CONTENTS

// Tutorial //

How To Set Up and Configure a Certificate Authority (CA) On Ubuntu 20.04

Published on April 28, 2020

Security Ubuntu Ubuntu 20.04 VPN

By Jamon Camisso

English

Not using Ubuntu 20.04?
Choose a different version or distribution.

Ubuntu 20.04

Introduction

A Certificate Authority (CA) is an entity responsible for issuing digital certificates to verify identities on the internet. Although public CAs are a popular choice for verifying the identity of websites and other services that are provided to the general public, private CAs are typically used for closed groups and private services.

Building a private Certificate Authority will enable you to configure, test, and run programs that require encrypted connections between a client and a server. With a private CA, you can issue certificates for users, servers, or individual programs and services within your infrastructure.

Some examples of programs on Linux that use their own private CA are OpenVPN and Puppet . You can also configure your web server to use certificates issued by a private CA in order to make development and staging environments match production servers that use TLS to encrypt connections.

In this guide, we'll learn how to set up a private Certificate Authority on an Ubuntu 20.04 server, and how to generate and sign a testing certificate using your new CA. You will also learn how to import the CA server's public certificate into your operating system's certificate store so that you can verify the chain of trust between the CA and remote servers or users. Finally you will learn how to revoke certificates and distribute a Certificate Revocation List to make sure only authorized users and systems can use services that rely on your CA.

Prerequisites

To complete this tutorial, you will need access to an Ubuntu 20.04 server to host your CA server. You will need to configure a non-**root** user with `sudo` privileges before you start this guide. You can follow our Ubuntu 20.04 initial server setup guide to set up a user with appropriate permissions. The linked tutorial will also set up a **firewall**, which is assumed to be in place throughout this guide.

This server will be referred to as the **CA Server** in this tutorial.

Ensure that the CA Server is a standalone system. It will only be used to import, sign, and revoke certificate requests. It should not run any other services, and ideally it will be offline or completely shut down when you are not actively working with your CA.

> **Note:** The last section of this tutorial is optional if you would like to learn about signing and revoking certificates. If you choose to complete those practice steps, you will need a second Ubuntu 20.04 server or you can also use your own local Linux computer running Ubuntu or Debian, or distributions derived from either of those.

Step 1 – Installing Easy-RSA

The first task in this tutorial is to install the `easy-rsa` set of scripts on your CA Server. `easy-rsa` is a Certificate Authority management tool that you will use to generate a private key, and public root certificate, which you will then use to sign requests from clients and servers that will rely on your CA.

Login to your CA Server as the non-root sudo user that you created during the initial setup steps and run the following:

```
$ sudo apt update
$ sudo apt install easy-rsa
```

You will be prompted to download the package and install it. Press `y` to confirm you want to install the package.

At this point you have everything you need set up and ready to use Easy-RSA. In the next step you will create a Public Key Infrastructure, and then start building your Certificate Authority.

Step 2 – Preparing a Public Key Infrastructure Directory

Now that you have installed `easy-rsa`, it is time to create a skeleton Public Key Infrastructure (PKI) on the CA Server. Ensure that you are still logged in as your non-root user and create an `easy-rsa` directory. Make sure that you **do not use sudo** to run any of the following commands, since your normal user should manage and interact with the CA without elevated privileges.

```
$ mkdir ~/easy-rsa
```

This will create a new directory called `easy-rsa` in your home folder. We'll use this directory to create symbolic links pointing to the `easy-rsa` package files that we've installed in the previous step. These files are located in the `/usr/share/easy-rsa` folder on the CA Server.

Create the symlinks with the `ln` command:

```
$ ln -s /usr/share/easy-rsa/* ~/easy-rsa/
```

Note: While other guides might instruct you to copy the `easy-rsa` package files into your PKI directory, this tutorial adopts a symlink approach. As a result, any updates to the `easy-rsa` package will be automatically reflected in your PKI’s scripts.

To restrict access to your new PKI directory, ensure that only the owner can access it using the `chmod` command:

```
$ chmod 700 /home/sammy/easy-rsa
```

Finally, initialize the PKI inside the `easy-rsa` directory:

```
$ cd ~/easy-rsa
$ ./easyrsa init-pki
```

```
Output
init-pki complete; you may now create a CA or requests.
Your newly created PKI dir is: /home/sammy/easy-rsa/pki
```

After completing this section you have a directory that contains all the files that are needed to create a Certificate Authority. In the next section you will create the private key and public certificate for your CA.

Step 3 – Creating a Certificate Authority

Before you can create your CA’s private key and certificate, you need to create and populate a file called `vars` with some default values. First you will `cd` into the `easy-rsa` directory, then you will create and edit the `vars` file with `nano` or your preferred text editor:

```
$ cd ~/easy-rsa
$ nano vars
```

Once the file is opened, paste in the following lines and edit each highlighted value to reflect your own organization info. The important part here is to ensure that you do not leave any of the values blank:

~/easy-rsa/vars

```
set_var EASYRSA_REQ_COUNTRY    "US"
set_var EASYRSA_REQ_PROVINCE   "NewYork"
set_var EASYRSA_REQ_CITY       "New York City"
set_var EASYRSA_REQ_ORG        "DigitalOcean"
set_var EASYRSA_REQ_EMAIL      "admin@example.com"
set_var EASYRSA_REQ_OU         "Community"
set_var EASYRSA_ALGO           "ec"
set_var EASYRSA_DIGEST         "sha512"
```

When you are finished, save and close the file. If you are using `nano`, you can do so by pressing `CTRL+X`, then `Y` and `ENTER` to confirm. You are now ready to build your CA.

To create the root public and private key pair for your Certificate Authority, run the `./easy-rsa` command again, this time with the `build-ca` option:

```
$ ./easyrsa build-ca
```

Copy

In the output, you'll see some lines about the OpenSSL version and you will be prompted to enter a passphrase for your key pair. Be sure to choose a strong passphrase, and note it down somewhere safe. You will need to input the passphrase any time that you need to interact with your CA, for example to sign or revoke a certificate.

You will also be asked to confirm the Common Name (CN) for your CA. The CN is the name used to refer to this machine in the context of the Certificate Authority. You can enter any string of characters for the CA's Common Name but for simplicity's sake, press ENTER to accept the default name.

Output

```
. . .
Enter New CA Key Passphrase:
Re-Enter New CA Key Passphrase:
```

```
. . .
Common Name (eg: your user, host, or server name) [Easy-RSA CA]:

CA creation complete and you may now import and sign cert requests.
Your new CA certificate file for publishing is at:
/home/sammy/easy-rsa/pki/ca.crt
```

Note: If you don’t want to be prompted for a password every time you interact with your CA, you can run the `build-ca` command with the `nopass` option, like this:

```
$ ./easyrsa build-ca nopass
```

Copy

You now have two important files — `~/easy-rsa/pki/ca.crt` and `~/easy-rsa/pki/private/ca.key` — which make up the public and private components of a Certificate Authority.

- `ca.crt` is the CA’s public certificate file. Users, servers, and clients will use this certificate to verify that they are part of the same web of trust. Every user and server that uses your CA will need to have a copy of this file. All parties will rely on the public certificate to ensure that someone is not impersonating a system and performing a Man-in-the-middle attack.
- `ca.key` is the private key that the CA uses to sign certificates for servers and clients. If an attacker gains access to your CA and, in turn, your `ca.key` file, you will need to destroy your CA. This is why your `ca.key` file should **only** be on your CA machine and that, ideally, your CA machine should be kept offline when not signing certificate requests as an extra security measure.

With that, your CA is in place and it is ready to be used to sign certificate requests, and to revoke certificates.

Step 4 – Distributing your Certificate Authority’s Public Certificate

Now your CA is configured and ready to act as a root of trust for any systems that you want to configure to use it. You can add the CA’s certificate to your OpenVPN servers, web servers, mail servers, and so on.

Any user or server that needs to verify the identity of another user or server in your network should have a copy of the `ca.crt` file imported into their operating system's certificate store.

To import the CA's public certificate into a second Linux system like another server or a local computer, first obtain a copy of the `ca.crt` file from your CA server. You can use the `cat` command to output it in a terminal, and then copy and paste it into a file on the second computer that is importing the certificate. You can also use tools like `scp`, `rsync` to transfer the file between systems. However we'll use copy and paste with `nano` in this step since it will work on all systems.

As your non-root user on the CA Server, run the following command:

```
$ cat ~/easy-rsa/pki/ca.crt
```

There will be output in your terminal that is similar to the following:

```
Output
-----BEGIN CERTIFICATE-----
MIIDSzCCAjOgAwIBAgIUcR9Crsv3FBEujrPZnZnU4nSb5TMwDQYJKoZIhvcNAQEL
BQAwFjEUMBIGA1UEAwwLRWFzeS1SU0EgQ0EwHhcNMjAwMzE4MDMxNjI2WhcNMzAw
. . .
. . .
-----END CERTIFICATE-----
```

Copy everything, including the `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` lines and the dashes.

On your second Linux system use `nano` or your preferred text editor to open a file called `/tmp/ca.crt`:

```
$ nano /tmp/ca.crt
```

Paste the contents that you just copied from the CA Server into the editor. When you are finished, save and close the file. If you are using `nano`, you can do so by pressing `CTRL+X`, then `Y` and `ENTER` to confirm.

Now that you have a copy of the `ca.crt` file on your second Linux system, it is time to import the certificate into its operating system certificate store.

On Ubuntu and Debian based systems, run the following commands as your non-root user to import the certificate:

Ubuntu and Debian derived distributions

```
$ sudo cp /tmp/ca.crt /usr/local/share/ca-certificates/
$ sudo update-ca-certificates
```

Copy

To import the CA Server’s certificate on CentOS, Fedora, or RedHat based system, copy and paste the file contents onto the system just like in the previous example in a file called `/tmp/ca.crt`. Next, you’ll copy the certificate into `/etc/pki/ca-trust/source/anchors/`, then run the `update-ca-trust` command.

CentOS, Fedora, RedHat distributions

```
$ sudo cp /tmp/ca.crt /etc/pki/ca-trust/source/anchors/
$ sudo update-ca-trust
```

Copy

Now your second Linux system will trust any certificate that has been signed by the CA server.

> **Note:** If you are using your CA with web servers and use Firefox as a browser you will need to import the public `ca.crt` certificate into Firefox directly. Firefox does not use the local operating system’s certificate store. For details on how to add your CA’s certificate to Firefox please see this support article from Mozilla on Setting Up Certificate Authorities (CAs) in Firefox.
>
> If you are using your CA to integrate with a Windows environment or desktop computers, please see the documentation on how to use `certutil.exe` to install a CA certificate.

If you are using this tutorial as a prerequisite for another tutorial, or are familiar with how to sign and revoke certificates you can stop here. If you would like to learn more about how to sign and revoke certificates, then the following optional section will explain each process in detail.

(Optional) – Creating Certificate Signing Requests and Revoking Certificates

The following sections of the tutorial are optional. If you have completed all the previous steps then you have a fully configured and working Certificate Authority that you can use as a prerequisite for other tutorials. You can import your CA's `ca.crt` file and verify certificates in your network that have been signed by your CA.

If you would like to practice and learn more about how to sign certificate requests, and how to revoke certificates, then these optional sections will explain how both processes work.

(Optional) – Creating and Signing a Practice Certificate Request

Now that you have a CA ready to use, you can practice generating a private key and certificate request to get familiar with the signing and distribution process.

A Certificate Signing Request (CSR) consists of three parts: a public key, identifying information about the requesting system, and a signature of the request itself, which is created using the requesting party's private key. The private key will be kept secret, and will be used to encrypt information that anyone with the signed public certificate can then decrypt.

The following steps will be run on your second Ubuntu or Debian system, or distribution that is derived from either of those. It can be another remote server, or a local Linux machine like a laptop or a desktop computer. Since `easy-rsa` is not available by default on all systems, we'll use the `openssl` tool to create a practice private key and certificate.

`openssl` is usually installed by default on most Linux distributions, but just to be certain, run the following on your system:

```
$ sudo apt update
$ sudo apt install openssl
```

When you are prompted to install `openssl` enter `y` to continue with the installation steps. Now you are ready to create a practice CSR with `openssl`.

The first step that you need to complete to create a CSR is generating a private key. To create a private key using `openssl`, create a `practice-csr` directory and then generate a key inside it. We will make this request for a fictional server called `sammy-server`, as opposed to creating a certificate that is used to identify a user or another CA.

```
$ mkdir ~/practice-csr
$ cd ~/practice-csr
$ openssl genrsa -out sammy-server.key
```

Copy

```
Output
Generating RSA private key, 2048 bit long modulus (2 primes)
. . .
. . .
e is 65537 (0x010001)
```

Now that you have a private key you can create a corresponding CSR, again using the `openssl` utility. You will be prompted to fill out a number of fields like Country, State, and City. You can enter a `.` if you'd like to leave a field blank, but be aware that if this were a real CSR, it is best to use the correct values for your location and organization:

```
$ openssl req -new -key sammy-server.key -out sammy-server.req
```

Copy

```
Output
. . .
-----
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:New York
Locality Name (eg, city) [Default City]:New York City
Organization Name (eg, company) [Default Company Ltd]:DigitalOcean
Organizational Unit Name (eg, section) []:Community
Common Name (eg, your name or your server's hostname) []:sammy-server
Email Address []:

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

```
A challenge password []:
An optional company name []:
```

If you would like to automatically add those values as part of the `openssl` invocation instead of via the interactive prompt, you can pass the `-subj` argument to OpenSSL. Be sure to edit the highlighted values to match your practice location, organization, and server name:

```
$ openssl req -new -key sammy-server.key -out server.req -subj \
$ /C=US/ST=New\ York/L=New\ York\ City/O=DigitalOcean/OU=Community/CN=sammy-server
```

To verify the contents of a CSR, you can read in a request file with `openssl` and examine the fields inside:

```
$ openssl req -in sammy-server.req -noout -subject
```

```
Output
subject=C = US, ST = New York, L = New York City, O = DigitalOcean, OU = Community, CN = samm
```

Once you're happy with the subject of your practice certificate request, copy the `sammy-server.req` file to your CA server using `scp`:

```
$ scp sammy-server.req sammy@your_ca_server_ip:/tmp/sammy-server.req
```

In this step you generated a Certificate Signing Request for a fictional server called `sammy-server`. In a real-world scenario, the request could be from something like a staging or development web server that needs a TLS certificate for testing; or it could come from an OpenVPN server that is requesting a certificate so that users can connect to a VPN. In the next step, we'll proceed to signing the certificate signing request using the CA Server's private key.

(Optional) – Signing a CSR

In the previous step, you created a practice certificate request and key for a fictional server. You copied it to the `/tmp` directory on your CA server, emulating the process that you would use if you had real clients or servers sending you CSR requests that need to be signed.

Continuing with the fictional scenario, now the CA Server needs to import the practice certificate and sign it. Once a certificate request is validated by the CA and relayed back to a server, clients that trust the Certificate Authority will also be able to trust the newly issued certificate.

Since we will be operating inside the CA's PKI where the `easy-rsa` utility is available, the signing steps will use the `easy-rsa` utility to make things easier, as opposed to using the `openssl` directly like we did in the previous example.

The first step to sign the fictional CSR is to import the certificate request using the `easy-rsa` script:

```
$ cd ~/easy-rsa
$ ./easyrsa import-req /tmp/sammy-server.req sammy-server
```

Copy

Output

```
. . .
The request has been successfully imported with a short name of: sammy-server
You may now use this name to perform signing operations on this request.
```

Now you can sign the request by running the `easyrsa` script with the `sign-req` option, followed by the request type and the Common Name that is included in the CSR. The request type can either be one of `client`, `server`, or `ca`. Since we're practicing with a certificate for a fictional server, be sure to use the `server` request type:

```
$ ./easyrsa sign-req server sammy-server
```

Copy

In the output, you'll be asked to verify that the request comes from a trusted source. Type `yes` then press `ENTER` to confirm this:

Output

```
You are about to sign the following certificate.
Please check over the details shown below for accuracy. Note that this request
has not been cryptographically verified. Please be sure it came from a trusted
```

```
source or that you have verified the request checksum with the sender.

Request subject, to be signed as a server certificate for 3650 days:

subject=
    commonName                = sammy-server


Type the word 'yes' to continue, or any other input to abort.
  Confirm request details: yes
. . .
Certificate created at: /home/sammy/easy-rsa/pki/issued/sammy-server.crt
```

If you encrypted your CA key, you’ll be prompted for your password at this point.

With those steps complete, you have signed the `sammy-server.req` CSR using the CA Server’s private key in `/home/sammy/easy-rsa/pki/private/ca.key`. The resulting `sammy-server.crt` file contains the practice server’s public encryption key, as well as a new signature from the CA Server. The point of the signature is to tell anyone who trusts the CA that they can also trust the `sammy-server` certificate.

If this request was for a real server like a web server or VPN server, the last step on the CA Server would be to distribute the new `sammy-server.crt` and `ca.crt` files from the CA Server to the remote server that made the CSR request:

```
$ scp pki/issued/sammy-server.crt sammy@your_server_ip:/tmp
$ scp pki/ca.crt sammy@your_server_ip:/tmp
```

At this point, you would be able to use the issued certificate with something like a web server, a VPN, configuration management tool, database system, or for client authentication purposes.

(Optional) – Revoking a Certificate

Occasionally, you may need to revoke a certificate to prevent a user or server from using it. Perhaps someone’s laptop was stolen, a web server was compromised, or an employee or contractor has left your organization.

To revoke a certificate, the general process follows these steps:

1. Revoke the certificate with the `./easyrsa revoke client_name` command.
2. Generate a new CRL with the `./easyrsa gen-crl` command.
3. Transfer the updated `crl.pem` file to the server or servers that rely on your CA, and on those systems copy it to the required directory or directories for programs that refer to it.
4. Restart any services that use your CA and the CRL file.

You can use this process to revoke any certificates that you've previously issued at any time. We'll go over each step in detail in the following sections, starting with the `revoke` command.

Revoking a Certificate

To revoke a certificate, navigate to the `easy-rsa` directory on your CA server:

```
$ cd ~/easy-rsa
```

Copy

Next, run the `easyrsa` script with the `revoke` option, followed by the client name you wish to revoke. Following the practice example above, the Common Name of the certificate is `sammy-server`:

```
$ ./easyrsa revoke sammy-server
```

Copy

This will ask you to confirm the revocation by entering `yes`:

```
Output
Please confirm you wish to revoke the certificate with the following subject:

subject=
    commonName                = sammy-server


Type the word 'yes' to continue, or any other input to abort.
  Continue with revocation: yes
. . .
Revoking Certificate 8348B3F146A765581946040D5C4D590A
. . .
```

Note the highlighted value on the `Revoking Certificate` line. This value is the unique serial number of the certificate that is being revoked. If you want to examine the revocation list in the last step of this section to verify that the certificate is in it, you'll need this value.

After confirming the action, the CA will revoke the certificate. However, remote systems that rely on the CA have no way to check whether any certificates have been revoked. Users and servers will still be able to use the certificate until the CA's Certificate Revocation List (CRL) is distributed to all systems that rely on the CA.

In the next step you'll generate a CRL or update an existing `crl.pem` file.

Generating a Certificate Revocation List

Now that you have revoked a certificate, it is important to update the list of revoked certificates on your CA server. Once you have an updated revocation list you will be able to tell which users and systems have valid certificates in your CA.

To generate a CRL, run the `easy-rsa` command with the `gen-crl` option while still inside the `~/easy-rsa` directory:

```
$ ./easyrsa gen-crl
```

If you have used a passphrase when creating your `ca.key` file, you will be prompted to enter it. The `gen-crl` command will generate a file called `crl.pem`, containing the updated list of revoked certificates for that CA.

Next you'll need to transfer the updated `crl.pem` file to all servers and clients that rely on this CA each time you run the `gen-crl` command. Otherwise, clients and systems will still be able to access services and systems that use your CA, since those services need to know about the revoked status of the certificate.

Transferring a Certificate Revocation List

Now that you have generated a CRL on your CA server, you need to transfer it to remote systems that rely on your CA. To transfer this file to your servers, you can use the `scp` command.

Note: This tutorial explains how to generate and distribute a CRL manually. While there are more robust and automated methods to distribute and check revocation lists like OCSP-Stapling, configuring those methods is beyond the scope of this article.

Ensure you are logged into your CA server as your non-root user and run the following, substituting in your own server IP or DNS name in place of `your_server_ip`:

```
$ scp ~/easy-rsa/pki/crl.pem sammy@your_server_ip:/tmp
```

Copy

Now that the file is on the remote system, the last step is to update any services with the new copy of the revocation list.

Updating Services that Support a CRL

Listing the steps that you need to use to update services that use the `crl.pem` file is beyond the scope of this tutorial. In general you will need to copy the `crl.pem` file into the location that the service expects and then restart it using `systemctl`.

Once you have updated your services with the new `crl.pem` file, your services will be able to reject connections from clients or servers that are using a revoked certificate.

Examining and Verifying the Contents of a CRL

If you would like to examine a CRL file, for example to confirm a list of revoked certificates, use the following `openssl` command from within your `easy-rsa` directory on your CA server:

```
$ cd ~/easy-rsa
$ openssl crl -in pki/crl.pem -noout -text
```

Copy

You can also run this command on any server or system that has the `openssl` tool installed with a copy of the `crl.pem` file. For example, if you transferred the `crl.pem` file to your second system and want to verify that the `sammy-server` certificate is revoked, you can use an `openssl` command like the following, substituting the serial number that you noted earlier when you revoked the certificate in place of the highlighted one here:

```
$ openssl crl -in /tmp/crl.pem -noout -text |grep -A 1 8348B3F146A765581946040D5C4D59
```

```
Output
    Serial Number: 8348B3F146A765581946040D5C4D590A
        Revocation Date: Apr  1 20:48:02 2020 GMT
```

Notice how the `grep` command is used to check for the unique serial number that you noted in the revocation step. Now you can verify the contents of your Certificate Revocation List on any system that relies on it to restrict access to users and services.

Conclusion

In this tutorial you created a private Certificate Authority using the Easy-RSA package on a standalone Ubuntu 20.04 server. You learned how the trust model works between parties that rely on the CA. You also created and signed a Certificate Signing Request (CSR) for a practice server and then learned how to revoke a certificate. Finally, you learned how to generate and distribute a Certificate Revocation List (CRL) for any system that relies on your CA to ensure that users or servers that should not access services are prevented from doing so.

Now you can issue certificates for users and use them with services like OpenVPN. You can also use your CA to configure development and staging web servers with certificates to secure your non-production environments. Using a CA with TLS certificates during development can help ensure that your code and environments match your production environment as closely as possible.

If you would like to learn more about how to use OpenSSL, our OpenSSL Essentials: Working with SSL Certificates, Private Keys and CSRs tutorial has lots of additional information to help you become more familiar with OpenSSL fundamentals.

Learn more about us →

About the authors

Jamon Camisso Author

Still looking for an answer? Ask a question Search for more help

Was this helpful? Yes No

Comments

8 Comments

Leave a comment...

This textbox defaults to using `Markdown` to format your answer.

You can type `!ref` in this text area to quickly search our full set of tutorials, documentation & marketplace offerings and insert the link!

Sign In or Sign Up to Comment

nesman • April 17, 2023

Not sure if this documentation is current anymore. Following it exactly copying each command but when you get to the basic part of ./easyrsa init-pki it says the file is not found.

Reply

Harman Singh Sidhu • October 4, 2022

Hi

Thank you for the guide so far but i seem to get some errors i can’t resolve. Any insight is welcome

40E7D43E297F0000:error:0700006C:configuration file routines:NCONF_get_string:no value:…/crypto/conf/conf_lib.c:315:group=<NULL> name=unique_subject Error checking certificate extensions from extfile section default 40E7D43E297F0000:error:0700006C:configuration file routines:NCONF_get_string:no

```
value:.../crypto/conf/conf_lib.c:315:group=default name=extensions
40E7D43E297F0000:error:0700006C:configuration file routines:NCONF_get_string:no
value:.../crypto/conf/conf_lib.c:315:group=CA_default name=email_in_dn
40E7D43E297F0000:error:0700006C:configuration file routines:NCONF_get_string:no
value:.../crypto/conf/conf_lib.c:315:group=CA_default name=rand_serial
40E7D43E297F0000:error:04000067:object identifier routines:OBJ_txt2obj:unknown object
name:.../crypto/objects/obj_dat.c:376: 40E7D43E297F0000:error:1100006E:X509 V3
routines:v2i_EXTENDED_KEY_USAGE:invalid object
identifier:.../crypto/x509/v3_extku.c:95:Codesigning 40E7D43E297F0000:error:11000080:X509
V3 routines:X509V3_EXT_nconf_int:error in
extension:.../crypto/x509/v3_conf.c:48:section=default, name=extendedKeyUsage,
value=serverAuth, Codesigning
```

Easy-RSA error:

signing failed (openssl output above may have more detail)

Reply

niranga12 • December 20, 2021

I want to get both server and client authentication for the same certificate. Can anyone help me on that. Currently I only able to get either server or client authentication. TIA!

Reply

Elon 1505 • September 28, 2021

I am working to make ubuntu20 as CA server and after this i am importing CSR for Ubuntu20 to sign it as per instructions given in Step4. However I am seeing this error.

dk@Ubuntu:~/easy-rsa$ sudo ./easyrsa sign-req server NNHS

Note: using Easy-RSA configuration from: ./vars

Using SSL: openssl OpenSSL 1.1.1f 31 Mar 2020

You are about to sign the following certificate. Please check over the details shown below for accuracy. Note that this request 127.0.0.1 Ubuntu has not been cryptographically verified. Please be sure it came from a trusted source or that you have verified the request checksum with the sender.

Request subject, to be signed as a server certificate for 1080 days:

subject= commonName = M commonName = 10.0.0.13 organizationalUnitName = SLT 127.0.0.1 Ubuntu organizationName = J localityName = S stateOrProvinceName = C countryName = U

Type the word 'yes' to continue, or any other input to abort. Confirm request details: yes Using configuration from /home/dk/easy-rsa/pki/safessl-easyrsa.cnf Enter pass phrase for /home/dk/easy-rsa/pki/private/ca.key: ca: Error on line 17 of config file "/home/dk/easy-rsa/pki/extensions.temp" 139736827954496:error:0E079065:configuration file routines:def_load_bio:missing equal sign:…/crypto/conf/conf_def.c:391:line 17

Easy-RSA error:

signing failed (openssl output above may have more detail)

Can anyone help how to fix this please?

Reply

cajunjoel • September 12, 2021

The `~/easy-rsa/vars` file needs the following added to it, otherwise the values aren't used:

```
set_var EASYRSA_DN "org"
```

From the `vars.example` file:

```
# Define X509 DN mode.
# This is used to adjust what elements are included in the Subject field as the DN
# (this is the "Distinguished Name.")
# Note that in cn_only mode the Organizational fields further below aren't used.
#
# Choices are:
#   cn_only  - use just a CN value
#   org      - use the "traditional" Country/Province/City/Org/OU/email/CN format

# set_var EASYRSA_DN     "cn_only"
```

Other than that, this is a great tutorial and has been immensely helpful!

Reply

Zachary Zebrowski • June 30, 2021

Article is a life saver; thank you. (Having to deal with non-let's encrypt ssl is a pain.)

Reply

selimb • November 25, 2020

Very helpful article. Thanks!

There's one thing I don't quite understand though:

> If this request was for a real server like a web server or VPN server, the last step on the CA Server would be to distribute the new sammy-server.crt and

> ca.crt files from the CA Server to the remote server that made the CSR request"

In which case would I need to distribute `ca.crt` to the remote server? What do I do with it?

Show replies ⌄ Reply

swschulz • July 17, 2020

I am working through this and I noticed this in the `build-ca` step:

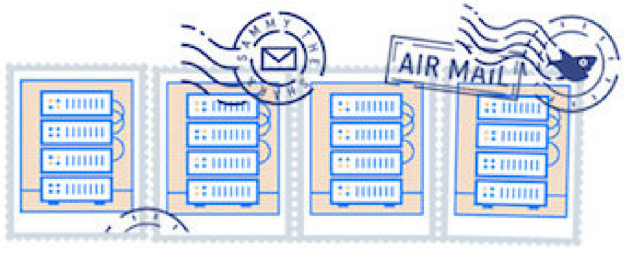
```
read EC key
writing EC key
Can't load /home/certs/easy-rsa/pki/.rnd into RNG
  140323442652480:error:2406F079:random number
  generator:RAND_load_file:Cannot open
  file:../crypto/rand/randfile.c:98:
  Filename=/home/certs/easyrsa/pki/.rnd
```

I do have a `pki/.rnd` file, but it concerns me that there may not be enough randomness if it truly cannot run the RNG.

This is on a brand new, patched 20.04 without other software installed. Were there other prerequisites?

Show replies ⌄ Reply

Talk to an expert →



Get our biweekly newsletter

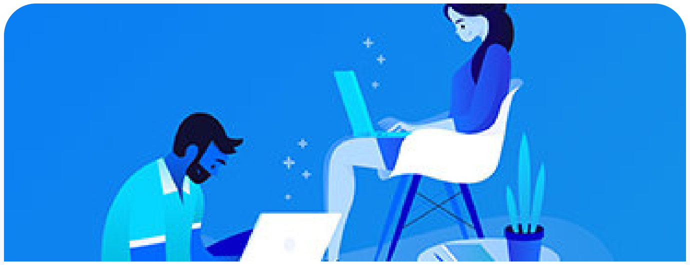
Sign up for Infrastructure as a Newsletter.

Sign up →

Hollie's Hub for Good

Working on improving health and education, reducing inequality, and spurring economic growth? We'd like to help.

Learn more →



Become a contributor

You get paid; we donate to tech nonprofits.

Learn more →

Featured on Community

Kubernetes Course   Learn Python 3   Machine Learning in Python   Getting started with Go   Intro to Kubernetes

DigitalOcean Products

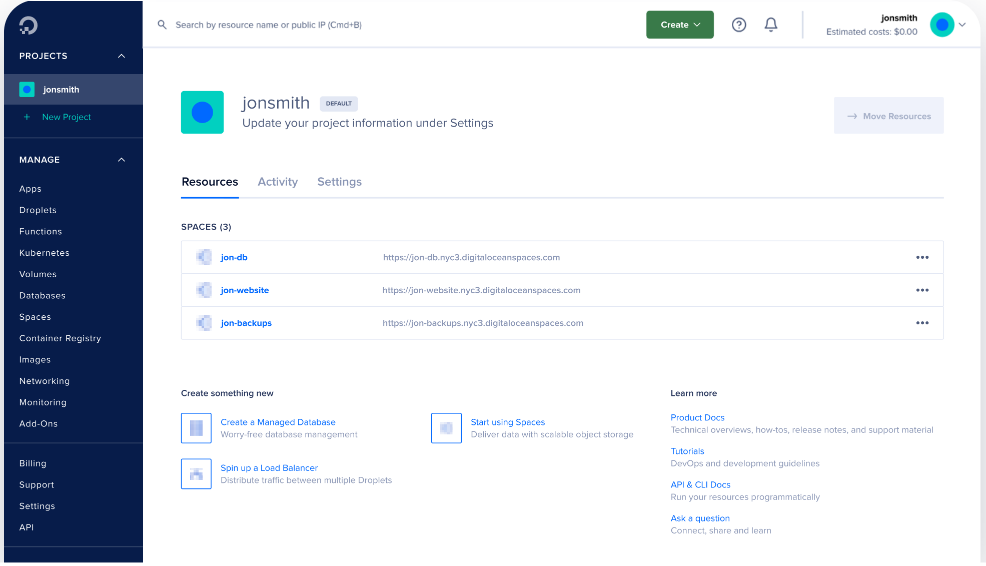
Cloudways   Virtual Machines   Managed Databases   Managed Kubernetes   Block Storage   Object Storage

Marketplace   VPC   Load Balancers

Welcome to the developer cloud

DigitalOcean makes it simple to launch in the cloud and scale up as you grow – whether you're running one virtual machine or ten thousand.

Learn more →



jonsmith

Update your project information under Settings

SPACES (3)

jon-db	https://jon-db.nyc3.digitaloceanspaces.com	•••
jon-website	https://jon-website.nyc3.digitaloceanspaces.com	•••
jon-backups	https://jon-backups.nyc3.digitaloceanspaces.com	•••

Create something new

Create a Managed Database
Worry-free database management

Start using Spaces
Deliver data with scalable object storage

Spin up a Load Balancer
Distribute traffic between multiple Droplets

Learn more

Product Docs
Technical overviews, how-tos, release notes, and support material

Tutorials
DevOps and development guidelines

API & CLI Docs
Run your resources programmatically

Ask a question
Connect, share and learn

Get started for free

Enter your email to get $200 in credit for your first 60 days with DigitalOcean.

Email address Send My Promo

New accounts only. By submitting your email you agree to our Privacy Policy.

Company

About

Leadership

Products

Products Overview

Droplets

Community

Tutorials

Q&A

Solutions

Website Hosting

VPS Hosting

Blog

Careers

Customers

Partners

Channel Partners

Referral Program

Affiliate Program

Press

Legal

Privacy Policy

Security

Investor Relations

DO Impact

Nonprofits

Kubernetes

Paperspace

App Platform

Functions

Cloudways

Managed Databases

Spaces

Marketplace

Load Balancers

Block Storage

Tools & Integrations

API

Pricing

Documentation

Release Notes

Uptime

CSS-Tricks

Write for DOnations

Currents Research

Hatch Startup Program

deploy by DigitalOcean

Shop Swag

Research Program

Open Source

Code of Conduct

Newsletter Signup

Meetups

Web & Mobile Apps

Game Development

Streaming

VPN

SaaS Platforms

Cloud Hosting for Blockchain

Startup Resources

Contact

Support

Sales

Report Abuse

System Status

Share your ideas