AUTOMATING

TLS in K8s

Part -1

How HTTPS works?

What is CA?

SSL configuration in nginx

Presented by

Sivakumar Reddy M

Founder of
Linuscode Technologies Pte Ltd
Singapore

LINUS CODE

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.

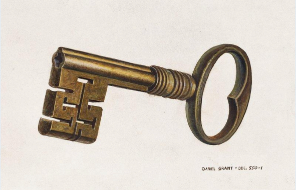
How HTTPS work?:

Before understanding how HTTPS works, it's essential to keep two key points in mind regarding cryptography, although you don't need to dive deep into the complex mathematical algorithms:

1. Any message encrypted with a public key can only be decrypted using the corresponding private key. For instance, if a message is encrypted with Trump's public key, only Trump's private key can unlock it. This principle ensures confidentiality in communication.
2. Anyone with access to Trump's public key can verify that a digital signature was created by the holder of the corresponding private key—essentially, only Trump. This process ensures authenticity and integrity, confirming that the message indeed comes from the claimed sender.

Let's have a simple example:

Think of a lock and key. The lock represents Trump's public key, which anyone can see and use. However, only Trump has the key to open it. Trump can give his lock(in an open state) to someone else, allowing them to use it to secure something inside. They can place their item in the locked box, lock it, and send it to Trump's address. Once it arrives, only Trump can use his key to unlock the box and access the contents inside.



What is CA?

CA stands for certificate authority. A Certificate Authority is an entity responsible for issuing digital certificates, which are used to verify the identity of organisations and individuals online. Here's a brief overview of what a CA does:

Functions of a Certificate Authority (CA):

1. **Issuing Certificates**: CAs issue digital certificates that authenticate the identity of a website, server, or individual. These certificates include the public key and information about the entity that owns the key.
2. **Verification**: Before issuing a certificate, a CA verifies the identity of the applicant. This process may involve confirming domain ownership. So we should have a domain to get the certificate.
3. **Certificate Management**: CAs manage the lifecycle of certificates, which includes issuing, renewing, and revoking them if they are no longer valid or if the entity's details change.
4. **Establishing Trust**: Browsers and operating systems come pre-installed with a list of trusted CAs. When a user visits a website with an SSL/TLS certificate issued by a trusted CA, the browser establishes a secure connection, ensuring that the site is legitimate.
5. **Maintaining Security Standards**: CAs are expected to adhere to strict security and operational guidelines to maintain the trust of users and organisations.

Popular Certificate Authorities (CAs) that are widely trusted by browsers include:

- Let's Encrypt
- DigiCert
- Comodo (now Sectigo)
- GlobalSign
- GoDaddy
- Thawte
- Entrust
- Symantec (now part of DigiCert)

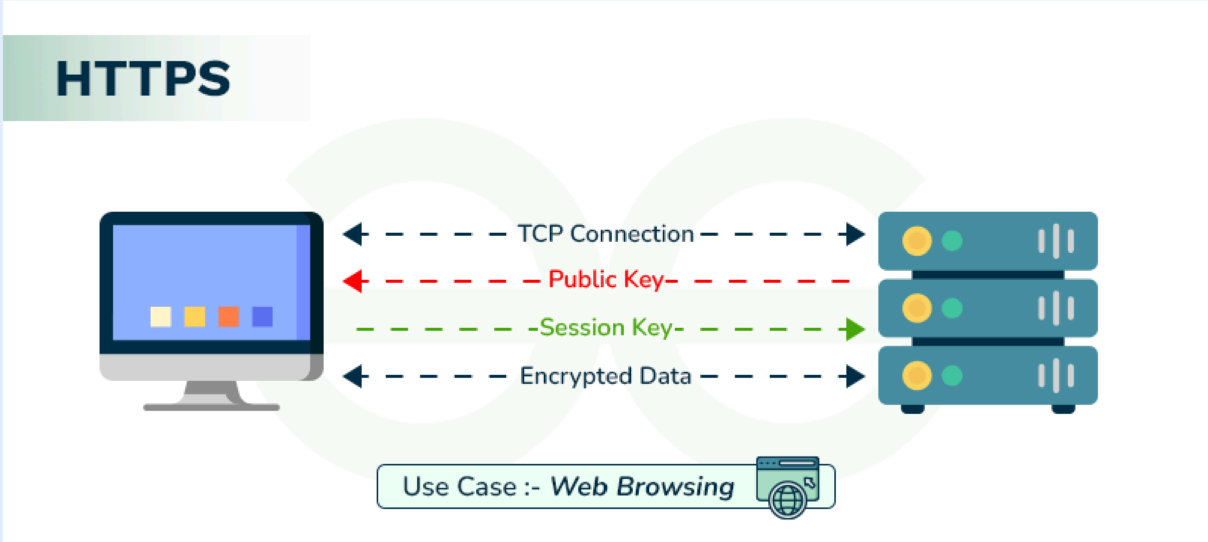
Browser Trust CAs:

Most web browsers come pre-installed with a list of trusted CAs, which are maintained in a **trust store of browser and Operating system**. Here's a brief overview of how this works:

- **Trust Store**: Browsers like Chrome, Firefox, Safari, and Edge maintain a list of trusted CAs. When a website presents its SSL certificate, the browser checks if the issuing CA is in this list.

- **Root Certificates**: The trusted CAs have root certificates, which serve as the basis for trust. If the browser recognizes the root certificate, it will trust the entire certificate chain issued by that CA.
- **Regular Updates**: Browsers regularly update their lists of trusted CAs, adding new ones and removing those that no longer meet security standards.

What happens when someone hits a website in the browser?



Client Request: When you enter a URL starting with "https://facebook.com" in your web browser, the browser initiates a connection to the server hosting the website.

Server Response: The server responds by sending its **digital certificate(issued by CA)** to the client (browser). This certificate includes the server's public key and below information

- Subject
- Public Key
- Issuer
- Serial Number
- Validity Period
- Signature Algorithm
- Extensions
- Digital Signature

Certificate Validation:

The browser checks the digital certificate against a list of trusted Certificate Authorities (CAs). If the certificate is valid and trusted, the connection can proceed. If not, the browser will display a warning to the user.

Session Key Creation:

- Once the certificate is validated, the client generates a unique session key (a symmetric key) for the session and encrypts it with the server's public key. **Remember here a message encrypted with a public key can only be decrypted by corresponding private key which is only with the server.**

  For example key can be K3f8t7G4vX2b9L1q

- This encrypted session key is then sent to the server.

Decryption:

The server uses its private key to decrypt the session key. Now both the client and server have a shared session key.

Secure Communication:

With the session key established, all subsequent data exchanged between the client and server is encrypted using this key. This ensures confidentiality and integrity, preventing eavesdropping or tampering by third parties.

Connection Closure: Once the communication is complete, the session key is discarded, and the connection is closed, ensuring that future sessions will require a new handshake and session key.

Let's host a website using HTTPS:

Even though there are automated ways of creating certificates, it's essential first to understand the manual process. For instance, we can use Certbot as an automated solution for obtaining SSL certificates from Let's Encrypt, which simplifies the process significantly. I can cover this automation later in the next part of Kubernetes.

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.

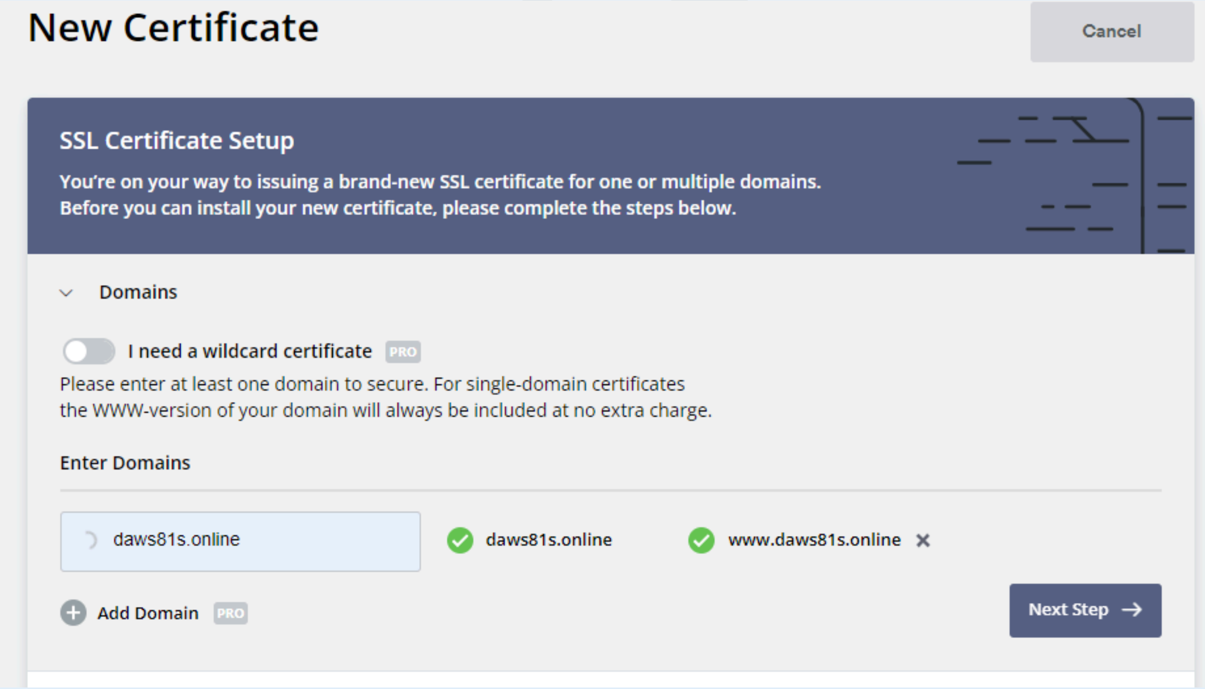
- CA. I chose CA as **sslforfree** https://www.sslforfree.com **you can get a free certificate for 90 days.**

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.
- We have 2 ways of generating certificates
  - Use the `openssl` command to generate a private key and a CSR (Certificate Signing Request). Upload the generated CSR to any Certificate Authority (CA) to obtain a certificate.
  - Alternatively, directly fill in the CSR details on the CA's website to receive all the required certificate files.

Let’s proceed with the second method, where we directly fill in the CSR on the CA’s website.

- Singup and login to sslforfree. Enter your domain name



New Certificate

SSL Certificate Setup

You're on your way to issuing a brand-new SSL certificate for one or multiple domains. Before you can install your new certificate, please complete the steps below.

Domains

I need a wildcard certificate PRO

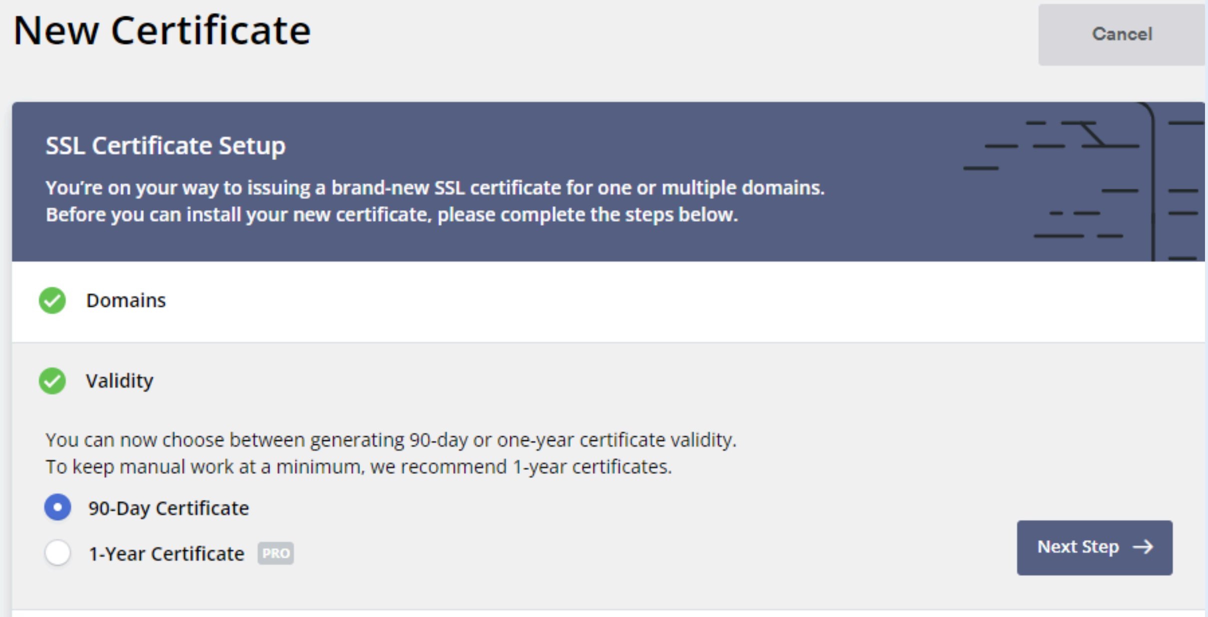
Please enter at least one domain to secure. For single-domain certificates the WWW-version of your domain will always be included at no extra charge.

Enter Domains

daws81s.online

✓ daws81s.online ✓ www.daws81s.online ✕

- Choose validity as 90 days.



New Certificate

SSL Certificate Setup

You're on your way to issuing a brand-new SSL certificate for one or multiple domains.
Before you can install your new certificate, please complete the steps below.

✓ Domains

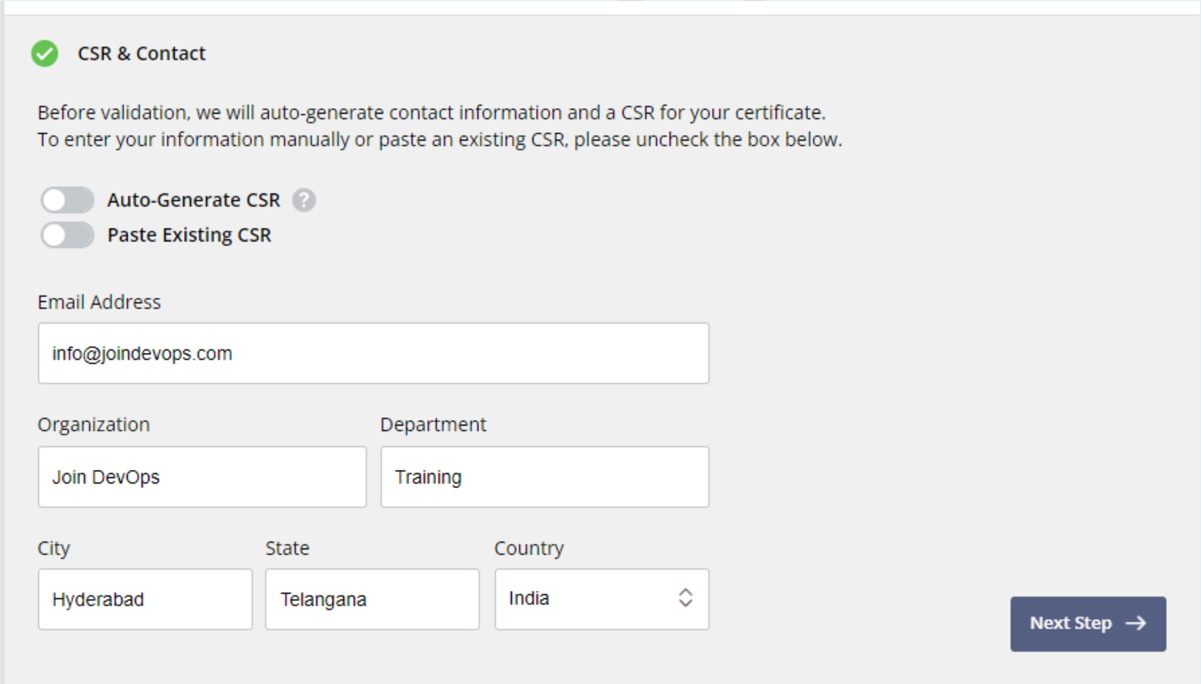
✓ Validity

You can now choose between generating 90-day or one-year certificate validity.
To keep manual work at a minimum, we recommend 1-year certificates.

● 90-Day Certificate

○ 1-Year Certificate PRO

- No need to select Add-Ons.
- Fill the CSR i.e certificate signing request



✓ CSR & Contact

Before validation, we will auto-generate contact information and a CSR for your certificate.
To enter your information manually or paste an existing CSR, please uncheck the box below.

○ Auto-Generate CSR ?

○ Paste Existing CSR

Email Address: info@joindevops.com

Organization: Join DevOps

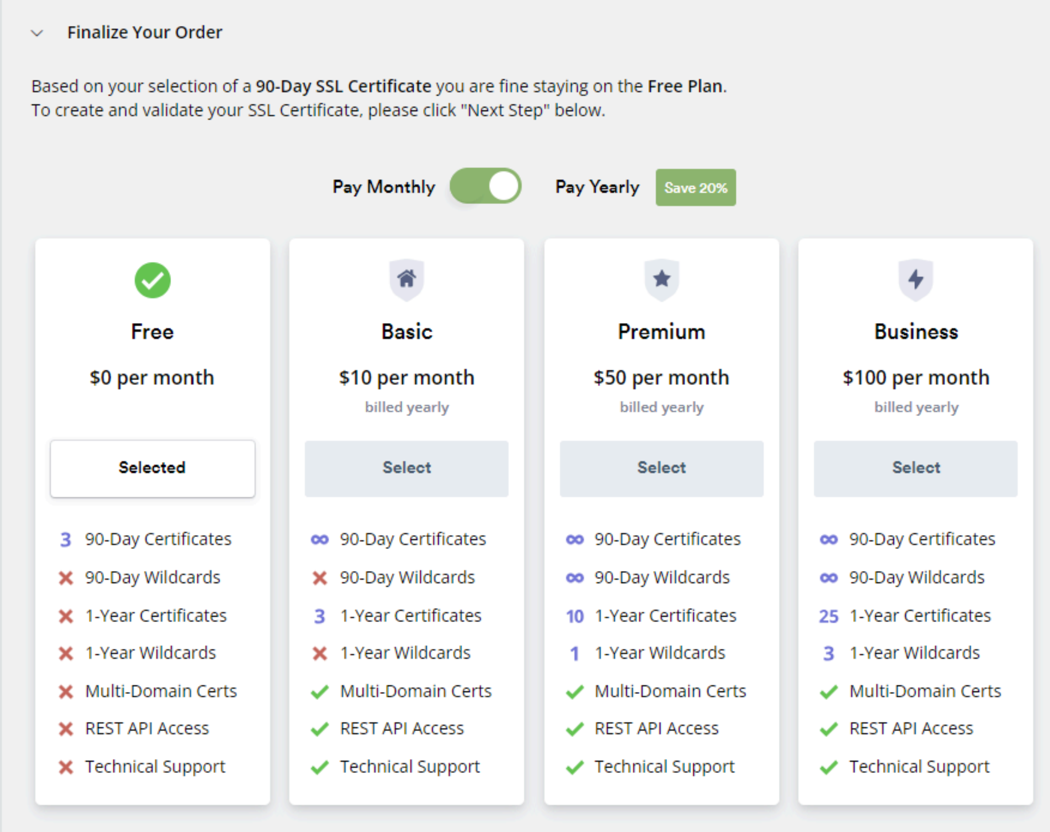
Department: Training

City: Hyderabad

State: Telangana

Country: India

- Finalise and select 90 days.



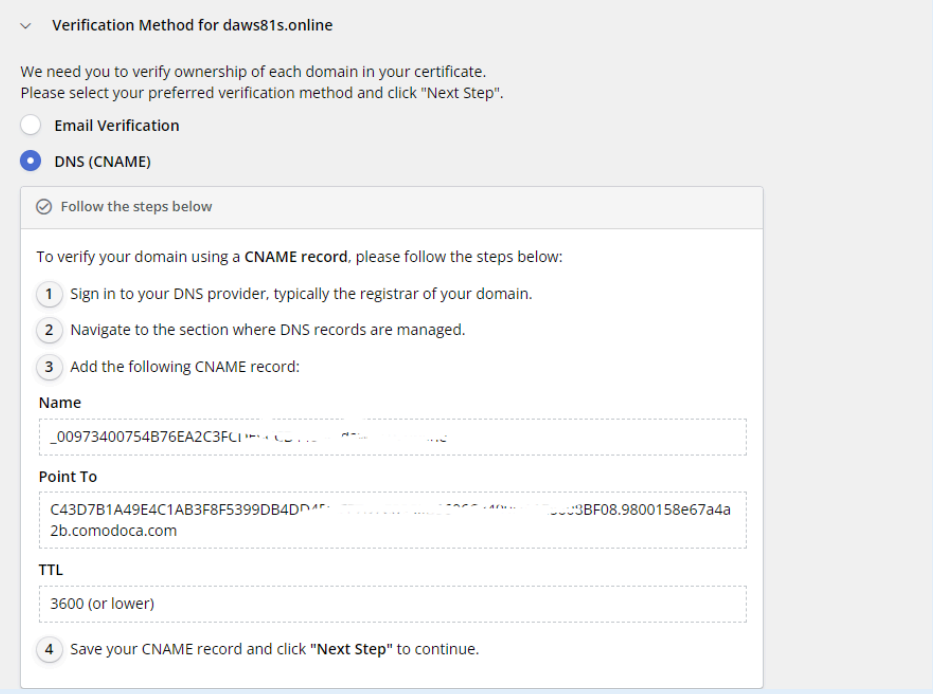
Finalize Your Order

Based on your selection of a **90-Day SSL Certificate** you are fine staying on the **Free Plan**.
To create and validate your SSL Certificate, please click "Next Step" below.

Pay Monthly ● Pay Yearly Save 20%

Free	Basic	Premium	Business
$0 per month	$10 per month billed yearly	$50 per month billed yearly	$100 per month billed yearly
Selected	Select	Select	Select
3 90-Day Certificates	∞ 90-Day Certificates	∞ 90-Day Certificates	∞ 90-Day Certificates
✗ 90-Day Wildcards	✗ 90-Day Wildcards	∞ 90-Day Wildcards	∞ 90-Day Wildcards
✗ 1-Year Certificates	3 1-Year Certificates	10 1-Year Certificates	25 1-Year Certificates
✗ 1-Year Wildcards	✗ 1-Year Wildcards	1 1-Year Wildcards	3 1-Year Wildcards
✗ Multi-Domain Certs	✓ Multi-Domain Certs	✓ Multi-Domain Certs	✓ Multi-Domain Certs
✗ REST API Access	✓ REST API Access	✓ REST API Access	✓ REST API Access
✗ Technical Support	✓ Technical Support	✓ Technical Support	✓ Technical Support

- Choose a DNS method to validate your domain. Create the CNAME record in your Route53 hosted zone. Select TTL as 1. Once records are created you can click verify domain.



Verification Method for daws81s.online

We need you to verify ownership of each domain in your certificate.
Please select your preferred verification method and click "Next Step".

○ Email Verification

● DNS (CNAME)

Follow the steps below

To verify your domain using a **CNAME record**, please follow the steps below:

1. Sign in to your DNS provider, typically the registrar of your domain.
2. Navigate to the section where DNS records are managed.
3. Add the following CNAME record:

Name

_00973400754B76EA2C3FC[redacted]

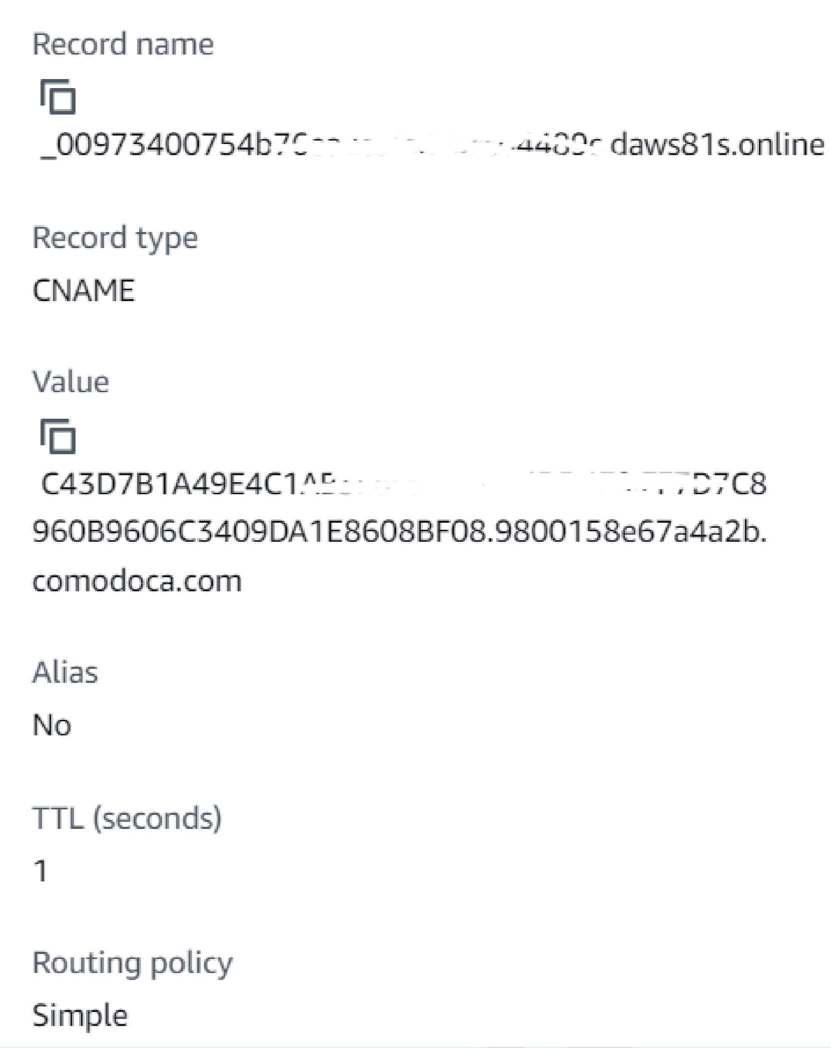
Point To

C43D7B1A49E4C1AB3F8F5399DB4D[redacted]3BF08.9800158e67a4a2b.comodoca.com

TTL

3600 (or lower)

4. Save your CNAME record and click **"Next Step"** to continue.



Record name

_00973400754b7[illegible]4480c daws81s.online

Record type

CNAME

Value

C43D7B1A49E4C1A[illegible]D7C8
960B9606C3409DA1E8608BF08.9800158e67a4a2b.
comodoca.com

Alias

No

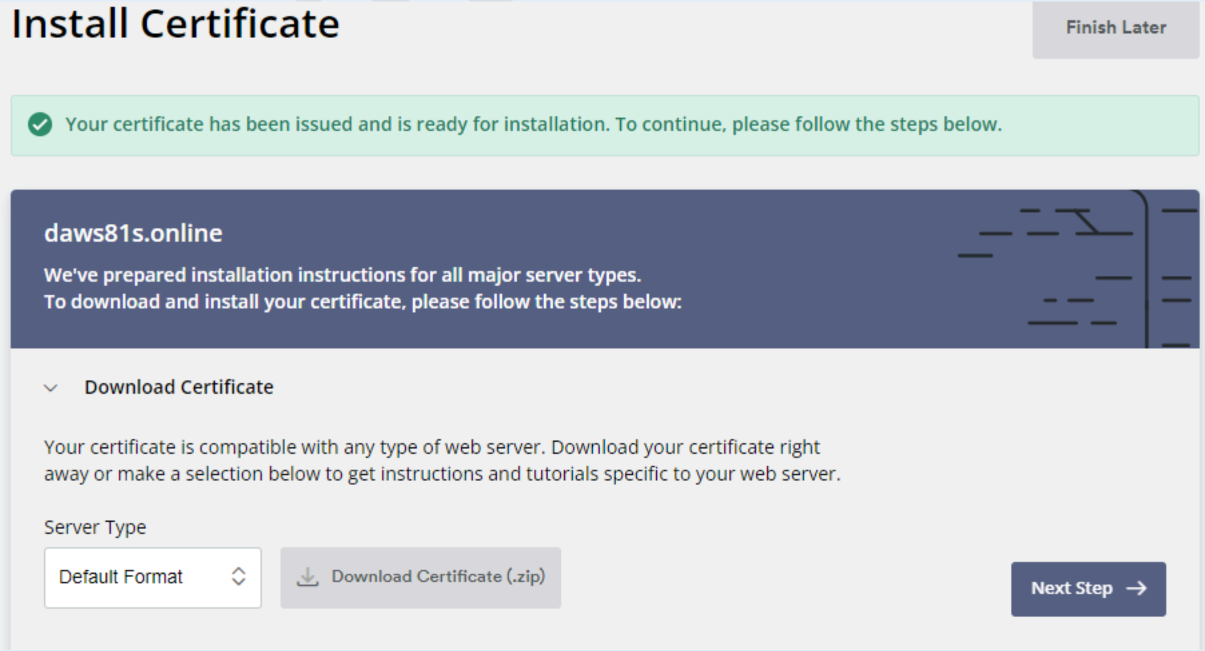
TTL (seconds)

1

Routing policy

Simple

- Select default format and download the certificate.



Install Certificate

Your certificate has been issued and is ready for installation. To continue, please follow the steps below.

daws81s.online

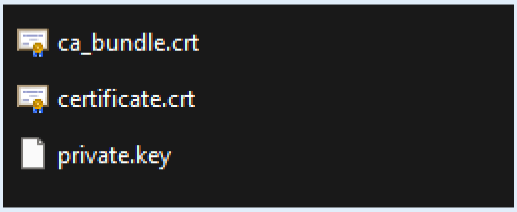
We've prepared installation instructions for all major server types.
To download and install your certificate, please follow the steps below:

Download Certificate

Your certificate is compatible with any type of web server. Download your certificate right away or make a selection below to get instructions and tutorials specific to your web server.

Server Type: Default Format

- **Congratulations you successfully got your SSL certificates**. Extract the zip file, you can see the content as below.



```
ca_bundle.crt
certificate.crt
private.key
```

- Let's understand the files.
    - **ca_bundle.crt:** is a file that contains a chain of certificates from a trusted Certificate Authority (CA). It is used to establish a chain of trust between your server's SSL certificate and the trusted root certificate of the CA. Root certificates are pre-installed in browsers.
    - **certificate.crt:** file, often referred to as an **SSL certificate** or **public key certificate**, is a digital file that certifies the ownership of a public key and is used to establish a secure connection between a web server and a client (such as a web browser)
    - **private.key:** is a crucial part of the public-private key pair used in asymmetric encryption for SSL/TLS certificates. It is a confidential key

SSL configuration in nginx:

Steps:

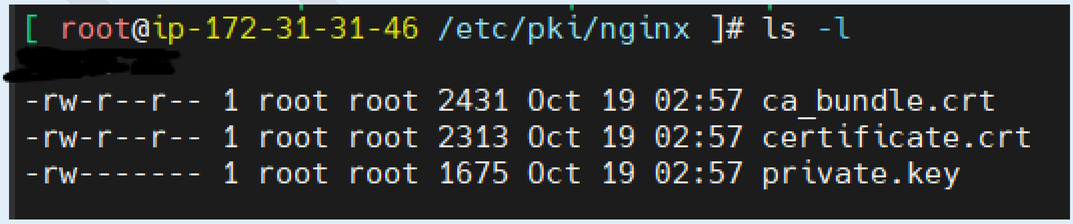
- Install nginx server in your EC2 instance.

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

- Create a folder /etc/pki/nginx

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

- Copy all the files downloaded from CA website.



```
[ root@ip-172-31-31-46 /etc/pki/nginx ]# ls -l
[redacted]
-rw-r--r-- 1 root root 2431 Oct 19 02:57 ca_bundle.crt
-rw-r--r-- 1 root root 2313 Oct 19 02:57 certificate.crt
-rw------- 1 root root 1675 Oct 19 02:57 private.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/ngx_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/certificate.crt";
       ssl_certificate_key "/etc/pki/nginx/private.key";
       ssl_trusted_certificate "/etc/pki/nginx/ca_bundle.crt";
       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.

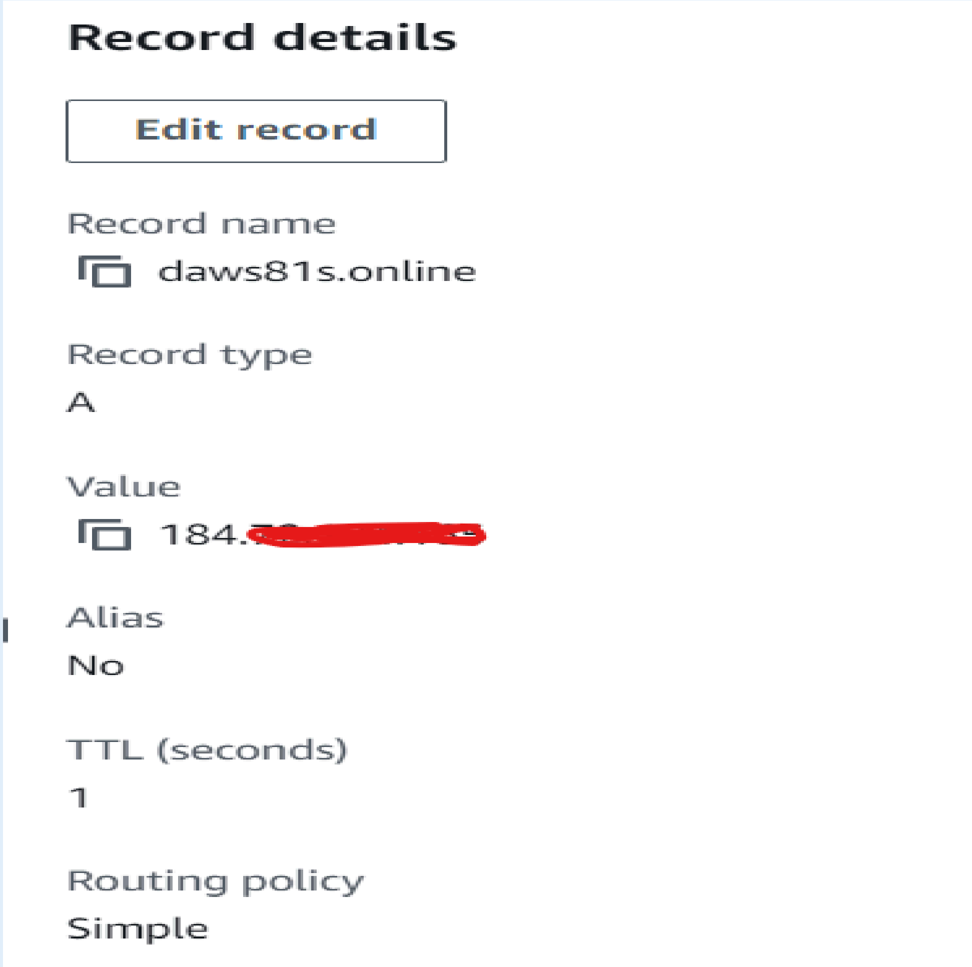
```
[ root@ip-172-31-31-46 /etc/nginx ]# nginx -t
```

```
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

Record name
daws81s.online

Record type
A

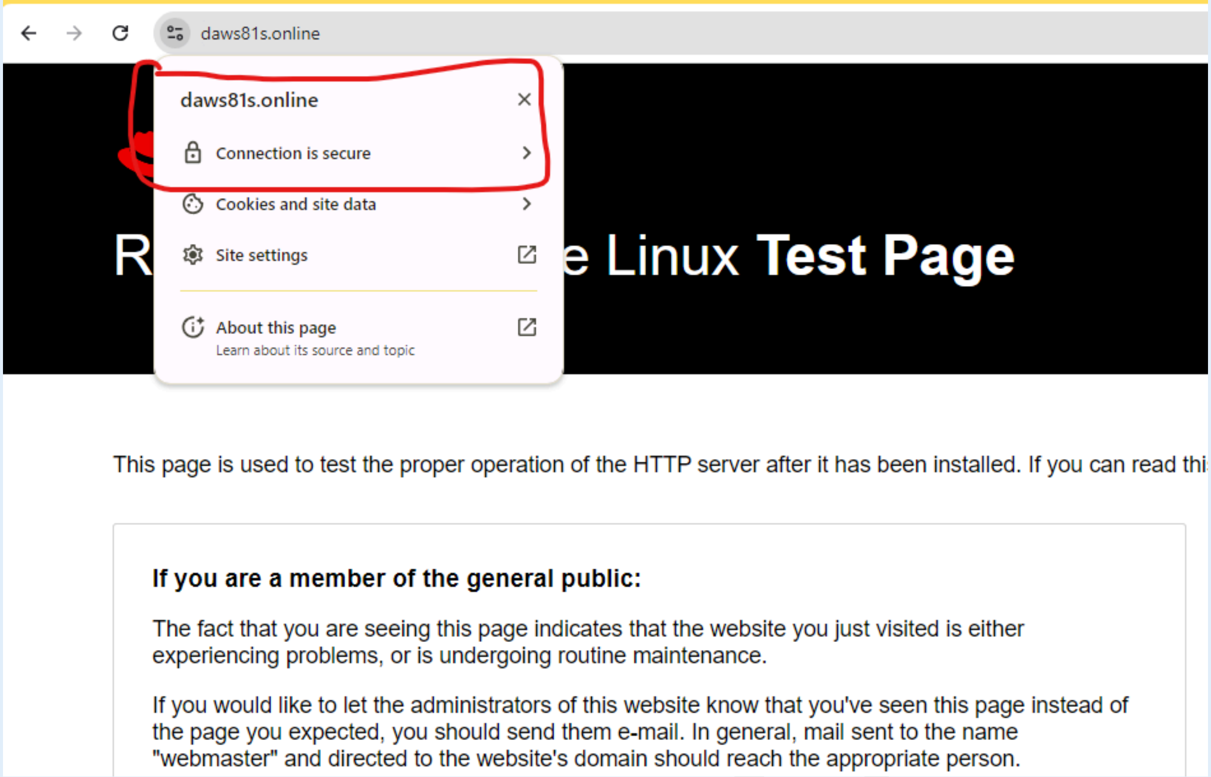
Value
184.[redacted]

Alias
No

TTL (seconds)
1

Routing policy
Simple

Congratulations, now your website is secure using https.



daws81s.online

R e Linux **Test Page**

This page is used to test the proper operation of the HTTP server after it has been installed. If you can read thi

If you are a member of the general public:

The fact that you are seeing this page indicates that the website you just visited is either experiencing problems, or is undergoing routine maintenance.

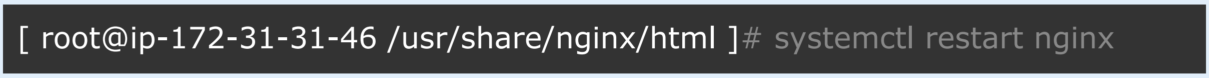
If you would like to let the administrators of this website know that you've seen this page instead of the page you expected, you should send them e-mail. In general, mail sent to the name "webmaster" and directed to the website's domain should reach the appropriate person.

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

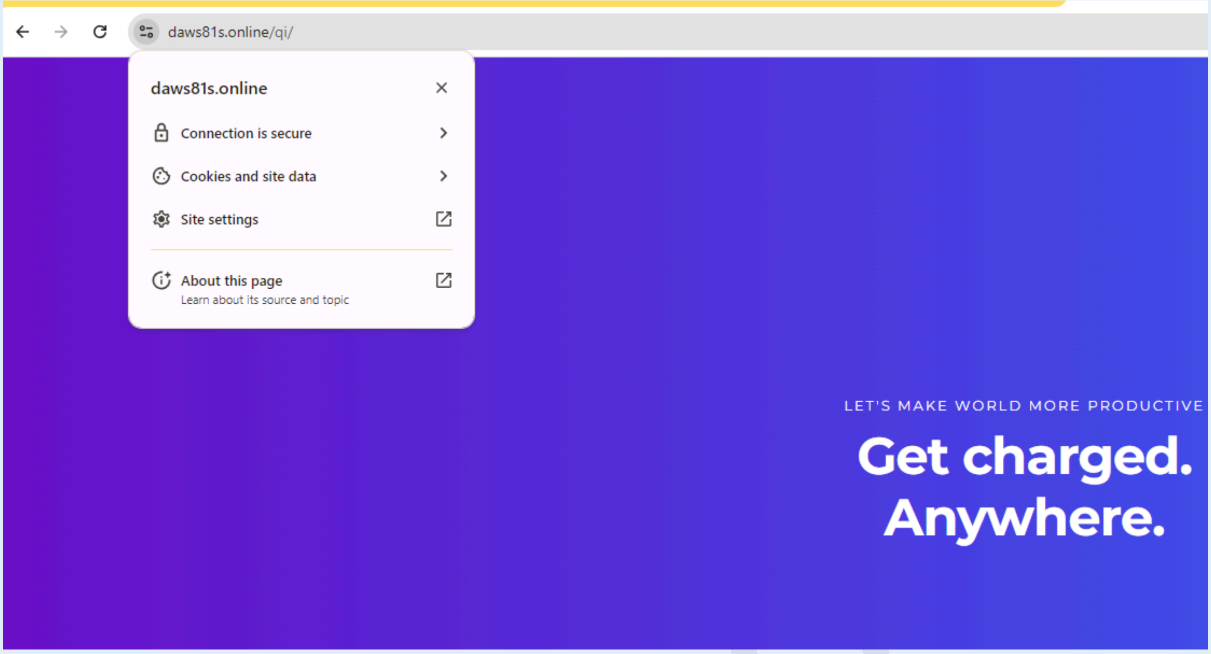
```
[ root@ip-172-31-31-46 ~ ]# cd /usr/share/nginx/html/

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
```



daws81s.online/qi/

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

Conclusion:

- If you have a valid domain with admin access, you can obtain a certificate and private key from any popular Certificate Authority (CA).
- Generate a Certificate Signing Request (CSR) and submit it to the CA, or fill in the CSR details directly on the CA's website.
- Once you receive the certificate files, configure them in your web server or application.
- This setup ensures that your website can securely communicate over HTTPS.