AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

LABORATORY MANUAL

IoT and APPLICATIONS

SUBJECT CODE: 2180709

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

B.E. 8th SEMESTER

NAME:______________________________

ENROLLMENT NO: ______________________________

BATCH NO: ______________________________

YEAR: ______________________________

Amiraj College of Engineering and Technology,

Nr.Tata Nano Plant, Khoraj, Sanand, Ahmedabad.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Amiraj College of Engineering and Technology,

Nr.Tata Nano Plant, Khoraj, Sanand, Ahmedabad.

CERTIFICATE

This is to certify that Mr. / Ms. ____

Of class____________________ Enrolment No ____________________________has

Satisfactorily completed the course in _____________________________________as

by the Gujarat Technological University for ____ Year (B.E.) semester___ of Computer

Science and Engineering in the Academic year_______.

Date of Submission:-

Faculty Name and Signature
Subject Teacher

Head of Department
Computer

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

B.E. 8th SEMESTER

SUBJECT: IoT AND APPLICATIONS

SUBJECT CODE: 2180709

List Of Experiments

Sr. No.	Experiments	Date of Performance.	Date of submission	Sign	Remark
1	Define and Explain Eclipse IoT Project.				
2	List and summarize few Eclipse IoT Projects.				
3	Sketch the architecture of IoT Toolkit and explain each entity in brief.				
4	Demonstrate a smart object API gateway service reference implementation in IoT toolkit.				
5	Write and explain working of an HTTP-to-CoAP semantic mapping proxy in IoT toolkit				
6	Describe gateway-as-a-service deployment in IoT toolkit.				

7	Explain application framework and embedded software agents for IoT toolkit.				
8	Explain working of Raspberry Pi.				
9	Connect Raspberry Pi with your existing system components.				
10	Give overview of Zetta.				

Practical: 1

AIM: Define and Explain Eclipse IOT Project.

Eclipse IoT is an ecosystem of entities (industry and academia) working together to create a foundation for IoT based exclusively on open source technologies. Their focus remains in the areas of producing open source implementations of IoT standard technology; creating open source frameworks and services for utilization in IoT solutions; and developing tools for IoT developers.

Smarthome Project

SmartHome is one of Eclipse IoT's major services. It aims to create a framework for building smart home solutions, and its focus remains heterogeneous environments, meaning assorted protocols and standards integration.

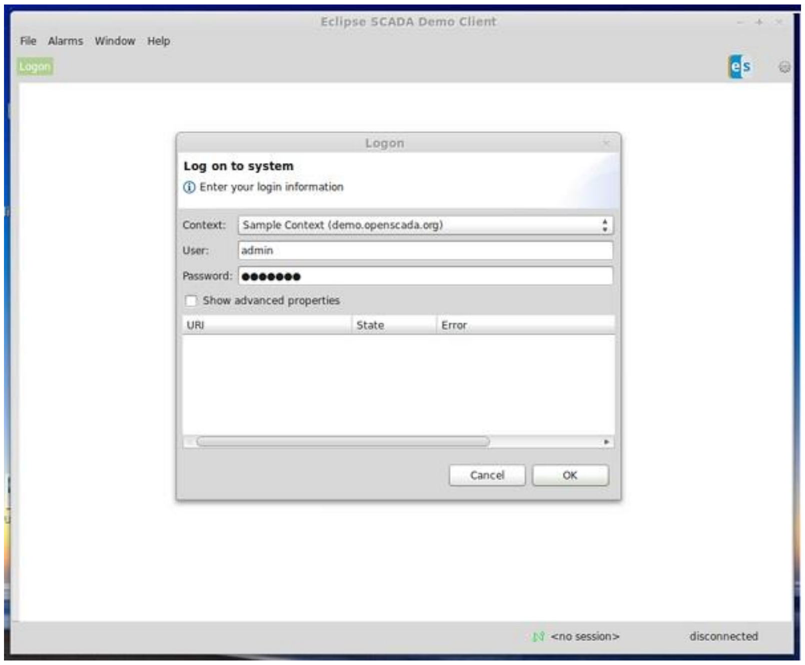
SmartHome provides uniform device and information access to facilitate interaction between devices. It consists of OSGi bundles capable of deployment in an OSGi runtime, with OSGi services defined as extension points.

OSGi bundles are Java class groups and other resources, which also include detailed manifest files. The manifest contains information on file contents, services needed to enhance class behavior, and the nature of the aggregate as a component. Review an example of a manifest below −

```
Bundle-Name : Hi Everyone                        // Bundle Name

Bundle-SymbolicName : xyz.xyz.hievery1           // Header specifying an identifier

Bundle-Description : A Hi Everyone bundle        // Functionality description

Bundle-ManifestVersion : 2                       // OSGi specification

Bundle-Version : 1.0.0                           // Version number of bundle

Bundle-Activator : xyz.xyz.Activator             // Class invoked on bundle activation

Export-Package : xyz.xyz.helloworld;version = "1.0.0" // Java packages available externally

Import-Package : org.osgi.framework;version = "1.3.0"   // Java packages needed from

// external source
```

Eclipse SCADA

Eclipse SCADA, another major Eclipse IoT service, delivers a means of connecting various industrial instruments to a shared communication system. It also post-processes data and sends data visualizations to operators. It uses a SCADA system with a communication service, monitoring system, archive, and data visualization.



Log on to system
Enter your login information

Context: Sample Context (demo.openscada.org)
User: admin
Password: ●●●●●●●
Show advanced properties

URI	State	Error

<no session> disconnected

It aims to be a complete, state-of-the-art open source SCADA system for developing custom solutions. Its supported technologies and tools include shell applications, JDBC, Modbus TCP and RTU, Simatic S7 PLC, OPC, and SNMP.

Practical: 2

AIM:List and summarize few Eclipse IoT Projects.

AllSeen Alliance (AllJoyn) -- The AllJoyn interoperability framework overseen by the AllSeen Alliance (ASA) is probably the most widely adopted open source IoT platform around.

Bug Labs dweet and freeboard -- Bug Labs started out making modular, Linux-based Bug hardware gizmos, but it long ago morphed into a hardware-agnostic IoT platform for the enterprise. Bug Labs offers a "dweet" messaging and alerts platform and a "freeboard" IoT design app. Dweet helps publish and describe data using a HAPI web API and JSON. Freeboard is a drag-and-drop tool for designing IoT dashboards and visualizations.

DeviceHive -- DataArt's AllJoyn-based device management platform runs on cloud services such as Azure, AWS, Apache Mesos, and OpenStack. DeviceHive focuses on Big Data analytics using tools like ElasticSearch, Apache Spark, Cassandra, and Kafka. There's also a gateway component that runs on any device that runs Ubuntu Snappy Core. The modular gateway software interacts with DeviceHive cloud software and IoT protocols, and is deployed as a Snappy Core service.

DSA -- Distributed Services Architecture facilitates decentralized device inter-communication, logic, and applications. The DSA project is building a library of Distributed Service Links(DSLinks), which allow protocol translation and data integration with third party sources. DSA offers a scalable network topology consisting of multiple DSLinks running on IoT edge devices connected to a tiered hierarchy of brokers.

Eclipse IoT (Kura) -- The Eclipse Foundation's IoT efforts are built around its Java/OSGi-based Kura API container and aggregation platform for M2M applications running on service gateways. Kura, which is based on Eurotech's Everywhere Cloud IoT framework, is often integrated with Apache Camel, a Java-based rules-based routing and mediation engine. Eclipse IoT sub-projects include the Paho messaging protocol framework, the Mosquitto MQTT stack for lightweight servers, and the Eclipse SmartHome framework. There's also a Java-based implementation of Constrained Application Protocol (CoAP) called Californium, among others.

Kaa -- The CyberVision-backed Kaa project offers a scalable, end-to-end IoT framework designed for large cloud-connected IoT networks. The platform includes a REST-enabled server function for services, analytics, and data management, typically deployed as a cluster of nodes coordinated by Apache Zookeeper. Kaa's endpoint SDKs, which support Java, C++ and C development, handle client-server communications, authentication, encryption, persistence, and data marshalling. The SDKs contain server-specific, GUI-enabled schemas translated into IoT object bindings. The schemas govern semantics and abstract the functions of a diverse group of devices.

Macchina.io -- Macchina.io provides a "web-enabled, modular and extensible" JavaScript and C++ runtime environment for developing IoT gateway applications running on Linux hacker boards. Macchina.io supports a wide variety of sensors and connection technologies including Tinkerforge bricklets, XBee ZB sensors, GPS/GNSS receivers, serial and GPIO connected devices, and accelerometers.

GE Predix -- GE's PaaS (Platform as a Service) software for industrial IoT is based on Cloud Foundry. It adds asset management, device security, and real-time, predictive analytics, and supports heterogeneous data acquisition, storage, and access. GE Predix, which GE developed for its own operations, has become one of the most successful of the enterprise IoT platforms, with about $6 billion in revenues. GE recently partnered with HPE, which will integrate Predix within its own services.

Home Assistant -- This up and coming grassroots project offers a Python-oriented approach to home automation. See our recent profile on Home Assistant.

Mainspring -- M2MLabs' Java-based framework is aimed at M2M communications in applications such as remote monitoring, fleet management, and smart grids. Like many IoT frameworks, Mainspring relies heavily on a REST web-service, and offers device configuration and modeling tools.

Node-RED -- This visual wiring tool for Node.js developers features a browser-based flow editor for designing flows among IoT nodes. The nodes can then be quickly deployed as runtimes, and stored and shared using JSON. Endpoints can run on Linux hacker boards, and cloud support includes Docker, IBM Bluemix, AWS, and Azure.

Open Connectivity Foundation (IoTivity) -- This amalgamation of the Intel and Samsung backed Open Interconnect Consortium (OIC) organization and the UPnP Forum is working hard to become the leading open source standards group for IoT. The OCF's open source IoTivity project depends on RESTful, JSON, and CoAP.

openHAB -- This open source smart home framework can run on any device capable of running a JVM. The modular stack abstracts all IoT technologies and components into "items," and offers rules, scripts, and support for persistence -- the ability to store device states over time. OpenHAB offers a variety of web-based UIs, and is supported by major Linux hacker boards.

OpenIoT -- The mostly Java-based OpenIoT middleware aims to facilitate open, large-scale IoT applications using a utility cloud computing delivery model. The platform includes sensor and sensor network middleware, as well as ontologies, semantic models, and annotations for representing IoT objects.

OpenRemote -- Designed for home and building automation, OpenRemote is notable for its wide-ranging support for smart devices and networking specs such as 1-Wire, EnOcean, xPL, Insteon, and X10. Rules, scripts, and events are all supported, and there are cloud-based design tools for UI, installation, and configuration, and remote updates and diagnostics.

Practical: 3

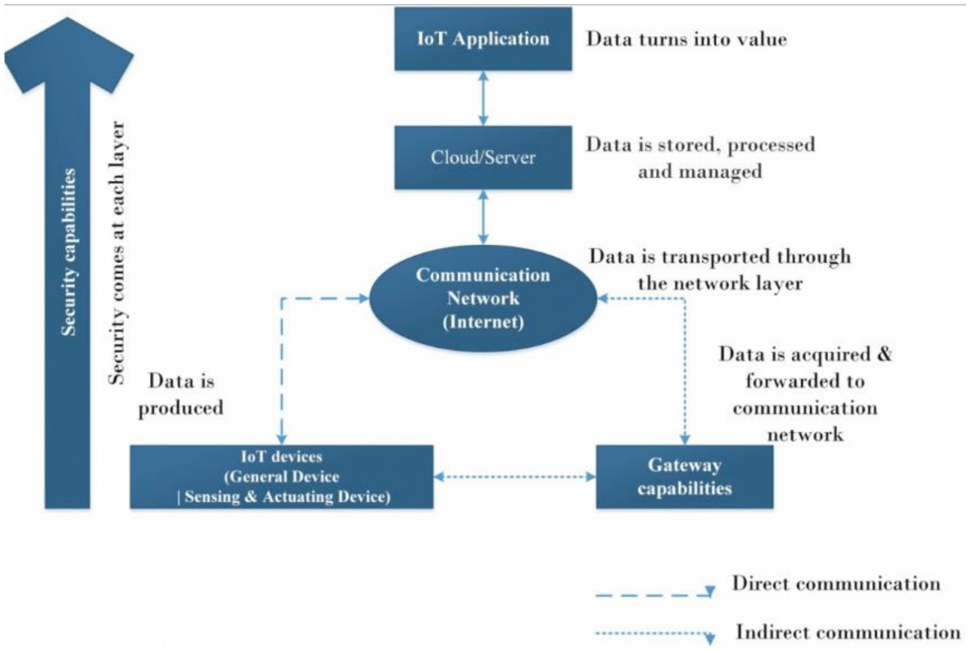
AIM: Sketch the architecture of IoT Toolkit and explain each entity in brief.

Architecture of an IOT System -

The different organizations and service providers define, implement and recognize IOT architecture in different ways. However, the basic architecture of an IOT system remains same underneath every implementation and business model. The basic architecture of an IOT system can be understood from a four-layer model as follow -

1) IOT devices and Gateways
2) Communication Network
3) Cloud or Server
4) IOT application

The data is generated, transported, processed and converted to useful insights by an IOT system. The basic architecture of an IOT system can be represented by the following block diagram -


IoT Application
Data turns into value
Cloud/Server
Data is stored, processed and managed
Communication Network (Internet)
Data is transported through the network layer
Data is acquired & forwarded to communication network
Security capabilities
Security comes at each layer
Data is produced
IoT devices (General Device | Sensing & Actuating Device)
Gateway capabilities
Direct communication
Indirect communication

1) IOT devices - Any device or equipment counts as an IOT device if it satisfies the following requirements -

a) It is capable of communicating with other devices and connect with an internet network. It must have hardware interfaces and firmware or operating system which can set up communication with other devices or connect to an internet network.

b) It must be equipped with sensors and/or actuators. The sensors may be collecting static or dynamic information from the physical world. The information or data collected by the sensor should be shared or exchanged with a server or cloud. The device may also have actuators to act upon or according to the processed data or insights sent back by the cloud or server.

c) The device must have a controller or processor to capture data, memory to store it (often temporarily) and firmware or operating system to process captured data or data received from the server or cloud.

Most of the IOT devices are built using standard IOT boards. These boards can be microcontroller boards or daughter boards (single board computers). Some of the popular IOT boards include Arduino, Raspberry Pi, Beagle Bone, CubieBoard, Pinnocio, Banana Pi and many others. The boards come with microcontroller or processor integrated with on-board memory (RAM and ROM), digital and analog GPIO (general purpose input output) pins and various communication channels (like USB, I2C, SPI, TWI, Ethernet). These boards can be stacked with other boards or sensors and actuators to form an IOT device (physical device).

The IOT devices can also be built by augmenting network interfaces, RF or Cellular transceivers with popular microcontrollers or processors. Such IOT devices are custom built for mission critical applications. Some of the leading microcontroller manufacturers include Texas Instruments (TI), ARM, Freescale, Intel, Microchip Technology, Atmel and Broadcom.

Based on the hardware design and capabilities, the IOT devices can be broadly categorized as follow -

1) General Devices
2) Sensing and Actuating Devices

General Devices - A general device is that device under IOT application domain which has embedded processing and communication capabilities. A general device can process some information and can connect to a communication network through wired or wireless interfaces. Basically, these devices only collect data and insights from a cloud or server and operate or perform data processing accordingly. For example, web controlled industrial machines or home appliances can be considered as general IOT devices.

Sensing and Actuating Devices - The sensing and actuating devices are equipped with sensors and actuators that enable them to interact and impact the real world. The sensors collect information pertaining to real physical quantities like temperature, humidity, light intensity, force, density etc and pass it to the on-board controller/processor. The controller or processor store the information (temporarily) and pass it on to the communication network. Through various layers of communication network, it is received at the cloud or server. The cloud process information and send back useful insights to operate actuators.

Role of Gateways

The IOT device may setup communication with other devices through a gateway or without a gateway. The gateways are basically required for protocol conversion. Suppose, an IOT device can send and receive data through Zigbee interface and so will communicate through Zigbee protocol. The communication network may be able to receive and send data through TCP-IP protocol. In such case, there will require a gateway which could convert data coming through the device using Zigbee protocol to data transmission through TCP-IP protocol and data coming from cloud or server through TCP-IP protocol to Zigbee protocol for reception by the IOT device. Since the communication network and the on-board network of the IOT device are different, the gateway act as a two-way bridge between the two networks.

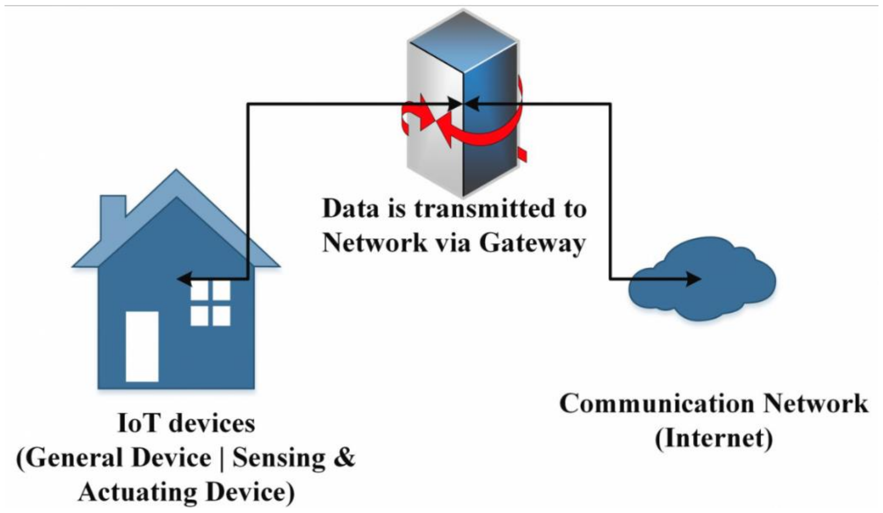
The gateway collects and extract the (sensor) data as per the device protocol, wrap and format it according to the protocol the communication network be operating at and push data to the communication network for transmission to the cloud or server. Same way, it receives and extract data, insights or information from the cloud or server, wrap and format it according to the network protocol utilized by the on-device network and push the cloud processed data to the IOT device.

So, a gateway may be required in either of the two scenarios -

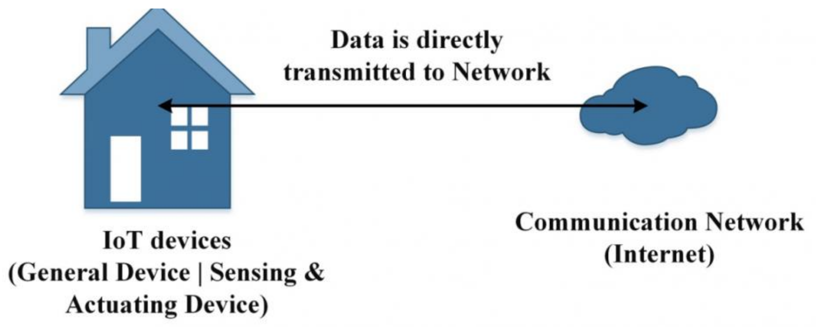
1) When the IOT device and the communication network may be operating at different protocols. Often, these protocols may be at different network layers. Like from the example above, the Zigbee is a physical layer protocol while the TCP-IP is a transport layer protocol. A wireless sensor network is another example of device to network communication through gateways.

2) One IOT device may need to communicate with another IOT device operating at different protocol. For example, a bluetooth device may communicate with other BLE devices over the air using a gateway.

So, the gateways provide indirect way of communication between device and cloud or one device and another device. In case of device to device communication, the IOT endpoints (individual IOT devices) may be co-located and communicating at different physical or link layer protocols (RF protocols like Bluetooth, Wi-Fi, Zigbee, Bluetooth-LE) through a gateway. Such a gateway is called edge gateway.


Data is transmitted to Network via Gateway
IoT devices (General Device | Sensing & Actuating Device)
Communication Network (Internet)

An IOT device can also connect to a cloud or other IOT device directly. In such case, the device and the communication network or the devices communicating with each other must be sharing and exchanging data using same protocol. So, there would be no need of protocol conversion and so any gateway. Usually, such device to device or device to network communication is possible through application layer protocols like Message Queuing Telemetry Transport (MQTT), Constrained Application Protocol (CoAP), Data Distribution Service (DDS), Advanced Message Queuing Protocol (AMQP), and Extensible Messaging and Presence Protocol (XMPP). For example, one ESP8266 IOT board can directly communicate with another ESP8266 board directly using MQTT protocol. MQTT is an application layer protocol.


Data is directly transmitted to Network
IoT devices (General Device | Sensing & Actuating Device)
Communication Network (Internet)

The IOT devices (IOT boards) may have a firmware, operating system or real time operating system to process data, perform messaging and communication, manage data storage and manage actuator operations. Some of the popular IOT operating systems are Embedded Linux, TinyOS, Snappy Ubuntu Core, Contiki, FreeRTOS, Mantis, ARM's mbedOS, RIOT OS, Windows 10, Nucleus RTOS, eCOS, SAFE ROTS, Android Things, Green Hills Integrity, WindRiver VxWorks and BrilloOS.

2) Communication Network - The communication network is generally the typical internet network having different layers (Physical, Link, Network, Transport and Application) and communication protocols operating at different layers.

3) Cloud/Server - The cloud or server is the edge of the IOT system. A cloud stores data collected from different and myriad of IOT devices and perform data mining and analytics to derive useful insights from it. It is also responsible for managing the connected devices and networks, manage device to device communications and implement IOT applications by operating and synchronizing different IOT devices and communication between together. The cloud may also communicate with other private and public cloud services to enable an IOT application.

4) IOT Application - The processing, mining and analysis of the data at the cloud is done by the IOT application. The IOT application is the piece of software at the cloud server which extracts data, manipulate it to derive useful insights and manage to securely push insights to the target IOT devices. For example, an IOT application designed for home automation might process data from sensors and send commands from the cloud to operate home appliances.

Practical: 4

AIM: Demonstrate a smart object API gateway service reference implementation in IoT toolkit.

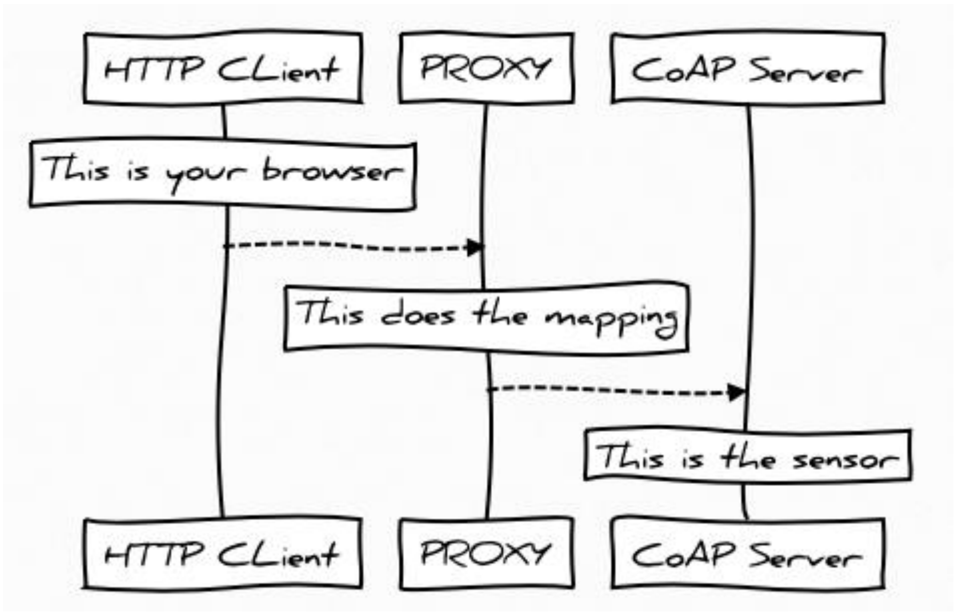
The Smart Object API is a Semantic Web Linked Data application for the Internet of Things (IoT). It consists of a URI-Object encapsulation of semantic and real-time data properties associated with features of interest. The Smart Object architecture roughly conforms to the Virtual Entity, the Information Model, and the Channel Model set out in the IoT-A Architecture Reference Model (IoT-A ARM). Supports direct interaction between smart sensors, smart gateways, cloud/internet services, and user devices. Interaction uses standard web protocols and formats and is semantically a superset of the CoAP protocol.

Service framework is to include object creation from semantic metadata, semantic database, discovery, and linkage, API capability keys, and threaded server.

Practical- 5

Aim: Write and explain working of an HTTP- to-CoAP semantic mapping proxy in IoT toolkit

The point of the draft (soon RFC) is to describe how to do HTTP-to-CoAP Mapping to allow HTTP clients to access a CoAP Server via a proxy. This works under the assumption that, despite the similarities between CoAP and HTTP, not all browsers will implement support for it and that some legacy devices will need proxying. Another assumption is that users will like to use their smartphones with their home sensors. The set up would look like this:



HTTP-to_CoAP Mapping Scenario with world-class graphics ;)

HTTP-to-CoAP Proxy

A HC proxy is accessed by an HTTP client which wants to access aresource on a CoAP server. The HC proxy handles the HTTP request by mapping it to the equivalent CoAP request, which is then forwarded to the appropriate CoAP server. The received CoAP response is then mapped to an appropriate HTTP response and finally sent back to the originating HTTP client.

See Figure 1 for an example deployment scenario. Here a HC proxy is located at the boundary of the Constrained Network domain, to avoid sending any HTTP traffic into the Constrained Network and to avoid any (unsecured) CoAP multicast traffic outside the Constrained Network.

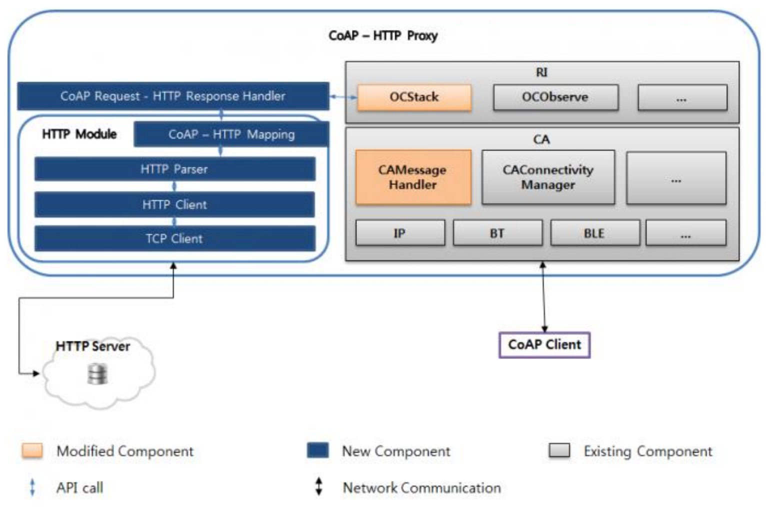
A DNS server (not shown) is used by the HTTP Client to resolve the IP address of the HC proxy and optionally also used by the HC proxy to resolve IP addresses of CoAP servers.

```
                                             Constrained Network
                                              .--------------------.
                                             /      .------.        \
                                            /       | CoAP |         \
                                           /        |server|          \
                                          ||        '------'          ||
                                          ||                          ||
     .--------.  HTTP Request   .------------.CoAPReq  .------.   ||
|  HTTP  |---------------->|HTTP-to-CoAP|----------->| CoAP |   ||
     | Client |<----------------|   Proxy    |<-----------|Server|   ||
     '--------'  HTTP Response  '------------'  CoAPResp '------'   ||
                                          ||                          ||
                                          ||   .------.               ||
                                          ||   | CoAP |               ||
                                           \   |server|  .------.     /
\  '------'  | CoAP |   /
                                             \           |server|  /
                                              \          '------' /
                                               '-----------------'
```

Figure 1: HTTP-To-CoAP Proxy Deployment Scenario

Normative requirements on the translation of HTTP requests to CoAP requests and of the CoAP responses back to HTTP responses are defined in the basic mapping of request methods and simple response code mapping between HTTP and CoAP, and leaves many details of the cross-protocol HC proxy for future definition. This document provides additional guidelines and more details for the implementation of a HC Proxy,which should be followed in addition to the normative requirements. Note that the guidelines apply to all forms of an HC proxy (i.e. Reverse, Forward, Intercepting) unless explicitly otherwise noted.

Proposed Architecture:



CoAP – HTTP Mapping: **Contains mapping between CoAP and HTTP Request, Response and Error codes.**

CoAP Request - HTTP Response Handler: Implements conversion of CoAP request to HTTP request and HTTP response to CoAP response with help of "CoAP-HTTP Mapping" module.

HTTP Parser: Implements functionality of parsing HTTP responses.

HTTP Client: Implements functionality of generating HTTP requests.

TCP Client: Implements TCP client to interact with HTTP servers.

Practical- 6

Aim: Describe gateway-as-a-service deployment in IoT toolkit.

The Internet of Things (IoT) is set to occupy a substantial component of future Internet. The IoT connects sensors and devices that record physical observations to applications and services of the Internet. As a successor to technologies such as RFID and Wireless Sensor Networks (WSN), the IoT has stumbled into vertical silos of proprietary systems, providing little or no interoperability with similar systems. As the IoT represents future state of the Internet, an intelligent and scalable architecture is required to provide connectivity between these silos, enabling discovery of physical sensors and interpretation of messages between things. This paper proposes a gateway and Semantic Web enabled IoT architecture to provide interoperability between systems using established communication and data standards. The Semantic Gateway as Service (SGS) allows translation between messaging protocols such as XMPP, CoAP and MQTT via a multi-protocol proxy architecture. Utilization of broadly accepted specifications such as W3C's Semantic Sensor Network (SSN) ontology for semantic annotations of sensor data provide semantic interoperability between messages and support semantic reasoning to obtain higher-level actionable knowledge from low-level sensor data.

While developing IoT solutions we come across common tasks of connecting “things” to the cloud. For devices that are not connected to the cloud directly, gateways are used. Such gateway can be a system on chip (SoC) device: cable modem, set top box, home or industrial automation gateway, smart phone, home entertainment system, laptop or PC. All these gateway should have some sort of interface (BLE/ZigBee/etc radios, adapter, digital or analog inputs) that can connect to the actual device: lamp, controller, sensor, appliance.

Semantic Gateway as Service (SGS)

The heart of the semantic IoT architecture is the SGS, which bridges low level raw sensor information with knowledge centric application services by facilitating interoperability at messaging protocol and data modeling level. The description below is complemented by Open Source code available at https://github.com/chheplo/node-sgs which is further being enhanced and evaluated in the context of CityPulse, a large multi-institutional EU FP7 supported project along with an effort for additional community engagement and development.

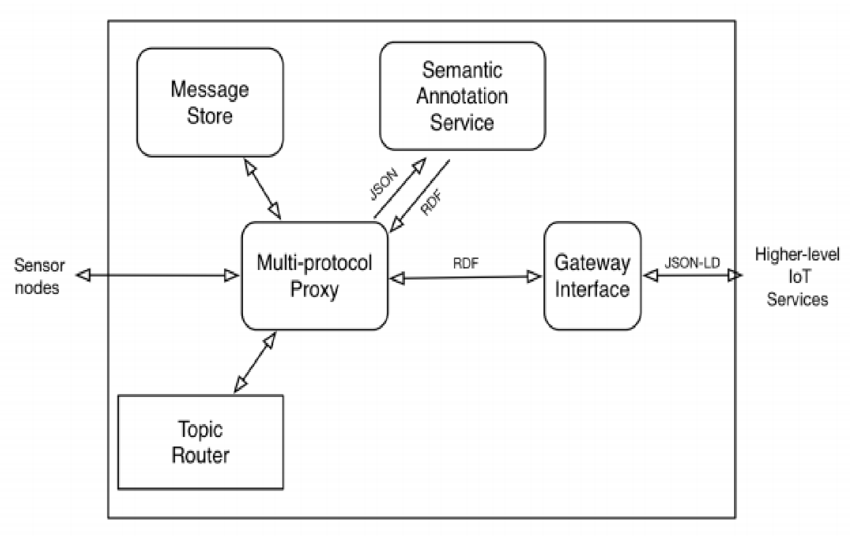


Fig. Gateway as a service architecture

The SGS has three core components as described in Figure:

(1) multi-protocol proxy,
(2) semantic annotation service,
(3) Gateway service interface.

The SGS also has components for required capabilities such as message store and topics router, which assist multi-protocol proxy and gateway service interface in translation between messaging protocol. At a high level, SGS architecture connects external sink nodes to the gateway component using primitive client agents, which support MQTT, XMPP or CoAP. In contrast, the gateway service interface connects cloud services or other SGSs via REST or pubsub protocol. Before raw sensor data is forwarded from proxy to gateway interface, it is annotated using SSN and domain specific ontologies. Although the semantically annotated data is in RDF format at the multi-protocol proxy, the gateway interface converts the data into JSON, specifically linked data (JSON-LD) format to support RESTful protocols.

Practical- 7

Aim: Explain application framework and embedded software agents for IoT toolkit.

IoT Frameworks

For an IoT framework to be reliable and dependable, some minimal set of measures should be satisfied to achieve integration and interoperability in IoT. These frameworks span across the IoT research communities ranging from academic research to organisational research which focus on integrating things in IoT. Since IoT paradigm itself is still in evolving state, we propose a set of minimal measures to be satisfied by IoT frameworks for integration. These are:

• Contract decoupling: An IoT system contains heterogeneous devices with disparate communication protocols. An integration framework should be competent enough to efficiently handle contract decoupling. Contract decoupling is the ability of service consumers and service producers toindependently evolve without terminating the contract between them [19]. For example, a service might be in a JSON format and the service consumer needs an input in XML. The framework should provide support to transform the message to the format that fulfils the contract between them.

• Scalability: Given the evolving nature of IoT and the predictions and calculations by [3] and [2], an efficient integration framework should be scalable and evolvable enough to support the billions of things soon to be connected to the internet.

• Ease of testing: An integration framework should support ease of testing and debugging. It should provide support for debugging defects and failures, integration testing, component testing, system testing, compatibility testing, installation test, functional and non-functional testing, performance testing and security testing.

• Ease of development: An IoT integration framework should provide a means of easy development for developers. The framework should exclude all complexities and provide proper documentation for non-developers and developers with basic programming knowledge to easily understand the internals of the framework.

• Fault tolerance: An IoT system has to be dependable and resilient. An intelligent integration framework should effectively handle faults as IoT devices can eventually toggle between offline and online states. The framework should provide self-healing mechanisms for transient faults (network faults, node level faults, etc.), unauthorised access error, server crash failure, omission failure (when the server does not receive incoming requests from client), timing fault, etc.

• Lightweight implementation: Integration frameworks should have a lightweight overhead both in its development and deployment stage. It should be lightweight and easy to install, uninstall, activate, deactivate, update, versioning and adaptable.
• Service coordination: Service coordination is the orchestration and choreography of services. Service orchestration is the coordination of multiple services by a mediator acting as a centralised component. Service choreography on the other hand, is the chaining of services together to execute a particular transaction. Integration frameworks should support at least either or both to achieve reliability.
• Inter domain operability: The framework should further be extensible to support inter domain communication. For example, in a smart car domain, an integration framework should also provide support for communication and interaction with traffic lights, road closure, etc. belonging to a smart city domain.

Application Framework for IOT

IoT applications have been developed and deployed in several domains such as transportation and logistics, healthcare, retail and supply chain, industry and environment [5, 6]. Despite their pervasiveness, developing IoT applications remains challenging and time-consuming. This is because it involves dealing with several related issues, such as lack of proper identification of roles of various stakeholders, as well as the lack of appropriate frameworks to address the large-scale and heterogeneity in IoT systems [7]. Another major challenge is the difficulty in achieving effective programming abstractions at different technology layers, ranging from device software to middleware services and end-user applications [8]. These difficulties increase the development time, resources and delay the deployment of the IoT applications. The complexity of IoT applications implies that it is inappropriate to develop one in an ad hoc manner and as a result, a framework is required. An IoT application framework can simplify the difficult process of coping with heterogeneous devices and software components, overcoming the complexities of distributed system technologies, handling a high volume of data, designing an architecture for the application, implementing it in a program, writing specific code to validate the application, and finally deploying it. A number of researchers have proposed several IoT application frameworks with each having its own strength and weakness. A study of the various application development frameworks for IoT is an important step for designing and developing a high-quality IoT application.

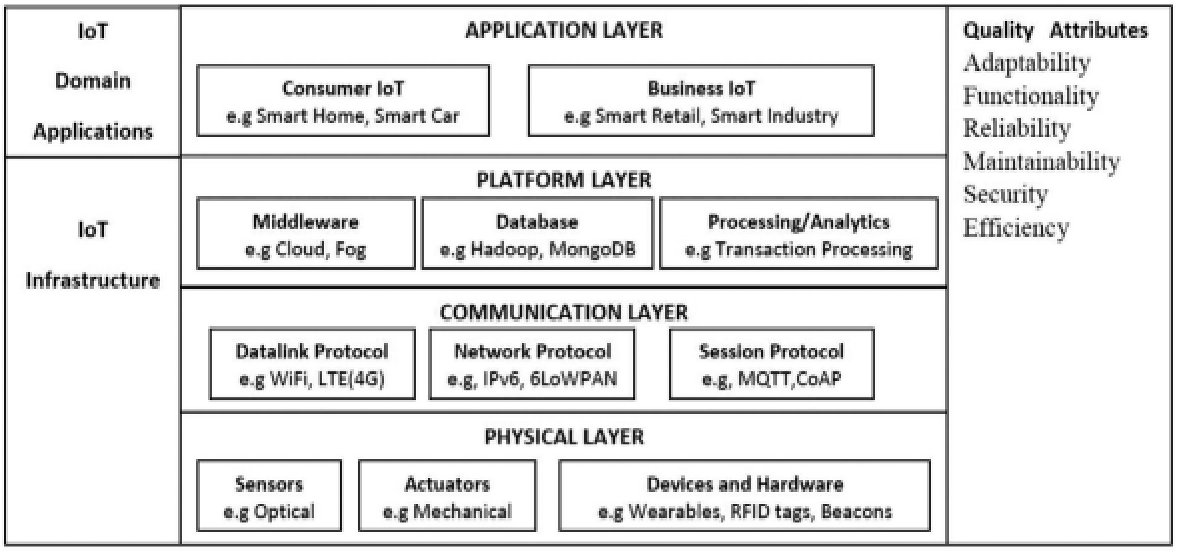


Fig. IoT technology Architecture

Agent Toolkits

Currently there are a variety of toolkits available on the market ranging from general agent development platforms, like AgentBuilder developed by Reticular Systems, to highly specialized tools, like Excalibur developed by the Technical University of Berlin which allows for the creation of autonomous agents in a complex computer-game environment. The AgentBuilder web site2 identifies numerous agent toolkits available on the market.

What are Agent Toolkits?

There is no universal definition of agent toolkits. Each vendor uses its own explanation of the term. For example, Reticular Systems states that its AgentBuilder toolkit application "is an integrated tool suite for constructing intelligent software agents". Authors of the Java Agent Development Environment (JADE) define their toolkit as "a software framework to make easy the development of agent application for interoperable multi-agent systems"

An agent toolkit is defined as any software package, application or development environment that provides agent builders with a sufficient level of abstraction to allow them to implement intelligent agents with desired attributes, features and rules. Some toolkits may offer only a platform for agent development, whereas others may provide features for visual programming. Agent toolkits may also provide an environment for running, monitoring, analyzing and testing agents, which is very important for both researchers and students learning about agent technologies. For example, in case of multi-agent systems, an agent development environment provides a context for agent interaction and sets of governing rules.

Why are Agent Toolkits needed?

The reasons why agent developers use agent toolkits is similar to those reasons why software developers who deal with object-oriented programming (OOP) prefer to use special development environments like Java VisualAge or Microsoft Visual Basic. First, they provide a certain level of abstraction in which programmers can develop their objects. Second, they incorporate some features of visual programming, which saves much time and makes development easier, more attractive and enjoyable. Third, they offer run-time testing and debugging environments. Finally, they allow programmers to reuse classes (definitions of objects) created by other programmers. Unfortunately, existing OOP development platforms and compilers do not support all facets of agent development. For example, they do not address the implementation of agent features, agent interaction rules, communication language, and a common knowledge base. This is why a new suite of agent toolkits has appeared on the market in the last few years: to create a development environment that fully supports agent creation, testing, and reuse.

Practical- 8

Aim: Explain working of Raspberry Pi.

The Raspberry pi is a single computer board with credit card size, that can be used for many tasks that your computer does, like games, word processing, spreadsheets and also to play HD video. It was established by the Raspberry pi foundation from the UK. It has been ready for public consumption since 2012 with the idea of making a low-cost educational microcomputer for students and children. The main purpose of designing the raspberry pi board is, to encourage learning, experimentation and innovation for school level students. The raspberry pi board is a portable and low cost. Maximum of the raspberry pi computers is used in mobile phones.

Raspberry Pi Technology

The raspberry pi comes in two models, they are model A and model B. The main difference between model A and model B is USB port. Model a board will consume less power and that does not include an Ethernet port. But, the model B board includes an Ethernet port and designed in china. The raspberry pi comes with a set of open source technologies, i.e. communication and multimedia web technologies.In the year 2014, the foundation of the raspberry pi board launched the computer module that packages a model B raspberry pi board into module for use as a part of embedded systems, to encourage their use.

Raspberry Pi Hardware Specifications

The raspberry pi board comprises a program memory (RAM), processor and graphics chip, CPU, GPU, Ethernet port, GPIO pins, Xbee socket, UART, power source connector. And various interfaces for other external devices. It also requires mass storage, for that we use an SD flash memory card. So that raspberry pi board will boot from this SD card similarly as a PC boots up into windows from its hard disk.

Essential hardware specifications of raspberry pi board mainly include SD card containing Linux OS, US keyboard, monitor, power supply and video cable. Optional hardware specifications include USB mouse, powered USB hub, case, internet connection, the Model A or B: USB WiFi adaptor is used and internet connection to Model B is LAN cable.

Memory

The raspberry pi model aboard is designed with 256MB of SDRAM and model B is designed with 51MB.Raspberry pi is a small size PC compare with other PCs. The normal PCs RAM memory is available in gigabytes. But in raspberry pi board, the RAM memory is available more than 256MB or 512MB

CPU (Central Processing Unit)

The Central processing unit is the brain of the raspberry pi board and that is responsible for carrying out the instructions of the computer through logical and mathematical operations. The raspberry pi uses ARM11 series processor, which has joined the ranks of the Samsung galaxy phone.

GPU (Graphics Processing Unit)

The GPU is a specialized chip in the raspberry pi board and that is designed to speed up the operation of image calculations. This board designed with a Broadcom video core IV and it supports OpenGL

Ethernet Port

The Ethernet port of the raspberry pi is the main gateway for communicating with additional devices. The raspberry pi Ethernet port is used to plug your home router to access the internet.

GPIO Pins

The general purpose input & output pins are used in the raspberry pi to associate with the other electronic boards. These pins can accept input & output commands based on programming raspberry pi. The raspberry pi affords digital GPIO pins. These pins are used to connect other electronic components. For example, you can connect it to the temperature sensor to transmit digital data.

XBee Socket

The XBee socket is used in raspberry pi board for the wireless communication purpose.

Power Source Connector

The power source cable is a small switch, which is placed on side of the shield. The main purpose of the power source connector is to enable an external power source.

UART

The Universal Asynchronous Receiver/ Transmitter is a serial input & output port. That can be used to transfer the serial data in the form of text and it is useful for converting the debugging code.

Display

The connection options of the raspberry pi board are two types such as HDMI and Composite.Many LCD and HD TV monitors can be attached using an HDMI male cable and with a low-cost adaptor. The versions of HDMI are 1.3 and 1.4 are supported and 1.4 version cable is recommended. The O/Ps of the Raspberry Pi audio and video through HMDI, but does not support HDMI I/p. Older TVs can be connected using composite video. When using a composite video connection, audio is available from the 3.5mm jack socket and can be sent to your TV. To send audio to your TV, you need a cable which adjusts from 3.5mm to double RCA connectors.

Model of a Raspberry Pi Board

The Raspberry Pi board is a Broadcom (BCM2835) SOC (system on chip) board. It comes equipped with an ARM1176JZF-S core CPU, 256 MB of SDRAM and 700 MHz,. The raspberry pi USB 2.0 ports use only external data connectivity options. The board draws its power from a micro USB adapter, with min range of 2. Watts (500 MA). The graphics, specialized chip is designed to speed up the operation of image calculations. This is in built with Broadcom video core IV cable that is useful if you want to run a game and video through your raspberry pi.

Model A Raspberry Pi Board

Features of Raspberry PI Model A

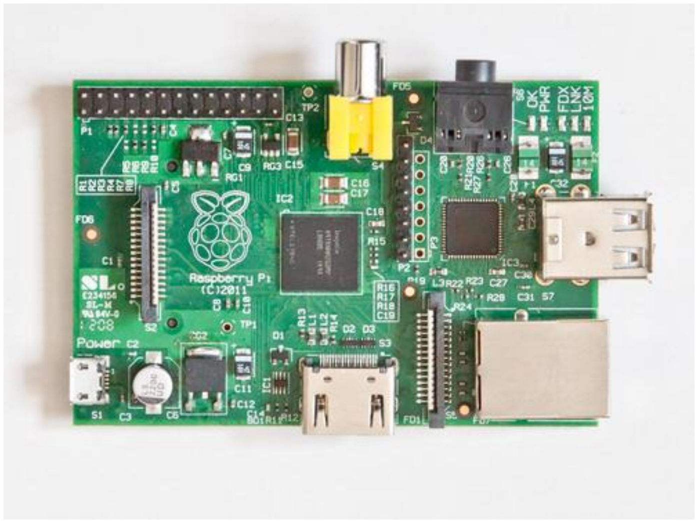
- The Model A raspberry pi features mainly includes

- 256 MB SDRAM memory
- Single 2.0 USB connector
- Dual Core Video Core IV Multimedia coprocessor
- HDMI (rev 1.3 & 1.4) Composite RCA (PAL and NTSC) Video Out
- 3.5 MM Jack, HDMI, Audio Out
- SD, MMC, SDIO Card slot on board storage
- Linux Operating system
- Broadcom BCM2835 SoC full HD multimedia processor
- 8.6cm*5.4cm*1.5cm dimensions

Practical- 9

Aim: Connect Raspberry Pi with your existing system components.

Python is the Pi’s recommended programming language, but Linux is its recommended operating system. Nearly every flavor of OS that works on Raspberry Pi—Raspbian, Pidora and more—is a riff on the Linux kernel.



The front of a Raspberry Pi Model B.

Right now, there are two versions of the Raspberry Pi for sale—Model A and Model B, though neither is newer than the other. Model A, which is $25, lacks Ethernet capability, has a single USB connecter, and 256MB of memory. Model B, which is $35, has double the memory, Ethernet, and a dual USB connector. The B is not an improvement on A, and in fact was available first; the A is just a lighter, cheaper version. The Foundation hasn't ruled out an eventual, more powerful Model C, but probably not for at least "two to three years."

Getting Started With Raspberry Pi

Raspberry Pi owes its low price tag to advances in integrated chips. Instead of having a CPU, a GPU, a USB controller, and memory each on their own individual chips, Raspberry Pi uses a system-on-a-chip with all those components on a single chip.

Without a lot of chips to take up space, the Pi itself can consist of a printed circuit board which boots up from an SD memory card. So it's not just cheap, it's simple, too.

Still, the $35 price tag is a bit misleading. You can't just buy a Raspberry Pi and expect it to work right out of the box. Here are the accessories you'll need to get up and running:

- A power supply. Raspberry Pi doesn't come with one, so you'll need a micro USB compatible cable in order to plug it into the wall.
- A case. There's no official one yet, so I put mine in this pink one from Adafruit. Unfortunately, despite what you may have heard, it does not fit in an Altoids tin.
- An HDMI cable or RCA video lead. You can't use your Pi without a visual display. You can either plug it into a computer monitor with HDMI input using an HDMI cable, or you can plug it into an analogue TV with a standard RCA composite video lead.
- A USB mouse and keyboard. Or else how will you interact with the Pi? Any wired or wireless mouse and keyboard should do; I'm using wireless Logitech products for both.
- An SD memory card. You'll need one to boot up the Pi. The Raspberry Pi foundation recommends at least 4 gigs to start, but as many as 32 if you want.
- A primary computer. I didn't get that you can't just get the Pi running without already owning another computer, Mac or PC. Hopefully you already have one of these, or this project just got a lot more expensive.
- An SD memory card reader. The Raspberry Pi doesn't need this, but your primary computer does so you can transfer installations from it to the Pi. A lot of computers come with a built-in card reader, but if yours doesn't, you might want to invest in one.

Now, let's fast-forward to the day when your Raspberry Pi and all its accessories arrive in the mail. Here's what to do, and when to do it.

- Put your Raspberry Pi in its case. Unless it's very customized, it should continue to have holes in it for all of the Pi's inputs.
- Put the Pi aside and go to your primary computer. Insert your SD card and format it according to the Foundation's directions. This will install a recovery program on it so you can save your card even if you break it with your tinkering.

- Download NOOBS on your primary computer. Short for New Out Of Box Software, it's the Raspberry Pi Foundation's fittingly named distro for first-time Pi users. A distro is a package installation of Linux and its associated software.
- Load your NOOBS download onto the newly formatted SD card.
- Time to get started with the Raspberry Pi. Slide the SD card into the underside of the Raspberry Pi, and make sure it's oriented correctly; it'd be bad to break your Pi before you turn it on!
- Connect it to the power supply, monitor, keyboard, and mouse.
- The Raspberry Pi will boot up and take you the NOOBS screen. If it doesn't, check your power supply and HDMI cables and make sure they're secure.
- Select an OS to install. If you select the default Raspbian, recommended for beginners, Adafruit has a great tutorial on the process. This install will take a while (20 minutes for me) so this is a good time to go do something else.
- Once the file copies, you'll get a notice that says, "Image applied successfully." Press return, and the Pi will reboot. Now it will boot into the operating system's graphical user interface, which looks a lot like Windows 98.

Now you're ready to use your Raspberry Pi however you like. You can run programs on it as if it were any other computer, or you can choose to work from the command line. Since it's a general purpose Linux machine, what you do from here is up to you.

A Raspberry Pi Model B all plugged in.

A word of caution, however, from somebody who already made this mistake: don't delete the NOOBS copy you downloaded on your primary computer. My husband and I wiped the Pi twice (and installed operating systems three times) in one night, so I know it saves time to have everything ready on your computer in case you want to start fresh for any reason.

Pi Project Tutorials for Beginners

With 512 MB on the Model B, Raspberry Pi isn’t the strongest computer in the world, but it’s still powerful enough for any project a beginner can think up.

See also: 12 Ways to Make The Most of Raspberry Pi

Here are ten awesome-sounding, relatively simple tutorials for beginners:

Print Server

This is the tutorial we used, so I can vouch for its ease of use. It makes use of CUPS (Common UNIX Printing System) and basically all you have to do is install it on your SD card and then teach Raspberry Pi the address of your printer.

XMBC Media Center

This seems to be one of the most popular uses of a Raspberry Pi. Since it is capable of running XMBC, a program that organizes all of your movies, TV, music, and more into one easy-to-use cloud-based corral, a Pi makes a perfect hub for streaming your media over your network.

Program Your Own Game

Sure, you could sit around playing Minecraft on your Pi, but you could also fulfill your secret dream of becoming a video game developer. Programmer Andy Balaam made a tutorial on the topic so thorough, it takes three hours to watch all of it.

Create an Information Kiosk

Brendan Nee was sick of arriving late for buses, so he programmed his Pi to display real-time arrival predictions for transit around his house. His step-by-step instructions are great for San Franciscans, but if you live somewhere else you’ll need to configure for another transit system.

A desktop computer built with a Raspberry Pi.

Build a Pi PC

You’ve already got the monitor, keyboard, and mouse for your Pi. Why not go the rest of the way and turn it into a self-contained computer? Mike Davis’s tutorial shows you how to attach the Pi to the back of the monitor to create a compact desktop PC.

Time Lapse Dolly

Instead of buying an expensive professional camera rig to take time lapse shots, Rick Adam wrote just 25 lines of Python code to build his own. The results are gorgeous time lapse movies that show a few hours in a couple of seconds.

Affordable Bitcoin Mining Rig

Instead of buying a $4,000 plus Bitcoin miner, you can set up your Raspberry Pi to do it for just $83. However, given the amount of energy required to mine Bitcoins, we highly doubt you’ll get rich off of a Raspberry Pi’s diminutive mining power.

Solar Powered Pi

Save electricity and run your Pi off the power of the sun with this tutorial. The creator says that this method will usually give you five hours of battery life on your Pi.

Web Server

Design your first website, and get it online, too, by turning your Raspberry Pi into your own home Web server. So long as you don’t expect your site to get loads of traffic, you can have the Pi host it instead of a pricey online host.

Raspberry Pi Internet Radio

With 300 lines of Python code, this is the most complicated tutorial on the list, but perhaps with the most payoff. Set up your Pi to load a playlist of streaming songs as well as display what’s playing with an LED display.

Practical- 10

Aim: Give overview of Zetta.

What is Zetta?

Zetta is an open source platform for IoT on which we can build APIs for device interaction. The platform is built on Node.js. People who are familiar with Node.js can easily get started with Zetta but, for beginners, a basic understanding of Node.js is required.

Let’s understand the Zetta platform and its characteristics

- Zetta is an open source platform, so anyone can use it free of cost. If you are passionate about Node.js (https://nodejs.org/), then you can contribute to this open source project. Currently, the community is small, but it’s growing. Basically, it’s a tool that will help to generate APIs which we can use to communicate between devices.
- Node.js is basically a server-side JavaScript. Developers can define devices as state machines using JavaScript. It is also cross-platform and is easily deployable in multiple cloud platforms.
- Zetta is an API driven platform. Every call is API based so that we can use these APIs for any other purpose like sending data to other analytics platforms.
- Zetta exposes Websocket endpoints to stream real-time events. This model of merging Hypermedia with Websocket streaming is acknowledged as Reactive Hypermedia.

- It can support almost all device protocols, and mediate them to HTTP. Connections between servers are also persistent, so that we can use the seamless services between servers in the cloud.
- We can create stateless applications in Zetta servers. Applications can be useful to connect devices, run different queries and interact between them. We can also write queries and triggers so that whenever new devices are attached, we will be notified.

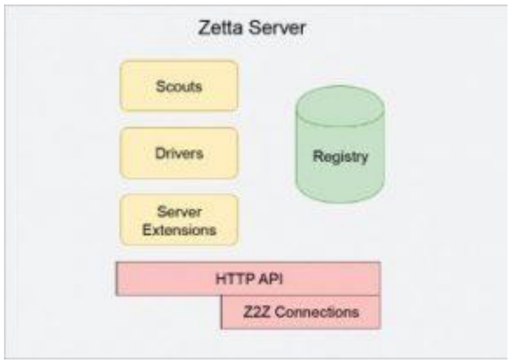


Figure 1: Zetta architecture

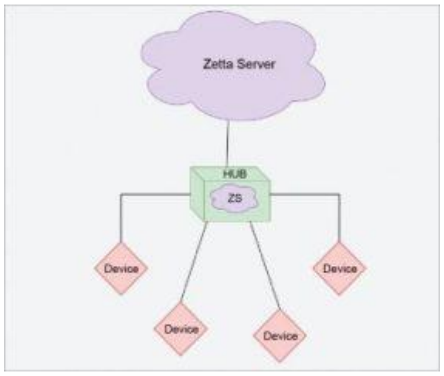


Figure 2: Zetta deployment

Zetta architecture: The Zetta server: The Zetta server is the main component of Zetta, which contains different sub-components like drivers, scouts, server extensions and apps. A Zetta server will run on a hardware hub such as BeagleBone Black, Raspberry Pi or Intel Edison. The server manages interactions between all the sub-components in order to communicate with devices and generate APIs, by which consumers can interact.

Scouts: Scouts provide a discovery mechanism for devices on the networks or to those which require system resources to understand and communicate with a specific type of protocol.

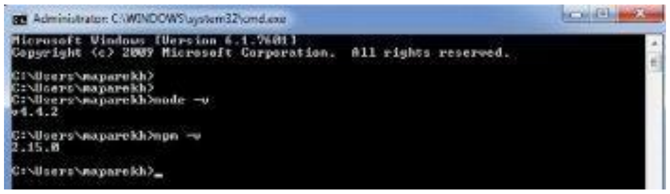
Scouts help Zetta to search for devices based on a particular protocol. They also fetch specific information about devices and whether those devices have already interacted with Zetta or not. They maintain security credentials and related details while communicating.

Drivers: Drivers are used to represent the devices in a state machine format. They are used for modelling devices and physical interaction between devices. Device models are used to generate different API calls.

Server extensions: These are used for extending functionalities. They are in a pluggable mode, and deal with API management, adding additional securities, etc.

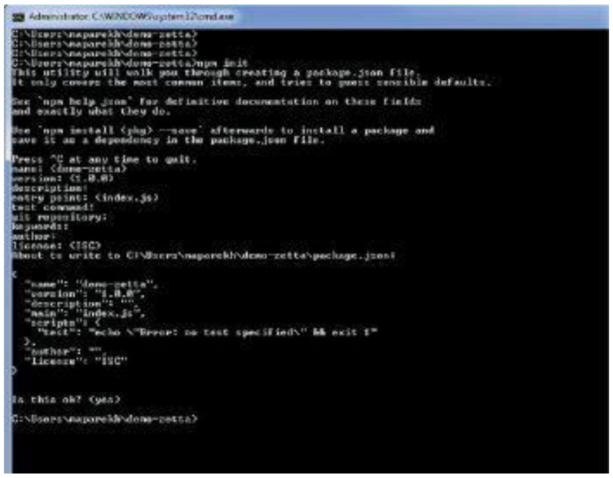
Registry: This is a database for the Zetta server, which stores information about the devices connected to the server. It is a persistence layer.

Secure linking: We can establish secure and encrypted tunnelling between different servers while communicating. This takes care of firewalls and network settings.



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\maparekh>
C:\Users\maparekh>
C:\Users\maparekh>node -v
v4.4.2

C:\Users\maparekh>npm -v
2.15.0

C:\Users\maparekh>_
```

Figure 3: Node.js version



```
C:\Users\maparekh\demo-zetta>
C:\Users\maparekh\demo-zetta>
C:\Users\maparekh\demo-zetta>
C:\Users\maparekh\demo-zetta>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (demo-zetta)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to C:\Users\maparekh\demo-zetta\package.json:

{
  "name": "demo-zetta",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}


Is this ok? (yes)

C:\Users\maparekh\demo-zetta>
```

Figure 4: Creating the Node.js project

Apps: Apps that are used for different interactions between devices or to fetch and process some data are created in JavaScript. Apps can be created based on sensor streams or changes in devices. They can be used to track certain kinds of events that happen in systems.

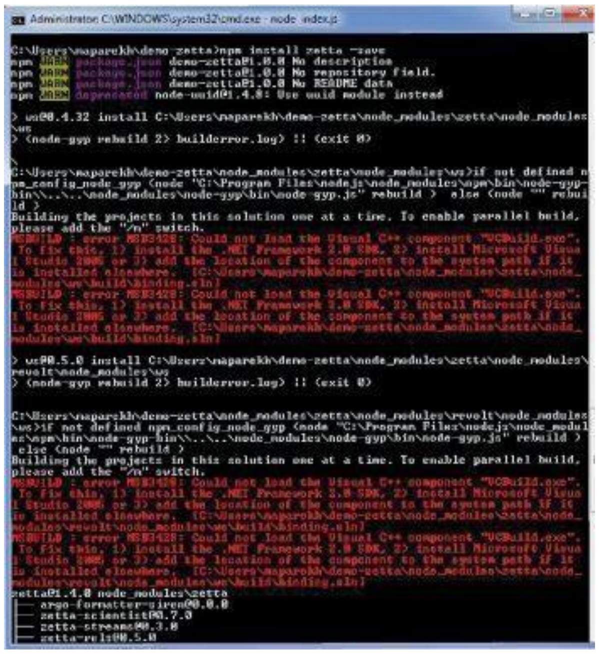
Zetta deployment: Now let us explore the deployment of Zetta.
1. The Zetta server runs on a hardware hub, which can be Raspberry Pi, Intel Edison or BeagleBone Black.
2. The hub is connected to devices, and they communicate via HTTP to the specific protocols used in the deployment.

3. Another similar server runs in the cloud, which has the same Node.js packages that are available on the Zetta server in the hub. Both the servers are connected.
4.
4. Zetta provides an API at the cloud endpoint so that consumers can use it. Hardware requirements: Zetta runs with approximately six connected devices per hub, which is the ideal scenario suggested by it. Hardware requirements are dependent upon the number of devices, the load of each device and the type of data flowing between them. The ideal minimum requirement is a 500MHz CPU, 500MB RAM and storage of 1GB-2GB. Zetta requires 500MB to be able to run. It supports all common operating systems with 32-bit and 64-bit versions.

Zetta installation and demo project: Now let's take a look at installing Zetta and a 'Hello world' version of the Zetta sample project. Before starting Zetta, here's a brief introduction to Node.js.

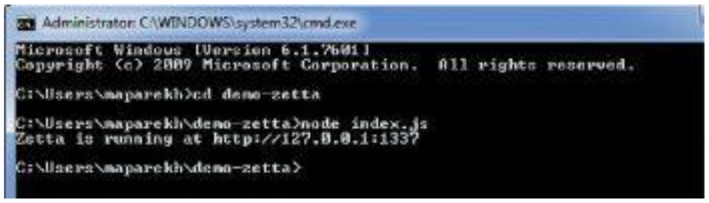
Node.js
this is built on Chrome's JavaScript runtime for building scalable and faster applications. It uses an event-driven, non-blocking IO model. It is popular and very efficient for real-time applications running across distributed systems. Basically, it's a server-side JavaScript.

More details about creating projects in Node.js are described in the following steps.



```
Administrator: C:\WINDOWS\system32\cmd.exe - node index.js

C:\Users\maparekh\demo-zetta>npm install zetta -save
npm WARN package.json demo-zetta@1.0.0 No description
npm WARN package.json demo-zetta@1.0.0 No repository field.
npm WARN package.json demo-zetta@1.0.0 No README data
npm WARN deprecated node-uuid@1.4.8: Use uuid module instead

> ws@0.4.32 install C:\Users\maparekh\demo-zetta\node_modules\zetta\node_modules
\ws
> (node-gyp rebuild 2> builderror.log) || (exit 0)

\
C:\Users\maparekh\demo-zetta\node_modules\zetta\node_modules\ws>if not defined n
pm_config_node_gyp (node "C:\Program Files\nodejs\node_modules\npm\bin\node-gyp-
bin\\..\..\node_modules\node-gyp\bin\node-gyp.js" rebuild )  else (node "" rebui
ld )
Building the projects in this solution one at a time. To enable parallel build,
please add the "/m" switch.
MSBUILD : error MSB3428: Could not load the Visual C++ component "VCBuild.exe".
 To fix this, 1) install the .NET Framework 2.0 SDK, 2) install Microsoft Visua
l Studio 2005 or 3) add the location of the component to the system path if it
is installed elsewhere.  [C:\Users\maparekh\demo-zetta\node_modules\zetta\node_
modules\ws\build\binding.sln]
MSBUILD : error MSB3428: Could not load the Visual C++ component "VCBuild.exe".
 To fix this, 1) install the .NET Framework 2.0 SDK, 2) install Microsoft Visua
l Studio 2005 or 3) add the location of the component to the system path if it
is installed elsewhere.  [C:\Users\maparekh\demo-zetta\node_modules\zetta\node_
modules\ws\build\binding.sln]

> ws@0.5.0 install C:\Users\maparekh\demo-zetta\node_modules\zetta\node_modules\
revolt\node_modules\ws
> (node-gyp rebuild 2> builderror.log) || (exit 0)


C:\Users\maparekh\demo-zetta\node_modules\zetta\node_modules\revolt\node_modules
\ws>if not defined npm_config_node_gyp (node "C:\Program Files\nodejs\node_modul
es\npm\bin\node-gyp-bin\\..\..\node_modules\node-gyp\bin\node-gyp.js" rebuild )
 else (node "" rebuild )
Building the projects in this solution one at a time. To enable parallel build,
please add the "/m" switch.
MSBUILD : error MSB3428: Could not load the Visual C++ component "VCBuild.exe".
 To fix this, 1) install the .NET Framework 2.0 SDK, 2) install Microsoft Visua
l Studio 2005 or 3) add the location of the component to the system path if it
is installed elsewhere.  [C:\Users\maparekh\demo-zetta\node_modules\zetta\node_
modules\revolt\node_modules\ws\build\binding.sln]
MSBUILD : error MSB3428: Could not load the Visual C++ component "VCBuild.exe".
 To fix this, 1) install the .NET Framework 2.0 SDK, 2) install Microsoft Visua
l Studio 2005 or 3) add the location of the component to the system path if it
is installed elsewhere.  [C:\Users\maparekh\demo-zetta\node_modules\zetta\node_
modules\revolt\node_modules\ws\build\binding.sln]
zetta@1.4.0 node_modules\zetta
├── argo-formatter-siren@0.0.0
├── zetta-scientist@0.7.0
├── zetta-streams@0.3.0
├── zetta-rels@0.5.0
```

Figure 5: Installing the Zetta Node.js module



```
Administrator: C:\WINDOWS\system32\cmd.exe

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\maparekh>cd demo-zetta

C:\Users\maparekh\demo-zetta>node index.js
Zetta is running at http://127.0.0.1:1337

C:\Users\maparekh\demo-zetta>
```

Figure 6: Node.js Zetta server status

Zetta installation: For Zetta installation, the first thing required is Node.js. As discussed, download the Node.js installer on to your system. This will install both Node.js and npm (node package manager). So we don't need to install anything separately; it's a complete package. We can verify the versions by using the commands shown in Figure 3.

Creating a Zetta project

1. Create a new directory to save the project, e.g., demo-zetta.
2. Now cd to that directory. Here, it's cd demo-zetta.
3. To create a new Node.js project, run the command given below:

npminit

4. You will be asked for basic information about the project. By default, it will choose the value if you press Enter. If you want to change the value, then do so and press Enterseveral times and finish the installation. Basically, it will create a package.json file, which contains meta data about the project and its dependencies.

5. Now we will install the Zetta Node.js module. Here, the -save option adds Zetta to thepackage.json dependencies list.

npm installzetta -save

After all these steps, we have a basic Zetta project, which contains a package.json file and a node_modules directory.

Next, let’s configure the Zetta server.

Zetta server configuration

We can install the Zetta server locally as a Zetta hub or in the cloud. We can link both the servers to access it from everywhere. Here we will install it locally for demo purposes.

1. Go to the demo-zetta directory.
2. Create a new file called index.js, and copy the code given below into it:

```
varzetta = require(‘zetta’);
zetta()
.name(‘Zetta Demo’)
.listen(1337, function(){
console.log(‘Zetta is running at http://127.0.0.1:1337’);
});
```

3. Now save and close the file.

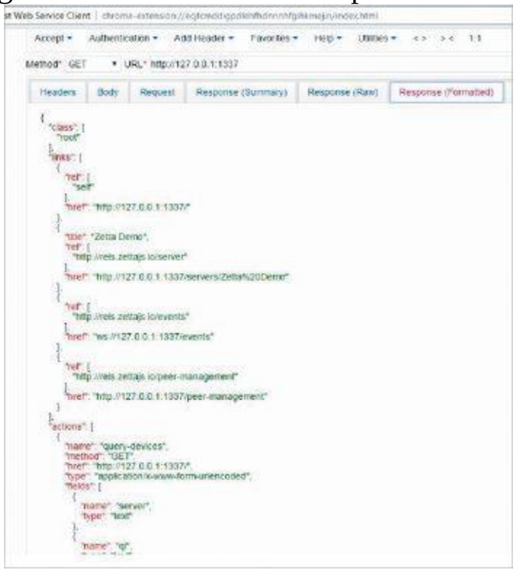
After performing the steps given above, we have configured a basic Zetta server hub.

Starting the server

In the demo-zetta directory, enter the command given below:

```
node index.js
```

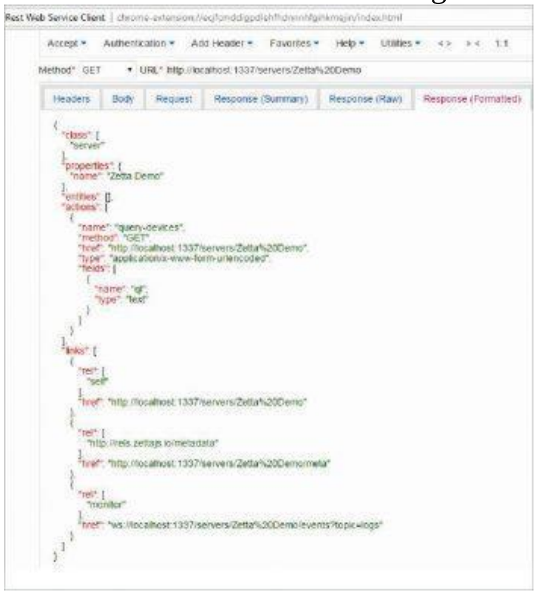
Figure 6 demonstrates the output of the above command. It shows the status of a running server.



Method* GET URL* http://127.0.0.1:1337

```
{
  "class": [
    "root"
  ],
  "links": [
    {
      "rel": [
        "self"
      ],
      "href": "http://127.0.0.1:1337/"
    },
    {
      "title": "Zetta Demo",
      "rel": [
        "http://rels.zettajs.io/server"
      ],
      "href": "http://127.0.0.1:1337/servers/Zetta%20Demo"
    },
    {
      "rel": [
        "http://rels.zettajs.io/events"
      ],
      "href": "ws://127.0.0.1:1337/events"
    },
    {
      "rel": [
        "http://rels.zettajs.io/peer-management"
      ],
      "href": "http://127.0.0.1:1337/peer-management"
    }
  ],
  "actions": [
    {
      "name": "query-devices",
      "method": "GET",
      "href": "http://127.0.0.1:1337/",
      "type": "application/x-www-form-urlencoded",
      "fields": [
        {
          "name": "server",
          "type": "text"
        },
        {
          "name": "ql",
```

Figure 7: The Zetta API call response



Method* GET URL* http://localhost:1337/servers/Zetta%20Demo

```
{
  "class": [
    "server"
  ],
  "properties": {
    "name": "Zetta Demo"
  },
  "entities": [],
  "actions": [
    {
      "name": "query-devices",
      "method": "GET",
      "href": "http://localhost:1337/servers/Zetta%20Demo",
      "type": "application/x-www-form-urlencoded",
      "fields": [
        {
          "name": "ql",
          "type": "text"
        }
      ]
    }
  ],
  "links": [
    {
      "rel": [
        "self"
      ],
      "href": "http://localhost:1337/servers/Zetta%20Demo"
    },
    {
      "rel": [
        "http://rels.zettajs.io/metadata"
      ],
      "href": "http://localhost:1337/servers/Zetta%20Demo/meta"
    },
    {
      "rel": [
        "monitor"
      ],
      "href": "ws://localhost:1337/servers/Zetta%20Demo/events?topic=logs"
    }
  ]
}
```

Figure 8: Demo Zetta server API response

Calling the Zetta API: Now, let’s call the Zetta API. We need to call the server’s root URL for that. Here we can use the curl command with http://127.0.0.1:1337 or any REST client tools. In the URL section of the REST client, enter http://127.0.0.1:1337 and submit the request. Now, in the response (formatted) section, you can see the response (see Figure 7). Check it for more information.

The Zetta server returns a JSON object that describes the root class of the API. The response demonstrates the current API state and links to resources given by the API. This is the basic API, which is not doing anything much as we don’t have devices attached. Once we add the devices, the API will show more information.

Zetta API is a built-in feature of Zetta, which automatically generates APIs for devices. We can deploy these APIs in the cloud, which allows any authorised user to communicate with these devices from anywhere.