

Your GPU Nodes Deserve Better: Flatcar Container Linux for AI Workloads

KubeCon + CloudNativeCon North America 2026

Jan Bronicki & Sayan Chowdhury

whoarewe



Jan Bronicki

Flatcar Maintainer

Software Engineer at Microsoft

@John15321

jan-bronicki



Sayan Chowdhury

Flatcar Maintainer

Software Engineer at Microsoft

@sayanchowdhury

yudocaa

Agenda

- The community behind Flatcar
- Flatcar in five minutes, live in a VM
- sysexts and Kubernetes, live in a VM
- Release channels
- The GPU node problem: NVIDIA drivers as sysexts, live on AWS
- Live: an LLM, GPU metrics, and Triton on Flatcar

Community Stewarded

What that means, and who we are

Community stewarded

- **Fully open, fully free:** Apache 2.0, public roadmap, public issue tracker
- **No single owner:** the project and trademark belong to the CNCF and the Linux Foundation
- **CNCF project:** neutral governance under the Cloud Native Computing Foundation
- **You can join:** SIG meetings are public; anyone can propose changes or maintain

Flatcar is a **CNCF** project



CLOUD NATIVE
COMPUTING FOUNDATION

The community

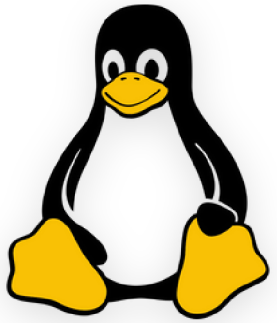


We work with

- **Gentoo** and **Fedora CoreOS**
- **systemd**, **Dracut**, **Grub**, **Afterburn**
- Openwall **oss-security** non-disclosure list
- Co-founders of the **UAPI Group**, cross-distro SIG for image-based Linux









Runs everywhere



In production at



What Problem Do We Solve?

Why the OS is still the odd one out

Everything is code. Almost.

Already declarative

- Dockerfiles
- Kubernetes manifests, Helm charts
- GitOps workflows
- CI/CD pipelines

Still treated differently

- Host OS installed interactively
- Changed over time
- Repaired manually when it breaks

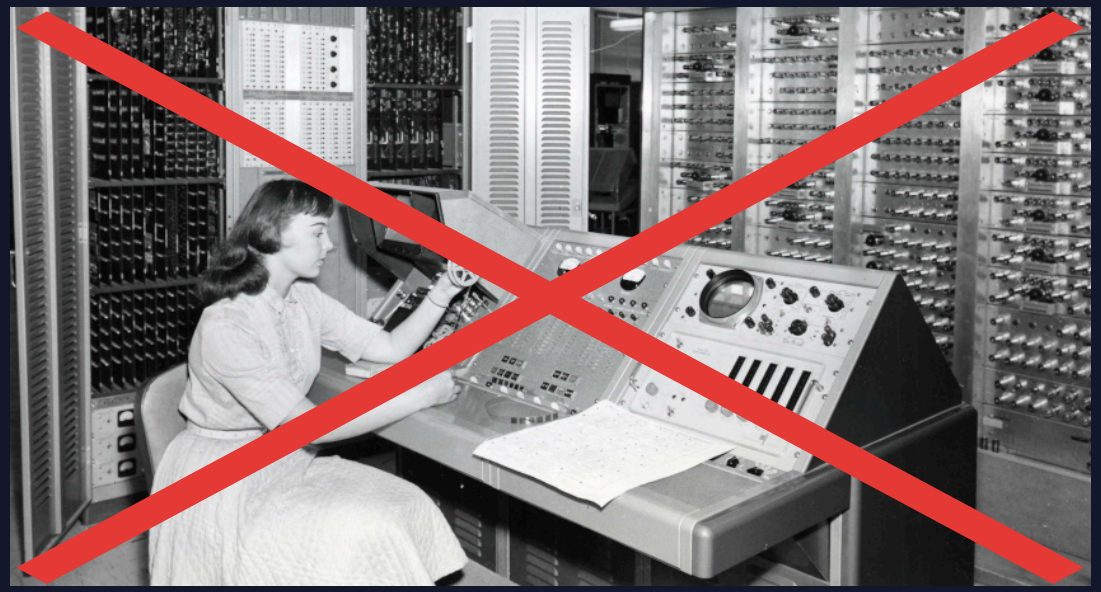
Flatcar?



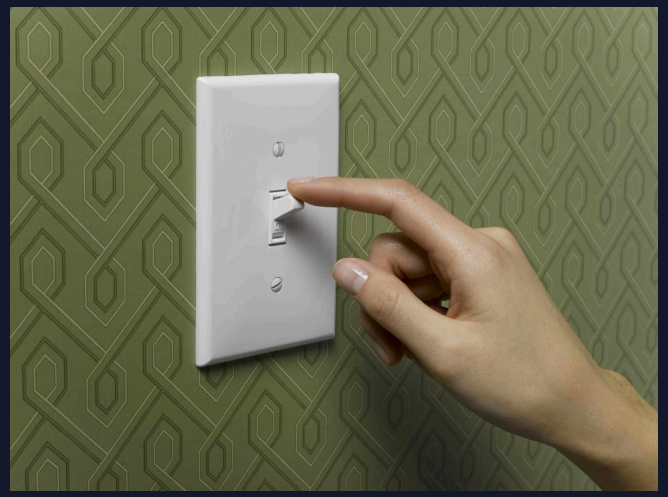
Flatcar!


RINGLING BROS. THE GREATEST SHOW ON EARTH BARNUM & BAILEY
RINGLING BROS. THE GREATEST SHOW ON EARTH BARNUM & BAILEY
RBBX 80701

The UX we chose



General-purpose Linux



Flatcar

Operating Systems as Code

A versioned, declarative artifact. Provisioned, not installed.

Provisioned, not Installed

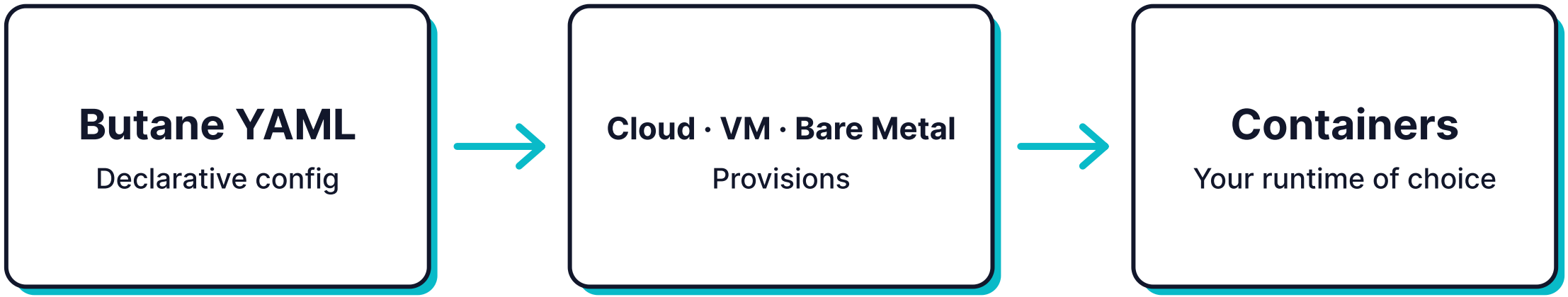
Traditional install

- Interactive choices during setup
- Every machine ends up a little different
- Manual updates, package churn, drift
- No separation between OS and config

Flatcar: provisioned

- One declarative config, applied at boot
- Every machine identical, reproducible
- Whole-OS atomic updates, no drift
- Same idea as containers, with the config in Git

Flatcar Workflow



The Container Philosophy, for the OS



	Containers	Operating System
Recipe	Dockerfile	Butane / Ignition
Image	Docker image versioned, immutable	Flatcar image versioned, immutable
Instance	Container disposable	Node reprovisionable
Update	New image tag replace, don't patch	New OS release A/B swap, don't patch

What Butane looks like




```
# Butane config header: declares this is a Flatcar config
variant: flatcar
version: 1.0.0

# Drop in a systemd unit. Flatcar installs and starts it at first boot
systemd:
  units:
    - name: nginx.service
      enabled: true
      contents: |
        [Service]
        ExecStart=/usr/bin/docker run --rm -p 80:80 nginx
        [Install]
        WantedBy=multi-user.target
```

Boot it in a VM

```
# Transpile Butane to an ignition config (JSON)
$ butane -o config.ign config.bu

# Boot Flatcar in QEMU with the config applied
$ ./flatcar_production_qemu.sh -i config.ign
```

The config gets handed to Flatcar as **user-data**. Same channel every major cloud, hypervisor, and bare-metal provisioner uses to seed an instance.

It came up serving

```
$ systemctl is-active nginx.service
active

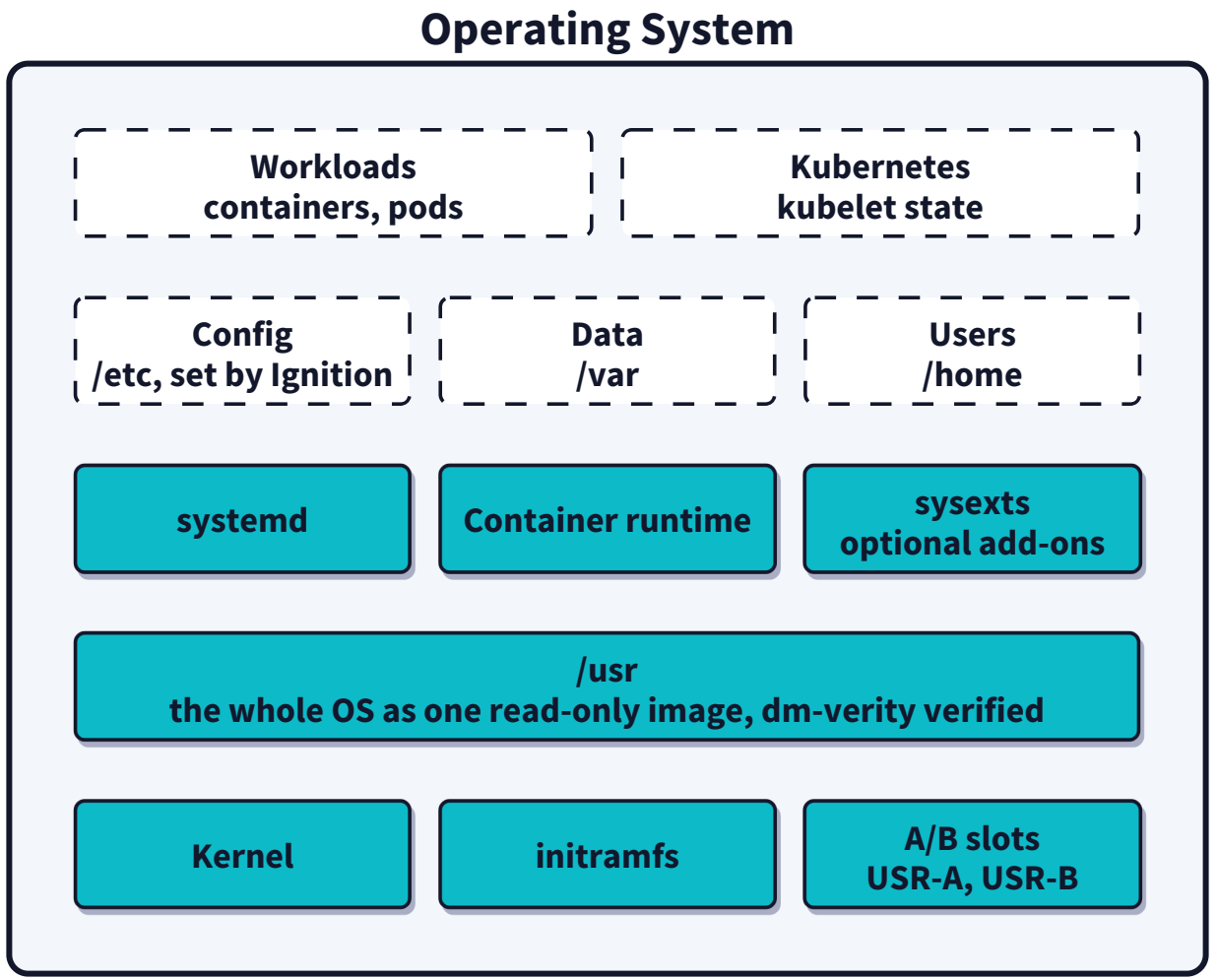
$ curl -sI http://localhost
HTTP/1.1 200 OK
Server: nginx/1.27.3
Content-Type: text/html
```

No login. No post-boot setup. The machine came up already doing its job.

Immutable by Design

- **First boot** provisions from config. After that: the base OS doesn't change.
- The OS is **read-only** and **dm-verity protected**.
- **No package updates**: the entire OS updates as one unit.
- Same config + same base image = **identical machine every time**.
- **sysexts** for optional add-ons.

What is Immutable?



Writable: yours

Read-only: Flatcar's

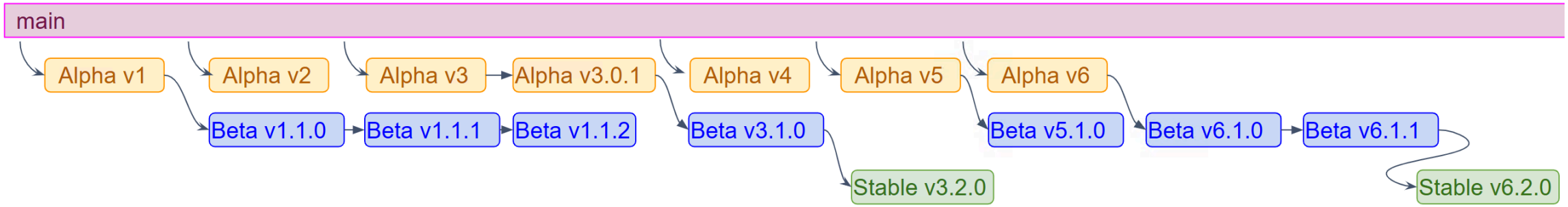
A/B Updates

1	**Fresh install**	A 1.2.3	B 1.2.3	Image ships both slots identical. Only true at first boot.
2	**Update staged**	A 1.2.3	B 1.2.4	New image written over the passive slot. No downtime.
3	**Reboot swaps**	A 1.2.3	B 1.2.4	Boot flips to B. If it fails, reboot back to A at 1.2.3.
4	**Steady running**	A 1.2.3	B 1.2.4	B live. A untouched: it's your rollback target.
5	**Next update**	A 1.2.5	B 1.2.4	1.2.5 overwrites the old 1.2.3 on A. Next reboot: A active, B rollback.



↻ The passive slot is **never auto-synced**. It's your rollback until the next update lands.

Channels



- **Alpha**: fully tested, may have incomplete features. Developers.
- **Beta**: production-ready. Canary alongside stable.
- **Stable**: widespread production. Promoted from beta.
- **LTS**: slower-moving track for environments that need it.

> Every commit runs the full test suite: unit, integration, and product tests across every supported cloud. By the time a version ships, even **Alpha**, it's been through hundreds of runs.

LIVE DEMO · 01

Flatcar on QEMU

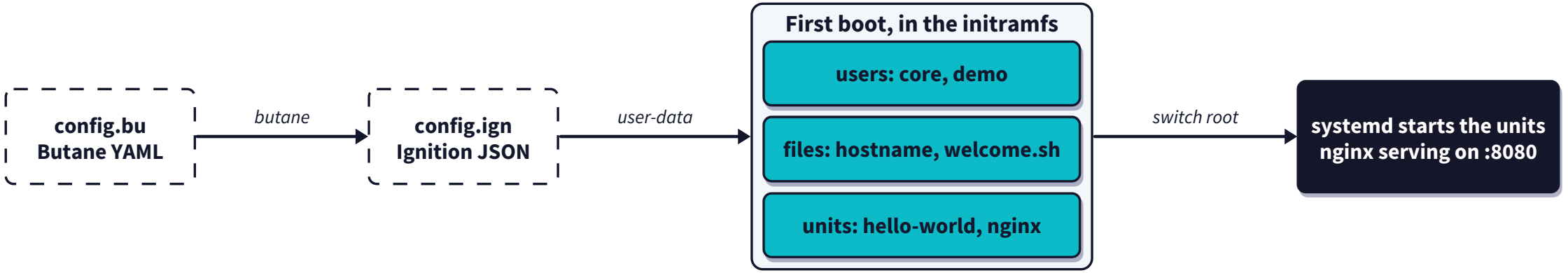
WATCH FOR

- No installer: the disk image **is** the OS
- Login prompt in seconds
- SSH works with a key Ignition put there

```
demos/01-getting-started-on-flatcar-on-qemu
$ ./demo-tmux.sh
# right pane, once sshd is up:
$ ssh -i ssh/demo_ed25519 \
    -p 2222 core@localhost
$ cat /etc/os-release
```

First boot does the work

Ignition runs **once**, in the initramfs, **before** the real system starts. Users, files, and units already exist when systemd comes up.



Your config | Applied by Ignition | Running system

LIVE DEMO · 02

Ignition doing real first-boot work

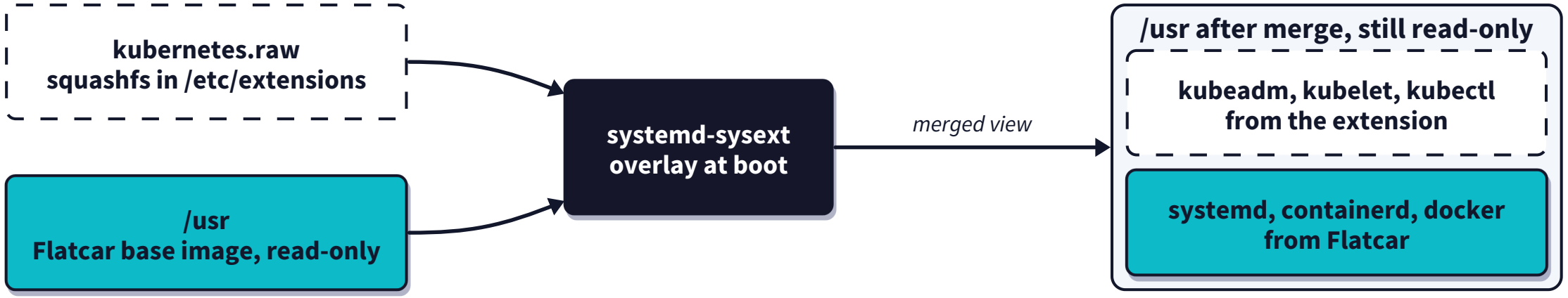
WATCH FOR

- Hostname and a second user, set declaratively
- Containers running before anyone logs in
- nginx reachable from the laptop on `:8080`

```
demos/02-ignition-first-boot-setup
$ ./demo-tmux.sh
$ hostname
$ groups demo
$ docker ps
$ journalctl -u hello-world.service
# back on the laptop:
$ curl http://localhost:8080
```

System extensions

No package manager, read-only `/usr`. A **sysext** is a squashfs image that systemd **overlays** onto `/usr` at boot. Add the file: the software appears. Remove it: gone.



LIVE DEMO · 03

Adding Kubernetes with a sysext

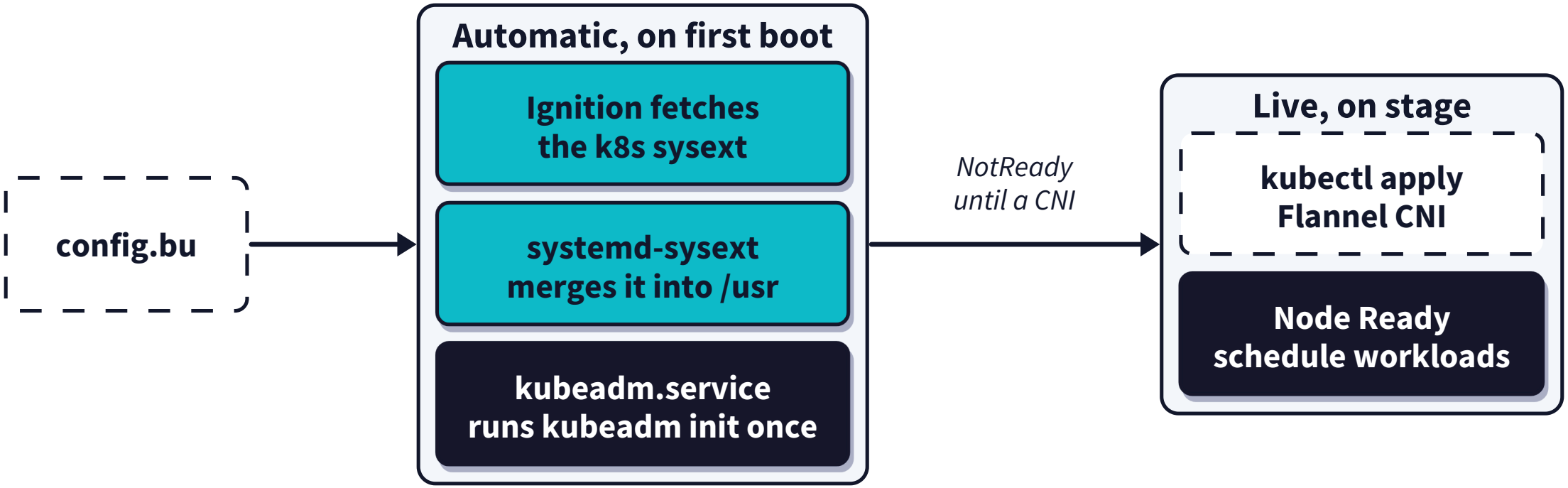
WATCH FOR

- `kubernetes` listed as a merged extension
- kubeadm, kubelet, kubectl on a box with no package manager
- `/usr` is still read-only

```
demos/03-system-extensions
$ ./demo-tmux.sh
$ systemd-sysext status
$ which kubeadm kubelet kubectl
$ kubeadm version
$ sudo touch /usr/bin/hello
# Read-only file system
```

A Kubernetes node from one config

Same sysext, plus one systemd unit that runs `kubeadm init` on first boot. A `ConditionPathExists` guard makes it run **exactly once**: reboots resume the cluster.



You | Flatcar | Kubernetes

LIVE DEMO · 04

Kubernetes on Flatcar

WATCH FOR

- `kubeadm init` already ran during boot
- Node goes **NotReady** to **Ready** with one apply
- A workload scheduled on the node

demos/04-deploy-kubernetes-with-flatcar

```
$ ./reset.sh && ./demo-tmux.sh
$ kubectl get nodes
# NotReady: no CNI yet
$ kubectl apply -f <flannel v0.28.10>
$ kubectl get nodes
$ kubectl get pods -A
$ kubectl taint node flatcar-demo \
    node-role.kubernetes.io/control-plane-
$ kubectl apply -f app-demo.yaml
```

GPU Nodes, Done Right

NVIDIA drivers without giving up immutability

The GPU node problem



- Drivers are tied to the kernel, so every OS update is a risk
- Hand-patched GPU nodes drift apart
- Rolling back a driver on a mutable host is painful

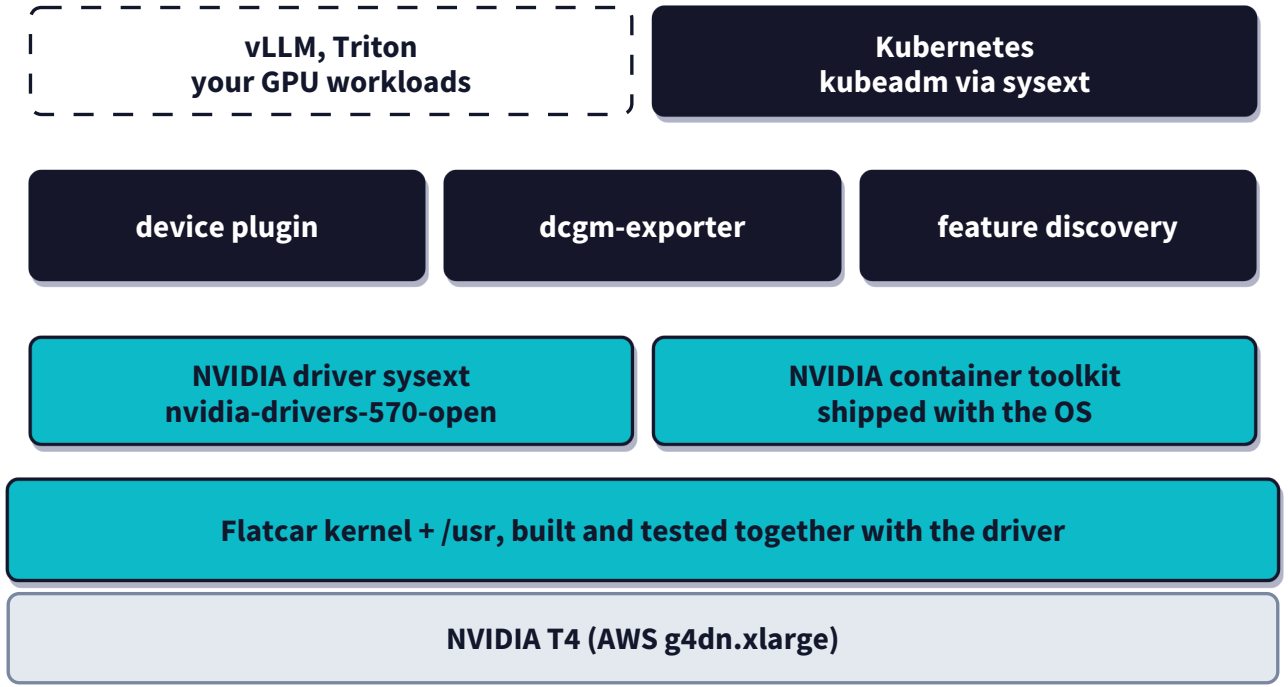
NVIDIA drivers as sysexts



- Driver shipped as a systemd-sysext image, merged over read-only `/usr`
- Pinned per kernel and Flatcar release, versioned like any artifact
- Updated and rolled back **with** the OS, atomically, through A/B

Who owns what on a GPU node

Flatcar ships the driver and the container toolkit, built against its own kernel. The GPU Operator runs with `driver.enabled=false` and `toolkit.enabled=false`: it only does the Kubernetes side.



Flatcar ships it | Kubernetes / GPU Operator | Your workload | Hardware

LIVE DEMO · 05

A GPU node on AWS

WATCH FOR

- `nvidia-smi` works on a fresh node: no driver install step
- The driver listed as a merged sysext
- A pod that asks for a GPU gets one

```
demos/05-nvidia-driver-on-flatcar-aws
$ ./demo-tmux.sh
$ systemd-sysext status
$ nvidia-smi
# CNI + untaint, then (full flags: COMMANDS.md)
$ helm install gpu-operator nvidia/gpu-operator \
    --set driver.enabled=false \
    --set toolkit.enabled=false
$ kubectl get pods -n gpu-operator
$ kubectl run gpu-test ... -- nvidia-smi
```

Driver upgrades ride the OS update

The driver is built **with** the kernel, so it is upgraded **with** the OS: one atomic A/B update. If the new driver misbehaves, reboot into the old slot.



LIVE DEMO · 06

NVIDIA driver upgrade in place

WATCH FOR

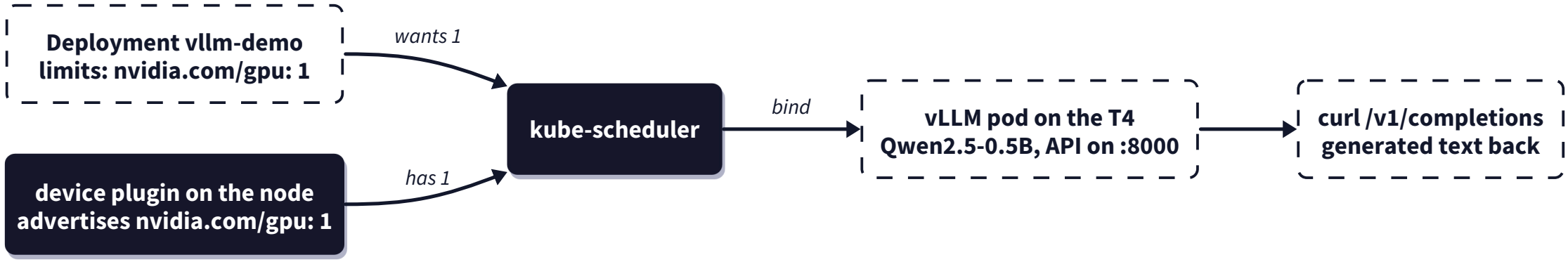
- Before: driver branch **570-open**
- One command: download, apply, reboot
- After: **580-open**, config untouched

demos/06-nvidia-driver-upgrade-in-place

```
$ ./demo-tmux.sh
$ systemd-sysext status
$ nvidia-smi
$ update_engine_client -status
$ update_engine_client -update
# reboots, SSH back in with the same key
$ systemd-sysext status
$ nvidia-smi
```

Scheduling a model onto the GPU

The device plugin advertises the GPU as a countable resource. A pod that asks for `nvidia.com/gpu: 1` lands on it, and a second one waits.



LIVE DEMO · 07

vLLM inference on Flatcar

WATCH FOR

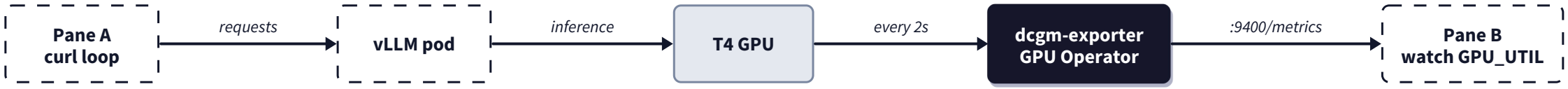
- A model generating text on the T4
- An OpenAI-style API, served from our node
- A second replica stuck: **Insufficient nvidia.com/gpu**

demos/07-vllm-inference-on-flatcar

```
$ kubectl apply -f vllm-demo.yaml
$ kubectl get pods -w
$ CLUSTER_IP=$(kubectl get svc vllm-demo \
    -o jsonpath='{.spec.clusterIP}')
$ curl $CLUSTER_IP:8000/v1/completions \
    -H "Content-Type: application/json" \
    -d '{"model":"Qwen/Qwen2.5-0.5B-Instruct",
         "prompt":"Flatcar Container Linux is",
         "max_tokens":40}'
$ kubectl scale deploy vllm-demo --replicas=2
```

Seeing the GPU work

`dcgm-exporter` comes with the GPU Operator. It reads GPU counters and serves them as Prometheus metrics on `:9400`, the same endpoint you would scrape into Grafana.



Your terminals / workload | GPU Operator | Hardware

LIVE DEMO · 08

GPU metrics with DCGM

WATCH FOR

- GPU utilization near 0 while idle
- A spike every time a request lands
- Metrics from the GPU Operator, nothing extra installed

demos/08-dcgm-metrics-demo

```
$ ./demo-tmux.sh
# pane A: the curl loop against vLLM
$ while true; do curl -s ... ; sleep 2; done
# pane B: the GPU utilization
$ watch -n1 "curl -s $DCGM_POD_IP:9400/metrics \
    | grep DCGM_FI_DEV_GPU_UTIL"
```

Not locked to one framework

Different model, different server, different API, same node. NVIDIA Triton serves a DenseNet image classifier, and the client runs in-cluster as a Job.



Your workload | Kubernetes | Hardware

LIVE DEMO · 09 (OPTIONAL)

Triton inference on Flatcar

WATCH FOR

- Server and client from a single apply
- The model fetched by an init container
- A correct prediction: **258, Samoyed**

demos/09-triton-inference-on-flatcar

```
$ kubectl apply -f triton-demo.yaml
$ kubectl get pods -w
$ kubectl logs job/triton-classify
# Predicted class index: 258 (Samoyed)
```

Try it yourself

github.com/flatcar/flatcar-demos

Local, QEMU, free

- 01 Flatcar on QEMU
- 02 Ignition first boot
- 03 System extensions
- 04 Kubernetes

AWS, NVIDIA T4

- 05 GPU node
- 06 Driver upgrade in place

GPU workloads

- 07 vLLM inference
- 08 DCGM metrics
- 09 Triton inference

Thank you!

flatcar.org

github.com/flatcar

discord.gg/PMYjFUsJyq

Join our Discord!

> **Office hours** · every 2nd Thu, 16:30 CEST **Dev sync** · every 4th Thu, 16:30 CEST

Everyone welcome. Users, contributors, or just curious. Ask questions, get help, share what you're building!