



多奇·教育訓練

# 容器應用的資安掃描與部署

徹底掌握最新的容器技術

多奇數位創意有限公司

技術總監 黃保翕 (Will 保哥)

<https://blog.miniasp.com>





Security Threats for Container Applications

容器應用的資安威脅

# 威脅概述

- 容器應用的常見資安威脅
  - 內部威脅
    - 來自企業內部的惡意或無意行為
  - 外部威脅
    - 外部攻擊者利用系統漏洞或配置錯誤發動攻擊
- 容器應用容易受到攻擊的原因
  - 容器化技術的普及性增加了攻擊面
  - 配置複雜性導致容易出現錯誤
  - 使用未經審查的第三方容器映象

# 典型攻擊方式

- 容器逃逸 (Container Escape)
- 供應鏈投毒 (Supply Chain Poisoning)
- 供應鏈劫持 (Supply Chain Hijacking)
- 惡意容器 (Malicious Containers)

# 容器逃逸 (Container Escape)

- 攻擊概述

- 攻擊者利用容器逃逸漏洞，突破容器隔離，取得宿主機的控制權

- 常見漏洞

- 容器引擎漏洞：如 Docker、runc 中的漏洞
- 配置錯誤：如使用不安全的磁碟掛載選項（-v）

- 防範措施

- 定期更新容器引擎與基礎設施
- 使用最小權限原則進行配置
- 實施嚴格的安全策略與監控

# 供應鏈投毒 (Supply Chain Poisoning)

- 攻擊概述

- 攻擊者通過供應鏈中的弱點，注入惡意程式碼或不安全的依賴項

- 常見漏洞

- 使用未經驗證的第三方容器映象
- 供應鏈中的其他軟體組件的安全漏洞

- 防範措施

- 使用官方和受信任的容器映象
- 定期掃描依賴項並進行更新
- 實施供應鏈安全策略

# 供應鍊劫持 (Supply Chain Hijacking)

- 攻擊概述

- 攻擊者通過控制供應鍊中的一個節點來劫持整個流程

- 常見漏洞

- 許可證 (License) 管理不當
- 供應鍊中的薄弱環節未經保護

- 防範措施

- 實施完整的供應鍊審計與監控
- 使用安全的軟體開發生命週期 (SDLC) 流程
- 定期進行供應鍊風險評估

# 惡意容器 (Malicious Containers)

- 攻擊概述

- 攻擊者構建並發布惡意容器，等待受害者下載並運行

- 常見漏洞

- 容器運行時的安全漏洞
- 使用來歷不明的容器映象

- 防範措施

- 嚴格審查並限制容器來源
- 實施容器安全掃描
- 強化運行時防護措施





Case Studies

案例分析

# 案例一：Docker 容器逃逸漏洞

- 攻擊過程描述
  - 攻擊者利用 Docker 的逃逸漏洞，突破容器的隔離層，獲得宿主機的控制權。
  - 通過在容器內執行惡意代碼，進一步獲取敏感信息或對系統進行破壞。
- 漏洞成因與影響分析
  - 漏洞成因：
    - 容器引擎中的安全漏洞。
    - 容器配置不當，如未設置適當的權限和隔離機制。
  - 影響分析：
    - 宿主機被攻擊者控制，可能導致數據泄露、系統損壞。
    - 容器化應用的信任被破壞，影響業務運營。

# 案例一：修復措施與最佳實踐

- 修復措施：
  - 立即更新 Docker 引擎和相關元件至最新版本。
  - 檢查並修正容器配置，確保使用最小權限原則。
- 最佳實踐：
  - 定期進行安全更新，保持軟體最新。
  - 實施嚴格的安全配置，如使用 SELinux 或 AppArmor 進行容器隔離。
  - 持續監控容器運行狀態，及時發現異常行為。

## 案例二：供應鏈攻擊導致的大規模數據泄露

### • 攻擊過程描述

- 攻擊者通過供應鏈中的一個節點進行攻擊，例如控制一個依賴項的更新，將惡意代碼注入其中。
- 當開發者更新依賴項時，惡意代碼被引入系統，並最終導致數據泄露。

### • 漏洞成因與影響分析

- 漏洞成因：
  - 使用未經驗證的第三方依賴項。
  - 缺乏對供應鏈的有效監控和審計。
- 影響分析：
  - 大量敏感數據被攻擊者竊取，造成企業聲譽損失。
  - 企業可能面臨法律責任和經濟損失。

## 案例二：修復措施與最佳實踐

- **修復措施**

- 立即識別並隔離受影響的系統，防止進一步數據泄露。
- 更新所有依賴項，移除惡意代碼。

- **最佳實踐**

- 實施嚴格的供應鏈安全策略，僅使用受信任的依賴項。
- 定期進行依賴項審計和更新，確保其安全性。
- 建立多層防護機制，如代碼簽名和完整性檢查。



Identifying and Preventing Container Threats

識別並防範容器威脅

# 識別威脅的最佳做法

- 使用自動化工具進行持續監控與掃描

- 使用工具如 Trivy、Aqua Security 等進行容器和依賴項的安全掃描。
- 實施持續監控系統，及時發現並響應異常行為。

- 建立應急響應計劃

- 制定並演練應急響應計劃，以便在發生安全事件時能迅速有效地進行處理。

- 定期進行安全審計與風險評估

- 定期審計系統和依賴項，確保其符合安全標準。
- 進行風險評估，識別並減少潛在威脅。

# 防範措施

- **加強員工的安全意識培訓**

- 定期進行安全培訓，提升員工的安全意識和技能。

- **實施多層防護策略**

- 網絡隔離：使用網絡分段和防火牆進行網絡隔離，減少攻擊面。
- 訪問控制：實施嚴格的訪問控制，確保僅授權用戶能夠訪問系統和數據。

- **維持軟體與依賴項的最新狀態**

- 定期更新軟體和依賴項，修補已知漏洞。
- 使用自動化工具進行更新和補丁管理，確保系統安全。



# 常見的容器漏洞掃描工具

- [Docker Scout](#) 增強軟體供應鏈安全的工具
- [Trivy](#) 一套開源的容器安全掃描工具
- [grype](#) 一個用於容器映像和檔案系統的漏洞掃描器
- [Syft](#) 用於從容器映像和檔案系統生成 SBOM  
與 grype 等掃描器一起使用時，對漏洞檢測特別有效
- [clair](#) 一套開源的應用程式容器靜態漏洞分析
- [OpenVAS](#) OpenVAS 是一個全功能的漏洞掃描器，支援容器

# Docker Scout

- [Docker Scout](#) 是一項專門用於提升軟體供應鏈安全性的解決方案，透過分析容器映像檔並產生軟體清單（SBOM），並與持續更新的漏洞數據庫進行比對，以發現安全漏洞。
- Docker Scout 前 3 個 Repository 是免費的
  - <https://scout.docker.com/>
- 快速上手: <https://docs.docker.com/scout/quickstart/>
  - docker scout cves --only-package express
  - docker scout quickview
- 映象檔分析: <https://docs.docker.com/scout/image-analysis/>



Permission Control for Container Applications

容器應用的權限控管

# 權限控管的重要性

- 權限控管的核心概念

- 權限控管是指管理和限制用戶或應用程式對系統資源的存取權限，以保護數據和系統免受未授權的訪問和操作。
- 目標是確保僅有授權的用戶和應用程式能夠存取和修改特定資源，減少潛在的安全風險。

- 權限控管對於資安的影響

- **防止未授權訪問:** 有效的權限控管可以防止攻擊者或內部人員未經授權訪問敏感數據和系統功能。
- **減少內部威脅:** 限制用戶和應用程式的權限，確保每個人只能存取其工作所需的最小資源，降低內部威脅風險。
- **合規性:** 符合各種法律法規和行業標準的要求，確保系統在合規性審計中表現良好。

# Kubernetes RBAC

- Docker 本身不具有「權限管理」的能力
- Kubernetes 由於是容器化的叢集管理架構，所以有 RBAC 能力
- 實踐的方式
  - 設定角色與角色繫結
  - 權限策略設計
    - 最小權限原則: 確保用戶和應用程式僅獲得執行其職責所需的最低權限。
    - 動態權限調整: 根據用戶和環境的變化動態調整權限，以應對臨時需求和變更。
  - 容器內部權限管理
    - 使用 Linux 能力 (Capabilities)
    - 遵循最小特權原則



Designing the Deployment Process for Container Applications

# 容器應用的部署流程設計

# 部署流程概述

- 定義

- 部署流程是指將應用程式從開發環境推送到生產環境的過程，確保應用程式能夠正常運行並滿足預期功能。

- 目標

- 確保應用程式的可靠性和穩定性。
- 提高部署效率，減少手動操作和出錯風險。
- 支援持續交付和快速迭代。

# 部署流程的基本組成部分

1. **建構 (Build):** 將原始碼轉換為可運行的 **Artifact** (如容器映象)。
2. **測試 (Test):** 確認應用程式功能正常且無重大缺陷。
3. **發布 (Release):** 將 **Artifact** 推送到準備部署的環境。
4. **部署 (Deploy):** 將應用程式部署到生產環境。
5. **監控 (Monitor):** 持續監控應用程式的運行情況，確保其性能和可用性。



# CI/CD 的介紹與實踐

- **CI (Continuous Integration):** 持續整合
  - 指的是頻繁地將程式碼變更整合到主幹並進行自動化測試。
- **CD (Continuous Delivery/Deployment):** 持續交付/部署
  - 指的是將已通過測試的程式碼自動部署到生產環境或準生產環境。

# 部署的最佳實踐

- **零停機部署 (Zero Downtime Deployment)**

- **定義:** 零停機部署是一種部署策略，旨在應用程序升級過程中不影響現有用戶的使用，無需停機。

- **藍綠部署 (Blue-Green Deployment)**

- **簡介:** 在藍綠部署中，兩套幾乎相同的生產環境（藍色和綠色）交替使用，升級新版本至綠色環境後，切換流量到綠色環境，確保零停機。

- **滾動更新 (Rolling Updates)**

- **簡介:** 滾動更新是一種逐步替換應用程序實例的部署方法，確保在任何時刻至少有一部分實例可用，從而實現零停機。

# 零停機部署 (Zero Downtime Deployment)

- 使用 **藍綠部署 (Blue-Green Deployment)** 或 **滾動更新 (Rolling Updates)**。
- 確保新版本與舊版本的兼容性。
- 使用負載均衡器來管理流量切換。

# 藍綠部署 (Blue-Green Deployment)

- 準備綠色環境，部署新版本應用程序。
- 執行測試，確保綠色環境正常運行。
- 切換流量至綠色環境，逐步減少藍色環境的流量。
- 一旦確認綠色環境穩定，將綠色環境作為新的生產環境。

# 滾動更新 (Rolling Updates)

- 部署新版本的部分實例，並將舊版本實例下線。
- 監控新實例的運行情況。
- 一旦確認新實例穩定，逐步增加新版本實例數量，同時減少舊版本實例數量。
- 完成所有實例的替換。

# 持續監控與回滾策略

- 持續監控 (Monitoring)

- 使用監控工具 ( 如 Prometheus 、 Grafana ) 監控應用程序性能和健康狀態。
- 設定警報和指標，以便及時發現和處理異常情況。

- 回滾策略 (Rollback)

- 準備回滾計劃，以便在部署失敗或出現問題時迅速恢復到上一穩定版本。
- 使用自動化工具實現快速回滾，減少停機時間和影響範圍。



# 部署 ASP.NET Core 應用程式至 Docker

# 部署 ASP.NET Core 網站到 Docker

- 官方文件
  - [Host ASP.NET Core in Docker containers](#)
  - [Docker images for ASP.NET Core](#)
  - [Visual Studio Container Tools with ASP.NET Core](#)
  - [Deploy an ASP.NET container to a container registry using Visual Studio](#)
- 容器範例
  - [ASP.NET Core Docker Sample](#)



# 全新的 .NET 容器基礎映象 ( base image )

- [Canonical 與 Microsoft 聯袂推出 .NET Chiseled Ubuntu 容器映像](#)

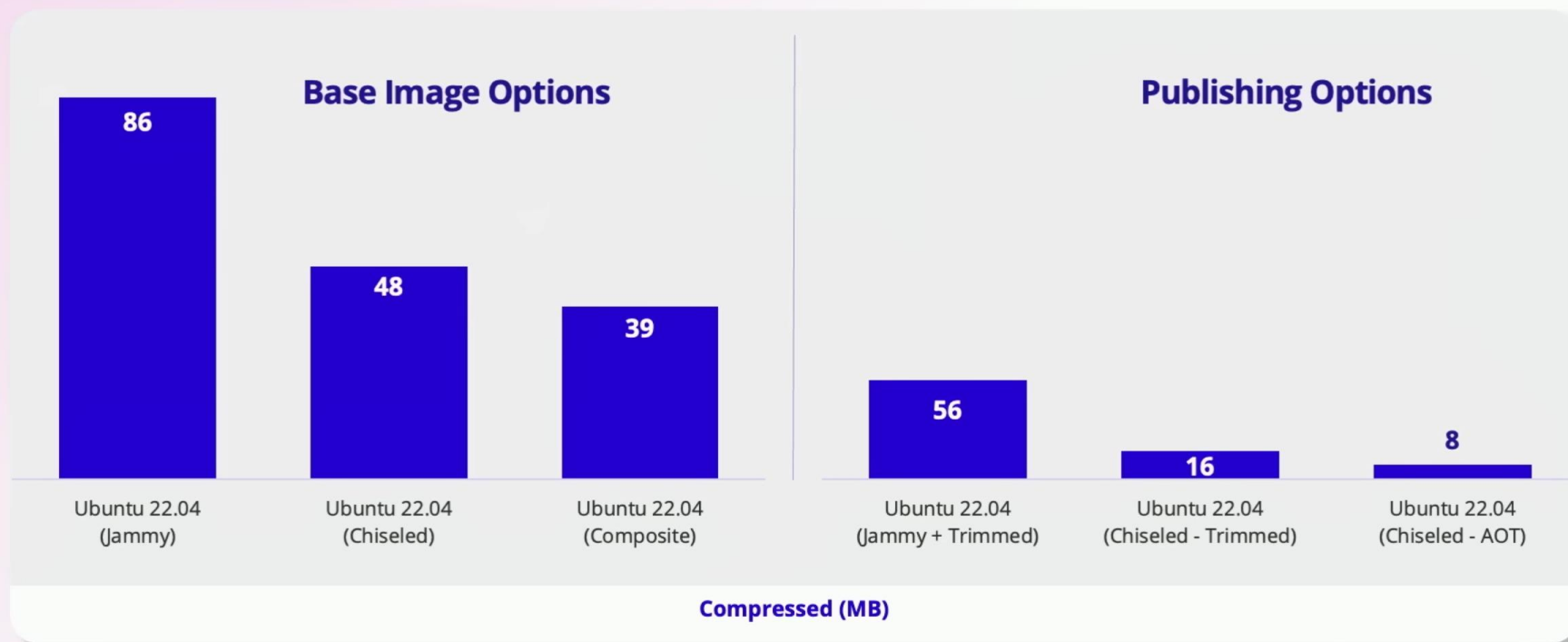
「Chiseled」這個詞在英文中通常用來形容某物或某人的外觀特徵，尤其是與清晰、鮮明的輪廓或線條有關。它最常用來描述臉部特徵，特別是當某人的臉部線條特別鮮明、角度分明時。例如，一個人可能被形容為有「chiseled jawline」，意即他有非常明確和突出的下巴輪廓。

- 相關連結
  - [Canonical announces the general availability of chiselled Ubuntu containers](#)
  - [Jammy Jellyfish Release Notes](#) (Ubuntu 22.04 LTS)
  - [ubuntu/dotnet-runtime - Docker Image | Docker Hub](#)  
(Chiselled Ubuntu for Chiselled .NET)
  - [Announcement: New approach for differentiating .NET 8+ images #4821](#)

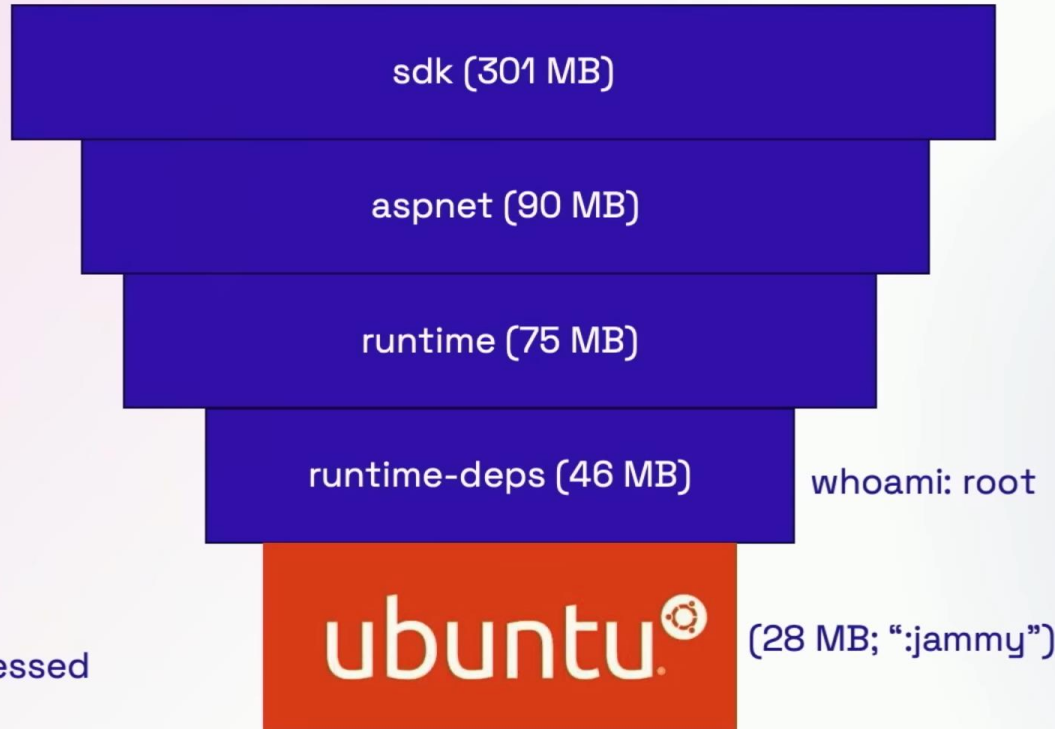
# 何謂 Chiselled Ubuntu Containers

- 為了打造**極致安全與生產環境**使用的 Containers
- 主要特色
  - **容器超小** (沒有文件、腳本、設定、程式碼、header files、相依套件)
  - **超級安全** (non-root、no shell、no package manager)
  - **容器合規** (最小化相依性、容易通過稽核)
  - **隨需組合** (需要的時候還是有工具可以加裝其他內容)
  - **套件相容** (glibc 與 musl libc)
  - **長期支援** (隨著 LTS 支援週期到 April 2027 截止)

# Container Size Improvements



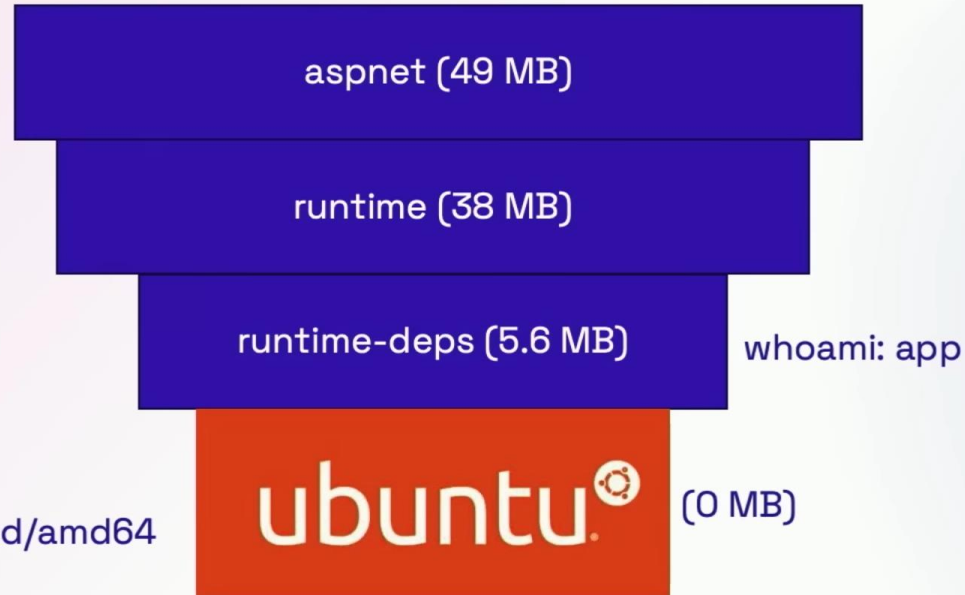
# .NET Containers (in general)



## Notes:

- Sizes are compressed
- Running sum

# .NET Containers (Chiseled)



## Notes:

- Sizes are compressed/amd64
- Running sum

# .NET Containers (Chiseled + .NET AOT)

releasesapi app (18.6 MB)

runtime-deps:aot (5.6 MB)

runtime-deps:extra (21 MB)

whoami: app

ubuntu (0 MB)

## Notes:

- Sizes are compressed/amd64
- Running sum
- “extra” contains ICU and tzdata

.NET

# 全新的 ASP.NET Core 8 容器體驗

- 初始化專案

- dotnet new web -n hello-web
- cd hello-web

[.NET 8 container workshop](#)

- 加入容器套件

- dotnet add package Microsoft.NET.Build.Containers

- 發行容器映像

- dotnet publish -t:PublishContainer
- docker run --rm -p 8080:8080 hello-web
- <http://localhost:8080/>

# 完全不需要寫 Dockerfile 就能發行容器

- 預設會自動選取正確的 .NET 基底容器映像

```
TERMINAL  PROBLEMS  OUTPUT  PORTS  2  DEBUG CONSOLE  bash - hello-dotnet  +  -  ...  ^

• will@WILLSUPERPC:~/hello-dotnet$ dotnet publish /t:PublishContainer
MSBuild version 17.8.3+195e7f5a3 for .NET
Determining projects to restore...
All projects are up-to-date for restore.
hello-dotnet -> /home/will/hello-dotnet/bin/Release/net8.0/hello-dotnet.dll
hello-dotnet -> /home/will/hello-dotnet/bin/Release/net8.0/publish/
Building image 'hello-dotnet' with tags 'latest' on top of base image 'mcr.microsoft.com/dotnet/runtime:8.0'.
Pushed image 'hello-dotnet:latest' to local registry via 'docker'.

• will@WILLSUPERPC:~/hello-dotnet$ dotnet run --rm hello-dotnet
Hello, Ubuntu 22.04.3 LTS on X64!

• will@WILLSUPERPC:~/hello-dotnet$ docker run -it --rm --entrypoint /bin/sh hello-dotnet -c "whoami"
app

• will@WILLSUPERPC:~/hello-dotnet$ docker image list hello-dotnet:latest
REPOSITORY      TAG       IMAGE ID       CREATED        SIZE
hello-dotnet    latest    6538b31868eb   About a minute ago   193MB

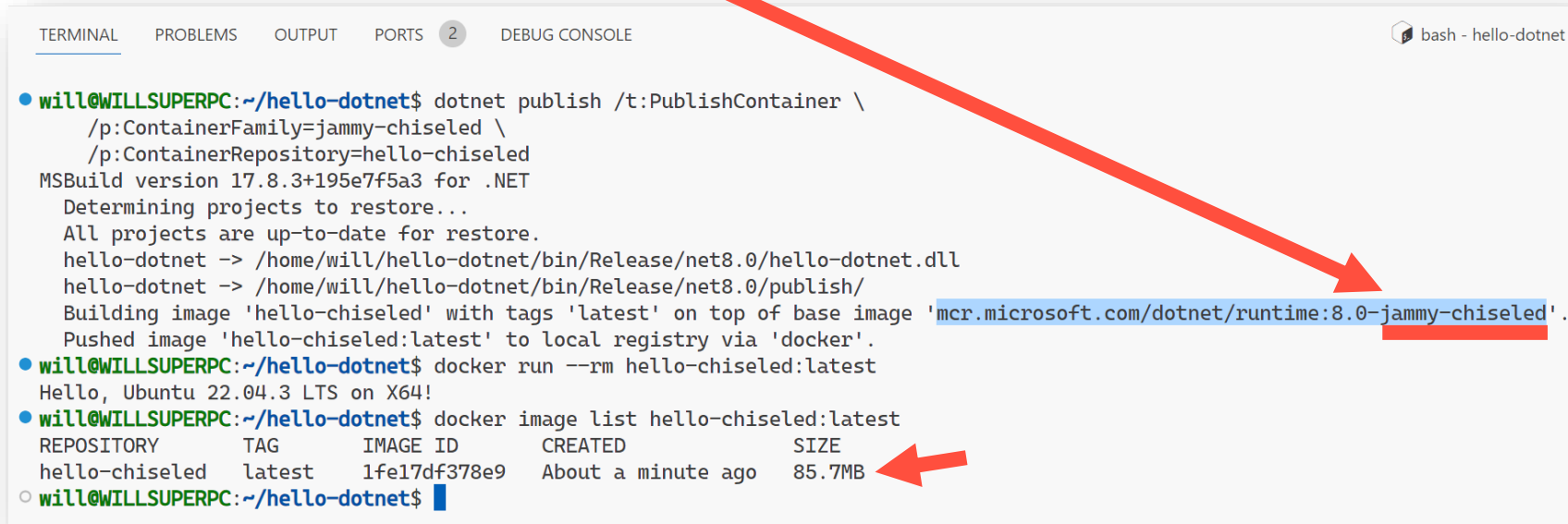
○ will@WILLSUPERPC:~/hello-dotnet$
```



# 改用 Ubuntu chiseled images 來建置

- 建置命令

```
dotnet publish /t:PublishContainer \  
  /p:ContainerFamily=jammy-chiseled /p:ContainerRepository=hello-chiseled
```



TERMINAL PROBLEMS OUTPUT PORTS 2 DEBUG CONSOLE bash - hello-dotnet

```
• will@WILLSUPERPC:~/hello-dotnet$ dotnet publish /t:PublishContainer \  
  /p:ContainerFamily=jammy-chiseled \  
  /p:ContainerRepository=hello-chiseled  
MSBuild version 17.8.3+195e7f5a3 for .NET  
Determining projects to restore...  
All projects are up-to-date for restore.  
hello-dotnet -> /home/will/hello-dotnet/bin/Release/net8.0/hello-dotnet.dll  
hello-dotnet -> /home/will/hello-dotnet/bin/Release/net8.0/publish/  
Building image 'hello-chiseled' with tags 'latest' on top of base image 'mcr.microsoft.com/dotnet/runtime:8.0-jammy-chiseled'.  
Pushed image 'hello-chiseled:latest' to local registry via 'docker'.  
• will@WILLSUPERPC:~/hello-dotnet$ docker run --rm hello-chiseled:latest  
Hello, Ubuntu 22.04.3 LTS on X64!  
• will@WILLSUPERPC:~/hello-dotnet$ docker image list hello-chiseled:latest  
REPOSITORY          TAG          IMAGE ID          CREATED           SIZE  
hello-chiseled      latest      1fe17df378e9     About a minute ago  85.7MB  
○ will@WILLSUPERPC:~/hello-dotnet$
```

# jammy-chiseled 的 base image 沒有 shell

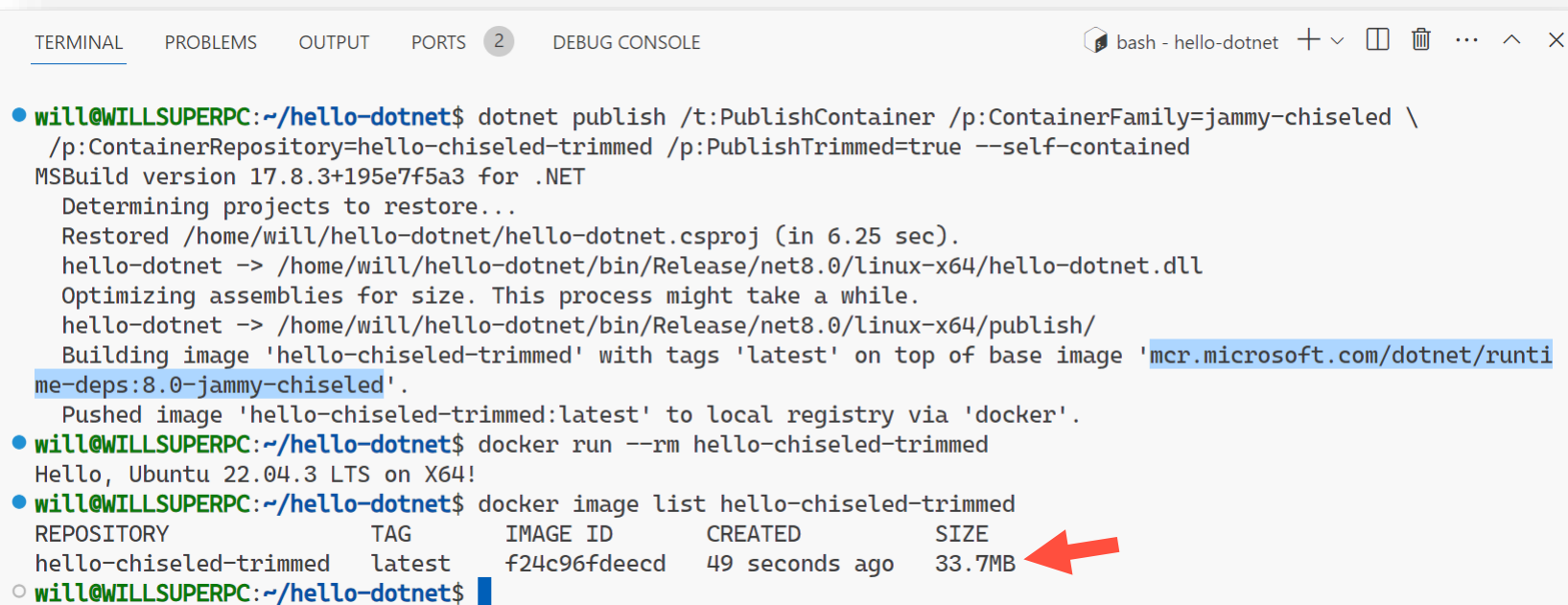
TERMINAL PROBLEMS OUTPUT PORTS 2 DEBUG CONSOLE bash - hello-dotnet

```
will@WILLSUPERPC:~/hello-dotnet$ docker run --rm --entrypoint /bin/sh hello-dotnet -c "cat /etc/os-release | head -n 1"
PRETTY_NAME="Debian GNU/Linux 12 (bookworm)"
will@WILLSUPERPC:~/hello-dotnet$ docker run --rm --entrypoint /bin/sh hello-chiseled -c "cat /etc/os-release | head -n 1"
docker: Error response from daemon: failed to create task for container: failed to create shim task: OCI runtime create failed: runc create failed: unable to start container process: exec: "/bin/sh": stat /bin/sh: no such file or directory: unknown.
will@WILLSUPERPC:~/hello-dotnet$
```

# 使用 Self-contained 部署

- 建置命令

```
dotnet publish /t:PublishContainer /p:ContainerFamily=jammy-chiseled\  
/p:ContainerRepository=hello-chiseled-trimmed /p:PublishTrimmed=true --self-  
contained
```

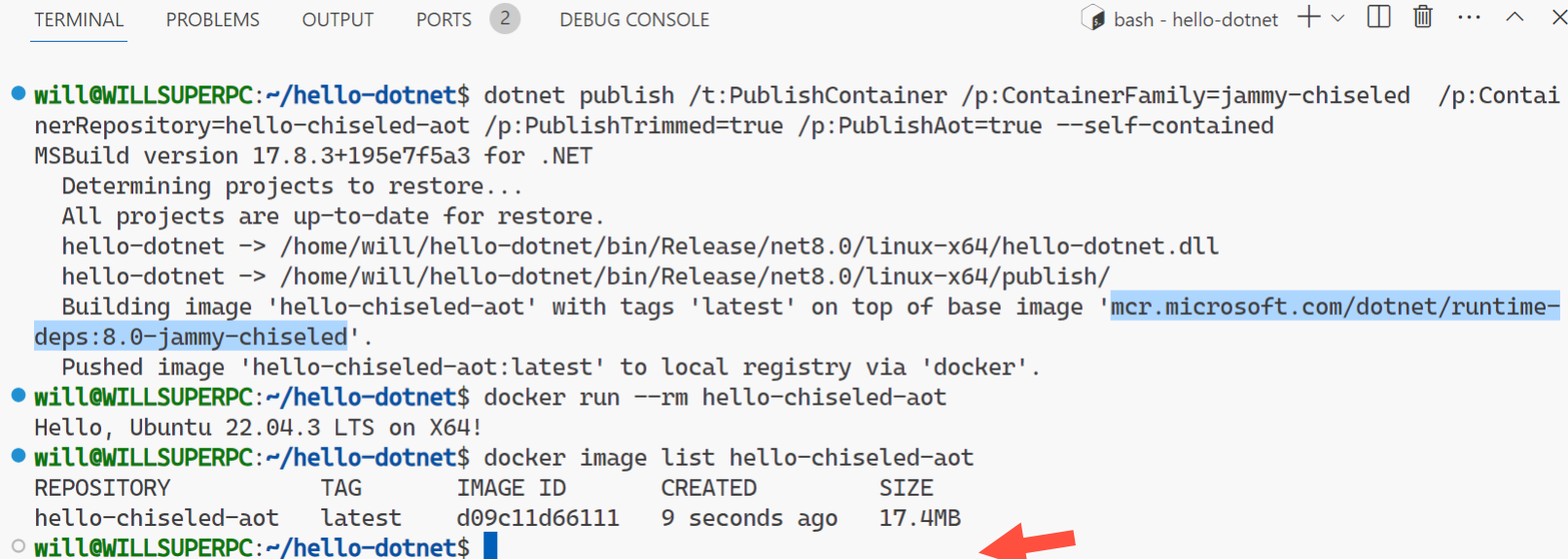


```
TERMINAL  PROBLEMS  OUTPUT  PORTS  2  DEBUG CONSOLE  bash - hello-dotnet  +  -  [ ]  [X]  ...  ^  X
```

```
• will@WILLSUPERPC:~/hello-dotnet$ dotnet publish /t:PublishContainer /p:ContainerFamily=jammy-chiseled \  
/p:ContainerRepository=hello-chiseled-trimmed /p:PublishTrimmed=true --self-contained  
MSBuild version 17.8.3+195e7f5a3 for .NET  
Determining projects to restore...  
Restored /home/will/hello-dotnet/hello-dotnet.csproj (in 6.25 sec).  
hello-dotnet -> /home/will/hello-dotnet/bin/Release/net8.0/linux-x64/hello-dotnet.dll  
Optimizing assemblies for size. This process might take a while.  
hello-dotnet -> /home/will/hello-dotnet/bin/Release/net8.0/linux-x64/publish/  
Building image 'hello-chiseled-trimmed' with tags 'latest' on top of base image 'mcr.microsoft.com/dotnet/runti  
me-deps:8.0-jammy-chiseled'.  
Pushed image 'hello-chiseled-trimmed:latest' to local registry via 'docker'.  
• will@WILLSUPERPC:~/hello-dotnet$ docker run --rm hello-chiseled-trimmed  
Hello, Ubuntu 22.04.3 LTS on X64!  
• will@WILLSUPERPC:~/hello-dotnet$ docker image list hello-chiseled-trimmed  
REPOSITORY          TAG          IMAGE ID      CREATED      SIZE  
hello-chiseled-trimmed  latest      f24c96fdeecd  49 seconds ago  33.7MB  
○ will@WILLSUPERPC:~/hello-dotnet$
```

# 使用 native AOT 部署

```
dotnet publish /t:PublishContainer /p:ContainerFamily=jammy-chiseled \  
/p:ContainerRepository=hello-chiseled-aot /p:PublishTrimmed=true \  
/p:PublishAot=true --self-contained
```

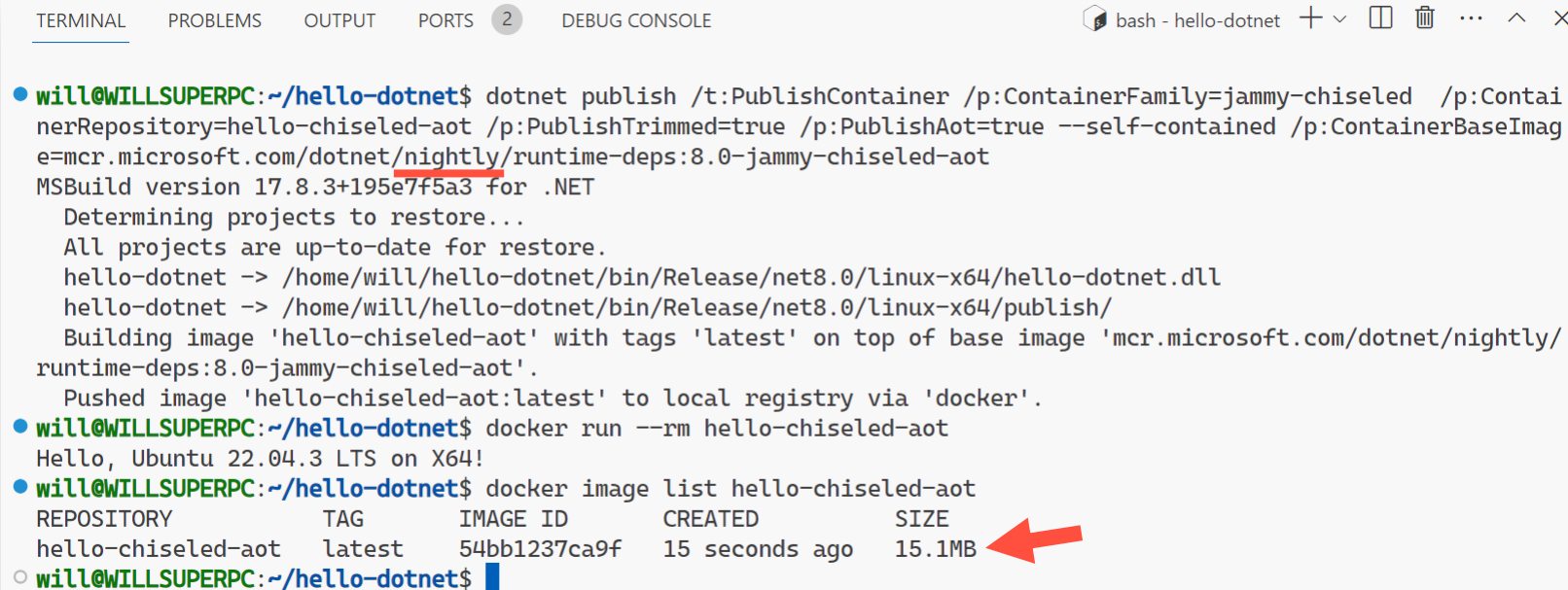


TERMINAL PROBLEMS OUTPUT PORTS 2 DEBUG CONSOLE bash - hello-dotnet

```
● will@WILLSUPERPC:~/hello-dotnet$ dotnet publish /t:PublishContainer /p:ContainerFamily=jammy-chiseled /p:ContainerRepository=hello-chiseled-aot /p:PublishTrimmed=true /p:PublishAot=true --self-contained  
MSBuild version 17.8.3+195e7f5a3 for .NET  
Determining projects to restore...  
All projects are up-to-date for restore.  
hello-dotnet -> /home/will/hello-dotnet/bin/Release/net8.0/linux-x64/hello-dotnet.dll  
hello-dotnet -> /home/will/hello-dotnet/bin/Release/net8.0/linux-x64/publish/  
Building image 'hello-chiseled-aot' with tags 'latest' on top of base image 'mcr.microsoft.com/dotnet/runtime-deps:8.0-jammy-chiseled'.  
Pushed image 'hello-chiseled-aot:latest' to local registry via 'docker'.  
● will@WILLSUPERPC:~/hello-dotnet$ docker run --rm hello-chiseled-aot  
Hello, Ubuntu 22.04.3 LTS on X64!  
● will@WILLSUPERPC:~/hello-dotnet$ docker image list hello-chiseled-aot  
REPOSITORY          TAG          IMAGE ID      CREATED        SIZE  
hello-chiseled-aot  latest      d09c11d66111  9 seconds ago  17.4MB  
○ will@WILLSUPERPC:~/hello-dotnet$
```

# 實驗性的 native AOT base image

```
dotnet publish /t:PublishContainer --self-contained /p:ContainerFamily=jammy-chiseled \
/p:ContainerRepository=hello-chiseled-aot /p:PublishTrimmed=true /p:PublishAot=true \
/p:ContainerBaseImage=mcr.microsoft.com/dotnet/nightly/runtime-deps:8.0-jammy-chiseled-aot
```



TERMINAL PROBLEMS OUTPUT PORTS 2 DEBUG CONSOLE bash - hello-dotnet

```
● will@WILLSUPERPC:~/hello-dotnet$ dotnet publish /t:PublishContainer /p:ContainerFamily=jammy-chiseled /p:ContainerRepository=hello-chiseled-aot /p:PublishTrimmed=true /p:PublishAot=true --self-contained /p:ContainerBaseImage=mcr.microsoft.com/dotnet/nightly/runtime-deps:8.0-jammy-chiseled-aot
MSBuild version 17.8.3+195e7f5a3 for .NET
Determining projects to restore...
All projects are up-to-date for restore.
hello-dotnet -> /home/will/hello-dotnet/bin/Release/net8.0/linux-x64/hello-dotnet.dll
hello-dotnet -> /home/will/hello-dotnet/bin/Release/net8.0/linux-x64/publish/
Building image 'hello-chiseled-aot' with tags 'latest' on top of base image 'mcr.microsoft.com/dotnet/nightly/runtime-deps:8.0-jammy-chiseled-aot'.
Pushed image 'hello-chiseled-aot:latest' to local registry via 'docker'.
● will@WILLSUPERPC:~/hello-dotnet$ docker run --rm hello-chiseled-aot
Hello, Ubuntu 22.04.3 LTS on X64!
● will@WILLSUPERPC:~/hello-dotnet$ docker image list hello-chiseled-aot
REPOSITORY          TAG          IMAGE ID      CREATED        SIZE
hello-chiseled-aot  latest      54bb1237ca9f  15 seconds ago  15.1MB
○ will@WILLSUPERPC:~/hello-dotnet$
```

# 分析 jammy-chiseled 的容器內容

- 使用 **anchore/syft** 容器進行分析

```
$ docker run --rm anchore/syft mcr.microsoft.com/dotnet/runtime:8.0-jammy-chiseled | grep dotnet | wc -l
168
$ docker run --rm anchore/syft mcr.microsoft.com/dotnet/runtime:8.0-jammy-chiseled | grep deb | wc -l
7
$ docker run --rm anchore/syft mcr.microsoft.com/dotnet/runtime:8.0-jammy-chiseled | grep deb
base-files                12ubuntu4.4          deb
ca-certificates           20230311ubuntu0.22.04.1 deb
libc6                     2.35-0ubuntu3.4      deb
libgcc-s1                 12.3.0-1ubuntu1~22.04 deb
libssl3                   3.0.2-0ubuntu1.10    deb
libstdc++6                12.3.0-1ubuntu1~22.04 deb
zlib1g                    1:1.2.11.dfsg-2ubuntu9.2 deb
```

# ASP.NET Core 8 的 base images

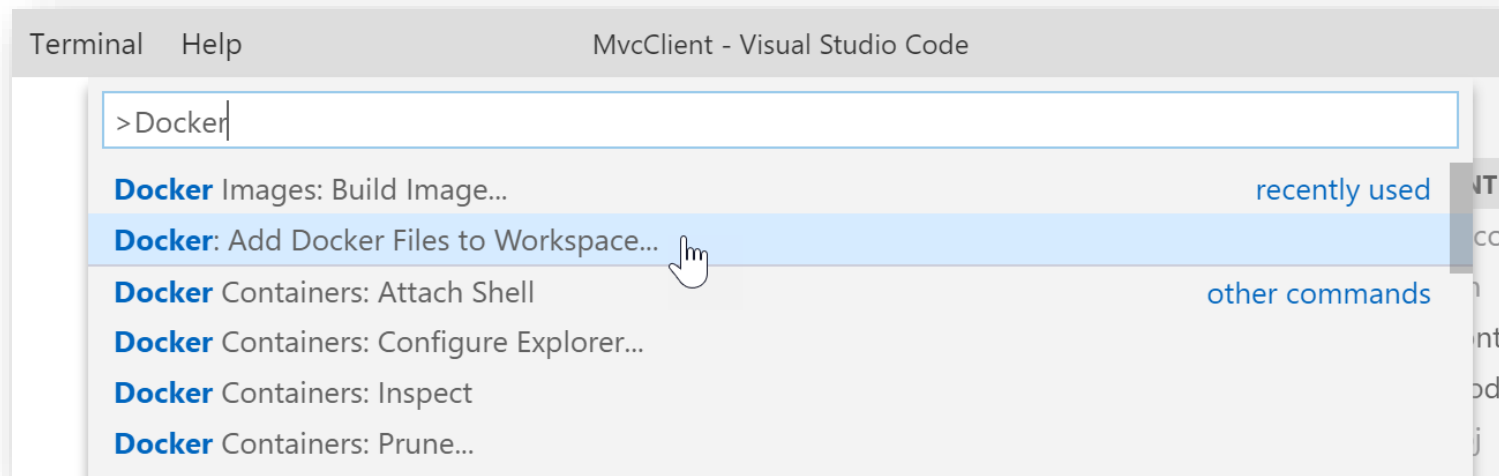
```
docker images --format "table  
{{.Repository}}\t{{.Tag}}\t{{.Size}}" | grep  
mcr.microsoft.com/dotnet/aspnet
```

|                                 |                    |       |
|---------------------------------|--------------------|-------|
| mcr.microsoft.com/dotnet/aspnet | 8.0-jammy          | 216MB |
| mcr.microsoft.com/dotnet/aspnet | 8.0-alpine         | 107MB |
| mcr.microsoft.com/dotnet/aspnet | 8.0                | 217MB |
| mcr.microsoft.com/dotnet/aspnet | 8.0-jammy-chiseled | 110MB |

- 注意：這些 **Size** 都是「未壓縮」的大小，實際上會小很多！

# 使用 VSCode Docker 擴充套件

- F1 > 搜尋 Docker > **Docker: Add Docker Files to Workspace**
- 該擴充套件會自動分析專案內容，自動產生必要的 Dockerfile







# 聯絡資訊

The Will Will Web

網路世界的學習心得與技術分享

<http://blog.miniasp.com/>

Facebook

Will 保哥的技术交流中心

<http://www.facebook.com/will.fans>

Twitter

[https://twitter.com/Will\\_Huang](https://twitter.com/Will_Huang)



多奇·教育訓練

**THANK YOU!**

Q&A