Zero-Touch OS Infrastructure for Container and Kubernetes Workloads

Kubernetes & CloudNative Meetup Berlin

January 11, 2024

Hello, I'm

Thilo

Thilo Fromm

Engineering Manager, Microsoft

Github: t-lo
Mastodon: @thilo@fromm.social
Email: thilofromm@microsoft.com

Outline

Foundational Concepts

Staying up to Date

Community

Outlook

Container Optimised Linux

Container Optimised Linux

Rethink the OS as an interchangeable commodity

- "Light switch" design philosophy

Handles like a container app / pod

- Simple, declarative configuration
- Innerworks are well abstracted
- Extensive automation

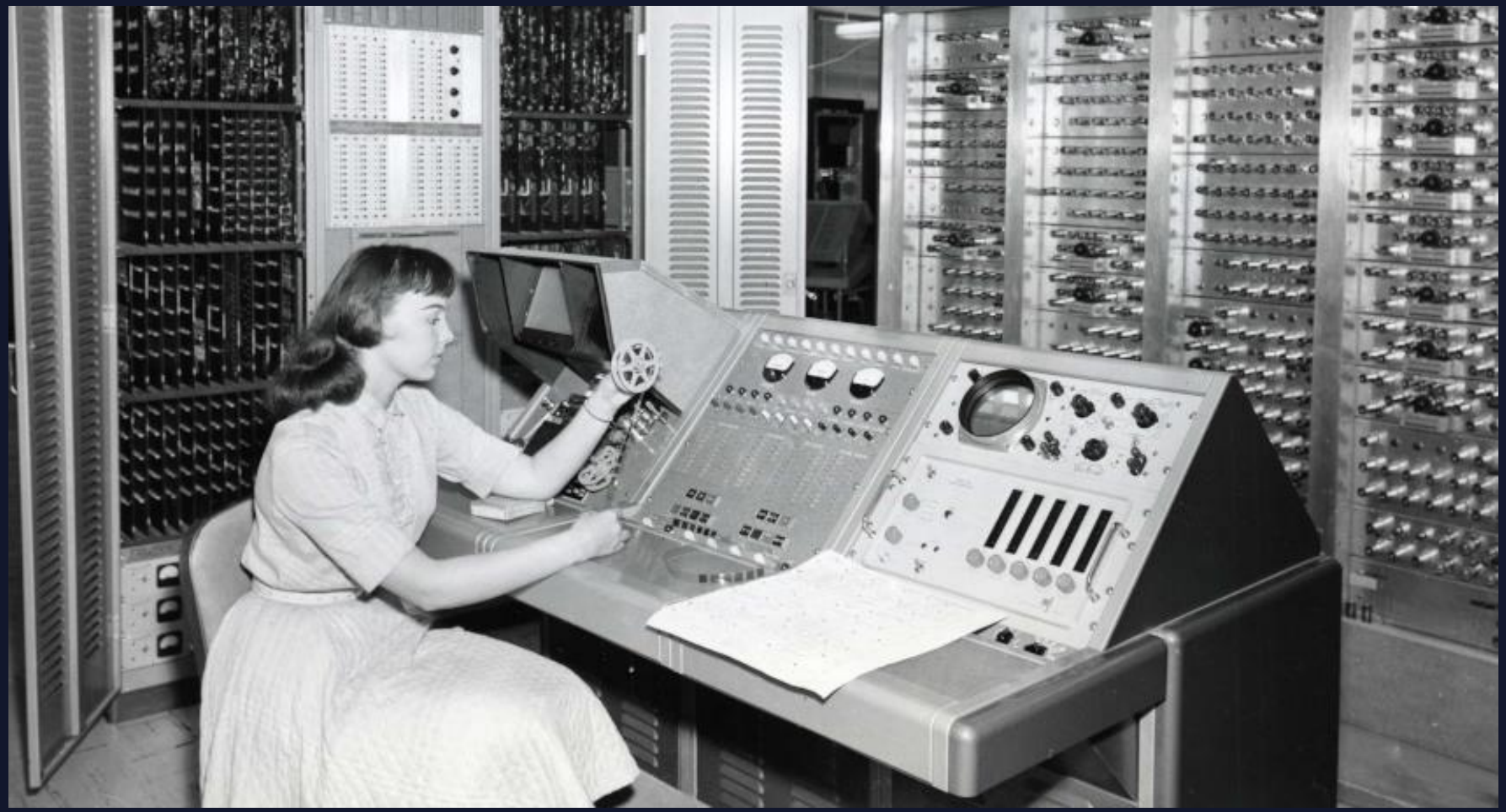
Leverage container isolation from the OS side

- Simple, well-defined interfaces between Apps and OS: Kernel; docker / containerd
- No shared dependencies (libs, config, etc.) between kernel and applications

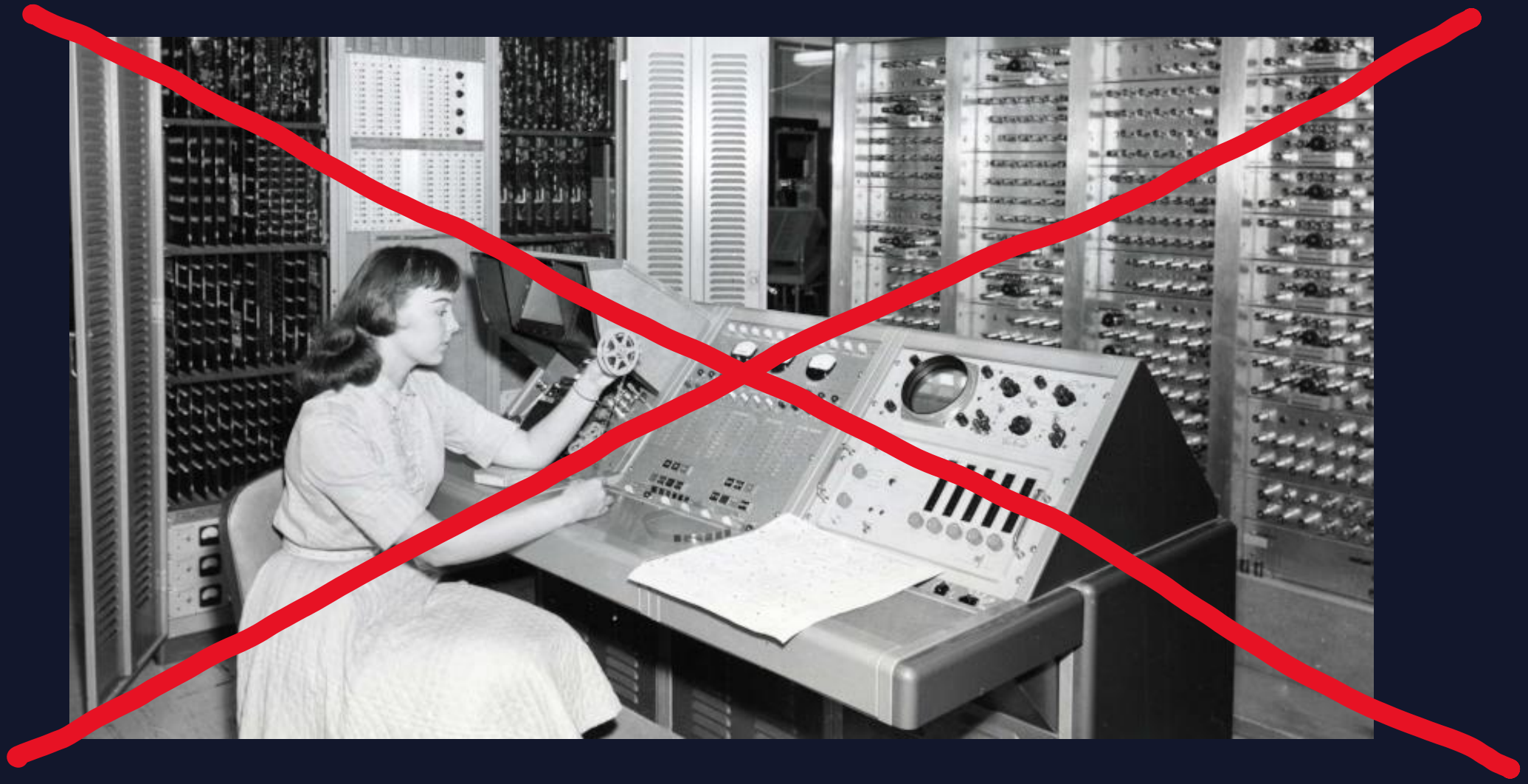
Image-Based OS

- Node is an “instance” of OS image
- Stateless provisioning, verifiable / attestable read-only OS

UX Philosophy

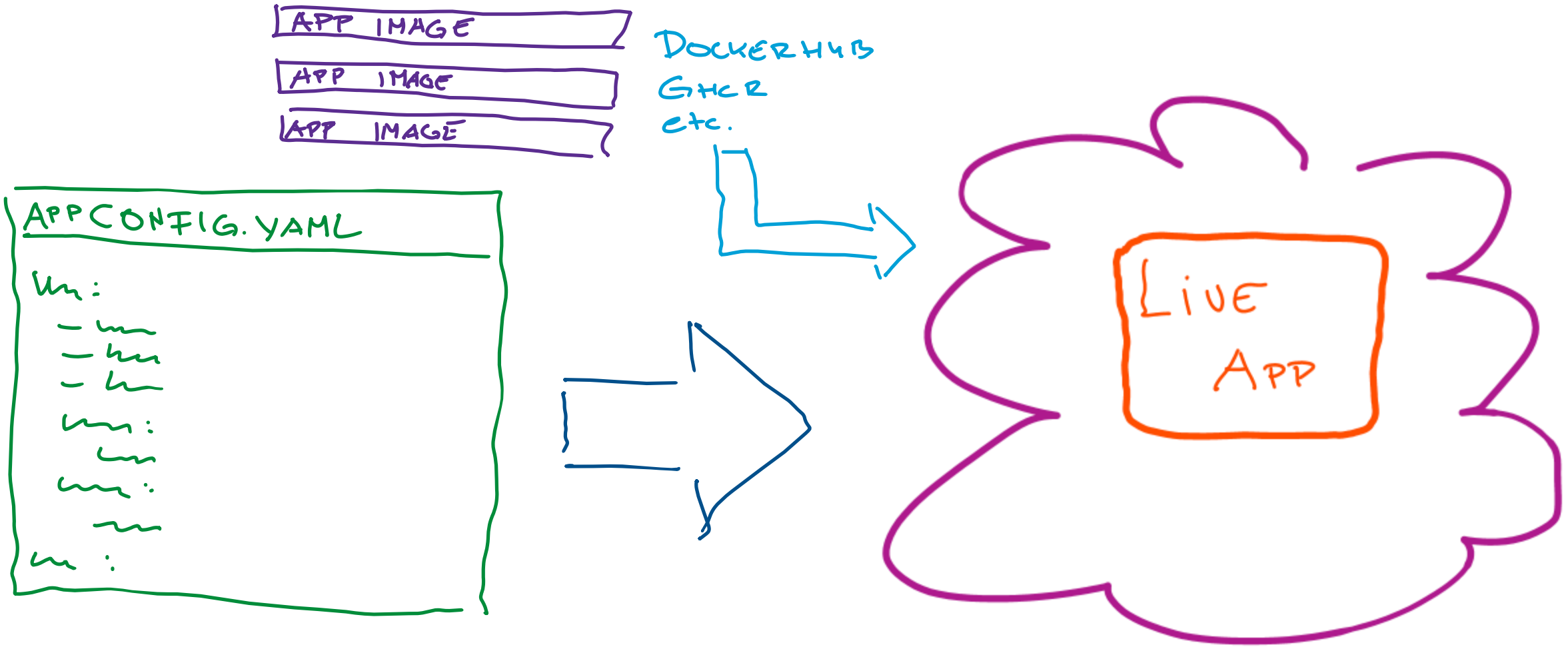


UX Philosophy

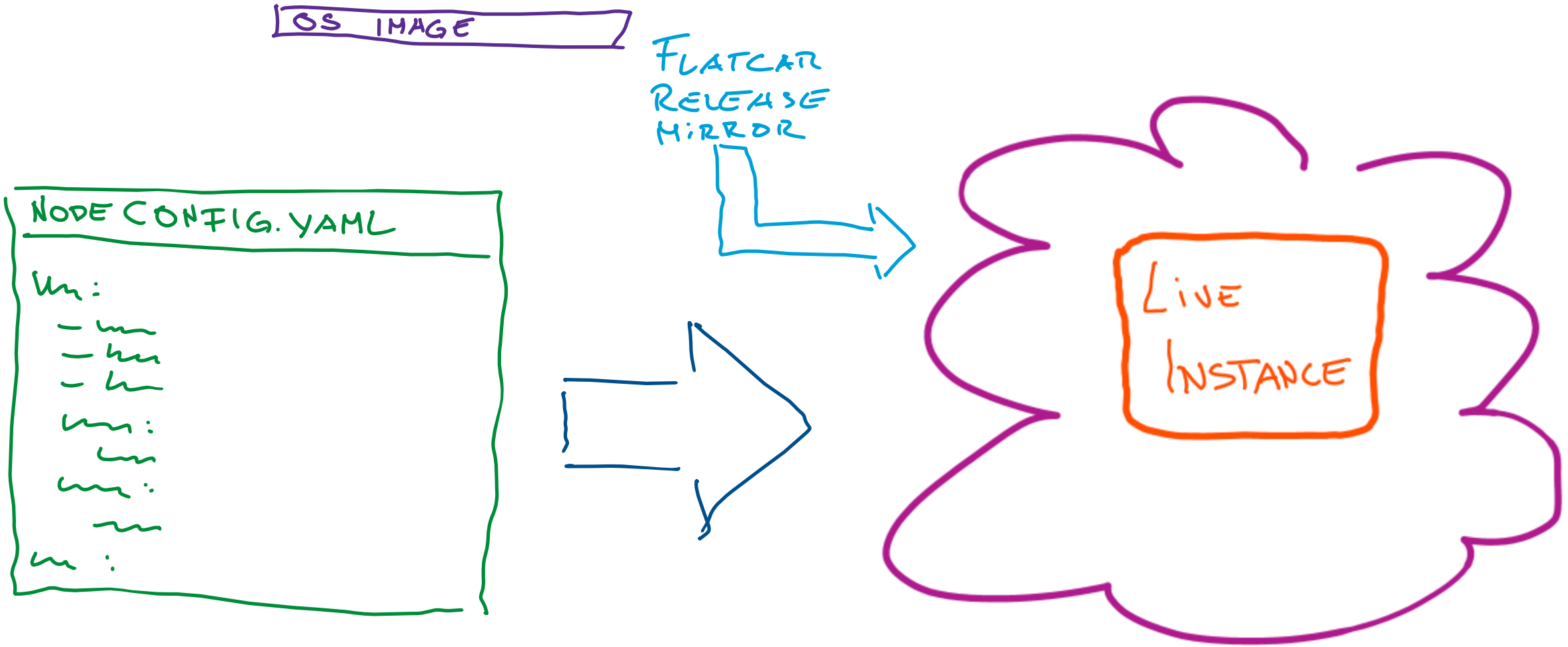


UX Philosophy

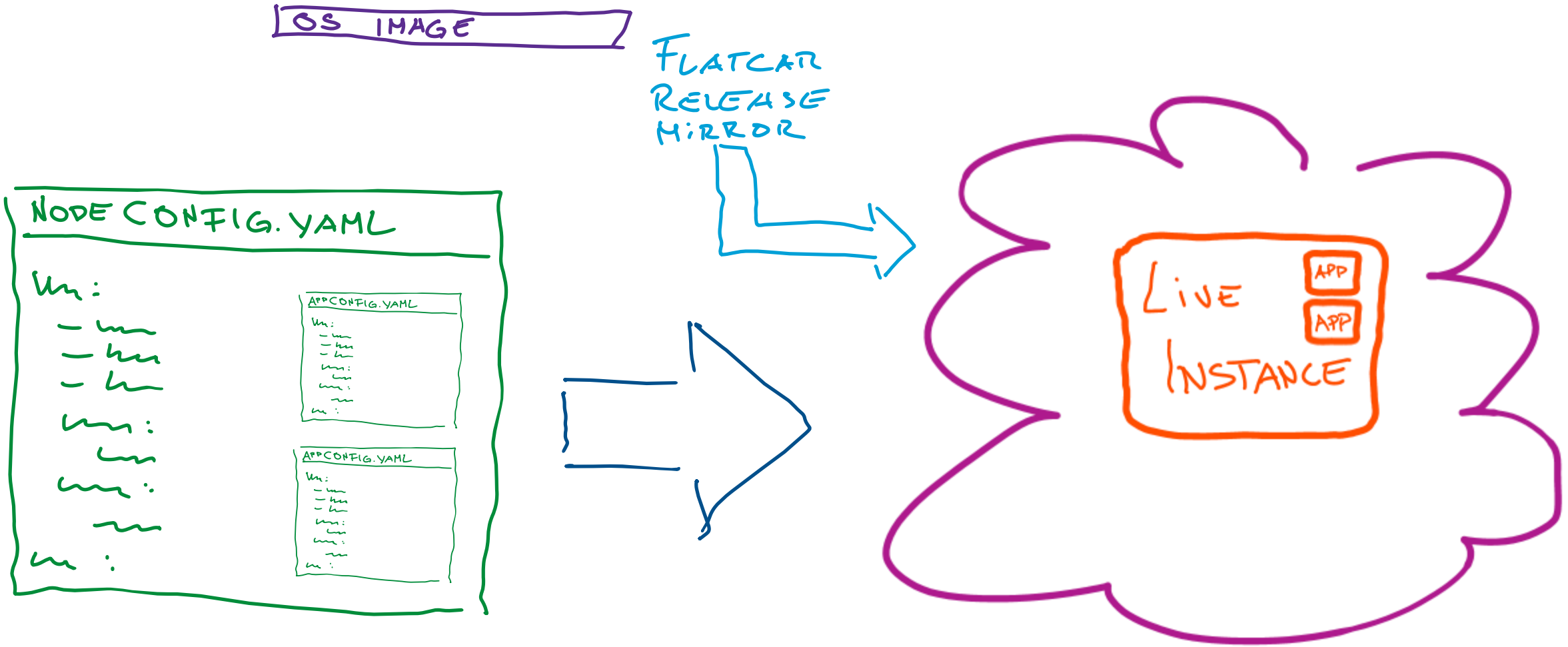
Container / Kubernetes App Provisioning



OS Provisions like a Container App



Bootstrap Initial Apps when Provisioning



Declarative configuration

```
passwd:
  users:
    - name: caddy
      no_create_home: true
      groups: [ docker ]

storage:
  files:
    - path: /srv/www/html/index.html
      mode: 0644
      user:
        name: caddy
      group:
        name: caddy
      contents:
        inline: |
          <html><body align="center">
          <h1>Hello KCD!</h1>
          <img src="kcd.png" alt="KCD logo" />
          </body></html>
    - path: /srv/www/html/kcd.png
      mode: 0644
      user:
        name: caddy
      group:
        name: caddy
      contents:
        local: kcd.png
```

```
systemd:
  units:
    - name: kcd-demo-webserver.service
      enabled: true
      contents: |
        [Unit]
        Description=KCD example static web server
        After=docker.service
        Requires=docker.service
        [Service]
        User=caddy
        TimeoutStartSec=0
        ExecStartPre=-/usr/bin/docker rm --force caddy
        ExecStart=docker run -i -p 80:80 \
                  -v /srv/www/html:/usr/share/caddy \
                  docker.io/caddy caddy file-server \
                  --root /usr/share/caddy --access-log
        ExecStop=/usr/bin/docker stop nginx1
        Restart=always
        RestartSec=5s
        [Install]
        WantedBy=multi-user.target
```

Handles like a Container App

Simple configuration, sensible defaults, applied on provisioning

- Storage / partitions / filesystems / luks, networking, kernel command line arguments
- Users, groups, ssh access, systemd units, custom directories and files (inline or download)
- No boilerplate configuration
- No OS configuration drift – node config is applied only once.

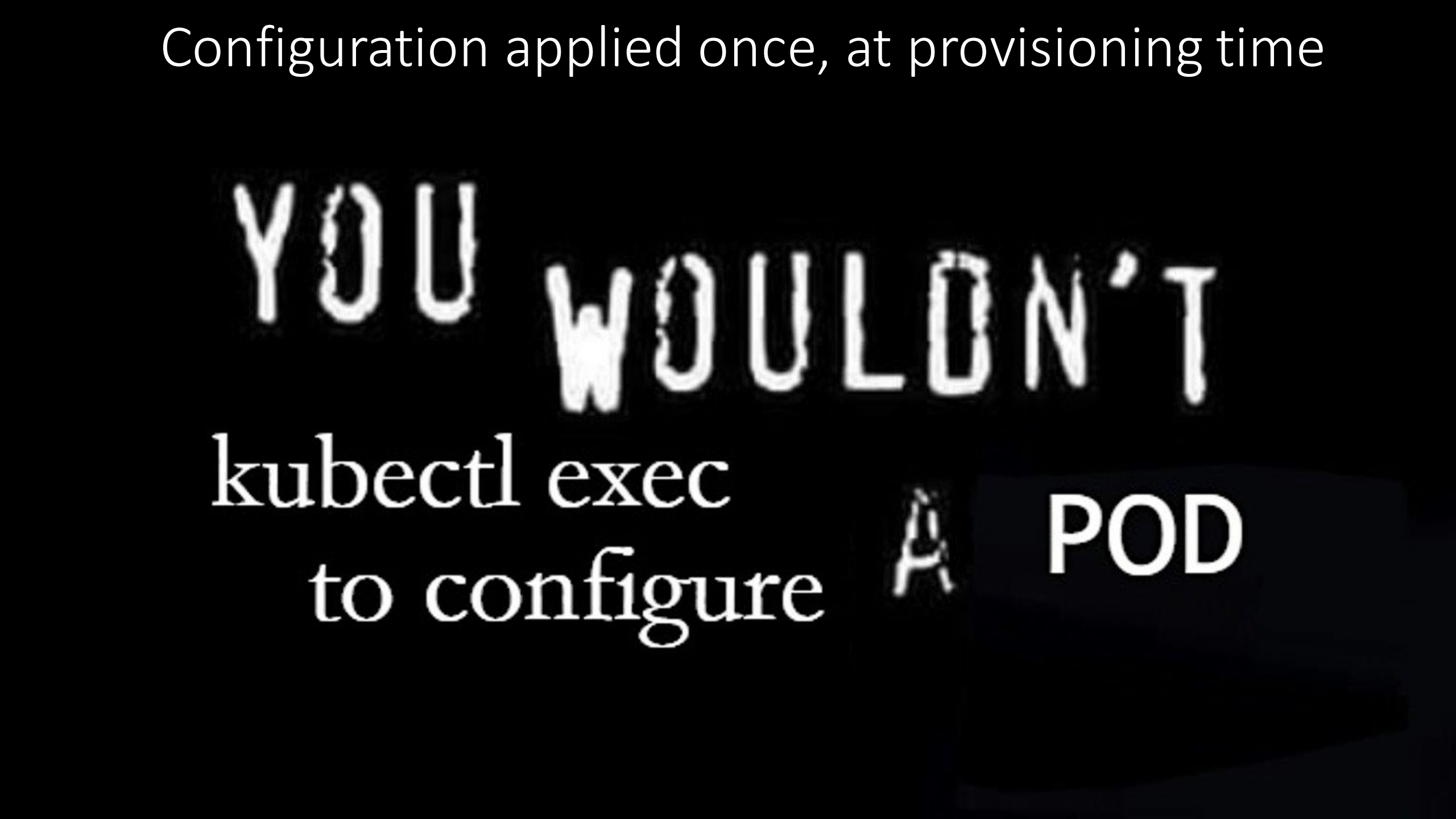
Extensive Automation

- Support for many cloud providers and private clouds included
- Terraform support, Go library
- ClusterAPI

Just Works

- Well tested on all supported vendors / private clouds / environments
- Strong focus on compatibility and support for existing workloads
  - (cgroups, Ignition v2 backwards compat)

Configuration applied once, at provisioning time



Large-Scale deployments? ClusterAPI!

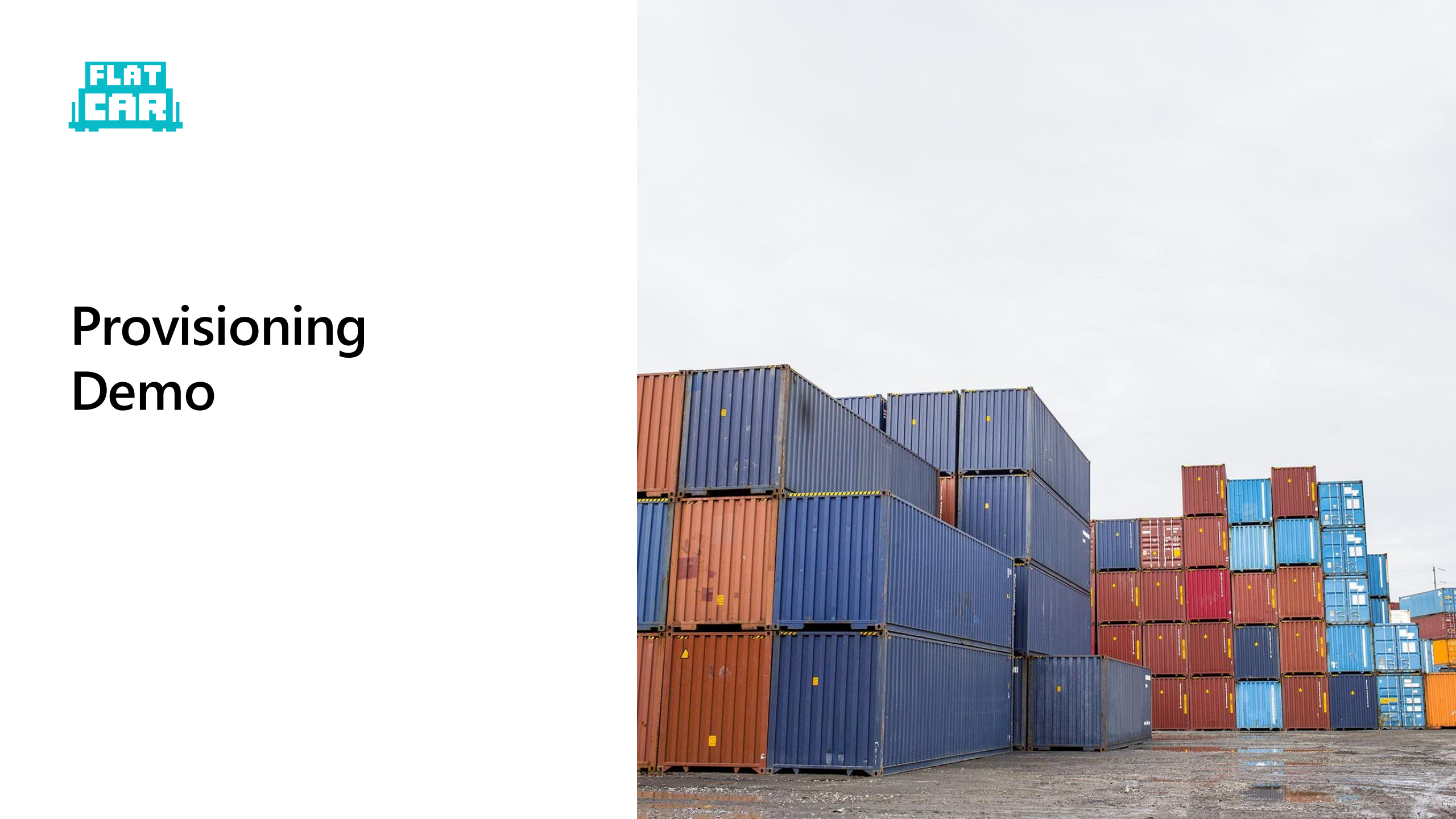
Supported out-of-the box by Core CAPI and image-builder

Multiple large vendors are supported

- AWS
- Azure
- VSphere
- OpenStack

GCP support is work-in-progress.

Piloting sysext CAPI deployments (composed at provisioning, updatable)



Provisioning Demo

Leverage Container Isolation

Container apps are self-contained and run isolated

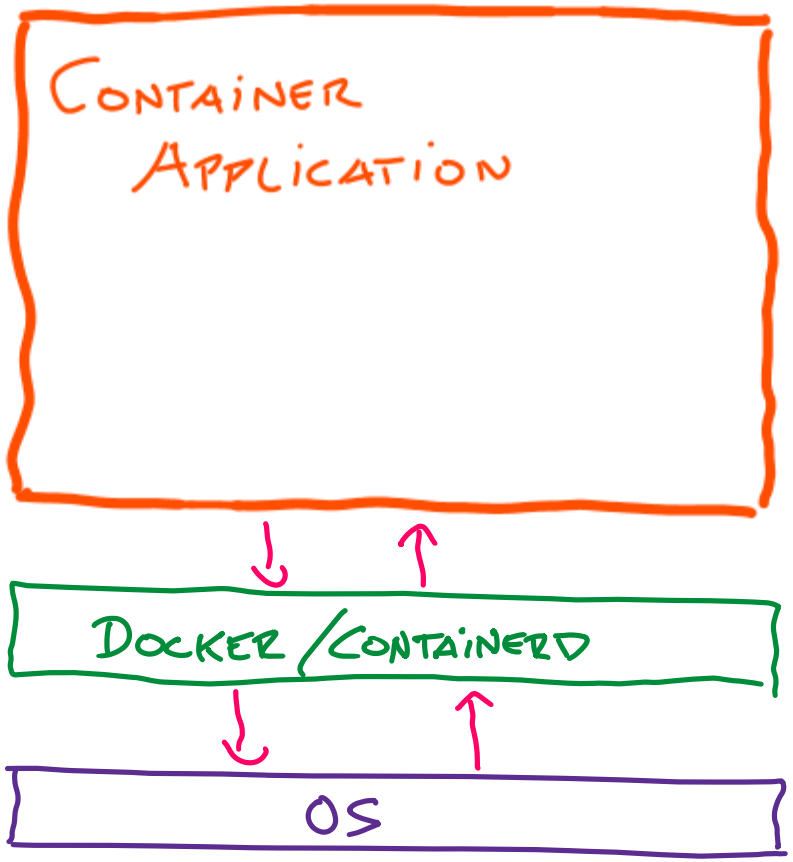
- From each other, but also from the OS
- ➔ portable apps

No inter-dependencies OS <-> App

- No shared libraries
- No shared binaries
- No shared configuration

Well-defined interfaces OS <-> App

- Docker or Containerd
- Kernel (drivers, networking , storage, etc)



Leverage Container Isolation from the OS side

Well-defined interfaces OS <-> App

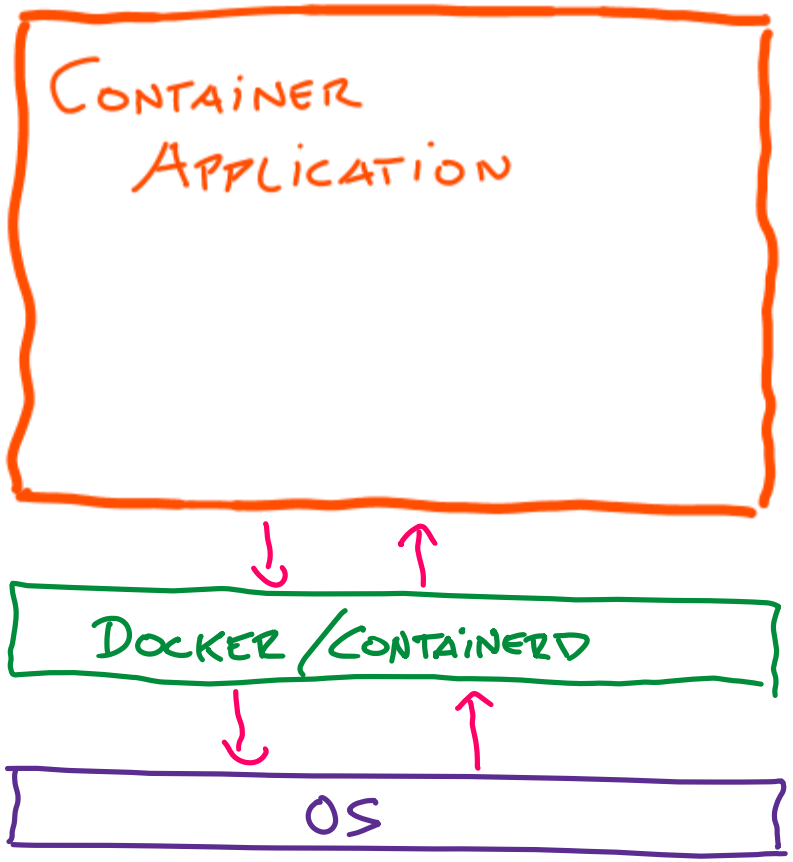
- Very few components
- Easy to test thoroughly

No inter-dependencies OS <-> App

- OS can be updated w/o impacting applications
- Major version jumps w/o side effects

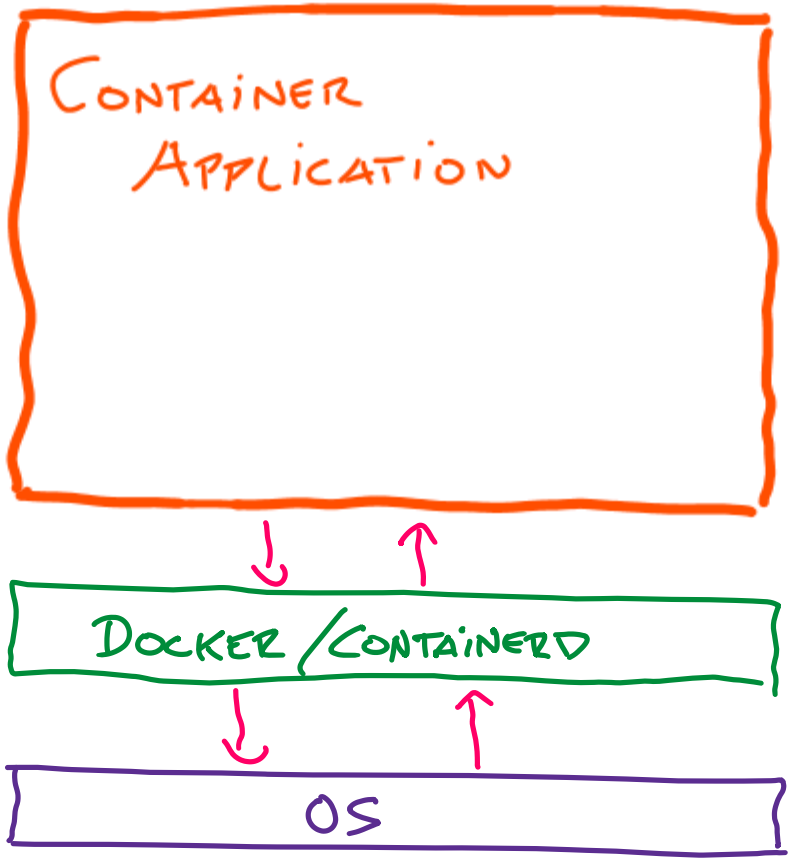
Container apps are self-contained and run isolated

➔ Interchangeable OS



Leverage Container Isolation from the OS side

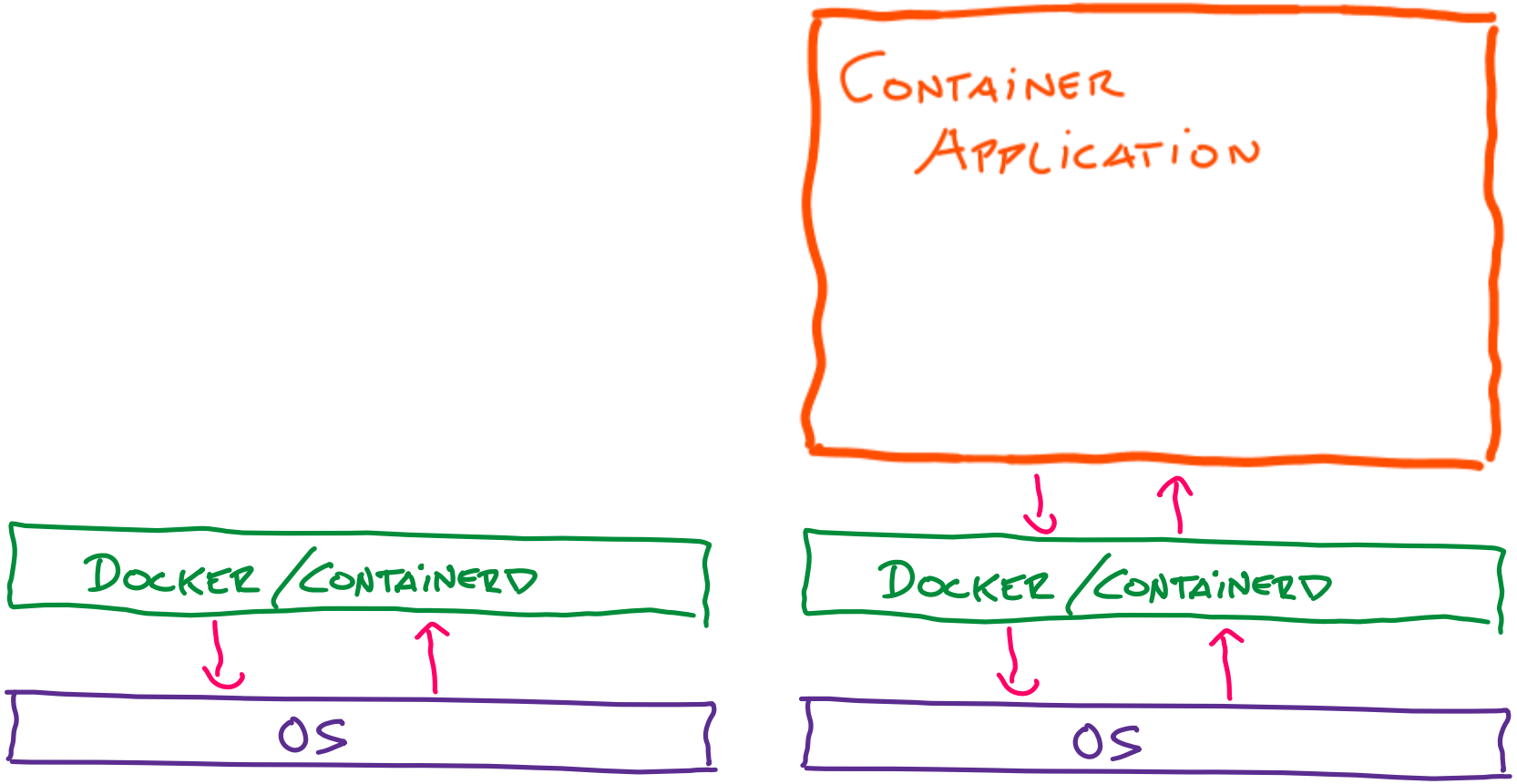
Atomic In-Place Updates



Leverage Container Isolation from the OS side

Atomic In-Place Updates

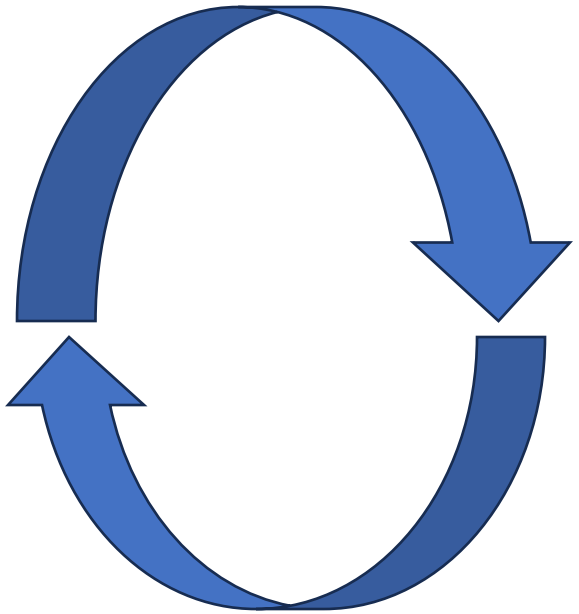
1. Stage

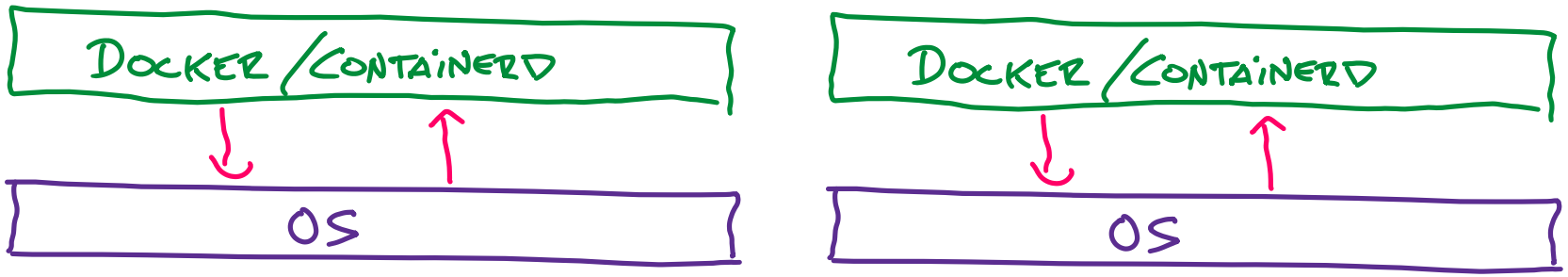


Leverage Container Isolation from the OS side

Atomic In-Place Updates

1. Stage
2. Activate (Reboot)

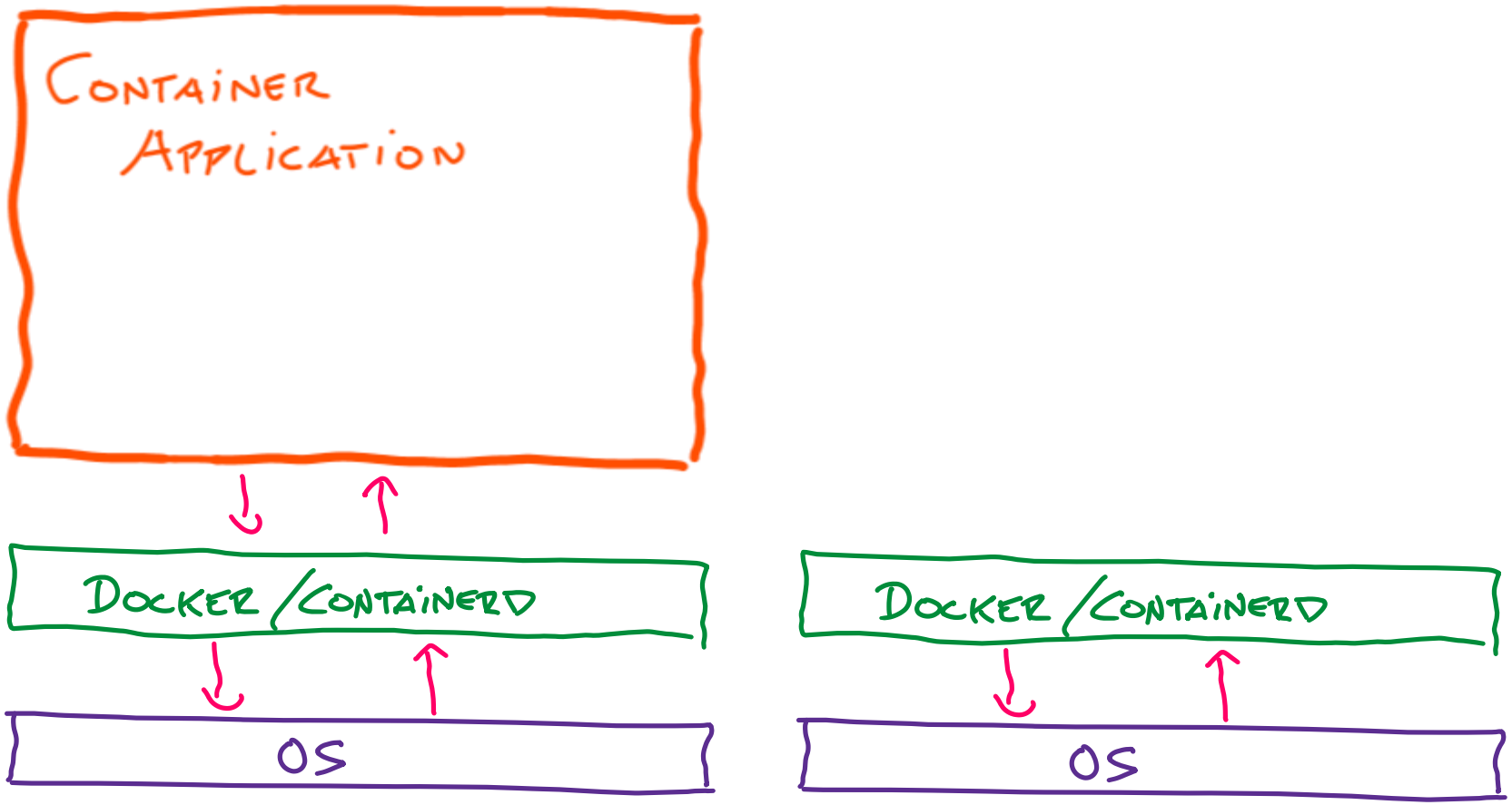




Leverage Container Isolation from the OS side

Atomic In-Place Updates

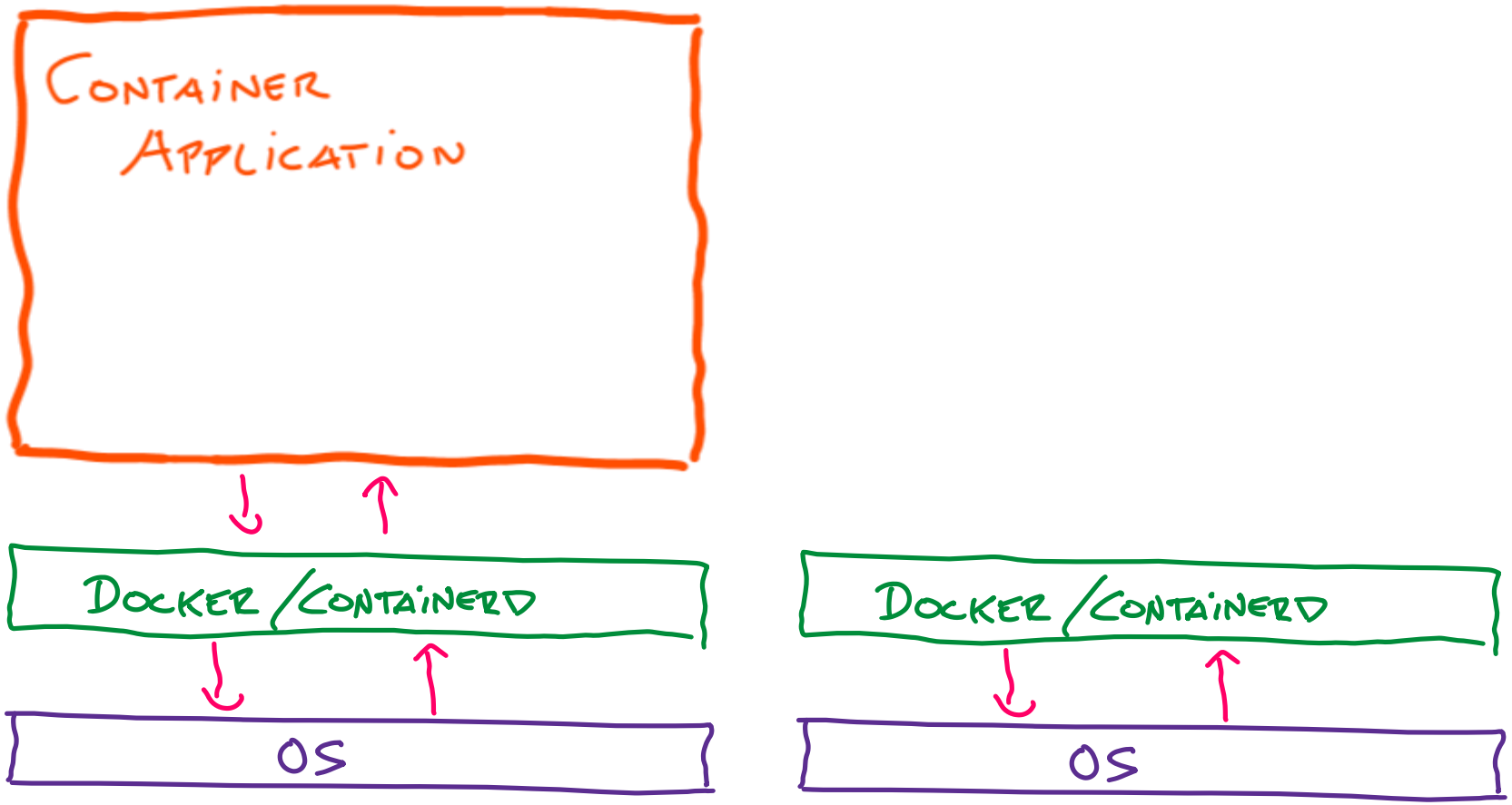
1. Stage
2. Activate
3. Done



Leverage Container Isolation from the OS side

Atomic In-Place Updates

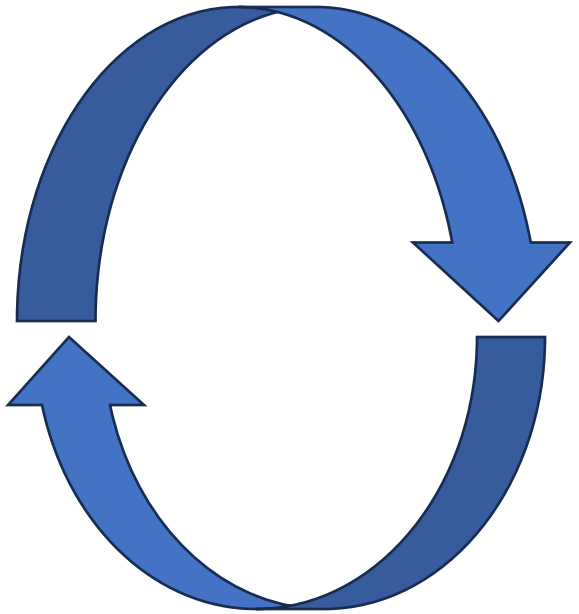
1. Stage
2. Activate
3. Done?

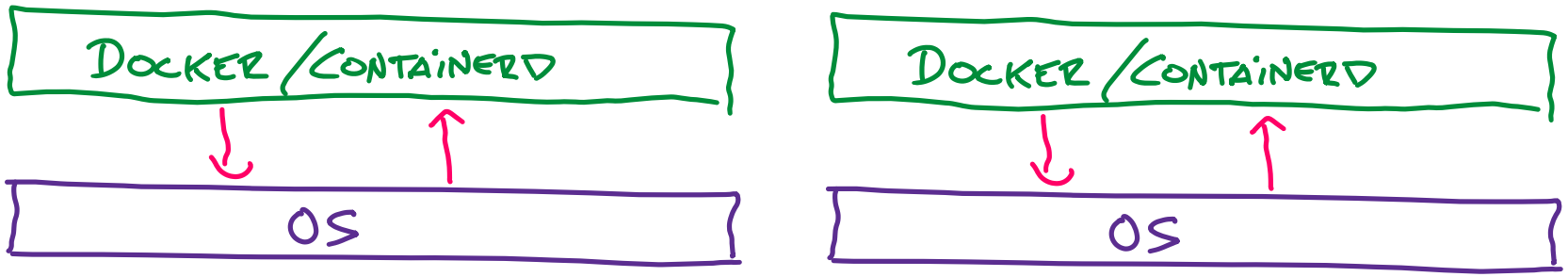


Leverage Container Isolation from the OS side

Atomic Roll-Backs

1. Stage
2. Activate
3. Done?
4. Roll Back

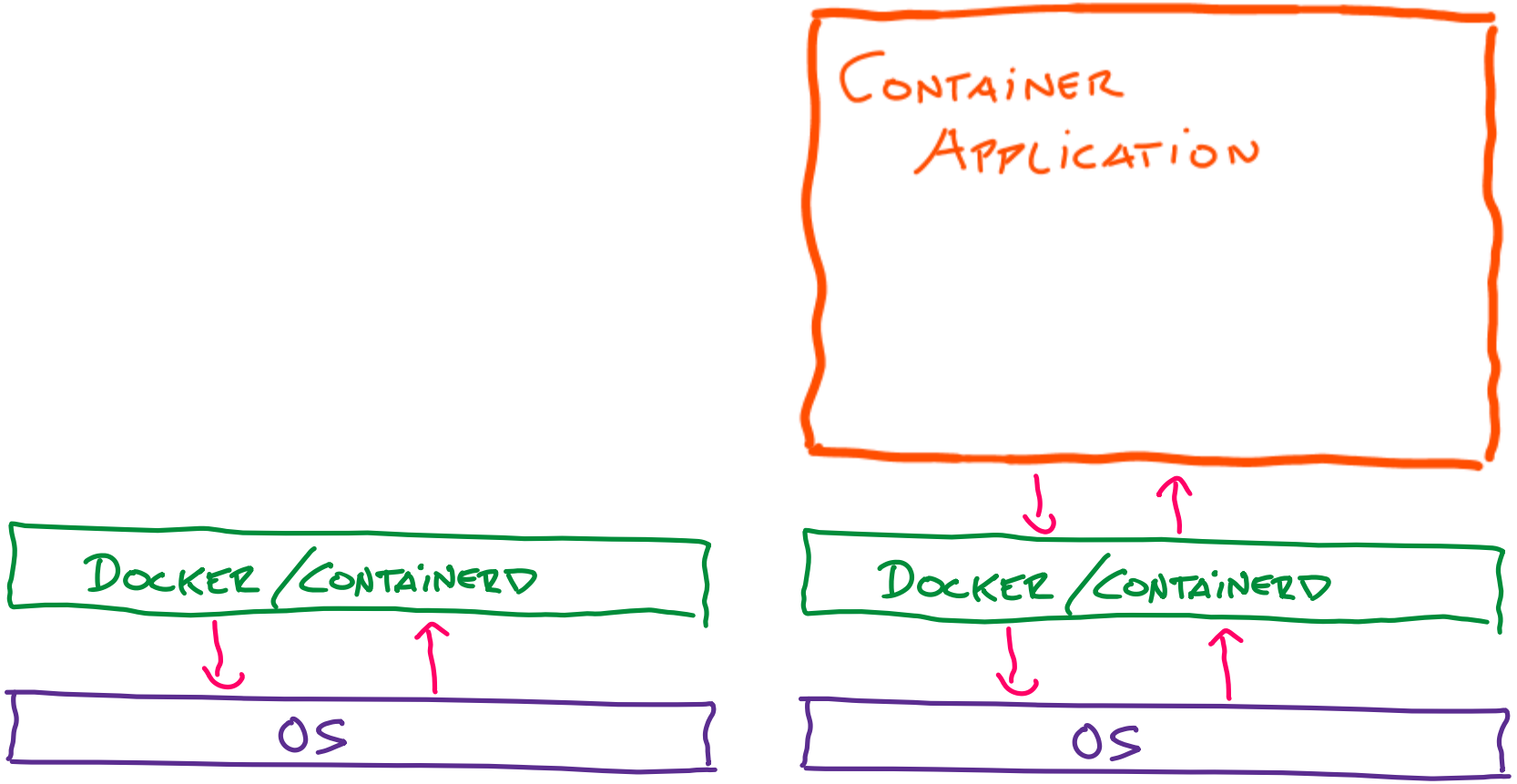




Leverage Container Isolation from the OS side

Atomic Roll-Backs

1. Stage
2. Activate
3. Done?
4. Roll Back



Update
Demo

(Usually automated.
Manual ONLY
for demo purposes)

Excursion: Flatcar Heritage

Flatcar Container Linux started as a Friendly Fork of the epochal
CoreOS Container Linux, which in turn was derived from
Chromium OS, which is based on
Gentoo Linux.

We always build from sources
like Gentoo does.

The OS is
immutable and shipped as a full disk image
uses A/B partitioning for updates (and rollbacks), distributed via a stateful protocol (Omaha)
like Chromium OS.

Flatcar includes
a minimal set of applications and tools tailored to only run containers
it uses declarative configuration applied at provisioning time
like CoreOS.

Image-Based OS

Provisioning uses disk images, no package management

- Full, self-contained disk images. No software version drift.
- Vendor support only included in vendor-specific images (AMI, VHD, etc.)
- Always built from scratch, SLSA attestation included

All OS binaries are in `/usr`, which is on a separate partition

- The /usr partition is read-only and dm-verity protected (root hash in initrd)
- There are actually 2 /usr partitions: one is active, the other is used to stage the next update

Updates ship a full OS partition image

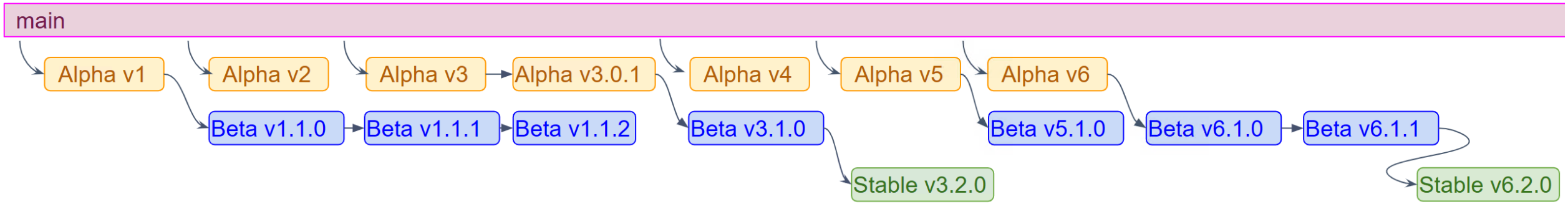
- In-place updates via A/B partitioning, retains node state (DB nodes using ephemeral disks etc.)
- OS version always correlates to specific version sets of all tools shipped
- Updating and re-provisioning result in the same node state

/etc is an overlayfs backed by /usr/share/flatcar/etc

- Distro default configs shipped (and updated) in /usr
- User changes in /etc override defaults

Managing Updates and Releases

Release Channels



To receive updates, Flatcar instances are subscribed to one of 4 channels

"Alpha" for development. Fully tested but may contain incomplete features.
"Beta" for production use. Recommended for canaries
"Stable" for widespread production use. Stabilised through user feedback / Beta canaries.
"LTS" for environments where any change imposes significant costs. Based on "golden" Stable.

Defaults to "stable" but can be customised using declarative configuration.

"Alpha" has a release cadence of 2-4 weeks (major releases).
"Beta" gets a new major release every 1-2 months.
"Stable" major releases occur every 3-4 months. Only the latest stable release is supported.
"LTS" is supported for at least 18 months.

Using Release Channels to Keep your Workloads Safe

Use stable for most workloads, and run a few Beta canaries

- Each Beta release passes our full test suite before release (Alpha too, for that matter)
  - However, some workloads might trigger edge cases which are hard to model in test cases
- Canaries smoke-test incoming changes and detect issues early

Report Issues detected by canaries

- We're not shipping a Stable that causes known issues
- The issue will be fixed in the next Beta, before changes go stable

==> Clusters will receive stable versions that are proven to work

Update Strategies

Update Strategies customisable via declarative configuration

- By default, nodes download & reboot as soon as update is available

Basic and advanced update strategies available

- Single node: Maintenance windows (date / time)
- Cluster w/o control plane: synchronisation via custom etcd lock (max number of nodes to reboot)
- Kubernetes: update operator (FLUO) w/ node draining, reboot, un-cordoning
  - Also, support for KureD (CNCF Sandbox project, originally by WeaveWorks)

Operate your own Update Server

Flatcar updates use the Omaha protocol

- Nebraska, an open source Omaha implementation, is part of our project.
- Nebraska supports a "downstream" mode for more control over upstream releases

Advantages

- Custom node grouping (e.g. by region), controlled roll-out
- Feedback on update roll-out as errors / rollbacks are reported back to Nebraska
- Overview of version distribution in your fleets


Community

Flatcar is a community driven distro, all work done openly

Day-to-day interactions

- Matrix and Slack channels for day-to-day interactions
  - Drop by and speak up any time, maintainers are always around

Long term planning and short term coordination

- Planning and Roadmap boards are public
- Status updates discussed in public monthly developer sync calls
- User input, demos, etc. in separate monthly Office Hours call

Focused Bug smashing and Doc writing days

- Live streamed, drop in and join us!

Flatcar Community

Community-driven FOSS project

No single vendor, full community stewardship

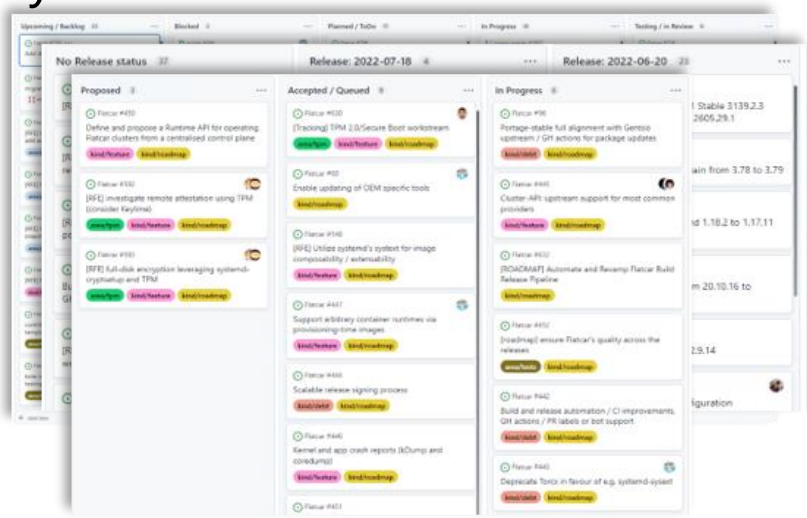
Matrix, Slack - Our day-to-day comms

Office hours - Every 2nd Tuesday, 3:30pm UTC

Dev Sync - Every 4th Tuesday, 3:30pm UTC

Roadmap, Implementation, Releases

Publicly planned, community driven



Portable, Easy to use SDK

Focus on low entry bar to OS Development

(Some Gentoo knowledge is useful though)

Used by Maintainers and in our automation

Includes easy-to-run, full test suite

```
git clone https://github.com/flatcar/scripts.git
cd scripts
git checkout alpha-3794.0.0

./run_sdk_container -t ./build_packages
./run_sdk_container -t ./build_image
./image_to_vm.sh --from=../build/images/amd64-usr/latest/ \
                 --format=qemu_uefi --image_compression_formats none

./run_local_tests.sh
```

Join us!

User feedback, bug reports, feature requests

We are a user-driven community and are always looking for your input

Good First Issues, Bug smashing and doc writing days

Join the contributors!

Become a Maintainer

Flatcar is a maintainers governed project.
Regular contributors are offered maintainership status.

Outlook

Systemd Sysext Integration

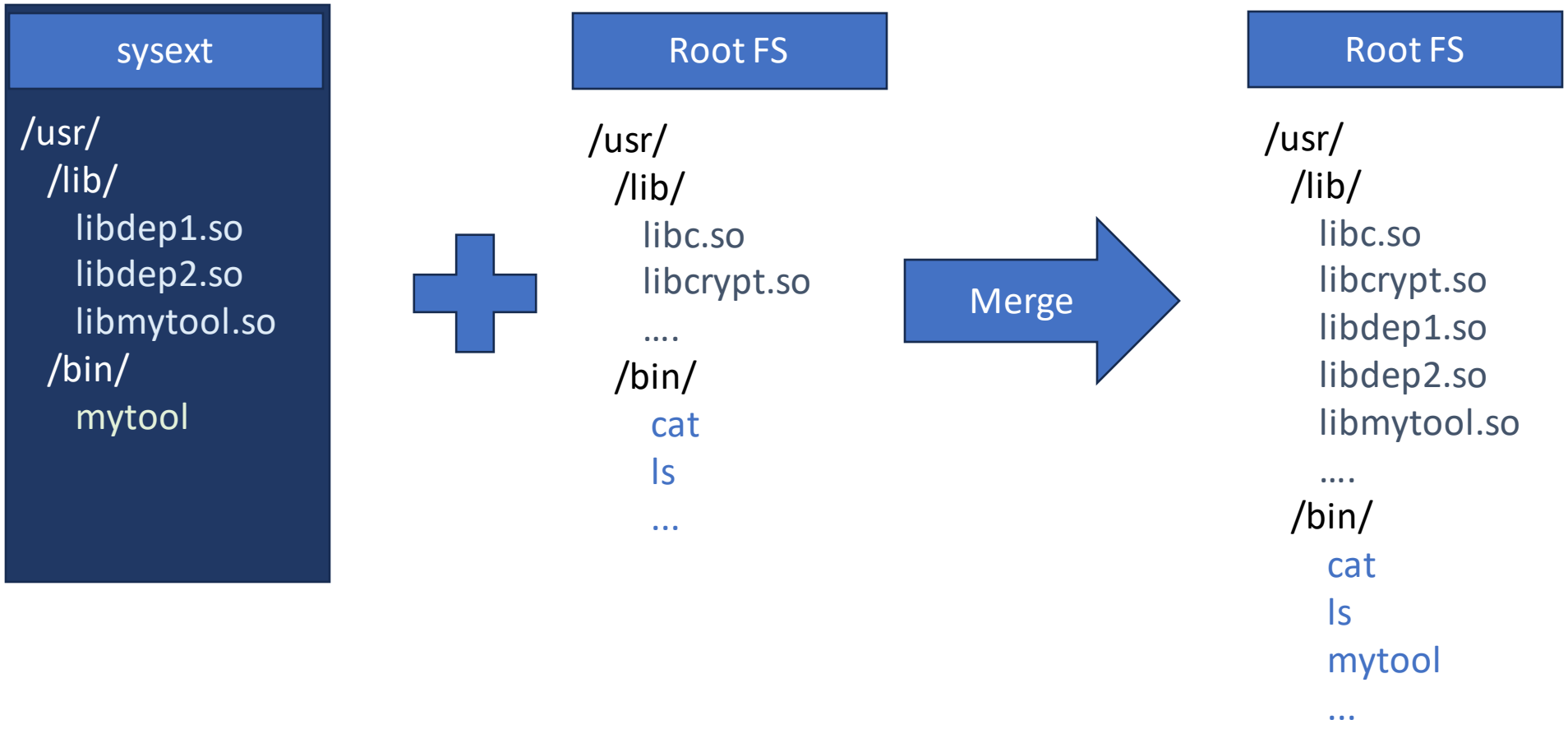
What's a sysext?

- Sysexts are OS extensions shipped in filesystem images
- These images are transparently mounted onto /usr (and optionally, /opt) and can be stacked
- It's a feature built into systemd starting July 2021 (v448 or later)

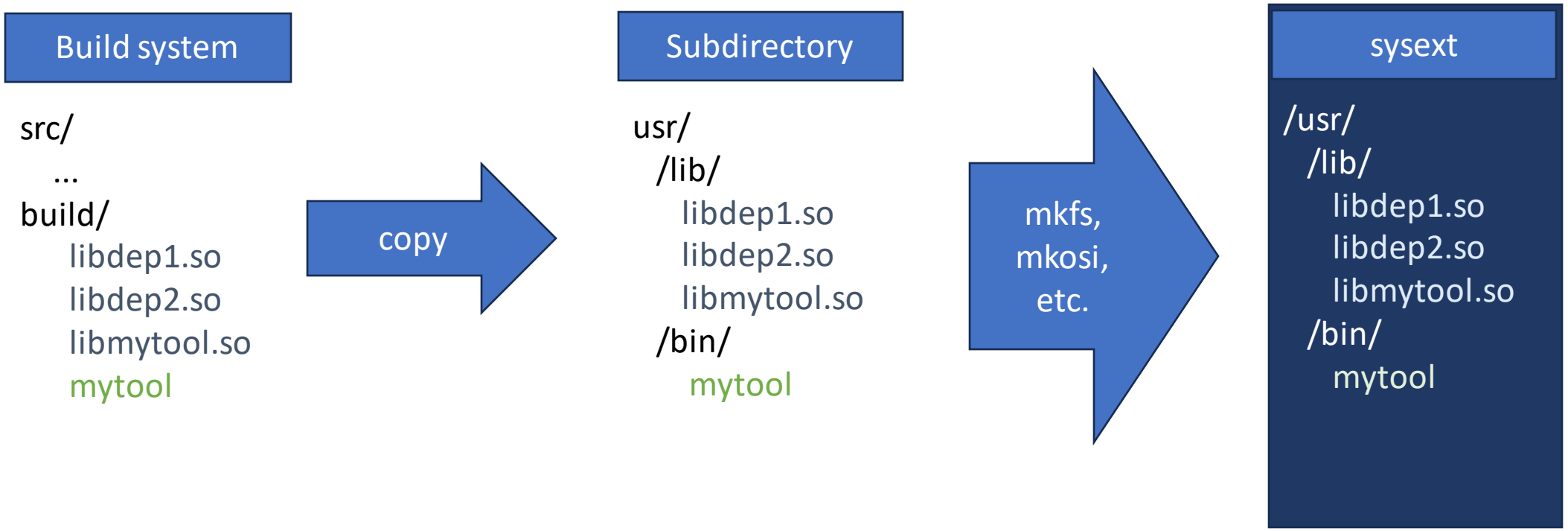
Towards composable images!

- We're looking into an even leaner base image, customisable via sysexts
- "Core" sysexts supported by Flatcar upstream, e.g. Kubernetes / CAPI, Podman, etc.

Using Sysexts



Building Sysexts



Sysext Demo

We've Taken the First Steps

Flatcar uses sysexts for OEM support

- Cloud-specific tools not part of base image - separate OEM partition on vendor specific images
- Now converted to / shipped as sysexts (Azure, QEmu, more to come)
- Eases maintenance and enables simple and secure updating

Support and build tools are available today

- Flatcar stable release fully supports applying (merging) sysext
- Alpha SDK ships with build tools for custom sysexts; lets you extend base image

Towards composable images!

- We're looking into an even leaner base image, customisable via sysexts
- List of "Core" sysexts supported by Flatcar upstream, e.g. Kubernetes / CAPI, Podman, …
- Find out more in our sysext bakery: https://github.com/flatcar/sysext-bakery

CNCF submission

Flatcar was submitted as an incubating project to the CNCF

- We're fundamentally a community-driven, cloud-native project
- We have passed Governance review and are currently working with TAGs Security and Runtime
- The help and guidance we receive from the CNCF is impressive
  - huge thanks to all CNCF folks involved!
- Check out the proposal: https://github.com/cncf/toc/pull/991

Wrap Up

Leverage Isolation of OS and Apps

Declarative Configuration at Provisioning

Atomic, Automated Updates

Community driven

Composable images with Sysext

Submitted to the CNCF for Incubation

Thank you

The Community's
Container Linux