Edge Device Management

Both the Red Hat and Non-Red Hat Kinds

Josh Swanson
Edge Dream Team
jswanson@redhat.com

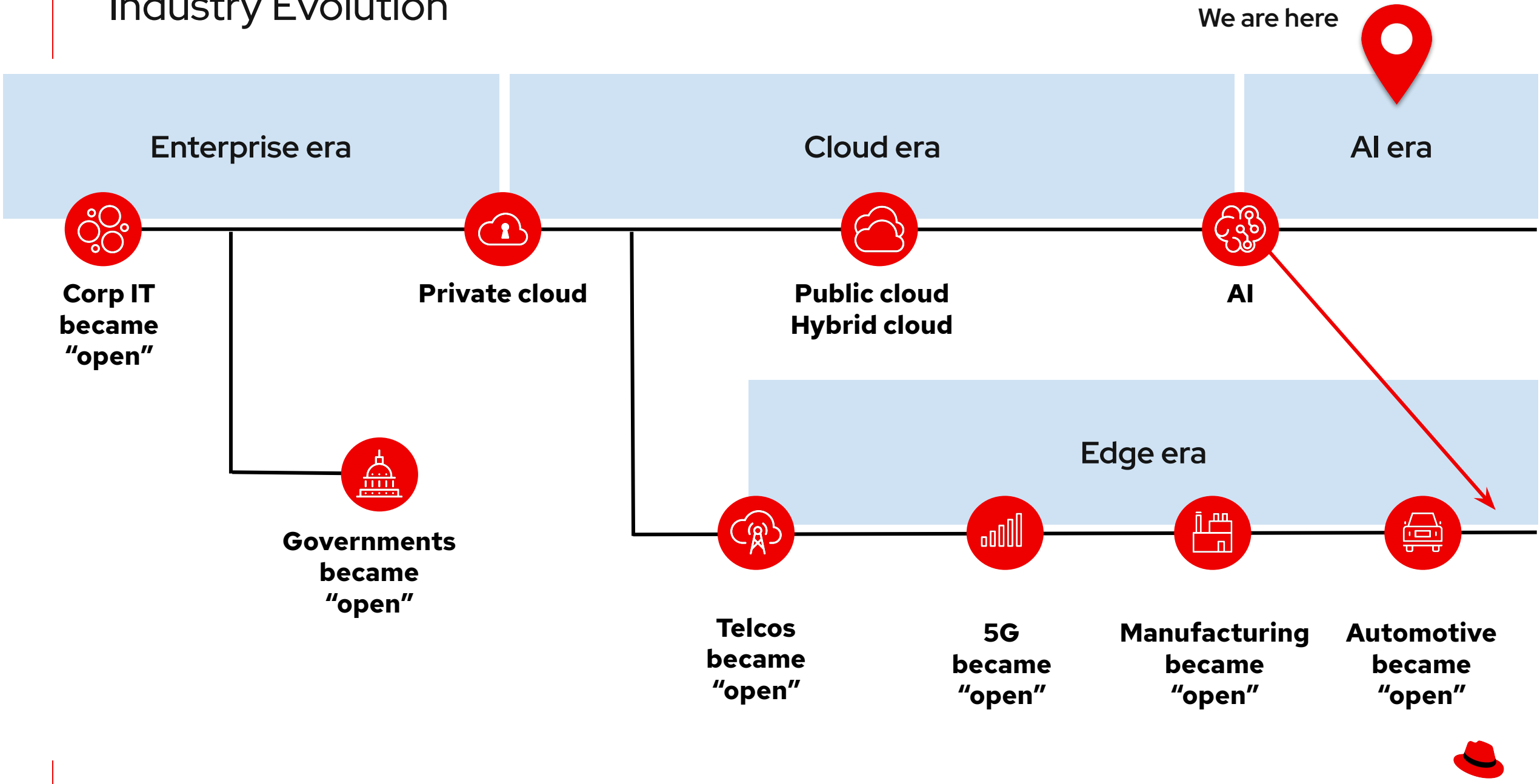
Red Hat

What we'll discuss today



- The Red Hat Edge Device Story
  - Image Mode
  - Edge Manager
- The Non-Red Hat Edge Device Story
  - Intro to Ansible Automation Platform
  - EDA
  - Ansible Mesh (for Edge)
- Managing Red Hat and Non-Red Hat Edge Devices
- Q&A

Industry Evolution

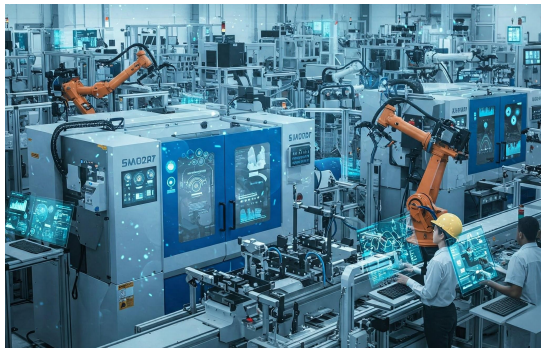


Use Cases



Industrial Edge

Facilitating smarter, connected, and efficient operations across various sectors, including transportation, oil&gas, and smart grid management.



Smart Manufacturing

Enhancing manufacturing efficiency through real-time analytics, predictive maintenance, and IoT-driven quality control, leading to sustainability, such as CO2 footprint optimization.



Retail

Automation and integration of monitoring and control processes, production and supply chain optimization, inventory optimization, and real-time analytics.



Image Mode for RHEL (bootc)

Red Hat

Outcomes

What does image mode fix today?



Less risk

Reduce the risk associated with updates with atomic transactions and rollbacks



Better builds

Improve the composability and repeatability of standard builds through layering



Move faster

Increase the speed of experimentation



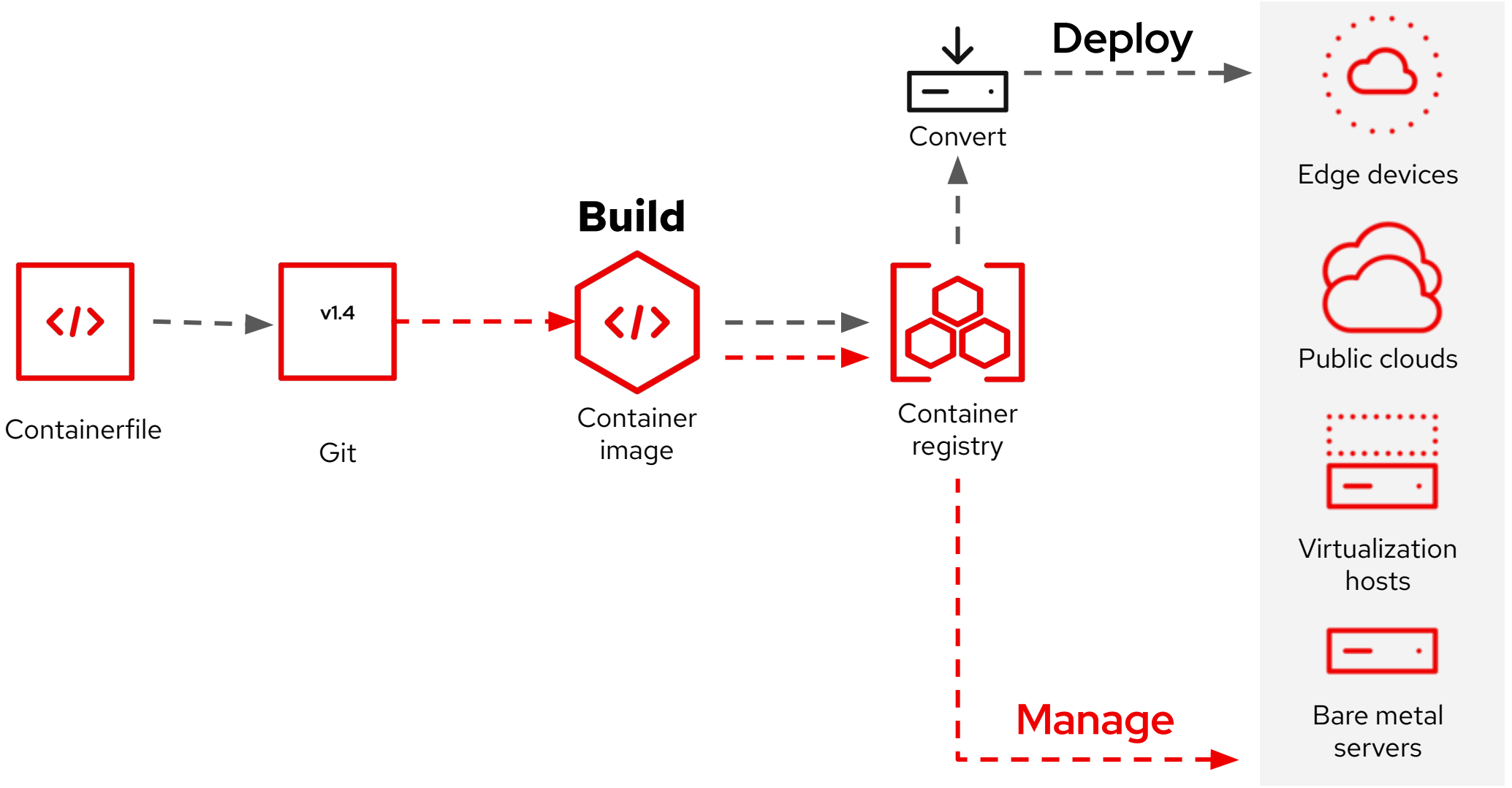
Streamline process

Simplify end to end management with a single process for OS and applications

Red Hat

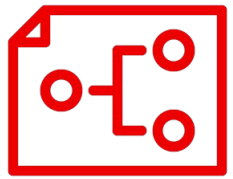
Image mode for Red Hat Enterprise Linux

Simple. Consistent. Anywhere.

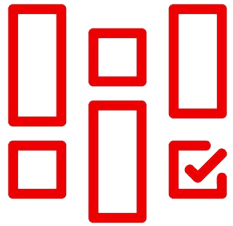


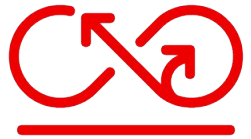
Red Hat

Build









Red Hat

Image mode for RHEL

A container-native workflow for the life cycle of a system

```
FROM rhel9/rhel-bootc:latest

RUN dnf install -y [software]
[dependencies] && dnf clean all

ADD [application]
ADD [configuration files]

RUN [config scripts]
```

Build

A *bootc* base image & container file is all that's needed to describe a system, applications, and dependencies. Use your existing container tools or pipelines to quickly create and test images.

Deploy

Easily convert to a VM/cloud image or deploy on bare metal using RHEL's installer. The container image includes full hardware drivers, but not cloud agents by default.

Manage

Designed for modern GitOps & CI/CD driven environments. Systems will auto-update from the container registry by default. More advanced control and automation is available via custom rollouts (e.g. Ansible). Intelligence via Insights and on-prem content curation via Satellite are planned for the future.

Red Hat

Image mode for RHEL

Encapsulate differences in a sequence of builds

```
# Derive standard operating environment
FROM rhel9/rhel-bootc:latest

RUN dnf install -y [system agents]
[dependencies] && dnf clean all

COPY [unpackaged application]
COPY [configuration files]

RUN [config scripts]
```

```
# Derive database server from SOE
FROM corp-repo/corp-soe:latest

RUN dnf install -y [database]
[dependencies] && dnf clean all

COPY [configuration files]

RUN [config scripts]
```


Red Hat

registry.redhat.io/rhel9/rhel-bootc

The RHEL bootc image is available in technology preview

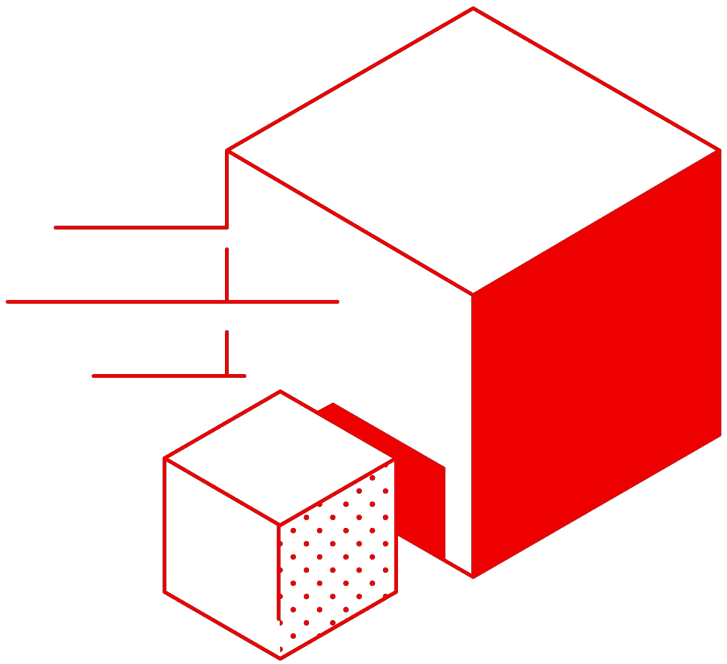


Image Specs:

- 439 rpms
- ~785M compressed
- ~2.2G on disk

Primary contents:

- systemd, kernel, bootc
- rpm-ostree[1]
- linux-firmware
- NetworkManager
- podman
- python
- Misc CLI tools: jq, sos

No cloud-init or virt agents

Documentation

Red Hat

bootc

A/B booting of container images



`bootc upgrade`

Download and stage an updated container image.

- Automatic updates on by default. Configurable using bootc-fetch-apply-updates.timer

`bootc rollback`

Rollback to the previous state. Staged updates are discarded

`bootc switch`

Change to a different reference image

`bootc install`

Install container image **`to-disk`** or **`to-filesystem`**

- Man page
- https://github.com/containers/bootc
- https://github.com/containers/podman-desktop-extension-bootc

Red Hat

Filesystem Layout

Similar to previous ostree setups - but better!!



Build Time
Everything is writable. e.g. /usr, /etc, /opt, ...

Run Time
All image content is read only

/var - RW, instance persistence. Not updated post install, e.g. `podman volume /var`

/etc - RW, 3-way merge like RHEL CoreOS. Machine local state (hostname, static IP)

Flexible partitioning & disk layout available via Anaconda

Defaults aimed at 80%

- Provides a balance of immutable updates w/ persistent config, logs, & container images.
- composefs backend offers better security

Powerful Configuration for the 20%

- Transient / and /etc
- RW possible for other directories via bindmounts and symlinks to /var

Red Hat

Deploy



Install via Kickstart

Deploy container images to bare metal using installation media

```
lang en_US.UTF-8
keyboard us
timezone Etc/UTC --isUtc
text
zerombr
clearpart --all --initlabel
autopart
reboot
user --name=admin-user --groups=wheel
sshkey --username=admin-user “ssh-rsa
AAAAB3Nza……..”

ostreecontainer --url quay.io/myimage:latest
```

NEW

Use existing provisioning workflows

- Red Hat Enterprise Linux boot media (isos)
- PXE & HTTP Boot for network based deployments

Kickstart and Anaconda are used for disk layout and select configurations

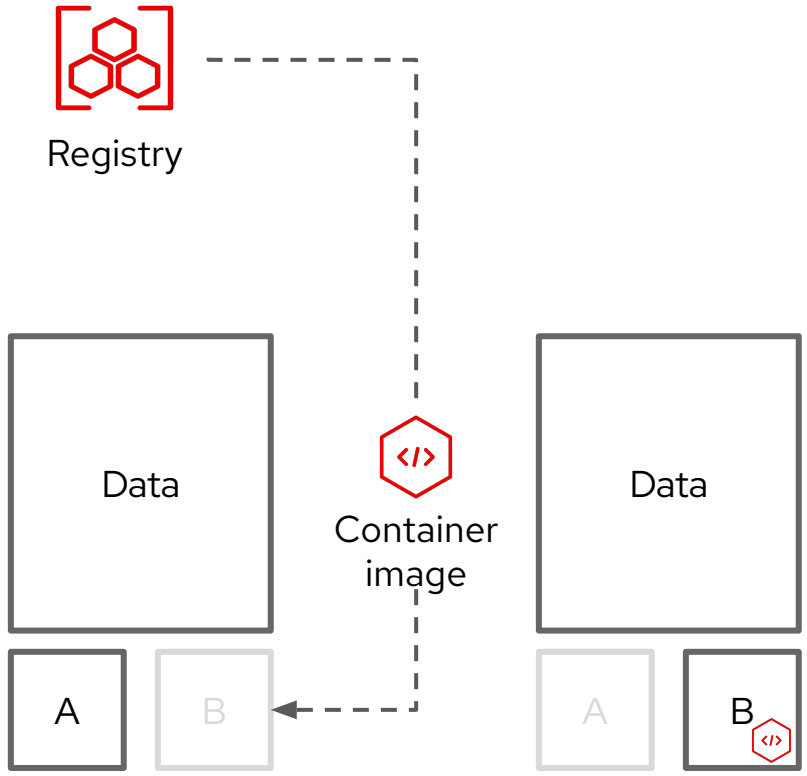
- `%packages` is ignored
- ostreecontainer will fetch the container image from a registry and write it to disk.

`%pre` and `%post` used for configuration

Red Hat

Bootc: Image-based updates perfected

Immutable by default - secure by design



Transactional updates (A → B model)

Bootc uses composefs and ostree to convert the container image into the root filesystem on the host..

Roll forward or backwards

Updates are staged in the background and applied when the system reboots. The transactional model enables rollbacks for additional assurance

Upgrades have never been easier

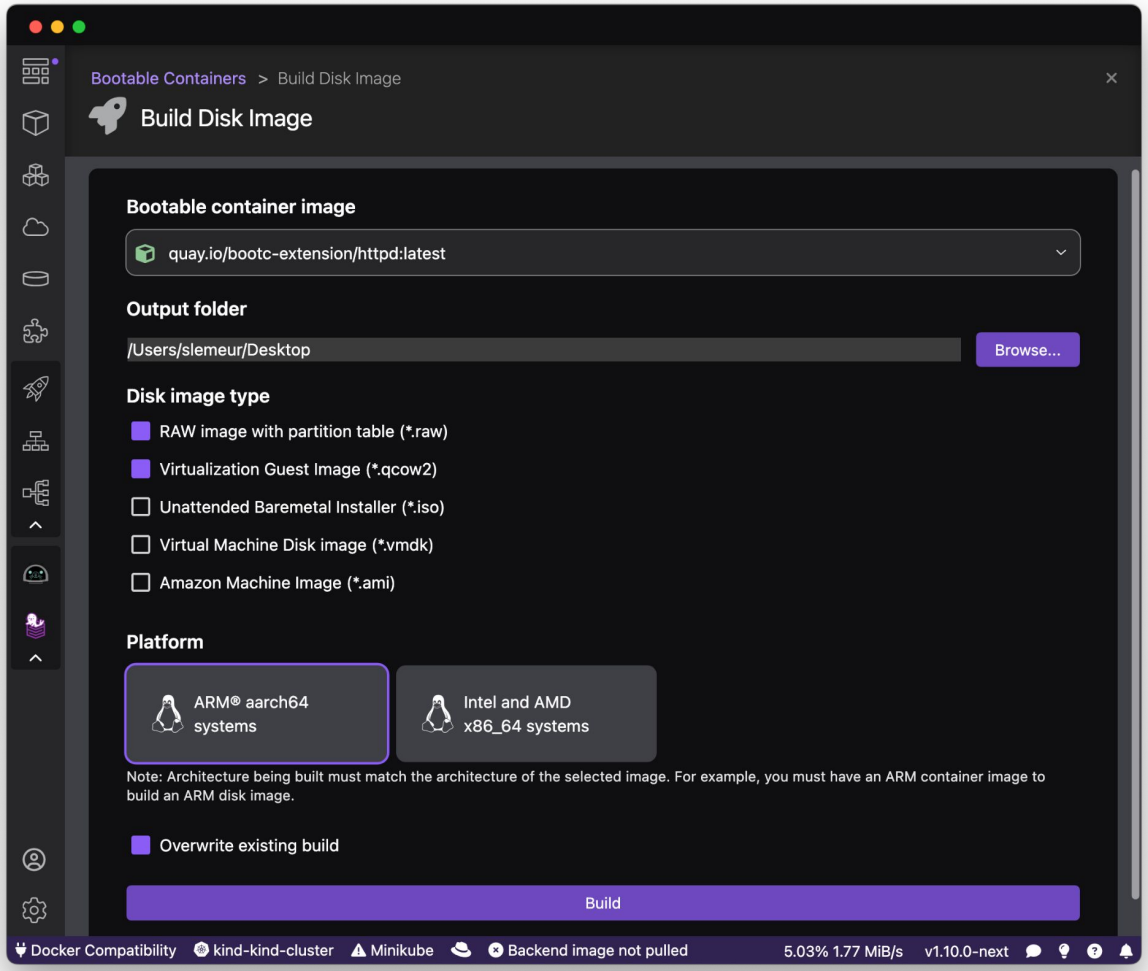
While there are some limits, bootc enables moving between minor releases of RHEL (9.4 → 9.5), as well as major releases (9.4 → 10.0)

Red Hat

Bootc image builder

Create **bootable container images** for bare metal to AWS and everywhere in between

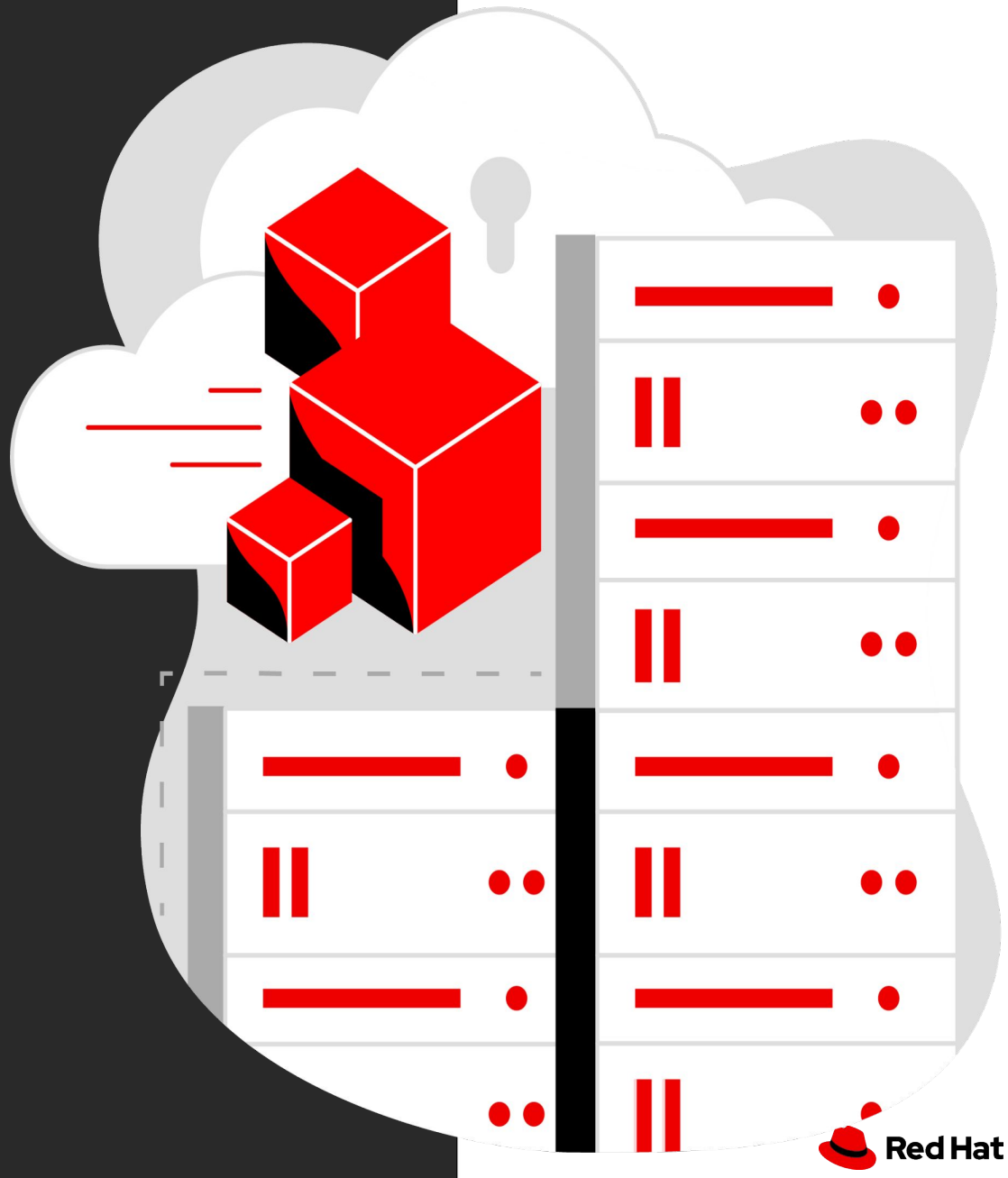
`qcow2`	QEMU Disk Image
`ami`	Amazon Machine Images
`vmdk`	Virtual Machine Disk Image (vSphere, etc.)
`vhd`	Azure / Hyper-V disk image
`gce`	Google Compute Engine image
`raw`	Raw disk image with MBR or GPT partition table
`anaconda-iso`	Bare Metal installer



- Available as a container image:
  - registry.redhat.io/rhel9/bootc-image-builder
- Bootable Container Extension for Podman Desktop
  - Build for x86 & multi-arch builds

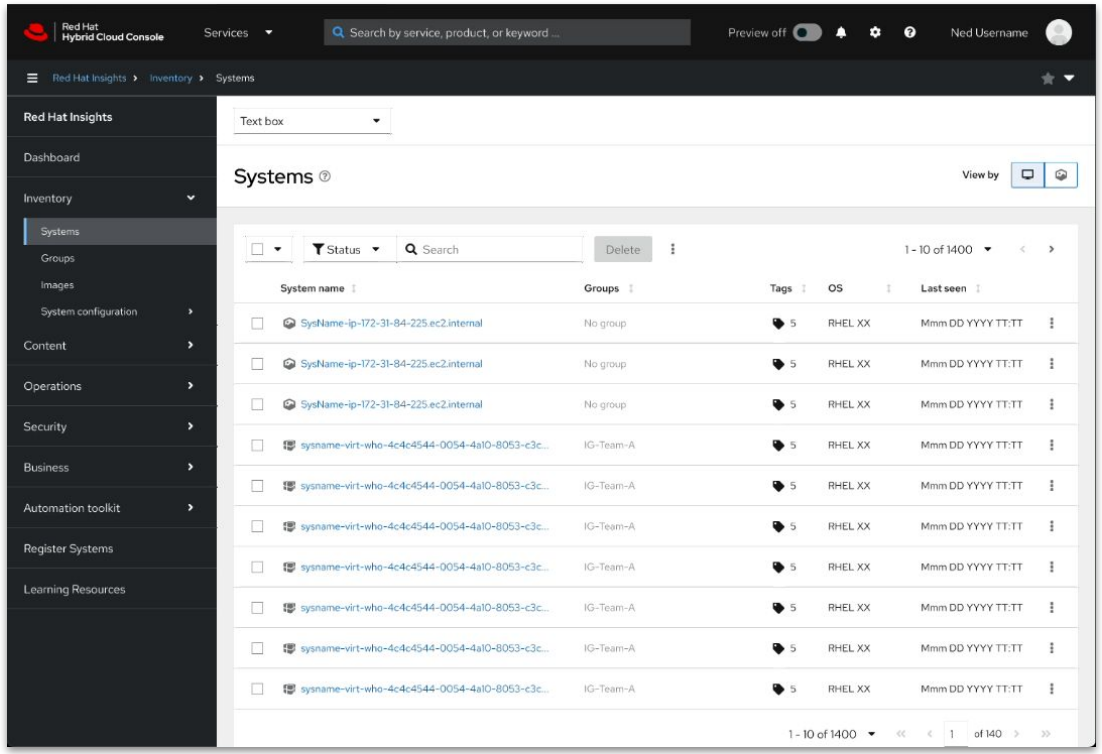
Red Hat

Manage



Management with Red Hat Insights

Visibility and reporting made simple



RHEL is RHEL

Image mode systems appear in inventory like package based systems.

Registration is simple

Activation keys can be baked into images via Containerfile, allowing auto registration to Insights at boot time.

Insights has ... Insights

Image mode systems can be scanned for security and operational recommendations.

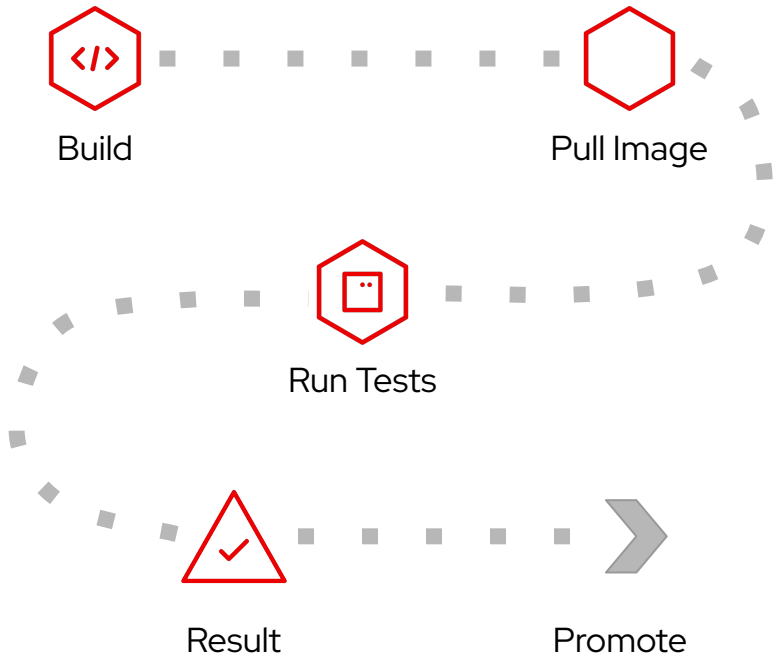
Updates are flexible

Image mode systems can be updated to new versions of images or remediated based on image-specific recommendations.

Red Hat

Validating OS updates has never been easier

CI pipelines used for apps now work with the OS



Test/validate as a container

Bootc images are deployed as bare metal or VMs, but we can also run and test them **as containers**. This enables faster and lighter weight testing/validation of each build's userspace.

Easy pipeline integration

Containers have broad support across Github, Gitlab, Gitea, Circle CI, Jenkins, etc for the common container related tasks and testing. Use any system you like..

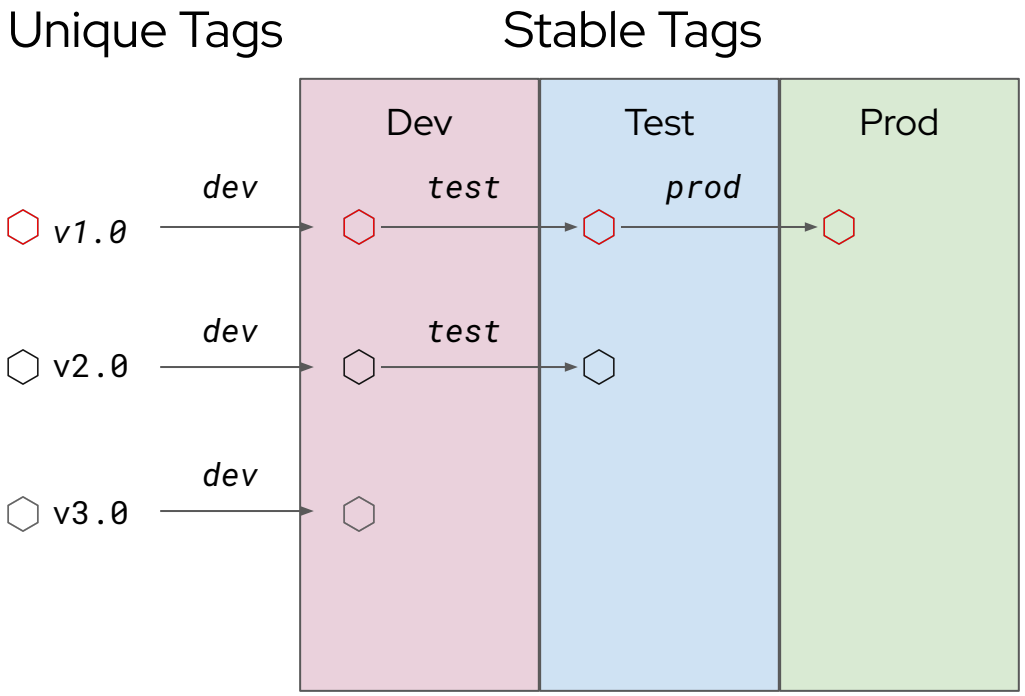
Simple promotion through registry tagging

Tags are a powerful tool to identify dev → test → prod promotions.

Red Hat

OS Updates via Container Registries

Tagging is powerful to version and promote updates



Tags offer simple versioning and visibility

Tags are simple to automate and use for promotions. Bootc will default to updating from a `repository:tag`.

Control updates via tagging

Combine tagging with the optional automatic updates to control fleets of systems via registry tags.

Standardized & scaleable infra

Container registries scale very well and any standard registry can be used.

Red Hat

Use Cases

Where does image mode fit today?



AI/ML Stacks

Perfectly version app dependencies from kernel, GPU & accelerator drivers, frameworks, runtimes, etc



1:1 App/Host

Manage the OS AND app as a single unit



Edge appliances

Easily manage a fleet of systems with registries and auto-updates



Standalone container hosts

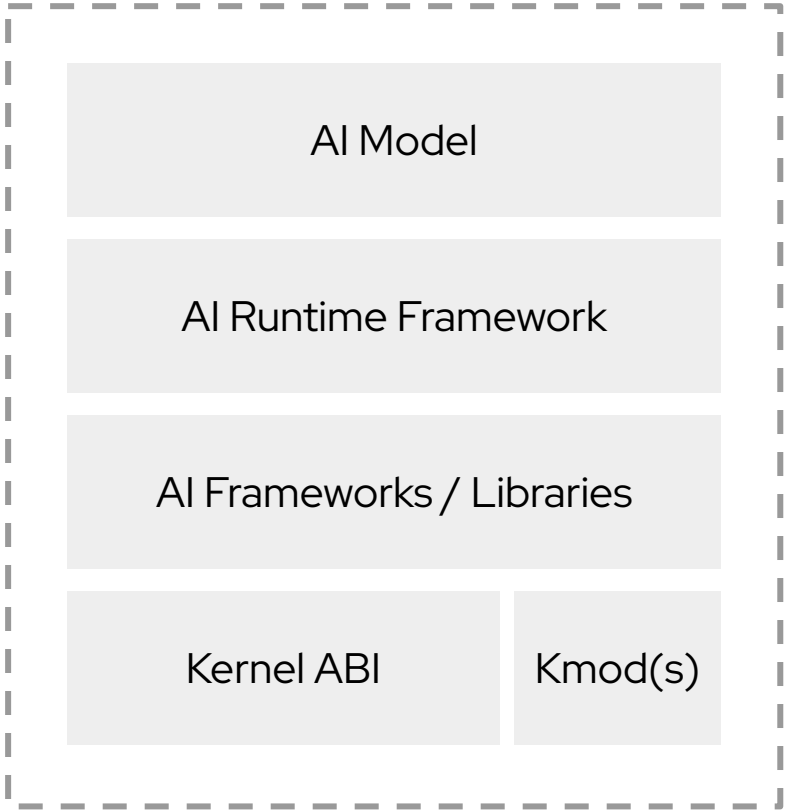
Use common toolchains and pipelines to build containerized applications and the hosting OS

Red Hat

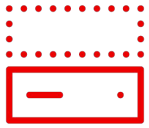
AI/ML Stacks

Deploy AI stacks confidently with image mode

- **Simple**: Much of the AI world already leverages containers, image mode helps deploy AI stacks quickly and efficiently.
- **Portable**: AI workloads often need to run in close proximity to data sources and image mode helps target multiple environments (better way to say that?)
- **Easy experimentation**: Image layering makes it effortless to test different models & frameworks and helps brings order to image sprawl.
- **Limit downtime**: version & test components and dependencies at build time before they go to production.



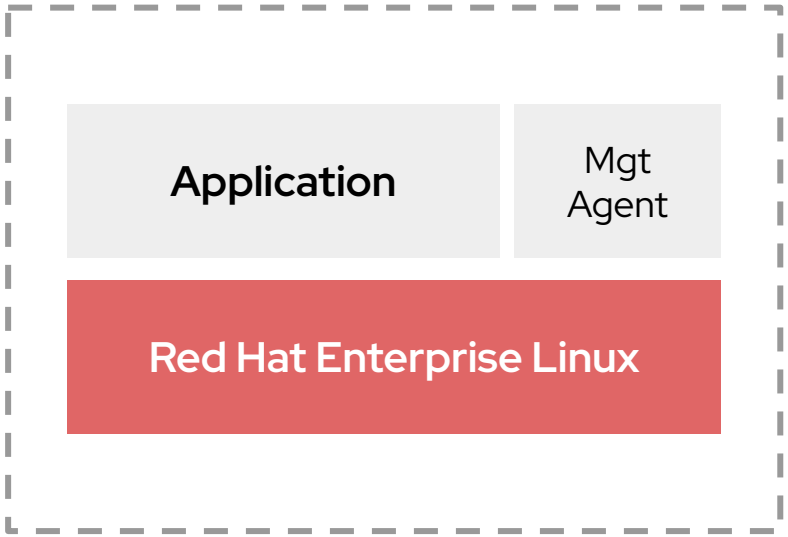
Red Hat



1:1 App/Host

Version & manage OS + app as a single unit

- **Efficient**: reduce the management footprint
- **Repeatable**: containers help enforce consistency and reproducibility making it easy to “scale up”
- **Control drift**: image mode encourages configuration at build time leading to a more consistent fleet.
- **Rollbacks**: A/B boot model means fast recoveries in the case of unforeseen issues.
- Leverage containers for the *uncontainerizable* apps

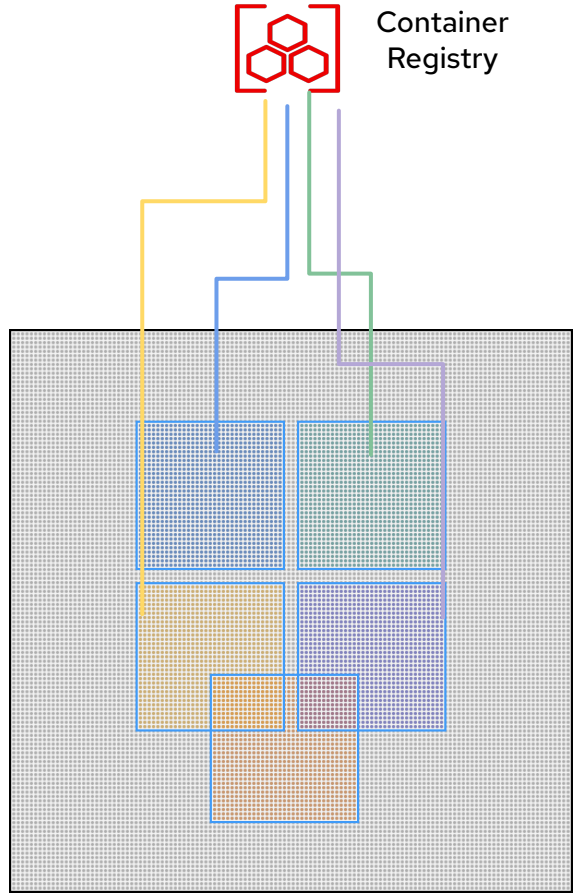


Red Hat

Edge Appliances

Registries and auto-updates make managing a fleet of identical systems a snap

- **Image-based updates**: provide increased reliability over the life of the system.
- **Rollbacks**: A/B boot model means fast recoveries in the case of unforeseen issues.
- **Updates at scale**: Control OS & App container versioning through industry standard registries and tags.
- Support for air-gapped and DIL environments

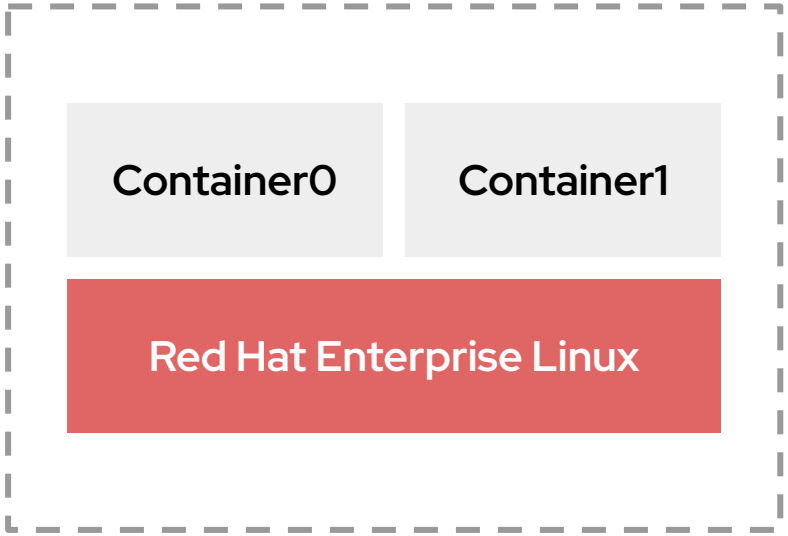


Red Hat

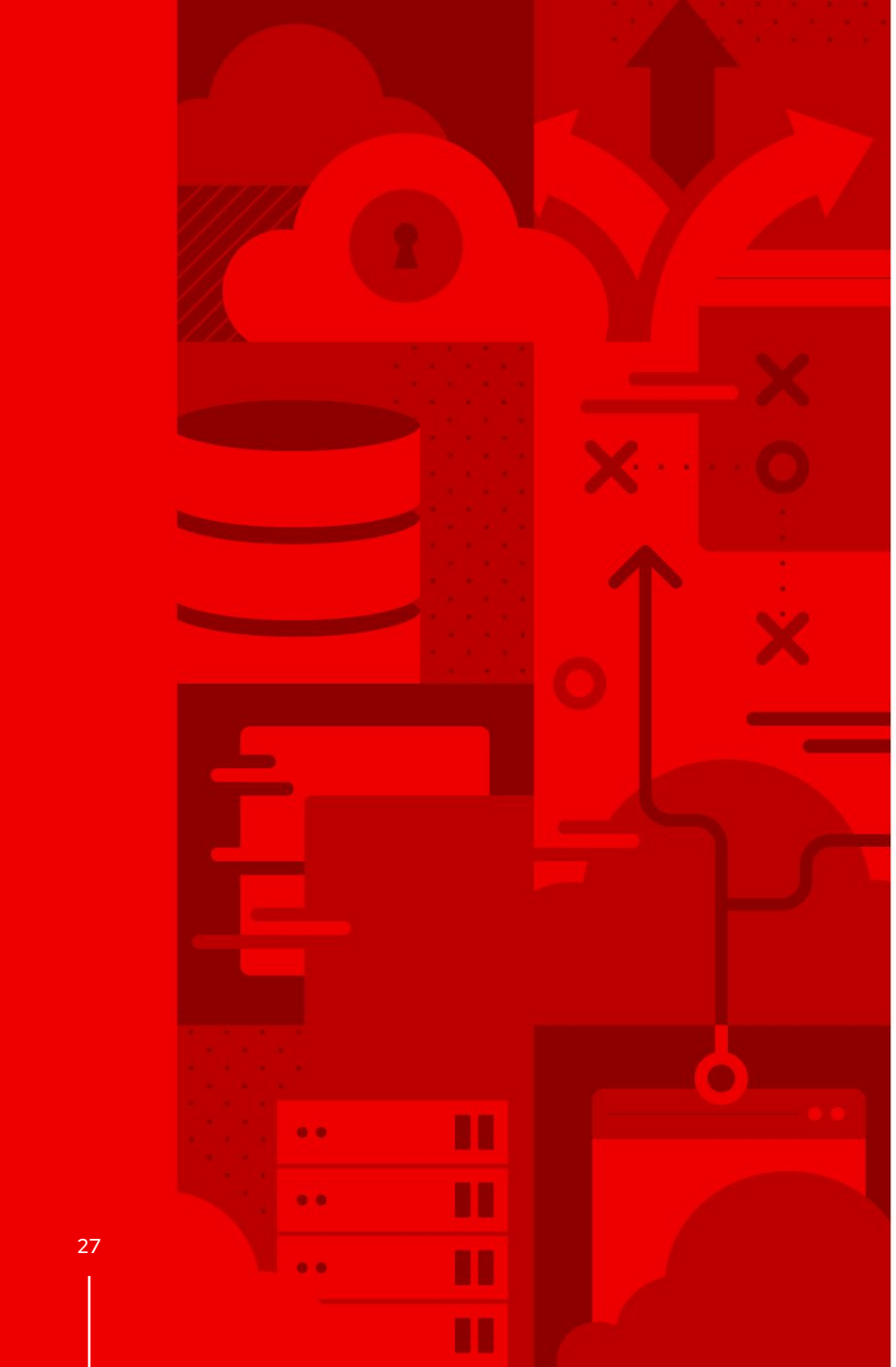
Standalone Container Hosts

Simplify and manage the OS in the same way as your applications

- **Flexible**: image mode provides a higher degree of host-level customization than previous immutable OSs.
- **Common tools**: use existing container tools and pipelines to build applications and the OS for better compatibility
- Align teams & operations around tools and process.



Red Hat



Edge Manager

Red Hat

Red Hat's edge computing platforms

Industry-specific solutions, co-developed with partners

Red Hat In-vehicle Operating System (RHIVOS)

Red Hat Device Edge

- Red Hat Ansible Automation Platform
- MicroShift (lightweight K8s)
- Red Hat Enterprise Linux (with edge features)

Red Hat OpenShift*

or

Red Hat Enterprise Linux (with edge features)

- AI & ML
- Management and automation
  - Red Hat Ansible Automation Platform
  - Red Hat Edge Manager
- Security
- Application and data services

Connected vehicles

Software-defined vehicles with connected services

Edge devices

Field-deployed endpoints and gateways with low footprint

Edge servers

Standard servers deployed in controlled environments

*OpenShift includes RHEL CoreOS, so no separate purchase needed for the Operating System

Edge customer challenges and opportunities

Mass scale of edge devices

People

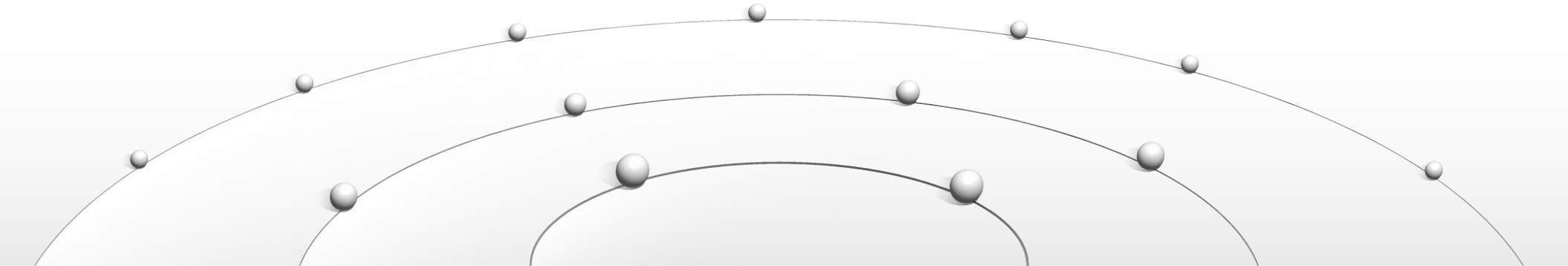
- Edge sites often **lack expert IT staff**
- Initial device setup to ongoing management **needs to be dead simple and robust**

Operations

- Many edge locations face **connectivity challenges**
- Solution must offer option for **on-premise management** that is on par or above cloud-based alternatives

Technology

- Current edge deployment process is **manual and complex**
- Simplifying this process is crucial for **enhancing user experience and adoption**



Red Hat Edge Manager

Comprehensive fleet management

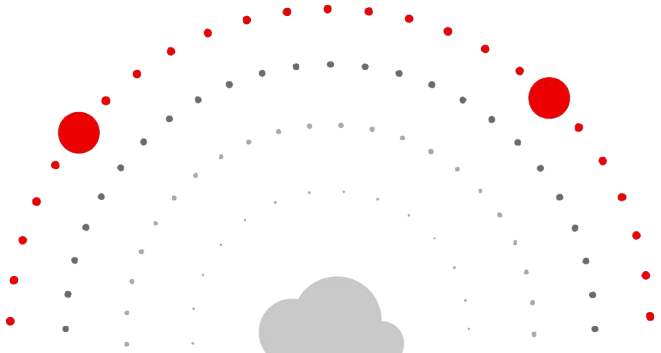
Simple

Intuitive edge operations

Bridge the IT skills gap at the edge with a user-friendly environment designed for ease of use and simplified management.

Flexible management options

Choose the deployment model that fits your strategy, with on-premises management offering comparable or superior value to cloud alternatives.



Scalable

Policy-driven deployment

Implement a desired-state configuration model for both applications and infrastructure, enabling consistent and scalable deployments, maximizing operational efficiency.

Resilient agent architecture

Use a robust agent-based architecture for scalable device management, maintaining connectivity and control even in challenging network conditions, without complex network configurations.

Built for edge

Hardened device communications

Employ mutual Transport Layer Security (mTLS) for robust and authenticated agent service communication. Establish a consistent security posture even for re-connected devices through rigorous identity verification.

Proactive device insights

Monitor critical device compute, memory and disk resources. Capture essential metrics and logs for effective issue remediation and operational awareness.

Lifecycle management

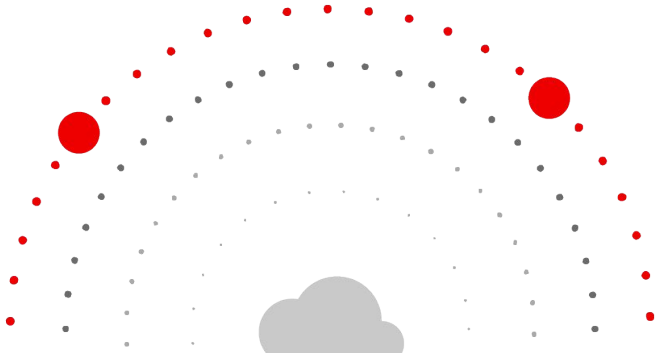
Provides secure onboarding, upgrades and decommissioning of applications and OS.

Red Hat Edge Manager

- Announcing Tech Preview at Summit
- GTM

  Red Hat Edge Manager (RHEM) will be available as part of the existing Red Hat Device Edge (RHDE) SKUs*

  - MW02581 Device Edge, Standard (1 socket up to 32 cores)
  - MW02577Device Edge, Premium (1 socket up to 32 cores)
- Early Access Program available.



*to be finalized at GA

Agent-based architecture

Network resilience

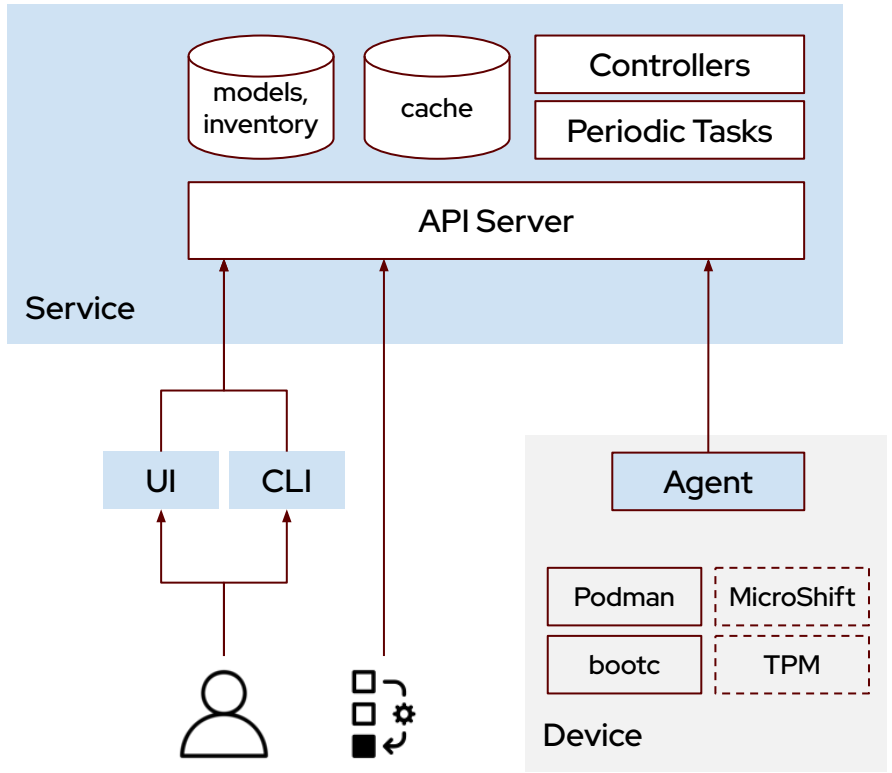
- Agent “calls home” when it has connectivity, so works with private, cellular, and intermittent networks. Without VPN.
- Can provide remote access when SSH cannot.

Safer, scheduled updates

- Downloads in background before making any changes.
- At scheduled time, transactionally updates operating system (if supported by OS) and host configuration.

Active status and progress reporting

- Reports update progress and changes to device and app health.



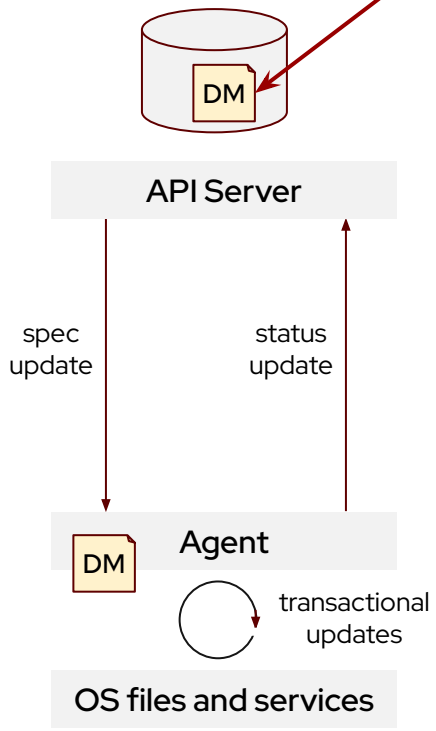
Model-based, declarative management

Models of current and desired state of managed objects.

- opinionated operating model
- versionable
- easy to sync to/from agent

Models are **declarative** (describe goal, not path to get there).

- Agent can implement autonomously and without network while considering local state.
- Enables roll back to previous version on error.



Device Model

```
spec:
 os:
   image: quay.io/flightctl-demos/rhel:9.6
 updatePolicy:
  updateSchedule:
    at: "0 2 * * *" # nightly at 2:00 AM
    timeZone: America/New_York
status:
 os:
   image: quay.io/flightctl-demos/rhel:9.5
 updated:
    status: Updating
    info: Download complete, waiting for
scheduled time to proceed.
```




Managing individual devices' Host OS and Configuration

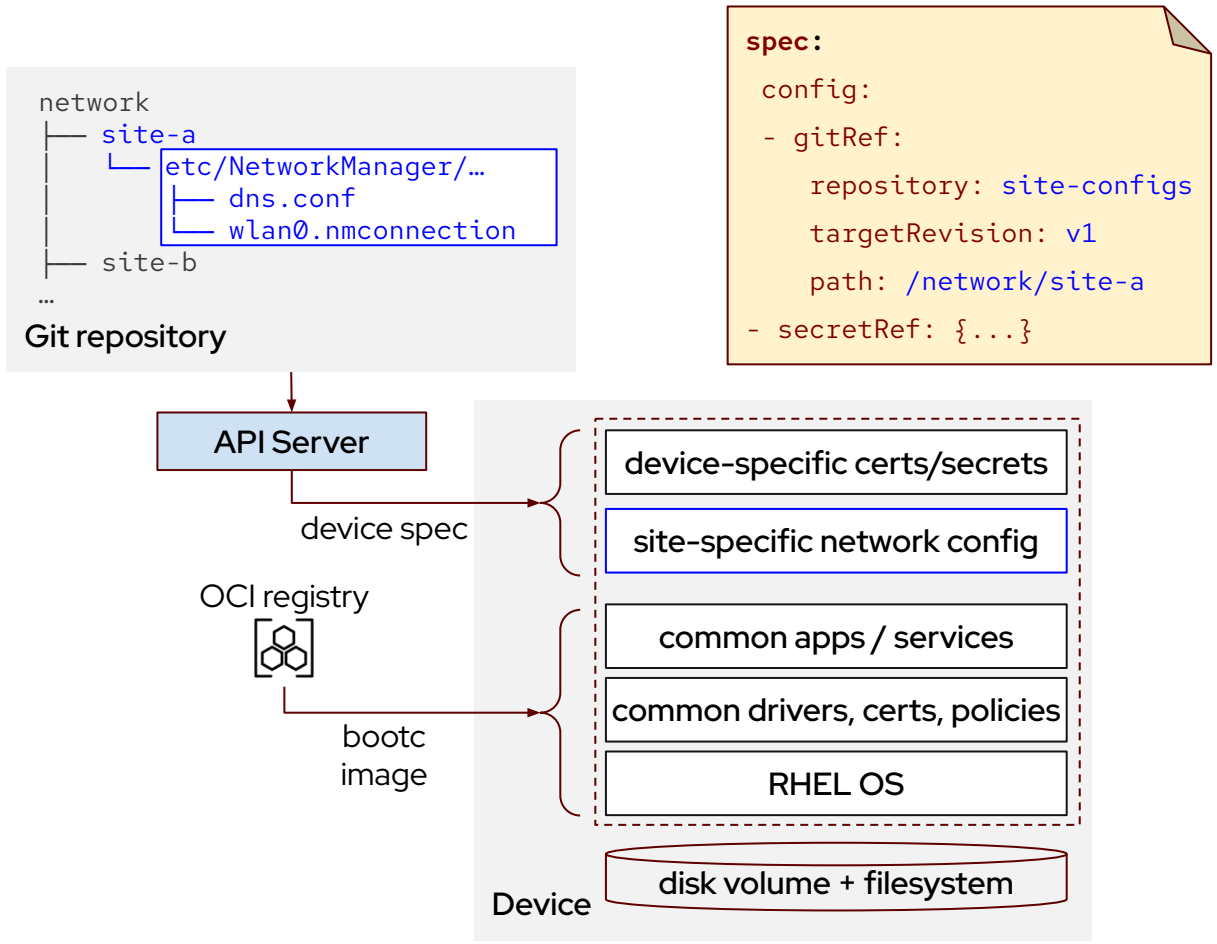
Image-mode RHEL to **minimize drift** between deployments.

- Agent *orchestrates* OS image update, but *delegates execution* to native tool (`bootc`).

Added simple model for syncing **device- or group-specific configuration** to disk and activating it without reboot:

- Service fetches configs from specified sources (Git, K8s Secrets, HTTPS), adds to device spec.
- Agent applies configs as "logical OS image layers" and signals systemd services or applications to reload config.

If simple model is too limiting, can fall back to user scripts in device lifecycle hooks or **automation through remote console**.



Managing individual devices' application workloads

Declare a list of **application workloads to deploy** to a device, running on Podman or MicroShift.

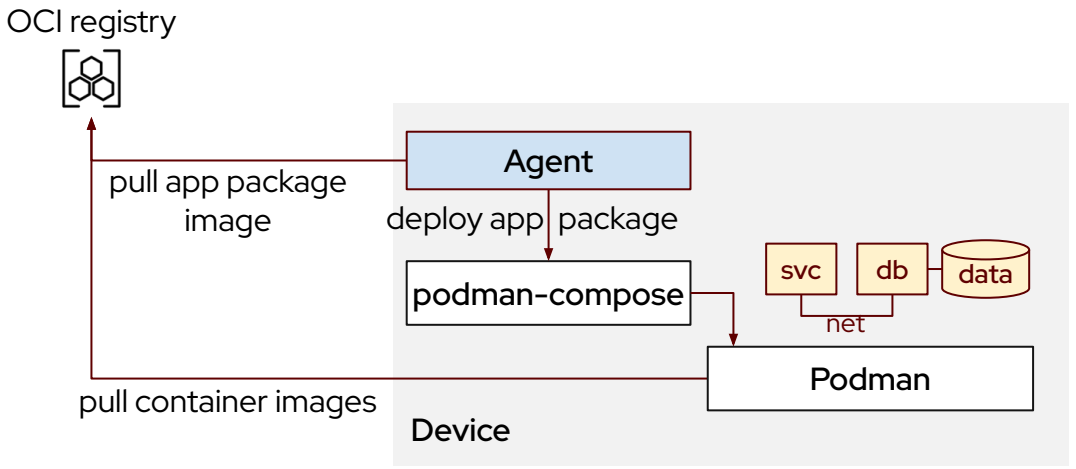
- Agent orchestrates app deploy/update, but delegates execution (e.g. `podman-compose`).
- Agent monitors application status and sends updates to service.

Pulls podman-compose **app packages from OCI registry**.

- Applications can be **multi-container** with logical volumes and networks.

Supports applications **embedded** in OS image.

```
spec:
 applications:
 - name: wordpress
   image: quay.io/flightctl-demos/wordpress-app:v1
   envVars:
     WORDPRESS_DB_HOST: "mysql"
     WORDPRESS_DB_USER: "user"
     […]
```

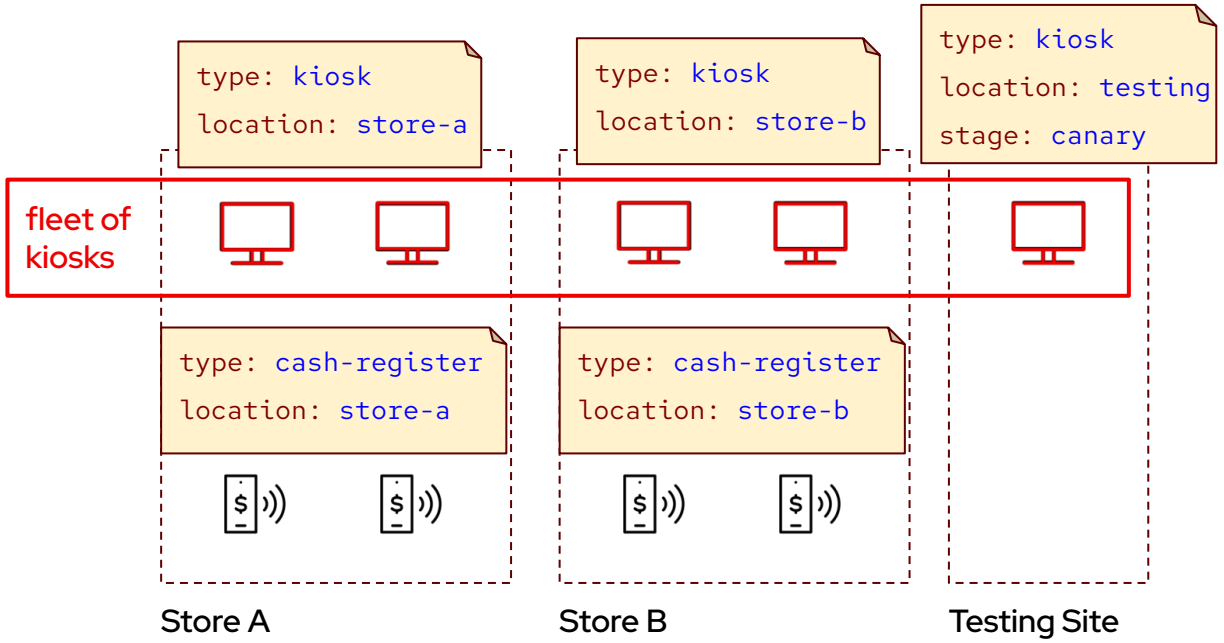


Scaling operations by managing fleets

Use **labels to organise** devices.

Use fleet abstraction to **manage many devices as one:**

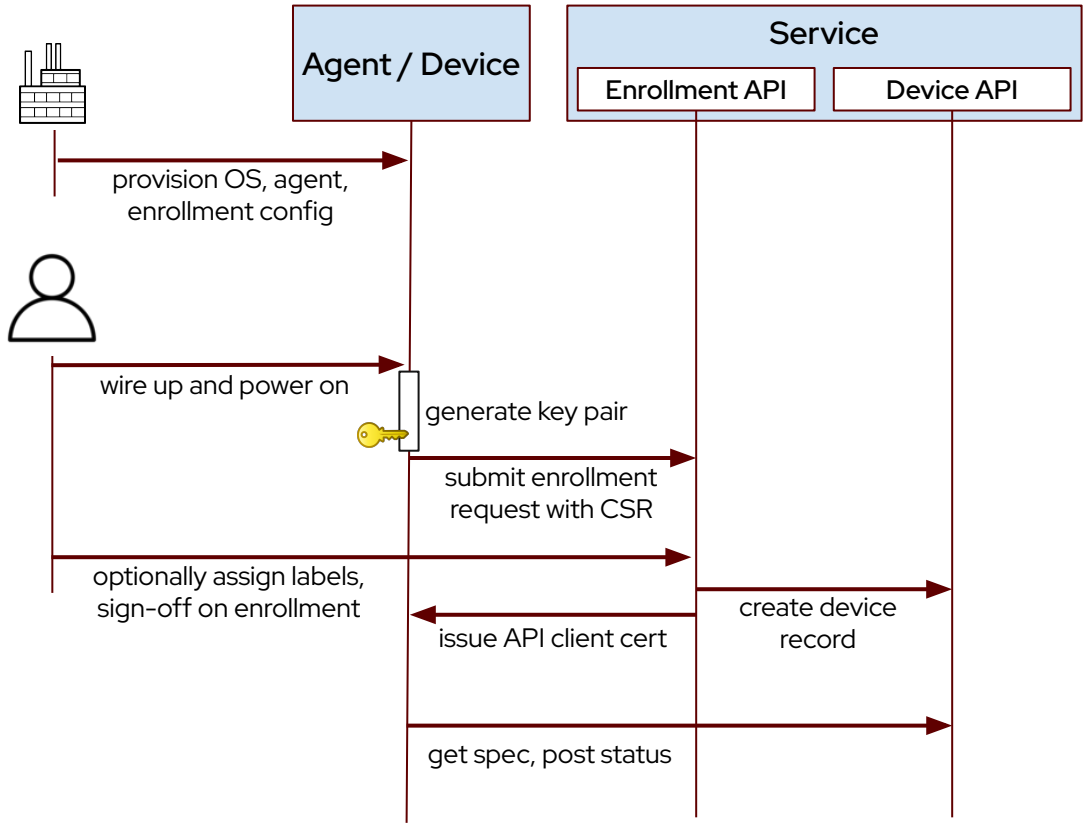
- Select devices into fleet based on labels.
  Selection applies to devices added or replaced later, too!
- Specify a **device template**. The service updates selected devices to conform with this template. Device's OS image, config, etc. can be selected based on assigned labels.
- Define a **rollout policy** that controls the order, grouping, and minimum service availability during rollout.
- Get a **live summary view** of the fleet's devices statuses.





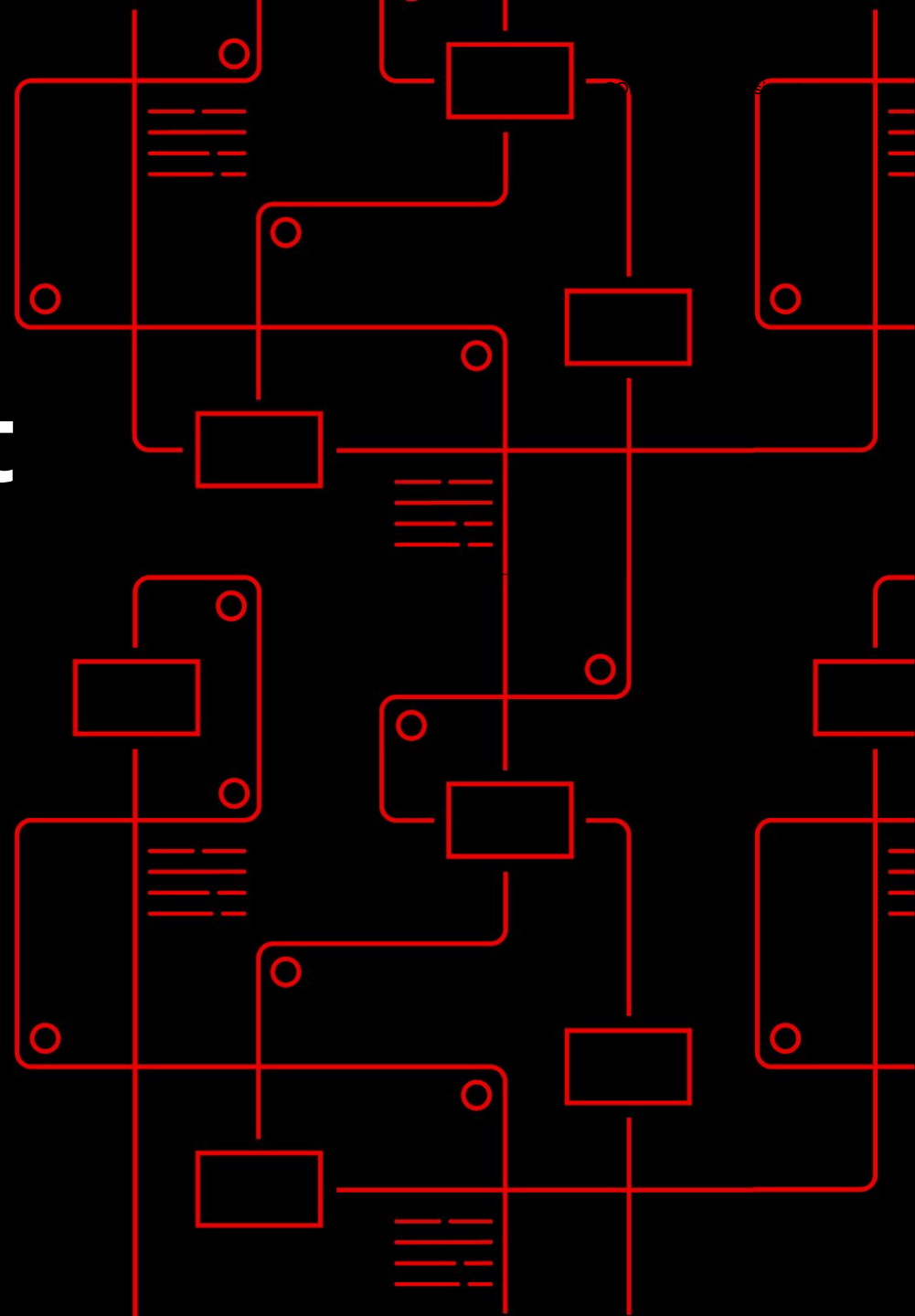
Device onboard to offboard

- Device provisioned with OS, agent, and enrollment config (*binds* device to owning management service).
- Agent generates pub/priv key pair: Pub key used as device crypto identity, priv key stored on disk or sealed in TPM.
- Agent sends Enrollment Request with system details and status and a PKCS#10 Certificate Signing Request (CSR).
- Only approver signed off on enrollment, a device record is created and the agent's unique client certificate is issued.
- When decommissioning, service instructs agent to wipe its priv key (making identity unusable) and records it as decom'ed. Factory reset and secure wipe on roadmap.

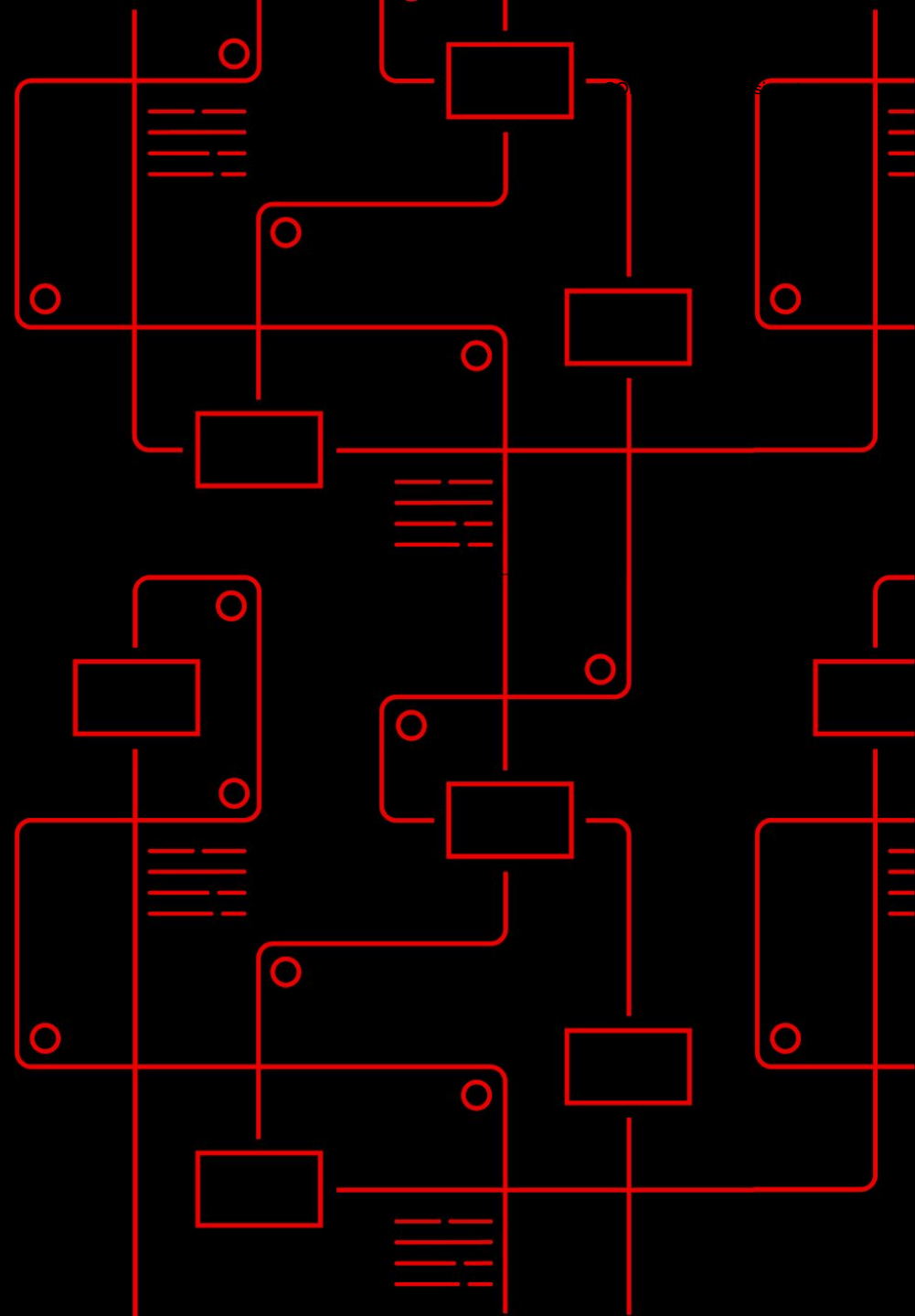




Managing Non-Red Hat Edge Devices

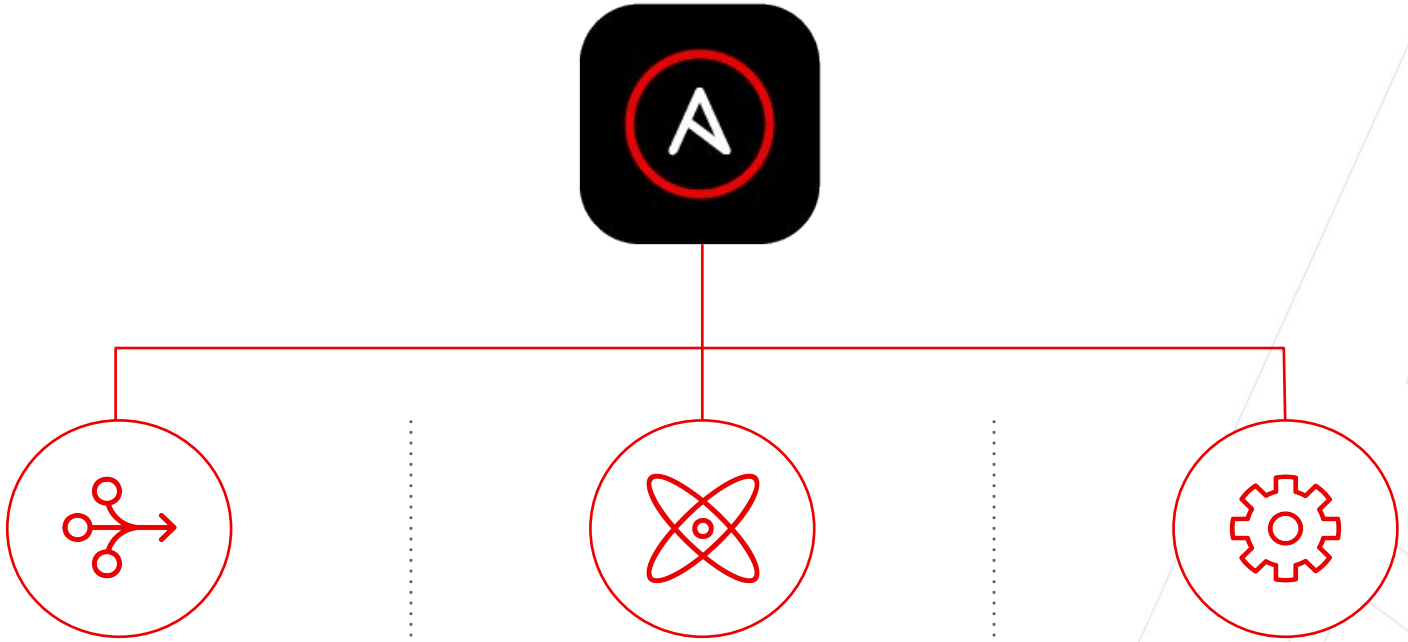


Red Hat Ansible Automation Platform Introduction



Red Hat
Ansible Automation
Platform

Enter Ansible, the de facto automation language



Simple

Human-readable language with quick adoption

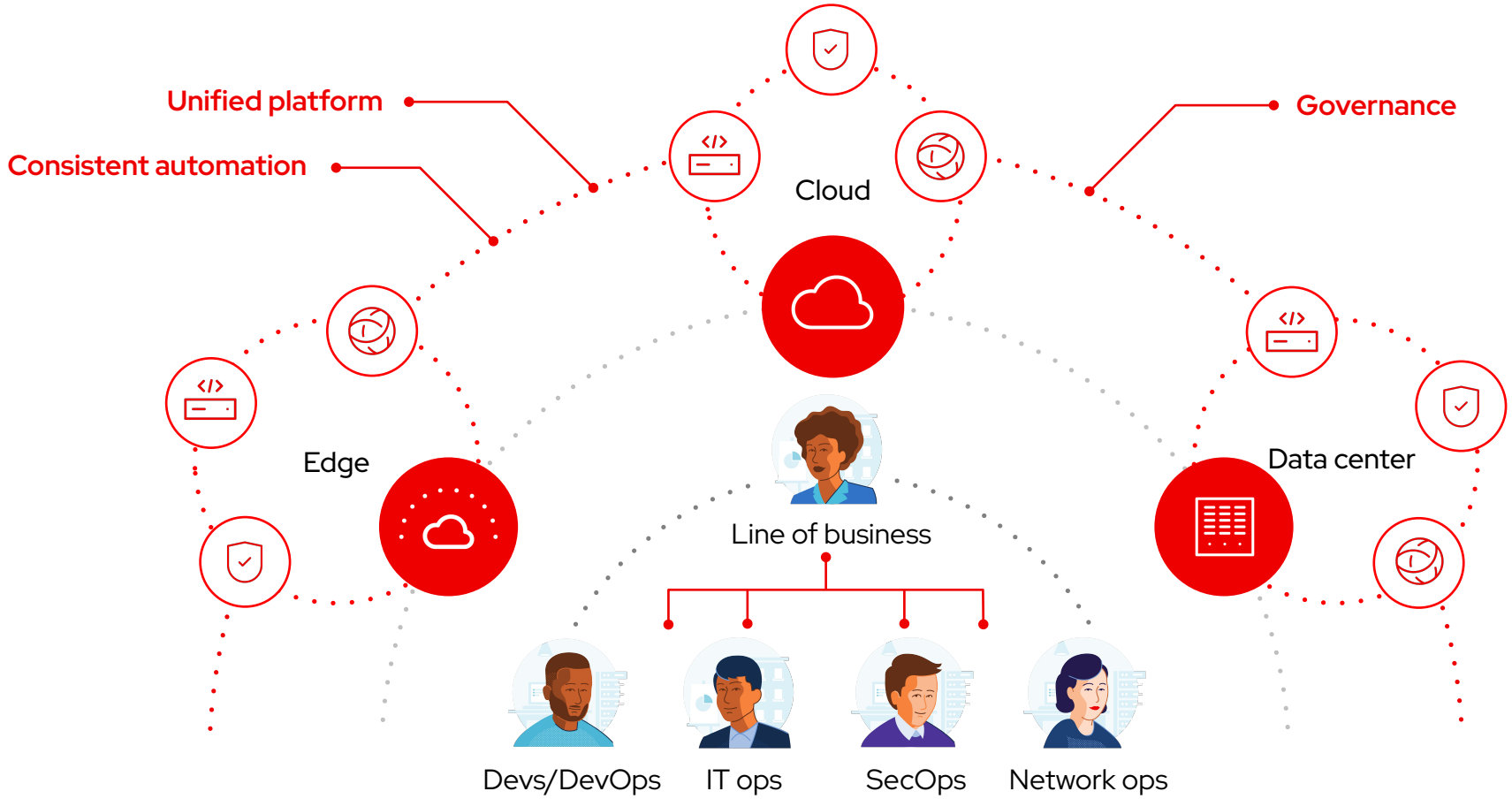
Powerful

Universal language spanning multiple IT domains

Agentless

Easily integrate with hybrid environments

Speaking the same language: datacenter and cloud to edge.



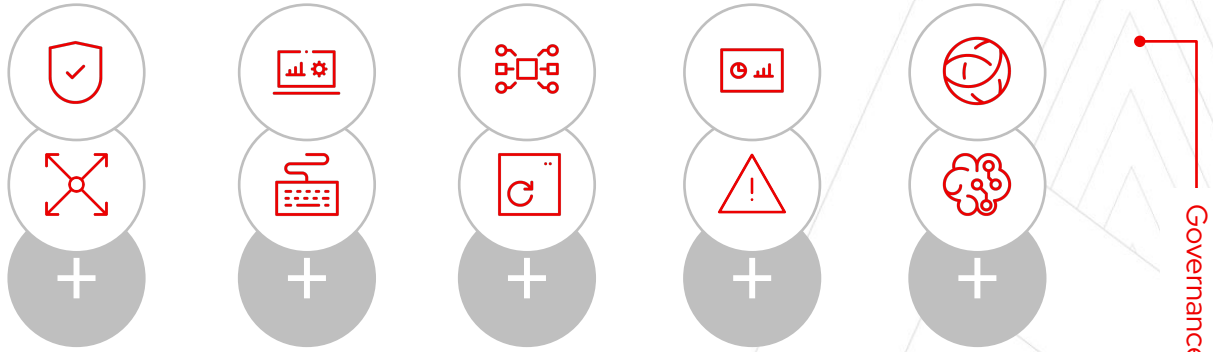
Red Hat
Ansible Automation
Platform

Many organizations share the same challenge.

Differing skills, roles, and responsibilities



Growing number of use cases

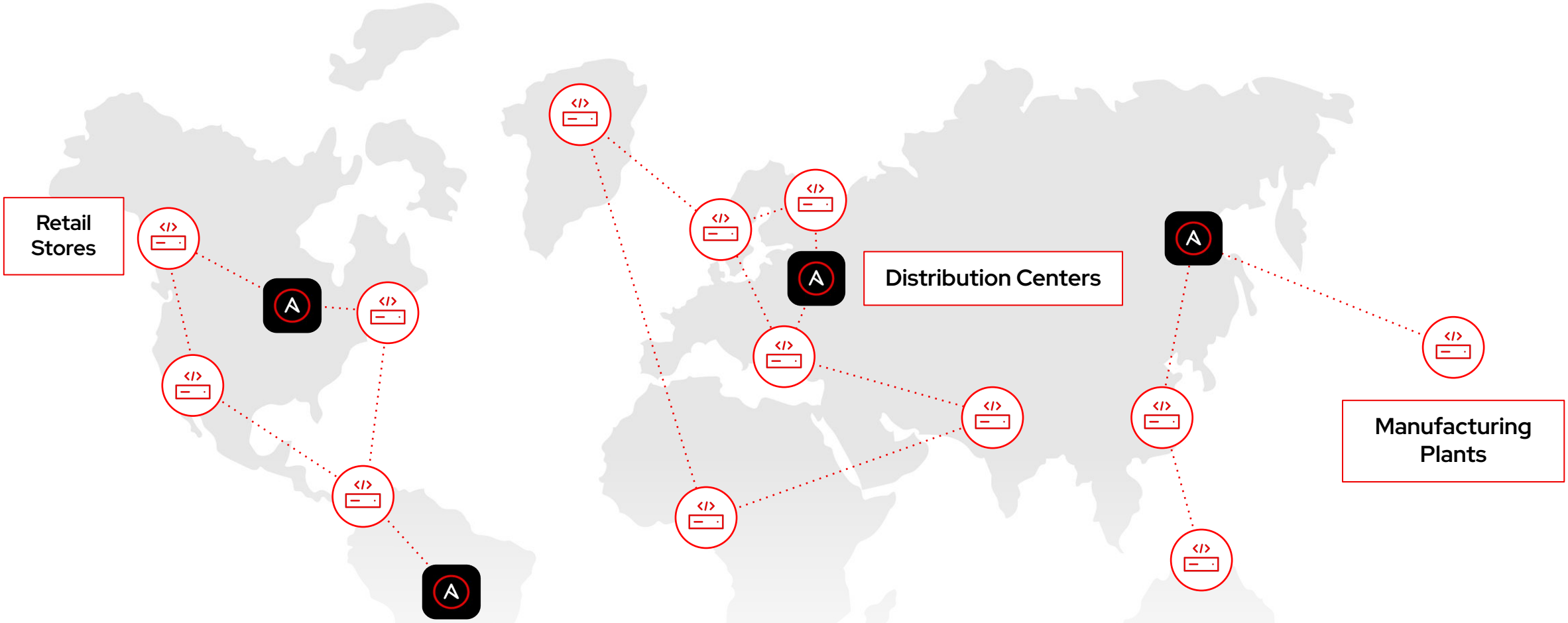


Multiple platforms and locations

Across multiple domains

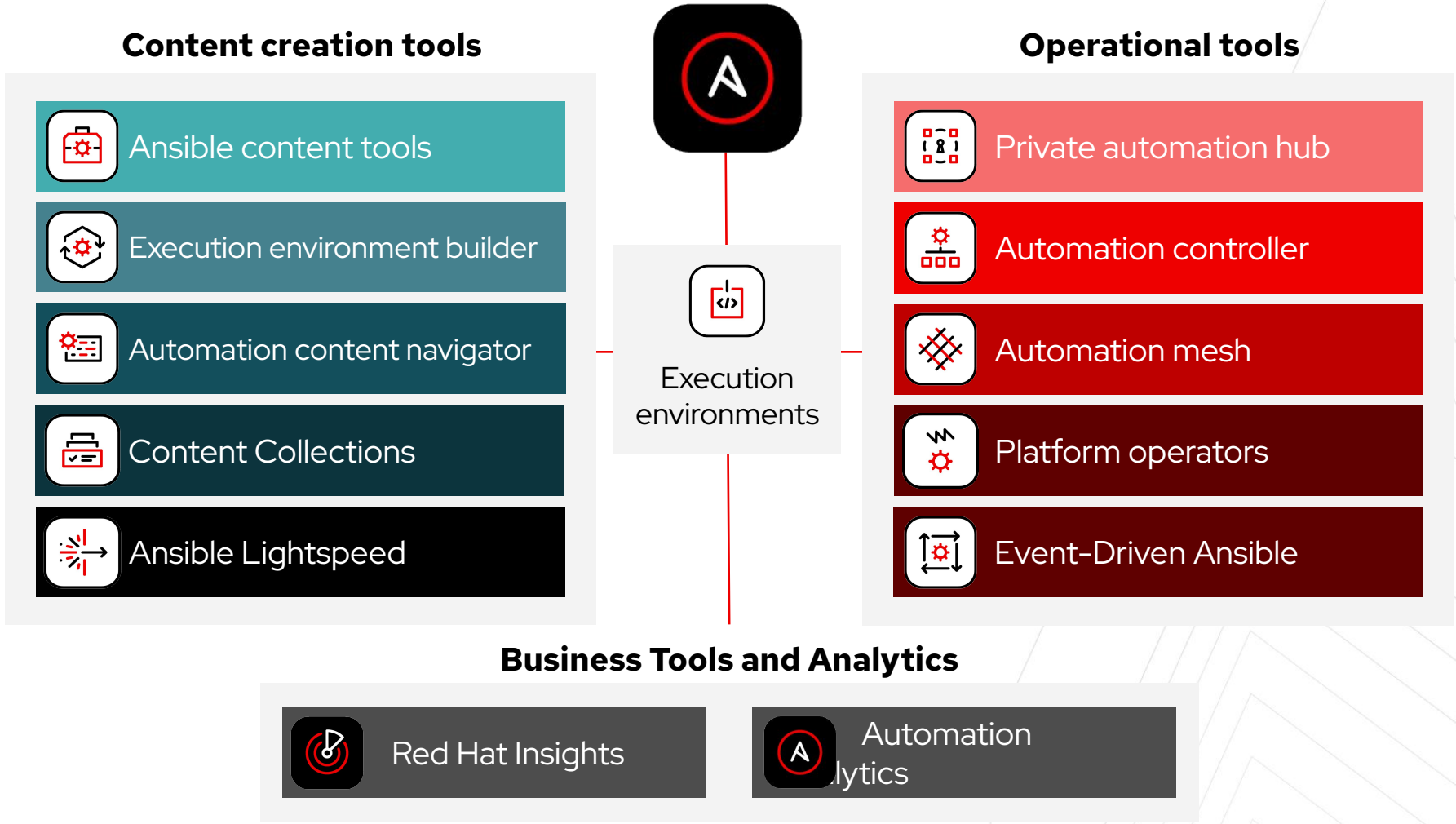


The flexibility to scale, wherever that may be.



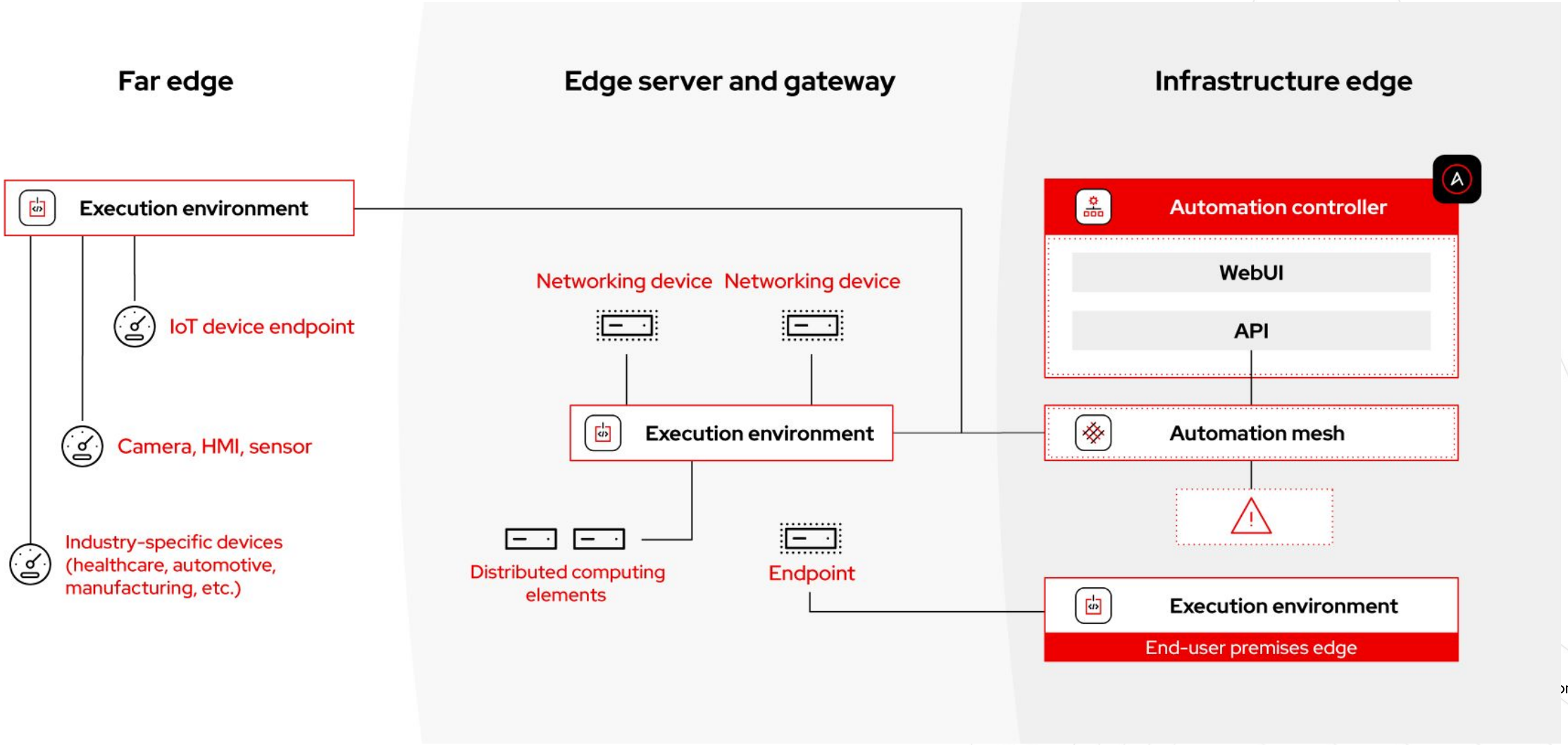
Red Hat
Ansible Automation
Platform

An integrated enterprise solution for the enterprise edge.

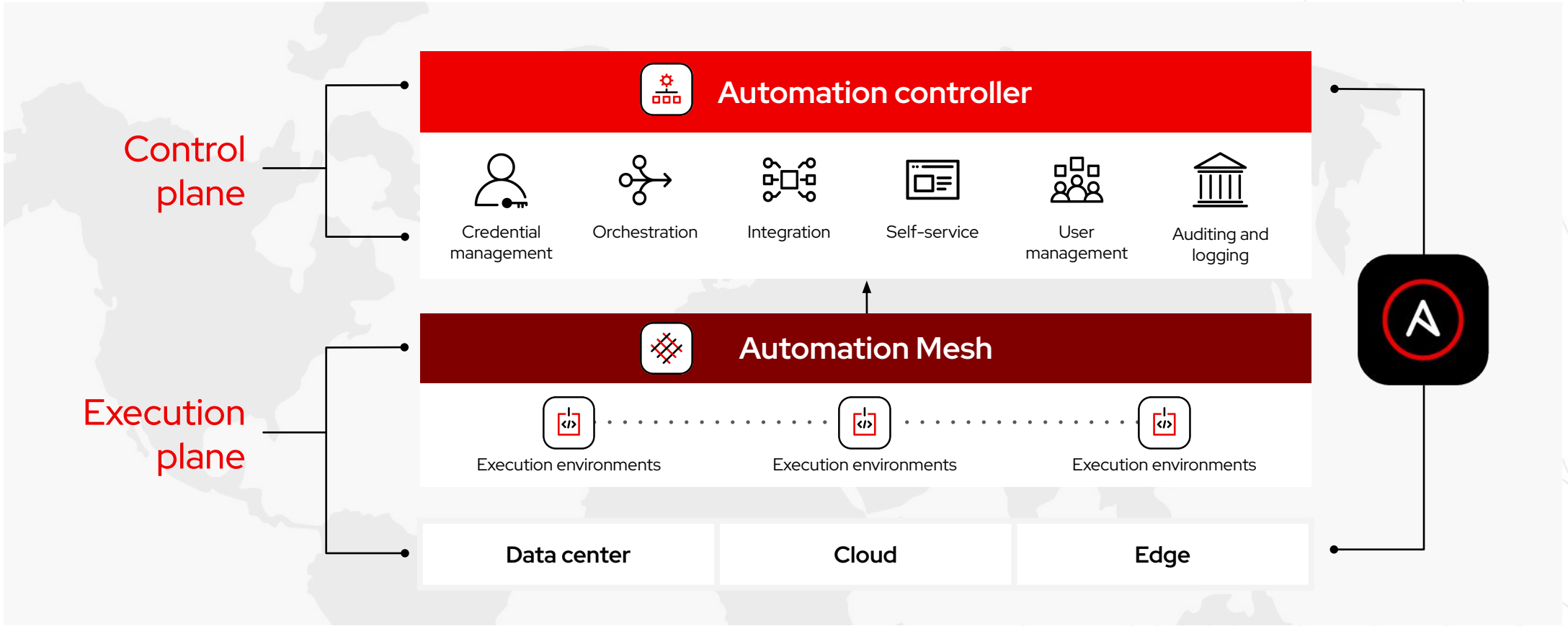


** Ansible Lightspeed with IBM Watson Code Assistant is available as non-supported technical preview.*
https://access.redhat.com/articles/7016058

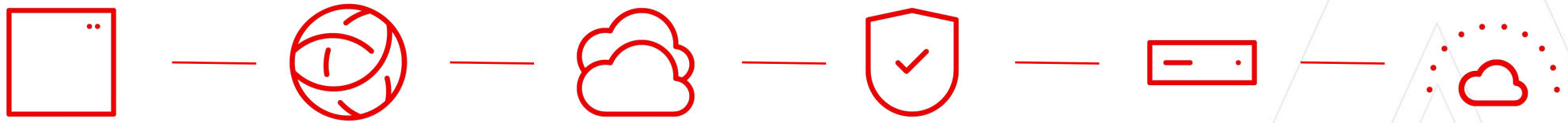
A distributed architecture with diverse workers.



A distributed architecture built for scale.

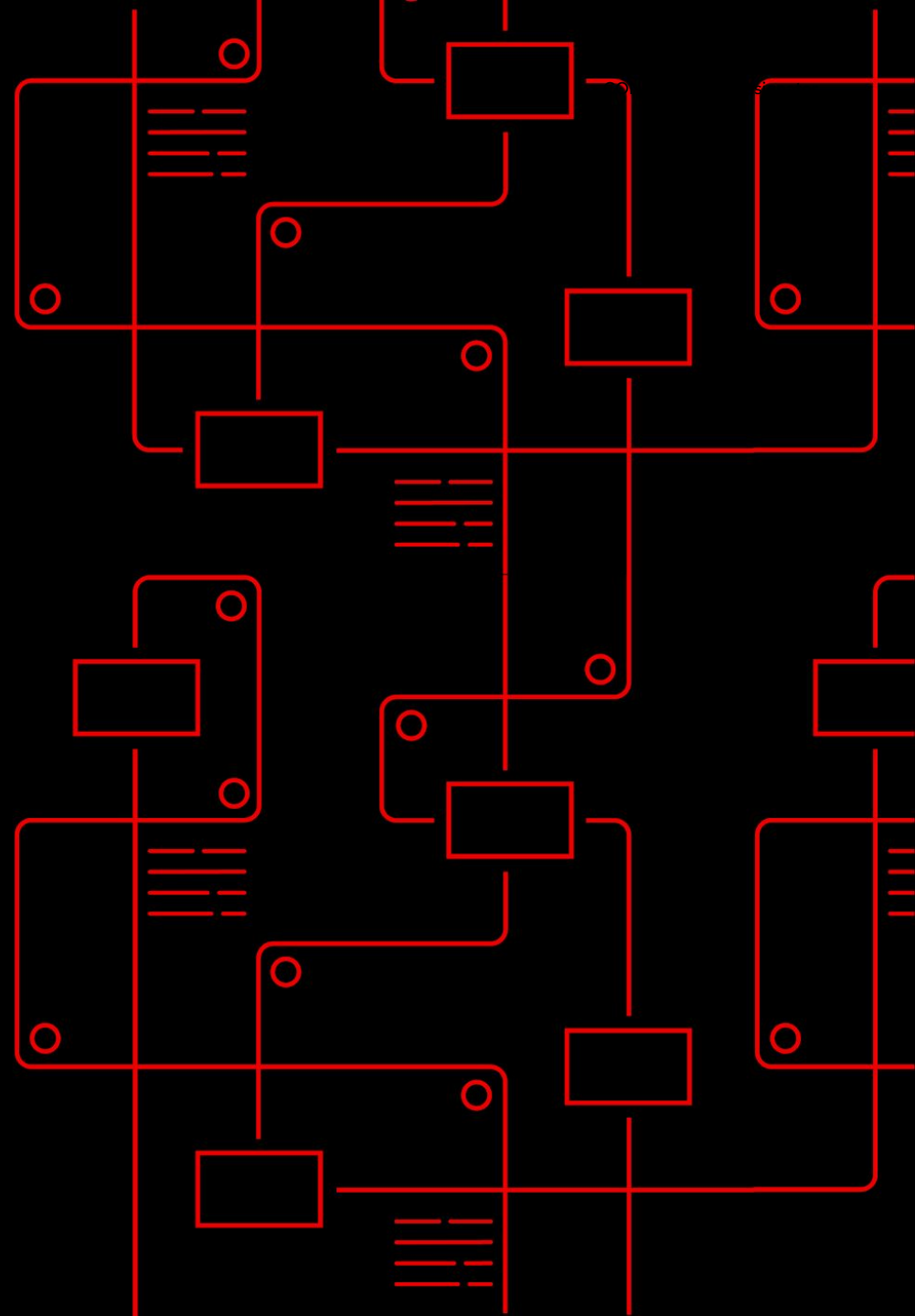


The capabilities you need across your IT and OT footprint.



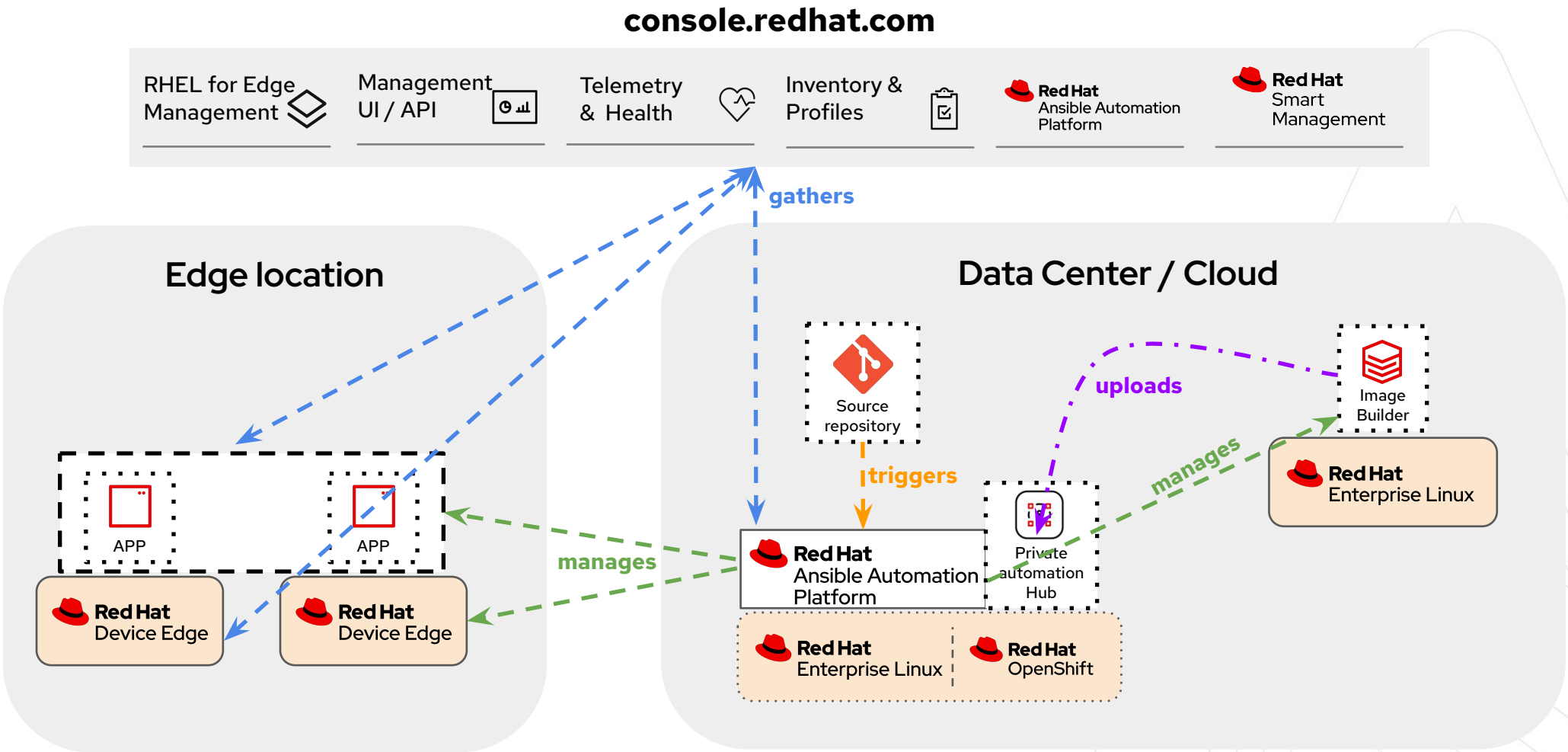
Applications	Network	Cloud	Security	Infrastructure	Edge
▸ DevOps ▸ CI/CD ▸ GitOps	▸ Configuration management ▸ Infrastructure awareness ▸ Network validation	▸ Orchestration ▸ Operationalisation ▸ Governance	▸ Investigation enrichment ▸ Threat hunting ▸ Incident response	▸ Deployment ▸ Provisioning ▸ Management	▸ Extend security ▸ Scalability ▸ Interoperability

Event-Driven Ansible Automation Platform for Edge



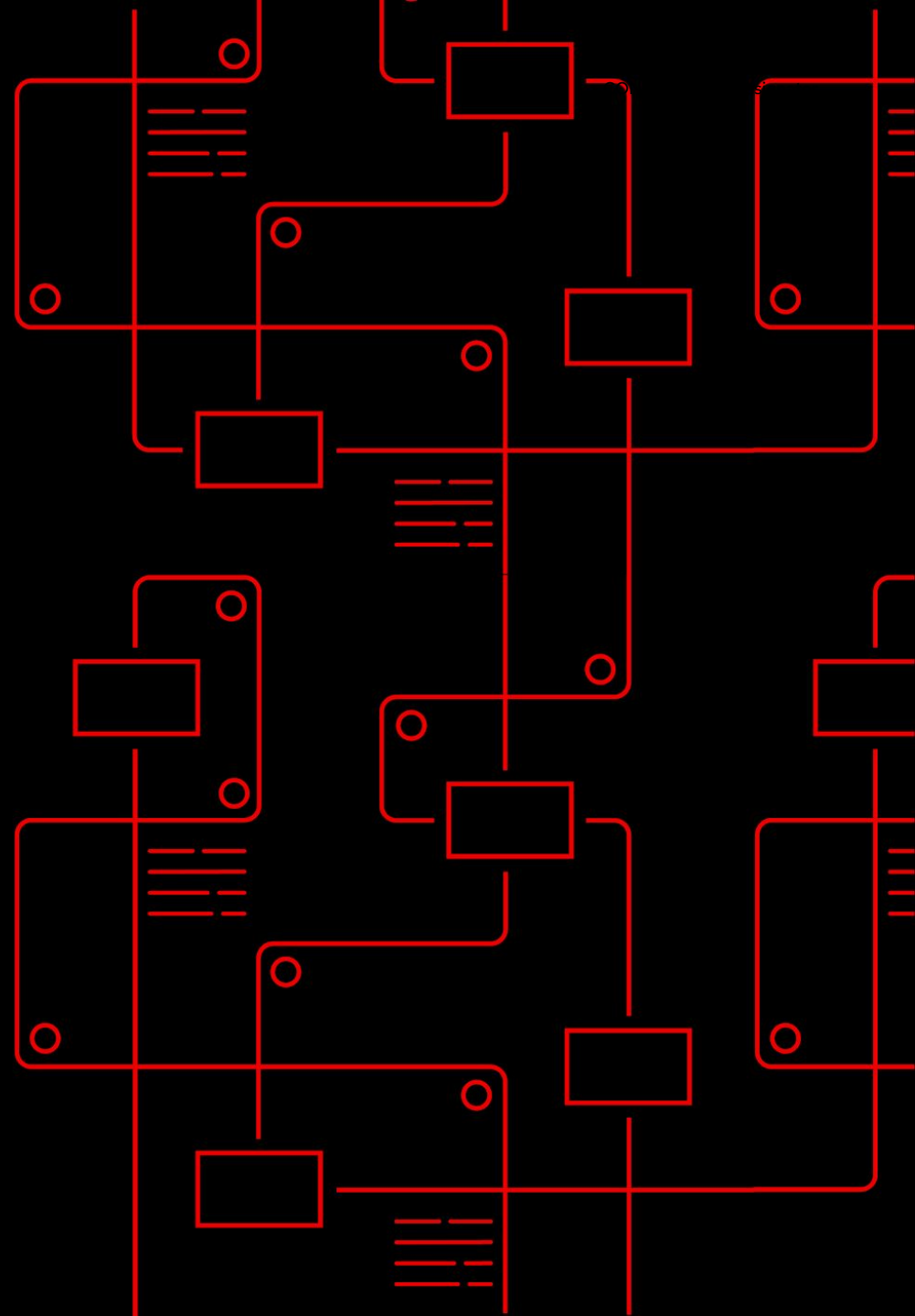
Red Hat
Ansible Automation
Platform

Demo architecture



Red Hat

Ansible Automation Platform Mesh Architecture Concepts



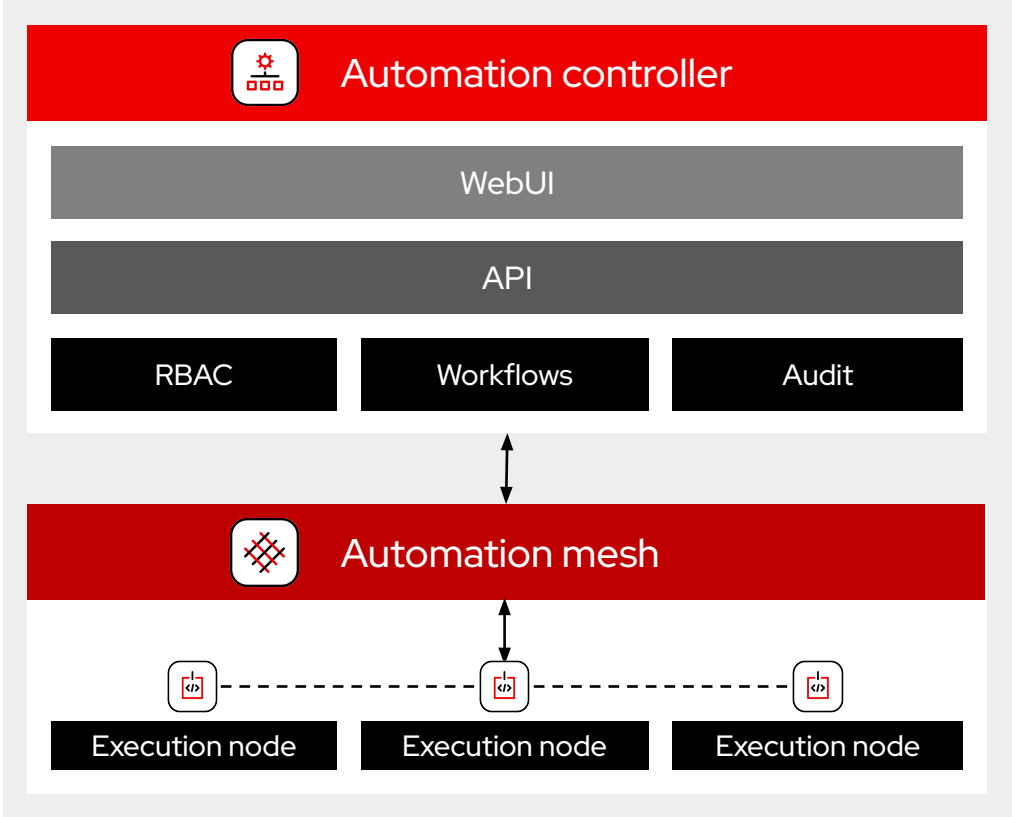
Automation mesh. Design and expand.

What is it?

- Simple, reliable framework to scale automation independently with little or no downtime
- Distributed and localized execution across worker nodes
- Scale across segmented and remote networks with native resiliency
- Flexible design choices to deliver automation across networks and design redundant topologies

Improved communication and security

- Worker node health checks
- Bi-directional, secured (TLS) communication between nodes
- Centralized, secured management with automation controller



Automation mesh node types. The execution plane.

What is it?

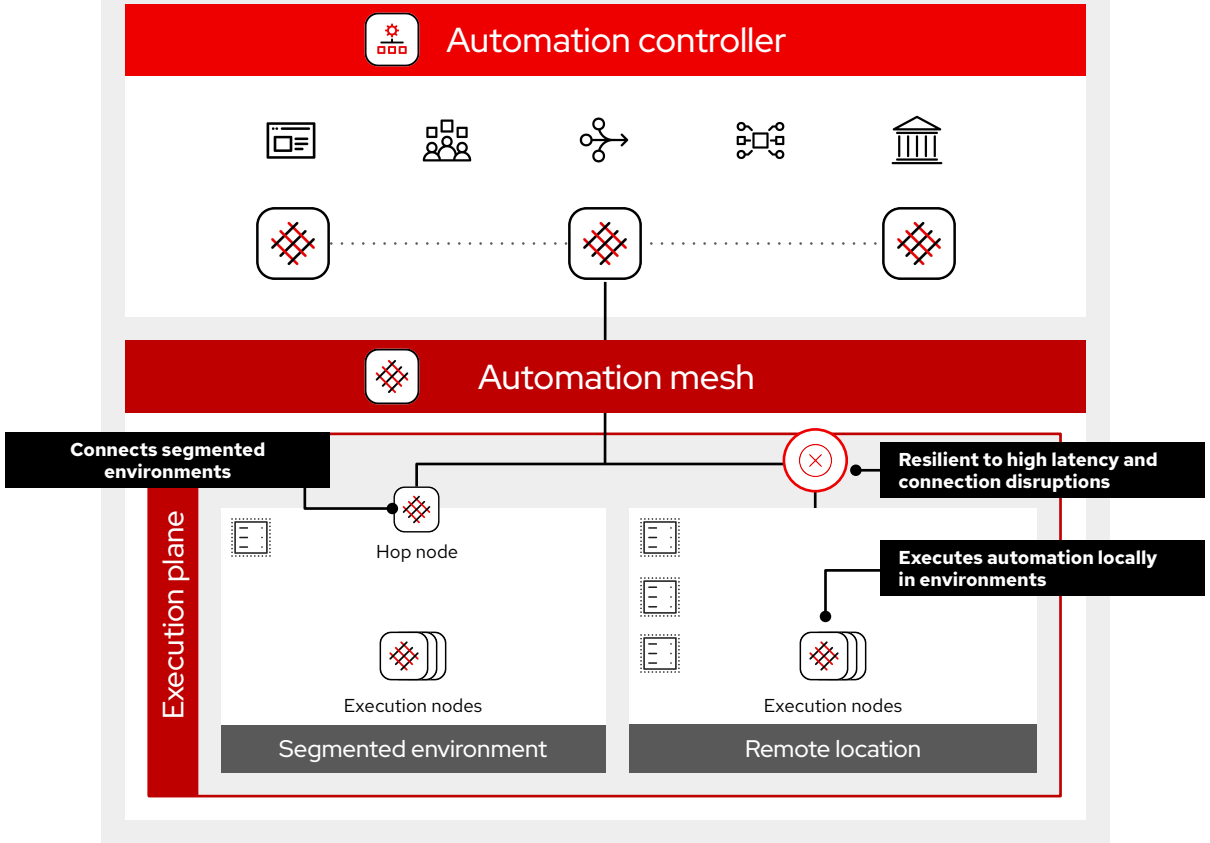
- Dedicated capacity to running playbooks
- Flexible designs possible across geographies and networks
- Resilient to high latency and connection disruptions
- Run automation without direct connection to controller

Execution node

- Dedicated capacity to run automation and no controller runtime functions executed

Hop node

- Dedicated to route traffic across execution nodes not directly connected to controller and cannot run automation



Automation mesh node types. The control plane.

What is it?

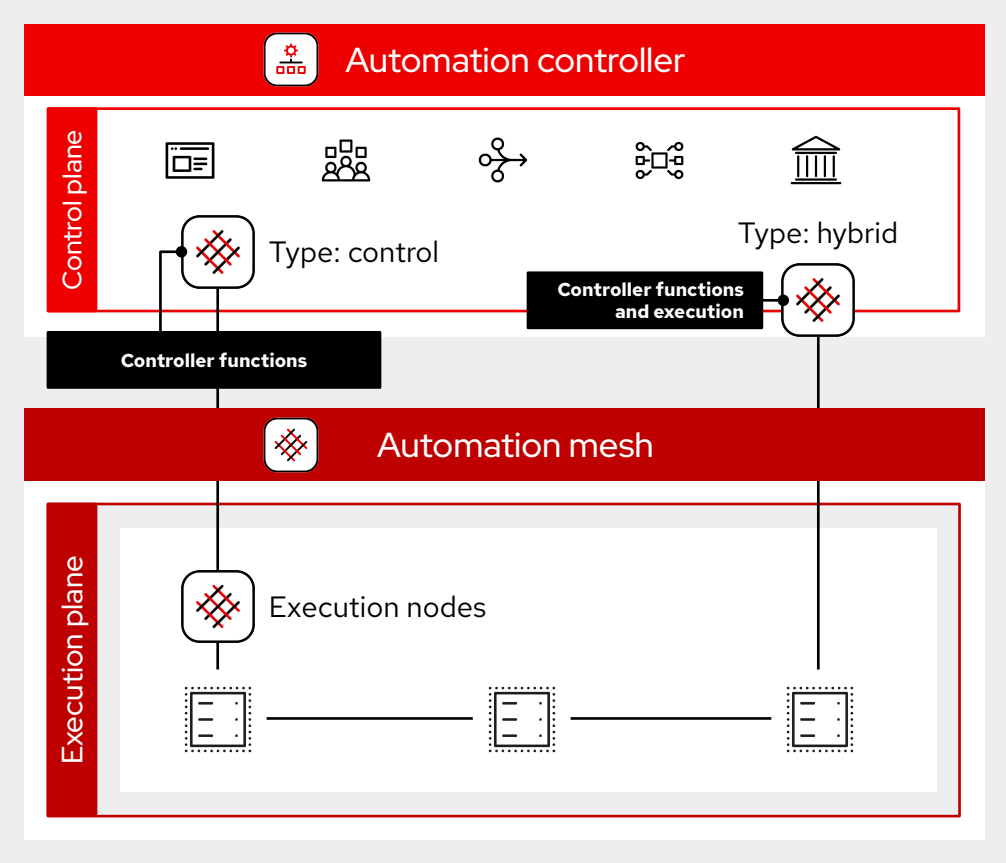
- Dedicated capacity for automation controller runtime functions
- Scale with little or no downtime
- Reduced operational overhead

Hybrid nodes

- Default node type for controller nodes
- Perform controller functions and execute automation

Control nodes

- Capacity is dedicated to controller functions
- Automation execution capability is disabled



Ansible Automation Platform Mesh for Edge

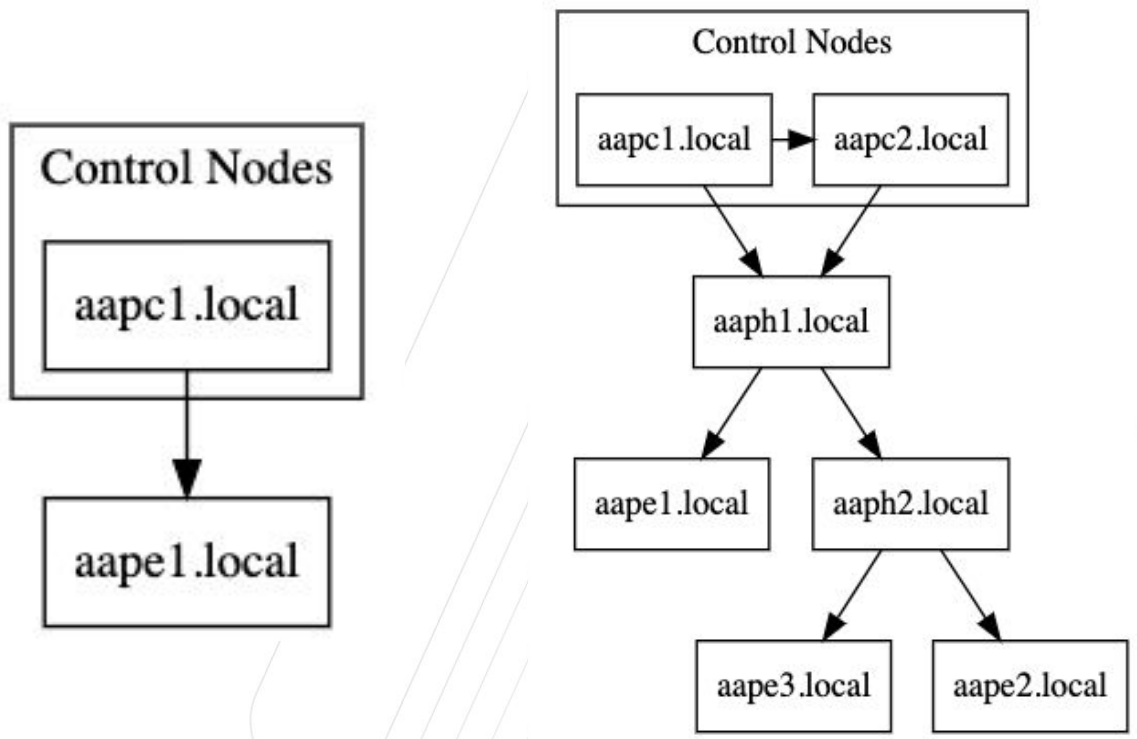


Red Hat
Ansible Automation
Platform

What is mesh?

Officially... an overlay network intended to ease the distribution of work across a large and dispersed collection of workers. Nodes establish **peer-to-peer** connections with each other via existing networks.

Configure your automation ***how*** you want it to work and ***where*** you want it to run



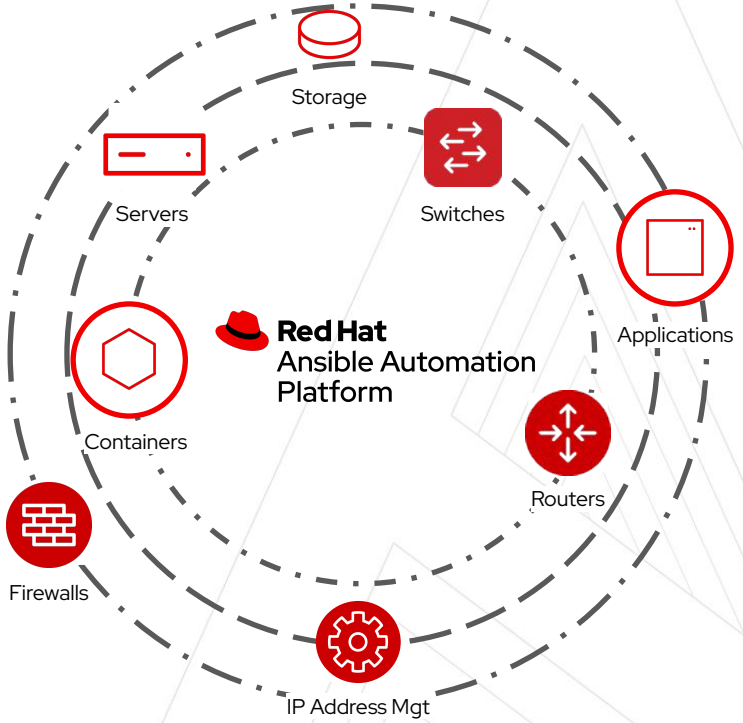
Use Cases ie why do I care about mesh

Remote Branch Office Use Case

Remote Branch Office

Retail, banks, warehouses have a need to: orchestrate, manage configurations, deploy applications, provision /deprovision, secure and comply, deliver continuously at *remote sites*.

Ansible Automation Platform provides a centralized way to deploy, standardize and validate their multi-vendor and multi-domain systems to all locations, using a common and human-readable language.



Red Hat

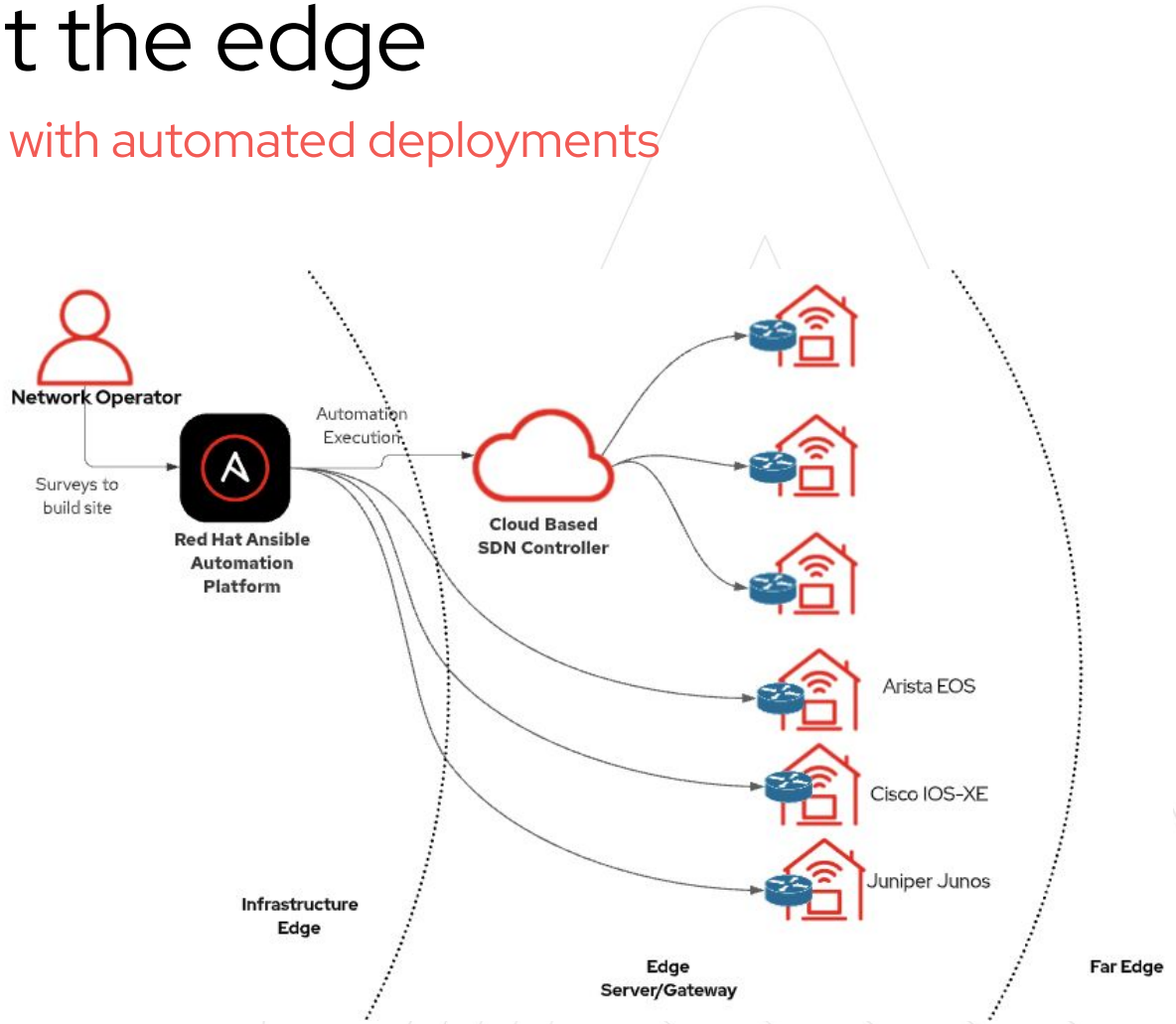
Networking at the edge

Expansion efficiency/simplification with automated deployments

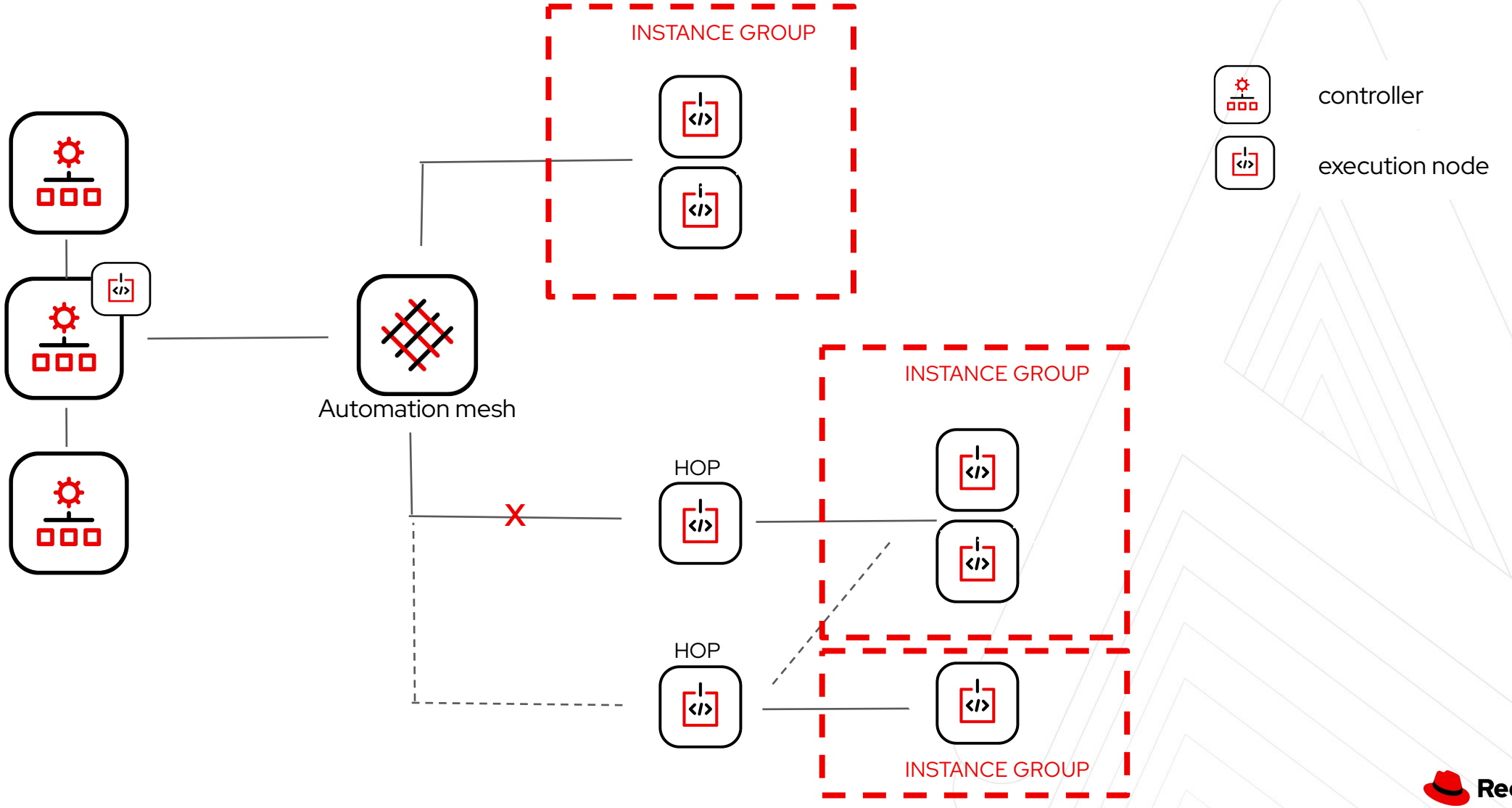
During IT expansion, manual intervention increases the security risks, reworks and additional operational costs, due to human error.

This is true for any corporate remote office but, when we add a component of scale things become even more challenging.

Retail, restaurant, supermarket, pharmacy chains must deploy thousands of new sites. In the best scenario, when having SDN controllers, building a single site requires at least 30 min per new branch.

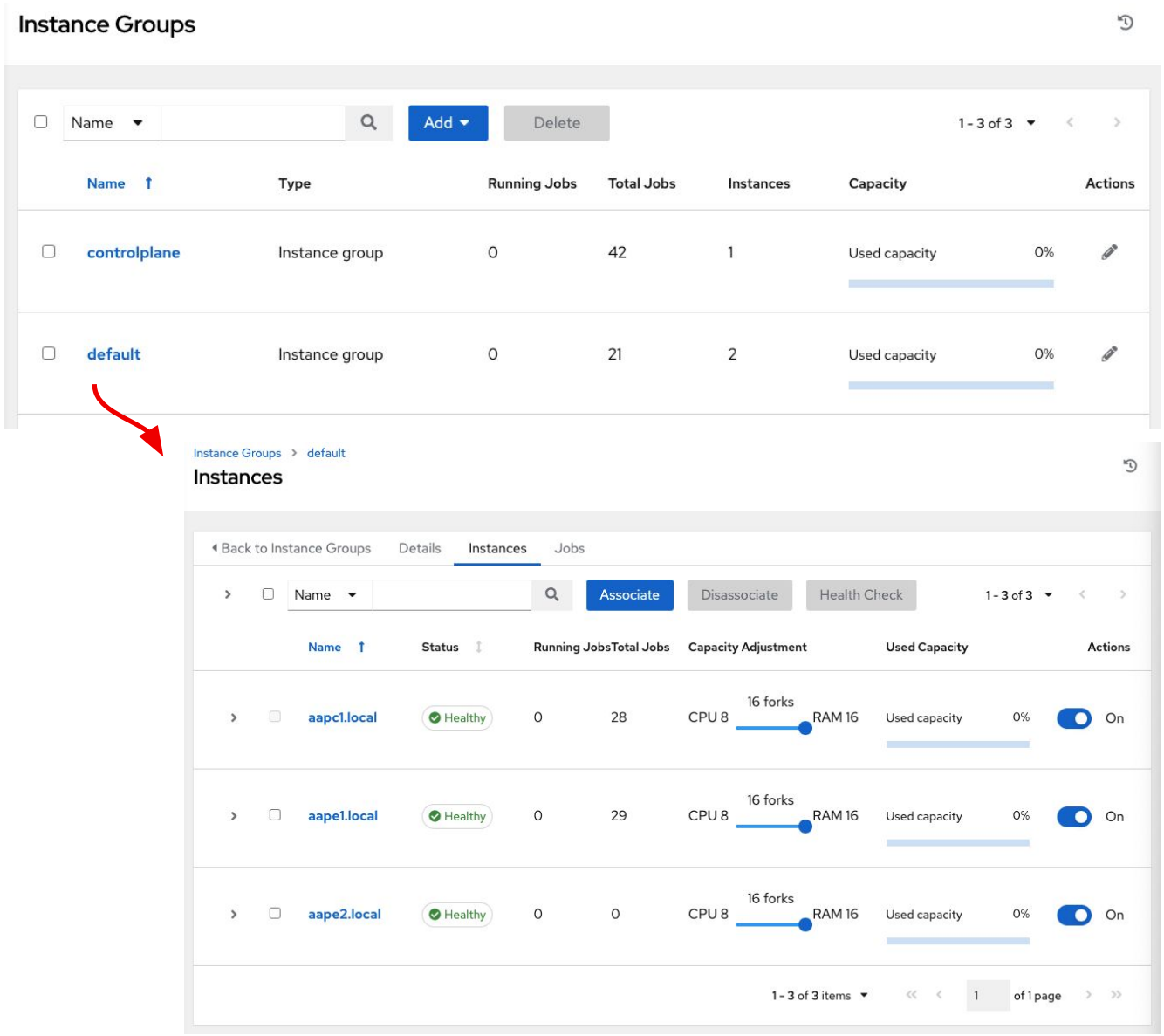


Mesh Simplified

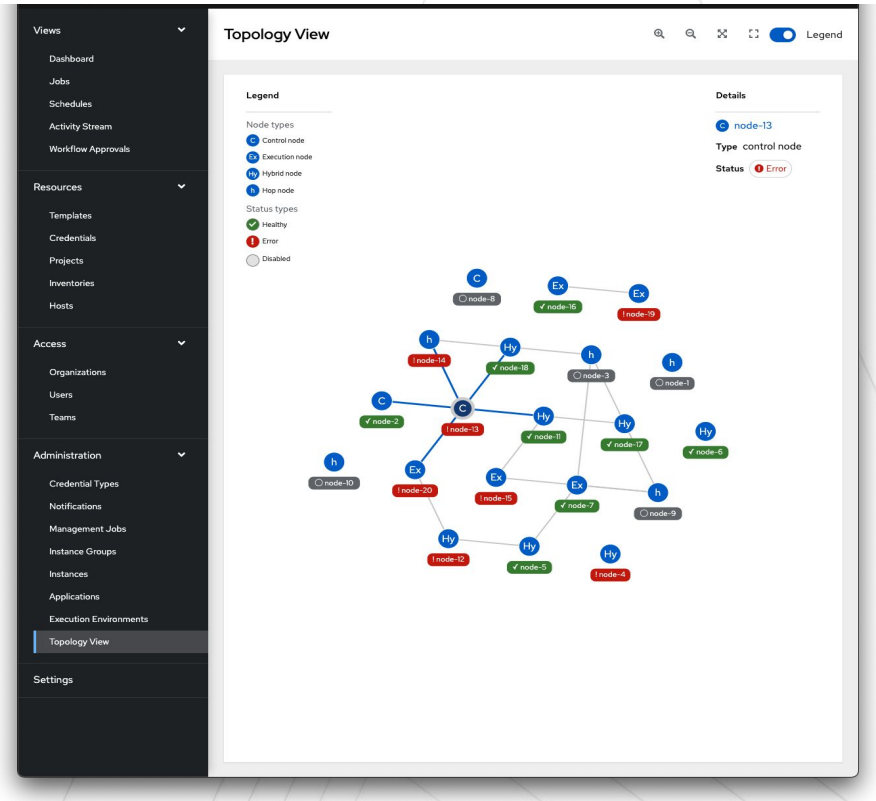


Red Hat

Mesh Visuals



* NEW *



Red Hat
Ansible Automation
Platform

Mesh Node Types - **control** plane

HYBRID

- **Default** for controller nodes
- Responsible for controller runtime functions like project updates, management jobs and [ansible-runner] task operations
- Will **also** be used for automation execution

CONTROL

- Controller nodes with execution capabilities **disabled**

```
[automationcontroller]

aapc1.local ansible_connection=local node_type=control
```


Red Hat

Mesh Node Types - **execution** plane

EXECUTION

- execute automation on behalf of controller nodes
- no control functions
- runs jobs under *ansible-runner* with *podman* isolation
- akin to isolated_nodes

HOP

- akin to a jump or relay host
- will *route* traffic to other execution nodes
- **cannot** execute automation

```
[execution_nodes]

aaph1.local node_type=hop
```

Execution and hop nodes are collectively called the ***execution plane***

Maybe:

- geographically separated *with* high latency from the control plane
- not even *directly* connected to the cluster

Red Hat

Graphing Automation Mesh

Inventory file:

```
[automationcontroller]
aapc1.local ansible_connection=local
node_type=control

[automationcontroller:vars]
peers=aaph1.local

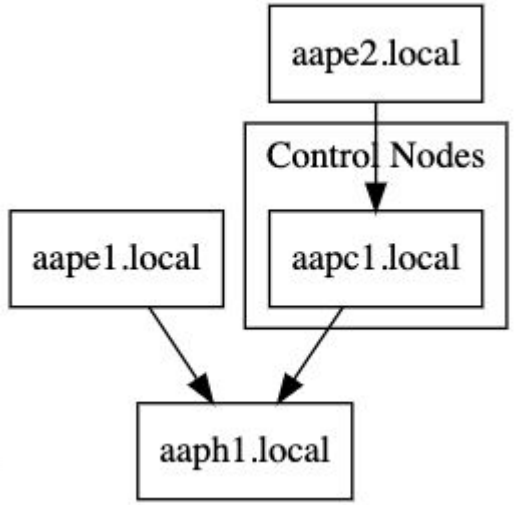
[execution_nodes]
aaph1.local node_type=hop
aape1.local peers=aaph1.local
aape2.local peers=aapc1.local
```

./setup.sh **-e generate_dot_file=/tmp/mesh.dot** ...
./setup.sh **-- --tag generate_dot_file** ...
[creates *mesh-topology.dot* in the $cwd in secs]

https://dreampuf.github.io/GraphvizOnline/

```
strict digraph "" {
    rankdir = TB
    node [shape=box];
    subgraph cluster_0 {
        graph [label="Control Nodes", type=solid];
        {
            rank = same;
            "aapc1.local";
        }
    }

    "aape1.local";
    "aaph1.local";
    "aape2.local";
    "aapc1.local" -> "aaph1.local";
    "aape1.local" -> "aaph1.local";
    "aape2.local" -> "aapc1.local";
}
```



Red Hat

Mesh Validation


```
TASK [Parse Mesh] ************************************************************
fatal: [aapc1.local]: FAILED! => {"msg": "Cannot have a peer to a node that doesn't exist: aaph1.local"}
```

Other common captures:

- *Cannot have nodes peering to themselves*
- *The following hosts cannot be members of both [automationcontroller] and [execution_nodes]*
- ***Two-way link(s) detected: Cannot have an inbound and outbound connection between the same two nodes***
- ***There are receptor nodes that do not have a path back to the control plane***

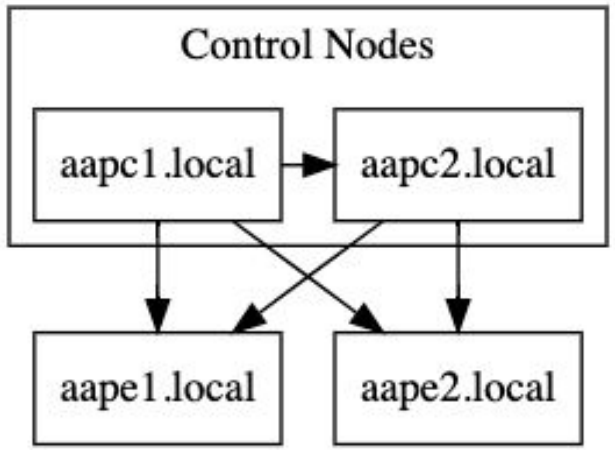
Red Hat

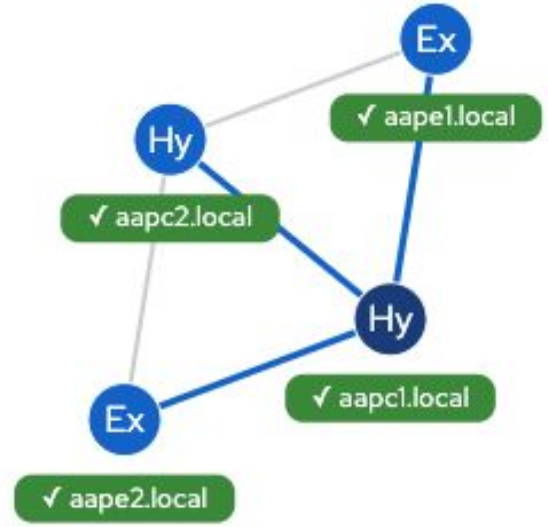
Minimal resilient config

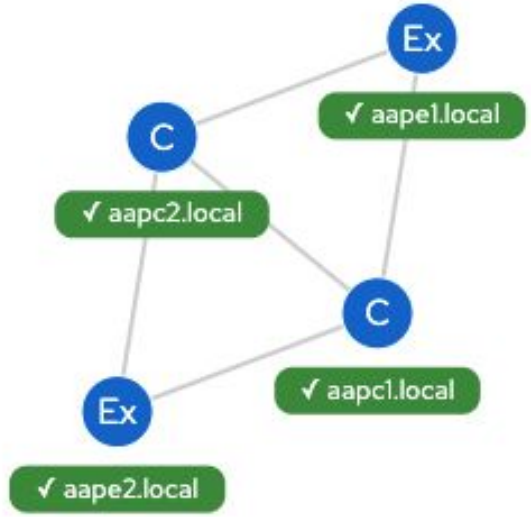
```
[automationcontroller]
aapc1.local ansible_connection=local
aapc2.local

[automationcontroller:vars]
peers=execution_nodes
node_type=control

[execution_nodes]
aape1.local
aape2.local
```

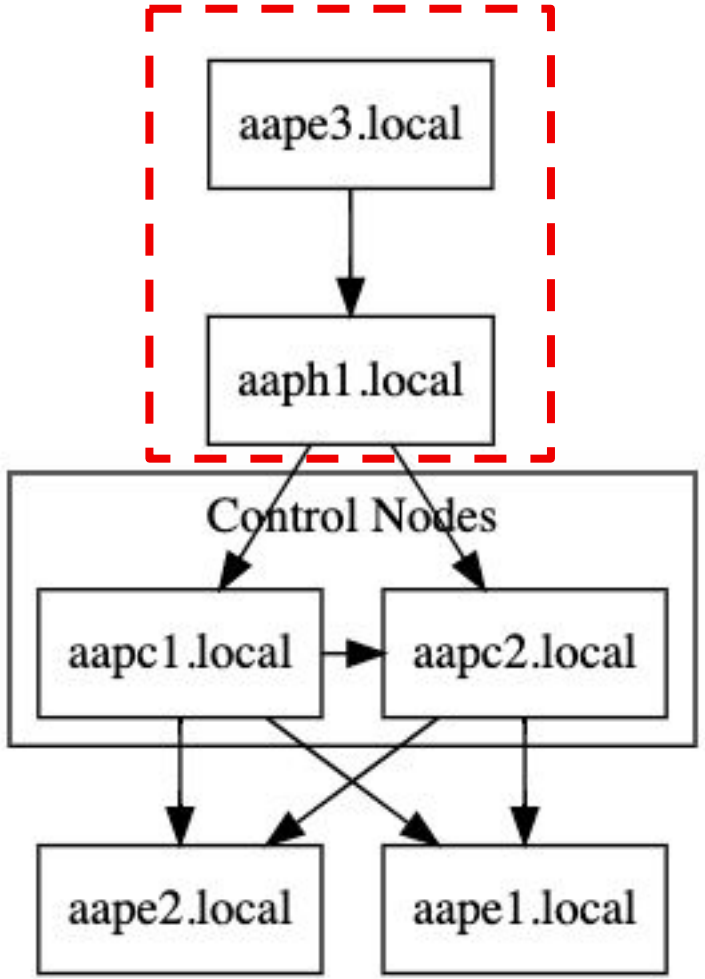






Red Hat

Local and remote/segregated execution

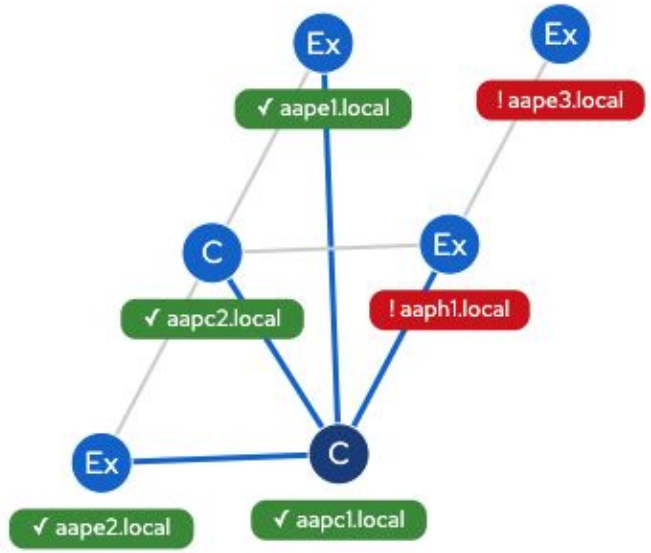


Scenario:

The business adds a:

1. Satellite office/store
2. Needs to automate in a DMZ
3. An outsourcing partner
4. ...

Resilient controllers (control only)
Resilient local execution nodes
Remote execution node reachable via a hop node



Red Hat

Local and remote/segregated execution


```
[automationcontroller]
aapc1.local ansible_connection=local
aapc2.local

[automationcontroller:vars]
node_type=control
peers=instance_group_local

[execution_nodes]
aape1.local
aape2.local
aape3.local peers=hop
aaph1.local

[instance_group_local]
aape1.local
aape2.local

[hop]
aaph1.local peers=automationcontroller

[instance_group_remote]
aape3.local
```

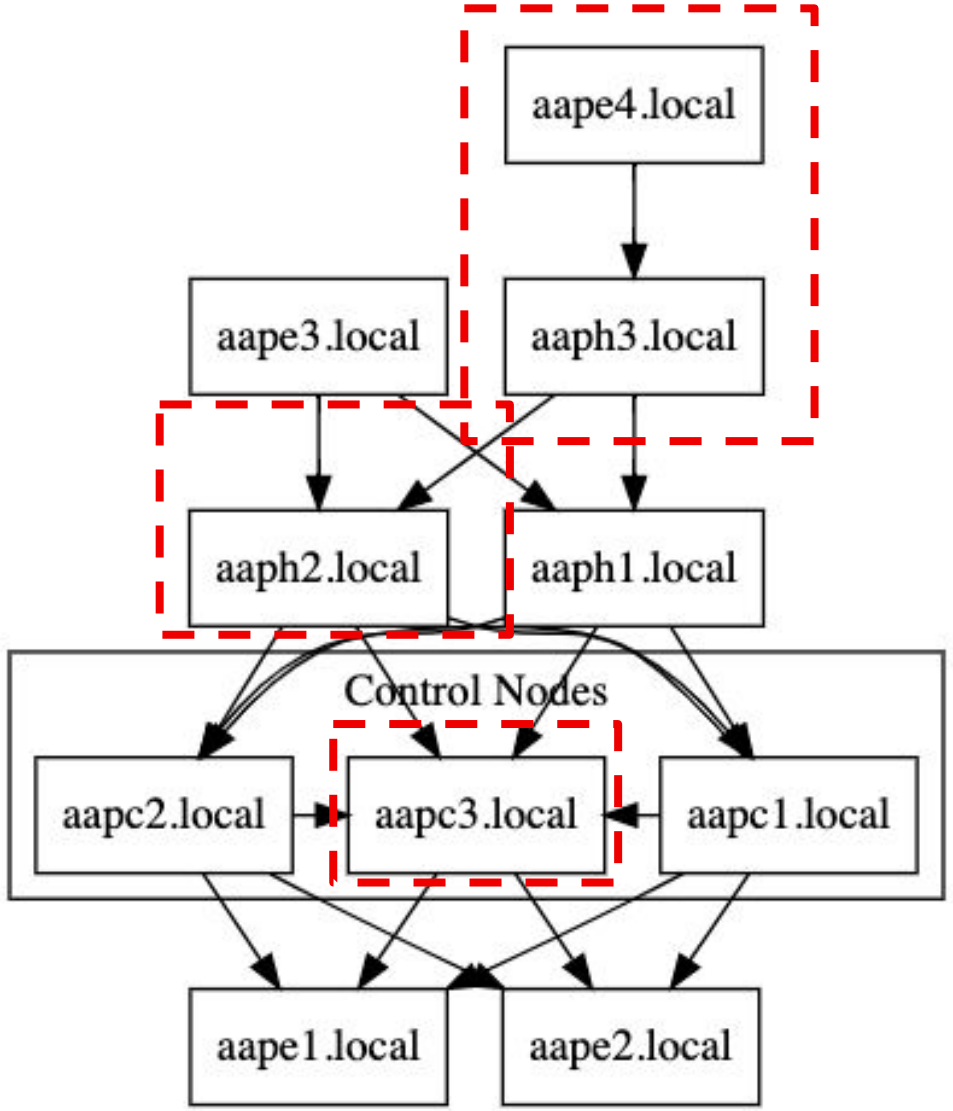
control only nodes (: vars) / auto peered together

peers the controllers to the *local* execution nodes group (ignores hop node)

define **all** execution node types here:
define which exec nodes need a hop
specify which nodes are hops

By using the *instance_group_* prefix this will add these nodes into the UI as an instance group via the installer automatically

Scaling Up and Out

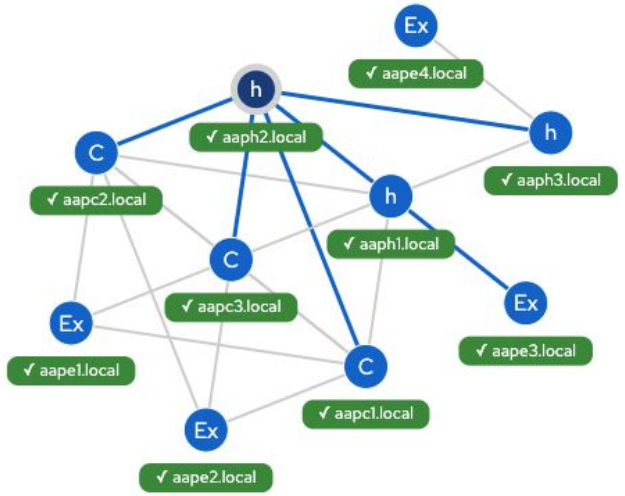


Scenario:

The business adds another:

1. Datacenter in another geo
2. Needs more resilience due to automation imperative
3. ...

Same as before with another controller node for extra capacity and a multi hopped execution node



Red Hat

Scaling Up and Out

1

```
[automationcontroller]
aapc1.local ansible_connection=local
aapc2.local
aapc3.local

[automationcontroller:vars]
node_type=control
peers=instance_group_local
```

control only nodes (: vars) / auto peered together

peers the controllers to the *local* execution nodes group

2

```
[execution_nodes]
aape1.local
aape2.local
aape3.local peers=local_hop
aape4.local peers=remote_multi_hop
aaph1.local node_type=hop
aaph2.local node_type=hop
aaph3.local node_type=hop

[instance_group_local]
aape1.local
aape2.local
```

define **all** execution node types here:
define which exec nodes need a hop
specify which nodes are hops

3

```
[instance_group_multi_hop_remote]
aape4.local

[instance_group_remote]
aape3.local

[local_hop]
aaph1.local
aaph2.local

[local_hop:vars]
peers=automationcontroller

[remote_multi_hop]
aaph3.local peers=local_hop
```

By using the *instance_group_* prefix this will add these nodes into the UI as an instance group via the installer automatically

peer the *local_hop* group back to the *automationcontroller* group
Peer the *remote_hop* group to the *local_hop* group

Red Hat

Designing your mesh

- Do you want to separate the control plane from any execution plane?
- Resilience for local and/or remote execution?
- Work with your network/security policies for best fit

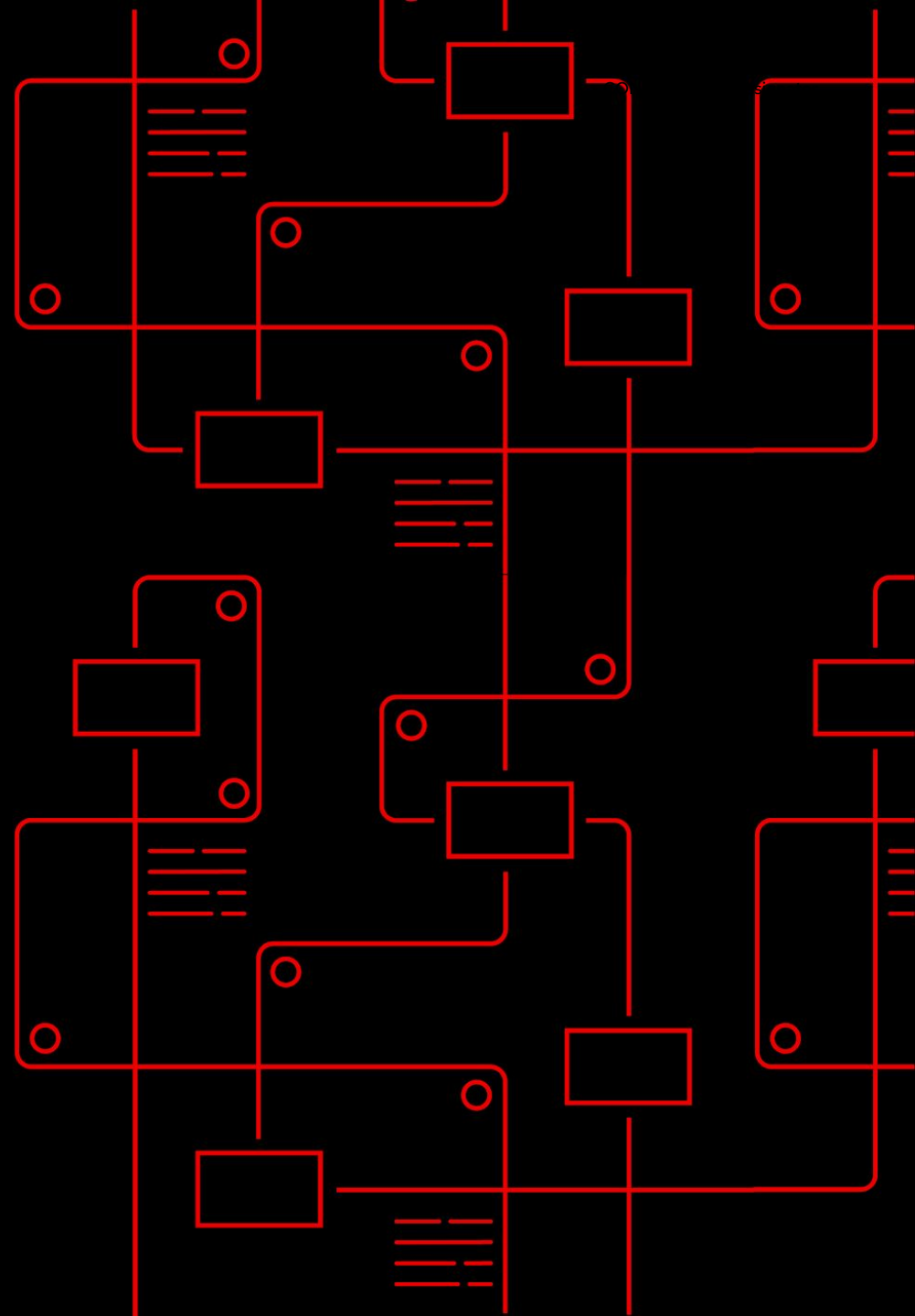
Local Autonomy for Edge Sites

Local autonomy is the ability of an edge site [retail store, manufacturing plant, oil refinery, etc] to continue to operate, possibly in a limited capacity, if the site becomes isolated.

This is an important consideration when architecting your automation platform.

Red Hat

Managing Red Hat and Non-Red Hat Edge Devices



Example: Retail Store

IDENTIAL designator

Red Hat Edge Manager

Red Hat Ansible Automation Platform

Store Network

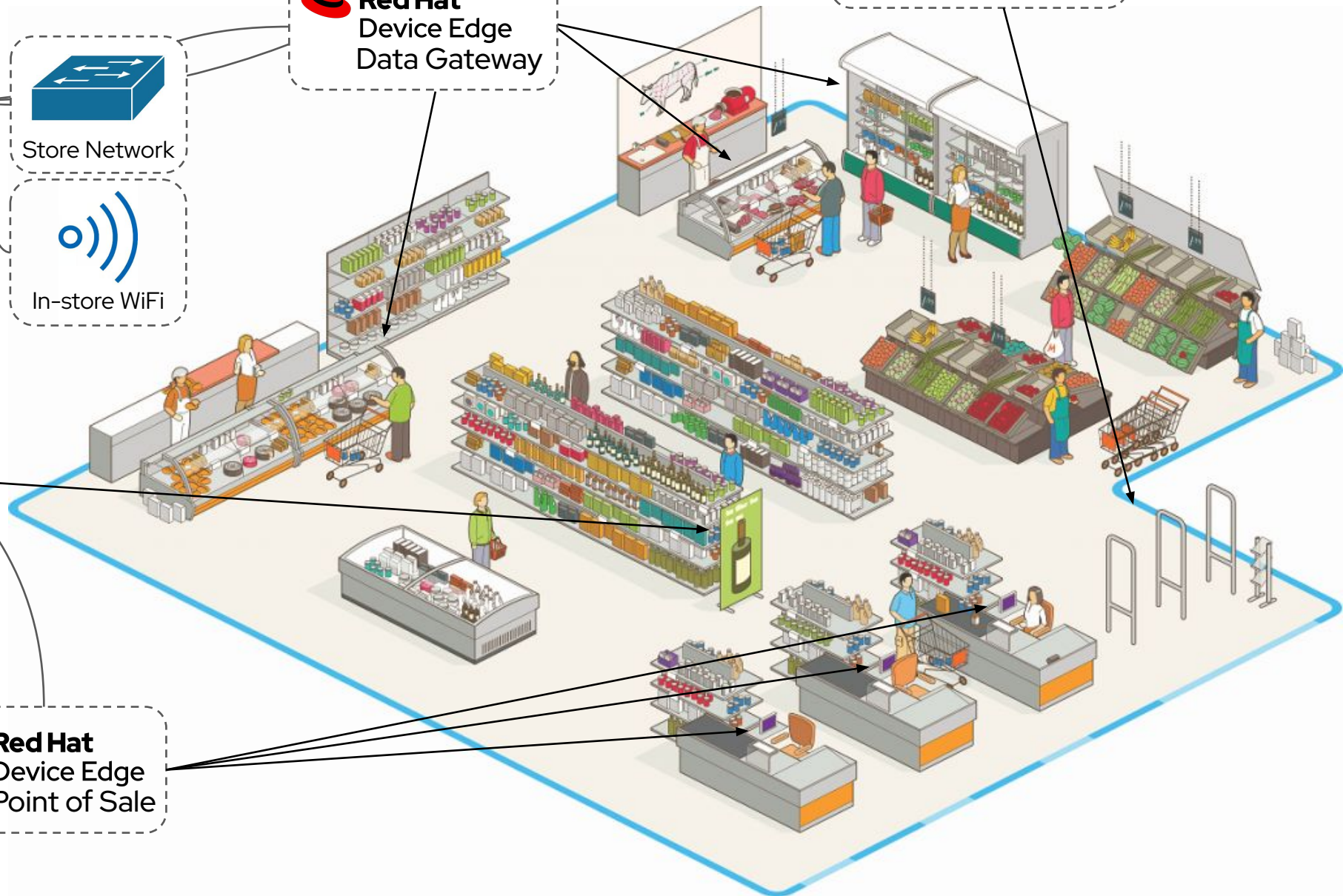
In-store WiFi

Red Hat Device Edge Data Gateway

Red Hat Device Edge Computer Vision

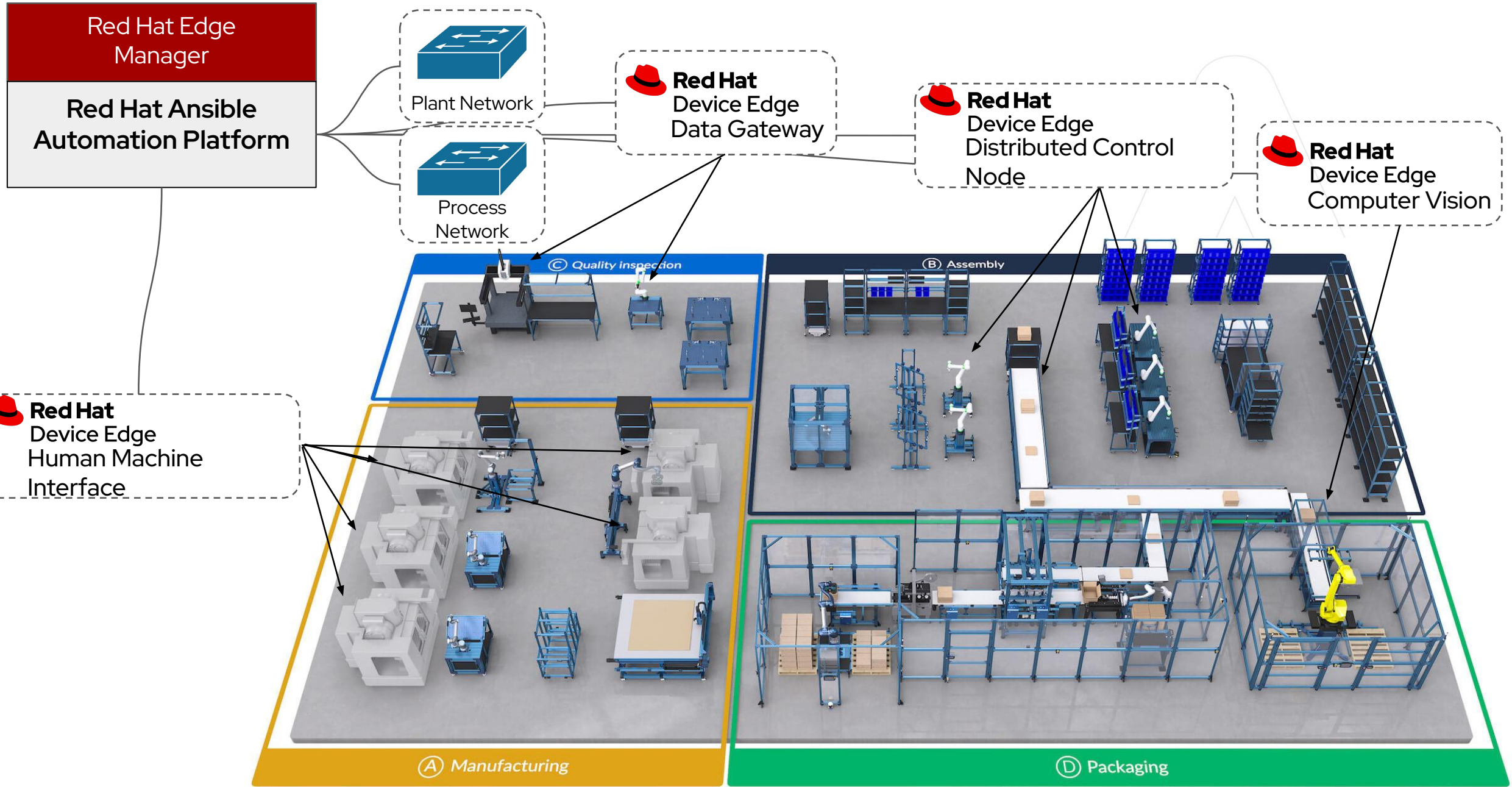
Red Hat Device Edge Intelligent Display

Red Hat Device Edge Point of Sale



Example: Manufacturing Plant

CONFIDENTIAL designator



Thank you

 linkedin.com/company/red-hat

 facebook.com/redhatinc

 youtube.com/user/RedHatVideos

 x.com/RedHat

Red Hat

Hands-On Lab for Image Mode RHEL

https://www.redhat.com/en/introduction-to-image-mode-for-red-hat-enterprise-linux-interactive-lab

Red Hat