Introduction to AWS IoT

Zero to Hero

Mahesh Neelakanta, Florida Atlantic University
Ben Snively, Amazon Web Services

June 2015

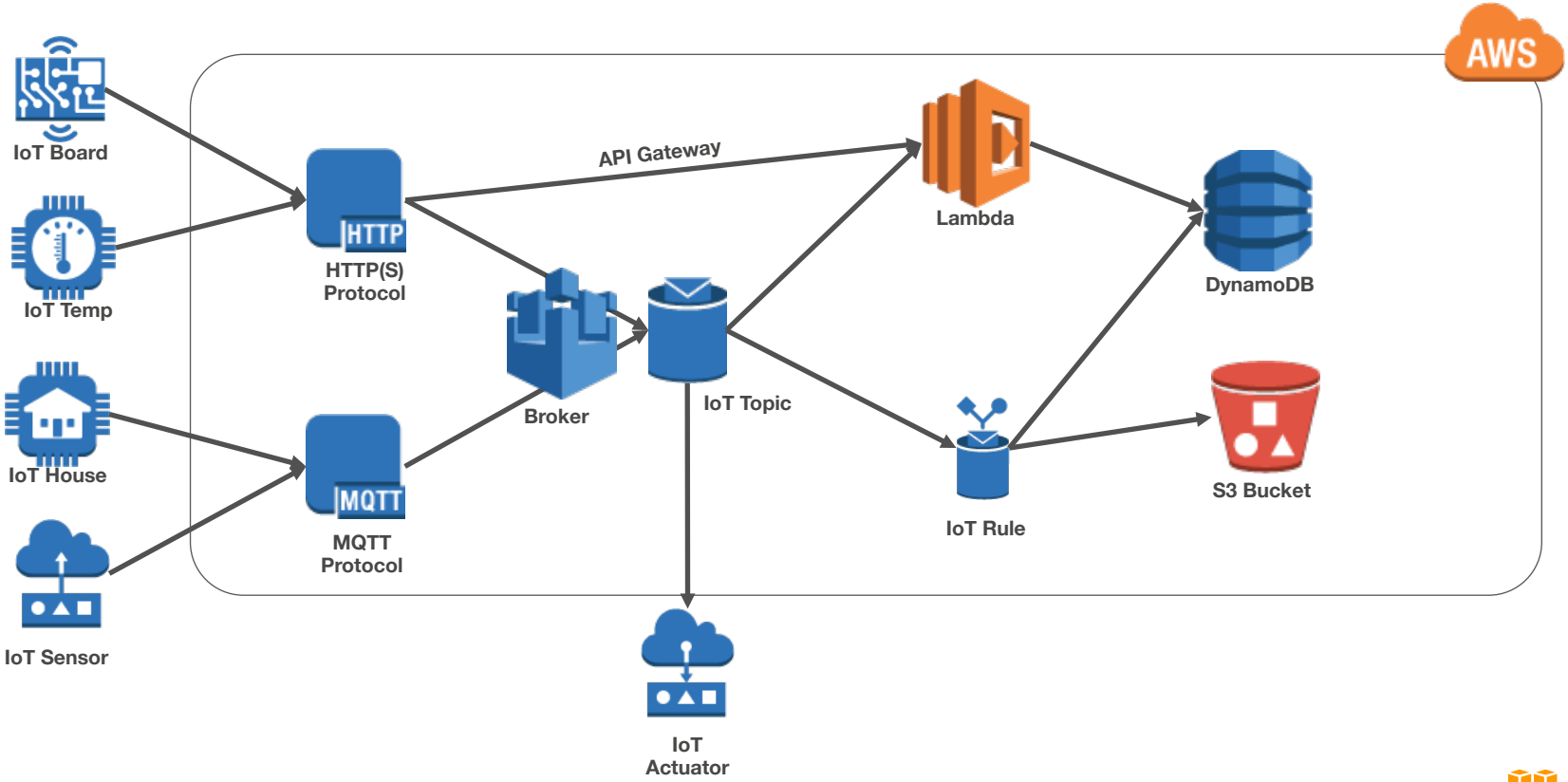
amazon web services

Agenda

- What is IoT?
- Why am I doing this?
- IoT Overview
- Serverless computing
- AWS Lambda
- AWS API Gateway
- AWS DynamoDB
- Use Cases
- Demo
- Q & A

What is IoT?

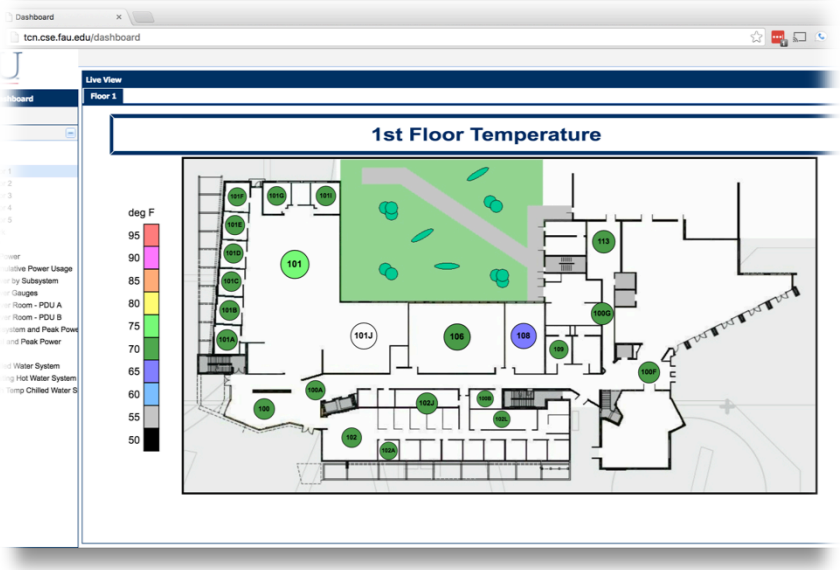
it's just a fancy "Message Queue" !



amazon web services

Why am I doing this?

FAU Green Building Metrics



What is my pool temperature?

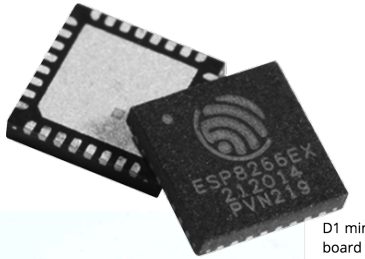
amazon web services

What can $4 USD get you these days?

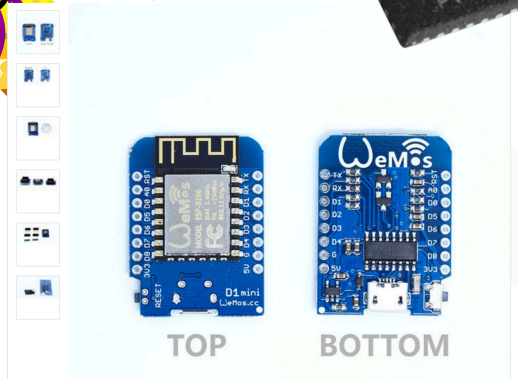
amazon web services

or

Or an Expressif ESP8266EX !!!!!



www.aliexpress.com



D1 mini V2 - Mini NodeMcu 4M bytes Lua WIFI Internet of Things development board based ESP8266 by WeMos

4.9 (5390 votes) | 7689 orders

Price: **US $4.00** / piece

Find more deals on the app

Shipping: Free Shipping to United States via China Post Ordinary Small Packet Plus

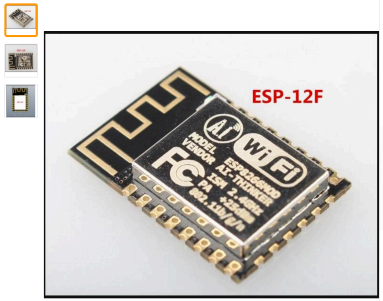
Estimated Delivery Time: 15-26 days (ships out within 7 business days)

Quantity: 1 piece (11637 pieces available)

Total Price: US $4.00

Buy Now | Add to Cart

Add to Wish List (2547 Adds)



2015 New version 1PCS ESP-12F (ESP-12E upgrade) ESP8266 remote seria... WIFI wireless module

5.0 (136 votes) | 828 orders

Price: ~~US $1.80 / piece~~

Discount Price: **US $1.71** / piece 5% off 2 days left

Find more deals on the app

Shipping: Free Shipping to United States via China Post Ordinary Small Packet Plus

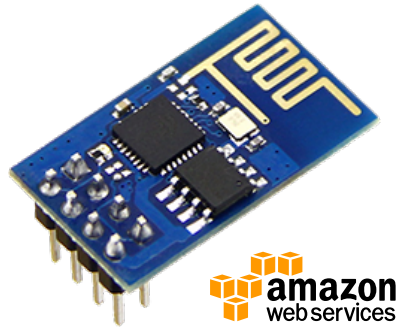
Estimated Delivery Time: 15-26 days (ships out within 7 business days)

Quantity: 1 piece (655 pieces at most per customer)

Total Price: US $1.71

Buy Now | Add to Cart

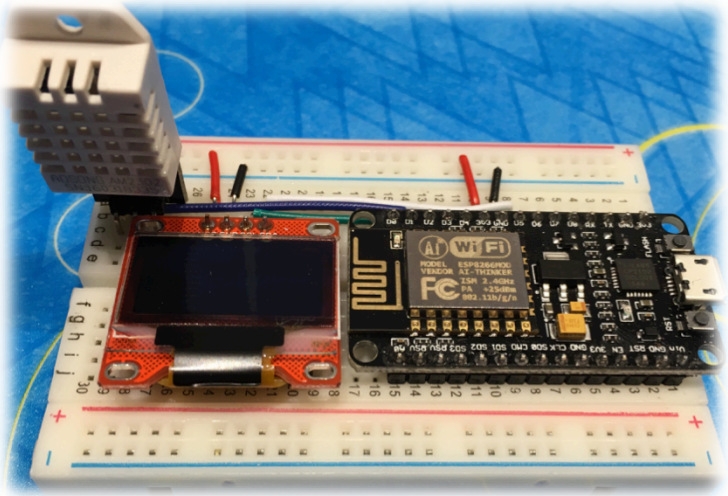
Add to Wish List (318 Adds)

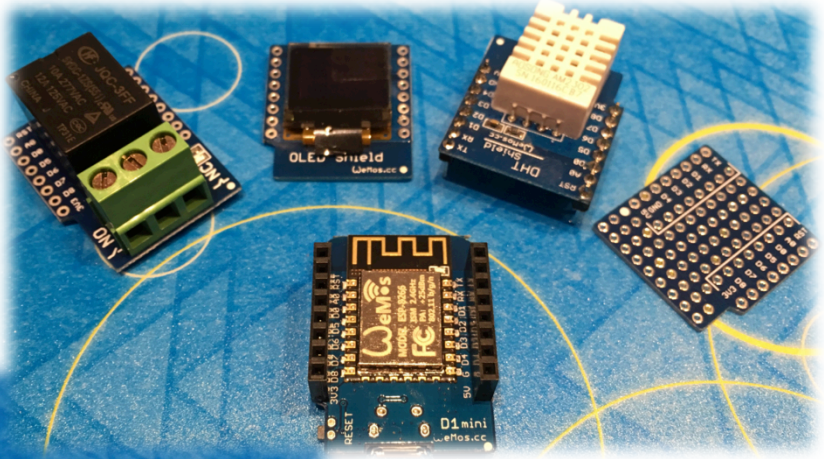


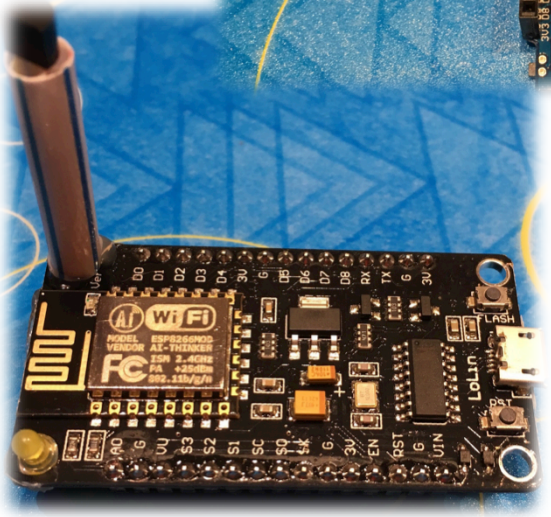
* iotawards.postscapes.com

amazon web services

Build your own or buy pre-built shields







amazon web services

ESP8266EX Strengths

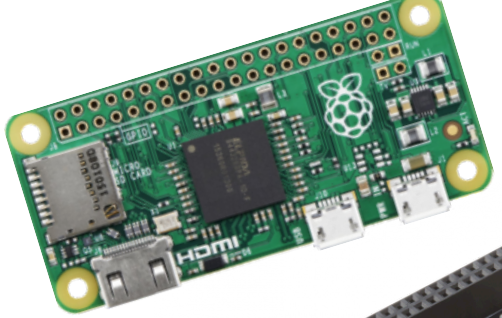
- Easy to program
  - LUA or Basic
  - Arduino Sketch
  - RTOS or C
- Standalone or co-processor
- Built-in TCP/IP Stack and Wi-Fi radio
- Ultra-CHEAP!

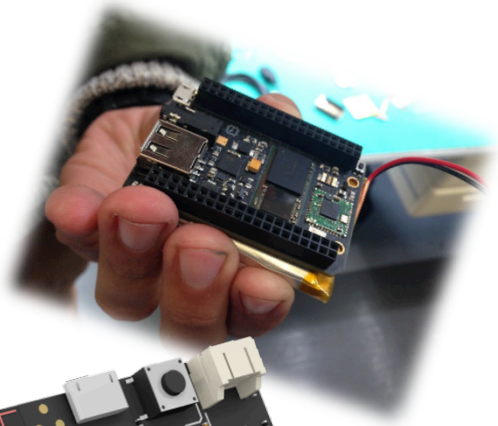
- Low power 32-bit MCU
- 10-bit ADC
- 802.11 b/g/n 2.4 GHz
  - AP or Station Mode
- SPI, UART, I2C, I2S, PWM, GPIO
- 11 I/O Pins
- <10 g (drones anyone?)

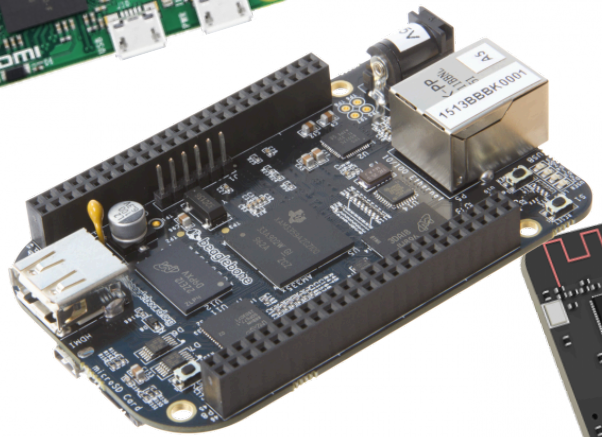
ESP8266EX Shortfalls

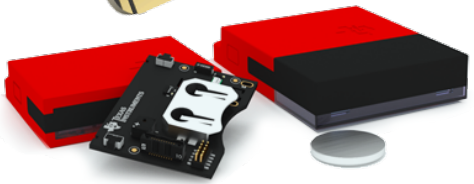
- Some libraries are not stable
- No support yet for TLS 1.2 (Only 1.1)
- Limited flexibility when compared to systems with an OS
- No built-in digital-analog converter
- But…
- It’s Ultra-CHEAP!

Other options









- C.H.I.P ($9)
- Raspberry PI ($5-25+)
- Beagle-bone ($50)
- Intel Edison ($50)
- BBC Micro Bit ($19)
- Particle Wi-Fi ($19) or GSM ($69)
- TI Sensortag ($30)
- Much more (atmel, digispark, arduino)
    - https://www.hackster.io/platforms/microcontroller

Example LUA code for ESP8266

Connect to the wireless network

```
print(wifi.sta.getip())
--nil
wifi.setmode(wifi.STATION)
wifi.sta.config("SSID","password")
print(wifi.sta.getip())
--192.168.18.110
```

Arduino like IO access

```
pin = 1
gpio.mode(pin,gpio.OUTPUT)
gpio.write(pin,gpio.HIGH)
gpio.mode(pin,gpio.INPUT)
print(gpio.read(pin))
```

HTTP Client

```
-- A simple http client
conn=net.createConnection(net.TCP, false)
conn:on("receive", function(conn, pl) print(pl) end)
conn:connect(80,"121.41.33.127")
conn:send("GET / HTTP/1.1\r\nHost:
www.nodemcu.com\r\n"
  .."Connection: keep-alive\r\nAccept: */*\r\n\r\n")
```

HTTP Server

```
-- a simple http server
srv=net.createServer(net.TCP)
srv:listen(80,function(conn)
  conn:on("receive",function(conn,payload)
  print(payload)
  conn:send("<h1> Hello, NodeMCU.</h1>")
  end)
end)
```

amazon web services

Example Arduino Sketch code for ESP8266

```
void setup() {
  // initialize serial communication at 115200 bits per second:
  Serial.begin(115200);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // Convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 3.2V):
  float voltage = sensorValue * (3.2 / 1023.0);
  // print out the value you read:
  Serial.println(voltage);
}
```

```
int inputPin = D3;  // pushbutton connected to digital pin D3
int val = 0;        // variable to store the read value

void setup() {
  pinMode(BUILTIN_LED, OUTPUT);  // set onboard LED as output
  pinMode(inputPin, INPUT);      // set pin as input
}

void loop() {
  val = digitalRead(inputPin);     // read the input pin
  digitalWrite(BUILTIN_LED, val);  // sets the LED to the button's value
}
```

```
void loop_web_test()
{
  const char* host = "www.google.com";
  int value = 0;

  delay(5000);

  Serial.print("connecting to ");
  Serial.println(host);

  // Use WiFiClient class to create TCP connections
  WiFiClient client;
  const int httpPort = 80;
  if (!client.connect(host, httpPort)) {
    Serial.println("connection failed");
    return;
  }

  // We now create a URI for the request
  String url = "/index.html";
  Serial.print("Requesting URL: ");
  Serial.println(url);

  // This will send the request to the server
  client.print(String("GET ") + url + " HTTP/1.1\r\n" +
               "Host: " + host + "\r\n" +
               "Connection: close\r\n\r\n");
  delay(500);

  // Read all the lines of the reply from server and print them to Serial
  while(client.available()){
    String line = client.readStringUntil('\r');
    Serial.print(line);
  }

  Serial.println();
  Serial.println("closing connection");
}
```

IoT Overview

What?

- Connect any “thing” to the internet (or intranet)
- Unidirectional or bi-Directional traffic-flow (publish/subscribe)
- Small amounts of data from one device
- Large amounts of data across 1000s of devices
- Mostly machine-to-machine (M2M)
- Time-series based data

- Temperature, humidity, soil moisture, luminance, on/off, speed, position, altitude, pressure, salinity, CO2, CO, air quality…
- Counters, gauges, levels, booleans…
- Sensor networks, vehicular networks, smart building,…
- Crowd sourced data

How?

Hardware

- Wi-Fi
- Bluetooth Low Energy (BLE)
- Serial
- USB
- XBee 2.4 GHz or 900 Hz
- nRF2401A 2.4 GHz
- Cellular
- “Browser”

Software

- MQTT Message Queue Telemetry Transport
- XMPP eXtensible Messaging and Presence Protocol
- CoAP Constrained Application Protocol
- HTTP(S) / REST
- UDP
- SMS
- Gateways and Proxies

Where?

Use a Platform

- AWS IoT
- Azure IoT
- IBM Bluemix
- IOT-Playground.com
- Telit IoT
- Vortex Cloud
- HiveMQTT
- thingspeak.com
- Sparkfun.com

Build your own

- Open Source MQTT Brokers (Mosquitto)
- Time Series Databases (TrailDB, DynamoDB, InfluxDB, Riak, Prometheus,…)
- Visualization tools (Grafana, Kibana, D3.js, ...)

When?

- Always “on” (but sleep to save power)
- Send data periodically (every minute, 5 minutes, 15 minutes, hourly, etc.)
- Retain last known state (shadows) in case of loss of connectivity

AWS IoT Implementation Details

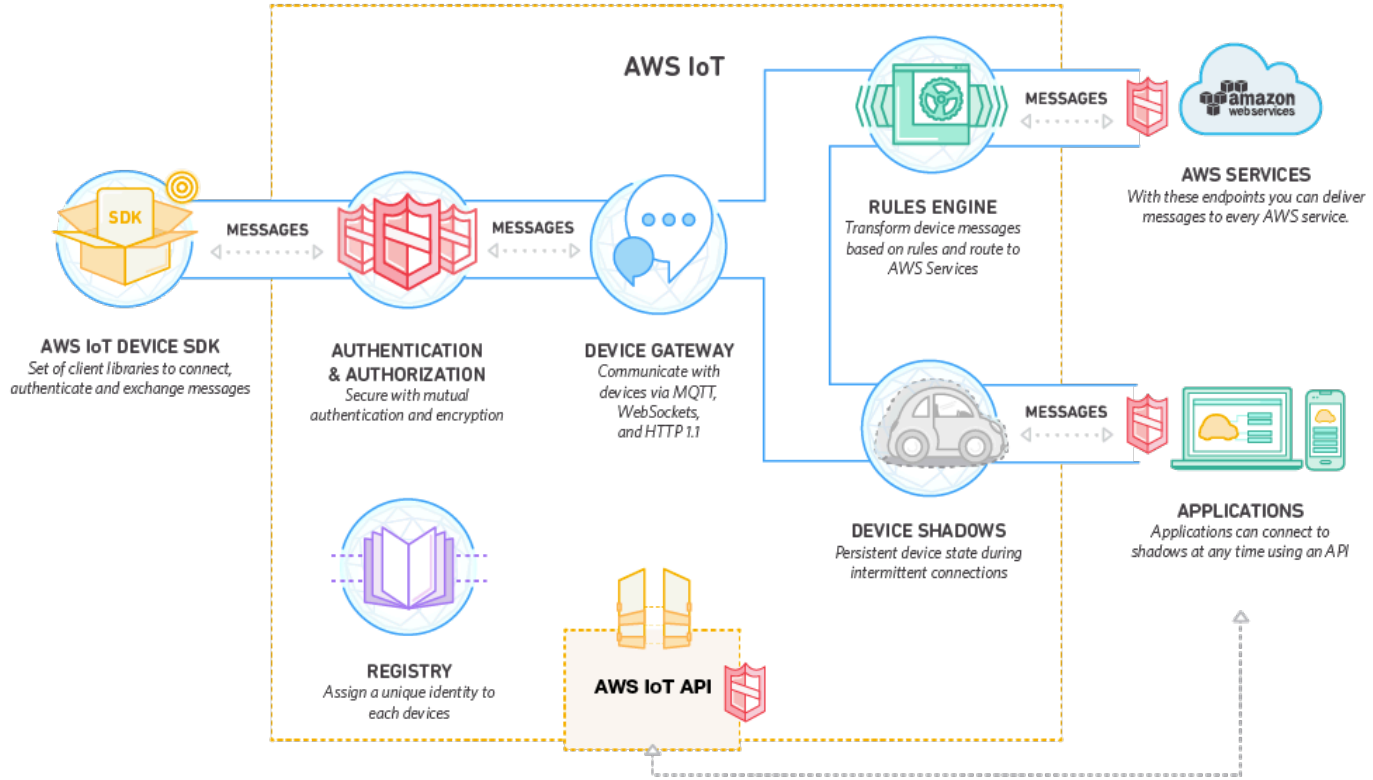
Service Features

- Provides access via MQTT, HTTPs (REST), Websockets
- Support for other “protocols” mentioned but not clarified
- Authentication and Authorization via X.509 certs or IAM key & secret or Cognito
- Device registry to track devices
- Device shadows to keep last-known-state
- Rules engine to “process, analyze and act” on data (ex: send to lambda or ddb)
- $5/million messages (512 byte max)
- 250k messages/month free
- Redundant and ‘serverless’

MQTT Features

- Publishers and subscribers
- Many to many
- Always “on”
- No built-in encryption
- Very lightweight
    - 2-5 byte header
    - topic name + body
    - 20-40 bytes tcp/ip v4

AWS IoT Implementation (Visual)



amazon web services

Serverless computing

What is serverless computing?

- VMs
  - Machine as the unit of scale
  - Abstracts the hardware
- Containers
  - Application as the unit of scale
  - Abstracts the OS
- Serverless
  - Functions as the unit of scale
  - Abstracts the language runtime

Amazon EC2

Amazon ECS

AWS Lambda

amazon web services

How do I choose?

- VMs
  - "I want to configure machines, storage, networking, and my OS" **Amazon EC2**
- Containers
  - "I want to run servers, configure applications, and control scaling" **Amazon ECS**
- Serverless
  - "Run my code when it's needed" **AWS Lambda**

amazon web services

AWS Lambda

Microservices and AWS Lambda

AWS Lambda + Amazon API Gateway is the *easiest* way to create microservices

- **Event handlers** one function per event type
- **Serverless backends** one function per API / path
- **Data processing** one function per data type

AWS Lambda: Serverless computing

Run code without servers. Pay only for the compute time you consume. *Be happy*.

Triggered by events or called from APIs:

- `PUT` to an Amazon S3 bucket
- Updates to Amazon DynamoDB table
- Call to an Amazon API Gateway endpoint
- Mobile app back-end call
- And many more…

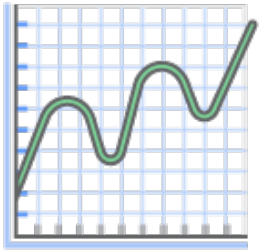
Makes it easy to:

- Perform real-time data processing
- Build scalable back-end services
- Glue and choreograph systems

amazon web services

Benefits of AWS Lambda

No servers to manage

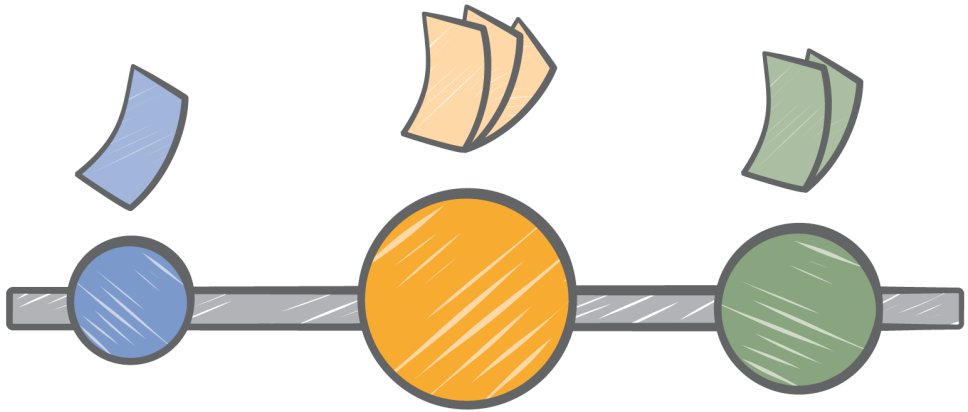


Continuous scaling

Never pay for idle – no cold servers (only happy accountants)

Pay-per request

- Buy compute time in 100 ms increments for 21 *micro*cents
- Request charge of 20 *micro*cents
- No hourly, daily, or monthly minimums
- No per-device fees



Free Tier

1 million requests and 400,000 GBs of compute every month, every customer

Lambda function to insert data to DDB

```
var AWS = require('aws-sdk');
var ddb = new AWS.DynamoDB();
var theContext;

exports.handler = function(event, context) {
    //uncomment if you want to debug in cloudwatch logs.
    console.log(event);
    theContext = context;
    messageArray = event;

    if (typeof(event.length) == "undefined") {
        messageArray = [event];
    }

    var itemsArray = [];

    for (i = 0; i < messageArray.length; i++) {
        var itemEntry = {
                "instance":{"N":messageArray[i].I},
                "timestamp":{"N":messageArray[i].T},
                "value":{"N":messageArray[i].V}
                };
        var item = {
                PutRequest: {
                 Item: itemEntry
                }
            };
        if (item) {
            itemsArray.push(item);
        }
    }
    var params = {
      RequestItems: {
        'fau_bocaraton_ee96' : itemsArray
      }
    };

    console.log("Putting items into DynamoDB: fau_bocaraton_ee96");
    //uncomment if you want to debug in cloudwatch logs.
    console.log(JSON.stringify(params));
    ddb.batchWriteItem(params, dynamoCallback);
}
```

Lambda function usage via cloudwatch logs

```
START RequestId: b4bc7494-2bf5-11e6-80d9-d7aeea5a302f Version: $LATEST
2016-06-06T14:48:18.157Z b4bc7494-2bf5-11e6-80d9-d7aeea5a302f [ { I: '3013276', V: '3', T: '1465224409' },
{ I: '3013277', V: '1', T: '1465224409' },
{ I: '3013278', V: '1', T: '1465224409' },
{ I: '3013279', V: '1', T: '1465224409' },
{ I: '3013284', V: '1', T: '1465224409' },
{ I: '3013285', V: '4', T: '1465224409' },
{ I: '3013300', V: '1', T: '1465224409' },
{ I: '3013302', V: '5', T: '1465224409' },
{ I: '3013303', V: '5', T: '1465224409' },
{ I: '3013304', V: '3', T: '1465224409' },
{ I: '3013306', V: '3', T: '1465224409' },
{ I: '3013307', V: '1', T: '1465224409' } ]
2016-06-06T14:48:18.158Z b4bc7494-2bf5-11e6-80d9-d7aeea5a302f Putting items into DynamoDB: fau_bocaraton_ee96
2016-06-06T14:48:18.158Z b4bc7494-2bf5-11e6-80d9-d7aeea5a302f{ "RequestItems": { "fau_bocaraton_ee96": [ { "PutRequest": { "Item": { "instance": { "N": "3013276" }
2016-06-06T14:48:18.231Z b4bc7494-2bf5-11e6-80d9-d7aeea5a302f Success
END RequestId: b4bc7494-2bf5-11e6-80d9-d7aeea5a302f
REPORT RequestId: b4bc7494-2bf5-11e6-80d9-d7aeea5a302f Duration: 73.96 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 47 MB
```

amazon web services

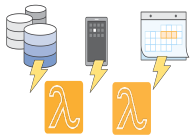
Using AWS Lambda

Bring your own code

- Node.js, Java, Python
- Bring your own libraries (even native ones)

Simple resource model

- Select power rating from 128 MB to 1.5 GB
- CPU and network allocated proportionately
- Reports actual usage



Flexible use

- Call or send events
- Integrated with other AWS services
- Build whole serverless ecosystems

Flexible authorization

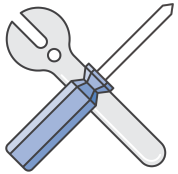
- Securely grant access to resources, including VPCs
- Fine-grained control over who can call your functions

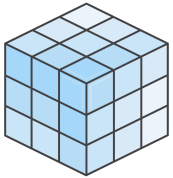
amazon web services

Using AWS Lambda

Programming model

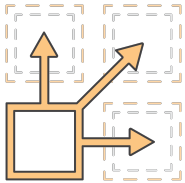
- AWS SDK built in (Python and Node.js)
- Eclipse plugin (Java)
- Lambda is the “webserver”
- Use processes, threads, / tmp, sockets normally





Stateless

- Persist data using Amazon DynamoDB, S3, or ElastiCache
- No affinity to infrastructure (can’t “log in to the box”)



Authoring functions

- Author directly using the console WYSIWYG editor
- Package code as a .zip and upload to Lambda or S3
- Plugins for Eclipse and Visual Studio
- Command line tools

Monitoring and logging

- Built-in metrics for requests, errors, latency, and throttles
- Built-in logs in Amazon CloudWatch Logs

But what *is* AWS Lambda?

Linux containers as an *implementation*, not a programming or deployment *abstraction*

- Process and network isolation, cgroups, seccomp, …

The world's biggest bin-packing algorithm

- High speed, highly distributed work routing and placement

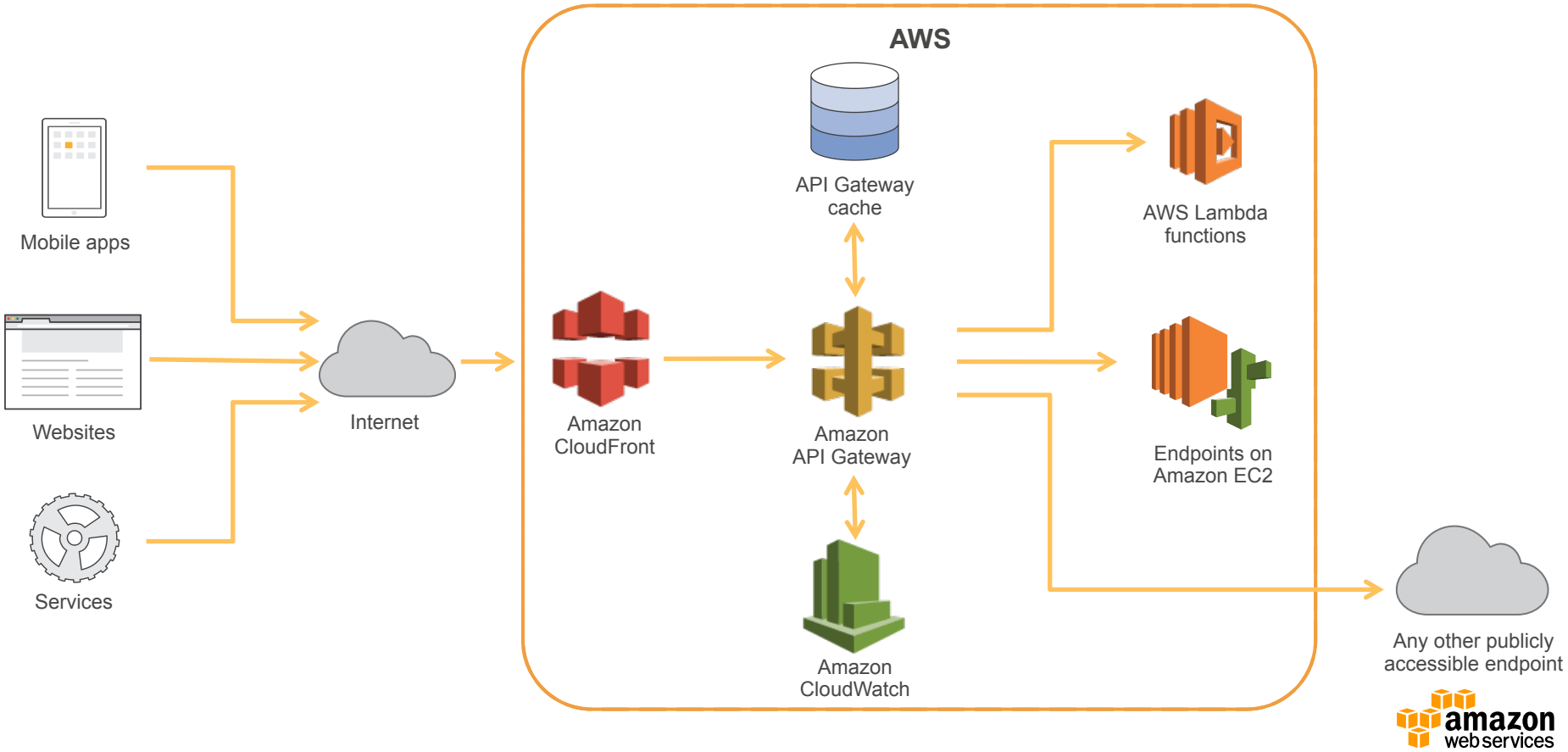
Predictive capacity management

- Purpose-built, massively scaled language runtime delivery service

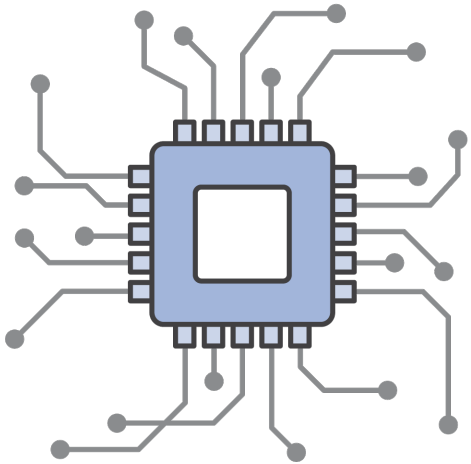
Swagger interpreter (API Gateway)

AWS API Gateway

Amazon API Gateway: Serverless APIs

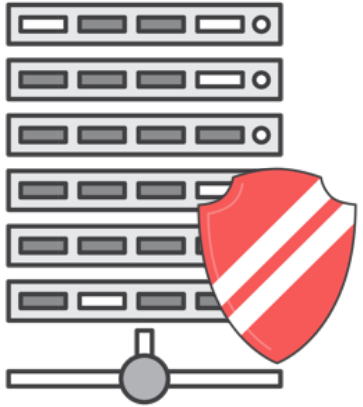


Benefits of Amazon API Gateway



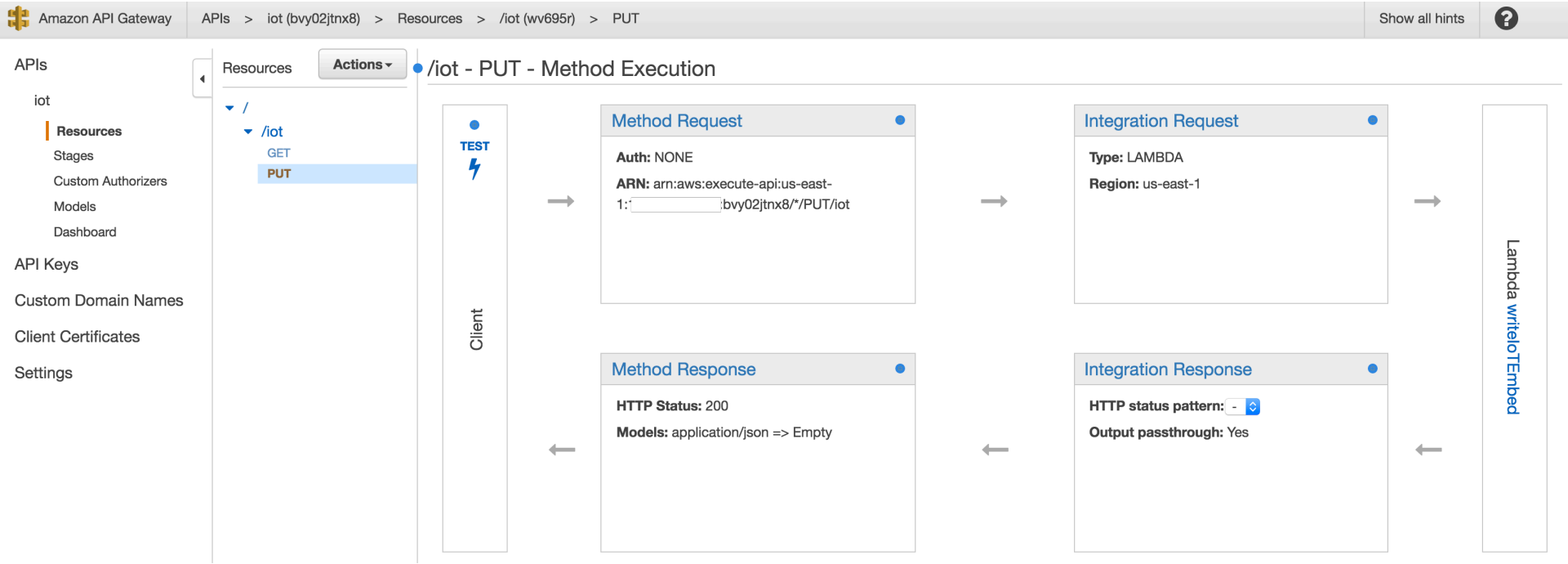
Create a unified API front end for multiple microservices

DDoS protection and throttling for back-end systems



Authenticate and authorize requests

API Gateway Example



amazon web services

Amazon DynamoDB

NoSQL database

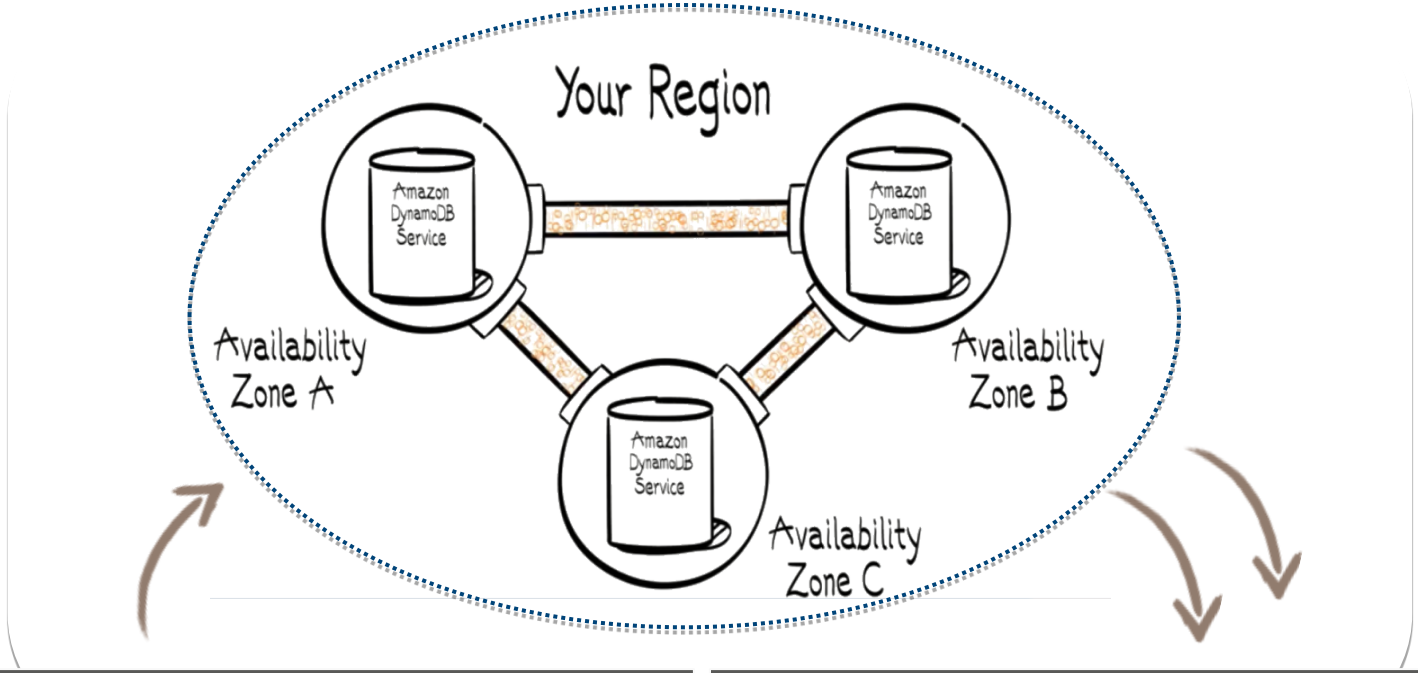
Fully managed

Single-digit millisecond latency

Massive and seamless scalability

Low cost

Automatic replication for rock-solid durability and availability



Writes

Replicated continuously to 3 AZs

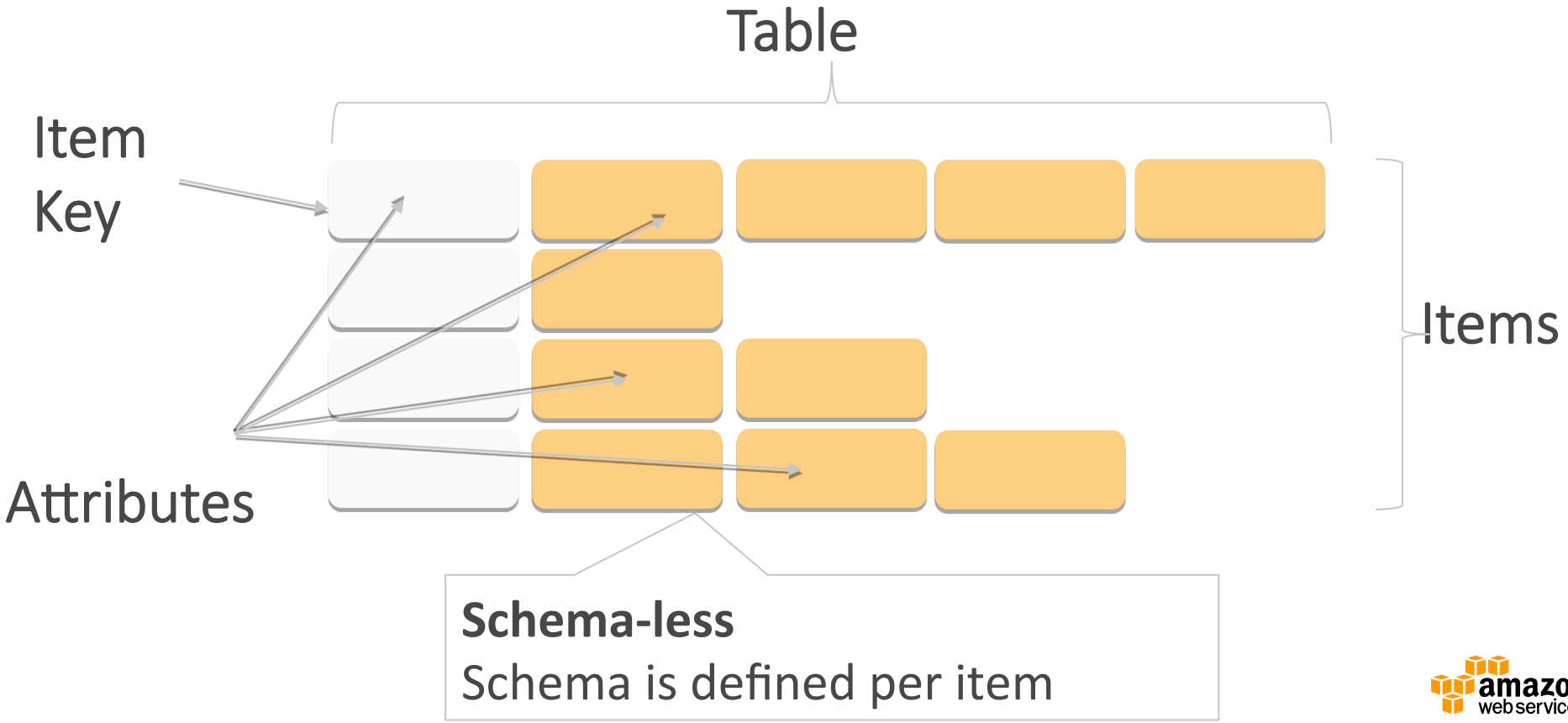
Persisted to disk (custom SSD)

Reads

Strongly or eventually consistent

No latency trade-off

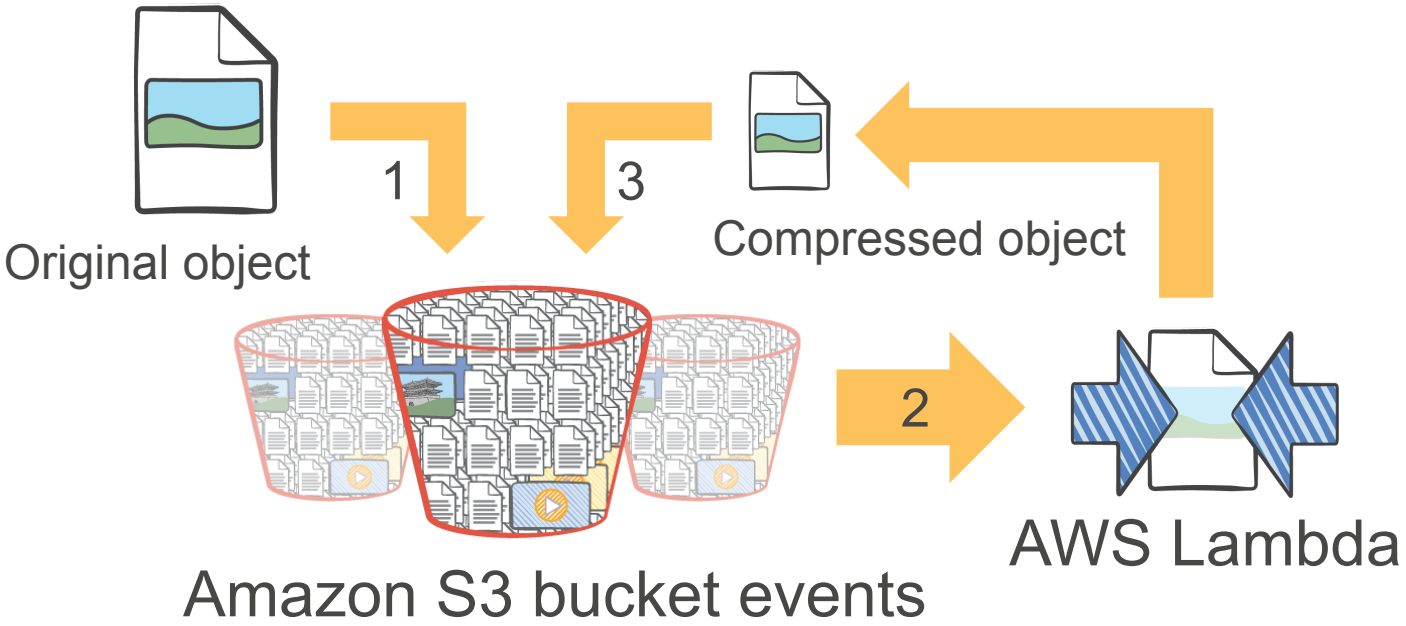
Amazon DynamoDB is a schemaless database



Use cases

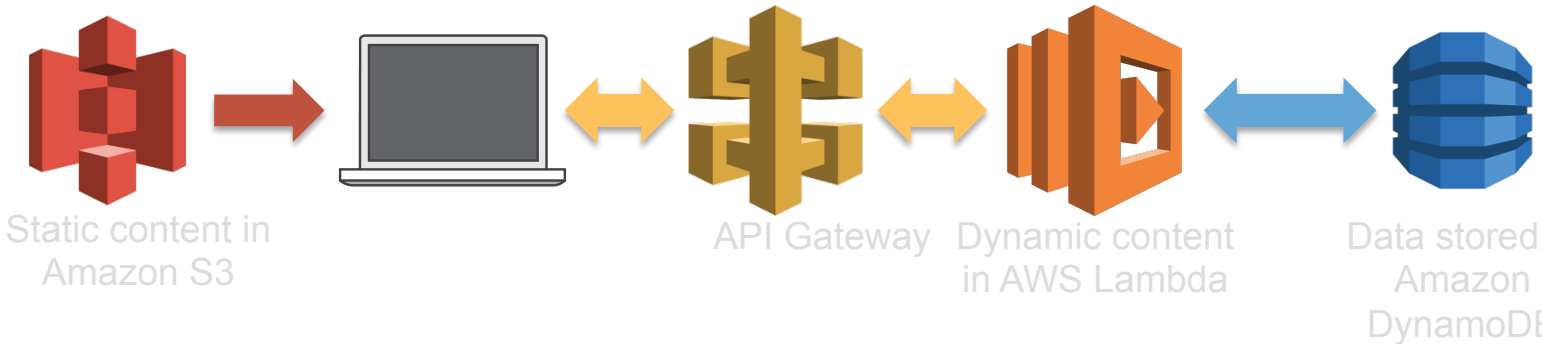
Lambda use case: Data processing

Example: Amazon S3 bucket triggers



Lambda use case: Serverless web apps

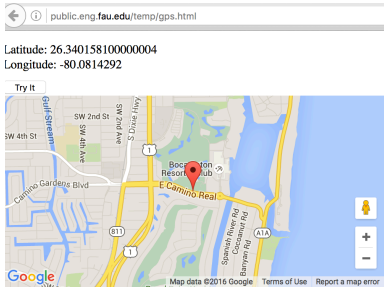
1. Amazon S3 for serving static content
2. AWS Lambda for dynamic content
3. Amazon API Gateway for https access
4. Amazon DynamoDB for NoSQL data storage



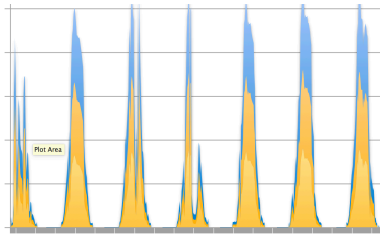
Lambda use case: IoT

- Process larger IoT Data Packets
  - Multiple data points within one packet (to save time/costs)
- Pre-processing before forwarding
  - Customized DB insertions
- Discarding or compressing data points

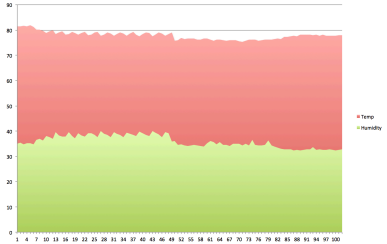
DEMO!



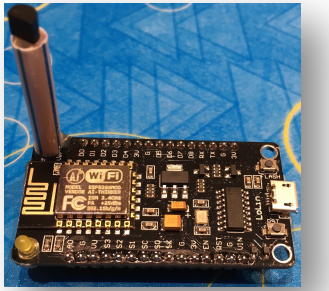
JavaScript GPS Page +AWS Console and MQTT.fx Subscriber



FAU C# BacNet and MQTT App + DynamoDB + Graph



ESP8266 and DHT22 Temperature + Rest API

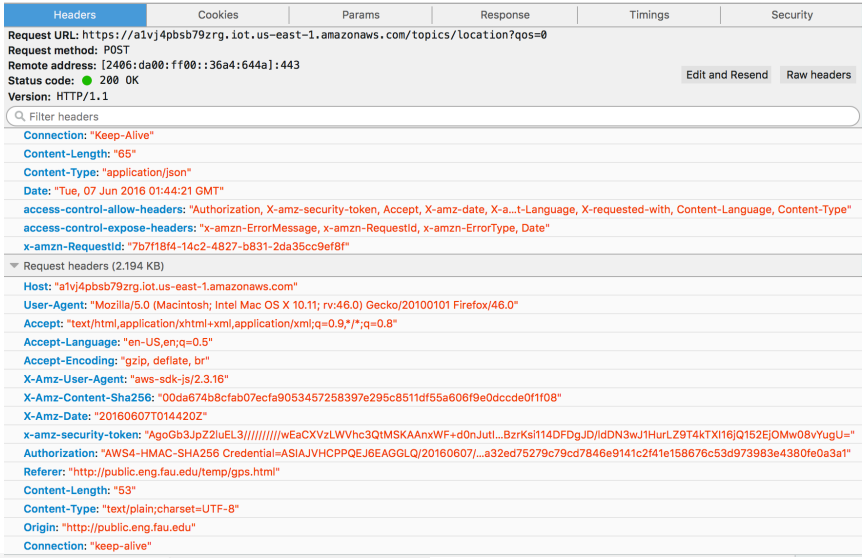


ESP8266 and 18B20 + Arduino IDE

JavaScript GPS

```
var creds = new AWS.CognitoIdentityCredentials({
    IdentityPoolId: 'us-east-1:0a115801-0884-4ca6-84c2-622640a2961d'
})

AWS.config.update({
    region: 'us-east-1',
    credentials: creds
});
```

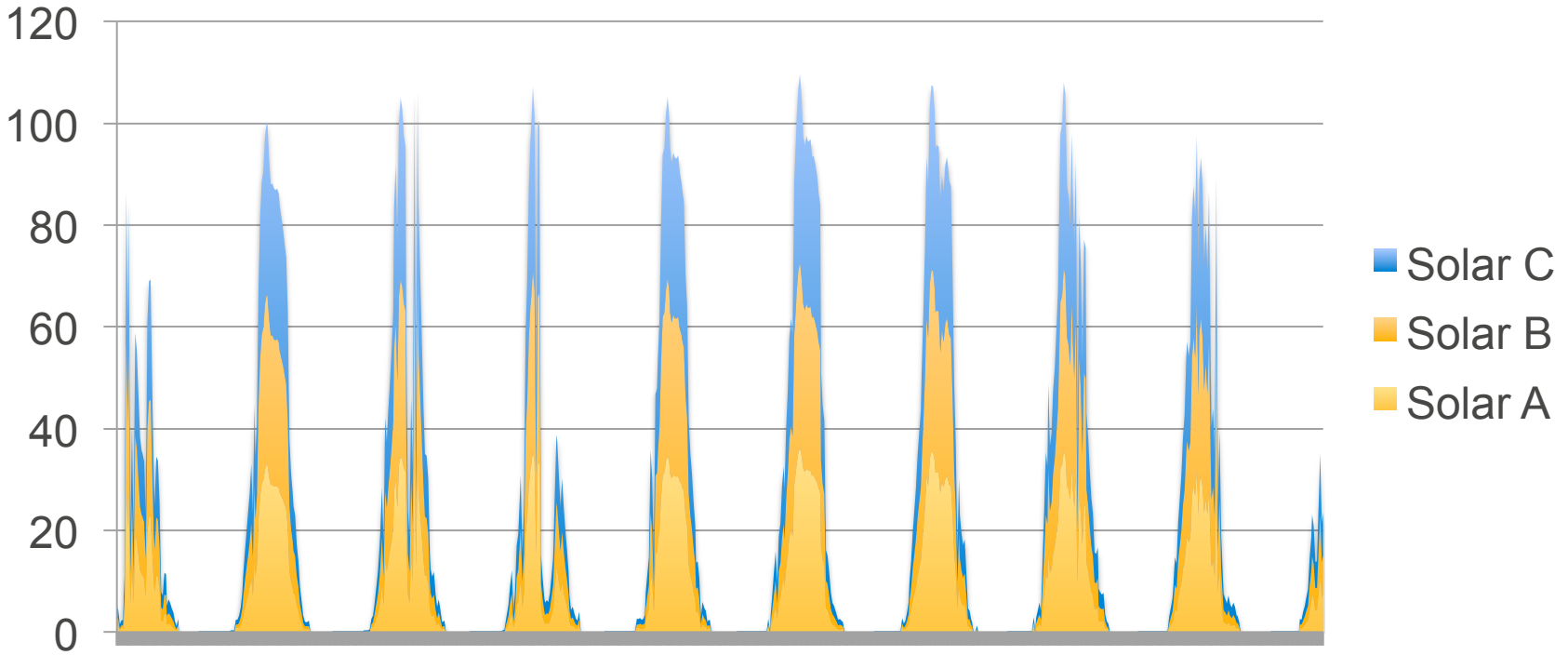
```
AWS.config.credentials.get(function(err) {
   if (err) {return console.error('Credentials error: ', err);}
   var credentials = AWS.config.credentials;
   cognito_accessKeyId = credentials.accessKeyId;
   cognito_secretAccessKey = credentials.secretAccessKey;
   cognito_sessionToken = credentials.sessionToken;
});
```




```
var iotdata = new AWS.IotData({
    endpoint: 'A1VJ4PBSB79ZRG.iot.us-east-1.amazonaws.com',
    accessKeyId : cognito_accessKeyId,
    secretAccessKey : cognito_secretAccessKey,
    sessionToken : cognito_sessionToken,
    region : 'us-east-1'
});

var params = {
    topic: 'location',
    payload: "{latitude: " + position.coords.latitude + ",longitude: " + position.coords.longitude + "}",
    qos: 0
};

iotdata.publish(params, function(err, data) {
    if (err) console.log(err, err.stack); // an error occurred
    else     console.log(data);           // successful response
});
```

amazon web services

FAU Green Building Solar Power Generation



amazon web services

FAU Green Building C# Code Snippets

```
public class DataPoint
{
    public string I { get; set; }   // Unique Instance ID
    public string V { get; set; }   // Current value read from controller
    public string T { get; set; }   // unix epoch (64 bit)
    public string to_json()
    {
        // Example: {"I":"3000027","V":"60.30528","T":"1463000642"}
        return ("{" + "\"I\":\"" + I + "\",\"V\":\"" + V + "\",\"T\":\"" + T + "\"}");
    }
}
```

```
using System.IO.BACnet;
using uPLibrary.Networking.M2Mqtt;
```

```
private const string IotEndpoint = "A1N1C2PVJQ95H0.iot.us-east-1.amazonaws.com";
private const int BrokerPort = 8883;
private const string Topic = "fau/bocaraton/ee96";
```

```
Stream stream_rootca = Assembly.GetEntryAssembly().GetManifestResourceStream("BacnetReadTest.aws_root_ca.pem");
Stream stream_iotcert = Assembly.GetEntryAssembly().GetManifestResourceStream("BacnetReadTest.b916134cda.pfx");
```

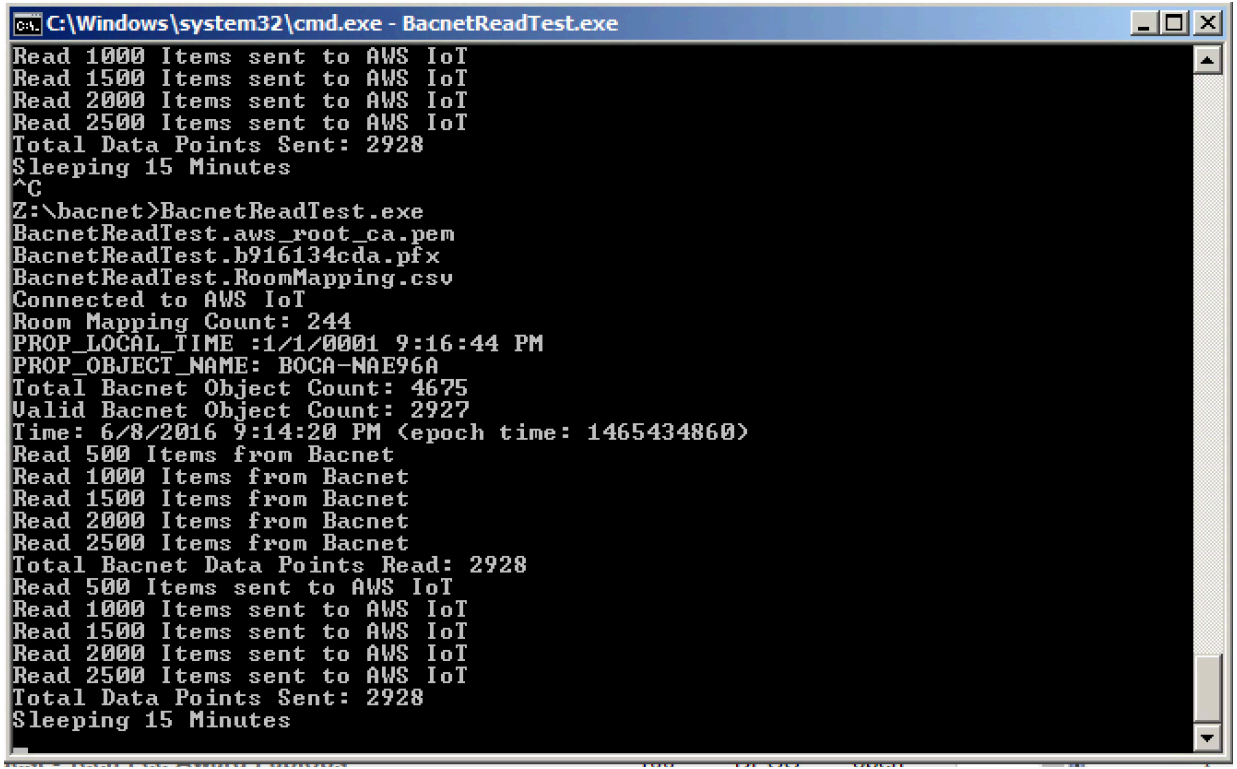
```
var clientCert = new X509Certificate2(StreamToByteArray(stream_iotcert));
var caCert = new X509Certificate(StreamToByteArray(stream_rootca));

MqttClient client = new MqttClient(IotEndpoint, BrokerPort, true, caCert, clientCert, MqttSslProtocols.TLSv1_2);
```

```
bacnet_client = new BacnetClient(new BacnetIpUdpProtocolTransport(0xBAC0, false));
bacnet_client.MaxSegments = BacnetMaxSegments.MAX_SEG65;
bacnet_client.DefaultSegmentationHandling = true;
bacnet_client.Start();
```

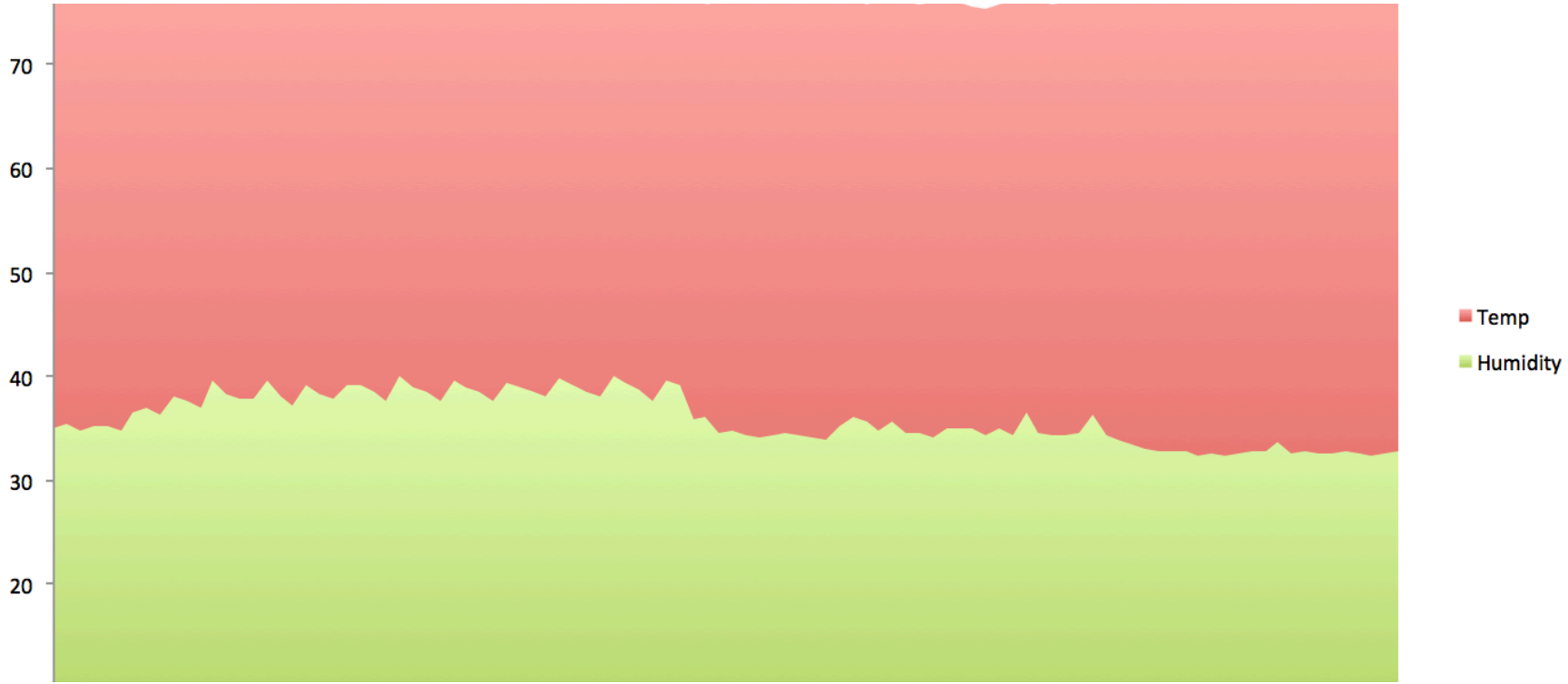
```
client.Publish(Topic, Encoding.UTF8.GetBytes(json_message));
```

amazon web services

C# BacNet to AWS IoT (via MQTT)



ESP8266 and DHT22 Temperature + Rest API



amazon web services

But what about the Pool Thermometer??

Still in development ☺

AWS Costs

Lambda		$0.00
US East (Northern Virginia) Region	**Usage**	
AWS Lambda Lambda-GB-Second		
AWS Lambda - Compute Free Tier - 400,000 GB-Seconds - US East (Northern Virginia)	13,373.375 seconds	$0.00
Total:		**$0.00**
AWS Lambda Request		
AWS Lambda - Requests Free Tier - 1,000,000 Requests - US East (Northern Virginia)	97,738 Requests	$0.00
Total:		**$0.00**
Region Total:		**$0.00**

IoT		$0.48
US East (Northern Virginia) Region	**Usage**	
AWS IoT USE1-AWSIoT-HTTP		
$5 per million messages	538 Messages	$0.01
Total:		**$0.01**
AWS IoT USE1-AWSIoT-MQTT		
$5 per million messages	92,548 Messages	$0.46
Total:		**$0.46**
AWS IoT USE1-AWSIoT-WEBSOCKET-MQTT		
$5 per million messages	645 messages	$0.01
Total:		**$0.01**
Region Total:		**$0.48**

API Gateway		$0.01
US East (Northern Virginia) Region	**Usage**	
Amazon API Gateway ApiGatewayRequest		
API Gateway Request - US East (Northern Virginia)	2,811 USE1-AmazonApiGateway-Request	$0.01
Total:		**$0.01**
Region Total:		**$0.01**

Bandwidth		
$0.000 per GB - data transfer in per month	0.510 GB	$0.00
$0.000 per GB - first 1 GB of data transferred out per month	1 GB	$0.00
$0.010 per GB - regional data transfer - in/out/between EC2 AZs or using IPs or ELB	0.230 GB	$0.01
$0.090 per GB - up to 10 TB / month data transfer out	0.172 GB	$0.02
Total:		**$0.03**
Region Total:		**$0.07**

Hardware Resources..

- Expressif EXP8266
  - https://espressif.com/en/products/hardware/esp8266ex/resources
- Kolban's book on ESP8266
  - http://neilkolban.com/tech/esp8266/
- Sparkfun (Thing board and kits)
  - https://www.sparkfun.com/products/13799
- Adafruit (Huzzah board and kits)
  - https://www.adafruit.com/products/2680
- Wemos (Mini D1 board and shields)
  - http://www.wemos.cc/
- Seedstudio (Wio Link or Node and Kits)
  - http://www.seeedstudio.com/
- Aliexpress (Chinese amazon.com line sales channel)
  - http://www.aliexpress.com/

Software Resources..

- NodeMCU (LUA)
    - http://nodemcu.com/index_en.html
- ESP8266 Basic
    - http://www.esp8266basic.com/
- ESP8266 Arduino Integration
    - https://github.com/esp8266/Arduino
- IOT-Playground (example esp8266 + temperature)
    - http://iot-playground.com/blog/2-uncategorised/41-esp8266-ds18b20-temperature-sensor-arduino-ide

Software and Technical Resources..

- Hackster (projects, code, diagrams, circuits galore!)
    - https://www.hackster.io
- C# BACnet SDK
    - https://sourceforge.net/projects/yetanotherbacnetexplorer/
- C# SDK for MQTT
    - https://github.com/ppatierno/m2mqtt
- AWS SDK for ESP8266
    - https://github.com/svdgraaf/aws-sdk-arduino
- Amazon IoT
    - https://aws.amazon.com/iot
- AWS IoT Developer Guide
    - http://docs.aws.amazon.com/iot/latest/developerguide/iot-dg.pdf
- ESP8266 Forum
    - http://www.esp8266.com/

Local Resources..

- Hacklabs
    - http://myhacklab.org/
- Maker faires
    - https://makerfairepalmbeach.com/
    - https://makerfairemiami.com/
    - http://www.infoworld.com/article/2972143/internet-of-things/real-time-protocols-for-iot-apps.html

IoT Resources..

- http://www.infoworld.com/article/2972143/internet-of-things/real-time-protocols-for-iot-apps.html
- http://www.allaboutcircuits.com/projects/introduction-to-the-mqtt-protocol-on-nodemcu/
- https://thingspeak.com/
- http://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt

Future Ideas

- Amazon Quicksight for BI and Visualization
- Non TLS based MQTT (lower cpu requirements)

Thank You!