AWS IoT Platform technical documentation

Problem definition

IoT Platform is a complex and sophisticated system that requires implementation of multiple processes, such as data ingestion, storing, processing and analytics from the edge devices. In addition, the modern smart platform also requires machine learning to implement anomaly detection, forecast, classification, computer vision tasks in edge and cloud. Moreover, companies must possess an extensive IT department with appropriate competition to utilize cloud service capabilities. It leads to a challenging task, especially for critical systems where low latency, robustness, and security are mandatory. Decomposing the complex solution, we could consider a few large architecture building blocks: edge platform, data platform and machine learning.

AWS Cloud significantly eliminates efforts to build the system and integrate facility into the platform. In the following sections, we consider a solution with AWS Cloud that simplifies building an data platform, edge platform and machine learning at the cloud and edge layer.

Data Platform Description

How Data Platform works

The entry point of Data Platform is the ingestion service AWS IoT Core for the edge layer. The next stages are storing and processing data for providing analytics and training ML models capabilities. As given in Figure 1-1.

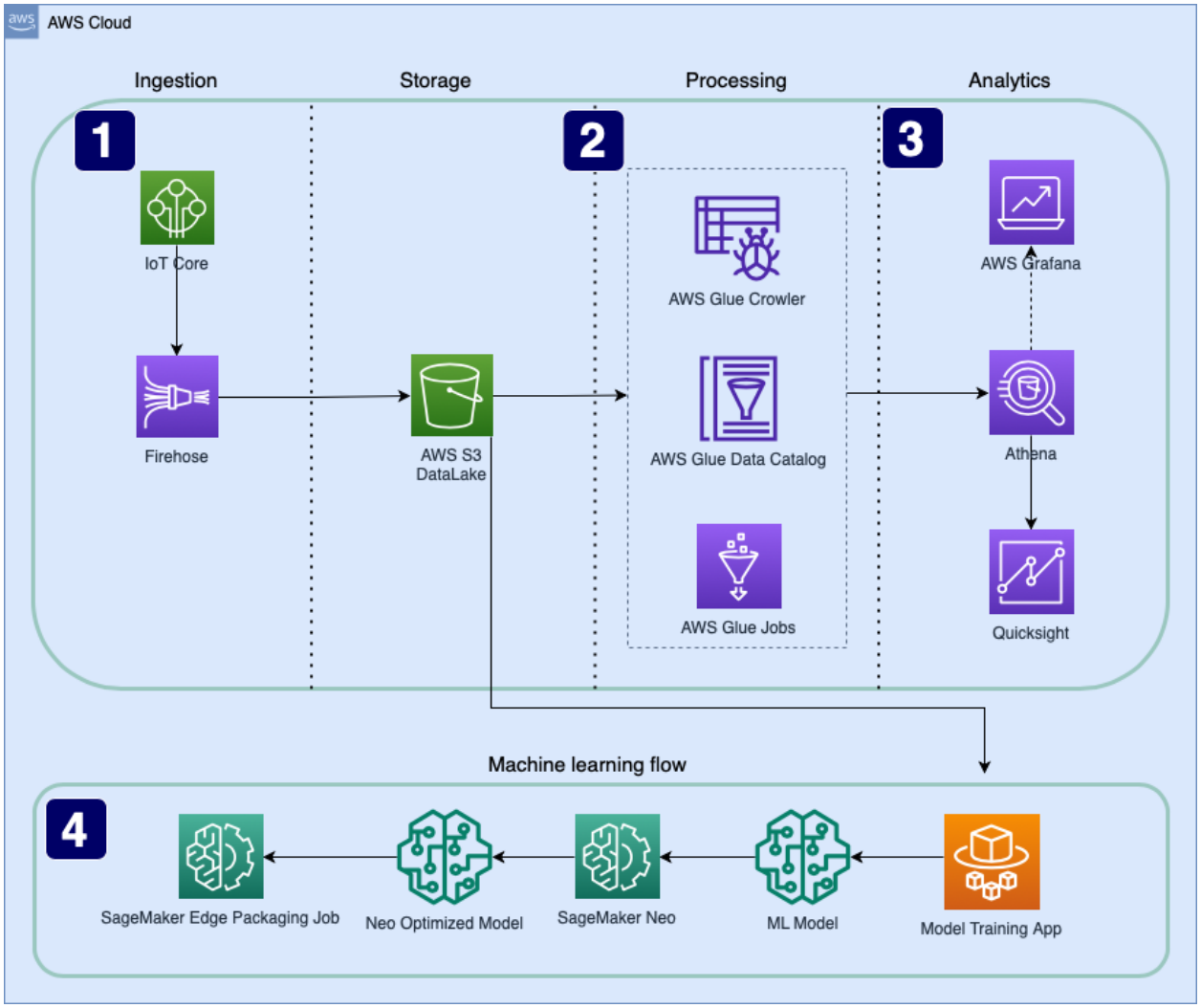

Figure 1-1

Inspect these parts in detail:

1) Having ingested device events from the edge site, we applied aggregation and transformation by AWS Firehose to accumulate and store batches at least for the last minute in an appropriate format, such as parquet or ORC. Nevertheless, according to AWS IoT best practices, it is better to process data at the edge layer before sending it to the cloud as much as possible.

2) Data processing and storage is a large building block for anomaly detection systems. In addition, this part prepares data for training machine learning models and visual quality control. We are considering building an ETL pipeline with AWS Glue and S3 as a storage system to implement this functionality.

3) In the case of visualization, we have at least two options: a business intelligence tool called AWS QuickSight and open-source visualization tool AWS Grafana with Athena connector.
4) Having prepared data for machine learning, we could train models using AWS Sagemaker components. Additionally, the pipeline is scalable for the hundreds of models that each model uses per device type.

   According to model training, we can consider such options as AWS Sagemaker API, AWS Sagemaker pipeline and any customer-specific method.
   In addition, AWS Sagemaker provides a component for model compilation. Hence, a compiled model is optimized for a specific platform or device type by NEO-compiler. Finally, IoT Core packages the compiled model to deploy in the edge layer.

AWS IoT Core Rules

AWS IoT Core uses a set of rules for routing. Each rule represents a SQL query from MQTT topics and a destination system. An example of a rule is shown in Figure 1-2.

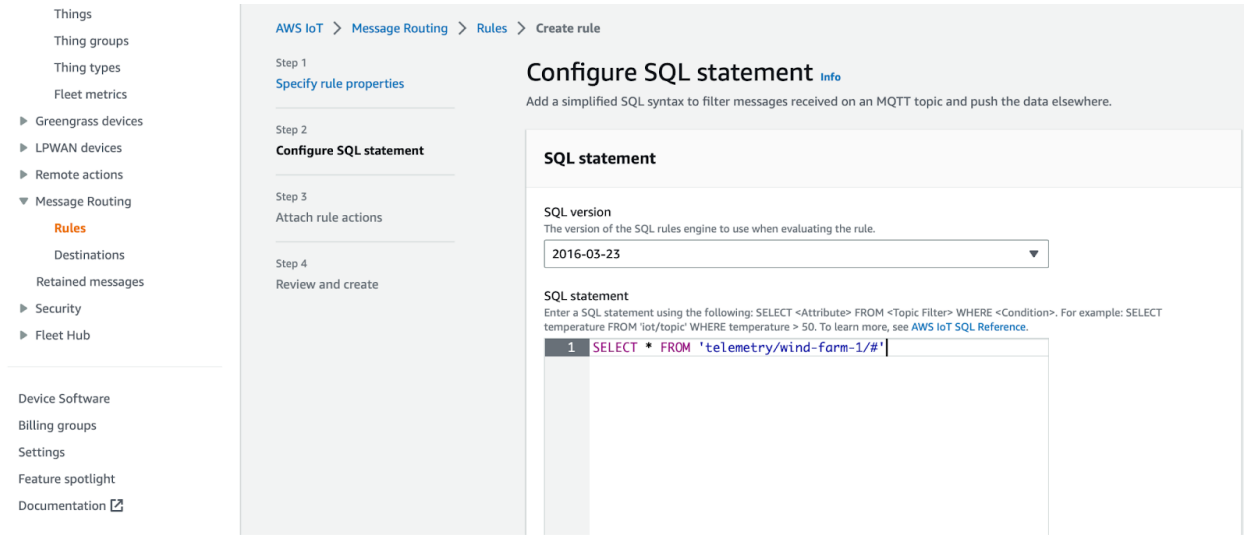

Figure 1-2

As a destination, AWS IoT Core provides a lot of options. In our case, AWS Firehose Stream for data aggregation and AWS Lambda for notifications are suitable choices. In addition, you can explore detailed settings in AWS Documentation.

Machine learning flow

In order to implement machine learning flow with edge capabilities, AWS Sagemaker provides three types of jobs: training jobs, compilation jobs and edge packaging jobs. AWS Sagemaker console is shown in Figure 1-3 below.

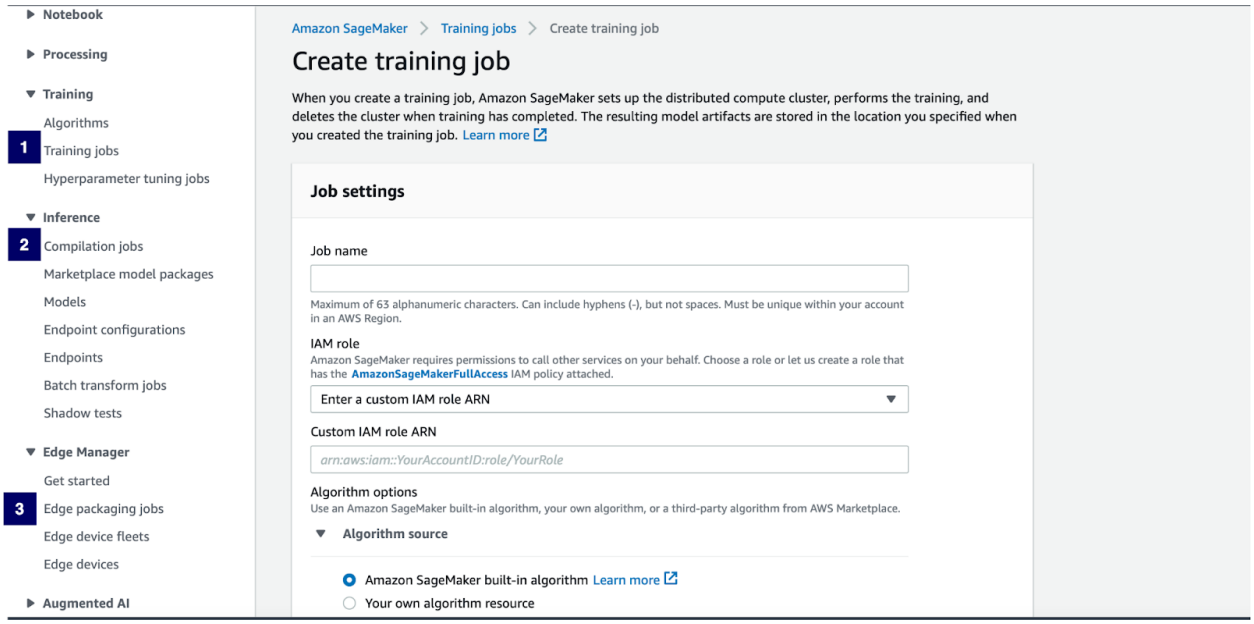

Figure 1-3

Inspect this functionality in detail:

1) In Training jobs, AWS Sagemaker uses the docker images to train models. There is a possibility to use AWS-provided images with the most popular frameworks, such as Tensorflow, Pytorch, MXNet, Scikit-learn etc. Furthermore, AWS provides Amazon SageMaker Python SDK to create various estimators and utilize Sagemaker API without directly using the docker images. Besides that, we can build a custom docker image hosted in ECR to train models.

   Additionally, before using training jobs, data scientists can bootstrap models in AWS Sagemaker Notebook.
2) The compilation process is responsible for getting an optimized model for various platforms or CPU-based and GPU-based device types. Therefore, AWS Sagemaker uses Neo Compiler to perform model compilation.

3) AWS Greengrass delivers the model as a standalone component to the edge layer. To address this task, AWS Sagemaker provides a packaging job. Having a packaged model, we could use it along with AWS Sagemaker Edge Agent, also known as an inference edge runtime.

In addition, we provide an automation solution on the basis of Step Functions. The flow includes the stages from training model to deployment. As given in Figure 1-2.

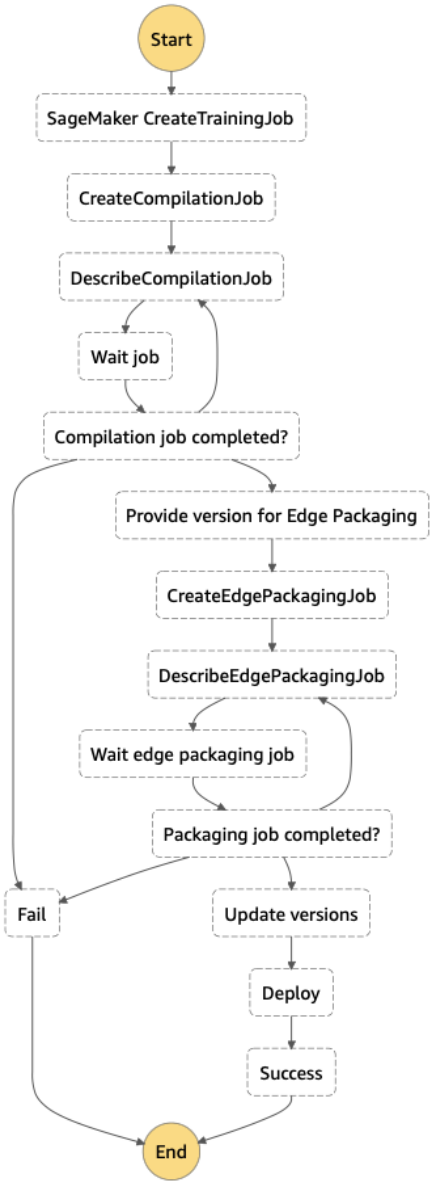

Figure 1-2

To modify the flow, you can go to AWS Step Function console and tune each job to your company's requirements. The steps configuration comes from AWS Sagemaker API request params. See details in API documentation. The job “Update versions” invokes AWS lambda function to update the model component version in AWS System Parameter Store. To modify the given logic, navigate to AWS Lambda Console and change the function called test-ml-lambda-for-state-machine.

Edge layer Description

AWS IoT Greengrass provides capabilities to build anomaly detection at the edge site without significant developing efforts using AWS-provided components. The runtime for these components at the edge site is called AWS Greengrass. Moreover, we can build custom Greengrass components written in Java, C++ or Python to implement customer-specific logic, such as inference and anomaly response.
Overall, a combination of AWS-provided Greengrass components and the custom components comprehend the implementation of an anomaly detection system. Additionally, AWS calls it the core device.

Speaking of security, the AWS IoT ecosystem supports zero trust by default. Consequently, IoT Greengrass components establish trust by authentication using X.509 certificates, security tokens and custom authorizers. So that all communications among client devices, Greengrass core devices and IoT Core are secured by TLS 1.2. As given in Figure 1-3.

In the next sections, we consider data ingestion, inference and anomaly response logic, and also deployment to the edge site.

Data ingestion

Data ingestion is a complicated task for any manufacturing company. To ingest data to the core devices, we use MQTT protocol over TLS 1.2 implemented in AWS Greengrass MQTT broker component. Besides that, if the client devices don’t support MQTT protocol, we are supposed to write MQTT adapter components. Additionally, integration with AWS services, such as AWS Cloudwatch, Kinesis Video Streams are secured by AWS-provided components.

Furthermore, we can extend ingestion functionality to the core devices by using community components. It enables data collection over protocols such as Modbus, **LoRaWaN**, WebRTC and so on.

As given in Figure 1-3.

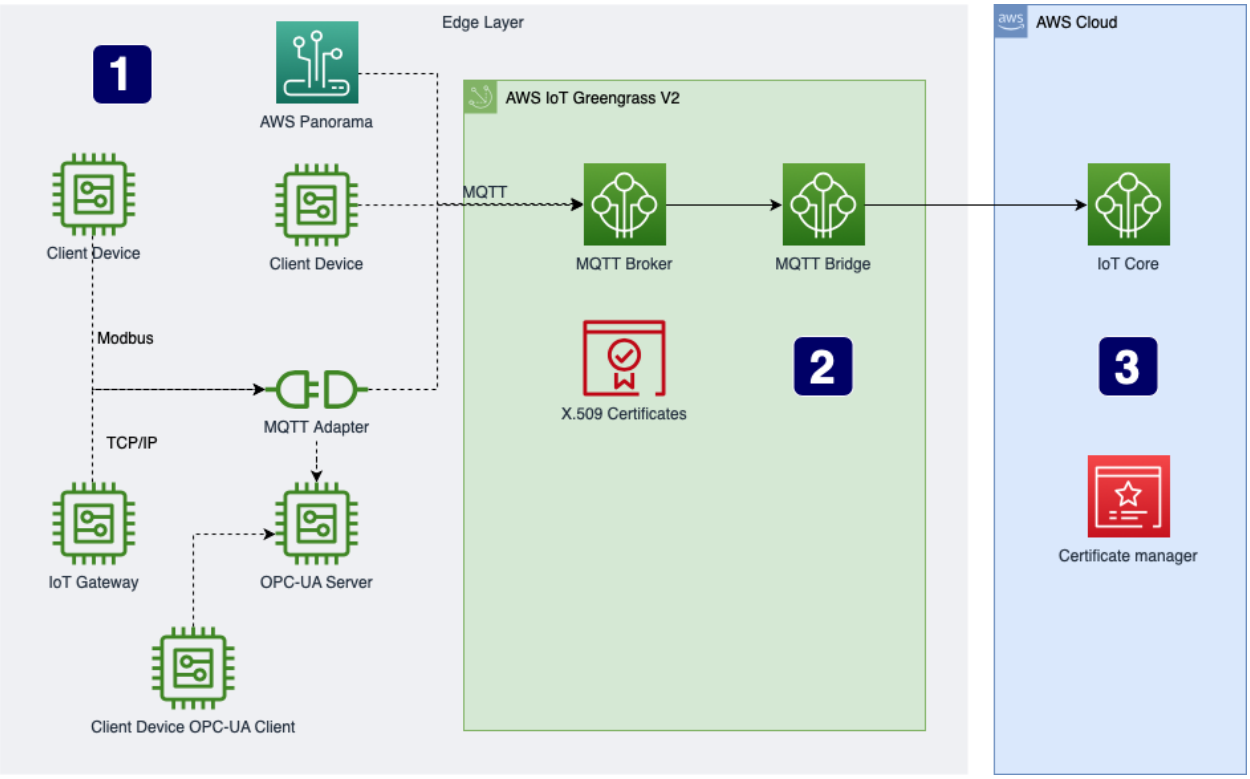

Figure 1-3

Further on, each stage of data ingestion is covered.

1. As we mentioned, the connection between the client devices and MQTT Broker Greengrass component requires using MQTT as a transport. It leads to a certain volume of work in writing adapters. Nevertheless, this approach allows us to encapsulate all customer-specific protocols outside and focus Greengrass components on an anomaly detection task.

   Additionally, we can manage video feeds from various sources, such as AWS panorama devices with embedded computer vision algorithms. Accordingly, we can perform anomaly detection for smart cities, evaluate manufacturing quality, improve supply chain logistics, and track retail visitors.

2. AWS IoT Ecosystem provides the components to host MQTT Brokers, such as the Moquette MQTT 3.1.1 and the EMQX MQTT 5.0 broker components. According to zero-trust principles, the connections with those must only be established by TLS 1.2.
3. Finally, the selected MQTT topics route to IoT Core using the MQTT Bridge component.

Inference at the edge

Inference logic is implemented in a Greengrass custom component with C++, Java or Python.
The component utilizes AWS Sagemaker Edge Agent GRCP API to manage the models and perform predictions. Accordingly, Edge Agent can load models from a deployed model auto-generated component, described in the section Machine learning flow, into memory. The overall approach is shown in Figure 1-4.

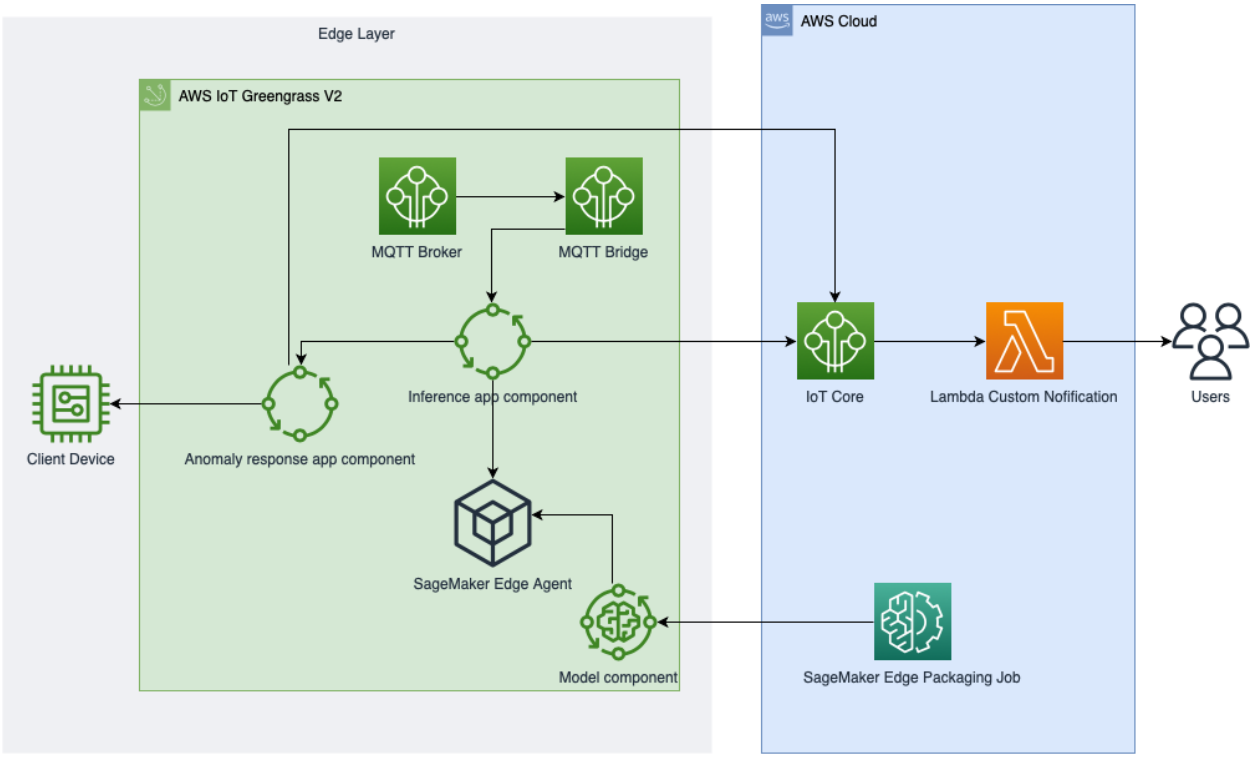

Figure 1-4

Inference app component reads data from AWS IoT Greengrass Core IPC service (local pubsub), manages the sample, and invokes Sagemaker Edge Agent API to implement the inference.

The inference app components has the following responsibilities:

- Listen to local pubsub
- pre-processing data to build an input sample for the inference
- inference interpretation
- edge processing before sending to AWS IoT Core

You can find a detailed description of the component development process in AWS documentation.

According to the desired performance, we can consider the various optimization techniques: implementing the component in low-overhead languages, such as C++; reducing the network communications by execution inference without Sagemaker Edge Agent; using a much faster machine learning model; applying batching to an input sample; upgrading hardware.
Regarding the inference platform, AWS-provided inference engine supports CPU-based inference and GPU-based inference, such as NVIDIA Jetson.

You can inspect a detailed example of an inference app in the Case Study section.

Anomaly response at the edge

Detecting anomalies logically to implement an anomaly response mechanism. For this purpose, we created another AWS Greengrass component for the edge site called the anomaly response app. As given in Figure 1-4. So that the primary responsibilities are listed below:

- Listen to local pubsub
- Choose the appropriate anomaly response for a particular anomaly
- Communicate with customer device management systems
- Notify customers about the executed commands through IoT Core

The notifications from AWS IoT Core can direct to Lambda, AWS SNS, AWS Cloudwatch or even HTTP Endpoint.

You can inspect a detailed example of an anomaly response app in the Case Study section.

Deployment

Deployment of AWS Greengrass components performed by AWS IoT platform. To create deployment we can use either AWS IoT Core console, or AWs IoT Core API. The approach with the console is shown in Figure 1-5.

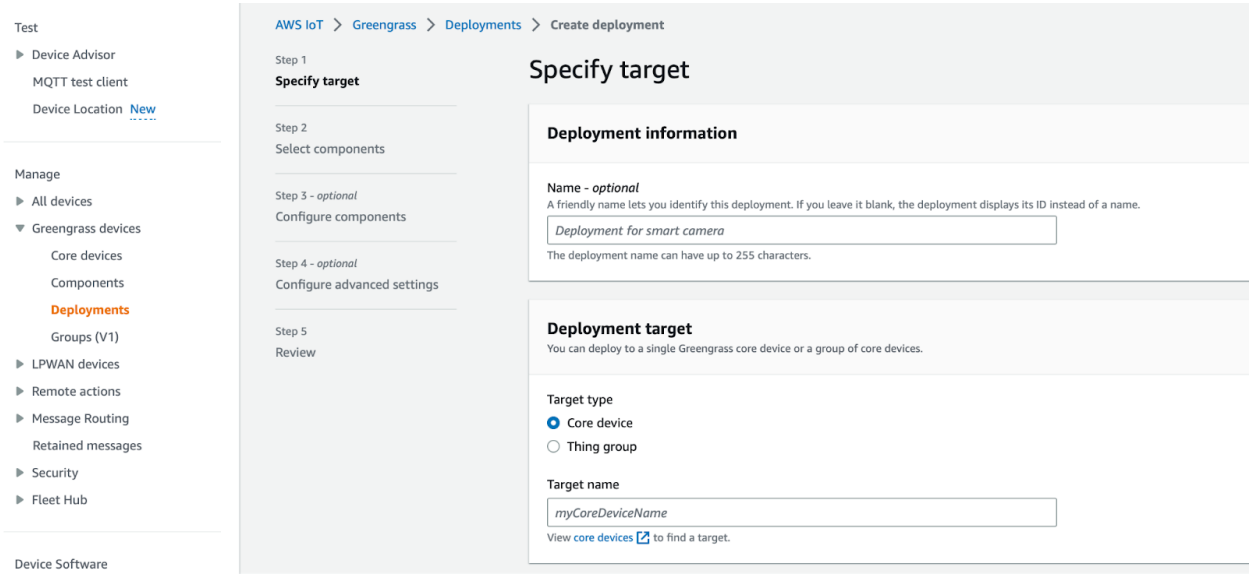

Figure 1-5

So that you can select the list of components with appropriate versions and configurations. We recommend using the following list:

1) aws.greengrass.SageMakerEdgeManager - The component delivers the AWS Sagemaker Edge Agent to the edge layer.
2) aws.greengrass.clientdevices.mqtt.Bridge - The component enables transfers messages from local MQTT broker to IoT Core.
3) aws.greengrass.clientdevices.mqtt.Moquette - the component enables local MQTT Broker.
4) aws.greengrass.LogManager - optionally uploads logs from Greengrass core devices to Amazon CloudWatch Logs.
5) aws.greengrass.Cli - develop and debug components locally.
6) aws.greengrass.clientdevices.IPDetector - Client devices use this information to discover core devices to which they can connect.

In addition, the list can be extended with inference app component, model component and anomaly response app component.

It's important to note that things or core devices can have only one active deployment at a time.

Demo ML applications

The solution consists of a simulation of the three client devices and the bootstrap script to create AWS Greengrass components, such as the inference and anomaly response apps.

Create Edge infrastructure for Demo

ADP IoT platform provides the docker container for demonstration purposes that includes codebase for GreengrassV2 components and client devices. The main goal of this container is to simplify the deployment process and provide reference of use cases.

Initial setup

At first, docker containers should be started with following commands in different terminal tabs as we are planned to use containers in interactive mode:

Code: shell

```
# Command to launch container for GreengrassV2 Core device
> docker run --name core-device \
 -e AWS_DEFAULT_REGION=YOUR-REGION \
 -e AWS_SECRET_ACCESS_KEY=YOUR-SECRET-ACCESS-KEY \
            -e       AWS_ACCESS_KEY_ID=YOUR-ACCESS-KEY-ID       -ti       \
CONTAINER-NAME:CONTAINER-VERSION

# Command to launch container for Client Thing device
> docker run --name client-thing \
 -e AWS_DEFAULT_REGION=YOUR-REGION \
 -e AWS_SECRET_ACCESS_KEY=YOUR-SECRET-ACCESS-KEY \
            -e       AWS_ACCESS_KEY_ID=YOUR-ACCESS-KEY-ID       -ti       \
CONTAINER-NAME:CONTAINER-VERSION
```

Core device setup

To prepare your core device to be deployed, open terminal tab where *core-device* was launched and enter following command into container's terminal:

```
Code: shell
> run-core-device
```

The script is a simple wrapper around GreengrassV2 Installer which will install AWS Nucleus component inside the container and start the Greengrass component. All the resources related to the demo will have prefixes *'adp-iot-demo'*. As a result, you will be able to see thing device, core device and all related policies.

Prepare components

To build and publish demon components into Greengrass registry and S3 bucket, open *client-thing* container's terminal and launch command:

```
Code: shell
> create-components
```

This script will initialize components, prepare artifacts. As well, s3 bucket for components will be created with the prefix *'adp-iot-demo'*. As a result, two component will be built and published into S3 bucket and Greengrass registry

Prepare client thing device

Preparation of client thing devices could be done by following commands in the *client-thing* terminal:

```
Code: shell
> create-client-thing  # The script will bootstrap resources for IoT thing
>  link-core-device-and-thing  # Create association between core-device and
```

Code: shell

```
client-thing
> create-core-deployment # Bootstrap some default recommendations for Core devices
```

Launch client thing device

Container provides two options to launch a client thing device code. First option is to send data into IoT Core directly via IoT endpoint. Second option is to send data via Greengrass core device with cloud discovery. In the demo we will be sending data via Greengrass. Run following command in *client-thing* container:

Code: shell

```
> run-client-device-greengrass # Send data via Greengrass
```

Machine learning flow

After setting up the main edge components, we need to prepare the AWS Model Greengrass component with the ML model. In order to automate the preparation pipeline, we described a state machine in AWS Step Function in the previous section. We recommend using that flow to perform model training, compilation, edge packaging and deploy a model component at the edge site. To execute the flow, navigate to AWS Step Function console and find a state machine called IoTMachineLearningFlow.

The primary machine learning job is called “Sagemaker training job” with the defined input, output and hyperparameters. You can discover a detailed description of API params in AWS documentation. The given job uses AWS-Provided image for PyTorch ML framework and the dataset from the previous section.

Inspect the major artifacts for training purposes:

1) sagemaker_submit_directory - contains the S3 location of an training code (gzipped) compatible with SageMaker TrainingJob;

2) sagemaker_program - the python module from the archive that is started by SageMaker TrainingJob, must process the command line arguments and store training results in the specified folder.

It's important to note that the name of execution turns into the name of all created jobs in AWS Sagemaker in the scope of execution.

As a result of execution, we have created and deployed the AWS Greengrass component with the model called SagemakerModelComponent to the core device.

Afterward, we can check the overall working of the edge layer using the MQTT test client of AWS IoT Core. Navigate to the MQTT test client tab and subscribe by the topic filter "anomaly/#". As given in Figure 1-6.

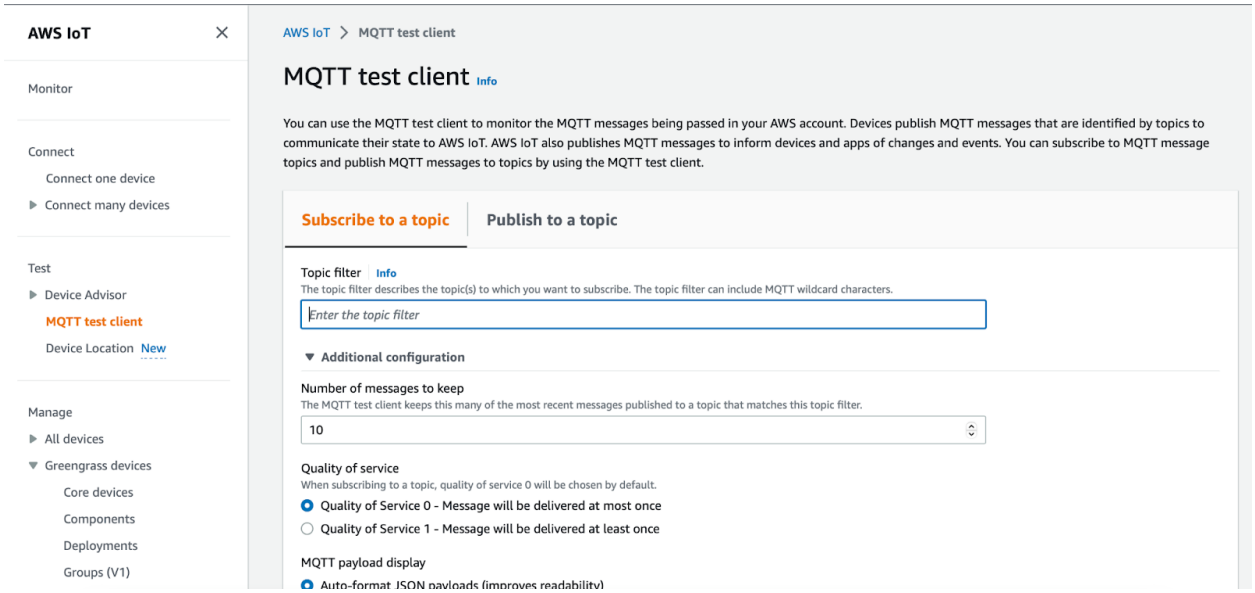

Figure 1-6

Having received data from the client devices, we could direct the messages to S3 by AWS Firehose Stream. Navigating to the tab Message Routing/Rules of AWS IoT Core, you can see pre-created rules. Ensure that the rule configuration matches the MQTT topic. As a destination, there is Firehose Stream under the tab Actions. To discover the configuration of the delivery stream, navigate to AWS Kinesis console and tab called Delivery streams. The output configuration can be found in the tab Configuration/Destination settings/Amazon S3 destination.
By following the S3 link of the configuration, you can explore the buckets created by Firehose.
Hence, the ingested data can be used as a dataset to train ML models.