

AUTOMATING TLS in K8s



Part - 2

Self signed certificates
Become our own CA
Making website secure using own CA



Presented by

Sivakumar Reddy M

*Founder of
Linuscode Technologies Pte Ltd
Singapore*



PART-2:

This is PART-2 of our case study **Automating TLS in K8** series. You can check PART-1 using below link where I explained

- How HTTPS work
- What is CA
- SSL configuration in Nginx

Link: <https://www.linkedin.com/feed/update/urn:li:activity:7254131450734620673>

Introduction:

We assisted a client with their Kubernetes migration, addressing several challenges along the way. One key issue we resolved was managing TLS certificates, which had previously been a manual and time-consuming process. By automating certificate management, we ensured a smoother, more secure transition to production.

Client Background:

The client is a growing company in the retail sector, providing e-commerce services to a large customer base. With increasing demand, they needed a scalable and secure infrastructure to support their expanding operations. Their shift to Kubernetes aimed to enhance scalability, but they encountered several roadblocks along the way, particularly in maintaining secure communications across their services.

Challenges:

The client faced a few key challenges during the migration:

1. **Manual Certificate Management:** Their TLS certificate management was tedious and required a lot of manual work, leading to risks like expired certificates and potential downtime.
2. **Complex Infrastructure:** Moving to Kubernetes made things more complex, so they needed a solid solution for secure communication across their services.
3. **Knowledge Gaps:** Their engineering team was not fully familiar with automated certificate management and best practices for implementing HTTPS in a cloud environment.

Solutions Implemented:

To help the client, we implemented several key solutions:

1. **Automated TLS Certificate Management:** We set up an automated system using tools like Cert-Manager. This made issuing, renewing, and revoking TLS certificates much easier and less prone to error.
2. **Training and Knowledge Transfer:** We held training sessions for the client's engineering team, covering:
 - How HTTPS works and why it's important for security.
 - The role of Certificate Authorities (CAs).
 - Understanding self-signed certificates and when to use them.
 - How to create an internal CA for their services.
 - Automating certificate management in Kubernetes.
3. **Security Best Practices:** We shared best practices for managing certificates within a Kubernetes environment to ensure ongoing security as they grew.

What are self signed certificates?:

- Self-signed certificates are SSL/TLS certificates that are signed with its own private key, rather than being signed by a trusted CA.
- Like any other SSL/TLS certificate, it includes a public key, the identity of the server or user, and information about the certificate's validity.
- The certificate can be used to encrypt communications just like a certificate issued by a CA. However, it does not provide the same level of authentication because it has not been validated by a third-party authority.

When are self signed certificates used?:

They are commonly used for internal testing or development environments where security is needed, but the certificate doesn't need to be trusted outside the organisation.

How to self signed certificates?:

Openssl is the popular tool to create self signed certificates. In our last PART-1 we got the trusted certificate from CA. in this PART-2 we use a self signed certificate in the same nginx server.

Find the below example command which I created a self signed certificate for my domain daws81s.online. Provide all the information prompted

```
[ ec2-user@ip-172-31-18-137 ~ ]$ openssl req -x509 -newkey rsa:4096 -sha256 -nodes -keyout daws81s.online.key -out daws81s.online.crt -days
```

[illegible]

You are about to be asked to enter information that will be incorporated into your certificate request.

There are quite a few fields but you can leave some blank

For some fields there will be a default value,

If you enter '.', the field will be left blank.

— — — — —

Country Name (2 letter code) [XX]:SG

```
State or Province Name (full name) []:Singapore
```

Locality Name (eg, city) [Default City]:Singapore

```
Organization Name (eg, company) [Default Company Ltd]:Linuscode
Technologies Pte Ltd
```

```
Organizational Unit Name (eg, section) []:CloudOperations
```

```
Common Name (eg, your name or your server's hostname) []:daws81s.online
```

```
Email Address []:siva@linuscode.com
```

```
100.27.209.167 | 172.31.18.137 | t3.micro | null
```

```
[ ec2-user@ip-172-31-18-137 ~ ]$ ls -l daws*  
-rw-r--r-- 1 ec2-user ec2-user 2244 Oct 25 13:10 daws81s.online.crt  
-rw----- 1 ec2-user ec2-user 3272 Oct 25 13:09 daws81s.online.key
```

Here I am writing the explanation about the options in OpenSSL command.

- **-x509:** This option tells OpenSSL to generate a self-signed certificate instead of a certificate signing request (CSR).
- **-newkey rsa:4096:**
 - **newkey:** Creates a new private key and a certificate request.
 - **rsa:4096:** Specifies the type of key and its size. Here, RSA is used with a key size of 4096 bits, which is considered strong encryption.
- **-sha256:** Specifies the hashing algorithm to use when creating the certificate. SHA-256 is a widely used hashing function that provides strong security and is recommended over older algorithms like SHA-1.
- **-nodes:** This option means "no DES" (Data Encryption Standard), and it tells OpenSSL not to encrypt the private key with a passphrase. The resulting key is unencrypted, making it easier to use in automated environments where manual passphrase entry is not practical.
- **-keyout daws81s.online.key:** Specifies the output file for the generated private key. Here, the private key will be saved in a file named daws81s.online.key.
- **-out daws81s.online.crt:** Specifies the output file for the generated certificate. The certificate will be saved in a file named daws81s.online.crt.

Let's host a website using HTTPS Self Signed Certificate:

Requirements:

- AWS EC2
- A valid domain with admin access. My domain is daws81s.online. Change your hostedzone from your hosting provider to Route53 for fast updating of records.

Steps:

Create an EC2 instance using any OS. I am going to use RHEL based here.

Here I am using my own AMI with the name **devops-practice**. You can find **this in the us-east-1 region** which we use in our demo sessions based on RHEL-9. Username is ec2-user and password is DevOps321. Make sure you select no-key while launching.

SSL configuration in nginx:

Steps:

- Install nginx server in your EC2 instance.

```
sudo dnf install nginx -y
```

- Create a folder /etc/pki/nginx

```
sudo mkdir -p /etc/pki/nginx
```

- Move the private key and certificate

```
[ ec2-user@ip-172-31-18-137 ~ ]$ sudo mv daws81s.online.*
/etc/pki/nginx/

100.27.209.167 | 172.31.18.137 | t3.micro | null
[ ec2-user@ip-172-31-18-137 ~ ]$ ls -l /etc/pki/nginx/
total 8
-rw-r--r-- 1 ec2-user ec2-user 2244 Oct 25 13:10 daws81s.online.crt
-rw----- 1 ec2-user ec2-user 3272 Oct 25 13:09 daws81s.online.key
```

- Modify the nginx configuration to enable SSL. Comment http and uncomment https. Find the full file here, replace your domain name in SSL section.

```
# For more information on configuration, see:
# * Official English Documentation: http://nginx.org/en/docs/
# * Official Russian Documentation: http://nginx.org/ru/docs/

user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /run/nginx.pid;

# Load dynamic modules. See /usr/share/doc/nginx/README.dynamic.
include /usr/share/nginx/modules/*.conf;

events {
```

```

worker_connections 1024;
}

http {
    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
        '$status $body_bytes_sent "$http_referer" '
        '"$http_user_agent" "$http_x_forwarded_for"';

    access_log /var/log/nginx/access.log main;

    sendfile        on;
    tcp_nopush      on;
    tcp_nodelay      on;
    keepalive_timeout 65;
    types_hash_max_size 4096;

    include          /etc/nginx/mime.types;
    default_type      application/octet-stream;

    # Load modular configuration files from the /etc/nginx/conf.d directory.
    # See http://nginx.org/en/docs/nginx_core_module.html#include
    # for more information.
    include /etc/nginx/conf.d/*.conf;

    # server {
    #     listen      80;
    #     listen      [::]:80;
    #     server_name  _;
    #     root         /usr/share/nginx/html;

    #     # Load configuration files for the default server block.
    #     include /etc/nginx/default.d/*.conf;

    #     error_page 404 /404.html;
    #     location = /404.html {
    #     }

    #     error_page 500 502 503 504 /50x.html;
    #     location = /50x.html {
    #     }
    # }

    #Settings for a TLS enabled server.

```



```

server {
    listen      443 ssl http2;
    listen      [::]:443 ssl http2;
    # replace your domain name below
    server_name daws81s.online www.daws81s.online;
    root        /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/daws81s.online.crt";
    ssl_certificate_key "/etc/pki/nginx/daws81s.online.key";
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout 10m;
    ssl_ciphers PROFILE=SYSTEM;
    ssl_prefer_server_ciphers on;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    error_page 404 /404.html;
        location = /40x.html {
        }

    error_page 500 502 503 504 /50x.html;
        location = /50x.html {
        }
    }
}

```

- Check if the nginx configuration is fine or not.

```

[ root@ip-172-31-31-46 /etc/nginx ]# nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful

```

- Start nginx service


```
systemctl start nginx
```

- Update EC2 instance public IP in Route53 records.

Record details

[Edit record](#)

Record name

 daws81s.online

Record type

A

Value

 184.50.191.100

Alias

No

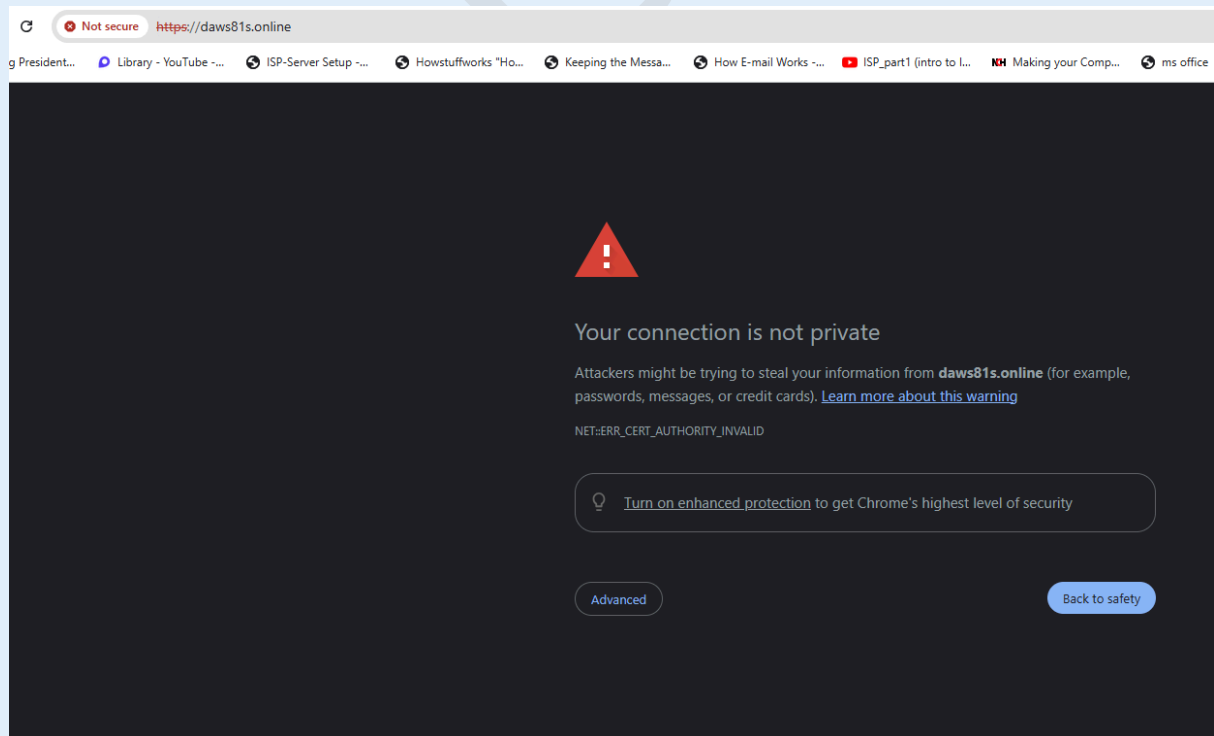
TTL (seconds)

1

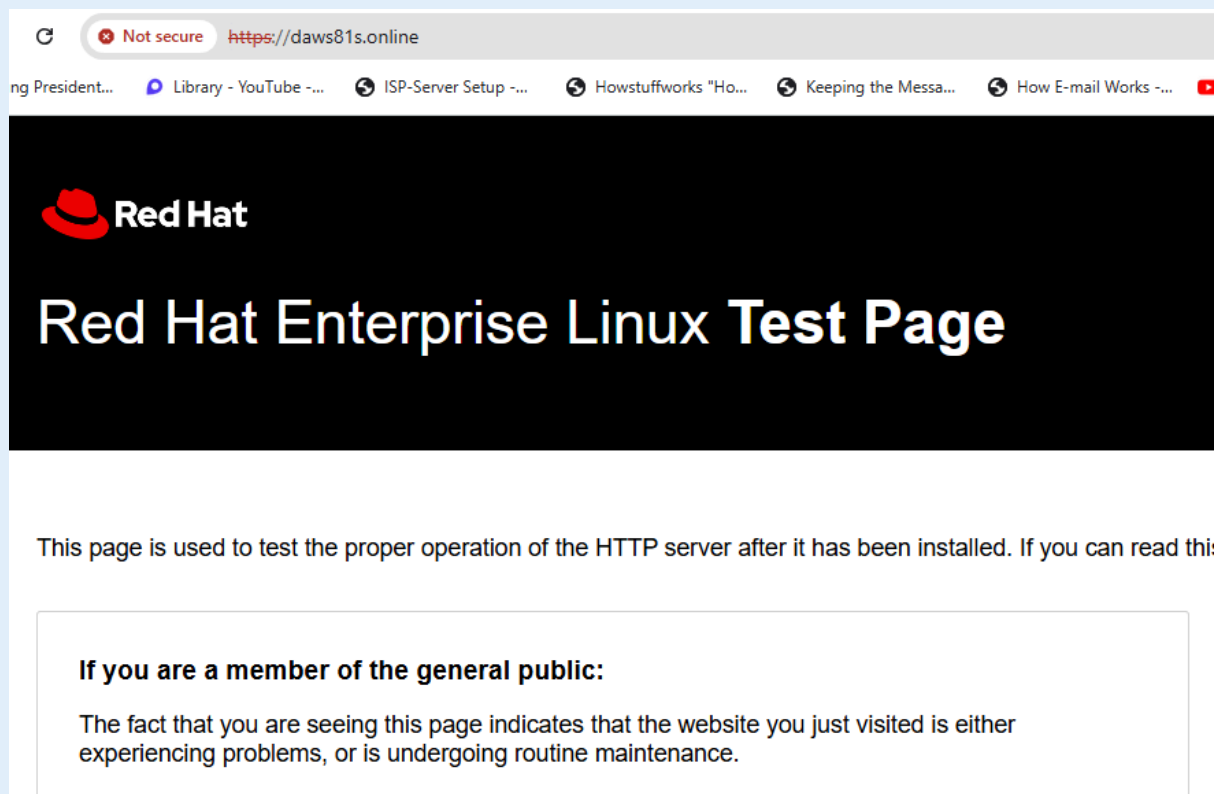
Routing policy

Simple

Congratulations, now your website is secure using https.



Click **Advanced and Proceed to daws81s.online (unsafe)**



Let's try a proper website. Remove the default content in nginx html folder.

```
[ root@ip-172-31-31-46 ~ ]# cd /usr/share/nginx/html/

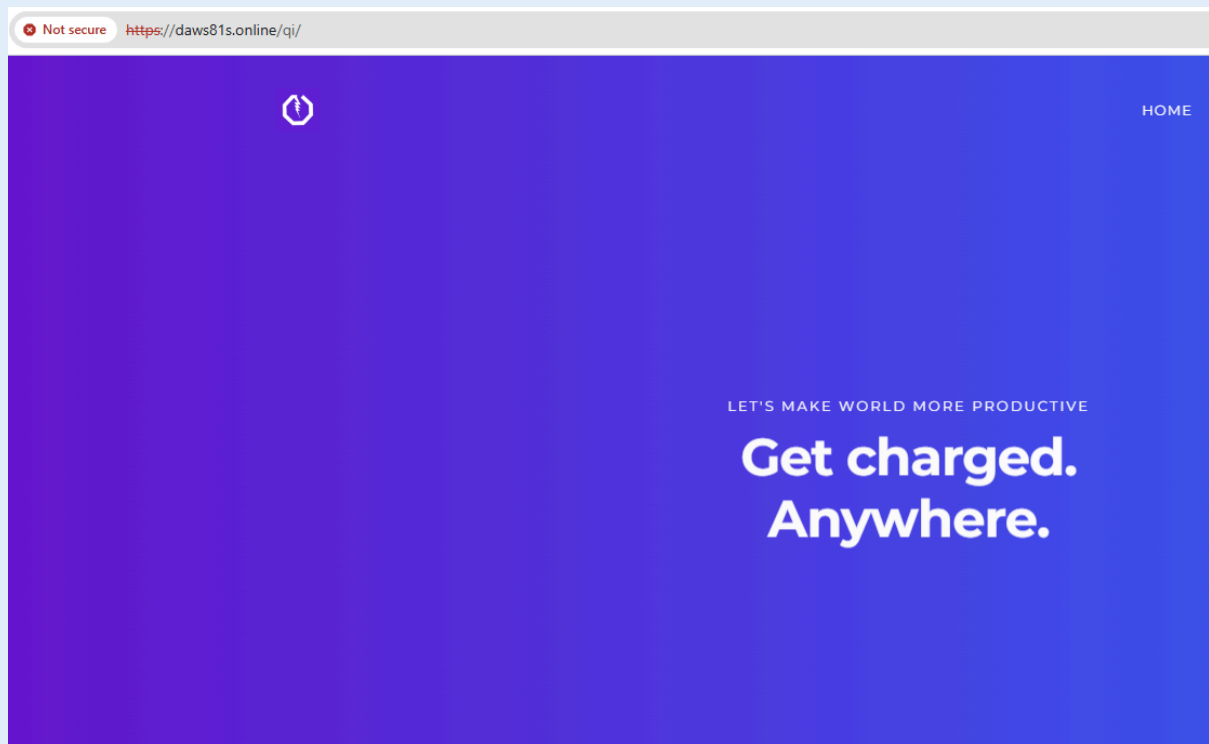
184.72.124.165 | 172.31.31.46 | t3.micro | null
[ root@ip-172-31-31-46 /usr/share/nginx/html ]# ls
404.html 50x.html icons index.html nginx-logo.png poweredby.png
system_noindex_logo.png

184.72.124.165 | 172.31.31.46 | t3.micro | null
[ root@ip-172-31-31-46 /usr/share/nginx/html ]# rm -rf index.html
```

- Clone the simple Qi Enabler site from below git.

```
[ root@ip-172-31-31-46 /usr/share/nginx/html ]# git clone
https://github.com/msivakumarreddy/qi.git
Cloning into 'qi'...
remote: Enumerating objects: 57, done.
remote: Counting objects: 100% (57/57), done.
remote: Compressing objects: 100% (55/55), done.
remote: Total 57 (delta 1), reused 57 (delta 1), pack-reused 0 (from 0)
Receiving objects: 100% (57/57), 1.86 MiB | 14.00 MiB/s, done.
Resolving deltas: 100% (1/1), done.
```

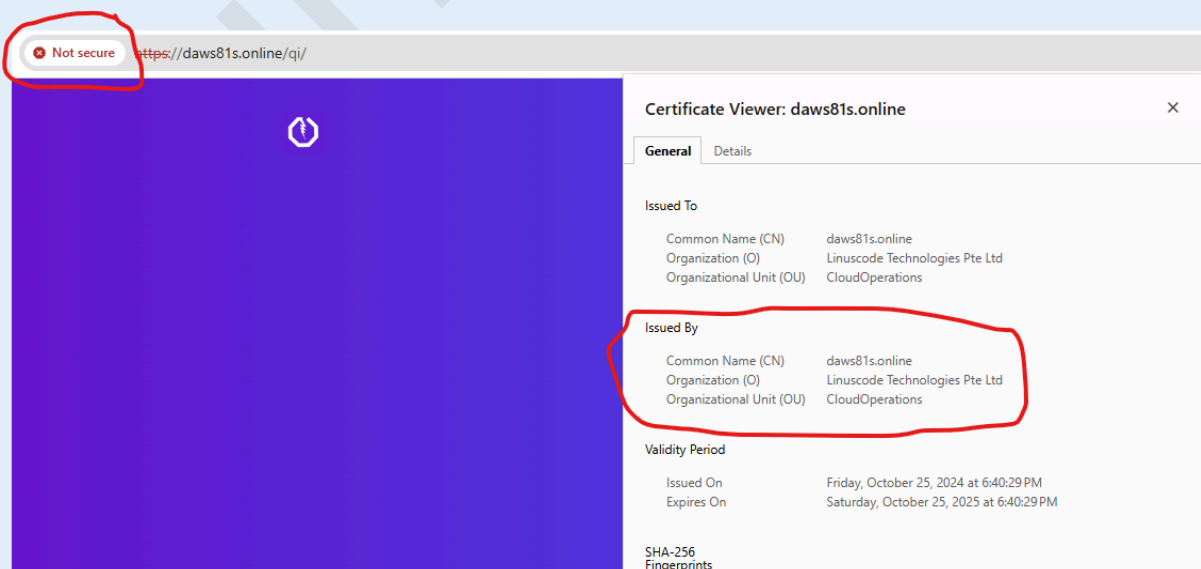
```
184.72.124.165 | 172.31.31.46 | t3.micro | null  
[ root@ip-172-31-31-46 /usr/share/nginx/html ]# systemctl restart nginx
```



Hurray, We are now able to make our website secure.

But wait,

Is our website really secure? Now we can clearly see it is insecure because it is not verified by CA instead it is verified by our own entity.



Limitations:

Not Trusted by Default: Browsers and clients will generally not trust self-signed certificates because they have not been signed by a trusted CA. Users will see a security warning when trying to access a website using a self-signed certificate.

No Identity Verification: A trusted CA verifies the identity of the entity requesting the certificate. Self-signed certificates do not go through this verification process, which means they do not provide assurance about the identity of the certificate owner.

Manual Distribution: To use self-signed certificates securely, they need to be manually distributed and installed in each client's trust store, which can be cumbersome for large environments.

Alternatives:

- **Certificates from a Public CA:** This we already saw in our last PART-1.
- **Internal CA:** Some organisations set up their own internal CA to issue certificates for internal services. This provides more control while still allowing easier management of trust within the organisation.

Internal own CA:

- An internal Certificate Authority (CA) is a private CA that is created and managed within an organisation to issue digital certificates for internal use.
- Unlike a public CA, which issues certificates that are trusted by external entities (such as websites accessed by internet users), an internal CA is used to manage and secure certificates for services, users, and devices within an organisation's own network.
- This allows the organisation to maintain control over how certificates are issued, used, and revoked.

Advantages:

- Cost savings
- Control and flexibility
- Security and privacy

When not to use own CA:

Step-1: Create a root private key that will be used to sign certificates. Enter the passkey for protection.

Step 2: Create the Root Certificate

Generate a root certificate using the private key. Enter the passphrase you set while generating the private key in the previous step.

```
[ ec2-user@ip-172-31-18-137 ~ ]$ openssl req -x509 -new -key
rootCA_linuscode.key -sha256 -days 3650 -out rootCA_linuscode.crt
Enter pass phrase for rootCA_linuscode.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a
DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:SG
State or Province Name (full name) []:Singapore
Locality Name (eg, city) [Default City]:Singapore
Organization Name (eg, company) [Default Company Ltd]:Linuscode
Technologies Pte Ltd
Organizational Unit Name (eg, section) []:IT Department
Common Name (eg, your name or your server's hostname) []:Linuscode Root
CA
Email Address []:siva@linuscode.com
```

Congrats. Technically you became root CA.

Now we need to create one Intermediate CA like we created root CA.

Why Intermediate CA:

Creating an Intermediate Certificate Authority (CA) offers several practical and security benefits, particularly for larger organisations or environments that require more controlled certificate management.

Enhanced Security:

The root CA is the most critical asset in the certificate chain, as it has the ultimate authority. If it were to be compromised, all certificates issued by it would lose their trust. By issuing certificates through an intermediate CA, you can keep the root CA offline and secure, reducing the risk of exposure.


```
.....+.....+.....+.....+.....+.....+.....+.....+.....+.....+.....+.....+
.+.....+.....+.....+.....+.....+.....+.....+.....+.....+.....+.....+.....+
+++++
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

Step-4: Create a certificate signing request for Intermediate CR.

```
[ ec2-user@ip-172-31-18-137 ~ ]$ openssl req -new -key
intermediateCA_linuscode.key -out intermediateCA_linuscode.csr
Enter pass phrase for intermediateCA_linuscode.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a
DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:SG
State or Province Name (full name) []:Singapore
Locality Name (eg, city) [Default City]:Singapore
Organization Name (eg, company) [Default Company Ltd]:Linuscode
Technologies Pte Ltd
Organizational Unit Name (eg, section) []:IT Department
Common Name (eg, your name or your server's hostname) []:Linuscode
Intermediate CA
Email Address []:info@linuscode.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Step-5: create a file called intermediateCA.linuscode.ext and enter below information.

```
vim intermediateCA_linuscode.ext
```

```
authorityKeyIdentifier=keyid,issuer
basicConstraints = CA:TRUE, pathlen:0
keyUsage = critical, digitalSignature, keyCertSign, cRLSign
```

.ext file information is important, because

1. Defining Key Usage and Extended Key Usage: The `.ext` file specifies critical extensions like `keyUsage` and `extendedKeyUsage`, which tell the browser the purpose of the certificate (e.g., server authentication). Without these extensions, browsers often flag the certificate as "unsupported" because it lacks the necessary usage permissions.
2. Adding Subject Alternative Names (SANs): Chrome (and most modern browsers) require the Subject Alternative Name (SAN) field to match the domain name, even if the Common Name (CN) is correct. The `.ext` file enables you to specify SANs, which is crucial for the certificate to be recognized for the domain.
3. Authority Information: The `.ext` file also includes `authorityKeyIdentifier` and `basicConstraints`, which help define the certificate hierarchy, specify whether the certificate is a CA, and indicate the chain of trust.

Without the `.ext` file, certificates may lack these essential extensions, causing errors or warnings in browsers. So, while it may feel like an additional step, including `.ext` files is important for compatibility and security in today's browser environment.

Step-6: Sign Intermediate CSR with root CA and generate the certificate

```
[ ec2-user@ip-172-31-18-137 ~ ]$ openssl x509 -req -in
intermediateCA_linuscode.csr -CA rootCA_linuscode.crt -CAkey
rootCA_linuscode.key -CAcreateserial -out intermediateCA_linuscode.crt
-days 825 -sha256 -extfile intermediateCA_linuscode.ext
```

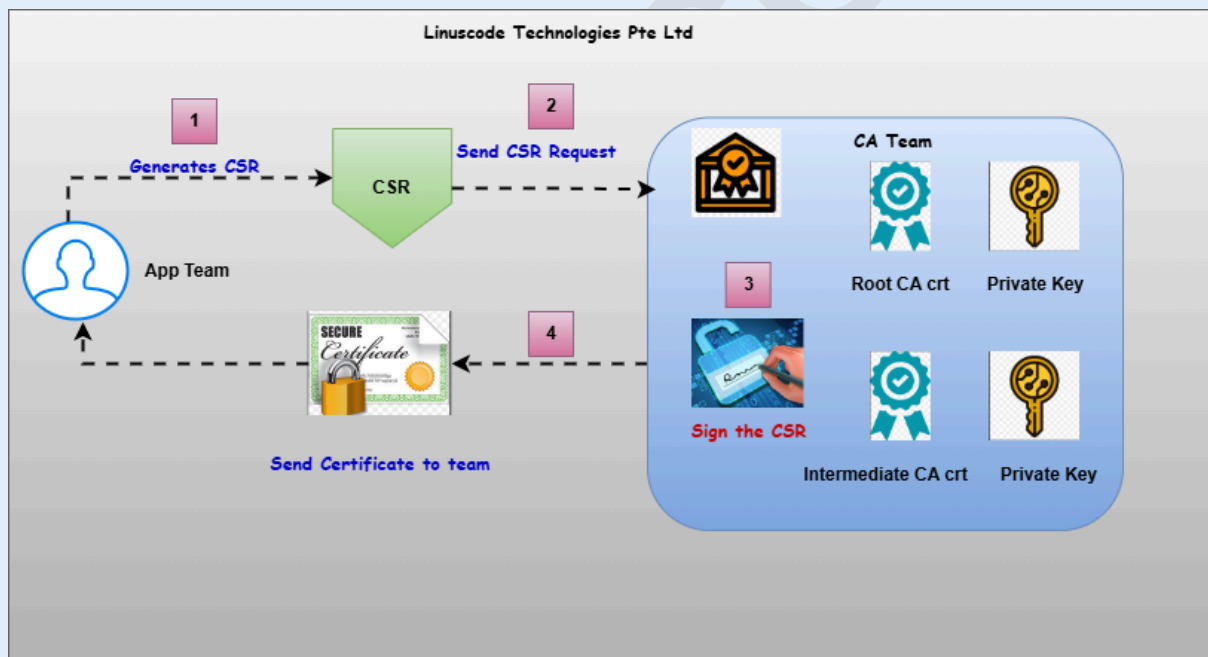
```
Certificate request self-signature ok
subject=C = SG, ST = Singapore, L = Singapore, O = Linuscode
Technologies Pte Ltd, OU = IT Department, CN = Linuscode Intermediate
CA, emailAddress = info@linuscode.com
Enter pass phrase for rootCA_linuscode.key:
```

Congratulations! 🎉 You've successfully created both the Root and Intermediate CAs, establishing a secure chain of trust. You're now ready to start issuing certificates for your domains with full authority! 🔧🔒

Now let's issue certificate to our domain daws81s.online

Step-7: Create a private key for your domain

Application team in our company creates a certificate signing request and then sends them to our internal CA to sign and provide a certificate.

[illegible]

```
[ ec2-user@ip-172-31-18-137 ~ ]$ openssl req -new -key daws81s.online.key -out daws81s.online.csr
```



What you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value,

If you enter '.', the field will be left blank.

Country Name (2 letter code) [XX]:SG

State or Province Name (full name) []:Singapore

Locality Name (eg, city) [Default City]:Singapore

Organization Name (eg, company) [Default Company Ltd]:Linuscode
Technologies Private Limited

Organizational Unit Name (eg, section) []:IT Department

Common Name (eg, your name or your server's hostname) []:daws81s.online

Email Address []:info@linuscode.com

Please enter the following 'extra' attributes
to be sent with your certificate request

A challenge password []:

An optional company name []:

Step-9: create a file called daws81s.online.ext and paste the below content

```
vim daws81s.online.ext
```

```
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth
subjectAltName = @alt_names

[alt_names]
DNS.1 = daws81s.online
```

Now its CA team turn. They use their intermediate certificate and private key to sign the CSR and send the digital certificate to the application team.

```
[ ec2-user@ip-172-31-18-137 ~ ]$ openssl x509 -req -in
daws81s.online.csr -CA intermediateCA_linuscode.crt -CAkey
intermediateCA_linuscode.key -CAcreateserial -out daws81s.online.crt
-days 365 -sha256 -extfile daws81s.online.ext
```

Certificate request self-signature ok

subject=C = SG, ST = Singapore, L = Singapore, O = Linuscode

```
Technologies Private Limited, OU = IT Department, CN = daws81s.online,  
emailAddress = info@linuscode.com  
Enter pass phrase for intermediateCA_linuscode.key:
```

The CA team now sends the intermediate certificate and domain certificate as a bundle to the application team.

```
cat daws81s.online.crt intermediateCA_linuscode.crt >  
daws81s.online.chain.crt
```

Now the application team has a crt bundle and private key with them. They can be configured in nginx as earlier.

```
[ ec2-user@ip-172-31-18-137 ~ ]$ ls -l daws*  
-rw-r--r-- 1 ec2-user ec2-user 3233 Nov  2 06:20 daws81s.online.chain.crt  
-rw-r--r-- 1 ec2-user ec2-user 1647 Nov  2 06:17 daws81s.online.crt  
-rw-r--r-- 1 ec2-user ec2-user 1119 Nov  2 05:59 daws81s.online.csr  
-rw-r--r-- 1 ec2-user ec2-user  202 Nov  2 06:16 daws81s.online.ext  
-rw----- 1 ec2-user ec2-user 1704 Nov  2 05:57 daws81s.online.key
```

Let's install these in the nginx server. Remove the previous self signed certificates if they exist. Slightly modify nginx conf to refer to bundle crt.

```
[ ec2-user@ip-172-31-18-137 ~ ]$ sudo systemctl stop nginx  
  
54.221.19.102 | 172.31.18.137 | t3.micro | null  
[ ec2-user@ip-172-31-18-137 ~ ]$ sudo rm -rf /etc/pki/nginx/*  
  
54.221.19.102 | 172.31.18.137 | t3.micro | null  
[ ec2-user@ip-172-31-18-137 ~ ]$ sudo cp daws81s.online.chain.crt  
/etc/pki/nginx/  
  
54.221.19.102 | 172.31.18.137 | t3.micro | null  
[ ec2-user@ip-172-31-18-137 ~ ]$ sudo cp daws81s.online.key  
/etc/pki/nginx/
```

```
#Settings for a TLS enabled server.

server {
    listen      443 ssl http2;
    listen      [::]:443 ssl http2;
    # replace your domain name below
    server_name daws81s.online www.daws81s.online;
    root        /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/daws81s.online.chain.crt";
    ssl_certificate_key "/etc/pki/nginx/daws81s.online.key";
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout 10m;
    ssl_ciphers PROFILE=SYSTEM;
    ssl_prefer_server_ciphers on;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;
}
```

Now restart the server.

```
sudo systemctl restart nginx
```

Now open the website in chrome.

Now you can see the certificate issued by our internal root CA.

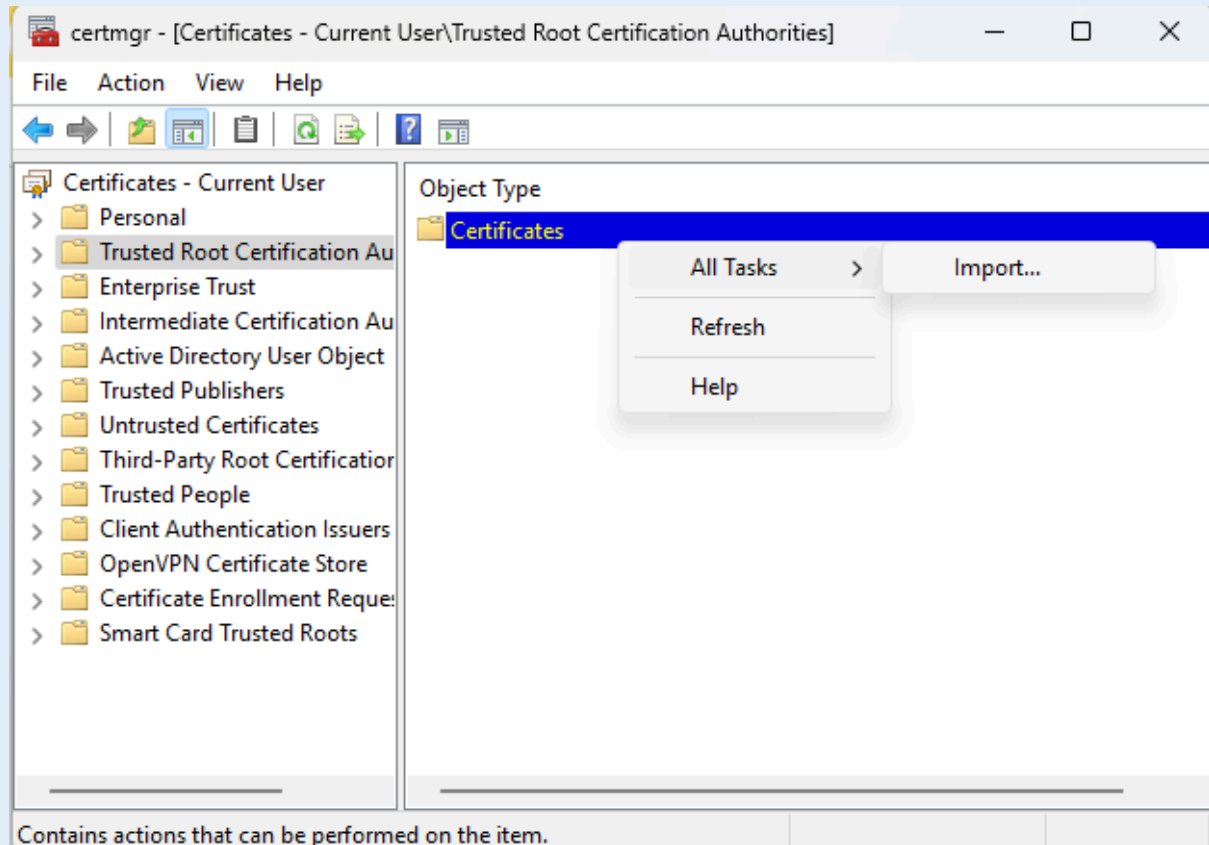
The screenshot shows a web browser window with the address bar displaying 'https://daws81s.online' and a 'Not secure' warning. The main content area shows the Red Hat logo and 'Red Hat Enterprise Linux Test'. A 'Certificate Viewer' sidebar is open on the right, showing details for the certificate issued to 'daws81s.online'. The 'Issued By' section is circled, showing it was issued by 'Linuscode Intermediate CA'.

Certificate Viewer: daws81s.online	
General Details	
Issued To	
Common Name (CN)	daws81s.online
Organization (O)	Linuscode Technologies Private Limited
Organizational Unit (OU)	IT Department
Issued By	
Common Name (CN)	Linuscode Intermediate CA
Organization (O)	Linuscode Technologies Pte Ltd
Organizational Unit (OU)	IT Department
Validity Period	
Issued On	Saturday, November 2, 2024 at 11:47:26 AM
Expires On	Sunday, November 2, 2025 at 11:47:26 AM
SHA-256 Fingerprints	

it may still show as "Not Secure" because browsers generally trust only standard, widely recognized CAs. However, you can resolve this by adding your root certificate and intermediate certificate to the trusted store of your computer's operating system. This way, your browser will recognize the certificate as trusted, and the "Not Secure" warning will disappear.

Copy the root certificate and intermediate certificate into your laptop.

Now let's import the certificates into our laptop. Press **Windows+R** and enter **certmgr.msc**



Import the root certificate here.

Welcome to the Certificate Import Wizard

This wizard helps you copy certificates, certificate trust lists, and certificate revocation lists from your disk to a certificate store.

A certificate, which is issued by a certification authority, is a confirmation of your identity and contains information used to protect data or to establish secure network connections. A certificate store is the system area where certificates are kept.

Store Location


☒ Current User

☐ Local Machine

To continue, click Next.

Next

Cancel

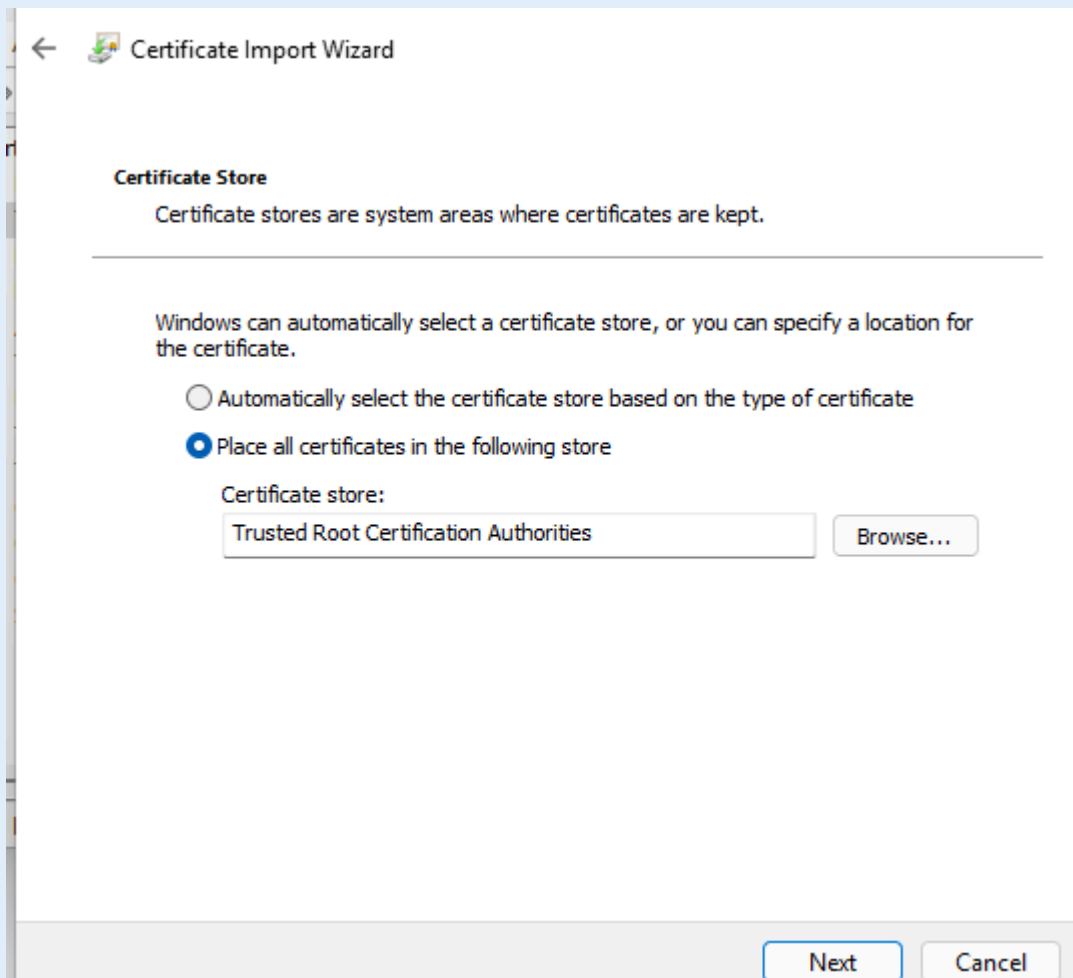
←  Certificate Import Wizard

File to Import
Specify the file you want to import.

File name:

Note: More than one certificate can be stored in a single file in the following formats:

- Personal Information Exchange- PKCS #12 (.PFX,.P12)
- Cryptographic Message Syntax Standard- PKCS #7 Certificates (.P7B)
- Microsoft Serialized Certificate Store (.SST)



Completing the Certificate Import Wizard

The certificate will be imported after you click Finish.

You have specified the following settings:

Certificate Store Selected by User	Trusted Root Certification Authorities
Content	Certificate
File Name	C:\linuscode\rootCA\rootCA_linuscode.crt

Finish

Cancel

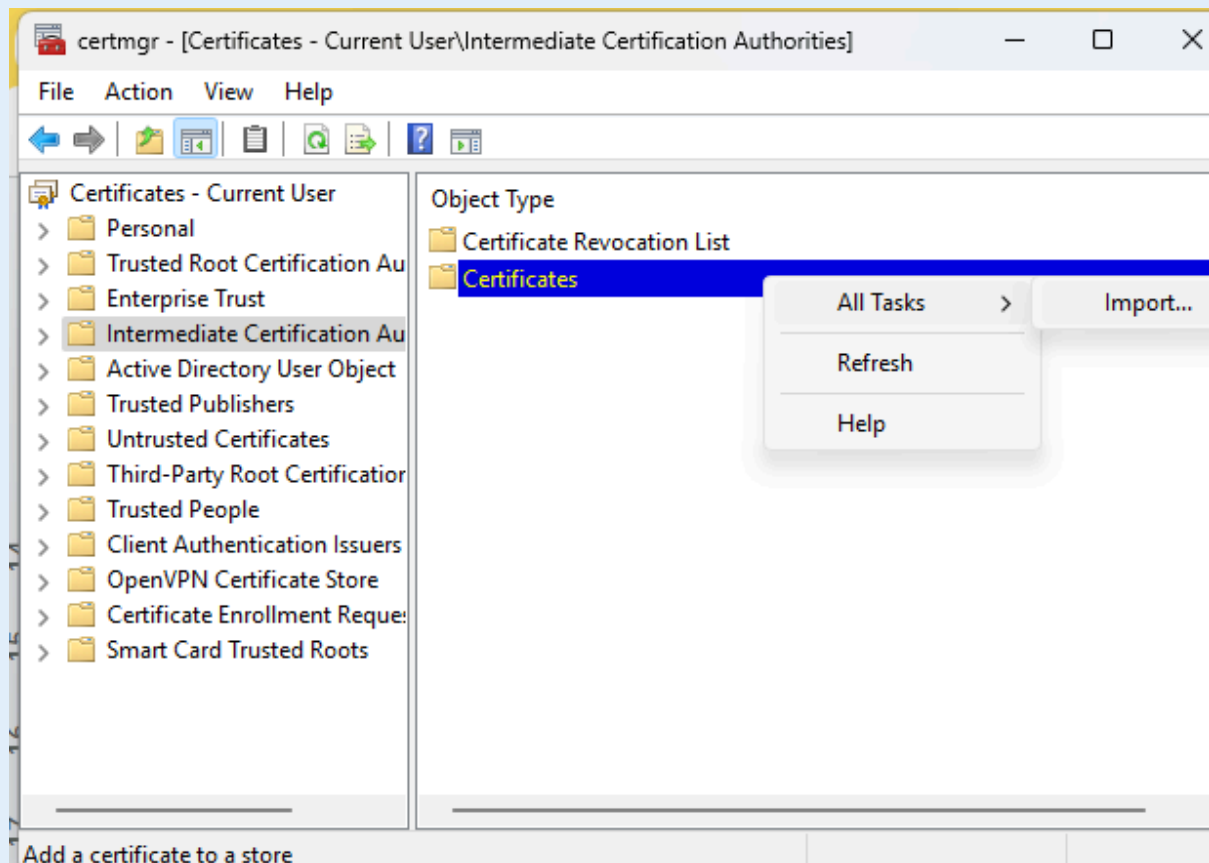
Certificate Import Wizard



The import was successful.

OK

Now let's install our intermediate certificate in similar way.



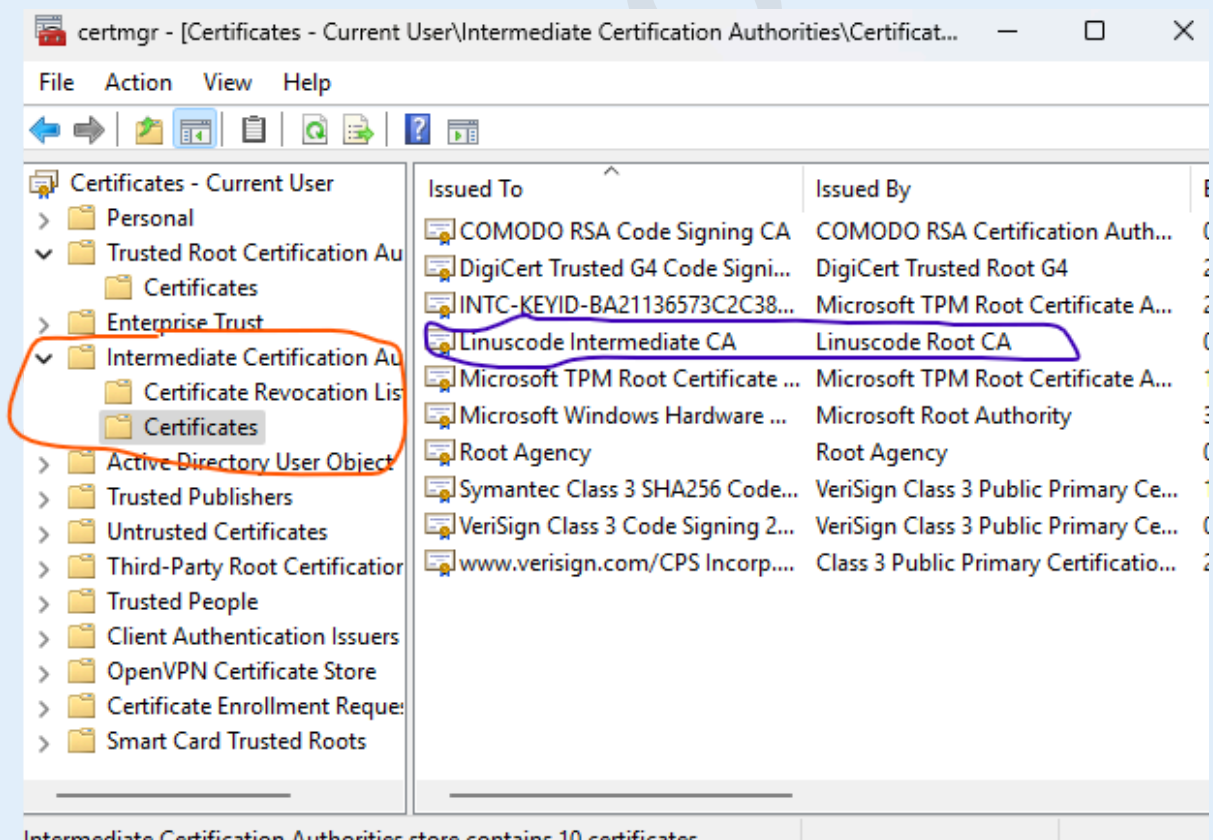
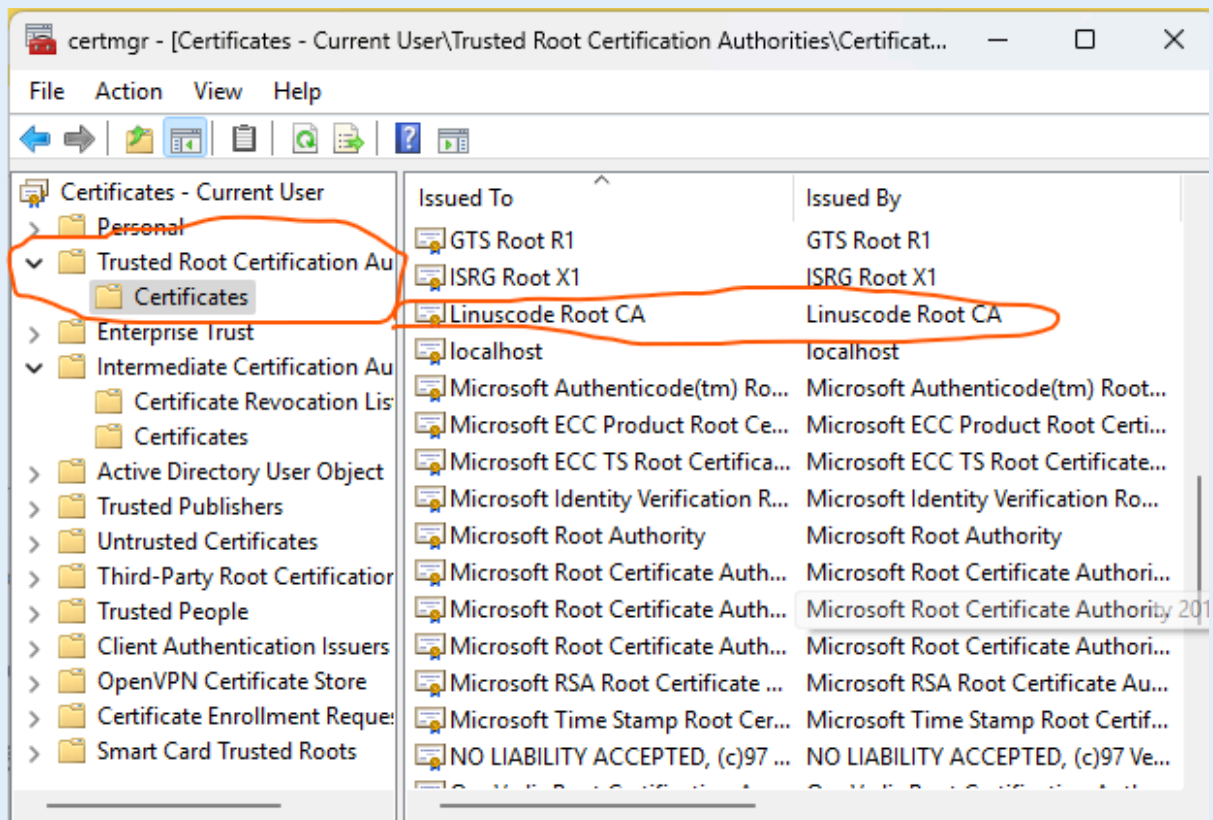
File to Import
Specify the file you want to import.

File name:

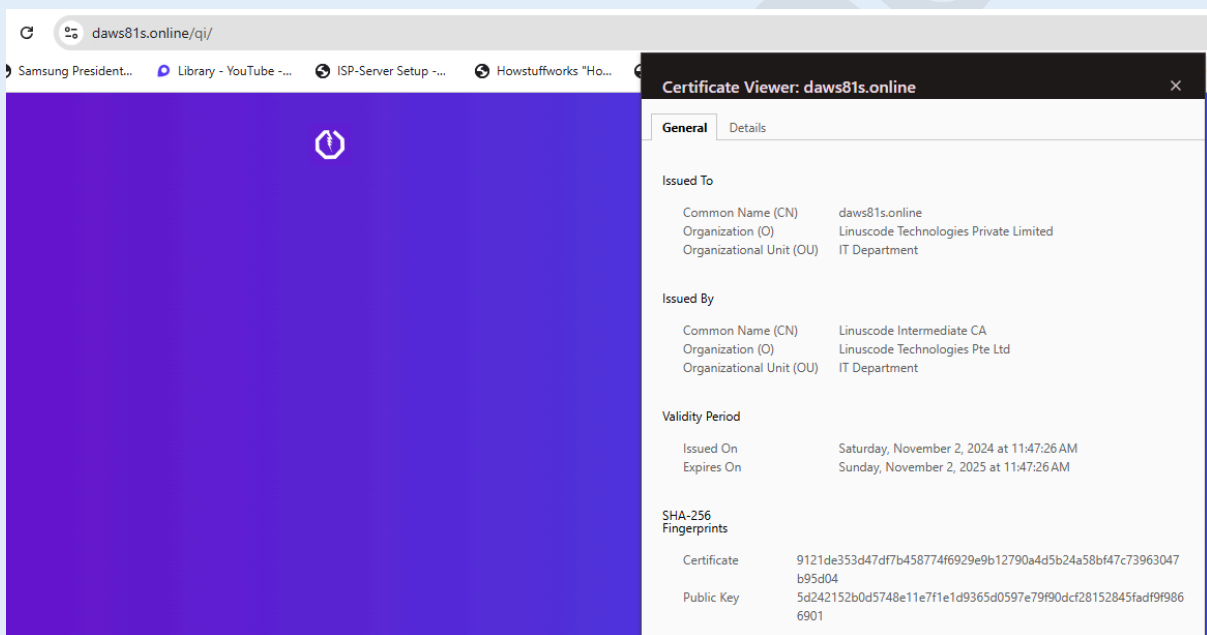
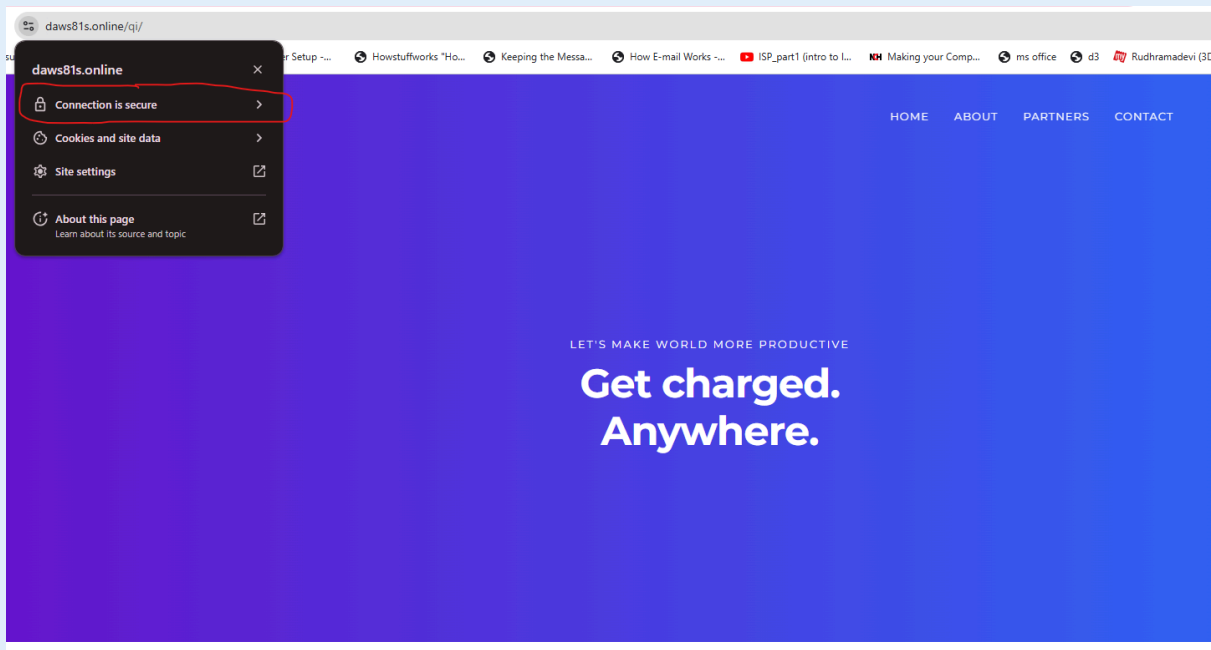
Note: More than one certificate can be stored in a single file in the following formats:

- Personal Information Exchange- PKCS #12 (.PFX,.P12)
- Cryptographic Message Syntax Standard- PKCS #7 Certificates (.P7B)
- Microsoft Serialized Certificate Store (.SST)

Make sure root certificate and intermediate certificate are available



Close the chrome browser and open your website again.



Now there are no warnings and our website is completely secure.

Companies use **configuration management tools** like **Ansible** to automate the installation of root and intermediate CA certificates across laptops and servers, ensuring consistent trust and security across all devices.

Conclusion:

- **Self-Signed Certificates for Development:** Using self-signed certificates is suitable for development (DEV) environments, providing security without the need for public trust.
- **Creating Internal Root and Intermediate CAs:** Establishing a company-specific Root CA and Intermediate CA allows for secure and controlled certificate management tailored to internal requirements.
- **Issuing Internal Certificates:** It's recommended to use internally issued certificates for internal applications, while public-facing applications should always utilise certificates from trusted external CAs for proper validation.
- **Understanding Certificate Authorities (CA):** We've successfully learned how Certificate Authorities (CAs) function and how to set up our own CA infrastructure to issue and manage digital certificates.