Dockerfile: Hardening e boas práticas

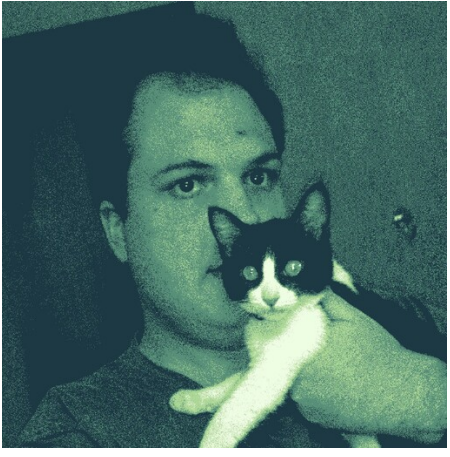
Caio Volpato (caioau.net)

Sorocoded 2a edição

16 Maio 2026

$ whoami

- Graduação em matemática aplicada e computacional (Unicamp)
- Atuou uns bons anos na área de dados
- Pós graduação em eng de software
- Nos últimos anos atuo como DevOps
- Certificações Kubernetes: CKA e CKAD
- Projetos:
  - Casa hacker - casahacker.org
  - CryptoRave - cryptorave.org



Texto original

Essa palestra foi originada de um texto do mesmo autor:
Dockerfile: Hardening e boas práticas

Somos um grupo de divulgação cientifica no medium, escrevemos sobre Computação, Matemática, Estatística e Ciência de Dados.

- Link: medium.com/computando-arte
- Fundado em Nov/2020
- A informação quer ser livre: Licenciado sob CC BY-SA 4.0
- /join: Por que e como fazer um blog técnico

Preâmbulo: Leis de Akin de design aero espacial

São no total 45 “leis”, vamos destacar 5:

> 1: Engineering is done with numbers. Analysis without numbers is only an opinion.

> 12: There is never a single right solution. There are always multiple wrong ones, though.

> 31: (Mo’s Law of Evolutionary Development) You can’t get to the moon by climbing successively taller trees.

> 33: (Patton’s Law of Program Planning) A good plan violently executed now is better than a perfect plan next week.

Por fim, a “lei” que define a tônica da nossa empreitada:

> 35: (de Saint-Exupery’s Law of Design) A designer knows that they have achieved perfection not when there is nothing left to add, but when there is nothing left to take away.

Em suma vamos pegar a imagem docker um app python REST hello world e vamos ir tirando pacotes e dependências até chegar o mínimo que necessário para o mesmo rodar.

Akin’s Laws of Spacecraft Design

Exemplo: App Python Rest (flask)

Vamos explorar esse exemplo de uma REST API “hello world”

Dockerfile v0

Nossa primeira interação do Dockerfile

Dockerfile v0 - resultado

Nosso v0 ficou com 570MB[1]

Tá bom pra você? Quase 600MB para um hello world simples.

Mas fica até o final porque vamos chegar em uma imagem 10x menor.

Porém a imagem base do python `python:3.9` tem sozinha 1.1GB

[1] tamanho sem compressão

dive

O dive é uma ferramenta para “dissecar” imagens docker, que permite visualizar camada a camada o que mudou.

github.com/wagoodman/dive

Dockerfile v0 - conclusões

- Logica errada de remoção arquivos de desnecessários
- Segredos vazados

dockerignore

Na mesma linha de um gitignore, o dockerignore controla o que é copiado nas instruções COPY evitando os vazamentos

dockerignore

```
*

!/src
!*.py
!requirements.txt
!entrypoint.sh
**/__pycache__/
**/*.py[cod]
*.so
```

Dockerfile v1

Melhorando o v0 vamos fazer a seguintes mudanças:

- Arrumar a remoção dos arquivos desnecessários
- Usuário não root para segurança
- venv para isolar as bibliotecas python

Dockerfile v1 - resultado

A imagem ficou 50MB menor sem os caches (10%) da anterior

USER nonroot basta?

Não, pois o processo do app rodando como não root, se algum binário que tiver o suid o app pode escalar privilégio.

Em tempo de execução use a opção `no-new-privileges` no “docker run/compose”

Boas práticas: OWASP Docker Security Cheat Sheet

Figura 1: commitstrip - chmod what?

Dockerfile v2

Nossa imagem está com 520MB, os maiores ofensores são os pacotes compiladores, porém em tempo de execução esses pacotes pesados não são necessários.

Edai surge o conceito multi-stage, onde a imagem é construída em estágios separados (múltiplos FROM), fazendo a compilação em um estagio maior e passando os artefatos pra uma imagem de execução menor.

Dockerfile v2 - outras práticas

- Atualizar imagem base para uma distro com suporte ativo
  - endoflife.date
  - dependabot / renovate
- Usar variante “slim” para as imagens runtime

resultado: 130MB (22% de v0)

golang para salvar o dia

golang permite facilmente criar binários compilados estaticamente, ou seja a imagem docker só tem o binário do app

Bora de exemplo?

Realmente só o binário basta?

No nosso caso, nosso app faltou a cadeia de certificados TLS/HTTPS

Como podemos ter apenas a cadeia de certificados, sem outros pacotes e binarios como shell que vem nas imagens bases das distros?

Ai surge as imagens distroless

Imagens distroless

As imagens Imagens distroless
github.com/GoogleContainerTools/distroless

São imagens base que contém o mínimo necessário para rodar apps em diferentes linguagem.

Como debugar imagens distroless

Sem shell e pacotes, como a gente faz o debug dessas imagens?

1. Criar variante -dev com shell e ferramentas de debug, se faltar algum pacote basta fazer um “apt install”
2. Anexar um container com as ferramentas/pacotes no mesmo , a la sidecar.

Sidecar debuging

Utilize --pid container:<nome_container> e --network container:<nome_container>

Por exemplo, utilizando imagem netshoot (com diversas ferramentas de rede)

```
docker run --rm -it --privileged --pid container:golang --network container:golang nicolaka/netshoot
```

Imagens chainguard

- Imagens com pacotes hardenizados e constantemente atualizados para mitigar CVEs
- Baseadas no alpine
- Usa glibc ao invés da musl do alpine
- Geração de SBOM automaticamente (`/var/lib/db/sbom/`)

Documentação e cursos muito bons: edu.chainguard.dev

Desvantagem: No plano gratuito apenas a tag latest está disponível.

Migrando nossa imagem debian-based pro chainguard

- Migrando os pacotes do debian pro alpine
  - build-essential (debian) -> build-base (alpine)
  - Dica: pra procurar o pacote que tem o comando ldd: apk search cmd:ldd
- Shell diferente (bash vs ash)

Fazendo scan de CVEs

Vamos usar o trivy para fazer o scan das imagens (o grype é muito bom tmb)

Resultado (v1): 1334 CVEs (6 são critical, 257 high, 894 medium)

v2 (multistage): 136 CVEs (4 critical, 6 high, 36 medium)

chainguard: 0 CVEs

Scanner de Heráclito

> Nenhum homem entra no mesmo rio duas vezes, pois não é o mesmo rio e ele não é o mesmo homem

> Nenhum scan é o mesmo duas vezes, pois não são as mesmas CVEs ou a imagem não é a mesma

Lição: uma imagem nunca é zero CVEs de vdd, novas CVEs serão descobertas, faça rebuild periodicamente e use renovabot / renovate

Debian: Estamos sendo justos com ele?

O trivy apontou ~1300 CVEs pra nossa imagem debian based

Estamos de fato em risco?

O debian “congela” as versões dos pacotes e aplica apenas patches em cima, os scanners não reconhecem a versão customizada do debian e reportam um falso positivo

O grype tem uma opção pra mostrar as vulnerabilidades que ainda serão endereçadas `--only-fixed` filtrando esses falso positivos.

Imagens chainguard distroless com melange

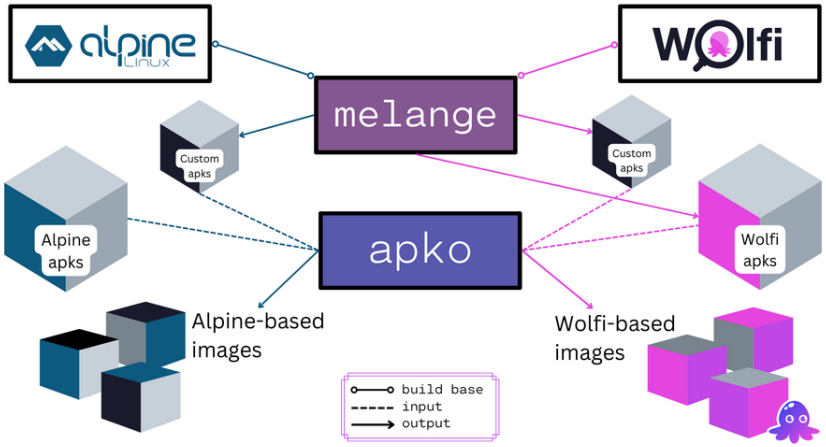


Figura 2: Sistema de build melange

Sistemas de build

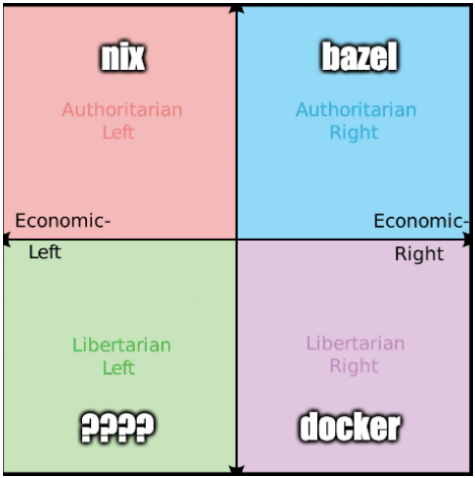


Figura 3: Bússola política dos sistemas de build

imagens chainguard distroless apenas Dockerfile

Aprender todo um novo sistema de build é complicado, rola obter um resultado semelhante usando somente Dockerfile?

Comparando resultados

Versão	Tamanho [MB]	Percentual de v0
python:3.9	1100	193%
v0	570	-
v1 (delete dos caches e índices)	520	91%
v2 (multistage)	130	22%
v4 (distroless chroot/melange)	65	11%
v5 (distroless com alpine+musl)	50	9%

Hadolint

github.com/hadolint/hadolint

São dois linters pelo preço de um, ele além de apontar boas práticas docker

ele tmb checa as instruções RUN para problemas de shell (shell check)

Conclusão

Essa palestra é bastante técnica com muitas práticas e que não são simples de serem adotadas, requerem um treinamento extenso de toda equipe.

Como conciliar esses problemas de segurança com a adesão da equipe / empresa ?

Fechando com Fernando Pessoa

> Sou um técnico, mas tenho técnica só dentro da técnica. Fora disso sou doido, com todo o direito a sê-lo. Com todo o direito a sê-lo, ouviram?

Fernando Pessoa (Álvaro de Campos)