To enable HTTPS on your application served by NGINX with an SSL certificate through a Network Load Balancer (NLB) on AWS, you can configure the NLB to handle the SSL termination, meaning the NLB will decrypt HTTPS traffic and forward it to your backend servers over HTTP. Here’s how you can set it up:

### 1. Configure the Network Load Balancer

1. SSL Certificate:

- Ensure you have an SSL certificate in AWS Certificate Manager (ACM) for the domain you want to secure.

- If you don’t have a certificate yet, you can request one through ACM in the same region as your load balancer.

2. Listener Setup:

- Go to Load Balancers in the EC2 Management Console.

- Select your Network Load Balancer, then navigate to the Listeners tab.

- Add a new listener for port 443 (HTTPS).

3. Listener Configuration:

- Select HTTPS as the protocol for the new listener.

- Choose the SSL certificate from ACM that you wish to use for HTTPS traffic.

- In the listener settings, forward traffic to a target group that points to your backend instances.

- Ensure that the target group uses HTTP (port 80) as its protocol since SSL termination will happen at the load balancer.

4. Health Checks:

- Configure health checks for your target group. Since the backend is listening on HTTP, ensure the health checks use HTTP and point to an appropriate health check path (e.g., `/health` if available on your server).

### 2. Configure Security Groups

For the instances behind the load balancer, ensure that:

- Port 80 is open to allow traffic from the load balancer.

- The NLB can access your instances on HTTP (port 80).

### 3. Update NGINX Configuration for SSL Handling by Load Balancer

Since the SSL termination happens at the NLB, your NGINX server will only need to listen on port 80. You can optionally configure NGINX to recognize when traffic is forwarded over HTTPS from the load balancer by using the `$scheme` variable and `X-Forwarded-Proto` header in your configuration.

Here's an example NGINX configuration for this setup:

```nginx

server {

listen 80;

server\_name yourdomain.com;

# Redirect HTTP to HTTPS if desired (optional)

location / {

if ($http\_x\_forwarded\_proto = "http") {

return 301 https://$host$request\_uri;

}

proxy\_pass http://localhost:3000;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $scheme;

# WebSocket support

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection "upgrade";

}

location /api {

proxy\_pass http://localhost:8000;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $scheme;

}

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

root /usr/share/nginx/html;

}

}

In this configuration:

- Redirect: The `if` block checks if the incoming request used HTTP (via `X-Forwarded-Proto`) and redirects to HTTPS. This ensures users are consistently on a secure connection.

- Proxy Headers: Setting `X-Forwarded-Proto` and `X-Real-IP` helps NGINX to log the original client's IP and the scheme, which is useful for accurate request logging and session management.

### 4. Test and Verify

After updating your NGINX configuration, test it with:

sudo nginx -t

If the test is successful, reload NGINX:

Then, try accessing your domain via HTTPS. The NLB should handle the SSL termination, and NGINX will receive requests over HTTP from the NLB.

### Optional: Additional Security and Best Practices

- HTTP to HTTPS Redirect: You can configure NGINX to redirect all HTTP traffic to HTTPS (as shown above) for consistency.

- Update DNS: Ensure your DNS is updated to point to the NLB’s HTTPS endpoint.

- Firewall and Security Groups: Confirm that only the load balancer can access your instances over port 80.

This setup will enable secure HTTPS access to your application through the Network Load Balancer with SSL termination handled by AWS.

TCP (Transmission Control Protocol) and TLS (Transport Layer Security) are distinct protocols that operate at different layers in the networking model and serve different purposes:

### 1. Layer of Operation

- TCP: Operates at the Transport Layer of the OSI model (Layer 4). It establishes a reliable connection between two endpoints (e.g., client and server) for data transfer.

- TLS: Operates at the Application Layer (Layer 7, above TCP) but is sometimes considered to sit between the Transport and Application Layers (Layer 5-7). It provides encryption and secure communication over a TCP connection.

### 2. Purpose

- TCP: Primarily used for establishing a reliable connection, ensuring ordered data delivery, retransmitting lost packets, and managing data flow. It handles packet loss and guarantees in-sequence data delivery.

- TLS: Provides security features like encryption, authentication, and integrity for data transmitted over TCP. It prevents eavesdropping, tampering, and message forgery, which is especially important for sensitive data (e.g., login credentials, financial transactions).

### 3. Functionality

- TCP: Manages data packets, ensures they are delivered correctly, and reorders them if needed. It handles connection setup, data transfer, and connection termination.

- TLS: Adds security layers to an existing TCP connection. It encrypts data to maintain confidentiality, uses certificates to establish authenticity, and checks message integrity to detect any tampering. TLS includes a handshake phase where encryption keys are negotiated.

### 4. Use Cases

- TCP: Used in most Internet communications, including HTTP, FTP, SMTP, and more. It provides the basic structure for data delivery without encryption.

- TLS: Used alongside TCP to secure communication channels, particularly in HTTPS (secure HTTP), FTPS (secure FTP), and SMTPS (secure SMTP). TLS encrypts data transmitted over TCP for applications where security is a priority.

### 5. Security Features

- TCP: Has no inherent security features; it is focused on reliable data transport, not on data confidentiality or integrity.

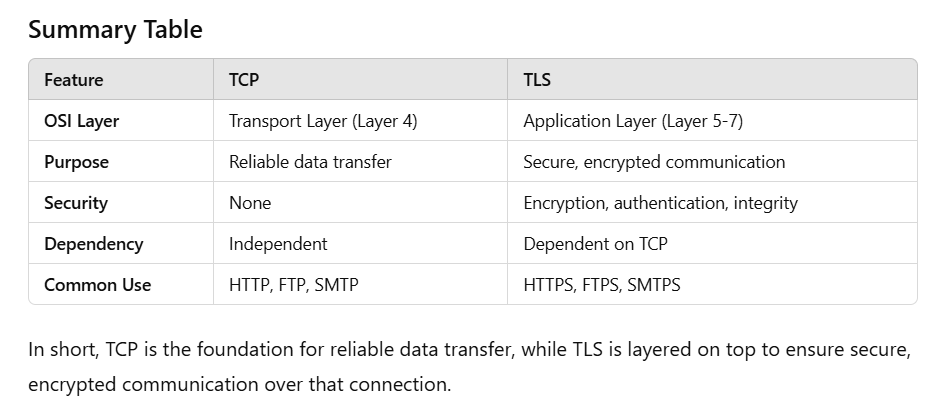
- TLS: Provides robust security features, including encryption (to keep data confidential), authentication (to verify identity using certificates), and data integrity (to prevent tampering).

### 6. Protocol Interaction

- TCP: Functions independently but serves as the foundation for higher-layer protocols that may add security.

- TLS: Requires an underlying reliable protocol like TCP to function, as it relies on TCP's guarantees to establish and maintain secure connections.

### Summary Table



In short, TCP is the foundation for reliable data transfer, while TLS is layered on top to ensure secure, encrypted communication over that connection.

The OSI (Open Systems Interconnection) model and the TCP/IP model are both conceptual frameworks for understanding and standardizing network communication. The OSI model is broader, with seven distinct layers, while the TCP/IP model is more streamlined, with four layers. Here's how they compare in terms of flow and functionality.

### OSI Model Flow

The OSI model has seven layers, each with specific functions. Here’s a breakdown from the top layer (closest to the user) to the bottom layer (closest to physical transmission):

1. Application Layer (Layer 7)

- Provides network services directly to applications, like web browsing and email.

- Example protocols: HTTP, FTP, SMTP.

2. Presentation Layer (Layer 6)

- Ensures data is in a usable format, handling encryption, compression, and translation between different data formats.

- Example functions: Data encryption (SSL/TLS).

3. Session Layer (Layer 5)

- Manages and maintains communication sessions, establishing, managing, and terminating connections.

- Example functions: Session establishment, maintenance, and teardown.

4. Transport Layer (Layer 4)

- Provides reliable or unreliable data transfer across the network, using flow control, error correction, and retransmission.

- Example protocols: TCP (reliable) and UDP (unreliable).

5. Network Layer (Layer 3)

- Handles logical addressing, routing, and forwarding to allow data to travel between networks.

- Example protocols: IP, ICMP.

6. Data Link Layer (Layer 2)

- Responsible for data transfer between adjacent network nodes, using MAC addressing for node-to-node data transfer.

- Example protocols: Ethernet, ARP.

7. Physical Layer (Layer 1)

- Deals with the physical connection between devices, including hardware, cables, and electrical signals.

- Example elements: Cabling, voltage levels, radio frequencies.

Data Flow in the OSI Model:

Data is generated at the Application layer, moves down through each layer (where it is encapsulated with headers), and reaches the Physical layer for transmission. When data is received, it flows back up the layers, with each layer handling and stripping away the corresponding headers.

### TCP/IP Model Flow

The TCP/IP model has four layers, simplifying some of the functions of the OSI model. Here’s the flow from top to bottom:

1. Application Layer

- Combines OSI’s Application, Presentation, and Session layers.

- Provides end-user services like file transfer, email, and web browsing.

- Example protocols: HTTP, FTP, SMTP, DNS.

2. Transport Layer

- Handles end-to-end data transfer, reliability, and error correction (similar to OSI’s Transport layer).

- Example protocols: TCP (reliable) and UDP (unreliable).

3. Internet Layer

- Manages logical addressing, routing, and packet forwarding between networks (similar to OSI’s Network layer).

- Example protocols: IP, ICMP, ARP.

4. Network Interface (Link) Layer

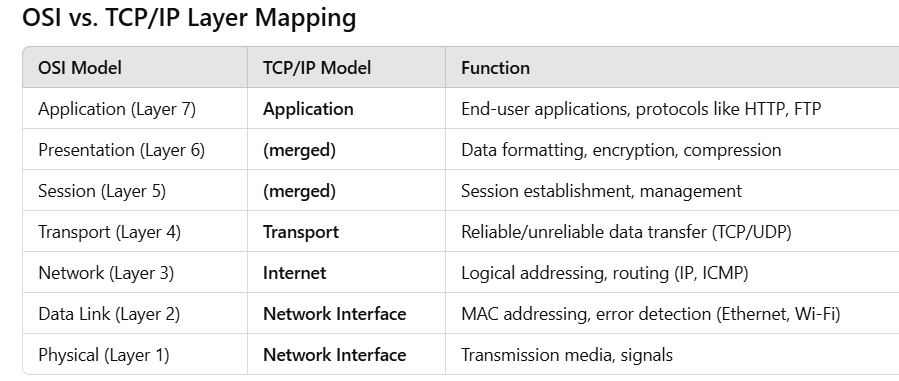
- Combines OSI’s Data Link and Physical layers, managing direct data transfer between network nodes.

- Example protocols and standards: Ethernet, Wi-Fi, ARP.

Data Flow in the TCP/IP Model:

Data starts at the Application layer, flows down through each layer (being encapsulated), and is transmitted via the Network Interface layer. When received, the data flows back up through each layer, where headers are removed, and the data is interpreted.

### OSI vs. TCP/IP Layer Mapping



### Key Differences in Flow

- OSI Model: Provides a more granular breakdown with seven layers, allowing for detailed distinctions like presentation (data format) and session (connection management).

- TCP/IP Model: Combines layers for simplicity, emphasizing practical use cases for Internet communication and focusing on key protocols like TCP and IP.

In both models, data encapsulation and decapsulation occur at each layer, adding or removing headers as data travels through the layers. The main difference lies in the layers’ abstraction level and specific functionalities each model emphasizes.