Red Hat Summit

Red Hat Device Edge Deep Dive

Red Hat | S&P Global Market Intelligence

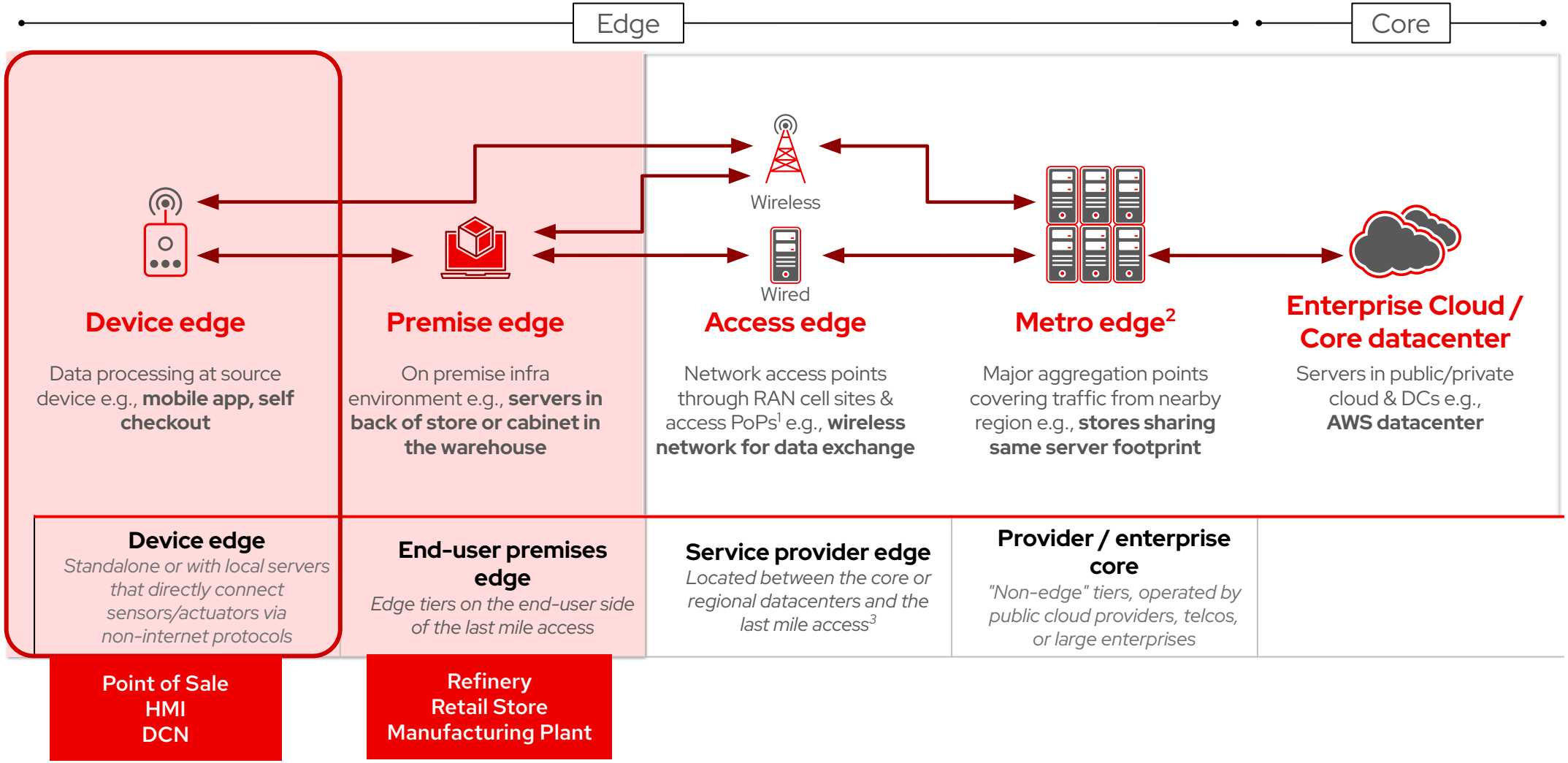
Agenda

- Red Hat's Approach to Edge
- What is Red Hat Device Edge?
- Overview of Image Builder + Demo
- Deploying a Composed Image
- Configuring Greenboot + Demo
- Building Applications into Images + Demo
- Deploying Microshift + Demo

Red Hat's Approach to Edge

Red Hat

So what and where is the edge?



1. Fixed network access equipment that connects from on premise to large aggregation centers; 2. Inclusive of potentially multiple layers of network including ISP data centers, edge data centers, etc.; 3. Commonly owned and operated by a telco or internet service provider and from which this provider serves multiple customers; 4. Mostly enterprise-grade
Source: BCG analysis

Retail Store Architecture

Back of Store
(Site Premise)

- AAP: network automation
- RHDE management
- ODF: data aggregation
- Virtualization: legacy workloads

Store Network

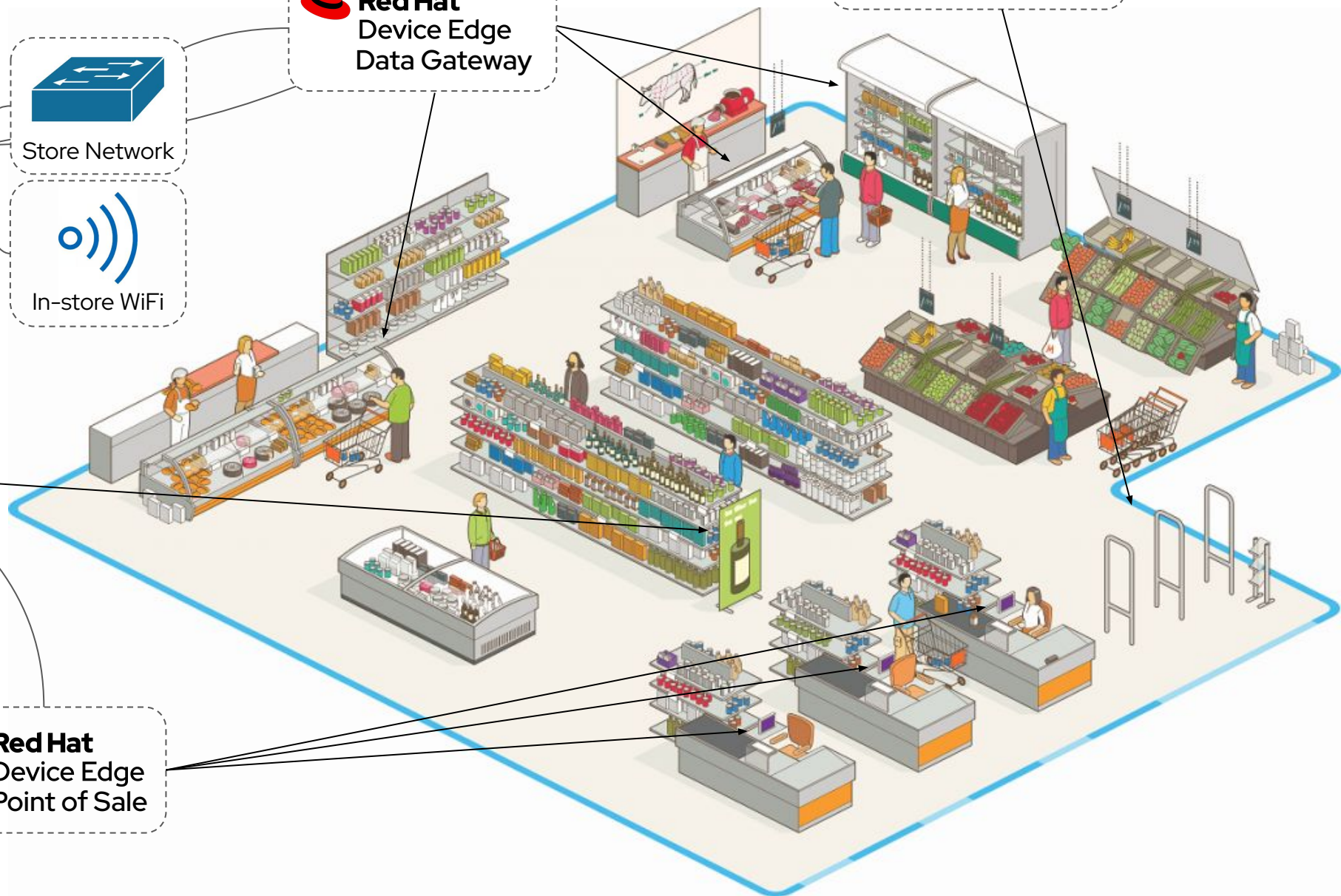
In-store WiFi

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



One Consistent Platform Across the Industrial Site

Red Hat Advanced Compute Platform

- Full IOT Automation: Network to PLCs

- Management of Device Edge Deployments

- Legacy and Next Generation Workload hosting

Plant Network

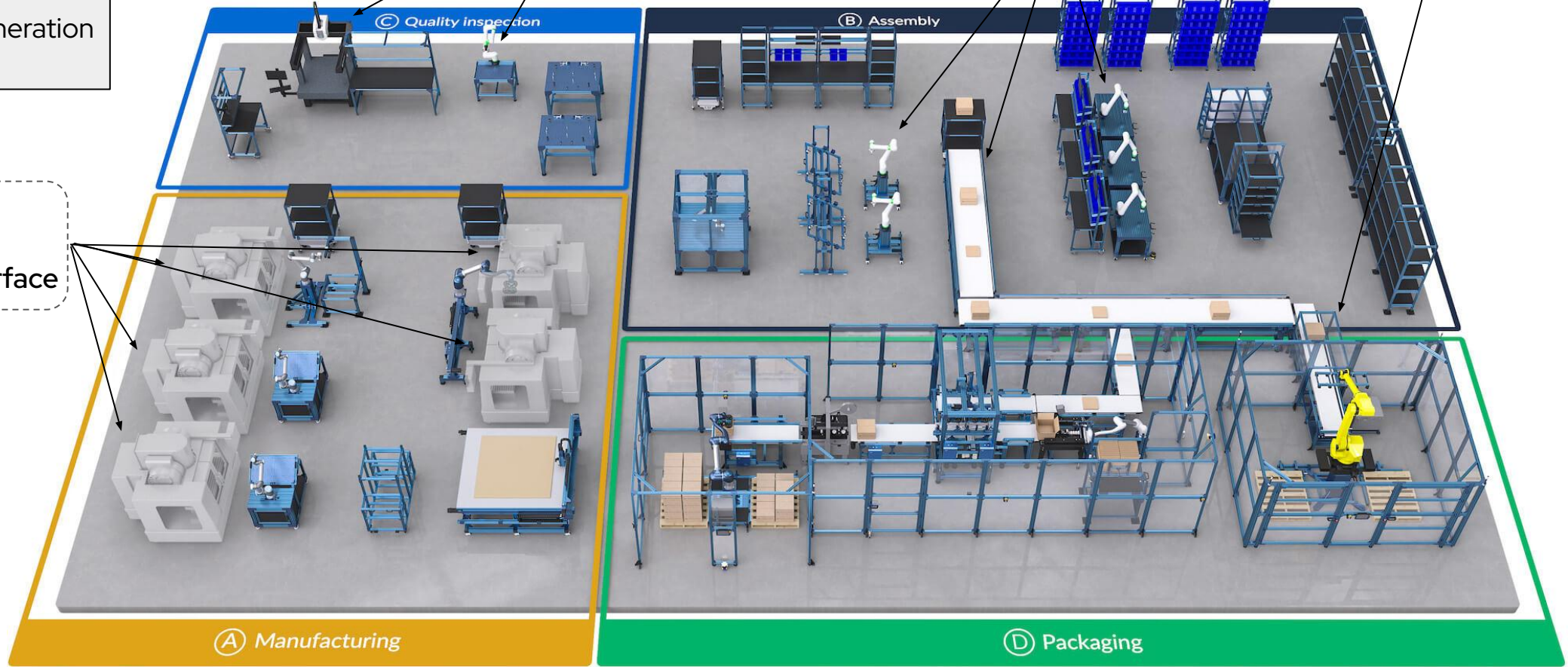
Process Network

Red Hat Device Edge Data Gateway

Red Hat Device Edge Distributed Control Node

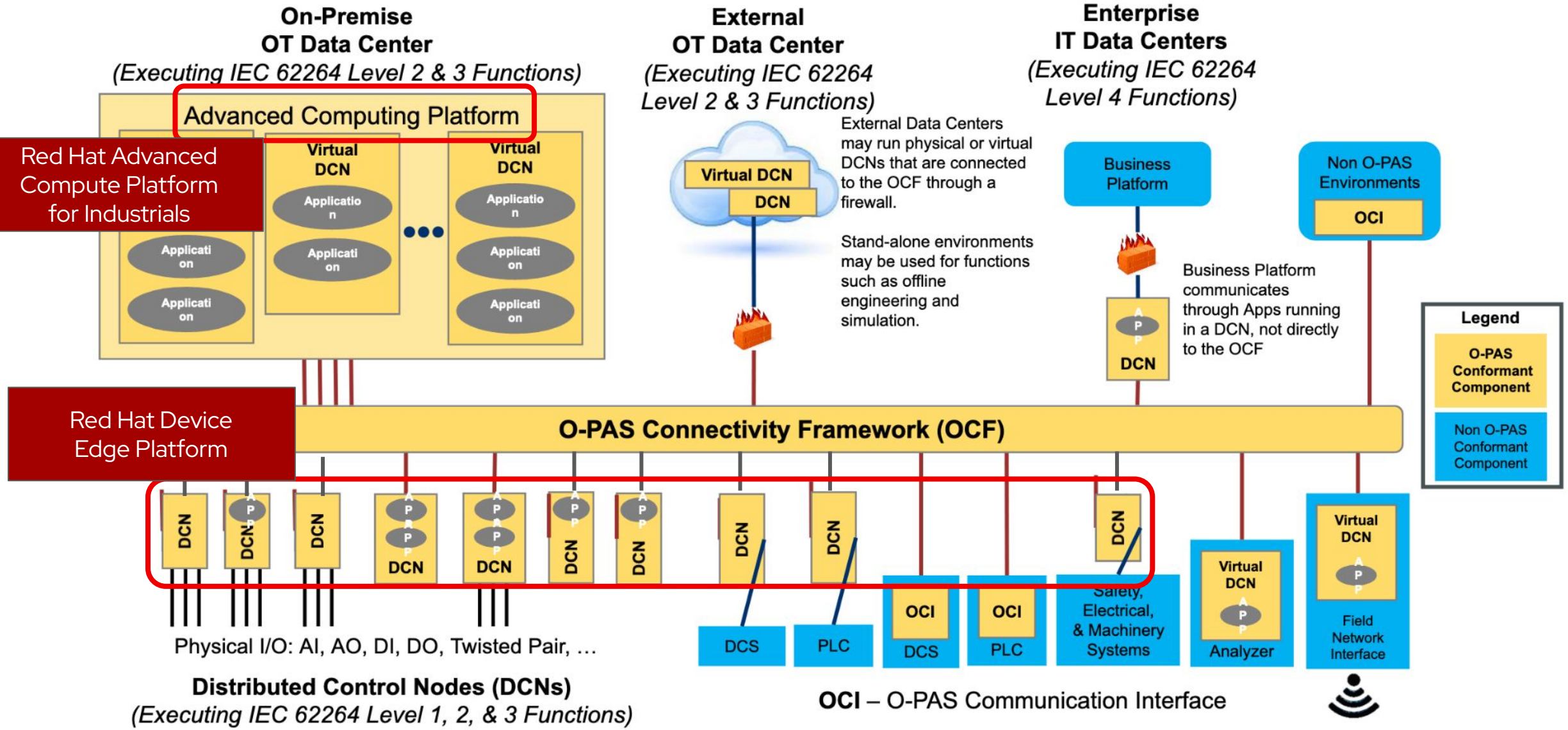
Red Hat Device Edge Computer Vision

Red Hat Device Edge Human Machine Interface



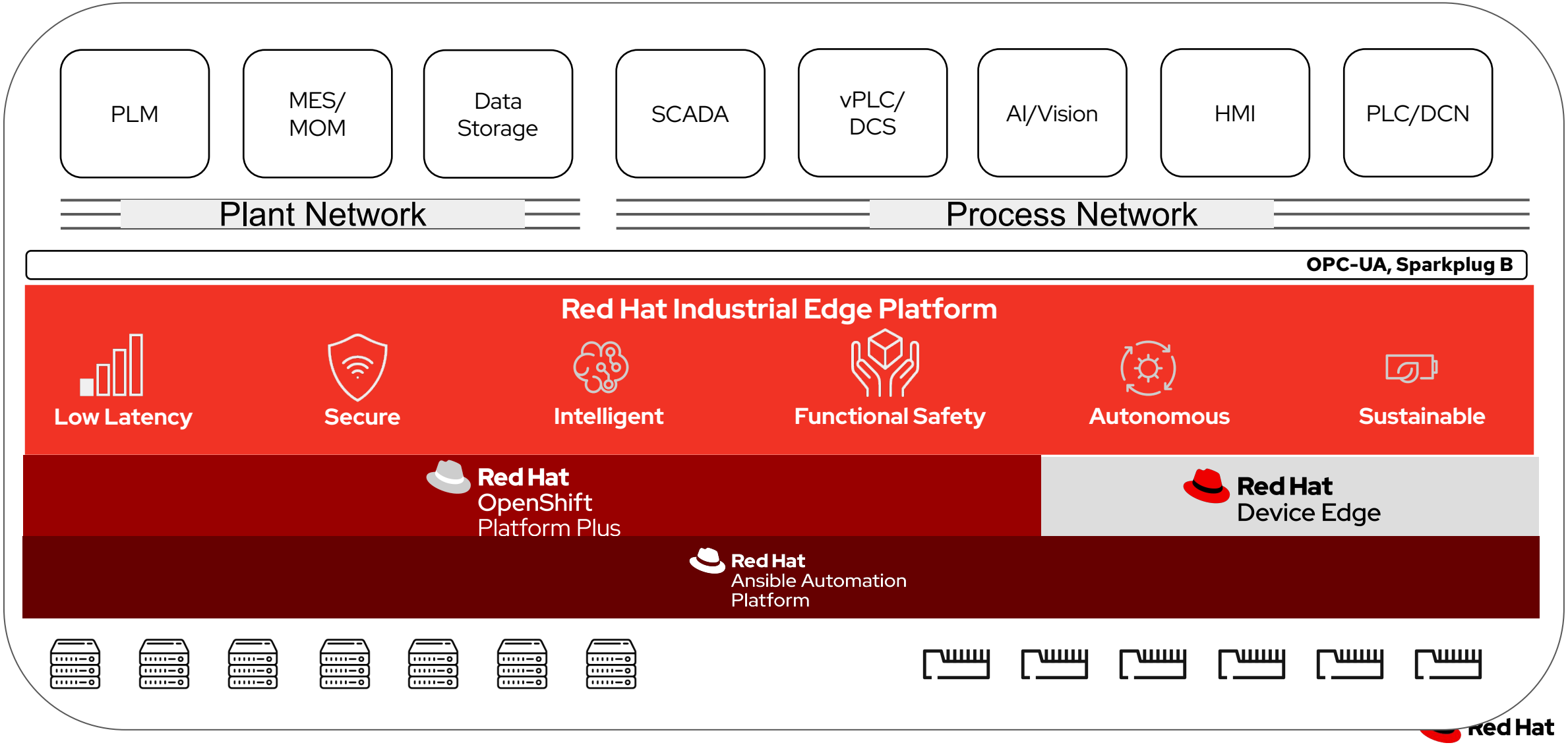
Alignment and Leadership of Industry Direction

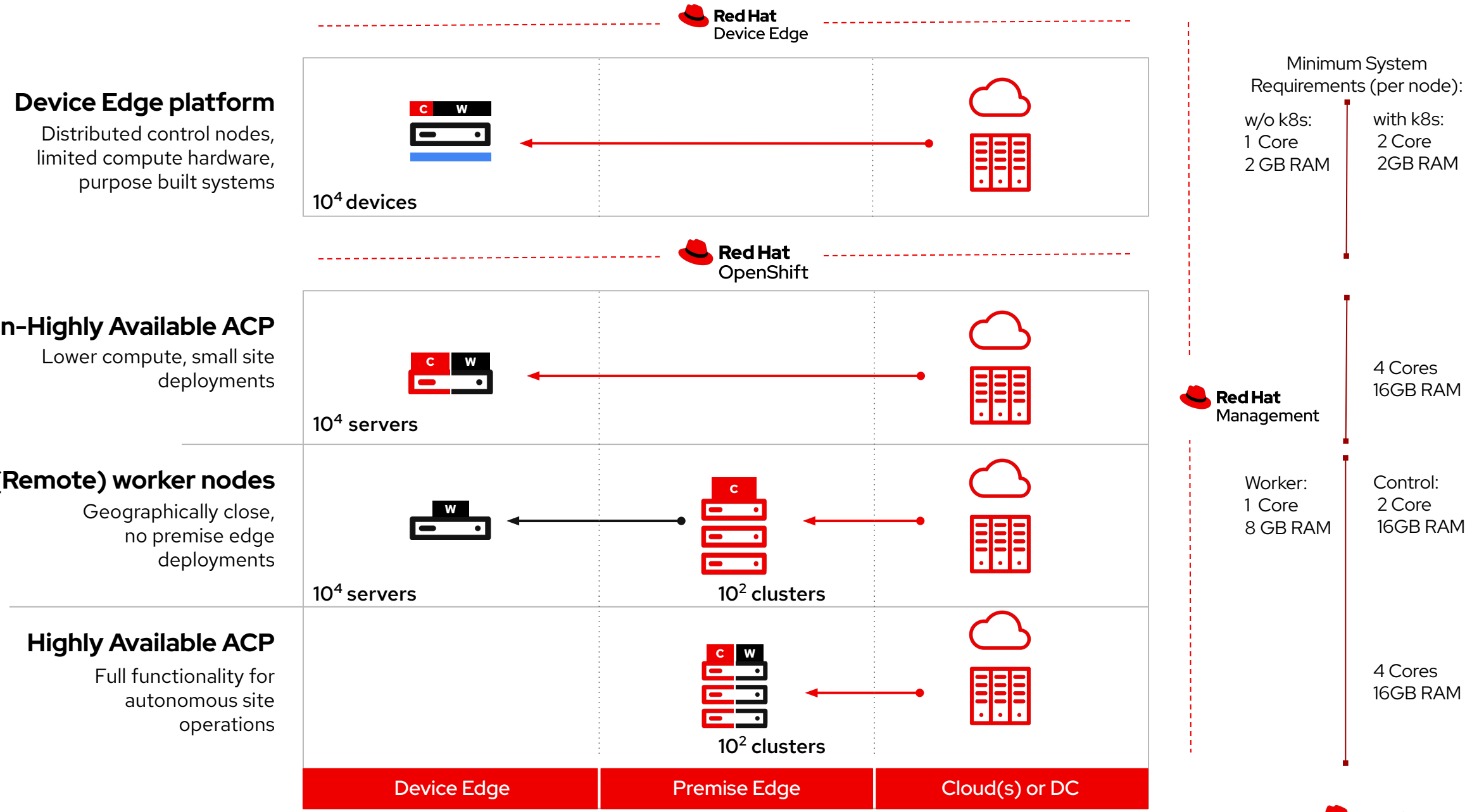
THE OPEN GROUP
OPEN PROCESS AUTOMATION™ FORUM



A Next Generation Approach to Industrial Operational Technology

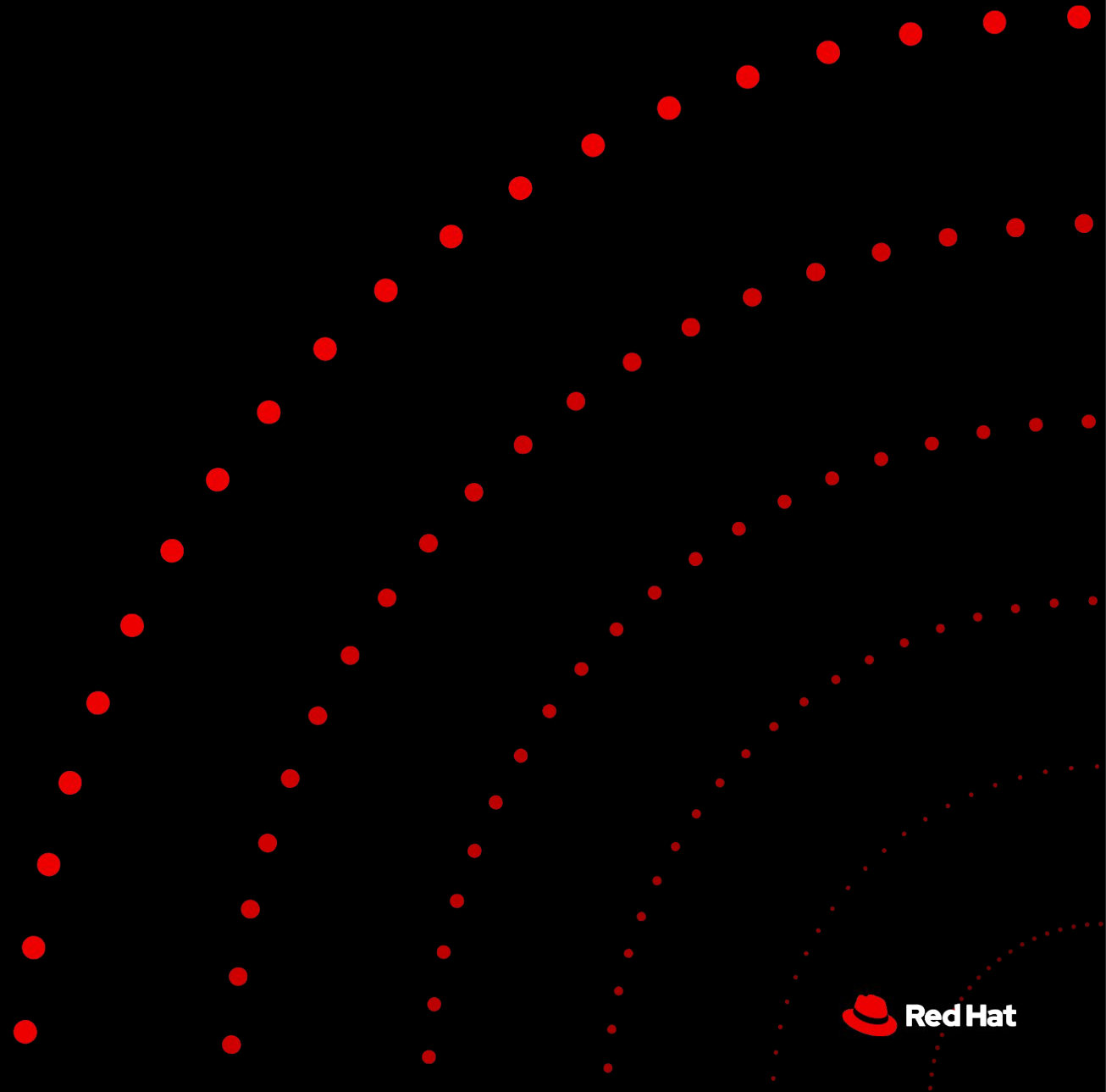
One Open Platform for Deterministic and Non-Deterministic Workloads




Red Hat
Device Edge
Device Edge platform
Distributed control nodes, limited compute hardware, purpose built systems
C
W
10^4 devices
Red Hat
OpenShift
Non-Highly Available ACP
Lower compute, small site deployments
10^4 servers
(Remote) worker nodes
Geographically close, no premise edge deployments
10^4 servers
10^2 clusters
Highly Available ACP
Full functionality for autonomous site operations
10^2 clusters
Device Edge
Premise Edge
Cloud(s) or DC
Minimum System Requirements (per node):
w/o k8s:
1 Core
2 GB RAM
with k8s:
2 Core
2GB RAM
Red Hat
Management
4 Cores
16GB RAM
Worker:
1 Core
8 GB RAM
Control:
2 Core
16GB RAM
4 Cores
16GB RAM
Cluster management and application deployment
Kubernetes node control
C
Control node
W
Worker node

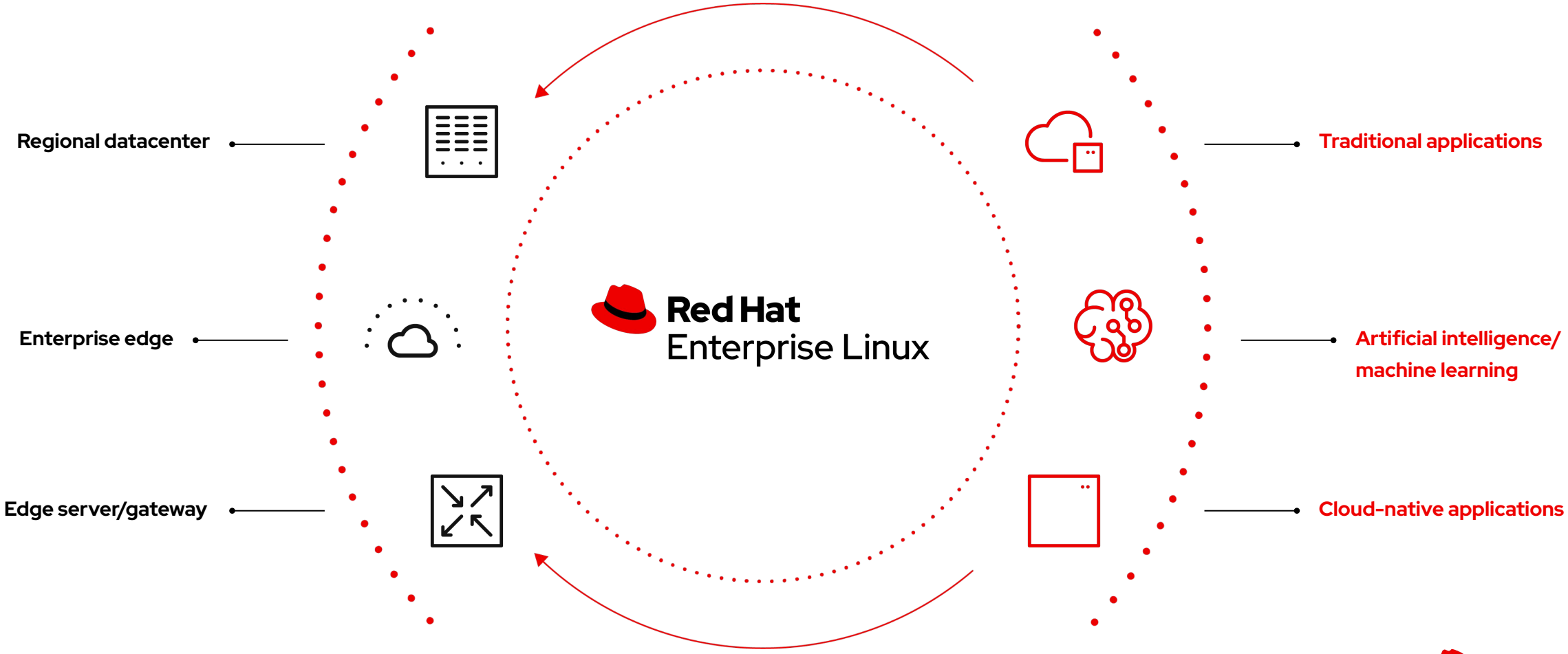
Red Hat

Red Hat Device Edge



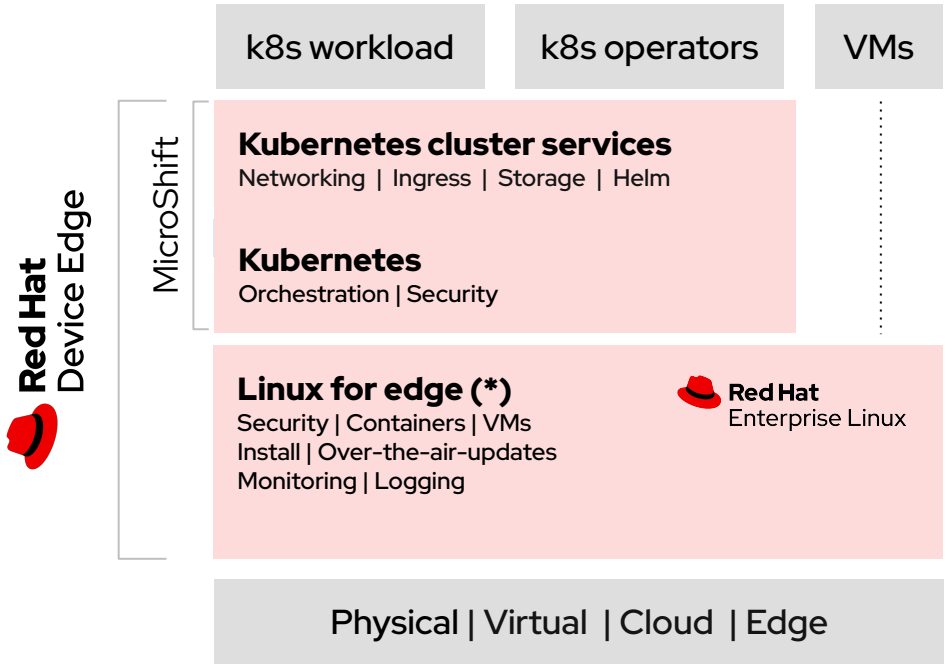
Red Hat

Flexibility and freedom to run workloads where they're needed



Red Hat

Device Edge Technical Overview

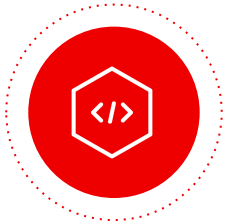


* recommended for edge deployments: Red Hat Enterprise Linux for Edge Images, rpm-ostree, immutable, atomic upgrade, over the air flavour of Red Hat Enterprise Linux.

Red Hat

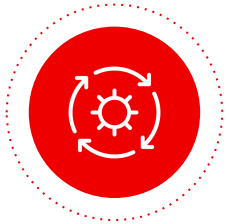
Red Hat Enterprise Linux for Edge

Ensured stability and deployment flexibility



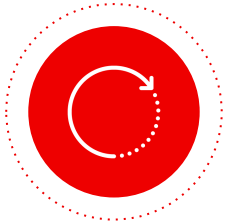
Quick image generation

Easily create purpose-built OS images optimized for the architectural challenges of the edge.



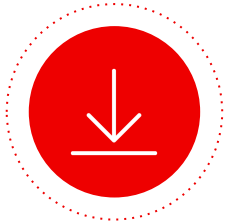
Edge management

Secure and scale with the benefits of zero-touch provisioning, fleet health visibility, and security remediations throughout the entire lifecycle.



Efficient over-the-air updates

Updates transfer significantly less data and are optimized for remote sites with limited or intermittent connectivity.

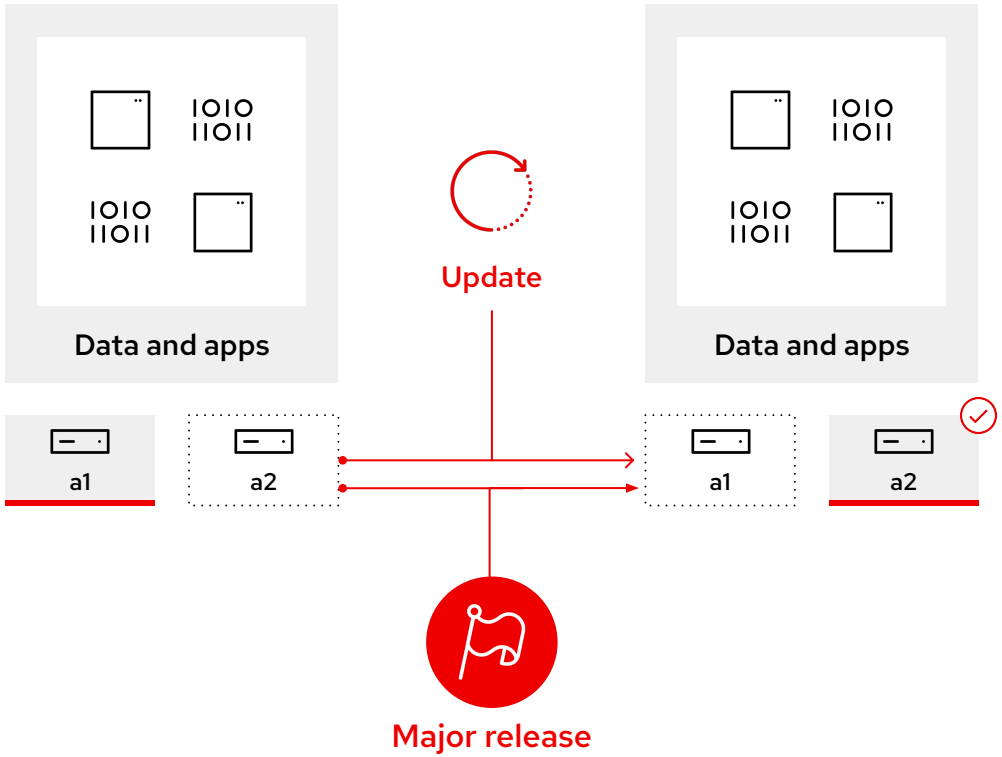


Intelligent rollbacks

Application-specific health checks detect conflicts and automatically reverts to last working OS update, preventing unplanned downtime.

rpm-ostree

Immutable OS and stateful config and storage

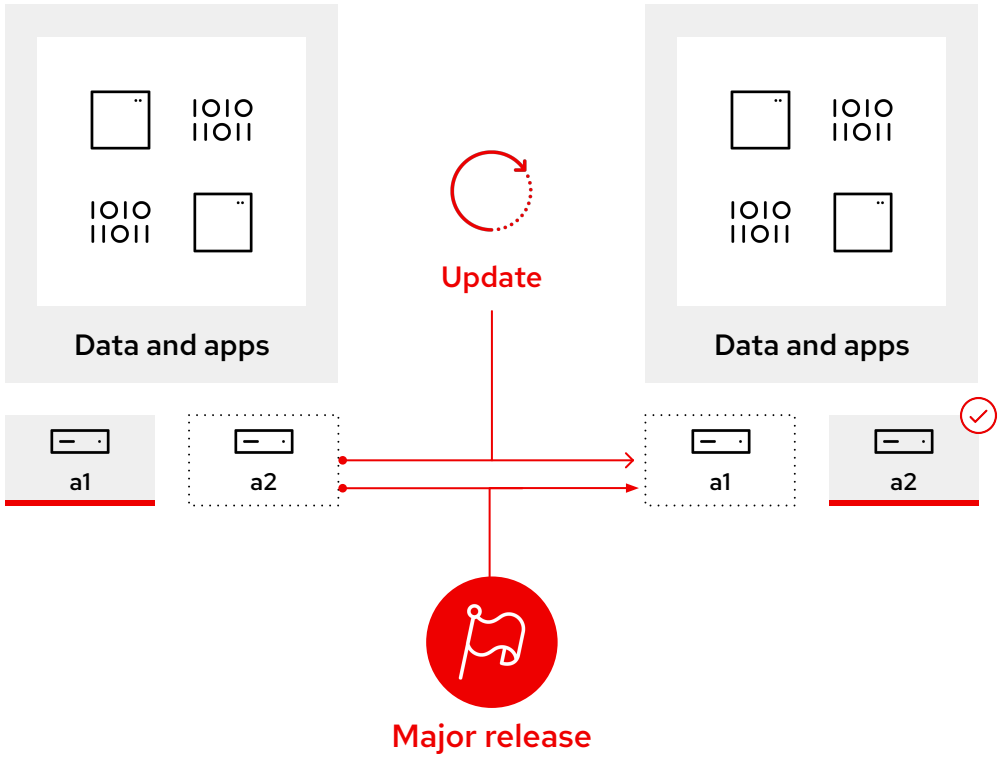


Transactional updates (A → B model)

- OS binaries and libraries (/usr*) are immutable and read-only.
- State (r/w) is maintained in /var and /etc.
- No inbetween state during updates.
- Updates are staged in the background and applied upon reboot.
- Reboots can be scheduled with maintenance windows to ensure the highest possible uptime.

rpm-ostree

Immutable OS and stateful config and storage

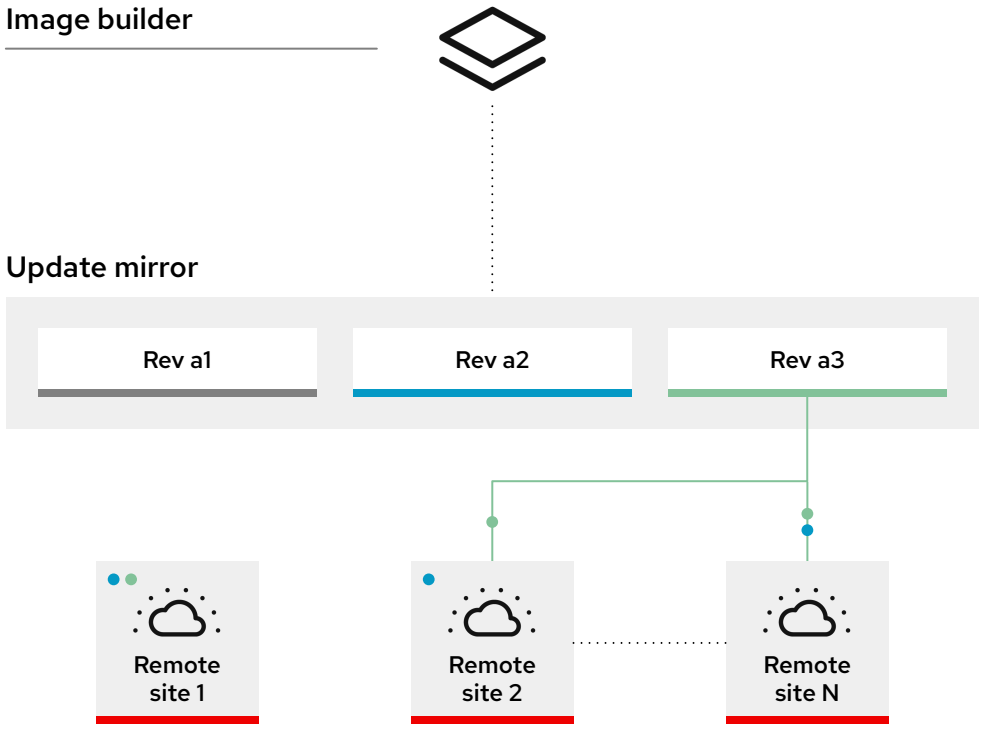


Enables seamless major release upgrades (Red Hat Enterprise Linux 8→ 9)

- Help extend the serviceable life of hardware in the field.

Efficient over the air (OTA) OS updates

Easy remote device mirroring: transfer only the deltas



- Ideal for disconnected, intermittent, or low-bandwidth (DIL) connections
- Transfers significantly less data over the network*
- Only transfers updated bits of OS content
- Static-deltas can be created to further reduce network usage
- Client initiated connections for firewall-friendly experience

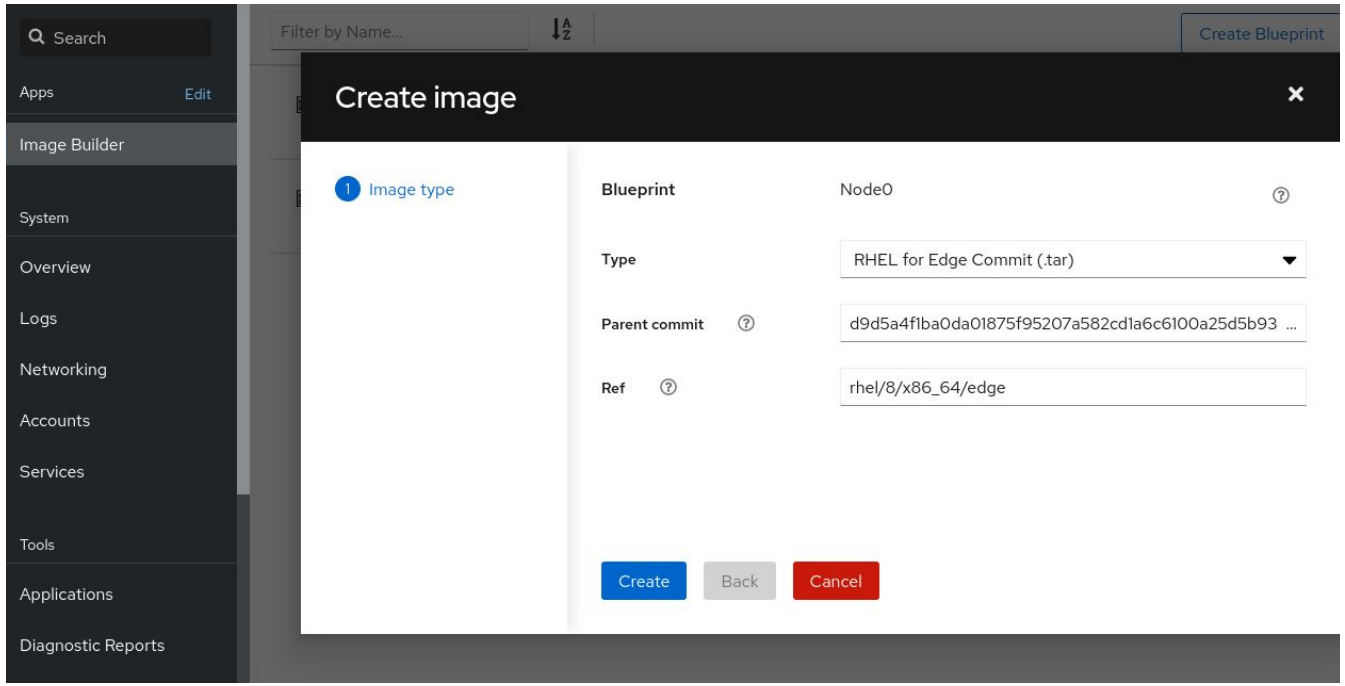
* When compared with traditional appliance image updates or individual package management strategies

Red Hat

Image Builder Overview + Demo.

Image builder

Fast image assembly and configuration



Edge profile generates a small, rpm-ostree image from the latest Red Hat Enterprise Linux 8.3+

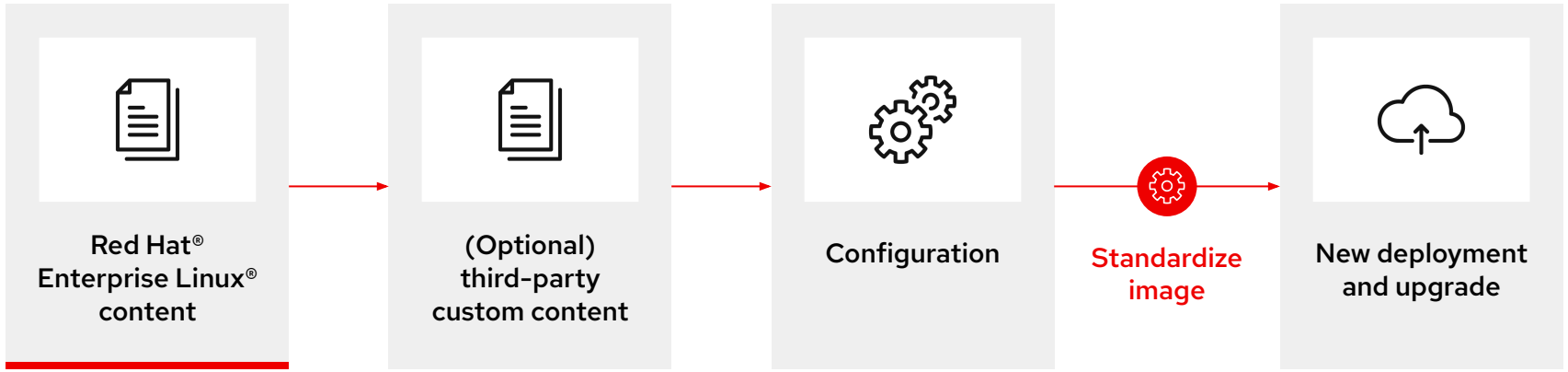
OS contents include:

- @core packages (small base install)
- Podman as the container engine
- Additional RPM content (optional)

Red Hat

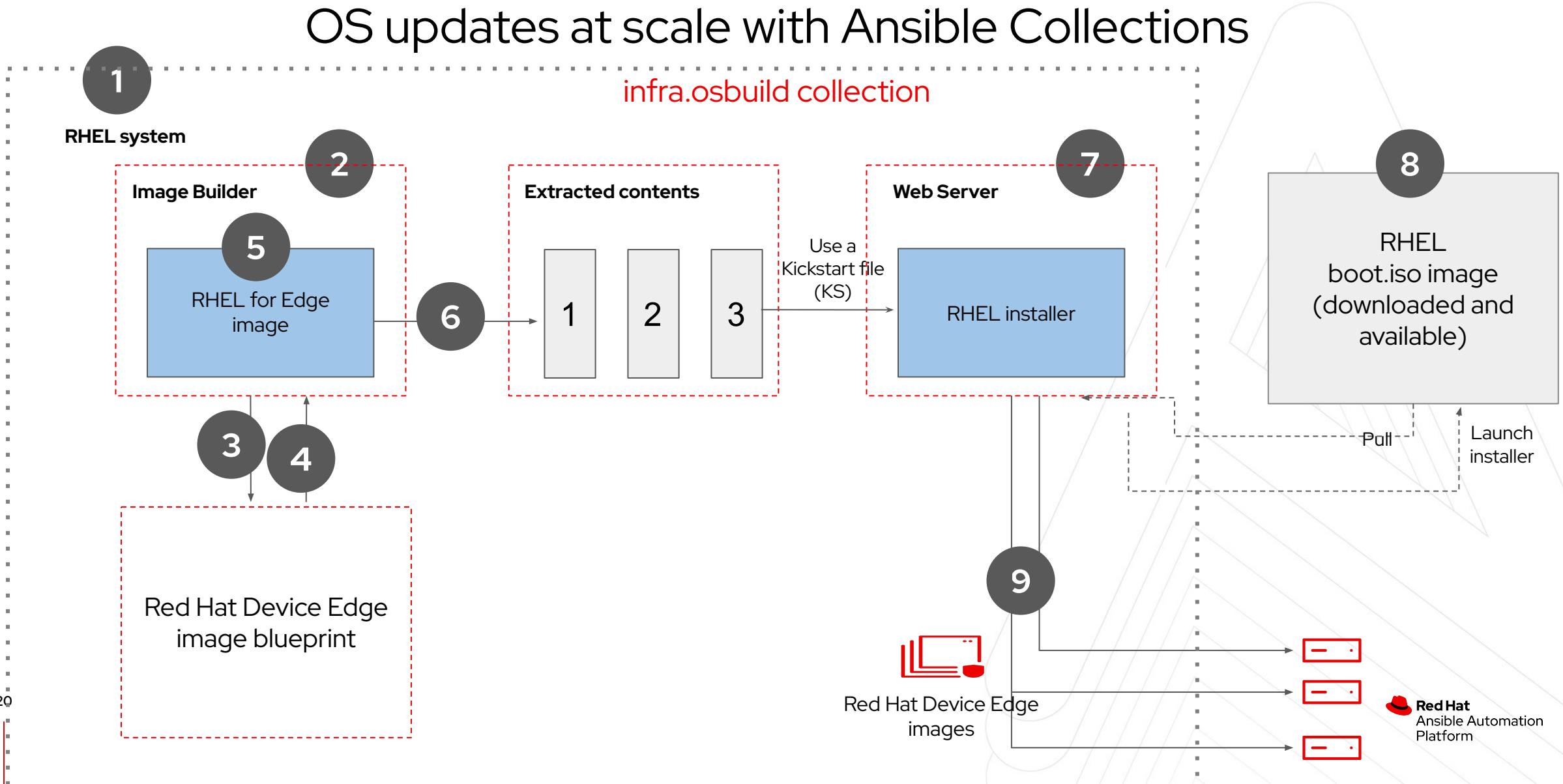
Image Builder Workflow

Standardize your fleet with image sets to ensure uniform performance



Red Hat

OS updates at scale with Ansible Collections



Deploying microshift containers on top of OS at scale

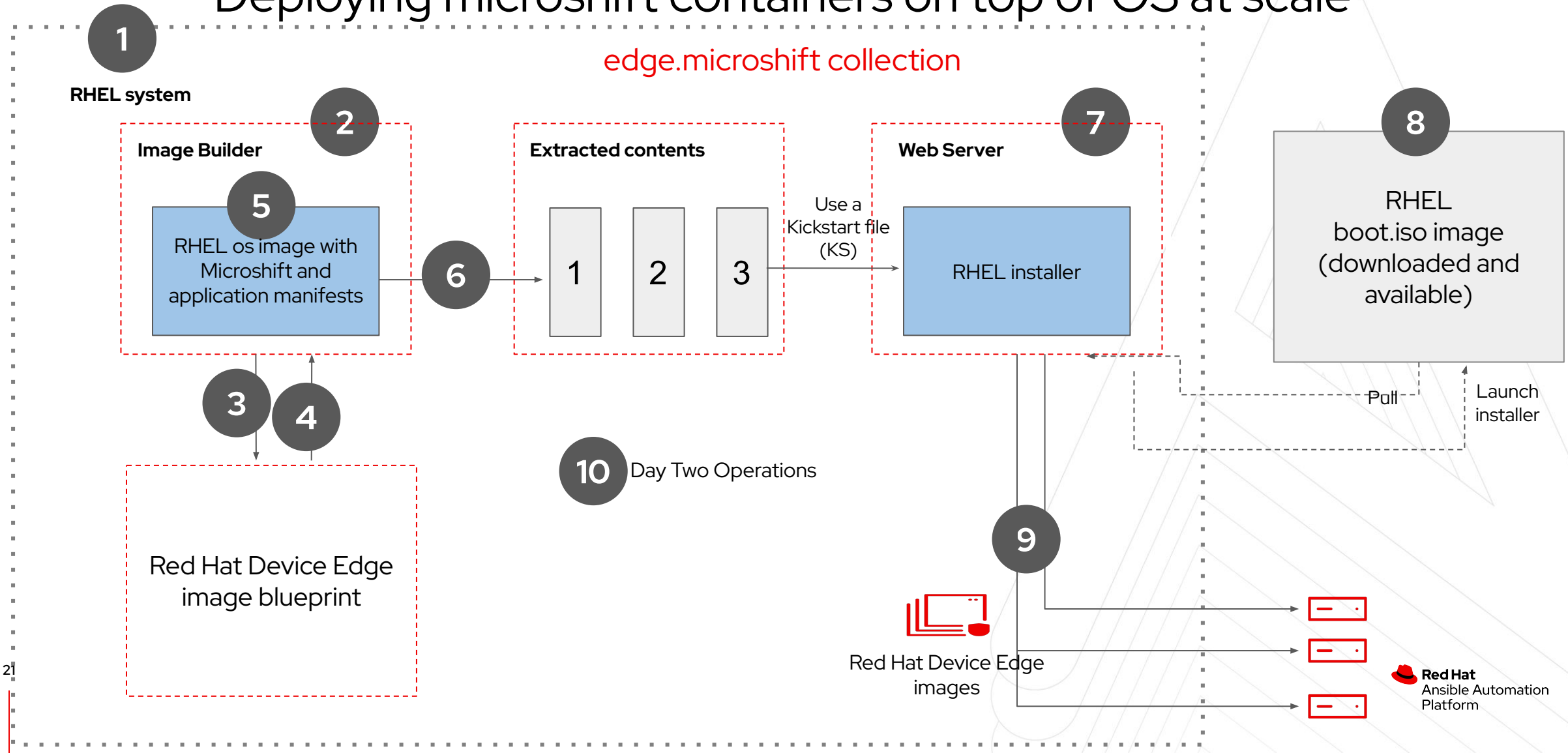


Image Builder Demo

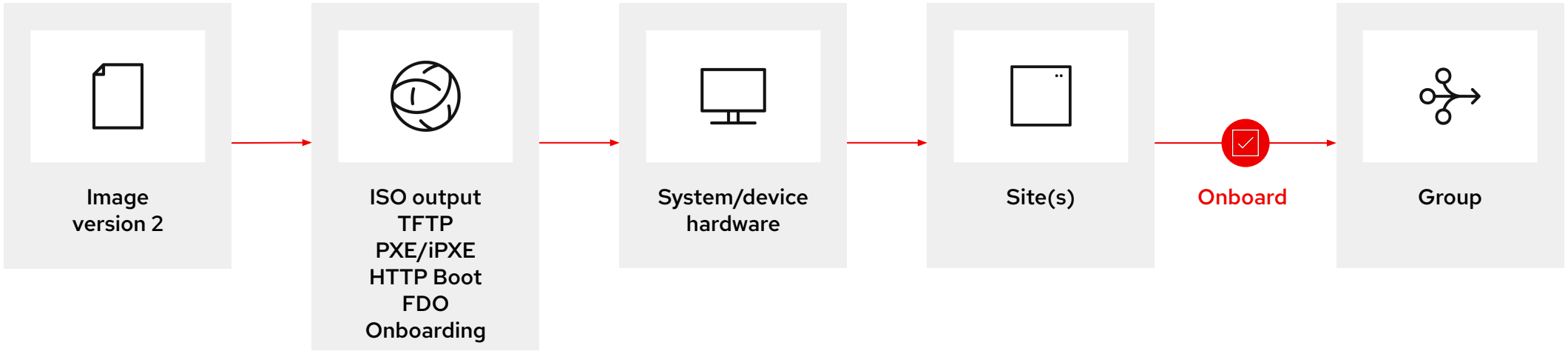
- Image Builder WebUI
- Automating Image Builder with Ansible

Deploying Red Hat Device Edge

Red Hat

Zero-touch onboarding

Systems onboard automatically at boot and can be sent directly to the site



Red Hat

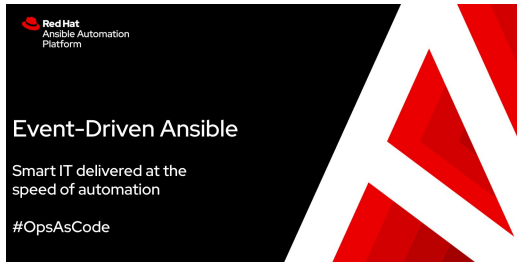
Initial Install Process and Call Home

1. Device is booted from installation source (USB/http/uefi)

2. Device installs RHEL for Edge without prompt/intervention

3. Device reboots and calls home to Ansible EDA

4. Ansible EDA triggers a workflow in Ansible Controller with host information



5. Device calls home to Controller to create a host entry in an Inventory and kick off provisioning workflow

6. Controller applies device configuration and pushes down edge workload

Red Hat

Call Home with Minimal Information

Using Systemd to run on next boot:

```
[Unit]
Description=Connect to WiFi
After=network.target
ConditionPathExists=!/var/tmp/wifi-connected

[Service]
Type=oneshot
ExecStartPre=/usr/bin/nmcli radio wifi on
ExecStartPre=/usr/bin/sleep 5
ExecStartPre=/usr/bin/nmcli dev wifi rescan
ExecStartPre=/usr/bin/sleep 5
ExecStartPre=/usr/bin/nmcli dev wifi list
ExecStart=/usr/bin/nmcli dev wifi connect lab-wifi password
'example-password'
ExecStopPost=/usr/bin/touch /var/tmp/wifi-connected
---
[Install]
WantedBy=default.target


[Unit]
Description=Register to Ansible Automation Platform
After=network.target
After=connect-wifi.service
ConditionPathExists=!/var/tmp/aap-registered

[Service]
Type=oneshot
ExecStart=/bin/bash /var/tmp/aap-auto-registration.sh
ExecStopPost=/usr/bin/touch /var/tmp/aap-registered

[Install]
WantedBy=default.target
EOF
```

A simple curl script:

```
#!/bin/bash
IP ADDRESS=\$(nmcli conn show lab-wifi | grep ip address | awk '{print \$4}')
MAC ADDRESS=\$(ip addr | grep wlp -A 1 | grep link | awk '{print \$2}' | sed
's/://g')
STUDENT='1'

JSON="{\
\"ip address\": \"\$IP ADDRESS\", \
\"other var\": \"\$OTHER VAR\", \
\"mac_address\": \"\$MAC_ADDRESS\" \
}"

/usr/bin/curl -H 'Content-Type: application/json' --data "\$JSON"
https://eda.device-edge.redhat-workshops.com/endpoint
```

Execution Flow:

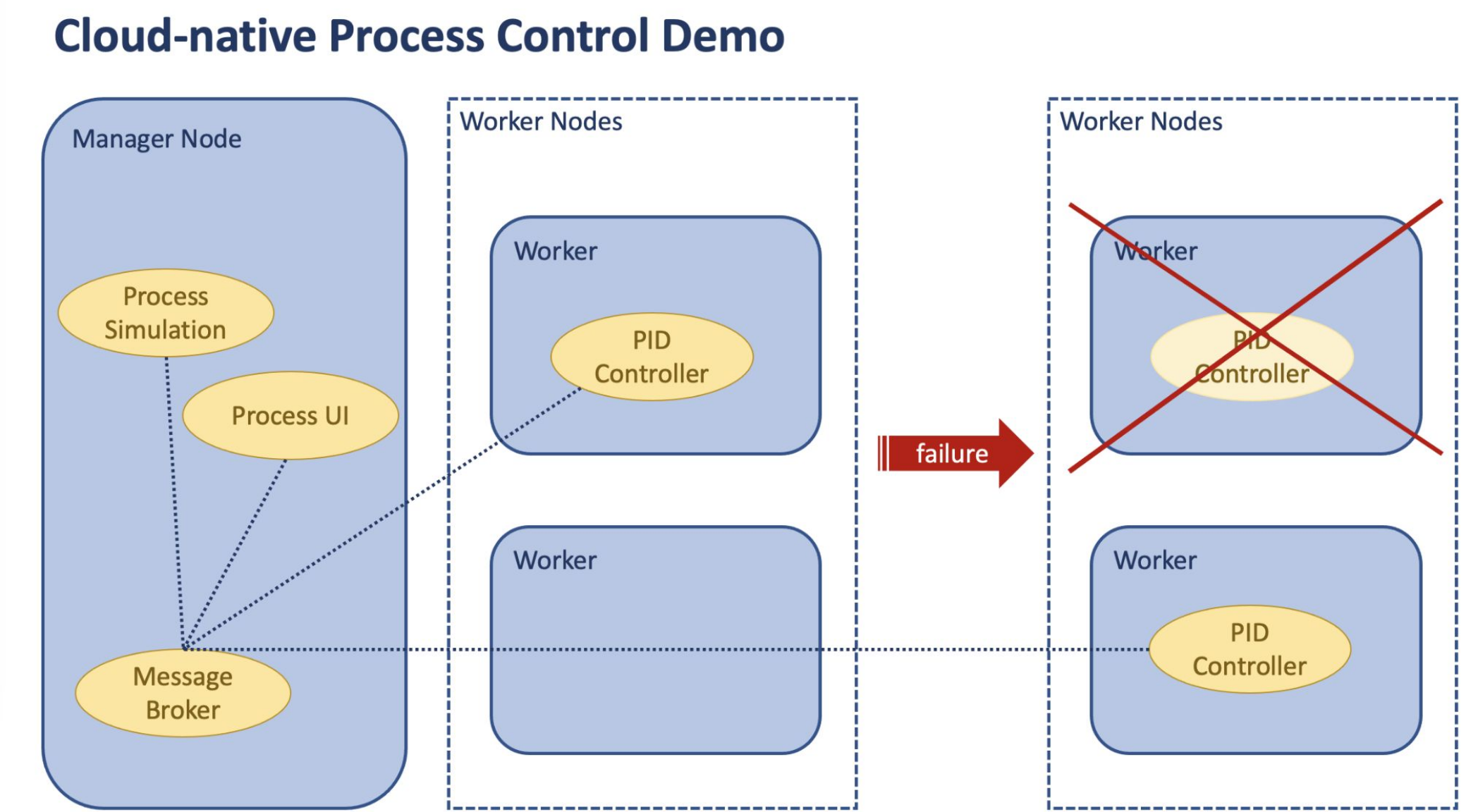
1. System connects to network (WiFi)
2. System collects connection information
3. System calls home to Ansible EDA
4. Ansible EDA calls workflow with system information
5. Device is onboarded

Deploying Red Hat Device Edge Demo

Red Hat

Embedding an Application into an Image

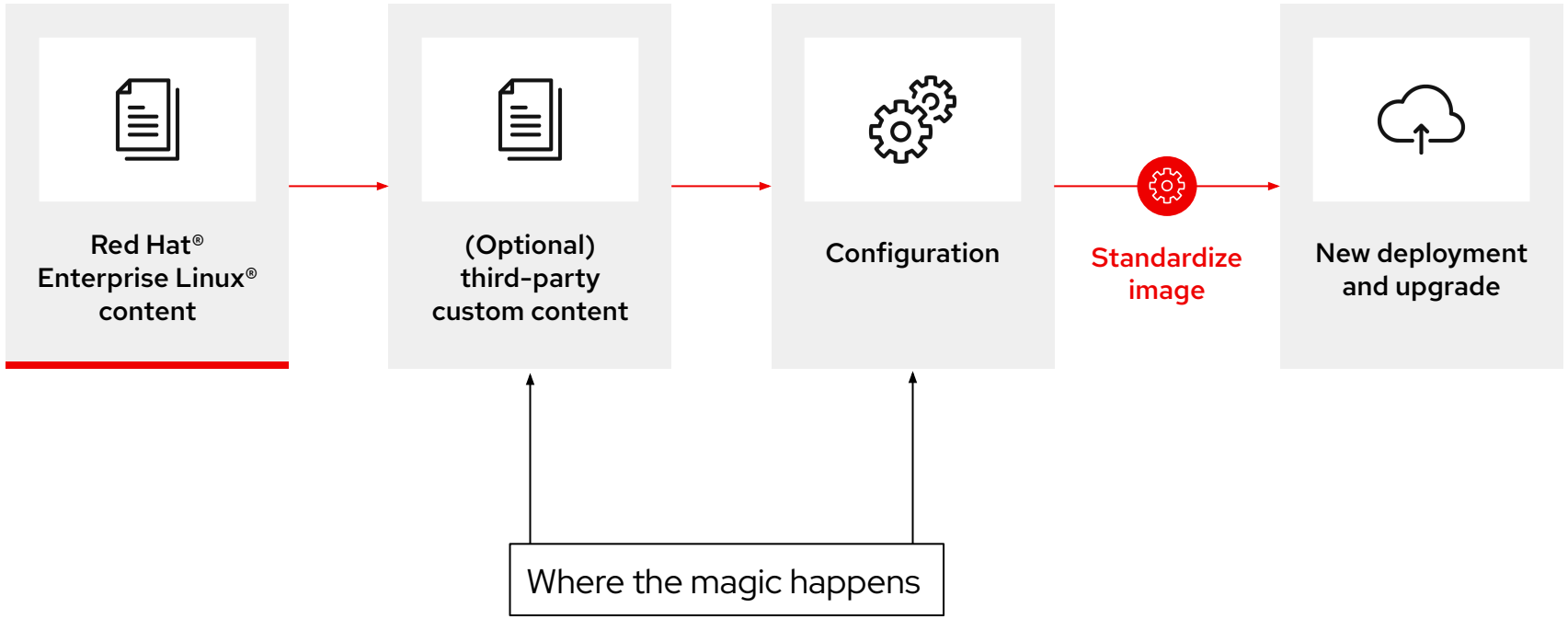
Our Example Process Control Workload



Note: Our DCN runs the control plane and worker plane on one system

Image Builder Workflow

Standardize your fleet with image sets to ensure uniform performance



Red Hat

Embedding an Application Examples

[[containers]] is used to specify container images that should be embedded into the image by Image Builder. The images will be available in the default image storage location for the operating system. On RHEL, images will be visible by podman using the `podman images` command.

Note: If 'name' is specified, then the container name will be re-written, otherwise the image name will match the source.

Using the infra.osbuild collection:

```
builder compose_containers:
  - name: mqtt
    source: quay.io/device-edge-workshops/process-control-mqtt:1.0.0
  - name: simulate
    source: quay.io/device-edge-workshops/process-control-simulate:1.0.0
  - name: control
    source: quay.io/device-edge-workshops/process-control-control:1.0.0
  - name: ui
    source: quay.io/device-edge-workshops/process-control-ui:1.0.0
```

Directly in toml:

```
[[containers]]
name = "mqtt"
source = "quay.io/device-edge-workshops/process-control-mqtt:1.0.0"

[[containers]]
name = "simulate"
source = "quay.io/device-edge-workshops/process-control-simulate:1.0.0"

[[containers]]
name = "control"
source = "quay.io/device-edge-workshops/process-control-control:1.0.0"

[[containers]]
name = "ui"
source = "quay.io/device-edge-workshops/process-control-ui:1.0.0"
```

Running a Workload with Podman

Kube YAML

`podman generate kube [UUID]`

```
apiVersion: v1
kind: Pod
Metadata:
  annotations:
    io.podman.annotations.ulimit:
nofile=524288:524288
  labels:
    app: thirstywilson-pod
  name: thirstywilson-pod
spec:
  containers:
  - image: ubi9/nginx-120
command: [“nginx”]
  args: [“g”, “daemon off;”]
    name: thirstywilson
    ports:
    - containerPort: 8080
      hostPort: 8080
    stdin: true
    tty: true
```

systemd unit file

`podman generate systemd --new [UUID]`

```
[Unit]
Description= nginx container
After=network-online.target

[Service]

Restart=on-failure
ExecStart=/usr/bin/podman run \
--cidfile=%t/%n.ctr-id \
--cgroups=no-conmon \
--rm \
--sdnotify=conmon \
-d \
-p 8080:8080 \
ubi9/nginx-120 nginx -g “daemon off;”

[Install]
WantedBy=multi-user.target
```

Quadlet

`/etc/containers/systemd/nginx.container`

```
[Service]
Restart=always

[Container]
ContainerName=nginx
Image=ubi9/nginx-120
PublishPort=8080:8080
Exec=nginx -g "daemon off;"

[Install]
WantedBy=default.target
```

Embedding an Application Examples

Generated quadlet systemd unit file:

```
[Install]
WantedBy=default.target

[Unit]
After=network-online.target
SourcePath=/etc/containers/systemd/process-control.kube
RequiresMountsFor=%t/containers

[X-Kube]
Yaml=/etc/containers/systemd/process-control.yaml
PublishPort=1883:1883

[Service]
KillMode=mixed
Environment=PODMAN_SYSTEMD_UNIT=%n
Type=notify
NotifyAccess=all
SyslogIdentifier=%N
ExecStart=/usr/bin/podman kube play --replace
--service-container=true --log-driver passthrough --publish 1883:1883
/etc/containers/systemd/process-control.yaml
ExecStop=/usr/bin/podman kube down
/etc/containers/systemd/process-control.yaml
```

Note: Quadlet generates files when called at runtime, so generate the files on a test system before embedding them via Image Builder.

kube yaml:

```
---
apiVersion: v1
kind: Pod
metadata:
  name: process-control
spec:
  containers:
    - name: mqtt
      image: docker.io/library/mqtt:latest
    - name: simulate
      image: docker.io/library/simulate:latest
    - name: control
      image: docker.io/library/control:latest
    - name: ui
      image: docker.io/library/ui:latest
      ports:
        - containerPort: 1881
          hostPort: 1881
```

Quadlet kube file:

```
[Install]
WantedBy=default.target

[Unit]
After=network-online.target

[Kube]
Yaml=process-control.yaml # Path to kube yaml
PublishPort=1881:1881 # External ports
```

Embedding an Application Examples

Also ensure target directories are created, services are set to start on boot, and firewall ports are allowed

Directly in toml:

```
[customizations.directories]
path = "/etc/containers/systemd"
mode = "0755"
user = "root"
group = "root"
ensure_parents = false

[customizations.firewall]
ports = “1881:tcp”

[customizations.services]
enabled = ["process-control”]
```

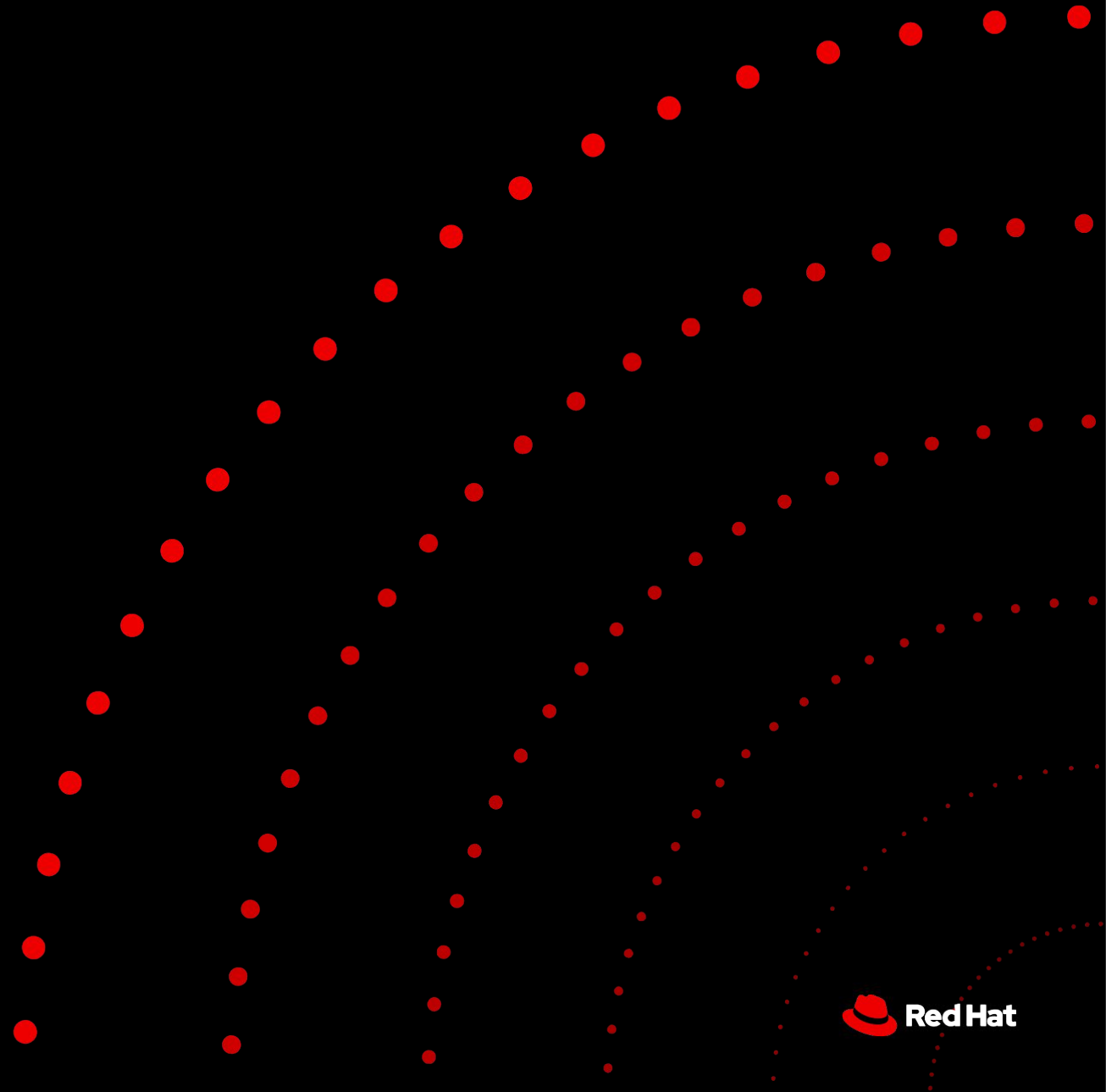
Using the infra.osbuild collection:

```
builder_compose_customizations:
  directories:
    - path: /etc/containers/systemd
      mode: '0755'
      user: root
      group: root
      ensure_parent: 'true’
  firewall:
    ports:
      - '1881:tcp'
  services:
    enabled:
      - process-control
```

Note: These modifications can be part of an upgraded image, or as part of a new system deployment

Embedding an Application into an Image Demo

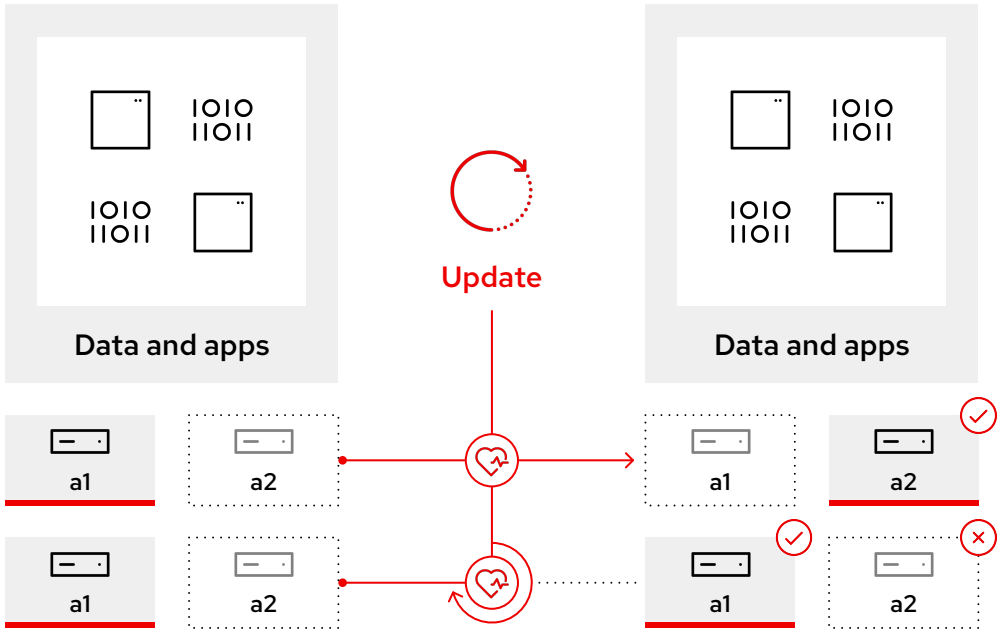
Greenboot



Red Hat

Intelligent rollbacks: Greenboot

Additional safeguard for application and OS compatibility



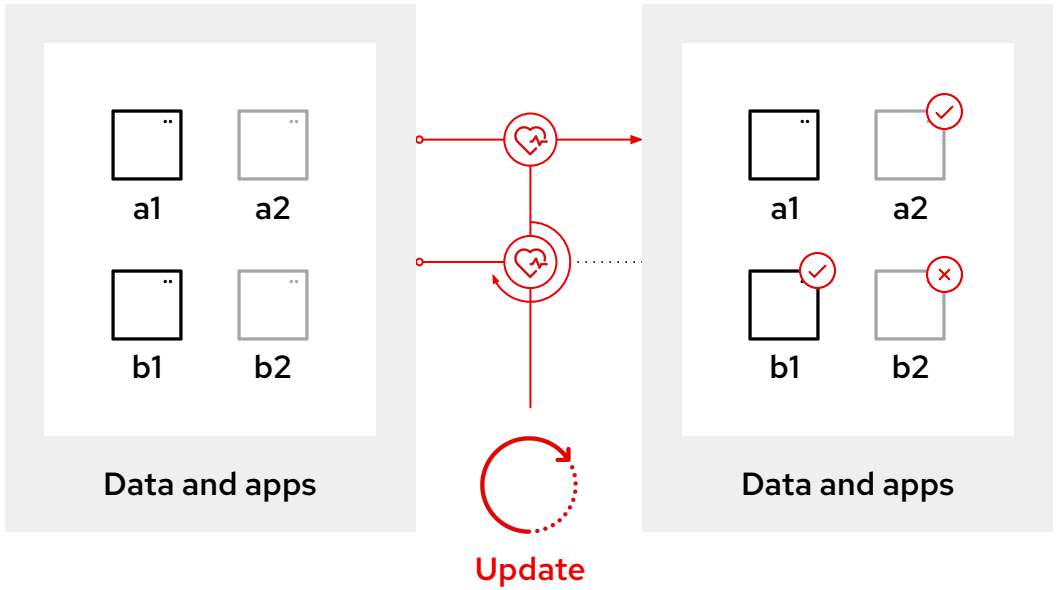
Custom health checks can determine if nodes are functioning properly

- Health checks are run during the boot process.
- If checks fail, a counter will track the number of attempts.
- In a failure state, the node will use rpm-ostree to rollback the update.
- Examples can include:
  - Basic name resolution
  - Service or container status or health

Red Hat

Intelligent rollbacks: Greenboot

Additional safeguard for application and OS compatibility



Podman will automatically roll back containers if new application versions exit on fail

- Requires use of Podman auto-update
- systemd units are used for managing containers
- **--sdnotify=container** adds the ability to wait and notify when the container's process(s) have started properly
- Podman can generate via:

```
# podman create --name test --label
io.containers.autoupdate=registry [registry]/[image:tag]
# podman generate systemd --new test
```

Creating a Greenboot Application Health Check

Using a simple shell script:

```
#!/bin/bash

/usr/bin/sleep 20

RETURN_CODE=$(/usr/bin/curl -s -o /dev/null -w '%{http_code}'
http://localhost:1881)

if [ $RETURN_CODE = '200' ]; then
    exit 0;
else
    exit 1;
fi
```

Now embed in image via file route:

```
builder compose_customizations:
  files:
    - path: /etc/greenboot/check/required.d/application-check.sh
      mode: '0755'
      user: root
      group: root
      data: "#!/bin/bash\n\n/usr/bin/sleep 20\n\nRETURN CODE=$(/usr/bin/curl -s
-o /dev/null -w '%{http code}' http://localhost:1881)\n\nif [ $RETURN_CODE =
'200' ]; then\n    exit 0;\nelse\n    exit 1;\nfi"
```

Tip: Use `sed -E ':a;N;$!ba;s/\r{0,1}\n/\\n/g'` to convert files into a single line with proper newlines embedded

- **/etc/greenboot/check/required.d** contains the health checks that must not fail (if they do, GreenBoot will initiate a rollback)
- **/etc/greenboot/check/wanted.d** contains health scripts that may fail. GreenBoot will log that the script failed, however, it will not rollback.
- **/etc/greenboot/green.d** contains scripts that should be run after GreenBoot has declared the boot successful
- **/etc/greenboot/red.d** contains scripts that should be run after GreenBoot has declared the boot as failed.

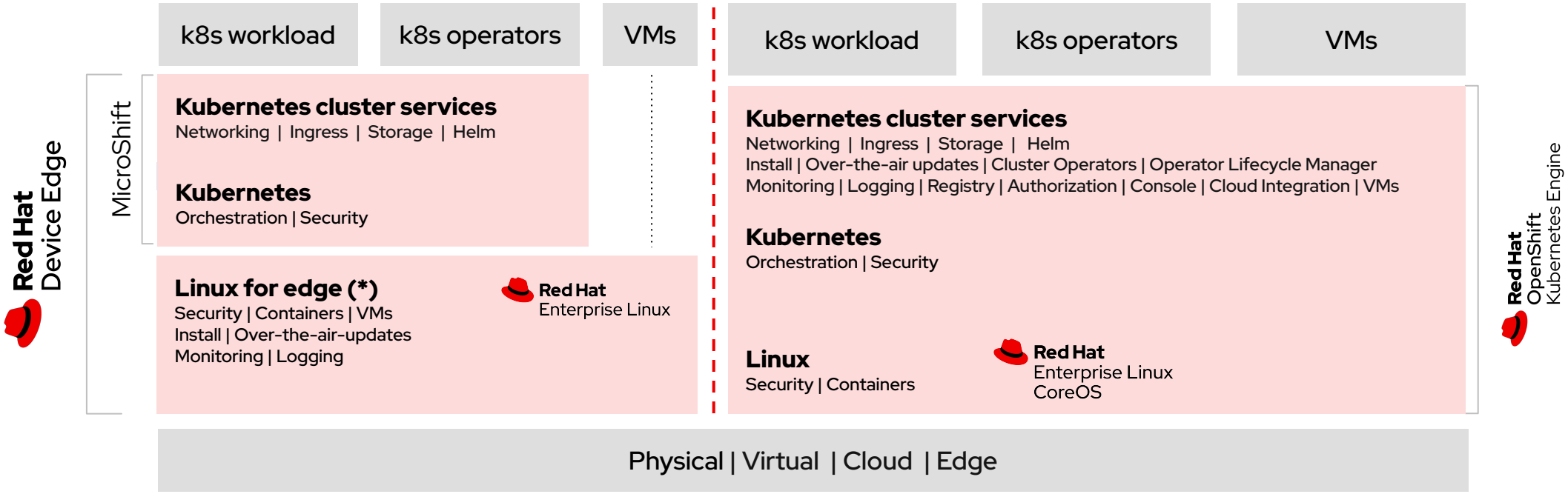
Greenboot Demo

Red Hat

Deploying Microshift to Run an Embedded Application

Red Hat

Device Edge with MicroShift compared to Openshift



* recommended for edge deployments: Red Hat Enterprise Linux for Edge Images, rpm-ostree, immutable, atomic upgrade, over the air flavour of Red Hat Enterprise Linux.

Adding Microshift to a Red Hat Device Edge Image

The infra.osbuild collection automagically manages repositories (sources) for Image Builder

Adding repositories:

```
builder_rhsm_repos:
  - "rhocp-4.13-for-rhel-9-x86_64-rpms"
  - "fast-datapath-for-rhel-9-x86_64-rpms"
```

Adding packages:

```
builder_compose_packages:
  - microshift
  - microshift-greenboot
```

Adding customizations:

```
builder compose_customization:
  firewall:
    ports:
      - '6443:tcp'
      - '1881:tcp'
  services:
    enabled:
      - microshift
      - deploy-pull-secret
```

These additions get the base platform of Microshift built into the image

Note: Microshift can be added or removed through normal rpm-ostree updates

Giving our Embedded Application to Microshift to Run

Microshift will read /etc/microshift/manifests looking for applications to deploy

kustomization:

```
---
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
namespace: process-control
resources:
  - process-control.yaml
```

kustomization.yaml has my application yaml
listed under **resources**.

Namespace and services:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: process-control
---
apiVersion: v1
kind: Service
metadata:
  name: mqtt
spec:
  ports:
  - port: 1883
    protocol: TCP
    targetPort: 1883
  selector:
    app: mqtt
  type: NodePort
---
apiVersion: v1
kind: Service
metadata:
  name: ui
spec:
  ports:
  - port: 1881
    protocol: TCP
    targetPort: 1881
  selector:
    app: ui
  type: NodePort
```

Deployments:

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mqtt-deployment
spec:
  selector:
    matchLabels:
      app: mqtt
  replicas: 1
  template:
    metadata:
      labels:
        app: mqtt
    spec:
      containers:
        - name: mqtt
          image:
quay.io/device-edge-workshops/process-control-mqt
t:1.0.0
          ports:
            - containerPort: 1883
              name: mqtt-port
```

Deploying Microshift to Run an Embedded Application Demo

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

facebook.com/redhatinc

youtube.com/user/RedHatVideos

twitter.com/RedHat

Red Hat