DevOps Shack

SSL/TLS Certificate Setup and Management in DevOps

A Comprehensive Guide to GitOps on Kubernetes

By DevOps Shack

DevOps Shack

SSL/TLS Certificate Setup and Management in DevOps: A Practical Guide

Table of Contents

1. Introduction

1.1 Overview of SSL/TLS

SSL (Secure Sockets Layer) and its successor TLS (Transport Layer Security) are cryptographic protocols designed to provide secure communication over the internet. They encrypt data transmitted between clients (browsers, applications) and servers, ensuring confidentiality, integrity, and authentication. TLS 1.2 and TLS 1.3 are the widely used secure versions today.

1.2 Importance of SSL/TLS in DevOps

In modern DevOps workflows, SSL/TLS plays a critical role in securing web applications, APIs, and cloud environments. The key benefits include:

- **Data Encryption** – Prevents unauthorized access and man-in-the-middle (MITM) attacks.
- **Authentication** – Ensures that the client is communicating with the intended server.
- **Data Integrity** – Protects against tampering during transmission.
- **Regulatory Compliance** – Helps meet security standards like GDPR, PCI- DSS, and HIPAA.

SSL/TLS certificates are essential in DevOps pipelines to maintain secure deployments, protect customer data, and build trust.

1.3 Choosing the Right SSL/TLS Management Approach

Managing SSL/TLS certificates in DevOps environments can be automated and streamlined using various tools and services. The best approach depends on:

- **Scale of deployment** – Small-scale apps vs. enterprise-level infrastructure.
- **Cloud provider** – AWS, Azure, Google Cloud offer managed certificate services.
- **Automation level** – Manual vs. fully automated certificate issuance and renewal.

Common SSL/TLS management options:

1. **Let’s Encrypt** – Free, automated certificate issuance and renewal.

2. **Cloud-managed solutions** – AWS Certificate Manager (ACM), Azure Key Vault, Google Cloud Certificates.
3. **Self-signed certificates** – For internal environments or testing.
4. **Commercial Certificate Authorities (CAs)** – Sectigo, DigiCert, GlobalSign, for enterprise-grade security.

This document will guide you through the **setup, deployment, automation, and management** of SSL/TLS certificates in DevOps environments, ensuring secure and scalable application deployments.

2. SSL/TLS Basics

2.1 What is SSL/TLS?

SSL (Secure Sockets Layer) and TLS (Transport Layer Security) are cryptographic protocols designed to encrypt communication over the internet. TLS has replaced SSL due to its enhanced security features, and TLS 1.3 is the latest version, offering improved performance and security.

How SSL/TLS Works:

1. **Client Hello** – The client (browser or application) initiates a handshake with the server, listing supported cryptographic algorithms.
2. **Server Hello** – The server selects the strongest mutual encryption method and sends its SSL/TLS certificate.
3. **Certificate Validation** – The client verifies the server's certificate against trusted Certificate Authorities (CAs).
4. **Key Exchange** – Both sides securely exchange cryptographic keys.
5. **Secure Communication Begins** – All transmitted data is now encrypted.

2.2 Types of SSL/TLS Certificates

SSL/TLS certificates come in different types based on validation level and functionality.

Based on Validation Level:

1. **Domain Validation (DV)**
   - Basic validation; verifies domain ownership.
   - Issued quickly (minutes to hours).
   - Suitable for blogs, personal sites.
2. **Organization Validation (OV)**
   - Requires business identity verification.
   - Ensures website legitimacy.
   - Used for e-commerce and company portals.

3. **Extended Validation (EV)**
    - Highest level of validation.
    - Displays a company name in the address bar (e.g., green bar in some browsers).
    - Recommended for financial institutions and government sites.

Based on Usage:

1. **Single Domain Certificate** – Protects one domain (e.g., example.com).
2. **Wildcard Certificate** – Secures a domain and its subdomains (*.example.com).
3. **Multi-Domain (SAN) Certificate** – Protects multiple domains (example.com, test.com).

2.3 Understanding Certificate Authorities (CAs)

Certificate Authorities (CAs) are trusted entities that issue SSL/TLS certificates. Examples include:

- **Free CAs**: Let's Encrypt (widely used, automated renewals).
- **Commercial CAs**: DigiCert, Sectigo, GlobalSign (offer extended validation, warranty, and additional security).

How CA-Issued Certificates Work:

- The website owner submits a Certificate Signing Request (CSR) to the CA.
- The CA verifies the domain or business identity.
- The certificate is issued and installed on the web server.
- Clients (browsers, apps) trust the certificate if the CA is in their trusted store.

2.4 Public vs. Private Certificates

- **Public Certificates**: Issued by a trusted CA, used for public-facing websites.

- **Private Certificates**: Used internally in enterprises, issued by private (e.g., Active Directory Certificate Services).

3. Generating & Configuring SSL/TLS Certificates

In this section, we'll cover the step-by-step process to generate, configure, and install SSL/TLS certificates using different methods, including **Let's Encrypt, OpenSSL, and cloud-managed services**.

3.1 Choosing the Right SSL/TLS Certificate Generation Method

Before generating a certificate, decide which method suits your use case:

Method	Best For	Pros	Cons
Let's Encrypt	Public websites	Free, automated, widely supported	Expires every 90 days, needs renewal automation
OpenSSL (Self-Signed)	Internal testing, private servers	No external CA required, instant	Not trusted by browsers without manual installation
Cloud CA (AWS ACM, Azure Key Vault, GCP)	Cloud-native applications	Fully managed, auto-renewal	Limited to cloud environments
Commercial CA (DigiCert, Sectigo, GlobalSign, etc.)	Enterprise, e-commerce, financial services	Extended validation, warranty	Expensive, manual validation required

3.2 Generating an SSL/TLS Certificate Using Let's Encrypt (Recommended for Public Websites)

Let's Encrypt provides **free, automated SSL certificates**. Here's how to generate one:

Step 1: Install Certbot

On a Linux server (Ubuntu/Debian):

```
sudo apt update
sudo apt install certbot
```

For NGINX or Apache, install the respective Certbot plugin:

```
sudo apt install python3-certbot-nginx  # For NGINX
sudo apt install python3-certbot-apache # For Apache
```

Step 2: Generate the Certificate

For NGINX:

```
sudo certbot --nginx -d example.com -d www.example.com
```

For Apache:

```
sudo certbot --apache -d example.com
```

This command:
✓ Requests a certificate from Let's Encrypt.
✓ Configures NGINX/Apache automatically.
✓ Sets up auto-renewal.

Step 3: Verify SSL

Certificate Run:

```
sudo certbot certificates
```

To renew manually:

```
sudo certbot renew --dry-run
```

3.3 Generating a Self-Signed SSL Certificate (For Internal Use & Testing)

Self-signed certificates are useful for development and internal applications.

Step 1: Generate a Private Key & CSR

```
openssl req -newkey rsa:2048 -nodes -keyout mykey.pem -out myrequest.csr
```

- mykey.pem → Private key
- myrequest.csr → Certificate Signing Request

Step 2: Create a Self-Signed Certificate

```
openssl x509 -req -days 365 -in myrequest.csr -signkey mykey.pem -out
mycertificate.pem
```

This generates mycertificate.pem, which can be installed on a web server.

Step 3: Configure NGINX/Apache to Use the Certificate

For **NGINX**, add to your config (/etc/nginx/sites-available/default):

```
server {
   listen 443 ssl;
   server_name example.com;
   ssl_certificate /path/to/mycertificate.pem;
   ssl_certificate_key /path/to/mykey.pem;
}
```

Then restart:

```
sudo systemctl restart nginx
```

For **Apache**, update ssl.conf:

```
<VirtualHost *:443>
   ServerName example.com
   SSLEngine on
   SSLCertificateFile /path/to/mycertificate.pem
   SSLCertificateKeyFile /path/to/mykey.pem
</VirtualHost>
```

Restart Apache:

```
sudo systemctl restart apache2
```

3.4 Generating SSL/TLS Certificates in Cloud Services (AWS, Azure, GCP)

If you're deploying on a cloud provider, they offer **managed certificate services**: **AWS Certificate Manager (ACM)**

- Issue SSL certificates for **free** (but only works with AWS services like ELB, CloudFront).
- Automatically renews certificates.

To request a certificate via AWS CLI:

```
aws acm request-certificate --domain-name example.com --validation-method DNS
```

Azure Key Vault for SSL Certificates

- Stores SSL certificates securely.
- Can be integrated with Azure App Services &

Kubernetes. To import a certificate to Azure Key Vault:

```
az keyvault certificate import --vault-name MyKeyVault --name MySSLCert --file mycertificate.pfx
```

Google Cloud Managed Certificates

- Works with Google Cloud Load Balancers.
- Auto-renewal

enabled. Create a managed

certificate:

```
apiVersion: networking.gke.io/v1beta1
kind: ManagedCertificate
metadata:
  name: my-managed-cert
spec:
  domains:
    - example.com
```

3.5 Automating SSL/TLS Certificate Issuance & Renewal

For Let's Encrypt certificates, auto-renewal is **crucial** (since they expire every 90 days). Certbot automatically installs a cron job for renewal. You can check it with:

```
systemctl list-timers | grep certbot
```

Or manually add a cron job:

```
crontab -e
```

Add this line:

```
0 3 * * * certbot renew --quiet
```

This renews certificates daily at 3 AM.

3.6 Verifying & Testing SSL/TLS Certificates

Once the SSL certificate is installed, test it using:

✅ **Check if SSL is active (CLI):**

```
openssl s_client -connect example.com:443
```

✅ **Online SSL Testing Tools:**

- SSL Labs (Qualys) – Full SSL report
- [Google Chrome DevTools](F12 > Security Tab) – Shows certificate validity

✅ **Check Expiry Date (Linux command):**

```
openssl x509 -enddate -noout -in /path/to/mycertificate.pem
```

4. Deploying SSL/TLS Certificates in Web Applications

This section covers how to deploy SSL/TLS certificates on different platforms, including **web servers (NGINX, Apache, IIS), cloud services (AWS, Azure, GCP), and containerized applications (Docker, Kubernetes).**

4.1 Configuring SSL for Web Servers

After generating an SSL certificate, it needs to be configured in the web server hosting your application.

4.1.1 Deploying SSL on NGINX

Step 1: Copy the SSL Certificate & Key

Ensure you have:

- fullchain.pem (SSL certificate)
- privkey.pem (Private key)

Place them in /etc/ssl/certs/ and /etc/ssl/private/ respectively.

Step 2: Edit the NGINX Configuration

Modify your site configuration file, usually located at /etc/nginx/sites-available/example.com:

```
server {
  listen 443 ssl;
  server_name example.com;

  ssl_certificate /etc/ssl/certs/fullchain.pem;
  ssl_certificate_key /etc/ssl/private/privkey.pem;

  ssl_protocols TLSv1.2 TLSv1.3;
  ssl_ciphers HIGH:!aNULL:!MD5;
}
```

Step 3: Restart NGINX

```
sudo systemctl restart nginx
```

4.1.2 Deploying SSL on Apache

Step 1: Enable SSL Module

```
sudo a2enmod ssl
```

Step 2: Configure Virtual Host for SSL

Edit the Apache SSL configuration file (/etc/apache2/sites-available/default-ssl.conf):

```
<VirtualHost *:443>
   ServerName example.com
   SSLEngine on
   SSLCertificateFile /etc/ssl/certs/fullchain.pem
   SSLCertificateKeyFile /etc/ssl/private/privkey.pem
   SSLCipherSuite HIGH:!aNULL:!MD5
</VirtualHost>
```

Step 3: Restart Apache

```
sudo systemctl restart apache2
```

4.2 Deploying SSL in Cloud Services

4.2.1 AWS: Using AWS Certificate Manager (ACM) with Load Balancer Step 1: Request a Certificate in AWS ACM

```
aws acm request-certificate --domain-name example.com --validation-method DNS
```

AWS will provide a DNS record to verify domain ownership.

Step 2: Attach SSL Certificate to an Application Load Balancer

- Go to **EC2 > Load Balancers**
- Select your **ALB**, go to **Listeners**, and add an HTTPS listener.

- Choose the **ACM Certificate** and save.

AWS ACM **automatically renews** the certificate!

4.2.2 Azure: Using Azure Application Gateway with SSL Step 1: Upload SSL Certificate to Azure Key Vault

Convert your certificate to .pfx:

```
openssl pkcs12 -export -out certificate.pfx -inkey privkey.pem -in fullchain.pem
```

Upload the .pfx to Azure Key Vault:

```
az keyvault certificate import --vault-name MyKeyVault --name MySSLCert --file
    certificate.pfx
```

Step 2: Assign Certificate to Azure Application Gateway

- Open **Azure Portal > Application Gateway**
- Go to **Listeners**, add HTTPS, and select your certificate from **Key Vault**

Azure will handle SSL termination and renewal.

4.2.3 Google Cloud: Using SSL with Load Balancer

Step 1: Create a Managed Certificate

Create a ManagedCertificate resource:

```
apiVersion: networking.gke.io/v1
kind: ManagedCertificate
metadata:
  name: my-managed-cert
spec:
  domains:
    - example.com
```

Step 2: Attach Certificate to Load Balancer

- Navigate to **Google Cloud Console > Load Balancers**

- Add an **HTTPS Listener** and select **ManagedCertificate**

4.3 Securing API Endpoints with SSL/TLS

APIs should enforce SSL to ensure **secure communication**.

4.3.1 Enforcing HTTPS in NGINX for APIs

If your API is running on NGINX, enforce HTTPS:

```
server {
  listen 80;
  server_name api.example.com; return
      301 https://$host$request_uri;
   }

   server {
  listen 443 ssl;
  server_name api.example.com;

  ssl_certificate /etc/ssl/certs/fullchain.pem;
  ssl_certificate_key /etc/ssl/private/privkey.pem;

  location / {
     proxy_pass http://backend_server;
     proxy_set_header X-Forwarded-Proto https;
      }
   }
```

4.3.2 Securing Kubernetes API Endpoints

If your API runs inside Kubernetes, use **Ingress with TLS**:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: api-ingress
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
spec:
  tls:
  - hosts:
    - api.example.com
    secretName: tls-secret
  rules:
  - host: api.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: api-service
            port:
              number: 443
```

Ensure your certificate is stored in Kubernetes as a secret:

```
kubectl create secret tls tls-secret --cert=fullchain.pem --key=privkey.pem
```

4.4 Redirecting HTTP to HTTPS

To enforce HTTPS, configure redirects:

For NGINX:

```
server
  { listen
  80;
  server_name example.com;
  return 301 https://$host$request_uri;
}
```

For Apache:

Add this to .htaccess:

```
RewriteEngine On
RewriteCond %{HTTPS} !=on
RewriteRule ^ https://%{HTTP_HOST}%{REQUEST_URI} [L,R=301]
```

4.5 Verifying SSL Deployment

After deploying the SSL certificate, test it using:

✅ **Check if SSL is active (CLI):**

```
openssl s_client -connect example.com:443
```

✅ **Online SSL Testing Tools:**

- SSL Labs (Qualys) – Full SSL report
- [Google Chrome DevTools](F12 > Security Tab) – Shows certificate validity

✅ **Check Expiry Date (Linux command):**

```
openssl x509 -enddate -noout -in /etc/ssl/certs/fullchain.pem
```

5. Automating SSL/TLS Management

Managing SSL/TLS certificates manually can be time-consuming, especially in production environments where frequent renewals and deployments are required. This section focuses on **automating SSL/TLS certificate issuance, renewal, and deployment** using tools like **Certbot, ACME clients, cloud automation, and Kubernetes cert-manager**.

5.1 Why Automate SSL/TLS Certificate Management?

◈ **Auto-Renewals:** Prevents downtime due to expired certificates.
◈ **Security Compliance:** Ensures HTTPS enforcement across all services.
◈ **Scalability:** Easily handles SSL/TLS for multiple domains and microservices.
◈ **DevOps Efficiency:** Reduces manual intervention with continuous integration.

5.2 Automating SSL/TLS Renewal with Certbot (Let's Encrypt)

Let's Encrypt certificates expire **every 90 days**, so automation is necessary. Certbot provides a built-in mechanism for auto-renewal.

5.2.1 Verify Certbot Auto-Renewal

By default, Certbot installs a renewal timer. Check if it's running:

```
systemctl list-timers | grep certbot
```

5.2.2 Manually Configure a Cron Job for Certbot

To ensure certificates renew automatically, add a cron job:

```
crontab -e
```

Add this line:

```
0 3 * * * certbot renew --quiet
```

This runs the renewal check daily at **3 AM**.

5.2.3 Test Auto-Renewal

Run a dry-run test:

```
sudo certbot renew --dry-run
```

If successful, it means renewal is working correctly.

5.3 Using ACME Clients for Automated SSL Management

ACME (Automated Certificate Management Environment) is the protocol used by **Let's Encrypt** and other CAs for automatic certificate issuance. Alternative ACME clients include:

ACME Client	Use Case	Pros	Cons
Certbot	Public websites	Widely supported, free	Requires manual setup
Caddy	Reverse proxies, API gateways	Built-in SSL automation	Less customizable
acme.sh	Lightweight automation	Shell script-based	Requires manual scripting
lego	Kubernetes & cloud	Integrates with cert-manager	More complex setup

5.3.1 Automating SSL with acme.sh

If you prefer a lightweight alternative to Certbot:

```
curl https://get.acme.sh | sh
```

Generate and install a certificate:

```
acme.sh --issue --dns dns_cf -d example.com

acme.sh --install-cert -d example.com --key-file /etc/nginx/ssl/privkey.pem
-- fullchain-file /etc/nginx/ssl/fullchain.pem
```

This integrates with **Cloudflare DNS** to automatically issue and deploy SSL certificates.

5.4 Cloud-Based SSL Management

Cloud providers offer fully managed SSL solutions with auto-

5.4.1 AWS Certificate Manager (ACM) Auto-Renewal

AWS ACM automatically renews certificates for **ALB, CloudFront, and API Gateway**.

To verify auto-renewal status:

```
aws acm describe-certificate --certificate-arn
arn:aws:acm:region:account- id:certificate/certificate-id
```

5.4.2 Azure Key Vault SSL Automation

Azure can auto-renew certificates stored in **Key Vault**.

Enable auto-renewal using PowerShell:

```
Set-AzKeyVaultCertificateIssuer -VaultName "MyKeyVault" -Name "DigiCert" -
AutoRenew
```

5.5 Automating SSL/TLS in Kubernetes with cert-manager

For **containerized applications**, Kubernetes **cert-manager** automates certificate issuance and renewal.

5.5.1 Install cert-manager in Kubernetes

Deploy cert-manager using Helm:

```
helm repo add jetstack https://charts.jetstack.io
```

helm install cert-manager jetstack/cert-manager --namespace cert-manager
-- create-namespace --set installCRDs=true

5.5.2 Configure Let’s Encrypt Issuer

Create a ClusterIssuer to automate SSL issuance:

```
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
```

Apply the configuration:

```
kubectl apply -f cluster-issuer.yaml
```

5.5.3 Issue SSL Certificates Automatically for Services

Modify your Kubernetes Ingress to request a TLS certificate:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-app-ingress
  annotations:
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
spec:
  tls:
  - hosts:
    - myapp.example.com
  secretName: myapp-tls
  rules:
  - host: myapp.example.com
```

```
http:
 paths:
    - path: /
     pathType: Prefix
     backend:
   service:
    name: my-app-service
       port:
     number: 80
```

This automatically provisions SSL certificates for myapp.example.com.

5.6 Monitoring SSL/TLS Expiry & Renewals

To avoid downtime, set up alerts for expiring certificates.

5.6.1 Monitor SSL Expiry with OpenSSL

Check certificate expiration for a domain:

```
openssl s_client -connect example.com:443 | openssl x509 -noout -dates
```

5.6.2 Use SSL Monitoring Tools

- SSL Labs – Comprehensive SSL tests
- Nagios – Monitors SSL certificates
- Zabbix – Sends alerts when certificates are near expiration

5.6.3 Automate Expiry Alerts via Cron Jobs

Create a script to check SSL expiry and send alerts:

```
#!/bin/bash
DOMAIN="example.com"
EXPIRY_DATE=$(openssl s_client -connect $DOMAIN:443 2>/dev/null | openssl
x509 -noout -enddate | cut -d= -f2)
DAYS_LEFT=$(( ($(date -d "$EXPIRY_DATE" +%s) - $(date +%s)) / 86400 ))
```

```
if [ "$DAYS_LEFT" -lt 15 ]; then
  echo "WARNING: SSL certificate for $DOMAIN expires in $DAYS_LEFT days!" | mail -s "SSL Expiry Alert" admin@example.com
fi
```

Schedule this script using a cron job:

```
0 6 * * * /path/to/ssl-check.sh
```

This **sends an email alert** if the certificate is expiring within **15 days**.

6. Best Practices for SSL/TLS Security

Setting up SSL/TLS certificates is just the first step. To ensure a robust security posture, you need to follow best practices for encryption, certificate management, and compliance. This section covers industry-recommended **SSL/TLS hardening strategies, secure cipher configurations, certificate lifecycle management, and compliance requirements**.

6.1 Enforcing Strong Encryption Standards

To ensure secure communication, avoid outdated and vulnerable encryption protocols.

6.1.1 Disable Weak Protocols (SSLv3, TLS 1.0, TLS 1.1)

TLS 1.2 or higher is recommended. To enforce this in Nginx:

```
ssl_protocols TLSv1.2 TLSv1.3;
```

For Apache:

```
SSLProtocol -SSLv3 -TLSv1 -TLSv1.1 +TLSv1.2 +TLSv1.3
```

6.1.2 Use Secure Cipher Suites

A strong cipher suite ensures encryption integrity. In Nginx, configure:

```
ssl_ciphers 'ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384';
```

```
ssl_prefer_server_ciphers on;
```

Check your web server's current SSL configuration:

```
openssl ciphers -v
```

Use Mozilla SSL Configuration Generator for best settings.

6.2 Enforcing HTTP Strict Transport Security (HSTS)

HSTS ensures browsers only connect via HTTPS, preventing downgrade attacks.

Enable HSTS in Nginx:

```
add_header Strict-Transport-Security "max-age=31536000; includeSubDomains; preload"
always;
```

In Apache:

```
Header always set Strict-Transport-Security "max-age=31536000; includeSubDomains; preload"
```

⚠ **Caution:** Ensure your site supports HTTPS fully before enabling HSTS to prevent accessibility issues.

6.3 OCSP Stapling for Faster Certificate Verification

Online Certificate Status Protocol (OCSP) stapling improves TLS handshake performance.

Enable OCSP stapling in Nginx:

```
ssl_stapling on;
ssl_stapling_verify on;
ssl_trusted_certificate /etc/ssl/certs/ca-certificates.crt;
```

For Apache:

```
SSLUseStapling on
SSLStaplingCache shmcb:/var/run/ocsp(128000)
```

Verify OCSP response:

```
openssl s_client -connect example.com:443 -status | grep "OCSP Response Status"
```

6.4 Secure Key and Certificate Storage

Proper storage and access control of private keys prevent security breaches.

6.4.1 Protect Private Keys

- **Set strict permissions:**

```
chmod 600 /etc/ssl/private/server.key
```

- **Use Hardware Security Modules (HSM)** for high-security environments.

- **Avoid hardcoding private keys** in application code or repositories.

6.4.2 Secure Cloud Storage for Certificates

- **AWS Key Management Service (KMS)** for cloud-based encryption.
- **Azure Key Vault** for managed certificate storage.

6.5 Regular SSL/TLS Auditing & Monitoring

Frequent audits help detect weak configurations and expired certificates.

6.5.1 Use SSL Scanner Tools

- **SSL Labs (Qualys):**

```
curl https://www.ssllabs.com/ssltest/analyze.html?d=example.com
```

- **Nmap SSL Scan:**

```
nmap --script ssl-enum-ciphers -p 443 example.com
```

- **Test certificate expiry:**

```
openssl x509 -noout -dates -in /etc/ssl/certs/example.com.pem
```

6.5.2 Automate Expiry Alerts with Monitoring Tools

Use **Nagios**, **Zabbix**, or **Prometheus + Grafana** to monitor certificates.

Example: Add a **Nagios SSL certificate check**:

```
define command
  { command_name
  check_ssl_cert
command_line $USER1$/check_http -H $HOSTADDRESS$ -S -C 15
}
```

This alerts if an SSL certificate is expiring in **15 days**.

6.6 Compliance with SSL/TLS Security Standards

Organizations must follow **regulatory standards** to avoid security risks.

6.6.1 PCI DSS (Payment Card Industry Data Security Standard)

◈ **Requirement:** Use TLS 1.2+ for secure transactions.
◈ **Verification:** Test with

```
curl -v --tlsv1.0 https://example.com
```

If the connection succeeds, TLS 1.0 is enabled (which is insecure).

6.6.2 GDPR & Data Protection Compliance

◈ **Requirement:** Encrypt all customer data in transit.
◈ **Action:** Use **HSTS, TLS 1.3, and strong ciphers**.

6.6.3 HIPAA (For Healthcare Applications)

◈ **Requirement:** Protect PHI (Protected Health Information) with encryption.
◈ **Action:** Use **AES-256** encryption for all medical data transmissions.

6.7 Revoking and Replacing Compromised Certificates

If a private key is compromised, immediately **revoke** and **replace** the certificate.

6.7.1 Revoking a Let's Encrypt Certificate

```
certbot revoke --cert-path /etc/letsencrypt/live/example.com/fullchain.pem
```

6.7.2 Generate a New Key and Certificate

```
openssl req -new -newkey rsa:4096 -nodes -keyout new_key.pem -out
new_csr.pem
```

Submit the new CSR to a Certificate Authority for a replacement.

7. Troubleshooting SSL/TLS Issues

Even with a properly configured SSL/TLS setup, issues can arise due to misconfigurations, expired certificates, or compatibility problems. This section covers **common SSL/TLS errors, debugging techniques, and troubleshooting steps** to resolve them.

7.1 Common SSL/TLS Errors and Their Fixes

Error	**Cause**	**Solution**
ERR_CERT_DATE_INVALID	Certificate expired or system time incorrect	Renew certificate, check system clock
ERR_SSL_VERSION_OR_CIPHER_MISMATCH	Outdated TLS version or weak ciphers	Enable TLS 1.2/1.3, update cipher list
NET::ERR_CERT_COMMON_NAME_INVALID	Incorrect domain name in certificate	Ensure CN matches domain, reissue cert
SEC_ERROR_EXPIRED_CERTIFICATE	Expired or revoked certificate	Renew certificate, check CRL/OCSP status
SSL Handshake Failed	Cipher mismatch, missing intermediate certs	Configure proper cipher suites, include full chain

Error	Cause	Solution
Too many redirects	Misconfigured HTTPS redirection	Fix redirect rules in Nginx/Apache

7.2 Checking SSL/TLS Certificate Expiry

To manually check certificate expiration:

```
openssl s_client -connect example.com:443 | openssl x509 -noout -dates
```

Example output:

```
notBefore=Mar 10 12:00:00 2025 GMT
notAfter=Jun 10 12:00:00 2025 GMT
```

If notAfter is close to the current date, renewal is required.

7.3 Verifying SSL/TLS Configuration

7.3.1 Check SSL Certificate Installation

Use **OpenSSL** to verify if the server is sending the correct certificate:

```
openssl s_client -connect example.com:443
```

Look for:
✅ **Certificate chain complete**
✅ **Common Name (CN) matches domain**
✅ **No self-signed certificate errors**

7.3.2 Validate Certificate Chain

Missing intermediate certificates can cause trust issues. Test with:

```
openssl verify -CAfile chain.pem server.crt
```

If verification fails, add missing intermediate certificates to the chain.

7.4 Debugging SSL/TLS Errors in Web Servers

7.4.1 Checking SSL Logs in Nginx

If SSL fails in **Nginx**, check logs:

```
tail -f /var/log/nginx/error.log
```

Common issues:
❌ SSL: error:1408F10B:SSL routines:ssl3_get_record:wrong version number – Fix by enabling TLS 1.2/1.3.
❌ ssl_stapling ignored, issuer certificate not found – Ensure correct **OCSP stapling** configuration.

7.4.2 Apache SSL Debugging

To check Apache's SSL configuration:

```
apachectl configtest
```

Restart Apache after fixing configuration issues:

```
systemctl restart apache2
```

7.5 Fixing Browser-Specific SSL Issues

7.5.1 Clear SSL Cache

For **Chrome**, clear SSL cache:

1. Open chrome://net-internals/#ssl.
2. Click **Clear SSL state**.

7.5.2 Check Mixed Content Errors

If HTTPS is enabled but resources load via HTTP, browsers may block them.
Use Chrome DevTools (**F12 > Console**) to identify mixed content issues.
Fix by replacing http:// URLs with https:// in your website's code.

7.6 Resolving SSL Certificate Revocation Issues

If a certificate has been **revoked**, check its **OCSP status**:

```
openssl s_client -connect example.com:443 -status | grep OCSP
```

If the response is revoked, issue a new certificate immediately.

7.7 Testing SSL/TLS Security Post-Configuration

7.7.1 Use SSL Labs Test

Scan your domain with:
👉 SSL Labs: https://www.ssllabs.com/ssltest/
Aim for **A+ rating** by fixing reported issues.

7.7.2 Scan for Weak Ciphers with Nmap

```
nmap --script ssl-enum-ciphers -p 443 example.com
```

If weak ciphers are found, update your web server's SSL configuration.

7.8 Automating SSL/TLS Issue Detection

Set up **automated SSL health checks** to detect issues early.

7.8.1 Monitor SSL Expiry Using Cron Jobs

Schedule a script to alert before expiration:

```
#!/bin/bash

DOMAIN="example.com"

EXPIRY_DATE=$(openssl s_client -connect $DOMAIN:443 2>/dev/null | openssl x509 -noout -enddate | cut -d= -f2)

DAYS_LEFT=$(( ($(date -d "$EXPIRY_DATE" +%s) - $(date +%s)) / 86400 ))


if [ "$DAYS_LEFT" -lt 15 ]; then

  echo "ALERT: SSL certificate for $DOMAIN expires in $DAYS_LEFT days!" | mail -s "SSL Expiry Warning" admin@example.com

fi
```

Schedule it in crontab -e:

```
0 7 * * * /path/to/ssl_check.sh
```

This runs daily at **7 AM** and sends alerts if an SSL certificate expires within **15 days**.

8. Conclusion and Final Recommendations

Securing web applications with **SSL/TLS certificates** is essential for ensuring encrypted communication, protecting user data, and maintaining trust. This document has covered **SSL/TLS certificate setup, renewal, automation,**
troubleshooting, and best practices to help DevOps teams efficiently manage certificates within their infrastructure.

8.1 Key Takeaways

✅ **SSL/TLS Implementation**

- Obtained and installed **SSL/TLS certificates** from trusted Certificate Authorities (CAs).
- Configured certificates in **Nginx, Apache, and cloud platforms**.
- Automated certificate renewal using **Certbot, ACME clients, and cloud providers**.

✅ **Security Enhancements**

- Enforced **TLS 1.2/1.3**, disabled weak protocols (SSLv3, TLS 1.0, TLS 1.1).
- Implemented **HSTS, OCSP stapling, and strong cipher suites**.
- Stored private keys securely and followed **certificate lifecycle best practices**.

✅ **Monitoring & Troubleshooting**

- Automated **SSL expiry alerts** with scripts and monitoring tools.
- Debugged SSL errors using **OpenSSL, Nmap, and SSL Labs**.
- Ensured **compliance with industry standards** (PCI DSS, GDPR, HIPAA).

8.2 Recommended Next Steps

1 Implement Infrastructure-as-Code (IaC) for SSL Management

- Use Terraform, Ansible, or Helm charts to define and automate certificate deployment.

2 **Adopt Certificate Management Platforms**

- Consider **AWS Certificate Manager (ACM), Let's Encrypt with Certbot, HashiCorp Vault** for centralized SSL management.

3 **Regularly Audit & Test SSL Configuration**

- Run scheduled **SSL vulnerability scans** using security tools like SSL Labs, Nmap, and Nessus.

4 **Train DevOps Teams on SSL Security Best Practices**

- Educate teams on **certificate revocation, chain of trust, and encryption hardening**.

5 **Prepare for Post-Quantum Cryptography (PQC)**

- Stay updated on **post-quantum cryptographic algorithms** to future-proof SSL/TLS security.

8.3 Final Thought: Automate & Stay Secure

SSL/TLS security isn't a **one-time task**—it requires **continuous monitoring, automation, and compliance**. By integrating SSL/TLS management into your DevOps pipeline, you can ensure **reliable encryption, prevent outages, and strengthen overall security posture**.

🔐 **Stay proactive, automate renewals, and audit SSL security regularly!**

Appendix: Additional Resources

📌 **Tools & Services for SSL/TLS Management**

- **Let's Encrypt** (Free SSL): https://letsencrypt.org/
- **SSL Labs Test**: https://www.ssllabs.com/ssltest/
- **Mozilla SSL Configuration Generator**: https://ssl-config.mozilla.org/
- **Nmap SSL Scanning**: https://nmap.org/nsedoc/scripts/ssl-enum- ciphers.html

📌 **Best Practices & Compliance**

- **PCI DSS Security Standards**: https://www.pcisecuritystandards.org/
- **GDPR Compliance Guide**: https://gdpr.eu/
- **HIPAA Encryption Standards**: https://www.hhs.gov/hipaa/