CONTENTS

[**HELM INSTALLATION AND ADDING HASHICORP REPO** 3](#_Toc56507459)

[Install helm 3](#_Toc56507460)

[Add repo 3](#_Toc56507461)

[**VAULT INSTALLATION** 3](#_Toc56507462)

[**INITIALIZING AND UNSEALING THE VAULT SERVERS** 6](#_Toc56507463)

[**INGRESS DEPLOYMENT** 8](#_Toc56507464)

[**SECRET ENGINES** 10](#_Toc56507465)

[**ENABLE ANY SECRET ENGINE USING CURL** 10](#_Toc56507466)

[**K/V SECRET ENGINE** 11](#_Toc56507467)

[KV secret engine version1 12](#_Toc56507468)

[KV secret engine version-2 14](#_Toc56507469)

[**AWS SECRET ENGINE** 19](#_Toc56507470)

[Using CLI 19](#_Toc56507471)

[Using UI 21](#_Toc56507472)

[Using API 23](#_Toc56507473)

[**DATABASE SECRET ENGINE** 27](#_Toc56507474)

[POSTGRESQL 27](#_Toc56507475)

[MYSQL 30](#_Toc56507476)

[**TRANSIT SECRET ENGINE** 34](#_Toc56507477)

[**TOTP SECRET ENGINE** 36](#_Toc56507478)

[**SSH SECRET ENGINE** 38](#_Toc56507479)

[**PKI SECRET ENGINE** 45](#_Toc56507480)

[**ACCESS MANAGEMENT** 51](#_Toc56507481)

[Admin policy with root privileges 55](#_Toc56507482)

[Admin policy with restricted access to secrets 55](#_Toc56507483)

[Creating policy from UI 56](#_Toc56507484)

[Creating policy from CLI 57](#_Toc56507485)

[Creating policy using api 57](#_Toc56507486)

[**Creating Users, adding them to groups & assigning the policy to either group or user** 59](#_Toc56507487)

[Creating User from UI 59](#_Toc56507488)

[Creating user from API 61](#_Toc56507489)

[Creating user Using CLI 62](#_Toc56507490)

[Creating Group from UI 63](#_Toc56507491)

[Login using userpass auth with curl command 65](#_Toc56507492)

[Generating aws ec2role credentials using new client-token 66](#_Toc56507493)

[**RAFT BACK-UP AND RESTORE** 67](#_Toc56507494)

[**VAULT-GITLAB INTEGRATION** 68](#_Toc56507495)

[Enable jwt auth method 69](#_Toc56507496)

[Using cli 69](#_Toc56507497)

[Pipeline script to read dynamic AWS credentials and database secrets from vault 72](#_Toc56507498)

**DEPLOYING A HA MODE VAULT SERVER WITH RAFT AND USE INGRESS LOAD BALANCER**

# HELM INSTALLATION AND ADDING HASHICORP REPO

Install helm and add hashicorp repo .

## Install helm

apt-get update

apt-get updatecurl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 > get\_helm.sh

chmod 700 get\_helm.sh

./get\_helm.sh

## Add repo

The [Vault Helm chart](https://github.com/hashicorp/vault-helm) is the recommended way to install and configure Vault on Kubernetes. In addition to running Vault itself, the Helm chart is the primary method for installing and configuring Vault to integrate with other services such as Consul/Raft for High Availability (HA) deployments.

To use the Helm chart, add the Hashicorp helm repository and check that you have access to the chart.

helm repo add hashicorp https://helm.releases.hashicorp.com

"hashicorp" has been added to your repositories

# VAULT INSTALLATION

The below ***override-values.yaml*** file is providing a subset of values for attributes that are commonly overriden when deploying Vault to production on Kubernetes. Create the below override-values.yaml file and finally install the latest Vault Helm chart in HA Integrated Storage mode using these overrides.

Generally, the vault server by default will be of type *CLUSTERIP*,to enable *node port* for ingress we need to override the default configuration.

This can be done by creating an **override-values.yaml** file as follows:

**override-values.yaml**

server:

ui:

enabled: true

ha:

enabled: true

replicas: 3

raft:

enabled: true

service:

type: NodePort

port: 8200

targetPort: 8200

protocol: TCP

We will provide this file as argument during installation, so that the vault gets installed with the custom configuration over the default one.

**Vault Installation:**

If you want default configurations you can use the following command.

helm install vault hashicorp/vault \

--set='server.ha.enabled=true' \

--set='server.ha.raft.enabled=true'

To enable vault with custom configurations use the following command.

helm install vault hashicorp/vault -f override-values.yml

**output:**

WARNING: Kubernetes configuration file is group-readable. This is insecure. Location: /root/.kube/config

WARNING: Kubernetes configuration file is world-readable. This is insecure. Location: /root/.kube/config

NAME: vault

LAST DEPLOYED: Mon Oct 12 04:40:53 2020

NAMESPACE: default

STATUS: deployed

REVISION: 1

TEST SUITE: None

NOTES:

Thank you for installing HashiCorp Vault!

Now that you have deployed Vault, you should look over the docs on using

Vault with Kubernetes available here:

https://www.vaultproject.io/docs/

Your release is named vault. To learn more about the release, try:

$ helm status vault

$ helm get vault

See that the pods are deployed using the following command,

kubectl get pods

**output:**

NAME READY STATUS RESTARTS AGE

vault-0 0/1 Running 0 8m12s

vault-1 0/1 Running 0 8m12s

vault-2 0/1 Running 0 8m12s

vault-agent-injector-bdbf7b844-6q7rs 1/1 Running 0 8m12s

To check the services running on the kubernetes cluster , created and the node ports:

kubectl get svc

**output:**

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes ClusterIP 172.20.0.1 <none> 443/TCP 20d

vault NodePort 172.20.195.254 <none> 8200:30126/TCP,8201:31427/TCP 6m27s

vault-active NodePort 172.20.12.236 <none> 8200:32192/TCP,8201:31915/TCP 6m27s

vault-agent-injector-svc ClusterIP 172.20.79.220 <none> 443/TCP 6m27s

vault-internal ClusterIP None <none> 8200/TCP,8201/TCP 6m27s

vault-standby NodePort 172.20.17.203 <none> 8200:32041/TCP,8201:30895/TCP 6m27s

# INITIALIZING AND UNSEALING THE VAULT SERVERS

**For Vault-0 :**

kubectl exec -ti vault-0 -- vault operator init

**output:**

Unseal Key 1: NpNQY6wy5AzUPSKiz2ygdb7YkAOnNS7i6Mo86FIaWhC/

Unseal Key 2: UIrps2q4ab6mnh282LkusBd7Ht1CAd980Pq+EehyUlCr

Unseal Key 3: /DFU5Rc4uyVH9C/HcLYy3HzTife45dtHJvolAqzbR3c8

Unseal Key 4: Idut7+kJiXEMgVgTMLkVGixJ0v8lUv/C7OBrxyfZavti

Unseal Key 5: HeB2SHWozRWD4URoUQqcJ/mvAnA8lMioprFeXkz6z+sj

Initial Root Token: s.9NuiX2lkeyVB8YhfC7ct3GDS

Vault initialized with 5 key shares and a key threshold of 3. Please securely

distribute the key shares printed above. When the Vault is re-sealed,

restarted, or stopped, you must supply at least 3 of these keys to unseal it

before it can start servicing requests.

Vault does not store the generated master key. Without at least 3 key to

reconstruct the master key, Vault will remain permanently sealed!

It is possible to generate new unseal keys, provided you have a quorum of

existing unseal keys shares. See "vault operator rekey" for more information.

**Unseal vault-0:**

kubectl exec -ti vault-0 -- vault operator unseal

give this command and give the unseal key until it reaches it key threshold and until status for sealed becomes false as follows.

Key Value

--- -----

Seal Type shamir

Initialized true

Sealed false

Total Shares 5

Threshold 3

Version 1.5.2

Cluster Name vault-cluster-72ffbb12

Cluster ID bd393a8d-a285-4ba6-a926-baf11a5d9bd2

HA Enabled true

HA Cluster n/a

HA Mode standby

Active Node Address <none>

Raft Committed Index 24

Raft Applied Index 24

**For Vault-1 :**

First join the vault to the cluster:

kubectl exec -ti vault-1 -- vault operator raft join <http://vault-0.vault-internal:8200>

**output:**

Key Value

--- -----

Joined true

unseal vault as follows by using the same unseal keys of vault-0 until it reaches key threshold:

kubectl exec -ti vault-1 -- vault operator unseal

**For Vault-2 :**

First join the vault to the cluster:

kubectl exec -ti vault-2 -- vault operator raft join <http://vault-0.vault-internal:8200>

**output:**

Key Value

--- -----

Joined true

unseal vault as follows by using the same unseal keys of vault-0 until it reaches key threshold:

kubectl exec -ti vault-2 -- vault operator unseal

# INGRESS DEPLOYMENT

Deploy an Ingress resource for the vault server application.

Create a file vault-ingress.yaml :

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

name: "vault"

namespace: "default"

annotations:

kubernetes.io/ingress.class: alb

alb.ingress.kubernetes.io/scheme: internet-facing

alb.ingress.kubernetes.io/subnets: subnet-0fb67b0aecf8222a4,subnet-069743f33a9835f0b

labels:

app: vault-app

spec:

rules:

- http:

paths:

- path: /\*

backend:

serviceName: vault

servicePort: 8200

For vault path, give only “ /\* ” because vault will not support any other sub path given in the yaml file.

Apply the yaml file to create ingress,

kubectl apply -f vault-ingress.yaml

**output:**

ingress.extensions/vault created

We can now check your deployed ingress using following command:

kubectl get ingress

**output:**

NAME HOSTS ADDRESS PORTS AGE

vault \* 1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com 80 7s

Once we are done with the ingress deployment you will be able access the vault from the dns address of the ingress. You can play around the vault.

# SECRET ENGINES

Secrets engines are components which store, generate, or encrypt data. Secrets engines are incredibly flexible, so it is easiest to think about them in terms of their function. Secrets engines operate by receiving API calls that meet [this interface](https://github.com/hashicorp/vault/blob/061bc8ed2bbed49eb4f54c3a04647fbbefb13e48/sdk/framework/backend.go#L105). The calls receive data from the caller, take some action on that data, and they return a result.

Secrets engines are enabled at a "path" in Vault. When a request comes to Vault, the router automatically routes anything with the route prefix to the secrets engine. In this way, each secrets engine defines its own paths and properties. To the user, secret engines behave similar to a virtual filesystem, supporting operations like read, write, and delete.

## ENABLE ANY SECRET ENGINE USING CURL

**awssecretengine.json:**

{

"type":"aws"

}

**Command:**

curl \

--header "X-Vault-Token: s.IZeqoi10Pez65z1hEqHvbiWb" \

--request POST \

--data @awssecretengine.json \

<http://52.90.88.15:8200/v1/sys/mounts/awsbycurl>

## K/V SECRET ENGINE

The kv secrets engine is used to store arbitrary secrets within the configured physical storage for Vault.

Key names must always be strings. If you write non-string values directly via the CLI, they will be converted into strings. However, you can preserve non-string values by writing the key/value pairs to Vault from a JSON file or using the HTTP API.

A v2 kv secrets engine can be enabled by:

$ vault secrets enable -version=2 kv

**Write arbitrary data:**

$vault kv put kv/pavan-secret my-value=s3cr3t

Success! Data written to: kv/pavan-secret

$ vault kv get kv/pavan-secret

**Output:**

====== Metadata ======

Key Value

--- -----

created\_time 2020-10-13T05:44:06.405484191Z

deletion\_time n/a

destroyed false

version 1

====== Data ======

Key Value

--- -----

my-value s3cr3t

$ vault kv delete kv/pavan-secret

Success! Data deleted (if it existed) at: kv/pavan-secret

KV SECRET ENGINE VERSION1 **:**

When running the kv secrets backend non-versioned only the most recently written value for a key will be preserved. The benefits of non-versioned kv is a reduced storage size for each key since no additional metadata or history is stored.

vault secrets enable -version=1 kv

Enable kv secret engine with version 1.

Path : kv-v1/

**Path to write secret into kv version-1:**

**Adding user-1 :**

**vault write kv-v1/user1 username=lahari**

**Success! Data written to: kv-v1/user1**

**Adding user-2:**

**vault write kv-v1/user2 username=pavan**

**Success! Data written to: kv-v1/user2**

**Path to Read Secret for kv version-1:**

/kv-v1/pathForSecret

**Example :**

**vault read kv-v1/user1**

**output:**

**Key Value**

**username lahari**

**Curl command :**

curl \

--header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" \

<http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/kv-v1/user2>

**output:**

{"request\_id":"02d7042d-a3b2-7848-5808-4e2a58f8e2f8","lease\_id":"","renewable":false,"lease\_duration":2764800,"data":{"username":"pavan"},"wrap\_info":null,"warnings":null,"auth":null}

**Path to list secret for kv version-1 :**

**vault list kv-v1/**

**output:**

**Keys**

**user1**

**user2**

**Path to delete the secret for kv version-1 :**

**First adding a new user murali:**

**vault write kv-v1/user3 username=murali**

**Success! Data written to: kv-v1/user3**

**List the users you can see the user3:**

**vault list kv-v1/**

**output:**

**Keys**

**user1**

**user2**

**user3**

**delete the secret in path user3 :**

**vault delete kv-v1/user3**

**Success! Data deleted (if it existed) at: kv-v1/user3**

NOTE :

Even with a ttl set, the secrets engine never removes data on its own.

### KV SECRET ENGINE VERSION-2

**Enable version-2 kv**

$ vault secrets enable -version=2 kv22

vault secrets enable -path=kv kv

When running v2 of the kv backend a key can retain a configurable number of versions. This defaults to 10 versions. The older versions' metadata and data can be retrieved.

When a version is deleted the underlying data is not removed, rather it is marked as deleted. Deleted versions can be undeleted. To permanently remove a version's data the destroy command or API endpoint can be used. Additionally all versions and metadata for a key can be deleted by deleting on the metadata command or API endpoint. Each of these operations can be ACL'ed differently, restricting who has permissions to soft delete, undelete, or fully remove data.

**Path To Write Secrets for KV version-2:**

**kv.json:**

{

"data":{

"key1":"value1"

}

}

curl \

--header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" \

--request POST \

--data @kv.json \

http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/kv1/lahari/secrets/data/path1/demosecret

**output:**

{"request\_id":"81a83867-195e-300d-51da-4ef80701d0d6","lease\_id":"","renewable":false,"lease\_duration":0,"data":{"created\_time":"2020-10-15T07:56:30.624252247Z","deletion\_time":"","destroyed":false,"version":1},"wrap\_info":null,"warnings":null,"auth":null}

Perform the same curl command by changing the data in json a version 2 will be created for the same secret the output will be as follows.

{"request\_id":"ab9c80ec-7be1-cc37-39b8-543f4f9e9467","lease\_id":"","renewable":false,"lease\_duration":0,"data":{"created\_time":"2020-10-15T08:00:03.126344223Z","deletion\_time":"","destroyed":false,"version":2},"wrap\_info":null,"warnings":null,"auth":null}

**Path to Read Secret for kv version-2**

**(kv path)kv/data/mysecret(name of your secret)**

**vault read kv2/lahari/secrets/data/path1/secret1**

Key Value

--- -----

data map[secret1:secret1]

metadata map[created\_time:2020-10-15T05:35:33.894083841Z deletion\_time: destroyed:false version:1]

**curl command to read secrets:**

With the following command you will get the latest version of secret:

curl --header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" <http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/kv1/lahari/secrets/data/path1/secret1>

{"request\_id":"968bb060-da21-d6ff-fd20-8991a8f6642a","lease\_id":"","renewable":false,"lease\_duration":0,"data":{"data":{"secret123":"secret1"},"metadata":{"created\_time":"2020-10-15T05:35:33.894083841Z","deletion\_time":"","destroyed":false,"version":1}},"wrap\_info":null,"warnings":null,"auth":null}

We can also read secrets based on versions:

curl --header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/kv1/lahari/secrets/data/path1/secret1?version=1

{"request\_id":"d4c0af5d-06c2-818e-4906-f1e8d88460d3","lease\_id":"","renewable":false,"lease\_duration":0,"data":{"data":{"secret1":"secret1"},"metadata":{"created\_time":"2020-10-15T05:35:33.894083841Z","deletion\_time":"","destroyed":false,"version":1}},"wrap\_info":null,"warnings":null,"auth":null}

**Path to List Secret for kv version-2 :**

**vault list kv2/lahari/secrets/metadata/path1**

Keys

----

secret1

secret3

**Curl command to list secrets:**

curl --header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" --request LIST <http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/kv1/lahari/secrets/metadata/path1>

**output:**

{"request\_id":"f9f57dfd-469b-495b-59a7-d0fcdb695b6d","lease\_id":"","renewable":false,"lease\_duration":0,"data":{"keys":["secret1","secret3"]},"wrap\_info":null,"warnings":null,"auth":null}

**Delete and undelete the version:**

This endpoint issues a soft delete of the specified versions of the secret. This marks the versions as deleted and will stop them from being returned from reads, but the underlying data will not be removed. It can be also be undone.

**Create a json file :**

Deletekv.json – u can give multiple versions to delete

{

"versions": [1]

}

curl \

--header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" \

--request POST \

--data @deletekv.json \

<http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/kv1/lahari/secrets/delete/path1/demosecret>

**After delete if you try to retrieve the version you delete then you will get the following output:**

curl --header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" [http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/kv1/lahari/secrets/data/path1/demosecret?"version"=1](http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/kv1/lahari/secrets/data/path1/demosecret?%22version%22=1)

{"request\_id":"84847309-5e78-171e-c815-3c2bc5d7ae13","lease\_id":"","renewable":false,"lease\_duration":0,"data":{"data":null,"metadata":{"created\_time":"2020-10-15T07:56:30.624252247Z","deletion\_time":"2020-10-15T08:07:31.163811942Z","destroyed":false,"version":1}},"wrap\_info":null,"warnings":null,"auth":null}

**UNDELETE THE VERSION:**

curl --header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" --request POST --data @deletekv.json http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/kv1/lahari/secrets/unde

lete/path1/demosecret

read the deleted version now:

curl --header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" [http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/kv1/lahari/secrets/data/path1/demosecret?"version"=1](http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/kv1/lahari/secrets/data/path1/demosecret?%22version%22=1)

{"request\_id":"24b28899-33ca-385c-7711-c30fb6705e23","lease\_id":"","renewable":false,"lease\_duration":0,"data":{"data":{"key1":"value1"},"metadata":{"created\_time":"2020-10-15T07:56:30.624252247Z","deletion\_time":"","destroyed":false,"version":1}},"wrap\_info":null,"warnings":null,"auth":null}

**Destroy secret will delete the version permanently :**

**Path :**

[**http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/kv1/lahari/secrets/destroy/path1/demosecret**](http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/kv1/lahari/secrets/destroy/path1/demosecret)

## AWS SECRET ENGINE

The AWS secrets engine generates AWS access credentials dynamically based on IAM policies. This generally makes working with AWS IAM easier, since it does not involve clicking in the web UI. Additionally, the process is codified and mapped to internal auth methods. The AWS IAM credentials are time-based and are automatically revoked when the Vault lease expires.

### Using CLI

Enable AWS secret engine :

vault secrets enable -path=aws aws

Configure the credentials that Vault uses to communicate with AWS to generate the IAM credentials:

vault write aws/config/root \

access\_key=ACCESS\_KEY\_ID \

secret\_key=SECRET\_ACCESS\_KEY

Configure a Vault role that maps to a set of permissions in AWS as well as an AWS credential type. When users generate credentials, they are generated against this role.

We create two roles one for ec2 and other for s3 role access as follows:

**Ec2 role:**

vault write aws/roles/ec2role \

credential\_type=iam\_user \

policy\_document=-<<EOF

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": "ec2:\*",

"Resource": "\*"

}

]

}

EOF

**S3 role:**

vault write aws/roles/s3role \

credential\_type=iam\_user \

policy\_document=-<<EOF

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": "s3:\*",

"Resource": "\*"

}

]

} EOF

After the secrets engine is configured and a user/machine has a Vault token with the proper permission, it can generate credentials.

Generate a new credential by reading from the /creds endpoint with the name of the role:

vault read aws/creds/ec2role

**output :**

Key Value

--- -----

lease\_id aws/creds/ec2role/e75QUrah9HuVm3c5KUBJaKg9

lease\_duration 1800

lease\_renewable true

access\_key AKIA4OJTSXOYK5G5ODEH

secret\_key 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

security\_token <nil>

vault read aws/creds/s3role

**output :**

Key Value

--- -----

lease\_id aws/creds/s3role/e75QUrah9HuVm3c5KUBJaKg9

lease\_duration 1800

lease\_renewable true

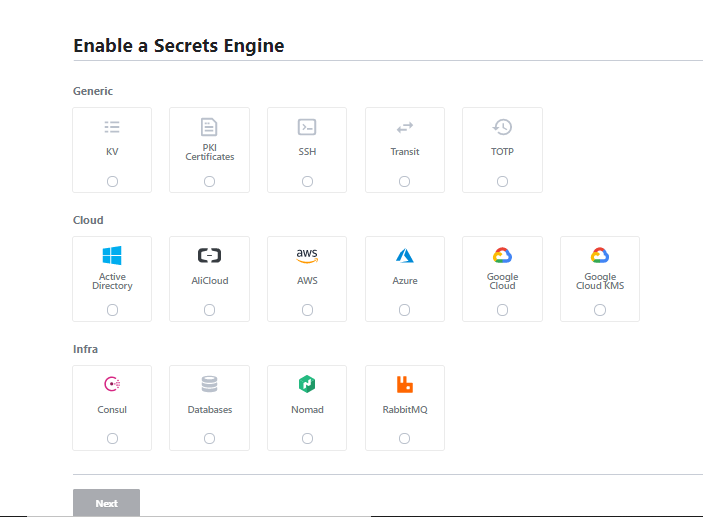
access\_key AKIA4OJTSXOYK5G5ODEH

secret\_key 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

security\_token <nil>

### Using UI

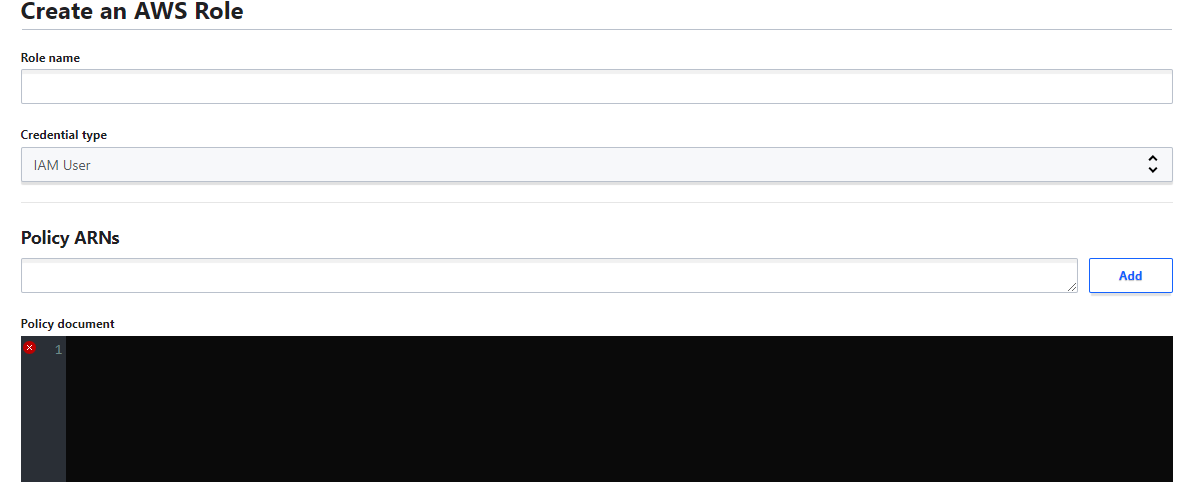
In UI, go to secrets and click on enable new engine, then you will be displayed with a list of available secret engines as follows:



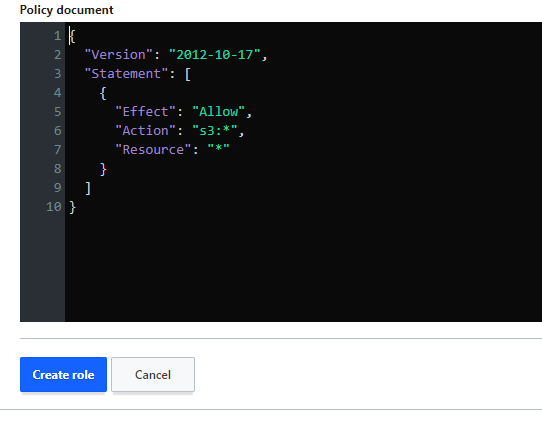
We select aws, there we need to configure the aws with the respective access key and secret key as follows:



And once it is done, we can create role by clicking on create role as follows:

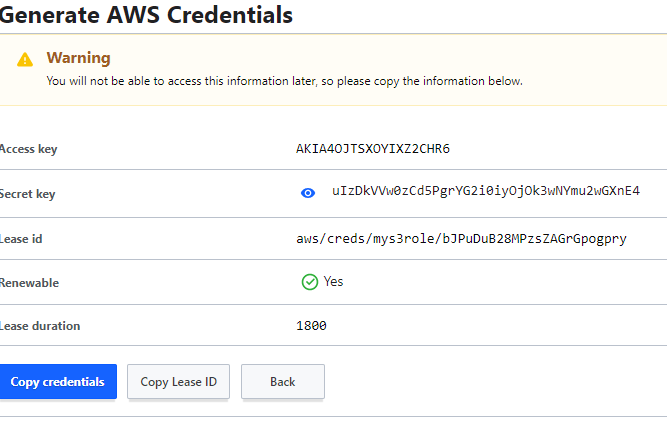


We provide role name and credential type and policy document for the roles we create like s3 and ec2 roles.



Once we are done and when we click on create role, the corresponding role gets created, depending on the policy document or the policy arn provided.

For generating creds using ui, we select the role and then click on generate to generate creds for the role as follows:



### Using API

**To Configuring root :**

Create a payload file with the aws credentials:

**payload.json:**

{

"access\_key": "AKIA4OJTSXOYKQC7CPMB",

"secret\_key": "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",

"region": "us-east-1"

}

Curl command to configure :

curl \

--header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" \

--request POST \

--data @payload.json \

http:// 1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/aws/config/root

To read root credentials:

curl --header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" <http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/aws/config/root>

**output :**

{"request\_id":"d26f0a93-eea0-b718-2cfc-58cd41cec4b8","lease\_id":"","renewable":false,"lease\_duration":0,"data":{"access\_key":"AKIA4OJTSXOYKQC7CPMB","iam\_endpoint":"","max\_retries":-1,"region":"us-east-1","sts\_endpoint":""},"wrap\_info":null,"warnings":null,"auth":null}

**To Configure lease :**

**Leasepayload.json:**

{

"lease": "30m",

"lease\_max": "1h"

}

curl \

--header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" \

--request POST \

--data @leasepayload.json \

http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/aws/config/lease

**To read the lease configuration:**

curl --header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/aws/

config/lease

**output:**

{"request\_id":"999ff9ed-dddf-39a3-b8b4-cd24ef9777da","lease\_id":"","renewable":false,"lease\_duration":0,"data":{"lease":"30m0s","lease\_max":"1h0m0s"},"wrap\_info":null,"warnings":null,"auth":null}

**Create/update role :**

**rolepayload.json:**

{

"credential\_type": “assumed role”

"role\_arns”: “arn:aws:iam::052578464564:policy/eksRead”

}

Configure using following curl command :

curl \

--header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" \

--request POST \

--data @rolepayload.json \

http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/aws/roles/eksaccess

read role using following command:

curl \

--header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" \

http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/aws/roles/eksaccess

**output:**

{"request\_id":"28839470-1559-671d-af25-54d309c3e1ba","lease\_id":"","renewable":false,"lease\_duration":0,"data":{"credential\_type":"iam\_user","default\_sts\_ttl":0,"iam\_groups":null,"max\_sts\_ttl":0,"permissions\_boundary\_arn":"","policy\_arns":["arn:aws:iam::052578464564:policy/eksRead"],"policy\_document":"","role\_arns":null,"user\_path":""},"wrap\_info":null,"warnings":null,"auth":null}

**List all the roles present in the path:**

curl \

--header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" \

--request LIST \

<http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/aws/roles>

**output:**

{"request\_id":"10d578d1-609e-ae1b-2c05-bac1f2cefe77","lease\_id":"","renewable":false,"lease\_duration":0,"data":{"keys":["ec2role","eksaccess","mys3role","s3role"]},"wrap\_info":null,"warnings":null,"auth":null}

**To delete role:**

curl \

--header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" \

--request DELETE http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/aws/roles

**To generate secrets:**

curl --header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/aws/creds/ec2role

{"request\_id":"449a7bd6-916e-fbef-6bb9-12ef57e704cb","lease\_id":"aws/creds/ec2role/k6kCQkLonEaXnGgN3gdF6gSh","renewable":true,"lease\_duration":1800,"data":{"access\_key":"AKIA4OJTSXOYBWJHU24P","secret\_key":"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","security\_token":null},"wrap\_info":null,"warnings":null,"auth":null}

**Reference Links**: <https://www.vaultproject.io/api-docs/secret/aws>

## DATABASE SECRET ENGINE

The database secrets engine generates database credentials dynamically based on configured roles. It works with a number of different databases through a plugin interface. There are a number of built-in database types and an exposed framework for running custom database types for extendibility. This means that services that need to access a database no longer need to hardcode credentials: they can request them from Vault, and use Vault's leasing mechanism to more easily roll keys. These are referred to as "dynamic roles" or "dynamic secrets".

### POSTGRESQL

Pull a Postgres server image with docker.

$docker pull postgres:latest

Create a Postgres database with a root user named root with the password rootpassword.

$docker run \

--name postgres \

--env POSTGRES\_USER=root \

--env POSTGRES\_PASSWORD=rootpassword \

--detach \

--publish 5432:5432 \

postgres

Connect to the Postgres database via the CLI within the postgres container.

$docker exec -it postgres psql

Once we are connected we will have a new prompt *root#*

Create a role named ro.

CREATE ROLE ro NOINHERIT;

CREATE ROLE

Grant the ability to read all tables to the role named ro.

GRANT SELECT ON ALL TABLES IN SCHEMA public TO "ro";

GRANT

Now, login to your vault server with the root token available,

$vault login

**ENABLE THE DATABASE SECRETS ENGINE**

Enable the database secrets engine at the database/ path.

$vault secrets enable database

**CONFIGURE POSTGRESQL SECRETS ENGINE**

Configure the database secrets engine with the connection credentials for the Postgres database.

$vault write database/config/postgresql \

plugin\_name=postgresql-database-plugin \ connection\_url="postgresql://{{username}}:{{password}}@54.255.248.129:54 32/postgres?sslmode=disable"

allowed\_roles=readonly \

username="root" \

password="rootpassword"

**CREATE A ROLE**

In [Step 2](https://learn.hashicorp.com/tutorials/vault/database-secrets#step-2-configure-postgresql-secrets-engine), we configured the PostgreSQL secrets engine with the allowed role named readonly. A role is a logical name within Vault that maps to database credentials. These credentials are expressed as SQL statements and assigned to the Vault role.

vault write database/roles/readonly \

db\_name=postgresql \

creation\_statements="CREATE ROLE \"{{name}}\" WITH LOGIN PASSWORD '{{password}}' VALID UNTIL '{{expiration}}'; \

GRANT SELECT ON ALL TABLES IN SCHEMA public TO \"{{name}}\";" \

default\_ttl="1h" \

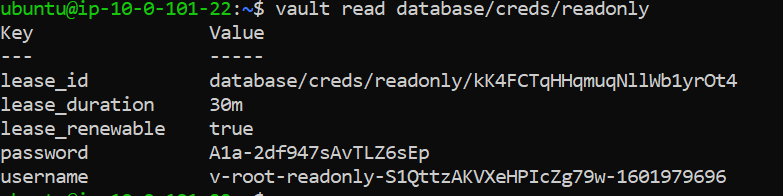
max\_ttl="2h"

**REQUEST POSTGRESQL CREDENTIALS**

The applications that require the database credentials read them from the secret engine's *readonly* role.

Read credentials from the readonly database role.

$vault read database/creds/readonly



**For Generating secrets dynamically using Curl command:**

$curl \

    --header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS " \

    --request GET \

<http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/database/creds/readonly>

Reference link :

<https://www.vaultproject.io/api-docs/secret/databases/postgresql>

<https://www.vaultproject.io/docs/secrets/databases/postgresql>

### MYSQL

**DOCKER COMMANDS TO RUN MYSQL CONTAINER**

docker pull mysql

docker run --name db -d -p 3306:3306 -e MYSQL\_ROOT\_PASSWORD=Welcome123 mysql

**VERIFYING THE USERS**

docker exec -it c737e4e2914e bin/bash

mysql -uroot –pWelcome123

check the users using following command,

select user from mysql.user;

**output :**

user |

+------------------+

| root |

| mysql.infoschema |

| mysql.session |

| mysql.sys |

| root |

+------------------+

**CONFIGURE THE MYSQL DATABASE ENGINE FOR VAULT ON ONE OF THE VAULT VMS**

The following command will create new user by replacing the {{name }} & {{password}} with the username and password which is created dynamically during **vault read**

vault write database/roles/my-sql-role \

db\_name=my-mysql-database \

creation\_statements="CREATE USER '{{name}}'@'%' IDENTIFIED BY '{{password}}';GRANT SELECT ON \*.\* TO '{{name}}'@'%';" \

default\_ttl="1h" \

max\_ttl="1h"

The following command will install the required plugin for mysql and then configure the root user credentials using which a dynamic user credentials are generated.

vault write database/config/my-mysql-database \

plugin\_name=mysql-database-plugin \

connection\_url="{{username}}:{{password}}@tcp(54.255.248.129:3306)/" \

allowed\_roles="my-sql-role" \

username="root" \

password="Welcome123"

Now, perform vault read operation to generate dynamic mysql credentials

vault read database/creds/my-sql-role

**output:**

Key Value

--- -----

lease\_id database/creds/my-sql-role/aGYp7UIjCqrc3rdRBWbUvwdP

lease\_duration 1h

lease\_renewable true

password A1a-kaJNJ0FiTd2BcahZ

username v-root-my-role-XARNzWzG7o6bfev2t

To check the created temporary users in sql :

docker exec -it c737e4e2914e bin/bash

mysql -uroot –pWelcome123

instead of root you can try using following username and password :

mysql -uv-root-my-role-XARNzWzG7o6bfev2t –pA1a-kaJNJ0FiTd2BcahZ

See all the users using following command ,

select user from mysql.user

**output:**

+----------------------------------+

| user |

+----------------------------------+

| root |

| v-root-my-role-2XZ1FFi4NYTGFdKVe |

| v-root-my-role-XARNzWzG7o6bfev2t |

| mysql.infoschema |

| mysql.session |

| mysql.sys |

| root |

+----------------------------------+

**Curl command to generate mysql secrets :**

curl \

--header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS " \

--request GET \

http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com /v1/database/creds/my-sql-role

**output:**

{"request\_id":"bbb61326-8a25-2861-e9c6-9d04c33fa39f","lease\_id":"database/creds/my-role/Vv8gD6ly9IXqYGoOhyNXOIXN","renewable":true,"lease\_duration":3600,"data":{"password":"A1a-O9SYxBREGFTmQvvZ","username":"v-root-my-role-r0By9OWlgdizdFjm7"},"wrap\_info":null,"warnings":null,"auth":null}

**To list roles in database using curl**

curl \

--header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" \

--request LIST \

<http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/database/roles>

**output:**

{"request\_id":"fd616847-74b8-f31a-3e88-5f52feb3bb74","lease\_id":"","renewable":false,"lease\_duration":0,"data":{"keys":["my-sql-role","readonly"]},"wrap\_info":null,"warnings":null,"auth":null}

**Reference Links**

<https://www.vaultproject.io/api-docs/secret/databases/mysql-maria>

<https://www.vaultproject.io/docs/secrets/databases/mysql-maria>

## TRANSIT SECRET ENGINE

The transit secrets engine handles cryptographic functions on data in-transit. Vault doesn't store the data sent to the secrets engine.

The primary use case for transit is to encrypt data from applications while still storing that encrypted data in some primary data store. This relieves the burden of proper encryption/decryption from application developers and pushes the burden onto the operators of Vault.

**Setting up transit secret engine:**

$ vault secrets enable transit

Success! Enabled the transit secrets engine at: transit/

Create a named encryption key- usually each application has its own encryption key

vault write -f transit/keys/my-key

Success! Data written to: transit/keys/my-key

**Encrypting the plain text :**

All plaintext data **must be base64-encoded**. The reason for this requirement is that Vault does not require that the plaintext is "text". It could be a binary file such as a PDF or image. The easiest safe transport mechanism for this data as part of a JSON payload is to base64-encode it.

base64 <<< "my secret data"

bXkgc2VjcmV0IGRhdGEK

vault write transit/encrypt/my-key plaintext=bXkgc2VjcmV0IGRhdGEK

Key Value

--- -----

ciphertext plbCR4gIDNY60tc2nnhHqEEveIb3oKpVWVWO1UGmzq3UNmaBrnez+Ww3OA==

key\_version 1

**Decrypting the cipher text as follows:**

vault write transit/decrypt/my-key ciphertext= vault:v1:plbCR4gIDNY60tc2nnhHqEEveIb3oKpVWVWO1UGmzq3UNmaBrnez+Ww3OA==

**Output:**

Key Value

--- -----

plaintext bXkgc2VjcmV0IGRhdGEK

**Decode using base 64:**

base64 --decode <<< "bXkgc2VjcmV0IGRhdGEK"

my secret data

**Rotate the underlying encryption key:**

This will generate a new encryption key and add it to the keyring for the named key:

vault write -f transit/keys/my-key/rotate

Success! Data written to: transit/keys/my-key/rotate

Upgrade already-encrypted data to a new key. Vault will decrypt the value using the appropriate key in the keyring and then encrypted the resulting plaintext with the newest key in the keyring.

vault write transit/rewrap/my-key ciphertext=vault:v1:plbCR4gIDNY60tc2nnhHqEEveIb3oKpVWVWO1UGmzq3UNmaBrnez+Ww3OA==

Key Value

--- -----

ciphertext vault:v2:/Db/dlUioKjALFlHhffZDViNBdmU0Rpf294ErSS9lUVZeBKtSk/w3eEZeQ==

key\_version 2

You can now decrypt the new key you will get the same plain text:

vault write transit/decrypt/my-key ciphertext=vault:v2:/Db/dlUioKjALFlHhffZDViNBdmU0Rpf294ErSS9lUVZeBKtSk/w3eEZeQ==

**output:**

Key Value

--- -----

plaintext bXkgc2VjcmV0IGRhdGEK

base64 --decode <<< "bXkgc2VjcmV0IGRhdGEK"

my secret data

## TOTP SECRET ENGINE

The TOTP secrets engine generates time-based credentials according to the TOTP standard. The secrets engine can also be used to generate a new key and validate passwords generated by that key.

The TOTP secrets engine can act as both a generator (like Google Authenticator) and a provider (like the Google.com sign in service).

**As a Provider**

The TOTP secrets engine can also act as a TOTP provider. In this mode, it can be used to generate new keys and validate passwords generated using those keys.

1. Enable the TOTP secrets engine:

$ vault secrets enable totp

Success! Enabled the totp secrets engine at: totp/

By default, the secrets engine will mount at the name of the engine. To enable the secrets engine at a different path, use the -path argument.

1. Create a named key, using the generate option. This tells Vault to be the provider:

$ vault write totp/keys/pavan \

generate=true \

issuer=Vault \

account\_name=pavan@test.com

Key Value

--- -----

barcode iVBORw0KGgoAAAANSUhEUgAAAMgAAADIEAAAAADYoy0BA...url otpauth://totp/Vault:user@test.com?algorithm=SHA1&digits=6&issuer=Vault&period=30&secret=V7MBSK324I7KF6KVW34NDFH2GYHIF6JY

The response includes a base64-encoded barcode and OTP url. Both are equivalent. Give these to the user who is authenticating with TOTP.

QR code is returned, it consists of base64-formatted PNG bytes. You can embed it in a web page by including the base64 string in an img tag with the prefix data:image/png;base64

<img src="data:image/png;base64,iVBORw0KGgoAAAANSUh.." />

**Usage:**

As a user, validate a TOTP code generated by a third-party app like google authenticator:

$ vault write totp/code/pavan code=886531

Key Value

--- -----

valid true

## SSH SECRET ENGINE

The Vault SSH secrets engine provides secure authentication and authorization for access to machines via the SSH protocol. The Vault SSH secrets engine helps manage access to machine infrastructure, providing several ways to issue SSH credentials.

The Vault SSH secrets engine supports the following modes.

* Signed ssh certificates
* One-time ssh passwords
* Dynamic ssh keys

**Signed ssh certificates**

The signed SSH certificates is the simplest and most powerful in terms of setup complexity and in terms of being platform agnostic. By leveraging Vault's powerful CA capabilities and functionality built into OpenSSH, clients can SSH into target hosts using their own local SSH keys.

Before a client can request their SSH key be signed, the Vault SSH secrets engine must be configured. This is done by vault administrator.

**Step-1: enable ssh secret engine with path ssh-client-signer**

vault secrets enable -path=ssh-client-signer ssh

Success! Enabled the ssh secrets engine at: ssh-client-signer/

vault write ssh-host-signer/config/ca generate\_signing\_key=true

**Step -2 :**

Configure Vault with a CA for signing client keys using the /config/ca endpoint. If you do not have an internal CA, Vault can generate a keypair for you.

cat generateca.json

{

"generate\_signing\_key":"true"

}

curl --header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" --request POST --data @generateca.json <http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/ssh-client-signer/config/ca>

{"request\_id":"7e1a03d9-9148-07cb-e5de-a00bf5925ed2","lease\_id":"","renewable":false,"lease\_duration":0,"data":{"public\_key":"ssh-rsa \n"},"wrap\_info":null,"warnings":null,"auth":null}

**Step-3:**

Add the public key to all target host's SSH configuration

sudo curl -o /etc/ssh/trusted-user-ca-keys.pem <http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/ssh-client-signer/public_key>

Add the path where the public key contents are stored to the SSH configuration file as the TrustedUserCAKeys option.

root@ip-10-44-0-183:/etc/ssh# vi sshd\_config

add the following line

TrustedUserCAKeys /etc/ssh/trusted-user-ca-keys.pem

Restart the SSH service to pick up the changes.

systemctl restart ssh.service

Create a named Vault role for signing client keys.

Because of the way some SSH certificate features are implemented, options are passed as a map. The following example adds the permit-pty extension to the certificate.

curl --header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" --request POST --data @ sshrole.json [http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/ ssh-client-signer/roles/my-role](http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/ssh-client-signer/config/ca)

**sshrole.json**

ubuntu@ip-10-44-0-183:~$ cat sshrole.json

{

"allow\_user\_certificates": true,

"allowed\_users": "\*",

"allowed\_extensions": "permit-pty,permit-port-forwarding",

"default\_extensions": [

{

"permit-pty": ""

}

],

"key\_type": "ca",

"default\_user": "ubuntu",

"ttl": "2400m0s"

}

**CLIENT SSH AUTHENTICATION**

The following steps are performed by the client (user) that wants to authenticate to machines managed by Vault. These commands are usually run from the client's local workstation.

Locate or generate the SSH public key. Usually this is id\_rsa.pub. If you do not have an SSH keypair, generate one:

**Step-1 : Generate ssh :**

ssh-keygen -t rsa –C ubuntu@

this will generate a private and public key pairs.

cat id\_rsa.pub

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDDF5ADOiw4P61jS8AWk6EYwduoZASs3AiruxvFJK6t7e9askH5/gKo84OQWUeGlYkK8Peqix7Aft65NFSQcx2EsaVCyu4AcNuaC7XCdDjImk6xb/p3QdAbq2NYLu8fCdAsfgHiqs/QSIdDo3+3J0Io8supPqfWniopGSjwh6ss0YEXgGz8PmfI7+7OZlPYRwn5aGbPPhKLqI+6WBCmlvOM+MKjtcVn4NL7kP1lL/MZQ+DwkcMT0cJF7z6Cd+HyKbM5hPozeF3YwKmSIzByVDydDtsOyMNLST3hDTGcAOItD7zZb1qPjTCFqFBUGm6baCxL

cat id\_rsa

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





-----END RSA PRIVATE KEY-----

**Step-2 :** Ask Vault to sign your **public key**.

**publickey.json :**

cat > publickey.json

{

"public\_key" : "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDDF5ADOiw4P61jS8AWk6EYwduoZASs3AiruxvFJK6t7e9askH5/gKo84OQWUeGlYkK8Peqix7Aft65NFSQcx2EsaVCyu4AcNuaC7XCdDjImk6xb/p3QdAbq2NYLu8fCdAsfgHiqs/QSIdDo3+3J0Io8supPqfWniopGSjwh6ss0YEXgGz8PmfI7+7OZlPYRwn5aGbPPhKLqI+6WBCmlvOM+MKjtcVn4NL7kP1lL/MZQ+DwkcMT0cJF7z6Cd+HyKbM5hPozeF3YwKmSIzByVDydDtsOyMNLST3hDTGcAOItD7zZb1qPjTCFqFBUGm6baCxL73k4qnFTniPq5WyXogrx ubuntu@"

}

**Curl command to sign the key :**

curl --header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" --request POST --data @publickey.json [http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/ssh-client-signer/sign/my-role](http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/ssh-client-signer/sign/my-role%20)  > signed-cert.pub

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 2725 0 2317 100 408 59410 10461 --:--:-- --:--:-- --:--:-- 69871

**Using cli you can use the following command:**

vault write -field=signed\_key ssh-client-signer/sign/my-role \

public\_key=@$HOME/ubuntu/id\_rsa.pub > signed-cert.pub

You will get a signed public key from the above command with which you can ssh into your system.

To View enabled extensions, principals, and metadata of the signed key.

ssh-keygen -Lf sign-cert.pub

**output:**

signed-cert.pub:

Type: ssh-rsa-cert-v01@openssh.com user certificate

Public key: RSA-CERT SHA256:bYxSlPQkRvzKlh7Vfnh7kDOv1LZAAc3Ut9+P4ffmU58

Signing CA: RSA SHA256:rMjg1OLURLDL5SMGk1rx7lPxyaYNO6inaIZ1Kp4y6yM

Key ID: "vault-root-6d8c5294f42446fcca961ed57e787b9033afd4b64001cdd4b7df8fe1f7e6539f"

Serial: 17277563621986071282

Valid: from 2020-10-14T04:58:10 to 2020-10-14T05:28:40

Principals:

ubuntu

Critical Options: (none)

Extensions:

permit-pty

signed-cert.pub:2: invalid key: invalid format

**ssh into your machine:**

SSH into the host machine using the signed key. You must supply both the signed public key from Vault **and** the corresponding private key as authentication to the SSH call.

ssh -i signed-cert.pub -i id\_rsa [ubuntu@10.44.0.183](mailto:ubuntu@10.44.0.183)

The authenticity of host '10.44.0.183 (10.44.0.183)' can't be established.

ECDSA key fingerprint is SHA256:AjO/KC3anL4bKwt5cd6qYhG16ulkXgMEf3wqTd0h0TQ.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added '10.44.0.183' (ECDSA) to the list of known hosts.

Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 5.4.0-1025-aws x86\_64)

\* Documentation: https://help.ubuntu.com

\* Management: https://landscape.canonical.com

\* Support: https://ubuntu.com/advantage

System information as of Wed Oct 14 05:03:43 UTC 2020

System load: 0.0 Processes: 106

Usage of /: 3.4% of 96.88GB Users logged in: 1

Memory usage: 64% IP address for eth0: 10.44.0.183

Swap usage: 0%

\* Kubernetes 1.19 is out! Get it in one command with:

sudo snap install microk8s --channel=1.19 --classic

https://microk8s.io/ has docs and details.

\* Canonical Livepatch is available for installation.

- Reduce system reboots and improve kernel security. Activate at:

https://ubuntu.com/livepatch

6 packages can be updated.

0 updates are security updates.

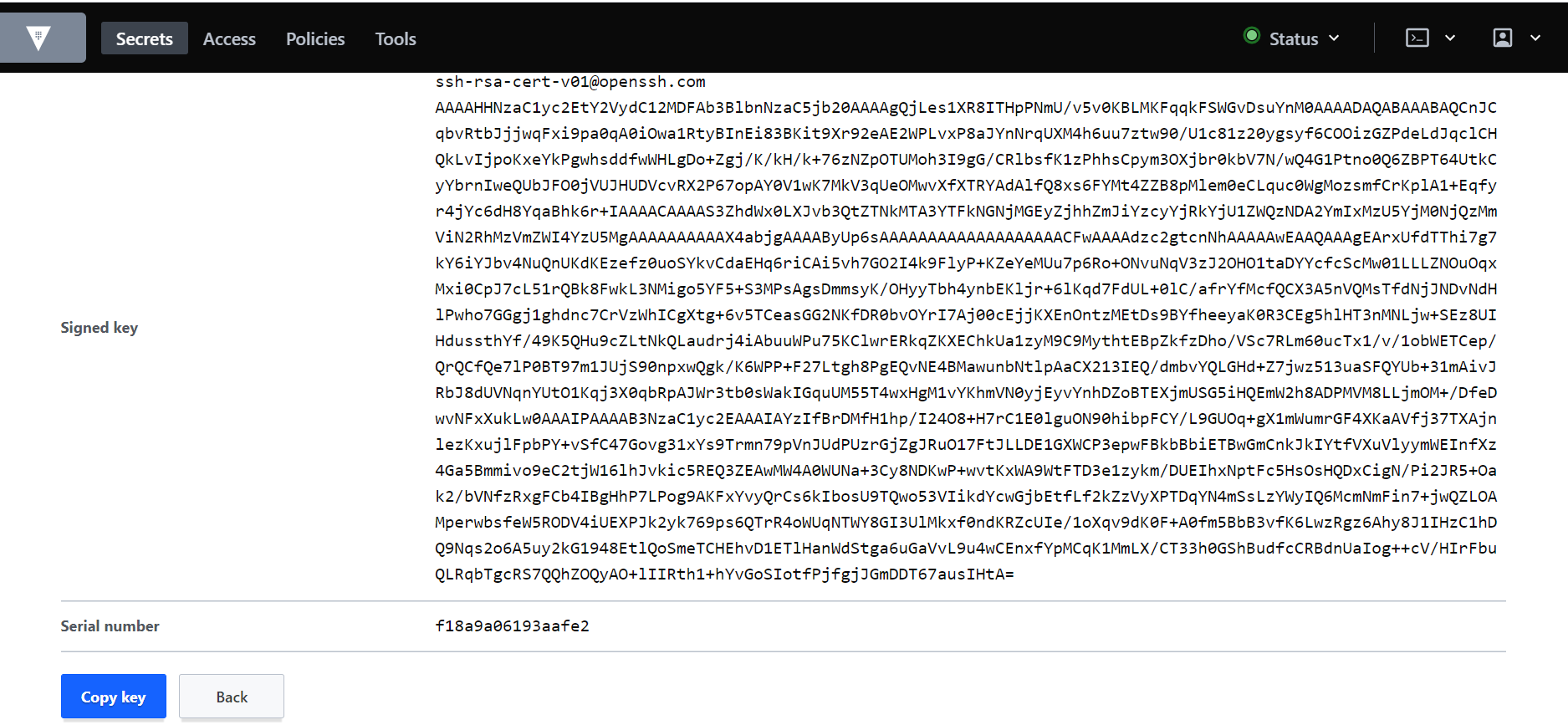
New release '20.04.1 LTS' available.

Run 'do-release-upgrade' to upgrade to it.

Last login: Wed Oct 14 04:55:25 2020 from 157.48.199.38



1. Paste you **ssh\_host\_rsa\_key.pub** in the public key.
2. Click on sign and then copy the generated signed key.

****

Set permissions on the certificate to be 0640 :

chmod 0640 /etc/ssh/ssh\_host\_rsa\_key-cert.pu

## PKI SECRET ENGINE

The PKI secrets engine generates dynamic X.509 certificates. With this secrets engine, services can get certificates without going through the usual manual process of generating a private key and CSR, submitting to a CA, and waiting for a verification and signing process to complete. Vault's built-in authentication and authorization mechanisms provide the verification functionality.

**STEP – 1:** Enable the PKI secrets engine:

vault secrets enable pki

Success! Enabled the pki secrets engine at: pki/

**STEP - 2 :** Increase the TTL by tuning the secrets engine. The default value of 30 days may be too short, so increase it to 1 year

vault secrets tune -max-lease-ttl=8760h pki

Success! Tuned the secrets engine at: pki/

**STEP – 3 :** Configure a CA certificate and private key. Vault can accept an existing key pair, or it can generate its own self-signed root. In general, we recommend maintaining your root CA outside of Vault and providing Vault a signed intermediate CA

vault write pki/root/generate/internal common\_name=myvault.com ttl=87600h

**output:**

Key Value

--- -----

certificate -----BEGIN CERTIFICATE-----

MIIDNTCCAh2gAwIBAgIUeumuCFhwbNcYPrrKa/7gcXFnTWgwDQYJKoZIhvcNAQEL

BQAwFjEUMBIGA1UEAxMLbXl2YXVsdC5jb20wHhcNMjAxMDE1MTAxNjQ0WhcNMzAx

MDEzMTAxNzEzWjAWMRQwEgYDVQQDEwtteXZhdWx0LmNvbTCCASIwDQYJKoZIhvcN

AQEBBQADggEPADCCAQoCggEBAM2b8BbDWIwzQFvJ+1MUPOBk0fGWxzldG9kxO1lP

jvvj2yeSbj1I6gKHOWPusqG53xfas9x1oTEf/Kpcogi0YxTn1Mgaa4avwCzJ8WJn

J15f6mySNFkh9TEr+NV5WS2ak61pR8D8kvuCBR5SdukStGyFNXh9pB2CnVl8rDmk

CjNFFQ3PSbJ9X5+3xcue1JX4SILoFJ9XnsLs7BLDzv3eibGm5UzZzKkDPQD2IlnN

KGsDoG4Ly4Wzg2zeDGhgE3pywYPhSgPKg+BV2NUzOpTh+fR4AMA4O6IYB9J3lN05

11baOKM8XsZC8SrXRpEqfpd4ieVwwtZ/Pgv8gpG75cVcIFMCAwEAAaN7MHkwDgYD

VR0PAQH/BAQDAgEGMA8GA1UdEwEB/wQFMAMBAf8wHQYDVR0OBBYEFGyWSm02JuNP

bglyH6gNKNIDXgMAMB8GA1UdIwQYMBaAFGyWSm02JuNPbglyH6gNKNIDXgMAMBYG

A1UdEQQPMA2CC215dmF1bHQuY29tMA0GCSqGSIb3DQEBCwUAA4IBAQB+lUzj2f8B

SdTN3n9MHFk8djtuL6HUKhZ1bXgNC+pSWvogPtznN4zW41kHYkOkbOZjqA+/Y7FA

MP1H+4kCLsRgCSuIO/ddCs2Nf4szPsnm0q2DdsAOZ7Dxxdk48JDszWXhdCHQX+Up

jnw/yz24febPi0qoePi2faLps5mF0fq1cFM8tdho9b9ctKOby9eJAvQyA6kI/JlV

rMbhrmDb7FLQtVTSll81WJEWyO2wUGVZ0oBZZfUA7qRVQSJC7PMYMj+0ojWhd7AS

BB0KEF0BPeOComZdUGyXR3m2VFXDdNYnxKYs5ycZn4Pvj7gF1dmxynybOW+ScWnJ

fbgE3ne5gG9C

-----END CERTIFICATE-----

expiration 1918117033

issuing\_ca -----BEGIN CERTIFICATE-----

MIIDNTCCAh2gAwIBAgIUeumuCFhwbNcYPrrKa/7gcXFnTWgwDQYJKoZIhvcNAQEL

BQAwFjEUMBIGA1UEAxMLbXl2YXVsdC5jb20wHhcNMjAxMDE1MTAxNjQ0WhcNMzAx

MDEzMTAxNzEzWjAWMRQwEgYDVQQDEwtteXZhdWx0LmNvbTCCASIwDQYJKoZIhvcN

AQEBBQADggEPADCCAQoCggEBAM2b8BbDWIwzQFvJ+1MUPOBk0fGWxzldG9kxO1lP

jvvj2yeSbj1I6gKHOWPusqG53xfas9x1oTEf/Kpcogi0YxTn1Mgaa4avwCzJ8WJn

J15f6mySNFkh9TEr+NV5WS2ak61pR8D8kvuCBR5SdukStGyFNXh9pB2CnVl8rDmk

CjNFFQ3PSbJ9X5+3xcue1JX4SILoFJ9XnsLs7BLDzv3eibGm5UzZzKkDPQD2IlnN

KGsDoG4Ly4Wzg2zeDGhgE3pywYPhSgPKg+BV2NUzOpTh+fR4AMA4O6IYB9J3lN05

11baOKM8XsZC8SrXRpEqfpd4ieVwwtZ/Pgv8gpG75cVcIFMCAwEAAaN7MHkwDgYD

VR0PAQH/BAQDAgEGMA8GA1UdEwEB/wQFMAMBAf8wHQYDVR0OBBYEFGyWSm02JuNP

bglyH6gNKNIDXgMAMB8GA1UdIwQYMBaAFGyWSm02JuNPbglyH6gNKNIDXgMAMBYG

A1UdEQQPMA2CC215dmF1bHQuY29tMA0GCSqGSIb3DQEBCwUAA4IBAQB+lUzj2f8B

SdTN3n9MHFk8djtuL6HUKhZ1bXgNC+pSWvogPtznN4zW41kHYkOkbOZjqA+/Y7FA

MP1H+4kCLsRgCSuIO/ddCs2Nf4szPsnm0q2DdsAOZ7Dxxdk48JDszWXhdCHQX+Up

jnw/yz24febPi0qoePi2faLps5mF0fq1cFM8tdho9b9ctKOby9eJAvQyA6kI/JlV

rMbhrmDb7FLQtVTSll81WJEWyO2wUGVZ0oBZZfUA7qRVQSJC7PMYMj+0ojWhd7AS

BB0KEF0BPeOComZdUGyXR3m2VFXDdNYnxKYs5ycZn4Pvj7gF1dmxynybOW+ScWnJ

fbgE3ne5gG9C

-----END CERTIFICATE-----

serial\_number 7a:e9:ae:08:58:70:6c:d7:18:3e:ba:ca:6b:fe:e0:71:71:67:4d:68

The returned certificate is purely informative. The private key is safely stored internally in Vault.

**Step-4 :** Update the CURL location and issuing certificates. These values can be updated in the future.

vault write pki/config/urls issuing\_certificates="http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/pki/ca" crl\_distribution\_points="http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/pki/crl"

Success! Data written to: pki/config/urls

**Step-5 :** Configure a role that maps a name in Vault to a procedure for generating a certificate. When users or machines generate credentials, they are generated against this role.

vault write pki/roles/example-dot-com \

allowed\_domains=my-website.com \

allow\_subdomains=true \

max\_ttl=72h

Success! Data written to: pki/roles/example-dot-com

**Usage :**

After the secrets engine is configured and a user/machine has a Vault token with the proper permission, it can generate credentials.

Generate a new credential by writing to the /issue endpoint with the name of the role:

vault write pki/issue/example-dot-com \

common\_name=www.my-website.com

Key Value

--- -----

certificate -----BEGIN CERTIFICATE-----



-----END CERTIFICATE-----

expiration 1603016891

issuing\_ca -----BEGIN CERTIFICATE-----



-----END CERTIFICATE-----

private\_key -----BEGIN RSA PRIVATE KEY-----



-----END RSA PRIVATE KEY-----

private\_key\_type rsa

serial\_number 3e:b2:b1:47:97:c6:58:86:eb:71:5a:e5:b7:64:55:ef:d7:a4:86:14

# ACCESS MANAGEMENT

We can manage access to secrets by creating policies and by assigning them to users or groups.

**Creation of policies:**

Here, we will have 3 policies each with different access restrictions.

1. **Apple policy** – This policy will restrict access in aws path to ec2role with some additional privileges to login with token, self-revoke and self-renew the token, list the entities present but view only the given entity, view auth methods, renew or update leases, lookup self-capabilities.
2. **Orange policy** – This policy will restrict access in aws path to s3role with some additional privileges to lookup self-capabilities, to view an entity with given id, to view the list of policies.
3. **Sql policy** – in databases access is restricted to my-sql-role with a capability to see self-capabilities on a paths.
4. **Admin policy** – admin policy will give you access to all paths in vault. You can have privileges like a root user.
5. **Admin policy with restricted access to secrets –** other than secrets this allow you to have access to every other path.

**Apple policy:**

# Allow a token to view the auth methods

path "sys/auth" {

capabilities = ["list","read"]

}

# Allow tokens to look up their own properties

path "auth/token/lookup-self" {

capabilities = ["read"]

}

# Allow tokens to renew themselves

path "auth/token/renew-self" {

capabilities = ["update"]

}

# Allow tokens to revoke themselves

path "auth/token/revoke-self" {

capabilities = ["update"]

}

# Allow a token to look up its own capabilities on a path

path "sys/capabilities-self" {

capabilities = ["update"]

}

# Allow a token to look up its own entity by id or name

path "identity/entity/\*" {

capabilities = ["list"]

}

path "identity/entity/id/5de92794-4421-603a-b5db-16f1b791cda7" {

capabilities = ["read", "list", "update", "create", "delete"]

}

# Allow a token to renew a lease via lease\_id in the request body; old path for

# old clients, new path for newer

path "sys/renew" {

capabilities = ["update"]

}

path "sys/leases/renew" {

capabilities = ["update"]

}

# Allow looking up lease properties. This requires knowing the lease ID ahead

# of time and does not divulge any sensitive information.

path "sys/leases/lookup" {

capabilities = ["update"]

}

#Allows access to list the secrets present in aws path

path "aws/\*" {

capabilities = ["list"]

}

#Allows a token to have the listed capabilities on the ec2role only

path "aws/roles/ec2role"{

capabilities = ["list","delete","create","update","read"]

}

#Allows a token to generate the credentials for ec2role only

path "aws/creds/ec2role" {

capabilities = ["read"]

}

**Orange policy:**

#Allows access to list the secrets present in aws path

path "aws/\*" {

capabilities = ["list"]

}

#Allows a token to have the listed capabilities on the ec2role only

path "aws/roles/s3role" {

capabilities = ["list","read","update","delete","create"]

}

#Allows a token to generate the credentials for ec2role only

path "aws/creds/s3role" {

capabilities = ["read"s]

}

# Allow a token to look up its own capabilities on a path

path "sys/capabilities-self" {

capabilities = ["update"]

}

# Allow a token to look up its own entity by id or name

path "identity/entity/\*" {

capabilities = ["list"]

}

path "identity/entity/id/9bcd5c15-c304-398d-bd72-d95fbc34bac9" {

capabilities = ["read","list","update","create","delete"]

}

#Allows a token to list the policies present

path "sys/policies/acl"

{

capabilities = ["list"]

}

**SQL policy:**

# Allow a token to look up its own capabilities on a path

path "sys/capabilities-self" {

capabilities = ["update"]

}

#Allows a token to list the roles in database path

path "database/\*" {

capabilities = ["list"]

}

#Allows a token to have the listed capabilities on my-sql-role only

path "database/roles/my-sql-role" {

capabilities = ["create", "read", "update", "delete", "list"]

}

#Allows a token to generate credentials for my-sql-role

path "database/creds/my-sql-role" {

capabilities = ["create", "read", "update", "delete", "list"]

}

### 

## ADMIN POLICY WITH ROOT PRIVILEGES

path "\*" {

capabilities = ["create", "read", "update", "delete", "list", "sudo"]

}

## ADMIN POLICY WITH RESTRICTED ACCESS TO SECRETS

path "auth/\*"

{

capabilities = ["create", "read", "update", "delete", "list", "sudo"]

}

path "sys/\*"

{

capabilities = ["create", "read", "update", "delete", "list", "sudo"]

}

path "identity/\*"

{

capabilities = ["create", "read", "update", "delete", "list", "sudo"]

}

**Policy to restrict each secret in kv :**

In vault I have to kv paths : kv1/pavan/secrets , kv1/lahari/secrets

#this allows a user to see all the secret engine paths that starts with kv1

path "kv1/\*" {

capabilities = ["list"]

}

#this allows the user to have access to secrets in the following path

path "kv1/lahari/secrets/\*" {

capabilities = ["read","list","update","create","delete"]

}

#this restricts the user from having access to only this secret

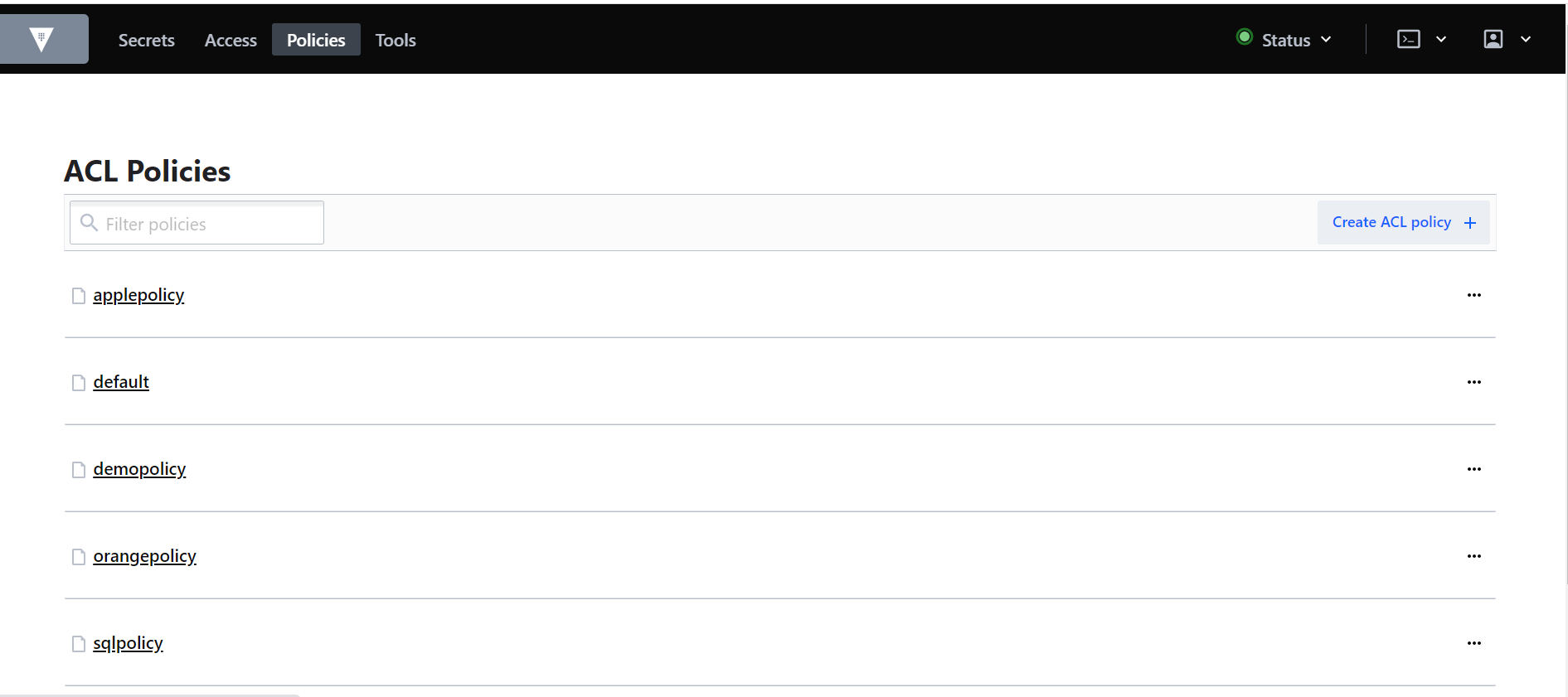
path "kv1/lahari/secrets/data/path1/secret1" {

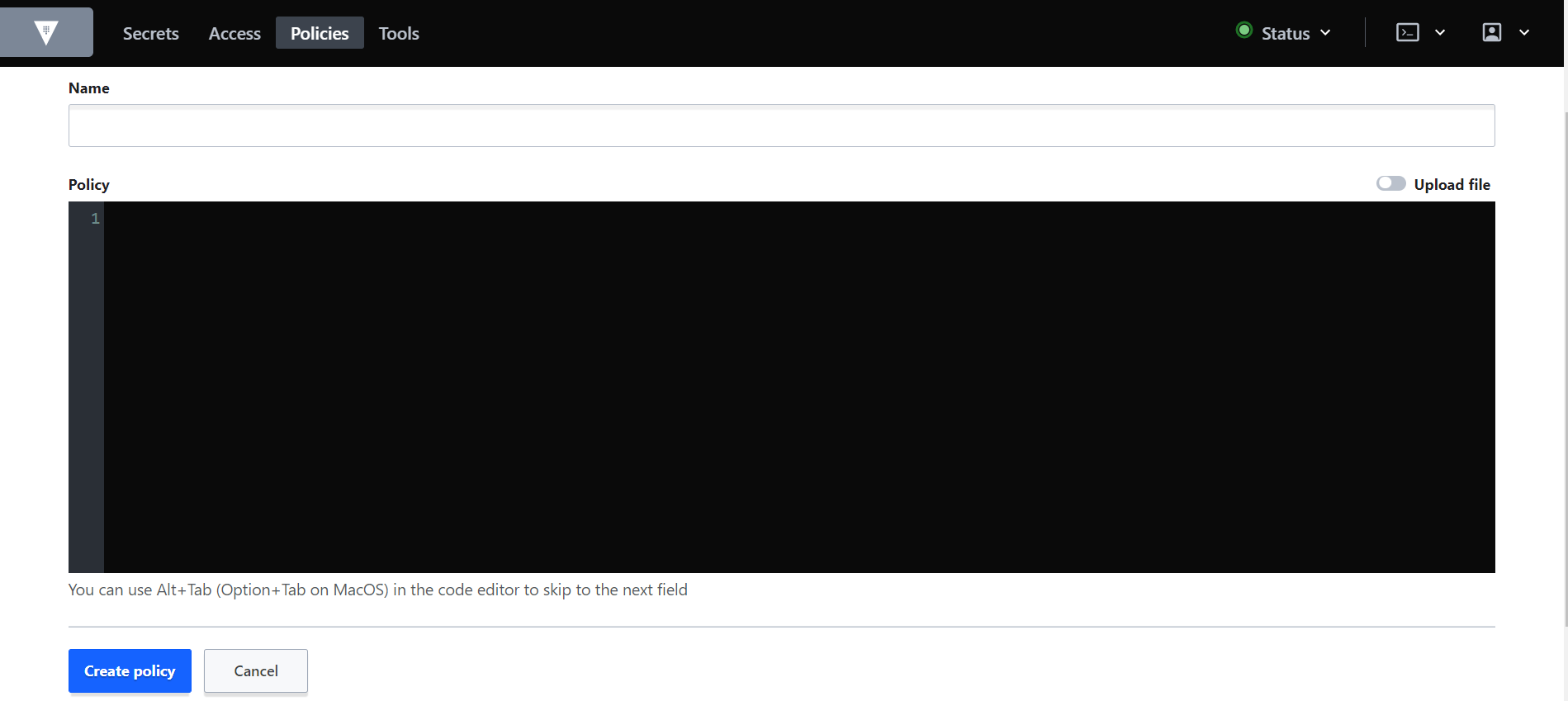
capabilities = [ "deny" ]

}

## Creating policy from UI

1. Login with root credentials or login with user who has permissions to create policy.
2. Goto policies and the click on create acl policy.

****

****

1. As seen in the above picture give the name for your policy and give the policy definition or upload a file with policy definition.
2. Click on create policy.

## Creating policy from CLI

You can view the existing policies using the following command

vault policy list

Create to new policy definition in a .hcl file for example “my-policy.hcl” and give its path in the following command.

vault policy write my-policy ./my-policy.hcl

you can delete a policy using following command

vault policy delete my-policy

**Reference Link:**

<https://www.vaultproject.io/docs/commands/policy>

## Creating policy using api

**To list policies :**

curl \

--header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" \

<http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/sys/policy>

**output:**

{"keys":["applepolicy","default","orangepolicy","sqlpolicy","root"],"policies":["applepolicy","default","orangepolicy","sqlpolicy","root"],"request\_id":"09f40408-863a-de0b-6831-8348fe2e8776","lease\_id":"","renewable":false,"lease\_duration":0,"data":{"keys":["applepolicy","default","orangepolicy","sqlpolicy","root"],"policies":["applepolicy","default","orangepolicy","sqlpolicy","root"]},"wrap\_info":null,"warnings":null,"auth":null}

**To read a policy:**

curl --header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" <http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/sys/policy/sqlpolicy>

**output:**

{"rules":"# Allow a token to look up its own capabilities on a path\npath \"sys/capabilities-self\" {\n capabilities = [\"update\"]\n}\n\npath \"database/\*\" {\n capabilities = [\"list\"]\n}\n\npath \"database/roles/my-sql-role\" {\n capabilities = [\"create\", \"read\", \"update\", \"delete\", \"list\"]\n}\npath \"database/creds/my-sql-role\" {\n\tcapabilities = [\"create\", \"read\", \"update\", \"delete\", \"list\"]\n}","name":"sqlpolicy","request\_id":"4c65cea4-6f6c-cb6c-aefb-b887b782d770","lease\_id":"","renewable":false,"lease\_duration":0,"data":{"name":"sqlpolicy","rules":"# Allow a token to look up its own capabilities on a path\npath \"sys/capabilities-self\" {\n capabilities = [\"update\"]\n}\n\npath \"database/\*\" {\n capabilities = [\"list\"]\n}\n\npath \"database/roles/my-sql-role\" {\n capabilities = [\"create\", \"read\", \"update\", \"delete\", \"list\"]\n}\npath \"database/creds/my-sql-role\" {\n\tcapabilities = [\"create\", \"read\", \"update\", \"delete\", \"list\"]\n}"},"wrap\_info":null,"warnings":null,"auth":null}

**To write a policy:**

Create a payload.json file with your policy

{

"policy": "# Manage auth backends broadly across Vault\npath \"auth/\*\"\n{\n capabilities = [\"create\", \"read\", \"update\", \"delete\", \"list\", \"sudo\"]\n}\n\n# List, create, update, and delete auth backends\npath \"sys/auth/\*\"\n{\n capabilities = [\"create\", \"read\", \"update\", \"delete\", \"sudo\"]\n}\n\n# List existing policies\npath \"sys/policy\"\n{\n capabilities = [\"read\"]\n}\n\n# Create and manage ACL policies broadly across Vault\npath \"sys/policy/\*\"\n{\n capabilities = [\"create\", \"read\", \"update\", \"delete\", \"list\", \"sudo\"]\n}\n\n# List, create, update, and delete key/value secrets\npath \"secret/\*\"\n{\n capabilities = [\"create\", \"read\", \"update\", \"delete\", \"list\", \"sudo\"]\n}\npath \"dynamic-aws-creds-vault-admin-path/\*\"\n{\n capabilities = [\"create\", \"read\", \"update\", \"delete\", \"list\", \"sudo\"]\n}\n# Manage and manage secret backends broadly across Vault.\npath \"sys/mounts\"\n{\n capabilities = [\"create\", \"read\", \"update\", \"delete\", \"list\", \"sudo\"]\n}\npath \"sys/mounts/\*\"\n{\n capabilities = [\"create\", \"read\", \"update\", \"delete\", \"list\", \"sudo\"]\n}\n\n# Read health checks\npath \"sys/health\"\n{\n capabilities = [\"read\", \"sudo\"]\n}"

}

curl \

--header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS" \

--request PUT \

--data @policypay.json\

http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com /v1/sys/policy/my-policy

Reference Link:

<https://www.vaultproject.io/api-docs/system/policy>

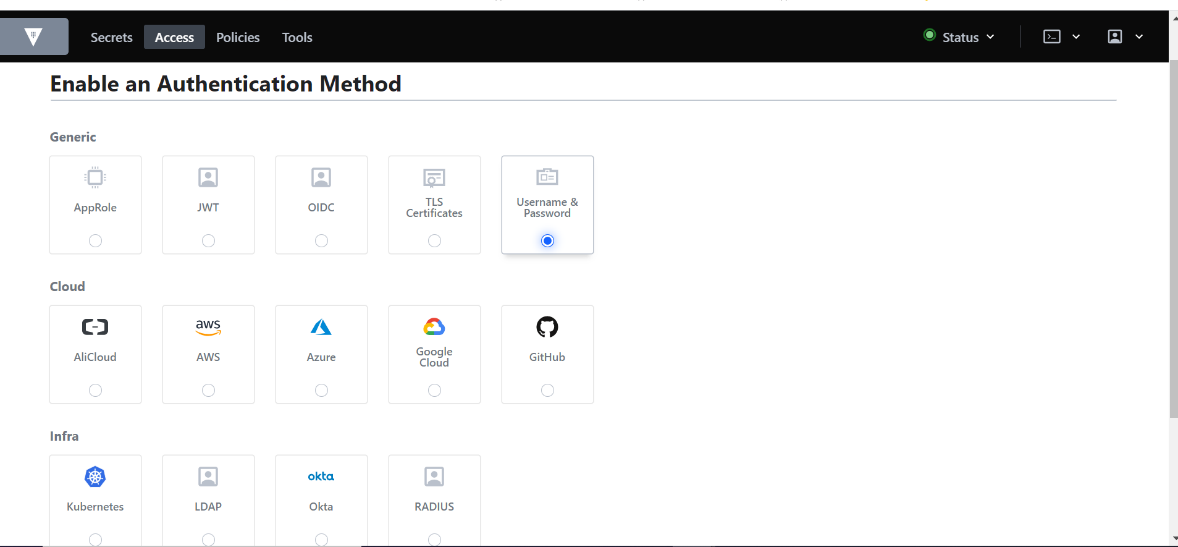
# Creating Users, adding them to groups & assigning the policy to either group or user

## Creating User from UI

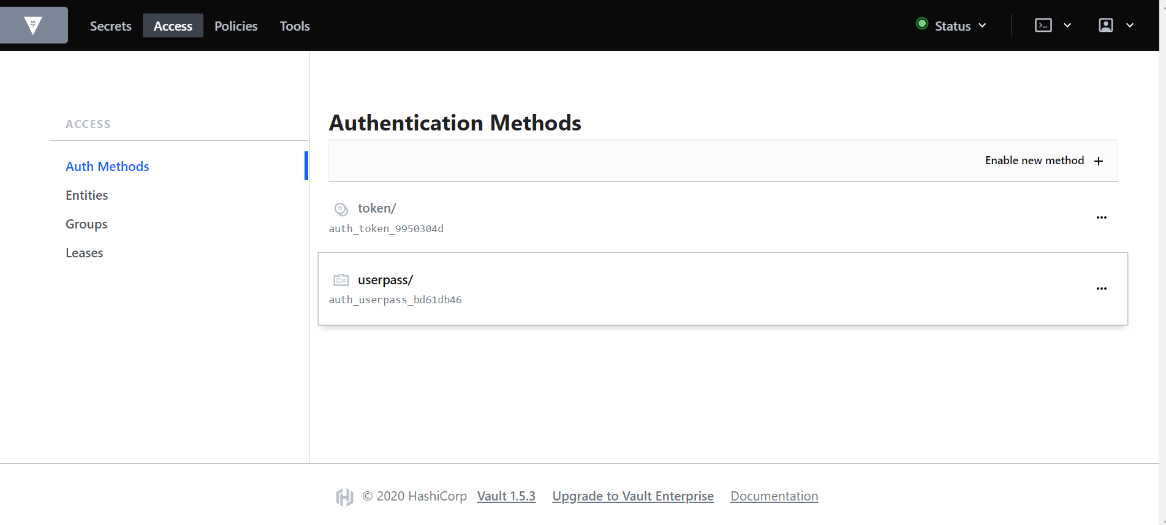
In our scenario, we created 3 users:

1. Pavan
2. Murali
3. sqluser

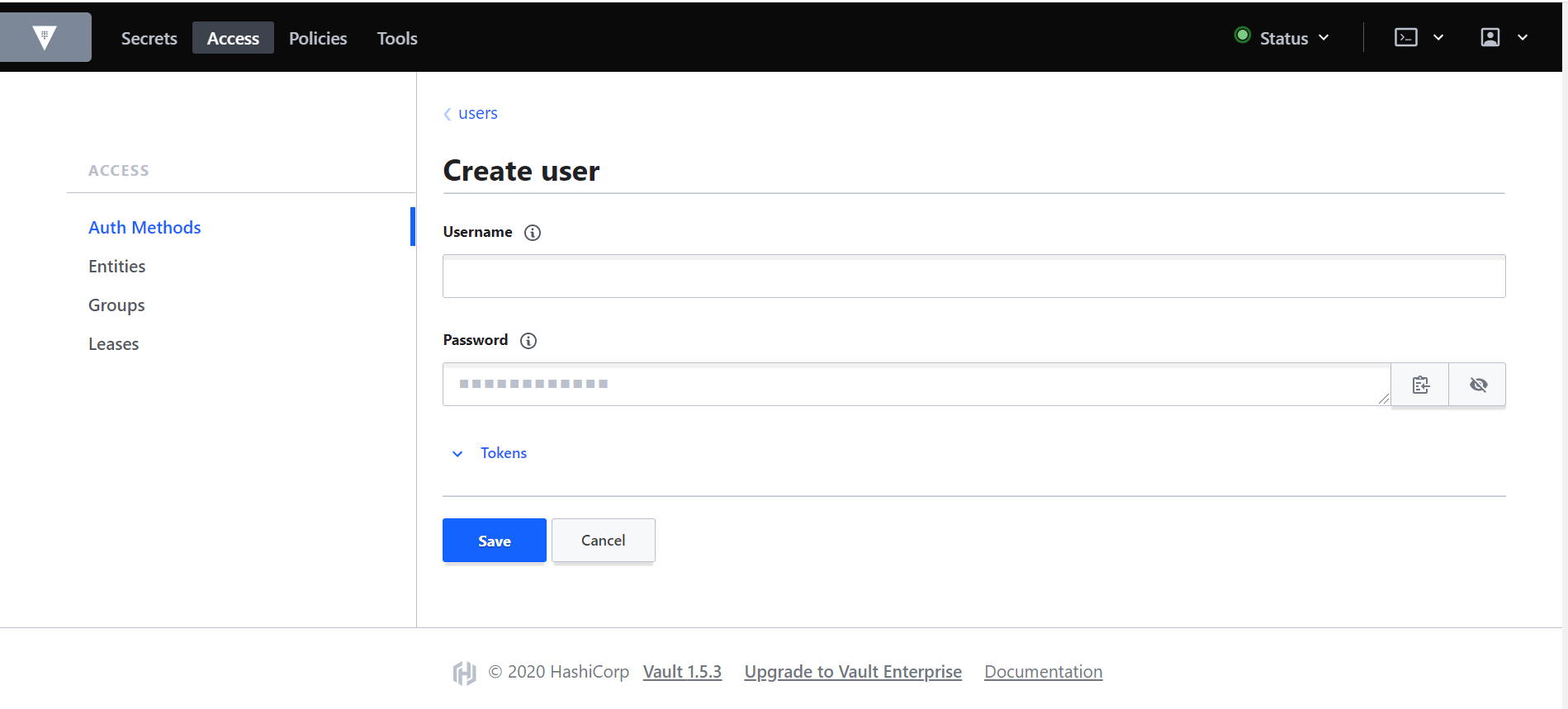
Goto access , click on enable method then select username & password and enable it as shown in following screenshot.

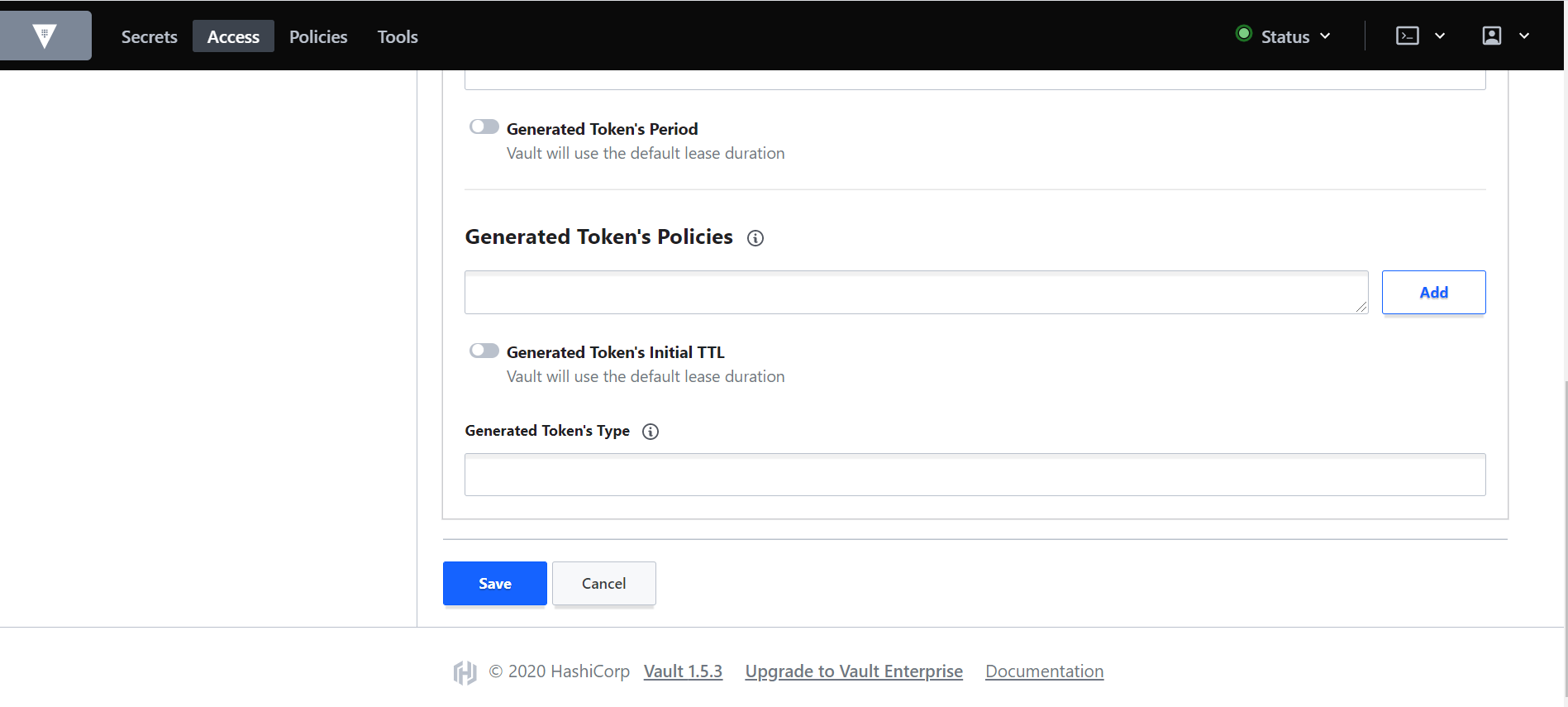


Now you can see userpass enabled.



Click on userpass and click on create user you will be able to see the following screen, give a username and password and save. If you want to configure more then click on tokens there you can assign a policy to the user and also set token lease.





In the above screen give the policy that you want to assign to the user.

## Creating user from API

Create a payload.json file with the configuration

{

“password” : “mypassword”

“policies” : “applepolicy”

}

curl \

--header "X-Vault-Token: s.9NuiX2lkeyVB8YhfC7ct3GDS " \

--request POST \

--data @payload.json \

http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com /v1/auth/userpass/users/newuser

**Create a payload.json with password:**

ubuntu@ip-172-31-87-119:~$ cat muralilogin.json

{

"password": "murali"

}

**Curl command:**

curl --header "X-Vault-Token:s.9ywq6OhmUyrGxl3e6m74dRbp" \

--request POST \

--data @muralilogin.json \

http://3.94.98.140:8200/v1/auth/userpass/users/murali

**Reference Link:**

<https://www.vaultproject.io/api-docs/auth/userpass>

## Creating user Using CLI

Enable the userpass and create using write command as follows:

vault auth enable userpass

vault write auth/userpass/users/newuser \

password=newpassword \

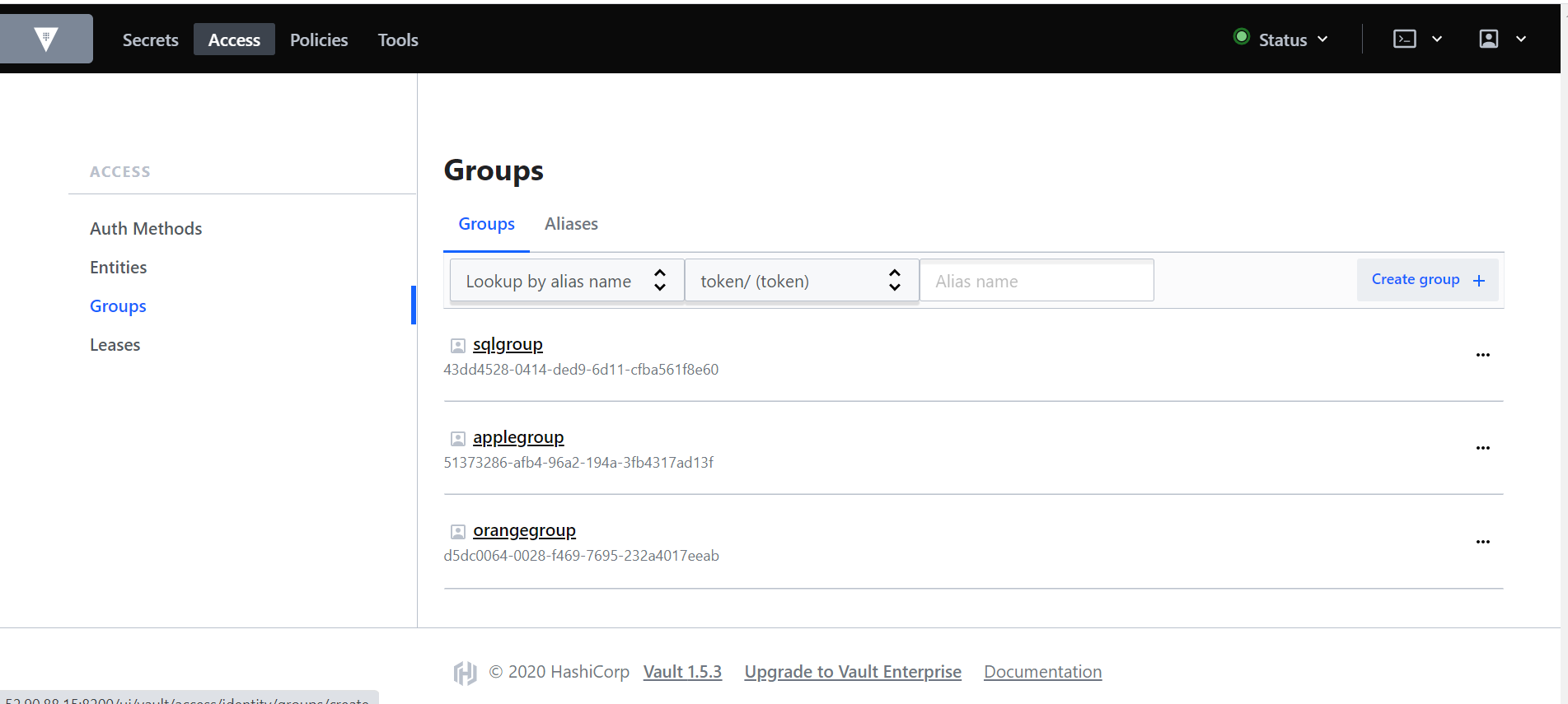
policies=applepolicy

**Reference Link:**

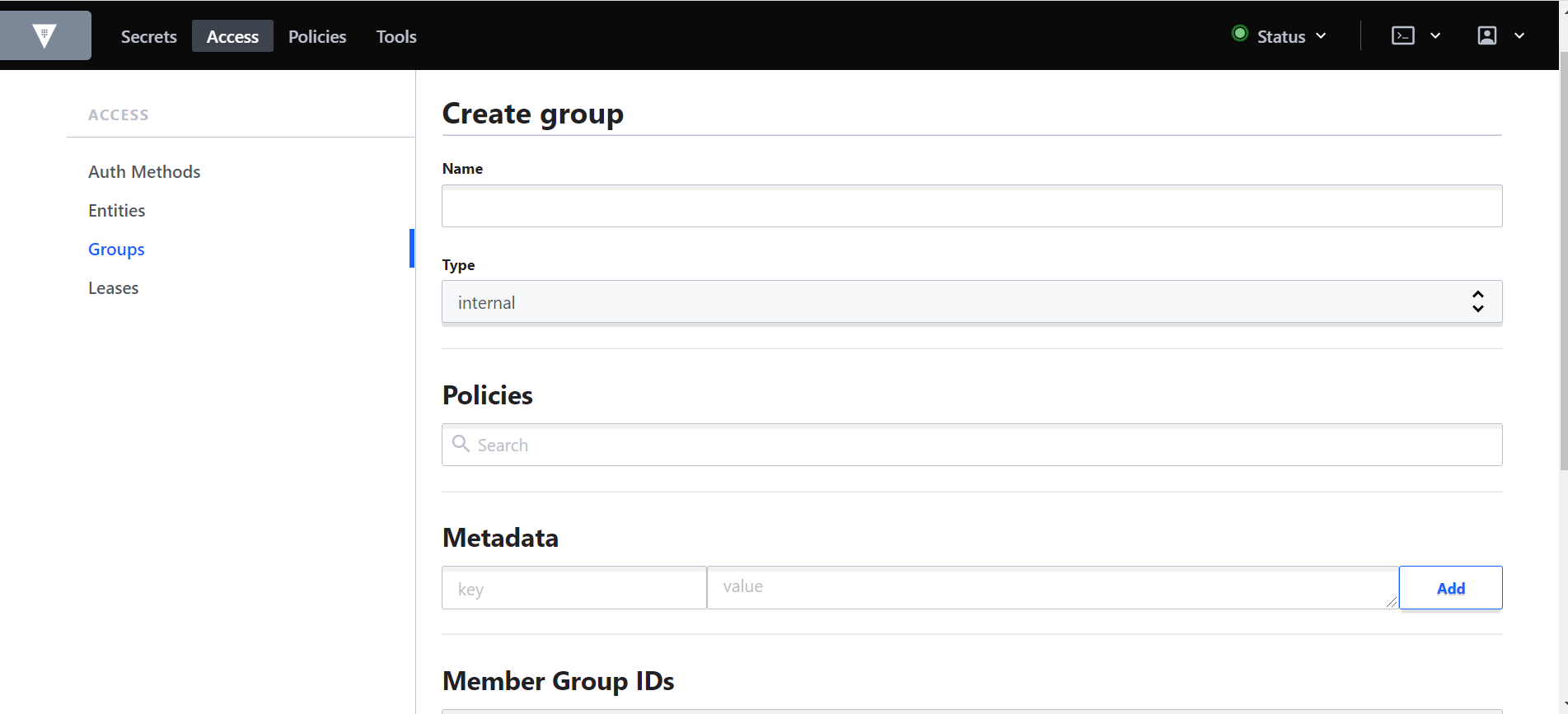
<https://www.vaultproject.io/docs/auth/userpass>

## Creating Group from UI

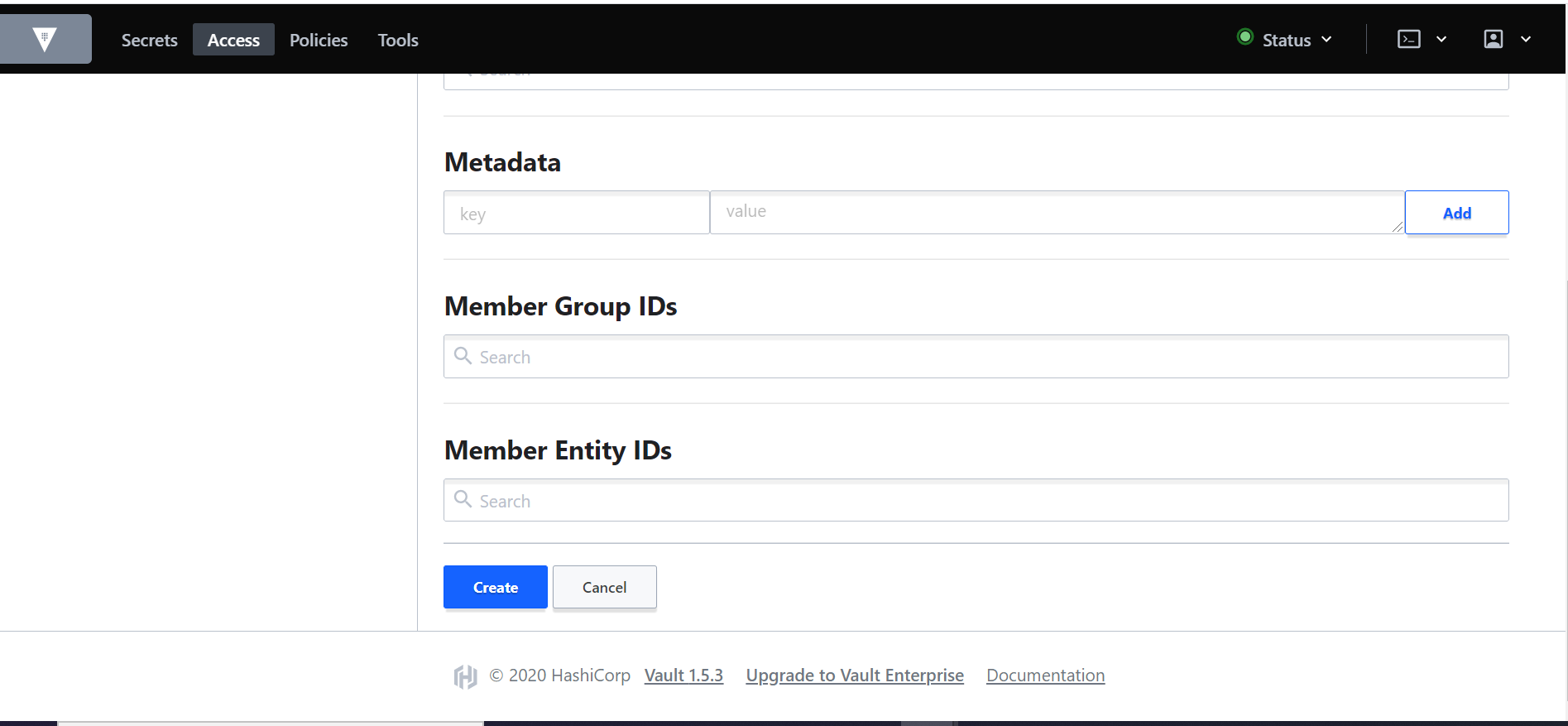
Goto access select groups and then click on create group as shown in the following screen



Give a name and search policy the select the policy it will be applied to all the members in the group



Select the entity id to add the user. Every user you want to add to group should be created and registered to vault. The entity will be created when user logs in for the first time.



In our case we created 3 groups

1. Apple group – with user pavan and with policy – applepolicy
2. Orange group – with user murali and with policy – orangepolicy
3. Sql group – with user sqluser and with policy - sqlpolicy

Reference Link : <https://www.vaultproject.io/api/auth/userpass>

## Login using userpass auth with curl command

**Curl command to login as user:**

create a json file with the password for the user

**login.json**

{

"password": "pavan"

}

**Command :**

curl \

--request POST \

--data @login.json \

<http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/auth/userpass/login/pavan>

**output:**

{"request\_id":"8a89d883-ab30-56c7-559b-62f2e545f40c","lease\_id":"","renewable":false,"lease\_duration":0,"data":null,"wrap\_info":null,"warnings":null,"auth":{"client\_token":"**s.2KBgVYJA157XbhBxo5QaFAXN**","accessor":"ZlF33H0OrZTHrAs00aEwcliB","policies":["applepolicy","default"],"token\_policies":["default"],"identity\_policies":["applepolicy"],"metadata":{"username":"pavan"},"lease\_duration":2764800,"renewable":true,"entity\_id":"5de92794-4421-603a-b5db-16f1b791cda7","token\_type":"service","orphan":true}}

Note :

every time a user logs in using curl a new client\_token will be generated he can use this token as **X-Vault-Token:** s.2KBgVYJA157XbhBxo5QaFAXN **"**

## Generating aws ec2role credentials using new client-token

Pavan has access to ec2 role to generate credentials by using the new client token we can use the following command.

curl --header "X-Vault-Token: s.2KBgVYJA157XbhBxo5QaFAXN" <http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/aws/creds/ec2role>

**output:**

{"request\_id":"7b5e8867-d946-65a5-ca03-bc34b33e8b2a","lease\_id":"aws/creds/ec2role/sKmw8mbYLkZxpCCGRdvYkify","renewable":true,"lease\_duration":1800,"data":{"access\_key":"AKIATSRJXG253T774U42","secret\_key":"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","security\_token":null},"wrap\_info":null,"warnings":null,"auth":null}

**List the roles in aws using the new token:**

curl \

--header "X-Vault-Token: s.2KBgVYJA157XbhBxo5QaFAXN" \

--request LIST \

http://1c05353a-default-vault-c083-1304294467.ap-southeast-1.elb.amazonaws.com/v1/aws/roles

**output:**

{"request\_id":"e3ee5b64-ae33-35dd-fa95-408e5e86984f","lease\_id":"","renewable":false,"lease\_duration":0,"data":{"keys":["ec2role","eksaccess","mys3role","s3role"]},"wrap\_info":null,"warnings":null,"auth":null}

Note :

The generated token will be vaild until the lease gets expired or until you revoke the token manually.

# RAFT BACK-UP AND RESTORE

**Policy to create and restore Raft snapshots :**

path "sys/storage/\*"

{

capabilities = ["update","create","read","delete","list"]

}

Assign this policy to the user to whom you want to give permission to create a backup.

**Login with that user and get client token :**

Backupuser.json:

{

“password” : “backupuser”

}

curl \

--data @backupuser.json \

<http://52.90.88.15:8200/v1/auth/userpass/login/userforbackup>

**output:**

{"request\_id":"82a1190b-f01b-b13a-0986-047833fff5e3","lease\_id":"","renewable":false,"lease\_duration":0,"data":null,"wrap\_info":null,"warnings":null,"auth":{"client\_token":"s.8wghothW3bmw5FRm0mgcawgA","accessor":"Cwg1mNU5gZj3Hda10SwAwtgu","policies":["backup\u0026restore-policy"],"token\_policies":["backup\u0026restore-policy"],"metadata":{"username":"userforbackup"},"lease\_duration":2764800,"renewable":true,"entity\_id":"c63a0aac-7a5c-bfd3-1e1a-e6aae3065579","token\_type":"service","orphan":true}}

**Now create a snapshot using the client token:**

**Using cli:**

vault operator raft snapshot save demo.snapshot

**Using api:**

curl \

--header "X-Vault-Token:s.8wghothW3bmw5FRm0mgcawgA" \

--request GET \

http://52.90.88.15:8200/v1/sys/storage/raft/snapshot > raft.snap

after taking snapshot try deleting a secret to see the dfrnce after restore.

**To restore vault data use the following command:**

**Using cli:**

vault operator raft snapshot restore demo.snapshot

**Using API:**

curl \

--header "X-Vault-Token:s.8wghothW3bmw5FRm0mgcawgA" \

--request POST \

--data-binary @raft.snap \

http://52.90.88.15:8200/v1/sys/storage/raft/snapshot

# VAULT-GITLAB INTEGRATION

Each job in gitlab has JSON Web Token (JWT) provided as environment variable named CI\_JOB\_JWT. This JWT can be used to authenticate with Vault using the [JWT Auth](https://www.vaultproject.io/docs/auth/jwt#jwt-authentication) method.

## ENABLE JWT AUTH METHOD

## USING CLI

**Enable jwt auth:**

vault auth enable jwt

**Creating a policy:**

create a policy by giving access to those “paths with secrets” that you want to access in your gitlab pipeline.

For eg:

I want to read my kv secrets in a gitlab pipeline so the policy can be as follows:

vault policy write myproject-staging - <<EOF

# Policy name: myproject-staging

#

# Read-only permission on 'secret/data/myproject/staging/\*' path

path "secret/\*" {

capabilities = [ "read" ]

}

EOF

If you want to have access to aws secrets, db secrets and kv secrets then the policy can be as follows:

vault policy write myproject-staging - <<EOF

path "secret/\*" {

capabilities = [ "read","list" ]

}

path "aws/\*"{

capabilities = [ "read" , "list","create","update","delete" ]

}

path "database/\*"{

capabilities = [ "read" , "list","create","update","delete" ]

}

EOF

**Create a Role in jwt auth :**

vault write auth/jwt/role/myproject-staging - <<EOF

{

"role\_type": "jwt",

"policies": ["myproject-staging"]

"token\_explicit\_max\_ttl": 900,

"user\_claim": "user\_email",

"bound\_claims": {

"project\_id": "22372964",

"ref": "master",

"ref\_type": "branch"

}

}

EOF

**Now, configure the JWT Authentication method:**

**vault write auth/jwt/config \**

**jwks\_url="https://gitlab.com/-/jwks" \**

**bound\_issuer="gitlab.com"**

[bound\_issuer](https://www.vaultproject.io/api/auth/jwt#inlinecode-bound_issuer) specifies that only a JWT with the issuer set to gitlab.com can use this method to authenticate, and that the JWKS endpoint (https://gitlab.com/-/jwks) should be used to validate the token.

**USING CURL:**

**Enable jwt auth method:**

curl \

--header "X-Vault-Token: **s.IZeqoi10Pez65z1hEqHvbiWb**" \

--request POST \

--data '{"type": "jwt"}' \

<http://52.90.88.15:8200/v1/sys/auth/newjwt>

**Creating a policy:**

cat gitlabpolicy.json

{

"policy":"path \"secret/\*\" {\n capabilities = [ \"read\",\"list\" ]\n}\npath \"aws/\*\"{\n capabilities = [ \"read\" , \"list\",\"create\",\"update\",\"delete\" ]\n}\npath \"database/\*\"{\n capabilities = [ \"read\" , \"list\",\"create\",\"update\",\"delete\" ]\n}"

}

curl --header "X-Vault-Token: s.IZeqoi10Pez65z1hEqHvbiWb" --request PUT --data @gitlabpolicy.json <http://52.90.88.15:8200/v1/sys/policy/gitlabpolicy>

**Creating Role:**

**jwtrole.json**

{

"role\_type": "jwt",

"policies": ["gitlabpolicy],

"token\_explicit\_max\_ttl": 900,

"user\_claim": "user\_email",

"bound\_claims": {

"project\_id": "22372964",

"ref": "master",

"ref\_type": "branch"

}

curl --header "X-Vault-Token: s.IZeqoi10Pez65z1hEqHvbiWb" --request POST --data @jwtrole.json http://52.90.88.15:8200/v1/auth/newjwt/role/devrole

**Configure jwt auth :**

**jwtconfig.json**

{

"jwks\_url": "https://gitlab.com/-/jwks" ,

"bound\_issuer":"gitlab.com"

}

curl --header "X-Vault-Token: s.IZeqoi10Pez65z1hEqHvbiWb" --request POST --data @jwtconfig.json http://52.90.88.15:8200/v1/auth/newjwt/config

## Pipeline script to read dynamic AWS credentials and database secrets from vault

image: docker:latest

services:

- mysql:latest

read\_secrets:

before\_script:

- apk add --update curl && rm -rf /var/cache/apk/\*

- apk add jq

- apk add --update py3-pip

- pip install awscli

- apk add --update mysql-client && rm -rf /var/cache/apk/\*

script:

# Check job's ref name

- echo $CI\_COMMIT\_REF\_NAME

# and is this ref protected

- echo $CI\_COMMIT\_REF\_PROTECTED

# Vault's address can be provided here or as CI variable

- export VAULT\_ADDR=http://52.90.88.15:8200/

# Authenticate and get token. Token expiry time and other properties can be configured

# when configuring JWT Auth - https://www.vaultproject.io/api/auth/jwt#parameters-1

- "export VAULT\_LOGIN\_OUTPUT=$(curl --request POST --data '{\"role\": \"myproject-staging\", \"jwt\": \"'$CI\_JOB\_JWT'\"}' http://52.90.88.15:8200/v1/auth/jwt/login)"

- export VAULT\_TOKEN=$(echo $VAULT\_LOGIN\_OUTPUT | jq -r '.auth.client\_token')

- echo $VAULT\_TOKEN

# Now use the VAULT\_TOKEN to read the secret and store it in an environment variable

# AWS secrets

- "export AWS\_DATA=$(curl -H \"X-Vault-Token: $VAULT\_TOKEN\" http://52.90.88.15:8200/v1/aws/creds/s3user)"

# Use the secret

- export AWS\_ACCESS\_KEY\_ID=$(echo $AWS\_DATA | jq -r '.data.access\_key')

- export AWS\_SECRET\_ACCESS\_KEY=$(echo $AWS\_DATA | jq -r '.data.secret\_key')

- echo $AWS\_ACCESS\_KEY\_ID

- echo $AWS\_SECRET\_ACCESS\_KEY

# aws configuration

- aws configure set aws\_access\_key\_id $AWS\_ACCESS\_KEY\_ID

- aws configure set aws\_secret\_access\_key $AWS\_SECRET\_ACCESS\_KEY

- aws configure set region us-east-1

- sleep 60

# creating s3 bucket

- aws s3api create-bucket --bucket gitlabs3bucket2214 --region us-east-1

- sleep 10

# database secrets

- echo "$VAULT\_TOKEN"

- "export DB\_DATA=$(curl -H \"X-Vault-Token: $VAULT\_TOKEN\" http://52.90.88.15:8200/v1/database/creds/mygitlabrole)"

- echo $DB\_DATA

# use db dynamic secrets

- export MYSQL\_ROOT\_USER=$(echo $DB\_DATA | jq -r '.data.username')

- echo $MYSQL\_ROOT\_USER

- export MYSQL\_ROOT\_PASSWORD=$(echo $DB\_DATA | jq -r '.data.password')

- echo $MYSQL\_ROOT\_PASSWORD

- mysql --version

- sleep 20

# connecting to db

- echo "SHOW databases;"| mysql -u"$MYSQL\_ROOT\_USER" -p"$MYSQL\_ROOT\_PASSWORD" -h 18.208.245.31 -P 3306

**Logs for a successful job:**

Running with gitlab-runner 13.6.0-rc1 (d83ac56c)

[2](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L2) on docker-auto-scale 72989761

[3](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L3)**Preparing the "docker+machine" executor**

01:00

[4](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L4)Using Docker executor with image docker:latest ...

[5](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L5)Starting service mysql:latest ...

[6](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L6)Pulling docker image mysql:latest ...

[7](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L7)Using docker image sha256:db2b37ec6181ee1f367363432f841bf3819d4a9f61d26e42ac16e5bd7ff2ec18 for mysql:latest with digest mysql@sha256:8c17271df53ee3b843d6e16d46cff13f22c9c04d6982eb15a9a47bd5c9ac7e2d ...

[8](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L8)Waiting for services to be up and running...

[9](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L9)\*\*\* WARNING: Service runner-72989761-project-22372964-concurrent-0-6172423d8949e70d-mysql-0 probably didn't start properly.

[10](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L10)Health check error:

[11](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L11)service "runner-72989761-project-22372964-concurrent-0-6172423d8949e70d-mysql-0-wait-for-service" timeout

[12](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L12)Health check container logs:

[13](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L13)Service container logs:

[14](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L14)2020-11-13T06:41:05.600899810Z 2020-11-13 06:41:05+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.22-1debian10 started.

[15](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L15)2020-11-13T06:41:05.777005492Z 2020-11-13 06:41:05+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'

[16](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L16)2020-11-13T06:41:05.821587897Z 2020-11-13 06:41:05+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.22-1debian10 started.

[17](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L17)2020-11-13T06:41:05.950492548Z 2020-11-13 06:41:05+00:00 [ERROR] [Entrypoint]: Database is uninitialized and password option is not specified

[18](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L18)2020-11-13T06:41:05.952034364Z You need to specify one of MYSQL\_ROOT\_PASSWORD, MYSQL\_ALLOW\_EMPTY\_PASSWORD and MYSQL\_RANDOM\_ROOT\_PASSWORD

[19](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L19)\*\*\*\*\*\*\*\*\*

[20](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L20)Pulling docker image docker:latest ...

[21](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L21)Using docker image sha256:6972c414f322dfa40324df3c503d4b217ccdec6d576e408ed10437f508f4181b for docker:latest with digest docker@sha256:9170b902404a630a982a2a6473442d3e2cc2342b66345f7a9cf148f8affcf5d3 ...

[23](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L23)**Preparing environment**

00:02

[24](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L24)Running on runner-72989761-project-22372964-concurrent-0 via runner-72989761-srm-1605249597-a23d82e5...

[26](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L26)**Getting source from Git repository**

00:01

[27](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L27)**$ eval "$CI\_PRE\_CLONE\_SCRIPT"**

[28](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L28)**Fetching changes with git depth set to 50...**

[29](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L29)Initialized empty Git repository in /builds/mindtree3/vault/.git/

[30](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L30)**Created fresh repository.**

[31](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L31)**Checking out fbaa2cc1 as master...**

[32](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L32)**Skipping Git submodules setup**

[34](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L34)**Executing "step\_script" stage of the job script**

01:47

[35](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L35)**$ apk add --update curl && rm -rf /var/cache/apk/\***

[36](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L36)fetch http://dl-cdn.alpinelinux.org/alpine/v3.12/main/x86\_64/APKINDEX.tar.gz

[37](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L37)fetch http://dl-cdn.alpinelinux.org/alpine/v3.12/community/x86\_64/APKINDEX.tar.gz

[38](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L38)(1/3) Installing nghttp2-libs (1.41.0-r0)

[39](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L39)(2/3) Installing libcurl (7.69.1-r1)

[40](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L40)(3/3) Installing curl (7.69.1-r1)

[41](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L41)Executing busybox-1.31.1-r19.trigger

[42](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L42)OK: 12 MiB in 23 packages

[43](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L43)**$ apk add jq**

[44](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L44)fetch http://dl-cdn.alpinelinux.org/alpine/v3.12/main/x86\_64/APKINDEX.tar.gz

[45](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L45)fetch http://dl-cdn.alpinelinux.org/alpine/v3.12/community/x86\_64/APKINDEX.tar.gz

[46](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L46)(1/2) Installing oniguruma (6.9.5-r2)

[47](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L47)(2/2) Installing jq (1.6-r1)

[48](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L48)Executing busybox-1.31.1-r19.trigger

[49](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L49)OK: 13 MiB in 25 packages

[50](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L50)**$ apk add --update py3-pip**

[51](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L51)(1/34) Installing libbz2 (1.0.8-r1)

[52](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L52)(2/34) Installing expat (2.2.9-r1)

[53](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L53)(3/34) Installing libffi (3.3-r2)

[54](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L54)(4/34) Installing gdbm (1.13-r1)

[55](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L55)(5/34) Installing xz-libs (5.2.5-r0)

[56](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L56)(6/34) Installing readline (8.0.4-r0)

[57](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L57)(7/34) Installing sqlite-libs (3.32.1-r0)

[58](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L58)(8/34) Installing python3 (3.8.5-r0)

[59](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L59)(9/34) Installing py3-appdirs (1.4.4-r1)

[60](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L60)(10/34) Installing py3-ordered-set (4.0.1-r0)

[61](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L61)(11/34) Installing py3-parsing (2.4.7-r0)

[62](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L62)(12/34) Installing py3-six (1.15.0-r0)

[63](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L63)(13/34) Installing py3-packaging (20.4-r0)

[64](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L64)(14/34) Installing py3-setuptools (47.0.0-r0)

[65](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L65)(15/34) Installing py3-chardet (3.0.4-r4)

[66](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L66)(16/34) Installing py3-idna (2.9-r0)

[67](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L67)(17/34) Installing py3-certifi (2020.4.5.1-r0)

[68](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L68)(18/34) Installing py3-urllib3 (1.25.9-r0)

[69](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L69)(19/34) Installing py3-requests (2.23.0-r0)

[70](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L70)(20/34) Installing py3-msgpack (1.0.0-r0)

[71](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L71)(21/34) Installing py3-lockfile (0.12.2-r3)

[72](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L72)(22/34) Installing py3-cachecontrol (0.12.6-r0)

[73](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L73)(23/34) Installing py3-colorama (0.4.3-r0)

[74](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L74)(24/34) Installing py3-distlib (0.3.0-r0)

[75](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L75)(25/34) Installing py3-distro (1.5.0-r1)

[76](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L76)(26/34) Installing py3-webencodings (0.5.1-r3)

[77](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L77)(27/34) Installing py3-html5lib (1.0.1-r4)

[78](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L78)(28/34) Installing py3-pytoml (0.1.21-r0)

[79](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L79)(29/34) Installing py3-pep517 (0.8.2-r0)

[80](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L80)(30/34) Installing py3-progress (1.5-r0)

[81](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L81)(31/34) Installing py3-toml (0.10.1-r0)

[82](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L82)(32/34) Installing py3-retrying (1.3.3-r0)

[83](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L83)(33/34) Installing py3-contextlib2 (0.6.0-r0)

[84](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L84)(34/34) Installing py3-pip (20.1.1-r0)

[85](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L85)Executing busybox-1.31.1-r19.trigger

[86](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L86)OK: 72 MiB in 59 packages

[87](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L87)**$ pip install awscli**

[88](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L88)Collecting awscli

[89](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L89) Downloading awscli-1.18.177-py2.py3-none-any.whl (3.4 MB)

[90](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L90)Collecting rsa<=4.5.0,>=3.1.2; python\_version != "3.4"

[91](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L91) Downloading rsa-4.5-py2.py3-none-any.whl (36 kB)

[92](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L92)Collecting PyYAML<5.4,>=3.10; python\_version != "3.4"

[93](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L93) Downloading PyYAML-5.3.1.tar.gz (269 kB)

[94](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L94)Collecting s3transfer<0.4.0,>=0.3.0

[95](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L95) Downloading s3transfer-0.3.3-py2.py3-none-any.whl (69 kB)

[96](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L96)Requirement already satisfied: colorama<0.4.4,>=0.2.5; python\_version != "3.4" in /usr/lib/python3.8/site-packages (from awscli) (0.4.3)

[97](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L97)Collecting docutils<0.16,>=0.10

[98](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L98) Downloading docutils-0.15.2-py3-none-any.whl (547 kB)

[99](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L99)Collecting botocore==1.19.17

[100](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L100) Downloading botocore-1.19.17-py2.py3-none-any.whl (6.8 MB)

[101](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L101)Collecting pyasn1>=0.1.3

[102](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L102) Downloading pyasn1-0.4.8-py2.py3-none-any.whl (77 kB)

[103](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L103)Collecting jmespath<1.0.0,>=0.7.1

[104](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L104) Downloading jmespath-0.10.0-py2.py3-none-any.whl (24 kB)

[105](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L105)Requirement already satisfied: urllib3<1.27,>=1.25.4; python\_version != "3.4" in /usr/lib/python3.8/site-packages (from botocore==1.19.17->awscli) (1.25.9)

[106](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L106)Collecting python-dateutil<3.0.0,>=2.1

[107](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L107) Downloading python\_dateutil-2.8.1-py2.py3-none-any.whl (227 kB)

[108](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L108)Requirement already satisfied: six>=1.5 in /usr/lib/python3.8/site-packages (from python-dateutil<3.0.0,>=2.1->botocore==1.19.17->awscli) (1.15.0)

[109](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L109)Using legacy setup.py install for PyYAML, since package 'wheel' is not installed.

[110](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L110)Installing collected packages: pyasn1, rsa, PyYAML, jmespath, python-dateutil, botocore, s3transfer, docutils, awscli

[111](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L111) Running setup.py install for PyYAML: started

[112](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L112) Running setup.py install for PyYAML: finished with status 'done'

[113](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L113)Successfully installed PyYAML-5.3.1 awscli-1.18.177 botocore-1.19.17 docutils-0.15.2 jmespath-0.10.0 pyasn1-0.4.8 python-dateutil-2.8.1 rsa-4.5 s3transfer-0.3.3

[114](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L114)**$ apk add --update mysql-client && rm -rf /var/cache/apk/\***

[115](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L115)(1/5) Installing mariadb-common (10.4.15-r0)

[116](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L116)(2/5) Installing libgcc (9.3.0-r2)

[117](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L117)(3/5) Installing libstdc++ (9.3.0-r2)

[118](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L118)(4/5) Installing mariadb-client (10.4.15-r0)

[119](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L119)(5/5) Installing mysql-client (10.4.15-r0)

[120](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L120)Executing busybox-1.31.1-r19.trigger

[121](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L121)OK: 104 MiB in 64 packages

[122](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L122)**$ echo $CI\_COMMIT\_REF\_NAME**

[123](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L123)master

[124](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L124)**$ echo $CI\_COMMIT\_REF\_PROTECTED**

[125](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L125)true

[126](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L126)**$ export VAULT\_ADDR=http://52.90.88.15:8200/**

[127](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L127)**$ export VAULT\_LOGIN\_OUTPUT=$(curl --request POST --data '{"role": "myproject-staging", "jwt": "'$CI\_JOB\_JWT'"}' http://52.90.88.15:8200/v1/auth/jwt/login)**

[128](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L128) % Total % Received % Xferd Average Speed Time Time Time Current

[129](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L129) Dload Upload Total Spent Left Speed

[130](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L130)100 1719 100 635 100 1084 14767 25209 --:--:-- --:--:-- --:--:-- 40928

[131](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L131)**$ export VAULT\_TOKEN=$(echo $VAULT\_LOGIN\_OUTPUT | jq -r '.auth.client\_token')**

[132](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L132)**$ echo $VAULT\_TOKEN**

[133](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L133)s.9qbISFZu0WBB9ys29Abu83FZ

[134](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L134)**$ export AWS\_DATA=$(curl -H "X-Vault-Token: $VAULT\_TOKEN" http://52.90.88.15:8200/v1/aws/creds/s3user)**

[135](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L135) % Total % Received % Xferd Average Speed Time Time Time Current

[136](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L136) Dload Upload Total Spent Left Speed

[137](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L137)100 316 100 316 0 0 1067 0 --:--:-- --:--:-- --:--:-- 1063

[138](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L138)**$ export AWS\_ACCESS\_KEY\_ID=$(echo $AWS\_DATA | jq -r '.data.access\_key')**

[139](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L139)**$ export AWS\_SECRET\_ACCESS\_KEY=$(echo $AWS\_DATA | jq -r '.data.secret\_key')**

[140](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L140)**$ echo $AWS\_ACCESS\_KEY\_ID**

[141](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L141)AKIA4OJTSXOYCVDK5GW2

[142](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L142)**$ echo $AWS\_SECRET\_ACCESS\_KEY**

[143](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L143)PmHVnKfz/dePnTh7YEymTTk9dse4CnYUhOAj0Emg

[144](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L144)**$ aws configure set aws\_access\_key\_id $AWS\_ACCESS\_KEY\_ID**

[145](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L145)**$ aws configure set aws\_secret\_access\_key $AWS\_SECRET\_ACCESS\_KEY**

[146](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L146)**$ aws configure set region us-east-1**

[147](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L147)**$ sleep 60**

[148](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L148)**$ aws s3api create-bucket --bucket gitlabs3bucket2214 --region us-east-1**

[149](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L149){

[150](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L150) "Location": "/gitlabs3bucket2214"

[151](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L151)}

[152](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L152)**$ sleep 10**

[153](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L153)**$ echo "$VAULT\_TOKEN"**

[154](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L154)s.9qbISFZu0WBB9ys29Abu83FZ

[155](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L155)**$ export DB\_DATA=$(curl -H "X-Vault-Token: $VAULT\_TOKEN" http://52.90.88.15:8200/v1/database/creds/mygitlabrole)**

[156](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L156) % Total % Received % Xferd Average Speed Time Time Time Current

[157](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L157) Dload Upload Total Spent Left Speed

[158](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L158)100 293 100 293 0 0 5528 0 --:--:-- --:--:-- --:--:-- 5528

[159](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L159)**$ echo $DB\_DATA**

[160](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L160){"request\_id":"1a9fbd9b-72b0-f94c-7758-97b7b5dec154","lease\_id":"database/creds/mygitlabrole/wWmEb4VzZ3yml6Jzfm30GxZr","renewable":true,"lease\_duration":3600,"data":{"password":"A1a-kCYwGFqwNKBBeyqi","username":"v-jwt-lahari-mygitlabro-0GVnyTOp"},"wrap\_info":null,"warnings":null,"auth":null}

[161](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L161)**$ export MYSQL\_ROOT\_USER=$(echo $DB\_DATA | jq -r '.data.username')**

[162](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L162)**$ echo $MYSQL\_ROOT\_USER**

[163](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L163)v-jwt-lahari-mygitlabro-0GVnyTOp

[164](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L164)**$ export MYSQL\_ROOT\_PASSWORD=$(echo $DB\_DATA | jq -r '.data.password')**

[165](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L165)**$ echo $MYSQL\_ROOT\_PASSWORD**

[166](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L166)A1a-kCYwGFqwNKBBeyqi

[167](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L167)**$ mysql --version**

[168](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L168)mysql Ver 15.1 Distrib 10.4.15-MariaDB, for Linux (x86\_64) using readline 5.1

[169](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L169)**$ sleep 20**

[170](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L170)**$ echo "SHOW databases;"| mysql -u"$MYSQL\_ROOT\_USER" -p"$MYSQL\_ROOT\_PASSWORD" -h 18.208.245.31 -P 3306**

[171](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L171)Database

[172](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L172)gitlabdb

[173](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L173)information\_schema

[174](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L174)mysql

[175](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L175)performance\_schema

[176](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L176)sys

[178](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L178)**Cleaning up file based variables**

00:01

[180](https://gitlab.com/mindtree3/vault/-/jobs/847209303#L180)**Job succeeded**