Comparison of AWS, Azure, and Oracle Device Models

W3C Web of Things Open Day
26.3.2018, Prague
Michael.Lagally@oracle.com

Outline

This presentation takes a focused view on the device concepts in the IoT cloud platforms of Amazon (AWS Device Shadows), Microsoft (Azure IoT Hub) and Oracle (IoT Cloud Service).

The device model for each cloud service is structured in the following way:

- High Level Concept
- Device Description
- Device Model
- Properties
- Actions
- Events
- Serialization format
- ~~Communication Protocols, Messages and Formats, API, Security~~

A complete overview of these cloud platforms is beyond the scope of this presentation − the strikeout sections are intentionally left out.

Terminology

Thing or Device

A physical or logical entity that is managed by an IoT Cloud Service

Device Description (aka. Thing Description)

A serialized representation of a Device

Device Model (aka. Device Templates or Thing Types)

A blueprint that describes the structure and interface of a group of devices

Properties, aka. Attributes or Resources

A part of the device that contains state information and may change over time

Events (aka. Messages or Alerts)

A notification mechanism between the Device and the IoT Cloud Service

MICROSOFT AZURE IOT HUB
DEVICE TWINS

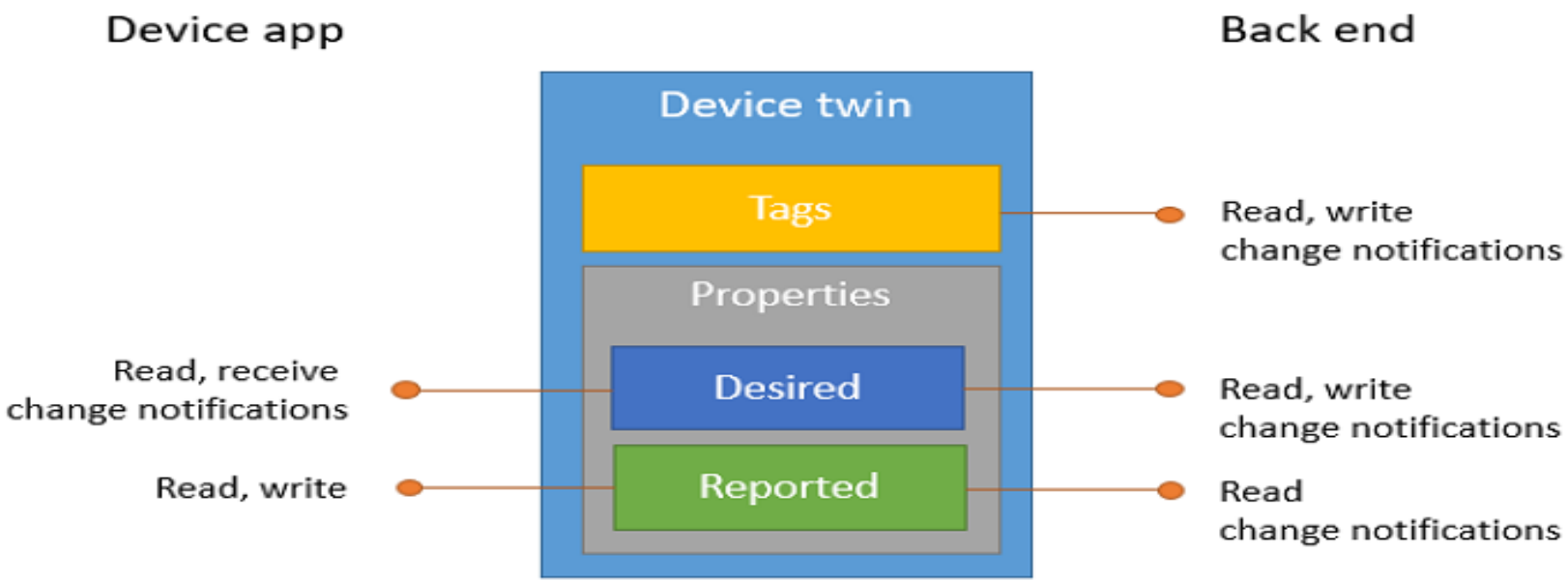
Microsoft Azure IoT Hub - Device Twins References

- The following is derived from public information available at:
- https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-node-node-getstarted
- https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-query-language
- https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/iot-hub/iot-hub-devguide-device-twins.md

Microsoft Azure IoT Hub - Device Twins
High Level Summary

- Microsoft's device twin is an abstraction of a device state using (desired and reported) properties and a set of tags, containing metadata values.
- Actions and events are not part of the model, but are handled by application code. Actions can contain multiple parameters and take objects as parameters.
- Messages are rather lightweight and the content can be selected by the application down to property level. The format of the messages is defined by applications only.
- The device twin model does not define a "template" or a mechanism to aggregate multiple devices into a combined device model.

Microsoft Azure IoT Hub
Device Twins



Source: Microsoft

Device twin

- A device twin is a JSON document that includes:
- **Tags**: A section where the solution back-end has access to. They are not visible to device apps.
- **Properties:** Used to synchronize device configuration or conditions.
- All property values can be of the following JSON types: boolean, number, string, object. Arrays are not allowed.

Properties

3 kinds of properties:

- **Desired properties**: Can be set by the solution back end and read by the device app. The app can also receive notifications of changes.
- **Reported properties:** The device app can set reported properties, and the solution back end can read and query them.
- **Device identity properties:** The root of the device twin JSON document contains the read-only properties from the corresponding device identity stored in the identity registry.

Actions

Actions do not correspond to a formal description in the JSON document, but are modeled via posting of a payload to a "method" endpoint. An action can take an arbitrary number of parameters.

Events

There is no dedicated event mechanism. Events can be modeled by using device-to-cloud messages, which are being sent under application control. The event payload format is not specified in the JSON document.

Serialization format


```
{
  "deviceId": "myDeviceId",
  "etag": "AAAAAAAAAAc=",
  "status": "enabled",
  "statusUpdateTime": "0001-01-01T00:00:00",
  "connectionState": "Disconnected",
  "lastActivityTime": "0001-01-01T00:00:00",
  "cloudToDeviceMessageCount": 0,
  "authenticationType": "sas",
  "x509Thumbprint": {
    "primaryThumbprint": null,
    "secondaryThumbprint": null
  },
  "version": 2,
  "tags": {
    "location": {
      "region": "US",
      "plant": "Redmond43"
    }
  },
  "properties": {
    "desired": {
      "telemetryConfig": {
        "configId": "db00ebf5-eeeb-42be-86a1-458cccb69e57",
        "sendFrequencyInSecs": 300
      },
      "$metadata": {
        ...
      },
      "$version": 4
    },
    "reported": {
      "connectivity": {
        "type": "cellular"
      },
      "telemetryConfig": {
        "configId": "db00ebf5-eeeb-42be-86a1-458cccb69e57",
        "sendFrequencyInSecs": 300,
        "status": "Success"
      },
      "$metadata": {
      ...
      },
      "$version": 7
    }
  }
}
```

AWS
DEVICE SHADOWS

References

- The following is derived from public information available at:
- https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html
- https://docs.aws.amazon.com/iot/latest/developerguide/aws-iot-how-it-works.html
- https://docs.aws.amazon.com/iot/latest/developerguide/iot-sdks.html
- https://docs.aws.amazon.com/iot/latest/apireference/API_Operations.html

High Level Concept

- AWS IoT **Device Shadows** enable Internet-connected devices to connect to the AWS Cloud and let applications in the cloud interact with Internet-connected devices.
- Devices report their state by publishing messages in JSON format on MQTT topics. Each MQTT topic has a hierarchical name that identifies the device whose state is being updated. When a message is published on an MQTT topic, the message is sent to the AWS IoT MQTT message broker, which is responsible for sending all messages published on an MQTT topic to all clients subscribed to that topic.

Thing – Device Model

AWS IoT provides a registry to manage *things*.

A thing is a representation of a specific device or logical entity: a physical device or sensor or a logical entity like an instance of an application.

Information about a thing is stored in the registry as JSON data. Here is an example thing:

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

Things are identified by a name. Things can have attributes, which are name-value pairs that can be used to store information about the thing, such as its serial number or manufacturer.

Thing types (Templates)

Thing types

Thing types store description and configuration information that is common for all things associated with the same thing type. This simplifies the management of all things.

Thing groups

Allow to manage several things at once. Groups can also contain groups — you can build a hierarchy of groups.

Attributes/Resources/Properties

- https://docs.aws.amazon.com/iot/latest/apireference/API_ThingAttribute.html
- Each attribute value in JSON objects can have a maximum length of 800 bytes.
- The specification is silent about the supported attribute types, the examples contain only simple types.
- However the rules engine is supporting all JSON types including arrays and objects.

Actions

- Actions do not correspond to a formal description in the JSON document, but are modeled via posting of a payload to a "method" endpoint.
- An action can take an arbitrary number of parameters.

Events

- There is no dedicated event mechanism. Events can be modeled by using device-to-cloud messages, which are sent under application control.
- They are not part of the JSON document.

Serialization format


```
{
    "deviceId" :  "myDeviceId" ,
    "etag" :  "AAAAAAAAAAc=" ,
    "status" :  "enabled" ,
    "statusUpdateTime" :  "0001-01-01T00:00:00" ,
    "connectionState" :  "Disconnected" ,
    "lastActivityTime" :  "0001-01-01T00:00:00" ,
    "cloudToDeviceMessageCount" :  0 ,
    "authenticationType" :  "sas" ,
    "x509Thumbprint" : {
        "primaryThumbprint" :  null ,
        "secondaryThumbprint" :  null
    },
    "version" :  2 ,
    "tags" : {
        "location" : {
            "region" :  "US" ,
            "plant" :  "Redmond43"
        }
    },
    "properties" : {
        "desired" : {
            "telemetryConfig" : {
                "configId" :  "db00ebf5-eeeb-42be-86a1-458cccb69e57" ,
                "sendFrequencyInSecs" :  300
            },
            "$metadata" : {
                ...
            },
            "$version" :  4
        },
        "reported" : {
            "connectivity" : {
                "type" :  "cellular"
            },
            "telemetryConfig" : {
                "configId" :  "db00ebf5-eeeb-42be-86a1-458cccb69e57" ,
                "sendFrequencyInSecs" :  300 ,
                "status" :  "Success"
            },
            "$metadata" : {
            ...
            },
            "$version" :  7
        }
    }
}
```

ORACLE
DEVICE MODEL

References

The following is derived from public information available at:

- https://docs.oracle.com/en/cloud/paas/iot-cloud/iotgs/toc.htm
- https://docs.oracle.com/en/cloud/paas/iot-cloud/iotrq/index.html
- http://www.oracle.com/technetwork/indexes/downloads/iot-client-libraries-2705514.html
- https://docs.oracle.com/en/cloud/paas/iot-cloud/develop/developing-device-software-using-client-software-libraries1.html
- http://www.oracle.com/webfolder/technetwork/tutorials/obe/cloud/iot/IoT%20Quick%20Start%20JavaScript/IoTQuickStartJavaScript.html

High Level Summary

- The Oracle IoT CloudService platform manages devices and corresponding messages and events. At the core of the service is the device model.
- Oracle's **device model** is a blueprint for defining devices. It can be used to create device instances and simulations without having a physical device available.
- This can significantly help to validate the device model, before actually building a device.
- Separating the blueprint from the device instance is useful to build aggregated devices, which incorporate multiple device models.
- A **device instance** is created by registering a device, which implements one or more device models, with the IoT cloud service.
- Device models and device descriptions are managed by the IoT Cloud Service. They can be exported and imported in a plain JSON format, which is similar to the "WoT TD" format.
- A simple WoT-TD - DM converter is available.

Device Model

- A device model is a common blueprint, that can be applied to multiple devices.
- It contains metadata, attributes, actions, formats and links.
- Device models are uniquely identifiable via “urn”
- A device can implement multiple device models simultaneously.

Attributes

- Attributes can have the following types: number, integer, number with range, integer with range, string, boolean, date&time, uri.
- Complex attributes (arrays, objects) are intentionally not included, since the integration with existing back-end services (e.g. Analytics) requires simple types.
- Attributes can be read only.

Actions

- Actions are triggered by sending a payload to the corresponding endpoint.
- Actions can contain only a single parameter.
- The reason for this apparent limitation is to enable very lightweight implementations.
- If complex parameters were needed they could be encapsulated in a JSON-string.

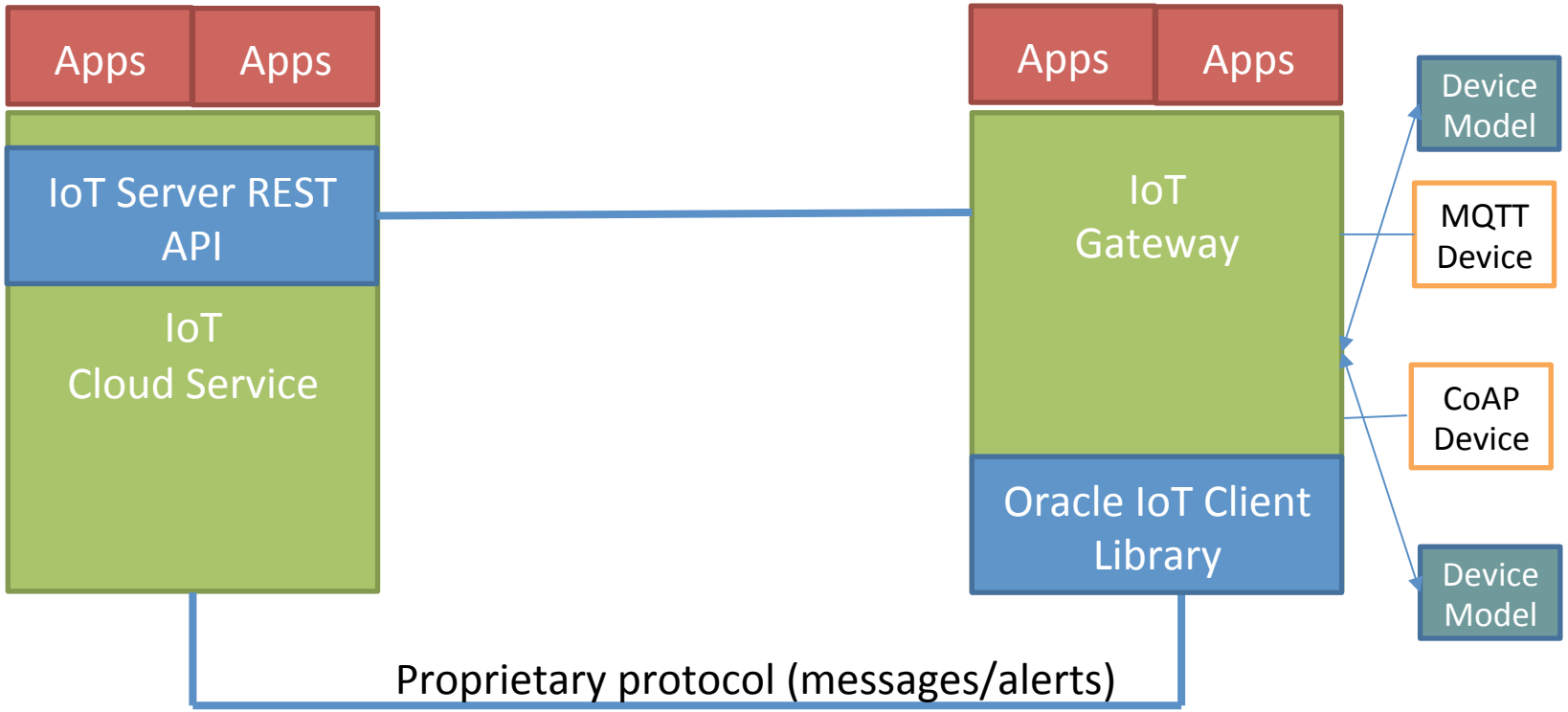
Formats

- Formats define the payload structure for messages.
- Formats are uniquely identifiable via “urn”.
- Formats are either regular “data” messages or “alert” messages that are sent under application control.
- The payload of the alert message is defined in the “formats” section.

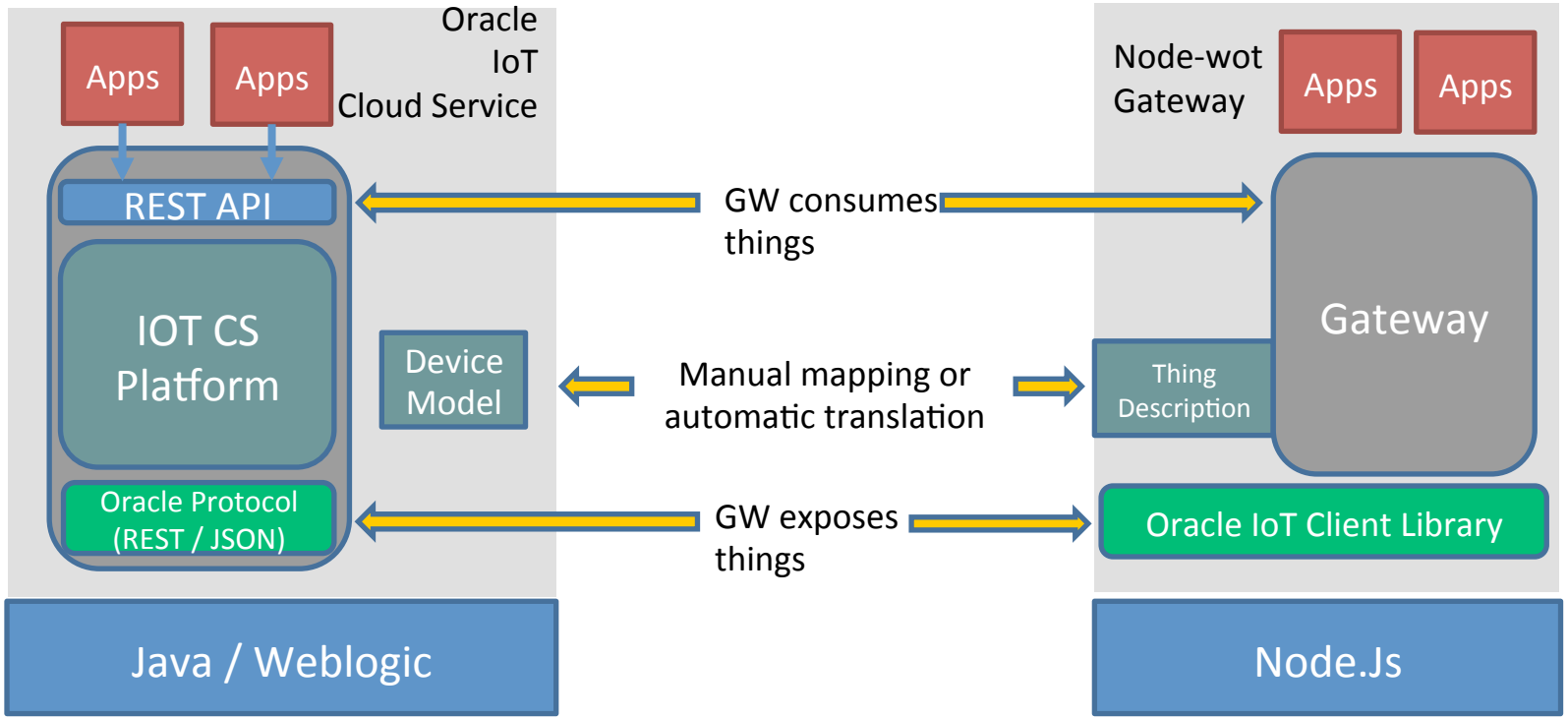
Device model serialization format

```
{
    "urn": "urn:com:oracle:iot:device:humidity_sensor",
    "name": "Humidity Sensor",
    "description": "Device model for sensor that measures humidity.",
    "attributes": [
        {
            "name": "humidity",
            "description": "Measures humidity between 0% and 100%",
            "type": "INTEGER",
            "range": "0,100"
        },
        {
            "name": "maxThreshold",
            "description": "Maximum humidity threshold",
            "type": "INTEGER",
            "range": "60,100",
            "writable": true,
            "defaultValue":80
        }
    ],
 "actions": [
    {
      "alias": "",
      "description": "",
      "name": “normalizeThreshold"
    }
 ],
     "formats": [
        {
            "urn":
"urn:com:oracle:iot:device:humidity_sensor:too_humid",
            "name": "tooHumidAlert",
            "description": "Sample alert when humidity reaches the
maximum humidity threshold",
            "type": "ALERT",
            "value": {
                "fields": [
                    {
                        "name": "humidity",
                        "type": "INTEGER",
                        "optional": false
                    }
                ]
            }
        }
    ]
}
```

Typical IoT CS deployment scenario



Node-wot + Oracle IoT Cloud Service Integration



3 Integration points

GW consumes devices exposed by IoT CS (from other vendors)
- Requires the GW to use the REST API of the IoT cloud service

IoT CS consumes devices connected to node-wot gateway
- Requires integration of the Oracle IoT Client library into the GW

Mapping of thing description to device model
- This mapping could be done manually in the first step
- No implementation effort

SUMMARY

Summary + Conclusion

- The 3 IoT server platforms share many commonalities between their device models. However there are significant differences.
- AWS and Azure only define a **data model**, no actions and events.
- None of the serialization formats are **interchangeable**.
- None of the formats defines a **protocol binding**.
- A **device manufacturer**, who wants to address these platforms, has to create code for 3 different environments. This may exceed the hardware capabilities on very small devices.

A **unified device model** will simplify the integration tasks across different platforms and will accelerate IoT market adoption.