aws

COMMUNITY DAY

Serverless AWS IoT

Mischa Spiegelmock
JetBridge Inc.
Nov. 2020

aws

COMMUNITY DAY

About Me

- Berkeley, CA native
- Software Engineer
- AWS Solutions Architect (Assoc.)
- Design and engineer cloud-native applications
- Founder, JetBridge Inc.


```
root@tinkerboard:~# xinput list
⎡ Virtual core pointer                    id=2    [master pointer  (3)]
⎜   ↳ Virtual core XTEST pointer          id=4    [slave  pointer  (2)]
⎜   ↳ fts_ts                              id=6    [slave  pointer  (2)]
⎜   ↳ Generic USB Audio                   id=7    [slave  pointer  (2)]
⎜   ↳ DaKai 2.4G RX                       id=11   [slave  pointer  (2)]
⎣ Virtual core keyboard                   id=3    [master keyboard (2)]
    ↳ Virtual core XTEST keyboard         id=5    [slave  keyboard (3)]
    ↳ gpio-keys                           id=8    [slave  keyboard (3)]
    ↳ Generic USB Audio                   id=9    [slave  keyboard (3)]
    ↳ DaKai 2.4G RX                       id=10   [slave  keyboard (3)]
    ↳ DaKai 2.4G RX                       id=12   [slave  keyboard (3)]
root@tinkerboard:~#
root@tinkerboard:~# xrandr
Screen 0: minimum 320 x 200, current 800 x 480, maximum 8192 x 8192
DSI-1 connected primary 800x480+0+0 (normal left inverted right x axis y axis) 0
mm x 0mm
   800x480       58.76*
HDMI-1 disconnected (normal left inverted right x axis y axis)
root@tinkerboard:~#
```

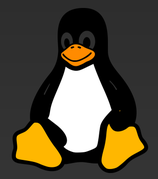
Serverless + IoT?

Technologies

- AWS **IoT Core** - Thing registry, MQTT

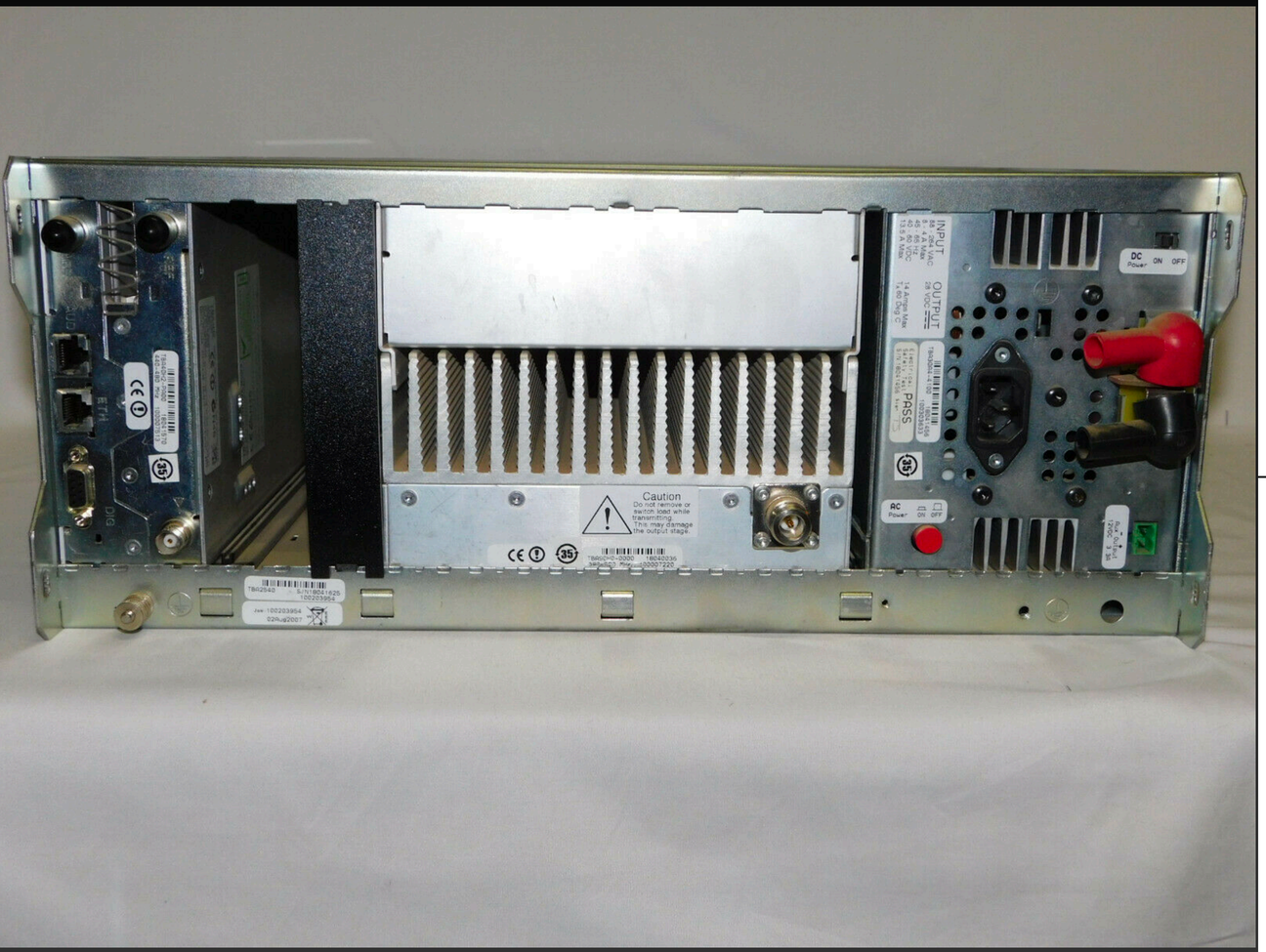
- AWS **IoT Greengrass** - deploy + run code on device

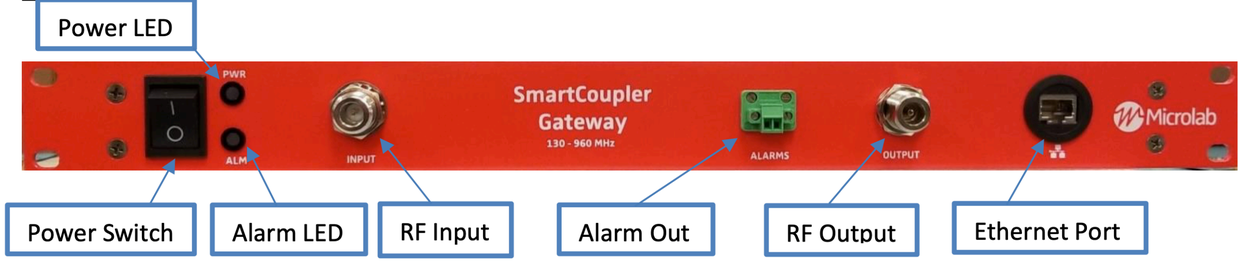
- AWS **Lambda** - define functions



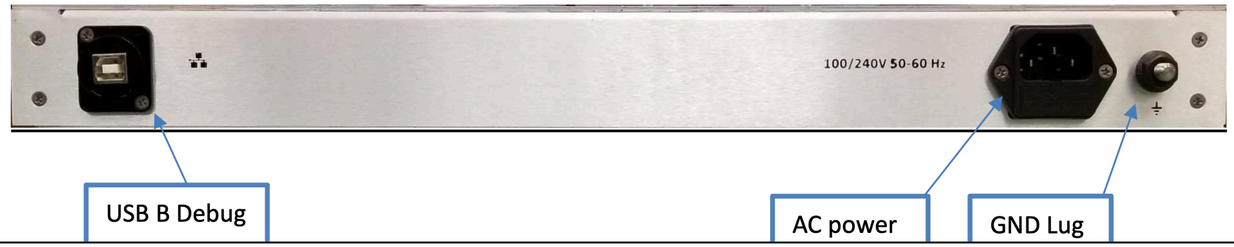
- **Linux**-based devices

System: Remote Monitoring


Caution
Do not remove or switch load while transmitting. This may damage the output stage.
INPUT
OUTPUT
DC Power
ON OFF
AC Power

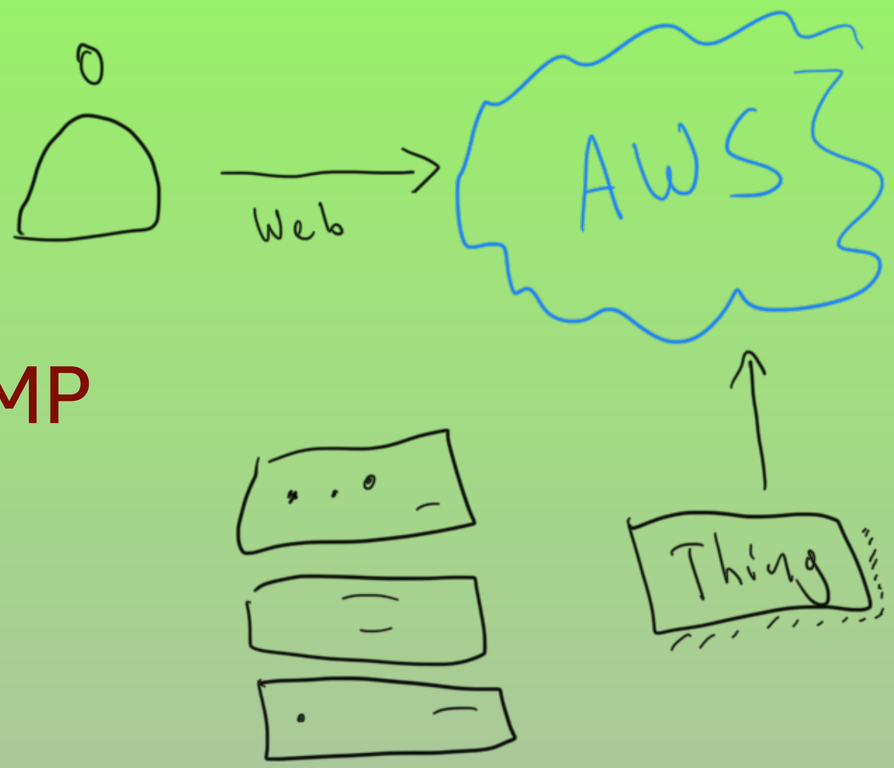

Power LED
PWR
ALM
INPUT
SmartCoupler Gateway
130 - 960 MHz
ALARMS
OUTPUT
Microlab
Power Switch
Alarm LED
RF Input
Alarm Out
RF Output
Ethernet Port

Rear (AC)

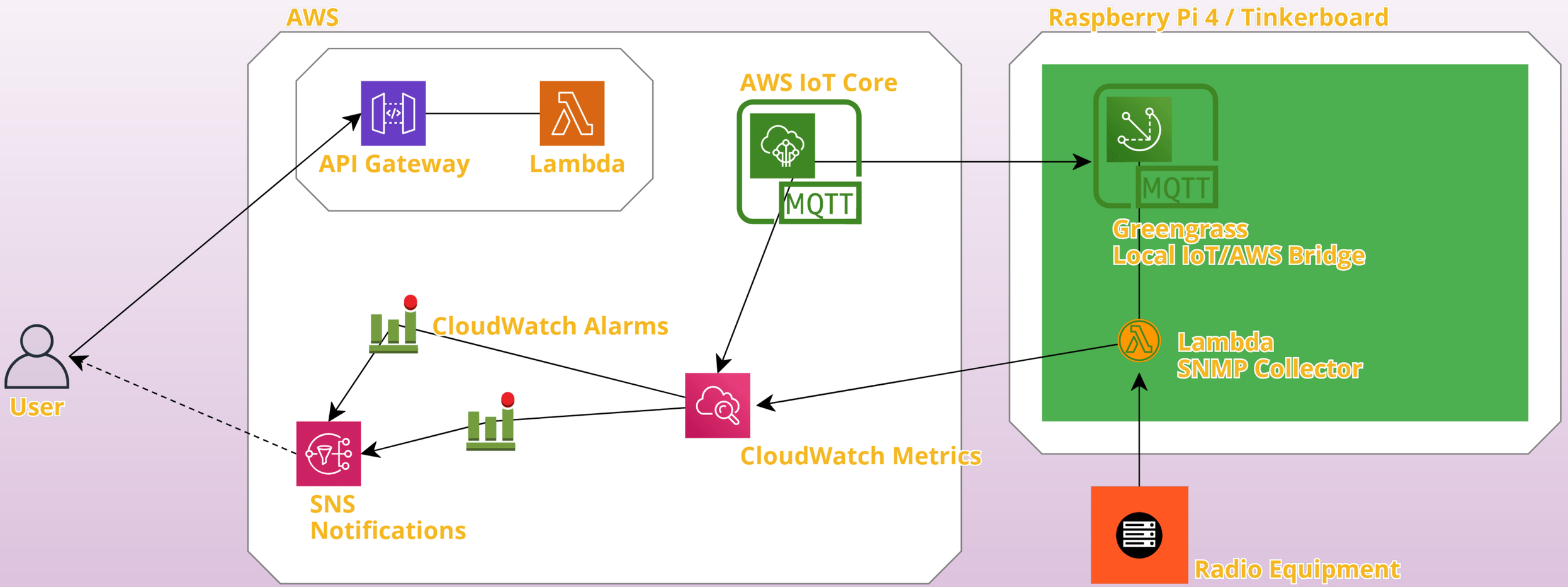

100/240V 50-60 Hz
USB B Debug
AC power
GND Lug


TB9100
tait

Deliverables



- **Monitor** radio hardware on-site via SNMP
  - (RFC1067 and friends)
- **Report** telemetry and faults to cloud
- **View** telemetry and faults in a web interface


AWS
API Gateway
Lambda
AWS IoT Core
MQTT
CloudWatch Alarms
User
SNS
Notifications
CloudWatch Metrics
Raspberry Pi 4 / Tinkerboard
MQTT
Greengrass
Local IoT/AWS Bridge
Lambda
SNMP Collector
Radio Equipment

Challenges

- Hardware
- OS image
- Provisioning
- Deployments / Updates
- Monitoring
- Fix stuff remotely
- GUI

```
function setup_greengrass {
    adduser  --system --uid 1200 ggc_user
    groupadd --system --gid 1200 ggc_group

    # initscript
    cat <<EOF >/etc/init.d/S02greengrass
#!/bin/sh
cd /greengrass/ggc/core
./greengrassd \$@
EOF
    chmod 755 /etc/init.d/S02greengrass

    # systemd unit file
    cat <<EOF >/etc/systemd/system/greengrass.service
[Unit]
Description=AWS Greengrass Service
After=network.target

[Service]
Type=simple
ExecStart=/etc/init.d/S02greengrass start
Restart=always
RestartSec=120
User=root
PIDFile=/run/greengrassd.pid

[Install]
WantedBy=multi-user.target
EOF
    # start GG at startup
    systemctl enable greengrass.service

    # install greengrass application
    tar Czxf / /tmp/overlay/greengrass-linux-armv7l-${GG_VERSION}.tar.gz

    # add root CA
    mkdir -p /greengrass/certs
    cp /tmp/overlay/root.ca.pem /greengrass/certs/
}

function setup_ssh {
    systemctl enable ssh

    mkdir -p /root/.ssh
    chmod 700 /root/.ssh
```

Hardware

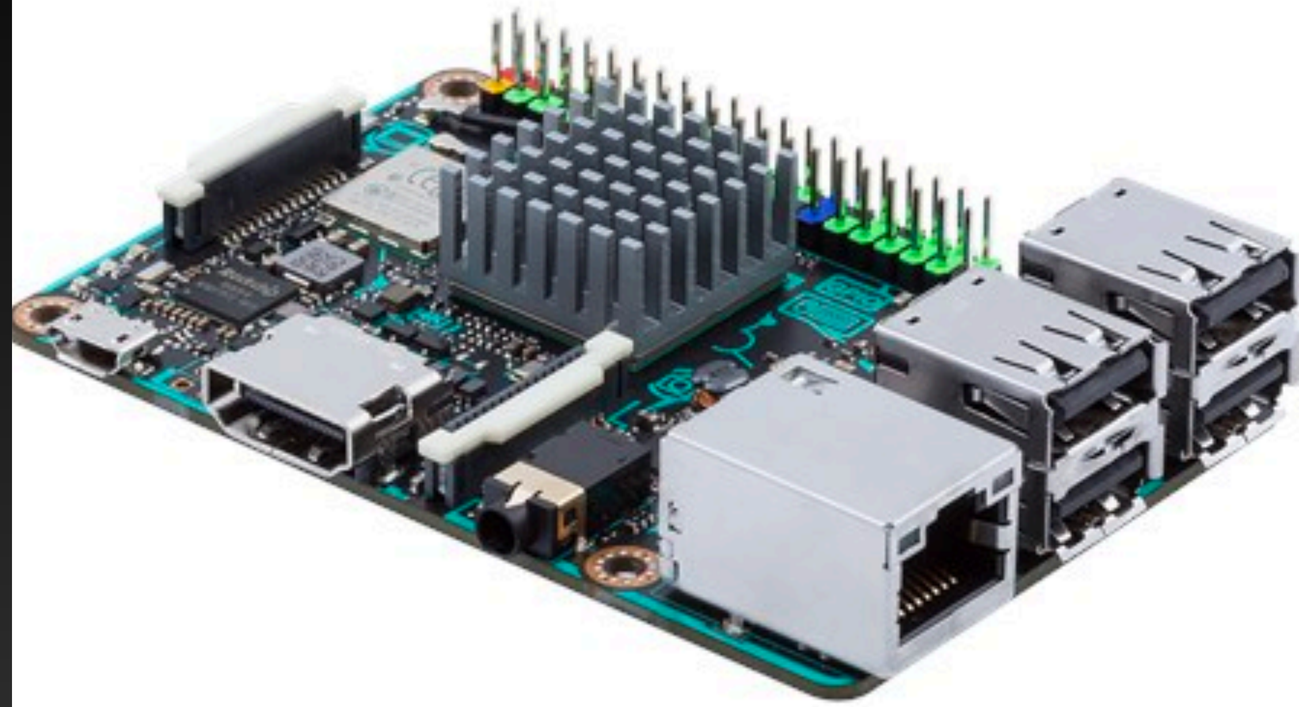
IoT “Thing”

- Raspberry Pi
- Asus Tinkerboard S

- AWS Greengrass Client

- Debian OS image

- Provisioning



Raspberry Pi

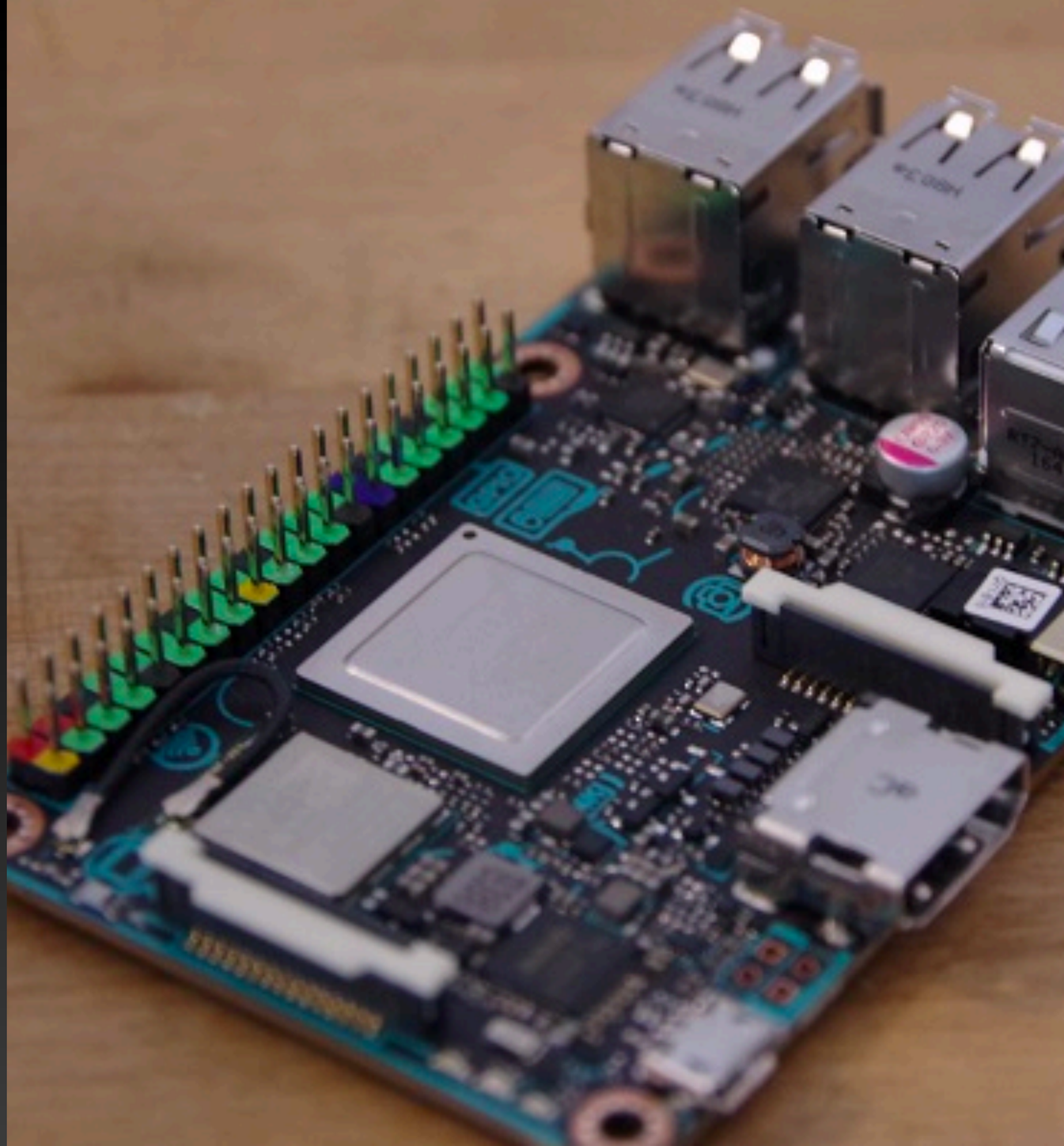
👍 Well-supported

👎 No internal flash

Asus Tinkerboard S

👍 Internal flash

👎 No pi-gen

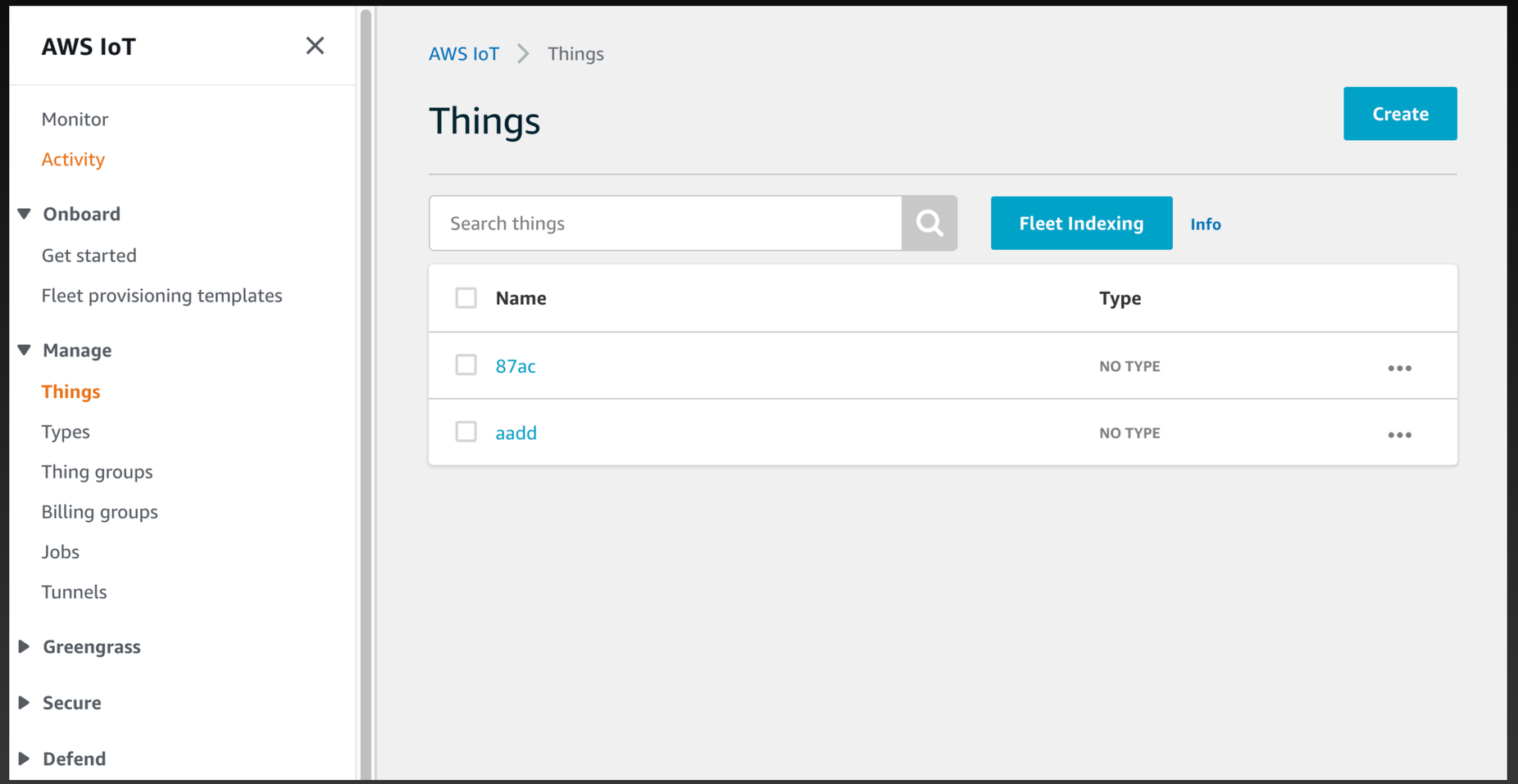


Pi-gen
Armbian
meta-aws
Yocto
Buildroot
...

OS Image

- Automated build (shell)
- Configure
  - Networking
  - Xorg
  - SSH
  - OS packages
- Greengrass

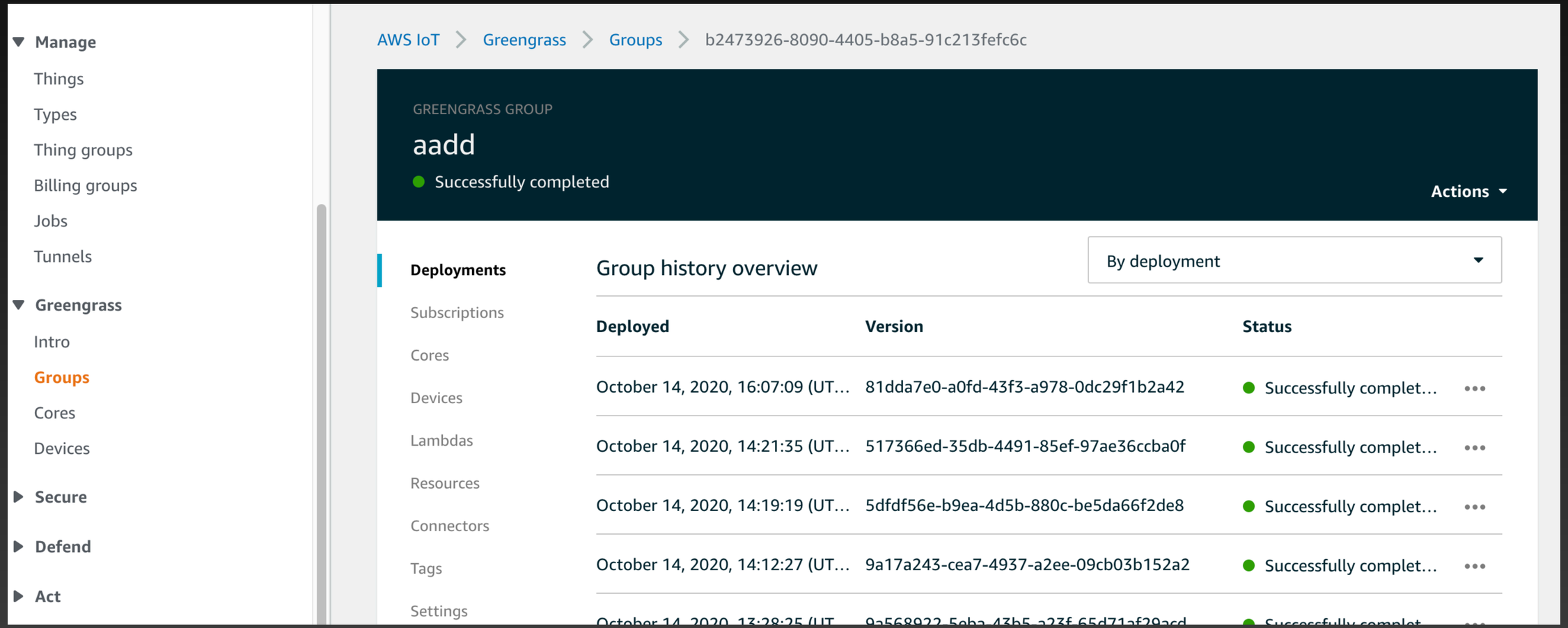
AWS IoT Core


AWS IoT
Monitor
Activity
Onboard
Get started
Fleet provisioning templates
Manage
Things
Types
Thing groups
Billing groups
Jobs
Tunnels
Greengrass
Secure
Defend
AWS IoT > Things
Things
Create
Search things
Fleet Indexing
Info
Name
Type
87ac
NO TYPE
aadd
NO TYPE

IoT Core

- “Thing” registry
- Authentication
  - Certificate - mutual TLS
  - Authorization - IAM policy document
- MQTT - messaging
- Shadow - fancy configuration
- Tunnels - remote administration

AWS Greengrass



Greengrass

"Run Lambdas On My Device"

- Client-side gateway ("core")
- MQTT client/server
- Lambda runner
- Code deployment client
- Subscriptions - Thing ↔ AWS

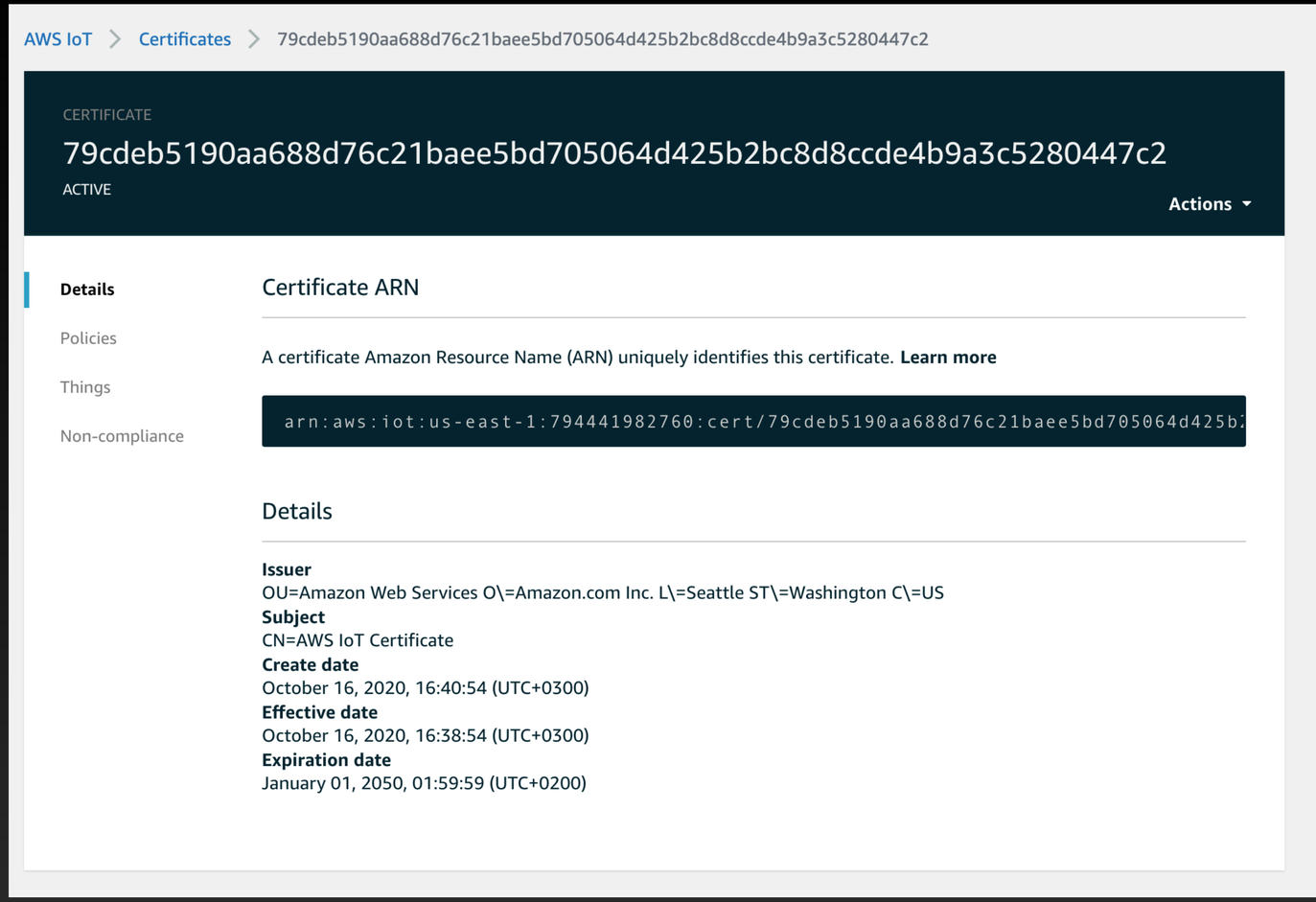
Greengrass
Configuration

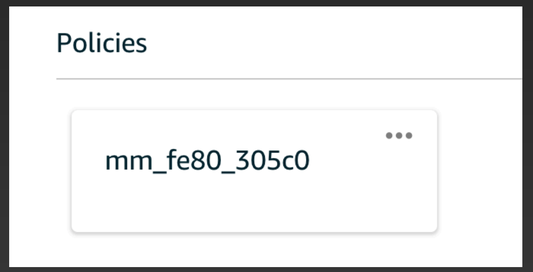
- Config
  - Root CA (Amazon)
  - Private key
  - Certificate
- Core Thing
  - Thing ARN
  - AWS endpoint

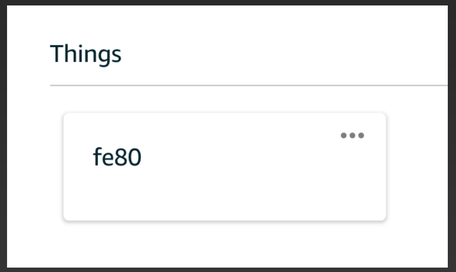
```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive" : 600,
    "ggDaemonPort": 8000,
    "systemComponentAuthTimeout": 5000
  },
  "runtime" : {
    "maxWorkItemCount" : 1024,
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

AWS IoT Authentication
X.509 MTLS

- Certificate provisioned:
  - Client Key
  - Client Certificate
  - Policy
  - Thing Attachment







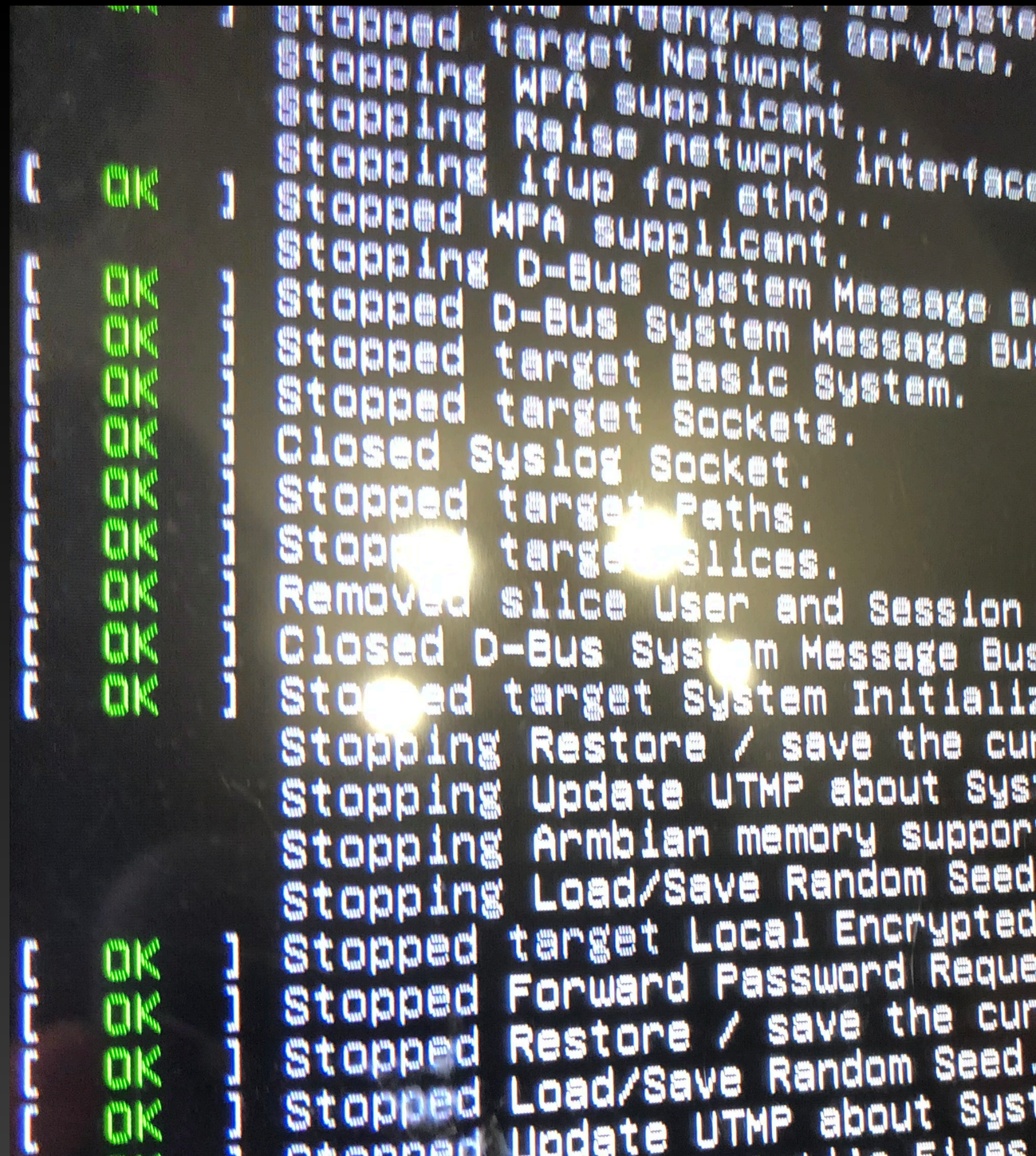
Provisioning
Connecting a Thing

- Create Certificates
- Create Thing
- Create Group Version
- Create Group
- Generate Greengrass config
- Create Deployment

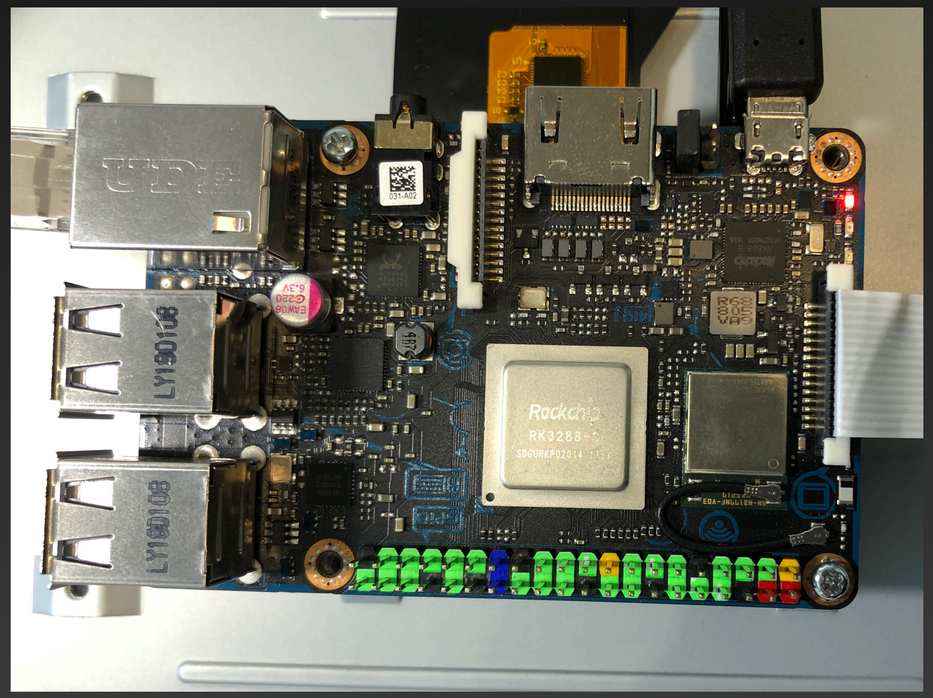
```
certs = self.create_certs()
thing = self.create_thing()
self.attach_thing_policy(certs=certs)

# create initial Group version
group_initial_version = self.generate_group_version(
    thing_arn=thing["thingArn"], certificate_arn=certs["certificateArn"]
)

# create Group
group = self.create_group(initial_version=group_initial_version)

# set thing shadow
initial_shadow = {}
if self.remote_addr:
    # save IP of who requested the provisioning
    initial_shadow["provision_addr"] = self.remote_addr
self.update_thing_shadow(initial_shadow)

# link thing to group - for our use later
iot.update_thing(
    thingName=self.name,
    attributePayload={
        "attributes": {THING_GROUP_ID_ATTR: group["Id"]},
        "merge": True,
    },
)

# perform initial deployment
self.create_deployment(
    thing_name=thing["thingName"],
    group=group,
    group_version_id=group["LatestVersion"],
)

config_doc = self.gen_config_doc(thing=thing, group=group)
return {
    "thing": thing,
    "group": group,
    "certificatePem": certs["certificatePem"],
    "publicKey": certs["keyPair"]["PublicKey"],
    "privateKey": certs["keyPair"]["PrivateKey"],
    "ggConfig": config_doc,
}
```

Install Greengrass

- .tar.gz
- Debian APT
- Docker
- Snap



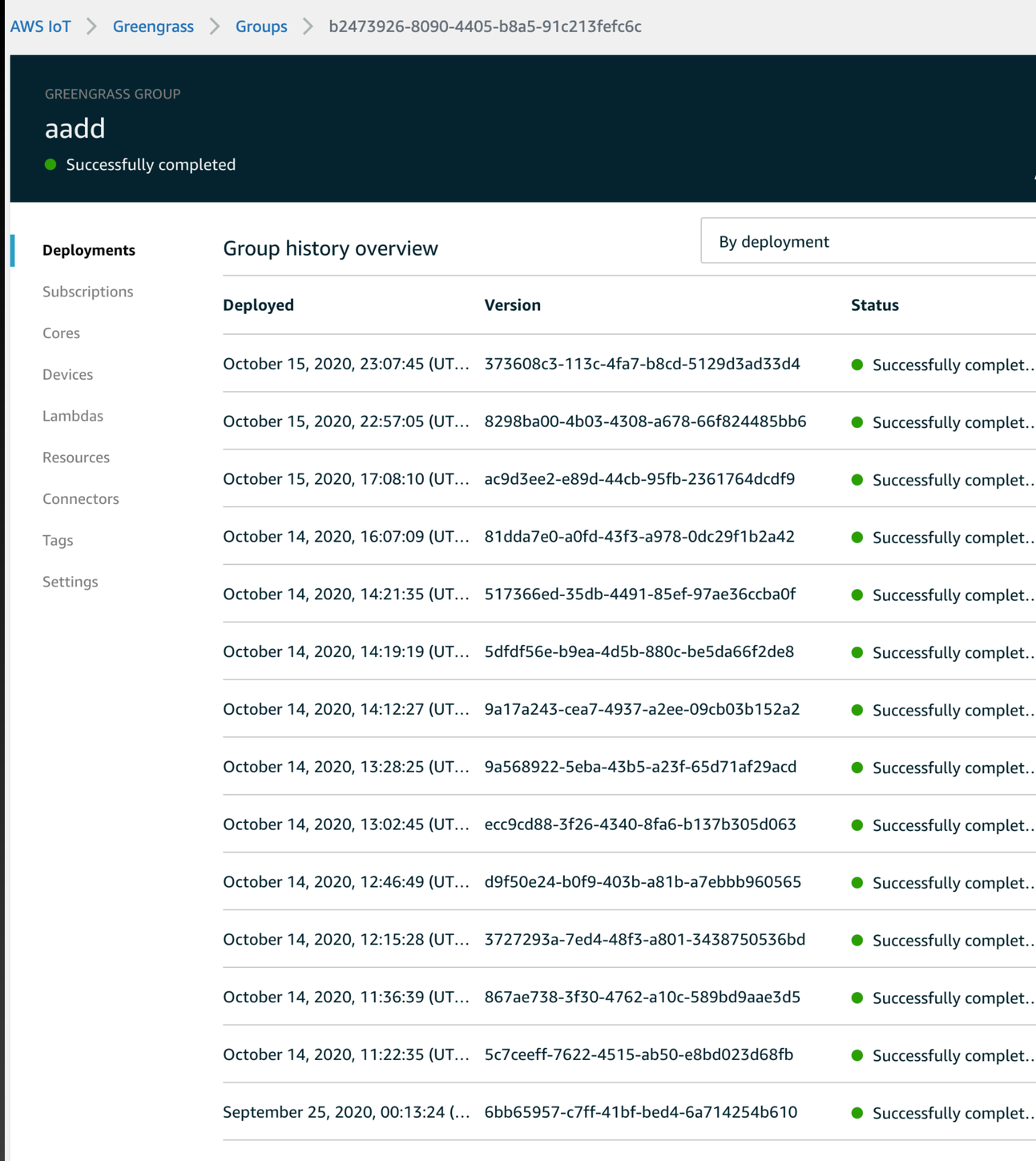
Now What?



Greengrass
Deployments

- CreateDeployment
  - GroupVersionId
- CreateGroupVersion

```
{
  "GroupId": "string",
  "CoreDefinitionVersionArn": "string",
  "DeviceDefinitionVersionArn": "string",
  "FunctionDefinitionVersionArn": "string",
  "SubscriptionDefinitionVersionArn": "string",
  "LoggerDefinitionVersionArn": "string",
  "ResourceDefinitionVersionArn": "string",
  "ConnectorDefinitionVersionArn": "string",
  "AmznClientToken": "string"
}
```



Function Definition

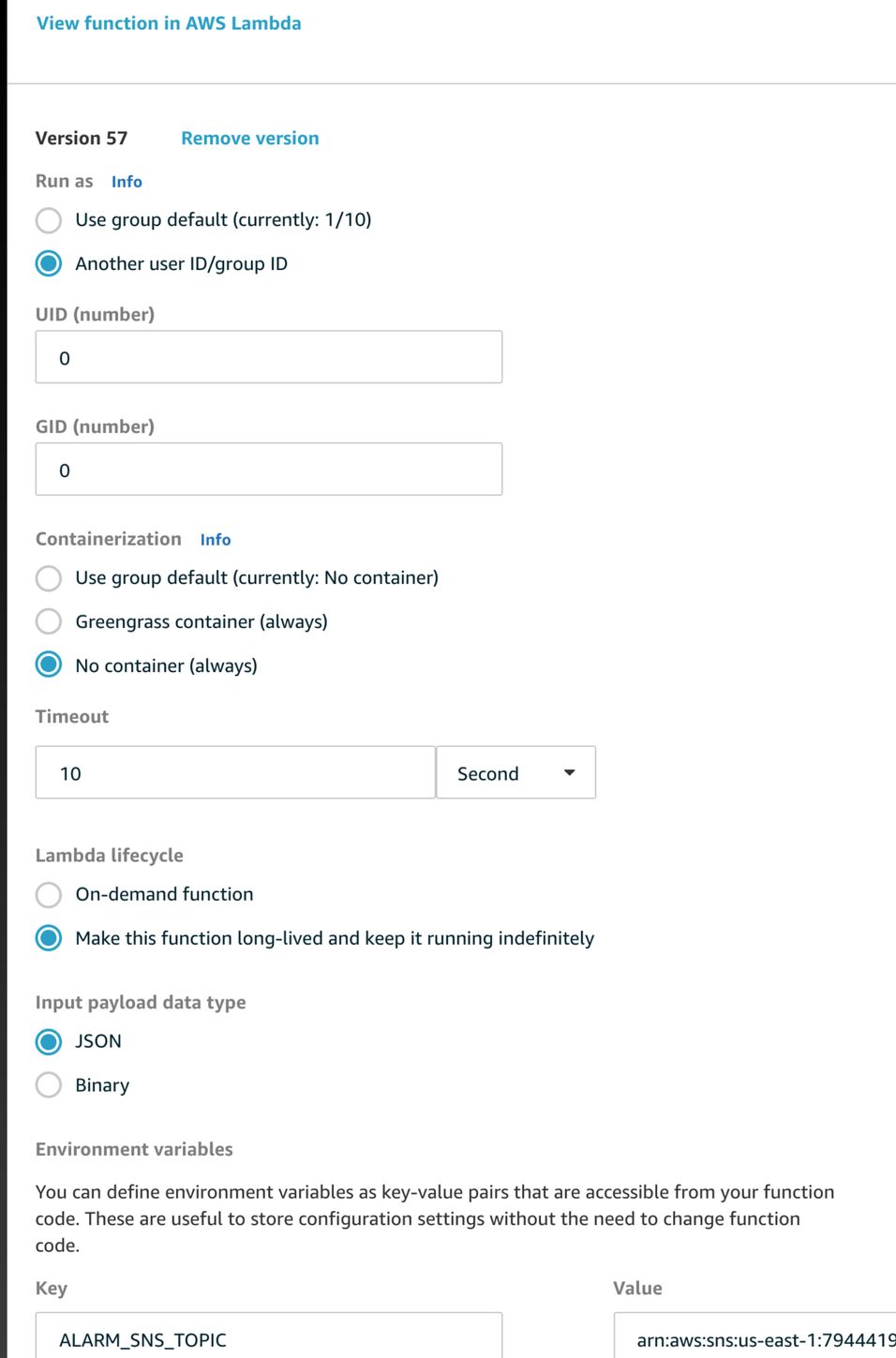
Lambdas on Things

- Put Lambdas on your Thing
- Special features:
  - UNIX UserID / GroupID
  - Run as root
  - Run in docker
  - Run forever

Function Definition

Lambdas on Things

- Put Lambdas on your Thing
- Special features:
  - UNIX UserID / GroupID
  - Run as root
  - Run in docker
  - Run forever



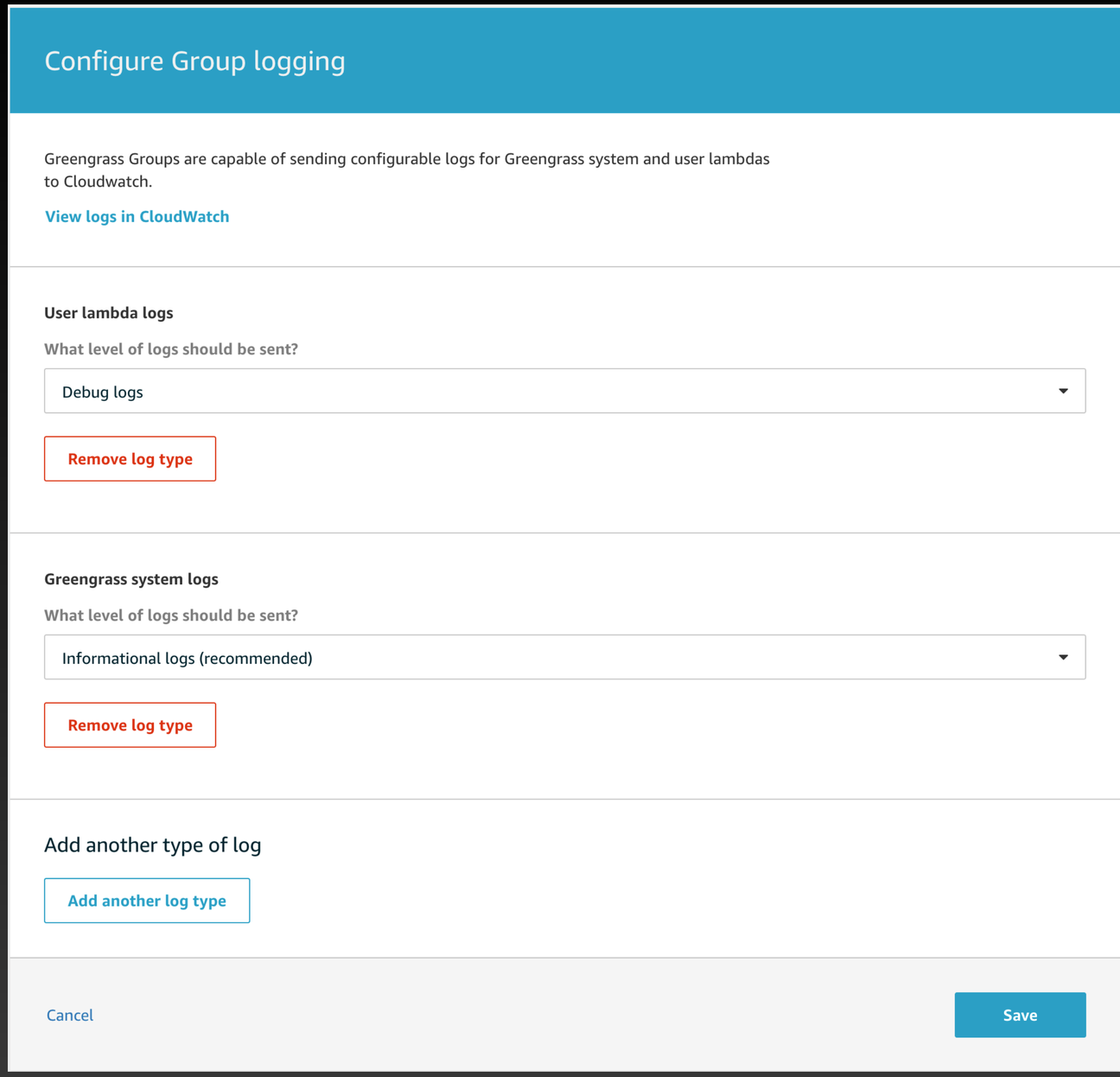
Greengrass
Other goodies

- OTA updates
- Machine learning inference
- Data stream management (→ Kinesis, IoT Analytics, S3)
- Secrets (with Secrets Manager)
- Connectors (MQTT-driven functionality)

Monitoring

Logging

- Direct to CloudWatch:
  - Lambdas
  - Greengrass System



Logging

- CloudWatch Logs subscriptions

```
logger: # handler for logs generated by monitor functions + greengrass runtime
  handler: handler.logger_entry
  memorySize: 128
  timeout: 3
  events:
    # these will need to be commented out at first deploy
    # there need to be logs that already exist for these subscriptions to be created
    - cloudwatchLog: "/aws/greengrass/GreengrassSystem/runtime"
    - cloudwatchLog: "/aws/greengrass/GreengrassSystem/GGConnManager"
    - cloudwatchLog: "/aws/greengrass/Lambda/#{AWS::Region}/#{AWS::AccountId}/${{self:custom.stackName}}-monitor"
    - cloudwatchLog: "/aws/greengrass/Lambda/#{AWS::Region}/#{AWS::AccountId}/${{self:custom.stackName}}-tunnel"
    - cloudwatchLog: "/aws/greengrass/Lambda/#{AWS::Region}/#{AWS::AccountId}/${{self:custom.stackName}}-reconfigure"
    - cloudwatchLog: "/aws/greengrass/Lambda/#{AWS::Region}/#{AWS::AccountId}/${{self:custom.stackName}}-system"
```

👀 **metromon** APP 19:12

```
[INFO]-ipc_client.py:202,Got work item with invocation id [e0a63a44-b59f-42d2-7b9d-ed6ab5e560cb]
[INFO]-handler_system.py:21,Rebooting! Hope it comes back up!
```

```
[INFO]-lambda_runtime.py:364,Caught signal 15. Stopping runtime.
```

```
[INFO]-ipc_client.py:223,Posting work result for invocation id [e0a63a44-b59f-42d2-7b9d-ed6ab5e560cb] to
http://localhost:8000/2016-11-01/functions/arn:aws:lambda:us-east-1:79444                 ·prod-system:58/work
[INFO]-ipc_client.py:232,Posted work result for invocation id [e0a63a44-b59f-42d2-7b9d-ed6ab5e560cb]
[INFO]-ipc_client.py:192,Getting work for function [arn:aws:lambda:us-east-1:79444                 ·prod-system:58] from
http://localhost:8000/2016-11-01/functions/arn:aws:lambda:us-east-1:794441                 ›rod-system:58/work
[INFO]-lambda_runtime.py:364,Caught signal 15. Stopping runtime.
```

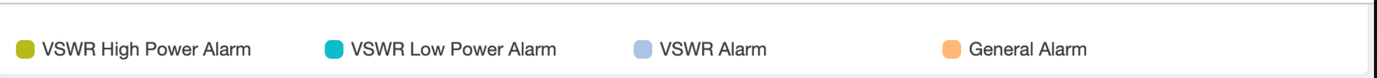
```
[INFO]-server.py:72,IPC connection closed code = 1006 (connection closed abnormally [internal]), no reason
[INFO]-Stopping...
[INFO]-Stopping...
[INFO]-Stopping...
[INFO]-Stopping...
[WARN]-monitor.py:216,Collector thread finished
```

```
Client connected to MQTT gateway.
Client: aadd
IP: 100.2.
Timestamp: 2020-10-26 17:12:45.780000
```

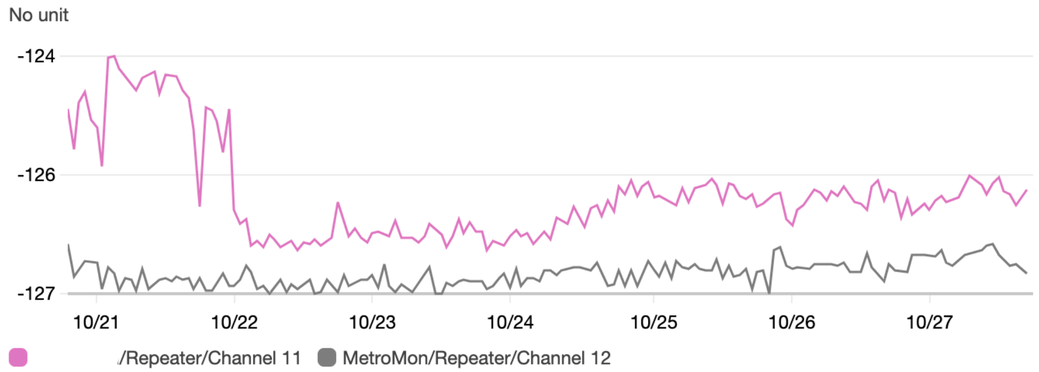
Logging To Slack

New

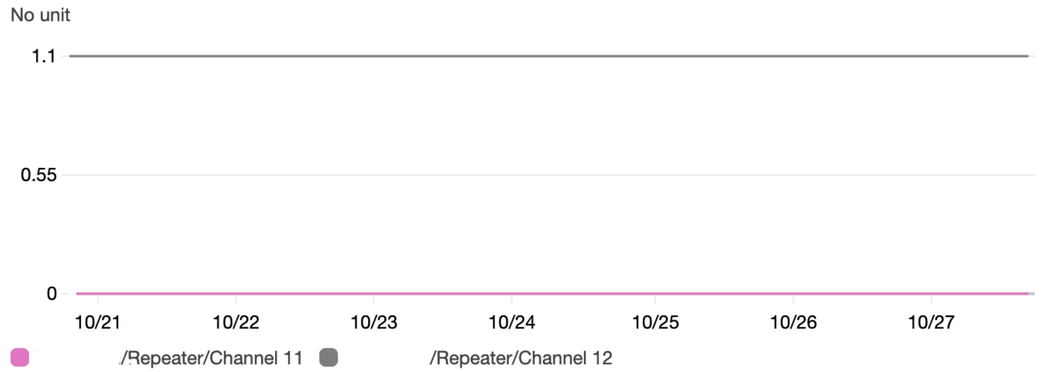
CloudWatch
Metrics



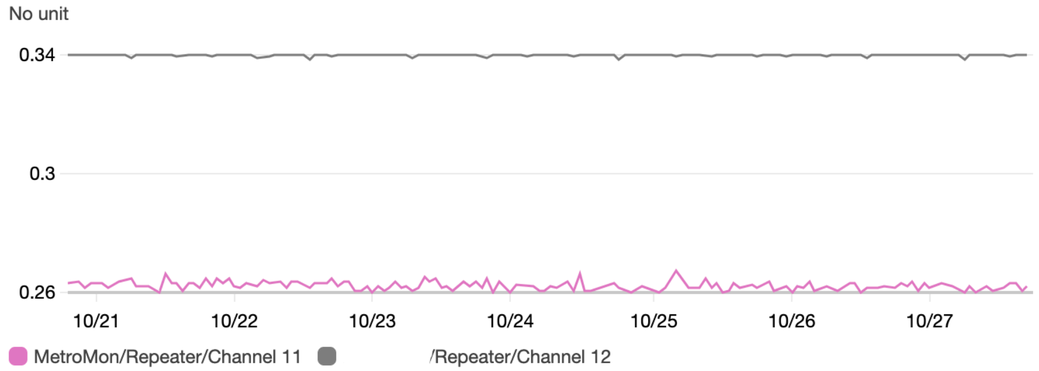
Repeater RX RSSI



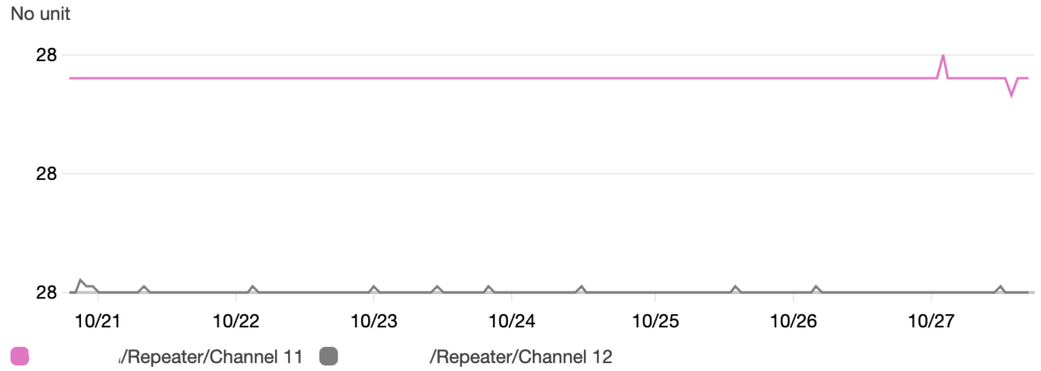
Repeater TX VSWR



PMU Out Amperage



PMU Out Voltage



Repeater Main/Battery Power State

Battery In Voltage

Secure Tunnel

```python
from os import path, getenv
from subprocess import Popen
import logging

log = logging.getLogger(__name__)


def start_tunnel_entry(event, context):
    """Entry point for initiating secure tunnels.

    https://docs.aws.amazon.com/iot/latest/developerguide/secure-tunneli
    """
    token = event["clientAccessToken"]
    services = event["services"]
    region = event["region"]

    thing_name = getenv("AWS_IOT_THING_NAME", "UNKNOWN")

    log.info(f"Starting tunnel for thing {thing_name} {services} in {reg

    # start up localproxy in background
    env = dict(AWSIOT_TUNNEL_ACCESS_TOKEN=token)
    args = [get_exe_path(), "-d", "localhost:22", "-r", region]
    Popen(args, env=env)


def get_exe_path() -> str:
    """Get absolute path to localproxy executable."""
    filepath = path.abspath(__file__)
    return path.join(path.dirname(filepath), "tunnel", "localproxy")
```

Subscriptions

CREATE A SUBSCRIPTION

Confirm and save your Subscription

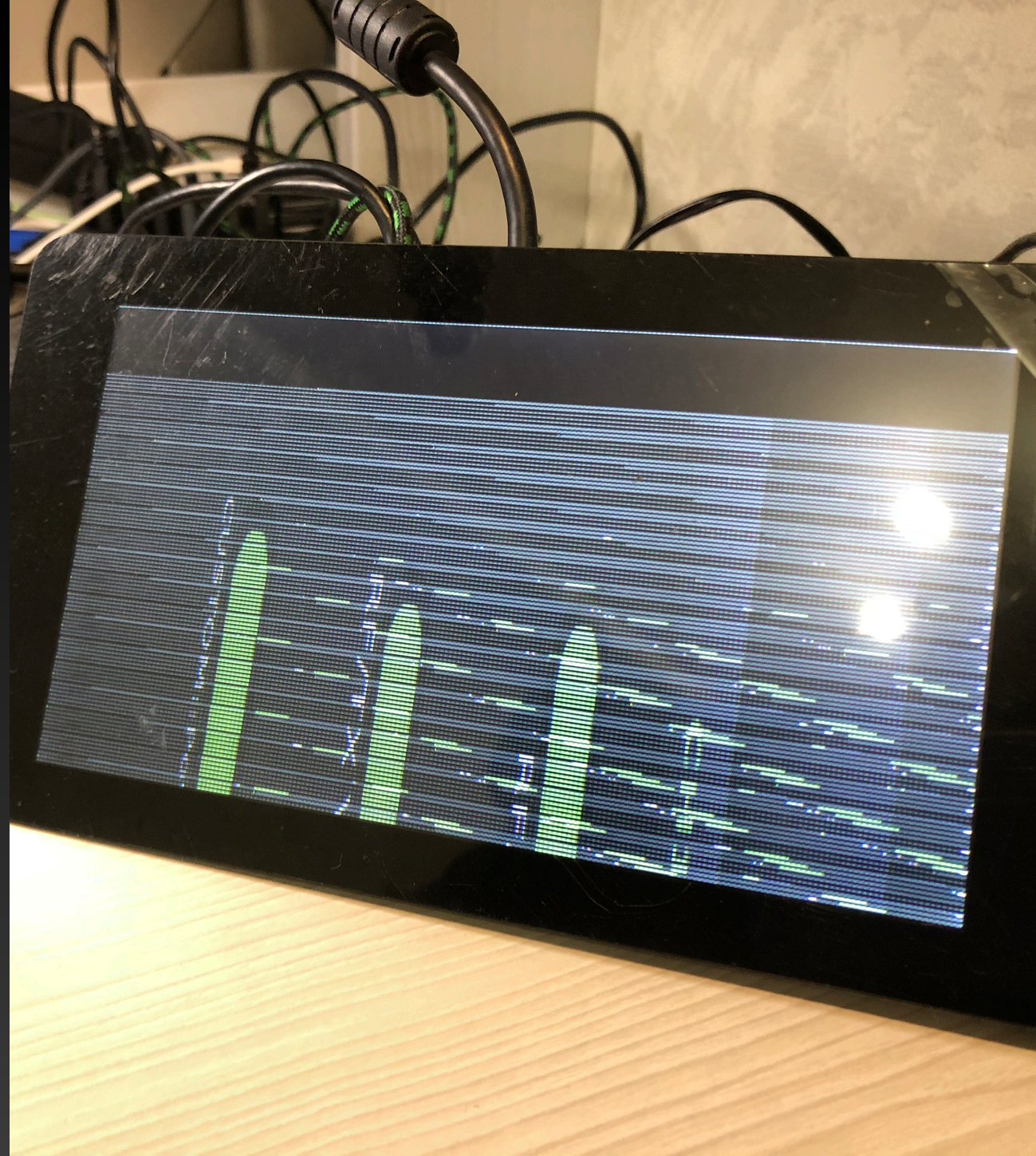
Your Subscription is complete and your objects are connected in this Group. You can now save, and then deploy your new Group definition to have this change take effect.

IoT Cloud	SERVICE	
hello/world/counter/trigger		
Greengrass_HelloWorld_Counter	LAMBDA	

Back Finish

Xorg
Lambdas With GUIs?

- Sort of!

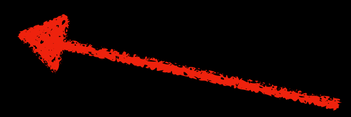


Xorg
Lambdas With GUIs?

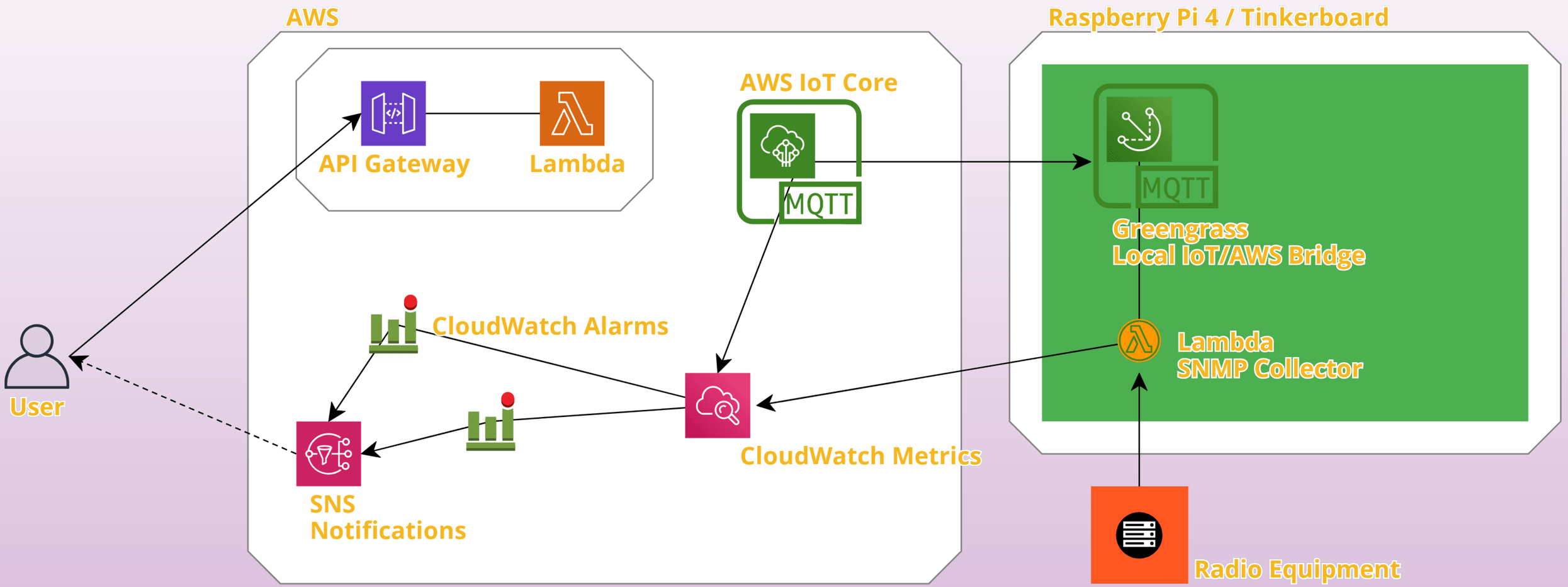
- Sort of!
- GUI wrapper
  - Import lambda code on DM start
- Long-lived DM restart lambda

```
def restart_display_manager():
    restart_gui = ["/etc/init.d/lxdm", "restart"]
    subprocess.run(restart_gui)
```

```python
from pathlib import Path


def find_lambda_system_module() -> Optional[str]:
    """Return import path to GUI code, if it exists."""
    # this is where our system code would be deployed to
    files = Path("/greengrass/ggc/deployment/lambda").glob(
        "arn.aws.lambda.*.*.function.myapp-*-system.*"
    )
    matches = list(files)
    log.info(f"Found deployments: {matches}")
    if not matches:
        log.info("Didn't find system function deployed")
        return None

    if len(matches) > 1:
        log.info("Warning: found multiple deployments of system code")

    return str(matches.pop().absolute())
```



```
def display_lambda_gui(system_module_path: str):
    sys.path.append(system_module_path)
    try:
        from device_gui.main import main as lambda_gui_main

        os.environ["MYAPP_DISPLAY_WRAPPED"] = "true"
        lambda_gui_main()
    except Exception as ex:
        log.exception(ex)
        display_default_gui(traceback.format_exc())
```


AWS
API Gateway
Lambda
AWS IoT Core
MQTT
CloudWatch Alarms
User
SNS
Notifications
CloudWatch Metrics
Raspberry Pi 4 / Tinkerboard
MQTT
Greengrass
Local IoT/AWS Bridge
Lambda
SNMP Collector
Radio Equipment

aws

COMMUNITY DAY

Thanks!

Mischa Spiegelmock
Founder, JetBridge Inc.
me@mish.dev