

Red Hat Edge Manager

Technical capabilities overview

Luis Arizmendi
Principal Edge Computing Specialist Solution Architect

Agenda

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

Red Hat Edge Manager Introduction

The Edge technical challenge

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**

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**

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

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

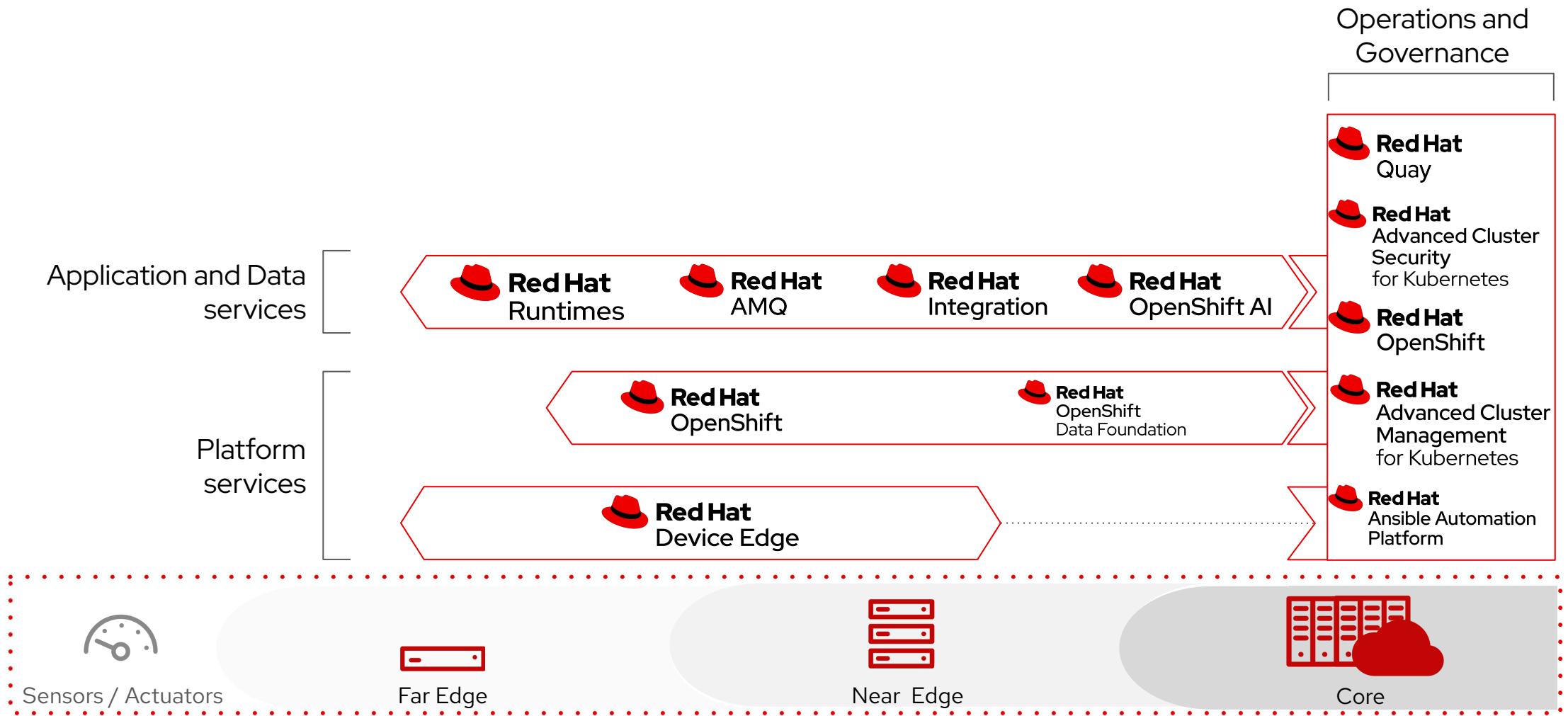


Target personas are different than in DC

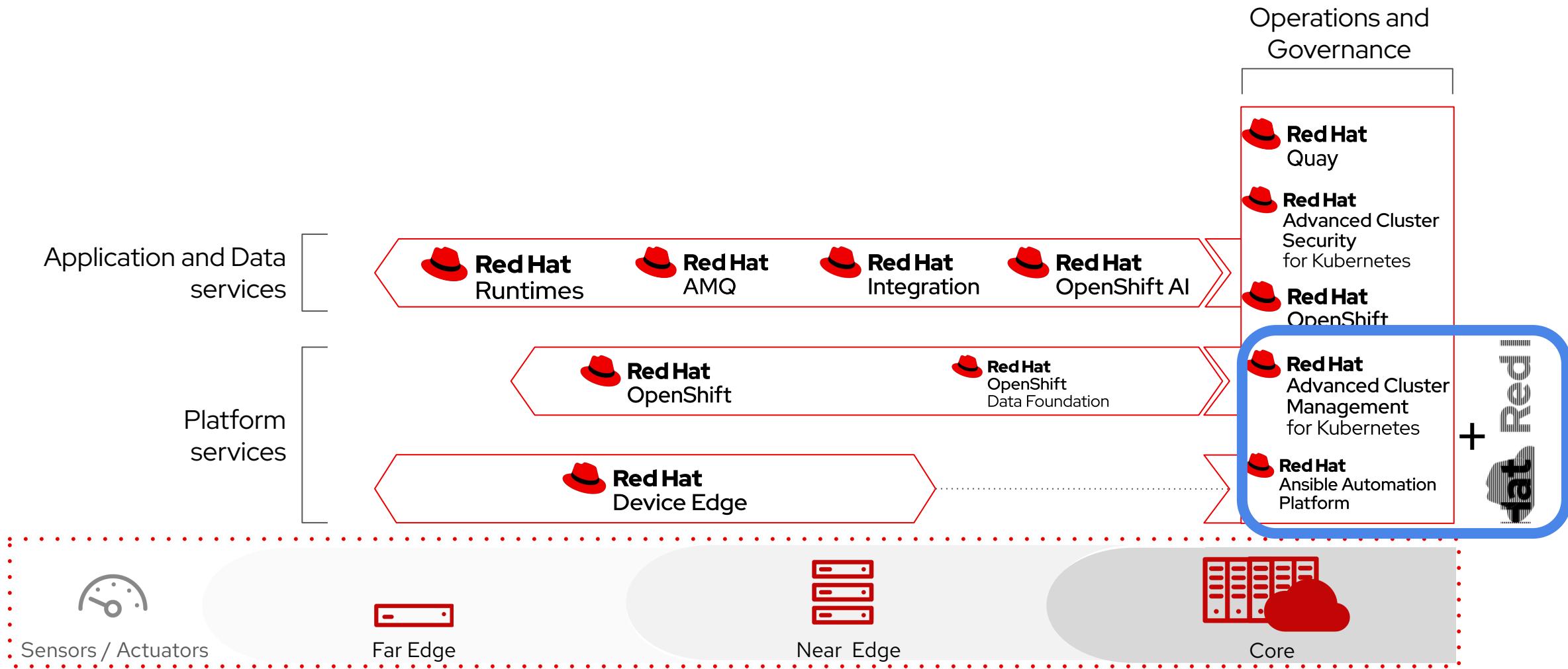
- UI (point-and-click) over CLI
- Self-Healing and unattended ops
- Simplified troubleshooting
- Single-pane-of-glass observability

Red Hat Edge Manager

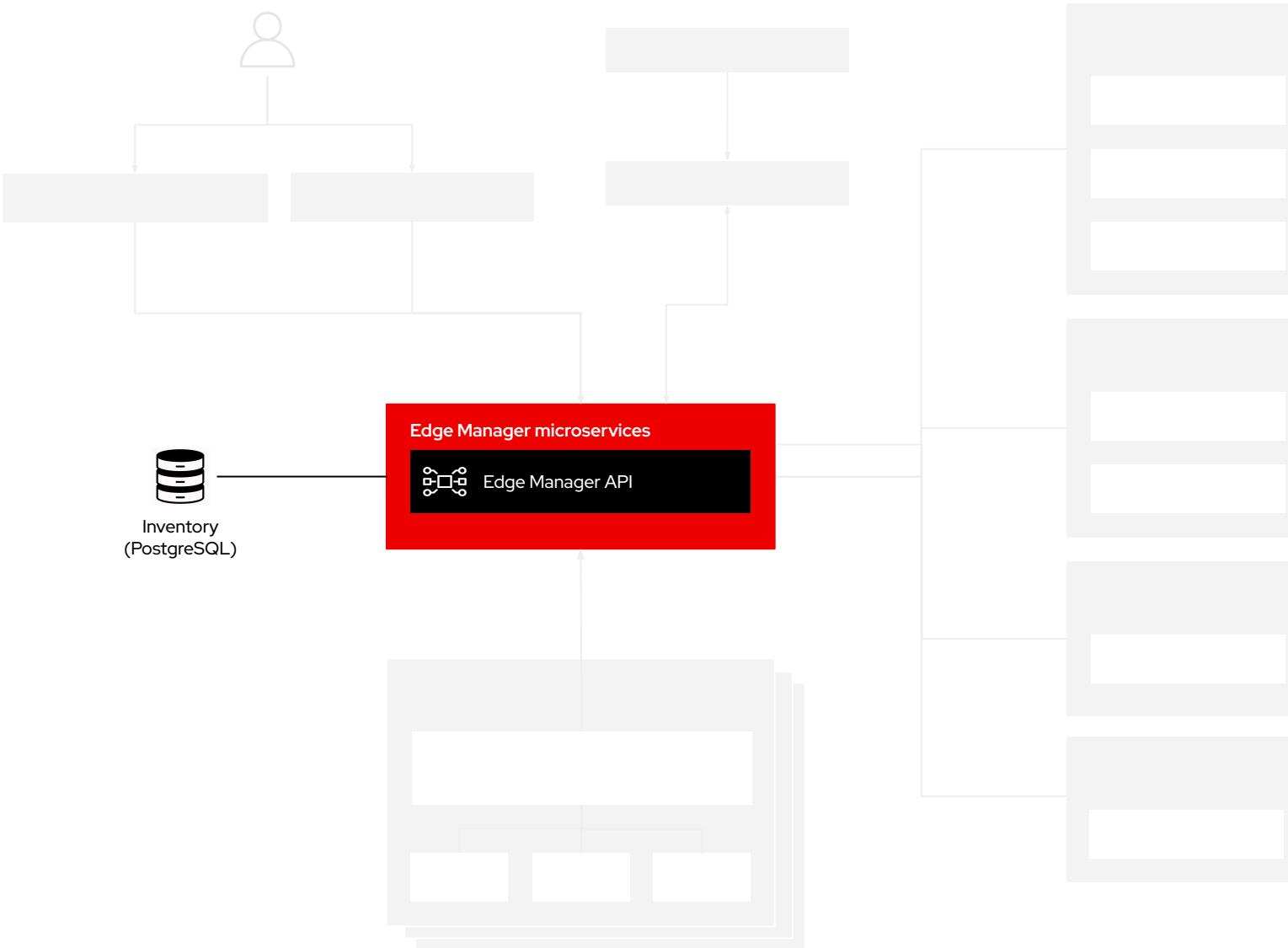
Red Hat Edge portfolio overview



Red Hat Edge portfolio overview



Red Hat Edge Manager architecture



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

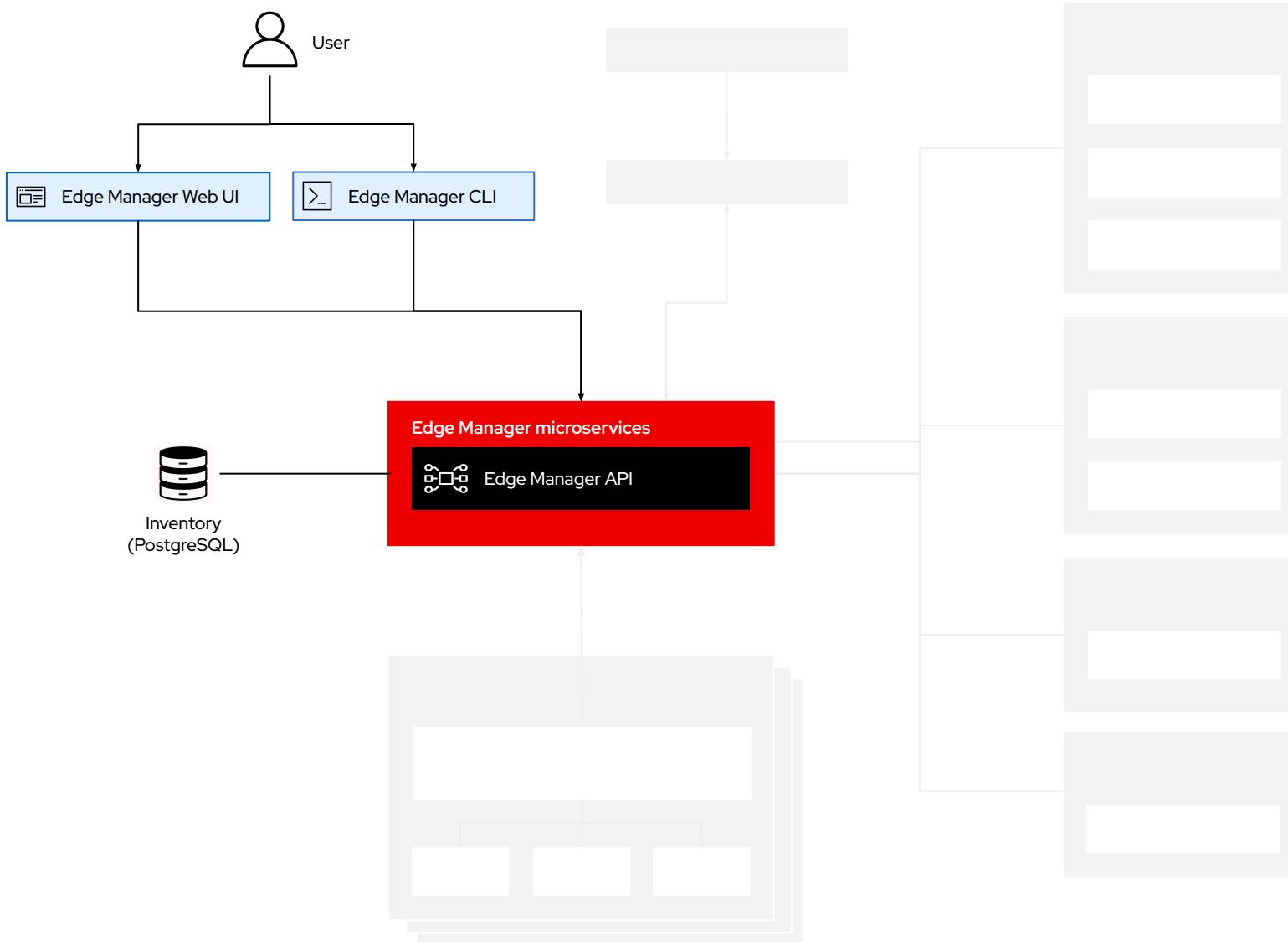


API Service used for User and Agent
communication



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

Red Hat Edge Manager architecture

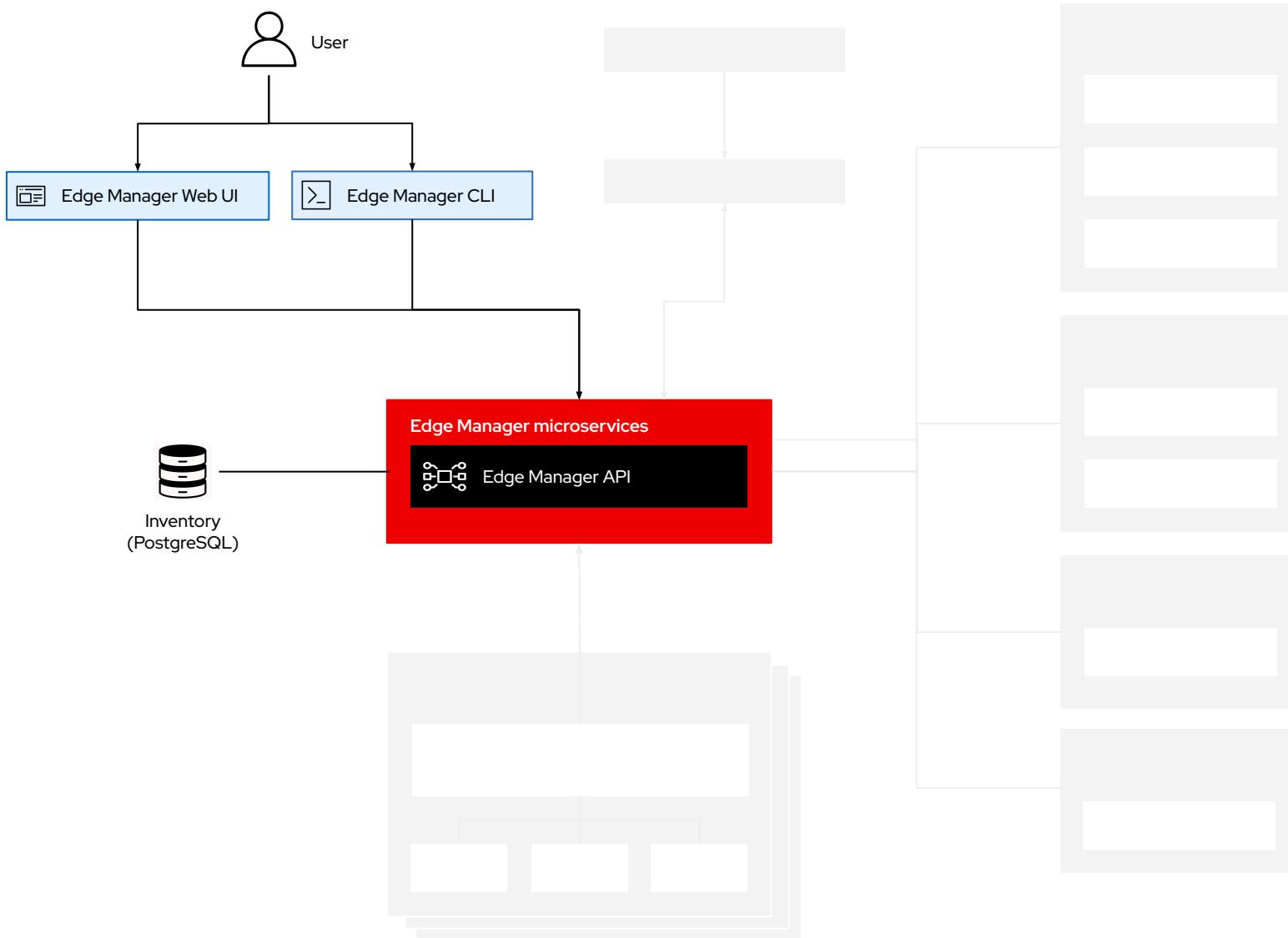


Web UI: Click-Ops manual interaction



Command-Line Interface (CLI)
enables automated interaction

Red Hat Edge Manager architecture



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 config. 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

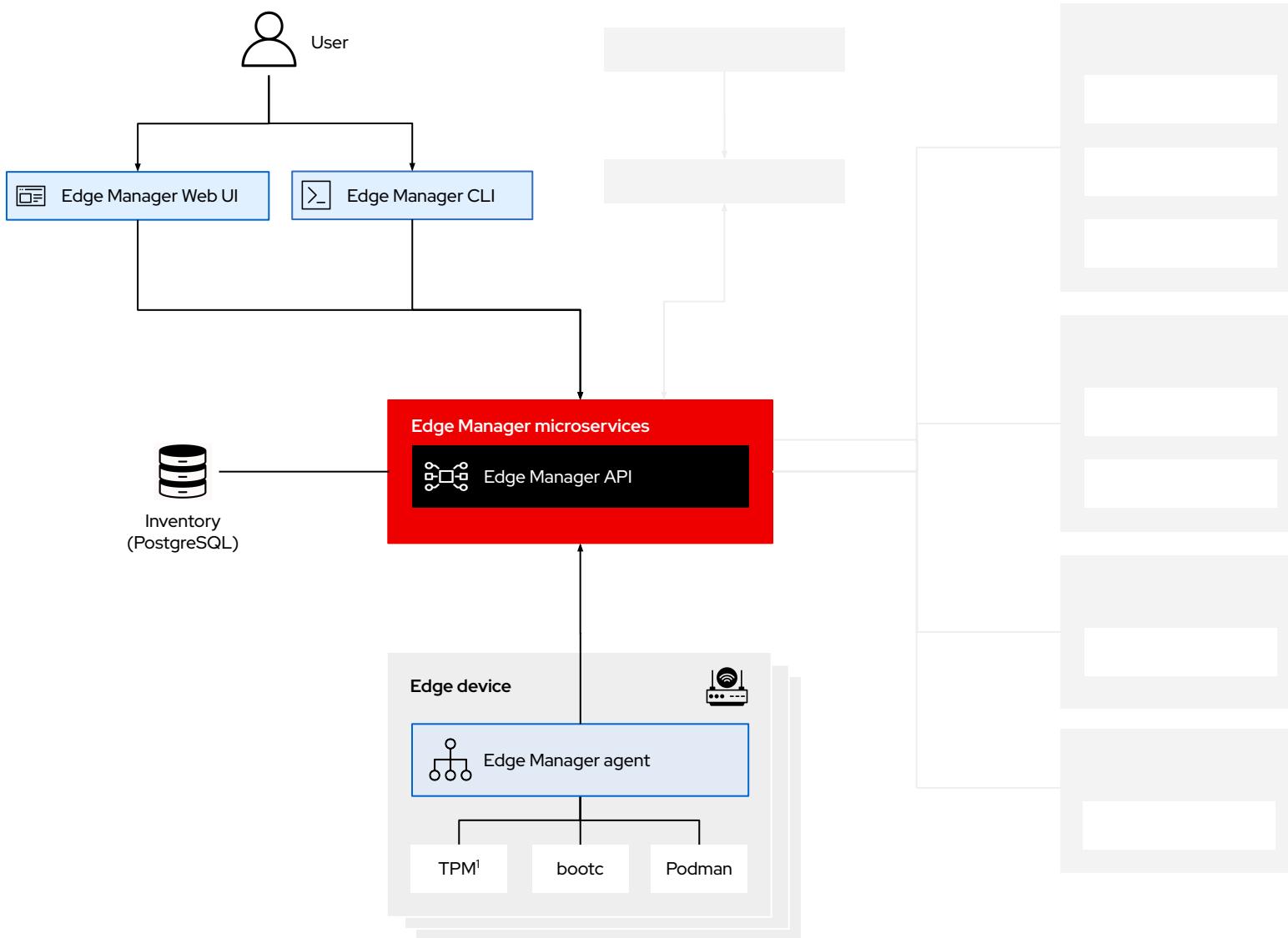
Login

Login to flight control

Version

Print flightctl version information

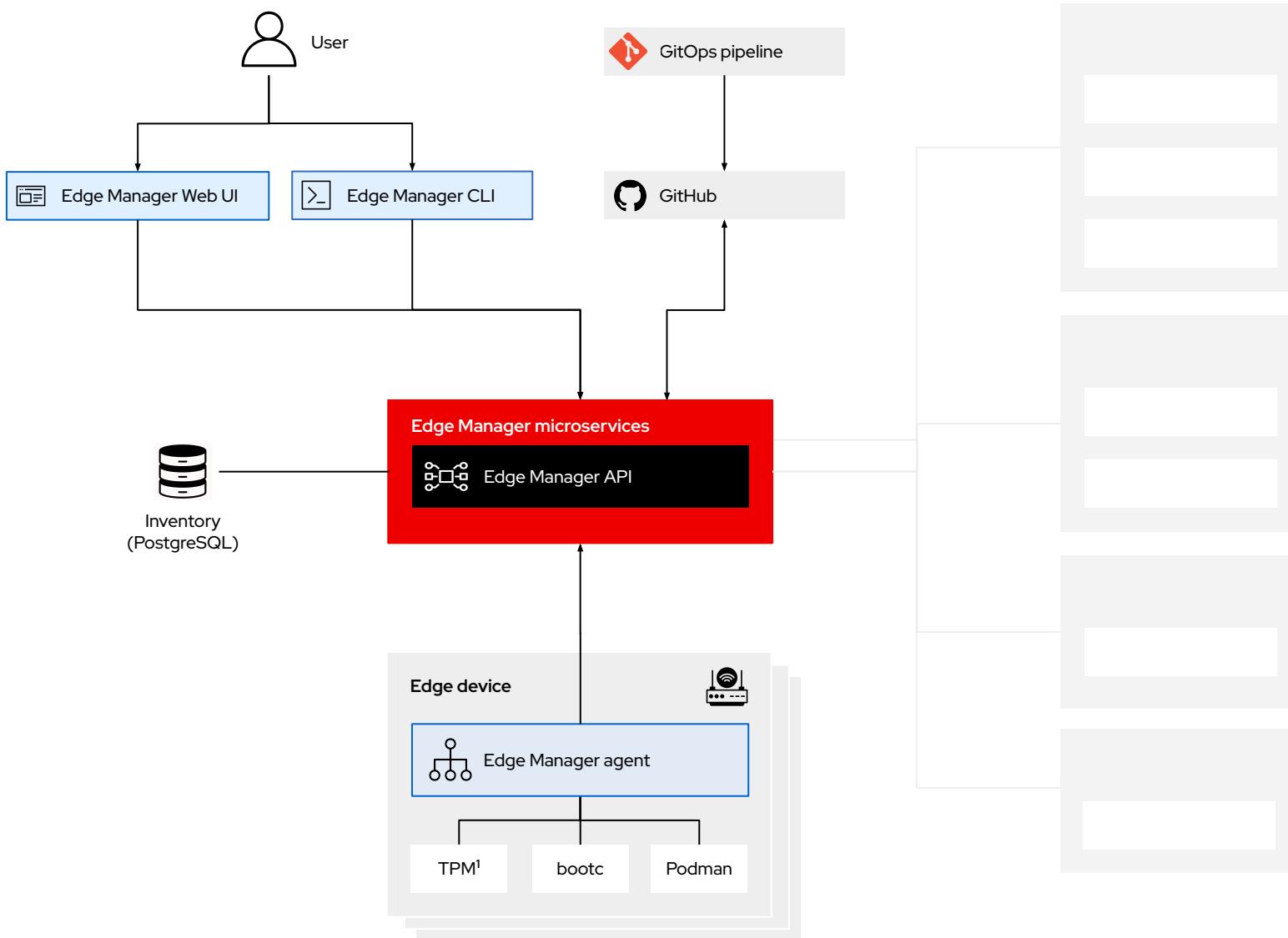
Red Hat Edge Manager architecture



Edge Manager agent required on each Edge Device

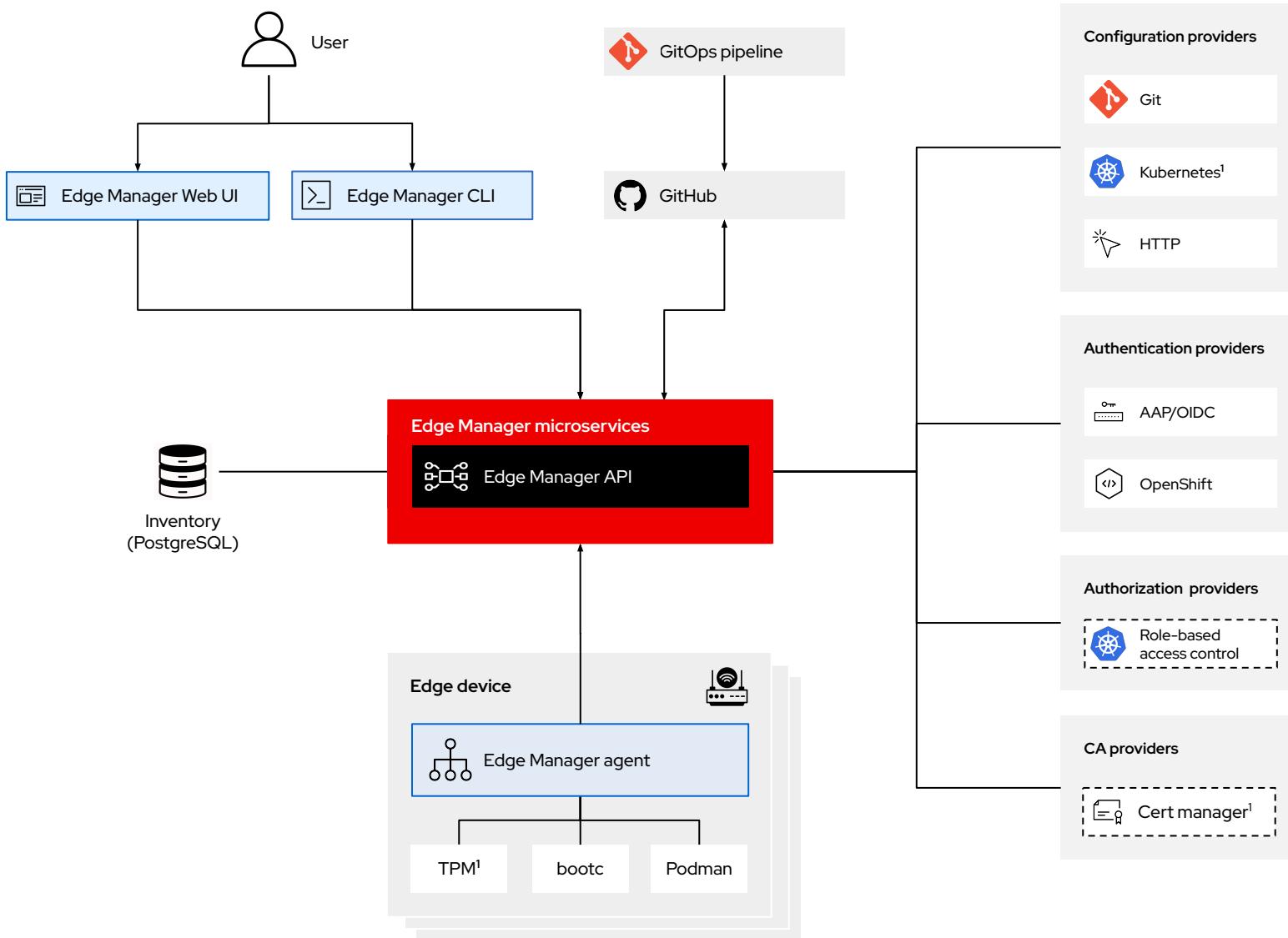
Agent is responsible for Enrollment and periodically checking in with the Edge Manager Service for changes in the device specification

Red Hat Edge Manager architecture



Modern integration with
GitOps Pipelines

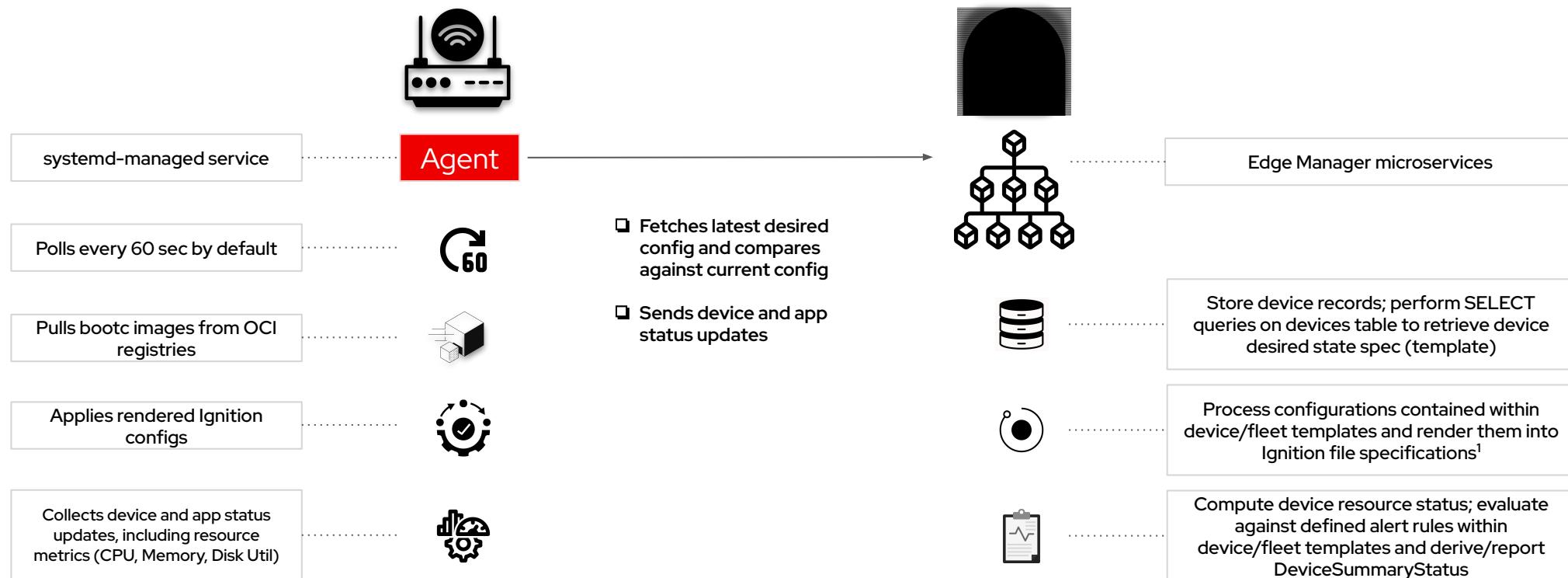
Red Hat Edge Manager architecture



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

Red Hat Edge Manager operation

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



Red Hat Edge Manager deployment options

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

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

The screenshot displays the Red Hat Ansible Automation Platform interface. On the left, a dark sidebar navigation menu includes links for Overview, Automation Execution, Automation Analytics, Application Links (with 'Red Hat Edge Manager' highlighted), Ansible Lightspeed, Access Management, Authentication Methods, Organizations, Teams, Users, Roles, OAuth Applications, Settings, and QuickStarts. The main content area features a header 'Welcome to the Ansible Automation Platform' with the tagline 'Empower, automate, connect: Unleash possibilities with the Ansible Automation Platform.' Below this are three summary cards: 'Resource Counts' (1 Hosts, 1 Ready; 1 Projects, 1 Ready; 1 Inventories, 1 Synced), 'Job Activity' (a line chart showing job counts over time with a peak on June 26th), and two tables: 'Jobs' (Recently finished jobs) and 'Projects' (Recently updated projects). The 'Jobs' table lists four successful tasks: 'Cleanup Activity Stream', 'Cleanup Job Details', 'Cleanup Expired OAuth 2 Tokens', and 'Cleanup Expired Sessions'. The 'Projects' table shows one project named 'Demo Project' with a status of 'Success'.

Edge Manager + Ansible Automation Platform

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



The diagram illustrates the integration of the Ansible Automation Platform (AAP) and Red Hat Enterprise Linux (RHEM) into the Edge Manager. It shows two separate components, AAP and RHEM, each with a red minus sign and a green plus sign icon, followed by the text "Installed". An arrow points from these two components to a larger black box containing a terminal session.

```
[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'.
```

Edge Manager + Ansible Automation Platform

Ansible Automation Platform Edge Manager collection

The screenshot shows the Red Hat Hybrid Cloud Console interface. On the left, there's a sidebar with various navigation options like Overview, Learning Resources, Automation Hub, Collections, Partners, Task Management, Connect to Hub, Automation Analytics, Ansible Lightspeed, Ansible Service on AWS, Documentation, Red Hat Insights, Inventory, Advisor, Policies, Registration Assistant, Remediations, and Tasks. The main content area is titled "Ansible Automation Platform > Automation Hub > Collections". It displays a collection named "Red Hat redhat.edge_manager" with a version of 1.1.0. The page includes sections for "Install", "Tools", "Infrastructure", and "Edge". It provides instructions for installation via ansible-galaxy and download, and a link to view the collection's signature. A note states that installing collections with ansible-galaxy is only supported in ansible-core >= 2.13.9. Below this, there's a section for the "Ansible Collection for Red Hat Edge Manager Service" which includes a description of its purpose.

```

---  

- 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  

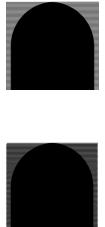
- name: Get all fleets  

  redhat.edge_manager.flightctl_resource_info:  

    kind: Fleet
  
```

*sample list

Edge Manager + Advanced Cluster Management



Advanced Cluster Management Operator deployed on OpenShift, delivering Edge Manager microservices



Edge Manager accessible via feature flag unlock

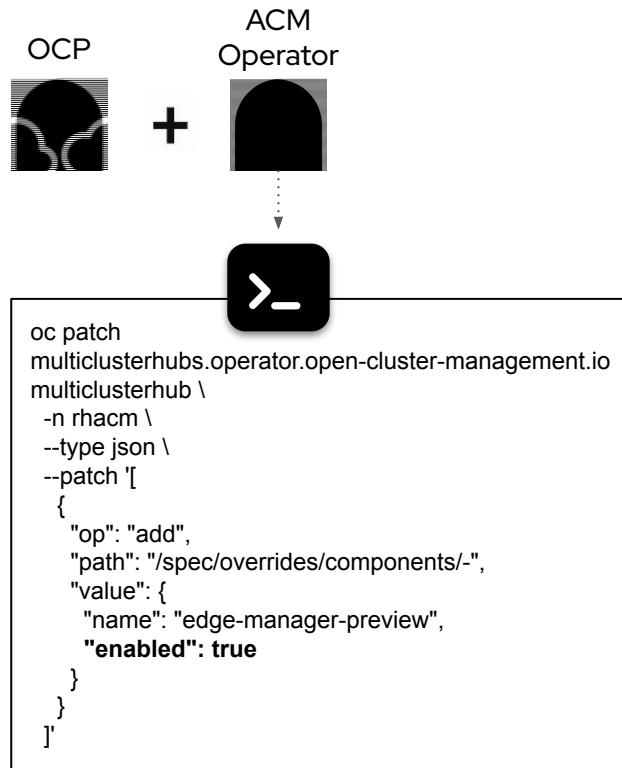
The screenshot shows the Red Hat Edge Manager interface running on a laptop. The left sidebar has a dark theme with the Red Hat OpenShift logo at the top. Under the 'Edge Management' section, there are links for Fleets, Devices, Repositories, Applications, Governance, and Credentials. The main content area is titled 'Overview' and shows 'Clusters' and 'Edge' tabs. It features three large green circles representing the status of different components: 'Application status', 'Device status', and 'System update status'. Each circle has a small question mark icon. Below each circle is a legend:

- Application status:** 0% Error, 0% Degraded, 0% Unknown, 100% Healthy
- Device status:** 0% Error, 0% Degraded, 0% Unknown, 0% Rebooting, 0% Powered Off, 100% Online
- System update status:** 0% Out-of-date, 0% Updating, 0% Unknown, 100% Up-to-date

A 'To do' section at the bottom says 'All good!'. A red 'Feedback' button is located in the bottom right corner of the main window.

Edge Manager + Advanced Cluster Management

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



Name	Namespace	Status	Labels	Pod selector
flightctl-api	open-cluster-management	1 of 1 pods	flightctl.servi...=flightctl..., installer.n...=multicluster..., installer....=open-cluster...	flightctl.service=flightctl-api
flightctl-db	open-cluster-management	1 of 1 pods	flightctl.servi...=flightctl..., installer.n...=multicluster..., installer....=open-cluster...	flightctl.service=flightctl-db
flightctl-periodic	open-cluster-management	1 of 1 pods	flightctl.servi...=flightctl-pe..., installer.n...=multicluster..., installer....=open-cluster...	flightctl.service=flightctl-periodic
flightctl-ui	open-cluster-management	1 of 1 pods	app=flightctl-ui, installer.n...=multicluster..., installer....=open-cluster..., role=frontend	app=flightctl-ui
flightctl-worker	open-cluster-management	1 of 1 pods	flightctl.servi...=flightctl-w..., installer.n...=multicluster..., installer....=open-cluster...	flightctl.service=flightctl-worker

Edge Manager + Advanced Cluster Management

Automation registration of edge device MicroShift clusters

Edge Device managed
by Edge Manager
running MicroShift



The screenshot shows the Red Hat Edge Manager interface. On the left, there is a sidebar with the following navigation items:

- Home (selected)
- Welcome
- Overview
- Search
- Infrastructure
 - Clusters (selected)
 - Automation
 - Host inventory
 - Virtual machines
- Edge Management
 - Fleets
 - Devices
 - Repositories

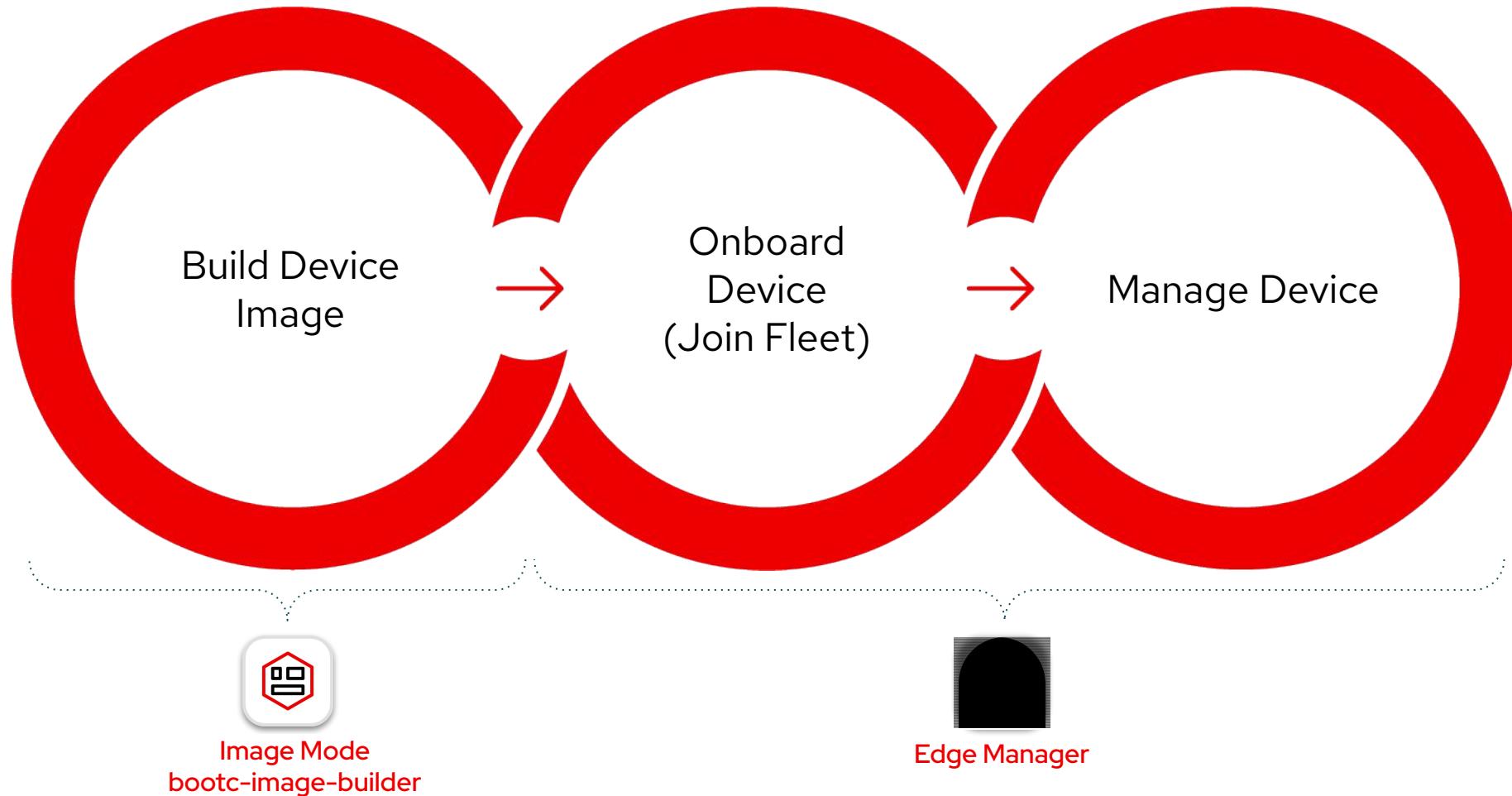
The main content area is titled "Clusters" and displays a table of cluster details. The table includes columns for Name, Namespace, Status, Infrastructure, Control plane type, Distribution version, Labels, Nodes, Add-ons, and Creation date. One cluster entry is visible:

Name	I514tit7n..strbl3uce45rcu119g
Namespace	I514tit7n..strbl3uce45rcu119g
Status	Ready
Infrastructure	Red Hat Device Edge
Control plane type	Standalone
Distribution version	MicroShift 4.18.11
Labels	microshiftVersion=4.18.11 10 more
Nodes	1
Add-ons	7
Creation date	Jul 30, 2025, 6:40 AM

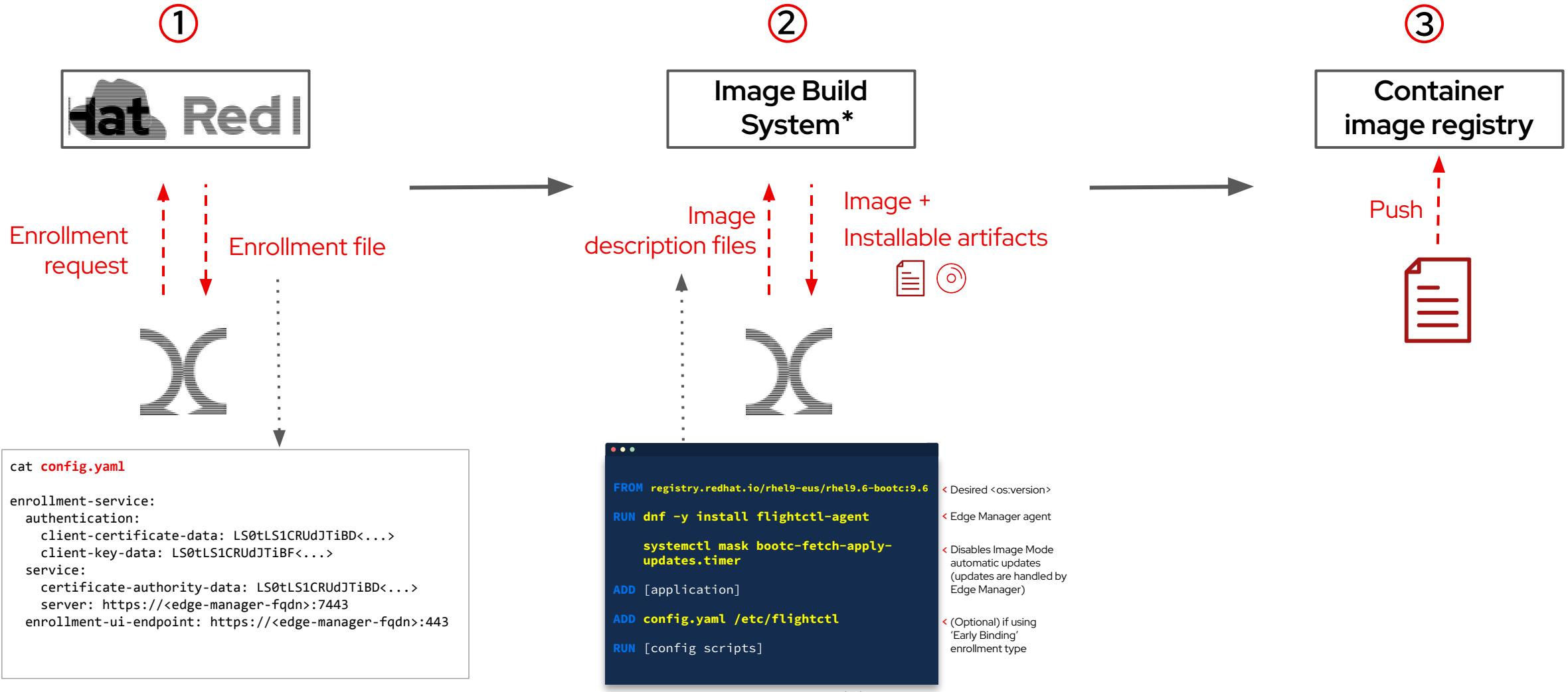
Provisioning Devices

Building a RHEL image *(with the RHEM Agent)*

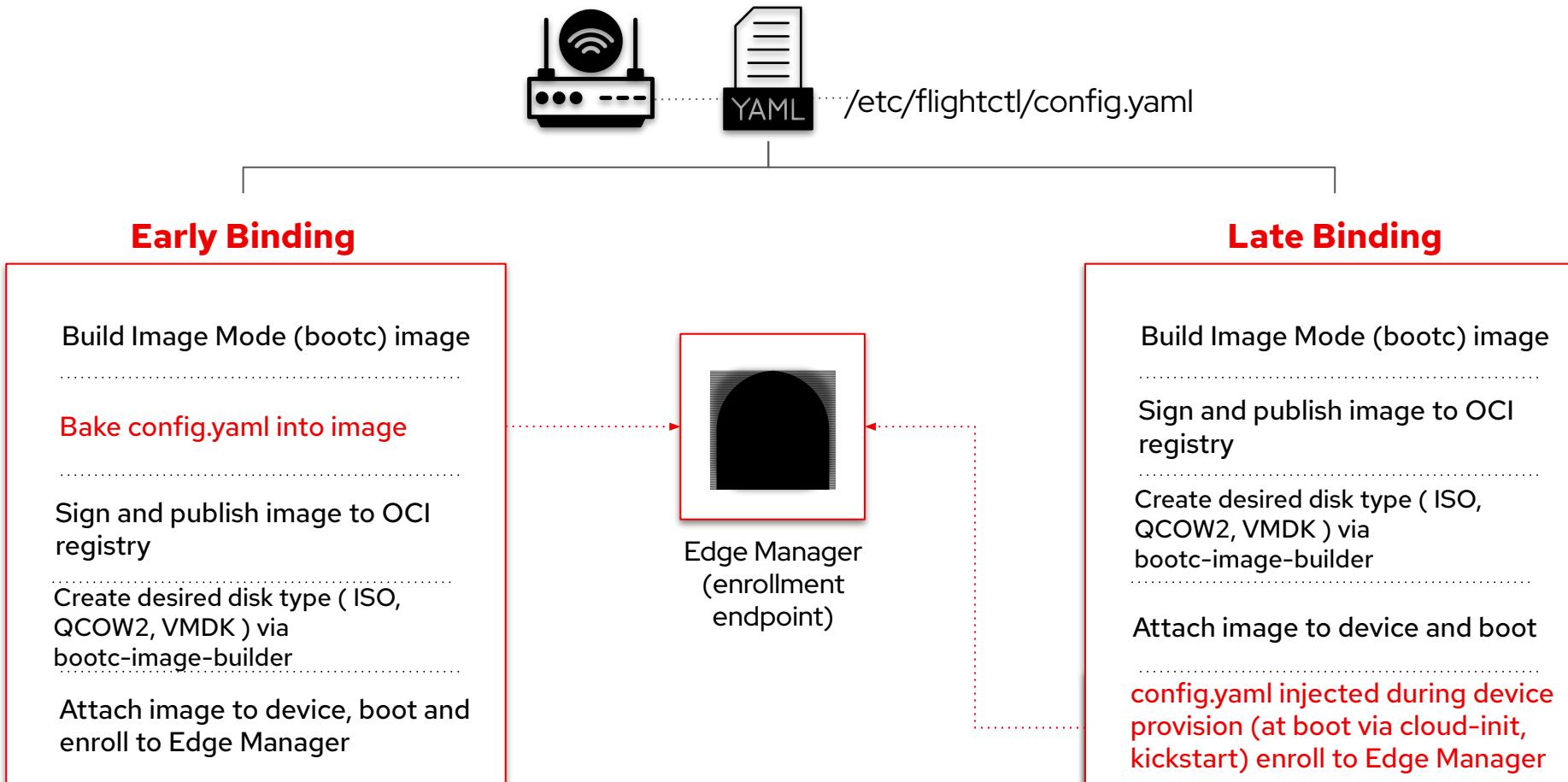
Device onboarding simplified workflow



Building a RHEL Image with RHEM agent

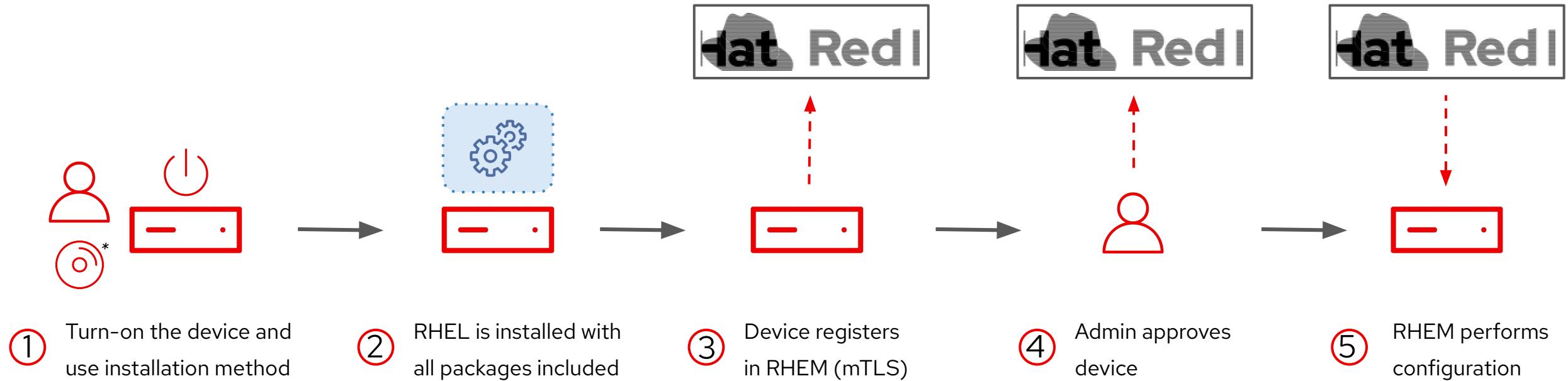


Building a RHEL Image with RHEM agent - Options



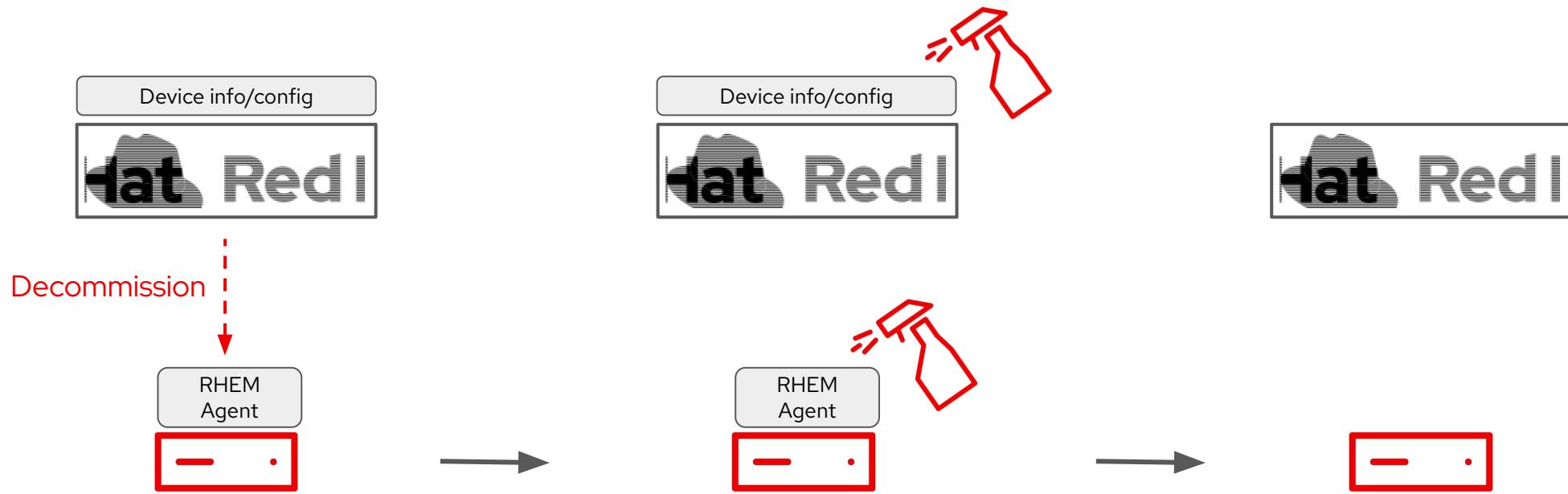
Device enrollment

Onboarding process



Device decommissioning

Device decommissioning



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

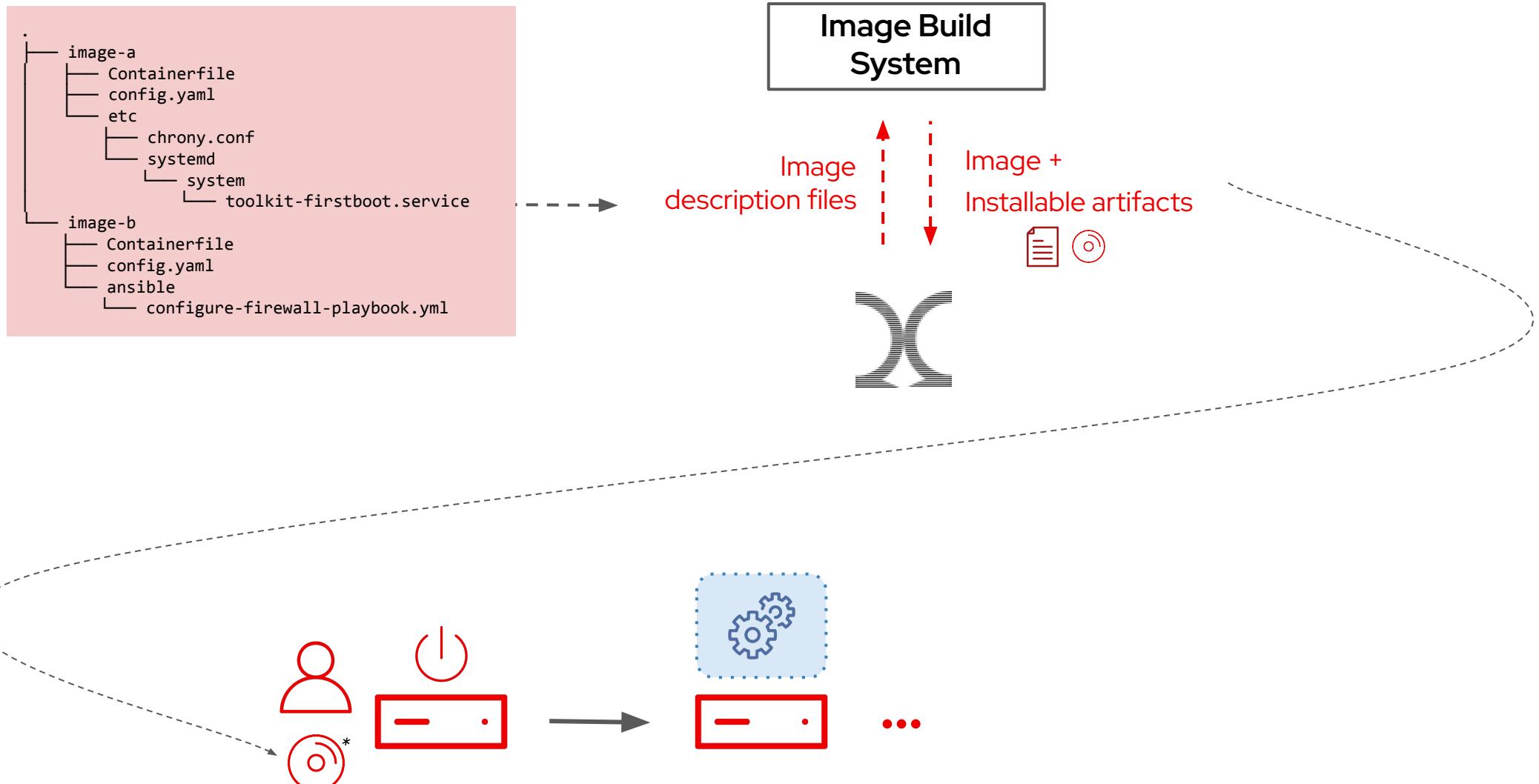
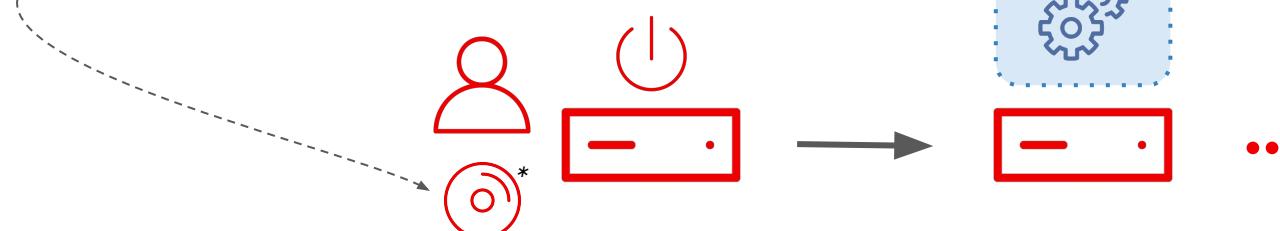
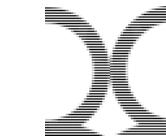


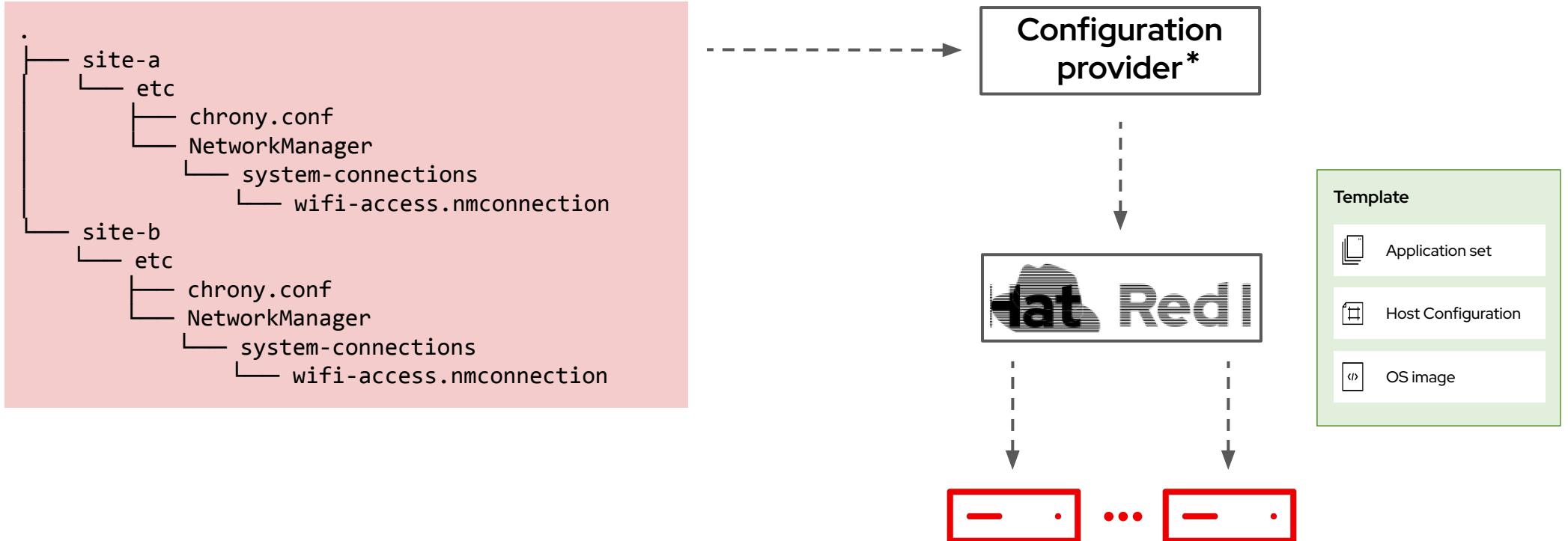
Image Build System

Image
description files

Image +
Installable artifacts



Managing OS configuration with RHEM



* 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.

Applications

Managing applications

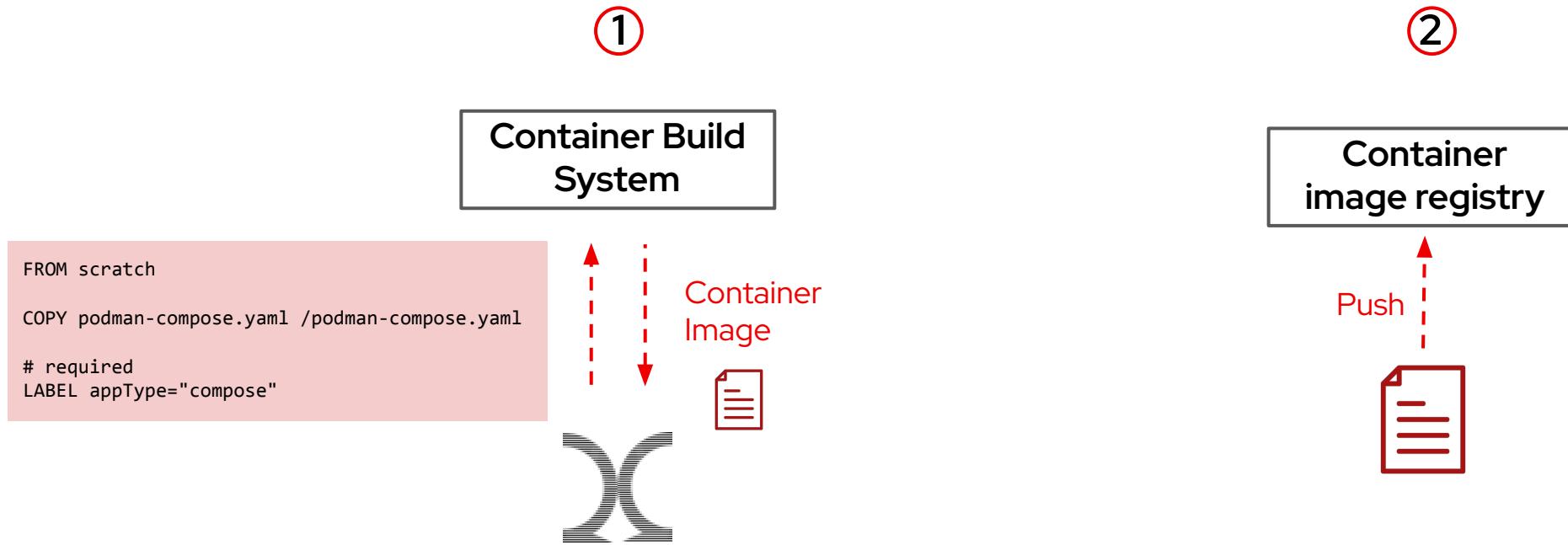
```
apiVersion: flightctl.io/v1alpha1
kind: Device
metadata:
  name: some_device_name
[...]
spec:
  applications:
    - name: my-inline
      appType: compose
      inline:
        - path: docker-compose.yaml
          content: |
            version: "3.8"
          services:
            service1:
              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
```

```
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"
[...]
```

Inline

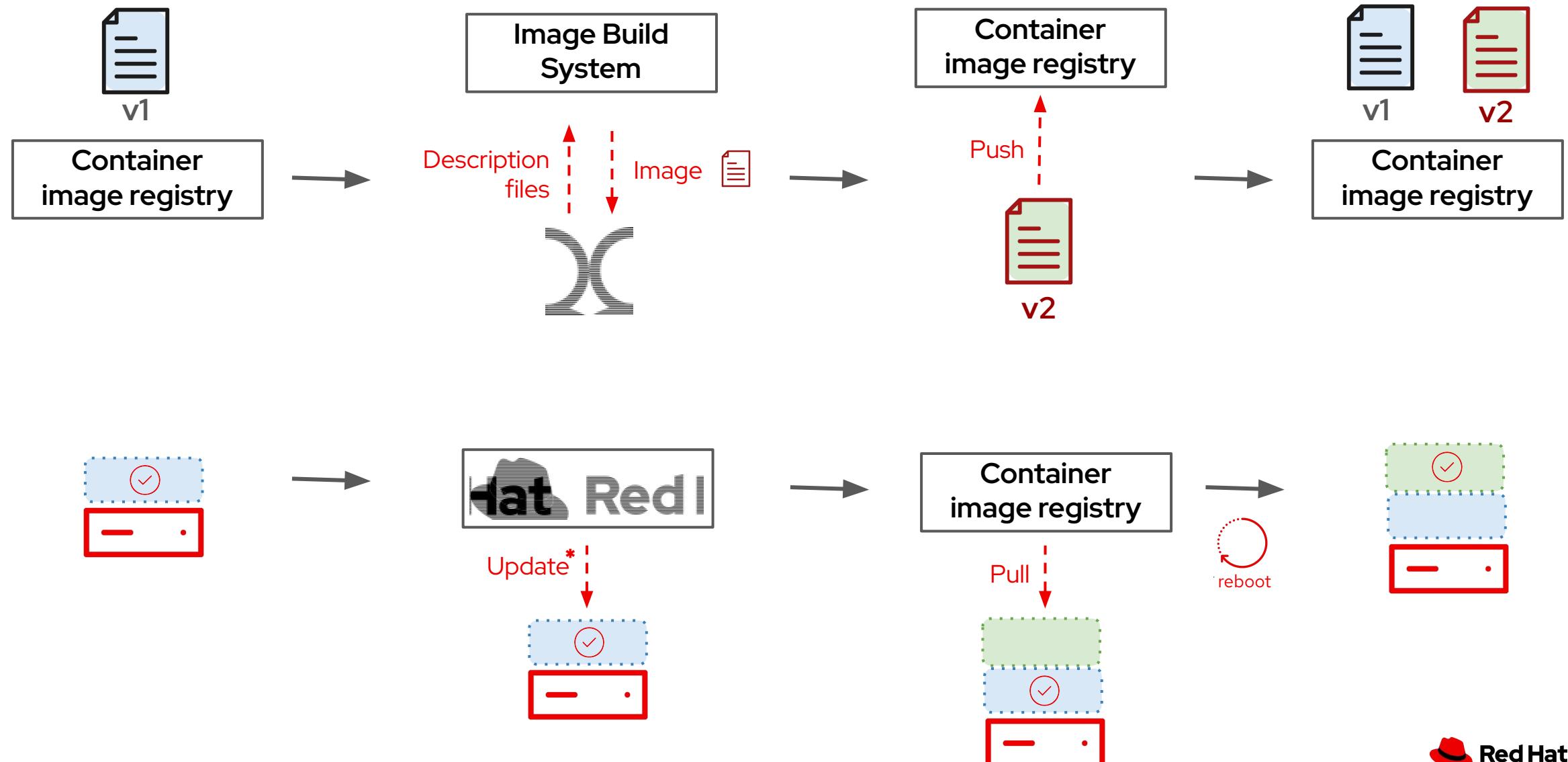
OCI Registry

Managing applications using OCI registry



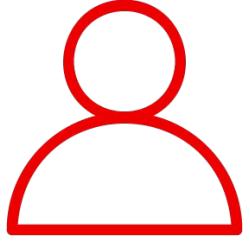
Device upgrade

Upgrade process

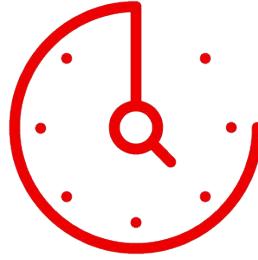


Configuration triggers

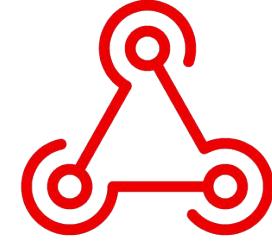
Configuration change triggering



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)

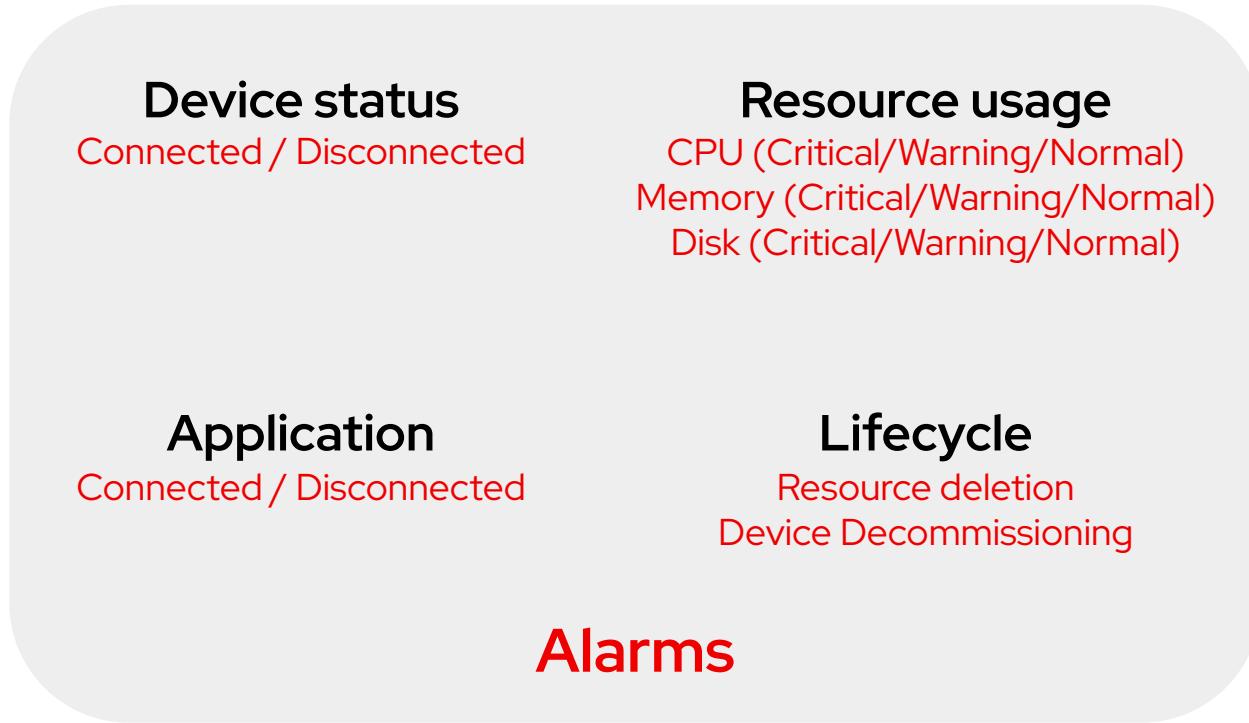
Device Observability

Alerts and monitoring

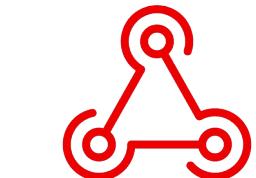
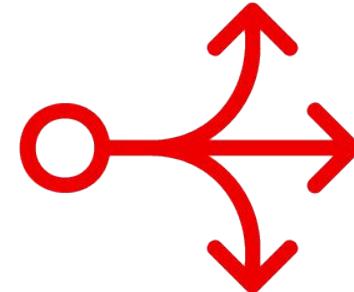
Device and fleet status

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



Email



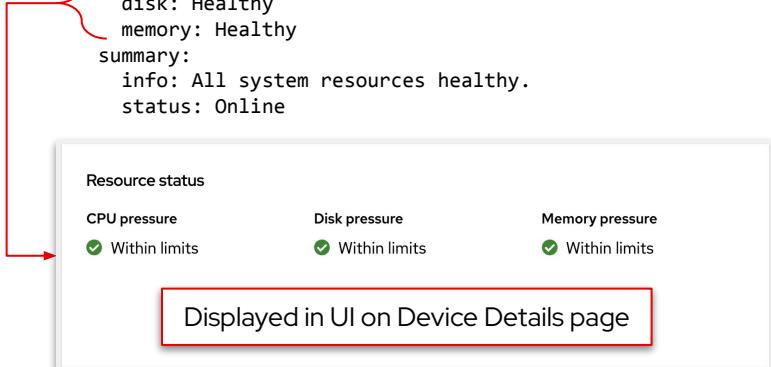
Webhook

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
```



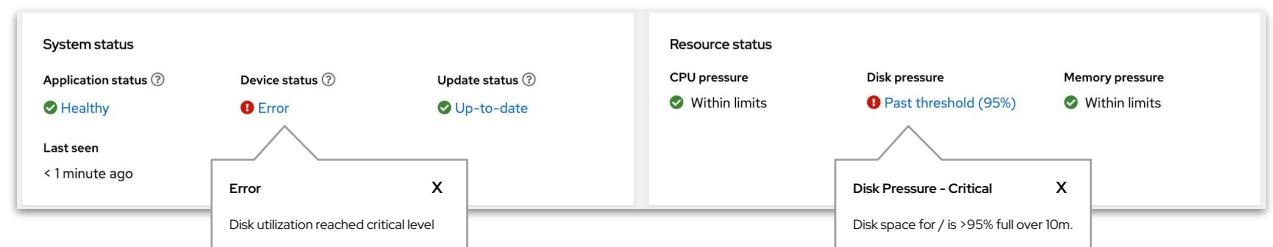
Create and insert custom resource monitor

```
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.
```



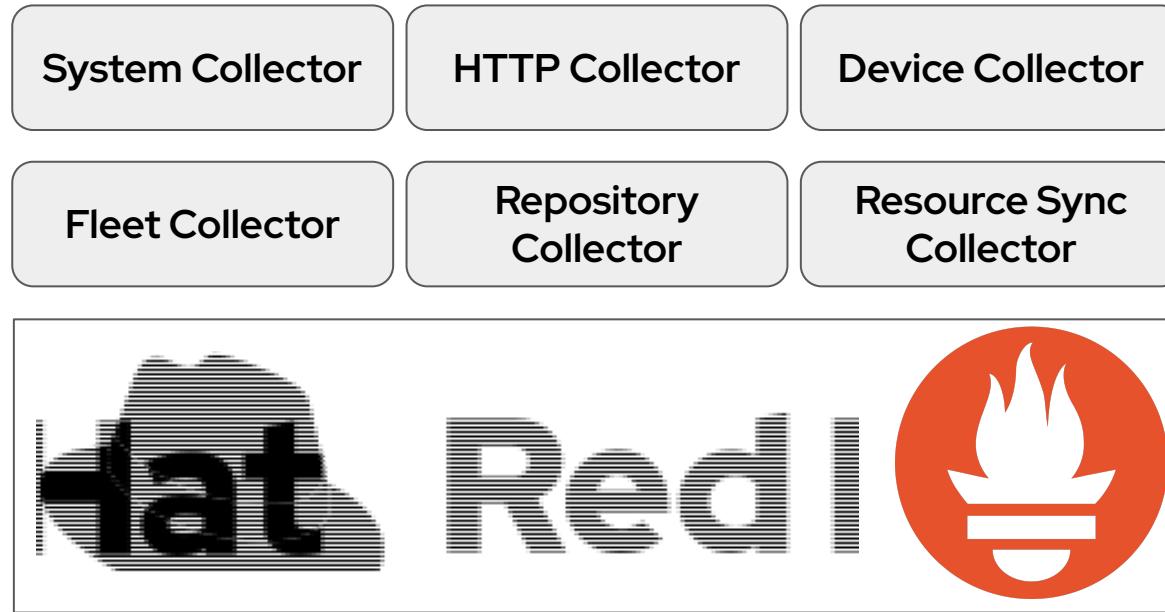
Apply updated configuration

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



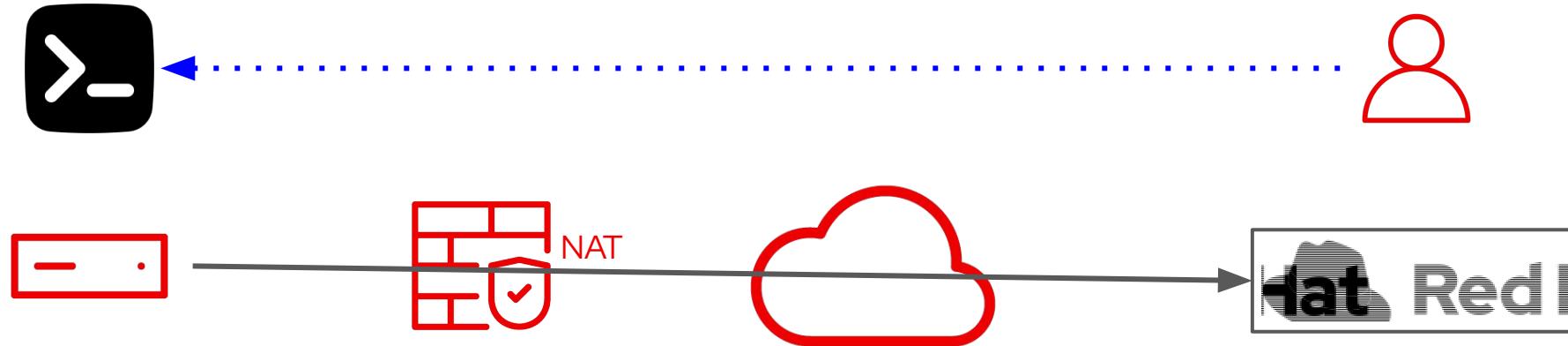
Metrics

Metrics with Prometheus endpoint



Accessing devices remotely

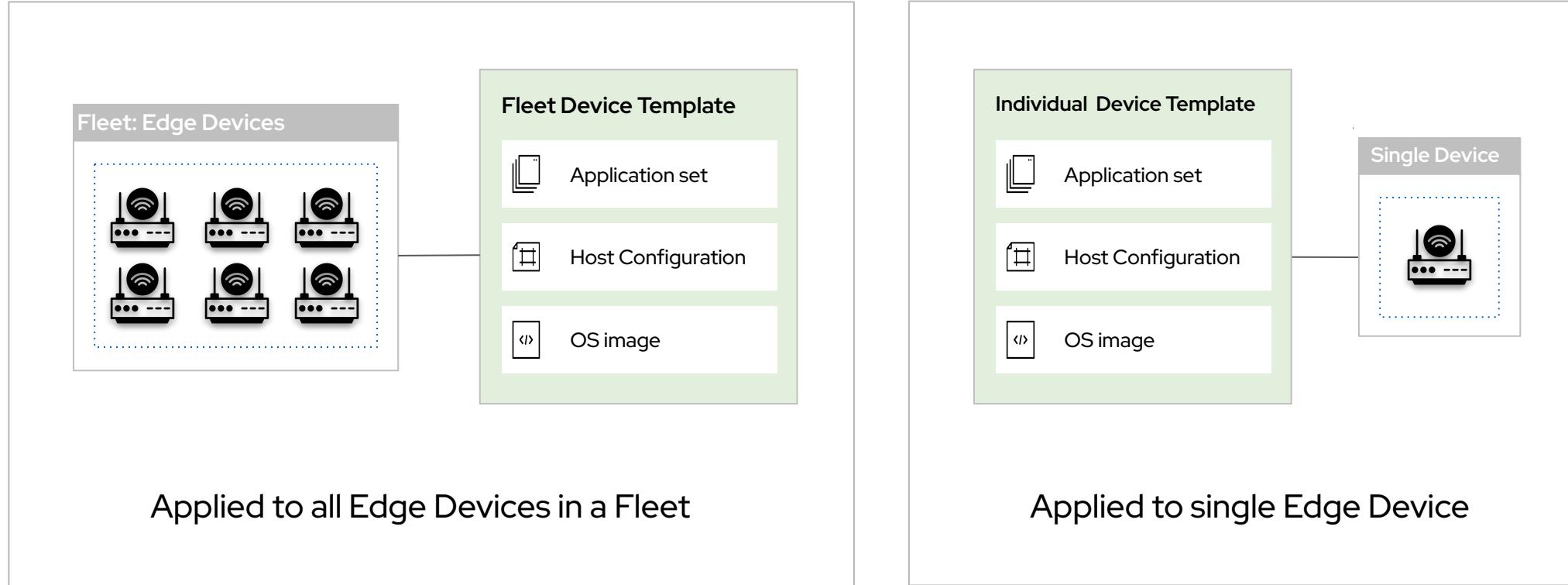
Device remote CLI access



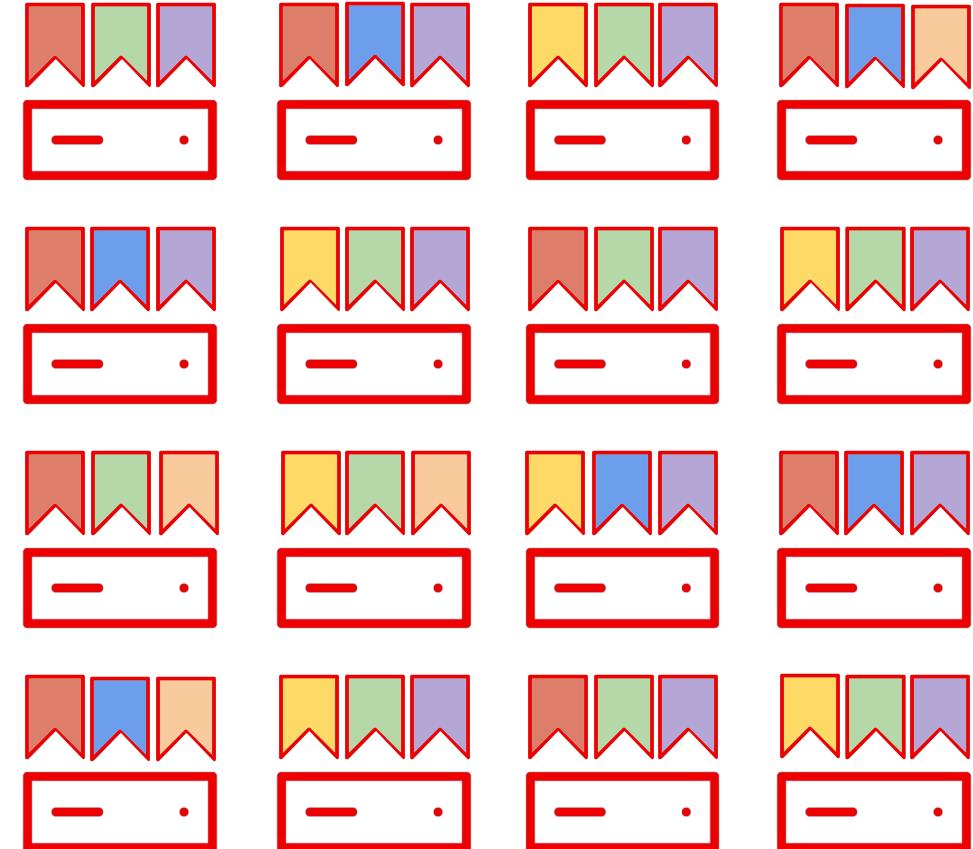
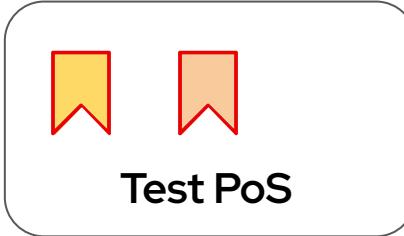
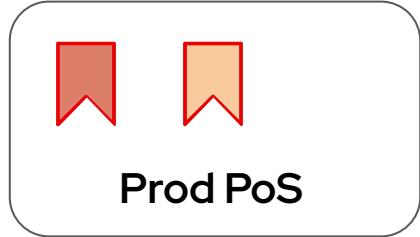
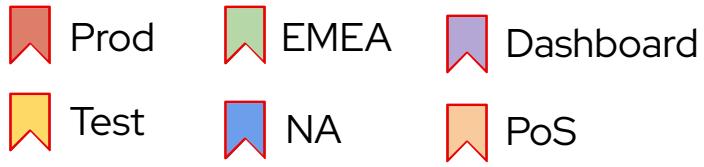
Fleet management

Fleets

Device templates

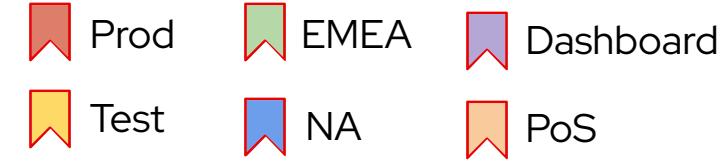


Using tags to create Fleets

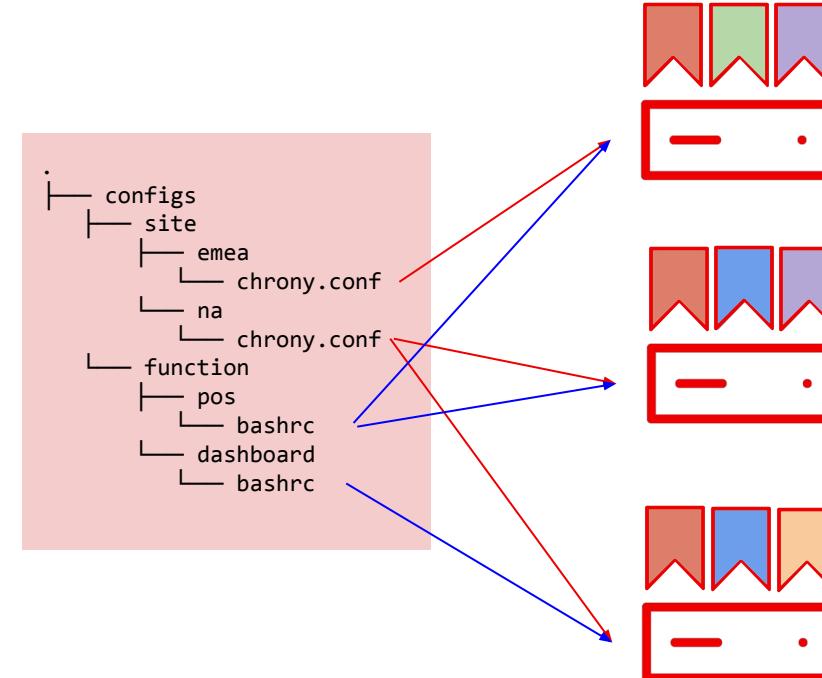


Device templates

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
```



Update strategy

Update strategy

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

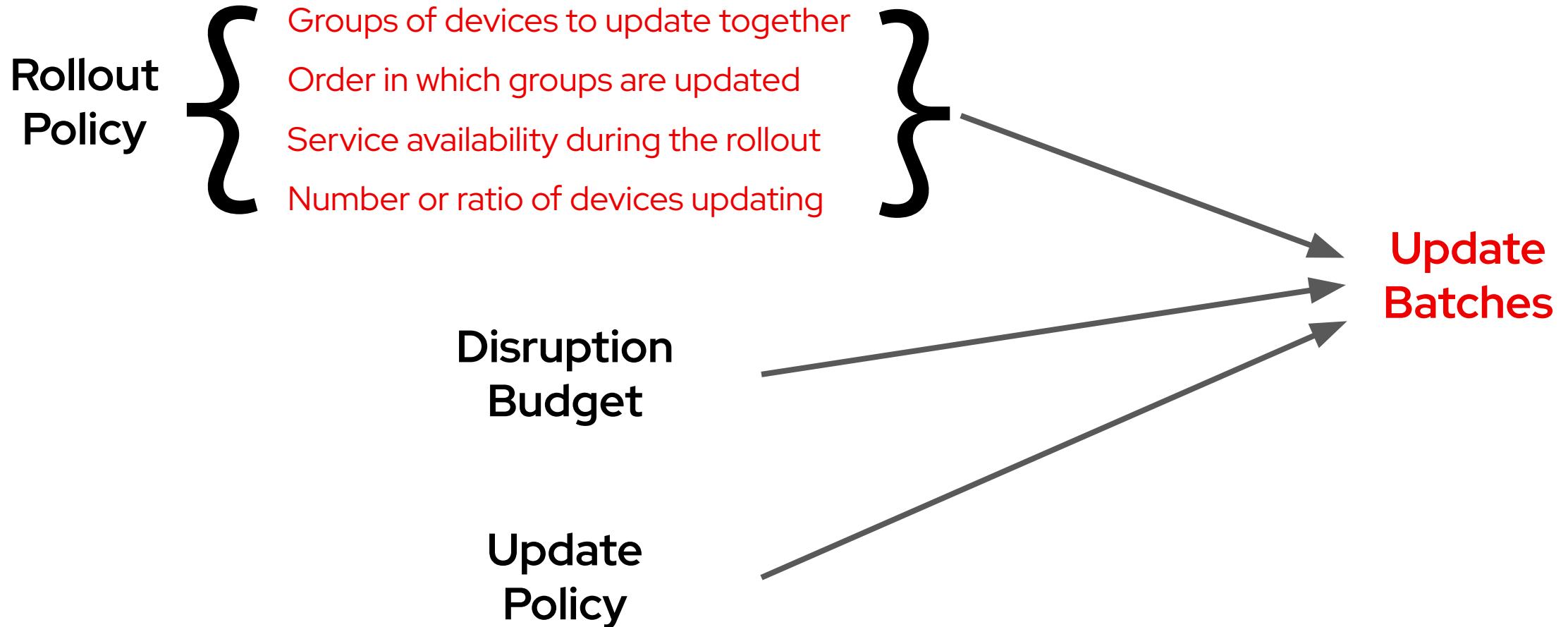
'Set disruption budget' and 'Set update policies' only applicable at Fleet level

The screenshot shows the 'Edit fleet' page in the Red Hat Edge Manager web interface. On the left, a sidebar lists steps: 1 General info, 2 Device template, 3 Updates (which is selected), and 4 Review and save. The main area has a title 'Edit fleet'. Under 'Advanced configurations', there are three checkboxes: 'Use basic configurations' (unchecked), 'Set rollout policies' (unchecked), 'Set disruption budget' (unchecked), and 'Set update policies' (unchecked). To the right of each checkbox is a callout box with a dashed arrow pointing to its description:

- 'Use basic configurations': Create one or more batches of devices based on device labels, incl. % success threshold before moving on to next batch
- 'Set rollout policies': Set limits on the number of devices that can update simultaneously
- 'Set update policies': Set combined (or separate) downloading and installing schedules (Daily | Weekly | time of day) for updates

At the bottom are 'Next', 'Back', and 'Cancel' buttons.

Update strategy



Summary

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

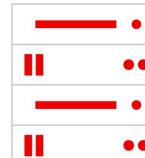
Employ mutual Transport Layer Security (mTLS) for robust and authenticated agent service communication. Establish a consistent security posture even for re-connected devices through rigorous identity verification.

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.



Thank you

