Vault

Implementation Foundations

Module 11: Authentication Methods

What You Will Learn

Authentication Overview

People Auth Methods

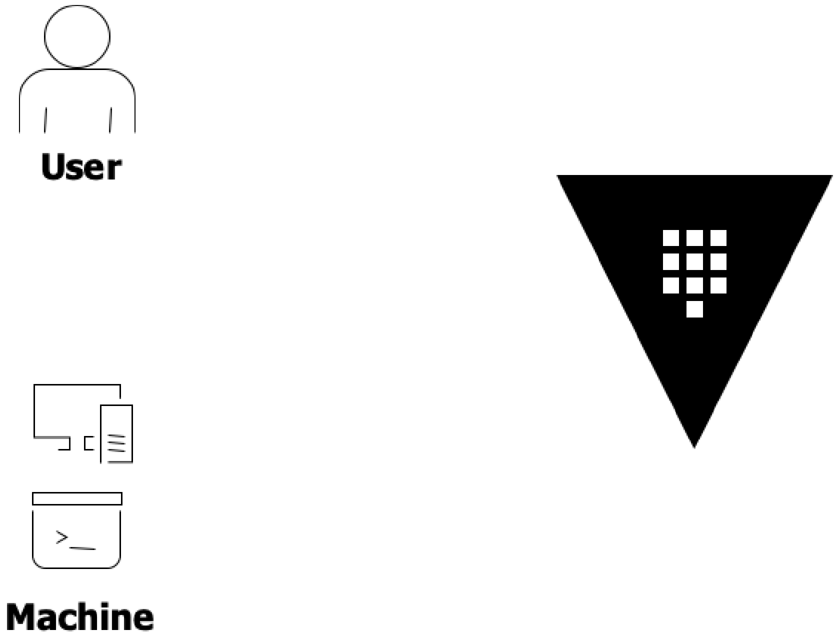
- LDAP
- OIDC

Machine Auth Methods

- Cloud Machine ID
- AppRole
- JWT (Kubernetes)

Authentication Method Overview

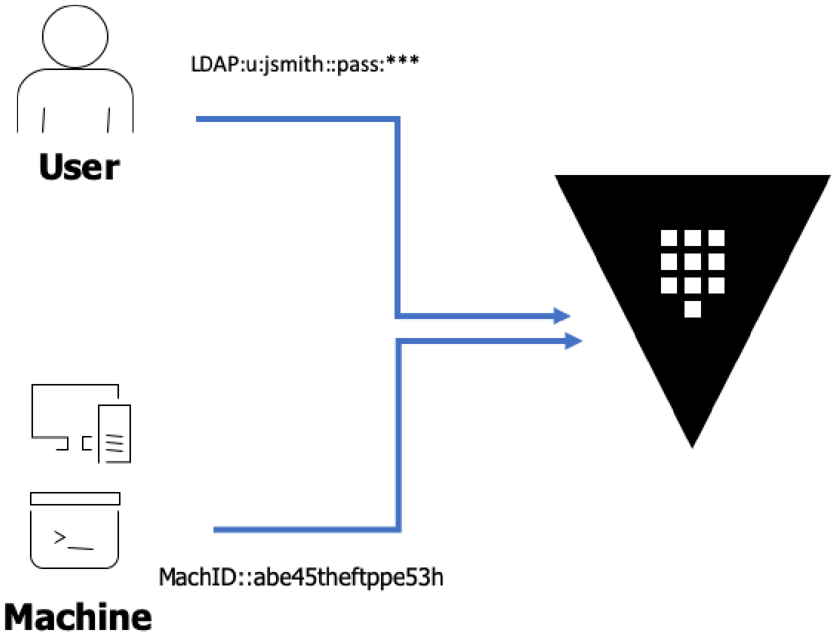
Auth Methods Overview - Workflow



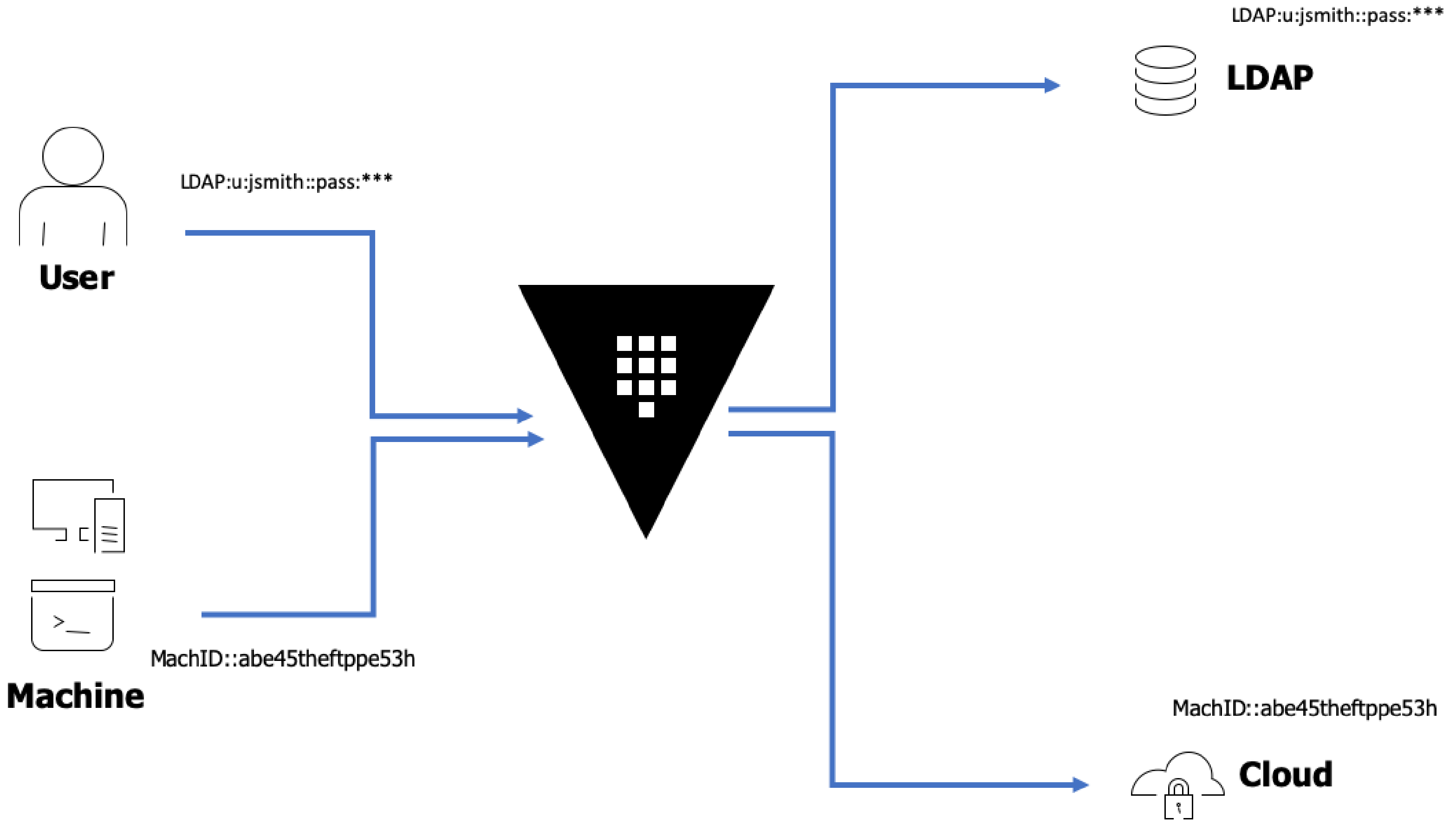
User

Machine

Auth Methods Overview - Workflow (1/2)



Auth Methods Overview - Workflow (2/2)



Authentication Types

People Methods

Method	Description
User/Pass	Generic Username and Password stored locally in Vault
GitHub	GitHub personal token
LDAP	LDAP or AD lookup
OKTA	OKTA Single Sign On

Machine Methods

Method	Description
Token	Generic Vault Token stored locally in Vault
JWT/OIDC	JSON Web Token or Open ID Connect Token
Cloud Machine ID	Cloud Specific Machine ID Lookup
TLS Cert	PKI Cert issued to a machine

Username & Password

People Method - User/Pass

Vault provides a simple username and password authentication method.

Characteristics

- Locally stored and managed
- Replication across performance replication link
- This should not be used for production

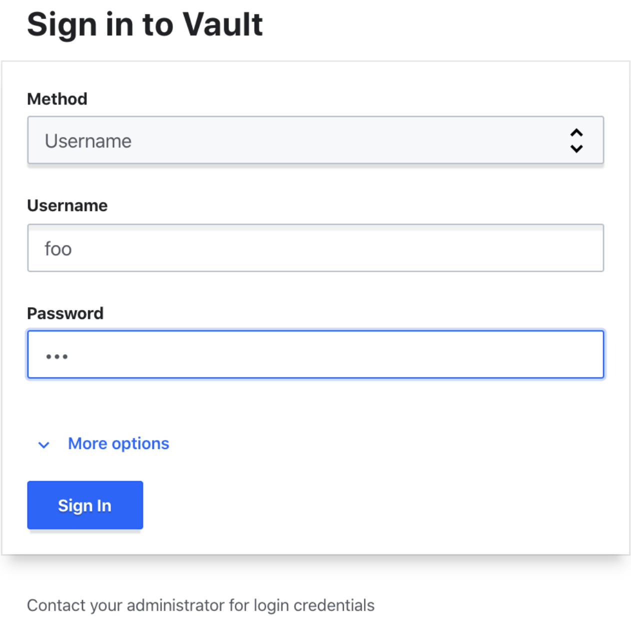
User/Pass Setup

Setup Steps:

```
$ vault write auth/userpass/users/mitchellh \
    password=foo \
    policies=admins
```

Directly link a user to a policy by specifying the policies flag. This policy will be bound to the token

UserPass - Command Line vs. UI



Sign in to Vault

Method: Username

Username: foo

Password: •••

Contact your administrator for login credentials

```
$ vault login -method=userpass \
    username=foo \
    password=bar
```

Additional Notes

- While this is a valid secret engine please only use this for testing
- DO NOT USE THIS FOR PRODUCTION

LDAP

LDAP Auth Setup - User Binding

The LDAP authentication method covers both standard LDAP implementations and Active Directory authentication.

Binding Attributes:

Vault Attribute	Description
binddn	Distinguished name of object to bind when performing user and group search.
bindpass	Password to use along with binddn when performing user search.
userdn	Base DN under which to perform user search.
userattr	Attribute on user attribute object matching the username passed when authenticating.

LDAP - User Lookup Via Group

Most of the time you will do a look up of a user via a group DN

Grouping Attributes:

Vault Attribute	Description
binddn	Distinguished name of object to bind when performing user and group search.
bindpass	Password to use along with binddn when performing user search.
userdn	Base DN under which to perform user search.
userattr	Attribute on user attribute object matching the username passed when authenticating.

LDAP Auth Setup - AD Example

```
$ vault write auth/ldap/config \
    url="ldap://ldap.example.com" \
    userdn="ou=Users,dc=example,dc=com" \
    groupdn="ou=Groups,dc=example,dc=com" \
    groupfilter="(&(objectClass=group)(member:1.2.840.113556.1.4.1941:={{.UserDN}}))" \
    groupattr="cn" \
    upndomain="example.com" \
    certificate=@ldap_ca_cert.pem \
    insecure_tls=false \
    starttls=true
```

This is the filter lookup to get the user and group information during authentication

LDAP Auth Setup - AD Example

```
$ vault write auth/ldap/config \
    url="ldap://ldap.example.com" \
    userdn="ou=Users,dc=example,dc=com" \
    groupdn="ou=Groups,dc=example,dc=com" \
    groupfilter="(&(objectClass=group)(member:1.2.840.113556.1.4.1941:={{.UserDN}}))" \
    groupattr="cn" \
    upndomain="example.com" \
    certificate=@ldap_ca_cert.pem \
    insecure_tls=false \
    starttls=true
```

This is the filter used to locate the userDN in a group. Notice the {{.UserDN}} this is the way vault injects data.

Notice the "member:1.2.840.113556.1.4.1941", this is needed when looking up via Active Directory

LDAP Auth Setup - LDAP Example

```
$ vault write auth/ldap/config \
    url="ldap://ldap.example.com" \
    userattr=sAMAccountName \
    userdn="ou=Users,dc=example,dc=com" \
    groupdn="ou=Users,dc=example,dc=com" \
    groupfilter="(&(objectClass=person)(uid={{.Username}}))" \
    groupattr="memberOf" \
    binddn="cn=vault,ou=users,dc=example,dc=com" \
    bindpass='My$ecrt3tP4ss' \
    certificate=@ldap_ca_cert.pem \
    insecure_tls=false \
    starttls=true
```

Notice the change in the lookup. There are two ways of looking up users in a group. In this example the {{.Username}} is used to inject the username for the search

LDAP Auth Setup - LDAP Example

```
$ vault write auth/ldap/config \
    url="ldap://ldap.example.com" \
    userattr=sAMAccountName \
    userdn="ou=Users,dc=example,dc=com" \
    groupdn="ou=Users,dc=example,dc=com" \
    groupfilter="(&(objectClass=person)(uid={{.Username}}))" \
    groupattr="memberOf" \
    binddn="cn=vault,ou=users,dc=example,dc=com" \
    bindpass='My$ecrt3tP4ss' \
    certificate=@ldap_ca_cert.pem \
    insecure_tls=false \
    starttls=true
```

These are the bind attributes that vault will use to lookup and authenticate ldap users. This must have the right level of permission)

LDAP Auth Setup

Vault Policy Mapping

```
$ vault write auth/ldap/groups/scientists policies=readOnly, writeOnly

$ vault write auth/ldap/users/intern groups=scientists policies=readOnly
```

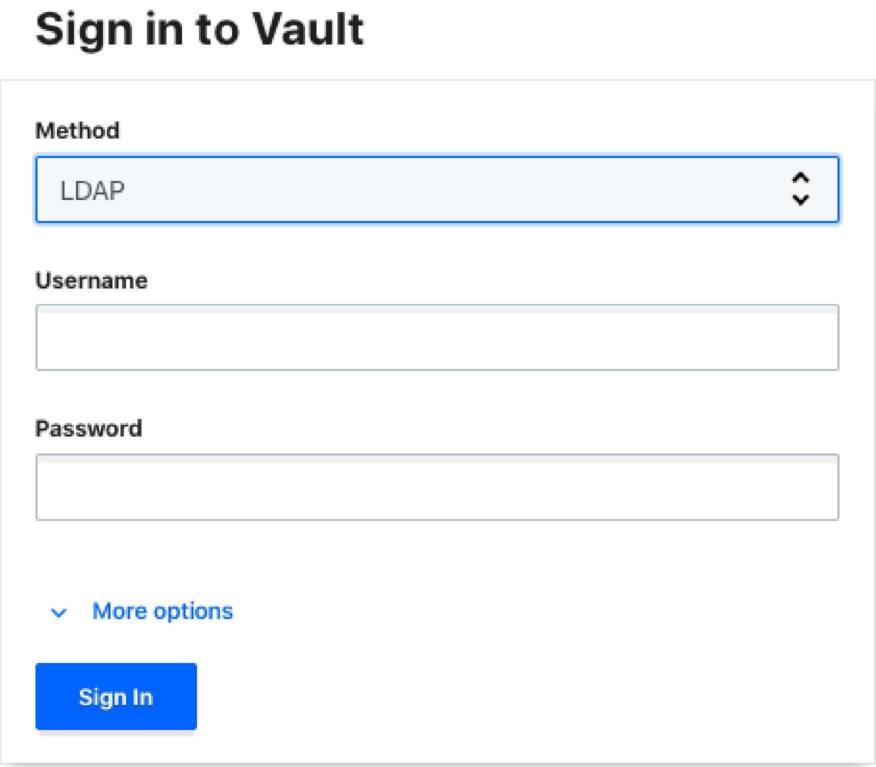
Once a connection has been configured policies have to be mapped to the LDAP group to link the user to a vault policy and issue a vault token with that policy's capabilities.

Notice you can link multiple policies to a group.

You can also link policies to specific users in a group.

Remember that policies are additive so the user intern would inherit the group policies.

LDAP Auth Setup – Authenticating



Sign in to Vault

Method: LDAP

CLI:

```
$ vault login -method=ldap \
username=intern
Password (will be hidden):
Successfully authenticated!
The policies that are associated
with this token are listed below:

default, readOnly, writeOnly
```

Additional Notes

- Know the LDAP schema and environment
  - URL endpoints
  - Regional locations
  - Bind credentials
  - TLS Cert requirement
- LDAP Search utilities
  - Test Bind credentials
  - Schema attributes
  - Query Optimization

Cloud Identity Auth Methods

Authentication via Cloud Machine ID

Each cloud provider gives a mechanism for machine entities to have a unique ID that can be validated via an API call.

Each cloud has it own method of how to achieve this but the workflow is similar between them.

AWS Authentication

Authentication via AWS

Authentication with AWS is provided via two methods: IAM and EC2

- IAM is more flexible and future proofs your authentication
- EC2 only supports EC2 instances

AWS Authentication Types

IAM Method

- Preferred method as this supports the most types of instances
- AWS STS provides an API call called sts:GetCallerIdentity
- Method Supports ec2, lamda, containers
- Vault Agent is the preferred method of access due to the workflow process of signing identifying material
- The instance does not need to have access to the STS endpoint as vault acts as the proxy
- Does not support MFA

EC2 Method

- Only supported by EC2 instances
- Leverages EC2 metadata to authenticate
- Uses the signature data retrieved from the AWS Metadata service
- Vault validates this data with the Metadata service

Additional Notes

- You can manage authentication roles by using EC2 tags
  - This allows a single AMI type to have a dynamic role based on tag
  - This is to apply a subset of privileges
  - Using this leverages a vault generated HMAC role tag that is submitted along with the EC2 signature
- Use IAM over EC2 whenever possible
- You can only use one or the other per instance of a authentication method
  - Namespaces allow for multiple configurations while maintaining sane APIs and policies

Setup Vault with AWS

```
$ vault write auth/aws/config/client secret_key=vCtSM8ZUEQ3.... access_k
```

EC2 Authentication Setup

```
$ vault write auth/aws/role/dev-role auth_type=ec2 bound_ami_id=ami-fce3
              policies=prod,dev max_ttl=500h
```

IAM Authentication Setup

```
$ vault write auth/aws/role/dev-role-iam auth_type=iam \
              bound_iam_principal_arn=arn:aws:iam::123456789012:role/MyR
              policies=prod,dev max_ttl=500h
```

Azure Authentication

Authentication via Azure

Azure

The Azure authentication method allows authentication against Vault using Azure Active Directory credentials.

- Uses at JWT signed by Azure AD
- Validated with the Azure Managed Service Identity API

Vault must have permissions to:

- `Microsoft.Compute/virtualMachines/*/read`
- `Microsoft.Compute/virtualMachineScaleSet/*/read`

Setup Azure Auth – Configure Vault

```
$ vault write auth/azure/config \
    tenant_id= 7cd1f227-ca67-4fc6-a1a4-9888ea7f388c \
    resource=https://vault.hashicorp.com \
    client_id=dd794de4-4c6c-40b3-a930-d84cd32e9699 \
    client_secret=IT3B2XfZvWnfB98s1cie8EMe7zWg483Xy8zY004=
```

The account that Vault will use must have the permissions set to do successful lookups.

Setup Azure Auth - Bind Types

You can bind roles to specific azure resource types. You can have more than one.

- Service Principal IDs
- Group IDs
- Locations
- Subscription IDs
- Resource Groups
- Scale Sets

Setup Azure Auth - Configure Role

```
$ vault write auth/azure/role/dev-role \
    policies="prod,dev" \
    bound_subscription_ids=6a1d5988-5917-4221-b224-904cd7e24a25 \
    bound_resource_groups=vault
```

- Creating a Vault role
- Linking Vault policies to the role
- Binding the auth to a certian subscription and resource group

Setup Azure Auth - Authenticating

```
$ vault write auth/azure/login \
    role="dev-role" \
    jwt="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..." \
    subscription_id="12345-..." \
    resource_group_name="test-group" \
    vm_name="test-vm"
```

- The role and the JWT are required for login
- `subscription_id`, `resource_group_name` are the bind values that must be passed if they are set by the role

GCP Authentication

GCP Authentication

The gcp auth method allows Google Cloud Platform entities to authenticate to Vault. This backend allows for authentication of:

- Google Cloud IAM service accounts
- Google Compute Engine (GCE) instances

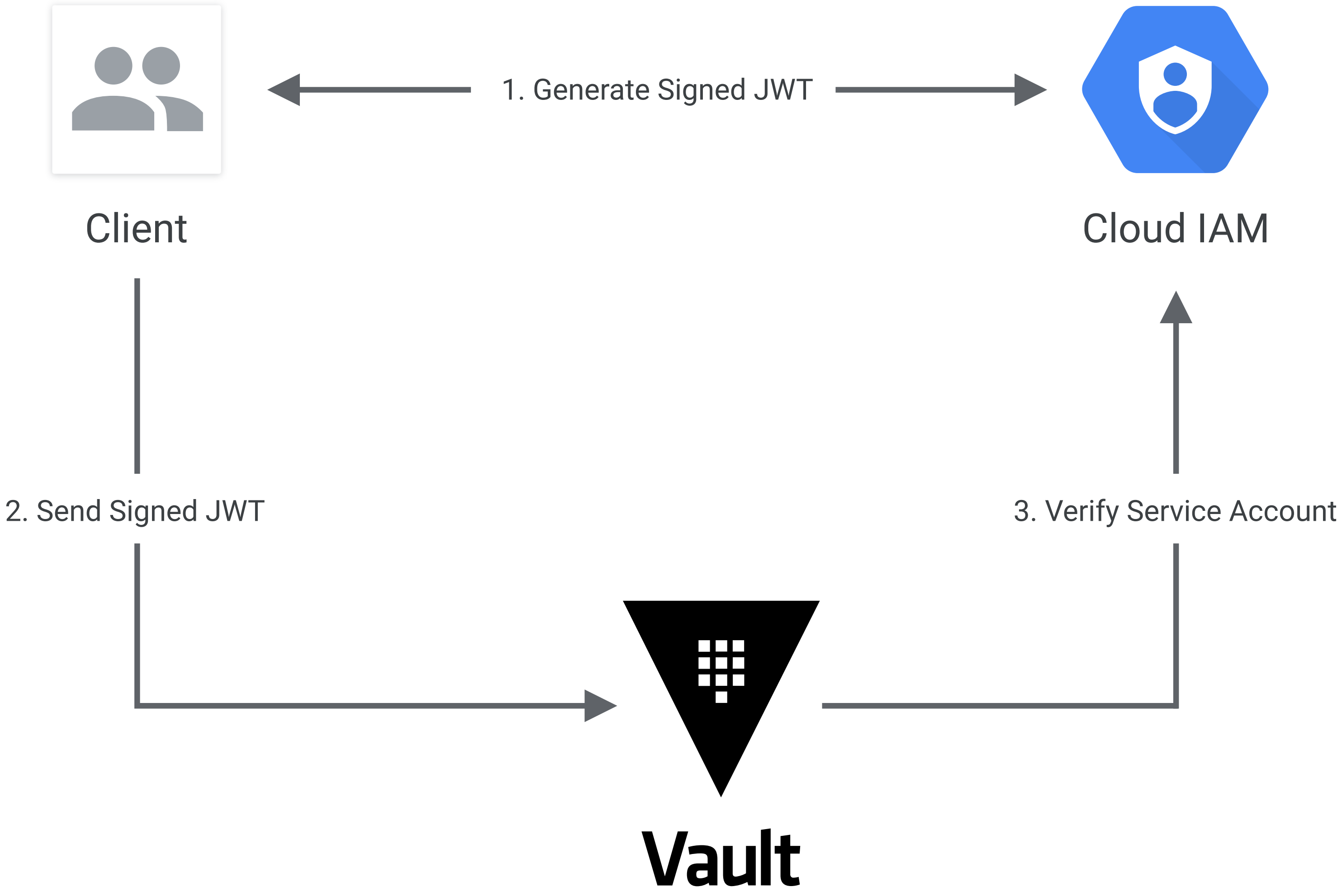
This backend focuses on identities specific to Google Cloud and does not support authenticating arbitrary Google or G Suite users or generic OAuth against Google.

Google Auth - Vault Configuration

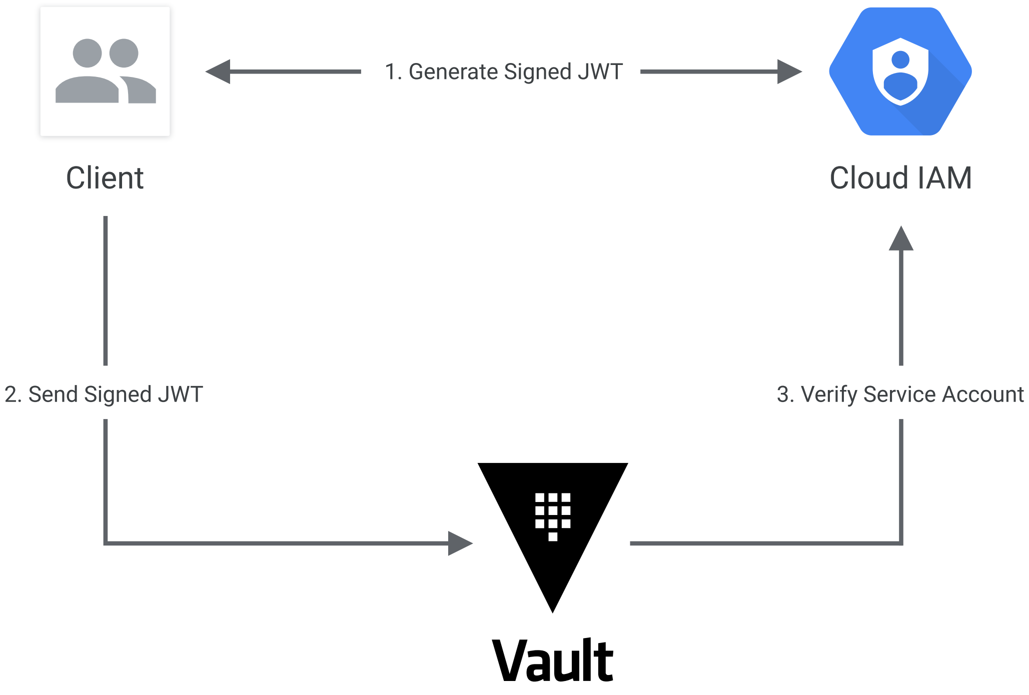
```
$ vault write auth/gcp/config \
        credentials=@creds.json
```

- Create a Vault Service account with the following roles
  - `roles/iam.serviceAccountKeyAdmin`
  - `roles/iam.serviceAccountTokenCreator`

Google Auth - IAM Workflow



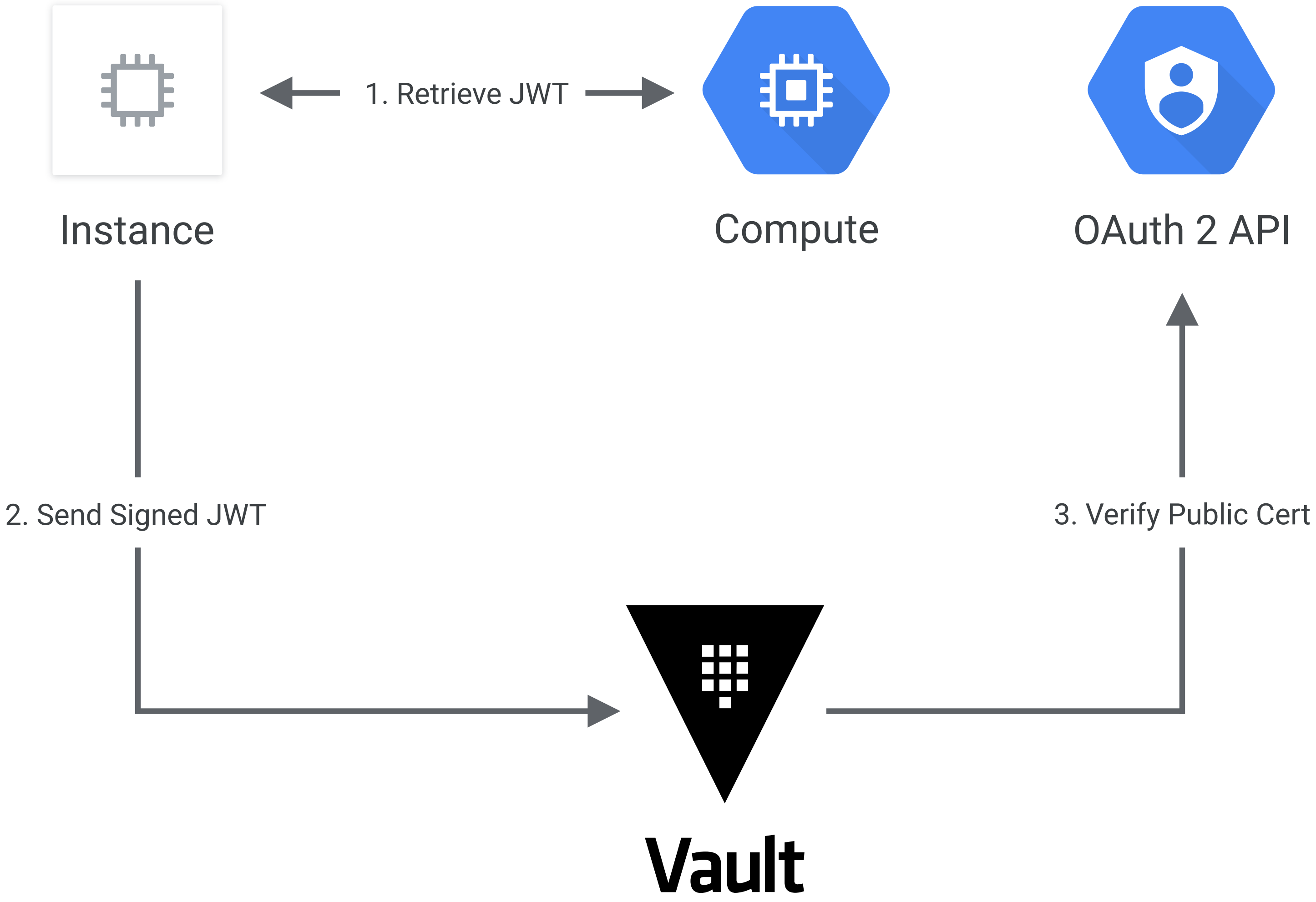
Google Auth - IAM Details



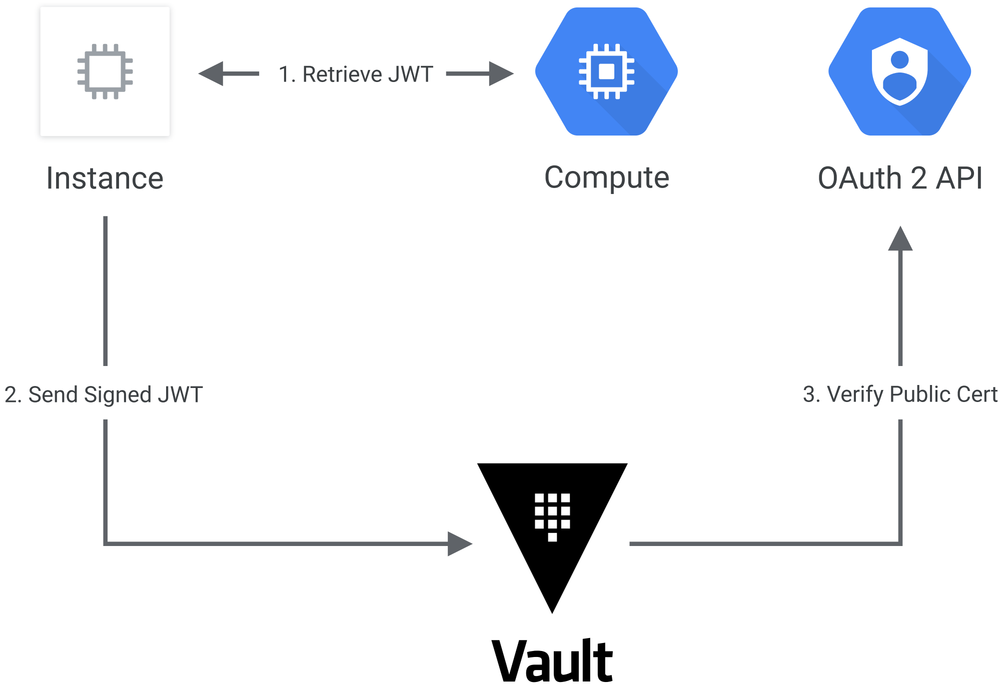
IAM Workflow:

- JWT token is generated by the client
- Client sends signed JWT to vault with role
- Vault inspects the JWT to ensure it is a valid GCP JWT
- Vault validates the token and issues a vault token

Google Auth - GCE Workflow



Google Auth - GCE Details



GCE Workflow

- Client obtains an instance identity metadata token
- Submits GCE JWT along with role name
- Vault inspects the JWT to ensure it is a valid GCP JWT
- Vault validates the token and issues a vault token

Google Auth - Role (IAM Type)

```
$ vault write auth/gcp/role/my-iam-role \
    type="iam" \
    policies="dev,prod" \
    bound_service_accounts="my-service@my-project.iam.gserviceaccount.co
```

- This is for the GCP IAM auth
- Note: `bound_service_accounts` is only required for iam-type roles

Google Auth - Login

CLI:

```
$ vault login -method=gcp role="vaultadmins" \
     credentials=@vault-tester.json \
     project="vault-auth-test" \
     bound_service_account="vault-tester@..."
```

Output:

```
Key                                  Value
--                                   --
token                                s.5tdfBZhRa5smZZPSBRZQDbAW
token_accessor                       aiMk8mLKFbCZbr5lcyPsMKHb
token_duration                       768h
token_renewable                      true
...                                  ...
```

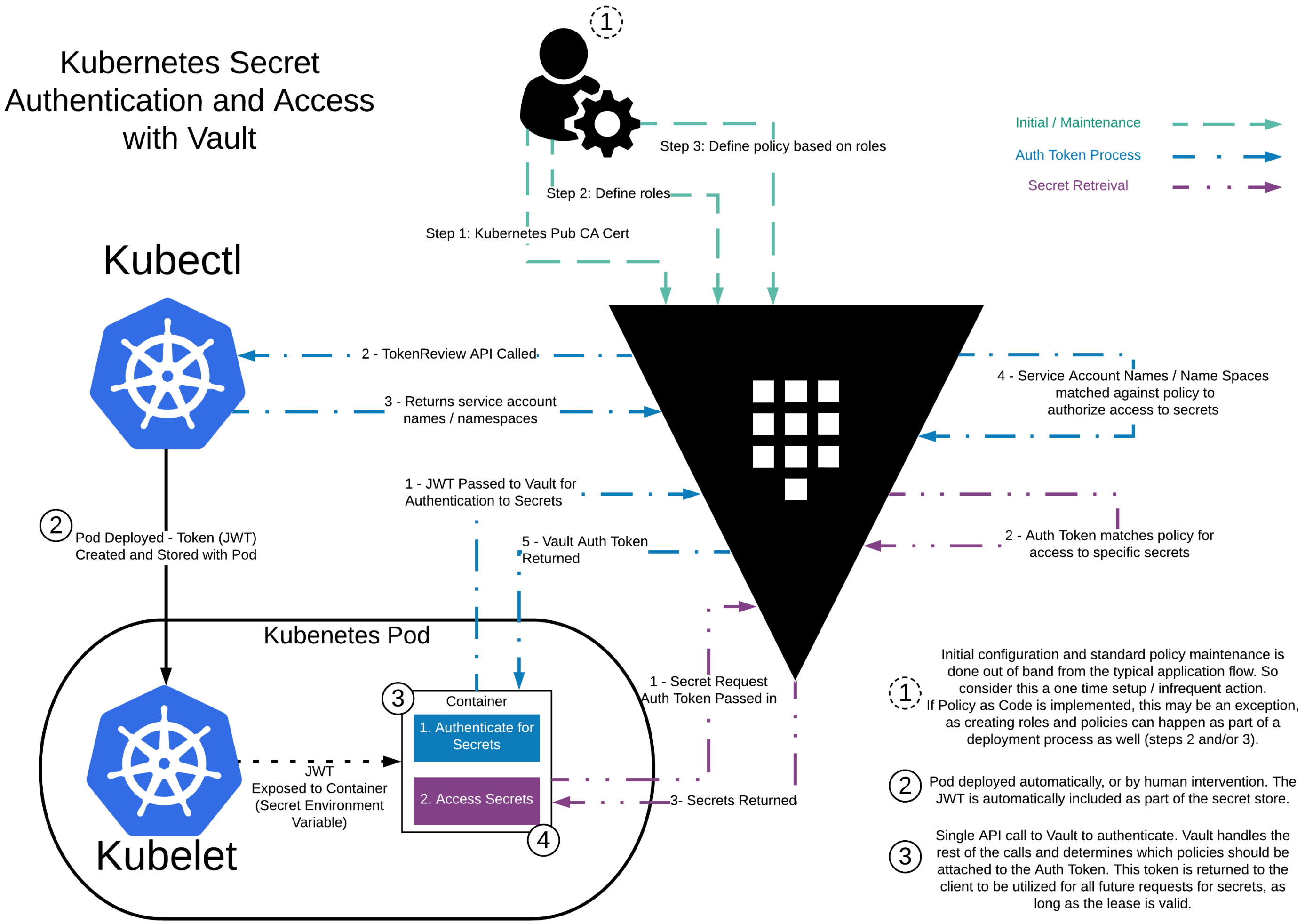
Google Auth - Role (GCE Type)

```
$ vault write auth/gcp/role/my-gce-role \
    type="gce" \
    policies="dev,prod" \
    bound_projects="my-project1,my-project2" \
    bound_zones="us-east1-b" \
    bound_labels="foo:bar,zip:zap" \
    bound_service_accounts="my-service@my-project.iam.gserviceaccount.co
```

- type `gce` is defined for this role
- Policies are bound to this role
- Just like Azure you can bind to specific attributes

Kubernetes

Kubernetes Workflow Overview



Kubernetes Secret Authentication and Access with Vault

Initial / Maintenance

Auth Token Process

Secret Retreival

1

Step 3: Define policy based on roles

Step 2: Define roles

Step 1: Kubernetes Pub CA Cert

Kubectl

2 - TokenReview API Called

3 - Returns service account names / namespaces

4 - Service Account Names / Name Spaces matched against policy to authorize access to secrets

1 - JWT Passed to Vault for Authentication to Secrets

2 Pod Deployed - Token (JWT) Created and Stored with Pod

5 - Vault Auth Token Returned

2 - Auth Token matches policy for access to specific secrets

Kubenetes Pod

1 - Secret Request Auth Token Passed in

3 Container

1. Authenticate for Secrets

2. Access Secrets

4

JWT Exposed to Container (Secret Environment Variable)

3- Secrets Returned

Kubelet

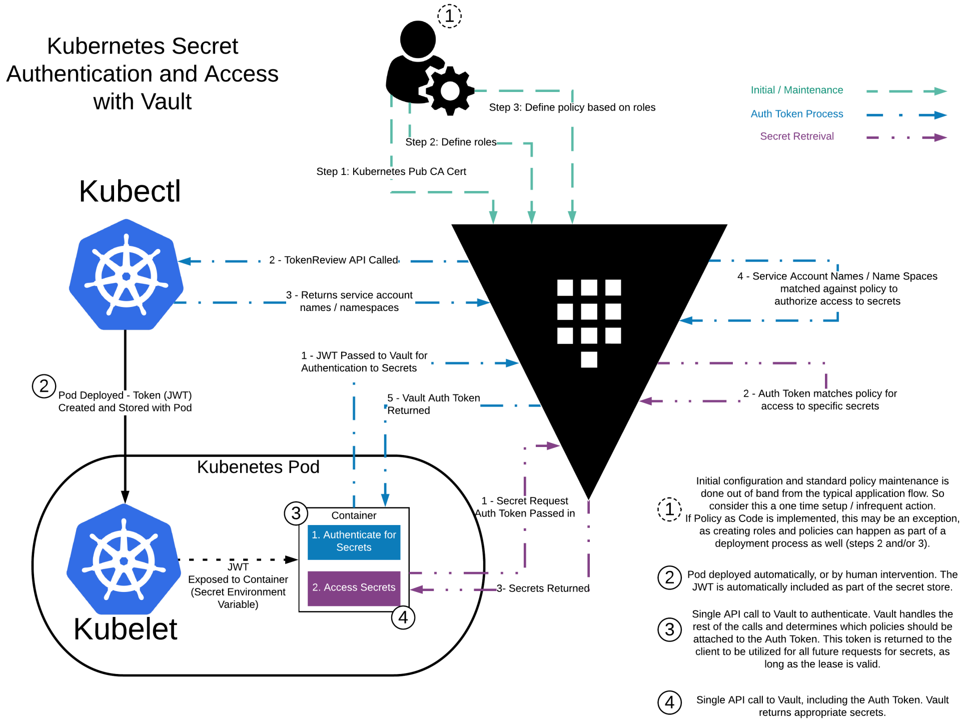
1 Initial configuration and standard policy maintenance is done out of band from the typical application flow. So consider this a one time setup / infrequent action. If Policy as Code is implemented, this may be an exception, as creating roles and policies can happen as part of a deployment process as well (steps 2 and/or 3).

2 Pod deployed automatically, or by human intervention. The JWT is automatically included as part of the secret store.

3 Single API call to Vault to authenticate. Vault handles the rest of the calls and determines which policies should be attached to the Auth Token. This token is returned to the client to be utilized for all future requests for secrets, as long as the lease is valid.

4 Single API call to Vault, including the Auth Token. Vault returns appropriate secrets.

Kubernetes Workflow Details


Kubernetes Secret Authentication and Access with Vault
1
Step 3: Define policy based on roles
Step 2: Define roles
Step 1: Kubernetes Pub CA Cert
Initial / Maintenance
Auth Token Process
Secret Retreival
Kubectl
2 - TokenReview API Called
3 - Returns service account names / namespaces
4 - Service Account Names / Name Spaces matched against policy to authorize access to secrets
1 - JWT Passed to Vault for Authentication to Secrets
2 - Auth Token matches policy for access to specific secrets
2
Pod Deployed - Token (JWT) Created and Stored with Pod
5 - Vault Auth Token Returned
Kubenetes Pod
1 - Secret Request Auth Token Passed in
3
Container
1. Authenticate for Secrets
2. Access Secrets
4
JWT Exposed to Container (Secret Environment Variable)
3- Secrets Returned
Kubelet

1. Initial configuration and standard policy maintenance is done out of band from the typical application flow. So consider this a one time setup / infrequent action. If Policy as Code is implemented, this may be an exception, as creating roles and policies can happen as part of a deployment process as well (steps 2 and/or 3).
2. Pod deployed automatically, or by human intervention. The JWT is automatically included as part of the secret store.
3. Single API call to Vault to authenticate. Vault handles the rest of the calls and determines which policies should be attached to the Auth Token. This token is returned to the client to be utilized for all future requests for secrets, as long as the lease is valid.
4. Single API call to Vault, including the Auth Token. Vault returns appropriate secrets.

Kubernetes Workflow

- Pod starts with either an init container or sidecar with vault agent
- Vault agent submits the JWT to vault for authentication
- Vault validates with the KubeAPI service
- Vault sends a token back to vault agent
- Vault agent either stores token or renders a config for pod

K8s Auth - Vault Config

Sets the configuration of the service account Vault will use to authenticate other pods:

```
$ vault write auth/kubernetes/config \
    token_reviewer_jwt="reviewer_service_account_jwt" \
    kubernetes_host=https://192.168.99.100:8443 \
    kubernetes_ca_cert=@ca.crt
```

This is an example of setting up a role for a pod to use. Note: the role is bound to a specific account and namespace

```
$ vault write auth/kubernetes/role/demo \
    bound_service_account_names=vault-auth \
    bound_service_account_namespaces=default \
    policies=default \
    ttl=1h
```

K8s Auth - Configure Vault Account

```
apiVersion: rbac.authorization.k8s.io/v1beta1
  kind: ClusterRoleBinding
metadata:
  name: role-tokenreview-binding
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:auth-delegator
subjects:
  - kind: ServiceAccount
    name: vault-auth
    namespace: default
```

Vault needs a role binding in Kubernetes

K8s Auth - Configure Vault Account

```
apiVersion: rbac.authorization.k8s.io/v1beta1
 kind: ClusterRoleBinding
metadata:
  name: role-tokenreview-binding
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:auth-delegator
subjects:
  - kind: ServiceAccount
    name: vault-auth
    namespace: default
```

Allow the service account to access the tokenreview API

K8s Auth - Configure Vault Account

```
apiVersion: rbac.authorization.k8s.io/v1beta1
 kind: ClusterRoleBinding
metadata:
  name: role-tokenreview-binding
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:auth-delegator
subjects:
  - kind: ServiceAccount
    name: vault-auth
    namespace: default
```

Specify the created service account

K8s Auth - Vault Agent

The vault agent can automatically manage getting secrets for a pod either through a side-car or init pattern

- For static secrets an init container is the best method
- For dynamic secrets a side-car is the best method
- Using the template block makes it easy to inject the secret into the pod
- We will be covering vault agent in more detail in another module

AppRole

AppRole Auth - Overview

- AppRole is used when no other 3rd party authentication method is available

- It is a more complicated workflow

- Creates a 2 factor system for creating OTP like authentication for machines

- Best used with

  - On Premise VMs
  - CI/CD Pipelines
  - Anywhere third party validation systems are not available

AppRole Auth - RoleID and SecretID

AppRole separates the authentication token into two parts

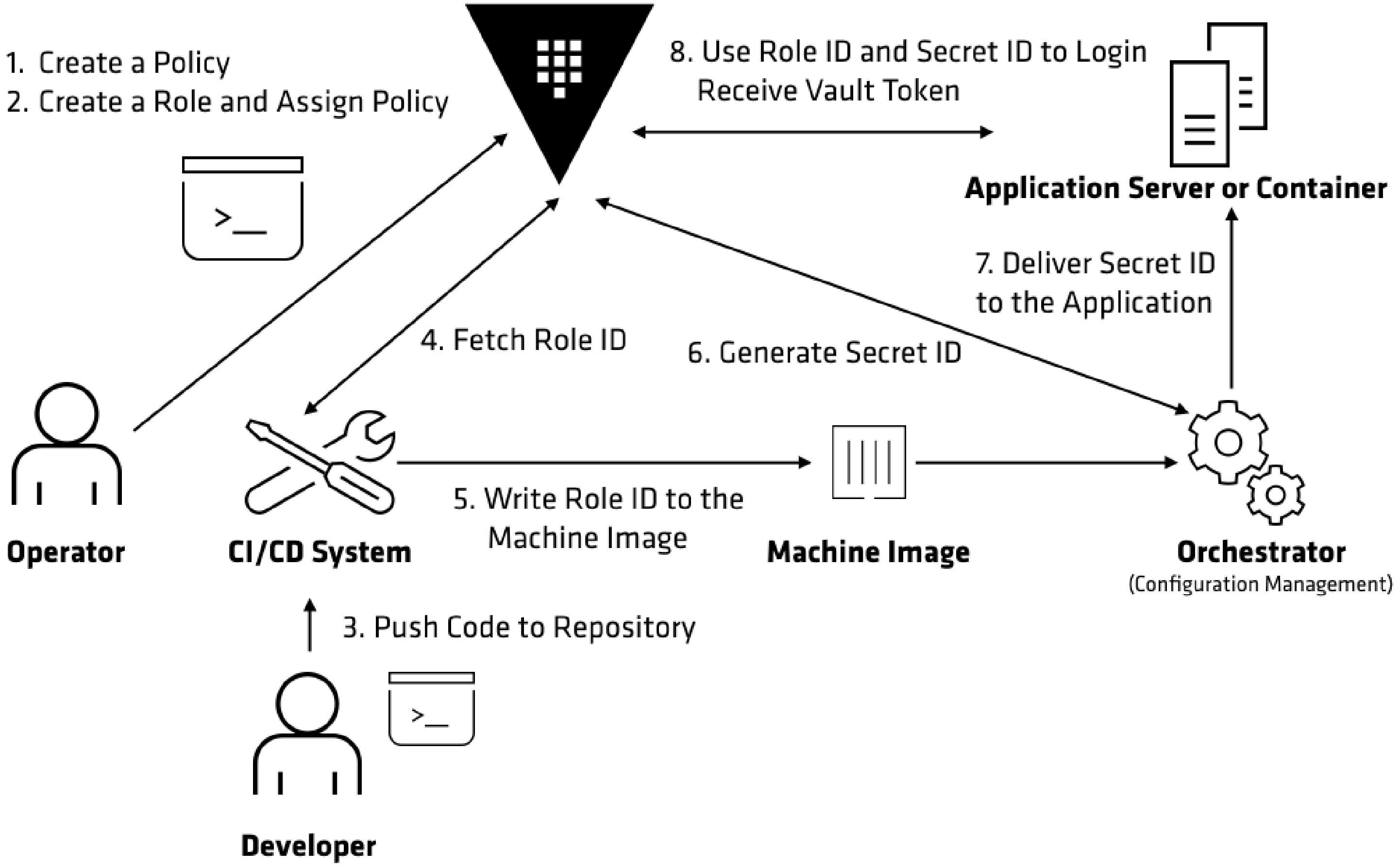
RoleID:

- Considered a "Username" for the machine
- This can be known and in clear text
- Is attached to a role on the machine
- Created by an operator or build system

SecretID:

- Considered the "Password" for the machine
- Generated dynamically by a build system or configuration automation
- Has a usage limit and short expiration

AppRole Auth - Detailed Workflow



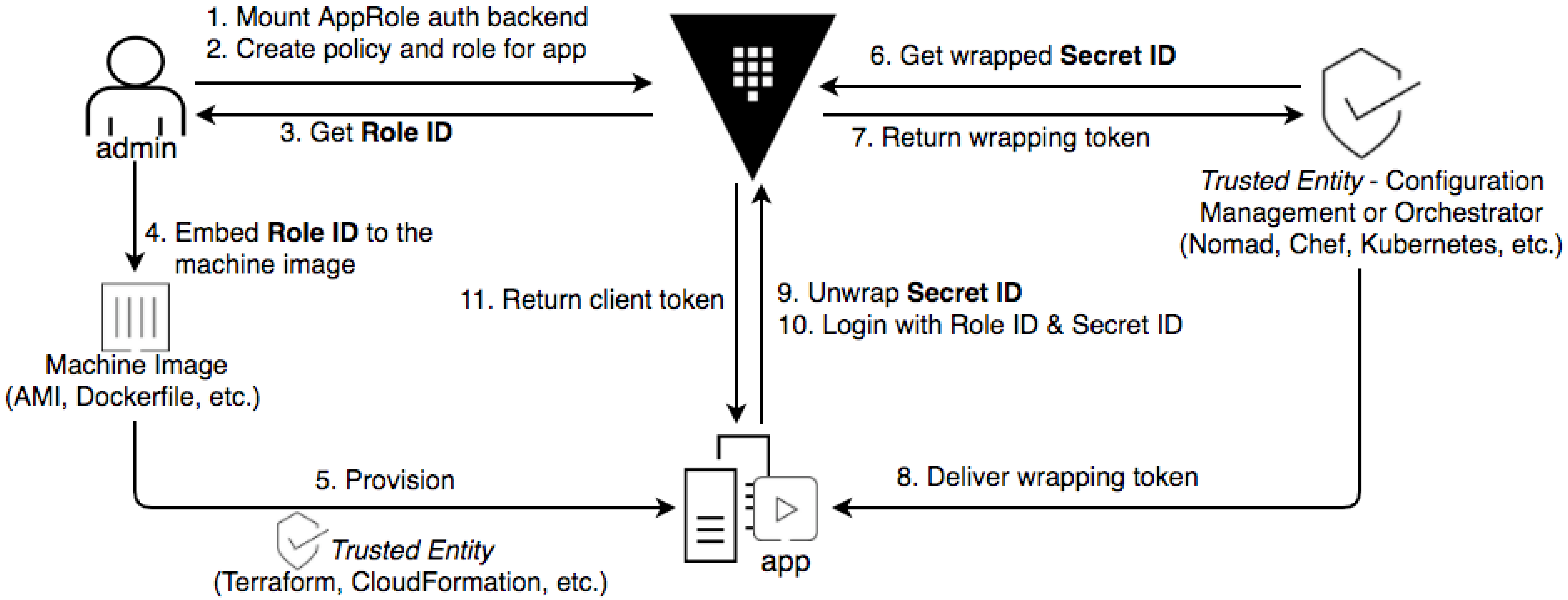
AppRole Auth - Things To Consider

There are several parameters to consider when creating a role that can generate roleID/secretIDs

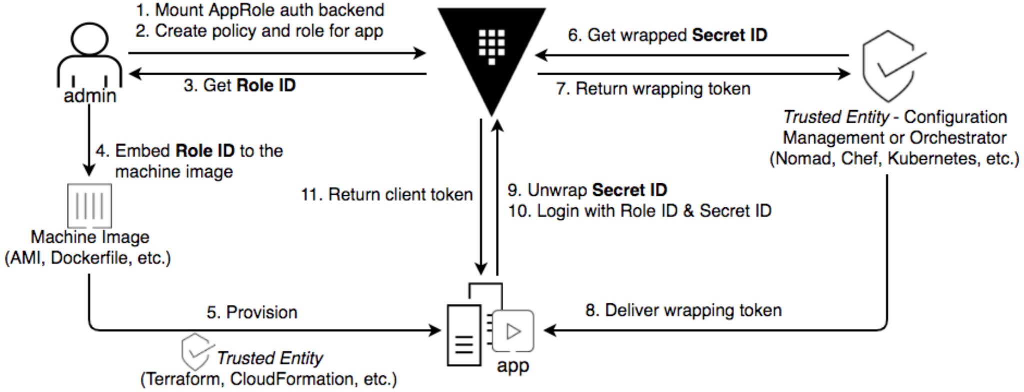
These can be set independently and are bound to the role assigned to generate appRole role/secretIDs

- How many times can an appRole token be used?
- How long will an appRole token live
- How many times can a vault token linked to an appRole token be used?
- How long can a vault token linked to an appRole token live?

AppRole Auth - Secure Delivery


1. Mount AppRole auth backend
2. Create policy and role for app
admin
3. Get **Role ID**
4. Embed **Role ID** to the machine image
Machine Image
(AMI, Dockerfile, etc.)
6. Get wrapped **Secret ID**
7. Return wrapping token
Trusted Entity - Configuration Management or Orchestrator
(Nomad, Chef, Kubernetes, etc.)
11. Return client token
9. Unwrap **Secret ID**
10. Login with Role ID & Secret ID
5. Provision
Trusted Entity
(Terraform, CloudFormation, etc.)
app
8. Deliver wrapping token

AppRole Auth - Workflow



1. Mount AppRole auth backend
2. Create policy and role for app
3. Get **Role ID**
admin
4. Embed **Role ID** to the machine image
Machine Image
(AMI, Dockerfile, etc.)
5. Provision
Trusted Entity
(Terraform, CloudFormation, etc.)
6. Get wrapped **Secret ID**
7. Return wrapping token
Trusted Entity - Configuration Management or Orchestrator
(Nomad, Chef, Kubernetes, etc.)
8. Deliver wrapping token
9. Unwrap **Secret ID**
10. Login with Role ID & Secret ID
11. Return client token
app

Secure Delivery Workflow

- Allows for a secure delivery of the secretID
- Wrapped with a one time use token
- Creates low surface area as Orchestrator can only get wrapped secretIDs

AppRole Auth - Wrap SecretID

```
$ vault write -wrap-ttl=60s -f auth/approle/role/jenkins/secret-id

Key                              Value
--                               --
wrapping_token:                  s.2kAzCgg1kN7vdpE1xxZxzpug
wrapping_accessor:               1N3YCs02iuZ75OCTi4eN0tuA
wrapping_token_ttl:              1m
wrapping_token_creation_time:    2019-12-16 17:17:13.956126 -0800 PST
wrapping_token_creation_path:    auth/approle/role/jenkins/secret-id
name: summary
```

Wrapping the SecretID gives a separate, one time use token to retrieve the real SecretID

AppRole Auth - Unwrap SecretID

```
$ VAULT_TOKEN=s.2kAzCgg1kN7vdpE1xxZxzpug vault unwrap

Key                   Value
--                    --
secret_id             7673bcf6-bbba-0fa6-a54c-51a6a3219c92
secret_id_accessor    e0104ca1-0afd-5d90-3b99-646bbcb5c179
```

AppRole Auth - Fetch Vault Token

```
$ vault write auth/approle/login role_id="675a50e7-cfe0-be76-e35f-49ec00
  secret_id="ed0a642f-2acf-c2da-232f-1b21300d5f29"

  Key                       Value
  --                        --
  token                     s.ncEw5bAZJqvGJgl8pBDM0C5h
  token_accessor            gIQFfVhUd8fDsZjC7gLBMnQu
  token_duration            1h
  token_renewable           true
  token_policies            ["default" "jenkins"]
  identity_policies         []
  policies                  ["default" "jenkins"]
  token_meta_role_name      jenkins
```

Chapter Summary

- Pick the right authentication method based on security and ease of access
- Keep human and machine auth methods separate
- Leverage namespaces to isolate similar configurations
- Vault agent is the preferred method of interfacing for machines

Reference links

- Authentication Methods Documentation
- Authentication Methods API Documentation
- Concept Review of Authentication Methods

Vault Authentication Methods Module Complete!