Internet of Things Platforms

Luciano Bononi, Marco Di Felice
(luciano.bononi@unibo.it)
Dept. of Computer Science and Engineering, University of Bologna
What is a IoT platform

- IoT platforms are collections of tools and services
  - To deal with IoT systems’ needs
  - They have standard ways to connect heterogeneous IoT end-devices
  - They provide standard APIs to allow data to flow in
  - Scalable: they provide storage and servers to deal with billions of devices and trillions of messages sent per hour
  - You pay what you use, and you can easily integrate new resources
  - They provide powerful data enrichment, analysis and AI tools fully integrated and easy to apply with a few clicks (not a deep programming capability required).
  - Powerful data dashboards and analysis (also visual dashboards) tools
  - Possibility to integrate other tools and platforms to realize even more extended functions
  - And more...
IoT Platform concept: everything in between devices and applications

Different Application Contexts (Industrial IoT, Industry 4.0, Home, etc.)

- Local IoT communication
- Internet Access Network
- IoT Communic. Gateway
- Internet
- CLOUD Services
- Applications
- Artificial Intelligence & Analytics
- Interoperable Services
- Data Transformation
- Digital Twins (device shadows)
- Data management
- Storage and Big Data
- Data Mgmt Service Platform (made of many integrated components)

Things (devices)
e.g. sensors/actuators
May be heterogeneous (types)
May be clustered or not (groups)
Must have unique Ids
Must be connected (IoT)

IoT Platform (left) border (towards the edge IoT): it depends on which platforms you use

SECURITY
Panoramic view of IoT Platforms

Leading IoT Platforms 2019
Based on customer reviews

Leaders
- Amazon AWS IoT
- Microsoft Azure IoT
- Oracle IoT Cloud
- Cisco Kinetic
- Google Cloud IoT
- AT&T M2X
- IBM Watson IoT
- SAP Leonardo IoT
- ETC Thingworx
- Huawei OceanConnect
- GE Predix
- Schneider Electric EcoStruxure
- Accenture CPaaS&AIP
- Bosch IoT Suite
- Alibaba IoT Cloud
- Siemens Mindsphere
- Software AG Cumulocity
- Vodafone IoT Platform
- Ericsson DC Platform
- C3 IoT Platform
- Dt Telekom/T-Systems CoT&HoC

Challengers
- Hitachi Lumada
- Orange IoT suite
- Telefonica IoT suite
- Vodafone IoT Platform
- Ericsson DC Platform
- C3 IoT Platform

Followers
- GE Predix
- Schneider Electric EcoStruxure
- Accenture CPaaS&AIP
- Bosch IoT Suite
- Alibaba IoT Cloud
- Siemens Mindsphere
- Software AG Cumulocity
- Vodafone IoT Platform
- Ericsson DC Platform
- C3 IoT Platform

Note: The ranking is entirely based on IoT Platform end-users views. User Centricity = Usability of the platform (incl. ease of use). Support (documentation, hotline, etc.), Value for money; Technology = Maturity of cloud services, application enablement, device management, connectivity management, data management & analytics, security, interoperability, and the completeness of the total offering (each considered independently). Source(s): IoT Analytics Research – July 2019

Credits: IoT Analytics
Overview of IoT platform building blocks

<table>
<thead>
<tr>
<th>Platform Type</th>
<th>Elements / Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advanced analytics platform</td>
<td>Streaming analytics, AI/Machine learning</td>
</tr>
<tr>
<td>Application enablement PaaS</td>
<td>Digital twins, Rules engine &amp; event management, Integrated development environment, Business app integration &amp; visualization, IoT application marketplace</td>
</tr>
<tr>
<td>IaaS / cloud backend</td>
<td>IoT hubs, Scalable storage, Multi-modal database support, Data processing</td>
</tr>
<tr>
<td>Device management PaaS</td>
<td>Deployment configuration, Device monitoring, Command &amp; control, OTA firmware updates, Edge application lifecycle</td>
</tr>
<tr>
<td>Connectivity platform</td>
<td>Connectivity orchestration, Connectivity management, Service provisioning, Billing management</td>
</tr>
</tbody>
</table>

Source: IoT Analytics

Credits: IoT Analytics
The 4 Stage IoT Solutions Architecture

The “Things”

- Primarily analog data sources
- Devices, machines, people, tools, cars, animals, clothes, toys, environment, buildings, etc.

<table>
<thead>
<tr>
<th>Stage 1</th>
<th>Stage 2</th>
<th>Stage 3</th>
<th>Stage 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensors/Actuators (wired, wireless)</td>
<td>Internet Gateways, Data Acquisition Systems (data aggregation, A/D, measurement, control)</td>
<td>Edge IT (analytics, pre-processing)</td>
<td>Data Center / Cloud (analytics, management, archive)</td>
</tr>
</tbody>
</table>

SW Stacks: Data Flow: Control Flow:

Credits: https://techbeacon.com/enterprise-it/4-stages-iot-architecture
Specialization of IoT platforms

Some IoT Platforms have been specialized for special contexts of application

Some other IoT platforms are more general purpose.

We will illustrate a short summary in the following
Google Cloud Platform

It provides a multi-layered secure infrastructure, predictive maintenance for equipment, solutions for smart cities & buildings, and real-time asset tracking.

**Features:**
- Machine learning capabilities for any IoT need.
- Real-time business insights for globally dispersed devices.
- AI capabilities.
- Provides support for a wide range of embedded operating systems.
- Location intelligence.

**Cost:** medium / high

**Overall:** Organizing, managing, and sharing documents is easy. It works with all operating systems. Overall it provides good features and functionalities and ease of use.

Credits: https://www.softwaretestinghelp.com/best-iot-platforms/
SalesForce IoT Cloud

- «transform all data which is generated by the customers, partners, devices, and sensors into relevant actions». It has partner connectors like AWS, Cisco Systems, etc.
- **Features:**
  - ✓ It allows you to test business ideas without programming.
  - ✓ It will provide you the real data about the product usage and performance.
  - ✓ It can work with the data from any device.
  - ✓ You can create device profiles for customer context data in CRM and for streaming data from the connected devices.
  - ✓ Using RESTful API, you can import data from any source.
  - ✓ No need of CS degree while creating and managing orchestration rules.
  - ✓ Real-time traffic view.
- **Overall:** Tool provides a good interface, ease of use etc. CRM in the cloud

https://www.salesforce.com/products/salesforce-iot/overview/
It helps in managing the development lifecycle for IoT applications.

- It provides flexibility to access data and IoT from on-premise, off-premise, and from the hybrid environment. Use of ThingWorx will give you increased uptime, reduced costs, role-based visibility & control, and improved compliance.

- **Features:**
  - Connect devices.
  - Analyze data.
  - Build and deploy solutions.
  - Industrial IoT and application data is accessible from on-premise web servers, off-premise cloud applications and as hybrid environments.

- **Cost:** medium

- **Overall:** It is a renowned solution for industrial IoT. With the help of ThingWorx, you can create an industrial IoT application fast. There is no need to write too many lines of code.

This platform will help you to capture and investigate the data for devices, machines, equipment and find out the understandings for better decisions.

This platform will allow you to optimize operations and resource. By providing the correct business insights and bidirectional communication facility, it will help in increasing the revenue to a great extent.

**Features:**
- AI and Analytics.
- Domain expertise.
- Provides flexible solutions.
- Provides security.
- Captures real-time data.
- Provides analytics service as an add-on.

**Cost:** medium

**Overall:** Platform provides good features and functionalities at an affordable price.
Amazon AWS IoT Core

- AWS IoT Core will help you to connect devices to the cloud.
- It is a managed cloud service. AWS IoT Core will allow devices to connect with the cloud and interact with the other devices and cloud applications. It provides support for HTTP, lightweight communication protocol, and MQTT.

**Features:**
- It can process a huge amount of messages.
- It is a reliable and secure platform to route the messages to AWS endpoints and other devices.
- Your applications will track and communicate even when not connected.
- You will be able to use other AWS services like AWS Lambda, Amazon Kinesis, and Amazon QuickSight etc.
- It allows secure access to your devices.

**Cost:** Medium/High. Metered services. 12 months free trial period is also available.

**Overall:** You can start in just three simple steps. Signup, learn from tutorials and start building. Tutorials are provided as a learning material. AWS IoT provides good integration options with other services.

Interesting high granularity metrics for payments: https://aws.amazon.com/it/iot-core/pricing/additional-details/#Connectivity
This IoT solution is designed for different industry needs. It can be used from manufacturing to transportation to retail. It provides solutions for remote monitoring, predictive maintenance, smart spaces, and connected products.

**Features:**
- It provides you with an open platform to build a robust application.
- It can be used by beginners as well as experts.
- There are two solutions to start with, as an IoT SaaS and with open source IoT Templates.

**Cost:** medium.

**Overall:** A free guide is provided on how to create IoT applications. The platform provides a good number of features and functionalities and it is easily scalable too.


With the help of Oracle IoT cloud, you can connect your devices to the cloud, perform analysis of data from these devices in real time, and perform integration of data with enterprise applications or web services. It supports integration with Oracle and non-oracle applications and IoT devices using REST API.

- **Features:**
  - It will allow you to create an IoT application and connect a device to JavaScript, Android, iOS, Java, and C POSIX.
  - It will help you to extend the supply chain, ERP, HR, and customer experience applications.
  - Operational efficiency and worker productivity will be improved.
  - It provides features like device virtualization, high-speed messaging, and endpoint management to connect.
  - To analyze the data, it provides features like stream processing and data enrichment.
  - Using REST API, integration can be done with Oracle and non-oracle applications and IoT devices.

- **Cost:** High. Price starts at $2.2513 OCPU per hour on a monthly basis. These prices are for Universal Credit services. For non-metered services, the prices start at $2500.

- **Overall:** It is a cloud-based service and is easy to use. It provides an integration option with Oracle and non-Oracle applications.

Cisco IoT Cloud Connect

- Cisco IoT cloud connect is a mobility cloud-based software suite. This IoT solution is for mobile operators. It will fully optimize and utilize the network. Cisco provides IoT solutions for networking, security, and data management.

- **Features:**
  - Granular and real-time visibility.
  - It provides updates for every level of the network.
  - For IoT security, it provides benefits of protecting the control system from human errors & attacks, increased visibility & control by defending malware and intrusion, and centralized security controls.

- **Cost:** Medium/High.

- **Verdict:** Cisco IoT cloud connect is for networking, security, and data management and provides updates at every level of the network.

Altair SmartWorks

- end-to-end IoT platform as a service.
- connect devices, collect data, manage devices and data, and build and run the app. It provides functionalities like device management, Listeners, rules, custom alarms, triggers, and data export etc.

- **Features:**
  - Using SmartWorks you can connect with any devices like sensors, gateways, machines, etc.
  - Using REST API, you can send XML or JSON data.
  - It has an open architecture.

- **Cost:** Free for two devices.

- **Overall:** Platform is easy to use and provides good features and functionalities.

https://www.altairsmartworks.com/
ThingSpeak

A simple platform for data collection and cloud services for advanced data analysis using MATLAB

ThingSpeak Features

- Collect data in private channels
- Share data with public channels
- RESTful and MQTT APIs
- MATLAB® analytics and visualizations
- Event scheduling

Works With

- MATLAB®
- Arduino®
- Particle Photon and Electron
- ESP8266 WiFi Module
- Raspberry Pi™
- LoRaWAN®
- Things Network
- Senet
- Libelium
- Beckhoff

About ThingSpeak

ThingSpeak is an IoT analytics platform service that allows you to aggregate, visualize, and analyze live data streams in the cloud. You can send data to ThingSpeak from your devices, create instant visualization of live data, and send alerts.

Collect

Send sensor data privately to the cloud.

Analyze

Analyze and visualize your data with MATLAB.

Act

Trigger a reaction.
AWS IoT platform

Some slides' content and images credits
In these slides we introduce you to the topics and concepts within the AWS Internet of Things (IoT) platform domain

- assumption: baseline knowledge of IoT topics
  - e.g. do you know the MQTT broker role? Publish/subscribe mechanisms?
  - e.g. do you have basic knowledge of communication networks and protocols?
  - e.g. do you know LoRa/LoRaWan or NB-IoT and Internet-based basic protocols?
- to learn AWS IoT terminology, AWS IoT core services and their functions, IoT protocols and communication solutions, and IoT security fundamentals
- to create a functional AWS IoT solution to solve the presented challenges
- by the end of the lesson, you will have a basic foundation in the AWS IoT services and how they integrate with other AWS services
- light focus on deeper content (such as AWS IoT Rules Engine or IoT Security)
What you will learn in the next steps

- More specifically:

**Prerequisites** (first class content)

- Introduction to AWS IoT
- A basic knowledge of the AWS console

**Course objectives** (first and second class content)

- Register a sensor within the AWS IoT Core
- Secure the sensor using IoT policies
- Establish communication with the sensor
- Collect data from the sensor
- Redirect the data using the AWS IoT rules engine
- Visualize the data using AWS IoT Analytics and Amazon QuickSight

Online reference material (and credits for the examples): [https://aws.amazon.com/it/training/](https://aws.amazon.com/it/training/)
In this class we will illustrate 7 modules

Module 1: Introduction to AWS IoT Telemetry
  • Introduction to the Foundation Series
  • The telemetry scenario
  • Business outcome

Module 2: AWS IoT Core Overview
  • Things and devices
  • Demonstration: Create a thing

Module 3: Device Security
  • Device security
  • Demonstration: Applying security

Module 4: Message Broker and AWS IoT Device Registry
  • Message broker
  • AWS IoT device registry
  • Demo

Module 5: AWS IoT Rules Engine
  • Completed business objectives
  • Rules engine overview
  • Demonstration: Applying rules

Module 6: Presenting the Data
  • AWS IoT Analytics
  • Demonstration: Visualizing the Data
  • Establish Communication with Our Thing

Module 7: Best Practices
  • Best Practices for AWS Telemetry
  • Knowledge check
  • Feedback
Telemetry scenario

E.g. example of cold chain maintenance for ice cream products

- we are going to use a microcontroller to record measurements from the following sources:
  - The freezer door with a digital I/O switch so that you can tell how long the door is open.
  - The on-board freezer’s internal temperature so that we see how well the trailer maintains its temperature.
  - The voltage going to the freezer unit so that we know if the freezer is being starved for power. And finally, a time and date stamp, which will tell you exactly when these problems occur so that you can correlate them with specific drivers and weather patterns.

- **Best practices to be defined**
  - Resolution of readings to have good data quality and support services (e.g. 5 secs)
    - Find a good compromise with bandwidth and data storage (as well as energy)
  - take measurements from all sensors and assemble them into a single string to send to AWS IoT.
    - Contextualize correlated data with time-coherence of observations (also good for getting time series data)
    - reduce communication overheads w.r.t. multiple separated packets
    - Use of UTC time (universal time) will make data transversal w.r.t. to time zones in future scalable uses (worldwide).
Considered the limitation of resources and scalability of messages, e.g. bottleneck communication to Lora gateways, as well as billing of AWS IoT and other cloud platforms.

Hint: try to adopt these **good practices** (at least from device to gateway, the semantic of information can be enriched in the gateway after reception of data, or by the cloud services in AWS IoT via the rules engine as we will see later).

Some examples:

- **Standard way by using MQTT JSON messages:**
  - `{«Counter»: 3423, «Temperature»: 22.345 } => 40 Bytes message, max 292 messages/day with LoRa (SF7)

- **Remove info which can be differently obtained** (e.g. counter as the header of message), spaces, compression of attribute names, rounding and approximation of values:
  - `{ «Counter»: 3423, «Temperature»: 22.345 } => 11 Bytes message, max 486 messages/day with LoRa (SF7)

- **Avoid to use JSON at all**
  - `{ «Counter»: 3423, «Temperature»: 22.345 } => 5 Bytes message, max 582 messages/day with LoRa (SF7)

- **Send raw 16 bit integers**
  - `0x080F` => 2 Bytes message, max 648 messages/day with LoRa (SF7)
what other kinds of information should you consider?

- Communication can be **bidirectional**
  - Collect information from trucks transporting ice creams
  - Send commands to trucks and perform actions

- First: **What data-driven actions** do you think might be useful in a food-service distribution truck?
  - Inform about ongoing emergency situation
  - Door open (thief, inaccurate checklist, etc.)
  - Risks for safety
  - Possibly collect information to identify the cause of problems, etc.
List of all your current (and future) questions about the system maintenance, e.g.:
- Are the frozen goods always melting at the same location on the route? If so, you will know where to look for clues.
  - Temperatures could go out of range while the truck is stuck in traffic, idling at a specific customer site, or cross-docking at a service center.
  - A simple GPS chip can provide this additional information at a low cost.
- Do problems happen only on hot days? An external temperature sensor located under the chassis can give you reliable outside temperature data to answer this question.

Possible actions (remedial): send info from the AWS cloud like
- A text message to the driver to check whether the freezer door is fully closed and latched.
- A message to the truck to switch to the backup battery bank or to turn on a warning light in the cab.

These are just one possible example...
To achieve a viable business outcome for the small ice cream distributor scenario:

- need to collect data from a sensor and format the data for consumption by the non-technical business manager.
- The business manager reviews the data to make process improvement decisions.
- Therefore, you must complete the following tasks:
  1. Establish communication with the IoT sensor.
  2. Collect and transform the data.
  3. Summarize and format the data for the business manager.

This is a good example of what AWS IoT is made for...
More specifically, to achieve your business objectives you need to:

- Register a sensor within AWS by adding a device using the AWS IoT console.
- Secure the sensor by using roles, policies, and certificates, in the AWS Identity and Access Management (IAM).
- Establish communication with the sensor by modifying an existing policy to suit the business objective.
- Collect data from the sensor by establishing communication to the device and collecting data.
- Transform and visualize data by using AWS IoT rules engine to transfer the data into AWS IoT Analytics and Amazon QuickSight. This enables you to transform and visualize the data into an easily readable format for the business decision-maker.
- Review best practices by identifying your IoT accomplishments and discussing AWS best practices for telemetry.

Now, we will see in next slides and demo how to explore the AWS IoT Core services and how you can use them to build a simple solution (also for the customer's ice cream).
AWS IoT Core
What is AWS IoT Core

- **AWS IoT Core**
  - It is a **managed cloud service** that lets connected devices easily and securely interact with cloud applications and other devices.
    - can support **billions of devices and trillions of messages**, and
    - can **process and route** those messages to AWS endpoints and to other devices reliably and securely.
    - It **supports your applications** to keep track of and communicate with all your devices, all the time, even when they are not connected.

- **AWS IoT Core** also makes it easy to use AWS and other Amazon services, e.g.
  - AWS Lambda (runs serverless code in response to events)
  - Amazon Kinesis (collect, elaborate and analyse data flows)
  - Amazon S3 (simple storage service)
  - Amazon SageMaker (machine learning modeling)
  - Amazon DynamoDB (noSQL DataBase)
  - Amazon CloudWatch (monitoring, observation for insights and management)
  - AWS CloudTrail (logging and monitoring user actions and use of APIs, conformance)
  - Amazon QuickSight (Business Intelligence and machine learning-based data analysis)
  - Alexa Voice Service (simply add voice interactions as MQTT messages with AWS)

allowing to build IoT applications that gather, process, analyze and act on data generated by connected devices, without having to manage any infrastructure.
How AWS IoT Core works

- Connect and Manage your devices

- easily connect any number of devices to the cloud and to other devices.
- supports HTTP, WebSockets, and MQTT (a lightweight communication protocol specifically designed to tolerate intermittent connections),
- minimize the code footprint on devices, and reduce network bandwidth requirements.
- supports other industry-standard and custom protocols, and devices can communicate with each other even if they are using different protocols.
Secure device connection and data

AWS IoT Core provides connected devices with automated configuration and authentication,
...as well as end-to-end encryption throughout all points of connection, so that data is never exchanged between devices and AWS IoT Core without proven identity.
In addition, you can secure access to your devices and applications by applying policies with granular permissions.
you can filter, transform, and act upon device data on the fly, based on business **rules** you define. You can update your rules to implement new device and application features at any time.

AWS IoT Core makes it easy to use AWS services like AWS Lambda, Amazon Kinesis, Amazon S3, Amazon DynamoDB, Amazon CloudWatch, and Amazon Elasticsearch Service for even more powerful IoT applications.
AWS IoT Core device’s state management

Read and set device state (at any time)

- AWS IoT Core stores the latest state of a connected device so that it can be **read or set at anytime**
- the device appears to your applications as if it were online all the time.
- **Device Shadow (or Digital Twin)** concept: This means that your application can read a device’s state even when it is disconnected, and also allows you to set a device state and have it implemented when the device reconnects.

Figure credits: AWS IoT
AWS IoT Core and Alexa integration

- Scalability to millions of devices integration with Alexa

The Alexa Voice Service (AVS) Integration for AWS IoT Core introduces a new virtual Alexa Built-in device in the cloud.
- a new set of AWS IoT-reserved MQTT topics to transfer MQTT-based audio messages between devices connected to AWS IoT Core and the new virtual Alexa Built-in device.
- This allows customers to send and receive audio messages over the reserved MQTT topics, interface with the device microphone and speaker, and manage the device-side state all while using the same secure IoT Core connection.
AWS IoT Core provides secure, bidirectional communication between internet-connected devices, such as sensors, actuators, embedded microcontrollers, or smart appliances, and the AWS Cloud. This enables you to collect, store, and analyze telemetry data from multiple devices. You can also create applications that enable your users to control these devices from their phones or tablets. AWS IoT Core is composed of six main components:

1. **Identity and access management** - Provides authentication, authorization, and easy onboarding of devices.
2. **Device gateway** - Securely connects IP-connected devices and edge gateways to the AWS Cloud and other devices at scale.
3. **Message broker** - Processes and routes data messages to the AWS Cloud.
5. **Device shadow** - Enables applications to interact with devices when they are offline through a digital twin.
6. **The registry** - Helps to enable device registration, and acts as a catalog of devices and their security certificates.

Through these six components, you can secure your data and interactions, process and act upon device data, and enable applications to interact with devices even when they are offline. Let’s quickly explore each of these components in more depth.
Concrete example (online AWS tutorial)

If not possible to access online AWS system, follow the next images for your reference (Onboard a Device).

Things can communicate with each other via the Device Gateway, even if they are using different protocols. The example at the left illustrates two things -- a connected light bulb, and a control unit -- both connected to the Device Gateway. The control unit can publish commands into the Device Gateway, and the light bulb can subscribe and listen for relevant commands.

Actuate commands (RGB switching of light bulb)

AWS IoT Device Gateway enables devices to securely and efficiently communicate with AWS IoT

RGB Light Bulb + Controller

AWS IoT

Luciano Bononi 2020
Concrete example (online AWS tutorial)

If not possible to access online AWS system, follow the next images for your reference.

The rules engine evaluates inbound messages published into AWS IoT, transforms and delivers them to another thing or a cloud based on business rules you define.

The example illustrates a rule that: 1) Evaluates commands published by the control unit, 2) Determines whether the command is "B", 3) If the command is "B", the rule transforms the message to "G" and relays "G" to the light bulb.
Concrete example (online AWS tutorial)

If not possible to access online AWS system, follow the next images for your reference.

The rules engine can also route messages to cloud endpoints such as AWS Lambda functions, or a DynamoDB table. The example shows the addition of a second rule that: 1) Evaluates commands published by the control unit, 2) Determines whether the command is "R", 3) If the command is "R", the rule delivers copies of the message to 3 endpoints -- a DynamoDB database table, a Lambda compute function, and Simple Notification Service (SNS) for push notifications to a mobile device.
Concrete example (online AWS tutorial)

If not possible to access online AWS system, follow the next images for your reference (Onboard a Device).

The AWS IoT includes **Device Registry** and **Device Shadows**, so you can register any thing you wish to represent in the cloud with a **name**, some **attributes**, and a persistent virtual 'shadow'.

The example shows that a thing has been created to represent the physical light bulb, with a virtual counterpart in the cloud.

Try turning off the physical light bulb. You'll notice that the Device Shadow remembers the color of the physical light bulb.

Next, try sending new "R" "G" and "B" commands from the control unit, with the physical light bulb still turned off.

The shadow will persist the desired future state of the bulb. When you **turn the physical bulb back on**, AWS IoT will command the physical bulb to **match its shadow**.

---

**Device Gateway**

**Rules Engine**

**Rule actions**

**Device Shadows**

**Build solutions**

**Done**

---

Enable or Disable the rules

- e.g. Simple Notification Service
- e.g. Lambda Compute Service
- e.g. Dynamo DB Service
- ...and many many others.

Switch on/off the physical light bulb

Device Shadow

(updates the light bulb color set and later re-synchronize with the physical bulb)
Concrete example (online AWS tutorial)

If not possible to access online AWS system, follow the next images for your reference (Onboard a Device).

AWS IoT makes it easy to build companion applications (Mobile Apps) that interact with your connected things. The example at the left shows a mobile application that reflects the color of your light bulb. The mobile app never communicates directly to the light bulb. Rather, the mobile app uses a REST API to read and set the state of the bulb's Device Shadow.
1- Identity and access management
AWS IoT Core provides a **secure communication channel** for devices to communicate with each other and other services. AWS IoT provides **authentication** by offering the following options:

- Certificates for mutual authentication by using Message Queuing Telemetry Transport (MQTT) over Transport Layer Security (TLS) v1.2
- SigV4 over HTTP
- MQTT over WebSockets, which is similar to other AWS services.

You can also use custom authentication tokens that are provided by your authentication or authorization service. AWS IoT also provides **flexible authorization options** and **fine-grained access control** through JavaScript Object Notation (JSON) policies.
The **AWS IoT device gateway** enables devices to securely and efficiently communicate with AWS IoT Core.

With the help of **AWS IoT message broker**, the IoT Device Gateway can exchange messages using a publish/subscribe model, which enables one-to-one and one-to-many communications.
AWS IoT message broker processes and routes data from your devices into AWS IoT Core. Message broker is scalable, has low-latency, and provides reliable message routing. It also uses a publish and subscribe model to decouple devices and applications.

The message broker allows a two-way message streaming between devices and applications and provides an opportunity for data transformation, re-routing, and enhancement with external data sources.
The **rules engine** listens for incoming messages that match a rule.

When a matching message is received, the **rule acts on the data**, such as writing data to an Amazon Simple Storage Service (Amazon S3) bucket, invoking an AWS Lambda function, or sending a message to an Amazon Simple Notification Service (Amazon SNS) topic (see previous example).
Often referred to as a **thing shadow** or a **digital twin**.

The Device Shadow service maintains a shadow for each device you connect to AWS IoT. A shadow is a memory log of the device status.

You can use the shadow to get and set the state of a device over MQTT or HTTP, regardless of whether the device is connected to the Internet. Each device's shadow is uniquely identified by the name of the corresponding thing.
The **AWS IoT registry** is a database of devices.

Using the registry for your devices is **optional**; however, the registry helps you to manage your device ecosystem effectively and acts as a repository for device certificates.

The registry also enables you to search for your *things* based on attributes and tags.
1-g AWS IoT components’ and services’ Summary

- AWS IoT Core
- AWS IoT device gateway
- AWS IoT message broker
- AWS IoT rules engine
- AWS IoT device shadow
- AWS IoT registry

Figure credits: AWS IoT
Things and Devices
Things vs. Devices

Things and Devices are often interchangeable terms, but they have distinctive definitions.

In practice:

- **Devices** are the **physical entities**, e.g. electronic components
  - physical devices or sensors (e.g. a light bulb or a switch on a wall)
- **Things** are **representations** of specific devices or logical entities.
  - physical devices or sensors (e.g. a light bulb or a switch on a wall)
  - logical entities (e.g. an instance of an application)
  - physical entity not directly connected to AWS IoT Core but related to devices that do connect to AWS (e.g., a car that has engine sensors on AWS).
Things information as JSON data in registry

Information about a thing is stored in the registry as JSON data (see just some examples below)

What is a thing?

A thing is identified by a name.

tethingName: truckSensor01

Here you see a name:value pair with "thingName" as the name identifier and "truckSensor01" as the name of the thing.

Thing attributes

A thing also can have attributes that store information about the thing. In this example, under attributes, you see the name:value pairs listed as "deviceId" with the value of T001 and "powerRequirements" with the value of 12v (volts). Other examples might include the device manufacturer, serial number, or the date received.

These thing attributes are not searchable.

You will learn more about this topic later in the course, but for now, remember to use the thingName as the defaultClientID for both the thing registry and the Device Shadow service. This makes fleet management easier for you as you increase the number of IoT devices.

Figure credits:
AWS IoT

Luciano Bononi 2020
Thing Types and Attributes

Thing type: A **thing type** is a method to organize IoT things into **logical categories**, such as light bulbs, thermostats, and motion sensors. These are groups of **things that share common attributes**. They have limitations:

- **Thing type limit**: you can associate things to only one thing type at a time.
- **Attribute limits**:
  - Things types can have up to 50 attributes.
  - There is no limit on the number of thing types you can create in your account.
  - Things that are not associated to a Thing Type can have only three attributes.
- **Searchable attributes**
  - Unlike the «name : value» attributes in the previous code example, **you can search the attributes of things that are associated with the same thing type**.

...continue
Naming: names for thing types do not require a universally unique name. However, they must be unique within your account.

**Rule:** Standardize your naming convention before the number of devices in your fleet grows too large. Doing so eases management, communication, and access rights. The standardization eliminates confusion when maintaining a large device fleet. (better than needing to change names later... because... see below)

Name Changes: After you create a thing type, you cannot change its name.

Removing thing types: you can associate or disassociate a thing from a thing type.

- Hint: disassociate your things from the thing type before deprecating or removing the thing type. After a thing type is deprecated or removed, you can decide whether to keep it for historical purposes or delete it. And maybe later you can give the thing a different thing type!

**Note:** You must wait five minutes after removing a thing type before it is eligible to be deleted.
Summary (example): Thing Type

Immutable thing types: you cannot change a thing type after creation

You can store description and configuration information for all the things with same type.

You can search for thing types by `thingTypeName` or by one of their `searchableAttributes`

You cannot change names of types, but you can deprecate a thing type for future things. All previous things remain unchanged.

More info on Thing Types on AWS IoT: [https://docs.aws.amazon.com/iot/latest/developerguide/thing-types.html](https://docs.aws.amazon.com/iot/latest/developerguide/thing-types.html) (pag 96)
Use **thing groups** to manage several things at once by categorizing them into groups. Groups can also contain groups, which enables you to build a hierarchy of groups.

- Inheritance of policies from groups to sub-groups (easy permissions)
- Things can be associated to up to 10 groups at the same time
- Similarly to thing types, thing groups can have non-unique names which cannot be changed after creation.
- However, things can be re-assigned to different groups.
- To possibly delete a thing group, no things must be existing with that group assigned.
Immutable thing group name: Can be created, updated, deleted

Version of the group (start at 1). Incremented each time it is updated.

(non mandatory)
Parent group of this group. Hierarchy tree (allowed).

Array of the full hierarchy of parent groups

Contains the group names and the Amazon Resource Name (ARN).

More info on Thing Types on AWS IoT: https://docs.aws.amazon.com/iot/latest/developerguide/thing-types.html (pag 96)
Let’s create a Thing in AWS IoT.
- We’ll use a thing type already existing
- Access AWS IoT
  - Services > Internet of Things > IoT Core > Onboard > Get started
    - use the Linux/OSX platform and the SDK (select Python - other options include NodeJS and Java)
    - Some prerequisites to consider: the device should have **Python and Git installed** and a **TCP connection to the public internet on port 8883**.
  - Choose Next
    - we’ll give our thing a name and call it **truckSensor01** (see option truckSensor02 already created as emergency)
  - choose the **Show optional configuration** link below the name
    - You’ll notice that after selecting this thing type, the **Set searchable thing attributes** appear. We’ll fill those in, in a moment. First, select **Thing type** and create or select a pre-created type (e.g. if available, select **sensorThingType**).
    - Add device ID, power reqs, trailer capacity and other parameters.
    - Add additional attributes (if any)
    - In case the thing has no type we can use the attributes under **Set non-searchable thing attributes**
  - Click **Next Step** to download the **connection kit** for linux/OSX
    - Gives public/private keys and initial script for connections (not needed now, we’ll see in next demos).
    - Look how to work on connection kit. (unzip it and chmod +x start.sh and then execute ./start.sh)
    - See the script that simulates data flowing (start.sh – you can execute it just to test the connection).
    - You may need «–user» in the start.sh script for command **python setup.py install --user**
    - Go on AWS IoT in **Test > Subscribe to topic** «sdk/test/Python» and see messages flowing in.
  - Choose **Done**

You created your first AWS IoT Thing!
- Go back on AWS IoT and look for truckSensor01 created, open it and scroll the attributes.
AWS IoT Core
Devices’ Security and Data Security
AWS IoT Core Devices’ Security

AWS IoT Core uses **certificates** to grant access from the device to the core components.

After the devices have been **authenticated**, AWS IoT Core policies **control which topics the device can publish and subscribe**: e.g. AWS Identity and Access Management (IAM) roles allow AWS IoT to access other AWS resources in your account on your behalf. For example, if you want to have a device publish its state to a DynamoDB table, IAM roles allow AWS IoT to interact with Amazon DynamoDB.

**All communication is encrypted** with **Transport Layer Security (TLS) version 1.2** to ensure that your **application protocols remain secure and confidential**. Not all devices support TLSv1.2, so when adding devices to your fleet, ensure that they support TLSv1.2.

Authentication and authorization perform different roles in securing the devices. Next slides provide more detail.
**Device Authentication**

**Authentication** enables every entity A to verify the identity of a client C or a server S.

**Server authentication** is the process by which devices or other clients ensure that they are communicating with an actual AWS IoT endpoint.

**Client authentication** is the process by which devices authenticate themselves with AWS IoT Core.

**Certificates** allow devices to connect to the AWS IoT Core. Certificates provide several benefits over other identification and authentication mechanisms (see next slides)
Assumption: you know the symmetric (e.g. AES) and asymmetric (e.g. RSA) encryption technologies. You know how to use symmetric (shared key) technologies (via Key Distribution Centers) and you know the differences with asymmetric public/private key schemes (via Certification Authorities), for authentication and encryption. Bonus: you know how «man in the middle» attacks work.

Certificates enable asymmetric keys to be used with devices, which means that you can transfer private keys into secure storage on a device.

This way, sensitive cryptographic material never leaves the device, in particular the private key (zero knowledge methodology).

Certificates also provide stronger client authentication over other methods, such as user name and password or bearer tokens, because the secret key never leaves the device. Also, they avoid «man in the middle» attacks by certifying the source (certification authority) and owner (device identity) of keys.
AWS IoT Core uses **X.509 certificates**

- a public key infrastructure standard
- It associates a public key with an identity contained in a certificate.
- X.509 certificates are issued by a third party called a certification authority (CA). The CA maintains one or more special certificates called CA certificates that it uses to issue X.509 certificates. Only the certification authority has access to CA certificates.

X.509 certificate chains are **used both** for server authentication by clients and client authentication by the server.

- Certificates are created within **AWS IoT Core** and downloaded to the device through TLS v1.2. Before the device can authenticate, the certificate must be activated.
- In addition, **certificates can be revoked** if there’s concern that the device has been compromised or is being decommissioned.
Authorization is the process of granting permissions to an authenticated identity.

- After identities are authenticated, they still do not have permissions to access resources.
- For devices and users to gain permissions, they must be authorized to do so.

Policies for AWS IoT define what an authenticated identity can do.

- Devices, mobile applications, web applications, and desktop applications all use an authenticated identity.
- The identity can execute AWS IoT operations only if it has a policy that grants it permission.
Authentication and authorization methods for users and devices: you can grant permissions using AWS IoT by

- identifying roles, policies, security credentials and IAM policies to devices and users.
- Both IAM policies and IoT policies are used to control different entities.
**AWS IoT policies**: control identity access to the IoT data plane.

- In AWS IoT, the **data plane** enables you to send data to and receive data from AWS IoT: it defines whether you can connect to the message broker, send or receive MQTT messages, or publish or subscribe to a specific topic.

**IAM policies**: grant an IAM identity, or AWS resource, permissions to access a service or perform a function with AWS.

**IAM roles**: work with IAM policies to grant AWS IoT Core permissions to access the rest of AWS.

- For example, AWS IoT receives data and communication from a device that is redirected to an AWS resource such as an S3 bucket, a DynamoDB table, or any AWS service.
Policies are **JSON documents** that contain a set of permissions (policy statements).
- The policies are attached to the device, user, or role to give them the permissions written in the policy.

Each statement contains:
- **Effect** - Allow or Deny access.
- **Action** - list of actions that the policy allows or denies.
- **Resource** - Specifies a list of resources to which the actions apply.
  - If no resource is listed, then the resource to which the action applies is the resource to which the policy is attached.

---

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["iot:Publish"],
        },
        {
            "Effect": "Allow",
            "Action": ["iot:Connect"],
        }
    ]
}
```

---

Figure credits: AWS IoT
AWS IoT also defines a set of **policy actions** that describe the operations and resources to which you can grant or deny access. Some examples:

- **iot:Connect** represents permission to connect to the AWS IoT message broker
- **iot:Subscribe** represents permission to subscribe to an MQTT topic
- **iot:Publish** represents permission to publish to an MQTT topic
- **iot:GetThingShadow** represents permission to get a device's shadow
Apply security certificates to the thing created in the previous demonstration.

- IoT > Secure > and look the certificate and policy created for the thing created
- Update the IoT policy to publish and subscribe to specific topics.
- IoT > Secure > Edit policy document
- copy the certificates to the device.

**Note:** copy the certificates and keys in the folder where the script is for execution on the device memory.

**Good practice:** only publish and allow the minimum information needed or produced for some reason.

As soon as policies are locked down and key and certificates are loaded the device is ready for sending and receiving data.
Message Broker, MQTT protocol and AWS IoT Device Registry
AWS IoT Message Broker

The AWS IoT message broker allows clients to communicate with AWS IoT Core and for AWS IoT Core to communicate with clients.

- Clients send data by **publishing** a message on a topic.
- Clients receive messages by **subscribing** to a topic.
- When the message broker receives a message, it forwards the message to all clients subscribed to the topic.

Topics: hierarchical strings (UTF-8) that use a forward slash (/) to separate the levels in the hierarchy. E.g. `sensor/temperature/room1`

- The message broker maintains a list of all client sessions and the subscriptions for each session.
- When a message is published on a topic, the broker checks for sessions with subscriptions that map to the topic.
- The broker then forwards the published message to all sessions that have a currently connected client.
Broker protocols

- The AWS IoT message broker supports the use of the:
  ✓ Message Queuing Telemetry Transport (MQTT) protocol to publish and subscribe

  - MQTT is a widely adopted, lightweight messaging protocol designed for constrained devices. Open Standard for publish/subscribe paradigm.
  - MQTT must be secured externally (e.g. via TLS)
  - HTTPS protocol to publish.
  - Both network protocols IP version 4 and IP version 6.
  - The message broker also supports MQTT over the WebSocket protocol to enable browser-based and remote applications to send and receive data using AWS credentials.
  - The way you connect to the message broker depends on the protocol you are using. Different protocols require different authentication mechanisms.

http://mqtt.org/
AWS IoT provides a **scalable device registry** that helps you manage things and organizes the resources associated with each device in the AWS Cloud.

- The **AWS IoT device registry** is a catalog of static metadata and attributes about the devices, such as serial numbers, manufacturer, firmware version, internal identifier (or badge number), and device capabilities, **including certificates**, written in JSON.

- You register your devices and associate up to three custom attributes with each one. You can also associate certificates and MQTT client IDs with each device to improve your ability to manage and troubleshoot them.

[JSON example]

```json
{
    "version": 3,
    "thingName": "MyLightBulb",
    "defaultClientId": "MyLightBulb",
    "thingTypeName": "LightBulb",
    "attributes": {
        "model": "123",
        "wattage": "75"
    }
}
```

https://docs.aws.amazon.com/iot/latest/developerguide/register-device.html
AWS IoT Device shadow

- AWS IoT Core saves the thing state and parameters within the device registry...
- ..and also in device shadow for devices that connect by using MQTT or WebSockets.
- A device's shadow is a JSON document that is used to store and retrieve current state information for a device and can be used to update a device that is offline or network-unreachable.
How to subscribe to an MQTT topic and verify communications to the device in AWS IoT.

Now that we have security for our thing and communication messages should be coming in. In IoT Core, choose TEST. Next, we will subscribe to the topic that we set in our IoT policy. It was called «sdk/test/Python» since the thing created is a virtual generated one just for tests, and we’ll click ‘Subscribe to topic.’

At the bottom, you can see the data coming in. The data shows the «Hello» Message and the sequence number. This confirms that we have communication.

In case you have a remotely connected thing instructed to provide more data of course you would get all the published data contained in the messages for the topic you subscribed.
Summary of what we have learned about AWS IoT so far:

- Registered a device
- Applied a security role to the device
- Customized the device's policy
- Downloaded the private and public certificates to the device
- Activated the certificates
- Established communication
- You can also view the data that the device is sending.

The objective in the next steps:

- is to visualize the data so that decision-makers at the ice cream company can review, understand, and finalize their decisions based on the received data.
- To get this result, we will need to route the data to AWS IoT Analytics and use Amazon QuickSight to visualize the data. We must introduce and use the AWS IoT rules engine to do that.

Now let’s have a look at AWS IoT rules engine
AWS IoT rules’ engine
Sensors produce incredible amounts of data:
- not all the data is always useful for any purpose,
- you can filter or direct data to different locations in the cloud.

The AWS IoT rules engine is one of the primary methods of filtering and directing communication from AWS IoT Core to other AWS services.

Rules define how your devices interact with the AWS services.
1) Data flows from devices (things) into the AWS IoT Core by defined MQTT topic streams.
2) Rules are analyzed and actions are performed based on the MQTT topic stream.
3) Rules define possible actions and forwarding of data to services, e.g. writing to Amazon DynamoDB database, use of AWS Lambda function to extract specific data, or saving data to Amazon Simple Storage Service (Amazon S3).

Before AWS IoT can perform these actions, you must grant it the appropriate IAM role and permissions to perform any required actions on your behalf.
When the actions are performed, you start paying for the AWS services you use.
Channels and Pipelines, Data Store and Data Sets

- **Channel**: A collection of data from an MQTT topic. Channels archive the raw, unprocessed messages coming from the Things before publishing the data to a pipeline.
- **Channel activity**: a transformation or action on the data coming in a channel
  - It defines Name of the Activity, channelName where the activity processes data from, and pointer to next activity in the pipeline (if any)
- **Pipeline**: a sequence of activities transforming data through the pipeline
  - It defines the input channel for the pipeline, and the output Data Store
- **Data Store**: a storage of data collected from a pipeline
- **Data Set**: a subset of Data Store extracted with a query.
Rules engine language:

- The rules engine uses SQL-like statements that are built within a structured development environment to filter and route MQTT messages.

The SQL rule listed in the example:

1) inserts all (*) messages sent to the `iot/test` topic
2) places them into the Amazon S3 bucket called `my-bucket`.

The SQL statement filters the messages, and the roleARN gives AWS IoT permission to write to the Amazon S3 bucket.
Let’s make a rule to direct a MQTT topic’s data «sdk/test/Python» to AWS IoT Analytics via a rule called «testAnalyze»:

We will create a rule «testAnalyze» with two associated actions, each one with a associated role and topic or channel.

1) Create a new rule (Act > Create a rule) and give it a name, e.g. «testAnalyze» and a description (what the rule does), and select the SQL version.

2) Create the SQL query SELECT (let you be guided by structured environment) so that the data is transformed appropriately.
   (e.g. SELECT sequence FROM 'sdk/test/Python' WHERE sequence > 5 )

3) Identify the data stream (MQTT topic specified as topic filter) «sdk/test/Python», this is equivalent to e.g. «SELECT * FROM sdk/test/Python» and

3) Put some conditions (if any) about when/how the rule is applied (e.g. sequence > 5).

4) Now add the action associated to the rule. You can select many of these predefined for the AWS services.
   e.g. select to «republish (transform) the messages (data) to an (different) AWS IoT topic». Now configure the action by giving the different target topic a name e.g. «test/analyze», and select the IAM role to give to AWS IoT the right to access the data.

5) If the IAM role for the rules engine is not created before we must create it with «Create a new role». We call the new role «testAnalyzeRole», then we list it, select it for the rule;
   Now select «Add Action» to conclude the configuration of the action, and finally «create rule» to finalize the creation of the rule for transforming the data. You can test the data is now flowing on both the two topics.

6) Let’s go back to the previous rule: choose «Act» and open the testAnalyze rule, and then add a second action «Send message to an IoT Analytics», and click on Configure. However, we do not have a channel to send messages and we must create it.

7) Create a channel with «Create a new resource» and call the channel «testAnalyzeChannel». This is the channel for data flowing between AWS IoT Core to AWS IoT Analytics. Decide how much to store the data (e.g. 1 week), which data to send, and then save the channel created.

8) Now we need a new role associated to the channel as well. Choose ‘Create new’ and call this role «testChannelAnalyticsRole», and finalize it with Create role. Now select «Add Action» to the rule to add this last action to the previous rule.

In conclusion, we have a rule «testAnalyze» with two associated actions, each one with a associated role and topic or channel.
AWS IoT Analytics
for presenting the data

(https://docs.aws.amazon.com/iotanalytics/latest/userguide/welcome.html)
AWS IoT Analytics: scope

AWS IoT Analytics is the tool that helps with formatting and analyzing the data flowing from devices.

✓ It automates the steps required to analyze data.
  – You **configure** the service to collect only the data you need from your devices,
  – You **apply** transformations to **process** the data,
  – You **enrich** the data with device-specific **metadata**, such as device type and location, before **storing** it.

✓ you can **analyze** your data by running queries using the built-in SQL query engine, or perform more **complex analytics** and **machine learning inference**.
AWS IoT Analytics defines specific entities.

- **Channel**: collects and archives raw, unprocessed message data before publishing this data to a pipeline.
- **Pipeline**: consumes messages from a channel and enables you to process and filter the messages before storing them in a data store.
- **Data Store**: this is not a database, but it is a scalable and queryable repository of your messages. You can have multiple data stores for messages that come from different devices or locations.
- **Data Set**: created to retrieve data from a data Store. To create a SQL dataset or a container dataset you make a query (SQL action) to the data store.
- **Data set contents**: these are the results collected and shown in the console.
AWS IoT Analytics: data processing summary

AWS IoT Analytics: basic data-oriented processing approach.

1) **Collect**: define an AWS IoT Analytics channel and select the specific data to collect, such as temperature sensor readings.
2) **Process**: configure AWS IoT Analytics pipelines to process your data. AWS IoT Analytics pipelines support transformations, such as Celsius to Fahrenheit conversion, conditional statements, message filtering, and message enrichment, using external data sources and AWS Lambda functions.
3) **Store**: data processed in the pipeline are stored in IoT-optimized data store for analysis.
4) **Analyze**: query the data store with built-in SQL query engine to answer specific business questions.
5) **Build**: build visualizations and dashboards to get business insights, e.g. using Amazon QuickSight.
Visualization of data

- Choose Services, Internet of Things, IoT Analytics
- Identify the channel created before «IoT Analytics» and select «create a pipeline», select attributes of data (all) and possibly transform and execute activities to enrich the data. (in this case we did all filtering and transformations in the rules applied before the pipeline).
- Create the data store «truckAnalyzeStore» for the data emerging by the pipeline

Situation: We have a channel coming in from AWS IoT Analytics. We have a pipeline that could target, transform, and even enrich the data, and then we have a data store set.

- Now create a Data Set «truckAnalyzeDataSet»: click on Analyze > Create a data set > Create SQL. The source is the truckAnalyzeStore that we just created.
- Define a SQL query, e.g. «SELECT * FROM truckAnalyzeStore» and initialize other parameters (if any) and «Create data set».
- Once the DataSet is created, execute the query on the data with Actions > Run now.
- Now enter Amazon QuickSight
- New Analysis > New Data Set > AWS IoT Analytics (or external CSV file here) and select the «truckAnalyzeDataSet», > Create data source > visualize
AWS IoT best practices
AWS IoT Best Practices

- For things, you should utilize granular device IDs and associate them with the collected and aggregated device data. This will provide overall flexibility and a better ability for you to scale horizontally.
- It is important that you have the IoT device create the timestamp when data is collected instead of when the device synchronizes with a collector or the cloud. This will avoid masking issues caused by latency and connectivity.
- Each device you create or onboard is given a unique certificate and key pair to enable fine-grained management, including certificate revocation.
- Devices must also support rotating and replacing certificates in order to ensure smooth operation as certificates expire.
- IoT policies should follow a strategy of least privilege for permissions. Publish/subscribe only the essential. As an example, the default policy grants access to publish and subscribe to all IoT topics, and that’s not good. That needs to be replaced with the specific topics that device will use.
- Make sure that ClientID and Thing Name also matches on the device connections, so there is no conflict. Choose a thing name and use it as MQTT client ID for registry and device shadow service (good organization with flexibility).
- Think about the frequency that your device will send data (impacts energy consumption, network bandwidth, and cost of the data sent by your device, which might be low-powered)
- Work backwards from your use case to determine the right frequency to sample your data, and make that parameter configurable.
- For analytics and visualization, when you are visualizing the data, the widget you use should reflect the KPIs that have been defined by your business questions and outcomes.