Secure Production Identity Framework for Everyone（SPIFFE）與SPIFFE執行環境（SPIRE）介紹

Outline

Abbreviations

1. Introduction
2. SPIFFE
3. SPIRE
4. SPIRE環境建置
5. SPIFFE/SPIRE相關應用

Outline

Abbreviations

1. Introduction
2. SPIFFE
3. SPIRE
4. SPIRE環境建置

Abbreviations

英文縮寫	英文全名	中文
SPIFFE	Secure Production Identity Framework for Everyone	人人共享的安全生產身分框架
CA	Certificate Authority	憑證認證機構
JWT	JavaScript Object Notation Web Token	基於JSON Object的編碼
PKI	Public Key Infrastructure	公開金鑰基礎建設
CSR	Certificate Signing Request	憑證簽署要求
API	Application Programming Interface	應用程式介面
TLS	Transport Layer Security	傳輸層安全性協定
SSL	Secure Sockets Layer	安全通訊協定
VPN	Virtual Private Network	虛擬私有網路
IP	Internet Protocol	網際網路通訊協定
SPIFFE ID	SPIFFE Identity	SPIFFE身分
SVID	SPIFFE Verifiable Identity Document	SPIFFE可驗證身分文件

Abbreviations

英文縮寫	英文全名	中文
URI	Uniform Resource Identifier	統一資源識別符
SPIRE	SPIFFE Runtime Environment	SPIFFE執行環境
CLI	Command-Line Interface	命令列介面
mTLS	mutual Transport Layer Security	雙向傳輸層安全標準

Outline

Abbreviations

1. Introduction
2. SPIFFE
3. SPIRE
4. SPIRE環境建置

Introduction

背景

- 過去軟體大多部署在同一個資料中心或內網，彼此之間的信任是「以網路邊界為核心」（Perimeter-Based Security），只要進到網路內部就被默認為可信。
- 隨著雲端、多租戶環境、容器化與動態拓樸的普及，系統邊界幾乎消失。攻擊者只要滲透進一個節點，就能橫向移動（Lateral Movement）並造成巨大風險。因此零信任（Zero Trust）主張「不預設信任任何實體」，每個請求都必須驗證身分與權限。
- 人人共享的安全生產身分框架（ Secure Production Identity Framework for Everyone, SPIFFE）應用零信任架構（Zero Trust Architecture），在雲原生（Cloud Native）、動態、異質的軟體環境下，解決傳統邊界式安全與靜態憑證無法應付的挑戰。

Introduction

身分

- 現實中的身分證，由政府機構所頒發並且在頒發前會查驗申請人的信息，以確保身分證上的資訊是充分可信的；身分證本身做了防偽處理，有特定的查驗手段很難被偽造。
- 數位世界的身分證明（即數位憑證）與現實中的身分證明文件具有類似的概念。數位憑證可由可信的憑證認證機構（Certificate Authority, CA）驗證完申請者身分後頒發，其中憑證的頒發過程中CA使用自己的私鑰對憑證內容簽章，可確保憑證未被竄改，並能追溯到可信的CA。
- 常見的數位憑證有：X.509憑證、JSON Web Token（JWT）。

Introduction

公開金鑰基礎建設（Public Key Infrastructure, PKI）

- PKI是一套利用公鑰、私鑰的公開金鑰加密方式，來保障網路上資訊安全性的基礎流程架構。
- 公開金鑰加密技術包含兩個金鑰：
  - 公鑰：是一個可以公開分享的金鑰，常用於加密數據。
  - 私鑰：是一個必須由使用者自行保密的金鑰，常用於解密由公鑰加密的數據。
- 舉例而言，在加密通訊中，公鑰用於加密，私鑰用於解密；在數位簽章（Digital Signature）中，私鑰用於簽名，公鑰用於驗證簽名。
- 因為加密和解密使用的金鑰不同，因此被稱為「非對稱加密」。

Introduction

X.509憑證

- X.509是用於編碼及交換公鑰憑證的標準化格式，用來描述「一張憑證裡應該包含什麼欄位、怎麼結構化」。
- 用途：PKI中最廣泛使用的憑證格式。例如 ：傳輸層安全性協定（Transport Layer Security, TLS）/ 安全通訊協定（Secure Sockets Layer, SSL）、虛擬私有網路（Virtual Private Network, VPN），幾乎都採用X.509。
- X.509憑證頒發流程：申請者生成一對金鑰，並將公鑰與身分資訊封裝成憑證簽署要求（Certificate Signing Request, CSR）發送給CA，CA在驗證申請者身分成功後，會使用自己的私鑰簽發一張數位憑證。該憑證內包含申請者的公鑰及身分相關資訊，並附帶CA的數位簽章，用於保證其真實性與完整性。

Introduction

JSON Web Token（JWT）

- JWT是一種基於JSON格式的Token，通常用來「傳遞身分認證的數據」。
- 用途：可作為使用者登入後的身分憑證、應用程式介面（Application Programming Interface, API）呼叫的授權憑證（Authorization Certificate）、分散式系統（Distributed System）中服務間的身分認證。
- JWT由三個部分所組成，分別為Header、Payload、Signature，而且每個部分都會用「.」作分隔。
  - Header：存放通用資訊，例如：設定JWT的簽名算法、設定JWT的Token類型。
  - Payload：存放使用者的資料，使用者可以依據自己的需求，自由決定要放什麼資訊在裡面。
  - Signature：實作數位簽名，用來儲存簽名的計算結果。

Outline

Abbreviations

1. Introduction
2. **SPIFFE**
3. SPIRE
4. SPIRE環境建置

SPIFFE

Secure Production Identity Framework for Everyone（SPIFFE）簡介

- SPIFFE是一套開源標準規格，用來在動態且異質的環境中安全地識別軟體系統。採用SPIFFE的系統，無論運行在哪裡都能輕鬆且可靠地進行相互驗證。
- 像微服務（Microservices）、容器編排器（Container Orchestrator）和雲端運算（Cloud Computing）這些分散式的設計方式，讓系統的運行環境越來越多變、複雜。傳統的資安方法（例如：只允許特定網際網路通訊協定（Internet Protocol, IP）位址之間的流量）在這種情況下很難應付。為了因應這樣的挑戰，組織內部需要一套完善的工作負載（Workload）身分框架，來確保安全性與可擴展性。

SPIFFE

- SPIFFE建立一個框架，能夠在不同環境與組織邊界之間，啟動並簽發服務的身分。這些規範的核心，是透過簡單的API定義一種稱為SPIFFE可驗證身分文件（SPIFFE Verifiable Identity Document, SVID），讓Workload可以利用這些身分文件來與其他Workload進行驗證，例如：建立TLS連線，或是簽署與驗證JWT。

SPIFFE

SPIFFE概念

- SPIFFE 框架包含以下幾個部分：
  - 工作負載（Workload）
  - SPIFFE身分（SPIFFE Identity, SPIFFE ID）
  - 信任網域（Trust Domain）
  - SPIFFE可驗證身分文件（SPIFFE Verifiable Identity Document, SVID）
  - 工作負載API（Workload API）
  - 信任憑證集（Trust Bundle）

SPIFFE

工作負載（Workload）

- Workload指的是一個單一用途、以特定配置部署的軟體，它可能包含多個同時運行的實例，而這些實例都執行相同的任務。Workload這個詞可以涵蓋各種類型的軟體系統定義，包括：
  - 一個執行Python網頁應用程式的網頁伺服器，部署在一個虛擬機（Virtual Machine）叢集（Cluster）上，並在前端配置了負載平衡器（Load Balancer）。
  - 一個MySQL資料庫的實例。
  - 一個處理佇列（Queue）中項目的工作程式。
  - 一組獨立部署但能協同運作的系統，例如：一個使用資料庫服務的網頁應用程式。這個網頁應用程式和資料庫，也都可以各自被視為獨立的Workload 。

SPIFFE

- 在SPIFFE的定義中，Workload的粒度（Granularity）通常比實體或虛擬節點更細，甚至細到節點上的單一程序（Process）。尤其是在容器編排器環境中，多個Workload可能同時存在於同一個節點上，但彼此之間仍保持隔離。
- Workload也可以橫跨多個節點。舉例來說，一個具備彈性擴展能力的網頁伺服器，可能會同時在多台機器上運行。
- 雖然Workload的粒度會依情境不同而有所差異，但是在SPIFFE的定義中，假設Workload之間有足夠的隔離性，使得惡意的Workload無法在憑證簽發後竊取其他Workload的憑證。

SPIFFE

SPIFFE身分（SPIFFE Identity, SPIFFE ID）

- SPIFFE ID是一字串編號，用來識別Workload，此編號具有唯一性與明確性。此外，SPIFFE ID也可以指派給執行Workload的平台或環境（例如：虛擬機器）。
- SPIFFE ID是一種類似於統一資源識別符（Uniform Resource Identifier, URI）的格式，其結構為：
  - spiffe://trust domain/workload identifier
- 上述SPIFFE ID的結構中，workload identifier是用來在一個信任網域（Trust Domain）內，唯一地辨識特定的Workload 。

SPIFFE

信任網域（Trust Domain）

- 在SPIFFE中，Trust Domain代表了一個可被信任的身分命名空間（Identity Namespace），也就是「一群Workload可彼此信任的範圍」。例如：一個Trust Domain可以代表執行自己獨立SPIFFE基礎架構的個人、組織、環境或部門。
- Trust Domain是由一個具備加密金鑰集合的CA所支援，並且CA掌管一組根金鑰（Root Keys）用來簽發、驗證與更新憑證（SVID），也就是說此CA為Trust Domain的信任根（Root of Trust）。
- 所有屬於同一Trust Domain的Workload，所獲得的身分文件（SVID）都能透過該Trust Domain的Root Keys進行驗證。
- 一般建議，若Workload位於不同的實體位置（例如：不同的資料中心或雲端區域），或是在採用不同資安策略的環境中（例如：測試或實驗環境與正式環境相比），應該將它們分屬於不同的Trust Domain。

SPIFFE

根金鑰（Root Keys）

- Root Keys指的是一組與Trust Domain綁定的金鑰對（公鑰 / 私鑰）。
- Root Keys是SPIFFE身分驗證的核心信任基礎，所有Workload的憑證（SVID）最終都要能追溯到Root Keys。在SPIFFE執行環境（SPIFFE Runtime Environment, SPIRE）裡，Root Keys通常由SPIRE Server持有與管理。
- 在SPIFFE裡，Root Keys主要有兩個功能：
    - 私鑰：透過間接的方式簽發憑證（SVID）。在SPIRE架構中，SPIRE Server使用Root Keys簽署中繼CA（Signing CA），而由Signing CA進一步簽發Leaf憑證（Workload的X.509-SVID）。當Workload間進行身分驗證時，系統會透過SVID所附的憑證鏈（Certificate Chain）驗證簽章，並比對其最上層CA是否能由Trust Bundle中的Root CA憑證驗證成功；若驗證通過，則表示該SVID來自可信任的 Trust Domain。

SPIFFE

- 公鑰：用來驗證身分文件（SVID）。Workload可以從由SPIRE Server發布與維護的Trust Bundle中取得Root Keys的公鑰。當一個Workload收到另一個Workload的SVID時，就用這把公鑰來驗證「這份SVID確實是由可信任的Trust Domain簽發的」。

- Root Keys的「私鑰」的所有權與職責：
  - 在一個Trust Domain裡，Root Keys的私鑰是最高機密，不能被SPIRE Agent或Workload所擁有。SPIRE Server是唯一持有Root Keys的私鑰的角色。
  - SPIRE Server使用Root Keys的私鑰來簽發SVID（X.509-SVID或JWT-SVID），以及保證整個Trust Domain的信任鏈（Trust Chain）完整。

- Root Keys的「公鑰」的所有權與職責：
  - SPIRE Server：由於Root Keys是由SPIRE Server所管理，所以SPIRE Server一定擁有公鑰。
  - SPIRE Agent：SPIRE Agent會從SPIRE Server同步信任憑證集（Trust Bundle），Trust Bundle內包含公鑰。
  - 所有Workload：Workload透過Workload API拿到Trust Bundle，Trust Bundle內包含公鑰。
  - 用途：
    - 驗證SVID的簽章，確認SVID是否由該Trust Domain所簽發。
    - 驗證X.509-SVID或JWT-SVID是否真實可信。

SPIFFE

信任網域（Trust Domain）的信任鏈（Trust Chain）

- Trust Chain是一條由多層金鑰與憑證構成的階層式信任結構。在SPIFFE中，每個Trust Domain都有自己的Trust Chain。Trust Chain是以Root Keys的私鑰→Root CA憑證→Signing CA→SVID的方式層層衍生而成。下表為完整Trust Chain的層次：

層級	名稱	持有者	功能	說明
1	Root Keys的私鑰	SPIRE Server	最上層私鑰	整個Trust Domain的信任根；用來簽發Root CA憑證
2	Root CA憑證	SPIRE Server	對外公鑰	屬於自簽憑證；包含Root Keys的公鑰
3	Signing CA（中繼CA）	SPIRE Server	下層簽發者	由Root CA簽出；實際簽發Workload SVID
4	Workload SVID（Leaf Certificate）	Workload	憑證持有者	代表Workload的身分；由Signing CA簽出

SPIFFE

SPIFFE可驗證身分文件（SPIFFE Verifiable Identity Document, SVID）

- SVID是SPIFFE定義的「可驗證身分文件」。
- Workload使用SVID來向資源或呼叫方證明自身身分。若一份SVID是由該SPIFFE ID所屬Trust Domain中的CA簽署，就會被視為有效。
- SVID內含一個SPIFFE ID，用來表示提交該文件的服務身分。它會將SPIFFE ID編碼在一份可加密驗證的文件中。SVID目前支援兩種格式：X.509-SVID或JWT-SVID。
- 由於Token容易受到重播攻擊（Replay Attack）的影響，攻擊者若在傳輸過程中取得Token，就可能用它來冒充某個Workload，因此建議在可能的情況下，優先使用X.509-SVID。
- 然而，在某些情境下可能只能使用JWT-SVID，例如：當系統架構中，兩個Workload之間存在應用層代理伺服器（L7 Proxy）或負載平衡器時。

SPIFFE

- SVID的內容如下：
  - SPIFFE ID（身分）
  - 公鑰（對應私鑰）
  - 有效期（短期）
  - 簽章（由Root Keys的私鑰以間接的方式來簽發）
- 用途：
  - Workload用來向其他Workload或服務證明「自己的身分」。

SPIFFE

工作負載API（Workload API）

- Workload API是Workload跟SPIRE Agent溝通的介面。
- Workload透過Workload API向本地的SPIRE Agent請求自己的SVID，並同時取得Workload之間用於驗證對方身分的信任憑證集（Trust Bundle）。
- Workload API支援以下兩種格式的SVID：X.509-SVID、JWT-SVID。

SPIFFE

- 對於X.509格式的身分文件（X.509-SVID）：
  - X.509-SVID的身分，以SPIFFE ID來描述。
  - 一把與該身分綁定的私鑰，可用來代表Workload簽署資料。同時，系統會建立一張對應的短期X.509憑證（X.509-SVID），可用來建立TLS連線或用於與其他Workload進行身分驗證。
  - 一組信任憑證集（Trust Bundle）。Workload可以用Trust Bundle來驗證其他Workload所提交的X.509-SVID是否可信。

SPIFFE

- 對於JWT格式的身分文件（JWT-SVID）：
  - JWT-SVID的身分，以 SPIFFE ID 表示。
  - 採用JWT。
  - 一組信任憑證集（Trust Bundle）。Workload可以用Trust Bundle來驗證其他Workload所提交的JWT-SVID是否可信。
- 就像AWS EC2 Instance Metadata API和Google GCE Instance Metadata API一樣，Workload API並不要求呼叫的Workload需要知道自己的身分，或是在呼叫API時持有任何驗證用的Token。這代表使用者的應用程式在部署時，不需要和Workload一起放置任何驗證憑證或密鑰。

SPIFFE

- Workload API與系統平台無關，它能夠在程序層級甚至核心（Kernel）層級辨識正在運行的服務，這使得它特別適合用於像Kubernetes這樣的容器編排系統。
- 為了降低金鑰外洩或遭到入侵的風險，所有私鑰（以及對應的憑證）都具備短期效期，並會自動且頻繁地進行金鑰輪替（Key Rotation）。Workload可以在相關金鑰到期前，透過SPIRE Agent的Workload API向SPIRE Server，請求新的SVID與Trust Bundle。

SPIFFE

信任憑證集（Trust Bundle）

- Trust Bundle是一組CA憑證集合（Certificates of Authorities），代表一個Trust Domain的信任根。而經由Root Keys的私鑰簽發出Root CA憑證（含公鑰）後，該憑證會被加入到Trust Bundle。
- 在SPIRE中，Trust Bundle由SPIRE Server維護，並透過SPIRE Agent發佈給Workload。Workload則使用Trust Bundle來驗證其他Workload的SVID是否可信。

SPIFFE

- 以下為Trust Bundle的主要用途:

1. 驗證身分：

   A. 當Workload A拿到Workload B的SVID時，Workload A會使用Trust Bundle裡的公鑰來檢查：這份SVID是不是由可信任的Trust Domain簽發的？簽名是否正確、資料是否未被竄改？

2. 建立跨域信任：

   A. 在多個Trust Domain的情境下，Trust Bundle可能包含多個CA憑證，可實現讓不同Trust Domain的Workload彼此驗證，達到跨Domain信任。

3. 支援金鑰輪替：

   A. 因為CA金鑰（私鑰與公鑰）會定期輪替，對應的CA憑證（內含公鑰）也會隨之更新，新舊根憑證可能在一段時間內同時存在。

   B. Trust Bundle的內容會因CA憑證更新也跟著經常更新，確保Workload永遠擁有最新且完整的信任根。

SPIFFE

- 以下為Trust Bundle分別以X.509-SVID與JWT-SVID的驗證做舉例:
  - X.509-SVID驗證：
    - Trust Bundle裡存放Root CA憑證的公鑰。
    - 當Workload A送出一張X.509-SVID給Workload B時，Workload B會使用Trust Bundle裡的Root CA憑證（內含公鑰）驗證這張憑證中，確認簽名是否正確以及這張憑證是否屬於可信任的 Trust Domain。

SPIFFE

- JWT-SVID驗證：
  - Trust Bundle裡存放JWT簽署者的公鑰。
  - 當Workload B（驗證方）收到Workload A（被驗證方）的JWT-SVID後，Workload B會使用Trust Bundle裡的JWT簽署者公鑰驗簽，確認這個JWT-SVID是由可信任的Trust Domain簽署的，以及確認Token沒有被竄改。

Outline

Abbreviations

1. Introduction
2. SPIFFE
3. **SPIRE**
4. SPIRE環境建置

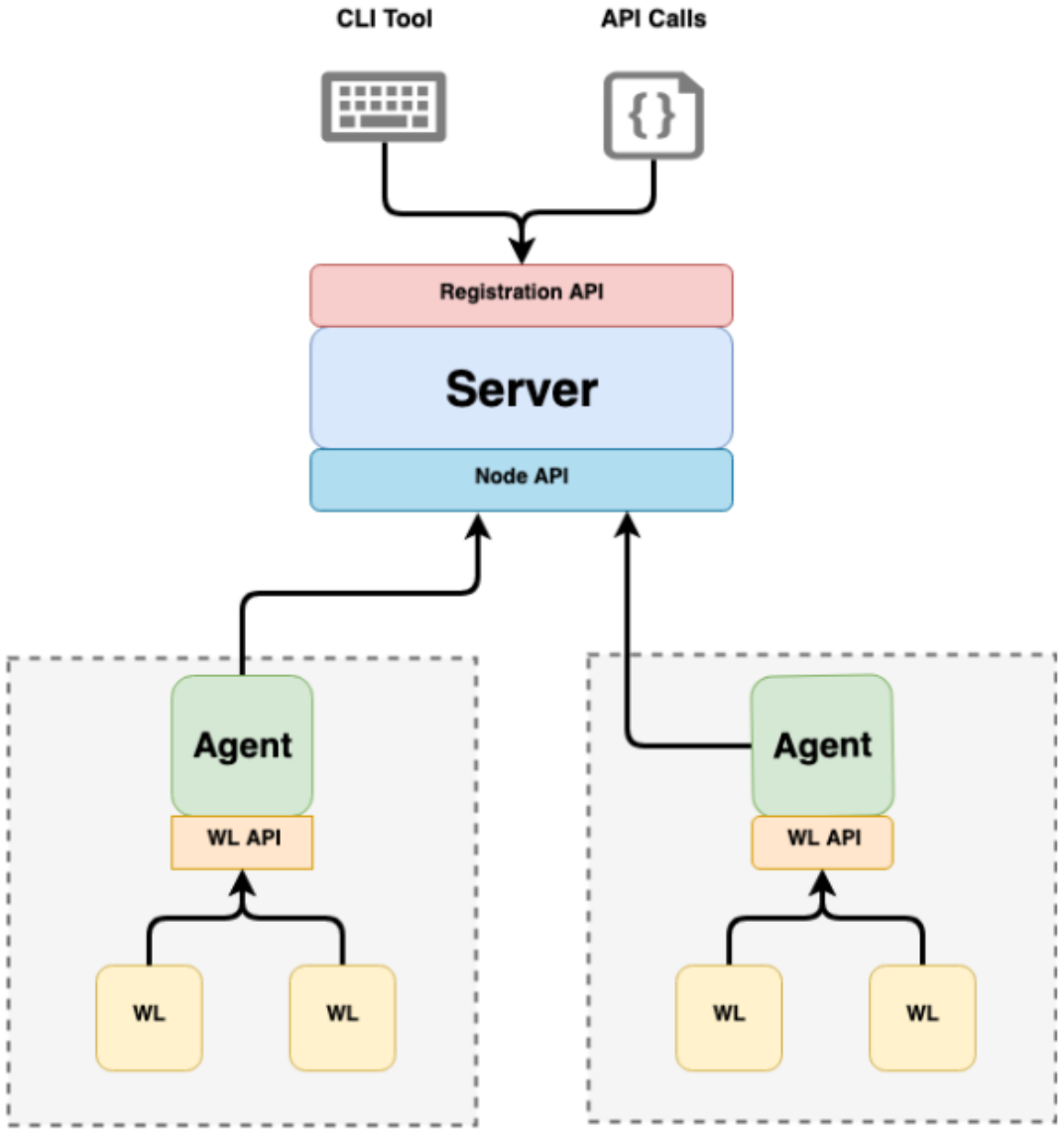
SPIRE

SPIFFE Runtime Environment（SPIRE）的基礎概念

- SPIRE是SPIFFE APIs的開源軟體實現，它執行節點和工作負載證明（Workload Attestation），以便根據預先定義的一組條件安全地向Workload發出SVID，並驗證其他Workload的SVID。
- SPIRE部署由一個SPIRE伺服器（Server）和一個或多個SPIRE代理程式（Agent）組成。
- SPIRE Server為透過SPIRE Agent向一組Workload頒發身分的簽章機構。SPIRE Server也維護Workload身分的註冊表（Registration），以及頒發這些身分必須驗證的條件。
- SPIRE Agent在本機向Workload提供Workload API給Workload，並且必須安裝在執行Workload的每個節點上。

SPIRE

SPIRE架構圖



SPIRE

SPIRE架構的組成元件

- Server：
  - 為整個系統的「信任根」，說明如下：
    - Server持有Root Keys（包含私鑰）與Root CA憑證：Root CA憑證是整個Trust Domain的最高層憑證，所有Signing CA（中繼憑證）與Workload的SVID都最終由Root Keys的私鑰所簽出或衍生。
    - Server簽發與管理SVID：Server根據節點驗證（Node Attestation）的結果，確認對方節點可信後，簽發對應的SVID給Agent與Workload。Server負責管理和頒發其配置的SPIFFE Trust Domain的所有身分
    - Server維護Trust Bundle：Server會發布包含Root CA憑證的Trust Bundle，所有Agent與Workload透過這份Trust Bundle來驗證他人身分。

SPIRE

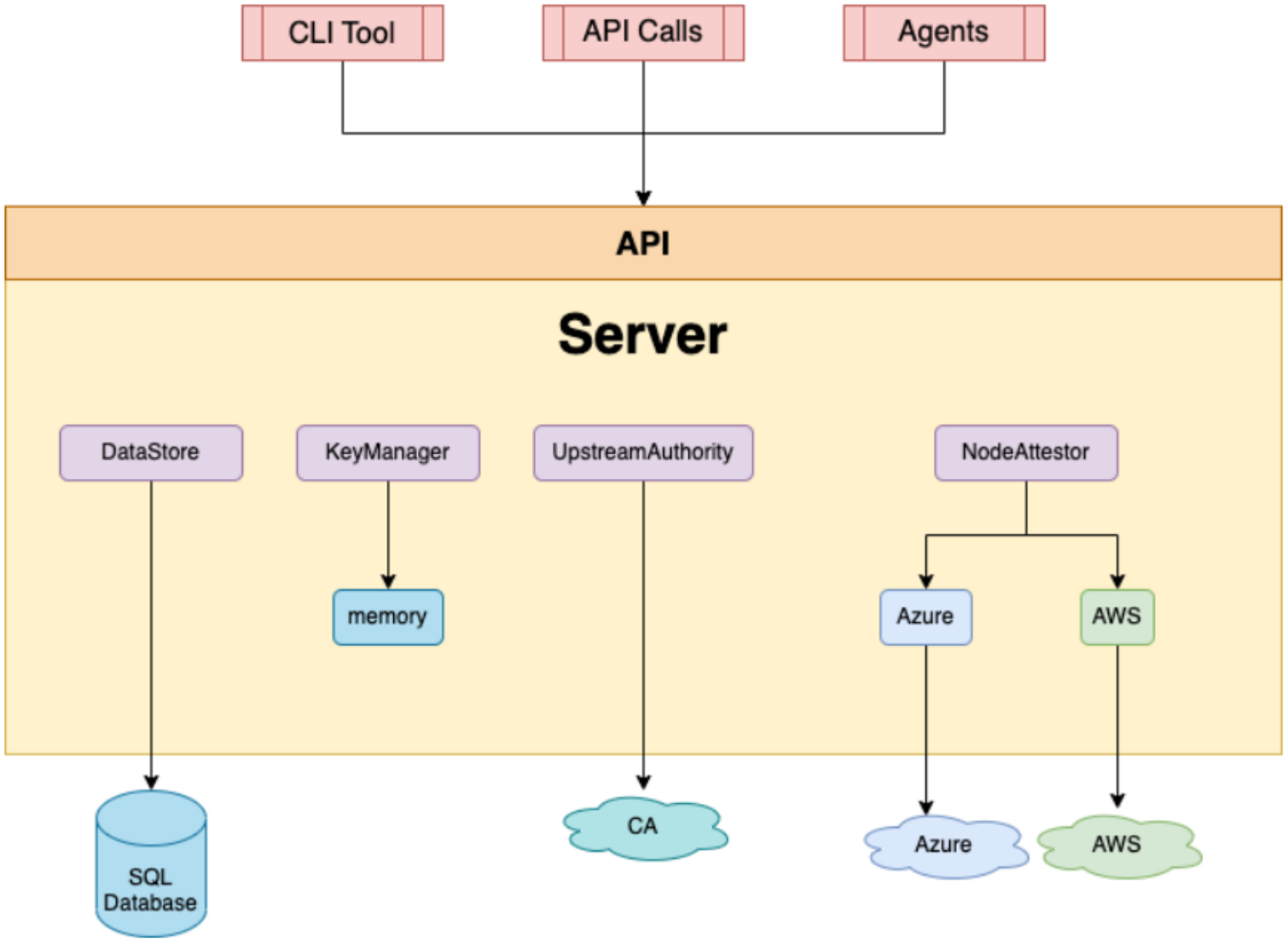
- Server提供兩個主要的API：
  - Registration API：讓管理者透過命令列介面（Command-Line Interface, CLI）或API Calls去註冊Workload的身分。
  - Node API：提供給Agent，用來驗證節點（Node Attestation）並讓Agent取得節點憑證（Node SVID）。
- Server儲存註冊條目（Registration Entries），Registration Entries用於指定選擇器（Selectors），這些Selectors決定了在何種條件下應頒發特定的SPIFFE ID和簽名金鑰。

SPIRE

- Agent：運行在每個執行已識別Workload的節點上。
  - 功能是與Server溝通，透過Node API取得SVID，並快取起來，直到有Workload需要使用其SVID。
  - Agent在節點上提供Workload API，並驗證呼叫該API的Workload身分，讓同一節點上的Workload能安全地取得對應的身分。
- Workload：
  - 即實際在節點上執行的應用程式或服務。
  - 不直接與Server互動，而是透過Agent來安全取得SVID。

SPIRE

SPIRE Server



SPIRE

SPIRE Server外掛（SPIRE Server Plugins）

- Server的行為是由一系列Plugins所決定的。SPIRE本身已內建多種Plugins，但也可以依照特定需求開發額外的Plugins，來擴充SPIRE的功能。
- 常見的Plugins類型包括：
  - 資料儲存外掛（DataStore Plugins）：儲存、查詢和更新各種資訊， 例如：Registration Entries、已通過驗證的節點以及這些節點所對應的Selectors。SPIRE內建一個DataStore Plugins，可支援MySQL、SQLite 3或PostgreSQL資料庫來儲存必要的資料。預設情況下會使用SQLite 3。
  - 金鑰管理外掛（Key Manager Plugins）：這類Plugins負責控制Server如何儲存用來簽署X.509-SVID與JWT-SVID的私鑰。

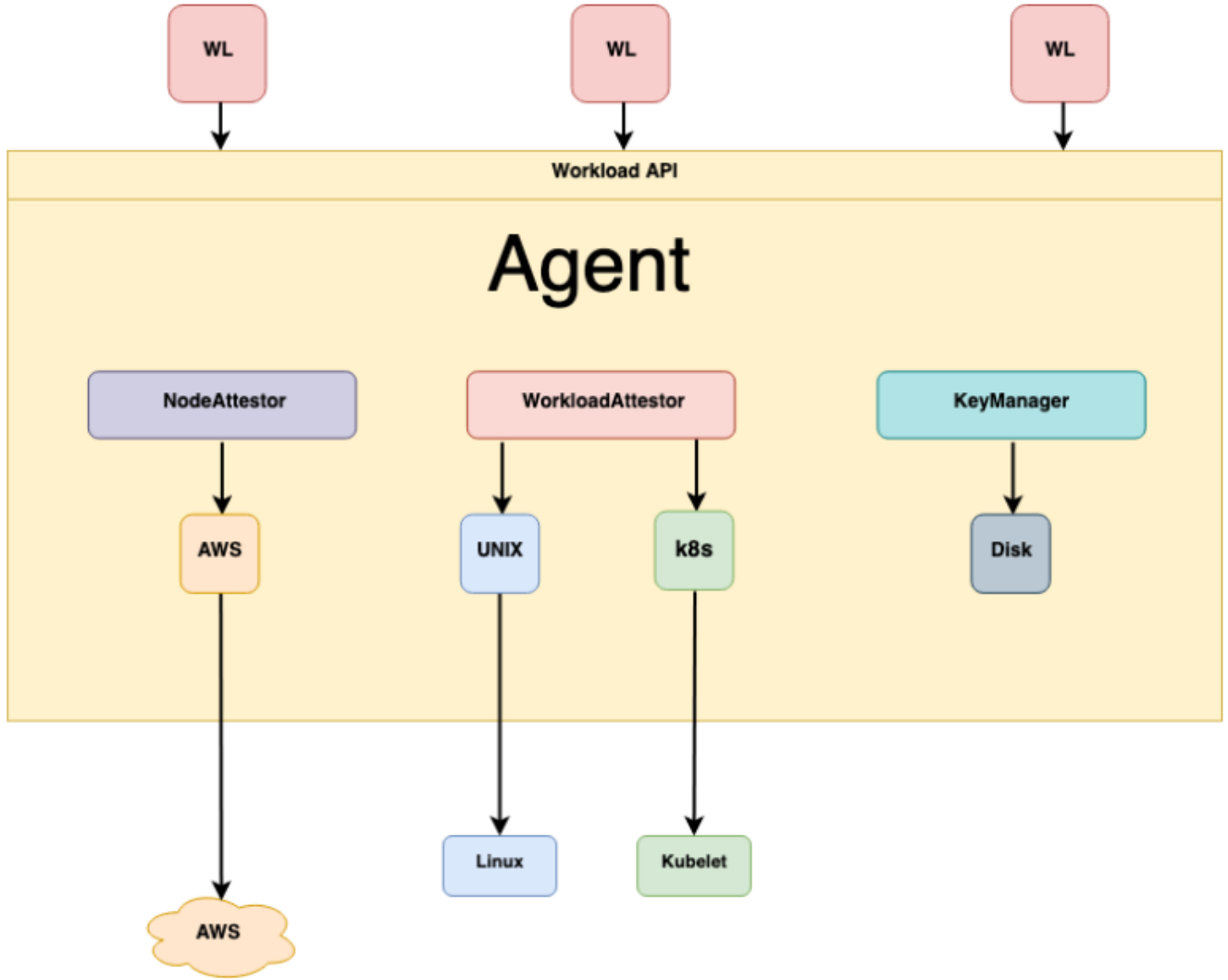
SPIRE

- 上游授權外掛（Upstream Authority Plugins）：預設情況下，Server會充當自己的CA。然而，若有需要也可以透過Upstream Authority Plugins，改用來自其他PKI系統的不同CA。
- 節點驗證外掛（Node Attestor Plugins）：與Agent的Node Attestor Plugins搭配，負責驗證Agent所運行節點的身分。

- 使用者可以透過設定Plugins與其他各種組態參數，來自訂SPIRE Server的行為。

SPIRE

SPIRE Agent



SPIRE

SPIRE Agent的主要組成元件

- 節點驗證外掛（Node Attestor Plugins）：與Server的Node Attestor Plugins搭配，用來驗證Agent所運行節點的身分。在節點上對Workload提供SPIFFE Workload API，並驗證呼叫該API的Workload身分。
- 工作負載驗證外掛（Workload Attestor Plugins）：透過向節點作業系統查詢Process資訊，並與使用者在註冊Workload屬性時提供給Server的Selectors進行比對，來驗證節點上Workload Process的身分。
- 金鑰管理外掛（Key Manager Plugins）：Agent使用Key Manager Plugins來產生與管理私鑰，並將這些私鑰用於簽發給Workload的X.509-SVID。

SPIRE

自訂Server與Agent外掛

- 使用者可以為特定平台與架構建立自訂的Server Plugins與Agent Plugins，以補足SPIRE尚未內建支援的部分。
- 例如：
  - 使用者可以為某些未列入節點驗證（Node Attestation）的架構，建立專屬的Server與Agent的Node Attestor Plugins。
  - 使用者也可以建立自訂的Key Manager Plugins，用來處理SPIRE目前尚未支援的私鑰管理方式。
- 由於SPIRE會在執行階段載入自訂Plugins，因此不需要重新編譯SPIRE就能啟用這些Plugins。

SPIRE

工作負載註冊（Workload Registration）

- 為了讓SPIRE能識別某個Workload，必須透過Registration Entries，在SPIRE Server上註冊該Workload。
- Workload Registration的目的，是告訴SPIRE如何識別這個Workload，以及應該分配給它哪一個SPIFFE ID。
- Registration Entries會將一個身分（以SPIFFE ID表示）對應到一組稱為Selectors的屬性。Workload必須具備這些屬性，才能被簽發對應的身分。
- 在Workload Attestation的過程中，Agent會利用這些Selectors的值來驗證Workload的身分。

SPIRE

驗證（Attestation）

- 在SPIRE的架構中，Attestation是指確認一個Workload的身分。SPIRE透過收集Workload本身的執行屬性，以及Agent所運行節點的屬性，並從可信任的第三方來源取得資料，將這些屬性與Workload註冊時所定義的一組Selectors進行比對，以此方式完成身分驗證。
- SPIRE為了執行驗證而查詢的第三方可信來源，會依不同平台而有所差異。
- SPIRE的Attestation分為兩個階段：
  - 節點驗證（Node Attestation）：確認執行Workload的節點身分。
  - 工作負載驗證（Workload Attestation）：在節點上確認特定Workload的身分。

SPIRE

- SPIRE採用具高度彈性的架構，可根據Workload所處的環境，使用多種不同的第三方來源進行節點與Workload的Attestation。
- 使用者可透過設定SPIRE的Agent與Server組態檔，指定要使用哪些可信任的第三方來源，以及在註冊Workload時，透過設定Selector值」來指定要使用哪些類型的屬性進行Attestation。

SPIRE

節點驗證（Node Attestation）

- SPIRE規定每個Agent在首次連線到Server時，必須進行自我身分驗證與確認，這個過程稱為Node Attestation。
- 在Node Attestation過程中，Agent與Server會共同驗證Agent所執行節點的身分，並透過Node Attestor Plugins的機制來完成這件事。所有Node Attestor都會檢查節點及其環境中只有該節點才擁有的的特定資訊，以此來證明節點的真實身分。
- 成功完成Node Attestation後，Agent會獲得一個唯一的SPIFFE ID。這個Agent的SPIFFE ID之後會作為其所管理Workload的「父身分」（父SPIFFE ID）。

SPIRE

- Node Attestors會回傳一組可選的Selectors給Server，用來識別特定的機器（例如：Amazon Instance ID）。這些由Node Attestors提供的Selectors，會成為與該Agent節點的SPIFFE ID所關聯的一組Selectors。
- 在進行Node Attestation時，Selectors並非必須，除非你需要將Workload對應到多個節點。

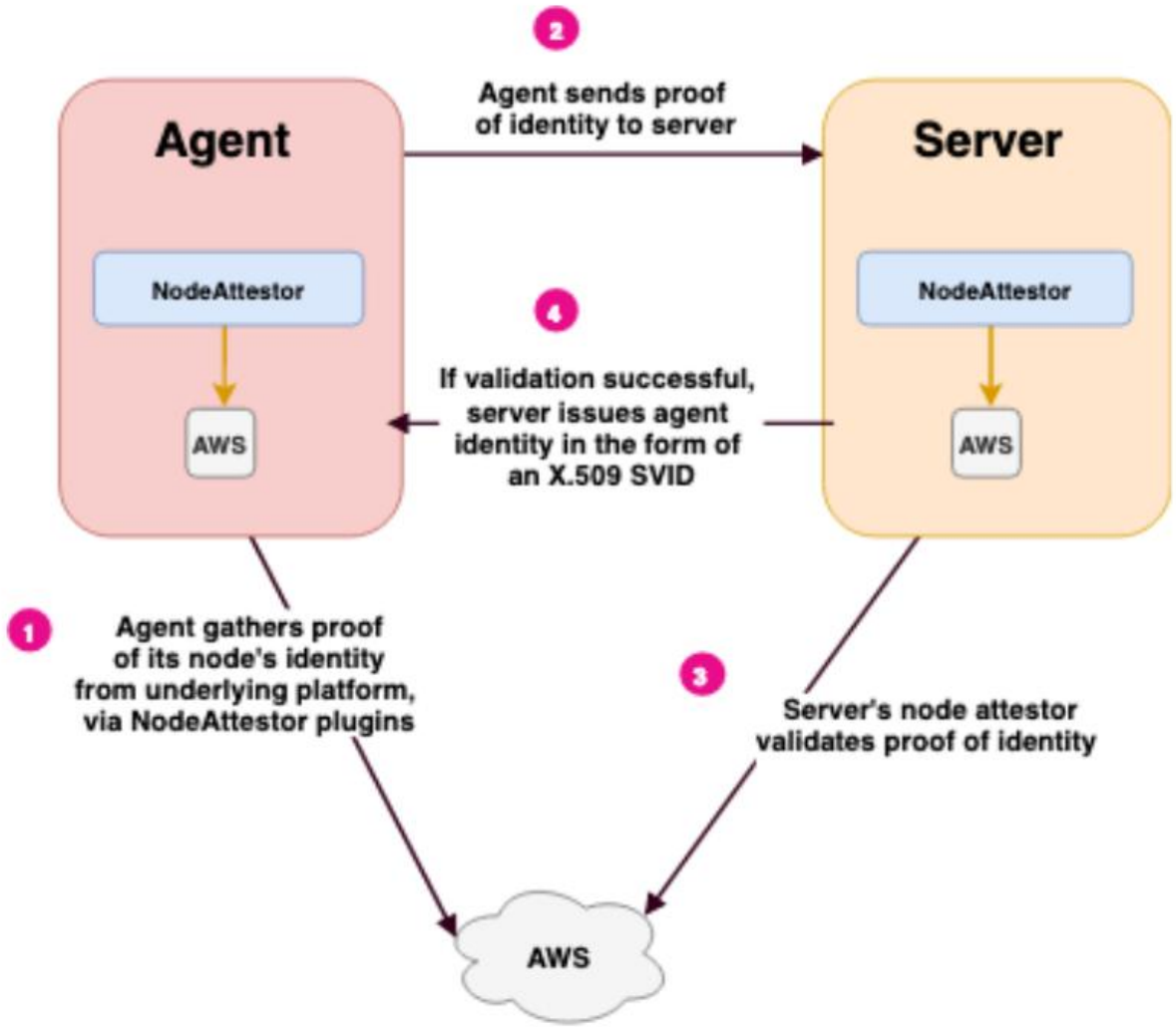
SPIRE

- 節點身分驗證的證明範例包括：
  - 由雲端平台提供給節點的身分文件（例如：AWS Instance Identity Document）。
  - 驗證儲存在節點上的硬體安全模組（Hardware Security Module, HSM）或可信平台模組（Trusted Platform Module, TPM）中的私鑰。
  - 在安裝Agent時，透過Join Token進行的人工驗證。
  - 由多節點軟體系統在安裝至節點時配置的識別憑證（例如：Kubernetes Service Account Token）
  - 其他機器身分的證明（例如：已部署的Server憑證）

SPIRE

節點驗證（Node Attestation）步驟

- 在這個範例中，底層平台是 AWS：



SPIRE

節點驗證（Node Attestation）步驟

1. Agent的AWS Node Attestor Plugins會向AWS查詢節點身分的證明，並將取得的資訊提供給Agent。
2. Agent將這份身分證明傳送給Server，而Server再將這些資料交給自己的AWS Node Attestor，進行Server端驗證。
3. Server會拿著Agent傳來的證明資料，呼叫AWS API來驗證身分證明。驗證完成後，Server為該Agent產生一個 SPIFFE ID（Agent 身分），並根據節點屬性生成對應的Selectors，然後再把這些資訊登錄回 Server的資料庫。

4. Server隨後會回傳一個SVID給該Agent節點，代表這個節點已被信任，往後這個Agent就能代表節點上的各個Workload，再去跟Server申請Workload SVID。

SPIRE

節點驗證外掛（Node Attestor Plugins）：

- Agent與Server都會透過各自的Node Attestor Plugins來檢查底層平台。
- SPIRE支援在多種環境中進行節點身分驗證，包括：
  - AWS的EC2實例（使用EC2 Instance Identity Document）
  - Microsoft Azure的虛擬機（使用Azure Managed Service Identities）
  - Google Cloud Platform的GCE實例（使用GCE Instance Identity Tokens）
  - 屬於Kubernetes叢集的節點（使用Kubernetes Service Account Tokens）

SPIRE

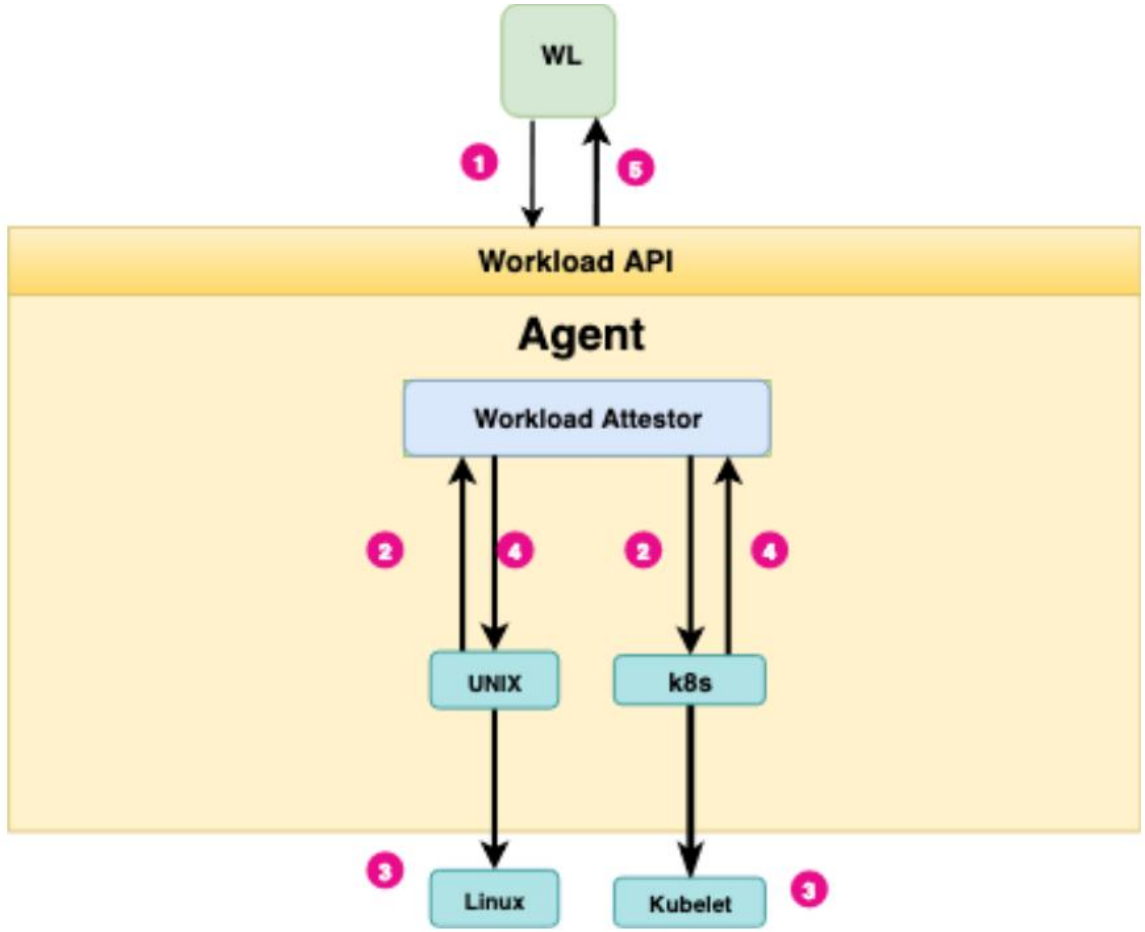
- 若平台本身無法直接辨識節點，SPIRE也提供其他Node Attestation方式，例如：
  - 使用Server產生的Join Token：
    - Join Token是SPIRE Server與SPIRE Agent之間共享的預先金鑰。Server安裝後可以產生這些Token，Agent在啟動時可用來驗證自身。為了避免被濫用，Join Token在使用後會立即失效。
  - 使用現有的X.509憑證。

工作負載驗證（Workload Attestation）

- Workload Attestation要解決的核心問題是：「這個程序是誰？」。
- Agent會透過查詢本地可用的權威來源（例如：節點的作業系統核心），來判斷呼叫Workload API的Process屬性，並據此識別該Workload。
- 接著，這些屬性會與你在註冊Workload時，透過Selectors提供給Server的資訊進行比對。

- 這類資訊可能包括：
  - 作業系統層級的排程方式：在類Unix系統中，可能是透過使用者ID（UID）、群組ID（GID）、檔案系統路徑等。
  - 由容器編排系統（例如：Kubernetes）所進行的排程方式：在這種情況下，Workload可能會透過其所使用的Kubernetes Service Account或Namespace來描述。
- 雖然Agent與Server都會參與Node Attestation，但在Workload Attestation中只有Agent會參與其中。

工作負載驗證（Workload Attestation）的步驟



SPIRE

工作負載驗證（Workload Attestation）流程

1. Workload透過Workload API請求一個SVID。在Unix系統中，這會透過Unix Domain Socket提供。
2. Agent會檢查節點的核心，以識別呼叫者的Process ID。接著，它會呼叫已設定的Workload Attestor Plugins，並將該Workload的Process ID傳給Workload Attestor Plugins。
3. Workload Attestors使用Process ID來發現有關Workload的其他信息，並根據需要查詢鄰近的平台特定元件。通常，這些元件也與Agent位於同一節點上。
4. Workload Attestors會將收集到的資訊，以Selectors的形式回傳給Agent。
5. Agent會將取得的Selectors與Registration Entries進行比對，以判斷Workload的身分，並將正確的快取SVID回傳給該Workload。

SPIRE

SPIRE如何為Workload簽發SVID

- 從Agent在節點上啟動，到同一節點上的Workload取得一份有效的身分（X.509 SVID）為止。
- 需要注意的是：若使用的是JWT格式的SVID，處理方式會有所不同。在這個簡化的示範中，工作負載運行於AWS EC2上。流程如下：

1. SPIRE Server啟動。
2. 若使用者未設定Upstream Authority Plugins，Server會產生一張自簽憑證（Self-Signed Certificate），由Server自己的私鑰進行簽署。之後，Server會使用這張憑證，為屬於該Server Trust Domain內的所有Workload簽發SVID。

SPIRE

3. 如果是第一次啟動，Server會自動產生一個Trust Bundle，並將其內容儲存在使用者於Data Store Plugins中指定的SQL資料庫 內。
4. Server啟用Registration API，讓使用者可以註冊Workload。
5. SPIRE Agent在執行Workload的節點上啟動。
6. Agent會執行Node Attestation，向Server證明它所運行節點的身分。例如：當運行在AWS EC2實例上時，通常會透過提供AWS Instance Identity Document給Server來完成Node Attestation。
7. Agent會透過TLS連線，將這份身分證明提交給Server，而這條連線的驗證則是依靠Agent啟動時設定的Bootstrap Bundle。

8. Server會呼叫AWS API來驗證這份身分證明。
9. AWS確認該文件有效。
10. Server會執行額外的驗證步驟，以確認Agent節點的更多屬性，並相應地更新其Registration Entries。例如：如果節點是透過AWS Instance Identity Document (IID) 驗證的，Node Attestor Plugins會再呼叫AWS API，以取得更多資訊，來建立額外的一組Selectors，例如自動擴展群組（Auto Scale Group）或實例標籤（Instance Tag）等資訊。
11. Server會簽發一個SVID給Agent，用來代表該Agent自身的身分。
12. Agent使用自己的SVID作為TLS用戶端憑證，與Server建立連線，並向Server取得它被授權的Registration Entries。

13. Server會透過Agent的SVID驗證Agent的身分。接著，Agent也會完成雙向傳輸層安全標準（mutual Transport Layer Security, mTLS）握手，並利用Bootstrap Bundle驗證Server的身分。
14. Server隨後會從資料庫中取出所有授權的Registration Entries，並傳送給Agent。
15. Agent接著會將Workload的CSR傳送給Server。Server會對這些請求進行簽署，並將簽發完成的Workload SVID回傳給Agent。Agent則會將這些SVID存放在快取中。
16. 完成所有啟動程序後，Agent開始在Workload API Socket上進行監聽。
17. 某個Workload呼叫Workload API，請求一份SVID。
18. Agent會啟動Workload Attestation程序，呼叫其Workload Attestors，並將該Workload的Process ID提供給它們。

SPIRE

19. Workload Attestor會透過Kernel與User space的呼叫，來收集關於該Workload的更多資訊。
20. Workload Attestors會將收集到的資訊，以Selectors的形式回傳給Agent。
21. Agent會將取得的Selectors與Registration Entries進行比對，以判斷該Workload的身分，並回傳正確的 SVID（已存在快取中）。

授權的註冊條目（Authorized Registration Entries）

- Server只會將Authorized Registration Entries傳送給Agent。為了取得這些Authorized Registration Entries，Server會執行以下步驟：

1. 查詢資料庫中，Workload的父SPIFFE ID為該Agent的SPIFFE ID的Registration Entries。
2. 查詢資料庫，取得與該Agent關聯的額外屬性（Selectors）。
3. 查詢資料庫中，凡是宣告至少符合其中一個Selectors的Registration Entries。
4. 遞迴地在資料庫中查詢所有以這些條目為「父SPIFFE ID」的註冊項目，並持續向下搜尋所有層級的子項。

- Server會將最終取得的這些Authorized Registration Entries傳送給Agent。

Outline

Abbreviations

1. Introduction
2. SPIFFE
3. SPIRE
4. **SPIRE環境建置**

SPIRE環境建置

系統環境與參考資料

- 系統版本：ubuntu 22.04 LTS
- Docker版本：28.3.3, build 980b856
- 參考資料：
  - https://spiffe.io/docs/latest/try/getting-started-k8s/
  - https://spiffe.io/docs/latest/microservices/envoy-x509/readme/

SPIRE環境建置

安裝SPIFFE的相關套件

- 安裝Ubuntu官方套件庫 (APT Repository) 中的docker.io套件：
```
$ sudo apt install docker.io
```
- 安裝ca-certificates（HTTPS憑證）、curl（下載工具）與gnupg（GNU Privacy Guard）：
```
$ sudo apt-get install -y ca-certificates curl gnupg
```
- 建立一個/etc/apt/keyrings目錄，並設定權限0755，用來存放Apt Repository的金鑰檔案：
```
$ sudo install -m 0755 -d /etc/apt/keyrings
```
- 從Docker官方下載GPG公鑰，轉換成APT可用的二進位格式，並存放到指定位置：
```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --
dearmor -o /etc/apt/keyrings/docker.gpg
```

SPIRE環境建置

- 建立Docker官方的軟體來源設定檔並存到/etc/apt/sources.list.d/docker.list裡。之後Ubuntu的Apt系統就知道要去哪個Server下載Docker：

```
$ echo “deb [arch=$(dpkg –print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu $(. /etc/os-release; echo “$VERSION_CODENAME”) stable” | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

- 更新APT套件索引，讓系統知道「現在多了一個新的來源（Docker官方），去抓一下那邊的套件清單。」：

```
$ sudo apt-get update
```

- 正式把Docker Engine、CLI、Containerd、Buildx與Compose全部安裝進使用者的系統，讓使用者可以立即使用Docker建立與管理容器環境：

```
$ sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

SPIRE環境建置

- 啟動Docker服務並設為開機自啟：
  ```
  $ sudo systemctl enable --now docker
  ```
- 驗證Docker Buildx外掛是否安裝成功：
  ```
  $ docker buildx version
  ```

SPIRE環境建置

下載kubectl與Minikube

- 安裝與驗證kubectl。kubectl是用來與Kubernetes叢集（cluster）溝通的主要CLI：
  - 從Kubernetes官方網站下載最新穩定版的kubectl：

```
$ curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
```

  - 安裝kubectl到系統執行路徑，設定正確權限：

```
$ sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

  - 驗證kubectl是否安裝成功與可執行，只顯示客戶端版本（不需要連線叢集）：

```
$ kubectl version --client
```

  - 執行完應該要出現版本資訊：

```
ubuntu@spiffe:~$ kubectl version --client
Client Version: v1.33.3
Kustomize Version: v5.6.0
```

SPIRE環境建置

- 安裝Minikube，Minikube能夠在macOS、Linux與Windows快速建立起單節點的Kubernetes Cluster。Kubernetes Cluster是一組協同運作的伺服器（Nodes），用來部署、管理與運行容器化應用程式的環境：
  - 下載最新版Minikube：

```
$ curl -LO
https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
```

  - 安裝Minikube到系統執行路徑：

```
$ sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

SPIRE環境建置

- 啟動Minikube，建立本地Kubernetes Cluster：

```
$ minikube start \
    --driver=docker \
    --extra-config=apiserver.service-account-signing-key-
file=/var/lib/minikube/certs/sa.key \
    --extra-config=apiserver.service-account-key-
file=/var/lib/minikube/certs/sa.pub \
    --extra-config=apiserver.service-account-issuer=api \
    --extra-config=apiserver.api-audiences=api,spire-server \
    --extra-config=apiserver.authorization-mode=Node,RBAC
```

- 切換kubectl的使用情境：

```
$ kubectl config use-context minikube
```

- 查看Cluster資訊：

```
$ kubectl cluster-info
```

- 確認節點狀態：

```
$ kubectl get nodes
```

SPIRE環境建置

- 畫面如下圖所示：

```
ubuntu@spiffe:~/spire$ kubectl cluster-info
Kubernetes control plane is running at https://192.168.49.2:8443
CoreDNS is running at https://192.168.49.2:8443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
ubuntu@spiffe:~/spire$ kubectl get nodes
NAME       STATUS   ROLES           AGE     VERSION
minikube   Ready    control-plane   2m43s   v1.33.1
```

SPIRE環境建置

下載SPIRE Server

- 準備一個工作目錄，然後把SPIRE的教學範例抓下來並切到Kubernetes快速上手的範例資料夾：

```
$ cd
$ mkdir spire
$ cd spire
$ git clone https://github.com/spiffe/spire-tutorials
$ cd ./spire-tutorials/k8s/quickstart/
```

- 建立一個名為spire的Kubernetes命名空間（Namespace），把之後SPIRE相關資源都放進同一個隔離區域。

```
$ kubectl apply -f spire-namespace.yaml
```

- 列出叢集裡所有命名空間

```
$ kubectl get namespaces
```

SPIRE環境建置

- 畫面如下圖所示：

```
ubuntu@spiffe:~/spire$ kubectl get namespaces
NAME              STATUS   AGE
default           Active   6m18s
kube-node-lease   Active   6m18s
kube-public       Active   6m18s
kube-system       Active   6m18s
spire             Active   48s
```

SPIRE環境建置

配置SPIRE Server

- 把SPIRE Server在叢集內運作所需的「身分、信任資料、權限」三件套建立好，為後續啟動Server Pod做準備。其中ServiceAccount（身分）表示Server以受控帳號運作；ConfigMap（信任）表示集中發佈與共享信任錨（Trust Anchor，在SPIFFE裡稱為Trust Bundle）；RBAC（權限）表示賦予Server合法的叢集/命名空間存取權：

```
$ kubectl apply \
    -f server-account.yaml \
    -f spire-bundle-configmap.yaml \
    -f server-cluster-role.yaml
```

- 畫面如右圖所示：

```
ubuntu@spiffe:~/spire/spire-tutorials/k8s/quickstart$ kubectl apply \
    -f server-configmap.yaml \
    -f server-statefulset.yaml \
    -f server-service.yaml
configmap/spire-server created
statefulset.apps/spire-server created
service/spire-server created
```

SPIRE環境建置

- 檢查在SPIRE Namespace裡的SPIRE Server部署是否正常運作：

```
$ kubectl get statefulset --namespace spire
```

```
$ kubectl get statefulset --namespace spire
NAME            READY   AGE
spire-server    1/1     2m27s
```

```
$ kubectl get pods --namespace spire
```

```
$ kubectl get pods --namespace spire
NAME              READY   STATUS    RESTARTS   AGE
spire-server-0    1/1     Running   0          2m35s
```

```
$ kubectl get services --namespace spire
```

```
$ kubectl get services --namespace spire
NAME           TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
spire-server   NodePort   10.109.12.172   <none>        8081:30966/TCP   2m43s
```

SPIRE環境建置

- 部署SPIRE Agent到Kubernetes的SPIRE Namespace：

1. 建立Agent的身分與權限：

```
$ kubectl apply \
      -f agent-account.yaml \
      -f agent-cluster-role.yaml
```

2. 提供設定，並在每個節點啟動Agent：

```
$ kubectl apply \
      -f agent-configmap.yaml \
      -f agent-daemonset.yaml
```

SPIRE環境建置

- 結果如下：

```
$ kubectl get daemonset --namespace spire

NAME           DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
spire-agent    1         1         1       1            1           <none>          6m45s

$ kubectl get pods --namespace spire

NAME                           READY   STATUS    RESTARTS   AGE
spire-agent-88cpl              1/1     Running   0          6m45s
spire-server-0                 1/1     Running   0          103m
```

SPIRE環境建置

- 在SPIRE Server Pod裡執行spire-server entry create，來註冊Registration Entries：

1. 建立Agent的節點身分（Node Entry）：

```
$ kubectl exec -n spire spire-server-0 -- \
    /opt/spire/bin/spire-server entry create \
    -spiffeID spiffe://example.org/ns/spire/sa/spire-agent \
    -selector k8s_psat:cluster:demo-cluster \
    -selector k8s_psat:agent_ns:spire \
    -selector k8s_psat:agent_sa:spire-agent \
    -node
```

2. 建立Workload（Pod）的身分（Workload Entry）：

```
$ kubectl exec -n spire spire-server-0 -- \
    /opt/spire/bin/spire-server entry create \
    -spiffeID spiffe://example.org/ns/default/sa/default \
    -parentID spiffe://example.org/ns/spire/sa/spire-agent \
    -selector k8s:ns:default \
    -selector k8s:sa:default
```

SPIRE環境建置

- 配置Workloads Container，來訪問SPIRE：

1. 建立Client Deployment。在SPIRE教學裡，client-deployment.yaml通常是啟動一個"Client" Workload（Pod），用來測試向SPIRE Agent申請SVID（透過本機Workload API）、跑個簡單的Client程式去呼叫Server，驗證mTLS/JWT-SVID等：

```
$ kubectl apply -f client-deployment.yaml
```

2. 進到標籤為app=client的Pod裡，用內部的spire-agent CLI透過Workload API的Unix Socket去抓取SVID與Trust Bundle，等同在該Pod內「扮演一個Workload」測試SPIRE是否能正常發證：

```
$ kubectl exec -it $(kubectl get pods -
o=jsonpath='{.items[0].metadata.name}' \
   -l app=client)  -- /opt/spire/bin/spire-agent api fetch -
socketPath /run/spire/sockets/agent.sock
```

SPIRE環境建置

- 結果如下：

```
ubuntu@spiffe:~/spire/spire-tutorials/k8s/quickstart$ kubectl exec -it $(kubectl get pods -o=jsonpath='{.items[0].metadata.name}' \
   -l app=client)  -- /opt/spire/bin/spire-agent api fetch -socketPath /run/spire/sockets/agent.sock
Received 1 svid after 2.740591ms

SPIFFE ID:              spiffe://example.org/ns/default/sa/default
SVID Valid After:       2025-08-08 12:15:39 +0000 UTC
SVID Valid Until:       2025-08-08 13:15:49 +0000 UTC
CA #1 Valid After:      2025-08-08 11:49:39 +0000 UTC
CA #1 Valid Until:      2025-08-09 11:49:49 +0000 UTC
```

SPIRE環境建置

結合Envoy以及X.509

- 接下來示範把Envoy跟SPIRE結合，使用X.509憑證做身分驗證（mTLS）。SPIRE發的憑證或金鑰不需要你手動放到容器裡，會由Envoy的SDS（Secret Discovery Service）動態向SPIRE取得。

1. 設定外部IP：這裡會需要一個可以分配外部IP的軟體，使用的是MetalLB。

```
$ kubectl apply -f
https://raw.githubusercontent.com/metallb/metallb/v0.13.7/config/man
ifests/metallb-native.yaml

#等待metallb開始
$ kubectl wait --namespace metallb-system \
                --for=condition=ready pod \
                --selector=app=metallb \
                --timeout=90s
```

SPIRE環境建置

A. 此步驟會建立兩個Metallb-System的Pod：

- Controller Pod（控制面，Deployment）：負責給Service分配外部IP。
- Speaker Pod（資料面，DaemonSet）：負責對外宣告與轉發外部IP。

B. Apply Metallb設定：首先確認Minikube所在網段，再把MetalLB的可分配IP範圍設成同一個網段，最後套用設定。目的：讓你建立type: LoadBalancer的Service時，MetalLB能發一個在同一子網、外部可達的EXTERNAL-IP。

- 查看Minikube的IP：

```
$ minikube ip
```

SPIRE環境建置

- 查看metallb-config.yaml裡面的IP網段是否為一樣的(xxx.xxx.xxx.200-xxx.xxx.xxx.250)，若有不同須將其改為相同的。
- 畫面如右圖所示：

```yaml
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: example
  namespace: metallb-system
spec:
  addresses:
  - 192.168.49.200-192.168.49.250
---
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: empty
  namespace: metallb-system
```

- 最後套用設定：

```
#工作目錄需在.../spire-tutorials/k8s/envoy-x509
$ kubectl apply -f metallb-config.yaml
```

SPIRE環境建置

2. 執行Workloads

   A. 此部分會建立三個Workload：

      - 兩個前端（Front-End）：模擬兩個獨立的前端服務 / 應用。
      - 一個後端（Back-End）：用Nginx（Web Server）提供靜態檔案的簡單示例服務。

   B. 部署所有Workloads：

      - 請先確定當前目錄為.../spire-tutorials/k8s/envoy-x509。
      - 啟用Kustomize（Kubernetes的宣告式組態客製化工具）一次性部署整個k8s/目錄下定義的所有Kubernetes資源。

```
$ kubectl apply -k k8s/.
```

SPIRE環境建置

- 畫面如下圖所示：

```
ubuntu@spiffe:~/spire/spire-tutorials/k8s/envoy-x509$ kubectl apply -k k8s/.
configmap/backend-balance-json-data created
configmap/backend-envoy created
configmap/backend-profile-json-data created
configmap/backend-transactions-json-data created
configmap/frontend-2-envoy created
configmap/frontend-envoy created
configmap/symbank-webapp-2-config created
configmap/symbank-webapp-config created
service/backend-envoy created
service/frontend created
service/frontend-2 created
deployment.apps/backend created
deployment.apps/frontend created
deployment.apps/frontend-2 created
```

SPIRE環境建置

- TLS憑證設定說明（在安裝流程中已生成，不需自己動手打）：
- 設定Envoy當「Server端」時的mTLS行為，並且把憑證與信任錨（Trust Bundle）都透過SDS向SPIRE Agent動態取得。
  - 這是Server端TLS（DownstreamTlsContext）：

```
# 指定TLS傳輸，DownstreamTlsContext代表Envoy為伺服器角色
transport_socket:
  name: envoy.transport_sockets.tls
  typed_config:
    "@type": type.googleapis.com/envoy.extensions.transport_sockets.tls.v3.DownstreamTlsContext
```

  - DownstreamTlsContext：此Listener是" Server "角色（下游Client連進來）。

SPIRE環境建置

- 透過SDS取得「自己的X.509-SVID（憑證+私鑰）」：

```
common_tls_context:
  # 告訴Envoy用SDS拿TlsCertificate
  tls_certificate_sds_secret_configs:
  # SPIEEF ID
  - name: "spiffe://example.org/ns/default/sa/default/backend"
    sds_config:
      resource_api_version: V3
      api_config_source:
        api_type: GRPC
        transport_api_version: V3
        grpc_services:
          envoy_grpc:
            cluster_name: spire_agent
```

  - tls_certificate_sds_secret_configs：叫SDS把Server端要出示的憑證 / 私鑰送來。
  - name 寫成 spiffe://.../backend：表示要的這份憑證對應的SPIFFE ID（X.509-SVID）。
  - cluster_name: spire_agent這行：指向事先定義好的gRPC連到SPIRE Agent的Cluster（通常是UDS：/run/spire/sockets/agent.sock）。

SPIRE環境建置

- 意義：不用把cert/key放檔案；SPIRE Agent會把最新的SVID動態下發（支援輪替）。

- 啟用mTLS：要求並驗證「Client端憑證」：

```
# 授權白名單，要求client必須是指定的SPIFFE ID
# 以下面寫法來看就是只允許frontend跟frontend-2這兩個身分連進來(零信任中以身分授權的作法)
combined_validation_context:
  # validate the SPIFFE ID of incoming clients (optionally)
  default_validation_context:
    match_typed_subject_alt_names:
    - san_type: URI
      matcher:
        exact: "spiffe://example.org/ns/default/sa/default/frontend"
    - san_type: URI
      matcher:
        exact: "spiffe://example.org/ns/default/sa/default/frontend-2"
```

  - match_typed_subject_alt_names：只接受SAN(URI)等於這兩個SPIFFE ID的Client端：“.../frontend”與“.../frontend-2”。只有這兩個前端的SVID可以連進來，其他憑證一律拒絕。

SPIRE環境建置

- 使用SDS拿Trust Bundle驗證對方鏈：

```
# 向SDS拿驗證用的信任bundle
validation_context_sds_secret_config:
  # trust domain
  name: "spiffe://example.org"
  sds_config:
    resource_api_version: V3
    api_config_source:
      api_type: GRPC
      transport_api_version: V3
      grpc_services:
        envoy_grpc:
          cluster_name: spire_agent
```

  - 這是驗證用的Bundle（根/中繼憑證集合），用來檢查對方的憑證鏈是否鏈到受信的Trust Domain（example.org）。
  - 同樣走SDS，由 SPIRE Agent提供，會跟著輪替更新。

SPIRE環境建置

C. 註冊服務：

- 為了獲得SPIRE頒發的X.509憑證，必須註冊服務。
  ```
  $ bash create-registration-entries.sh
  ```
- 畫面如右圖所示：