HTTP VS HTTPS

1. Core Concepts: HTTP vs HTTPS

HTTP (HyperText Transfer Protocol)

- **Unencrypted**, plain-text protocol for transferring data over the web.
- Operates over **TCP port 80**.
- Anyone in the network path (e.g., ISPs, proxies, attackers) can **read or modify** the data.
- No data integrity or confidentiality guarantees.

HTTPS (HyperText Transfer Protocol Secure)

- HTTP **secured with TLS (Transport Layer Security)**.
- Operates over **TCP port 443**.
- **Encrypts the entire communication channel**, preventing eavesdropping and tampering.
- Provides:
    - **Authentication** (via certificates)
    - **Integrity** (data is untampered)
    - **Confidentiality** (data is encrypted)

---

2. TLS: The Foundation of HTTPS

What is TLS?

TLS (Transport Layer Security) is a cryptographic protocol that provides **secure communication** over a network. It replaces the older SSL.

TLS Key Responsibilities

- **Encryption**: Protects the confidentiality of data in transit.
- **Authentication**: Verifies the identity of the server (and optionally the client).
- **Integrity**: Ensures data isn't altered in transit.

TLS Handshake (Simplified Flow)

1. **Client Hello**:
    - Client sends supported TLS versions, cipher suites, and a random value.
2. **Server Hello**:
    - Server picks a TLS version and cipher suite, sends its certificate, and its random value.
3. **Certificate Exchange**:

- Server sends its **TLS certificate** (issued by a trusted Certificate Authority).

4. **Key Exchange**:
    - Client and server agree on a **shared session key** via asymmetric encryption (RSA or ECDHE).
5. **Session Encryption Starts**:
    - All further communication is encrypted using the symmetric key derived above.

---

3. HTTPS in Kubernetes Ingress: Full Explanation

Ingress is a Kubernetes API object that manages **external access to services**, usually HTTP(S), and uses **Ingress Controllers** (like NGINX) to handle actual traffic.

Scenario:

You want to expose your app securely via HTTPS on domain www.example.com.

A. What Happens Without TLS (HTTP Only)

1. Your app runs in a pod and is exposed via a **Service** (often ClusterIP).
2. You configure an **Ingress** resource with:

```
path: /
pathType: Prefix
backend:
  service:
    name: myapp-service
    port:
      number: 80
```

3. When user accesses http://www.example.com, the traffic flows:
   Client → Ingress Controller (NGINX) → Service → Pod
    - **All in plain HTTP.**
    - Data is exposed to MITM (Man-In-The-Middle) attacks.
    - No identity verification or encryption.
    - Bad for login, payments, APIs, etc.

B. What Happens With HTTPS (TLS Termination via Ingress)

Step-by-Step HTTPS + TLS in Ingress

1. You Issue a Certificate

- Use **cert-manager** to automatically issue TLS certificates from Let's Encrypt or any CA.
- Define a **ClusterIssuer** or **Issuer** like:

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: admin@example.com
    privateKeySecretRef:
      name: letsencrypt-prod
    solvers:
      - http01:
          ingress:
            class: nginx
```

2. You Create an Ingress with TLS Section

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: myapp-ingress
  annotations:
    cert-manager.io/cluster-issuer: letsencrypt-prod
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
    nginx.ingress.kubernetes.io/force-ssl-redirect: "true"
spec:
  ingressClassName: nginx
  rules:
    - host: www.example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: myapp-service
                port:
                  number: 80
  tls:
    - hosts:
        - www.example.com
      secretName: myapp-tls
```

What Happens Internally

1. **cert-manager notices the annotation** and creates a TLS challenge (e.g., HTTP-01).
2. **Let's Encrypt sends a challenge request** to http://www.example.com/.well-known/acme-challenge/<token>.
3. The Ingress routes that to a temporary pod that solves the challenge.

4. If the challenge passes, **Let's Encrypt issues a TLS certificate**, stored in the secret myapp-tls.

Now Ingress knows how to:

- **Serve HTTPS** (using the myapp-tls certificate)
- **Terminate TLS**: It handles decryption and forwards **plain HTTP to backend**.

---

4. TLS Termination in Kubernetes Ingress

What is TLS Termination?

It means that **Ingress Controller (e.g., NGINX)** is responsible for handling TLS encryption/decryption.

Sequence with TLS Termination:

1. **Client (browser) initiates HTTPS** connection to www.example.com.
2. **Ingress Controller**:
    - Receives request on port 443.
    - Uses the TLS cert from myapp-tls secret.
    - Handles the **TLS handshake**.
    - **Decrypts** the HTTPS to plain HTTP.
3. **Ingress Controller → App Service** via internal traffic (usually HTTP).

Important:

Even though the app pod speaks HTTP, it's safe because:

- TLS is **terminated at the edge** (Ingress).
- Internal traffic stays within the **Kubernetes cluster**, protected by network policies, firewalls, or private VPC.

---

5. Differences Summarized

Feature	**HTTP**	**HTTPS**
Port	80	443
Encryption	❌ None	✅ Encrypted via TLS
Security	❌ Vulnerable	✅ Secure
Identity Verification	❌ No	✅ TLS Certificate
Kubernetes Ingress	Simple Ingress Rule	Requires TLS config, cert-manager, secret

Feature	HTTP	HTTPS
Ingress Annotation	Not needed	cert-manager.io/cluster-issuer, ssl-redirect, etc.
TLS Termination	❌ No	✅ Yes (by Ingress Controller)

---

6. TLS Traffic Flow in Kubernetes

With TLS + Ingress

```
[Client]
   |
   | HTTPS (TLS handshake)
   |
[Ingress Controller (NGINX)]
   |
   | HTTP (plain, inside cluster)
   |
[Service]
   |
   | HTTP
   |
[Pod]
```

Without TLS

```
[Client]
   |
   | HTTP (plain, unsecured)
   |
[Ingress Controller]
   |
   | HTTP
   |
[Pod]
```

---

7. When You Might Want End-to-End TLS

If internal traffic also needs to be encrypted (e.g., for zero trust environments or compliance), then:

- Ingress **does not terminate TLS**.
- It **passes encrypted HTTPS** to backend services.
- Backend pods must also have TLS support and certificates.

This is called **TLS passthrough** (requires a different Ingress config or L4 LoadBalancer).

Conclusion

- **HTTPS in Kubernetes via Ingress is achieved using TLS termination.**

- **Ingress Controller** acts like a reverse proxy that offloads encryption tasks.
- **cert-manager + Let's Encrypt + Ingress annotations** automate TLS issuance and renewal.
- TLS is **essential for securing production traffic**, especially when dealing with user data, logins, or payments.
- HTTP is fine for **internal dev/test use** but must never be used over public networks for sensitive workloads.