Workshop Wiki Page

This slidedeck is designed to be delivered with the instructions detailed on the following internal wiki page.
Feel free to contact us if needed : @antoeddi & @desmaisn

https://w.amazon.com/index.php/AWS/EMEA/SA/Partner/IoT_Simple_Workshop

amazon web services

IoT Workshop

Dedicated for AWS Partners
25th November 2015
Clichy, France

Philippe Desmaison
Ecosystem Solutions Architect
Amazon Web Services
desmaisn@amazon.fr

Antoine Eddi
Ecosystem Solutions Architect
Amazon Web Services
antoeddi@amazon.fr

amazon web services

WiFi

SSID: Guest
Password: <latest password>

amazon web services

Agenda

– Part I – Getting started

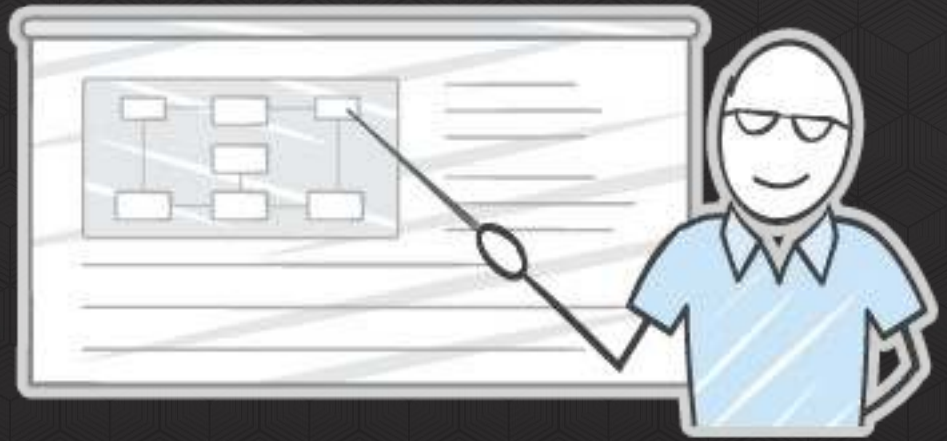
- Introduction
  - Logistics
  - Goals of this workshop
  - Basics about IoT for non IoT specialized attendees
  - Basics about AWS for non AWS specialized attendees
- AWS IoT concepts
  - Architecture overview

Agenda

– Part II – Deep dive on AWS IoT components

- AWS IoT components (Front)
  - Device SDKs
  - Devices, Broker & Security
- Live POC
  - Presentation
- AWS IoT components (Back)
  - Rules Engine
  - Device Shadow

Getting Started



Goals

amazon web services

Goals of the workshop

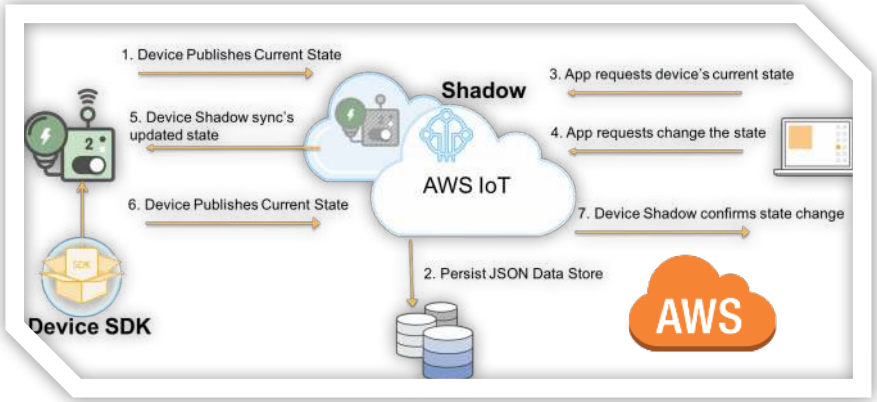
ONLY SLIDES +
DEMO EFFECT

STEP BY STEP
APPROACH

amazon web services

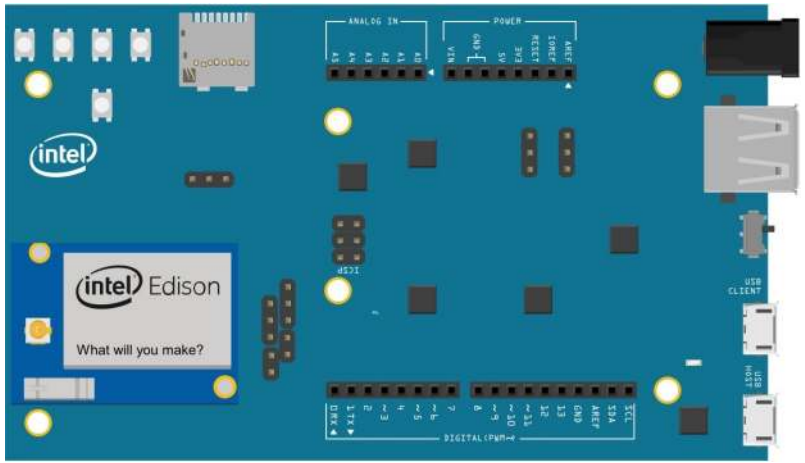
Goals of the workshop

Start of the day



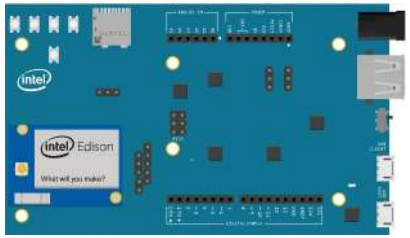
End of the day

Live Proof of concept



AWS IoT

Live Proof of concept – Device Preparation



Intel Edison

- Yocto Project (Linux foundation project)
  (Embedded Linux based on templates)

  +

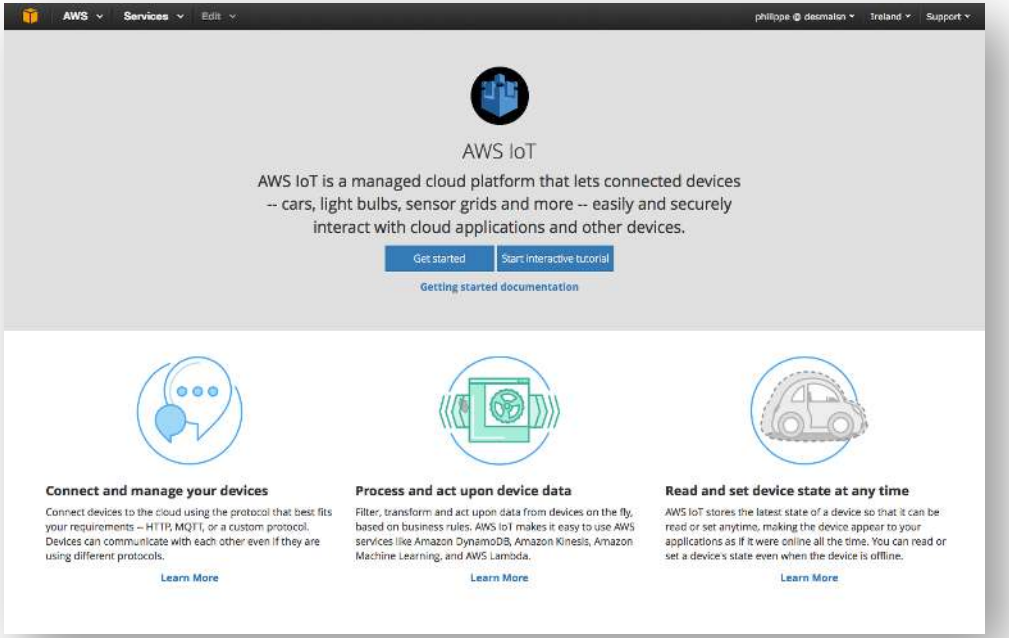
- NodeJS

Wifi configuration already done

amazon web services

Live Proof of concept – AWS IoT side

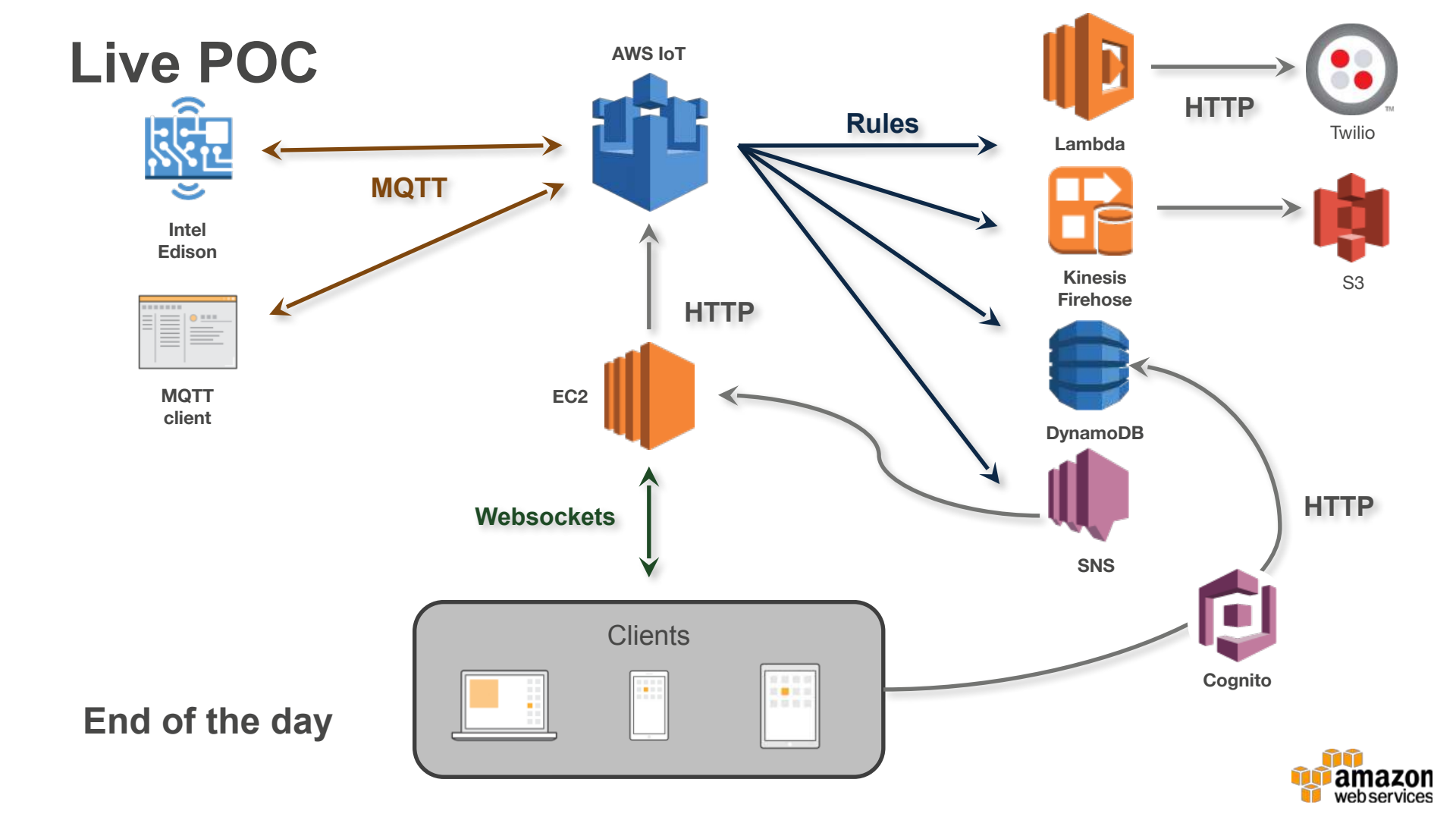
AWS IoT



AWS IoT service not configured.
We'll start from scratch.

amazon web services

Live POC



End of the day

amazon web services

IoT Basics

amazon web services

Protocols / Interaction

Device

Protocols – MQTT

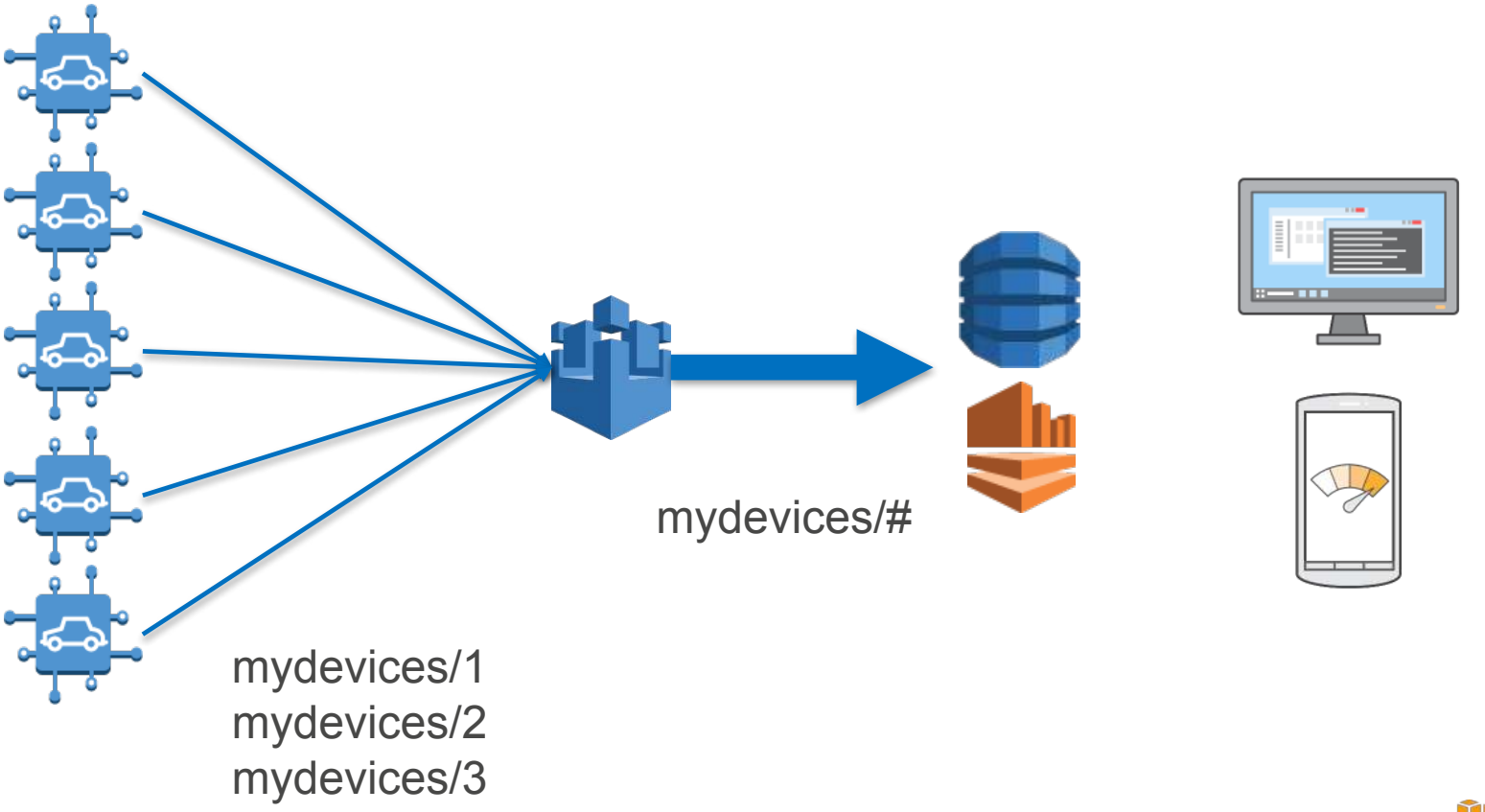
- OASIS standard protocol (v3.1.1)
- Lightweight, pub-sub, transport protocol that is useful for connected devices
- MQTT is used on oil rigs, connected trucks, and many more sensitive and resource-sensitive scenarios.
- Customers have needed to build, maintain and scale a broker to use MQTT with cloud applications

MQTTS vs HTTPS:

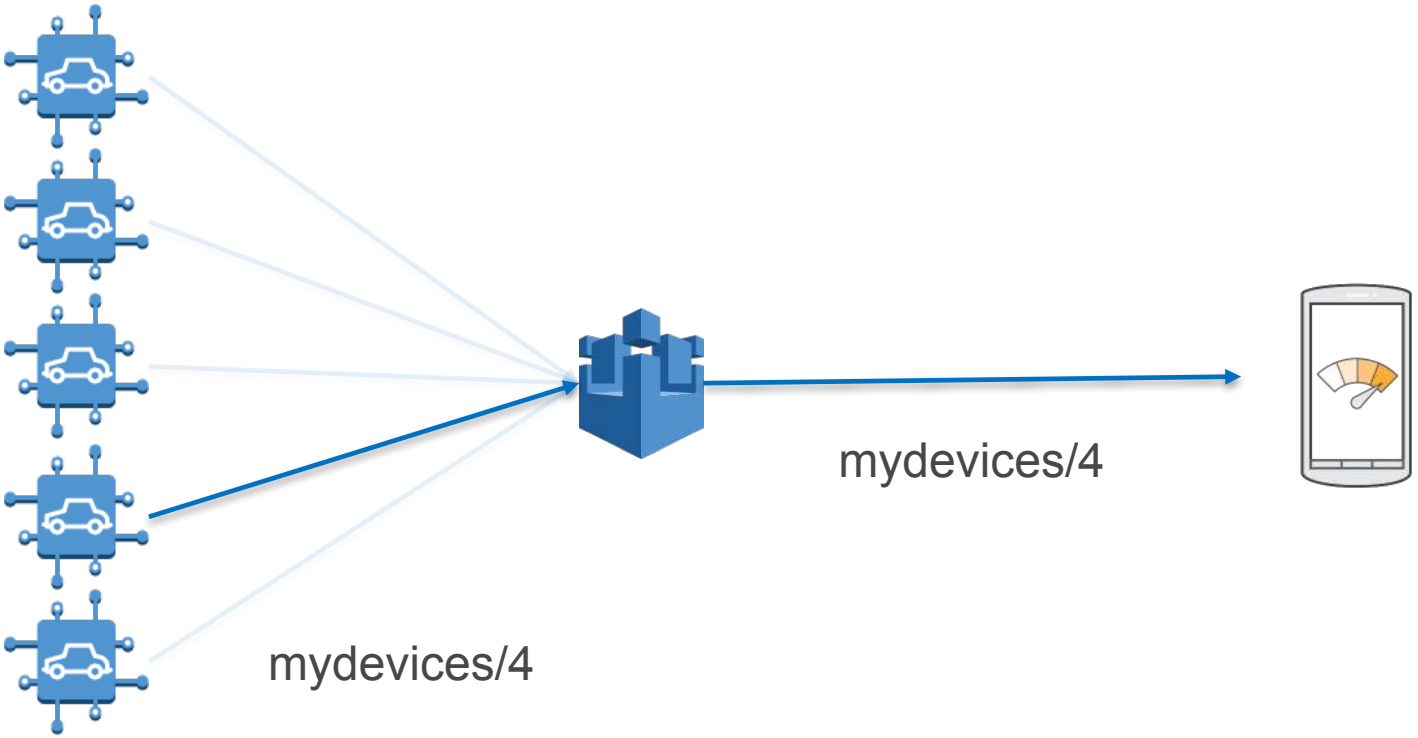
- 93x faster throughput
- 11.89x less battery to send
- 170.9x less battery to receive
- 50% less power to keep connected
- 8x less network overhead

Source: http://stephendnicholas.com/archives/1217

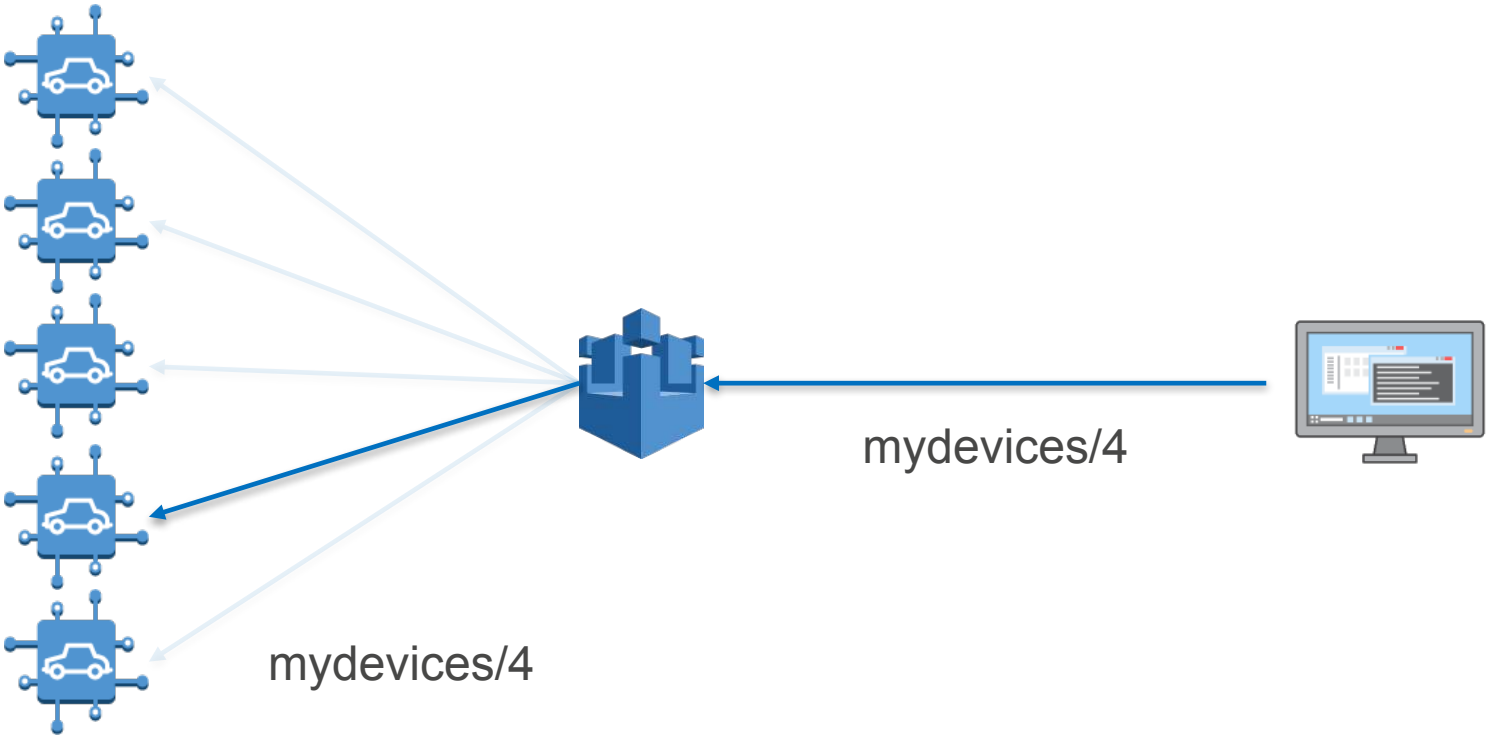
Protocols – MQTT – Use Cases



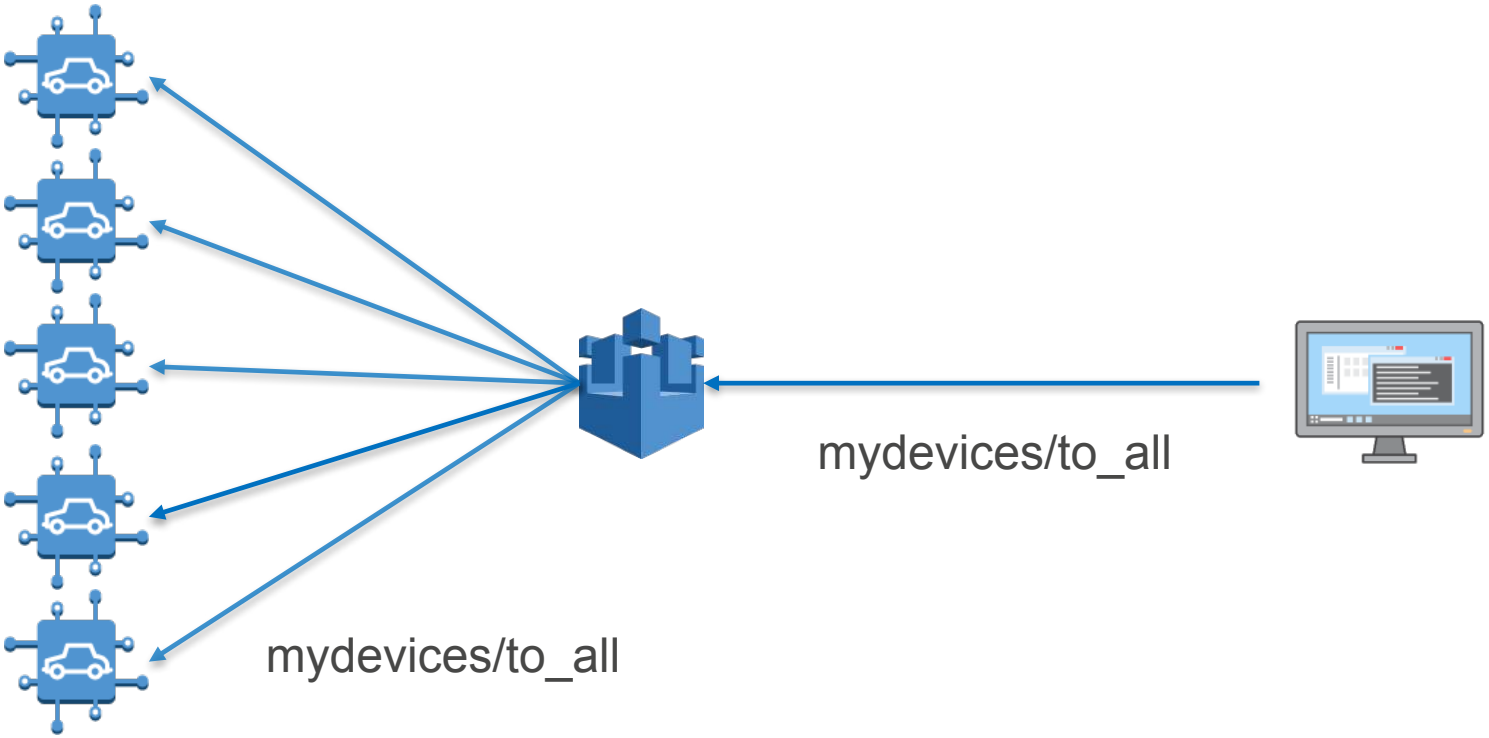
Protocols – MQTT – Use Cases



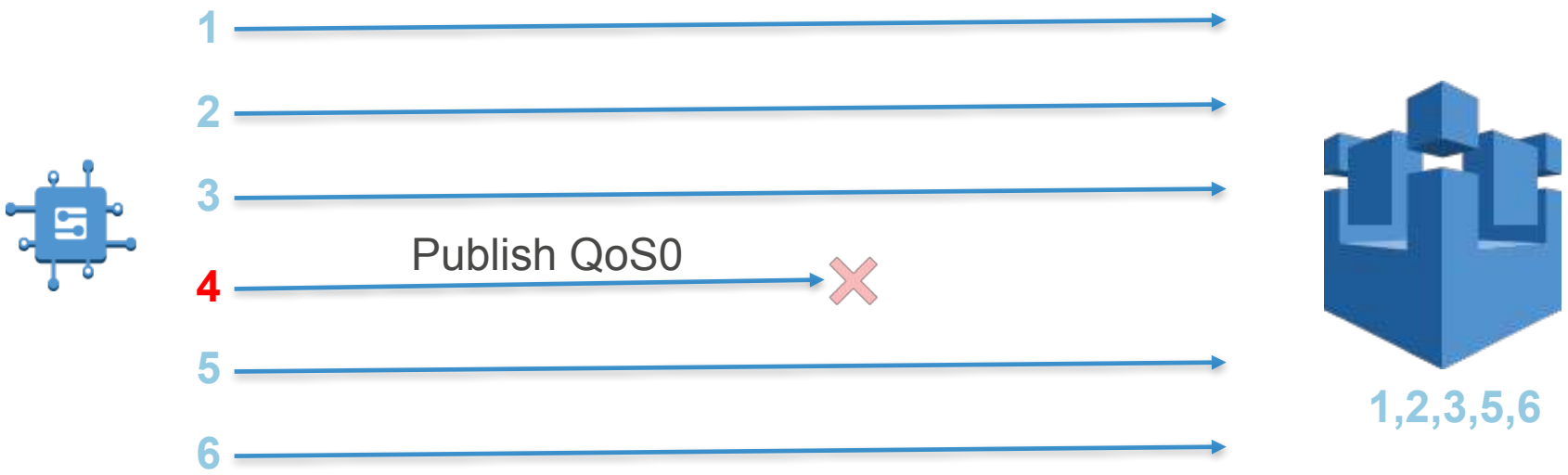
Protocols – MQTT – Use Cases



Protocols – MQTT – Use Cases



Protocols – MQTT – QoS 0 (at most once) (default in AWS IoT)



Protocols – MQTT – QoS 1 (at least once)

1

2

3

PUBLISH QoS1

4

5

PUBLISH QoS1

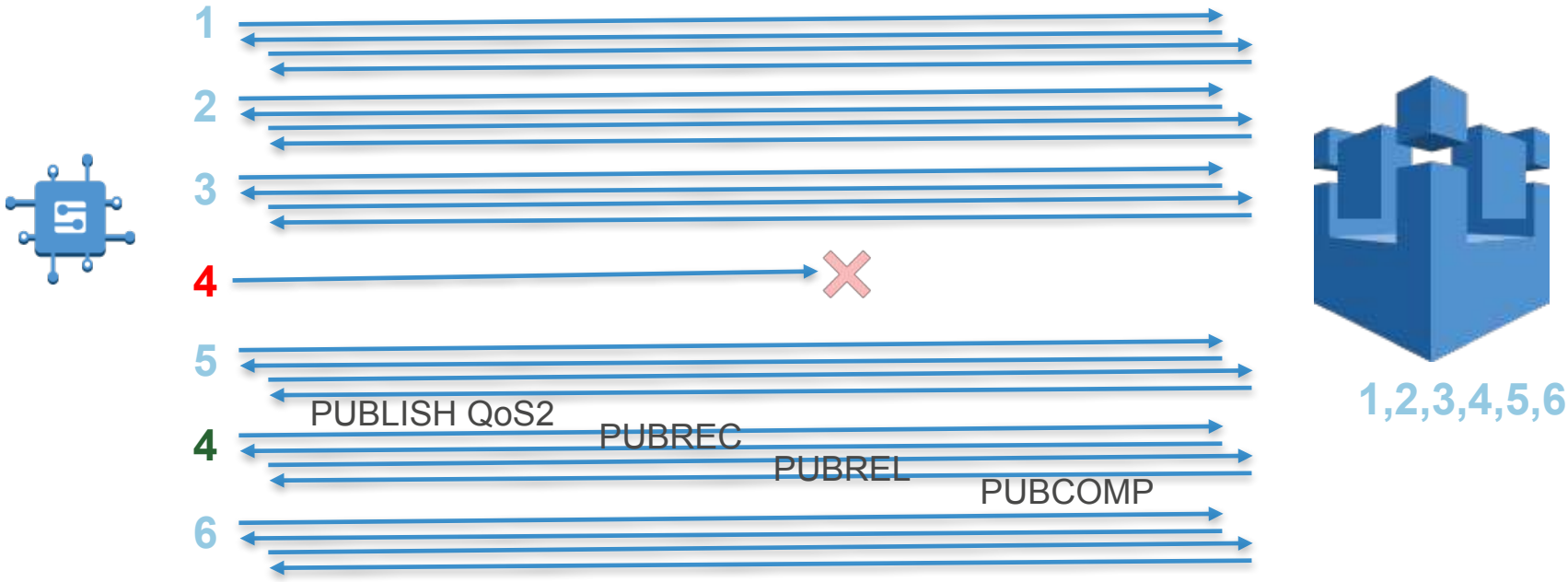
4

PUBACK

6

1,2,3,4,5,6

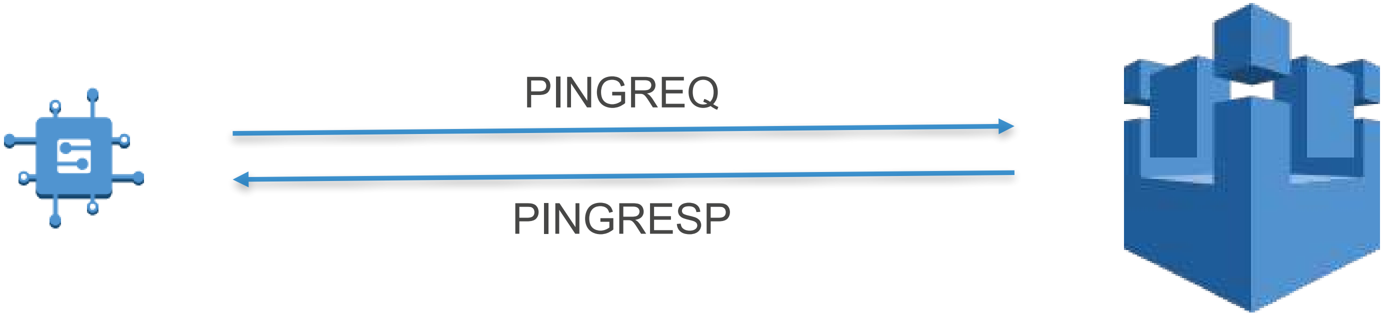
Protocols – MQTT – QoS 2 (only once)
Not available yet in AWS IoT



Protocols – MQTT – Keep Alive

Although TCP/IP in theory notifies you when a socket breaks, in practice, particularly on things like mobile and satellite links, which often “fake” TCP over the air and put headers back on at each end, it’s quite possible for a TCP session to “black hole”, i.e. it appears to be open still, but in fact is just dumping anything you write to it onto the floor.

Andy Stanford-Clark on the topic “***Why is the keep-alive needed?***“ *



* https://groups.google.com/forum/#!msg/mqtt/zRqd8JbY4oM/XrMwIQ5TU0EJ

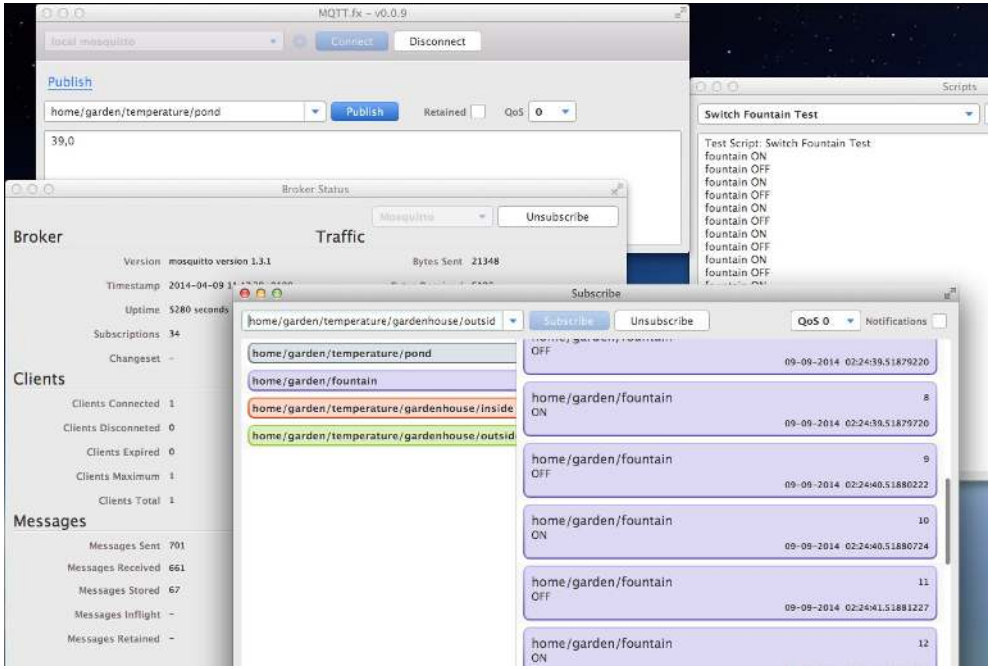
amazon web services

Live POC Time !

amazon web services

First MQTT Pub/Sub

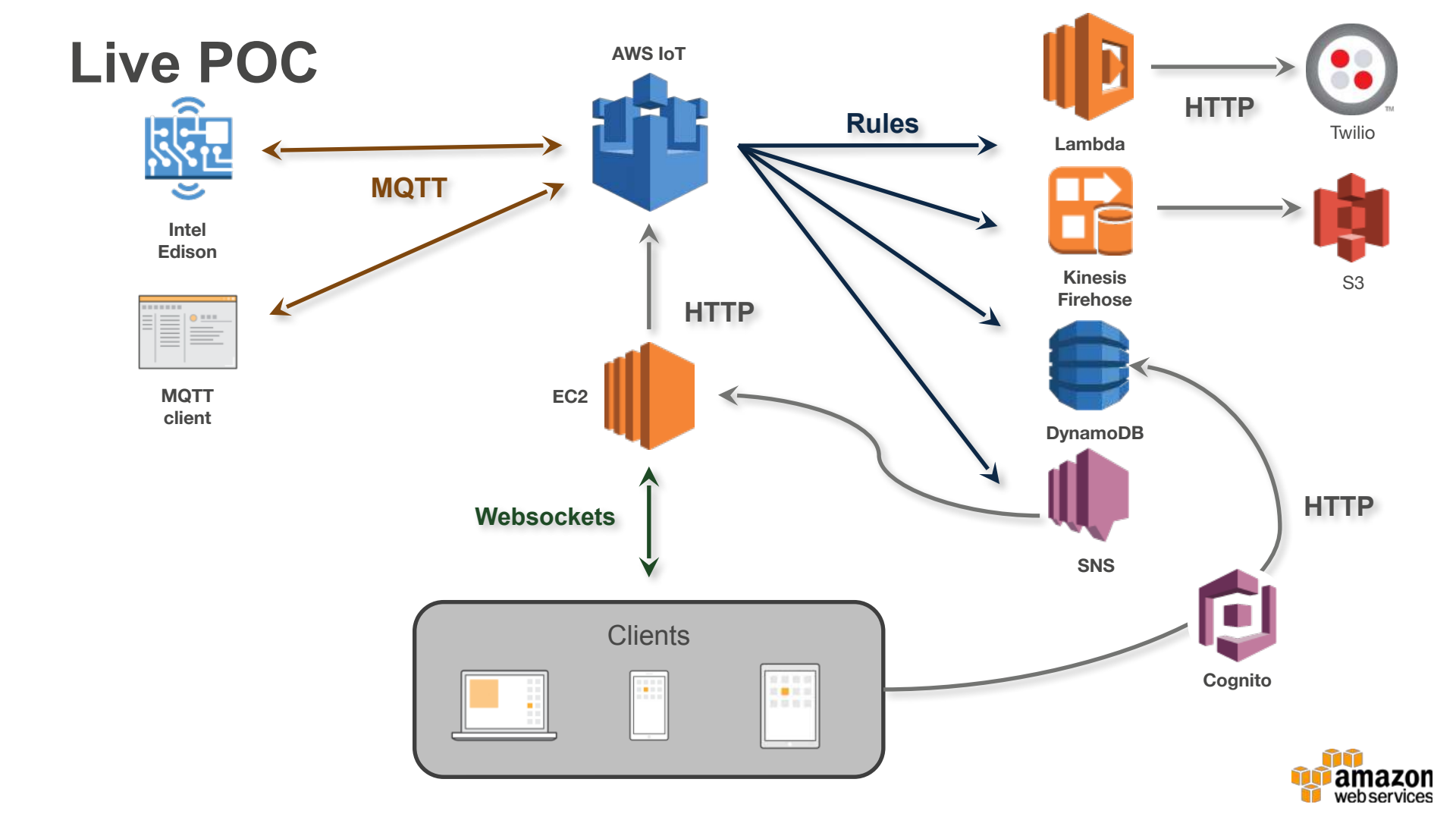
http://mqttfx.jfx4ee.org/



MQTT.fx is a *MQTT* Client written in Java based on Eclipse Paho

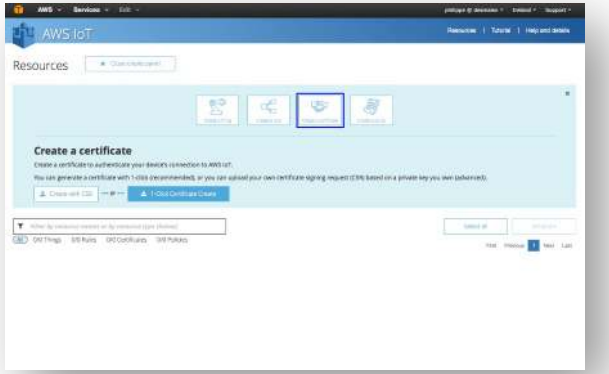
amazon web services

Live POC

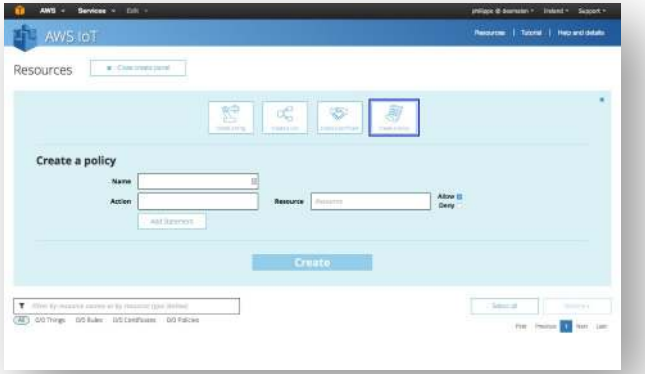

AWS IoT
Intel Edison
MQTT
MQTT client
Rules
Lambda
HTTP
Twilio
Kinesis Firehose
S3
HTTP
EC2
DynamoDB
Websockets
SNS
HTTP
Clients
Cognito

Live POC – AWS IoT configuration

AWS IoT



Create a Certificate

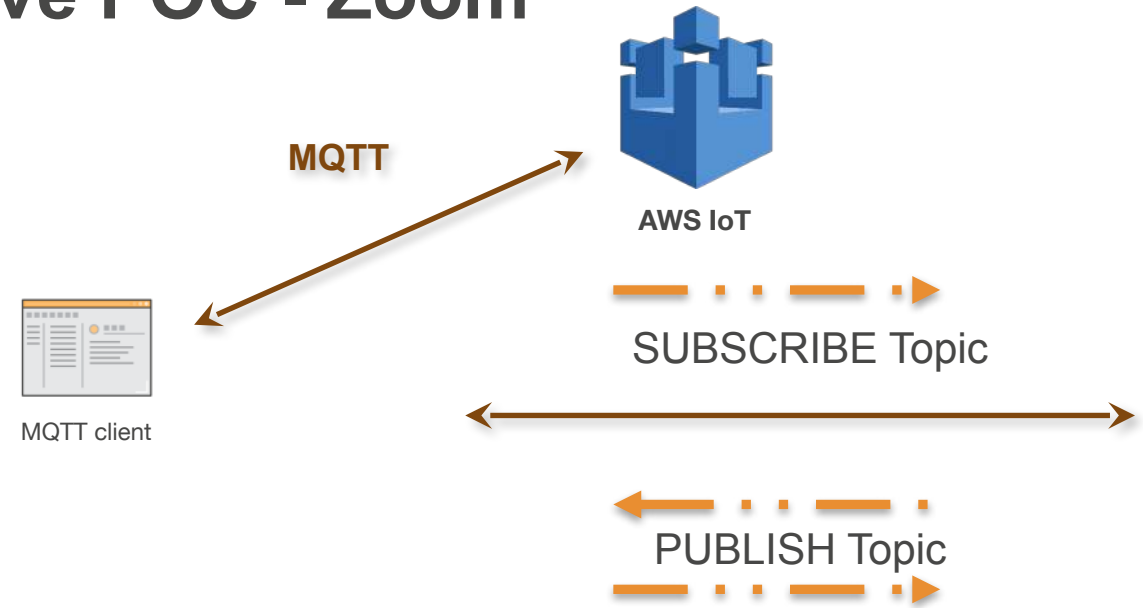


Create a Policy

We'll deep dive on the « Why » of those concepts in a second step

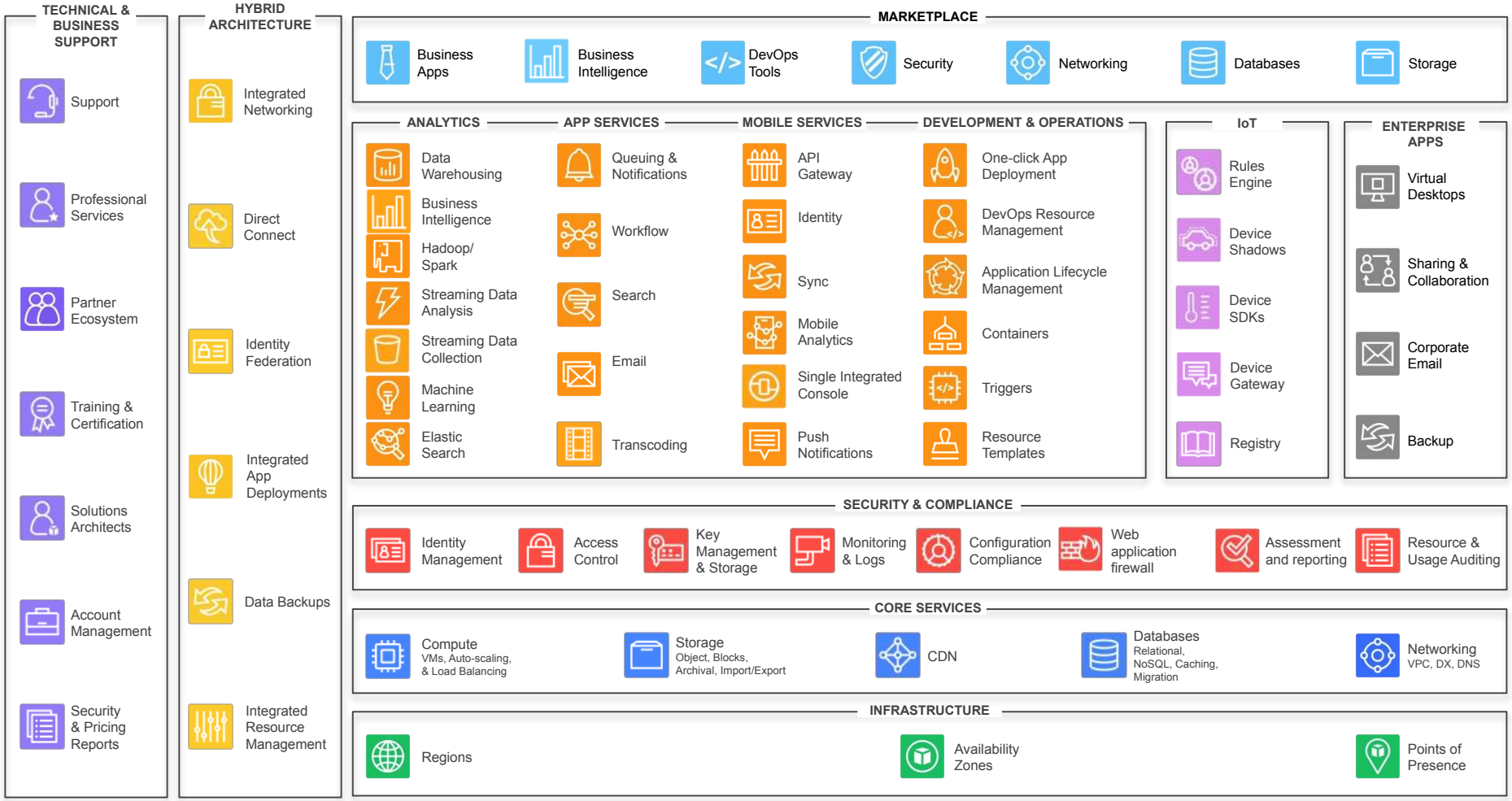
amazon web services

Live POC - Zoom

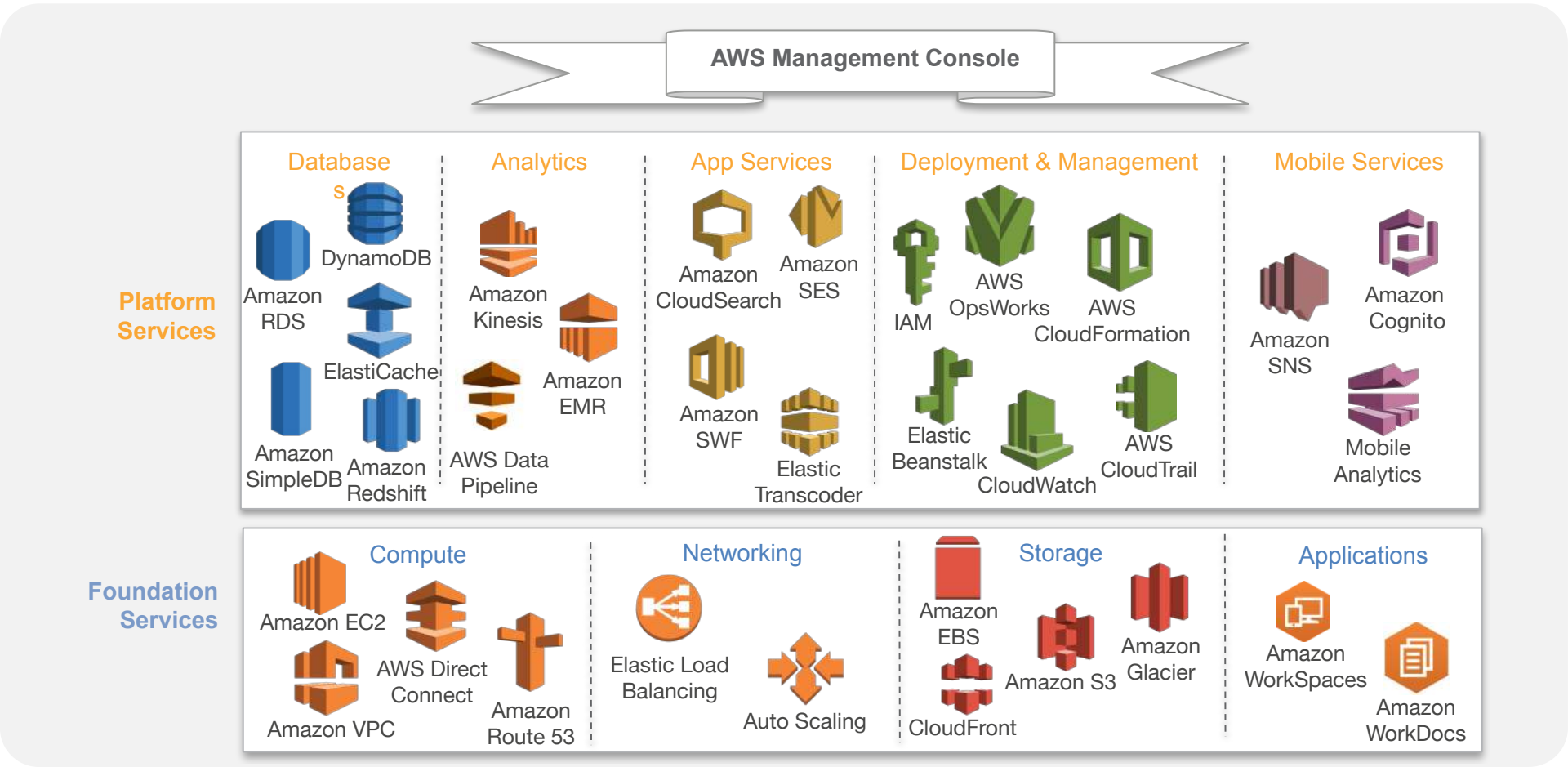


AWS Basics

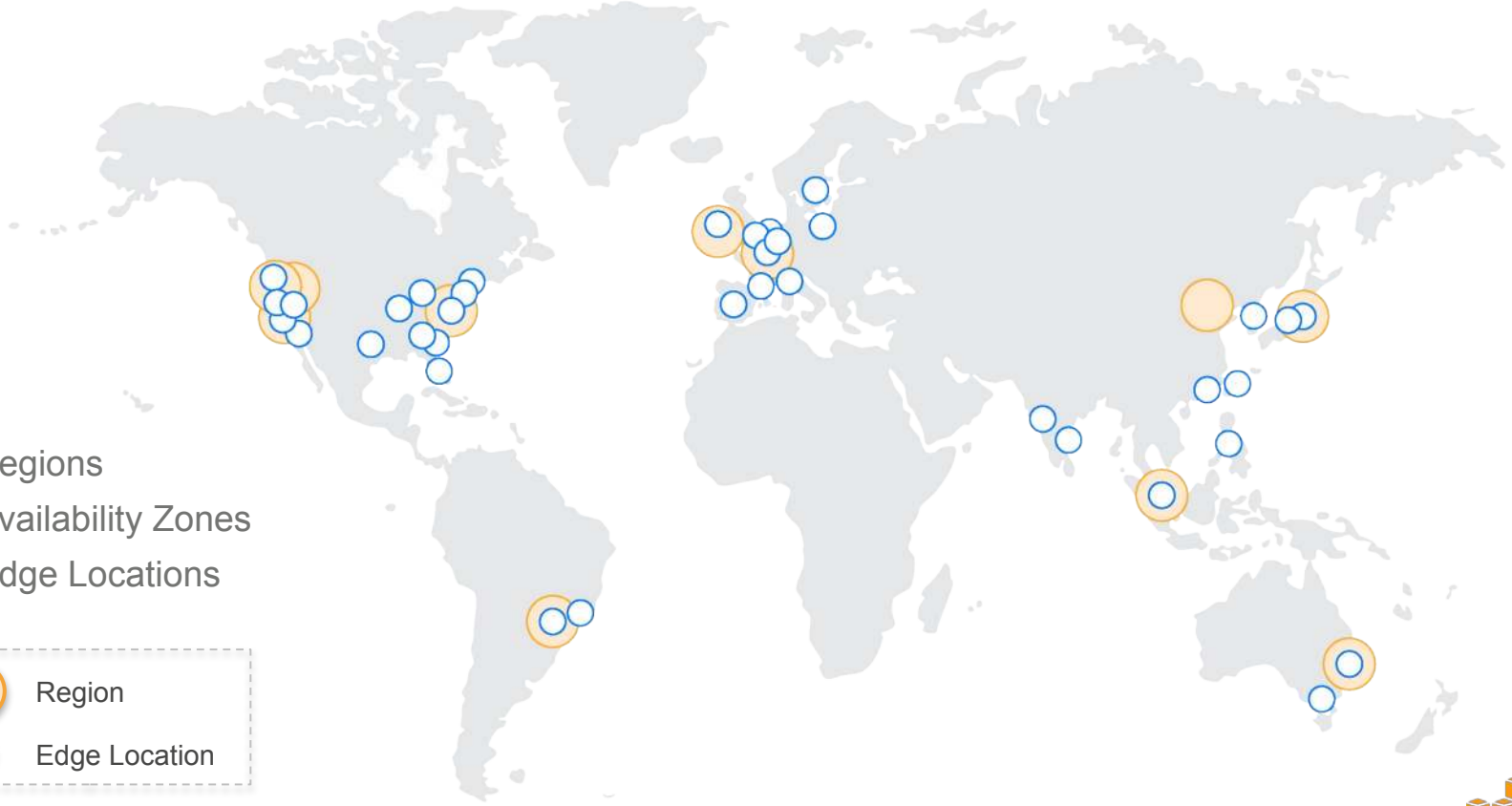
amazon web services


TECHNICAL & BUSINESS SUPPORT
Support
Professional Services
Partner Ecosystem
Training & Certification
Solutions Architects
Account Management
Security & Pricing Reports
HYBRID ARCHITECTURE
Integrated Networking
Direct Connect
Identity Federation
Integrated App Deployments
Data Backups
Integrated Resource Management
MARKETPLACE
Business Apps
Business Intelligence
DevOps Tools
Security
Networking
Databases
Storage
ANALYTICS
Data Warehousing
Business Intelligence
Hadoop/ Spark
Streaming Data Analysis
Streaming Data Collection
Machine Learning
Elastic Search
APP SERVICES
Queuing & Notifications
Workflow
Search
Email
Transcoding
MOBILE SERVICES
API Gateway
Identity
Sync
Mobile Analytics
Single Integrated Console
Push Notifications
DEVELOPMENT & OPERATIONS
One-click App Deployment
DevOps Resource Management
Application Lifecycle Management
Containers
Triggers
Resource Templates
IoT
Rules Engine
Device Shadows
Device SDKs
Device Gateway
Registry
ENTERPRISE APPS
Virtual Desktops
Sharing & Collaboration
Corporate Email
Backup
SECURITY & COMPLIANCE
Identity Management
Access Control
Key Management & Storage
Monitoring & Logs
Configuration Compliance
Web application firewall
Assessment and reporting
Resource & Usage Auditing
CORE SERVICES
Compute
VMs, Auto-scaling, & Load Balancing
Storage
Object, Blocks, Archival, Import/Export
CDN
Databases
Relational, NoSQL, Caching, Migration
Networking
VPC, DX, DNS
INFRASTRUCTURE
Regions
Availability Zones
Points of Presence

AWS Services



AWS Global Infrastructure



11 Regions

30 Availability Zones

53 Edge Locations

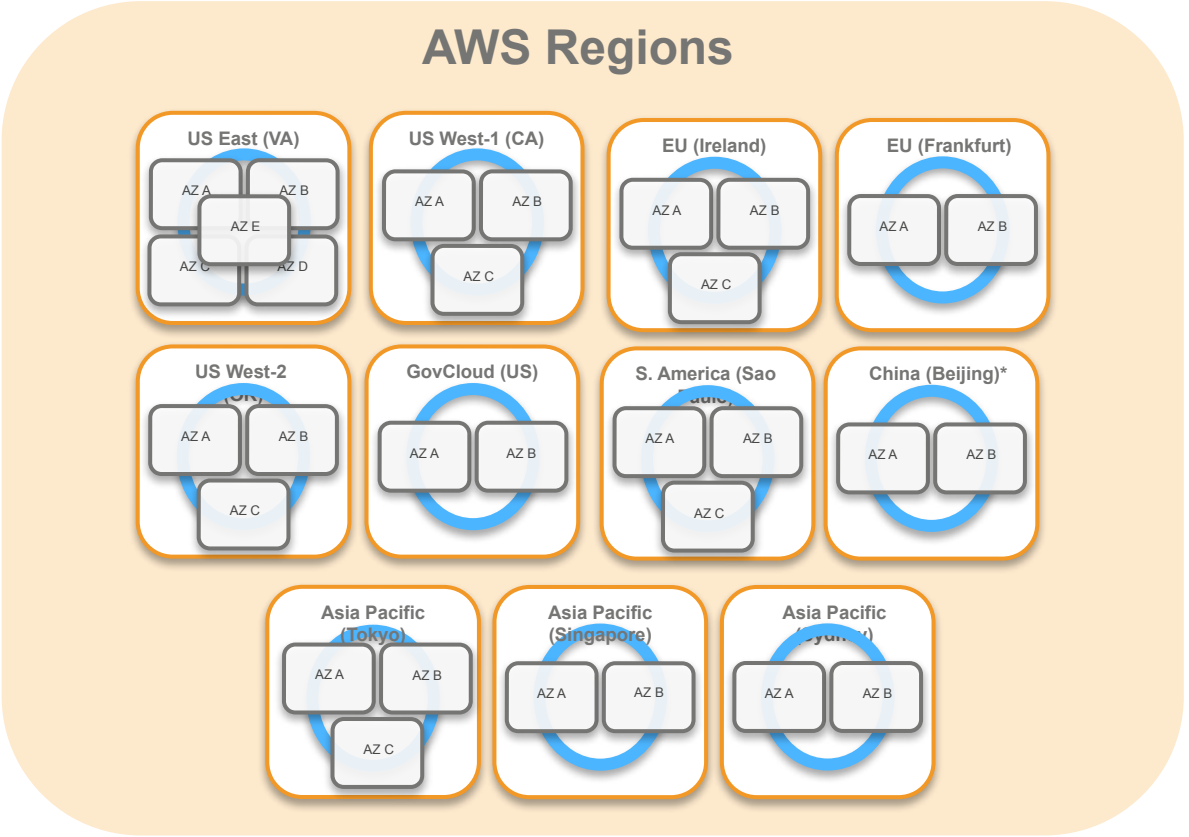
Region

Edge Location

amazon web services

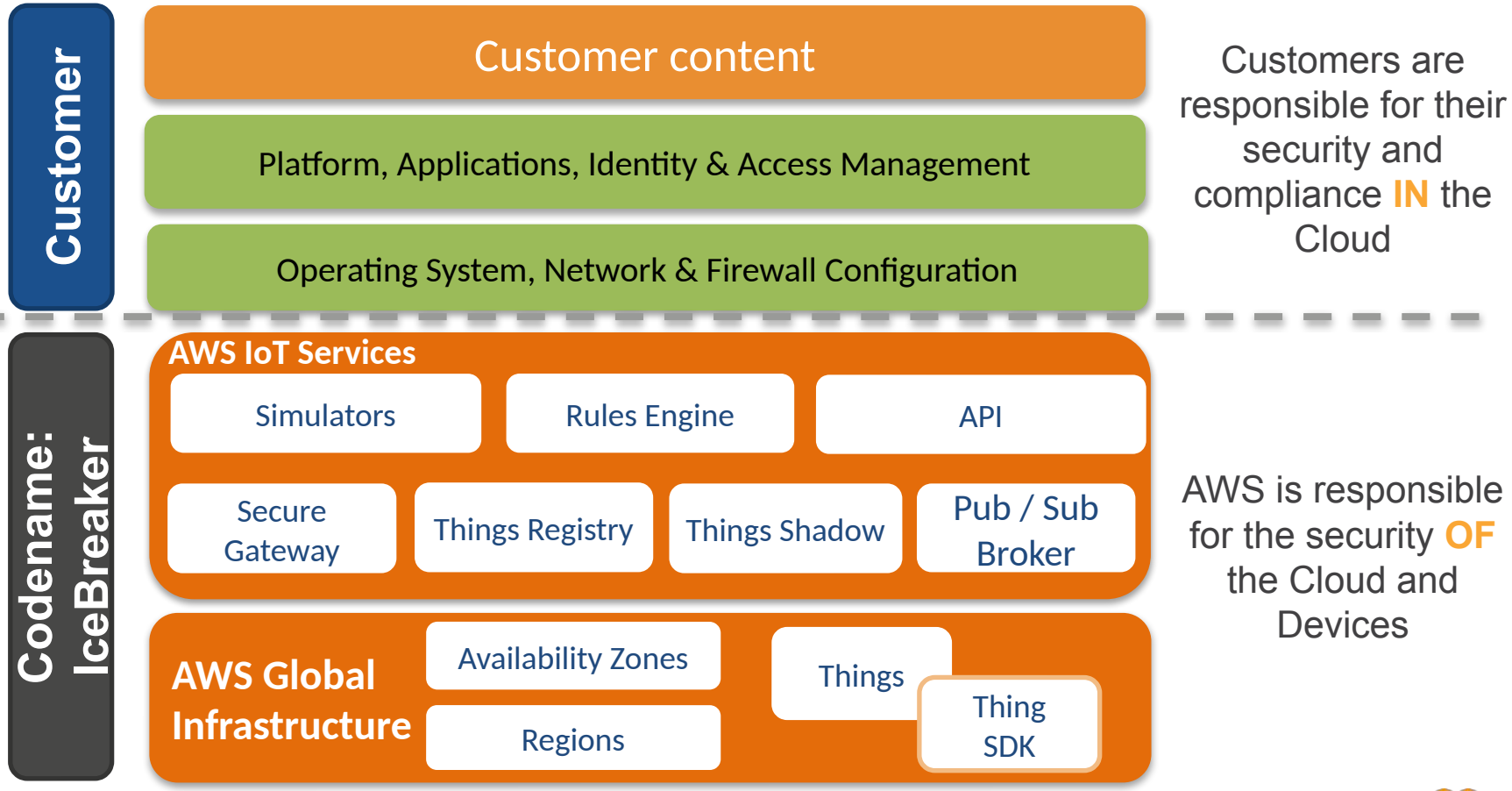
AWS Regions and Availability Zones

Your data stays where you put it



amazon web services

Architecture for IoT



AWS Foundational Services for IoT

Amazon **IoT**

Amazon **Lambda**

Amazon **S3**

Amazon **Kinesis**

Amazon **DynamoDB**

Amazon **Redshift**

Amazon **SNS**

Amazon **SQS**

Value add Services from Amazon.com

amazon echo

amazondash

amazon web services

Amazon
Lambda

No Infrastructure to Manage

Focus on business logic, not infrastructure

Customer uploads code; AWS Lambda handles

Event driven, fully managed compute

All you need is code™

amazon web services

Amazon **S3**

Store anything

Object storage

Scalable

99.999999999% durability

amazon web services

Amazon **Kinesis**

Real-time processing

High throughput; elastic

Easy to use

EMR, S3, Redshift, DynamoDB

Integrations

amazon web services

Amazon
DynamoDB

NoSQL Database

Seamless scalability

Zero administration

Single digit millisecond latency

amazon web services

Amazon
Redshift

Relational data warehouse

Massively parallel

Petabyte scale

Fully managed

$1,000/TB/year

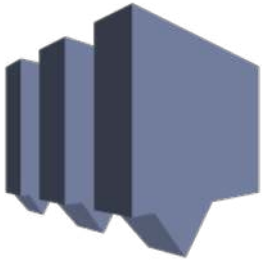
amazon web services

Amazon
SQS

Reliable, highly scalable, queue service for storing messages as they travel between instances

Feature	Details
Reliable	Messages stored redundantly across multiple availability zones
Simple	Simple APIs to send and receive messages
Scalable	Unlimited number of messages
Secure	Authentication of queues to ensure controlled access

amazon web services



Amazon
SNS

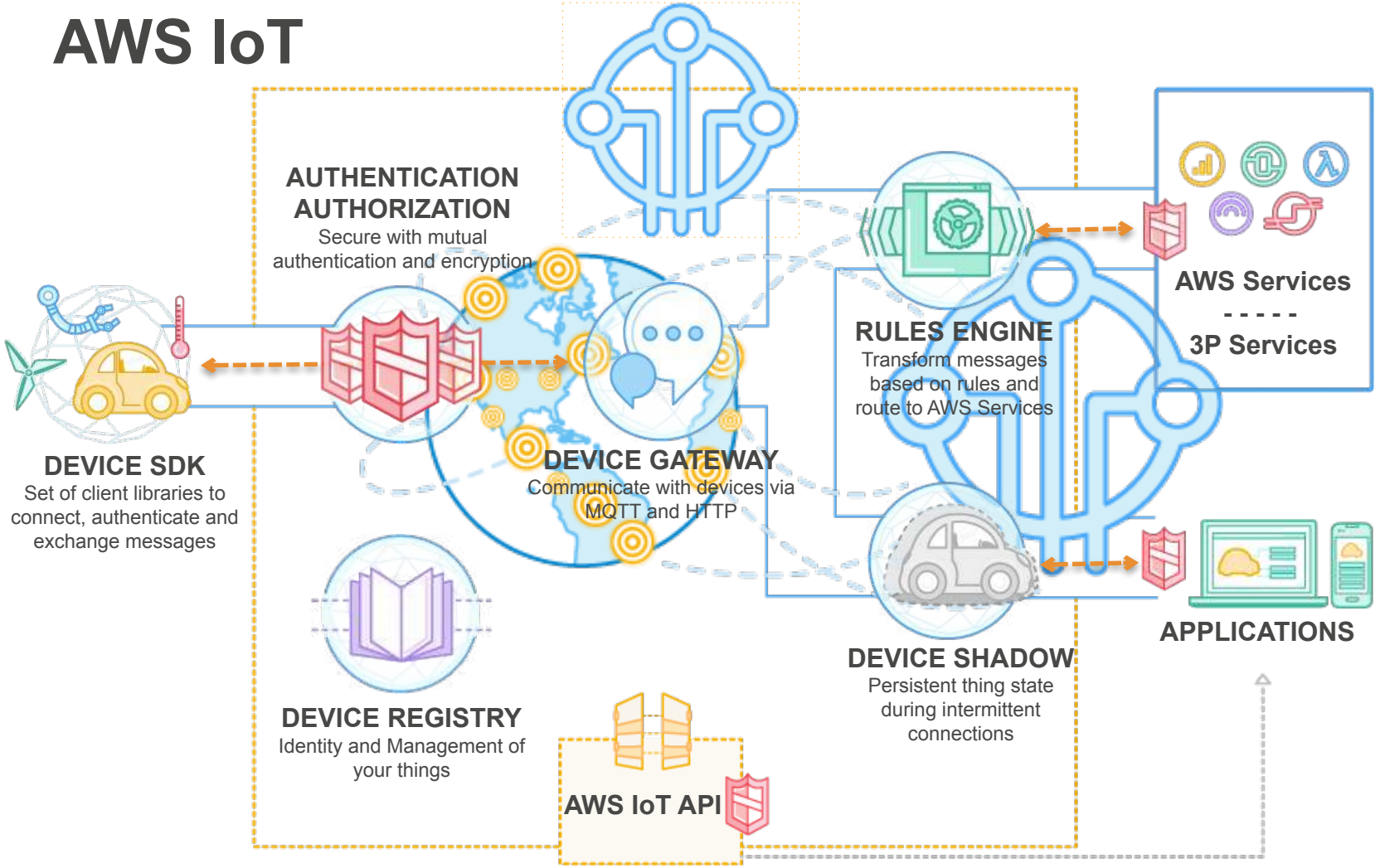
Amazon SNS is a fast, flexible, fully managed pub-sub messaging service. App notification service to send push notifications, email, and SMS messages; or as an enterprise-messaging infrastructure.

amazon web services

Overview of AWS IoT

amazon web services

AWS IoT

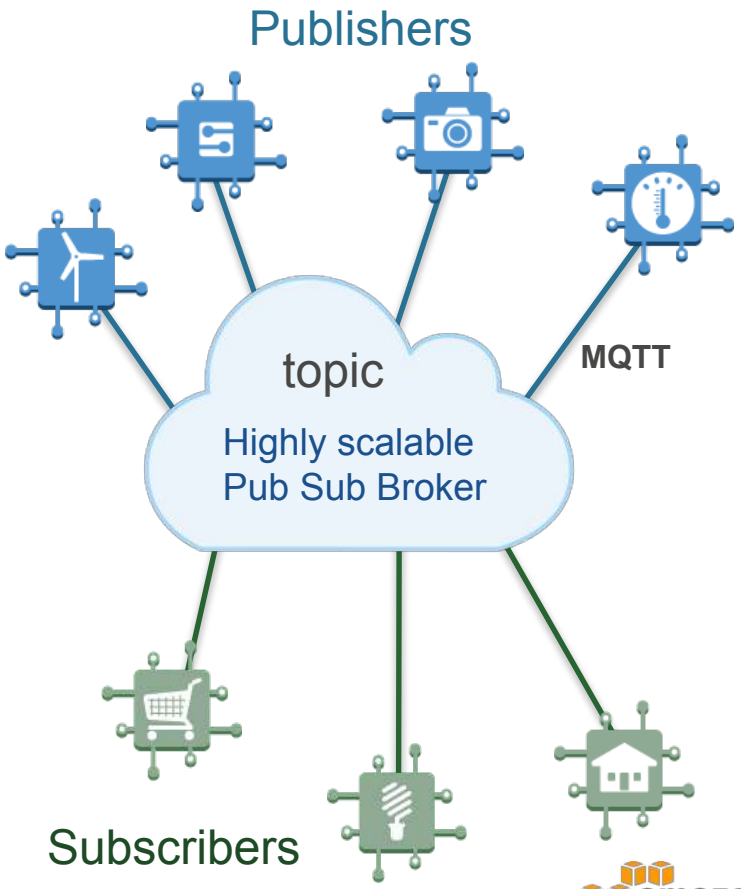

AUTHENTICATION
AUTHORIZATION
Secure with mutual authentication and encryption
RULES ENGINE
Transform messages based on rules and route to AWS Services
AWS Services
- - - - -
3P Services
DEVICE SDK
Set of client libraries to connect, authenticate and exchange messages
DEVICE GATEWAY
Communicate with devices via MQTT and HTTP
APPLICATIONS
DEVICE SHADOW
Persistent thing state during intermittent connections
DEVICE REGISTRY
Identity and Management of your things
AWS IoT API

AWS IoT: Securely Connect Devices

Multi-protocol Message Gateway
Millions of devices and apps can connect over MQTT or HTTP.

Elastic Pub Sub Broker
Go from 1 to 1-billion long-lived connections with zero provisioning

Secure by Default
Connect securely via X509 Certs and TLS v1.2 Client Mutual Auth



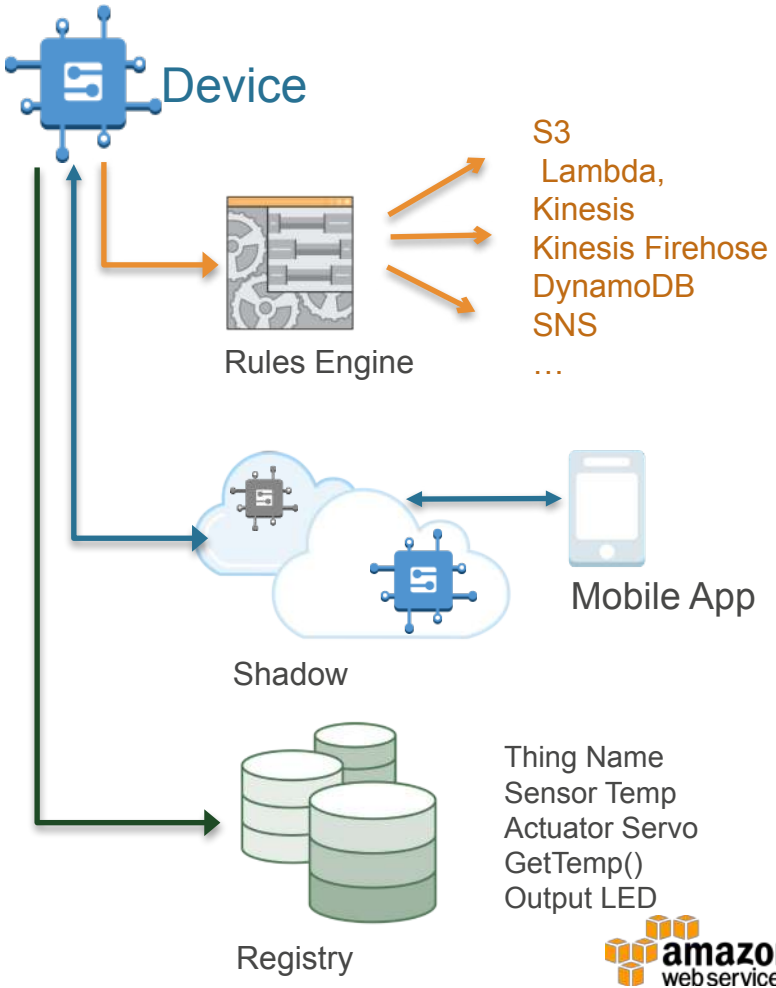
amazon web services

AWS IoT: Front Door to AWS

Device Registry
Cloud alter-ego of a physical device. Persists metadata about the device.

Device Shadows
Apps and devices can access "RESTful" Shadow (state) that is in sync with the device

Rules and Actions
Match patterns and take actions to send data to other AWS services or republish



AWS IoT Rules Engine

- Rules Engine evaluates inbound messages published into AWS IoT, transforms and delivers to the appropriate endpoint based on business rules.
- External endpoints can be reached via AWS Lambda and Amazon Simple Notification Service (SNS).

Actions

Invoke a Lambda function

Put object in an S3 bucket

Insert, Update, Read from a DynamoDB table

Publish to an SNS Topic or Endpoint

Publish to a Kinesis stream /

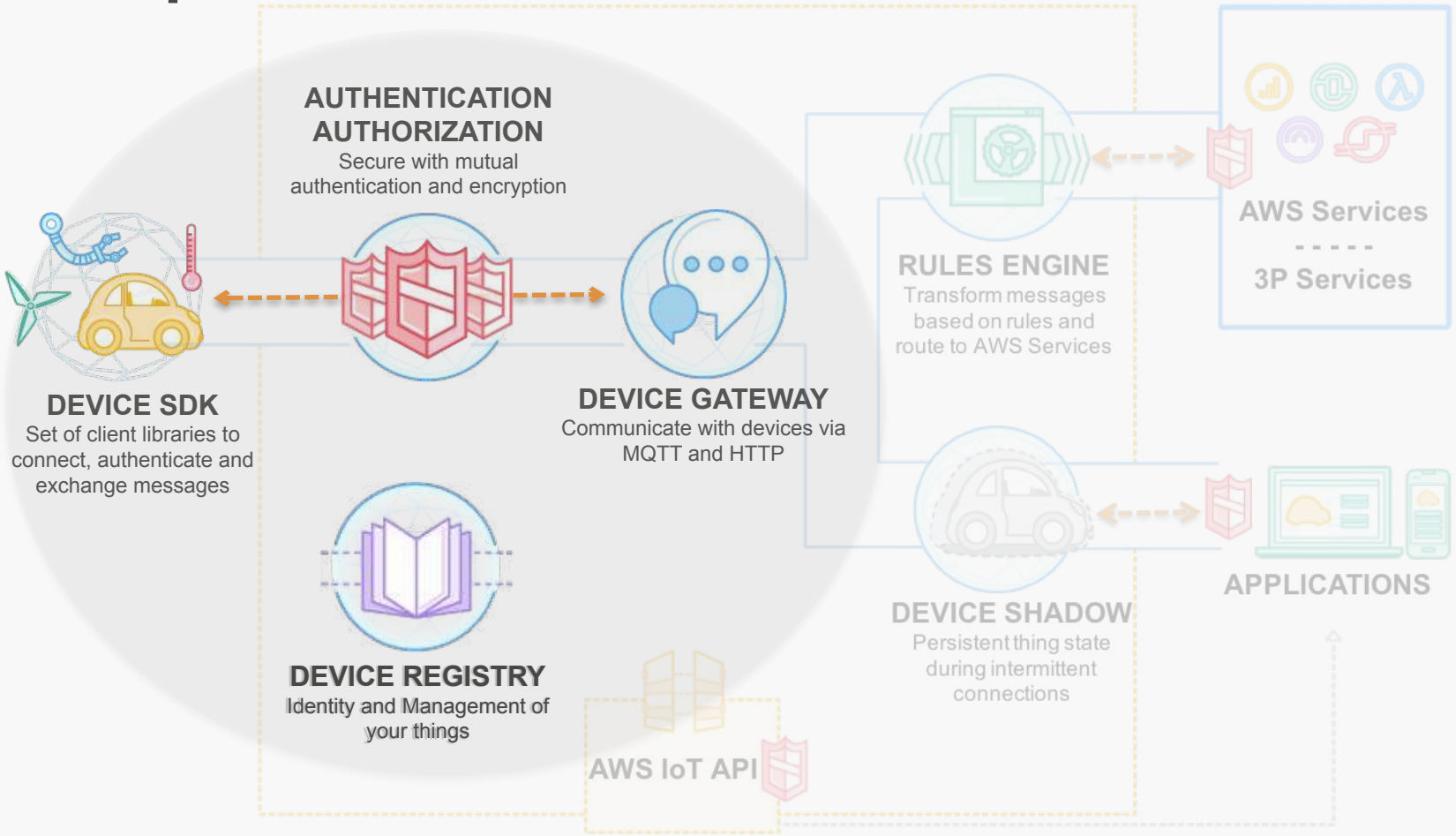
Amazon Kinesis Firehose

Republish to AWS IoT

Deep dive on AWS IoT components

amazon web services

Deep dive on AWS IoT



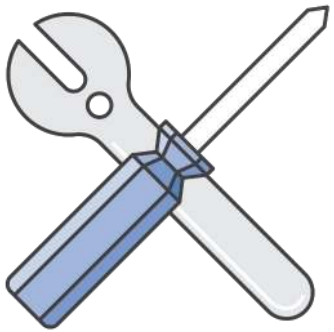
Device SDKs

amazon web services

Getting Started

- How do I get started?

AWS IoT SDKs



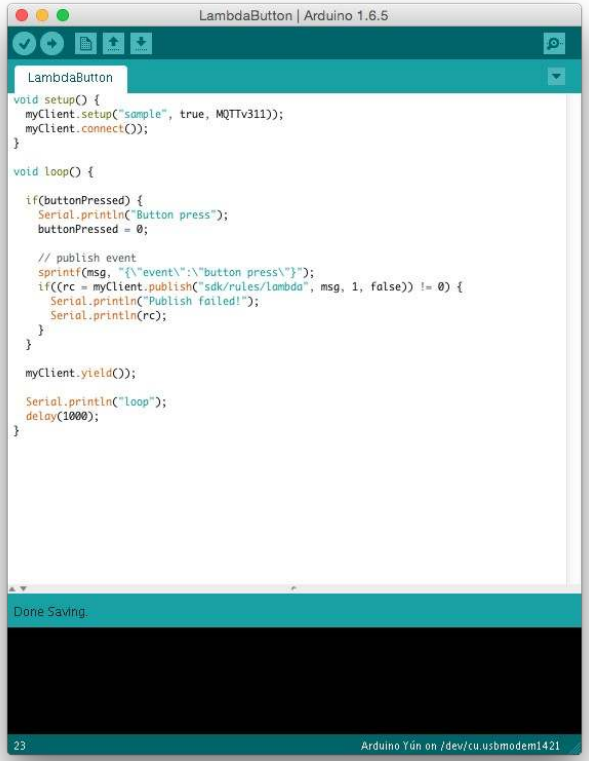
IoT Starter Kits

Getting Started – SDKs

- Arduino (Arduino Yún)
- Node.js (Ideal for Embedded Linux)
- C – Embedded (Ideal for embedded OS)

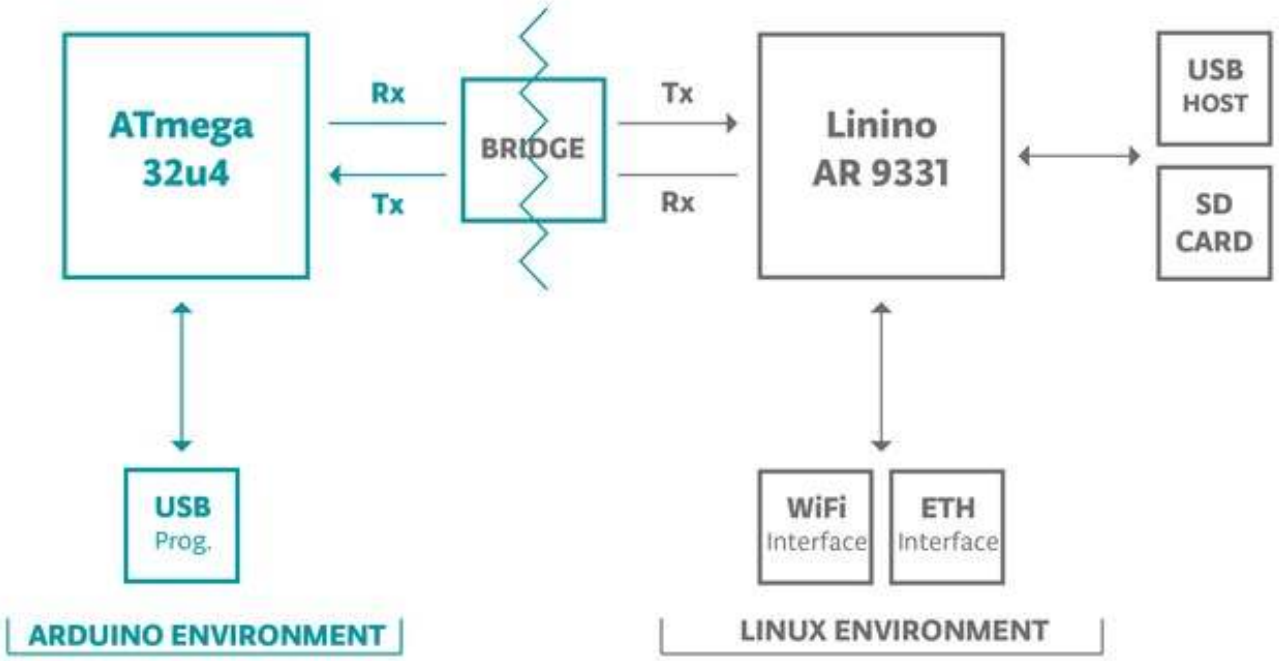
Getting Started – Arduino Yún SDK

- Arduino IDE
- Libraries
- Hardware Ecosystem



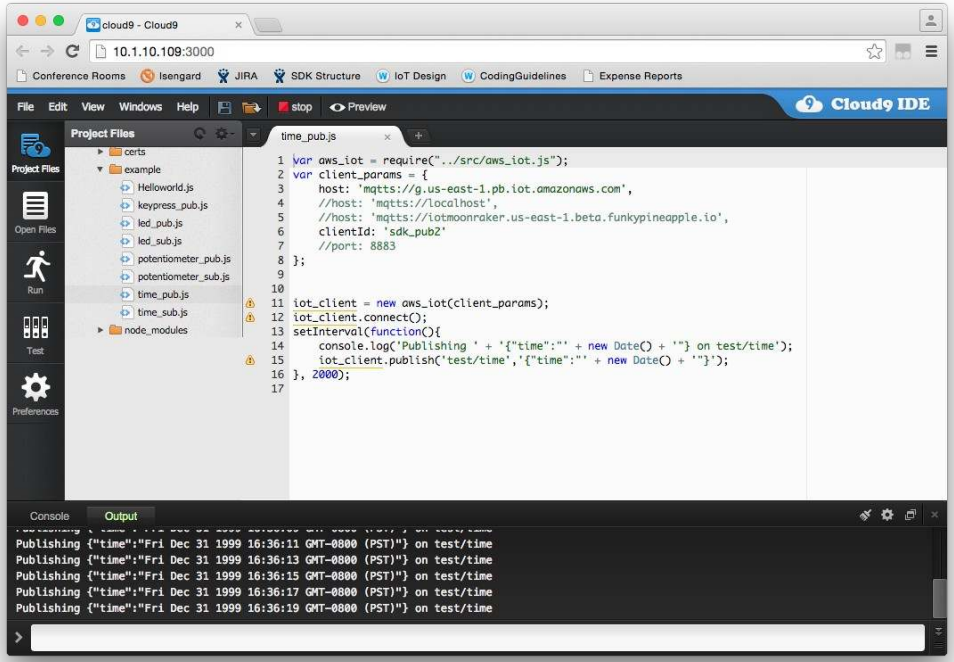
amazon web services

Getting Started – Arduino Yún SDK



Getting Started – Node.js SDK

- Easy install with NPM
- Supports Embedded
- Linux Boards
- High level, but easy access to hardware



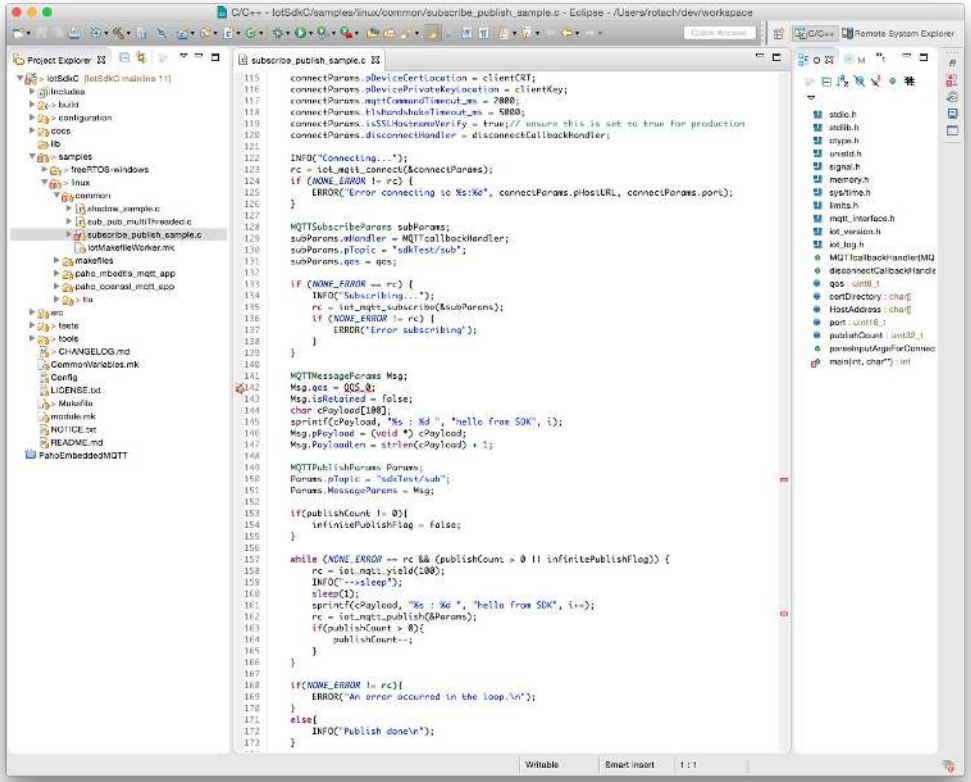
Getting Started – Node.js SDK

```
/**
* example: publish current time every 2 seconds
*
*/

var aws_iot = require("../lib/aws_iot.js");
var client_params = {
        host: 'mqtts://g.us-east-1.pb.iot.amazonaws.com',
        clientId: 'sdk_pub2'
};


iot_client = new aws_iot.mqtt(client_params);
iot_client.connect();

// publish every two seconds
setInterval(function(){
        iot_client.publish('topic/time','current time is ' + new Date());
}, 2000);
```

amazon web services

Getting Started – Embedded C SDK

- Deeply embedded
- Port to your platform
- Delivered as source
- w/ POSIX port



amazon web services

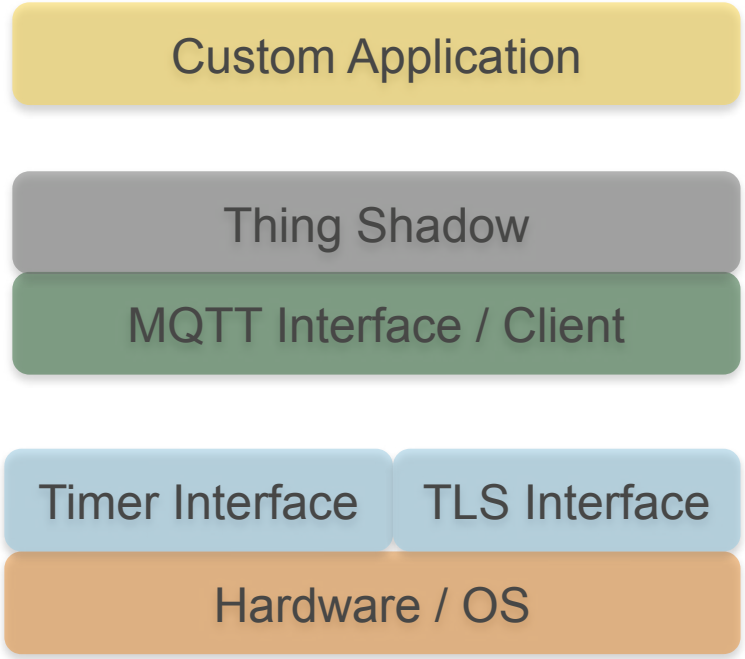
Getting Started –Embedded C SDK

- Memory Conscious

- TLS
- MQTT Client
- Shadow SDK
- Examples

```
ShadowParameters_t sp;
sp.pUniqueThingId = "connectedWindow-1";
sp.pHost = HostAddress;
sp.port = port;
sp.pClientCRT = clientCRT;
sp.pClientKey = clientKey;
sp.pRootCA = rootCA;

INFO("Shadow Init");
rc = iot_shadow_init(&mqttClient);

INFO("Shadow Connect");
rc = iot_shadow_connect(&mqttClient, &sp);

if (NONE_ERROR != rc) {
    ERROR("Shadow Connection Error");
}

rc = iot_shadow_register_delta(&mqttClient, &windowActuator);

if (NONE_ERROR != rc) {
    ERROR("Shadow Register Delta Error");
}

// loop and publish a change in temperature
while (NONE_ERROR == rc) {
    rc = iot_shadow_yield(&mqttClient, 2000);
    sleep(1);
    INFO("\n=========================================================================================
    INFO("On Device: window state %s", windowOpen?"true":"false");
    // increment temperature randomly
    temperature += 1.23f;
    pJsonStringToUpdate = iot_shadow_init_json_document();
    rc = iot_shadow_add_reported(pJsonStringToUpdate, 2, &temperatureHandler, &windowActuat
    if (rc == NONE_ERROR) {
        iot_finalize_json_document(pJsonStringToUpdate);
        INFO("Update Shadow: %s", pJsonStringToUpdate);
        rc = iot_shadow_update(&mqttClient, pJsonStringToUpdate, ShadowUpdateStatusCallback
    }
    INFO("*****************************************************************************
}
```

C SDK – SDK Architecture



- Layered
- Well-defined Interfaces
- Porting Points

Official IoT Starter Kits, Powered by AWS

AVNET BROADCOM

MARVELL

RENESAS

TEXAS INSTRUMENTS

MICROCHIP

intel

MEDIATEK

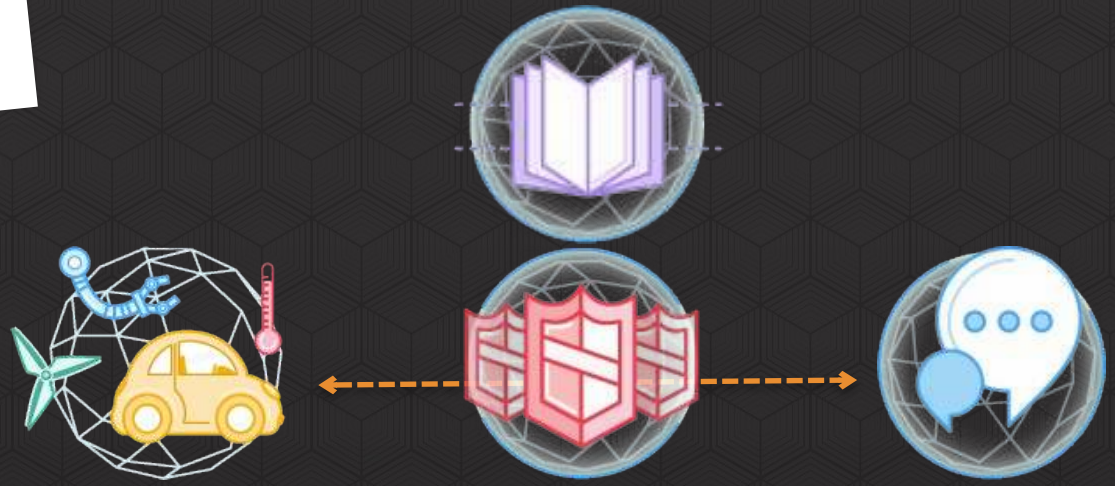
ARROW

QUALCOMM

seeed
Grow the Difference

seeed
Grow the Difference

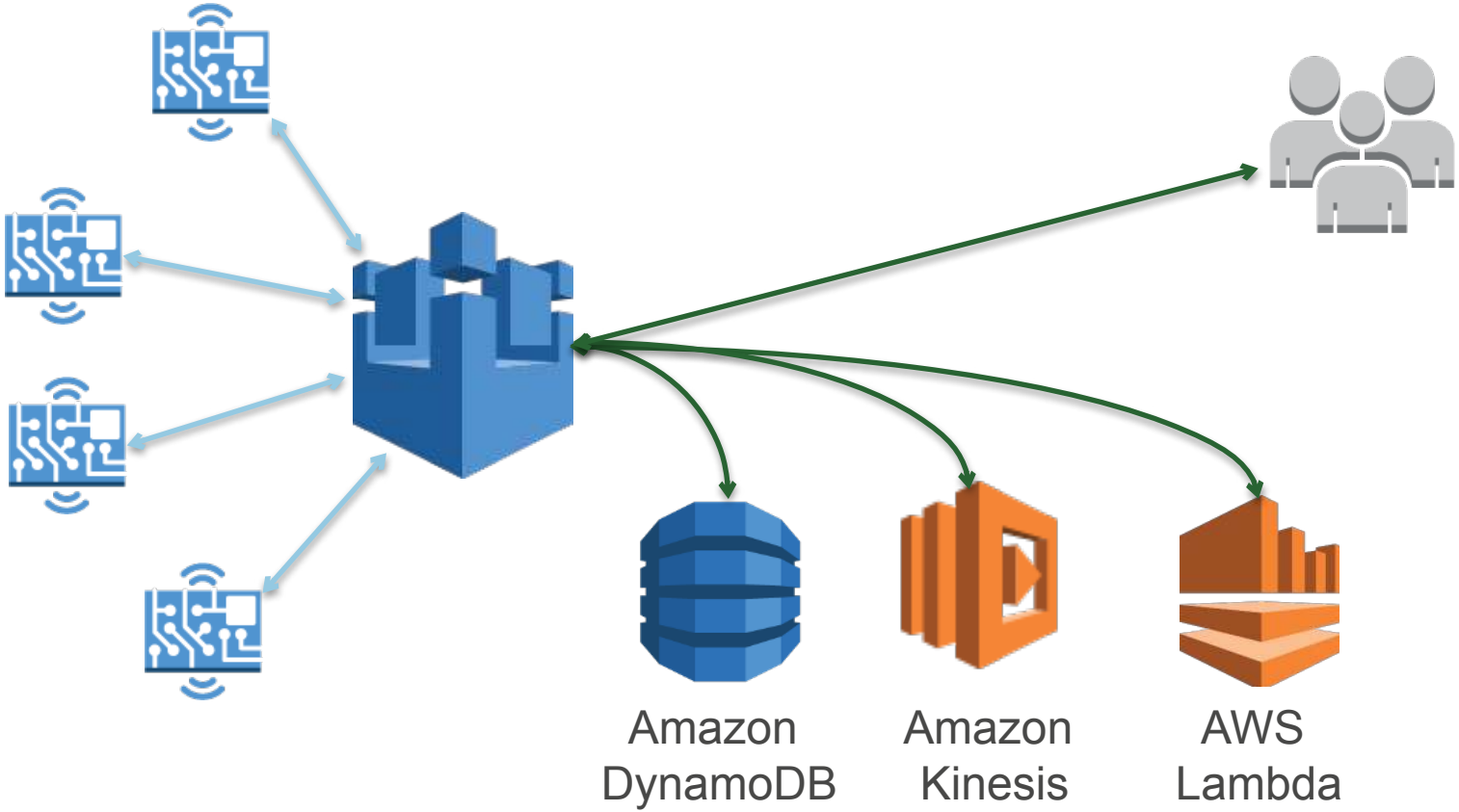
amazon web services



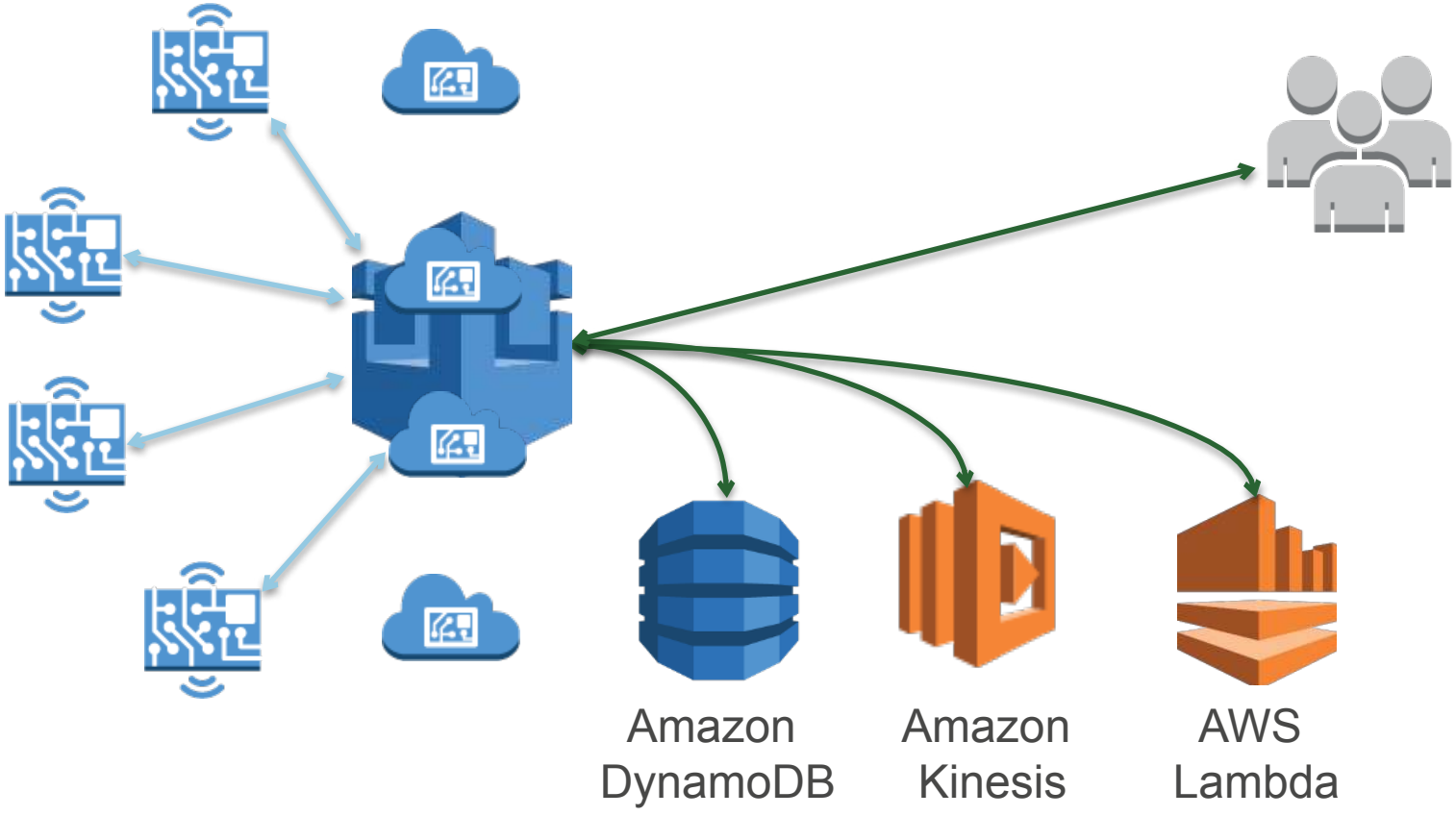
Devices, Broker & Security

amazon web services

The System



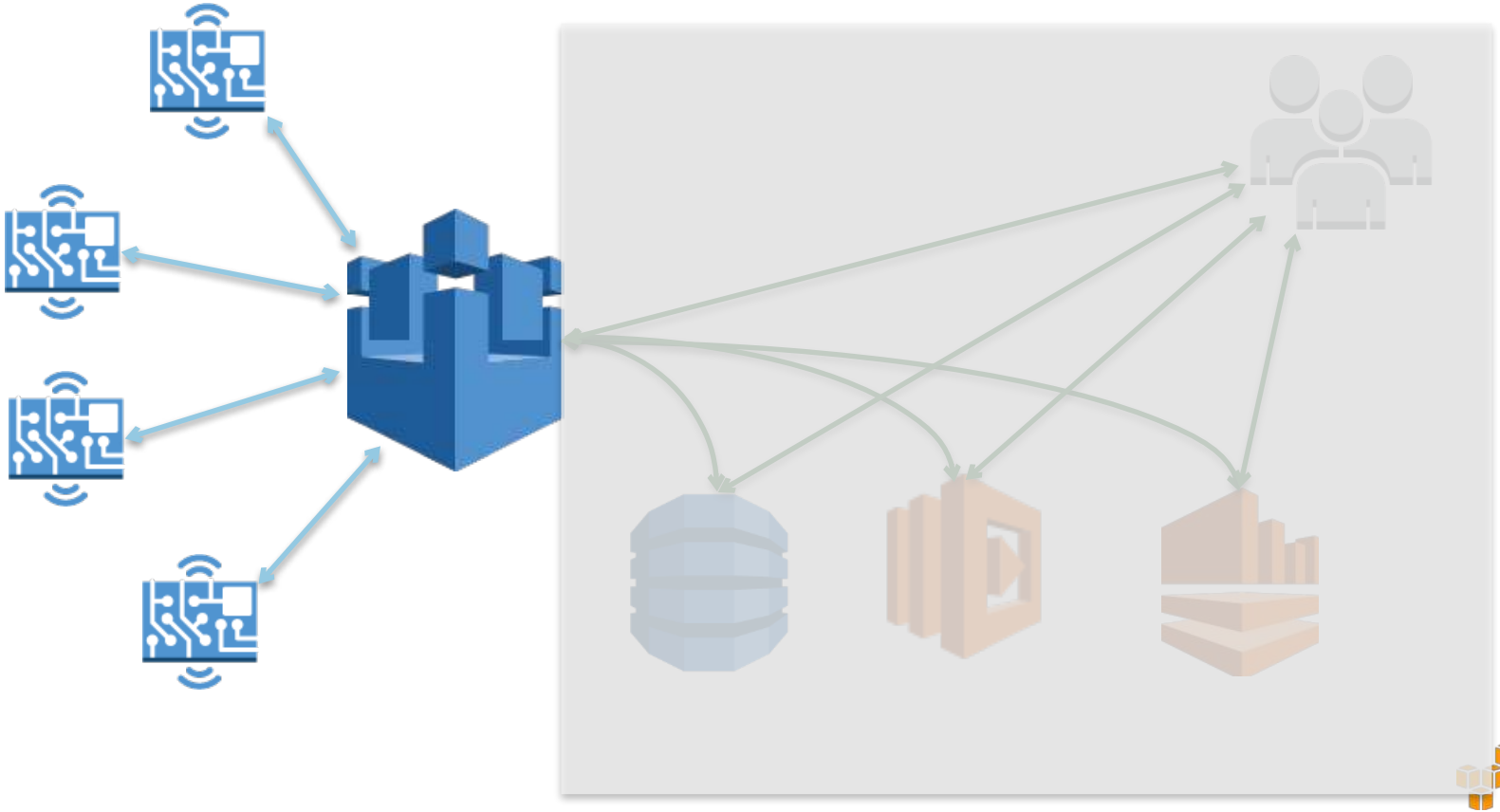
The System



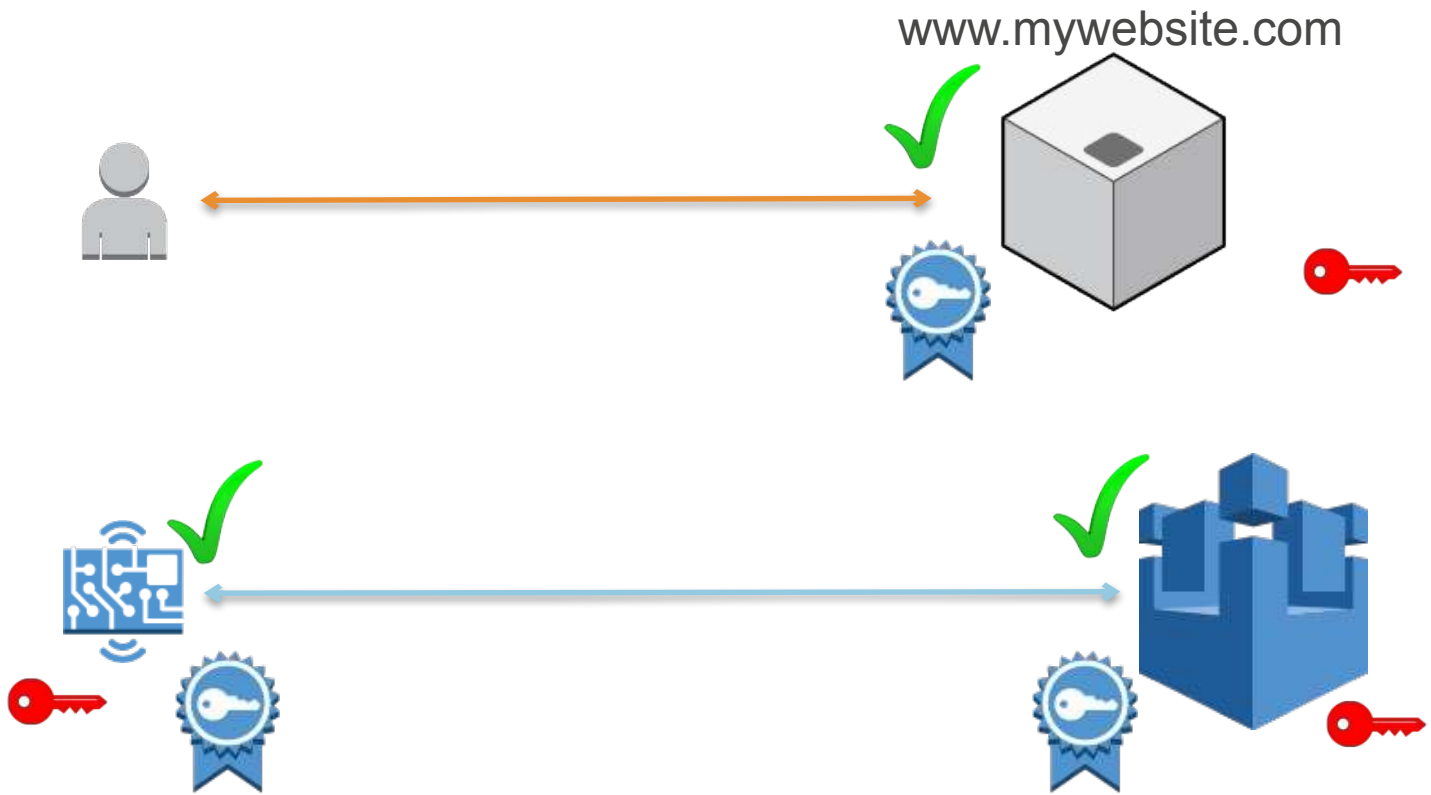
Requirements

- Secure Communications with Things
- Strong Thing Identity
- Fine-grained Authorization for:
- Thing Management
- Pub/Sub Data Access
- AWS Service Access

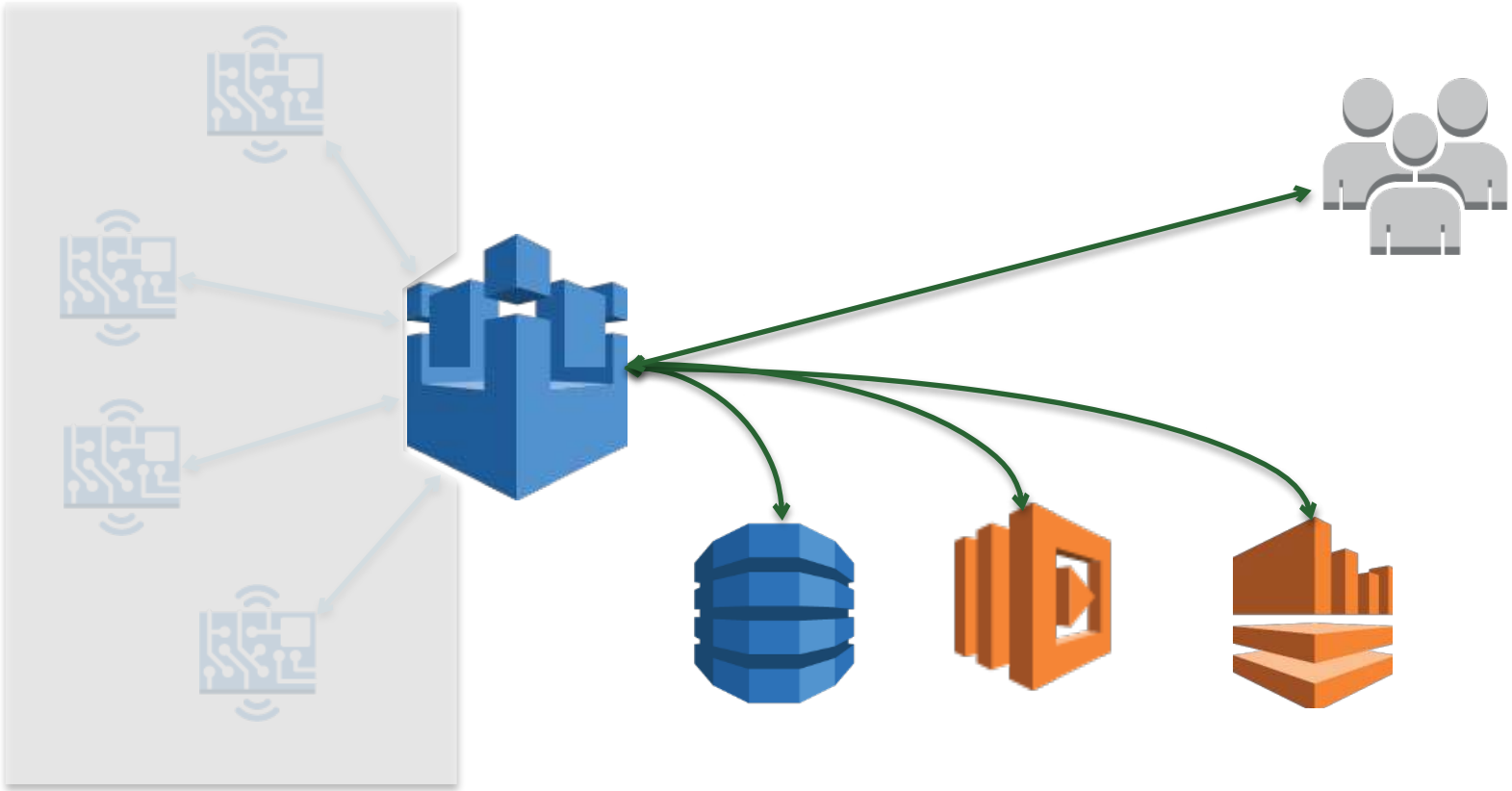
Talking to Things



Mutual Auth TLS



Talking to Non-Things

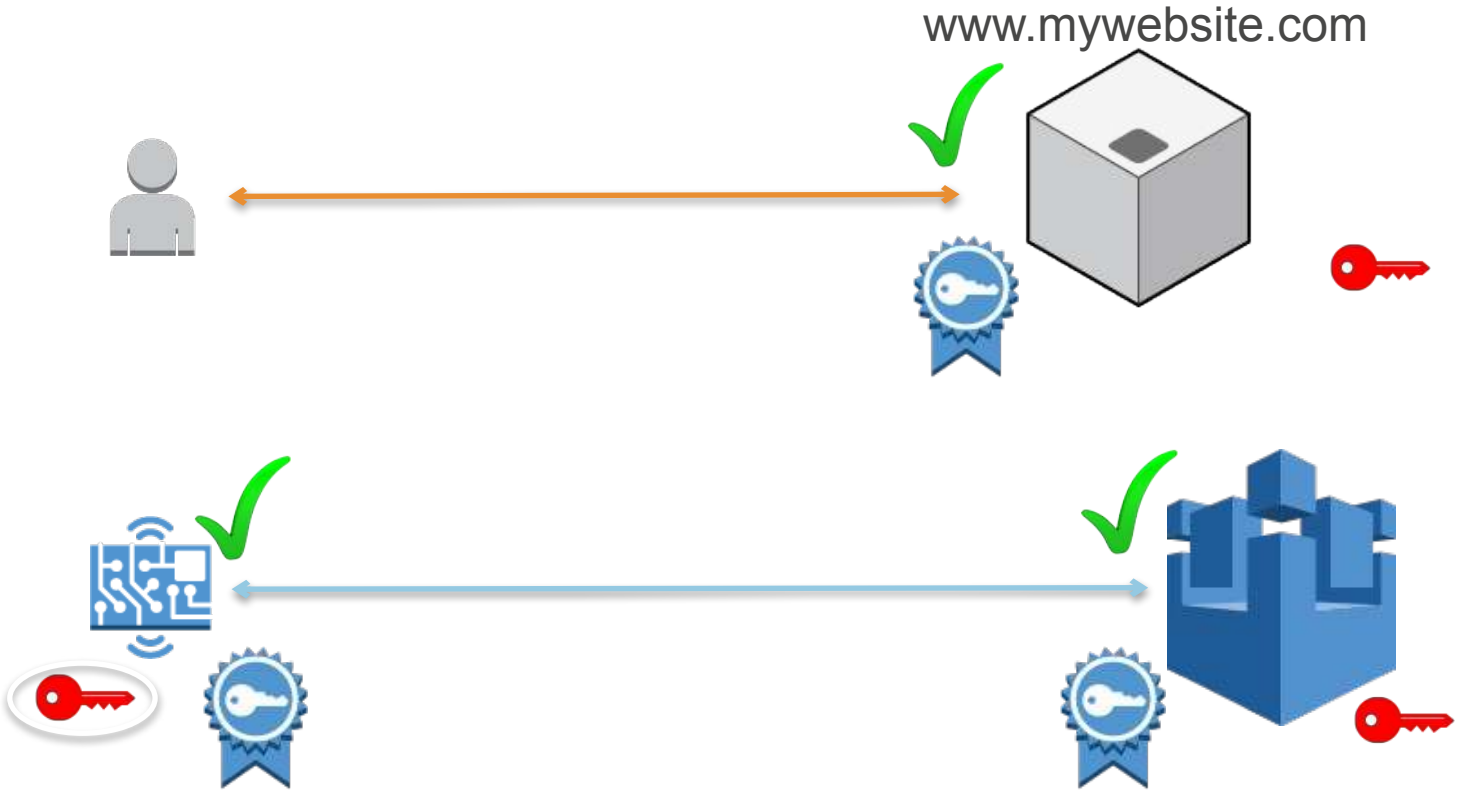


AWS Auth + TLS

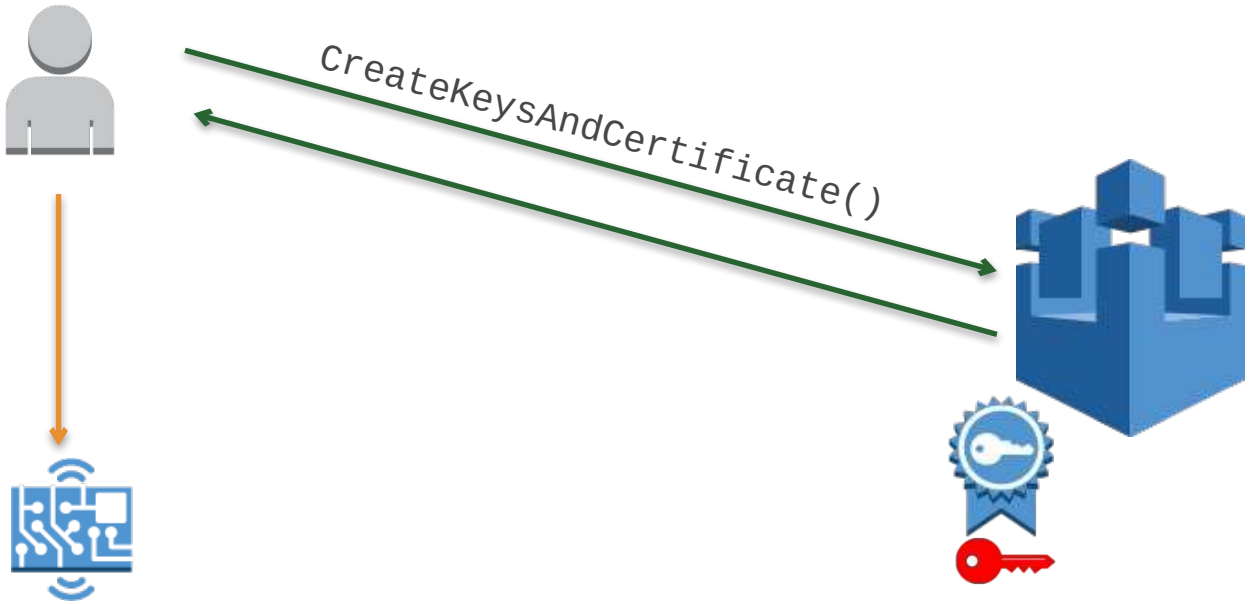
One Service, Two Protocols

	MQTT + Mutual Auth TLS	AWS Auth + HTTPS
Server Auth	TLS + Cert	TLS + Cert
Client Auth	TLS + Cert	AWS API Keys
Confidentiality	TLS	TLS
Protocol	MQTT	HTTP

Back To Certs and Keys



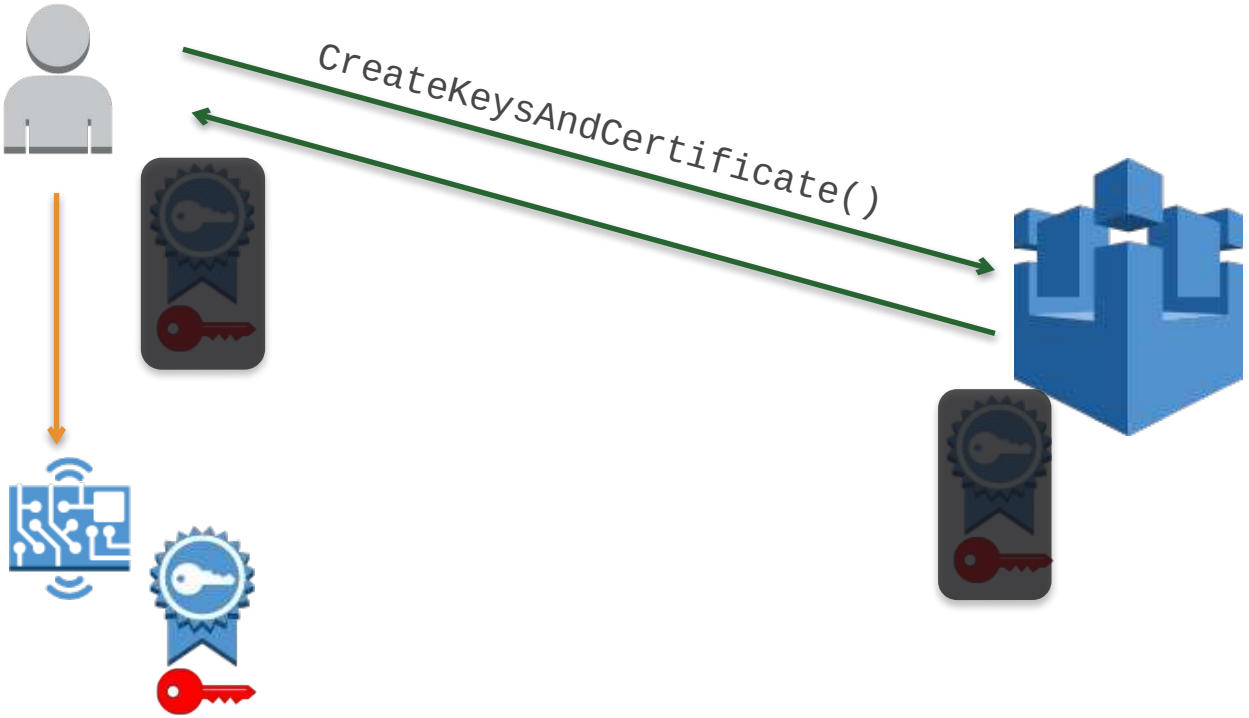
AWS-Generated Keypair



Actual Commands

```
$ aws iot create-keys-and-certificate --set-as-active
{
    "certificateArn":
        "arn:aws:iot:us-east-1:123456972007:cert/d7677b0…SNIP…026d9",
    "certificatePem":
        "-----BEGIN CERTIFICATE-----…SNIP…-----END CERTIFICATE-----",
    "keyPair": {
        "PublicKey":
            "-----BEGIN PUBLIC KEY-----…SNIP…-----END PUBLIC KEY-----",
        "PrivateKey":
            "-----BEGIN RSA PRIVATE KEY-----…SNIP…-----END RSA PRIVATE KEY-----"
    },
    "certificateId":
        "d7677b0…SNIP…026d9"
}
```

AWS-Generated Keypair



Client Generated Keypair

Certificate Signing Request

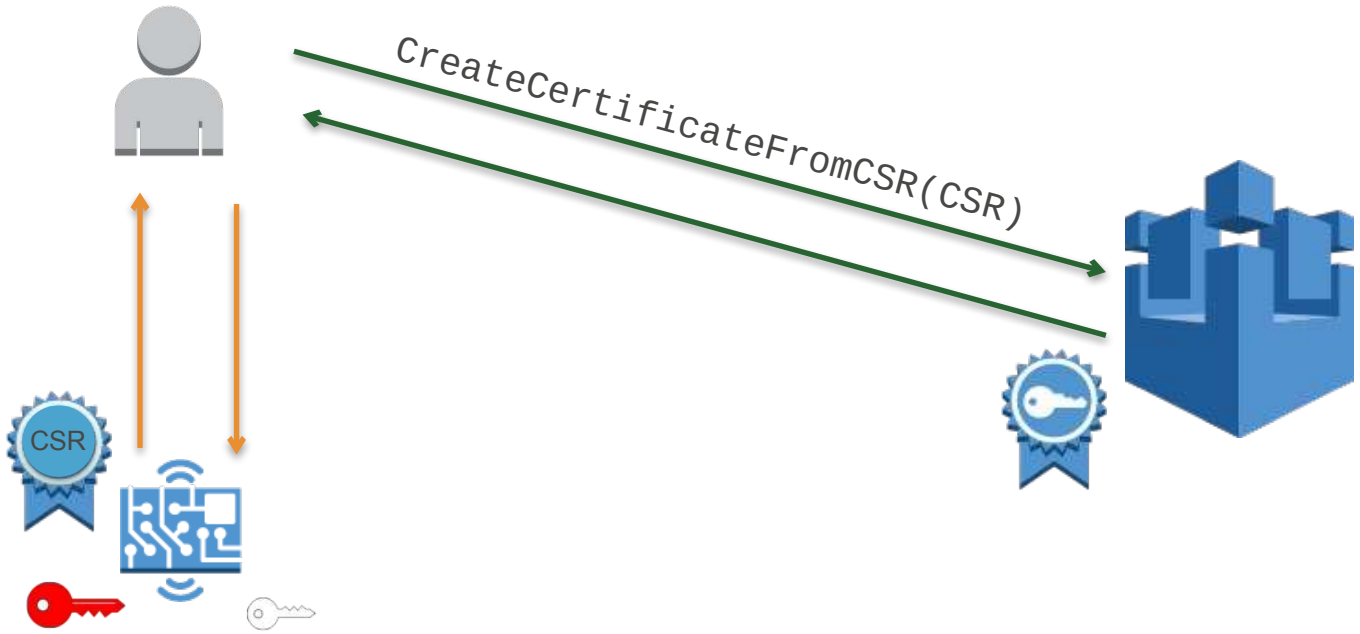
Dear Certificate Authority,

I'd really like a certificate for %NAME%, as identified by the key pair with public key %PUB_KEY%. If you could sign a certificate for me with those parameters, it'd be super spiffy.

Signed (Cryptographically),

- The holder of the private key

Client Generated Keypair



Actual Commands

```
$ openssl genrsa -out ThingKeypair.pem 2048
Generating RSA private key, 2048 bit long modulus
....+++
...+++
e is 65537 (0x10001)

$ openssl req -new -key ThingKeypair.pem -out Thing.csr
-----
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:NY
Locality Name (eg, city) [Default City]:New York
Organization Name (eg, company) [Default Company Ltd]:ACME
Organizational Unit Name (eg, section) []:Makers
Common Name (eg, your name or your server's hostname) []:John Smith
Email Address []:jsmith@acme.com
```

Actual Commands

```
$ aws iot create-certificate-from-csr \
   --certificate-signing-request file://Thing.csr \
   --set-as-active
{
    "certificateArn":
        "arn:aws:iot:us-east-1:123456972007:cert/b5a396e…SNIP…400877b",
    "certificatePem":
        "-----BEGIN CERTIFICATE-----…SNIP…-----END CERTIFICATE-----",
    "certificateId":
        "b5a396e…SNIP…400877b"
}
```

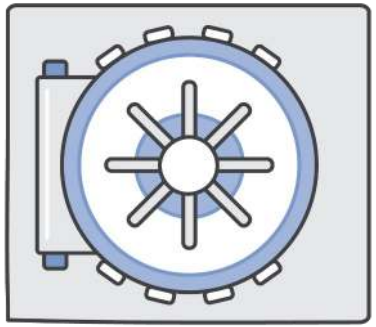
Private Key Protection – Test & Dev

```
$ openssl genrsa -out ThingKeypair.pem 2048
Generating RSA private key, 2048 bit long modulus
.....................+++
.................................+++
e is 65537 (0x10001)

$ ls -l ThingKeypair.pem
-rw-rw-r-- 1 ec2-user ec2-user 1679 Sep 25 14:10 ThingKeypair.pem

$ chmod 400 ThingKeypair.pem ; ls -l ThingKeypair.pem
-r-------- 1 ec2-user
```

Private Key Protection – Software Threats

- chroot
- SELinux
- OTP Fuses

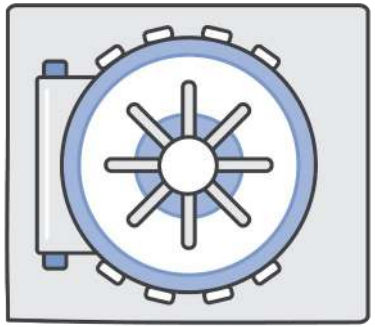
Protect your Keys



Private Key Protection – Hardware Threats

- TPMs
- Smartcards
- Locks and Boxes
- FIPS-style hardware

Protect your Keys



Identity Revocation

```
$ aws iot list-certificates
{
    "certificateDescriptions": [
        {
            "certificateArn":
                "arn:aws:iot:us-east-1:123456972007:cert/d7677b0…SNIP…026d9",
            "status": "ACTIVE",
            "certificateId":
                "d7677b0…SNIP…026d9"
            "lastModifiedDate": 1443070900.491,
            "certificatePem":
                "-----BEGIN CERTIFICATE-----…SNIP…-----END CERTIFICATE-----",
            "ownedBy": "123456972007",
            "creationDate": 1443070900.491
        }
    ]
}
```

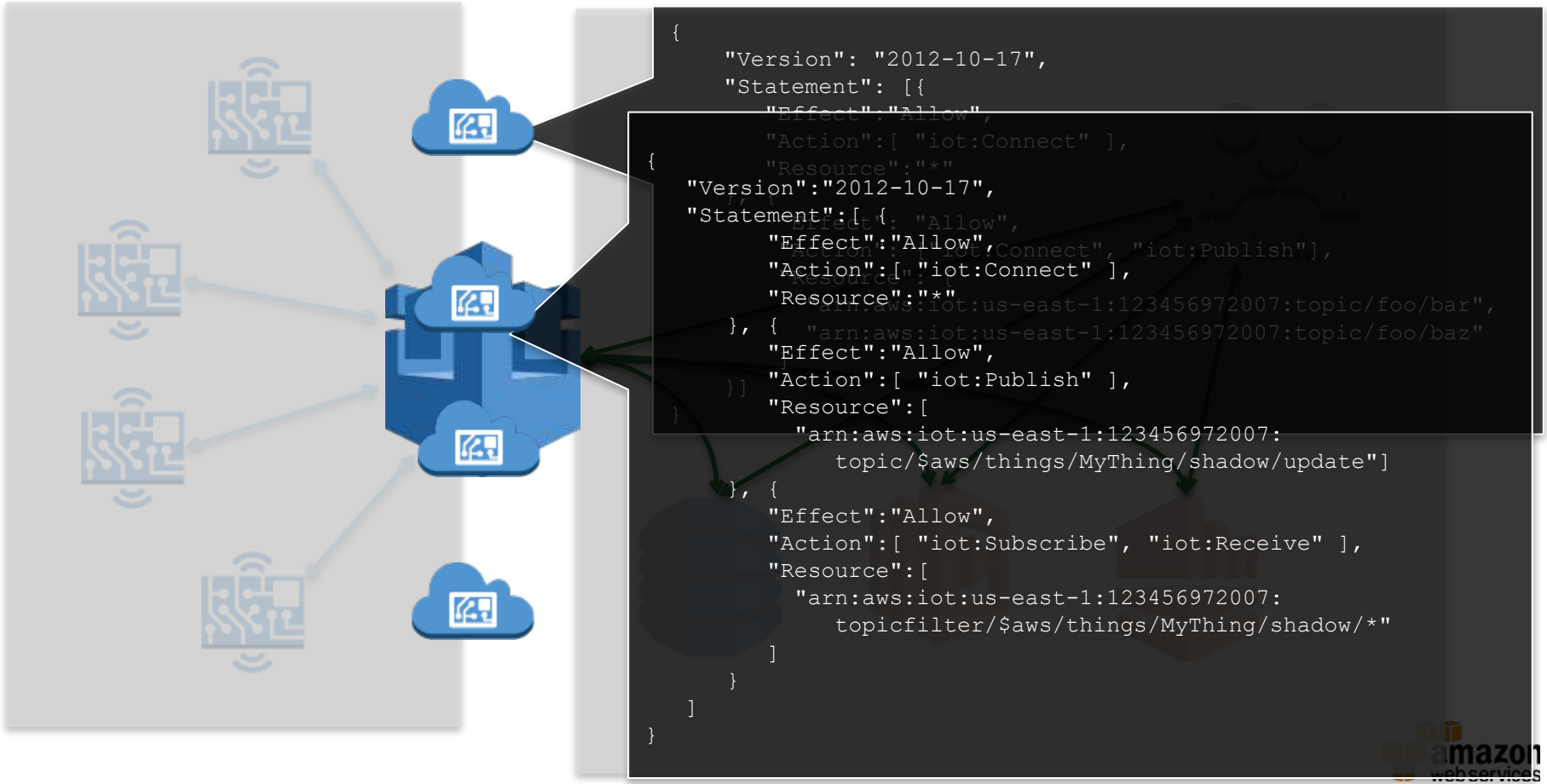
Identity Revocation

```
$ aws iot update-certificate --certificate-id "d7677b0…SNIP…026d9" --new-status REVOKED

$ aws iot list-certificates
{
    "certificateDescriptions": [
        {
            "certificateArn":
                "arn:aws:iot:us-east-1:123456972007:cert/d7677b0…SNIP…026d9",
            "status": "REVOKED",
            "certificateId":
                "d7677b0…SNIP…026d9"
            "lastModifiedDate": 1443192020.792,
            "certificatePem":
                "-----BEGIN CERTIFICATE-----…SNIP…-----END CERTIFICATE-----",
            "ownedBy": "123456972007",
            "creationDate": 1443070900.491
        }
    ]
}
```

Requirements

- Secure Communications with Things
- Strong Thing Identity
- Fine-grained Authorization for:
- Thing Management
- Pub/Sub Data Access
- AWS Service Access

Data Access Control - MQTT



Actual Commands

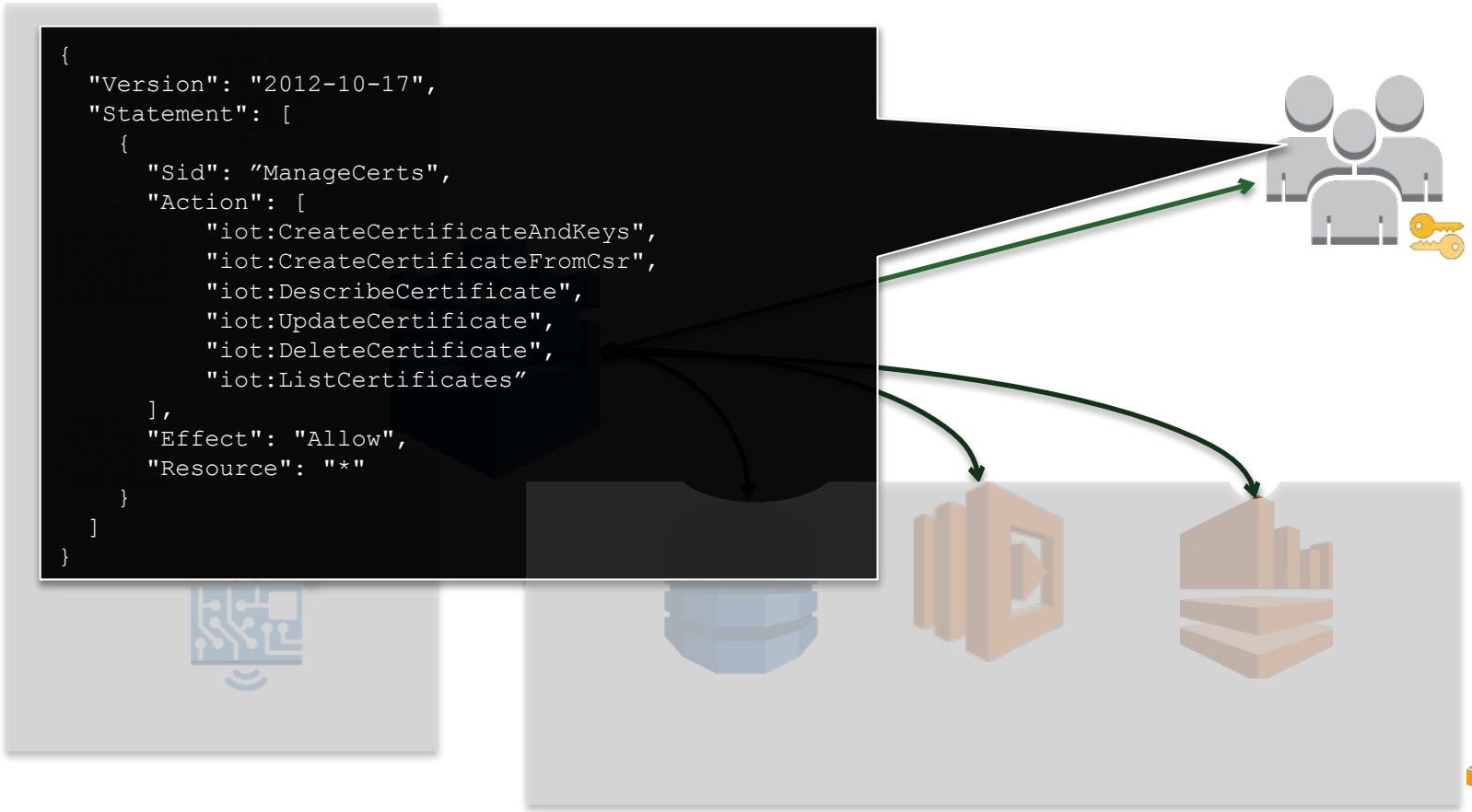
```
$ cat MyThingPolicy.json
{
   "Version":"2012-10-17",
   "Statement":[ {
         "Effect":"Allow",
         "Action":[ "iot:Connect" ],
         "Resource":"*"
      }, {
         "Effect":"Allow",
         "Action":[ "iot:Publish" ],
         "Resource":["arn:aws:iot:us-east-1:123456972007:
              topic/$aws/things/MyThing/shadow/update"]
      }, {
         "Effect":"Allow",
         "Action":[ "iot:Subscribe", "iot:Receive" ],
         "Resource":["arn:aws:iot:us-east-1:123456972007:
              topicfilter/$aws/things/MyThing/shadow/*"
         ]
      }
   ]
}
```

Actual Commands

```
$ aws iot create-policy\
  --policy-name MyThingPolicy\
  --policy-document file://MyThingPolicy.json
{
    "policyName": "MyThingPolicy",
    "policyArn": "arn:aws:iot:us-east-1:123456972007:policy/MyThingPolicy",
    "policyDocument": "...SNIP...",
    "policyVersionId": "1"
}
```

```
$ aws iot attach-principal-policy\
   --principal "arn:aws:iot:us-east-1:123456972007:cert/b5a396e…SNIP…400877b”\
   --policy-name "MyThingPolicy"
```

Managing Things



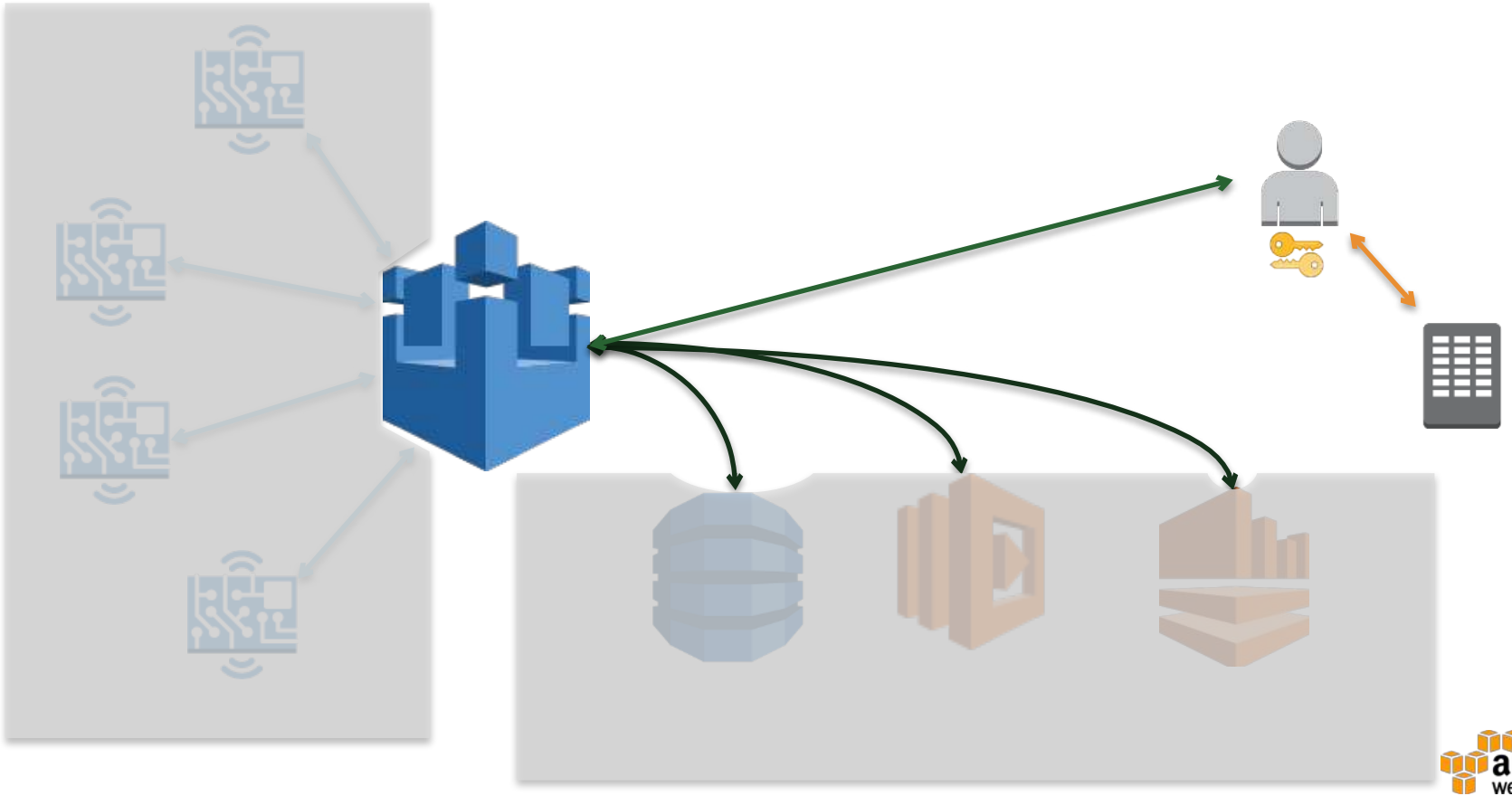
amazon web services

About Things

```
$ aws iot create-keys-and-certificate --set-as-active
{
    "certificateArn":
        "arn:aws:iot:us-east-1:123456972007:cert/d7677b0…SNIP…026d9",
    "certificatePem":
        "-----BEGIN CERTIFICATE-----…SNIP…-----END CERTIFICATE-----",
    "keyPair": {
        "PublicKey":
            "-----BEGIN PUBLIC KEY-----…SNIP…-----END PUBLIC KEY-----",
        "PrivateKey":
            "-----BEGIN RSA PRIVATE KEY-----…SNIP…-----END RSA PRIVATE KEY-----"
    },
    "certificateId":
        "d7677b0…SNIP…026d9"
}
```

Managing Things

Identity Federation



Requirements

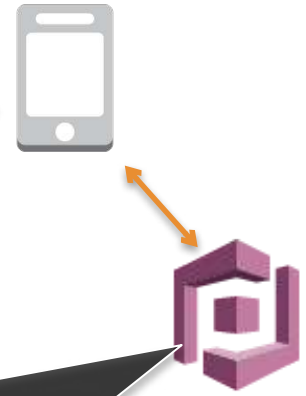
- Secure Communications with Things
- Strong Thing Identity
- Fine-grained Authorization for:
- Thing Management
- Pub/Sub Data Access
- AWS Service Access

Data Access Control – AWS APIs

```
{
   "Version":"2012-10-17",
   "Statement":[ {
         "Effect":"Allow",
         "Action":[ "iot:Connect" ],
         "Resource":"*"
      }, {
         "Effect":"Allow",
         "Action":[ "iot:GetThingShadow" ],
         "Resource":[
           "arn:aws:iot:us-east-1:123456972007:thing/MyThing"]
      }, {
         "Effect":"Allow",
         "Action":[ "iot:Publish" ],
         "Resource":[ "arn:aws:iot:us-east-1:123456972007:
              topic/$aws/things/MyThing/shadow/update"]
      }
   ]
}
```

Mobile Users as Things

```
{
   "Version":"2012-10-17",
   "Statement":[ {
         "Effect":"Allow",
         "Action":[ "iot:Connect" ],
         "Resource":"*"
      }, {
         "Effect":"Allow",
         "Action":[ "iot:GetThingShadow" ],
         "Resource":[
           "arn:aws:iot:us-east-1:123456972007:
             thing/${cognito-identity.amazonaws.com:aud}"]
      }, {
         "Effect":"Allow",
         "Action":[ "iot:Publish" ],
         "Resource":[ "arn:aws:iot:us-east-1:123456972007:topic/$aws/things/
           ${cognito-identity.amazonaws.com:aud}/shadow/update"]
      }
   ]
}
```



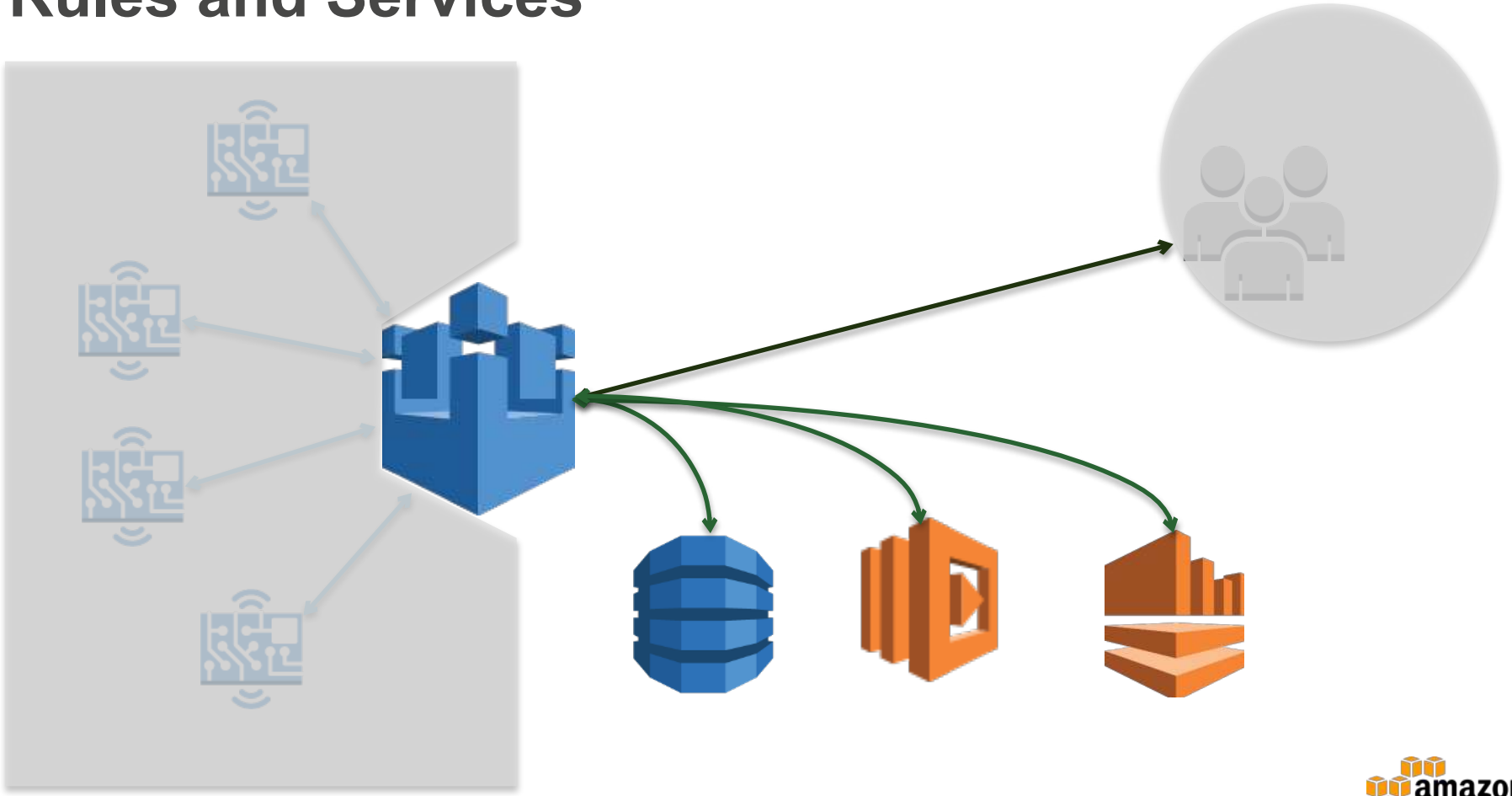
Protocol Convergence

	MQTT + Mutual Auth TLS	AWS Auth + HTTPS
Server Auth	TLS + Cert	TLS + Cert
Client Auth	TLS + Cert	AWS API Keys
Confidentiality	TLS	TLS
Protocol	MQTT	HTTP
Identification	AWS ARNs	AWS ARNs
Authorization	AWS Policy	AWS Policy

Requirements

- Secure Communications with Things
- Strong Thing Identity
- Fine-grained Authorization for:
- Thing Management
- Pub/Sub Data Access
- AWS Service Access

Rules and Services



Actual Commands

```
$ cat ThingRoleTrustPolicy.json
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"",
         "Effect":"Allow",
         "Principal":{
            "Service":"iot.amazonaws.com"
         },
         "Action":"sts:AssumeRole"
      }
   ]
}
```

Actual Commands

```
$ aws iam create-role\
  --role-name thing-actions-role\
  --assume-role-policy-document file://ThingRoleTrustPolicy.json
{
    "Role": {
        "AssumeRolePolicyDocument": …SNIP…
        "RoleId": "AROAIQ4HBGG7V7F27E32K",
        "CreateDate": "2015-09-27T16:29:56.438Z",
        "RoleName": "thing-actions-role",
        "Path": "/",
        "Arn": "arn:aws:iam::123456972007:role/thing-actions-role"
    }
}
```

Actual Commands

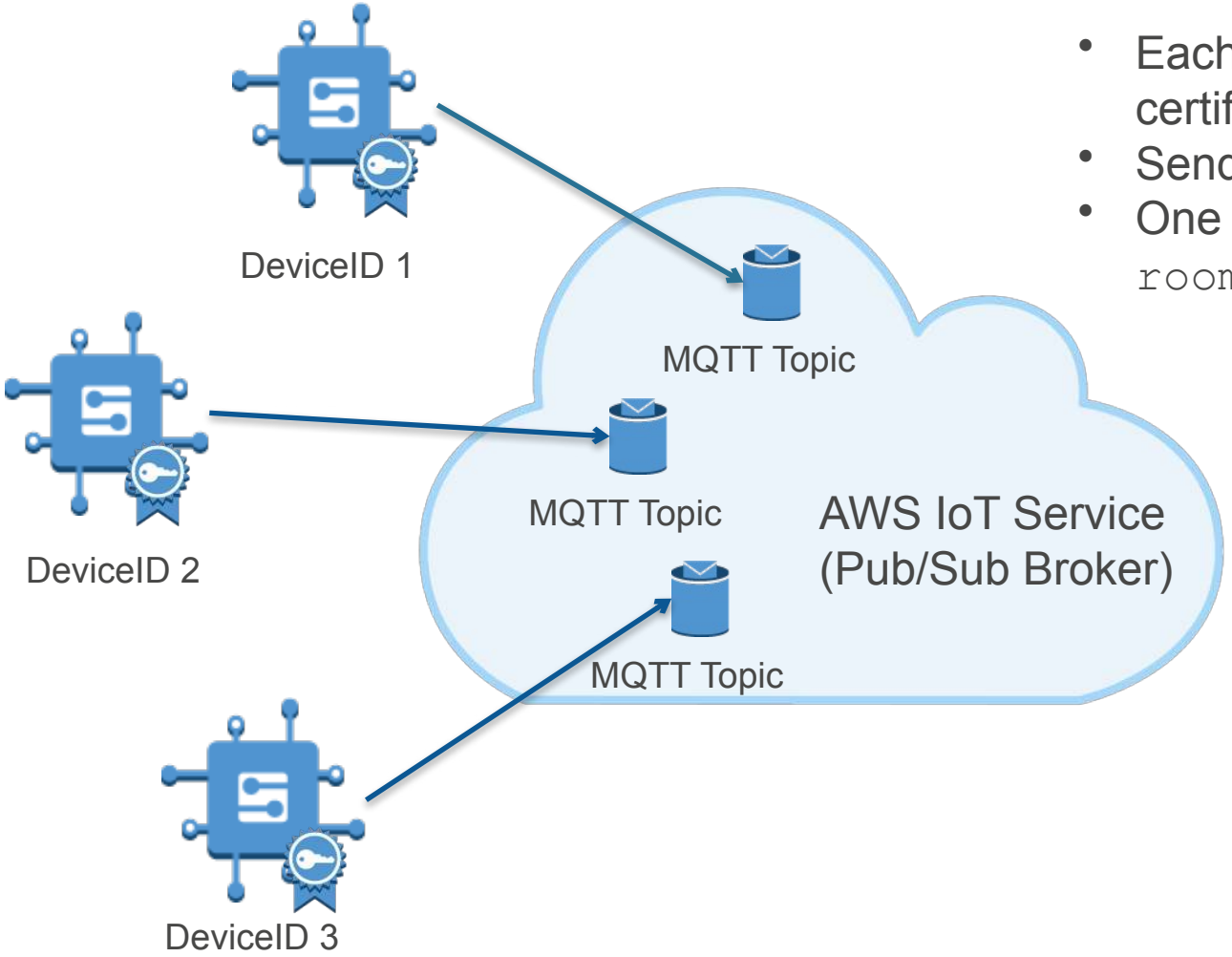
```
$ cat ThingRolePolicy.json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DDBAccess",
      "Action": [
        "dynamodb:PutItem",
        "dynamodb:UpdateItem"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:dynamodb:us-east-1:123456972007:table/MyThingTable"
    },
  ]
}
```

Actual Commands

```
$ aws iam create-policy\
  --policy-name thing-role-policy\
  --policy-document file://ThingRolePolicy.json
```

```
{
    "Policy": {
        "PolicyName": "thing-role-policy",
        "CreateDate": "2015-09-27T16:32:17.998Z",
        "AttachmentCount": 0,
        "IsAttachable": true,
        "PolicyId": "ANPAINCEAOD5EEXOLZWAI",
        "DefaultVersionId": "v1",
        "Path": "/",
        "Arn": "arn:aws:iam::123456972007:policy/thing-role-policy",
        "UpdateDate": "2015-09-27T16:32:17.998Z"
    }
}
```

```
$ aws iam attach-role-policy\
  --role-name "thing-actions-role"\
  --policy-arn "arn:aws:iam::123456972007:policy/thing-role-policy"
```

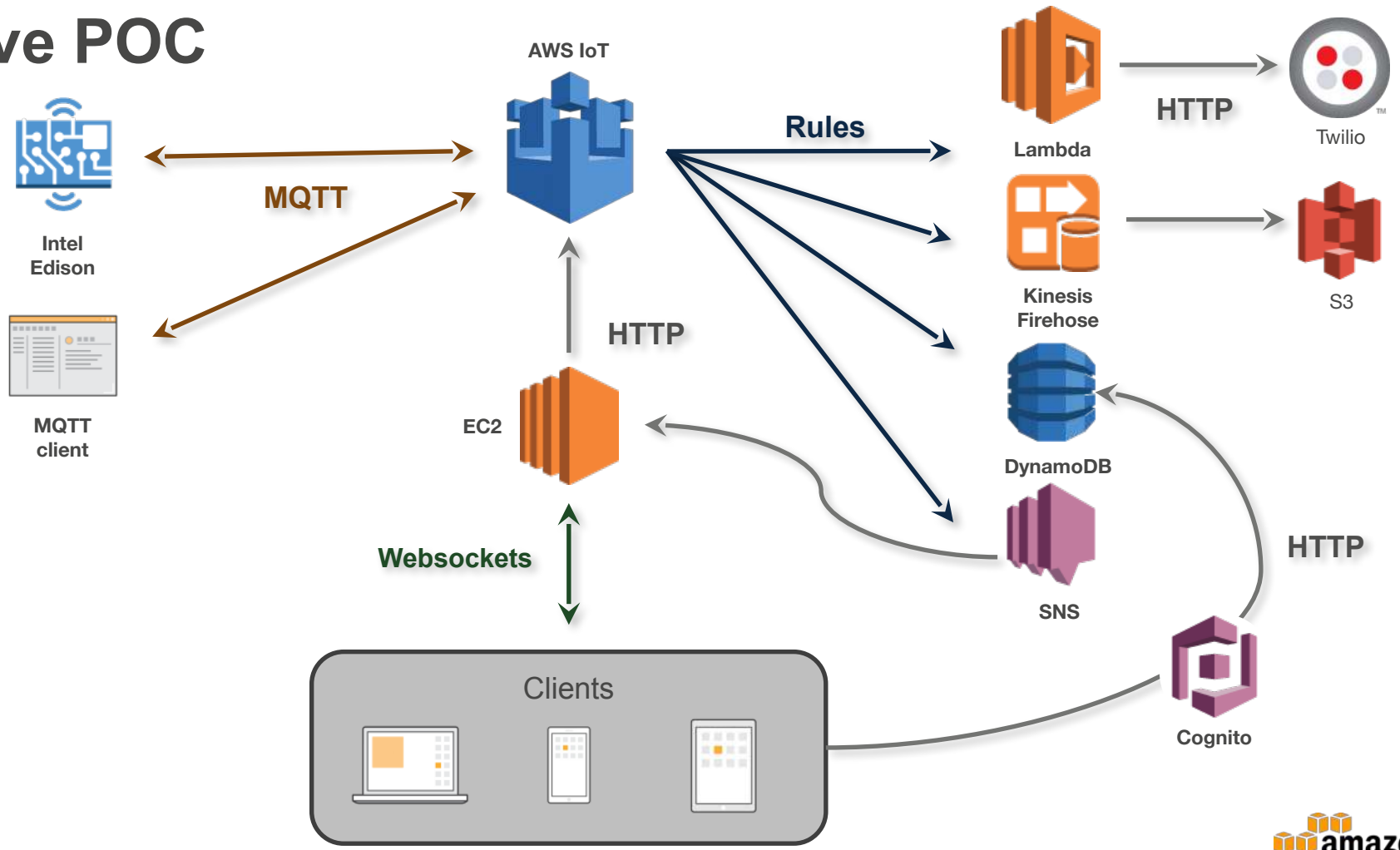


- Each device uses certificate authentication
- Send messages via MQTT
- One topic per device: `rooms/ac/${deviceID}`

Live POC Time !

amazon web services

Live POC


AWS IoT
Intel Edison
MQTT
MQTT client
Rules
Lambda
HTTP
Twilio
Kinesis Firehose
S3
HTTP
EC2
DynamoDB
Websockets
SNS
HTTP
Clients
Cognito

Config Edison

Intel Edison

MQTT

AWS IoT

J5 http://johnny-five.io/

Johnny Five & Edison.io are dedicated libraries for managing Edison devices in Node.js

```
$ wget https://signed-url-to-archive -O Edison.zip
$ unzip Edison.zip && rm Edison.zip
$ cd Edison
$ npm install aws-iot-device-sdk johnny-five edison-io
```

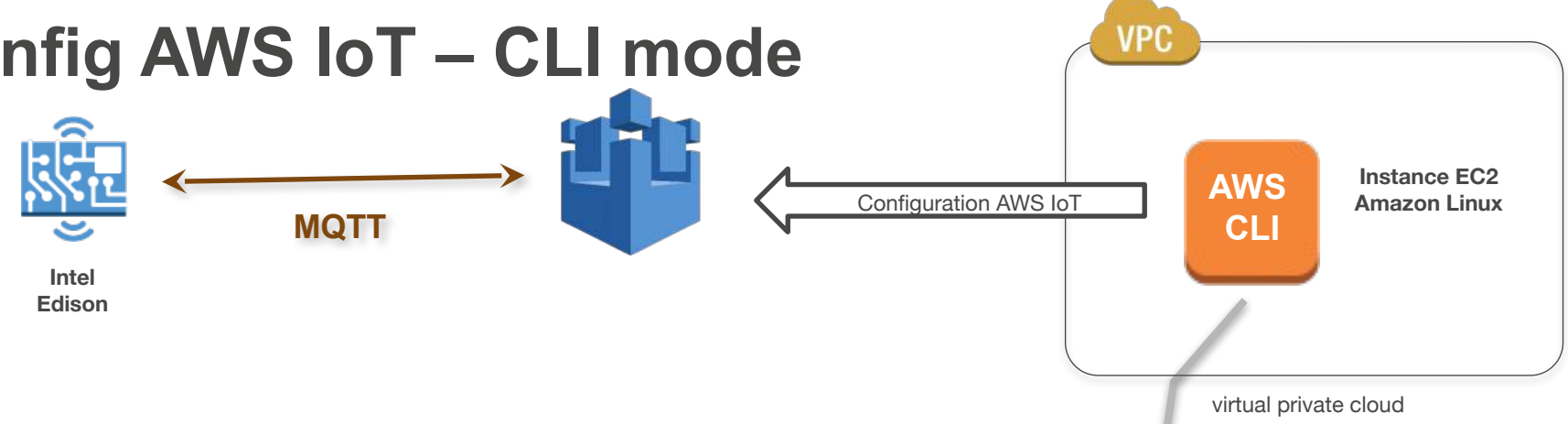
intel

AWS IoT SDK Node.js

https://github.com/aws/aws-iot-device-sdk-js

amazon web services

Config AWS IoT – CLI mode

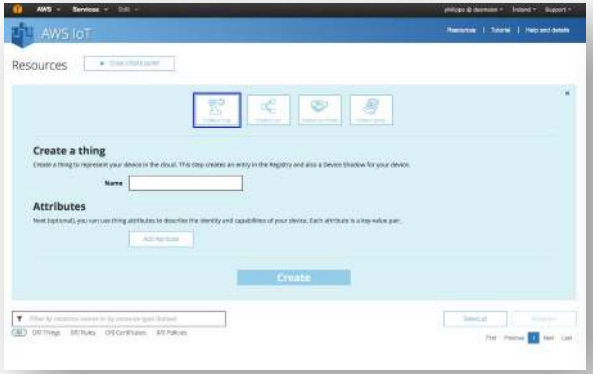


Thing & Certificates & Policy creation, Policy attachment

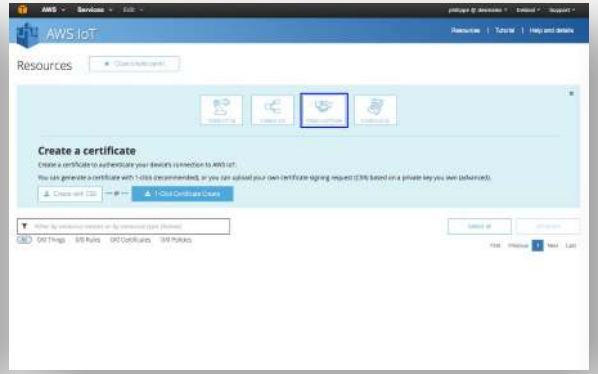
```
$ aws iot create-thing -thing-name "MyThing"

$ aws iot create-keys-and-certificate --set-as-active

$ aws iot create-policy\
  --policy-name MyThingPolicy\
  --policy-document file://MyThingPolicy.json

$ $ aws iot attach-principal-policy\
   --principal "arn:aws:iot:us-east-1:123456972007:cert/b5a396e…SNIP…400877b"\
   --policy-name "MyThingPolicy"
```

Config AWS IoT - Console mode

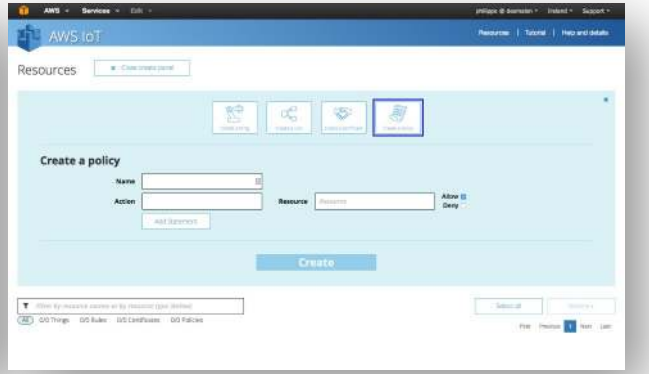
AWS IoT



Create a Thing



Create Certificates



Create/Attach a Policy

Edison-Basic.js

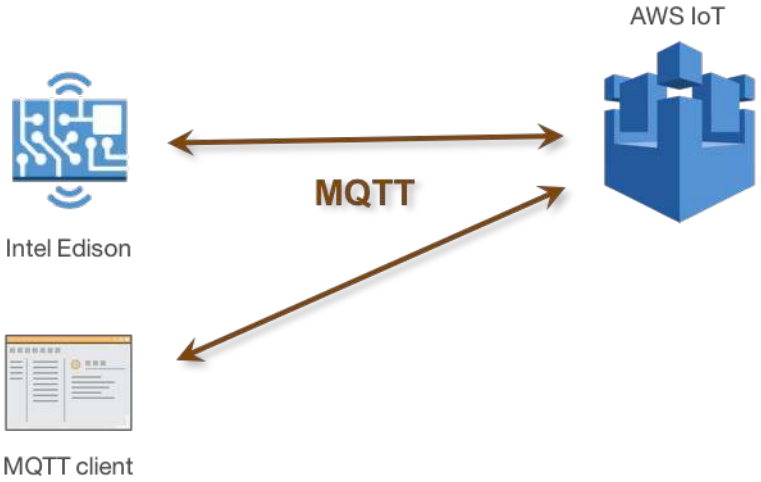
Node.js

```
//--- AWS IoT SDK module import and init ---//
const awsIot = require('aws-iot-device-sdk');

const name = 'Edison-basic';
const config = require('./config.json');
const device = awsIot.device({
        keyPath: __dirname + '/certificates/private.pem.key',
        certPath: __dirname + '/certificates/certificate.pem.crt.txt',
        caPath: __dirname + '/certificates/rootCA.pem',
        clientId: clientname,
        region: config.iot_region
});

//--- Edison hardware modules import and init ---//
const five = require("johnny-five");
const Edison = require("edison-io");
const board = new five.Board({
        io: new Edison(),
        repl: false
});
```

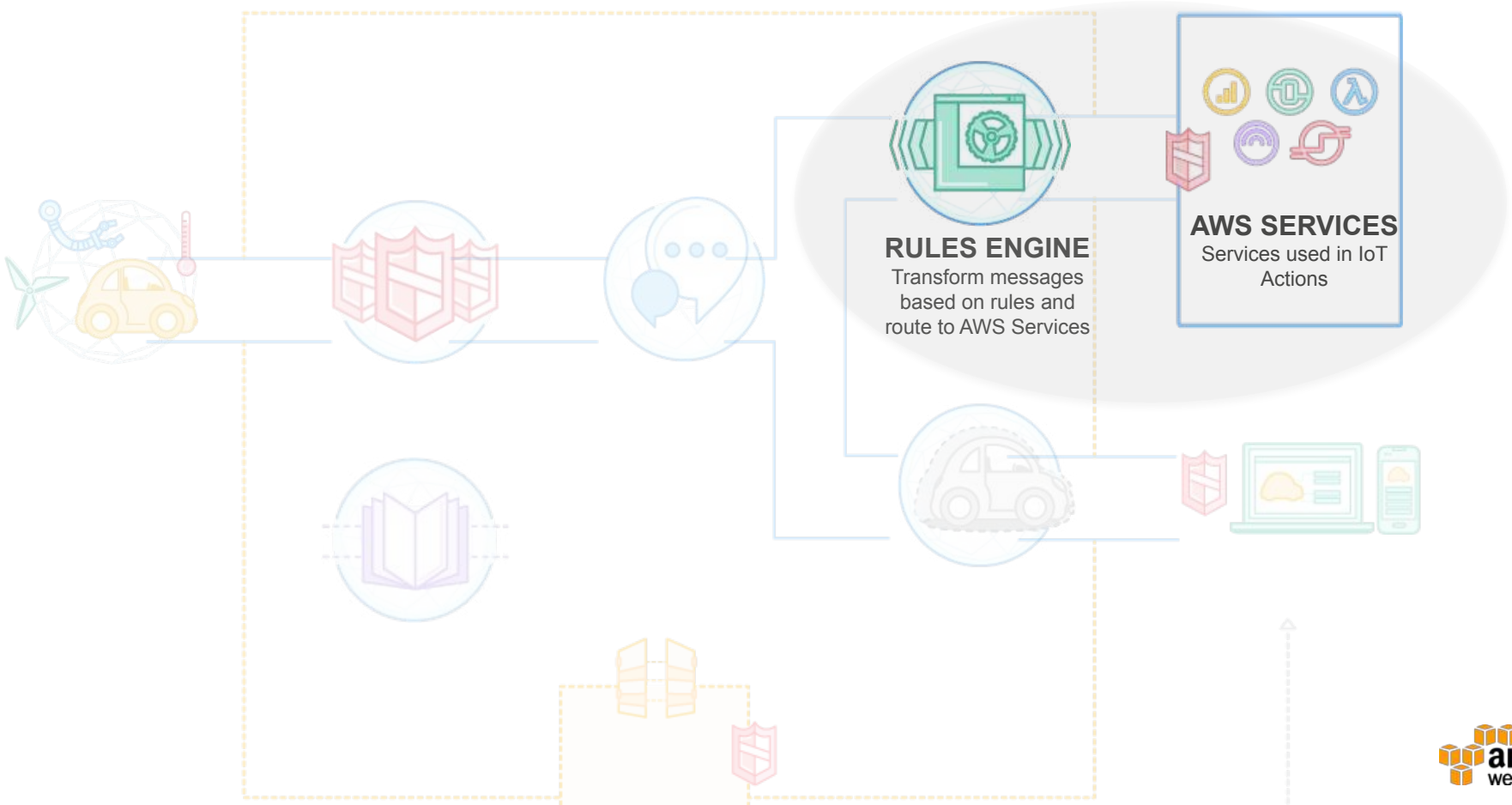
Intel Edison

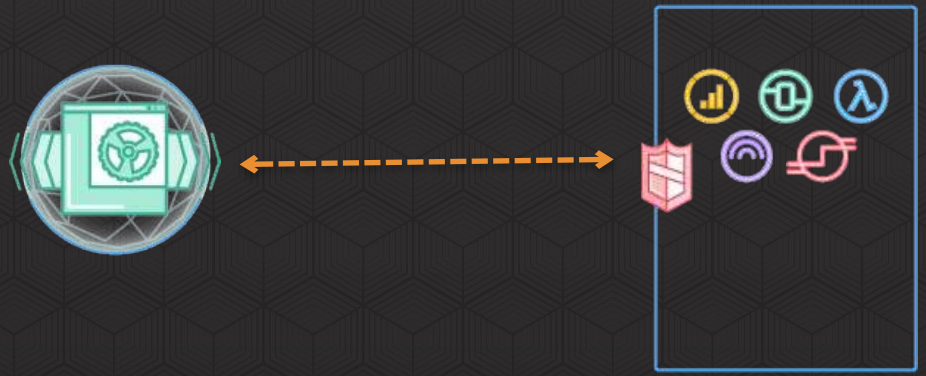
To be continued in a TextEditor

amazon web services



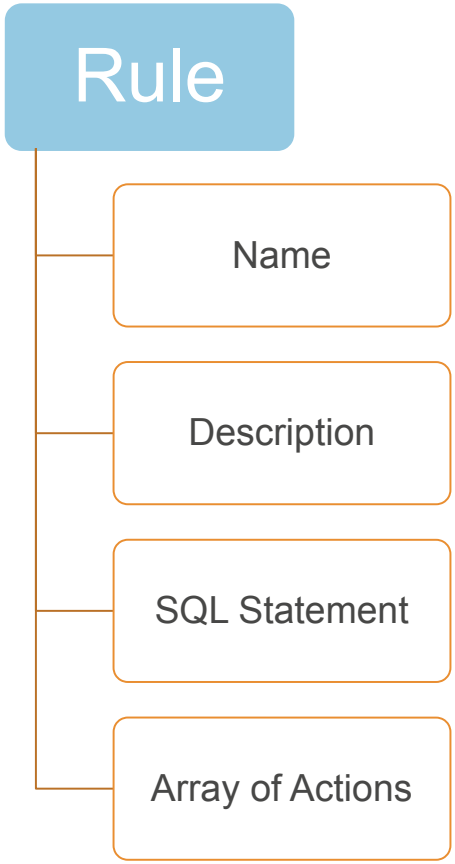
Deep dive on AWS IoT

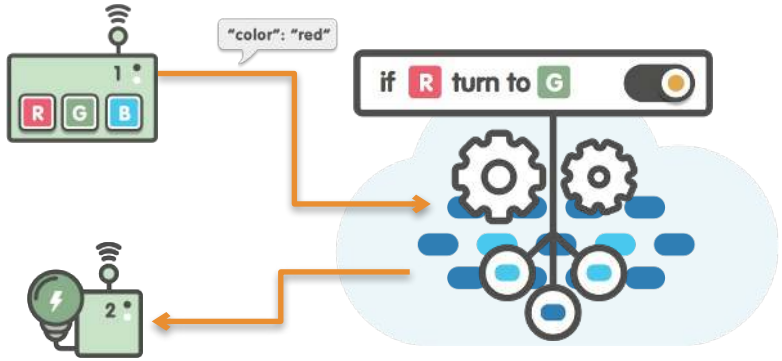




Rules Engine

AWS IoT Rules Engine Basics



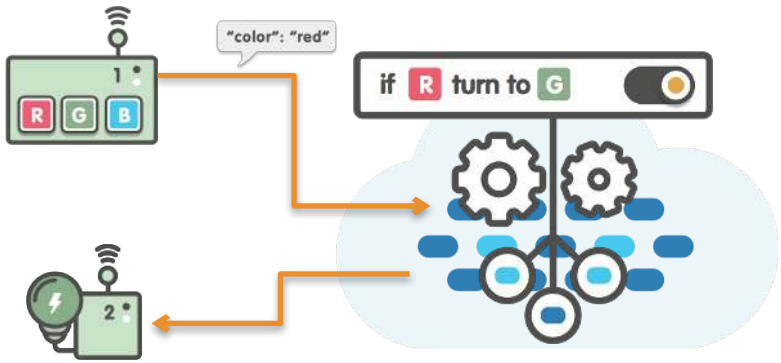


SELECT * FROM ‘things/thing-2/color’
WHERE color = ‘red’

AWS Services, Native

amazon web services

AWS IoT Rules Engine Basics



SELECT * FROM 'things/thing-2/color'
WHERE color = 'red'

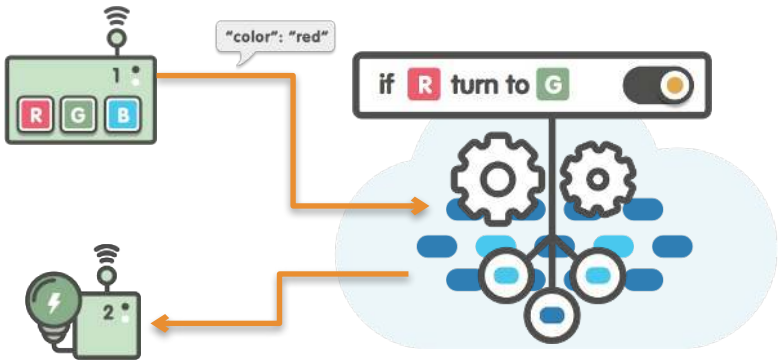
Simple & Familiar Syntax

- SQL Statement to define topic filter
- Optional WHERE clause
- Advanced JSON support

Functions improve signal : noise

- String manipulation (regex support)
- Mathematical operations
- Context based helper functions
- Crypto support
- UUID, Timestamp, rand, etc.

AWS IoT Rules Engine's Flexibility




```
SELECT *, clientId() as MQTTClientId
FROM 'one/rule'
WHERE
startsWith(topic(2), 'Vac123') AND
(state = 'SWEEP' OR bin.size < 30)",
"actions":
[{
"republish": {
    "topic":
"controllers/${substring(topic(3),
3, 5)}",
}]
```

AWS IoT – SQL Reference

AWS IoT Rules Engine – Format Example

```
{
    "sql":"SELECT 'IDLE' AS status FROM 'vacuum/+/events' WHERE
event = 'COMPLETE'",
    "actions": [
        {
            "dynamoDB": {
                "tableName":"vaccum-status",
                "hashKeyField":"vacuum_id",
                "hashKeyValue":"${topic(2)}",
                "payloadField":"statusDocument",
                "roleArn":"arn:aws:iam::77777:role/
rules_action_ddb"
            }
        }
    ]
}
```

AWS IoT – SQL Reference

SELECT **DATA** FROM…

- SELECT *
- SELECT deviceid, temp
- SELECT coords.latitude
- SELECT a.another_level.b
  - Returns {“b” : 3}
- SELECT a..b
  - Returns {“b” : 3}

SAMPLE PAYLOAD

```
{
        “deviceid” : “iot123”,
        “temp” : 54,
        “humidity” : 32,
        “coords” : {
                “latitude” : 47.615694,
                “longitude” : -122.3359976
        },
        “a” : {
                “another_level” : {
                        {“b” : 3},
                        {“b” : 5}
        }}
}
```

amazon web services

AWS IoT – SQL Reference

SELECT **DATA** FROM…

- SELECT deviceid AS client
- SELECT md5(deviceid) AS hashed_id

- **Substitution Templates**
- ${expression}
- ${topic() - md5(deviceid)}
- ${deviceid - temp}

SAMPLE PAYLOAD

```
{
        “deviceid” : “iot123”,
        “temp” : 54,
        “humidity” : 32,
        “coords” : {
                “latitude” : 47.615694,
                “longitude” : -122.3359976
        },
        “a” : {
                “another_level” : {
                        {“b” : 3},
                        {“b” : 5}
        }}
}
```

amazon web services

AWS IoT – SQL Reference

SELECT **DATA** FROM TOPIC WHERE FILTER

- Properties from the JSON Object in the payload
- “.” Operator
- “..” Operator
- “*” Operator
- Apply **functions** to attribute value

AWS IoT – SQL Reference

SELECT **DATA** FROM **TOPIC** WHERE **FILTER**

- Like scanning a database table
- Default source is an MQTT topic

- **EXAMPLES**:
- FROM mqtt(‘my/topic’)
- FROM mqtt(‘my/wildcard/+/topic’)
- FROM (‘my/topic’)

AWS IoT – SQL Reference

SELECT **DATA** FROM **TOPIC** WHERE **FILTER**

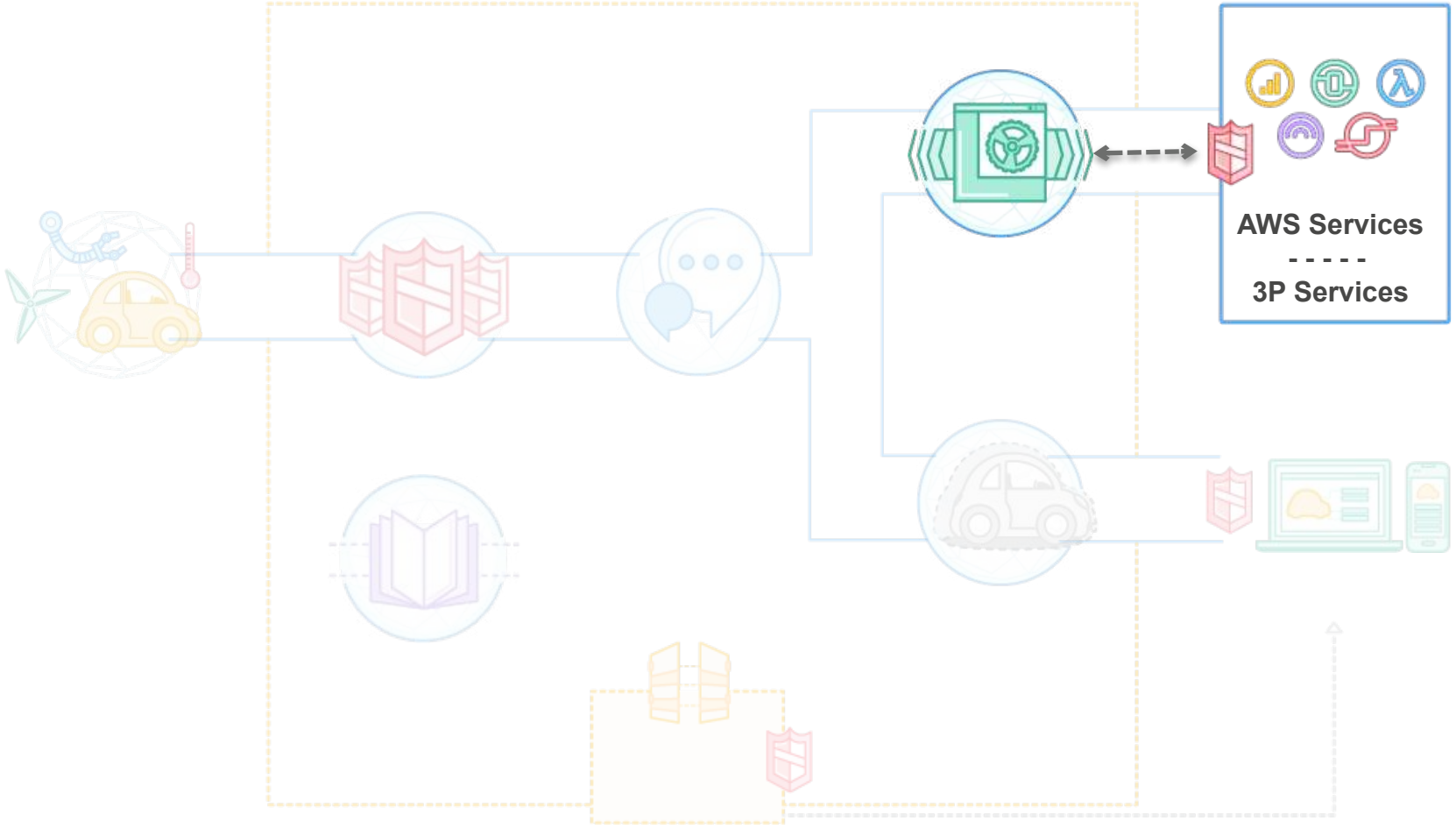
Token	Meaning	Example
=	Equal, comparison	color = 'red'
<>	Not Equal, comparison	color <> 'red'
AND	Logical AND	color = 'red' AND siren = 'on'
OR	Logical OR	color = 'red' OR siren = 'on'
()	Parenthesis, grouping	color = 'red' AND (siren = 'on' OR isTest)
+	Addition, arithmetic	5 + 3
-	Substitution, arithmetic	5 - 4
/	Division, arithmetic	8 / 2

AWS IoT – SQL Reference

SELECT **DATA** FROM **TOPIC** WHERE **FILTER**

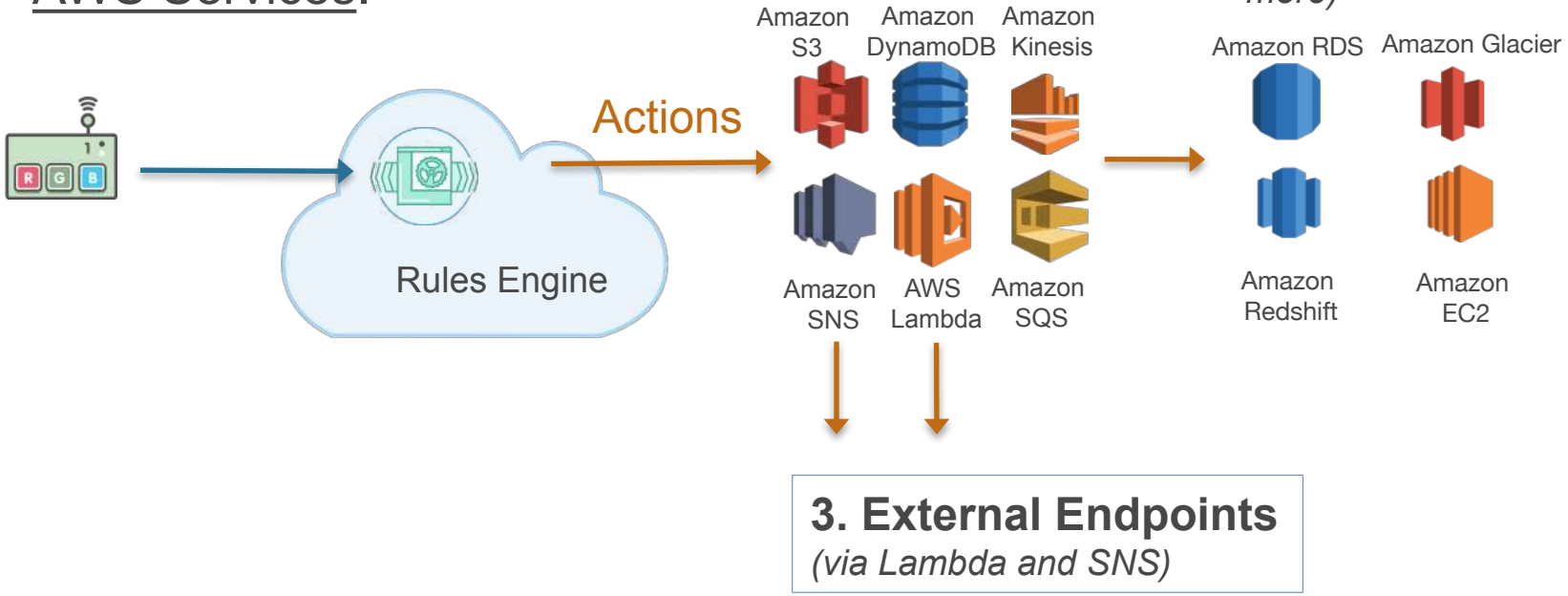
Token	Meaning	Example
<	Less than, comparison	color = 'red'
<=	Less than or equal	color <> 'red'
>	Greater than, comparison	color = 'red' AND siren = 'on'
>=	Greater than or equal	color = 'red' OR siren = 'on'
CASE … WHEN … THEN … ELSE … END	Case statement	CASE location WHEN 'home' THEN 'off' WHEN 'work' THEN 'on' ELSE 'silent' END

AWS IoT Rules Engine Actions



AWS IoT Rules Engine

Rules Engine connects AWS IoT to External Endpoints and AWS Services.



AWS IoT Rules Engine

Rules Engine evaluates inbound messages published into AWS IoT, transforms and delivers to the appropriate endpoint based on business rules.

<u>External endpoints</u> can be reached via Lambda and Amazon Simple Notification Service (Amazon SNS).

Actions

Invoke a Lambda function

Put object in an S3 bucket

Insert, Update, Read from a DynamoDB table

Publish to an SNS Topic or Endpoint

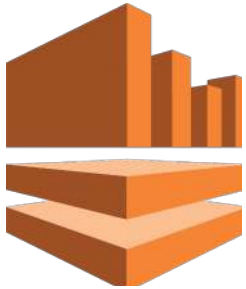
Publish to an Amazon Kinesis stream

Publish to Amazon Kinesis Firehose

Republish to AWS IoT

Processing Options

AWS **Lambda**



Amazon **Kinesis**

Amazon **SNS**

External
web service/
Webhooks

Amazon **SQS**

Worker

Processing Options

AWS **Lambda**

- Processes a **single event** at a time (no batching)
- **Enrich** data with context information from other sources
- Perform transformations
- Run any node.js / Java function
- No infrastructure to manage!

Processing Options

Amazon **SNS**

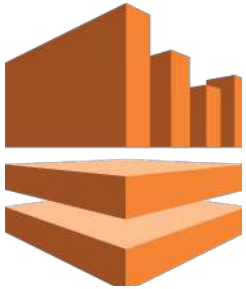
- Great for alerts: Sends push notifications, emails and SMS
- Call other systems via HTTP POST / webhooks (on AWS or on-premises)
- SNS Topics support multiple subscribers, incl. AWS Lambda and Amazon SQS

Processing Options

Amazon **SQS**

- Great when events arrive with varying frequency
- Buffer data for asynchronous processing
- Ensure that no event data is lost
- SNS Topics support multiple subscribers, incl. AWS Lambda and Amazon SQS
- Easily deploy SQS workers on AWS Elastic Beanstalk (or Amazon EC2)

Processing Options



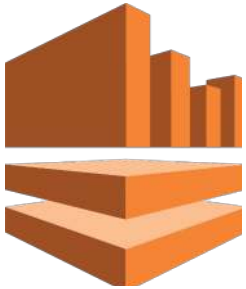
Amazon **Kinesis**

- Provides access to a "rolling window" of event data
- Scalable, can consume events from a multitude of different rules / topics / devices
- Supports many independent, concurrent readers (&writers)
- Multiple processing options:

KCL application

AWS **Lambda**

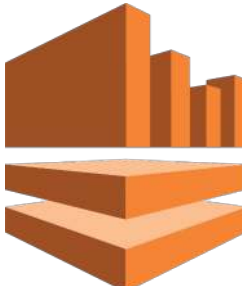
Processing Options



Amazon **Kinesis**

- Scalable way to connect **many** different systems to the stream of events, e.g., custom KCL code, Complex Event Processing (CEP) products
- Amazon Kinesis is a **hub** for all stream processing needs

Example:



Amazon **Kinesis**

1. Read last N events from stream
2. Determine maximum and rate of increase since beginning
3. Decide if alert should be sent

amazon web services

Storage Options

Amazon **S3**

Amazon **Redshift**

Amazon **RDS**

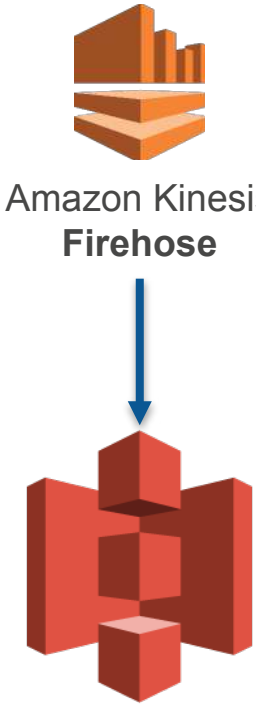
Amazon **DynamoDB**

Storage Options: Amazon S3

Amazon **S3**

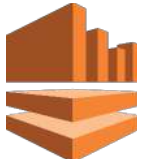
- Actions can directly write into (JSON) files on S3
- Very simple to configure, just provide bucket name
- Results in **1 file per event**
  - Lots of small files can be hard to handle
  - Inefficient when processing with Hadoop / Amazon EMR or when importing into Redshift
- Useful when you have a very low frequency of events, e.g. when you only want to log outliers to S3

Storage Options: Amazon S3 (cont'd)



- Buffer data using Amazon **Kinesis** or Amazon Kinesis **Firehose** to get fewer, larger files
- Buffering, compression & output to S3 is built into Firehose – no other infrastructure needed!
- Kinesis Connector Library can be extended to perform transformation, filter or serialize data
  - Additional Control over Buffering & Output Formats
  - Added complexity: Requires Amazon EC2 workers running Kinesis Connector Library

Storage Options: Amazon Redshift



Amazon Kinesis
Firehose

Amazon **Redshift**

- Actions can forward data Amazon Kinesis **Firehose**
  - Buffering & output to Redshift is built into Firehose
  - Very easy to setup
  - Fully managed
- Use Amazon **Kinesis** as an alternative
  - More control: Use Kinesis Connector Library to perform transformation, filter or serialize data
  - Added complexity: Requires Kinesis Connector Library etc. to execute on Amazon EC2

Storage Options: Amazon DynamoDB

Amazon
DynamoDB

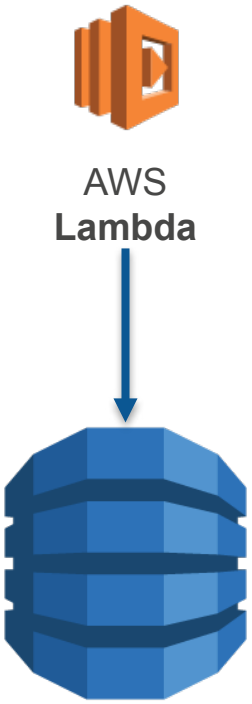
- Actions can directly **write** into Amazon DynamoDB
- Creates one row per event, can define:
  - Hash Key, Range Key and attributes to store
  - E.g. Hash Key = deviceID, range key=timestamp…
- Very simple to configure, just provide table & field names
- Adding GSIs and LSIs provides additional flexibility and enables different queries
- SELECTs can **read** from DynamoDB for fast lookups

Storage Options: Amazon DynamoDB

Amazon
DynamoDB

```
{ "sql": "SELECT * FROM 'rooms/ac/+'",
  "ruleDisabled": false,
  "actions": [{
      "dynamoDB": {
          "tableName": "my-dynamodb-table",
          "roleArn": "arn:aws:iam::X:role/mbl305-
demo-role",
          "hashKeyField": "roomID",
          "hashKeyValue": "${topic(3)}",
          "rangeKeyField": "timestamp",
          "rangeKeyValue": "${timestamp()}"
          "payloadField" :
    } }]}
```

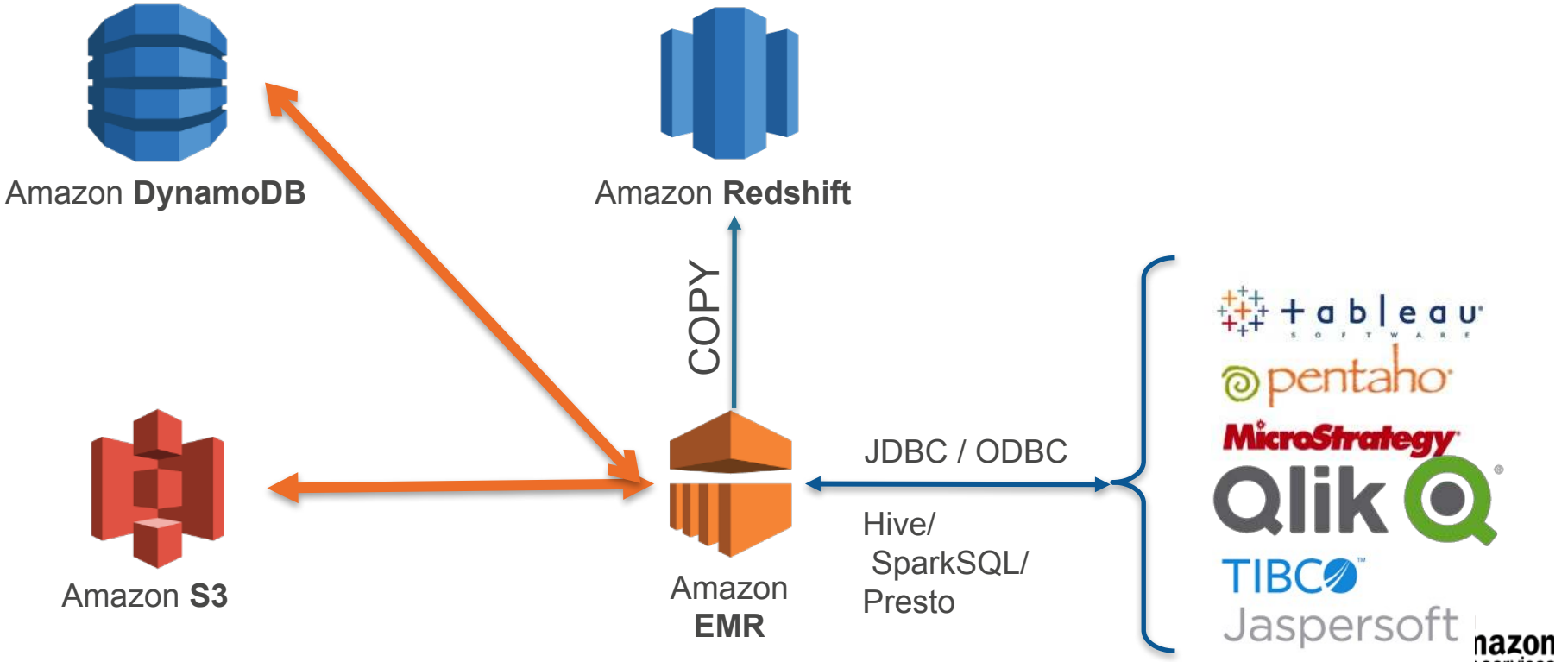
Storage Options: Amazon DynamoDB (cont'd)



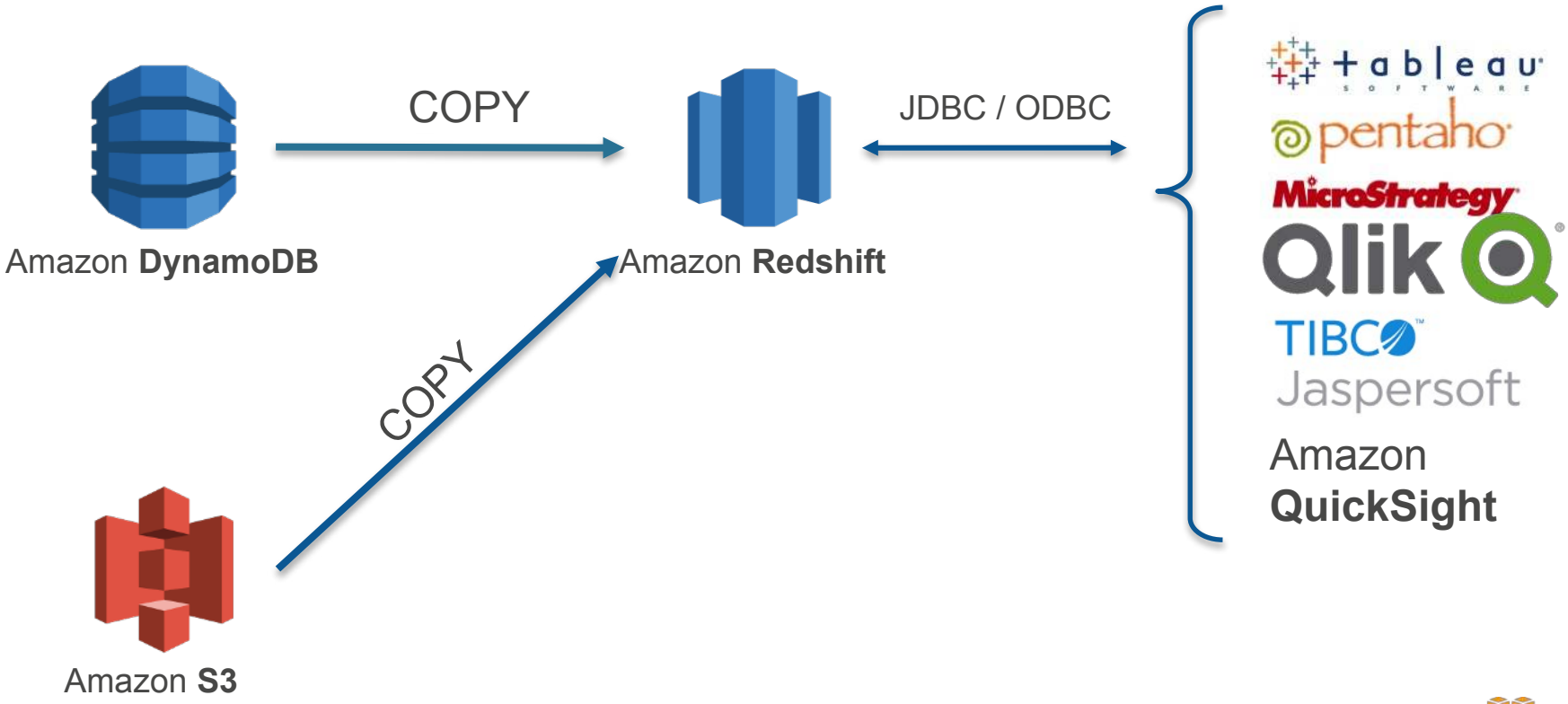
- AWS Lambda function provides additional flexibility:
  - Transform data
  - Write into different/multiple tables
  - Enrich data with contextual information pulled in from other sources

- Only able to process one event at a time! (i.e., AWS Lambda –when called from AWS IoT– cannot aggregate events before writing to DynamoDB)

Query & Analyze the Stored Data

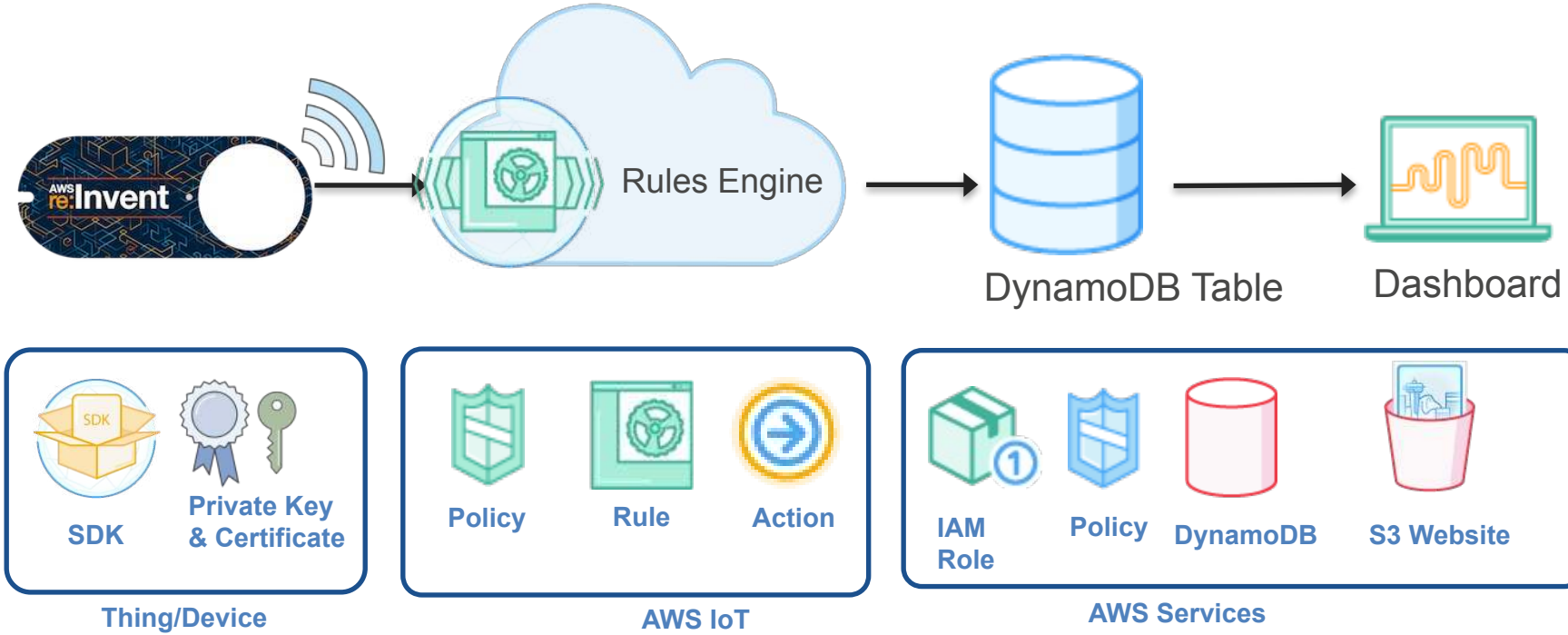


Query & Analyze the Stored Data (cont'd)



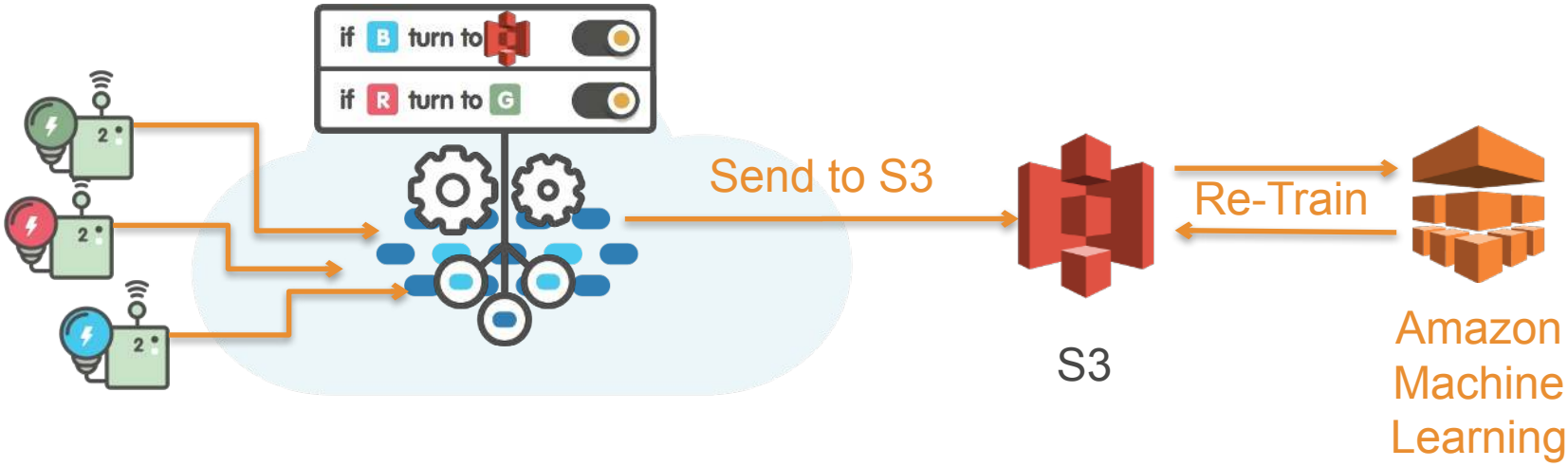
Examples

AWS IoT to Amazon DynamoDB to Dashboard



Select * from ‘iotbutton/+’

AWS IoT Rules Engine for Machine Learning



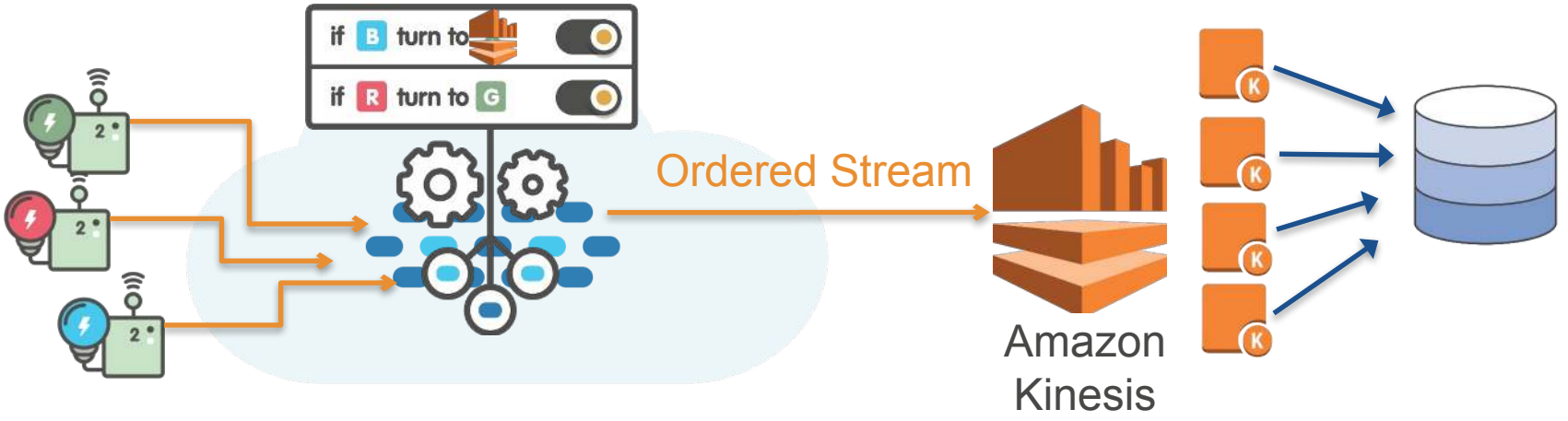
Anomaly Detection

Amazon Machine Learning can feed predictive evaluation criteria to the Rules Engine

Continuous Improvement Around Predication

Continuously look for outliers and re-calibrate the Amazon Machine Learning models

AWS IoT Rules Engine & Stream Data



N:1 Inbound Streams of Sensor Data **(Signal to Noise Reduction)**

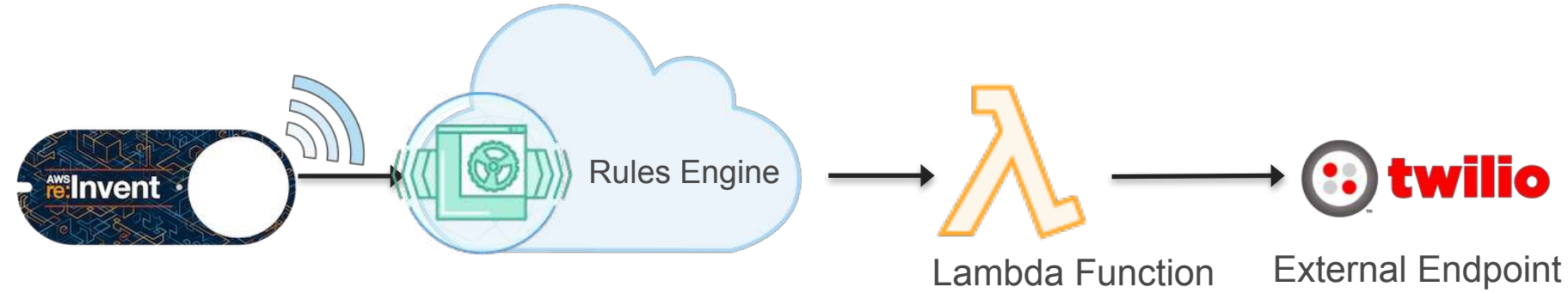
Rules Engine filters, transforms sensor data then sends aggregate to Amazon Kinesis

Amazon Kinesis Streams to Enterprise Applications

Simultaneously stream processed data to databases, applications, other AWS Services

amazon web services

AWS IoT to AWS Lambda to and External Endpoint



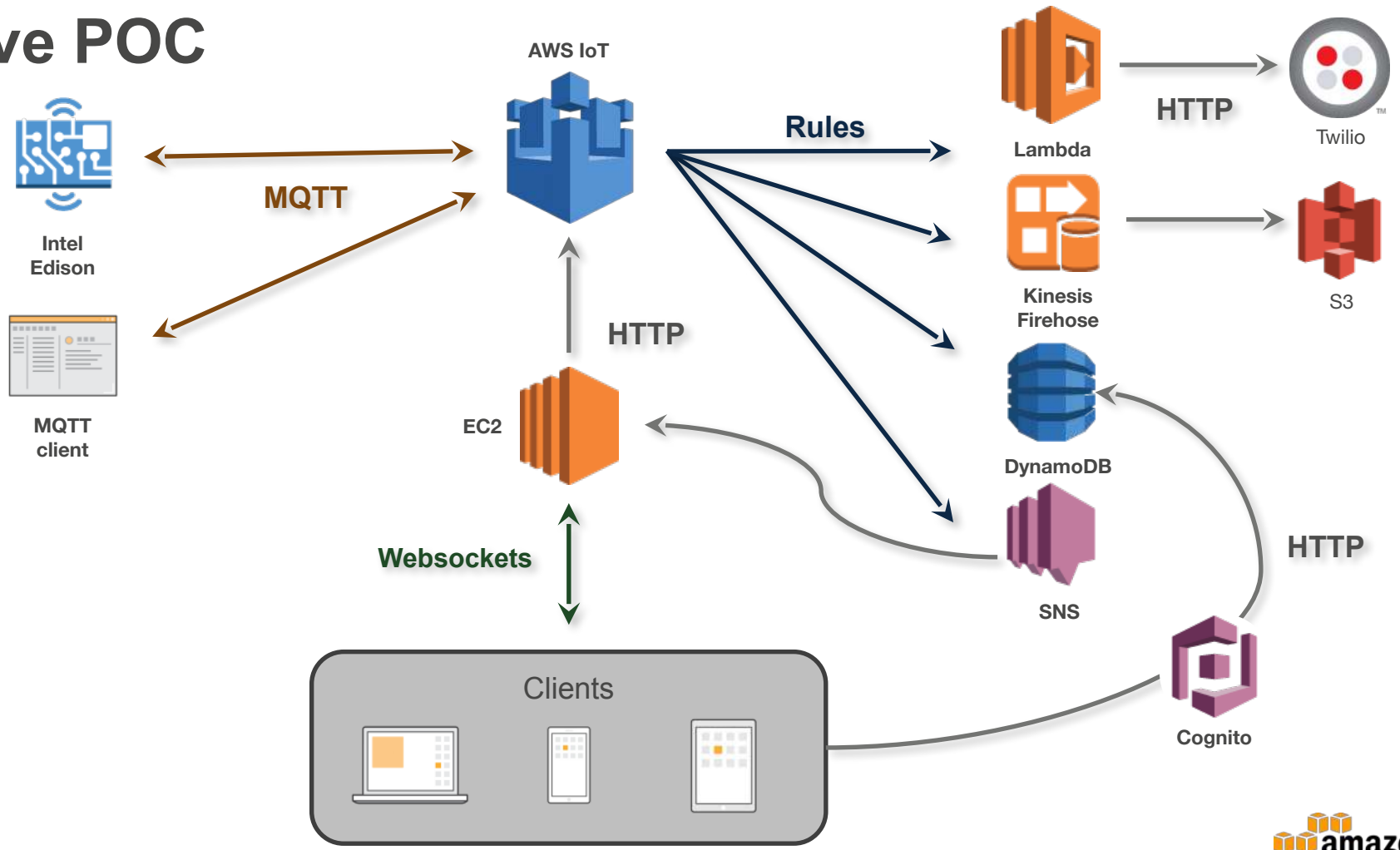
Select * from ‘iotbutton/+’

amazon web services

Live POC Time !

amazon web services

Live POC


AWS IoT
Intel Edison
MQTT
MQTT client
Rules
Lambda
HTTP
Twilio
Kinesis Firehose
S3
HTTP
EC2
DynamoDB
Websockets
SNS
HTTP
Clients
Cognito

Send-sms.js

Node.js

```
const config = require('./config.json');
const client = require('twilio')(config.accountSid, config.authToken);

exports.handler = function(event, context)
{
    client.messages.create({
        to: config.phoneNumber,
        from: "Lambda",
        body: "Edison has been turned off."
    }, function(error, message) {
        if (error) {
            context.fail(error);
        }
        else {
            context.succeed(message);
        }
    });
};
```

AWS Lambda

Twilio

amazon web services

DynamoDB

DynamoDB

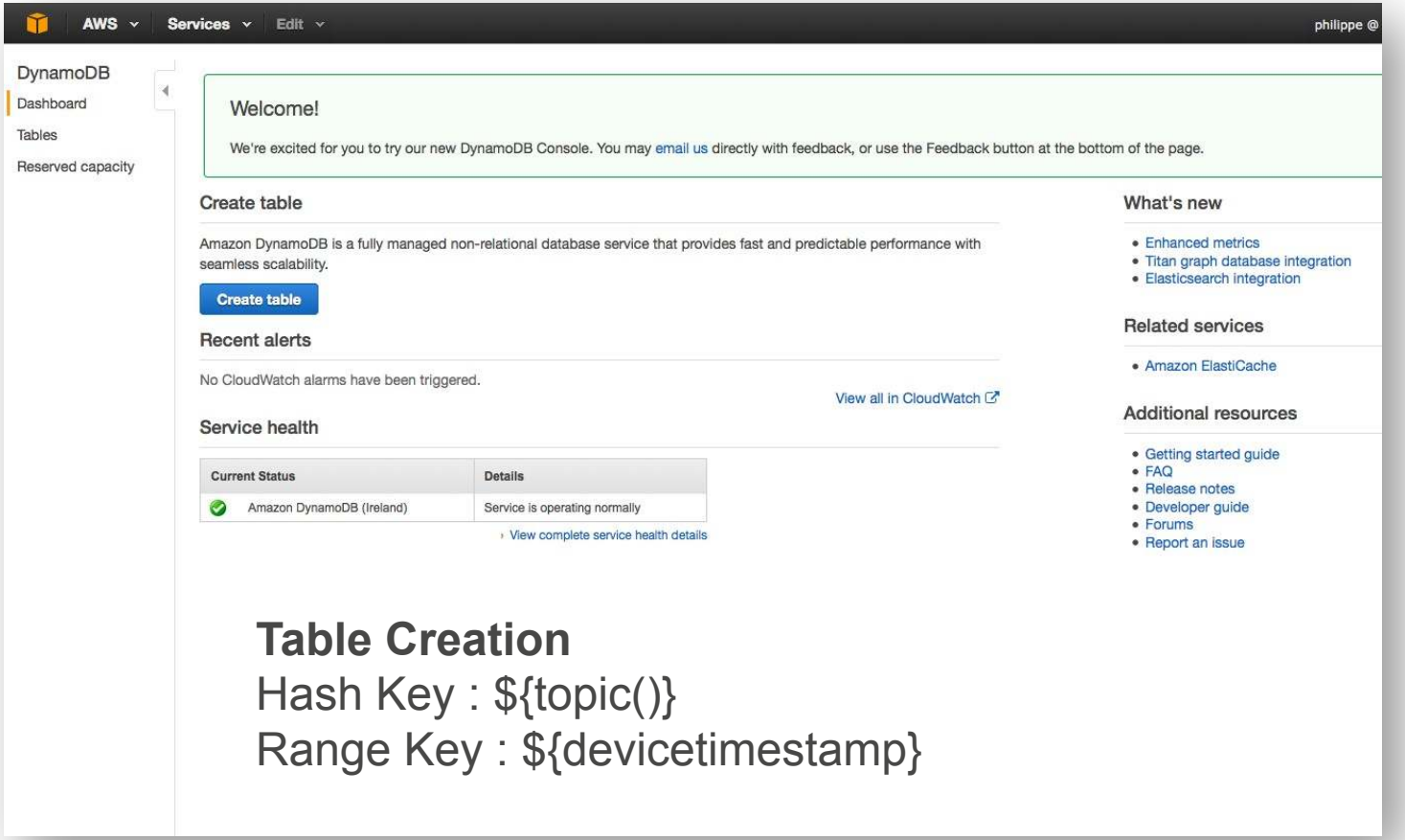


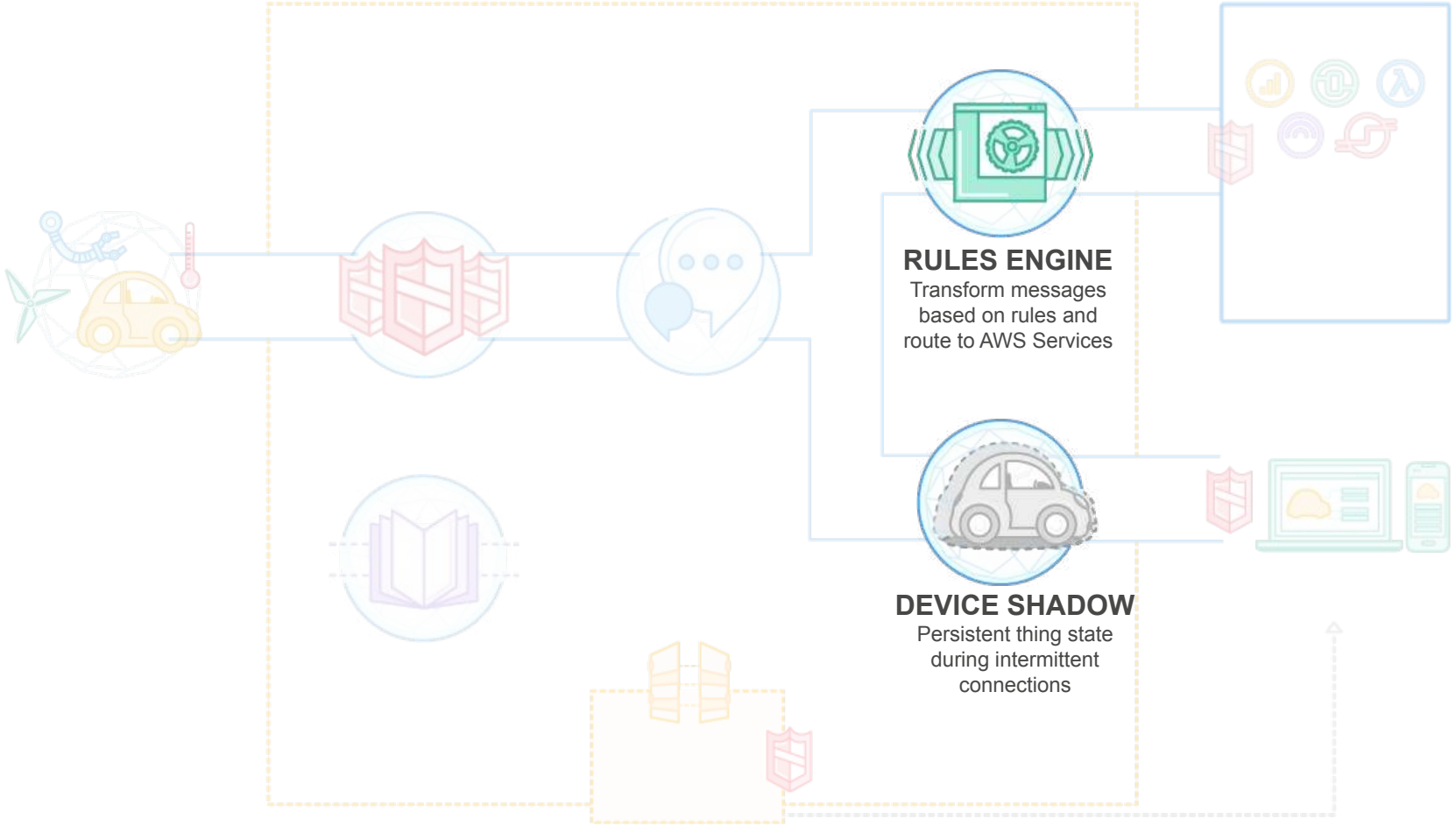
Table Creation

Hash Key : ${topic()}

Range Key : ${devicetimestamp}

amazon web services

Deep dive on AWS IoT



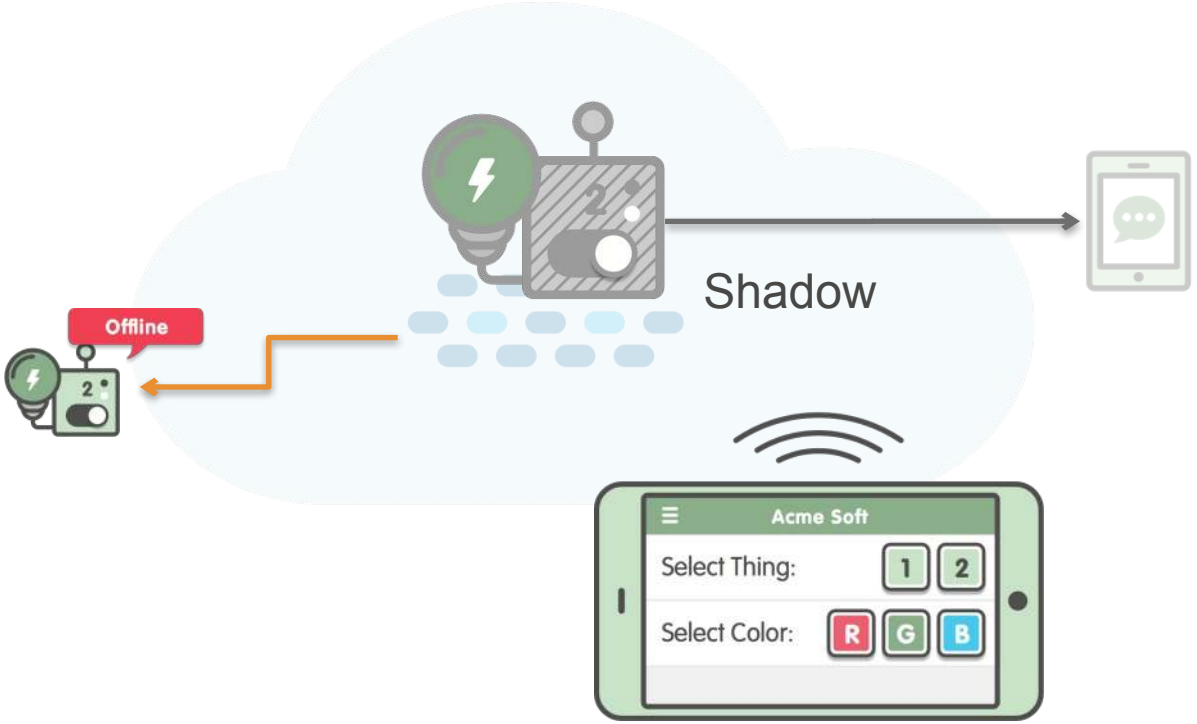
RULES ENGINE
Transform messages based on rules and route to AWS Services

DEVICE SHADOW
Persistent thing state during intermittent connections

AWS IoT Device Shadow

Virtual representation of your device in the cloud

- **Device State**
  - desired
  - reported
- **Device metadata**
  - **Sensors**
- **Version**
- **clientToken**
- **timestamp**



Protocols – AWS IoT Shadow

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

Thing

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

Shadow

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

Mobile/Web App

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

amazon web services

AWS IoT Shadow Flow

1. Device Publishes Current State

Shadow

AWS IoT

3. App requests device's current state

4. App requests change the state

5. Device Shadow sync's updated state

6. Device Publishes Current State

7. Device Shadow confirms state change

Device SDK

2. Persist JSON Data Store

amazon web services

Protocols – AWS IoT Shadow Topics (MQTT)

DEVICE SHADOW
Persistent thing state during intermittent connections

- **$aws/things/{thing}/shadow/…**

- **Publish**
- **…/get**: to get the latest shadow state
- **…/update**: to update the shadow state
- **…/delete**: to remove the shadow state

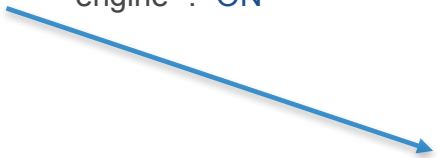
- **Subscribe**
- **…/accepted**: shadow accepted message
- **…/rejected**: shadow rejected message
- **…/delta**: differences between desired and reported

Protocols – AWS IoT Shadow Use Case

```
{
"state" : {
      “desired” : {
                  "engine" : "ON”,
                  “tires”: {
                        “LF”:40,
                        “RF”:38,
                        “LR”:37,
                        “RR”:39
                  },
                  “CCD”: {
                        “A”:0,
                        “B”:8,
                        “C”:7,
                        “D”:9
                  }

      },
      "reported" : {
                  "engine" : ”OFF”,
                  “tires”: {
                        “LF”:40,
                        “RF”:38,
                        “LR”:37,
                        “RR”:39
                  },
                  “CCD”: {
```

Protocols – AWS IoT Shadow Use Case

"engine" : ”ON”



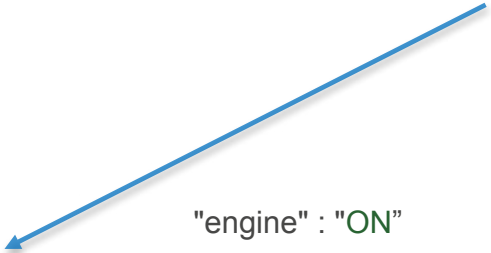
```
{
"state" : {
        “desired” : {
                "engine" : "ON”,
                “tires”: {
                        “LF”:40,
                        “RF”:38,
                        “LR”:37,
                        “RR”:39
                },
                “CCD”: {
                        “A”:0,
                        “B”:8,
                        “C”:7,
                        “D”:9
                }

        },
        "reported" : {
                "engine" : ”OFF”,
                “tires”: {
                        “LF”:40,
                        “RF”:38,
                        “LR”:37,
                        “RR”:39
                },
                “CCD”: {
```

Protocols – AWS IoT Shadow Use Case

"engine" : ”OFF”

```
{
"state" : {
        “desired” : {
                "engine" : "ON”,
                “tires”: {
                        “LF”:40,
                        “RF”:38,
                        “LR”:37,
                        “RR”:39
                },
                “CCD”: {
                        “A”:0,
                        “B”:8,
                        “C”:7,
                        “D”:9
                }

        },
        "reported" : {
                "engine" : ”OFF”,
                “tires”: {
                        “LF”:40,
                        “RF”:38,
                        “LR”:37,
                        “RR”:39
                },
                “CCD”: {
```

Protocols – AWS IoT Shadow Use Case



"engine" : "ON”

```
{
"state" : {
        “desired” : {
                        "engine" : "ON”,
                        “tires”: {
                                “LF”:40,
                                “RF”:38,
                                “LR”:37,
                                “RR”:39
                        },
                        “CCD”: {
                                “A”:0,
                                “B”:8,
                                “C”:7,
                                “D”:9
                        }

        },
        "reported" : {
                        "engine" : ”OFF”,
                        “tires”: {
                                “LF”:40,
                                “RF”:38,
                                “LR”:37,
                                “RR”:39
                        },
                        “CCD”: {
```

amazon web services

Protocols – AWS IoT Shadow Use Case

"engine" : "ON”

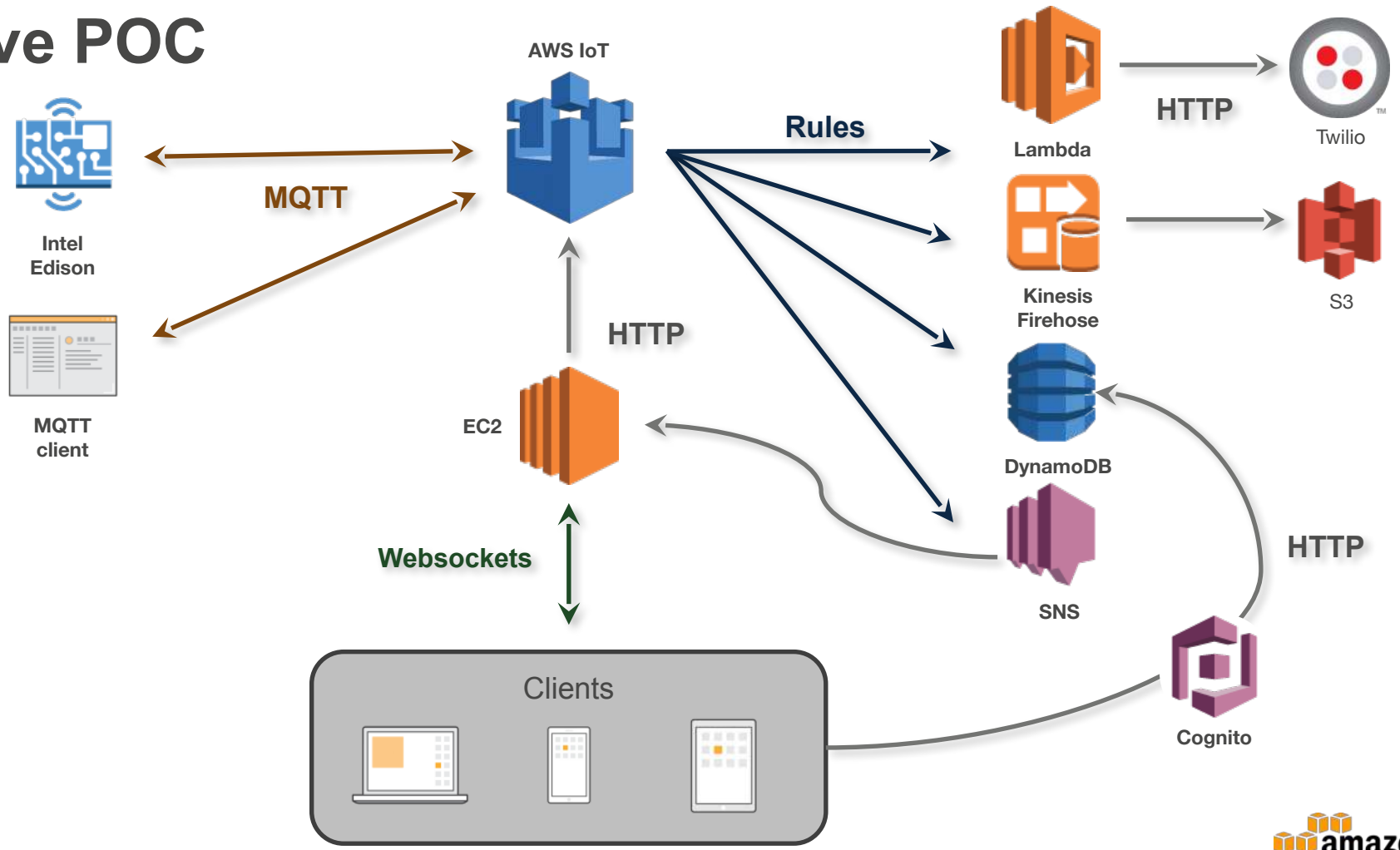
```
{
"state" : {
        “desired” : {
                "engine" : "ON”,
                “tires”: {
                        “LF”:40,
                        “RF”:38,
                        “LR”:37,
                        “RR”:39
                },
                “CCD”: {
                        “A”:0,
                        “B”:8,
                        “C”:7,
                        “D”:9
                }

        },
        "reported" : {
                "engine" : ”OFF”,
                “tires”: {
                        “LF”:40,
                        “RF”:38,
                        “LR”:37,
                        “RR”:39
                },
                “CCD”: {
```

amazon web services

Live POC Time !

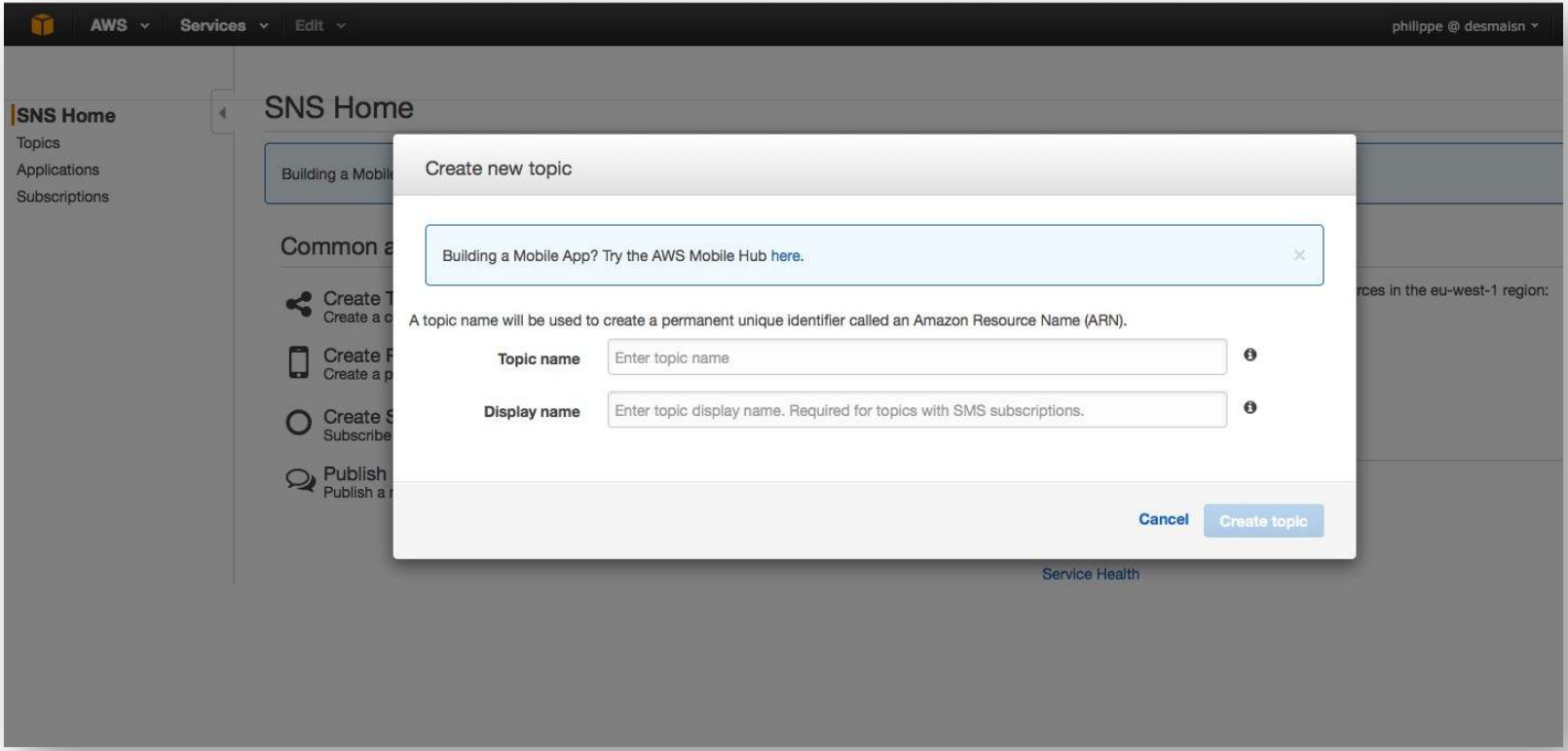
amazon web services

Live POC


AWS IoT
Intel Edison
MQTT
MQTT client
Rules
Lambda
HTTP
Twilio
Kinesis Firehose
S3
HTTP
EC2
DynamoDB
Websockets
SNS
HTTP
Clients
Cognito

Amazon SNS

Amazon SNS



Name of SNS Topic : edisonReportedUpdate

amazon web services

Edison.js

Node.js

```
//--- AWS IoT SDK module import and init ---//
const awsIot = require('aws-iot-device-sdk');

const name = 'Edison';
const config = require('./config.json');
const thingShadow= awsIot.thingShadow({
        keyPath: __dirname + '/certificates/private.pem.key',
        certPath: __dirname + '/certificates/certificate.pem.crt.txt',
        caPath: __dirname + '/certificates/rootCA.pem',
        clientId: thingName,
        region: config.iot_region
});

//--- Edison hardware modules import and init ---//
const five = require("johnny-five");
const Edison = require("edison-io");
const board = new five.Board({
        io: new Edison(),
        repl: false
});
```

Intel Edison

To be continued in a TextEditor

amazon web services

Website.js with Websockets

Node.js

To be continued in a TextEditor

EC2

```
 //--- Config and init ---//
const express = require('express.io');
const app = express();
app.http().io()

app.use(express.static(__dirname + '/static'));
app.listen(process.env.PORT || 80);

//--- Websocket events ---//
app.io.sockets.on('connection', function (client)
{
    console.log('New client (id: ' +  client.id + ') connected');
        client.emit('update_front', reported);

        client.on('update_back', function(data){
                updateEdison(JSON.parse(data));
                console.log('Client (id: ' + this.id + ') send update: ' + data);
        });

        client.on('disconnect', function() {
                console.log('Client (id: ' +  this.id + ') disconnected');
        });
});
```

amazon web services

Index.html with javascript

Index.html

To be continued in a TextEditor

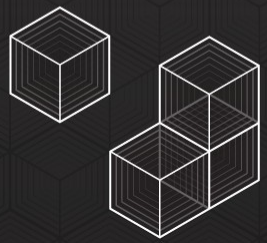
Browser

```
//----------------------//
  // Websocket led synchro //
  //----------------------//

  //-- Init websocket connection --//
  var socket = io.connect();

  //-- On message received through update_front --//
  socket.on('update_front', function(data) {
     if (typeof data.red !== 'undefined') {
        updateLed('red', data.red);
     }
     if (typeof data.green !== 'undefined') {
        updateLed('green', data.green);
     }
     if (typeof data.blue !== 'undefined') {
        updateLed('blue', data.blue);
     }
     console.log(socket.id, data);
  });
```

amazon web services

Thanks !

amazon web services



amazon web services