Red Hat
Edge

Red Hat Edge Manager

Technical capabilities overview

Luis Arizmendi
Principal Edge Computing Specialist Solution Architect

Red Hat

Agenda

1. Red Hat Edge Manager Introduction
2. Provisioning devices
3. Managing devices
4. Device observability
5. Fleet management

Red Hat Edge Manager Introduction

Red Hat

The Edge technical challenge

Red Hat

Edge Computing market facts on device management

On-premise deployments over SaaS

Large scale requires a different approach

Target personas are different than in DC

Images from unsplash.com by Priscilla Du Preez, Andrew Wulf and Bruce Mars

Red Hat

Edge Computing ~~market facts on device management~~ device management requirements

On-premise deployments over SaaS

- ➔ Simplified deployment at scale
- ➔ Device Zero-Touch-Provisioning
- ➔ Physical and network security
- ➔ Ownership tracking

Large scale requires a different approach

Target personas are different than in DC

Images from unsplash.com by Priscilla Du Preez, Andrew Wulf and Bruce Mars

Red Hat

Edge Computing ~~market facts on device management~~ device management requirements

On-premise deployments over SaaS

- ➔ Simplified deployment at scale
- ➔ Device Zero-Touch-Provisioning
- ➔ Physical and network security
- ➔ Ownership tracking

Large scale requires a different approach

- ➔ Declarative configurations and APIs
- ➔ Compliance policies
- ➔ “Pull” model over “Push” model
- ➔ Device insights and status monitoring

Target personas are different than in DC

Images from unsplash.com by Priscilla Du Preez, Andrew Wulf and Bruce Mars

Red Hat

Edge Computing ~~market facts on device management~~ device management requirements

On-premise deployments over SaaS

- ➔ Simplified deployment at scale
- ➔ Device Zero-Touch-Provisioning
- ➔ Physical and network security
- ➔ Ownership tracking

Large scale requires a different approach

- ➔ Declarative configurations and APIs
- ➔ Compliance policies
- ➔ “Pull” model over “Push” model
- ➔ Device insights and status monitoring

Target personas are different than in DC

- ➔ UI (point-and-click) over CLI
- ➔ Self-Healing and unattended ops
- ➔ Simplified troubleshooting
- ➔ Single-pane-of-glass observability

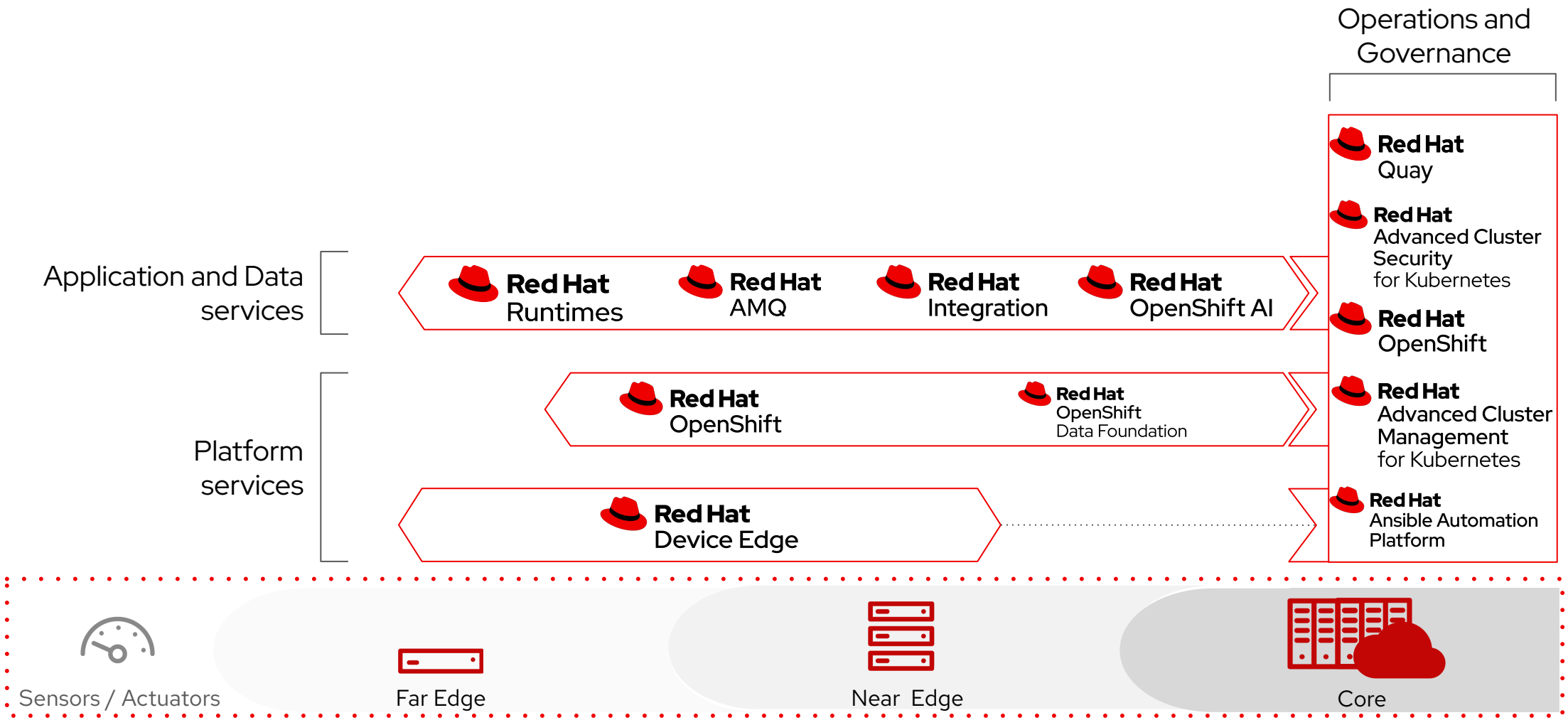
Red Hat

Images from unsplash.com by Priscilla Du Preez, Andrew Wulf and Bruce Mars

Red Hat Edge Manager

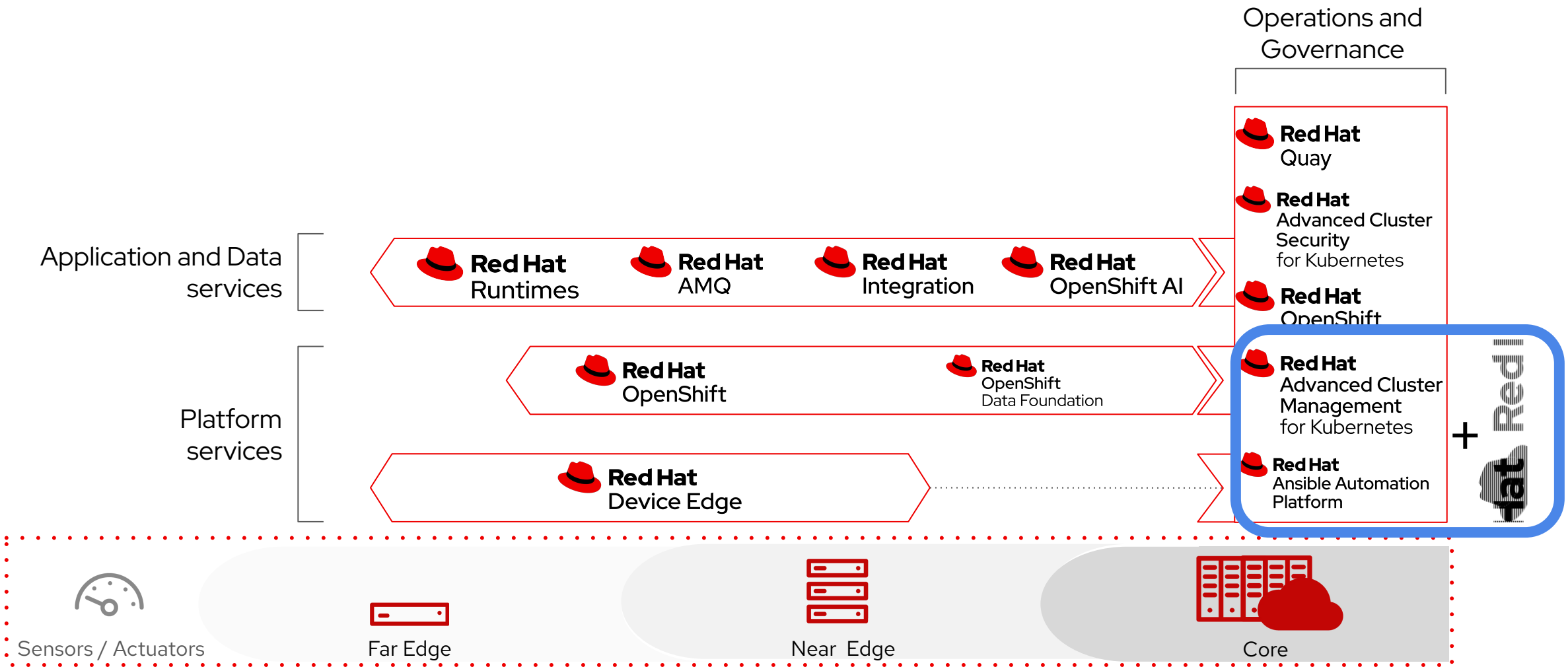
Red Hat

Red Hat Edge portfolio overview

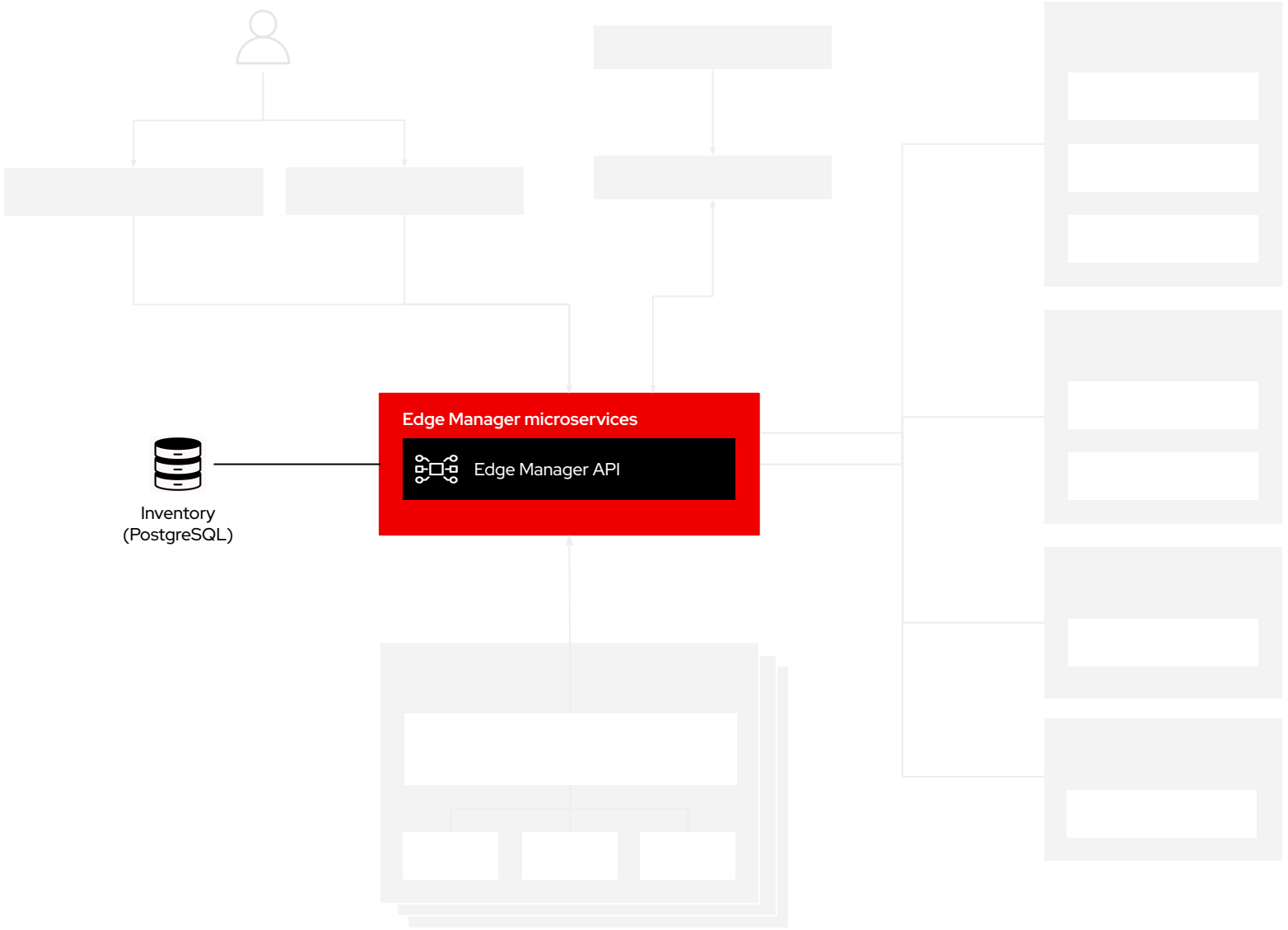


Red Hat

Red Hat Edge portfolio overview



Red Hat Edge Manager architecture



Tech Preview

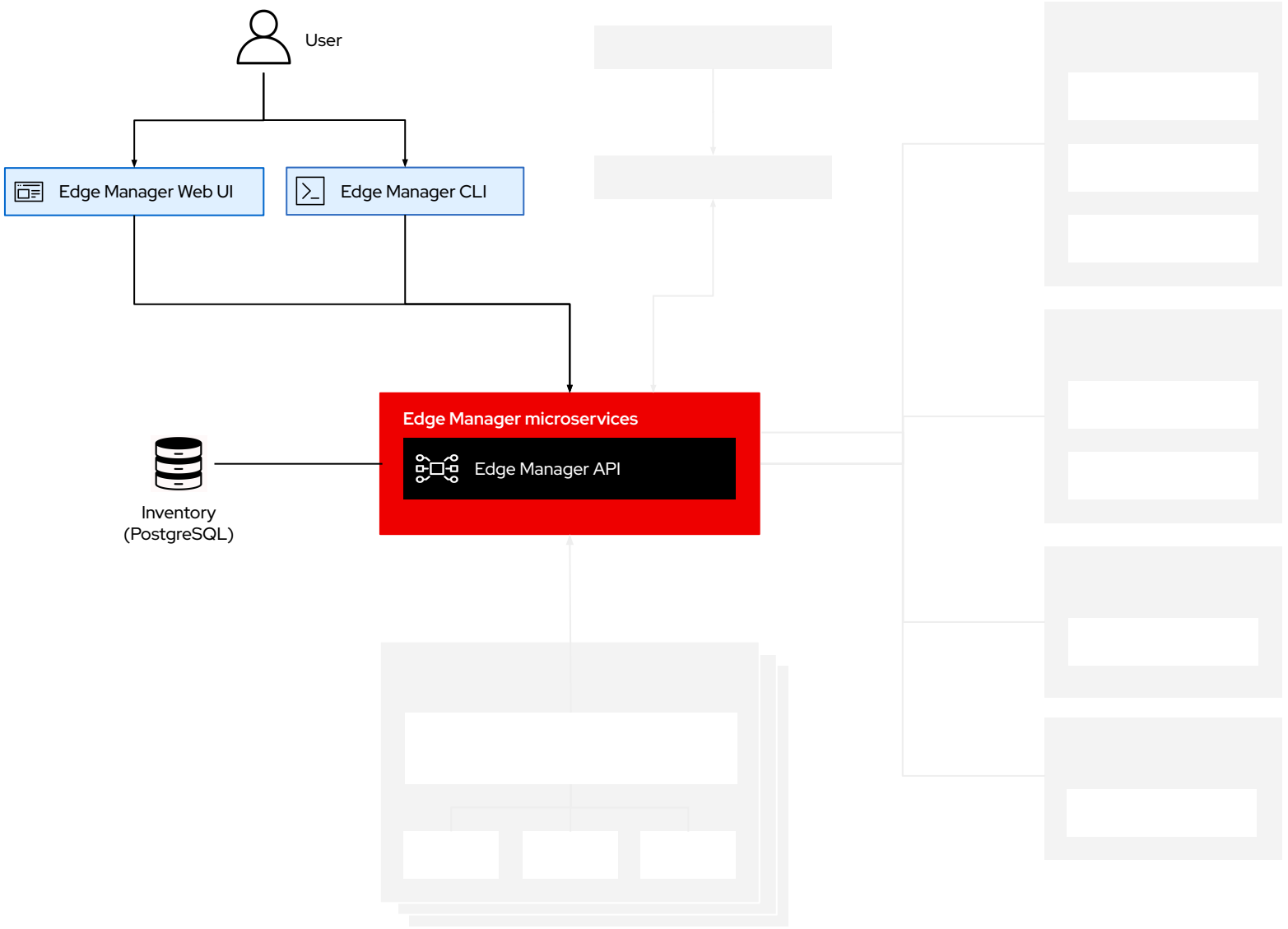
Red Hat Edge Manager (RHEM) microservices that handle enrolling devices, managing inventory, and reporting device status

API Service used for User and Agent communication

PostgreSQL DB that contains Device Inventory (as well as the target device configuration)

Red Hat Edge Manager

Red Hat Edge Manager architecture



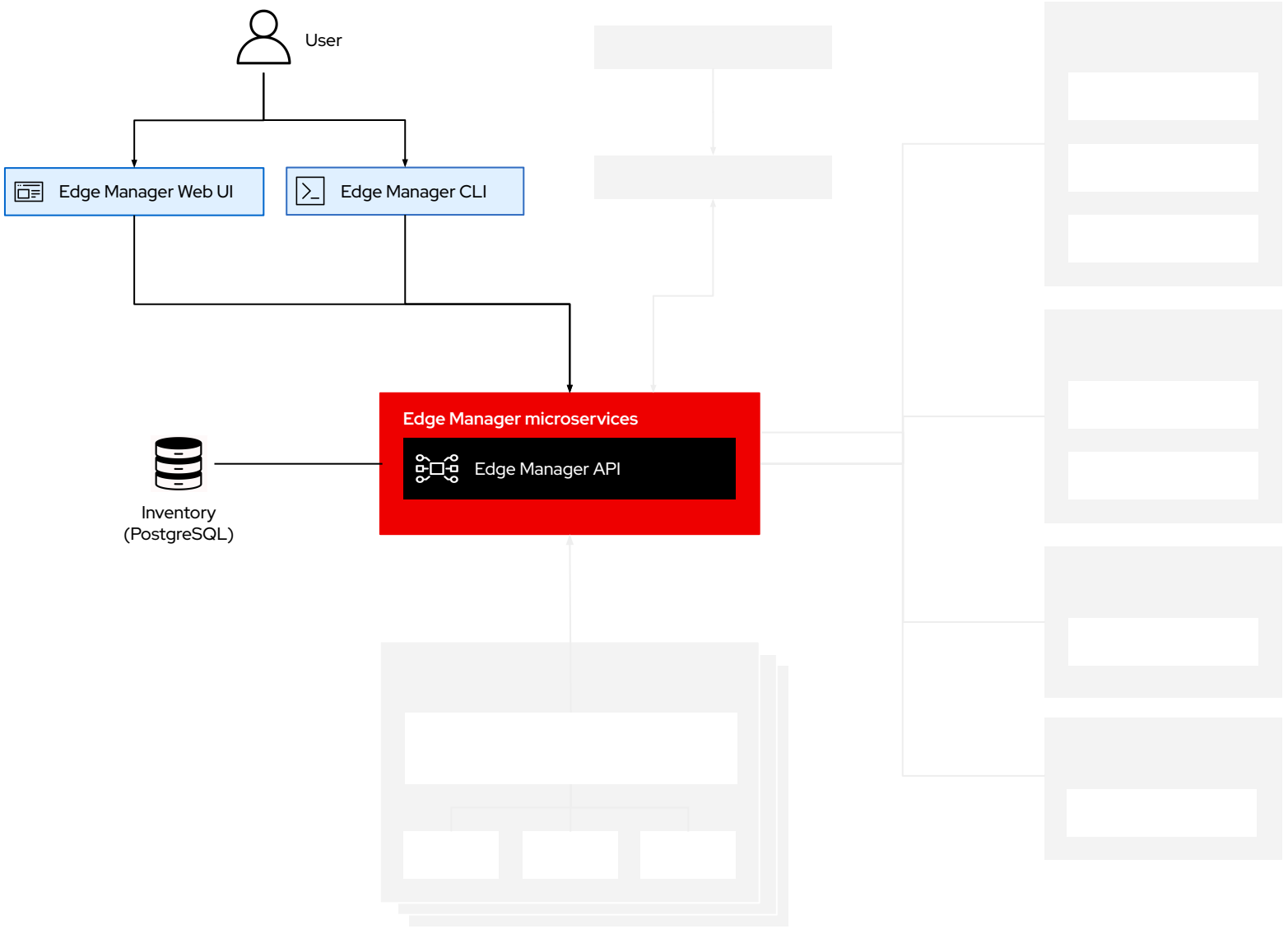
Tech Preview

Web UI: Click-Ops manual interaction

Command-Line Interface (CLI) enables automated interaction

Red Hat Edge Manager

Red Hat Edge Manager architecture



Tech Preview

Command-Line Interface (CLI)

Available commands

`Apply`
Apply a configuration to a resource by file name or stdin

`Approve`
Approve a certificate signing or enrollment request

`Certificate`
Request a new certificate for a device with "certificate request"

`Completion`
Generate autocomplete script

`Console`
Connect a console to the remote device through the server

`CSR-generate`
Generate a CSR resource conifg. Yaml based on a .csr file and optional additional parameters

`Decommission`
Decommission a device

`Delete`
Delete resource by resources or owner

`Deny`
Deny a certificate signing or enrollment request

`Enrollmentconfig`
Get enrollment config for devices

`Get`
Display one or many resources

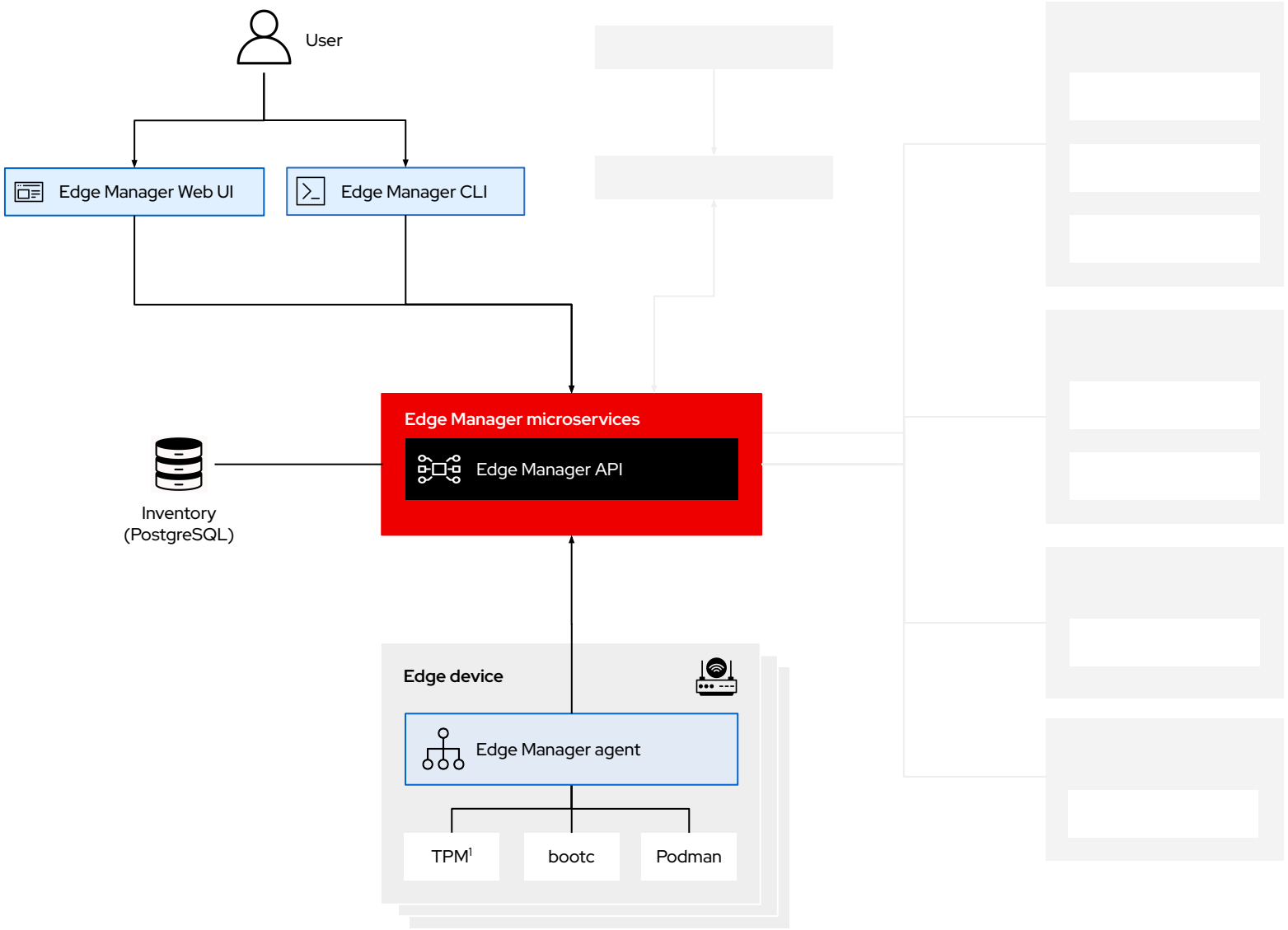
`Help`
Help about my command

`Login`
Login to flight control

`Version`
Print flightctl version information

Red Hat Edge Manager

Red Hat Edge Manager architecture



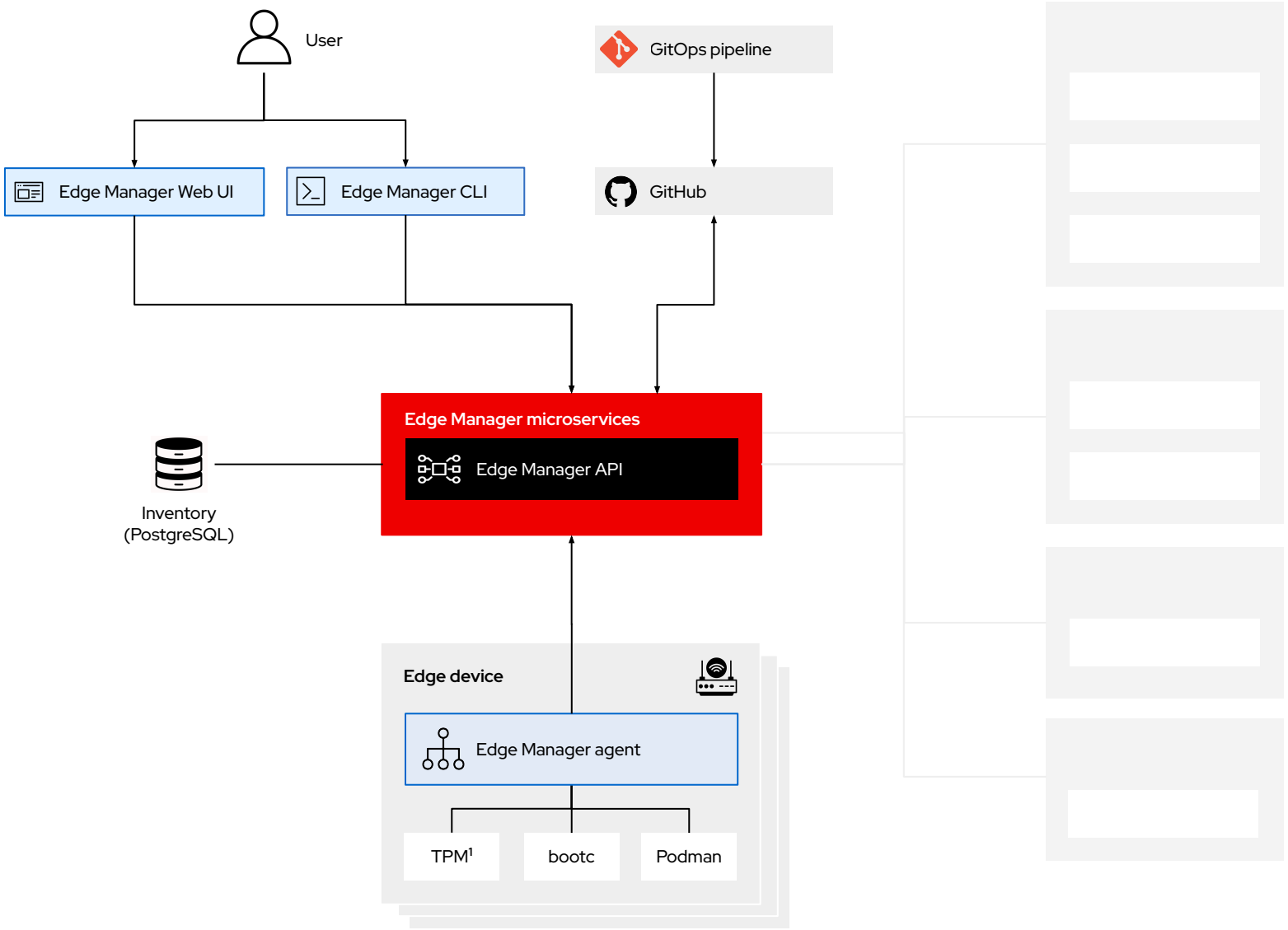
Tech Preview

Edge Manager agent required on each Edge Device

Agent is responsible for Enrollment and periodically checking in with the Edge Manager Service for changes in the device specification

Red Hat Edge Manager

Red Hat Edge Manager architecture



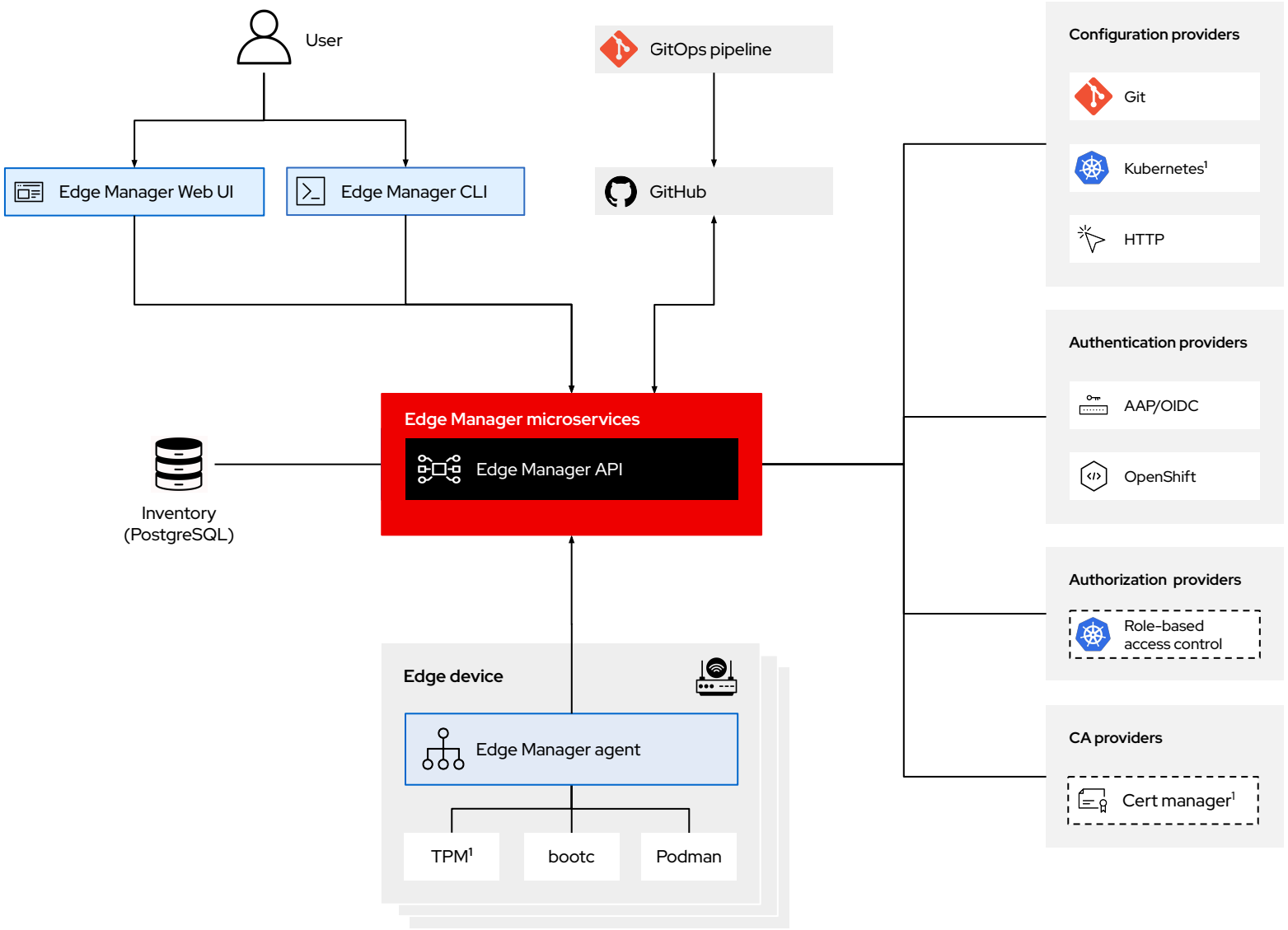
Tech Preview

Modern integration with
GitOps Pipelines

Red Hat
Edge Manager

Red Hat Edge Manager architecture

Tech Preview



Pluggable Architecture for Configuration, Authentication, Authorization, and CA provider integration

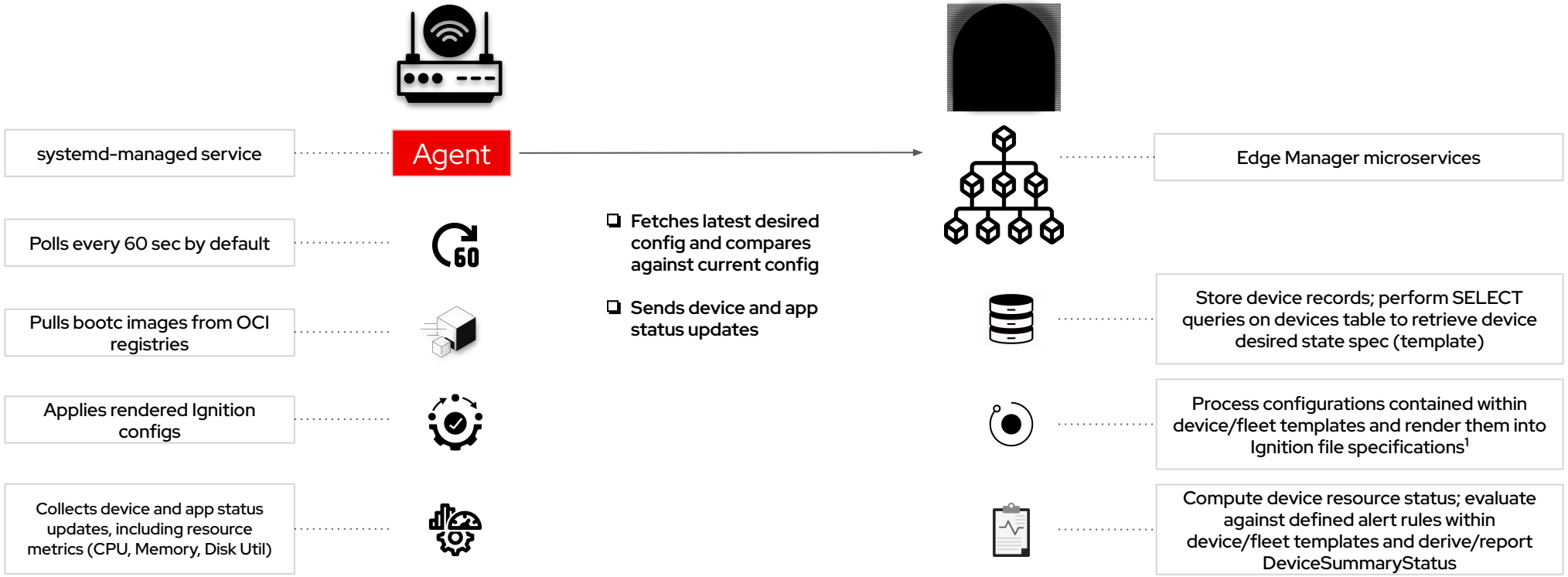
1 RoadMap Items

Red Hat Edge Manager

Red Hat Edge Manager operation

Tech Preview

mTLS communication with the edge manager backend microservices uses an agent-based architecture, primarily operating in pull mode. This design aims for robustness in challenging network environments and strives to be as lightweight (on the device) as possible



1 If using GitOps, transforms content from GitHub/Gitlab repos as part of Ignition spec

Red Hat Edge Manager

Red Hat Edge Manager deployment options

Tech Preview

Edge Manager + Ansible Automation Platform

- **Ansible Automation Platform** installed and utilized for Edge Manager OAUTH
- **Edge Manager** accessible via AAP Application Links left-hand navigation menu

Documentation

Edge Manager + Advanced Cluster Management

- **Advanced Cluster Management** Operator deployed on OpenShift, delivering Edge Manager microservices
- **Edge Manager** accessible via feature flag unlock

Documentation

Please Note
The commercial model for Edge Manager (PnP) at GA is still being formulated

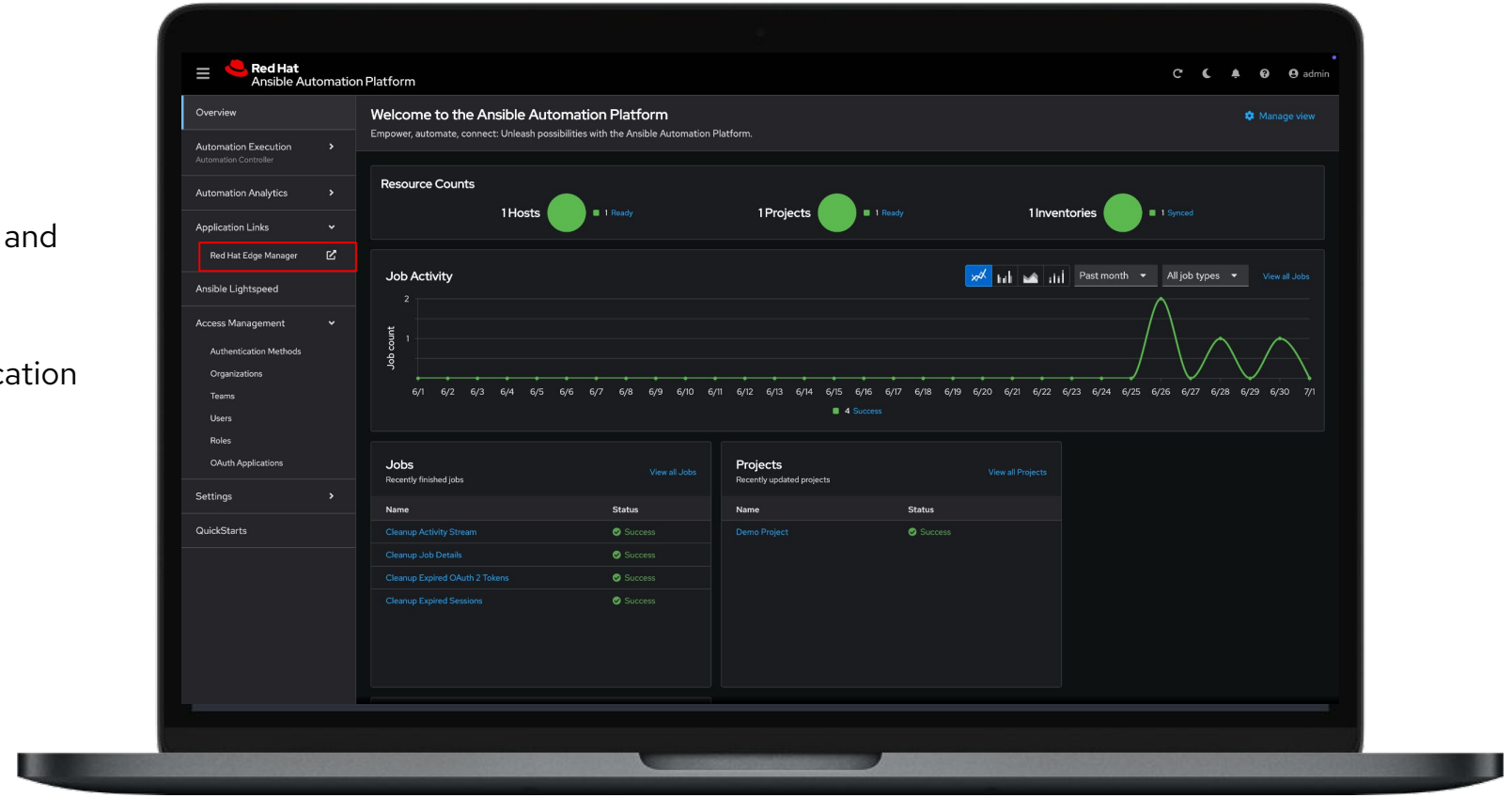
The models discussed in this section (RHEM+AAP and RHEM+ACM) could change and/or evolve further prior to GA

Tech Preview

Edge Manager + Ansible Automation Platform

Ansible Automation Platform installed and utilized for Edge Manager OAUTH

Edge Manager accessible via AAP Application Links left-hand navigation menu



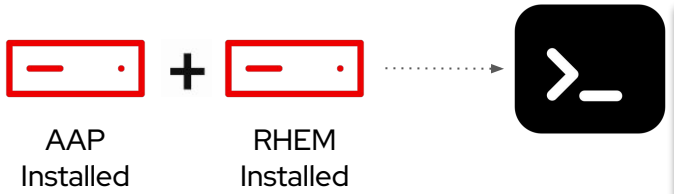
DEMO

Red Hat Edge Manager

Tech Preview

Edge Manager + Ansible Automation Platform

Edge Manager Microservices on Edge Manager server (running as Podman Quadlets)



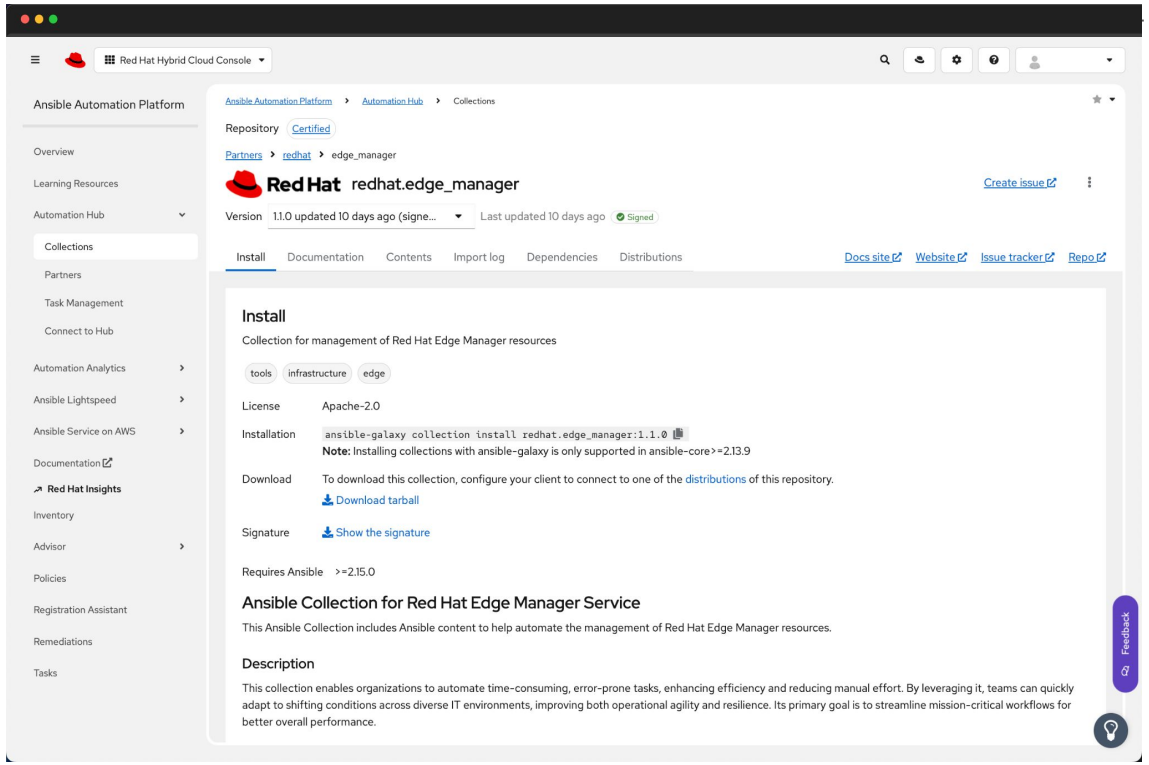

```
[root@rhem-01 ~]# systemctl list-units flightctl-*.service
  UNIT                                LOAD   ACTIVE SUB     DESCRIPTION
  flightctl-api-init.service          loaded active exited  flightctl-api-init.service
  flightctl-api.service               loaded active running Flight Control API server
  flightctl-cli-artifacts-init.service loaded active exited  flightctl-cli-artifacts-init.service
  flightctl-cli-artifacts.service     loaded active running Flight Control CLI Artifacts service
  flightctl-db.service                loaded active running PostgreSQL Database for Flightctl
  flightctl-kv.service                loaded active running Flight Control Key Value service
  flightctl-network.service           loaded active exited  flightctl-network.service
  flightctl-periodic.service          loaded active running Flight Control Periodic service
  flightctl-ui-init.service           loaded active exited  flightctl-ui-init.service
  flightctl-ui.service                loaded active running Flight Control UI
  flightctl-worker.service            loaded active running Flight Control Worker service

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.
11 loaded units listed. Pass --all to see loaded but inactive units, too.
To show all installed unit files use 'systemctl list-unit-files'.
```

Tech Preview

Edge Manager + Ansible Automation Platform

Ansible Automation Platform Edge Manager collection




```yaml
---
 - name: Create a new device
   redhat.edge_manager.flightctl_resource:
     kind: Device
     name: "test-ansible-device-2"
     resource_definition: "{{ lookup('file', 'device.yml') | from_yaml }}"

 - name: Get information about a specific device
   redhat.edge_manager.flightctl_resource_info:
     kind: Device
     name: "test-ansible-device"

 - name: Get all devices
   redhat.edge_manager.flightctl_resource_info:
     kind: Device

 - name: Get all fleets
   redhat.edge_manager.flightctl_resource_info:
     kind: Fleet
```

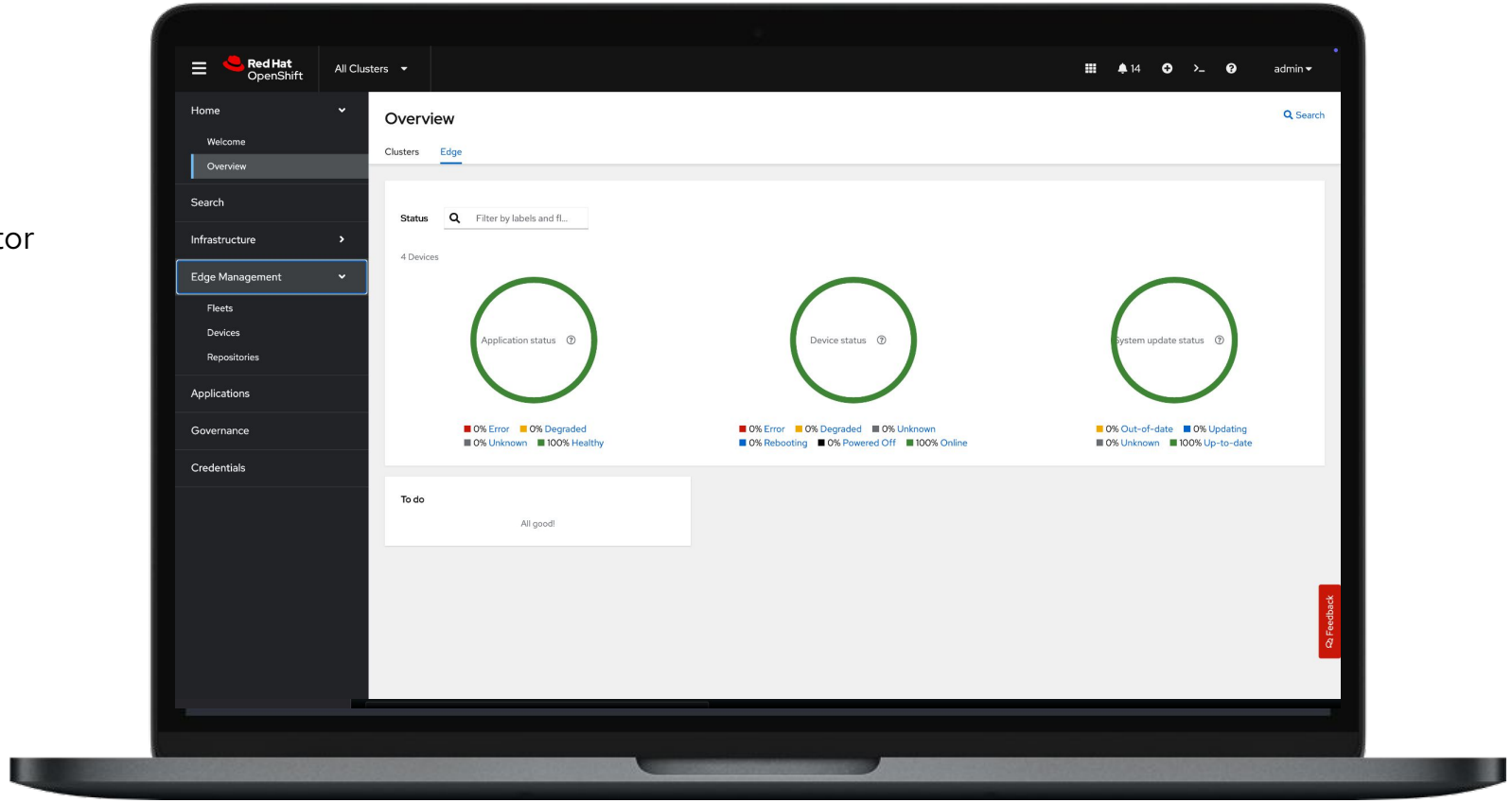
*sample list

Red Hat Edge Manager

Tech Preview

Edge Manager + Advanced Cluster Management

- **Advanced Cluster Management** Operator deployed on OpenShift, delivering Edge Manager microservices
- **Edge Manager** accessible via feature flag unlock



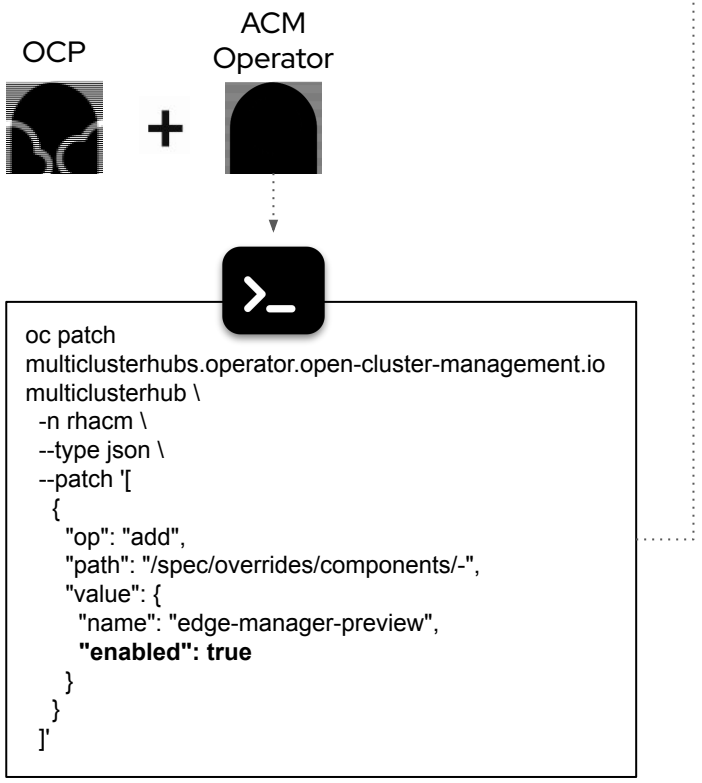
DEMO

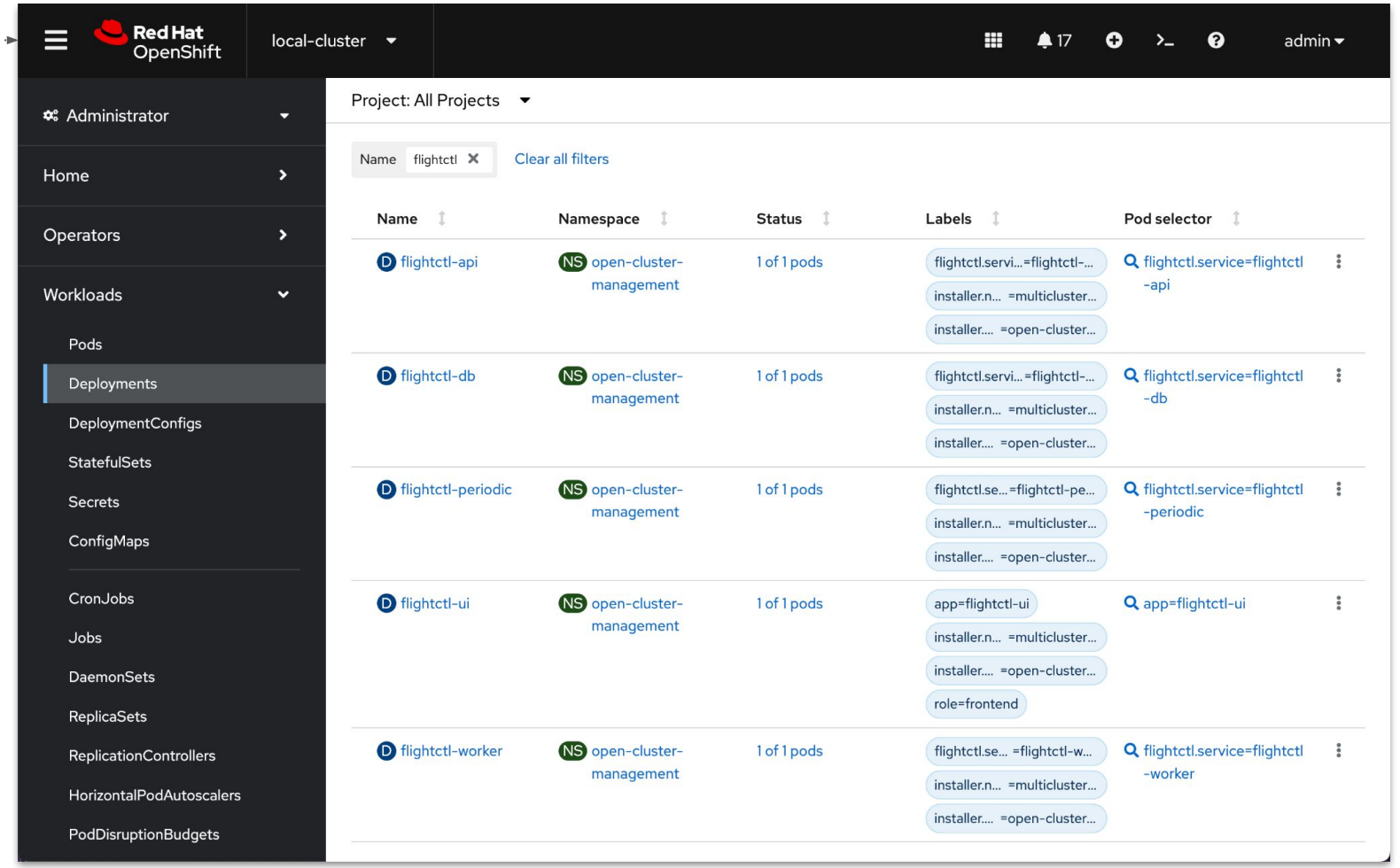
Red Hat Edge Manager

Tech Preview

Edge Manager + Advanced Cluster Management

Edge Manager Microservices on OpenShift Container Platform (running within Pods)



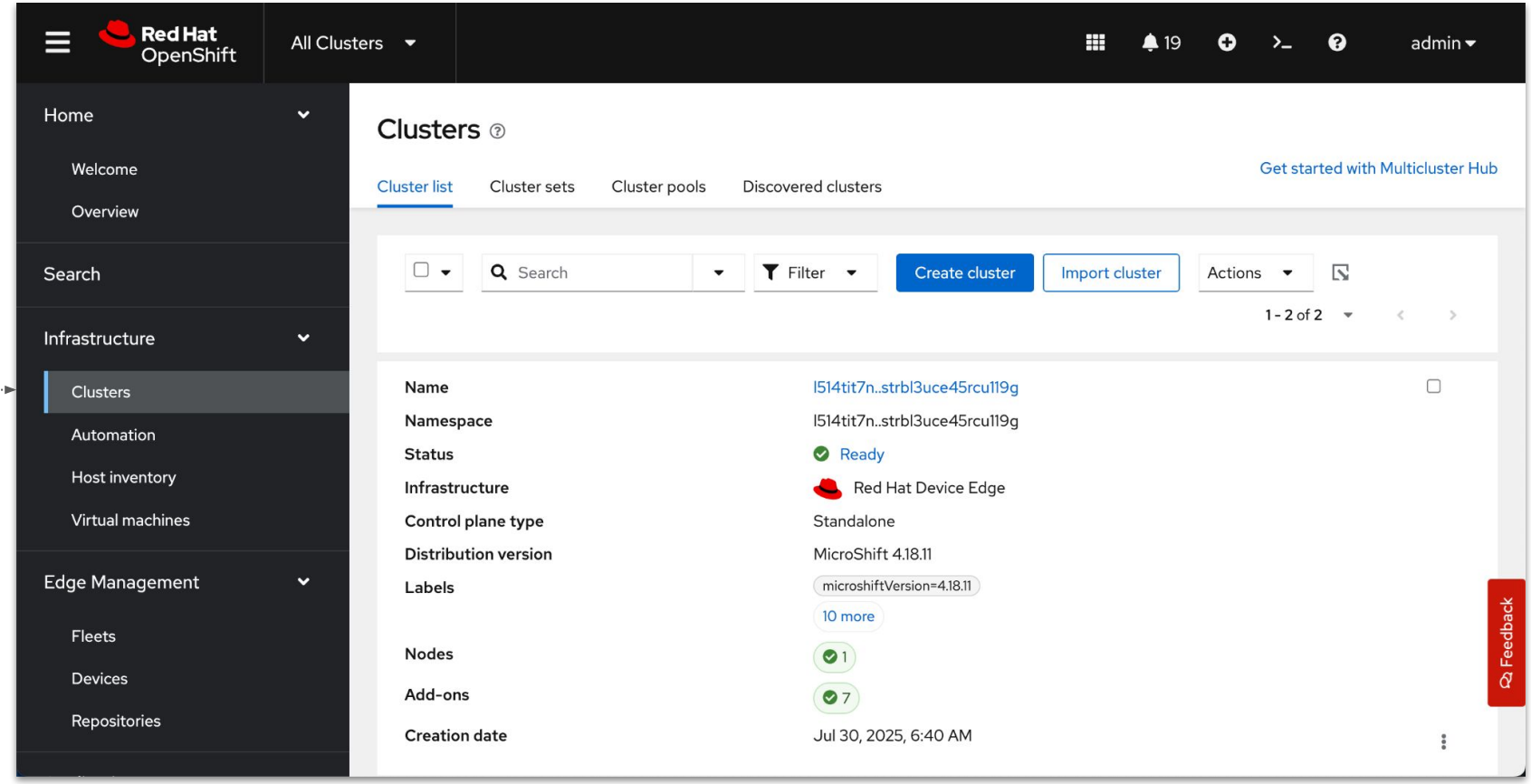


Tech Preview

Edge Manager + Advanced Cluster Management

Automation registration of edge device MicroShift clusters

Edge Device managed by Edge Manager running MicroShift



Red Hat Edge Manager

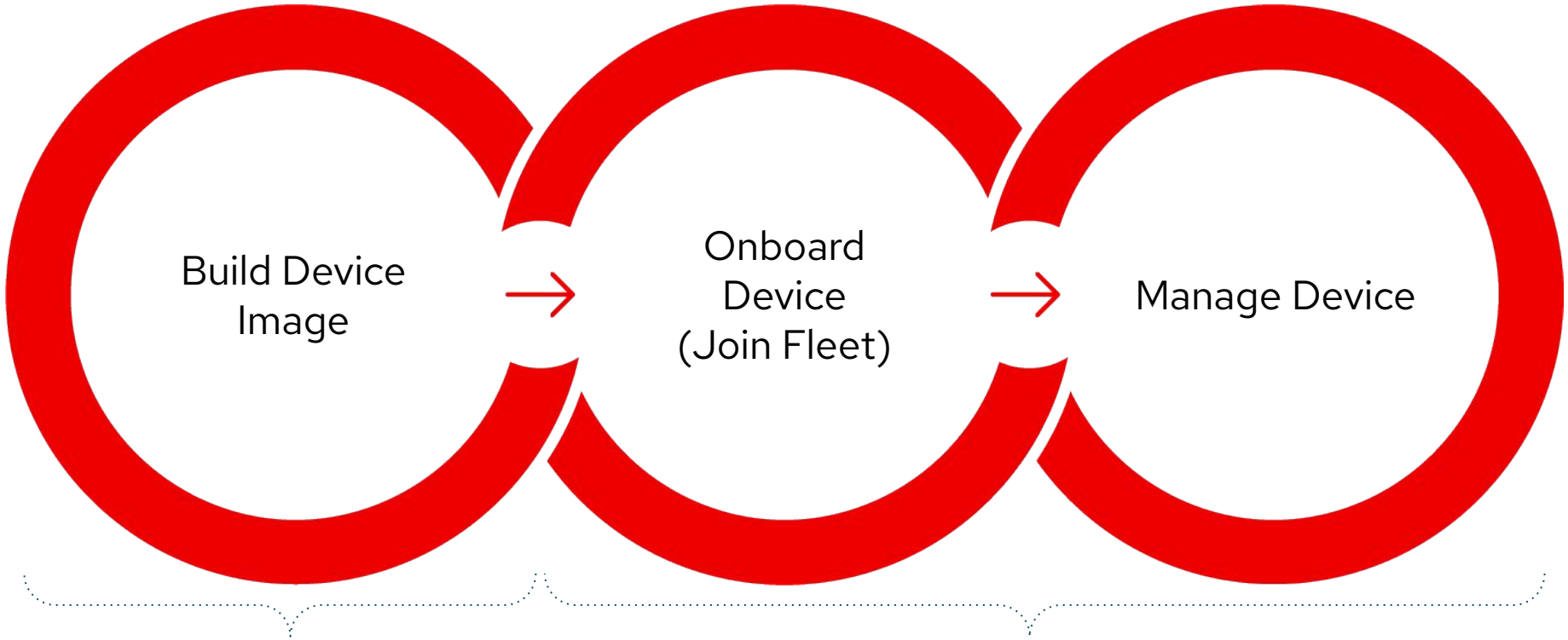
Provisioning Devices

Building a RHEL image

(with the RHEM Agent)

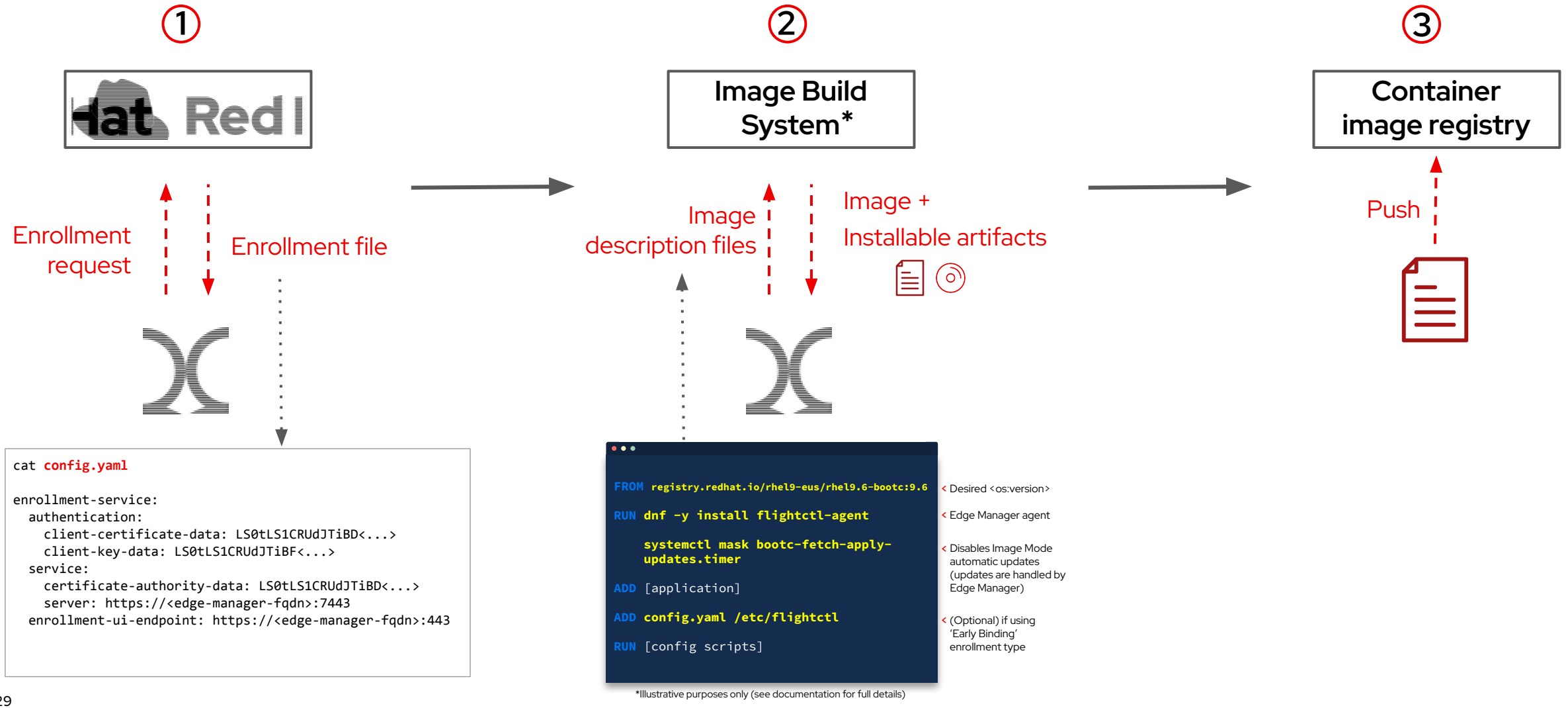
Red Hat

Device onboarding simplified workflow



Red Hat Edge Manager

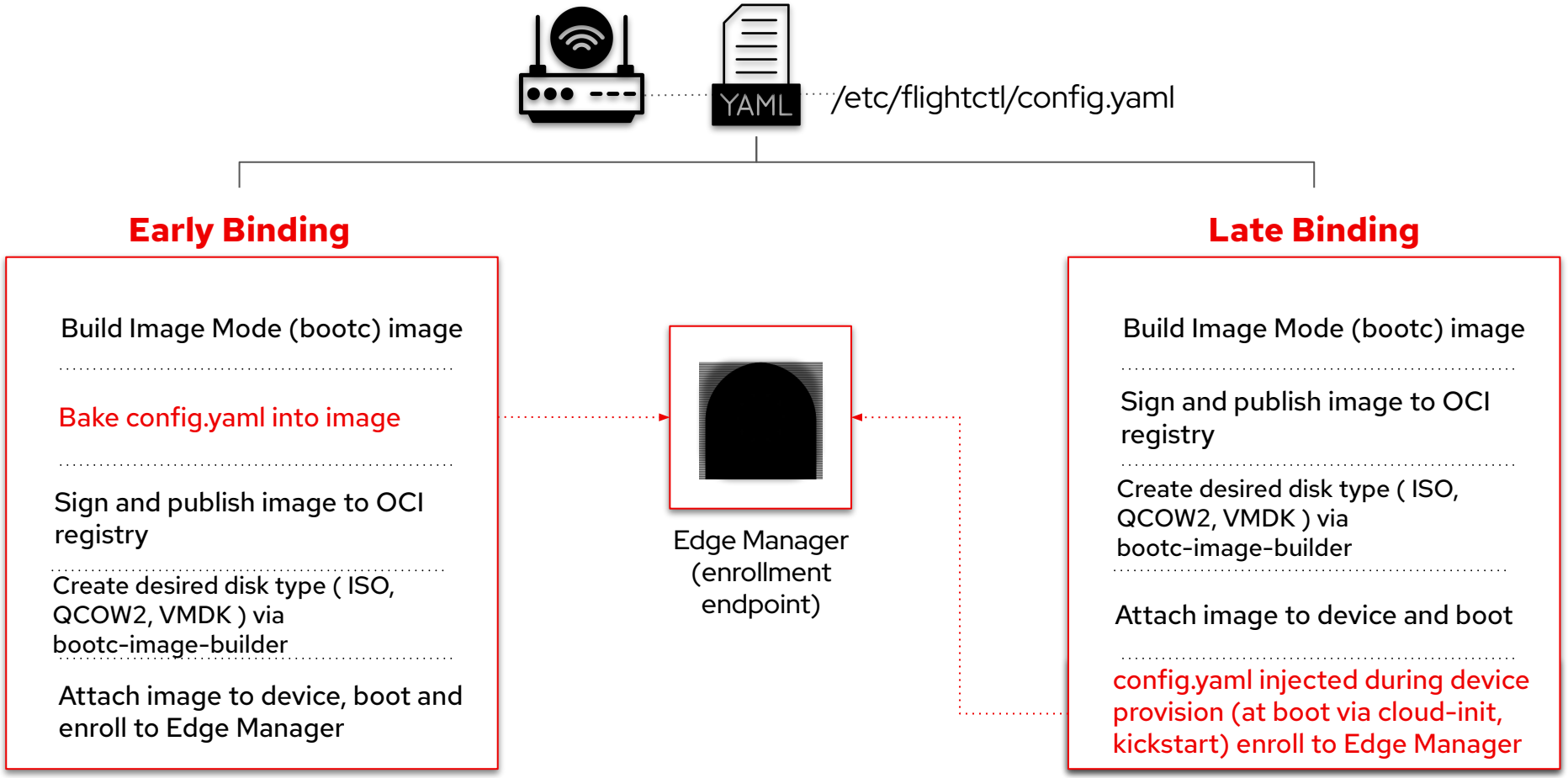
Building a RHEL Image with RHEM agent



* Build using standard container image tooling. Some examples here: https://github.com/luisarizmendi/bootc-build-scenarios

Red Hat

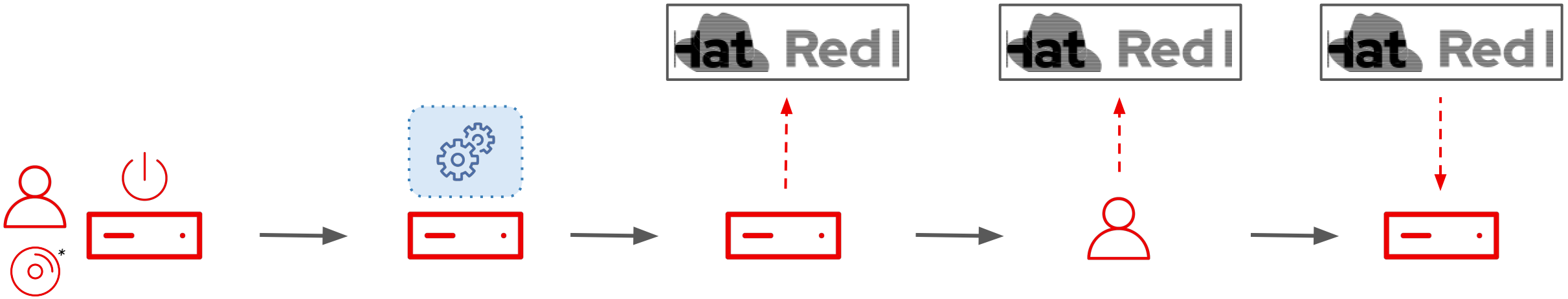
Building a RHEL Image with RHEM agent - Options



Device enrollment

Red Hat

Onboarding process



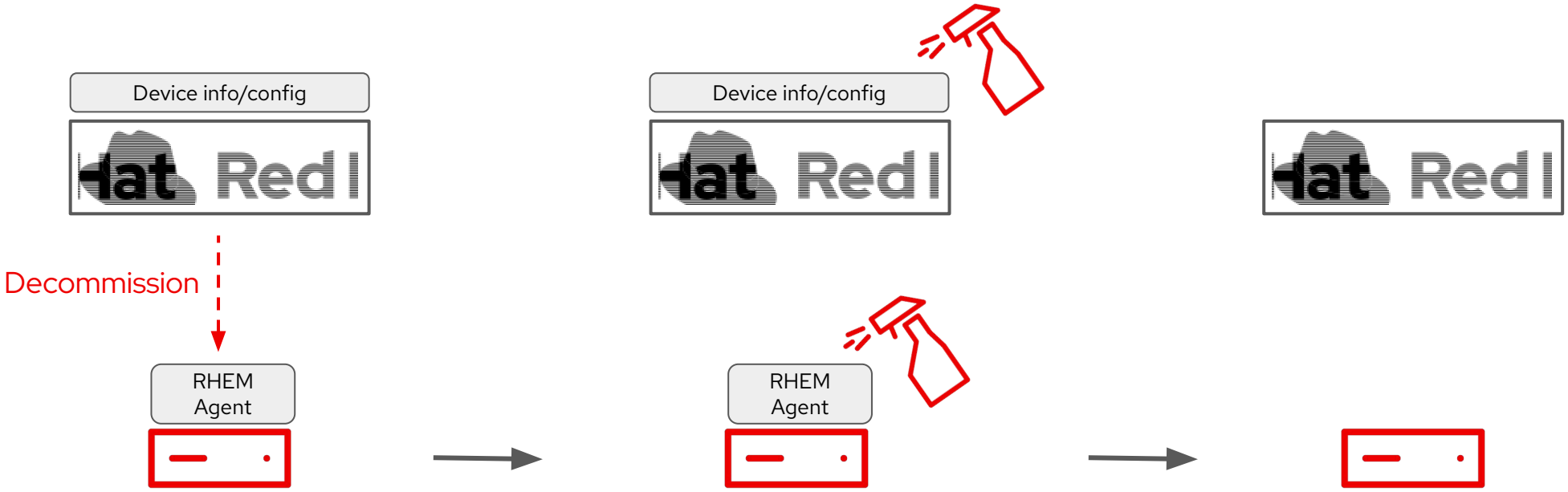
1. Turn-on the device and use installation method
2. RHEL is installed with all packages included
3. Device registers in RHEM (mTLS)
4. Admin approves device
5. RHEM performs configuration

Red Hat

Device decommissioning

Red Hat

Device decommissioning



Red Hat

Managing Devices

Device configurations

Embedding vs Runtime configuration

Prefer adding configuration and APPs to the OS image itself at build-time

so they get tested, distributed, and updated together ("lifecycle-bound").

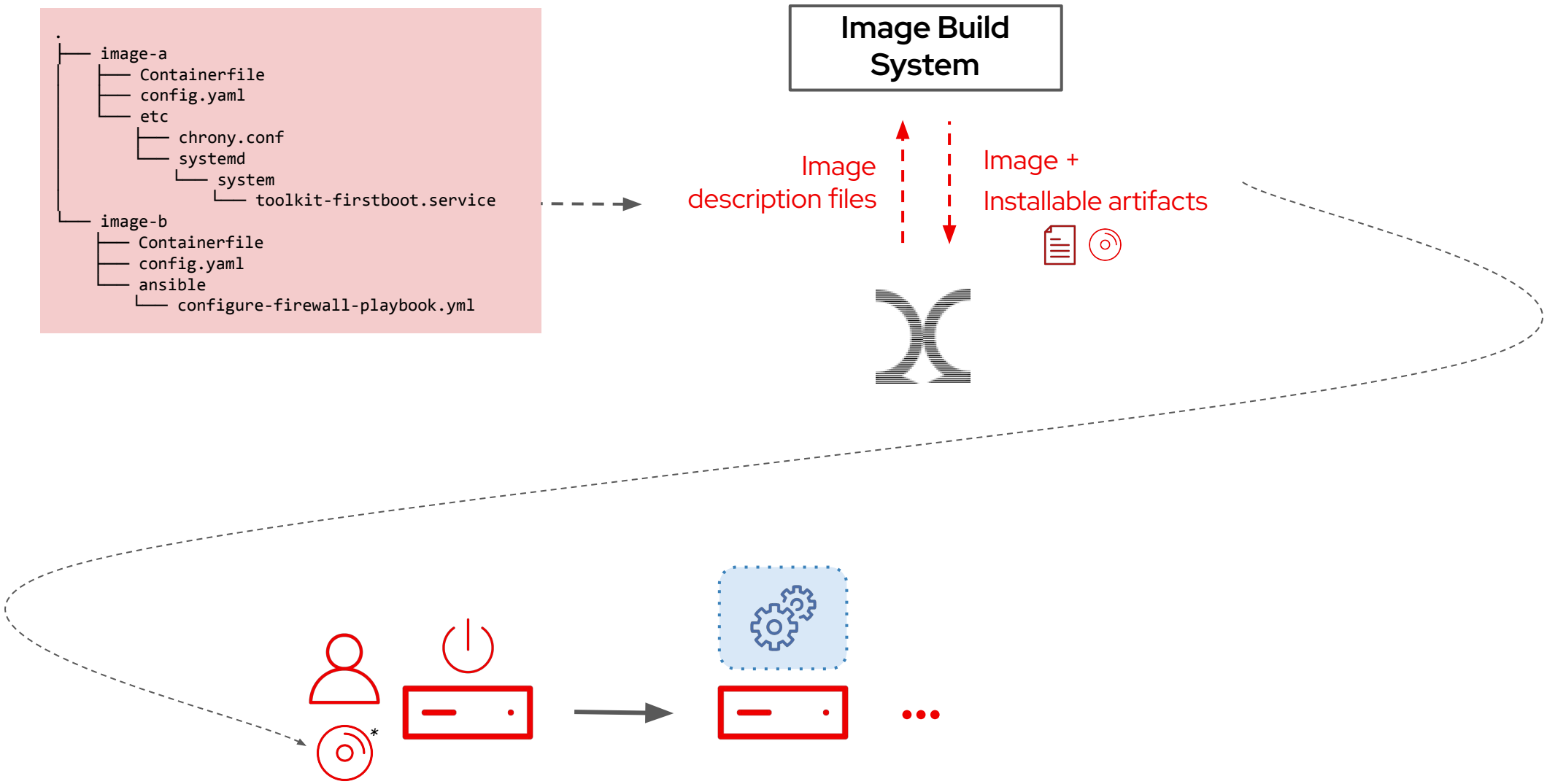
When this is not feasible or desirable, use RHEM to dynamically configure at runtime

Examples:

- Configuration that is deployment- or site-specific (e.g. site-specific network credential)
- Secrets that are not secure to distribute via the image
- APPs that need to be added, updated, or deleted without reboot or on a faster cadence than the OS

Embedding OS configuration in the RHEL image

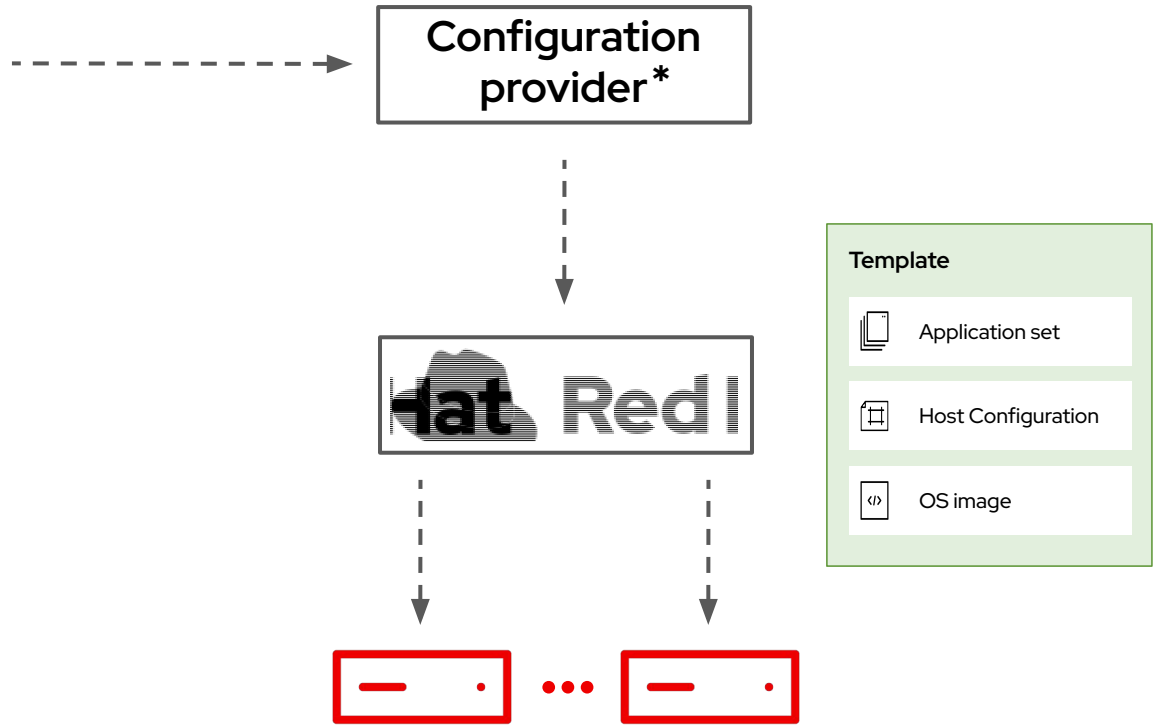
```
.
├── image-a
│   ├── Containerfile
│   ├── config.yaml
│   └── etc
│       ├── chrony.conf
│       └── systemd
│           └── system
│               └── toolkit-firstboot.service
└── image-b
    ├── Containerfile
    ├── config.yaml
    └── ansible
        └── configure-firewall-playbook.yml
```



Red Hat

Managing OS configuration with RHEM

```
.
├── site-a
│   └── etc
│       ├── chrony.conf
│       └── NetworkManager
│           └── system-connections
│               └── wifi-access.nmconnection
└── site-b
    └── etc
        ├── chrony.conf
        └── NetworkManager
            └── system-connections
                └── wifi-access.nmconnection
```



* RHEM currently supports the following configuration providers:

- **Git Config Provider:** Fetches device configuration files from a Git repository.
- **Kubernetes Secret Provider:** Fetches a Secret from a Kubernetes cluster and writes its content to the device's file system.
- **HTTP Config Provider:** Fetches device configuration files from an HTTP(S) endpoint.
- **Inline Config Provider:** Allows specifying device configuration files inline in the device manifest without querying external systems.

Red Hat

Applications

Managing applications

```
apiVersion: flightctl.io/v1alpha1
kind: Device
metadata:
  name: some_device_name
[...]
spec:
  applications:
    - name: my-inline
      appType: compose
      inline:
        - path: docker-compose.yaml
          content: |
            version: "3.8"
            services:
              service1:
                image: quay.io/flightctl-tests/alpine:v1
                command: ["sleep", "infinity"]
                volumes:
                  - my-data:/data
            volumes:
              my-data:
                external: true
      volumes:
        - name: my-data
          image:
            reference: quay.io/flightctl-tests/models/gpt2
            pullPolicy: IfNotPresent
```

Inline

```
apiVersion: flightctl.io/v1alpha1
kind: Device
metadata:
  name: some_device_name
spec:
[...]
  applications:
  - name: wordpress
    image: quay.io/flightctl-demos/wordpress-app:latest
    envVars:
      WORDPRESS_DB_HOST: "mysql"
      WORDPRESS_DB_USER: "user"
      WORDPRESS_DB_PASSWORD: "password"
[...]
```

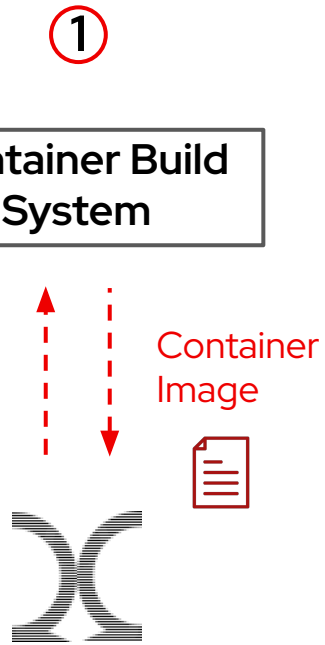
OCI Registry

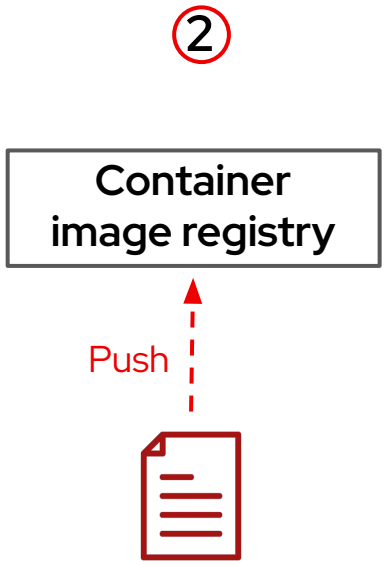
Managing applications using OCI registry

```
FROM scratch

COPY podman-compose.yaml /podman-compose.yaml

# required
LABEL appType="compose"
```

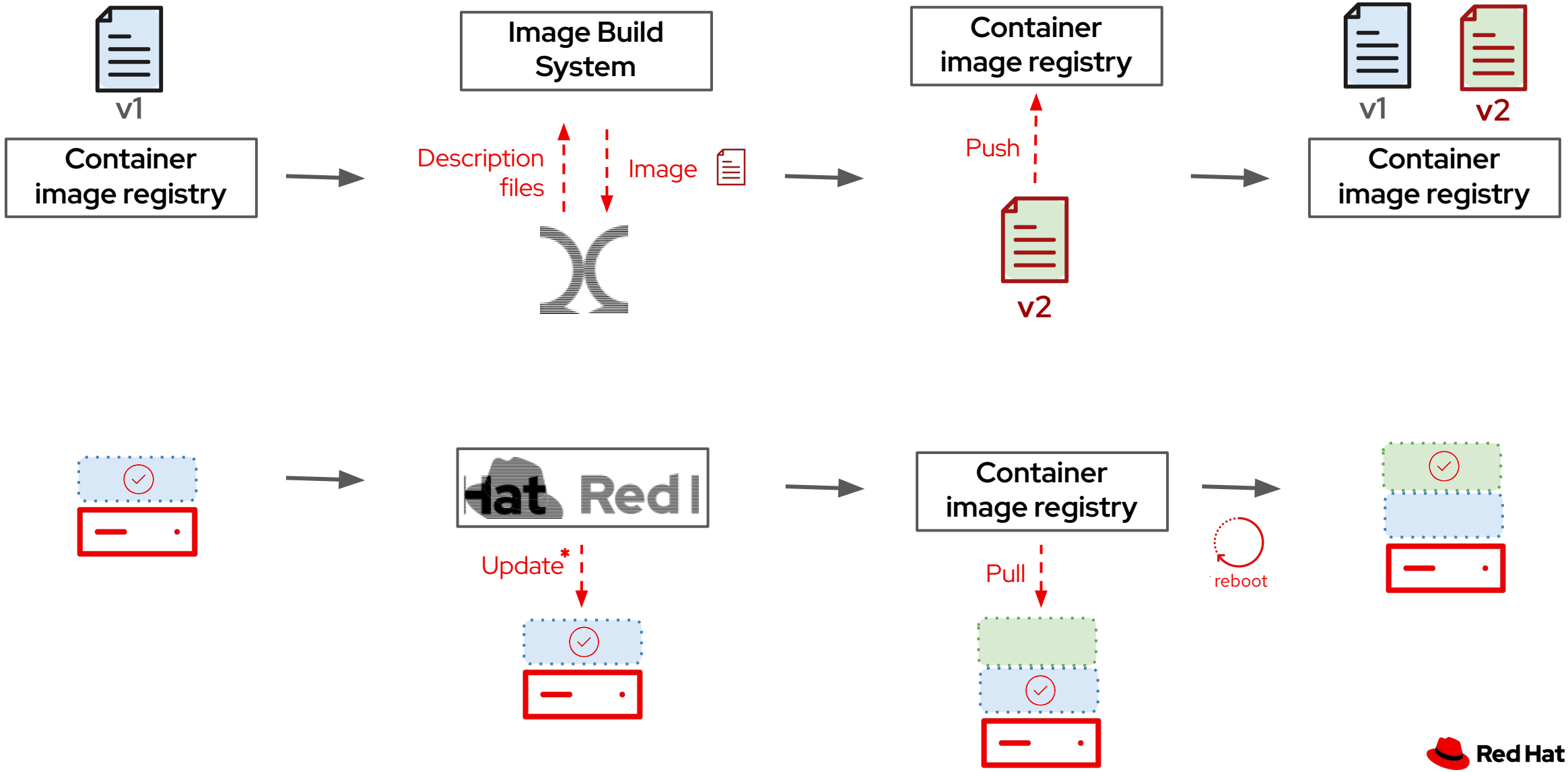




Red Hat

Device upgrade

Upgrade process

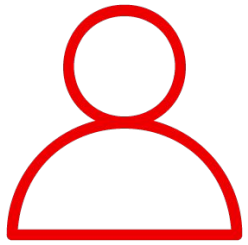


* Automatic or manual

Red Hat

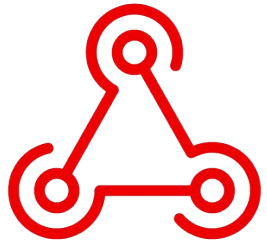
Configuration triggers

Configuration change triggering



Manual

Scheduling*



Hooks**

* Time-based scheduling for update and download operations (eg. download update artifacts such as OS image layers)
** Agent run user-defined commands at specific points in the device's lifecycle (before/after updating or before/after rebooting)

Device Observability

Red Hat

Alerts and monitoring

Device and fleet status

Status States	Application Status	Device Status	System Update (Status States)
Online	No errors or degradations reported for any of the configuration Applications	System Resources (CPU, Memory, Disk Util) are Healthy	Up-to-Date: Device has successfully applied the latest device/fleet specification (in conformance)
Degraded	Applications are in a transient state (like Preparing or Starting)	Resources (CPU, Memory, Disk Util) reaching Warning levels	Out-of-Date: Newer device specification is available but has not yet been applied (or there was a problem encountered while updating)
Error	Any configured application reporting Error status	Resources (CPU, Memory, Disk Util) reaching Critical levels	Updating: Device is actively in the process of applying a new device specification
Unknown	Application status is currently not known (e.g. if device is disconnected)	Device has been disconnected and has not been 'last seen' by the service for > 5 min	Unknown: Device can enter this state if device is disconnected while an update was in progress
Status States >>	Preparing: An application is getting ready to run	Rebooting: Device is rebooting as part of an update process	
	Starting: An application is in the process of starting up	Powered Off: Device is powered off	
	Running: An application is actively executing	Last Seen: Indicates last time device sent status update to service	
	Completion: Application has finished its execution		

Alarms

Device status
Connected / Disconnected

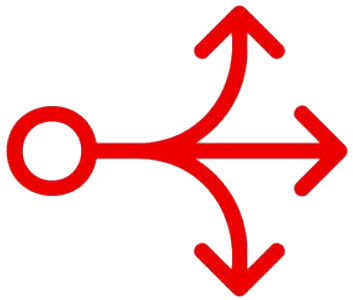
Resource usage
CPU (Critical/Warning/Normal)
Memory (Critical/Warning/Normal)
Disk (Critical/Warning/Normal)

Application
Connected / Disconnected

Lifecycle
Resource deletion
Device Decommissioning

Alarms

Email



slack

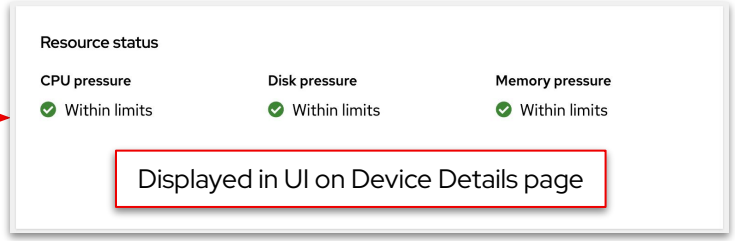
Webhook

Red Hat

Custom monitoring

Output device or fleet spec to .yaml

```
[root@rhem-01 ~]# flightctl get device/$mydevice -o yaml >
edgedevice-01.yaml && cat edgedevice-01.yaml
apiVersion: flightctl.io/v1alpha1
kind: Device
  <...>
  labels:
    alias: edgedevice-01
    type: kiosk
    region: east
 name: h367dba0a737nhll0gh55fiao13731sk710bs5dh3004u9j00
  resourceVersion: "1
spec:
  applications: []
  config: []
  os:
    image: quay.io/kenosborn/edgedevice-base:v1
<...>
Status:
<...>
spec:
  resources:
    cpu: Healthy
    disk: Healthy
    memory: Healthy
  summary:
    info: All system resources healthy.
    status: Online
```

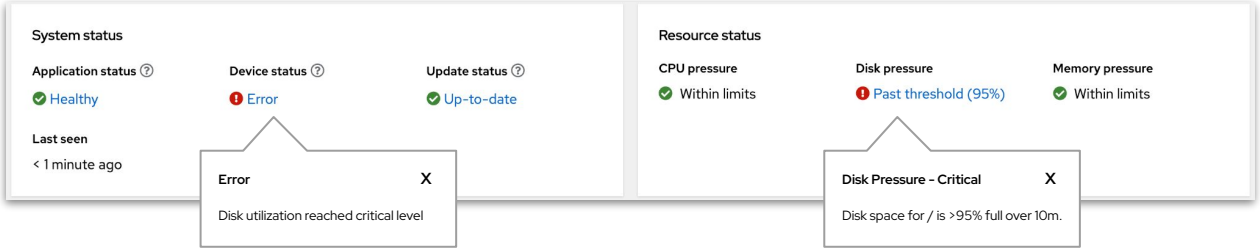


Create and insert custom resource monitor

```
spec:
  <...>
  resources:
  - monitorType: Disk
    samplingInterval: 5s
    path: /
    alertRules:
    - severity: Warning
      duration: 1m
      percentage: 75
      description: Disk space for / is >75% full for over 1m.
    - severity: Critical
      duration: 10m
      percentage: 95
      description: Disk space for / is >95% full over 10m.
```

Apply updated configuration

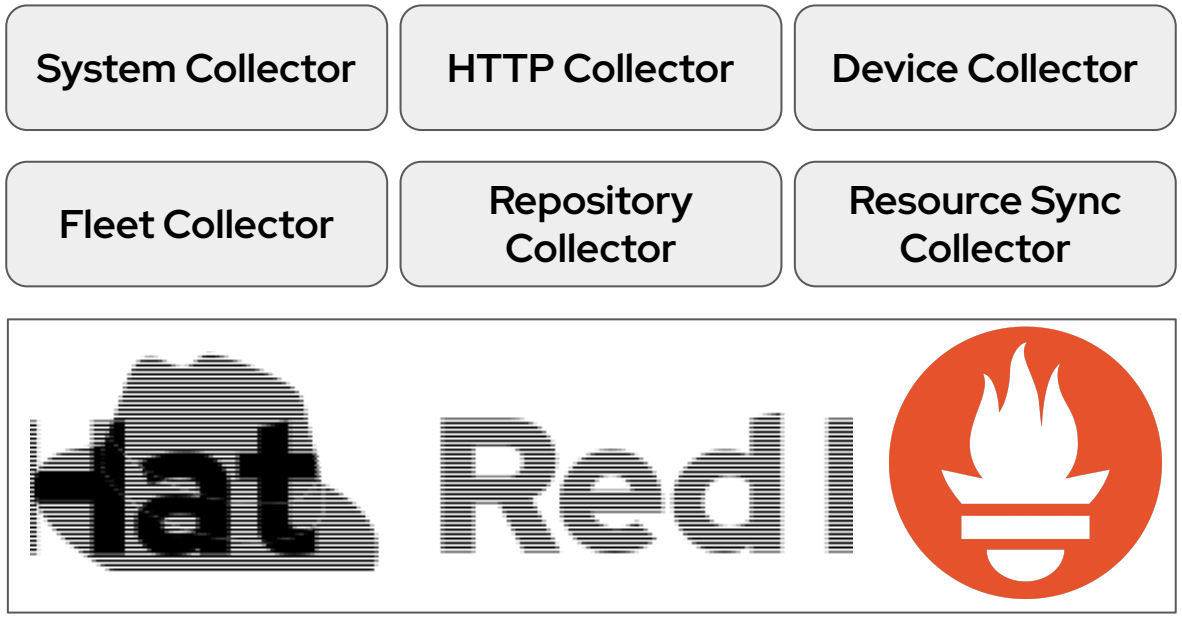
```
[root@rhem-01 ~]# flightctl apply -f edgedevice-01.yaml
device: applying
edgedevice-01.yaml/h367dba0a737nhll0gh55fiao13731sk710bs5dh3004u9j00:
200 OK
```



Red Hat Edge Manager

Metrics

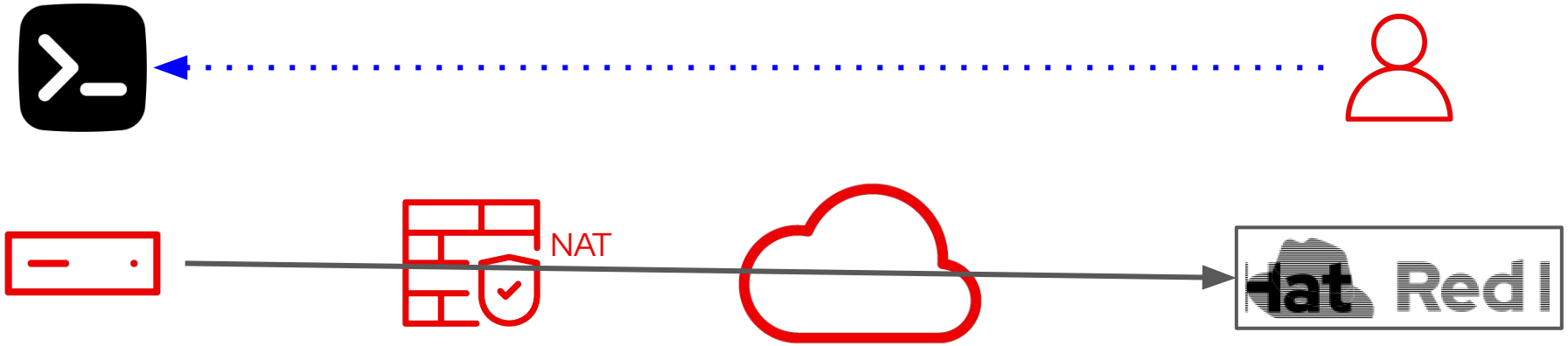
Metrics with Prometheus endpoint



Red Hat

Accessing devices remotely

Device remote CLI access



Red Hat

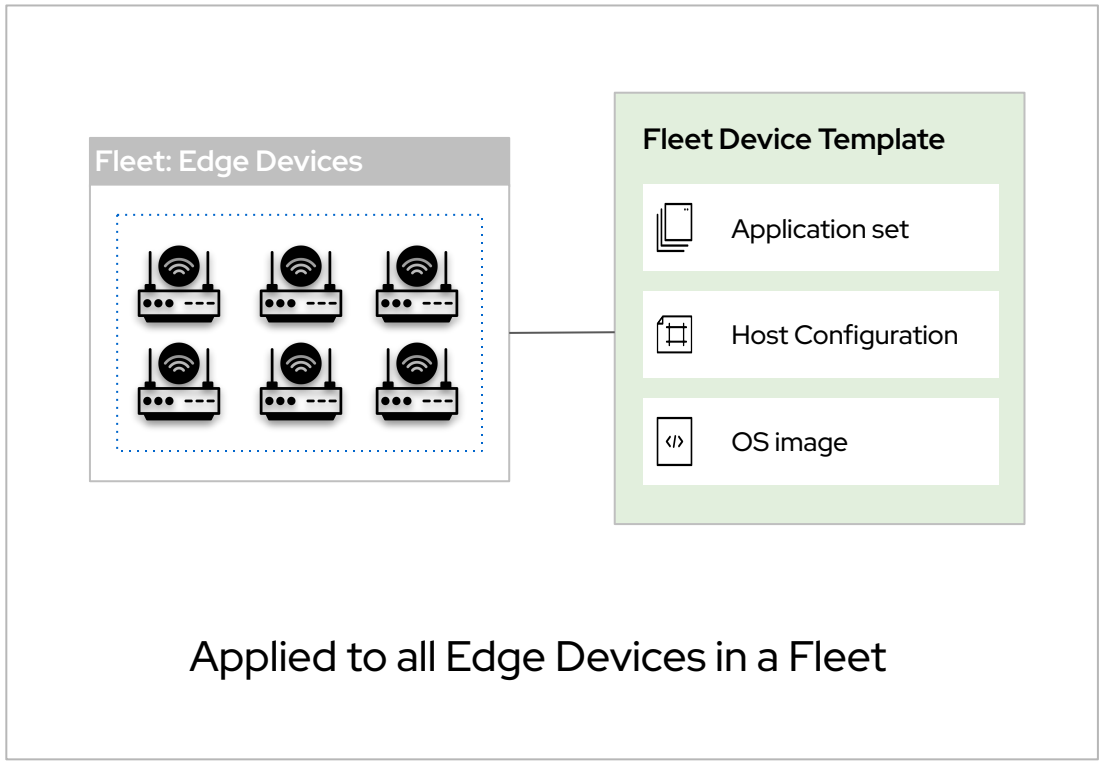
Fleet management

Red Hat

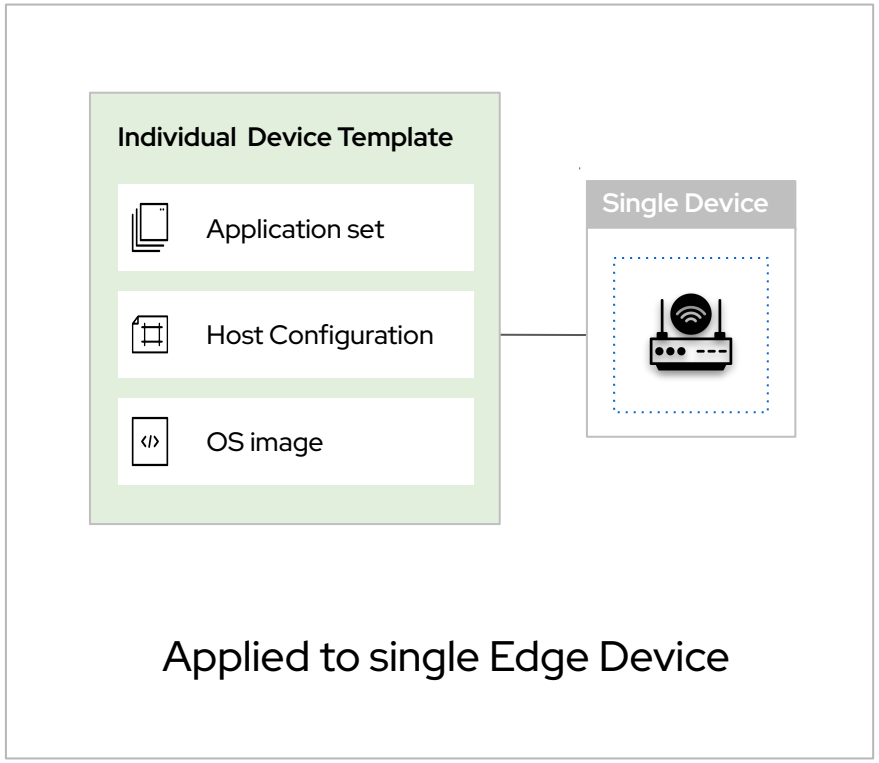
Fleets

Red Hat

Device templates

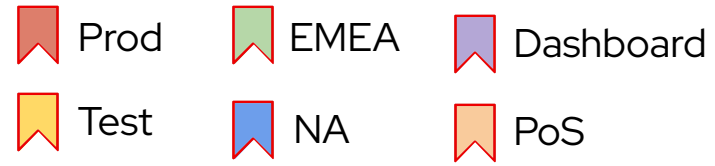


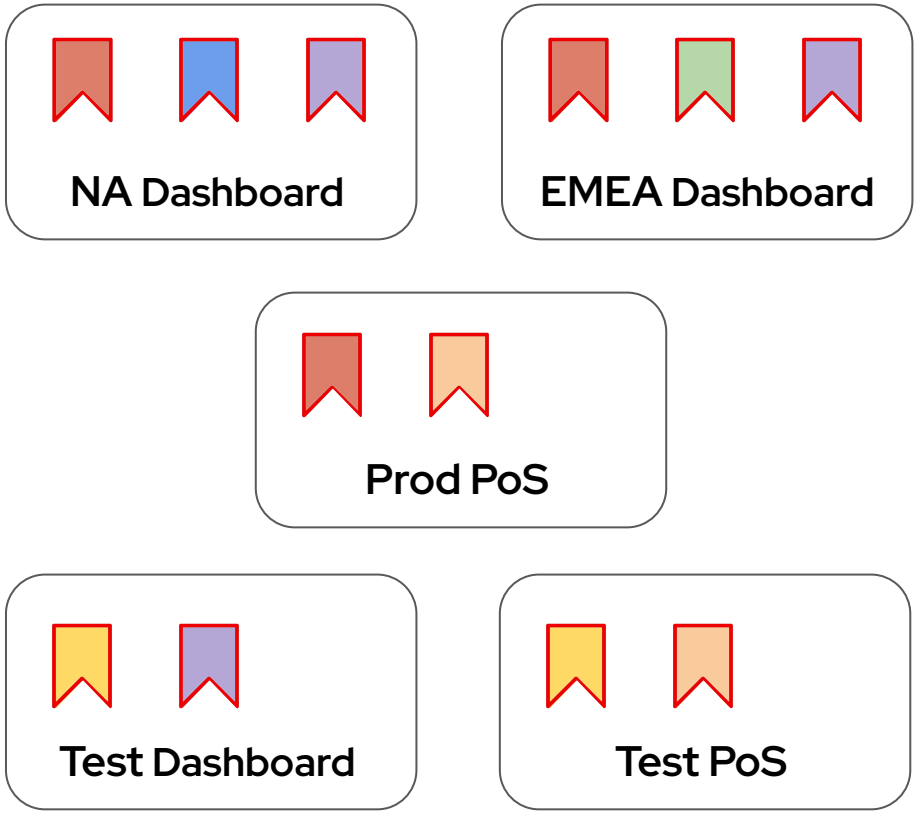
Applied to all Edge Devices in a Fleet

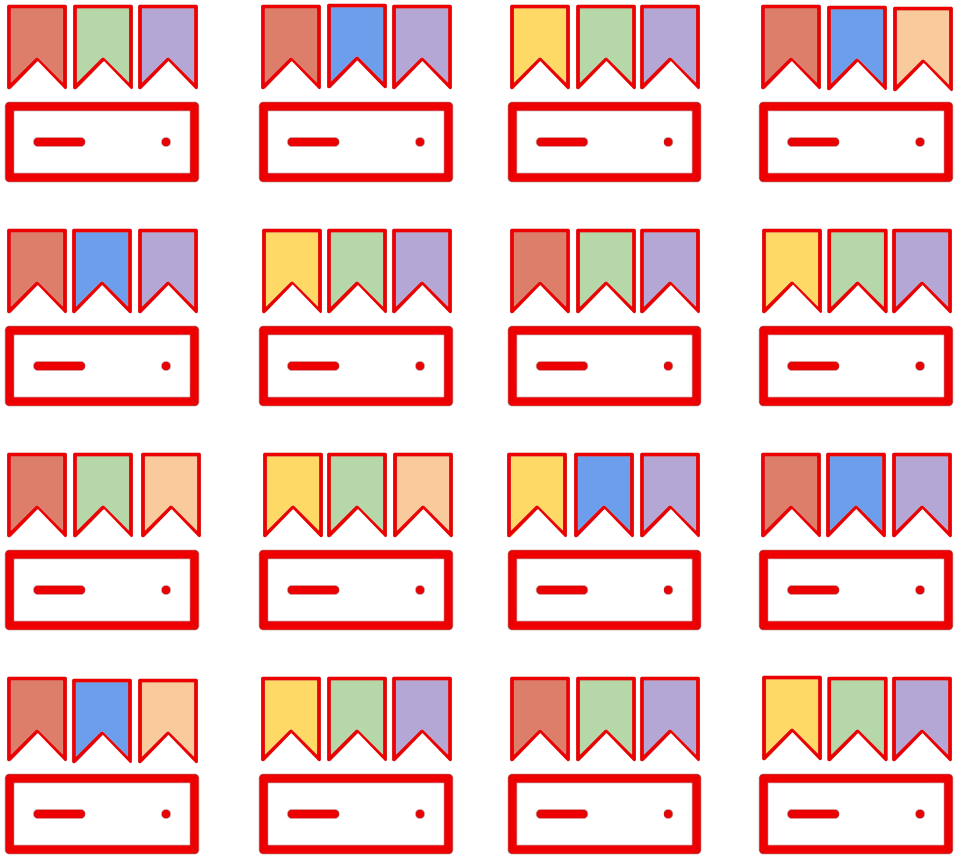


Applied to single Edge Device

Using tags to create Fleets



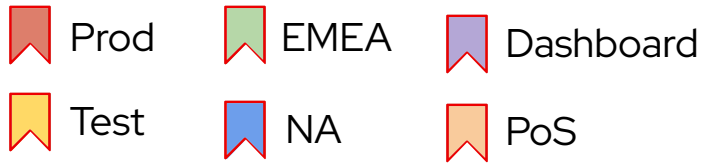




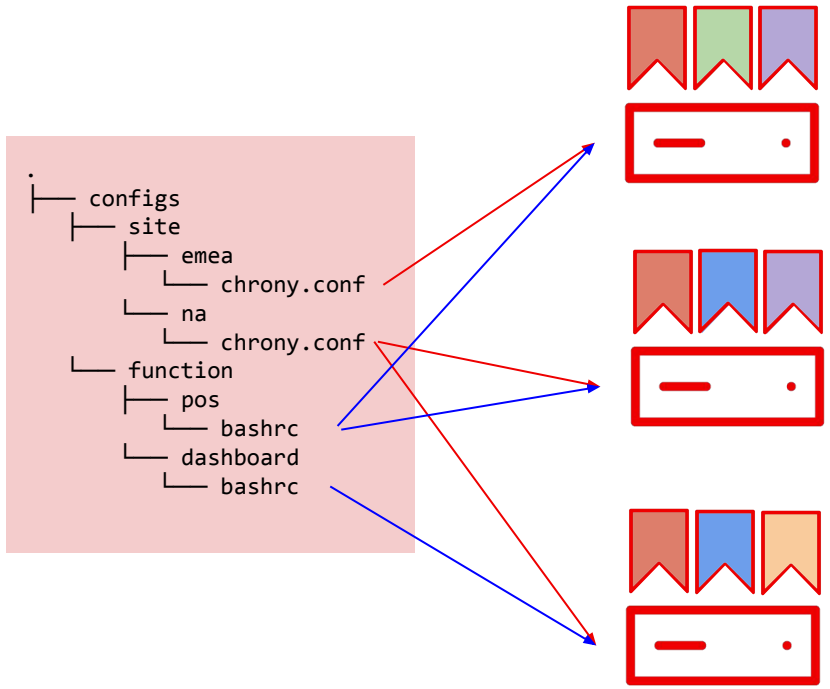
Red Hat

Device templates

Using Fleet templating




```
apiVersion: flightctl.io/v1alpha1
kind: Fleet
metadata:
  labels: {}
  name: workshop
  resourceVersion: "4"
spec:
  selector:
    matchLabels:
      fleet: "workshop"
  template:
    metadata:
      labels:
        fleet: workshop
    spec:
      applications:
      - envVars:
          IMAGE_NAME: quay.io/luisarizmendi/modelcar-hardhat:v1
        image: quay.io/luisarizmendi/compose-triton:latest
        name: triton
      config:
      - gitRef:
          mountPath: /etc
          path: /configs/site/{{ .metadata.labels.site }}
          repository: user01-fleet-mgmt
          targetRevision: main
        name: site-config
      - gitRef:
          mountPath: /etc
          path: /configs/function/{{ .metadata.labels.function }}
          repository: user01-fleet-mgmt
          targetRevision: main
        name: function-config
      systemd:
        matchPatterns:
          - "flightctl-agent.service"
      os:
        image: quay.io/luisarizmendi/type-1:prod
```



* The syntax for these placeholders matches that of https://pkg.go.dev/text/template

Red Hat

Update strategy

Update strategy

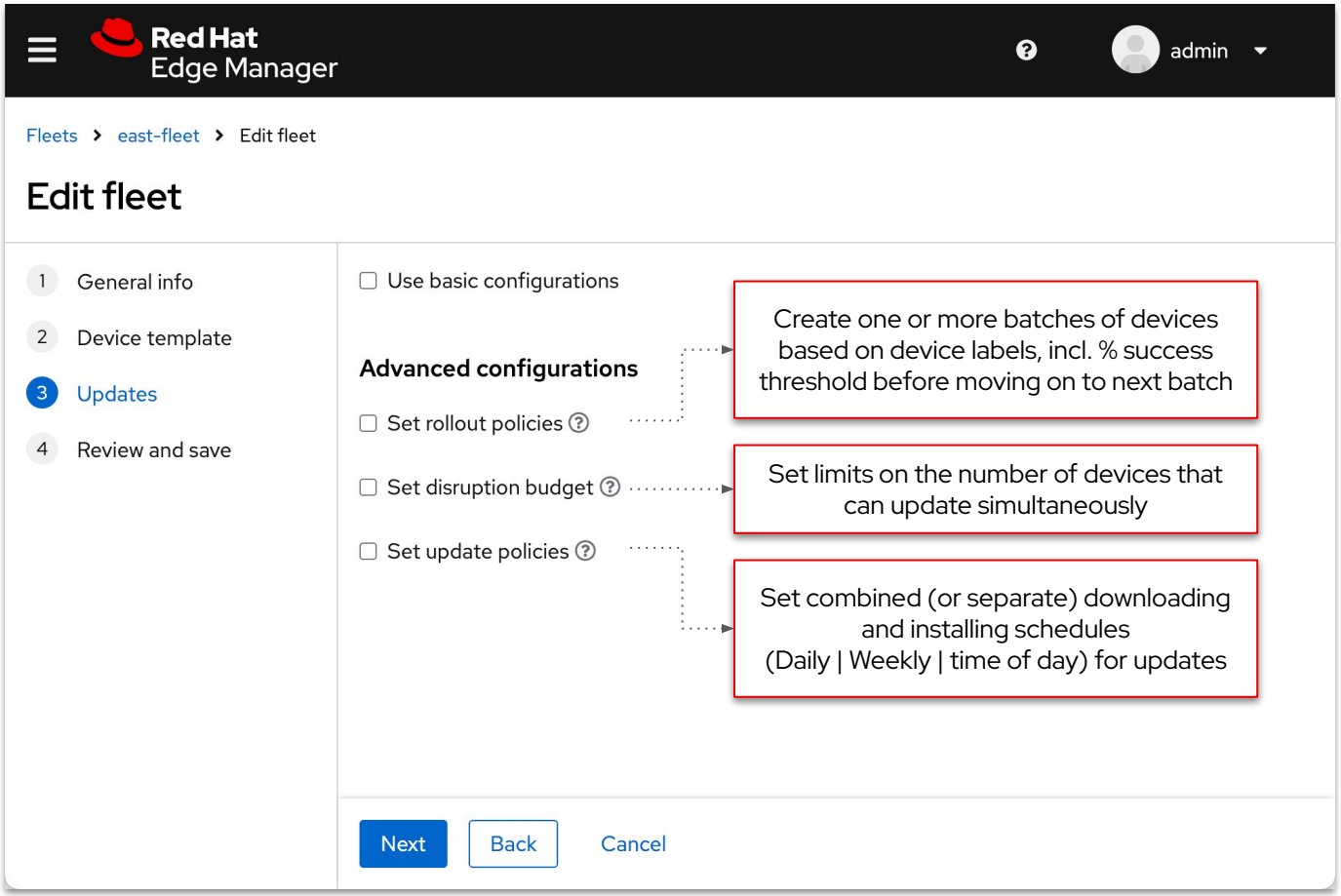
Did you know?

An ‘Update’ is defined as any device or fleet specification that would trigger a change on a device when the device agent checks in with Edge Manager (bootc image swap, new application, etc.)

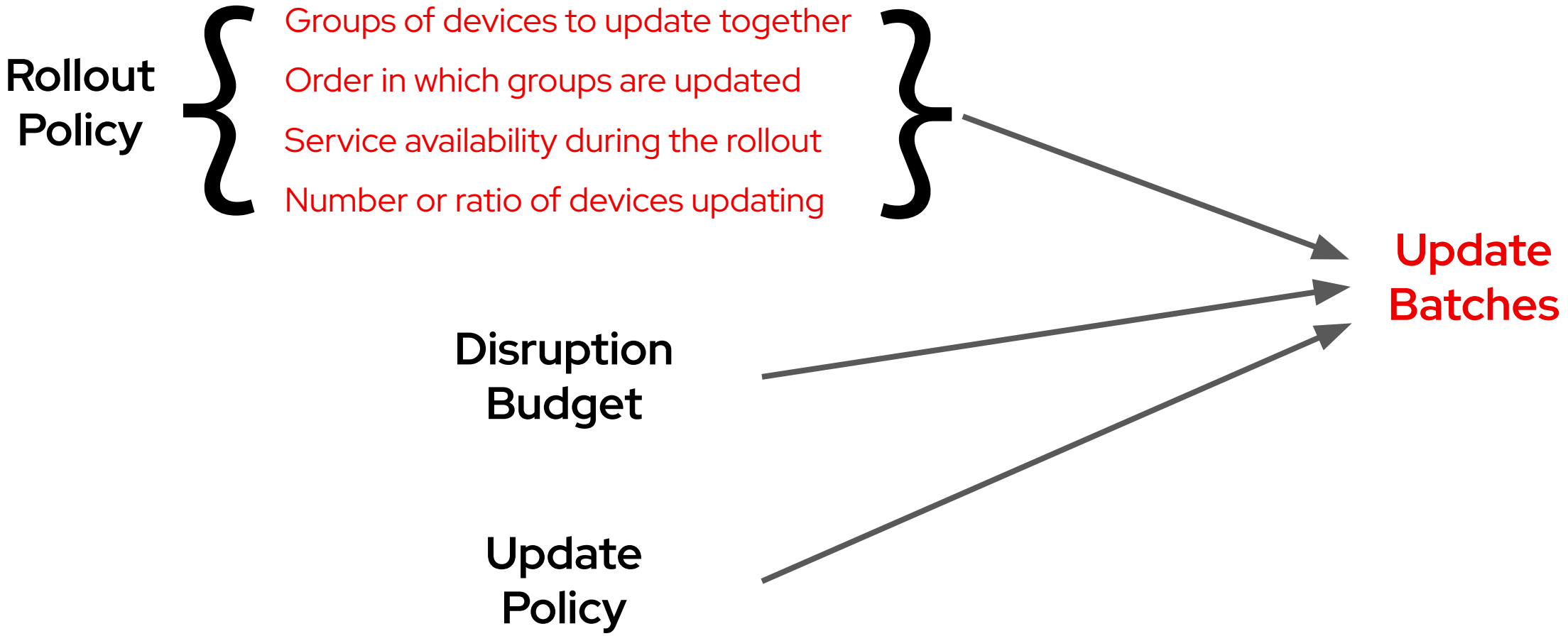
Default update policy (‘basic configuration’)
All devices that are a member of a fleet will receive updates as soon as they are available

Additional update policies (‘advanced configurations’)
Utilized for scenarios that requires more precise control over the order of updates, amount of simultaneous updates and update schedules

Policies can be utilized at Fleet or individual Device level
‘Set disruption budget’ and ‘Set update policies’ only applicable at Fleet level



Update strategy



Red Hat

Summary

Tech Preview

Red Hat Edge Manager

Comprehensive fleet management

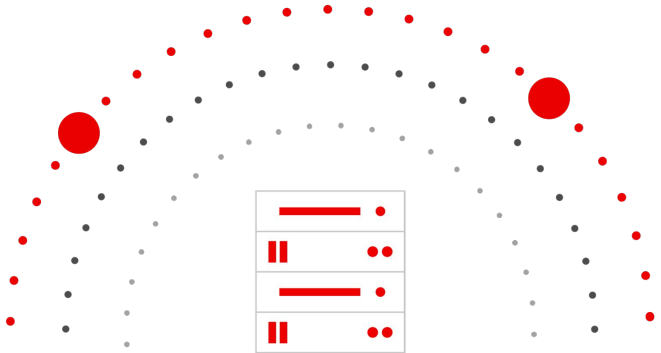
Simple

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 pull-mode management

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

Thank you

Red Hat