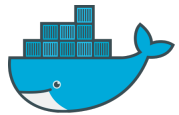
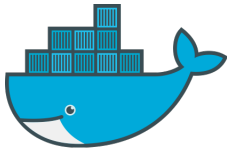
MTLS in a Microservices 🌍

@diogomonica





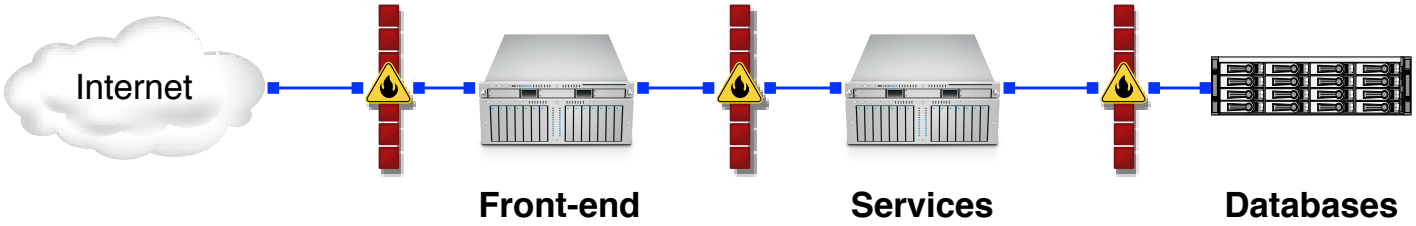
- Infrastructure Company
- Moving 1+ billion containers annually
- Infra: Go, some python

- Mobile payments company
- Moving $48 billion annually
- Infra: Java & Ruby, some Go

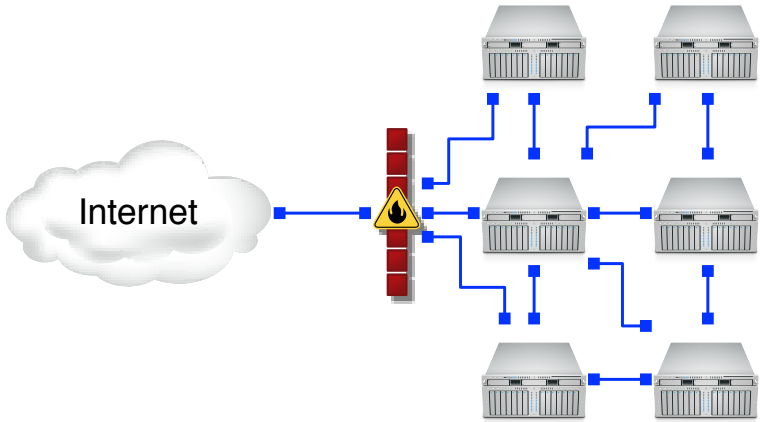
Microservice Security Goals

- Provide common security infrastructure
- Follow the principle of least-privilege
- Better security monitoring
- Automated end-to-end secure service deployment

The Security boundary is the service

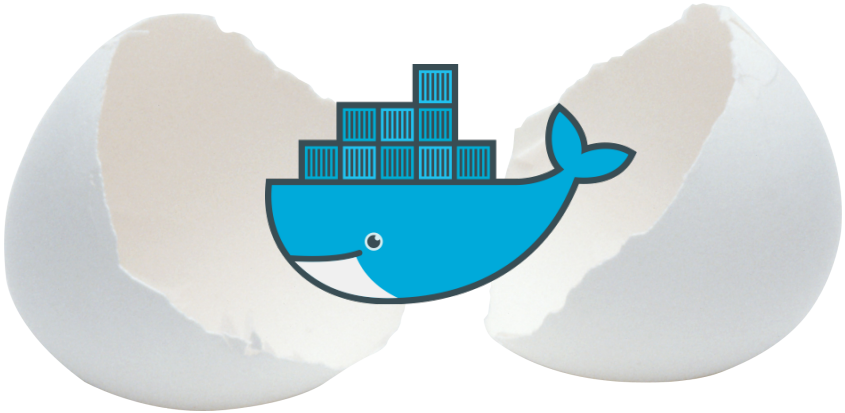


VS

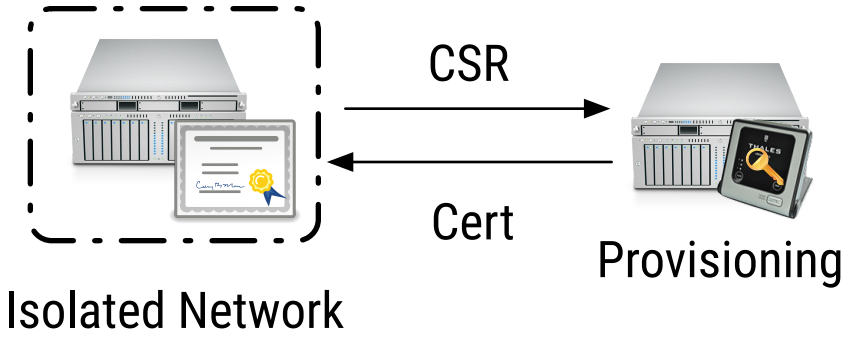


Every service call should be authorized and authenticated

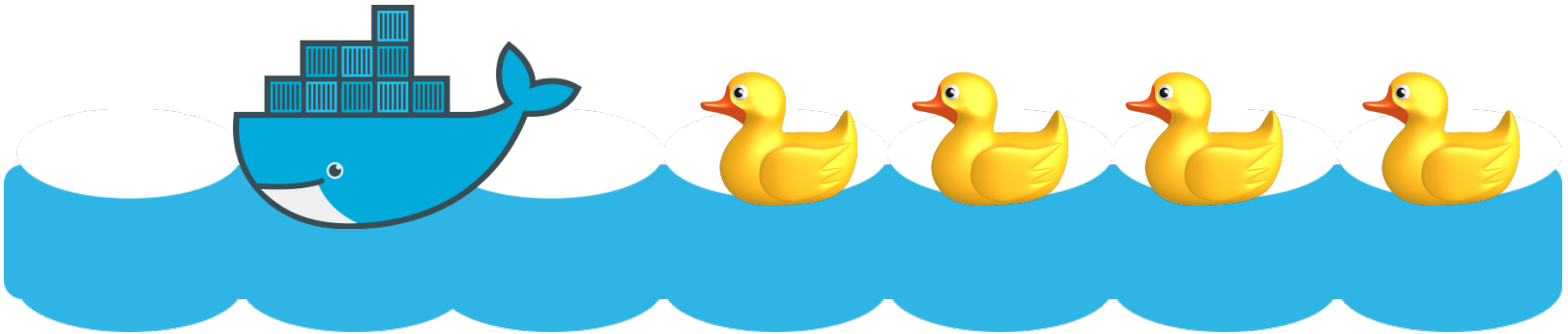
One node, one ID



Your provisioning system is your Registration Authority


CSR
Cert
Isolated Network
Provisioning

Trust on first use



MTLS

Supported by everything

Key-material stays secret

MTLS

Confidentiality

Integrity

Authentication

Key-material stays secret

Supported by everything

Confidentiality

Integrity

MTLS

Authentication

Running a PKI

Confusing for Engineers

Unforgiving

A LOT of Certs!

Revocation?

Key-material
stays secret

Supported by
everything

Authentication

Encryption

MTLS

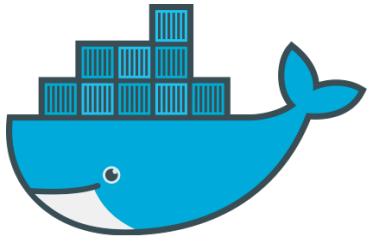
Running a PKI

Confusing
for Engineers

Unforgiving

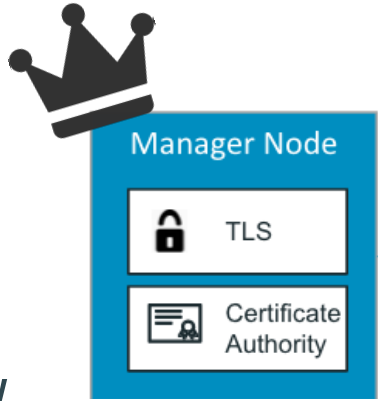
A LOT of Certs!

Revocation?



SWARM

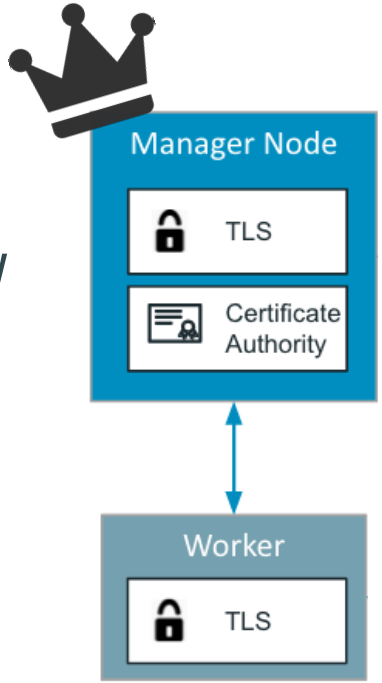
Mutual TLS by default



- First node generates a new self-signed CA.

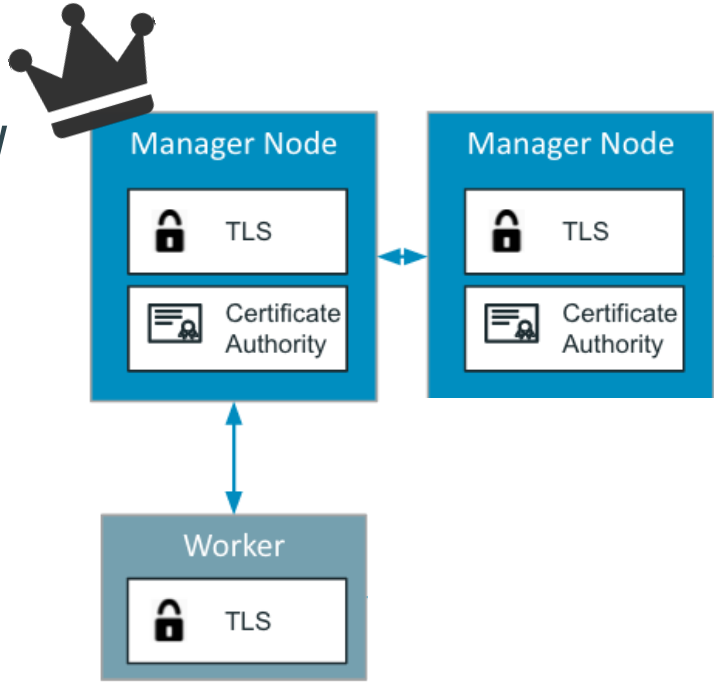
Mutual TLS by default

- First node generates a new self-signed CA.
- New nodes can get a certificate issued w/ a token.



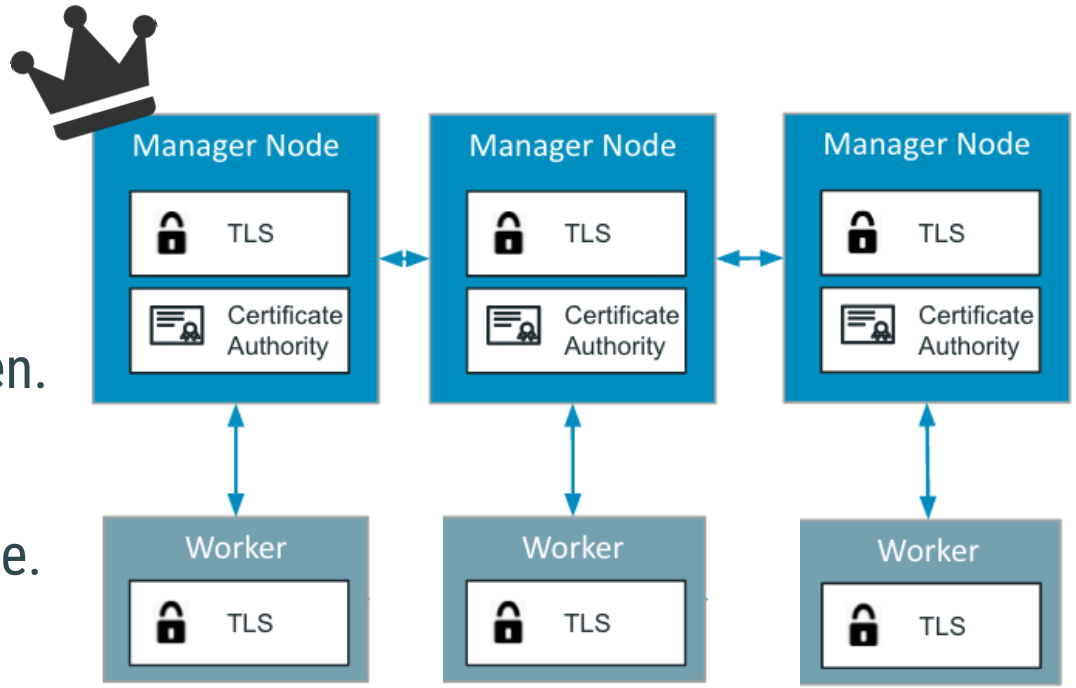
Mutual TLS by default

- First node generates a new self-signed CA.
- New nodes can get a certificate issued w/ a token.
- Workers and managers identified by their certificate.



Mutual TLS by default

- First node generates a new self-signed CA.
- New nodes can get a certificate issued w/ a token.
- Workers and managers identified by their certificate.
- Communications secured with Mutual TLS.



The Token

Token Version

SWMTKN-1-mx8susrv1etsmc8omaom825bet6-cm6zts22rl4hly2

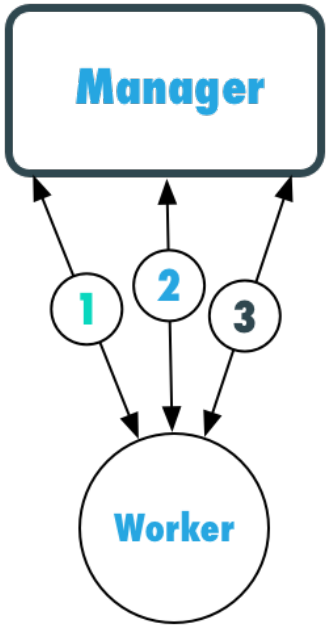
Prefix to allow VCS searches for leaked Tokens

Cryptographic Hash of the CA Root Certificate for bootstrap

Randomly generated Secret

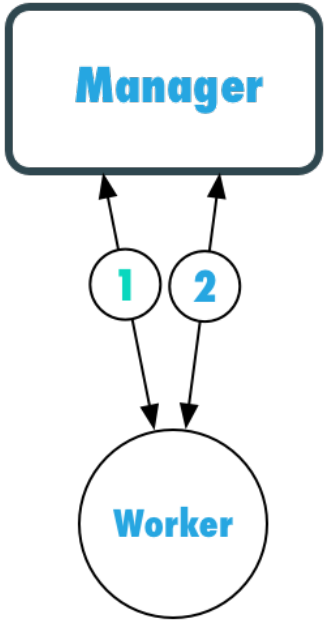
Bootstrap

1. Retrieve and validate Root CA Public key material.
2. Submit new CSR along with secret token.
3. Retrieve the signed certificate.



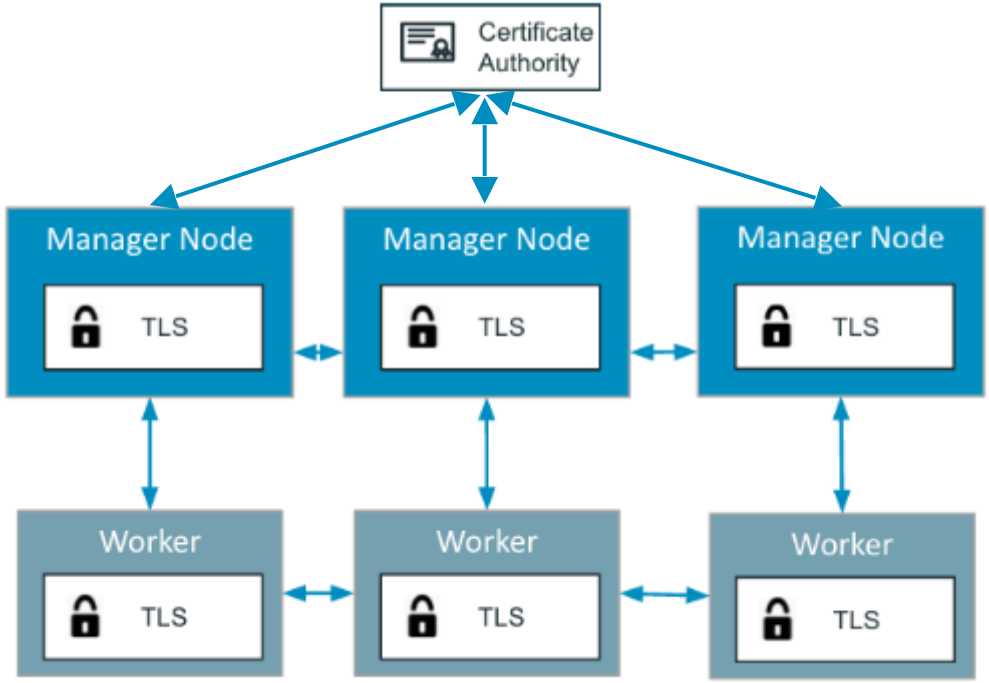
Automatic Certificate Rotation

1. Submit new CSR using old key-pair.
2. Retrieve the new signed certificate.

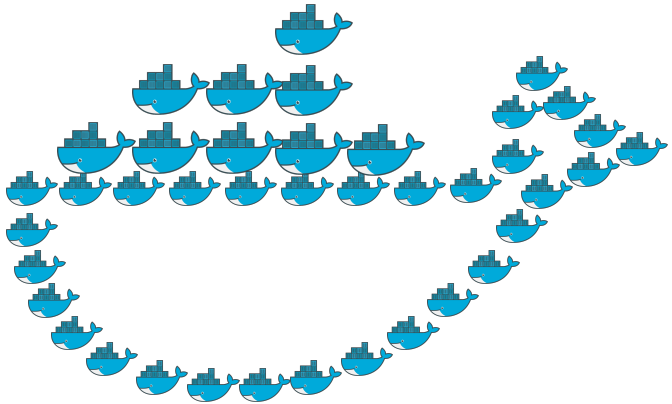


Support for External CAs

- Managers support BYO CA.
- Forwards CSRs to external CA.

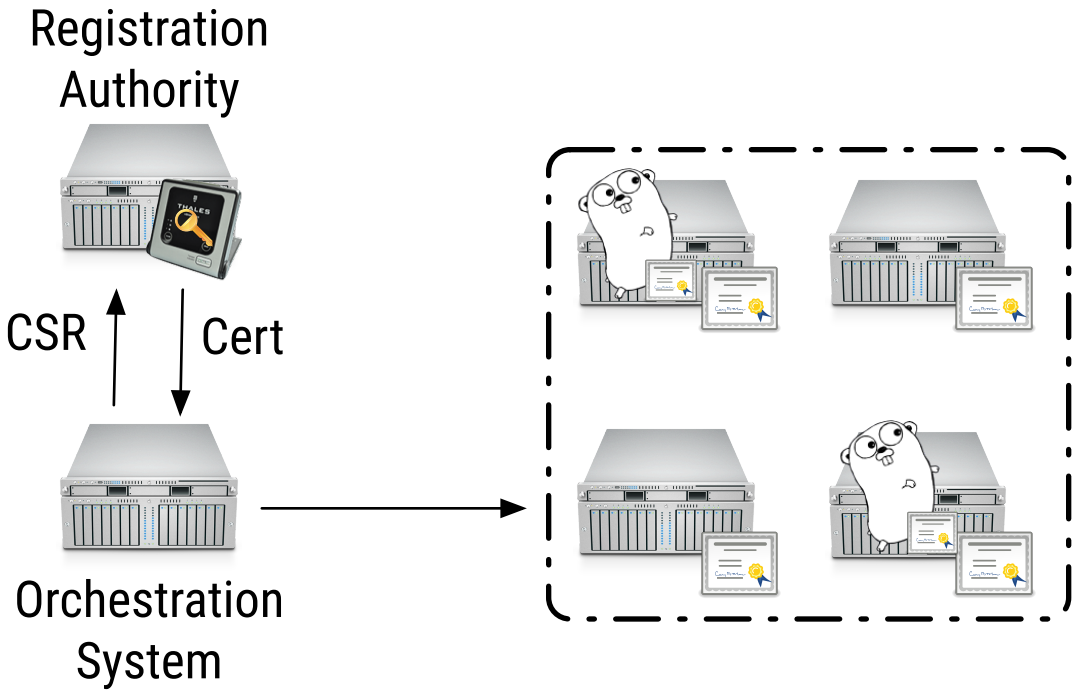


Demo

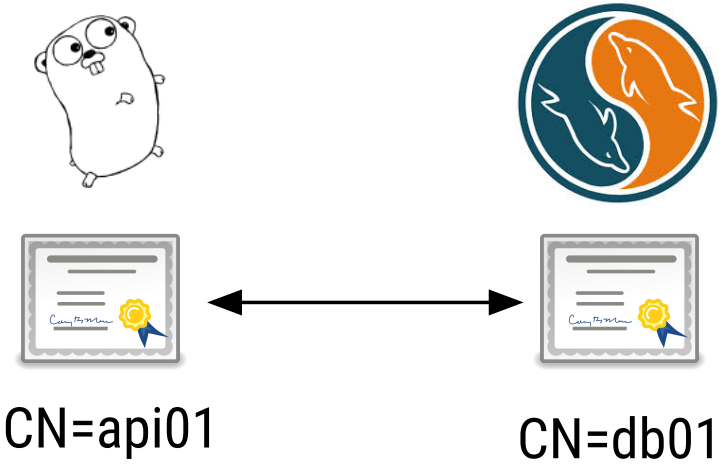


One app, one ID

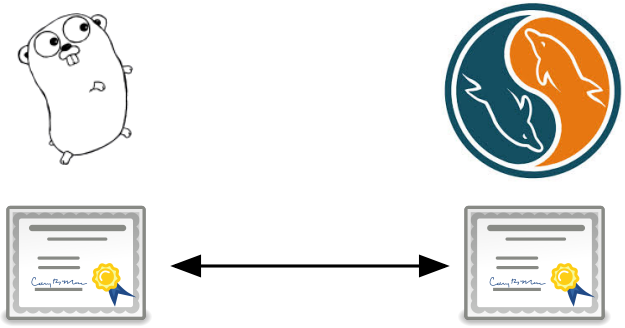
Your Orchestration System is your Registration Authority


Registration
Authority
CSR
Cert
Orchestration
System

MTLS for service authentication


CN=api01
CN=db01

MTLS for service authorization



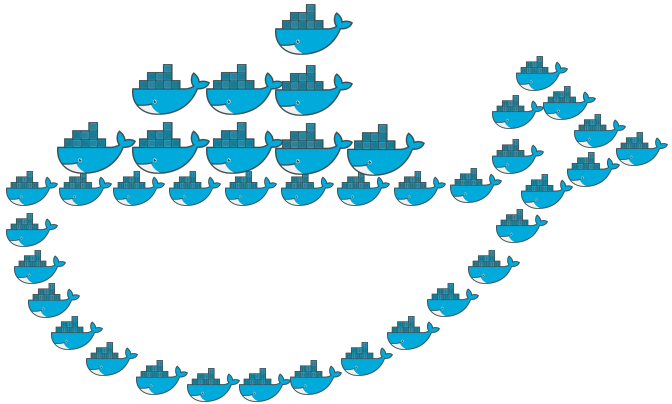
CN=api01
OU=web-api
O=production

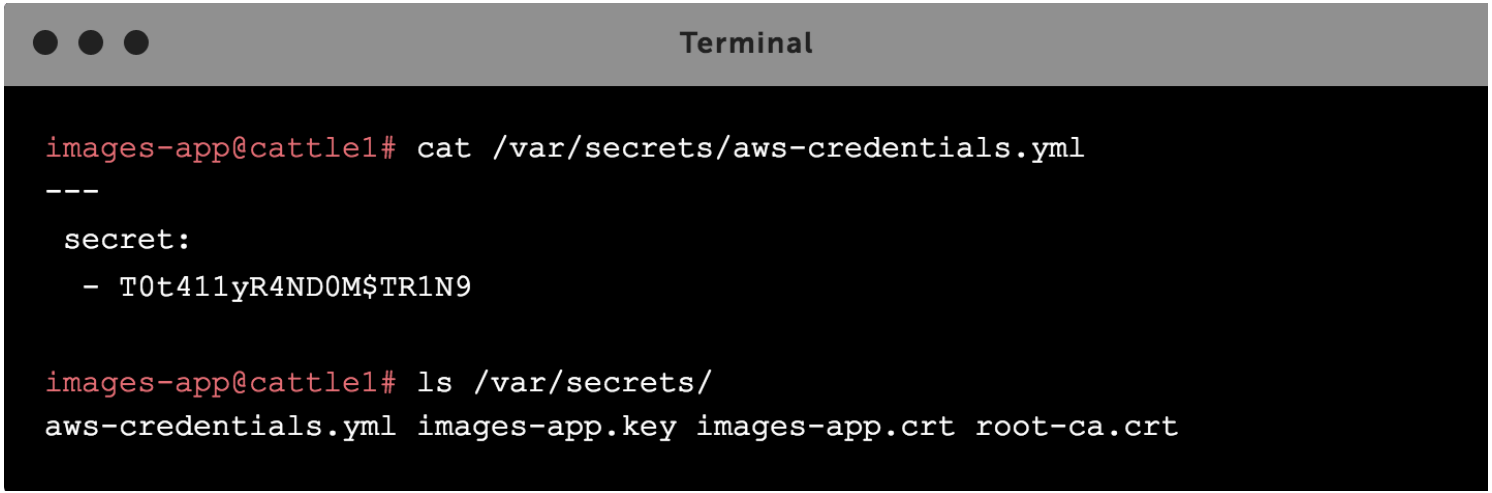
CN=db01
OU=credit-card-db
O=production

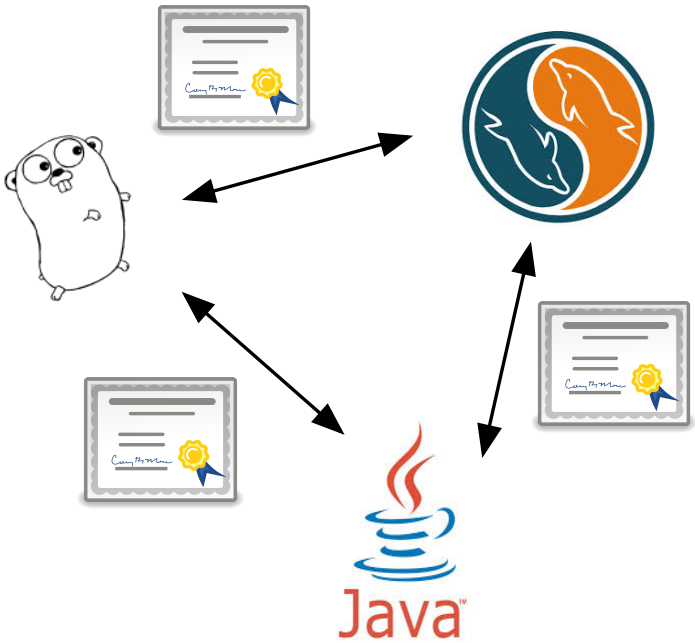
```
[ { "permission":
        { "method": "GET", "resource": "/user" },
    "allow": ["web", "fulfillment", "payments"] },

  { "permission":
        { "method": "POST", "resource": "/user" },
    "allow": ["signup", "web"] },

  { "permission":
        { "method": "DELETE", "resource": "/user/.*" },
    "allow": ["web"]
}]
```

Sane access to raw secrets

Demo




Terminal
images-app@cattle1# cat /var/secrets/aws-credentials.yml
---
secret:
- T0t411yR4ND0M$TR1N9
images-app@cattle1# ls /var/secrets/
aws-credentials.yml images-app.key images-app.crt root-ca.crt


Java

Thank you

@diogomonica