



*Open Source 101*

# Recipes & Runtimes

## Making Sense of Containers in 2026

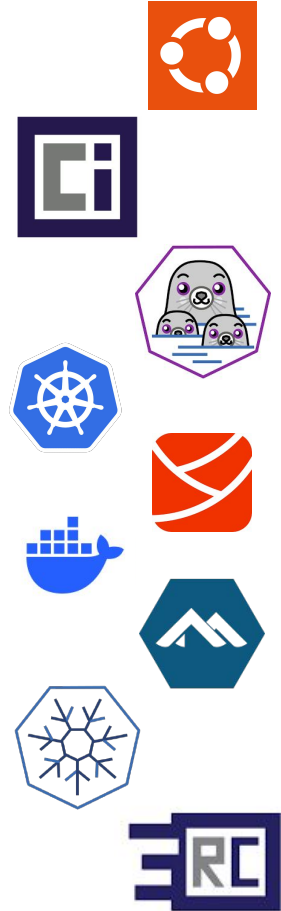
*Soundarya Rangarajan*  
*Engineering @ Canonical*  
*16<sup>th</sup> June 2026*



# How did we get here?

2026

Provenance  
Minimal attack surface  
Supply-chain poisoning  
....





# Let's rewind...

2013



2015

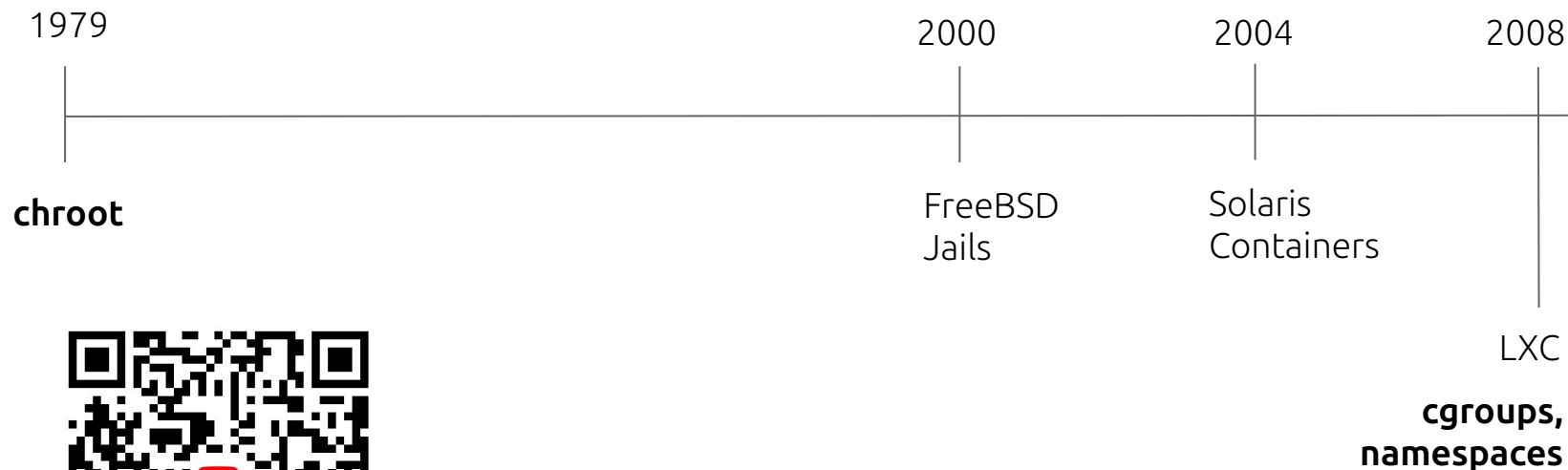


2019





# I mean, *really* rewind





Let's containerize our micro service!



~~Let's containerize our micro service!~~

Let's level-up our micro bakery!



Disclaimer:

I know nothing about baking.

Take my analogies with a pinch of salt (or should I say sugar?)



Baking at home and  
shipping to customers



Baking in cloud  
kitchens around the  
city and shipping to  
customers nearby



Recreate the same taste. Every single time.



Ensure the whole kitchen doesn't go down if something goes wrong with our baking.



Recreate the same taste. Every single time.  
*Reproducible environment.*

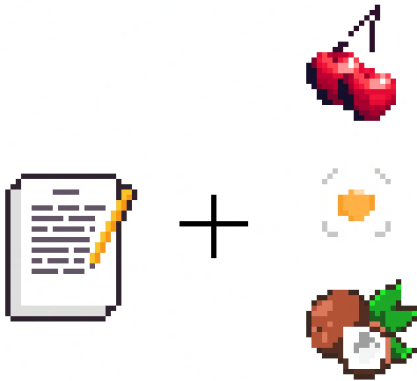


Ensure the whole kitchen doesn't go down if something goes wrong with our baking.  
*Isolation.*



# #1 Container Image

Bake kit =  
Recipe + Ingredients



A container image is a blueprint to create a running container.

It contains everything needed to run an app:

Code

Runtime

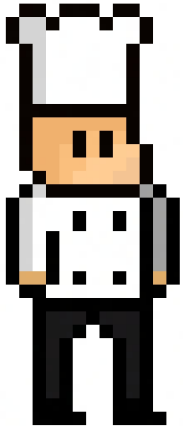
Libraries

Env, settings



## #2 Container

A chef,  
In an isolated part of a cloud  
kitchen



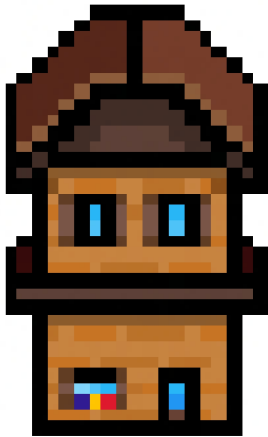
A container is the running  
process actually serving your  
app.

“Isolated, lightweight  
process”



# #3 Container runtime

An isolated part of a cloud kitchen, just for your baking



The specialized software component that takes a container image, unpacks it, and turns it into a running isolated process (a container) on a host OS



# Long time, no code

Time for a practical example



```
`git checkout v0-basic-webapp`  
To inspect the code and tinker
```



# #4 Image hardening

Fewer ingredients = tighter control



## **Attack surface:**

Every component that could potentially contain a CVE, or be abused if compromised.



# Image hardening



Goal (i) : Reduce **exposure**. What are the chances things could go wrong?  
Reduce vulnerability exposure.



Goal (ii) : Reduce **blast radius**. If things do go wrong, just how bad can it get?  
Reduce post-compromise capability.



# Image hardening: *How?*

1. Remove unnecessary packages

Multi-stage builds and ``slim`` base images.

Don't add build deps in final image. Only runtime deps.



# Let's optimize our naive Dockerfile...



```
`git checkout v2-multistage`  
To inspect the code and tinker
```





# Image hardening: *How?*

## 2. Minimal base image

- a. Alpine
- b. Distroless
- c. Chisel

The screenshot shows the Docker Hub interface for the 'alpine' image. At the top, there is a blue header with the Docker Hub logo and a search bar. Below the header, the breadcrumb 'Explore / Official Images / alpine' is visible. The main content area features the 'alpine' image icon, which is a 3D cube with a white 'A' on it. To the right of the icon, the text reads 'alpine Docker Official Image · 1B+ · 10K+'. Below this, a description states: 'A minimal Docker image based on Alpine Linux with a complete package index and only 5 MB in size!'. A tag 'OPERATING SYSTEMS' is displayed below the description. At the bottom, there are tabs for 'Overview' and 'Tags'.

The screenshot shows the GitHub repository page for 'distroless' under the 'GoogleContainerTools' organization. The repository is public and has 4 branches and 0 tags. The main branch is 'main'. A recent commit by 'loosebazooka' is shown, which is a merge pull request #2100 from 'GoogleContainerTools/updat...'. The commit message is 'Merge pull request #2100 from GoogleContainerTools/updat...'. Below the commit, there are two files listed: '.cloudbuild' with the description 'Add cloud build config for lifecycle tagging' and '.github' with the description 'Merge pull request #2058 from GoogleContainerTools/de...'. The repository name 'distroless' is highlighted in orange.



# Comparing the approaches

IMAGE	ID	DISK USAGE	CONTENT SIZE	EXTRA
dessert-shop:naive	7209228cba2d	1.79GB	414MB	U
dessert-shop:v2	d6f729c5a686	330MB	80.7MB	U
dessert-shop:v3	b84048dba141	208MB	53.4MB	U



**Rockcraft:** an open-source tool to build OCI-compliant container images.

- ✓ **Declarative.** No imperative scripts.
- ✓ **Pebble.** Process orchestrator.
- ✓ **Chisel!** Trusted Ubuntu, but *just* what you need.



Let's try it out...

``git checkout v4-rockcraft``  
To inspect the code and tinker





# Chisel slices repo

The screenshot shows the GitHub repository page for `canonical / chisel-releases`. The navigation bar includes links for Code, Issues (71), Pull requests (55), Agents, Actions, Security and quality, and Insights. The left sidebar shows the file tree with the `slices` directory expanded, listing files like `apache2-bin.yaml`, `apache2-data.yaml`, `apache2.yaml`, `apparmory.yaml`, `apt.yaml`, `aspnetcore-runtime-10.0.yaml`, `aspnetcore-targeting-pack-10...`, `base-files.yaml`, and `chisel.yaml`. The main content area shows the `slices` directory with a commit by `alesancor1` titled "fix(26.04): add missing dependency to libnode127 (#1015)". Below this, a message states "This branch is 253 commits ahead of and 79 commits behind main". A table lists the commit history for the `slices` directory.

Name	Last commit message
..	
apache2-bin.yaml	feat(26.04): introduce v3 format (#903)
apache2-data.yaml	feat(26.04): introduce v3 format (#903)
apache2.yaml	feat(26.04): shorten coreut11s slice names (#992)
apparmory.yaml	feat(26.04): shorten coreut11s slice names (#992)
apt.yaml	feat(26.04): introduce v3 format (#903)



## #5 Non-root users

Image hardening isn't just about a smaller image.  
It also involves stricter runtime restrictions.

CVE-2019-5736 : Container escape vulnerability





# #6 Rootless runtimes

User



docker CLI



dockerd (running as HOST ROOT)



runc (running as HOST ROOT)



container



# #6 Rootless runtimes



User



docker CLI



dockerd (running as HOST ROOT)



runc (running as HOST ROOT)



container

User (alice)



dockerd-rootless / podman



runc/crun



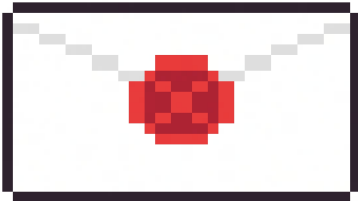
container



# #7 Image signing

Do I know this ingredient is from a trustworthy source?

Sealed: stamp of authority, and a tamper proof



A cryptographic process that verifies the authenticity and integrity of container images



# #8 Provenance

Manufacture date, location,  
batch no., I need to see it all.

How was this container  
image produced?

Full chain of custody.





# #9 SBOMs

A full and complete ingredient list.



...



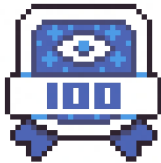
Software bill of materials.

Details every operating system package, third-party library, and application dependency inside a container environment.



# #10 Attestations

Certification badges on the pack



What facts can be cryptographically asserted about this image?

Passed vulnerability scanning

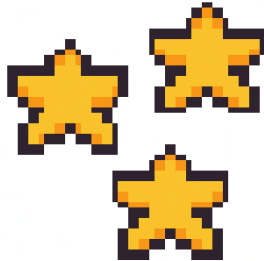
Built from commit abc123

Image has SBOM attached



# #11 SLSA

Levels of food safety



Supply-chain Levels for  
Software Artifacts

Checklist of standards.

“A common language to talk  
about how secure software &  
supply chains are”

<https://slsa.dev/>



Thank you! Questions?

