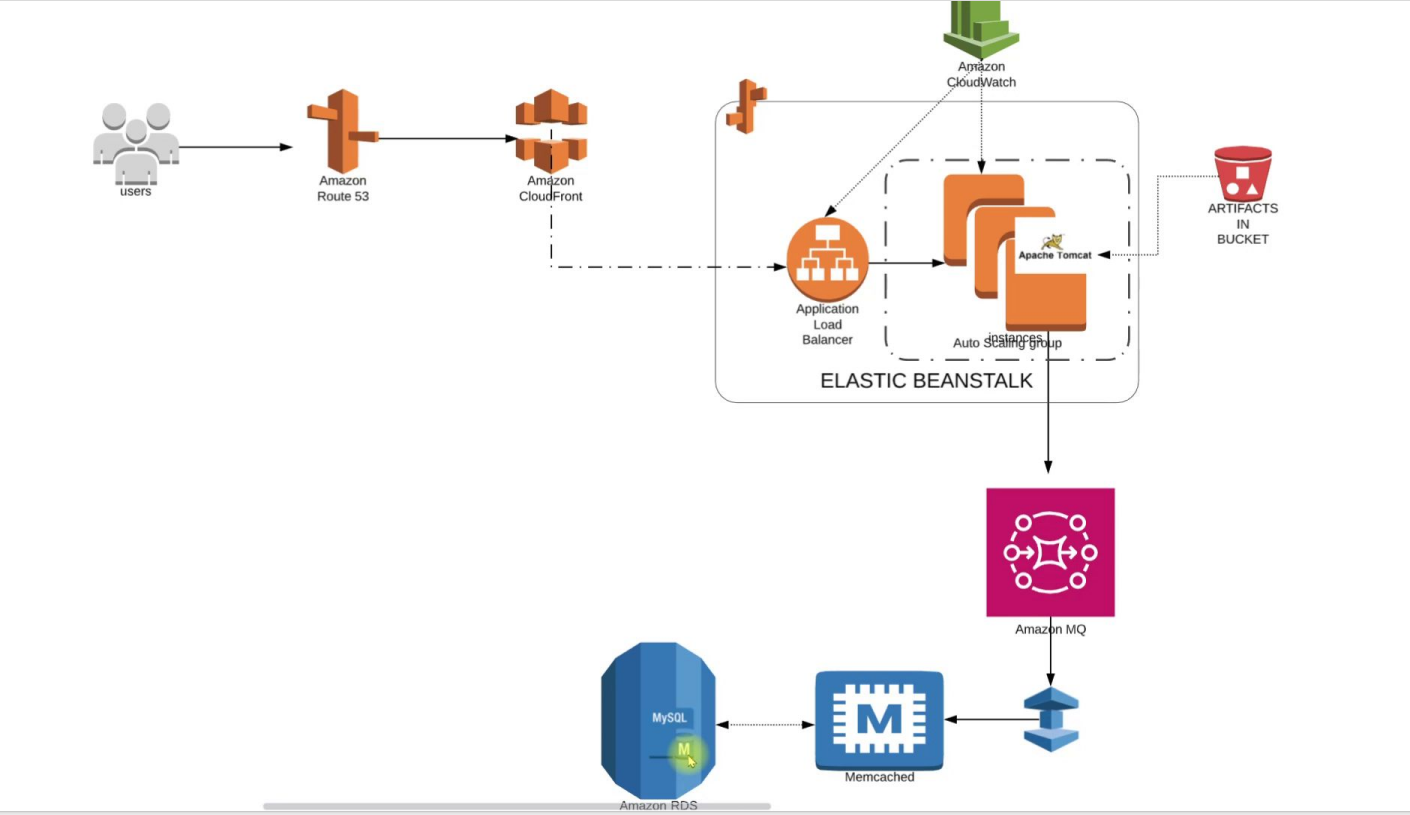
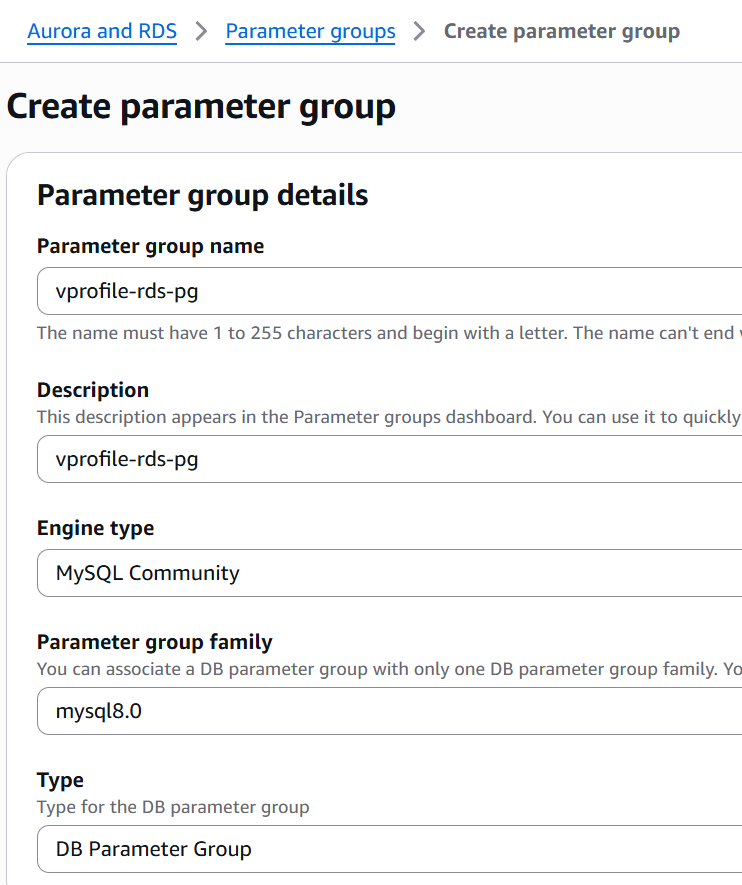
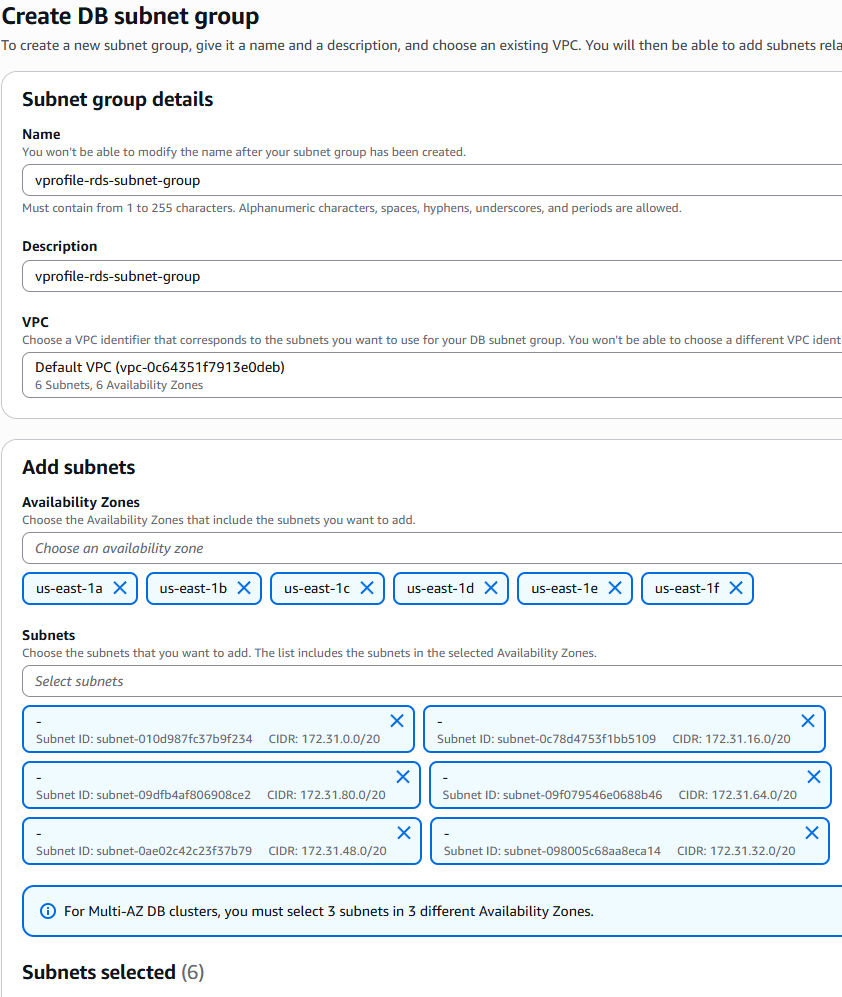
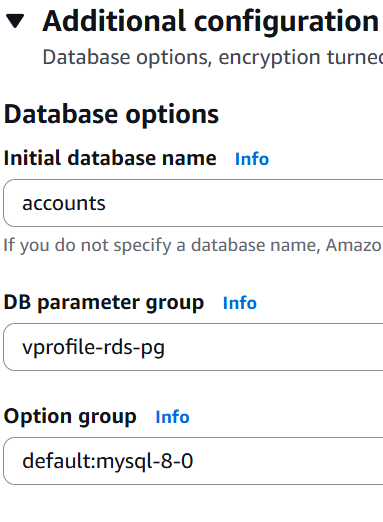
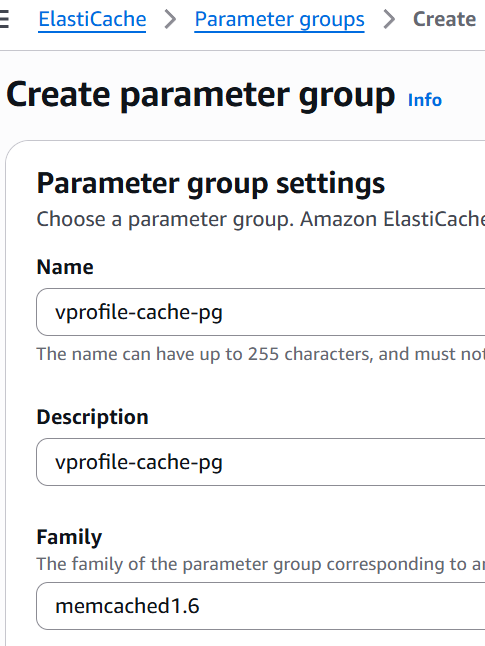
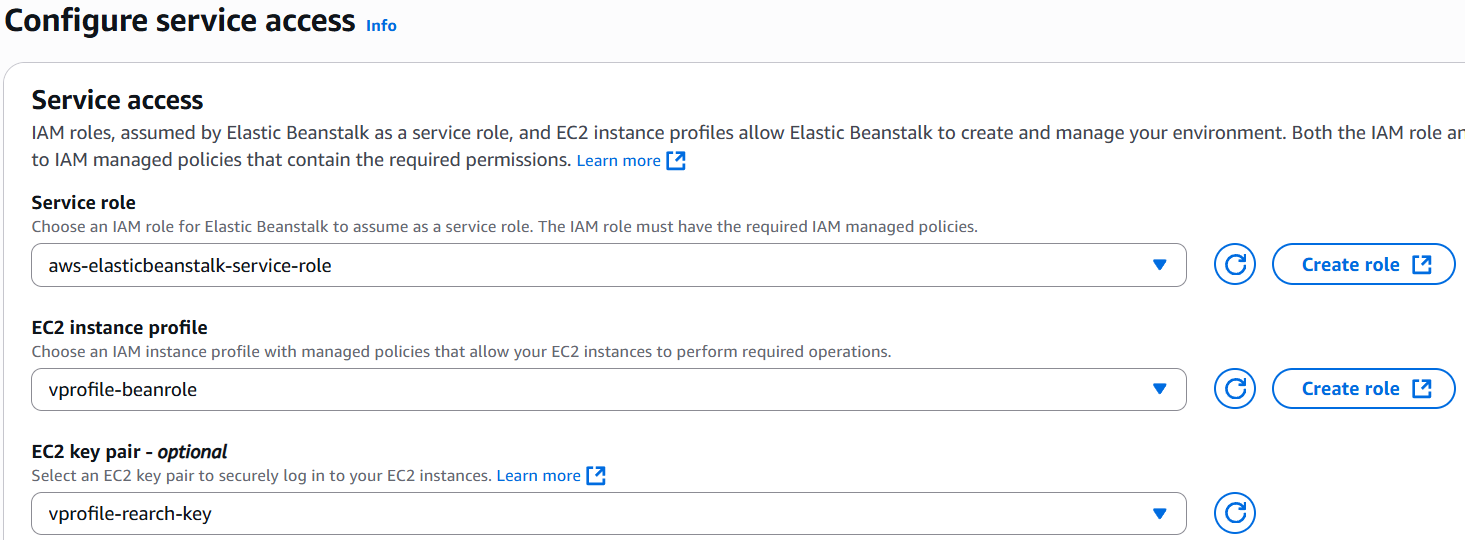
* Different types of **SERVICES**:
  + IaaS (Infrastructure as a Service):
    - It provides raw computing resources i.e. servers, networking, storage -- on demand.
    - You manages: OS, runtime, data, applications
    - Provider manages: Physical servers, virtualization, networking, storage
    - Examples:
      * AWS EC2
      * GCE (Google Compute Engine)
      * Azure Virtual Machines
    - Devops usecase:
      * Provisioning servers quickly for deployments, running custom environments, hosting databases, setting up CI/CD agents.
  + PaaS (Platform as a Service):
    - It provides a ready-to-use platform for building, testing, and deploying applications. You don’t worry about OS or infrastructure — only your code and configurations.
    - You manage: Applications & data.
    - Provider manages: OS, middleware, runtime, scaling, infrastructure..
    - Examples:
      * AWS Elastic Beanstalk
      * Google App Engine
      * Heroku
      * Azure App Service
    - Devops usecase:
      * Deploying apps quickly without managing servers — great for microservices, CI/CD pipelines, and rapid testing.
  + SaaS (Software as a Service):
    - It provides fully functional end-user applications over the internet. No setup, no server — just use the software.
    - You manage: Nothing (just use the app).
    - Provider manages: Everything (infrastructure, platform, software, updates).
    - Examples:
      * Gmail
      * Google Docs
      * Slack
      * Jira
      * Dropbox
    - Devops usecase:
      * Using SaaS tools for collaboration, project management, monitoring (e.g., Datadog), or CI/CD (e.g., GitHub Actions).
  + Analogy:
    - IaaS → Renting an empty apartment (you bring furniture, design, etc.).
    - PaaS → Renting a fully furnished apartment (you just bring your stuff and live).
    - SaaS → Booking a hotel room (everything is ready; you just check in and use).
* Services that are gonna used:
  + Beanstalk
    - It’ll manage the EC2 instances.
    - It has load balancer and auto scaling as well.
  + S3/EFS
    - Storage
  + RDS Instances
    - Databases
  + Elastic cache
    - In place of Memcached
  + Active MQ
    - In place of Rabbit MQ
  + Route 53
    - For DNS
  + Cloudfront
    - For CDN
      * A CDN is a network of servers located in different regions that cache and deliver website content (images, CSS, JS, videos, HTML) to users from the server closest to them.
      * Example: If your site is hosted in the US, a user in India will get data from a nearby CDN server in India, not directly from the US — making it load much faster.
* ****
* **Quick Overview:**
  + Here user will access out URL which will be resolved to an end-point in **Route 53** (before it was happening inside **GoDaddy**).
    - That end-point will be of **Amazon CloudFront (CDN)** which will cache so many things to serve the Global audience.
  + **Elastic Beanstalk:**
    - After cloud-front, it’ll fo to **Application Load Balancer** (which is a part of **Elastic Beanstalk**).
    - After that it’ll go to the **EC2** instances(which is part of **Auto Scaling Group**).
    - Means Entire frontend will be managed by Beanstalk.
  + For backend;
    - **Amazon MQ** instead of **RabbitMQ**
    - **Elastic Cache** instead of **Memcache**
    - **RDS** instead of **database running on EC2** instances.
* Requirements:
  + Keypair:
    - For beanstalk instance login
  + Security Groups:
    - 1. Elastic cache
    - 2. RDS
    - 3. Active MQ
  + Create :
    - RDS
    - Amazon Elastic Cache
    - Amazon Active MQ
  + Create Elastic Beanstalk Environment
  + Update SG of backend to allow traffic from Bean SG
* Flow of Execution:
  + Launch EC-2 instance for DB initializing
  + Login to the instance and initialize RDS DB
  + Change health-check on beanstalk to /login
  + Add 443 HTTPS listener to ELB
  + Build artifact with backend information
  + Deploy artifact to Beanstalk
  + Create CDN with SSL certificate
  + Update entry in GoDaddy DNS zones

**PROCESS OF CONFIGURING THE SERVERS AND APPS**

* Created security group:
  + **vprofile-backend-sg**:
    - **All traffic** from it self for the communication between backend servers.
* Inside **RDS**:
  + Create one parameter group:
    - 
    - Create subnet group:
      * 
    - Now create **database**:
      * Standard create
      * MySQL (engiene type)
      * Give one **DB instance identifier**
      * Under **connectivity** section,
        + select your created **subnet group**.
        + Additional details: database port as **3306**
      * 
* Create **Elastic Cache**:
  + Create **Parameter Group**
    - 
    - So many options are there inside the family like Redis, Memcache, Valkey
  + Create **Subnet Group**
  + Create **Cache** (select **Memcache**)
    - Give the correct port that is mentioned inside the *application.properties* file.
    - Choose **backend-sg** as the security group.
* Create **Amazon MQ**:
  + Engiene type: RabbitMQ
  + Give one username and password.
  + Private access
  + Choose your security group
* Launch one EC-2 instance (ubuntu) (temporarily only):
  + We’ll use this to setup database inside RDS.
  + Install mysql-client, git.
  + Copy the DNS of RDS and execute the command:
    - **mysql -h <DNS> -u <username> -p**
      * Then enter password
    - Example:
      * **mysql -h vprofile-rds.c8v0iamky18r.us-east-1.rds.amazonaws.com -u admin -p**
  + I clone the git repo to get the db\_backup file and update the RDS database.
  + **mysql -h** *<DNS>* **-u** *<username>* **-p***<password; no space between -p and password>* **accounts** **< src/main/resources/db\_backup.sql** (it is the db\_backup path in my case)
  + Now DB initialization was successful. You can delete the instance now.
* Create one IAM role for beanstalk application:
  + Service: EC2
  + Policies:
    - AdministratorAccess-AWSElasticBeanstalk
    - AWSElasticBeanstalkCustomPlatformforEC2Role
    - AWSElasticBeanstalkRoleSNS
    - AWSElasticBeanstalkWebTier
* Now, **BEANSTALK**:
  + Environment: Web server environment
  + Platform: tomcat; branch: tomcat 10
  + Presets: custom configuration
  + 
    - If you don’t see that ***aws-elasticbeanstalk-service-role*** then just create one clicking the link (**create role**) at the right.
    - All the options will be auto selected. Just click next, next and create.
  + Under **Instance Setting**, *enable* public ip address. And select all the subnets under **instance subnets**.
  + Keep the **database** option as **disabled** only. Because we have our own RDS.
  + Select **GP3** as the **root volume**.
  + Don’t select any security group. Leave it blank so that beanstalk will create one by itself. Later we can edit the roles.
  + Under capacity section, use **Load Balanced** as environment type.
  + Tomcat runs on port 8080; but in beanstalk it runs on port 80.
  + Enable stickiness in the Processes section.
  + Rolling updates and deployments: Deployment policies (need to explore it separately)
    - In my case, I chose **rolling** with batch size of **50%** (for percentage you need to select that radio button for percentage)
* Now open the backend-sg and add the instance security group (created by beanstalk) in its inbound rule for **All Traffic**.
* Copy dns of RDS, AmazonMQ, Elastic cache (along with that port, username and password as well if applicable)
  + Then update the application.properties file according to that.
  + Like in place of *db01*, write the *dns of RDS* i.e. *xyz.com*, and update username and password as well.
* After updating the application.properties, execute the command **mvn install** and then the artifact will be generated (***.war*** file inside the folder *target*).
* Now just upload the **.war** file in the beanstalk application.
* You can see:
  + Initially both the instance will be healthy (inside target group)
  + Then as soon as the deployment happens, one will become unhealthy.
  + When the deployment completes, the instance will become healthy.
* Edit **Instance traffic and scaling** section inside **configuration** and add one listener for **HTTPS** traffic (port 443).
* VPC:
  + Virtual Private Cloud: a logically isolated network inside AWS where you can launch resources like EC2, RDS, Load Balancers, etc.
  + One VPC spans across an entire region.
  + By default, one subnet is there for each AZs inside the region.
* RDS:
  + It is a managed service - not just an instance.
  + Under the hood, AWS does run EC2 instance to host your database.
  + But those instances are hidden from you. You can’t ssh to that.
  + Instead AWS gives you service interface (RDS console, API, CLI etc)
    - Launch databases.
    - Configure settings (via parameter groups, option groups).
    - Scale up/down.
    - Enable Multi-AZ.
    - Manage backups, snapshots, failovers.
  + RDS is *“****not*** *just an EC2 running a DB,”* but: *“A managed database platform, where AWS controls the underlying infra (EC2, storage, networking, OS, patching), and you only control the database engine and how your app connects to it.”*
  + Simply,
    - Creating EC2 instance, installing the database things and running it: **IAAS (Infrastructure as a Service)**
    - Using RDS: **PAAS (Platform as a Service)**
* RDS objects:
  + DB Parameter Group:
    - Acts like a configuration file of RDS.
    - Examples of parameters:
      * max\_connections (limit of DB connections)
      * innodb\_buffer\_pool\_size
      * log\_min\_duration\_statement
    - Default parameter group is applied automatically.
    - You can create one custom parameter group and apply them to DB instances for file-tuning.
  + DB Subnet Group:
    - Inside which subnet of your VPC (either default or custom), your DB instance will be placed in.