

# 從DevSecOps 談微服務/容器應用開發 的資訊安全

主講人：查士朝

國立臺灣科技大學資訊管理系 教授兼系主任

國立臺灣科技大學資通安全研究與教學中心 主任



# 查(出丫)士朝 (彳么')

- 台大資管博士
- 現職 臺灣科技大學
  - 資訊管理系教授兼系主任
  - 資通安全研究與教學中心主任
- 經歷
  - 曾任意藍科技股份有限公司資深技術顧問
  - 曾任資誠企業管理顧問股份有限公司協理
  - 曾任台灣大學工商管理學系兼任助理教授
- 近年來執行多項科技部與產學合作計畫
- 協助多家機構建立資訊安全管理制度
- 協助政府建立物聯網及手機應用程式安全標準
- 專長
  - 資訊安全架構
  - 資訊安全程式設計
  - 新興科技資訊安全風險分析與管理

## 個人擁有認證

### 技術

- Google Associate Android Developer
- CCDH (Developer for Apache Hadoop (CCDH))
- EMCISA
- RFID+
- MCSD.NET
- MCSE
- Sun Certified Enterprise Architect (SCEA)
- Sun Certified Java Programmer (SCEA)
- Linux Professional Institute Certification Level 1 and 2
- CKA, CKAD, CKS

### 資訊安全技術

- OSCP
- Certified Wireless Security Professional

### 資訊安全管理

- CCFP
- CCSK (Certified Cloud Security Knowledge)
- CSSLP
- CISSP
- CISM
- BS7799 LA
- IEC 62443-2-1 LA

### 管理/資訊治理

- Certified Scrum Master
- ITIL Service Manager
- PMP

# 簡報大綱





## DevSecOps 的簡介與相關概念

DevSecOps 的主要面向

建立適合推動 DevSecOps 的文化

將安全納入 DevOps 的程序中

從 4C 的考量

結論

# 開發 (Development)

微服務 (Microservice)

持續整合 (Continuous Integration)

持續交付 (Continuous Delivery)

持續部署 (Continuous Deployment)

架構及程式 (Infrastructure as Code)

雲原生架構 (Cloud Native)

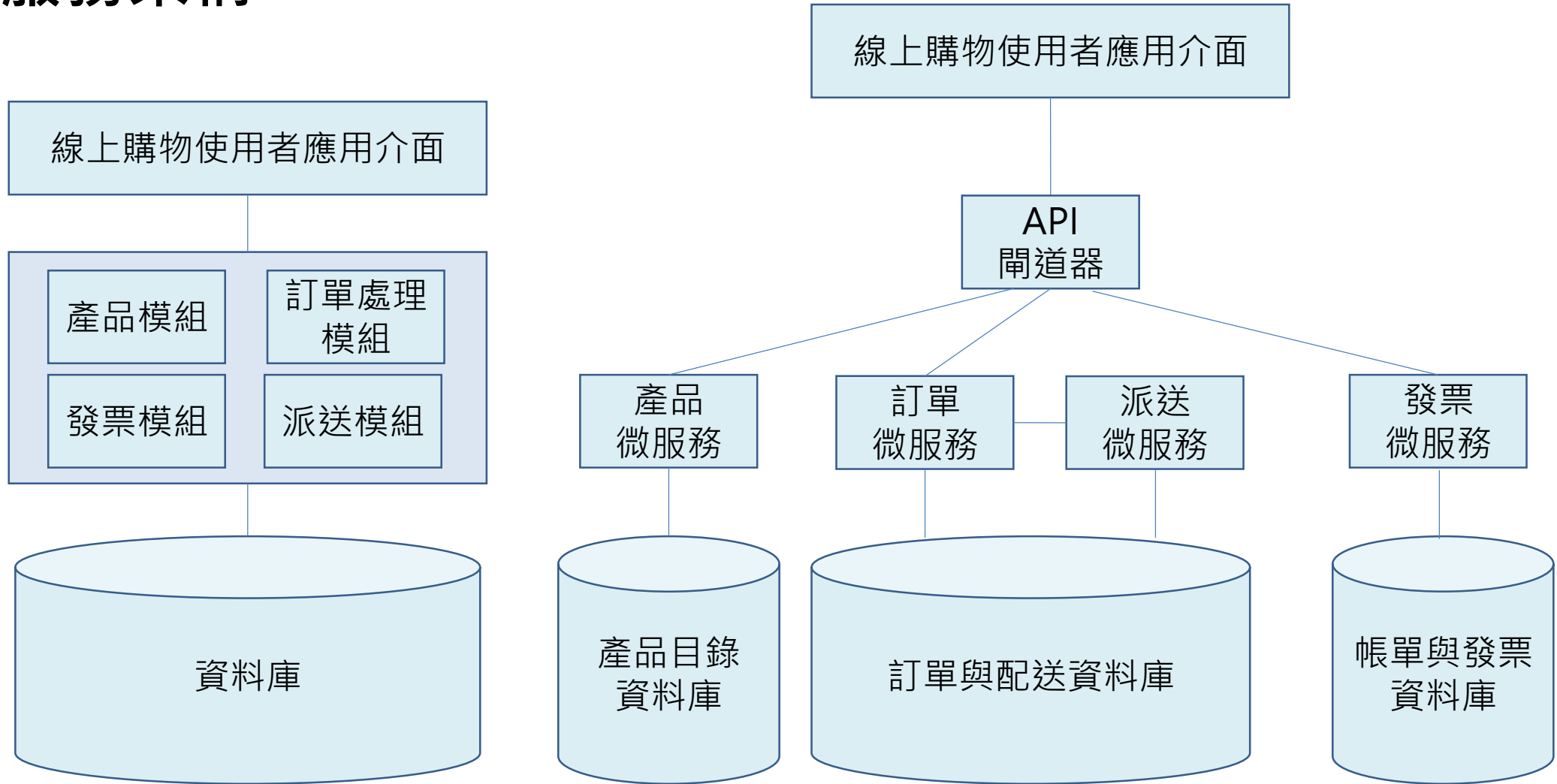
容器技術 (Container)

雲端運算 (Cloud Computing)

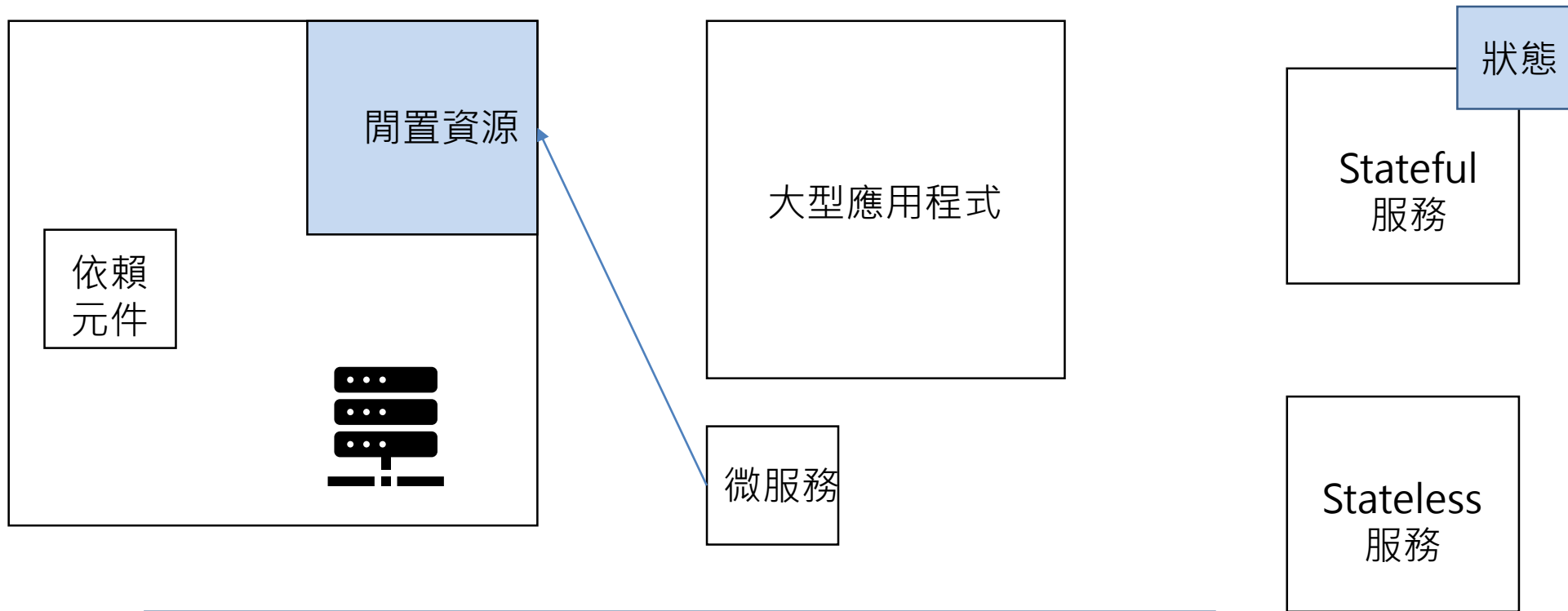
虛擬化技術 (Virtualization)

# 骨幹管理與維運 (Operations)

# 微服務架構

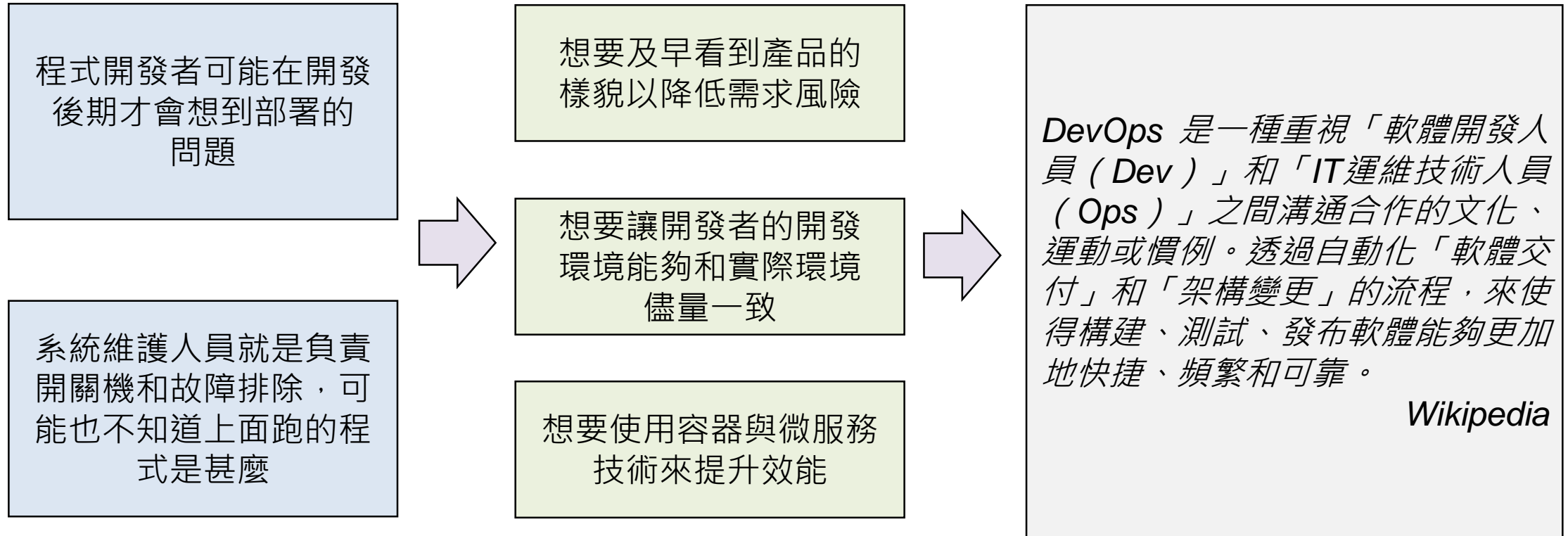


# 容器技術微服務架構促成了 DevOps 的成形



當服務的部署需要考慮到骨幹時，就開發就不能不看骨幹

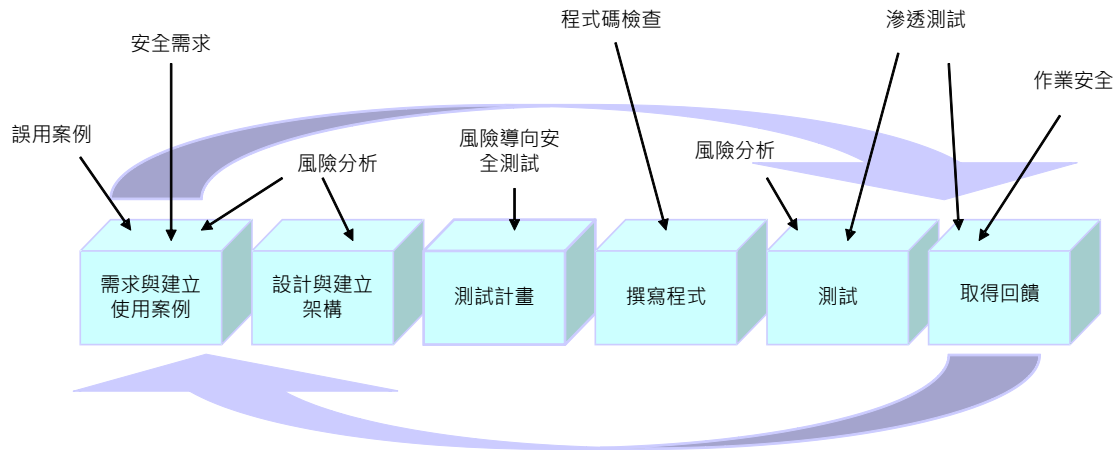
# 從 DevOps 到 DevSecOps



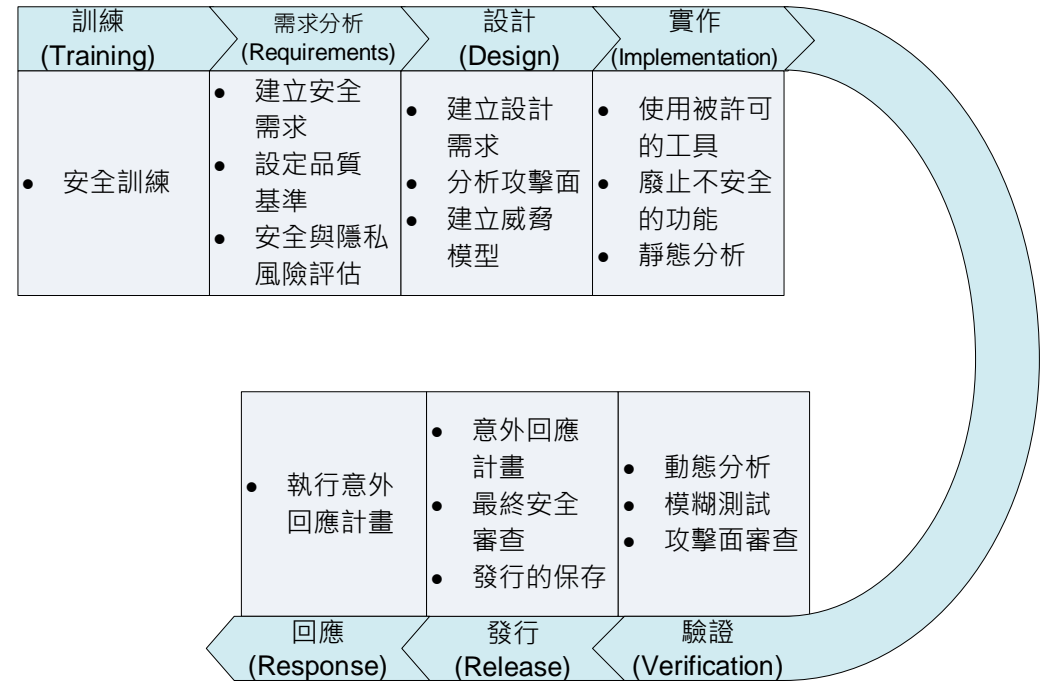
DevSecOps 是將資安納入 DevOps 的軟體開發與交付程序中，而能夠透過對考量資訊安全的程序，來確保能夠快速交付考量資訊安全的軟體、服務，與應用



# DevSecOps 與軟體開發生命週期安全

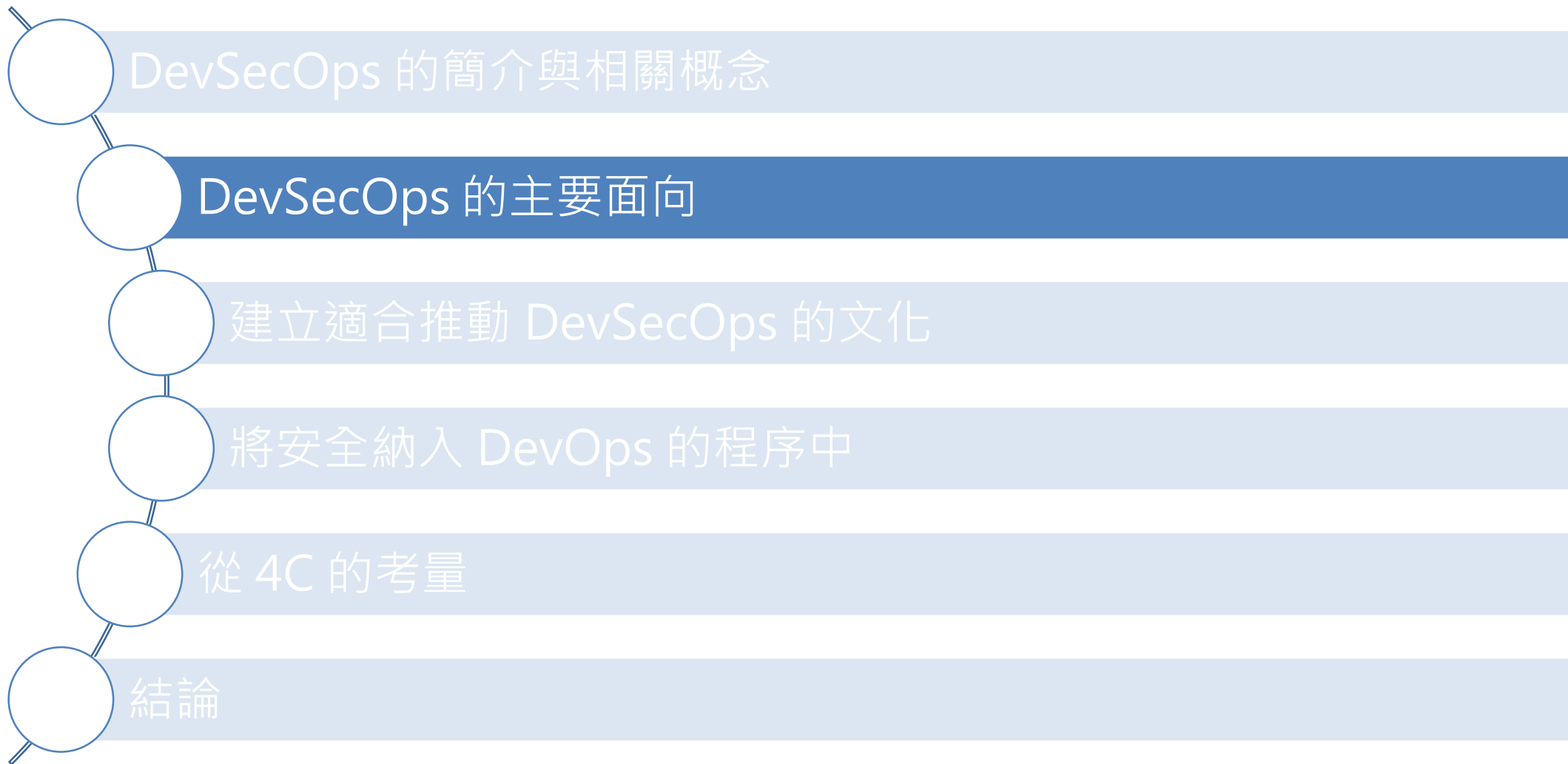


McGraw's Touch Point



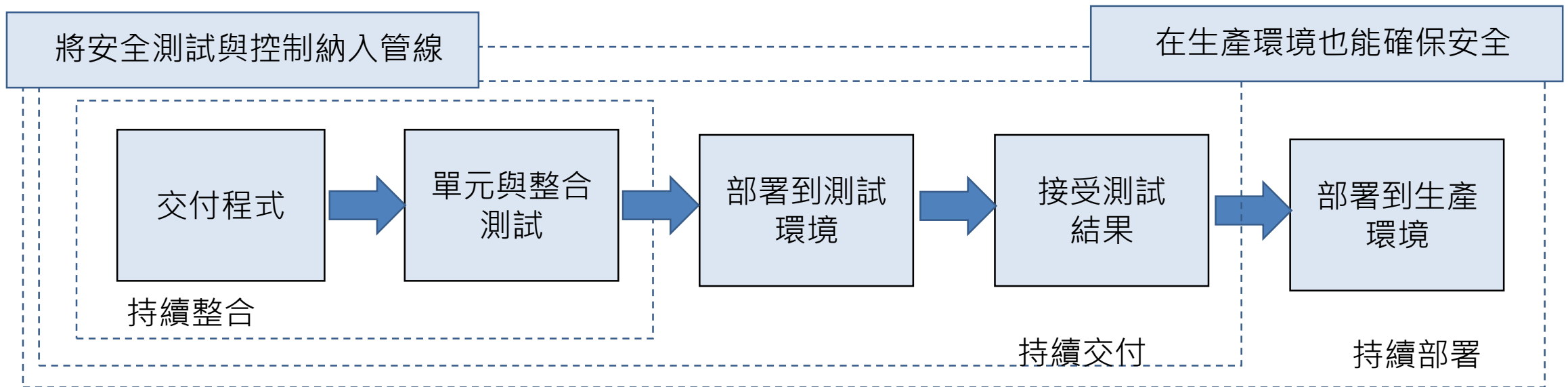
微軟的 SDL

DevSecOps 強調能夠更進一步的透過自動化方法，將資訊安全納入整個軟體開發生命週期當中，以持續落實安全

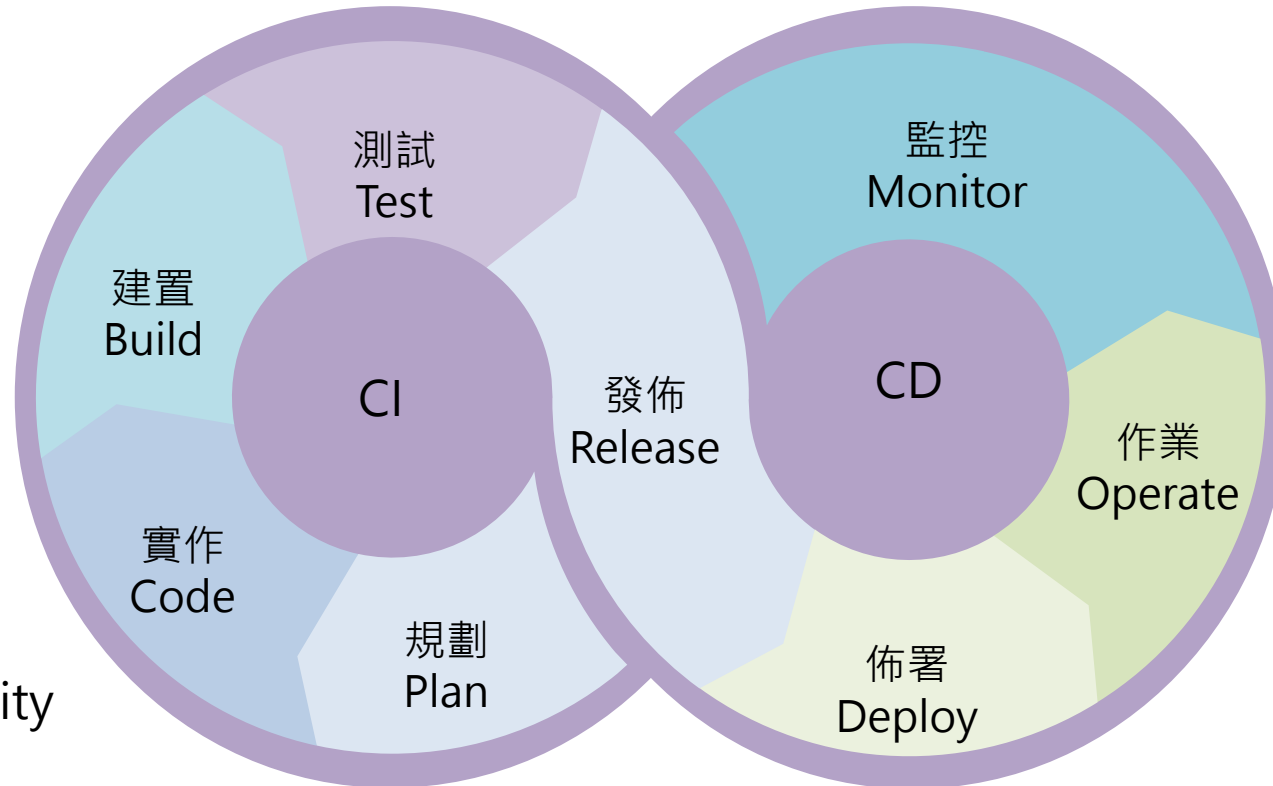


# 一般還是會比較著重在 DevOps 或 CI/CD 的延伸

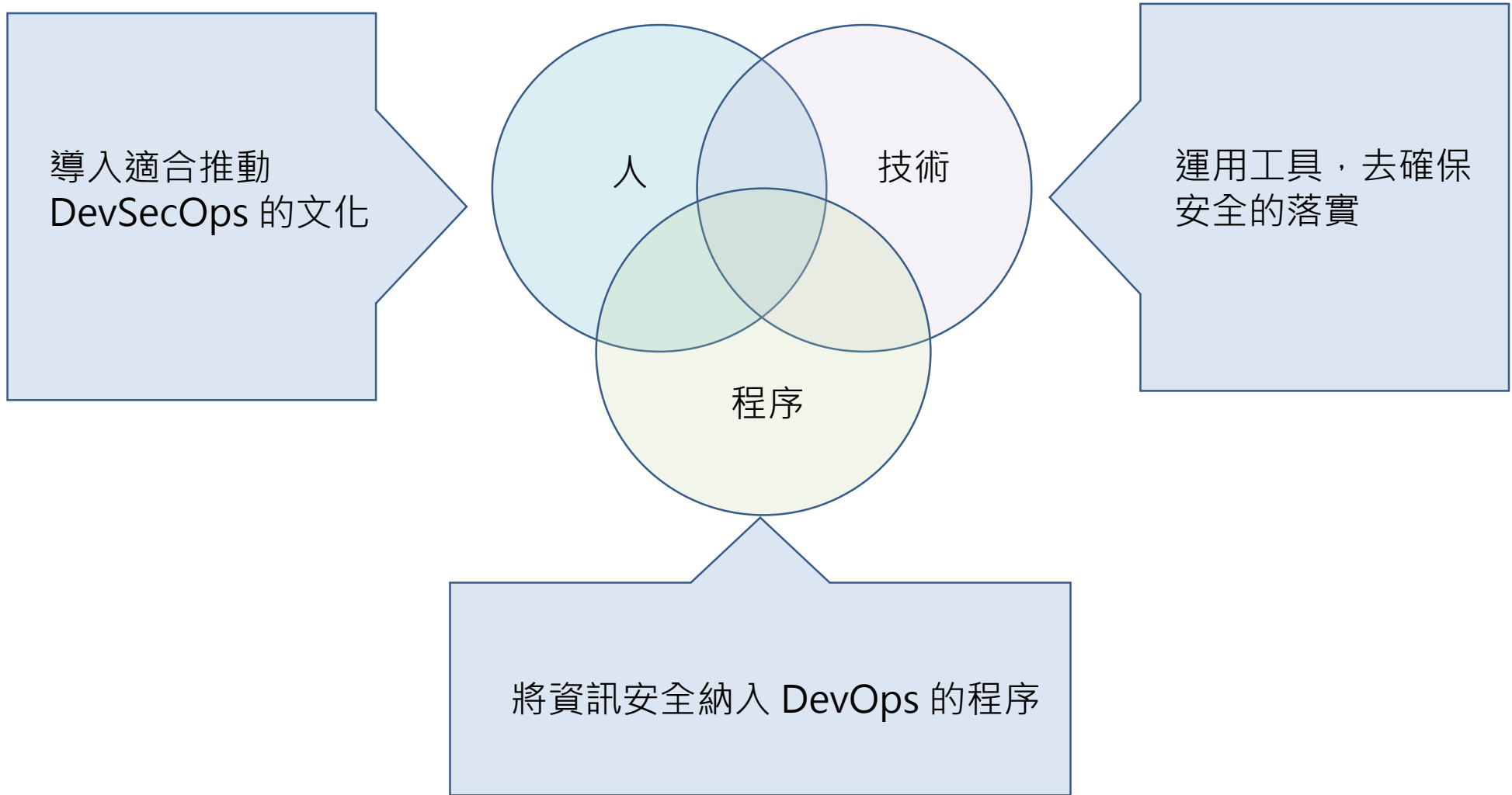
- DevSecOps 的關鍵元素與實作步驟：
  - 導入程式開發管線與 CI/CD 管線的概念 (Concept of Pipelines and the CI/CD pipeline)
  - 善用工具以完成 CI/CD 管線 (Building Blocks for CI/CD pipelines)
  - 設計與執行 CI/CD 管線 (Designing and Executing the CI/CD pipeline )
  - 自動化的策略 (Strategies for Automation)
  - 對於 CI/CD 管線中自動化工具的需求 (Requirements for Automation Tools in CI/CD Pipelines)



# 從 CI/CD 到持續安全



持續安全  
Continuous Security





加強資安認知

定義資訊安全的角色  
與責任

由新聞事件或案例中  
學習

# 建立適合推動 DevSecOps 的文化

- 要將資訊安全與 DevOps 結合
- 善用驅動持續安全的能力以進行加速
  - 從領導開始推動
  - 著重在重視安全的價值觀建立
  - 及早將安全納入 (Shift Left)
  - 加強開發與作業人員的能力
  - 測試的自動化
  - 由一些很小的動作開始
  - 讓團隊成員有所準備
  - 將工作視覺化並且主動通知
  - 執行監控



# 敏捷開發宣言 Manifesto for Agile Software Development

- 藉著親自並協助他人進行軟體開發，我們正致力於發掘更優良的軟體開發方法。透過這樣的努力，我們已建立以下價值觀：
  - 個人與互動重於流程與工具 (Individuals and interactions over processes and tools)
  - 可用的軟體重於詳盡的文件 (Working software over comprehensive documentation)
  - 與客戶合作重於合約協商 (Customer collaboration over contract negotiation)
  - 回應變化重於遵循計劃 (Responding to change over following a plan)
- 也就是說，雖然右側項目有其價值，但我們更重視左側項目。

過去資訊安全在軟體開發領域不是那麼被強調的項目

# 持續整合宣言 Manifesto of CI

CI 強調測試，但安全不見得有被測試

- 測試本身的價值在影響到觀察者行為後才會彰顯 (A test has zero value until its result affects the behavior of an observer)
- 對於測試結果的最好觀察者是會引起變更的開發者或程式包裝者 (The best observer for a test result is the developer or packager who instigated and/or initiated the change being integrated)
- 持續整合的價值與變更的大小成反比。在多次循環中的小變更勝過於一次包裹以後做很大的變更 (The benefit of continuous integration is inversely proportional to the size of the change. Many iterative small changes far outweigh a massive bundled change.)
- 對於開發者或程式包裝者的快速回應可以引起他們的注意 (Rapid feedback to the developer or packager of the change will cause them to pay attention. Aim for hours not days to provide test results)
- 程式包裝者只負責他們可以貢獻的那部分 (Packagers only take responsibility for tests that they can contribute to)
- 程式包裝者只尊重會產生可靠結果的測試 (Packagers respect tests and testing systems that produce reliable results. Conversely they ignore and shun tests and systems that are flakey)
- 程式包裝者不應該需要去找尋外部的測試結果 Packagers should not have to search for test results outside of their workflow.
- 程式包裝者應該要能夠在他們自己的電腦去執行測試 (Packagers should be able to run individual tests on their own machines)
- 理想的測試是能夠與其變更結合的 (The ideal test can be updated in lockstep with the changes it is testing. Aim to store a test along with the software that the test is most related to)
- 測試最好在影響到他人前進行 (The best place to test a change is before that change affects anyone else)
- 符合這些原則的小套件會比不能滿足這些原則的大套件要有價值 (A small suite of tests that follows these principles is more valuable to continuous integration than large suite of tests that does not)

# 堅固耐用宣言 (The Rugged Manifesto for Rugged DevOps)

- 我是嚴格的，更重要的是，我的程式碼是堅固耐用的 (I am rugged and, more importantly, my code is rugged)
- 我了解軟體已經成為現今社會的基礎 (I recognize that software has become a foundation of our modern world)
- 我了解我所扮演角色的重大責任 (I recognize the awesome responsibility that comes with this foundational role)
- 我了解我的程式可能被使用在我沒有預期到的地方，而且會比我預期的使用得更久 (I recognize that my code will be used in ways I cannot anticipate, in ways it was not designed, and for longer than it was ever intended)
- 我了解我的程式會被聰明的攻擊者持續的攻擊，而會對於我們實體安全、經濟安全，與國家安全造成威脅 (I recognize that my code will be attacked by talented and persistent adversaries who threaten our physical, economic, and national security)
- 我了解這些現實，而且我選擇嚴格的面對 (I recognize these things - and I choose to be rugged)
- 我嚴格因為我拒絕成為弱點的來源 (I am rugged because I refuse to be a source of vulnerability or weakness)
- 我嚴格因為我確保我的程式能夠支援他的任務 (I am rugged because I assure my code will support its mission)
- 我嚴格因為我程式能夠面對這些挑戰 (I am rugged because my code can face these challenges and persist in spite of them)
- 我嚴格不是因為這容易，而是因為我必須要面對這些挑戰 (I am rugged, not because it is easy, but because it is necessary and I am up for the challenge)



# 將安全融入開發與作業程序宣言 Manifesto of DevSecOps

- 在開發過程中不要忘了資料安全與隱私的議題，在實現創新產品之前，要先確認是安全的以後才能放行，不只依靠掃描工具，不等待淪落為攻擊與錯誤的犧牲者
- 抱持以下價值：
  - 勇於嘗試勝過總是說不 (Leaning in over Always Saying "No")
  - 以科學的方法確保安全勝過恐懼、不確定性與疑惑 (Data & Security Science over Fear, Uncertainty and Doubt)
  - 接納貢獻與合作勝過只有安全的需求 (Open Contribution & Collaboration over Security-Only Requirements)
  - 可透過 API 使用的安全服務勝過強制的安全控制與文書作業 (Consumable Security Services with APIs over Mandated Security Controls & Paperwork)
  - 使用以業務為導向的安全分數而非安全橡皮圖章 (Business Driven Security Scores over Rubber Stamp Security)
  - 透過紅藍軍測試勝過只依賴掃描出的理論上弱點 (Red & Blue Team Exploit Testing over Relying on Scans & Theoretical Vulnerabilities)
  - 24x7 的主動安全監控勝過被通知有意外事件後進行回應 (24x7 Proactive Security Monitoring over Reacting after being Informed of an Incident)
  - 分享威脅情資而非只保留在自己手中 (Shared Threat Intelligence over Keeping Info to Ourselves)
  - 合規的作業而非只使用檢查表去進行檢查 (Compliance Operations over Clipboards & Checklists)

高階主管應宣導資訊安全重要性並提供資源

業務單位人員應該配合參與安全測試

開發團隊在應用程式中加入資訊安全設計

安全團隊執行監控、稽核與更新

QA 團隊加入資訊安全的測試案例

建構與整合團隊於管道中加入安全測試工具

骨幹團隊建立、維護，與更新骨幹並確保安全性

部屬與發佈團隊維護設定之安全性



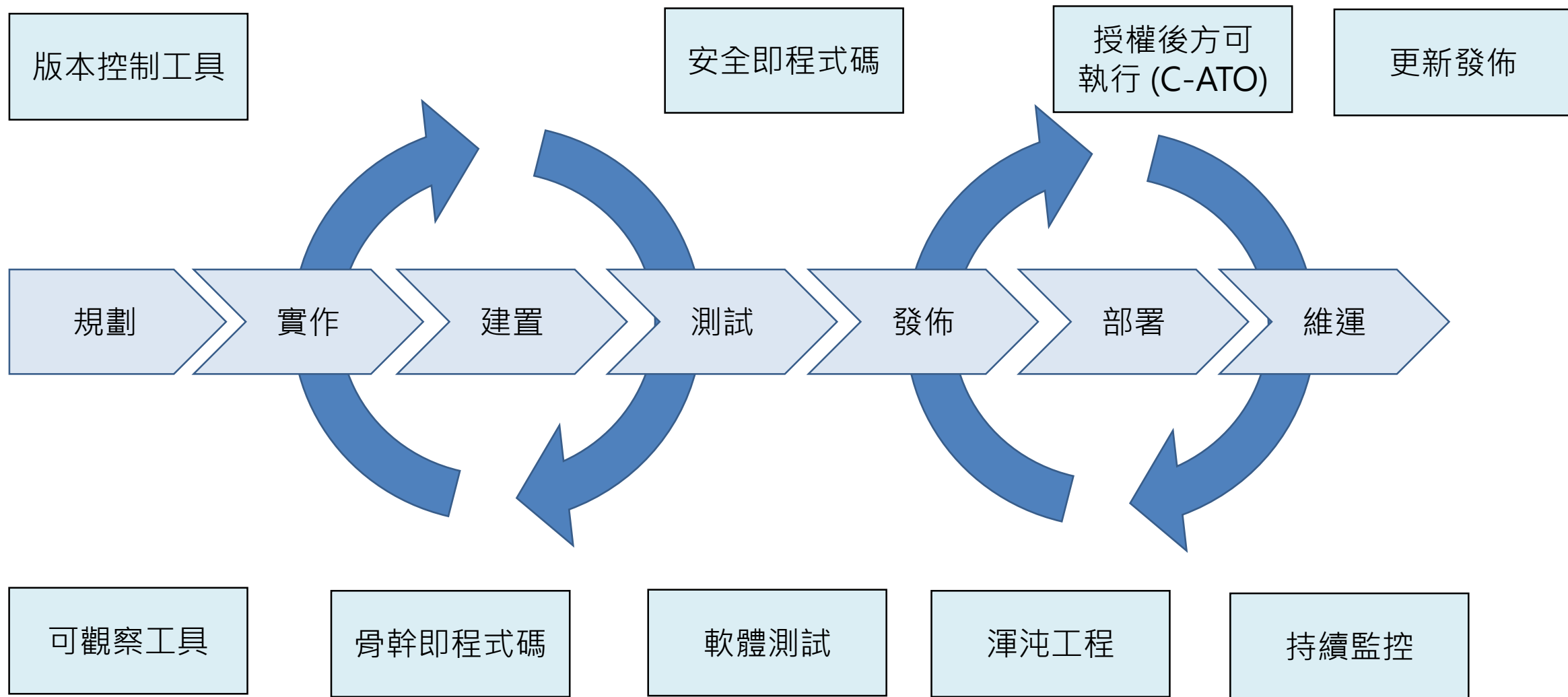
# 資通安全責任等級分級辦法附表十 資通系統防護基準 (節錄)

		高	中	普
系統與服務獲得	需求階段	針對系統安全需求 ( 含機密性、可用性、完整性 ) ，以檢核表方式進行確認。		
	設計階段	一、根據系統功能與要求，識別可能影響系統之威脅，進行風險分析及評估。 二、將風險評估結果回饋需求階段之檢核項目，並提出安全需求修正。		無要求。
	開發階段	一、執行「源碼掃描」安全檢測。 二、具備系統嚴重錯誤之通知機制。 三、等級「中」及「普」之所有控制措施。	一、應針對安全需求實作必要控制措施。 二、應注意避免軟體常見漏洞及實作必要控制措施。 三、發生錯誤時，使用者頁面僅顯示簡短錯誤訊息及代碼，不包含詳細之錯誤訊息。	
	測試階段	一、執行「滲透測試」安全檢測。 二、等級「中」及「普」之所有控制措施。	執行「弱點掃描」安全檢測。	

# 資通安全責任等級分級辦法附表十 資通系統防護基準 (節錄)

		高	中	普
系統與服務 獲得	部署與維運 階段	一、於系統發展生命週期之維運階段，須注意版本控制與變更管理。 二、等級「普」之所有控制措施。		一、於部署環境中應針對相關資通安全威脅，進行更新與修補，並關閉不必要服務及埠口。 二、資通系統相關軟體，不使用預設密碼。
	委外階段	資通系統開發如委外辦理，應將系統發展生命週期各階段依等級將安全需求（含機密性、可用性、完整性）納入委外契約。		
	獲得程序	開發、測試及正式作業環境應為區隔。		無要求。
	系統文件	應儲存與管理系統發展生命週期之相關文件。		





# 使用版本控制工具來提升 DevSecOps 安全

- 版本控制工具本身就可以追蹤程式的變更
- 如果是自己架的版本，要做好安全設定 (身分鑑別、存取控制，連線安全)
- 如果是使用 github 或 gitlab，本身有些安全實務可以遵循。目前許多平台也有提供進階安全功能：
  - 程式碼掃描
  - 授權合規判斷
  - 相依性檢視
  - 秘密資訊偵測

## Best practices for user security

Outside of instance-level security measures (SSL, subdomain isolation, configuring a firewall) that a site administrator can implement, there are steps your users can take to help protect your enterprise.

### Enabling two-factor authentication

Two-factor authentication (2FA) is a way of logging in to websites and services that requires a second factor beyond a password for authentication. In GitHub Enterprise Server's case, this second factor is a one time authentication code generated by an application on a user's smartphone. We strongly recommend requiring your users to enable two-factor authentication on their accounts. With two-factor authentication, both a user's password and their smartphone would have to be compromised to allow the account itself to be compromised.

For more information on configuring two-factor authentication, see "[About two-factor authentication](#)".

### Requiring a password manager

#### In this article

[Enabling two-factor authentication](#)[Requiring a password manager](#)[Restrict access to teams and repositories](#)

## About GitHub Advanced Security

---

GitHub has many features that help you improve and maintain the quality of your code. Some of these are included in all plans, such as dependency graph and Dependabot alerts. Other security features require a license for GitHub Advanced Security to run on repositories apart from public repositories on GitHub.com.

For more information about purchasing GitHub Advanced Security, see "[About billing for GitHub Advanced Security](#)."

## About Advanced Security features

---

A GitHub Advanced Security license provides the following additional features:

- **Code scanning** - Search for potential security vulnerabilities and coding errors in your code. For more information, see "[About code scanning](#)."
- **Secret scanning** - Detect secrets, for example keys and tokens, that have been checked into the repository. For more information, see "[About secret scanning](#)."
- **Dependency review** - Show the full impact of changes to dependencies and see details of any vulnerable versions before you merge a pull request. For more information, see "[About dependency review](#)."



<https://docs.github.com/en/get-started/learning-about-github/about-github-advanced-security>

# License Compliance ULTIMATE

Introduced in GitLab 11.0.

If you're using [GitLab CI/CD](#), you can use License Compliance to search your project's dependencies for their licenses. You can then decide whether to allow or deny the use of each license. For example, if your application uses an external (open source) library whose license is incompatible with yours, then you can deny the use of that license.

You can take advantage of License Compliance by either:

- [Including the job](#) in your existing `.gitlab-ci.yml` file.
- Implicitly using [Auto License Compliance](#), provided by [Auto DevOps](#).

Try License Compliance scanning to search project dependencies in GitLab Ultimate. [It's free for 30 days.](#)

To detect the licenses in use, License Compliance uses the [License Finder](#) scan tool that runs as part of the CI/CD pipeline. For the job to activate, License Finder needs to find a compatible package definition in the project directory. For details, see the [Activation on License Finder documentation](#). GitLab checks the License Compliance report, compares the licenses between the source and target branches, and shows the information right on the merge request. Denied licenses are indicated by a `x` red icon next to them as well as new licenses that need a decision from you. In addition, you can [manually allow or deny](#) licenses in your project's license compliance policy section. If a denied license is detected in a new commit, GitLab blocks any merge requests containing that commit and instructs the developer to remove the license.

**i** If the license compliance report doesn't have anything to compare to, no information is displayed in the merge request area. That is the case when you add the `license_scanning` job in your `.gitlab-ci.yml` for the first time. Consecutive merge requests have something to compare to and the license compliance report is shown properly.

**i** License Compliance detected 6 licenses and policy violations for the source branch only; approval required **!** [Manage licenses](#) [View full report](#) [Collapse](#)

**Denied**  
Out-of-compliance with this project's policies and should be removed

**x** [MIT License](#) Used by `async`, `debug`, `lodash`, and [10 more](#)

# Secret Detection ALL TIERS

Version history ⋮

A recurring problem when developing applications is that developers may unintentionally commit secrets and credentials to their remote repositories. If other people have access to the source, or if the project is public, the sensitive information is then exposed and can be leveraged by malicious users to gain access to resources like deployment environments.

GitLab 11.9 includes a new check called Secret Detection. It scans the content of the repository to find API keys and other information that should not be there.

GitLab displays identified secrets visibly in a few places:

- [Security Dashboard](#)
- Pipelines' **Security** tab
- Report in the merge request widget

 Secret scanning detected **3 new critical** severity vulnerabilities. 

#### New

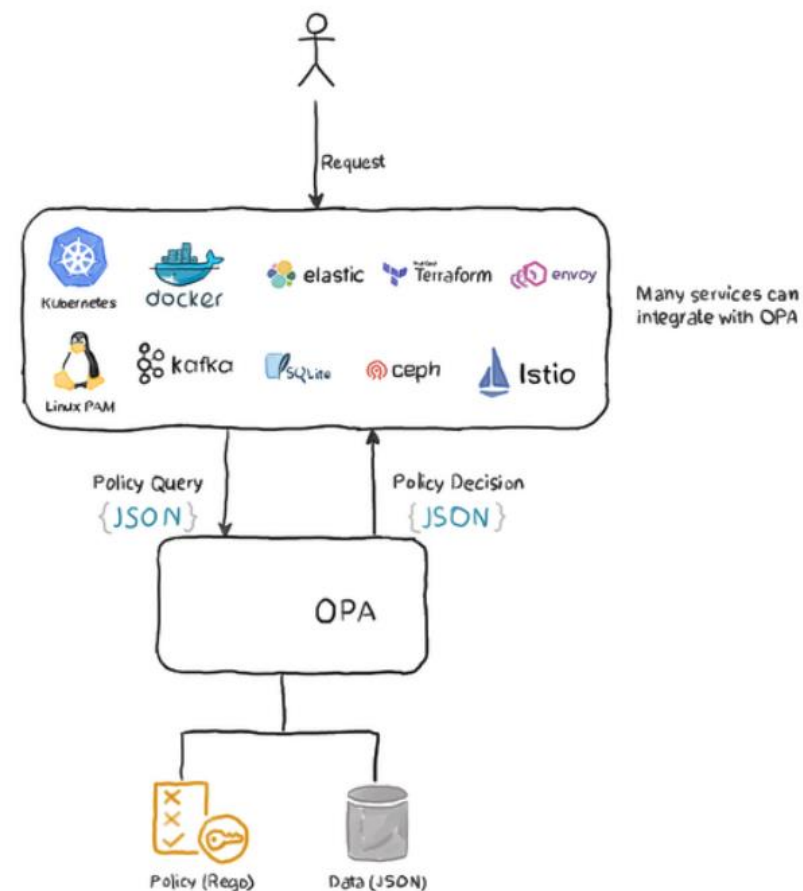
- Critical [Password in URL in dir/sub-dir/file.gl](#)
- Critical [Password in URL in dir/sub-dir/file.gl](#)
- Critical [Password in URL in dir/sub-dir/file.gl](#)

# 對 Infrastructure as Code (IaC) 的安全強化

- 使用 Git 儲存去保存骨幹設定
  - 可以使用 Ansible 或 Terraform 等去對設定進行描述
- 可以透過 diff 工具去比較差異，並確保變更是經過審查
- 不應允許直接 Access 伺服器
- 要保護 Repository
- 可以透過掃描工具，檢查是否足夠安全

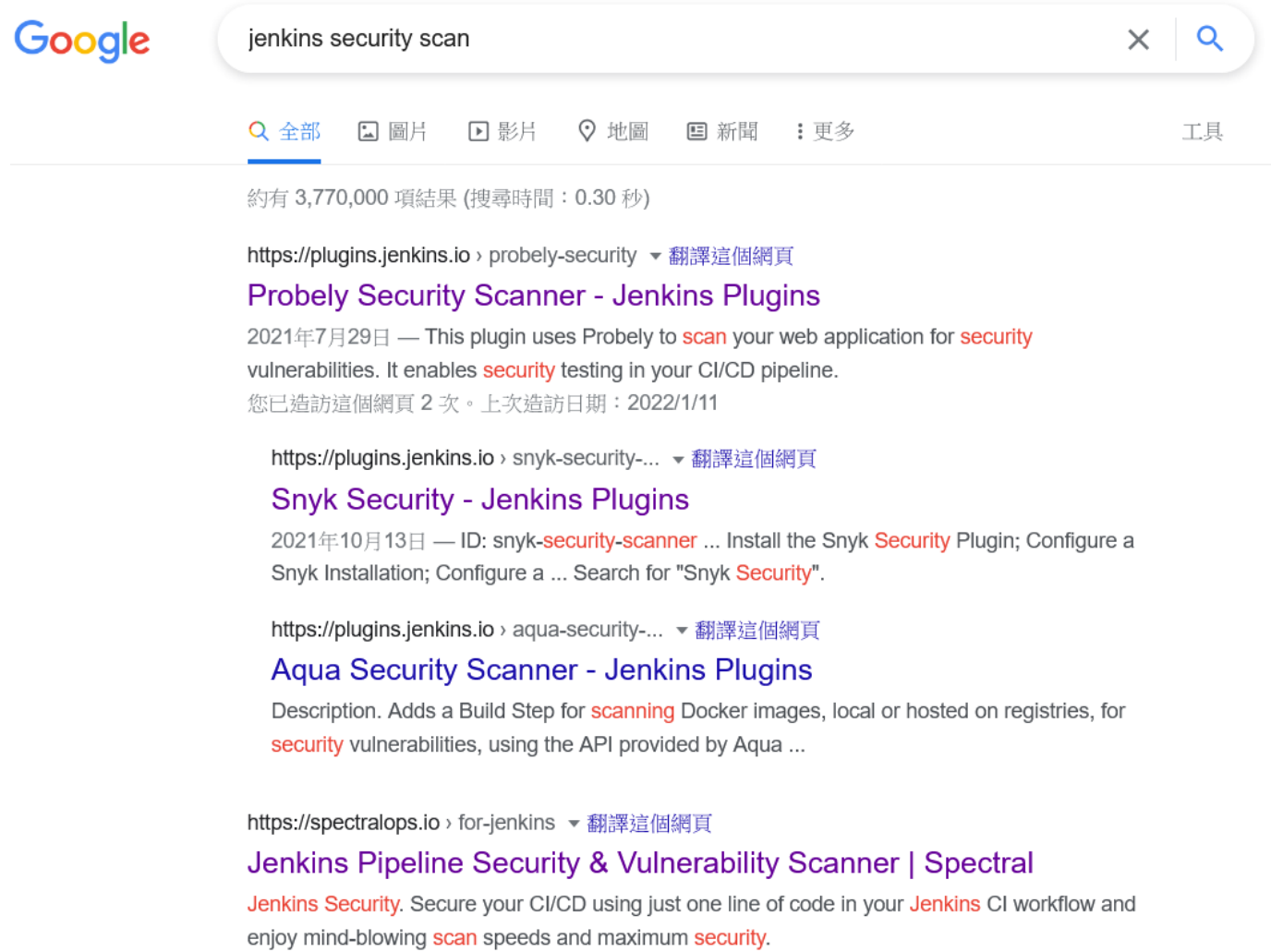
# Security as Code

- 宣告安全需求，讓系統運作時去確保相關安全要求有被落實，例如：
  - Open Policy Agent (OPA)
  - Aqua's Apolicy,
  - HashiCorp's Sentinel
- 政策種類 (NIST SP 800-204c)
  - 網路政策與零信任政策
  - 實作架構政策 (例如容器不可以 root 執行)
  - 儲存政策
  - 存取控制政策
  - 供應鏈政策





# 可以與 CI/CD 工具整合，要求進行某些檢查或掃描



Google jenkins security scan

全部 圖片 影片 地圖 新聞 更多 工具

約有 3,770,000 項結果 (搜尋時間：0.30 秒)

<https://plugins.jenkins.io/probely-security> 翻譯這個網頁  
**Probely Security Scanner - Jenkins Plugins**  
2021年7月29日 — This plugin uses Probely to **scan** your web application for **security** vulnerabilities. It enables **security** testing in your CI/CD pipeline.  
您已造訪這個網頁 2 次。上次造訪日期：2022/1/11

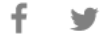
<https://plugins.jenkins.io/snyk-security-scanner> 翻譯這個網頁  
**Snyk Security - Jenkins Plugins**  
2021年10月13日 — ID: snyk-**security-scanner** ... Install the Snyk **Security** Plugin; Configure a Snyk Installation; Configure a ... Search for "Snyk **Security**".

<https://plugins.jenkins.io/aqua-security-scanner> 翻譯這個網頁  
**Aqua Security Scanner - Jenkins Plugins**  
Description. Adds a Build Step for **scanning** Docker images, local or hosted on registries, for **security** vulnerabilities, using the API provided by Aqua ...

<https://spectralops.io/for-jenkins> 翻譯這個網頁  
**Jenkins Pipeline Security & Vulnerability Scanner | Spectral**  
**Jenkins Security**. Secure your CI/CD using just one line of code in your **Jenkins** CI workflow and enjoy mind-blowing **scan** speeds and maximum **security**.

# 建立 c-ATO 機制

## authorization to operate (ATO)



### Abbreviation(s) and Synonym(s):


[ATO](#)

[security authorization \(to operate\)](#)

Security Authorization (to Operate)

Security Authorization(to Operate)

### Definition(s):

 The official management decision given by a senior organizational official to authorize operation of an information system and to explicitly accept the risk to organizational operations (including mission, functions, image, or reputation), organizational assets, individuals, other organizations, and the Nation based on the implementation of an agreed-upon set of security controls.

### Source(s):

[CNSSI 4009-2015](#) from [NIST SP 800-53 Rev. 4](#), [NIST SP 800-53A Rev. 1](#), [NIST SP 800-37 Rev. 1](#)

[NIST SP 800-137](#) under Authorization (to operate) from [CNSSI 4009](#)

[NIST SP 800-161](#) under Authorization (to operate) from [NIST SP 800-53 Rev. 4](#)

[NIST SP 800-30 Rev. 1](#) under Authorization (to operate) from [CNSSI 4009](#)

[NIST SP 800-39](#) under Authorization(to operate)

[NIST SP 800-37 Rev. 1](#) [Superseded] under Authorization (to operate)

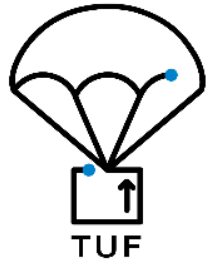
[NIST SP 800-53 Rev. 4](#) [Superseded] under Authorization (to operate)

[NIST SP 800-53 Rev. 4](#) [Superseded] under Authorization(to operate)

持續檢視程式符合性

透過儀錶板檢視執行中狀態

# 有些考量應用程式部署與持續安全的框架



## The Update Framework

A framework for securing software update systems



A framework to secure the integrity of software supply chains

Software supply chain protection

Supply chain compromises are becoming a frequent occurrence. in-toto can help you protect your software supply chain.



# 雲原生安全的四個層級

程式碼安全 (Code security)

容器安全 (Container security)

叢集安全 (Cluster security)

雲端或平台安全 (Cloud security)

# 微服務應用系統的安全策略 (NIST SP 800-204) — 身分與存取管理

## MS-SS1 身分鑑別

- 對於敏感資料的存取不要只使用 API 金鑰鑑別
- 每個身分識別符應該要有獨立 ID，並且要被使用簽章或 HMAC 加密
- 每把 API 金鑰只能限定給一個應用程式使用
- 每把 API 金鑰要和應用有相襯的保證等級，並且使用裝置或使用者資訊強化
- 如果要使用無狀態憑證，應該限制有效期間，並且加解密金鑰不可存於函式庫
- 如果使用 OAuth 或 OpenID 等標準，應考慮部署安全

## MS-SS2 存取控制

- 應由存取控制伺服器控管存取控制政策。可由 API 閘道器先做初步控管，再由離服務較近的為閘道器或服務進一步控管。
- 可允許將存取控制暫存，但要做好控制
- 要能夠支援對於個別資源的存取控制
- 要妥善限制存取識別符的可存取範圍
- 應該要有存取控制伺服器集中管制對於各為服務的身分鑑別與存取權限，避免由各服務自己做

# 微服務應用系統的安全策略 (NIST SP 800-204) — 服務發掘機制 (Service Discovery Mechanism)

## MS-SS3 服務註冊設定

- 應該要提供服務註冊的能力
- 服務註冊能力應該要具有一定 QoS 的保證能力
- 應用服務與服務登錄間應該要透過安全連線溝通
- 應該要維持服務註冊服務與服務之間的低耦合性
- 如果要使用第三方登錄服務，應確保登錄服務會對服務狀態進行檢查
- 對於比較大型的為服務應用，應該使用採用分散式之架構

# 微服務應用系統的安全策略 (NIST SP 800-204) — 安全通訊

## MS-SS4 安全通訊

- 客戶端應該直接對服務做呼叫
- 客戶端對於 API 閘道器或是服務與服務間的通訊應該要經過雙向鑑別與加密通道
- 時常需要互動的服務應該要確定相連服務的狀態



# 微服務應用系統的安全策略 (NIST SP 800-204) — 安全監控

## MS-SS5 安全監控

- 安全監控必須要在閘道器與服務等級執行，以偵測不合適的行為，並做出警報與回應
- 可以建立一個集中式的儀表板以掌握各服務的狀態
- 可以建立正常服務的基線，以便發現異常會被入侵的情況

# 微服務應用系統的安全策略 (NIST SP 800-204) — 可獲得性與彈性的改進

## MS-SS6 實作斷路器

- 透過斷路器代理人，以限制對於某元件的信賴

## MS-SS7 實作負載平衡器

- 負載平衡器功能應該與個別服務分開
- 應該要保護負載平衡器與微服務平台的連線
- 可以對微服務的健康狀態進行檢查

## MS-SS8 流量限制

- 依照骨幹與應用程式的限制去限制流量
- 限制應該依照已知的 API 使用規畫
- 要能夠偵測重送攻擊

# 微服務應用系統的安全策略 (NIST SP 800-204) — 正確性確保 (Integrity Assurance)

## MS-SS9 新版本微服務的導入

- 所有不同版本的服務要求都應該要經由集中的機制，以便進行區分，並在控制的情況下進行轉換
- 對既有版本的監控，以便決定是否要加速進行對於新版本的轉換
- 使用效能與功能正確性，作為決定要調整至新版本的比率
- 應該要考量客戶對於新版本或既有版本的偏好來進行發佈

## MS-SS10 處理連線的持續

- 客戶的連線資訊應該要被安全的儲存
- 避免可以由架構資訊得知綁定的伺服器資訊 (例如 Session ID 裡面放伺服器版本)
- 內部的授權識別符不應該要被發還給使用者，而可以被閘道器攔截

# 微服務應用系統的安全策略 (NIST SP 800-204) — 抵抗網路攻擊(Counter Internet-based Attack)

## MS-SS11 避免機密誤用與填充攻擊 (Preventing credential abuse and stuffing attacks)

- 所有機密誤用的防止機制應該要採用離線策略，例如特定地點嘗試登入失敗應該要禁止連線。而也應該要考慮存取識別符的安全性而避免重送攻擊等情況
- 應該要使用憑證填充偵測機制 (撞庫攻擊偵測)，發現可能的攻擊事件並且通知使用者
- 應該做出分散式阻斷服務攻擊的偵測，並在要在系統中斷前做出警報
- 掃描上傳的檔案、容器的記憶體，與檔案系統，以防止惡意程式

# 程式碼安全

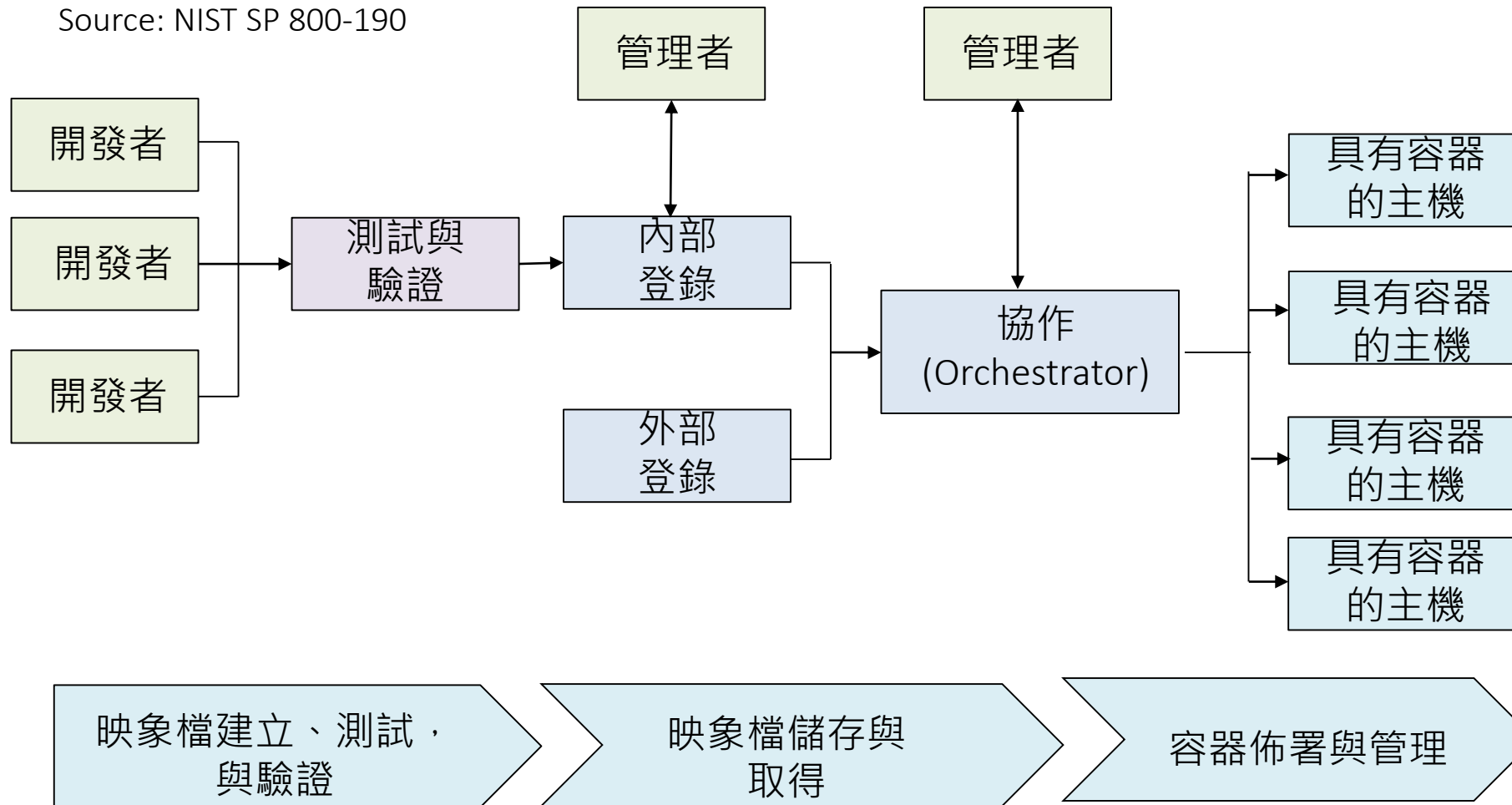
- 安全軟體開發生命週期 (Secure Software Development Lifecycle, S-SDL)
  - 在使用自動化工具之前，可以讓開發團隊學習常見應用程式安全最佳實務
  - 使用既有的安全開發套件 (SDK 或 DevKits)
    - 例如可以在 IDE 工具加掛 Lintian Extension
  - 人工程式碼審查
    - 例如可以在版本控制工具當中，設定 Merge 規則，要求一定要經過審查
    - 可以推動 Security Champions 計畫，去帶動資安意識
- 自動化安全測試
  - 靜態程式碼安全測試 (Static Application Security Test, SAST)
  - 秘密偵測 (Secret Detection)：避免程式當中有些秘密資訊就被存到程式庫當中
  - 依賴程式庫掃描 (Dependency Scan)
  - 動態程式碼安全測試 (Dynamic Application Security Test, DAST)
  - Web API Fuzz Testing

# 容器安全

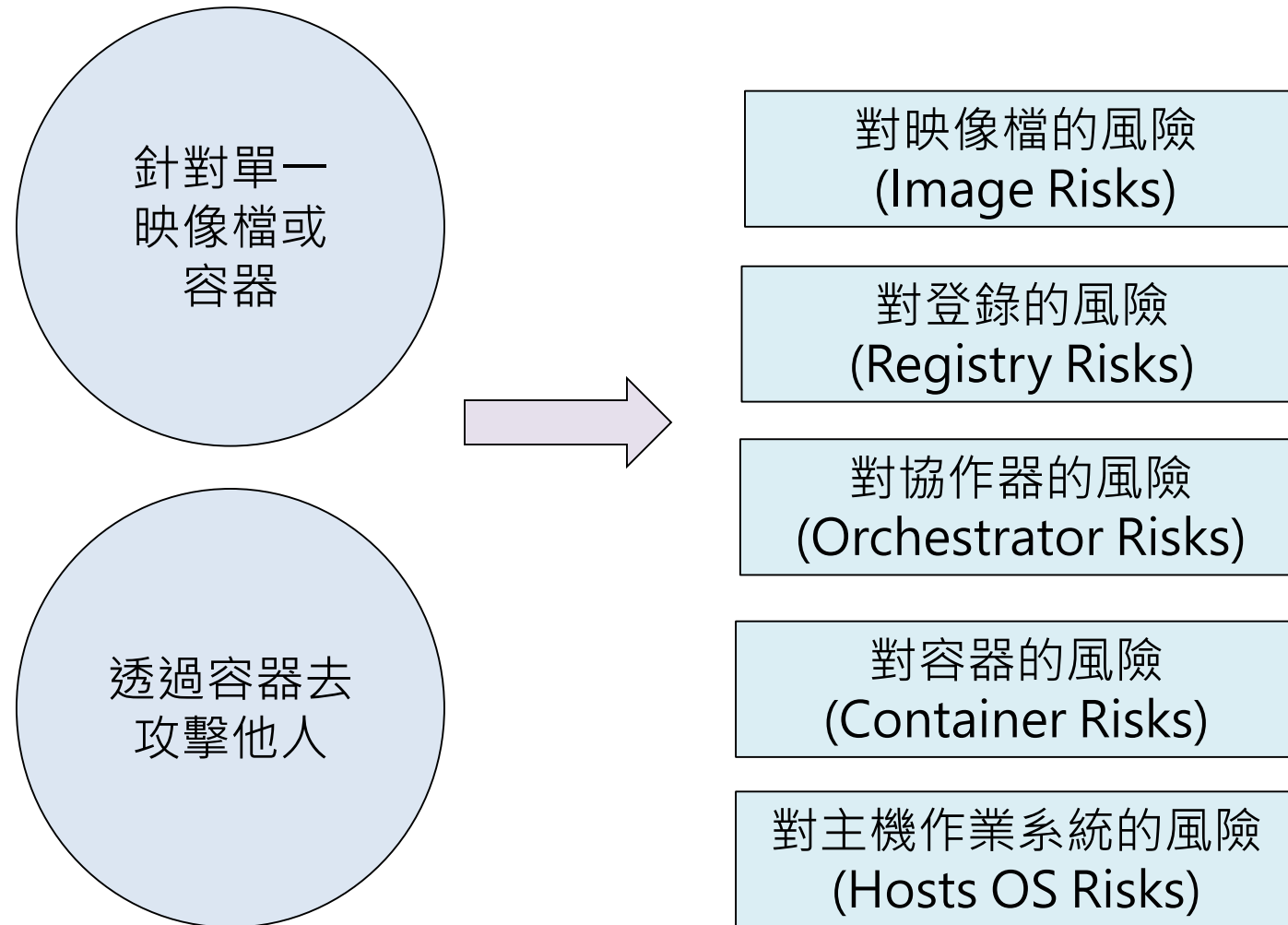
- 主要由三個面向進行考量
  - 容器映像
  - 容器登錄
  - 容器網路安全
    - 避免直接給容器固定 IP 位置而可被連線，而透過妥善的網路設置以確保安全
- 可以先對應 NIST SP 800-190

# NIST SP 800-190 容器技術的架構層次、元件，與生命週期階段

Source: NIST SP 800-190

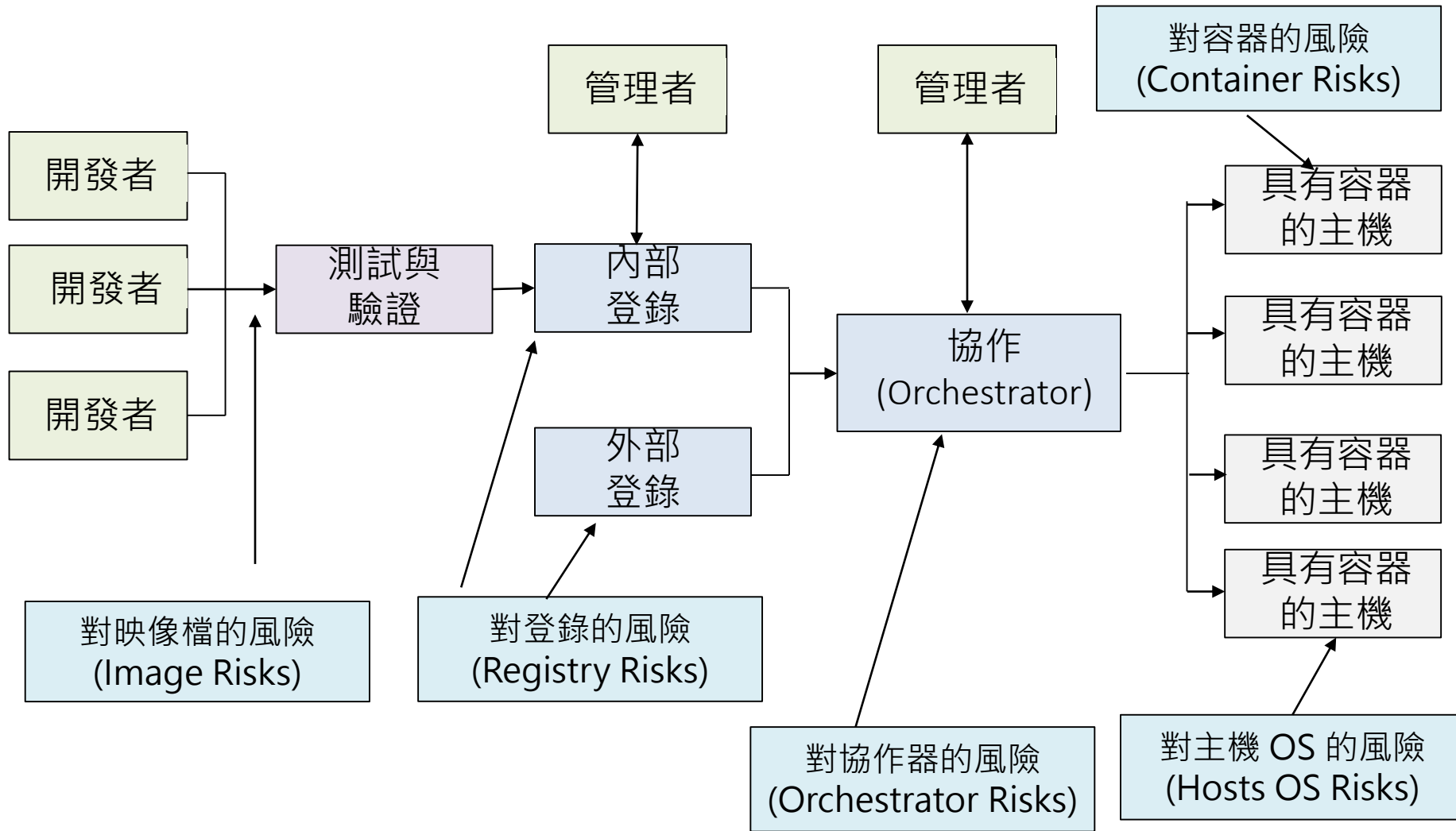


# NIST SP 800-190 主要風險種類





# NIST SP 800-190 風險與生命週期



# NIST SP 800-190 對映像檔的風險與對策

- 映像檔因為未更新而沒修補弱點 (Image Vulnerabilities) :
  - 考量映像檔中各元件的更新狀態
- 映像檔設定缺陷 (Image configuration defects)
  - 檢驗映像檔設定的安全性
- 內有惡意軟體 (Embedded Malware)
  - 對映像檔進行惡意軟體掃描
- 內有明文秘密 (Embedded clear text secrets)
  - 應該將機敏資訊存於映像檔之外
- 使用不可信的映像檔 (Use of Untrusted Image)
  - 建立信賴的映像檔或外部登錄清單

# 對登錄的風險與對策

- 對登錄的不安全連線 (Insecure connections to registries)
  - 確保映像檔的上傳或下載連線有加密
- 登錄當中的過期映像檔 (Stale images in registries)
  - 移除掉登錄當中不安全的映像檔，或是在映像檔名稱當中，加入可以識別版本或最新版本的資訊
- 不充分的身分鑑別與授權限制 (Insufficient authentication and authorization restrictions)
  - 對於存取任何包含敏感或專屬資訊的映像檔的下載應該要經過身分鑑別，並且確保有獲得授權
  - 限制可以將映像檔上傳至登錄的使用者之身分

# NIST SP 800-190 對協作器的風險與對策

- 無限制的管理存取 (Unbounded administrative access)
  - 依照最小權限原則給予每個映像檔的執行權限
- 未經授權的存取 (Unauthorized access)
  - 提升使用協作器的身分鑑別機制之等級
- 容器間網路隔離不良 (Poorly separated inter-container network traffic)
  - 按照敏感度等級去分割網路
- 將不同敏感度等級的工作放在一起跑 (Mixing of workload sensitivity levels)
  - 按照敏感度等級去佈署容器
- 未妥善管理節點間的信賴機制 (Orchestrator node trust)
  - 確保叢集內各節點的安全性

# NIST SP 800-190 對容器的風險與對策

- 執行中軟體的弱點 (Vulnerabilities within the runtime software)
  - 偵測執行中容器的弱點
- 無限制的容器網路存取 (Unbounded network access from containers)
  - 控制容器送出的網路流量
- 不安全的容器執行環境設定 (Insecure container runtime configurations)
  - 自動符合容器設定標準
- 具有有弱點的應用程式 (App vulnerabilities)
  - 偵測容器中應用程式的異常行為
- 異常的容器 (Rogue containers)
  - 區隔開發、測試、線上環境
  - 紀錄容器的操作行為並留下稽核軌跡

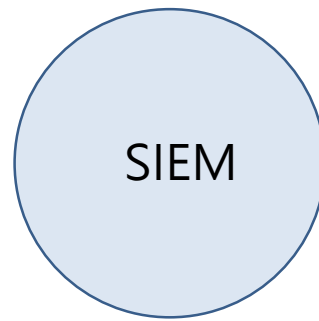
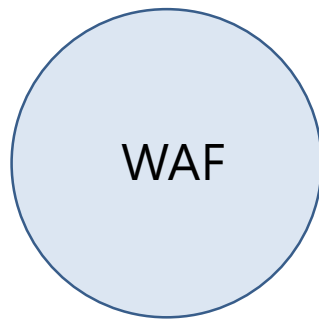
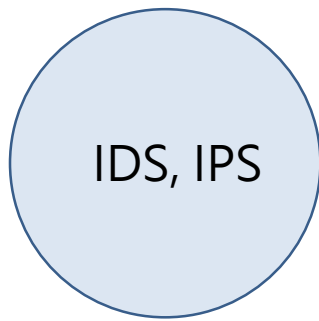
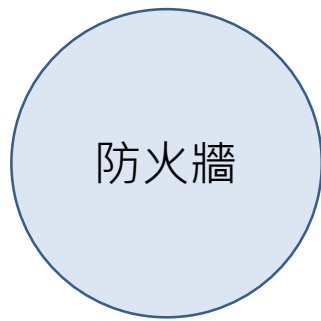
# 容器映像安全

- 容器映像提供，應該要選擇官方的來源
- 使用容器映像檔時可以掃描其安全性
- 建立最小或精簡 (Slim) 的映像檔
- 使用無底層 shell 的 Distroless base 映像
- 從源頭開始建立容器 (Scratch Containers)
- 在容器中不使用特權帳號 (Rootless Containers)
- 如果程式有經過中間層 (Middle Layer)的元件，留意其安全性與可能弱點
- 不要都包在一起，而可建立多階段的建構
- 使用 secret 去儲存資料而避免資料外洩
- 使用 Dockerfile 語法檢查 (Dockerfile linter)
- 對 Dockerfile 進行保護

# NIST SP 800-190 對主機作業系統的風險與對策

- 過大的攻擊面 (Large attack surface)
  - 使用特別針對容器機制的作業系統，或是按照最佳實務 (如 NIST SP 800-123) 去對作業系統做設定
- 共享的核心 (Shared kernel)
  - 區隔容器與非容器程序的執行
- 主機作業系統元件弱點 (Host OS component vulnerabilities)
  - 管理主機作業系統中執行軟體元件的版本
- 不合適的存取權限 (Improper user access rights)
  - 對對作業系統的存取做鑑別，以區分是來自使用者或是容器的操作
- 主機檔案系統的篡改 (Host OS file system tampering)
  - 確保容器只能被給予最小所需的檔案存取權限

# 雲端或平台安全須更進一步考慮到虛擬化的安全元件



.....



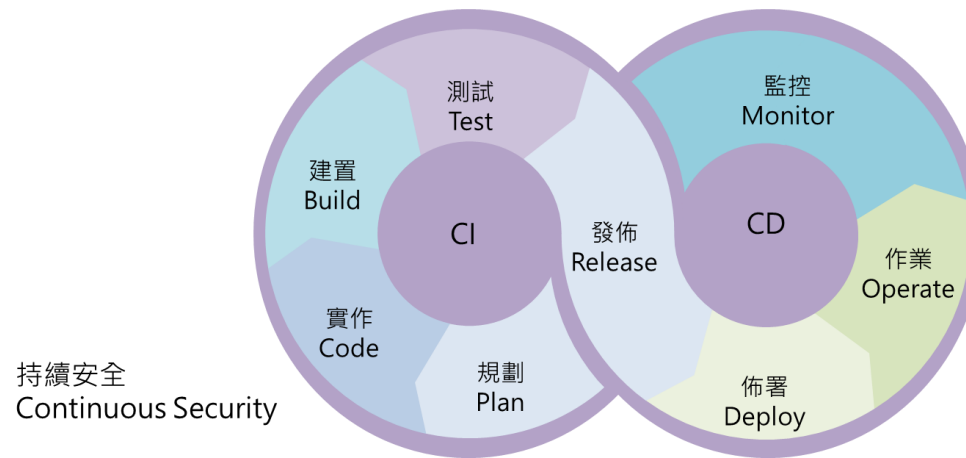


# 對於 DevSecOps 的常見誤解

- 誤解只要所有的程式可以可以直接搬到微服務或採用容器架構：如果不去針對程式去進行重構，或是沒考慮到運行時的骨幹安全議題，不見會更好。
- 誤解 DevSecOps 只考慮應用系統安全：應用程式安全只是其中一部分，要在 CI/CD 的管道中做到持續安全，還需要考慮到骨幹安全
- 誤解 DevSecOps 只是合規即代碼 (Compliance as Code)：合規及代碼將法規或符合性要求以設定呈現，並用 IaC 的方式帶入。這沒有在 CI/CD 的管道中帶入自動保護機制，也沒有考慮到開發者與作業人員的文化改變
- 誤解 DevSecOps 只考慮雲原生安全：不指雲原生，放到雲端服務也適用
- 誤解 DevSecOps 只是在 DevOps 的工具上加入安全考量：DevSecOps 尋求以更科學的方式，建立安全模型

# 結論

- 如果 DevOps 是整合 CI 與 CD，則 DevSecOps 可以說是要做到持續安全 (Continuous Security)
- DevSecOps 強調能夠將資訊安全，整合入一體化的系統開發、部署，與維運的生命週期中，目標是能夠透過持續安全的程序，來確保能夠快速交付安全的軟體
- 雖然技術可以幫助我們更有效率地去控管安全風險，但基本上需要透過各種不同面向的方法：
  - 從文化上，就要建立重視資訊安全的文化，並且告知系統開發與維護的相關人員，如何一同去落實資訊安全
  - 可以建立本身在系統的 CI/CD 程序，並且在各階段納入資訊安全的考量
    - DevSecOps 更強化上線後的持續安全
  - 在應用程式的角度，需要從程式碼安全、容器安全、叢集安全、雲端安全的角度，去落實資訊安全



# 感謝各位的聆聽

