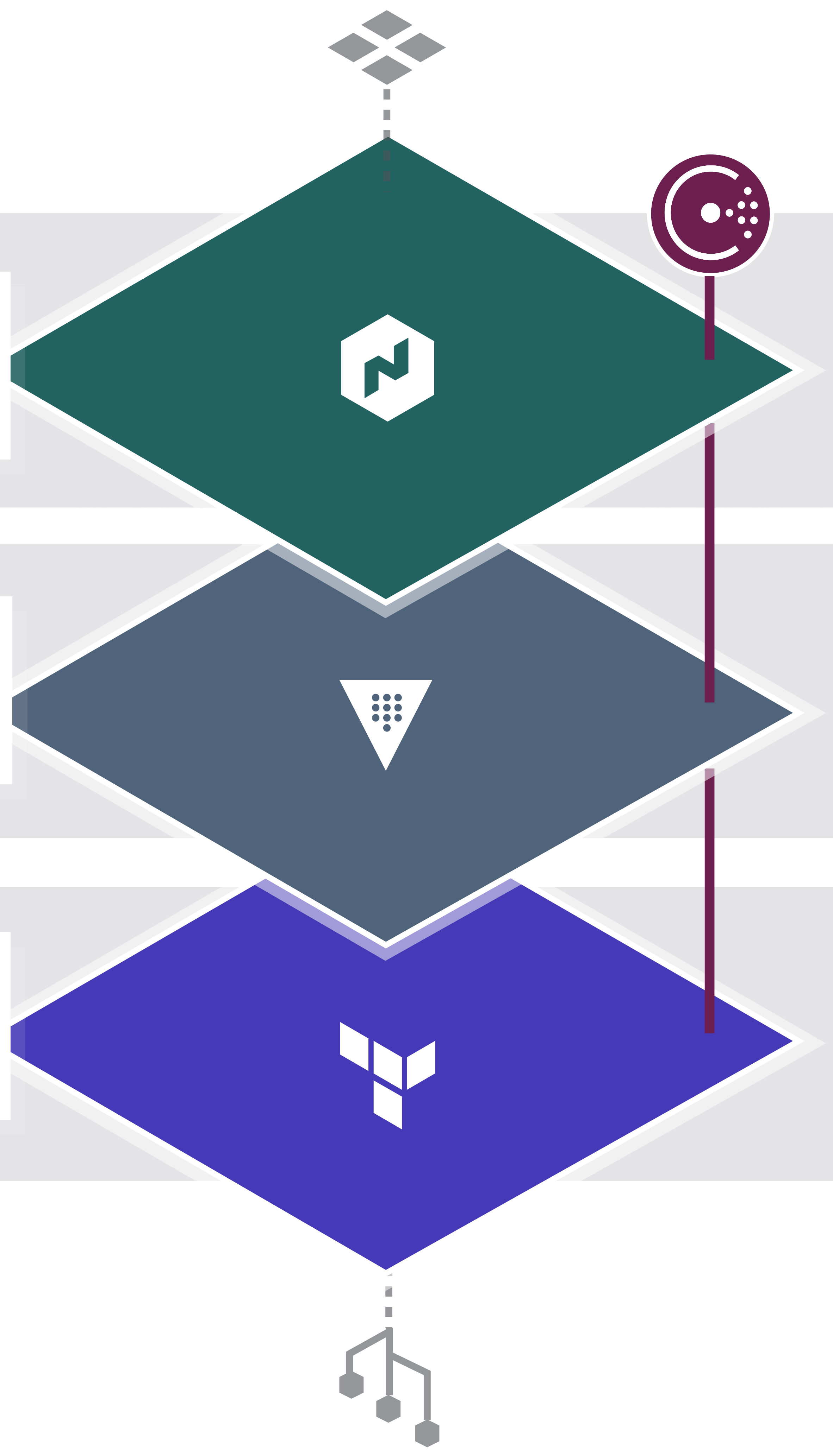
HashiCorp Vault

ARMON DADGAR

@armon

PROVISION, SECURE AND RUN ANY INFRASTRUCTURE



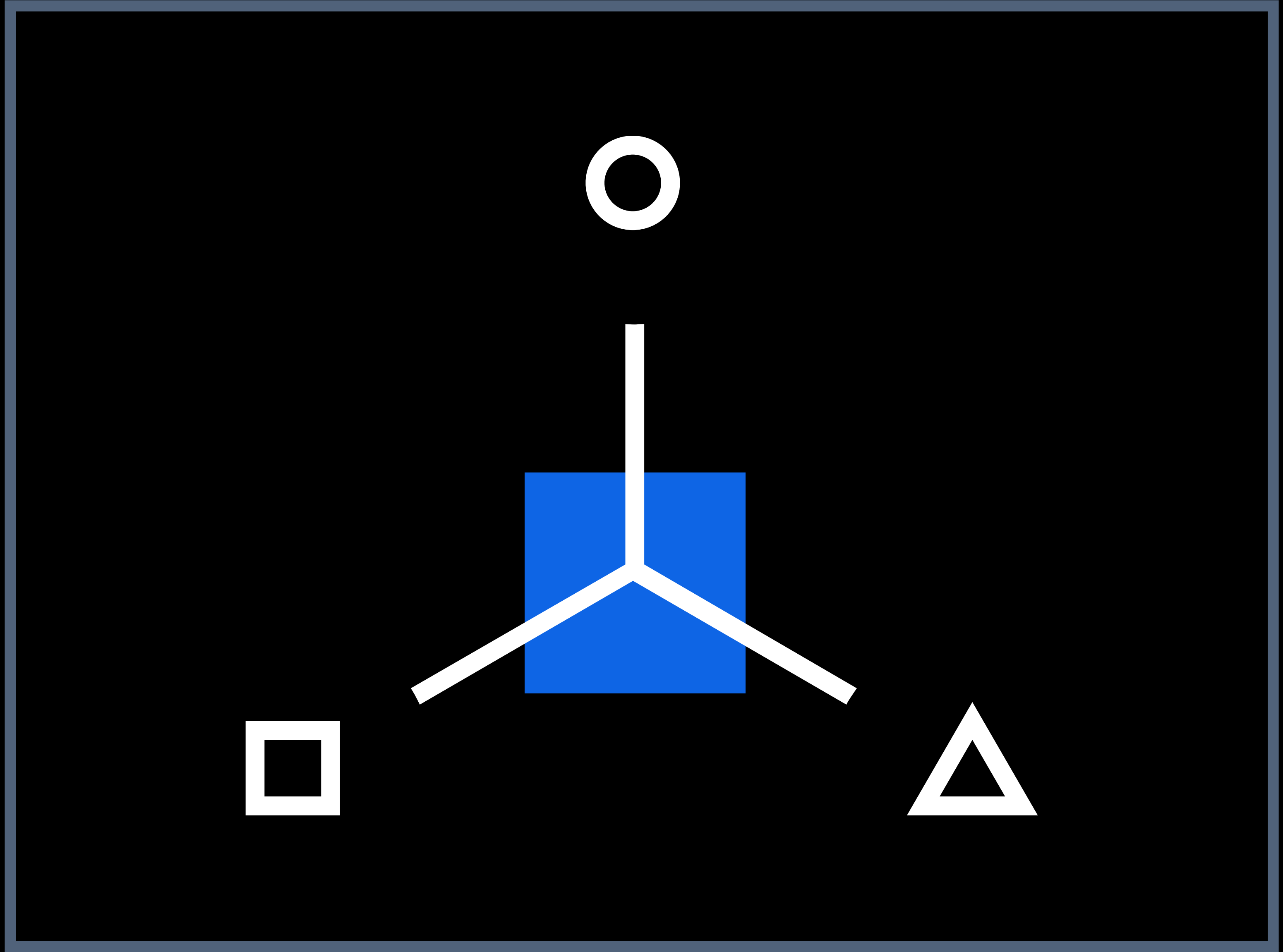
	OSS TOOL SUITE	PRODUCT SUITE
RUN Applications	Nomad | Consul	Consul Enterprise Nomad Enterprise
SECURE Application Infrastructure	Vault	Vault Enterprise
PROVISION Infrastructure	Vagrant | Packer | Terraform	Terraform Enterprise
	FOR INDIVIDUALS	**FOR TEAMS**

AGENDA

- USE CASES
- INTRO TO VAULT
- IDENTITY IN 0.9
- K8S INTEGRATION

USE CASES

Secrets Management



Eliminate Secret Sprawl

Encryption as a service

Securely Store Data

Identity & Access Management

Govern Secret Access

SECRET MANAGEMENT

What is a “Secret”?

- Secret is anything used for authentication or authorization
  - User/Pass, API token, TLS certificate, etc
- Sensitive is anything that is confidential
  - SSN, Credit Card, Email, etc

Questions in Secret Management

- How do applications get secrets?
- How do humans acquire secrets?
- How are secrets updated?
- How is a secret revoked?
- When were secrets used?
- What do we do in the event of compromise?

State of the World

- Secret Sprawl
- Decentralized keys
- Limited visibility
- Poorly defined “break glass” procedures

VAULT INTRO

Vault Goals

- Single Source for Secrets
- Programmatic application access (Automated)
- Operator access (Manual)
- Practical security
- Modern datacenter friendly

Vault Features

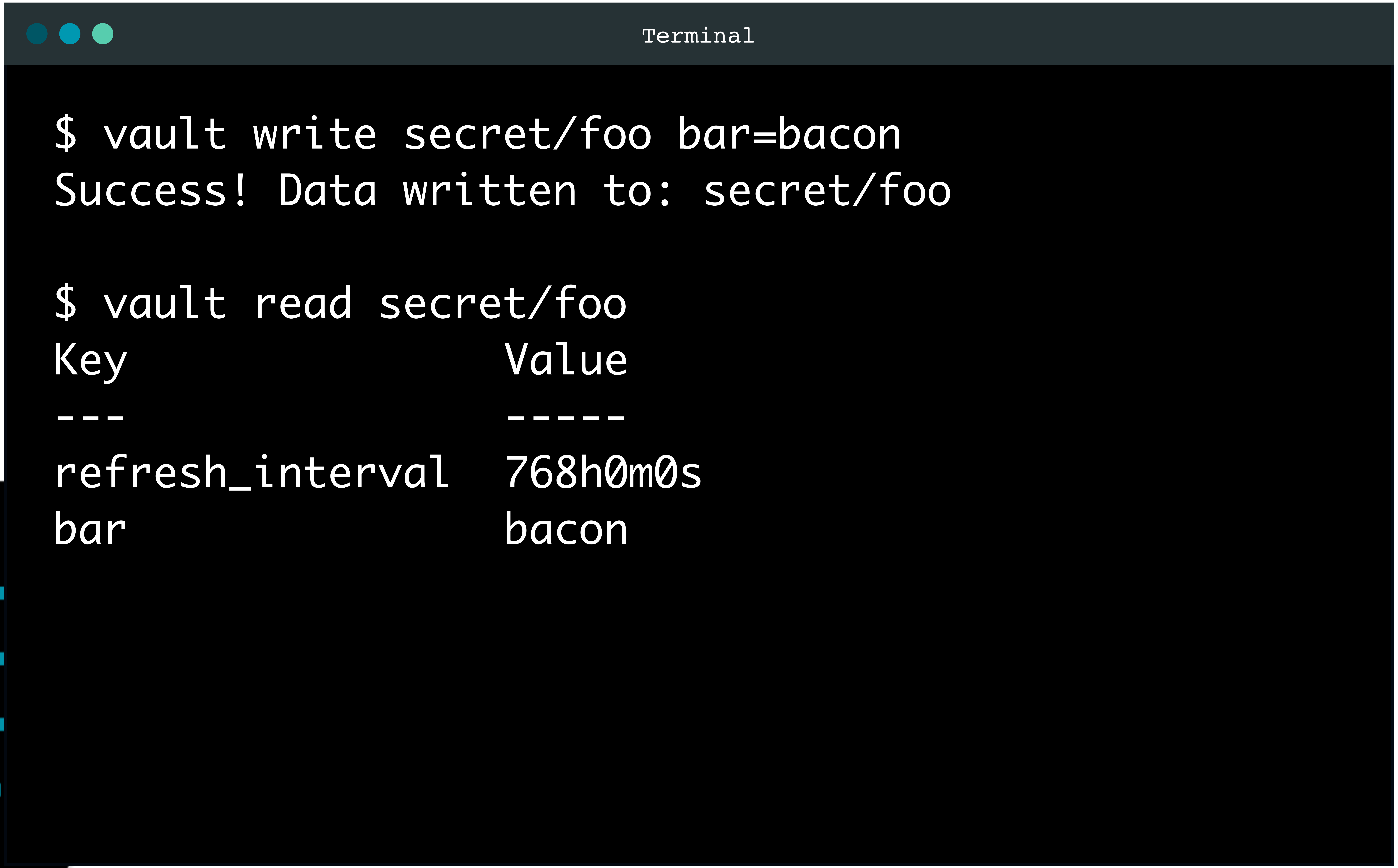
- Secure secret storage (in-memory, Consul, file, RDBMS, and more)
- Dynamic secrets
- Leasing, renewal, and revocation
- Auditing
- Rich ACLs
- Multiple client authentication methods

Security Principles

- Confidentiality, Integrity, Availability
- Least Privilege (Need to know access)
- Privilege Separation (Separation of Controls)
- Privilege Bracketing (Time limiting access)
- Non-Repudiation (Can’t Deny Actions)
- Defense in Depth

Secure Secret Storage

- Data is encrypted in transit and at rest
- 256bit AES in GCM mode
- TLS 1.2 for clients
- No HSM required
- Provides Confidentiality and Integrity

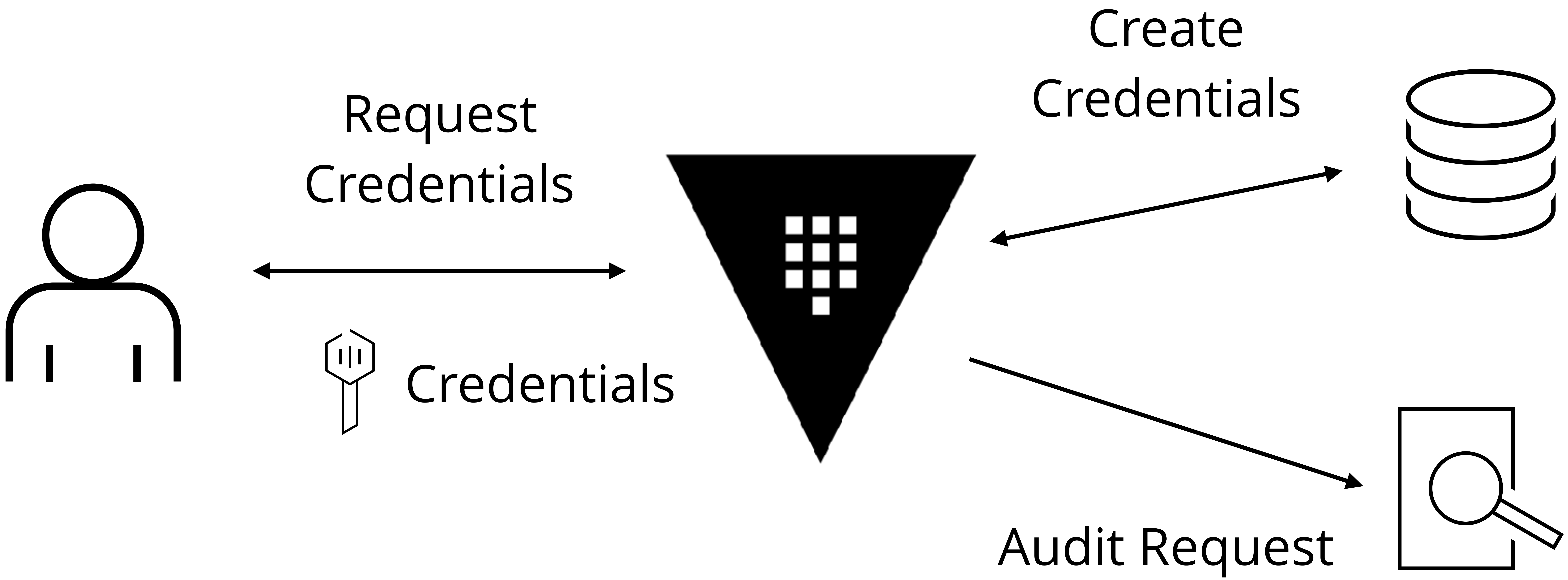


```
$ vault write secret/foo bar=bacon
Success! Data written to: secret/foo

$ vault read secret/foo
Key                 Value
---                 -----
refresh_interval    768h0m0s
bar                 bacon
```

Dynamic Secrets

- Never provide “root” credentials to clients
- Provide limited access credentials based on role
- Generated **on demand** when requested
- Leases are enforceable via revocation
- Audit trail can identify point of compromise


Request Credentials
Credentials
Create Credentials
Audit Request

Dynamic Secrets

- Pluggable Backends
- Grow Support with time
- Split from Core (Defense in Depth)

- AWS
- Consul
- MSSQL
- MySQL
- PostgreSQL
- Cassandra

- MongoDB
- RabbitMQ
- Transit
- PKI
- SSH
- Custom

Leasing, Renewal, Revocation

- Every dynamic secret (and token) has a lease
- Secrets are revoked at the end of the lease unless renewed
- Secrets may be revoked early by operators
  - “Break Glass” procedure
  - Enforceable leases

Why Leasing?

- Privilege Bracketing
- Non-Repudiation
- Secret Updates bounded
- Enables Revocation

Authenticating, Authorizing, Auditing (AAA)

- Authorization backends
  - Machine Oriented (TLS, Tokens, AppRole)
  - User Oriented (User/Pass, LDAP, GitHub)
- Rich ACLs for Authorization
  - Default Deny - “need to know” (Least Privilege)
- Request/Response auditing, fail closed (Non-Repudiation)

Highly Available

- Active/Standby model
  - Consul used for leader election
  - Automatic failover
- Multi-Datacenter replication (Premium)
- Disaster Recovery replication (Pro)

Unsealing Vault

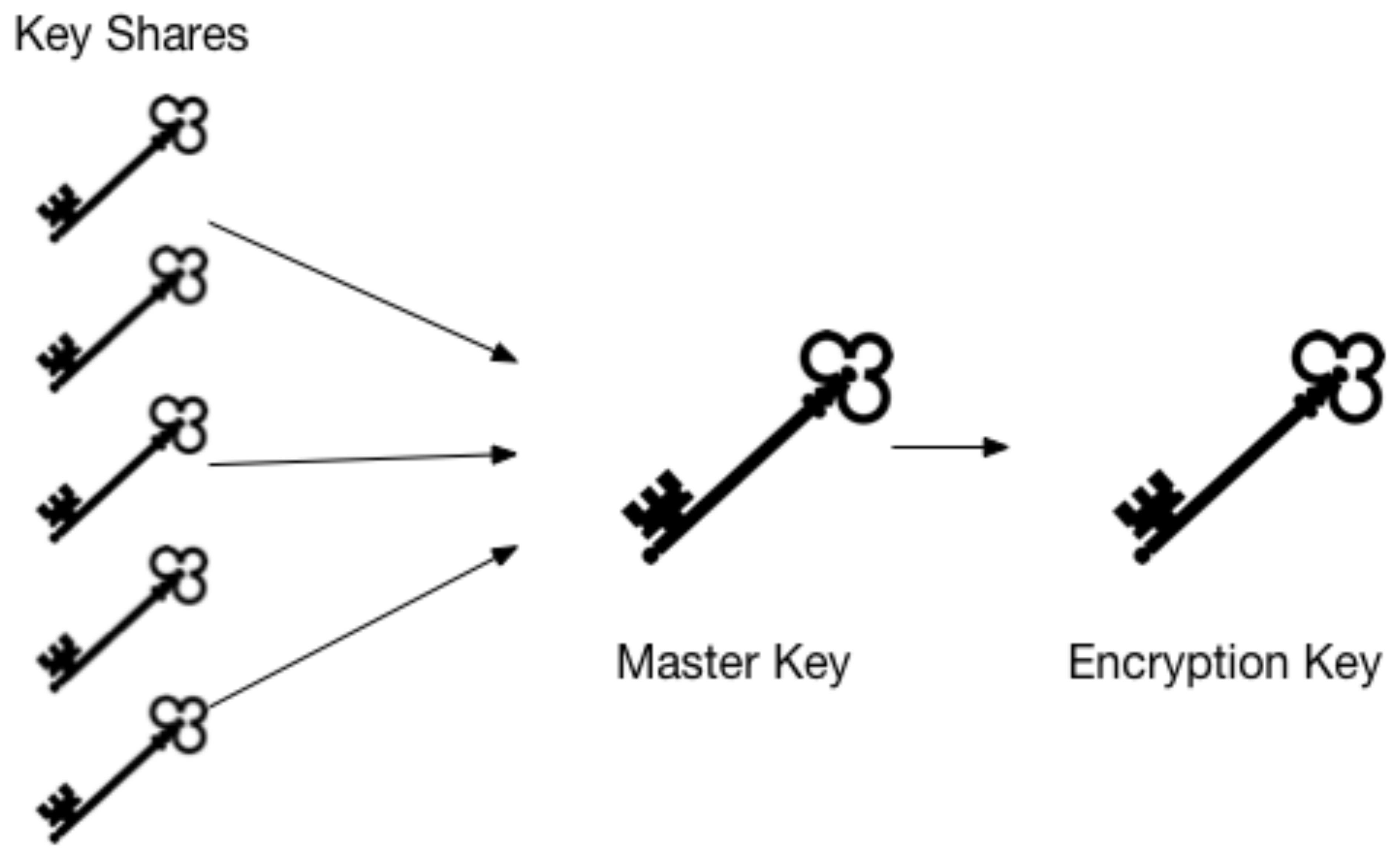
- Data at rest is encrypted
- Requires decryption key
- Must be provided *online*

Protecting the keys

- Master key is the “key to the kingdom”
- All data could be decrypted
- Protect against insider attack
- “Two-Man Rule”

Shamir Secret Sharing

- Protect Encryption Key with Master Key
- Split Master Key into **N** shares
- **T** shares required
- Quorum of key holders to unseal
  - Default **N**:5, **T**:3
- No Access (Privilege Separation)



Automated Unsealing

- Shamir Secret Splitting for Human Key Holders
- Automated Unsealing with Machine Key Holders
- Hardware Devices, HSM
- Cloud Key Management Systems

Summary

- Solve the “Secret Sprawl” problem
- Protects against insider threats (ACLs and Secret Sharing)
- Protects against external threats (Cryptosystem)
- Applies Security Principles

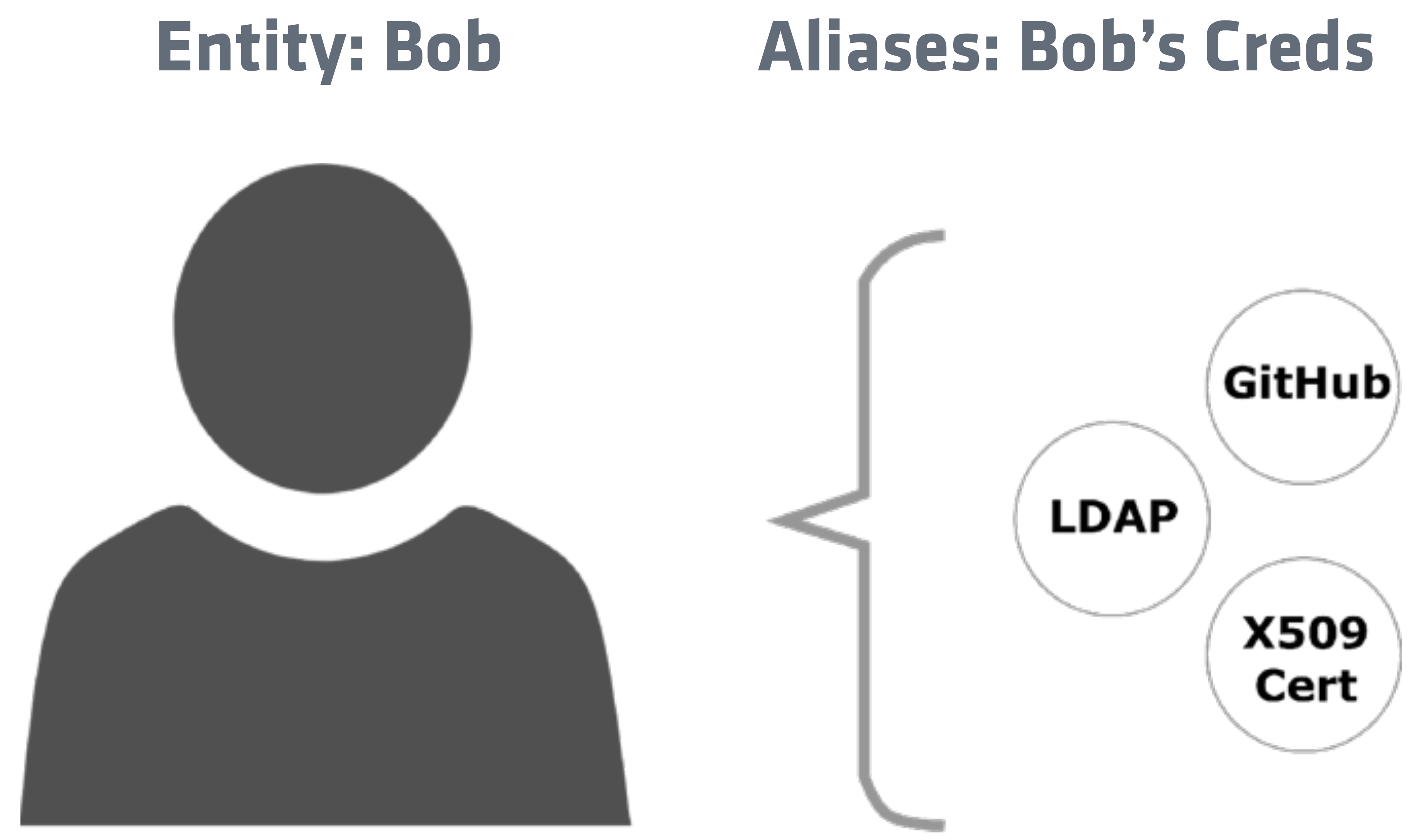
IDENITITY

NEW IN 0.9

ENTITIES & ALIASES

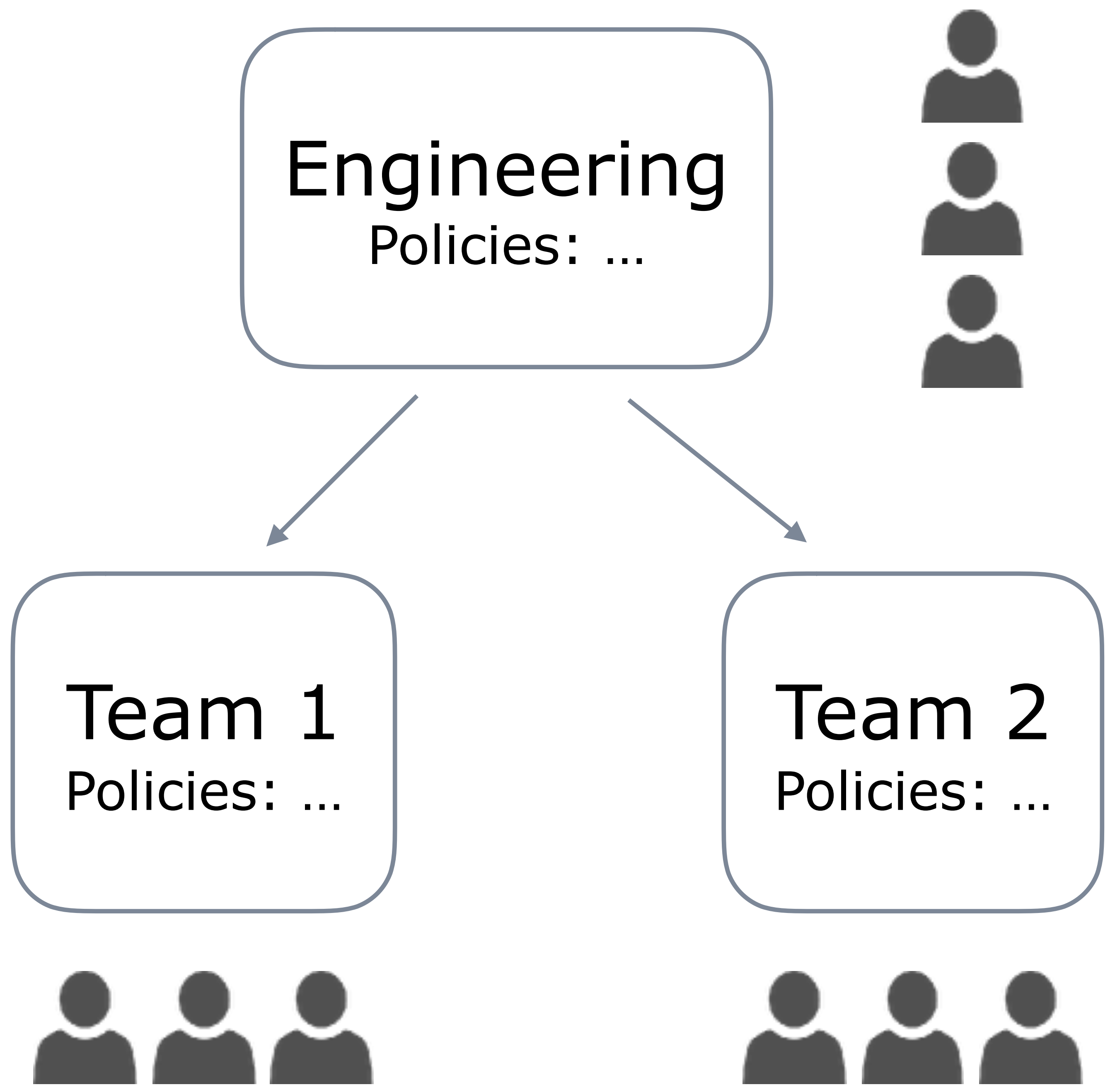
- **Entity**:A representation of a single person or system that is consistent across logins.
  - Can assign policies and metadata to an Entity

- **Alias**: Mapping between an Entity and auth backend.



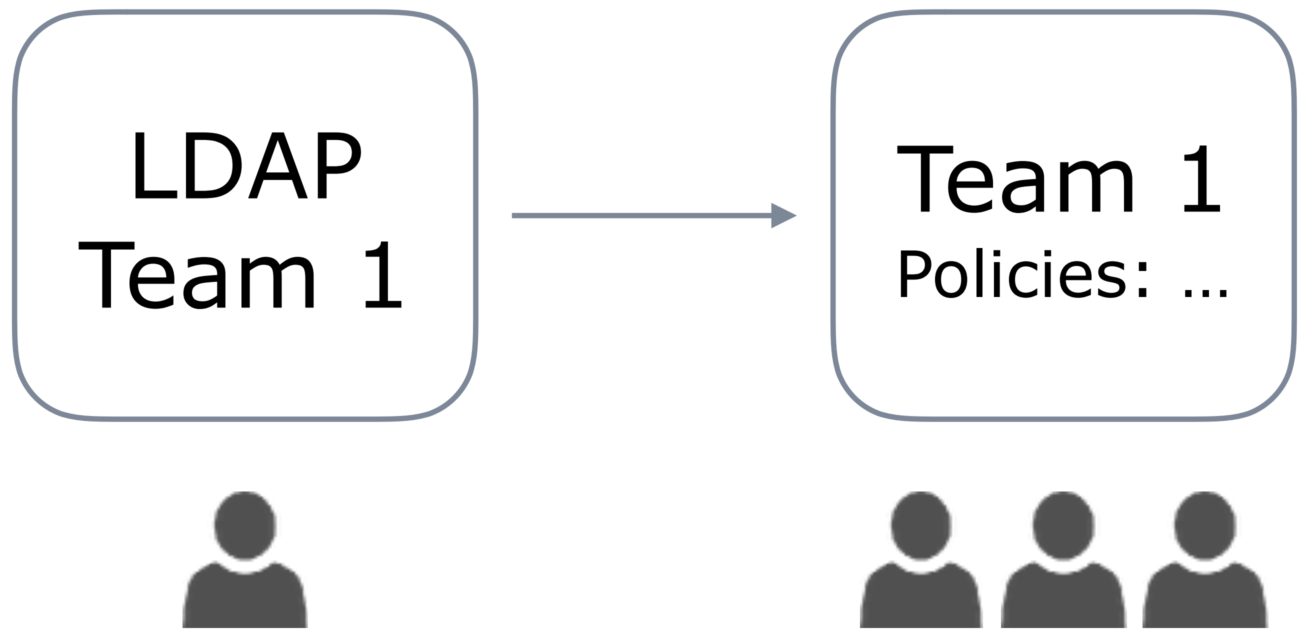
IDENTITY GROUPS

- Manage **entities** by placing them into **groups**
- Groups can contain both entities and other groups
- Groups can have a set of policies and metadata that is inherited by the member entity or subgroups.



IDENTITY GROUPS ALIASES

- Mapping from internal group to an external group in an third party authentication provider
- If part of external group: automatically add user to the Identity Group, inheriting the policies and metadata
- Currently works with LDAP, Github, and Okta auth backends



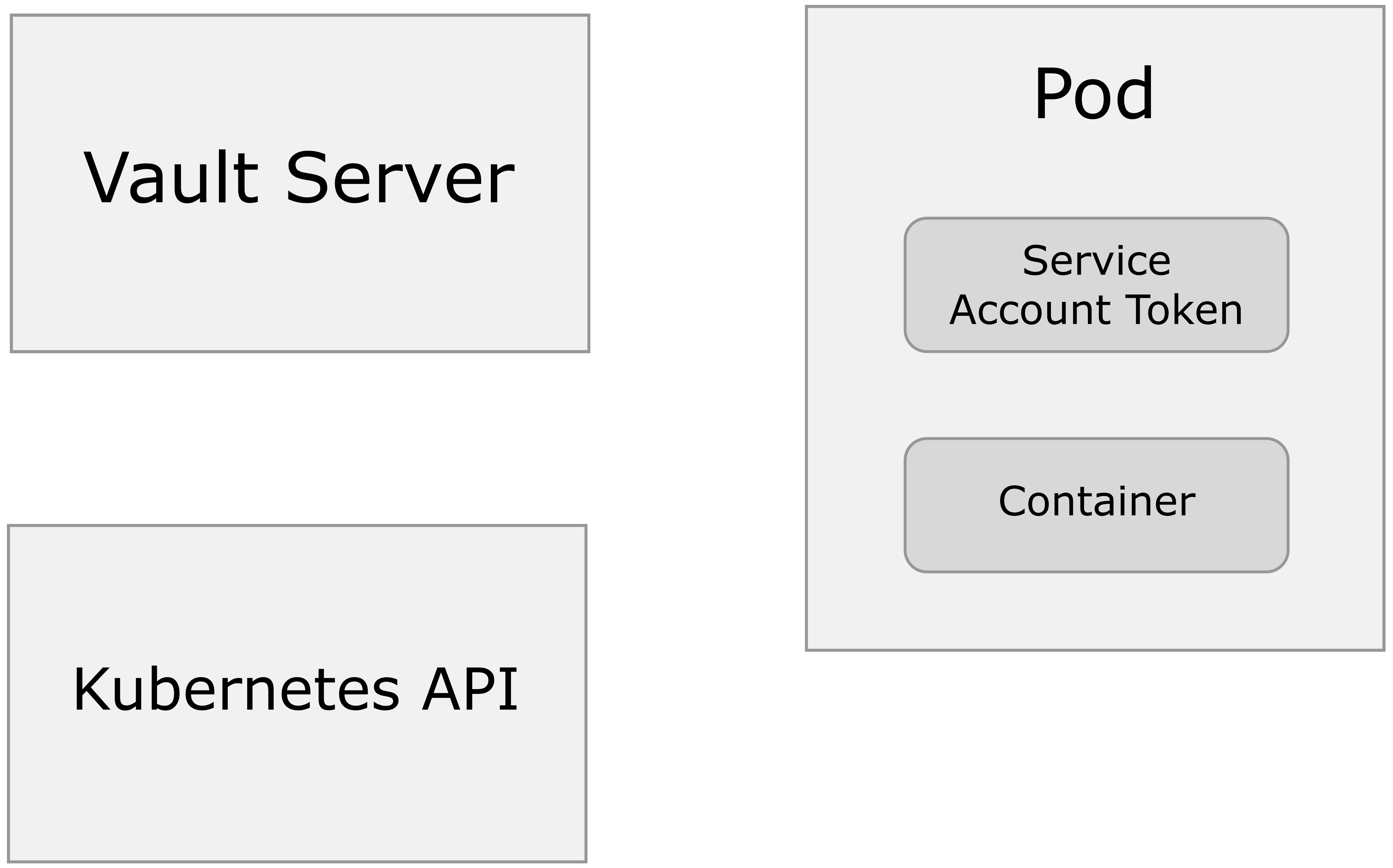
K8S AUTH BACKEND

NEW IN 0.8.3

KUBERNETES AUTH BACKEND

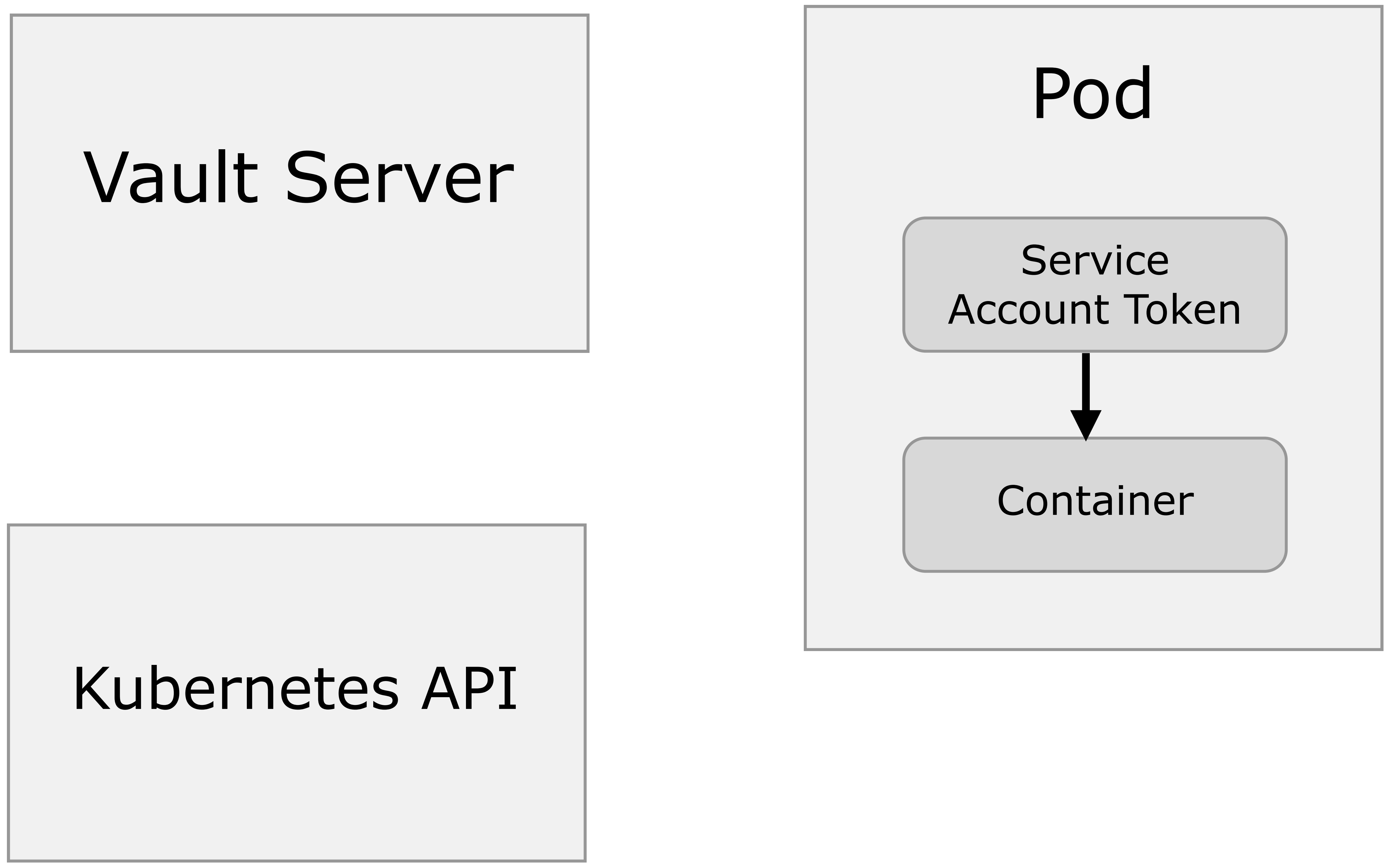
- Binds Kubernetes service accounts to a Vault role
- Uses the TokenReview API to validate JWTs
- Doesn’t need any external pieces

KUBERNETES AUTH BACKEND



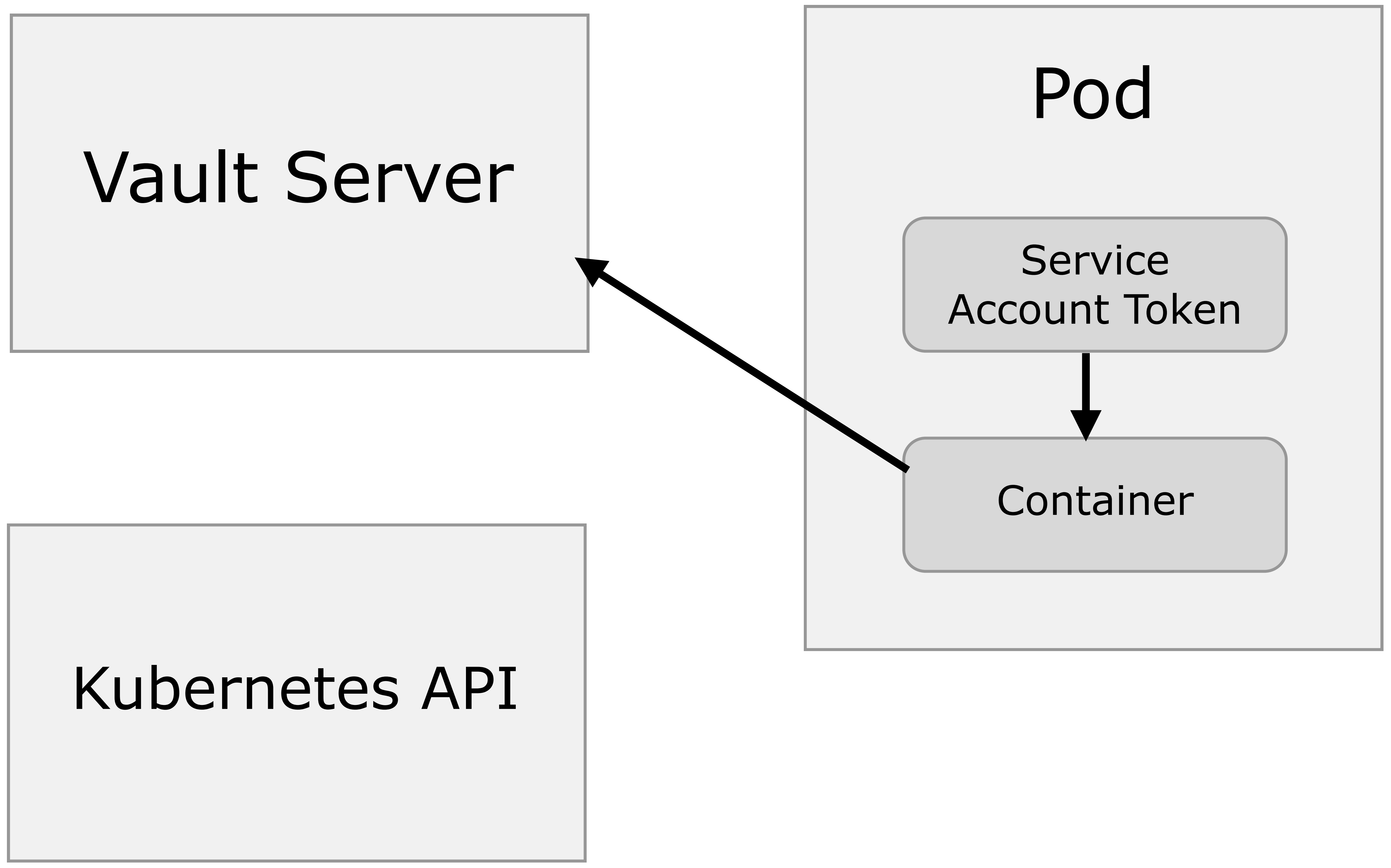
KUBERNETES AUTH BACKEND

1. App reads Service Account JWT



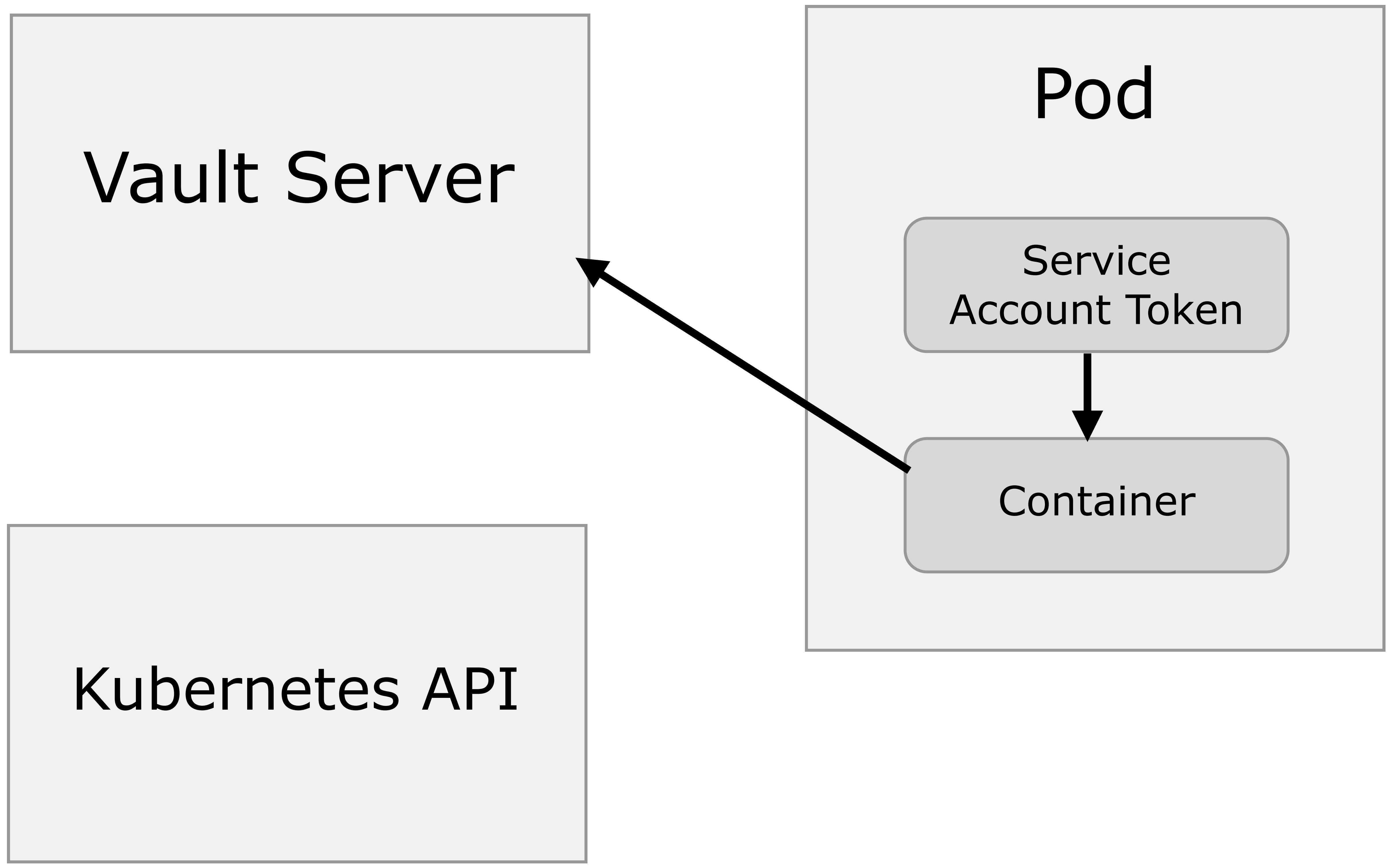
KUBERNETES AUTH BACKEND

1. App reads Service Account JWT
2. App sends JWT and role name to Vault



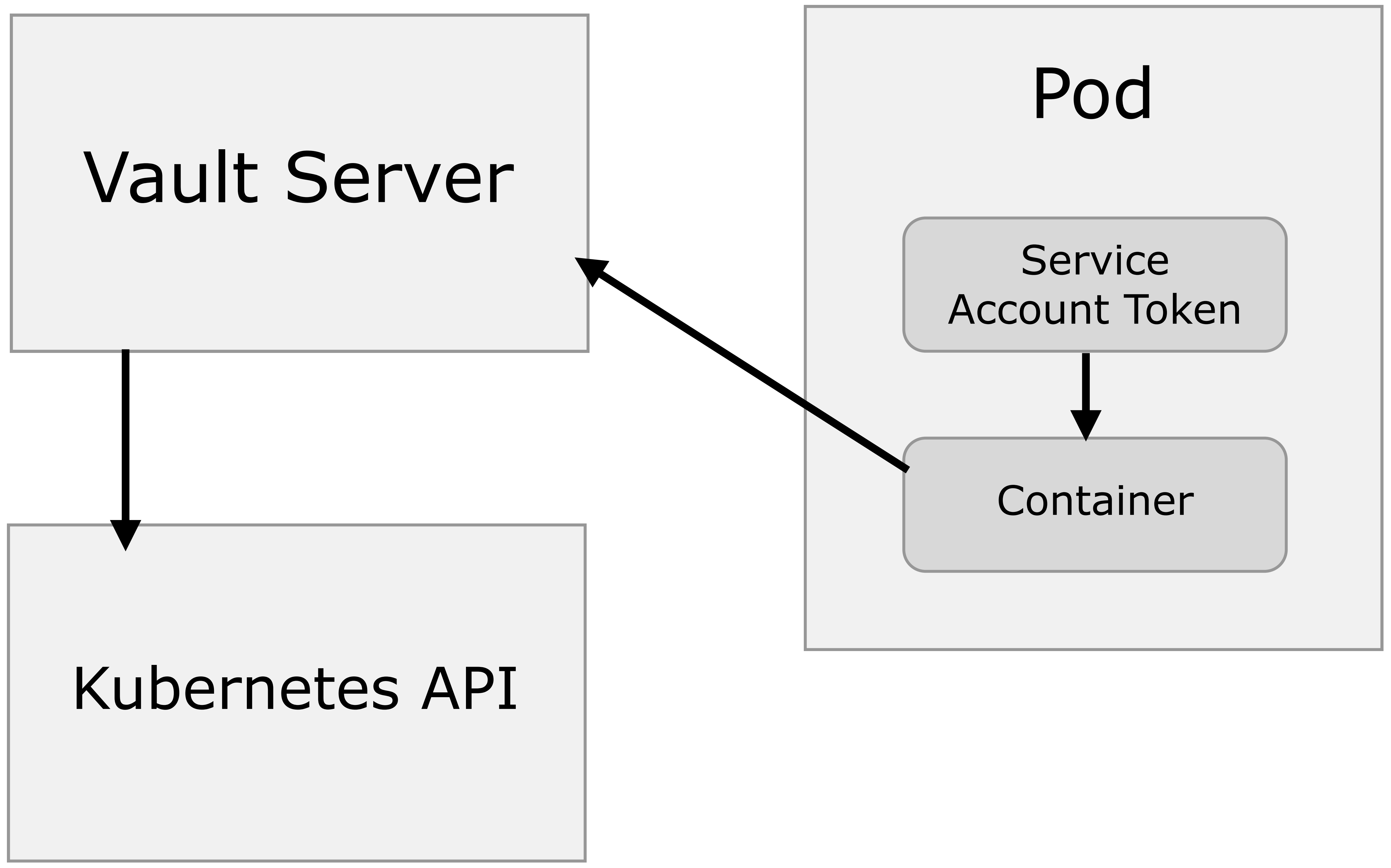
KUBERNETES AUTH BACKEND

1. App reads Service Account JWT
2. App sends JWT and role name to Vault
3. Vault validates JWT signature and verifies S.A. is authorized to use the role



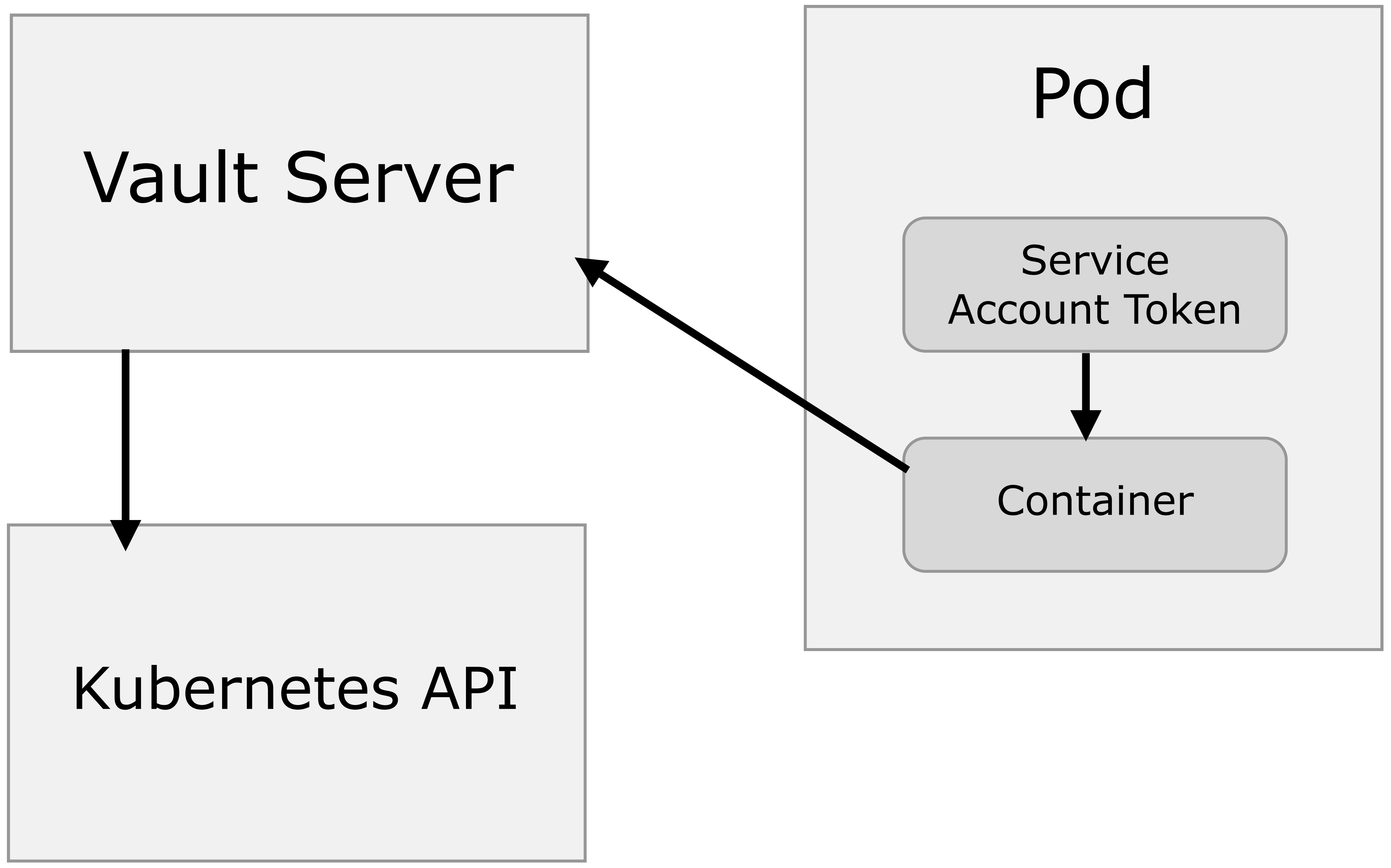
KUBERNETES AUTH BACKEND

1. App reads Service Account JWT
2. App sends JWT and role name to Vault
3. Vault validates JWT signature and verifies S.A. is authorized to use the role
4. Vault calls into the TokenReview API



KUBERNETES AUTH BACKEND

1. App reads Service Account JWT
2. App sends JWT and role name to Vault
3. Vault validates JWT signature and verifies S.A. is authorized to use the role
4. Vault calls into the TokenReview API
5. Vault returns token



Configuration

Kubernetes Auth Backend

Terminal

```
$ vault auth-enable kubernetes

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

Create a Role

Kubernetes Auth Backend

Terminal

```
$ vault write auth/kubernetes/role/demo \
    bound_service_account_names=vault-auth \
    bound_service_account_namespaces=default \
    policies=kube-auth
    period=60s
```

Login

Kubernetes Auth Backend

Terminal

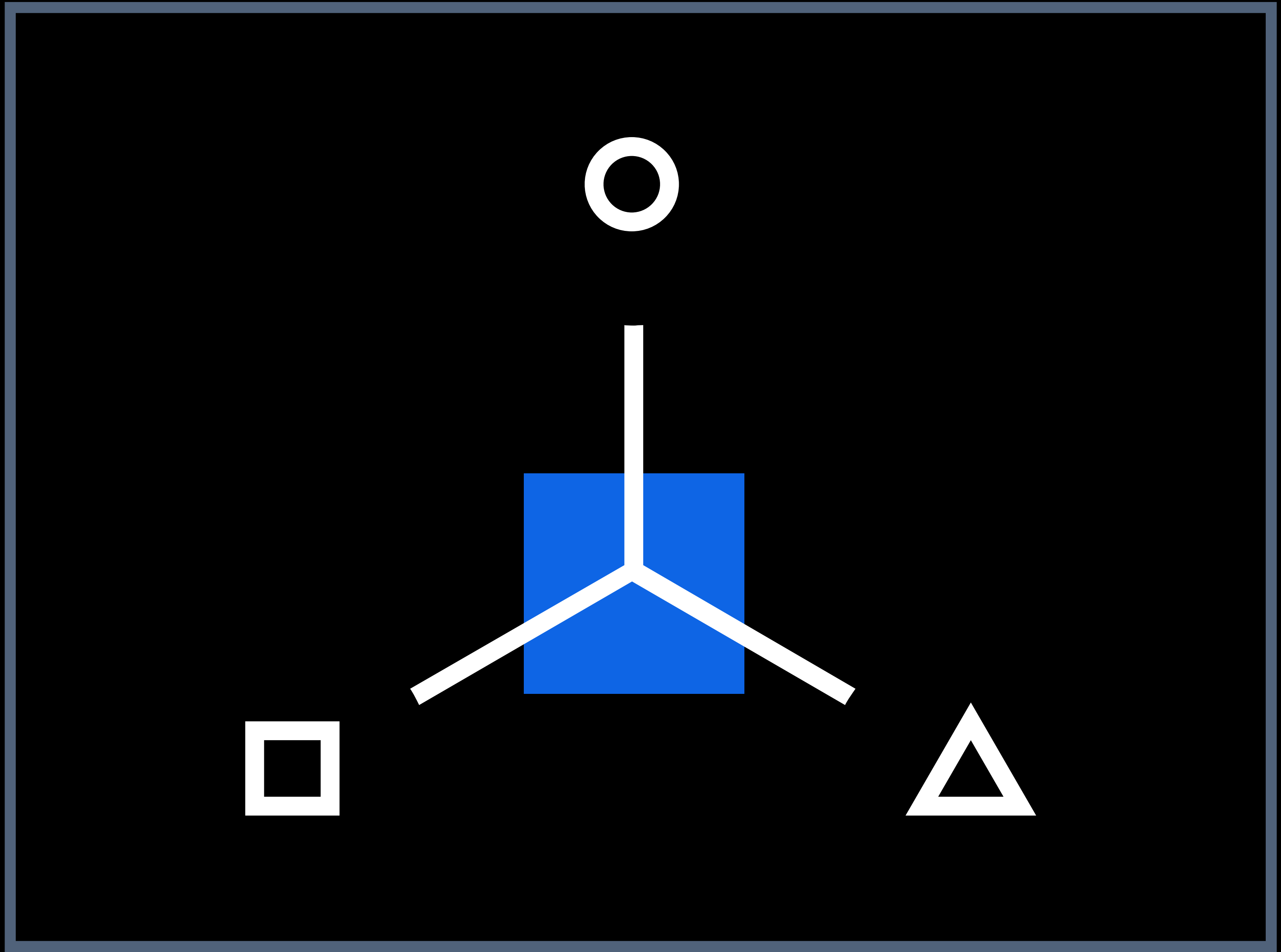
```
$ vault write auth/kubernetes/login \
     role=demo \
     jwt="…"

Key                                    Value
---                                    -----
token                                  19ca864c-7fdd…
token_accessor                         4169f43e-b4ec…
token_duration                         768h0m0s
token_renewable                        true
token_policies                         [default, kube-auth]
token_meta_role                        "demo"
token_meta_service_account_name        "vault-auth"
token_meta_service_account_namespace   "default"
token_meta_service_account_uid         "d77f89bc-9055…"
```

SUMMARY

USE CASES

Secrets Management



Eliminate Secret Sprawl

Encryption as a service

Securely Store Data

Identity & Access Management

Govern Secret Access

Vault Intro

- Modern Secrets Management
- Native K8S Integration
- Lots of features we didn’t cover (Dynamic SSL/TLS, SSH Brokering, MFA, etc)

HashiCorp

Thank you!

We are hiring: https://www.hashicorp.com/jobs