Vault

Implementation Foundations

Module 13: Deploying Secrets with Vault

What You Will Learn

- Putting it All Together
- Getting Secrets Using Vault Agent
- Getting Secrets Using Platform Native Capabilities
- Using Application Libraries

Deploying Secrets Overview

Putting It All Together

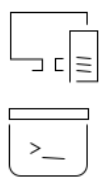
We have covered all the core concepts needed to expose secrets safely.

- Authentication Methods
- Policies
- Vault Tokens
- Secret Engines

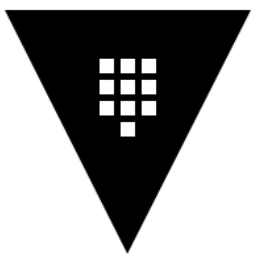
How do you put these all together for consumption?

Vault Workflow (1/5)

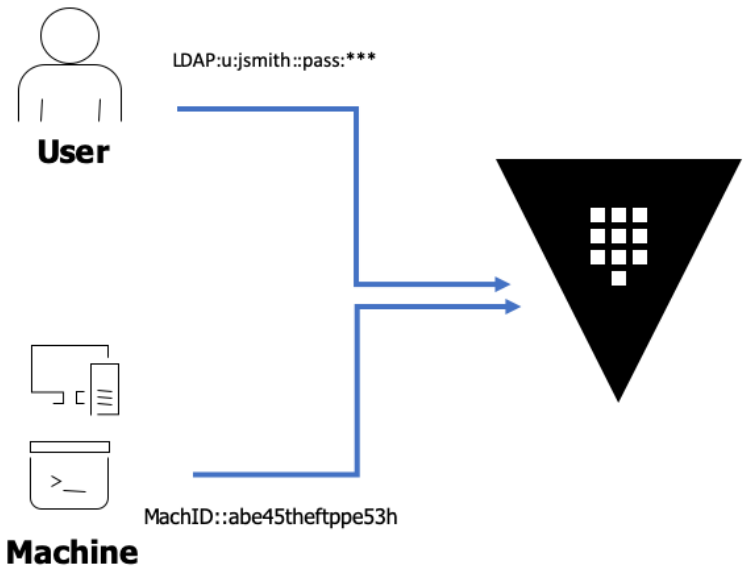
User



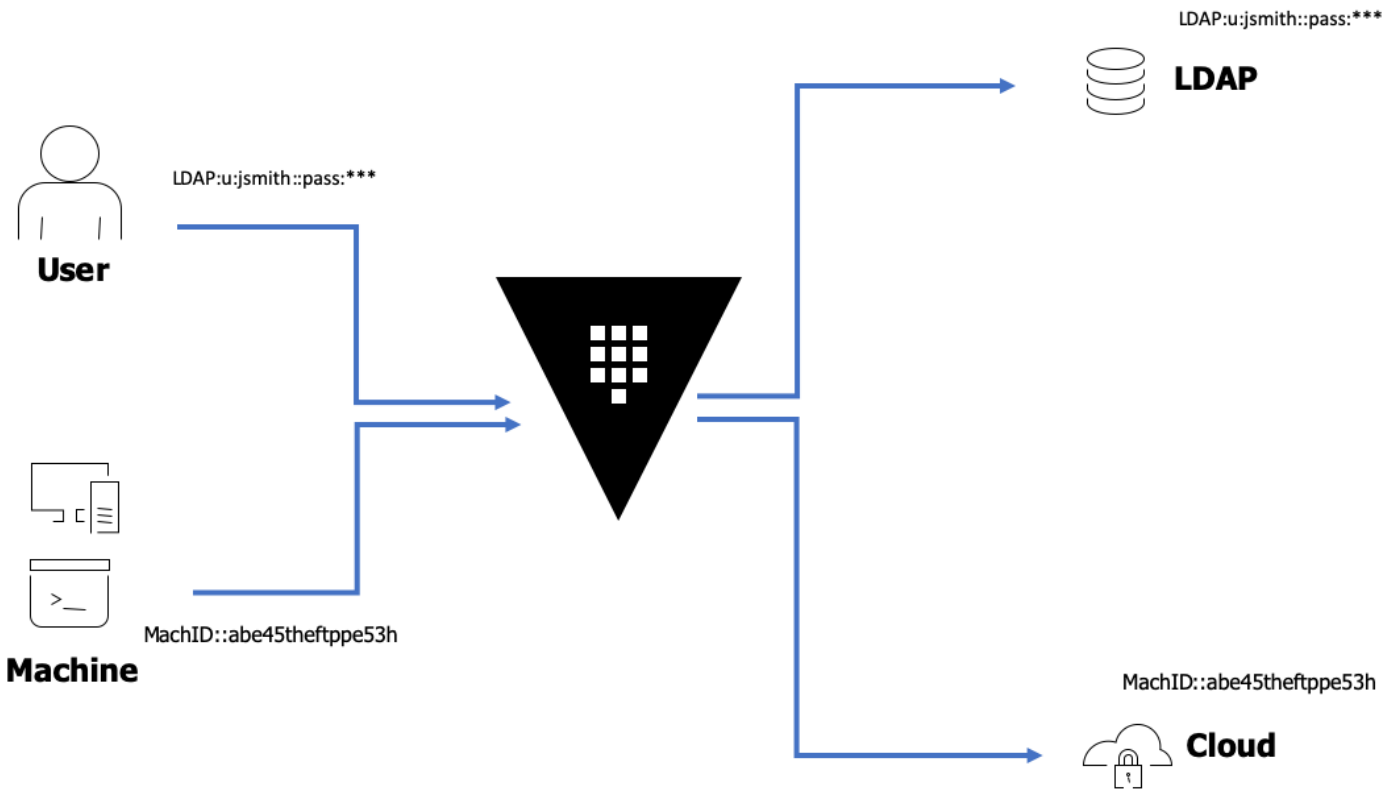
Machine



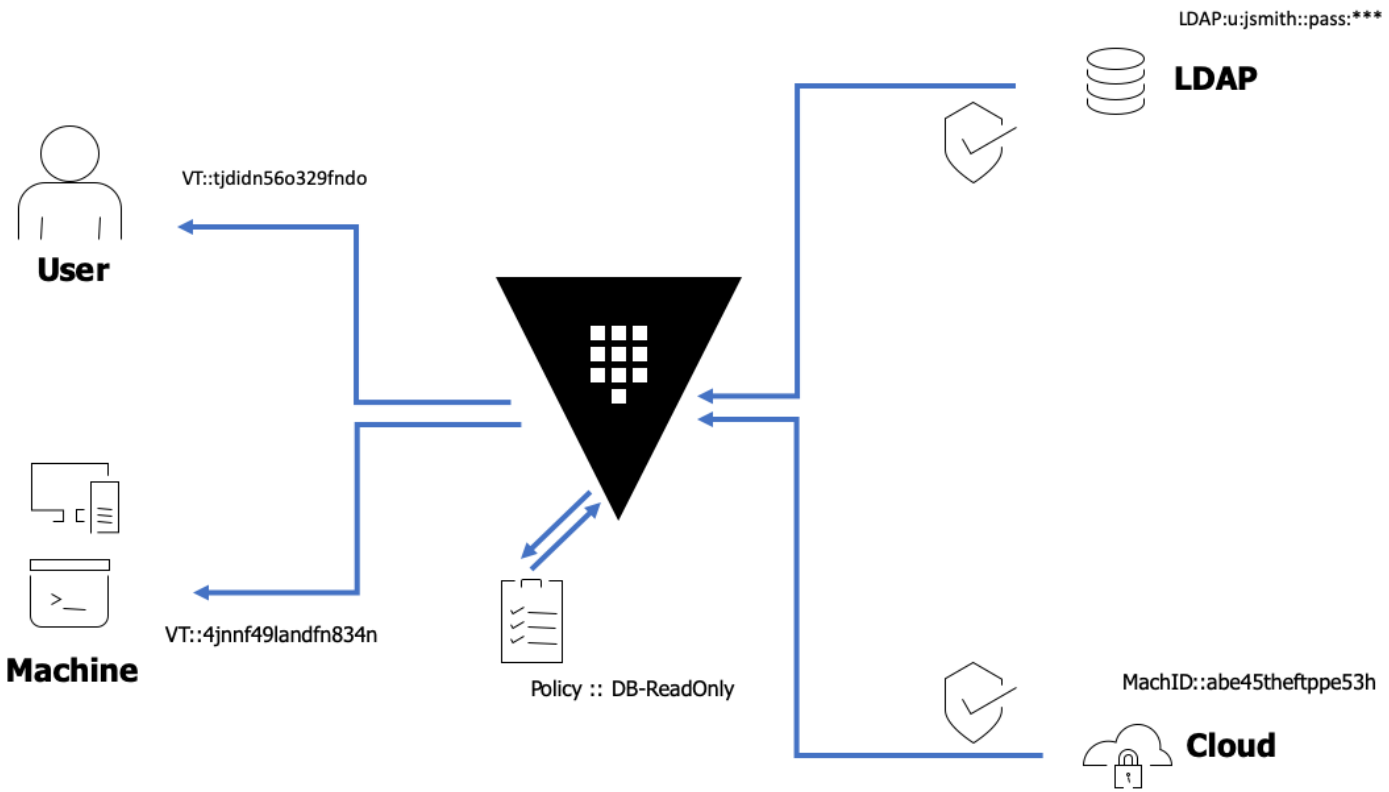
Vault Workflow (2/5)



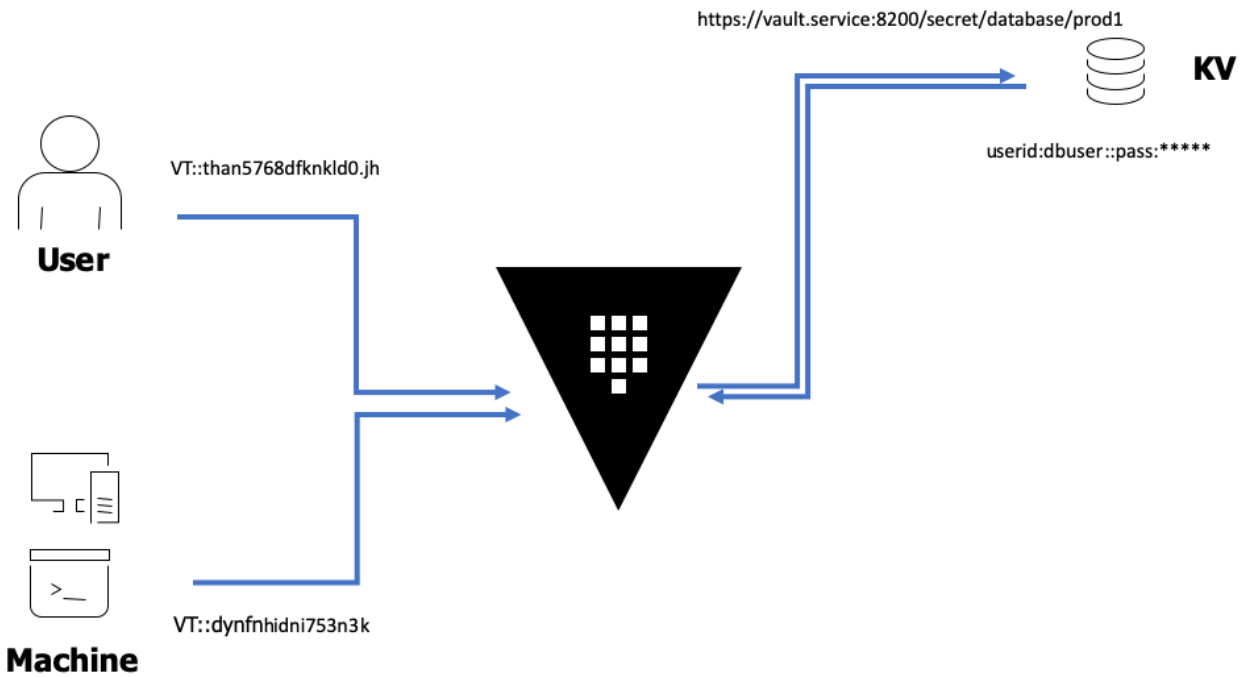
Vault Workflow (3/5)



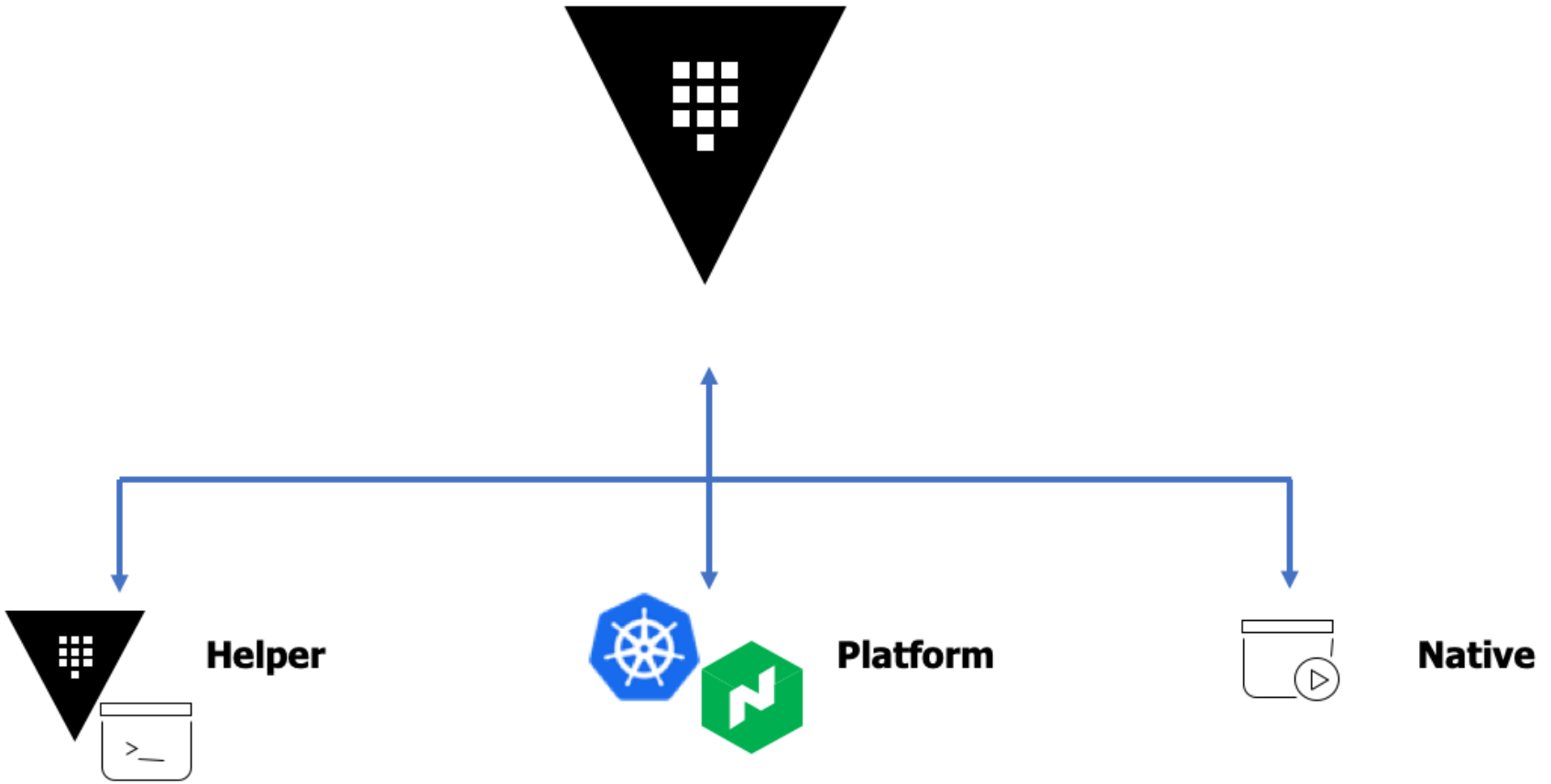
Vault Workflow (4/5)



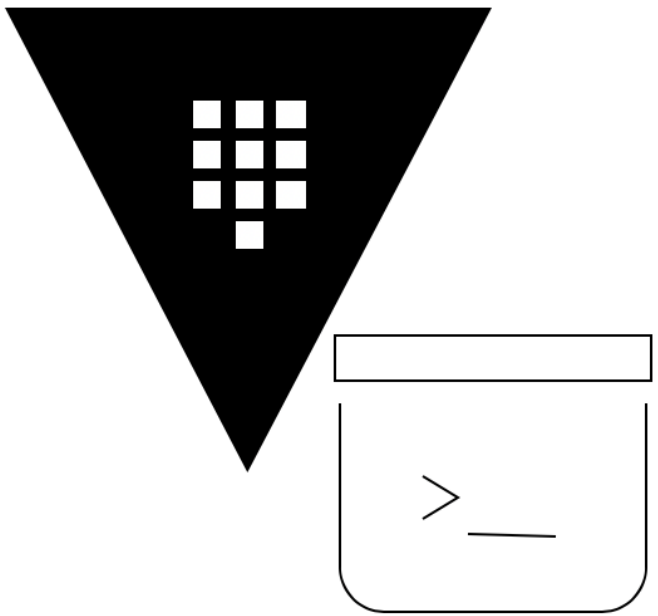
Vault Workflow (5/5)



Vault Secrets Injection Methods



Vault Secrets Helper Method



Vault Agent

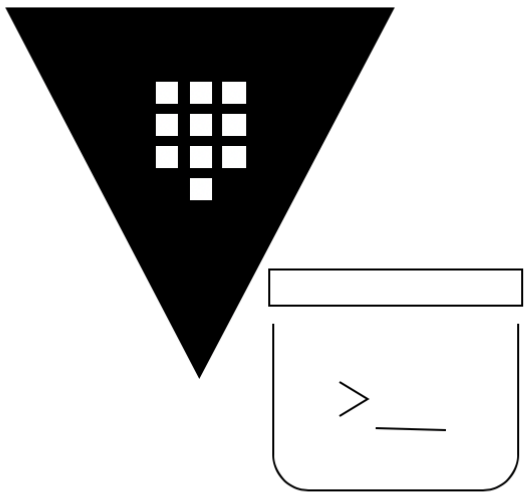
- Provides a transparent way of authenticating
- Automatically grabs a vault token
- Support several Auth Methods
- Uses the consul-template format

Vault Secrets Platform Method

Platform Method

- The platform supports the brokering of secrets
- Kubernetes
- Nomad
- Leverages separate template workflow to inject secrets

Vault Secrets Native Method



Native Integration

- Application has native integration
- Vault Aware deployments
- High portability
- Requires code changes
- Ultimate goal

Vault Agent Overview

Vault Agent Overview (1/2)

- Agent is configured with three parts: Method, Sink, and Template
- The Method is how the agent will authenticate with Vault
- Several Auth Methods are supported
  - Cloud Providers
  - JWT/Kubernetes
  - Certificate
  - App/Role

Vault Agent Overview (2/2)

- The Sink is where the vault token will be written to
  - Currently only file writing is support
  - The token can be encrypted
- The template uses the consul-template format to automatically generate configuration files for apps to consume on startup.

Vault Agent Setup

There are three main configuration blocks for the agent:

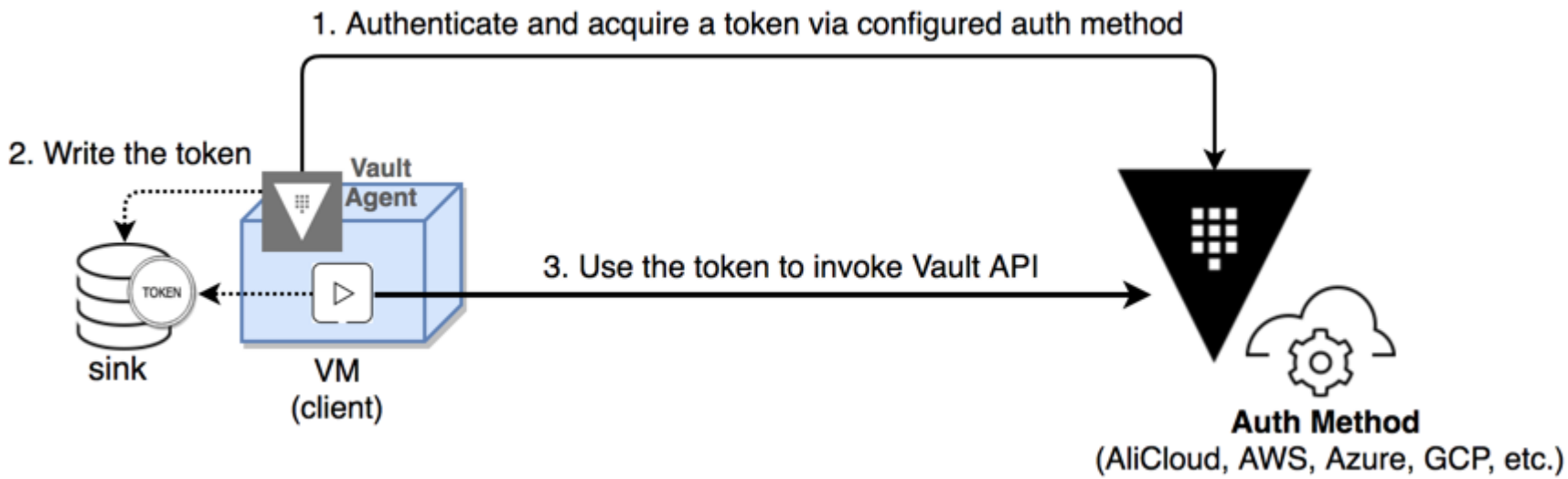
- vault{} - Tells the agent where the vault server is
- auto-auth{} - Tells the agent how to authenticate with Vault
- template{} - Directs the agent on how to generate application configuration templates to consume

Configuration Example

```
#vault-agent.hcl
# Location of the Vault Server
vault {
    address = "https://<YOUR_VAULT_SERVER:8200"
}
auto_auth {
    # The authentication method
    method {
        type   = <AUTH_TYPE>
        config = {
            <CONFIG_PARAMETERS>
        }
    }
    # Token Sink
    sink {
        type   = "file"
        config = {
            path = "tmp/super-secret"
        }
    }
}
template {
    # Location of the consul-tempate file
    source      = "<PATH_TO_CTMPL_FILE>"

    # Location of the output file and file name
    destination = "<PATH_AND_FILE_NAME>"
}
```

- `vault{ }` : defines where the Vault server is
- `auto_auth { }` : defines how to authenticate with Vault
- `method { }` : defines the authentication method
- `sink { }` : defines where the token is stored)
- `template { }` : defines the template and output file

The Vault Agent Workflow



Vault Agent Template

```
# Define the endpoint to access secret
{{ with secret "my-app/database/production" }}

#Retrieve secret
{{ .Data.username }}
{{ .Data.password }}
```

- Vault agent supports consul template format:
- Easily remove secrets with minimal application change
- CTMPL file in
- Config file output

Vault Agent Template Example

```
{{ with secret "my-app/database/production" }}
<Context>

  <Resource name="jdbc/mkyongdb" auth="Container" type="javax.sql.DataSo
                maxActive="50" maxIdle="30" maxWait="10000"
                username="{{ .Data.username }}"
                password="{{ .Data.password }}"
                driverClassName="com.mysql.jdbc.Driver"
                url="jdbc:mysql://localhost:3306/mkyongdb"/>

</Context>
```

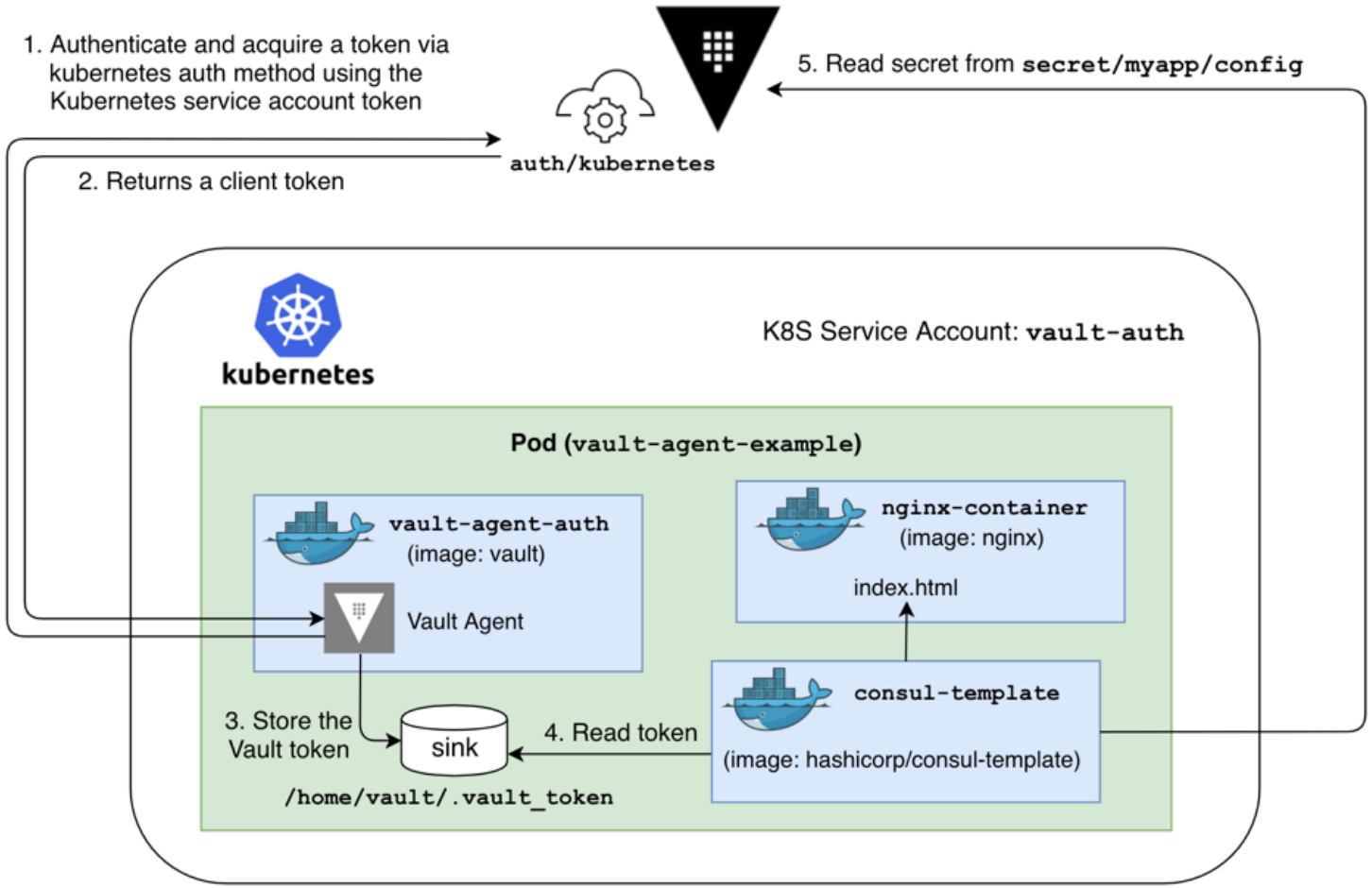
This defines the endpoint to retrieve the secret

Vault Agent Template Example

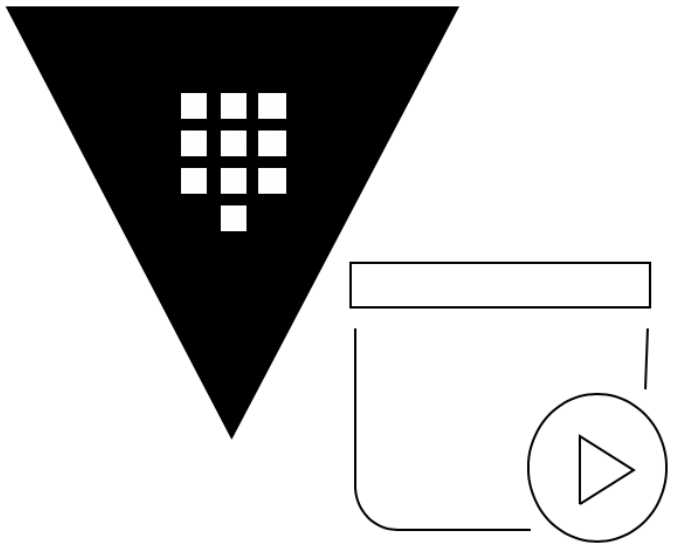
```
{{ with secret "my-app/database/production" }}
<Context>

  <Resource name="jdbc/mkyongdb" auth="Container" type="javax.sql.DataSo
                maxActive="50" maxIdle="30" maxWait="10000"
                 username="{{ .Data.username }}"
                 password="{{ .Data.password }}"
                driverClassName="com.mysql.jdbc.Driver"
                url="jdbc:mysql://localhost:3306/mkyongdb"/>

</Context>
```

This interpolates the vault key/value data when generating the output file

Vault Agent Kubernetes Example



Vault Native Integration



- Most flexibility
- Most portability
- Requires application changes

Vault Native Integration Considerations

- Vault libraries are community supplied
- Must handle vault token (Initial, Renew, Refresh)
- Application must know how to handle the vault workflow

Chapter Summary

- Vault has auth workflows designed for humans and machines
- There are three main ways to inject Secrets
  - Helper
  - Platform
  - Native
- Consider the business requirements and capabilities

Reference links

- Vault Agent
- Vault Libraries
- CSI Driver

Vault Deploying Secrets Module Complete!