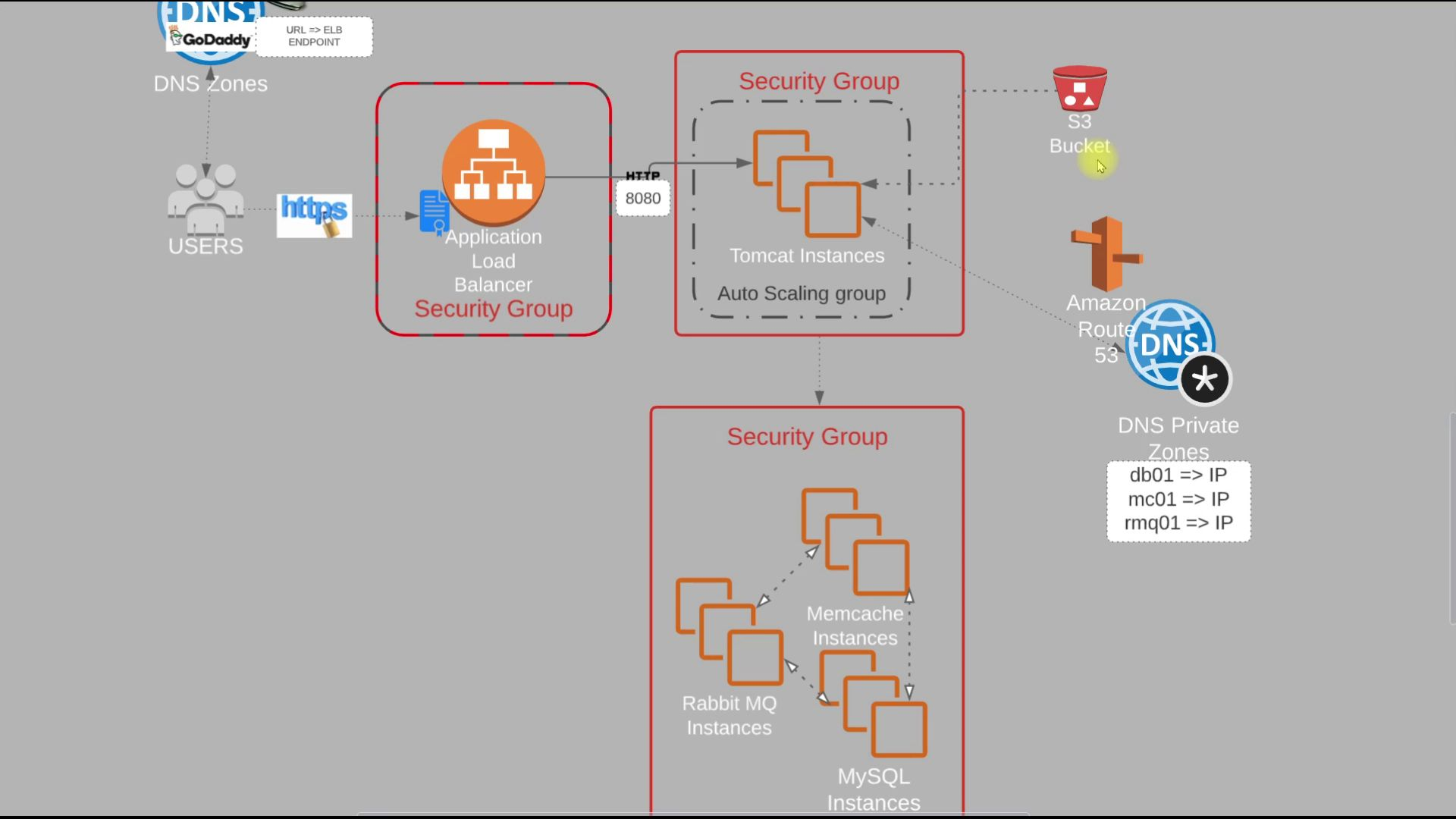
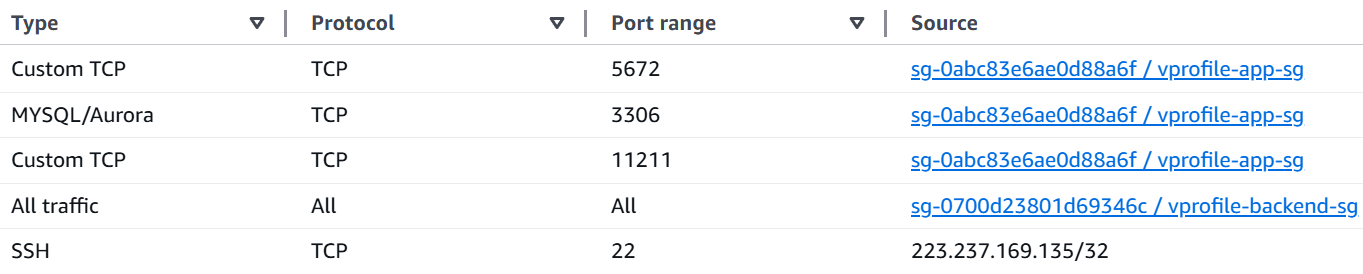
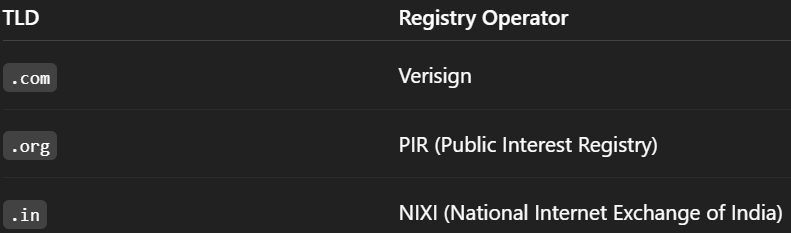
* 
  + It is the flow of our project.
  + 3 security groups are there:
    - *sg-elb* : for load balancer
      * Inbound rules will be for http & https, from all ipv4 & ipv6 addresses
      * Don’t alter outbound rule. Keep it by default
    - *sg-app* : for tomcat
      * Inbound rules will be having **sg-elb**, and ssh from *my ip*.
      * Don’t alter outbound rule. Leave it as it was by default.
    - *sg-backend* : for the backend services like memcache, rabbitmq, mysql
      * Inbound rules will be having **sg-app** with ports mentioned in the code repo for the services like MySql, RabbitMq, Memcache.
      * Also, inbound rule should contain **sg-backend** itself and allowed for **All Traffic**, so that the backend services will be able to communicate with each other.
      * And **ssh** for *my ip*.
      * 
      * Don’t alter outbound rule. Leave it as it was by default.
  + One key pair will be there to ssh the instances
  + Now, we’ll create 4 instances
    - vprofile-db01 : (sg-backend) : amazon linux
    - vprofile-app01 : (sg-app) : **ubuntu** because tomcat is there in the package of ubuntu. But in centos we need to install tomcat, create the tomcat.service and all.
    - vprofile-mc01 : (sg-backend) : amazon linux
    - vprofile-rmq01 : (sg-backend) : amazon linux
  + **Route 53:**
    - The app instance (tomcat) should connect to the backend instances through a IP address.
    - So, there should be something that will map the **hostname** to the IP addresses.
    - So, ***Route 53*** is used as the dns service in AWS.
    - Hosted Zone:
      * A container for DNS records for a domain.
      * 2 types:
        + Private: for internal use within VPC (ex: db.local)
        + Public: for publicly accessible domains (ex: example.com)
    - Creating in AWS console:
      * I created **hosted zone**, because I just want to resolve the hostnames internally.
      * Domain name: vprofile.in (anything can be given; but same entries should be there in *application.properties* file)
      * Type: private hosted zone (as I just want internal resolution)
      * Region: N. Verginia (for me)
      * VPC: default VPC
* ***Domain buying and selling*** 
  + There is a central entity **ICANN**.
  + It’s a non-profit organization that
    - Manages global DNS
    - verifies the uniqueness of a domain world-wide.
  + ICANN doesn’t sell the domains directly, rather it allows the companies like *godaddy* (called as registrars)to sell the domain from its registry.
  + Let you are assigning a IP to that domain (for now static), it’ll not be stored directly to that ICANN database.
  + Rather, it’ll be stored in a nameserver like GoDaddy’s DNS, AWS Route 53, Cloudflare, or any custom DNS provider.
* ***DNS Hierarchy consists of 2 types of companies*** 
  + ***Registrars*** 
    - These are the companies to which users contacts to buy a domain.
    - They interacts with users and Registry Operators.
    - Lets you choose the domain, provides UI to manage DNS, and Contact **TLD registry** (or **Registry Operators**) on your behalf.
  + ***Registry Operators*** 
    - These companies manages the databases of domains under a proper TLD (i.e. **.com**, **.in**, **.net** etc)
    - There is one-to-one mapping of Registry operators to a particular TLD. For example:
      * 
* ***So, what happens when you buy a domain from the Registrars?*** 
  + When you buy one domain, the **Registrar** will send info to the **Registry Operator** of that particular TLD
  + For example *Versign* (as it is the *Registry Operator* for **.com**)for **.com** TLD if the domain is in the form of *example.com*
* What if you are trying to buy a domain where that TLD that doesn’t exists?
  + You cannot register **example.alok** unless **.alok** is a valid, ICANN-approved TLD.
  + No registrars like GoDaddy's, AWS Route 53 can sell this.
  + If you want to buy this, then contact to ICANN, pay some application fees, be the Registry Operator for this TLD.
* ***What Happens When a User Enters a Domain in the Browser?*** 
  + Browser Cache Check
    - Browser checks its DNS cache for the domain’s IP.
  + OS Cache Check
    - If not found, browser asks the operating system, which checks the local system cache (hosts file or DNS cache).
  + DNS Resolver (usually ISP or custom like Google DNS)
    - If the OS doesn't have it, it contacts a recursive DNS resolver (like 8.8.8.8 or your ISP's DNS).
    - You can check using this command in cmd: **ipconfig /all**
  + Root DNS Server Lookup
    - The resolver asks a Root DNS Server: *"Where can I find* ***.com*** *TLD servers?"*
  + TLD DNS Server Lookup
    - The root server replies with the TLD nameservers for .com.
  + Authoritative Nameserver Lookup
    - The resolver asks the **.com** TLD server: *"Where is example.com hosted? What are its nameservers?"*
  + TLD Server Replies with Nameservers
    - It returns the authoritative nameservers (like AWS Route 53 nameservers).
  + Authoritative Nameserver Response
    - The resolver now contacts the Route 53 nameserver asking: *"What is the IP address for example.com?"*
  + IP Address Returned
    - The authoritative nameserver responds with the actual IP address.
  + Response Passed Back
    - The DNS resolver caches the result and returns the IP to the OS → browser.
  + Browser Makes HTTP Request
    - The browser uses that IP to connect to the server and load the website.