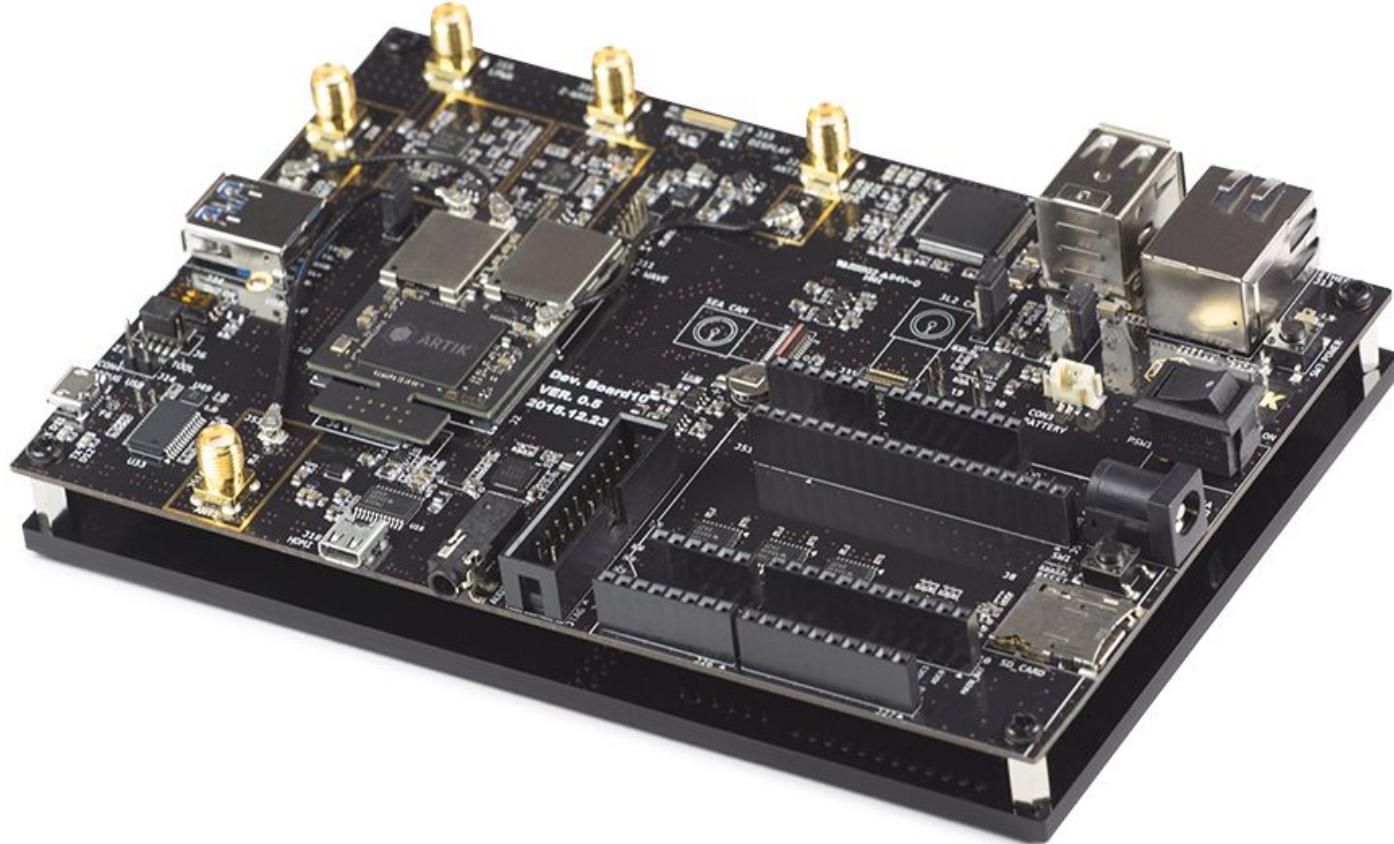
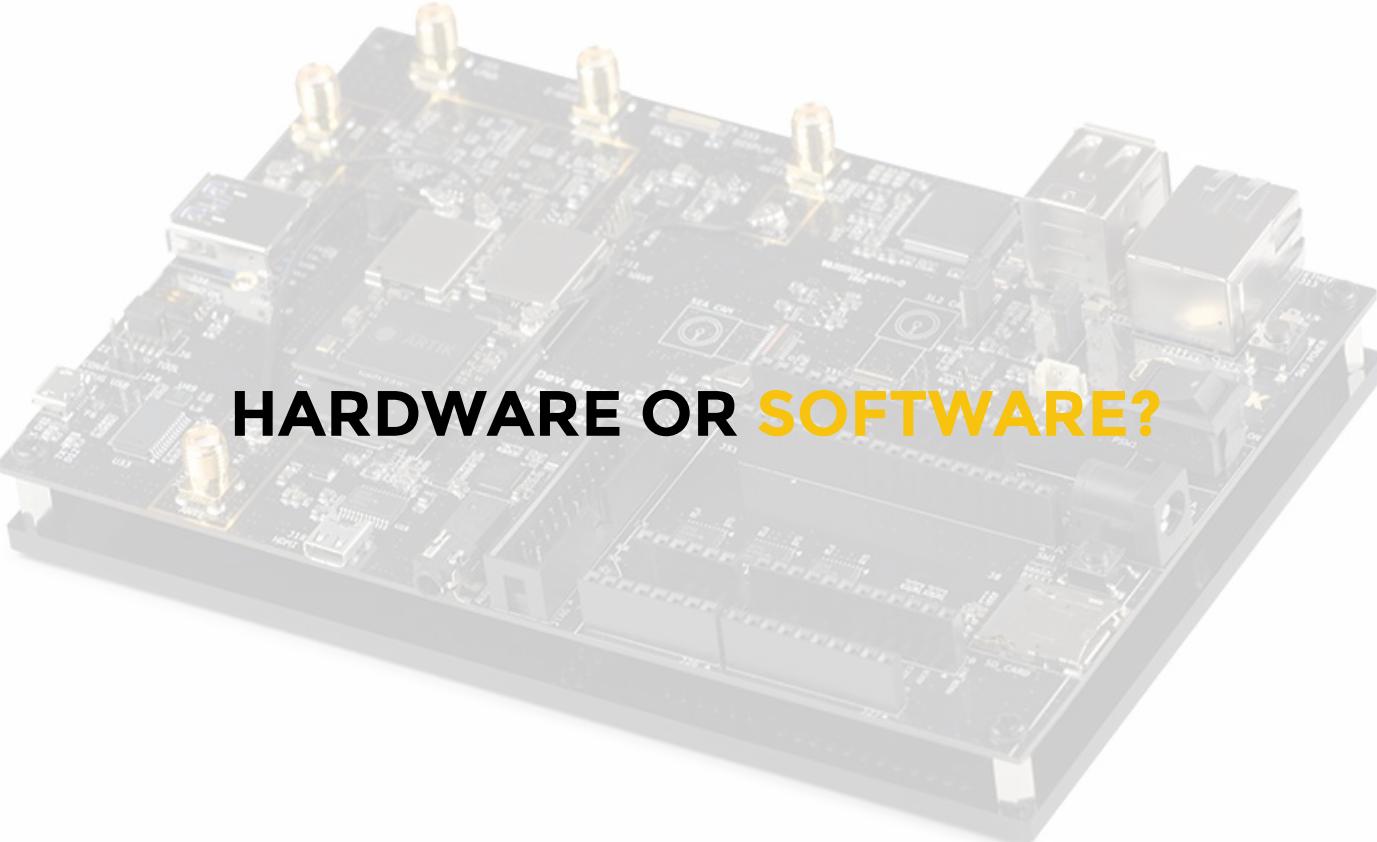


resin.io

Samsung Artik Workshop
2016/08/09

Ronald McCollam
Solutions Architect
ronald@resin.io [@RonaldMcCollam](https://twitter.com/RonaldMcCollam)





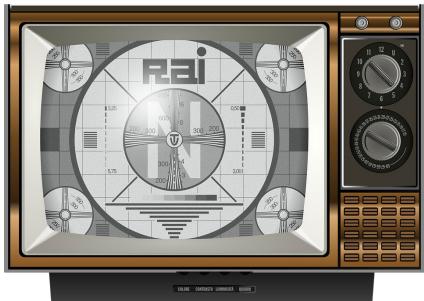
HARDWARE OR SOFTWARE?



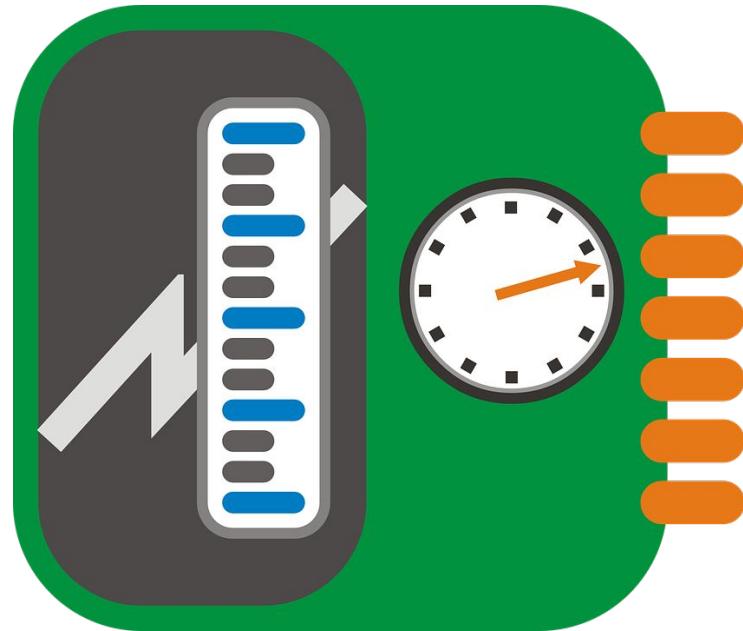
HARDWARE OR SOFTWARE?



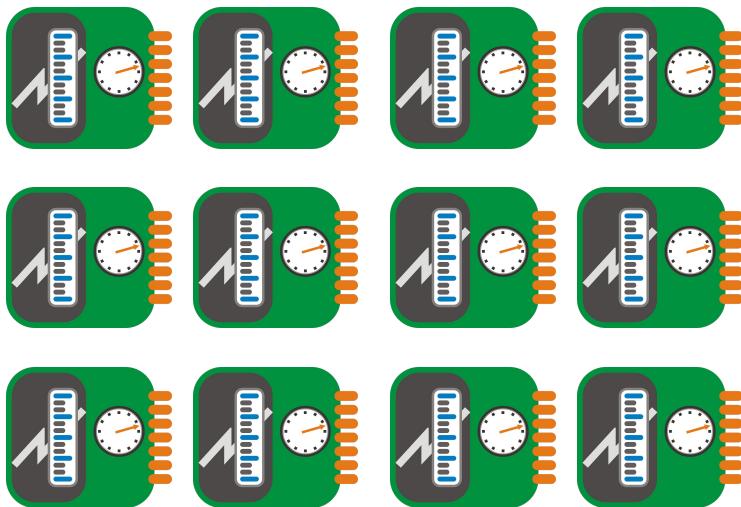
HARDWARE OR SOFTWARE?



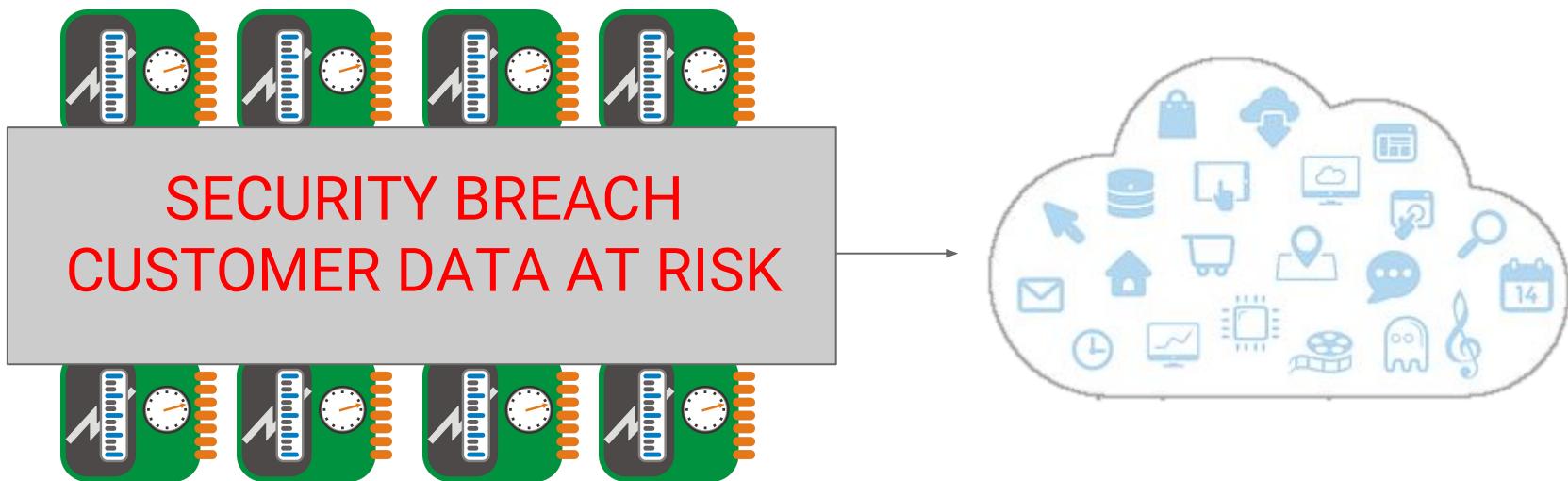
HARDWARE OR SOFTWARE?



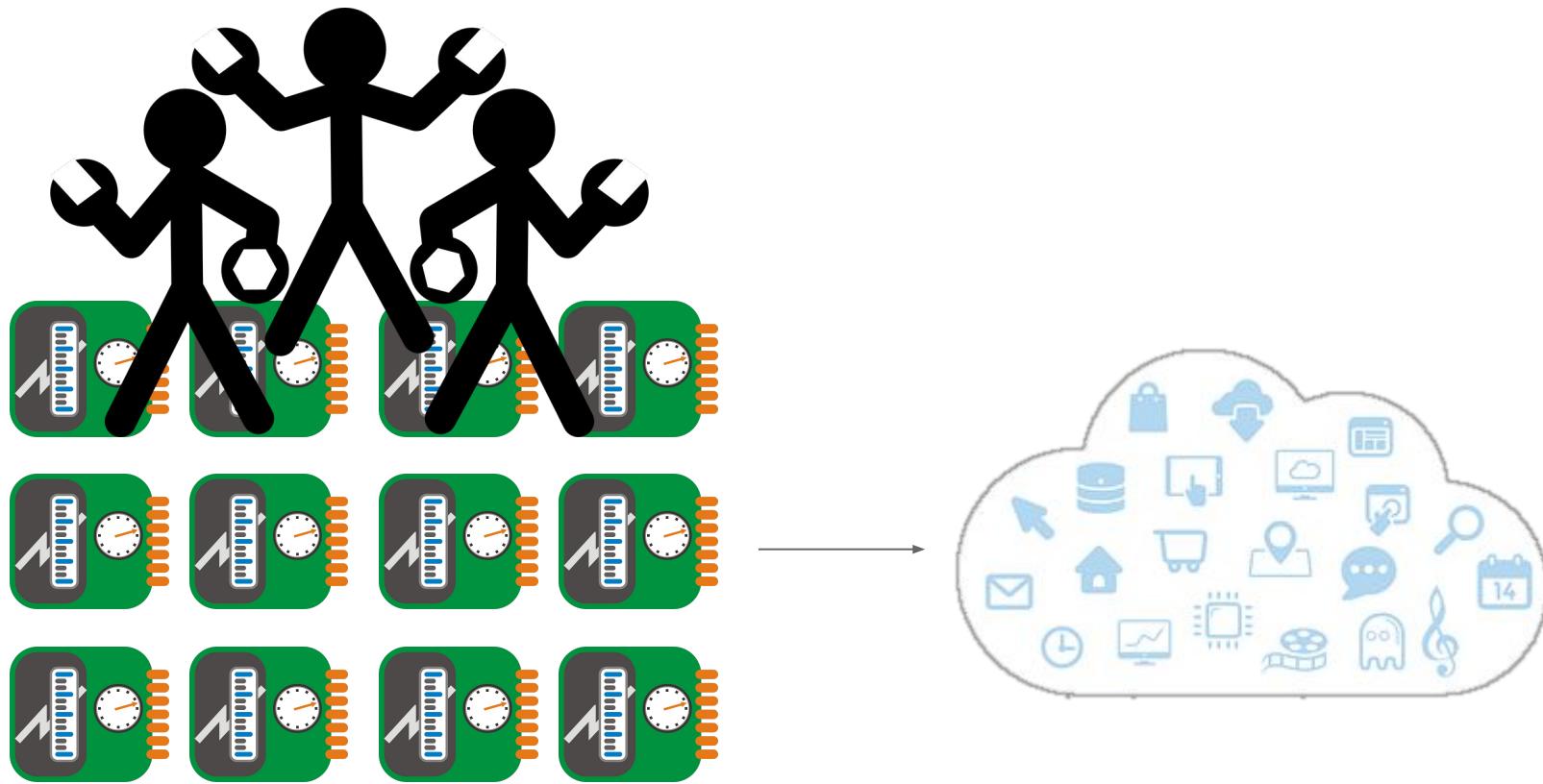
HARDWARE OR SOFTWARE?



HARDWARE OR SOFTWARE?



HARDWARE OR SOFTWARE?



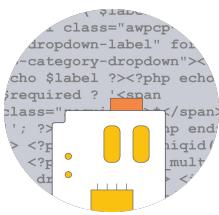


WE LIVED THE
PROBLEM

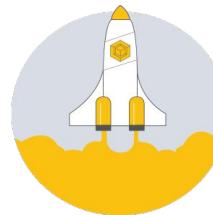


HOW DOES RESIN.IO HELP USERS?

"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



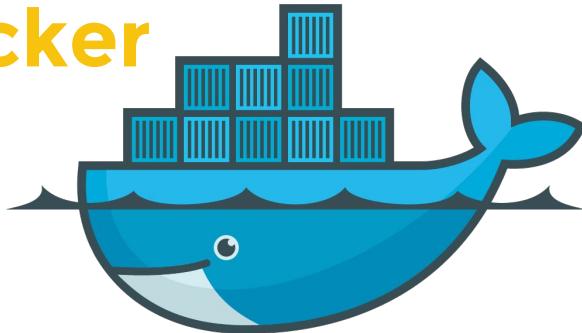
TECHNICAL BACKGROUND

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



BACKGROUND:

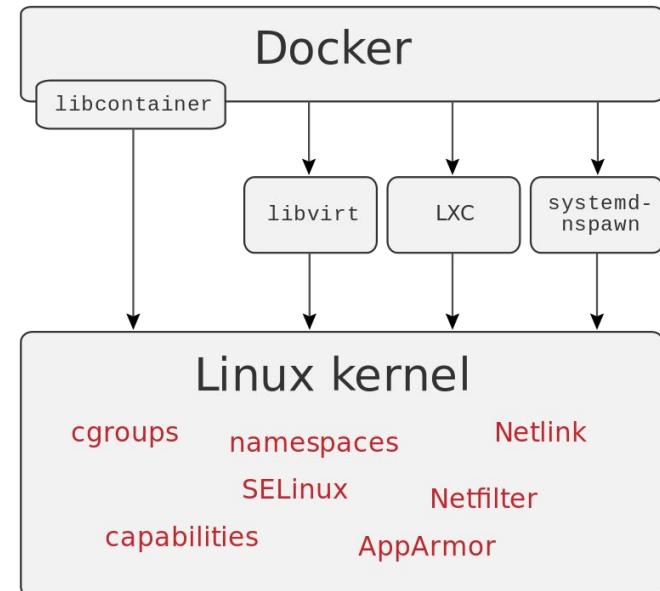
docker



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

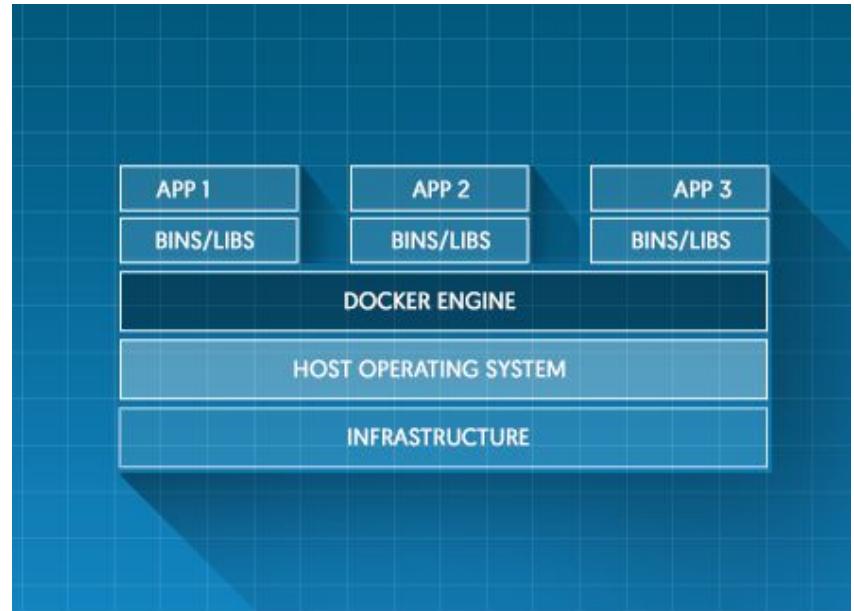
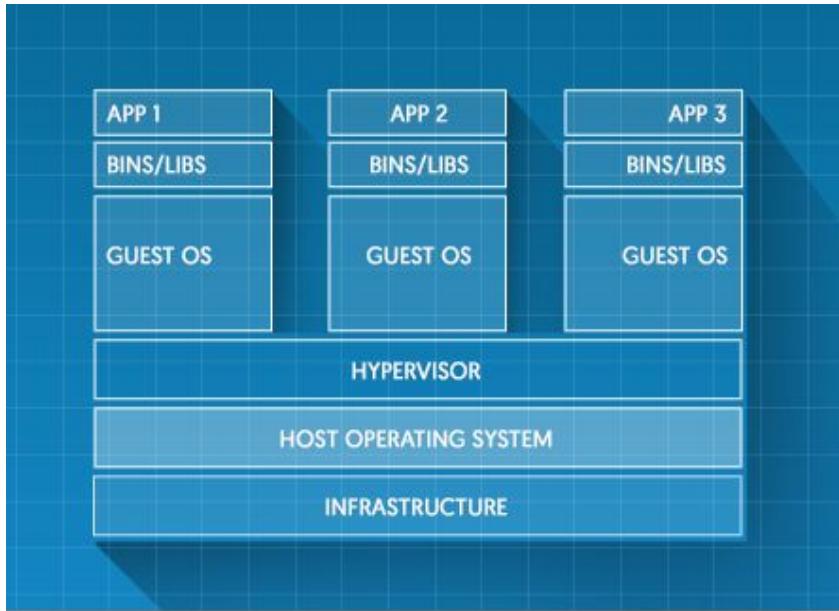
COPY . /usr/src/app
WORKDIR /usr/src/app

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



BACKGROUND: docker

Virtual machines vs. Docker

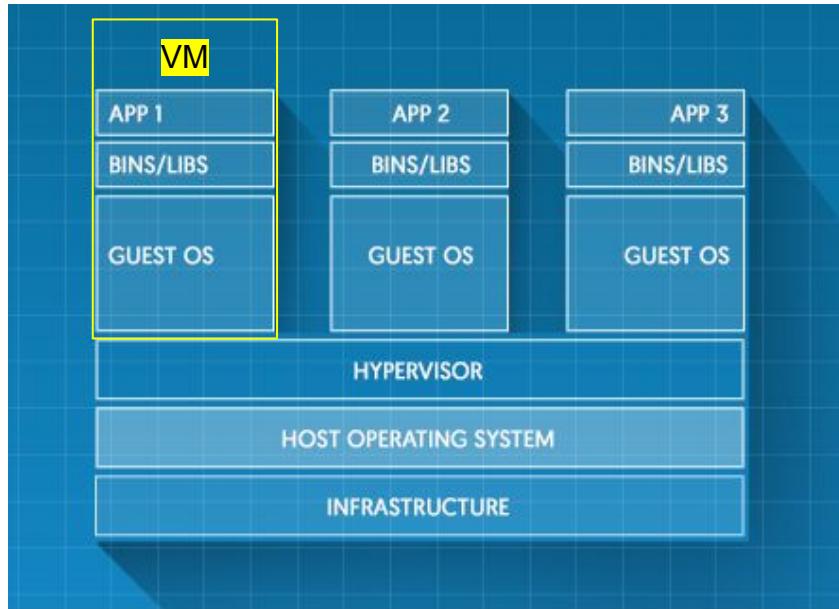


(shamelessly stolen from <https://www.docker.com/what-docker>)



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

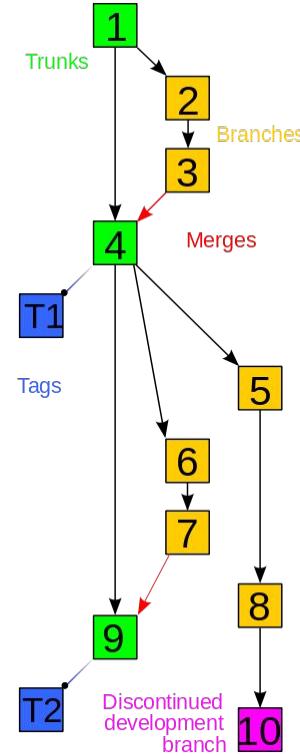
git clone	Copy data from a repository
git commit	Save your changes
git push	Send data to a repository



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 server¹)
- 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'

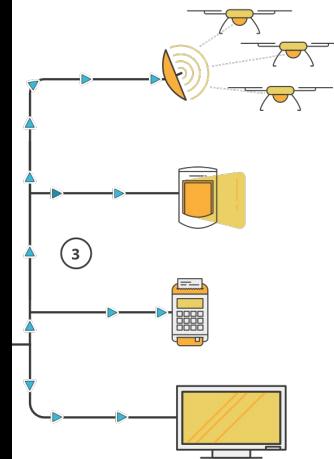
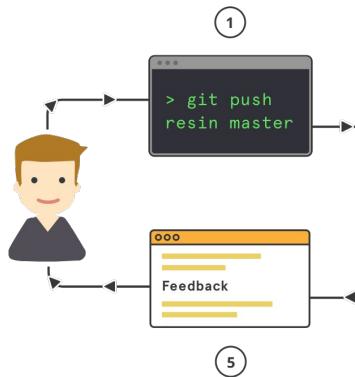
Pulling old image for caching purposes

```
[=====] 1 kB/1 kB
```

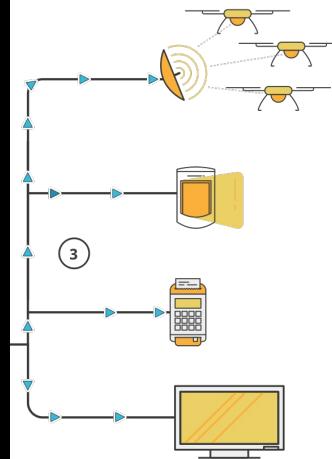
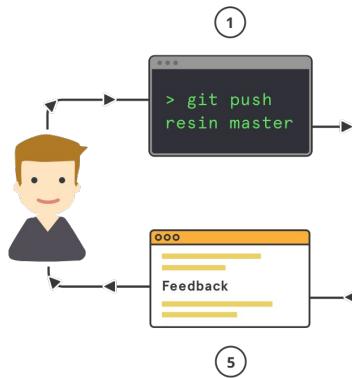
...



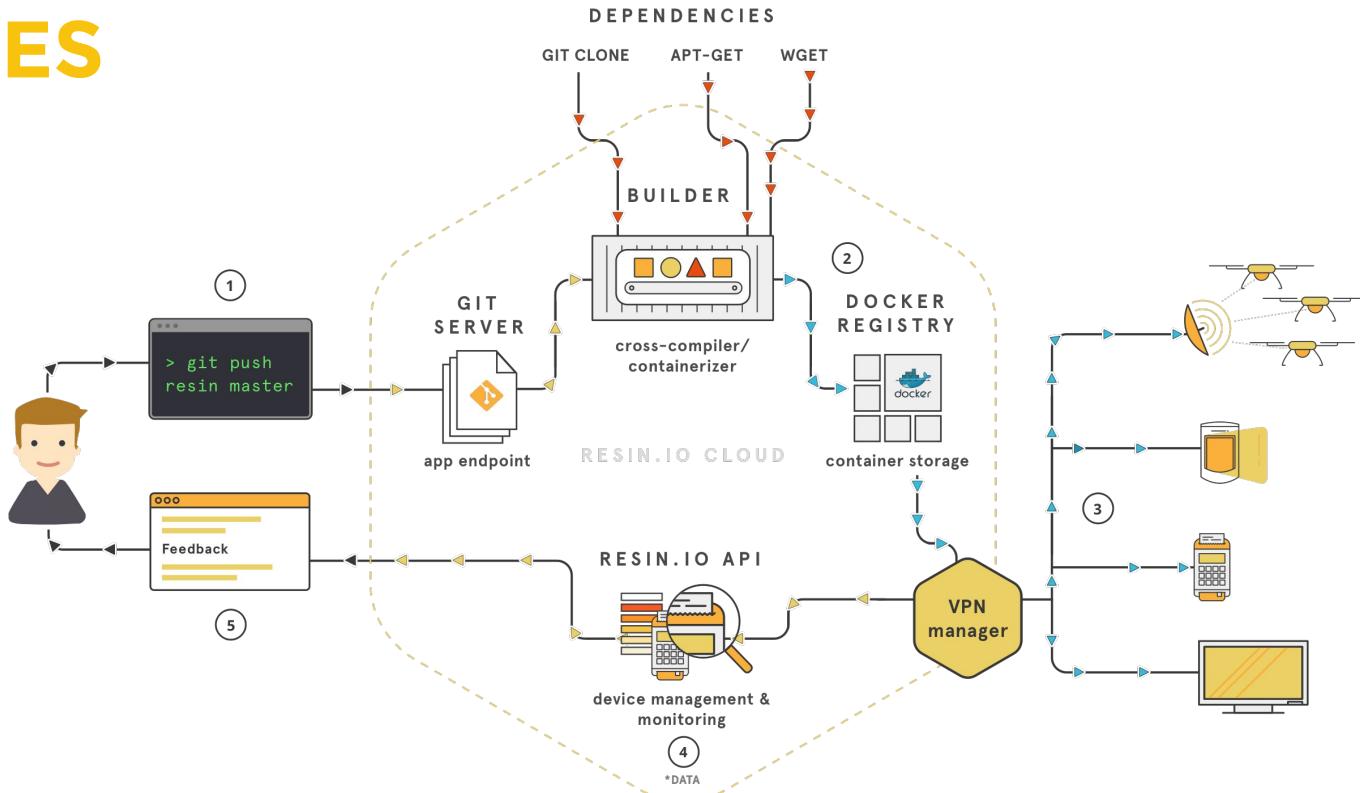
BEHIND THE SCENES



BEHIND THE SCENES



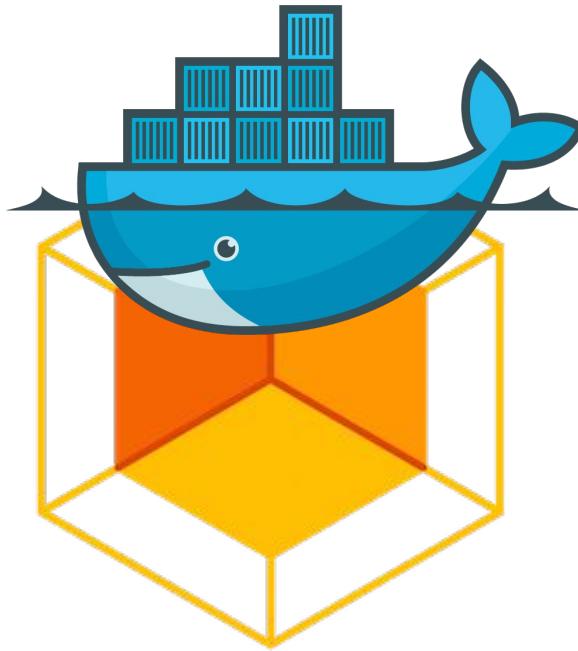
BEHIND THE SCENES



SO WHAT?



WHY DOCKER ON AN EMBEDDED DEVICE?

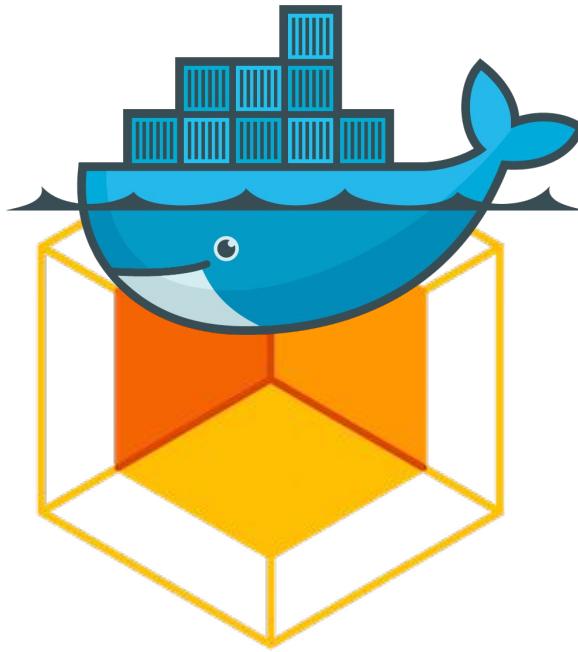


WHY DOCKER ON AN EMBEDDED DEVICE?





WHY DOCKER ON AN EMBEDDED DEVICE?



WHY DOCKER ON AN EMBEDDED DEVICE?



WHY DOCKER ON AN EMBEDDED DEVICE?



USING DOCKER + RESIN.IO

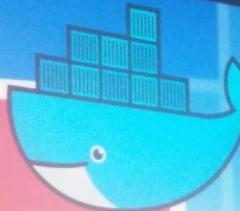


TO UPDATE DEVICES OVER THE AIR

Moby's
Cool Hacks



Moby's ool Hacks

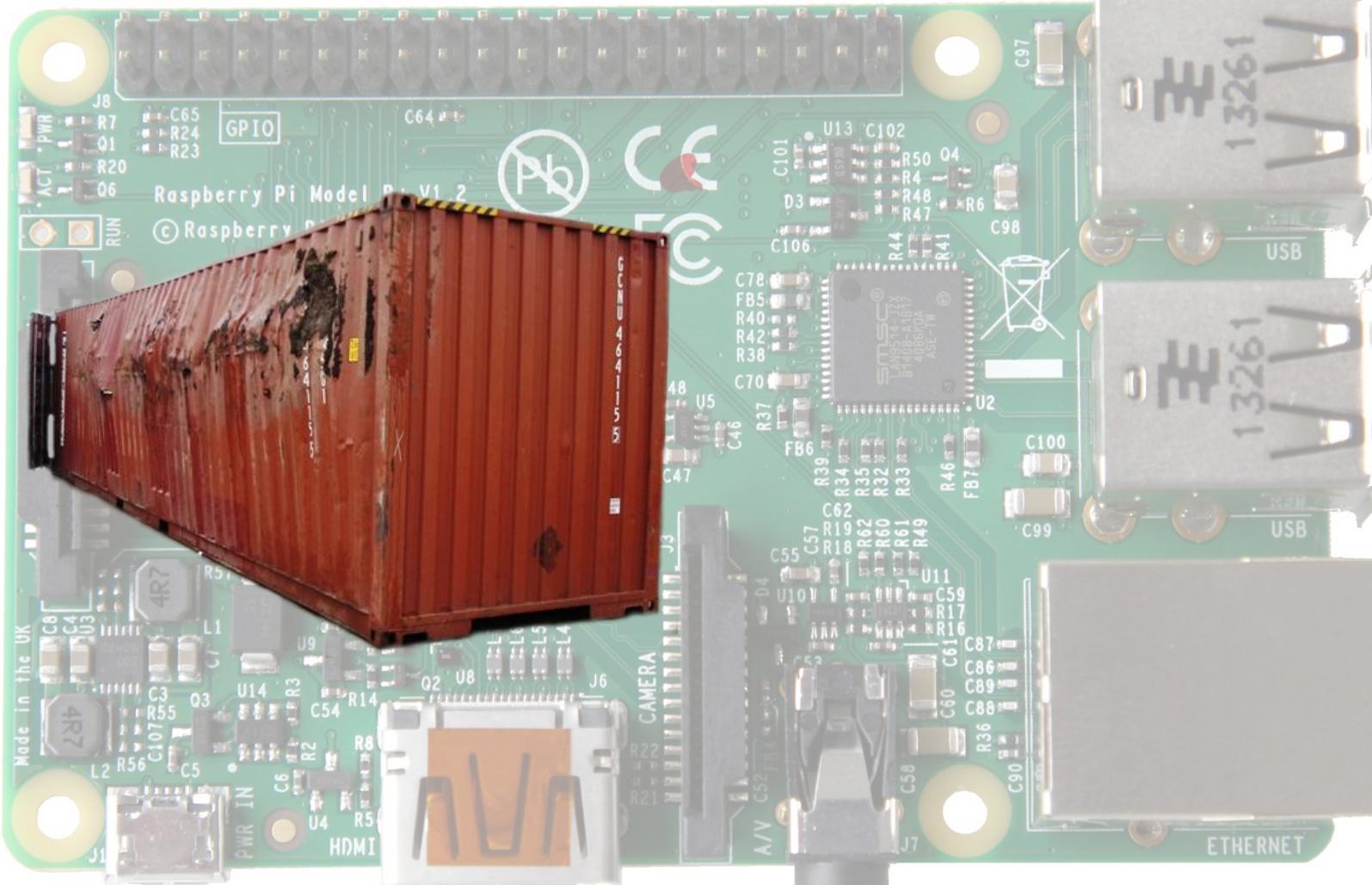


```
petraship@archanamart:~/parrot$ docker run -it --rm petraship/archanamart:java-archanamart
java -version
java version "1.8.0_45"
Java(TM) SE Runtime Environment (build 1.8.0_45-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.45-b02, mixed mode)

petraship@archanamart:~/parrot$ git commit -m "commit a whale Docker image"
[master 42f3c6d] commit a whale Docker image
 1 file changed, 1 insertion(+), 1 deletion(-)
petraship@archanamart:~/parrot$ git push origin master
Delta object: 1 file, 1 delta
  Delta compression using up to 8 threads.
Compressing: 1000 objects (424K)
Total: 4 (delta 2), reused 0 (delta 0)
Starting build for petraship/dockercon2015, user petraship
Build in local
Putting old image for caching purposes
```

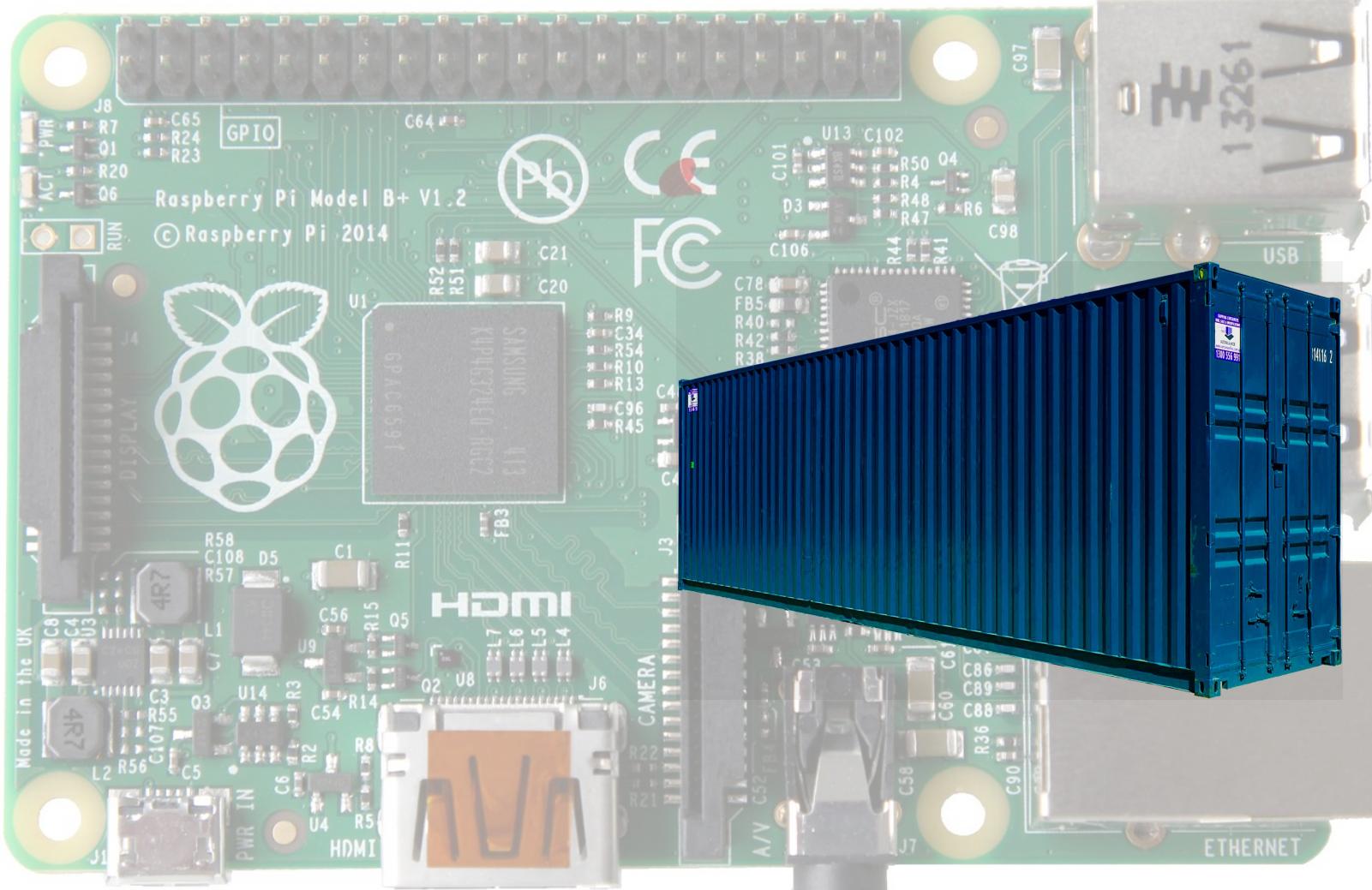












LET'S GET HANDS ON!



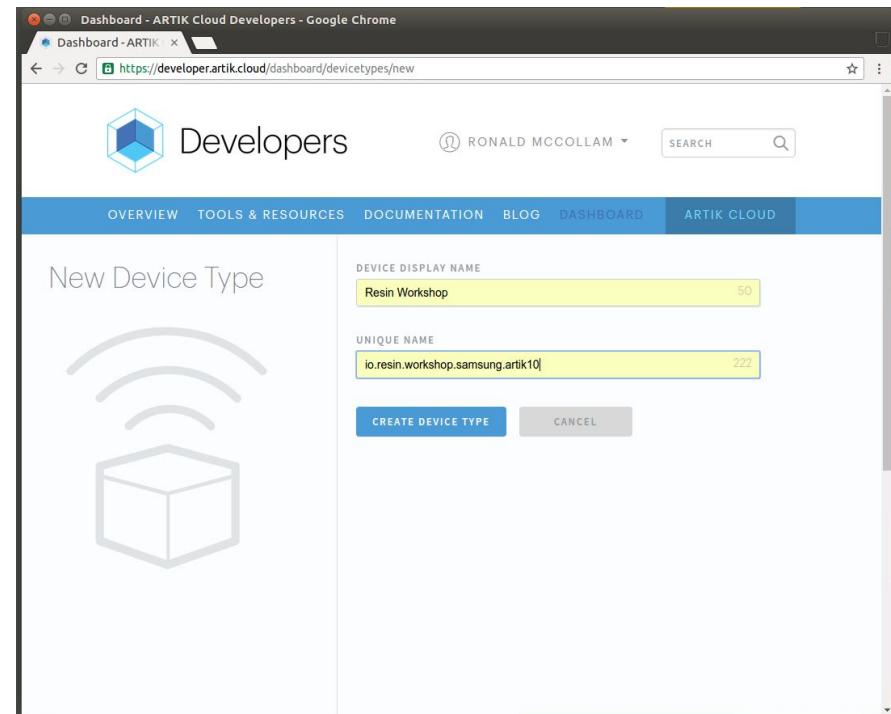
bit.ly/artikworkshop



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"

Device Fields
Describe fields for each piece of data produced by this device.

Device Actions
Describe actions that this device is capable of receiving.

Activate Manifest
Publish this device manifest on the ARTIK Cloud platform.

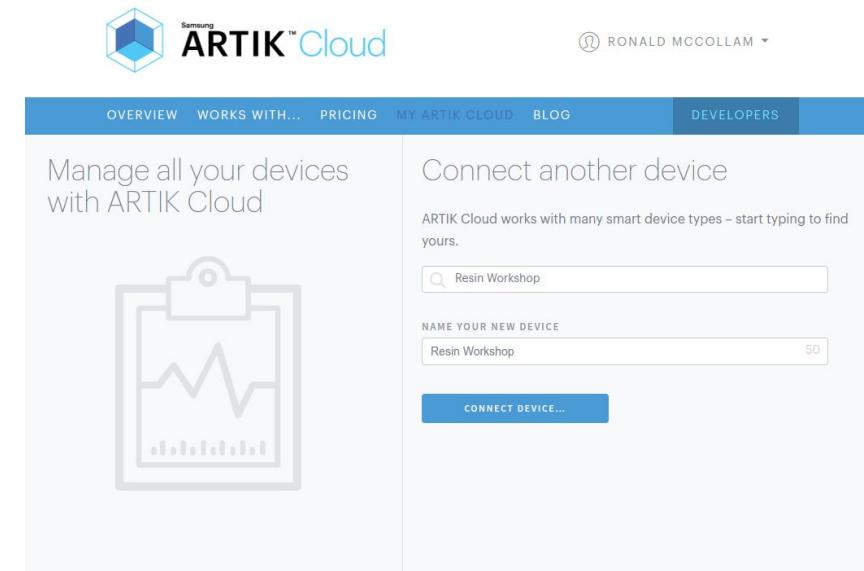
FIELD NAME presses <input type="checkbox"/> Is Collection <i>(if the field contains an array)</i>	BROWSE STANDARD FIELDS» 33
DATA TYPE Integer	UNIT OF MEASUREMENT BROWSE » Type unit symbol
DESCRIPTION Number of times a button has been pressed 87	
TAGS (COMMA SEPARATED) <input type="text"/>	
SAVE CANCEL	

NEW FIELD **NEW FIELD GROUP**



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

The screenshot shows the ARTIK Cloud dashboard. At the top, there's a navigation bar with the ARTIK logo, user profile, and links for Overview, Works With..., Pricing, My ARTIK Cloud, Blog, and Developers. Below the navigation is a section titled "Your Connected Devices". It contains a button "+ Connect another device..." and a card for a device named "Resin Workshop" which was connected on July 27, 2016. A red circle highlights the gear icon in the top right corner of the device card. Below this, a modal window titled "Device Info" is open, displaying details for the "Resin Workshop" device. The modal includes fields for Device Type (Resin Workshop), Connected Since (July 27, 2016), Last Data Transfer (August 1, 2016), Device ID (d18ec6...), Device Type ID (dt761d...), Name (Resin Workshop), and a "GENERATE DEVICE TOKEN..." button. At the bottom of the modal are "SAVE CHANGES" and "DELETE" buttons.



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!)

The screenshot shows the Artik Cloud Developers API console interface. At the top, there's a navigation bar with links for Overview, Tools & Resources, Documentation, Blog, Dashboard, and Artik Cloud (which is currently selected). Below the navigation is a search bar and a user profile icon for Ronald McCollam.

The main area displays the 'Users' endpoint documentation. It includes three methods: 'Get Current User Profile' (selected), 'Get User Devices', and 'Get User Device Types'. The 'Get Current User Profile' method is highlighted with a red circle around its 'GET' button. Below the documentation, a detailed call attempt is shown:

- Call:** `https://api.artik.cloud/v1.1/users/self`
- Request Headers:** `{ "Content-Type": "application/json", "Authorization": "Bearer b74fc3e[REDACTED]" }`
- Response Code:** `200`

A small yellow hexagonal logo is located in the bottom right corner of the page.

WORKSHOP



WORKSHOP PREP

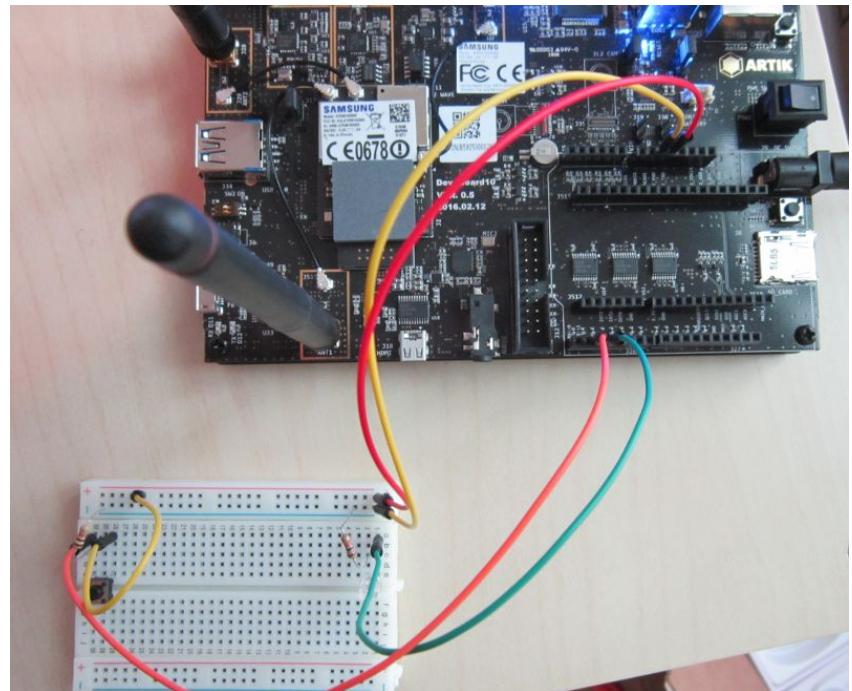
- Create account on resin.io
- Create an application
- Flash your device
- Clone my project

```
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

The screenshot shows the 'DEVICE ENVIRONMENT VARIABLES' section of a configuration interface. At the top, there's a 'New environment variable' section with a note: 'You can configure device-specific environment variables here. These variables can redefine (override) application environment variables of the same name.' Below this is a table for adding new variables, with columns for 'Name' and 'Value'. A blue 'ADD' button and a grey 'CLEAR' button are located to the right of the table. Further down, there's a table for 'APPLICATION ENVIRONMENT VARIABLES' containing two entries: 'ARTIKCLOUD_DEVICE_ID' with value 'd18ec64fce2344b192416960ad0e88c9' and 'ARTIKCLOUD_DEVICE_TOKEN' with value '9abde807ee454626b1ccebbf2032af91'. Each entry has a 'Redefine' button to its right. At the bottom, there's a section for 'OTHER DEVICE ENVIRONMENT VARIABLES'.

Name	Value	Actions
ARTIKCLOUD_DEVICE_ID	d18ec64fce2344b192416960ad0e88c9	Redefine
ARTIKCLOUD_DEVICE_TOKEN	9abde807ee454626b1ccebbf2032af91	Redefine



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!

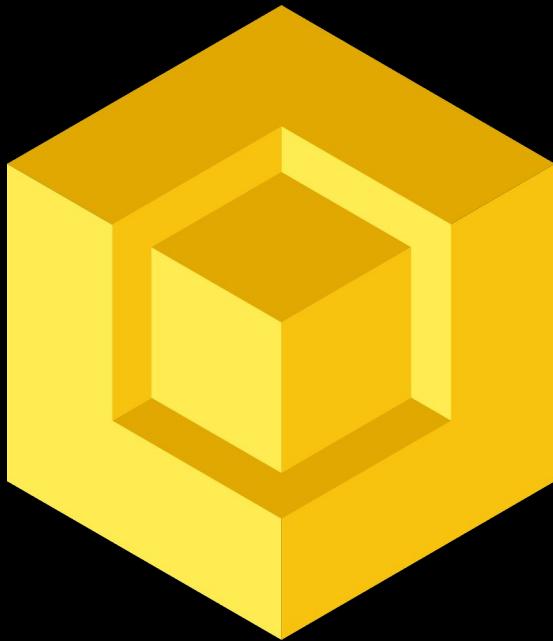


QUESTIONS?



THE FUTURE





Thank you!

Ronald McCollam
Solutions Architect

ronald@resin.io @RonaldMcCollam

APPENDIX



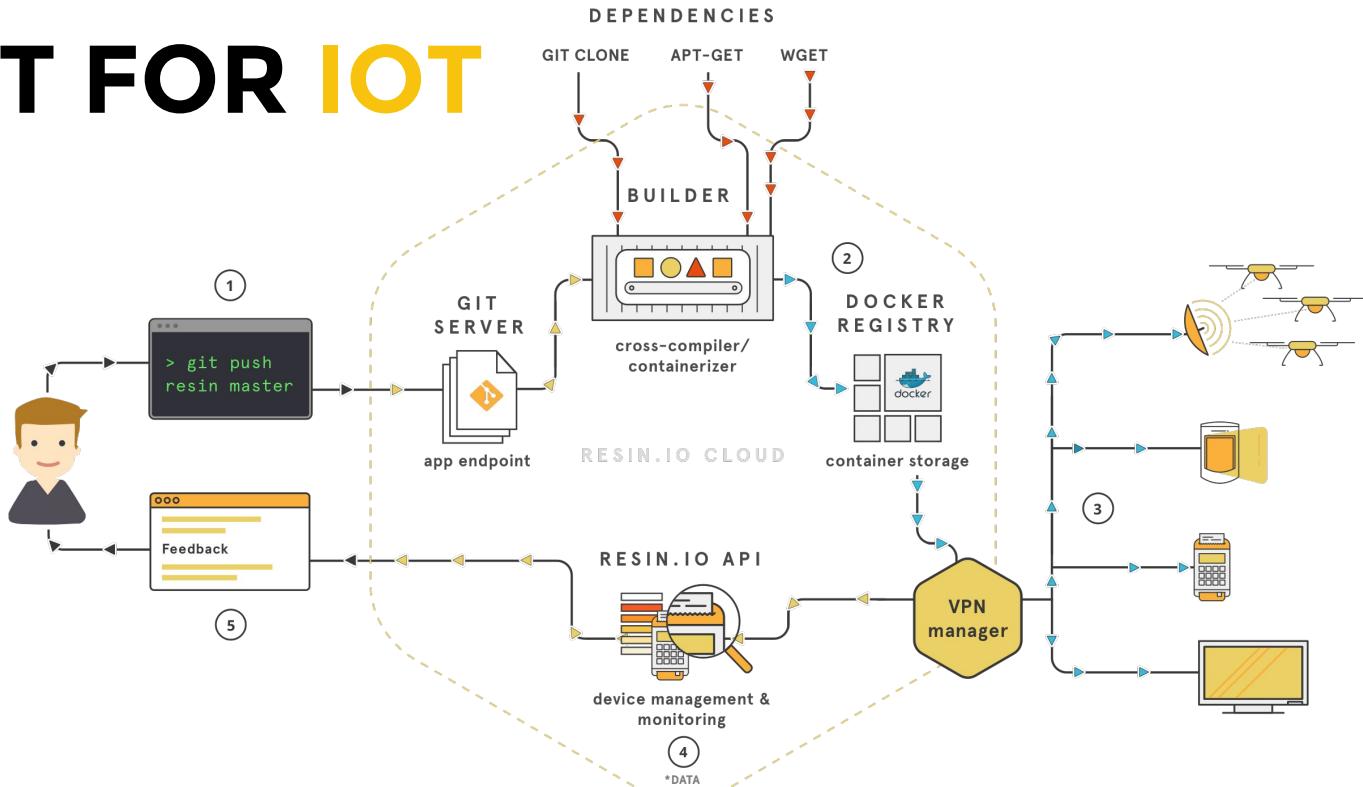
BASE IMAGES

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

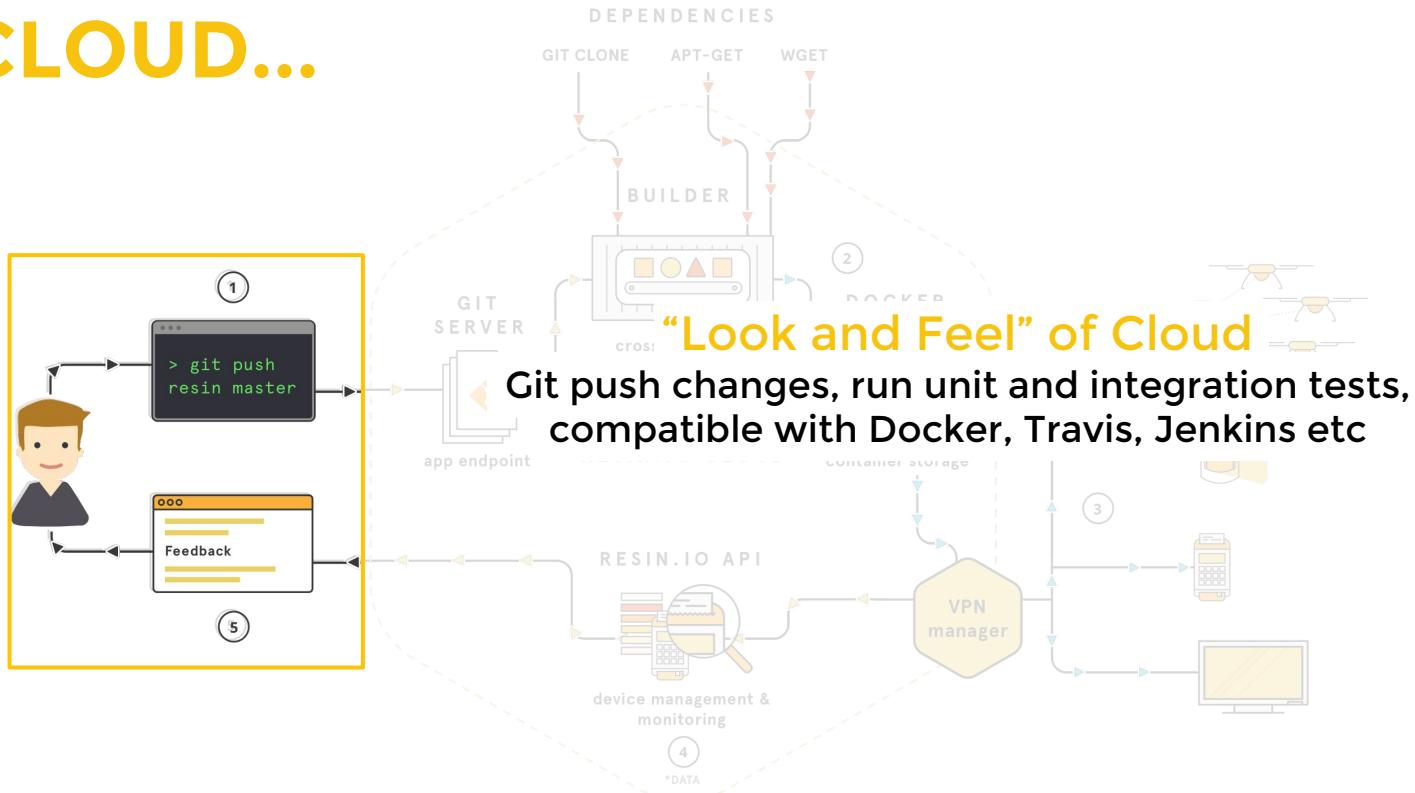
- Debian, Alpine, Fedora (coming soon)
- Python, Node.js, Go variants

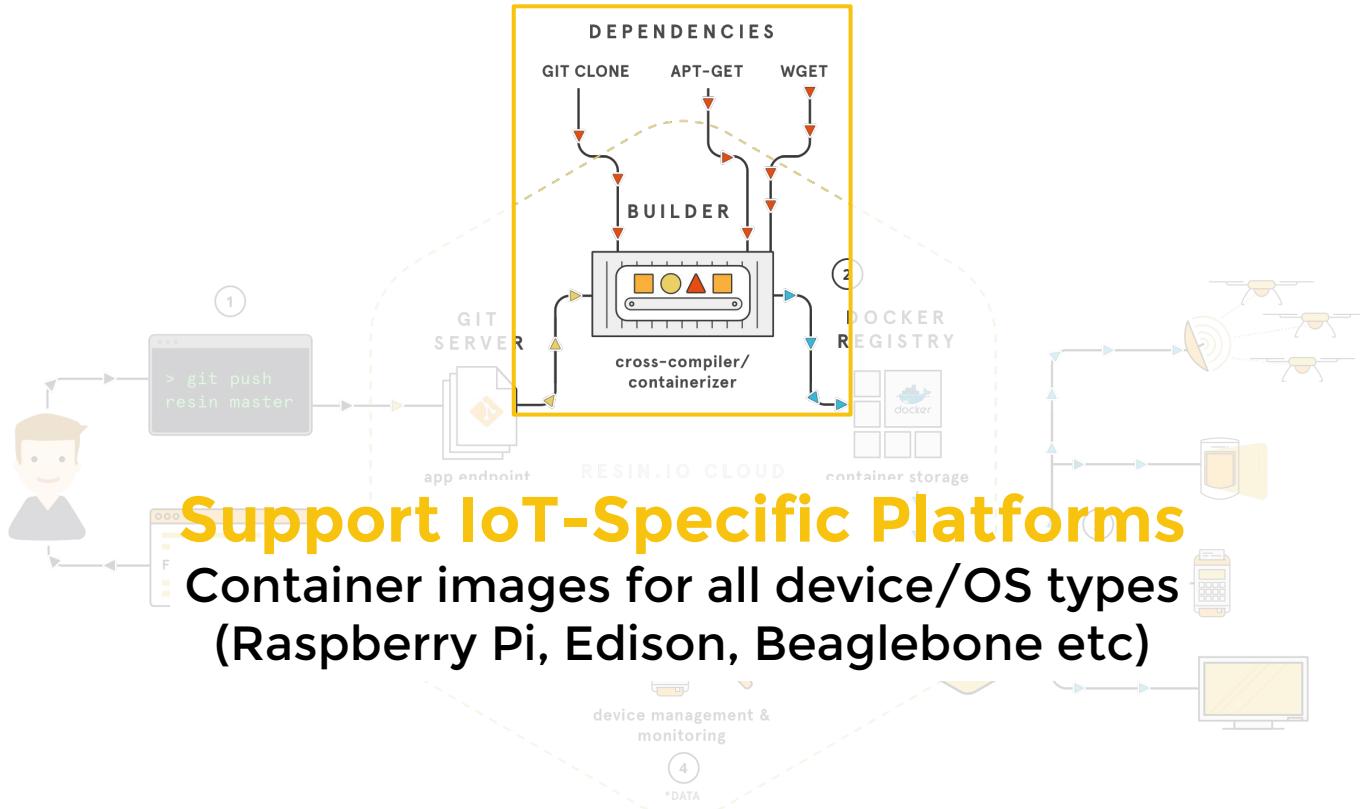


PURPOSE BUILT FOR IOT



BEST PRACTICES FROM THE CLOUD...





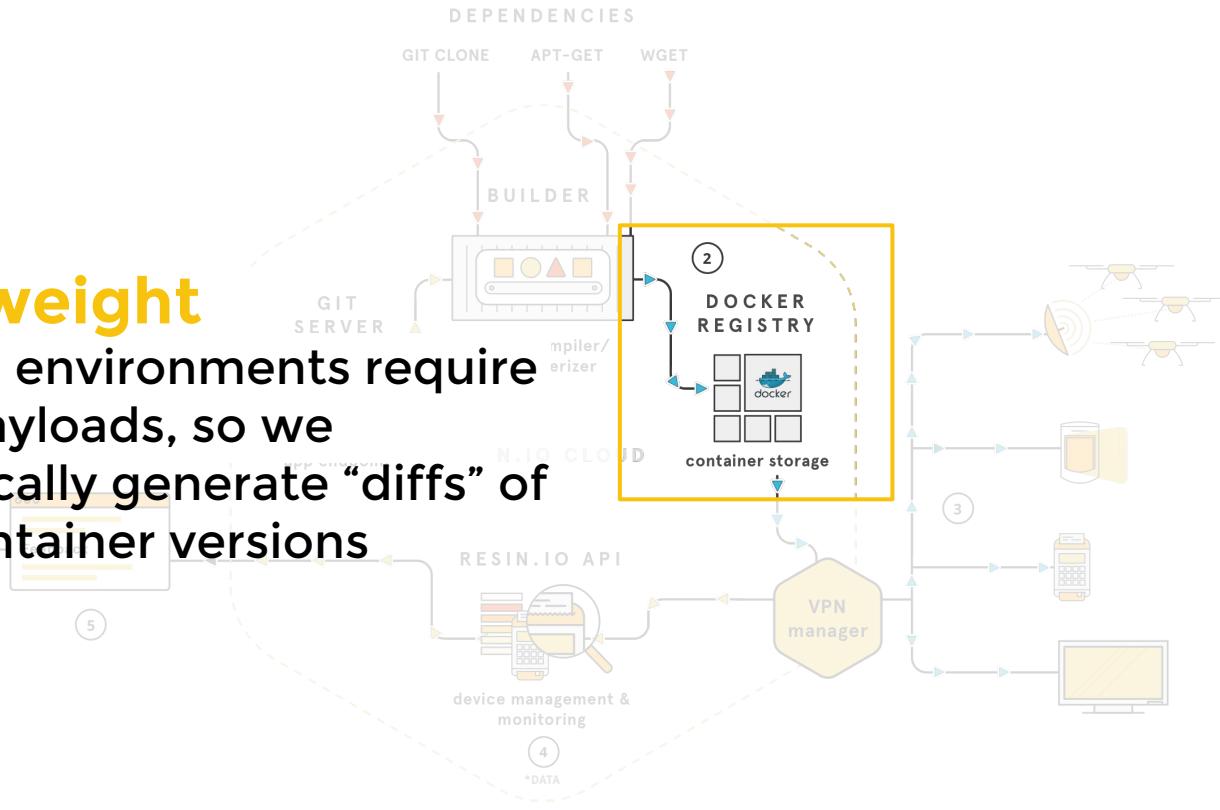
Support IoT-Specific Platforms

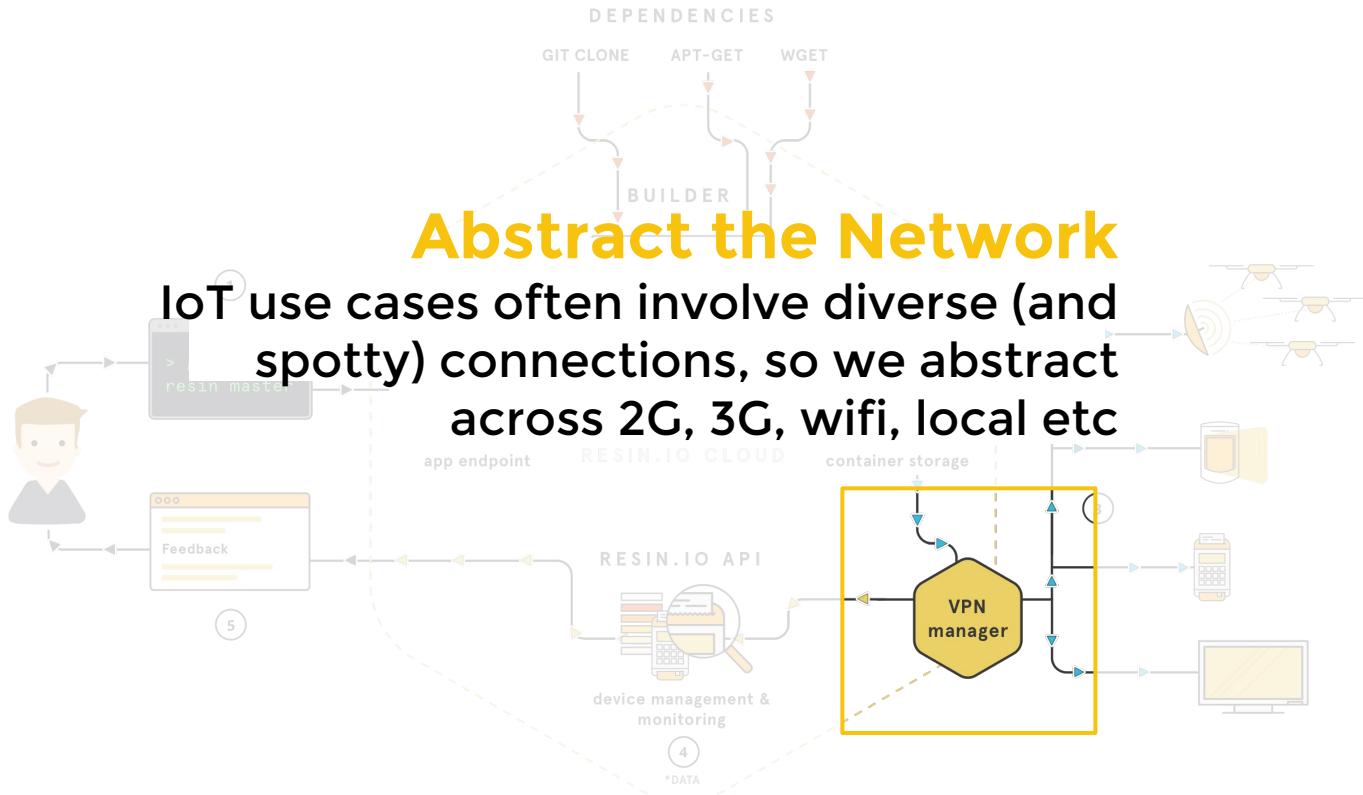
Container images for all device/OS types
(Raspberry Pi, Edison, Beaglebone etc)

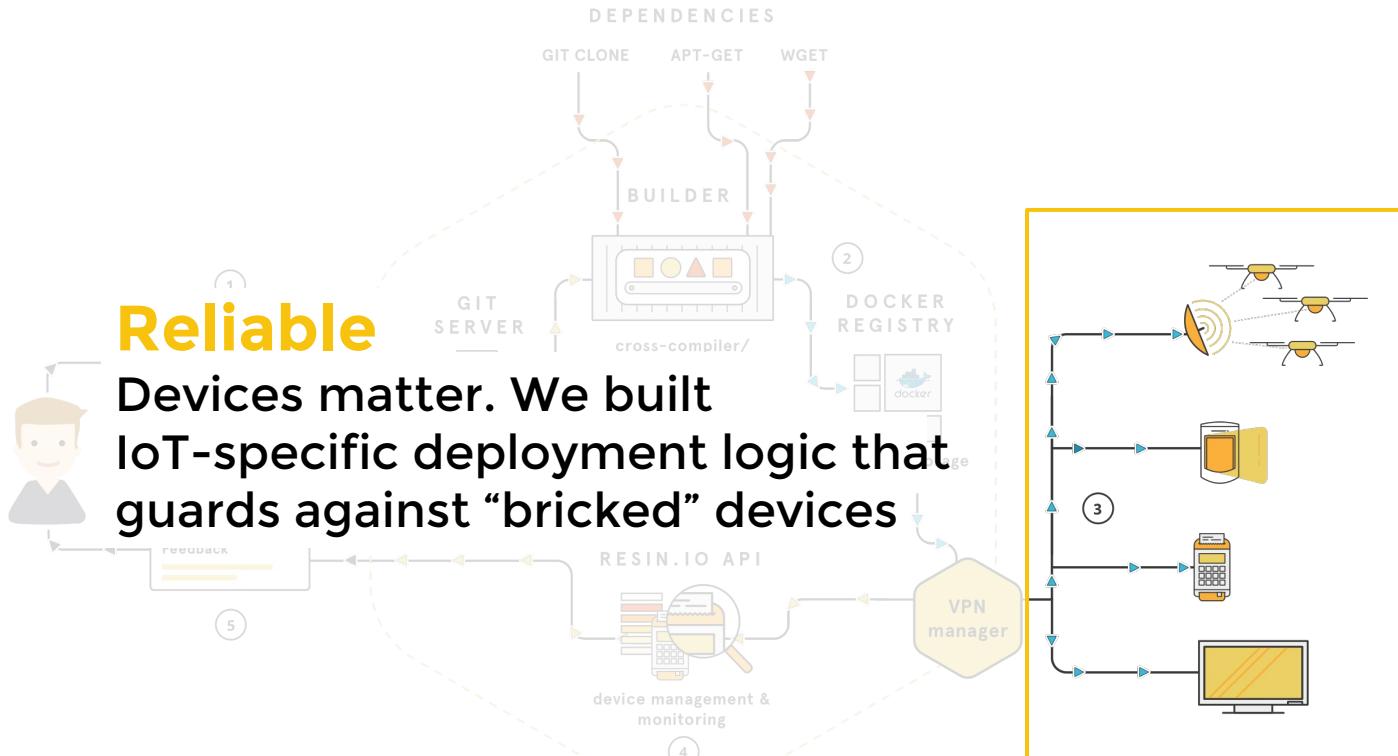


Lightweight

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





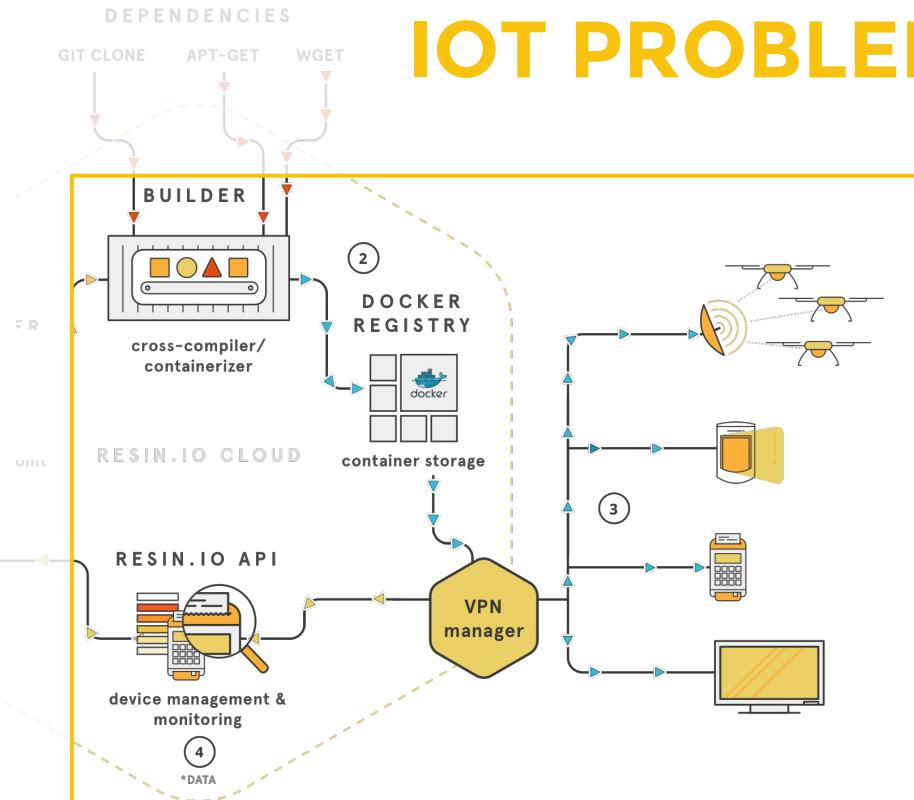


...SOLVING HARD IOT PROBLEMS

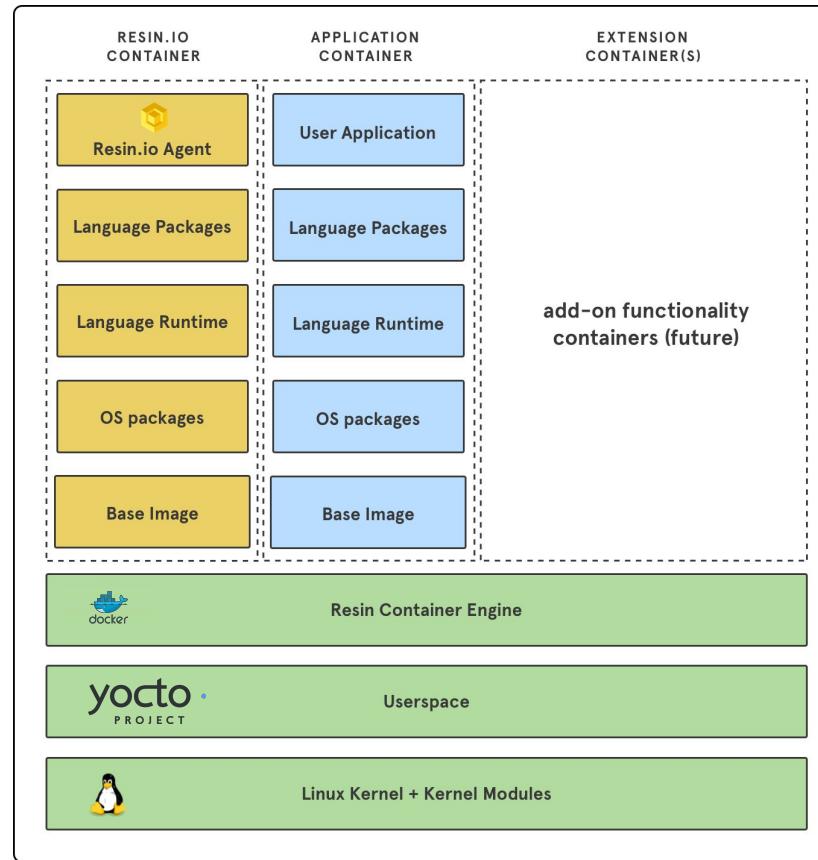
Architecture agnostic
IoT devices have diverse CPU architectures. Our builders compile for the target device architecture

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

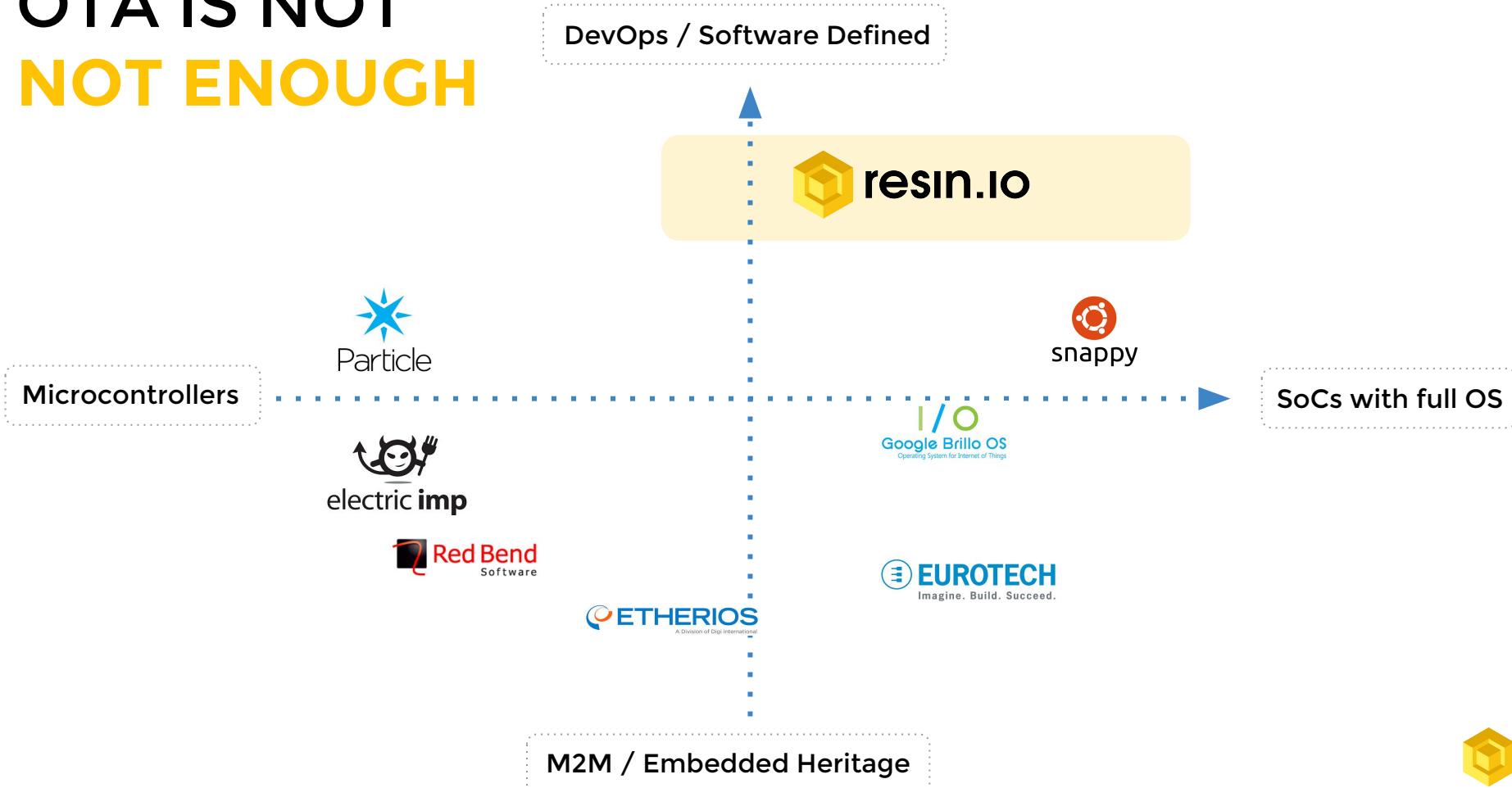
Fail-Safe Deployments
IoT-specific deployment logic that guards against “bricked” devices



ON-DEVICE S/W ARCHITECTURE

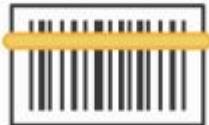


OTA IS NOT NOT ENOUGH



USE CASES ACROSS THE INDUSTRIAL INTERNET OF THINGS

Manufacturing & Logistics



Retail / Point of Service



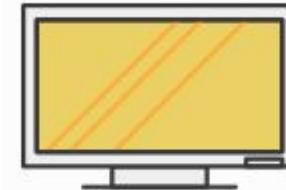
Smart Buildings



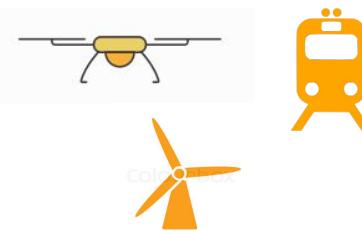
Industrial Lighting



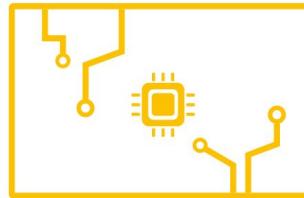
Digital Signage



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.



Burn. Better.

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

[DOWNLOAD FOR WINDOWS](#) ▾*v1.0.0-beta.10*