Vault as CA with PKI backend

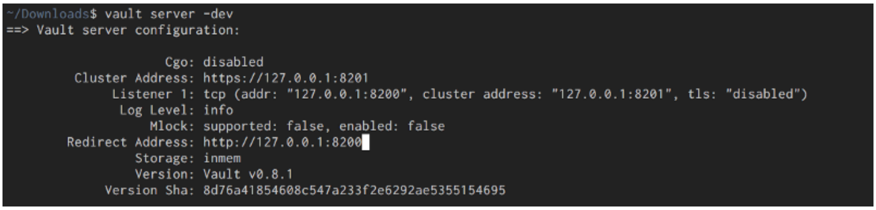
I'm gonna show how to run your own CA within pki framework , and be able to generate private keys and sign certificates. We will do this with **vault** , just because it's the fastest way to get it done.

Download and run Vault:

Make sure you get it from https://www.vaultproject.io/downloads.html or you build it by hand whatever you prefer.

We will run this in development mode for this tutorial but make sure you do something better if you're running this in production.

```
vault server -dev
```



```
~/Downloads$ vault server -dev
==> Vault server configuration:

                     Cgo: disabled
         Cluster Address: https://127.0.0.1:8201
              Listener 1: tcp (addr: "127.0.0.1:8200", cluster address: "127.0.0.1:8201", tls: "disabled")
               Log Level: info
                   Mlock: supported: false, enabled: false
        Redirect Address: http://127.0.0.1:8200
                 Storage: inmem
                 Version: Vault v0.8.1
             Version Sha: 8d76a41854608c547a233f2e6292ae5355154695
```

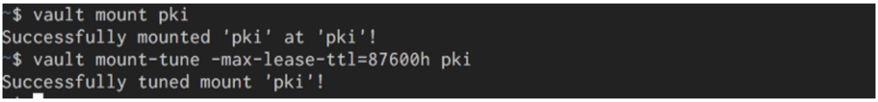
That simple , one thing to notice is that you will need to export and env variable to let vault know which vault server to use:

```
export VAULT_ADDR='http://127.0.0.1:8200'
```

Creating the PKI backend

Vault has plug-able backends , so we need to mount the backend we want to use , for that reason:

```
vault mount pki
vault mount-tune -max-lease-ttl=87600h pki
```

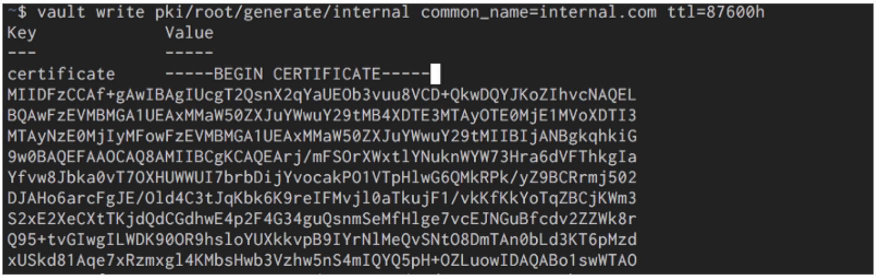


```
~$ vault mount pki
Successfully mounted 'pki' at 'pki'!
~$ vault mount-tune -max-lease-ttl=87600h pki
Successfully tuned mount 'pki'!
```

Generate the Root certificate

We will issue certs directly from the root , not using intermediates , so let’s create the root cert that the CA will use:

```
vault write pki/root/generate/internal
common_name=internal.com ttl=87600h
```

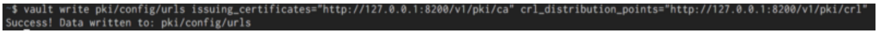


```
~$ vault write pki/root/generate/internal common_name=internal.com ttl=87600h
Key             Value
---             -----
certificate     -----BEGIN CERTIFICATE-----
MIIDFzCCAf+gAwIBAgIUcgT2QsnX2qYaUEOb3vuu8VCD+QkwDQYJKoZIhvcNAQEL
BQAwFzEVMBMGA1UEAxMMaW50ZXJuYWwuY29tMB4XDTE3MTAyOTE0MjE1MVoXDTI3
MTAyNzE0MjIyMFowFzEVMBMGA1UEAxMMaW50ZXJuYWwuY29tMIIBIjANBgkqhkiG
9w0BAQEFAAOCAQ8AMIIBCgKCAQEArj/mFSOrXWxtlYNuknWYW73Hra6dVFThkgIa
Yfvw8Jbka0vT7OXHUWWUI7brbDijYvocakPO1VTpHlwG6QMkRPk/yZ9BCRrmj502
DJAHo6arcFgJE/Old4C3tJqKbk6K9reIFMvjl0aTkujF1/vkKfKkYoTqZBCjKWm3
S2xE2XeCXtTKjdQdCGdhwE4p2F4G34guQsnmSeMfHlge7vcEJNGuBfcdv2ZZWk8r
Q95+tvGIwgILWDK90OR9hsloYUXkkvpB9IYrNlMeQvSNtO8DmTAn0bLd3KT6pMzd
xUSkd81Aqe7xRzmxgl4KMbsHwb3Vzhw5nS4mIQYQ5pH+OZLuowIDAQABo1swWTAO
```

Keys and certs will be store in the backend

Configure CRLs for the CA:

```
vault write pki/config/urls
issuing_certificates="http://127.0.0.1:8200/v1/pki/ca"
crl_distribution_points="http://127.0.0.1:8200/v1/pki/crl"
```

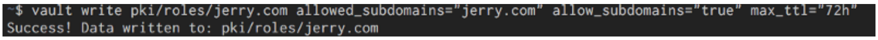


```
~$ vault write pki/config/urls issuing_certificates="http://127.0.0.1:8200/v1/pki/ca" crl_distribution_points="http://127.0.0.1:8200/v1/pki/crl"
Success! Data written to: pki/config/urls
```

Creating a Role

We gonna create a role like a policy that allows us to generate certs/keys or credentials :

```
vault write pki/roles/jerrycom allowed_domains="jerry.com"
allow_subdomains="true" max_ttl="72h"
```



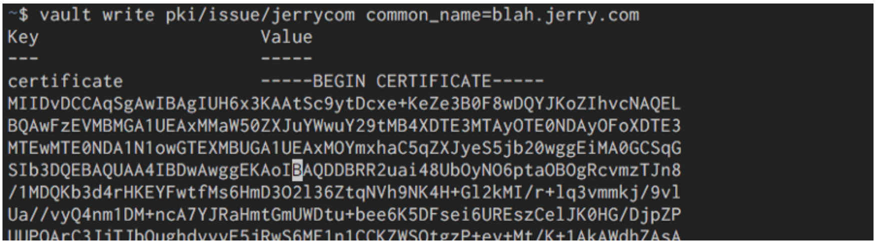
```
~$ vault write pki/roles/jerry.com allowed_subdomains="jerry.com" allow_subdomains="true" max_ttl="72h"
Success! Data written to: pki/roles/jerry.com
```

So the policy allows me to generate credentials for the domain jerry.com and it allows the creation of subdomains.

Issuing a crt and a private key

Finally we get to create the credentials that we will need to use in different services over tls

```
vault write pki/issue/jerrycom common_name=blah.jerry.com
```



```
~$ vault write pki/issue/jerrycom common_name=blah.jerry.com
Key                         Value
---                         -----
certificate                 -----BEGIN CERTIFICATE-----
MIIDvDCCAqSgAwIBAgIUH6x3KAAtSc9ytDcxe+KeZe3B0F8wDQYJKoZIhvcNAQEL
BQAwFzEVMBMGA1UEAxMMaW50ZXJuYWwuY29tMB4XDTE3MTAyOTE0NDAyOFoXDTE3
MTEwMTE0NDA1N1owGTEXMBUGA1UEAxMOYmxhaC5qZXJyeS5jb20wggEiMA0GCSqG
SIb3DQEBAQUAA4IBDwAwggEKAoIBAQDDBRR2uai48UbOyNO6ptaOBOgRcvmzTJn8
/1MDQKb3d4rHKEYFwtfMs6HmD3O2l36ZtqNVh9NK4H+Gl2kMI/r+lq3vmmkj/9vl
Ua//vyQ4nm1DM+ncA7YJRaHmtGmUWDtu+bee6K5DFsei6UREszCelJK0HG/DjpZP
```

From the command above you will get a key and a crt ,

```
-----BEGIN CERTIFICATE-----
```

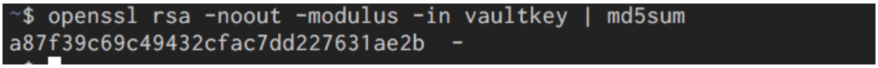
and

```
-----BEGIN RSA PRIVATE KEY-----
```

If you want to verify if these match , as the crt has been signed with the priv key , save the crt in a file and the key in a different file and run:

For the key:

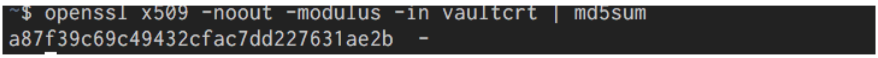
```
openssl rsa -noout -modulus -in vaultkey | md5sum
```



```
~$ openssl rsa -noout -modulus -in vaultkey | md5sum
a87f39c69c49432cfac7dd227631ae2b  -
```

For the CRT:

```
openssl x509 -noout -modulus -in vaultcrt | md5sum
```



```
~$ openssl x509 -noout -modulus -in vaultcrt | md5sum
a87f39c69c49432cfac7dd227631ae2b  -
```

You could also load them into nginx and test them with a browser for example .