

# Kaniko

## Rootless Container Builds in 2026



slides

**Martin Zihlmann**  
DevOps @ Scandit



linkedin

# Running `docker build` in CI

To run `docker build` inside a CI container, the Docker sidecar needs `privileged`:

```
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: docker
    image: docker:dind
+   securityContext:
+     privileged: true
```

 setup

`privileged: true` gives the container **root-level access to the underlying host.**

Context: [MR!48](#)

`list host /dev`



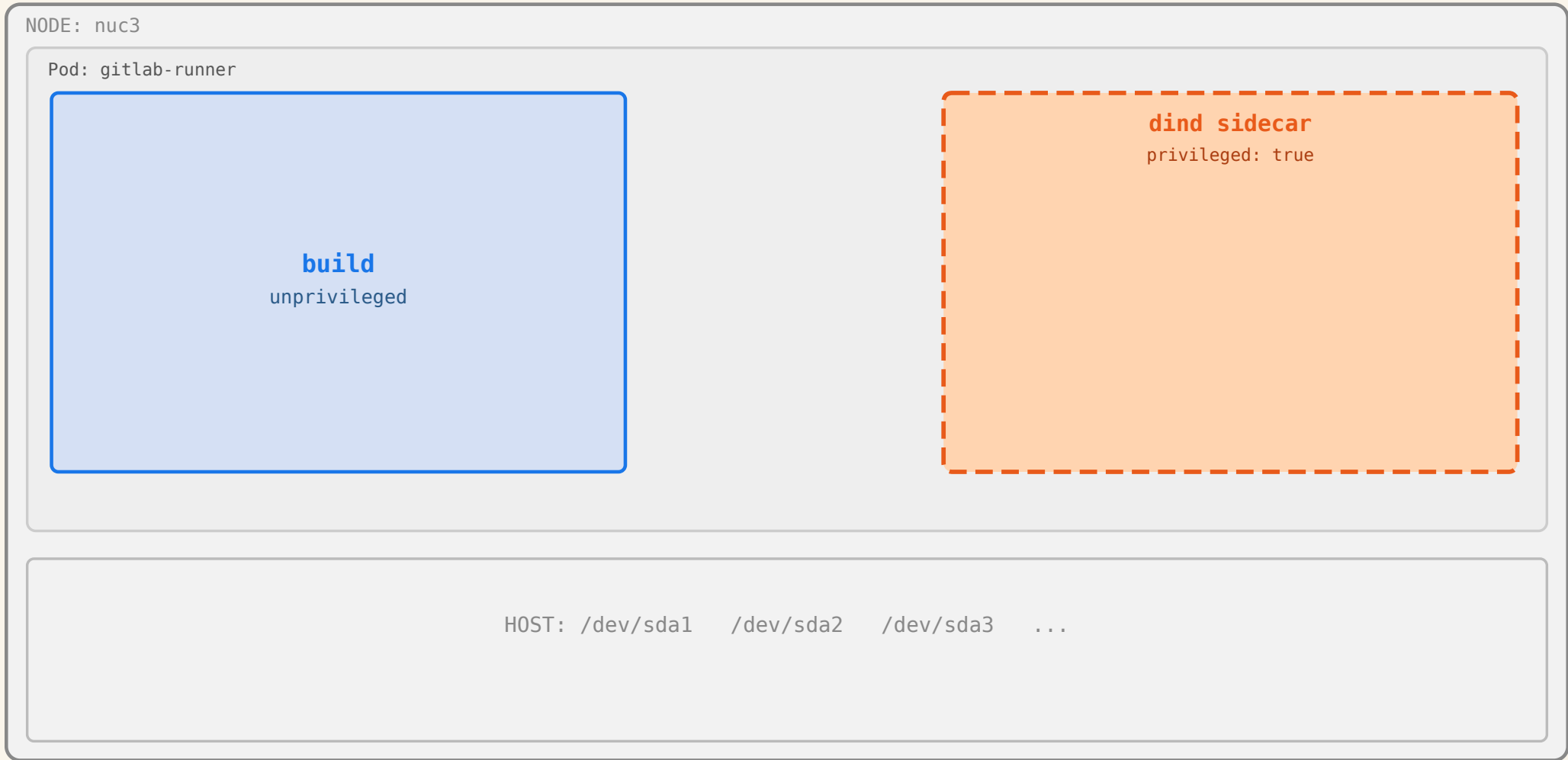
`read host shadow`



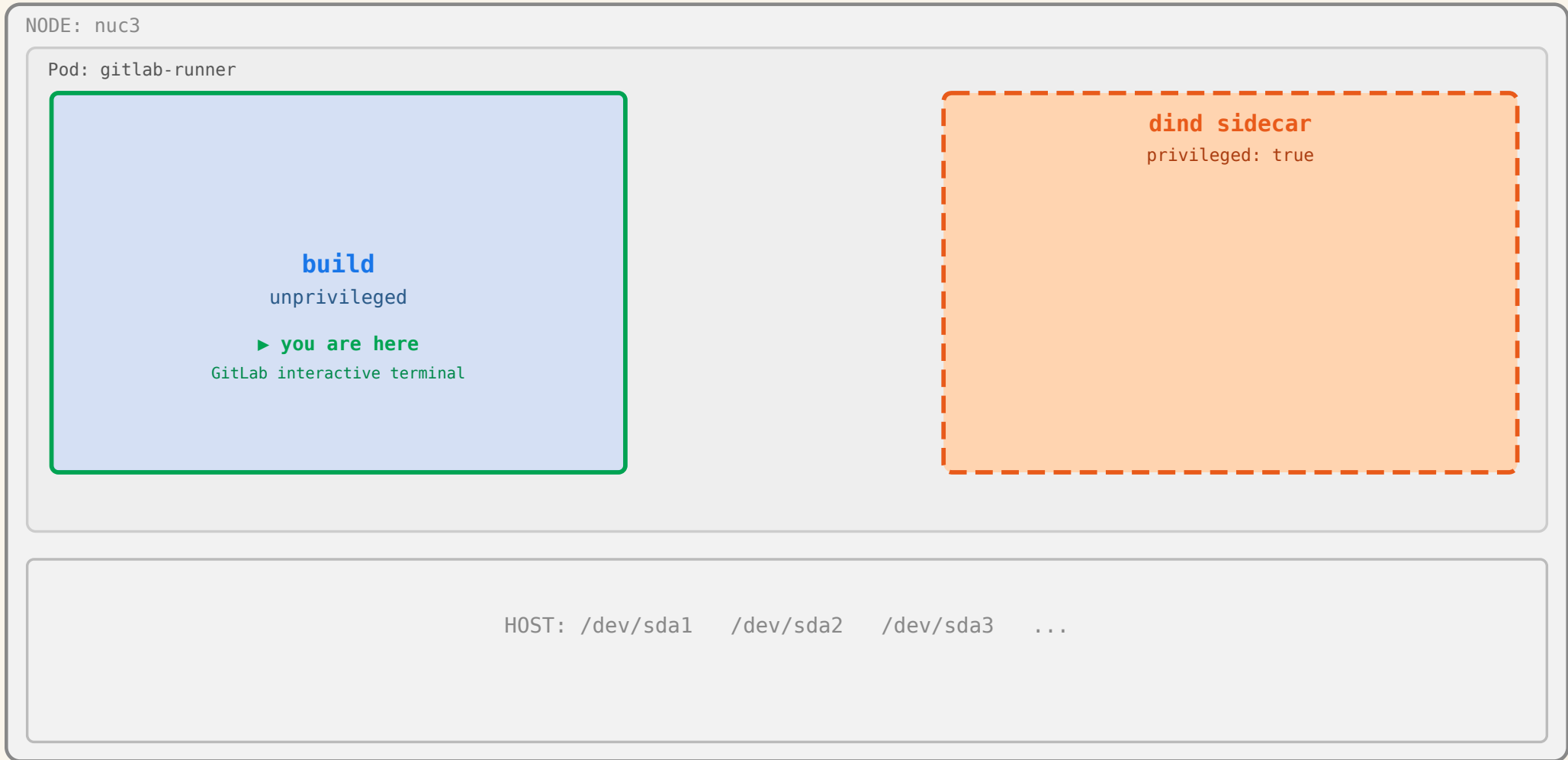
`inject SSH key`



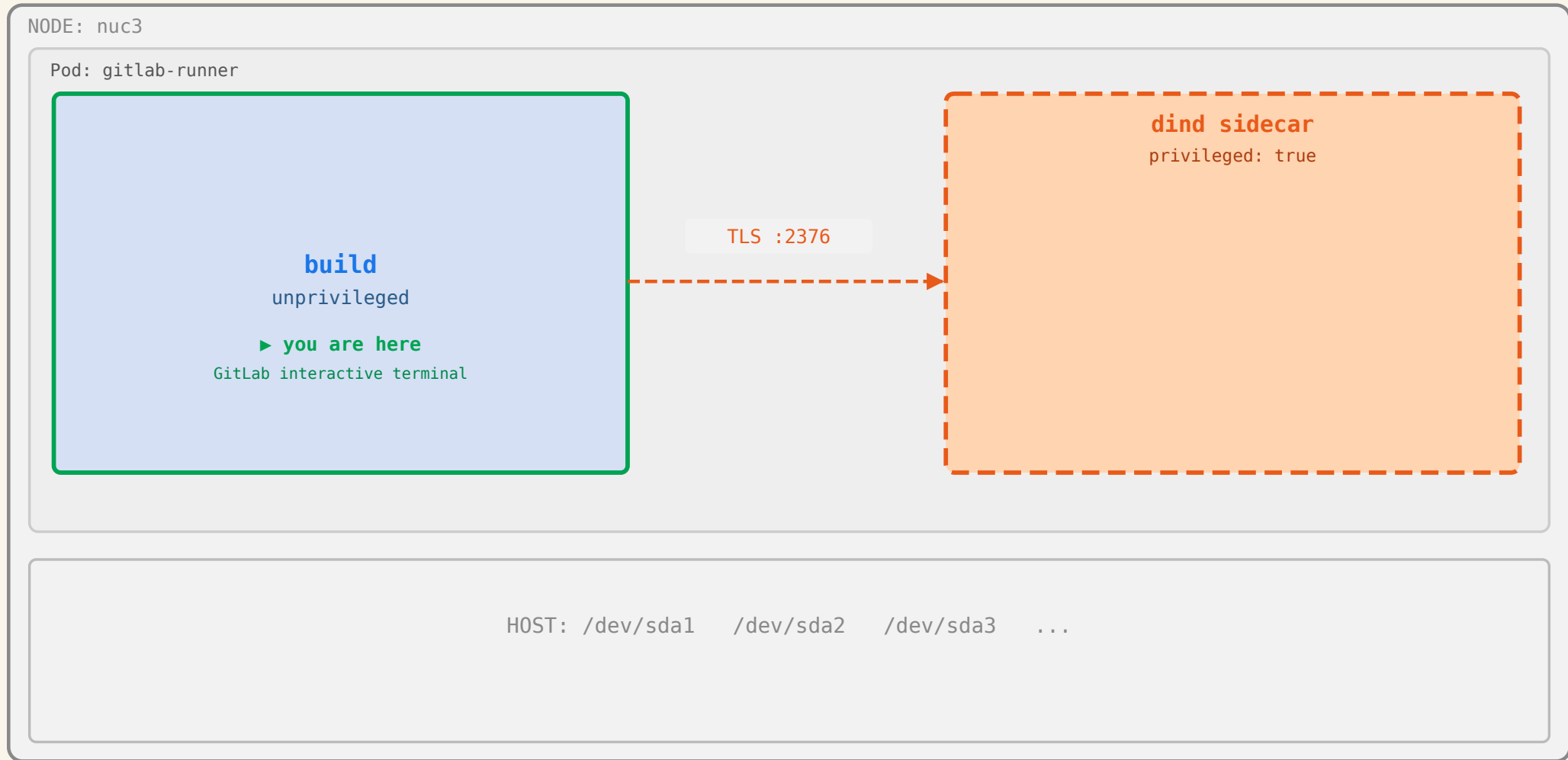
# The Attack Path



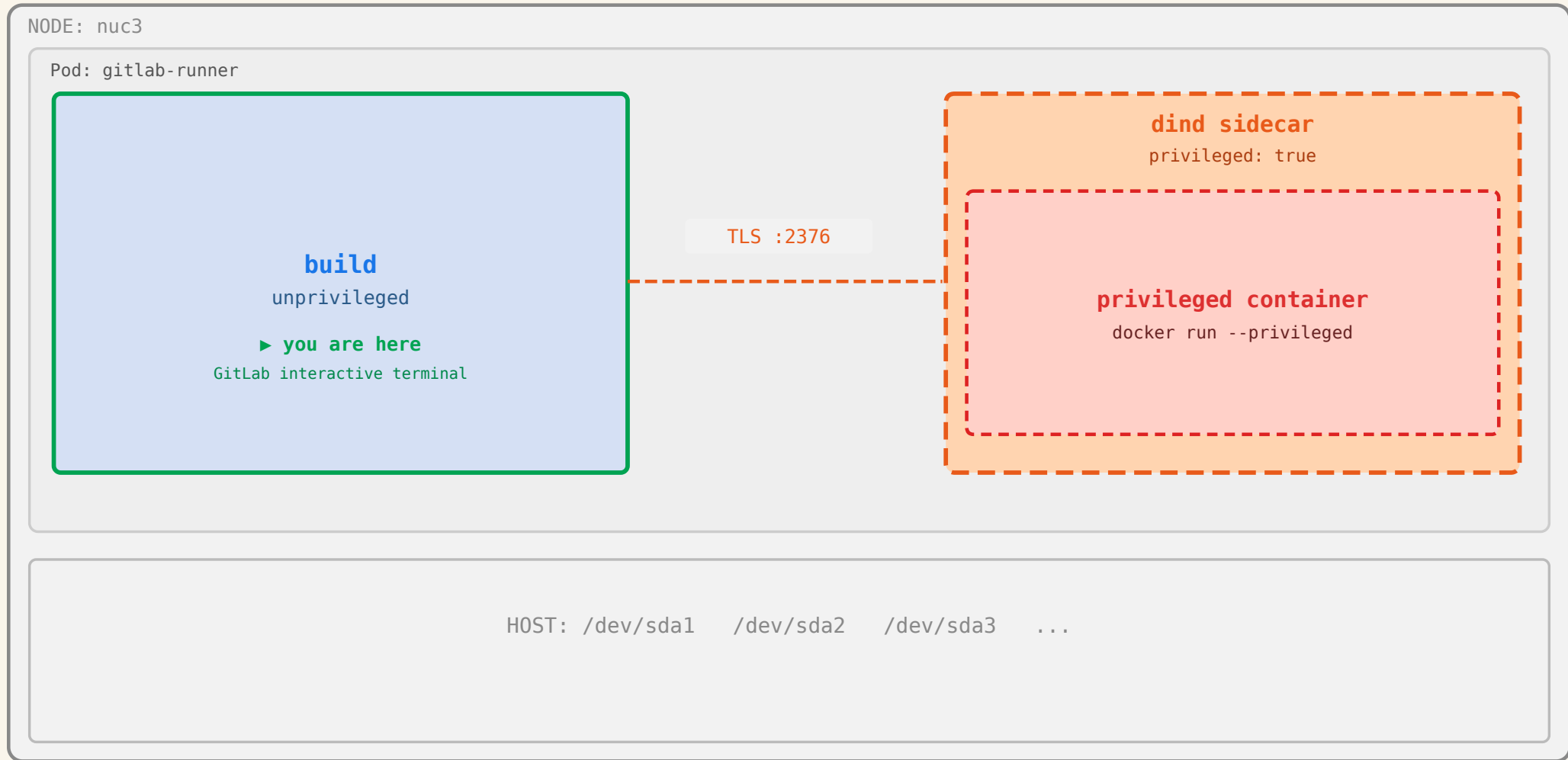
# The Attack Path



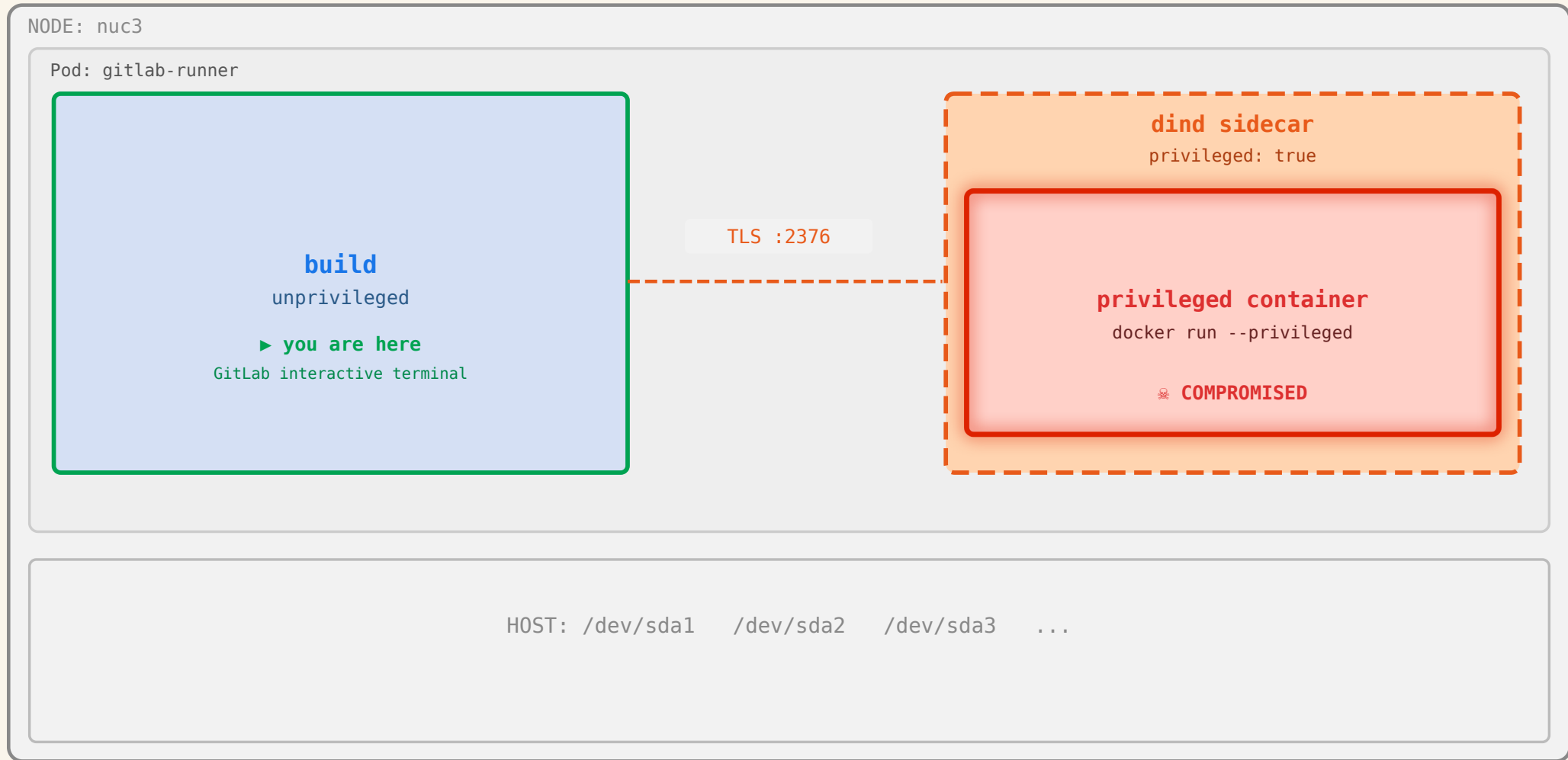
# The Attack Path



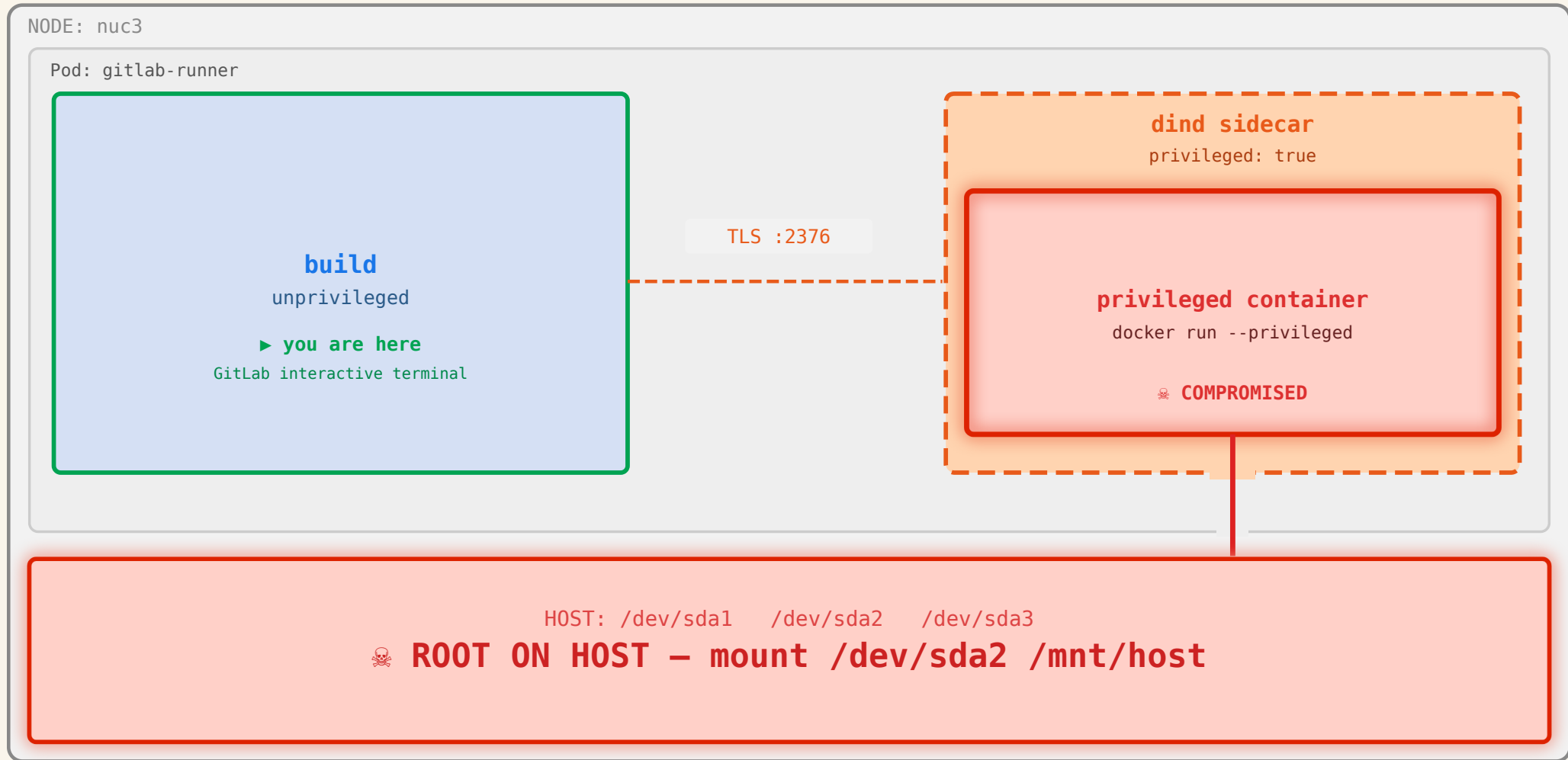
# The Attack Path



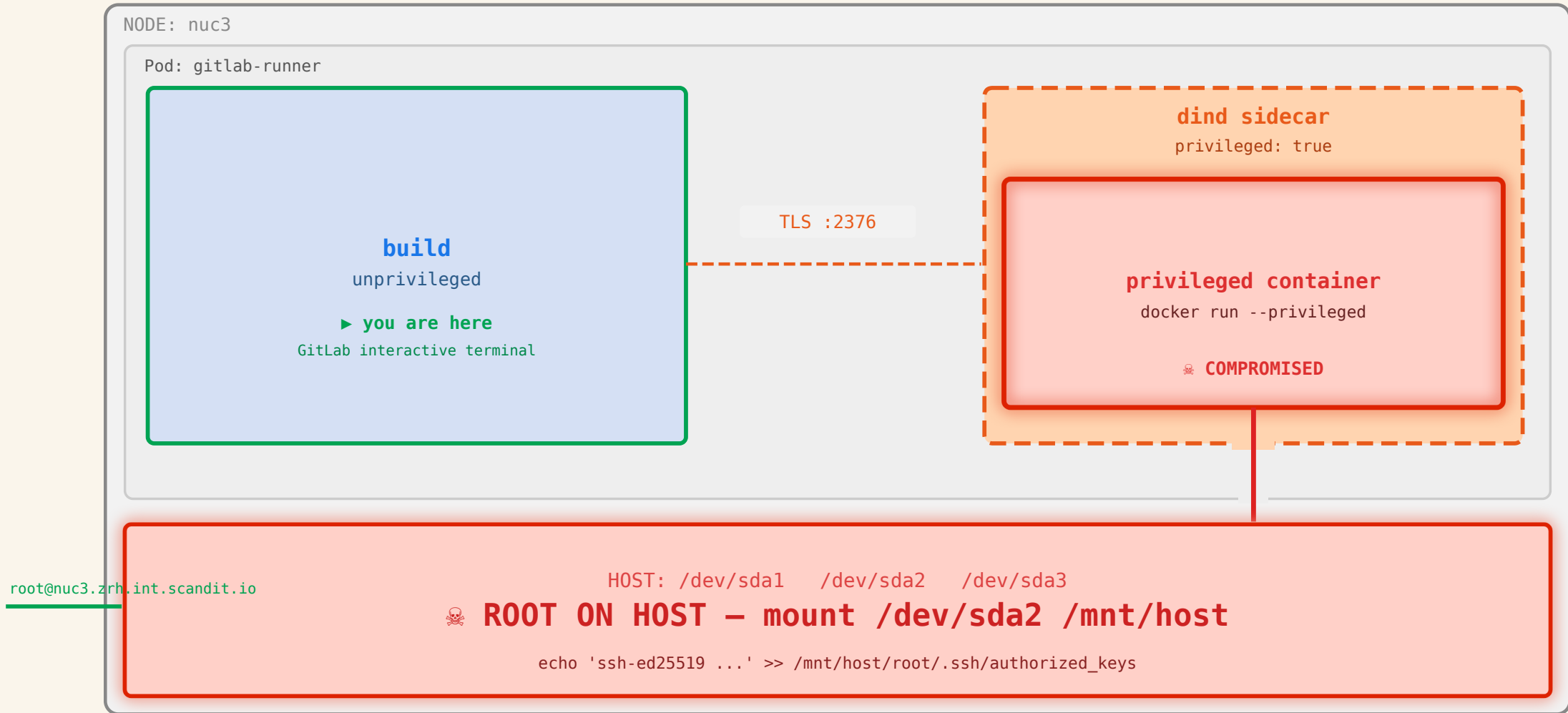
# The Attack Path



# The Attack Path



# The Attack Path



Kaniko: Rootless Container  
Builds in 2026

Poor Man's  
Dock

Kaniko - The Dockerfile  
Interpreter

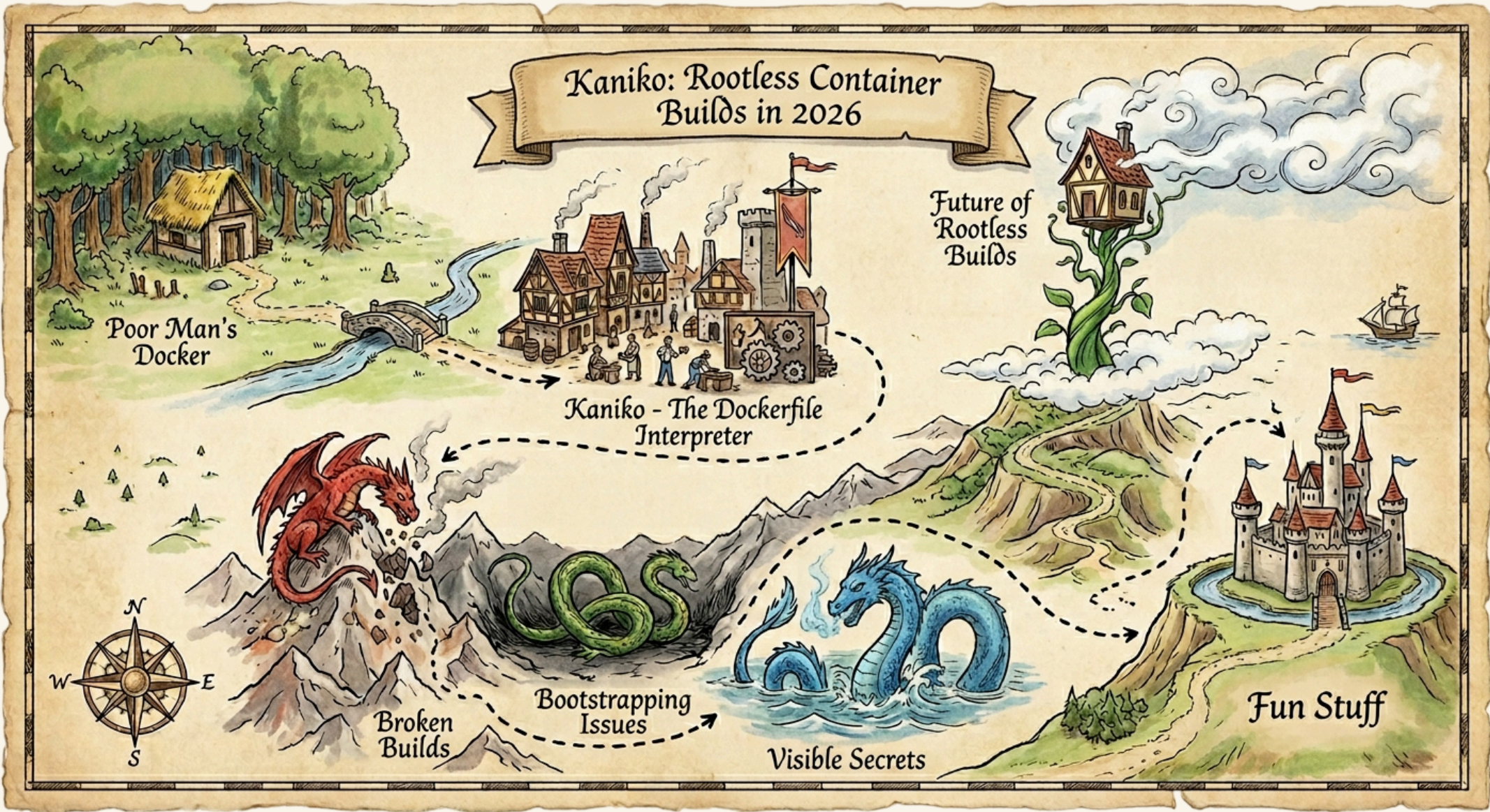
Future of  
Rootless  
Builds

Broken  
Builds

Bootstrapping  
Issues

Visible Secrets

Fun Stuff



PART 1

# Poor Man's Docker

a container under the hood is just the  
**unshare** syscall



## Creating a container — manually

```
sudo unshare --mount --pid --fork --mount-proc sh
```

 run

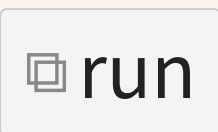
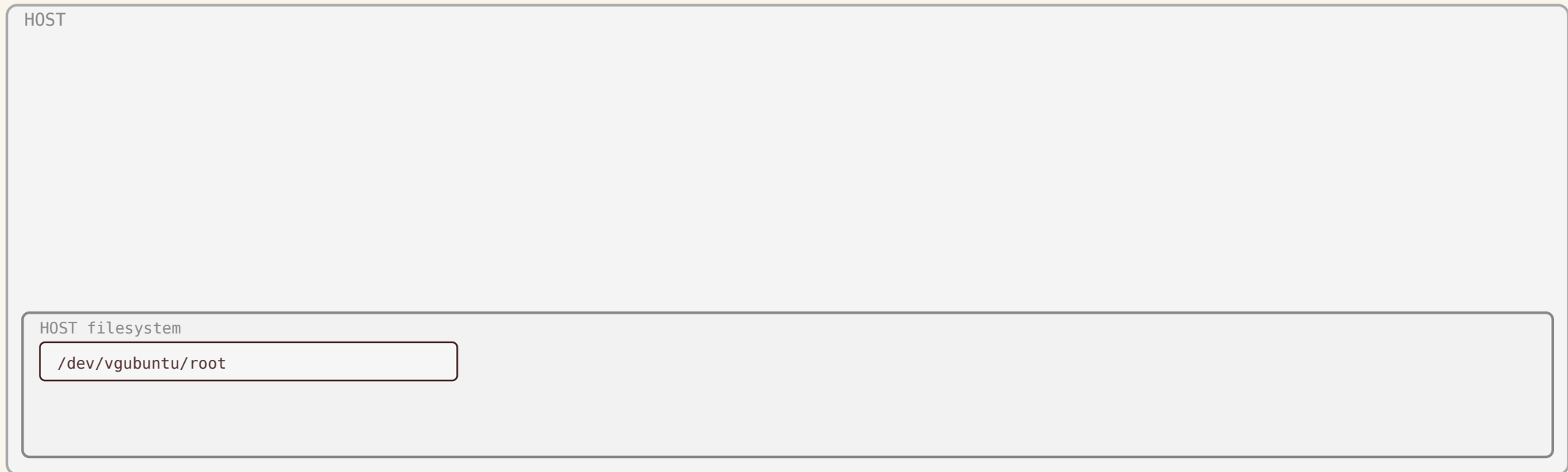
## Creating a container — manually

```
sudo unshare --mount --pid --fork --mount-proc sh
```

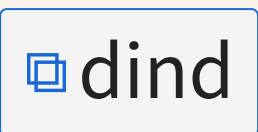
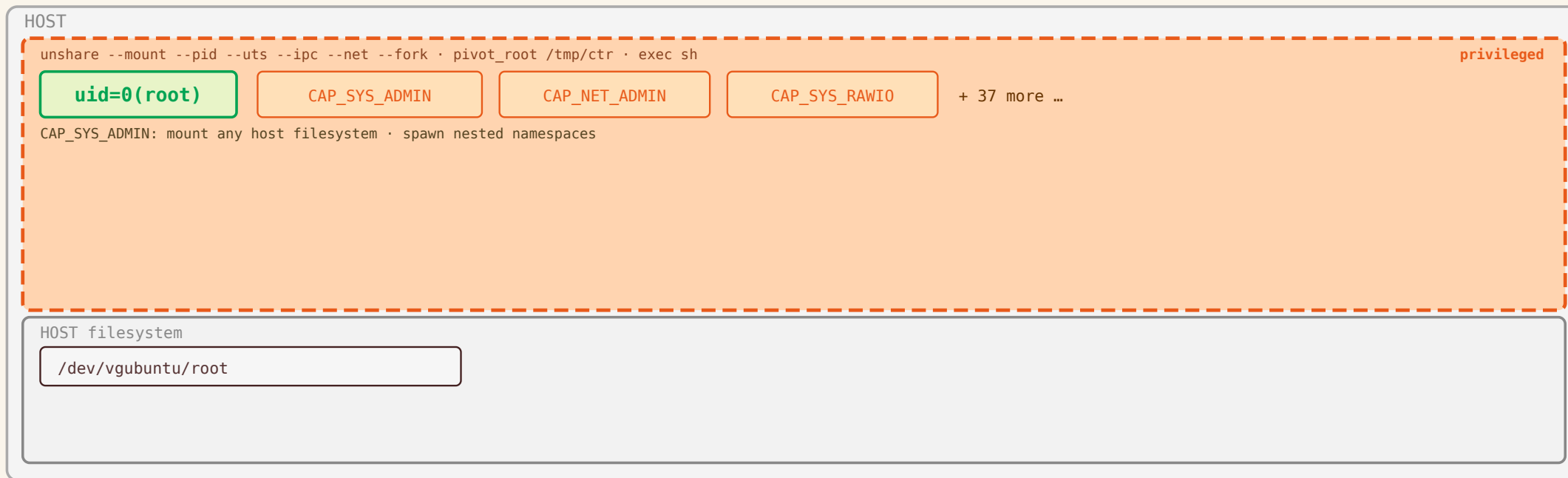
 run

root in the container is **root on the host**

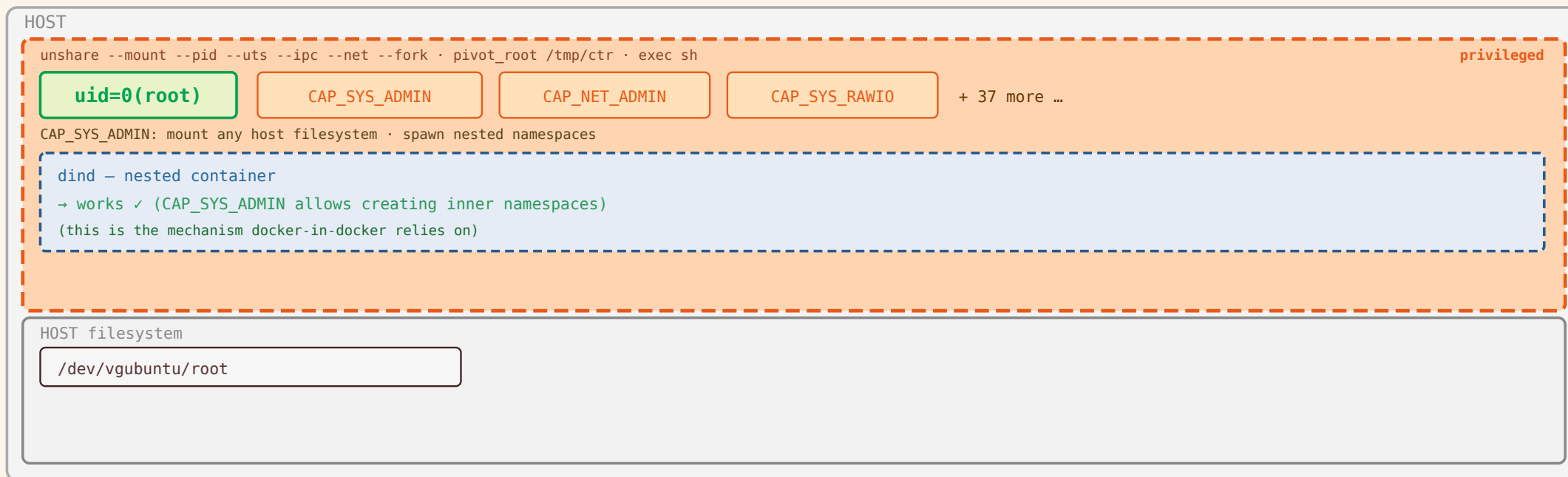
# Privileged container



# Privileged container

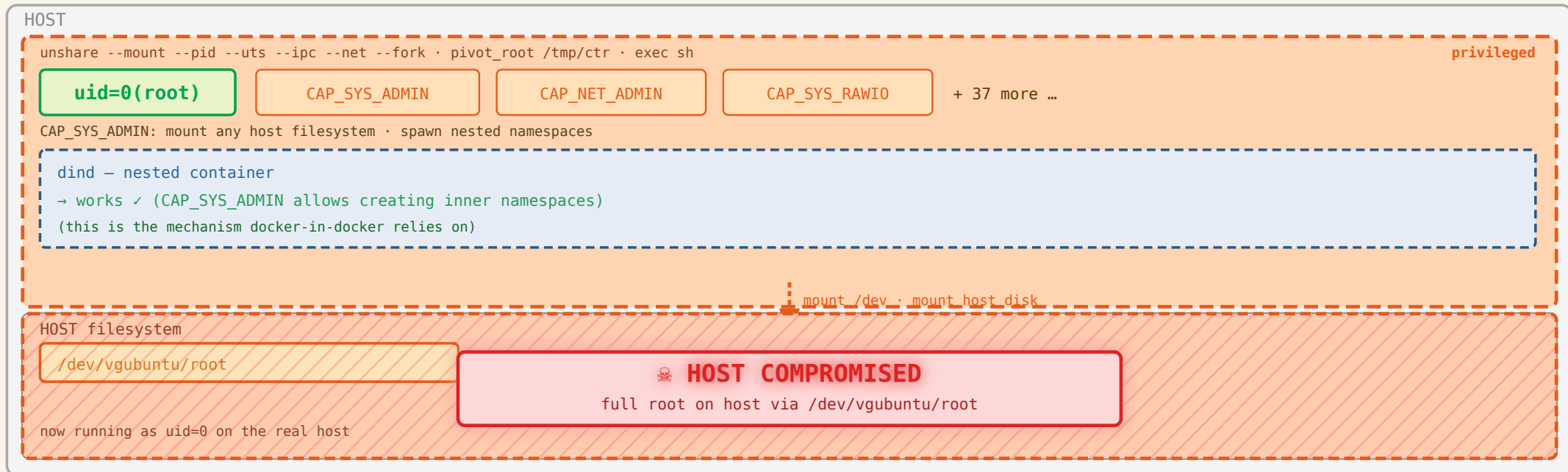


# Privileged container

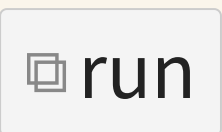
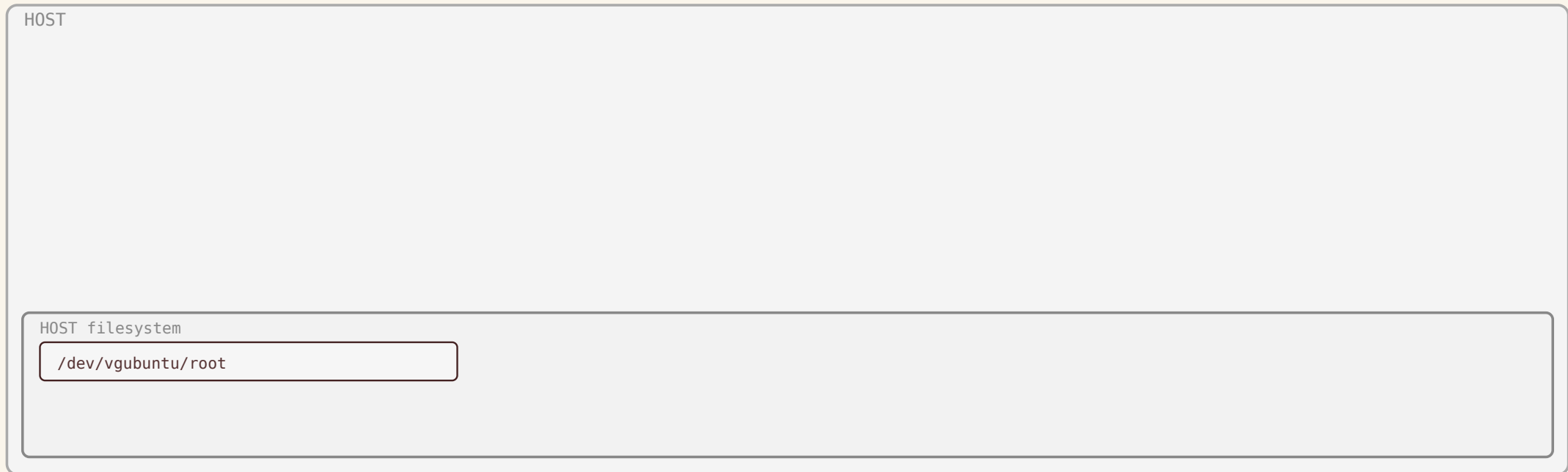


breakout

# Privileged container



# Unprivileged container



# Unprivileged container

HOST

```
unshare --mount --pid --uts --ipc --net --fork - pivot_root /tmp/ctr - setpriv --bounding-set=-all sh
```

**uid=0(root)** ~~CAP\_SYS\_ADMIN~~ ~~CAP\_NET\_ADMIN~~ ~~CAP\_SYS\_RAWIO~~ + 37 more ... **unprivileged**

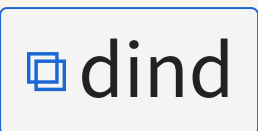
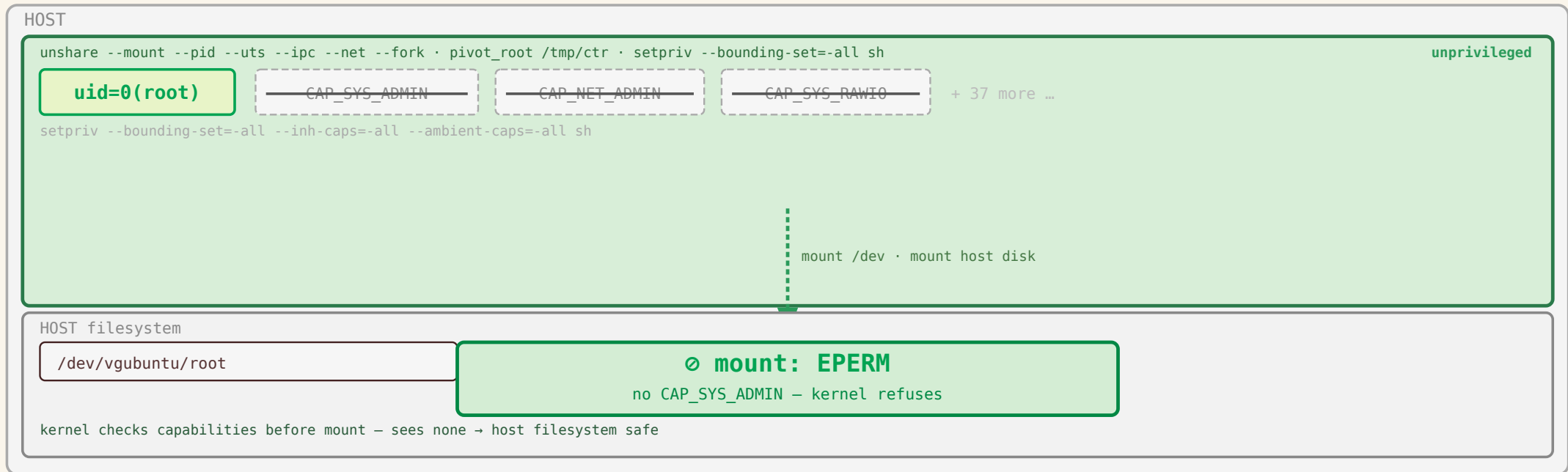
```
setpriv --bounding-set=-all --inh-caps=-all --ambient-caps=-all sh
```

HOST filesystem

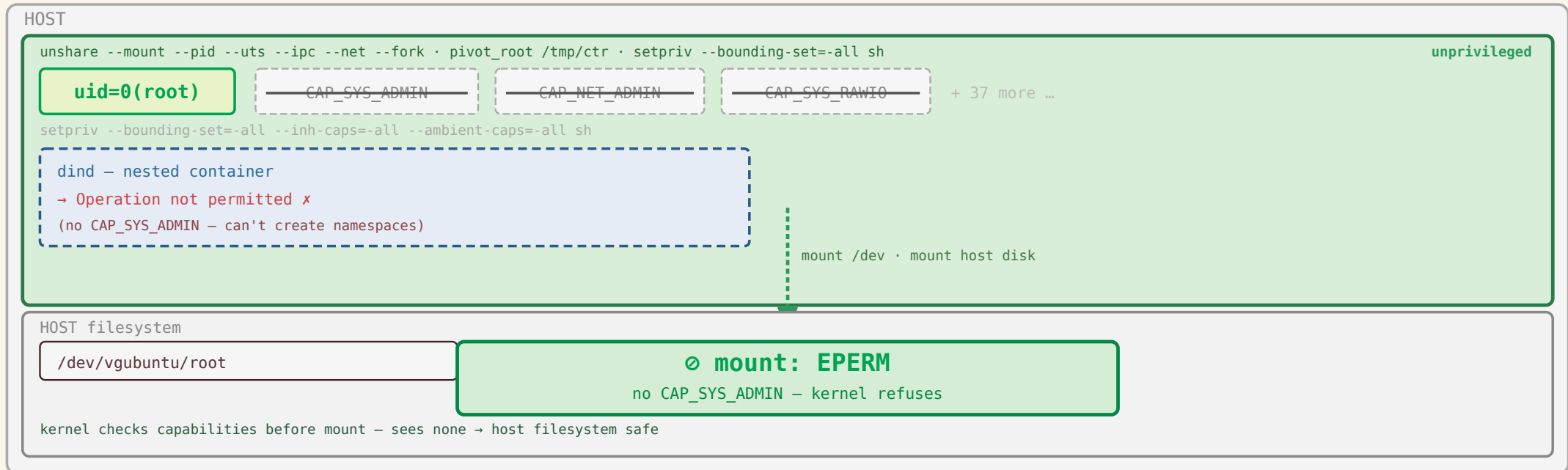
```
/dev/vgubuntu/root
```

 **breakout**

# Unprivileged container



# Unprivileged container



The cruel irony of

**CAP\_SYS\_ADMIN**

The same capability that lets you  
**create** containers

also lets you **escape** them.

The kernel can't tell which you intend – once granted, it applies to all uses.

# What if we skip the Docker daemon?

## Dockerfile

```
FROM alpine

RUN apk add --no-cache curl

RUN curl -L -o /tmp/crane.tar.gz \
  https://.../go-containerregistry_Linux_x86_64.tar.gz \
  && tar -xzf /tmp/crane.tar.gz \
  -C /usr/local/bin \
  && rm /tmp/crane.tar.gz

RUN crane version
```



## .gitlab-ci.yml


```
job:
  image: alpine
  script:
    - apk add --no-cache curl
    - |
      curl -L -o /tmp/crane.tar.gz \
        https://.../go-containerregistry_Linux_x86_64.tar.gz
      tar -xzf /tmp/crane.tar.gz \
        -C /usr/local/bin
      rm /tmp/crane.tar.gz

    - crane version
```




# What if we skip the Docker daemon?

```
Dockerfile
FROM alpine
RUN apk add --no-cache curl
RUN curl -L -o /tmp/crane.tar.gz \
  https://.../go-containerregistry_Linux_x86_64.tar.gz \
  && tar -xzf /tmp/crane.tar.gz \
  -C /usr/local/bin \
  && rm /tmp/crane.tar.gz
RUN crane version
```




same base

```
.gitlab-ci.yml
job:
  image: alpine
  script:
    - apk add --no-cache curl
    - |
      curl -L -o /tmp/crane.tar.gz \
        https://.../go-containerregistry_Linux_x86_64.tar.gz
      tar -xzf /tmp/crane.tar.gz \
        -C /usr/local/bin
      rm /tmp/crane.tar.gz
    - crane version
```




# What if we skip the Docker daemon?

```
Dockerfile
FROM alpine
RUN apk add --no-cache curl
RUN curl -L -o /tmp/crane.tar.gz \
  https://.../go-containerregistry_Linux_x86_64.tar.gz \
  && tar -xzf /tmp/crane.tar.gz \
  -C /usr/local/bin \
  && rm /tmp/crane.tar.gz
RUN crane version
```




same base

```
.gitlab-ci.yml
job:
  image: alpine
  script:
    - apk add --no-cache curl
    - |
      curl -L -o /tmp/crane.tar.gz \
        https://.../go-containerregistry_Linux_x86_64.tar.gz \
      tar -xzf /tmp/crane.tar.gz \
      -C /usr/local/bin
      rm /tmp/crane.tar.gz
    - crane version
```



# What if we skip the Docker daemon?

```
Dockerfile
FROM alpine
RUN apk add --no-cache curl
RUN curl -L -o /tmp/crane.tar.gz \
  https://.../go-containerregistry_Linux_x86_64.tar.gz \
  && tar -xzf /tmp/crane.tar.gz \
  -C /usr/local/bin \
  && rm /tmp/crane.tar.gz
RUN crane version
```



🚫 needs Docker daemon (privileged)

```
.gitlab-ci.yml
job:
  image: alpine
  script:
    - apk add --no-cache curl
    - |
      curl -L -o /tmp/crane.tar.gz \
        https://.../go-containerregistry_Linux_x86_64.tar.gz \
        tar -xzf /tmp/crane.tar.gz \
        -C /usr/local/bin
      rm /tmp/crane.tar.gz
    - crane version
```




✓ runs unprivileged today

same base

# What if we skip the Docker daemon?

```
Dockerfile
FROM alpine
RUN apk add --no-cache curl
RUN curl -L -o /tmp/crane.tar.gz \
  https://.../go-containerregistry_Linux_x86_64.tar.gz \
  && tar -xzf /tmp/crane.tar.gz \
  -C /usr/local/bin \
  && rm /tmp/crane.tar.gz
RUN crane version
```



🚫 needs Docker daemon (privileged)

```
.gitlab-ci.yml
job:
  image: alpine
  script:
    - apk add --no-cache curl
    - |
      curl -L -o /tmp/crane.tar.gz \
        https://.../go-containerregistry_Linux_x86_64.tar.gz \
      tar -xzf /tmp/crane.tar.gz \
      -C /usr/local/bin
      rm /tmp/crane.tar.gz
    - crane version
```



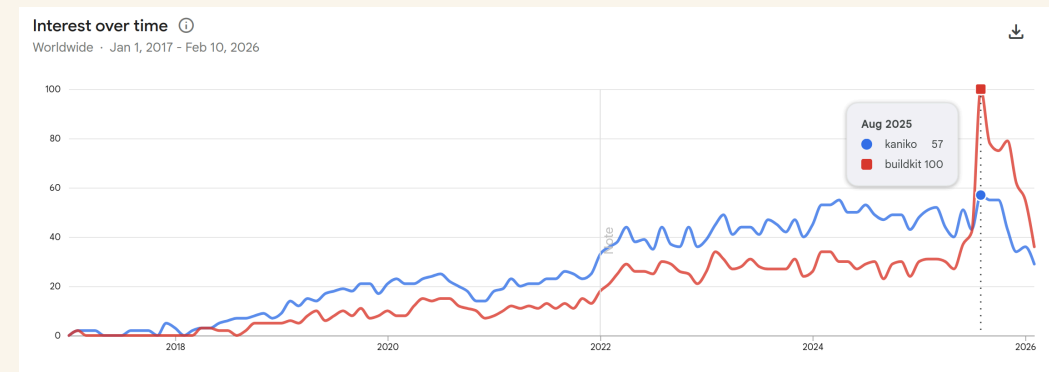
✓ runs unprivileged today

Same result – but one already works without Docker.  
Can we formalize the right side into a proper image builder?

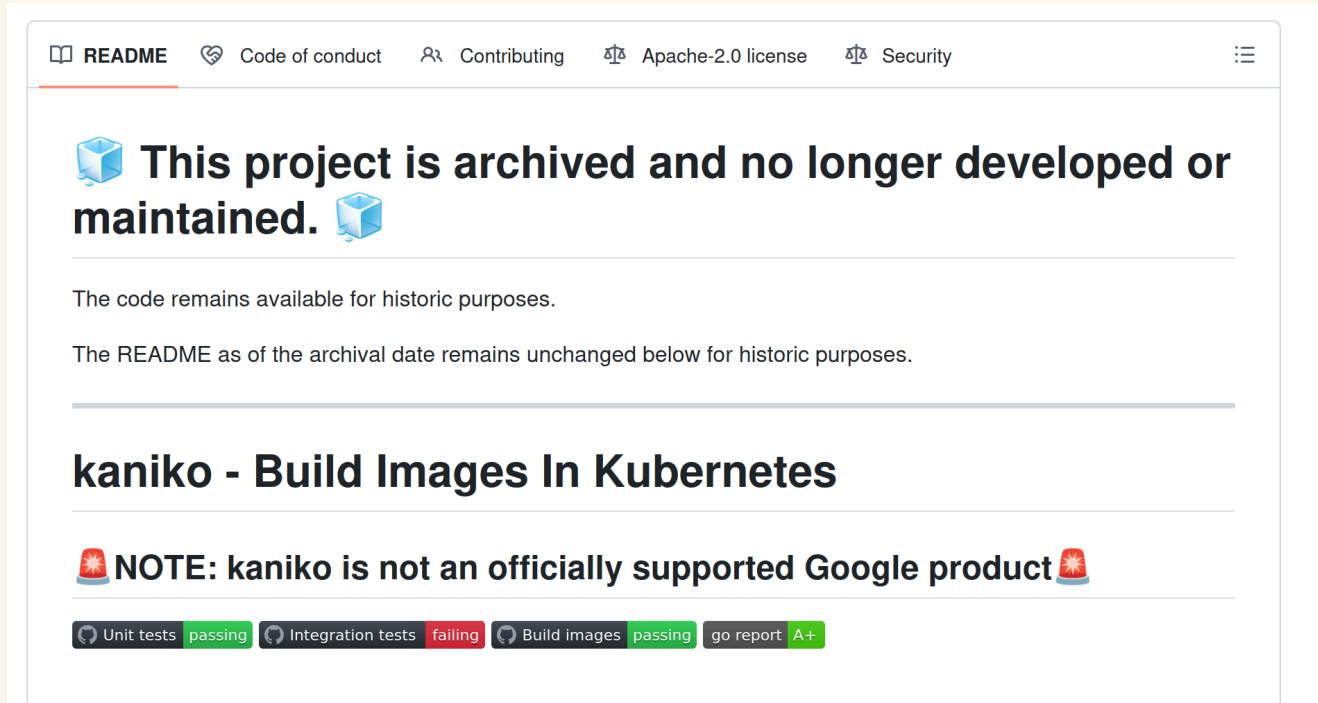
**What if we just write an interpreter for  
Dockerfiles?**

# Google's Brainchild

Kaniko started as a random experiment by **Priya Wadhwa** and **Dan Lorenc** at Google — just to prove that building a Docker image without privileges is possible.



# Then Google archived it.



The screenshot shows the GitHub README for the 'kaniko' project. At the top, there are navigation links: README (underlined), Code of conduct, Contributing, Apache-2.0 license, and Security. The main content area features a blue cube icon followed by the text: "This project is archived and no longer developed or maintained." Below this, it states: "The code remains available for historic purposes." and "The README as of the archival date remains unchanged below for historic purposes." The project title "kaniko - Build Images In Kubernetes" is displayed in a large, bold font. A red warning icon is followed by the text: "NOTE: kaniko is not an officially supported Google product". At the bottom, there is a status bar with four items: "Unit tests" (passing), "Integration tests" (failing), "Build images" (passing), and "go report" (A+).

So we forked it:

`github.com/osscontainertools/kaniko`

PART 2

# Kaniko — The Dockerfile Interpreter

# Parser

Parser

```
FROM alpine
```

```
RUN apk add curl
```

```
RUN echo "hello" \  
> /app.txt
```

# Interpreter

# Snapshotter

# Parser

```
Parser
▶ FROM alpine

RUN apk add curl

RUN echo "hello" \
> /app.txt
```

# Interpreter

# Snapshotter

# Parser

```
Parser  
FROM alpine  
RUN apk add curl  
RUN echo "hello" \  
> /app.txt
```

# Interpreter

# Snapshotter

# Parser

Parser

```
FROM alpine
```

```
RUN apk add curl
```

```
▶ RUN echo "hello" \  
> /app.txt
```

# Interpreter

# Snapshotter

# Parser

## Parser

```
FROM alpine
```

```
RUN apk add curl
```

```
RUN echo "hello" \  
> /app.txt
```

# Interpreter

## Interpreter

```
/busybox/  
/kaniko/
```

# Snapshotter

# Parser

Parser

► FROM alpine

RUN apk add curl

RUN echo "hello" \  
> /app.txt

# Interpreter

Interpreter

/busybox/  
/kaniko/

/bin/  
/etc/  
/lib/  
/usr/

# Snapshotter

# Parser

## Parser

FROM alpine

▶ RUN apk add curl

RUN echo "hello" \  
> /app.txt

# Interpreter

## Interpreter

```
/busybox/  
/kaniko/  
  
/bin/  
/etc/  
/lib/  
/usr/  
  
/usr/bin/curl  
+ /usr/lib/libcurl.so...
```

# Snapshotter

# Parser

## Parser

FROM alpine

RUN apk add curl

```
▶ RUN echo "hello" \  
> /app.txt
```

# Interpreter

## Interpreter

```
/busybox/  
/kaniko/  
  
/bin/  
/etc/  
/lib/  
/usr/  
/usr/bin/curl  
+ /usr/lib/libcurl.so...  
/app.txt
```

# Snapshotter

# Parser

## Parser

```
FROM alpine

RUN apk add curl

RUN echo "hello" \
  > /app.txt
```

# Interpreter

## Interpreter

```
/busybox/
/kaniko/

/bin/
/etc/
/lib/
/usr/
/usr/bin/curl
+ /usr/lib/libcurl.so...
/app.txt
```

# Snapshotter

## Snapshotter

# Parser

## Parser

► FROM alpine

RUN apk add curl

RUN echo "hello" \  
> /app.txt

# Interpreter

## Interpreter

```
/busybox/  
/kaniko/  
  
/bin/  
/etc/  
/lib/  
/usr/  
/usr/bin/curl  
+ /usr/lib/libcurl.so...  
/app.txt
```

# Snapshotter

## Snapshotter

```
Layer 0 · base  
FROM alpine  
/bin/ /etc/ /lib/ /usr/ ...
```

# Parser

## Parser

FROM alpine

▶ RUN apk add curl

RUN echo "hello" \  
> /app.txt

# Interpreter

## Interpreter

```
/busybox/  
/kaniko/  
  
/bin/  
/etc/  
/lib/  
/usr/  
/usr/bin/curl  
+ /usr/lib/libcurl.so...  
/app.txt
```

# Snapshotter

## Snapshotter

Layer 0 · base  
**FROM alpine**  
/bin/ /etc/ /lib/ /usr/ ...

Layer 1 · diff  
**RUN apk add curl**  
+/usr/bin/curl +/usr/lib/libcurl\*

# Parser

## Parser

FROM alpine

RUN apk add curl

▶ RUN echo "hello" \  
> /app.txt

# Interpreter

## Interpreter

```
/busybox/  
/kaniko/  
  
/bin/  
/etc/  
/lib/  
/usr/  
/usr/bin/curl  
+ /usr/lib/libcurl.so...  
/app.txt
```

# Snapshotter

## Snapshotter

Layer 0 · base  
**FROM alpine**  
/bin/ /etc/ /lib/ /usr/ ...

Layer 1 · diff  
**RUN apk add curl**  
+/usr/bin/curl +/usr/lib/libcurl\*

Layer 2 · diff  
**RUN echo "hello" > /app.txt**  
+/app.txt (6 B)

# Parser

## Parser

```
FROM alpine

RUN apk add curl

RUN echo "hello" \
  > /app.txt
```

# Interpreter

## Interpreter

```
/busybox/
/kaniko/

/bin/
/etc/
/lib/
/usr/
/usr/bin/curl
+ /usr/lib/libcurl.so...
/app.txt
```

# Snapshotter

## Snapshotter

```
Layer 0 · base
FROM alpine
/bin/ /etc/ /lib/ /usr/ ...
```

```
Layer 1 · diff
RUN apk add curl
+/usr/bin/curl +/usr/lib/libcurl*
```

```
Layer 2 · diff
RUN echo "hello" > /app.txt
+/app.txt (6 B)
```

## myimage:latest

image = union of all layer diffs  
config + 3 layers (OCI manifest)

# There is no container



PART 3

# Here be Dragons





# Custom Kaniko Images

Kaniko is a **container image**, not an executable. Running it from a different base can have weird effects.

We want crane in our kaniko image — let's build a custom one:

```
FROM alpine

RUN apk add --no-cache curl

RUN curl -L -o /tmp/crane.tar.gz https://github.com/.../go-containerregistry_Linux_x86_64.tar.gz \
    && tar -xzf /tmp/crane.tar.gz -C /usr/local/bin

COPY --from=martizih/kaniko:v1.27.4 /kaniko /kaniko
ENV SSL_CERT_DIR=/kaniko/ssl/certs
ENTRYPOINT ["/kaniko/executor"]
```





# Now build with it

```
FROM alpine
```

```
RUN apk add --no-cache curl
```

```
RUN curl -L -o /tmp/crane.tar.gz https://github.com/.../go-containerregistry_Linux_x86_64.tar.gz \  
&& tar -xzf /tmp/crane.tar.gz -C /usr/local/bin \  
&& rm /tmp/crane.tar.gz
```

```
RUN crane version
```

official kaniko

  build

 run

kaniko-custom

  build

 run



# Why the build looks fine but the image is broken

Dockerfile

```
FROM alpine
```

```
RUN apk add --no-cache curl
```

```
RUN curl -L ... \  
&& tar -C /usr/local/bin \  
&& rm /tmp/crane.tar.gz
```

```
RUN crane version
```

Official kaniko

■ snapshotted ■ kaniko

```
/busybox/
```

```
/kaniko/
```



Custom kaniko

■ snapshotted ■ kaniko

```
/busybox/
```

```
/kaniko/
```

```
/usr/local/bin/crane  
↑ pre-installed in kaniko base
```





# Why the build looks fine but the image is broken

```
Dockerfile
```

```
FROM alpine
```

```
RUN apk add --no-cache curl
```

```
RUN curl -L ... \  
&& tar -C /usr/local/bin \  
&& rm /tmp/crane.tar.gz
```

```
RUN crane version
```

```
Official kaniko
```

```
■ snapshotted ■ kaniko
```

```
/bin/  
/busybox/  
/etc/  
/kaniko/  
/lib/  
/tmp/  
/usr/
```

```
Custom kaniko
```

```
■ snapshotted ■ kaniko
```

```
/bin/  
/busybox/  
/etc/  
/kaniko/  
/lib/  
/tmp/  
/usr/  
  
/usr/local/bin/crane  
↑ pre-installed in kaniko base
```



# Why the build looks fine but the image is broken

Dockerfile

```
FROM alpine
```

```
▶ RUN apk add --no-cache curl
```

```
RUN curl -L ... \  
&& tar -C /usr/local/bin \  
&& rm /tmp/crane.tar.gz
```

```
RUN crane version
```

Official kaniko

■ snapshotted ■ kaniko

```
/bin/  
/busybox/  
/etc/  
/kaniko/  
/lib/  
/tmp/  
/usr/  
/usr/bin/curl
```



Custom kaniko



■ snapshotted ■ kaniko

```
/bin/  
/busybox/  
/etc/  
/kaniko/  
/lib/  
/tmp/  
/usr/  
/usr/bin/curl  
/usr/local/bin/crane  
↑ pre-installed in kaniko base
```





# Why the build looks fine but the image is broken

| Dockerfile   | Official kaniko  | Custom kaniko   |
|--|--|---|
| <pre>FROM alpine  RUN apk add --no-cache curl  ▶ RUN curl -L ... \   &amp;&amp; tar -C /usr/local/bin \   &amp;&amp; rm /tmp/crane.tar.gz  RUN crane version</pre> | <p>■ snapshotted   ■ kaniko</p> <pre>/bin/ /busybox/ /etc/ /kaniko/ /lib/ /tmp/ /usr/ /usr/bin/curl /usr/local/bin/crane</pre>  | <p>■ snapshotted   ■ kaniko</p> <pre>/bin/ /busybox/ /etc/ /kaniko/ /lib/ /tmp/ /usr/ /usr/bin/curl /usr/local/bin/crane ↑ pre-installed in kaniko base</pre> <p>x no delta</p>  |



# Why the build looks fine but the image is broken

| Dockerfile   | Official kaniko   | Custom kaniko  |
|--|---|--|
| <pre>FROM alpine  RUN apk add --no-cache curl  RUN curl -L ... \   &amp;&amp; tar -C /usr/local/bin \   &amp;&amp; rm /tmp/crane.tar.gz</pre> <p>▶ RUN crane version</p> | <p>Official kaniko</p> <p>■ snapshotted   ■ kaniko</p> <pre>/bin/ /busybox/ /etc/ /kaniko/ /lib/ /tmp/ /usr/ /usr/bin/curl /usr/local/bin/crane</pre> <pre>\$ crane version v0.21.5</pre> <p><i>build succeeds – crane is present in both</i></p> | <p>Custom kaniko</p> <p>■ snapshotted   ■ kaniko</p> <pre>/bin/ /busybox/ /etc/ /kaniko/ /lib/ /tmp/ /usr/ /usr/bin/curl /usr/local/bin/crane ↑ pre-installed in kaniko base</pre> <p>x no delta</p> <pre>\$ crane version v0.21.5</pre> |



# Why the build looks fine but the image is broken

## running

```
$ docker run kaniko-official \  
  crane version  
  
$ docker run kaniko-custom \  
  crane version
```

## Official kaniko

```
■ snapshotted  ■ kaniko  
/bin/  
  
/etc/  
  
/lib/  
/tmp/  
/usr/  
/usr/bin/curl  
/usr/local/bin/crane
```

```
$ crane version  
v0.21.5 ✓
```



## Custom kaniko

```
■ snapshotted  ■ kaniko  
/bin/  
  
/etc/  
  
/lib/  
/tmp/  
/usr/  
/usr/bin/curl
```

```
$ crane version  
crane: not found x
```





# Fix: wipe pre-installed tools before the build

Dockerfile

```
FROM alpine
```

```
RUN apk add --no-cache curl
```

```
RUN curl -L ... \  
&& tar -C /usr/local/bin \  
&& rm /tmp/crane.tar.gz
```

```
RUN crane version
```

Official kaniko

snapshotted  kaniko

```
/busybox/
```

```
/kaniko/
```



Custom kaniko

snapshotted  kaniko

```
/busybox/
```

```
/kaniko/
```

```
/usr/local/bin/crane  
↑ deleted before build
```





# Fix: wipe pre-installed tools before the build

Dockerfile

```
FROM alpine
```

```
RUN apk add --no-cache curl
```

```
RUN curl -L ... \  
&& tar -C /usr/local/bin \  
&& rm /tmp/crane.tar.gz
```

```
RUN crane version
```

Official kaniko

snapshotted  kaniko

```
/busybox/
```

```
/kaniko/
```



Custom kaniko

snapshotted  kaniko

```
/busybox/
```

```
/kaniko/
```





# Fix: wipe pre-installed tools before the build

```
Dockerfile
```

```
FROM alpine
```

```
RUN apk add --no-cache curl
```

```
RUN curl -L ... \  
&& tar -C /usr/local/bin \  
&& rm /tmp/crane.tar.gz
```

```
RUN crane version
```

```
Official kaniko
```

```
■ snapshotted ■ kaniko
```

```
/bin/  
/busybox/  
/etc/  
/kaniko/  
/lib/  
/tmp/  
/usr/
```



```
Custom kaniko
```

```
■ snapshotted ■ kaniko
```

```
/bin/  
/busybox/  
/etc/  
/kaniko/  
/lib/  
/tmp/  
/usr/
```





# Fix: wipe pre-installed tools before the build

Dockerfile

```
FROM alpine
```

```
▶ RUN apk add --no-cache curl
```

```
RUN curl -L ... \  
&& tar -C /usr/local/bin \  
&& rm /tmp/crane.tar.gz
```

```
RUN crane version
```

Official kaniko

```
■ snapshotted ■ kaniko  
/bin/  
/busybox/  
/etc/  
/kaniko/  
/lib/  
/tmp/  
/usr/  
/usr/bin/curl
```



Custom kaniko

```
■ snapshotted ■ kaniko  
/bin/  
/busybox/  
/etc/  
/kaniko/  
/lib/  
/tmp/  
/usr/  
/usr/bin/curl
```





# Fix: wipe pre-installed tools before the build

Dockerfile

```
FROM alpine
```

```
RUN apk add --no-cache curl
```

```
▶ RUN curl -L ... \  
&& tar -C /usr/local/bin \  
&& rm /tmp/crane.tar.gz
```

```
RUN crane version
```

Official kaniko

```
■ snapshotted  ■ kaniko  
/bin/  
/busybox/  
/etc/  
/kaniko/  
/lib/  
/tmp/  
/usr/  
/usr/bin/curl  
/usr/local/bin/crane
```



Custom kaniko

```
■ snapshotted  ■ kaniko  
/bin/  
/busybox/  
/etc/  
/kaniko/  
/lib/  
/tmp/  
/usr/  
/usr/bin/curl  
/usr/local/bin/crane  
↑ properly snapshotted ✓
```





# Fix: wipe pre-installed tools before the build

## Dockerfile

```
FROM alpine

RUN apk add --no-cache curl

RUN curl -L ... \
  && tar -C /usr/local/bin \
  && rm /tmp/crane.tar.gz
```

► RUN crane version

## Official kaniko

■ snapshotted    ■ kaniko

```
/bin/
/busybox/
/etc/
/kaniko/
/lib/
/tmp/
/usr/
/usr/bin/curl
/usr/local/bin/crane
```

```
$ crane version
v0.21.5
```

*build succeeds – crane is present in both*

## Custom kaniko

■ snapshotted    ■ kaniko

```
/bin/
/busybox/
/etc/
/kaniko/
/lib/
/tmp/
/usr/
/usr/bin/curl
/usr/local/bin/crane
↑ properly snapshotted ✓
```

```
$ crane version
v0.21.5
```





# Fix: wipe pre-installed tools before the build

```
running
```

```
$ docker run kaniko-official \
  crane version

$ docker run kaniko-custom \
  crane version
```

```
Official kaniko
```


■ snapshotted   ■ kaniko

```
/bin/

/etc/

/lib/
/tmp/
/usr/
/usr/bin/curl
/usr/local/bin/crane
```

```
$ crane version
v0.21.5 ✓
```



```
Custom kaniko
```


■ snapshotted   ■ kaniko

```
/bin/

/etc/

/lib/
/tmp/
/usr/
/usr/bin/curl
/usr/local/bin/crane
↑ properly snapshotted ✓
```

```
$ crane version
v0.21.5 ✓
```





# Bootstrapping

What if we build our custom kaniko image *using* kaniko itself?

```
FROM alpine

RUN apk add --no-cache curl

RUN curl -L -o /tmp/crane.tar.gz https://github.com/.../go-containerregistry_Linux_x86_64.tar.gz \
  && tar -xzf /tmp/crane.tar.gz -C /usr/local/bin \
  && rm /tmp/crane.tar.gz

COPY --from=martizih/kaniko:v1.27.4 /kaniko /kaniko
ENV SSL_CERT_DIR=/kaniko/ssl/certs
ENTRYPOINT ["/kaniko/executor"]
```





## Now build with it

```
FROM alpine
```

```
RUN apk add --no-cache curl
```

```
RUN curl -L -o /tmp/crane.tar.gz https://github.com/.../go-containerregistry_Linux_x86_64.tar.gz \  
&& tar -xzf /tmp/crane.tar.gz -C /usr/local/bin \  
&& rm /tmp/crane.tar.gz
```

```
RUN crane version
```

  build



# Why the COPY /kaniko disappears silently

```
kaniko_custom Dockerfile

FROM alpine

RUN apk add --no-cache curl

RUN curl -L ... \
  && tar -C /usr/local/bin \
  && rm /tmp/crane.tar.gz

COPY --from=kaniko \
  /kaniko /kaniko

ENTRYPOINT ["/kaniko/executor"]
```

## docker build

■ snapshotted ■ kaniko



## kaniko build



■ snapshotted ■ kaniko

```
/busybox/      - pre-existing
/kaniko/       - pre-existing
```





# Why the COPY /kaniko disappears silently

| kaniko_custom Dockerfile   | docker build   | kaniko build   |
|--|--|--|
| <pre>kaniko_custom Dockerfile  ▶ FROM alpine  RUN apk add --no-cache curl  RUN curl -L ... \   &amp;&amp; tar -C /usr/local/bin \   &amp;&amp; rm /tmp/crane.tar.gz  COPY --from=kaniko \   /kaniko /kaniko  ENTRYPOINT ["/kaniko/executor"]</pre> | <p>■ snapshotted   ■ kaniko</p> <pre>/bin/  /etc/  /lib/ /tmp/ /usr/</pre>  | <p>■ snapshotted   ■ kaniko</p> <pre>/bin/ /busybox/      - pre-existing /etc/ /kaniko/       - pre-existing /lib/ /tmp/ /usr/</pre>  |



# Why the COPY /kaniko disappears silently

```
kaniko_custom Dockerfile

FROM alpine

▶ RUN apk add --no-cache curl

RUN curl -L ... \
  && tar -C /usr/local/bin \
  && rm /tmp/crane.tar.gz

COPY --from=kaniko \
  /kaniko /kaniko

ENTRYPOINT ["/kaniko/executor"]
```


**docker build**

■ snapshotted    ■ kaniko

```
/bin/

/etc/


/lib/
/tmp/
/usr/
/usr/bin/curl
```



**kaniko build**

■ snapshotted    ■ kaniko

```
/bin/
/busybox/           - pre-existing
/etc/
/kaniko/            - pre-existing
/lib/
/tmp/
/usr/
/usr/bin/curl
```





# Why the COPY /kaniko disappears silently

kaniko\_custom Dockerfile

```
FROM alpine
```

```
RUN apk add --no-cache curl
```

```
▶ RUN curl -L ... \  
&& tar -C /usr/local/bin \  
&& rm /tmp/crane.tar.gz
```

```
COPY --from=kaniko \  
/kaniko /kaniko
```

```
ENTRYPOINT ["/kaniko/executor"]
```

docker build

■ snapshotted ■ kaniko

```
/bin/
```

```
/etc/
```

```
/lib/
```

```
/tmp/
```

```
/usr/
```

```
/usr/bin/curl
```

```
/usr/local/bin/crane
```



kaniko build

■ snapshotted ■ kaniko

```
/bin/
```

```
/busybox/ - pre-existing
```

```
/etc/
```

```
/kaniko/ - pre-existing
```

```
/lib/
```

```
/tmp/
```

```
/usr/
```

```
/usr/bin/curl
```

```
/usr/local/bin/crane
```





# Why the COPY /kaniko disappears silently

kaniko\_custom Dockerfile

```
FROM alpine

RUN apk add --no-cache curl

RUN curl -L ... \
  && tar -C /usr/local/bin \
  && rm /tmp/crane.tar.gz

▶ COPY --from=kaniko \
  /kaniko /kaniko

ENTRYPOINT ["/kaniko/executor"]
```

docker build

■ snapshotted ■ kaniko

```
/bin/

/etc/

/lib/
/tmp/
/usr/
/usr/bin/curl
/usr/local/bin/crane
/kaniko/
├ executor
└ ssl/certs ...
captured ✓
```



kaniko build

■ snapshotted ■ kaniko

```
/bin/
/busybox/      - pre-existing
/etc/
/kaniko/       - pre-existing
/lib/
/tmp/
/usr/
/usr/bin/curl
/usr/local/bin/crane
```

**snapshotter: /kaniko excluded**

writes here never reach the output image





# Why the COPY /kaniko disappears silently

## run the image

```
$ docker run kaniko-docker \
  --no-push ...

$ docker run kaniko-kaniko \
  --no-push ...
```

## docker build

```
■ snapshotted ■ kaniko

/bin/

/etc/

/lib/
/tmp/
/usr/
/usr/bin/curl
/usr/local/bin/crane
/kaniko/
├─ executor
├─ ssl/certs ...
captured ✓
```

```
$ /kaniko/executor ...
```

**works ✓**



## kaniko build

```
■ snapshotted ■ kaniko

/bin/

/etc/

/lib/
/tmp/
/usr/
/usr/bin/curl
/usr/local/bin/crane
```

```
$ /kaniko/executor ...
```

```
exec: no such file or directory x
/kaniko/ was never committed to a layer
```





# Fix: rename /kaniko before the build

```
kaniko_custom Dockerfile

FROM alpine

RUN apk add --no-cache curl

RUN curl -L ... \
  && tar -C /usr/local/bin \
  && rm /tmp/crane.tar.gz

COPY --from=kaniko \
  /kaniko /kaniko

ENTRYPOINT ["/kaniko/executor"]
```

## docker build

■ snapshotted ■ kaniko



## kaniko build



■ snapshotted ■ kaniko

```
/busybox/ ← pre-existing
/kaniko2/ ← renamed ✓
```





# Fix: rename /kaniko before the build

| kaniko_custom Dockerfile   | docker build   | kaniko build  |
|--|--|---|
| <pre>kaniko_custom Dockerfile  ▶ FROM alpine  RUN apk add --no-cache curl  RUN curl -L ... \   &amp;&amp; tar -C /usr/local/bin \   &amp;&amp; rm /tmp/crane.tar.gz  COPY --from=kaniko \   /kaniko /kaniko  ENTRYPOINT ["/kaniko/executor"]</pre> | <p>■ snapshotted   ■ kaniko</p> <pre>/bin/  /etc/  /lib/ /tmp/ /usr/</pre>  | <p>■ snapshotted   ■ kaniko</p> <pre>/bin/ /busybox/      ← pre-existing /etc/ /kaniko2/      ← renamed /lib/ /tmp/ /usr/</pre>  |



# Fix: rename /kaniko before the build

kaniko\_custom Dockerfile

```
FROM alpine
```

```
▶ RUN apk add --no-cache curl
```

```
RUN curl -L ... \  
&& tar -C /usr/local/bin \  
&& rm /tmp/crane.tar.gz
```

```
COPY --from=kaniko \  
/kaniko /kaniko
```

```
ENTRYPOINT ["/kaniko/executor"]
```

docker build

■ snapshotted ■ kaniko

```
/bin/
```

```
/etc/
```

```
/lib/
```

```
/tmp/
```

```
/usr/
```

```
/usr/bin/curl
```



kaniko build

■ snapshotted ■ kaniko

```
/bin/
```

```
/busybox/ - pre-existing
```

```
/etc/
```

```
/kaniko2/ - renamed
```

```
/lib/
```

```
/tmp/
```

```
/usr/
```

```
/usr/bin/curl
```





# Fix: rename /kaniko before the build

kaniko\_custom Dockerfile

```
FROM alpine
```

```
RUN apk add --no-cache curl
```

```
▶ RUN curl -L ... \  
&& tar -C /usr/local/bin \  
&& rm /tmp/crane.tar.gz
```

```
COPY --from=kaniko \  
/kaniko /kaniko
```

```
ENTRYPOINT ["/kaniko/executor"]
```

docker build

■ snapshotted ■ kaniko

```
/bin/
```

```
/etc/
```

```
/lib/
```

```
/tmp/
```

```
/usr/
```

```
/usr/bin/curl
```

```
/usr/local/bin/crane
```



kaniko build

■ snapshotted ■ kaniko

```
/bin/
```

```
/busybox/ - pre-existing
```

```
/etc/
```

```
/kaniko2/ - renamed
```

```
/lib/
```

```
/tmp/
```

```
/usr/
```

```
/usr/bin/curl
```

```
/usr/local/bin/crane
```





# Fix: rename /kaniko before the build

kaniko\_custom Dockerfile

```
FROM alpine
```

```
RUN apk add --no-cache curl
```

```
RUN curl -L ... \  
&& tar -C /usr/local/bin \  
&& rm /tmp/crane.tar.gz
```

```
► COPY --from=kaniko \  
/kaniko /kaniko
```

```
ENTRYPOINT ["/kaniko/executor"]
```

docker build

■ snapshotted ■ kaniko

```
/bin/  
  
/etc/  
  
/lib/  
/tmp/  
/usr/  
/usr/bin/curl  
/usr/local/bin/crane  
/kaniko/  
└─ executor  
└─ ssl/certs ...  
captured ✓
```



kaniko build

■ snapshotted ■ kaniko

```
/bin/  
/busybox/ ← pre-existing  
/etc/  
/kaniko2/ ← renamed  
/lib/  
/tmp/  
/usr/  
/usr/bin/curl  
/usr/local/bin/crane  
/kaniko/  
└─ executor  
└─ ssl/certs ...  
captured ✓
```





# Fix: rename /kaniko before the build

## run the image

```
$ docker run kaniko-docker \  
--no-push ...  
  
$ docker run kaniko-kaniko \  
--no-push ...
```

## docker build

```
■ snapshotted ■ kaniko  
  
/bin/  
  
/etc/  
  
/lib/  
/tmp/  
/usr/  
/usr/bin/curl  
/usr/local/bin/crane  
/kaniko/  
  └─ executor  
  └─ ssl/certs ...  
captured ✓
```

```
$ /kaniko/executor ...
```

**works ✓**



## kaniko build

```
■ snapshotted ■ kaniko  
  
/bin/  
  
/etc/  
  
/lib/  
/tmp/  
/usr/  
/usr/bin/curl  
/usr/local/bin/crane  
/kaniko/  
  └─ executor  
  └─ ssl/certs ...  
captured ✓
```

```
$ /kaniko/executor ...
```

**works ✓**





# No Sandboxing

```
FROM busybox  
RUN ls -la /kaniko
```

Sandboxing requires privileges, so kaniko doesn't



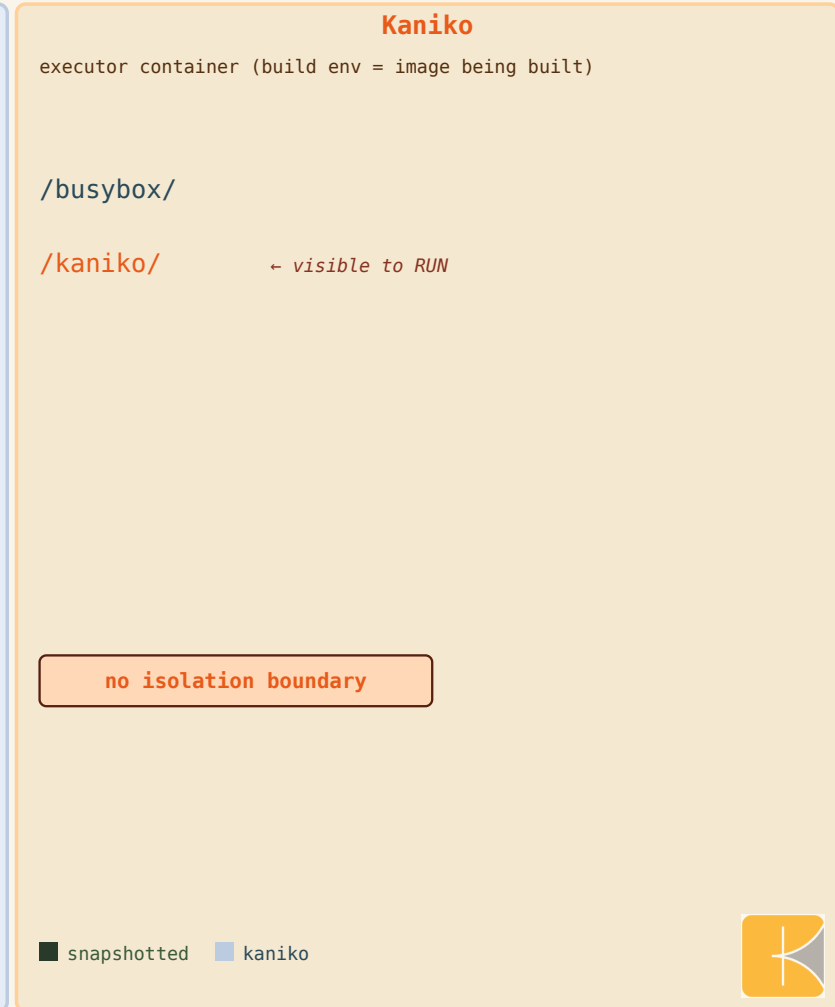


# Build isolation: what RUN can see

```
Dockerfile

FROM busybox

RUN ls -la /kaniko
```





# Build isolation: what RUN can see

Dockerfile

```
FROM busybox
```

```
RUN ls -la /kaniko
```

### DinD


build daemon

image being built (isolated)

- /bin/
- /etc/
- /lib/
- /tmp/
- /usr/

build tooling – not mounted into image container

encapsulated ✓


■ snapshotted 

### Kaniko

executor container (build env = image being built)

- /bin/
- /busybox/
- /etc/
- /kaniko/ ← visible to RUN
- /lib/
- /tmp/
- /usr/

no isolation boundary

■ snapshotted ■ kaniko 



# Build isolation: what RUN can see

```
Dockerfile

FROM busybox

▶ RUN ls -la /kaniko
```

### DinD

build daemon

image being built (isolated)

```
/bin/
/etc/
/lib/
/tmp/
/usr/
```

build tooling – not mounted into image container

encapsulated ✓

```
$ ls -la /kaniko
ls: /kaniko: No such file or directory
build tooling correctly isolated ✓
```

■ snapshotted



### Kaniko

executor container (build env = image being built)

```
/bin/
/busybox/
/etc/
/kaniko/ ← visible to RUN
/lib/
/tmp/
/usr/
```

no isolation boundary

```
$ ls -la /kaniko
-rwxr-xr-x executor
drwx----- .docker/ ← registry creds!
drwxr-xr-x ssl/
...
build tooling + credentials exposed to payload x
```





# Security trade-offs

**Docker DinD**  
privileged: true

```
build daemon
image being built (isolated)
/bin/
/etc/
/lib/
/tmp/
/usr/
```

build daemon area (not accessible to payload):

```
~/ .docker/config.json
registry credentials - hidden behind inner wall ✓
```

↑ inner wall keeps credentials out of payload reach

⚠ **weak outer wall**  
privileged: true grants near-host kernel access to the build container

**Kaniko**  
unprivileged

executor container (build env = image being built)

```
/bin/
/busybox/
/etc/
/kaniko/
  .docker/config.json
/lib/
/tmp/
/usr/
```

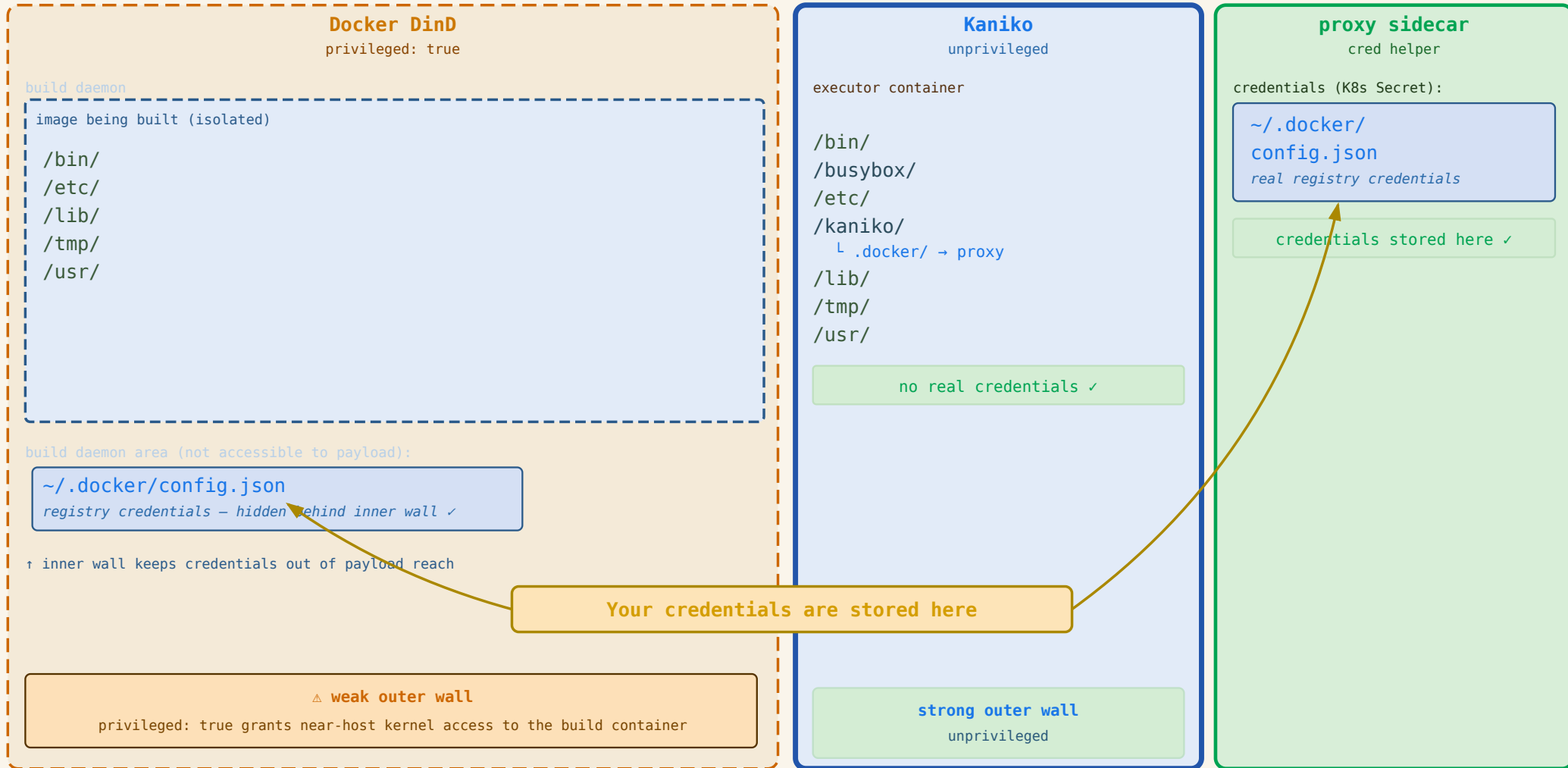
**strong outer wall**  
no --privileged needed - container is properly isolated from host

Your credentials are stored here

*DinD protects your passwords — Kaniko protects your servers*



# Fix: credential proxy sidecar



*Your token lives in the proxy – never touches kaniko's filesystem*

# All Dragons slayed



# All Dragons slayed



*Kaniko's philosophy: don't invent your own security model, just use K8s correctly.*

# All Dragons slayed



*Kaniko's philosophy: don't invent your own security model, just use K8s correctly.*

...but we're still root `uid=0`

PART 4

# Future of rootless builds



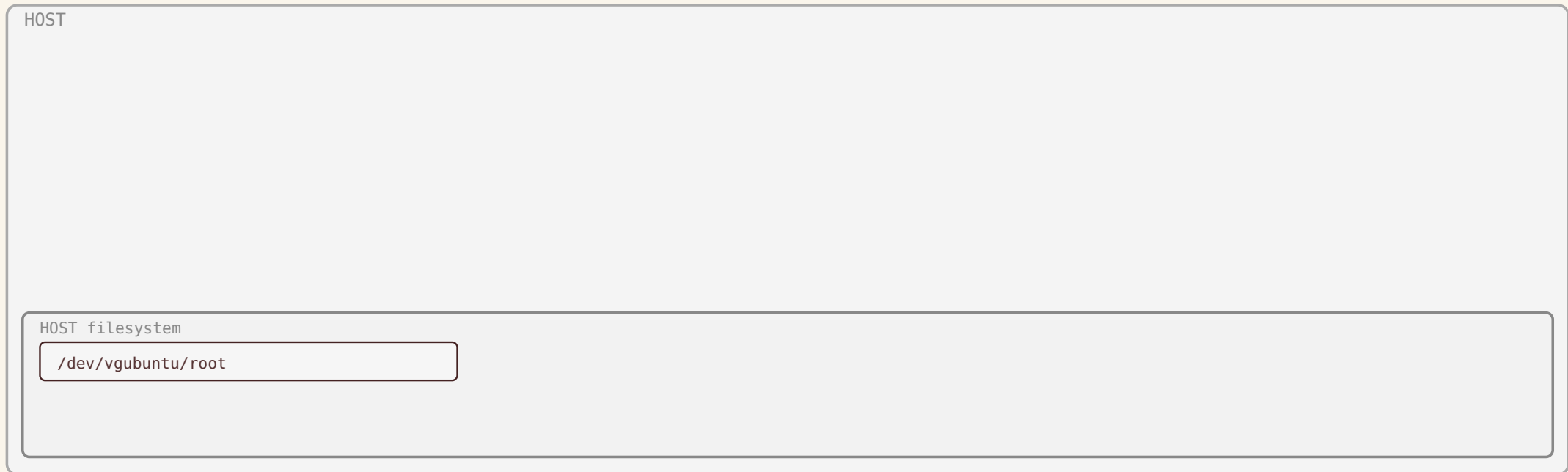
# Running `docker build` in CI — k8s 1.36

With user namespaces, container root maps to an unprivileged UID on the host:

```
apiVersion: v1
kind: Pod
spec:
+ hostUsers: false
  containers:
  - name: docker
    image: docker:dind
    securityContext:
      privileged: true
```

Container UID 0 → Host UID ~65536: `privileged` is still required for `dind`, but escaping the container no longer gives host root.

# Privileged container — mapped root



 run

# Privileged container — mapped root

HOST

```
unshare --mount --pid --uts --ipc --net --fork - pivot_root /tmp/ctr - exec sh
```

**uid=0(root)**  
host uid: 65536

CAP\_SYS\_ADMIN    CAP\_NET\_ADMIN    CAP\_SYS\_RAWIO    + 37 more ...

capabilities valid within user namespace only – not on the real host

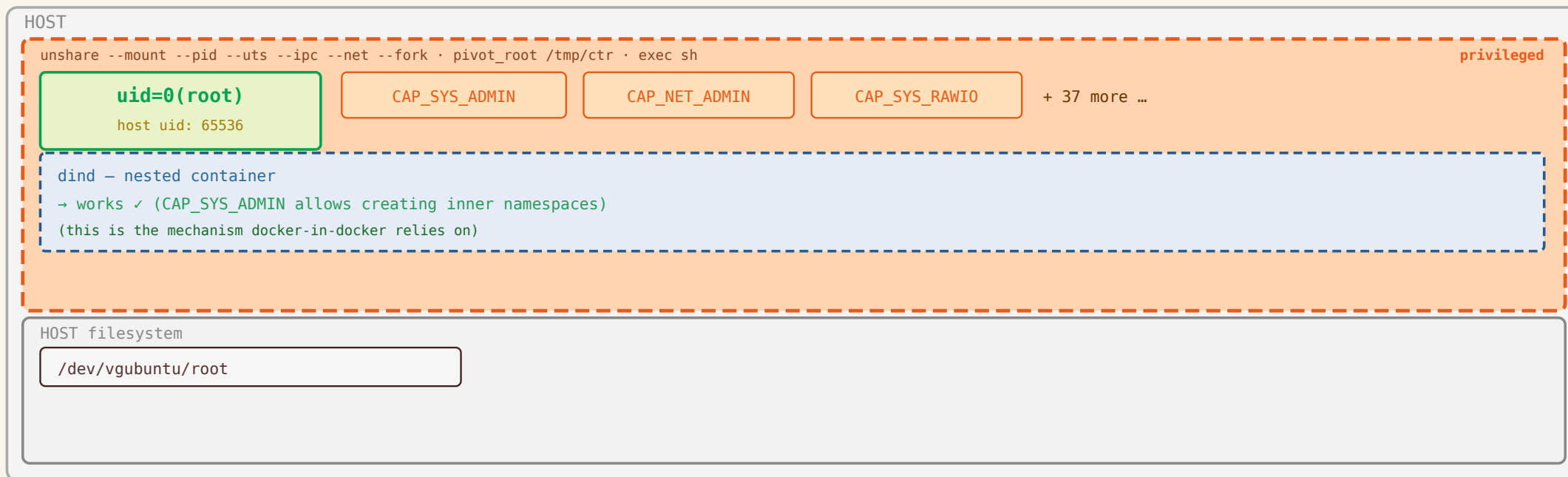
privileged

HOST filesystem

/dev/vgubuntu/root

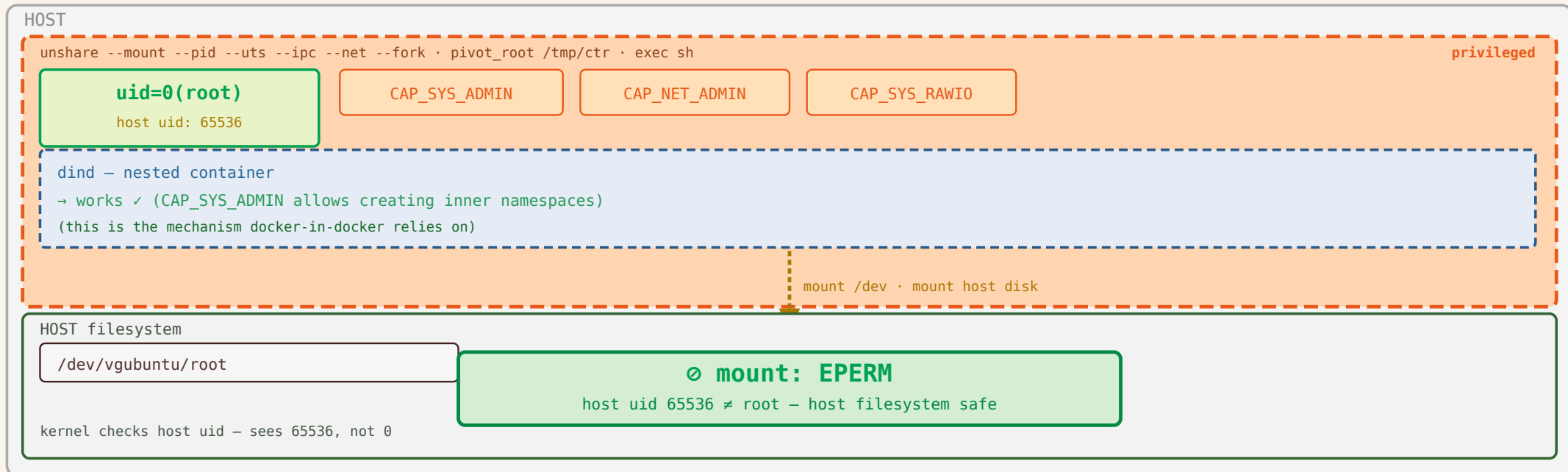
 dind

# Privileged container — mapped root



breakout

# Privileged container — mapped root



# The landscape — before K8s 1.36

| Tool       | Root user | CAP_SYS_ADMIN | Kernel features        | Dockerfile | Complexity       |
|------------|-----------|---------------|------------------------|------------|------------------|
| ko / Jib   | No        | No            | –                      | No         | build plugin     |
| Nix        | No        | No            | –                      | No         | Nix toolchain    |
| Podman†    | No        | No            | user ns                | Yes        | job pod + sysctl |
| buildkitd† | No        | optional      | user ns, FUSE          | Yes        | daemon + sysctl  |
| Kaniko     | root      | No            | –                      | Yes        | job pod          |
| Podman     | root      | No            | –                      | Yes        | job pod          |
| buildkitd  | root      | optional      | –                      | Yes        | daemon pod       |
| Dind       | root      | Yes           | cgroups, ns, overlayfs | Yes        | sidecar          |

† rootless variant

## The landscape — after K8s 1.36 `hostUsers: false`

| Tool      | Root user | CAP_SYS_ADMIN | Kernel features        | Dockerfile | Complexity    |
|-----------|-----------|---------------|------------------------|------------|---------------|
| ko / Jib  | No        | No            | —                      | No         | build plugin  |
| Nix       | No        | No            | —                      | No         | Nix toolchain |
| Kaniko    | root      | No            | —                      | Yes        | job pod       |
| Podman    | root      | No            | —                      | Yes        | job pod       |
| buildkitd | root      | optional      | —                      | Yes        | daemon pod    |
| DinD      | root      | Yes           | cgroups, ns, overlayfs | Yes        | sidecar       |

† rootless variant · root in pod → uid 65536 on host · CAP\_SYS\_ADMIN scoped to user namespace

PART 4

# Fun Stuff

# Non-Docker Applications

Kaniko lets an agent **boot into any Docker image**, mid-task, no privileges needed.

[coder.zrh.int.scandit.io/tasks](https://coder.zrh.int.scandit.io/tasks)

 prompt



# Questions?

[osscontainertools/kaniko](https://github.com/osscontainertools/kaniko)

