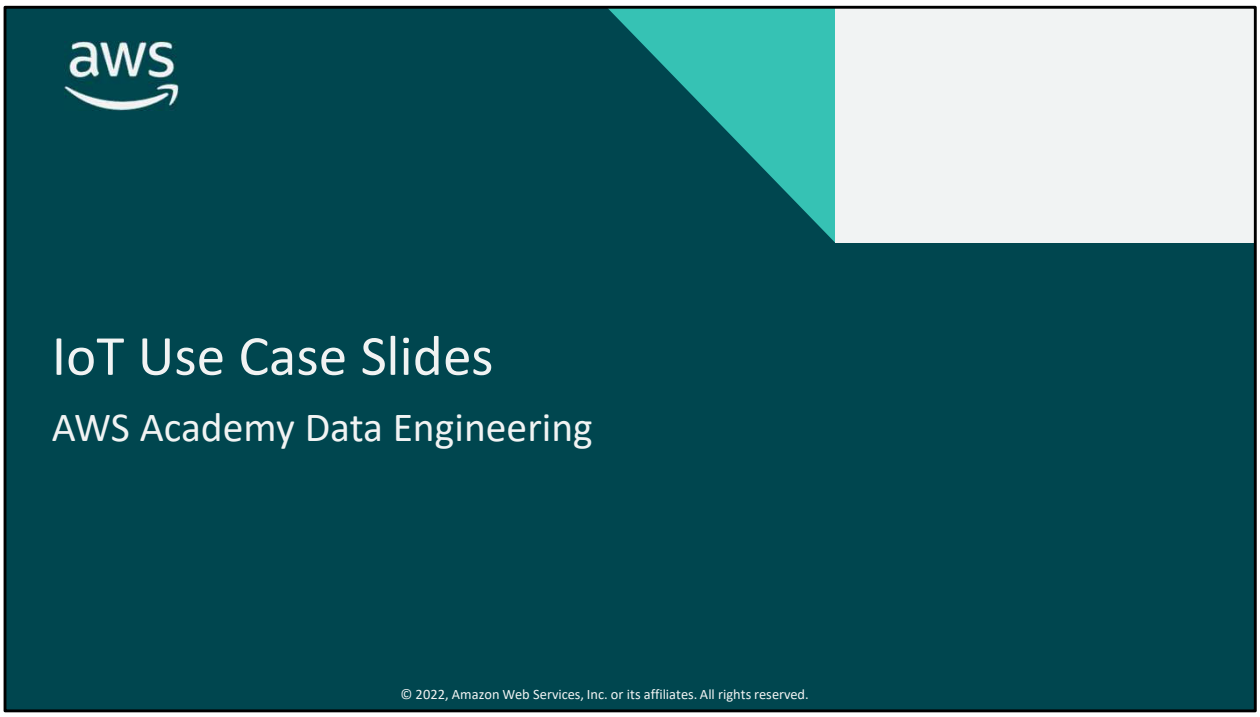


|Slide number 1

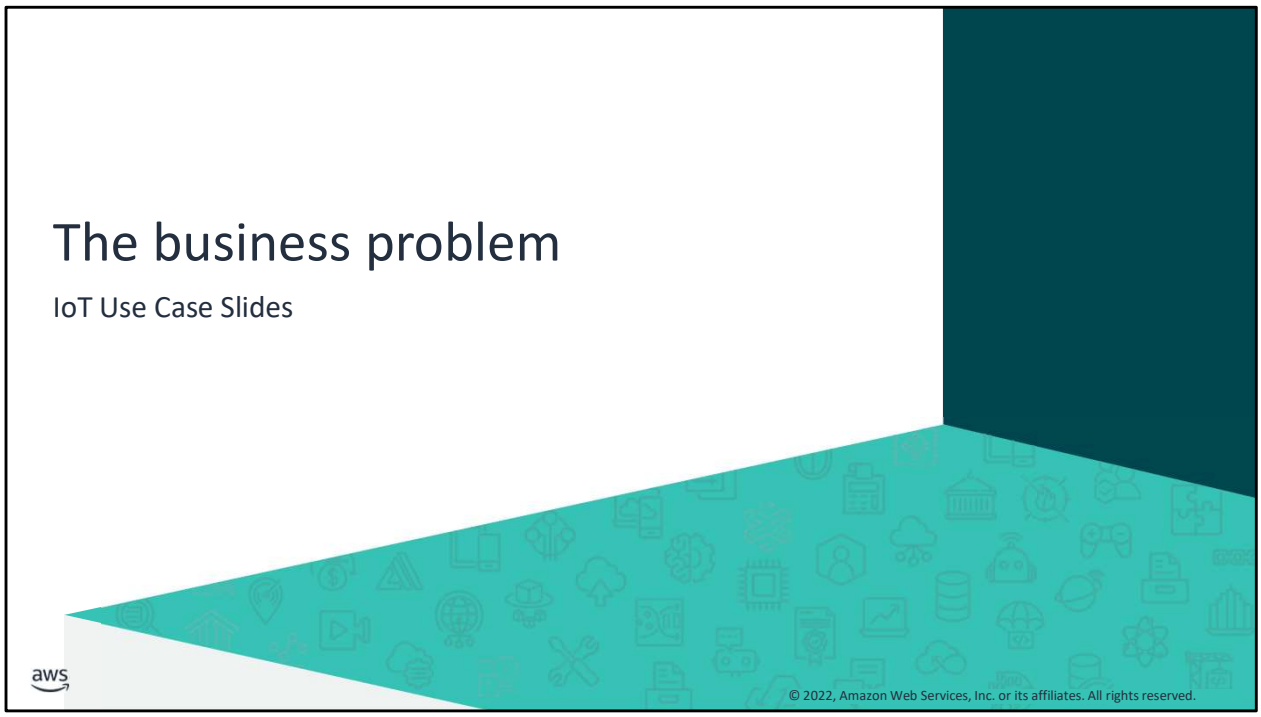
|Instructor notes

|This architecture is based on a demo that was recorded as part of the AWS re:Invent 2021 presentation titled **Create an Analytics Pipeline from your IoT data on AWS**. Although not needed to use the deck, there is a file with example e-bike data available in the Educator Files in the course.

|

|Student notes

This presentation walks through a use case for Internet of Things (IoT) analytics. The presentation suggests an architecture to support the ingestion and processing layer, storage layer, and analysis and visualization layer.



|Slide number 2

|Instructor notes

|The business problem mirrors one of the use cases that was presented in The Elements of Data module. The services used reflect content that begins with module 7. You might choose to review each part of this use case as you deliver the related modules on ingestion, storage, and analysis and visualization, or you might walk through the use case as a separate review after covering all modules.

|

|Student notes

The scenario presented here follows one option that was presented in the pipeline planning activity in The Elements of Data module. As you have learned in this course, it is important to work backwards from the business problem to determine what type of pipeline you need to build.

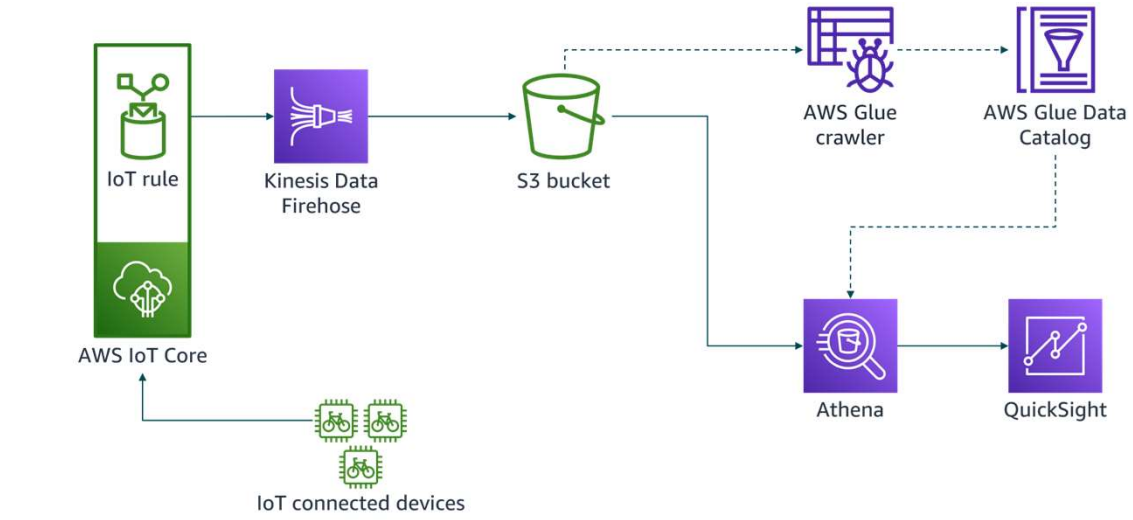
Working backwards from the business problem

- Your startup e-bike company has partnered with a local government to pilot an e-bike rental program in several suburban neighborhoods.
- The government's goal is to reduce carbon emissions and traffic congestion.
- At this stage, they need to collect and analyze trends in data about how the bikes are being used. They also need to determine what data they want to track as they expand the program.
- They know that they want to look at where the bikes have traveled, battery life, and engine use levels (off, low, medium, high).
- Each of your 50 bikes is equipped with an Internet of Things (IoT) device, which is powered by the bike battery and uses the wireless phone network to transmit data. Devices use the MQTT message protocol to send and receive JSON data. The devices are currently configured to send a message every 2 minutes.
- You need to build a proof-of-concept pipeline to collect and analyze data from the IoT devices that are built in to the e-bikes.

aws

|Slide number 3
|Instructor notes
|Key points to call out are that the company is a startup, and the goal is exploratory analysis of trends in the data. The data is being sent from 50 IoT devices every 2 minutes. This implies that you need a streaming ingestion solution but not real-time analysis. The JSON format is semistructured.
|
|You want to select services that provide a quick startup with limited investment in resources but also something that will be scalable. Your consumers are business users, so the analysis tools should allow them easy access to visualizations.
|
|The processing requirements are limited to cleaning the data. It is coming from 50 different devices, but all share the same semistructured file format. IoT data is noisy and might have corrupted values. Your choices should also make it possible for you to expand into other analytics or machine learning (ML) use cases without starting over.
|
|Student notes
What are the key characteristics that are described in the business problem that will drive the decisions that you make to build the pipeline?

The IoT reference architecture



|Slide number 4
|Instructor notes
|The final architecture is presented here and is repeated at the end of the deck.
|
|Student notes
The diagram on this slide illustrates the proposed AWS architecture for the proof-of-concept application. The IoT devices on the bikes will send messages to AWS IoT Core. An IoT rule will transform the messages and send them to an Amazon Kinesis Data Firehose delivery stream. The delivery stream is configured as a subscriber to the topic and might perform additional transformations to prepare the data for analysis.

The delivery stream will aggregate records for delivery to an Amazon Simple Storage Service (Amazon S3) bucket (which could be part of a data lake). An AWS Glue crawler runs to discover information about the data that is added to the bucket, including its schema. The crawler stores that information in the AWS Glue Data Catalog. Information from the catalog is used to query the data in place by using Amazon Athena. Business users can access and visualize query results by using Amazon QuickSight.

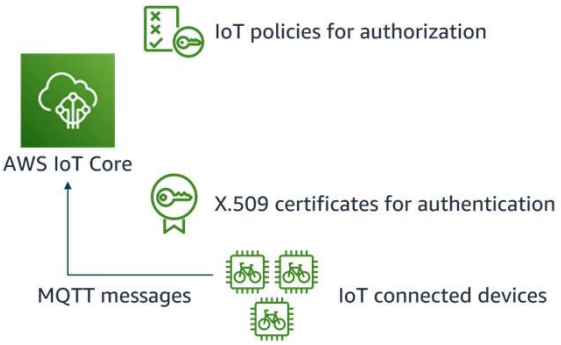
All the services in this architecture are serverless. This means that you do not need to provision any servers to run any parts of the application, and capacity will scale

automatically. There is also no upfront cost to get started—you will pay for the services as they are used to ingest, process, store, and analyze data.

|Slide number 5
|Instructor notes
|In this example, processing is limited. The services that were identified have options to filter and transform data to serve the described purpose without additional processing.
|
|Student notes
First, let’s look at the ingestion and processing layer.

Connect devices to AWS IoT Core

1 Connect IoT devices to AWS IoT Core, and verify that messages are being published to your topic



|Slide number 6
|Instructor notes
|IoT security is not discussed in depth. The key point at this level is that you must secure access between devices, as well as between devices and cloud services. The “Security” section of the *AWS IoT Core Developer Guide* provides more in-depth coverage of this topic.
|
|Student notes
To get started, you would need to connect your devices to AWS IoT Core and start publishing messages to a designated topic. For this example, the topic will be **ebike/stats**. You might connect actual devices for testing, or you could use test data with a device simulator to test the flow.

In AWS IoT Core, you can use the options in the **Connect** menu to connect one or many devices. This involves creating *thing* objects in AWS IoT Core to represent your devices. You also configure the security certificate that controls access to the device and the IoT policies that are required to control access to AWS IoT resources. Within AWS IoT Core, you can subscribe to the ebike/stats topic by using the MQTT test client to verify that the messages from your device are being published to the topic.

Take note of the security components that are involved in using IoT devices. Typically, devices use X.509 certificates to secure access between IoT devices. AWS IoT Core uses

these certificates to secure the communications between the devices and AWS IoT Core, and to authenticate IoT devices. AWS IoT Core policies provide permissions to authorize actions within AWS IoT Core, such as connecting to the message broker. AWS IoT Core policies are JSON documents and follow the same conventions as AWS Identity and Access Management (IAM) policies.

Both AWS IoT Core policies and IAM policies are used with AWS IoT Core to control the operations that an identity (also called a *principal*) can perform. IoT policies are used to authorize devices with X.509 certificates, and IAM policies are attached to an IAM user, group, or role.

Example: MQTT message

```
{
  "format": "json",
  "topic": "ebike/stats",
  "timestamp": 1665102292810,
  "payload": {
      "device-id": "rLdMw4VRZ",
      "location": "{ 'latitude': 38.9696, 'longitude': -77.3861 }",
      "battery-pcnt": 75,
      "engine-use": "off",
      "rented": 1,
      "last-pickup": "Oak",
      "last-drop": "Oak"
      }
}
```

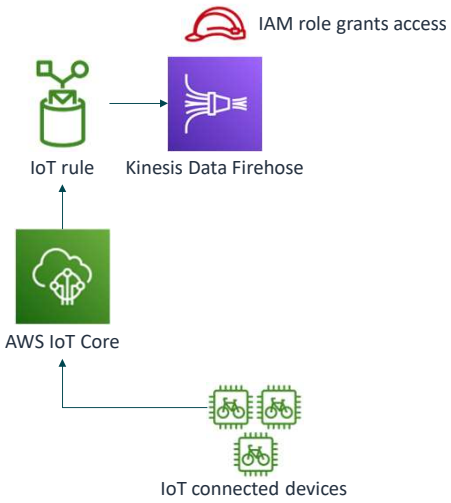
Topic and timestamp

Device data collected from sensors

|Slide number 7
|Instructor notes
|The *Designing MQTT Topics for AWS IoT Core* whitepaper provides information about MQTT communication patterns. The whitepaper also provides a set of best practices for MQTT topic design. The link to the whitepaper is included in the Content Resources for the course.
|The example is a fictitious record, but uses a latitude and longitude in Herndon, VA. The rented field is a Boolean field indicating whether the bike is currently being rented (1) . The pickup and drop fields are bike stations where renters can get or leave bikes. The example was built using the AWS IoT Device Simulator available through the AWS Solutions Library.
|
|Student notes
This slide shows an example of how an MQTT message that is sent to the ebike/stats topic might provide attributes for a given IoT device.

Configure IoT rules to route and filter messages

2 Configure an IoT rule to route new messages to the Amazon Kinesis Data Firehose delivery stream



|Slide number 8

|Instructor notes

|The “Rules” section of the *AWS IoT Core Developer Guide* provides detailed information about how rules work and what can be done with them.

|

|Student notes

When device data is being published and devices are successfully connected to AWS IoT Core, you can create a rule to select messages and route them to subscribers. In this example, the rule will route messages from the ebike/stats topic to a Kinesis Data Firehose delivery stream. An IAM role will be used to grant permissions to the IoT rule to write records to the delivery stream.

Example: IoT rule

```
{
     "ruleArn": "arn:aws:iot:us-east-2:623616953939:rule/ebike",
     "rule": {
          "ruleName": "ebike",
          "sql": "SELECT device-id as id,location as bike_loc,battery-pcnt as
              battery,engine-use as engine FROM ebike/stats WHERE rented=1",
          "description": "",
          "createdAt": "2022-10-06T21:51:00+00:00",
          "actions": [
          {
               "firehose": {
                    "roleArn": "arn:aws:iam::account-id:role/service-role/ebikes-role",
                    "deliveryStreamName": "PUT-S3-MXjPG",
                    "separator": ",",
                    "batchMode": false
               }
          }
          ],
     "ruleDisabled": false,
     "awsIotSqlVersion": "2016-03-23"
...
```

Filter attributes and records

Deliver to Kinesis Data Firehose

IAM role that grants permissions to put records on the stream

|Slide number 9
|Instructor notes
|
|Student notes

In this example, you see the components that make up an IoT rule—the SELECT statement and the action to be taken.

The SELECT statement designates what data attributes should be included. The FROM clause designates the relevant MQTT topic, and you can use the WHERE clause to filter which records should be included. The **actions** section tells the rule where to route the results of the query.

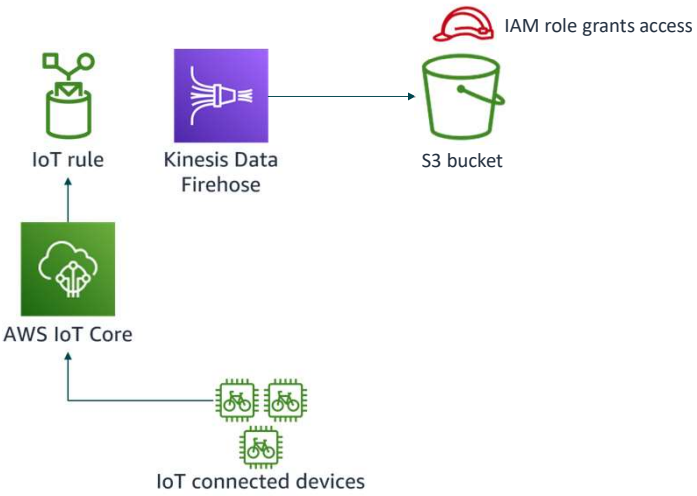
In this example, the SELECT statement limits which attributes to include (**device-id**, **location**, **battery-pcnt**, and **engine-use**) and maps them to different field names. A rule could also augment data by adding additional attributes. For example, if you wanted to include a field to identify these records as part of the pilot program, you might update the SELECT clause to include ", pilot as stage" after "engine-use as engine".

The FROM clause indicates the topic to query—**ebike/stats** in this example. The WHERE clause limits the scope of records to only those where the value of **rented** is 1.

The actions in this example tell the rule to deliver the query results to a Kinesis Data Firehose delivery stream that is called **PUT-S3-MXjPG**. You can name the stream as you like, but in this example, the user chose to create the delivery stream as part of creating the rule in the AWS Management Console. This made it easy to create the stream and the IAM permissions that are required to access the stream.

Configure the delivery stream

3 Configure the Kinesis Data Firehose delivery stream to aggregate and transform records before delivering them to Amazon S3



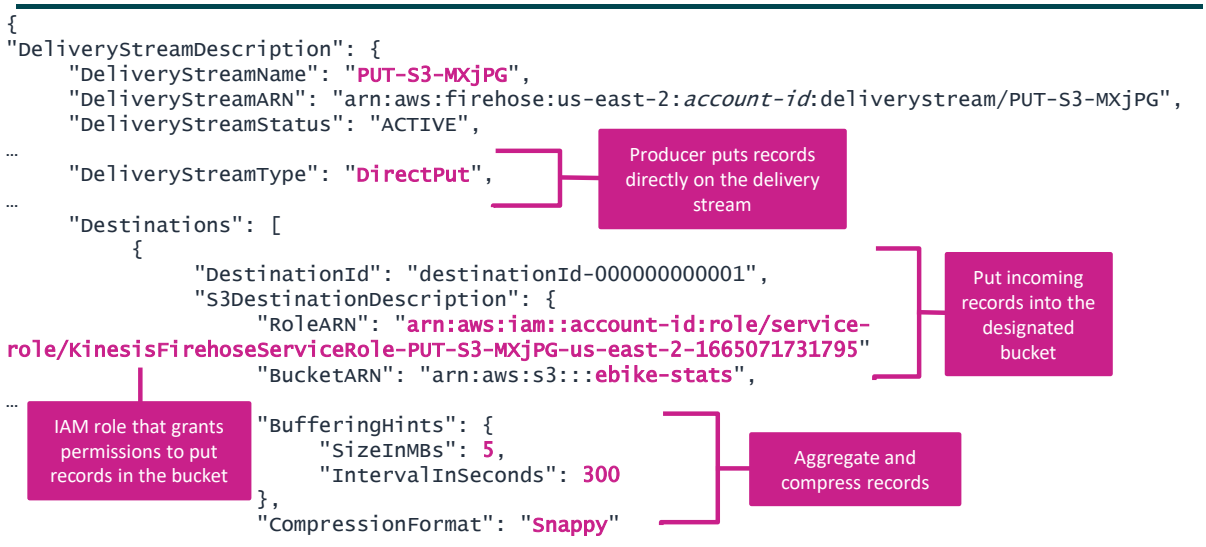
|Slide number 10
|Instructor notes
|
|Student notes

As part of configuring the Kinesis Data Firehose, you designate the location where the delivery stream should put the records that it receives. You can configure options on the delivery stream to determine how it should aggregate records and send them as a batch to Amazon S3. You can configure records to be aggregated based on a time interval or payload size.

With Kinesis Data Firehose, you also have the option to apply transformations to the data on the stream by using an AWS Lambda function before the data is delivered to Amazon S3. You can also transform the format of the data. For example, you might convert the JSON format of the messages to Apache Parquet or ORC. These formats are common for data lakes because they are columnar formats. Columnar formats can save space and enable faster queries compared to row-oriented formats, such as JSON. You can also specify a compression method to use on the records before delivering them to Amazon S3.

Note that an IAM role is needed to grant permissions to the delivery stream to write records to the S3 bucket that you select.

Configuration options for Kinesis Data Firehose



|Slide number 11

|Instructor notes

|The “Record Transformation and Record Format Conversion” page of the *Amazon Kinesis Data Firehose Developer Guide* provides more information about transforming data by using a Lambda function and converting record formats.

|

|Student notes

This slide illustrates excerpts of the Kinesis Data Firehose definition. This is the stream that is identified in the IoT rule that was described on the previous slides.

The top section identifies the name of the delivery stream and gives its unique Amazon Resource Name (ARN).

The next excerpt shows the stream type of **DirectPut**. This indicates that the producer application (the data source for the delivery stream) will write directly to the delivery stream. This option is available for many AWS services, including AWS IoT Core. For producers that cannot use DirectPut, you would put an Amazon Kinesis Data Streams stream as the data source in front of the Kinesis Data Firehose stream. You might also use a Kinesis data stream as the source for the Kinesis Data Firehose stream if you needed to deliver records to multiple consumers. However, in this example, because the only destination is an S3 bucket, DirectPut can be used.

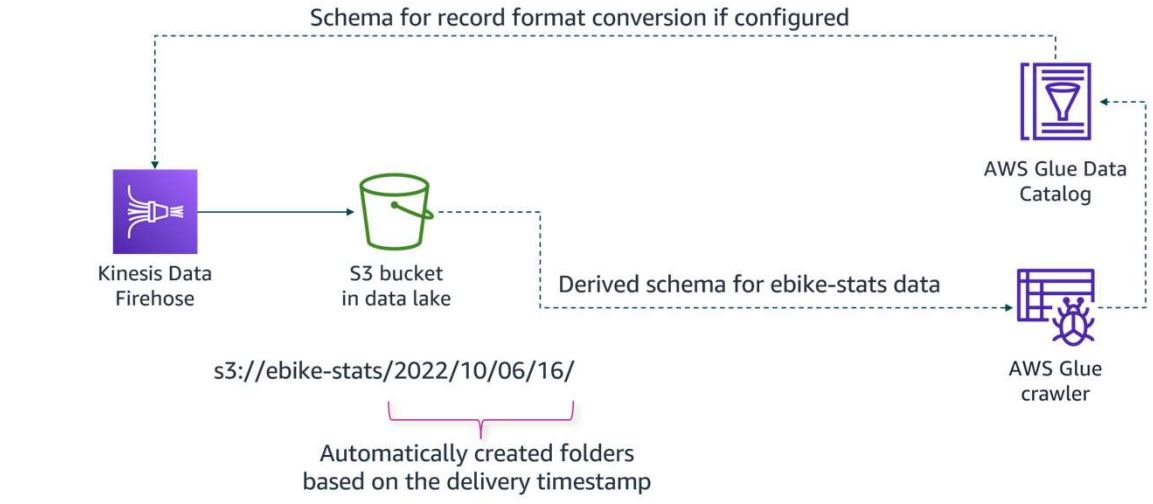
The **Destinations** section describes where the Kinesis Data Firehose delivery stream should deliver its records. In this example, they will be delivered to an S3 bucket called **ebike-stats**.

The **BufferingHints** section determines how the delivery stream batches your records based on size and interval. The **CompressionFormat** section indicates what—if any—compression method to use.

For the initial setup of this example, the default duration and payload size are used, and Snappy will compress the files before delivery. Lambda transformations or data format conversions are not included.

|Slide number 12
|Instructor notes
|
|Student notes
Now let's look at how the IoT data is stored and made available to consumers.

Organize and catalog data



|Slide number 13
|Instructor notes
|
|Student notes

As part of configuring the Kinesis Data Firehose delivery stream, you designate the bucket where data will be stored. When the data arrives in the bucket, prefixes (or folders) will automatically be generated within the bucket to organize the data that is being delivered.

To make the data discoverable within your data lake, the next step is to set up an AWS Glue crawler. The crawler will identify the schema and publish the schema and metadata to the AWS Glue Data Catalog. You can use the AWS Glue console to create a crawler and connect it to the ebike-stats bucket. You can set the crawler to run on a schedule based on how frequently you expect users to need new data in their reports. Initially, you might use the default of on demand for testing.

As noted in the previous section, you can configure Kinesis Data Firehose to convert the file format of records on the stream before delivering them to Amazon S3. AWS Glue provides this feature. When you configure record format conversion in your delivery stream, you designate an AWS Glue database and table to use. AWS Glue provides the schema in the Data Catalog that Kinesis Data Firehose then references to interpret the incoming data and convert it into the specified format before writing it to Amazon S3.

Now, your IoT data is stored in your pipeline. You have an ingestion pipeline to continually bring in the data in and process and transform it through the IoT rule and Kinesis Data Firehose transformations.

Example: Information for an AWS Glue crawler

```
{
    "Crawler": {
        "Name": "ebikecrawler",
        "Role": "service-role/AWSGlueServiceRole-ebikes",
        "Targets": {
            "S3Targets": [
                {
                    "Path": "s3://ebike-stats",
            ]
…
        },
        "DatabaseName": "default",
… },
        "State": "READY",
        "CrawlElapsedTime": 0,
        "CreationTime": "2022-10-22T16:50:57+00:00",
        "LastUpdated": "2022-10-22T16:50:57+00:00",
        "LastCrawl": {
            "Status": "SUCCEEDED",
            "LogGroup": "/aws-glue/crawlers",
…
```

Name of the crawler and role with permissions to access your S3 bucket

S3 location to crawl

AWS Glue database where results are written

Successful crawler run and the location of logs in CloudWatch

|Slide number 14

|Instructor notes

|You can create, edit, and view crawlers in the AWS Glue console. To produce the view on this slide view, the AWS Command Line Interface (AWS CLI) command was **aws glue get-crawler --name ebikecrawler**.

|

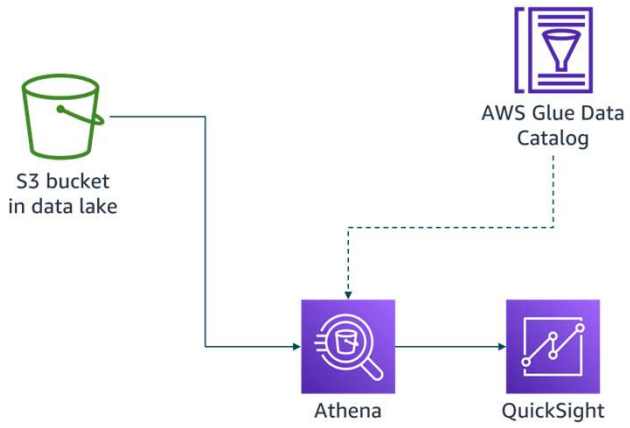
|Student notes

You can use the AWS Glue console to create a crawler. When you create the crawler, you can choose to automatically create the IAM role that you need to grant permissions to your crawler to access your S3 bucket.

You can also use the AWS Command Line Interface (AWS CLI) to work with crawlers. For example, the code sample on this slide reflects the results of running the following command from the AWS CLI: **aws glue get-crawler --name ebikecrawler**.

|Slide number 15
|Instructor notes
|
|Student notes
This section looks at how you can use Amazon Athena and Amazon QuickSight to work with the IoT data that is stored in Amazon S3.

Query and visualize data in Amazon S3



|Slide number 16
|Instructor notes
|
|Student notes
You can use Athena to perform queries on the data that is stored in the ebike/stats bucket. Because of the AWS Glue crawler that you set up (**ebikecrawler**), Athena can apply the schema from the Data Catalog and query the ebike-stats data.

In the Athena console, you can save queries and create views based on the data in the S3 bucket.

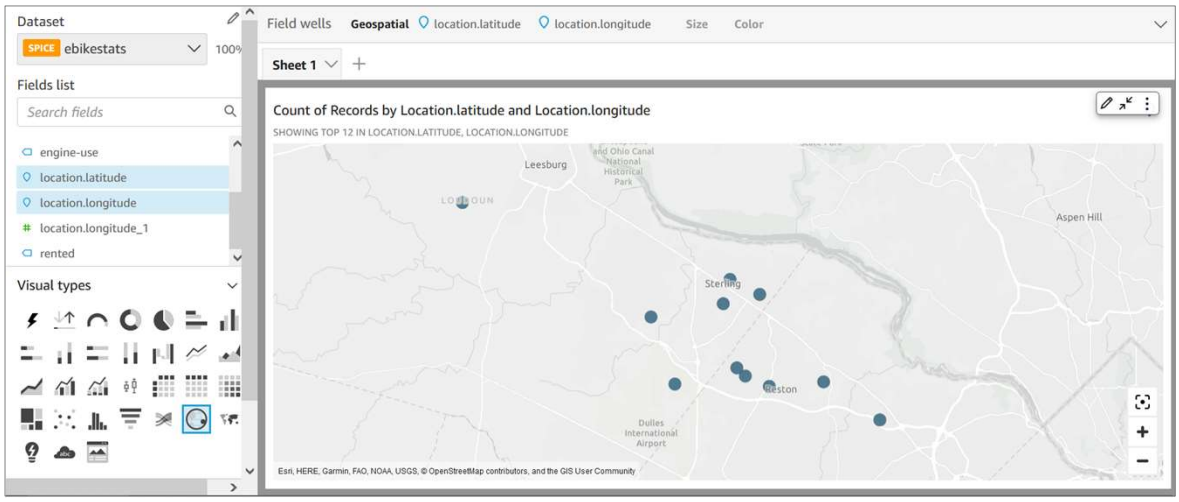
Example: Athena query

```
{
    "NamedQuery": {
        "Name": "rented_bikes",
        "Database": "default",
        "QueryString": "SELECT * FROM \"default\".\"ebike_stats\" where
\"column1.rented\" = true;",
        "NamedQueryId": "ccb0fb5c-827f-44e7-8af8-1b5ab5a8bfed",
        "WorkGroup": "primary"
    }
}
```


|Slide number 17
|Instructor notes
|You can use the Athena console to run the query that is shown on the slide.
|
|The code shown here was produced by using the following AWS CLI command to retrieve a saved Athena query: **aws Athena get-named-query --named-query-id**. You can find the named query ID in the Athena console for any query that you save.
|
|Student notes
This example shows information about a saved Athena query that is called **rented_bikes**. The query selects only bikes where **rented** equals **true** from the ebike-stats data.

By saving this query as a view in Athena, you can select it as a dataset in QuickSight.

Example: Geospatial visualization in QuickSight



|Slide number 18
|Instructor notes
|In the test data that was used for this example, after connecting the dataset, it was necessary to use the **Edit data** option to change the field types on the latitude and longitude fields. They came in as decimal fields and were modified to support the geospatial visualization option.
|
|Student notes
To work with the data in QuickSight, you can create a dataset by selecting a data source. For the example that is shown on the slide, an Athena view was used. After selecting the view, which was created from the **rented-bikes** query, as a dataset in QuickSight, you can further transform and visualize the data.

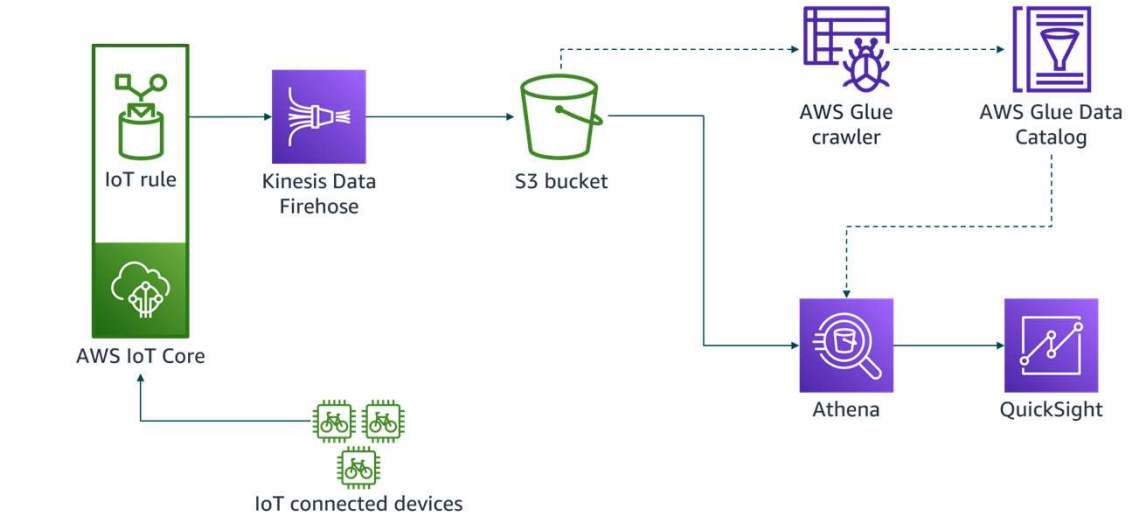
In this example, the "Points on map" visualization option can be used to plot the locations of rented bikes. With this visualization, the team can quickly see the distribution of where the bikes have been used.

This example was generated by using a small number of sample records, but you can see how plotting the incoming IoT data from the e-bikes could help identify patterns of where the bikes are used.

This visualization could be added to a dashboard and shared with appropriate sets of users from within QuickSight.

|Slide number 19
|Instructor notes
|
|Student notes
This section reviews the architecture that you walked through for this use case.

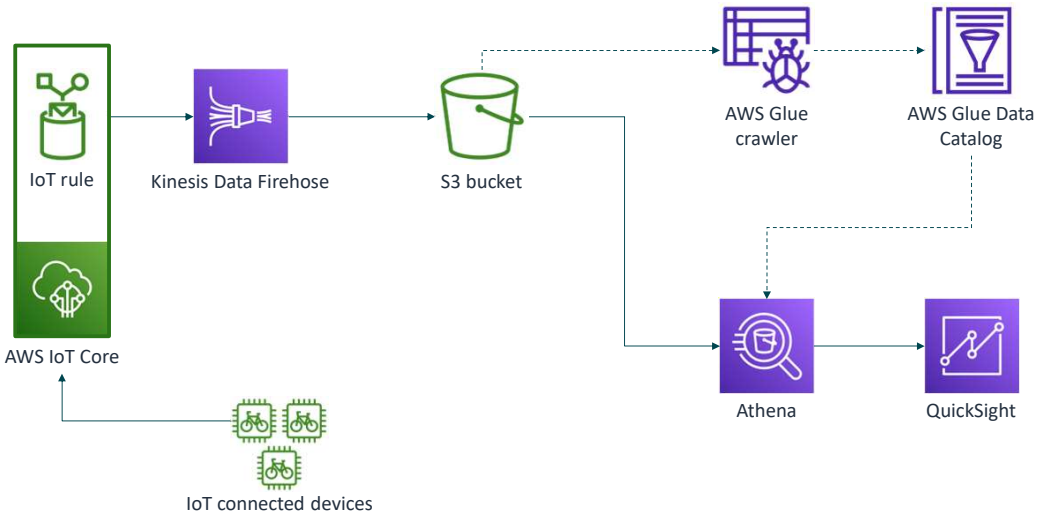
Review of the IoT reference architecture



|Slide number 20
|Instructor notes
|
|Student notes

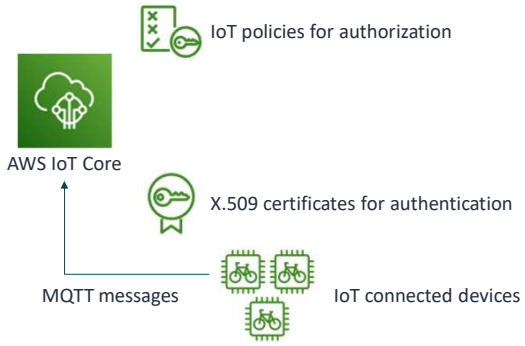
Through this use case, you have seen how AWS IoT Core can subscribe to a topic that is published by IoT devices on e-bikes. An IoT rule routes the data to a Kinesis Data Firehose delivery stream. Kinesis Data Firehose delivers the data to Amazon S3, where an AWS Glue crawler runs to derive the schema from the data. Athena can query the data without moving it from Amazon S3, by using the schema information in the AWS Glue Data Catalog. QuickSight can use Athena as a data source, and create and transform a dataset from the source data for a variety of visualizations.

Source: The IoT reference architecture

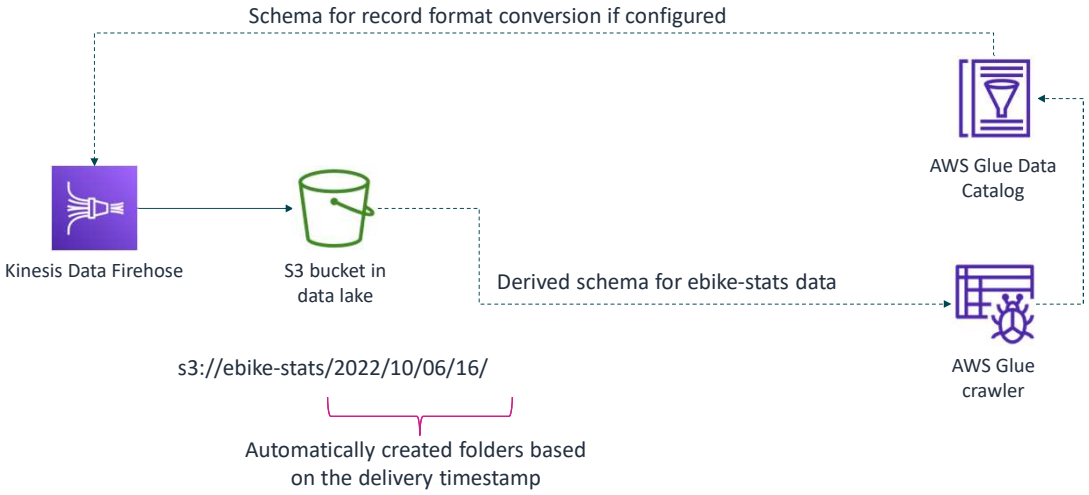


Source: Connect devices to AWS IoT Core

1. Connect IoT devices to AWS IoT Core, and verify that messages are being published to your topic



Source: Organize and catalog data



Source: Query and visualize data in Amazon S3