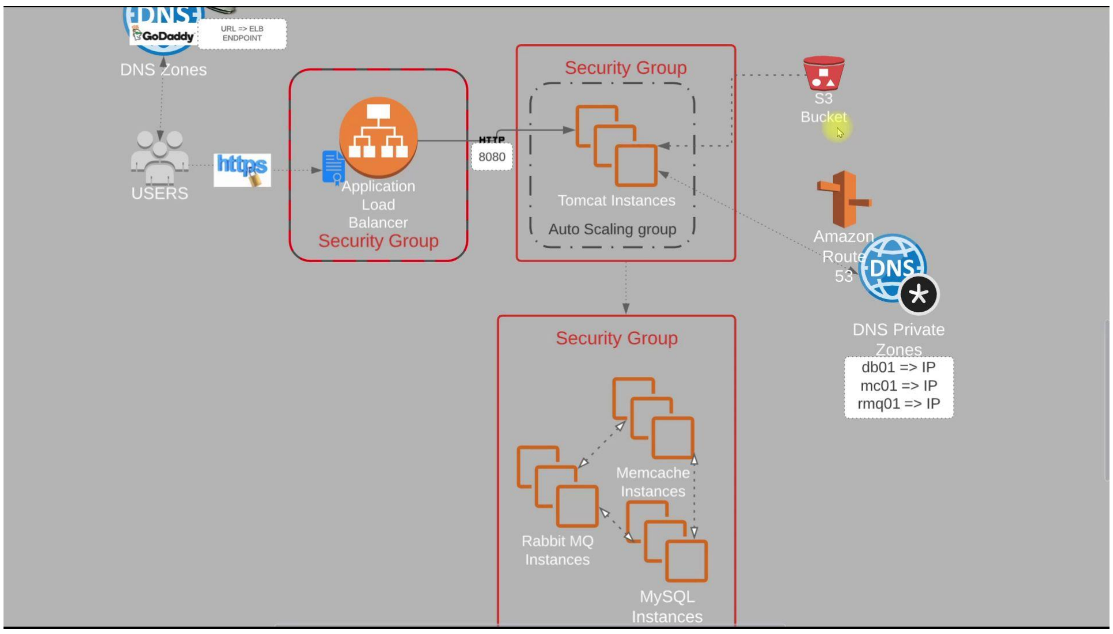
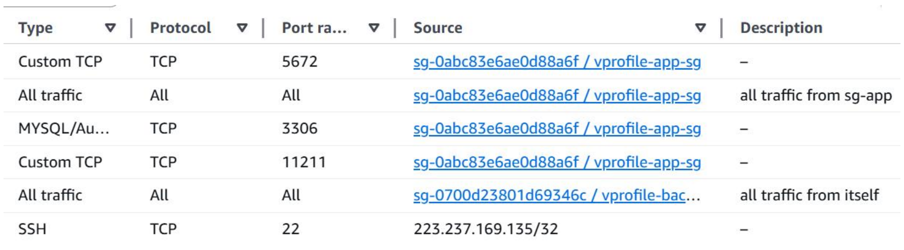
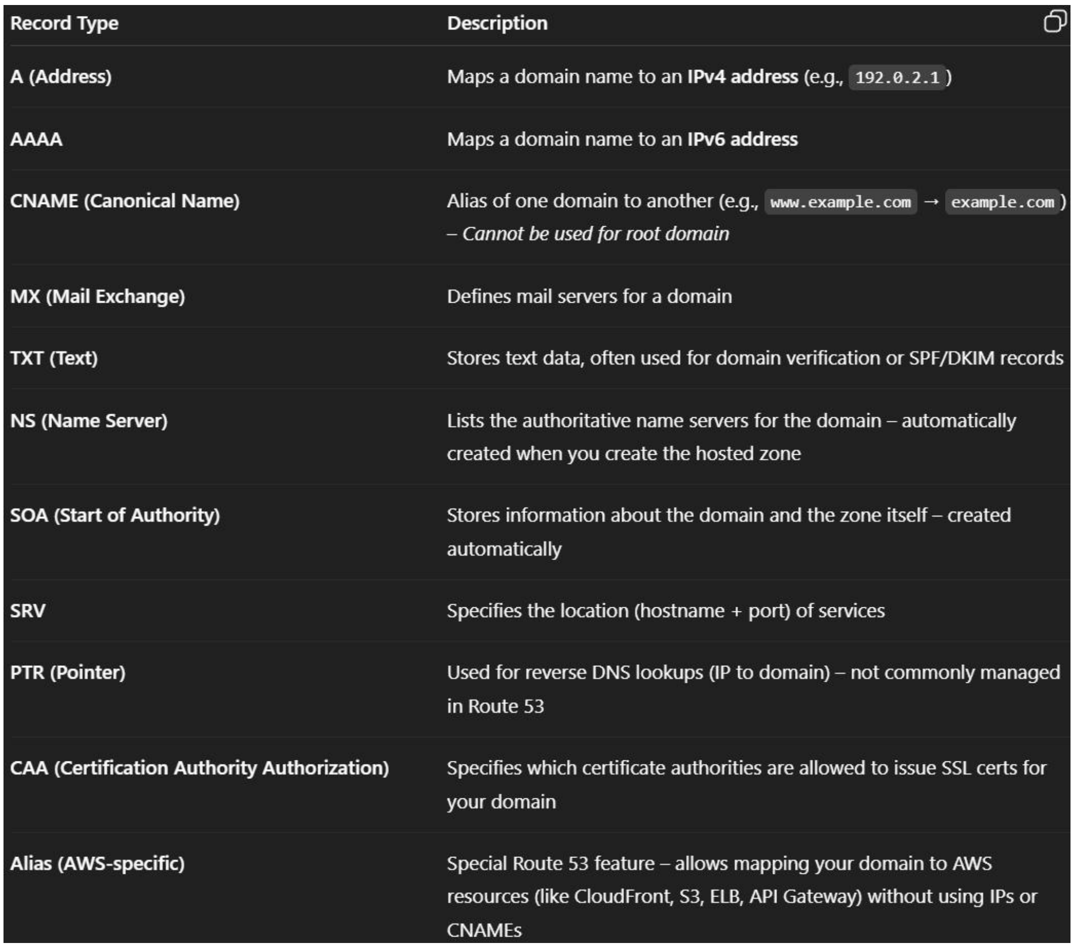
➢ 

- 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*.
    - Also add the **vprofile-app-sg** in the inbound rule for **All Traffic**, otherwise you'll not be able to ping it from the *app* instance.
    - 

Type	Protocol	Port ra...	Source	Description
Custom TCP	TCP	5672	sg-0abc83e6ae0d88a6f / vprofile-app-sg	–
All traffic	All	All	sg-0abc83e6ae0d88a6f / vprofile-app-sg	all traffic from sg-app
MYSQL/Au...	TCP	3306	sg-0abc83e6ae0d88a6f / vprofile-app-sg	–
Custom TCP	TCP	11211	sg-0abc83e6ae0d88a6f / vprofile-app-sg	–
All traffic	All	All	sg-0700d23801d69346c / vprofile-bac...	all traffic from itself
SSH	TCP	22	223.237.169.135/32	–

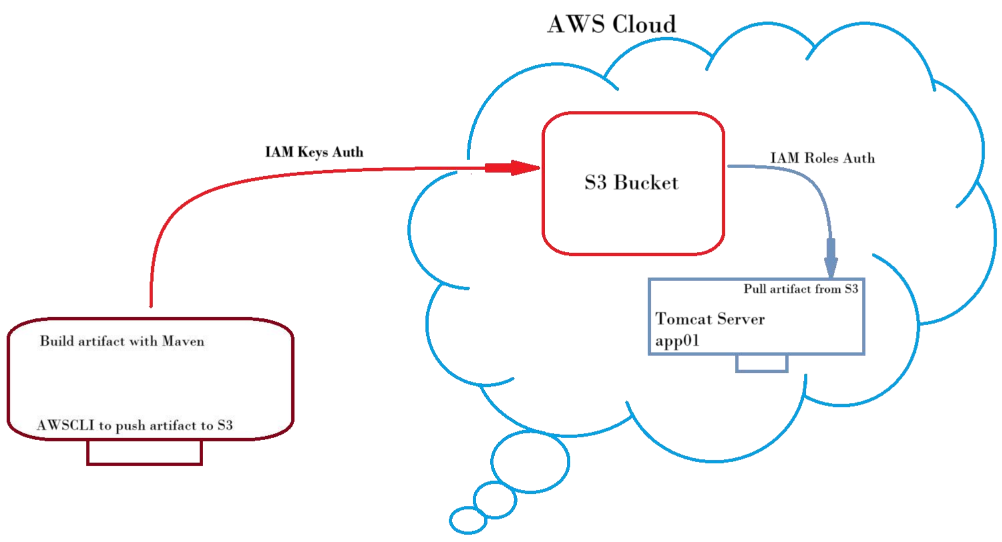
    - Don't alter outbound rule. Leave it as it was by default.

- 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:** (for DNS related things, please refer to end of the document..)
  - 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
  - After creating the *hosted zone*, go inside that and create one record.
    - For the vprofile-db01
      - Enter the record name: db01
      - Value: give the **private IP** of the instance vprofile-db01
      - Type: A (means domain to IPv4 mapping)
    - Do the same for the remaining instances (app01 is not mandatory, as we'll be using load balancer).

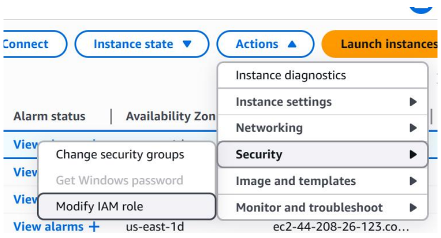


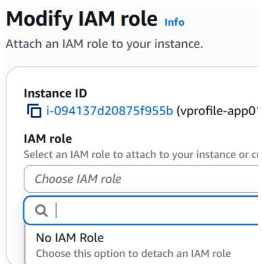
Record Type	Description
A (Address)	Maps a domain name to an **IPv4 address** (e.g., `192.0.2.1`)
AAAA	Maps a domain name to an **IPv6 address**
CNAME (Canonical Name)	Alias of one domain to another (e.g., `www.example.com` → `example.com`) – *Cannot be used for root domain*
MX (Mail Exchange)	Defines mail servers for a domain
TXT (Text)	Stores text data, often used for domain verification or SPF/DKIM records
NS (Name Server)	Lists the authoritative name servers for the domain – automatically created when you create the hosted zone
SOA (Start of Authority)	Stores information about the domain and the zone itself – created automatically
SRV	Specifies the location (hostname + port) of services
PTR (Pointer)	Used for reverse DNS lookups (IP to domain) – not commonly managed in Route 53
CAA (Certification Authority Authorization)	Specifies which certificate authorities are allowed to issue SSL certs for your domain
Alias (AWS-specific)	Special Route 53 feature – allows mapping your domain to AWS resources (like CloudFront, S3, ELB, API Gateway) without using IPs or CNAMEs

Build & Deploy Artifacts:



- We'll build the artifact from the source code and push that to AWS S3 Bucket.
- And on the tomcat instance app01, we'll fetch that artifact and deploy to the tomcat service.
- Build artifact using **Maven**, **AWSCLI** to communicate with S3 bucket.

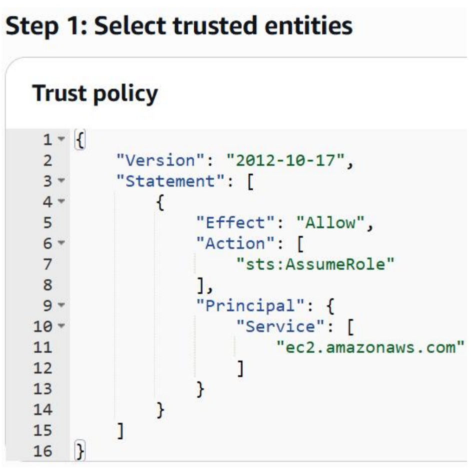
- Create one S3 bucket keeping all the values default. Just give one unique name.
- Create one IAM user, select *Attach Policies Directly* and choose *AmazonS3FullAccess*.
    - Go to the user. Under *Security Credentials* tab, create one **Access Key** for the user for **CLI**.
- Now, IAM Role. For any EC2 instance, you can modify the IAM rule as below.
    - 

    - You need to create Role to select inside the drop-down.
        - 

Modify IAM role Info
Attach an IAM role to your instance.

Instance ID
i-094137d20875f955b (vprofile-app0

IAM role
Select an IAM role to attach to your instance or c
Choose IAM role

No IAM Role
Choose this option to detach an IAM role
    - Search for **IAM >> Roles >> Create Role** :
        - Trusted Entity Service : AWS Service
        - Services or Use cases : EC2
        - Permission Policies : AmazonS3FullAccess
        - Give any name and Create the role.
    - IAM roles doesn't list any IAM user, they just define a rule to make sure which can access the instance.
        - 

Step 1: Select trusted entities

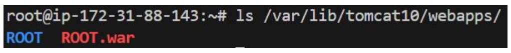
Trust policy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Principal": {
                "Service": [
                    "ec2.amazonaws.com"
                ]
            }
        }
    ]
}
```

    - It means: ***Allow EC2 instances to assume this role***
    - Assumable means: When EC2 assumes this role, AWS gives that instance temporary credentials so it can access other AWS services (like S3, DynamoDB, etc.), based on the permissions you attach to this role.

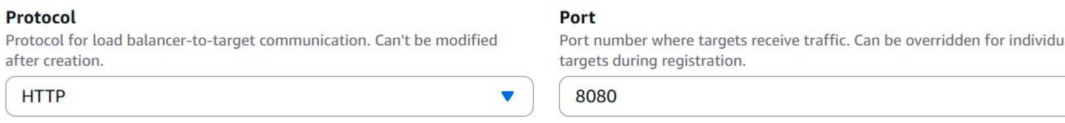
  - **NOTE --**
    - IAM Role gives the instance to access something. Like in my case, I gave **AmazonS3FullAccess**, so the instance can access S3. If we didn't have attach this role then instance won't be able to access S3.
    - Security group ensures who can access the instance and where the instance can send traffic to; whereas IAM Role gives access to the instance for some particular things that is mentioned in the IAM Role.

- Now, add this **IAM Role** inside the **app01** instance.
- So till now, *S3 bucket is created, IAM user, IAM Keys, IAM Role auth is done* ✓.
- Now, we need to build and upload it to AWS S3. First make sure that correct domains (as mentioned in the **Route 53**) are there inside the ***application.properties*** file.
- To build: **mvn install**
  - It'll create one folder *"target"*, inside that one **.war** will be there (vprofile-v2.war in my case which is the war version of the folder vprofile-v2 which is inside the *"target"* folder only)
- Copy that **.tar** file to the AWS S3ucket
  - aws s3 cp target/vprofile-v2.war s3://my-bucket-name
- Now, **ssh** to the **app01** instance.
  - systemctl daemon-reload
  - systemctl stop tomcat10
  - rm -rf /var/lib/tomcat10/webapps/ROOT
  - cp s3://my-bucket-name/vprofile-v2.war /var/lib/tomcat10/webapps/ROOT.war
  - systemctl start tomcat10   (now it'll extract that .war file)



```
root@ip-172-31-88-143:~# ls /var/lib/tomcat10/webapps/
ROOT  ROOT.war
```

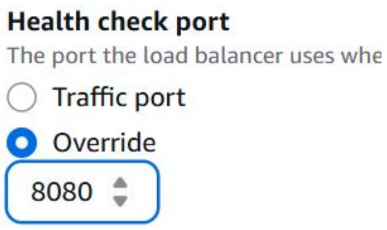
- Temporarily add 8080 port from "my ip" in sg-app and try to open the app-instance using port 8080 (8080 is alternate port of HTTP, but in our case tomcat is running on port 8080 so it is required)
- Now we'll create one target group (to attach in the Load Balancer).
  - Select protocol: HTTP >>> 8080



Protocol
Protocol for load balancer-to-target communication. Can't be modified after creation.
HTTP

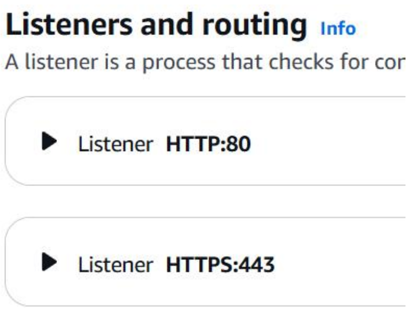
Port
Port number where targets receive traffic. Can be overridden for individu
targets during registration.
8080

  - Override the port: 8080 (default is 80)



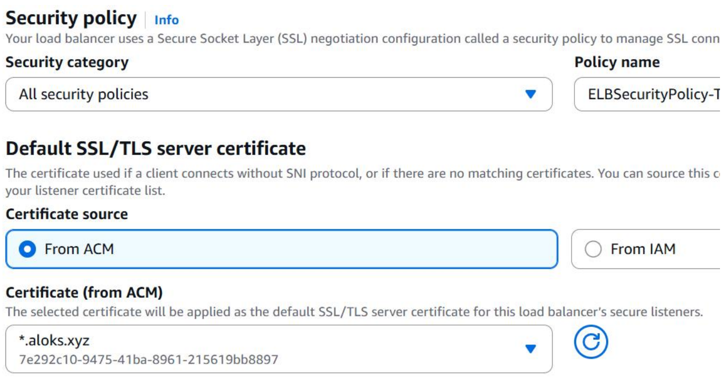
Health check port
The port the load balancer uses whe
Traffic port [✓ Override]
8080

- Now, create **ELB** (Elastic Load Balancer)
  - I added 2 listeners (for both HTTP and HTTPS)



Listeners and routing Info
A listener is a process that checks for cor
Listener **HTTP:80**
Listener **HTTPS:443**

  - For HTTPS, you need to add the ACM certificate. Foe me, I created one before.

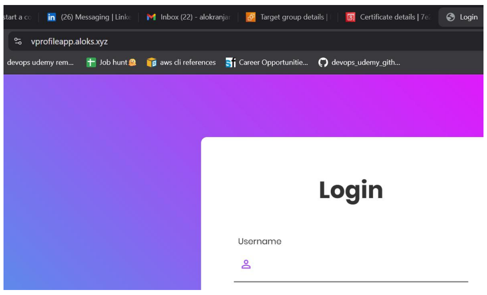


Security policy Info
Your load balancer uses a Secure Socket Layer (SSL) negotiation configuration called a security policy to manage SSL conn
Security category: All security policies
Policy name: ELBSecurityPolicy-T

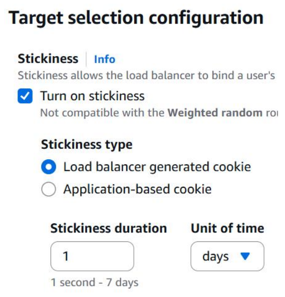
Default SSL/TLS server certificate
The certificate used if a client connects without SNI protocol, or if there are no matching certificates. You can source this c
your listener certificate list.
Certificate source
[✓ From ACM] From IAM

Certificate (from ACM)
The selected certificate will be applied as the default SSL/TLS server certificate for this load balancer's secure listeners.
*.aloks.xyz
7e292c10-9475-41ba-8961-215619bb8897

  - Copy the DNS name of the Load Balancer.
    - Go to **GoDaddy** and open your product (the purchased domain)

- Click on the **DNS** button and *add a new record.*
    - **Type**: CNAME, **Name**: name of your website(anything), **Value**: Paste the DNS of the load balancer.
    - 

    - Now it is accessible thorough the domain as well.
    - NOTE: add your **name**(that you have given above) in the beginning
    - <name>.<domain>.<extension>
    - For me, *vprofileapp.aloks.xyz*

- Now ***Autoscaling Group***:
    - Create an AMI of the app instance.
    - Create a Launch Template using that AMI, app-sg, particular key-pair.
        - Also, go to **Advanced Details** section and under the **IAM Instance Profile** drop-down, select that s3-admin that you have created (otherwise app-instance will not be able to access S3).
    - Create **Auto Scaling Group** using that *Launch Template*, use the *Created Target Group* for the load balancer, and choose the appropriate options.
    - As we have multiple app instance to which the Load Balancer can route to.
        - Turn on Stickiness in **Target Groups**.
            - **Target Groups** >> **Attributes** tab >> **Turn on stickiness**
            - 

Target selection configuration

Stickiness | Info

Stickiness allows the load balancer to bind a user's

[✓] Turn on stickiness

Not compatible with the **Weighted random** ro

Stickiness type

[● Load balancer generated cookie]

○ Application-based cookie

Stickiness duration: 1

Unit of time: days

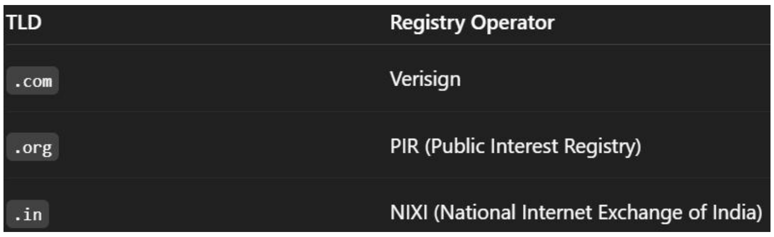
1 second - 7 days

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:



TLD	Registry Operator
`.com`	Verisign
`.org`	PIR (Public Interest Registry)
`.in`	NIXI (National Internet Exchange of India)

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.
-