Module 14: Dynamic Secrets

What You Will Learn

- What is a Dynamic Secret
- Dynamic Secret Types
- Databases
- PKI
- Cloud Credentials
- Encryption Keys

Dynamic Secrets

Dynamic Secret Overview

- Credential that is created at time of request
- Time bound to a vault token lease
- Deleted automatically
- Unique to the requestor

Why Should I Use It?

- Tradition workflows expose secret at different points
- Shared service accounts make normal rotation difficult
- Hard to audit
- Long request and creation process

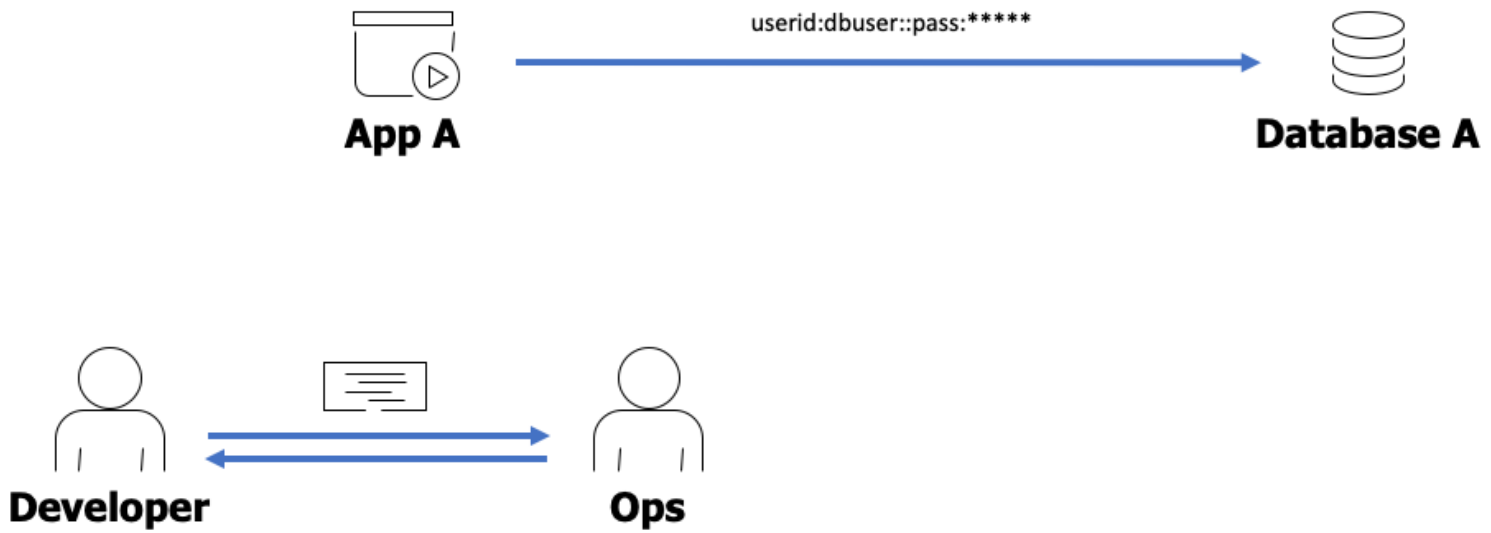
Supported Dynamic Secrets

Secrets Engines	Description
Cloud Keys	Generate Cloud API Keys
Databases	Generate database username/password
PKI	Generate leaf certificates
Encryption Keys	Generate data encryption keys
Consul	Generate Consul ACL tokens
RabbitMQ	Generate RabbitMQ user credentials
SSH	Generates signed SSH keys

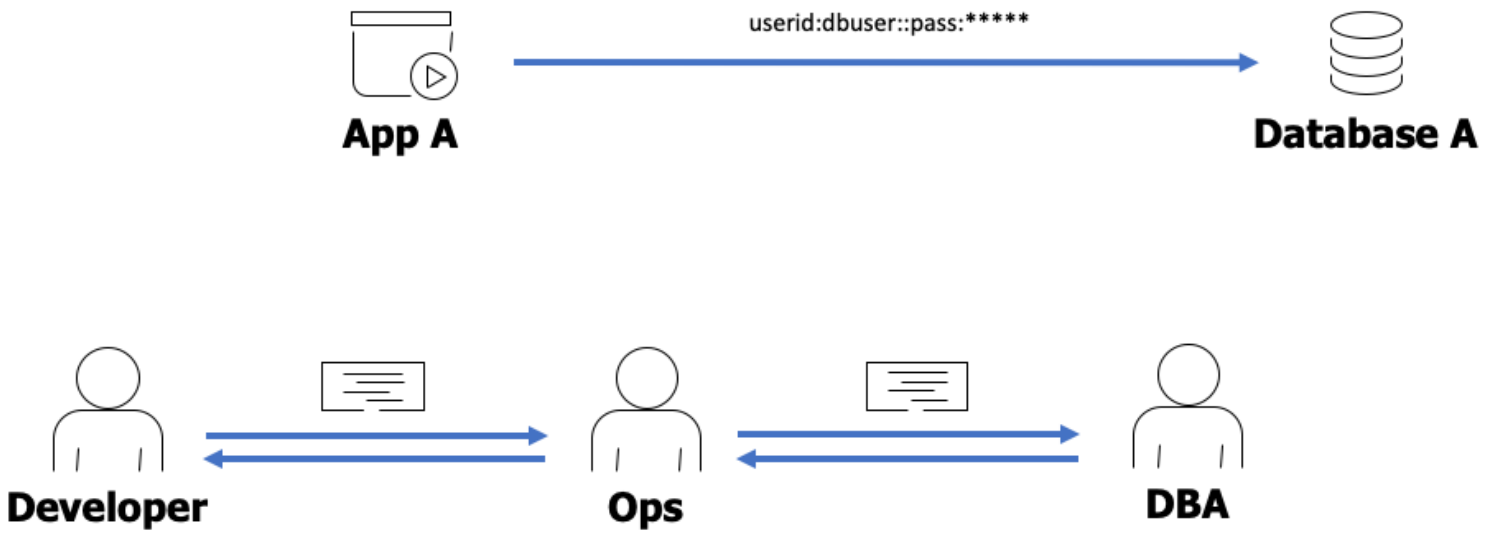
Dynamic Database Secrets Engine

Current Database Workflow Example

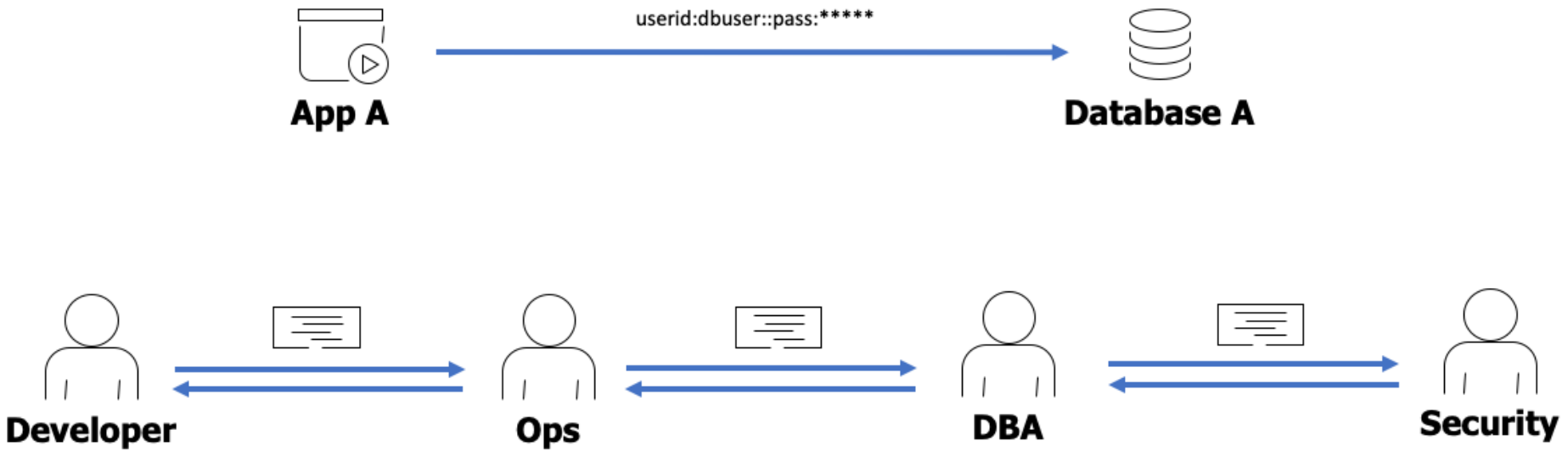
Developer Submits Request



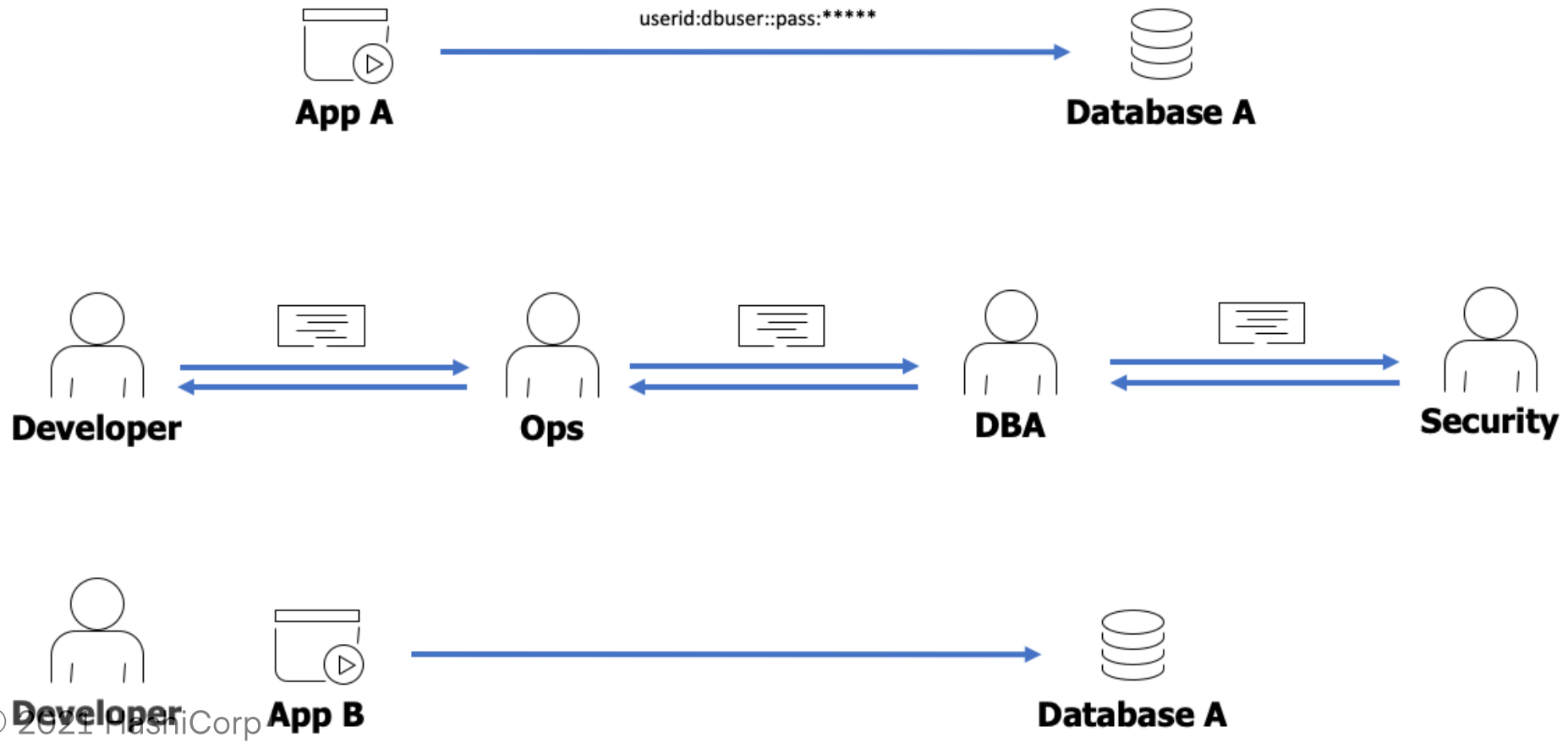
Operations Submits Request



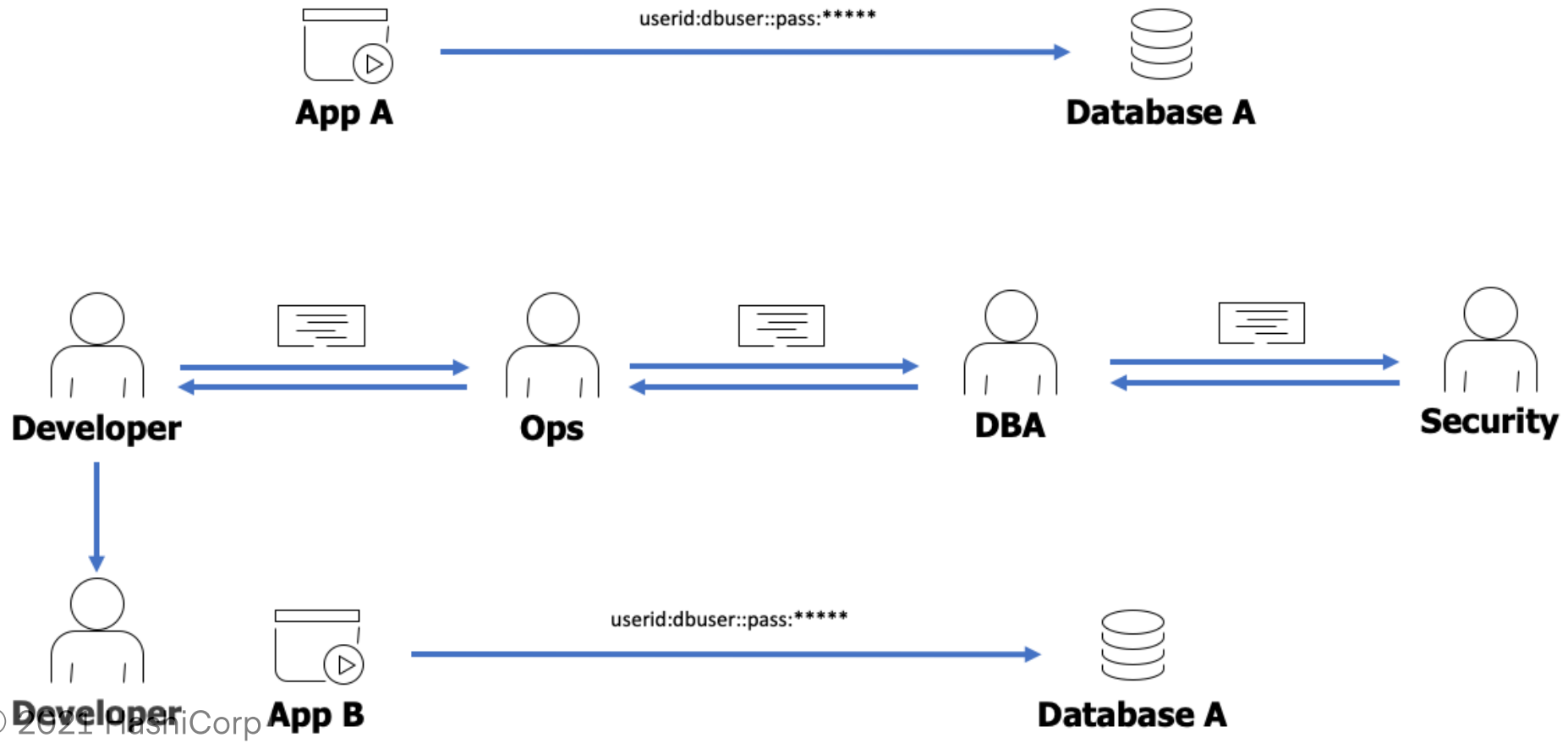
DBA Submits Request



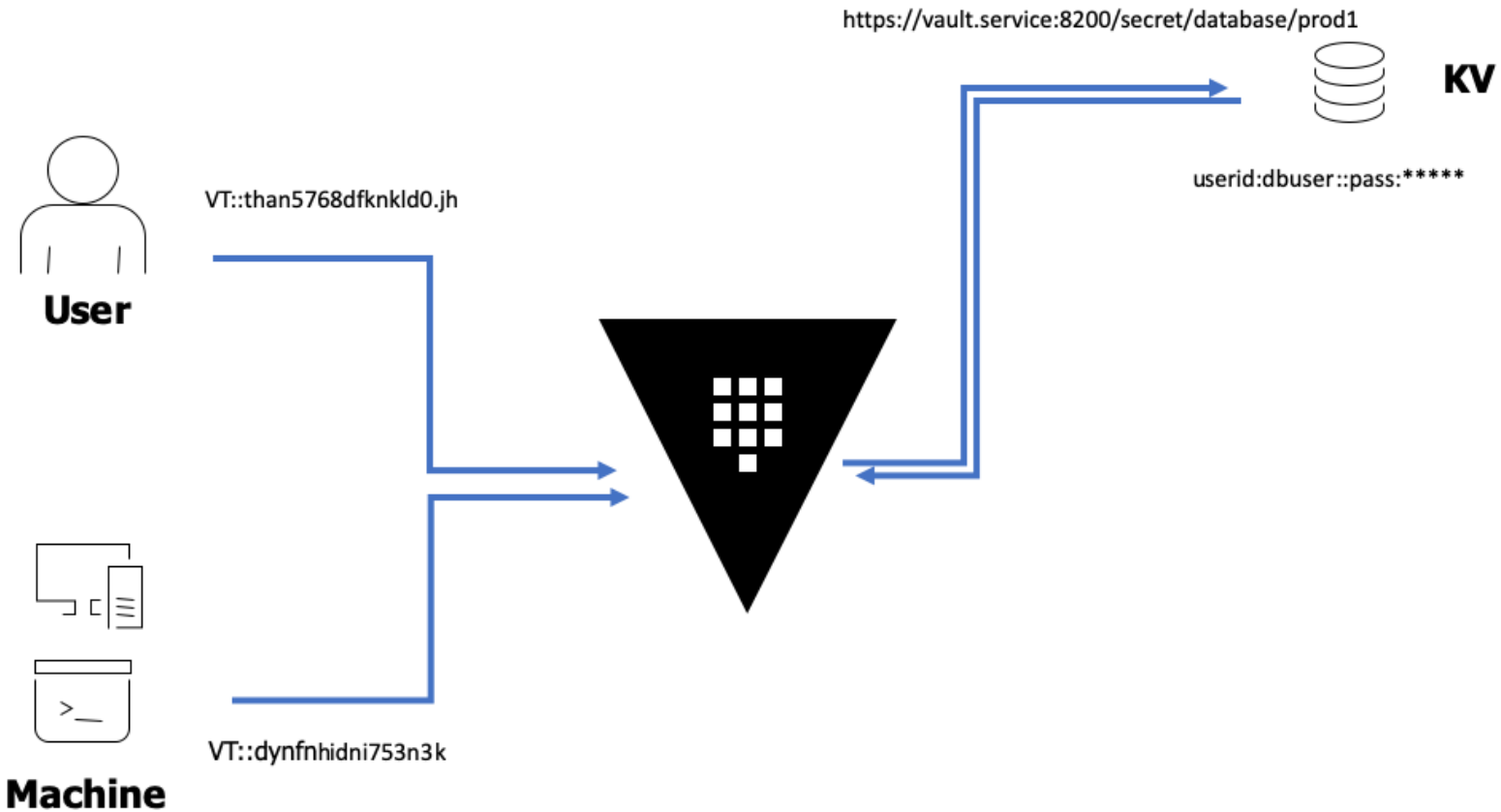
New Developer Needs Access



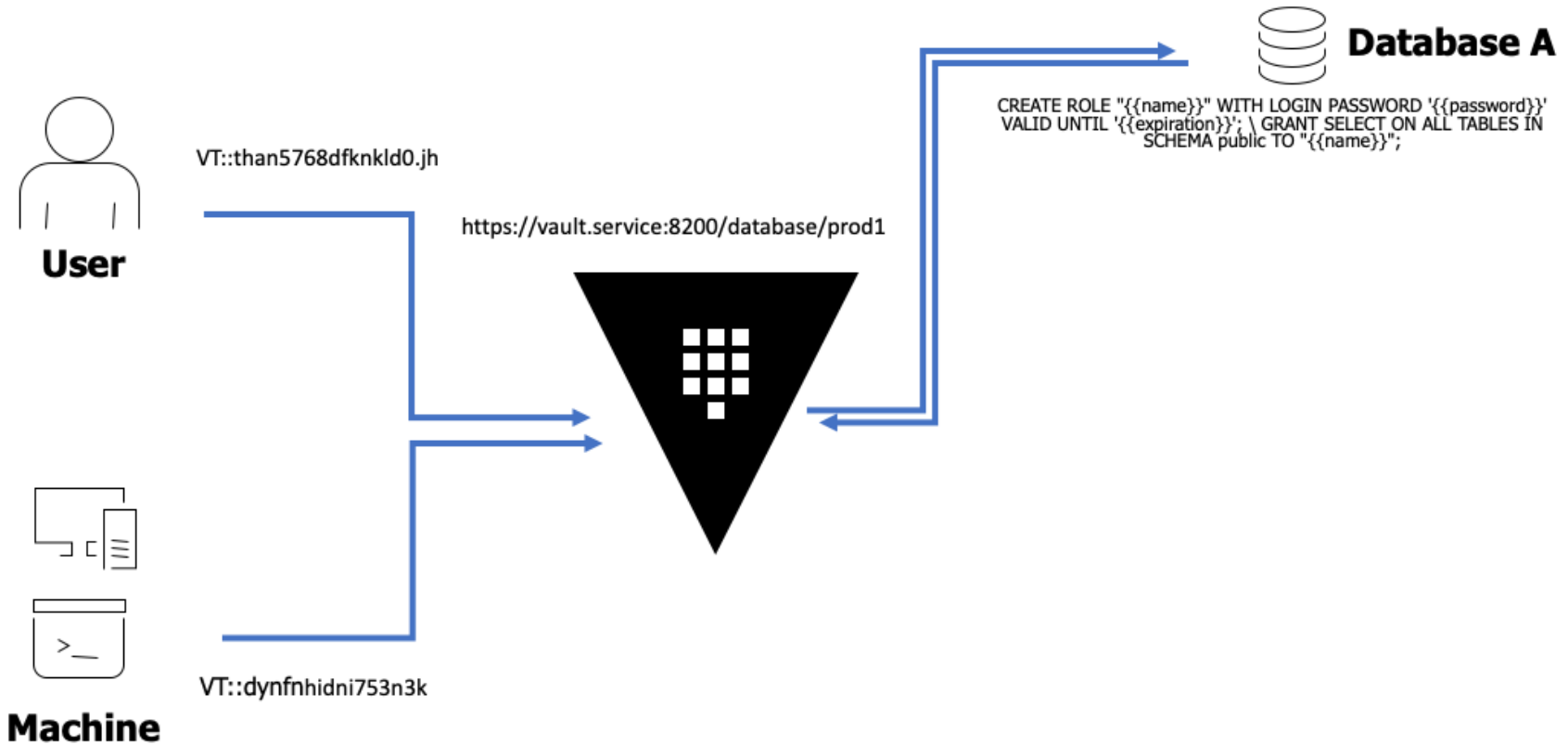
Credential Leak



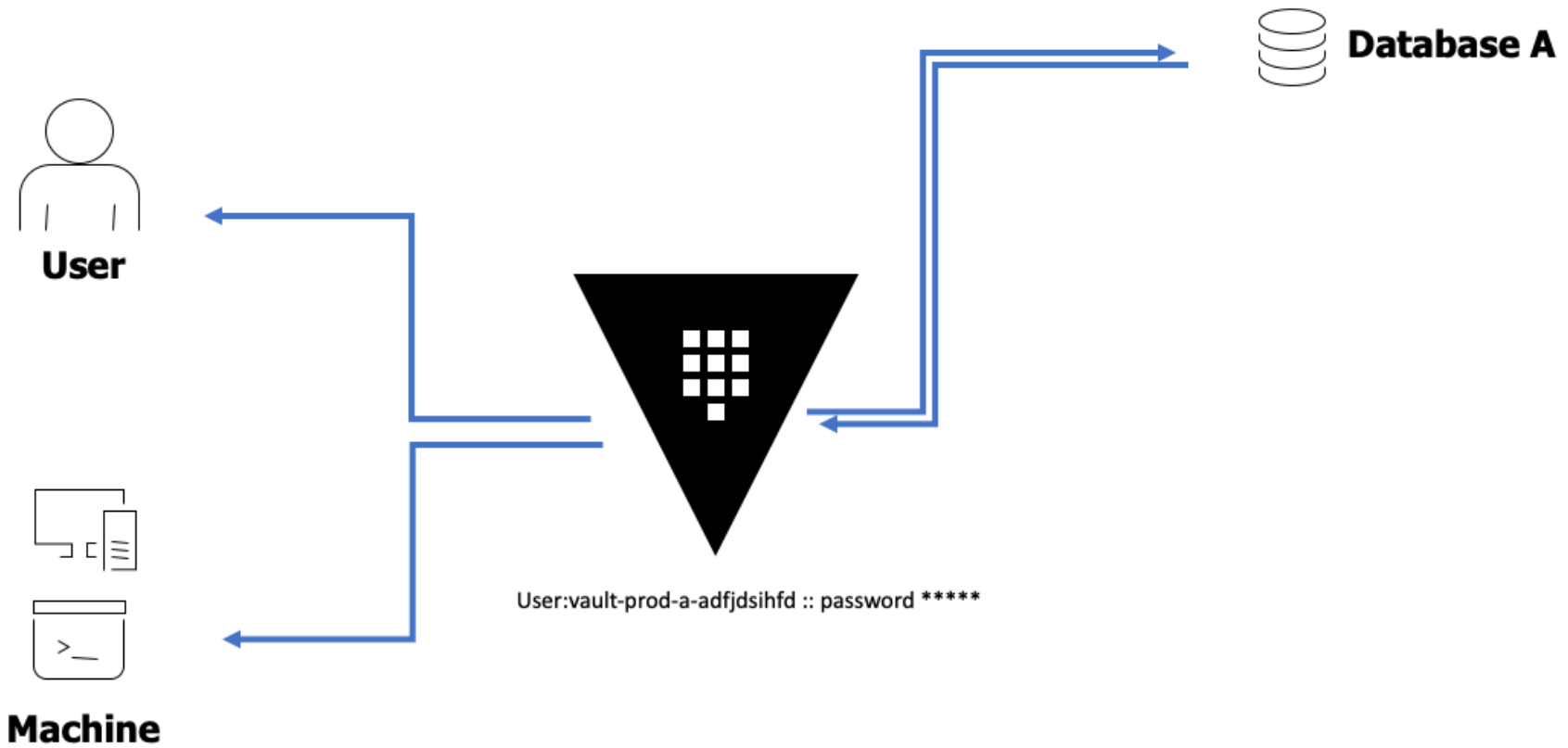
Dynamic Database Secrets Workflow



Machine Access Database Engine



Dynamic Credentials Returned



Dynamic Database Credentials Characteristics

- Unique Username and Password
- Granular permissions control (SQL Statement)
- *At Will* rotation strategy
- Granular auditing

Dynamic Database Secrets Requirements

- Vault user with enough permission to create database accounts
  - Account creation type (General access vs. administrative)
- Understanding of current account creation workflow
  - Are there audit systems in place
- Application endpoint changes
  - The most effective adoption is **after** static secret workflow is implemented

Dynamic Database Secrets Setup

```
$ vault secrets enable -path=db/myapp/prod database
```

Best practice is to enable a database engine per database

Vault Privileged DB Connection

```
$ vault write db/myapp/prod/config/postgresql-prod-1 \
    plugin_name=postgresql-database-plugin \
    allowed_roles="my-role" \
    connection_url="postgresql://{{username}}:{{password}}@localhost:5432/" \
    username="root" \
    password="root" \
    root_rotation_statements="ALTER ROLE {{username}} WITH PASSWORD '{{password}}';
```

- plugin_name: defines which DB plugin to use
- allowed_roles: defines which roles can generate credentials
- connection_url: defines the database connection string)
- root_rotation_statements: defines the optional Vault rotation string

Note: This will vary based on the database you are connecting to. Consult the database engine documentation for you specific database.

Dynamic Database Role Creation

```
CREATE ROLE "role-creation"
  WITH LOGIN PASSWORD '{{password}}'
  VALID UNTIL '{{expiration}}';
GRANT SELECT ON ALL TABLES
  IN SCHEMA public
  TO "role-creation";
```

- Create users with common SQL
- Best practice is to create the SQL needed in a file
- Creates a version control around database access

Note: Because users are created based of standard SQL this allows for a high level of granular access control vs. traditional service accounts.

Dynamic Database Role Creation

```
$ vault write db/myapp/prod/roles/readonly db_name=postgresql-prod-1 \
        creation_statements=@readonly.sql \
        default_ttl=1h max_ttl=24h
```

Write the role to the DATABASE/roles/ROLE_ID endpoint

- db_name: name given at the database config endpoint
- creation_statement: statement or file that contains the SQL
- default_ttl: how long a credential is valid without renewing
- max_ttl: the maximum time a credential can be valid

Retrieving Database Credential

```
$ vault read db/myapp/prod/creds/my-role
Key                Value
--                  --
lease_id           database/creds/my-role/2f6a614c-4aa2-7b19-24b9-ad944a
lease_duration     1h
lease_renewable    true
password           8cab931c-d62e-a73d-60d3-5ee85139cd66
username           v-root-e2978cd0-
```

With a valid vault token simply hit the end point similar to fetching a static secret

Vault Agent Change

```
{{ with secret "db/myapp/prod/creds/my-role" }}
<Context>

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

</Context>
```

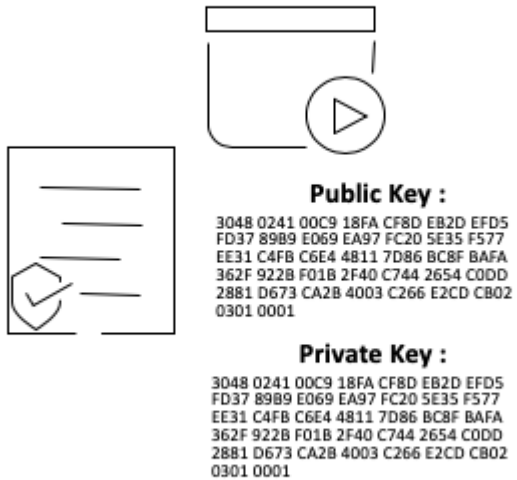
- name: developer changes one line in the vault agent template file (path to secret)
- {{Data.username}}, {{Data.password}} interpolates the vault key/value data when generating the output file

Dynamic Database Best Practices

- Good endpoint structure is critical
- Try to implement in a 1 to many fashion
- Separate service accounts endpoints from administrative accounts endpoints
- Use namespaces to make endpoint management easier for developers

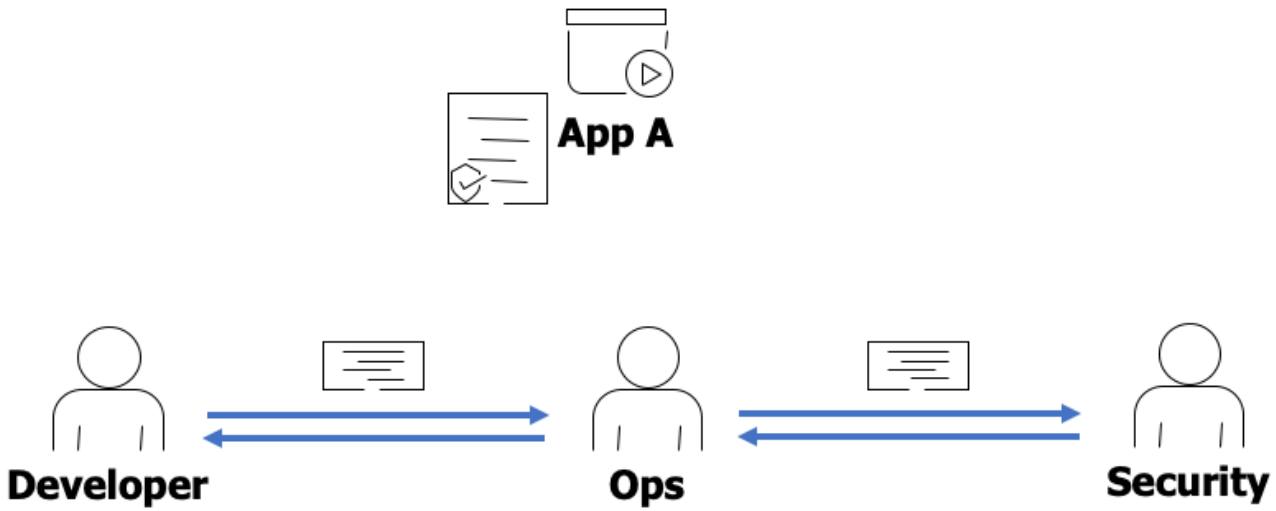
Dynamic PKI Certificates Secrets Engine

PKI Certificate Overview

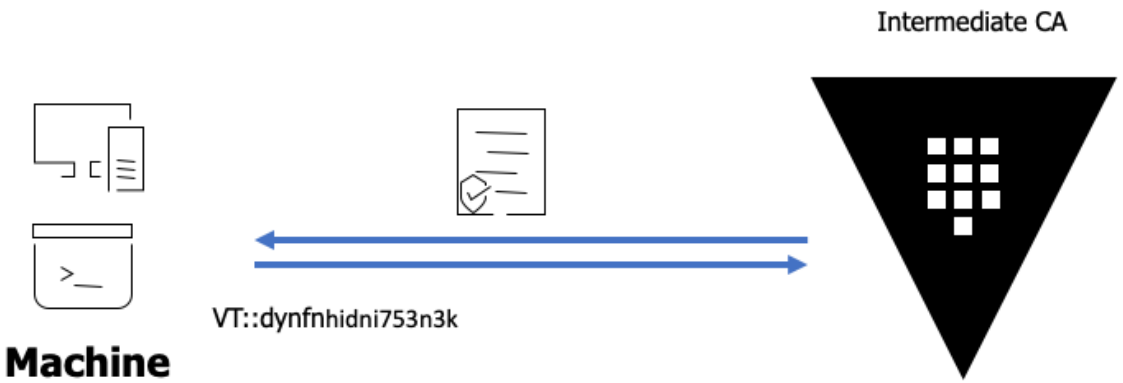


- Public Key Infrastructure
- Validating and establishing trusted communication
- Asymmetrical Key Pair

PKI Workflow



Dynamic PKI Certificate Workflow



Dynamic PKI Certificate Advantages

- Private Key never exposed
- Short lived certificates
- Rotation at will
- On Demand generation vs. manual process

Dynamic PKI Setup - Configure Engine

Enable the PKI endpoint:

```
$ vault secrets enable -path=pki_int pki
Successfully mounted 'pki' at 'pki_int'!
```

Configure the lease for the certificate to match the issuing CA:

```
$ vault secrets tune -max-lease-ttl=43800h pki_int
Successfully tuned mount 'pki_int'!
```

Dynamic PKI Setup - Generate CSR

Generate a CSR to be signed for vault

```
$ vault write pki_int/intermediate/generate/internal \
common_name="myvault.com Intermediate Authority" ttl=43800h

csr --BEGIN CERTIFICATE REQUEST--
MIICsjCCAZoCAQAwLTErMCkGA1UEAxMibXl2YXVsdC5jb20gSW50ZXJtZWRpYXRl
IEF1dGhvcml0eTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAJU1Qh8l
.......
--END CERTIFICATE REQUEST--
```

Dynamic PKI Setup - Upload Signed Certificate

Upload the signed certificate

```
$ vault write pki_int/intermediate/set-signed \
certificate=@signed_certificate.pem

Success! Data written to: pki_int/intermediate/set-signed
```

Dynamic PKI Setup - Configure CRL

Best practice is to configure any generated certificate to call back to vault to check if the cert has been revoked. The CRL URL has to be a FQDN. This allows security to be able to revoke a cert before its expiration.

```
$ vault write pki_int/config/urls \
issuing_certificates="http://[FQDN]:8200/v1/pki_int/ca" \
crl_distribution_points="http://[FDQN]:8200/v1/pki_int/crl"

Success! Data written to: pki_int/ca/urls
```

Dynamic PKI Setup - Configure Role

```
$ vault write pki_int/roles/example-dot-com \
        allowed_domains="example.com" \
        allow_subdomains=true \
        max_ttl="720h"

Success! Data written to: pki_int/roles
```

Define certificate parameters when configuring the role. There are several parameters that can be configured. For example:

`allowed_domains` - Specifies the domains of the role

`allow_bare_domains` - Specifies if clients can request certificates matching the value of the actual domains themselves

`allow_subdomains` - Specifies if clients can request certificates with CNs that are subdomains of the CNs allowed by the other role options

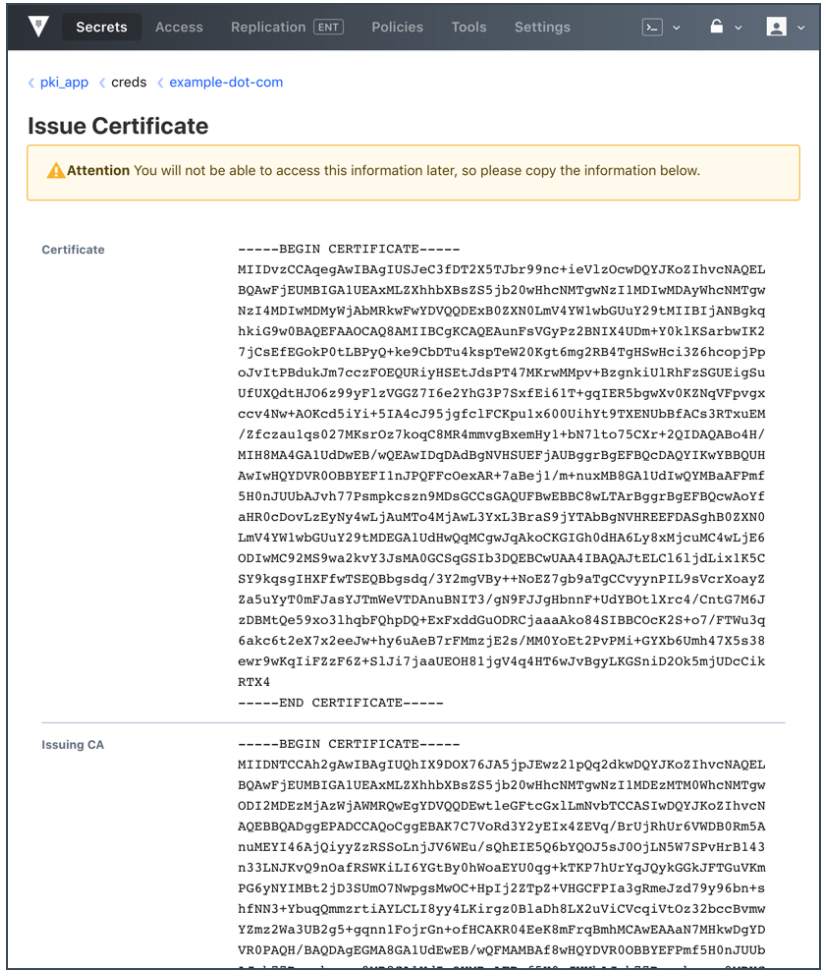
`allow_glob_domains` - Allows names specified in allowed_domains to contain glob patterns (e.g. ftp*.example.com)

Requesting A Dynamic Certificate

```
template {
     destination = "/etc/tls/safe.crt"
     contents = <<EOH
     {{ with secret "pki/issue/example-dot-com" "common_name=foo.example
     {{ .Data.certificate }}{{ end }}
     EOH
}
```

An application can request a valid certificate using the Vault Agent template stanza

Self Service Certificate Generation



Operators can generate certificates in a self service manner through the user interface.

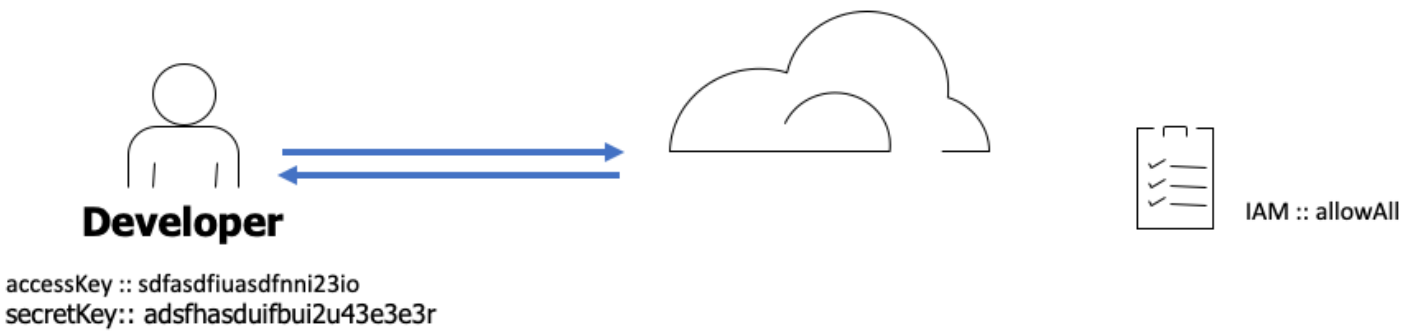
- Select the PKI Role
- Enter the common name
- Set the TTL
- Click Generate

Things Of Note For PKI Secret Engine

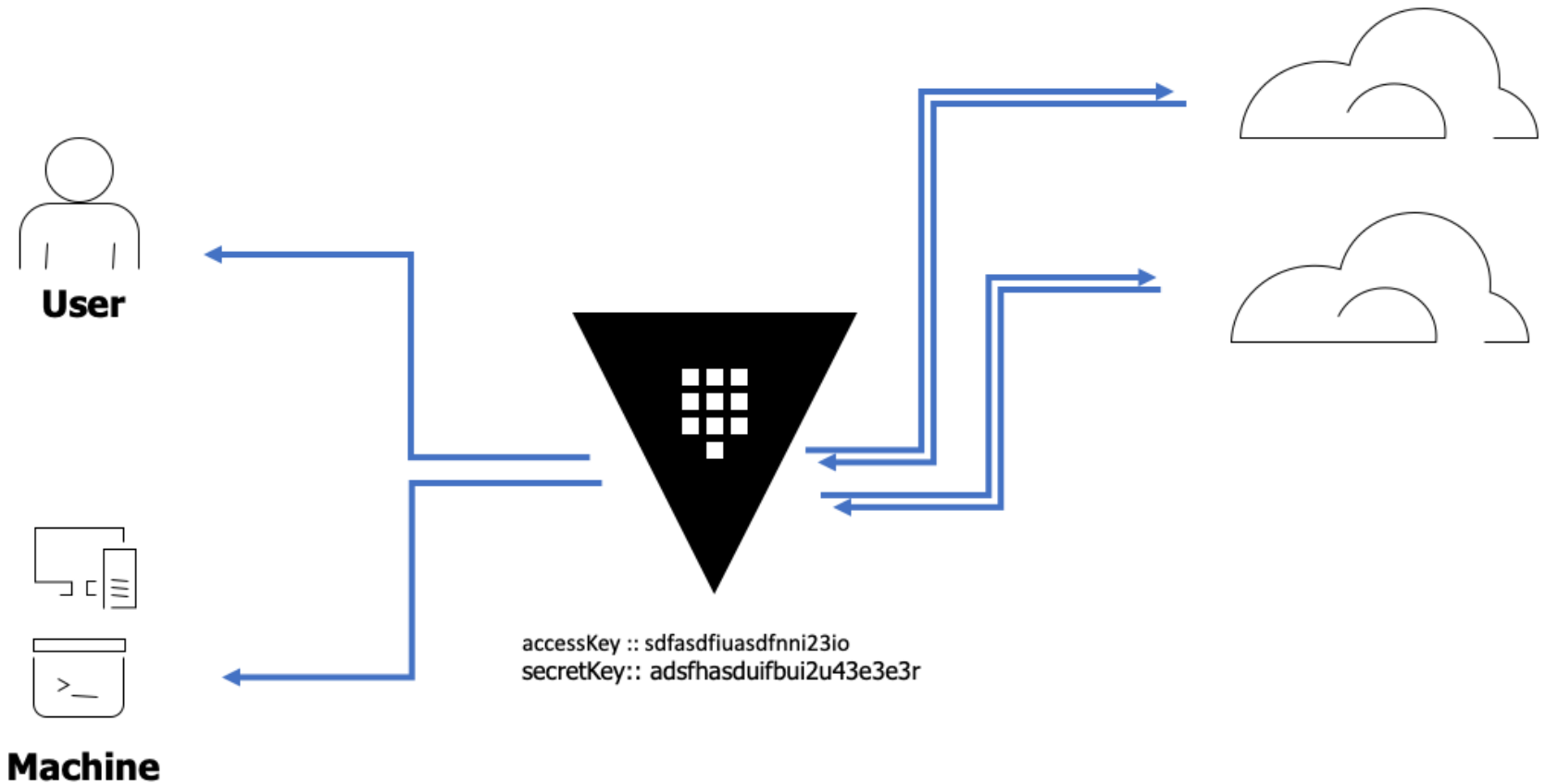
- While you can make vault a root CA, don't.
- Have a good process to handle certificate expiration
- Avoid heavy wildcards

Dynamic Cloud Credentials Secrets Engine

Cloud Credential Workflow



Dynamic Cloud Credential Workflow



Dynamic Cloud Credential Requirements

- Setup a privileged cloud account with the ability to create API keys
- IAM roles established for the generated accounts
- Can have many roles endpoints for granular access controls
  - Just for a single service
  - Just for a single account
  - Just for a single instance of a service
- Implementation will slightly vary based on cloud vendor capabilities

Dynamic Cloud Credentials - AWS

Dynamic Cloud Credential Configuration

```
$ vault write aws/config/root \
    access_key=AKIAJWVN5Z4FOFT7NLNA \
    secret_key=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 \
    region=us-east-1
```

This is the configuration for the vault account used to generate AWS API keys

access_key, secret_key: these are the API keys for the account

Using STS tokens still requires vault to use an IAM user that has permissions to the sts:GetFederationToken

Credential IAM Configuration

```
$ vault write aws/roles/my-role \
   policy_arns=arn:aws:iam::aws:policy/AmazonEC2ReadOnlyAccess,arn:aws:iam::aws:policy IAMRe
   credential_type=iam_user \
   policy_document=-<<EOF
       {
         "Version": "2012-10-17",
         "Statement": [
           {
             "Effect": "Allow",
             "Action": "ec2:*",
             "Resource": "*"
           }]}
       EOF
```

- Each role can have a discrete IAM Role either in line or with an ARN
- Using `policy_arns` defines the AWS IAM policies for generated credentials
- An inline policy can be defined using the `policy_document`

Generating Credential Request

```
$ vault read aws/creds/my-role
Key                Value
--                 --
lease_id           aws/creds/my-role/f3e92392-7d9c-09c8-c921-575d62fe80d
lease_duration     768h
lease_renewable    true
access_key         AKIAIOWQXTLW36DV7IEA
secret_key         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
security_token     <nil>
```

STS Federation Token

An STS federation token inherits a set of permissions that are the combination (intersection) of four sets of permissions:

- The permissions granted to the aws/config/root credentials
- The user inline policy configured in the Vault role
- The managed policy ARNs configured in the Vault role
- An implicit deny policy on IAM or STS operations.

Dynamic Cloud Credential - IAM Vault

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "ec2:*",
      "sts:GetFederationToken"
    ],
    "Resource": "*"
  }
}
```

The IAM policy the Vault user needs to have access to the federation token API

Dynamic Cloud Credential - STS Role

```
$ vault write aws/roles/ec2_admin \
    credential_type=federation_token \
    policy_document=@policy.json
```

The setup sets the `credential_type` as a federation token

Generate a Federated Token Credential

```
$ vault write aws/sts/ec2_admin ttl=60m

Key                Value
lease_id           aws/sts/ec2_admin/31d771a6-fb39-f46b-fdc5-945109106422
lease_duration     60m0s
lease_renewable    true
access_key         ASIAJYYYY2AA5K4WIXXX
secret_key         HSs0DYYYYYY9W81DXtI0K7X84H+OVZXK5BXXXX
security_token     AQoDYXdzEEwasAKwQyZUtZaCjVNDiXXXXXXXXgUgBBVUU...
```

STS AssumeRole

You can also allow vault to use sts:AssumeRole. Assuming a role cross accounts allow the generation of credentials for multiple accounts from a single vault account.

Prerequisites:

- An IAM role
- IAM inline policies and/or managed policies attached to the IAM role

STS Assume Role Vault Configuration

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "sts:AssumeRole",
    "Resource": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:role/RoleNameTo
  }
}
```

Configuration must have an IAM attached

Assume Role Target IAM Configuration

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:user/VAULT-AWS-R
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Target IAM must have trust established with the Vault account

Assume Role - Create Role

```
$ vault write aws/roles/deploy \
    role_arns=arn:aws:iam::ACCOUNT-ID-WITHOUT-HYPHENS:role/RoleNameToAss
    credential_type=assumed_role
```

Get credentials based on assumed role

Assume Role - Get Credentials

```
$ vault write aws/sts/deploy ttl=60m

Key                Value
lease_id           aws/sts/deploy/31d771a6-fb39-f46b-fdc5-945109106422
lease_duration     60m0s
lease_renewable    true
access_key         ASIAJYYYY2AA5K4WIXXX
secret_key         HSs0DYYYYYY9W81DXtI0K7X84H+OVZXK5BXXXX
security_token     AQoDYXdzEEwasAKwQyZUtZaCjVNDiXXXXXXXXgUgBBVUUbSyujLj...
```

Dynamic Cloud Credentials

Azure

Azure Specifics

Things to consider with Azure Dynamics Secrets

- Can use an existing SP or generate a new one
- Dynamic should be preferred over existing
- Easier for dynamic clean up

If Dynamic SP is used:

- An Azure role_id or role_name must be provided
- Groups can be used as well associated by group_name or object_id

Configure Secrets Engine Azure

```
$ vault write azure/config \
subscription_id=$AZURE_SUBSCRIPTION_ID \
tenant_id=$AZURE_TENANT_ID \
client_id=$AZURE_CLIENT_ID \
client_secret=$AZURE_CLIENT_SECRET

Success! Data written to: azure/config
```

If you are running Vault on a MSI enabled instance the client_id and client_secret will be inherited.

Azure Basic Role Setup

```
$ vault write azure/roles/my-role application_object_id=$EXISTING_APP_OB

$ vault write azure/roles/my-role ttl=1h azure_roles=-<<EOF
    [
        {
            "role_name": "Contributor",
            "scope":  "/subscriptions/$UUID/resourceGroups/Website"
        }
    ]
EOF
```

- A role can use an existing service principal or generate a new one.
- This sets up the Azure Role for the new AZ service principal

Azure Credential Request

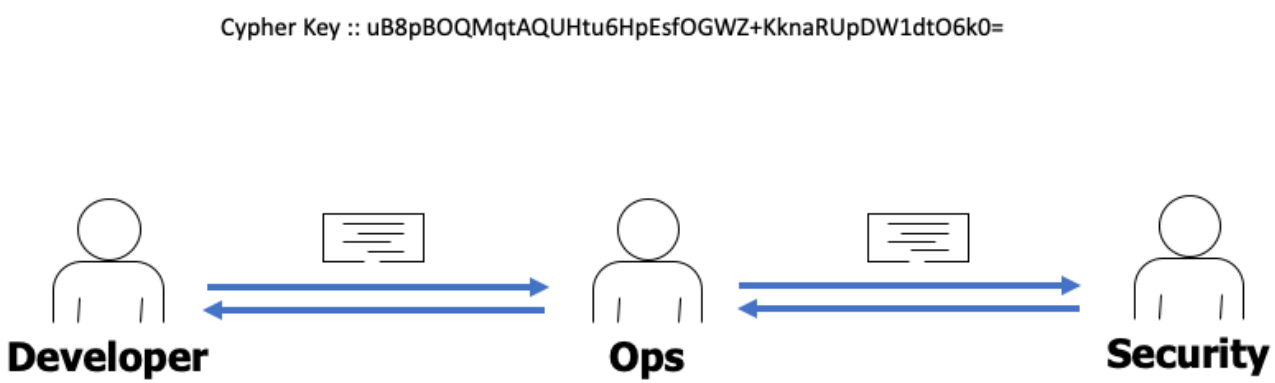
```
$ vault read azure/creds/my-role

Key                 Value
- -                  --
lease_id            azure/creds/sp_role/1afd0969-ad23-73e2-f974-962f7ac1c
lease_duration      60m
lease_renewable     true
client_id           408bf248-dd4e-4be5-919a-7f6207a307ab
client_secret       ad06228a-2db9-4e0a-8a5d-e047c7f32594
```

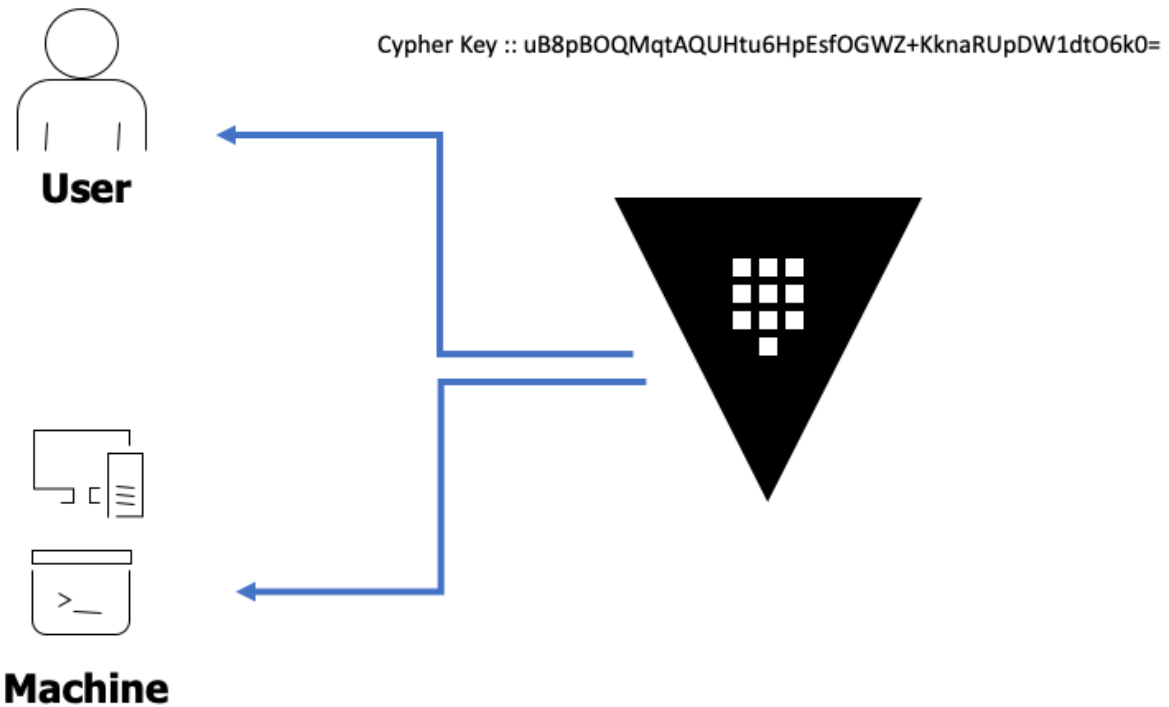
To get cloud credentials simply read from the the endpoint

Dynamic Encryption Keys

Encryption Keys Workflow



Dynamic Encryption Key Workflow



Enable Engine

Enable the transit engine

```
$ vault secrets enable transit
```

Create a data key decryption endpoint

```
$ vault write -f transit/keys/orders
```

Generate A Dynamic DataKey

Generate data key by hitting the data key end point for orders

```
$ vault write -f transit/datakey/plaintext/orders

Key           Value
--            --
ciphertext    vault:v1:Aj7SUcmCAEiBhNhYBZkXKzO4kuOIDCwDx6FX ...
plaintext     uB8pBOQMqtAQUHtu6HpEsfOGWZ+KknaRUpDW1dtO6k0=
```

When you request a key you will get the plaintext key along with a key encrypted by the orders transit key. This allows for the key to be stored securely with the data

Retrieving the DataKey

To decrypt the data key the application has to submit the cypher text version of the key to vault to get the plaintext key.

```
$ vault write transit/decrypt/orders ciphertext="vault:v1:cZNHVx+sxdMErX

Key          Value
---          -----
plaintext    Y3JlZGl0LWNhcmQtbnVtYmVyCg==
```

Things To Note

- When a datakey is requested it is not stored in vault
- Every time and endpoint is hit a new key is created
- You can have several datakey endpoints adding another level of encryption

Chapter Summary

- Dynamic Secrets are point at request time credentials
- They have a Lease ID, Lease Duration, Renewable properties and tunable configuration
- Dynamic Secrets include;
  - Database Creds
  - PKI and SSH Certificates
  - Cloud Access Creds

Reference links

- Continue Learning about Dynamic Secrets
- Database Secrets
- PKI Certificate

Vault Dynamic Secrets Module Complete!