

n|u null **Ahmedabad**

Securing Containers from Day One

Container Security



Kumar **Ashwin**

0xcardinal.com

About Me

Kumar **Ashwin**

- Security Consultant @ Payatu
- Manages null Study Groups for six different Security Domains and also contributes to other communities
- 0xcatholic.com
- 0xCardinal on social platforms

Agenda

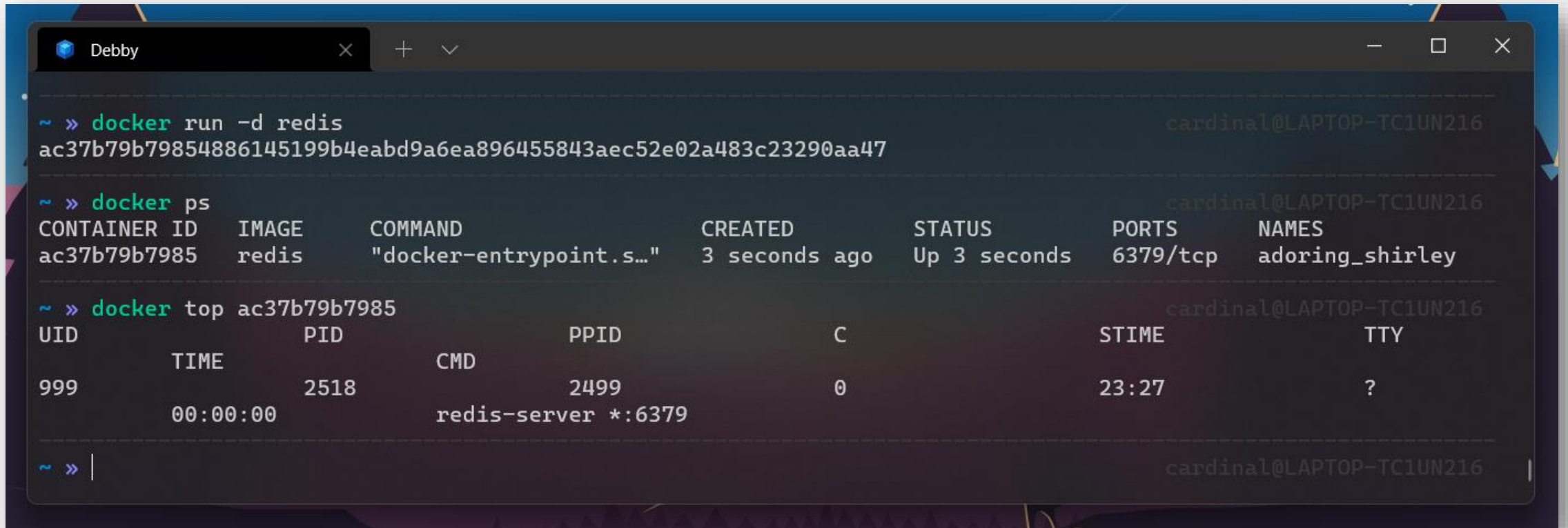
- What are containers?
- Why we need containers?
- VM v/s Containers
- Cgroups & Namespaces
- Docker Primer
- Build Optimization
- Security
- Resources
- QnA



<https://giphy.com/gifs/rockstargames-usz0fqhUiVxSs6lUKB>

What are containers?

Containers are nothing but **just another Linux process** which is isolated from other processes running on the same host.

A terminal window titled 'Debby' with standard window controls. It shows three Docker commands and their outputs. The first command runs Redis in the background. The second command lists running containers. The third command shows details for a specific container.

```
~ » docker run -d redis
ac37b79b79854886145199b4eabd9a6ea896455843aec52e02a483c23290aa47

~ » docker ps
CONTAINER ID   IMAGE    COMMAND                  CREATED        STATUS        PORTS          NAMES
ac37b79b7985   redis    "docker-entrypoint.s..." 3 seconds ago  Up 3 seconds  6379/tcp       adoring_shirley

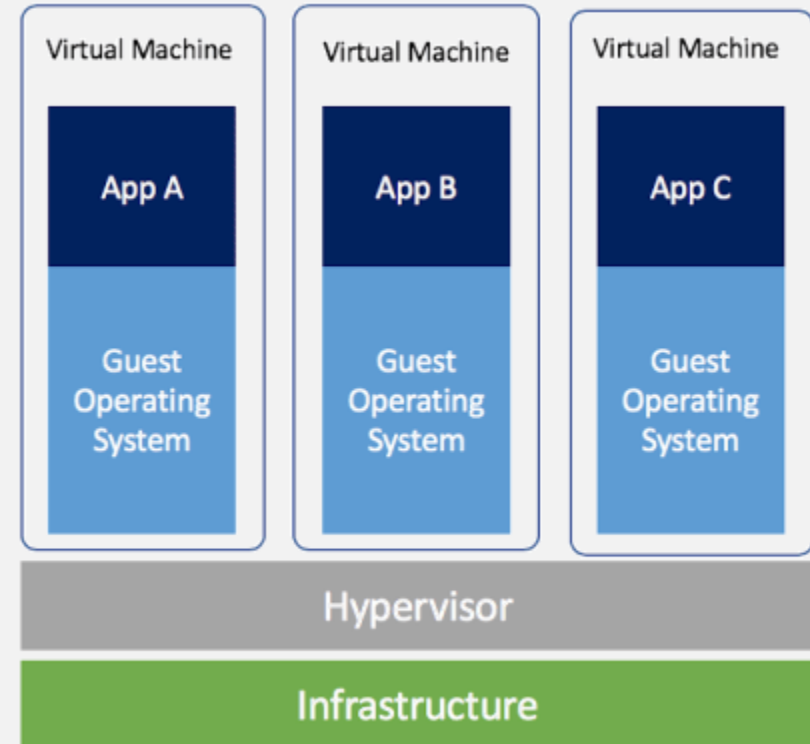
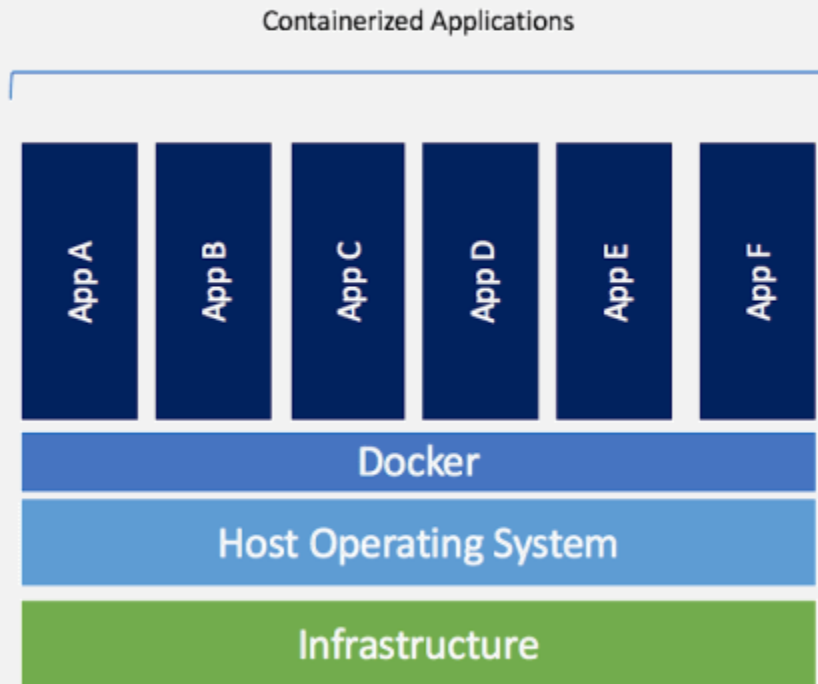
~ » docker top ac37b79b7985
UID            PID        PPID          C             STIME        TTY
999            2518       2499          0             23:27        ?
00:00:00      redis-server *:6379
```

**Why do we
need
containers?**



IT WORKS
ON MY MACHINE

Virtual Machines v/s Containers



cgroups & namespaces

namespaces

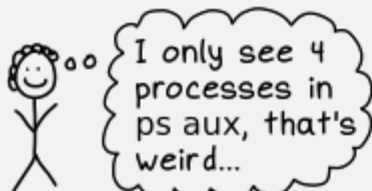
It defines what a container can **see**, uses syscalls to do so.

JULIA EVANS
@b0rk

namespaces

14

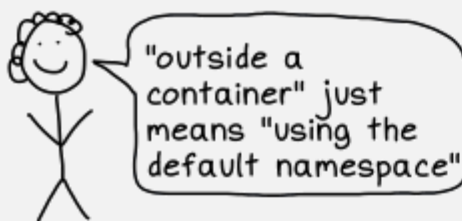
inside a container,
things look different



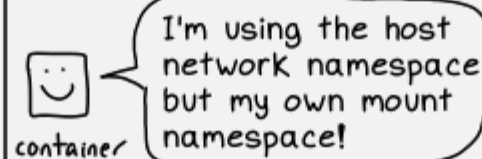
why things look different:
namespaces



there's a default
("host" namespace)



processes can have
any combination
of namespaces



every process has
7 namespaces

```
$ lsns -p 273
      NS TYPE
4026531835 cgroup
4026531836 pid
4026531837 user
4026531838 uts
4026531839 ipc
4026531840 mnt
4026532009 net
```

you can also see a
process's namespace with:
\$ ls -l /proc/273/ns

Demo : namespaces

Will share how the name spaces work using unshare and creating namespaces for user and network, and demonstrating the difference.

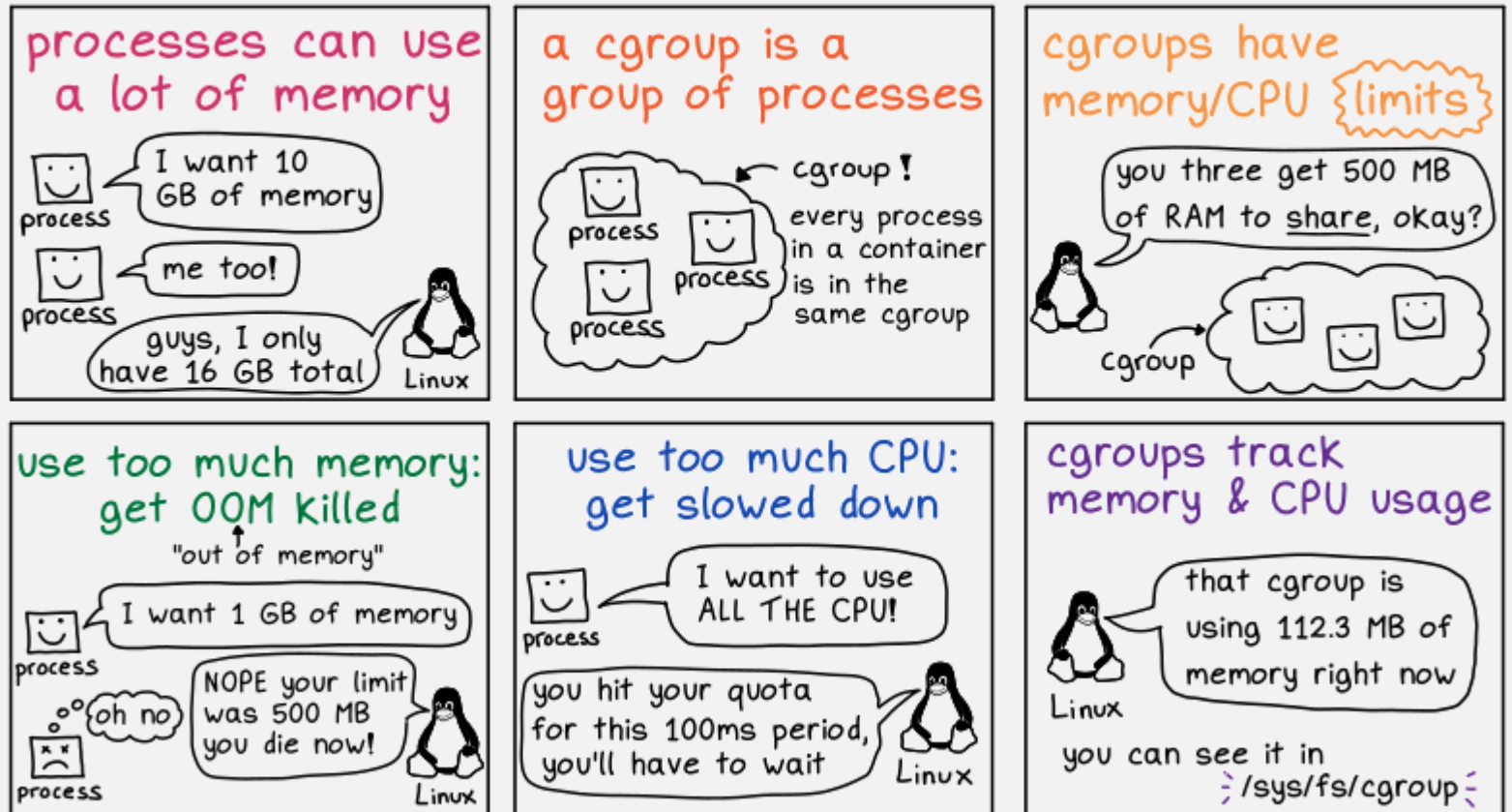
cgroups

It defines what a container can **use or access**, uses syscalls to do so.

JULIA EVANS
@b0rk

cgroups

13





Myth busting

**Docker itself is not a
Container.**

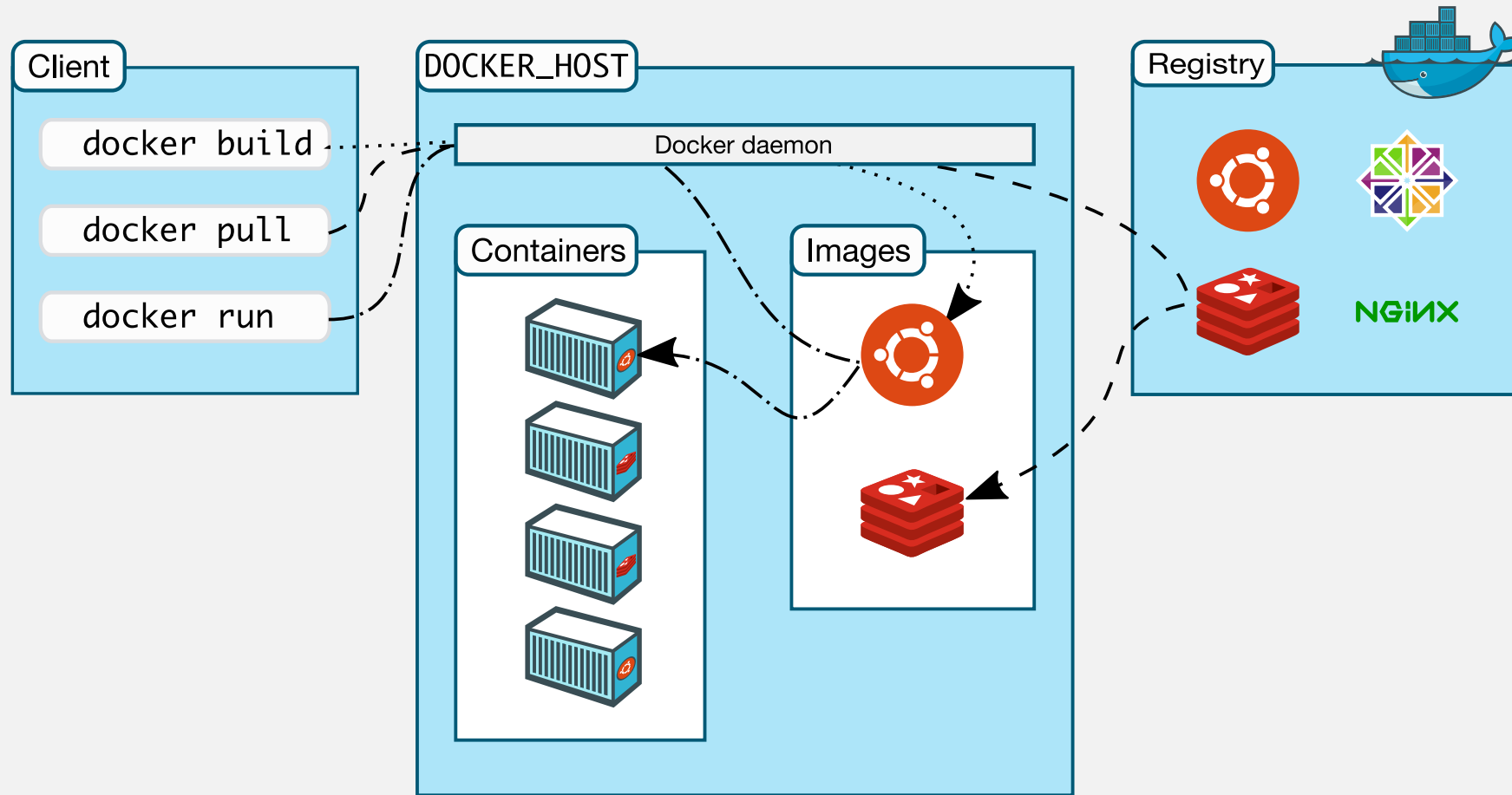
What is Docker?

- Docker is a **container engine**, which

is a piece of software that accepts user requests, including command line options, pulls images, and from the end user's perspective runs the container.

- Other than docker there are other container engines as well, like - **RKT, CRI-O, and LXD**

Docker Architecture



Docker Basics

- There are different **images**, which are stored in the **registry**, from where it **pull** the image, to create different containers.
- Generally, a **Dockerfile** (contains commands/instructions) is used to build a container.
- If you want to run a multi-container Docker application – **Docker Compose** is your go to tool.
- Common docker/docker compose commands, that are generally used –
 - `docker pull <image-name>` – used to pull images from registry
 - `docker run [args] <image-name>` – used to run a container from the image defined
 - `docker build [args]` – used to build a container out of a **Dockerfile**
 - `docker-compose up` – used to build a multi-container setup from `docker-compose.yml`
 - `docker ps` – used to list down the active containers
- For every RUN, COPY, ADD instruction in a **Dockerfile**, a layer is created.

Dockerfile

 Dockerfile

```
1  # Pulling the nginx image from the dockerhub
2  FROM nginx:latest
3
4  # Copying the required files inside the container
5  ADD index.html /usr/share/nginx/html
6  ADD linux.png /usr/share/nginx/html
7
8  # Exposing the Port to interact with the server outside the container
9  EXPOSE 80 443
10
11 # Running some command
12 CMD ["nginx", "-g", "daemon off;"]
13
```

*** insecure**

Build Optimization

- Minimize number of layers. Improves performance.
- Multi-Stage Builds
- Do not install unnecessary packages
- Decouple the application
- Slim down the image – using `docker-slim`
 - It promises to slim down the image by 30x - <https://github.com/docker-slim/docker-slim>

Why Dockerfile Security?

- Here we will be talking about securing the images pre-build and what are the practices that we can follow.
- A great start point to look for security issues are Dockerfile.
- Dockerfiles?
 - These are the blueprints of the system/container that is to be created.
 - Infrastructure as a Code (IaaS)
 - One of the main components for the entire supply chain security.

Security Best-Practices

- Prefer minimal base images
- Use .dockerignore file to exclude files from build
- Create Golden Images
 - Golden Images are hardened base images than can be used further development
- Do not run containers as root
 - Adding this in the Dockerfile will help in chainging the user
- Do not commit secrets in Dockerfile or Containers
 - Can use [BuildKit](#) to pass secrets to use in containers securely
- Use COPY instead of ADD, wherever possible

```
FROM alpine:latest
RUN useradd -u 1234 non-root-user
USER non-root-user
```

More Security Stuff

- Use linters like [hadolint](#), that will help to build best practice Docker Images.



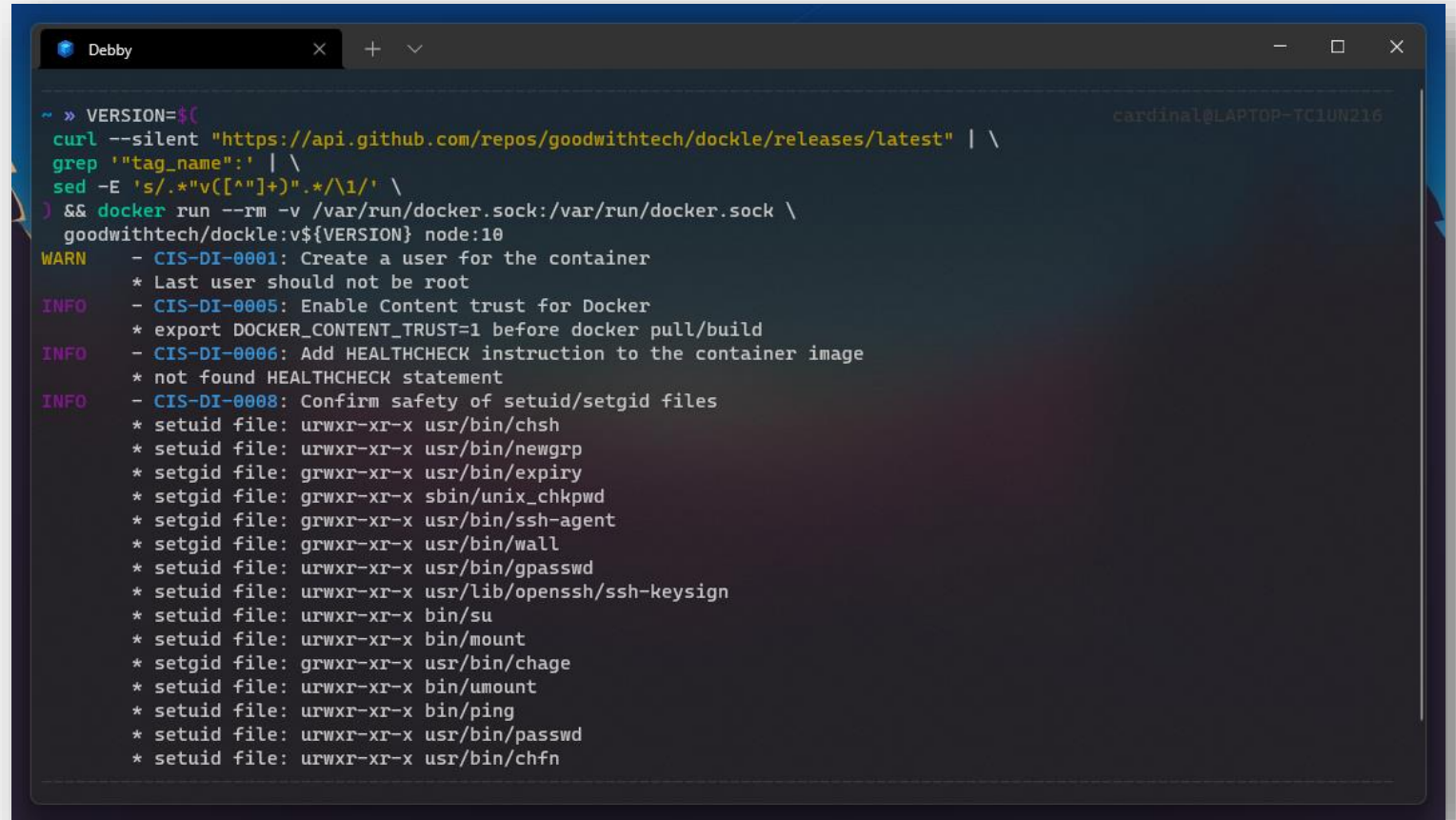
Dockerfile Linter

A smarter Dockerfile linter that helps you build [best practice Docker images](#). The linter is parsing the Dockerfile into an AST and performs rules on top of the AST. It additionally is using the famous [Shellcheck](#) to lint the Bash code inside RUN instructions. Please [help me improve the linter](#) with your suggestions.

```
Always tag the version of an image explicitly
1 FROM debian
   node_verion is referenced but not assigned (did you mean 'node_version'?).
   Delete the apt-get lists after installing something
   Avoid additional packages by specifying '--no-install-recommends'
2 RUN export node_version="0.10" \
3 && apt-get update && apt-get -y install nodejs="$node_version"
4 COPY package.json usr/src/app
   Use WORKDIR to switch to a directory
   Pin versions in npm. Instead of `npm install <package>` use `npm install <package>@<version>`
5 RUN cd /usr/src/app \
6 && npm install node-static
   Valid UNIX ports range from 0 to 65535
7 EXPOSE 80000
```

More Security Stuff

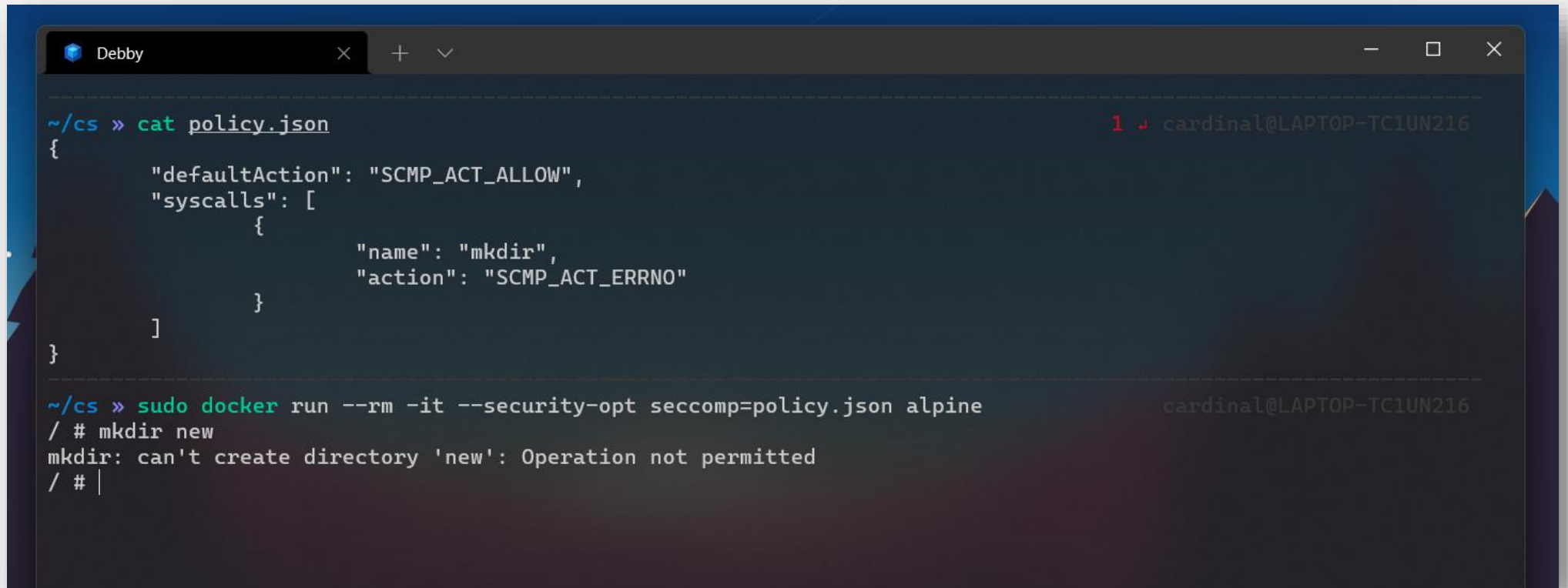
- Use dockle to scan images against CIS benchmarks.
- [CIS Benchmark security Comparision](#)



```
~ » VERSION=$(
curl --silent "https://api.github.com/repos/goodwithtech/dockle/releases/latest" | \
grep '"tag_name":' | \
sed -E 's/.*"v([^"]+)"*/\1/' \
) && docker run --rm -v /var/run/docker.sock:/var/run/docker.sock \
goodwithtech/dockle:v${VERSION} node:10
WARN - CIS-DI-0001: Create a user for the container
* Last user should not be root
INFO - CIS-DI-0005: Enable Content trust for Docker
* export DOCKER_CONTENT_TRUST=1 before docker pull/build
INFO - CIS-DI-0006: Add HEALTHCHECK instruction to the container image
* not found HEALTHCHECK statement
INFO - CIS-DI-0008: Confirm safety of setuid/setgid files
* setuid file: urwxr-xr-x usr/bin/chsh
* setuid file: urwxr-xr-x usr/bin/newgrp
* setgid file: grwxr-xr-x usr/bin/expiry
* setgid file: grwxr-xr-x sbin/unix_chkpwd
* setgid file: grwxr-xr-x usr/bin/ssh-agent
* setgid file: grwxr-xr-x usr/bin/wall
* setuid file: urwxr-xr-x usr/bin/gpasswd
* setuid file: urwxr-xr-x usr/lib/openssh/ssh-keysign
* setuid file: urwxr-xr-x bin/su
* setuid file: urwxr-xr-x bin/mount
* setgid file: grwxr-xr-x usr/bin/chage
* setuid file: urwxr-xr-x bin/umount
* setuid file: urwxr-xr-x bin/ping
* setuid file: urwxr-xr-x usr/bin/passwd
* setuid file: urwxr-xr-x usr/bin/chfn
```

Seccomp

- It is a security feature in the Linux kernel. It can be used to restrict any system calls in the docker container.
- E.g.,



```
~/cs » cat policy.json
{
  "defaultAction": "SCMP_ACT_ALLOW",
  "syscalls": [
    {
      "name": "mkdir",
      "action": "SCMP_ACT_ERRNO"
    }
  ]
}

~/cs » sudo docker run --rm -it --security-opt seccomp=policy.json alpine
/ # mkdir new
mkdir: can't create directory 'new': Operation not permitted
/ # |
```

The screenshot shows a terminal window titled 'Debby' with a dark theme. The first command is `cat policy.json`, which displays a JSON configuration for the seccomp filter. The configuration sets the default action to `SCMP_ACT_ALLOW` and defines a single system call rule for `mkdir` with the action `SCMP_ACT_ERRNO`. The second command is `sudo docker run --rm -it --security-opt seccomp=policy.json alpine`, which starts an Alpine Linux container. Inside the container, the user runs `mkdir new`, which fails with the error `mkdir: can't create directory 'new': Operation not permitted`. The prompt `/ #` indicates the user is root inside the container.

Some Resources

- [Containers from Scratch](#) [YouTube]
- [Dockerfile Best Practices](#)
- [Multi-stage builds](#)
- [dockle checkpoint comparison](#)
- [Tips for optimizing builds](#)
- [Dockerfile Tutorial](#)
- [Practical guide to write Dockerfile](#)

QnA

Securing Containers from Day One

Thank You!

Kumar **Ashwin**

twitter.com/0xcardinal