SSL Certificate to EC2 instance

What already exists:
Domain name hosted on Route 53
CNAME record mapped to EC2 instance running the server

AWS Certificate Manager (ACM) - does provision, manage, and deploy public and private SSL/TLS certificates with AWS services and other internal connected resources.

Problem:
AWS Certificate Manager (ACM) and EC2 need an intermediary AWS service to handle SSL/TLS certificates because ACM does not directly attach certificates to EC2 instances (unless the instance is using Nitro Enclaves).
Some AWS services that can be used in between EC2 and ACM:

1. **Elastic Load Balancing (ELB)**
    - Use an **Application Load Balancer (ALB)** or **Network Load Balancer (NLB)**.
    - ACM integrates with ELB to manage SSL/TLS certificates for secure HTTPS connections.
    - The load balancer then routes traffic securely to EC2 instances.
2. **Amazon CloudFront**
    - If your EC2 instances serve static/dynamic content, you can use **CloudFront** as a CDN.
    - ACM issues SSL/TLS certificates to CloudFront distributions to secure connections.
    - CloudFront fetches data from EC2 (or S3) and serves it securely to users.
3. **Amazon API Gateway**
    - If your EC2 instances expose APIs, you can front them with **API Gateway**.
    - ACM provides SSL/TLS certificates for custom domains used in API Gateway.
    - API Gateway can route API requests securely to EC2-based backends.
4. **AWS Elastic Beanstalk**
    - If your application is hosted on **Elastic Beanstalk**, it automatically provisions a load balancer.
    - ACM integrates with Beanstalk to secure applications with HTTPS.
    - Internally, it uses ELB to route traffic to EC2 instances.
5. **AWS App Runner**
    - If you're deploying a containerized app and don't want to manage EC2 directly, **App Runner** is an alternative.
    - It automatically provisions ACM certificates for secure custom domains.

Common Setup:

1. **ACM (Manages SSL/TLS certs) → ELB (Handles HTTPS) → EC2 (Backend server)**
2. **ACM → CloudFront → EC2**
3. **ACM → API Gateway → EC2**

Why Use an Application Load Balancer (ALB)?

1. **Traffic Distribution-** ALB evenly distributes incoming traffic across multiple EC2 instances, preventing any single server from getting overloaded.
2. **High Availability & Failover**-If one EC2 instance fails, ALB automatically redirects traffic to a healthy instance, ensuring your application stays online.

3. **Secure HTTPS Connections**-ALB integrates with AWS Certificate Manager (ACM) to enable SSL/TLS encryption, securing data transmission over HTTPS.
4. **Scalability & Performance-**ALB supports auto-scaling, allowing your application to handle increasing traffic efficiently without manual intervention.

Features 1,2,4 are useful if we plan to scale up the ec2 in the future

Recommended Approach: Use ALB + ACM + Route 53

Since the domain is already in **Route 53**:

Step 1: Create an Application Load Balancer (ALB) in front of the EC2 instance.
Step 2: Attach ACM certificate to the ALB.
Step 3: In Route 53, update A-record to point to the ALB's DNS name instead of the EC2 IP.
Step 4: If a new EC2 instance is needed, simply add it to the ALB Target Group, and no changes

Pricing:

Public SSL/TLS certificates are free through AWS Certificate Manager (ACM). You only pay for the AWS resources you use to run your application. Example: ALB, EC2

https://aws.amazon.com/elasticloadbalancing/pricing/

ALB Pricing example from AWS: The application receives one new connection per second, with each lasting two minutes, and processes five requests per second, totaling 1.08 GB of data per hour. The ALB uses 60 rules to route requests. The monthly cost is calculated as $22.42, which includes both LCU-based charges and a fixed hourly charge.

For a website used by a maximum of 500 customers, the cost would be significantly lower than the example provided, likely around $6–$10 per month, depending on the actual traffic and usage patterns. The pricing depends on factors like active connections, processed bytes, and rule evaluations.

Services to stop once the Budget is reached:

To identify all the resources incurring costs, use the Cost Explorer.

Some services can be **stopped** to save money and later restarted (eg Amazon EC2 instances, Amazon RDS databases), while some resources can only be **deleted** to stop the charges (eg NAT Gateway, storage in Amazon S3).

EC2 Stopping Methods

1. **Using Lambda and CloudWatch (Custom Automation)**:
    - You can create an **AWS Lambda function** triggered by an **Amazon CloudWatch Alarm** that monitors your EC2 instance usage or costs.
    - Once the threshold is exceeded, the Lambda function can automatically stop the EC2 instance using the stop-instances API.
    - This method gives you more flexibility and control over which instances to stop and under what conditions.
2. **Using AWS Budget Actions**:
    - With **AWS Budget Actions**, you can automatically stop EC2 instances when your set budget is exceeded.
    - You configure an **AWS Budget** with a specified cost or usage limit, and set up an **action** to stop EC2 instances once the budget threshold is breached.
    - This method is simpler and does not require coding, but it is limited to stopping EC2 instances only and doesn't offer granular control over other resources.

Your first two action-enabled budgets are free (regardless of the number of actions you configure per budget) per month. Afterwards each subsequent action-enabled budget will incur a $0.10 daily cost.