amazon web services | Partner Network

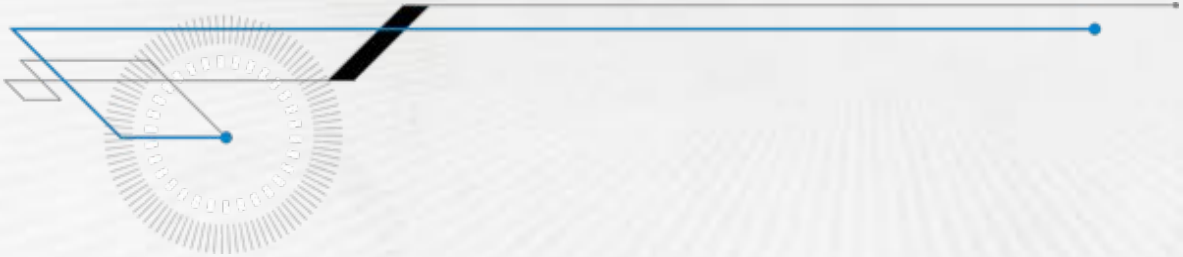
AWS IoT, Greengrass Building Blocks and Workshop

August 8th 2017

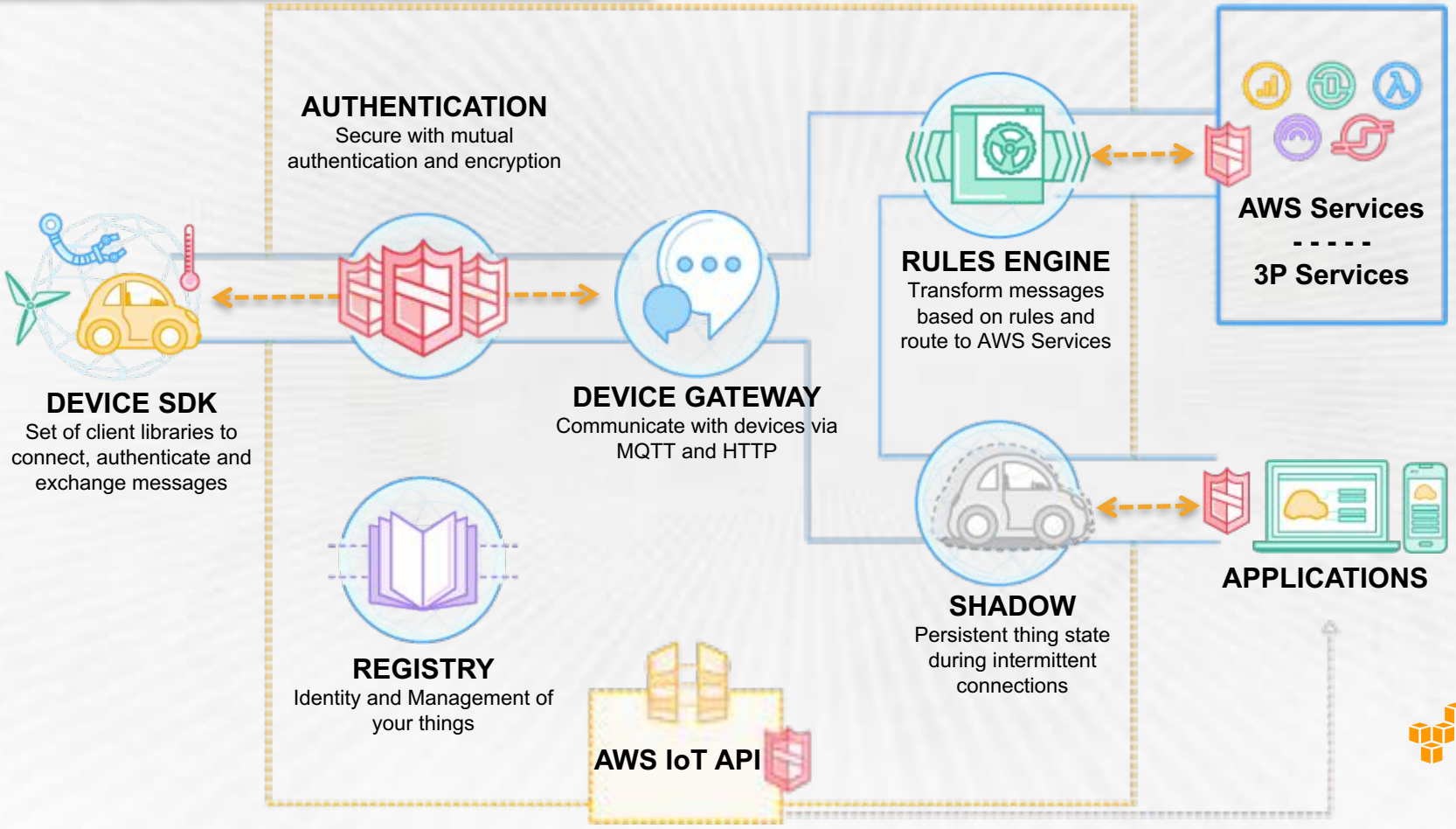
Craig Williams, Partner Solutions Architect & IoT Specialist

amazon web services

AWS IoT Deep Dive



AWS IoT Overview



What does AWS IoT data look like?

PUBLISH turbines/ev-gen/123 (qos: 0)

```
{
  "timestamp": "2016-11-29T10:00:00",
  "temperature": 125,
  "humidity": 95,
  ”rotor-freq": 6455,
  “output”: 480,
  “output-freq”: 60
}
```

amazon web services

What does it cost?

- $5 per million messages
- Messages are 512 bytes
- Non-subscribed shadow messages aren’t billed
- PING messages are billed

Publish on/off to the sprinkler

```
SUBSCRIBE
  macdonald/sprinkler-456
```

Sprinkler

Device Gateway

Control logic

amazon web services

Publish on/off to the sprinkler?

```
PUBLISH
  macdonald/sprinkler-456
  { "water": "on" }
```

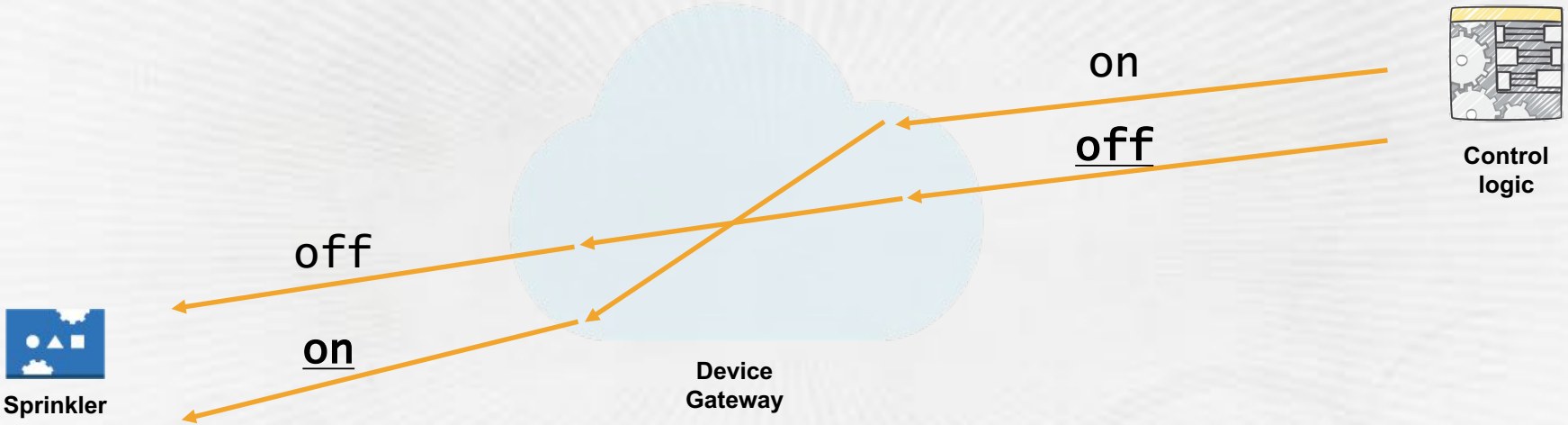
Sprinkler

Device Gateway

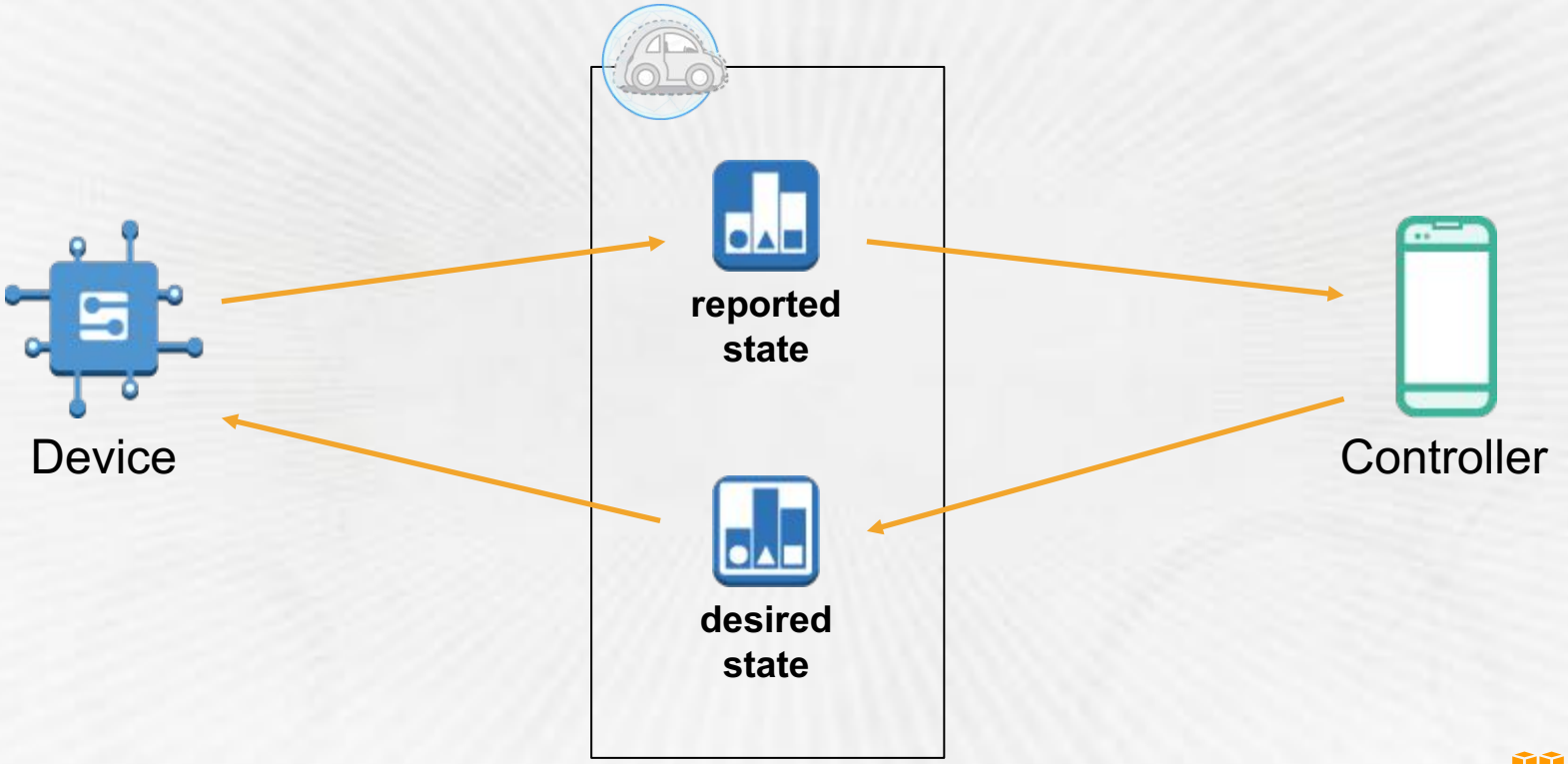
Control logic

amazon web services

Direct publishing: why not?



Device Shadows



amazon web services

AWS IoT Shadow

Thing

Report its current state to one or multiple shadows
Retrieve its desired state from shadow

Shadow

Shadow reports delta, desired and reported states along with metadata and version

Mobile App

Set the desired state of a device
Get the last reported state of the device
Delete the shadow

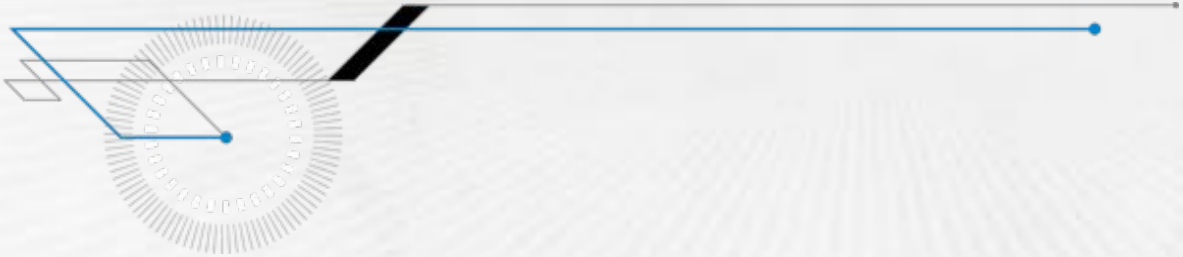
```
{
"state" : {
        “desired" : {
            "lights": { "color": "RED" },
            "engine" : "ON"
        },
        "reported" : {
            "lights" : { "color": "GREEN"  },
        "engine" : "ON"
        },
        "delta" : {
            "lights" : { "color": "RED"  }
         } },
"version" : 10
}
```

amazon web services

Last Will and Testament

- CONNECT message parts:
- Protocol: MQTT 3.1.1
- ClientId: abc
- Username: (not supported)
- Password: (not supported)
- CleanSession: (not supported)
- KeepAlive: 60 seconds
- LastWill PUBLISH message:
- Topic: foo/bar
- QoS: 1
- Payload: {"foo": "bar"}

Greengrass Deep Dive



green·grass

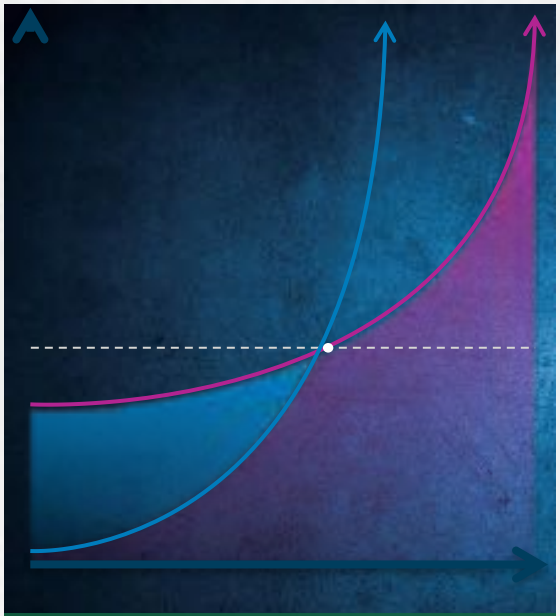
noun

noun: **greengrass**; plural noun: **greengrass core**

a set of locally-deployable software distributions that let devices operate and communicate locally while retaining benefits of analytics and other high-level services in the cloud

Why Greengrass?

Law of physics



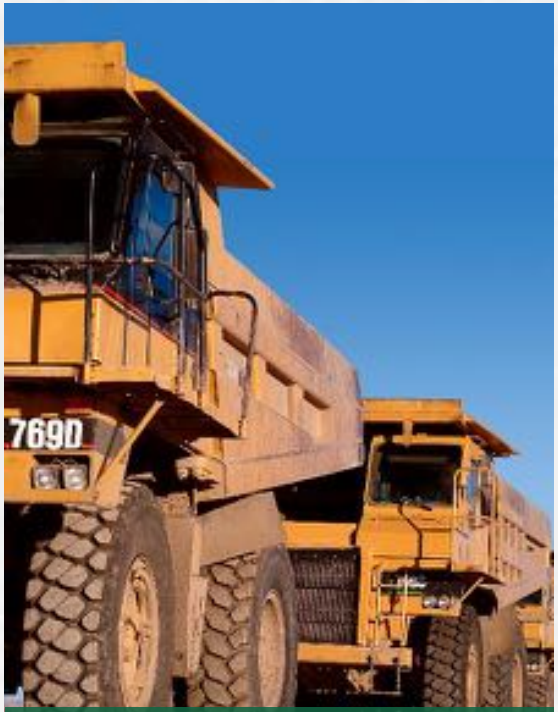
Law of economics

Law of the land

amazon web services

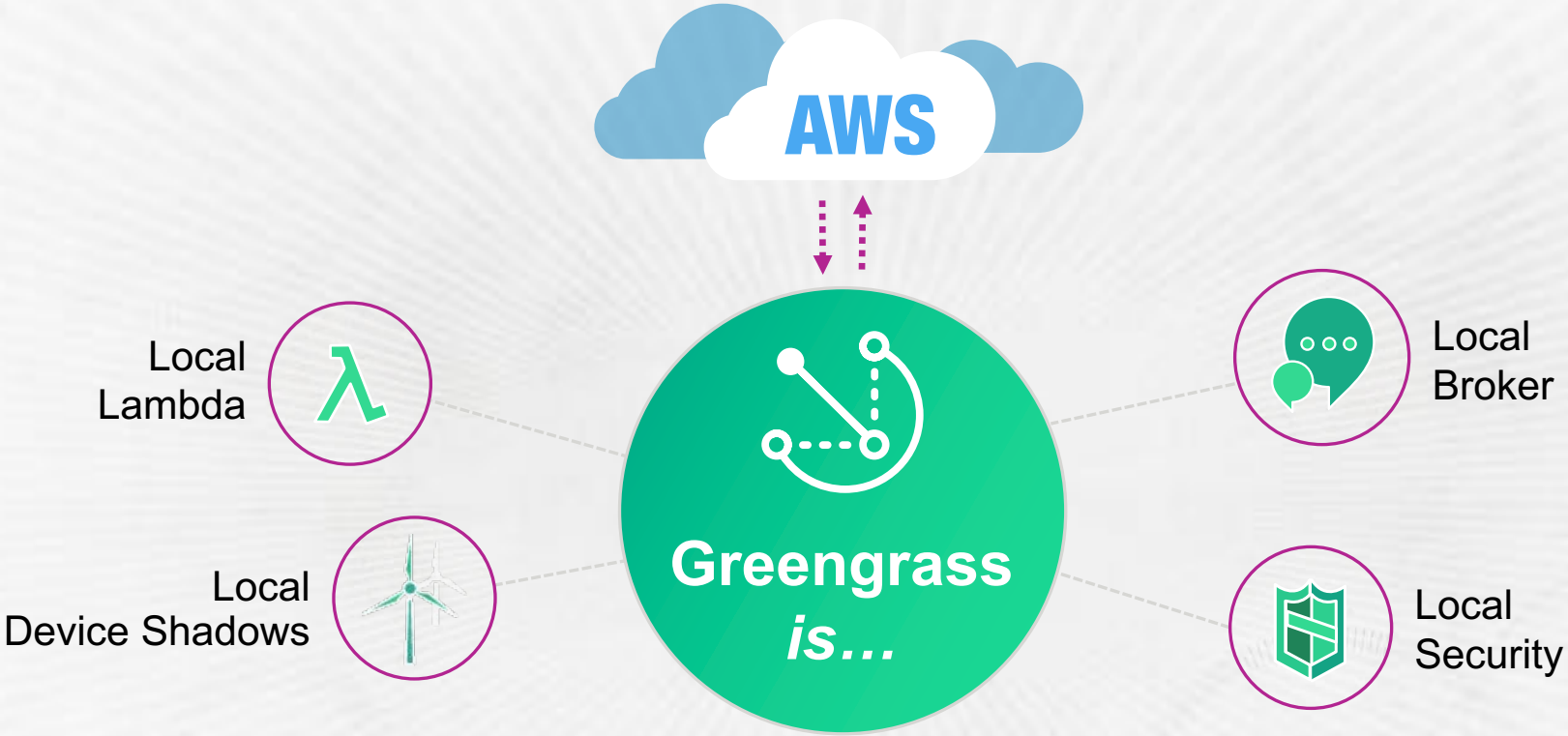
Most machine data never reaches the cloud

Medical equipment



Industrial machinery

Extreme environments


AWS
Local
Lambda
Local
Device Shadows
Greengrass
is…
Local
Broker
Local
Security

NOT hardware—you bring your own

Greengrass Core (GGC)

The runtime responsible for Lambda execution, messaging, device shadows, security, and for interacting directly with the cloud

Greengrass Core (GGC)

- Min Single-Core 1GHz
- Min 128MB RAM
- x86 and ARM
- Linux (Ubuntu or Amazon)

Greengrass Core (GGC)

The sky is the limit.

GGC takes advantage of your device's compute, memory, storage, and peripherals

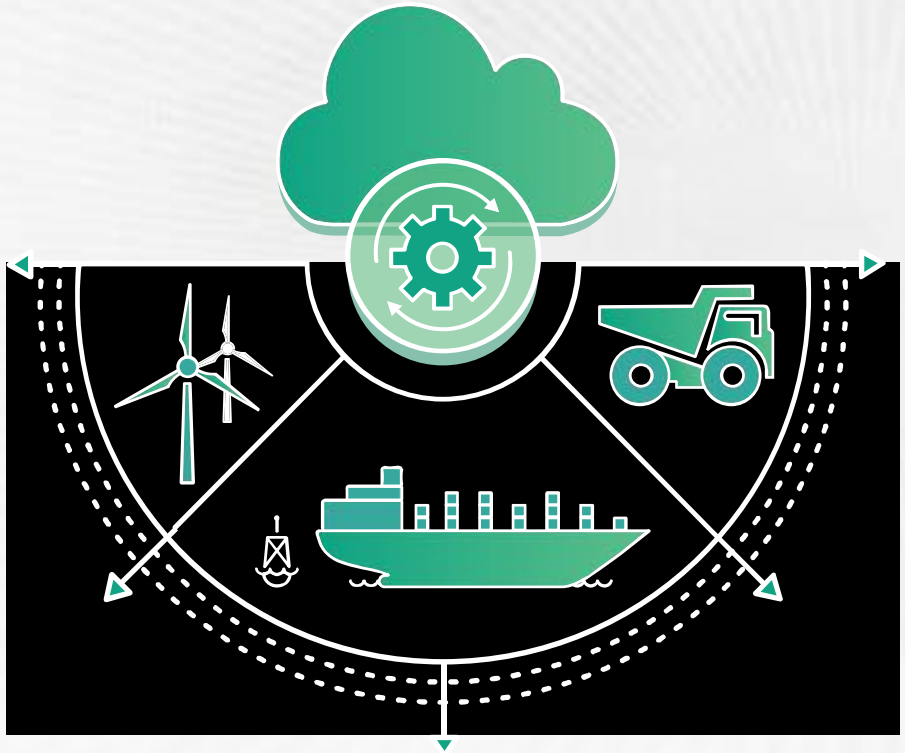
IoT Device SDK

Any device that uses the IoT Device SDK can be configured to interact with Greengrass Core via the local network

---

AWS IoT Device SDK supports C, C++, Python, JavaScript

Devices work together locally



A Greengrass Group is a set of Cores and other devices configured to communicate with one another

Devices work together with the cloud

Greengrass works with AWS IoT to maintain long-lived connections and process data via the rules engine

Your Lambda functions can also interact directly with other AWS services

Local Lambda

Greengrass runs Lambdas written in Python 2.7

Invoke Lambda functions with messaging and shadow updates

Local Lambda

Lambdas are event-driven compute functions

With Greengrass you can write Lambda functions in the cloud and deploy them locally

Local Lambda – What you can do

Command and control

Offline operation

Data filtering & aggregation

Iterative learning

Shadows

JSON documents that represent state of your devices and Lambdas

Define them however is logical to you—a car, an engine, a fleet

Sync to the cloud or keep them local

Shadows – What you can do

Device state (current and desired)

Granular device state (only synched to the cloud for debug)

Dynamic configuration (e.g,. numeric factors of an ML model)

Messaging

Local MQTT Pub/Sub messaging

Define subscriptions between publishers and subscribers

Apply MQTT topic filters

Security

Mutual auth, both locally and also with the cloud

Certificate on your devices can be associated to SigV4 credentials in the cloud

You can directly call any AWS service from Greengrass

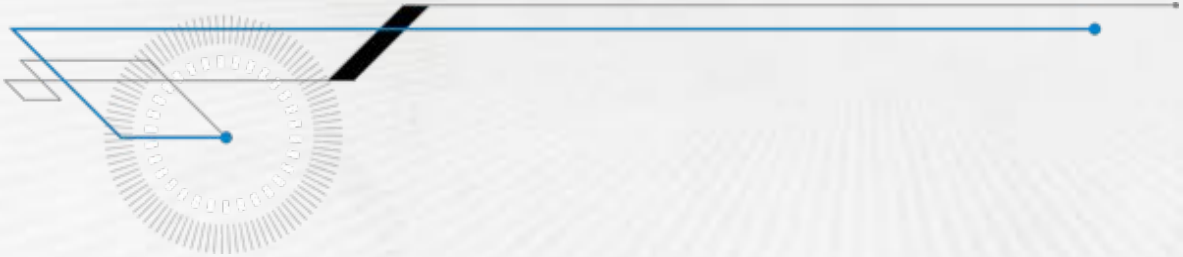
Security

The local GGC has it’s own root certificate.

Each device certificate is signed by the GGC’s root certificate.

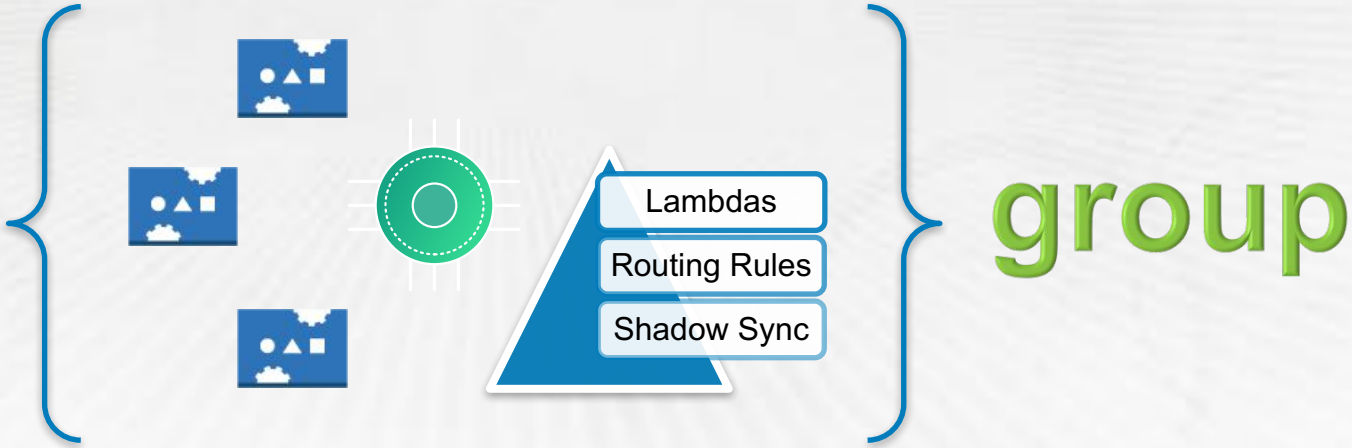
Greengrass core now uses the local CA and device certificate for authentication.

Greengrass technical deep dive



Groups

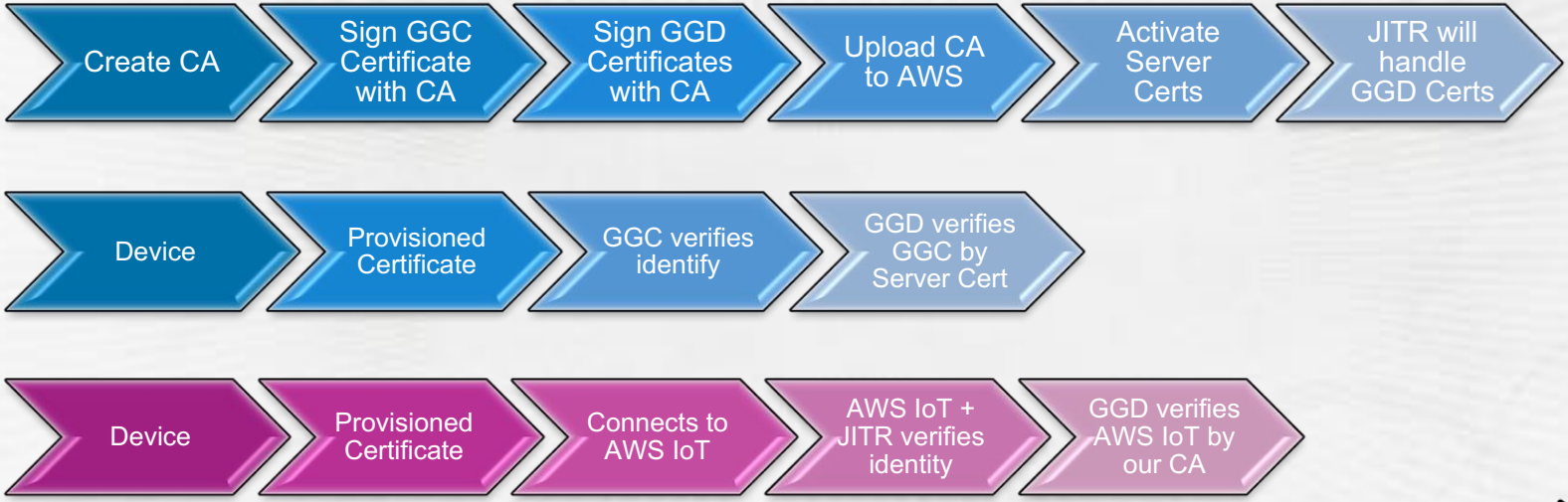
- Configuration construct, groups hold memberships
- Each membership list has only 1 Greengrass core
- Each device belongs to 1 or more memberships
- You need to have an ARN to be in a group



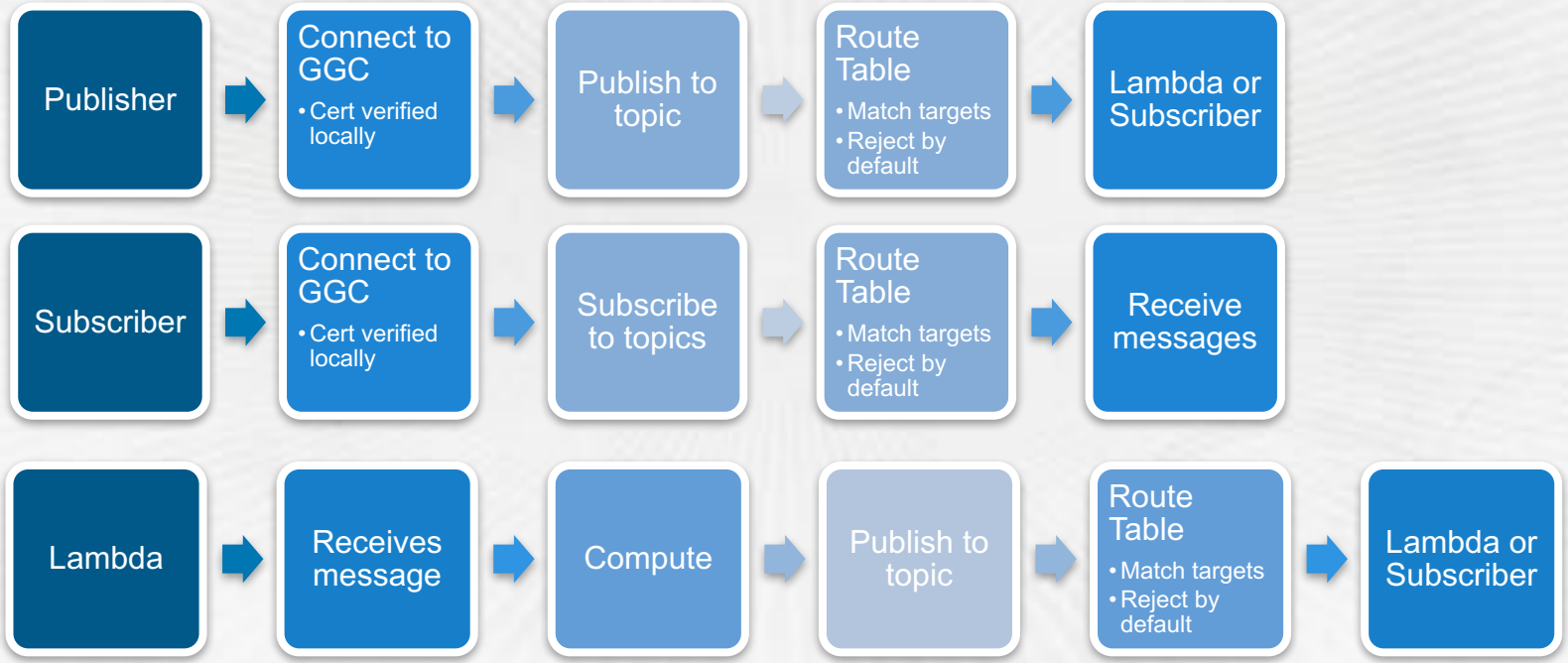
Groups + Deployments

- Deploy Lambda functions to your GG cores
- Deploy route table configuration
- Centrally manage all your devices

Security


Create CA
Sign GGC Certificate with CA
Sign GGD Certificates with CA
Upload CA to AWS
Activate Server Certs
JITR will handle GGD Certs
Device
Provisioned Certificate
GGC verifies identify
GGD verifies GGC by Server Cert
Device
Provisioned Certificate
Connects to AWS IoT
AWS IoT + JITR verifies identity
GGD verifies AWS IoT by our CA

Broker Specifics



Configuration

- Requirements
  - Linux – 4.4 with OverlayFS enabled (we test on 4.4 but 3.18+ with OverlayFS is an option)
  - glibc libraries 2.14
  - python 2.7
  - SQLite 3+
- Hardware Requirements
  - x86_64, ARMv61, AArch64
  - 128MB Ram
  - 1Ghz CPU

Configuration…

- Packages
  - Download the Greengrass Core package and extract to /greengrass
  - Download the Python SDK package and extract to /greengrass
  - Make a note of the CLI package, you will need to install this for the CLI to recognize GG commands

Configuration…

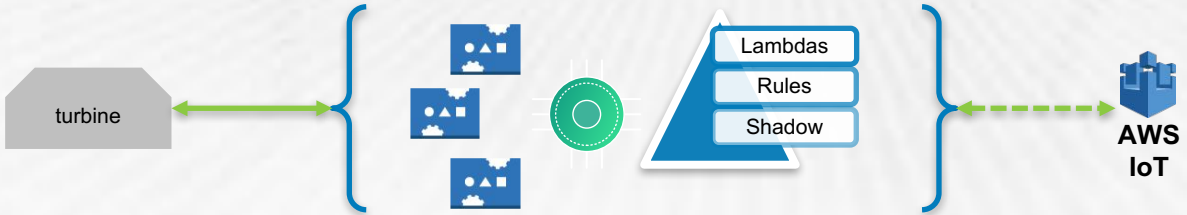
- Security
  - Setup an IAM account with AWS IoT and Lambda permissions
  - Configure your environment with the IAM access keys
- Certificates
  - Create your own CA
  - Create a server certificate + key for the GGC
  - Create a cloud certificate for communication between AWS IoT -> GGC
  - Create device certificates + keys, all signed by your CA
  - Upload your CA to AWS IoT
  - Activate your CA and Cloud Certificate

Use Cases

- Edge compute
  - Lambda functions contain local business logic + rules
  - Low latency communication between devices
  - Full application stack can be moved to Lambda benefiting from container like scaling
- Data filtering
  - Lambda functions are used to filter data
  - Local databases can be used for storage
  - Only upload snapshots or data or compressed data

Use Cases - example

- Gas Turbine Filter Maintenance
  - Each turbine consists of multiple devices/sensors
  - Each turbine has a single Greengrass core
  - Lambda functions have enough logic to perform edge machine learning
  - Models and Lambda changes can be centrally updated
  - Lambda filters data and using it's edge ML can determine when to request maintenance/replacement of filters
  - Only required data is sent up to AWS IoT once an hour



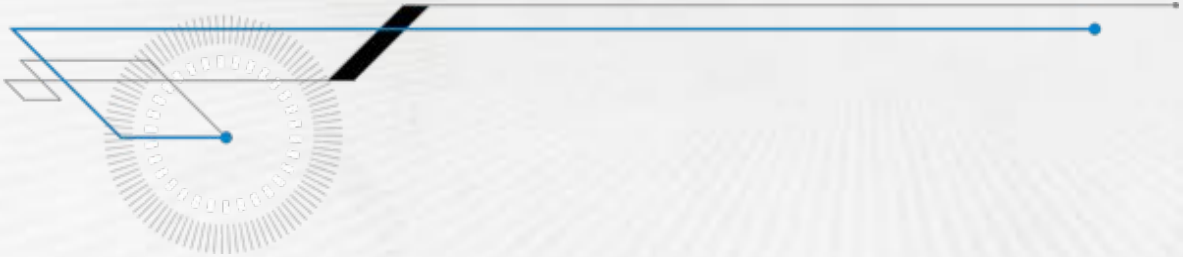
FAQ

- What about shadow sizes?
  - Shadows can be synced to the cloud and so the limit matches the cloud. 8k
- Can GGC's communicate with each other?
  - Not natively, you have to create a proxy using a connected "device" that will relay between two cores
- Can we use wildcards in the route table?
  - No, explicit rules have to exist for all targets and sources
- Can we subscribe to cloud topics?
  - Yes, the source is "cloud" and you specify a local target
- Can we compile for other architectures?
  - No, please make your request known to the BD team
- How do we update Greengrass core?
  - Manual, package managers

Greengrass pricing

Active Devices	Price
3	Free for 1 year
3–10,000	$0.16/month $1.49/year
10,000+	Call us

Building blocks of AWS IoT



how did we pick these services?

“these services are the backbone of *any IoT solution*
built on AWS to support *highly resilient and scalable systems*”

The IoT Building Blocks

- AWS IoT
- API Gateway
- Cognito
- Lambda
- SNS and SQS
- Kinesis + Kinesis Firehose
- DynamoDB

API Gateway

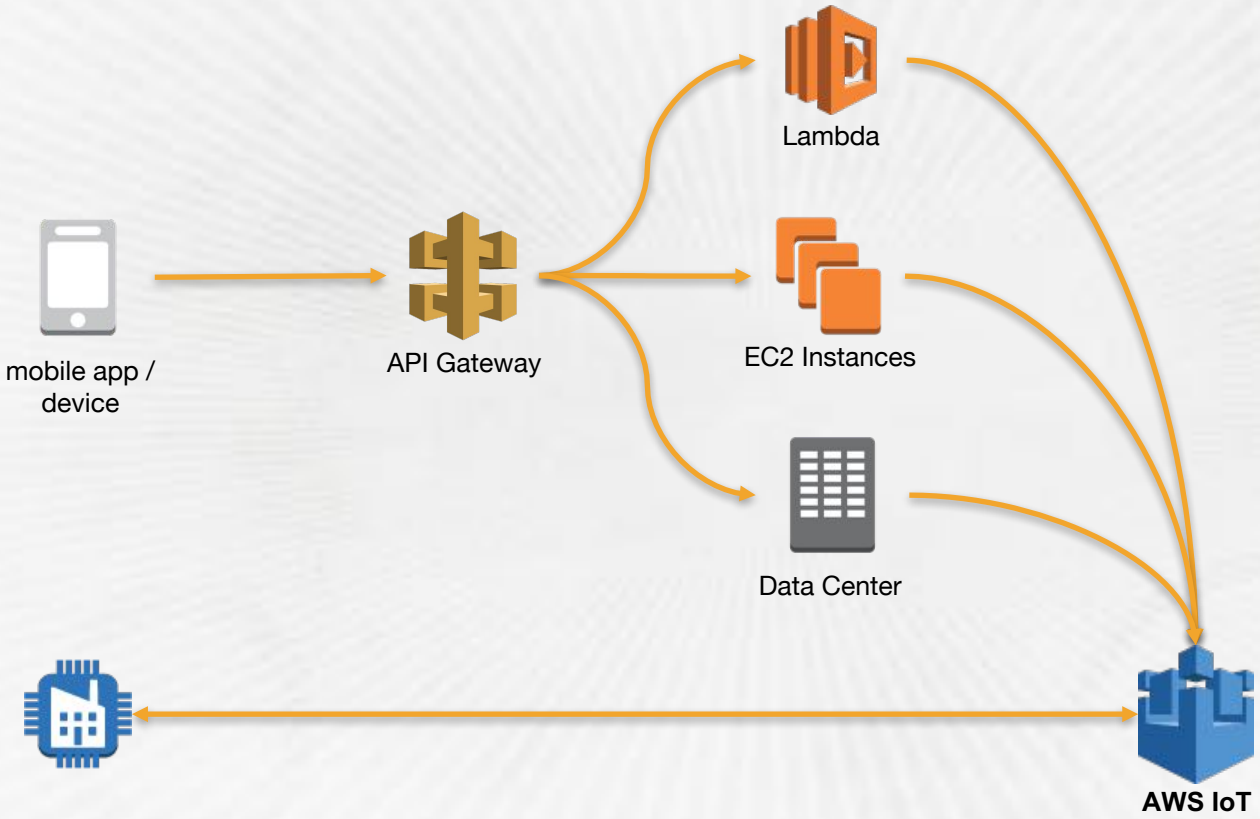
API Gateway

Build, deploy and manage APIs.

- Great first entry-point for applications especially mobile
- Presents a unified front for all your backend services
- Direct integration with AWS Lambda
- Integrated with Cognito for authentication
- Protection from unwanted traffic
- SDK generation, caching and throttling
- Support swagger imports and exports

amazon web services

API Gateway & IoT Usage



amazon web services

Authentication

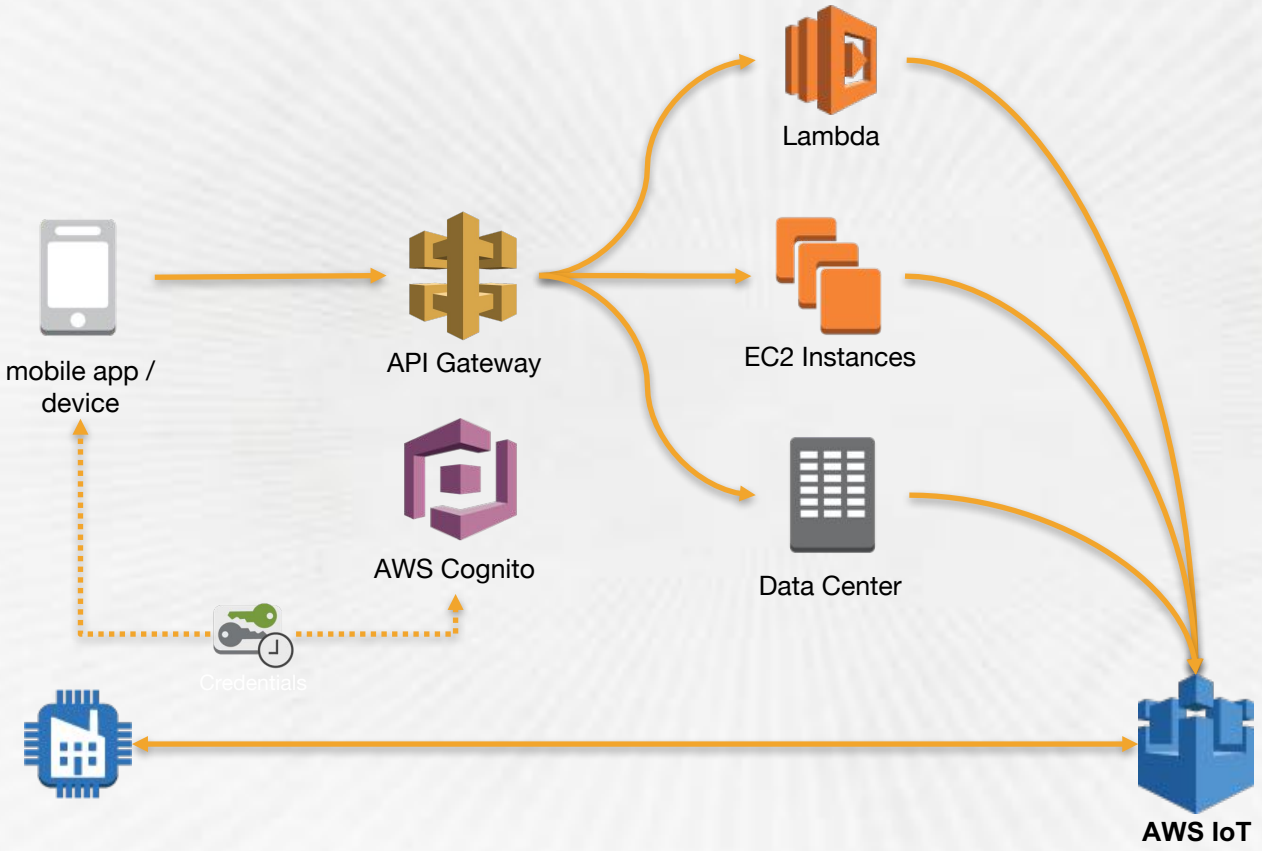
Cognito

User Identity and App Data Synchronization

- Cognito is a simple user-data synchronization and identity service that helps you securely manage and synchronize app data for your users across their mobile devices.
- Authenticate users with popular public identity providers(Amazon, Facebook, Google and any other OpenID Connect compatible identity provider) or support unauthenticated guest users and use the AWS Cloud to save and sync user data for their mobile.
- Amazon Cognito can integrate with your existing identity System.

amazon web services

API Gateway, Cognito & IoT Usage



amazon web services

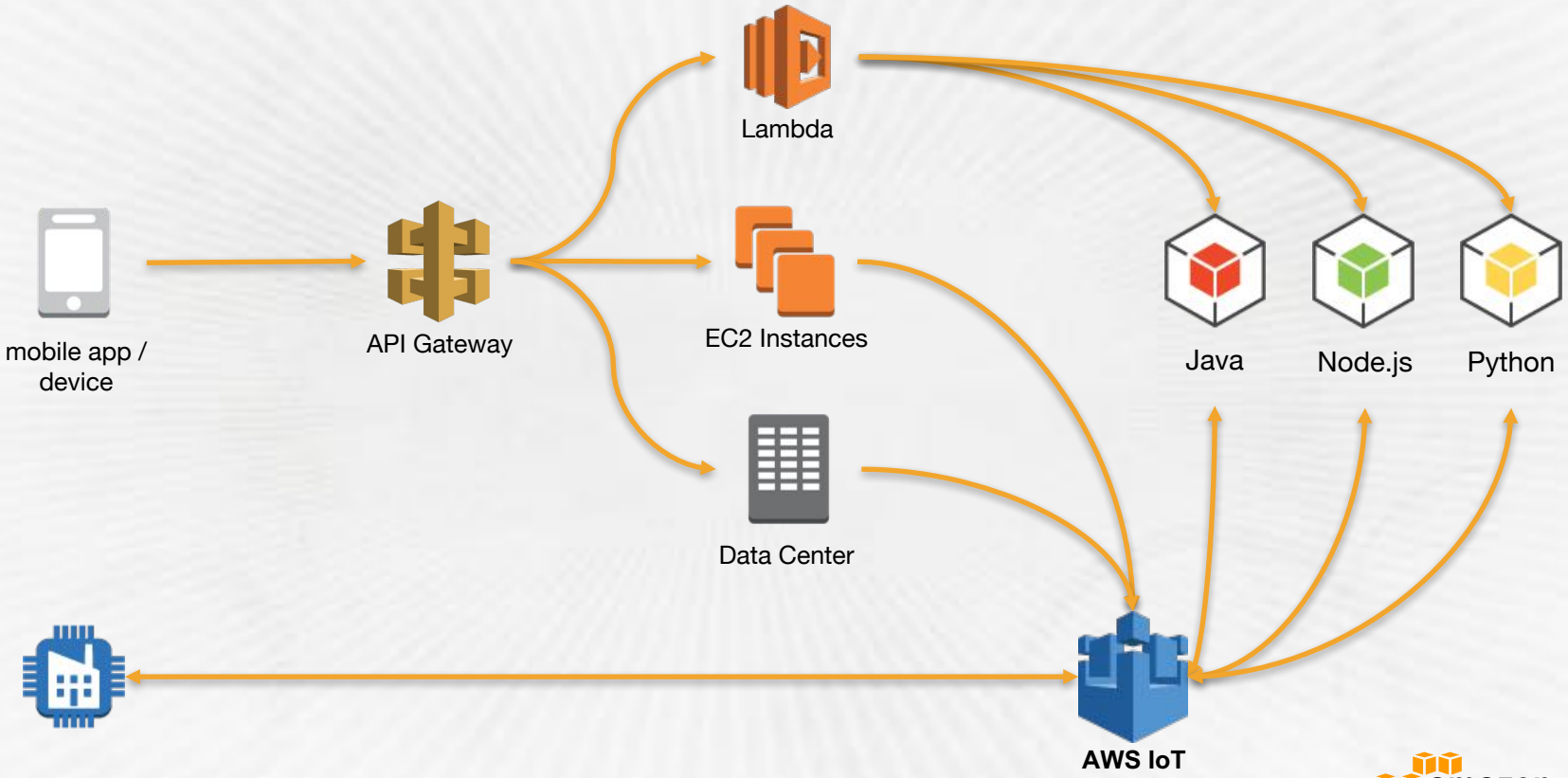
Lambda

AWS Lambda

Serverless event driven compute service

- Bring your own code
- Simple resource model
- Flexible invocation paths
- Permissions integrated with all services, Cognito/APIG
- Completely stateless
- Integration with AWS IoT makes this a key service

Lambda & IoT Usage



SNS

Simple Notification Service (SNS)

Push Notification Service

- Set up, operate, and send notifications
- Publish messages from an application and immediately deliver them to subscribers or other applications
- Push messages to mobile devices
- Supports notifications over multiple transport protocols – Email,SMS,HTTP/HTTPS

amazon web services

SQS

Simple Queue Service (SQS)

Message Queue Service

- Managed and scalable message queue
- Building block for distributed systems
- Highly available and durable
- Pay for what you use

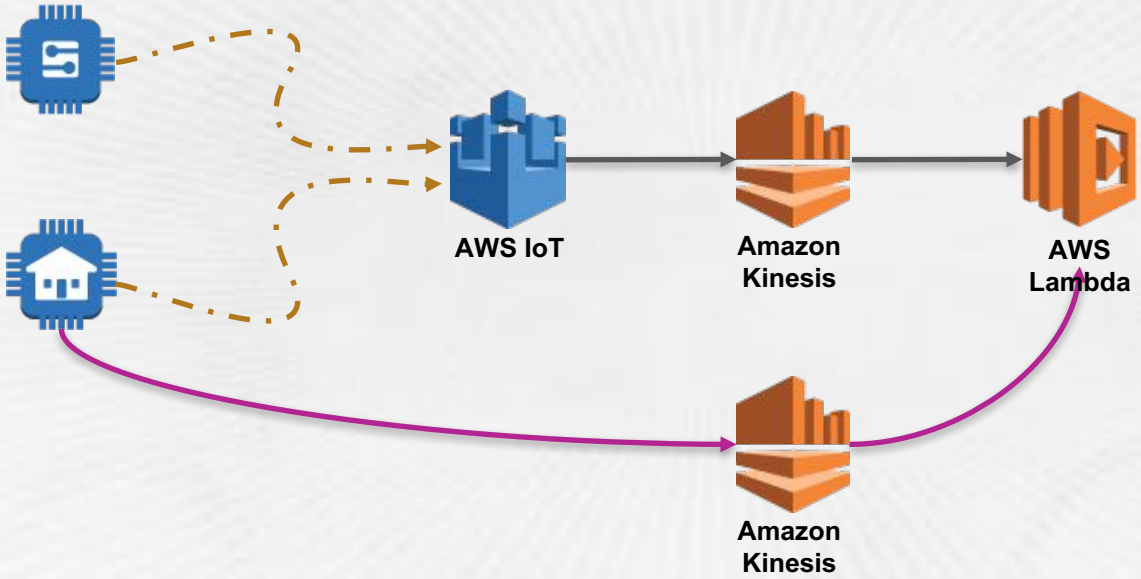
Streaming with Kinesis

Kinesis

Real-time Processing of Streaming Big Data

- Fully-managed service for real time processing of streaming data, at any scale
- Kinesis can continuously capture and store terabytes of data per hour from hundreds of thousands of sources
- Integrate with S3, Dynamo DB, and Amazon Redshift
- Build custom applications on top of Kinesis data

Kinesis and AWS IoT



amazon web services

DynamoDB

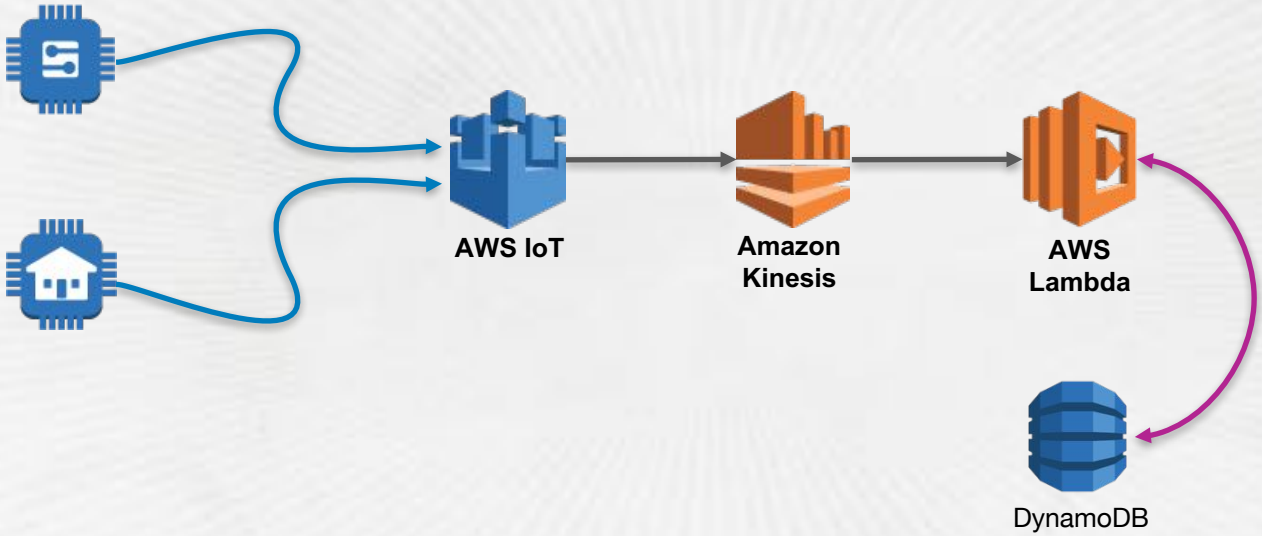
DynamoDB

Predictable and Scalable NoSQL Data Store

- Fast, fully-managed NoSQL Database Service
- Capable of handling any amount of data
- Durable and Highly Available
- Fast predictable performance backed by SSD storage
- Can scale up to millions of IOPS
- Simple and Cost Effective

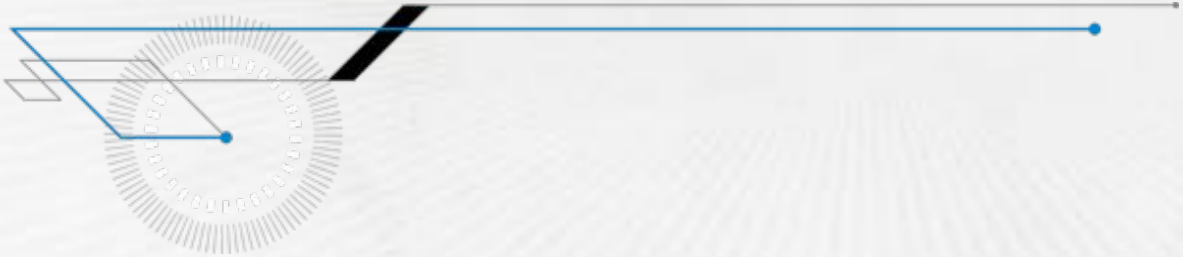
amazon web services

DynamoDB, Lambda and AWS IoT

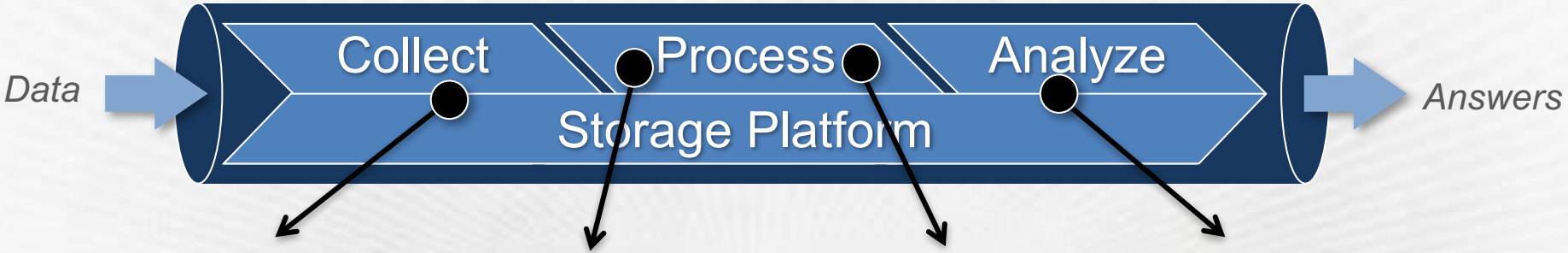


amazon web services

AWS IoT & Big Data



Big Data & Analytics Lifecycle



Data Collection and Storage

- Amazon S3
- Amazon Kinesis
- Amazon DynamoDB
- Amazon RDS (Aurora)

Event Processing

- AWS Lambda
- KCL Apps

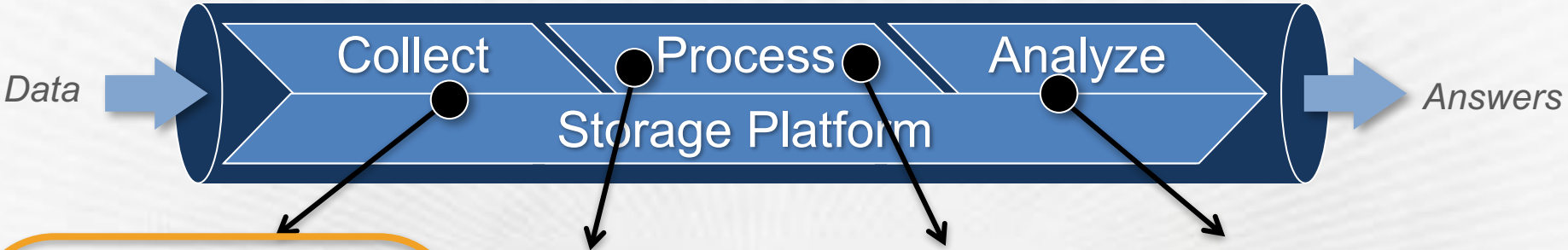
Data Processing

- Amazon EMR

Data Analysis

- Amazon Redshift
- Amazon ML
- Amazon QuickSight

Big Data & Analytics Lifecycle



Data Collection and Storage

- Amazon S3
- Amazon Kinesis
- Amazon DynamoDB
- Amazon RDS (Aurora)

Event Processing

- AWS Lambda
- KCL Apps

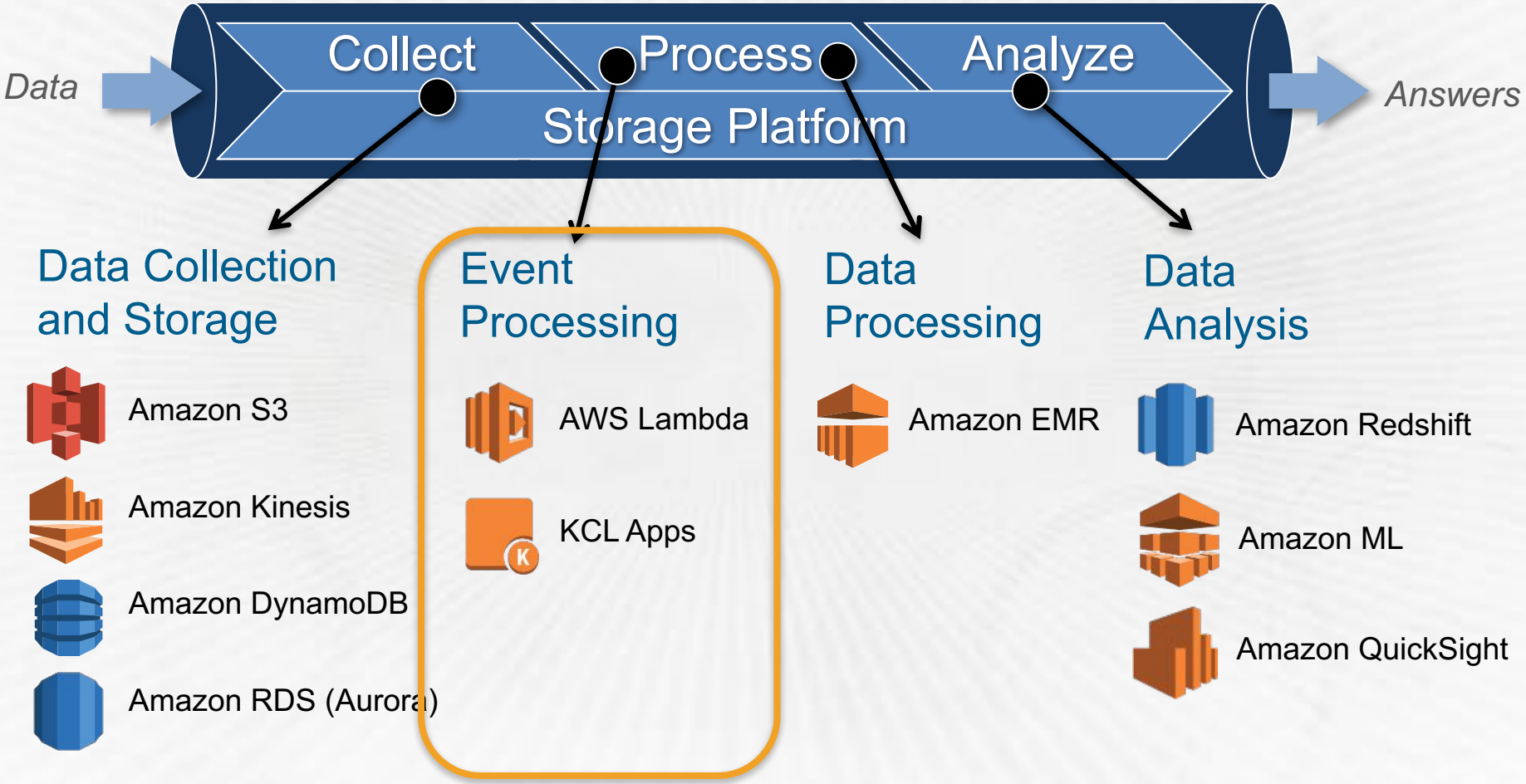
Data Processing

- Amazon EMR

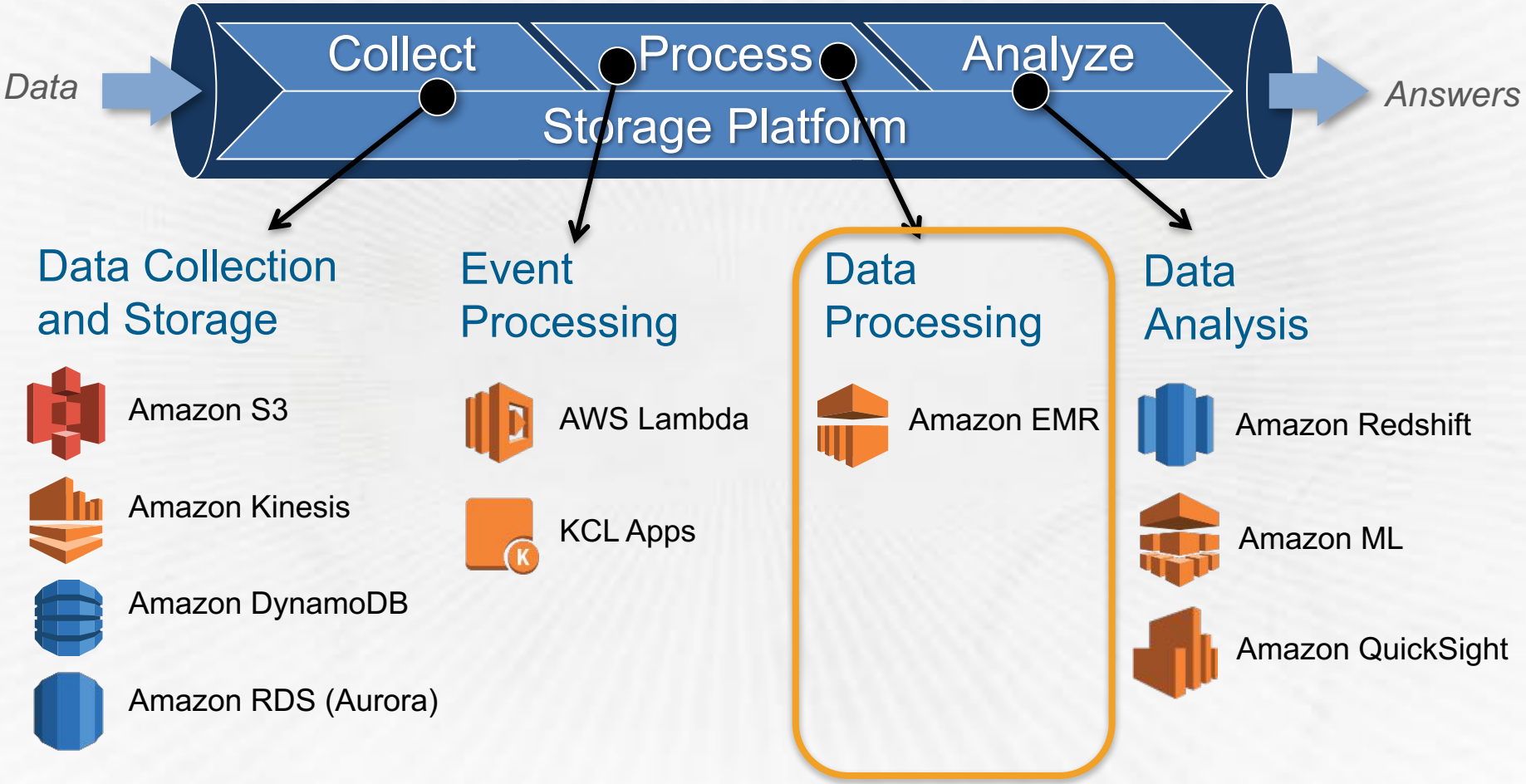
Data Analysis

- Amazon Redshift
- Amazon ML
- Amazon QuickSight

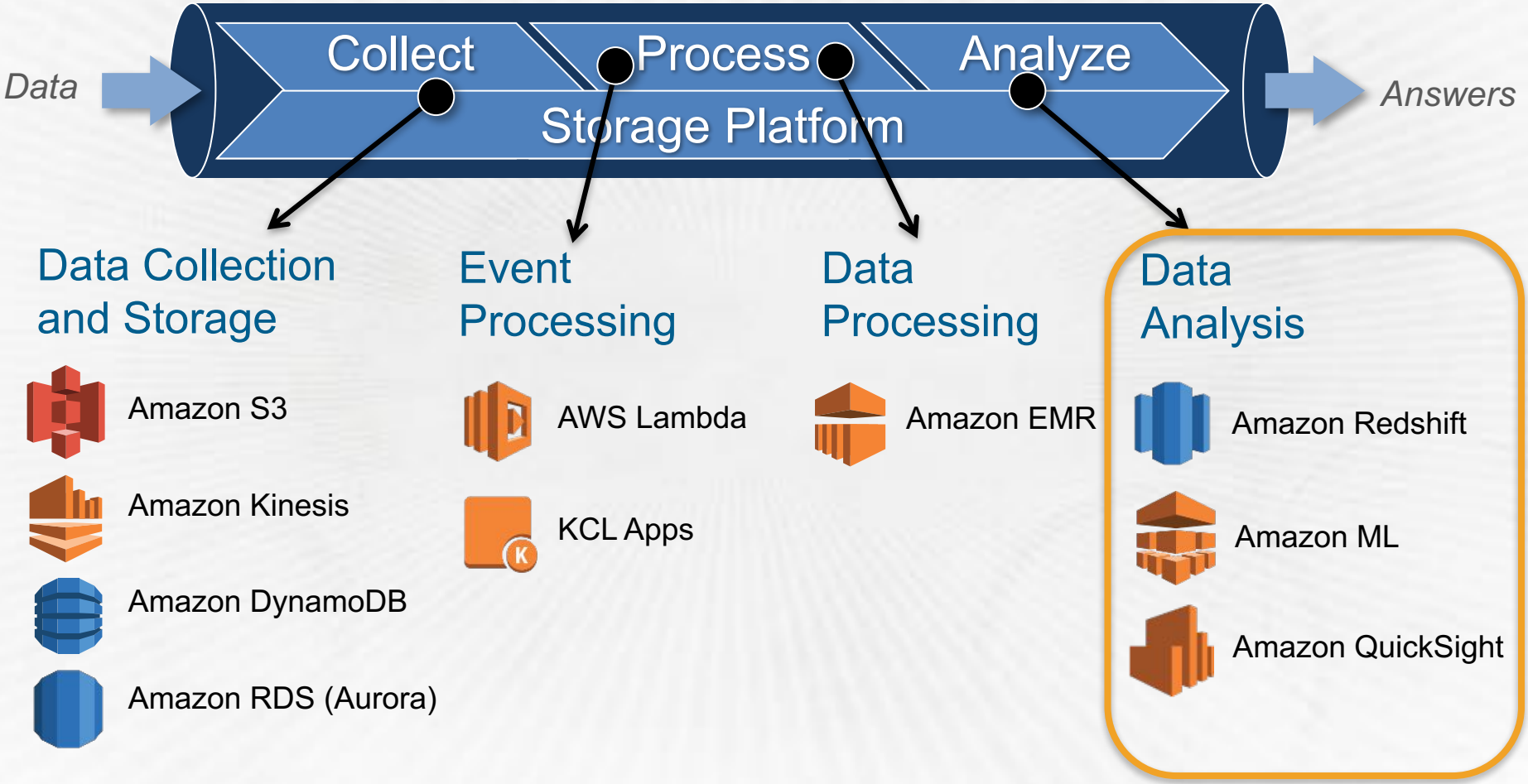
Big Data & Analytics Lifecycle



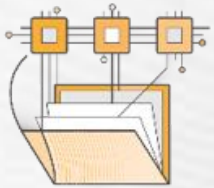
Big Data & Analytics Lifecycle



Big Data & Analytics Lifecycle

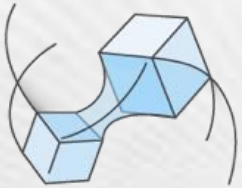


AWS Storage is a **Platform**

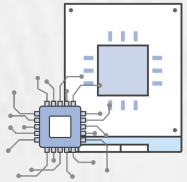


Amazon EFS

File



Amazon EBS



Amazon EC2 Instance Store

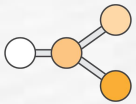
Block

Amazon S3

Amazon Glacier

Object

Data Transfer



Internet/VPN

Amazon S3 Transfer Acceleration

Amazon CloudFront

AWS Direct Connect

Amazon Kinesis Firehose

AWS Storage Gateway

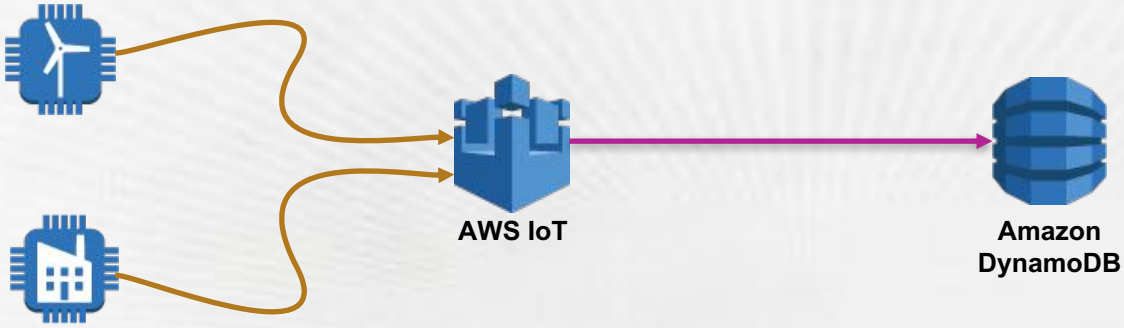
AWS Snowball

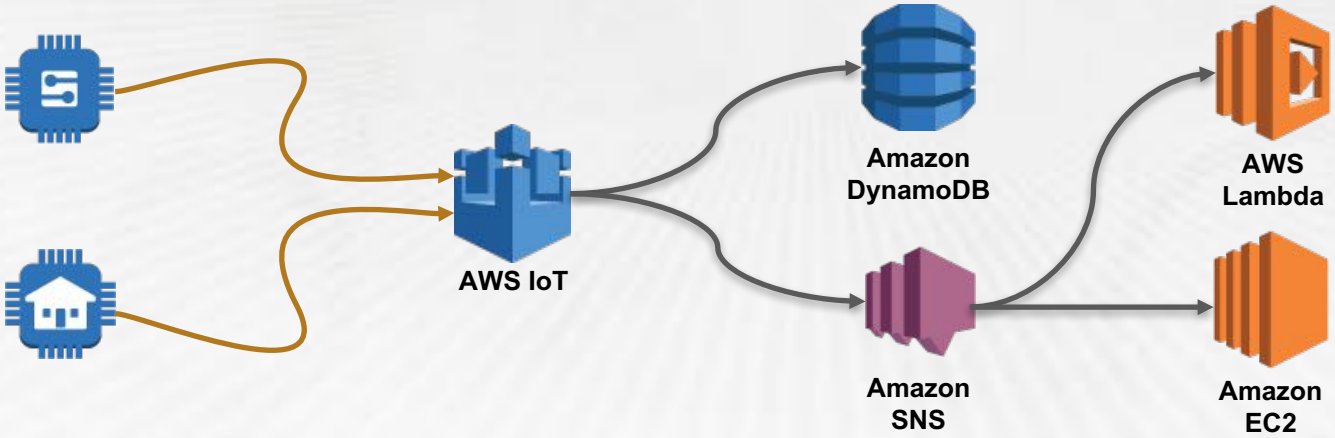
ISV Connectors

Patterns we'll find in the labs

- We'll look at patterns that focus just on AWS IoT
- Patterns that combine multiple AWS Services vs Single Services
- A look at patterns that apply to existing architecture, especially mobile
- Patterns for Mobile, API Gateway and then AWS IoT
- Some ideas around critical event architectures

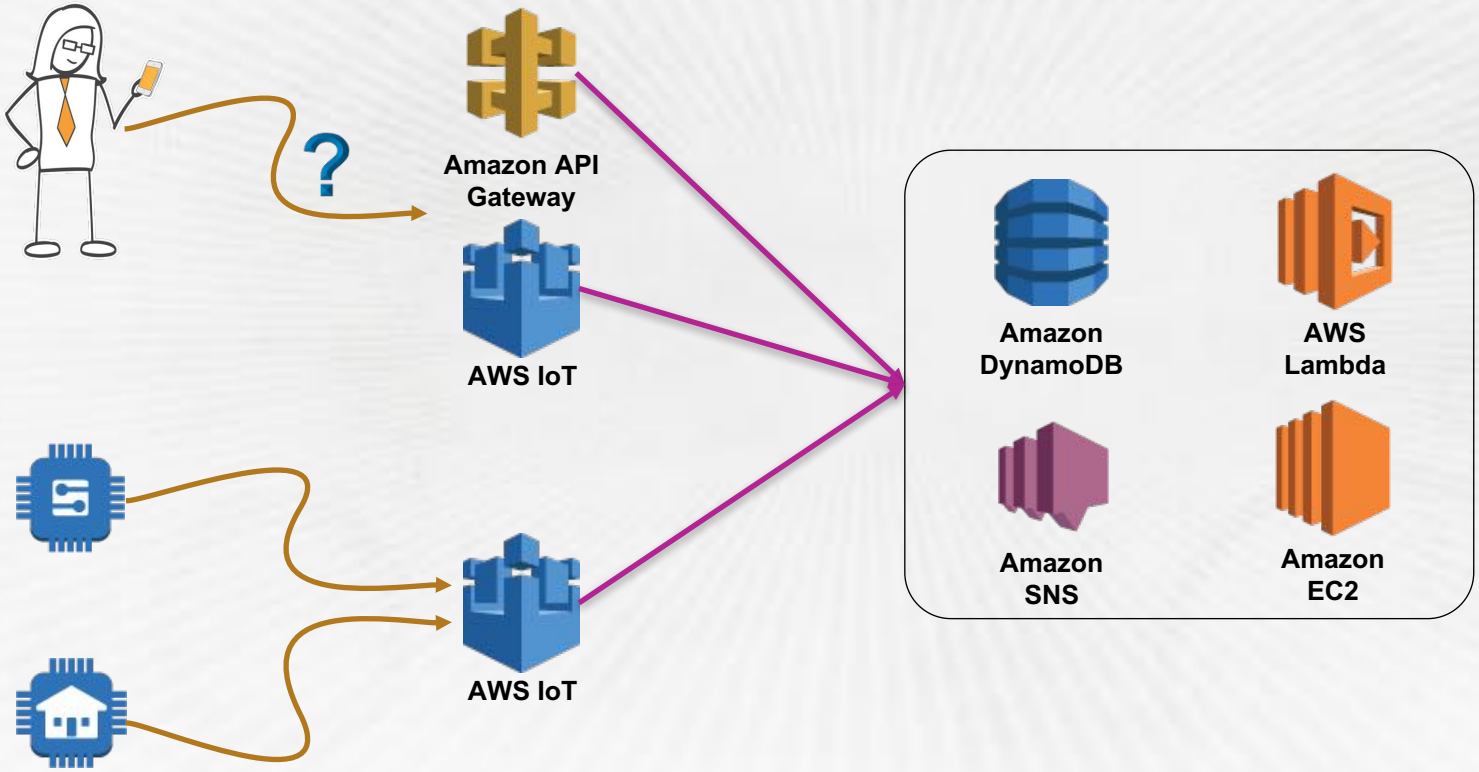
Problematic Single Service Pattern





amazon web services

Fragmented Architecture



amazon web services

Where is my logic?

- With all the services available to us now our logic could be:
  - In Lambda?
  - In AWS IoT rules?
  - In API Gateway transformation (VTL) rules? (it's where?)
  - In EC2 instances connected with SQS, SNS or Both…
  - Client side? Now includes apps, browsers, mobile devices and things!
  - In PL/SQL?

amazon web services

Where is my logic? Found.

- Recognize existing architecture and integrate AWS IoT appropriately.
  - This may result in heavier use of the API Gateway
  - It can often result in longer term migration to AWS Lambda
- Can it all be in Lambda?
  - Sure, being event driven it's great for rapid decision making on thousands of events
  - Remember to couple with SQS and SNS to keep it resilient in the event of failures
- Overall
  - Don't forget good architecture, centralized logic, resilient and modular
  - AWS IoT rules are great for routing information, decisions don't "have" to be there

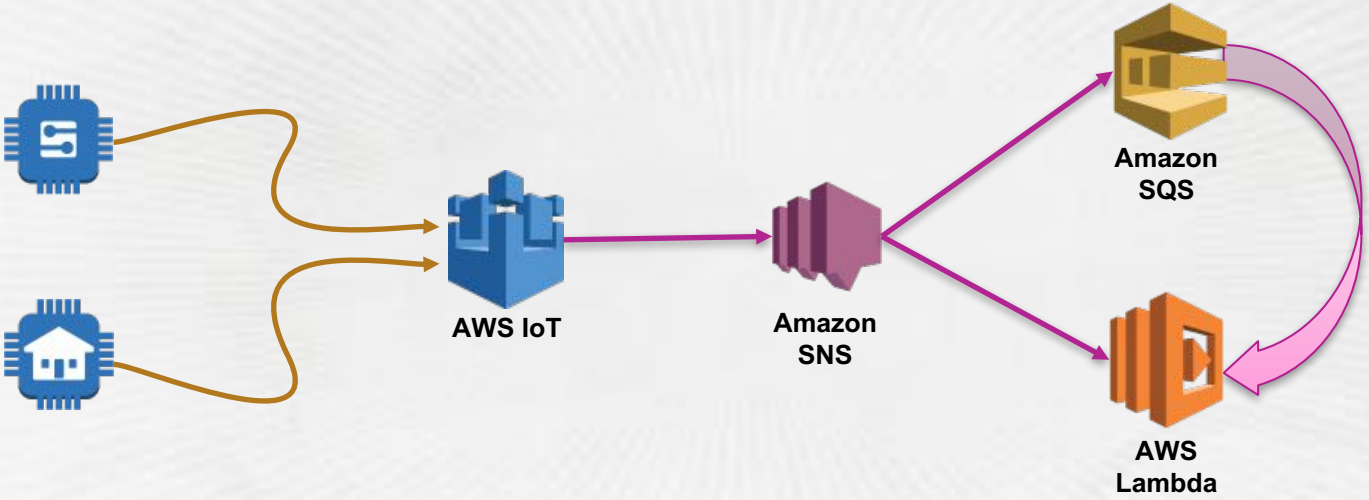
Debugging Labs

- Use the MQTT client in the AWS Console and subscribe to #.
- Verify your topics are getting messages.
- Verify your next step “role” in IAM.
- Check logs at the next step via CloudWatch.

Lab 1 – Getting Setup

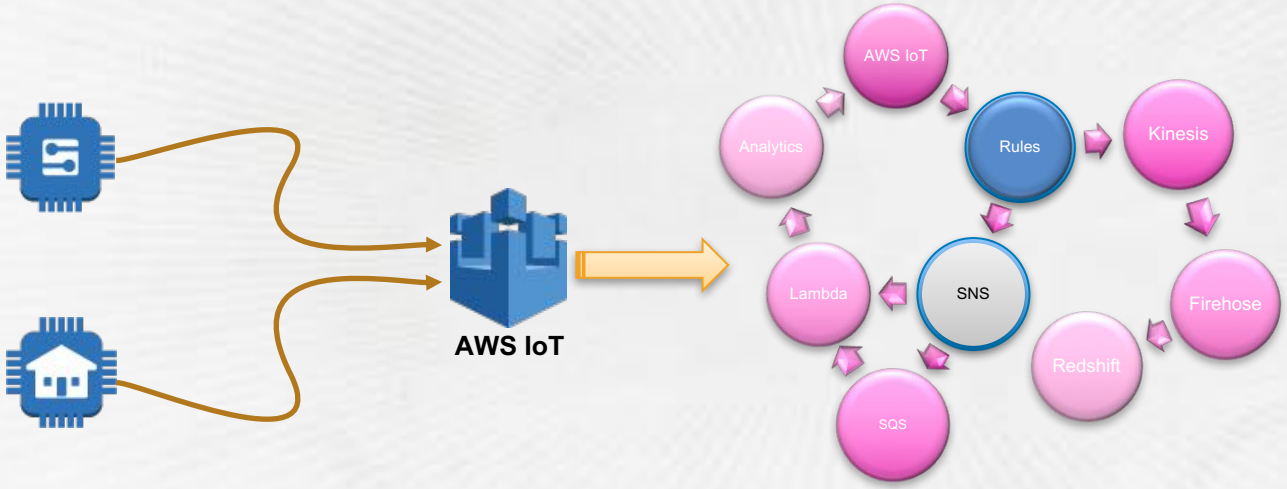
- Login into the AWS Console and open the AWS IoT dashboard.
- Create a new “Thing”.
- Create a new set of certificates, make sure you download them!
- Create a new security policy.
- Associate the security policy, thing and certificates.
- Configure your local client. (Intel Edison or RPi)

Lab 2 – The SNS hook



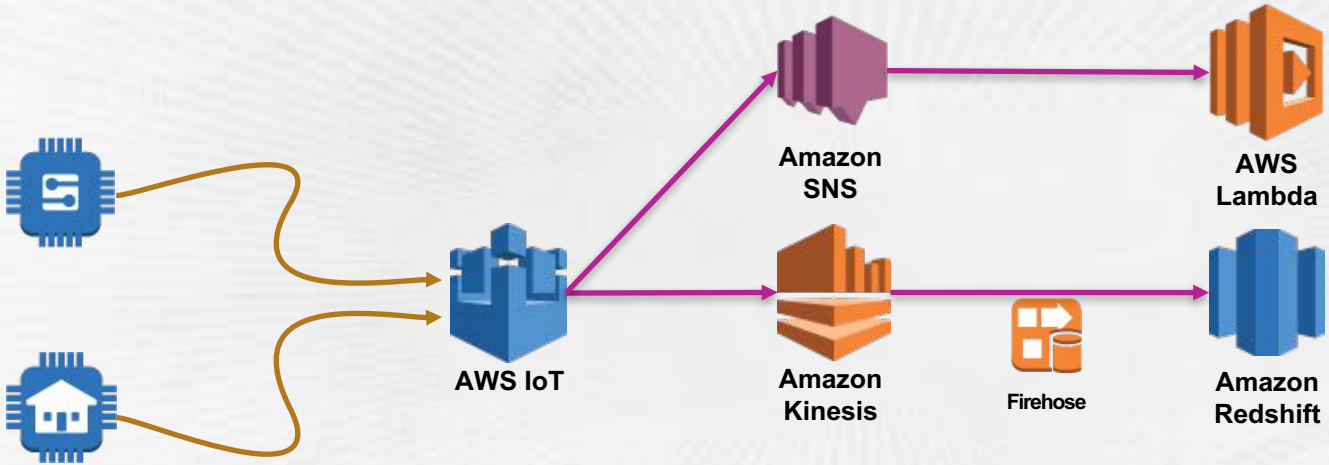
amazon web services

Lab 2 – The SNS hook – Expanded Look



amazon web services

Lab 3 – Streaming data to Redshift

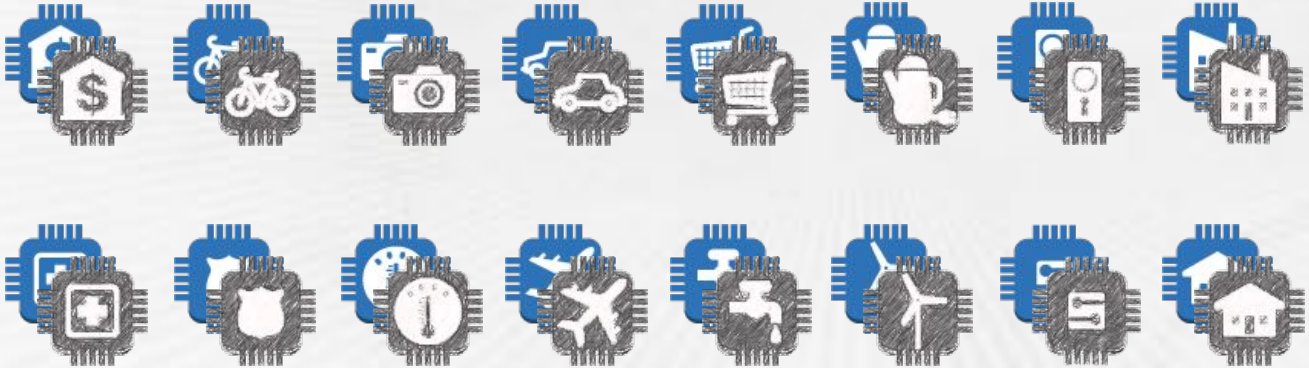


Lab 4 – Websockets

- AWS IoT now supports Websockets …
- Good for control from potential non-thing devices like
  - Mobile Applications
  - Websites
  - Regular desktop/pc control
- Might not be too great on the “thing”
  - Requires more overhead to simply run Websockets
  - Would require custom data transmission format or
  - There are some MQTT over Websockets libraries
  - Not as standard as native MQTT
  - Websockets are not magical

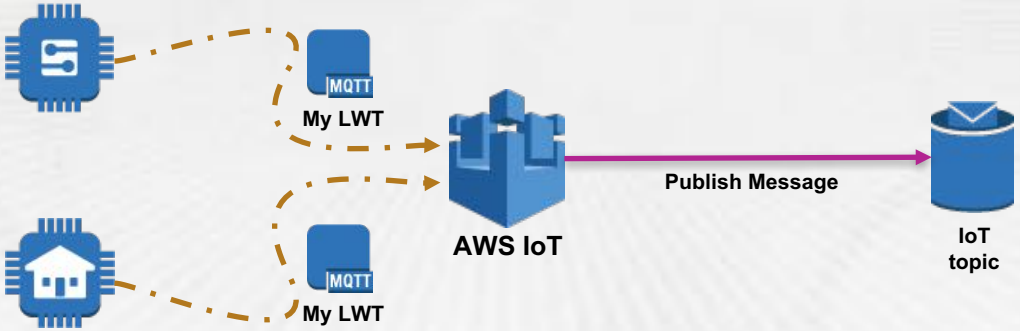
Lab 5 – Thing Shadows

- Build a client to monitor our Shadow
- Run a second client to trigger shadow updates and see state change



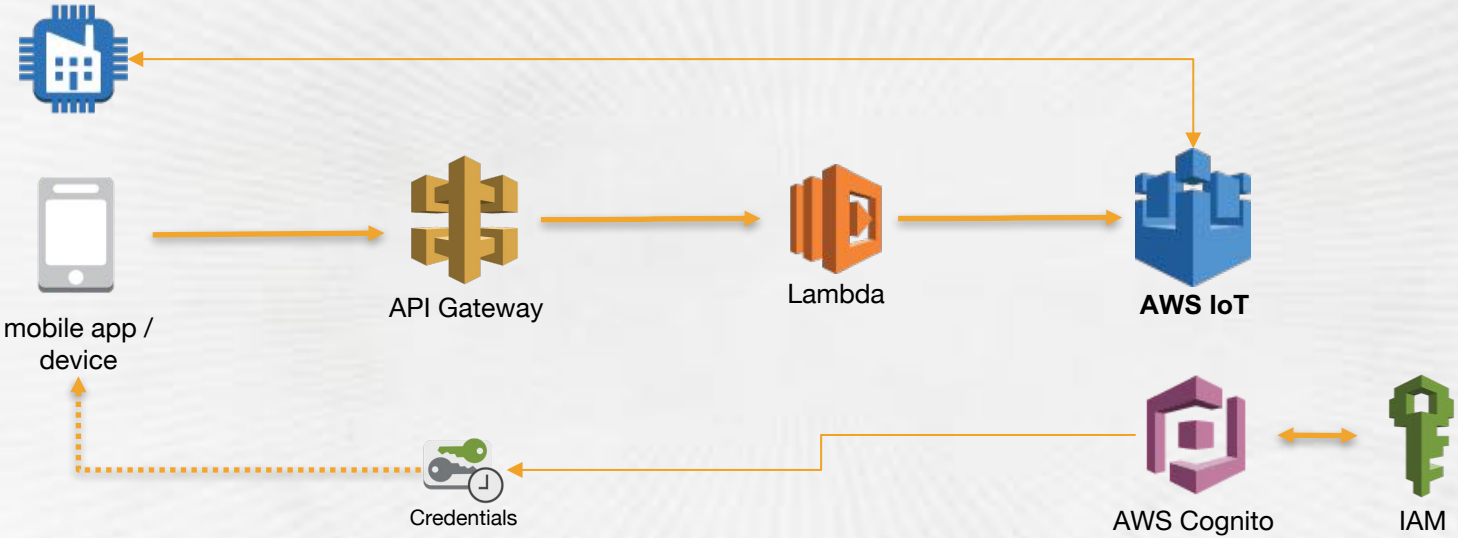
Lab 6 – Failure scenarios

- What happens when our client disconnects unexpectedly?
- How do you sent a “help me” when you’re offline?
- Lucky for us, MQTT has support for LWT - Last Will and Testament
- Setup a message that is published to a topic when a client disconnects abruptly



amazon web services

Lab 7 – API Gateway and Mobile C&C



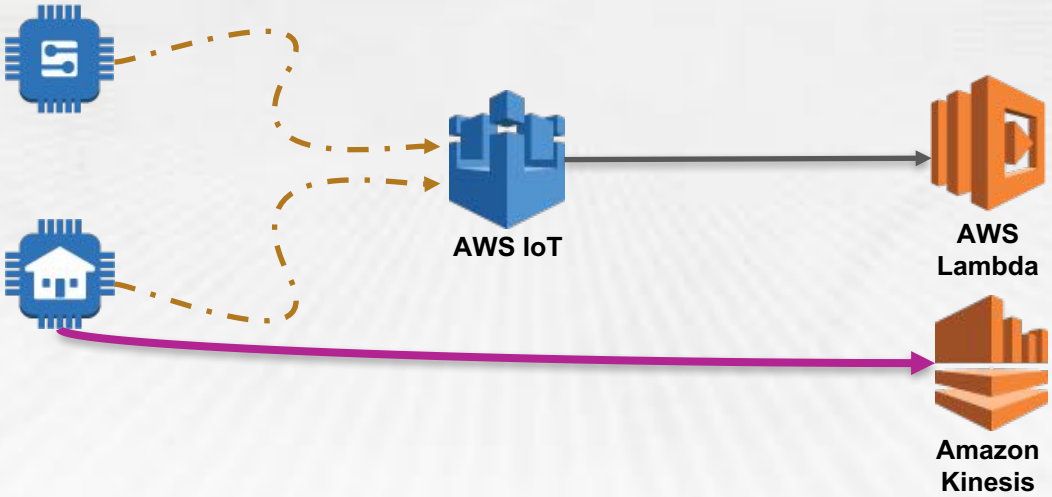
amazon web services

Lab 8 – Streaming data to DynamoDB

- Great for quick storage of large numbers of events
- Should be used as a parallel stream
- Data can be crunched later, let’s get it somewhere safe
- Good for Lambda lookups

Lab 9 – Switch to Kinesis, critical events

- Using regular message streaming to AWS IoT
- What happens during a critical event
- Enable high volume streaming directly to Kinesis



amazon web services

Thank You!

craiwill@amazon.com

Please give us feedback, http://bit.ly/iotfeedbackboston