Introduction to Edge ML with AWS IoT platform

Dr. Rajiv Misra

Professor, Dept. of Computer Science & Engg. Indian Institute of Technology Patna rajivm@iitp.ac.in

After Completion of this lecture you will knowing the following:

- Introduction to AWS IoT platform
- Layered architecture of AWS IoT
- Concepts of AWS IoT Core
- Understanding of AWS greengrass
- Event-Driven architecture with sensor data in AWS IoT

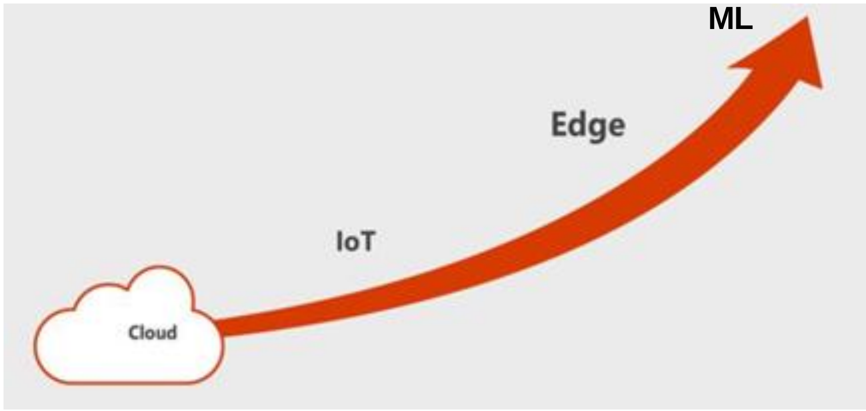
Recapitulate: Traditional IoT platform

Cloud
Globally available, unlimited compute resources

IoT
Harnessing signals from sensors and devices, managed centrally by the cloud

Edge
Intelligence offloaded from the cloud to IOT devices

ML
Breakthrough intelligence capabilities, in the cloud and on the edge



AWS IoT: Introduction

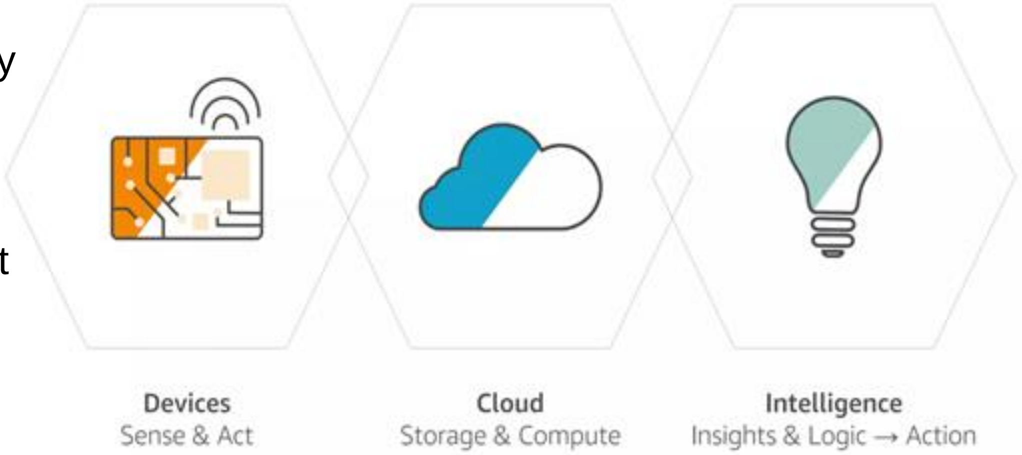
AWS IoT started in 2015 with Amazon acquiring a company called telemetry.

It started with several cloud services with a very simple IoT device management and M2M.

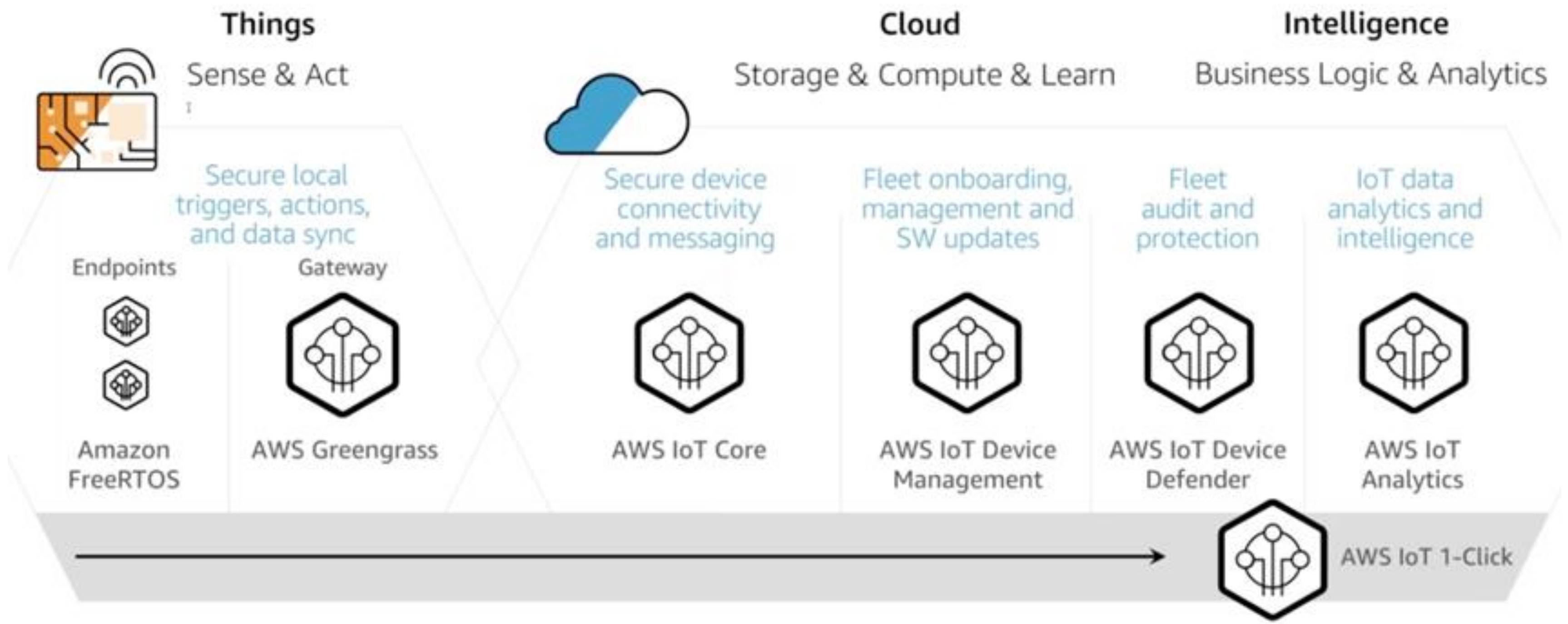
Now it has been expanded significantly.

AWS IoT architecture consists of three different layers:

- Things
- Cloud
- Intelligence

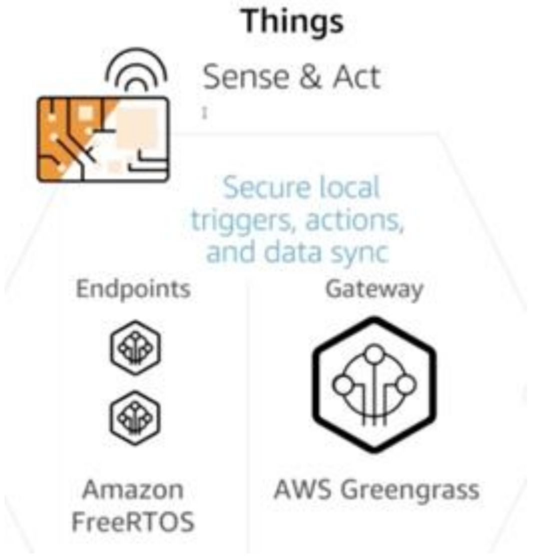


AWS IoT Architecture: Services Suite


Things
Sense & Act
Cloud
Storage & Compute & Learn
Intelligence
Business Logic & Analytics
Secure local triggers, actions, and data sync
Endpoints
Gateway
Amazon FreeRTOS
AWS Greengrass
Secure device connectivity and messaging
AWS IoT Core
Fleet onboarding, management and SW updates
AWS IoT Device Management
Fleet audit and protection
AWS IoT Device Defender
IoT data analytics and intelligence
AWS IoT Analytics
AWS IoT 1-Click

AWS IoT Architecture: Things

- Things comprises of components which are on premises at the field and on the devices side, which actually sense data and act.
- Amazon offers a couple of products for this layer.
- First one is Amazon FreeRTOS which is a real-time operating system that can run on top of a microcontroller with 64 KB of memory or more
- Then AWS greengrass which is the edge computing software act as a interfacing with the local devices running either Amazon FreeRTOS or the AWS IoT devices SDK



AWS IoT Architecture: Cloud

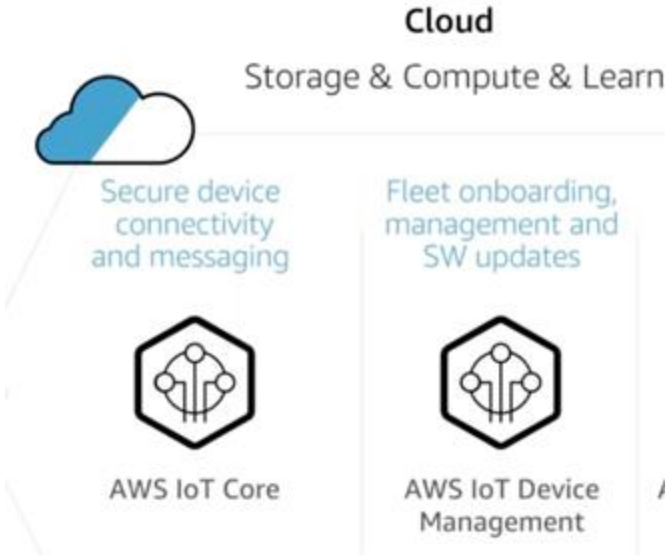
When it comes to cloud there are two important aspects:

The first one is **AWS IoT core** and as the name suggests it is the core building block of the AWS IoT platform and is responsible for registering the devices, so it acts as the device registry.

It also exposes endpoints for MQTT WebSockets and HTTP for the devices to talk to each other and to talk to the cloud and it is also the touch point for applications that want to control the devices running in the field.

AWS IoT core acts as an interface between the applications for example a mobile app that is talking to a device and similarly a device that is sending sensor data to the cloud.

Second one is **IoT device management** which supports bulk onboarding of devices because registering one device at a time in industrial use cases is not feasible, so it supports bulk onboarding and also has properties like over-the-air software updates, maintenance and performing bulk jobs, operations and so on.

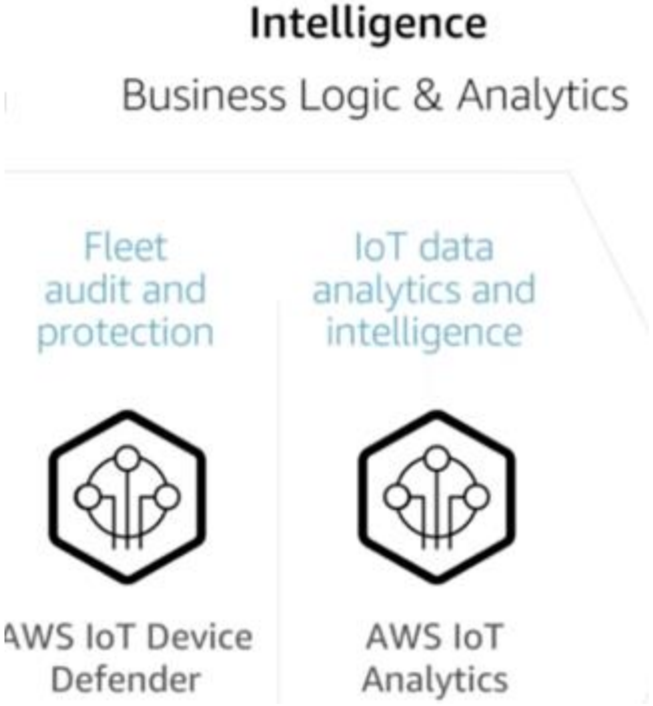

Cloud
Storage & Compute & Learn
Secure device connectivity and messaging
Fleet onboarding, management and SW updates
AWS IoT Core
AWS IoT Device Management

AWS IoT Architecture: Intelligence

AWS IOT device defender is all about security and if there is a drift between the preferred configuration and the policy and what is currently running and it automatically raises alerts.

It also maintains a highly secure footprint of all the devices and if there is any anomaly it raise an alert so that is the fleet audit or protection service.

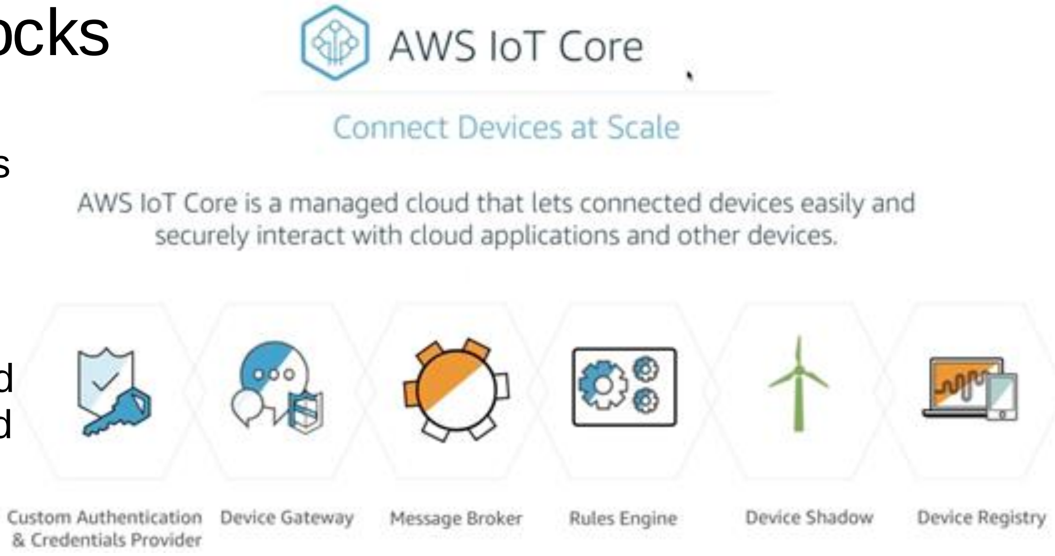
Finally, AWS IoT analytics which is an analytic solution and this service is responsible for analyzing the trends, visualizing and from there feeding to more powerful systems like quick site or redshift and so on.


Intelligence
Business Logic & Analytics
Fleet audit and protection
IoT data analytics and intelligence
AWS IoT Device Defender
AWS IoT Analytics

AWS IoT Core: Building Blocks

AWS IOT core is all about connecting devices to the cloud, the moment you bring in your first device that is going to become available you need to talk to AWS IoT core.

The workflow is very straightforward you need to register your device with AWS IoT core and that is going to act as the digital identity of your device.


AWS IoT Core
Connect Devices at Scale
AWS IoT Core is a managed cloud that lets connected devices easily and securely interact with cloud applications and other devices.
Custom Authentication & Credentials Provider
Device Gateway
Message Broker
Rules Engine
Device Shadow
Device Registry

The moment you register a device you receive a set of credentials for the device and you're going to embed those credentials in the device and once the device has those credentials and it connects to the cloud it gets authenticated, authorized and it shows up in the device registry.

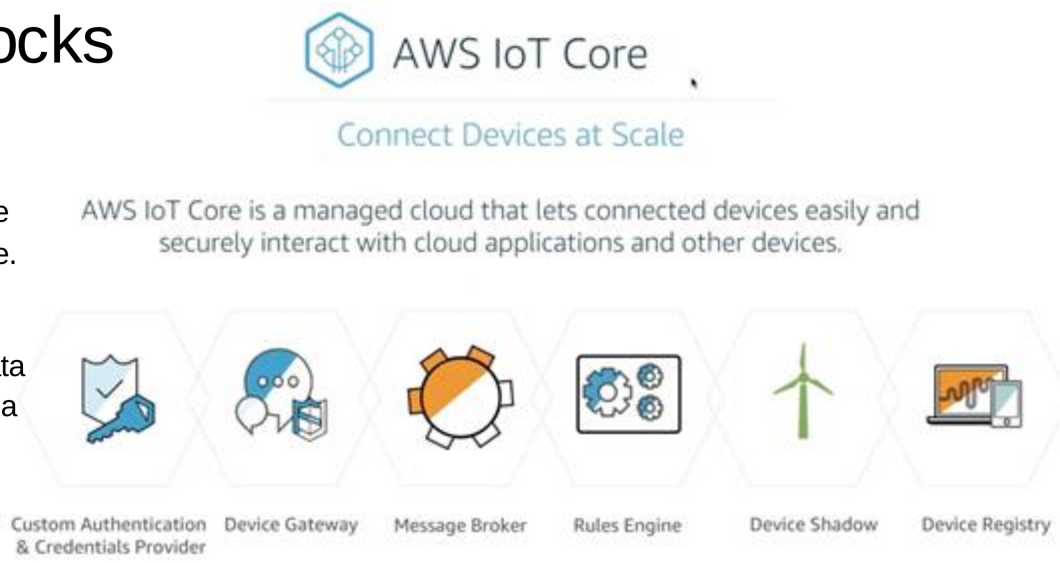
The device could be running a microcontroller, a single board computer, a slightly more powerful machine that can talk to an Modbus or canvas internally or, even an automobile device like a car.

After that it can send messages to the cloud and it can receive commands from the cloud.

AWS IoT Core: Building Blocks

When you zoom into AWS IoT core, the first one is all about authentication and authorization and the second one is device gateway which is the cloud endpoint for talking to the IoT core.

Message broker which is based on MQTT WebSockets and HTTP for publishing and subscribing messages or feeding data from the device to the cloud but it is predominantly uses it for a communication between devices at the cloud to send some metadata or telemetry and to receive some settings or commands.


AWS IoT Core
Connect Devices at Scale
AWS IoT Core is a managed cloud that lets connected devices easily and securely interact with cloud applications and other devices.
Custom Authentication & Credentials Provider
Device Gateway
Message Broker
Rules Engine
Device Shadow
Device Registry

There is a rules engine which decides how the messages will flow into rest of the system and the rules engine is ANSI SQL compliant that writes simple select statements that will filter the messages and apply a rule and the outcome of this rule can be hooked to a lambda function to take further action.

The device shadow is the digital twin or digital identity of the physical device and all the changes that are made to the device will first get applied to the device shadow and then it gets propagated all the way to the device. When the device state changes it automatically gets synchronized with the device shadow. It acts as the buffer between the desired state and the current state.

The job of the AWS IoT core is to make sure that the desired configuration is matching with the current configuration or not.

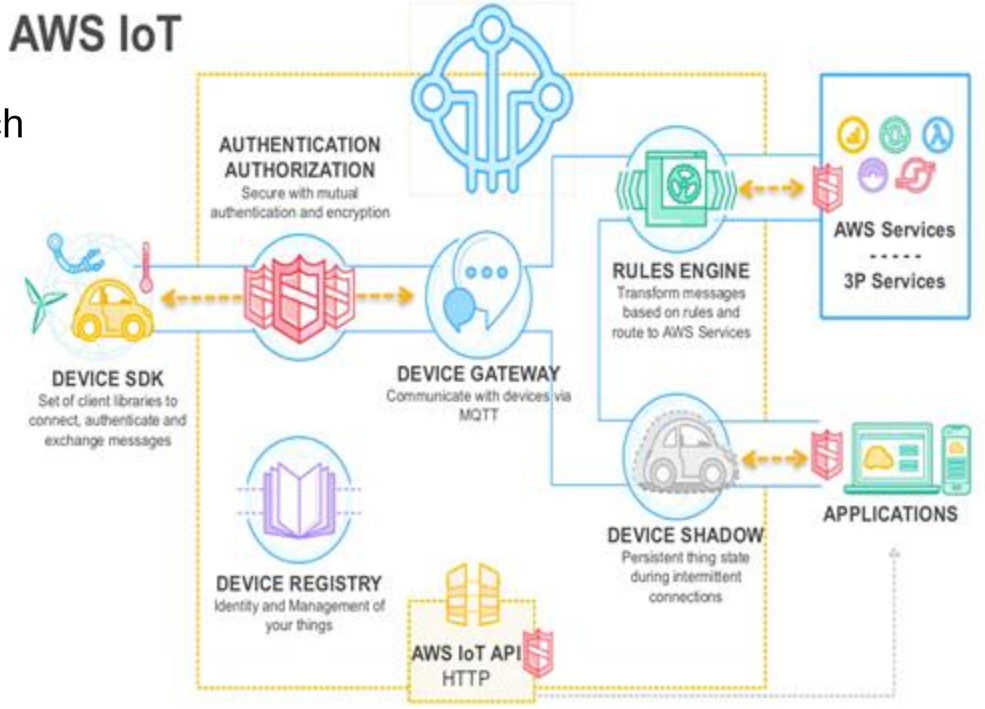
Device registry is a huge database repository but meant for the devices or things that you connect to AWS IoT.

AWS IoT Core: Summary

To put things in perspective, for using multiple building blocks of AWS IoT, the device SDK which is supported in variety of languages like node.js, Python, C, Java where SDK is used to connect your device to the cloud.

The first touch point is authentication and authorization and then the device gateway for communication and further it goes to a rules engine and device shadow which maintains a replica of this state

The rules engine is responsible for extending the IoT platform to rest of AWS services like dynamo DB, Neptune, redshift, AWS sage maker and to third-party services.

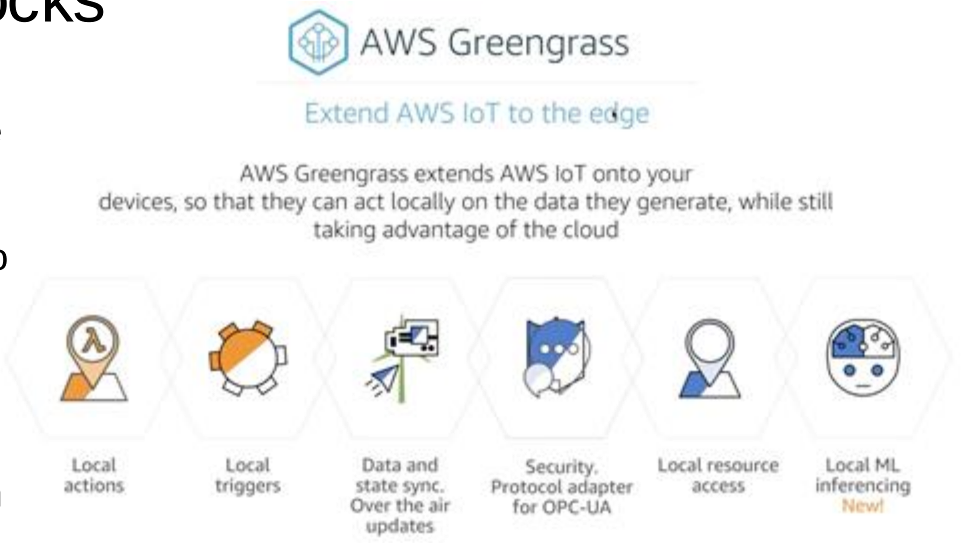

AWS IoT
AUTHENTICATION AUTHORIZATION
Secure with mutual authentication and encryption
DEVICE SDK
Set of client libraries to connect, authenticate and exchange messages
DEVICE GATEWAY
Communicate with devices via MQTT
RULES ENGINE
Transform messages based on rules and route to AWS Services
AWS Services

3P Services
DEVICE SHADOW
Persistent thing state during intermittent connections
APPLICATIONS
DEVICE REGISTRY
Identity and Management of your things
AWS IoT API
HTTP

AWS Greengrass: Building Blocks

AWS Greengrass extends AWS IOT to your devices so they can act locally and the data that they generate or filter is filtered before it is sent to the cloud.

Like AWS IoT core there is a message broker built into green grass so devices can continue to talk to each other, there is a compute layer which is based on lambda to write functions that are running locally and triggered when a specific condition is met and these triggers will actually fire lambda functions that perform an action.


AWS Greengrass
Extend AWS IoT to the edge
AWS Greengrass extends AWS IoT onto your devices, so that they can act locally on the data they generate, while still taking advantage of the cloud
Local actions
Local triggers
Data and state sync. Over the air updates
Security. Protocol adapter for OPC-UA
Local resource access
Local ML inferencing New!

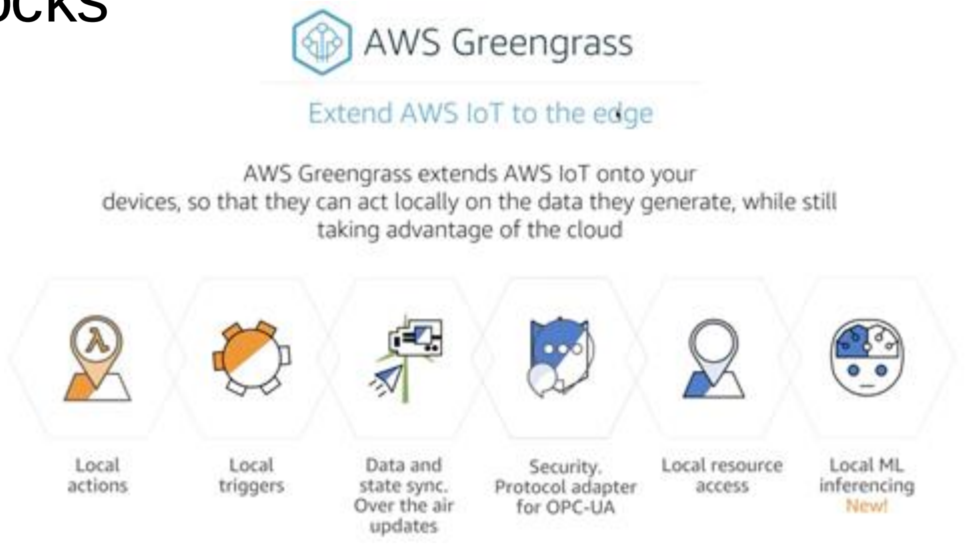
Greengrass also have the data and state synchronized with the cloud with the help of local device shadows and the cloud device shadow. If something updated locally, it first gets written to the device shadow running on the edge and then it eventually gets synchronized with the cloud.

Greengrass provides local resource access. For example, you want to talk to a local database which already has some metadata or material asset tracking information you can you can query that directly, talk to the file system, databases or anything that is accessible within the network.

AWS Greengrass: Building Blocks

The most recent feature of greengrass is the ability to run machine learning inferencing on the edge and this is one of the key drivers because there are three aspects when it comes to IOT.

First one is the learning part which is happening in the cloud, where you train machine learning models then you have decision-making that takes place at the edge and where fully trained machine learning models are used and they make decisions on behalf of the cloud and the action phase that is directly done by the devices.


AWS Greengrass
Extend AWS IoT to the edge
AWS Greengrass extends AWS IoT onto your devices, so that they can act locally on the data they generate, while still taking advantage of the cloud
Local actions
Local triggers
Data and state sync. Over the air updates
Security. Protocol adapter for OPC-UA
Local resource access
Local ML inferencing New!

For example, a machine learning model trained in the cloud to find an anomaly is deployed on the edge and because an anomaly is found with a very critical device, the machine learning model decides that one of the other equipments need to be shut down and that decision will result in an action where an actuator or a relay or another interface physically shuts down a malicious or a vulnerable machine to avoid any eventuality or any fatalities.

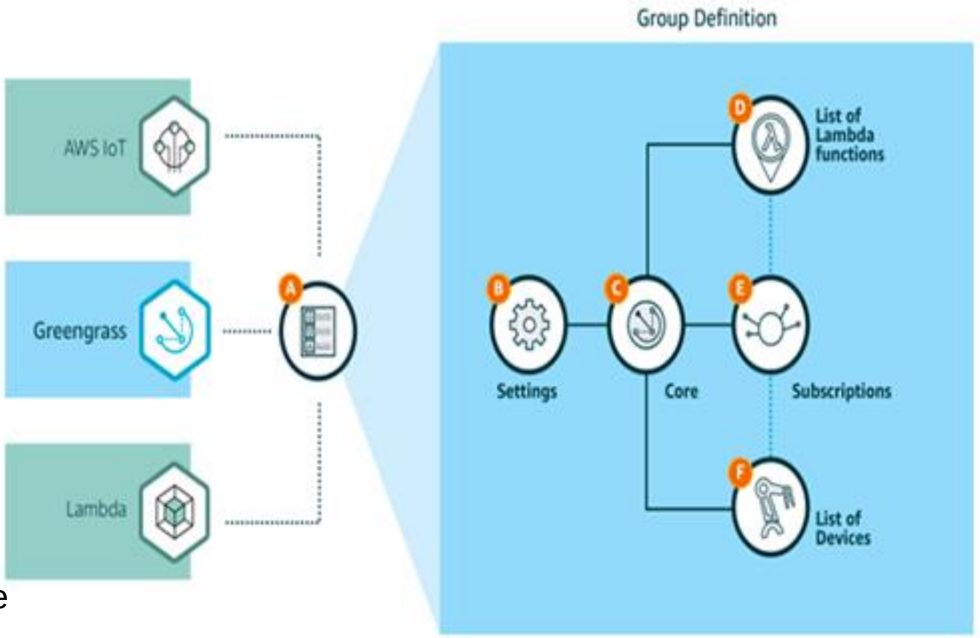
Thus the learn, decide and act cycle that happens with the cloud, edge and devices and performing the decision part run locally by ML inferencing.

AWS Greengrass Group: Cloud Capabilities to the Edge

AWS IoT Greengrass Group: An AWS IoT Greengrass group is a collection of settings and components, such as an AWS IoT Greengrass core, devices, and subscriptions. Groups are used to define a scope of interaction. For example, a group might represent one floor of a building, one truck, or an entire mining site. Since the group acts as the logical boundary for all the devices, it enforces consistent configuration and policies to all the entities.

AWS IoT Greengrass Core: This is just a device in AWS IoT Core registry that doubles up as an edge device. It is an x86 and ARM computing device running the Greengrass runtime. Local devices talk to the Core similar to the way they interact with AWS IoT Core.

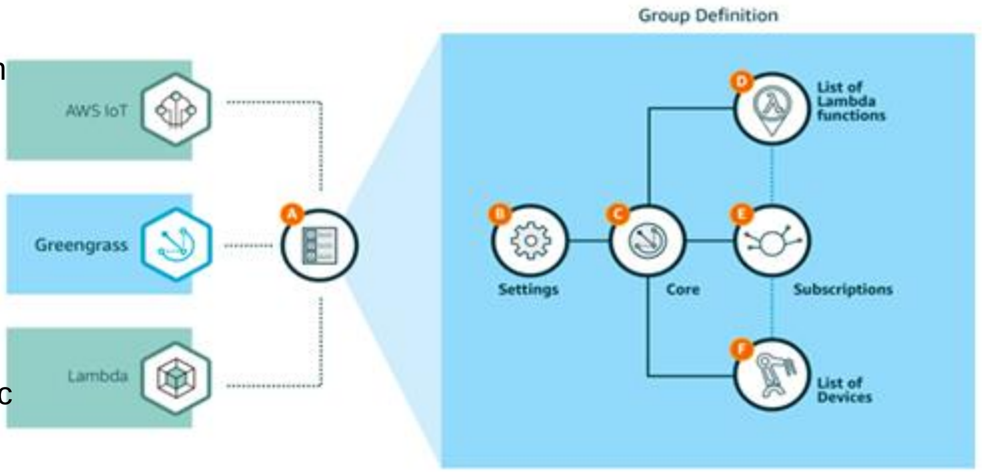
AWS IoT Devices: These are the devices that are a part of the Greengrass group. Once devices become a part of the group, they automatically discover the Core to continue the communication. Each device has a unique identity and runs AWS IoT Device SDK. Existing devices can be added to a Greengrass Group.



AWS Greengrass Group: Cloud Capabilities to the Edge

Lambda Functions: As discussed earlier, Lambda provides the local compute capabilities for AWS IoT Greengrass. Each function running within the Core uses Greengrass SDK to interact with the resources and devices. Lambda functions can be customized to run within the Greengrass sandbox container or directly as a process within the device OS.

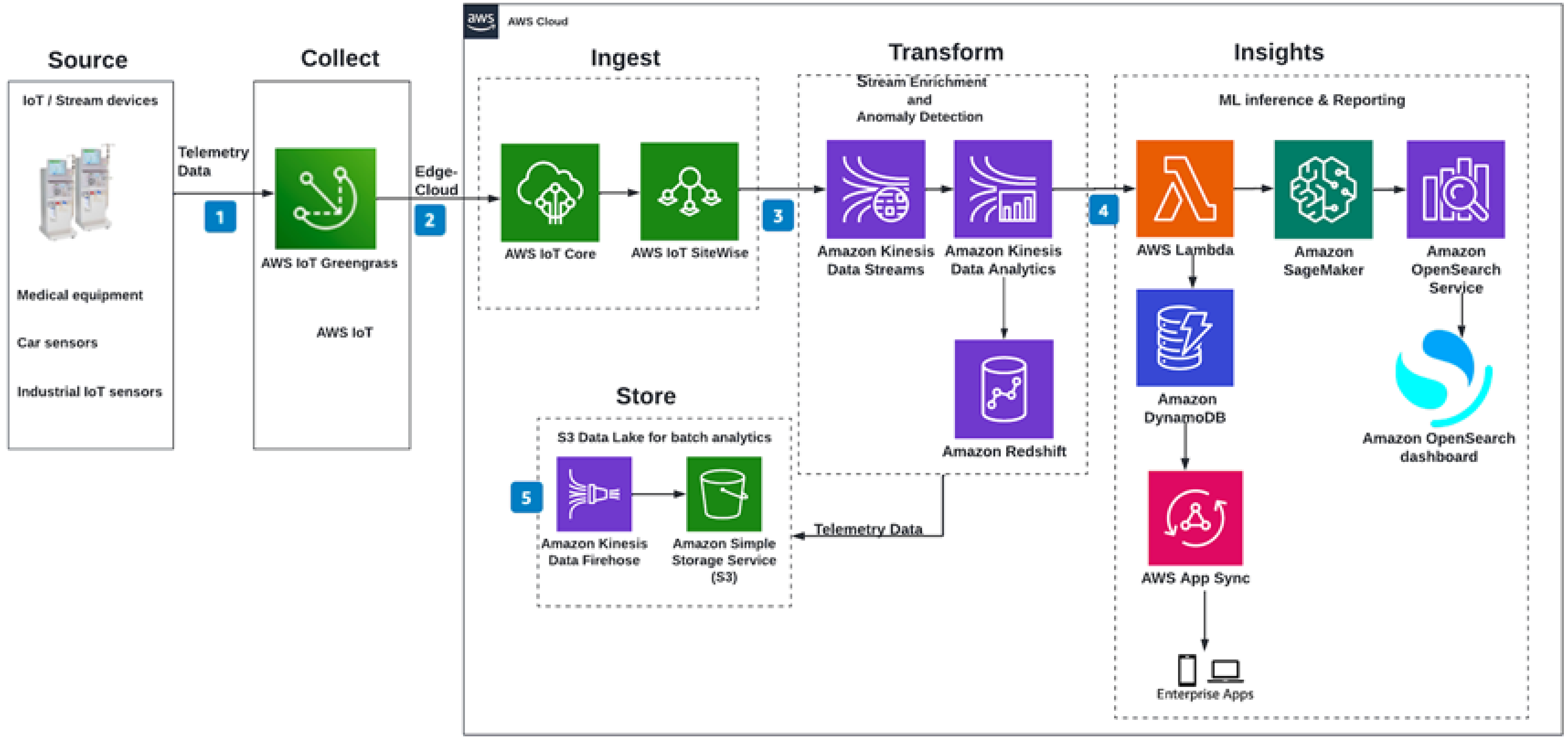
Subscriptions: AWS IoT Greengrass subscriptions connect the resources declaratively. It maintains a list of publishers and subscribers that exchange messages. For another scenario, a Lambda function may publish messages to a topic to which the device is subscribed. Subscriptions eliminate the strong dependency between publishers and consumers by effectively decoupling them.



Connectors: AWS IoT Greengrass Connectors allows developers to easily build complex workflows on AWS IoT Greengrass without having to worry about understanding device protocols, managing credentials, or interacting with external APIs. Based on a declarative mechanism, Connects extend the edge computing scenarios to 3rd party environments and services. Connectors rely on Secrets for maintaining the API keys, passwords, and credentials needed by external services.

ML Inferencing: This is one of the recent additions to AWS IoT Greengrass. The trained model is first uploaded to an Amazon S3 bucket that gets downloaded locally. A Lambda function responsible for inferencing inbound data stream publishes the predictions to a MQTT topic after loading the local model. Since Python is a first-class citizen in Lambda, many existing modules and libraries can be used to perform ML inferencing at the edge.

AWS IoT: Event-driven architecture with sensor data



AWS IoT: Event-driven architecture with sensor data

Phase 1:

- Data originates in IoT devices such as medical devices, car sensors, industrial IoT sensors.
- This telemetry data is collected using AWS IoT Greengrass, an open-source IoT edge runtime and cloud service that helps your devices collect and analyze data closer to where the data is generated.
- When an event arrives, AWS IoT Greengrass reacts autonomously to local events, filters and aggregates device data, then communicates securely with the cloud and other local devices in your network to send the data.

Phase 2:

- Event data is ingested into the cloud using edge-to-cloud interface services such as AWS IoT Core, a managed cloud platform that connects, manages, and scales devices easily and securely.
- AWS IoT Core interacts with cloud applications and other devices.
- You can also use AWS IoT SiteWise, a managed service that helps you collect, model, analyze, and visualize data from industrial equipment at scale.

AWS IoT: Event-driven architecture with sensor data

Phase 3:

- AWS IoT Core can directly stream ingested data into Amazon Kinesis Data Streams.
- The ingested data gets transformed and analyzed in near real time using Amazon Kinesis Data Analytics with Apache Flink and Apache Beam frameworks.
- Stream data can further be enriched using lookup data hosted in a data warehouse such as Amazon Redshift.

Phase 4:

- Amazon Kinesis Data Analytics can persist SQL results to Amazon Redshift after the customer's integration and stream aggregation (for example, one minute or five minutes).
- The results in Amazon Redshift can be used for further downstream business intelligence (BI) reporting services, such as Amazon QuickSight.
- Amazon Kinesis Data Analytics can also write to an AWS Lambda function, which can invoke Amazon SageMaker models.
- Amazon SageMaker is a the most complete, end-to-end service for machine learning.

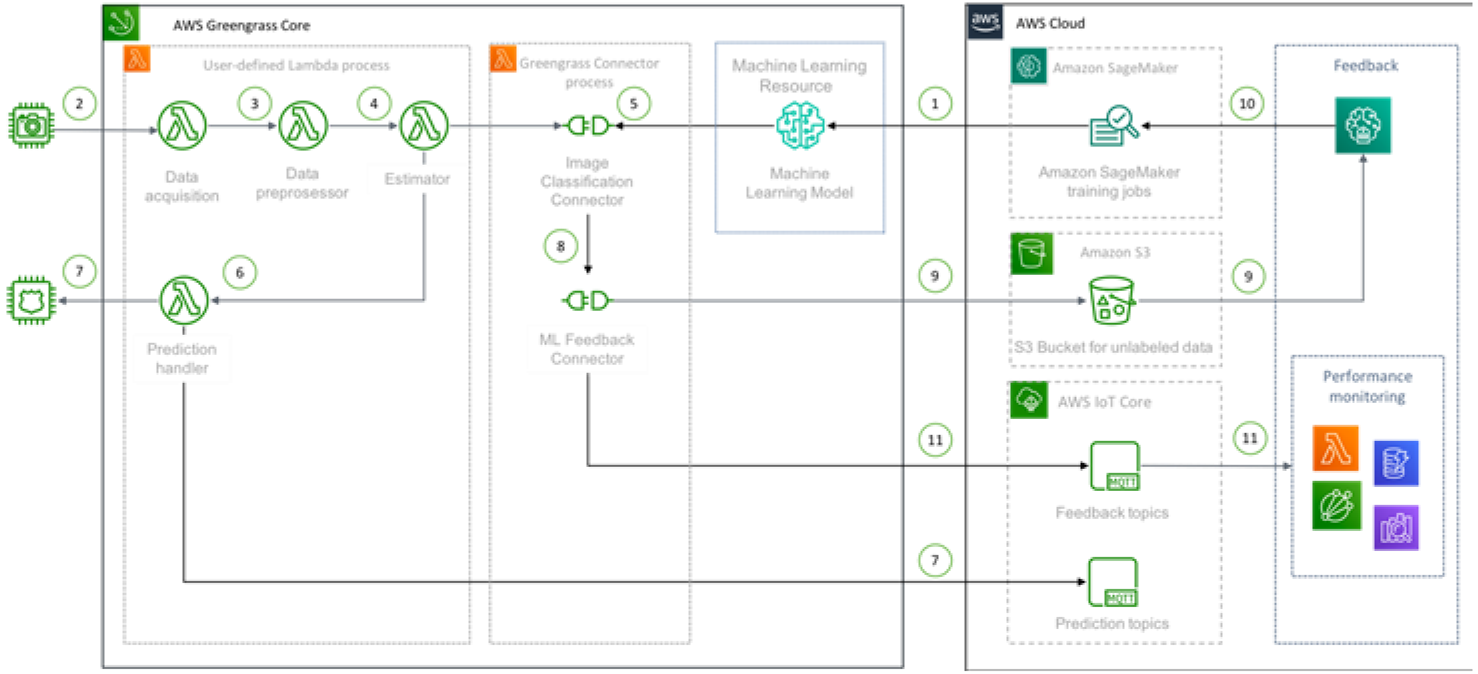
AWS IoT: Event-driven architecture with sensor data

Phase 5:

- Once the ML model is trained and deployed in SageMaker, inferences are invoked in a micro batch using AWS Lambda.
- Inferenced data is sent to Amazon OpenSearch Service to create personalized monitoring dashboards using Amazon OpenSearch Service dashboards.
- The transformed IoT sensor data can be stored in Amazon DynamoDB.
- Customers can use AWS AppSync to provide near real-time data queries to API services for downstream applications.
- These enterprise applications can be mobile apps or business applications to track and monitor the IoT sensor data in near real-time.
- Amazon Kinesis Data Analytics can write to an Amazon Kinesis Data Firehose stream, which is a fully managed service for delivering near real-time streaming data to destinations like Amazon Simple Storage Service (Amazon S3), Amazon Redshift, Amazon OpenSearch Service, Splunk, and any custom HTTP endpoints or endpoints owned by supported third-party service providers, including Datadog, Dynatrace, LogicMonitor, MongoDB, New Relic, and Sumo Logic.

Use Case: Greengrass Machine Learning Inference

This use case describe the steps in setting up Greengrass Machine Learning Inference, using Greengrass Image Classification ML Connector with model trained with Amazon SageMaker, and Greengrass ML Feedback connector to send data back to AWS for model retraining or prediction performance analysis.



Use Case: Greengrass Machine Learning Inference

The common design patterns of using Greengrass Connectors:

1. Creates a Amazon SageMaker training job to create the model. When the Greengrass configuration is being deployed, the Greengrass Core will download the model from the *Amazon SageMaker* training job as a local machine learning resource.
2. Data acquisition - This function periodically acquire the raw data inputs from a image source. In this example, we are using static images to simulate image sources.
3. Data preprocessor - This function pre-process the image by resize to the images used to train the model.
4. Estimator - This function predict the data input with the connector via IPC
5. Greengrass ML Image Classification Connector - The Connector loads the model from local Greengrass resource and invoke the model.
6. The process will handle the prediction result, with object detected and confidence level.
7. The result can be used to trigger an action, or send it back to the cloud for further processing.
8. Greengrass ML Feedback Connector - Greengrass ML Feedback Connector sends field data back to AWS according to the sampling strategy configured
9. Greengrass ML Feedback Connector sends unlabeled data to AWS
10. Unlabled data can be labeled using Amazon Ground Truth, and the labeled data can be used to retrain the model
11. Greengrass ML Feedback Connector sends prediction performance which can be used for realtime performance analysis.

Use Case Greengrass ML Inference: Deployment

The main steps for deployment are:

1. *Prerequisites.* Ensure there is an AWS IoT certificate and private key created and accessible locally for use.
2. *Train the ML model.* We will use an example notebook from Amazon SageMaker to train the model with the Image Classification Algorithm provided by Amazon SageMaker.
3. *Generate and launch the CloudFormation stack.* This will create the Lambda functions, the Greengrass resources, and an AWS IoT thing to be used as the Greengrass Core. The certificate will be associated with the newly created Thing. At the end, a Greengrass deployment will be created and ready to be pushed to the Greengrass core hardware.
4. *Create the config.json file*, using the outputs from the CloudFormation. Then place all files into the `/greengrass/certs` and `/greengrass/config` directories.
5. *Deploy to Greengrass*. From the AWS Console, perform a Greengrass deployment that will push all resources to the Greengrass Core and start the MLI operations.

Use Case Greengrass ML Inference: Deployment

Prerequisites:

- **AWS Cloud.**

  Ensure you have an AWS user account with permissions to manage `iot`, `greengrass`, `lambda`, `cloudwatch`, and other services during the deployment of the CloudFormation stack.

- **Local Environment**

  Ensure a recent version of the AWS CLI is installed and a user profile with permissions mentioned above is available for use.

- **Greengrass Core AWS IoT**

  Greengrass Core SDK Software which can be installed using pip command `sudo pip3.7 install greengrasssdk`

Use Case Greengrass ML Inference: Deployment

Train the model with Amazon SageMaker:

We will train the model using algorithm provided by Amazon SageMaker, Amazon SageMaker Image Classification Algorithm and Caltech-256 dataset.

- Login to Amazon SageMaker Notebook Instances console https://console.aws.amazon.com/sagemaker/home?#/notebook-instances
- Select `Create notebook instance`
- Enter a name in `Notebook instance name`, such as `greengrass-connector-training`
- Use the default `ml.t2.medium` instance type
- Leave all default options and select `Create notebook instance`
- Wait for the instance status to be `InService`, and select `Open Jupyter`
- Select `SageMaker Example` tab, expand `Sagemaker Neo Compilation Jobs`, `Image-classification-fulltraining-highlevel-neo.ipynb`, select `Use`
- Keep default option for the file name and select `Create copy`

Use Case Greengrass ML Inference: Deployment

Train the model with Amazon SageMaker:

- We are to use transfer learning approach with `use_pretrained_model=1`. Locate the cell that configure the `hyper-parameters` and add the additional `use_pretrained_model=1`. Details of the hyperparameters can be found in Amazon SageMaker Developer Guide - Image Classification Hyperparameters
- We will also be setting the prefix for our training job so that the Cloudformation Custom Resources is able to get the latest training job. Configure a `base_job_name` in the `sagemaker.estimator`. Locate the cell that initialize the `sagemaker.estimator` and add the `base_job_name`, for example, using `greengrass-connector` as the prefix. You will need this name prefix when creating the stack.
- Add a cell below the cell that do the training `ic.fit()` and the command `ic.latest_training_job.name` in the empty cell. This will give you the name of the training job that you can verify to make sure the Cloudformation stack picks up the correct job.
- Select the `Cell` from thet notebook menu and `Run All`

Use Case Greengrass ML Inference: Deployment

Launch the CloudFormation Stack:

Prior to launching the accelerator locally, a CloudFormation package needs to be created, and then the CloudFormation stack launched from the Template. Follow the steps below to create the package via the command line, and then launch the stack via the CLI or AWS Console.

The CloudFormation template does most of the heavy lifting. Prior to running, each *input* template needs to be processed to an *output* template that is actually used. The package process uploads the Lambda functions to the S3 bucket and creates the output template with unique references to the uploaded assets.

Use Case Greengrass ML Inference: Deployment

Configure the Greengrass Core:

With the stack deployed, we use one output from the CloudFormation stack, the *GreengrassConfig* value, along with the certificate and private key to complete the `config.json` so that Greengrass Core can connect and authenticate.

Starts the Greengrass Core:

With the Greengrass configuration `config.json` in place, start the Greengrass Core.

Use Case Greengrass ML Inference: Deployment

Deploy Cloud Configurations to the Greengrass Core:

From the AWS Console of AWS IoT Greengrass, navigate to the Greengrass Group you created with the Cloudformation, and perform *Actions->Deploy* to deploy to the Greengrass Core machine.

Use Case Greengrass ML Inference: Testing

To test out this accelerator without any hardware, you can install the Greengrass on an EC2 to simulate as a Greengrass Core

1. Create a EC2 running Greengrass, using the Cloudformation template in `cfn/greengrass_core_on_ec2-s3_models.cfn.yml`
2. Once the instance is created, copy the `greengrass-setup.zip` to the EC2
3. In the EC2, extract `greengrass-setup.zip` into `/greengrass` folder using command `sudo unzip -o greengrass-setup.zip -d /greengrass`
4. Restart the Greengrass daemon using the command `sudo systemctl restart greengrass`

Lecture Summary

- Introduction to AWS IoT platform
- Layered architecture of AWS IoT
- Concepts of AWS IoT Core
- Understanding of AWS greengrass
- Event-Driven architecture with sensor data in AWS IoT

THANK YOU!