

KC Platform

Building an industrial Linux platform with Isar

Jani Hautakangas

Konecranes | Isar Community Meetup

© 2026 Konecranes. All rights reserved. 'Konecranes', 'Moves what matters' and  are either registered trademarks or trademarks of Konecranes Global Corporation.

KONECRANES



Konecranes in one slide

Finnish company, roots in 1910, independent since 1994. We make things that **lift heavy things**, and the service business that keeps them running.

- **Industrial cranes:** overhead cranes in factories, steel mills, waste-to-energy plants
- **Port equipment:** ship-to-shore cranes, straddle carriers, terminal automation
- **Service:** maintenance, modernisations and remote monitoring

Roughly **16,000 people in 50 countries**, sales around **EUR 4 billion**.

More and more of that equipment is **software-defined**: automation, anti-collision, remote diagnostics. That software runs on industrial PCs on and around the crane, and those PCs run the **KC Platform**.

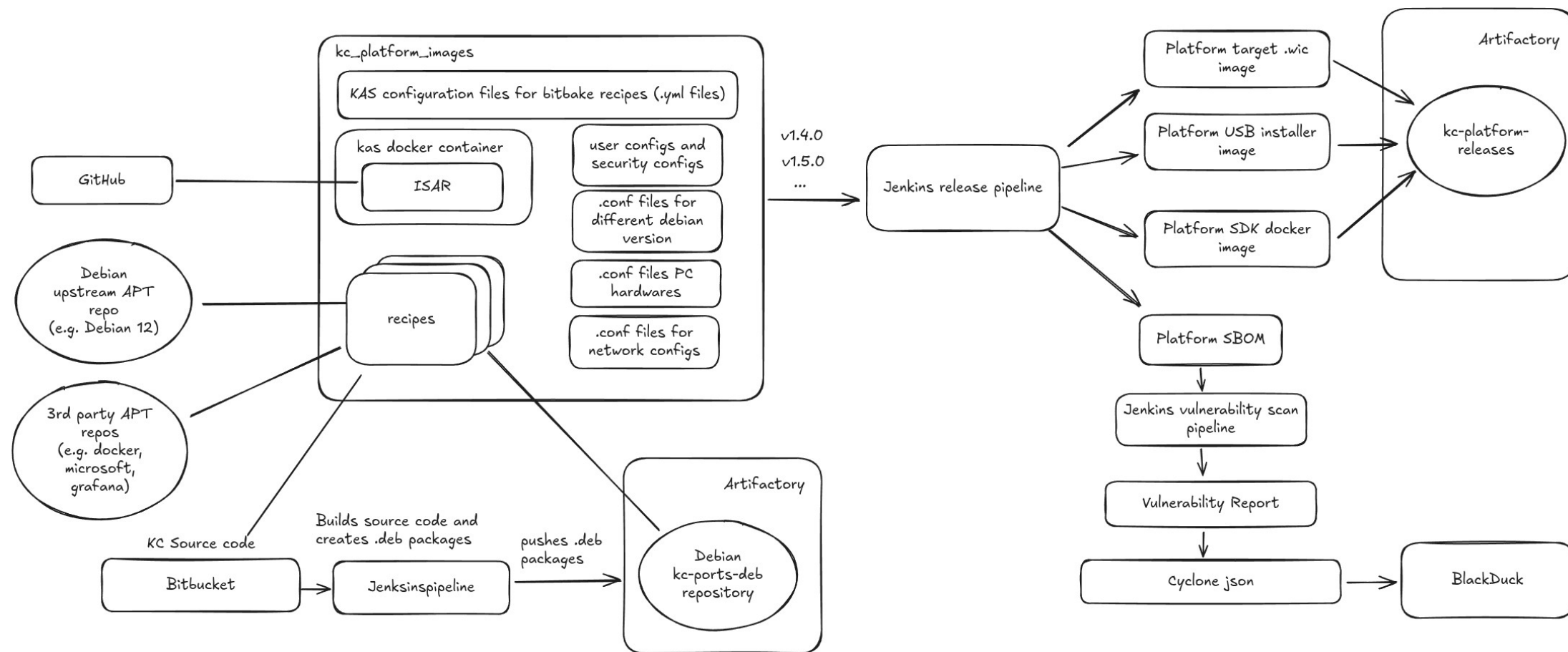
What is the KC Platform?

A **Debian-based Linux platform** that Konecranes industrial applications run on. One git repository, built with **lsar + kas**.

- **13 product images** from one tree, each a different application stack on the same base
- **Real targets:** Siemens SIMATIC IPC and Advantech industrial PCs; one unified amd64 lsar configuration, hardware detected at runtime from DMI data
- **Virtual targets:** QEMU, VirtualBox, VMware; the same image developers boot locally, so there is no separate developer build
- **Debian 10 through 13** live side by side in the same tree; products stay in the field for decades, so old bases do not simply go away
- **Shipped as:** a bootable .wic image plus a USB installer image

One build description, many products. The differences are **configuration**.

From source to release



Why Isar

We needed Yocto's build model with **Debian's distribution**.

- **Debian does the distro work:** a security team, an LTS story, and 20+ years of packages we do not have to maintain ourselves. Hard to argue with for equipment that lives 20 years.
- **Binary packages, not a from-scratch build:** no cross-compiling the world. A full image is minutes, not hours, and a laptop is a perfectly good build machine.
- **BitBake recipes and layers:** the model our embedded engineers already knew from Yocto, so nothing new to learn.
- **.deb is already our interface:** application teams build their own .deb packages in their own pipelines and push them to Artifactory. Isar just installs them.
- **We only build what is genuinely ours:** out-of-tree kernel modules, configuration packages, a handful of KC tools. Everything else is upstream binary.

The build is **reproducible and pinned**: Isar by tag and commit, apt sources by snapshot, our packages by version.

Composing the build: kas + Isar

Every build is a **combination of small YAML fragments**, one per independent axis.

- **kas/platform/*.yml**: 13 products, each setting hostname, preferred versions and which apps to preinstall
- **kas/board/*.yml**: amd64, qemu-amd64, qemu-amd64-sb, virtualbox, vmware
- **kas/opt/*.yml**: Debian release, secure boot, TPM/LUKS, SDK, runtime, test
- **Kconfig menu**: `./kas-container menu` gives developers a menuconfig-style picker; CI passes the exact same YAML list on the command line

On the Isar side: **one image recipe per product** in `recipes-core/images/`, and a **customizations-<product> package** built with `dpkg - raw` for everything device-specific, such as network units, journald limits, nginx config, systemd presets and udev rules.

Nothing is copied into the rootfs by hand. If it is on the device, **it came from a .deb**, so it can be listed, upgraded and audited.

Four Debian releases, one tree

A product does not move to a new Debian because the platform did. Each release is a **self-contained axis**.

- **One fragment per release:** `kas/opt/debian10..13.yml` sets a single key, `distro:`; everything else hangs off `conf/distro/debian<NN>.conf`
- **Sources travel with the release:** `DISTRO_APT_SOURCES` points at one `.list` per release, naming the matching suite for every repo in it
- **Eight third-party repos:** `docker`, `grafana`, `postgresql`, `timescaledb`, `nginx`, `zabbix`, `nodesource`, `llvm`. Every key is fetched by URL and pinned by `sha256sum` in `THIRD_PARTY_APT_KEYS`
- **Pinning, per vendor:** a preferences file in `DISTRO_APT_PREFERENCES` for each one, so Debian's own package never quietly wins the version race
- **Our packages:** one Artifactory repo with a suite per release. Build-time only: the source and the apt auth file are stripped from the rootfs before the image is sealed

Debian 10 and Debian 13 build from the same recipes on the same day. Moving a product forward is a **one-fragment change** in the build; the real work is in the application.

One build run, everything we ship

A tagged commit goes into Jenkins and comes out as a complete release.

- **Runtime image** (.wic.img): GPT + EFI, built with WIC, expands the root partition on first boot
- **USB installer image**: its own Isar-built initramfs that flashes the runtime image onto the internal disk of the IPC
- **SDK image**: Isar SDK, matched to the exact release, so application teams build against what actually ships
- **SBOM**: syft runs in R00TFS_POSTPROCESS_COMMAND, output feeds a Gype scanning pipeline and a CycloneDX report
- **Release notes**: generated from the changelog in the same post-process step

All of it from the same recipe set the developer runs on their laptop.

Testing the image, not the recipe

The build is only half of it. The image has to be **proven to boot and behave**.

- **BATS** (Bash Automated Testing System) test suites: shared ones plus a set per product
- **LAVA** (Linaro Automation and Validation Architecture) orchestrates the runs, booting the real release image under **QEMU**
- Results emitted as **JUnit XML**, visible directly in Jenkins
- Runs on **every pull request**, and failing tests block the merge
- The **same test tarball** runs locally and in CI, so a failure is reproducible on a laptop

Coverage today: services start and stop as configured, users and password policy behave, network comes up, web UIs answer, secure boot chain verifies.

Next: the same suite under LAVA against real Siemens and Advantech hardware.

Recent: the jump to Isar v1.1

We moved from **Isar v0.11 to v1.1** (and kas-container 5.2) in one step. It went better than we expected; the interesting part is what we carry on top.

Isar is pinned by **tag and commit** in `kas/kc-ports.yml`; kas applies our patch series from `isar-patches/` at checkout. We carry **10 patches**.

Patches get **dropped as upstream grows the feature**; that is the goal. **Which of ours are worth upstreaming?**

Security: verified boot and encrypted root

The most recent block of work, and the part where Isar carried most of the weight.

- **Secure Boot:** UEFI to shim to GRUB to kernel, verified by firmware. Enabled by adding one kas fragment, `kas/opt/secure-boot.yml`, on top of any product image.
- **Signed out-of-tree modules:** our SIMATIC IPC modules are signed during the package build with Isar's `pkg.sign` build profile and `SIGNATURE_KEYFILE/SIGNATURE_CERTFILE`. The MOK is enrolled at install time.
- **Kernel lockdown** on, signatures checked at every module load.
- **Snakeoil keys** for developer builds via `PREFERRED_PROVIDER_sb-mok-keys`; production keys are injected by Jenkins and never enter git.
- **Testable without hardware:** a dedicated `qemu-amd64-sb` board config runs the full verified chain under QEMU in CI.
- **TPM 2.0 sealed root encryption:** the root ships LUKS2-encrypted from the `wic rootfs-luks` plugin; first boot swaps the build passphrase for a key sealed to **PCR7**. Shipping today on the Debian 13 platforms.

Alongside it, on the OS itself: **PAM-backed login** in front of every web UI on the device, and a **password quality policy** enforced platform-wide with `pam_passwdqc`.

What we are working on next

- **dm-verity** for read-only rootfs integrity.
- **Signed in-place updates** on all platforms, with A/B partitions.
- **LAVA on real hardware**: the same BATS suite on Siemens and Advantech IPCs, which is also what secure boot and TPM need in order to be properly validated.

Driving all of it, two regulatory deadlines:

- **IEC 62443-4-2 SL-2**: boot integrity, rootfs integrity, signed updates
- **EU Cyber Resilience Act**: reporting of exploited vulnerabilities from **September 2026**, full applicability **December 2027**. SBOM generation and vulnerability handling are already in the pipeline.

Most of this is not novel, but it is a lot easier to argue for when the base is Debian and the build already tells you exactly what is on the device.

Thank you.

13 product images, 5 boards and 4 Debian releases from one lsar tree

Debian binaries mean we maintain configuration, not a distribution

Secure boot, signed modules and SBOMs came mostly from the tools

Questions?

