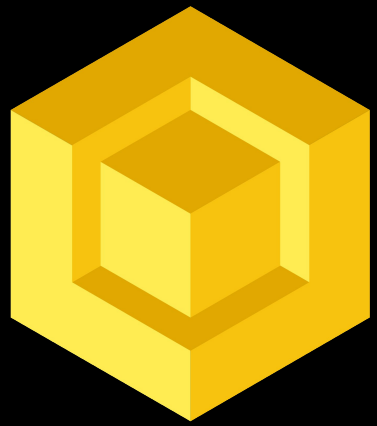
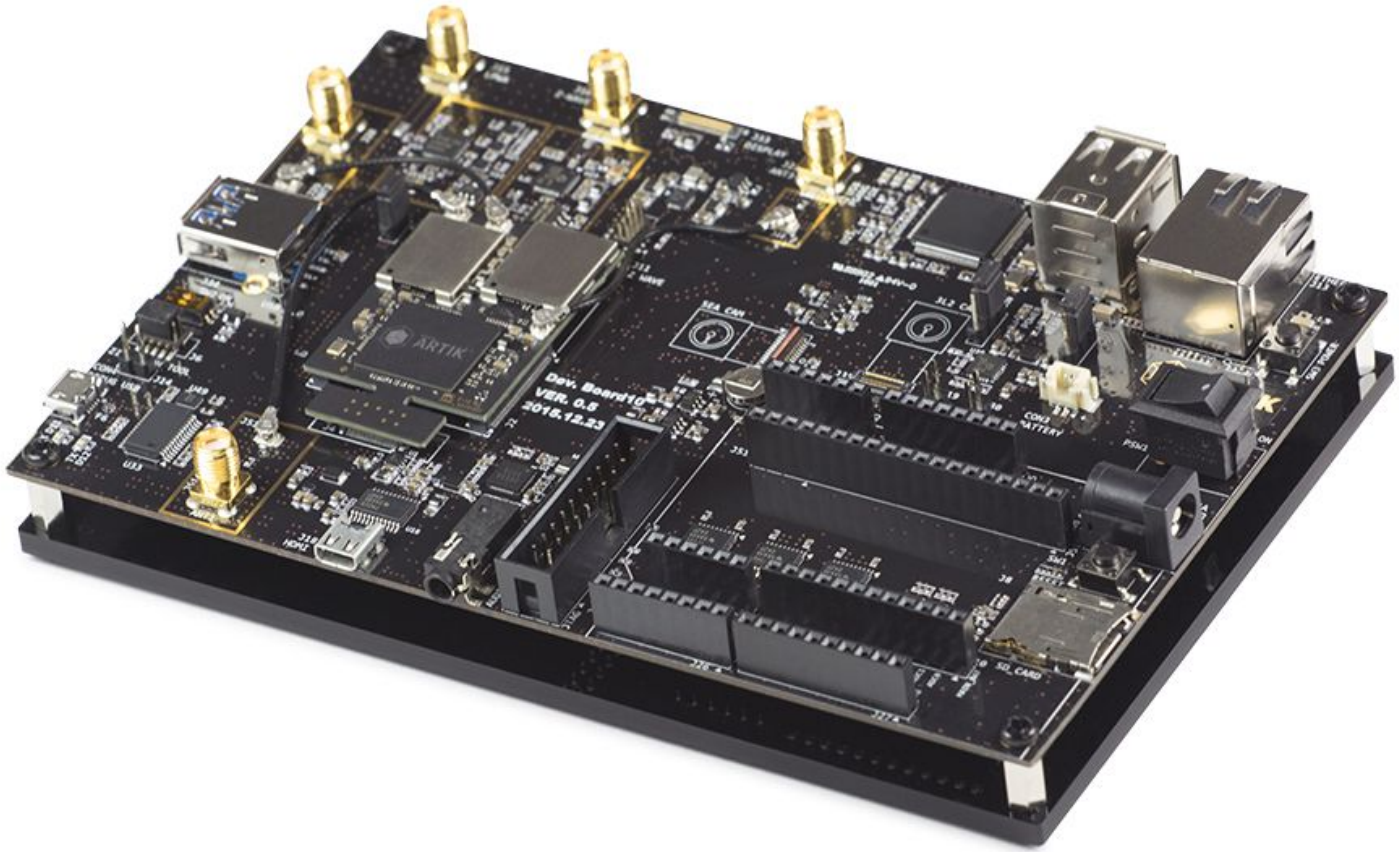
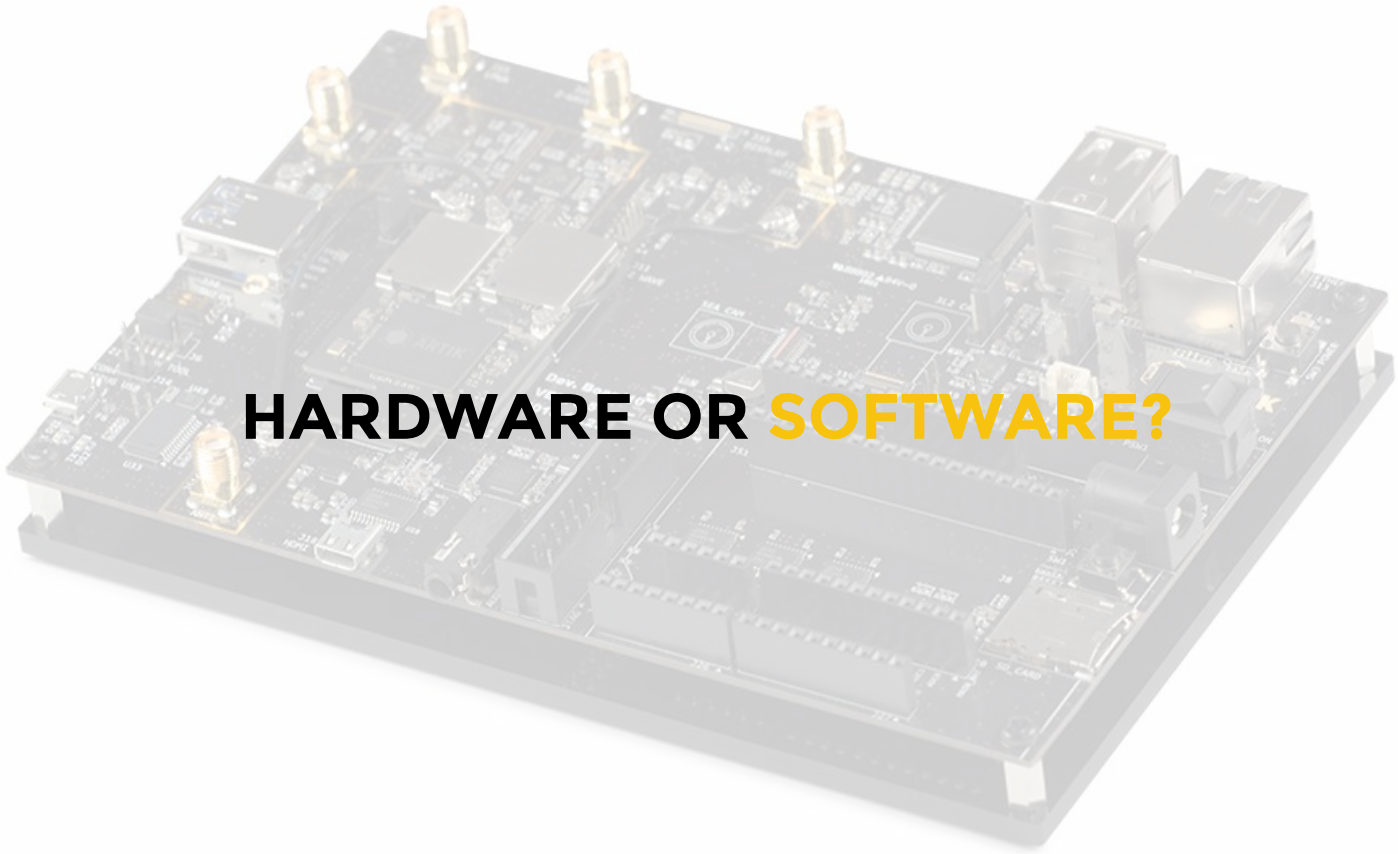


# resin.io

*Samsung Artik Workshop*
*2016/08/09*

Ronald McCollam
Solutions Architect
ronald@resin.io @RonaldMcCollam


ARTIK
Dev. Board1C
VER. 0.5
2015.12.23
SEA CAN
CON3
BATTERY
SD_CARD

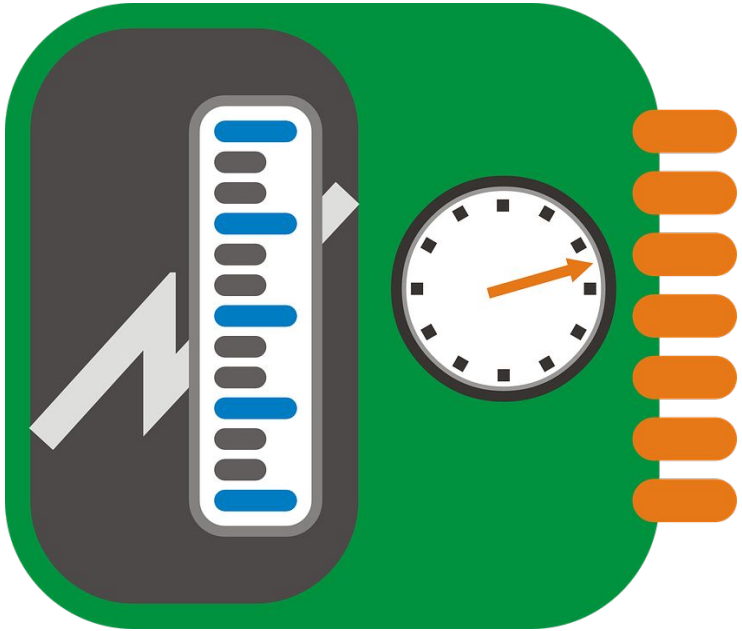


# HARDWARE OR SOFTWARE?

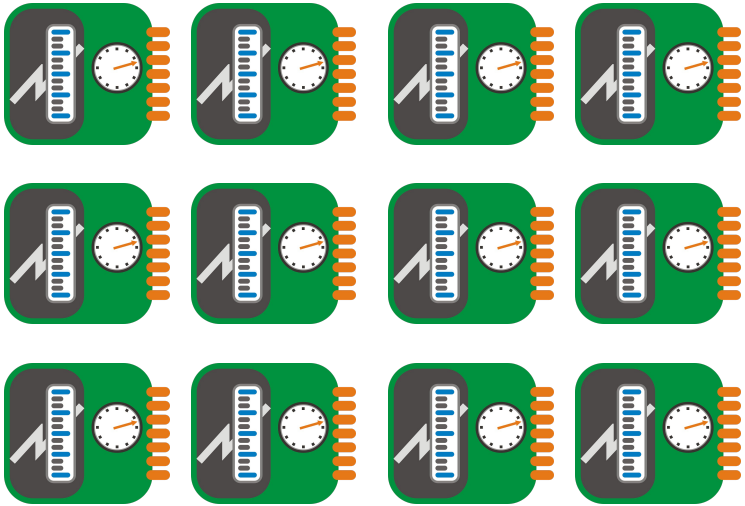
# HARDWARE OR SOFTWARE?

# HARDWARE OR SOFTWARE?

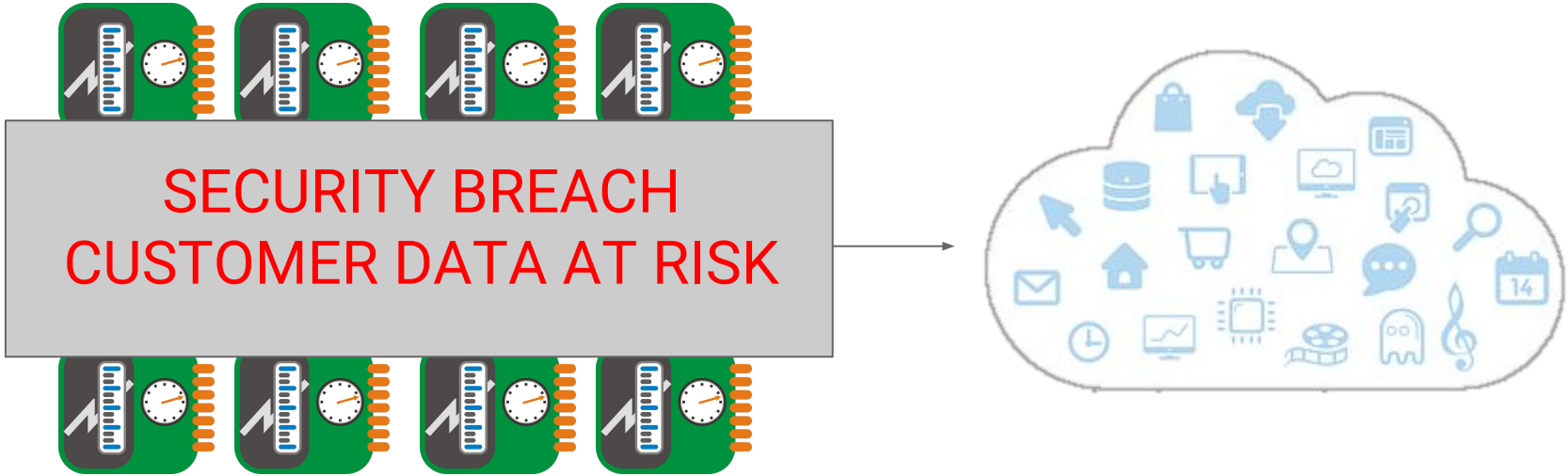
# HARDWARE OR SOFTWARE?



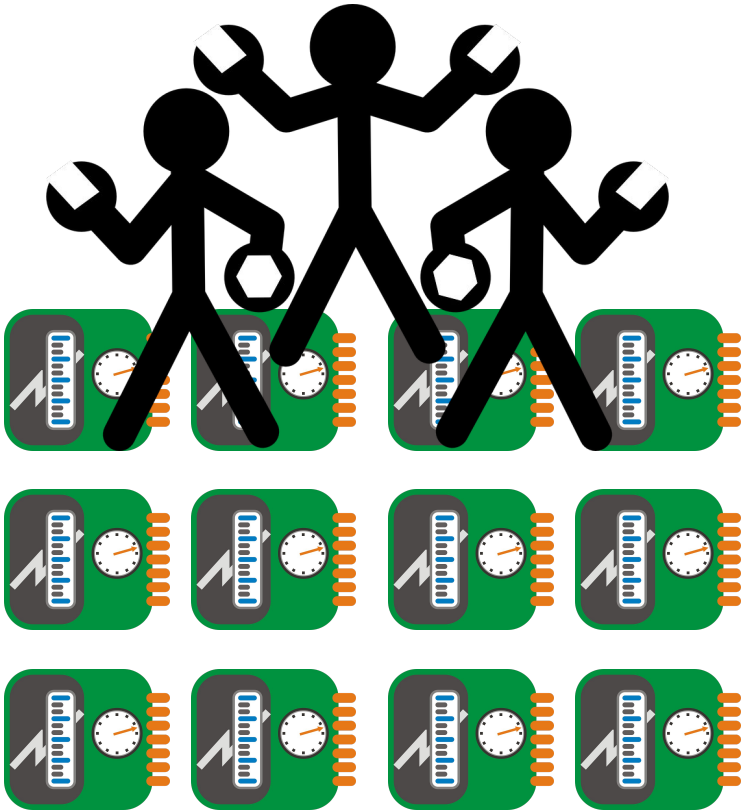
# HARDWARE OR SOFTWARE?



# HARDWARE OR SOFTWARE?



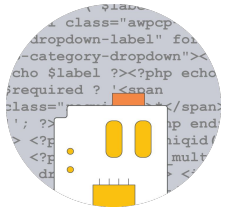
# HARDWARE OR SOFTWARE?




WE LIVED THE
PROBLEM

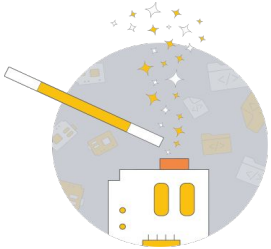
# HOW DOES RESIN.IO HELP USERS?

*“It’s just Git push and forget about it. It’s that easy.” - Sam Levy, Pact Coffee*



**Develop**
with fast feedback

**Deploy & Configure**
just like the cloud



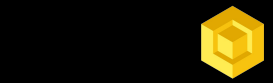
**Provision**
without sweat

**Securely**
out of the box

**At Scale**
for any size fleet

**Uniformly**
across device form factors

# TECHNICAL BACKGROUND

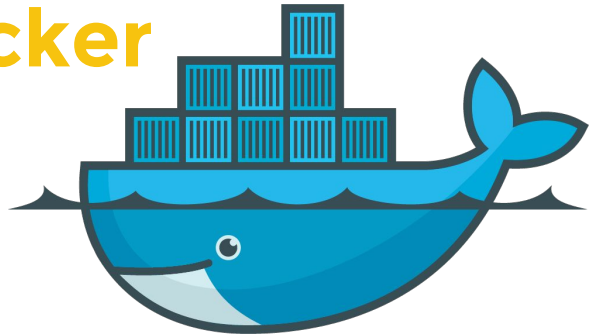


# TECHNICAL BACKGROUND

*(or, everything you never realized you wanted to know about **git** and **Docker**)*

# BACKGROUND:

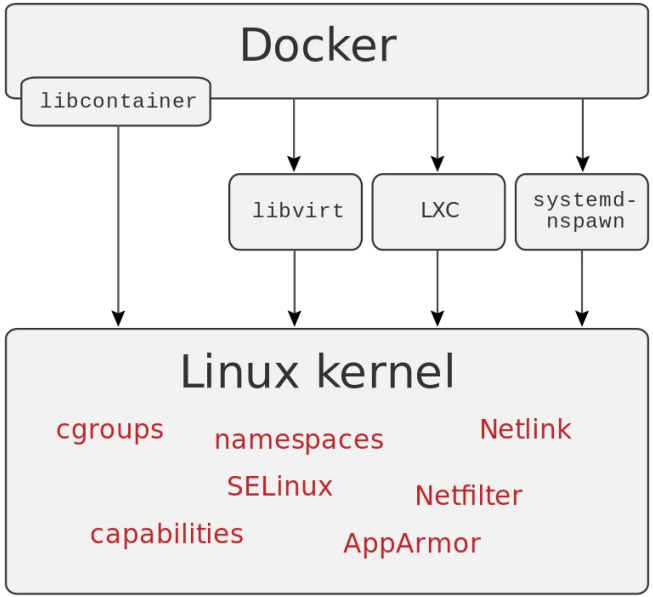
## docker



**docker** is a **software container** solution that eases automation of **deployment** and **management** of operating environments.  It has some similarities with virtualization technologies but is **not virtualization**.

*“Docker containers wrap up a piece of software in a complete filesystem that contains everything it needs to run: code, runtime, system tools, system libraries – anything you can install on a server. This guarantees that it will always run the same, regardless of the environment it is running in.”*

https://www.docker.com/what-docker



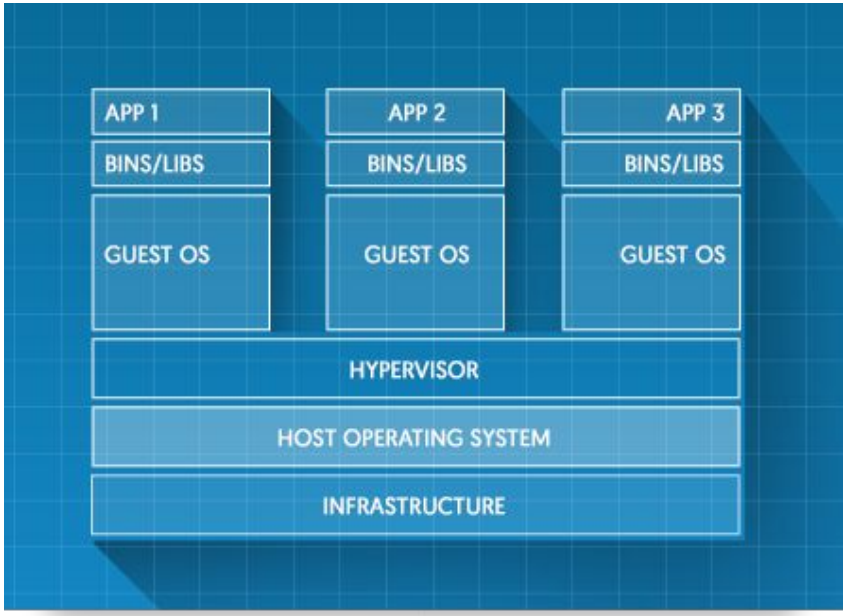
# BACKGROUND: docker

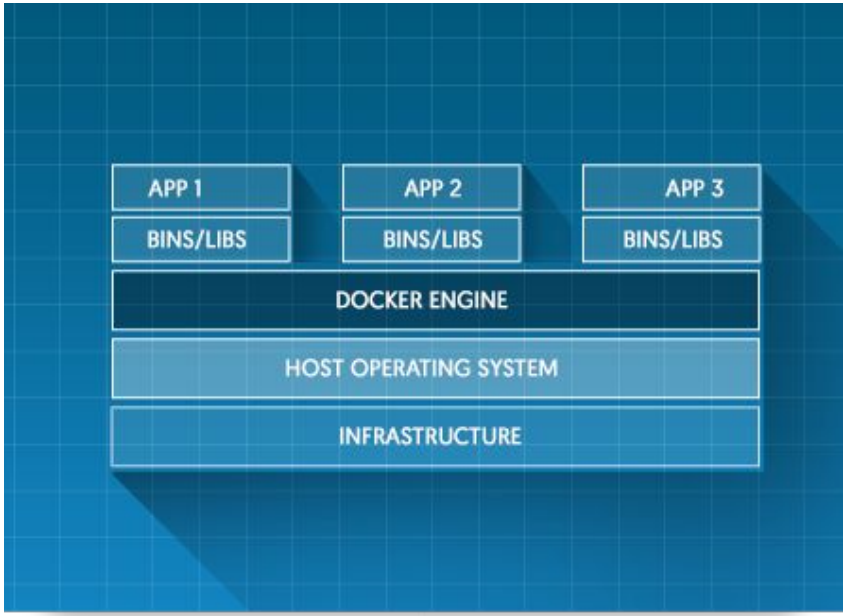
- Commonly used in server environments to simplify replication and deployment of services
- **Containers** are everything that a service (or application) needs to run -- all code, all data, all configuration, packaged up into one unit
- Containers are defined by a **Dockerfile** -- a simple flat text file defining what resources are used
- **Layers** allow caching and fast image reuse

```
FROM resin/rpi-raspbian:jessie-20160511

RUN apt-get update
RUN apt-get -y install fbi

COPY . /usr/src/app
WORKDIR /usr/src/app

CMD ./prestart.sh && ./start.sh
```

# BACKGROUND: docker

Virtual machines vs. Docker

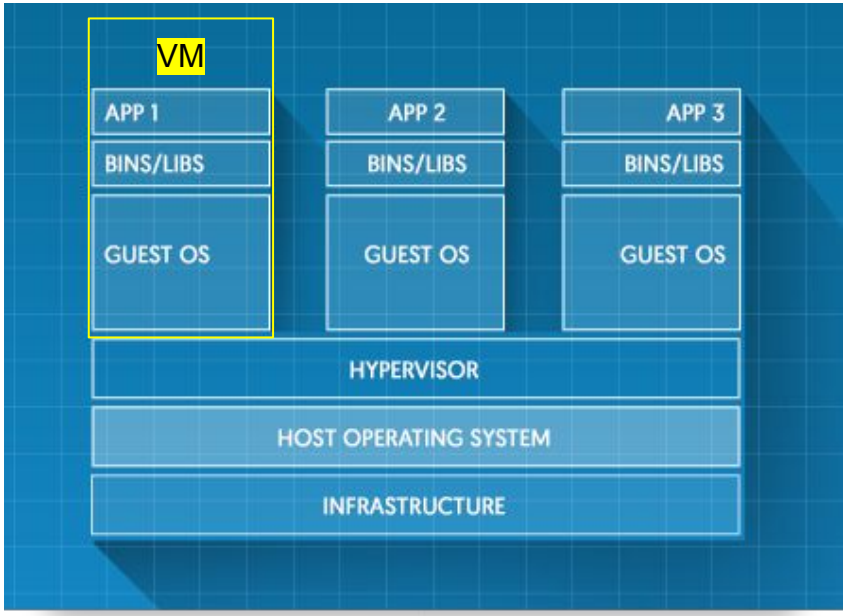


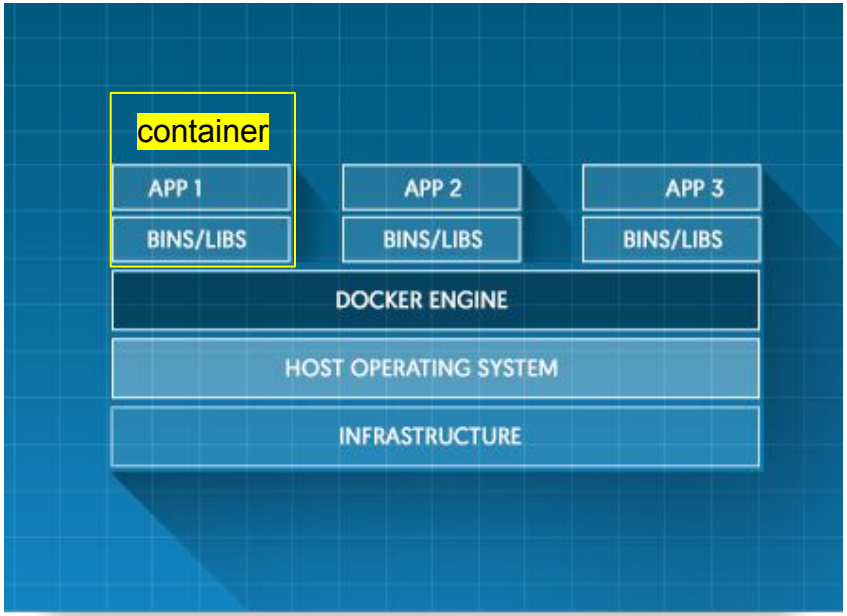




(shamelessly stolen from https://www.docker.com/what-docker)

# BACKGROUND: docker

Virtual machines vs. Docker




Containers are smaller and directly access **native** hardware

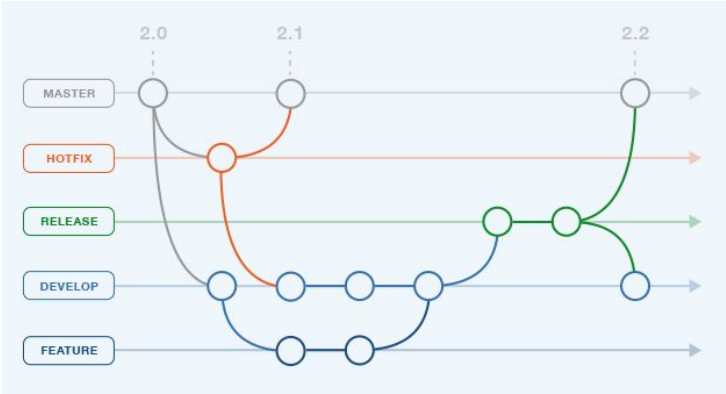
# BACKGROUND:

git

**`git`** is a **distributed revision control system** with a focus on speed and non-linear workflows. Originally designed in 2005 to manage the code for the Linux kernel after existing solutions were found to be inadequate.

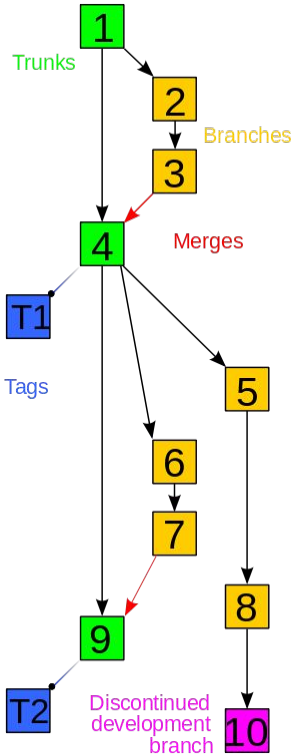
**Typical workflow / important concepts**

| | |
|---|---|
| `git clone` | Copy data from a repository |
| `git commit` | Save your changes |
| `git push` | Send data to a repository |




# BACKGROUND: git

- Provides history of revisions to code
- Distributed -- each branch is equally valid as a source and target (i.e. no privileged or special master server[1])
- Allows "**branches**" -- multiple separate streams of work that can be **merged** to build a software release
- Extremely popular, particularly in the Free/Open Source software world




[1] - In practice, sites like **github.com** are often used as a "master" repository, though this is not any technical restriction imposed by git.

# BACKGROUND: git

- `git clone` - copy data from a source repository

```
$ git clone https://github.com/resin-io-playground/samsung-workshop

Cloning into 'samsung-workshop'...
remote: Counting objects: 29, done.
remote: Compressing objects: 100% (23/23), done.
remote: Total 29 (delta 7), reused 28 (delta 6), pack-reused 0
Unpacking objects: 100% (29/29), done.
Checking connectivity... done.
```

# BACKGROUND: git

- `git commit` - save your changes

```
$ your_favorite_editor myCode.py
$ git add myCode.py      #[1]
$ git commit -m 'Added a new variable called foo'

[master e81821a] Description of change
 1 file changed, 1 insertion(+)
```

[1] - You could also use "`git add .`" or "`git add *`". git is smart enough to pick up only files that actually have changes.

# BACKGROUND: git

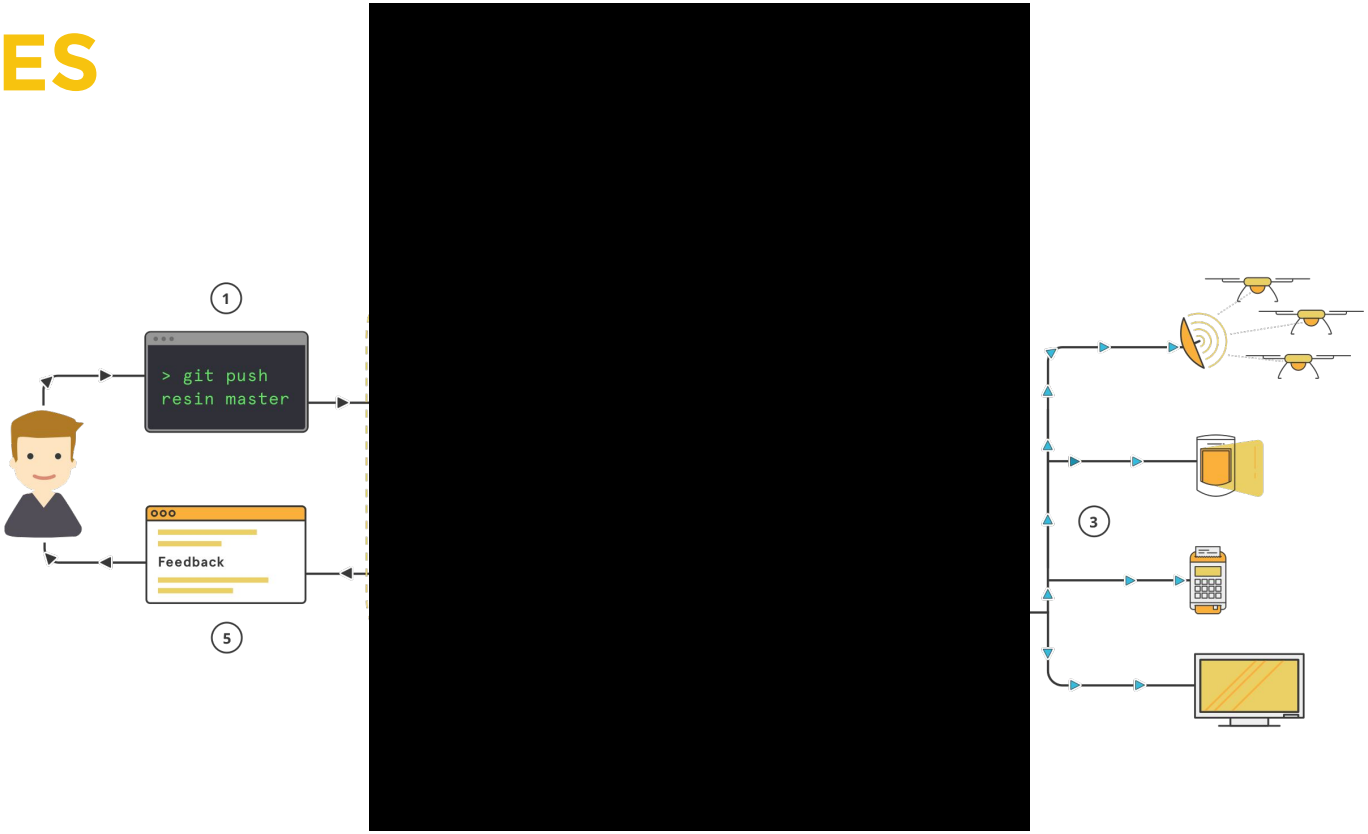
- `git push` - send data to a repository

```
$ git push                    # to default 'upstream' (e.g. to github)
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 1.39 KiB | 0 bytes/s, done.
Total 4 (delta 2), reused 0 (delta 0)

To mccollam@github.com/myProject.git
   d58d881..a8194c0  master -> master
```

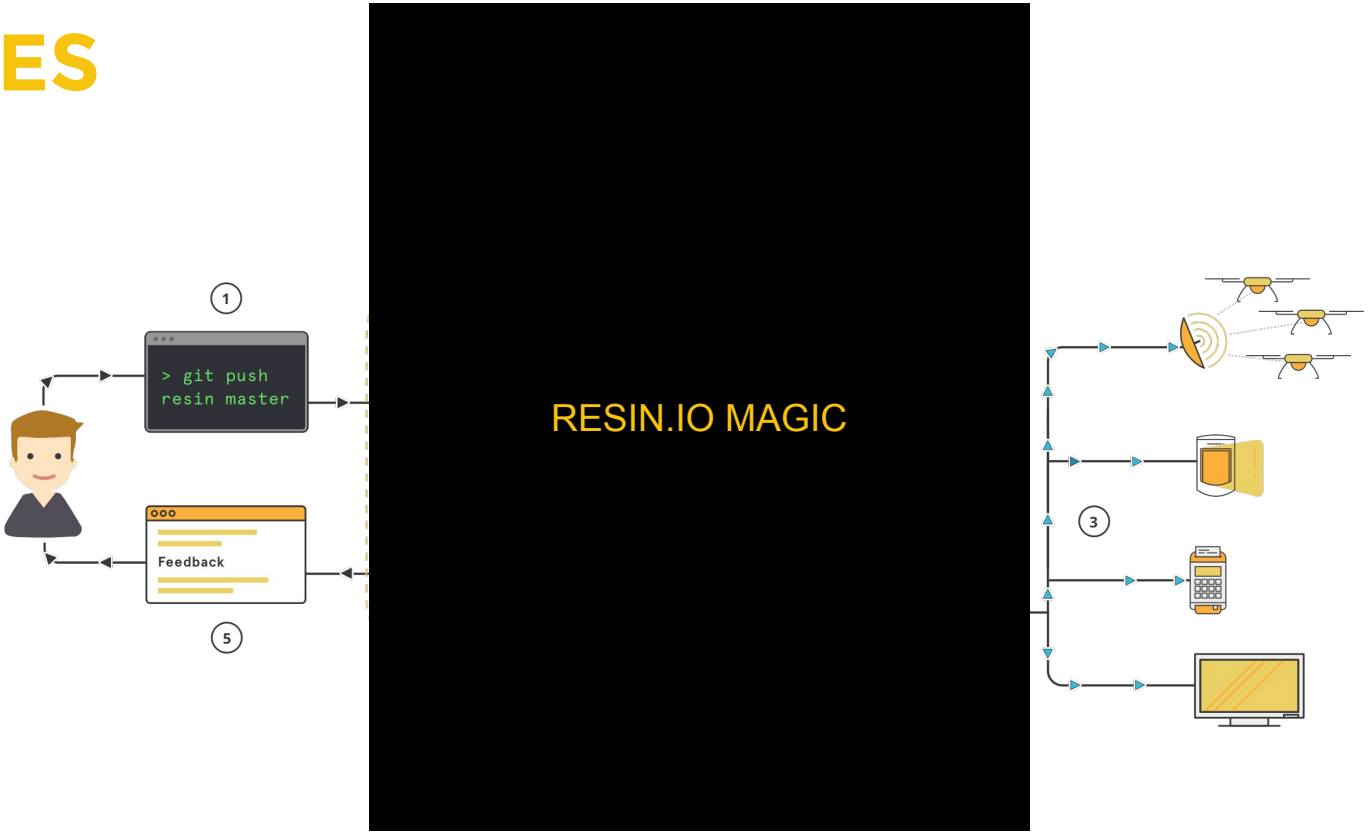
# BACKGROUND: git

```
$ git push resin master          # push master branch to resin 'upstream'
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 1.39 KiB | 0 bytes/s, done.
Total 4 (delta 2), reused 0 (delta 0)

Starting build for mccollam/myProject, user mccollam
Building on 'local'
Pulling old image for caching purposes
[==================================================>]     1 kB/1 kB
...
```

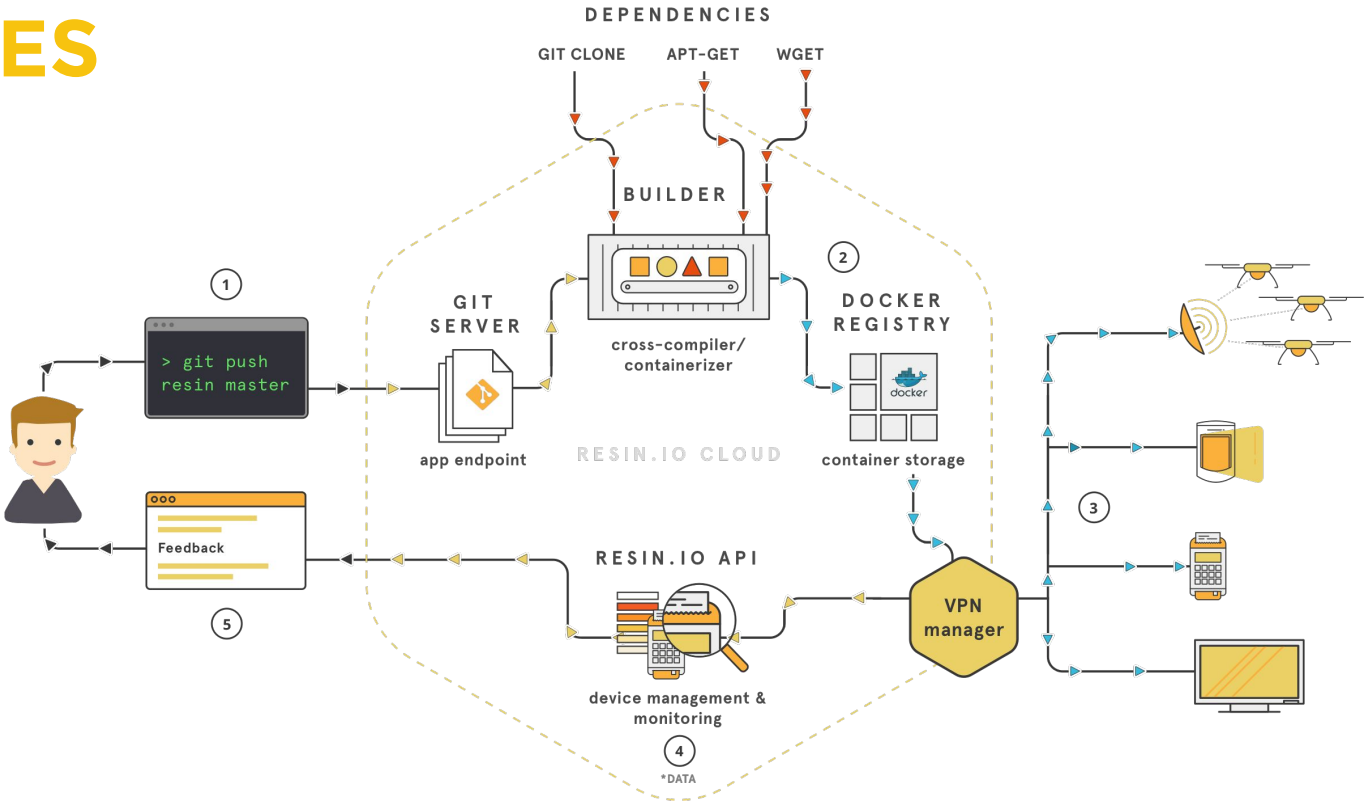
# BEHIND THE SCENES



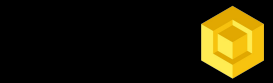
# BEHIND THE SCENES



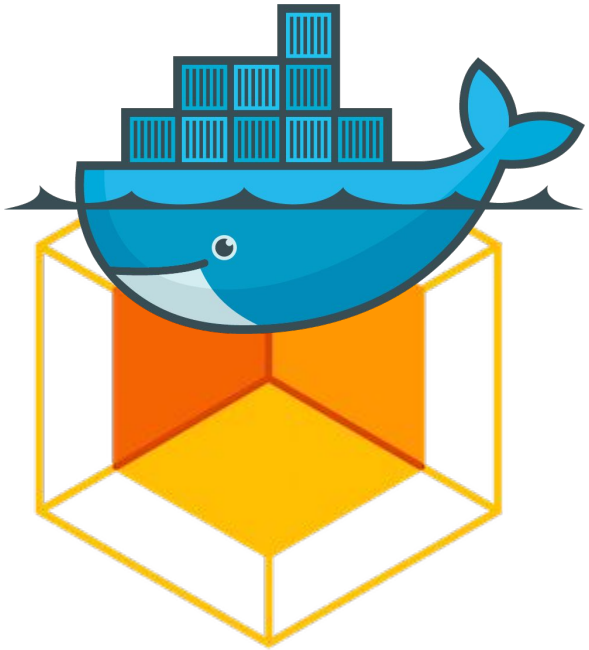
# BEHIND THE SCENES



# SO WHAT?

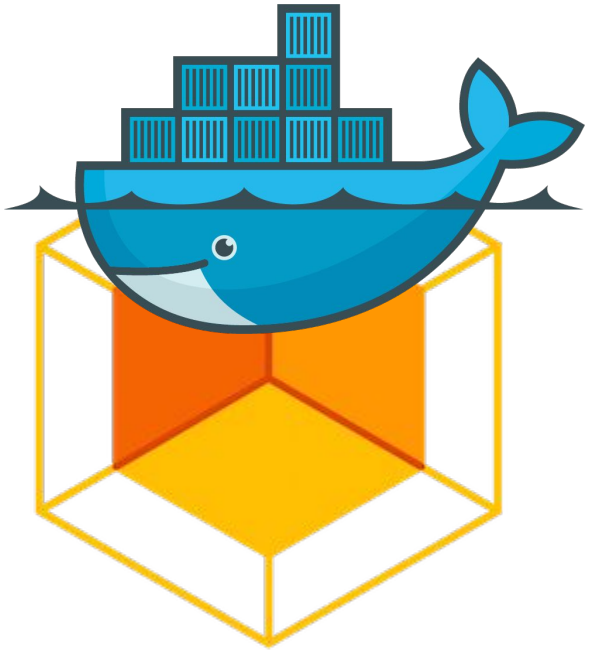


# WHY DOCKER ON AN EMBEDDED DEVICE?



# WHY DOCKER ON AN EMBEDDED DEVICE?

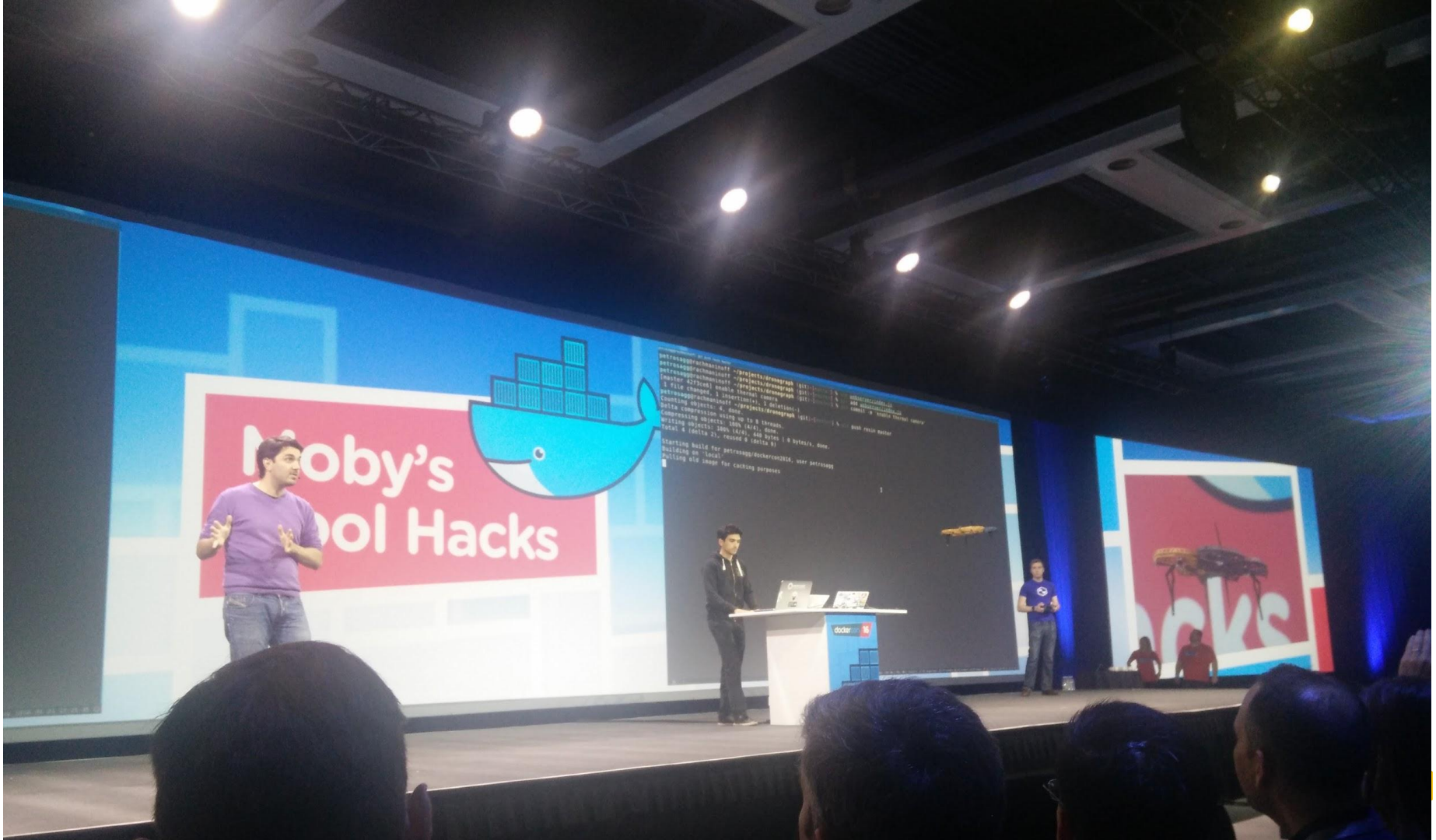
# WHY DOCKER ON AN EMBEDDED DEVICE?



# WHY DOCKER ON AN EMBEDDED DEVICE?

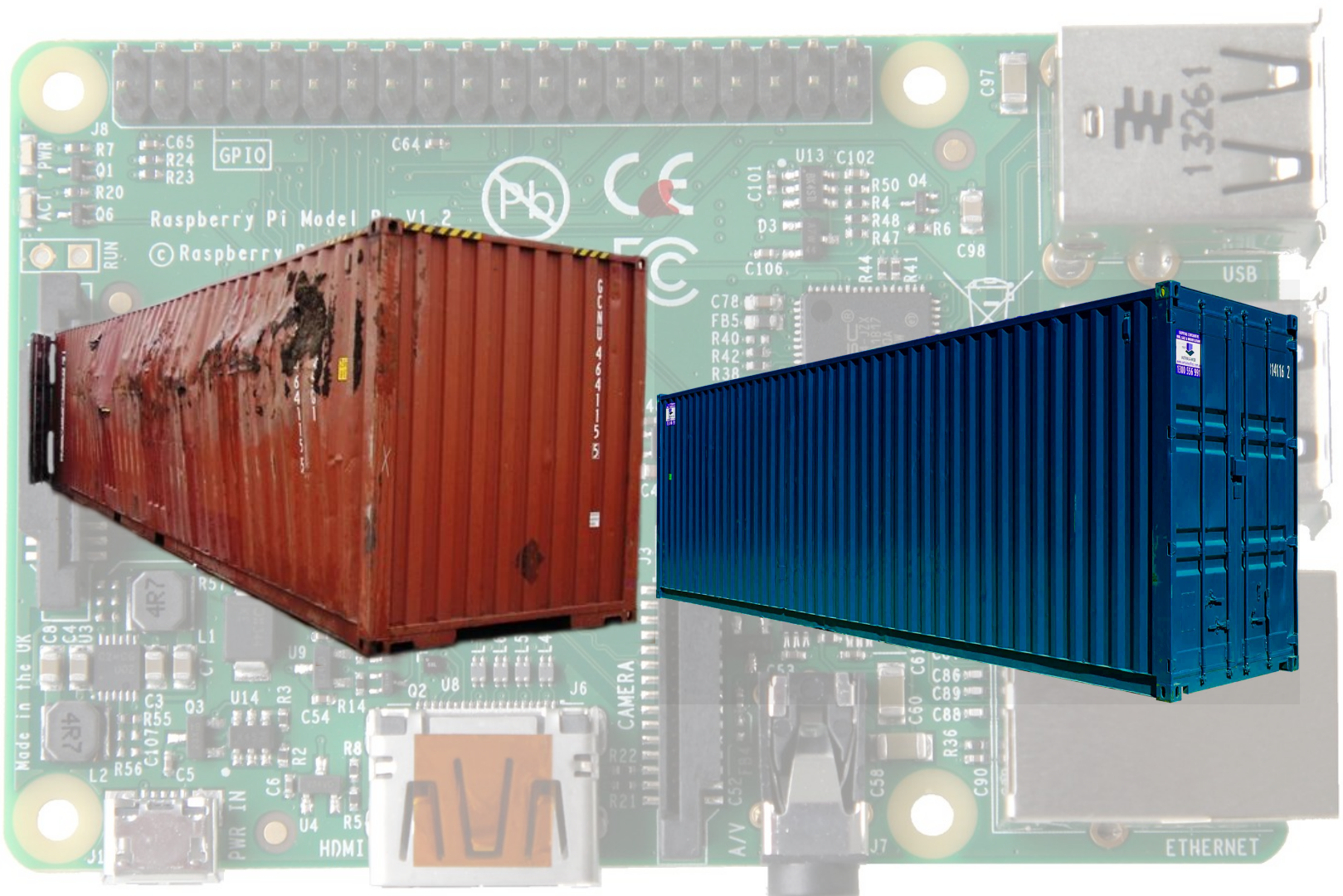
# WHY DOCKER ON AN EMBEDDED DEVICE?

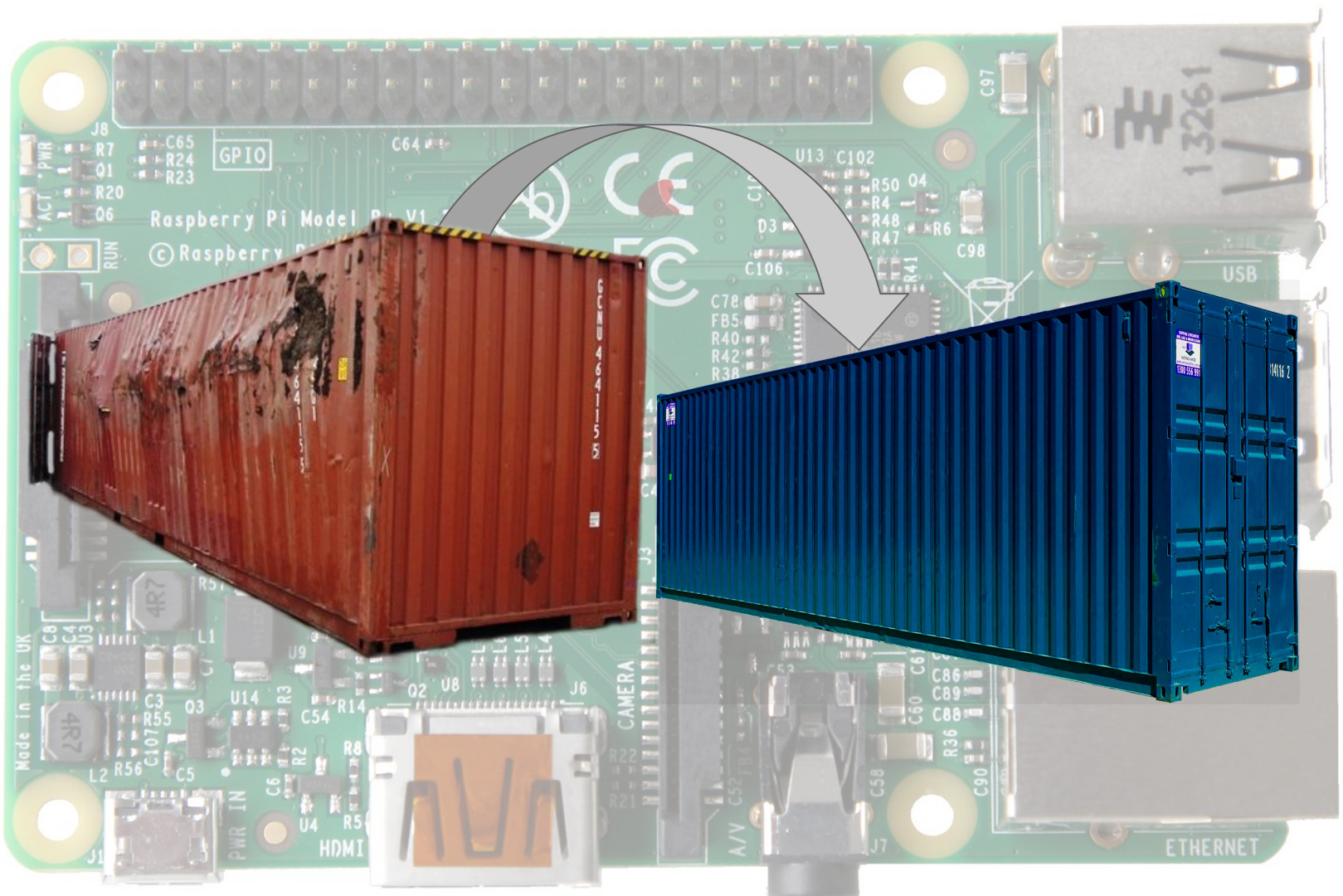

USING DOCKER + RESIN.IO
TO UPDATE DEVICES OVER THE AIR
Moby's
ool Hacks

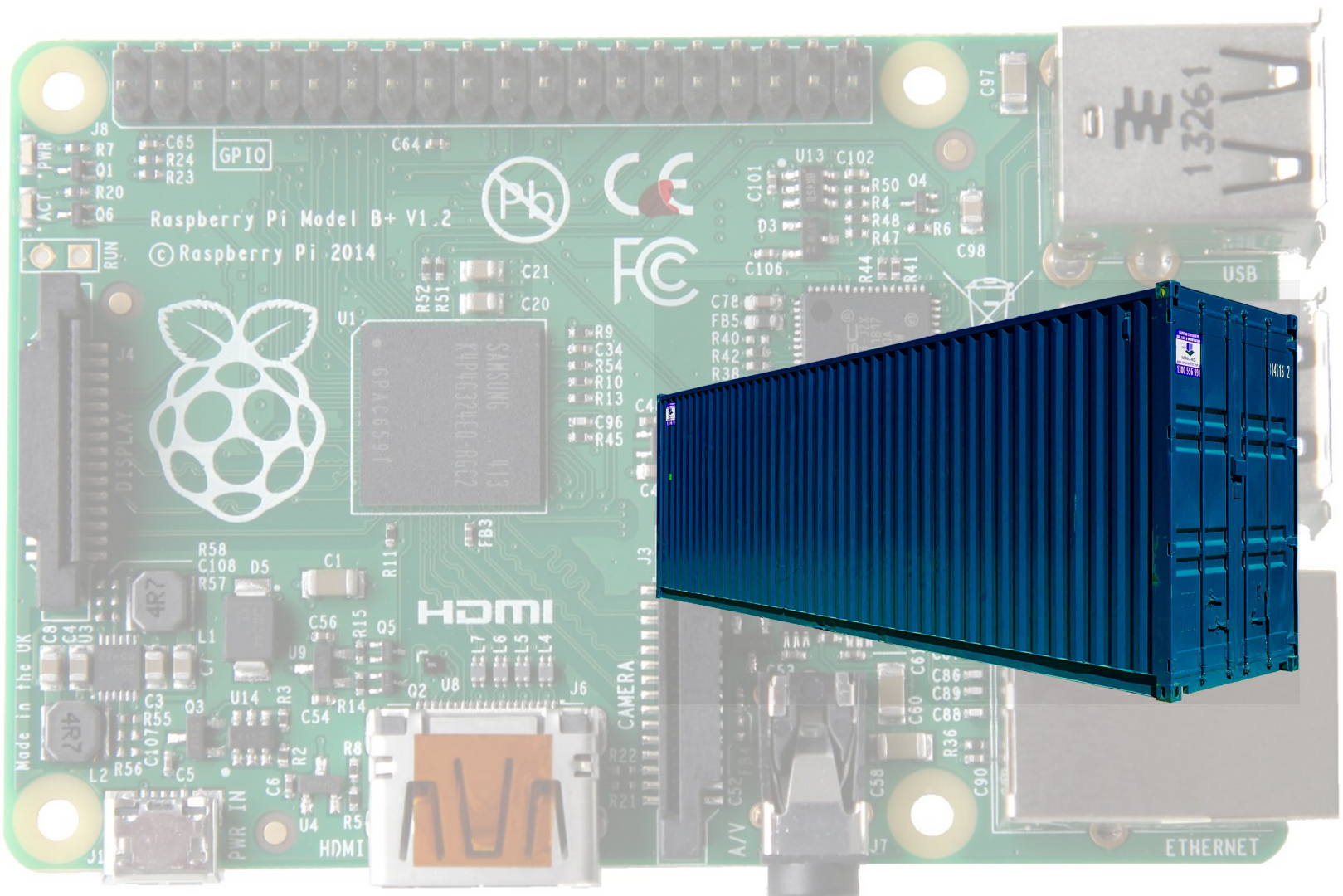

oby's
ool Hacks


Drone Control Panel
Gyroscope
Thermal Camera
oby's
ool Hacks


GPIO
Raspberry Pi Model B V1.2
USB
USB
ETHERNET
HDMI
CAMERA
A/V
PWR IN
Made in the UK


GPIO
Raspberry Pi Model B V1.2
USB
HDMI
CAMERA
A/V
ETHERNET
PWR IN
Made in the UK


GPIO
Raspberry Pi Model B+ V1
USB
HDMI
CAMERA
DSI
A/V
ETHERNET
PWR IN
Made in the UK


Raspberry Pi Model B+ V1.2
© Raspberry Pi 2014
GPIO
HDMI
CAMERA
DISPLAY
USB
ETHERNET
PWR IN
A/V
Made in the UK

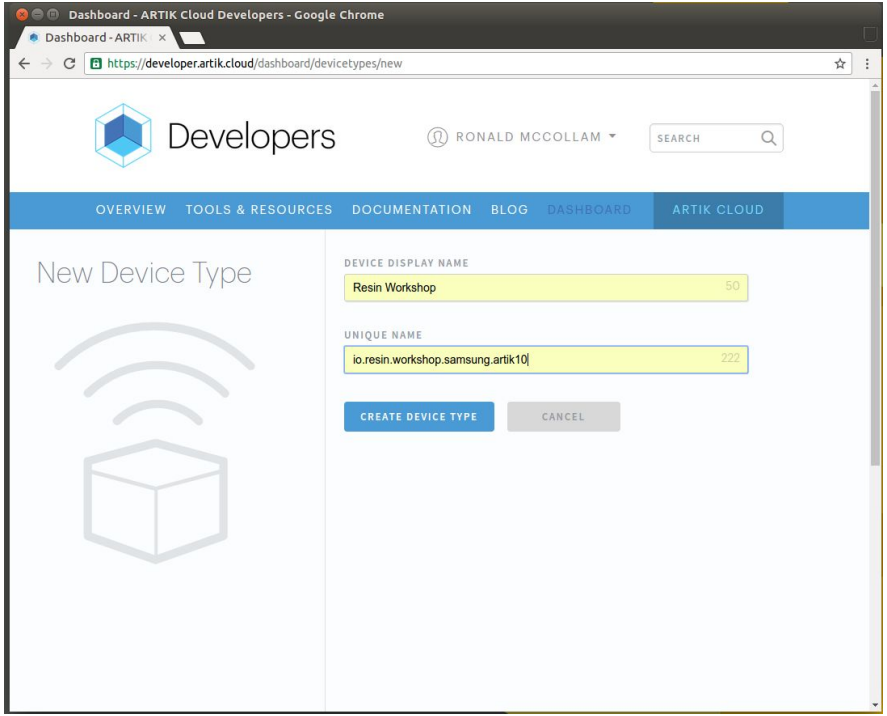
# LET'S GET HANDS ON!

# bit.ly/artikworkshop

# PRE-WORKSHOP: ARTIK CLOUD CREATE DEVICE

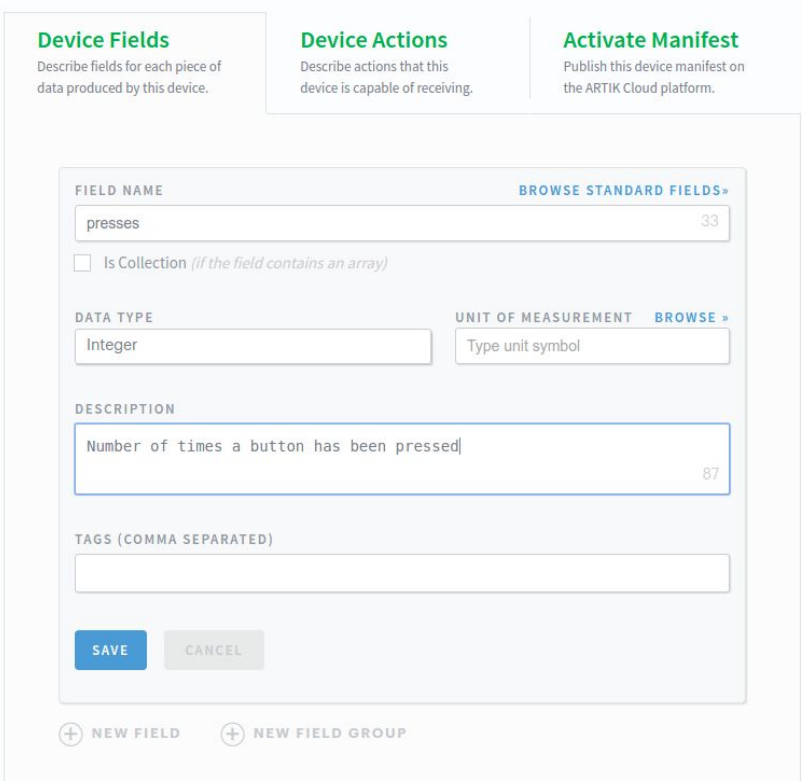
- Log in to developer.artik.cloud
- Your device types - New device
- Fill in names
  - Display name is up to you
  - Unique name could be e.g.

    `com.samsung.workshop.yourname`



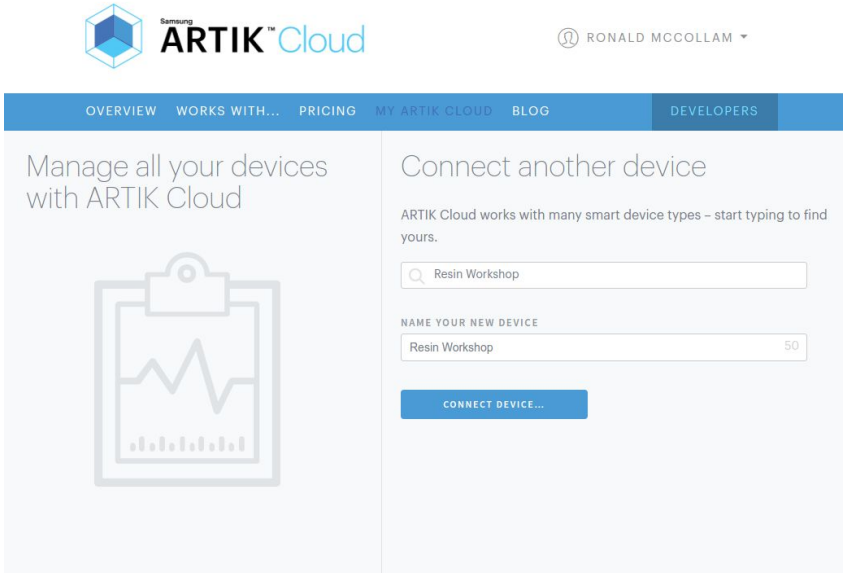
# PRE-WORKSHOP: ARTIK CLOUD MANIFEST

- Click "New manifest"
- Fill in fields and click "Save"
- Select "Activate Manifest" tab and click "Activate manifest"



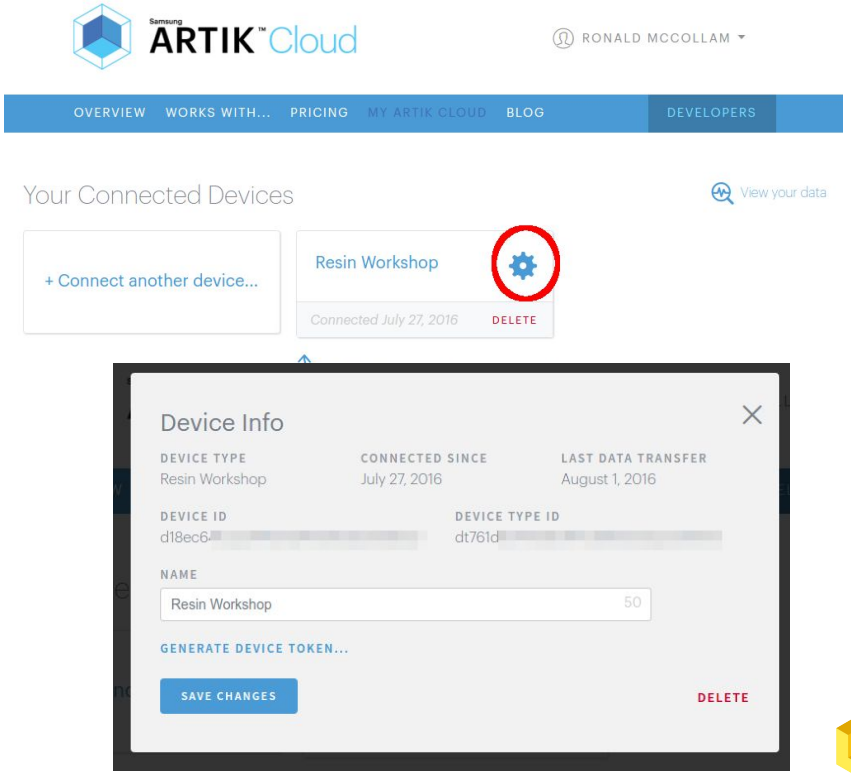
# PRE-WORKSHOP: ARTIK CLOUD CONNECT DEVICE

- Go to **`artik.cloud/my`**
- Click "Devices" then "Connect another device"
- Find the device you just created and click "Connect device..."



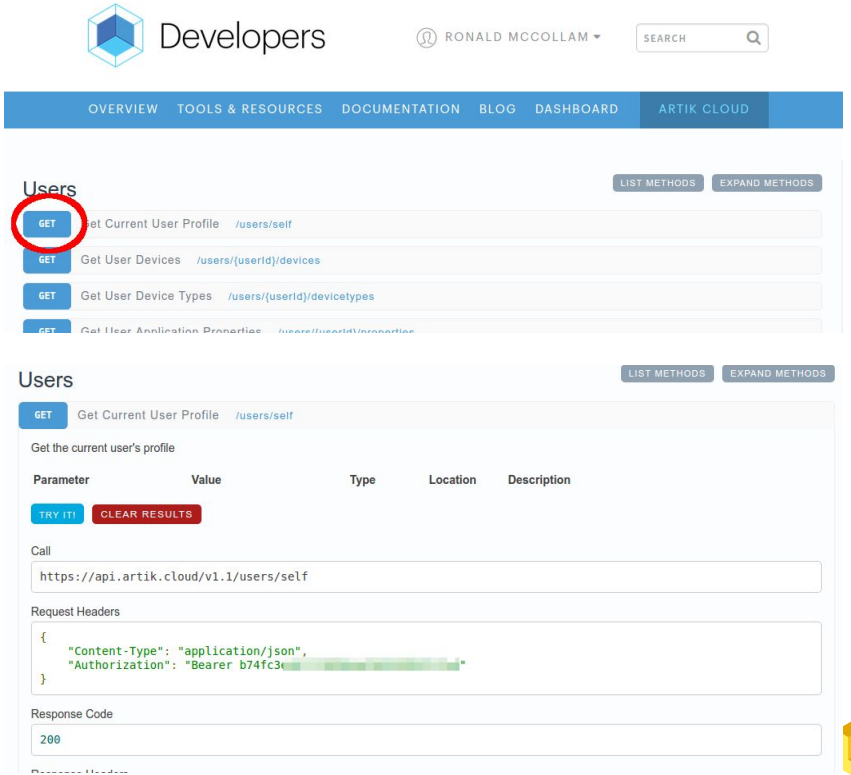
# PRE-WORKSHOP: ARTIK CLOUD CONNECT DEVICE

- Under "Your Connected Devices", click the gear icon in the upper right of the device you just created
- Copy the "Device ID" string and paste it into a text document or similar for later



# PRE-WORKSHOP: ARTIK CLOUD CREDENTIALS

- Go to `developer.artik.cloud/api-console`
- Click "Get" next to "Get Current User Profile
- Click "Try it!"
- Copy the string next to the word "Bearer" after "Authorization" and paste it in a text document (do not mix it up with the device ID from before!)



# WORKSHOP

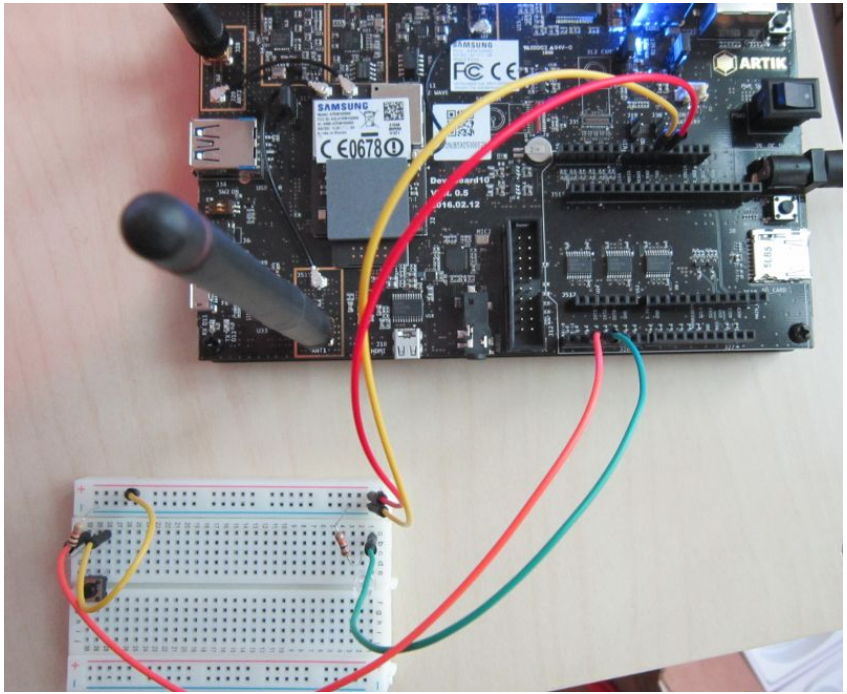
# WORKSHOP PREP

- Create account on resin.io
- Create an application
- Flash your device
- Clone my project

```
git clone https://github.com/resin-io-projects/artik-workshop
```

# WORKSHOP PREP: BREADBOARD

- Connect the breadboard:
  - + to +5v
  - - to GND
- Bottom-most row to J26 pin 2 (**third** from the left!)
- Top-most row to J26 pin 3 (**fourth** from the left!)



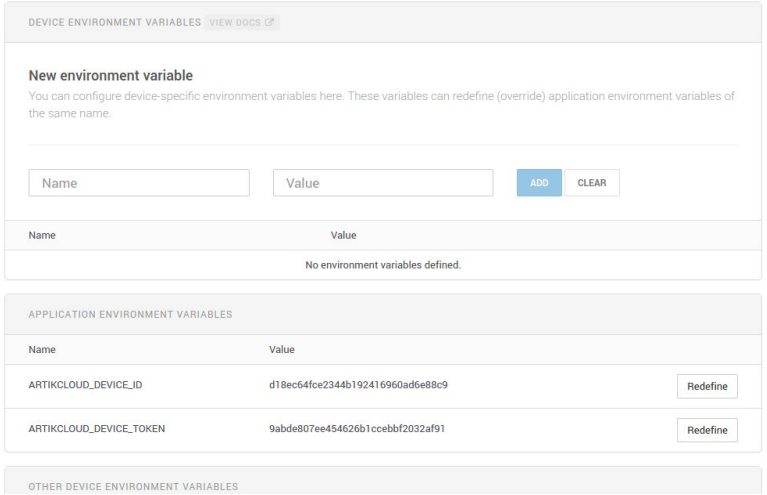
# WORKSHOP BACKGROUND

We will use git to **clone** and **push** a simple application to the Artik.  This application will use the Artik's GPIO functionality to detect presses of a button and toggle an LED.

Later we will fix bugs and even add cloud functionality to the application, all without ever having to re-flash the Artik or risk bricking it.

# WORKSHOP PREP: DEVICE SETUP

- Open your device
- Click "Environment Variables" on the left
- Add **`ARTIKCLOUD_DEVICE_ID`** and paste your device ID
- Add **`ARTIKCLOUD_DEVICE_TOKEN`** and paste your authentication




# WORKSHOP DOCKERFILE.TEMPLATE

```
FROM resin/%%RESIN_MACHINE_NAME%%-alpine-python:latest

WORKDIR /app
COPY . ./

RUN pip install -r ./requirements.txt

CMD /app/buttons.py
```

# WORKSHOP BUTTONS.PY

```
while True:
  if workshop.is_pushed(button):
    print "Button pushed!"
    presses = presses + 1
    plug.toggle_light("192.168.5.XXX")
    # Uncomment the next line to work with the cloud!
    #workshop.send_pressed_message(cloud, presses)
    time.sleep(.25)
```

# WORKSHOP PUSH TO ARTIK

Copy from the top right of the resin.io dashboard:

```
git remote add resin XXXX
```

Now push the code:

```
git push resin master
```

# TRY IT!

# STEP 1: FIXING A BUG

```
while True:
  if workshop.is_pushed(button):
    print "Button pushed!"
    presses = presses + 1
    plug.toggle_light("192.168.5.XXX")
    # Uncomment the next line to work with the cloud!
    #workshop.send_pressed_message(cloud, presses)
    workshop.toggle(led)
    time.sleep(.25)
```

# STEP 2: DEPLOY

```
git add buttons.py
git commit -m 'Fixed the switch IP address'
git push resin master
```

# TRY IT!

# STEP 3: ADDING A FEATURE

```
while True:
  if workshop.is_pushed(button):
    print "Button pushed!"
    presses = presses + 1
    plug.toggle_light("192.168.5.XXX")
    # Uncomment the next line to work with the cloud!
    #workshop.send_pressed_message(cloud, presses)
    workshop.toggle(led)
    time.sleep(.25)
```

# STEP 4: DEPLOY

```
git add buttons.py
git commit -m 'Integrate with Artik Cloud'
git push resin master
```

# WATCH IT WORK

https://artik.cloud/my

Click "**Charts**"

(You may need to add the chart and click the play button)

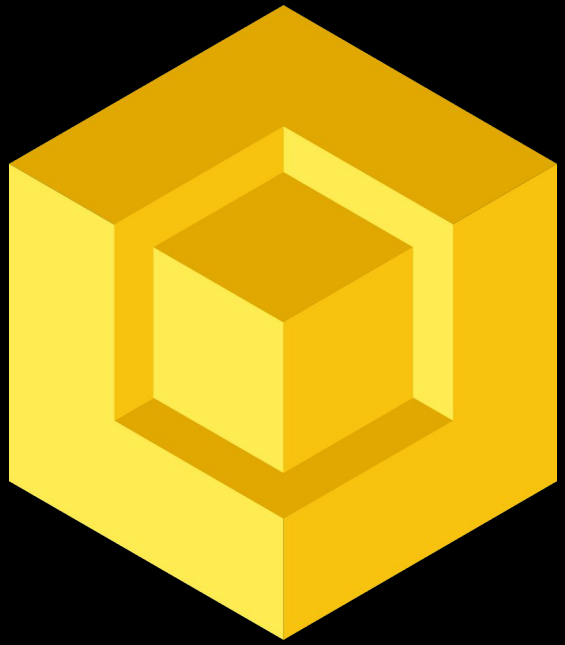
# TRY IT!

CONGRATULATIONS!

You just deployed and updated a remote application!

# QUESTIONS?

# THE FUTURE



*Thank you!*

Ronald McCollam
Solutions Architect

ronald@resin.io @RonaldMcCollam

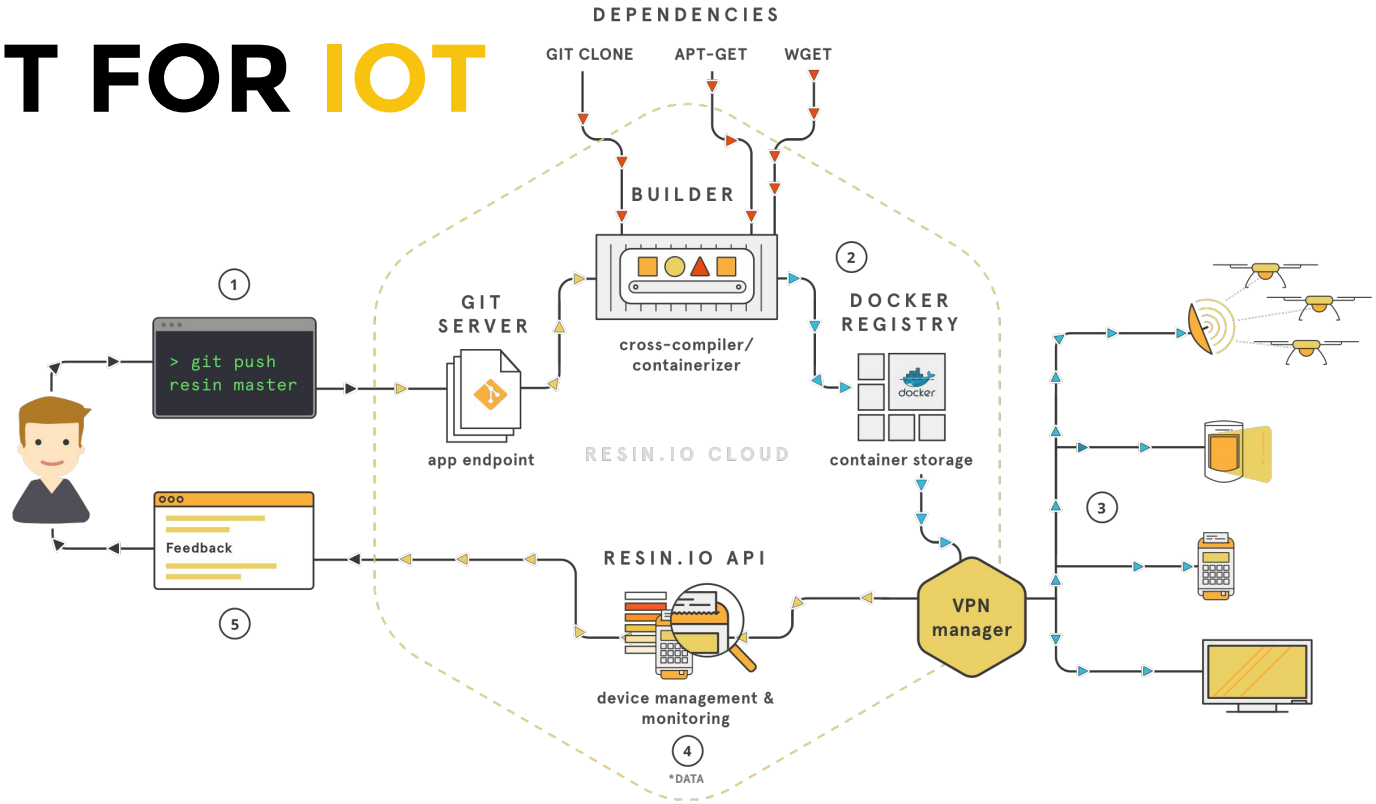
# APPENDIX

# BASE IMAGES

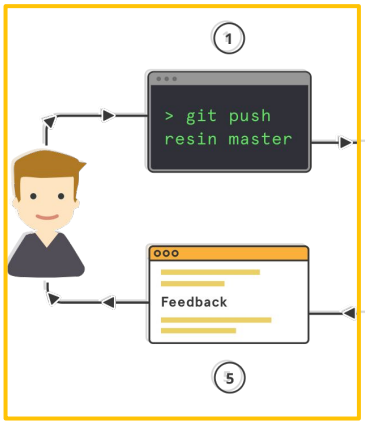
http://docs.resin.io/runtime/resin-base-images/

- Debian, Alpine, Fedora (coming soon)
- Python, Node.js, Go variants

# PURPOSE BUILT FOR IOT

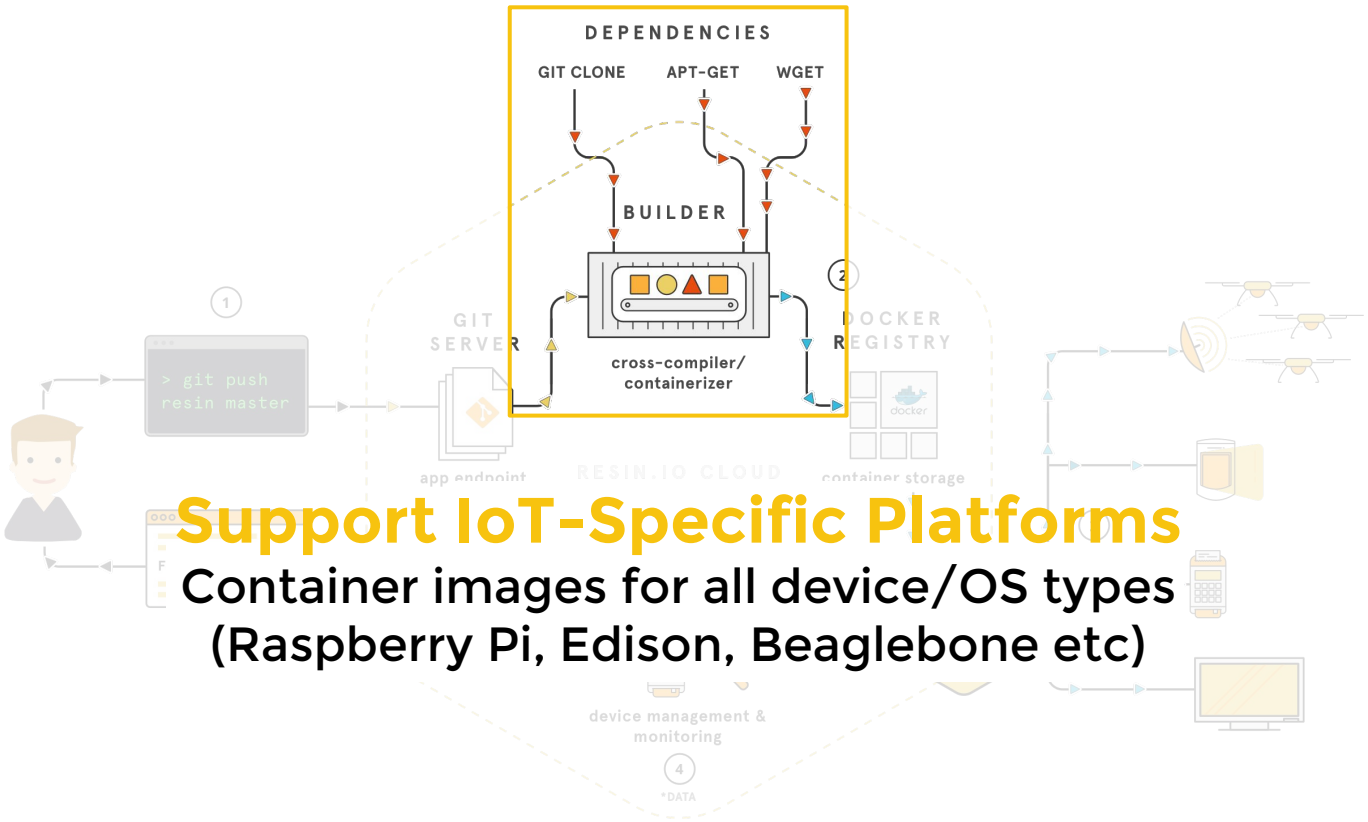


# BEST PRACTICES FROM THE CLOUD...




## "Look and Feel" of Cloud

Git push changes, run unit and integration tests, compatible with Docker, Travis, Jenkins etc

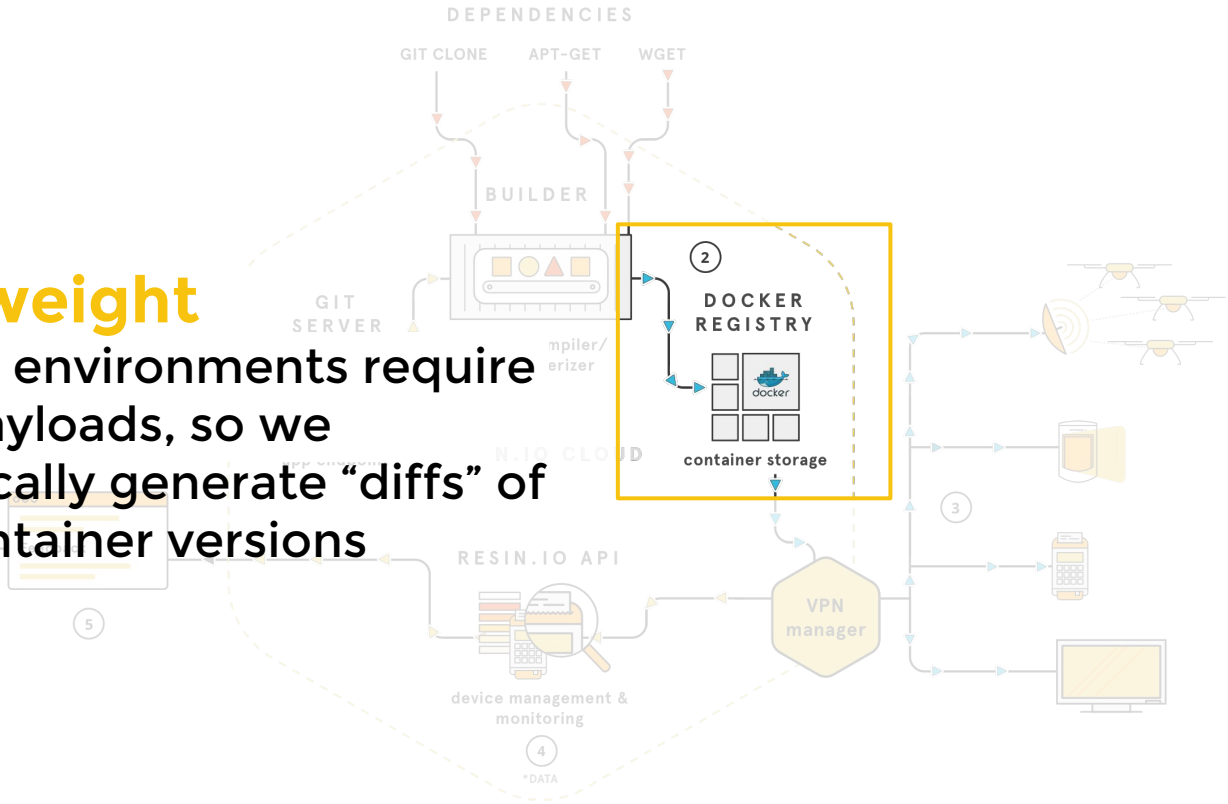



# Support IoT-Specific Platforms

Container images for all device/OS types
(Raspberry Pi, Edison, Beaglebone etc)

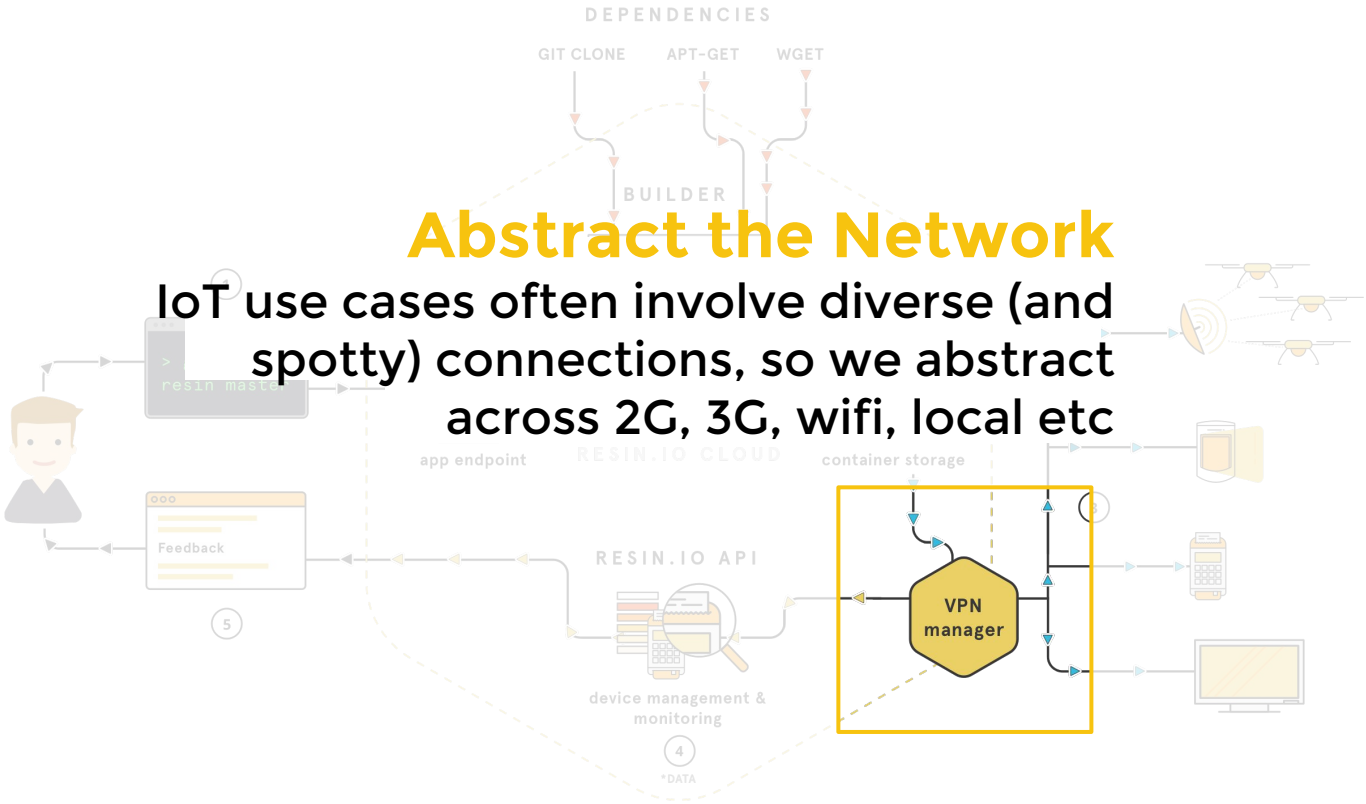
# Lightweight

Remote environments require small payloads, so we dynamically generate “diffs” of new container versions

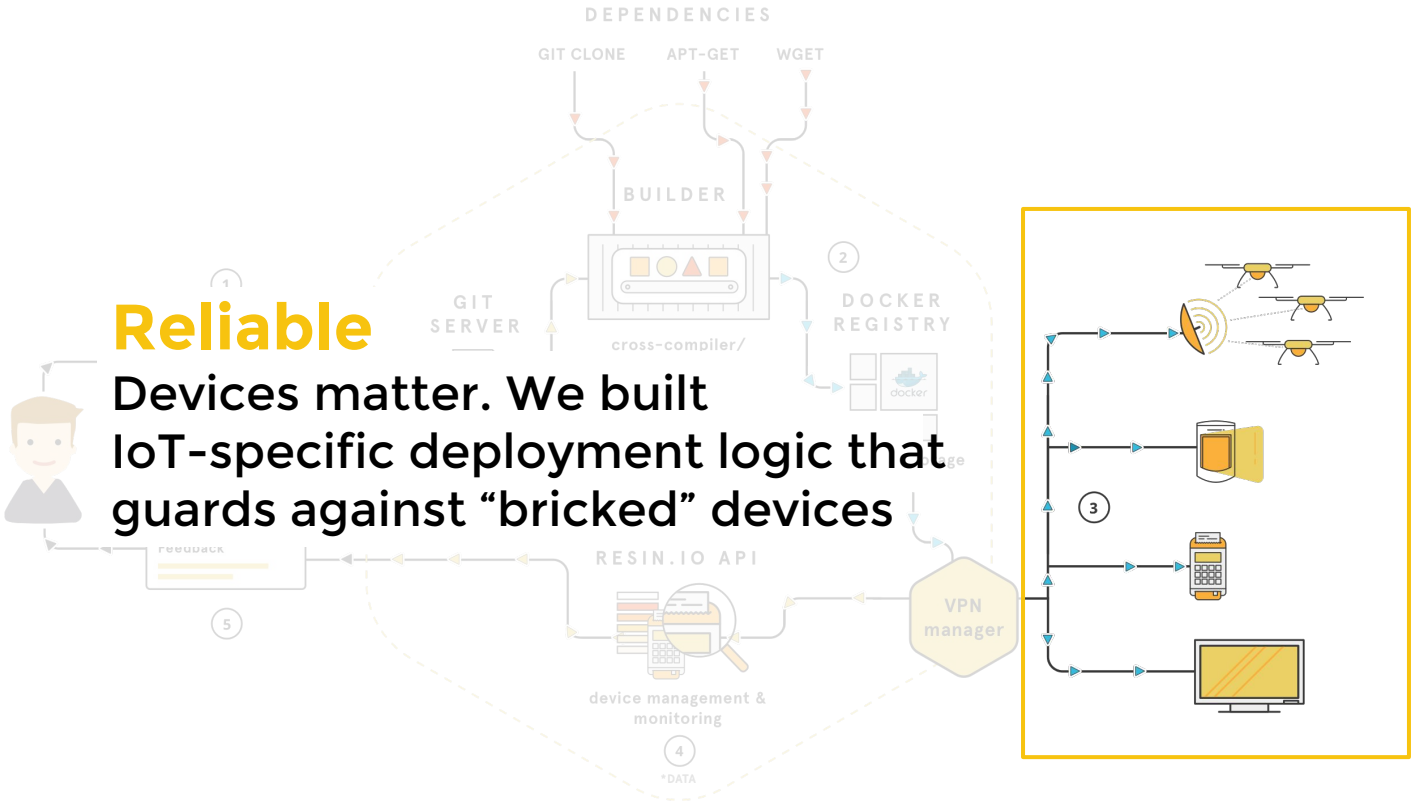


# Abstract the Network

IoT use cases often involve diverse (and spotty) connections, so we abstract across 2G, 3G, wifi, local etc



# PURPOSE BUILT FOR IoT




## Reliable

Devices matter. We built IoT-specific deployment logic that guards against “bricked” devices

# ...SOLVING HARD IOT PROBLEMS

## Architecture agnostic

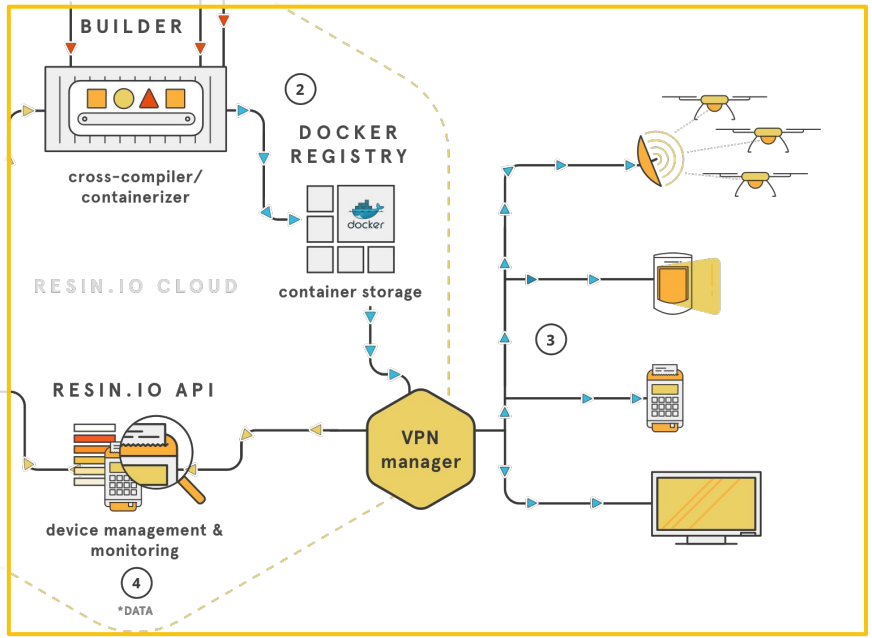
IoT devices have diverse CPU architectures. Our builders compile for the target device architecture

## Lightweight

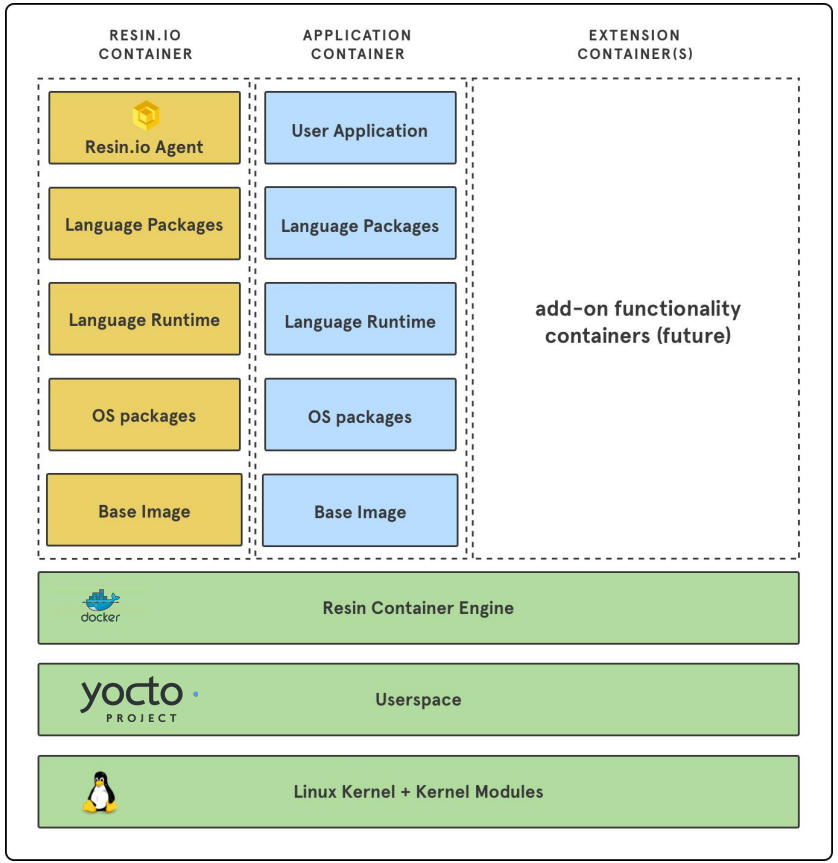
Remote environments require small payloads, so we dynamically generate “diffs” of new container versions

## Fail-Safe Deployments

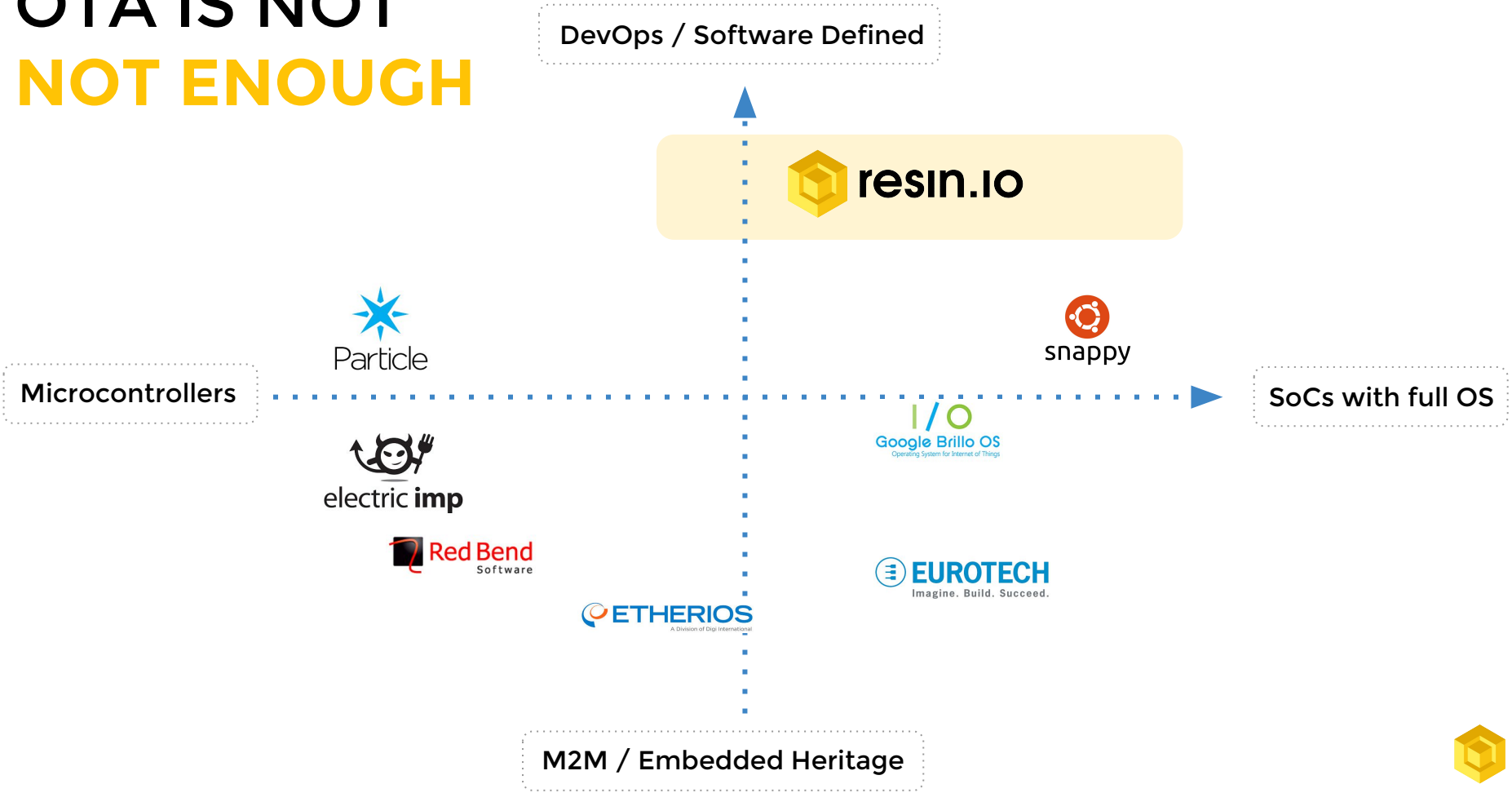
IoT-specific deployment logic that guards against “bricked” devices



# ON-DEVICE S/W ARCHITECTURE



# OTA IS NOT NOT ENOUGH



# USE CASES ACROSS THE INDUSTRIAL INTERNET OF THINGS

Manufacturing & Logistics

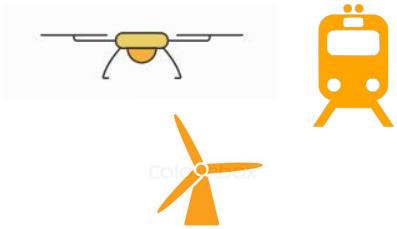
Retail / Point of Service

Smart Buildings

Industrial Lighting

Digital Signage

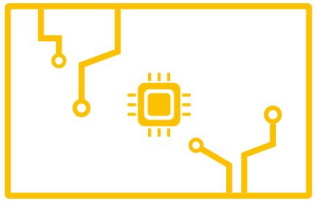
Many Others...



# DEVICE PROVISIONING PROCESS

1

Download preconfigured OS from resin.io server



2

Write image to device storage (at lab / factory)

3

Power device up and connect to network.

---

As soon as the device is online, it automatically connects to the resin server, and provisions itself into the application fleet its image was configured for.

Resin can generate images preloaded with the user's container, which runs before provisioning. This means that the device can execute arbitrary network provisioning/bootstrapping steps before provisioning to the resin.io server.

ETCHER
BY RESIN.IO

Repository DOWNLOAD FOR WINDOWS

# Burn. Better.

Burn images to SD cards & USB drives, **safe & easy.**

DOWNLOAD FOR WINDOWS

*v1.0.0-beta.10*

Tweet Star 674