

-----🔥 What Is This Project About?-----

You're taking an existing application called **vProfile** (a Java web app that used EC2, MySQL, RabbitMQ, and Memcache) and **refactoring** it — that means **rebuilding it with better architecture** using modern AWS services.

🧠 Why Refactor?

Before (Lift & Shift):

- You installed everything manually on EC2: Tomcat, MySQL, RabbitMQ, Memcache.
- You had to handle **scaling**, **backups**, **monitoring**, and **updates** manually.

Problems:

- Too much manual work.
- Difficult to scale.
- Hard to maintain.
- Not cost-efficient.

After (Refactored Architecture):

- Use **managed AWS services** like:
 - Elastic Beanstalk for Tomcat.
 - RDS for MySQL.
 - Amazon MQ for RabbitMQ.
 - ElastiCache for Memcache.
 - CloudFront for CDN.

- Route 53 for DNS.
- AWS manages **scaling, backups, monitoring**.
- You focus only on your application code.

AWS Services Used in the New Architecture

Purpose	Service	Why Use It?
Host Application	Elastic Beanstalk	PaaS to deploy Java (Tomcat) apps without managing EC2
Store App Files	S3	Store WAR files or configs
Load Balancer	Automatically via Beanstalk	Distributes traffic
Auto Scaling	Automatically via Beanstalk	Adds/removes EC2s
Database	Amazon RDS (MySQL)	Fully managed MySQL DB
Cache	Amazon ElastiCache	Managed Memcache/Redis
Message Broker	Amazon MQ (ActiveMQ)	Replaces RabbitMQ
Monitoring	CloudWatch	Monitors and auto-scales
DNS	Route 53	Domain and endpoint management
CDN	CloudFront	Faster content delivery globally

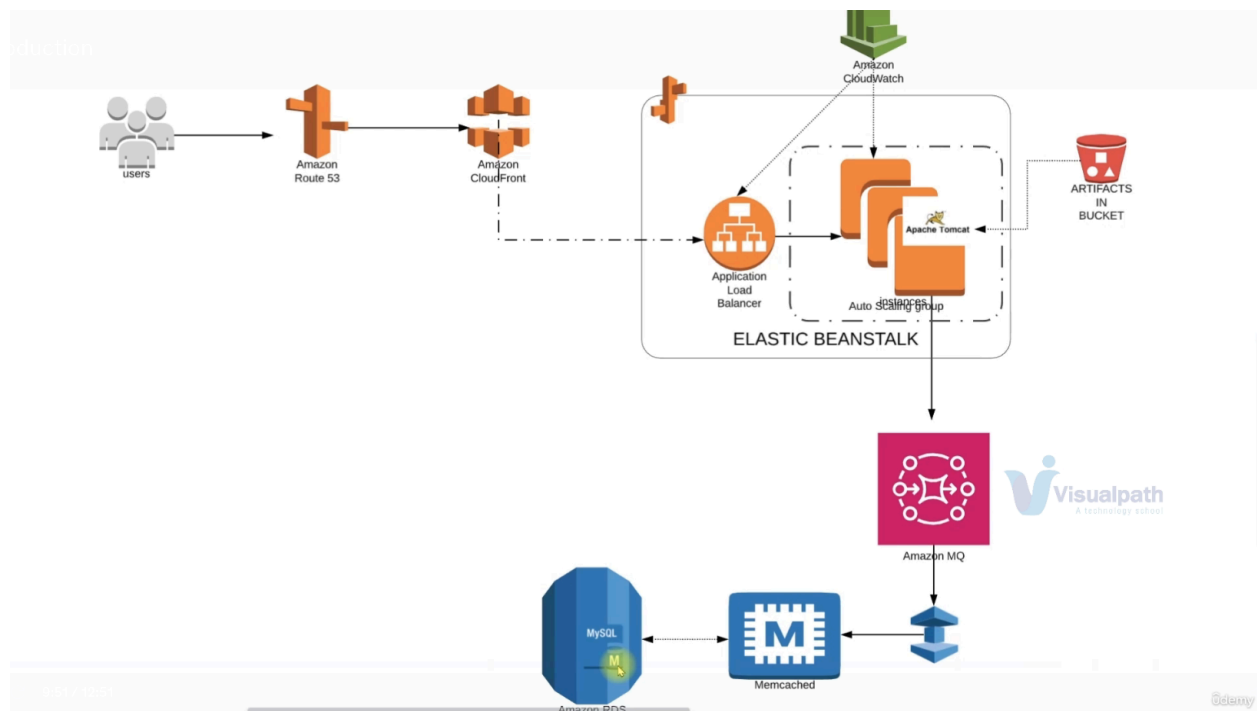
Architecture Flow

Here's what happens when a user opens your website:

1. **User enters your domain** → DNS is managed by **Route 53**
2. **Route 53** points to **CloudFront** → a CDN that caches your app globally

3. **CloudFront** forwards the request to **Elastic Load Balancer**
4. The **Load Balancer** sends it to **EC2 instance (Tomcat)** created by **Elastic Beanstalk**
5. Your Tomcat app connects to:
 - **Amazon RDS** for MySQL database
 - **Amazon MQ** for messaging
 - **ElastiCache** for caching
6. **CloudWatch** monitors everything and scales up/down automatically
7. Your app runs faster, scales easily, and you don't manage infrastructure manually.

Architecture diagram :



----- 🎯 Security groups & key pairs -----

1. Create a **backend security group** for services like RDS, ElastiCache, and Amazon MQ.
2. Configure the security group to allow internal communication.
3. Create a **key pair** to access the EC2 instance that Elastic Beanstalk will create later.

🔑 Step-by-Step: Creating the Backend Security Group

✅ Step 1: Go to EC2 → Security Groups

1. Open AWS Console.
2. Go to **EC2** → **Security Groups** → Click **Create security group**.

✅ Step 2: Enter Security Group Details

- **Security group name:** `vprofile-rearch-backend-sg`
- **Description:** `Security group for backend services in vProfile rearchitecture project`
- **VPC:** Choose your default VPC or the one you're using.

✅ Step 3: No Inbound Rules Yet

- **DO NOT** add any inbound rules while creating.
- Leave **outbound rules** as default (allow all outbound traffic).

- Click **Create security group**.
-

✅ **Step 4: Edit Inbound Rules for Internal Communication**

Now let's allow backend services to talk to each other (RDS, ElastiCache, MQ):

1. Select the newly created security group.
2. Click **Actions** → **Edit Inbound Rules**.
3. Click **Add rule**.
 - **Type**: All traffic
 - **Protocol**: All
 - **Port range**: All
 - **Source**: **Custom** → Select the same **Security Group ID** (self-reference)
 - This means: "Allow all traffic between resources inside this security group."
4. Optional: Add a description: **Allow internal communication between backend services**
5. Click **Save rules**.

✅ Done! Now backend resources can interact securely.

🔑 **Step-by-Step: Create a Key Pair for Beanstalk EC2**

Why?

Elastic Beanstalk manages everything for you, **but** if your application fails to start, or you need to inspect logs inside the EC2 instance, this key will let you **SSH into it**.

✓ Step 1: Go to EC2 → Key Pairs

1. Navigate to **EC2 → Key Pairs**
 2. Click **Create Key Pair**
-

✓ Step 2: Configure Key Pair

- **Name:** `vprofile-rearch-key`
- **Key pair type:** RSA (default)
- **Private key format:** `.pem` (for Linux/macOS) or `.ppk` (for Windows/Putty)
- Click **Create key pair**

📁 It will automatically download the key file to your system.

✓ Save this file securely — without it, you **cannot access** the EC2 instance if needed.



RDS

We are provisioning a **MySQL RDS instance** using best practices:

- Create a **custom parameter group**.
 - Create a **DB subnet group**.
 - Launch a **MySQL RDS instance** with internal-only access (no public IP).
 - Store connection credentials securely for future use.
-

🔧 Step-by-Step Summary

✓ 1. Create a Custom Parameter Group

Why? RDS doesn't give shell access (like `my.cnf`), so parameter changes are managed through **parameter groups**.

- Go to RDS Console → **Parameter groups** → **Create parameter group**
 - Fill out:
 - **Name:** `vprofile-rds-rearch-paragrp`
 - **Description:** (same or descriptive)
 - **Engine type:** `MySQL`
 - **Parameter group family:** `mysql8.0`
 - **Type:** `DB Parameter Group`
 - Click **Create**
 - You can now go into it and tweak ~28 pages of settings (only for advanced DB use cases).
-

✓ 2. Create a DB Subnet Group

Why? RDS needs to be placed in **at least 2 subnets in different AZs** (even for single instance). Best practice for networking.

- Go to **Subnet groups** → Click **Create DB Subnet Group**
- Fill out:
 - **Name:** `vprofile-rds-rearch-subgrp`
 - **Description:** `Subnet group for vprofile`
 - **VPC:** Your default or custom VPC

- **Subnets:** Select 2 or more subnets across different AZs
 - Click **Create**
-

✓ 3. Create the RDS Instance

Now time to launch the actual **MySQL DB** that our app will connect to.

Go to **RDS Dashboard** → **Create Database**

Choose:

- **Creation method:** **Standard Create**
- **Engine:** **MySQL**
- **Version:** **8.0.xx** (any 8.0 is fine)
- **Template:** **Free tier**

Settings:

- **DB identifier:** **vprofile-rds-rearch**
- **Master username:** **admin**
- **Password:** Choose **Auto-generate** → *Make sure to save it!*

Instance Type:

- **db.t3.micro** or **db.t4g.micro** (Free-tier eligible)
- **Storage:** 20 GB, **gp3**
- **Disable auto-scaling** (unchecked)

Connectivity:

- **VPC:** default
- **Subnet group:** Select the one we just created
- **Public access:** **No** (internal only)
- **Security group:** Select the previously created `vprofile-rearch-backend-sg`
- **Port:** `3306`

Additional Config:

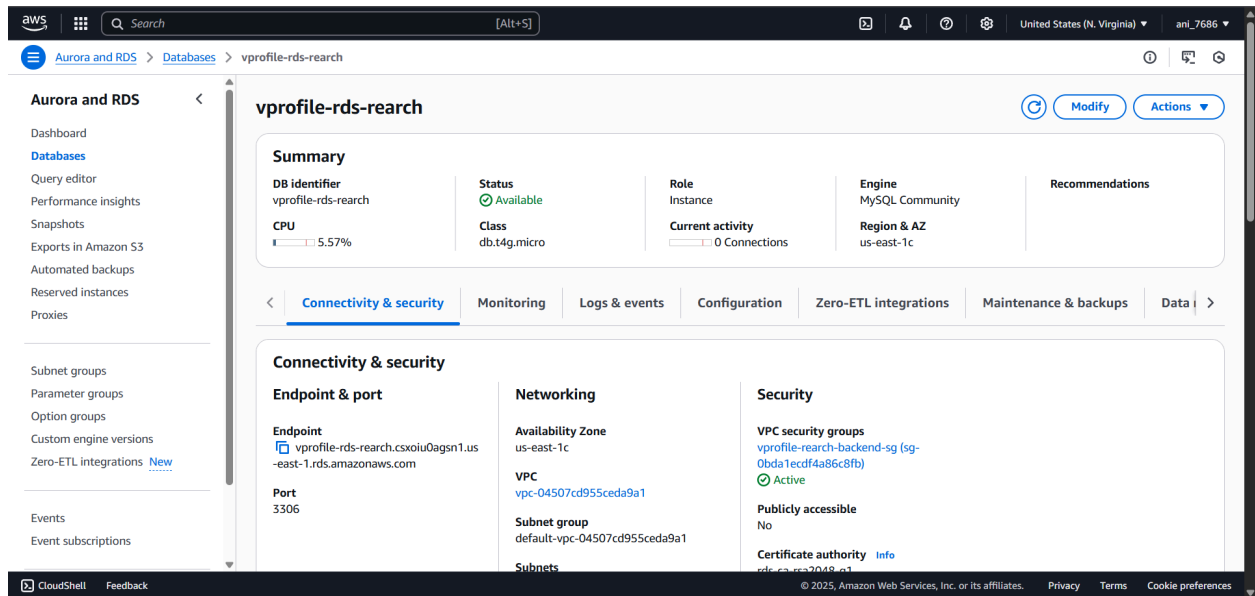
- **Database name:** `accounts` (must match app requirement)
- **Authentication:** Password only
- **Monitoring/Logs:** Optional (can skip for this project)
- **Maintenance:** Leave auto minor upgrade on (depends on your project)

 Click **Create Database**



Important: Save Credentials

- After creation starts, you'll see a **popup with username and password**.
- Save it in a safe location (sticky notes, Notepad, password manager).
- If you lose the password, **you must reset it** — can't recover the original.



ElasticCache

Set up **ElastiCache (Memcached)** to support fast in-memory caching for the vProfile app. We'll do this using:

1. A **parameter group**
2. A **subnet group**
3. A **cache cluster** with all best practices and free-tier-safe settings.



Step-by-Step Setup of ElastiCache (Memcached)




1. Create a Parameter Group




Used to control Memcached behavior (config options like `max_connections`, etc.)

- Go to **ElastiCache > Parameter Groups**

- Click **Create parameter group**
 - **Name:** `vprofile-rearch-paragrp`
 - **Description:** Same or "vprofile parameter group"
 - **Family:** `memcached 1.6`
- Click **Create**

 Like in RDS, this gives you a bunch of config options. You can leave defaults.

✓ 2. Create a Subnet Group

 Determines where in the VPC your cache nodes will be deployed (must span multiple AZs)

- Go to **Subnet Groups** → Click **Create**
 - **Name:** `vprofile-rearch-subgrp`
 - **Description:** Subnet group for vprofile
 - **VPC:** Default (or your custom one)
 - **Subnets:** All AZs/subnets selected by default (verify under "Manage")

✓ Click **Create**

✓ 3. Create Memcached Cluster

Go to **ElastiCache Dashboard** →

Click **Create Cache** → **Choose Memcached** → Click **Design your own cache**

Then follow these detailed steps:

 **Basic Settings:**

- **Name:** `vprofile-rearch-cache`
- **Engine version:** `1.6.x` (1.6 is mandatory, patch version like .22, .17 etc. is fine)
- **Port:** `11211` (required by application)
- **Parameter group:** Select `vprofile-rearch-paragrp` created above

Node Settings:


- **Node type:** `cache.t2.micro` (half GB RAM, smallest + free-tier safe)
- **Number of nodes:** `1` (for project purposes; real apps may use more)

Network:

- **Subnet group:** Choose `vprofile-rearch-subgrp`
- **AZ preference:** No preference (1 node only, so no HA needed)
- **Security group:** Use `vprofile-rearch-backend-sg` created in earlier lectures

Notifications:

- **SNS topic:** Skip / Disable notifications

 Click **Next**, review settings, and then click **Create Cluster**

Recap Table

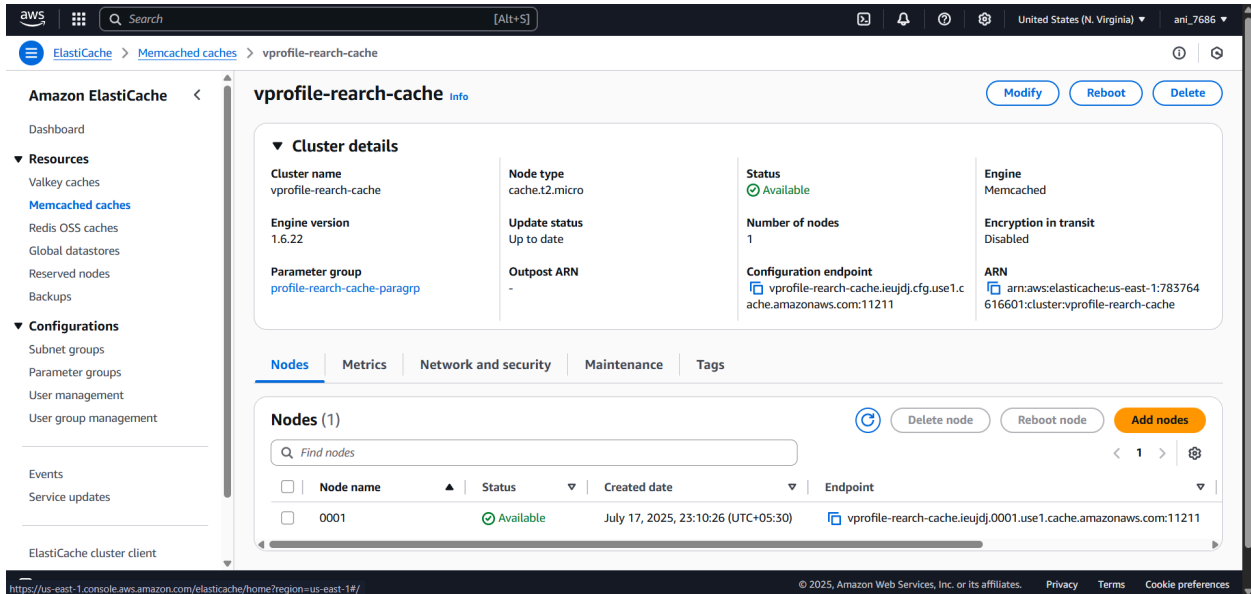
Component

What You Did

Parameter Group Created with Memcached 1.6 for config control

Subnet Group Covers all AZs for Memcached nodes

Cache Cluster Single node, `cache.t2.micro`, port `11211`, backend SG, private network only



Amazon MQ

Set up **RabbitMQ (message broker)** using the **Amazon MQ** managed service instead of manually installing and managing it on EC2.

✅ **Fully managed**, easier, safer, and integrates cleanly with the AWS ecosystem.

Why Use Amazon MQ for RabbitMQ?

- No need to set up and maintain RabbitMQ on your own EC2 instance.
- AWS handles:

- Patching
 - Failover (if using clusters)
 - Monitoring
 - High availability (if required)
- You only provide minimal setup and credentials.
-

Steps to Set Up RabbitMQ via Amazon MQ

✓ 1. Search and Launch Amazon MQ

- Go to AWS Console
 - Search for **Amazon MQ**
 - Click **Get started**
-

✓ 2. Select RabbitMQ as Broker Engine

- Select **RabbitMQ** (not ActiveMQ)
 - Click **Next**
-

✓ 3. Select Deployment Type

- Choose: **Single-instance broker**
 - Cheapest

- No high availability
- Best for testing and non-critical workloads
- In real-time production, you'd use **clustered deployment** for HA (high availability)

✓ 4. Configure Broker

Field	Value
Name	<code>vprofile-rearch-rabbitmq</code>
Instance Type	<code>mq.t3.micro</code>
Engine Version	<code>3.13</code> (or any available 3.x)
Username	<code>rabbit</code>
Password	(Choose 12+ characters, secure)

🔒 Save the username and password securely (you'll need it in `application.properties`)

✓ 5. Additional Settings

Setting	Value
Broker Configuration	Default
CloudWatch Logs	Disabled for this project (optional)
Access Type	Private (used within VPC only)
VPC	Default
Subnets	Auto-selected within VPC
Security Group	Use <code>vprofile-rearch-backend-sg</code>
Encryption	Default (AWS-managed)
Maintenance	Default (optional auto minor upgrades)

✓ 6. Final Steps

- Click **Next**
- Review all configurations
- Click **Create broker**

📌 It will take a few minutes to create.
Once complete, you'll get the **RabbitMQ broker endpoint**.

The screenshot shows the AWS Management Console interface for an Amazon MQ broker. The breadcrumb navigation indicates the path: Amazon MQ > Brokers > vprofile-rearch-rabbitmq. The left sidebar shows 'Amazon MQ' with sub-links for 'Brokers' and 'Configurations'. The main content area displays the details for the broker 'vprofile-rearch-rabbitmq', which is in a 'Pending modifications' state. The details are organized into four columns: Specifications, CloudWatch Logs, Security and network, and Maintenance. The Specifications column lists the ARN, Broker name, Broker status (Creation in progress), Broker instance type (mq.t3.micro), Deployment mode (Single-instance broker), and Broker engine (RabbitMQ). The CloudWatch Logs column shows the General log is disabled. The Security and network column lists the VPC, Subnet(s), Security group(s), and Public accessibility. The Maintenance column shows the Automatic minor version upgrade is enabled and the Maintenance window is Wednesday 17:00 - 19:00 UTC. A 'Pending modifications' section at the bottom of the Maintenance column shows the Broker configuration is pending configuration.

Specifications	CloudWatch Logs	Security and network	Maintenance
ARN arn:aws:mq:us-east-1:783764616601:broker:vprofile-rearch-rabbitmq:b-57541e1f-c598-4d2e-94c0-fd8673dab904	General Disabled - Logs	VPC vpc-04507cd955ceda9a1	Automatic minor version upgrade Yes
Broker name vprofile-rearch-rabbitmq		Subnet(s) subnet-066dea58033a2de34	Maintenance window Wednesday 17:00 - 19:00 UTC
Broker status Creation in progress		Security group(s) sg-0bda1ecdf4a86c8fb	Pending modifications Broker configuration Pending configuration
Broker instance type mq.t3.micro		Public accessibility No	
Deployment mode Single-instance broker			
Broker engine RabbitMQ			

🎯 DB Initialization

Since **RDS is private** and not publicly accessible, we:

- Launch a **temporary EC2 instance** in the **same VPC**.
- Install MySQL client + Git.
- Connect to RDS.
- **Clone the vprofile source code** from GitHub.
- **Import the SQL schema** into RDS.

🔧 Step-by-Step Summary

✅ 1. Verify RDS Availability

- Go to **RDS Console**
 - Confirm status is **Available**
 - **Endpoint** (e.g., `vprofile-db.xxxxxx.rds.amazonaws.com`)
 - **Port**: 3306
 - **Database name**: `accounts`
 - **Username**: `admin`
 - **Password**: (from earlier sticky note)
-

✓ 2. Launch EC2 Instance for MySQL Client

Setting	Value
Name	<code>mysql-client</code>
AMI	Ubuntu Server 24.04
Instance Type	<code>t2.micro</code> (Free tier)
Key Pair	Use existing or create new
Security Group	<code>vpro-mysql-client-sg</code>
Inbound Rule	SSH (Port <code>22</code>) from My IP

Launch the instance.

✓ 3. Allow MySQL Access from Client to RDS

- Go to **Security Groups** → Find `vprofile-rearch-backend-sg`
 - Edit **Inbound Rules**
 - Add rule:
 - **Type:** MySQL/Aurora
 - **Port:** 3306
 - **Source:** `vpro-mysql-client-sg` (your new EC2 SG)
 - 💡 This allows EC2 to access private RDS.
-

✓ 4. SSH into EC2 & Install Tools

```
ssh -i path/to/key.pem ubuntu@<public-ip>
```

```
sudo -i # become root
```

```
apt update && apt install mysql-client git -y
```

✓ 5. Connect to RDS

```
mysql -h <rds-endpoint> -u admin -p accounts
```

```
# or this, but NOT recommended:
```

```
mysql -h <endpoint> -u admin -pYourPassword accounts
```

Once logged in, type:

```
show databases;  
  
use accounts;  
  
show tables;  
  
exit;
```

✓ 6. Clone vProfile Source Code

```
git clone https://github.com/hkcoder/vprofile-project.git  
  
cd vprofile-project  
  
git checkout awsrefactor
```

✓ 7. Initialize DB with SQL Schema

```
mysql -h <rds-endpoint> -u admin -p accounts <  
src/main/resources/db_backup.sql
```

Connect to the RDS MySQL database named `accounts` using the `admin` user, and **execute all the SQL statements** found inside `src/main/resources/db_backup.sql` to set up the database schema (i.e., create tables, insert initial data, etc).

Then verify:

```
mysql -h <rds-endpoint> -u admin -p accounts  
  
show tables;
```

✓ You should see multiple tables created — schema initialized!

✓ 8. Cleanup: Terminate EC2

Once DB is ready:

- Go to **EC2 Console**
- Select **mysql-client**
- Click **Instance State** → **Terminate**



Key Concepts Recap

Concept	Explanation
RDS is Private	So EC2 in same VPC is required to connect

Security Group Rule Required for **RDS SG to allow EC2 SG** on port 3306

Temporary EC2 Used only for importing schema; safe to terminate after

Git + MySQL Client Needed to download source and import SQL

✓ Elastic Beanstalk

Elastic Beanstalk is a **Platform-as-a-Service (PaaS)** that automates:

- EC2
- Auto Scaling Group
- Load Balancer
- S3 (for app artifact)
- CloudWatch (monitoring)
- IAM roles
- Deployment lifecycle

It simplifies deployments—**no need to manually set up EC2, target groups, ALB, etc.**

Steps Covered in This Lecture

1. IAM Role Creation

Before creating the Beanstalk environment, you **create a custom IAM role** manually with proper permissions because the default one lacks required permissions.

📌 Required IAM policies:

- `AdministratorAccess`
- `AWSElasticBeanstalkFullAccess`
- `AWSElasticBeanstalkCustomPlatformforEC2Role`
- `AWSElasticBeanstalkWebTier`
- `AWSElasticBeanstalkMulticontainerDocker`

➡ Role name: `vprofile-rearch-bean-role`

The screenshot shows the AWS IAM console interface. The left sidebar contains navigation links for Identity and Access Management (IAM), Access management, and Access reports. The main content area displays the details for the role 'vprofile-rearch-bean-role'. The 'Summary' tab is active, showing the role's ARN, creation date, and instance profile ARN. The 'Permissions' tab is also visible, showing a list of attached policies.

Policy name	Type	Attached entities
AdministratorAccess-AWSElasticBeanstalk	AWS managed	1
AWSElasticBeanstalkCustomPlatformfor...	AWS managed	1

2. Creating Beanstalk Application

🔧 Basic Details:

- Application Name: e.g., `vprofile-rearch`

- **Environment Name:** e.g., `vprofile-env`
 - **Domain:** Choose a unique subdomain (e.g., `vpro-rearch.us-east-1.elasticbeanstalk.com`)
-

3. Platform Configuration

- **Platform:** Tomcat 10 with Java Corretto 21
 - Use **custom configuration** (not sample app)
-

4. Service Role & Instance Profile

- **Service Role:** Created just now
 - **Instance Profile Role:** The one you manually created
 - **Key Pair:** For SSH access to EC2 instances
-

5. VPC, Subnets & Public IP

- Use **default VPC**
 - Enable **Public IP** for instances
 - Select **all subnets**
-

6. Database (RDS)

- Don't create a new RDS inside Beanstalk
- Use the RDS already created and initialized

7. Tags

- e.g., `project: vprofile`

8. Storage Volume

- Use **gp3 volume** (not container default) to force Beanstalk to use **Launch Template** (required for newer setups)

9. Load Balancer & Capacity

- Choose **Application Load Balancer**
- Public visibility
- Health check stickiness enabled
- **Min 2, Max 4 instances**
- Instance Type: `t2.micro`

10. Auto Scaling Triggers

- Based on **CPU Utilization** and **Network Out**

11. Deployment Policies

Different types of deployment policies in Beanstalk:

Policy Name	Behavior	Notes
All at once	Fastest, but downtime	Not recommended for production
Rolling	Few instances at a time	Safer
Rolling with Additional Batches	Add extra instances during update	Costly, but zero downtime
Immutable	Launch full new set, swap traffic	Safest, most expensive
Traffic Splitting	Canary testing with % traffic to new version	Best for A/B testing or safe prod deployments

➡ For this project, "**Rolling with 50%**" was chosen.

12. Health Reporting & Monitoring

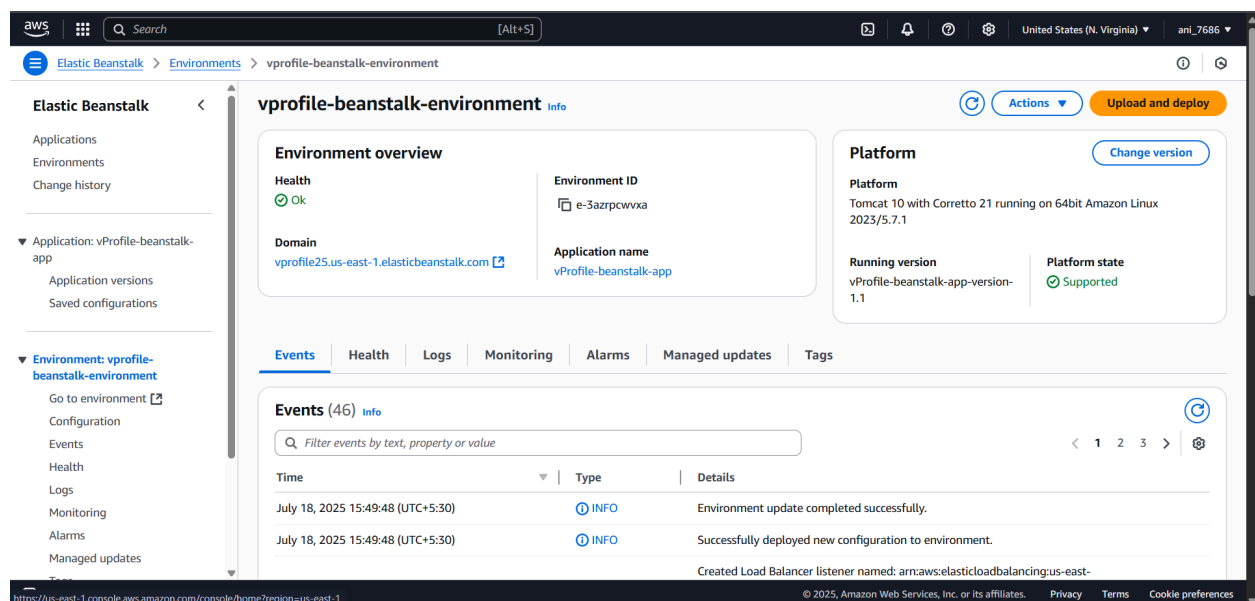
- **Enhanced Monitoring** (recommended for multi-instance deployments)
 - Set update strategy and notifications
 - Set **environment variables** if needed (e.g., database host, port, user) — **not used in this lecture**
-

Final Step:

Submit the configuration and **Beanstalk will create** the environment with:

- Auto Scaling Group
- Load Balancer
- EC2 Instances (Tomcat)
- S3 bucket
- CloudWatch monitoring
- IAM roles

Once the environment is ready, you can upload the **artifact (WAR file)** and deploy it.



----- **Update on security groups & ELB** -----

You now have the following components ready:

- **RDS (Database)**
- **Amazon MQ (RabbitMQ)**

- **Memcached**
 - **Elastic Beanstalk environment** (running default app)
-

Goal of this Lecture:

To **allow the Beanstalk application (EC2 instances)** to connect to **backend services (RDS, MQ, Memcached)** by modifying **security group rules**.

Steps to Allow Beanstalk App to Access Backend:

1. Identify the Security Group of Beanstalk Instances

- Go to the **EC2** → **Instances** section.
- Select any of the **Beanstalk-created instances**.
- Go to **Security** tab.
- Click on the **Instance security group** (NOT load balancer).
- Copy the **Security Group ID**.

2. Update Backend Security Group Rules

- Go to **EC2** → **Security Groups**.
- Find the **Vprofile backend security group** (should be tagged/named accordingly).
- Click on it → Go to **Inbound rules**.
- **Add rule:**
 - **Type:** All traffic (or specific port if you want tighter control)
 - **Source:** **Custom** → Paste **Beanstalk instance security group ID**
 - **Description (optional):** *"Allow traffic from Beanstalk instance SG"*

- Click **Save rules**.

✓ VProfile App Deployment to Elastic Beanstalk with Backend Services

1. Gather Backend Information

- **RDS (MySQL)**: Get the **endpoint**, **username**, **password** (default port **3306**).
- **Amazon MQ (RabbitMQ)**: Copy **connection URL**, **port 5671**, **username/password**.
- **ElastiCache (Memcached)**: Copy the **configuration endpoint** (default port **11211**).

2. Clone & Setup Code

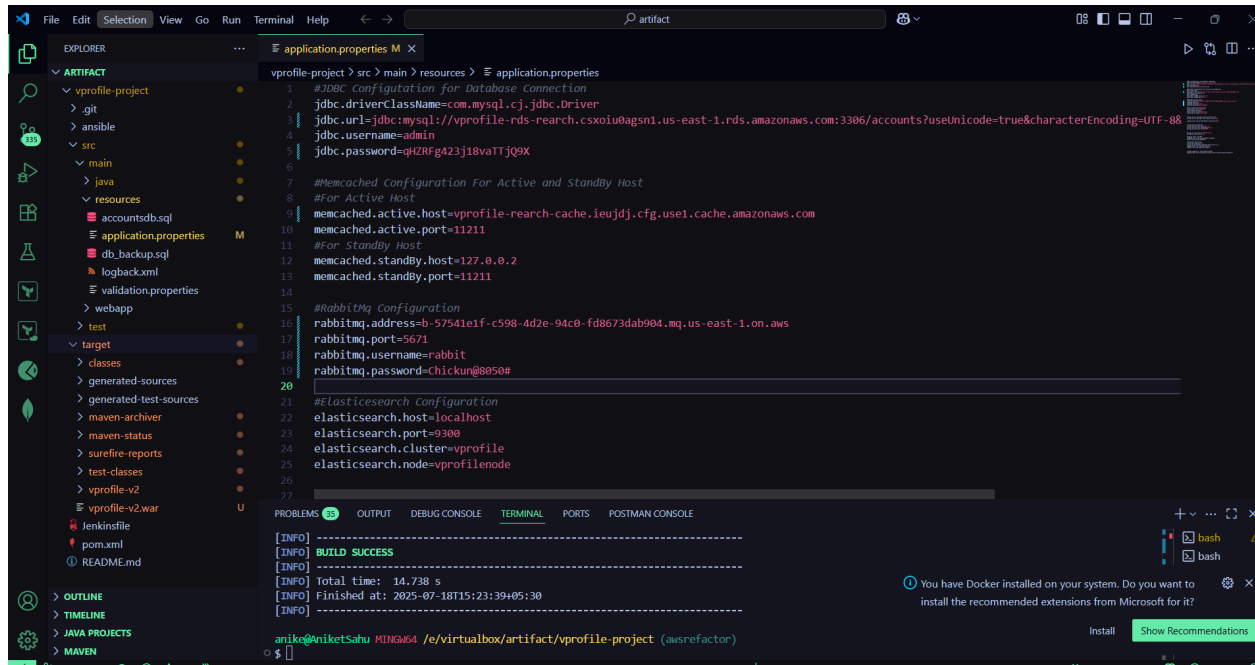
- GitHub repo: <https://github.com/hkhcoder/vprofile-project>
- Clone in VSCode → switch to branch **awsrefactor**.

3. Update **application.properties**

Modify the following fields under:

`src/main/resources/application.properties`

- Replace placeholders with actual backend service endpoints.
- Change port numbers if required (e.g., RabbitMQ from **5672** → **5671**).
- Paste correct credentials.



4. Build the Artifact

- Open terminal → set default terminal to **Git Bash** (or terminal of your choice).
- Check versions:
 - `mvn -version` → should be Maven 3.9.9+
 - Java version should be **17+**

Run:

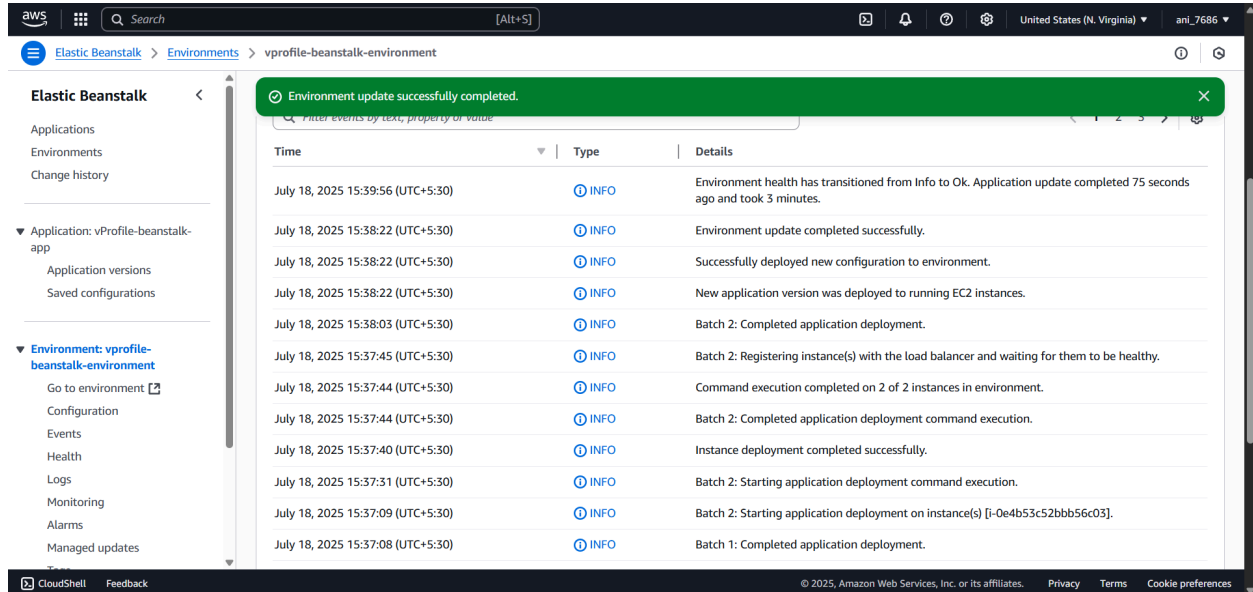
`mvn install`

- Output WAR: `target/vprofile-v2.war`

5. Upload WAR to Beanstalk

- Navigate to your Elastic Beanstalk environment.
- Click **Upload and Deploy** → Choose WAR file.
- Provide version name, e.g., `vprofile-research-beanapp-version-1.9`.

- Keep **Rolling Deployment** (batch size 50%).



6. Monitor Deployment

- Watch deployment progress in **Events**.
- Go to **Target Group** in Load Balancer:
 - One instance becomes **draining** → **unhealthy** → redeployed → back to **healthy**.
 - Then the second instance follows.

7. Access App via Beanstalk URL

Test: Open the environment URL → Login using:

Username: admin_vp

Password: admin_vp

•

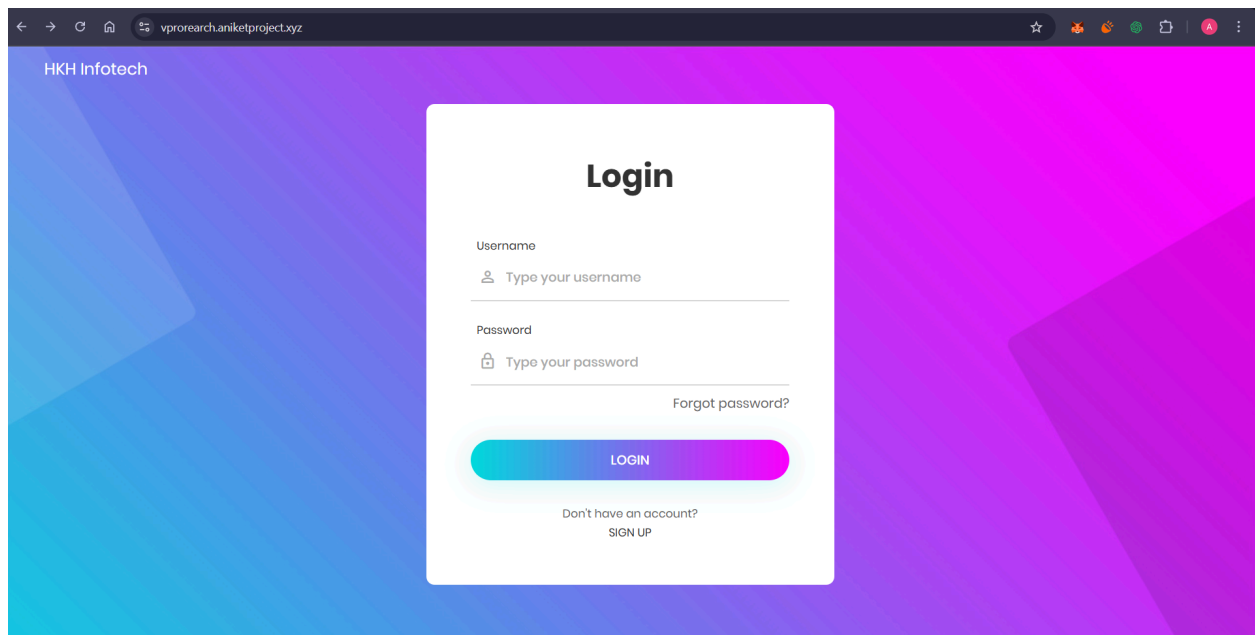
8. Set Up HTTPS (Secure Connection)

- Add HTTPS listener on port 443 with ACM Certificate.

- Go to Configuration → Load balancer → Add Listener (HTTPS 443).
- Choose existing SSL certificate (e.g., [hkhinfotech.xyz](#)).
- Save & Apply changes.





9. Create Custom Domain (Optional)

- In GoDaddy (or your registrar):
 - Create **CNAME** record:
 - Name: [vproresearch](#)
 - Value: Beanstalk environment endpoint
 - Final URL: <https://vproresearch.yourdomain.com>



10. Final Validation

- Open secure domain.
- Verify:

-  Login page loads
-  App connects to RDS, Memcached, and RabbitMQ
-  No backend errors
-  HTTPS secure (SSL cert is valid)

Takeaways:

- Full-stack deployment using **AWS managed services**: RDS, ElastiCache, Amazon MQ.
- Application is **cloud-native** with externalized config and HTTPS support.
- You now understand real-world deployment with **Elastic Beanstalk**, including blue/green-like deployment using **rolling updates**.

CloudFront

Why Are We Using AWS CloudFront?

Your app is hosted in **North Virginia (USA)**. If someone from **India** or **Europe** tries to access your site, the data has to travel all the way from the US, which can be **slow**.

That's why we use **AWS CloudFront** — a **Content Delivery Network (CDN)**.

CloudFront Benefits:

- It stores (caches) copies of your website content in **AWS edge locations** around the world.
 - When a user accesses your site:
 - CloudFront serves it from the **nearest location**, making it **faster**.
 - Your original server gets **less traffic** (reduced load).
-

What Did You Just Do?

✅ Step 1: Remove Old Domain Mapping

Earlier, your domain (vproresearch.hkxinfoteck.xyz) was pointing directly to the **Elastic Beanstalk load balancer**.

You removed this mapping in **GoDaddy DNS**.

✅ Step 2: Create CloudFront Distribution

You opened AWS > searched **CloudFront** > clicked **Create Distribution**, then filled out:

Setting	Value
Origin domain	Your Elastic Beanstalk load balancer URL
Origin protocol	Match viewer (supports both HTTP and HTTPS)
Viewer protocol policy	HTTP and HTTPS
Allowed methods	All methods (GET, POST, PUT, DELETE...)
Cache settings	Legacy (cache headers, cookies, query strings)
WAF (Firewall)	Skipped (to avoid cost)
Edge locations	All (to serve globally)

Alternate domain name (CNAME) `vproresearch.hkhinfoteck.xyz`

SSL Certificate Selected from ACM (must match your domain)

You clicked **Create**, and AWS started **deploying the distribution**.

This takes **5–15 minutes**.

✓ **Step 3: Add Domain Mapping in GoDaddy**

Once CloudFront was created, AWS gave you a domain like:

CopyEdit

`d1abcdefg1234.cloudfront.net`

You went to GoDaddy > DNS settings and added a **CNAME record**:

Field	Value
Name	<code>vproresearch</code>
Type	<code>CNAME</code>
Value	<code>d1abcdefg1234.cloudfront.net</code> (your CloudFront URL)

This means:

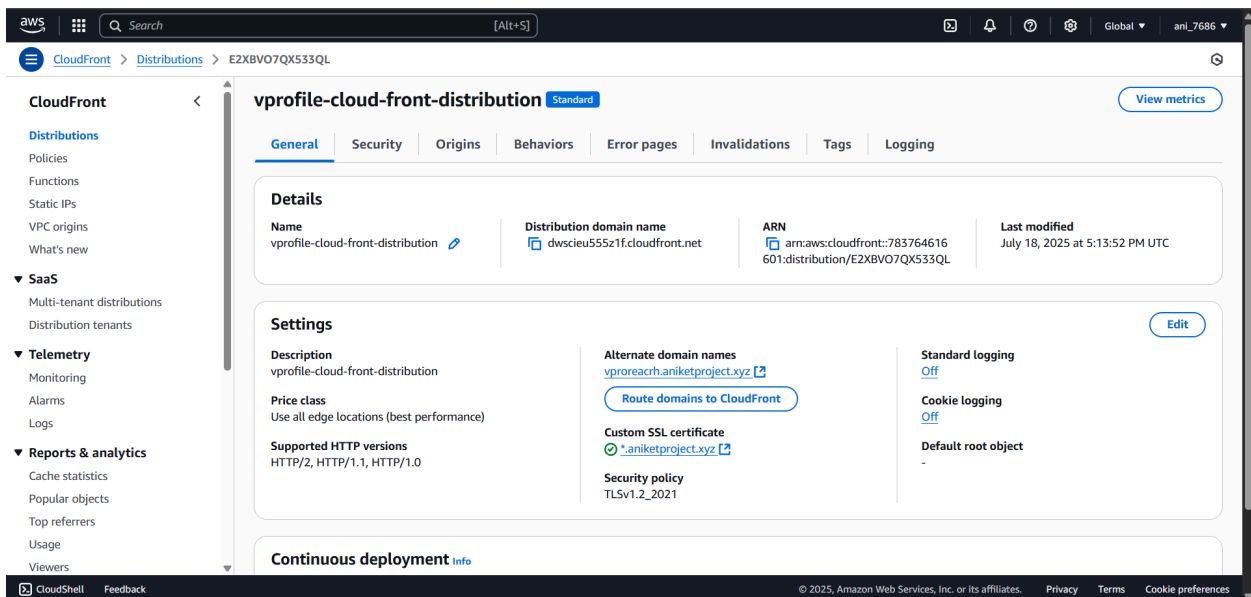
When a user goes to `https://vproresearch.hkhinfoteck.xyz`, they are now **routed through CloudFront**.

How to Test If It's Working

Wait for CloudFront Status: Deployed

- Go to AWS > CloudFront > Your distribution.
- Status should be:
State: Enabled
Last Modified: (few minutes ago)

If it's still "In progress", wait for a few more minutes.



The screenshot shows the AWS CloudFront console interface. The left sidebar contains navigation links for CloudFront, Distributions, Policies, Functions, Static IPs, VPC origins, and What's new. The main content area displays the configuration for a distribution named 'vprofile-cloud-front-distribution'. The 'General' tab is selected, showing details such as the distribution domain name, ARN, and last modified date. The 'Settings' section includes options for alternate domain names, price class, supported HTTP versions, custom SSL certificate, security policy, standard logging, cookie logging, and default root object. A 'Continuous deployment' section is also visible at the bottom.

How to Verify CloudFront is Working

You just set up CloudFront, but how do you confirm that it's **actually being used** when users visit your site?

Steps:

1. **Open a private/incognito window** in a different browser (like Firefox if you were using Chrome earlier).
2. Press **F12** to open Developer Tools.
3. Go to the **Console** or **Network** tab.

In the browser address bar, go to your domain:

`https://vproresearch.hkhinfoteck.xyz`

4. In the Network tab, click on the main GET request (your page).
5. Go to **Headers** > Scroll down to look for ****Via**** header.



If it shows something like:

`via: 1.1 abcdefg.cloudfront.net (CloudFront)`



It means the request went **through CloudFront**, and content was served either from:

- The **nearest cache (edge)** if it's already stored
- Or fetched from the **load balancer** and **cached for future users**

----- 🏗️ Final Architecture Review -----

Here's the **refactored AWS architecture** for your VProfile app:

User



Route 53 (DNS)



CloudFront (CDN with caching and global edge locations)



Elastic Load Balancer (from Beanstalk)







Beanstalk-managed EC2 instances (your app)



Backend services:

- ↳ Amazon RDS (MySQL DB)
- ↳ Amazon MQ (Message Queue)
- ↳ ElastiCache (Memcached)
- ↳ S3 (artifact storage)

Benefits:

-  Fast global access
-  Secure with SSL via ACM
-  Managed services = less maintenance
-  Scalable & monitored with CloudWatch