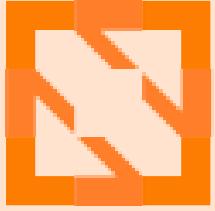




KubeCon



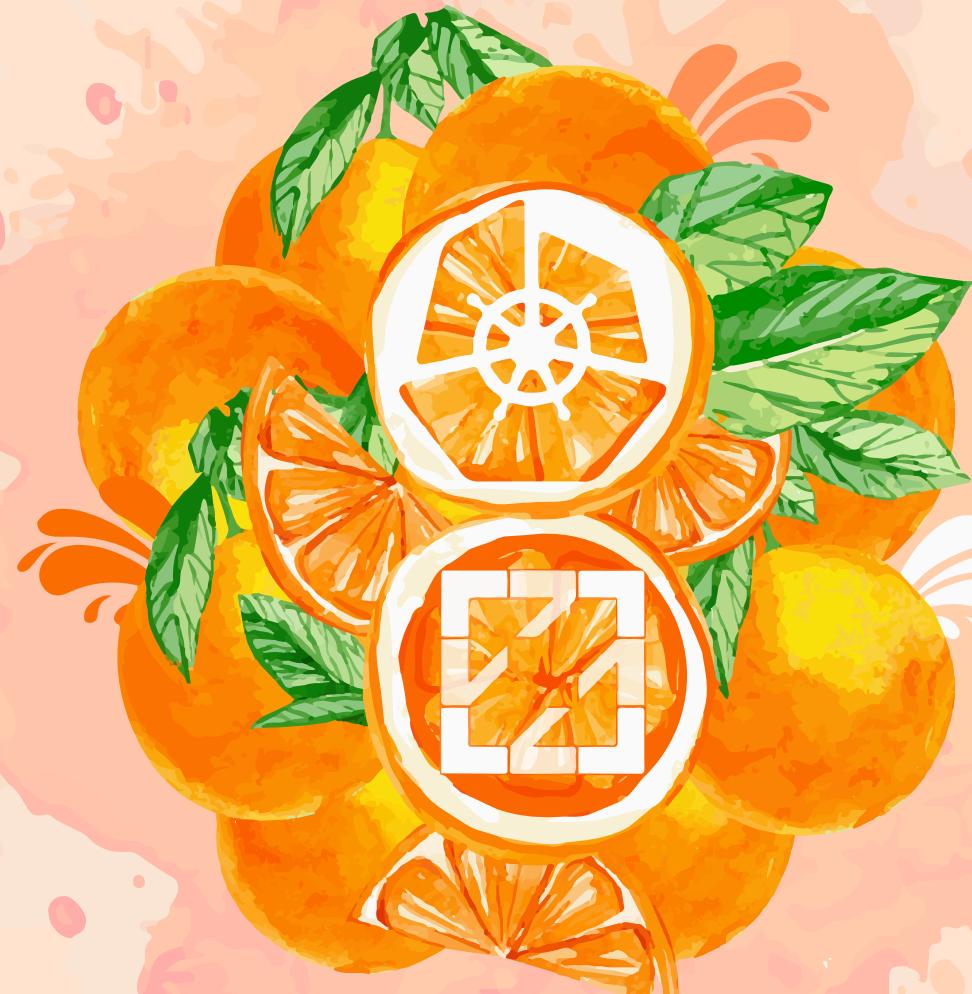
CloudNativeCon

---

Europe 2022

---

WELCOME TO VALENCIA





KubeCon



CloudNativeCon

Europe 2022

# Kubernetes IoT Edge Working Group

Edge Device Onboarding and  
Management

Kate Goldenring  
Microsoft

Steven Wong  
VMware

Thursday, May 19 • 11:00 - 11:35 room 4f <https://sched.co/ytoP>



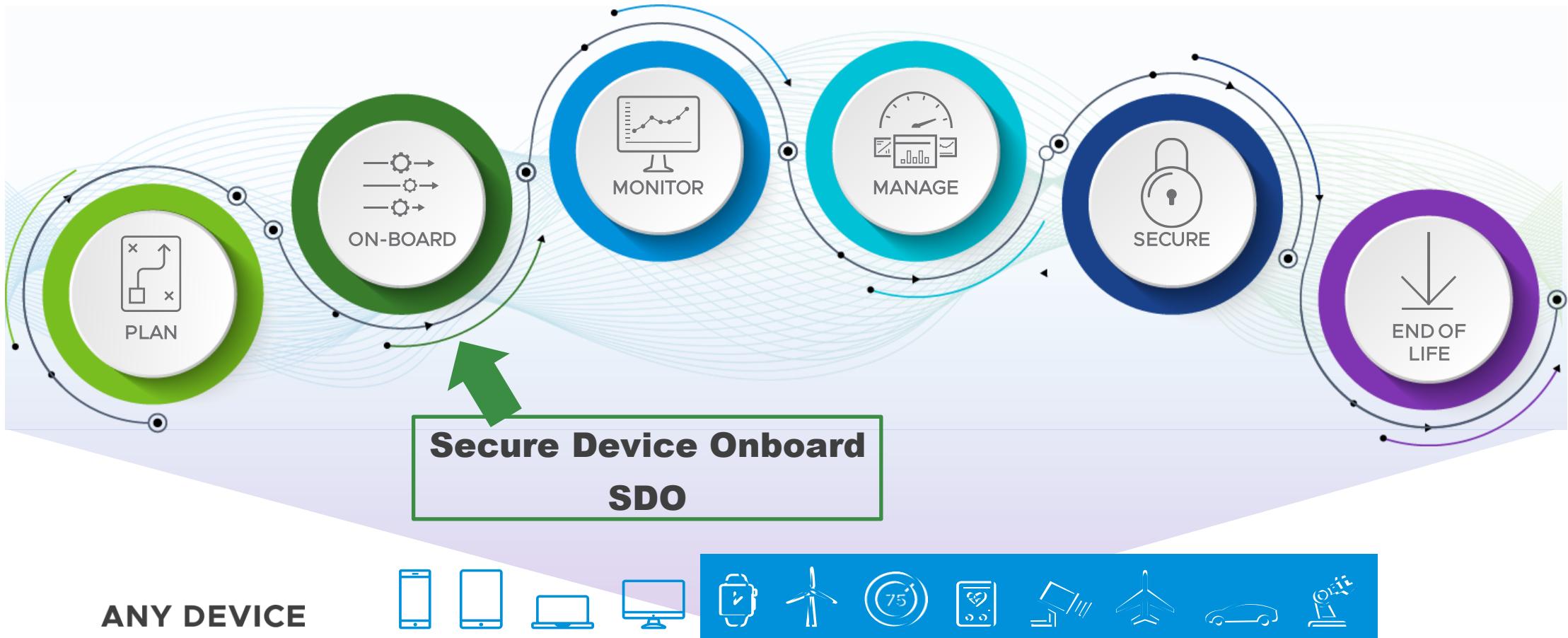
# Agenda

Overview of a few OSS projects that enable device onboarding and management across a range of edge devices

- Compute device onboarding with SDO
- Constrained device use and management with Akri
- The IoT Edge Working Group – Get involved!

# Device operation is a multi-dimensional challenge

Device lifecycle – a time dimension





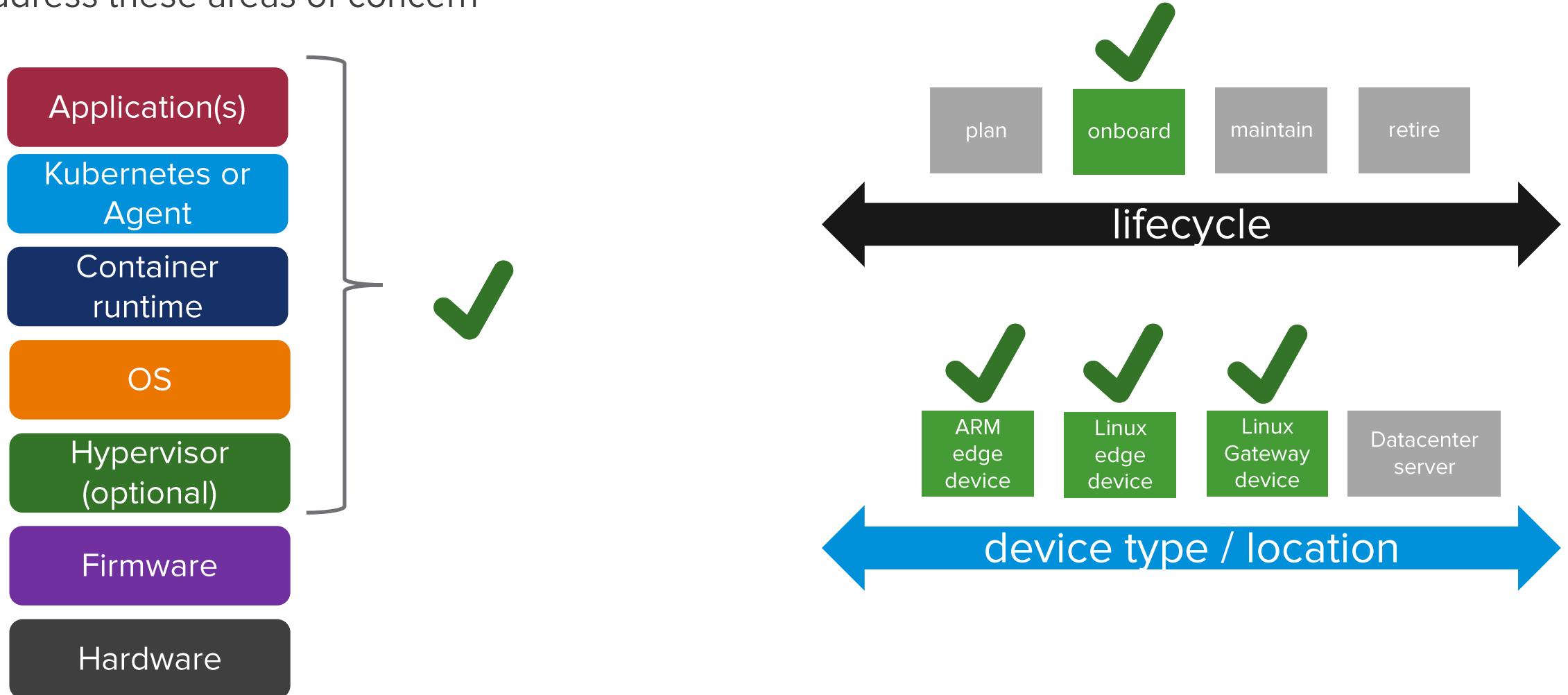
## Helicopter visit to offshore rig

Maybe not an efficient way to perform  
“over the air updates”  
at scale

Photo: [Wikipedia / Tobias Klenze / CC-BY-SA 4.0](#).

# Secure Device Onboard

Address these areas of concern



# FDO Secure Onboard

An automatic onboarding protocol for IoT devices



<https://github.com/secure-device-onboard>



The [FIDO Device Onboard Specification](#) is an open spec being shepherded by the FIDO Alliance



The LF Edge umbrella organization is fostering an [open source implementation](#) of the FIDO Device Onboard Specification

Origin is the Secure Device Onboard Project published in open source by Intel in 2020

## FDO – Pre-req

Manufacturer hosts two API services

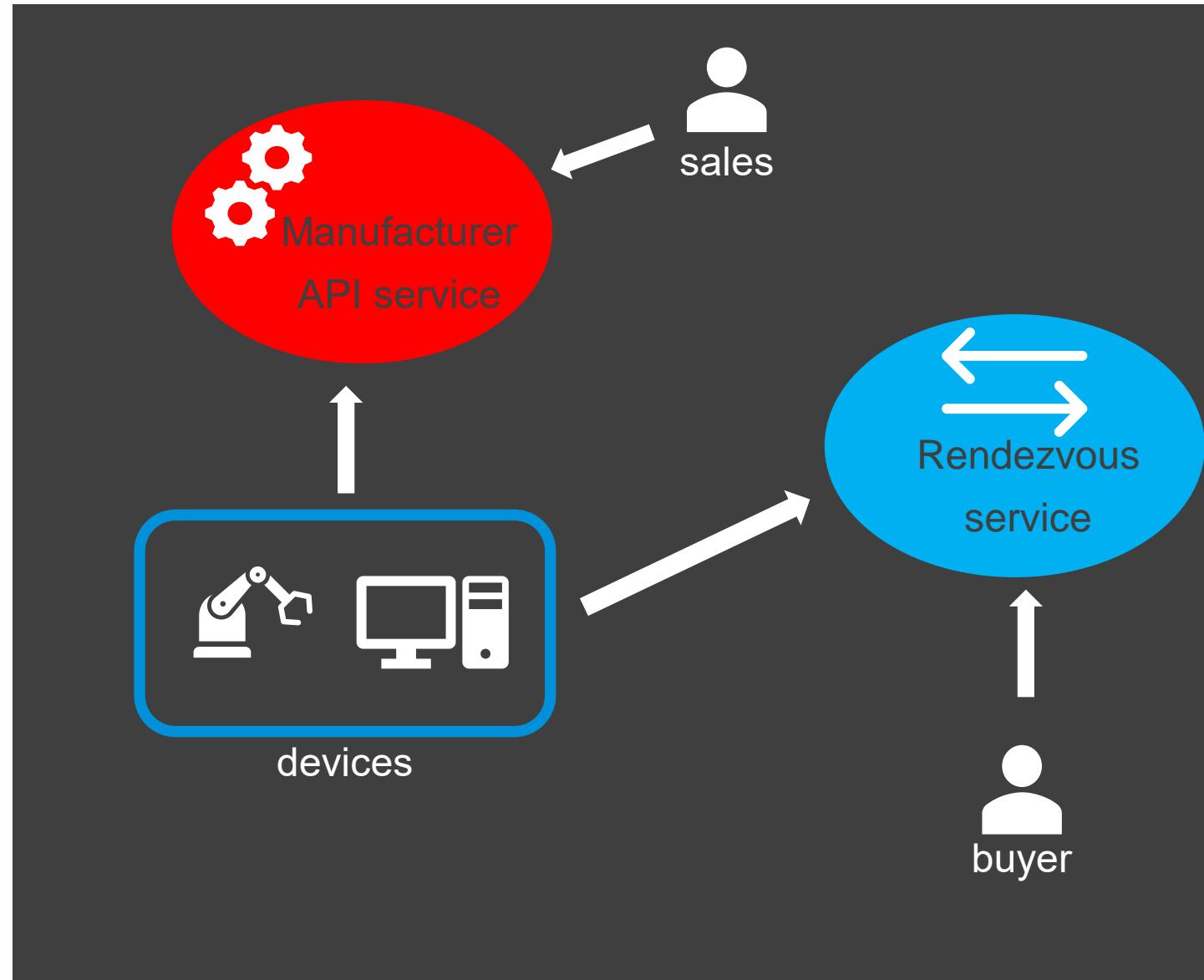
Device manufacturer makes arrangements to operate a Manufacturer Server and a Rendezvous Server.

Manufacturer server will be used interact with:

1. Device upon creation
2. Sales operation business logic upon first sale

Rendezvous Server will be used after sale by:

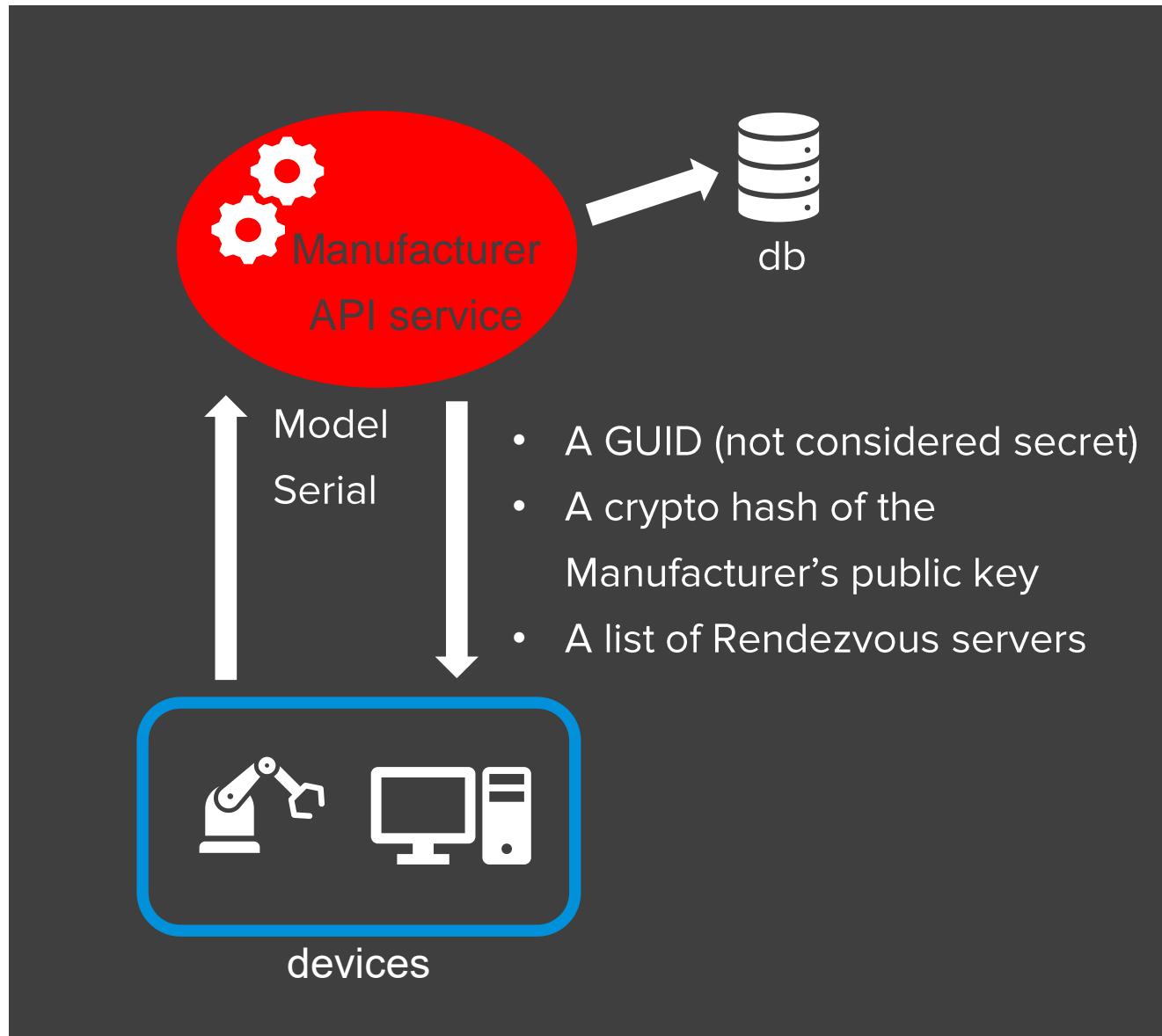
1. Device Owner after sale
2. Device on first power up



# Step 1 - DI

## Device Initialization

1. Device is created. Should have a TPM or PUF for best security
2. An Executable (built from [client SDK](#)) is installed in device and run.  
It makes a call to the Manufacturer API, reporting device model and serial number.
3. Manufacturer ID Hash & Rendezvous server address(es) are returned and stored securely
4. The Manufacturer server maintains a record that this occurred, indexed by the device serial number







## Step 2 device is sold

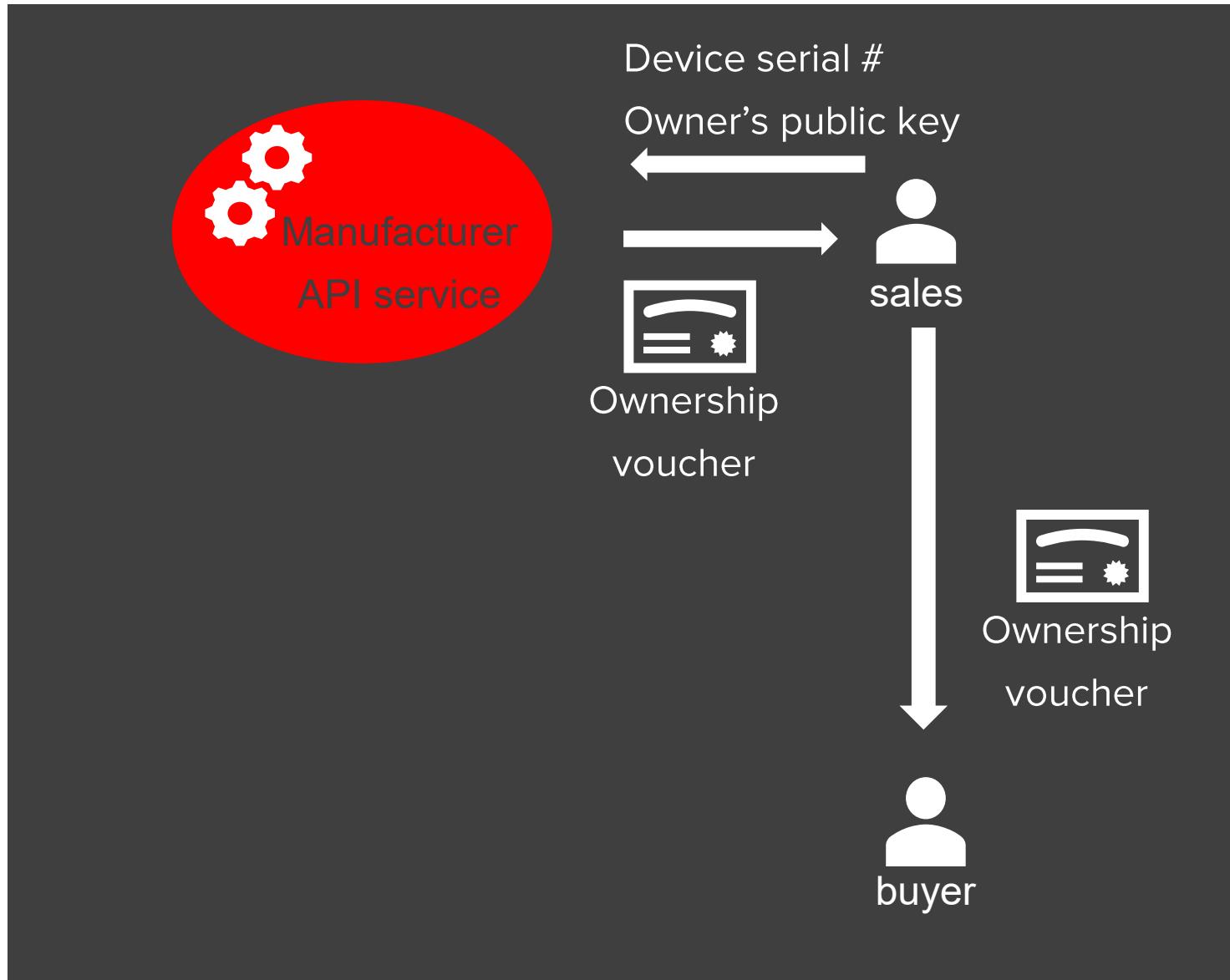
An ownership voucher is generated

Manufacturer's business logic requests a voucher generation

The voucher goes to the new owner (could be a distributor/reseller).

The voucher has:

- A voucher ID
- The GUID issued to the device
- Chain
  - Manufacturer's public key
  - Owner(s) public key
- Hash of chain of all former owners including Manufacturer



## Step 3 –TOO, transfer of ownership phase zero

Owner asks Rendezvous server for service for a particular device for a limited time

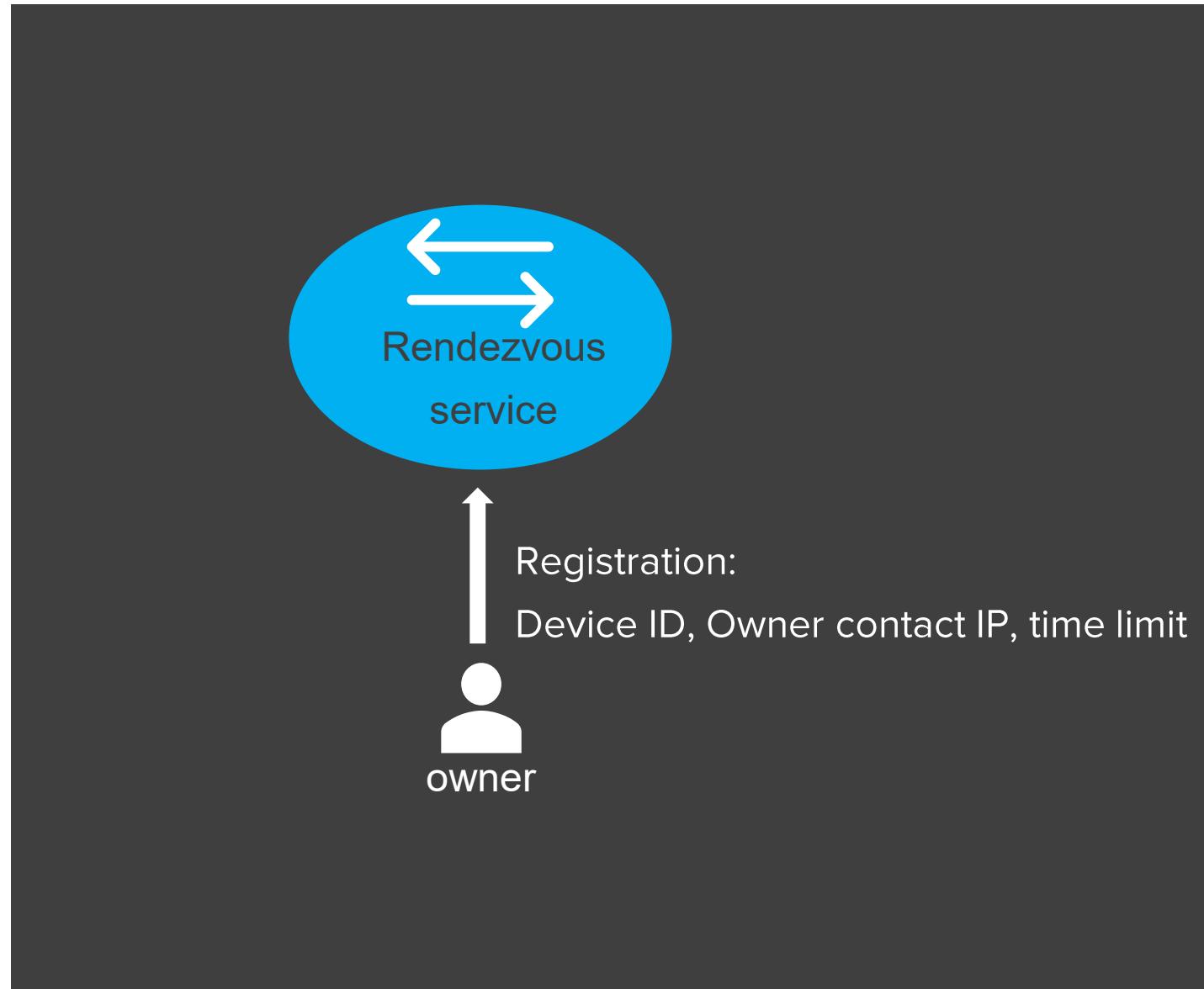
Transfer Ownership 0 (TOO) - Owner

Establishes the mapping of device to the  
Owner's IP address

Pre-req

Owner Service is running:

- Listening on an IP reachable by the device
- configured with the private key counterpart to the public key in the device ownership voucher



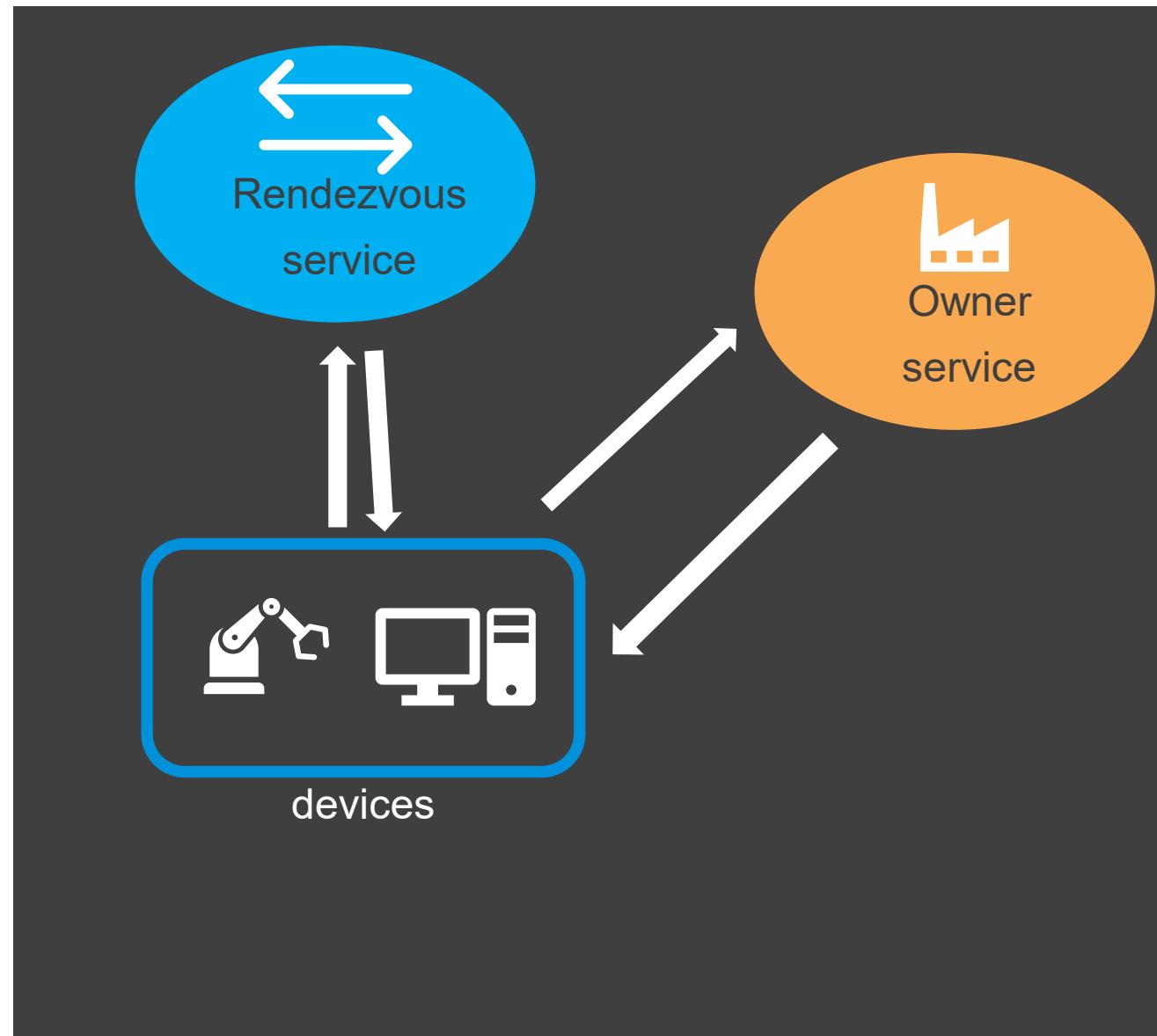
## Step 4 – (TO1 and TO2)

Device is powered on and runs FDO client executable

Device learns how to contact owner

Device contacts Owner

- Mutual trust is established
- Ownership transfer is completed,
- Configuration key pairs are delivered and acted upon
  - Linux “reference example” runs a bash script



## Recorded demo – STEP 1 device initialization

This extracts and installs a prebuilt version of an FDO client SDK from a container, then uses it to engage in API operations with the Manufacturer server



Init a new device  
after production as  
Manufacturer

## Step 2 get a voucher

Specify Manufacturer Server address, credentials and device serial number

```
sdo@sdobuild: ~
sdo@sdobuild:~$ docker run --rm -it -v /tmp/ovoutput:/src/client-sdk-fidoiot/data portainer/fido-client utils/02-generate-voucher.sh http://192.168.1.8:8039 apiuser MCKP2UGEYg5Cfcz3599b3df serial9999
Generating Ownership Voucher for 3599b3df
Ownership Voucher successfully generated under data/3599b3df.bin
sdo@sdobuild:~$ █
```

voucher and choosing profile here

# Configure Portainer to onboard device and supply it with a profile

You could write code to do this, or maybe use Postman to make REST calls

The screenshot shows the Portainer UI with a dark blue sidebar on the left and a white main content area. The sidebar has a navigation menu with sections like Home, EDGE COMPUTE, and SETTINGS. Under SETTINGS, the 'Edge Compute' section is currently selected. The main content area displays a configuration page for the FDO Management Service. It includes a toggle switch labeled 'Enable FDO Management Service' which is turned on. Below the switch, there is a note: 'When enabled, this will allow Portainer to interact with FDO Services.' There are three input fields: 'Owner Service Server' with the value 'http://192.168.1.8:8042', 'Owner Service Username' with the value 'apiUser', and 'Owner Service Password' with a redacted value. At the bottom of the page are two 'Save Settings' buttons, one in each corner.

Home

EDGE COMPUTE

Edge Devices

Edge Groups

Edge Stacks

Edge Jobs

SETTINGS

Users

Environments

Registries

Authentication logs

Settings

Authentication

Edge Compute

Help / About

! When enabled, this will allow Portainer to interact with an OpenAMT MPS API.

Save Settings

! When enabled, this will allow Portainer to interact with FDO Services.

Enable FDO Management Service

Owner Service Server

Owner Service Username

Owner Service Password

Save Settings

# Create a device profile

This one updates Linux and Installs Kubernetes  
profile is in deck

The screenshot shows a user interface for managing device profiles. On the left is a dark sidebar with a 'Settings' dropdown and icons for Authentication, Edge Compute, and Help / About. The main area has a header 'Device Profiles' with a note: 'Add, Edit and Manage the list of device profiles available during FDO device setup'. Below are buttons for 'Add Profile', 'Duplicate', and 'Remove'. A table lists existing profiles:

Name	Created
Docker Standalone + Edge	2022-04-25 10:46:41
Ubuntu-upgrade+Tanzu-Kubernetes	2022-05-09 06:09:25

# Profile content

```
#!/bin/bash

env > env.log

export AGENT_IMAGE=portainer/agent:2.11.0
export GUID=$(cat /src/client-sdk-fidoiot/DEVICE_GUID.txt)
export DEVICE_NAME=$(cat /src/client-sdk-fidoiot/DEVICE_name.txt)
export EDGE_ID=$(cat /src/client-sdk-fidoiot/DEVICE_edgeid.txt)
export EDGE_KEY=$(cat /src/client-sdk-fidoiot/DEVICE_edgekey.txt)
export AGENTVOLUME=/src/client-sdk-fidoiot/data/portainer_agent_data/

# update OS since it may have stayed in the distribution channel for a long time
sudo apt update -y
sudo apt full-upgrade -y
sudo apt autoremove -y
sudo apt clean -y
sudo apt autoclean -y

# we are assuming curl and Docker are already installed

#Install kubectl
curl -LO https://dl.k8s.io/release/v1.22.8/bin/linux/amd64/kubectl
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

# create a user account for running Kubernetes
adduser sdouser --disabled-password --home "/home/sdouser" --gecos "Sdo User"
usermod -aG sudo, docker sdouser

# install tanzu cli
cd /home/sdouser
su -c "curl -LO https://github.com/vmware-tanzu/community-edition/releases/download/v0.12.0/tce-linux-amd64-v0.12.0.tar.gz" sdouser
su -c "tar xzvf tce-linux-amd64-v0.12.0.tar.gz" sdouser
cd tce-linux-amd64-v0.12.0

export ALLOW_INSTALL_AS_ROOT=true
./install.sh

# deploy a k8s cluster
tanzu uc create mycluster

curl https://downloads.portainer.io/portainer-ce211-edge-agent-setup.sh | bash -s -- ${EDGE_ID} ${EDGE_KEY} 1
```

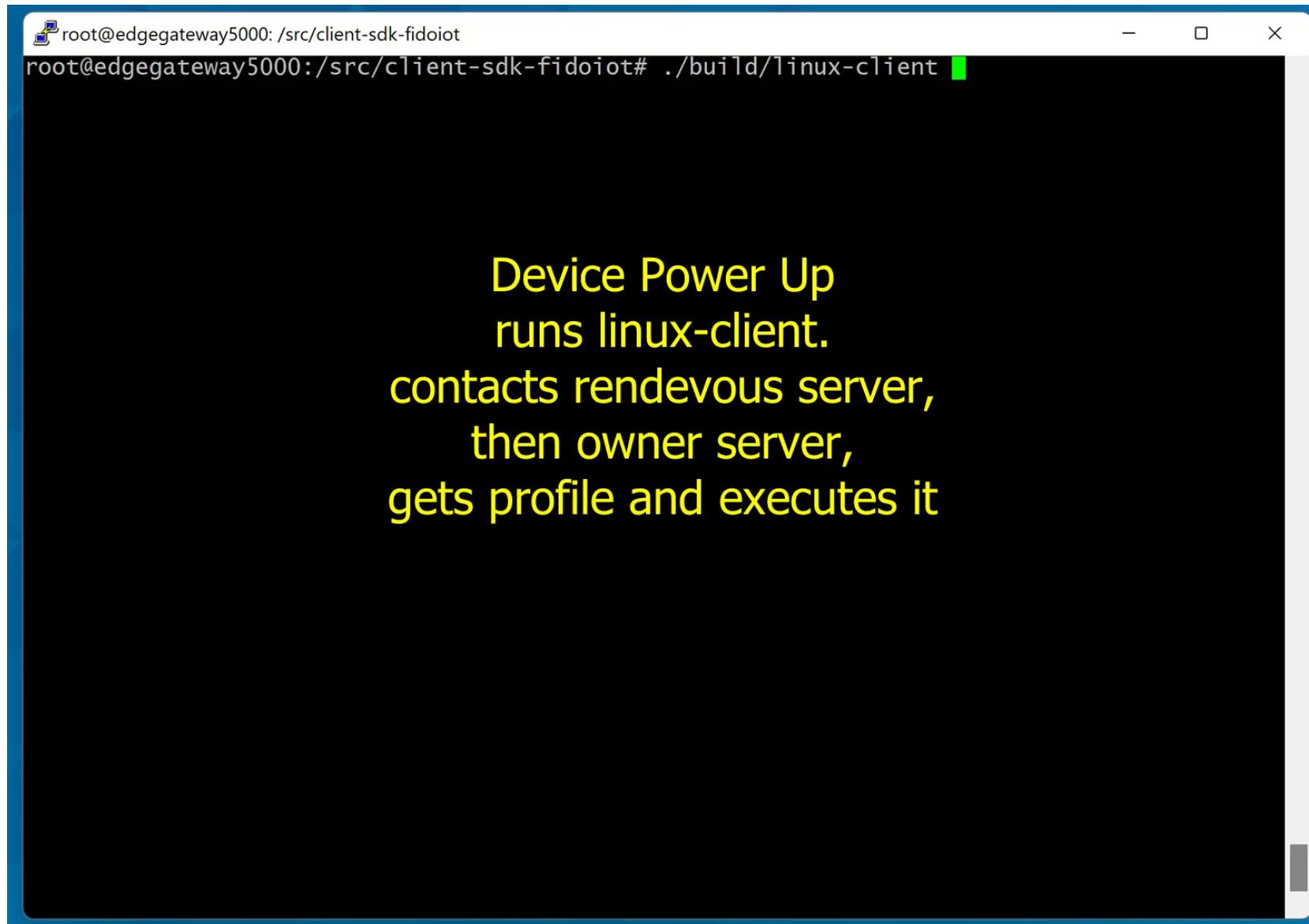
# Get Rendezvous and Owner servers ready for the device

Upload the voucher, name the device, pick the profile we just created from a drop down menu

The screenshot shows the Portainer Edge Compute interface. On the left, there's a sidebar with 'EDGE COMPUTE' and 'SETTINGS' sections. Under 'EDGE COMPUTE', 'Edge Devices' is selected and highlighted in blue. Under 'SETTINGS', 'Environments' is also selected and highlighted in blue. The main content area is titled 'Import Device Set up'. It contains several input fields and instructions:

- Import Voucher:** A note stating "You are setting up a Portainer Edge Agent that will initiate the communications with the Portainer instance and your FDO Devices." Below it is a link labeled "Import Voucher".
- Upload:** A button with an upward arrow icon.
- Device details:** A section with a note "Device name will serve as your reference name in Portainer".
- Device Name:** An input field containing "e.g. FDO-Test01".
- Suffix starting number:** An input field containing "1".
- Portainer server URL:** An input field containing "e.g. https://10.0.0.10:9443".
- Device Profile:** A note saying "Select a profile for your device".

# Turn on device (runs /src/client/client-sdk-fidiot/build/linux-client)



A terminal window with a dark background and light-colored text. The title bar shows the session is run as root on edgegateway5000. The command entered is ./build/linux-client. The terminal window contains yellow text describing the device's actions:

Device Power Up  
runs linux-client.  
contacts rendevous server,  
then owner server,  
gets profile and executes it



DELEDDGEGATEWAY50001

kubectl shell

Dashboard



Custom Templates



Namespaces



Helm



Applications



ConfigMaps &amp; Secrets



Volumes



&gt; Cluster



EDGE COMPUTE



Edge Devices



Edge Groups



Edge Stacks



Edge Jobs



SETTINGS



&gt; Users



&gt; Environments



Registries



&gt; Authentication logs



&gt; Settings



## Application list

Applications

admin

[my account](#)[log out](#)

Applications

Port mappings

Stacks

Table settings

Remove

Add application with form

Create from manifest

Search...



Name

Stack

Namespace

Image

calico-kube-controllers system

-

kube-system

projects.registry.vmware.com/tce/repo-12@sha256:

calico-node system

-

kube-system

projects.registry.vmware.com/tce/repo-12@sha256:

coredns system

-

kube-system

k8s.gcr.io/coredns:coredns:v1.8.4

etcd-mycluster-control-plane system

-

kube-system

k8s.gcr.io/etcd:3.5.0-0

kapp-controller external

-

tkg-system

projects.registry.vmware.com/tce/kapp-controller-r

kube-apiserver-mycluster-control-plane system

-

kube-system

k8s.gcr.io/kube-apiserver:v1.22.7

kube-controller-manager-mycluster-control-plane system

-

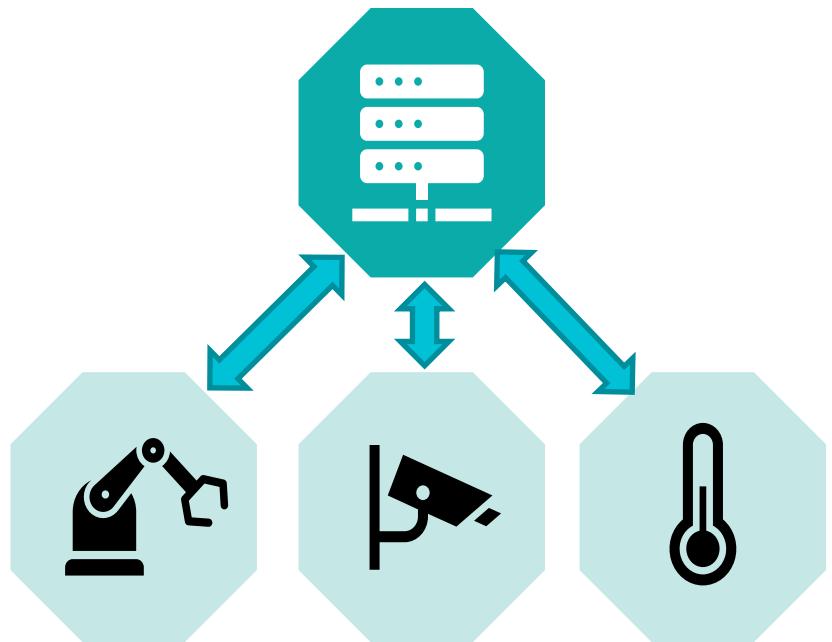
kube-system

k8s.gcr.io/kube-controller-manager:v1.22.7

KubeCon  
Europe 2022

CloudNativeCon

# How can we easily give servers at the edge access to the variety of devices around them?



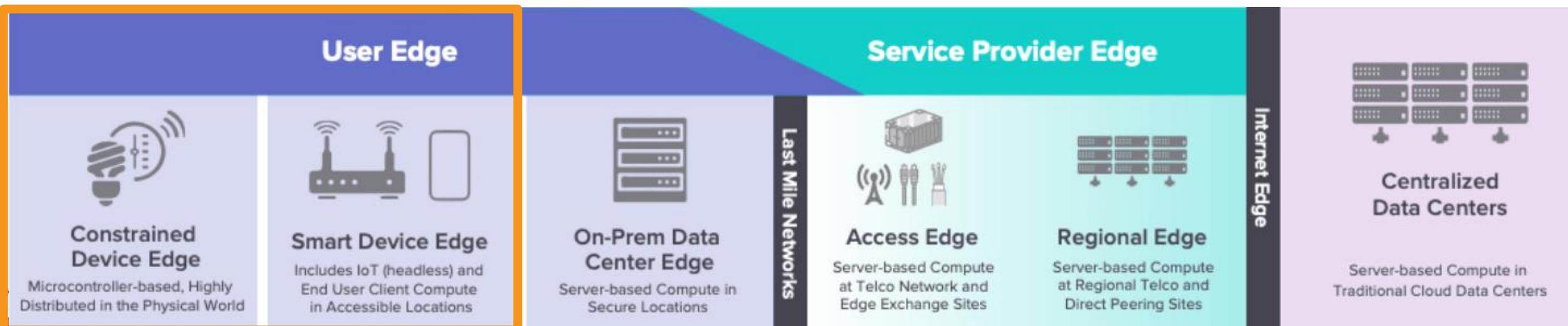
# Tiny Edge

Edge nodes are surrounded by a variety of IoT devices

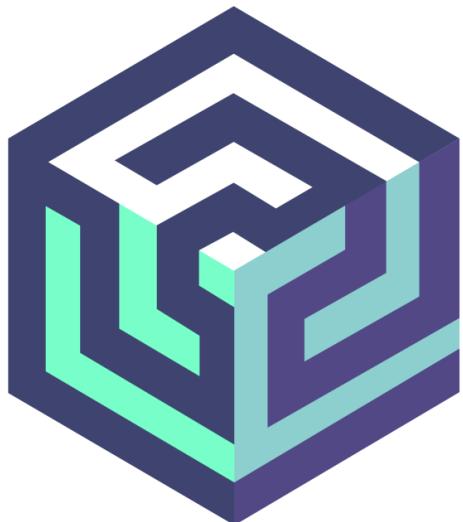
These IoT devices are often too small to run Kubernetes

**How can these devices be dynamically leveraged and managed by Kubernetes workloads?**

Source: LF Edge white paper [\*Sharpening the Edge: Overview of the LF Edge Taxonomy and Framework\*](#)



# Akri: A Kubernetes Resource Interface for the edge



Discovers IoT devices. Handles dynamic appearance and disappearance of devices



Connects IoT devices to a Kubernetes cluster by representing them as a native Kubernetes resources



Schedules workloads based on what devices are connected to the cluster

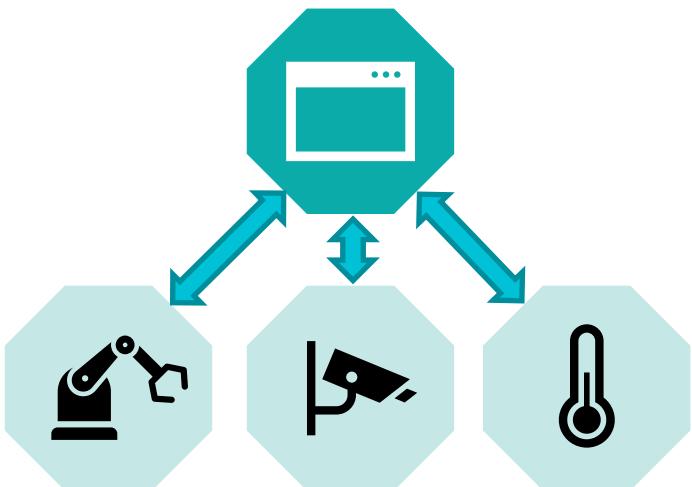


Built with Rust to optimize for the edge and low footprint clusters (K3s, MicroK8s, etc.)



Open-source [CNCF Sandbox Project](#)

# Requesting Akri Kubernetes Resources



```
apiVersion: v1
kind: Pod
metadata:
  name: user-monitoring-app
spec:
  containers:
  - name: user-monitoring-app
    image: user-monitoring-app:latest
    resources:
      requests:
        memory: "10Mi"
        akri.sh/akri-udev-therm-$ID1: "1"
        akri.sh/akri-onvif-video-$ID2: "1"
        akri.sh/akri-opcua-robot-$ID3: "1"
```

# Akri for device use and management

Configure Akri's Controller to automatically deploy Kubernetes Pods or Jobs to discovered devices.

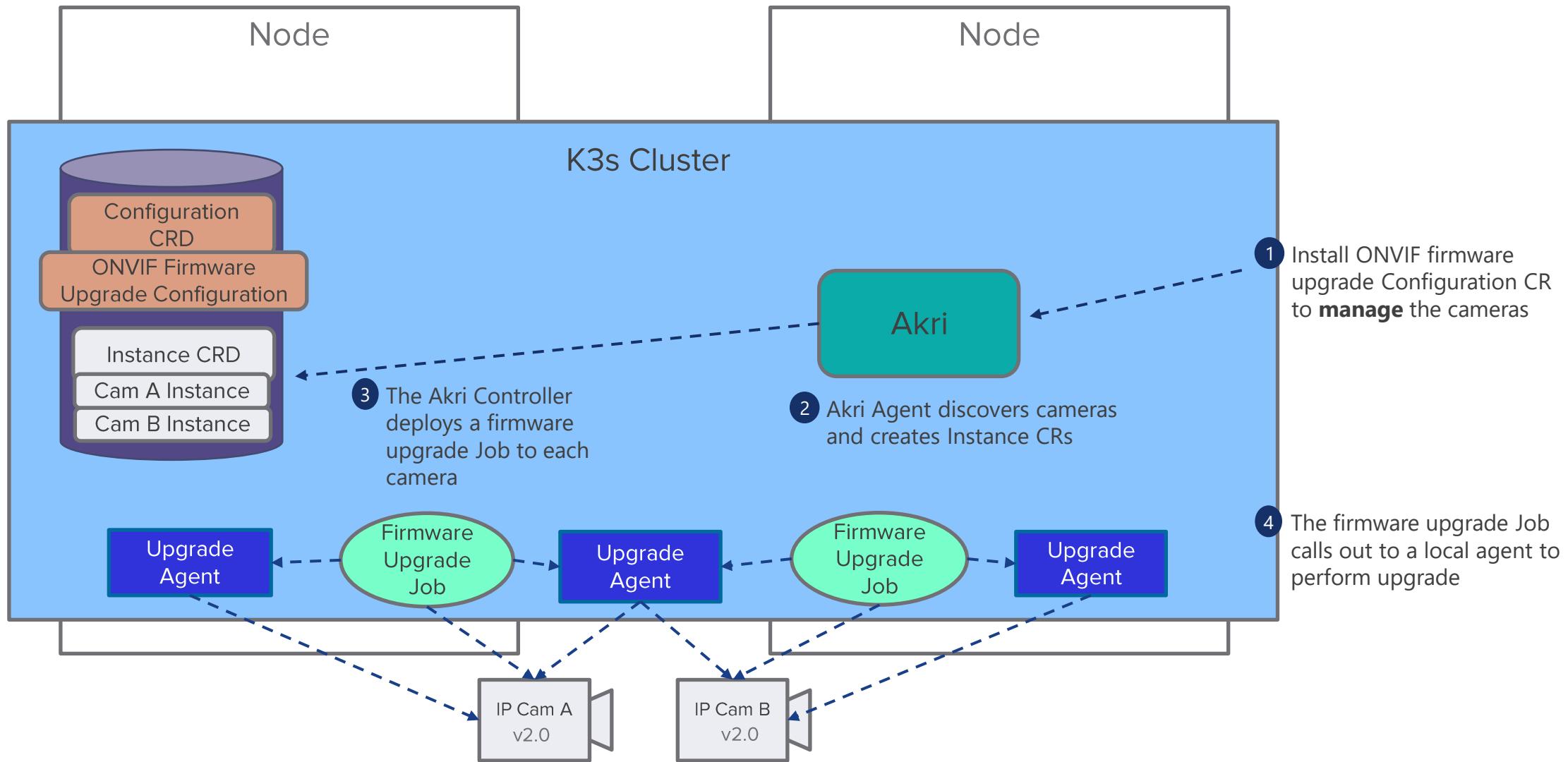


Long-running Kubernetes Pods – Ideal for device use



Single-task Kubernetes Jobs – Ideal for device management (introduced in Akri v0.8.4)

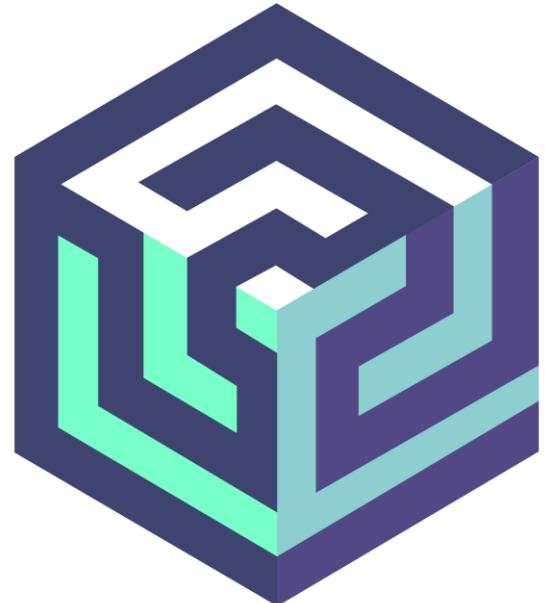
# IP Camera Firmware Upgrade with Akri



# Integrating and Extending Akri

Goal: The open source, standard way to connect devices to a cluster and manage devices from a cluster

- A platform made to be pluggable and extensible
- Plug in your workloads: Enables you to declaratively state what devices your workloads need
- Discover your devices: [Create new Discovery Handlers](#) (i.e. proprietary, Modbus, CoAP, ZeroConf, MQTT)
- Integrate: existing upgrade solutions, device registry, certificate store solution
- Let us know where we can grow!



# Akri Resources



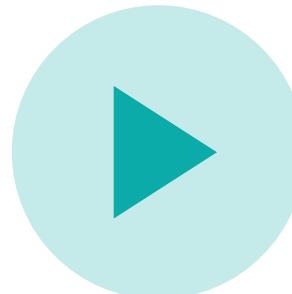
Learn more  
[docs.akri.sh](https://docs.akri.sh)



CNCF Webinar: Discovering & Managing IoT Devices with Akri  
<https://youtu.be/9wCQCV0m5Kk>



Join the community via  
Slack and monthly Zooms!  
[K8s #akri Slack](https://k8s slack.com/archives/akri)



Try it out: End to end demo  
discovering mock USB  
cameras  
<https://docs.akri.sh/demos>

# A Transition: IoT Edge WG moving to CNCF

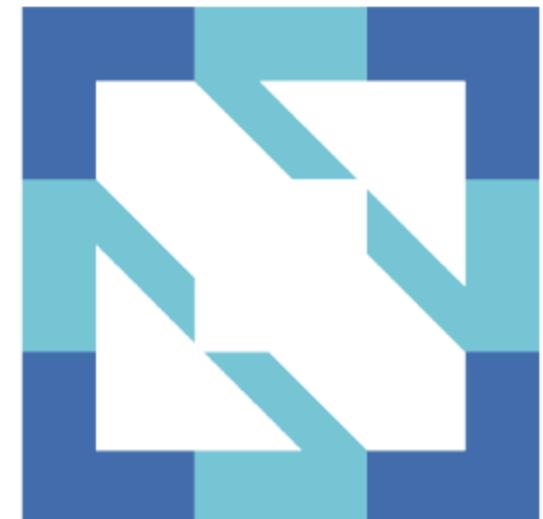
Goal: Build cohesion around heterogeneity of devices and solutions on the edge

Grown beyond Kubernetes specific solutions

Transitioning from Kubernetes to CNCF under Runtime

TAG: <https://github.com/cncf/tag-runtime#working-groups>

Tune into existing Slack to stay updated and involved in new charter and WG resources



# Kubernetes IoT Edge WG Resources



[GitHub \(current, future\)](#)



Previous working group meetings  
[recordings](#)



Join the community via  
Slack and bi-monthly  
Zooms!

[K8s #wg-iot-edge Slack](#)



Google group  
<https://groups.google.com/forum/#!forum/kubernetes-wg-iot-edge>

# Speaker Contact Information



Steven Wong | VMware



@cantbewong



@cantbewong



Kate Goldenring | Microsoft



@KateGoldenring



@kate\_goldenring



This deck is available here:  
<https://via.vmw.com/EiAB>

## Q&A

Thank You

IoT edge WG Slack

<https://kubernetes.slack.com/messages/wg-iot-edge>