UNIVERSITY of LIMERICK

OLLSCOIL LUIMNIGH

Theses: Docker Containers Deployed Using Bluemix

Name: Thomas Flynn

ID: 16117743

Supervisor: Dr. Sean McGrath

Course: Information & Network Security MEng

Year: 2017

Department: Electronic & Computer Engineering

1 Abstract

This theses focuses on using the DevOps concept of infrastructure as code to deploy Docker Containers on IBM Bluemix in order to provide the cloud infrastructure required to perform data acquisition from sensors for the purposes of real time data analytics.

Extensive research into the current state of the open-source cloud ecosystem was carried out as a result of this work. This research will provide future students with the knowledge needed to make an informed decision on what technology to choose when developing applications using state of the art open-source technology.

2 Acknowledgments

Firstly, I would like to express my sincere gratitude to my supervisor Sean McGrath for both his guidance in shaping the backbone of the theses and for keeping me on track when I got distracted by some fancy new technology. I am also grateful for being given the chance to lecture at the UL IoT Summer School which was an experience I thoroughly enjoyed.

Michael Barry, his industry experience and analytics expertise significantly helped me in both the design and implementation stages of the theses.

Pierre Le Chanu and Dorian Couespel for both their help during the Summer School as well as their collaboration with my theses.

The New Stack Podcast, they made this theses possible by both introducing me to Docker and informing me of the rapidly changing open source cloud landscape.

The ECE department and INS class for all their help throughout the year.

Lastly I’d like to thank my sisters Hilda and Kiera for all their love and support.

Table of Contents

Illustration Index

3 Introduction

Managing increasingly complex enterprise computing environments is one of the toughest challenges that organizations have to tackle in the era of cloud computing, big data and IoT. Containers are the new revolution in the cloud computing world that are helping tackle this problem, they are more lightweight than virtual machines, and can significantly decrease both the start up time of instances and the storage and processing overhead with respect to traditional virtual machines.

Infrastructure as code is a concept of describing your infrastructure and its configuration as a script or set of scripts which allows environments to be replicated in a much less error-prone manner. Infrastructure automation brings agility to both development and operations because any authorized team member can modify the scripts while applying good development practices.

IBM and Docker offer integrated container solutions that can meet the diverse needs of enterprises. Supporting the creation and deployment of multi-platform, multi-container workloads across hybrid infrastructures, IBM and Docker accelerate application delivery and enable application lifecycle management for Dockerized containers.[1]

This project is based on open source standards and is aimed at using DevOps concepts to provide the cloud infrastructure required to create smart campus applications related to geospatial tracking and real time analytics. The sensors transmit data periodically to the server using a lightweight IoT(Internet of Things) protocol known as MQTT (MQ Telemetry Transport). This data is then transformed and loaded into the appropriate database where it can be made available to analytic services and the public smart campus API.

This report will cover in detail exactly what objectives were achieved and help provide future students with sufficient documentation in order build on top of this project. During the course of the theses a significant amount of Development Operations aspects were researched such as GitHub repositories, continuous integration/deployment pipelines, Container orchestration and microservice frameworks such as Istio. This area of the project should reduce the operational complexity for future students who want to develop scalable Smart Campus Applications using microservices.

The repository for this theses can be found on Github at https://github.com/16117743/INS-Thesis-Documentation

3.1 The New Stack Podcast

The New Stack podcast proved to be a vital source of information for this theses. The podcast provides analysis and explanations about application development and management at scale.[2] Awareness of Docker containers came about through this podcast.

Illustration 1: The New Stack Logo

3.2 Internet of Things

The Internet of Things really comes together with the connection of sensors and machines. The true value that the Internet of Things creates is at the intersection of gathering data and leveraging it. All the data gathered by all the sensors in the world isn't worth anything if there isn't any infrastructure in place to analyze it in real time.[3]

Cloud-based applications are the key to using leveraged data. The Internet of Things doesn't function without cloud-based applications to interpret and transmit the data coming from all these sensors.

3.3 Cognitive IoT

Cognitive IoT is the use of cognitive computing technologies in combination with data generated by connected devices and the actions those devices can perform. Cognition involves three key elements:

- Understanding
- Reasoning
- Learning

In a computer, system *understanding* means being able to take in large volumes of both structured and unstructured data and derive meaning from it—that is, establish a model of concepts, entities and relationships. *Reasoning* means using that model to be able to derive answers or solve related problems without having the answers and solutions specifically programmed. And *learning* means being able to automatically infer new knowledge from data, which is a key component in understanding at scale.[4]

3.4 UL IoT Summer School

The Summer School is a three-week introduction to the core technology topics needed to support the Internet of Things. On completion, the student will be well placed to play a part in this emerging technology ecosystem.

Illustration 2: UL IoT Logo

At the end of the school, each group will present their concept project and how it relates to the core IoT concepts.[5]

The learning material that was prepared for the summer school students can be found in the appendix section of this report.

3.5 Open-source

The traditional open-source models are breaking down as commercial entities are quick to hire the most productive open-source contributors and commercialize parts of that IP. This is where foundations become important. The Cloud Foundry foundation has been used as the base of products from HP(Helion) and IBM(Bluemix). So you have the home of an initiative in direct competition with other commercial entities looking to leverage the same open-source project.[6]

Which is where the foundation comes in. The Cloud Foundry Foundation is a neutral party that can more readily bang together the various heads in order to do what is right for the project, as opposed to what is right for individual entities. This topic will be discussed further in section 5 of the report.

3.6 About DevOps

DevOps is a philosophy that brings together operations and development. It is an approach that promotes closer collaboration between lines of business, development and IT operations. It is an enterprise capability that enables the continuous delivery, continuous deployment and continuous monitoring of applications. This topic will be discussed further in section 6 of the report.

3.7 Objectives

The main objectives of this theses were:

- Provide the cloud infrastructure needed for storing sensor data
- Automate the build process to reduce operational complexity for the developer
- Design a geospatial tracking feature that utilizes the infrastructure
- Perform A/B testing on the feature
- Provide sufficient documentation for future students

3.8 Unfinished Work

By making the scope of the project so large, a lot was learned about time and project management, unfortunately though it leaves you with the feeling of a job half done. Half of the proposed infrastructure was successfully implemented, Redis and InfluxDB were not fully integrated. The design of a geospatial tracking feature was not realized so naturally it was impossible to perform A/B testing.

Certain aspects of the automation pipeline still needs work such as the linking of the Cloud Foundry Node-RED app to the container infrastructure and the automation of getting a public IP address for the HAProxy container when it gets recreated.

3.9 Report layout

Chapter 4: Discusses Docker and components of the Bluemix stack such as Node-Red, OpenWhisk and Cloudant.

Chapter 5: Discusses new open source foundations and their related projects.

Chapter 6: Discusses the various aspects of DevOps such as container orchestration.

Chapter 7: Discusses various DevOps vendor tools.

Chapter 8: Discusses various theoretical aspects of the theses such as the MQTT protocol.

Chapter 9: Discusses the project planning and work completed in each of the 3 semesters.

Chapter 10: Discusses the chosen components of the open source stack.

Chapter 11: This is the main section of the theses, it describes in detail the majority of the work undertaken.

Chapter 12: This section describes in detail the Jenkins CI/CD pipeline.

Chapter 13: Discusses the implementation and testing of the Mosca and HAProxy container.

Chapter 14: Discusses aspects of the Bluemix container service such as logging and monitoring.

Chapter 15: Discusses a use case for the theses and an architectural diagram of how it could be implemented.

Chapter 16: Discusses Node-Red and OpenWhisk functionality of the theses.

Chapter 17: Discusses 2 bleeding edge technologies, Istio and Iota.

Chapter 18: Discusses possible theses topics for future students.

Chapter 19: Conclusion

3.10 Acronyms

CI – continuous integration

CD – continuous delivery

IoT – Internet of things

MQTT – Message queue telemetry transport

HTTP – Hyper text transfer protocol

LB – Load balancer

HA- High availability

RAM – Random access memory

PaaS - Platform as a Service

SLA – Service level agreement

IP – Internet protocol

ETL – Extract Transform Load

DSL – Domain Specific Language

4 Docker and Bluemix

4.1 Docker

Released as an open source project in 2013, Docker has rapidly achieved widespread use. It was originally designed to access Linux kernel namespace features via LXC, but has switched to use its own “libcontainer” library that directly accesses the kernel. Docker also provides mechanisms to deploy applications into the containers with a “Dockerfile”.[7]

Illustration 3: Docker Logo

Docker supports an assortment of storage backends, most support copy-on-write semantics. Layered filesystems make it possible for multiple containers to run on the same underlying base image. When a container is started from an image, it uses the filesystem of that image. Modifications to the filesystem by a process in the container are written to a filesystem specific to that container, which makes it possible to run multiple containers independently on top of a single image. This copy-on-write mechanism (in contrast to a complete copy of the root file system for each container) makes the deployment of containers very fast, and reduces disk and RAM occupancy.[7]

The Dockerfile deployment process makes use of this by creating a new image from an existing base image, adding additional layers that describe differences between the base image and the fully deployed application. Docker’s popularity is due, at least in part, to an included system that handles revision control; this system makes it very easy to share docker images via public (such as the default Docker Hub repositories) or private repositories. Typically, public repositories are hosted for free, while the private repositories are not. Docker creates an abstraction for machine-specific setting settings such as networking, storage, logging, and Linux distribution.[7]

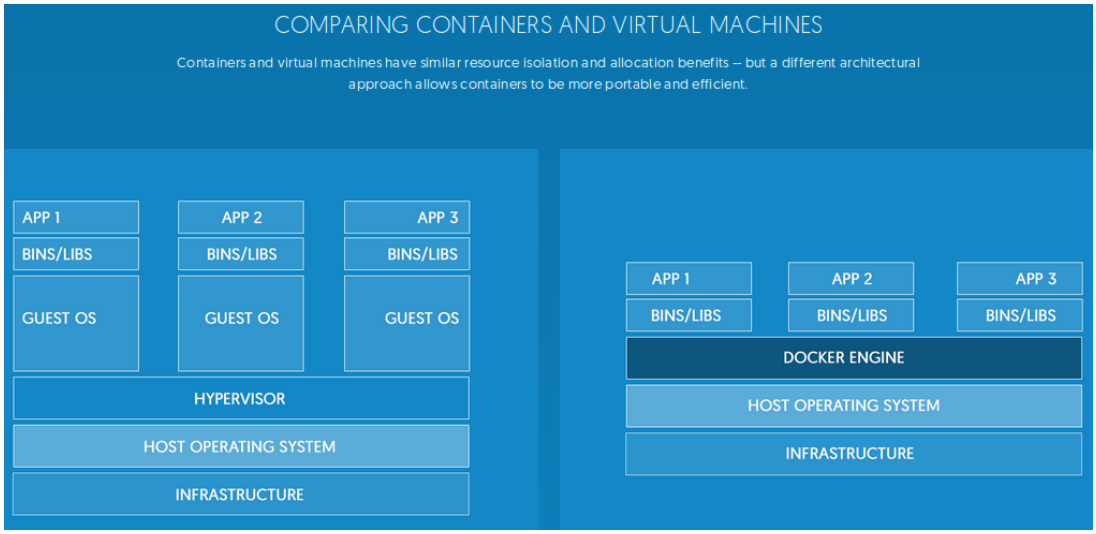


Illustration 4: Docker vs Virtual Machine

4.2 Bluemix Platform

Bluemix is an implementation of IBM's Open Cloud Architecture based on Cloud Foundry, an open source Platform as a Service (PaaS). Bluemix delivers enterprise-level services that can easily integrate with your cloud applications without you needing to know how to install or configure them. [1]

Illustration 5: Bluemix Logo

IBM and Docker offer integrated container solutions that can meet the diverse needs of enterprises. Supporting the creation and deployment of multi-platform, multi-container workloads across hybrid infrastructures, IBM and Docker accelerate application delivery and enable application lifecycle management for Dockerized containers.

4.2.1 Cloud Foundry

Cloud Foundry is an open source cloud platform as a service (PaaS) on which developers can build, deploy, run and scale applications on public and private cloud models. VMware originally created Cloud Foundry and it is now part of Pivotal Software.[8]

Illustration 6: Cloud Foundry

Cloud Foundry is licensed under Apache 2.0 and supports Java, Node.js, Go, PHP, Python and Ruby. The open source PaaS is highly customizable, allowing developers to code in multiple languages and frameworks. This eliminates the potential for vendor lock-in, which is a common concern with PaaS.

4.3 IBM IoT Platform

4.3.1 IBM Watson

IBM Watson is a technology platform that uses natural language processing and machine learning to reveal insights from large amounts of unstructured data.[9]

Illustration 7: IBM Watson Logo

4.3.2 Node Red

Node-RED is a tool for wiring together the Internet of Things in new and interesting ways, including hardware devices, APIs, and online services. It is built on top of Node.js and takes advantage of the huge node module ecosystem to provide a tool that is capable of integrating many different systems. Its lightweight nature makes it ideal to run at the edge of the network.[10]

Illustration 8: Node-RED Logo

4.4 IBM OpenWhisk

4.4.1 About Serverless Computing

Serverless computing refers to a model where the existence of servers is simply hidden from developers. I.e. that even though servers still exist developers are relieved from the need to care about their operation. They are relieved from the need to worry about low-level infrastructural and operational details such as scalability, high-availability, infrastructure-security, and so forth. Hence, serverless computing is essentially about reducing maintenance efforts to allow developers to quickly focus on developing value-adding code.[11]

Serverless computing encourages and simplifies developing microservice-oriented solutions in order to decompose complex applications into small and independent modules that can be easily exchanged.

Serverless computing does not refer to a specific technology; instead if refers to the concepts underlying the model described prior. Nevertheless some promising solutions have recently emerged easing development approaches that follow the serverless model such as OpenWhisk.

4.4.2 About OpenWhisk

Cloud is a game-changer for app development, and serverless computing is revolutionizing developers' access to some of the most powerful services cloud has to offer, including cognitive intelligence, data analytics and Internet of Things.[12]

Illustration 9: OpenWhisk Logo

In traditional cloud computing models, developers often have to maintain their infrastructure and worry about when and how fast to scale, as well as potential resiliency issues if they are deploying apps and features in multiple regions. They are also charged for computing power even when an app is idling —a particularly painful point for startups and small companies with limited resources and early growth applications.

Serverless computing, which many are hailing as the next era of cloud computing, relieves many of these hassles by abstracting away infrastructure, running code and scaling on-demand. For developers, serverless platforms with a strong cognitive stack, such as IBM Bluemix OpenWhisk, gives them unprecedented access to powerful services such as Watson APIs, the Watson IoT Platform and weather intelligence.

As one of the few serverless platforms built on open standards, OpenWhisk acts as the invisible force within apps, binding together relevant events, actions and triggers. As data continues to grow and proliferate across all industries, serverless is certainly primed to become a standard for resourcefulness, scalability and connecting into the power of cognitive and IoT tools running on the cloud.

4.5 Cloudant

Cloudant is an IBM software product, which is primarily delivered as a cloud-based service. Cloudant is a non-relational, distributed database service of the same name. Cloudant is based on the Apache-backed CouchDB project and the open source BigCouch project.[13]

Illustration 10: Cloudant Logo

Cloudant's service provides integrated data management, search, and analytics engine designed for web applications. Cloudant scales databases on the CouchDB framework and provides hosting, administrative tools, analytics and commercial support for CouchDB and BigCouch.

4.5.1 Cloudant Geospatial

Cloudant Geospatial combines the advanced geospatial queries of a Geographic Information System (GIS) with the flexibility and scalability of Cloudant's database-as-a-service (DBaaS) capabilities.

Cloudant Geo:

- Enables web and mobile developers to enhance their applications using geospatial operations that go beyond simple bounding boxes.
- Integrates with existing GIS applications, so that they can scale to accommodate different data sizes, concurrent users, and multiple locations.
- Provides a NoSQL capability for GIS applications, so that large streams of data can be acquired from devices, sensors, and satellites. This data can then be stored, processed, and syndicated across other web applications.

4.5.2 Data flexibility: JSON data store

The majority of requests and responses to and from Cloudant use the JavaScript Object Notation (JSON) for formatting the content and structure of the data and responses.

JSON is used because it is the simplest and easiest solution for working with data using a web browser. This is because JSON structures can be evaluated and used as JavaScript objects within the web browser environment. JSON also integrates with the server-side JavaScript used within Cloudant.

Cloudant's RESTful API makes every document in your database accessible as JSON. It is also compatible with Apache CouchDB™, enabling you to access an abundance of language libraries and tools. Schema flexibility makes Cloudant an excellent fit for multi-structured data, unstructured data and fast-changing data models.

5 Open Source

Over the past twenty years, open source software has emerged as a major force throughout our information-driven economy. Open source practices have drastically changed the way software is developed. And now there is an unending array of tools and Internet-based services that make it possible, if not downright easy, for nearly everyone to create and distribute useful, important software.[14]

Open source is based on freedom. That freedom includes access to the source code, freedom to collaborate, and ultimately, the freedom to innovate. In open source, no one person or company owns a project. Open source is a philosophy and a movement, and what makes open source thrive is the community that grows up around it.

5.1 IBM

IBM has a long history of working in open technology. IBM's strong support for Linux with patent pledges and significant technical resources and investment in the face of some of the legal uncertainty surrounding its use brought about a change in posture toward open source for many enterprises. Under the right circumstances, open source can be a compelling and trusted alternative to proprietary software.[15]

IBM's commitment and contribution to open source is unrivaled in the industry. They have worked hard over the years to establish a solid and respected reputation in open source circles. They work hard to invest in ensuring interoperability, portability, and various other enterprise characteristics by investing in the community and helping to shape programs that can help deliver those characteristics. They value and work toward open governance because they feel that this is the best way to ensure the long-term success and viability of open source projects. Thousands of IBM developers are working every day in open source projects that matter.

IBM evaluates open source projects by looking closely at five aspects of the project:

- **Responsible licensing**—They look to understand the open source license that is associated with the technology.
- **Accessible commit process**—They seek to ensure that there is a clearly defined process for making contributions that welcomes outside contributors.
- **Diverse ecosystem**—They confirm that there are multiple vendors and ISVs that are delivering offerings based on the technology.
- **Participative community**—They require that there be a process for contributors to grow their technical eminence in the community.
- **Open governance**—They evaluate the governance model to determine whether it is *truly* open.

5.2 Node.js Foundation

Illustration 11: Node.js Logo

In 2014, the Node.js community went through a dark period where the community developing the technology suffered a schism and forked into two independent projects: Node.js and io.js. Because of IBM's reputation in the open communities, they were asked to help resolve the differences. IBM worked within both communities to bring the factions together under a single open governance model at the Node.js Foundation under the Linux Foundation's collaborative projects program. The community has reconciled, and it is on a steady cadence of long-term and experimental releases. IBM helped to heal the fork and bring the community back to a single, unified code base. IBM is also one of the leading contributors to the Node.js community.[15]

5.3 Strongloop

Illustration 12: StrongLoop Logo

StrongLoop began in 2013 offering an open-source enterprise version of Node.js. Acquired by IBM in 2015, the StrongLoop team continues to build LoopBack, the open-source Node.js API Framework. IBM continues to contribute and support the StrongLoop community through these projects that provide key technologies for the API economy.[16]

5.3.1 LoopBack project

LoopBack is a highly-extensible, open-source Node.js framework that enables you to:

- Create dynamic end-to-end REST APIs with little or no coding.
- Access data from major relational databases, MongoDB, SOAP, and REST APIs.
- Incorporate model relationships and access controls for complex APIs.
- Use separable components for file storage, third-party login, and OAuth 2.0.

5.4 Open API

Illustration 13: Open API Logo

The Open API Initiative (OAI) was created by a consortium of forward-looking industry experts who recognize the immense value of standardizing on how REST APIs are described. SmartBear Software is donating the Swagger Specification directly to the OAI as the basis of this Open Specification. APIs form the connecting glue between modern applications. Nearly every application uses APIs to connect with corporate data sources, third party data services or other applications. Creating an open description format for API services that is vendor neutral, portable and open is critical to accelerating the vision of a truly connected world.[17]

5.5 Open Container Initiative

The Open Container Initiative (OCI) is a lightweight, open governance structure (project), formed under the auspices of the Linux Foundation, for the express purpose of creating open industry standards around container formats and runtime. The OCI was launched on June 22nd 2015.[18]

Illustration 14: Open Container Initiative Logo

The project gained wide attention because of the promise of portability, agility and interoperability in a broad range of infrastructures.

5.5.1 runC

RunC, a lightweight universal container runtime, is a command-line tool for spawning and running containers according to the Open Container Initiative (OCI) specification. The governance umbrella created by Docker, Google, IBM, Microsoft, Red Hat, and many other partners to create a common and standardized runtime specification has a readable spec document for the runtime elements of a container, and a usable implementation based on code contributed to the OCI by Docker. It includes *libcontainer*, the original lower-layer library interface originally used in the Docker engine, to set up the operating system constructs that we call a container.[19]

Illustration 15: runC logo

5.5.2 Benefits of using runC

Even before the OCI and runC existed, many core Docker engine developers used a runC-precursor, nsinit, that allowed a simplified entry point into running and debugging low-level container features without the overhead of the entire Docker daemon interface. Now that runC exists, this is definitely one continuing use case, especially for someone potentially exposing a new Linux isolation feature. For example, the checkpoint/restore capability using the Linux Checkpoint/Restore In Userspace (CRIU) project first was made available via runC, and just now is being prepared for addition to the Docker daemon at the layer above runC.[19]

5.5.3 Diego and Garden

Cloud Foundry uses the Diego architecture to manage application containers. Diego components assume application scheduling and management responsibility from the Cloud Controller.

Garden is the containerisation layer used by Diego. Garden provides a platform-neutral, lightweight container abstraction that can be backed by multiple backends (most importantly, a Linux backend and a Windows backend). Garden enables Diego to support buildpack apps and Docker apps.

Using runC as the garden backend has two major advantages. Firstly it lets us reuse some awesome code and be part of the Open Container community. Secondly it means Cloud Foundry applications will be using not only the same kernel primitives as Docker apps, but also the exact same runtime container engine.[20]

5.6 Cloud Native Computing Foundation

Illustration 16: Cloud Native Computing Foundation Logo

The Cloud Native Computing Foundation (CNCF) is a nonprofit organization committed to advancing the development of cloud native technology and services by creating a new set of common container technologies informed by technical merit and end user value, and inspired by Internet-scale computing. As a shared industry effort, CNCF members represent container and cloud technologies, online services, IT services and end user organizations focused on promoting and advancing the state of cloud native computing for the enterprise.[21]

5.6.1 Kubernetes

Illustration 17: Kubernetes Logo

Since the launch of Kubernetes in 2014, IBM has been a contributor to its development, which was built by the open community. By pairing a Docker-powered engine with the simple management capabilities of Kubernetes, Bluemix developers can access a highly usable interface and dashboard to easily write code within a container and quickly deploy it to multiple apps. This clear view also enables developers to see where and when their code is running at any given point in time.[22]

IBM released a beta of Kubernetes support in the IBM Bluemix Container Service on March 19, 2017. The IBM Bluemix Container Service, now combines Docker and Kubernetes to deliver powerful tools, an intuitive user experience, and built-in security and isolation to enable rapid delivery of applications all while leveraging Cloud Services including cognitive capabilities from Watson.[23]

Using a single Kubernetes dashboard you can manage security compliance throughout your DevOps pipeline by automatically scanning both Docker images and live containers for known vulnerabilities and the presence of malware. Verify appropriate container settings and application configurations. Review risk ratings and match container service policies to those of your organization.

5.6.2 Bluemix Container Service

Illustration 18: Bluemix Container Service Logo

IBM Container Service provides a native Kubernetes experience that's secure, easy to use, removes the distractions related to managing your clusters, and extends the power of your apps with Watson and other cloud services, binding them with Kubernetes Secrets. It applies pervasive security intelligence to your entire DevOps pipeline by automatically scanning Docker images and live containers for vulnerabilities and malware, leveraging IBM's X-Force Exchange.[24]

5.6.3 Kubernetes Architecture

The master is responsible for exposing the application program interface (API), scheduling the deployments and managing the overall cluster. Each node runs a container runtime, such as Docker or rkt, along with an agent that communicates with the master. The node also runs additional components for logging, monitoring, service discovery and optional add-ons. Nodes are the workhorses of a Kubernetes cluster. They expose compute, networking and storage resources to applications. Nodes can be virtual machines (VMs) running in a cloud or bare metal servers running within the data center.[25]

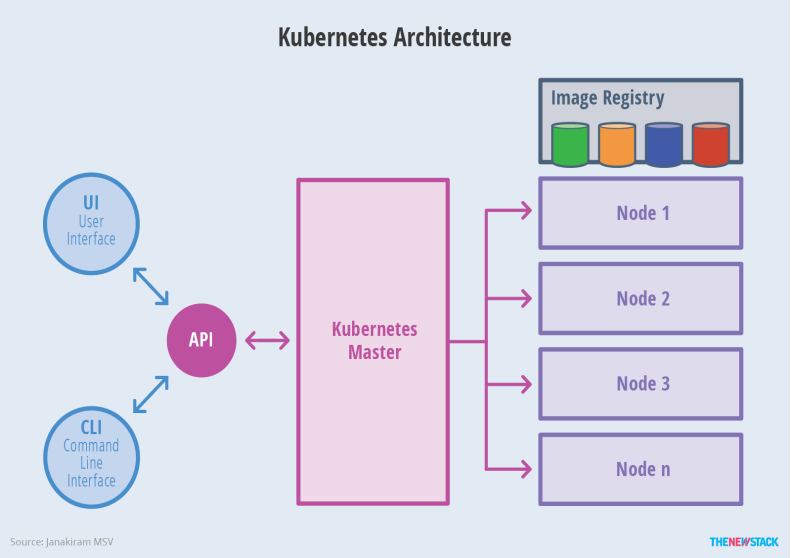


Illustration 19: Kubernetes Architecture

A pod is a collection of one or more containers. The pod serves as Kubernetes' core unit of management. Pods act as the logical boundary for containers sharing the same context and resources. The grouping mechanism of pods make up for the differences between containerization and virtualization by making it possible to run multiple dependent processes together. At runtime, pods can be scaled by creating replica sets, which ensure that the deployment always runs the desired number of pods.

Replica sets deliver the required scale and availability by maintaining a pre-defined set of pods at all times. A single pod or a replica set can be exposed to the internal or external consumers via services. Services enable the discovery of pods by associating a set of pods to a specific criterion. Pods are associated to services through key-value pairs called labels and selectors. Any new pod with labels that match the selector will automatically be discovered by the service. This architecture provides a flexible, loosely-coupled mechanism for service discovery.

StatefulSets are valuable for applications that require one or more of the following.

- Stable, unique network identifiers.
- Stable, persistent storage.
- Ordered, graceful deployment and scaling.
- Ordered, graceful deletion and termination.

6 DevOps

The wall between Dev and Ops has been removed, it has been replaced by a pipeline of continuous delivery. Removing the wall between Dev and Ops is the most important shift of all that will come as container adoption becomes more widespread, accelerated by open source development. The software required to manage clusters will be the orchestration platforms, working on data planes that make container-based clusters fully mobile. The infrastructure itself becomes centered on the application.[26]

6.1 Cloud-Native

The concept of cloud native has set the stage for how organizations develop a programmable infrastructure, and the complexity is astounding. There are a number of overlapping domains, especially when we think about container management, said Chris Ferris, distinguished engineer and chief technical officer of open technology at IBM Cloud in an interview with The New Stack.

This overlap may lead to different integrations that allow customers to build from components that suit their needs, and potentially an architecture that everybody can agree upon. What is needed is fault tolerance, self healing, easy roll-outs, versioning, and the ability to easily scale up or down. Containers should run on a cloud service or your own hardware — and have them just run at whatever scale is necessary, never going down and never paging the Ops team. This is what people call orchestration.[26]

6.2 Orchestration and Automation

“I think largely, when we think about automation, you think about writing the scripts that are integrated into a platform, like Chef, Jenkins or Ansible ...that is actually driving the actual behavior; and we think about orchestration as the platforms themselves that are providing that facility to be able to orchestrate the order in which things are going. That’s the orchestration. The automation itself is just the actual execution of the point-in-time script.”- Chris Ferris [26]

6.3 Conclusion

From automation and orchestration to scheduling, cluster management, service discovery, security and more, it all adds up to a change in how we think about automation and orchestration. Embracing this change, and learning to focus on developing best practices, is a larger journey for the entire container ecosystem; a journey fueled by open source, the need for for connectivity, passionate communities of users, and economic factors in the business models of vendors.[26]

6.4 Continuous Integration

Continuous Integration *(CI)* is a strategy for how a developer can integrate code to the mainline continuously - as opposed to frequently.[27]

CI was created for agile development. It organizes development into functional user stories. These user stories are put into smaller groups of work, sprints. The idea of continuous integration is to find issues quickly, giving each developer feedback on their work and Test Driven Development (TDD) evaluates that work quickly. With TDD, you build the test and then develop functionality until the code passes the test. Each time, when you make new addition to the code, its test can be added to the suite of tests that are run when you build the integrated work. This ensures that new additions don't break the functioning work that came before them, and developers whose code does in fact "break the build" can be notified quickly.[28]

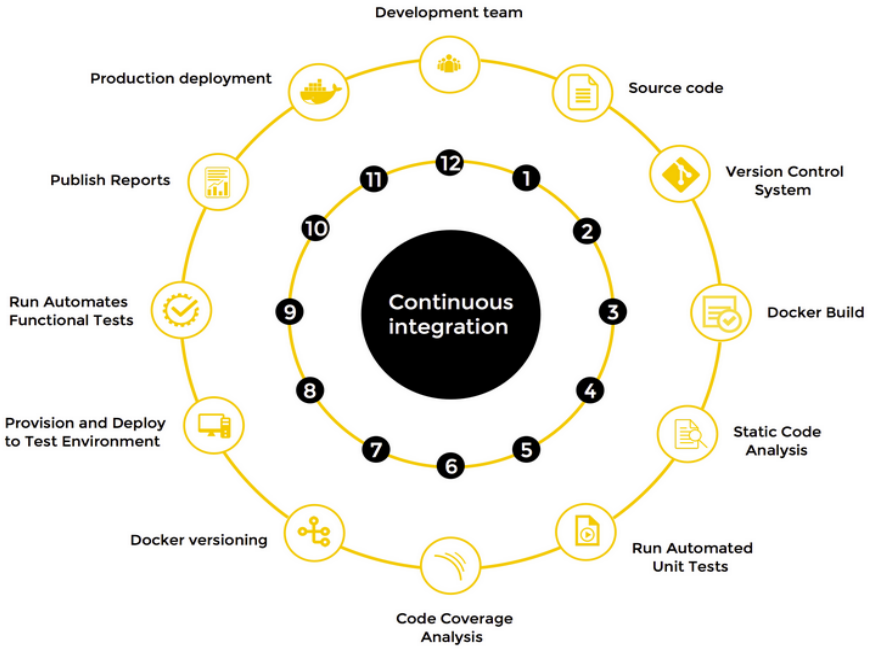


Illustration 20: Continuous Integration

6.5 Continuous Delivery

Continuous Delivery (CD) is a core technique that stems from Continuous Integration (CI), where as per business needs, quality software is deployed frequently and predictably to Production in an automated fashion. This improves feedback and reduces shelf-time of new ideas, and thereby improves the sustainability of businesses. Continuous Delivery includes Continuous Deployment and Continuous Testing.[28]

7 DevOps Vendor tools

7.1 Chef

Chef has been around since 2009. It was influenced by Puppet and CFEngine. Chef supports multiple platforms including Ubuntu, Debian, RHEL/CentOS, Fedora, Mac OS X, Windows 7, and Windows Server. It is often described as easier to use — particularly for Ruby developers, because everything in Chef is defined as a Ruby script and follows a model that developers are used to working in. Chef has a passionate user base, and the Chef community is rapidly growing while developing cookbooks for others to use.[29]

Illustration 21: Chef Logo

7.1.1 How it works

In Chef, three core components interact with one another — *Chef server*, *nodes*, and *Chef workstation*. Chef runs *cookbooks*, which consist of *recipes* that perform automated steps — called *actions* — on nodes, such as installing and configuring software or adding files. The Chef server contains configuration data for managing multiple nodes. The configuration files and resources stored on the Chef server are pulled down by nodes when requested. Examples of *resources* include file, package, cron, and execute.

Users interact with the Chef server using Chef's command-line interface, called *Knife*. Nodes can have one or more *roles*. A role defines *attributes* (node-specific settings) and recipes for a node and can apply them across multiple nodes. Recipes can run other recipes. The recipes in a node, called a *run list*, are executed in the order they are listed. A Chef workstation is an instance with a local Chef repository and Knife installed on it.[29]

7.2 Puppet

Puppet has been in use since 2005. Many organizations, including Google, Twitter, Oracle, and Rackspace, use it to manage their infrastructure. Puppet, which tends to require a steeper learning curve than Chef, supports a variety of Windows and *nix environments. Puppet has a large and active user community. It has been used in thousands of organizations with installations running tens of thousands of instances.[29]

Illustration 22: Puppet Logo

7.2.1 How it works

Puppet uses the concept of a master server — called the *Puppet master* — which centralizes the configuration among *nodes* and groups them together based on type. For example, if you had a set of web servers that were all running Tomcat with a Jenkins WAR, you'd group them together on the Puppet master. The *Puppet agent* runs as a daemon on systems. This enables you to deploy infrastructure changes to multiple nodes simultaneously. It functions the same way as a deployment manager, but instead of deploying applications, it deploys infrastructure changes.[29]

7.3 Ansible

Ansible is a radically simple IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs.

ANSIBLE

Illustration 23: Ansible Logo

Designed for multi-tier deployments since day one, Ansible models your IT infrastructure by describing how all of your systems inter-relate, rather than just managing one system at a time.

It uses no agents and no additional custom security infrastructure, so it's easy to deploy - and most importantly, it uses a very simple language (YAML, in the form of Ansible Playbooks) that allow you to describe your automation jobs in a way that approaches plain English.[30]

7.3.1 Architecture

Ansible works by connecting to your nodes and pushing out small programs, called "Ansible modules" to them. These programs are written to be resource models of the desired state of the system. Ansible then executes these modules (over SSH by default), and removes them when finished.

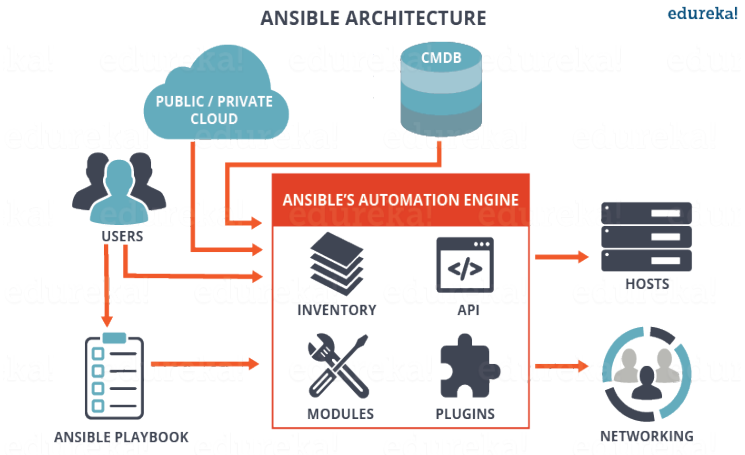


Illustration 24: Ansible Architecture

Your library of modules can reside on any machine, and there are no servers, daemons, or databases required. Typically you'll work with your favorite terminal program, a text editor, and probably a version control system to keep track of changes to your content.[30]

7.3.2 SSH Keys

Passwords are supported, but SSH keys with ssh-agent are one of the best ways to use Ansible. Though if you want to use Kerberos, that's good too. Lots of options! Root logins are not required, you can login as any user, and then su or sudo to any user.

Ansible's "authorized_key" module is a great way to use ansible to control what machines can access what hosts. Other options, like kerberos or identity management systems, can also be used.[30]

7.3.3 Playbooks

Playbooks can finely orchestrate multiple slices of your infrastructure topology, with very detailed control over how many machines to tackle at a time. This is where Ansible starts to get most interesting.

Ansible's approach to orchestration is one of finely-tuned simplicity, as we believe your automation code should make perfect sense to you years down the road and there should be very little to remember about special syntax or features.[30]

7.4 Jenkins

7.4.1 About

Jenkins is an open source automation server. With Jenkins, organizations can accelerate the software development process through automation. Jenkins manages and controls development lifecycle processes of all kinds, including build, document, test, package, stage, deployment, static analysis and many more.[31]

Illustration 25: Jenkins Logo

You can set up Jenkins to watch for any code changes in places like SVN and Git, automatically do a build with tools like Ant and Maven, initiate tests and then take actions like rolling back or rolling forward in production.

Thanks to its extensibility and a vibrant, active community, Jenkins has grown significantly. Today the Jenkins community offers more than 1,300 plugins that allow Jenkins to integrate with almost any popular technology. It is by far the most dominant automation server and, as of December 2016, there are more than 133,000 active installations and an estimated 1+ million users around the world.

7.4.2 Architecture

Jenkins is primarily a set of Java classes that model the concepts of a build system in a straight-forward fashion (and if you are using Jenkins, you've seen most of those already). There are classes like Project, Build, that represents what the name says. The root of this object model is Hudson, and all the other model objects are reachable from here.

Then there are interfaces and classes that model code that performs a part of a build, such as SCM for accessing source code control system, Ant for performing an Ant-based build, Mailer for sending out e-mail notifications.[31]

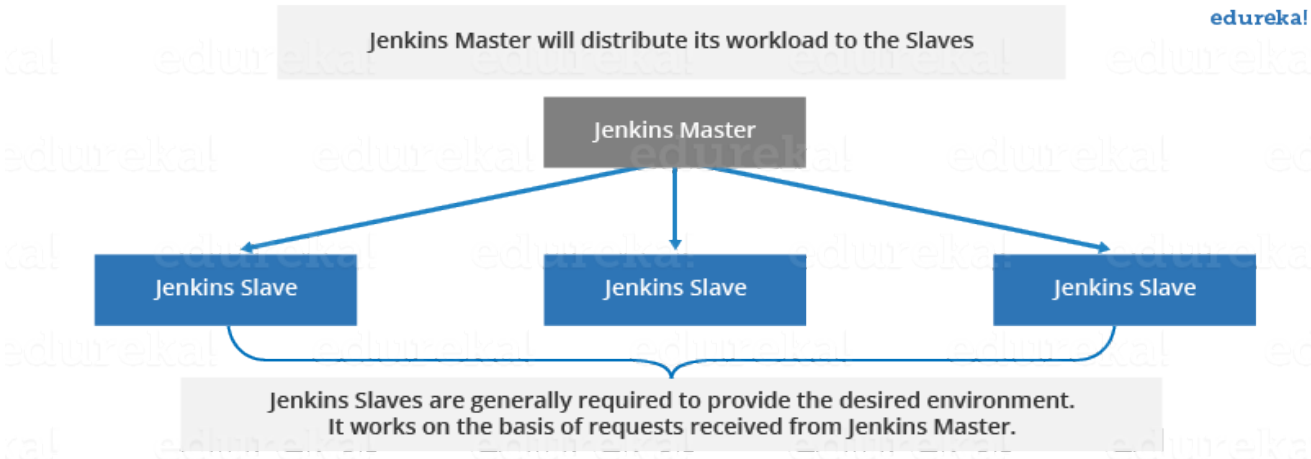


Illustration 26: Jenkins Architecture

7.4.3 Jenkins Master

The main Jenkins server is the Master. The Master's job is to handle:

- Scheduling build jobs.
- Dispatching builds to the slaves for the actual execution.
- Monitor the slaves (possibly taking them online and offline as required).
- Recording and presenting the build results.
- A Master instance of Jenkins can also execute build jobs directly.

7.4.4 Jenkins Slave

A Slave is a Java executable that runs on a remote machine. Following are the characteristics of Jenkins Slaves:

- It hears requests from the Jenkins Master instance.
- Slaves can run on a variety of operating systems.
- The job of a Slave is to do as they are told to, which involves executing build jobs dispatched by the Master.
- You can configure a project to always run on a particular Slave machine, or a particular type of Slave machine, or simply let Jenkins pick the next available Slave.

7.4.5 Jenkins Build Pipeline

It is used to know which task Jenkins is currently executing. Often several different changes are made by several developers at once, so it is useful to know which change is getting tested or which change is sitting in the queue or which build is broken. This is where pipeline comes into picture. The Jenkins Pipeline gives you an overview of where tests are up to. In build pipeline the build as a whole is broken down into sections, such as the unit test, acceptance test, packaging, reporting and deployment phases. The pipeline phases can be executed in series or parallel, and if one phase is successful, it automatically moves on to the next phase (hence the relevance of the name "pipeline") .[32]

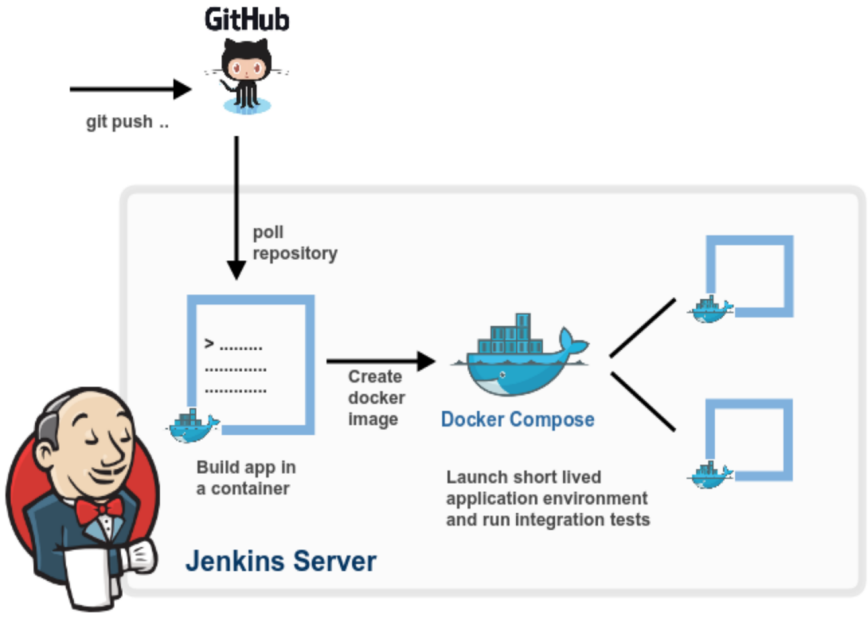


Illustration 27: Jenkins and Compose

7.5 Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a Compose file to configure your application's services. Then, using a single command, you create and start all the services from your configuration.[33]

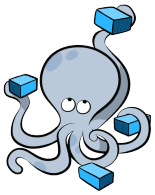


Illustration 28: Docker Compose Logo

Compose is great for development, testing, and staging environments, as well as CI workflows. Using Compose is basically a three-step process.

1. Define your app's environment with a `Dockerfile` so it can be reproduced anywhere.
2. Define the services that make up your app in `docker-compose.yml` so they can be run together in an isolated environment.
3. Lastly, run `docker-compose up` and Compose will start and run your entire app.

The real advantage of Compose is for applications that revolve around a single-purpose server that could easily scale out if architectural complexity is not a requirement. Development environments, which tend to use an all-in-one method of operation, fit well into this requirement. The community's reception of Compose has been notably positive, but the practicality of its usage and the lack of ability to create a long-term vision around it tend to minimize the actual legitimacy of adopting it as a container orchestration technology.

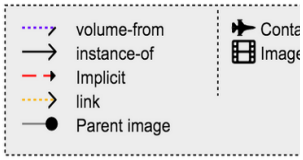


Illustration 29: Legend

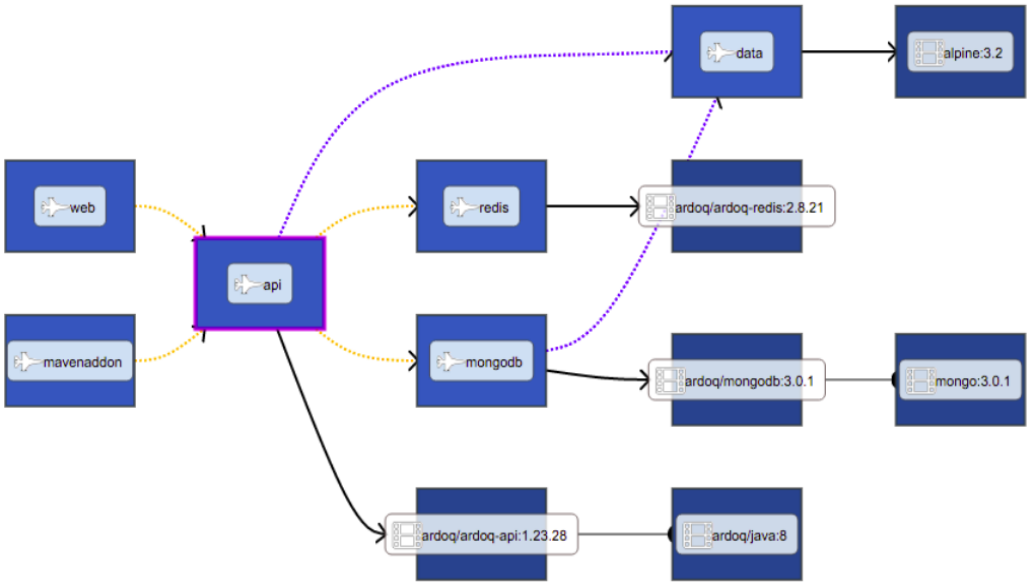


Illustration 30: Visualize Docker Compose

7.6 Amalgam8

IBM has made available an open source project named Amalgam8 which takes the tedium out of microservice management, enabling faster development, more control, and better resiliency of microservices without impacting existing implementation code. It provides advanced DevOps capabilities such as systematic resiliency testing, red/black deployment, and canary testing necessary for rapid experimentations and insight.[34]

Illustration 31: Amalgam8 Logo

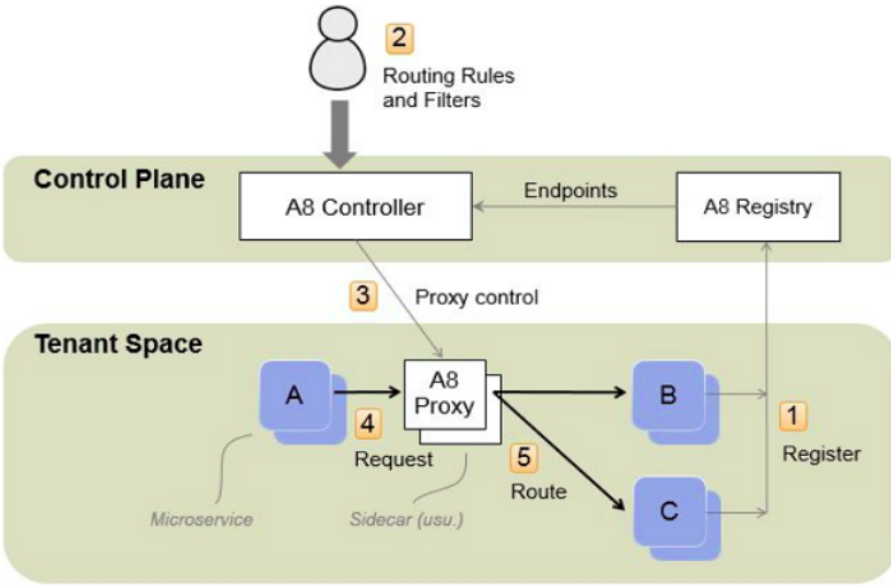


Illustration 32: Amalgam8 Architecture

At the heart of Amalgam8 are two multi-tenanted services:

- ***Registry –*** A high-performance service registry that provides a centralized view of all the microservices in an application, regardless of where they are actually running.
- ***Controller –*** A tool that monitors the Registry and provides a REST API for registering routing and other microservice control-rules, which it uses to generate and send control information to proxy servers running within the application

Applications run as tenants of these two servers. They register their services in the Registry and use the Controller to manage proxies, usually running as sidecars of the microservices.

- Microservice instances are registered in the *A8 Registry*.
- An *Administrator* specifies routing rules and filters (such as version rules and test delays) to control traffic flow between microservices.
- An *A8 Controller* monitors the A8 Registry and administrator input, and then generates control information that is sent to the A8 Proxies.
- Requests to microservices are made through an *A8 Proxy* (usually a client-side sidecar of another microservice).
- The A8 Proxy forwards requests to an appropriate microservice, depending on the request path and headers and the configuration specified by the controller.

8 Theory

8.1 MQTT

The protocol is intended for use on wireless and low-bandwidth networks. A mobile application that uses MQTT sends and receives messages by calling an MQTT library. The messages are exchanged through an MQTT messaging server. The MQTT client and server handle the complexities of delivering messages reliably for the mobile app and keep the cost of network management small.[35]

The MQTT protocol is lightweight in the sense that clients are small, and it uses network bandwidth efficiently. The MQTT protocol supports assured delivery and fire-and-forget transfers. In the protocol, message delivery is decoupled from the application. The extent of decoupling in an application depends on the way an MQTT client and MQTT server are written. Decoupled delivery frees up an application from any server connection, and from waiting for messages.

The MQTT V3.1 protocol is published; see MQTT V3.1 Protocol Specification. The specification identifies a number of distinctive features about the protocol:

- It is a publish/subscribe protocol.
  - In addition to providing one-to-many message distribution, publish/subscribe decouples applications. Both features are useful in applications that have many clients.
- It is not dependent in any way on the message content.
- It runs over TCP/IP, which provides basic network connectivity.

8.1.1 Quality of service

At most once (Qos 0)

Messages are delivered according to the best efforts of the underlying Internet Protocol network. Message loss might occur. Use this quality of service with communicating ambient sensor data, for example. It does not matter if an individual reading is lost, if the next one is published soon after.

At least once (Qos 1)

Messages are assured to arrive but duplicates might occur.

Exactly once (Qos 2)

Messages are assured to arrive exactly once.

Use this quality of service with billing systems, for example. Duplicate or lost messages might lead to inconvenience or imposing incorrect charges.

8.1.2 Client

When talking about a client it almost always means an MQTT client. This includes publisher or subscribers, both of them label an MQTT client that is only doing publishing or subscribing. (In general a MQTT client can be both a publisher & subscriber at the same time). A MQTT client is any device from a micro controller up to a full fledged server, that has a MQTT library running and is connecting to an MQTT broker over any kind of network. This could be a really small and resource constrained device, that is connected over a wireless network and has a library strapped to the minimum or a typical computer running a graphical MQTT client for testing purposes, basically any device that has a TCP/IP stack and speaks MQTT over it.[36]

8.1.3 Broker

The counterpart to a MQTT client is the MQTT broker, which is the heart of any publish/subscribe protocol. Depending on the concrete implementation, a broker can handle up to thousands of concurrently connected MQTT clients. The broker is primarily responsible for receiving all messages, filtering them, decide who is interested in it and then sending the message to all subscribed clients**.** It also holds the session of all persisted clients including subscriptions and missed messages. Another responsibility of the broker is the authentication and authorization of clients. And at most of the times a broker is also extensible, which allows to easily integrate custom authentication, authorization and integration into backend systems.[36]

8.1.4 MQTT Connection

The MQTT protocol is based on top of TCP/IP and both client and broker need to have a TCP/IP stack. The MQTT connection itself is always between one client and the broker, no client is connected to another client directly. The connection is initiated through a client sending a CONNECT message to the broker. The broker response with a CONNACK and a status code. Once the connection is established, the broker will keep it open as long as the client doesn't send a disconnect command or it looses the connection.[36]

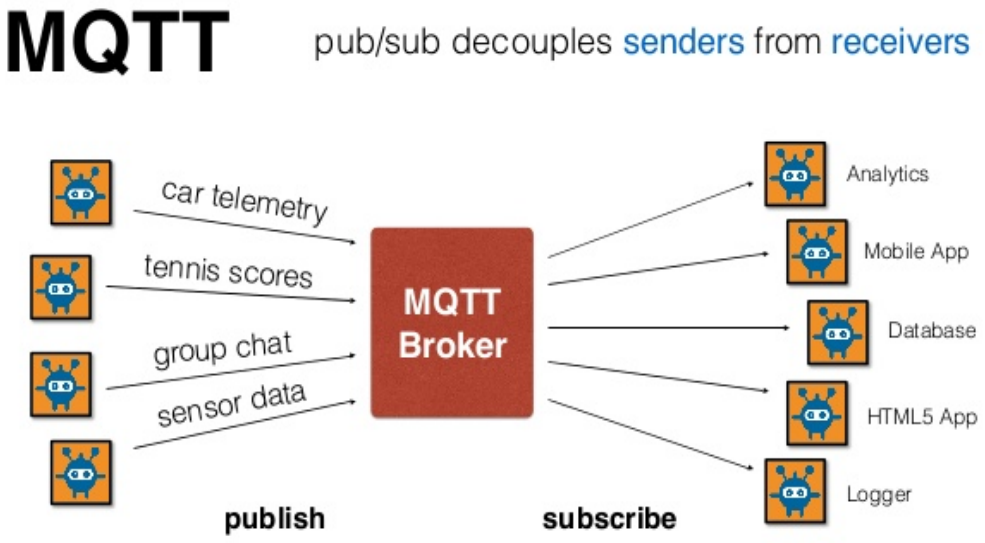


Illustration 33: MQTT Broker Diagram

8.2 IoT Architecture

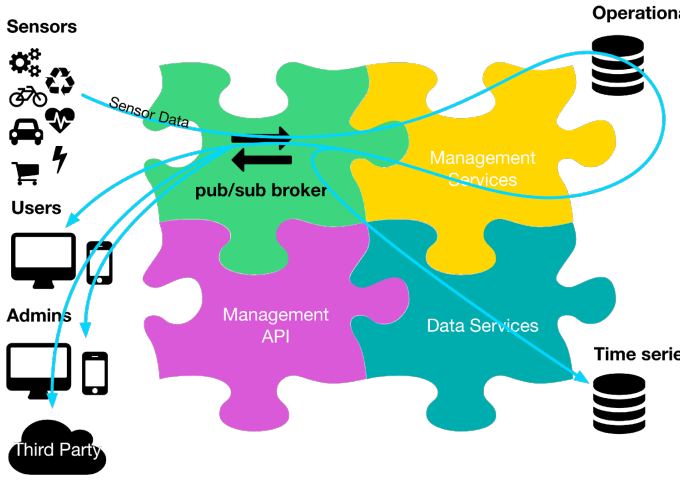


Illustration 34: IoT pub/sub data

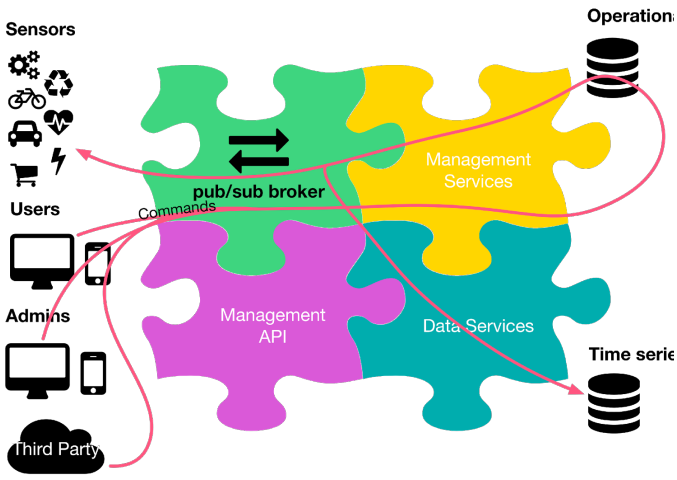


Illustration 35: IoT pub/sub commands

The main idea of the above architecture is to expose a pub/sub broker up front to the users and the devices. This design makes it is possible to direct the interaction between the users and the sensors, both for commands and data. This approach reduces latency, and allows for offline handling of time-series data and analytics.[37]

8.2.1 Sensor API

A sensor API, which is called by the sensors to deliver data readings and receive commands.

8.2.2 Public API

A public API, which is called by the sensors to retrieve real-time data, historical data, and to manage the devices.

8.2.3 Operations Services

A set of operational services, which are responsible for authentication and authorization, among other things; these services manage their own database.

8.2.4 Data Services

A set of data services, which are responsible for storing and analyzing the data, either in real time or offline.

8.3 REST API

REST (REpresentational State Transfer) is an architectural style, and an approach to communications that is often used in the development of Web services. The use of REST is often preferred over the more heavyweight SOAP (Simple Object Access Protocol) style because REST does not leverage as much bandwidth, which makes it a better fit for use over the Internet. The SOAP approach requires writing or using a provided server program (to serve data) and a client program (to request data).[38]

Illustration 36: REST Logo

REST'S decoupled architecture, and lighter weight communications between producer and consumer, make REST a popular building style for cloud-based APIs.

REST architecture involves reading a designated Web page that contains an XML file. The XML file describes and includes the desired content. Once dynamically defined, consumers may access the interface.

8.3.1 REST Architecture Example Diagram

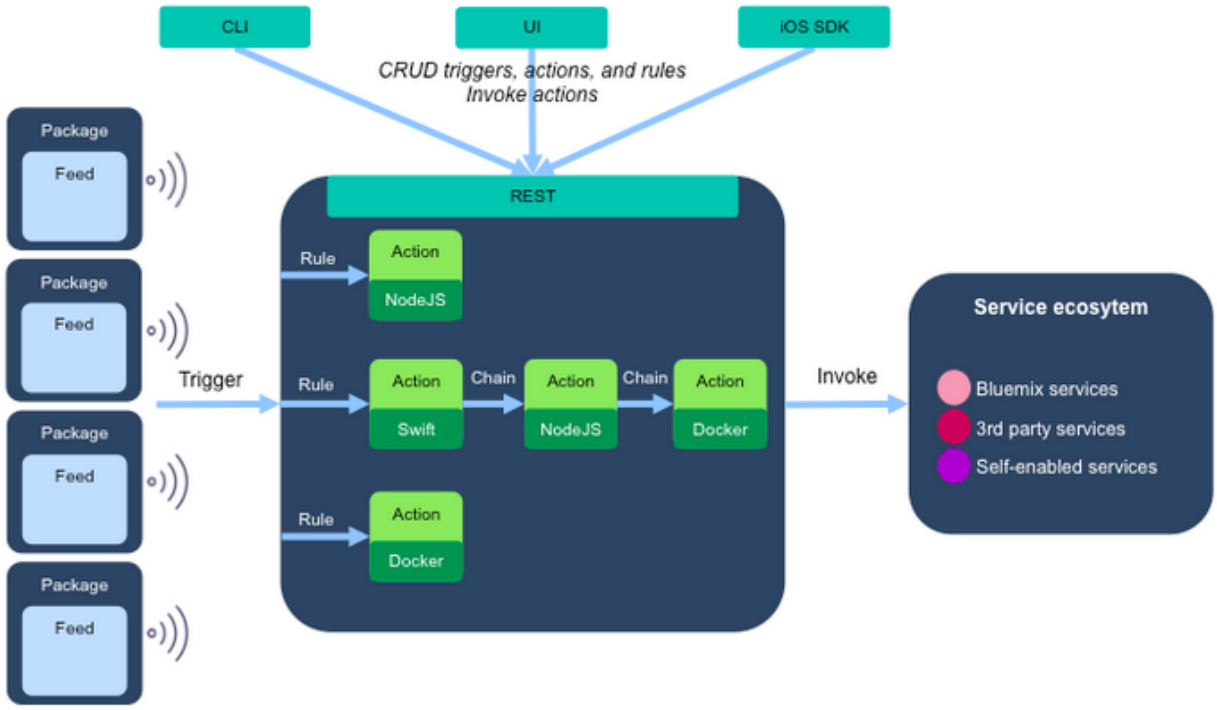


Illustration 37: REST Example Diagram

8.4 REST and MQTT

8.4.1 REST and MQTT Mirroring

This technique is applied to all REST API services that manage resources that can change state.**[31]**

REST poses a problem when services depend on being up to date with data they don't own and manage. Being up to date requires polling, which quickly add up in a system with enough interconnected services. Microservices need to know when the state they are interested in changes without unnecessary polling.**[39]**

The mirroring technique uses the end-point URL as an MQTT topic. A downstream microservice using a REST API endpoint can also subscribe to the MQTT topic that matches the end-point syntax. A microservice starts by making an HTTP GET request to obtain the initial state of the resource. Subsequent changes to the resource made by any other microservice are tracked through the MQTT events.

1. The service with a REST API will field requests by client services as expected. This establishes the baseline state.**[40]**
2. All the services will simultaneously connect to a message broker.
3. API service will fire messages notifying about data changes (essentially for all the verbs that can cause the change, in most cases POST, PUT, PATCH and DELETE).
4. Clients interested in receiving data updates will react to these changes according to their functionality.
5. In cases where having the correct data is critical, client services will forgo the built-up baseline + changes state and make a new REST call to establish a new baseline before counting on it.

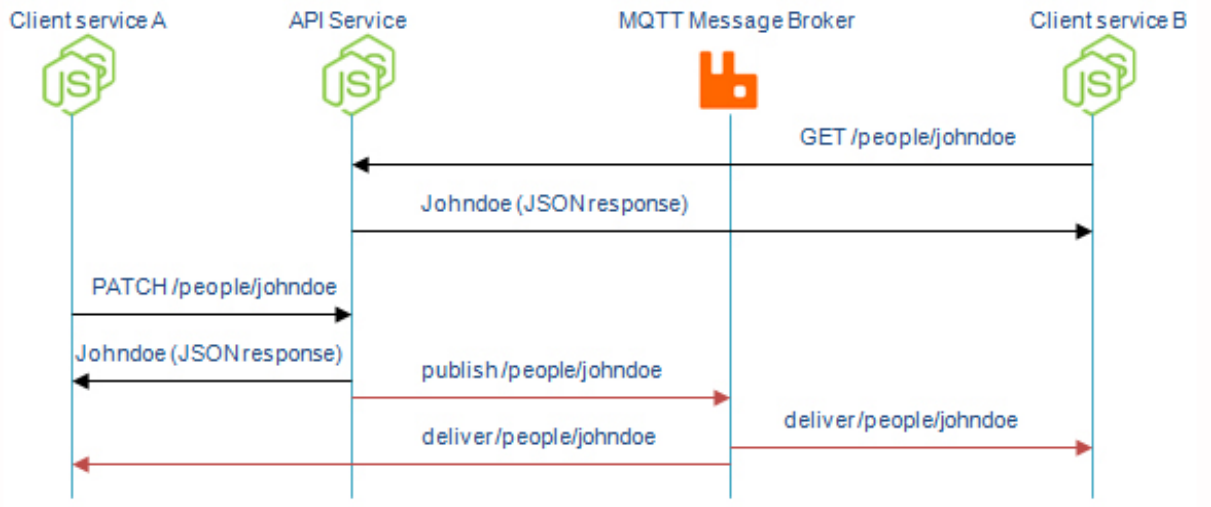


Illustration 38: REST and MQTT Mirroring

8.4.2 Message broker with REST endpoints

Another set of abstractions is to enable the configuration and of existing message systems such as MQTT using REST endpoints. There are two interfaces, a message interface and a REST interface, to shared stored resources. There are two message operations supported, to enable clients or brokers to publish updates to REST endpoints through a message interface, and to publish updates of REST endpoints to brokers or clients subscribing through a message interface. There is also a subscribe operation using the message interface. Incoming messages update resources, and resource updates send outgoing messages.[41]

The convention is to use URIs as topics that allow the system to construct URLs for the REST resource endpoints. The payload consists of the resource representation, e.g. JSON. This handles incoming and outgoing publish operations and acts as a message broker with REST access to topics and, if there is storage, topic history storage. REST Message Broker allows message clients to publish to REST endpoints and subscribe to REST endpoints. An application can monitor an MQTT endpoint by observing the REST resource it publishes to, and an application can publish to an MQTT client by updating to the REST endpoint the MQTT endpoint is subscribed to. For example, sensors could connect with MQTT and publish updates for application software to read, and application software could update a REST endpoint, resulting in the system publishing the update to a sensor that is subscribed to the URL as topic.

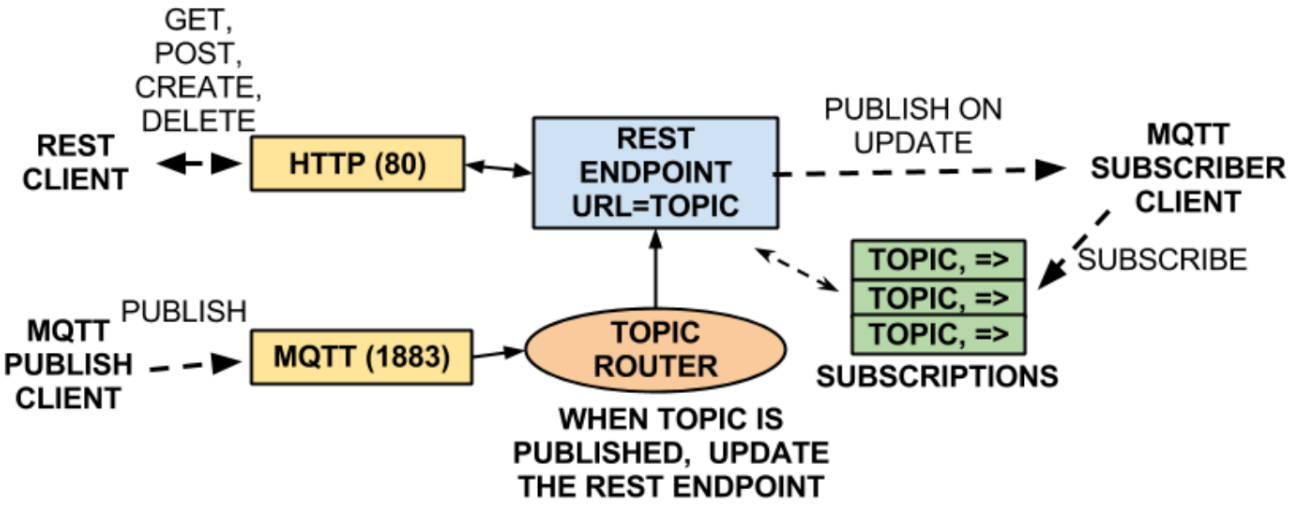


Illustration 39: REST and MQTT Interface

The above example shows how an MQTT broker endpoint can be added to a REST server to enable REST clients and MQTT clients to share data by mapping URLs to topics. When either a REST client updates a resource an MQTT client publishes to a topic, the corresponding resource update is published to any subscribers.[41]

8.5 Service Discovery

Services typically need to call one another. In a monolithic application, services invoke one another through language-level method or procedure calls. In a traditional distributed system deployment, services run at fixed, well known locations (hosts and ports) and so can easily call one another using HTTP/REST or some RPC mechanism. However, a modern microservice based application typically runs in a virtualized or containerized environments where the number of instances of a service and their locations changes dynamically.[42]

8.5.1 Problem

How does the client of a service - the API gateway or another service - discover the location of a service instance?

8.5.2 Forces

- Each instance of a service exposes a remote API such as HTTP/REST, or Thrift etc. at a particular location (host and port)
- The number of services instances and their locations changes dynamically.
- Virtual machines and containers are usually assigned dynamic IP addresses.
- The number of services instances might vary dynamically. For example, an Autoscaling Group adjusts the number of instances based on load.

8.5.3 Solution

When making a request to a service, the client makes a request via a router that runs at a well known location. The router queries a service registry, which might be built into the router, and forwards the request to an available service instance.

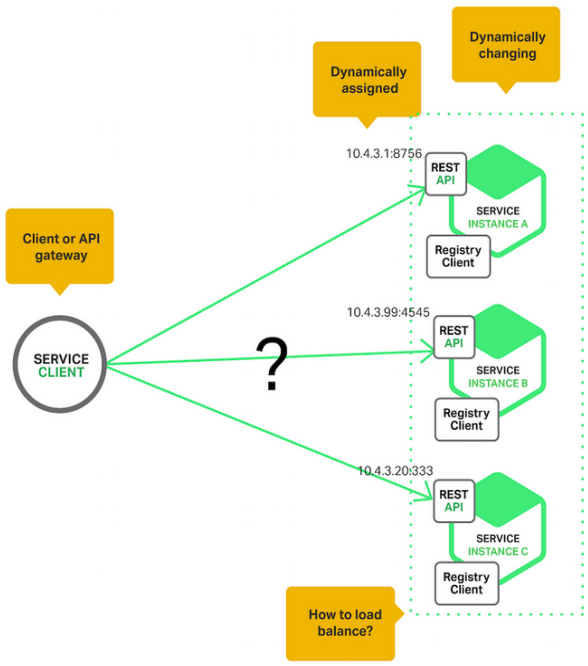


Illustration 40: Service Discovery

8.6 Scaling dimensions

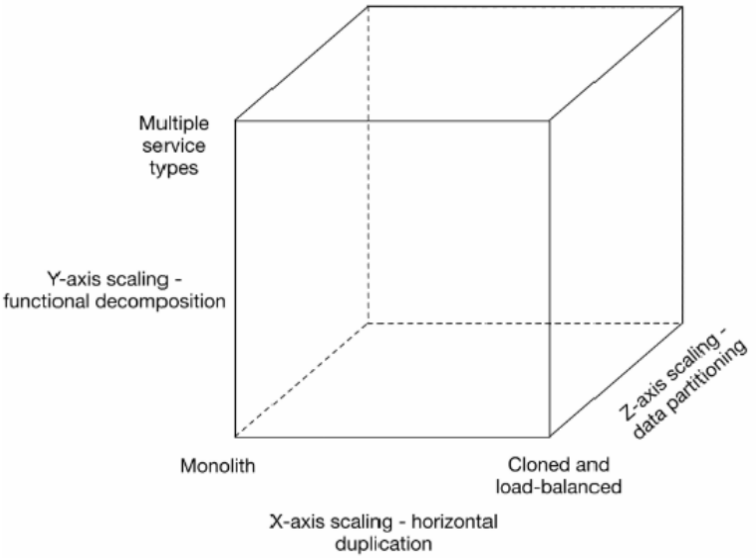


Illustration 41: Scaling Cube

8.6.1 X-axis scaling

X-axis scaling consists of running multiple copies of an application behind a load balancer. If there are N copies then each copy handles 1/N of the load. This is a simple, commonly used approach of scaling an application.

One drawback of this approach is that because each copy potentially accesses all of the data, caches require more memory to be effective. Another problem with this approach is that it does not tackle the problems of increasing development and application complexity.[43]

8.6.2 Y-axis scaling

Unlike X-axis and Z-axis, which consist of running multiple, identical copies of the application, Y-axis axis scaling splits the application into multiple, different services. Each service is responsible for one or more closely related functions. There are a couple of different ways of decomposing the application into services. One approach is to use verb-based decomposition and define services that implement a single use case such as checkout. The other option is to decompose the application by noun and create services responsible for all operations related to a particular entity such as customer management. An application might use a combination of verb-based and noun-based decomposition. [43]

8.6.3 Z-axis scaling

When using Z-axis scaling each server runs an identical copy of the code. In this respect, it's similar to X-axis scaling. The big difference is that each server is responsible for only a subset of the data. Some component of the system is responsible for routing each request to the appropriate server. One commonly used routing criteria is an attribute of the request such as the primary key of the entity being accessed. Another common routing criteria is the customer type. For example, an application might provide paying customers with a higher SLA than free customers by routing their requests to a different set of servers with more capacity.[43]

8.7 Load balancing and follow-on traffic

The client attempts to connect with the service.

- The LB accepts the connection, and after deciding which host should receive the connection, changes the destination IP (and possibly port) to match the service of the selected host (note that the source IP of the client is not touched).
- The host accepts the connection and responds back to the original source, the client, via its default route, the LB.
- The LB intercepts the return packet from the host and now changes the source IP (and possible port) to match the virtual server IP and port, and forwards the packet back to the client.
- The client receives the return packet, believing that it came from the virtual server, and continues the process.

There are generally two specific issues with handling follow-on traffic once it has been load balanced: connection maintenance and persistence.

8.7.1 Connection maintenance

Connection maintenance requires two key capabilities. The first is the ability to keep track of open connections and the host service they belong to. Second, the load balancer must be able to continue to monitor that connection so the connection table can be updated when the connection closes.[44]

8.7.2 Persistence

When the client uses multiple short-lived TCP connections (for example, Port 80: HTTP) to accomplish a single task. In some cases, like standard web browsing, it doesn't matter and each new request can go to any of the back-end service hosts; however, there are many instances where it is extremely important that multiple connections from the same user go to the same back-end service host and not be load balanced. This concept is called persistence, or server affinity.

In modern HTTP transactions, the server can specify a "keep-alive" connection, which turns those multiple short-lived connections into a single long-lived connection, which can be handled just like the other long-lived connections. However, this provides only a little relief, mainly because, as the use of web and mobile services increases, keeping all the connections open longer than necessary strains the resources of the entire system. That's why today for the sake of scalability and portability many organizations are moving toward building stateless applications that rely on APIs. This basically means that the server will forget all session information to reduce the load on the resources and in these cases, the state is maintained by passing session IDs as well as through the concept of persistence.

One of the most basic forms of persistence is source-address affinity, which involves simply recording the source IP address of incoming requests and the service host they were load balanced to, and making all future transactions go to the same host. Two ways to accomplish this are by using SSL session IDs and cookies. SSL persistence tracks SSL session using SSL session IDs, which means that even when the client's IP address changes, the load balancer will recognize the session being persistent based on session ID.[44]

8.8 Extract Transform Load

The three words in Extract Transform Load each describe a process in the moving of data from its source to a formal data storage system (most often a data warehouse).

8.8.1 Extract

The extraction process is the first phase of ETL, in which data is collected from one or more data sources and held in temporary storage where the subsequent two phases can be executed. During extraction, validation rules are applied to test whether data has expected values essential to the data warehouse. Data that fails the validation is rejected and further processed to discover why it failed validation and remediate if possible.[45]

8.8.2 Transform

In the transformation phase, the data is processed to make values and structure consistent across all data. Typical transformations include things like date formatting, resorting rows or columns of data, joining data from two values into one, or, conversely, splitting data from one value into two. The goal of transformation is to make all the data conform to a uniform schema.[45]

8.8.3 Load

The load phase moves the transformed data into the permanent, target database. Once loaded, the ETL process is complete, although in many organizations ETL is performed regularly in order to keep the data warehouse updated with the latest data.[45]

8.8.4 The need for ETL

ETL takes data that is heterogeneous and makes it homogeneous. Without ETL it would be impossible to programmatically analyze heterogeneous data and derive business intelligence from it.

8.8.5 The difference between heterogeneous and homogeneous

A DDBMS may be classified as homogeneous or heterogeneous. In a homogeneous system, all sites use the same DBMS product. In a heterogeneous system, sites may run different DBMS products, which need not be based on the same underlying data model, and so the system may be composed of relational, network, hierarchical and object-oriented DBMSs.

Homogeneous systems are much easier to design and manage. This approach provides incremental growth, making the addition of a new site to the DDBMS easy, and allows increased performance by exploiting the parallel processing capability of multiple sites.[45]

8.9 Similar Projects

8.9.1 TRANSIT: Flexible pipeline for IoT data with Bluemix and OpenWhisk

This Blog post helped explain how to interface Node-Red with OpenWhisk.[46] The proposed architecture uses the Watson IoT Platform and Message Hub in order to handle MQTT client messages. These messages are published to a Kafka server which in turns uses OpenWhisk to decode the base64 encoded data. The data is stored using IBM Object Storage which is then sent to Apache Spark for analytics.

This article also helped with understanding IoT simulation using Node-Red, however the architecture of this theses is aimed more towards using Containers to handle the load balancing of MQTT clients.

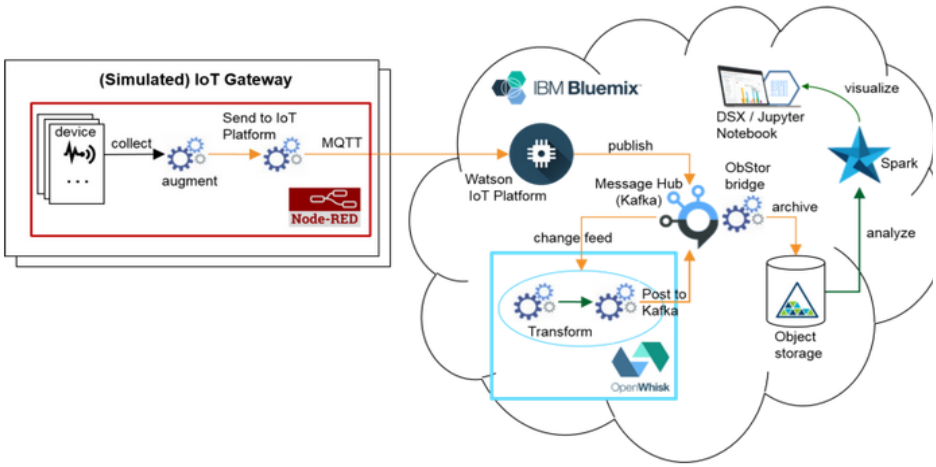


Illustration 42: TRANSIT Diagram

8.9.2 How to Build a High Availability MQTT Cluster for the Internet of Things

A Significant amount of the proposed architecture for this project is borrowed from this blog post. [47]

It details the following.

- Setting up a Mosca Broker with Redis
- Dockerizing the MQTT server
- Adding HAProxy as a load balancer
- Access Control List configuration
- Making MQTT secure with SSL

9 Project Planning

9.1 Requirements Analysis

9.1.1 Requirements description

A student, who has been given the task of creating a smart campus analytics application that uses information obtained from university campus sensors, wants to use a microservices architecture to develop a public API. The student should be able to:

- Have all the necessary infrastructure in place to start writing code
- Obtain sensor data from the database
- Make code changes that trigger an automated build
- Have automated integration tests that prevent insecure commits
- See their application automatically deployed once all tests have passed

9.1.2 Identify the business processes

BP1: Public API development automation process

9.1.3 Identify IT processes that support each of the business processes

BP1: Public API development automation process

ITP1: Automated Infrastructure build process

ITP2: Automated Integration testing process

ITP3: Automated deployment process process

9.1.4 Identify the activities within each of the IT processes

ITP1: Automated Infrastructure build process

A1: Container build activity

A2: Integration server trigger build activity

ITP2: Automated Integration testing process

A1: Set up testing environment activity

A2: Run tests activity

ITP3: Automated deployment process process

A1: Image security scan activity

A2: Container orchestration deployment activity

9.2 Autumn semester

During the Autumn semester the requirements were not as clearly defined. The main objective at this stage was to research the technology needed to perform data acquisition of sensors with a particular focus on the topic of load balancing. Below is a High level diagram that provided enough detail to go about developing the architecture of the system. The scope of the project was stretched too far with the inclusion of a front end web application. This aspect of the system was later removed at suggestion of the project supervisor due to time constraints.

The research at this stage focused heavily on geospatial tracking requirements, which lead to the suggestion of using a nosql graph database. This too was also removed in order to reduce the complexity of the project. As can be seen in the diagram below, the backend lacked detail and merely described the major software processes involved in the system.

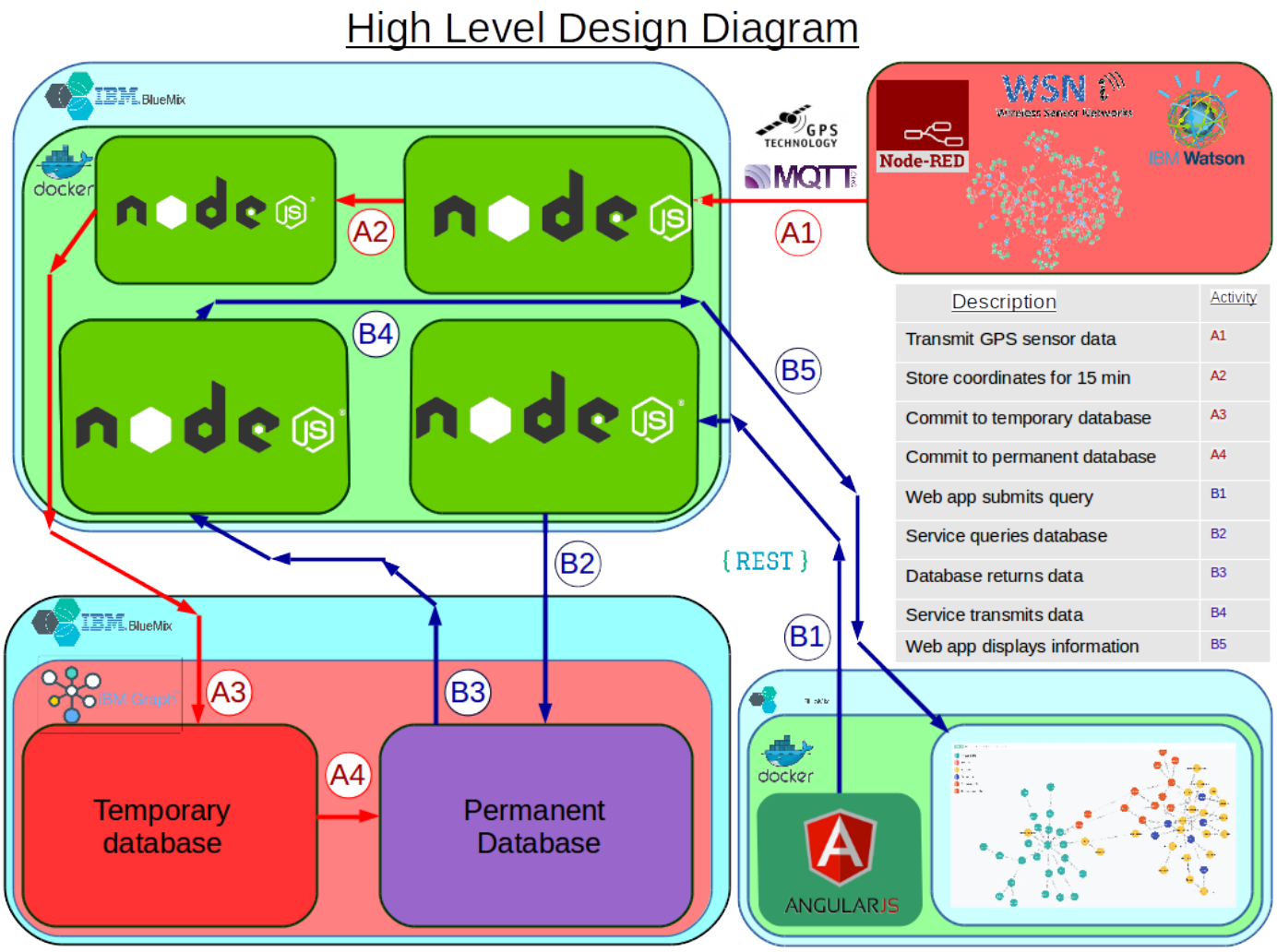


Illustration 43: High Level Design Diagram

9.3 Autumn work completed

Work commenced on week 5 of the Autumn semester. Weeks 5,6,7,8 logs can be found on the Github repository. These logs provide a more in depth view of the work completed.

9.3.1 Week 5 [48]

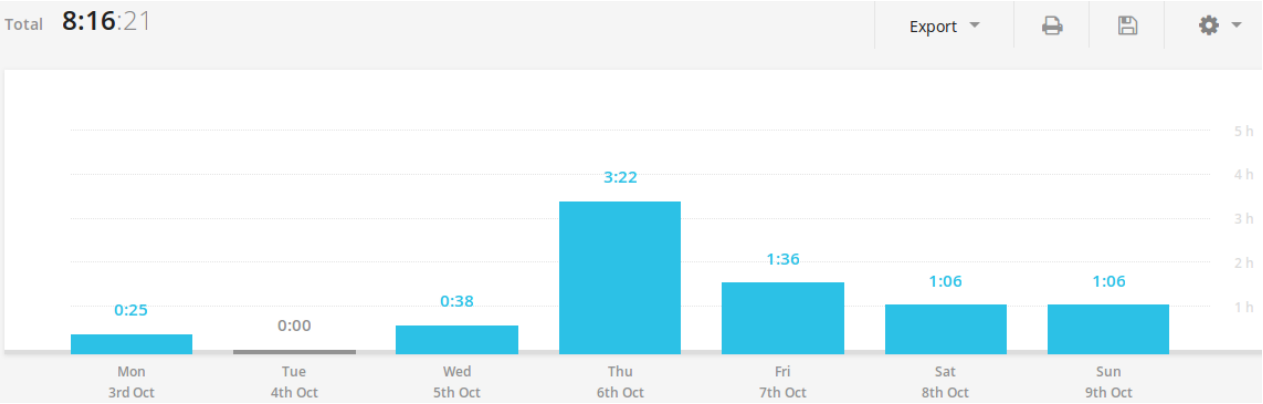


Illustration 44: Week 5 Bar Chart

Tasks Completed

- Create Toggl Project Labels
- Create Project folders
- Create Log template
- Arrange meeting with supervisor
- Backup files for installation of Ubuntu
- Initial Research
- Research Bluemix
- Meet with supervisor
- Download Ubuntu ISO
- Install Ubuntu

9.3.2 Week 6 [49]

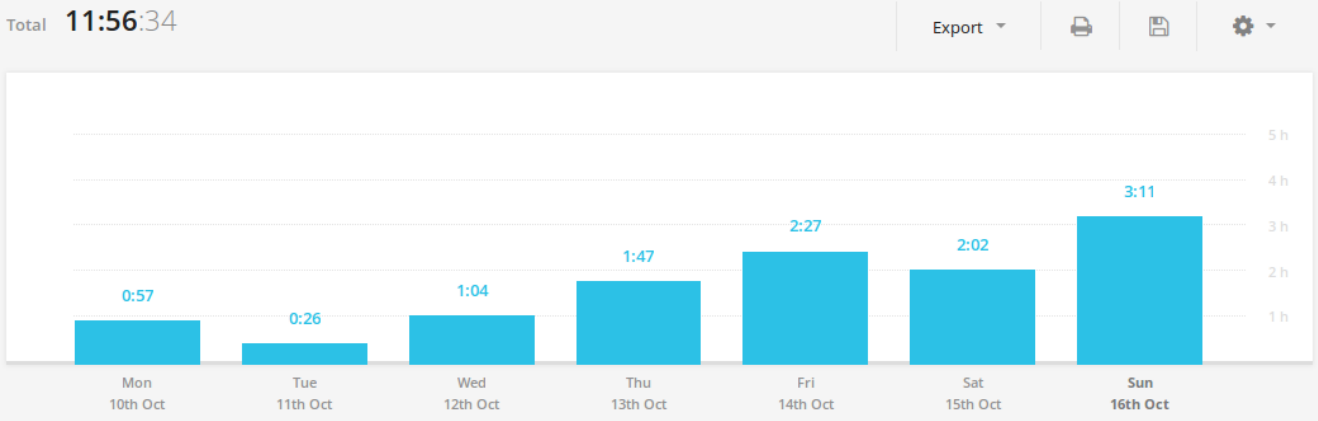


Illustration 45: Week 6 Bar Chart

Tasks Completed

- Collected Bluemix competitor bookmarks
- Read Bluemix competitor bookmarks
- Create Github repositories
- Take photos of hand written notes
- Finish week 5 log
- Installed Docker
- Amazon container service research
- Microsoft Azure container service research
- IBM Bluemix research
- Docker research
- Initial database research
- NoSQL database research
- Neo4j graph database research

9.3.3 Week 7 [50]

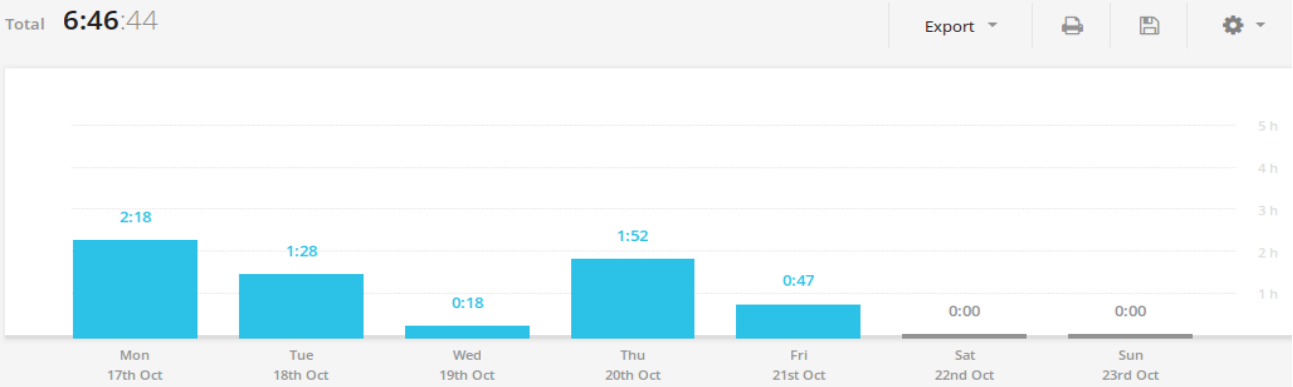


Illustration 46: Week 7 Bar Chart

Tasks Completed

- Week6 log
- layout project plan
- layout critical path
- Sprint Gantt chart
- Summer Gantt chart
- Autumn report project plan section
- Commit report to github
- Two page sumary of project

Log Entries

17/10/16: Today I laid out a rough project plan for my project. After reviewing it I determined the critical path of the project.

18/10/16: Today I created the gantt chart for the spring semester project plan.

19/10/16: Today I created the gantt chart for the summer semester project plan.

20/10/16: Today I spent considerable time creating my high level design diagram. I should have shown my supervisor a rough drawing of it before I put it together on powerpoint as I know it most likely need changes.

21/10/16: Today I did a little research on development operations, learning about the various responsibilities associated with the role such as load balancing and automation.

9.3.4 Week 8 [51]

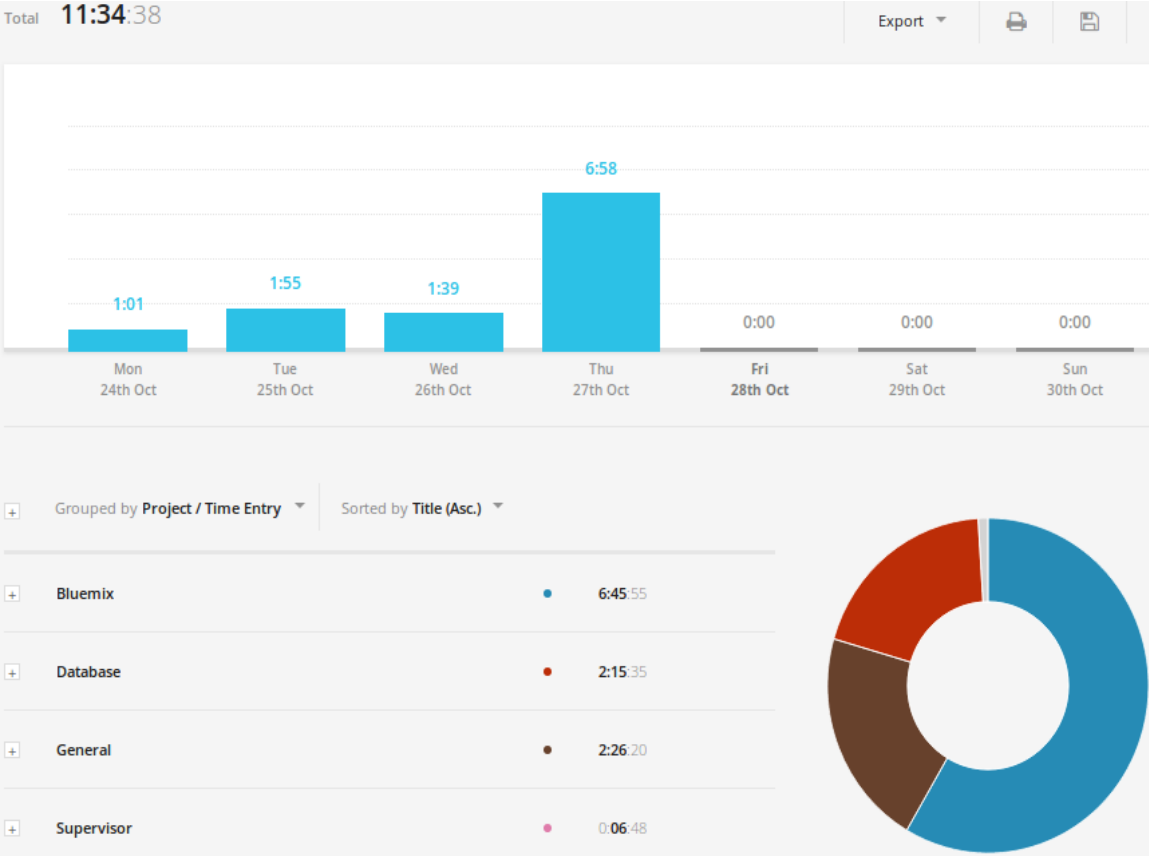


Illustration 47: Week 8 Bar Chart

Tasks Completed

- Week 7 log
- Create Bluemix account
- Email supervisor
- Simulate IoT sensor database
- Install Bluemix CLI
- Install Bluemix CLI plugins
- Docker Bluemix demo

9.4 Spring semester

During the spring semester a significant amount of research was carried out on open source technology. The main problem encountered with this was the variety of technology, it was difficult to define the exact architecture of the system as there was too much to choose from.

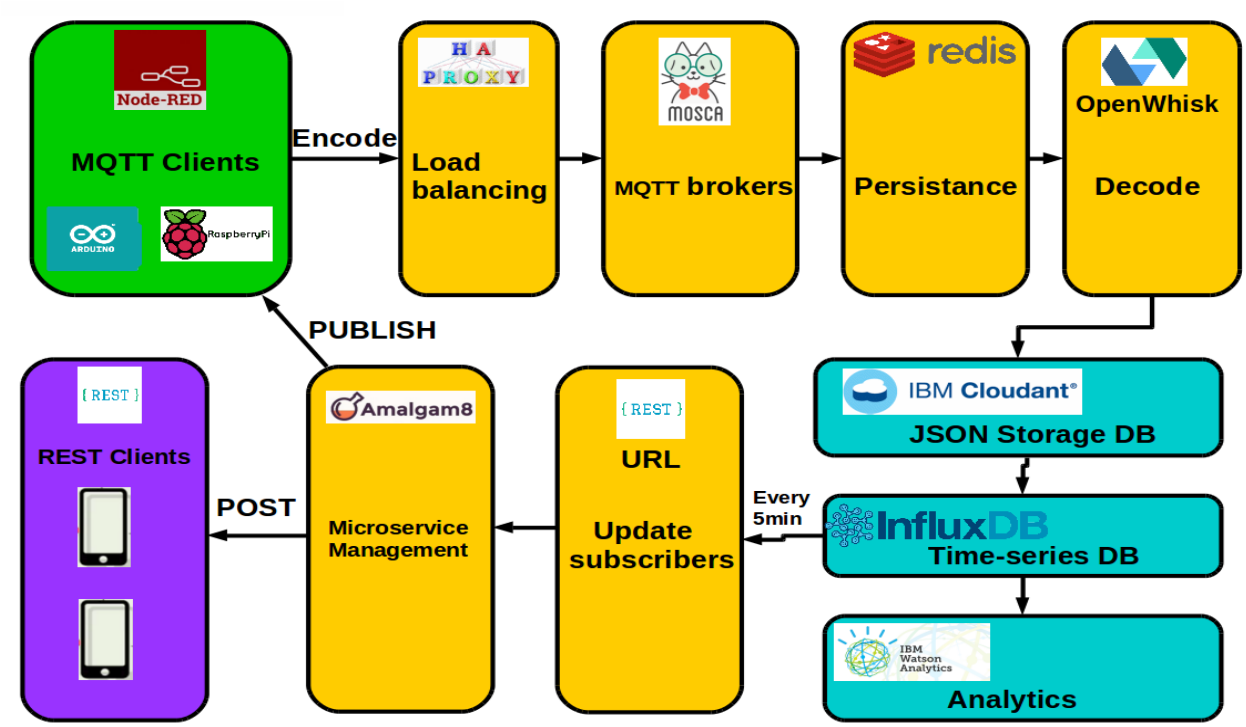


Illustration 48: Overall Architecture

Node-RED simulates MQTT clients and injects GeoJSON messages encoded in base64 to the HAProxy Container. These messages are then load balanced between two Mosca brokers that cache the data in Redis for persistence. A serverless OpenWhisk action is triggered to decode the data from base64 and store it in Cloudant as JSON.

InfluxDB will organise the data into a time-series for efficiency. The continuous queries will be performed in relation to both vital information for microservices registered in the A8 registry as well as various Watson Analytics applications. The continuous queries precompute expensive queries into another time series in real-time.

The topic URL has MQTT clients subscribed to it. Both MQTT and REST clients will be updated once the Amalgam8 load balancer decides which version of the microservice to run. These services can be viewed as downstream services, their execution is dependent on the InfluxDB queries which are acting as upstream services for both Amalgam8 and the IBM Watson Analytics platform.

9.5 Spring work completed

9.5.1 Week 1 [52]

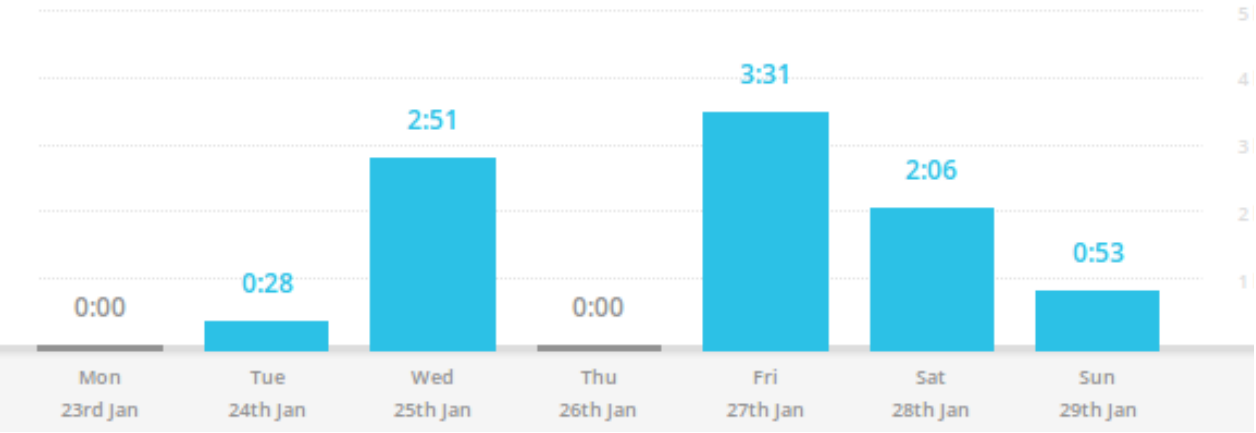


Illustration 49: Week 1 Bar Chart

Tasks Completed

- **Meet with supervisor**
- **Report Template**
- **Research MQTT**
- **Research Kafka**
- **Find similar project**
- **Research databases**

Week 1 Log Entries

Entry 24/01/17:

"Today I met with my supervisor. We discussed various aspects of the project. My supervisor advised me to focus on the back-end architecture for now as the scope of the project is too large."

Entry 25/01/17:

"Today I ran into some problems setting up the academic free trial. I had to email the Bluemix support team in order to resolve the issue."

Week 1 Time Log

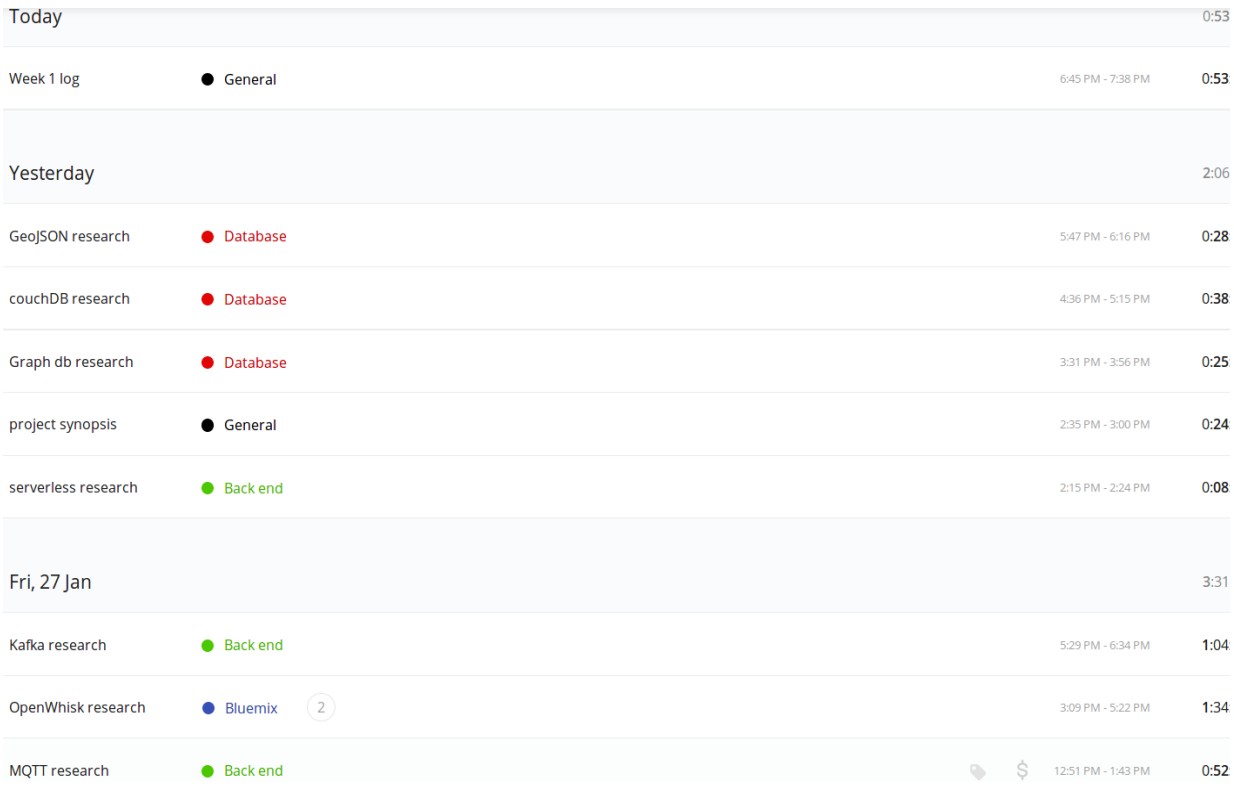


Illustration 50: Week 1 Time Log 2

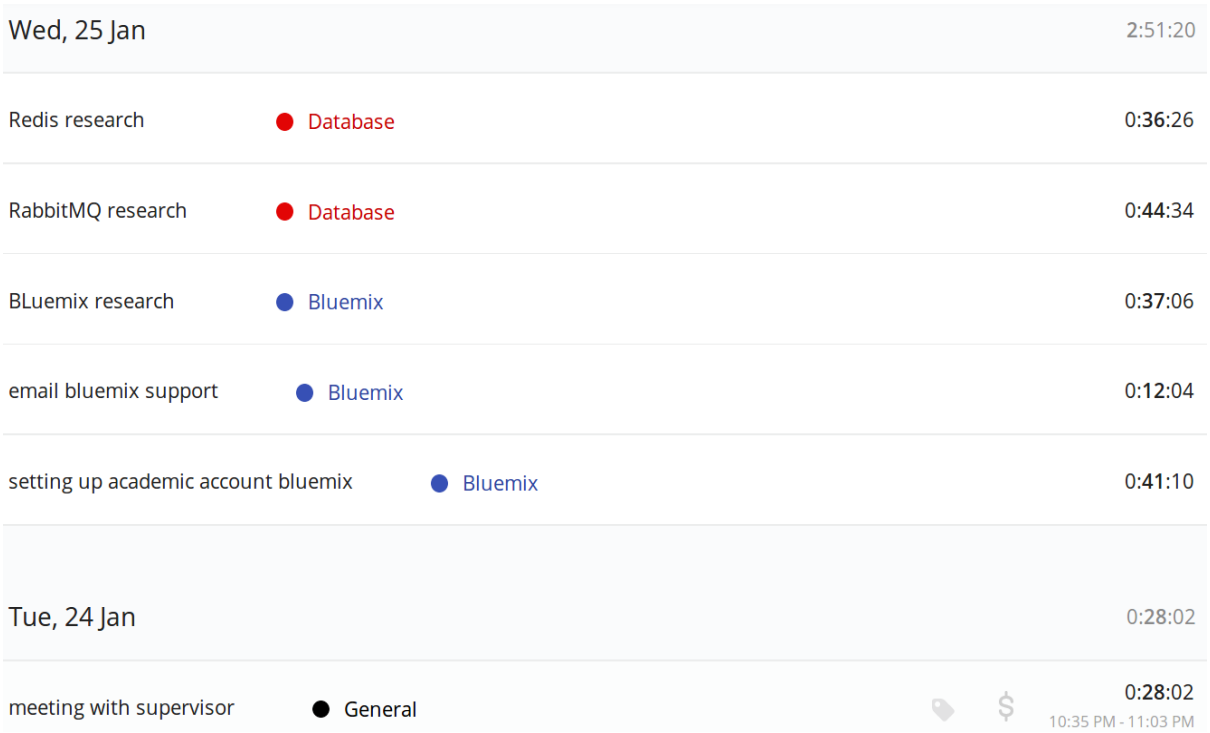


Illustration 51: Week 1 Time Log 1

9.5.2 Week 2 [53]

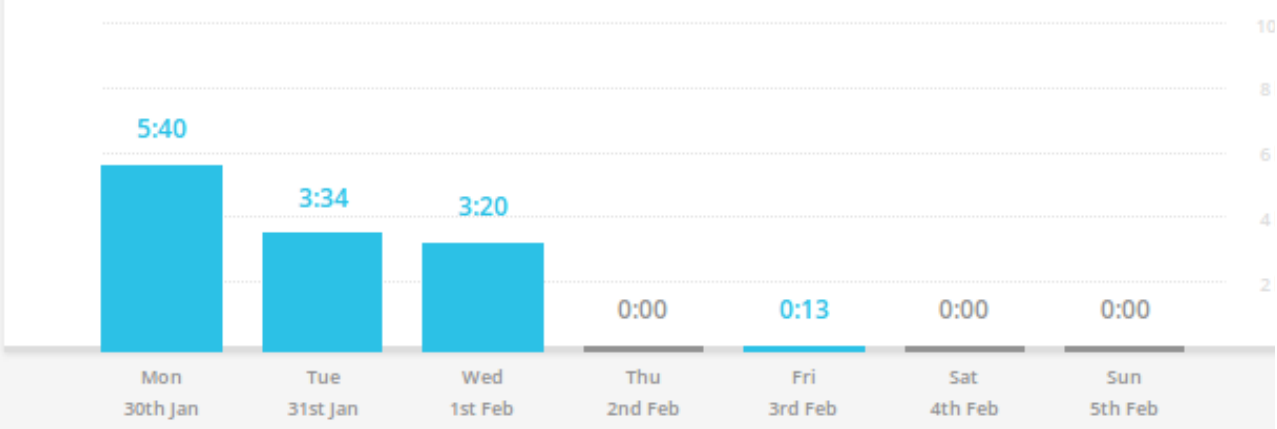


Illustration 52: Week 2 Bar Chart

Tasks Completed

- **Create new Bluemix account for academic free trial**
- **IBM Container Service research**
- **IBM Container group research**
- **Amalgam8 research**
- **General research (jumping between mqtt brokers and bluemix links)**
- **MQTT container research**
- **Microservices from Theory to Practice (IBM document)**
- **Mosca research**
- **Questions & answers (self evaluation)**
- **Research links document for supervisor**
- **(rough) architectural diagram**

Week 2 Time Log

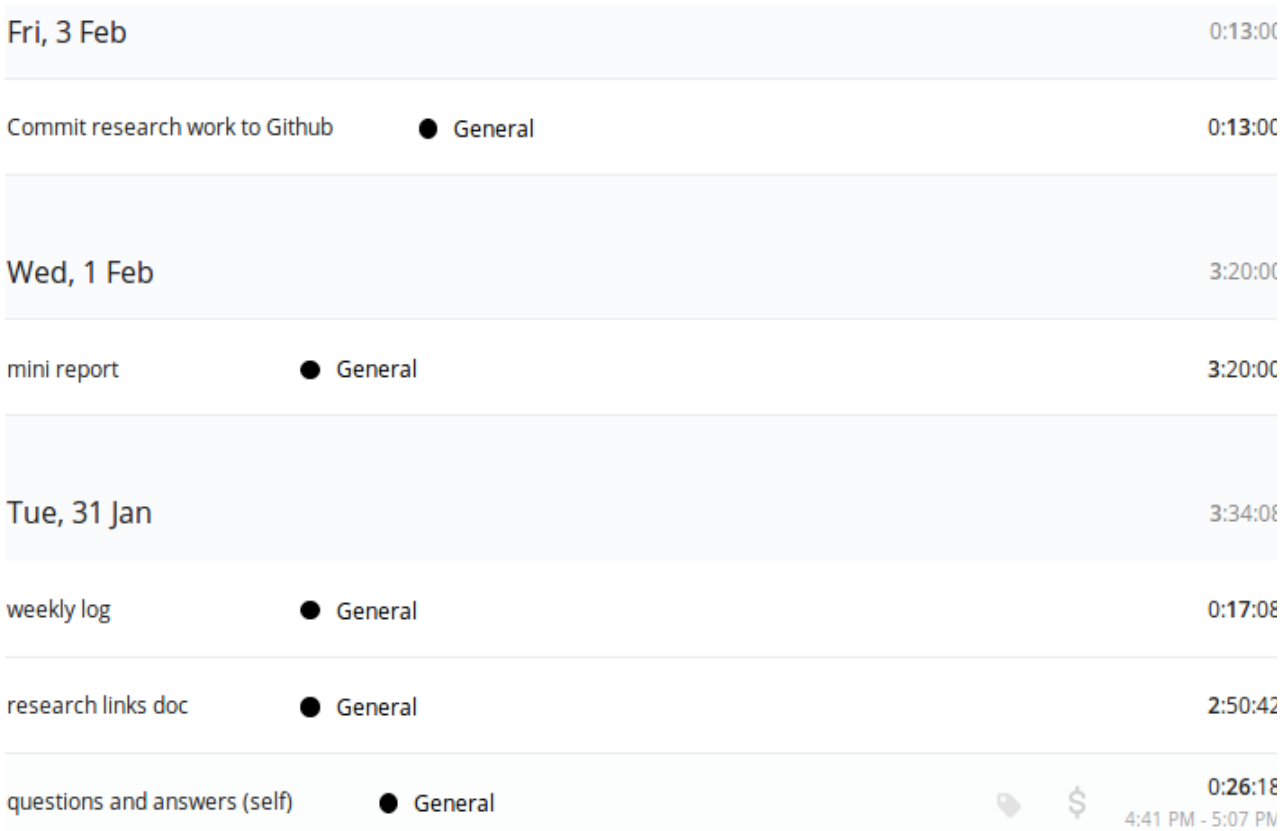


Illustration 53: Week 2 Time Log 2

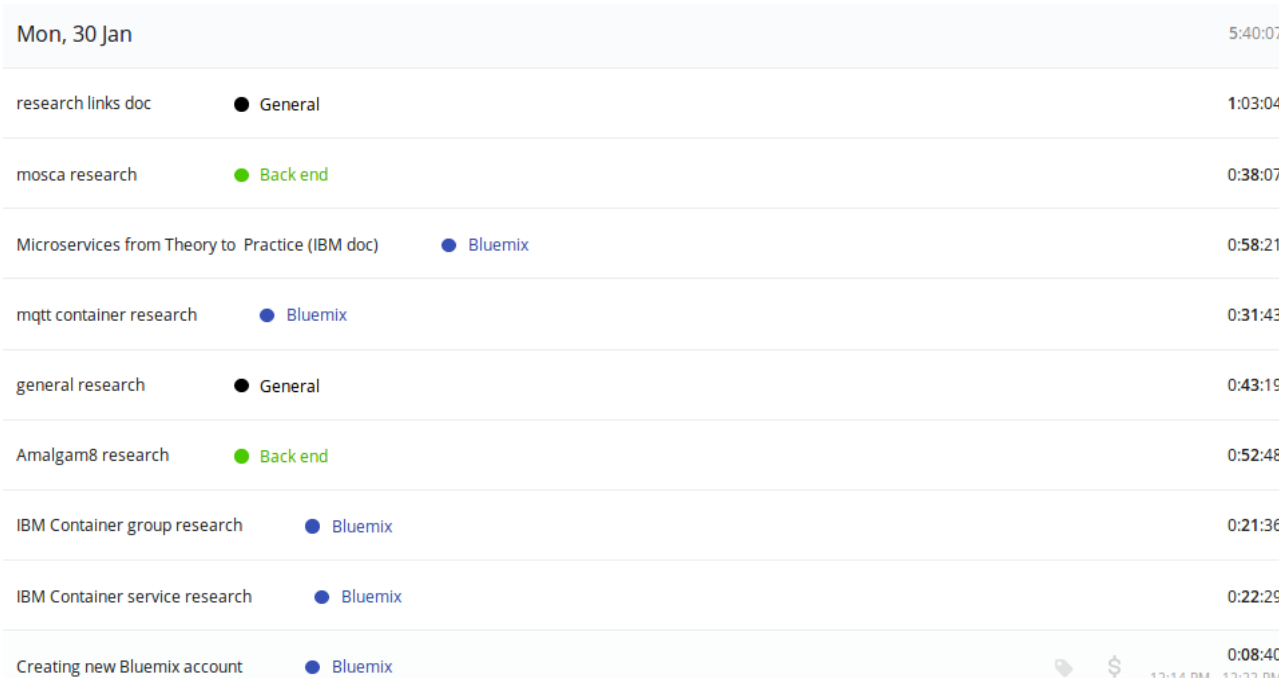


Illustration 54: Week 2 Time Log 1

9.5.3 Week 3 Log [54]

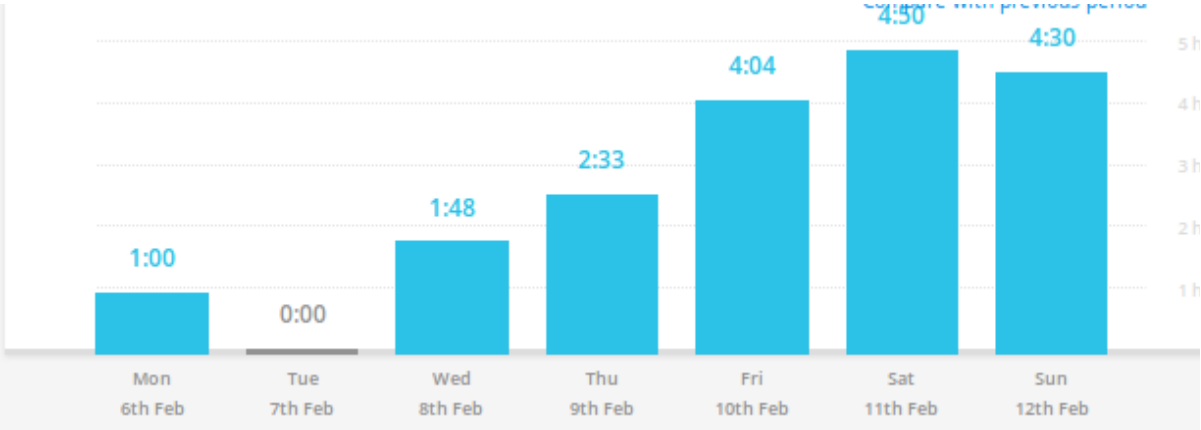


Illustration 55: Week 3 Bar Chart

Week 3 Time Log

Today			4:30:00
log book	General	6:53 PM - 8:53 PM	2:00:00
node-red	Front end	1:52 PM - 4:22 PM	2:30:00
Yesterday			4:50:04
node-red	Front end (2)	12:04 PM - 9:34 PM	3:43:00
DevOps secure pipeline	Security/Devops	5:11 PM - 6:18 PM	1:07:04
Fri, 10 Feb			4:04:09
doodling	General	10:12 PM - 11:34 PM	1:21:55
mosca broker	Bluemix (2)	3:54 PM - 5:36 PM	2:42:14
Thu, 9 Feb			2:33:22
mosca broker	Bluemix	4:43 PM - 6:52 PM	2:09:40
Amalgam8 research	Security/Devops	1:48 PM - 2:10 PM	0:23:42
Wed, 8 Feb			1:48:58
mosca broker	Bluemix	5:52 PM - 7:40 PM	1:48:58
Mon, 6 Feb			1:00:00
meeting with supervisor	General	11:00 AM - 12:00 PM	1:00:00

Illustration 56: Week 3 Time Log

Tasks completed

- **Mosca Broker working locally**
- **Mosca Broker on Bluemix**
- **Node-RED injects msgs**
- **Node-RED sends to broker on bluemix**
- **Node-RED encodes and decodes base 64**
- **Node-RED stores data in cloudant**
- **Security DevOps pipeline tutorial**

Week 3 Log Entries

Entry 08/01/17:

“While still recovering from a flu, I managed to familiarize myself with the cloud foundry Bluemix CLI. I made my goal to simply get a custom container image onto Bluemix. I ran into various problems such as logging into the american “ng” domain and not the “eu-gb” europe domain. Half of the delay was due to poor memory of how I did this last time. The other half being my state of mind (illness).”

Entry 09/01/17:

“Today I managed to get a mosca broker running on my local Docker host. Using Mosquitto service to pub and sub.”

Entry 10/01/17:

“Today I deployed a mosca broker on IBM containers. I carried out the same test I did locally.”

Entry 11/01/17:

“Today I used Node-RED for the first time properly. I managed to get data flowing from an inject node to my broker running on IBM Container on Bluemix.”

Entry 12/01/17:

“Today I managed to encode/decode base 64 using Node-RED. I also managed to get the decoded data into a cloudant database. I ran into many problems this week, but time invested carried me through.”

9.5.4 Week 4 [55]

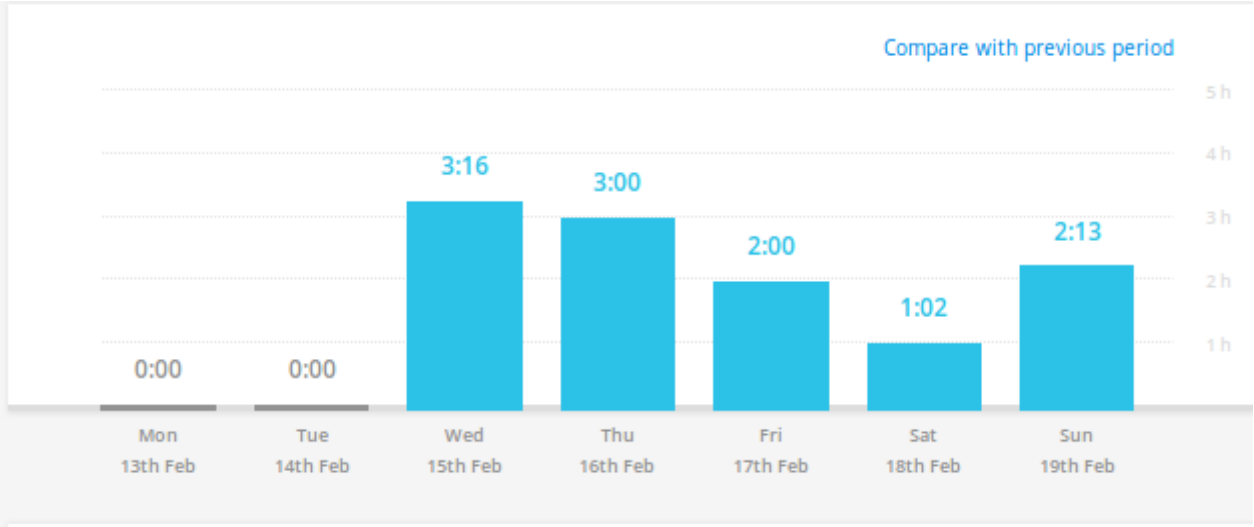


Illustration 57: Week 4 Bar Chart

Week 4 Time Log

Yesterday			2:13:42
report draft	General	5:46 PM - 7:59 PM	2:13:42
Sat, 18 Feb			1:02:23
report draft	General	1:46 PM - 2:48 PM	1:02:23
Fri, 17 Feb			2:00:00
report draft	General	4:44 PM - 6:44 PM	2:00:00
Thu, 16 Feb			3:00:00
Kafka research	Database	7:07 PM - 8:07 PM	1:00:00
Redis research	Database	5:07 PM - 7:07 PM	2:00:00
Wed, 15 Feb			3:16:15
HAProxy	Back end	2:23 PM - 5:39 PM	3:16:15

Illustration 58: Week 4 Time Log

Tasks Completed

- **Attempted to implement HAProxy container**
- **Redis research**
- **Kafka research**
- **ELK Stack research**
- **Report draft**
- **Literature survey spring report**

Week 4 Log Entries

Entry 15/02/17:

"Today I struggled to get the HAProxy container working. I spent the first hour and a half just trying to attach an "overide config" as a volume to the container. Once "**docker logs <haproxy container id>**" stopped reporting errors, I knew at least the container was building successfully. The next step is to export HAProxy logs in order to find out more information as to what the problem might be."

Entry 16/02/17:

"Today I researched both Kafka and Redis, trying to understand the difference between them and why/where you would use one over the other."

Entry 17/02/17:

"Today I started my Spring report, this consisted of laying out the template and putting in section headings. I also research Amalgam8 microservices framework."

Entry 18/02/17:

"Today I worked on "section 2 Literature survey" of my spring report. I also researched container monitoring and logging using the ELK stack."

Entry 19/02/17:

"Today I continued work on my Spring report, finishing off the literature survey section."

9.6 Summer semester

At the end of the Spring semester, this was the proposed plan for summer. The plan was too ambitious and severely underestimated the time needed to complete each of the tasks.

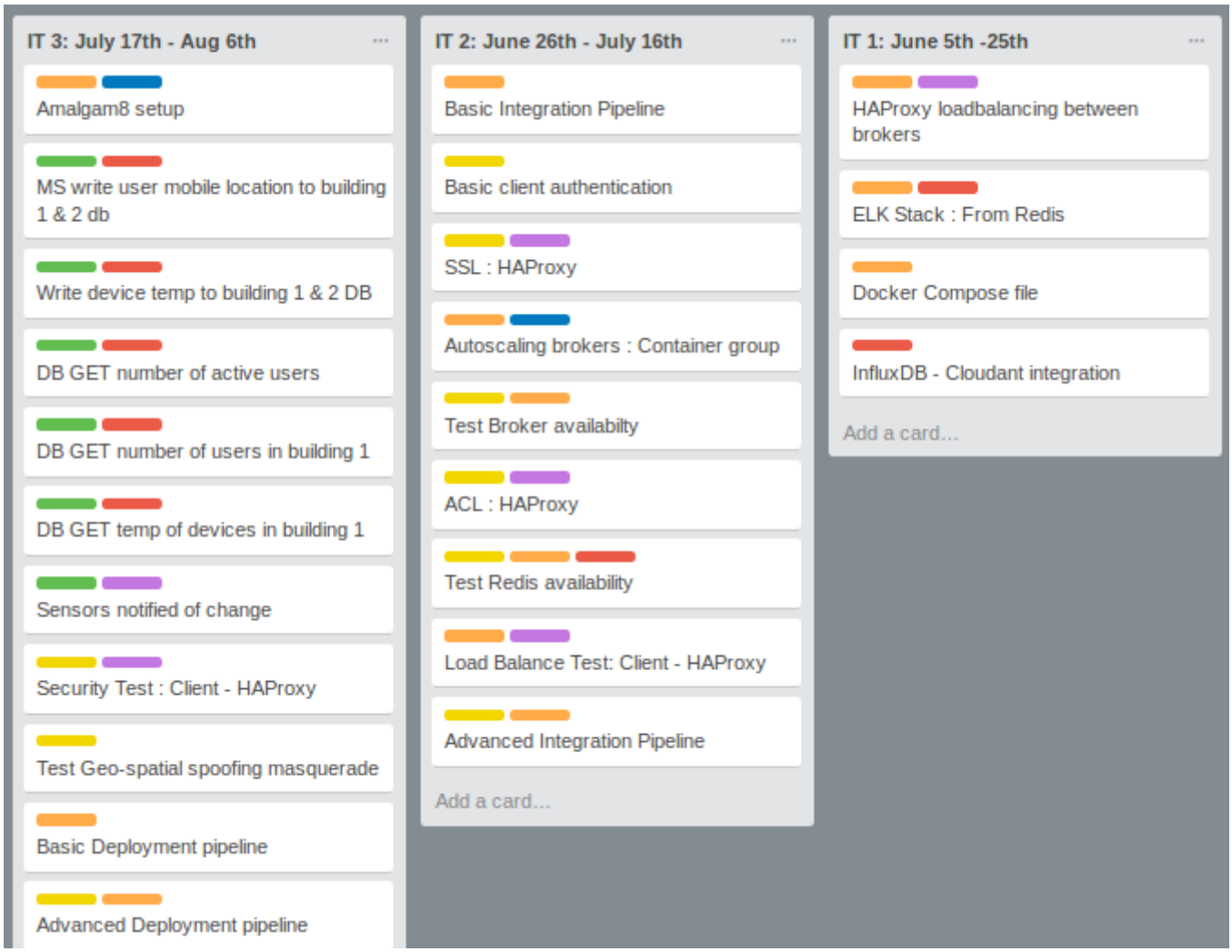


Illustration 59: Trello Iteration Planning

9.6.1 Tasks completed

- Docker-compose file working
- HAProxy load balancing between brokers
- Basic integration pipeline using Jenkins
- Redis and InfluxDB building in pipeline

9.6.2 IoT summer school

In July, time was allocated towards creating workshop learning material for students of the IoT summer school. This was not seen as a deviation as one of the objectives of the theses was to provide sufficient documentation for future students looking to build on top of this project. The learning material achieves this important goal.

Two separate lab manuals were written up. The first deals with learning MQTT and Node-Red. The students divided themselves into groups of 4. Part 1 of the lab involved getting each member of the group communicating with each other using MQTT topics. Part 2 dealt with using the Watson analytics translation node. The objective was to translate the phrase “hello world” in Chinese into English.

The second lab focused on using IBM’s serverless computing platform OpenWhisk. The students had to use an OpenWhisk function node to decode the message “hello world” that was encoded in base-64.

The learning material can be found in the Appendix section of the report and is also available on Github.

9.7 Project constraints

The memory limit for deploying containers in a single namespace is 2GB using the academic free-trial on Bluemix.

Budget:

1x HAProxy = 256MB

2x Mosca Brokers = 256MB * 2 = 512MB

1x Redis = 256MB

1x InfluxDB = 256MB

Total Used:

1.28 GB

9.8 Database Comparison

Editorial information provided by DB-Engines				
Name	**Cloudant** X	**Elasticsearch** X	**InfluxDB** X	**Redis** X
Description	Database as a Service offering based on Apache CouchDB	A modern search and analytics engine based on Apache Lucene	DBMS for storing time series, events and metrics	In-memory data structure store, used as database, cache and message broker
Database model	Document store	Search engine	Time Series DBMS	Key-value store
DB-Engines Ranking Trend Chart	Score 5.36 Rank #47 Overall #8 Document stores	Score 108.31 Rank #11 Overall #1 Search engines	Score 6.77 Rank #43 Overall #1 Time Series DBMS	Score 114.03 Rank #10 Overall #1 Key-value stores
Website	cloudant.com	www.elastic.co/products/-elasticsearch	www.influxdata.com/time-series-platform/influxdb	redis.io
Technical documentation	docs.cloudant.com	www.elastic.co/guide	docs.influxdata.com/influxdb	redis.io/documentation
Developer	IBM, Apache Software Foundation	Elastic		Salvatore Sanfilippo
Initial release	2010	2010	2013	2009
Current release		5.2.1, February 2017	v1.1.1, December 2016	3.2.8, February 2017
License	commercial	Open Source	Open Source	Open Source
Cloud-based	yes	no	no	no
Implementation language	Erlang	Java	Go	C
Server operating systems	hosted	All OS with a Java VM	Linux OS X	BSD Linux OS X Windows
Data scheme	schema-free	schema-free	schema-free	schema-free
Typing	no	yes	Numeric data and Strings	partial
XML support	no		no	no
Secondary indexes	yes	yes	no	no
SQL	no	no	no	no
APIs and other access methods	RESTful HTTP/JSON API	Java API RESTful HTTP/JSON API	HTTP API JSON over UDP	proprietary protocol

Illustration 60: db-engines 1

Server-side scripts	View functions (Map-Reduce) in JavaScript	yes	no	Lua
Triggers	yes	yes	no	no
Partitioning methods	Sharding	Sharding	Sharding	Sharding
Replication methods	Master-master replication Master-slave replication	yes	selectable replication factor	Master-slave replication
MapReduce	yes	no	no	no
Consistency concepts	Eventual Consistency	Eventual Consistency		Eventual Consistency
Foreign keys	no	no	no	no
Transaction concepts	no	no	no	Optimistic locking, atomic execution of commands blocks and scripts
Concurrency	yes	yes	yes	yes
Durability	yes	yes	yes	yes
In-memory capabilities	no		yes	yes
User concepts	Access rights for users can be defined per database		simple rights management via user accounts	Simple password-based access control

Illustration 61: db-engines 2

[56]

10 Open Source Stack

10.1 HAProxy

HAProxy is an open source software load balancer for both layer four and layer seven. Written in C and runing on top of Linux, HAProxy uses an event-driven model to reduce memory usage and gain high performance.[57] HAProxy also uses a single-buffering technique to reduce cost memory copy.

Illustration 62: HAProxy Logo

Although HAProxy is high performance, its design goal is to handle a large number of concurrent connections. It can only load balance traffic at the flow level and is unable to vertically divide a single elephant TCP flow into multiple mice flows.[58] HAProxy creates a session for each connection. A session consists of two TCP connections, one from the client to the load balancer and one from the load balancer to the server. The user needs to specify the load balancing policy in a config file before starting HAProxy.

10.1.1 Reasons for using MQTT over Websockets

- Connect directly web applications and things, so you can bootstrap a whole application with very little backend.
- Any other applications that don't want to use the 1883/8883 port and want to go over HTTP/HTTPS instead - this could be so that there is less of a chance of being blocked by a firewall, as most firewalls will let HTTP traffic through

10.2 Mosca MQTT broker

10.2.1 Mosca

Mosca sits between your system and the devices. It is open source and written in Javascript. Mosca is embeddable within any node application, allowing complete customization and the implementation of the pub-sub architecture.[59]

Illustration 63: Mosca Logo

10.2.2 Mosca Features

- MQTT 3.1 and 3.1.1 compliant.
- QoS 0 and QoS 1.
- Various storage options for QoS 1 offline packets, and subscriptions.
- Usable inside any other Node.js app.

10.3 Redis

Redis is an open source, BSD licensed, advanced key-value cache and store. It is often referred to as a data structure server since keys can contain strings, hashes, lists, sets, sorted sets, bitmaps and hyperlogs.[60]

Illustration 64: Redis Logo

Redis is a bit different from Kafka in terms of its storage and various functionalities. At its core, Redis is an in-memory data store that can be used as a high-performance database, a cache, and a message broker. It is perfect for real-time data processing.

Redis also has various clients written in several languages which can be used to write custom programs for the insertion and retrieval of data.[61] This is an advantage over Kafka since Kafka only has a Java client. The main similarity between the two is that they both provide a messaging service. But for the purpose of log aggregation, the various data structures of Redis do it more efficiently.

10.4 InfluxDB

InfluxDB is an open-source time series database developed by InfluxData.[62] It is written in Go and optimized for fast, high-availability storage and retrieval of time series data in fields such as operations monitoring, application metrics, Internet of Things sensor data, and real-time analytics.[63]

Illustration 65: InfluxDB Logo

InfluxDB has no external dependencies and provides an SQL-like language with builtin time-centric functions for querying a data structure composed of measurements, series, and points. Each point consists of several key-value pairs called the fieldset and a timestamp. When grouped together by a set of key-value pairs called the tagset, these define a series. Finally, series are grouped together by a string identifier to form a measurement.[64]

Values can be 64-bit integers, 64-bit floating points, strings, and booleans.

Points are indexed by their time and tagset.

Retention policies are defined on a measurement and control how data is downsampled and deleted.

Continuous Queries run periodically, storing results in a target measurement.

Continuous queries are created when you issue a select statement with an `into` clause. Instead of returning the results immediately like a normal select query, InfluxDB will instead store this continuous query and run it periodically as data is collected. Only cluster and database admins are allowed to create continuous queries.[65]

Continuous queries let you precompute expensive queries into another time series in real-time.

11 Docker-compose

11.1 Project directory structure

```
-Git-repo

  - docker-compose.yml

  - /haproxy

      - Dockerfile

      - haproxy.cfg

  - /mosca

      - /bin

      - /public

      - /lib

      - Dockerfile

      - index.js

      - package.json

  - /influxdb

      - Dockerfile

      - entrypoint.sh

      - influxdb.conf

  - /redis

      - Dockerfile

      - docker-entrypoint.sh
```

11.2 Docker-compose file

```
version: "2"
services:
  haproxy:
    build: ./haproxy
    image: registry.ng.bluemix.net/tomspace2/haproxy-image1
    container_name: ha_name
    depends_on:
    - mosca_srv1
    ports:
    - "80:80"
    - "1883:1883"
    networks:
    - mynetwork
    links:
    - "mosca_srv1"
    - "mosca_srv2"
    environment:
      BACKENDS: "mosca_srv1"
      DNS_ENABLED: "true"
  mosca_srv1:
    build: ./mosca2
    image: registry.ng.bluemix.net/tomspace2/mosca-image1
    depends_on:
    - redis
    networks:
      mynetwork:
        aliases:
          - mosca_srv1
    links:
    - "redis"
  mosca_srv2:
    image: registry.ng.bluemix.net/tomspace2/mosca-image1
    depends_on:
    - redis
    networks:
      mynetwork:
        aliases:
          - mosca_srv2
    links:
    - "redis"
  influxdb:
    build: ./influxdb
    image: registry.ng.bluemix.net/tomspace2/influxdb-image1
    networks:
      mynetwork:
        aliases:
          - influxdb
  redis:
    build: ./redis
    image: registry.ng.bluemix.net/tomspace2/redis-image1
    networks:
      mynetwork:
        aliases:
          - redis
  networks:
    mynetwork:
```

11.3 Docker-compose description

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a Compose file to configure your application's services. Then, using a single command, you create and start all the services from your configuration.

The Compose file is a YAML file defining services, networks and volumes. The default path for a Compose file is `./docker-compose.yml`. A container definition contains configuration which will be applied to each container started for that service, much like passing command-line parameters to docker run.

When you specify image as well as build, Compose names the built image with the name specified and optional tag specified in image. The build command takes the string argument which is a path to a directory containing a Dockerfile. When the value supplied is a relative path, it is interpreted as relative to the location of the Compose file. Compose will build and tag it with a generated name, and use that image thereafter.

The docker-compose.yml file defines 5 services. The first is the HAProxy service. The **`build:`** command tells compose that the dockerfile for this service can be found in the ./haproxy directory.

`image:` specifies the images to start the container from, the location of the image is in the Bluemix registry **`registry.ng.bluemix.net/tomspace2/haproxy-image1`**

The HAProxy container depends on the Mosca MQTT servers because it needs to link with them, so therefore it needs to be built after Mosca.

`ports:` specifies the ports to be exposed. (`HOST:CONTAINER`). In this case the ports 80 and 1883 are exposed for both HTTP and MQTT traffic respectively.

`Networks:` specifies the network the container is to use.

`Links:` Containers for the linked service will be reachable at a hostname identical to the alias "mosca_srv1" and "mosca_srv2".

The mosca_srv1 and mosca_srv2 services both use the same image and they both depend on the redis service. They have their own alias network names "mosca_srv1" and "mosca_srv2".

Influxdb service has no dependencies and currently does not expose any ports, however in the dockerfile for influxdb it exposes port 8086 to other containers in the same network only.

The redis service has a similar configuration to influxdb except that it exposes port 6379 to other containers on the same network.

11.4 HAProxy

11.4.1 Dockerfile

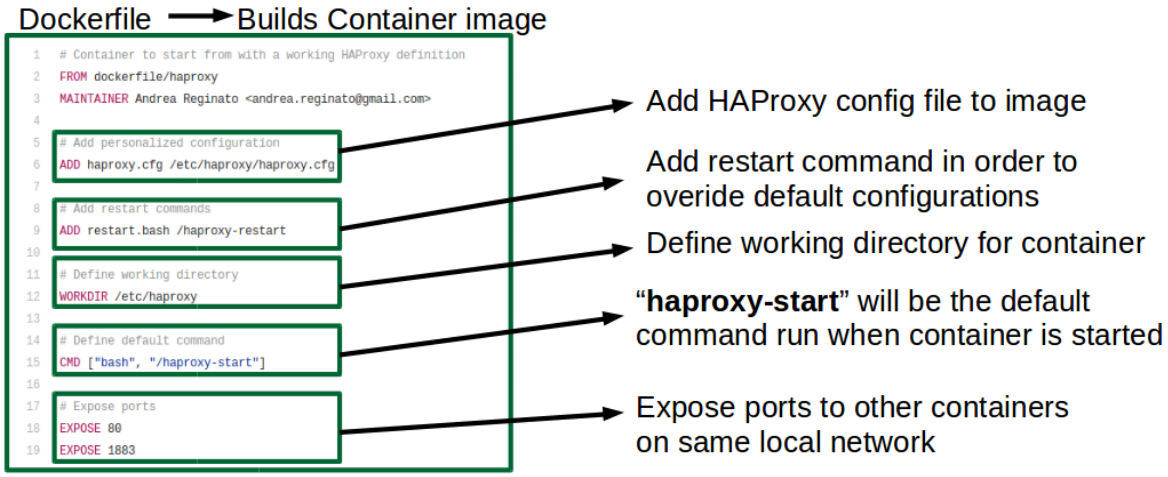


Illustration 66: HAProxy Dockerfile

11.4.2 haproxy.cfg

```
global
    maxconn 2000
    pidfile /var/run/haproxy.pid
    log     127.0.0.1 local0
    log     127.0.0.1 local1 notice

    # echo "" | nc -U /var/run/haproxy.sock
    stats   socket /var/run/haproxy.sock mode 777

+resolvers docker
+  nameserver dns "${DNS_TCP_ADDR}:${DNS_TCP_PORT}"
+
defaults
```

```
frontend web
  bind   *:80
  stats enable
  stats uri / # must be present to see the logs
  stats auth admin:admin

  default_backend mosca

backend mosca
listen mqtt
  bind *:1883
  mode tcp
  option tcplog
  balance leastconn
-  server mosca_1 178.62.122.204:1883 check
-  server mosca_2 178.62.104.172:1883 check
+  server mosca1 mosca_srv1:1883 check resolvers docker resolve-prefer ipv4
+  server mosca2 mosca_srv2:1883 check resolvers docker resolve-prefer ipv4
```

11.4.3 Configuration

Above is the HAProxy config file for load balancing between 2 Mosca Brokers. The HAProxy listens for all requests coming to port 1883 and forwards them to the MQTT containers (mosca-srv1 and mosca-srv2) using the *leastconn* balance mode (selects the server with the least number of connections).

Changes had to be made to the original haproxy.cfg file. The previous implementation used static IP addresses, DNS resoultion was implemented in order to enable developers to rebuild mosca containers without being required to update the haproxy.cfg file every time. This problem took a significant amount of time to solve as multiple issues with getting basic HAProxy functionality working first had to be solved.

When using the ”–link” option, docker creates a new entry in the containers /etc/hosts file with the IP address and name provided by the ”link” directive. Docker will also update this file when the remote container (here mosca_srv1) IP address is changed (IE when restarting the container).

HAProxy doesn’t do file system IOs at run time. Furthermore, HAProxy doesn’t use **/etc/hosts** file directly. The glibc might use it when HAProxy asks for DNS resolution when parsing the configuration file.

If mosca_srv1 IP gets changed, then the **/etc/hosts** file is updated accordingly, however HAProxy is not aware of the change and the application may fail. A quick solution would be to reload **HAProxy** process in its container, to force it taking into account the new IP.

A more reliable solution, is to use HAProxy 1.6 DNS resolution capability to follow-up the IP change. With this purpose in mind, 2 new tools were added to the **HAProxy** container:

In 1.5 and before, HAProxy performed DNS resolution when parsing configuration, in a synchronous mode and using the glibc (hence /etc/resolv.conf file).

Now, HAProxy can perform DNS resolution at runtime, in an asynchronous way and update server IP on the fly. This is very convenient in environment like Docker or Bluemix where server IPs can be changed at any time.

A dnsmasq is used as an interface between /etc/hosts file (where docker stores server IPs) and HAProxy:

```
resolvers docker

  nameserver dnsmasq 127.0.0.1:53
```

Now whenever the IP addresses of “mosca_srv1” and “mosca_srv2” changes, HAProxy is notified. This removes operational complexity away from the developer.

11.5 Mosca

11.5.1 Dockerfile

```
FROM mhart/alpine-node:4

MAINTAINER Matteo Collina <hello@matteocollina.com>

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app/

COPY ./ /usr/src/app/

RUN apk update && \

    apk add make gcc g++ python git && \

    npm install --unsafe-perm --production && \

    apk del make gcc g++ python git

EXPOSE 80

EXPOSE 1883

ENTRYPOINT ["/usr/src/app/bin/server.js", "-d", "/db", "--http-port",
"80", "--http-bundle", "-v"]
```

11.5.2 Configuration

- This Dockerfile builds the image for the Mosca container.
- The image is built on top of the base image mhart.alpine-node:4 found on Dockerhub.
- The directory **`/usr/src/app`** is created within the container and is set as the working directory.
- All files within the /mosca directory are then copied into the image.
- Dependencies are then installed followed by exposing the ports 80 and 1883. This should be expanded to open ports 8080 and 8883 once SSL/TLS functionality is introduced.
- The `ENTRYPOINT` configures the container to run as an executable.

11.5.3 Mosca Configuration

```
var mosca = require('mosca')

var ascoltatore = {
  type: 'redis',
  redis: require('redis'),
  db: 12,
  port: 6379,
  return_buffers: true, // to handle binary payloads
  host: "localhost"
};

var moscaSettings = {
  port: 1883,
  backend: ascoltatore,
  persistence: {
    factory: mosca.persistence.Redis
  }
};

var server = new mosca.Server(moscaSettings);
server.on('ready', setup);

server.on('clientConnected', function(client) {
        console.log('client connected', client.id);
});

// fired when a message is received
server.on('published', function(packet, client) {
  console.log('Published', packet.topic, packet.payload);
});

// fired when the mqtt server is ready
function setup() {
  console.log('Mosca server is up and running')
}
```

Illustration 67: Mosca Redis Configuration

Ascoltatore focuses on providing a simple and unique abstraction for all supported brokers, in this case Redis. The moscaSettings variable is configured to connect to the Redis server using Ascoltatore.

If using a non-standard redis port or redis is running on a different server, it is necessary to set the host and port in the persistence section.

```
persistence: {
  factory: mosca.persistence.Redis
  host: 'your redis host',
  port: 'your redis port'
}
```

Illustration 68: Redis Persistence

11.5.4 Authentication and Authorization

With Mosca it is possible to authorize a client defining three methods.

- **`authenticate`**
- **`authorizePublish`**
- **`authorizeSubscribe`**

These methods can be used to restrict the accessible topics for specific clients. Below is an example of a client that sends a username and a password during the connection phase and where the username will be saved and used later on. (To verify if a specific client can publish or subscribe for the specific user).

```
var authenticate = function(client, username, password, callback) {
  var authorized = (username === 'alice' && password.toString() === 'secret
  if (authorized) client.user = username;
  callback(null, authorized);
}
```

Illustration 69: Mosca Authenticate

The function “authenticate” accepts the connection if the username and password are valid.

```
var authorizePublish = function(client, topic, payload, callback
  callback(null, client.user == topic.split('/')[1]);
}
```

Illustration 70: Mosca Authorize Publish

In this case the client authorized as alice can publish to /users/alice taking the username from the topic and verifying it is the same as the authorized user.

```
var authorizeSubscribe = function(client, topic, callback
  callback(null, client.user == topic.split('/')[1]);
}
```

Illustration 71: Mosca Authorize Subscribe

In this case the client authorized as alice can subscribe to /users/alice taking the username from the topic and verifying it is the same of the authorized user.

With this logic someone that is authorized as 'alice' will not be able to publish to the topic `users/bob`.

This logic will be implemented in future to prevent car IoT devices publishing/subscribing to the “mobile-iotp” topic and similarly prevent mobile phone devices publishing/subscribing to the “car-iotp” topic. It is important to note that the code described in this section was not successfully implemented into the project.

11.6 Redis

11.6.1 Dockerfile

```
FROM alpine:3.6

RUN addgroup -S redis && adduser -S -G redis redis

RUN apk add --no-cache 'su-exec>=0.2'

ENV REDIS_VERSION 4.0.1

ENV REDIS_DOWNLOAD_URL http://download.redis.io/releases/redis-4.0.1.tar.gz

ENV REDIS_DOWNLOAD_SHA
2049cd6ae9167f258705081a6ef23bb80b7eff9ff3d0d7481e89510f27457591

RUN set -ex; \
   apk add --no-cache --virtual .build-deps \
         coreutils \
         gcc \
         linux-headers \
         make \
         musl-dev \
   ; \
   wget -O redis.tar.gz "$REDIS_DOWNLOAD_URL"; \
   echo "$REDIS_DOWNLOAD_SHA *redis.tar.gz" | sha256sum -c -; \
   mkdir -p /usr/src/redis; \
   tar -xzf redis.tar.gz -C /usr/src/redis --strip-components=1; \
   rm redis.tar.gz; \
   grep -q '^#define CONFIG_DEFAULT_PROTECTED_MODE 1$'
/usr/src/redis/src/server.h; \
   sed -ri 's!^(#define CONFIG_DEFAULT_PROTECTED_MODE) 1$!\1 0!'
/usr/src/redis/src/server.h; \
   grep -q '^#define CONFIG_DEFAULT_PROTECTED_MODE 0$'
/usr/src/redis/src/server.h; \
   make -C /usr/src/redis -j "$(nproc)"; \
   make -C /usr/src/redis install; \
   rm -r /usr/src/redis; \
   apk del .build-deps

RUN mkdir /data && chown redis:redis /data

VOLUME /data

WORKDIR /data

COPY docker-entrypoint.sh /usr/local/bin/

ENTRYPOINT ["docker-entrypoint.sh"]

EXPOSE 6379

CMD ["redis-server"]
```

11.7 Influxdb

11.7.1 Dockerfile

```
FROM alpine:3.5
RUN echo 'hosts: files dns' >> /etc/nsswitch.conf
RUN apk add --no-cache tzdata
ENV INFLUXDB_VERSION 1.3.3
RUN set -ex && \
    apk add --no-cache --virtual .build-deps wget gnupg tar ca-certificates
&& \
    update-ca-certificates && \
    for key in \
        05CE15085FC09D18E99EFB22684A14CF2582E0C5 ; \
    do \
        gpg --keyserver ha.pool.sks-keyservers.net --recv-keys "$key" || \
        gpg --keyserver pgp.mit.edu --recv-keys "$key" || \
        gpg --keyserver keyserver.pgp.com --recv-keys "$key" ; \
    done && \
    wget -q https://dl.influxdata.com/influxdb/releases/influxdb-$
{INFLUXDB_VERSION}-static_linux_amd64.tar.gz.asc && \
    wget -q https://dl.influxdata.com/influxdb/releases/influxdb-$
{INFLUXDB_VERSION}-static_linux_amd64.tar.gz && \
    gpg --batch --verify influxdb-${INFLUXDB_VERSION}-
static_linux_amd64.tar.gz.asc influxdb-${INFLUXDB_VERSION}-
static_linux_amd64.tar.gz && \
    mkdir -p /usr/src && \
    tar -C /usr/src -xzf influxdb-${INFLUXDB_VERSION}-
static_linux_amd64.tar.gz && \
    rm -f /usr/src/influxdb-*/influxdb.conf && \
    chmod +x /usr/src/influxdb-*/* && \
    cp -a /usr/src/influxdb-*/* /usr/bin/ && \
    rm -rf *.tar.gz* /usr/src /root/.gnupg && \
    apk del .build-deps
COPY influxdb.conf /etc/influxdb/influxdb.conf
EXPOSE 8086
VOLUME /var/lib/influxdb
COPY entrypoint.sh /entrypoint.sh
ENTRYPOINT ["/entrypoint.sh"]
CMD ["influxd"]
```

11.8 Redis and InfluxDB

The dockerfiles for Redis and InfluxDB successfully build the images for the containers, however the containers do not run when deployed. From the perspective of this theses, the infrastructure has been provided but work to get these components functioning has yet to be completed.

12 Jenkins pipeline

12.1 About Jenkins 2

Jenkins 2 brings Pipeline as code, a new setup experience and other UI improvements all while maintaining total backwards compatibility with existing Jenkins installations.[66]

Jenkins Pipeline is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins. Pipeline provides an extensible set of tools for modeling simple-to-complex delivery pipelines "as code" via the Pipeline DSL.

Jenkins is, fundamentally, an automation engine which supports a number of automation patterns. Pipeline adds a powerful set of automation tools onto Jenkins, supporting use cases that span from simple continuous integration to comprehensive continuous delivery pipelines. By modeling a series of related tasks, users can take advantage of the many features of Pipeline:

- **Code**: Pipelines are implemented in code and typically checked into source control, giving teams the ability to edit, review, and iterate upon their delivery pipeline.
- **Durable**: Pipelines can survive both planned and unplanned restarts of the Jenkins master.
- **Pausable**: Pipelines can optionally stop and wait for human input or approval before continuing the Pipeline run.
- **Versatile**: Pipelines support complex real-world continuous delivery requirements, including the ability to fork/join, loop, and perform work in parallel.
- **Extensible**: The Pipeline plugin supports custom extensions to its DSL and multiple options for integration with other plugins.

The flowchart below is an example of one continuous delivery scenario easily modeled in Jenkins Pipeline:

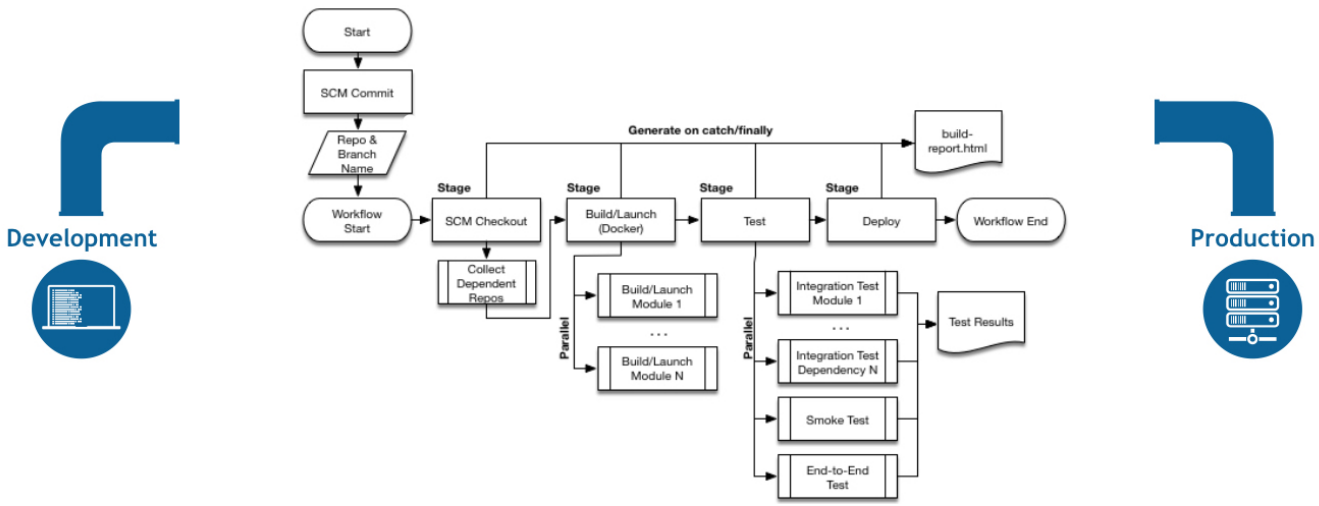


Illustration 72: Jenkins Pipeline

12.2 IBM Cloud DevOps plugin

With this Jenkins plugin, You can publish test results to DevOps Insights, add automated quality gates, and track your deployment risk. You can also send your Jenkins job notifications to other tools in your toolchain, such as Slack and PagerDuty. To help you figure out when code was deployed, the system can add deployment messages to your Git commits and your related Git or JIRA issues. You can also view your deployments on the Toolchain Connections page.[67]

This plugin provides Post-Build Actions and CLIs to support this inteigration. DevOps Insights aggregates and analyzes the results from unit tests, functional tests, code-coverage tools, static security code scans and dynamic security code scans to determine whether your code meets predefined policies at gates in your deployment process. If your code does not meet or exceed a policy, the deployment is halted, preventing risky changes from being released. You can use DevOps Insights as a safety net for your continuous delivery environment, a way to implement and improve quality standards over time, and a data visualization tool to help you understand your project's health.

12.2.1 Environment variables

These environment variables and credentials are used by the IBM Cloud DevOps plugin to interact with DevOps Insights. Here is an example of setting them in the declarative pipeline format.

```
environment {
        IBM_CLOUD_DEVOPS_CREDS = credentials('BM_CRED')
        IBM_CLOUD_DEVOPS_ORG = 'dlatest'
        IBM_CLOUD_DEVOPS_APP_NAME = 'Weather-App'
        IBM_CLOUD_DEVOPS_TOOLCHAIN_ID = '1111111-aaaa-2222-bbbb-
333333333'
IBM_CLOUD_DEVOPS_WEBHOOK_URL = 'https://jenkins:5a55555a-a555-5555-
5555-a555aa55a555:55555555-5555-5555-5555-555555555555@devops-
api.ng.bluemix.net/v1/toolint/messaging/webhook/publish' //Optional
}
```

12.2.2 Adding Cloud DevOps steps

The Cloud DevOps plugin adds four steps to Jenkins pipelines for you to use. These steps can be used in your pipelines to interact with DevOps Insights.

- `publishBuildRecord`, which publishes build information to DevOps Insights
- `publishTestResult`, which publishes test results to DevOps Insights
- `publishDeployRecord`, which publishes deployment records to DevOps Insights
- `evaluateGate`, which enforces DevOps Insights policies

12.2.3 Publishing build records

Publish build records with the `publishBuildRecord` step. This step requires four parameters. It can also accept one optional parameter.

`gitBranch`	The name of the Git Branch that the build uses.
`gitCommit`	The Git commit ID that the build uses.
`gitRepo`	The URL of the Git repository.
`result`	The result of the build stage. The value should be `SUCCESS` or `FAIL`.
buildNumber	*Optional:* The custom build number if you want to specify yourself

12.3 Jenkinsfile (Declarative Pipeline)

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                sh 'make'
            }
        }
        stage('Test'){
            steps {
                sh 'make check'
                junit 'reports/**/*.xml'
            }
        }
        stage('Deploy') {
            steps {
                sh 'make publish'
            }
        }
    }
}
```

agent indicates that Jenkins should allocate an executor and workspace for this part of the Pipeline.[68]

stage describes a stage of this Pipeline.

steps describes the steps to be run in this `stage`

sh executes the given shell command

12.4 Jenkins master setup

The Jenkins master is setup to run on the local Ubuntu dev machine. This master should be migrated to Kubernetes in future so multiple developers can access it.

12.4.1 Jenkins credentials

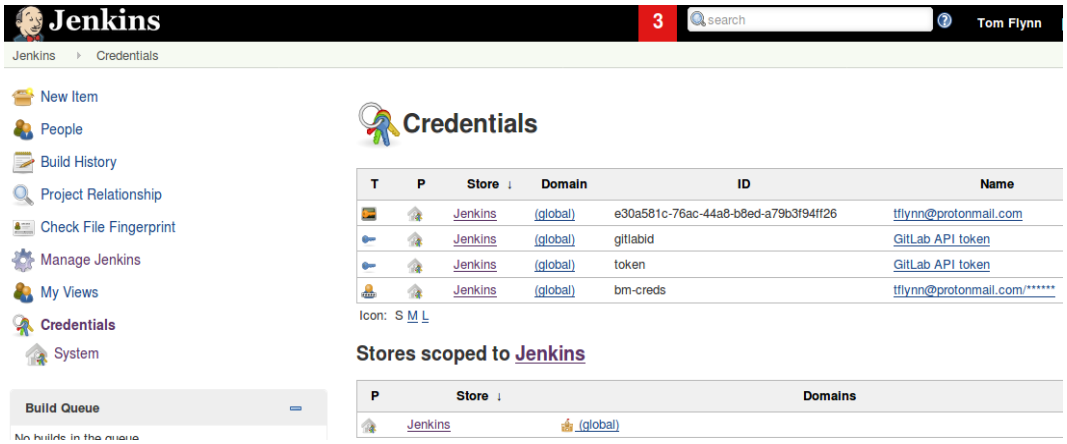


Illustration 73: Jenkins Credentials

12.4.2 Jenkins pipeline configuration

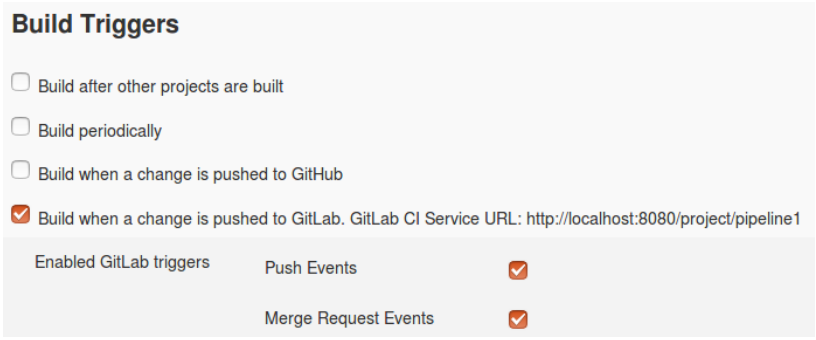


Illustration 74: Build triggers

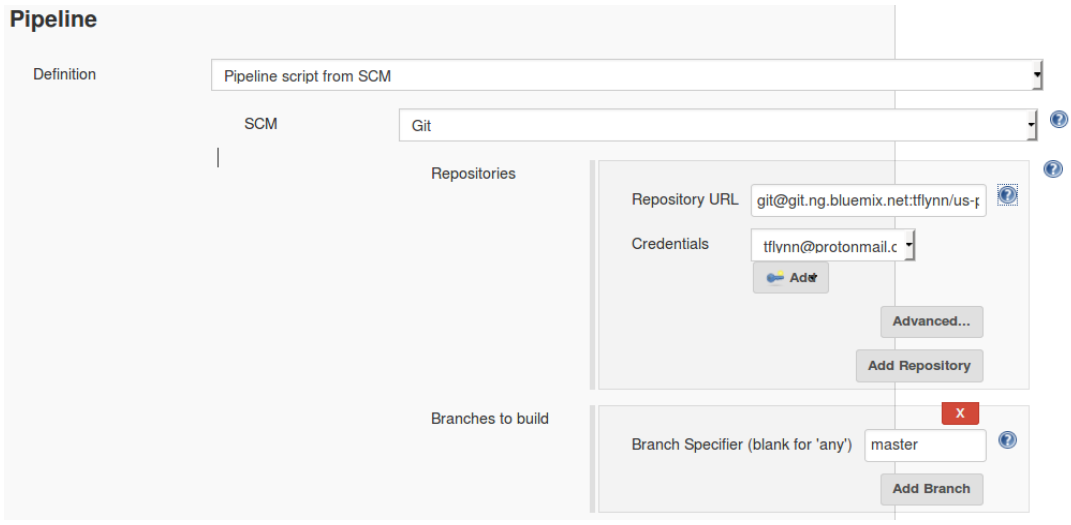


Illustration 75: Pipeline configuration

The pipeline is configured as “Pipeline script from SCM”. This definition is in the Jenkinsfile.

12.5 Jenkinsfile

```
pipeline {
   agent any
   environment {
          IBM_CLOUD_DEVOPS_CREDS = credentials('bm-creds')
          IBM_CLOUD_DEVOPS_ORG = 'tom1'
          IBM_CLOUD_DEVOPS_APP_NAME = 'us-nodered-app'
          IBM_CLOUD_DEVOPS_TOOLCHAIN_ID = '5a81013a-9b2d-4e17-a53b-1a908a213914'
          IBM_CLOUD_DEVOPS_WEBHOOK_URL = 'https://jenkins:deb89eb9-8061-4db6-b43a-
1a6eb8362b27:71020b52-212e-4184-b1c9-0f5072768803@devops-
api.ng.bluemix.net/v1/toolint/messaging/webhook/publish'
   }
    stages {
        stage('Build') {
          environment {
                GIT_COMMIT = sh(returnStdout: true, script: 'git rev-parse HEAD').trim()
                GIT_BRANCH = 'master'
                GIT_REPO = 'git@git.ng.bluemix.net:tflynn/us-pipeline-repo.git'
          }
          steps {
                sh 'bx api https://api.ng.bluemix.net'
                sh 'bx login -u $IBM_CLOUD_DEVOPS_CREDS_USR -p
$IBM_CLOUD_DEVOPS_CREDS_PSW -o $IBM_CLOUD_DEVOPS_ORG -s space-us'
                sh '''
                       sudo bx ic init
                        sudo bx cr login
                       docker-compose build
                '''
          }
          post {
                 success {
                      publishBuildRecord gitBranch: "${GIT_BRANCH}", gitCommit: "$
{GIT_COMMIT}", gitRepo: "${GIT_REPO}",  result:"SUCCESS"
                 }
                 failure {
                      publishBuildRecord gitBranch: "${GIT_BRANCH}", gitCommit: "$
{GIT_COMMIT}", gitRepo: "${GIT_REPO}", result:"FAIL"
                 }
            }#end post
       }#end Build stage

       stage('testing') {
            steps {
                echo 'Performing integration tests....'
            }
            post {
                  success {
                      echo 'integration passed all tests'
                  }
                  failure {
                      echo 'integration failed to pass all tests'
                  }
            }#end post
       }#end test stage

       stage('Deploy') {
            steps {
                sh 'COMPOSE_HTTP_TIMEOUT=200 docker-compose -p projectName up'
                sh 'bx ic ip-bind 169.46.159.224 ha_name'
            }
       }#end deploy stage
   }#end pipeline
```

12.6 Pipeline description

The pipeline is divided into 3 stages, build, test, and deploy.

The `agent` section specifies where the entire Pipeline, or a specific stage, will execute in the Jenkins environment depending on where the `agent` section is placed. The line **`"agent any"`** means execute the Pipeline, or stage, on any available agent.

The environment variables and credentials are used by the IBM Cloud DevOps plugin to deploy the CloudFoundry Node-RED application “us-nodered-app”.

There are multiple steps within the build stage. The steps section defines a series of one or more steps to be executed in a given stage directive.

```
sh 'bx api https://api.ng.bluemix.net'
```

Executes a shell command to set the API endpoint.

```
sh 'bx login -u $IBM_CLOUD_DEVOPS_CREDS_USR -p $IBM_CLOUD_DEVOPS_CREDS_PSW -o $IBM_CLOUD_DEVOPS_ORG -s space-us'
```

Logs into Bluemix in the us-south region using the Jenkins credentials stored in the Jenkins master.

```
sudo bx ic init
```

Initialize IBM Containers CLI.

```
sudo bx cr login
```

Login into the IBM Bluemix Container Registry.

```
docker-compose build
```

Executes the build commands in the docker-compose file, creating the latest version of the container images.

The `post` section defines actions which will be run at the end of the Pipeline run or stage. A number of post-condition blocks are supported within the `post` section: `always`, `changed`, `failure`, `success`, `unstable`, and `aborted`. These blocks allow for the execution of steps at the end of the Pipeline run or stage, depending on the status of the Pipeline. On successful completion of the build stage a build record is published to the Gitlab repository.

The testing stage merely echoes a message that integration tests are being performed. Given more time, a simple test would have been implemented here.

The deploy stage has two commands for deploying the docker-compose application on Bluemix.

```
sh 'COMPOSE_HTTP_TIMEOUT=200 docker-compose -p projectName up'
```

This command spins up the containers defined in the docker-compose.yaml file. The http timeout is set to 200 to overide the default value of 60, this was done to prevent the deployment from failing during periods poor network conditions.

```
sh 'bx ic ip-bind 169.46.159.224 ha_name'
```

Once the containers are deployed on Bluemix, HAProxy needs to be given a public IP address. This part of the pipeline needs to be automated using the command "ip request" which returns an available public IP address that a container can use. However at the moment the IP address is hardcoded into the pipeline. The deployment might fail when the IP address 169.46.159.224 is not available.

12.7 Future work

The pipeline needs considerable work but considering the time constraints and learning curve, this is a reasonable solution as it meets the minimum requirements for a CI/CD pipeline.

The DevOps insights functionality was not implemented in this solution. The plugin has quite a lot of powerful features for integrating with Bluemix such as image scan security which is crucial for continuous deployment. A working CD pipeline on Bluemix was put together in the Spring semester however the build times were considerably slow because each job in the pipeline had to install the Bluemix CLI everytime. Jenkins was chosen over this because of it's extensive plugin catalog.

Docker-compose is perfectly fine for building and testing, however Kubernetes should be considered for deployment as it addresses many of the problems found in production. Kuberenetes would have been implemented into this project except for the fact that the free cluster available on bluemix will not accept non HTTP traffic. In order to expose a Kubernetes service on Bluemix on port 1883, you need an ingress filter which is not available in the free cluster.

The Node-RED app needs to be integrated into the pipeline. One solution is to put the app into a container and then simply link it to HAProxy. This will also resolve the problem of updating the broker IP address within Node-RED as well as connecting over public IP.

13 Implementation and testing

13.1 Mosca implementation

13.1.1 IBM Container Service Login

Illustration 76: Bluemix Logo

```
tom@tom-pc:~/bluemixcli/Bluemix_CLI$ cf login -a api.eu-gb.bluemix.ne
API endpoint: api.eu-gb.bluemix.net

Email> 16117743@studentmail.ul.ie

Password>
```

Illustration 77: Container Service Login

13.1.2 Setting Namespace and Authentication with IBM Container's Registry

```
tom@tom-pc:~/BM/mosca/mosca$ cf ic namespace set tomspace2
tomspace2
tom@tom-pc:~/BM/mosca/mosca$ cf ic init
Deleting old configuration file...
Generating client certificates for IBM Containers...
Storing client certificates in /home/tom/.ice/certs/...

Storing client certificates in /home/tom/.ice/certs/containers-api.ng.bluemix.net/f3

OK
The client certificates were retrieved.

Checking local Docker configuration...
OK

Authenticating with the IBM Containers registry host registry.ng.bluemix.net...
OK
You are authenticated with the IBM Containers registry.
Your organization's private Bluemix registry: registry.ng.bluemix.net/tomspace2
```

Illustration 78: Setting Namespace

13.1.3 Bluemix Dashboard

Illustration 79: Bluemix Dashboard

13.1.4 Cloning Mosca Repository From Github

```
tom@tom-pc:~/BM/mosca$ git clone https://github.com/mcollina/mosca.gi
Cloning into 'mosca'...
remote: Counting objects: 4681, done.
```

Illustration 80: Cloning Mosca Repository

13.1.5 Pushing Mosca Image to the "tomspace2" Registry

```
tom@tom-pc:~/BM/mosca/mosca$ cf ic build -t mosca-image
Sending build context to Docker daemon 7.669 MB
Step 1 : FROM mhart/alpine-node:4
4: Pulling from mhart/alpine-node
```

Illustration 81: Pushing Mosca Image

13.2 Mosca testing

The Mosca image was pushed to the Bluemix image registry after testing the functionality of the Mosca Broker on the local Ubuntu machine. The test consisted of using the Mosquitto service installed on the local machine. Mosquitto verified the Mosca Broker's functionality by publishing and subscribing in different terminals.

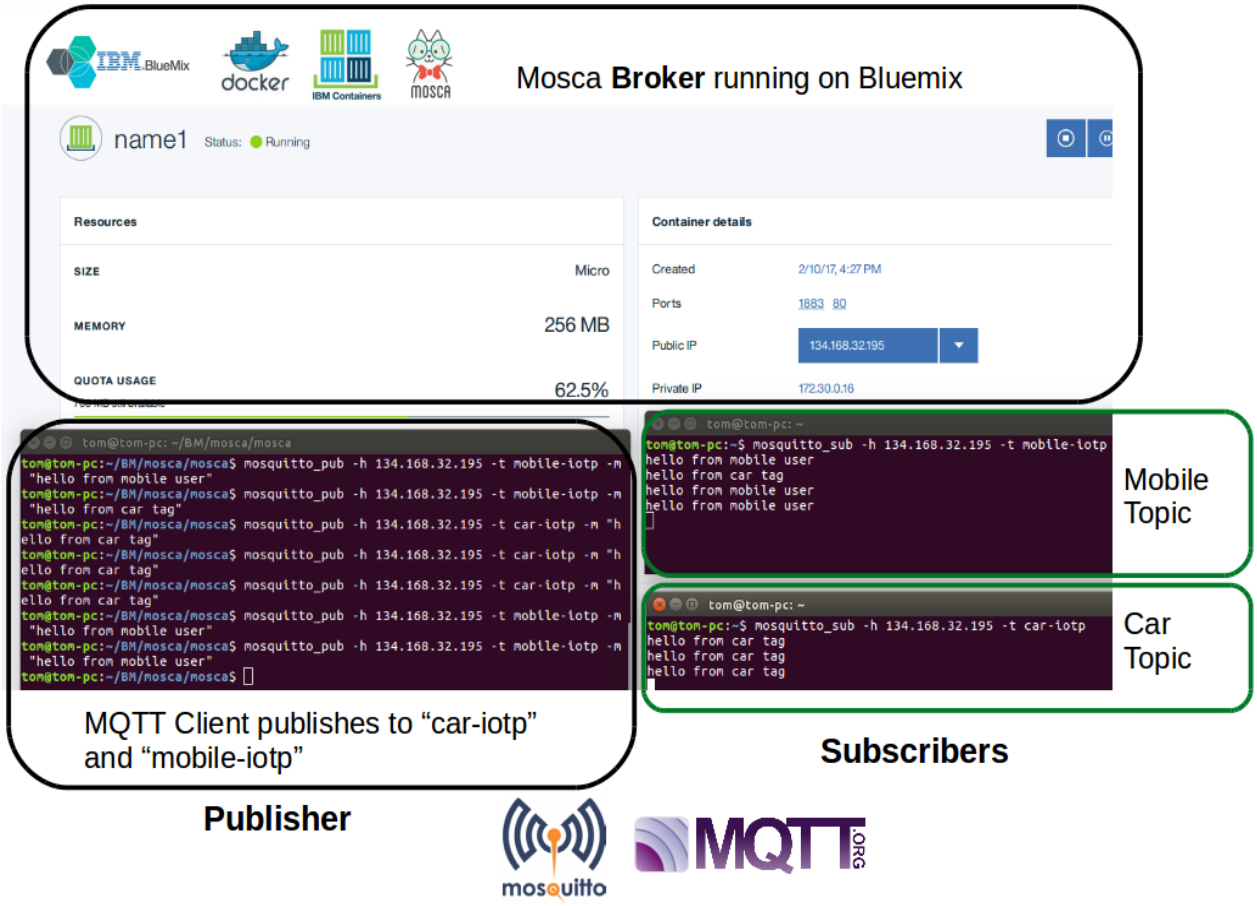


Illustration 82: Mosca Broker Test

13.2.1 Broker

The top half of the above figure shows the Bluemix Dashboard. It displays the Mosca Broker running in a Container called "name1". It has the ports 80 and 1883 exposed.

13.2.2 Mosquitto

The terminal on the left is using Mosquito to publish to the topics "mobile-iotp" and "car-iotp". These messages are being sent to the Mosca Broker Container running on Bluemix.

The terminals on the right are using Mosquito to subscribe to the mobile-iotp and car-iotp topics.

13.3 HAProxy

13.3.1 Testing

A Node-RED flow was configured to inject MQTT messages periodically to the public IP of the HAProxy container. To confirm that the HAProxy was functioning, a second Node-RED flow was used to subscribe to the publishing topic and IP address of the HAProxy container. Despite the fact that the messages are being divided between the two separate containers, all message were received by the subscribing Node-RED flow.

This test is sufficient for confirming basic functionality but a more detailed approach is needed to measure all the various metrics involved such as Quality of service, load balancing method (least connection, round robin) and message frequency. These tests require knowledge of the exact number of messages sent vs received, by comparing these two metrics we can gain valuable insight from various configurations of the HAProxy.

13.3.2 HAProxy before test

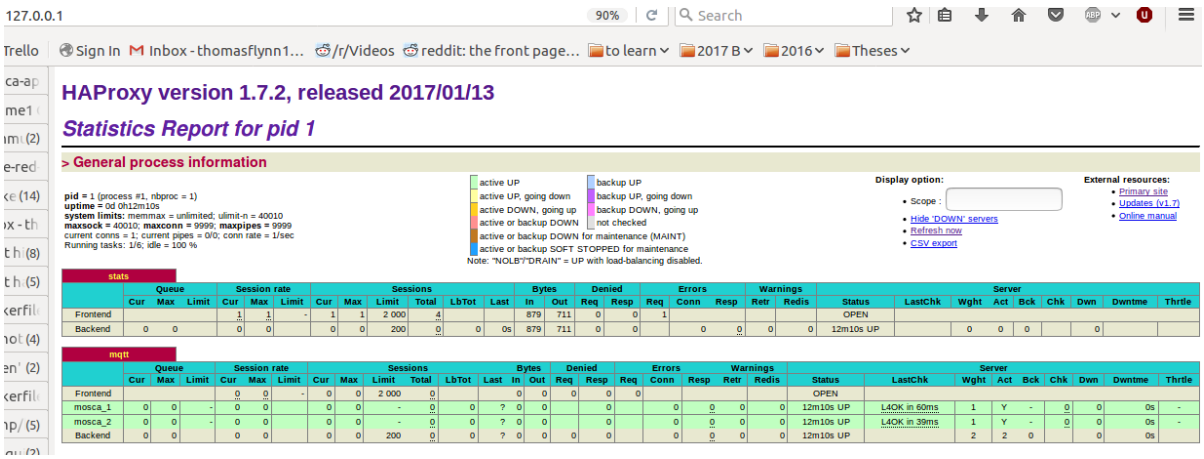


Illustration 83: HAProxy before test

13.3.3 HAProxy after test

Using the admin portal it can be observed in illustration 84 that traffic is indeed being load balanced between the 2 Mosca brokers.

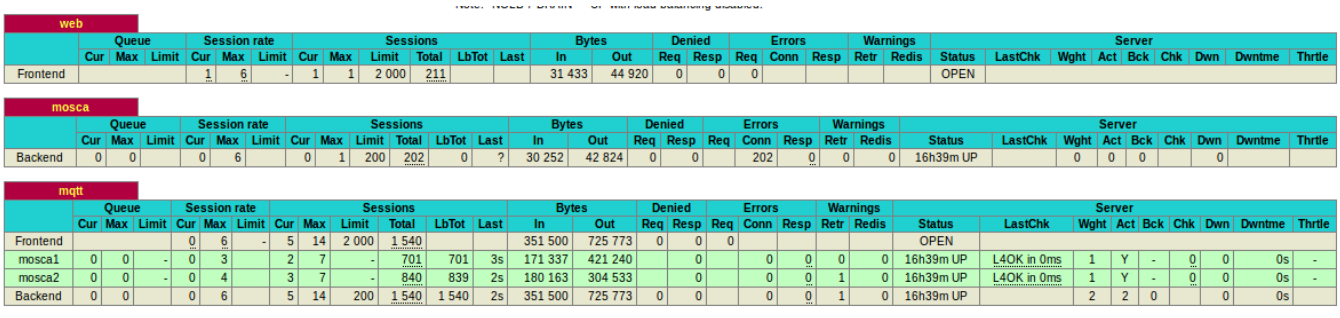


Illustration 84: HAProxy after test

14 IBM Bluemix Container Service

14.1 Monitoring and logging

IBM's platform logging and monitoring solution runs on robust and popular open source offerings today. Logging is delivered from the Elasticsearch ELK stack. Monitoring is delivered from a Graphite and Grafana stack. Both are configured to use a common high-speed Kafka bus.

Illustration 85: IBM Containers Logo

When data is collected from IBM Bluemix Container Service, users benefit from the use of a system crawler. The system crawler automatically collects select compute instance-related logs and metrics. Additionally, the crawler can be configured to collect more, all without agents or sidecar configurations.

Data flows into the logging and monitoring service through an initial Logstash connection point. After the data is in the private network, the data is processed then stored in Elastic Search or Graphite. Users gain access to the data through the Multitenant Proxy.[69]

14.1.1 Monitoring Overview

Container metrics are collected from outside of the container, without having to install and maintain agents inside of the container. In-container agents can have significant overheads and setup times for short-lived, lightweight cloud instances and auto-scaling groups, where containers can be rapidly created and destroyed. This out-of-band data collection approach eliminates these challenges and removes the burden of monitoring from the users.[69]

14.1.2 Logging Overview

Similar to metrics, container logs are monitored and forwarded from outside of the container by using crawlers. The data is sent by the crawlers to a multi-tenant Elasticsearch in Bluemix, just like logs that are collected by other in-container agents, but without the hassle of having to install the agents inside the container.[69]

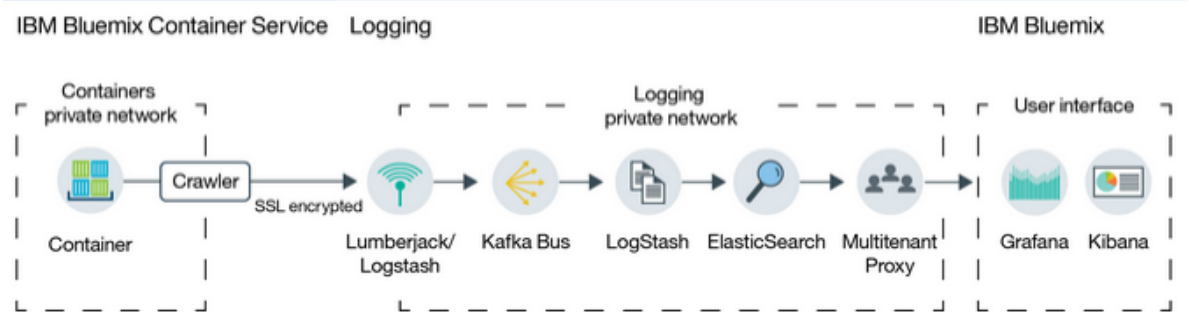


Illustration 86: Bluemix Container Service Logging

14.2 HAProxy Logging and monitoring

Below is a screenshot of the HAProxy container running on Bluemix. Metrics such as CPU usage, memory utilization and bytes entering the container can be observed.

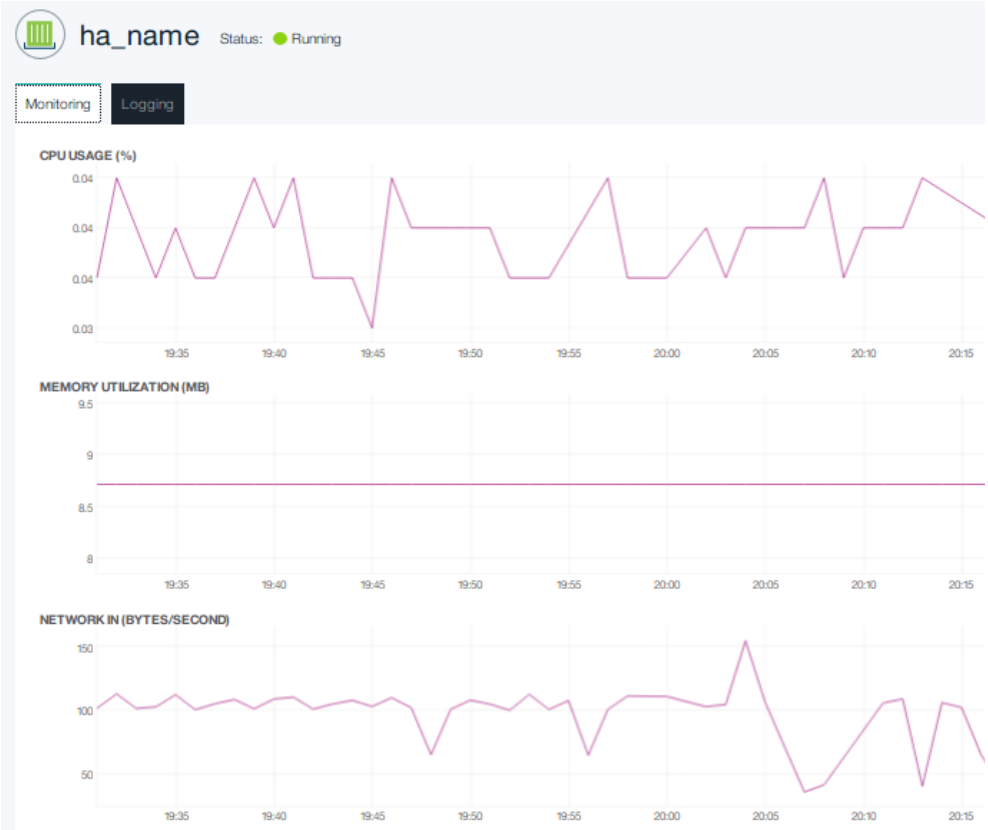


Illustration 87: HAProxy monitoring

14.3 Image security

Below is a screenshot of the container images in the “tomspace2” registry. Attached to each image is a security report.

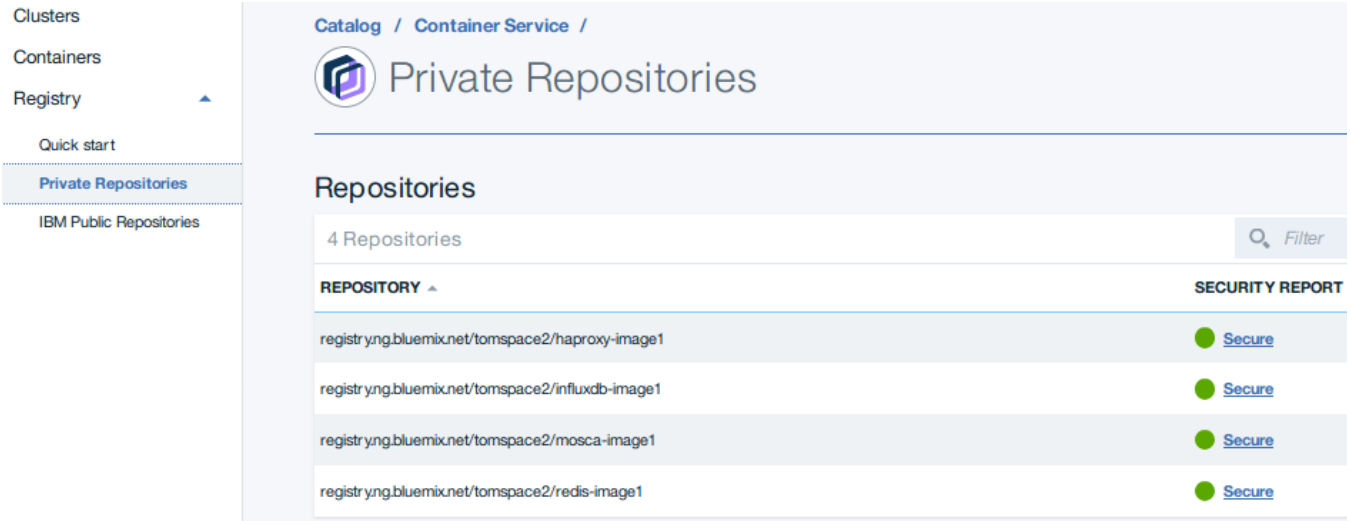


Illustration 88: Image Security

14.3.1 Image policy status

Below is a screenshot of the policy status of the Mosca image. 3 policies can be observed.

- Image has installed packages with know vulnerabilities
- Image has remote logins enabled
- Image has remote logins enabled and some users have easily guessed passwords

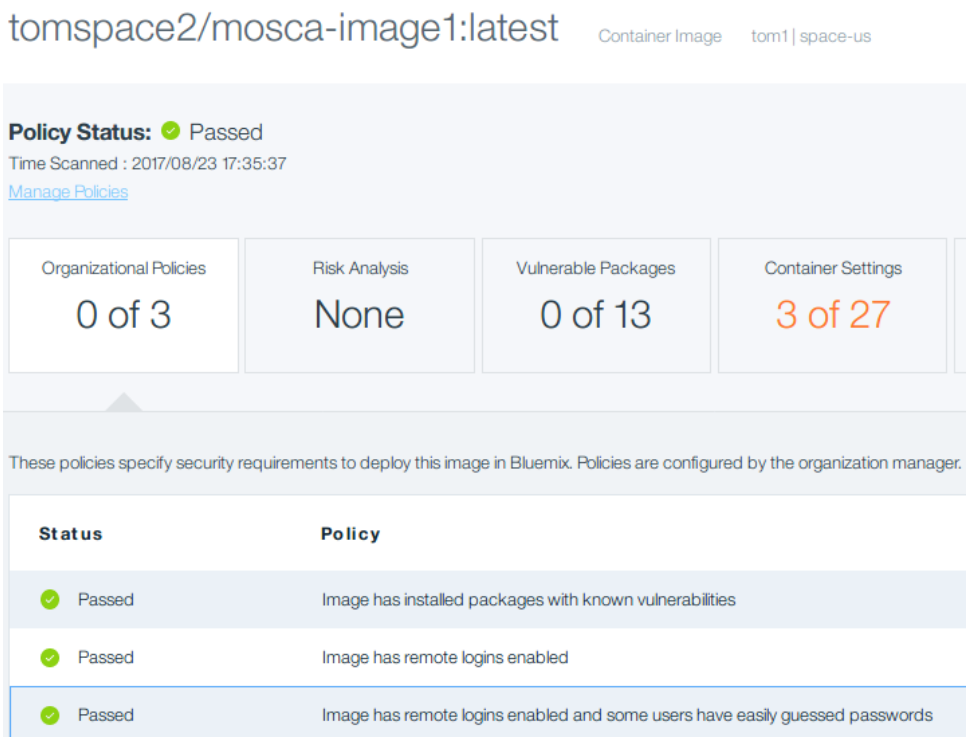


Illustration 89: Image Policy Status

14.4 Vulnerability advisor

Observe the actions that can be enabled for an image policy-- warn or block.

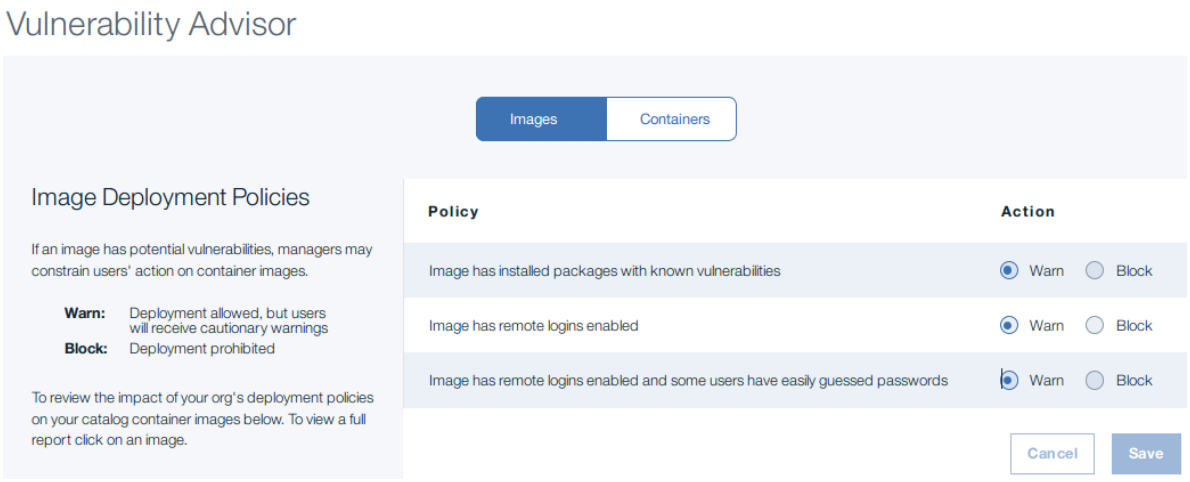


Illustration 90: Vulnerability Advisor

15 Application

15.1 Infrastructure use case

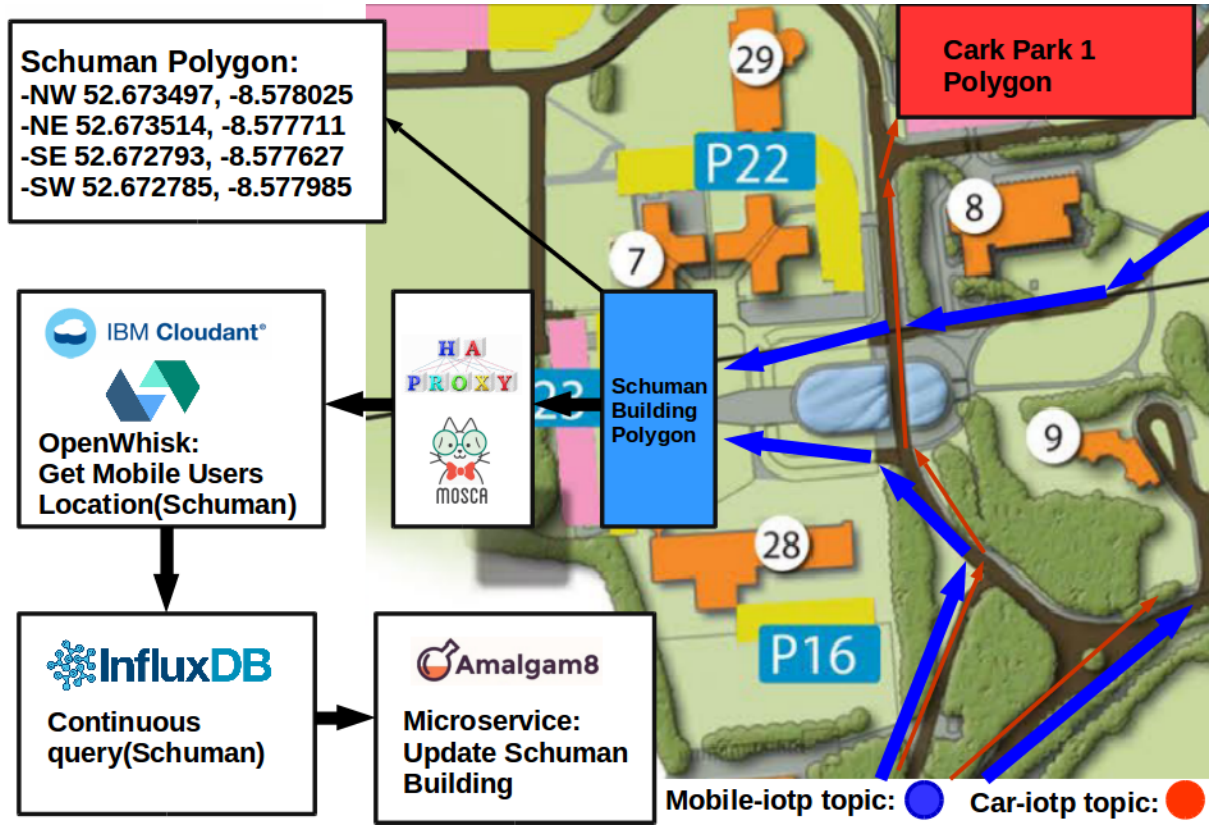


Illustration 91: Use Case Schuman Building

The above figure illustrates how data flows through the system. OpenWhisk is used to perform spatial queries on the Cloudant database. This has the advantage of populating the time-series database with strictly relevant information for the associated downstream microservices in the Amalgam8 A8 registry.

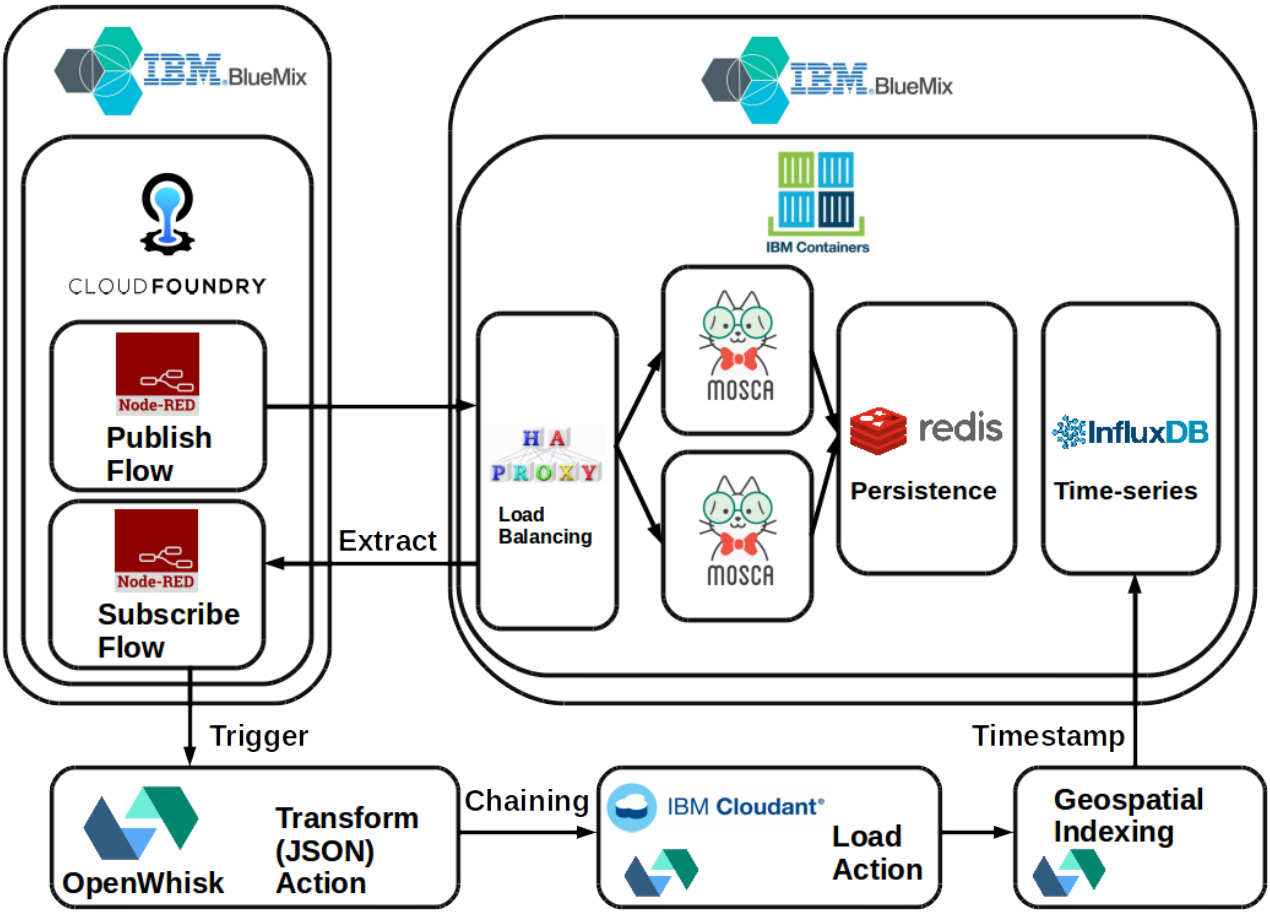
Using the DevOps features within Amalgam8, future students will be able to experiment with different versions of microservices. For example, 80% of mobile users would receive an update from “Update Schuman Building V1” while the other 20% would receive an update from “Update Schuman Building V2”. This is known as A/B testing and will be a crucial aspect of developing and testing future smart campus applications.

The continuous integration pipeline will prevent insecure changes made by developers from reaching the deployment stage if any of the automated tests fail. The developer also doesn’t have to wait for long security scans that would otherwise slow down development. Once the developer gets quick feedback from the integration pipeline, they can continue working on other features while they wait for their previous commit to be pushed through to the deployment stage.

15.2 Application integration

Below is an architectural diagram that shows how a Cloud Foundry application could be integrated with the container infrastructure. The publish flow would be used for injecting messages to the broker and the subscribe flow would be responsible for triggering the corresponding OpenWhisk action. A series of OpenWhisk are triggered, these actions can be grouped together into whats called a “package”. The purpose of the package is to transform the GPS sensor readings into JSON and store it in Cloudant. Whenever data is stored in the Cloudant database, an OpenWhisk action is triggered, this action performs geospatial indexing and then timestamps the data before storing it in the InfluxDB time-series database.

It is important to highlight that this functionality has not been implemented into the project as focus towards infrastructure was prioritized.



16 Node-RED and OpenWhisk

16.1 Node-RED

Node-RED provides a browser-based flow editor that makes it easy to wire together flows using the wide range of nodes in the palette. Flows can be then deployed to the runtime in a single-click. While Node-Red is based on Node.js, JavaScript functions can be created within the editor using a rich text editor. A built-in library allows you to save useful functions, templates or flows for re-use.[70]

Illustration 92: Node-RED logo

16.1.1 Flows

Node-RED programs or flows are a collection of nodes wired together to exchange messages. Under the hood, a flow consists of a list of JavaScript objects that describe the nodes and their configurations, as well as the list of downstream nodes they are connected to, the wires.[70]

16.1.2 Messages

Messages passed between nodes in Node-RED are, by convention, JavaScript Objects called msg, consisting of a set of named properties. These messages often contain a msg.payload property with the payload of the message. Nodes may attach other properties to a message, which can be used to carry other information onto the next node in the flow. When this happens, these extra properties will be documented in the node documentation that appears in the node info pane when you select a node in the Node-RED workspace.[70]

16.1.3 Nodes

Nodes are blocks that represent components of a larger system, in Node-RED's case usually the devices, software platforms and web services that are to be connected. Further blocks can be placed in between these components to represent software functions that wrangle and transform the data in transit.[70]

there are three core node types:

- Input nodes – generate messages for downstream nodes.
- Output nodes – consume messages, for example to send data to an external service or pin on a device, and may generate response messages.
- Processing nodes – messages that process data in some way, emitting new or modified messages.

16.2 IBM OpenWhisk

The OpenWhisk serverless architecture accelerates development as a set of small, distinct, and independent actions. By abstracting away infrastructure, OpenWhisk frees members of small teams to rapidly work on different pieces of code simultaneously, keeping the overall focus on creating user experiences customers want.[71]

Illustration 93: OpenWhisk Logo

16.2.1 key architectural concepts:

- **Triggers:** A class of events emitted by event sources.

- **Actions:** Encapsulate the actual code to be executed which support multiple language bindings including NodeJS, Swift and arbitrary binary programs encapsulated in Docker Containers. Actions invoke any part of an open ecosystem including existing Bluemix services for analytics, data, cognitive, or any other 3rd party service.

- **Rules:** An association between a trigger and an action
  .
- **Packages:** Describe external services in a uniform manner.

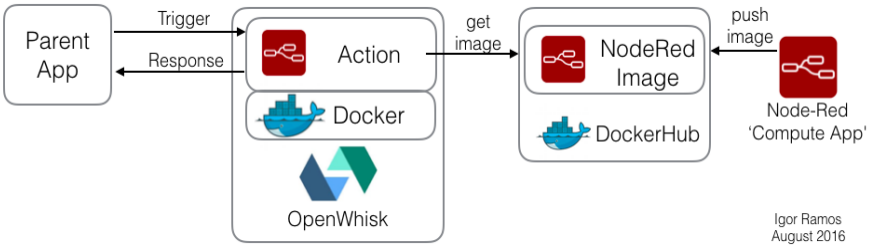


Illustration 94: OpenWhisk Diagram 1

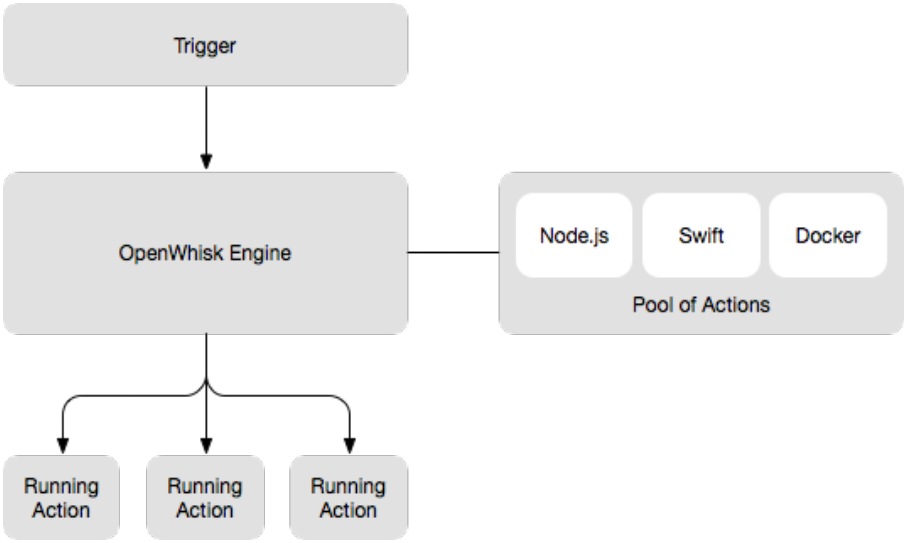


Illustration 95: OpenWhisk Diagram 2

16.3 Node-Red Application

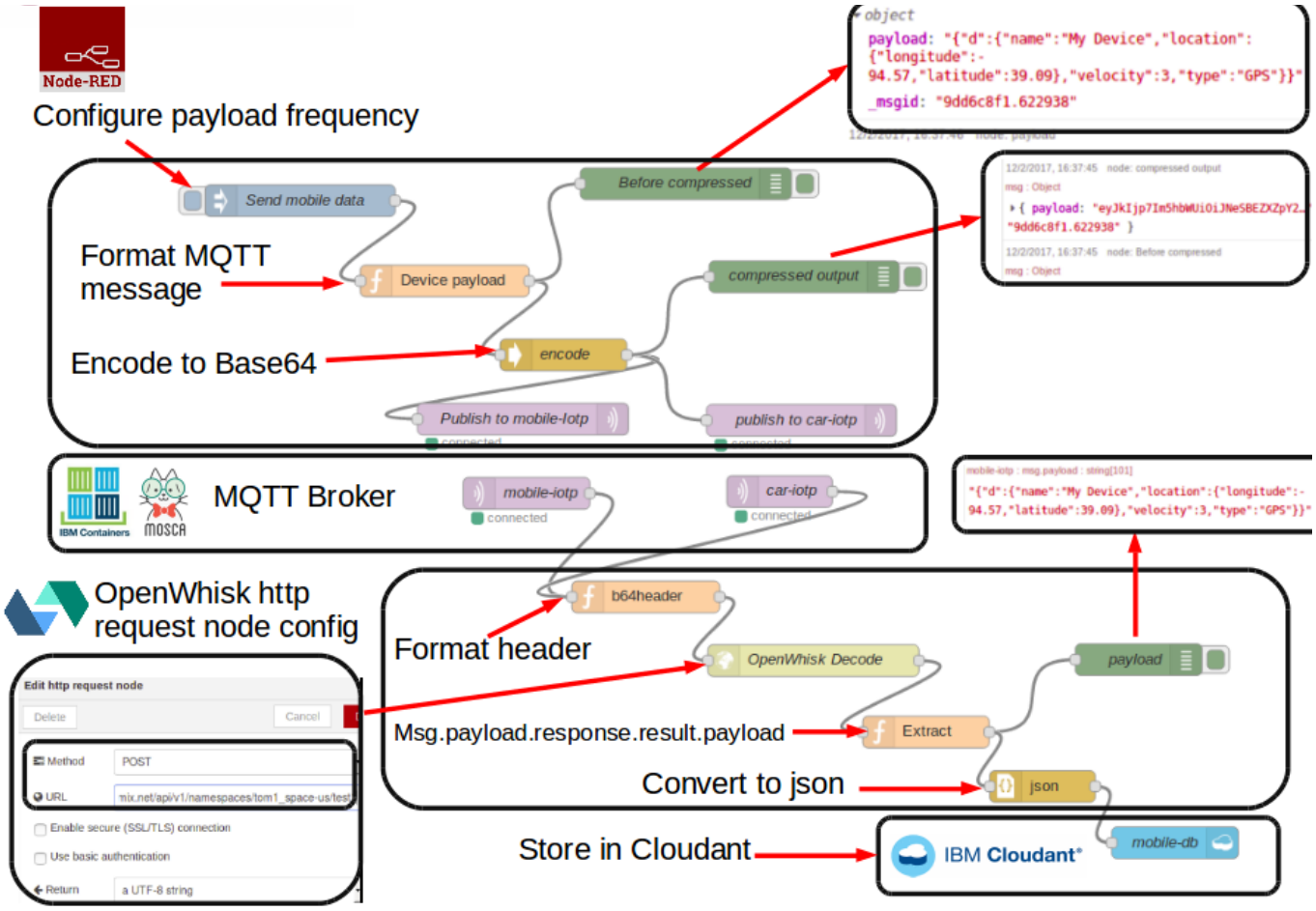


Illustration 96: Primary Flow

16.3.1 Primary Flow

This flow represents the the core application functionality of the theses. Data is flowing from the simulated IoT gateway to the Mosca broker hosted on Bluemix. The base64 encoded data is then decoded using OpenWhisk and stored as JSON in the Cloudant database.

16.3.2 Test Cloudant Flow

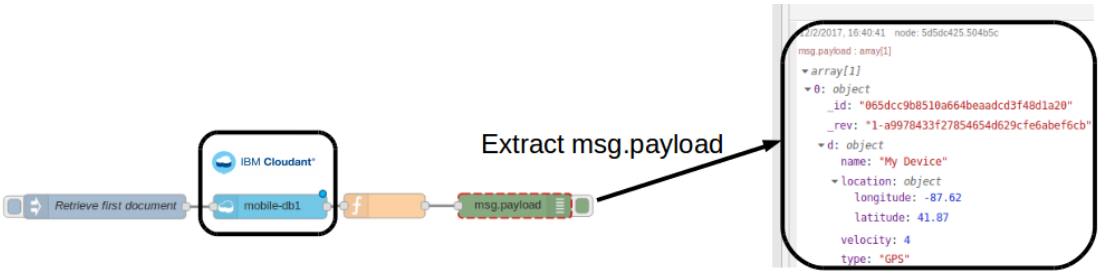


Illustration 97: Test Cloudant Flow

The above flow verifies that data has reached the Cloudant database in JSON format.

16.4 OpenWhisk Action

16.4.1 Base64 Decode Action Configuration

Illustration 98: OpenWhisk Decode Action

16.4.2 Node-RED http post configuration

Node-RED has an OpenWhisk node available that makes it straightforward to call actions, however invoking the action through the http request node was done as a quick fix for the problems encountered using the OpenWhisk node.

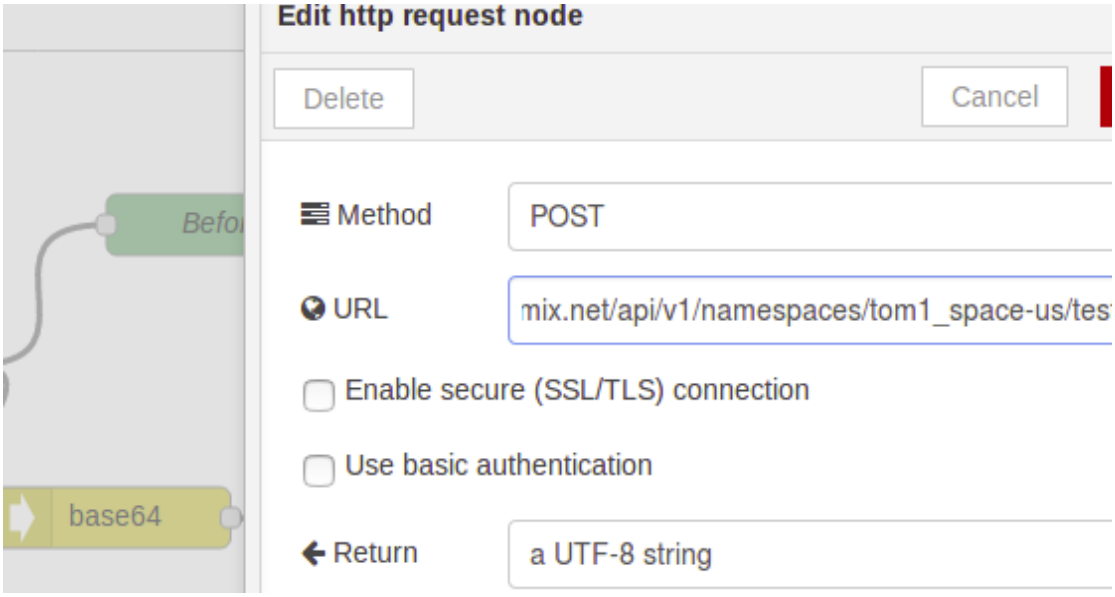


Illustration 99: Http Request Node

17 The leading edge

While carrying out research for this theses, many new exciting technologies were discovered. The two that stand out and that are believed to become vital in the coming years are Istio and Iota.

17.1 Istio

On May 24, 2017, IBM and Google announced the launch of Istio, an open technology that provides a way for developers to seamlessly connect, manage and secure networks of different microservices—regardless of platform, source or vendor.[72]

Illustration 100: Istio Logo

Istio is the result of a joint collaboration between IBM, Google and Lyft as a means to support traffic flow management, access policy enforcement and the telemetry data aggregation between microservices. It does all this without requiring developers to make changes to application code by building on earlier work from IBM, Google and Lyft. Istio currently runs on Kubernetes platforms, such as the IBM Bluemix Container Service. Its design, however, is not platform specific.

As microservices scale dynamically, problems such as service discovery, load balancing and failure recovery become increasingly important to solve uniformly. The individual development teams manage and make changes to their microservices independently, making it difficult to keep all of the pieces working together as a single unified application.

Before combining forces, IBM, Google, and Lyft had been addressing separate, but complementary, pieces of the overall problem.

- **IBM's Amalgam8 project**, a unified service mesh that was created and open sourced last year, provided a traffic routing fabric with a programmable control plane to help its internal and enterprise customers with A/B testing, canary releases, and to systematically test the resilience of their services against failures.
- **Google's Service Control** provided a service mesh with a control plane that focused on enforcing policies such as ACLs, rate limits and authentication, in addition to gathering telemetry data from various services and proxies.
- **Lyft developed the Envoy proxy** to aid their microservices journey, which brought them from a monolithic app to a production system spanning 10,000+ VMs handling 100+ microservices. IBM and Google were impressed by Envoy's capabilities, performance, and the willingness of Envoy's developers to work with the community.

IBM, Google and Lyft realised it would be beneficial to combine their efforts by creating a first-class abstraction for routing and policy management in Envoy, and expose management plane APIs to control Envoys in a manner that can be easily integrated with CI/CD pipelines. In addition to developing the Istio control plane, IBM also contributed several features to Envoy such as traffic splitting across service versions, distributed request tracing with Zipkin and fault injection. Google hardened Envoy on several aspects related to security, performance, and scalability.[72]

17.1.1 Architecture

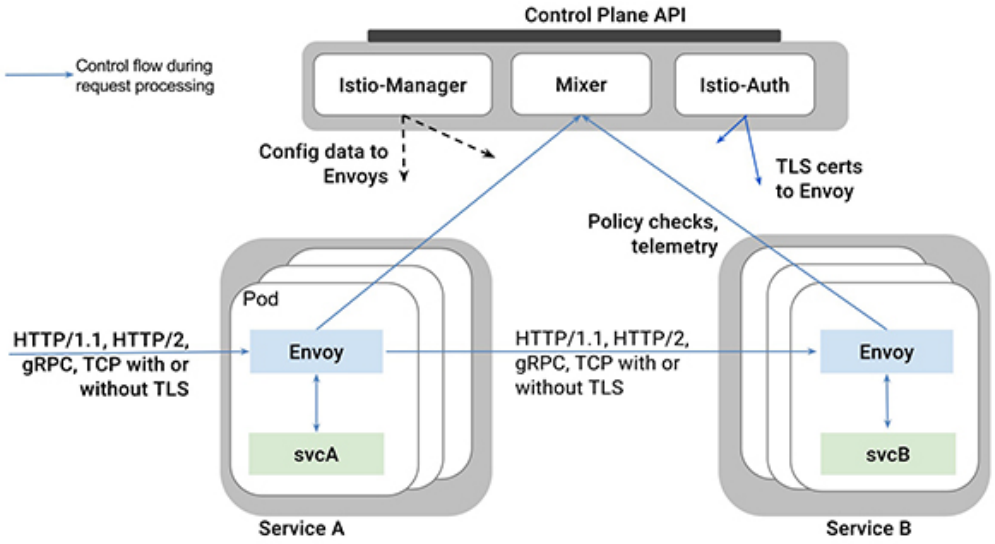


Illustration 101: Istio Architecture

Istio converts disparate microservices into an integrated service mesh by introducing programmable routing and a shared management layer. By injecting Envoy proxy servers into the network path between services, Istio provides sophisticated traffic management controls such as load-balancing and fine-grained routing. This routing mesh also enables the extraction of a wealth of metrics about traffic behavior, which can be used to enforce policy decisions such as fine-grained access control and rate limits that operators can configure. Those same metrics are also sent to monitoring systems. This way, it offers improved visibility into the data flowing in and out of apps.

It is possible to enforce authentication and authorization between any pair of communicating services. Today, the communication is automatically secured via mutual TLS authentication with automatic certificate management.[72]

- Automatic zone-aware load balancing and failover for HTTP/1.1, HTTP/2, gRPC, and TCP traffic. Fine-grained control of traffic behavior with rich routing rules, fault tolerance, and fault injection.
- A pluggable policy layer and configuration API supporting access controls, rate limits and quotas. Automatic metrics, logs and traces for all traffic within a cluster, including cluster ingress and egress.
- Secure service-to-service authentication with strong identity assertions between services in a cluster.

17.1.2 Auth

Istio Auth's aim is to enhance the security of microservices and their communication without requiring service code changes.[73] It is responsible for:

- Providing each service with a strong identity that represents its role to enable interoperability across clusters and clouds
- Securing service to service communication
- Providing a key management system to automate key and certificate generation, distribution, rotation, and revocation

17.1.3 Auth Architecture

The illustration 102 shows the Istio Auth architecture, which includes three components: identity, key management, and communication security. It describes how Istio Auth is used to secure service-to-service communication between service A, running as service account "foo", and service B, running as service account "bar".

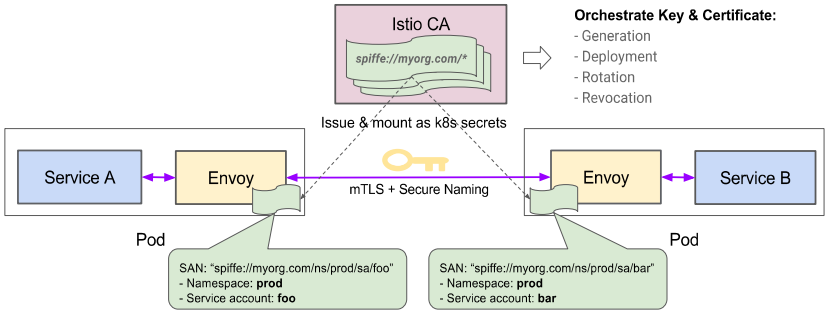


Illustration 102: Auth Architecture

17.1.4 Identity

When running on Kubernetes, Istio Auth uses Kubernetes service accounts to identify who runs the service because of the following reasons:

- A service account is **the identity (or role) a workload runs as**, which represents that workload's privileges. For systems requiring strong security, the amount of privilege for a workload should not be identified by a random string (i.e., service name, label, etc), or by the binary that is deployed.
- Service accounts enable strong security policies by offering the flexibility to identify a machine, a user, a workload, or a group of workloads (different workloads can run as the same service account).
- The service account a workload runs as won't change during the lifetime of the workload.
- Service account uniqueness can be ensured with domain name constraint

17.2 IOTA

IOTA provides efficient, secure, lightweight, real time micro-transactions without fees. It is open-source, decentralized cryptocurrency, engineered specifically for Internet of Things, its real-time micro transactions and providing ecosystem that is ready and flexible for scale.[74]

Illustration 103: IOTA Logo

IOTA is often consider as an alternative coin (altcoin), but the truth is that IOTA is not just an altcoin, it is an extension of blockchain ecosystem. IOTA goes beyond blockchain**.** IOTA is based on Tangle instead of blockchain.[74]

17.2.1 Tangle vs. Blockchain

Tangle retains the blockchain features of the distributed ledger and secure transactions, but does not work with blocks. Instead of blocks (regular blockchain), Tangle uses the form of a Directed Acyclic Graph (DAG).[74]

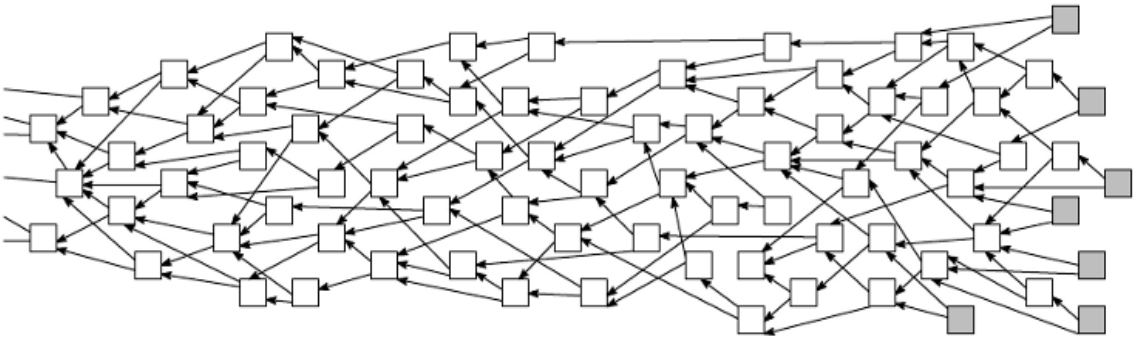


Illustration 104: Directed Acyclic Graph

17.2.2 How the Tangle works

In the Tangle, each transaction forms a new block and it is verified by itself. To be able proceed this verification it has to first verify two randomly chosen transactions in the network.

IOTA's architecture is inherently decentralized on a level that no blockchain can match, due to the fact that users and validators are one and the same in Tangle. This model has another benefit, unlimited scaling. The problem that we face right now with cryptocurrencies (based on blockchain) is scalability. The IOTA network could become indefinitely scalable with zero cost because every new transaction verifies two new transactions in the network.[74]

17.2.3 From the Internet of Things to Internet of Everything

The challenge is not in development of the smart devices or machines, but at the communication level. There will be soon more machines than humans, we need to be able to speak with machines and we need to provide a network for machines to communicate with another machines (devices).

We aren't talking just about Internet of Things (IoT), now we are talking about Internet of Everything (IoE).[74]

17.2.4 IOTA creates an environment for IoE

The interconnectivity is the key for our new economy. With interconnectivity comes incredible huge amount of data that devices send and receive every seconds. To take full benefit from the IoT and IoE we have to find a way how to connect the billions of cyber-physical systems together. The network we set up has to gain our trust.

Human kind is nowadays dependent on Internet. The consequences of depending on one single channel could be catastrophic. IOTA developed a protocol on which the machines can communicate via other channels, such as Bluetooth, Z-wave, ZigBee or LoRa. The future of connectivity might lie beyond the Internet itself.[74]

17.2.5 IoE: Sensors as an example

The number of sensors due to demand in SmartHomes and SmartCities is growing rapidly. Sending and receiving data is just the beginning of the chain. There is no doubt we will need more computational power. The big data in our new economy needs to be analyzed first and reported as soon as possible. Cloud storage nowadays cannot provide us real-time results that we require. We need smart sensors, that have computational ability itself and are able to work with other stations as well. Instead of Cloud computing we talk about Fog computing and Mist computing.

There will be billions of sensors by the 2025. Data won't be shared freely and that is a potential for cooperation among companies and individuals. IOTA will provide the platform for exchange (data for money). In the future when you gather some data, you can sell it. Selling it means to get compensation with IOTA currency. There won't be just distributed computing, data and storage, but also distributed bandwidth and energy. The future of Internet of Everything is coming.[74]

17.2.6 History of IOTA

IOTA was born as an idea to create a compensation layer that will act as the building blocks of the *Economy of Things* where machines can trade resources effectively and securely on-demand.

IOTA main network went live on 11th July 2017. The protocol and data transfer on the distributed and decentralized Tangle has been launched.[74]

The first stage: IOTA core live, establishing the IOTA Foundation, moving the community’s chat over to Slack, completing some partnerships, IOTA on exchange market, spreading into the world

The second stage: extending IOTA’s utillity, open IOTA to anyone that wants to build on top of the IOTA network

The third stage: development of hardware (project's name is *Jinn project*), a brand new type of microprocessor for IoT, a hardware implementation hasher for IOTA’s ‘Curl hash function’

18 Suggested projects

18.1 Smart campus open source project (DevOps)

Your goal will be to setup an open source project on Github. You will take on various responsibilities and carry out extensive research in the area of development and operations. This project, if sucessfully completed will provide students with the ability to develop and collaborate together on building microservices applications.

Primary objectives:

- Learn how to use Git version control

- Study at least 2 open source projects (e.g OpenWhisk and Istio)

- Setup your own repository using what you have learned from the case studies

Secondary objectives:

- Study microservices and how they can all be managed from a single repository

- Study continuous integration/delivery pipelines

- Study the Jenkins build tool (or something similar)

- Deploy your very own integration pipeline

- Integrate at least 1 testing framework (e.g Mocha.js)

- Demonstrate automated tests being executed after making a commit to the repository

- Demonstrate what happens when a test is failed vs succeeded

- Demonstrate how another student can make a commit to the repository

Responsibilities:

- You will demonstrate your administrative control over the repository.

- Provide sufficient documentation and instructions so that future students can easily start working on their own smart campus microservices without conflicting with others work.

18.2 Smart campus mobile applications using Ionic library

Ionic is a powerful HTML5 SDK that helps you build native-feeling mobile apps using web technologies like HTML, CSS, and Javascript.[75] Ionic currently requires AngularJS in order to work at its full potential. Ideally your application should use mqtt to interact with a node.js backend or RESTful microservices.

Primary objectives:

- Create a prototype app that has a detailed view of the UL campus and class rooms

- The app communicates with microservices to get current statistics (e.g number of people)

- Sections of the map are updated independently by individual microservices (e.g Schuman Service)

Secondary objectives:

- Prototype for Library (e.g feature for notifying user about available desks)

- Prototype for Gym (e.g feature for notifying user about available threadmills)

- Prototype for Car park (e.g feature for notifying user about available spaces)

Link to Ionic/Mosca integration:

https://blog.codecentric.de/en/2014/09/home-pi-reloaded-home-automation-ionic-mqtt/

18.3 Smart campus public/sensor API

Using Kubernetes and Istio, develop an API that will handle the data flow coming from sensors.

Primary Objectives:

- Sensors should be able to call the sensor API to transmit and store data.

- Sensors should be able to call public API to get data readings

Secondary Objectives:

- Demonstration of service version controlling (v1, v2, v3)

- Demonstration of load balancing modes (round robin, random, and weighted least request)

- Demonstration of service policy and control (e.g only bob can use serviceBob)

18.4 Graph database design for both a WSN and/or student survey application

Using mocked WSN/survey data, your objective is to design a suitable graph database for WSN and/or student surveys.

Objectives:

- Study graph databases

- Study Gremlin (a graph traversal language)

- Draw up a graph schema for survey info (e.g linking gender with department nodes)

- feed data to Watson

- train Watson

18.5 Migration of a monolithic app to a microservices based architecture

Your objective is to take a reasonably large monolithic application and break it down into microservices using Istio and Kubernetes.

18.6 RMI

Dan Geer raises the argument that embedded systems without a remote management interface "and thus out of reach, are a life form," and "as the purpose of life is to end, an embedded system without a remote management interface must be so designed to be certain to die no later than some fixed time. Conversely, an embedded system with a remote management interface must be sufficiently self-protecting that it is capable of refusing a command," said Geer, speaking at The Security of Things Forum.[76]

Your goal is to design a remote management interface for a Raspberry Pi running Docker.

Primary goals:

- Research remote management interfaces such as Resin.io [77]

- Trigger a Jenkins/Ansible job on the Raspberry Pi from a webhook

- Learn Docker

- Run a Mosca/Node-RED Container

- Research edge computing

18.7 Iota library

The IOTA Sandbox is a service provided by the Foundation that makes it possible to start using IOTA without having to go through the hassle of installing, configuring and setting up your local client. Thanks to the redundant node setup and a dedicated GPU farm, you don't even have to worry about doing Proof of Work, which means that you can start testing and developing new applications, even with tiny IoT devices, all by simply making API calls to the Sandbox environment.

The easiest way to use the Sandbox environment is to simply use one of the official libraries in your preferred language. Currently IOTA has Sandbox support for Javascript, but the Python, Java and C# libraries are under way.

Primary goals:

- Learn the library for your chosen language (currently Python or Javascript)

- Brainstorm use cases for IOTA

- Develop a project plan around your idea

- Integrate your idea with an embedded device

Other ideas:

- Investigate the use of IOTA on embedded devices, in particular the ASIC chips required for doing computation

- Investigate how multiple embedded devices can perform transactions using multiple communication technologies such as Bluetooth, Z-wave, ZigBee or LoRa

- Investigate the lower limits of IOTA by testing minimum bandwidth and minimum computational power required, discuss your findings

- Create a decentralized smart campus app, the application should require minimum backend support

- Investigate how IOTA might be integrated with an existing Blockchain application

18.8 IBM Hyperledger

Investigate Blockchain use cases for supply chain, capital markets, manufacturing and healthcare. Create a decentralized smart campus application using Hyperledger.[78] For getting started you can set up and run a blockchain network with IBM-supplied Docker Compose script and images.

19 Conclusion

This theses touches on a lot of topics such as DevOps, Cloud ecosystem, Containers, Open source, Internet of things and even cryptocurrency. The learning outcome from taking on such a project has been significant. There are now many exciting projects that can be undertaken by students with the discovery of these new technologies. Now that most of the infrastructure required for analytics is in place, it will be exciting to see what projects will be created from this theses.

20 References

[1]"IBM", *Docker*, 2017. [Online]. Available: https://www.docker.com/ibm. [Accessed: 30- Aug- 2017].

[2]"Podcasts - The New Stack", *The New Stack*, 2017. [Online]. Available: https://thenewstack.io/podcasts/. [Accessed: 30- Aug- 2017].

[3]T. WIRED, "The Internet of Things Is Far Bigger Than Anyone Realizes", *WIRED*, 2017. [Online]. Available: https://www.wired.com/insights/2014/11/the-internet-of-things-bigger/. [Accessed: 30- Aug- 2017].

[4]"What is cognitive IoT?", *IBM Big Data & Analytics Hub*, 2017. [Online]. Available: http://www.ibmbigdatahub.com/blog/what-cognitive-iot. [Accessed: 30- Aug- 2017].

[5]S. User, "IOT Info", *Iot.ul.ie*, 2017. [Online]. Available: http://www.iot.ul.ie/index.php/summer-school-details. [Accessed: 30- Aug- 2017].

[6]B. Kepes, "The changing face of open-source software", *Computerworld*, 2017. [Online]. Available: https://www.computerworld.com/article/2968315/open-source-tools/the-changing-face-of-open-source-software.html. [Accessed: 30- Aug- 2017].

[7]2017. [Online]. Available: http://datasys.cs.iit.edu/events/DataCloud2014/DataCloud2014-paper-4-fullpaper-Gerlach.pdf. [Accessed: 30- Aug- 2017].

[8]"About Cloud Foundry | Open Source Cloud Application Platform", *Cloud Foundry*, 2017. [Online]. Available: https://www.cloudfoundry.org/foundation/. [Accessed: 30- Aug- 2017].

[9]"What is IBM Watson?", Ibm.com, 2016. [Online]. Available: http://www.ibm.com/watson/what-is-watson.html. [Accessed: 03- Nov- 2016].

[10]"Node-RED", Nodered.org, 2016. [Online]. Available: https://nodered.org/. [Accessed: 03- Nov- 2016].

[11]"What Is Serverless Computing, and Why Should I Care?", Developer.ibm.com, 2017. [Online]. Available: https://developer.ibm.com/openwhisk/what-is-serverless-computing/. [Accessed: 30- Aug- 2017].

[12]J. McGee, J. MSV and M. Boyd, "Three Startups Using IBM OpenWhisk Serverless to Transform Their Industries - The New Stack", The New Stack, 2017. [Online]. Available: https://thenewstack.io/future-serverless-3-startups-using-serverless-cognitive-iot-transform-industries/. [Accessed: 30- Aug- 2017].

[13]I. Analytics, "IBM Cloudant – Managed Database - Watson Data Platform", Ibm.com, 2017. [Online]. Available: https://www.ibm.com/analytics/us/en/technology/cloud-data-services/cloudant/. [Accessed: 30- Aug- 2017].

[14]D. Cassel and M. Gienow, "The Power of Community in Open Source - The New Stack", The New Stack, 2017. [Online]. Available: https://thenewstack.io/power-community-open-source/. [Accessed: 30- Aug- 2017].

[15]C. computing, "IBM's approach to open technology", Ibm.com, 2017. [Online]. Available: https://www.ibm.com/developerworks/cloud/library/cl-open-architecture-update/. [Accessed: 30- Aug- 2017].

[16]"StrongLoop - Open-source solutions for the API developer community", Strongloop.com, 2017. [Online]. Available: https://strongloop.com/. [Accessed: 30- Aug- 2017].

[17]"About", Open API Initiative, 2017. [Online]. Available: https://www.openapis.org/about. [Accessed: 30- Aug- 2017].

[18]"About | Open Container Initiative", Opencontainers.org, 2016. [Online]. Available: https://www.opencontainers.org/about. [Accessed: 03- Nov- 2016].

[19]"runC: The little container engine that could", Opensource.com, 2017. [Online]. Available: https://opensource.com/life/16/8/runc-little-container-engine-could. [Accessed: 30- Aug- 2017].

[20]C. Childers, "Garden and runC - Cloud Foundry", Cloud Foundry, 2017. [Online]. Available: https://www.cloudfoundry.org/garden-and-runc/. [Accessed: 30- Aug- 2017].

[21]"About - Cloud Native Computing Foundation", Cloud Native Computing Foundation, 2016. [Online]. Available: https://www.cncf.io/about. [Accessed: 03- Nov- 2016]

[22]"IBM Launches Bluemix Container Service with Kubernetes", Www-03.ibm.com, 2017. [Online]. Available: https://www-03.ibm.com/press/us/en/pressrelease/51843.wss. [Accessed: 30- Aug- 2017].

[23]C. Rosen, "Kubernetes now available on IBM Bluemix Container Service - Bluemix Blog", Bluemix Blog, 2017. [Online]. Available: https://www.ibm.com/blogs/bluemix/2017/03/kubernetes-now-available-ibm-bluemix-container-service/. [Accessed: 30- Aug- 2017].

[24]"Kubernetes From IBM Bluemix Container Service", Ibm.com, 2017. [Online]. Available: https://www.ibm.com/cloud-computing/bluemix/containers. [Accessed: 30- Aug- 2017].

[25]J. MSV, P. Waterhouse and J. MSV, "Kubernetes: An Overview - The New Stack", The New Stack, 2017. [Online]. Available: https://thenewstack.io/kubernetes-an-overview/. [Accessed: 30- Aug- 2017].

[26]B. Ball and A. Williams, "Bridging Realities: Orchestration and Programmable Infrastructure - The New Stack", The New Stack, 2017. [Online]. Available: https://thenewstack.io/bridging-realities-orchestration-programmable-infrastructure/. [Accessed: 30- Aug- 2017].

[27]"practice continuous integration", Ibm.com, 2017. [Online]. Available: https://www.ibm.com/devops/method/content/code/practice_continuous_integration/. [Accessed: 30- Aug- 2017].

[28]H. Home, "Top Benefits of Continuous Integration - Hiring | Upwork", Hiring | Upwork, 2017. [Online]. Available: https://www.upwork.com/hiring/for-clients/top-benefits-of-continuous-integration/. [Accessed: 30- Aug- 2017].

[29]"Infrastructure automation", Ibm.com, 2017. [Online]. Available: https://www.ibm.com/developerworks/library/a-devops2/index.html. [Accessed: 30- Aug- 2017].

[30]A. Hat, "How Ansible Works | Ansible.com", Ansible.com, 2017. [Online]. Available: https://www.ansible.com/how-ansible-works. [Accessed: 30- Aug- 2017].

[31]"About Jenkins", CloudBees, 2017. [Online]. Available: https://www.cloudbees.com/jenkins/about. [Accessed: 30- Aug- 2017].

[32]"Jenkins Tutorial | Continuous Integration Using Jenkins | Edureka", Edureka Blog, 2017. [Online]. Available: https://www.edureka.co/blog/jenkins-tutorial/. [Accessed: 30- Aug- 2017].

[33]A. AS, "Visualizing Docker Compose with Ardoq - Ardoq Blog", Ardoq, 2017. [Online]. Available:

https://ardoq.com/visualizing-docker-compose/. [Accessed: 23- Feb- 2017].

[34]E. Norman and E. Norman, "Amalgam8: Framework for composition and orchestration of microservices - Bluemix Blog", Bluemix Blog, 2017. [Online]. Available: https://www.ibm.com/blogs/bluemix/2016/07/amalgam8-framework-for-microservices-orchestration/?S_TACT=M16103KW. [Accessed: 23- Feb- 2017].

[35]"IBM Knowledge Center", Ibm.com, 2017. [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_7.5.0/com.ibm.mm.tc.doc/tc00000_.htm. [Accessed: 30- Aug- 2017].

[36]"MQTT Essentials Part 3: Client, Broker and Connection Establishment", HiveMQ, 2017. [Online]. Available: http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment. [Accessed: 30- Aug- 2017].

[37]M. Collina, "An Internet of Things System - How To Build It Faster", nearForm, 2017. [Online].

Available: http://www.nearform.com/nodecrunch/internet-of-things-how-to-build-it-faster/. [Accessed:23- Feb- 2017].

[38]"Learn REST: A Tutorial", Rest.elkstein.org, 2016. [Online]. Available: http://rest.elkstein.org/. [Accessed: 03- Nov- 2016].

[39]"Microservices from Theory to Practice: Creating Applications in IBM Bluemix Using the

Microservices Approach", Google Books, 2017. [Online]. Available: https://books.google.ie/books?

id=eOZyCgAAQBAJ&pg=PA95&lpg=PA95&dq=ibm+mqtt+mirroring&source=bl&ots=sggWe9DQfH&sig=5j7UvNldc8Hgon7L8mDC45_09lc&hl=en&sa=X&ved=0ahUKEwj5q6G1r7HSAhVoCcAKHXcwAtIQ6AEIIjAB#v=onepage&q=ibm%20mqtt%20mirroring&f=false. [Accessed: 27- Feb- 2017].

[40]"REST and MQTT: Yin and Yang of Micro-Service APIs", Dejan Glozic, 2017. [Online]. Available:https://dejanglozic.com/2014/05/06/rest-and-mqtt-yin-and-yang-of-micro-service-apis/. [Accessed: 27- Feb-2017].

[41]M. Koster, M. Koster and V. profile, "Event Models for RESTful APIs", Iot-datamodels.blogspot.ie, 2017. [Online]. Available: http://iot-datamodels.blogspot.ie/2013/05/event-models-for-restful-apis.html.[Accessed: 28- Feb- 2017].

[42]"Server-side service discovery pattern", Microservices.io, 2017. [Online]. Available:

http://microservices.io/patterns/server-side-discovery.html. [Accessed: 23- Feb- 2017].

[43]"The Scale Cube", Microservices.io, 2017. [Online]. Available: http://microservices.io/articles/scalecube.html. [Accessed: 30- Aug- 2017].

[44]"Load Balancing 101: Nuts and Bolts", F5.com, 2017. [Online]. Available: https://f5.com/resources/white-papers/load-balancing-101-nuts-and-bolts. [Accessed: 30- Aug- 2017].

[45]"What is Extract Transform Load: Definition | Informatica US", Informatica.com, 2017. [Online]. Available: https://www.informatica.com/services-and-training/glossary-of-terms/extract-transform-load-definition.html. [Accessed: 30- Aug- 2017].

[46]"TRANSIT: Flexible pipeline for IoT data with Bluemix and OpenWhisk – OpenWhisk", Medium, 2017. [Online]. Available: https://medium.com/openwhisk/transit-flexible-pipeline-for-iot-data-with-bluemix-and-openwhisk-4824cf20f1e0#.kv9gk4j2g. [Accessed: 23- Feb- 2017].

[47]"How to Build an High Availability MQTT Cluster for the Internet of Things", Medium, 2017.

[Online]. Available: https://medium.com/@lelylan/how-to-build-an-high-availability-mqtt-cluster-for-the-internet-of-things-8011a06bd000#.iku8iazv6. [Accessed: 23- Feb- 2017].

[48]T. Flynn, "16117743/INS-Thesis-Documentation", GitHub, 2016. [Online]. Available: https://github.com/16117743/INS-Thesis-Documentation/blob/master/Autumn logs/week-5.pdf. [Accessed: 04- Nov- 2016].

[49]T. Flynn, "16117743/INS-Thesis-Documentation", GitHub, 2016. [Online]. Available: https://github.com/16117743/INS-Thesis-Documentation/blob/master/Autumn logs/week-6.pdf. [Accessed: 04- Nov- 2016].

[50]T. Flynn, "16117743/INS-Thesis-Documentation", GitHub, 2016. [Online]. Available: https://github.com/16117743/INS-Thesis-Documentation/blob/master/Autumn logs/week-7.pdf. [Accessed: 04- Nov- 2016].

[51]T. Flynn, "16117743/INS-Thesis-Documentation", GitHub, 2016. [Online]. Available: https://github.com/16117743/INS-Thesis-Documentation/blob/master/Autumn logs/week-8.pdf. [Accessed: 04- Nov- 2016].

[52]T. Flynn, "Spring Week 1 Log", 2017. [Online]. Available: https://github.com/16117743/INS-Thesis-Documentation/blob/master/Spring%20logs/Spring_Wk1.pdf. [Accessed: 23- Feb- 2017].

[53]T. Flynn, "Spring Week 2 Log", 2017. [Online]. Available: https://github.com/16117743/INS-Thesis-Documentation/blob/master/Spring%20logs/Spring_Wk 2 .pdf. [Accessed: 23- Feb- 2017].

[54]T. Flynn, "Spring Week 3 Log", 2017. [Online]. Available: https://github.com/16117743/INS-Thesis-Documentation/blob/master/Spring%20logs/Spring_Wk 3 .pdf. [Accessed: 23- Feb- 2017].

[55]T. Flynn, "Spring Week 4 Log", 2017. [Online]. Available: https://github.com/16117743/INS-Thesis-Documentation/blob/master/Spring%20logs/Spring_Wk 4 .pdf. [Accessed: 23- Feb- 2017].

[56]"Cloudant vs. Elasticsearch vs. InfluxDB vs. Redis Comparison", Db-engines.com, 2017. [Online].
Available: http://db-engines.com/en/system/Cloudant%3BElasticsearch%3BInfluxDB%3BRedis. [Accessed: 27- Feb- 2017].

[57]"HAProxy - The Reliable, High Performance TCP/HTTP Load Balancer", Haproxy.org, 2017. [Online]. Available: http://www.haproxy.org/. [Accessed: 23- Feb- 2017].

[58]J. Wu, "A Network Function Virtualization based Load Balancer for TCP", Master of Science, ARIZONA STATE UNIVERSITY, 2017.

[59]M. Collina, "GitHub - mcollina/mosca: MQTT broker as a module", Github.com, 2017. [Online].Available: https://github.com/mcollina/mosca. [Accessed: 23- Feb- 2017].

[60]"Redis", Redis.io, 2017. [Online]. Available: https://redis.io/. [Accessed: 23- Feb- 2017].

[61]"Kafka vs. Redis: Log Aggregation Capabilities and Performance", Logz.io, 2017. [Online]. Available: http://logz.io/blog/kafka-vs-redis/. [Accessed: 23- Feb- 2017].

[62]"GitHub - influxdata/influxdb: Scalable datastore for metrics, events, and real-time analytics",Github.com, 2017. [Online]. Available: https://github.com/influxdata/influxdb. [Accessed: 25- Feb- 2017].

[63]"InfluxDB", En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/InfluxDB.[Accessed: 25- Feb- 2017].

[64]"DevOps Automation Cookbook", Google Books, 2017. [Online]. Available:

https://books.google.ie/booksid=k_SoCwAAQBAJ&pg=PA176&redir_esc=y#v=onepage&q&f=false.[Accessed: 25- Feb- 2017].

[65]"InfluxData Documentation", Docs.influxdata.com, 2017. [Online]. Available:

https://docs.influxdata.com/influxdb/v0.8/api/continuous_queries/. [Accessed: 25- Feb- 2017].

[66]"Jenkins 2 Overview", Jenkins 2 Overview, 2017. [Online]. Available: https://jenkins.io/2.0/. [Accessed: 30- Aug- 2017].

[67]"IBM/ibm-cloud-devops", GitHub, 2017. [Online]. Available: https://github.com/IBM/ibm-cloud-devops. [Accessed: 30- Aug- 2017].

[68]"Pipeline", Pipeline, 2017. [Online]. Available: https://jenkins.io/doc/book/pipeline/. [Accessed: 30- Aug- 2017].

[69]"Bluemix Container monitoring and logging", Console.ng.bluemix.net, 2017. [Online]. Available:https://console.ng.bluemix.net/docs/containers/monitoringandlogging/container_ml_ov.html.[Accessed: 23- Feb- 2017].

[70]"data-flow – Node RED Programming Guide", Noderedguide.com, 2017. [Online]. Available: http://noderedguide.com/tag/data-flow/. [Accessed: 30- Aug- 2017].

[71]"OpenWhisk", OpenWhisk, 2016. [Online]. Available: https://developer.ibm.com/openwhisk/. [Accessed: 03- Nov- 2016].

[72]"IBM, Google & Lyft Give New Istio Microservices Mesh a Ride", The developerWorks Blog, 2017. [Online]. Available: https://developer.ibm.com/dwblog/2017/istio/. [Accessed: 30- Aug- 2017].

[73]"Istio / Auth", Istio.io, 2017. [Online]. Available: https://istio.io/docs/concepts/network-and-auth/auth.html. [Accessed: 30- Aug- 2017].

[74]"IOTA makes bright future for Internet of Things, it's not just a cryptocurrency based on blockchain", Medium, 2017. [Online]. Available: https://medium.com/@MartinRosulek/how-iota-makes-future-for-internet-of-things-af14fd77d2a3. [Accessed: 30- Aug- 2017].

[75]"Build Amazing Native Apps and Progressive Web Apps with Ionic Framework and Angular", Ionic Framework, 2017. [Online]. Available: https://ionicframework.com/. [Accessed: 30- Aug- 2017].

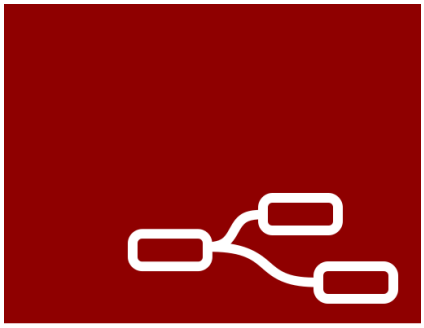
[76]P. Thibodeau, "Embedded systems are a 'life form'", Computerworld, 2017. [Online]. Available: https://www.computerworld.com/article/2489154/internet/embedded-systems-are-a--life-form-.html. [Accessed: 30- Aug- 2017].

[77]"Home | Resin.io", Resin.io, 2017. [Online]. Available: https://resin.io/. [Accessed: 30- Aug- 2017].

[78]"IBM Blockchain based on Hyperledger Fabric from the Linux Foundation", 2017. [Online]. Available: https://www.ibm.com/blockchain/hyperledger.html. [Accessed: 30- Aug- 2017].

Appendix 1: Node-RED lab

IoT Summer School



Lab

Lab objectives

In part 1 you will sign up to Bluemix and setup a blank Node-RED flow.

In part 2 you will send messages to members of your group using mqtt nodes and topics.

In part 3 you will create a new flow and explore Cloudant and Watson Analytics nodes.

Learning outcomes

On successful completion of this lab you will have learned about

- MQTT pub/sub protocol and QoS

- Controlling data through MQTT topics using Node-RED flows

- How messages flow through nodes

- Debug and mqtt nodes

- Cloudant database nodes

- Watson Analytics nodes

Topic structure

→ ***iot-summer-school***

→ **group-1-iotp**

→ **Ann**

→ **Bob**

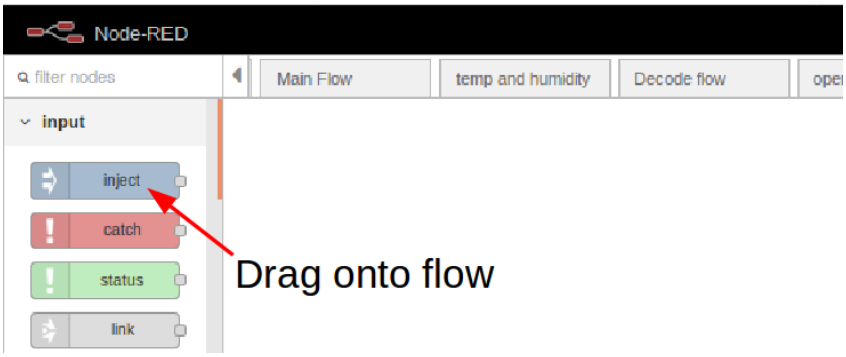
→ **Joe**

→ **group-2-iotp**

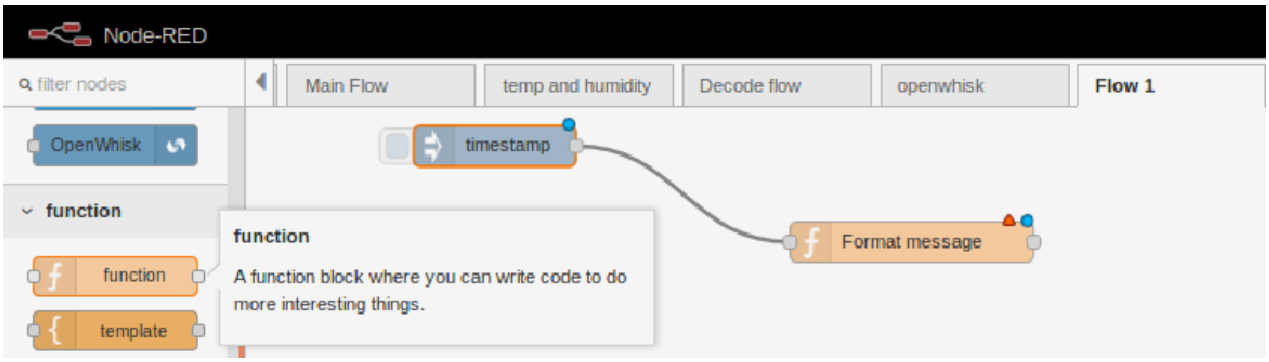
→ **Jim**

→ **…**

Step 1: Add inject node to Flow 1

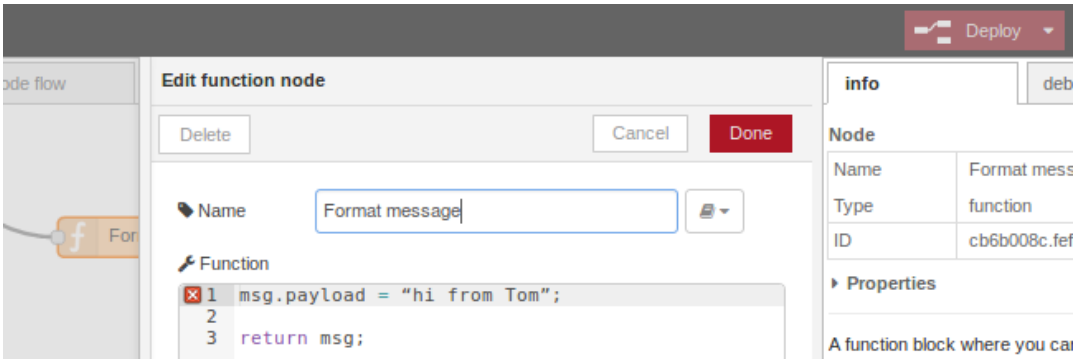


Step 2: Add function node to Flow 1



Connect the output of the inject node to the input of the function node as shown in the above figure.

Messages can be modified within the node by double clicking on the function node.

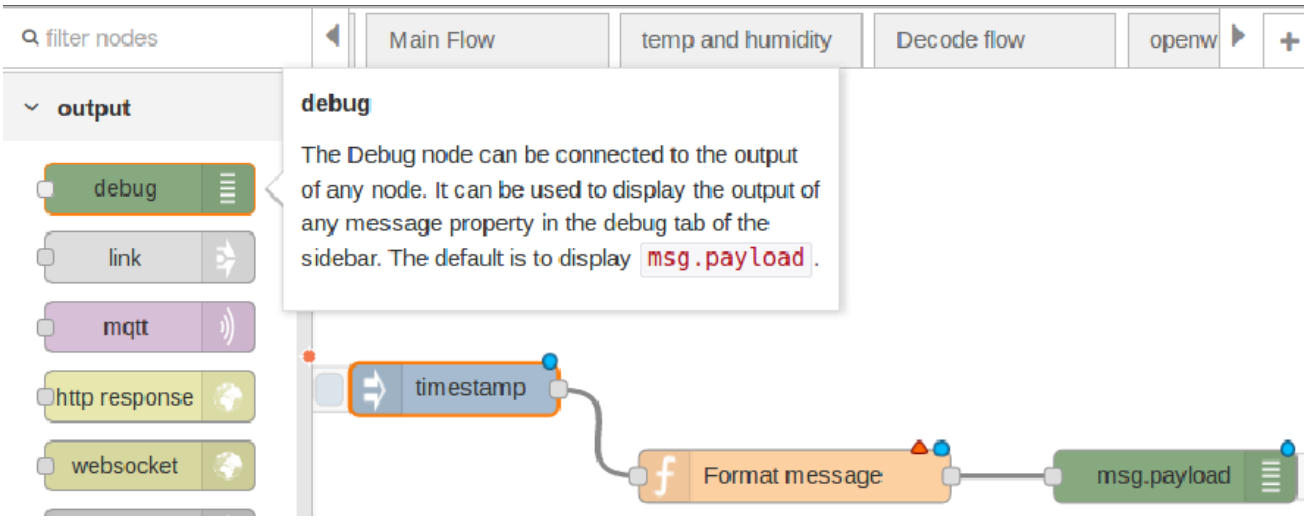


Step 3: Add line of code

Add the following line but do not copy and paste.

msg.payload = "hi from <insert your name>";

Step 4: Add debug node to Flow 1

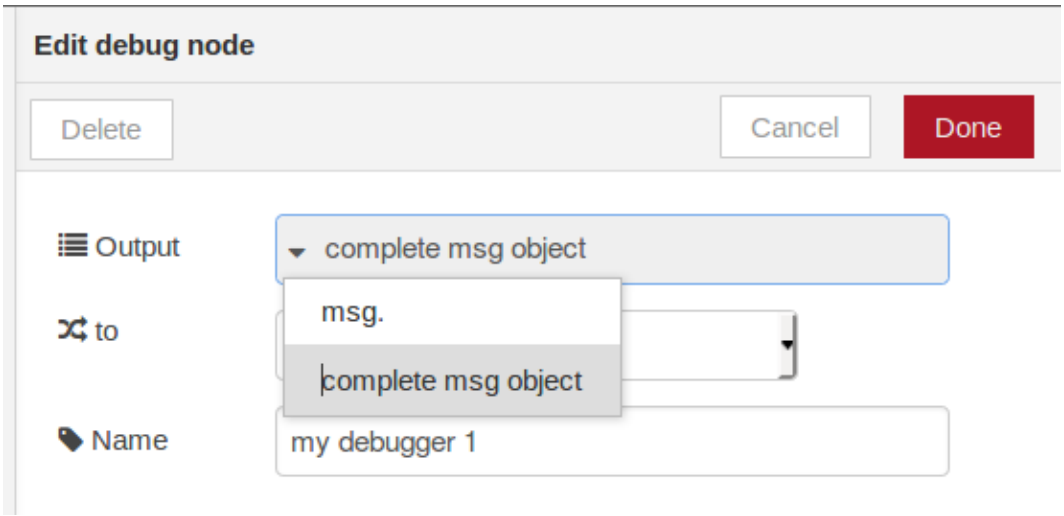


Step 5: Edit debug node

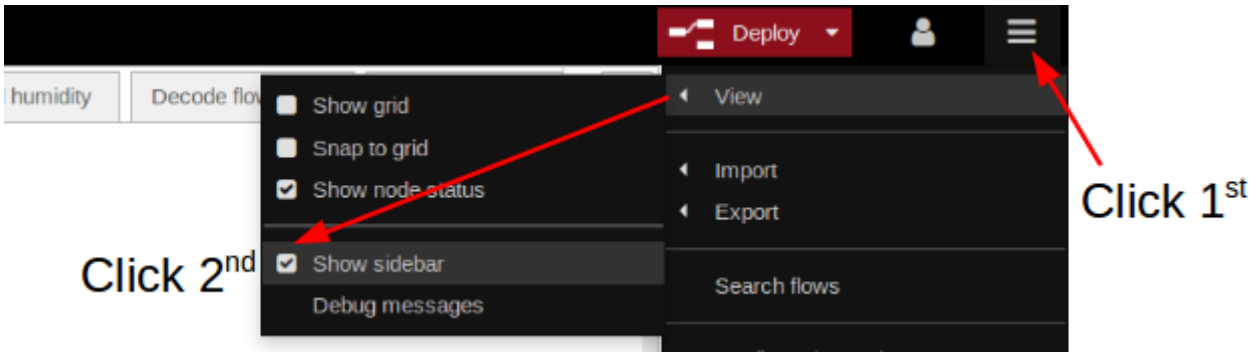
Change the output option to “complete msg object”.

Why do we need to see the complete message?

To help us understand the structure and attributes of messages as they pass through nodes.

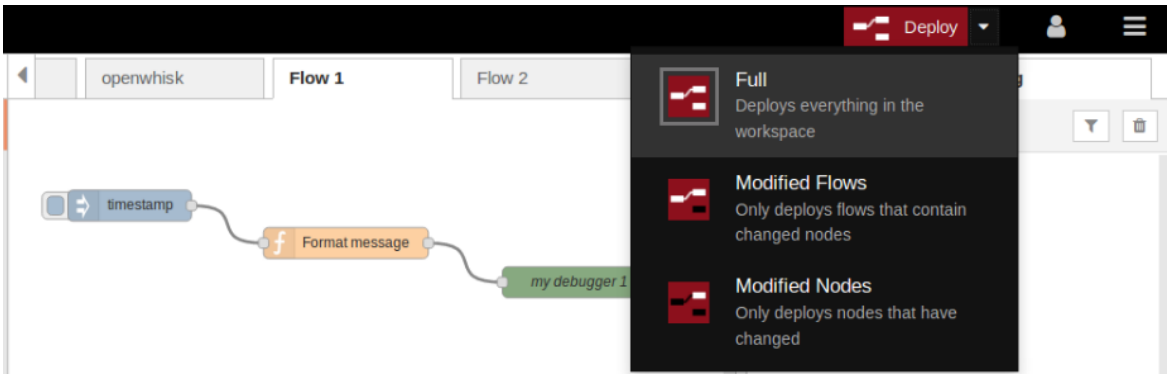


I cannot see my debug tab?

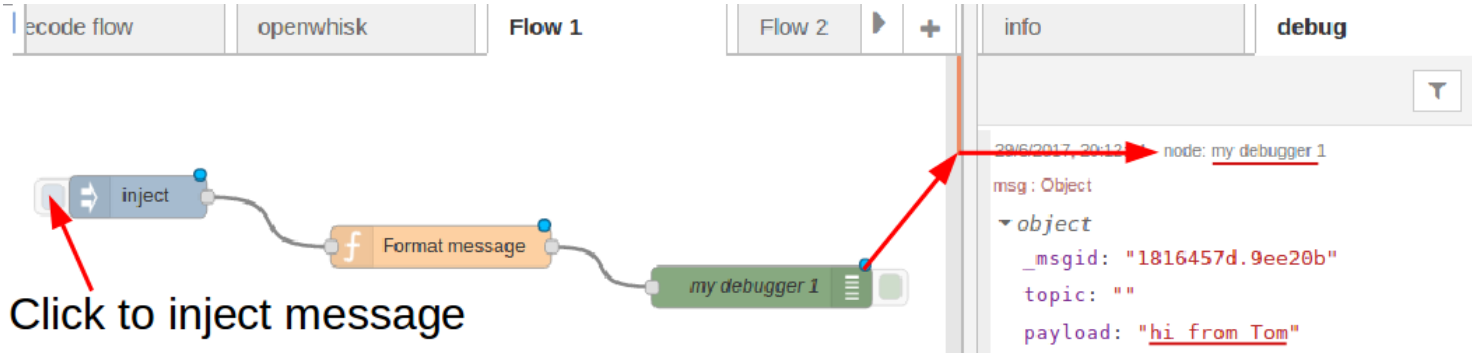


Step 6: Deploy Flow 1

After clicking on the drop-down arrow, click “Full”. Then click on the deploy button to deploy the flow.



Step 7: Test debug node



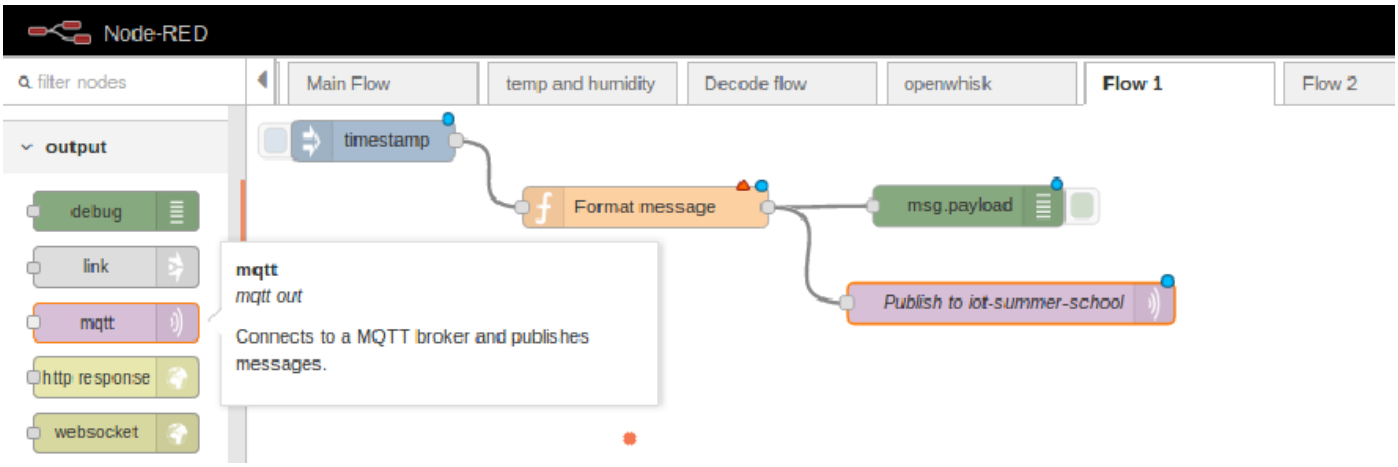
In the debug tab on the right, notice how each new message has a uniquely generated **msg._msgid**

Notice that **msg.topic** is blank.

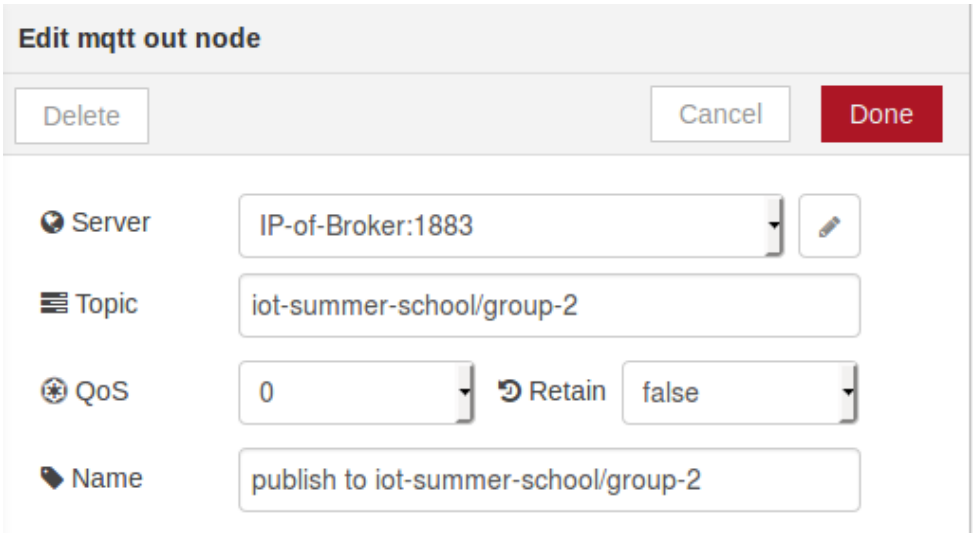
Notice the value of **msg.payload** is the value applied in the function node (“hi from <your name>”)

Step 8: Publish to the *“iot-summer-school/group-<insert group number>”* topic

Add the mqtt out node and wire it to the output of the function node



Double click on the mqtt output node and make the following changes.



What is the mqtt out node going to do with the message?

The node will publish to the broker on "iot-summer-school/group-2" topic.

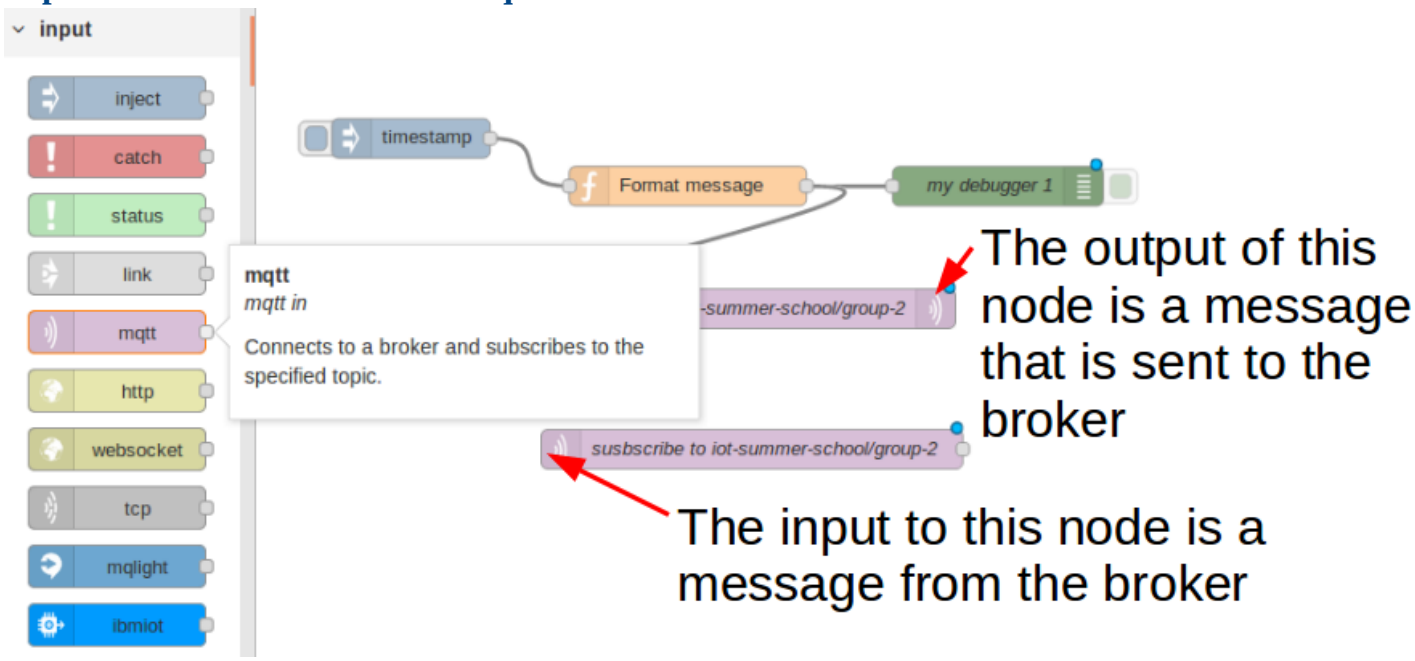
Any mqtt in node that is subscribed to that topic will receive a message with a payload containing

"hi from <insert your name>"

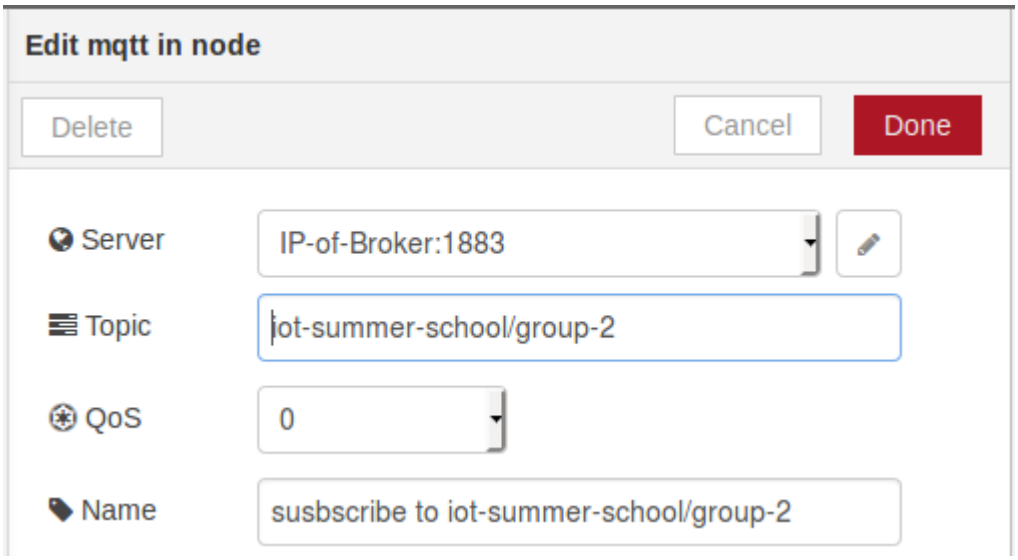
Will my flow also receive the message it just published?

No, we have only used an mqtt out node. This node only publishes messages and does not subscribe.

Step 9: Add subscribe node or "mqtt in" node



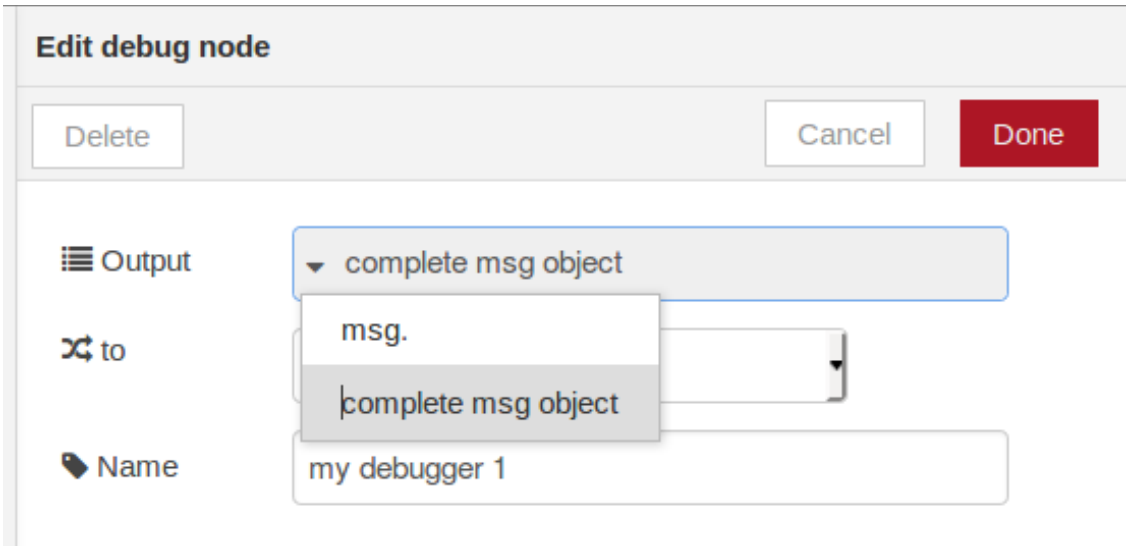
Step 10: Edit the mqtt in node



Step 11: Add another debug node

Make the same change as last time, we do this because we want to see all message attributes.

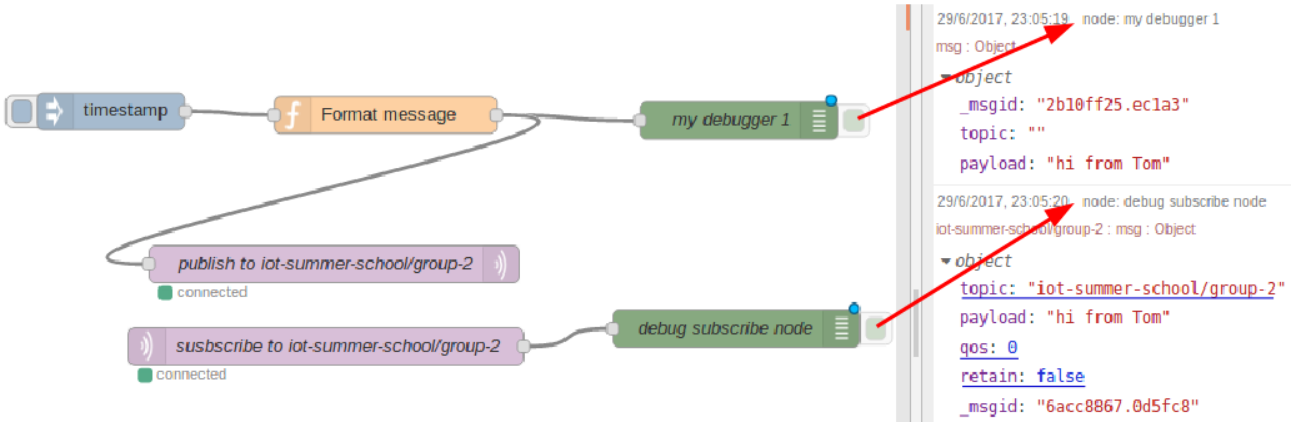
Then wire the output of the mqqt in node to the input of the new debug node.



Step 12: Deploy the application

Remember to hit the deploy button after making a change to a flow.

Step 13: Check the debug tab



Notice the message captured in the “debug subscribe node” has the following attributes

topic: “iot-summer-school/group”

qos: 0

retain: false

And most importantly the payload contains “hi from <your name>”

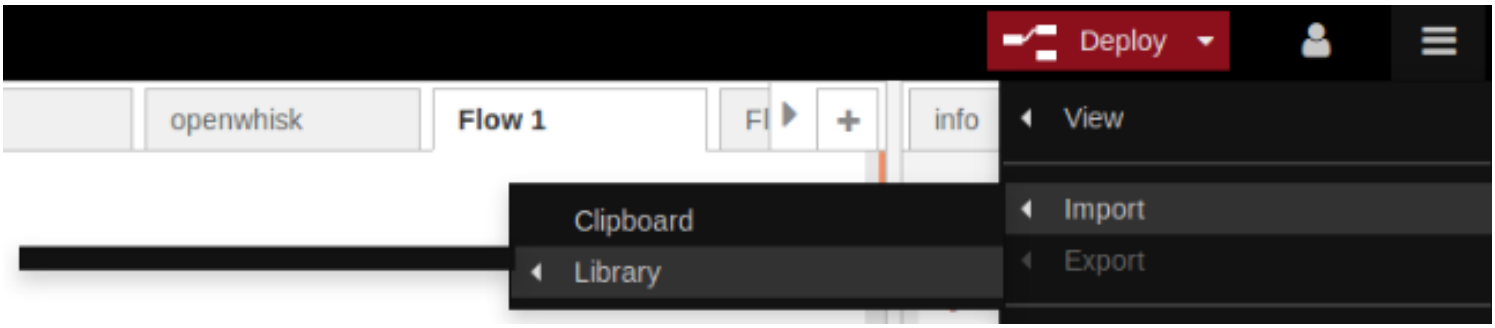
Step 14: Help your group members

After you get the subscribe node working, go help group members that are having problems.

When everyone in the group can send and receive messages, continue to the next stage and create a new flow called “Flow 2”.

If you are struggling to keep up or having problems with your flow, do not worry.

Everyone will be importing the part 1 solution provided below to use as the starting point for Flow 2.



Flow 2 Starting point

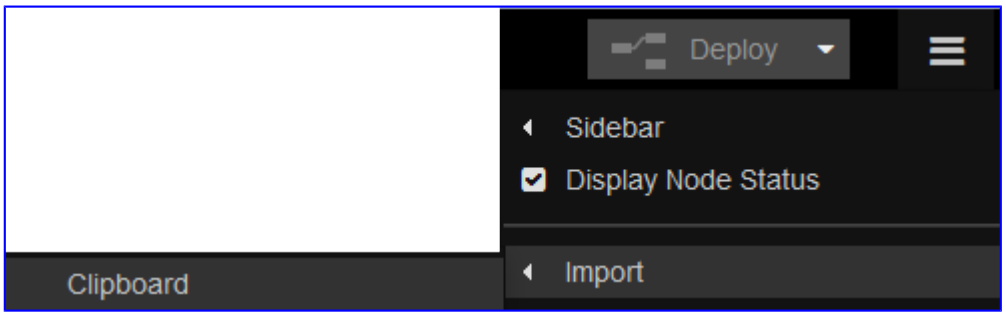
[{"id":"31a61f50.5b8548","type":"tab","label":"Flow 3"},
{"id":"1186df17.507979","type":"inject","z":"31a61f50.5b8548","name":"","topic":"","payload":"","payloadType":"date","repeat":"","crontab":"","once":false,"x":105.27777099609375,"y":136.30001831054688,"wires":[["cb6b008c.fef0e"]]},
{"id":"cb6b008c.fef0e","type":"function","z":"31a61f50.5b8548","name":"Format message","func":"msg.payload = \"hi from John Doe\";\n\nreturn msg;","outputs":1,"noerr":0,"x":327.4721984863281,"y":139.17776489257812,"wires":[["8379426f.cbb51","b99ddb50.e3c97"]]},
{"id":"8379426f.cbb51","type":"debug","z":"31a61f50.5b8548","name":"my debugger 1","active":true,"console":"false","complete":"true","x":613.7777099609375,"y":143.73880004882812,"wires":[]},{"id":"b99ddb50.e3c97","type":"mqtt out","z":"31a61f50.5b8548","name":"publish to iot-summer-school/group-2","topic":"iot-summer-school/group-2","qos":"0","retain":"false","broker":"de850a9d.6952c8","x":286.111083984375,"y":260.0000305175781,"wires":[]},{"id":"fa49e949.94837","type":"mqtt in","z":"31a61f50.5b8548","name":"susbscribe to iot-summer-school/group-2","topic":"iot-summer-school/group-2","qos":"0","broker":"de850a9d.6952c8","x":279.4444580078125,"y":326.6666259765625,"wires":[["97725758.16c928"]]},
{"id":"97725758.16c928","type":"debug","z":"31a61f50.5b8548","name":"debug subscribe node","active":true,"console":"false","complete":"true","x":611.6666870117188,"y":312.2216796875,"wires":[]},{"id":"de850a9d.6952c8","type":"mqtt-broker","z":"","broker":"134.168.46.151","port":"1883","clientid":"","usetls":false,"compatmode":true,"keepalive":"60","cleansession":true,"willTopic":"","willQos":"0","willPayload":"","birthTopic":"","birthQos":"0","birthPayload":""}]

8 Lab part 3: Watson translate service

Step 1: Import Flow 2 starting point

Select the flow above “Flow 2 starting point” and copy it to the clipboard (Ctrl-C). Import the flow into Node-RED using by selecting the node-RED menu

and select the import from clipboard option.



You will be presented with a form in which you create nodes by entering json data.

Import your copied flow by pasting (Ctrl-P) from the clipboard into the the form

Step 2: Edit function node

Change from

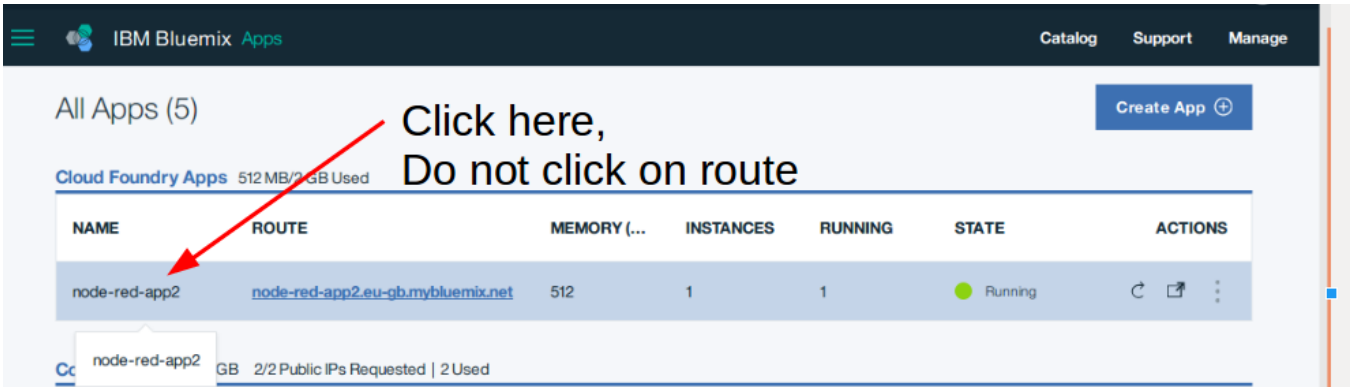
msg.payload = “hi from <your name>”

to

msg.payload = "你好，世界";

You are now going to use the Watson language translate service to convert msg.payload from Chinese into english.

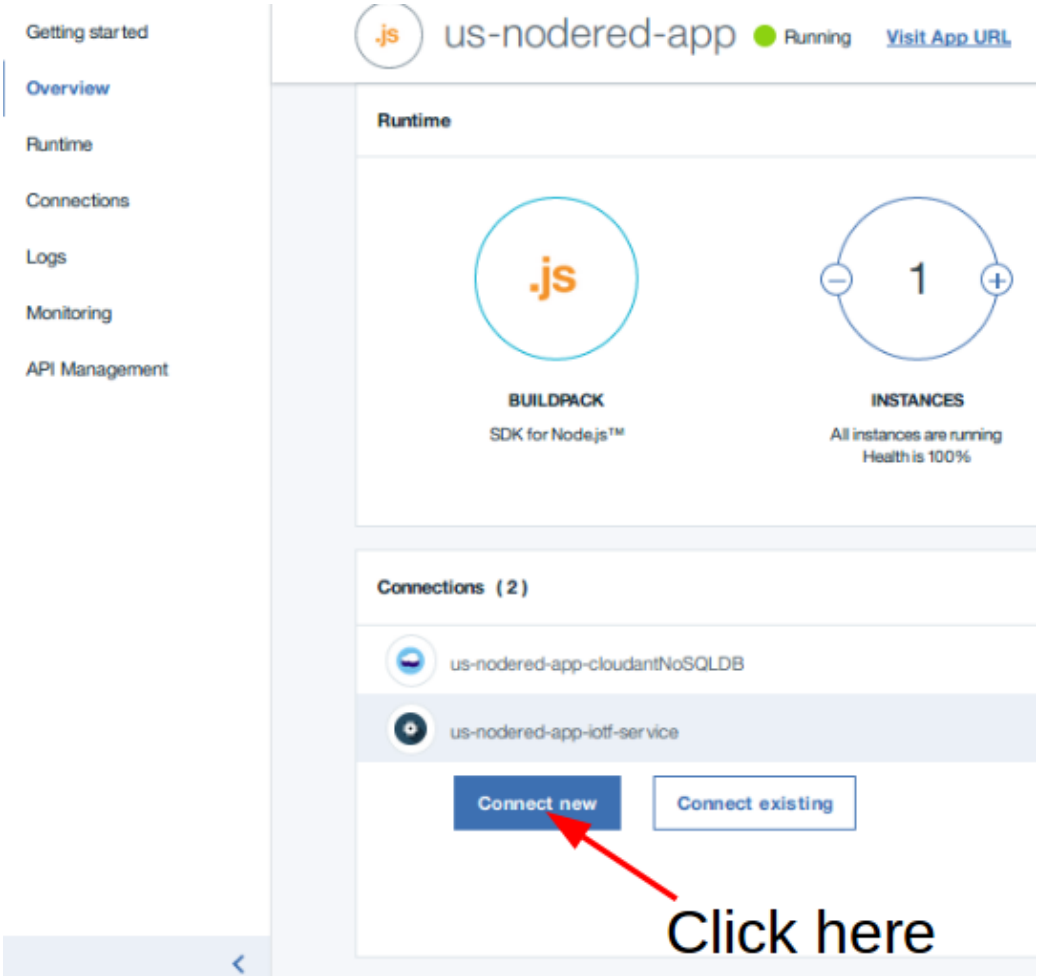
Step 3: Go to Cloud Foundry App control panel



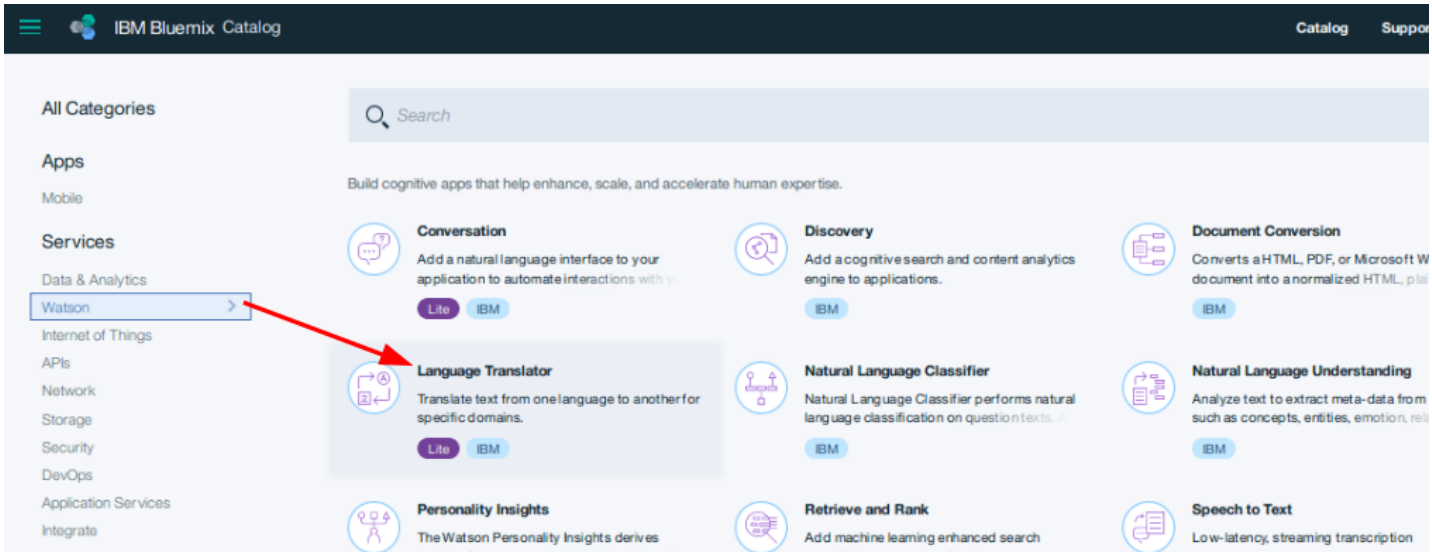
By clicking on route, you will be taken to another Node-RED flow editor screen.

However, we want to go to the Cloud Foundry App control panel instead.

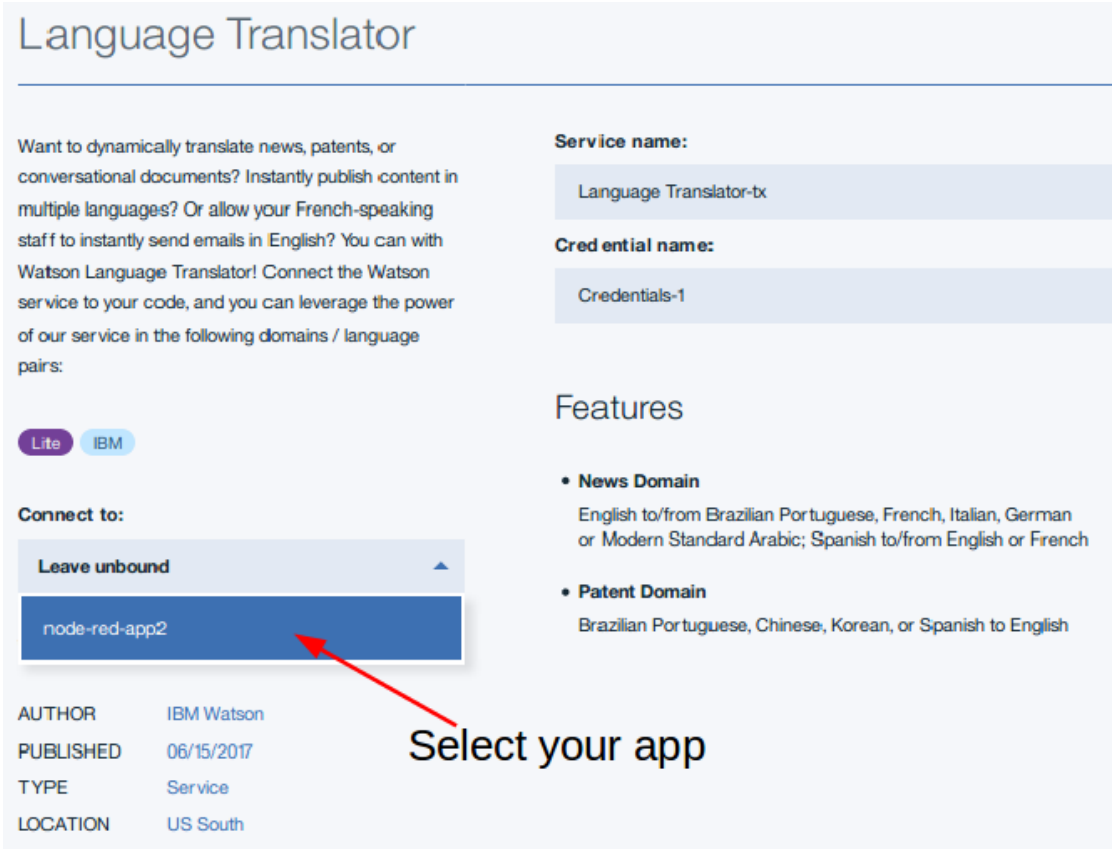
Step 4: Click “Connect new” under Connections



Step 5: Select Language Translator

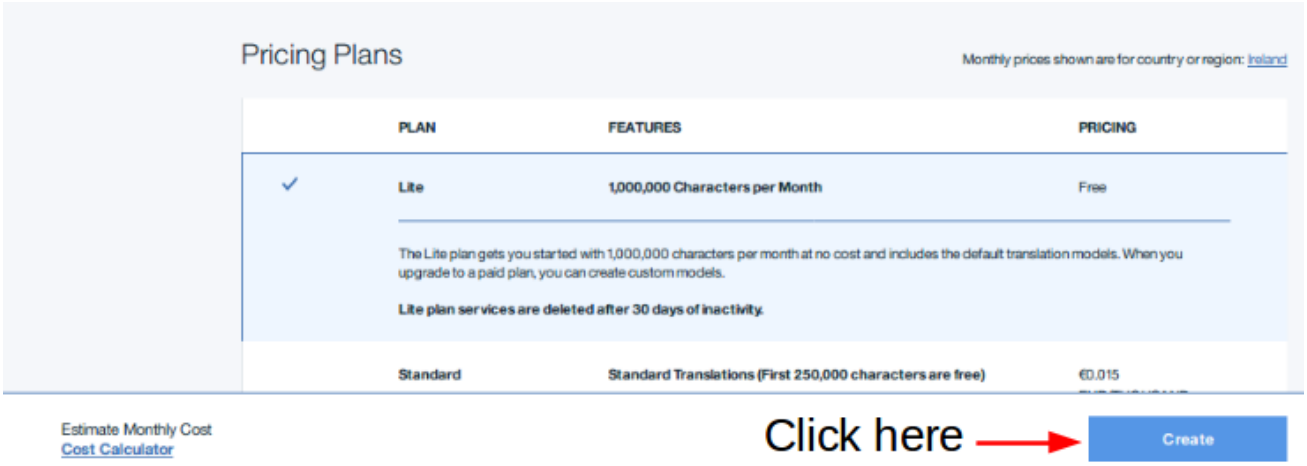


Step 6: Connect translator service to Cloud Foundry App



Step 7: Create translator service

In the bottom right hand side of the page. Click on “create” to create the service connected to your app.

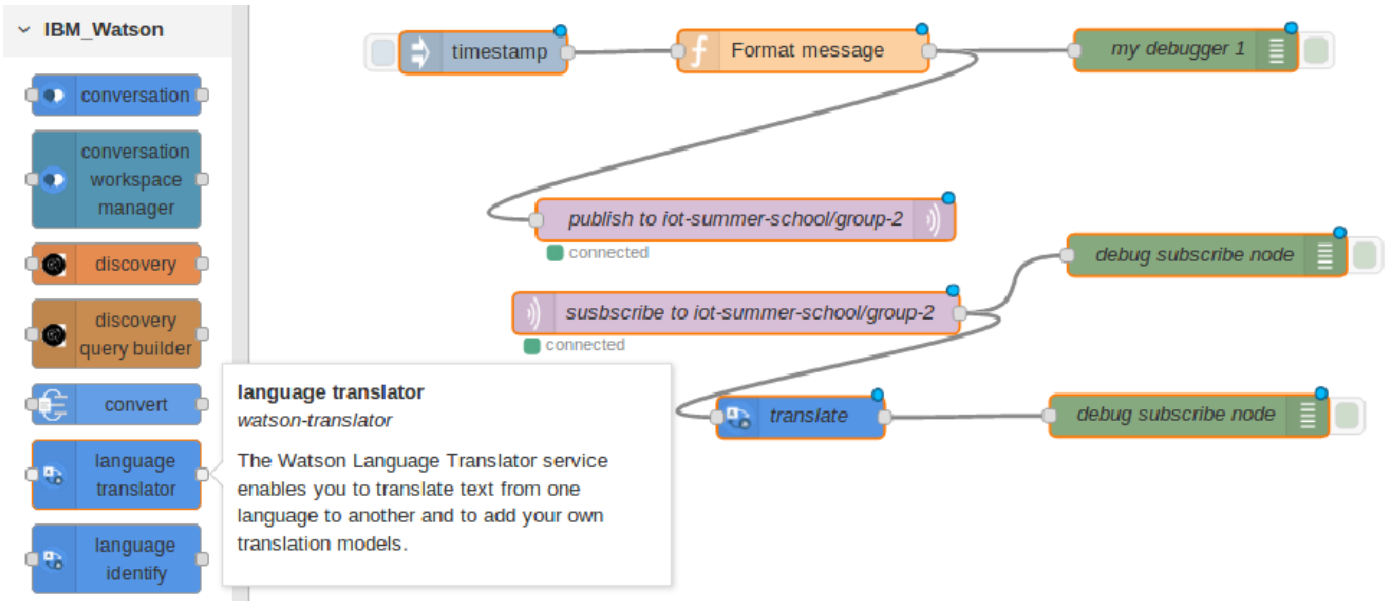


Step 8: Restart Cloud Foundry application

After creating a service you should get a prompt asking you to restart the application. Select yes.

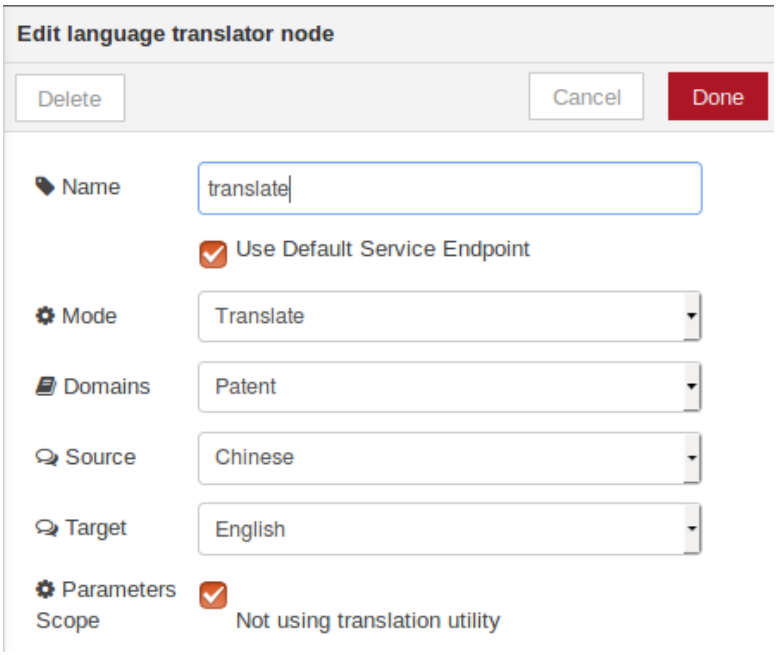
Warning: If you do not restart your application after adding a service, you will not see the service in the list of available nodes in the Node-RED flow editor screen.

Step 9: Add Watson “language translator” node



Add another debug node and wire it up to the output of the translate node (as shown in the above figure).

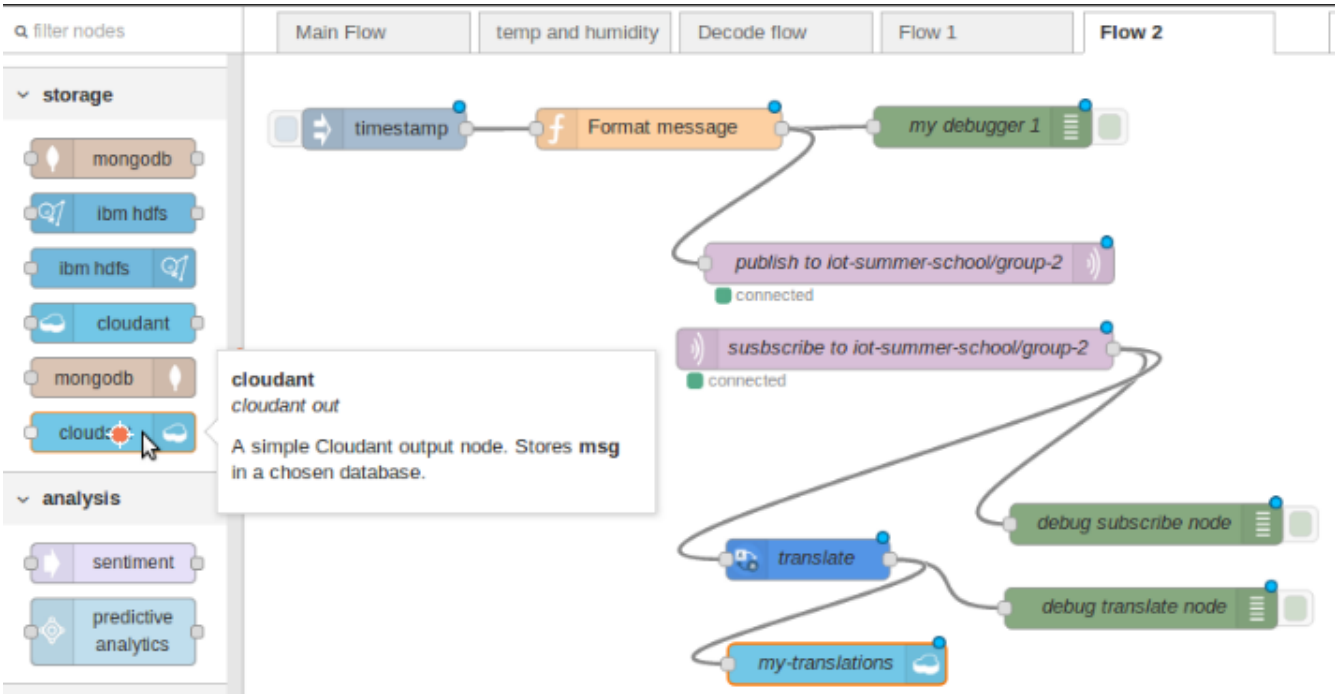
Step 10: Edit language translator node



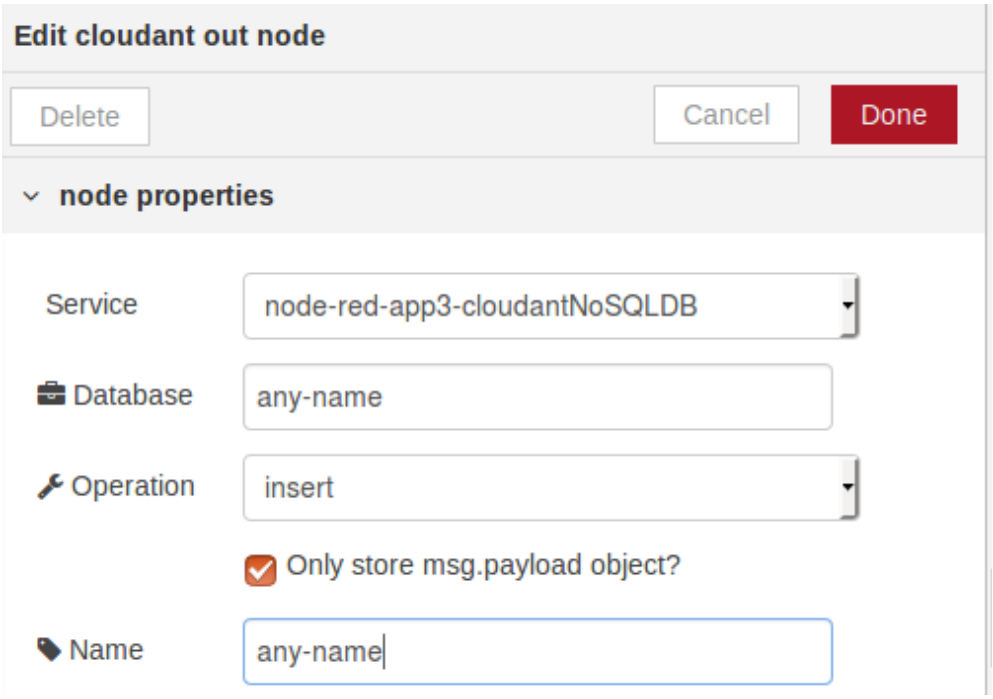
Step 11: Deploy and test translate using debugger node

Remember to hit the deploy button after making a change. You should see the message “Hello world” as the result of inputting “你好，世界” into the Watson translate service.

Step 12: Add Cloudant out node and connect to translate node



Step 13: Edit Cloudant node



Step 14: Deploy and click the inject node

Click on the inject node to fire messages.

What is happening when I click inject now?

When you click on the inject node, a msg flows into the function node.

In this node the msg.payload value is being changed to “你好，世界”.

The message is then sent to an mqtt broker running on Bluemix.

An mqtt in node subscribes to the topic **“iot-summer-school/group<group number>/<your-name>”.**

Now the message is flowing from the “mqtt in node” to both the “debug subscribe node” and the “translate” node.

After translating msg.payload from Chinese to english, the msg.payload is then stored in cloudant.

Appendix 2: OpenWhisk lab

IoT Summer School

OpenWhisk

Serverless Computing Lab

1 About OpenWhisk

Cloud is a game-changer for app development, and serverless computing is revolutionizing developers' access to some of the most powerful services cloud has to offer, including cognitive intelligence, data analytics and Internet of Things.

In traditional cloud computing models, developers often have to maintain their infrastructure and worry about when and how fast to scale, as well as potential resiliency issues if they are deploying apps and features in multiple regions. They are also charged for computing power even when an app is idling —a particularly painful point for startups and small companies with limited resources and early growth applications.

Serverless computing, which many are hailing as the next era of cloud computing, relieves many of these hassles by abstracting away infrastructure, running code and scaling on-demand. For developers, serverless platforms with a strong cognitive stack, such as IBM Bluemix OpenWhisk, gives them unprecedented access to powerful services such as Watson APIs, the Watson IoT Platform and weather intelligence.

As one of the few serverless platforms built on open standards, OpenWhisk acts as the invisible force within apps, binding together relevant events, actions and triggers. As data continues to grow and proliferate across all industries, serverless is certainly primed to become a standard for resourcefulness, scalability and connecting into the power of cognitive and IoT tools running on the cloud.

2 What is Serverless Computing?

Serverless computing refers to a model where the existence of servers is simply hidden from developers. I.e. that even though servers still exist developers are relieved from the need to care about their operation. They are relieved from the need to worry about low-level infrastructural and operational details such as scalability, high-availability, infrastructure-security, and so forth. Hence, serverless computing is essentially about reducing maintenance efforts to allow developers to quickly focus on developing value-adding code.

Serverless computing encourages and simplifies developing microservice-oriented solutions in order to decompose complex applications into small and independent modules that can be easily exchanged.

Serverless computing does not refer to a specific technology; instead if refers to the concepts underlying the model described prior. Nevertheless some promising solutions have recently emerged easing development approaches that follow the serverless model – such as OpenWhisk.

The term 'Serverless' is confusing since with such applications there are both server hardware and server processes running somewhere, but the difference to normal approaches is that the organization building and supporting a 'Serverless' application is not looking after the hardware or the processes - they are outsourcing this to a vendor.

3 Prerequisites

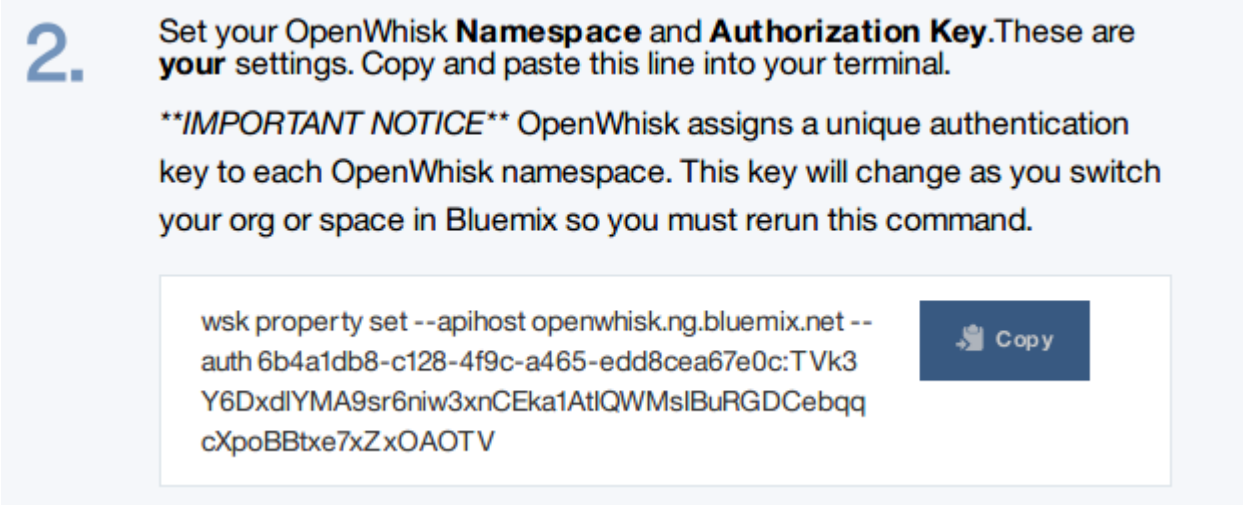
It is assumed you have completed the IoT Summer School Node-RED lab. At this stage you should have a Cloud Foundry Node-RED application in the United Kingdom domain.

Click the link provided below.

https://console.bluemix.net/openwhisk/learn/cli

Copy and paste the command “wsk propert set….”

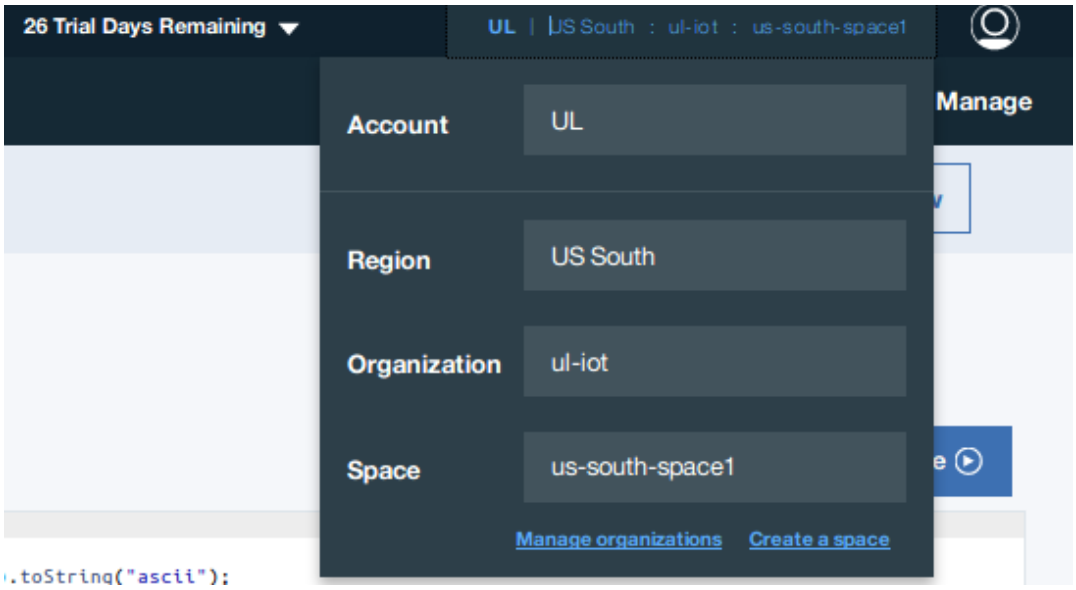
Email this command to 16117743@studentmail.ul.ie



After receiving the email, the teaching assistant will use this command on their Ubuntu machine that has the OpenWhisk CLI installed. The command will generate the authorization code needed for your Node-RED flow to interact with your own OpenWhisk actions.

4 OpenWhisk Lab

OpenWhisk is currently only available in the US South region. During this lab you will need to use the Bluemix dashboard in order to switch between your OpenWhisk actions in the US South region and your Node-RED application in the United Kingdom region.



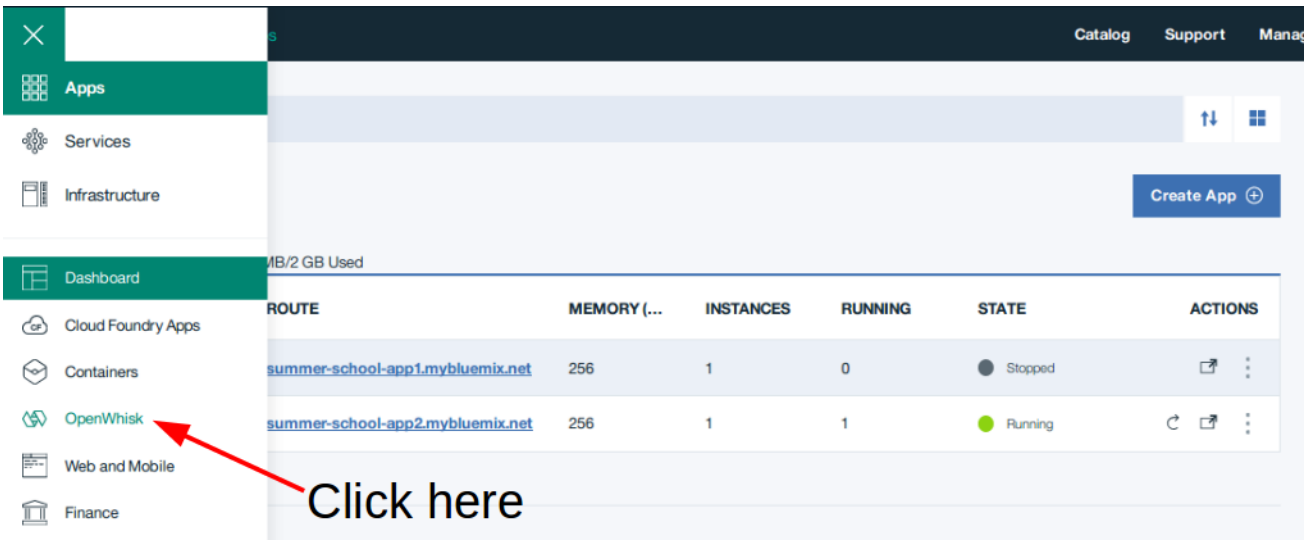
Should I just put my Node-RED application in the US South domain?

No. Your Node-RED application will be networking with a broker in the United Kingdom domain. The application would have significant latency and poor quality of service if deployed in the US South domain.

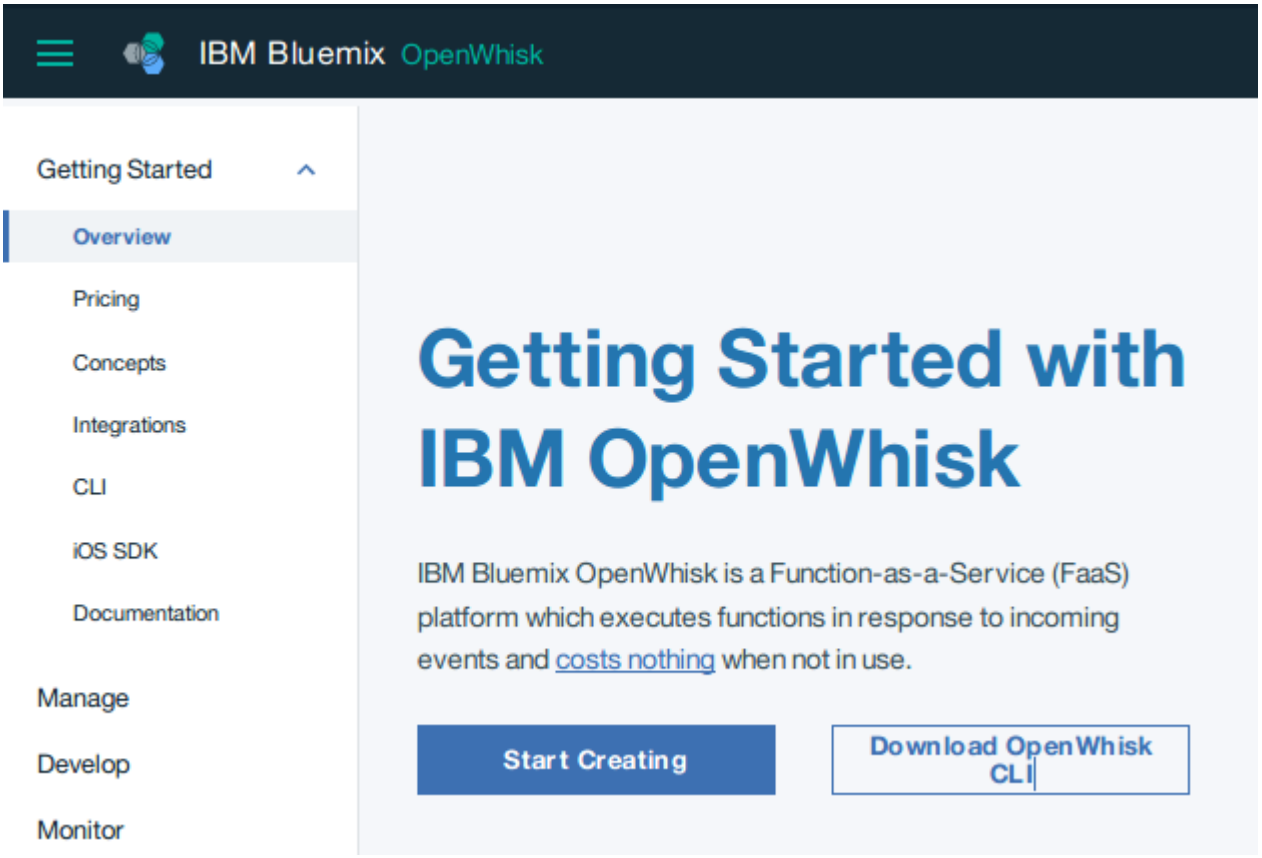
Does this mean there will be additional latency between my application and OpenWhisk actions, considering they are in two different regions?

Yes. If your application has strict real time analytic constraints, consider measuring the latency of the OpenWhisk action against an analytics instance running in the United Kingdom domain. Based upon these results make a decision on your implementation.

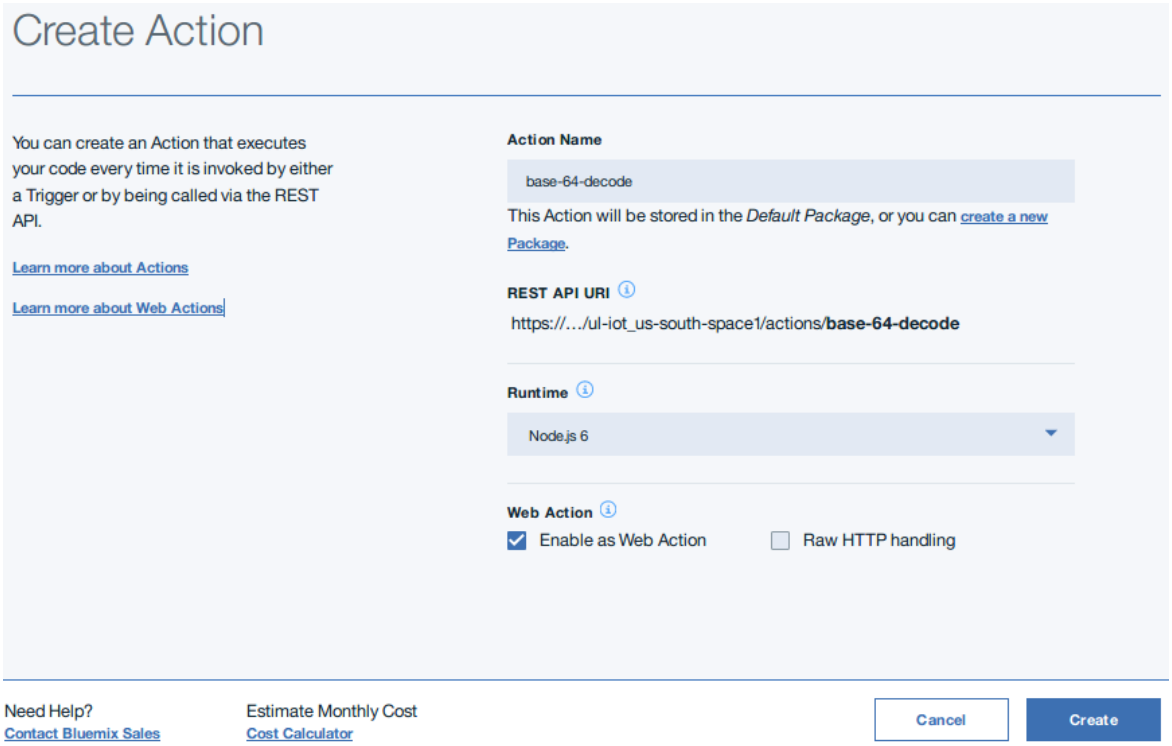
Step 1: Click on OpenWhisk



Step 2: Click on "Start Creating"



Step 3: Create Action



Step 4: Enter base64 Node.js code

Type and do not copy the below code

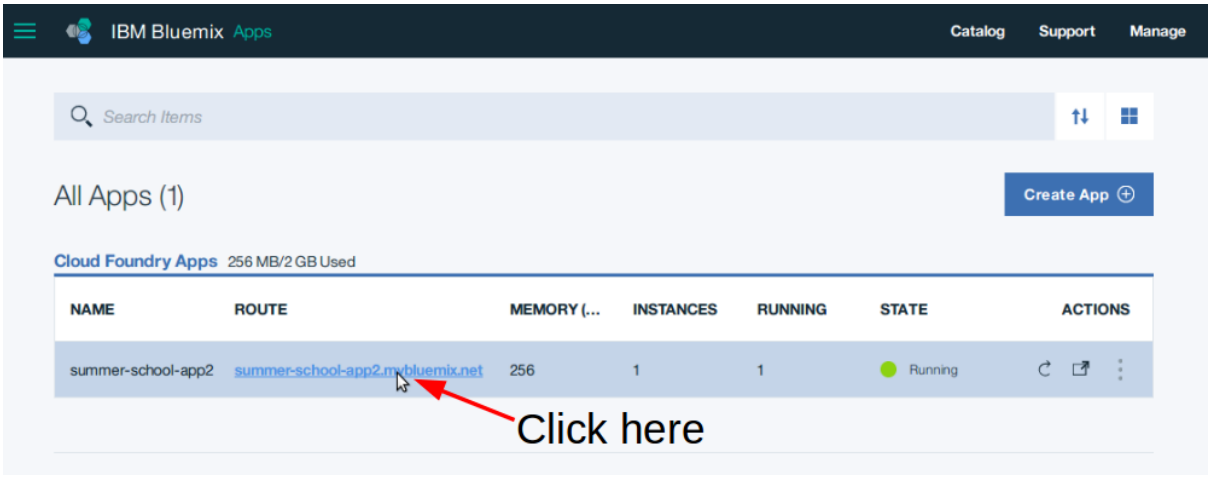
```
function main(params) {

    var b64string = params.name;

    var decodedString = Buffer.from(b64string, 'base64').toString("ascii");

    return { payload: decodedString };

}
```

Step 5: Go back to Bluemix main page

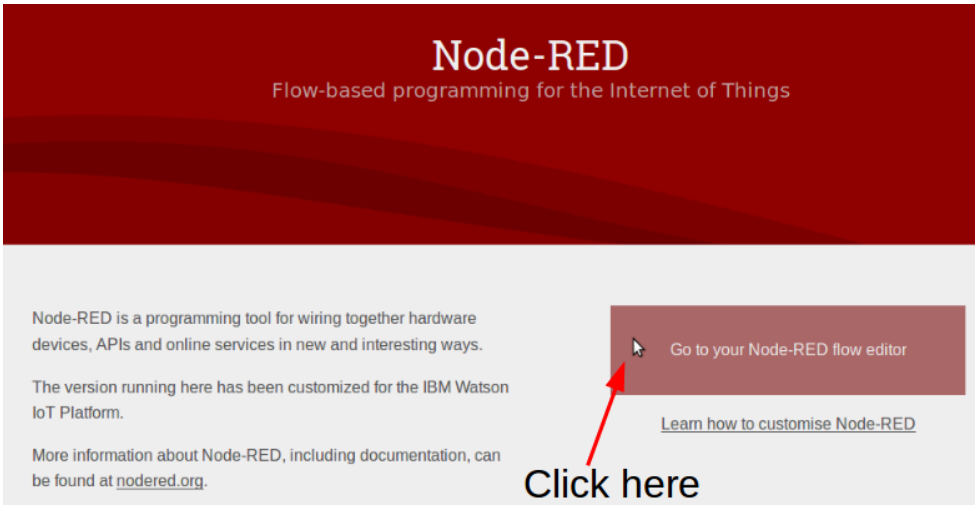
After clicking on save in the previous step, click on Bluemix as shown in the figure below.

Step 6: Switch region and click on Cloud Foundry App Route

If you don't see your Cloud Foundry application please check to see if you are in the United Kingdom region.

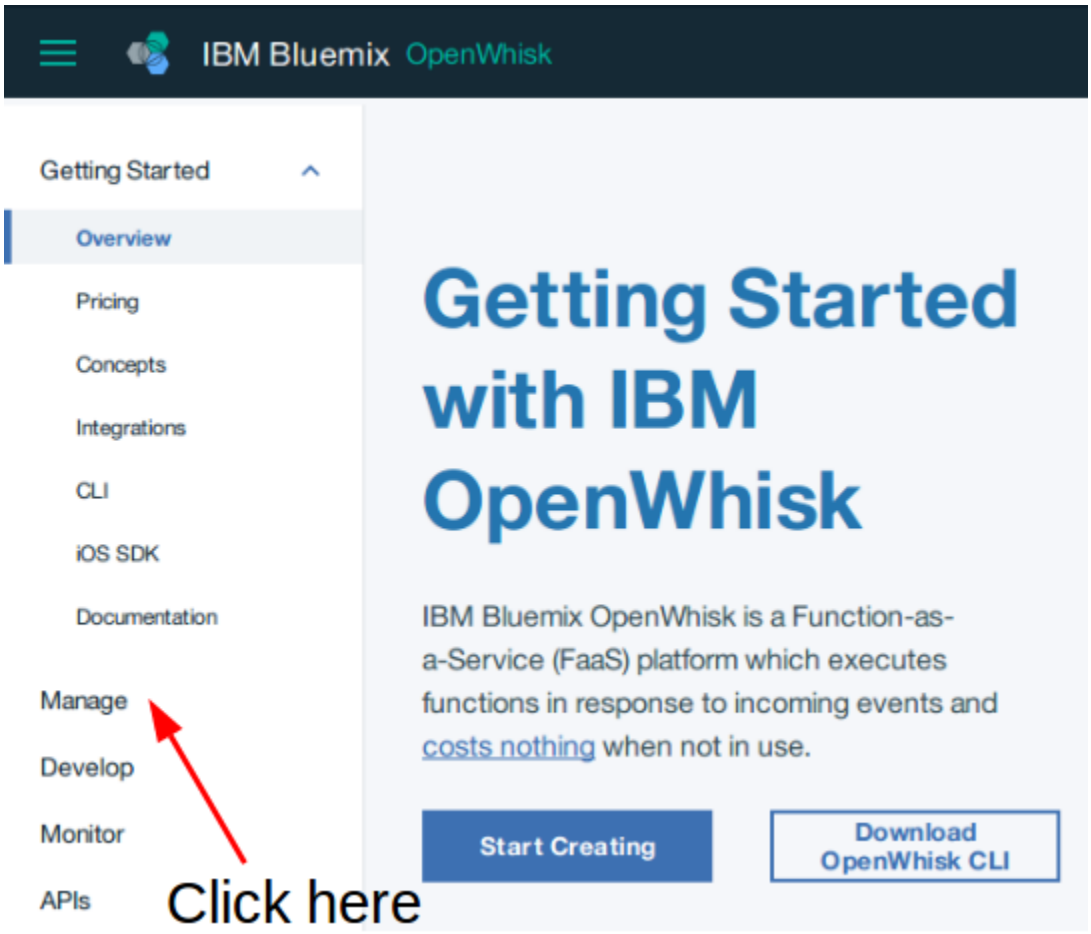


Step 7: Open Node-RED Flow editor in a separate tab

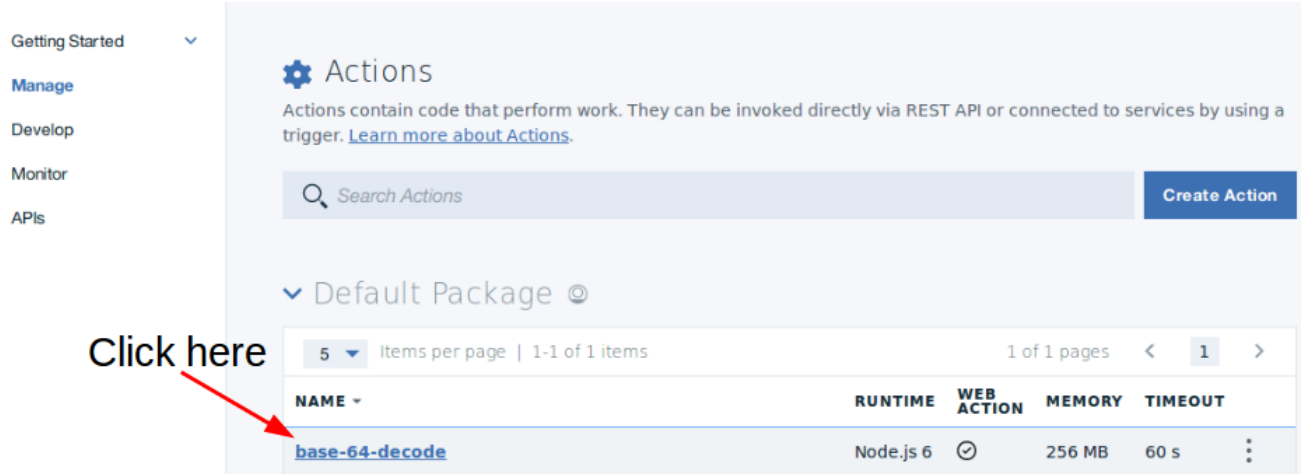


Step 8: Repeat step 1 to get back to the OpenWhisk screen

We want to be able to see our OpenWhisk Actions so we can implement changes. Do this by clicking on "manage". If you do not see OpenWhisk in the dashboard, check to see if you are in the US South region.



Step 9: Click on base-64-decode action



Step 10: Import Node-RED flow

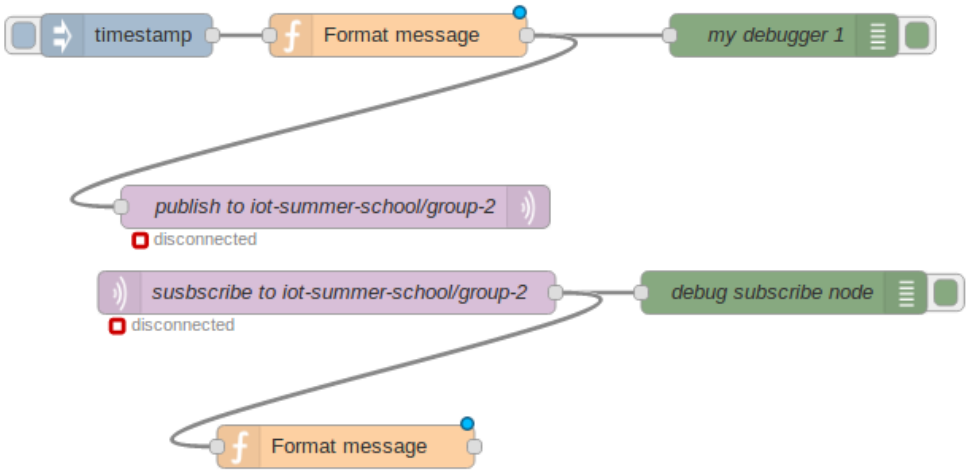
Switch to the tab that has the Node-RED flow.

Import the flow below.

[{"id":"8db20fe8.6ffa28","type":"tab","label":"Flow 3"},
{"id":"678eb277.25345c","type":"inject","z":"8db20fe8.6ffa28","name":"","topic":"","payload":"","payloadT
ype":"date","repeat":"","crontab":"","once":false,"x":102.88333129882812,"y":104.88333129882812,"wires"
:[["c3533458.8458c"]]},{"id":"c3533458.8458c","type":"function","z":"8db20fe8.6ffa28","name":"Format
message","func":"msg.payload = \"aGVsbG8gd29ybGQ=\";\n\nreturn
msg;","outputs":1,"noerr":0,"x":304.0777587890625,"y":104.76107788085938,"wires":
[["1575cb96.78f8bc","5ccb9ffb.4eb058"]]},
{"id":"1575cb96.78f8bc","type":"debug","z":"8db20fe8.6ffa28","name":"my debugger
1","active":true,"console":"false","complete":"true","x":580,"y":100,"wires":[]},
{"id":"5ccb9ffb.4eb058","type":"mqtt out","z":"8db20fe8.6ffa28","name":"publish to iot-summer-
school/group-2","topic":"iot-summer-school/group-
2","qos":"0","retain":"false","broker":"e639cd1b.be689","x":487.7165832519531,"y":203.58334350585938,"
wires":[]},{"id":"732958c3.79252","type":"mqtt in","z":"8db20fe8.6ffa28","name":"susbscribe to iot-
summer-school/group-2","topic":"iot-summer-school/group-
2","qos":"0","broker":"e639cd1b.be689","x":477.04998779296875,"y":266.24993896484375,"wires":
[["6e8658da.a96bf"]]},{"id":"6e8658da.a96bf","type":"function","z":"8db20fe8.6ffa28","name":"Format
message","func":"msg.payload = {\n   \"name\": msg.payload\n};\n\nreturn
msg;","outputs":1,"noerr":0,"x":470,"y":340,"wires":[["bc105a8.c4ed1a8"]]},
{"id":"bc105a8.c4ed1a8","type":"openwhisk-
action","z":"8db20fe8.6ffa28","name":"","func":"","namespace":"ul-iot_us-south-space1","action":"b64-
decode","params":[{"disabled":true}],"service":"937c50b4.19728","edit":false,"x":490,"y":420,"wires":
[["bd8e1c78.320a4"]]},
{"id":"bd8e1c78.320a4","type":"debug","z":"8db20fe8.6ffa28","name":"","active":true,"console":"false","co
mplete":"false","x":690,"y":420,"wires":[]},{"id":"e639cd1b.be689","type":"mqtt-
broker","z":"","broker":"IP-of-
Broker","port":"1883","clientid":"","usetls":false,"compatmode":true,"keepalive":"60","cleansession":true,"w
illTopic":"","willQos":"0","willPayload":"","birthTopic":"","birthQos":"0","birthPayload":""},
{"id":"937c50b4.19728","type":"openwhisk-service","z":"","name":"b64-
decode","api":"https://openwhisk.ng.bluemix.net/api/v1"}]

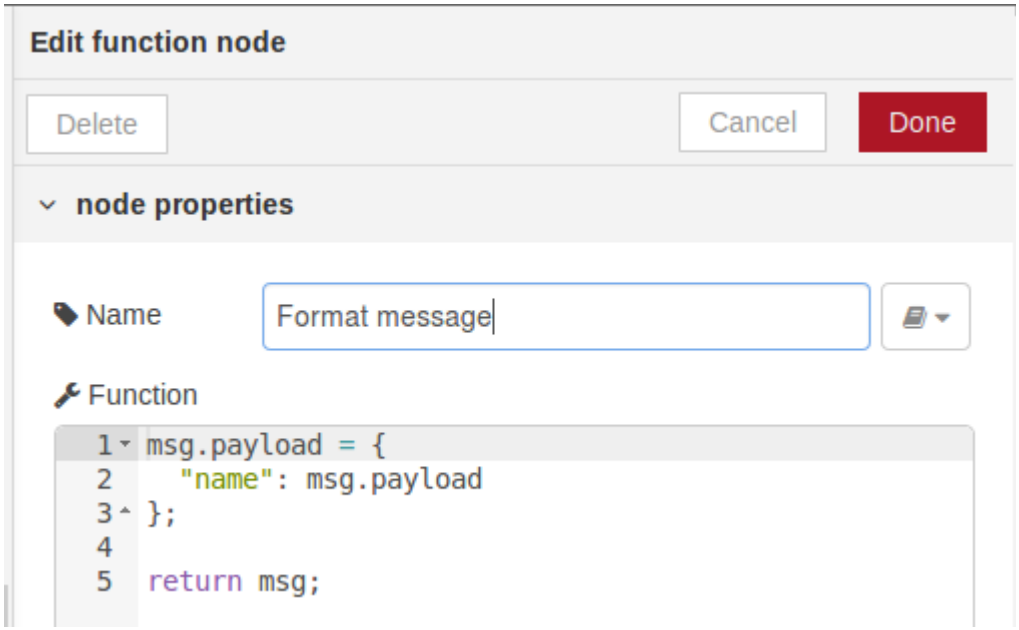
Step 11: Add a function node

Wire up the function node to the output of the mqtt subscribe node.



Step 12: Edit the function node

This function node changes the payload containing the base64 encoded value “hello world” into a key/value pair. This is required for the OpenWhisk action to be able to accept the arguments.



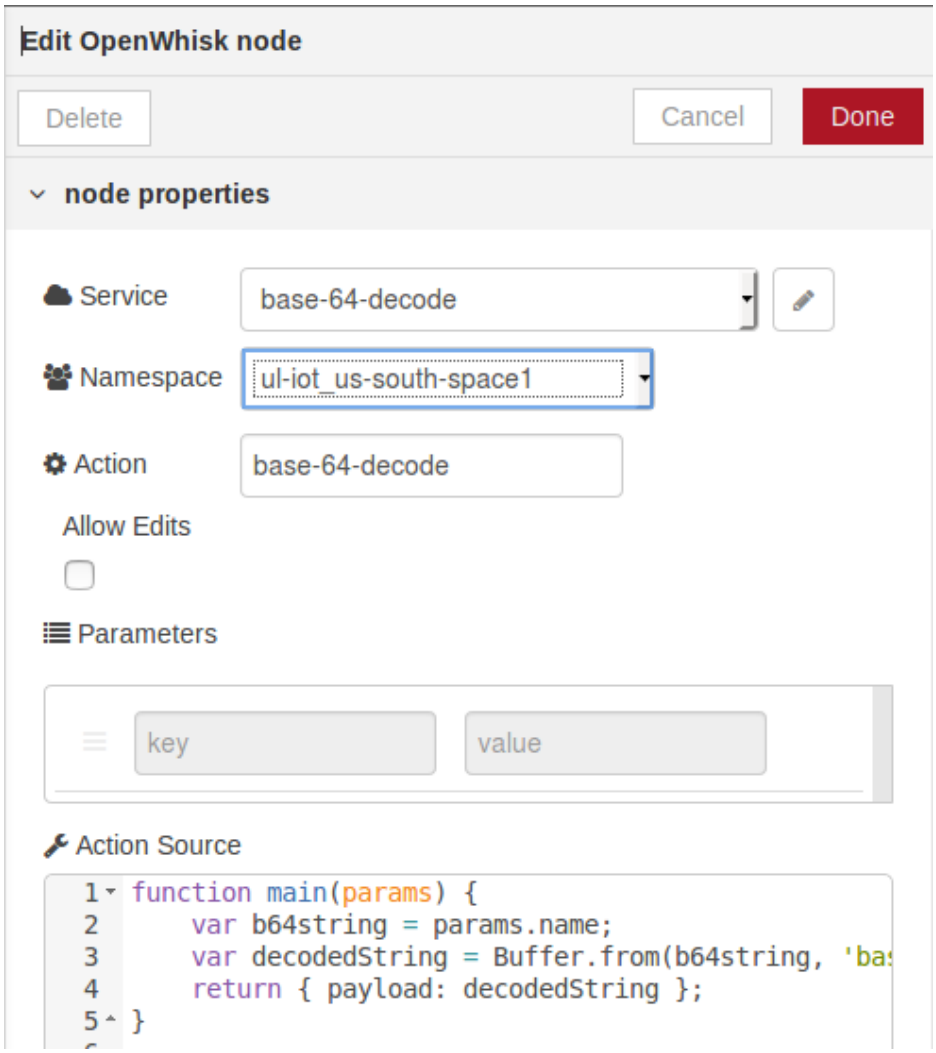
Step 13: Add OpenWhisk Action Node

The openwhisk-action node can be found under “function” node list. Drag the node onto the flow.

Connect the output of the newly added function node to the input of the openwhisk-action node.

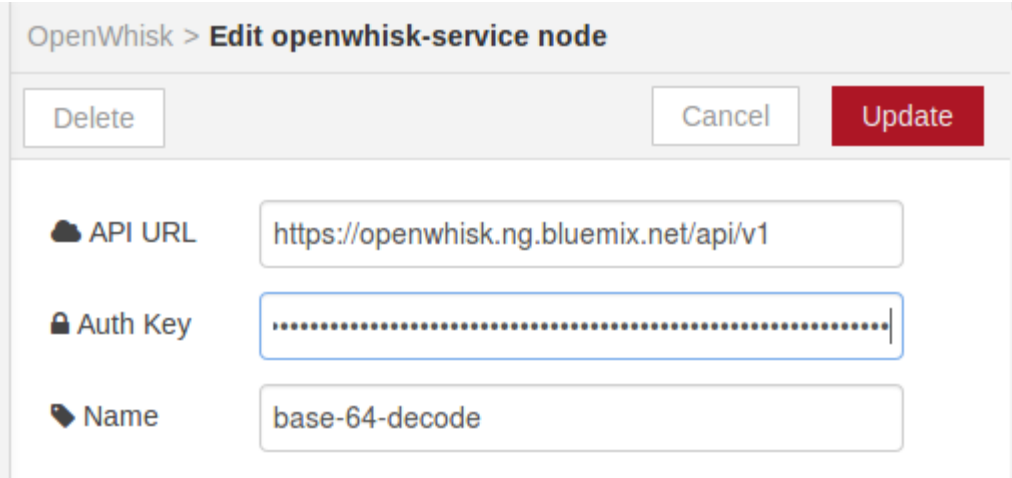
Step 14: Edit OpenWhisk node

Click on the pencil icon to the right of the service drop-down menu.



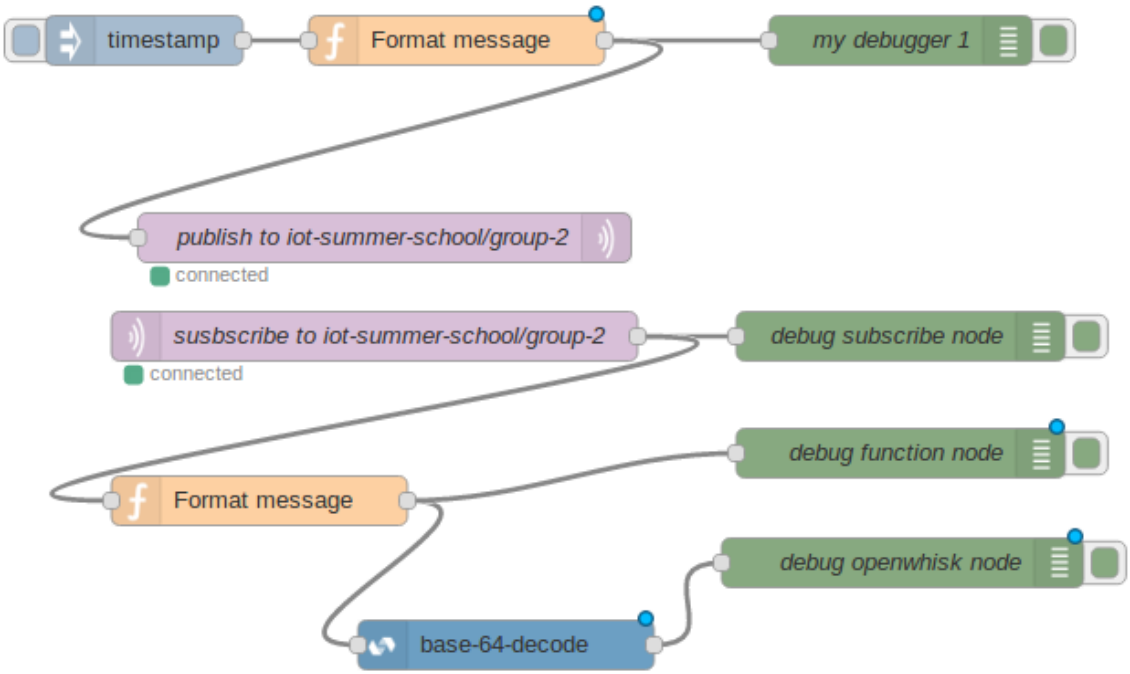
Step 15: Paste in the Auth Key obtained from the OpenWhisk CLI command

Make sure that the name field matches the name of your OpenWhisk action. If you are unsure about this step, please raise your hand and ask for assistance.



Step 16: Add debug nodes

Wire up the debug nodes as shown in the figure below.



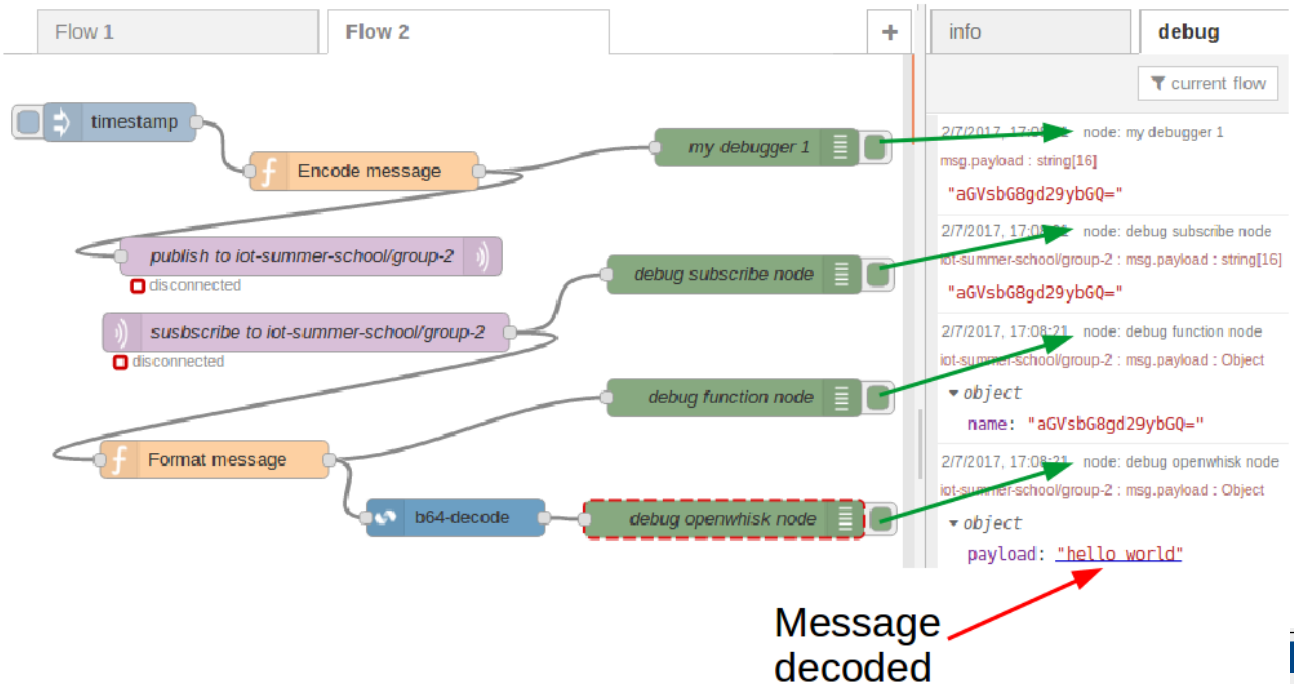
Step 17: Configure mqtt nodes

Both the mqtt nodes need to be configured. Change the topic and ip address field.

If you are unsure about this step, please raise your hand and ask for assistance.

Step 18: Deploy and test

Click the inject node to test the OpenWhisk action.



5 FAQ

I am not understanding the encoding/decoding part of this lab?

In the first function node “Encode message” you are encoding the message “hello world” into base64.

The result of this is “aGVsbG8gd29ybGQ=”.

This value is assigned to msg.payload and is then sent to the mqtt broker using the “mqtt out node”.

The “mqtt in node” subscribes to the topic “iot-summer-school/group<group number>/<your name>”.

This is the topic the “mqtt out node” published to.

The “mqtt in node” receives a message with a payload

msg.payload = “aGVsbG8gd29ybGQ=”

This message flows into the function node “format message” where the value is formatted into JSON.

```
msg.payload = {
    "name": msg.payload
};
return msg;
```

OpenWhisk accepts a single argument in JSON format. So we create a JSON key/value pair.

Where “name” is the key and “aGVsbG8gd29ybGQ=” is the value.

OpenWhisk can access this value using **params.name** as seen in the OpenWhisk action code below.

```
var b64string = params.name;
var decodedString = Buffer.from(b64string, 'base64').toString("ascii");
return { payload: decodedString };
```

The above code decodes the value “aGVsbG8gd29ybGQ=”, the result of which is “hello world”.

This value decodedString is then returned.

I am not seeing OpenWhisk in the dropdown menu?

Check the top right hand corner to see if you are in a US-south region. If you are in the US-South region and still cannot see an option for OpenWhisk, call over the teaching assistant.

How might I use OpenWhisk for my analytics application?

One use case would be to have a trigger attached to an mqtt feed. For example

Listening to topic **“iot-summer-school/temperature”**

Every time a message is published to this topic, the associated action is triggered by the “trigger node”.