Red Hat Summit

Connect

Automation at the edge

An Ansible approach

Red Hat

Red Hat

Hendrik van Niekerk

Senior Solution Architect - Edge team

Red Hat

What is the edge?

Images: unsplash.com, Moises Rivera

Introducing Red Hat Device Edge

Red Hat
Device Edge

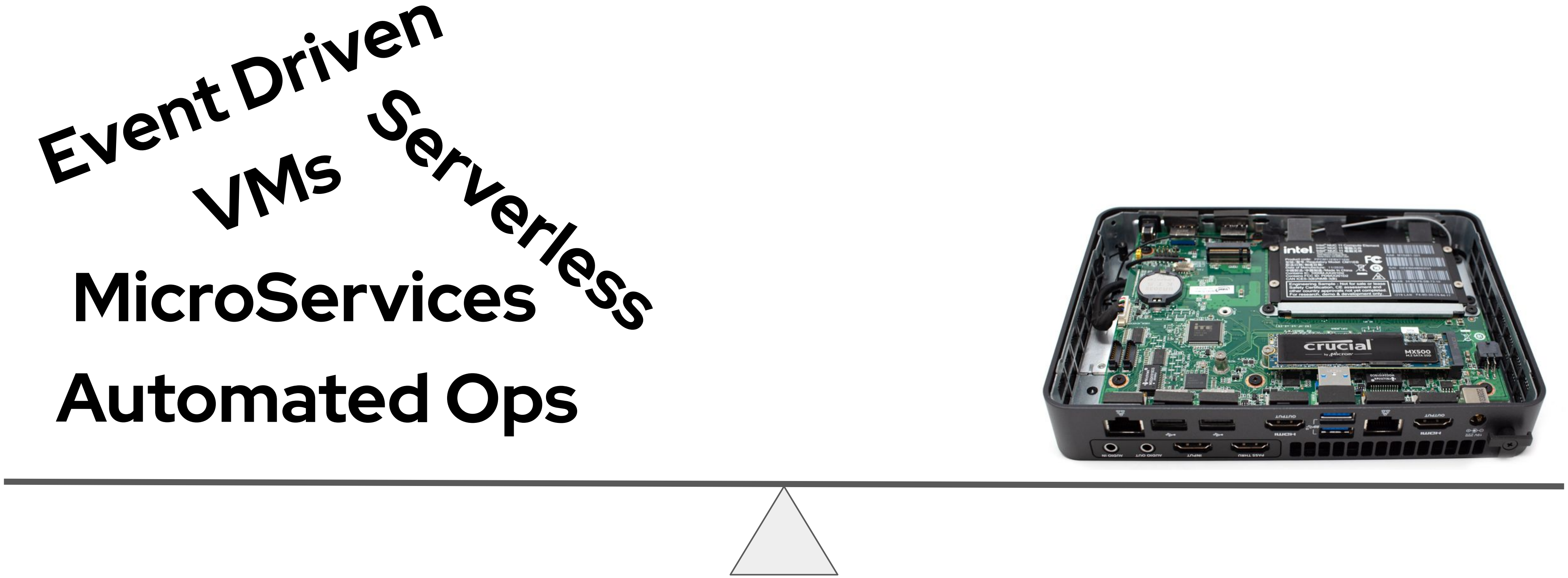
Combines Kubernetes * + Red Hat Enterprise Linux

Address the needs of small devices at the farthest edge

** Kubernetes is optional, you can use just OStree or RPM RHEL with Device Edge if you don’t need a Kubernetes API*

The feature-rich vs small-footprint trade off

The right balance between functionality and hardware footprint



Event Driven

VMs

Serverless

MicroServices

Automated Ops

Image: Intel NUC 11 Compute Element CM11EBv716W (intel.com)

RH Device Edge is just RHEL

(...delivered in slightly “different way")

What is Red Hat Device Edge? (explained with a metaphor)

Red Hat
Enterprise Linux

Image by parblusa (pixabay.com)

What is Red Hat Device Edge? (explained with a metaphor)

Red Hat
Enterprise Linux

Your longtime best friend

Image by parblusa (pixabay.com)

What is Red Hat Device Edge? (explained with a metaphor)

Red Hat OpenShift

Image from Wikimedia Commons

What is Red Hat Device Edge? (explained with a metaphor)

Red Hat OpenShift

Agile, powerful, feature rich...

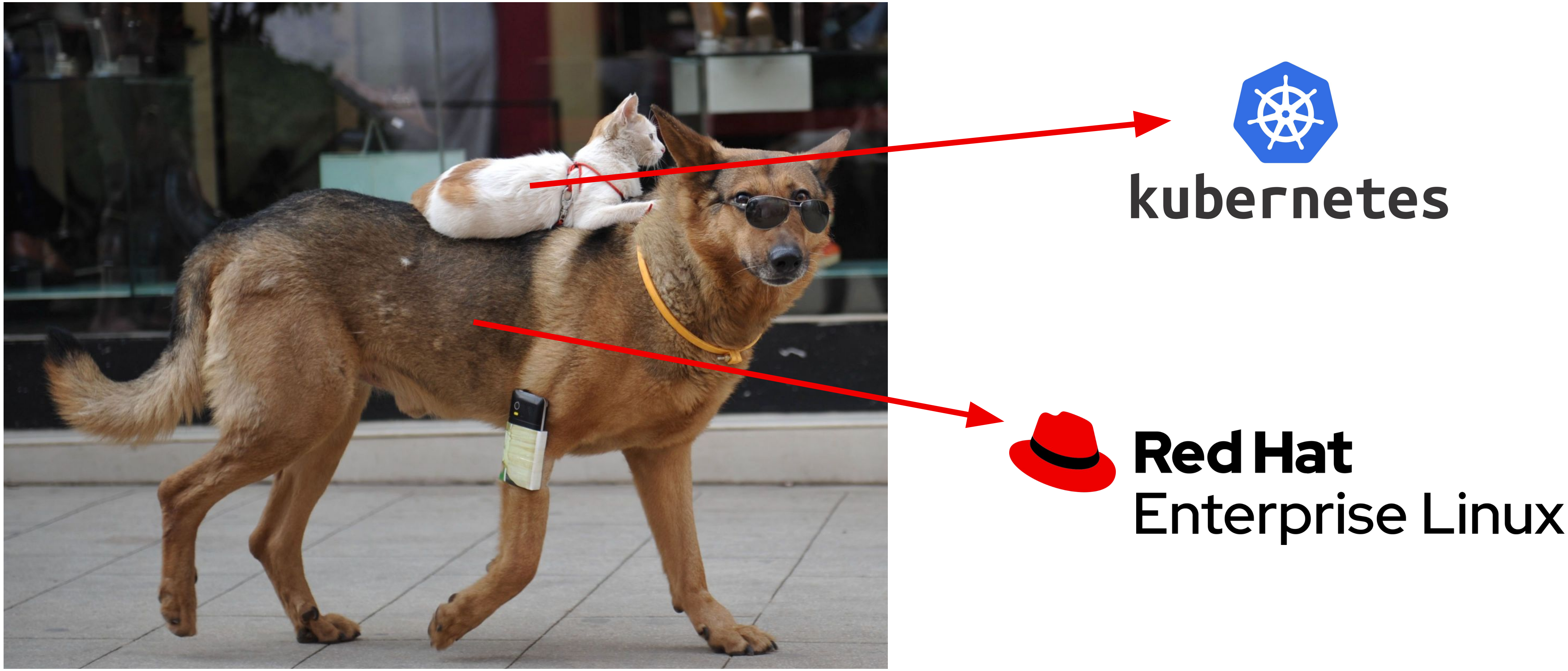
Image from Wikimedia Commons

What is Red Hat Device Edge? (explained with a metaphor)

Red Hat
Device Edge

Image by HAP/Quirky China News

What is Red Hat Device Edge? (explained with a metaphor)



kubernetes

Red Hat Enterprise Linux

Image by HAP/Quirky China News

What is Red Hat Device Edge? (explained with a metaphor)

kubernetes

With a little bit of OpenShift

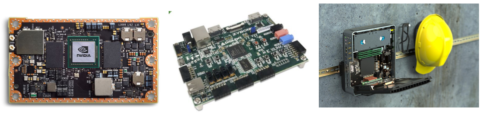
Red Hat Enterprise Linux

“for Edge”

Image by HAP/Quirky China News

When using Red Hat Device Edge?

What is a “field-deployed device”?



Field-deployed device

- **single board** computer, system on chip, etc.
- limited to few, **resource-limited HW** configs
- **not out-of-band manageable**, i.e. not remotely recoverable
- **mass-*imaged* centrally, “plug&walk” provisioning** (via FIDO Device Onboard)
- **no option to boot via USB/ISO, PXE**
- **no physical access control**
- uplink **network may be disconnected**, rarely available, firewalled/NATed, slow, costly, …

Server in controlled environment

- server-standard board
- extensible (CPU, RAM, accelerators, NICs,…)
- out-of-band manageable (via BMC and mgmt. network), i.e. remotely recoverable
- *installed* on site via installation medium
- option to boot via USB/ISO, PXE
- physical access controls in place
- uplink network is mostly available, high bandwidth, low latency, cheap, …

What are some of the differences?

System layout

Users

Booting and updates

Package manager

System layout:

Introducing OSTree

Who manage the OS updates/deployments?

OSTree is a **transactional file system manager** for Linux-based operating systems

System layout **change summary**

`/usr` read-only

`/var` is shared between deployments, `/etc` is individual (copied) and `/usr` is part of the deployment

R/W symbolic links to `/var`
chroot directory from `/sysroot`

What’s the **benefit** of these changes?

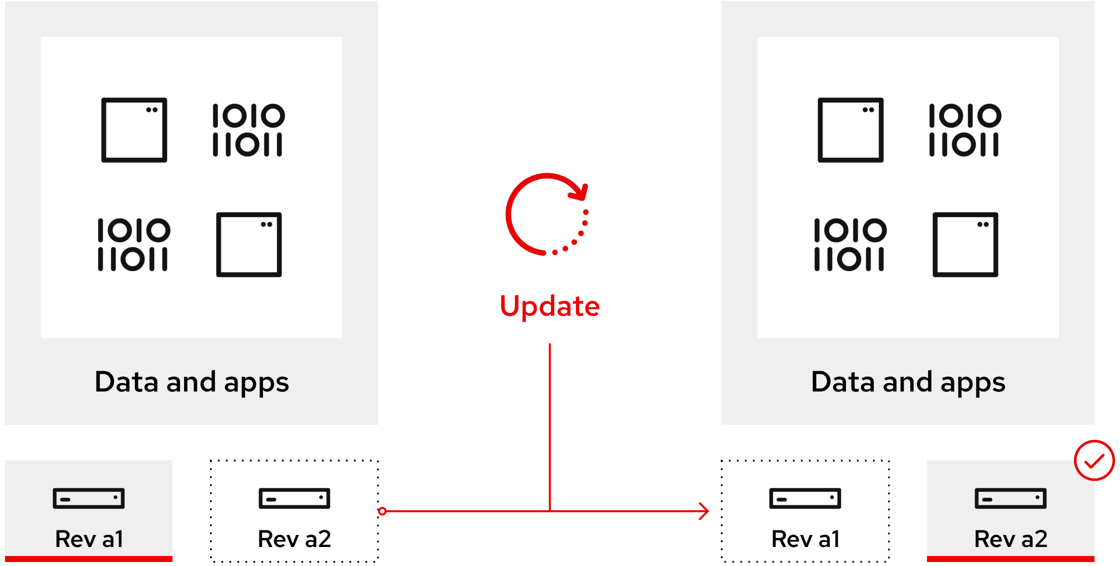
Better system **consistency** across multiple devices

Easier **reproducibility**

Better **isolation** between pre and post change system state

What about updates, and making sure they don’t break the system?

Let’s start with the update process



1. **Pull** in a new file system

- Upgrade knows nothing about packages or Apps
- It replaces the complete file system

2. **Store** the new file system

- Stores many filesystem and checks one out to be the root
- Keeps track of what’s been checked in

3. **Deploy** the new file system

- Checks out one of the file systems stored to be the root
- Checks out by creating hard links

OSTree and APPs lifecycle demo steps: https://github.com/luisarizmendi/edge-demos/blob/main/demos/upgrade-and-rollback/README.md

Things to bear in mind about **OSTree**

Update only the differences

- Limit network Bandwidth usage
- Reduce install/offline time
- No file duplicates on deployments

First copy, then update:

- Resiliency on updates (ie. power)
- Stored in RAM before merged
  (need enough RAM for large files) **(!)**
- Filesystem corruption = no update **(!)**
  (It's rare and dual partition solves it)
- Each deployment has its own `/etc`

OSTree VS VM templating

- Not best friends
- OSTree for baremetal systems or VM in-place updates

Operating System automatic updates

OSTree can automate upgrades

Three different modes

“*Check*”: Auto. show available updates
“*Stage*”: Auto. download updates
“*None*”: disabled

Reboot is not automatic

(by default…you could change it)

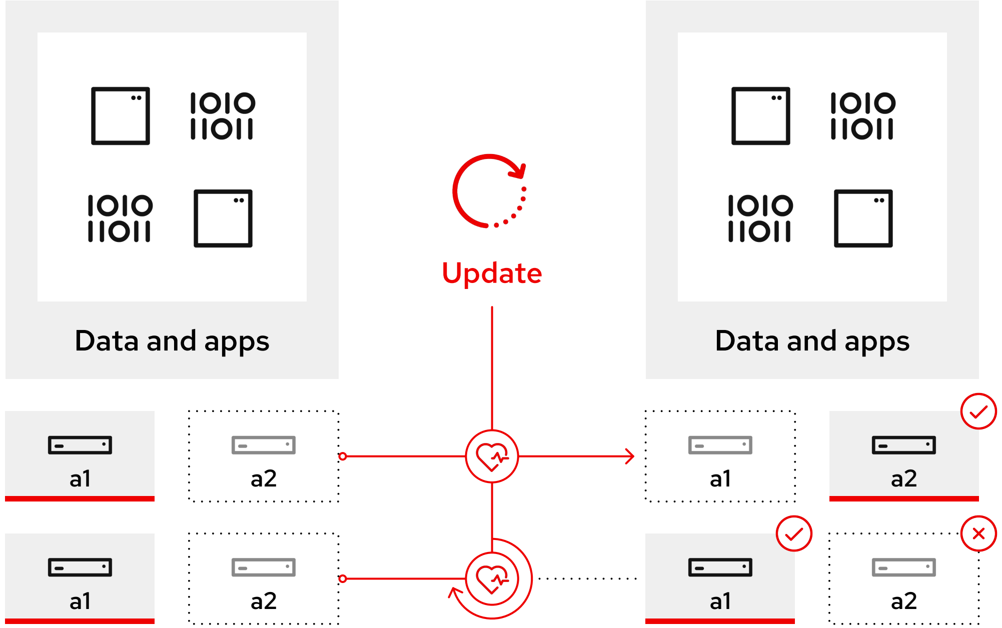
Daily checks for updates

Can be customized
Update can be automatic on new versions

Someone broke the latest version, what now?

Intelligent rollbacks: Greenboot

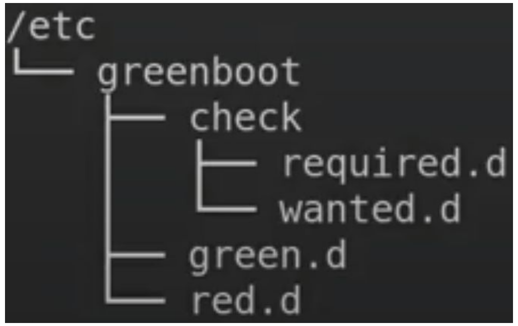
Additional safeguard for application and OS compatibility



Custom healthchecks to determine if nodes are working properly

- Healthchecks are run during the **boot process.**
- If checks fail, a counter will track the number of attempts.
- In a failure state, the node will **use rpm-ostree to rollback the update**.

Greenboot directory structure



```
/etc
└── greenboot
    ├── check
    │   ├── required.d
    │   └── wanted.d
    ├── green.d
    └── red.d
```

- `/etc/greenboot/check/required.d`

  Health checks* that **must not fail**

- `/etc/greenboot/check/wanted.d`

  Health checks* that **may fail**

- `/etc/greenboot/green.d`

  Scripts to be run **after successful boot**

- `/etc/greenboot/red.d`

  Scripts to be run **after failed boot** (3 attempts to boot in case of failure, 3 times it will be executed)

* Health checks can be done with Systemd services instead of Shell scripts

Boot and updates **change summary**

Deployment chroot bind at boot time

GIT principles: updates differences only, possible multiple branches

Updates can be rollback after completion or cancelled at any time

What’s the **benefit** of these changes?

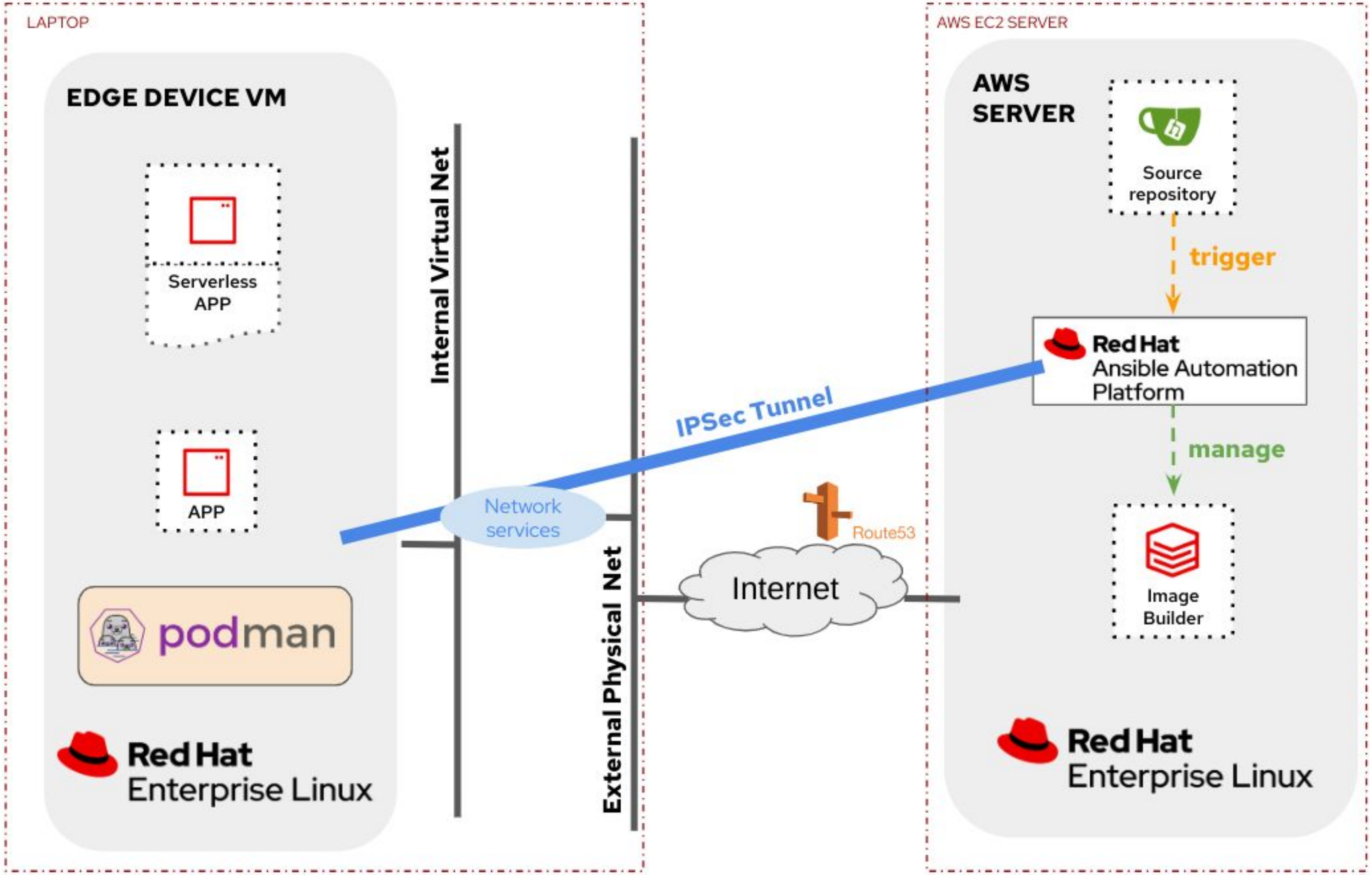
Git-like based system updates **improve tracking and recovery** times

Updates minimize bandwidth consumption

Automatic self-healing capability that minimize system failures on updates

How could this look in reality?

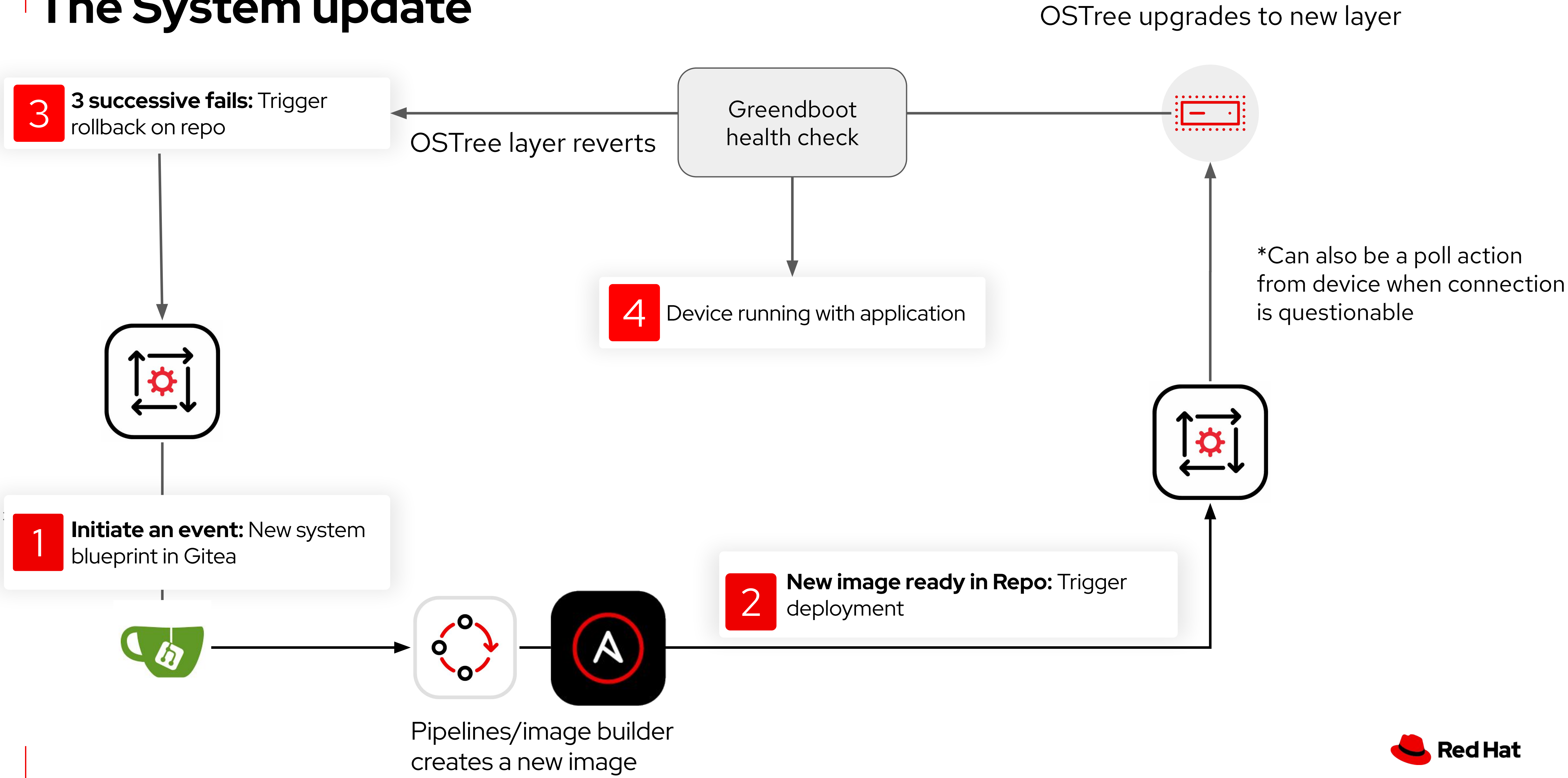
A practical example



Architecture of RHDE-AAP lab - https://github.com/luisarizmendi/rhde-aap-gitops-demo?tab=readme-ov-file

Let’s start by looking at the build

The System update



OSTree upgrades to new layer

3 **3 successive fails:** Trigger rollback on repo

OSTree layer reverts

Greendboot health check

4 Device running with application

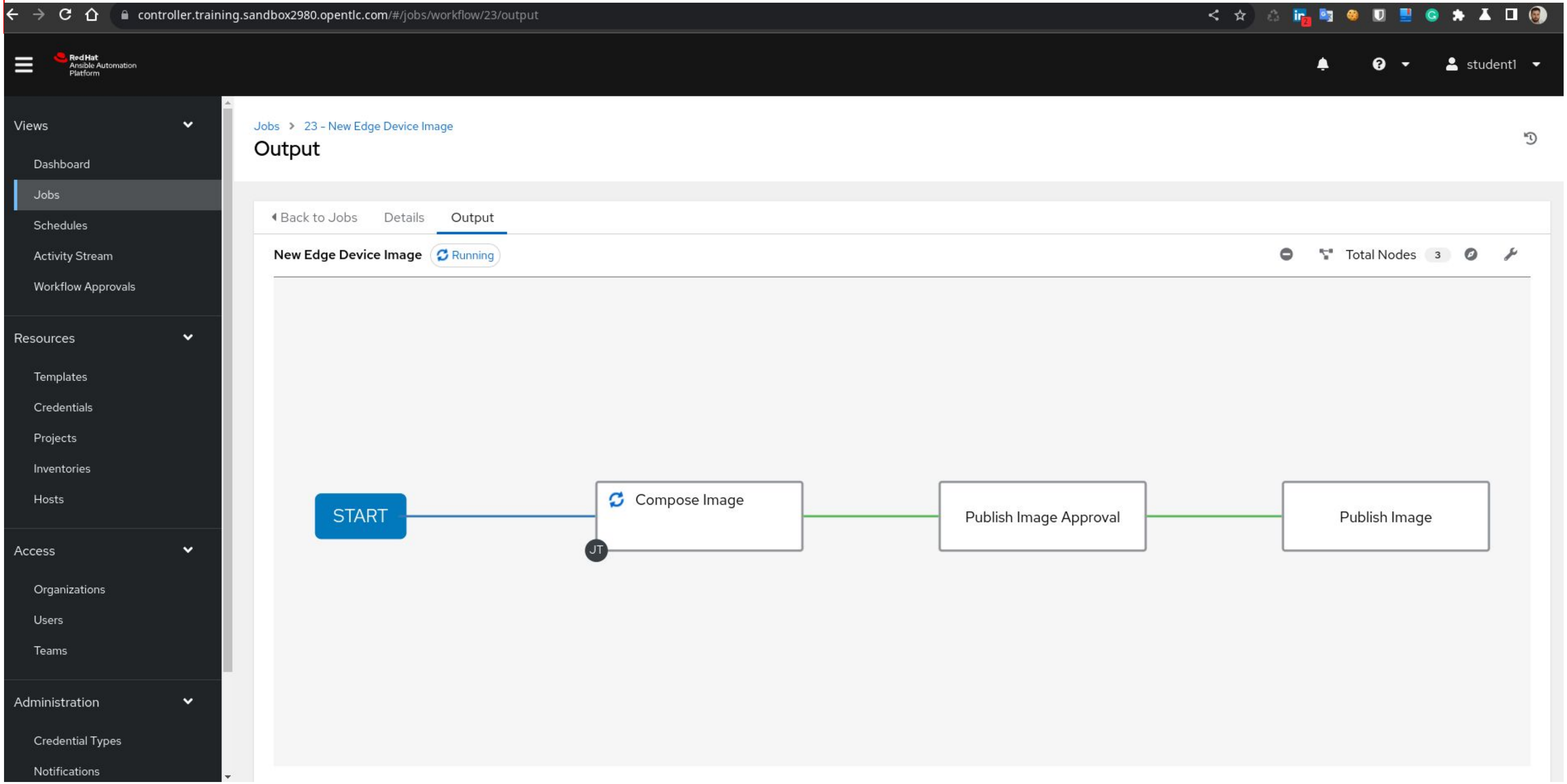
*Can also be a poll action from device when connection is questionable

1 **Initiate an event:** New system blueprint in Gitea

2 **New image ready in Repo:** Trigger deployment

Pipelines/image builder creates a new image

The build job



controller.training.sandbox2980.opentlc.com/#/jobs/workflow/23/output

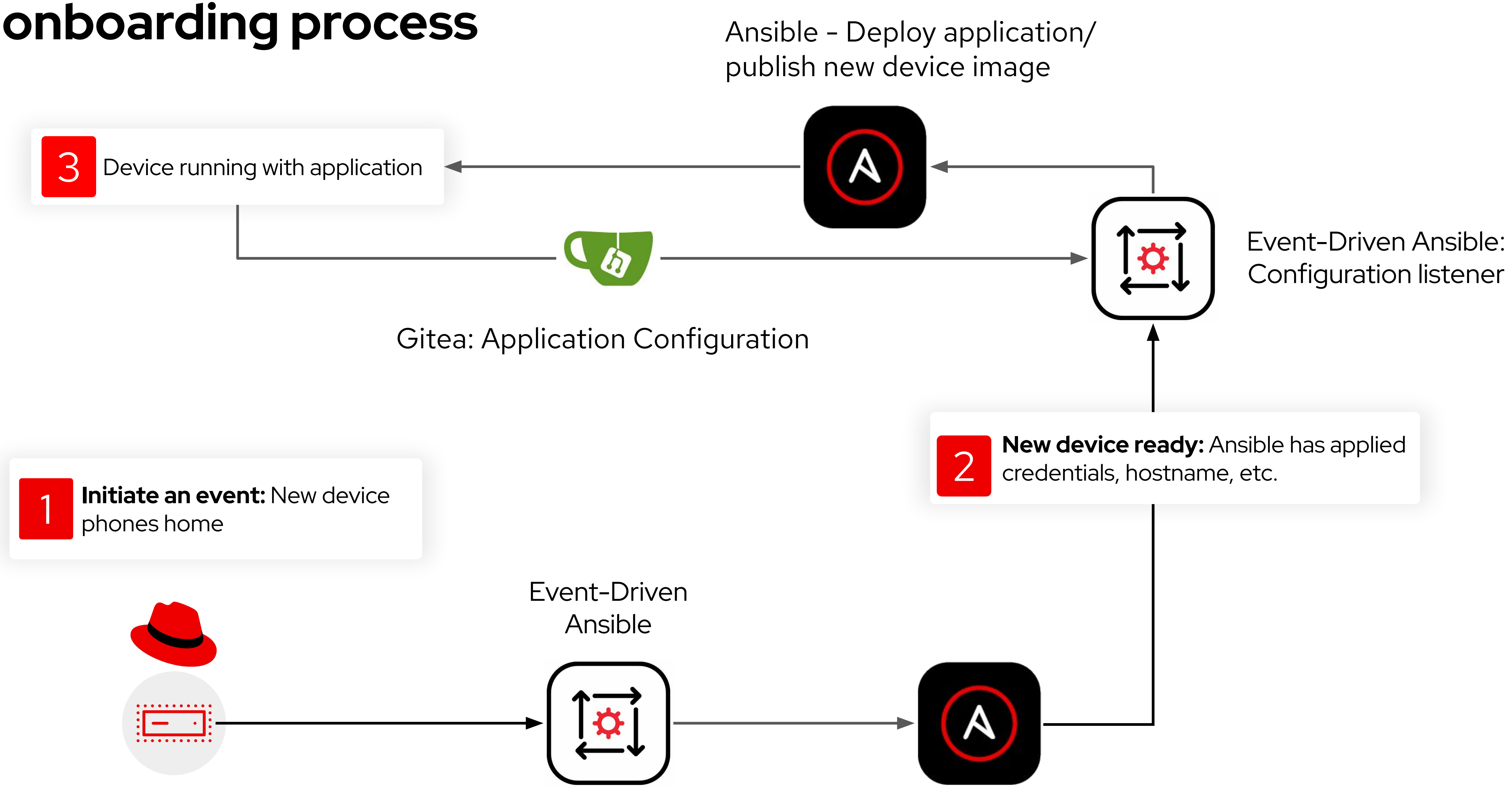
Jobs > 23 - New Edge Device Image

Output

New Edge Device Image · Running

Total Nodes 3

The onboarding process



Ansible - Deploy application/ publish new device image

3 Device running with application

Event-Driven Ansible: Configuration listener

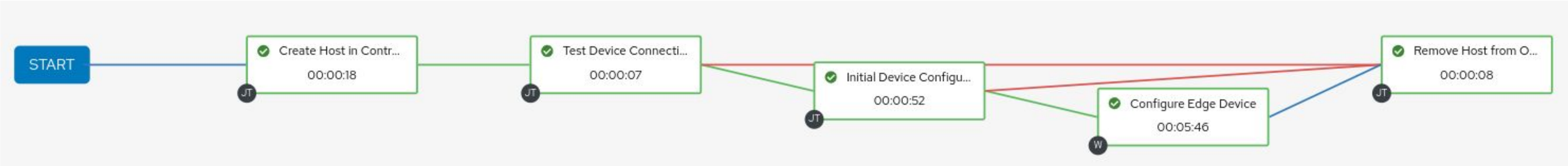
Gitea: Application Configuration

2 **New device ready:** Ansible has applied credentials, hostname, etc.

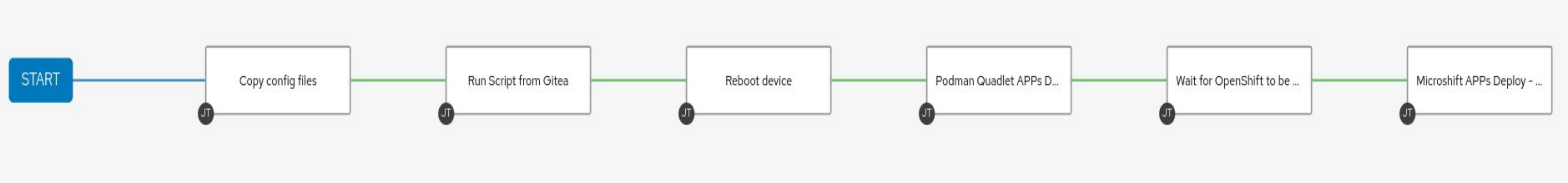
1 **Initiate an event:** New device phones home

Event-Driven Ansible

Device Onboarding



Device configuration

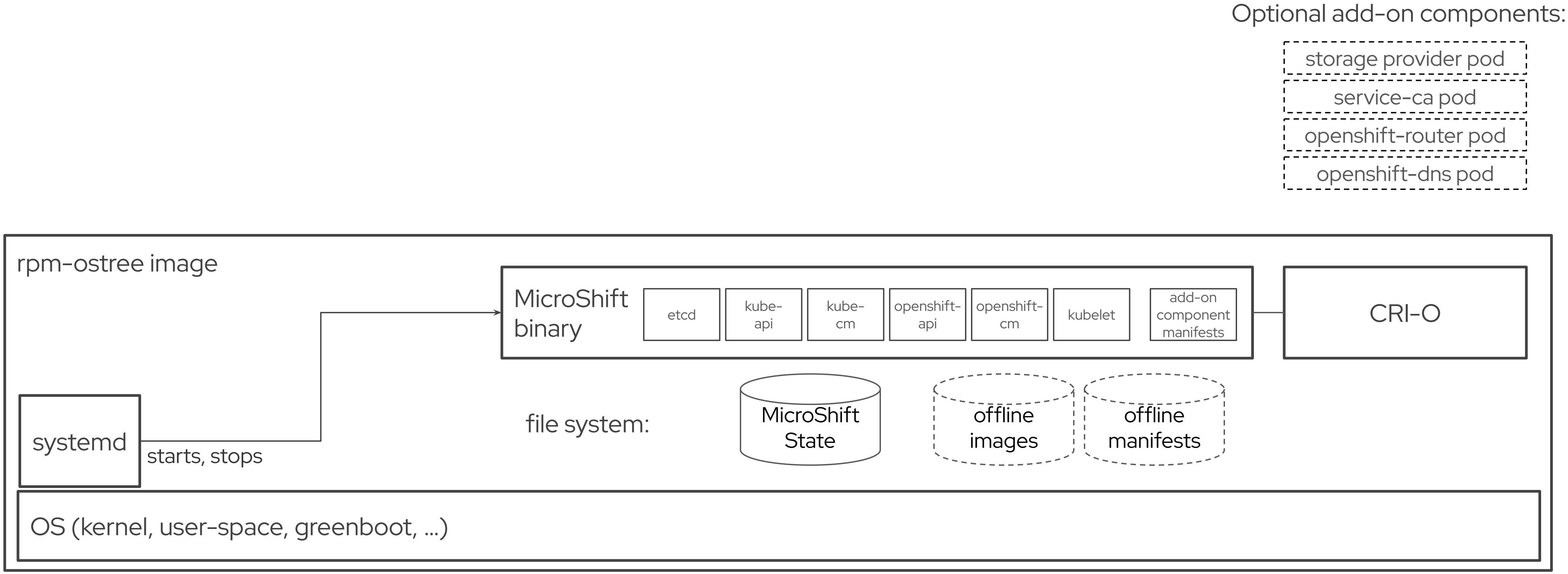


MicroShift

MicroShift bits is an optional component of Red Hat Device Edge

Enabling Kubernetes workloads

MicroShift architecture (RPM-based, embedded in rpm-ostree)



Enabled APIs

kubernetes

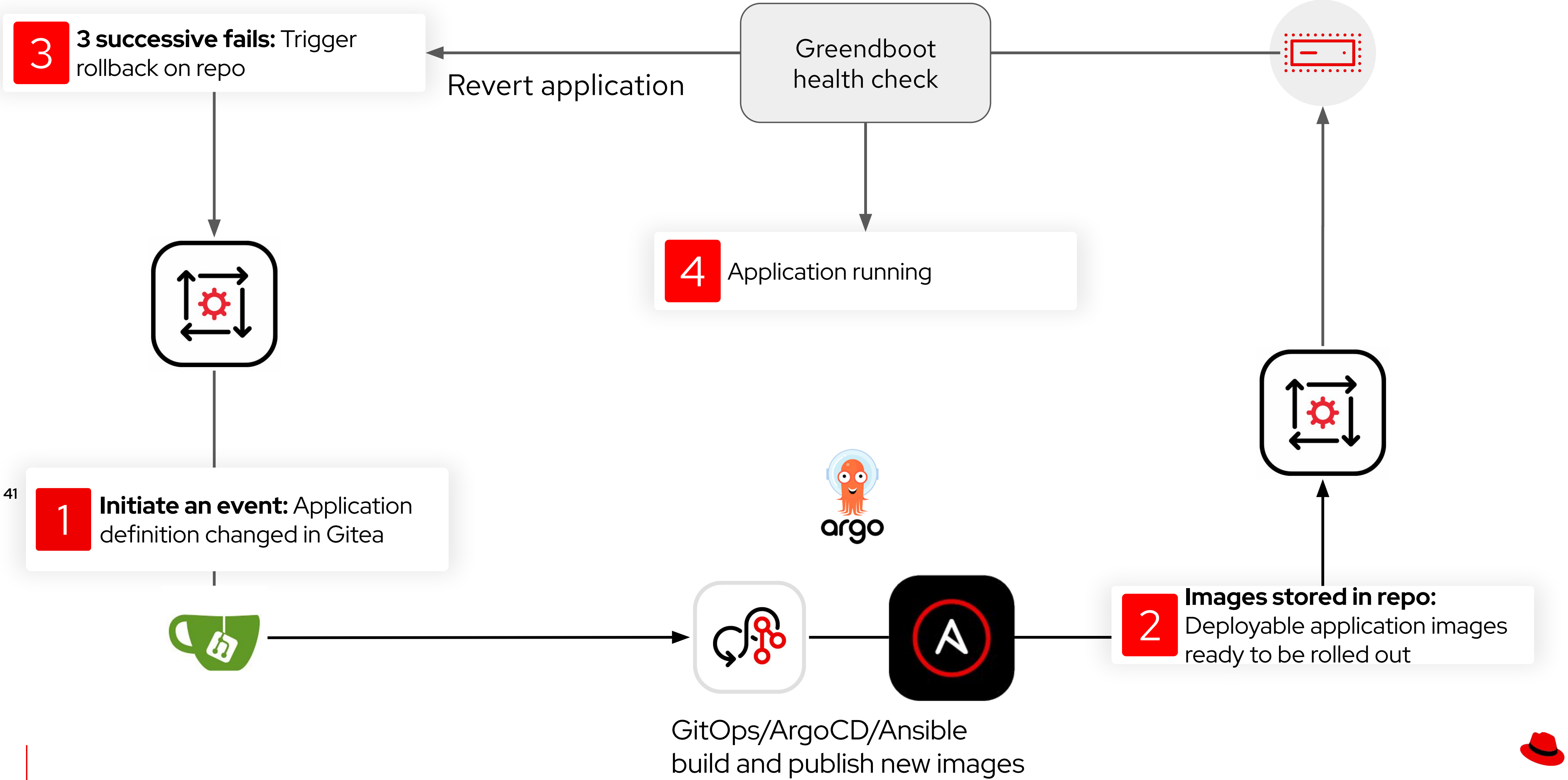
Standard kubernetes APIs

OPENSHIFT

route.openshift.io/v1

security.openshift.io/v1

The Application update



3 **3 successive fails:** Trigger rollback on repo

Revert application

Greendboot health check

Microshift/podman new deployment

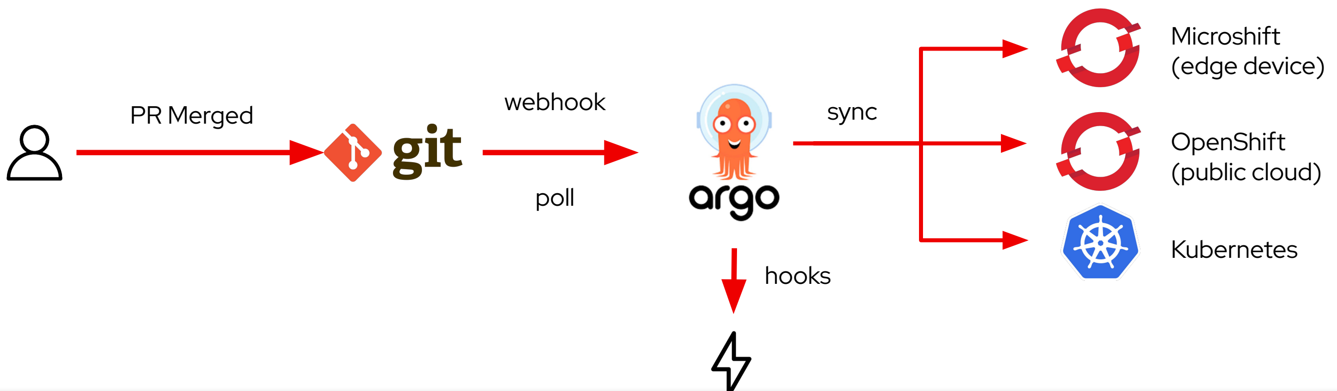
4 Application running

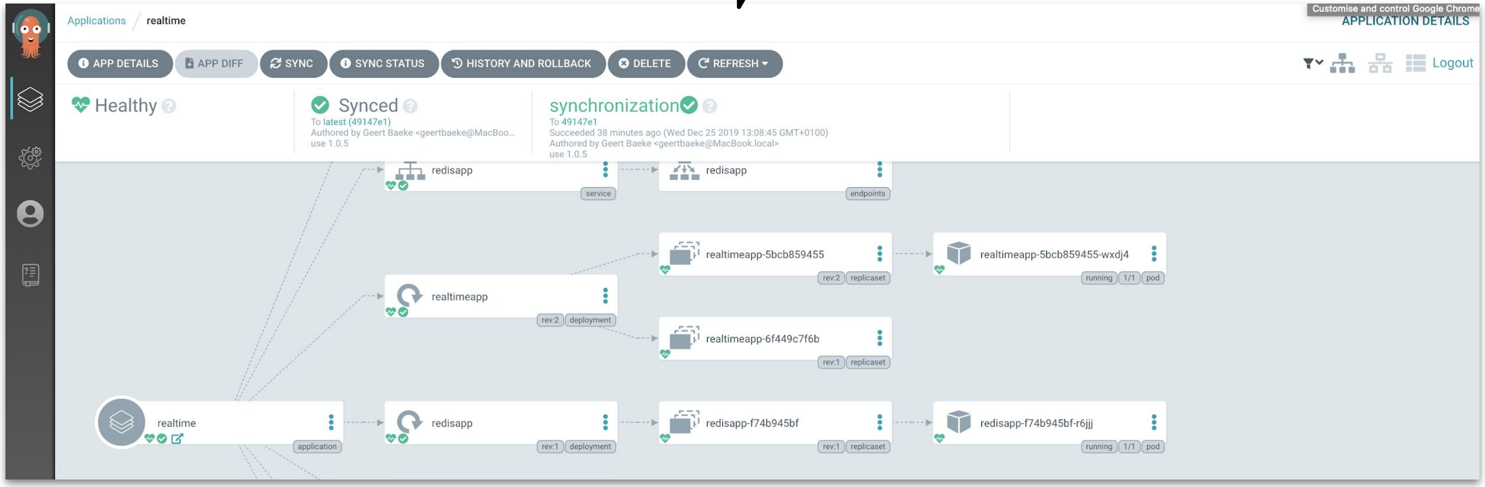
1 **Initiate an event:** Application definition changed in Gitea

2 **Images stored in repo:** Deployable application images ready to be rolled out

GitOps/ArgoCD/Ansible build and publish new images

Argo CD for declarative GitOps continuous delivery





Synced
To latest (49147e1)
Authored by Geert Baeke <geertbaeke@MacBoo...
use 1.0.5

synchronization
To 49147e1
Succeeded 38 minutes ago (Wed Dec 25 2019 13:08:45 GMT+0100)
Authored by Geert Baeke <geertbaeke@MacBook.local>
use 1.0.5

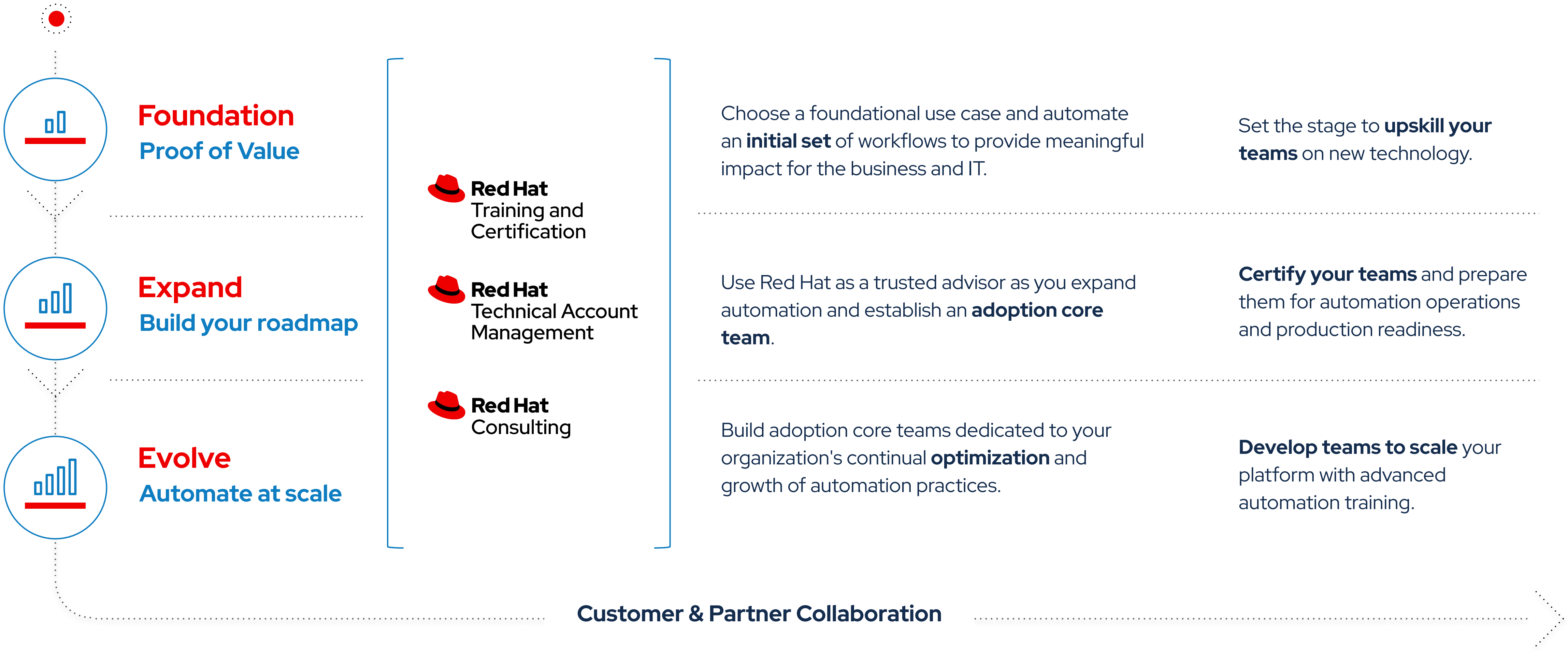
- Configurations versioned in Git
- Automatically syncs configuration from Git
- Drift detection, visualization and correction
- Granular control over sync order
- Rollback and rollforward to any Git commit
- Manifest templating support (Helm, Kustomize, etc)
- Visual insight into sync status

References:

- Try this setup yourself: https://github.com/luisarizmendi/rhde-aap-gitops-demo

Your automation journey with Red Hat Services

A customized approach that meets your needs where you are



Foundation
Proof of Value

Choose a foundational use case and automate an **initial set** of workflows to provide meaningful impact for the business and IT.

Set the stage to **upskill your teams** on new technology.

Expand
Build your roadmap

Use Red Hat as a trusted advisor as you expand automation and establish an **adoption core team**.

Certify your teams and prepare them for automation operations and production readiness.

Evolve
Automate at scale

Build adoption core teams dedicated to your organization's continual **optimization** and growth of automation practices.

Develop teams to scale your platform with advanced automation training.

Red Hat Training and Certification

Red Hat Technical Account Management

Red Hat Consulting

Customer & Partner Collaboration

Red Hat
Summit

Connect

Thank you

in linkedin.com/company/red-hat

f facebook.com/redhatinc

youtube.com/user/RedHatVideos

twitter.com/RedHat

Red Hat

Edge hardware vendors as partners

ASRock®
Industrial

ONLOGIC

ADVANTECH

SUPERMICRO®

Simply NUC®

WELOTEC®

And many more on:
https://catalog.redhat.com/

Red Hat

Red Hat