Confidential Computing with SCONE

- Protecting Data, Code, and Secrets of Applications -

Christof Fetzer

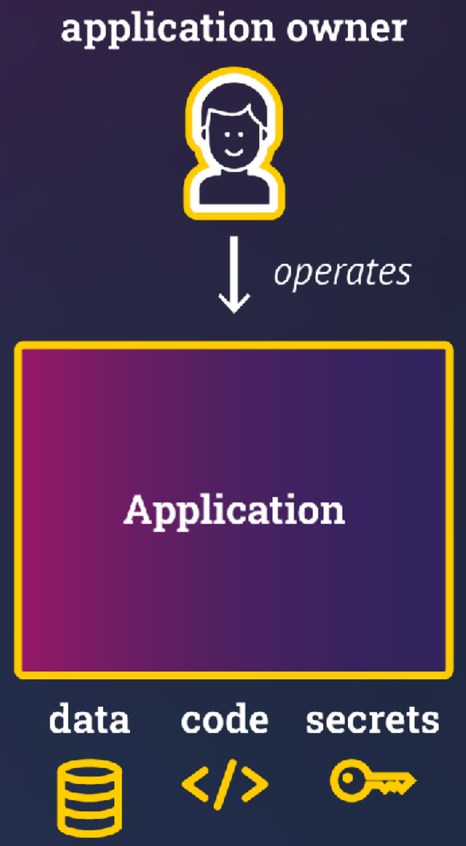
https://sconedocs.github.io

Motivation

- **Role**: **application owner**

Objectives:

- provide an application to clients
- protect **data**, **code**, and **secrets** of the application



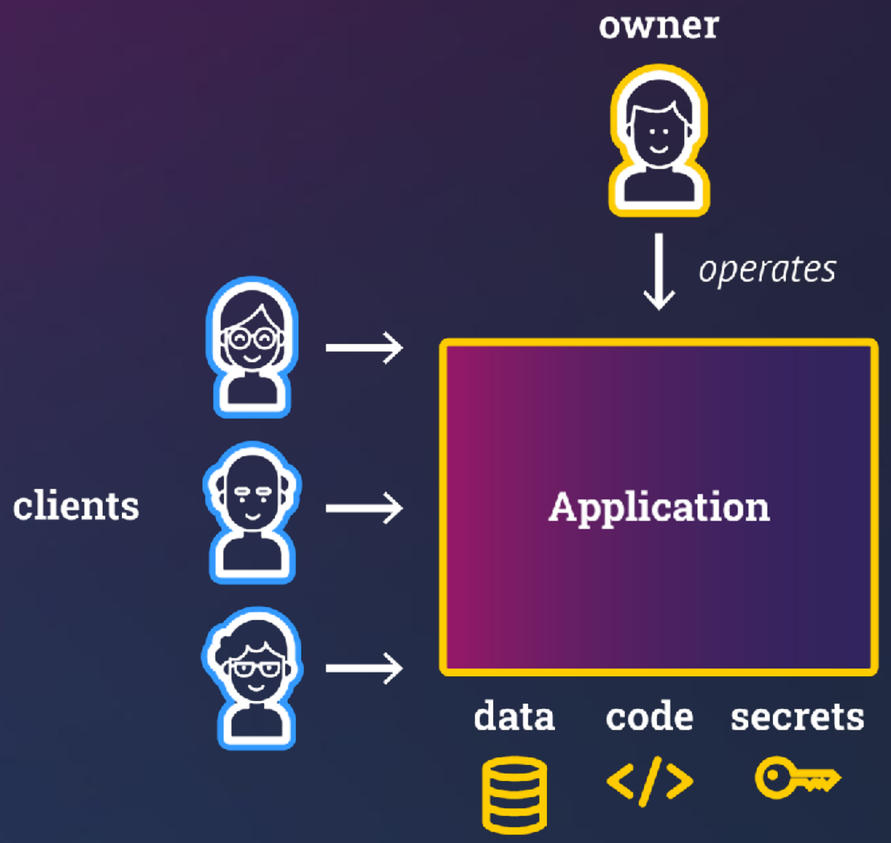
Motivation

- **Role: application owner**

Objectives:

  - provide an application to clients
  - protect **data, code, and secrets** of the application

- **Role:** clients
  - can connect to the application
  - access their data



Requirements

- example domain: eHealth -

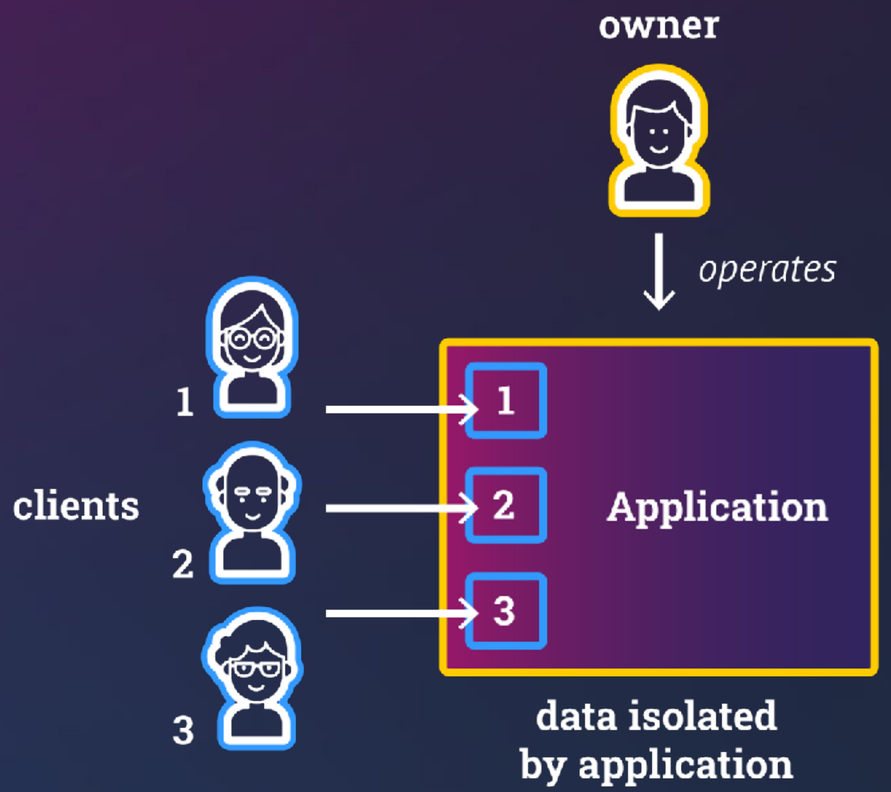
Isolation of data

- **Role: application owner**

Objectives:

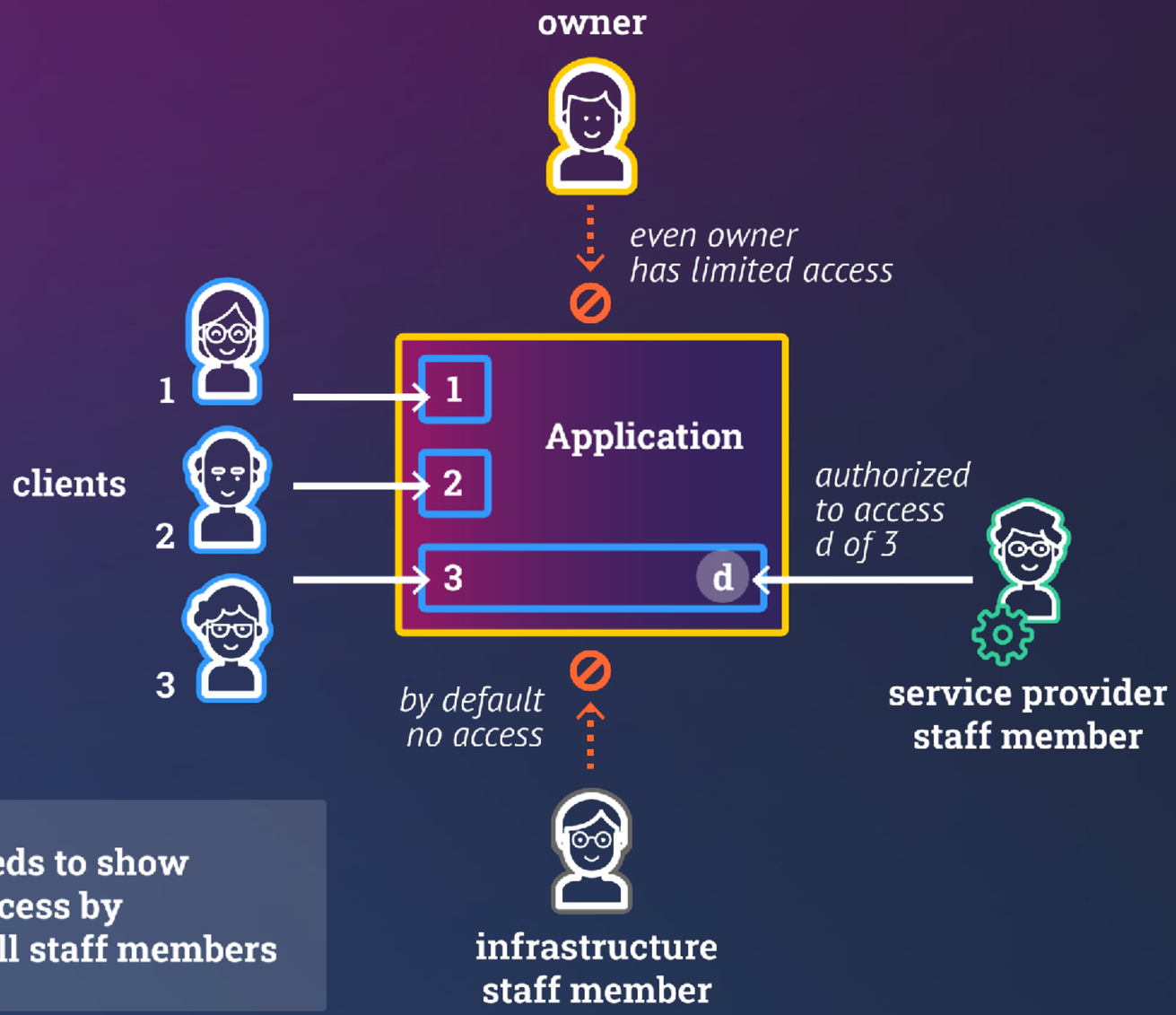
  - provide an application to clients
  - protect **data, code, and secrets** of the application

- **Role:** clients
  - can connect to the application
  - access their data
  - **application isolates data of clients**



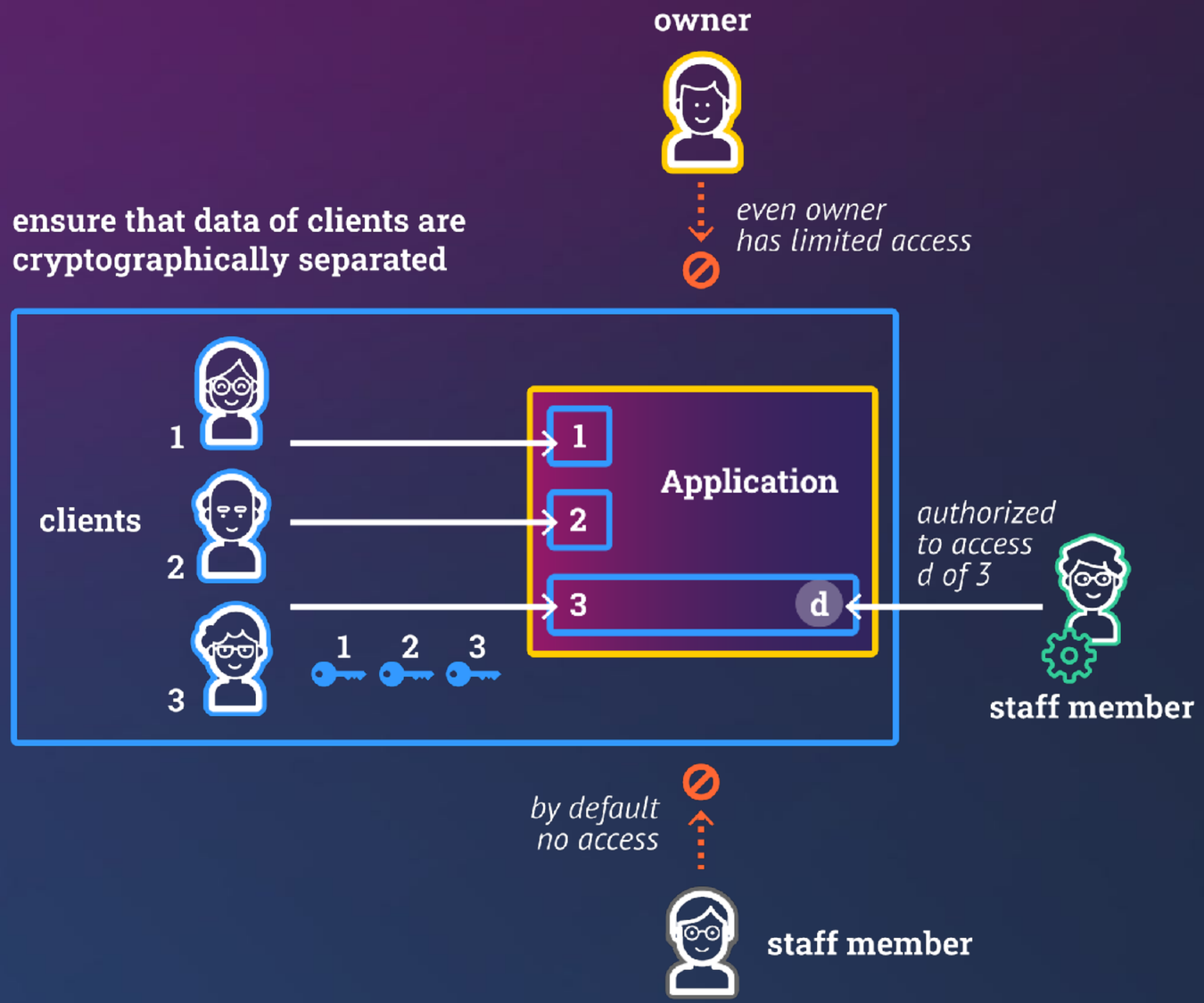
Limited Access by Owner & Staff

Example: eHealth

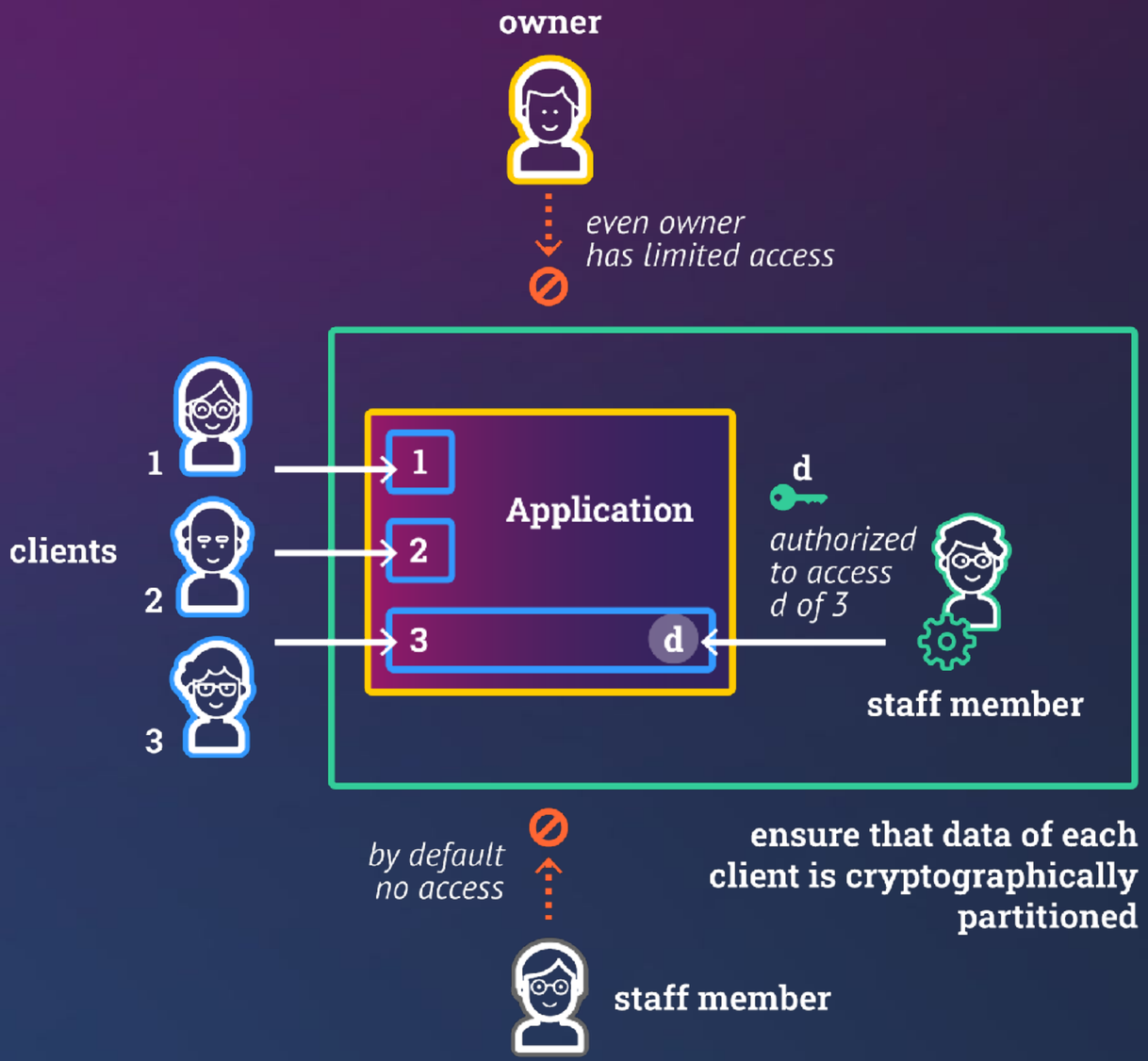


owner needs to show
limited access by
owner & all staff members

Divide and Conquer



Divide and Conquer



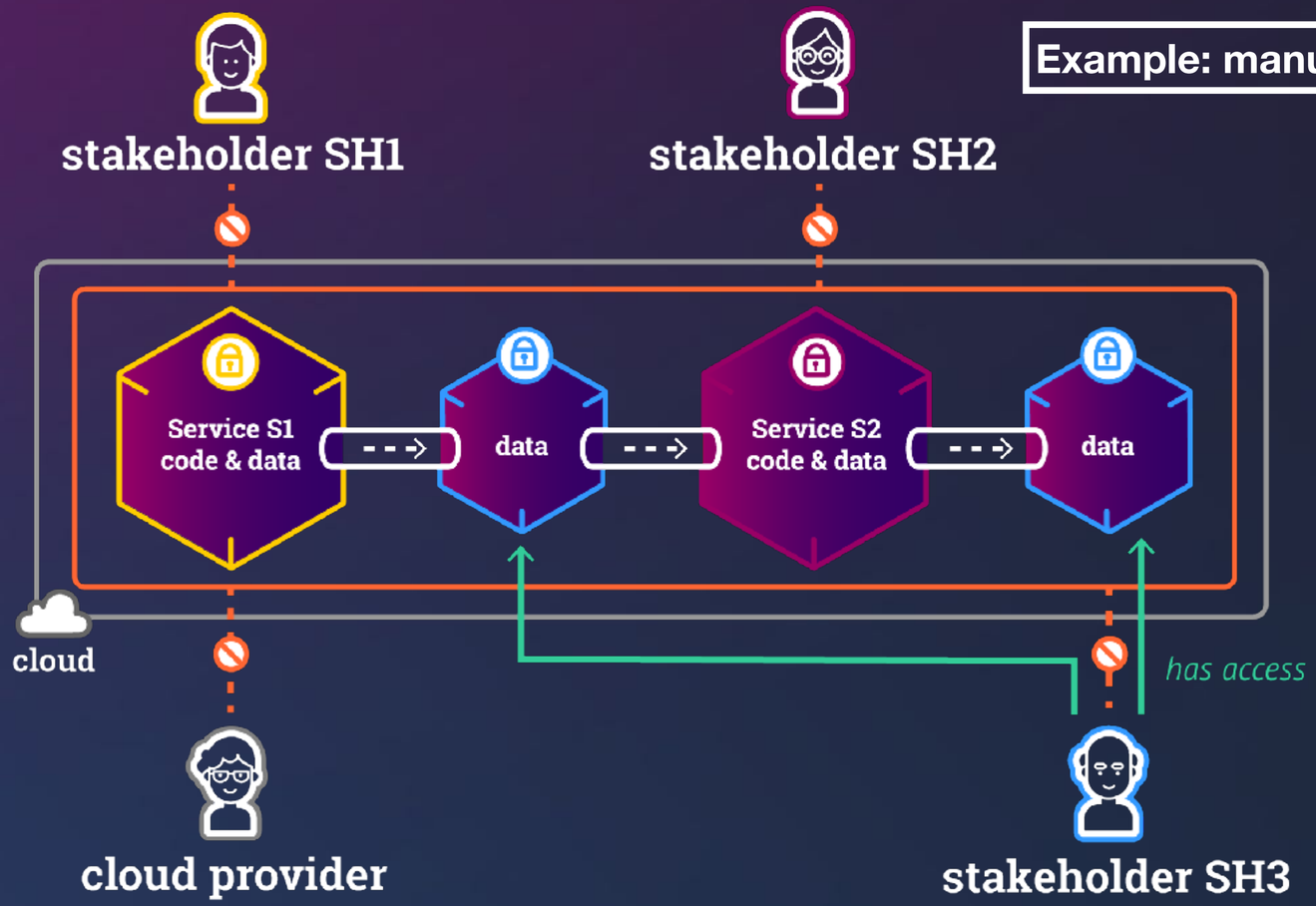
https://sconedocs.github.io

Objective eHealth: Support Machine Learning

- Protecting Data, Code and Keys -

Use Case: Multiple Stakeholder Computation!

- Confidential workflow connects **confidential services**
- Each **stakeholder** controls its IP via own policies
- Even **operator** of workflow cannot look into individual service



Example: eHealth - future

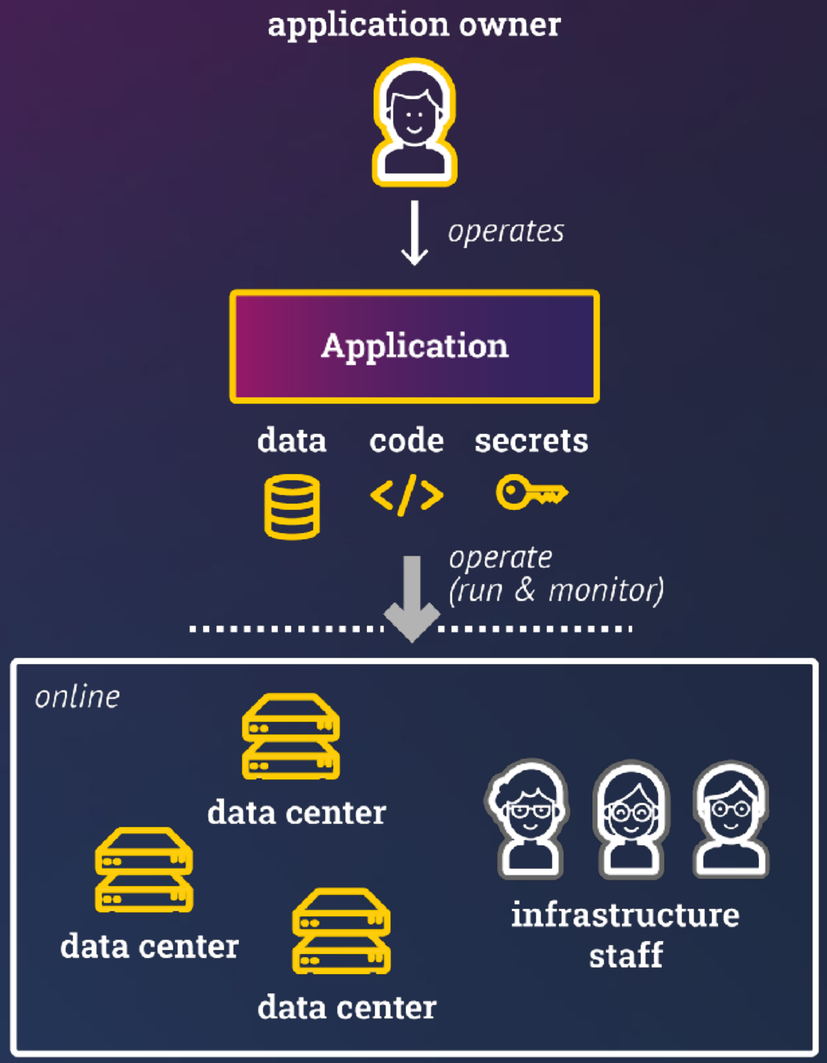
Managed **Kubernetes** cluster

Business Problem

Problem Description

Problem: application owner cannot operate the application

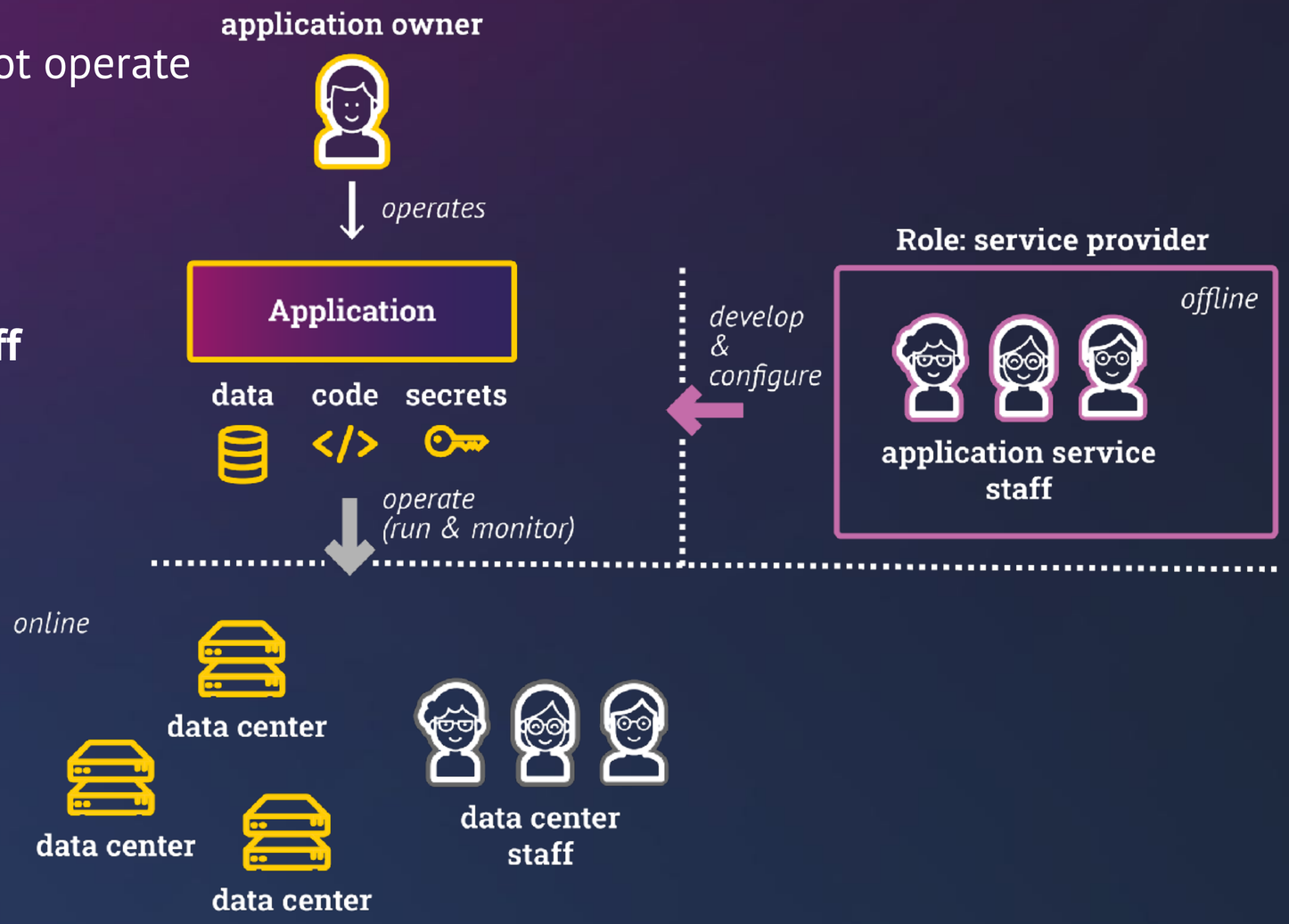
- **lack of data centers || trusted infrastructure staff**
- lack of application service staff



Problem Description

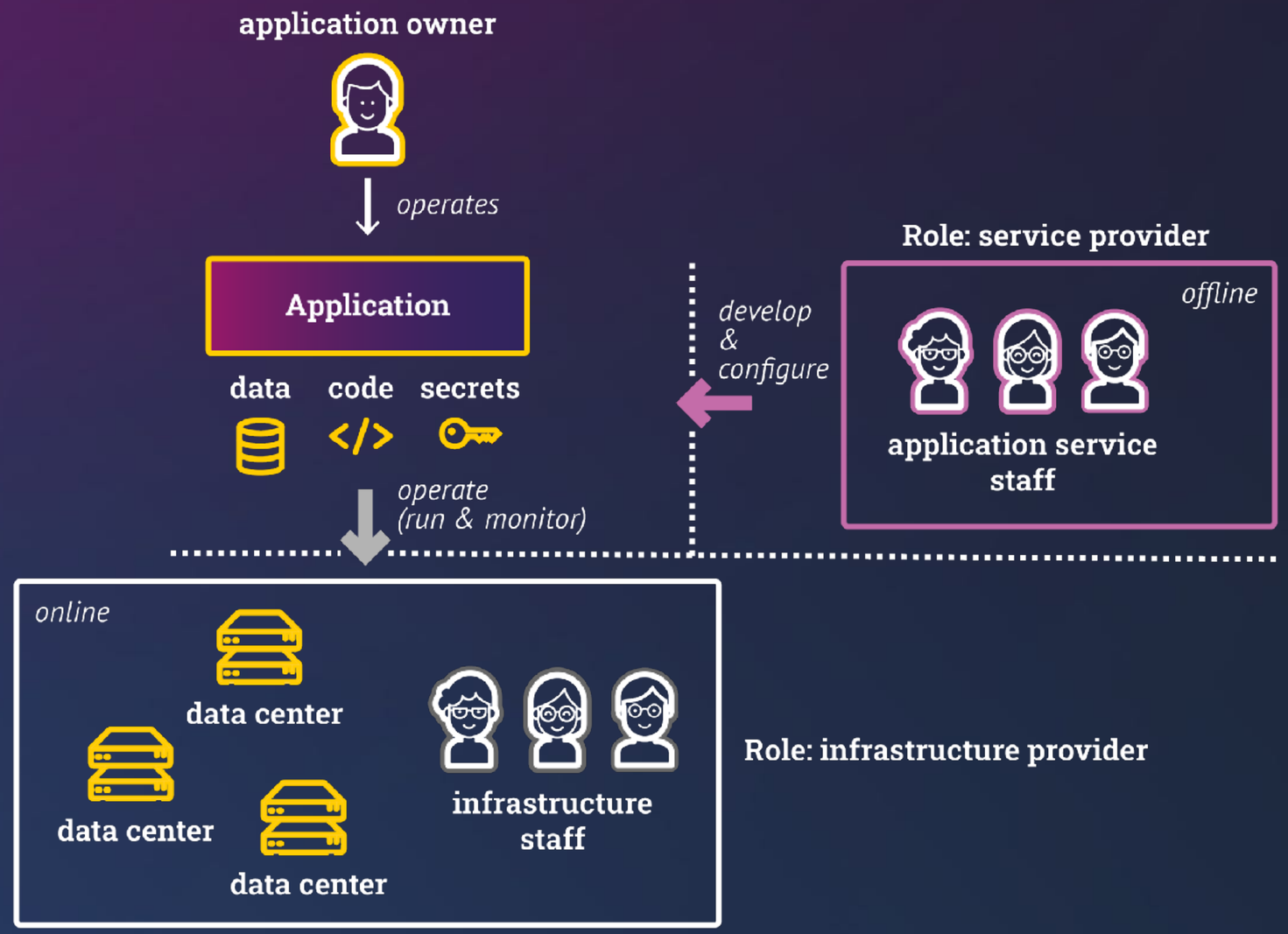
Problem: application owner cannot operate the application

- lack of data centers || trusted infrastructure staff
- **lack of application service staff**



Approach: Outsource!

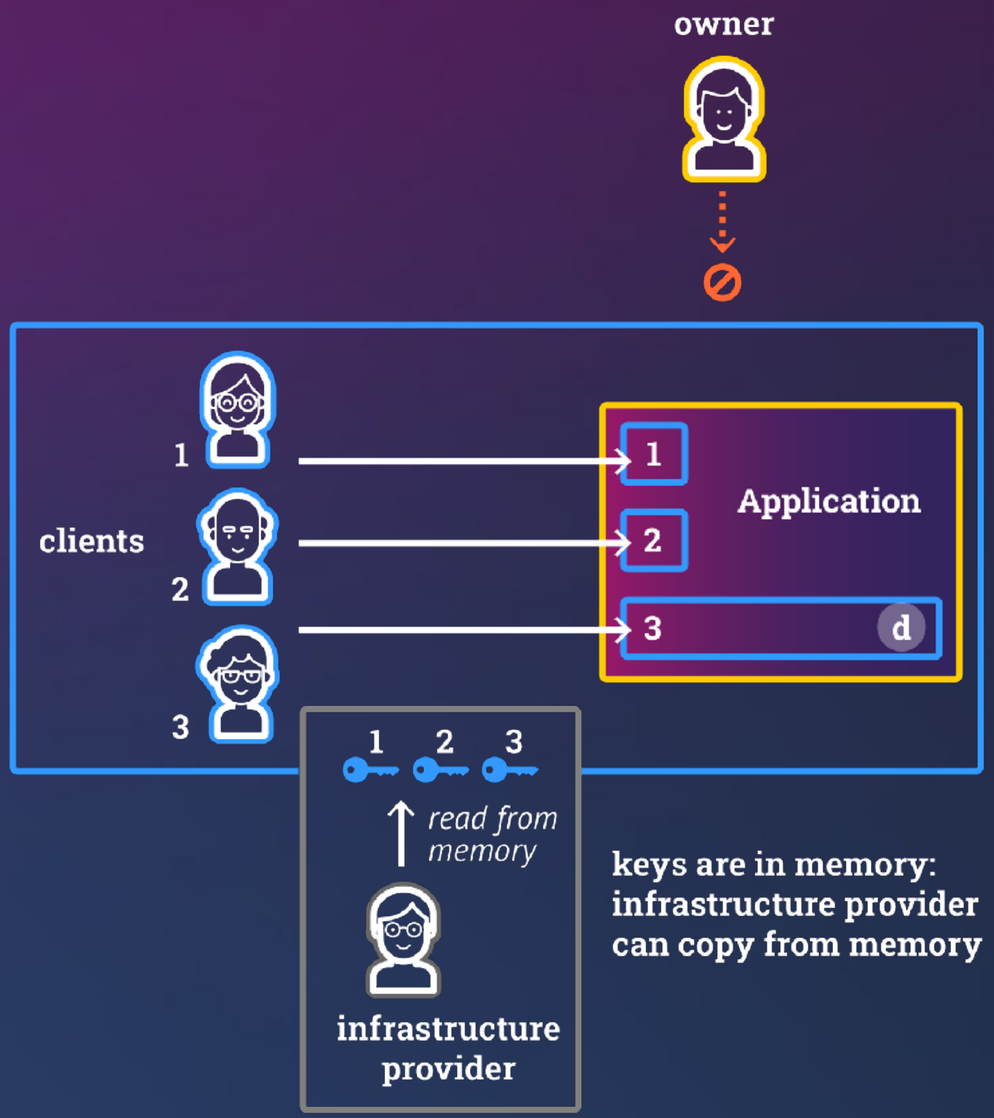
- Approach: external entities
  - operate data centers, and
  - manage application development



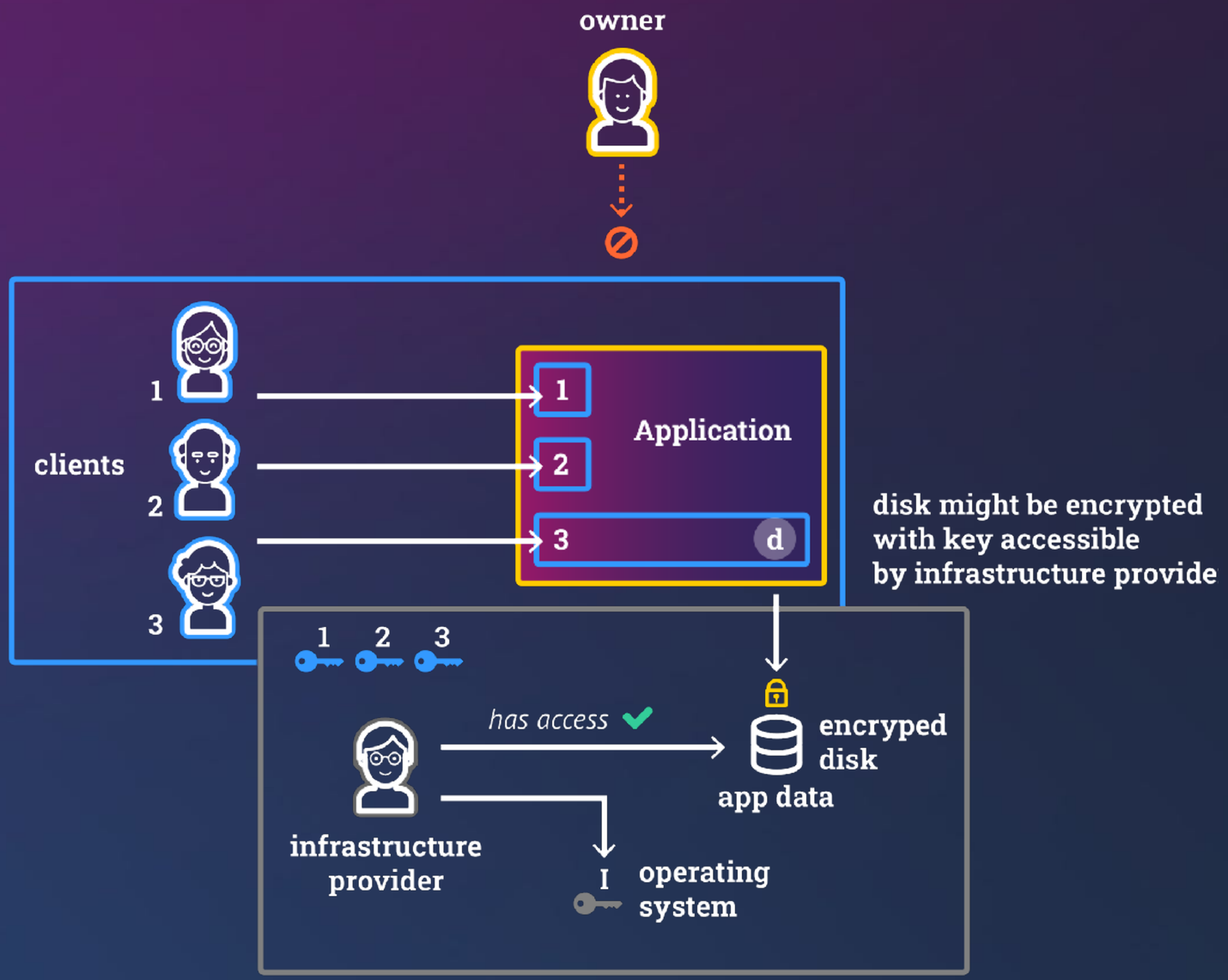
Technical Problem Description

- a small selection -

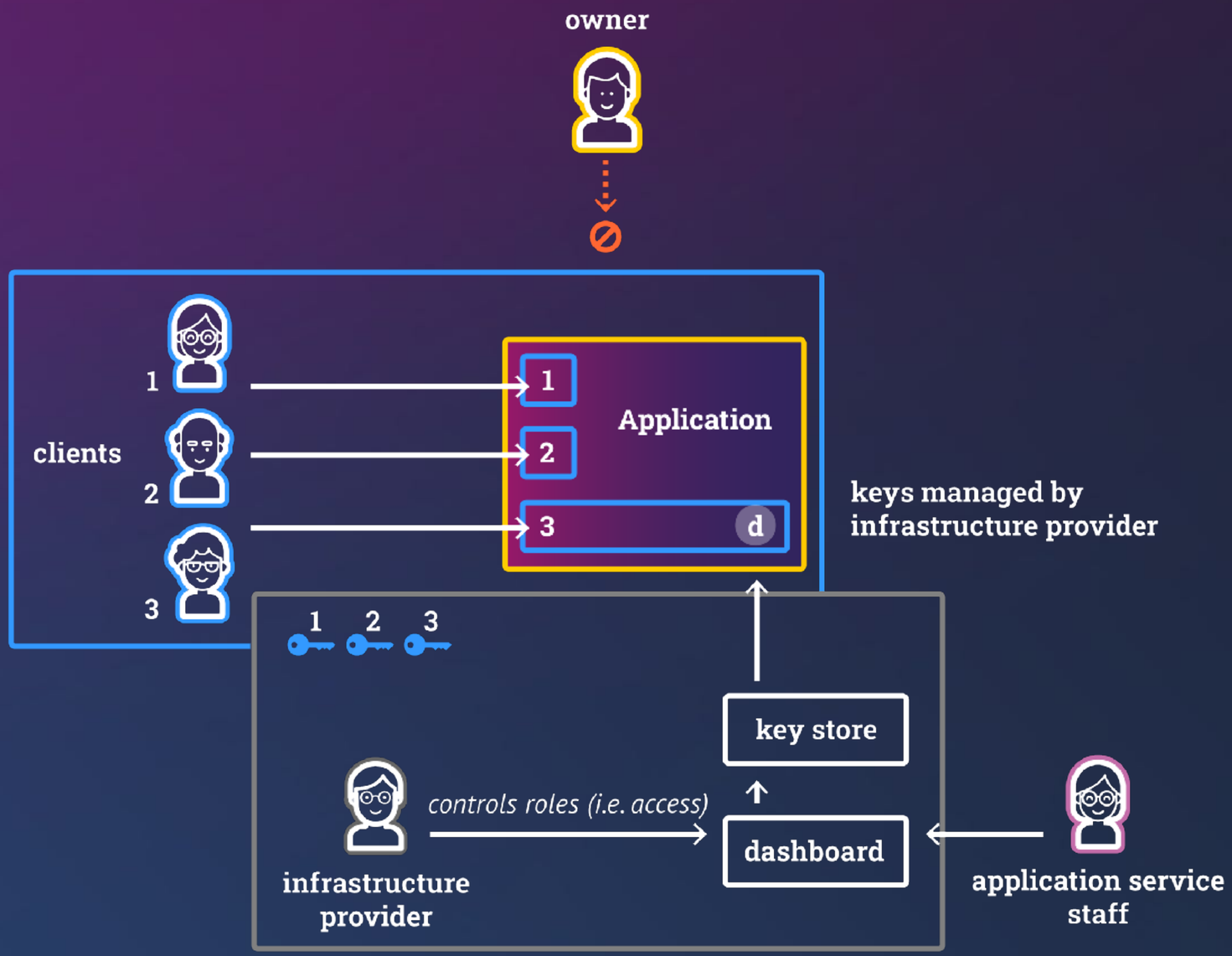
Problem: Hardware & Admin Access



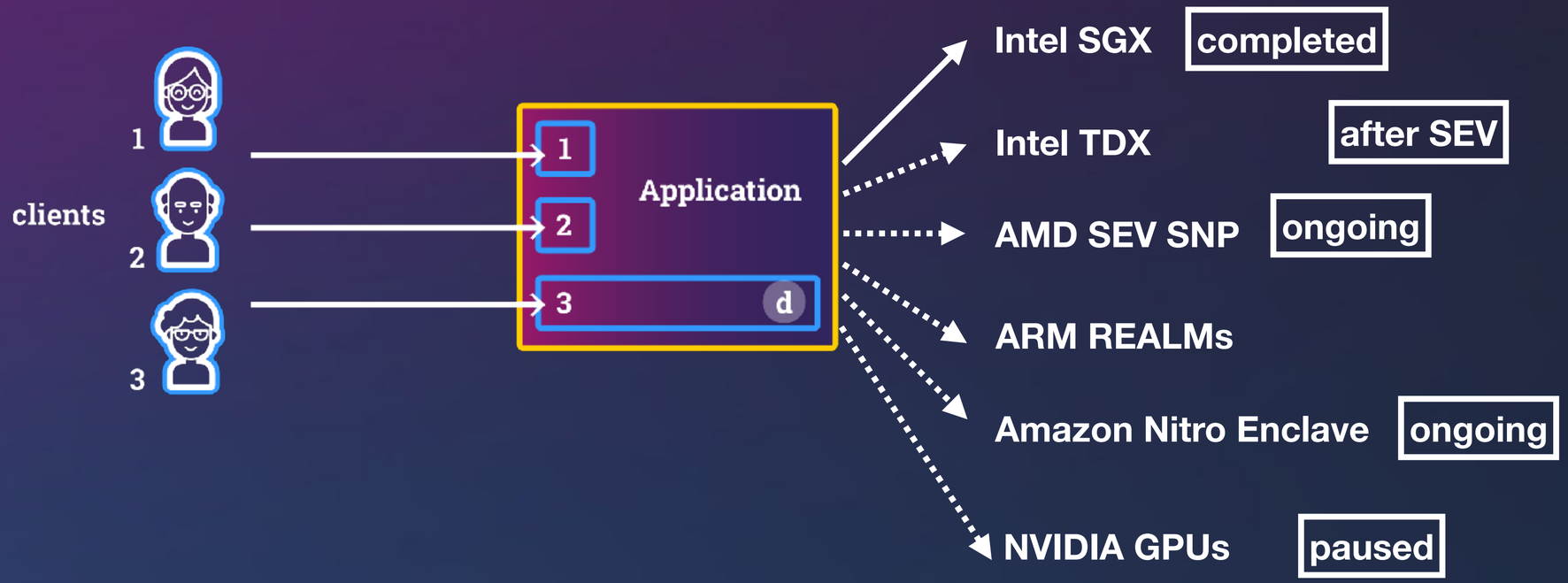
Problem: Encrypted Disks



Problem: Key Management



Supporting Different CPUs/GPUs



SCONE architecture is independent of hardware: it requires access to some TEE.

Threat Model & Implications

- we need to support untrusted components / stakeholders -

Threat Model

owner

infrastructure quickly evolving, application owner cannot vouch for security

Implication:
We need to ensure no access to source code, data or any keys

infrastructure provider

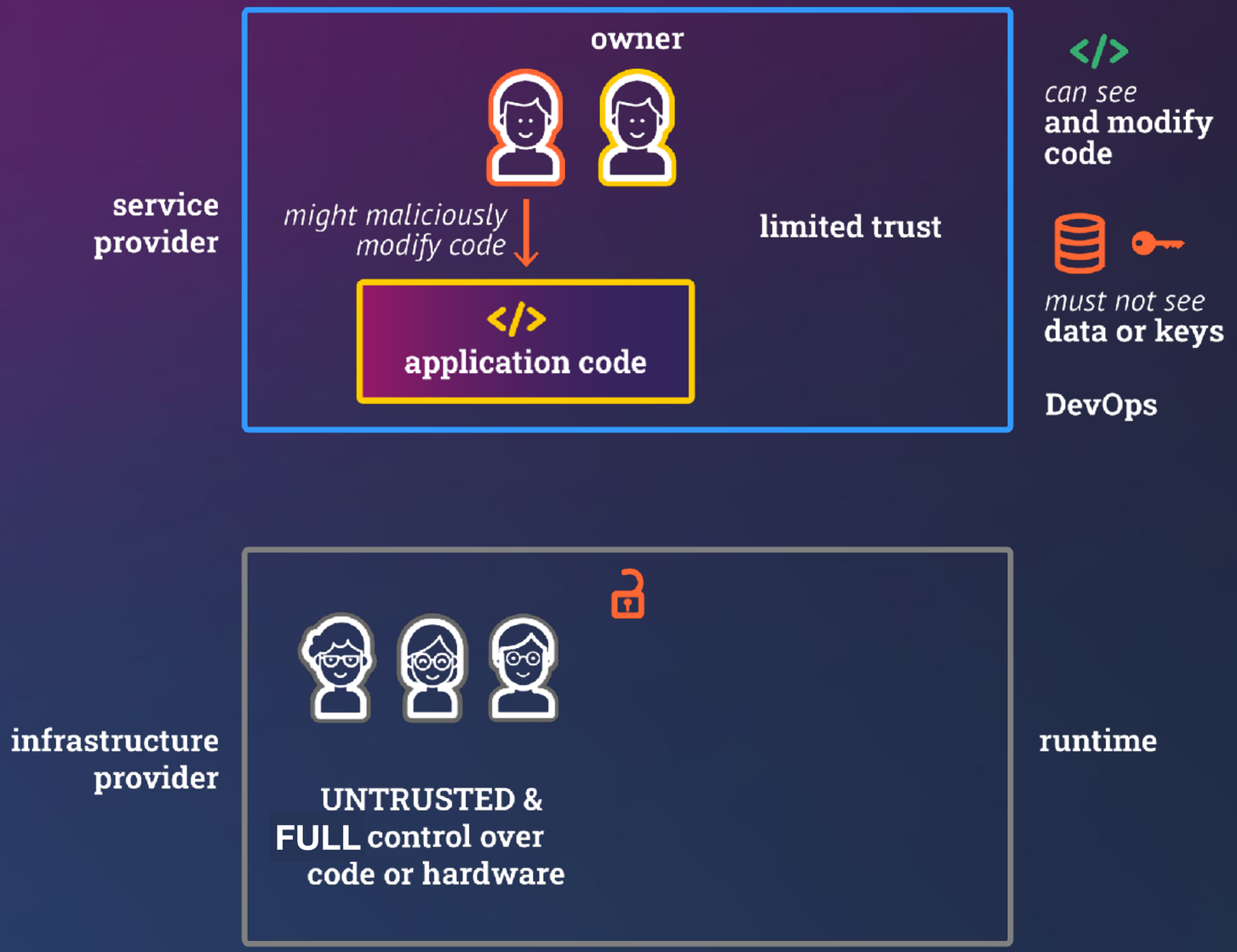
UNTRUSTED & **FULL** control over code or hardware

managed hypervisor, operating system, Kubernetes, key store, access control, ...staff members

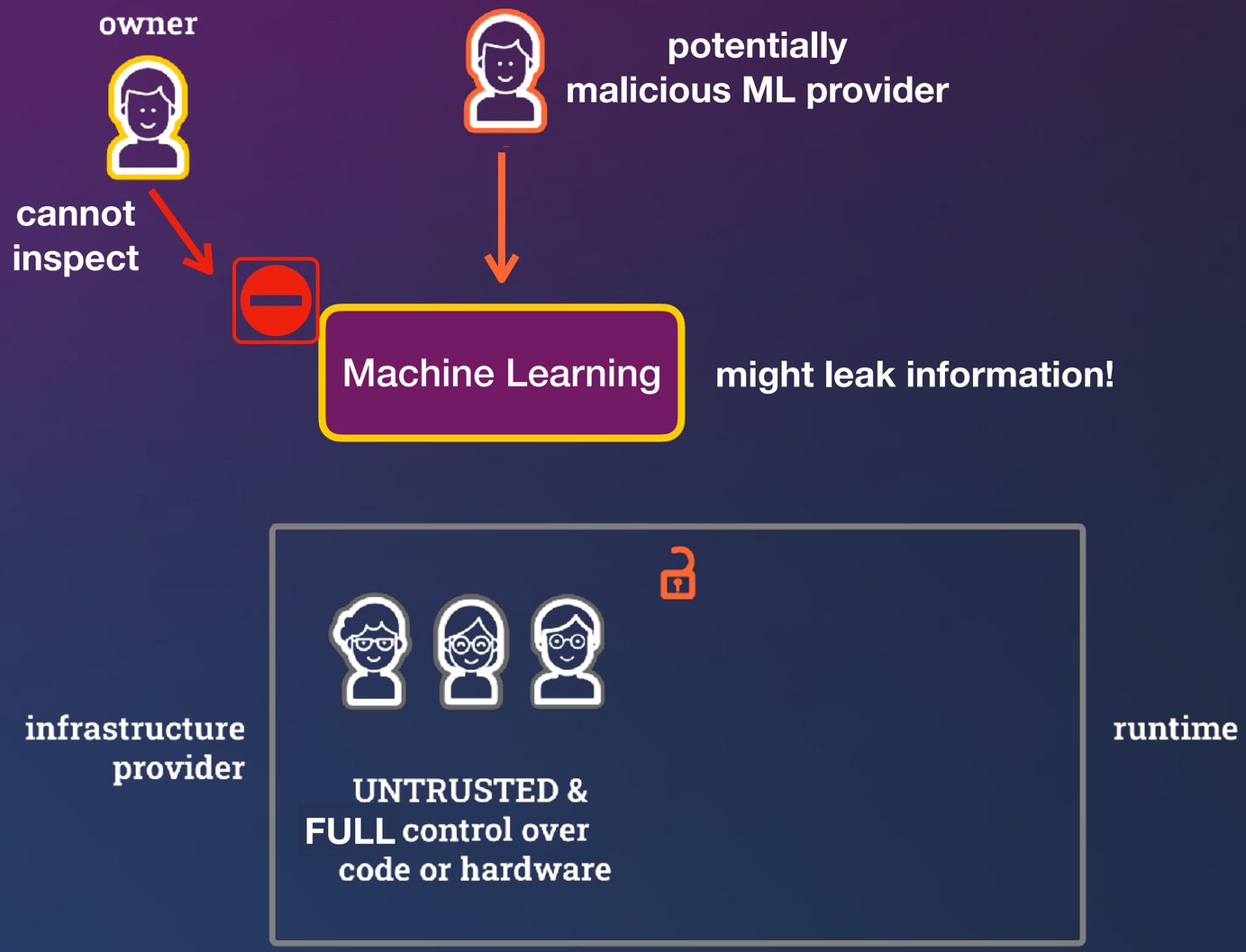
are ALL UNTRUSTED

runtime

Threat Model: Modified Code



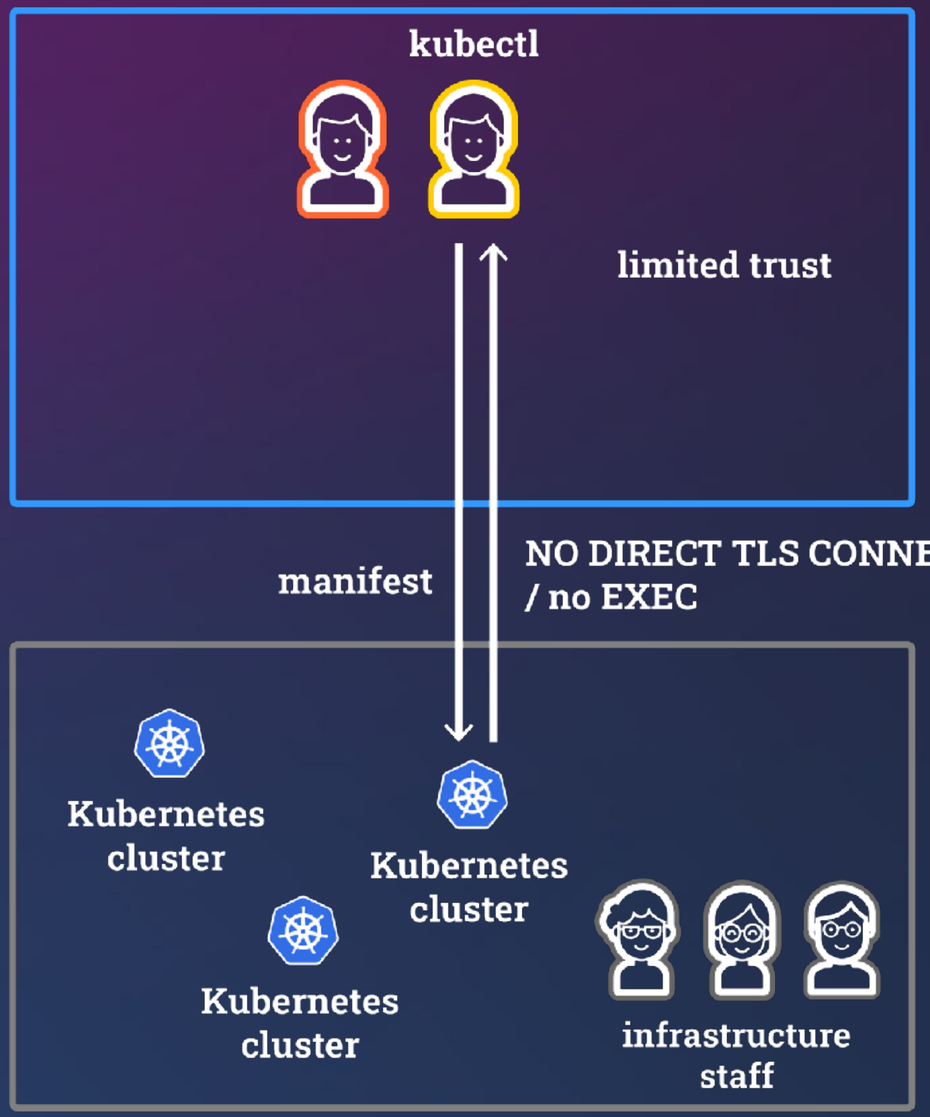
Example: Machine Learning Code



Approach

1. Level: No Direct Data Access by Service Staff

limit service staff access to infrastructure



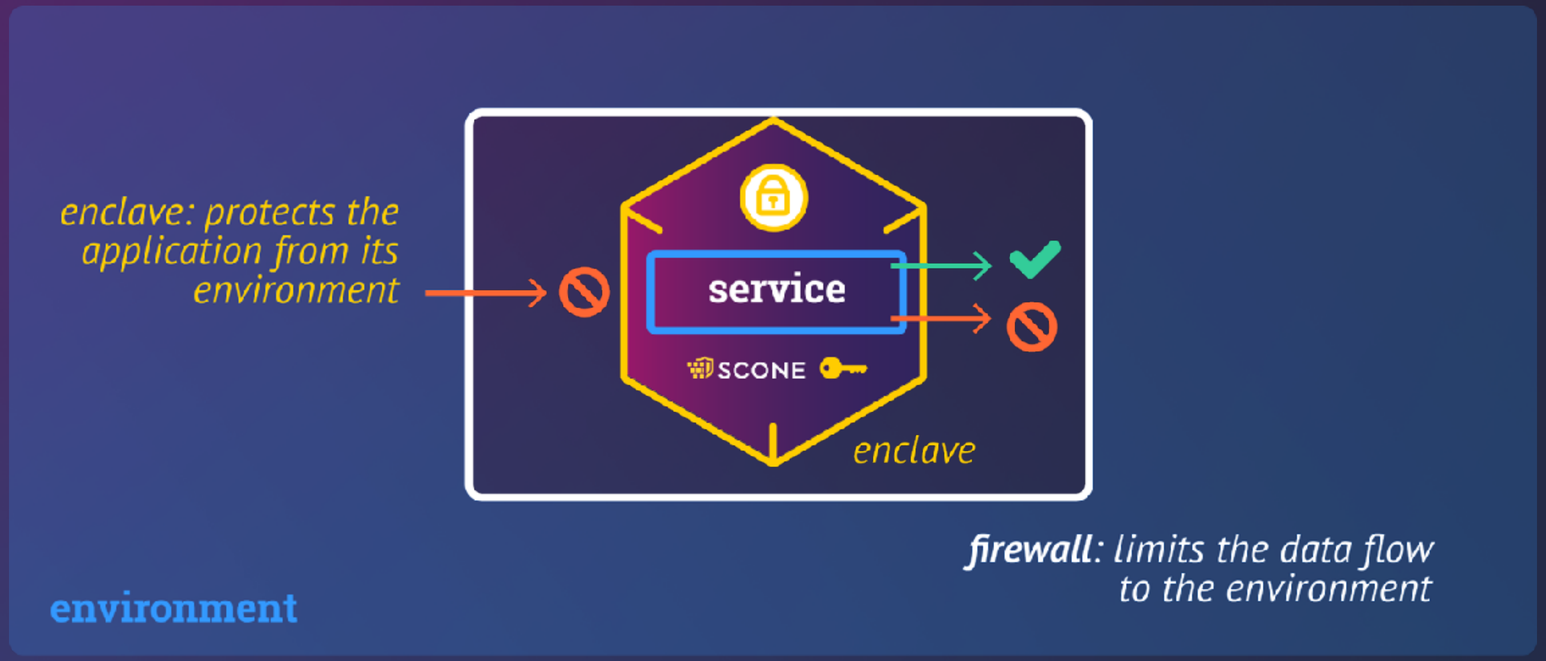
we can monitor all interactions by service provider

must not be able to access runtime cluster

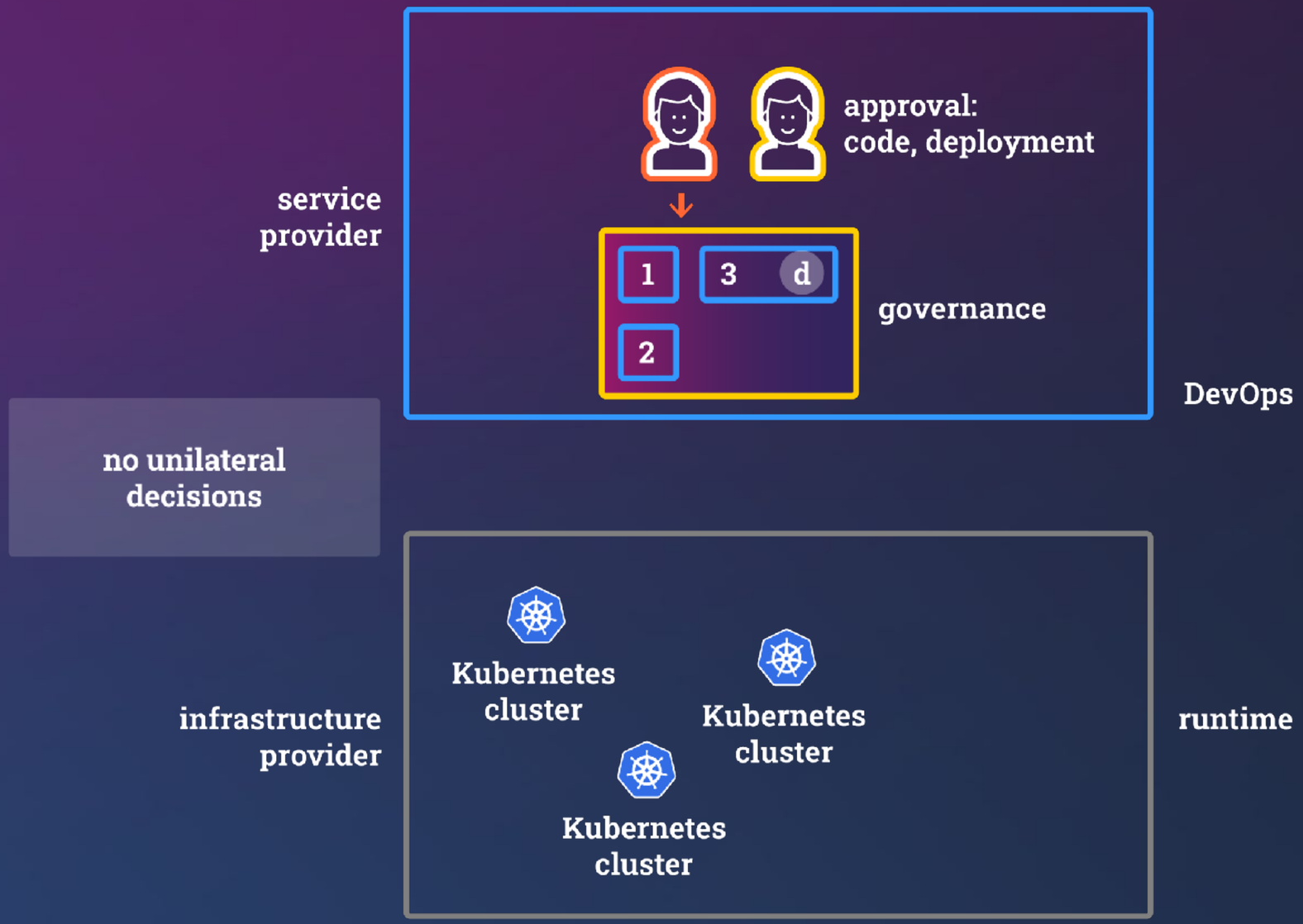
Level 2: TEE + Sandbox

- protecting data, code & secrets under policy control -

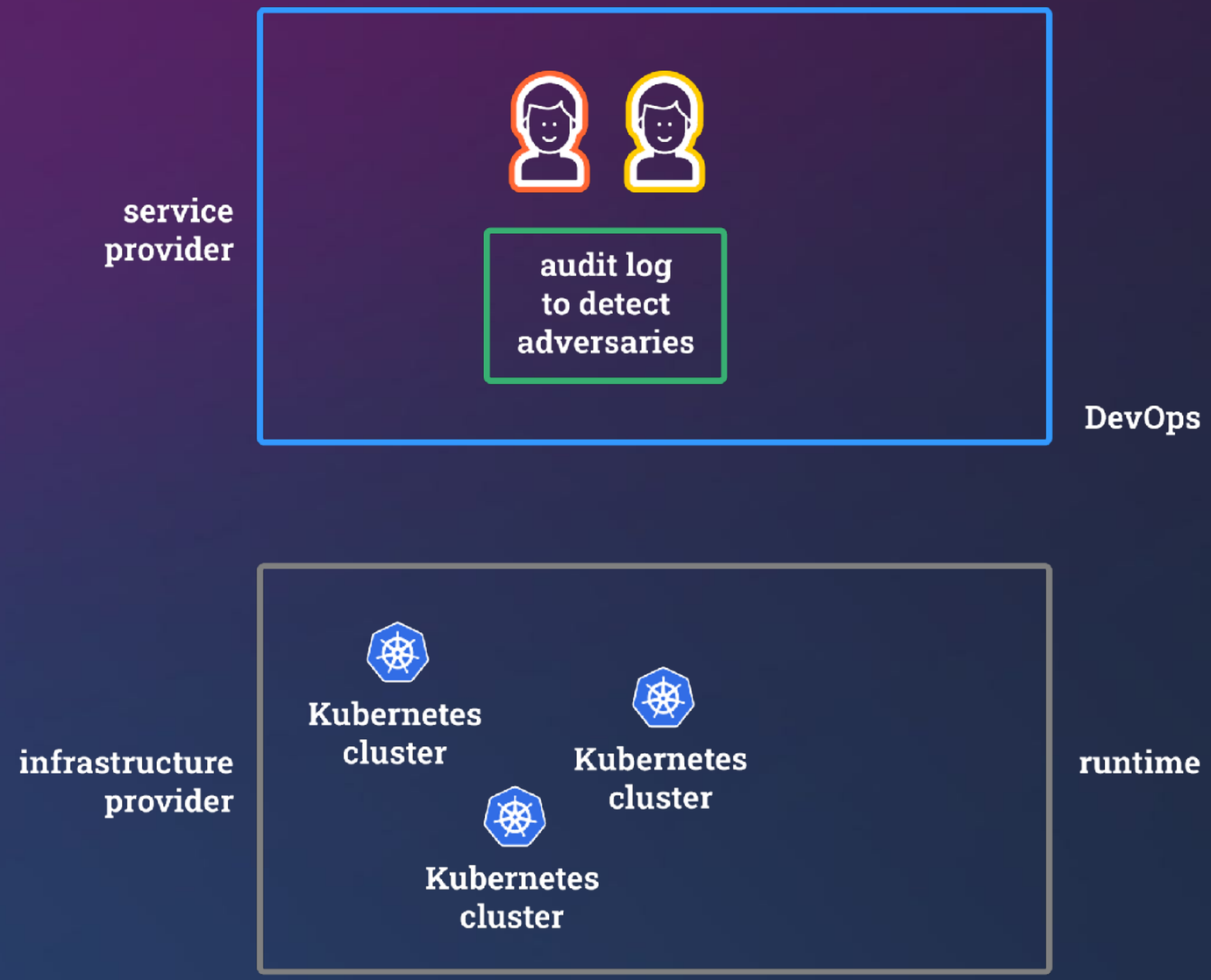
all communication of services controlled by policies



3. Level: Governance



4. Level: Non-Repudiation



Details

https://sconedocs.github.io

Sconification

- Transforming Native Application into Confidential Application -

„3“ Steps To Confidential App

1

Build app-specific images

2

Build application mesh

3

Start application (with generated helm chart)

```
# build the application by building images and using custom images
sconectl apply -f FastApi.yml       # generates a confidential image
sconectl apply -f Meshfile.yml      # generates and uploads the policies

# deploy the application
helm install secure-doc-management target/helm    # use helm chart to install
```

„3“ Steps To Confidential App

build in a trusted environment

test & code audit

1

Build app-specific images

2

Build application mesh

3

Start application (with generated helm chart)

```
# build the application by building images and using custom images
sconectl apply -f FastApi.yml       # generates a confidential image
sconectl apply -f Meshfile.yml      # generates and uploads the policies

# deploy the application
helm install secure-doc-management target/helm    # use helm chart to install
```

„3“ Steps To Confidential App

1

Build app-specific images

2

Build application mesh

3

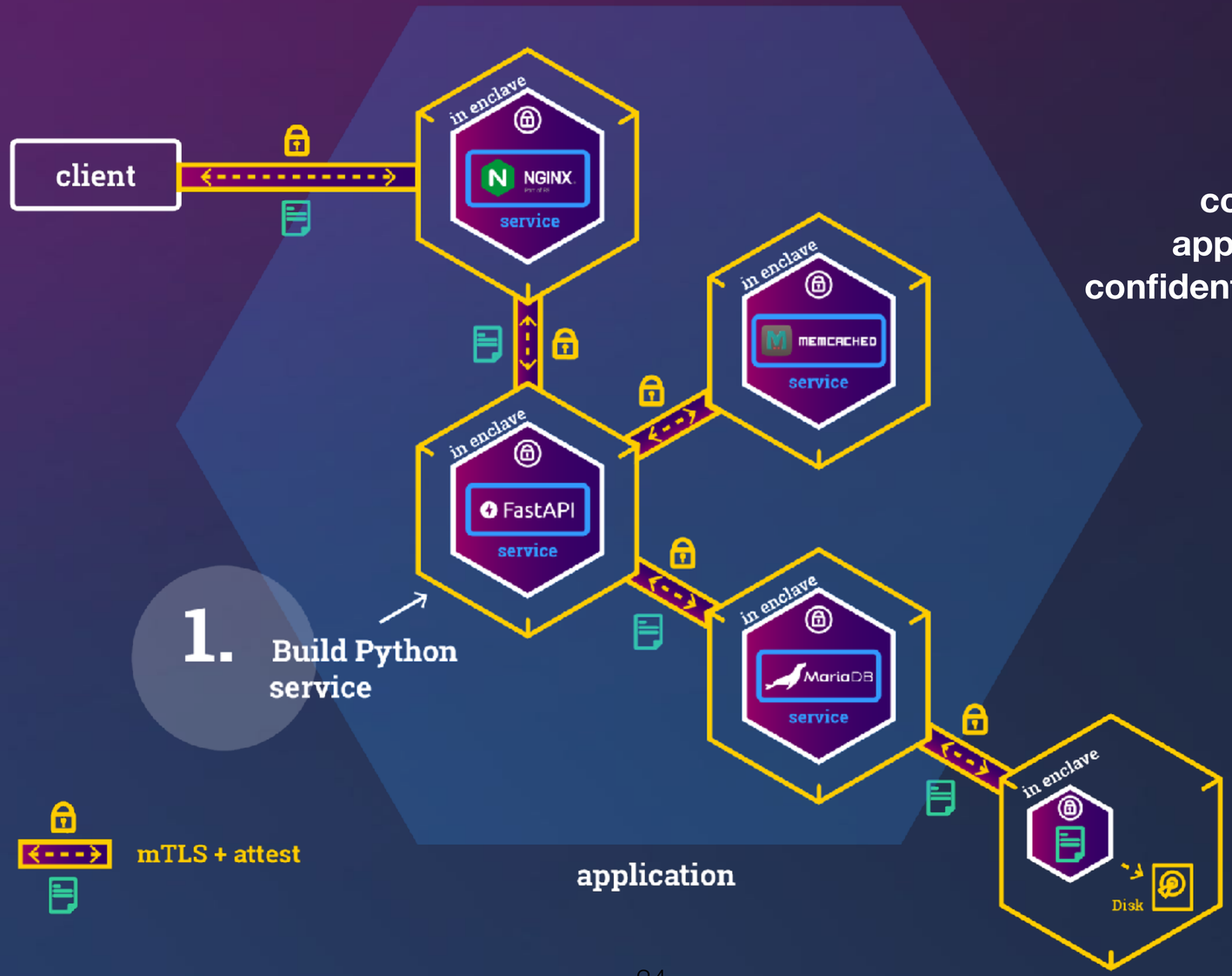
Start application (with generated helm chart)

execution in an untrusted environment

```
# build the application by building images and using custom images
sconectl apply -f FastApi.yml       # generates a confidential image
sconectl apply -f Meshfile.yml      # generates and uploads the policies

# deploy the application
helm install secure-doc-management target/helm    # use helm chart to install
```

Example Application

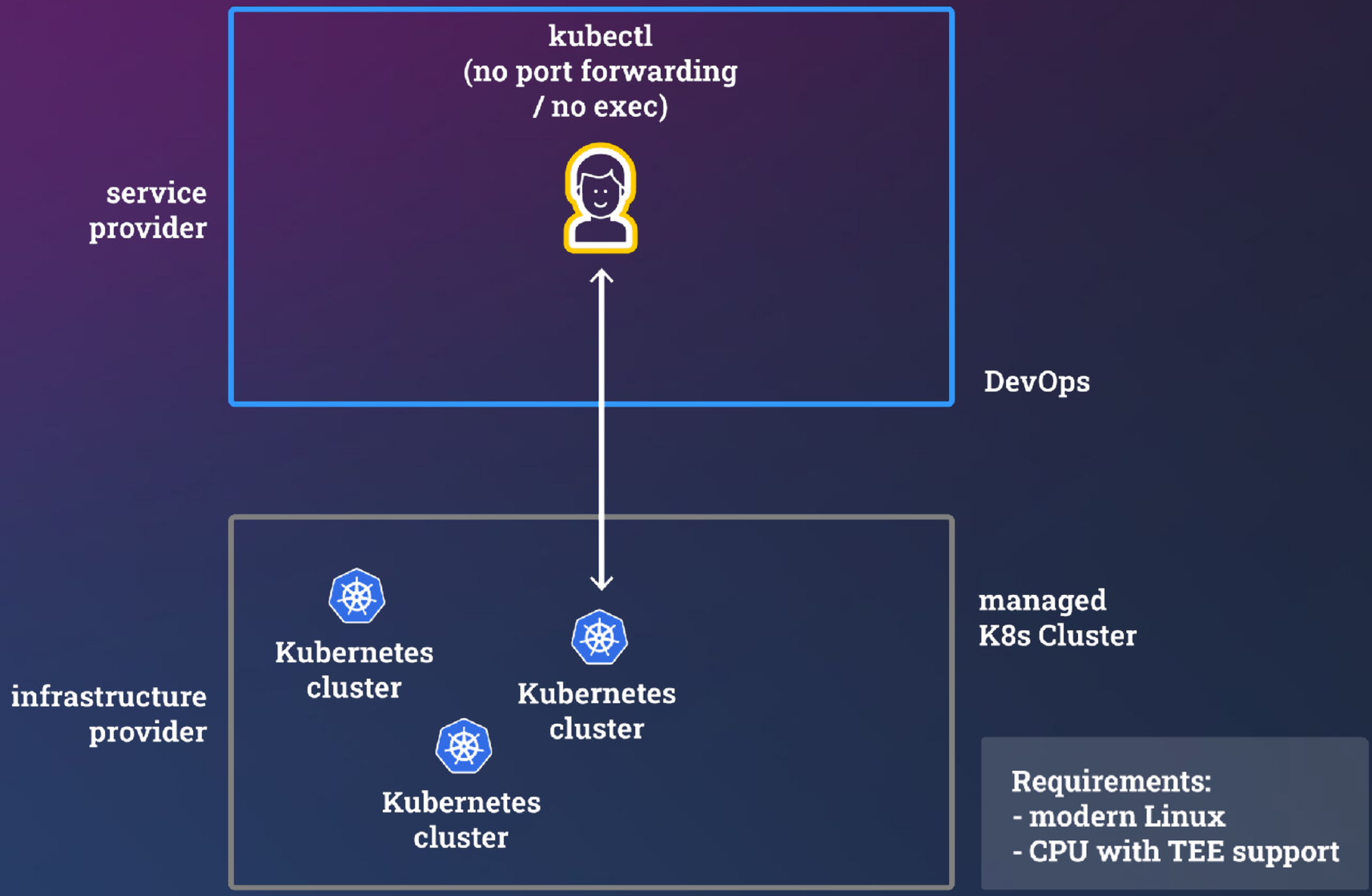


confidential application is a confidential service mesh

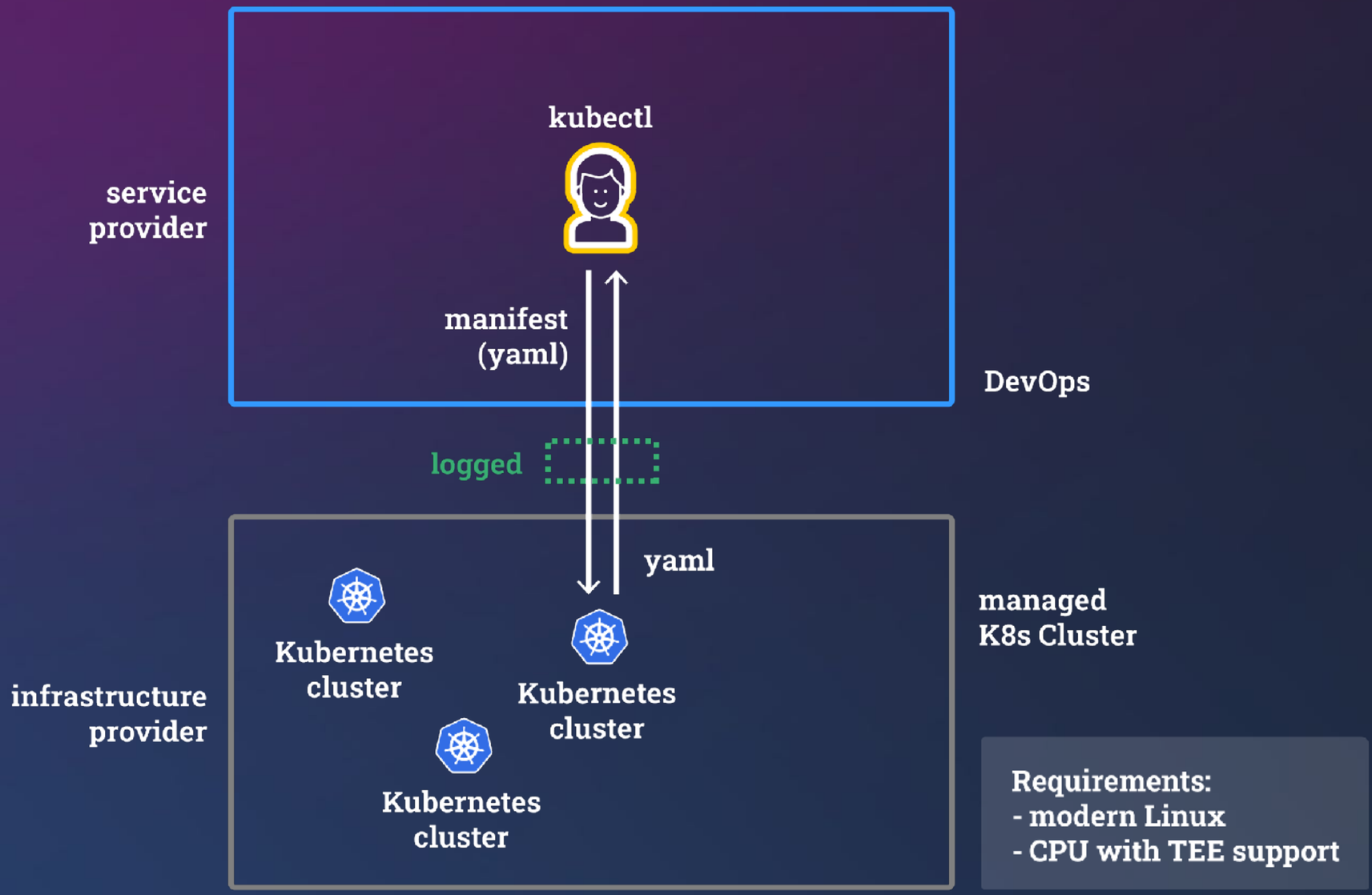
Level 1: Isolation

- establishing trust anchor with SCONE -

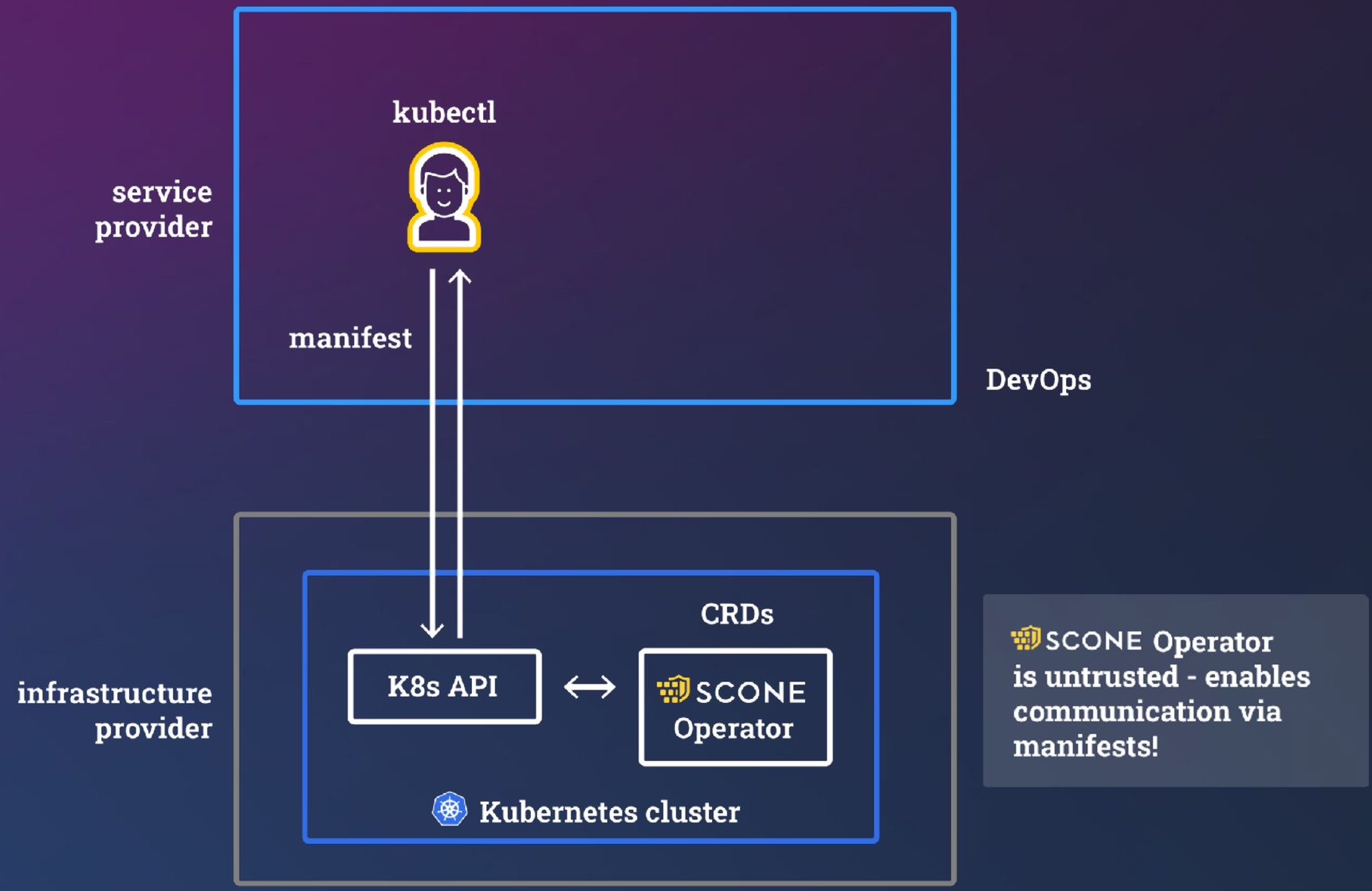
Remote Kubernetes Cluster



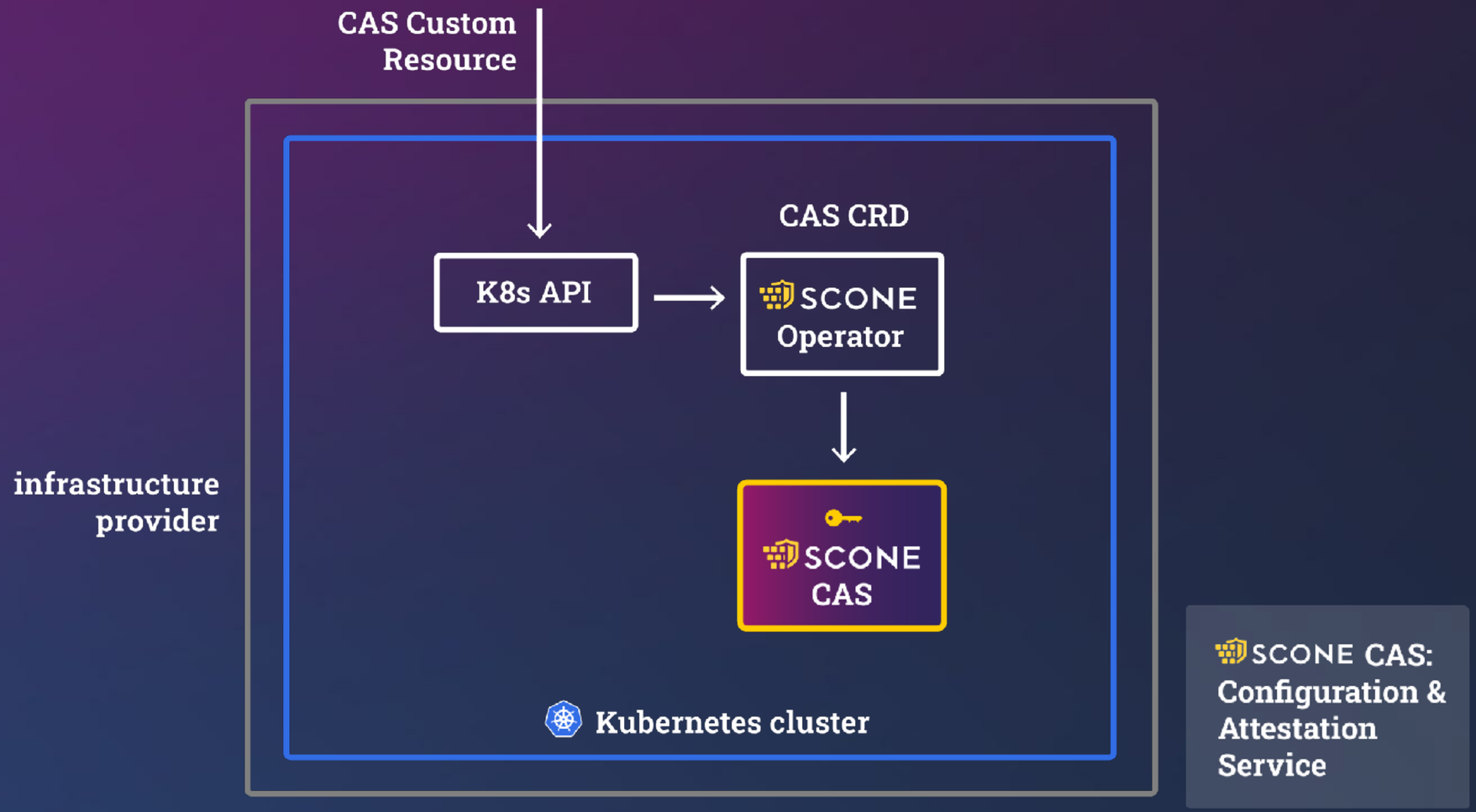
Accesses Can be Logged



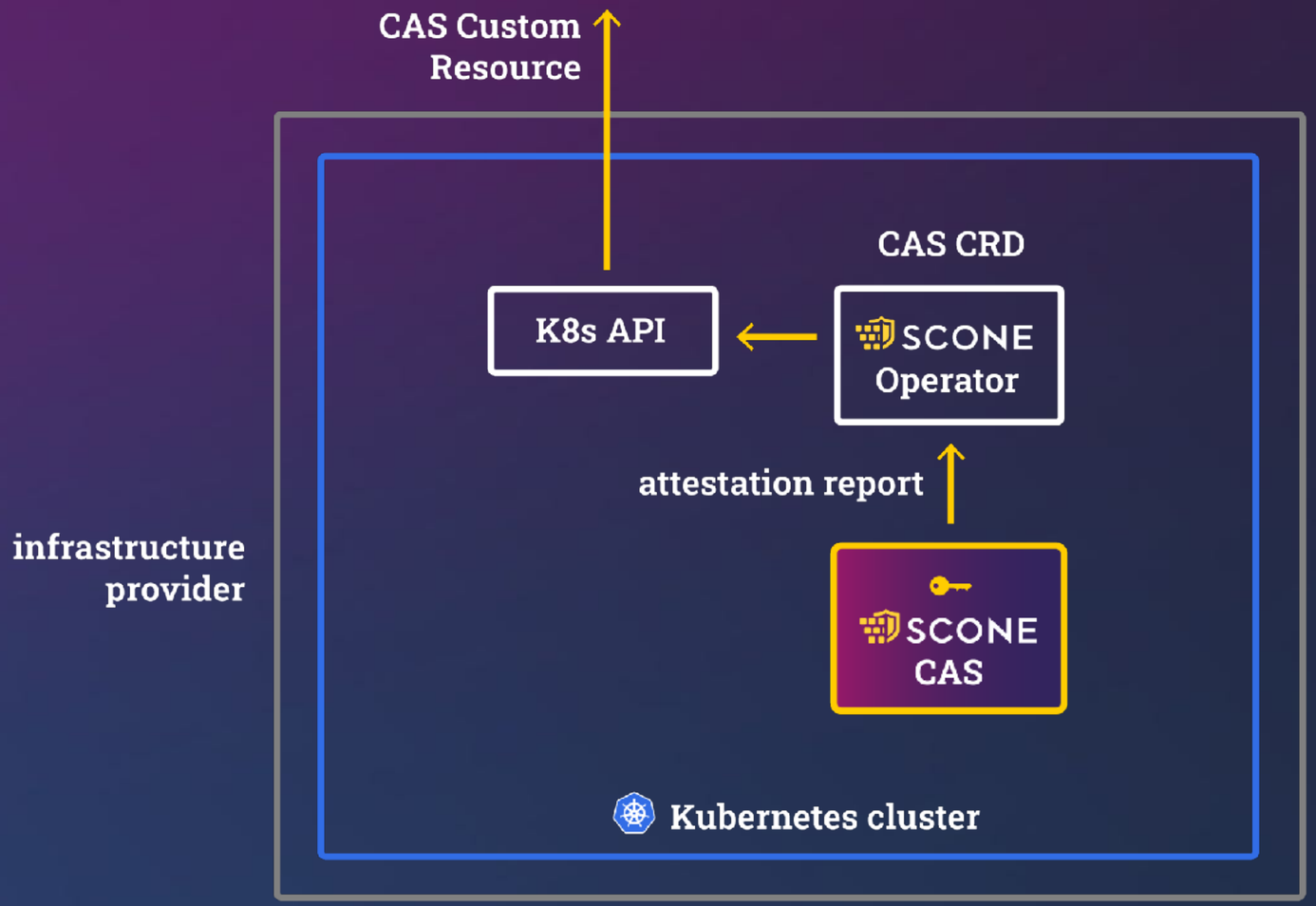
SCONE Operator



SCONE CAS: Policy Engine in TEE

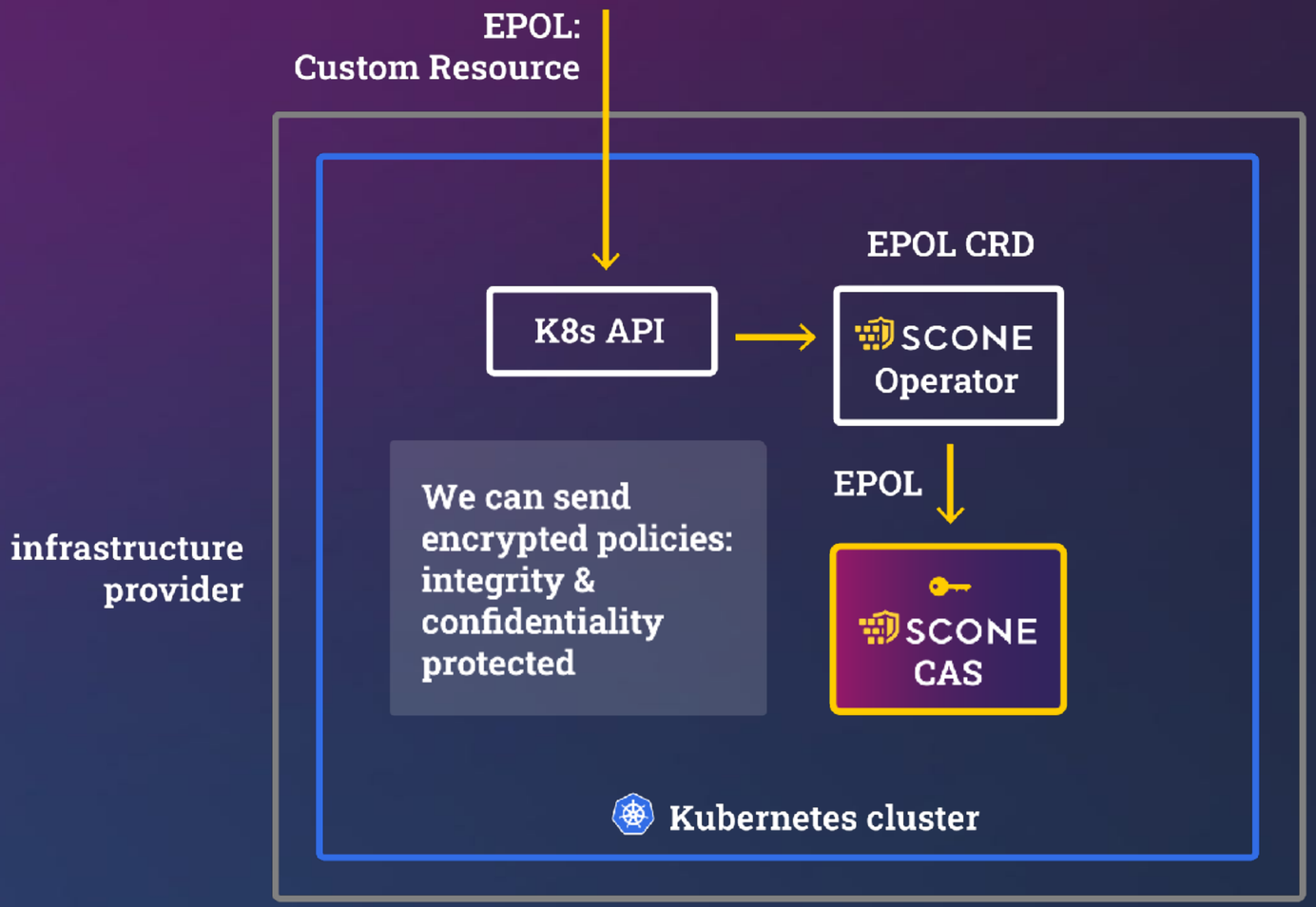


SCONE CAS

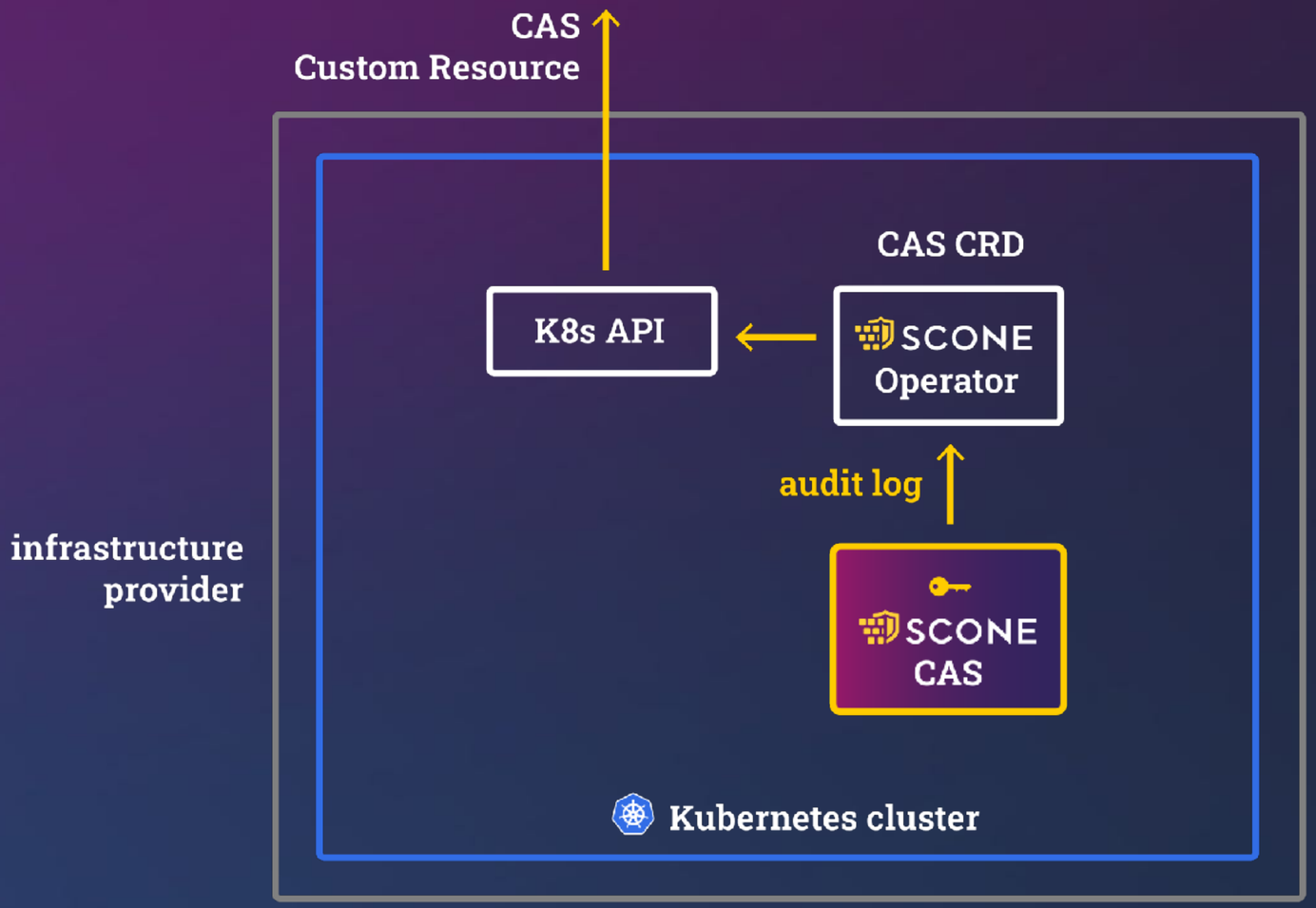


We can verify the attestation report

Encrypted Policies

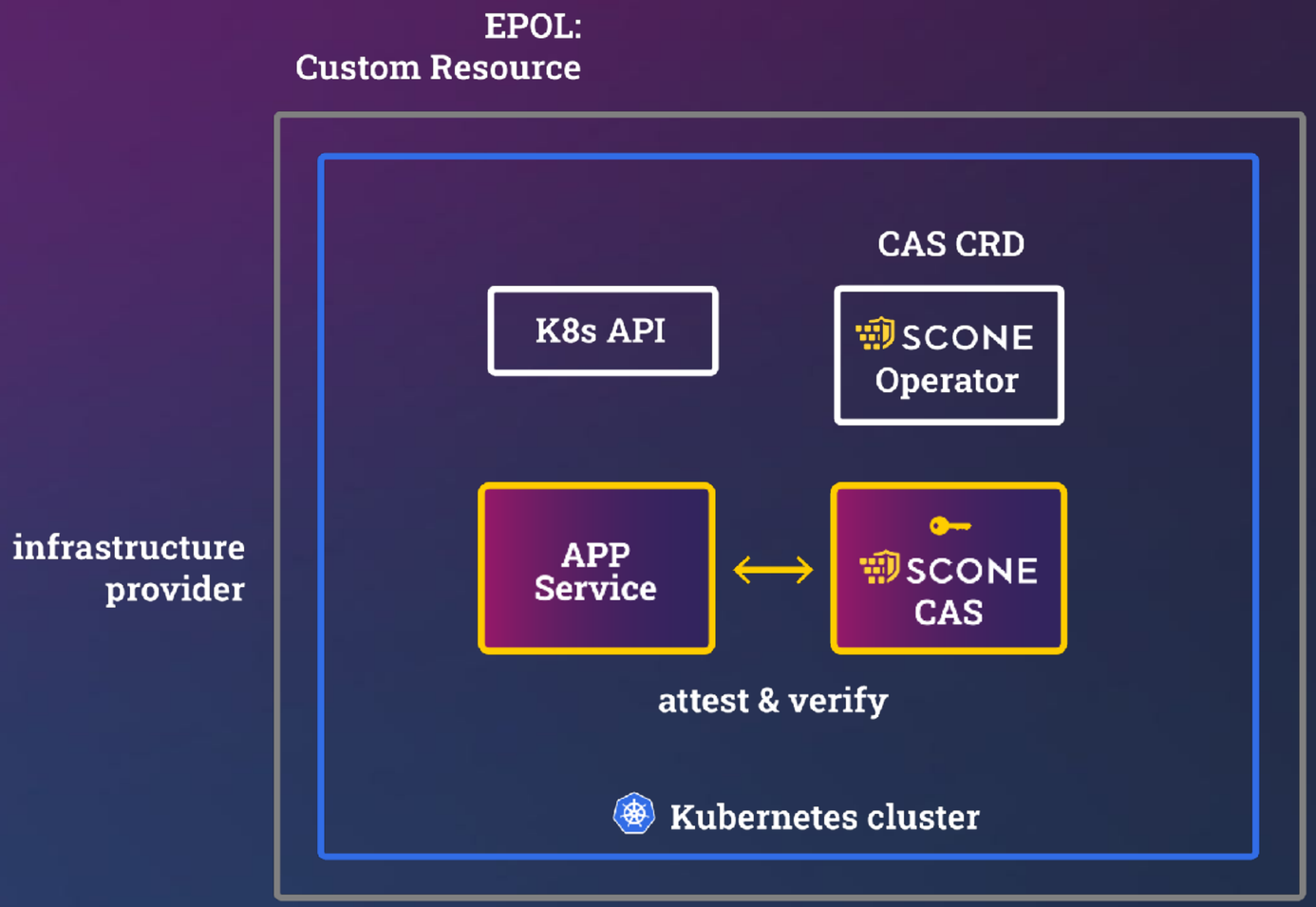


Audit Log



We can verify the creation via cryptographic audit log

Starting Confidential Applications

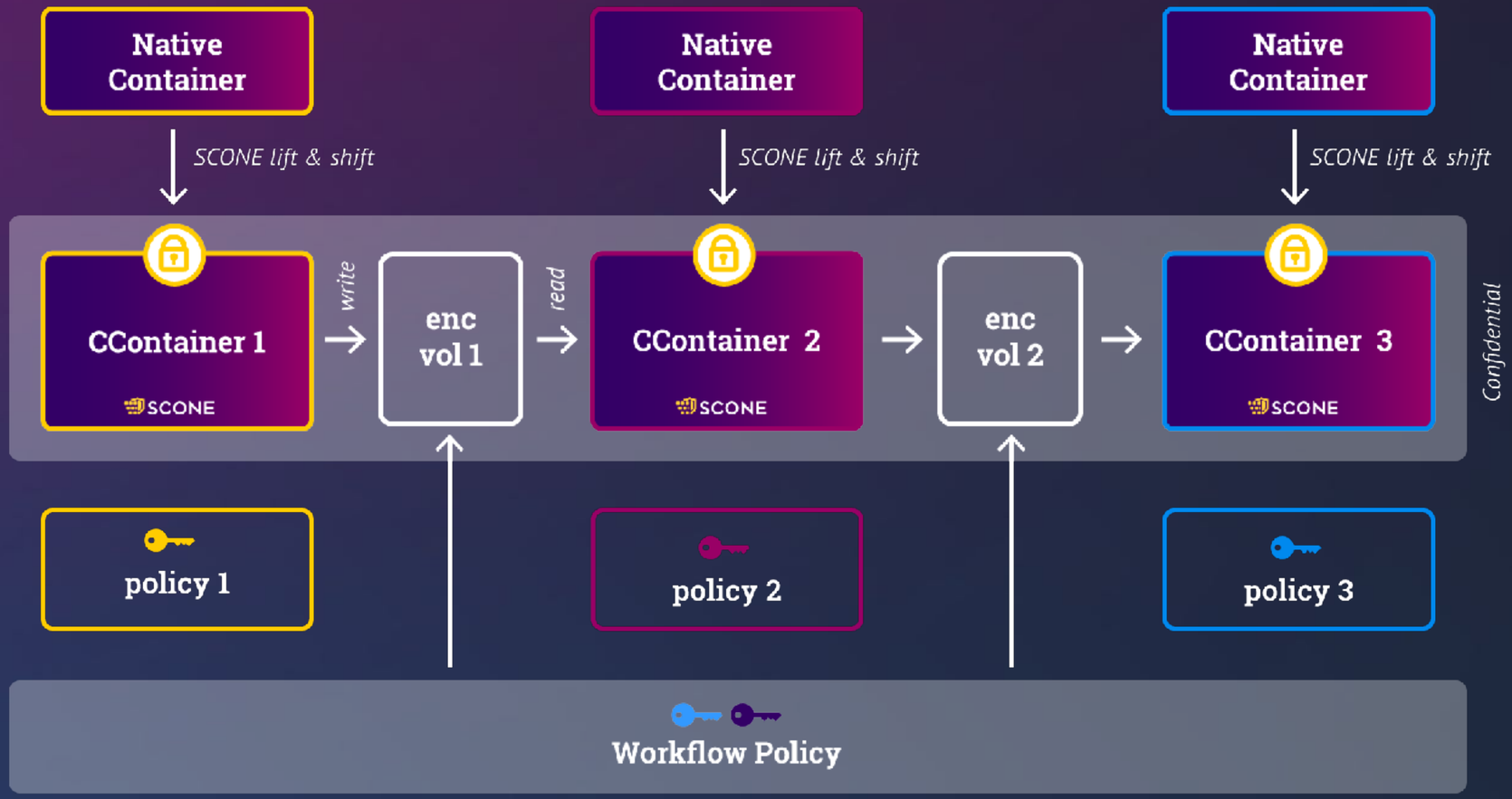


2. Level Confidential Workflows

- TEE & Sandboxing under Policy Control -

2. Level: Confidential Workflow

1-step binary transformation **images**



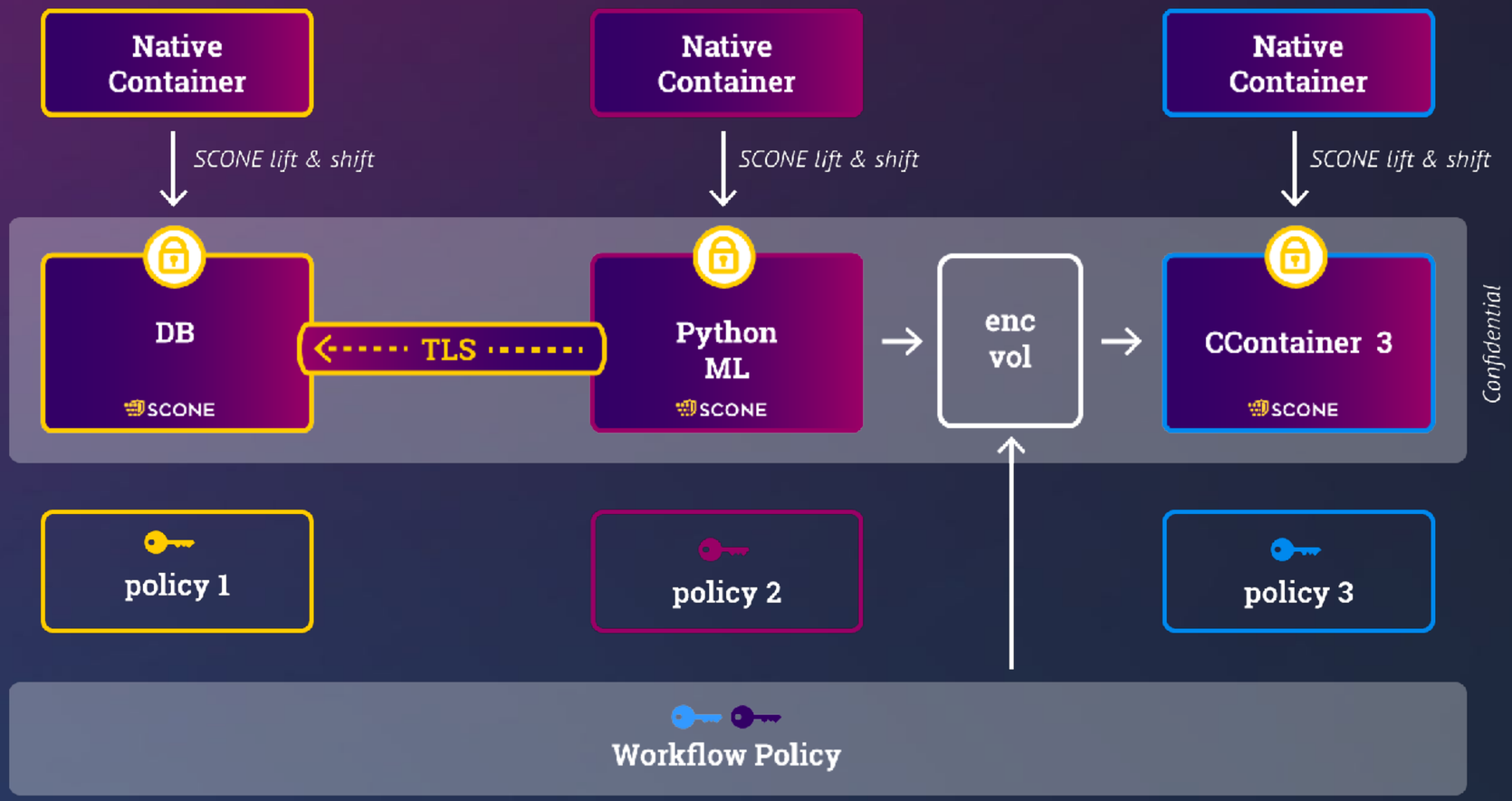
Each policy protects resources of its stakeholder

All stakeholders: can inspect workflow policies (no secrets).

A **policy** can connect a workflow

Use Case: Multiple Stakeholder Computation!

1-step binary transformation **images**



Each policy protects resources of its stakeholder

Application Domains:
Federated Learning,
eHealth, Manufacturing, …

A **policy** can connect a workflow

Level 3: Governance

- Multiple-Eyes Principle -

Protecting Against Insider Attacks

- malicious policies / code changes -

Insider Attack

An insider with policy access could change the policy

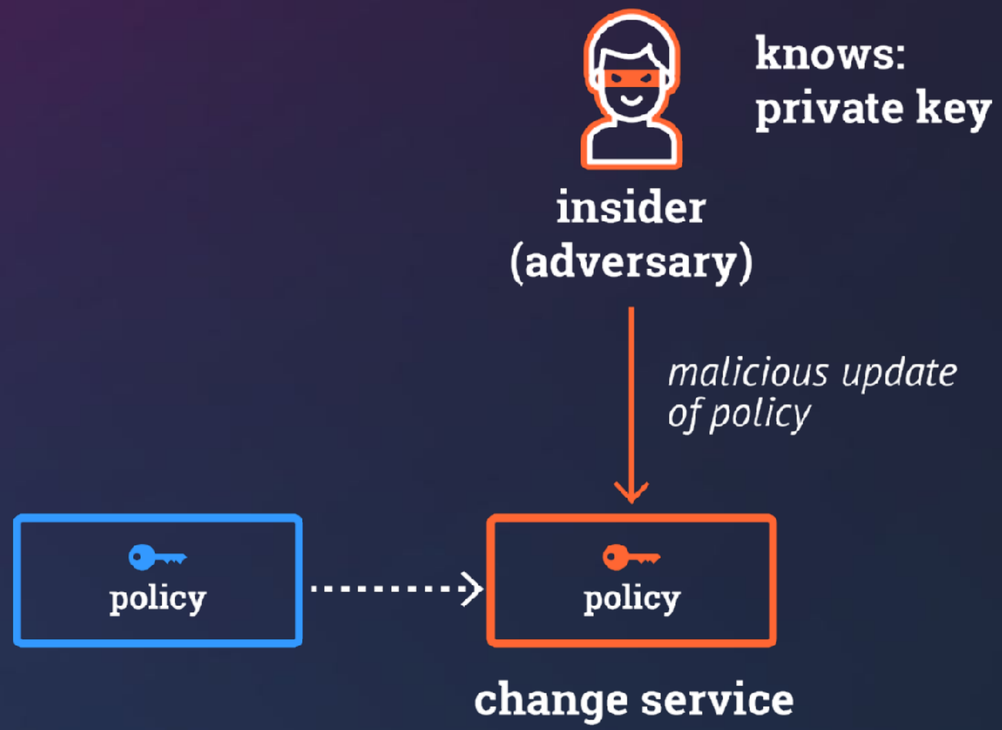
- to retrieve secrets, or
- to change the service

We can prevent this by

- **creating read-only policies**
- **exporting to a certain policy version only**
- **governance**

We can detect this by

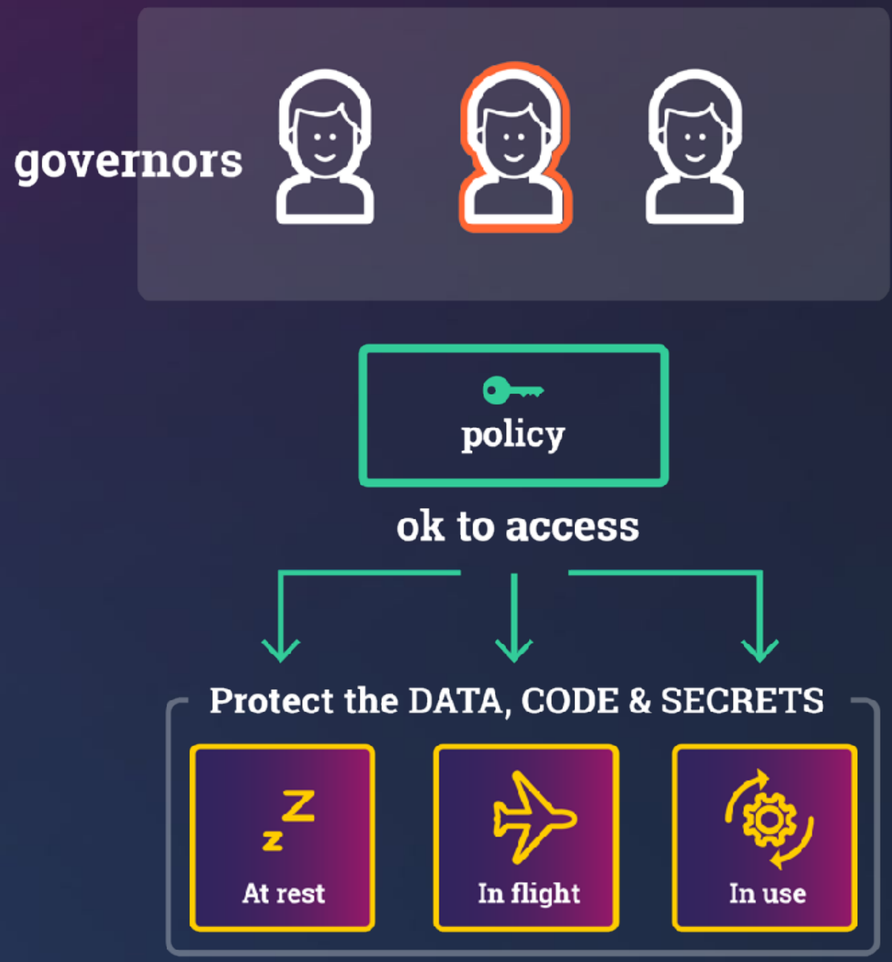
- auditing the immutable history of policies



Governance

Application owner

- wants to operate an application in a cloud
- hires admins that operate application
  - most are trusted
  - some might work for an adversary
- governance via governors



Governance

Application owner

- wants to operate an application in a cloud
- hires admins that operate application
  - most are trusted
  - some might work for an adversary
- governance via governors

Governance

Malicious Governors

- what if a majority is malicious?
- We should at least be able to detect his
  - by verifying policies and audit logs

Example

Requires an update to a session to be signed by

- both $veto_member1 and $veto_member2
- as well as at least 2 of the 3 $voter`s.

```
access_policy:
  read: NONE
  update:
     - require-all:
          - require-at-least-2:
               - signer: $voter1
               - signer: $voter2
               - signer: $voter3
          - require-all:
               - signer: $veto_member1
               - signer: $veto_member2
```

Example

Extend such that

- $owner can approve independently of the others

```
access_policy:
  read: NONE
  update:
    require-at-least-1:
      - signer: $owner
      - require-all:
          - require-at-least-2:
              - signer: $voter1
              - signer: $voter2
              - signer: $voter3
          - require-all:
              - signer: $veto_member1
              - signer: $veto_member2
```

Level 4: Non-Repudiation

- Audit Log -

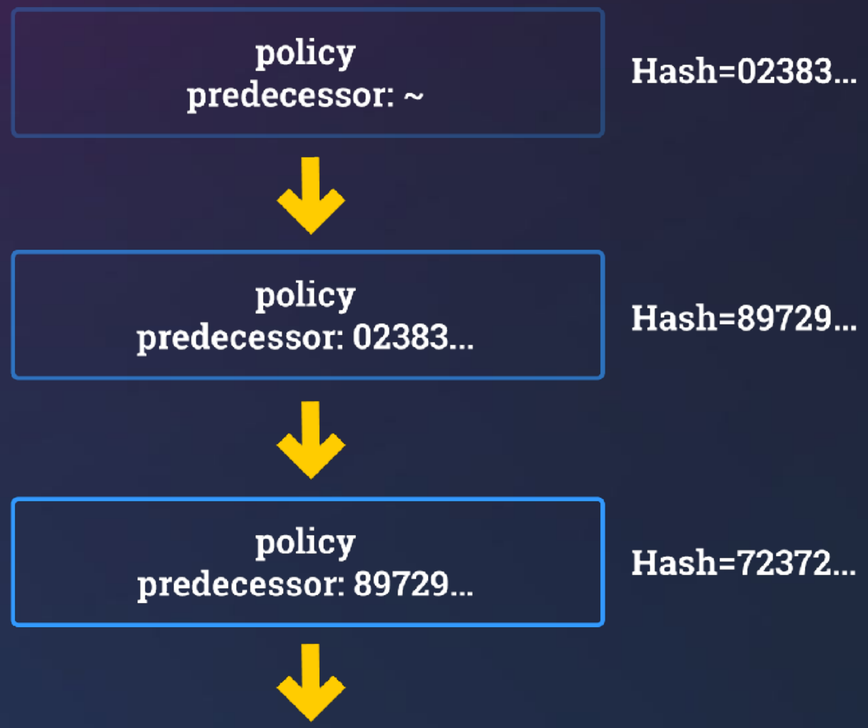
Predecessor List

Problem:

- any vulnerability in the past?
- concurrent updates of policy?

Approach:

- we chain together all policies (with same path name)
- **chain is append only**
  - policies cannot be deleted (no new start!)
- one can verify the past policies



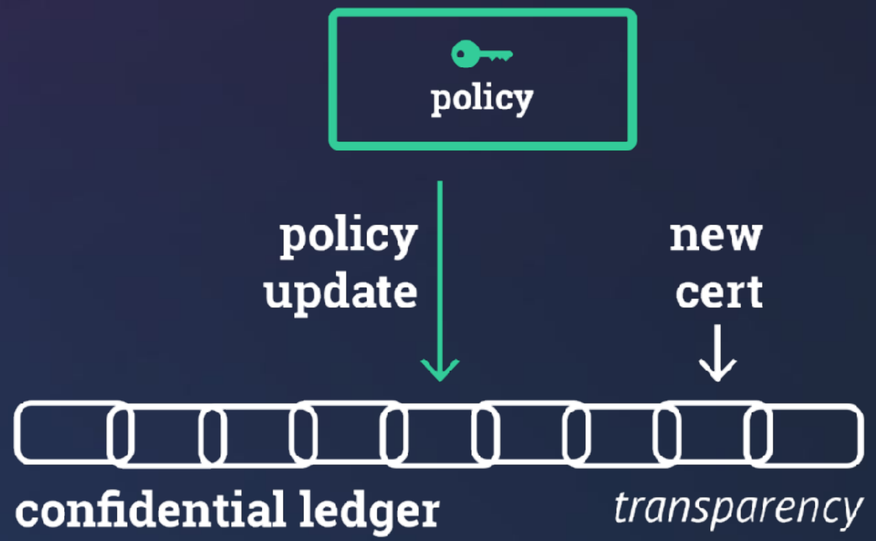
Audit

Problem:

- signed ledger of events needed
- hook to monitor updates in real-time

Approach:

- append-only ledger
- SCONE CAS:
  - appends security relevant events
  - signs all entries
- notifications via web hooks
  - can also store locally and push later (across air-gap)



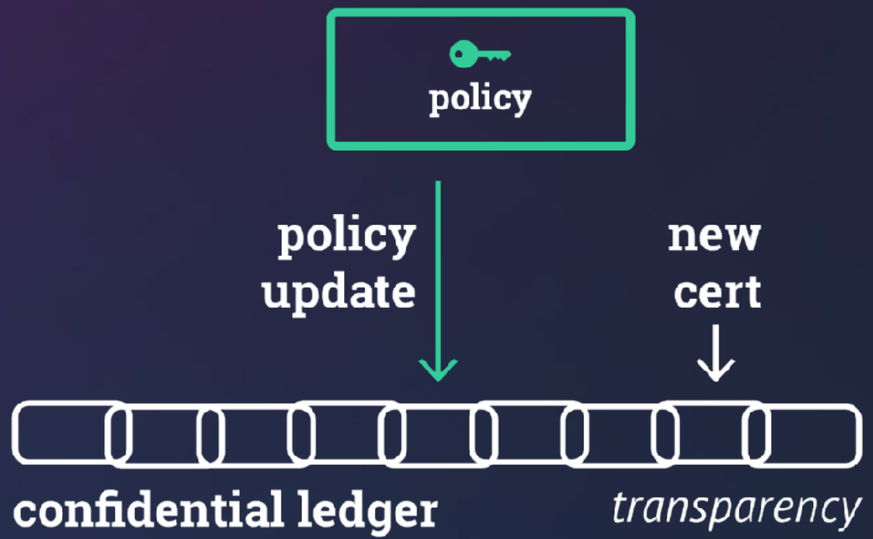
Audit Log Options

Audit-log

- **mode**: disabled/signed/unsigned
- **sink**: file / network
  - **file**: path
  - **network**:
    - url
    - server_ca_certificate

Verification:

- verify scone audit log with scone CLI



Summary

Governance and Audit

Services must not leak any data!

- despite managed by an external entity

Trust in system:

- show that data cannot leak now
- show that no data was leaked in the past
- show that we cannot leak data in the future

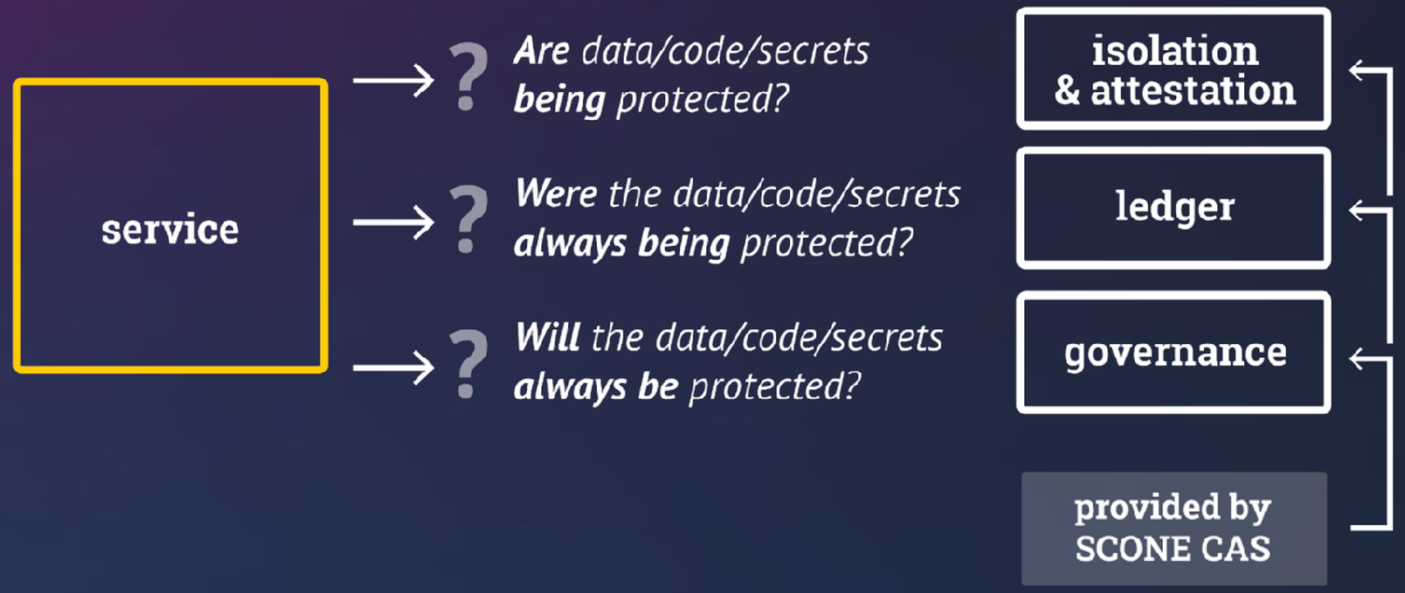
Enforced by SCONE CAS

Services must not leak any data!

- ensure integrity, confidentiality, consistency of data, code, and secrets

Trust in system:

- show that data cannot leak now
- show that no data was leaked in the past
- show that we cannot leak data in the future



Questions?!?

info@scontain.com