WHITE PAPER

A Reference Architecture For The Internet of Things

By Paul Fremantle
Co-Founder & CTO, WSO2

1. Introduction

IoT is an umbrella term that includes multiple different categories:

- Wireless sensor/actuator networks
- Internet-connected wearables
- Low power embedded systems
- RFID enabled tracking
- Use of mobile phones to interact with the real world (e.g. sensing)
- Devices that connect via Bluetooth-enabled mobile phones to the Internet
- Smart homes
- Connected cars

And many more

The result is that no single architecture will suit all these areas and the requirements each area brings. However, a modular scalable architecture that supports adding or subtracting capabilities, as well as supporting many requirements across a wide variety of these use cases is inherently useful and valuable. It provides a starting point for architects looking to create IoT solutions as well as a strong basis for further development. This paper proposes such a reference architecture. The reference architecture must cover multiple aspects including the cloud or server-side architecture that allows us to monitor, manage, interact with and process the data from the IoT devices; the networking model to communicate with the devices; and the agents and code on the devices themselves, as well as the requirements on what sort of device can support this reference architecture.

The reference architecture we propose is inherently vendor neutral and not specific to a set of technologies, although it is highly influenced by best-of-breed open-source projects and technology. In addition, we provide a mapping of this reference architecture to WSO2's open source products and projects and we have demonstrated an implementation of this reference architecture on the WSO2 platform.

We also explore areas where this reference architecture can be extended further and as well as areas where we expect to see further work.

2. The Internet of Things – An Overview

The Internet of Things, or IoT, refers to the set of devices and systems that interconnect real-world sensors and actuators to the Internet. This includes many different systems, including

- Internet connected cars
- wearable devices including health and fitness monitoring devices, watches, and even human implanted devices;
- smart meters and smart objects;
- home automation systems and lighting controls;
- smartphones that are increasingly being used to measure the world around them; and
- wireless sensor networks that measure weather, flood defenses, tides and more.

The growth of the number and variety of devices that are collecting data is incredibly rapid.A study by Cisco[1] estimates that the number of Internet-connected devices overtook the human population in 2010, and that there will be 50 billion Internet-connected devices by 2020.

There are of course two key aspects to the IoT: the devices themselves and the server-side architecture that supports them. In fact there is often a third-category as well; in many cases there may be a low power gateway that performs aggregation, event processing, bridging, etc. that might sit between the device and the wider Internet.

In both cases, the devices probably have intermittent connections based on factors such as GPRS connectivity, battery discharging, radio interference, or simply being switched off.

There are effectively three classes of devices:

- The smallest devices have embedded 8-bit System-On-Chip (SOC) controllers. A good example of this is the open source hardware platform Arduino: e.g the Arduino Uno platform and other 8-bit Arduinos. These typically have no operating system.
- The next level up are the systems based on Atheros and ARM chips that have a very limited 32-bit architecture. These often include small home routers and derivatives of those devices. Commonly, these run a cut-down or embedded Linux platform, such as OpenWRT, or dedicated embedded operating systems. In some cases they may not use an OS, e.g. the Arduino Zero, or the Arduino Yun.
- The most capable IoT platforms are full 32-bit or 64-bit computing platforms. These systems, such as the Raspberry Pi or the BeagleBone, may run a full Linux OS or another suitable Operating System, such as Android. In many cases these are either mobile phones or based on mobile-phone technology. These devices may also act as gateways or bridges for smaller devices, e.g. if a wearable connects via Bluetooth Low Energy to a mobile phone or Raspberry Pi, which then bridges that onto the wider Internet.

The communication between devices and the Internet or to a gateway includes many different models:

- Direct Ethernet or Wi-Fi connectivity using TCP or UDP (we will look at protocols for this later)
- Bluetooth Low Energy
- Near Field Communication (NFC)
- Zigbee or other mesh radio networks
- SRF and point-to-point radio links
- UART or serial lines
- SPI or I2C wired buses

Figure 1 below illustrates the two major modes of connectivity.

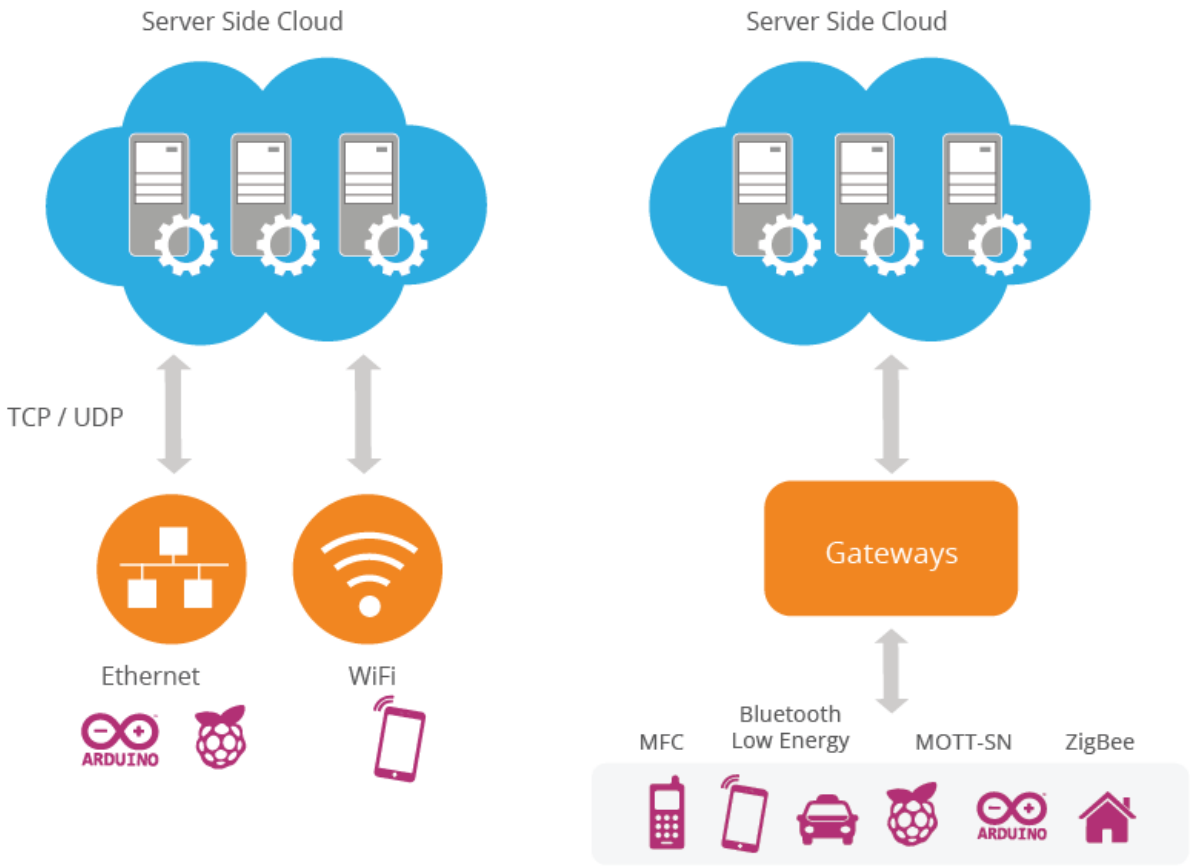

Figure 01

This section has provided a short overview of IoT devices and systems. It is not designed to be comprehensive or even extensive, but simply to provide enough background to support the discussion of requirements and capabilities below. There are many further resources available, which are too numerous to list. However, we can point readers to an academic survey, which is available here[2].

2.1 Is There Value in A Reference Architecture for the IoT?

There are several reasons why a reference architecture for IoT is a good thing:

- IoT devices are inherently connected – we need a way of interacting with them, often with firewalls, network address translation (NAT) and other obstacles in the way.
- There are billions of these devices already and the number is growing quickly; we need an architecture for scalability. In addition, these devices are typically interacting 24x7, so we need a highly-available (HA) approach that supports deployment across data centers to allow disaster recovery (DR).
- The devices may not have UIs and certainly are designed to be “everyday” usage, so we need to support automatic and managed updates, as well as being able to remotely manage these devices.
- IoT devices are very commonly used for collecting and analyzing personal data. A model for managing the identity and access control for IoT devices and the data they publish and consume is a key requirement.

Our aim is to provide an architecture that supports integration between systems and devices.

In the next section, we will dig into these requirements deeper and outline the specific requirements we are looking for in a range of categories.

3. Requirements for a Reference Architecture

There are some specific requirements for IoT that are unique to IoT devices and the environments that support them, e.g. many requirements emerge from the limited formfactors and power available to IoT devices. Other requirements come from the way in which IoT devices are manufactured and used. The approaches are much more like traditional consumer product designs than existing Internet approaches. Of course there are a number of existing best practices for the server-side and Internet connectivity that need to be remembered and factored in.

We can summarize the overall requirements into some key categories:

- Connectivity and communications
- Device management
- Data collection, analysis, and actuation
- Scalability
- Security
- HA
- Predictive analysis
- Integration

3.1 Connectivity and Communications

Existing protocols, such as HTTP, have a very important place for many devices. Even an 8-bit controller can create simple GET and POST requests and HTTP provides an important unified (and uniform) connectivity. However, the overhead of HTTP and some other traditional Internet protocols can be an issue for two main reasons. Firstly, the memory size of the program can be an issue on small devices. However, the bigger issue is the power requirements. In order to meet these requirements, we need a simple, small and binary protocol. We will look at this in more detail below. We also require the ability to cross firewalls.

In addition, there are devices that connect directly and those that connect via gateways. The devices that connect via a gateway potentially require two protocols: one to connect to the gateway, and then another from the gateway to the cloud.

Finally, there is obviously a requirement for our architecture to support transport and protocol bridging, e.g. we may wish to offer a binary protocol to the device, but allow an HTTP-based API to control the device that we expose to third parties.

3.2 Device Management

While many IoT devices are not actively managed, this is not necessarily ideal. We have seen active management of PCs, mobile phones, and other devices become increasingly important, and the same trajectory is both likely and desirable for IoT devices. What are the requirements for IoT device management? The following list covers some widely desirable requirements:

- The ability to disconnect a rogue or stolen device
- The ability to update the software on a device
- Updating security credentials

- Remotely enabling or disabling certain hardware capabilities
- Locating a lost device
- Wiping secure data from a stolen device
- Remotely re-configuring Wi-Fi, GPRS, or network parameters

The list is not exhaustive, and conversely covers aspects that may not be required or possible for certain devices.

3.3 Data Collection, Analysis, and Actuation

A few IoT devices have some form of UI, but in general IoT devices are focused on offering one or more sensors, one or more actuators, or a combination of both. The requirements of the system are that we can collect data from very large numbers of devices, store it, analyze it, and then act upon it.

The reference architecture is designed to manage very large numbers of devices. If these devices are creating constant streams of data, then this creates a significant amount of data. The requirement is for a highly scalable storage system, which can handle diverse data and high volumes.

The action may happen in near real time, so there is a strong requirement for real-time analytics. In addition, the device needs to be able to analyze and act on data. In some cases this will be simple, embedded logic. On more powerful devices we can also utilize more powerful engines for event processing and action.

3.4 Scalability

Any server-side architecture would ideally be highly scalable, and be able to support millions of devices all constantly sending, receiving, and acting on data. However, many “high-scalability architectures” have come with an equally high price – both in hardware, software, and in complexity. An important requirement for this architecture is to support scaling from a small deployment to a very large number of devices. Elastic scalability and the ability to deploy in a cloud infrastructure are essential. The ability to scale the serverside out on small cheap servers is an important requirement to make this an affordable architecture for small deployments as well as large ones.

3.5 Security

Security is one of the most important aspects for IoT. IoT devices are often collecting highly personal data, and by their nature are bringing the real world onto the Internet (and viceversa). This brings three categories of risks:

- Risks that are inherent in any Internet system, but that product/IoT designers may not be aware of
- Specific risks that are unique to IoT devices
- Safety to ensure no harm is caused by, for instance, misusing actuators

The first category includes simple things such as locking down open ports on devices (like the Internet-attached fridge that had an unsecured SMTP server and was being used to send spam).

The second category includes issues specifically related to IoT hardware, e.g. the device may have its secure information read. For example, many IoT devices are too small to support proper asymmetric encryption. Another specific example is the ability for someone to attack the hardware to understand security. Another example - university security researchers who famously reverse-engineered and broke the Mifare Classic RFID card solution[3]. These sort of reverse engineering attacks are an issue compared with pure web solutions where there is often no

available code to attack (i.e. completely server-side implementation).

Two very important specific issues for IoT security are the concerns about identity and access management. Identity is an issue where there are often poor practices implemented. For example, the use of clear text/ Base64 encoded user IDs/passwords with devices and machine-to-machine (M2M) is a common mistake. Ideally these should be replaced with managed tokens such as those provided by OAuth/OAuth2[4].

Another common issue is to hard-code access management rules into either client- or server-side code. A much more flexible and powerful approach is to utilize models such as "Attribute Based Access Control" and "Policy Based Access Control". The most well known of these approaches is that provided by the XACML standard[5]. Such approaches remove access control decisions from hard-coded logic and externalize them into policies, which enabled the following:

- More powerful and appropriate decisions;
- Can potentially be based on contexts such as location, or which network is being used, or the time of day;
- Access control can be analyzed and audited; and
- Policies can be updated and changed, even dynamically, without recoding or modifying devices.

Our security requirements therefore should support

- Encryption on devices that are powerful enough;
- A modern identity model based on tokens and not userids/passwords;
- The management of keys and tokens as smoothly/remotely as possible; and
- Policy-based and user-managed access control for the system based on XACML.

This concludes the set of requirements that we have identified for the reference architecture. Of course, any given architecture may add further requirements. Some of those may already be met by the architecture, and some may require further components to be added. However, our design is for a modular architecture that supports extensions, which copes with this demand.

In the next section we introduce the architecture and approach.

4. The Architecture

The reference architecture consists of a set of components. Layers can be realized by means of specific technologies, and we will discuss options for realizing each component. There are also some cross-cutting/vertical layers such as access/identity management.

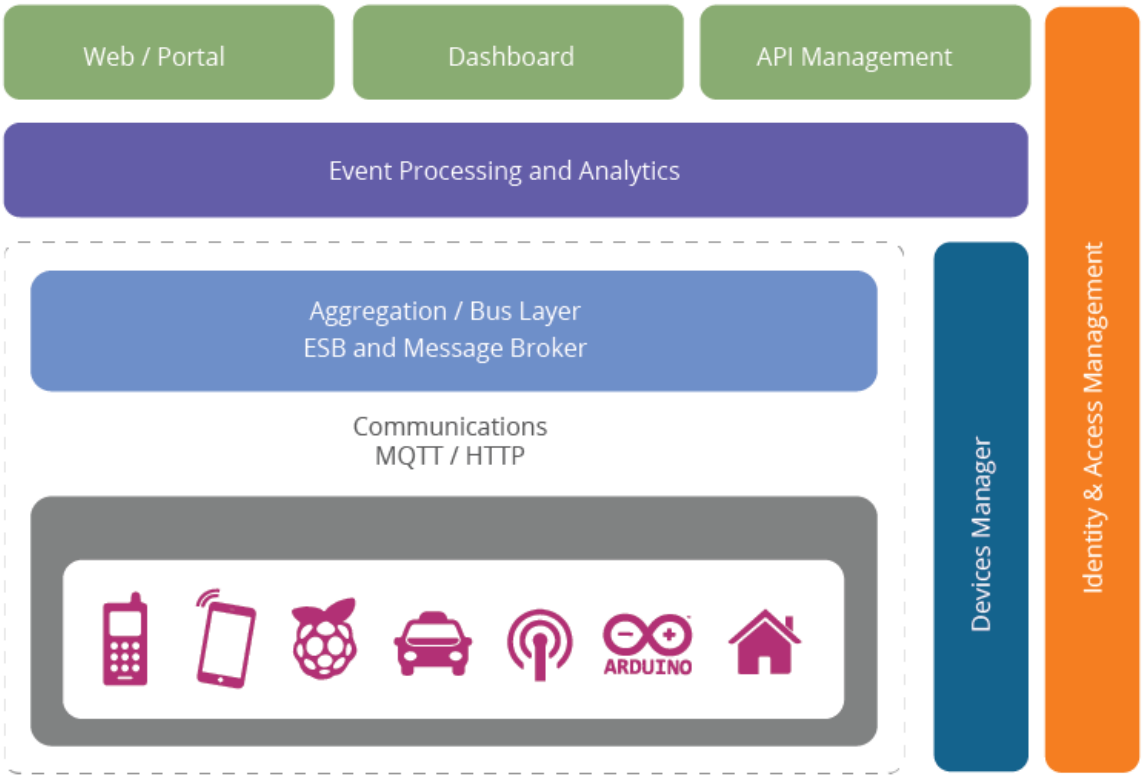

Figure 02: Reference architecture for IoT

The layers are

- Client/external communications - Web/Portal, Dashboard, APIs
- Event processing and analytics (including data storage)
- Aggregation/bus layer – ESB and message broker
- Relevant transports - MQTT/HTTP/XMPP/CoAP/AMQP, etc.
- Devices

The cross-cutting layers are

- Device manager
- Identity and access management

4.1 The Device Layer

The bottom layer of the architecture is the device layer. Devices can be of various types, but in order to be considered as IoT devices, they must have some communications that either indirectly or directly attaches to the Internet. Examples of direct connections are

- Arduino with Arduino Ethernet connection
- Arduino Yun with a Wi-Fi connection
- Raspberry Pi connected via Ethernet or Wi-Fi
- Intel Galileo connected via Ethernet or Wi-Fi Examples of indirectly connected device include
- ZigBee devices connected via a ZigBee gateway
- Bluetooth or Bluetooth Low Energy devices connecting via a mobile phone

- Devices communicating via low power radios to a Raspberry Pi

There are many more such examples of each type.

Each device typically needs an identity. The identity may be one of the following:

- A unique identifier (UUID) burnt into the device (typically part of the System-on-Chip, or provided by a secondary chip)
- A UUID provided by the radio subsystem (e.g. Bluetooth identifier, Wi-Fi MAC address)
- An OAuth2 Refresh/Bearer Token (this may be in addition to one of the above)
- An identifier stored in nonvolatile memory such as EEPROM

For the reference architecture we recommend that every device has a UUID (preferably an unchangeable ID provided by the core hardware) as well as an OAuth2 Refresh and Bearer token stored in EEPROM.

The specification is based on HTTP; however, (as we will discuss in the communications section) the reference architecture also supports these flows over MQTT.

4.2 The Communications Layer

The communication layer supports the connectivity of the devices. There are multiple potential protocols for communication between the devices and the cloud. The most wellknown three potential protocols are

- HTTP/HTTPS (and RESTful approaches on those)
- MQTT 3.1/3.1.1
- Constrained application protocol (CoAP)

Let's take a quick look at each of these protocols in turn.

HTTP is well known, and there are many libraries that support it. Because it is a simple textbased protocol, many small devices such as 8-bit controllers can only partially support the protocol – for example enough code to POST or GET a resource. The larger 32-bit based devices can utilize full HTTP client libraries that properly implement the whole protocol.

There are several protocols optimized for IoT use. The two best known are MQTT[6] and CoAP[7]. MQTT was invented in 1999 to solve issues in embedded systems and SCADA. It has been through some iterations and the current version (3.1.1) is undergoing standardization in the OASIS MQTT Technical Committee[8]. MQTT is a publish-subscribe messaging system based on a broker model. The protocol has a very small overhead (as little as 2 bytes per message), and was designed to support lossy and intermittently connected networks. MQTT was designed to flow over TCP. In addition there is an associated specification designed for ZigBee-style networks called MQTT-SN (Sensor Networks).

CoAP is a protocol from the IETF that is designed to provide a RESTful application protocol modeled on HTTP semantics, but with a much smaller footprint and a binary rather than a text-based approach. CoAP is a more traditional client-server approach rather than a brokered approach. CoAP is designed to be used over UDP.

For the reference architecture we have opted to select MQTT as the preferred device communication protocol, with HTTP as an alternative option.

The reasons to select MQTT and not CoAP at this stage are

- Better adoption and wider library support for MQTT;
- Simplified bridging into existing event collection and event processing systems; and
- Simpler connectivity over firewalls and NAT networks

However, both protocols have specific strengths (and weaknesses) and so there will be some situations where CoAP may be preferable and could be swapped in.

In order to support MQTT we need to have an MQTT broker in the architecture as well as device libraries. We will discuss this with regard to security and scalability later.

One important aspect with IoT devices is not just for the device to send data to the cloud/ server, but also the reverse. This is one of the benefits of the MQTT specification: because it is a brokered model, clients connect an outbound connection to the broker, whether or not the device is acting as a publisher or subscriber. This usually avoids firewall problems because this approach works even behind firewalls or via NAT.

In the case where the main communication is based on HTTP, the traditional approach for sending data to the device would be to use HTTP Polling. This is very inefficient and costly, both in terms of network traffic as well as power requirements. The modern replacement for this is the WebSocket protocol[9] that allows an HTTP connection to be upgraded into a full two-way connection. This then acts as a socket channel (similar to a pure TCP channel) between the server and client. Once that has been established, it is up to the system to choose an ongoing protocol to tunnel over the connection.

For the reference architecture we once again recommend using MQTT as a protocol with WebSockets. In some cases, MQTT over WebSockets will be the only protocol. This is because it is even more firewall-friendly than the base MQTT specification as well as supporting pure browser/JavaScript clients using the same protocol.

Note that while there is some support for WebSockets on small controllers, such as Arduino, the combination of network code, HTTP and WebSockets would utilize most of the available code space on a typical Arduino 8-bit device. Therefore, we only recommend the use of WebSockets on the larger 32-bit devices.

4.3 The Aggregation/Bus Layer

An important layer of the architecture is the layer that aggregates and brokers communications. This is an important layer for three reasons:

1. The ability to support an HTTP server and/or an MQTT broker to talk to the devices;
2. The ability to aggregate and combine communications from different devices and to route communications to a specific device (possibly via a gateway)
3. The ability to bridge and transform between different protocols, e.g. to offer HTTPbased APIs that are mediated into an MQTT message going to the device.

The aggregation/bus layer provides these capabilities as well as adapting into legacy protocols. The bus layer may also provide some simple correlation and mapping from different correlation models (e.g. mapping a device ID into an owner's ID or vice-versa).

Finally the aggregation/bus layer needs to perform two key security roles. It must be able to act as an OAuth2 Resource Server (validating Bearer Tokens and associated resource access scopes). It must also be able to act as a policy enforcement point (PEP) for policy-based access. In this model, the bus makes requests to the identity and access management layer to validate access requests. The identity and access management layer acts as a policy

decision point (PDP) in this process. The bus layer then implements the results of these calls to the PDP to either allow or disallow resource access.

4.4 The Event Processing and Analytics Layer

This layer takes the events from the bus and provides the ability to process and act upon these events. A core capability here is the requirement to store the data into a database. This may happen in three forms. The traditional model here would be to write a serverside application, e.g. this could be a JAX-RS application backed by a database. However, there are many approaches where we can support more agile approaches. The first of these is to use a big data analytics platform. This is a cloud-scalable platform that supports technologies such as Apache Hadoop to provide highly scalable mapreduce analytics on the data coming from the devices. The second approach is to support complex event processing to initiate near real-time activities and actions based on data from the devices and from the rest of the system.

Our recommended approach in this space is to use the following approaches:

- Highly scalable, column-based data storage for storing events
- Map-reduce for long-running batch-oriented processing of data
- Complex event processing for fast in-memory processing and near real-time reaction and autonomic actions based on the data and activity of devices and other systems
- In addition, this layer may support traditional application processing platforms, such as Java Beans, JAX-RS logic, message-driven beans, or alternatives, such as node.js, PHP, Ruby or Python.

4.5 Client/External Communications Layer

The reference architecture needs to provide a way for these devices to communicate outside of the device-oriented system. This includes three main approaches. Firstly, we need the ability to create web-based front-ends and portals that interact with devices and with the event-processing layer. Secondly, we need the ability to create dashboards that offer views into analytics and event processing. Finally, we need to be able to interact with systems outside this network using machine-to-machine communications (APIs). These APIs need to be managed and controlled and this happens in an API management system.

The recommended approach to building the web front end is to utilize a modular front-end architecture, such as a portal, which allows simple fast composition of useful UIs. Of course the architecture also supports existing Web server-side technology, such as Java Servlets/ JSP, PHP, Python, Ruby, etc. Our recommended approach is based on the Java framework and the most popular Java-based web server, Apache Tomcat.

The dashboard is a re-usable system focused on creating graphs and other visualizations of data coming from the devices and the event processing layer.

The API management layer provides three main functions:

- The first is that it provides a developer-focused portal (as opposed to the userfocused portal previously mentioned), where developers can find, explore, and subscribe to APIs from the system. There is also support for publishers to create, version, and manage the available and published APIs;
- The second is a gateway that manages access to the APIs, performing access control checks (for external requests) as well as throttling usage based on policies. It also performs routing and load-balancing;
- The final aspect is that the gateway publishes data into the analytics layer where it is stored as well as

processed to provide insights into how the APIs are used.

4.6 Device Management

Device management (DM) is handled by two components. A server-side system (the device manager) communicates with devices via various protocols and provides both individual and bulk control of devices. It also remotely manages software and applications deployed on the device. It can lock and/or wipe the device if necessary. The device manager works in conjunction with the device management agents. There are multiple different agents for different platforms and device types.

The device manager also needs to maintain the list of device identities and map these into owners. It must also work with the identity and access management layer to manage access controls over devices (e.g. who else can manage the device apart from the owner, how much control does the owner have vs. the administrator, etc.)

There are three levels of device: non-managed, semi-managed and fully managed (NM, SM,FM).

Fully managed devices are those that run a full DM agent. A full DM agent supports:

- Managing the software on the device
- Enabling/disabling features of the device (e.g. camera, hardware, etc.)
- Management of security controls and identifiers
- Monitoring the availability of the device
- Maintaining a record of the device's location if available
- Locking or wiping the device remotely if the device is compromised, etc.

Non-managed devices can communicate with the rest of the network, but have no agent involved. These may include 8-bit devices where the constraints are too small to support the agent. The device manager may still maintain information on the availability and location of the device if this is available.

Semi-managed devices are those that implement some parts of the DM (e.g. feature control, but not software management).

4.7 Identity and Access Management

The final layer is the identity and access management layer. This layer needs to provide the following services:

- OAuth2 token issuing and validation
- Other identity services including SAML2 SSO and OpenID Connect support for identifying inbound requests from the Web layer
- XACML PDP
- Directory of users (e.g. LDAP)
- Policy management for access control (policy control point)

The identity layer may of course have other requirements specific to the other identity and access management for a given instantiation of the reference architecture. In this section we have outlined the major components of the reference architecture as well as specific decisions we have taken around technologies. These decisions are motivated by the specific requirements of IoT architectures as well as best practices for building agile, evolvable, scalable Internet architectures. Of course there are other options, but this reference architecture utilizes proven

approaches that are known to be successful in real-life IoT projects we have worked on.

5. Mapping to the WSO2 Platform

A reference architecture is useful as-is. However, it is even more useful if there is a real instantiation. In this section we provide a mapping into products and capabilities of the WSO2 platform to show how the reference architecture can be implemented.

The WSO2 platform is a completely modular, open-source enterprise platform that provides all the capabilities needed for the server-side of this architecture. In addition, we also provide some reference components for the device layer – it is an intractable problem to provide components for all possible devices, but we do provide either sample code and/or supported code for certain popular device types.

An important aspect of the WSO2 platform is that it is inherently multi-tenant. This means that it can support multiple organizations on a single deployment with isolation between organizations (tenants). This is a key capability for deploying this reference architecture as a software-as-a-service (SaaS) offering. It is also used by some organizations on-premise to support different divisions or departments within a group.

The WSO2 platform supports deployment on three different targets:

1. Traditional on-premise servers including Linux, Windows, Solaris, and AIX
2. Public cloud deployment including Amazon EC2, Microsoft Azure, and Google Compute Engine
3. Hybrid or private cloud deployment on platforms including OpenStack, Suse Cloud, Eucalyptus, Amazon Virtual Private Cloud, and Apache Stratos.

The WSO2 platform is based on a technology called WSO2 Carbon, which is in turn based on OSGi. Each product in the platform shares the same kernel based on Carbon. In addition, each product is made from features that are composed to provide the required functionality. Features can be added and subtracted as needed. All the products work together using standard interoperable protocols, such as HTTP, MQTT and AMQP[10]. All the WSO2 products are available under the Apache Software License v2.0 which is a businessfriendly, non-viral Open Source License[11]. Figure 3 shows the IoT reference architecture layered with the corresponding WSO2 product capabilities.

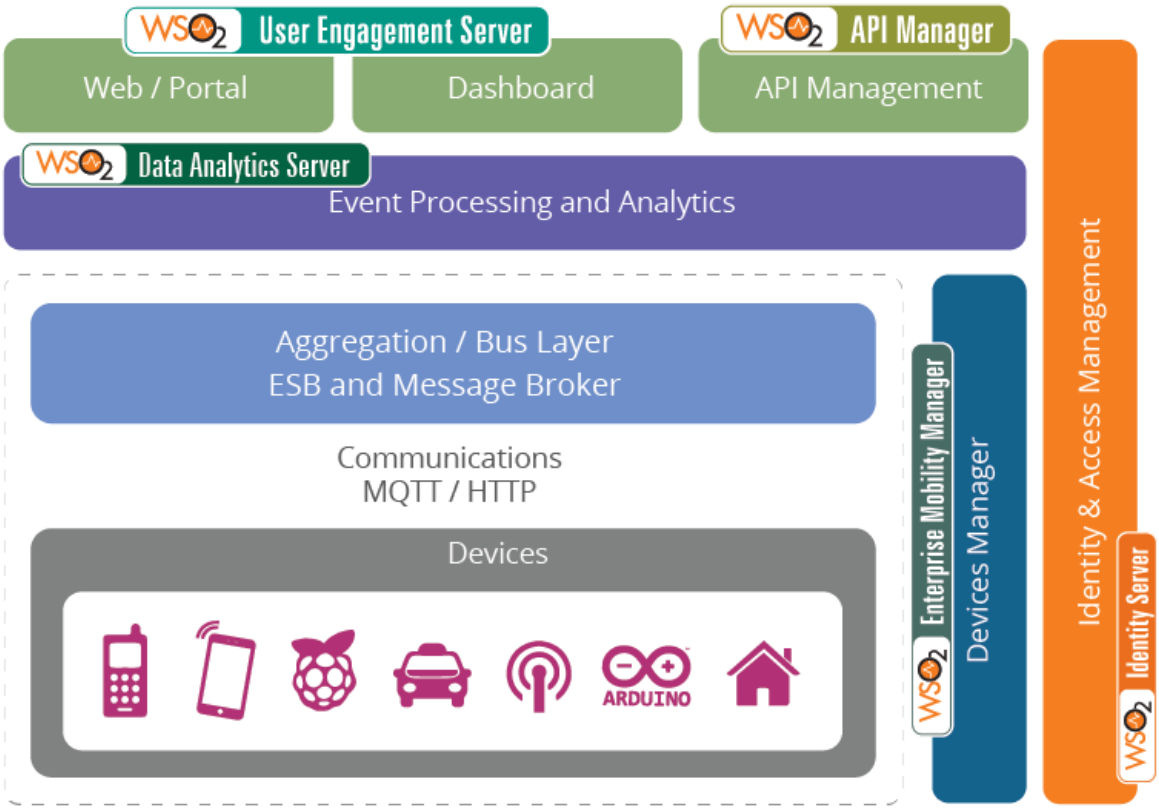

Figure 03: IoT reference architecture mapped to WSO2 platform components

The Device Layer

We support any device. We have a reference device management capability on any Linuxbased or Android-based device, which can be ported to other 32-bit platforms.

WSO2 also can help with MQTT client code for many device platforms ranging from Arduino to Android.

The Aggregation/Bus Layer

We provide two core products that implement this layer:

- WSO2 Enterprise Service Bus (ESB), which provides HTTP, MQTT, AMQP and other protocol support, protocol mediation and bridging, data transformation, OAuth2 Resource Server support, XACML Policy Enforcement Point (PEP) support and many other capabilities. WSO2 ESB is highly scalable providing linear scalability and elastic scalability. In one deployment it handles more than 2bn requests/day. Please note that WSO2 ESB does not currently support WebSockets but this is on the roadmap.
- WSO2 Message Broker (MB), which provides the ability to act as an MQTT broker. WSO2 MB also provides AMQP capabilities and can provide both persistent and nonpersistent messaging. WSO2 MB is highly scalable and supports elastic scalability. Please note that the WSO2 MB MQTT support is currently in beta.

The Analytics and Event Processing Layer

WSO2 offers a complete platform for data analytics with WSO2 Data Analytics Server (WSO2 DAS, available in Q4 2015), an industry first that combines the ability to analyze the same data at rest and in motion with predictive analysis. WSO2 DAS replaces WSO2 Business Activity Monitor and WSO2 Complex Event Processor. WSO2's analytics platform offers the flexibility to scale to millions of events per second, whether running on-premises and in the cloud.

The External Communications Layer

Our mapping provides the capabilities of this layer with the following products:

- WSO2 User Engagement Server (UES)
    - This product supports creating and managing portal-based and traditional Web UIs including supporting full personalization.
    - It is also used by the DAS to manage and host analytics dashboards.
- WSO2 API Manager which
    - Manages the lifecycle of the APIs and supports API publishers;
    - Offers a developer-focused portal for developers to find, explore and subscribe to APIs;
    - Issues and manages OAuth2 tokens to external developers (note that when WSO2 Identity Server is also available – see below – then this function is delegated to that system);
    - Gateways external requests and provides throttling and PEP capabilities;
    - Publishes usage, version and other data into the DAS; and
    - Integrates with WSO2 ESB

The Device Management Layer

WSO2 Enterprise Mobility Management (EMM) provides

- Mobile device management for iOS, Android, and IoT devices
- A full appstore for managing applications and provisioning applications on to managed devices
- Integration with the identity layer as well as DAS for mobile analytics

The Identity and Access Management Layer

WSO2 Identity Server supports this aspect, and provides the following capabilities:

- OAuth2 identity provider, issuing, revoking and managing tokens;
- Single sign-on support including SAML2 SSO and OpenID Connect support;
- Support for other identity protocols including WS-Federation (Passive), OpenID 2.0, Kerberos, Integrated Windows Authentication (IWA), and others
- Full support for XACML (including versions 2.0 and 3.0), acting as a PDP, PIP, and PAP;
- The ability to integrate between different identity providers and service providers including identity brokering
- Support for identity provisioning including SPML and SCIM support.

The WSO2 platform is the only modular, open source platform to provide all these capabilities (and more). As such it is the ideal basis for creating and deploying this IoT reference architecture.

One further aspect that is highly worth considering is the use of a platform-as-a-service (PaaS). WSO2 provides the WSO2 Private PaaS product that's based on the Apache Stratos project. This provides a managed, elastically scalable, HA deployment of the products mentioned above and also manages tenancy, self-service subscription, and many other aspects. It also supports managing many other useful server-side capabilities including PHP, MySQL, MongoDB and others. We have not shown the PaaS layer on the IoT reference architecture as some deployments may not need this capability.

6. Conclusion

In this paper we have outlined the following:

- What our definition of the IoT is
- Why a reference architecture is valuable
- The requirements on the reference architecture
- The instantiation of the reference architecture and how it meets those requirements
- A mapping of that reference architecture into the WSO2 platform

The IoT space is evolving rapidly and we expect this paper – and the associated technologies – to evolve in sync. However, despite the emerging nature of this space, this paper and the reference architecture are based on real-world projects that we have deployed with customers to support IoT capabilities. As such, we have great confidence this is a useful, deployable, and effective reference architecture. WSO2 is due to release WSO2 IoT Server in 2016 featuring platform support for IoT.

---

[1] http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf
[2] http://www.sciencedirect.com/science/article/pii/S138912861000156
[3] http://www.cs.bham.ac.uk/~garciaf/publications/Attack.MIFARE.pdf
[4] http://oauth.net/
[5] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
[6] http://mqtt.org/
[7] http://tools.ietf.org/html/draft-ietf-core-coap-18
[8] https://www.oasis-open.org/committees/mqtt/
[9] http://tools.ietf.org/html/rfc6455
[10]AMQP is an enterprise messaging protocol that supports pub/sub as well as queuing. It provides considerably higher qualities of service than MQTT including transactions. See http://amqp.org
[11] http://www.apache.org/licenses/LICENSE-2.0.html