Building Zero Trust based Authentication in Healthcare with SPIRE

Speakers

Bobby Samuel

Staff VP, AI Engineering

Anthem Inc

Frederick F. Kautz IV

Head Edge Infrastructure

Doc.ai

Emiliano Berenbaum

Chief Technologist

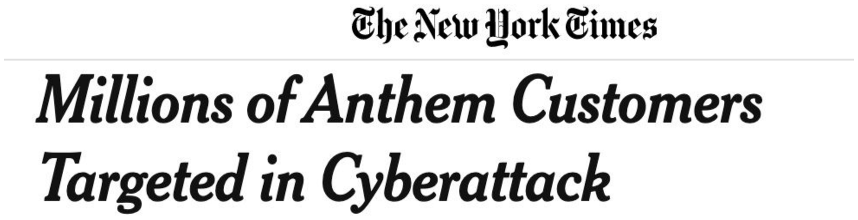
HPE

Madhukesh Wali

Software Engineer

HPE

New Operating & Threat Models Redefining Healthcare



The New York Times

Millions of Anthem Customers Targeted in Cyberattack

2015 data breach

Rising healthcare costs fueling the drive for innovation

As a result…

- **We are adopting a Zero Trust based security model for our next-gen cloud native architectures**
- **SPIFFE and SPIRE Projects will help us build a foundation based on strong identity and authentication.**

Anthem

HealthOS Overview

Anthem AI is delivering a suite of products to transform healthcare. **At the core of each product is a DocBot tool that can be delivered through multiple channels** of front-end delivery platforms.

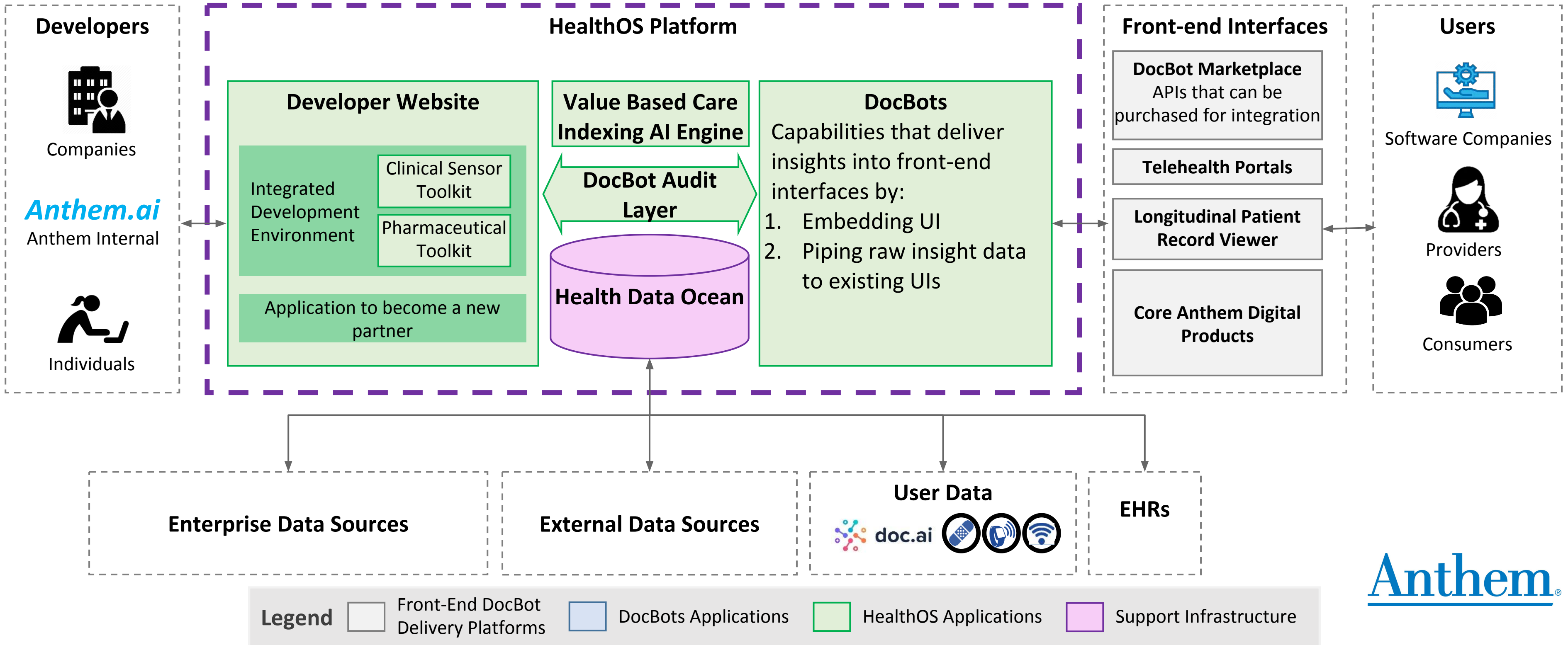
At the heart of HealthOS, **powering these DocBots, is a vast ocean of secure, developer-accessible health data**. Initial internal Anthem use cases will validate and shape the platform.

Once validated, **HealthOS will be made accessible to third party developers** who create applications based on a provider order board, **unlocking a supply and demand driven application marketplace** available across healthcare ecosystem stakeholders

Anthem®

HealthOS Approach

Starting with internal Anthem use cases, the team is exploring the potential to enable an environment for third parties to develop 'DocBot' capabilities to deliver into 'Front-end Interfaces' leveraging an ocean of de-identified health data



Anthem

The Emergence of Zero Trust Architecture

We’re defending our infrastructure with 11th century techniques!

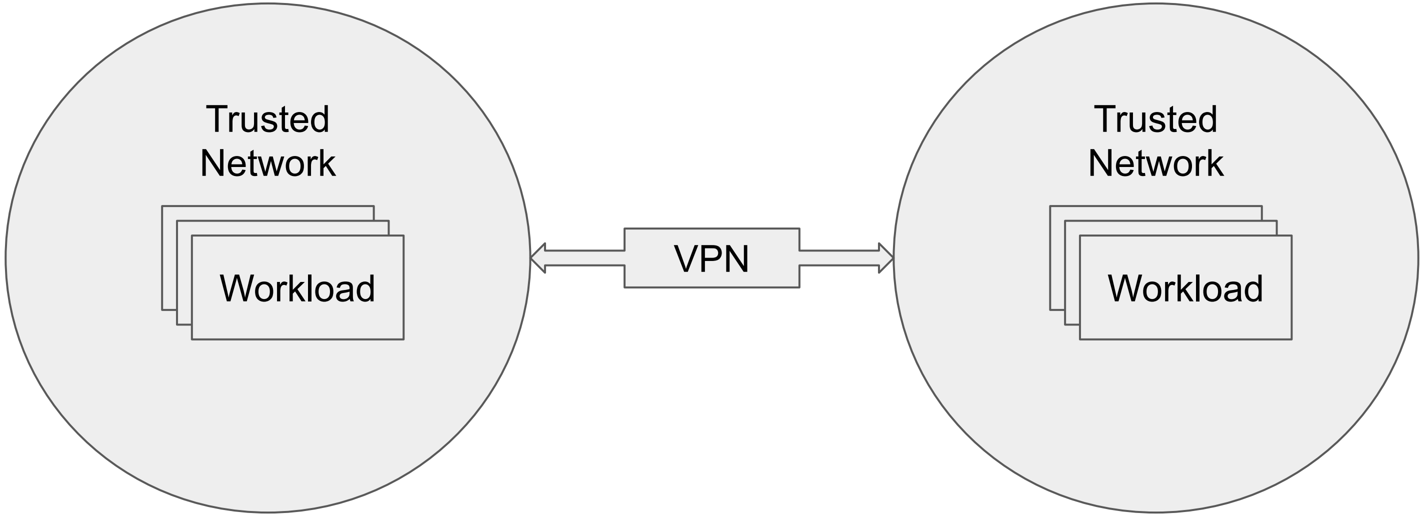
H.C. Steensen https://commons.wikimedia.org/wiki/File:KronborgCastle_HCS.jpg

We're defending our infrastructure with 11th century techniques!

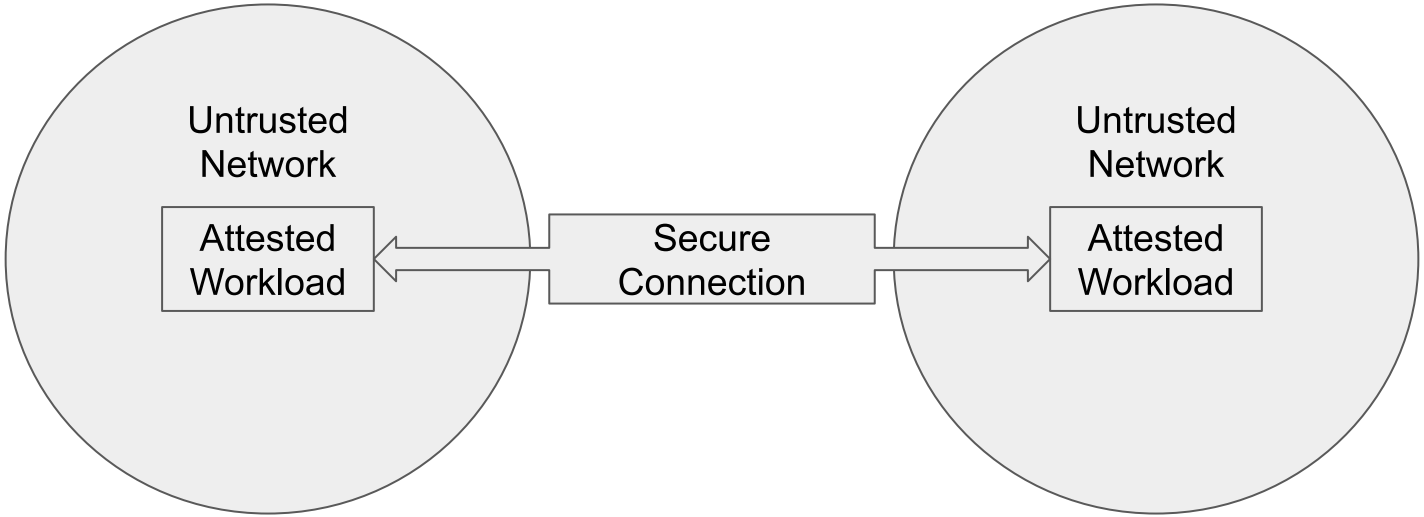
What if the attack starts here?

H.C. Steensen https://commons.wikimedia.org/wiki/File:KronborgCastle_HCS.jpg

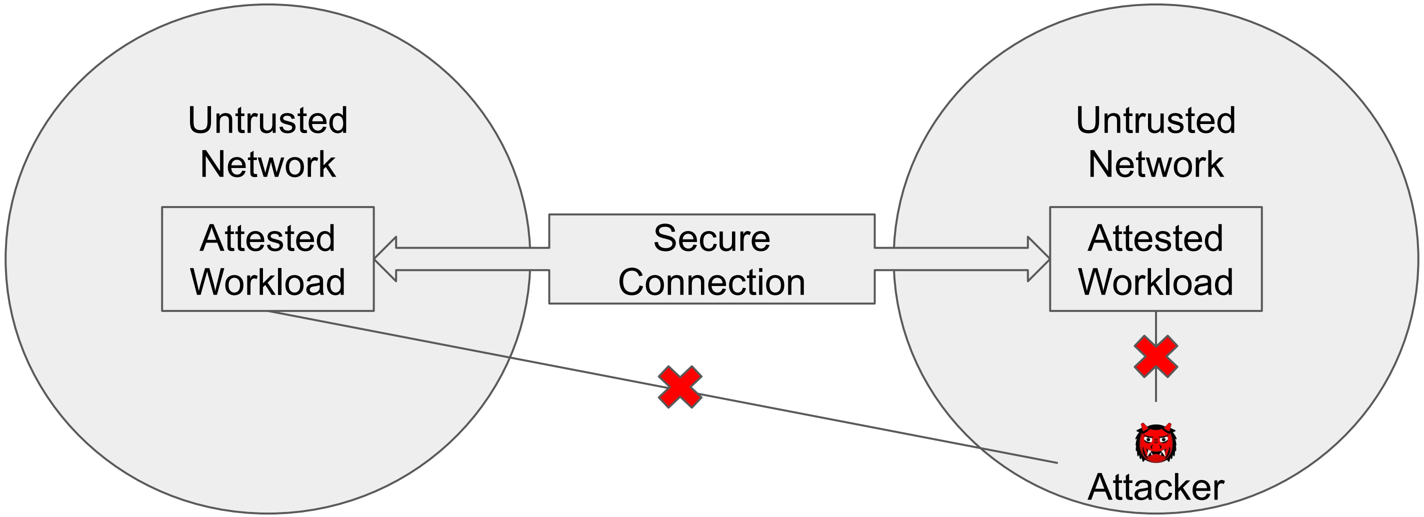
Perimeter Defense



Zero Trust Environment



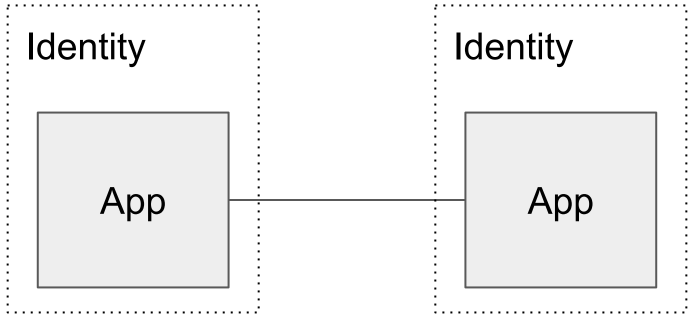
Zero Trust Environment



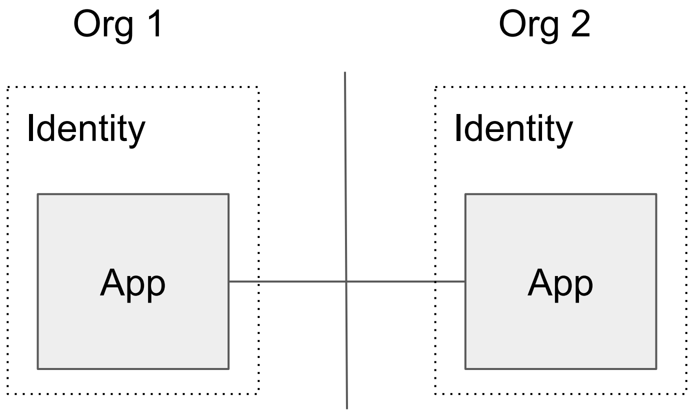
How do we achieve this?

- Establish trust domains
- Attest workloads
- Establish policy
- Establish trust between organizations

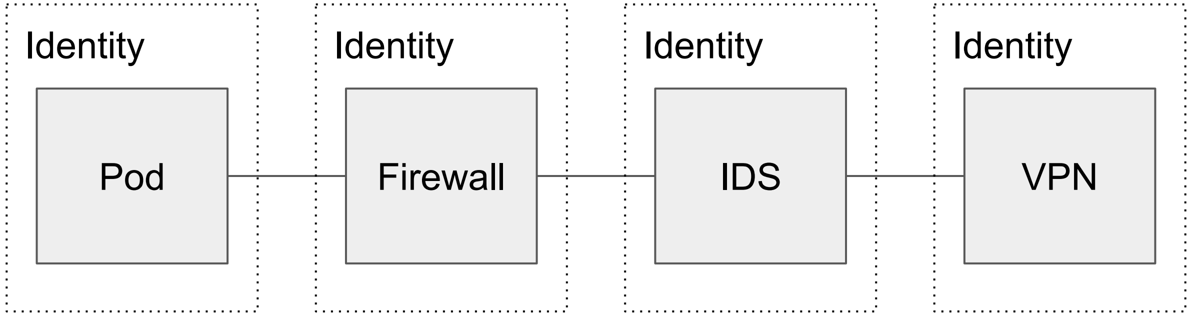
Application Pattern



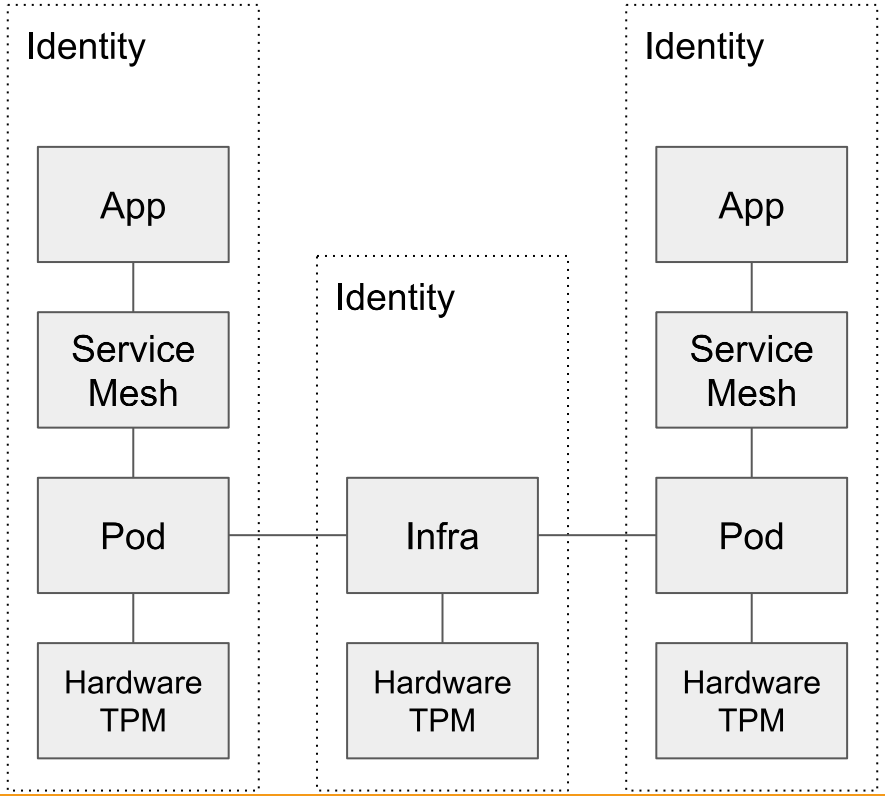
Application Pattern



Infrastructure Pattern



Cross-cutting Identity



No AuthN No Zero Trust

Fine grained identity and authentication foundational to operating a zero trust based security model

Zero Trust Cloud Native Technologies @ Doc.ai

- SPIFFE and SPIRE provide workload identity and authentication as the foundational layer

- OPA (Open Policy Agent) provides authorization

- NSM (Network Service Mesh) provides cross-cluster connectivity and network policy

Service Authentication for Zero Trust Security Model With SPIRE

SPIFFE & SPIRE

The cloud-native service identity plane for zero trust security model

spiffe

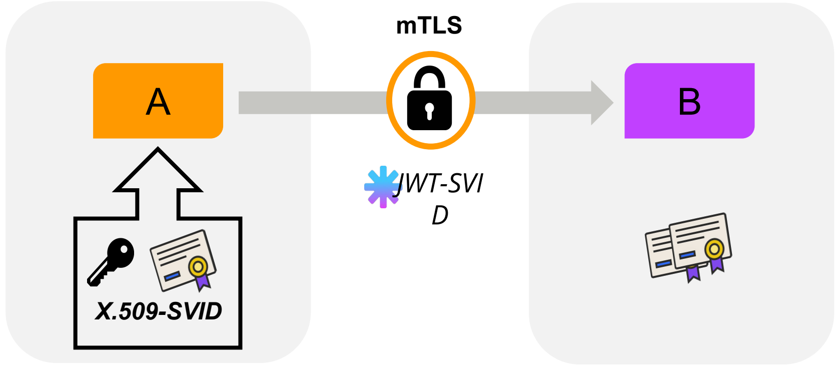
Launched in 2017 and joined the CNCF in 2018.

Integrated into various open-source projects.

Extensive contributions by HPE and other Fortune 2000 enterprises.

SPIRE

Standard and toolchain for workload identity and authentication

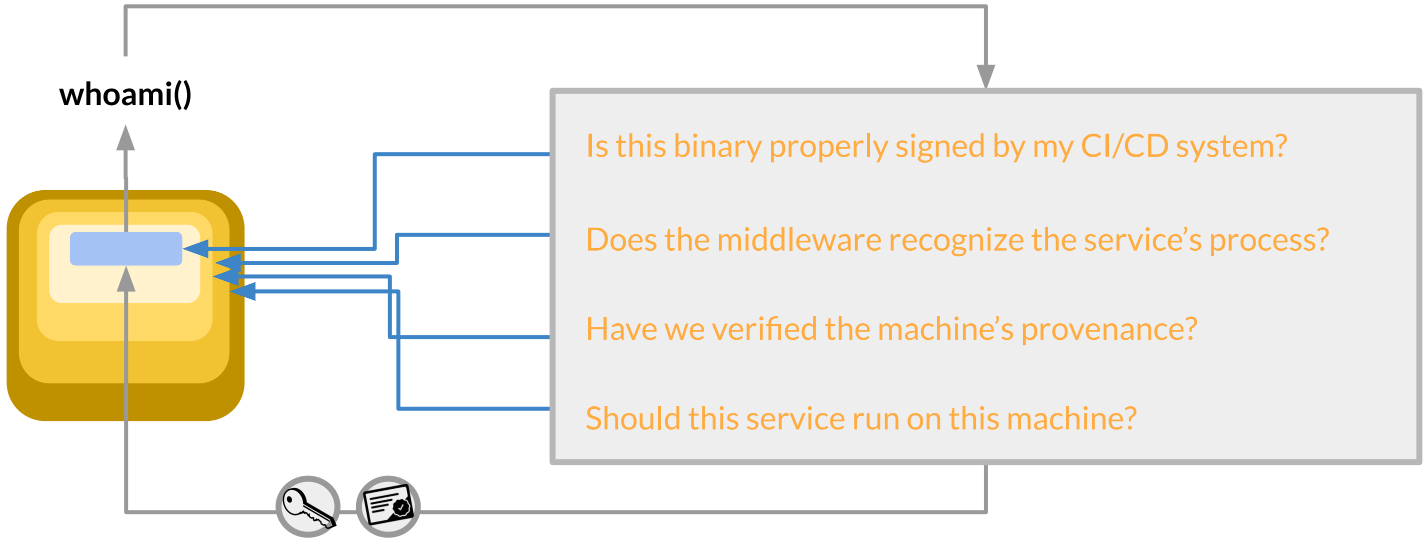


- MFA engine to define, attest, issue, and revoke deliver dynamic service identities.
- Reduces reliance on platform-specific network-based security controls.
- Eliminates use of static service credentials.

SPIFFE

- SPIFFE ID
- SPIFFE Verified Identity Document (SVID)
- Workload API
- Federation API

SPIRE *verifies* SPIFFE passports issued to software.


whoami()
Is this binary properly signed by my CI/CD system?
Does the middleware recognize the service's process?
Have we verified the machine's provenance?
Should this service run on this machine?

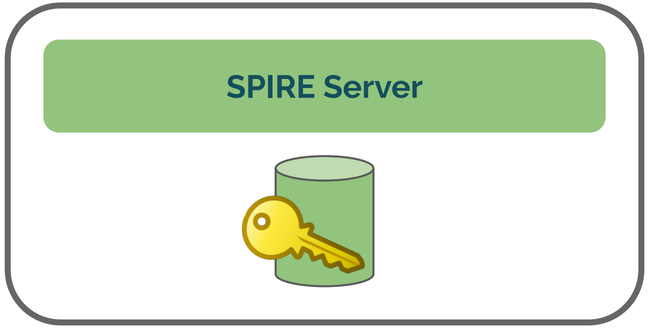
spiffe://acme.com/billing/payments

selector: aws:sg:sg-edcd9784
selector: k8s:ns:payments
selector: k8s:sa:pay-svc
selector: docker:image-id:442ca9

The SPIRE Server (eg.acme.com) maintains a list of:

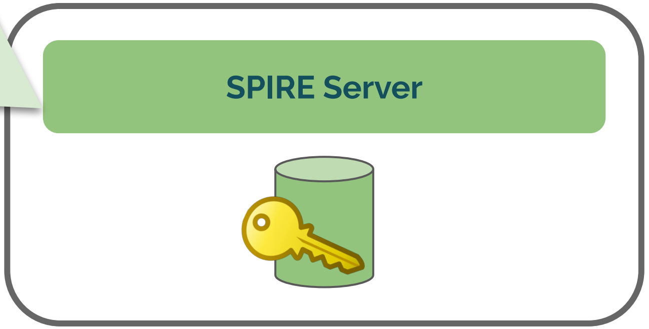
1) *local service identities (eg. /billing/payments).*

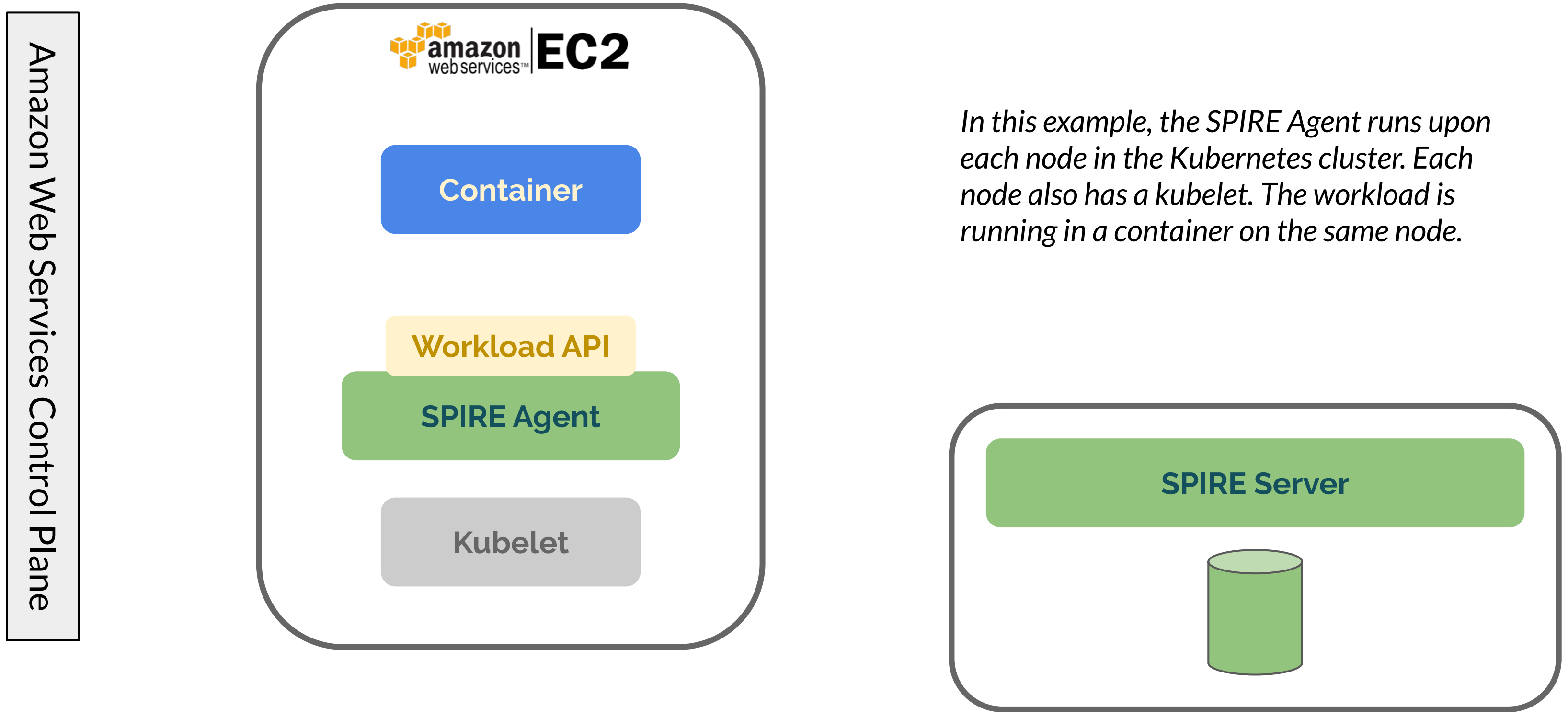
2) *Conditions that must be matched by a service to be entitled to an identity.*



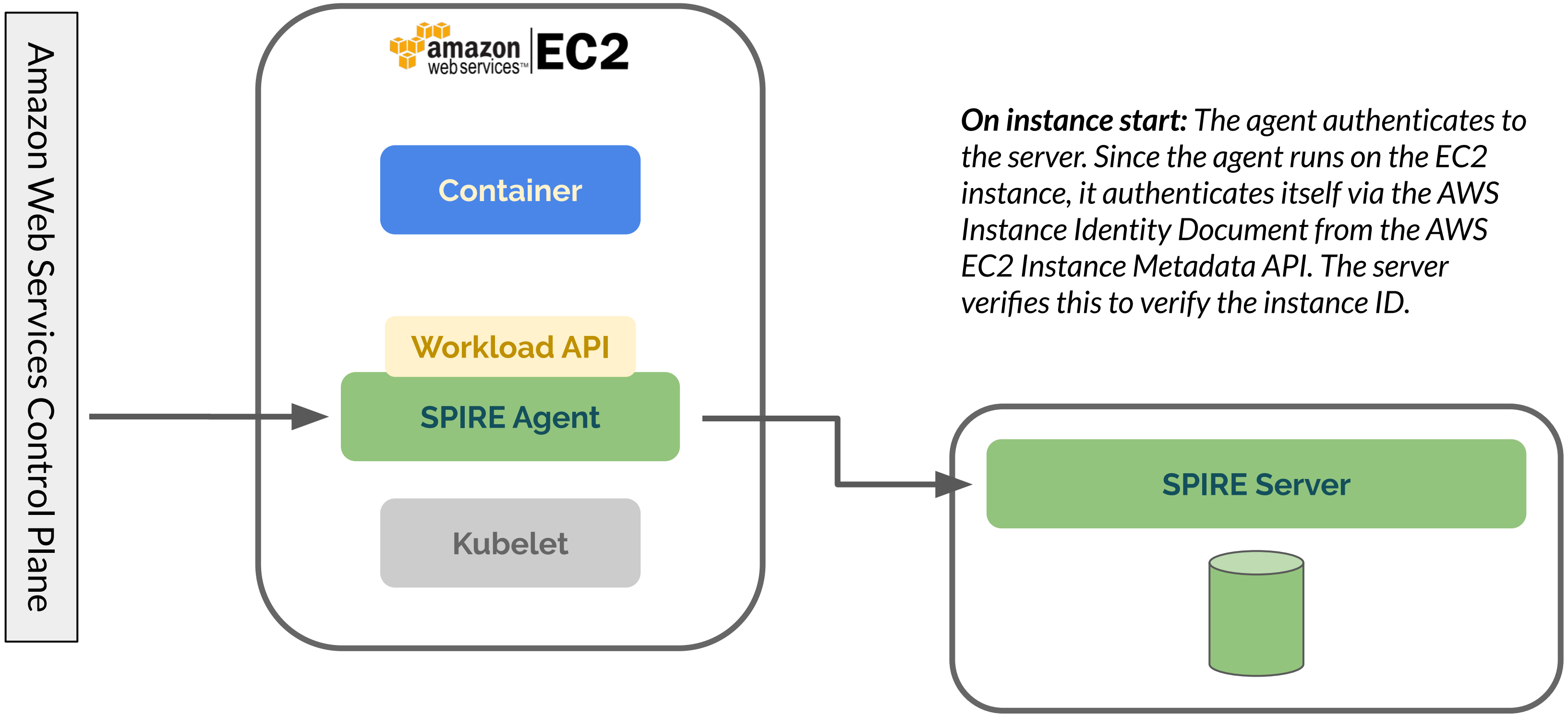
To be issued the ID ***spiffe://acme.com/billing/payments****, a service must be:*

1) *Running on an EC2 instance in the security group 'sg-edcd9784'.*
2) *Running in a Kubernetes pod in the namespace 'payments'.*
3) *Running in a Kubernetes pod associated with the service account 'pay-svc'.*
4) *Running in a Docker container started from image with the ID '442ca9'.*

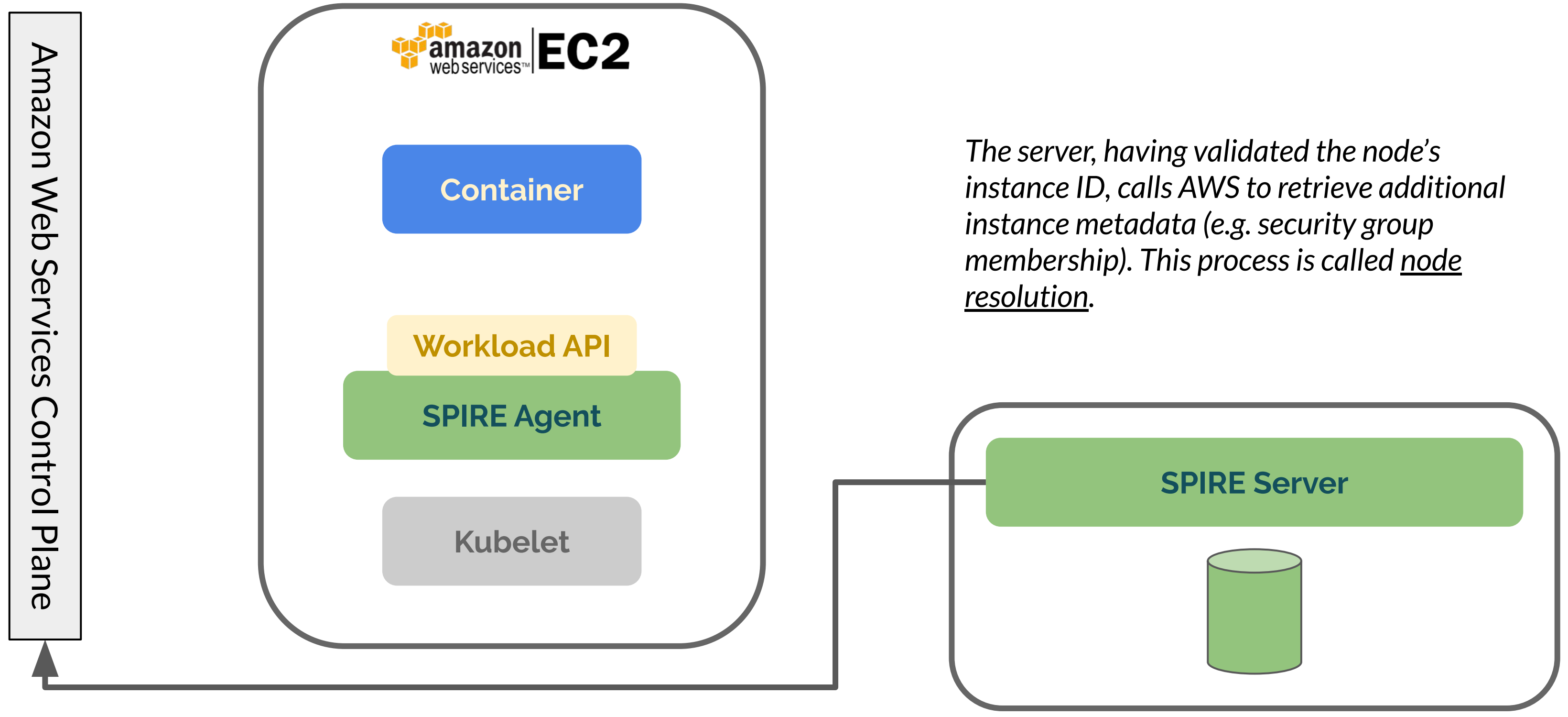




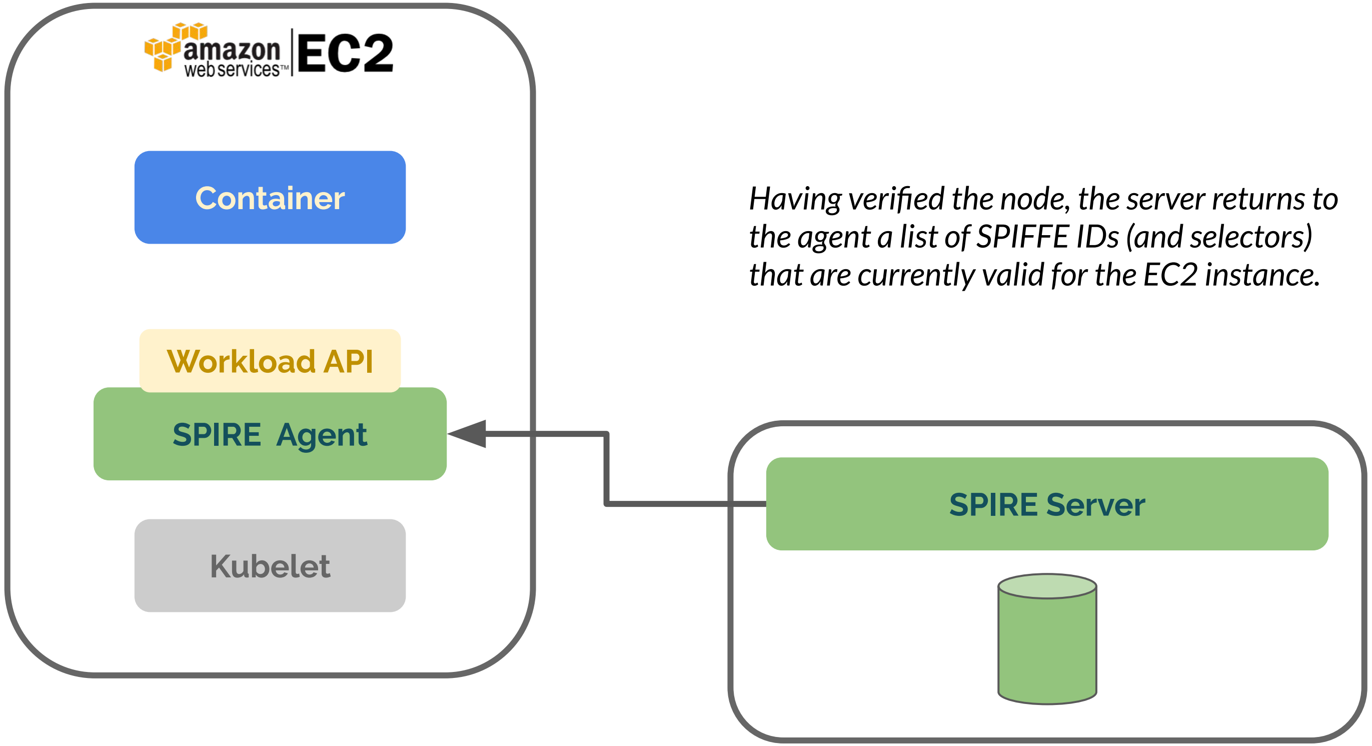
In this example, the SPIRE Agent runs upon each node in the Kubernetes cluster. Each node also has a kubelet. The workload is running in a container on the same node.



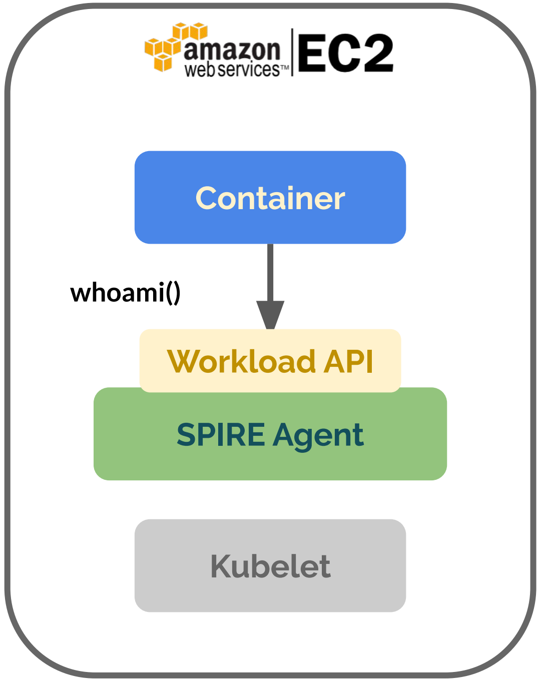
On instance start: *The agent authenticates to the server. Since the agent runs on the EC2 instance, it authenticates itself via the AWS Instance Identity Document from the AWS EC2 Instance Metadata API. The server verifies this to verify the instance ID.*



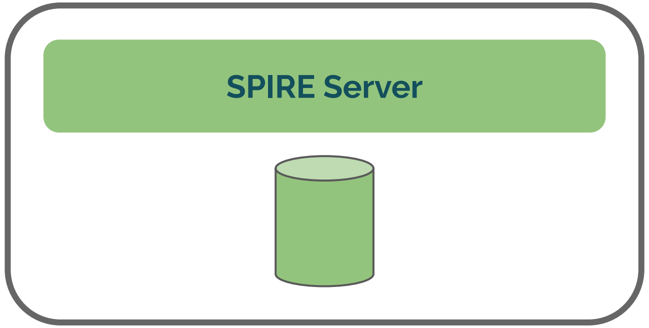
The server, having validated the node's instance ID, calls AWS to retrieve additional instance metadata (e.g. security group membership). This process is called node resolution.

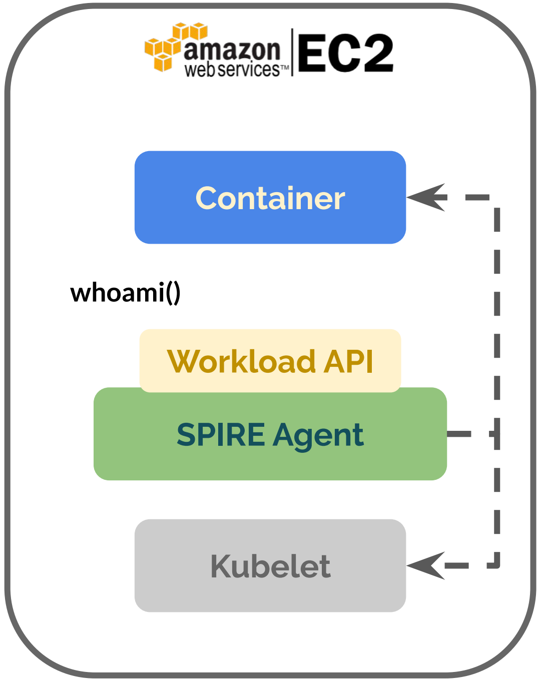


Having verified the node, the server returns to the agent a list of SPIFFE IDs (and selectors) that are currently valid for the EC2 instance.



***At service start time:** The service (running in a container that is part of a Kubernetes pod) requests identity from the agent (via the Workload API that's exposed as a Unix Domain Socket).*

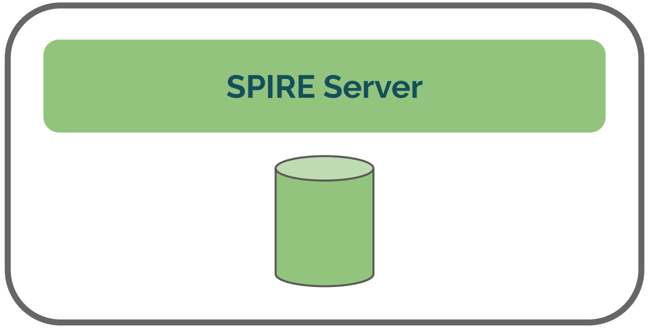


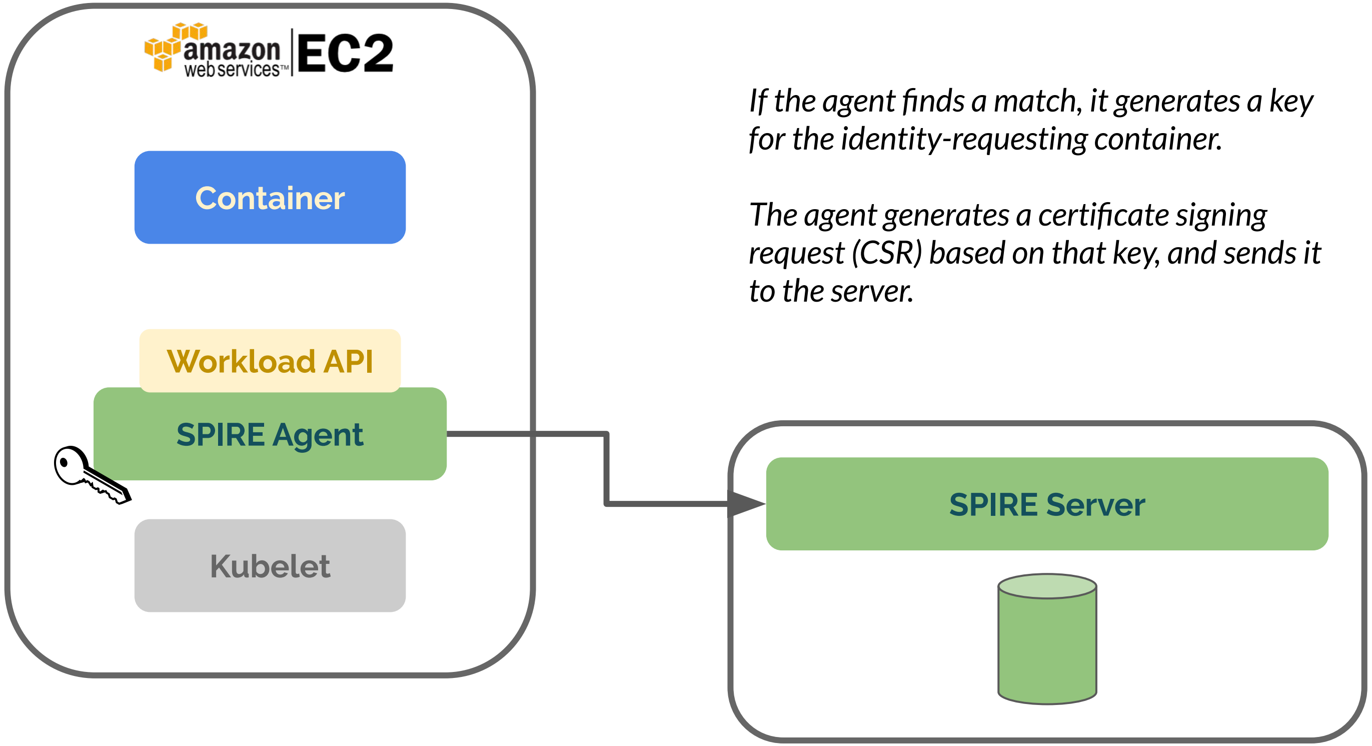


The agent retrieves metadata from the kernel about the calling process, and against selectors in policies retrieved from the server.

The agent also interrogates the local kubelet for Kubernetes-specific metadata about the process.

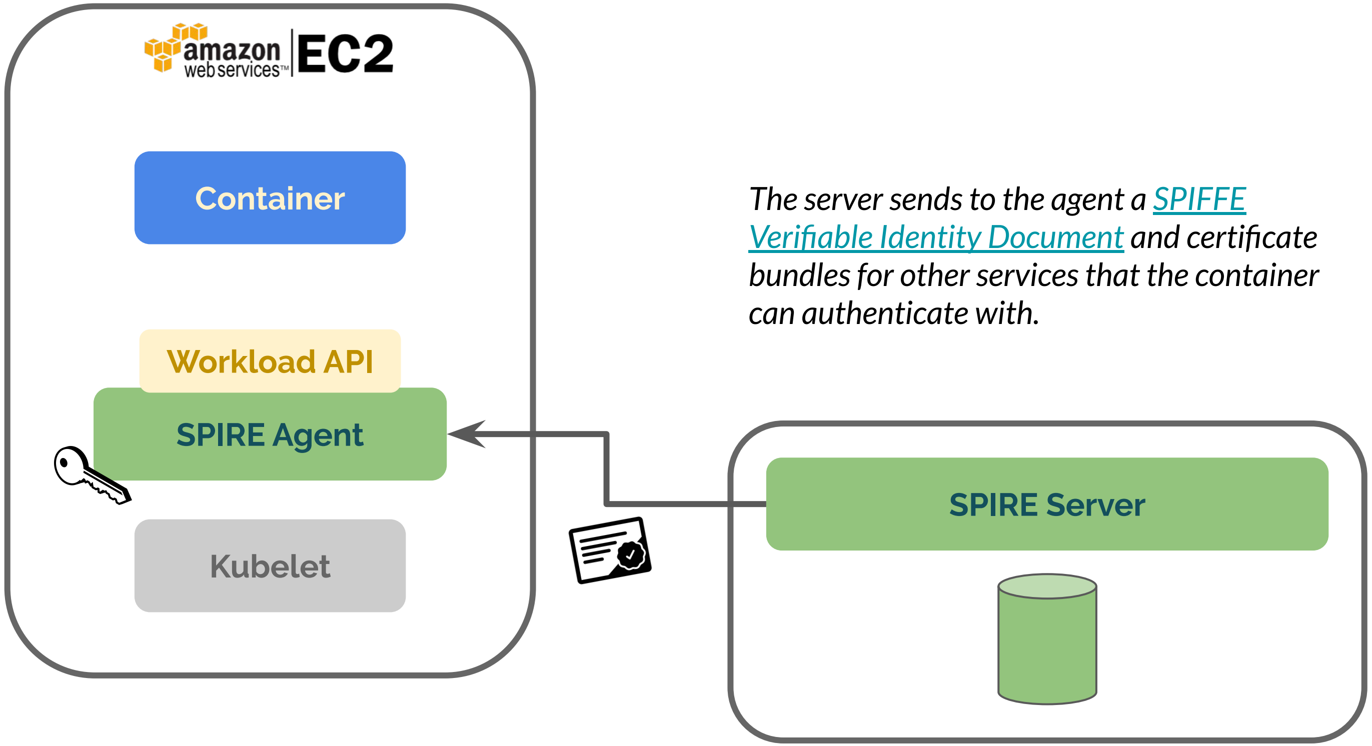
The agent then correlates this against the policies it previously retrieved from the server.



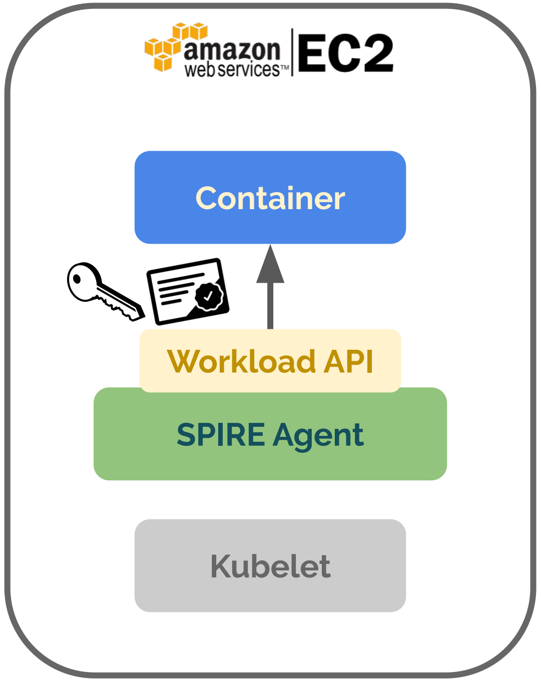


If the agent finds a match, it generates a key for the identity-requesting container.

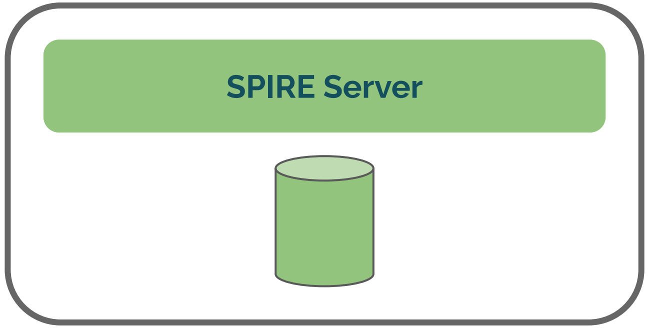
The agent generates a certificate signing request (CSR) based on that key, and sends it to the server.



The server sends to the agent a SPIFFE Verifiable Identity Document and certificate bundles for other services that the container can authenticate with.



The agent returns the SVID and certificate bundles to the container.

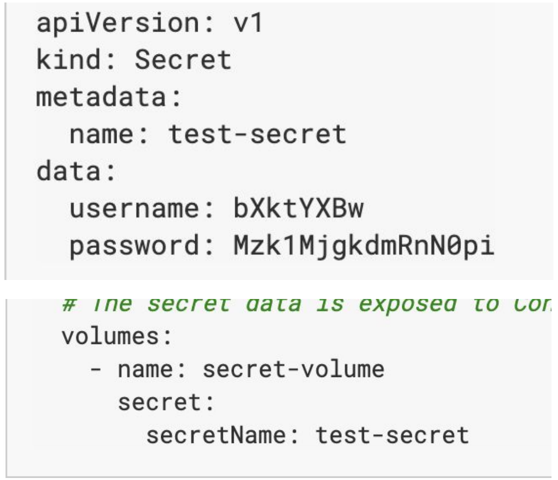


Demo

Demo Scenarios

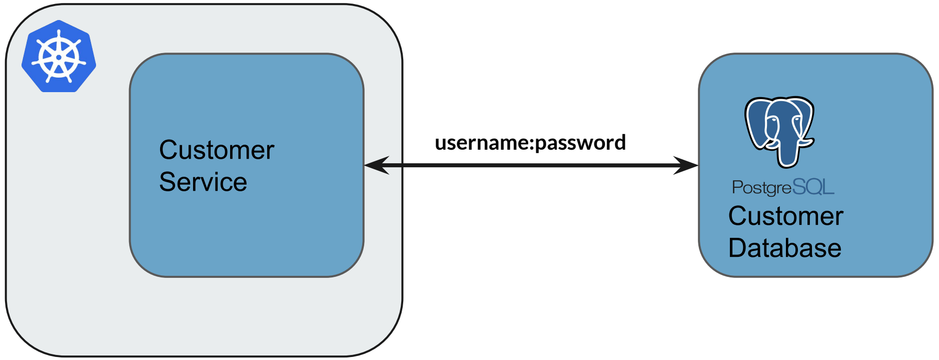
- **X.509 SVID Authentication to Postgres**
- Secretless authentication to AWS RDS

Authentication to Postgres



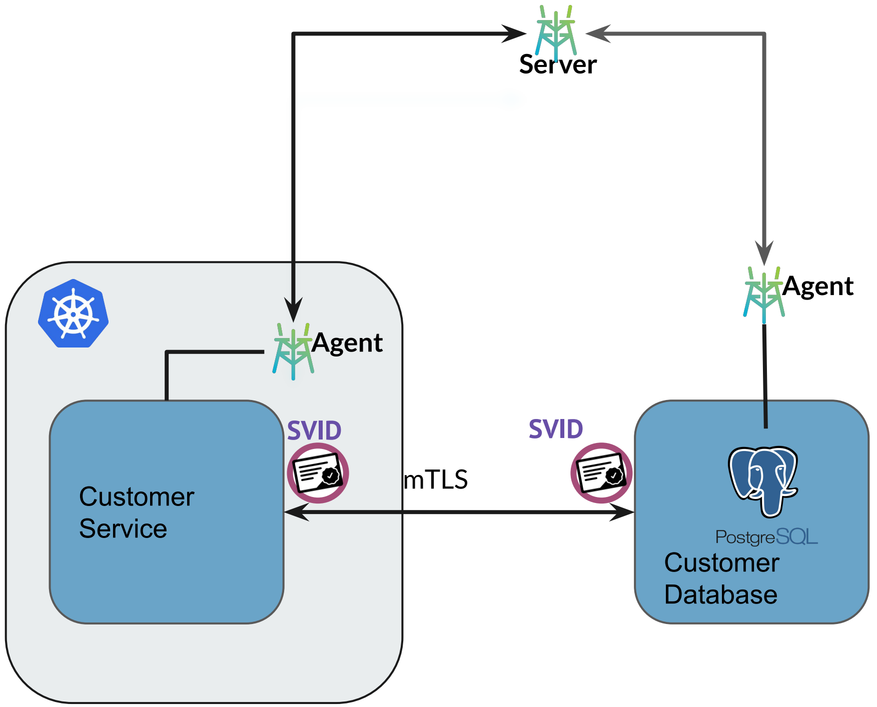
```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
data:
  username: bXktYXBw
  password: Mzk1MjgkdmRnN0pi
```

```
  # The secret data is exposed to Co
  volumes:
    - name: secret-volume
      secret:
        secretName: test-secret
```



X.509 SVID Authentication to Postgres

Postgres authentication is configured to only accept a valid x509 certificate where the certificate matches the requirement for the postgres account.

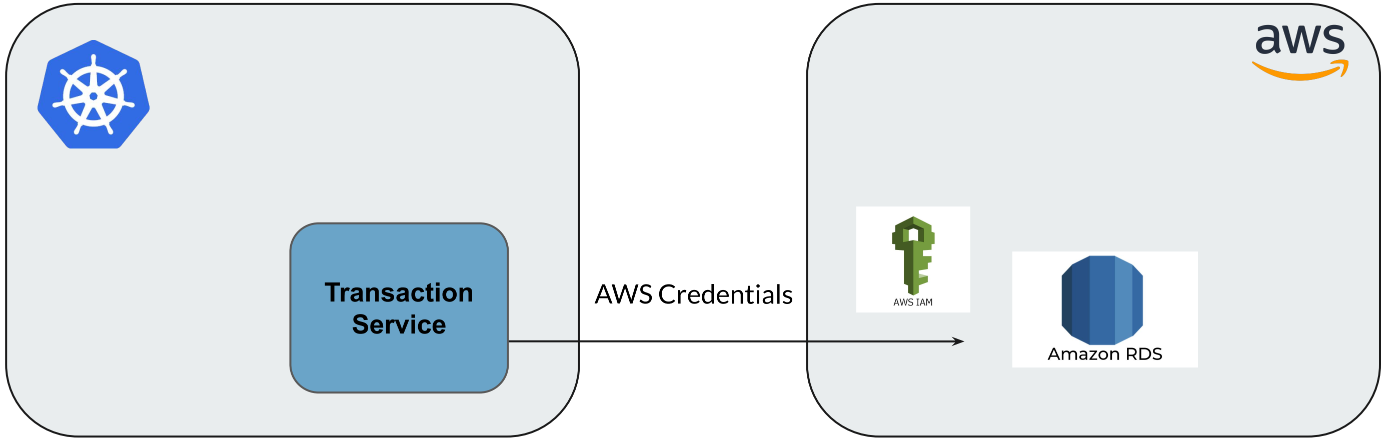


Demo Scenarios

- X.509 SVID Authentication to Postgres
- **Secretless authentication to AWS RDS**

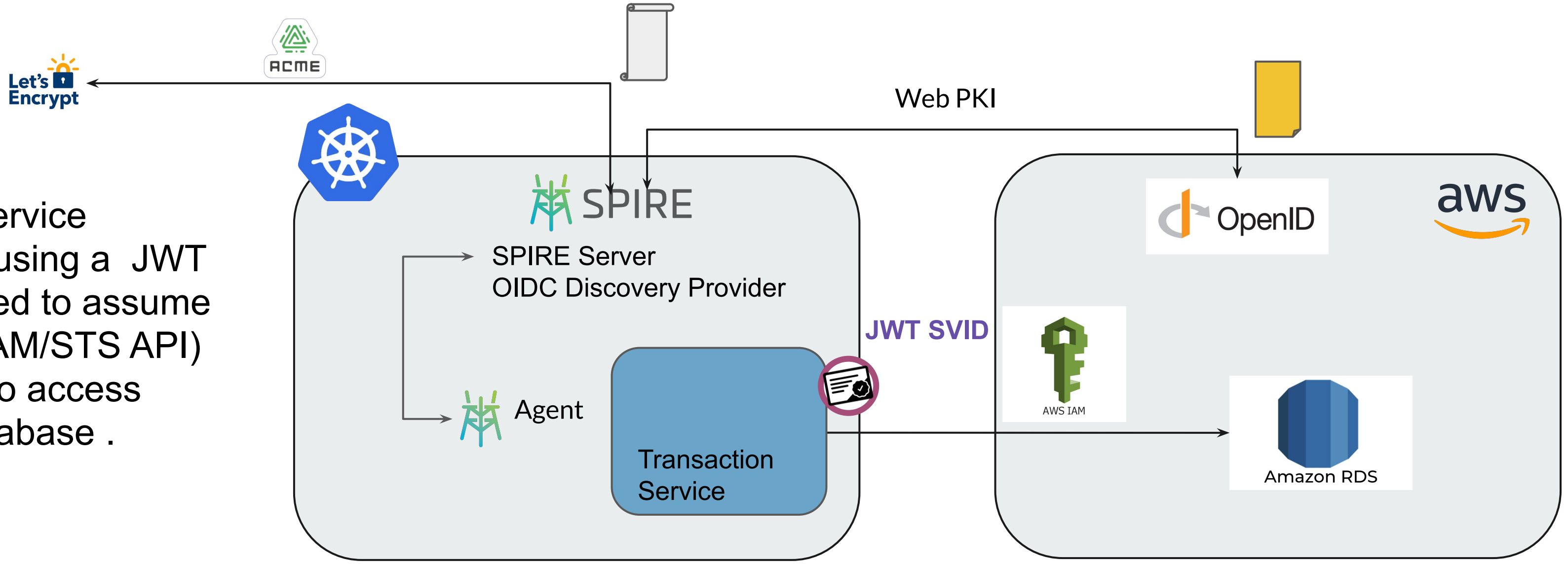
Authentication to AWS RDS

Requires storing AWS Credentials to authenticate to the RDS



Secretless authentication to AWS RDS

Transaction Service authenticates using a JWT SVID configured to assume a Role(AWS IAM/STS API) with privilege to access AWS RDS database .



Get Started

spiffe.io

spiffe.slack.com

github.com/spiffe/spire