**Step-by-Step Implementation**

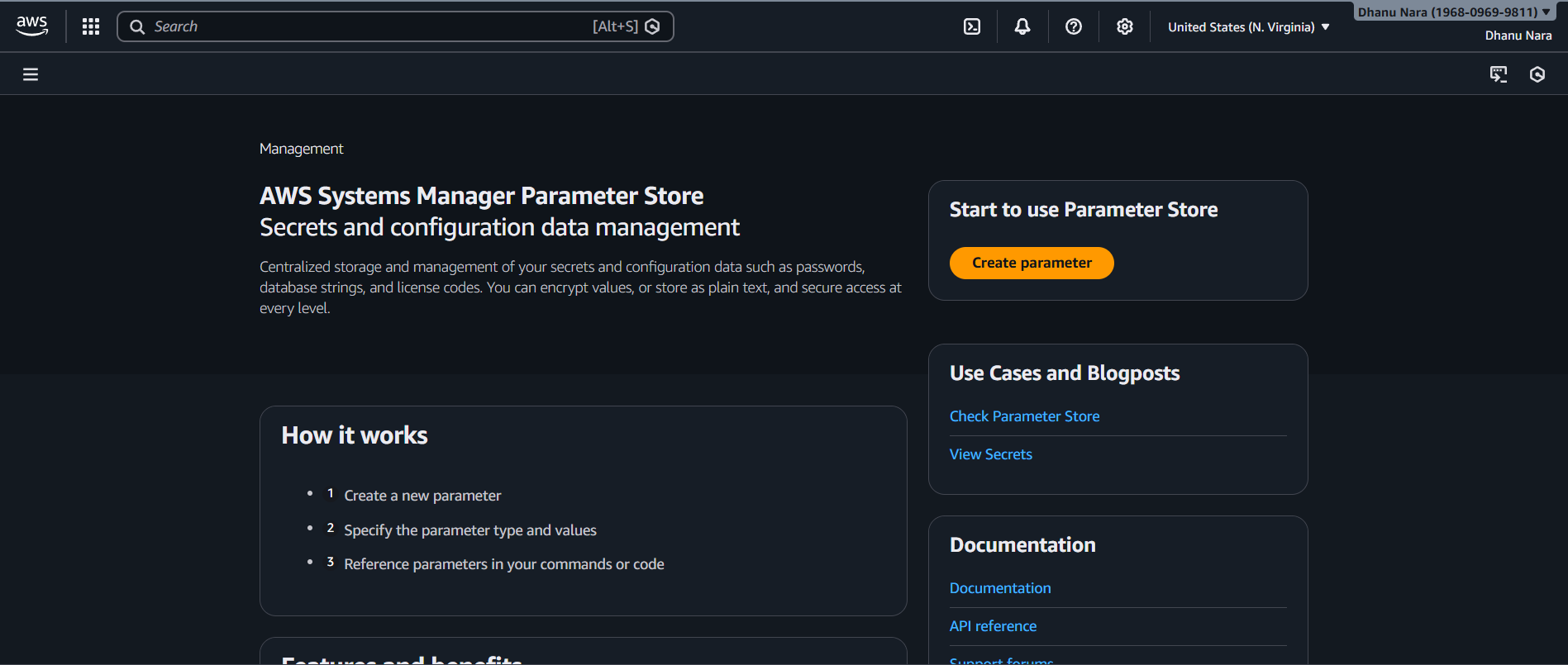
**Open AWS Systems Manager**

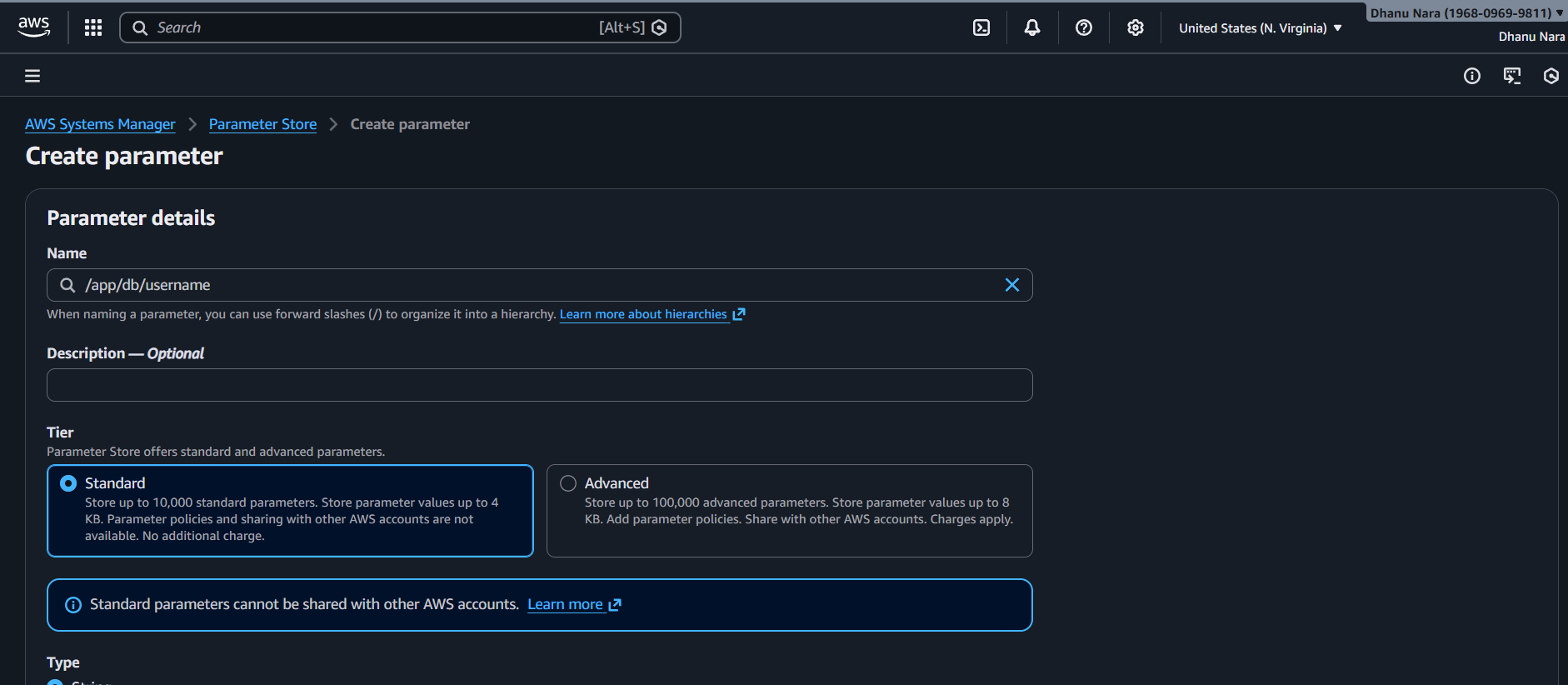
1. Go to AWS Console
2. Search for **Systems Manager**
3. Click **Parameter Store** from the left menu



**Create a Parameter**

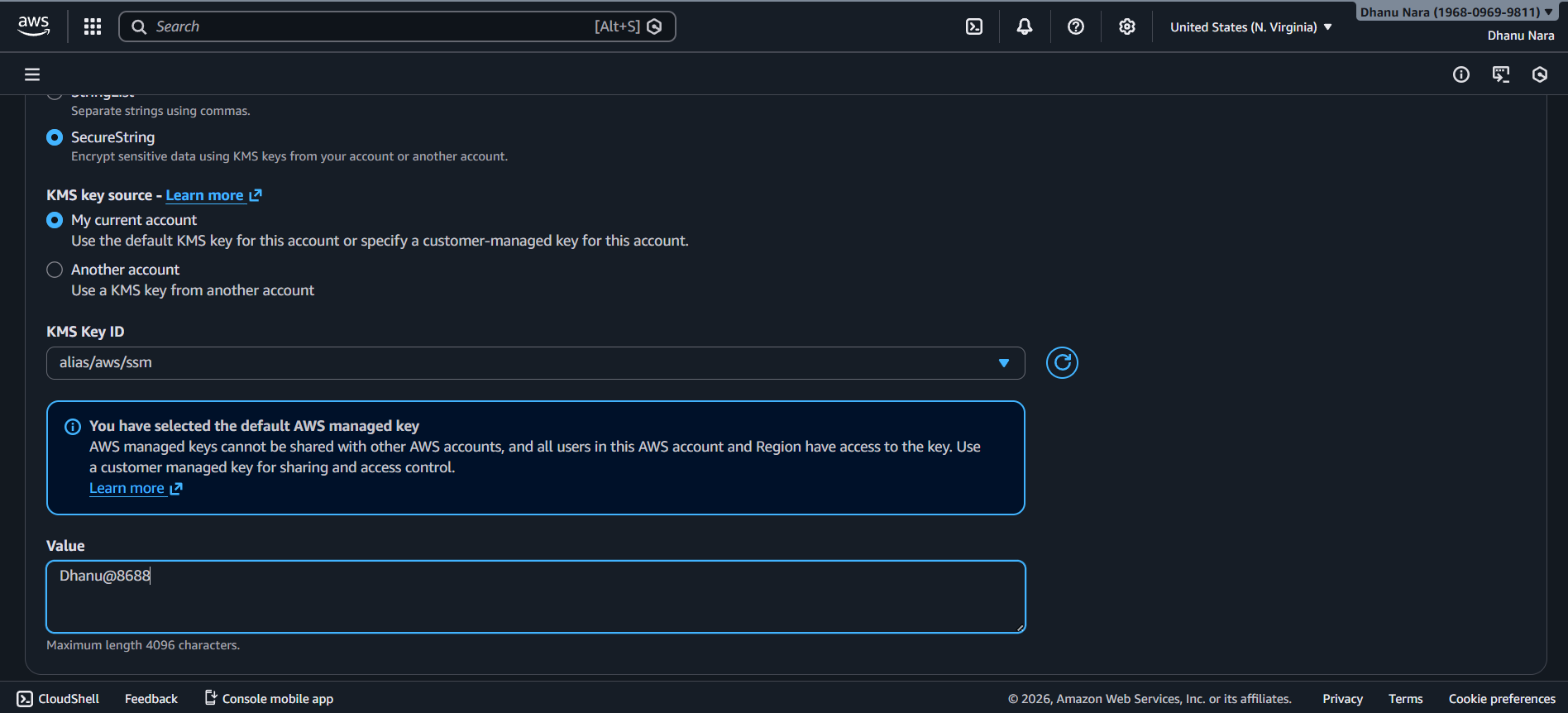
1. Click **Create parameter**
2. Fill the details:
   * **Name**: /app/db/username
   * **Tier**: Standard
   * **Type**: String
   * **Value**: admin
   * Click **Create parameter**



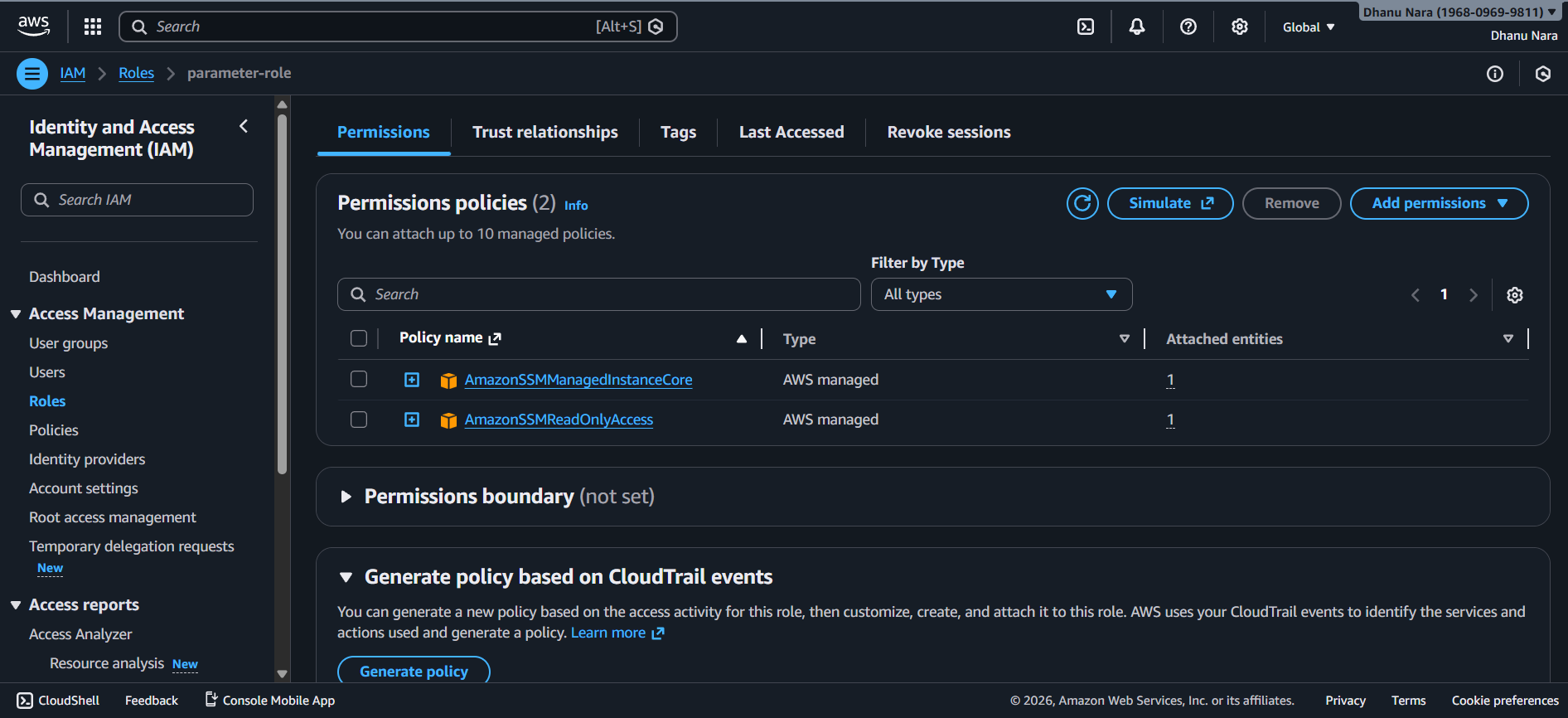


**Create a Secure Parameter**

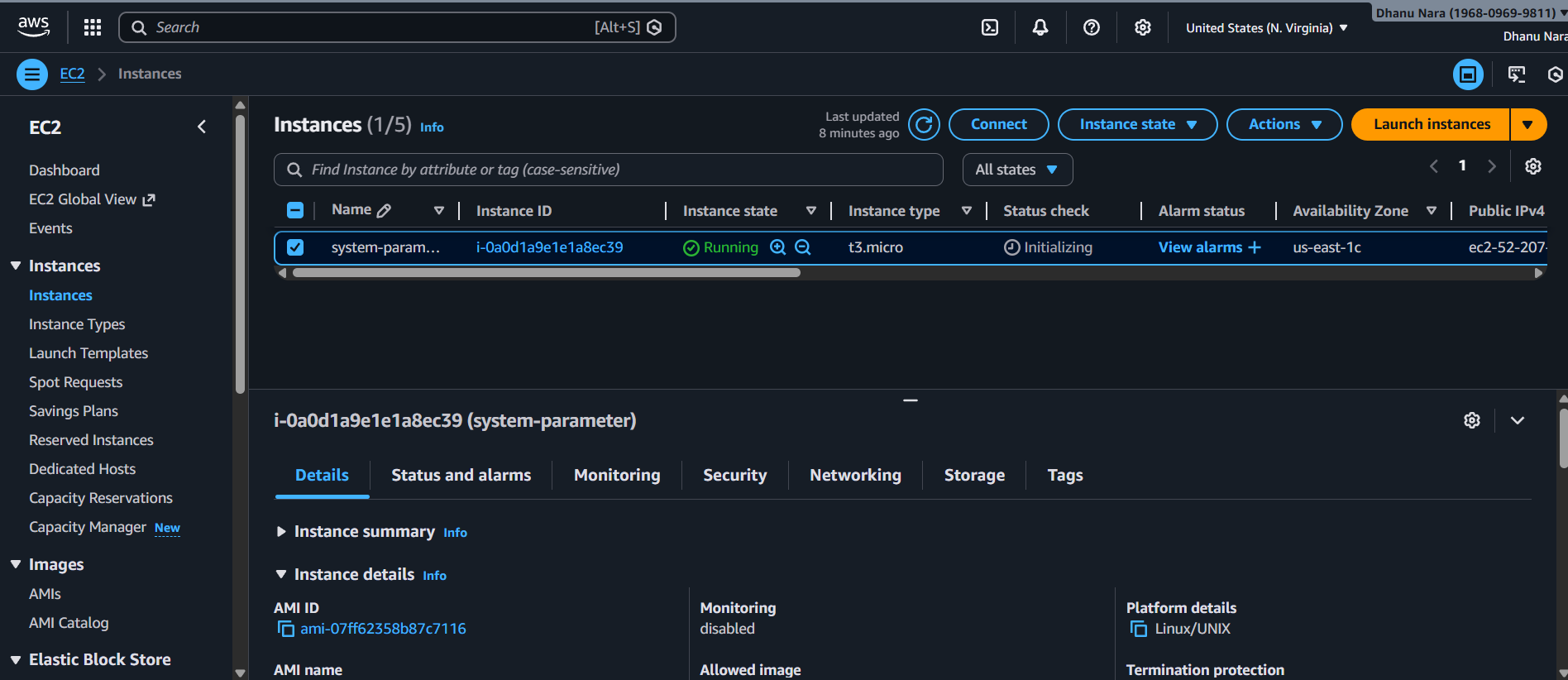
1. Click **Create parameter** again
   * **Name**: /app/db/password
   * **Tier**: Standard
   * **Type**: SecureString
   * **KMS Key**: aws/ssm
   * **Value**: dhanu@8688
   * Click **Create parameter**



Create an IAM role with AmazonSSMManagedInstancecore and AmazonSSMReadOnlyAccess

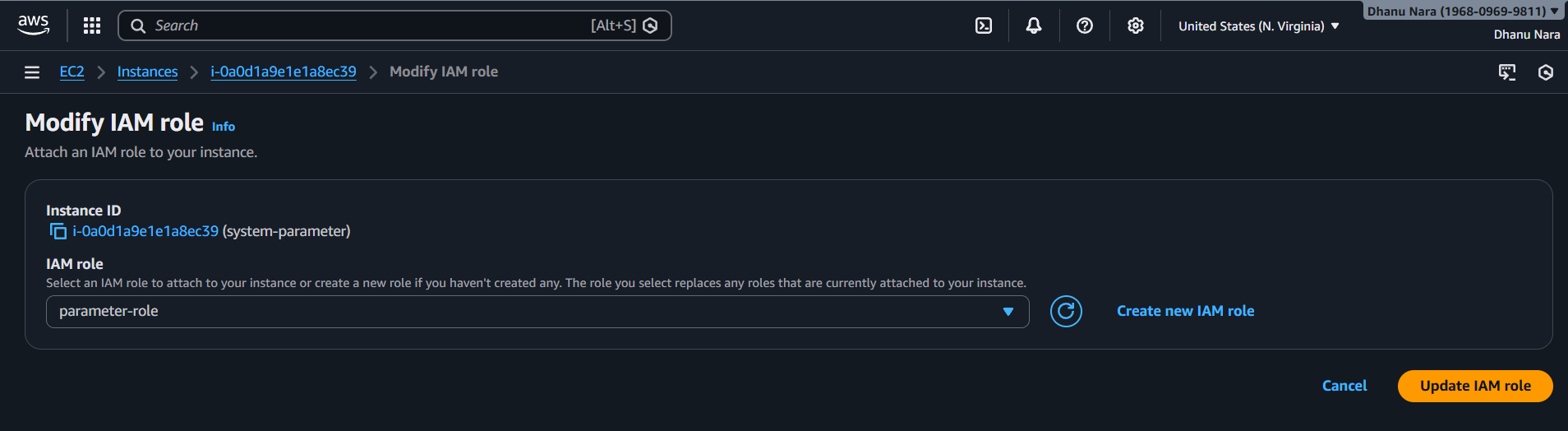


Launch an EC2 Instance



**Attach IAM Role to EC2**

1. Go to **EC2 → Instances**
2. Select your instance
3. Click **Actions → Security → Modify IAM role**
4. Attach a role with:
   * AmazonSSMManagedInstanceCore
   * AmazonSSMReadOnlyAccess
5. Save changes



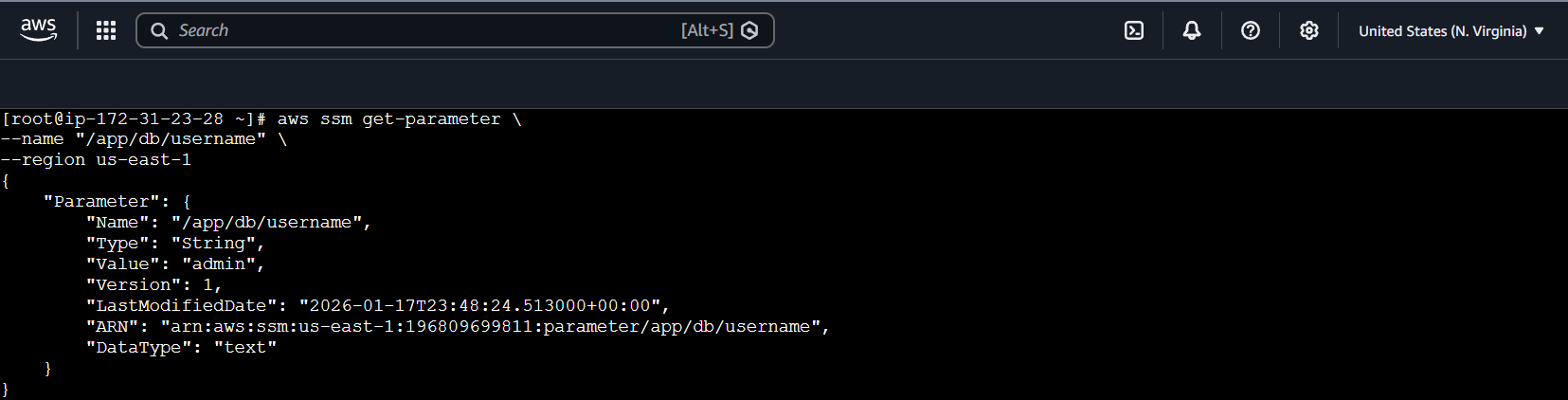
**Retrieve Parameter from EC2**

Run the following command:

aws ssm get-parameter \

--name "/app/db/username" \

--region us-east-1



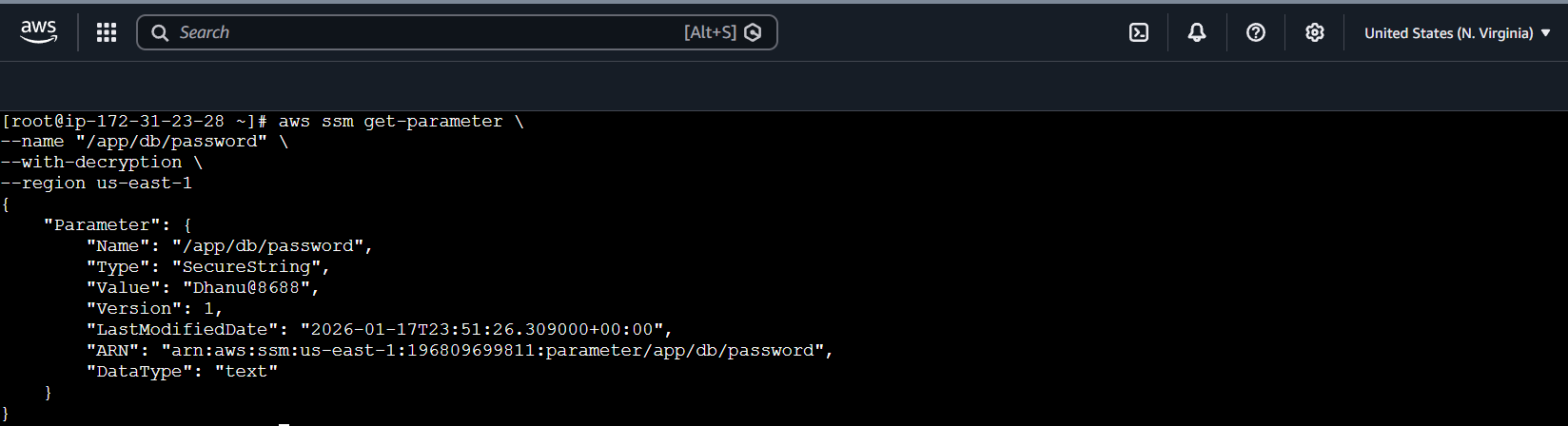
For secure parameter:

aws ssm get-parameter \

--name "/app/db/password" \

--with-decryption \

--region us-east-1



**Verify Output**

Stored values returned securely from Parameter Store.

**Conclusion**

AWS Systems Manager Parameter Store provides a secure and centralized way to manage application configuration and secrets. By using IAM roles and encryption, sensitive data is protected and easily accessible without hardcoding values in applications, making it ideal for DevOps and production environments.

AWS WAF

**Title**

**Securing a Web Application Using AWS WAF with Application Load Balancer**

**Objective**

Deploy a web application on EC2, expose it via an Application Load Balancer (ALB), protect it using AWS WAF (Web ACL), and verify WAF protection by blocking a SQL injection attack.

**Prerequisites**

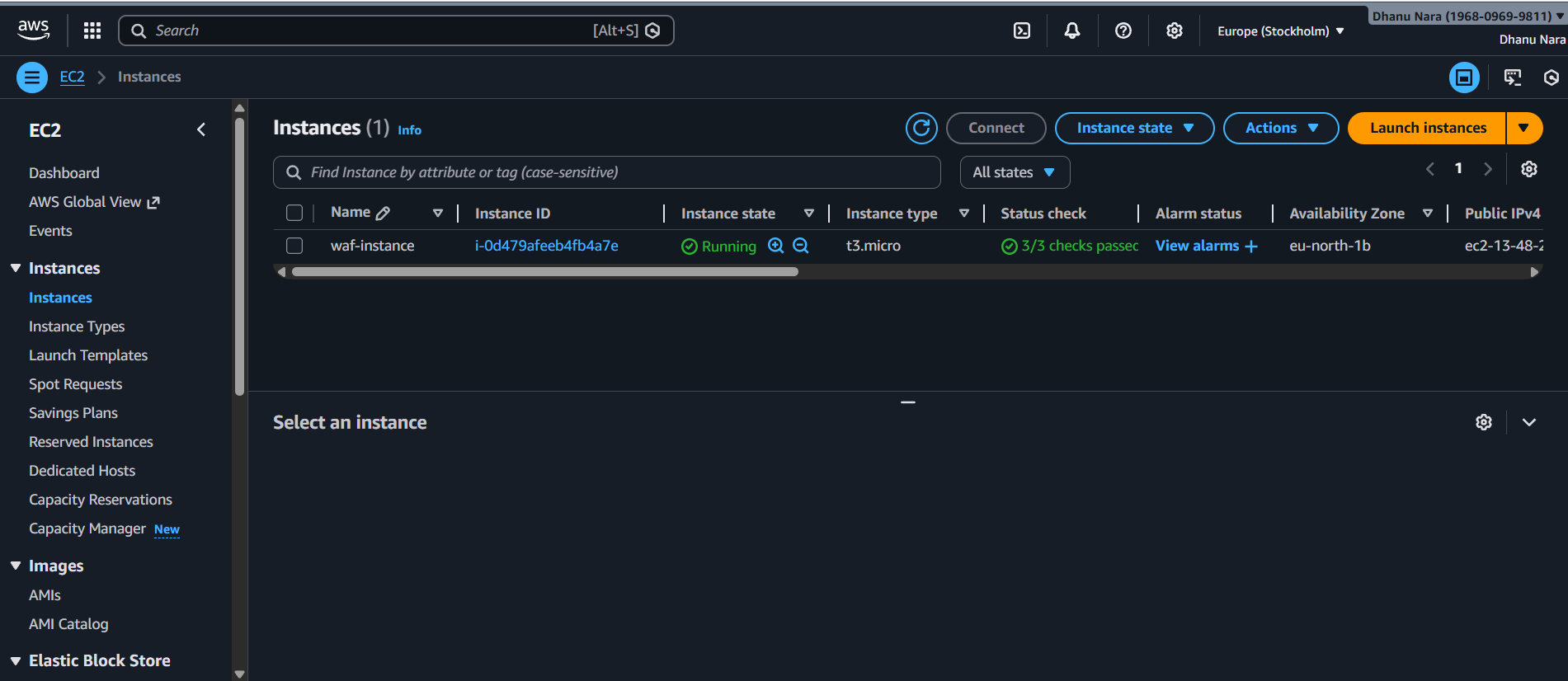
Before starting this task, the following are required:

1. AWS Account with administrative access
2. One EC2 key pair
3. Basic knowledge of:
   * EC2
   * Security Groups
   * Application Load Balancer
   * AWS WAF
4. Region selected: **Europe (Stockholm – eu-north-1)**

**STEP-BY-STEP IMPLEMENTATION**

**Launch EC2 Instance**

1. Go to **EC2 → Launch Instance**
2. AMI: **Amazon Linux 2**
3. Instance type: **t2.micro**
4. Key pair: Select existing key
5. Network settings:
   * VPC: Default
   * Subnet: Public subnet
6. Security Group (EC2-SG):
   * SSH (22) → Your IP
   * HTTP (80) → **ALB Security Group**
7. Launch instance



**Install Apache Web Server**

Install apache

sudo yum install httpd -y

sudo systemctl start httpd

sudo systemctl enable httpd

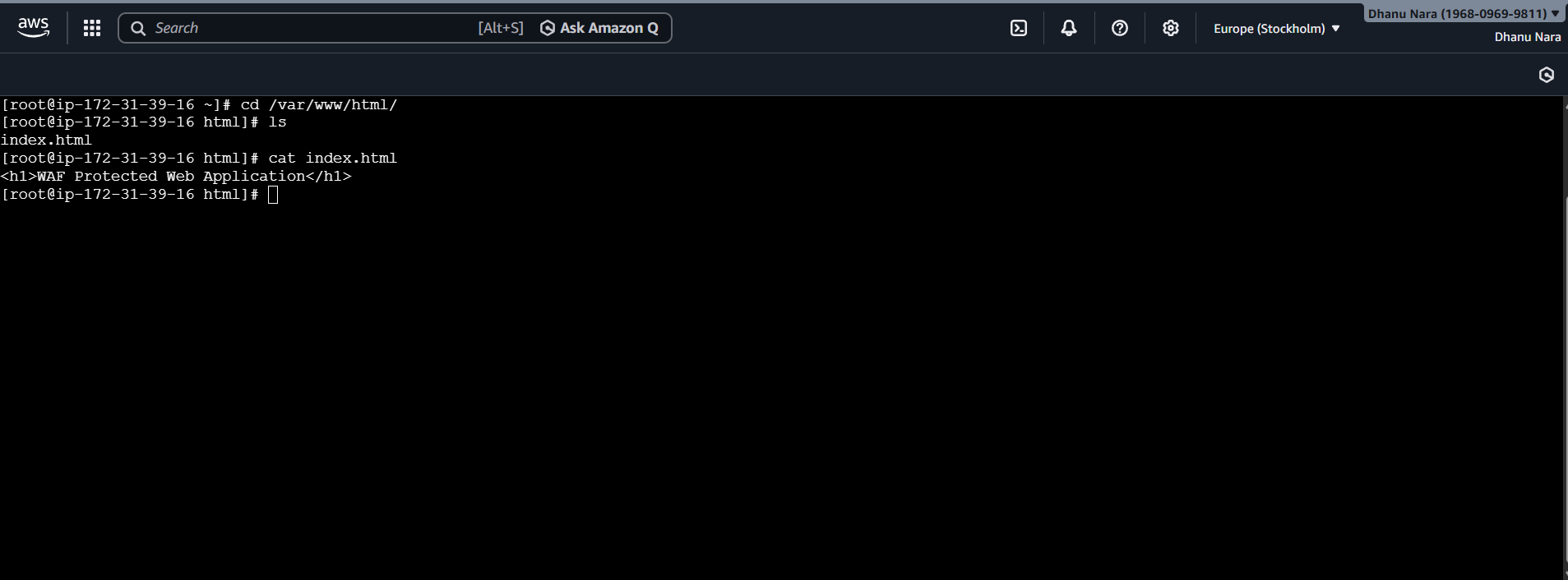
Create web page:

sudo vi /var/www/html/index.html

Add:

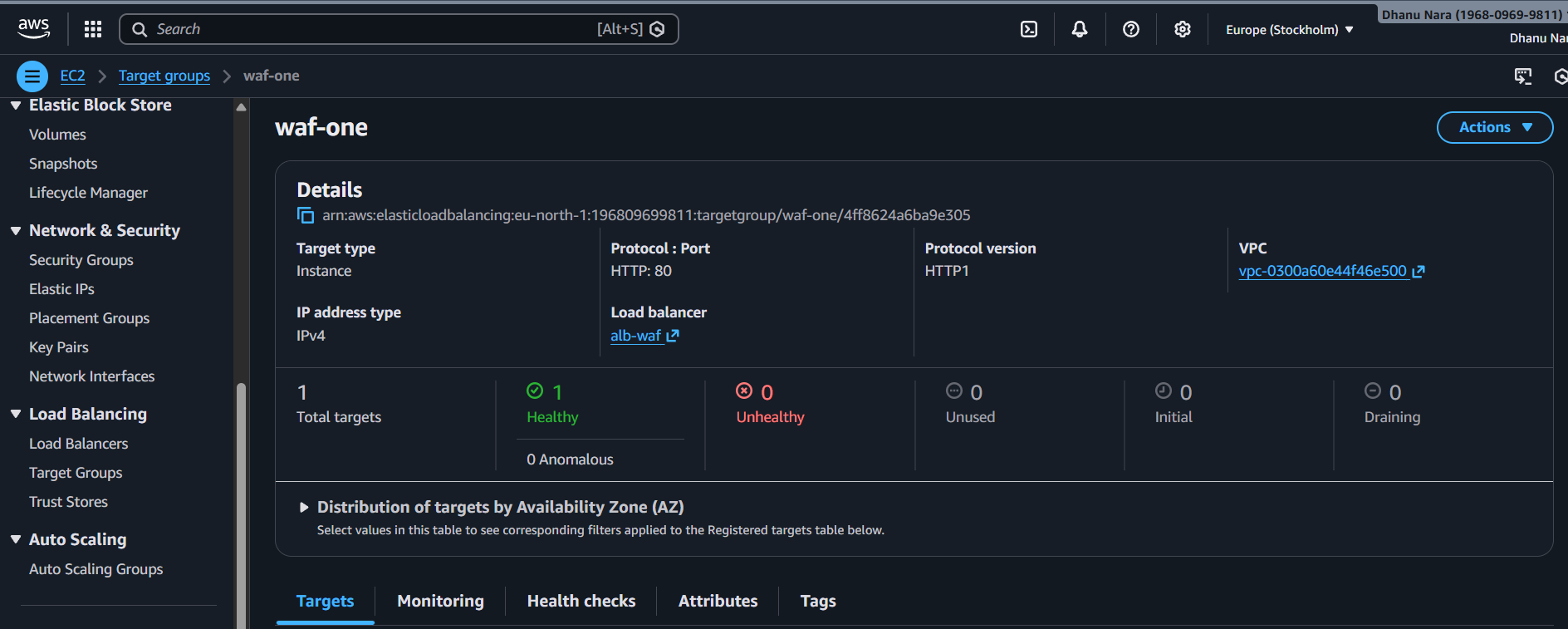
<h1>WAF Protected Web Application</h1>

Save and exit.



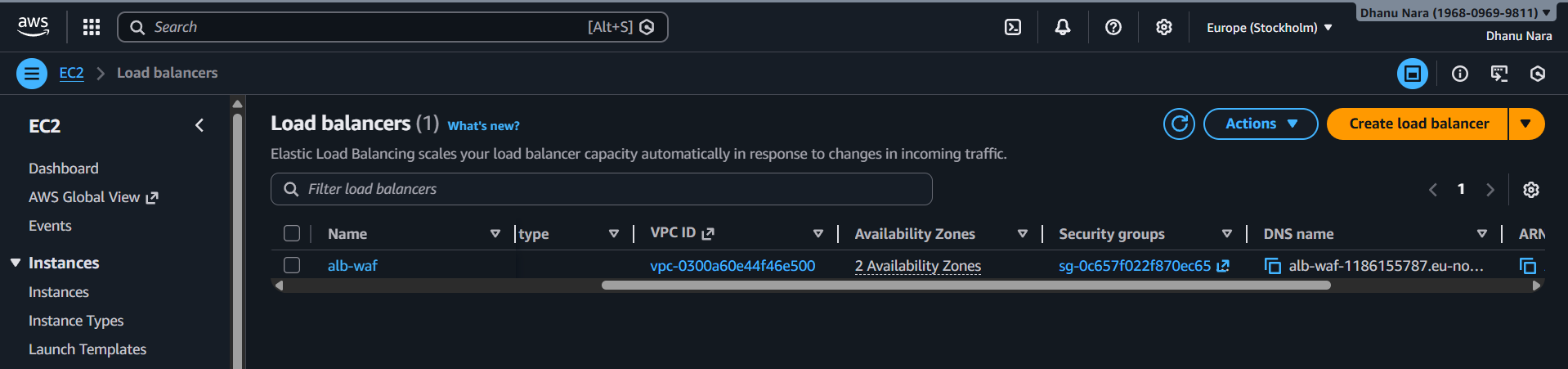
**Create Target Group**

1. Go to **EC2 → Target Groups**
2. Create Target Group:
   * Target type: Instances
   * Protocol: HTTP
   * Port: 80
3. Register EC2 instance
4. Health check status should be **Healthy**



**Create Application Load Balancer**

1. Go to **EC2 → Load Balancers → Create**
2. Select **Application Load Balancer**
3. Configuration:
   * Scheme: Internet-facing
   * IP type: IPv4
4. Listener:
   * HTTP : 80
5. Select VPC and **at least 2 public subnets**
6. ALB Security Group:
   * HTTP (80) → 0.0.0.0/0
7. Attach target group
8. Create ALB



**Verify Application via ALB**

Copy ALB DNS name and open in browser:

http://<ALB-DNS>

Expected Output:

WAF Protected Web Application



**Create AWS WAF Web ACL (Protection Pack)**

1. Go to **AWS WAF & Shield**
2. Click **Protection packs (web ACLs)**
3. Click **Create protection pack (web ACL)**

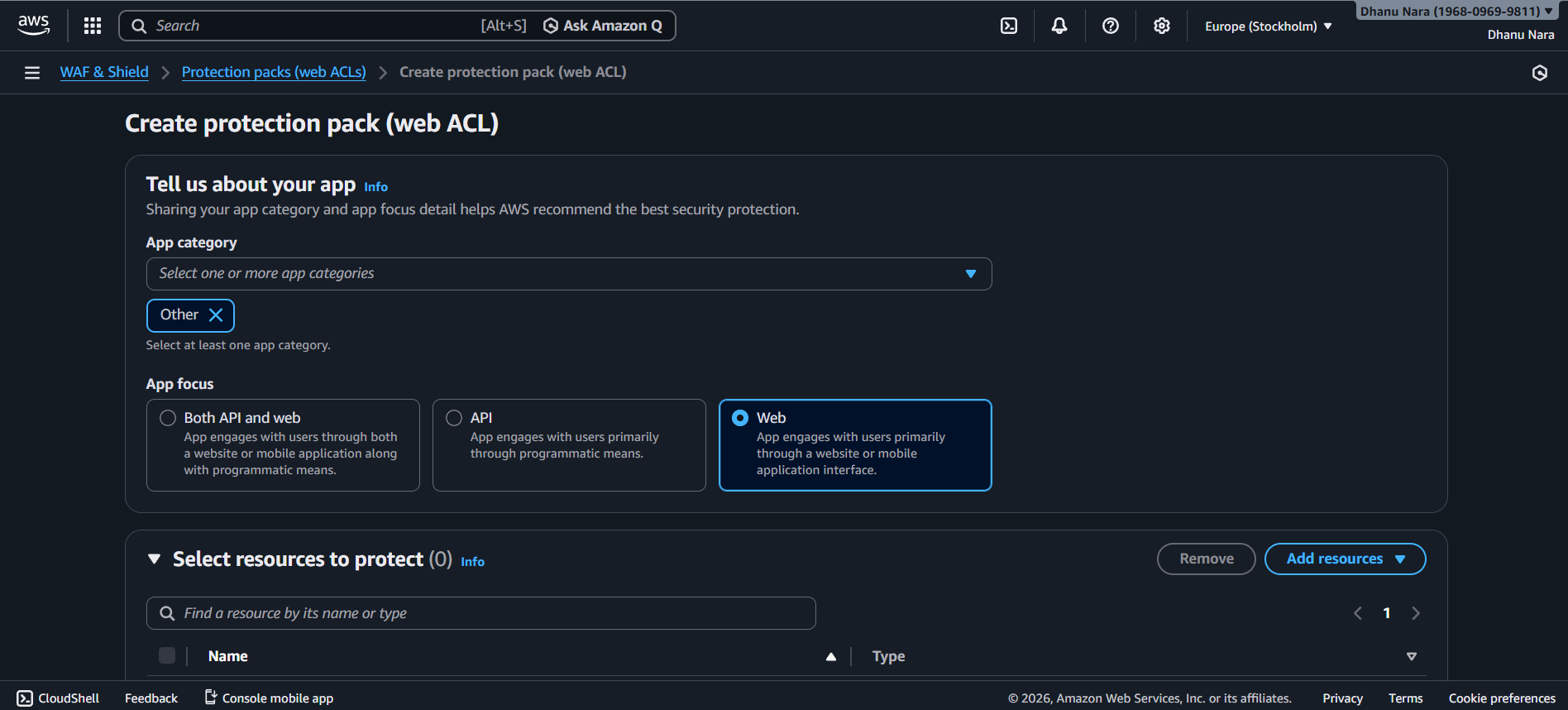
**Tell us about your app:**

* App category: **Other**
* App focus: **Web**

Top of Form

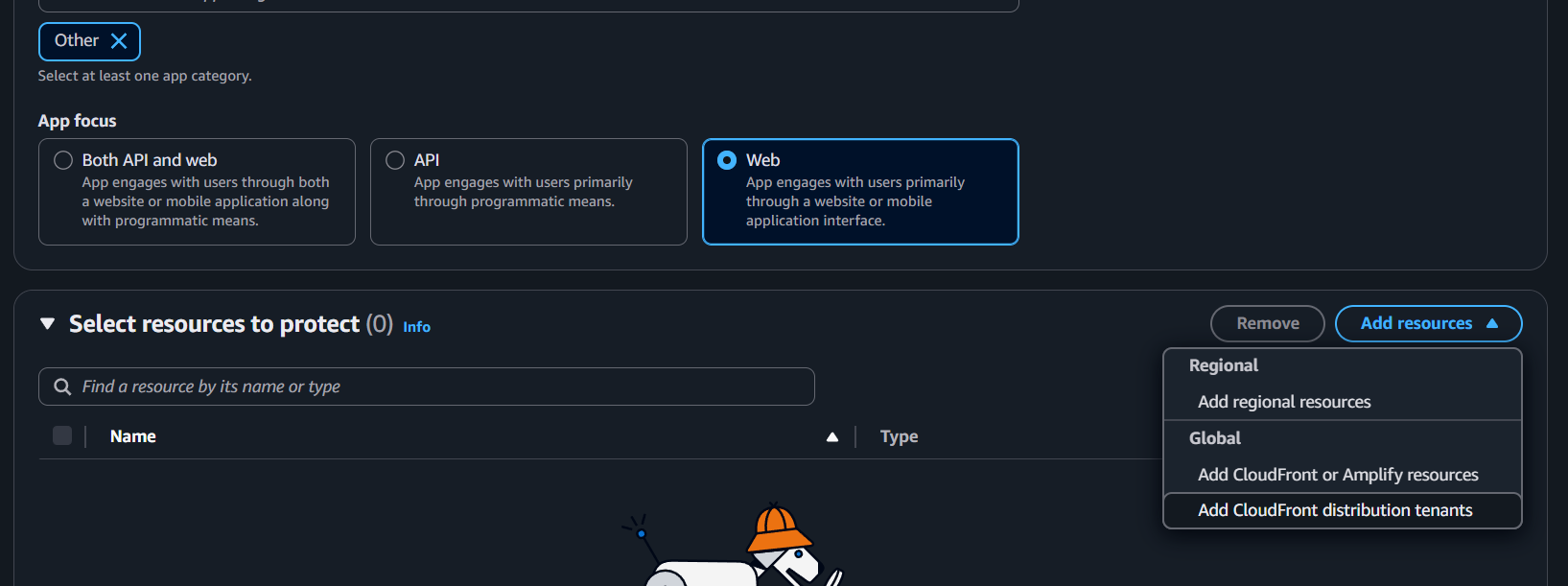
Bottom of Form

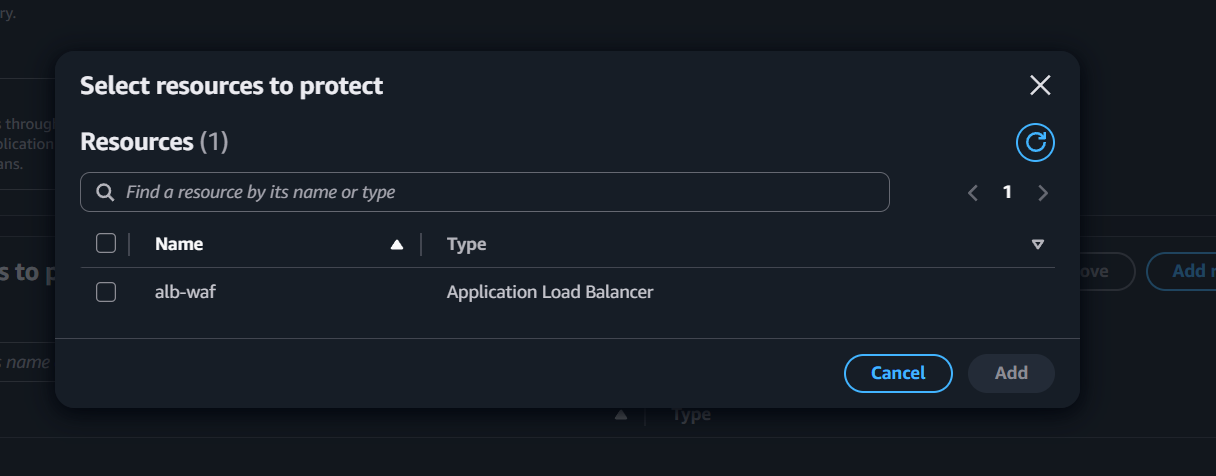




**Associate Resource (ALB)**

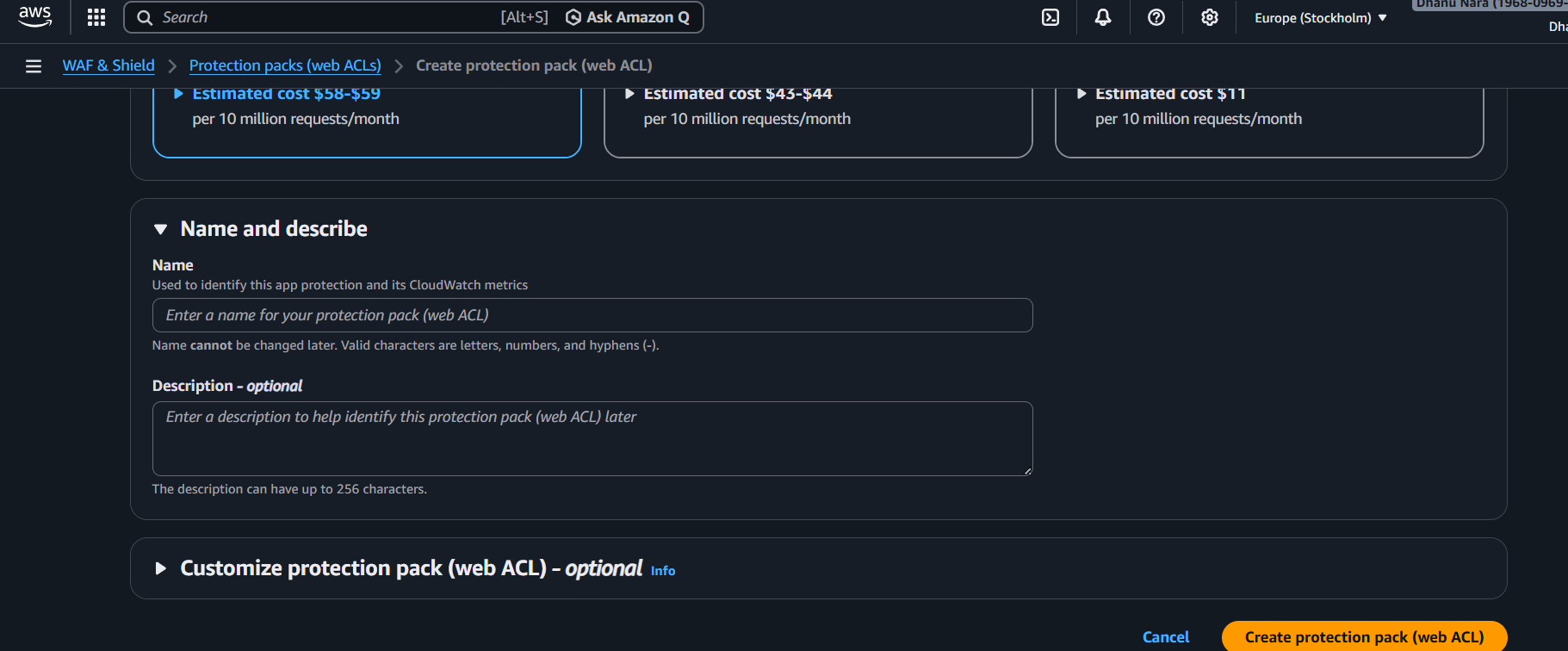
1. In **Select resources to protect**
2. Click **Add resources**
3. Choose **Regional**
4. Select **Application Load Balancer**

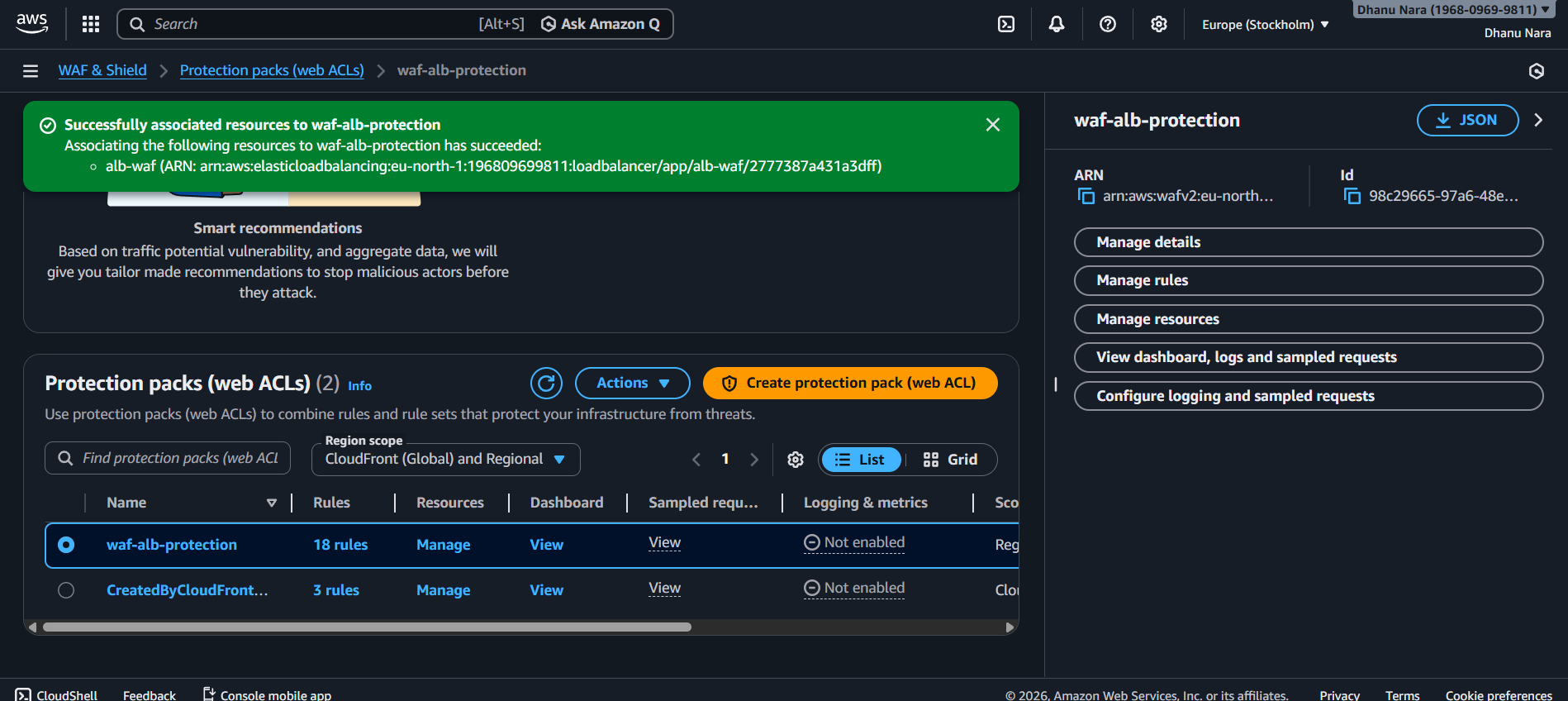




**Name and Create Web ACL**

* Name: waf-alb-protection
* Description: Optional
* Click **Create protection pack (web ACL)**





**Normal Traffic Test**

Open browser:

http://<ALB-DNS>

Expected:

* Page loads successfully

**SQL Injection Attack Test**

Open browser and test:

http://<ALB-DNS>/?id=1' OR '1'='1

Expected Response:

403 Forbidden



**RESULT**

* Web application is successfully deployed
* ALB distributes traffic
* AWS WAF blocks malicious requests
* SQL injection attempt is denied with **403 Forbidden**

**CONCLUSION**

In this task, a web application was deployed on EC2 and exposed via an Application Load Balancer. AWS WAF was configured and associated with the ALB to provide Layer-7 security. The effectiveness of WAF was validated by simulating a SQL injection attack, which was successfully blocked, ensuring secure access to the application.

AWS SQS

**Title**

AWS SQS – Create a Queue and Send & Receive Messages

**What is AWS SQS**

Amazon Simple Queue Service (SQS) is a **fully managed message queuing service** that allows applications to **send, store, and receive messages** reliably without losing data.

It helps **decouple components** so that systems can work independently and scale easily.

**Objective**

To understand how Amazon SQS works by creating a queue, sending messages to it, and receiving those messages using the AWS Console

**Prerequisites**

* AWS account
* Basic understanding of AWS Console navigation

**Step-by-Step Implementation**

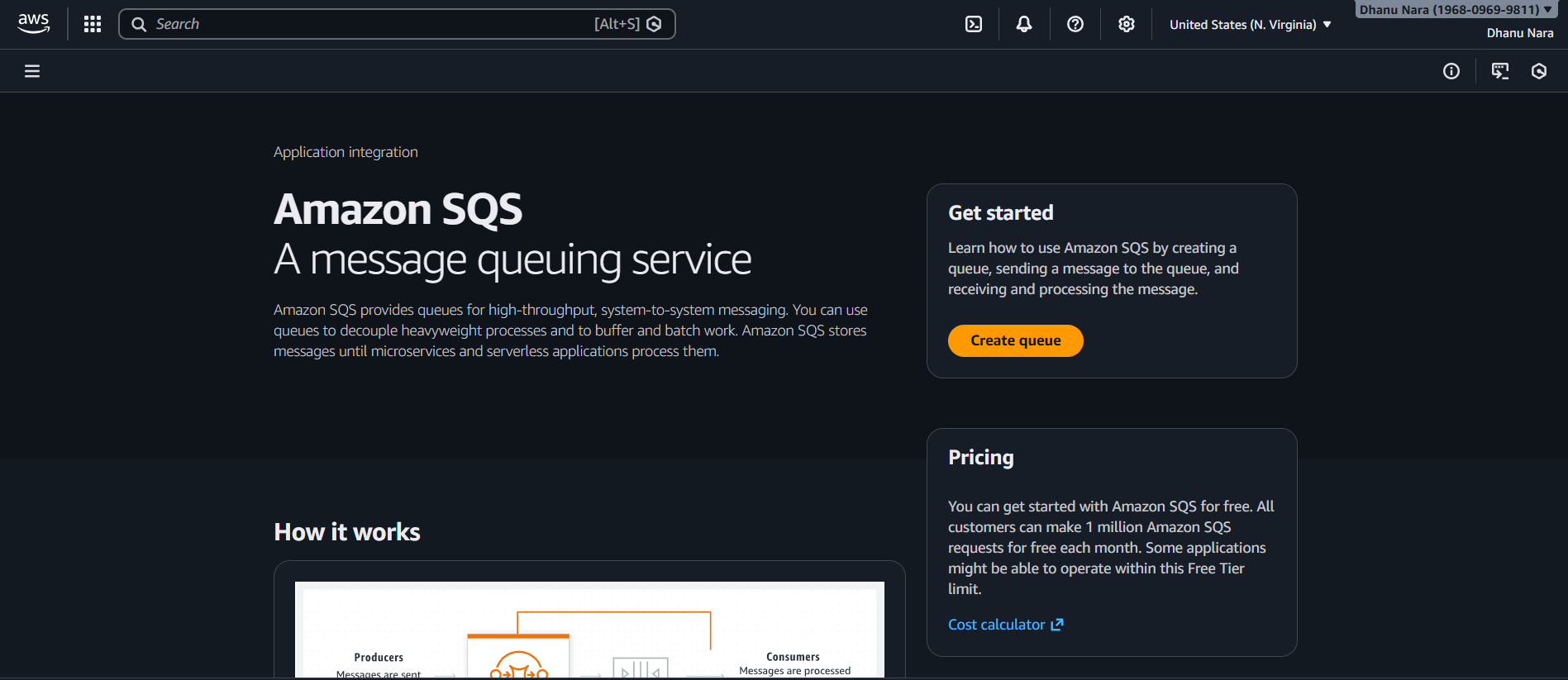
**Step 1: Open AWS SQS**

1. Login to AWS Console
2. Search for **SQS**
3. Click **Amazon Simple Queue Service**

**Step 2: Create a Queue**

1. Click **Create queue**
2. Select **Standard queue**
3. Queue name:

my-first-sqs-queue



**Configure Queue Settings**

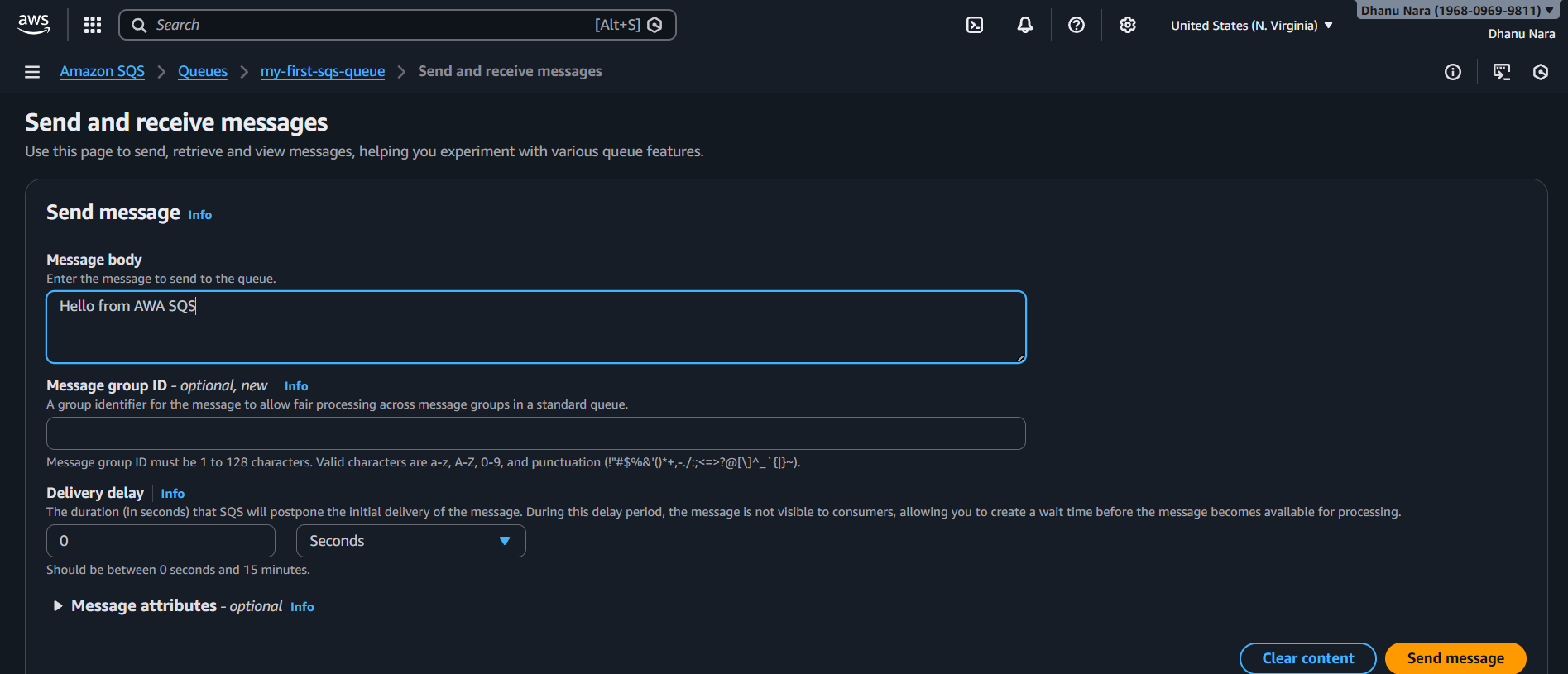
* Visibility timeout: leave default (30 seconds)
* Message retention: default (4 days)
* Delivery delay: 0 seconds
* Encryption: default (SQS-managed)

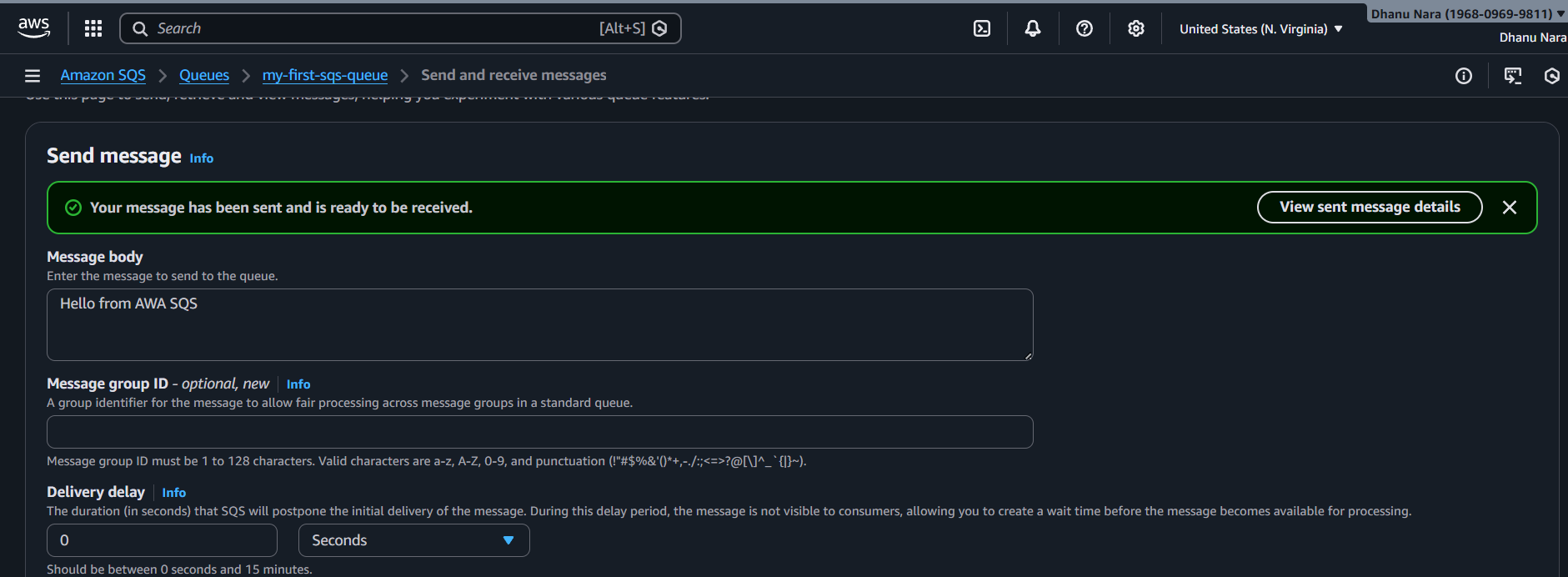
Click **Create queue**

**Send a Message to the Queue**

1. Open your queue (my-first-sqs-queue)
2. Click **Send and receive messages**
3. Under **Message body**, type:
4. Hello from AWS SQS
5. Click **Send message**

✅ Message is now stored in the queue



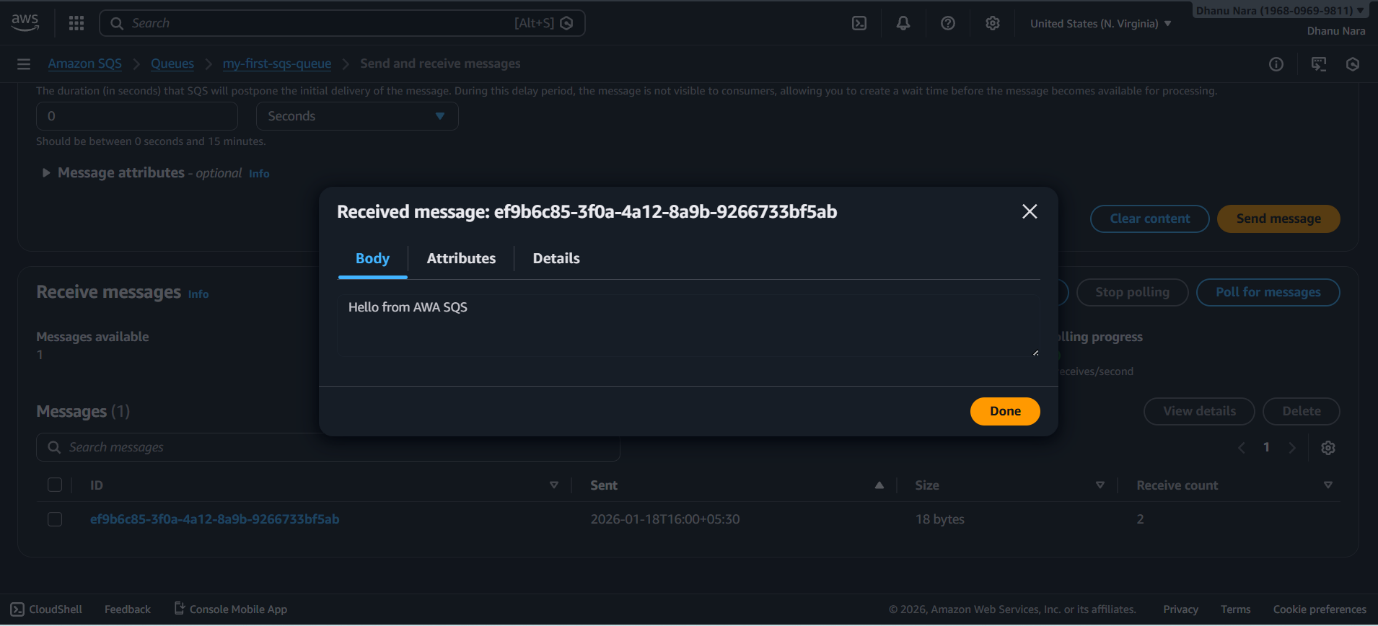


**Receive the Message**

1. In the same page, click **Poll for messages**
2. Wait a few seconds

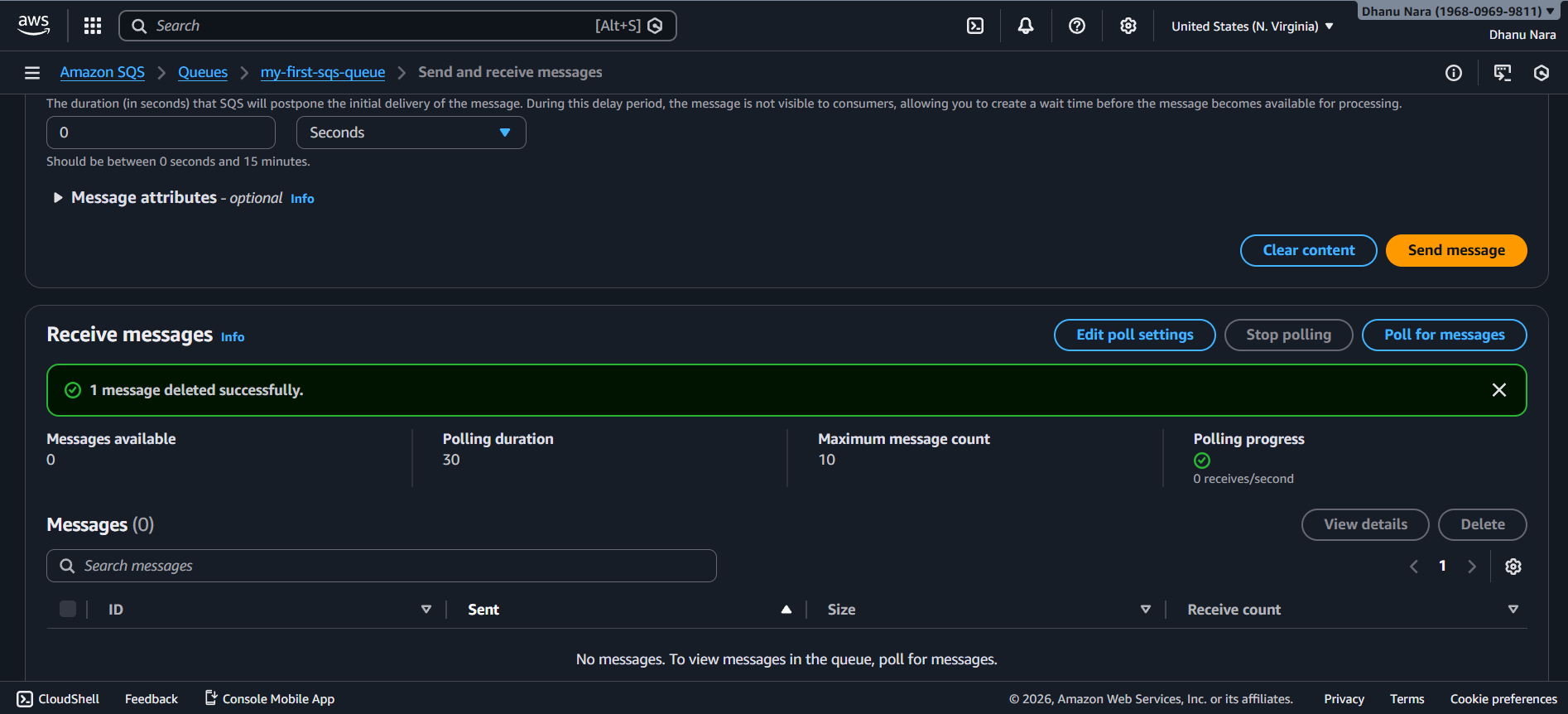
Now the message will be availabe

Hello from AWS SQS



**Delete the Message**

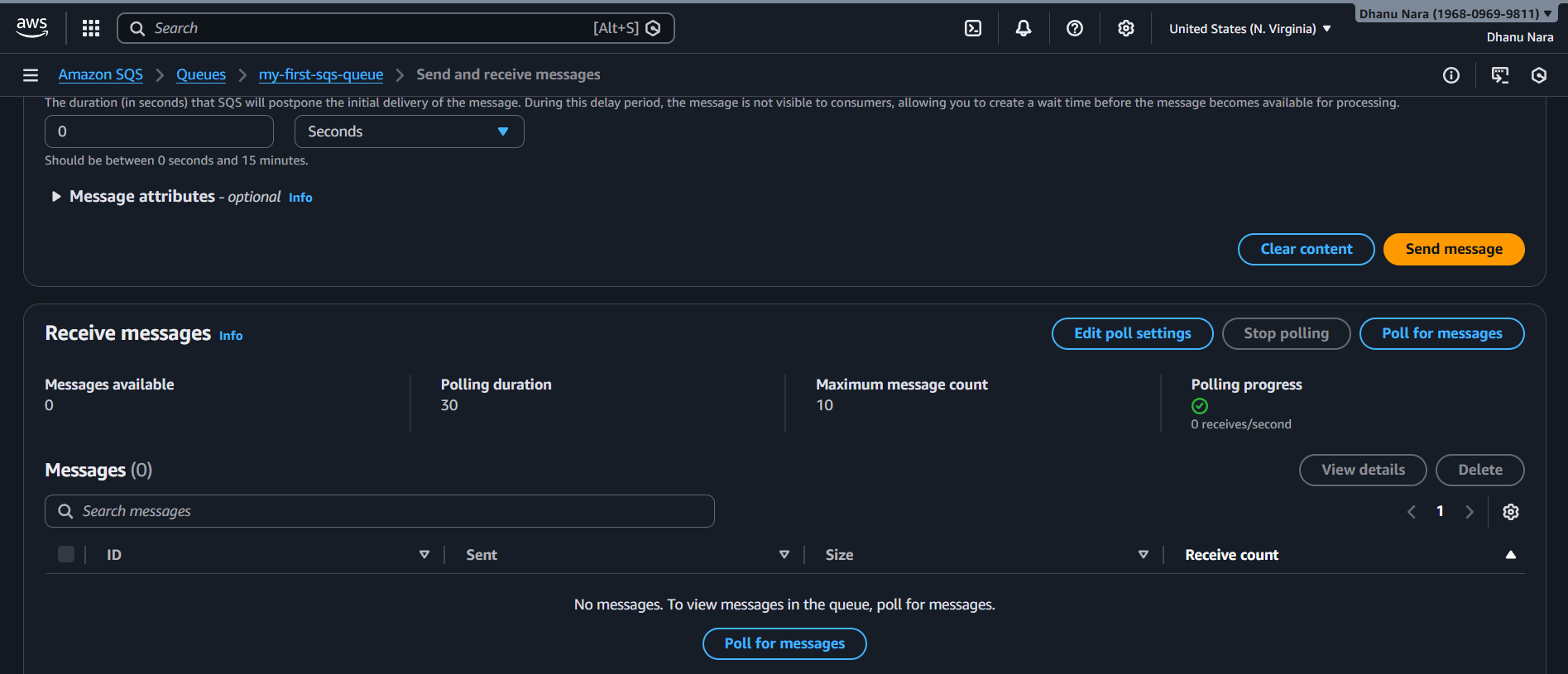
1. Select the received message
2. Click **Delete**



**Verify Queue is Empty**

* Click **Poll for messages** again
* No messages will appear

✅ Queue is now empty



**Conclusion**

AWS SQS provides a simple, reliable way to exchange messages between applications.  
By using SQS, we ensure **loose coupling**, **fault tolerance**, and **scalability** without managing servers.

Amazon MQ

**Title**

Amazon MQ (ActiveMQ) – Create Broker and Send/Receive Message Using Web Console

**Objective**

To create an Amazon MQ broker using Apache ActiveMQ, access the ActiveMQ Web Console, send a message to a queue, verify the message, and delete (consume) it to understand basic message queue functionality.

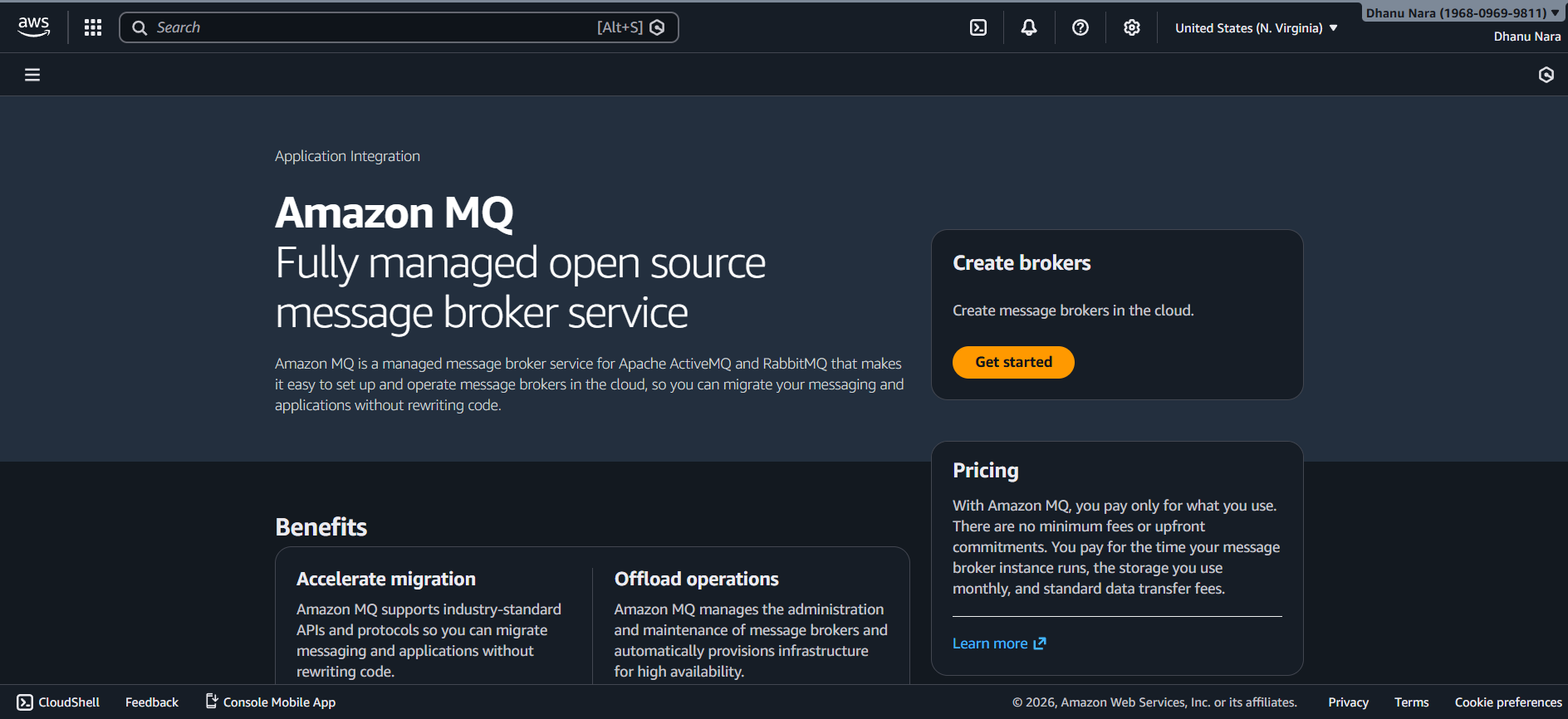
**Prerequisites**

* AWS account
* Basic AWS Console knowledge
* Default VPC available
* Internet access from browser

**Step-by-Step Implementation**

**Step 1: Open Amazon MQ Service**

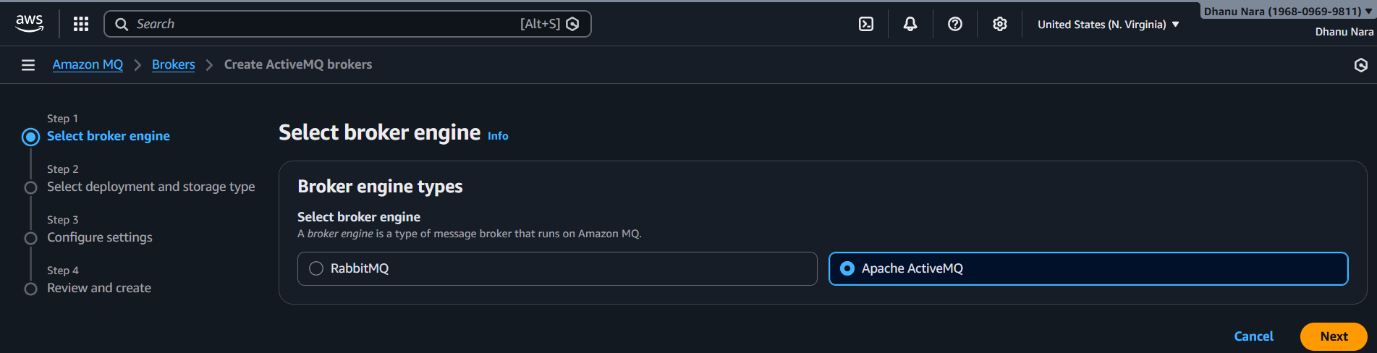
1. Login to **AWS Console**
2. Search for **Amazon MQ**
3. Click **Amazon MQ → Brokers**
4. Click **Create broker**

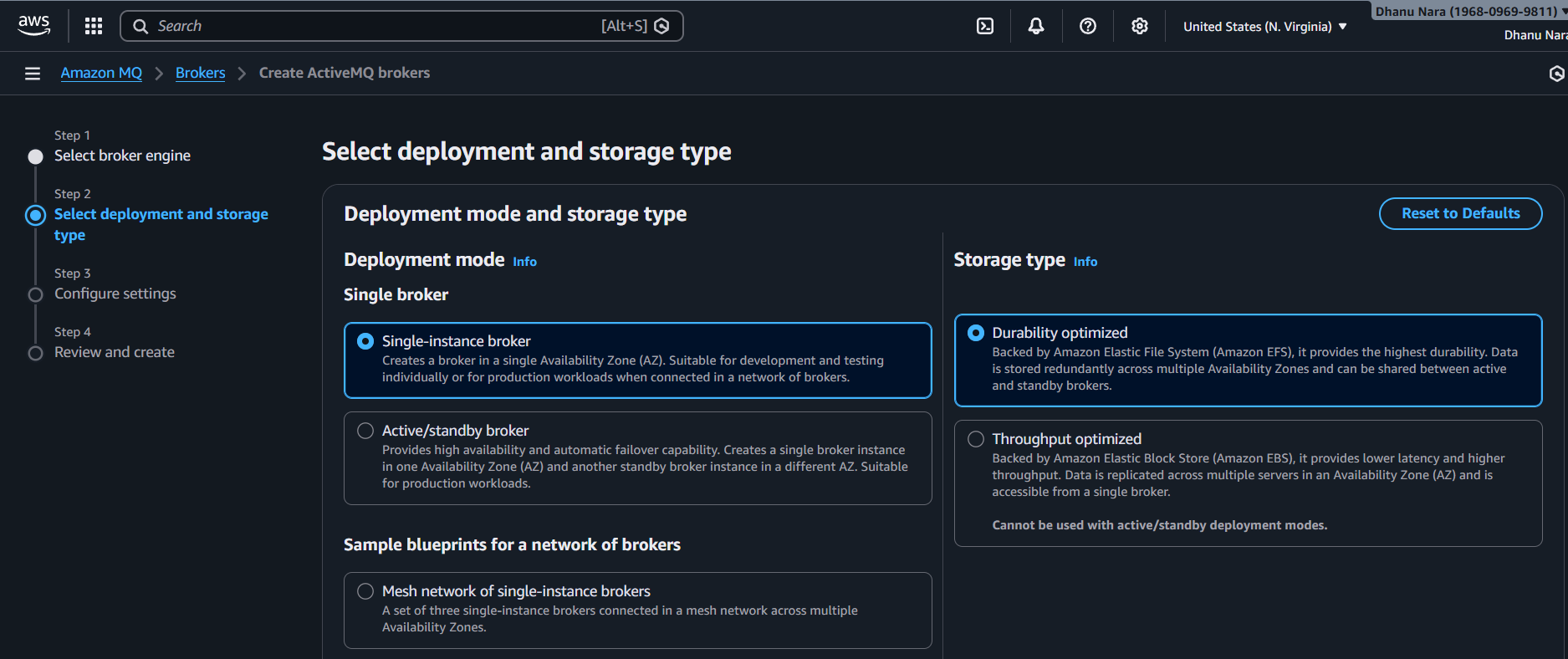


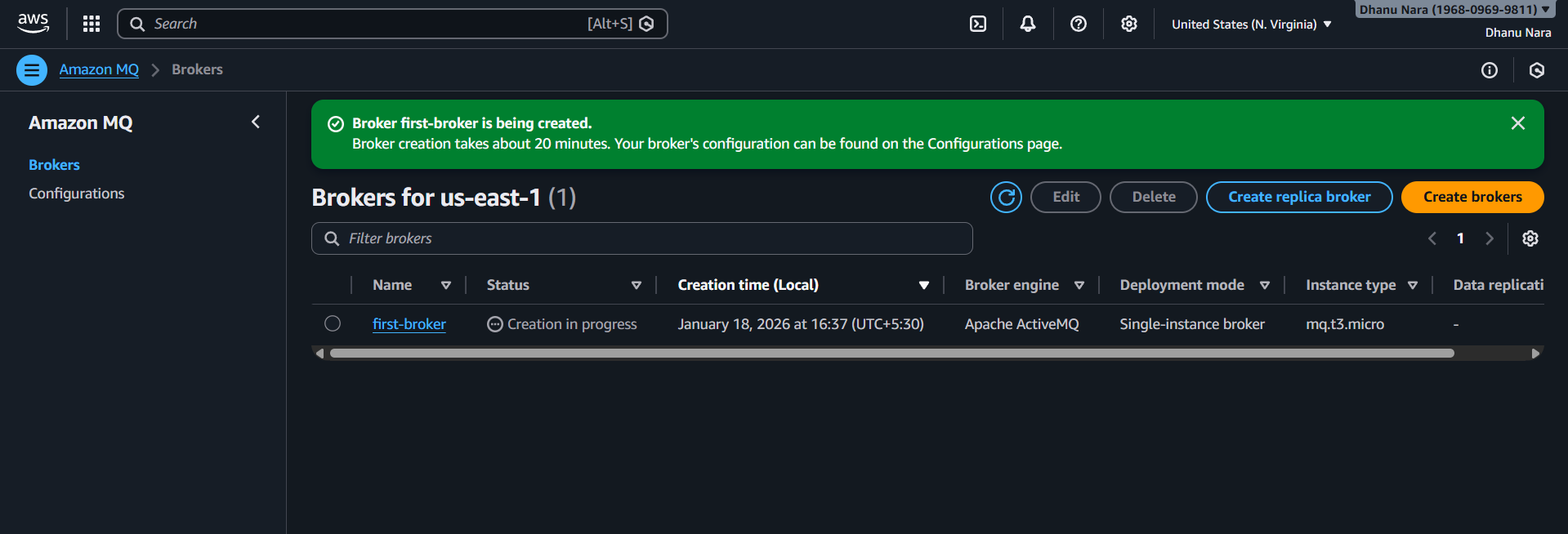


**Create Amazon MQ Broker**

1. **Broker engine**: Apache ActiveMQ
2. **Deployment mode**: Single-instance broker
3. **Broker instance type**: mq.t3.micro (free-tier friendly)
4. Click **Next**







**Configure Broker Details**

1. **Broker name**: first-broker
2. **Authentication**: Simple authentication
3. Create **username and password**  
   (Save this – required for login)
4. Click **Next**

**Network and Security**

1. **VPC**: Default VPC
2. **Subnet**: Any default subnet
3. **Security Group**:  
   Select **default security group**  
   (No new SG required)
4. Click **Next**

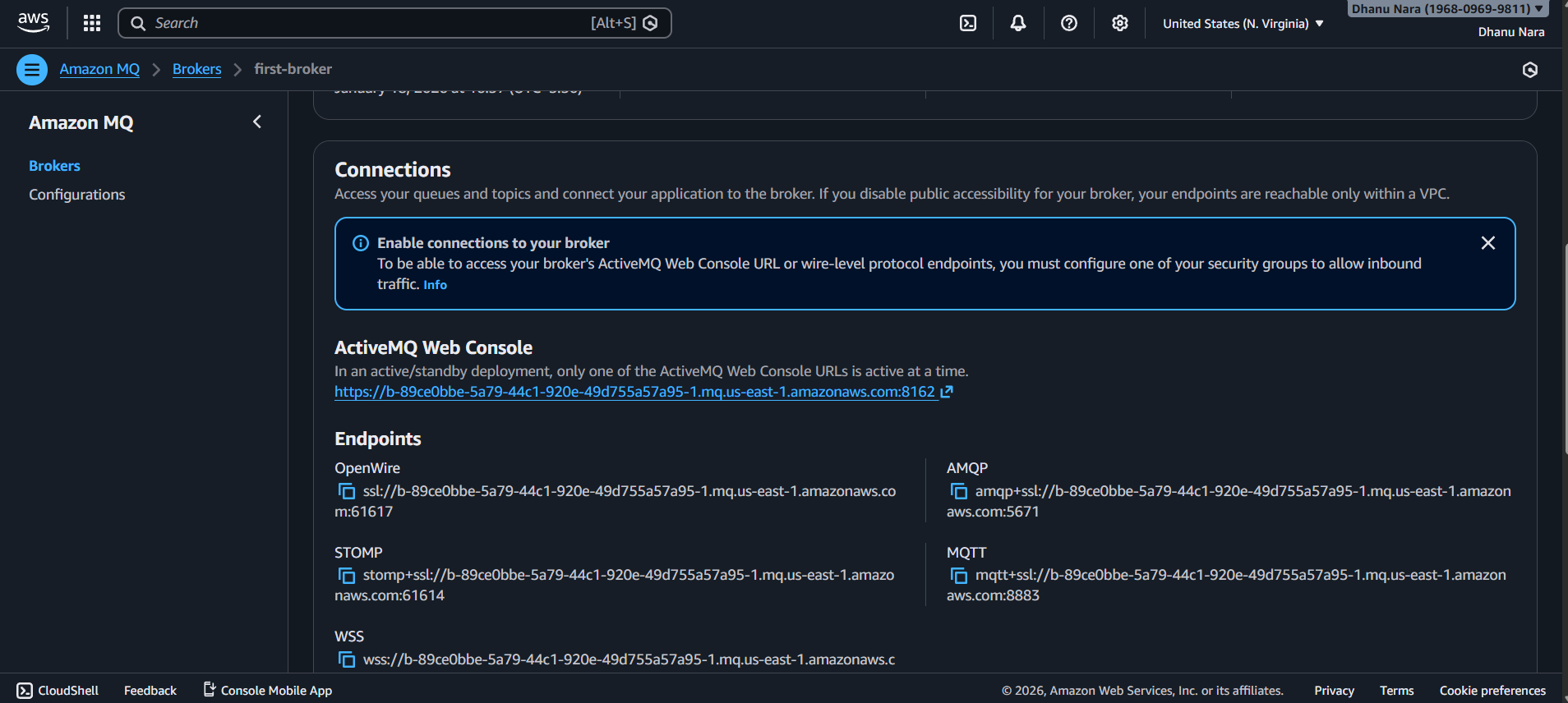
**Review and Create Broker**

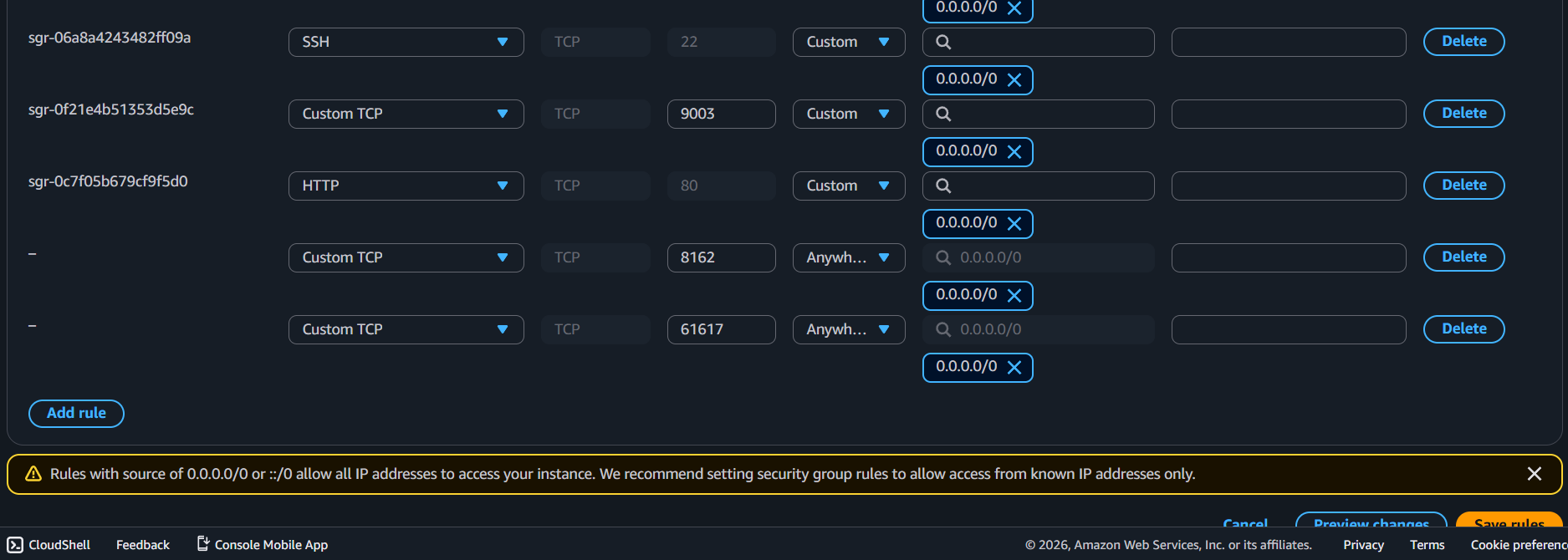
1. Review all settings
2. Click **Create broker**
3. Wait **5–10 minutes** until status becomes **Running**

**Open ActiveMQ Web Console**

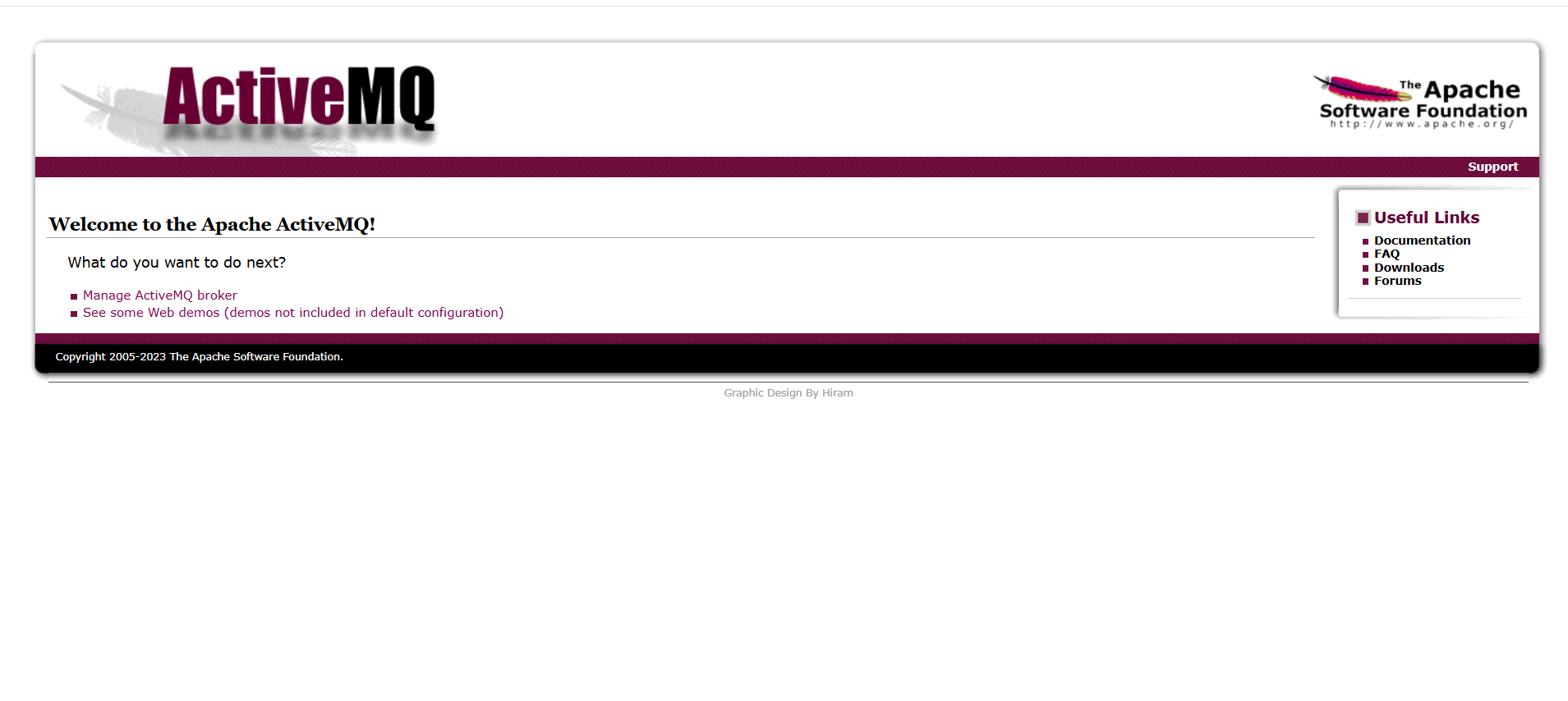
1. Open broker → first-broker
2. Scroll to **Connections**
3. Click **ActiveMQ Web Console URL** (port 8162)
4. Login using the **username/password** created earlier

✅ ActiveMQ console opens successfully

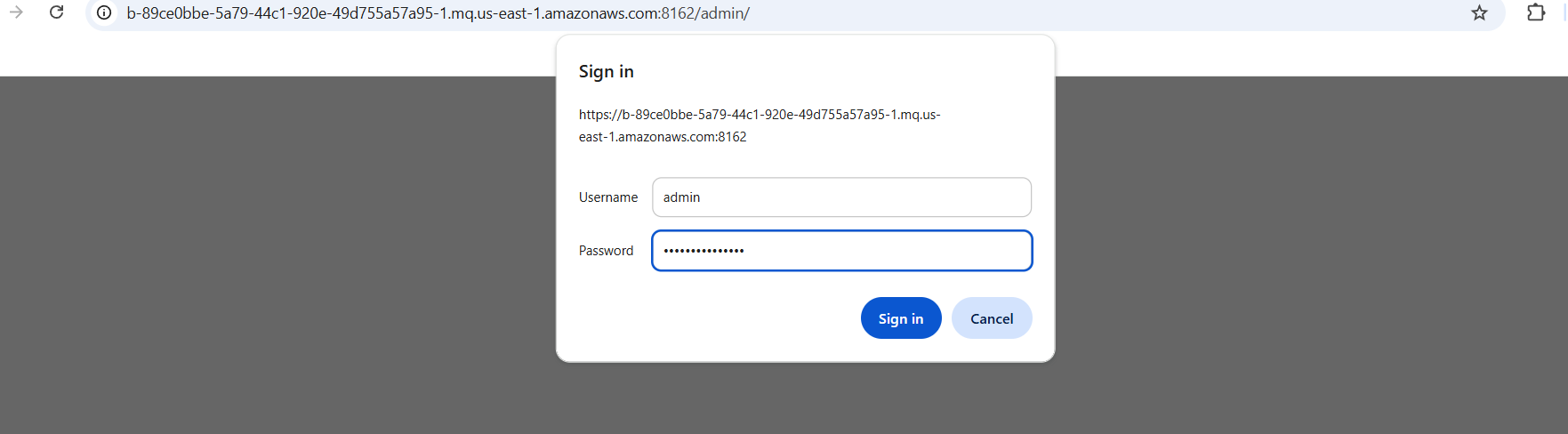


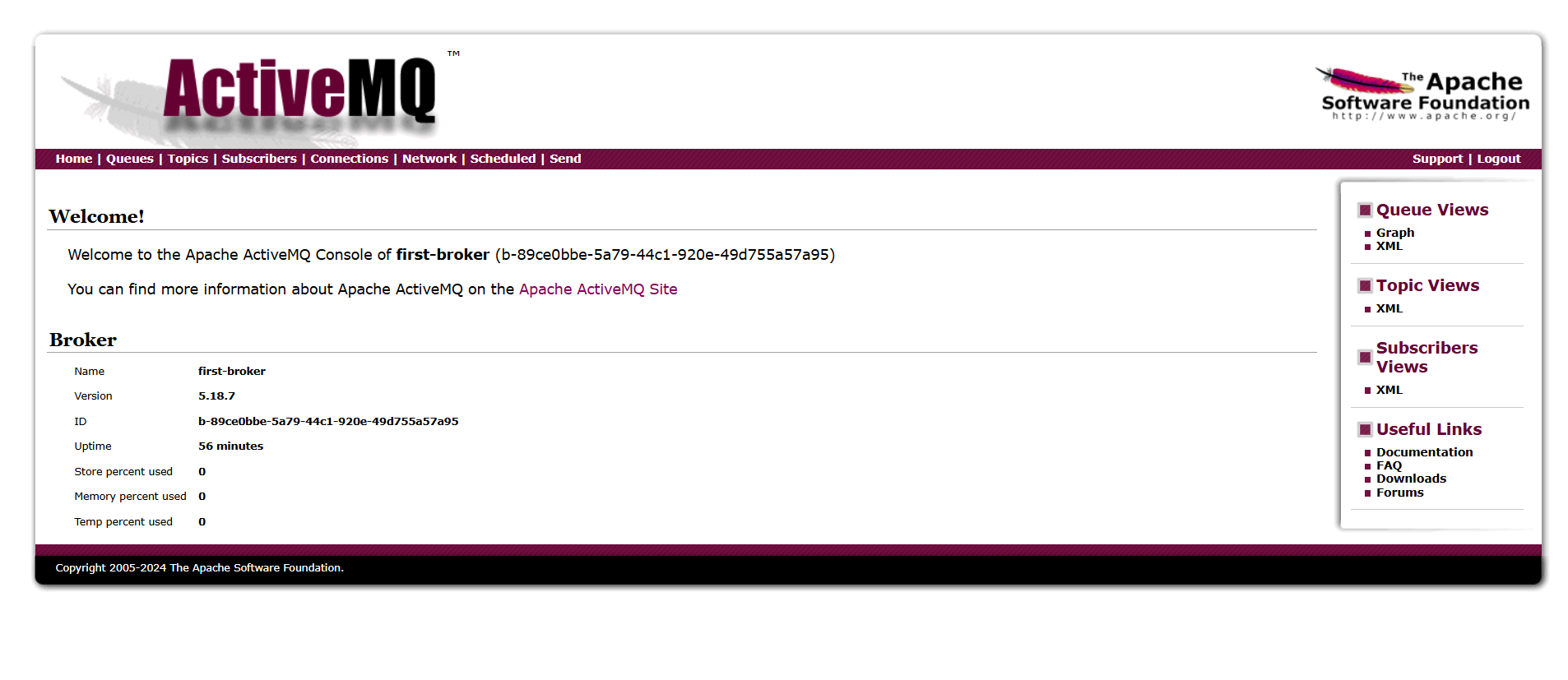


**Paste the URL under the Amazon MQ in the browser**



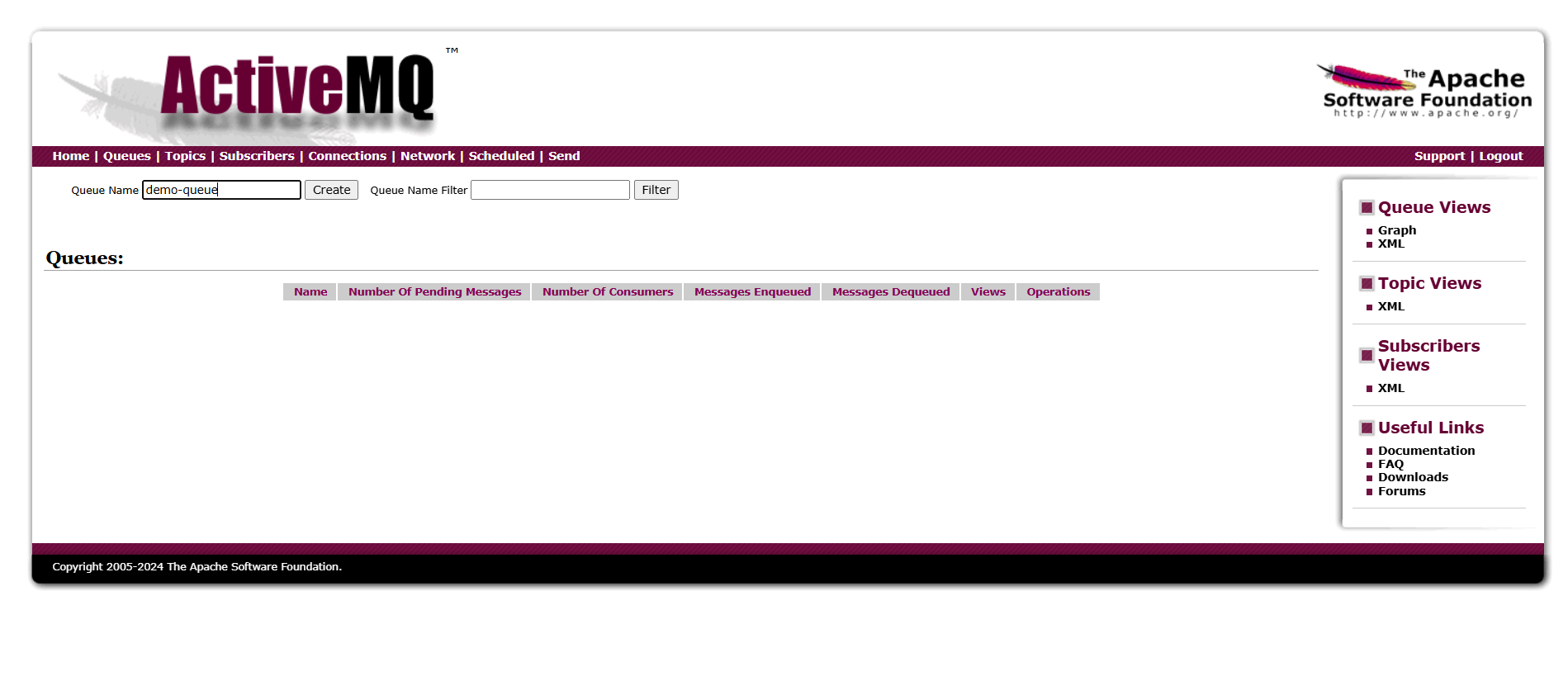
Click on manage ActiveMQ broker

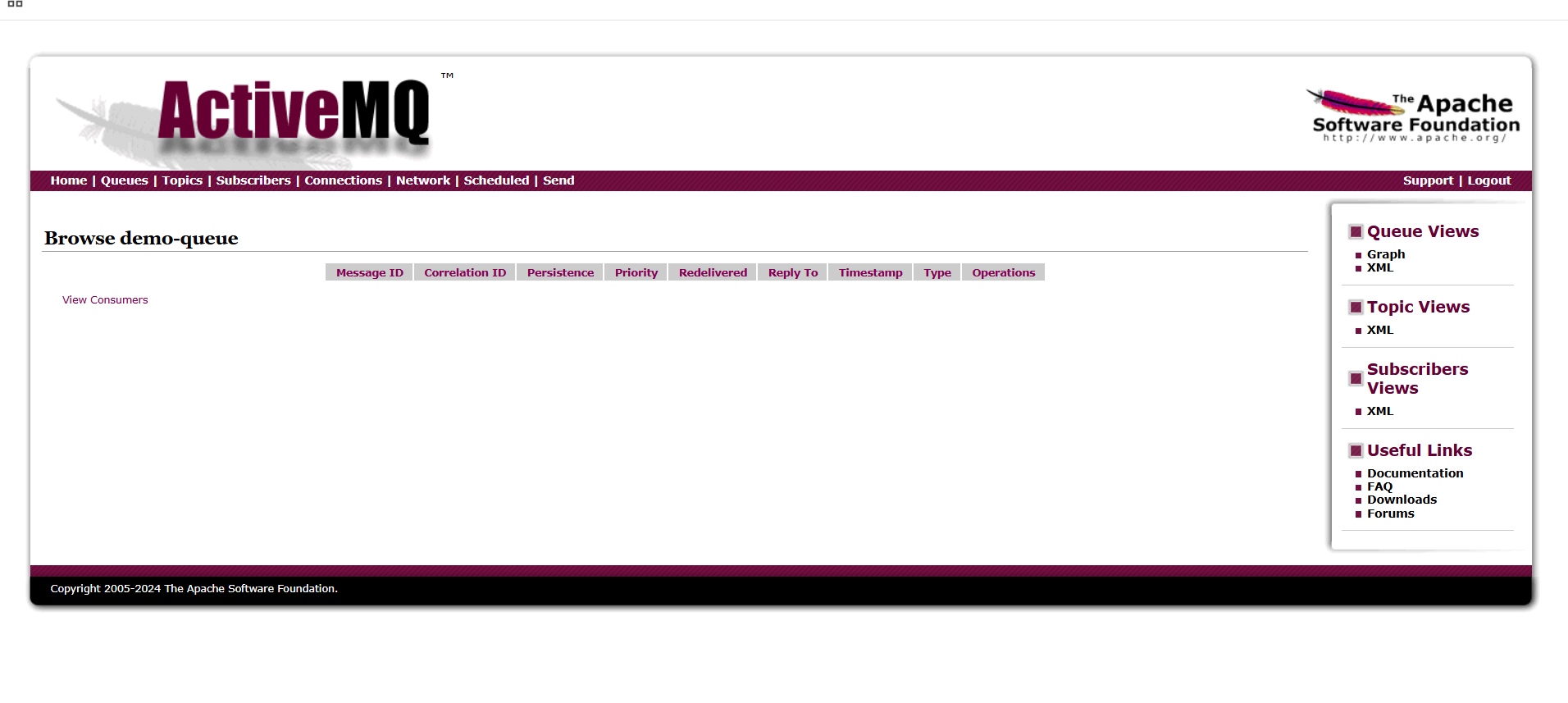




**Step 3: Create a Queue**

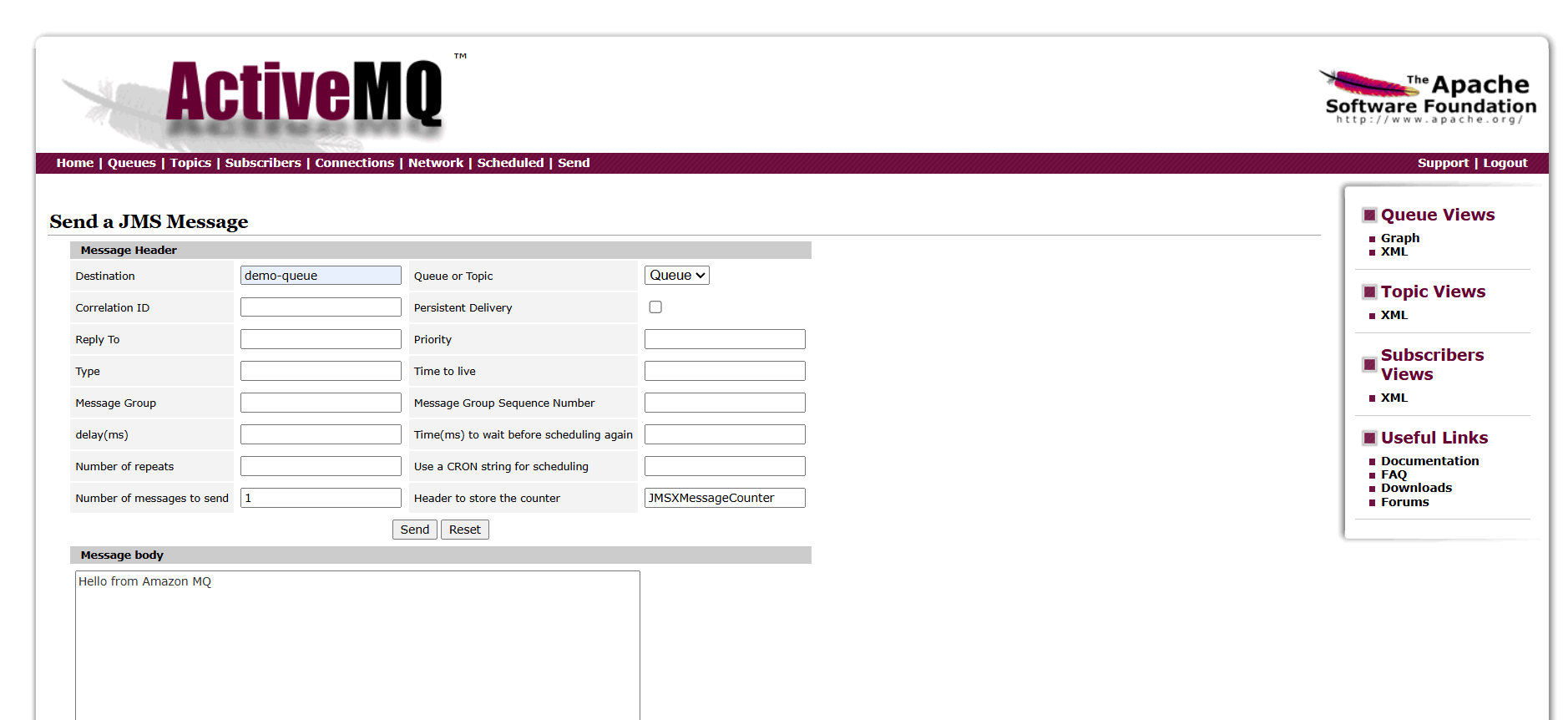
1. Click **Queues**
2. Click **Create**
3. Queue name:
4. Demo-queue
5. Click **Create**





**Send a message**

1. Click **test-queue**
2. Scroll to **Send a message**
3. Message body:
4. Hello from Amazon MQ
5. Click **Send**



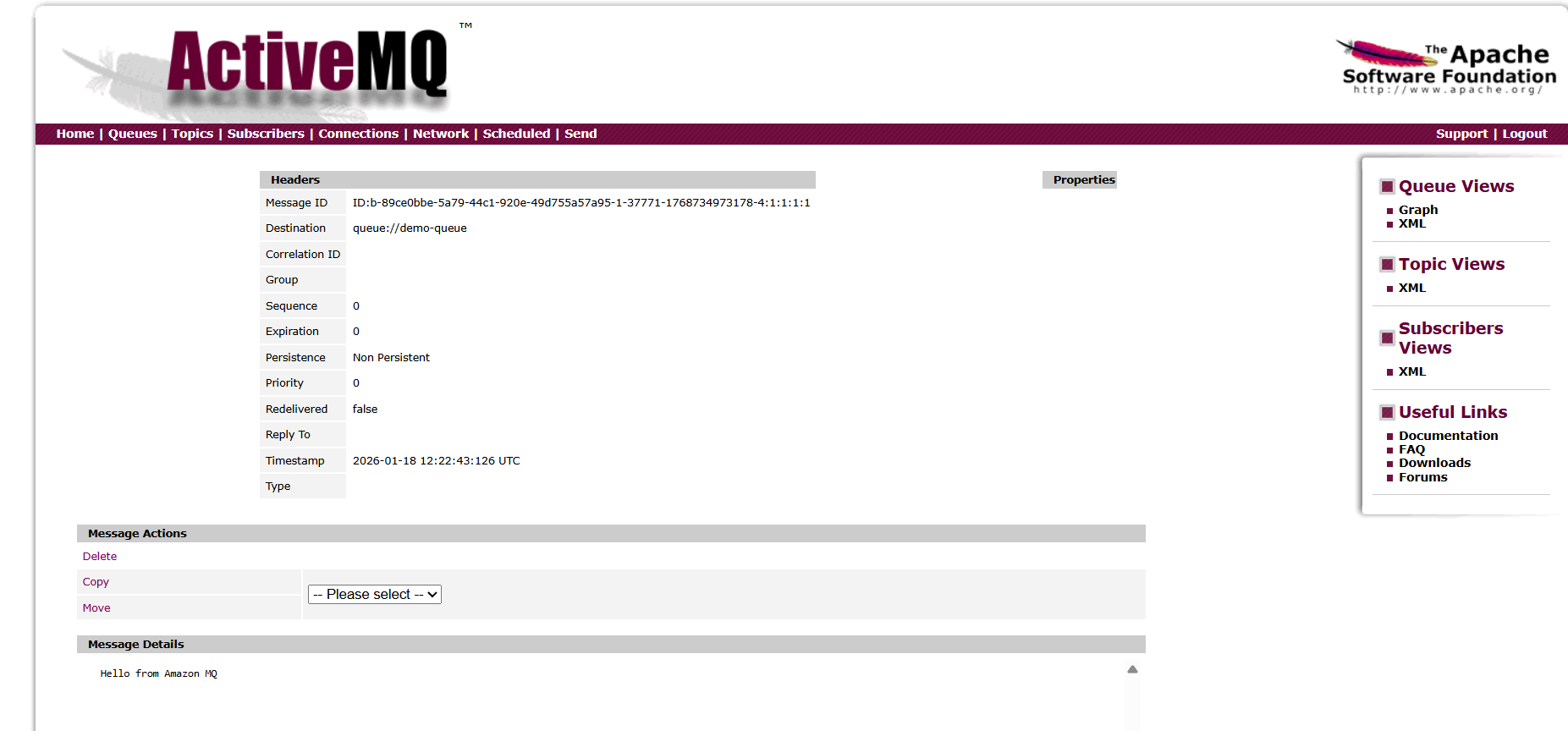
**Verify / Receive the Message**

1. Still on **demo-queue** page
2. Scroll to **Browse Messages**
3. Click **Browse**

You should see:

Hello from Amazon MQ

✅ Message successfully received



**Conclusion**

In this task, Amazon MQ was successfully configured using an ActiveMQ broker to demonstrate a complete messaging workflow. A broker was created and accessed through the ActiveMQ Web Console, where a queue (demo-queue) was used to send and manage messages. A test message was produced and sent to the queue, and its successful delivery was verified by browsing the queue and inspecting the message headers and payload. This confirms that the broker, queue, and message producer are functioning correctly. Overall, the task demonstrates how Amazon MQ enables reliable message-based communication between applications, making it suitable for decoupling services and building scalable, event-driven architectures.