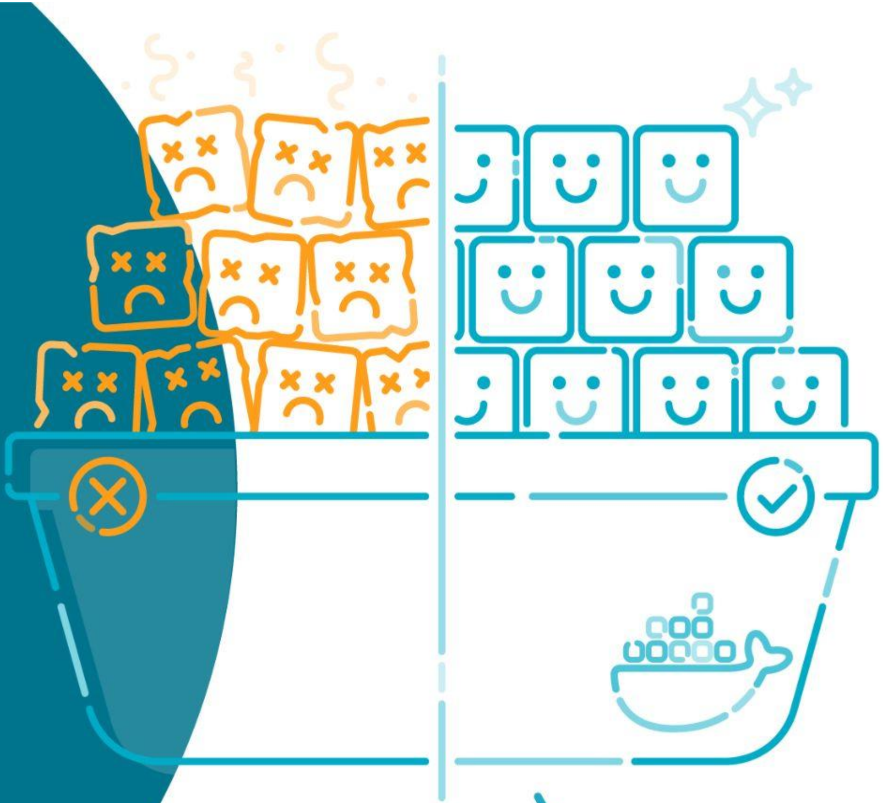
SYSDIG WEBINARS

Top Dockerfile Best Practices

ALVARO IRADIER

Integrations Engineer,
Sysdig



sysdig

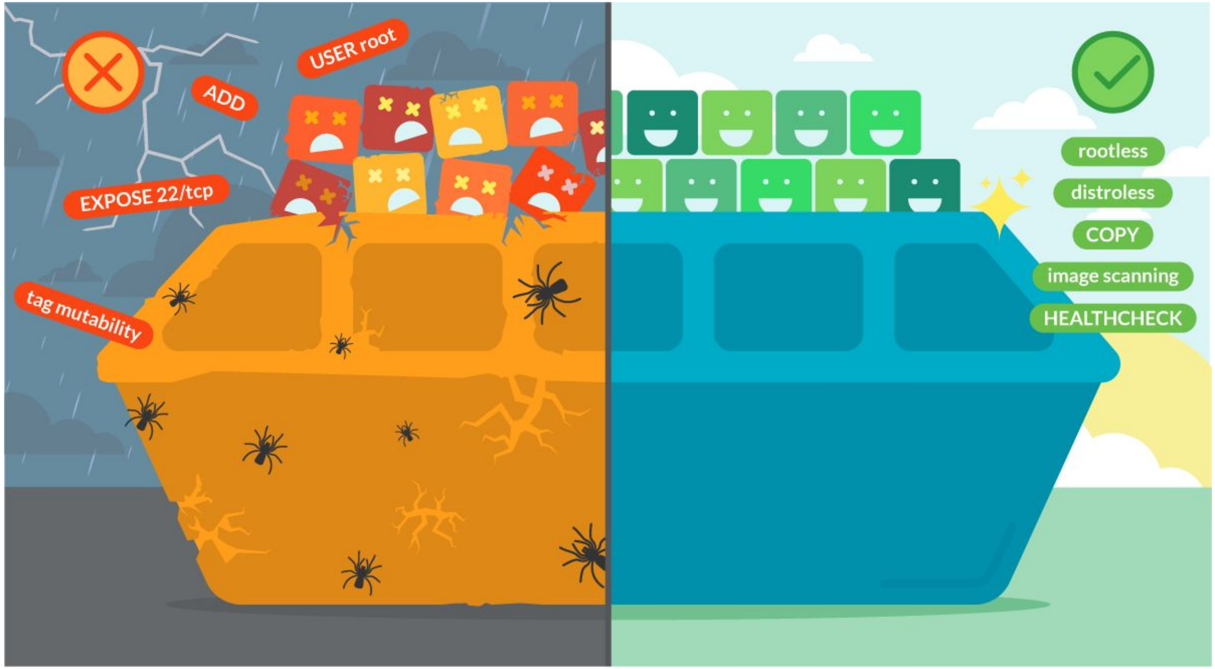
Agenda

- Why
- Shifting left security
- 1. Unnecessary privileges
- 2. Reduce attack surface
- 3. Credentials & Confidentiality
- 4. Linting & Scanning
- 5. Beyond image building

Examples available at: https://github.com/airadier/webinar-dockerfile-best-practices

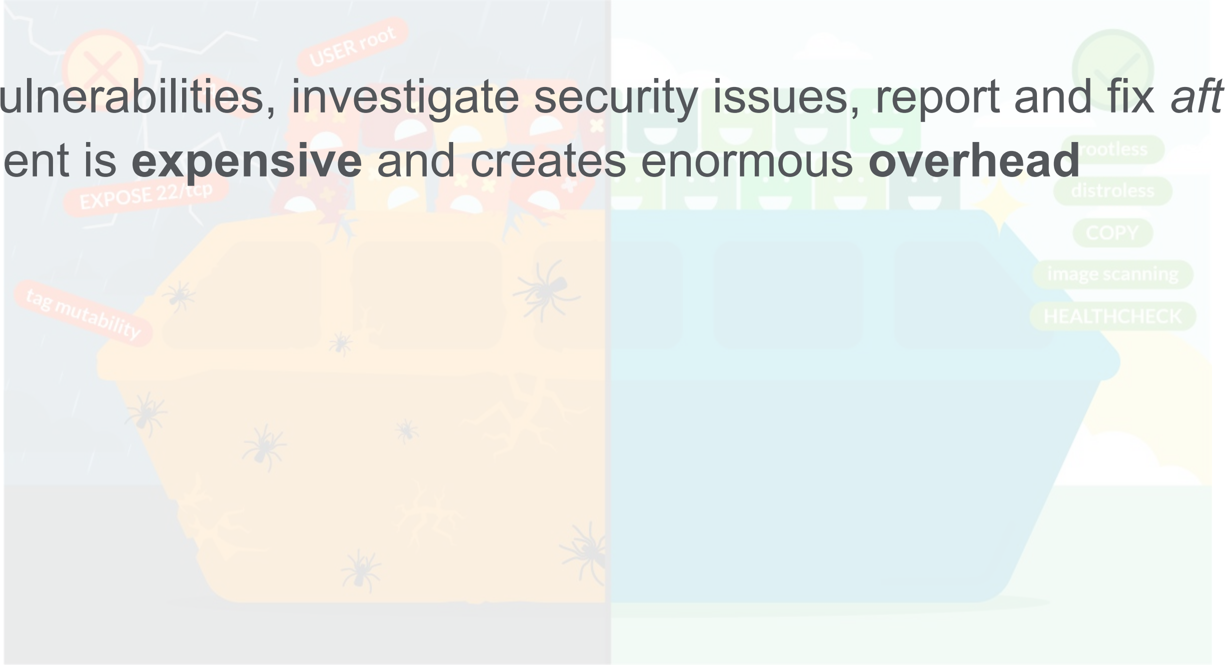
Why

Why


USER root
ADD
EXPOSE 22/tcp
tag mutability
rootless
distroless
COPY
image scanning
HEALTHCHECK

Why

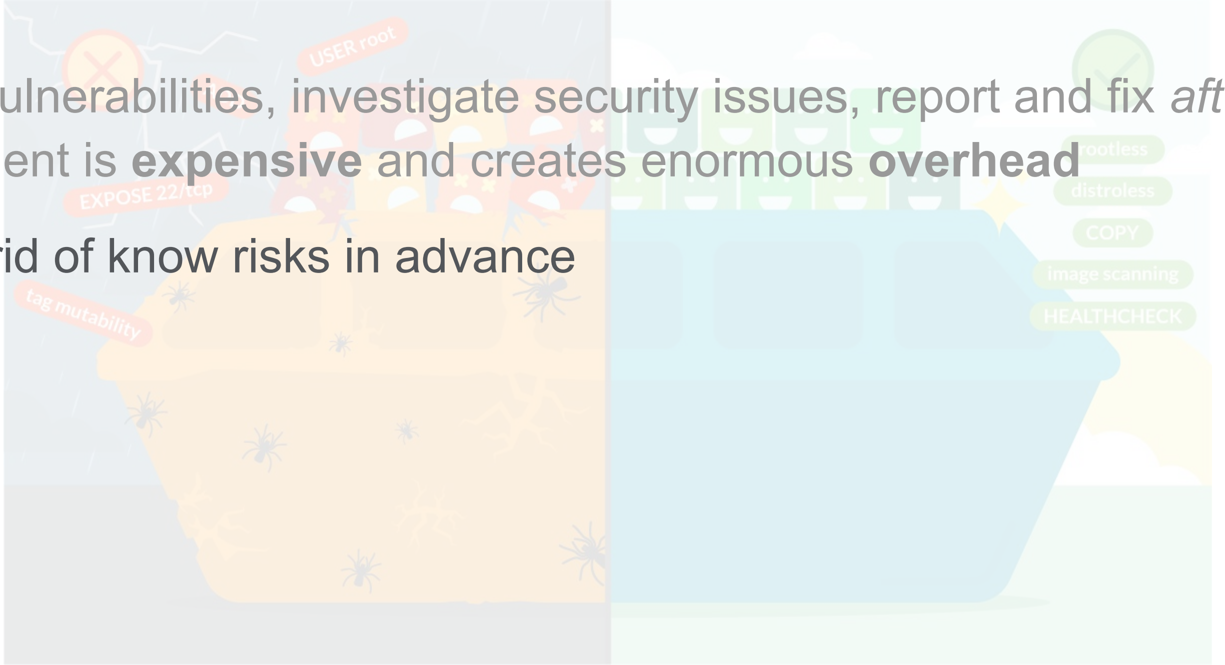
- Detect vulnerabilities, investigate security issues, report and fix *after* deployment is **expensive** and creates enormous **overhead**



Why

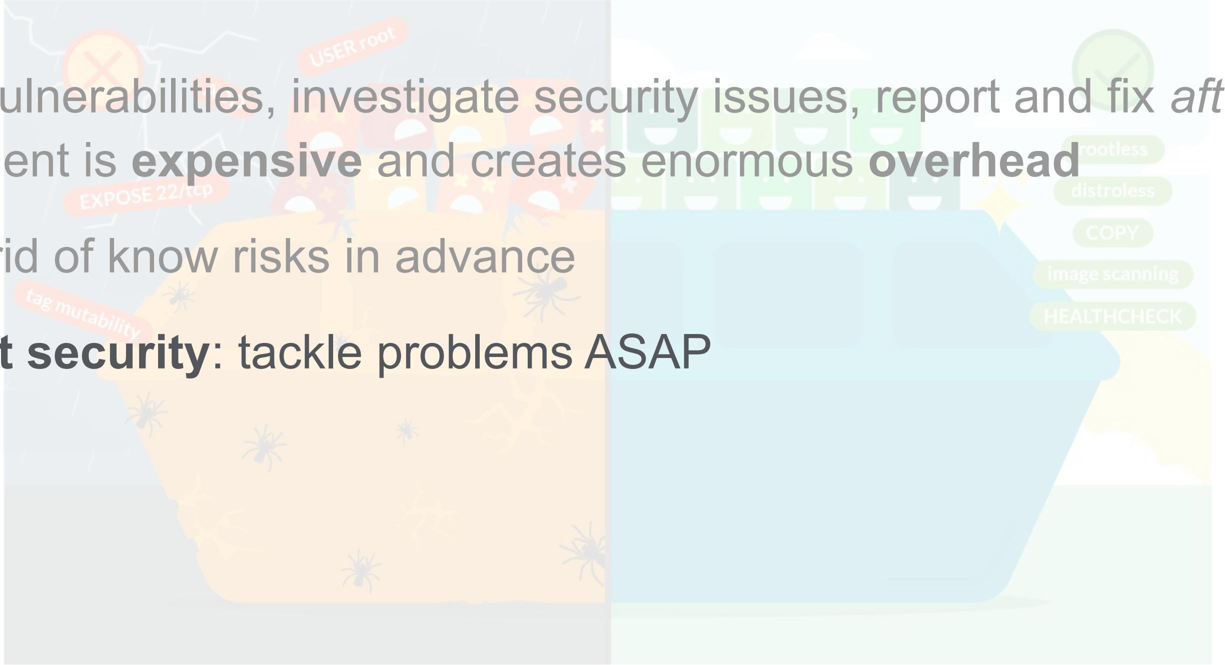
- Detect vulnerabilities, investigate security issues, report and fix *after* deployment is **expensive** and creates enormous **overhead**

- So, get rid of know risks in advance

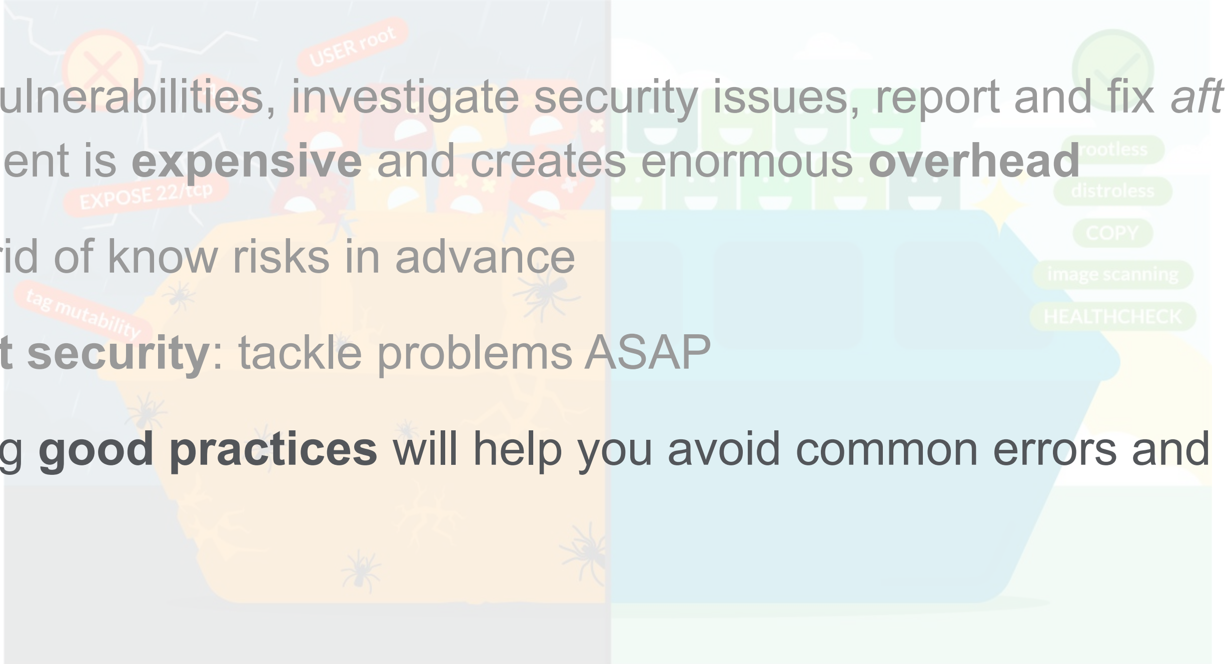


Why

- Detect vulnerabilities, investigate security issues, report and fix *after* deployment is **expensive** and creates enormous **overhead**
- So, get rid of know risks in advance
- **Shift left security**: tackle problems ASAP



Why



- Detect vulnerabilities, investigate security issues, report and fix *after* deployment is **expensive** and creates enormous **overhead**
- So, get rid of know risks in advance
- **Shift left security**: tackle problems ASAP
- Following **good practices** will help you avoid common errors and pitfalls

Shifting left security

sysdig

Shifting left security

Security traditionally managed later on the software life cycle:

Shifting left security

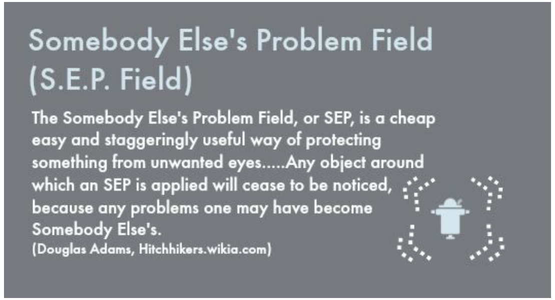
Security traditionally managed later on the software life cycle:

- You deliver your part of the cycle.

Shifting left security

Security traditionally managed later on the software life cycle:

- You deliver your part of the cycle.
- Whatever happens next is “Somebody Else’s Problem"



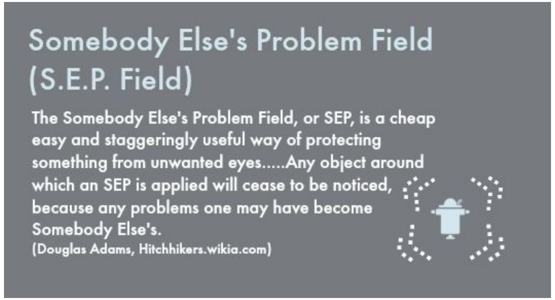
Somebody Else's Problem Field
(S.E.P. Field)

The Somebody Else's Problem Field, or SEP, is a cheap easy and staggeringly useful way of protecting something from unwanted eyes.....Any object around which an SEP is applied will cease to be noticed, because any problems one may have become Somebody Else's.
(Douglas Adams, Hitchhikers.wikia.com)

Shifting left security

Security traditionally managed later on the software life cycle:

- You deliver your part of the cycle.
- Whatever happens next is “Somebody Else’s Problem"
- Periodic auditor checks



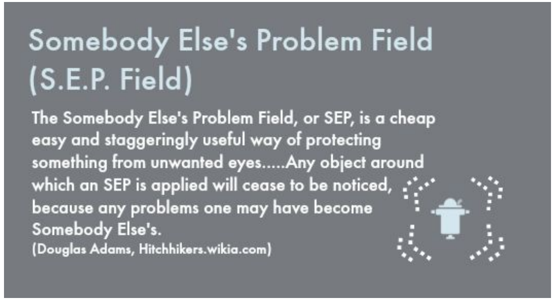
Somebody Else's Problem Field (S.E.P. Field)

The Somebody Else's Problem Field, or SEP, is a cheap easy and staggeringly useful way of protecting something from unwanted eyes.....Any object around which an SEP is applied will cease to be noticed, because any problems one may have become Somebody Else's.
(Douglas Adams, Hitchhikers.wikia.com)

Shifting left security

Security traditionally managed later on the software life cycle:

- You deliver your part of the cycle.
- Whatever happens next is “Somebody Else’s Problem"
- Periodic auditor checks
- InfoSec recommendations, benchmarks and validation: often seen by other parts of the company as a “barrier” that slows down the deployment of newest & shiniest software versions



Somebody Else's Problem Field (S.E.P. Field)

The Somebody Else's Problem Field, or SEP, is a cheap easy and staggeringly useful way of protecting something from unwanted eyes.....Any object around which an SEP is applied will cease to be noticed, because any problems one may have become Somebody Else's.

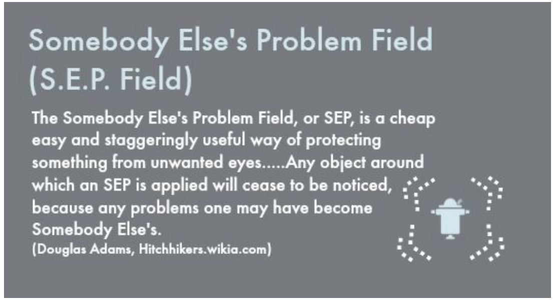
(Douglas Adams, Hitchhikers.wikia.com)

Shifting left security

Security traditionally managed later on the software life cycle:

- You deliver your part of the cycle.
- Whatever happens next is “Somebody Else’s Problem"
- Periodic auditor checks
- InfoSec recommendations, benchmarks and validation: often seen by other parts of the company as a “barrier” that slows down the deployment of newest & shiniest software versions

Shifting-left means basically two things



Somebody Else's Problem Field (S.E.P. Field)

The Somebody Else's Problem Field, or SEP, is a cheap easy and staggeringly useful way of protecting something from unwanted eyes.....Any object around which an SEP is applied will cease to be noticed, because any problems one may have become Somebody Else's.

(Douglas Adams, Hitchhikers.wikia.com)

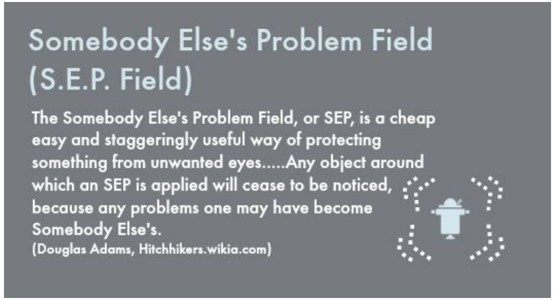
Shifting left security

Security traditionally managed later on the software life cycle:

- You deliver your part of the cycle.
- Whatever happens next is “Somebody Else’s Problem"
- Periodic auditor checks
- InfoSec recommendations, benchmarks and validation: often seen by other parts of the company as a “barrier” that slows down the deployment of newest & shiniest software versions

Shifting-left means basically two things

- Implementing security closer to the earlier stages of the life cycle, into the **developers realm** (repositories and builds)



Somebody Else's Problem Field
(S.E.P. Field)

The Somebody Else's Problem Field, or SEP, is a cheap easy and staggeringly useful way of protecting something from unwanted eyes.....Any object around which an SEP is applied will cease to be noticed, because any problems one may have become Somebody Else's.
(Douglas Adams, Hitchhikers.wikia.com)

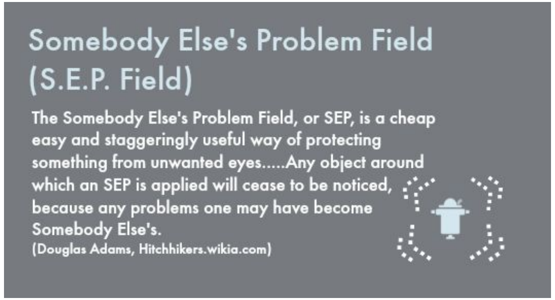
Shifting left security

Security traditionally managed later on the software life cycle:

- You deliver your part of the cycle.
- Whatever happens next is “Somebody Else’s Problem"
- Periodic auditor checks
- InfoSec recommendations, benchmarks and validation: often seen by other parts of the company as a “barrier” that slows down the deployment of newest & shiniest software versions

Shifting-left means basically two things

- Implementing security closer to the earlier stages of the life cycle, into the **developers realm** (repositories and builds)
- **Everyone shares the security responsibilities**, and collaborates towards common goals



Somebody Else's Problem Field (S.E.P. Field)

The Somebody Else's Problem Field, or SEP, is a cheap easy and staggeringly useful way of protecting something from unwanted eyes.....Any object around which an SEP is applied will cease to be noticed, because any problems one may have become Somebody Else's.

(Douglas Adams, Hitchhikers.wikia.com)

Shifting left security

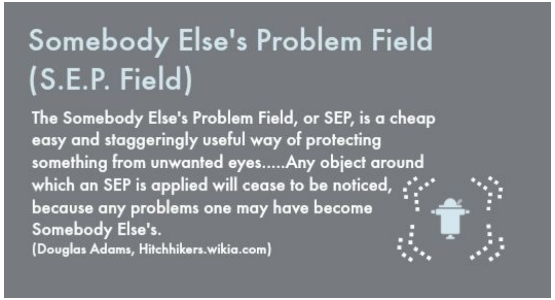
Security traditionally managed later on the software life cycle:

- You deliver your part of the cycle.
- Whatever happens next is “Somebody Else’s Problem"
- Periodic auditor checks
- InfoSec recommendations, benchmarks and validation: often seen by other parts of the company as a “barrier” that slows down the deployment of newest & shiniest software versions

Shifting-left means basically two things

- Implementing security closer to the earlier stages of the life cycle, into the **developers realm** (repositories and builds)
- **Everyone shares the security responsibilities**, and collaborates towards common goals

Dockerfile best practices are applied in **earlier stages**

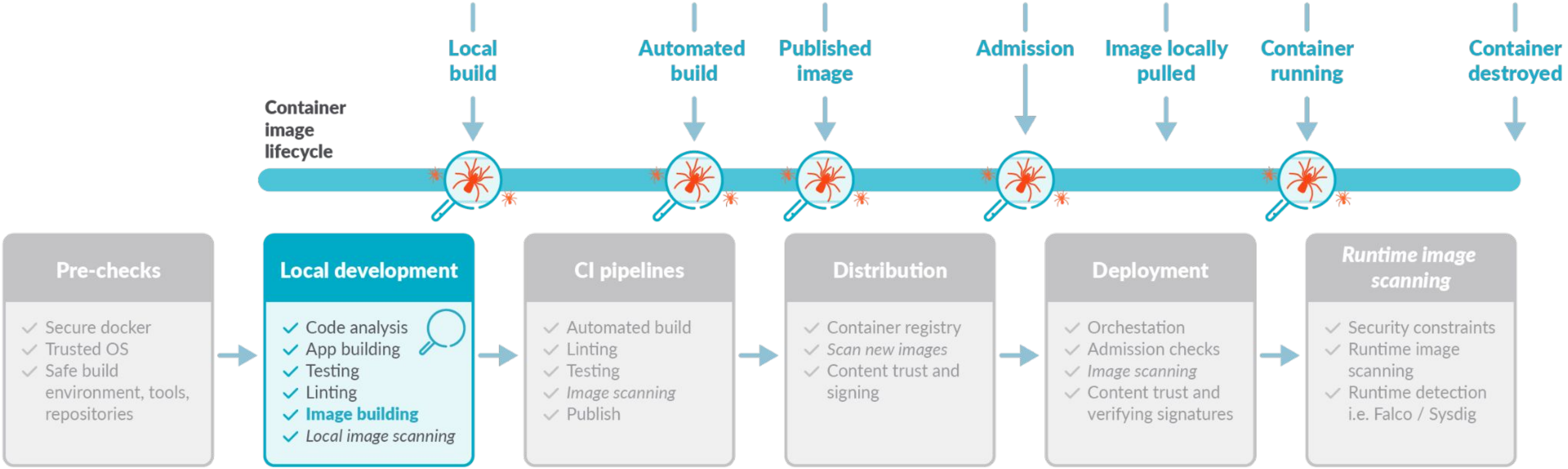


Somebody Else's Problem Field (S.E.P. Field)

The Somebody Else's Problem Field, or SEP, is a cheap easy and staggeringly useful way of protecting something from unwanted eyes.....Any object around which an SEP is applied will cease to be noticed, because any problems one may have become Somebody Else's.

(Douglas Adams, Hitchhikers.wikia.com)

Shifting left security



Local build

Automated build

Published image

Admission

Image locally pulled

Container running

Container destroyed

Container image lifecycle

Pre-checks

- ✓ Secure docker
- ✓ Trusted OS
- ✓ Safe build environment, tools, repositories

Local development

- ✓ Code analysis
- ✓ App building
- ✓ Testing
- ✓ Linting
- ✓ **Image building**
- ✓ *Local image scanning*

CI pipelines

- ✓ Automated build
- ✓ Linting
- ✓ Testing
- ✓ *Image scanning*
- ✓ Publish

Distribution

- ✓ Container registry
- ✓ *Scan new images*
- ✓ Content trust and signing

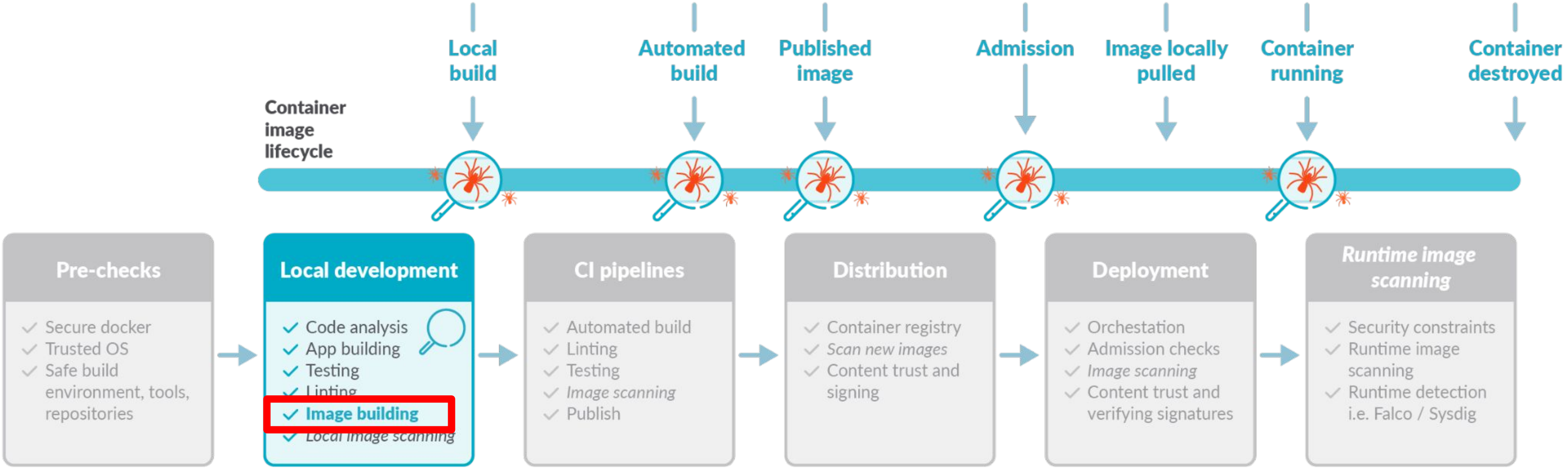
Deployment

- ✓ Orchestation
- ✓ Admission checks
- ✓ *Image scanning*
- ✓ Content trust and verifying signatures

Runtime image scanning

- ✓ Security constraints
- ✓ Runtime image scanning
- ✓ Runtime detection i.e. Falco / Sysdig

Shifting left security



Container image lifecycle

Local build

Automated build

Published image

Admission

Image locally pulled

Container running

Container destroyed

Pre-checks

- ✓ Secure docker
- ✓ Trusted OS
- ✓ Safe build environment, tools, repositories

Local development

- ✓ Code analysis
- ✓ App building
- ✓ Testing
- ✓ Linting
- ✓ **Image building**
- ✓ *Local image scanning*

CI pipelines

- ✓ Automated build
- ✓ Linting
- ✓ Testing
- ✓ *Image scanning*
- ✓ Publish

Distribution

- ✓ Container registry
- ✓ *Scan new images*
- ✓ Content trust and signing

Deployment

- ✓ Orchestation
- ✓ Admission checks
- ✓ *Image scanning*
- ✓ Content trust and verifying signatures

Runtime image scanning

- ✓ Security constraints
- ✓ Runtime image scanning
- ✓ Runtime detection i.e. Falco / Sysdig

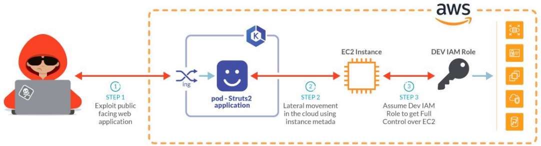
1. Unnecessary privileges

Unnecessary privileges

- **Problem**: containers often running with more privileges than required
  - Our recent report highlighted that 58% of images are running the container entrypoint as root (UID 0)

58%
run as root

- **Risks**:
  - Attackers can exploit vulnerabilities or bugs to gain access to other services or resources
  - https://sysdig.com/blog/lateral-movement-cloud-containers/
  - Openshift or others blocking root containers


aws
ing
pod - Struts2 application
EC2 Instance
DEV IAM Role
1
STEP 1
Exploit public facing web application
2
STEP 2
Lateral movement in the cloud using instance metada
3
STEP 3
Assume Dev IAM Role to get Full Control over EC2

Unnecessary privileges

- Step 1: Initial Dockerfile

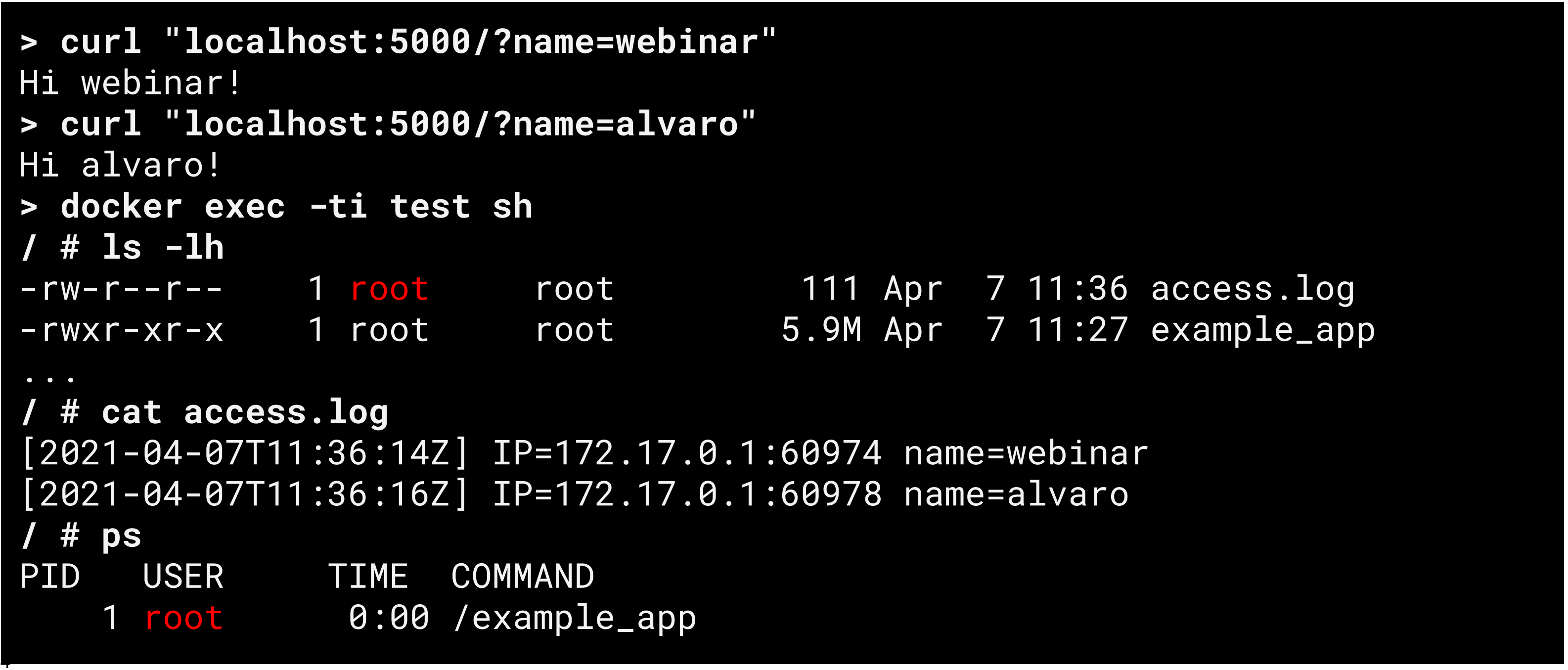
```
FROM alpine
COPY ./src/example_app /example_app
ENTRYPOINT /example_app
```

```
> docker build -f Dockerfile1 -t test1_1 .
[+] Building 0.5s (7/7) FINISHED

> docker run --name test --rm -p 5000:5000 test1_1
Listening at :5000
```

Unnecessary privileges

- Testing the app



```
> curl "localhost:5000/?name=webinar"
Hi webinar!
> curl "localhost:5000/?name=alvaro"
Hi alvaro!
> docker exec -ti test sh
/ # ls -lh
-rw-r--r--    1 root     root         111 Apr  7 11:36 access.log
-rwxr-xr-x    1 root     root        5.9M Apr  7 11:27 example_app
...
/ # cat access.log
[2021-04-07T11:36:14Z] IP=172.17.0.1:60974 name=webinar
[2021-04-07T11:36:16Z] IP=172.17.0.1:60978 name=alvaro
/ # ps
PID   USER     TIME  COMMAND
    1 root      0:00 /example_app
```

Unnecessary privileges

- Step 2: running as non-root

```
FROM alpine
COPY ./src/example_app /app/example_app
WORKDIR /app
USER 1000
ENTRYPOINT /app/example_app
```

```
> docker build -f Dockerfile2 -t test1_2 .
[+] Building 0.5s (7/7) FINISHED
> docker run --name test --rm -p 5000:5000 test1_2
panic: open access.log: permission denied
```

Unnecessary privileges

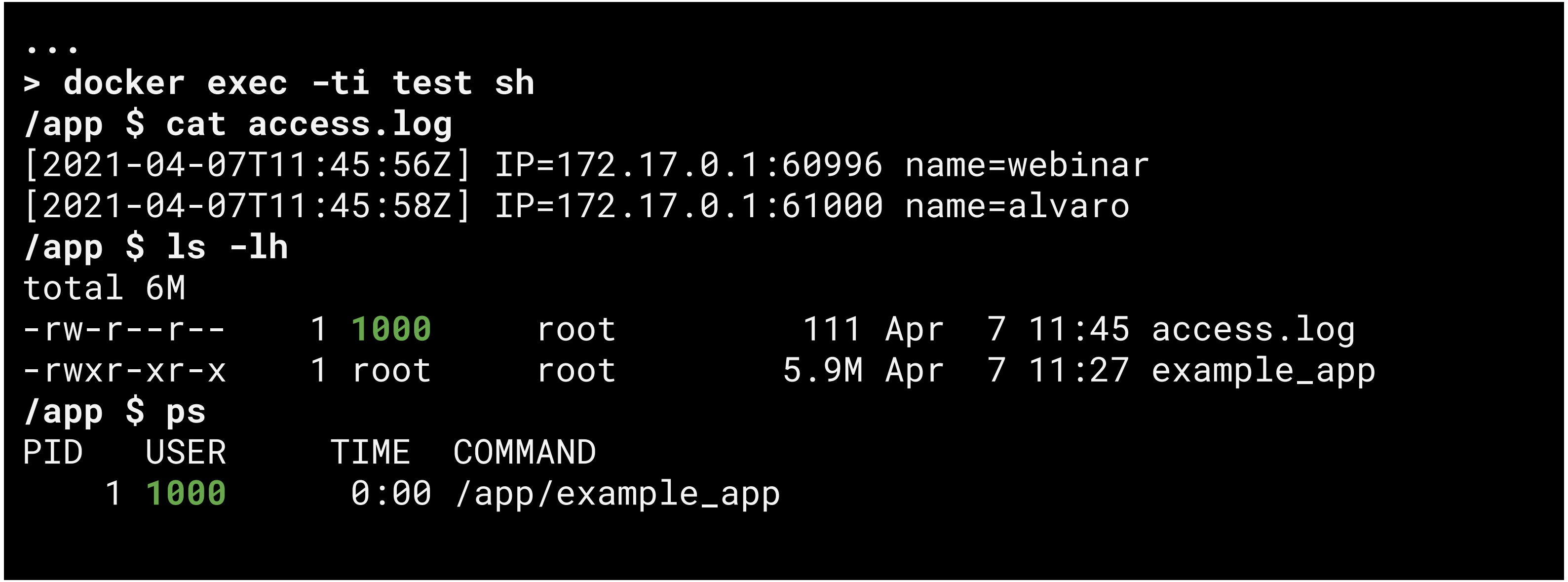
- Step 3: fix permissions

```
FROM alpine
COPY ./src/example_app /app/example_app
WORKDIR /app
RUN chown 1000:1000 /app
USER 1000
ENTRYPOINT /app/example_app
```

```
> docker build -f Dockerfile3 -t test1_3 .
[+] Building 0.5s (7/7) FINISHED
> docker run --name test --rm -p 5000:5000 test1_3
Listening at :5000
```

Unnecessary privileges

- Testing the **rootless** app



```
...
> docker exec -ti test sh
/app $ cat access.log
[2021-04-07T11:45:56Z] IP=172.17.0.1:60996 name=webinar
[2021-04-07T11:45:58Z] IP=172.17.0.1:61000 name=alvaro
/app $ ls -lh
total 6M
-rw-r--r--    1 1000     root         111 Apr  7 11:45 access.log
-rwxr-xr-x    1 root     root        5.9M Apr  7 11:27 example_app
/app $ ps
PID   USER     TIME  COMMAND
    1 1000      0:00 /app/example_app
```

Unnecessary privileges

- Step 4: unspecific UID

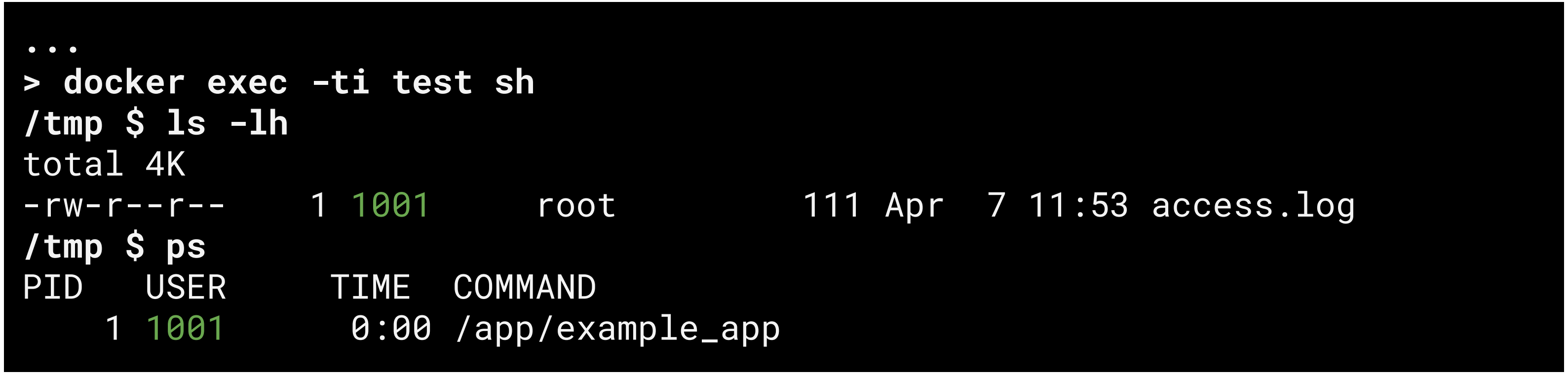
```
> docker run -u 1001 --name test --rm -p 5000:5000 test1_3
panic: open access.log: permission denied
```

```
FROM alpine
COPY ./src/example_app /app/example_app
USER 1000
WORKDIR /tmp
ENTRYPOINT /app/example_app
```

```
> docker run -u 1001 --name test --rm -p 5000:5000 test1_4
Listening at :5000
```

Unnecessary privileges

- Verifying the container is running with UID 1001



```
...
> docker exec -ti test sh
/tmp $ ls -lh
total 4K
-rw-r--r--    1 1001     root         111 Apr  7 11:53 access.log
/tmp $ ps
PID   USER     TIME  COMMAND
    1 1001      0:00 /app/example_app
```

Unnecessary privileges

- **Recap**:
  - Step1: We started from a container running with default root user
  - Step 2: We make it run as **non-root** user, and find a permissions issue
  - Step 3: We fix permissions in the Dockerfile
    - Try to run it as other user UID, permissions issue again
  - Step 4: Allow running as **any user** by writing to /tmp
    - Separate app folder and data folder - we can have **persistence**

Unnecessary privileges

- **Prevention**:
  - Follow the principle of **least privilege** so your service or application only has access to the resources and information necessary to perform its purpose
  - **USER directive**, run as non-root by default
  - Allow running with **random UIDs**
    - Required in some environments, as Openshift
    - Simplifies permissions with host mounts: match container and host UIDs

2. Reduce attack surface

Reduce attack surface

- **Problem**: including unnecessary packages or exposing unused ports
- **Risks**:
  - Your system is more exposed to attacks
  - Using components not under your control

Reduce attack surface

- **Don't**:
  - **Build the application externally, copy into the container**
  - Bad reproducibility

```
FROM alpine
COPY ./src/example_app /example_app
USER 1000
WORKDIR /tmp
ENTRYPOINT /example_app
```

Reduce attack surface

- **Don't**:
  - **Build directly inside the final container**
    - Big image size and multiple layers, build toolchain included in the container, other unrequired packages, remainings of the application source code, ...

```
FROM alpine
RUN apk add go
COPY ./src/ /src
WORKDIR /src
RUN go build .
ENTRYPOINT /src/example_app
```

Reduce attack surface

- **Multistage builds**:

```
#This is the "builder" stage
FROM golang:1.16 as builder
WORKDIR /my-go-app
COPY src .
RUN GOOS=linux GOARCH=amd64 go build .

#This is the final stage, and we copy artifacts from "builder"
FROM alpine
COPY --from=builder /my-go-app/example_app /bin/example_app
ENTRYPOINT ["/bin/example_app"]
```

Reduce attack surface

- **Multistage builds**:
  - Reproducible builds, always same build environment
  - Minimal image size, no build tools or undesired packages

```
> docker images
test2_1                                                       latest        681b9e590ae9   5 minutes ago    448MB
test2_2                                                       latest        37d02efde1e4   35 seconds ago   7.55MB
...
```

Reduce attack surface

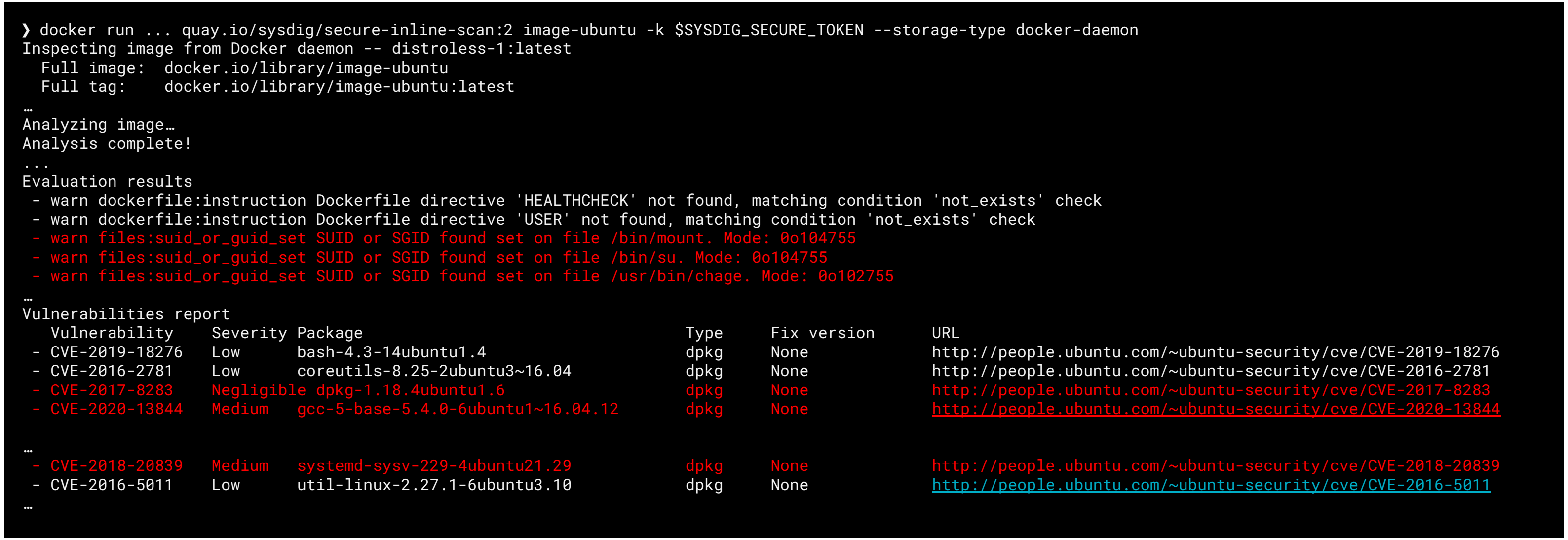
- **Example: multistage build with nodejs and typescript** (https://github.com/kevinpollet/typescript-docker-multi-stage-build):

```
FROM node:14-alpine AS builder
WORKDIR /usr/src/app
COPY typescript-docker-multi-stage-build/package*.json ./
RUN npm ci
COPY typescript-docker-multi-stage-build/tsconfig*.json ./
COPY typescript-docker-multi-stage-build/src src
RUN npm run build

FROM node:14-alpine
ENV NODE_ENV=production
WORKDIR /usr/src/app
RUN chown node:node .
USER node
COPY typescript-docker-multi-stage-build/package*.json ./
RUN npm install
COPY --from=builder /usr/src/app/lib/ lib/
EXPOSE 3000
ENTRYPOINT [ "node", "lib/server.js" ]
```

Reduce attack surface

- **Don't**: Use big, generic distro images if not needed (i.e. ubuntu)

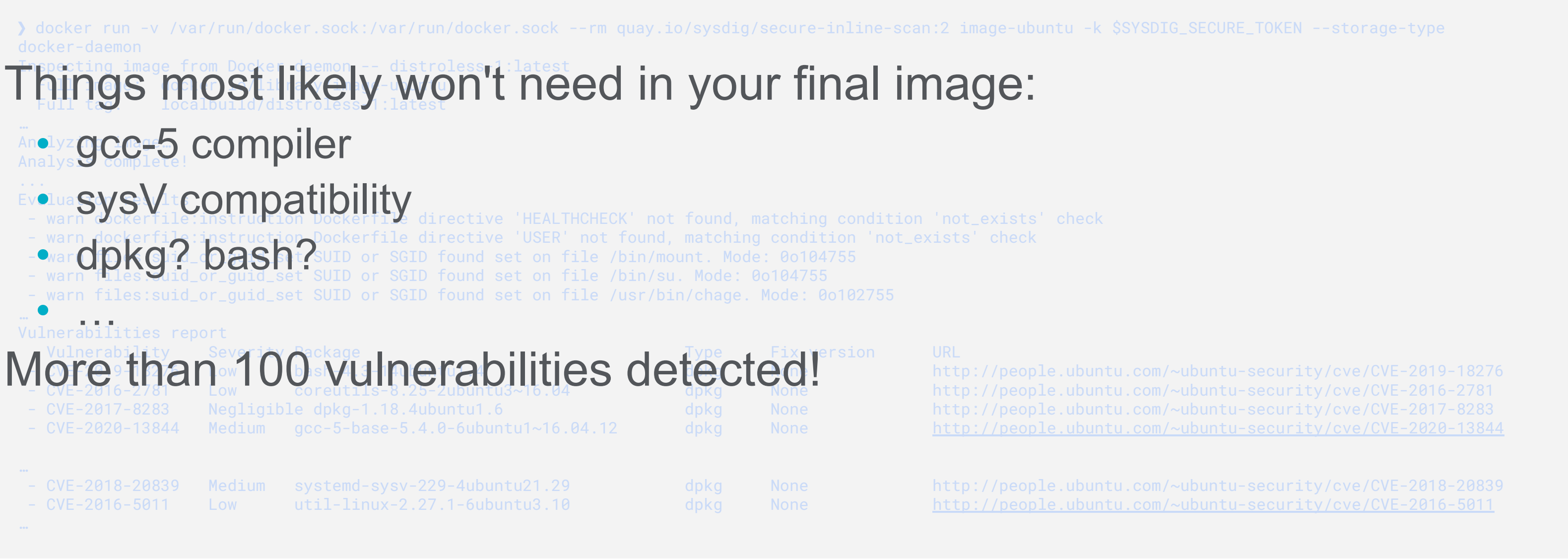


```
❯ docker run ... quay.io/sysdig/secure-inline-scan:2 image-ubuntu -k $SYSDIG_SECURE_TOKEN --storage-type docker-daemon
Inspecting image from Docker daemon -- distroless-1:latest
  Full image:  docker.io/library/image-ubuntu
  Full tag:    docker.io/library/image-ubuntu:latest
…
Analyzing image…
Analysis complete!
...
Evaluation results
 - warn dockerfile:instruction Dockerfile directive 'HEALTHCHECK' not found, matching condition 'not_exists' check
 - warn dockerfile:instruction Dockerfile directive 'USER' not found, matching condition 'not_exists' check
 - warn files:suid_or_guid_set SUID or SGID found set on file /bin/mount. Mode: 0o104755
 - warn files:suid_or_guid_set SUID or SGID found set on file /bin/su. Mode: 0o104755
 - warn files:suid_or_guid_set SUID or SGID found set on file /usr/bin/chage. Mode: 0o102755
…
Vulnerabilities report
   Vulnerability    Severity Package                                  Type     Fix version      URL
 - CVE-2019-18276   Low      bash-4.3-14ubuntu1.4                     dpkg     None             http://people.ubuntu.com/~ubuntu-security/cve/CVE-2019-18276
 - CVE-2016-2781    Low      coreutils-8.25-2ubuntu3~16.04            dpkg     None             http://people.ubuntu.com/~ubuntu-security/cve/CVE-2016-2781
 - CVE-2017-8283    Negligible dpkg-1.18.4ubuntu1.6                   dpkg     None             http://people.ubuntu.com/~ubuntu-security/cve/CVE-2017-8283
 - CVE-2020-13844   Medium   gcc-5-base-5.4.0-6ubuntu1~16.04.12       dpkg     None             http://people.ubuntu.com/~ubuntu-security/cve/CVE-2020-13844

…
 - CVE-2018-20839   Medium   systemd-sysv-229-4ubuntu21.29            dpkg     None             http://people.ubuntu.com/~ubuntu-security/cve/CVE-2018-20839
 - CVE-2016-5011    Low      util-linux-2.27.1-6ubuntu3.10            dpkg     None             http://people.ubuntu.com/~ubuntu-security/cve/CVE-2016-5011
…
```

Reduce attack surface

- **Don't: Use big, generic distro images if not needed (i.e. ubuntu)**
- Things most likely won't need in your final image:
  - gcc-5 compiler
  - sysV compatibility
  - dpkg? bash?
  - ...
- More than 100 vulnerabilities detected!



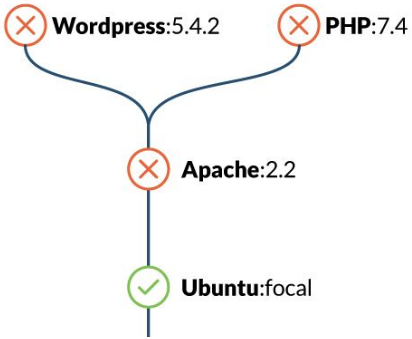
```
❯ docker run -v /var/run/docker.sock:/var/run/docker.sock --rm quay.io/sysdig/secure-inline-scan:2 image-ubuntu -k $SYSDIG_SECURE_TOKEN --storage-type
docker-daemon
Inspecting image from Docker daemon -- distroless-1:latest
  Full image:  docker.io/library/image-ubuntu
  Full tag:    localbuild/distroless-1:latest
…
Analyzing image…
Analysis complete!
...
Evaluation results
 - warn dockerfile:instruction Dockerfile directive 'HEALTHCHECK' not found, matching condition 'not_exists' check
 - warn dockerfile:instruction Dockerfile directive 'USER' not found, matching condition 'not_exists' check
 - warn files:suid_or_guid_set SUID or SGID found set on file /bin/mount. Mode: 0o104755
 - warn files:suid_or_guid_set SUID or SGID found set on file /bin/su. Mode: 0o104755
 - warn files:suid_or_guid_set SUID or SGID found set on file /usr/bin/chage. Mode: 0o102755
…
Vulnerabilities report
   Vulnerability     Severity Package                           Type     Fix version       URL
 - CVE-2019-18276    Low      bash-4.3-14ubuntu1.4              dpkg     None              http://people.ubuntu.com/~ubuntu-security/cve/CVE-2019-18276
 - CVE-2016-2781     Low      coreutils-8.25-2ubuntu3~16.04     dpkg     None              http://people.ubuntu.com/~ubuntu-security/cve/CVE-2016-2781
 - CVE-2017-8283     Negligible dpkg-1.18.4ubuntu1.6            dpkg     None              http://people.ubuntu.com/~ubuntu-security/cve/CVE-2017-8283
 - CVE-2020-13844    Medium   gcc-5-base-5.4.0-6ubuntu1~16.04.12 dpkg    None              http://people.ubuntu.com/~ubuntu-security/cve/CVE-2020-13844

…
 - CVE-2018-20839    Medium   systemd-sysv-229-4ubuntu21.29     dpkg     None              http://people.ubuntu.com/~ubuntu-security/cve/CVE-2018-20839
 - CVE-2016-5011     Low      util-linux-2.27.1-6ubuntu3.10     dpkg     None              http://people.ubuntu.com/~ubuntu-security/cve/CVE-2016-5011
…
```

Reduce attack surface

- Don't: **Official images might not be the best fit *per se***
  - Regarding security and minimalism, they might not be updated that often and can include extra packages for general use cases

- Don't: **Use outdated images**
  - New security vulnerabilities are discovered continuously
  - Stick to the latest security patches
  - No need to always go with the latest version (breaking changes)



Reduce attack surface

- **Don't**: **Use docker.io/johndoehacker/mycustom-node-image:latest**

```
FROM docker.io/johndoehacker/mycustom-node-image:latest
...
```

  - Inherit all of the problems and vulnerabilities from that image
  - Who builds and publishes that image?
  - Is it updated regularly?
  - How is it built?
  - Are we sure the published version is really from the public Dockerfile?

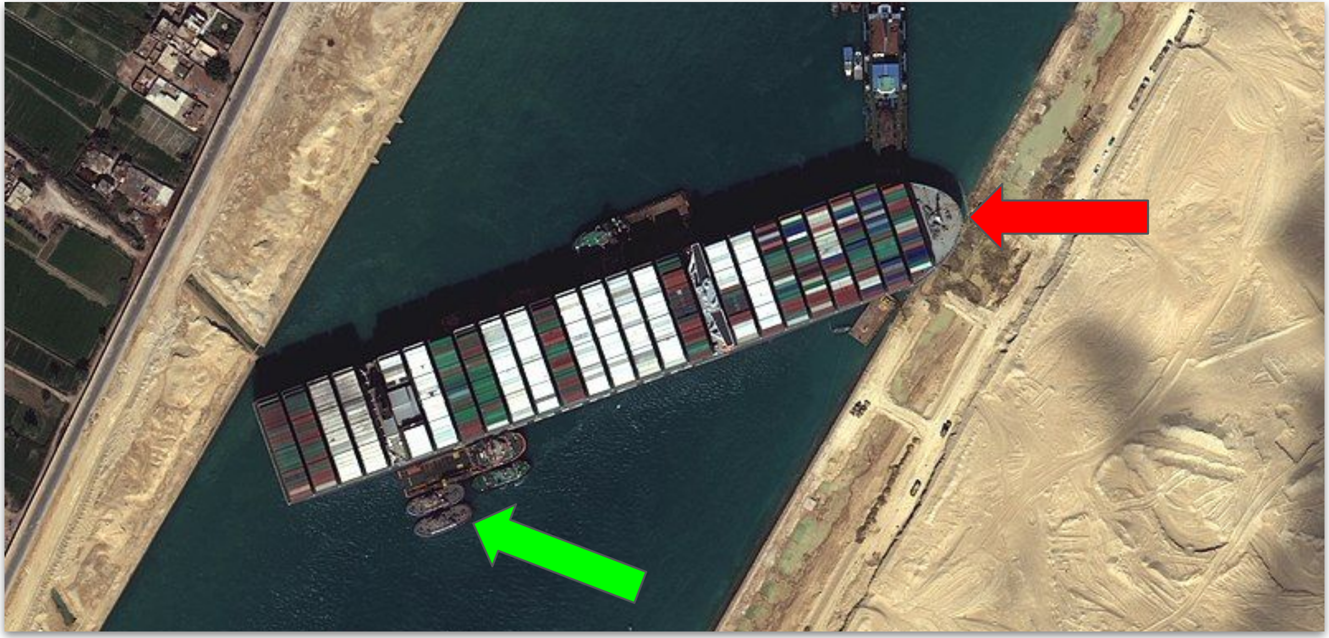
Reduce attack surface

- Prefer **verified** and **official** images from **trusted repositories and providers**
- Check for optimized vs generic versions
  - Example: bitnami/node vs official node image
    - customized versions on top of a minideb distribution
    - frequently updated with the latest bug fixes
    - signed with Docker Content Trust
    - pass a security scan for tracking known vulnerabilities

Reduce attack surface

- **Be minimal**
    - *alpine* versions
    - FROM scratch
    - Distroless (https://github.com/GoogleContainerTools/distroless):
        - i.e.: `FROM gcr.io/distroless/base-debian10`
            - Basic set of packages, including just required libraries like *glibc*, *libssl*, and *openssl*.
        - Slimmer: `FROM gcr.io/distroless/static-debian10`
            - For statically compiled applications like Go that don't require libc

Reduce attack surface



Reduce attack surface

- When using custom images, check for the **image source** and the Dockerfile, and **build your own base image**

- Define a **versioning strategy**:
  - Stick to stable or long-term support versions
  - Rebuild periodically
    - To get the latest packages from the base distro
    - npm, go mod offer ways to specify version ranges (keep up with latest security updates)
  - Plan in advance.
    - Be ready to drop old versions
    - Migrate before base image reaches the end of its life and stops receiving updates

Reduce attack surface

- **Prevention**:
  - Keep images *minimal*, only required stuff should be included
  - Carefully choose the image that best fits your use case
  - Use trusted, verified base images
  - Use stable and well maintained versions, with frequent updates
  - Update and rebuild your own images often

3. Credentials & Confidentiality

sysdig

Credentials & Confidentiality

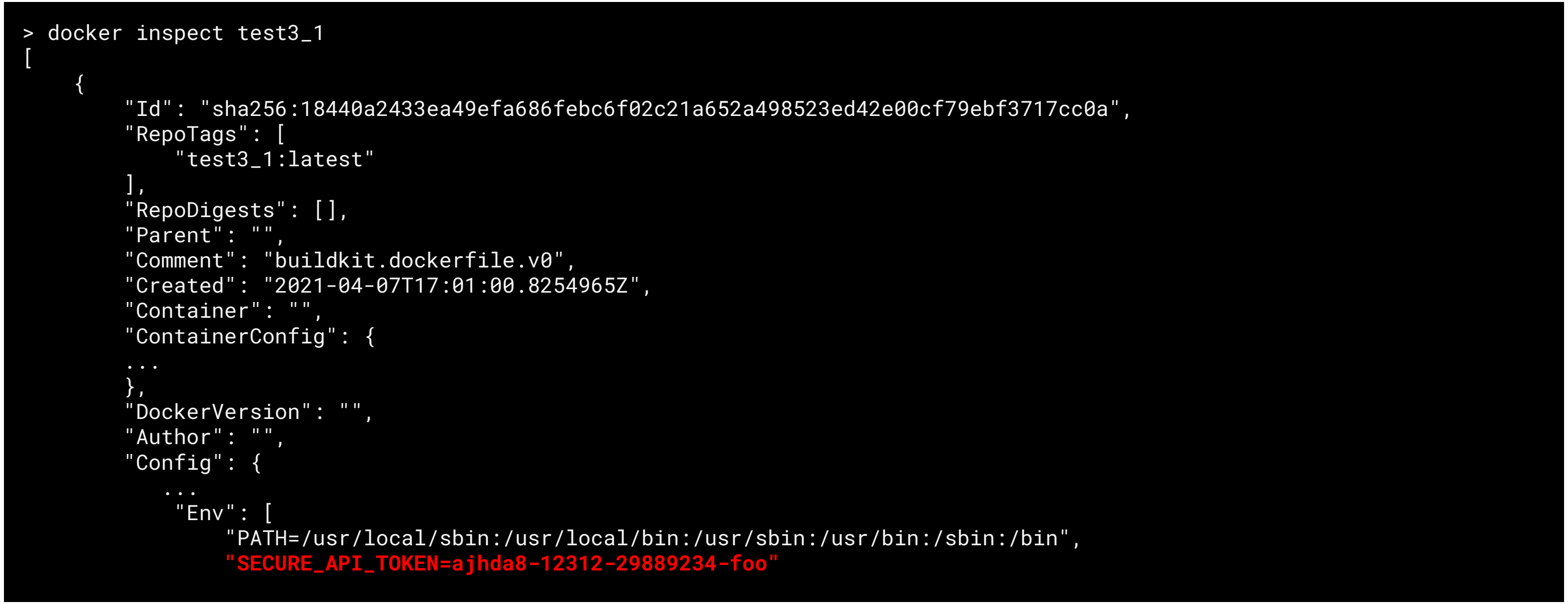
- **Problem**: leaking credentials or confidential information in your images
- **Risks**:
  - Attackers can use leaked credentials to access your systems
  - Exposal of confidential or sensitive information

Credentials & Confidentiality

- **Don't**:
  - Include hard coded credentials
  - Add credentials file or environment variables

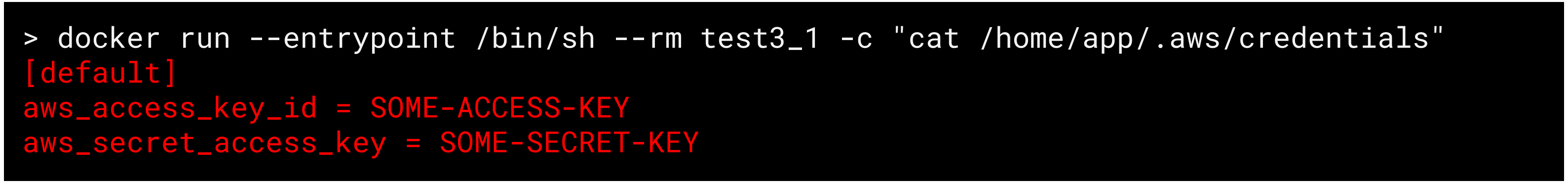
```
FROM alpine
...
ENV SECURE_API_TOKEN=ajhda8-12312-29889234-foo
COPY aws_credentials /home/app/.aws/credentials
...
ENTRYPOINT /example_app
```

Credentials & Confidentiality



```
> docker inspect test3_1
[
    {
        "Id": "sha256:18440a2433ea49efa686febc6f02c21a652a498523ed42e00cf79ebf3717cc0a",
        "RepoTags": [
            "test3_1:latest"
        ],
        "RepoDigests": [],
        "Parent": "",
        "Comment": "buildkit.dockerfile.v0",
        "Created": "2021-04-07T17:01:00.8254965Z",
        "Container": "",
        "ContainerConfig": {
        ...
        },
        "DockerVersion": "",
        "Author": "",
        "Config": {
           ...
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
                "SECURE_API_TOKEN=ajhda8-12312-29889234-foo"
```

Credentials & Confidentiality



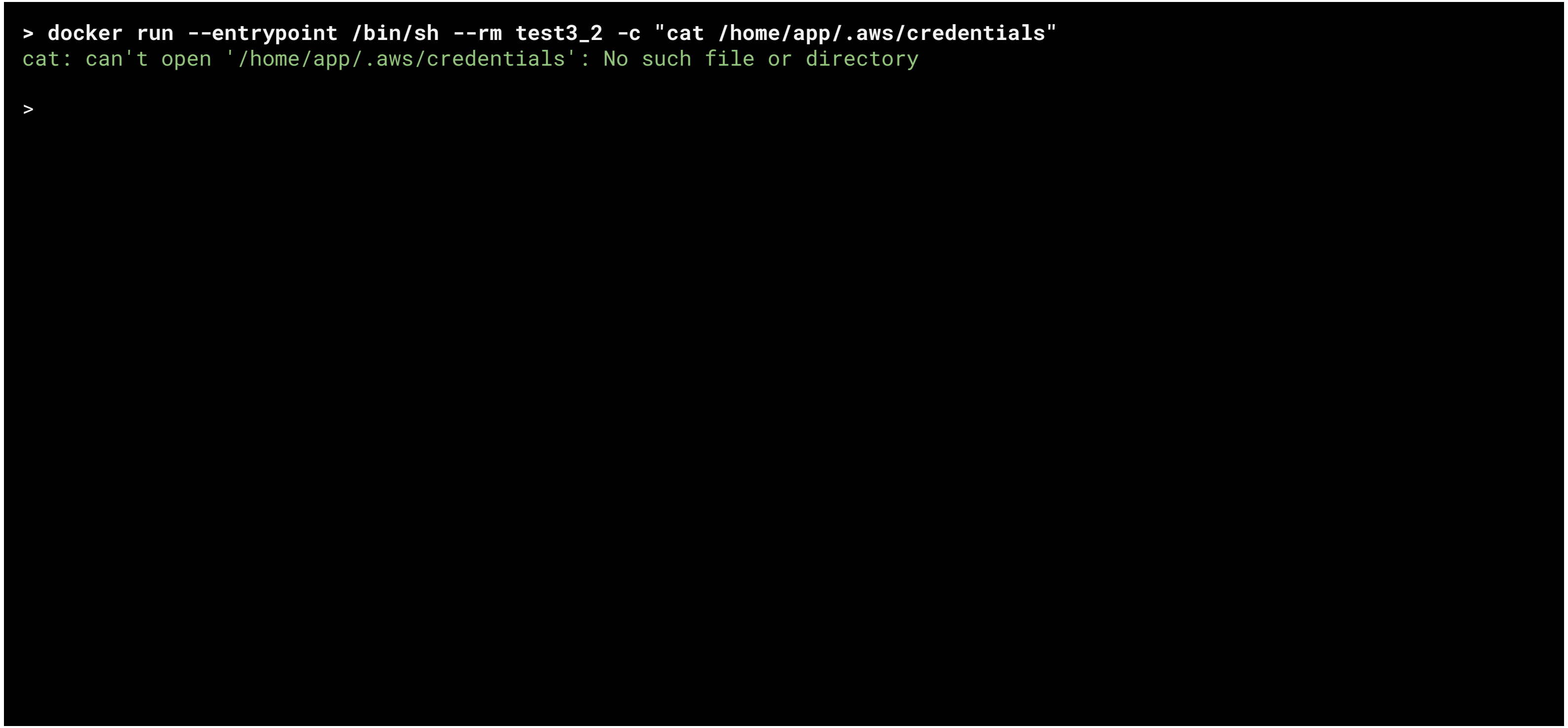
```
> docker run --entrypoint /bin/sh --rm test3_1 -c "cat /home/app/.aws/credentials"
[default]
aws_access_key_id = SOME-ACCESS-KEY
aws_secret_access_key = SOME-SECRET-KEY
```

Credentials & Confidentiality

- **Even if the file is removed!!**

```
FROM alpine
...
COPY aws_credentials /home/app/.aws/credentials
…
RUN rm /home/app/.aws/credentials
ENTRYPOINT /example_app
```

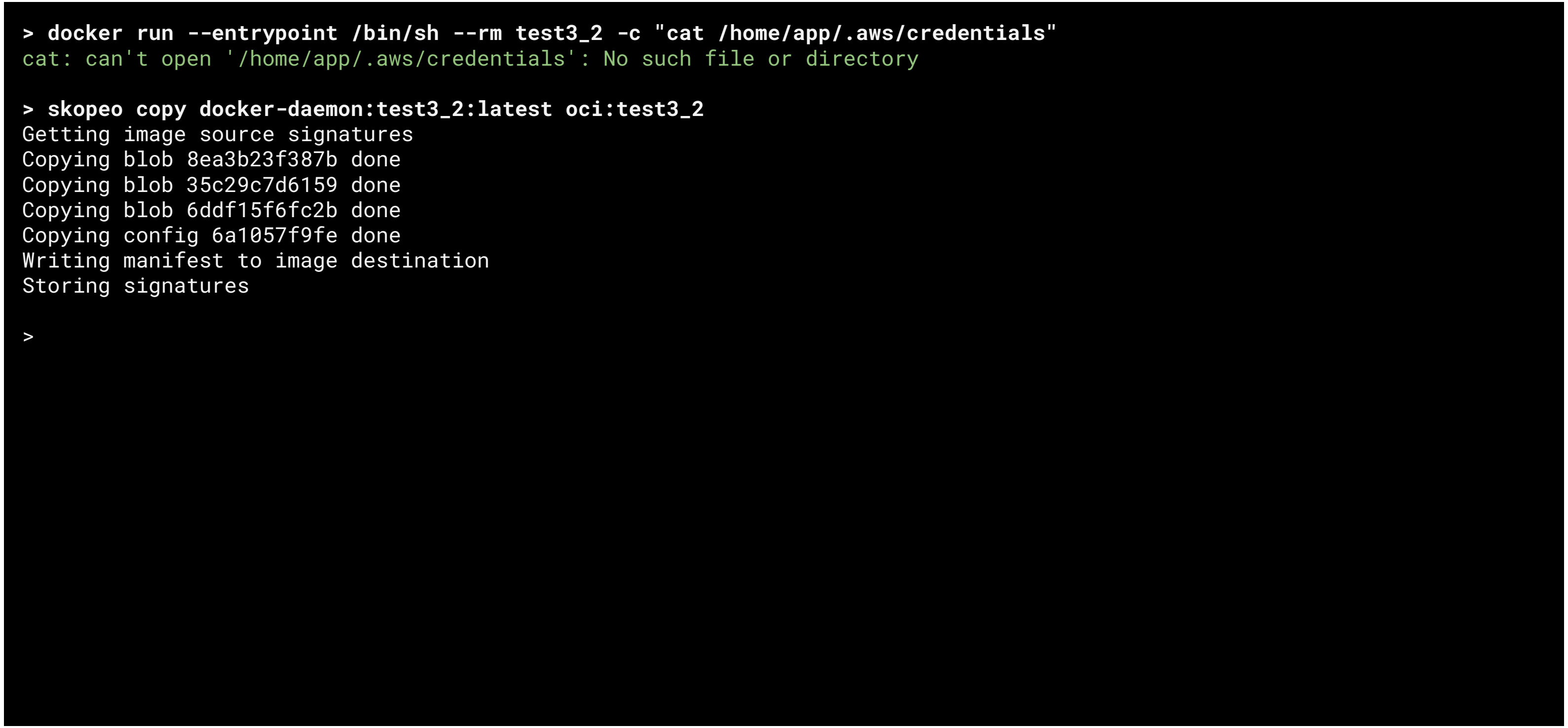
Credentials & Confidentiality



```
> docker run --entrypoint /bin/sh --rm test3_2 -c "cat /home/app/.aws/credentials"
cat: can't open '/home/app/.aws/credentials': No such file or directory

>
```

Credentials & Confidentiality

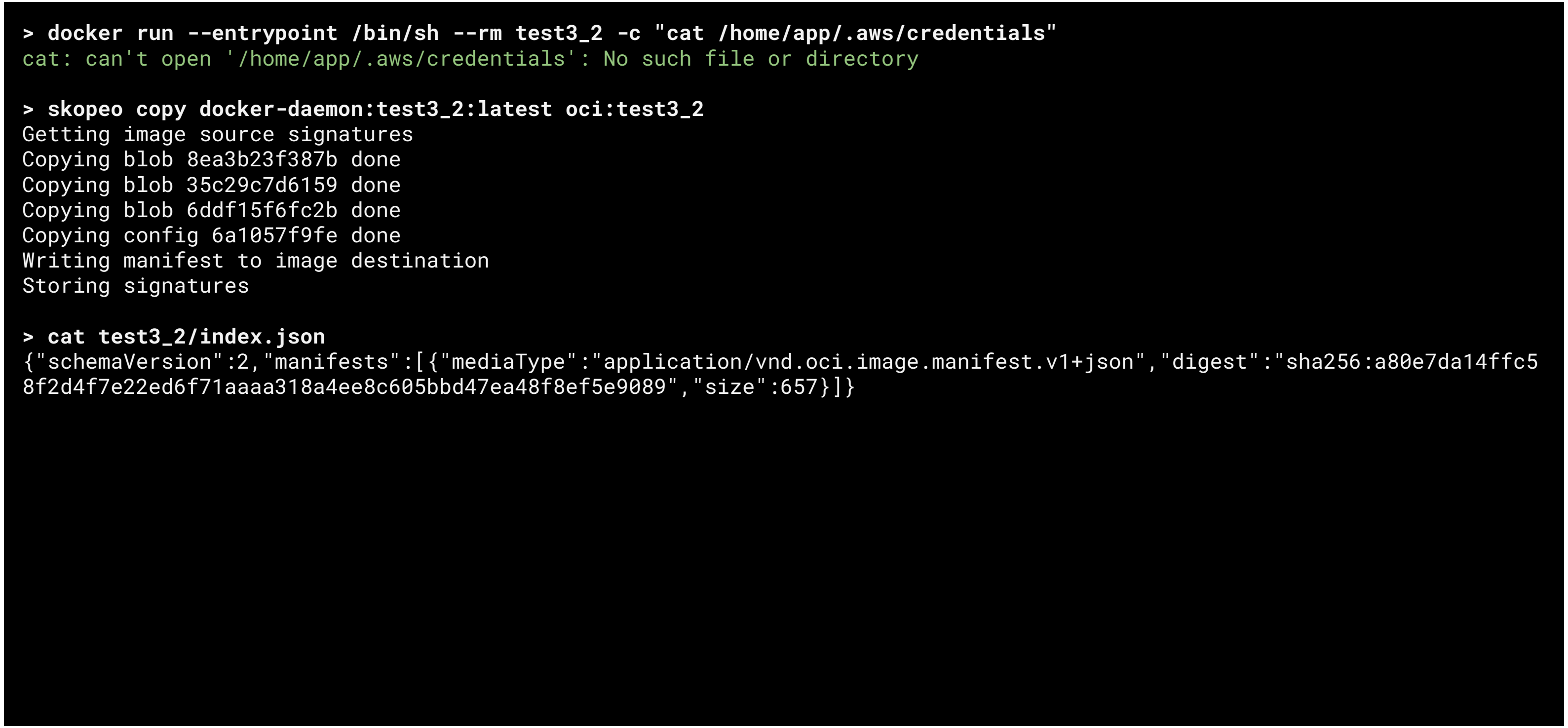


```
> docker run --entrypoint /bin/sh --rm test3_2 -c "cat /home/app/.aws/credentials"
cat: can't open '/home/app/.aws/credentials': No such file or directory

> skopeo copy docker-daemon:test3_2:latest oci:test3_2
Getting image source signatures
Copying blob 8ea3b23f387b done
Copying blob 35c29c7d6159 done
Copying blob 6ddf15f6fc2b done
Copying config 6a1057f9fe done
Writing manifest to image destination
Storing signatures

>
```

Credentials & Confidentiality

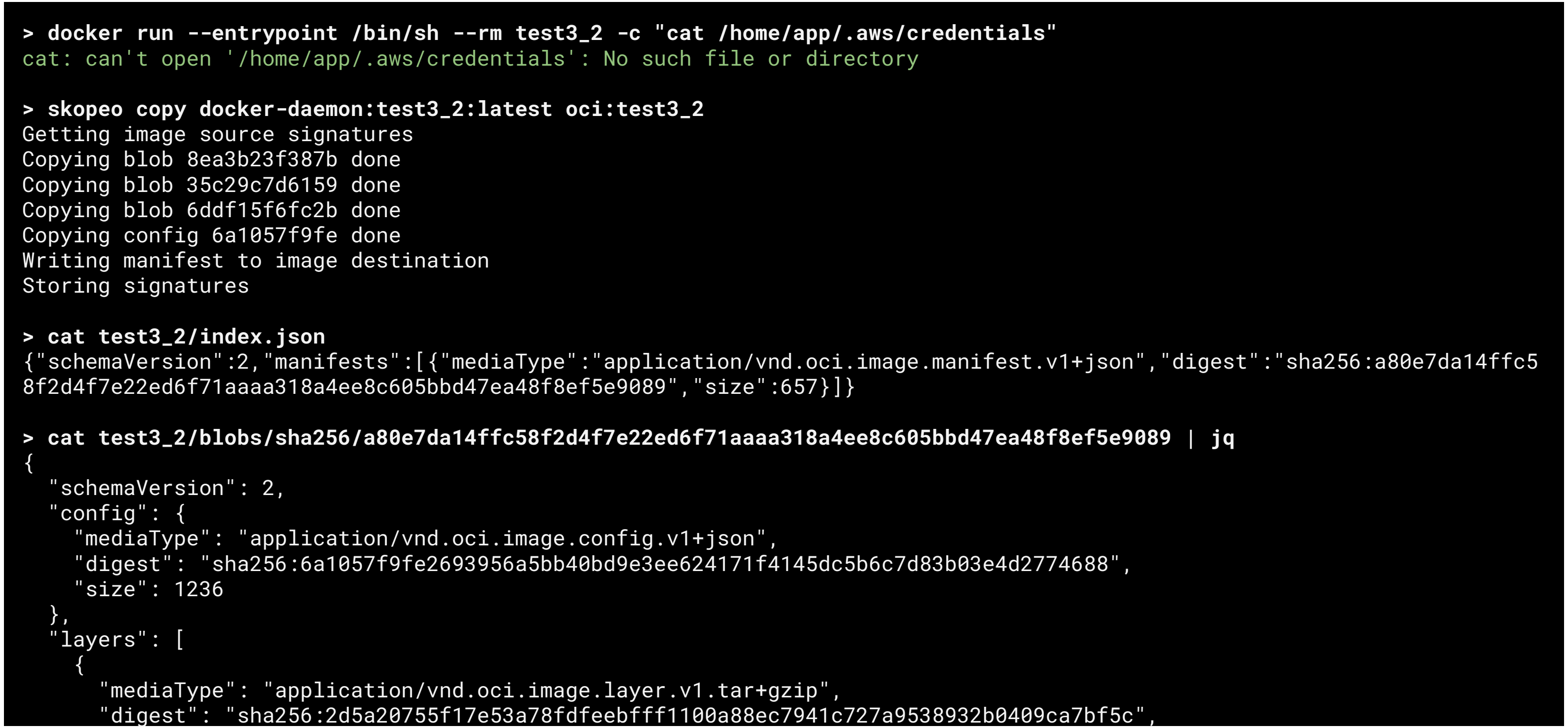


```
> docker run --entrypoint /bin/sh --rm test3_2 -c "cat /home/app/.aws/credentials"
cat: can't open '/home/app/.aws/credentials': No such file or directory

> skopeo copy docker-daemon:test3_2:latest oci:test3_2
Getting image source signatures
Copying blob 8ea3b23f387b done
Copying blob 35c29c7d6159 done
Copying blob 6ddf15f6fc2b done
Copying config 6a1057f9fe done
Writing manifest to image destination
Storing signatures

> cat test3_2/index.json
{"schemaVersion":2,"manifests":[{"mediaType":"application/vnd.oci.image.manifest.v1+json","digest":"sha256:a80e7da14ffc5
8f2d4f7e22ed6f71aaaa318a4ee8c605bbd47ea48f8ef5e9089","size":657}]}
```

Credentials & Confidentiality



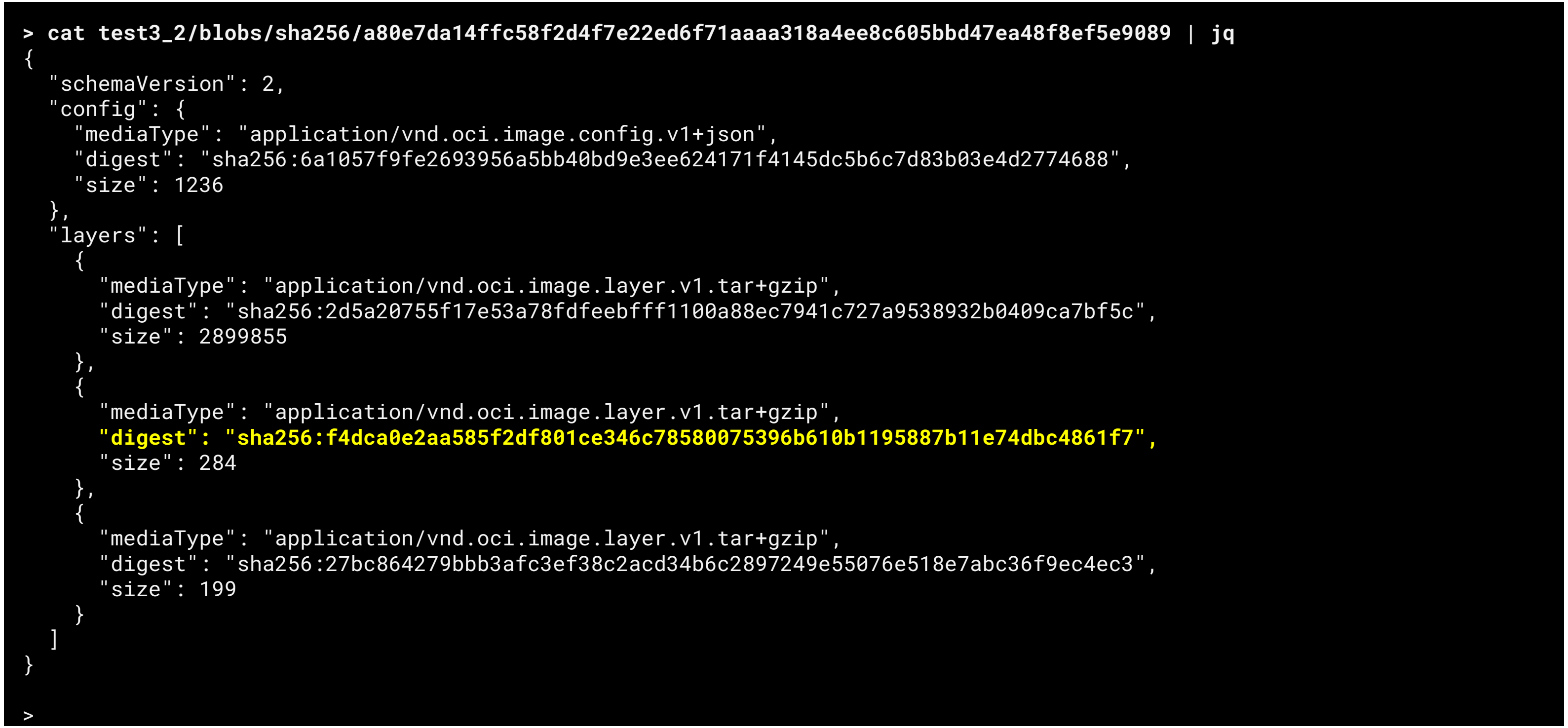
```
> docker run --entrypoint /bin/sh --rm test3_2 -c "cat /home/app/.aws/credentials"
cat: can't open '/home/app/.aws/credentials': No such file or directory

> skopeo copy docker-daemon:test3_2:latest oci:test3_2
Getting image source signatures
Copying blob 8ea3b23f387b done
Copying blob 35c29c7d6159 done
Copying blob 6ddf15f6fc2b done
Copying config 6a1057f9fe done
Writing manifest to image destination
Storing signatures

> cat test3_2/index.json
{"schemaVersion":2,"manifests":[{"mediaType":"application/vnd.oci.image.manifest.v1+json","digest":"sha256:a80e7da14ffc5
8f2d4f7e22ed6f71aaaa318a4ee8c605bbd47ea48f8ef5e9089","size":657}]}

> cat test3_2/blobs/sha256/a80e7da14ffc58f2d4f7e22ed6f71aaaa318a4ee8c605bbd47ea48f8ef5e9089 | jq
{
  "schemaVersion": 2,
  "config": {
    "mediaType": "application/vnd.oci.image.config.v1+json",
    "digest": "sha256:6a1057f9fe2693956a5bb40bd9e3ee624171f4145dc5b6c7d83b03e4d2774688",
    "size": 1236
  },
  "layers": [
    {
      "mediaType": "application/vnd.oci.image.layer.v1.tar+gzip",
      "digest": "sha256:2d5a20755f17e53a78fdfeebfff1100a88ec7941c727a9538932b0409ca7bf5c",
      "size": 2899855
    },
    {
      "mediaType": "application/vnd.oci.image.layer.v1.tar+gzip",
      "digest": "sha256:f4dca0e2aa585f2df801ce346c78580075396b610b1195887b11e74dbc4861f7",
```

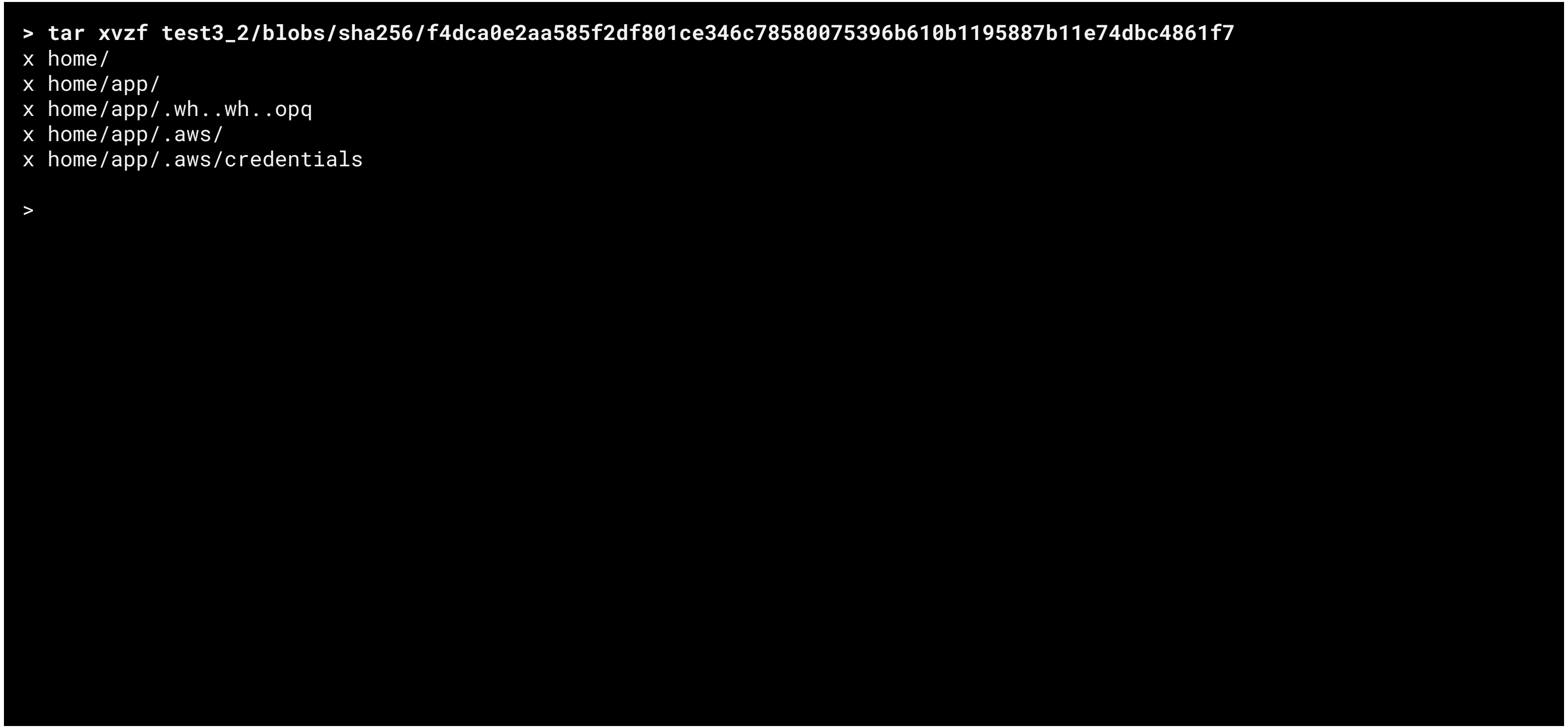
Credentials & Confidentiality



```
> cat test3_2/blobs/sha256/a80e7da14ffc58f2d4f7e22ed6f71aaaa318a4ee8c605bbd47ea48f8ef5e9089 | jq
{
  "schemaVersion": 2,
  "config": {
    "mediaType": "application/vnd.oci.image.config.v1+json",
    "digest": "sha256:6a1057f9fe2693956a5bb40bd9e3ee624171f4145dc5b6c7d83b03e4d2774688",
    "size": 1236
  },
  "layers": [
    {
      "mediaType": "application/vnd.oci.image.layer.v1.tar+gzip",
      "digest": "sha256:2d5a20755f17e53a78fdfeebfff1100a88ec7941c727a9538932b0409ca7bf5c",
      "size": 2899855
    },
    {
      "mediaType": "application/vnd.oci.image.layer.v1.tar+gzip",
      "digest": "sha256:f4dca0e2aa585f2df801ce346c78580075396b610b1195887b11e74dbc4861f7",
      "size": 284
    },
    {
      "mediaType": "application/vnd.oci.image.layer.v1.tar+gzip",
      "digest": "sha256:27bc864279bbb3afc3ef38c2acd34b6c2897249e55076e518e7abc36f9ec4ec3",
      "size": 199
    }
  ]
}

>
```

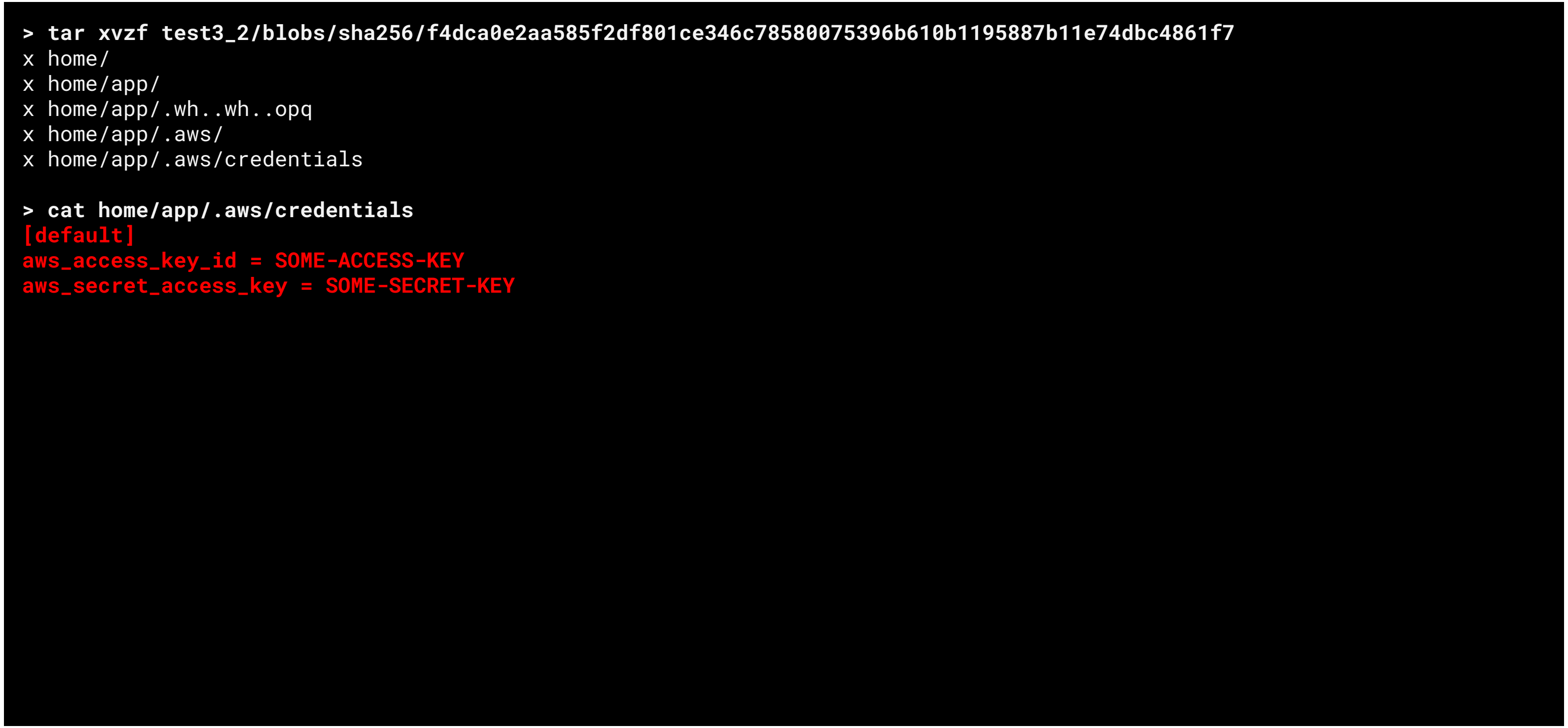
Credentials & Confidentiality



```
> tar xvzf test3_2/blobs/sha256/f4dca0e2aa585f2df801ce346c78580075396b610b1195887b11e74dbc4861f7
x home/
x home/app/
x home/app/.wh..wh..opq
x home/app/.aws/
x home/app/.aws/credentials

>
```

Credentials & Confidentiality



```
> tar xvzf test3_2/blobs/sha256/f4dca0e2aa585f2df801ce346c78580075396b610b1195887b11e74dbc4861f7
x home/
x home/app/
x home/app/.wh..wh..opq
x home/app/.aws/
x home/app/.aws/credentials

> cat home/app/.aws/credentials
[default]
aws_access_key_id = SOME-ACCESS-KEY
aws_secret_access_key = SOME-SECRET-KEY
```

Credentials & Confidentiality

- **Don't**:
  - Forget the **layered** nature of images (each command creates a new layer)
    - Removing a file in a layer layer still takes space and file can be accessed
    - Combine commands to reduce number of layers, i.e.:

      ```
      RUN apt-get install wget && wget https://…/downloadedfile.tar && tar xvzf
      downloadedfile.tar && rm downloadedfile.tar && apt-get remove wget
      ```
    - Optimize layers, place commands **less likely to change** (and easier to cache) **first**

Credentials & Confidentiality

- **Example (unoptimized layers)**

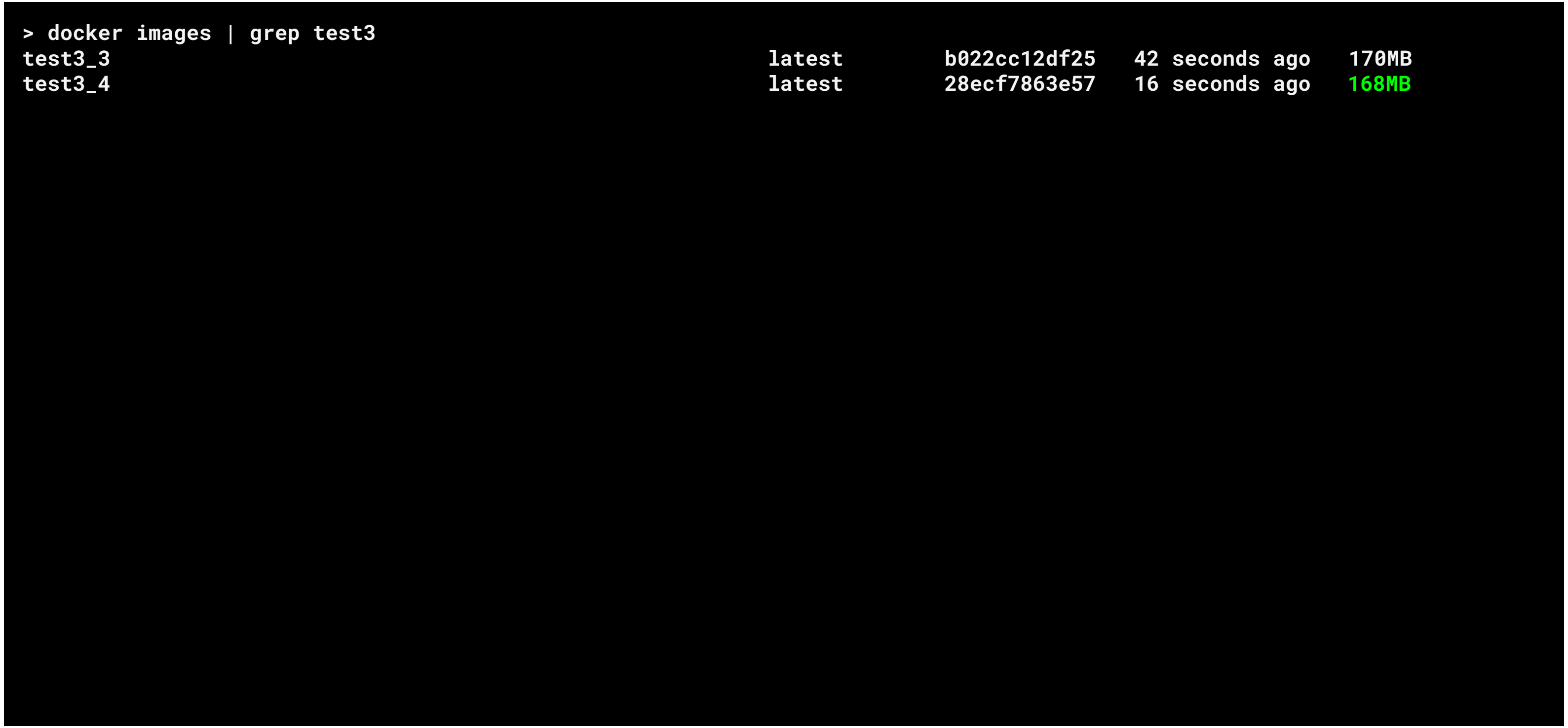
```
FROM ubuntu
COPY source/* .
RUN apt-get update
RUN apt-get install -y wget nodejs
RUN wget https://bit.ly/3urGNtE -O downloadedfile.tgz
RUN tar xvzf downloadedfile.tgz
RUN rm downloadedfile.tgz
RUN apt-get -y remove wget
ENTRYPOINT ["/usr/bin/node", "/main.js"]
```

Credentials & Confidentiality

- **Example (layer optimization)**

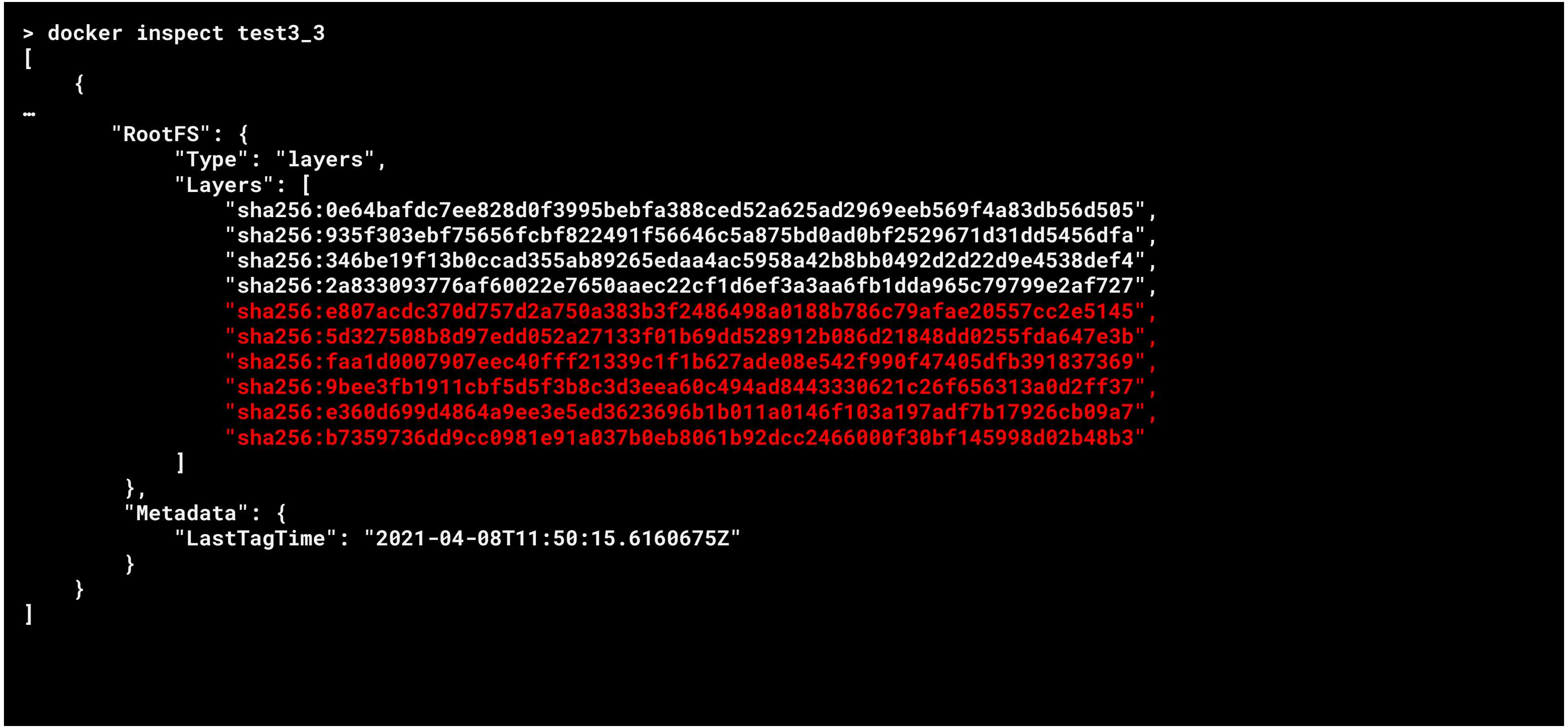
```
FROM ubuntu
COPY source/* .
RUN apt-get update && \
    apt-get install -y wget nodejs && \
    wget https://bit.ly/3urGNtE -O downloadedfile.tgz && \
    tar xvzf downloadedfile.tgz && \
    rm downloadedfile.tgz && \
    apt-get -y remove wget
ENTRYPOINT ["/usr/bin/node", "/main.js"]
```

Credentials & Confidentiality



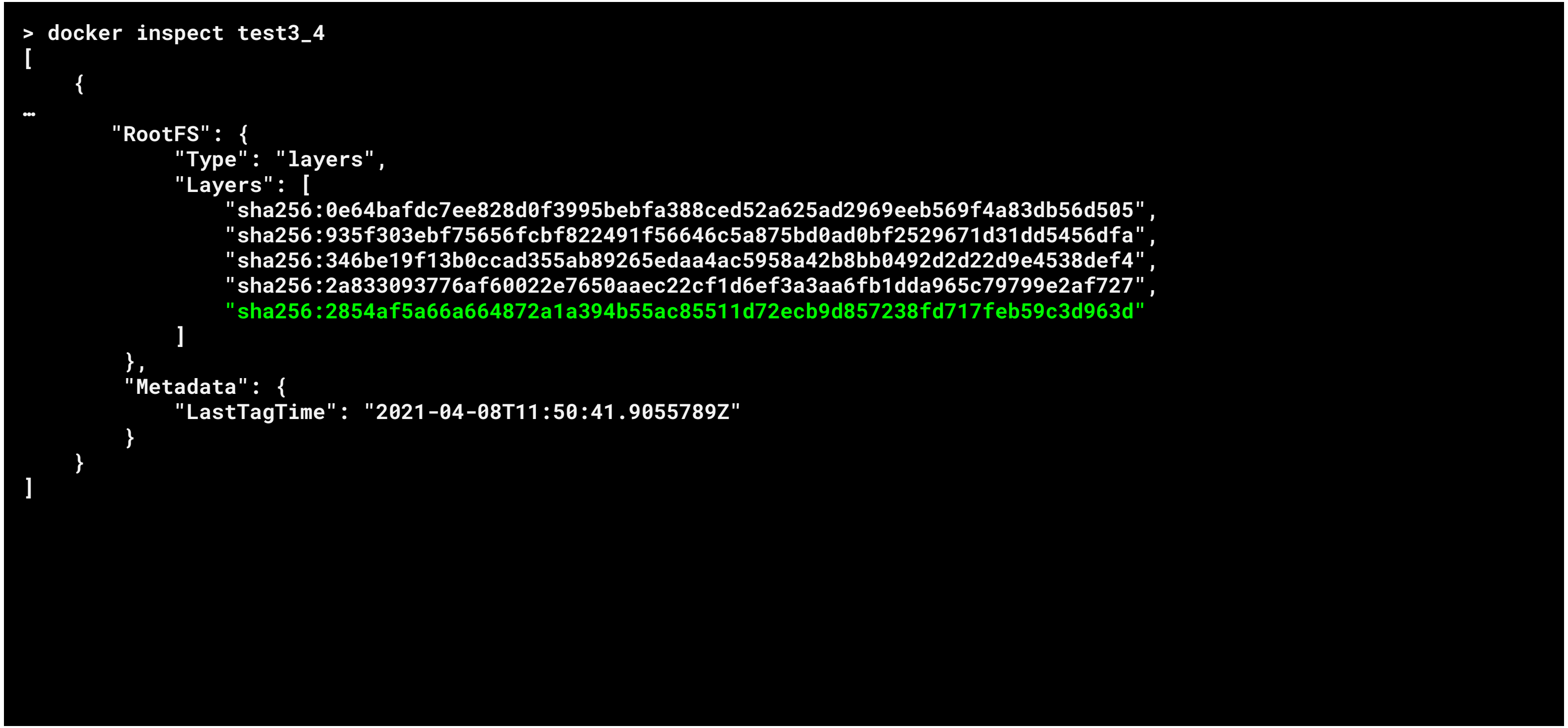
```
> docker images | grep test3
test3_3                                                         latest        b022cc12df25   42 seconds ago    170MB
test3_4                                                         latest        28ecf7863e57   16 seconds ago    168MB
```

Credentials & Confidentiality



```
> docker inspect test3_3
[
    {
…
        "RootFS": {
            "Type": "layers",
            "Layers": [
                "sha256:0e64bafdc7ee828d0f3995bebfa388ced52a625ad2969eeb569f4a83db56d505",
                "sha256:935f303ebf75656fcbf822491f56646c5a875bd0ad0bf2529671d31dd5456dfa",
                "sha256:346be19f13b0ccad355ab89265edaa4ac5958a42b8bb0492d2d22d9e4538def4",
                "sha256:2a833093776af60022e7650aaec22cf1d6ef3a3aa6fb1dda965c79799e2af727",
                "sha256:e807acdc370d757d2a750a383b3f2486498a0188b786c79afae20557cc2e5145",
                "sha256:5d327508b8d97edd052a27133f01b69dd528912b086d21848dd0255fda647e3b",
                "sha256:faa1d0007907eec40fff21339c1f1b627ade08e542f990f47405dfb391837369",
                "sha256:9bee3fb1911cbf5d5f3b8c3d3eea60c494ad8443330621c26f656313a0d2ff37",
                "sha256:e360d699d4864a9ee3e5ed3623696b1b011a0146f103a197adf7b17926cb09a7",
                "sha256:b7359736dd9cc0981e91a037b0eb8061b92dcc2466000f30bf145998d02b48b3"
            ]
        },
        "Metadata": {
            "LastTagTime": "2021-04-08T11:50:15.6160675Z"
        }
    }
]
```

Reduce attack surface



```
> docker inspect test3_4
[
    {
…
       "RootFS": {
            "Type": "layers",
            "Layers": [
                "sha256:0e64bafdc7ee828d0f3995bebfa388ced52a625ad2969eeb569f4a83db56d505",
                "sha256:935f303ebf75656fcbf822491f56646c5a875bd0ad0bf2529671d31dd5456dfa",
                "sha256:346be19f13b0ccad355ab89265edaa4ac5958a42b8bb0492d2d22d9e4538def4",
                "sha256:2a833093776af60022e7650aaec22cf1d6ef3a3aa6fb1dda965c79799e2af727",
                "sha256:2854af5a66a664872a1a394b55ac85511d72ecb9d857238fd717feb59c3d963d"
            ]
        },
        "Metadata": {
            "LastTagTime": "2021-04-08T11:50:41.9055789Z"
        }
    }
]
```

Credentials & Confidentiality

- **Example (cache optimization)**

```
FROM ubuntu
~~COPY source/* .~~
RUN apt-get update && \
    apt-get install -y wget nodejs && \
    wget https://bit.ly/3urGNtE -O downloadedfile.tgz && \
    tar xvzf downloadedfile.tgz && \
    rm downloadedfile.tgz && \
    apt-get -y remove wget
COPY source/* .
ENTRYPOINT ["/usr/bin/node", "/main.js"]
```

Credentials & Confidentiality

- **Don't**: Leak files from the build context

```
docker build -t myimage .
```

  - The "." parameter is the build context
  - All the files in the build context are sent to the docker daemon
    - You can copy confidential or unnecessary files into the container, like configuration files, credentials, backups, lock files, temporary files, sources, subfolders, dotfiles, etc.
  - COPY and ADD commands work from the build context

Credentials & Confidentiality

- **Example**

```
docker build -f Dockerfile -t myimage .
```

```
...
COPY . /my-app
```

This would copy everything inside the build context, which for the “.” example, includes the Dockerfile itself.

Credentials & Confidentiality

Good practices:

- Use a **clean build context**

```
docker build -f Dockerfile -t myimage files/
```

- Use **.dockerignore** file
- Prefer **COPY over ADD**, and **avoid wildcards**
  - COPY is more explicit, more predictable and less error prone
  - ADD can add files from a URL or from a .tar file

4. Linting & Scanning

Linting

- Tools like [Haskell Dockerfile Linter]() (hadolint) can detect bad practices in your Dockerfile, and even expose issues inside the shell commands executed by the RUN instruction.
- Image scanners (like Sysdig's) are also capable of detecting bad practices via customizable rules
- **Automate**: Consider incorporating such a tool in your CI pipelines.

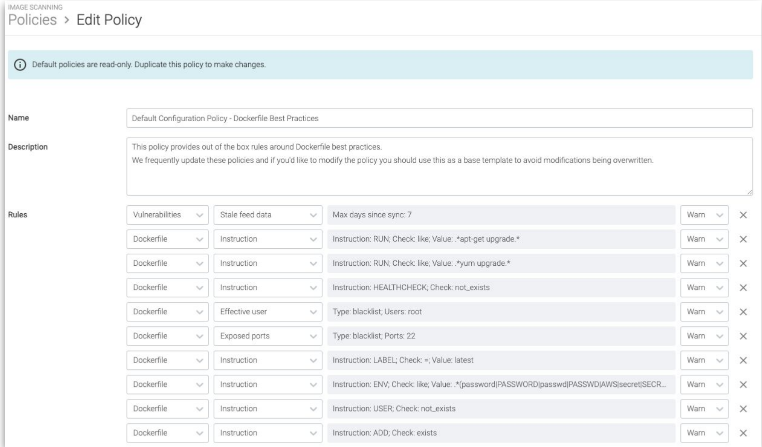


IMAGE SCANNING

Policies › Edit Policy

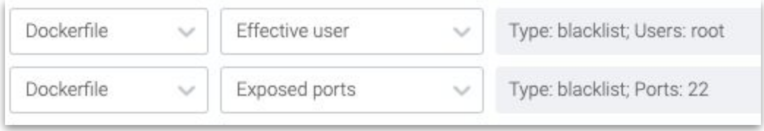
ⓘ Default policies are read-only. Duplicate this policy to make changes.

Name: Default Configuration Policy - Dockerfile Best Practices

Description: This policy provides out of the box rules around Dockerfile best practices.
We frequently update these policies and if you'd like to modify the policy you should use this as a base template to avoid modifications being overwritten.

Rules

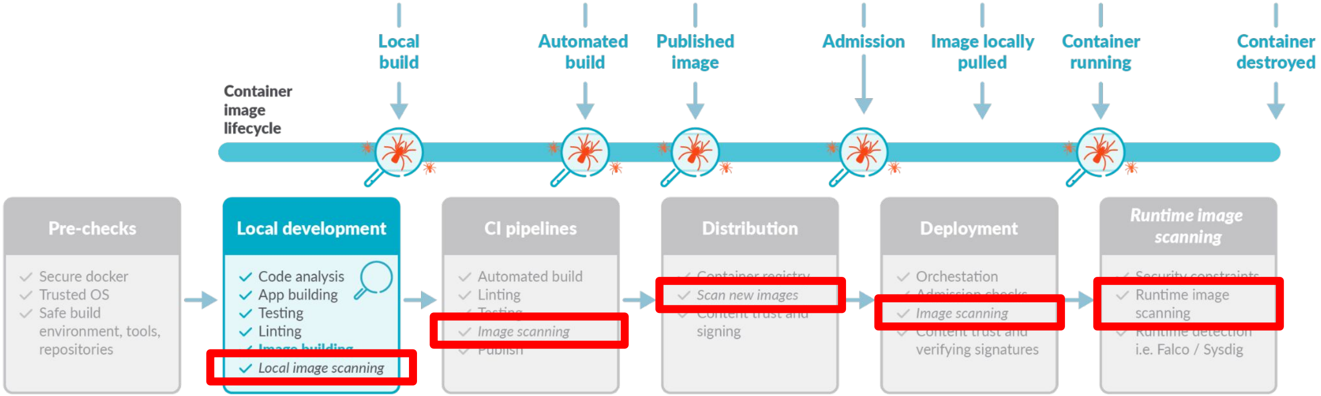
Vulnerabilities	Stale feed data	Max days since sync: 7	Warn
Dockerfile	Instruction	Instruction: RUN; Check: like; Value: .*apt-get upgrade.*	Warn
Dockerfile	Instruction	Instruction: RUN; Check: like; Value: .*yum upgrade.*	Warn
Dockerfile	Instruction	Instruction: HEALTHCHECK; Check: not_exists	Warn
Dockerfile	Effective user	Type: blacklist; Users: root	Warn
Dockerfile	Exposed ports	Type: blacklist; Ports: 22	Warn
Dockerfile	Instruction	Instruction: LABEL; Check: =; Value: latest	Warn
Dockerfile	Instruction	Instruction: ENV; Check: like; Value: .*(password|PASSWORD|passwd|PASSWD|AWS|secret|SECR...	Warn
Dockerfile	Instruction	Instruction: USER; Check: not_exists	Warn
Dockerfile	Instruction	Instruction: ADD; Check: exists	Warn



Dockerfile	Effective user	Type: blacklist; Users: root
Dockerfile	Exposed ports	Type: blacklist; Ports: 22

Image Scanning

- **Image scanning** can be implemented at different stages
  - Detect bad practices and known vulnerabilities
  - The earlier the scan is performed, the better



Container image lifecycle

Local build

Automated build

Published image

Admission

Image locally pulled

Container running

Container destroyed

Pre-checks
- ✓ Secure docker
- ✓ Trusted OS
- ✓ Safe build environment, tools, repositories

Local development
- ✓ Code analysis
- ✓ App building
- ✓ Testing
- ✓ Linting
- ✓ Image building
- ✓ *Local image scanning*

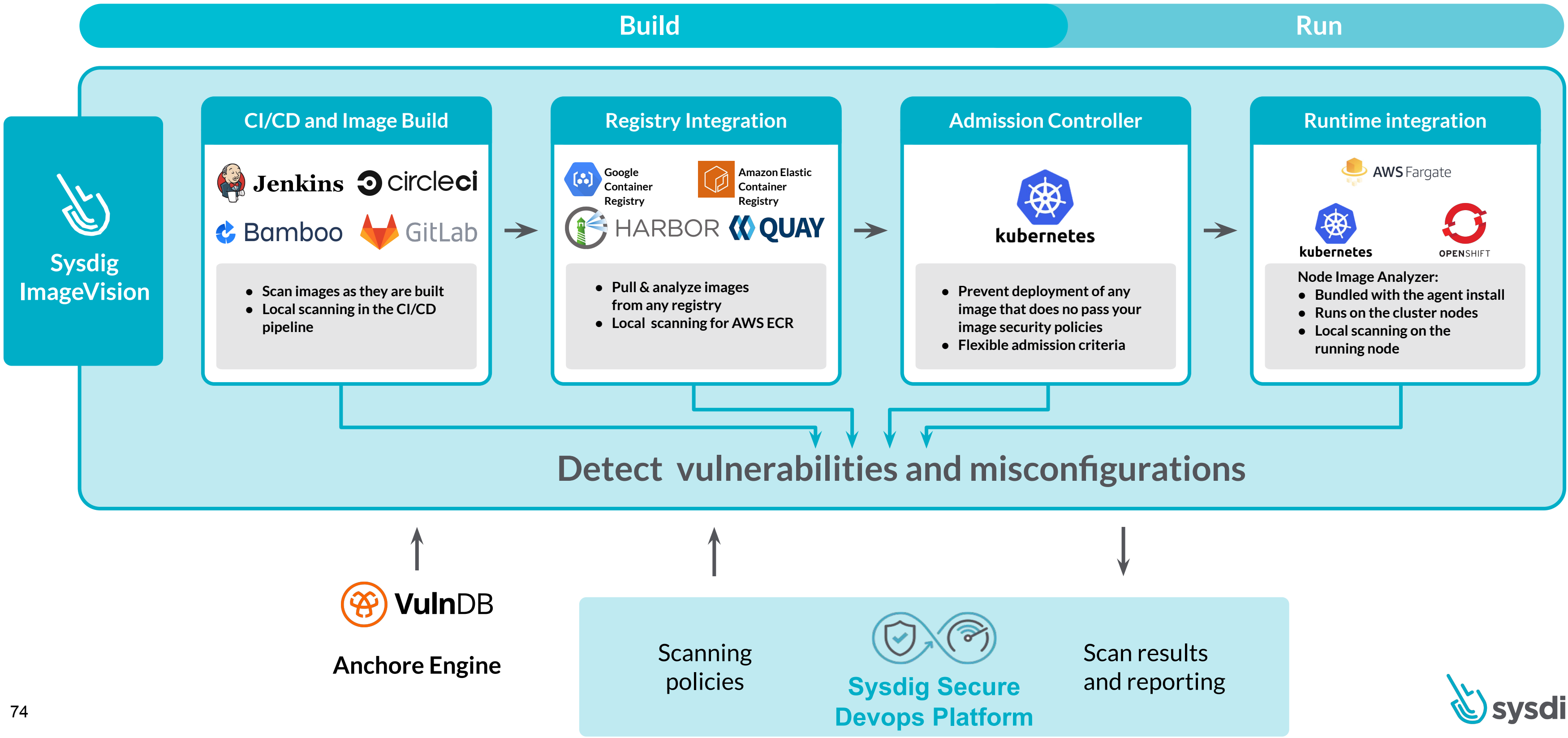
CI pipelines
- ✓ Automated build
- ✓ Linting
- ✓ Testing
- ✓ *Image scanning*
- ✓ Publish

Distribution
- ✓ Container registry
- ✓ *Scan new images*
- ✓ Content trust and signing

Deployment
- ✓ Orchestation
- ✓ Admission checks
- ✓ *Image scanning*
- ✓ Content trust and verifying signatures

Runtime image scanning
- ✓ Security constraints
- ✓ Runtime image scanning
- ✓ Runtime detection i.e. Falco / Sysdig

Image Scanning



Build

Run

Sysdig ImageVision

CI/CD and Image Build

- Scan images as they are built
- Local scanning in the CI/CD pipeline

Registry Integration

- Pull & analyze images from any registry
- Local scanning for AWS ECR

Admission Controller

- Prevent deployment of any image that does no pass your image security policies
- Flexible admission criteria

Runtime integration

Node Image Analyzer:

- Bundled with the agent install
- Runs on the cluster nodes
- Local scanning on the running node

Detect vulnerabilities and misconfigurations

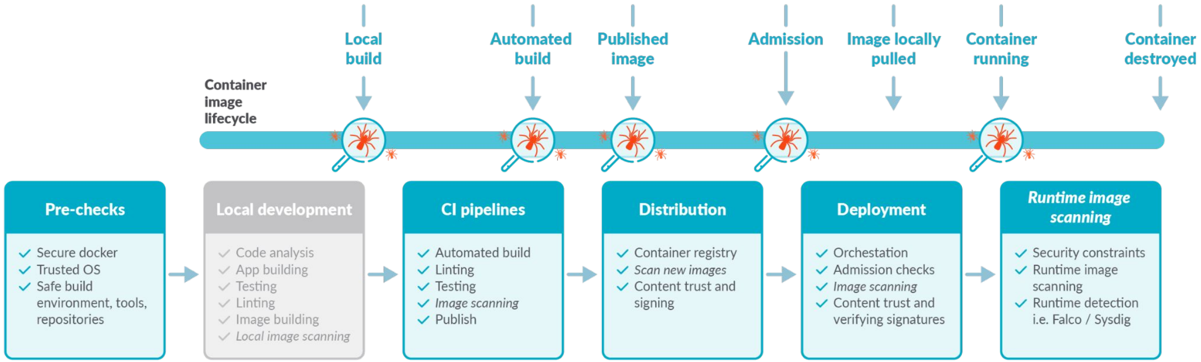
VulnDB

Anchore Engine

Scanning policies

Sysdig Secure Devops Platform

Scan results and reporting



Container image lifecycle

Local build → Automated build → Published image → Admission → Image locally pulled → Container running → Container destroyed

Pre-checks

- ✓ Secure docker
- ✓ Trusted OS
- ✓ Safe build environment, tools, repositories

Local development

- ✓ Code analysis
- ✓ App building
- ✓ Testing
- ✓ Linting
- ✓ Image building
- ✓ *Local image scanning*

CI pipelines

- ✓ Automated build
- ✓ Linting
- ✓ Testing
- ✓ *Image scanning*
- ✓ Publish

Distribution

- ✓ Container registry
- ✓ *Scan new images*
- ✓ Content trust and signing

Deployment

- ✓ Orchestation
- ✓ Admission checks
- ✓ *Image scanning*
- ✓ Content trust and verifying signatures

Runtime image scanning

- ✓ Security constraints
- ✓ Runtime image scanning
- ✓ Runtime detection i.e. Falco / Sysdig

5. Beyond image building

Beyond image building

So far we talked about "making" coffee...

Beyond image building

Now it is time to enjoy our cup of coffee

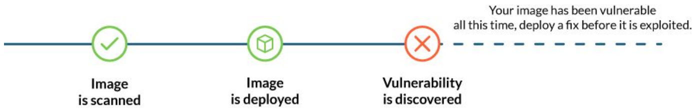
Beyond image building - runtime

- Remember **unnecessary privileges**?
  - The **orchestrator** or **runtime** environment (i.e., docker run, kubernetes, etc.) has the last word on who is the running container effective user.
  - Avoid **running your environment as root**
  - Openshift and some Kubernetes clusters will apply **restrictive policies by default**, **preventing root** containers from running or using a **random UID**

Beyond image building - runtime

- **Restrict application capabilities** on runtime
  - In case your container is compromised, the range of action available to an attacker is limited
  - --cap-drop flag in Docker
  - securityContext.capabilities.drop in Kubernetes
  - AppArmor in Docker or Kubernetes
  - Seccomp in Docker or Kubernetes.

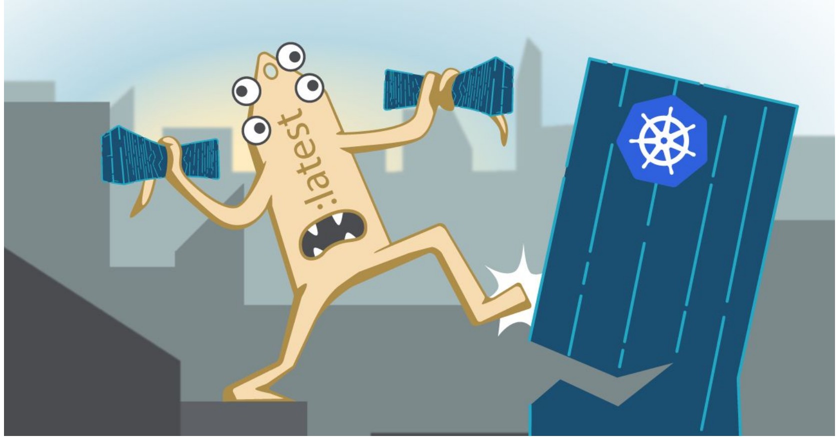
Beyond image building - runtime detection


Your image has been vulnerable
all this time, deploy a fix before it is exploited.
Image
is scanned
Image
is deployed
Vulnerability
is discovered

- **New vulnerabilities are discovered daily**
- **Remember: Update often + versioning strategy**
- Runtime detection is key
  - **Re-evaluate** scanned images to detect new applying vulnerabilities
  - **Runtime threat detection**. Falco can help.
  - **Respond** to suspicious activity. i.e. stop or pause container + **forensics**

Beyond image building - mutant tags

- **Beware!** *Attack of the mutant tags*



https://sysdig.com/blog/toctou-tag-mutability/

https://www.youtube.com/watch?v=j8K6EjOPhxs

Beyond image building - and more!

- The docker socket is a big privileged door into your host system
  - Make sure your /var/run/docker.sock has the correct permissions
  - If docker is exposed via TCP, make sure it is properly protected.

Beyond image building - and more!

- The docker socket is a big privileged door into your host system
  - Make sure your /var/run/docker.sock has the correct permissions
  - If docker is exposed via TCP, make sure it is properly protected.
- Use docker content trust, Docker notary, Harbor notary, or similar tools to digitally sign your images and then verify them on runtime.

Beyond image building - and more!

- The docker socket is a big privileged door into your host system
  - Make sure your /var/run/docker.sock has the correct permissions
  - If docker is exposed via TCP, make sure it is properly protected.
- Use docker content trust, Docker notary, Harbor notary, or similar tools to digitally sign your images and then verify them on runtime.
- Use Docker health-checks or Kubernetes livenessProbes
  - Critical for long running or persistent services in order to ensure they are healthy

Q & A

sysdig

Seeing is Securing