Deploying an Application in a Private Subnet EC2, Attaching an Internal NLB, API Gateway through VPC link and Configuring Custom Domain Name For end user

---

The AWS services we require to perform the task are

1. VPC
    - 1.1. Subnet
    - 1.2. Route Table
    - 1.3. Internet Gateway
    - 1.4. NAT Gateway
    - 1.5. EC2
2. Network Load Balancer (NLB)
    - 2.1. Target Groups routing
    - 2.2. VPC & Subnet selection
    - 2.3. Internal / Internet Facing
3. API Gateway
    - 3.1. Type (REST/HTTP/Web-Socket)
        - 3.1.1. Resources
        - 3.1.2. Methods
        - 3.1.3. Integrations
        - 3.1.4. Stages
        - 3.1.5. Authorizers (IAM, Lambda, Cognito)
        - 3.1.6. API keys
        - 3.1.7. Logs
        - 3.1.8. Responses
    - 3.2. VPC Links
    - 3.3. Custom Domain Names
        - 3.3.1. ACM
        - 3.3.2. Route 53
            - 3.3.2.1. Hosted Zone
            - 3.3.2.2. Name Servers
            - 3.3.2.3. DNS Record

VPC- Subnet configuration:

- Create a VPC and 2 Subnets inside that VPC.
- Public Subnet will have a Public IP and can Communicate with internet via Internet Gateway
- Private Subnet won't have a Public IP. We need to assign a NAT gateway in public subnet and instance in private subnet can communicate to internet via NAT gateway.

Our application is deployed in the ec2 of private subnet.

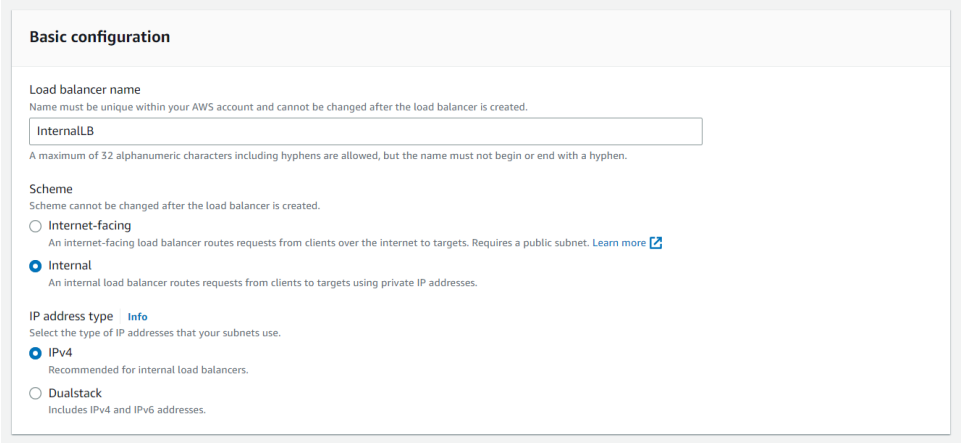
Target Group:

load balancer routes requests to the targets in a target group and performs health checks on the targets.

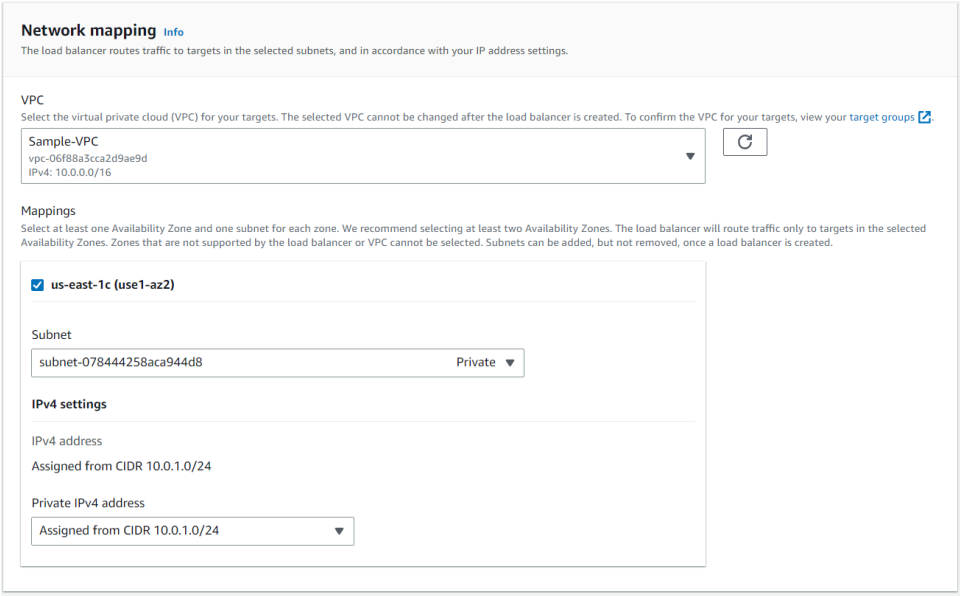
- Create a Target Group and choose EC2 as target type.
- Select your VPC and HTTP protocol. Click next.
- Register targets by selecting targets from Available instances section. Click on create target group.

NLB configuration:

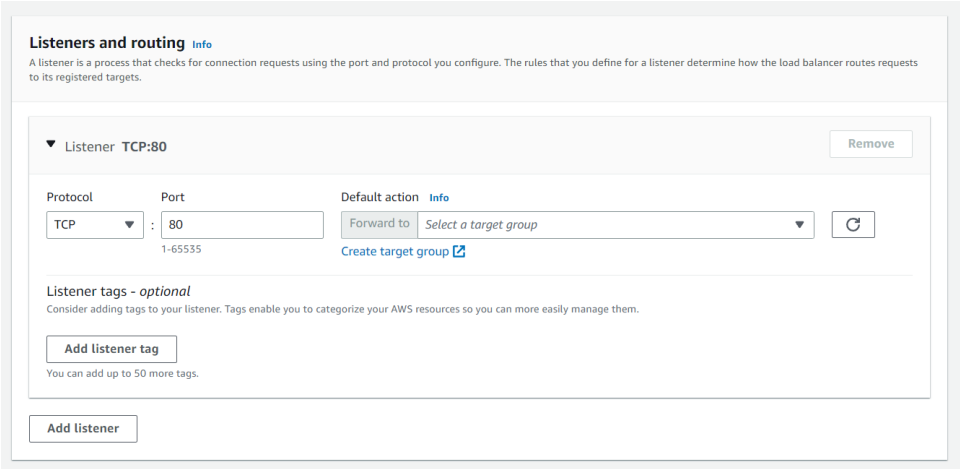
- In this project we are going to configure an Internal NLB. An internal load balancer routes requests from clients to targets using private IP addresses.



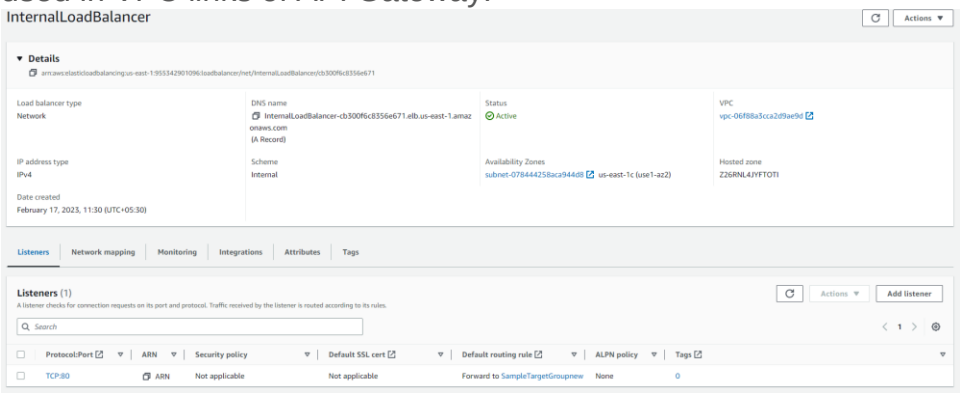
- Network Mapping. Select the VPC and private subnet.



- In listener and routing select the Target Group. The rules that you define for a listener determine how the load balancer routes requests to its registered targets.

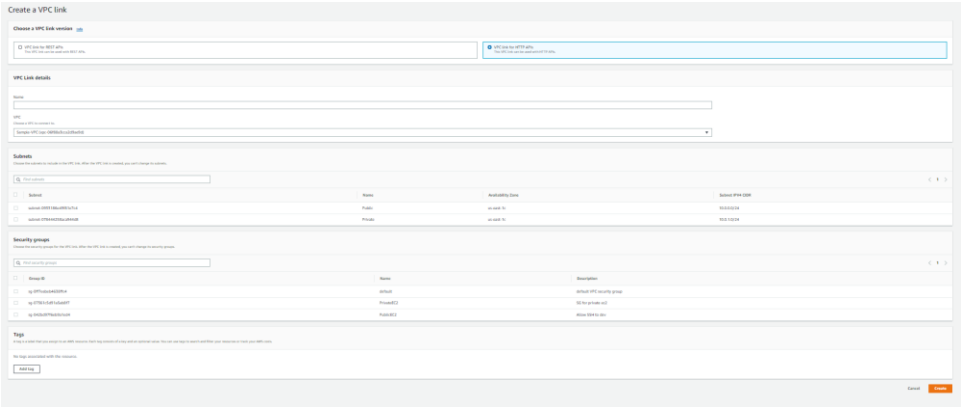


- Create an Internal NLB. Keep a note of DNS name of load balancer. It will be used in VPC links of API Gateway.



API Gateway Configuration:

- Here we are configuring both HTTP and REST Api.
- For HTTP API Create a VPC link.



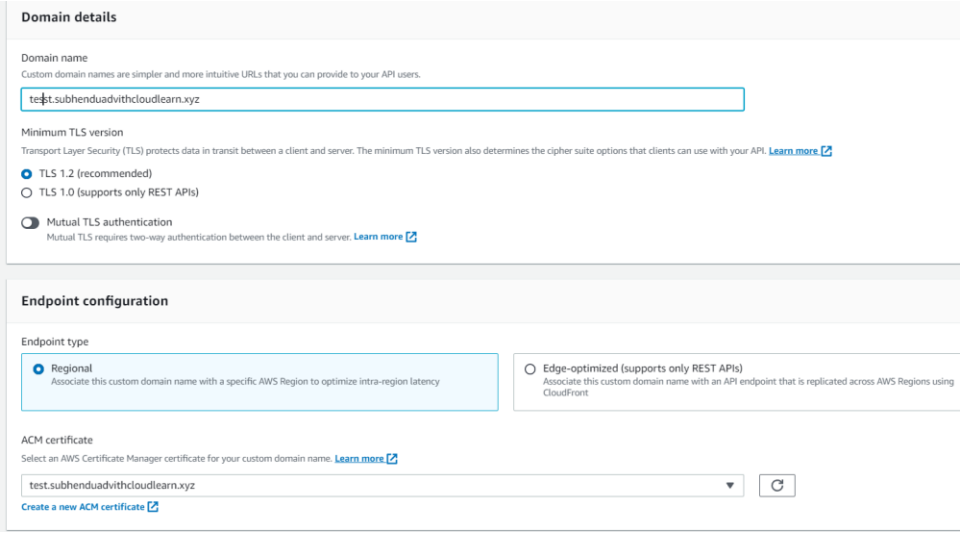
- Now build a HTTP Api and Specify the backend services that your API will communicate with. These are called integrations. For HTTP integration, API Gateway sends the request to the URL that you specify and returns the response from the URL.
- Configure routes, Define Stages and create an API gateway.
- Similarly for REST API. Create an API with defining Resources, Methods, Integrations. Deploy it to the stage.
- Now we will get Invoke URL for the API gateway. This is the default end point where end users can access our API. We can Setup Custom Domain Names and disable default end point.

Custom Domain Names:

To setup of Custom Domain name, we need to have

- Route 53 Nameserver for DNS.
- Wild card certificates should be stored in ACM. Else we can request ACM for a certificate with DNS verification.

Now Create a Custom Domain Name and provide certificates.



- Configure Path for the CDN and update Route53 DNS A records(alias) to API Gateway.

We can access our Api from Custom URL from a Browser or Postman.

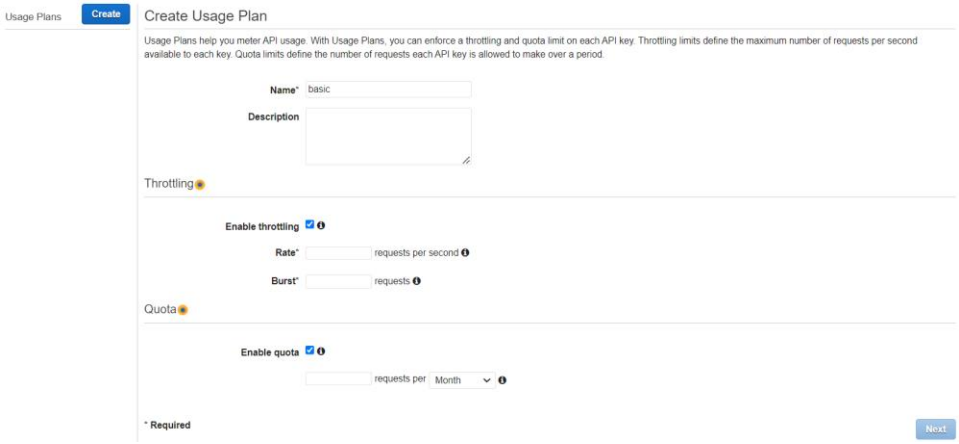
<u>Attaching Authorizers to Api Gateway:</u>

We can authorize gateway in 3 methods.

- API Keys
- Lambda Authorizer
- Cognito Authorizer

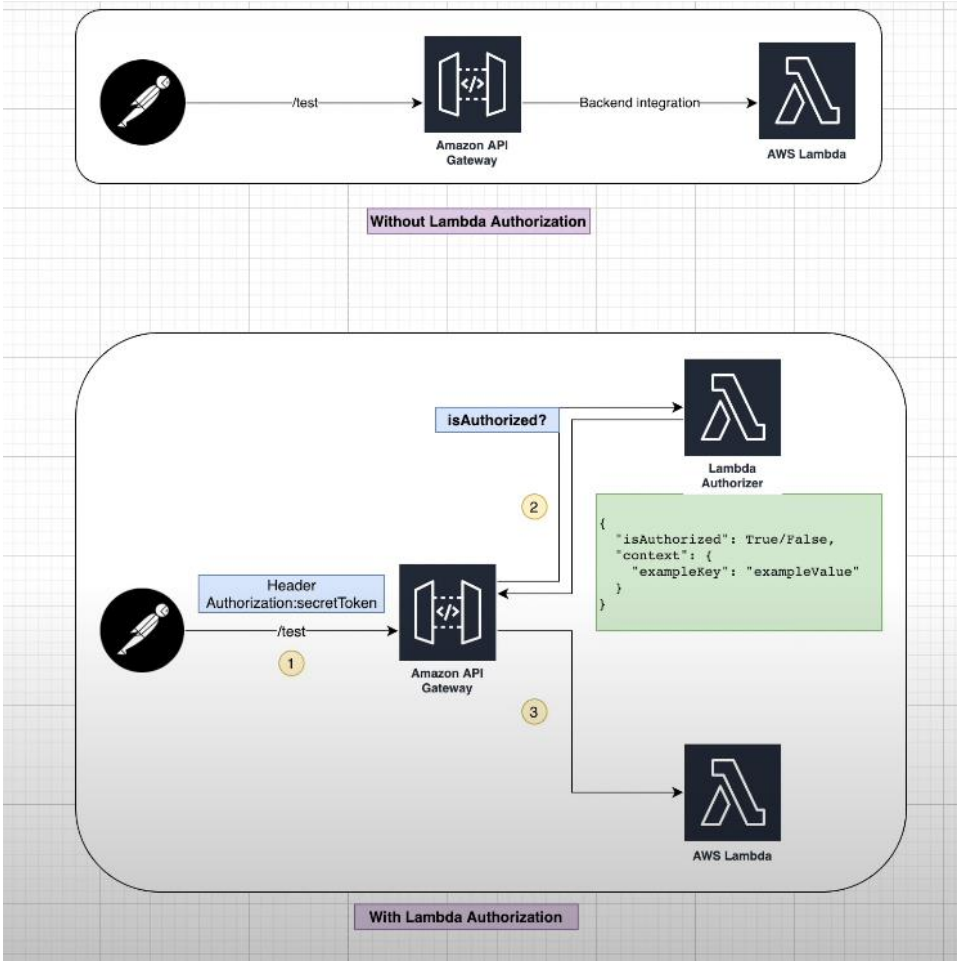
<u>API Keys:</u>

- Create Usage Plan. Usage plan helps us in metering our API, we can enable throttling and quota limit on each API key.



- Associate API stages to the Usage plan, Subscribers will only be allowed to access the API stages associated to their usage plan.
- In next step we have to subscribe an API key to the usage plan. The API key will be provided to our customers.
- We can create an API Key and add it to usage plans.
- Go to API resources, select method, in method request section set API Key required to true.
- Next step, Deploy the API, go to API keys section and copy the API key, test it with postman.

Lambda Authorizer:



- The client calls a method on an API Gateway API method, passing a bearer token or request parameters.
- API Gateway checks whether a Lambda authorizer is configured for the method. If it is, API Gateway calls the Lambda function.
- If the call succeeds, the Lambda function grants access by returning an output object containing at least an IAM policy and a principal identifier.

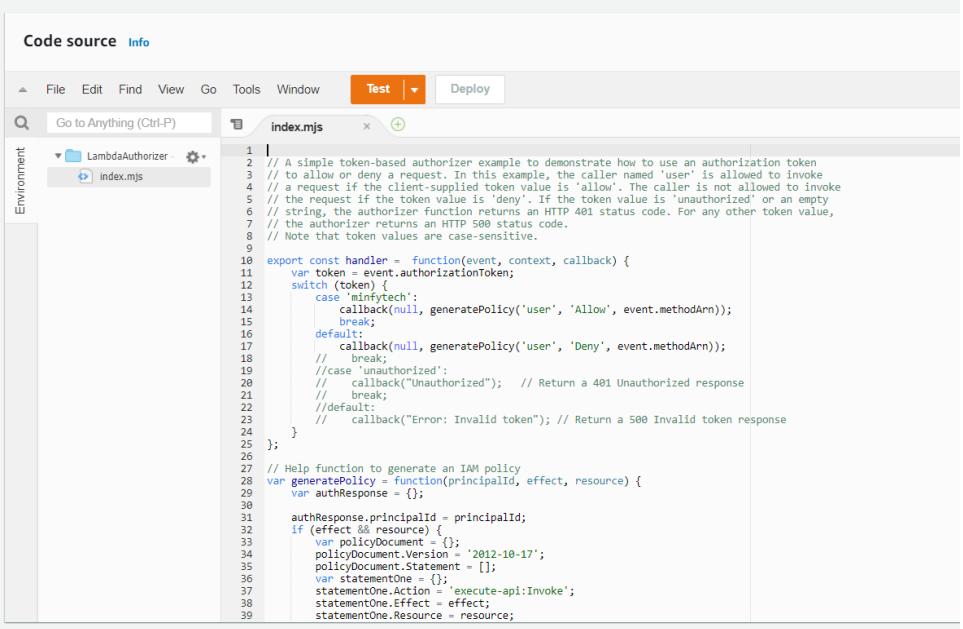
- API Gateway evaluates the policy.
    - If access is denied, API Gateway returns a suitable HTTP status code, such as `403 ACCESS_DENIED`.
    - If access is allowed, API Gateway executes the method. If caching is enabled in the authorizer settings, API Gateway also caches the policy so that the Lambda authorizer function doesn't need to be invoked again.

Configuring Lambda Function as an Authorizer:

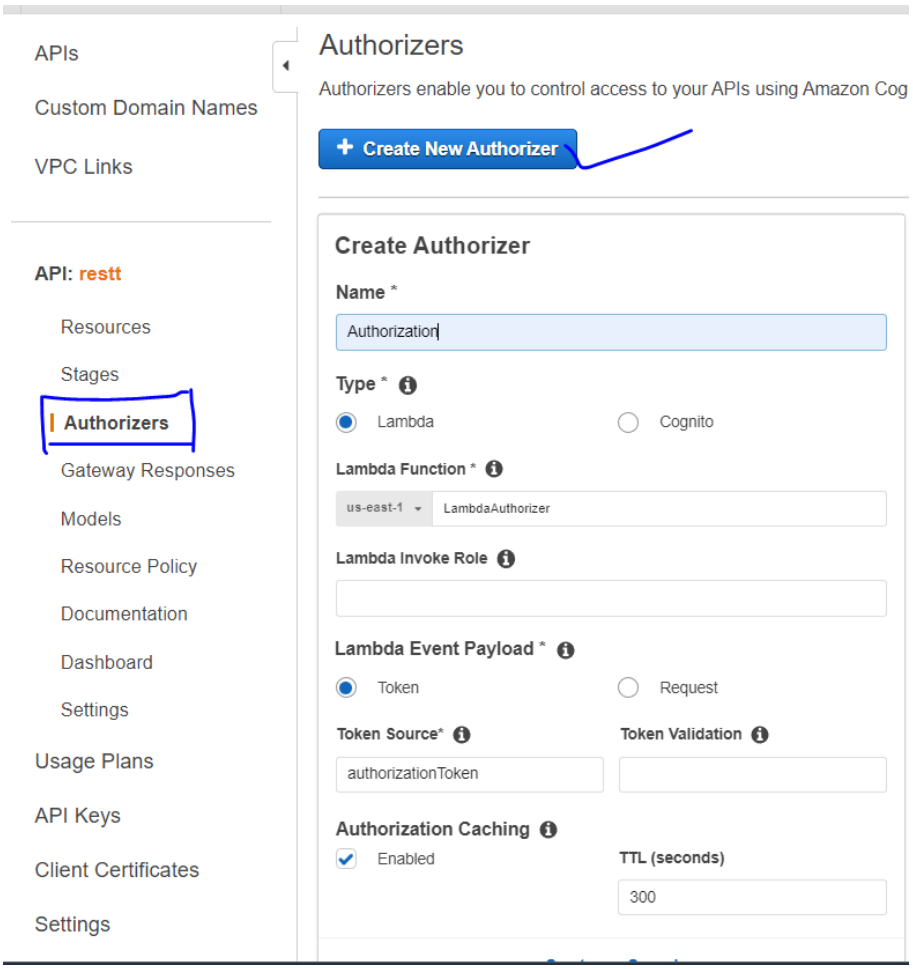
Let’s create a token-based lambda authorizer function.

- In the Lambda console, choose **Create function.**
- Choose **Author from scratch**.
- Choose **Author from scratch**.
- Choose **Create function**.

The following code Helps us to verify the request, If the token passed in authorized or not.



Now move to API Gateway and select our API, Navigate to Authorizers, Create a New Authorizer.



- Now Navigate to resources -> methods -> Method Request, Select Lambda as authorizer and save.
- Next deploy the API to effect the changes.
  We can check the same via Postman.

<u>Cognito Authorizer:</u>

- Your app users can sign in to your user pool with a user name and password, or sign in with a third-party identity provider.
- Once the user signs-in through the sign in console provided by Cognito, it returns an Authorization token (JWT), using this token a user can access our backend resources.

First let us create a Cognito user pool,

- Configure sign-in and sign-up experience.
- Configure Security requirements.
- Integrate app with a callback URL.

- Now let’s create a user from our end (user can also self-register themselves from the Hosted UI).
- Once user is created and verified sign-up is done, next we need sign in.
- Once a user signs in into the Cognito user pool, user can obtain a Token, using this token a user can access our backend resources.

- Now move to API Gateway and select our API, Navigate to Authorizers, Create a New Authorizer.
- Now Navigate to resources -> methods -> Method Request, Select Cognito as authorizer and save.
- Deploy API

We can test the same from postman.