Redbooks
ibm.com/redbooks

Hybrid Cloud Event Integration
Integrate Your Enterprise and Cloud with Bluemix Integration Services

Jesse Aulsebrook
Richard Scott Balson
Maxime Cenatiempo
Vasfi Gucer
Shamim Hossain
Muhammad Atif Mehmood
Raj Mehra
Duy Nguyen
Bancha Setthanan
Amar Shah

Cloud

WebSphere

IBM

Redbooks

International Technical Support Organization

Hybrid Cloud Event Integration: Integrate Your Enterprise and Cloud with Bluemix Integration Services

February 2016

SG24-8281-00

Note: Before using this information and the product it supports, read the information in “Notices” on page xix.

First Edition (February 2016)

This edition applies to IBM Bluemix Version 1.0.

Contents

Figures

Tables

Examples

Notices

THIS PAGE INTENTIONALLY LEFT BLANK

Preface

The event-centric hybrid cloud integration revolves around applications running based on events or messages. The new event-centric approach to hybrid cloud aims to simplify the task of managing these messages while increasing the overall reliability of the system. Event-centric applications work well in the cloud due to the varying intensity and frequency of events. These fluctuations fit well into a cloud infrastructure that can dynamically scale to fit those needs. An event-centric approach cuts down on communication overhead for an application, thus helping to speed up the development process.

IBM® Hybrid Integration Services is a set of hybrid cloud capabilities in IBM Bluemix® that allows businesses to create hybrid clouds by connecting their Bluemix environment to on-premises systems at the application programming interface (API), data, or event level.

In November 2015, the IBM International Technical Support Organization (ITSO) IBM Redbooks® team published a Redbooks publication that covers hybrid cloud scenarios with Bluemix for API and data integrations, *Hybrid Cloud Data and API Integration: Integrate Your Enterprise and Cloud with Bluemix Integration Services*, SG24-8277, and can be found at the following website:

http://www.redbooks.ibm.com/Redbooks.nsf/RedbookAbstracts/sg248277.html?Open

This book, *Hybrid Cloud Event Integration: Integrate Your Enterprise and Cloud with Bluemix Integration Services*, SG24-8281, is a companion book to SG24-8277 and focuses on event-centric hybrid cloud integrations with Bluemix.

Authors

This book was produced by a team of specialists from around the world working in IBM Melbourne, Australia.

Jesse Aulsebrook is an IT Consultant with IBM Global Business Services® (GBS). He has over three years of experience in web application development and Enterprise Integration. He is currently working in the IBM Watson™ space with a focus towards analytics. Jesse is very passionate about cloud application development and spends most of his spare time working with Bluemix offerings. Jesse also provides specialist technical support to the Australian Football League on match days over weekends.

Richard Scott Balson is a Technical Analytics Consultant with IBM GBS in the Business Analytics and Strategy team. Richard has a PhD in Biomedical Engineering, where he used analytical tools and models to improve the understanding of the mechanisms involved in neurological disorders. He also has experience developing analytics and statistical models for business. Richard is currently focused on developing advanced analytics methodologies using multiple tools for both structured and unstructured data.

Maxime Cenatiempo is a long-term IBMer based in Sydney. He has been working for nine years as an IBM Certified IT Specialist dedicated to client success. Throughout his career, he had the opportunity to work across various industries in multiple countries. He has been focusing on innovative technologies in diverse domains including enterprise service bus, Business Process Management, mobile app development and cloud computing. He is currently working as an IBM MobileFirst/Bluemix Technical Specialist and holds a Master's degree in Computer Science from the National Institute of Applied Sciences in France.

Vasfi Gucer is an IBM Redbooks Project Leader with the IBM International Technical Support Organization. He has more than 18 years of experience in the areas of systems management, networking hardware, and software. He writes extensively and teaches IBM classes worldwide about IBM products. His focus has been on cloud computing for the last four years. Vasfi is also an IBM Certified Senior IT Specialist, Project Management Professional (PMP), IT Infrastructure Library (ITIL) V2 Manager, and ITIL V3 Expert.

Shamim Hossain is an IBM Certified Cloud Solution Advisor and Cloud Solution Architect. He leads a cloud consultancy laboratory in IBM Australia to develop born-on-the-cloud applications using agile methodologies and design thinking. He is a Redbooks publication thought leader and official IBM Cloud Computing and Smarter Computing Ambassador. He holds a Master of Telecommunications Engineering from the University of Melbourne and a Bachelor of Computer System Engineering (first-class honors) from Monash University. His expertise and interests include different areas of cloud computing, mobile computing, optical fibre communications, broadband, Internet engineering and the Internet of Things. He co-authored a book entitled *Cloud Computing Service and Deployment Models: Layers and Management* by IGI Global.

Muhammad Atif Mehmood is a Managing Consultant in the Cloud Development and Integration Services practice with IBM Australia. He has over seven years of experience with Enterprise Integration and Business Process Management both on sales and delivery projects. Atif is currently focused on IBM Bluemix hybrid cloud development platform and Solution as a Service offerings from IBM. Atif holds a Master's degree in Computer Science from University of Melbourne, Australia.

Raj Mehra is an IBM Cloud Technical Rock Star and a Watson Ambassador who is working as a Senior Architect for IBM Cloud Software Services, ANZ. He is a Master Certified IT Specialist with the Open Group. He has over 18 years experience architecting Business Process Management, service-oriented architecture and system integration solutions using IBM middleware products. During the last two years, he has been focussing on emerging technologies to deliver hybrid cloud and mobile solutions using IBM Watson and other services on IBM Bluemix. In this role, he has earned recognition for his thought leadership and work in client hackathons.

Duy Nguyen is an IBM Certified IT Architect with solid experience in IBM and open technologies. He is also an IBM Advisory Software Engineer working in the Cloud Engineering and Services team in the IBM CIO Office, Transformation and Operations group in the US. His daily job is helping IBM to transform using new technologies, specifically cloud, mobile, and other cutting-edge initiatives. He is focusing on mobile and cloud, including the creation of mobile solutions for IBM employees, and providing his expertise in assisting IBM clients with enterprise mobile and cloud engagements as needed. His core experiences are in web, security, cloud, and mobile technologies.

Bancha Setthanan is a Managing Consultant with IBM GBS Cloud Development and Integration Services in Melbourne, Australia. He has experience in software analysis, design and development in Cloud Application Development, Business Process Management, Service-Oriented Architecture, and Enterprise Integration. He specializes in API management, hybrid cloud integration, and modern web application frameworks, such as ionic and Meteor.

Amar Shah is a Serviceability Architect working with the IBM Integration Bus Level3 support team. He is responsible for worldwide support for clients of IBM Integration Bus and the serviceability enhancement of products. He is also the designated Lab Advocate for key clients and gets involved in providing advice and consultancy on product solution and usage best practices. Amar Shah has been associated with IBM for the past 17 years and holds a Master's degree in Software Systems from Birla Institute of Technology, Pilani, India.

Thanks to the following people for their contributions to this project:

Erick Griffin, Matthew Hamann, Valerie Lampkin
IBM US

Simon Gormley, Rob Nicholson, Rob Phippen, Nick Plumb, Andrew Schofield, Naomi Scott
IBM UK

Ashley Fernandez
IBM Australia

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  https://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

Part 1

Introduction to hybrid cloud concepts and products

In this part, we introduce the hybrid cloud concepts within the context of several use cases and also describe some of the IBM products and services that you can use to implement a hybrid cloud environment with IBM Bluemix.

The following chapters are covered in Part 1:

- Chapter 1, “Introduction to hybrid clouds” on page 3
- Chapter 2, “Introduction to IBM Bluemix services for hybrid cloud” on page 9
- Chapter 3, “Introduction to IBM messaging and integration products” on page 37

1

Introduction to hybrid clouds

In this chapter, we introduce the business challenges for integrating the enterprise's on-premises applications with the cloud solutions. Finally, in this chapter we look at business real-world use cases to showcase the business value of hybrid cloud solutions. In addition, we set the stage for how hybrid cloud solutions help drive business value to enterprises.

This chapter has the following sections:

- 1.1, "Business challenges for seamless integration between cloud and on-premises applications" on page 4
- 1.2, "Hybrid cloud customer scenarios and use cases" on page 5

1.1 Business challenges for seamless integration between cloud and on-premises applications

In this section, we cover the business challenges and drivers that motivate companies to integrate their cloud and on-premises applications. We also look at an approach to seamlessly integrate cloud and on-premises applications to implement hybrid cloud solutions.

1.1.1 Rapid innovation

In today's business environment, the established business models are being frequently challenged. New entrants in the marketplace are rapidly developing innovative solutions. A number of businesses are using the cloud environment to experiment and validate with new ideas with minimal costs. These solutions might use new channels to conduct their business. The remarkable success for Uber is one such example. The taxi industry around the globe is reevaluating their business models. Such kinds of changes are occurring at a rapid rate and it is not just limited to one industry. The established businesses are also using the cloud to fast track innovation.

The IBM Bluemix environment is used by an increasing number of businesses to rapidly develop and validate ideas using prototype applications. These applications can serve as a seed for the development of full-blown enterprise applications or can open new, more effective channels for conducting business.

1.1.2 Using enterprise solutions

The established business usually has a number of enterprise applications. These applications support the core business needs. In many cases, the enterprise applications are a very valuable asset that provides an organization with a competitive edge over its peers. Some of the legacy applications might have been developed using some old technology, which might not be directly compatible with the emerging programming models.

1.1.3 Best-in-class solution with cloud and on-premises applications

To optimize the business outcomes, the enterprise needs to continue to derive the benefits from their stable enterprise applications along with harnessing the innovation and exploitation of new business channels that are facilitated by the born-on-the-cloud applications.

The first challenge for such a solution is secure connectivity. The IBM Bluemix Secure Gateway service can securely connect the on-premises applications to the cloud applications running on IBM or third-party cloud.

The other major challenge would be the connectivity between disparate technologies. The message-oriented middleware like IBM MQ and IBM Integration Broker provide a robust and mechanism connectivity between heterogeneous applications and programming models. They can be used to expose capabilities of existing applications to the born-on-the cloud applications and vice versa.

A combination of these two capabilities would greatly ease the goal of building enterprise hybrid solutions.

1.2 Hybrid cloud customer scenarios and use cases

In this section, we cover how hybrid cloud solutions solve the business challenges of several industries. The goal in this section is to showcase how different industries benefit from hybrid cloud solutions.

1.2.1 CompanyA background

CompanyA is a traditional market research company. They were facing much competition from newer opponents. They were looking for new ways to protect and grow their market share. They used the IBM Bluemix environment as an innovation platform and organized a hackathon to come up with the most innovative and cost-effective way of gaining a competitive advantage over their peers.

The competition was won by a voting app that is capable of running on a browser in a desktop or mobile device. The users provide their details along with their preferred option, with one click of a button to vote, and this information is recorded in a relational database. As an incentive for voters, each vote goes in the draw for a periodic prize.

The voting app prototype was tested by friendly parties to validate the concept. It received very positive feedback.

CompanyA is now ready to implement the solution in production. They want to evolve this solution to their main enterprise data collection tool and provide additional capabilities, such as the current leader board and providing a list of all the votes for a certain option.

1.2.2 CompanyB background

CompanyB is a utilities company. They support a large infrastructure of assets spread over a large area. The company uses IBM Maximo® Asset Management to manage its assets.

In the current environment, CompanyB receives calls from the public about broken infrastructure. They manually allocate work to maintenance teams using Maximo Asset Management. Maintenance teams are dispatched and problems are fixed.

The current system is not efficient. Often, there are many problems that are associated with weather phenomena, such as excessive rain or heat. A number of calls arrive at the same time and it causes delays in dispatching teams to fix the problems.

CompanyB is looking at smarter ways to capture problems and trigger work allocation.

1.2.3 CompanyC background

CompanyC is a fast growing company. It is growing by consolidating the business from many smaller companies. CompanyC established a master CRM of all its clients by using a cloud-based Salesforce application. The acquired companies use their existing applications for their day-to-day business.

CompanyC wants to maintain client details in one place: The Salesforce application. They want to synchronize the master data from Salesforce to all other applications. They are looking for a cost-effective way of synchronizing the client details.

1.2.4 CompanyD background

CompanyD is a dominant telecommunication provider. They have a strong market share. The rapid entry of new competitors in the marketplace is putting pressure on their client base. CompanyD brought out some competitive offerings, but that has not provided the expected benefits because they are targeted to a generic audience and the message is not effective for the clients who signed up with a competitor.

CompanyD wants to ensure that the competitive offerings' messages are reaching the right clients. They also want to ensure that the message reaches them before they decide to switch to the competitor.

1.2.5 CompanyA challenges, strategy, and solutions

The first challenge for CompanyA is the regulatory compliance requirement. They need to store the data captured into a database inside the corporate firewall. They overcome this challenge by establishing a Secure Gateway between IBM Bluemix and their corporate data center.

The solution is popular with the clients but then it slows down due to a heavy workload. To improve the performance, they use IBM Bluemix MQ Light service to quickly record data. The slow task of persisting data to a database is carried out by a background job.

The enterprise application architects want access to the votes' data as they are being cast. They want to build an application that consumes this data. The infrastructure management team wants to manage the infrastructure where business data is stored on a temporary basis. The solution is reconfigured to use the enterprise IBM MQ server instead of the IBM MQ Light service on Bluemix.

Finally, an enterprise application is built by using the enterprise service bus, the IBM Integration Broker. Additional functionality is provided, real-time results are published, and users get access to votes that have been cast for each candidate.

See Chapter 6, "Asynchronous processing through IBM MQ Light service" on page 109 for implementation of this scenario.

1.2.6 CompanyB challenges, strategy, and solutions

CompanyB uses the Internet of Things (IOT) device to instrument the critical infrastructure. The device monitors critical parameters for the infrastructure. When any of the parameters move to an unacceptable level, the device sends a message to the IBM Bluemix Internet of Things service in Bluemix. The device is registered with the IOT server. IOT service accepts messages only from preregistered devices.

The IOT service is connected to an application in IBM Bluemix. The application is connected to the corporate database using the Secure Gateway. On receipt of a message, the application creates a work order on IBM Maximo.

The solution reduces the time and effort required to initiate work requests.

See Chapter 8, "Integrating events from Internet of Things with Enterprise Asset Management systems" on page 217 for implementation of this scenario.

1.2.7 CompanyC challenges, strategy, and solutions

CompanyC investigates the options for synchronizing the data from Salesforce on the cloud to a number of applications in multiple data centers.

The most flexible option is a capability of the Salesforce application to invoke a Representational State Transfer (REST) application programming interface (API) when the master client information changes. CompanyC uses the IBM StrongLoop® service on IBM Bluemix to build a REST Service meeting using the Salesforce API specification. The API receives the request and stores the data to a local database. Copies of this database can be distributed to multiple applications who can write their own code to import the updated data into their own application.

This scenario is implemented in Chapter 7, “Synchronizing data from Salesforce to a remote enterprise system” on page 189.

1.2.8 CompanyD challenges, strategy, and solutions

CompanyD identifies the client behavior patterns when they are considering switching their service provider. They use the IBM Analytics Solutions in the IBM Bluemix environment to automate the process of identifying at-risk customers. They send targeted offers to the clients at risk in order to maintain client loyalty.

Refer to Chapter 9, “Demonstration of analytics and real-time event detection” on page 253 for implementation of a similar IBM Analytics Solutions scenario in the IBM Bluemix environment.

2

Introduction to IBM Bluemix services for hybrid cloud

This chapter provides an introduction to hybrid integration services on IBM Bluemix as well as information about the key capabilities of each service, focusing on manageability, security, scalability, and so on.

This chapter contains the following sections:

Important: Because this book focuses on hybrid cloud event integration with IBM Bluemix, we do not cover some of the Bluemix hybrid integration services, such as *API Management* and *Dataworks*. These services are covered in detail with scenarios and demonstration videos in the companion IBM Redbooks publication *Hybrid Cloud Data and API Integration: Integrate Your Enterprise and Cloud with Bluemix Integration Services*, SG24-8277, which is available at the following link:

`http://www.redbooks.ibm.com/Redbooks.nsf/RedbookAbstracts/sg248277.html?Open`

2.1 Secure Gateway

The *IBM Secure Gateway service* in Bluemix provides a secure way to access your on-premises or cloud data from your application running in Bluemix over a secure passage that is the gateway. The basic scenario for Bluemix to on-premises integration is the integration between the new born-on-the-cloud Systems of Engagement (SOEs) and the existing legacy Systems of Record (SORs). Data is typically accessed from a SOR, such as a database or an application or web service.

The Secure Gateway works by using a client on the on-premises side to connect to your Bluemix organization. This is shown in Figure 2-1.

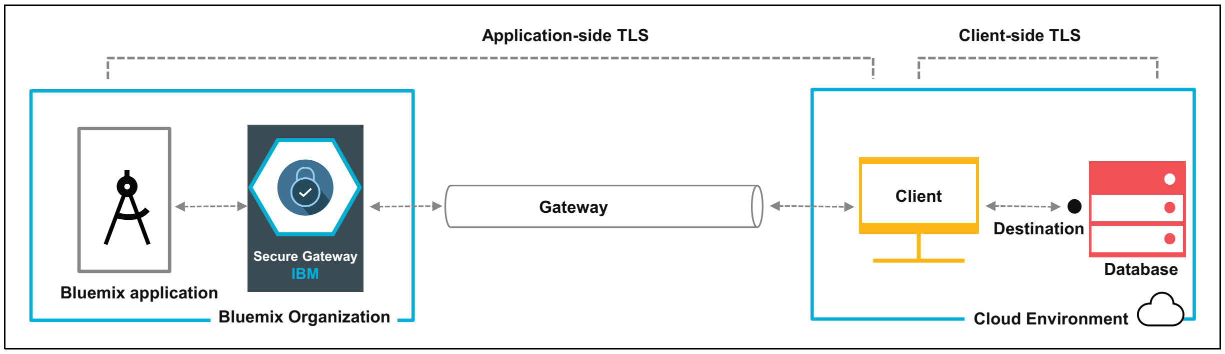


Figure 2-1 Secure Gateway high-level architecture overview

Table 2-1 defines some key terms for Secure Gateway.

Table 2-1 Key terms for Secure Gateway

Term	Definition
Client	The process that establishes the on-premises or cloud side of the gateway and forwards requests to the destinations.
Gateway	The tunnel between your Bluemix app and on-premises or cloud environment.
Destination	The endpoint at which your on-premises data can be accessed.
Application-side TLS	Security between your Bluemix app and on-premises or cloud client.
Client-side TLS	Security between the client and on-premises or cloud destination or data.

The basic steps to set up and use the Secure Gateway service are as follows:

1. Provision a Secure Gateway service and bind it to your application.
2. Create a gateway (Name It).
3. Connect the gateway to a client (Connect It).
4. Add a destination to the gateway (Add Destinations).
5. Use the destination in your application.

Note: You can provision only one Secure Gateway service per space. You can have multiple Bluemix applications bind to the same Secure Gateway instance.

The gateway contains the configuration information for establishing a tunnel between the Secure Gateway client running in your environment and Bluemix. When adding a gateway, you can choose to enforce increased security over who is able to start a gateway. An optional security token can be provided when they connect the Secure Gateway client.

Figure 2-2 shows the Add Gateway window of the Secure Gateway showing the three steps to set up the gateway and destination.

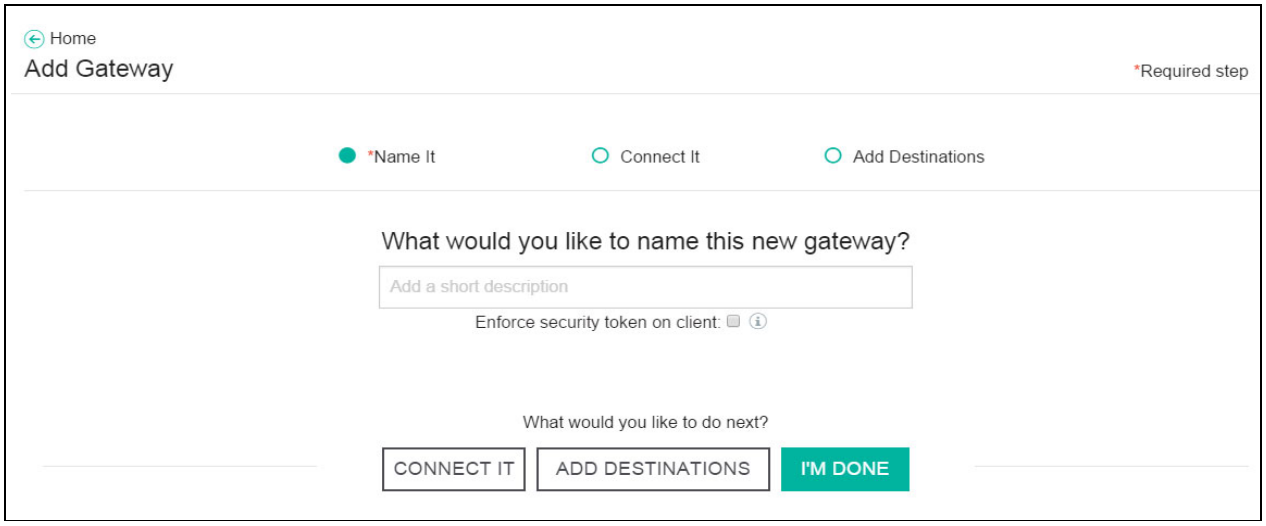


Figure 2-2 Add Gateway window

When you name your gateway, you can connect your gateway to a client. At the time of this writing, the following three options were available for the client:

- Client installer: IBM Secure Gateway native client installer for different platforms like Linux, Microsoft Windows, and Mac OS. A Docker container running the Secure Gateway client. Docker provides a convenient *run anywhere* option.
- Docker: The Docker image built with the same capabilities of the client installer option but with the portability advantages of Docker, which can be run anywhere.
- IBM DataPower®: Appliance optimized solution with the same base features with the Docker client option, but with additional security enforcement capabilities.

The gateway client is a process that runs in the on-premises or cloud side of the gateway. It has network visibility to the Bluemix application and to the destinations. Multiple destinations might need multiple clients. The gateway client initiates a connection with the gateway in Bluemix and forwards requests from the gateway to the destinations.

When you connect your gateway to a client using one of the above listed clients, you can add a destination to your gateway. Figure 2-3 shows the window to create a destination. The destination specifies how to connect to the system resource that is a SOR, database of record, and so on. You need only one destination per system resource. The destination consists of a name, host name or IP address, the port, and protocol.

Note: If you are using IBM DataWorks, you do not need to create a destination.

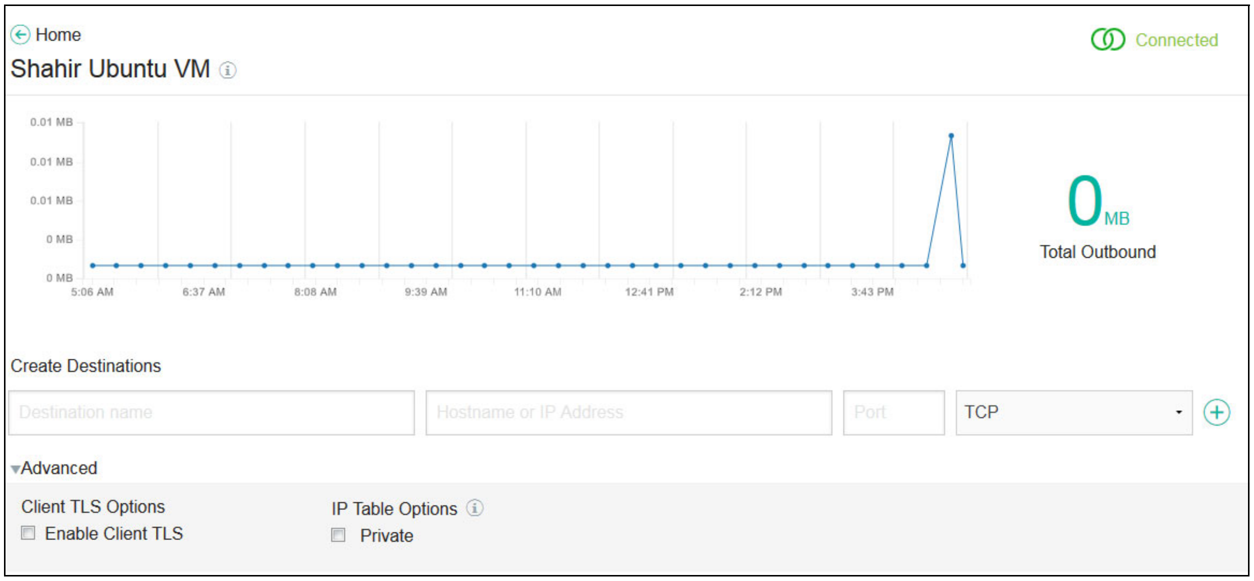


Figure 2-3 Create Destination window

Each destination can optionally use Transport Layer Security (TLS) protocol:

- Application-side TLS

  Secures access between the Bluemix app and the cloud environment client.

- Client-side TLS

  Secures access between the cloud environment client and the destination.

- The two can be set independently

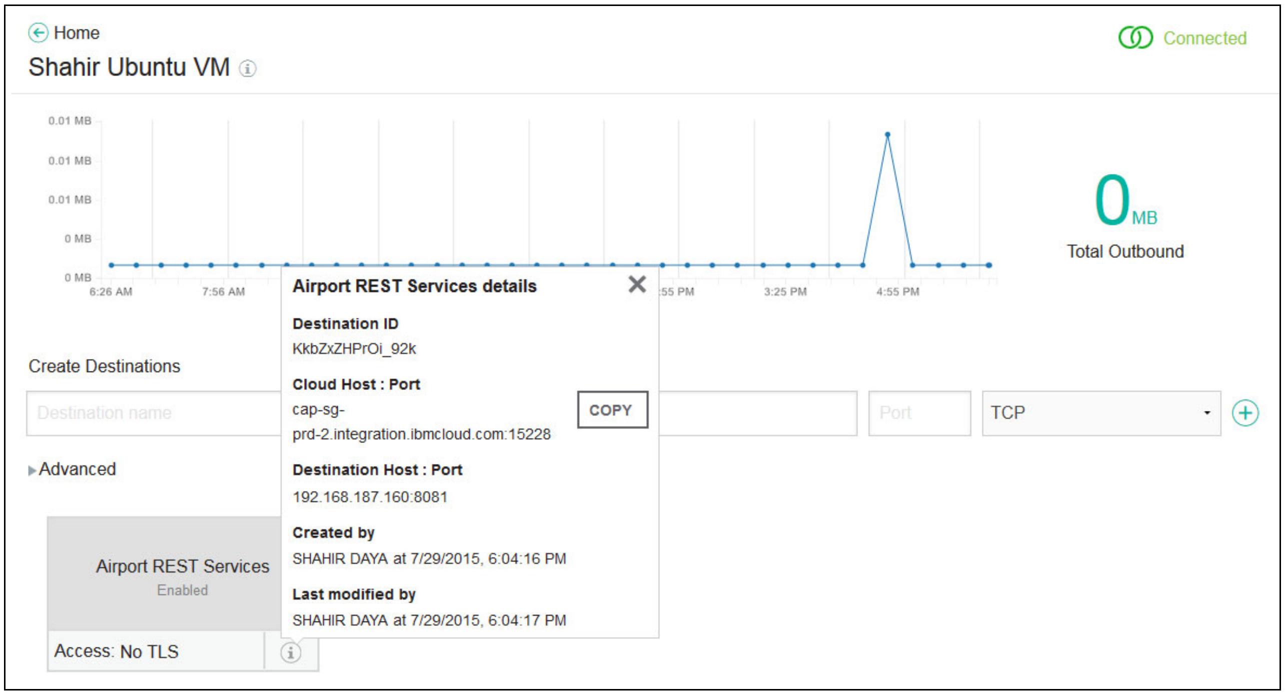
For application-side security, the following protocol options apply:

- TCP
  - No TLS: No certificates, no encryption
  - No authentication is provided
  - Bluemix application communicates directly to the gateway
- TLS Server Side
  - TLS is enabled
  - Secure gateway generates a certificate to prove its authority
  - You need to accept the server certificate into your Bluemix application truststore
- TLS Mutual Auth
  - Option 1: Auto-generate certificate and private key
  - Option 2: Upload existing keys to the gateway
- HTTP
- HTTPS

For client-side security, you can enable client TLS. Client TLS is required for connecting to an HTTPS backend. The destination's certificate is verified against known certificate authorities. If the certificate (PEM) is self-signed, it must be attached to the cloud environment client. Attaching the certificate can be done during the creation and edit of the destination or via the Secure Gateway REST API.

Additionally, the IP Table Options can be used to limit the IP addresses or ports that can access the destination. The IP or port number entered into the IP table must be the external IP address that the Secure Gateway server sees, not the local IP address of the machine.

Figure 2-4 shows the details of a destination. The Cloud Host:Port is used by the application to connect securely to the destination.



Airport REST Services details

Destination ID
KkbZxZHPrOi_92k

Cloud Host : Port
cap-sg-prd-2.integration.ibmcloud.com:15228

Destination Host : Port
192.168.187.160:8081

Created by
SHAHIR DAYA at 7/29/2015, 6:04:16 PM

Last modified by
SHAHIR DAYA at 7/29/2015, 6:04:17 PM

Figure 2-4 Destination details contain cloud host and port

Figure 2-5 shows a typical simple use case. An SOE web application developed in Java Platform, Enterprise Edition runs in a Java Liberty runtime in Bluemix. The web application needs to access data that is stored in a MySQL database that is hosted in a corporate data center. In this case also, the Secure Gateway service is used to securely connect from the Public Bluemix to the corporate data center to enable making Java Database Connectivity (JDBC) calls over a secure channel.

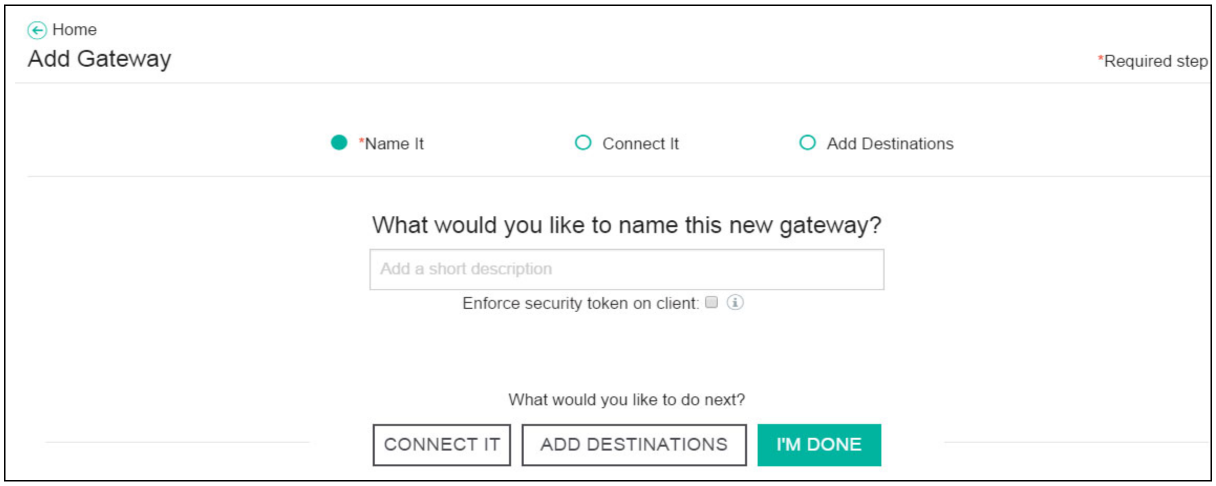


Figure 2-5 Bluemix SOE web application requiring integration with an on-premises relational database

Figure 2-7 on page 15 shows a more complex use case for Secure Gateway. An SOE mobile application uses Bluemix as the mobile backend as a service (MBaaS). An IBM Cloudant® NoSQL DB service provides data storage for the mobile application. A NodeJS runtime provides the necessary orchestration of service calls to fulfill the needs of user-interface interactions. The mobile application needs to call a SOAP-based web service that resides in the corporate data center. The Secure Gateway service is used to securely connect from the Public Bluemix to the corporate data center. SOAP requests go through the Secure Gateway over the secured channel.

In summary, the Secure Gateway service brings hybrid integration capability to your Bluemix environment. It provides secure connectivity from Bluemix to other applications and data sources running on-premises or in other clouds.

2.1.1 Two typical examples of using Secure Gateway

You now see two typical examples of using Secure Gateway. Figure 2-6 on page 14 shows a simple use case.

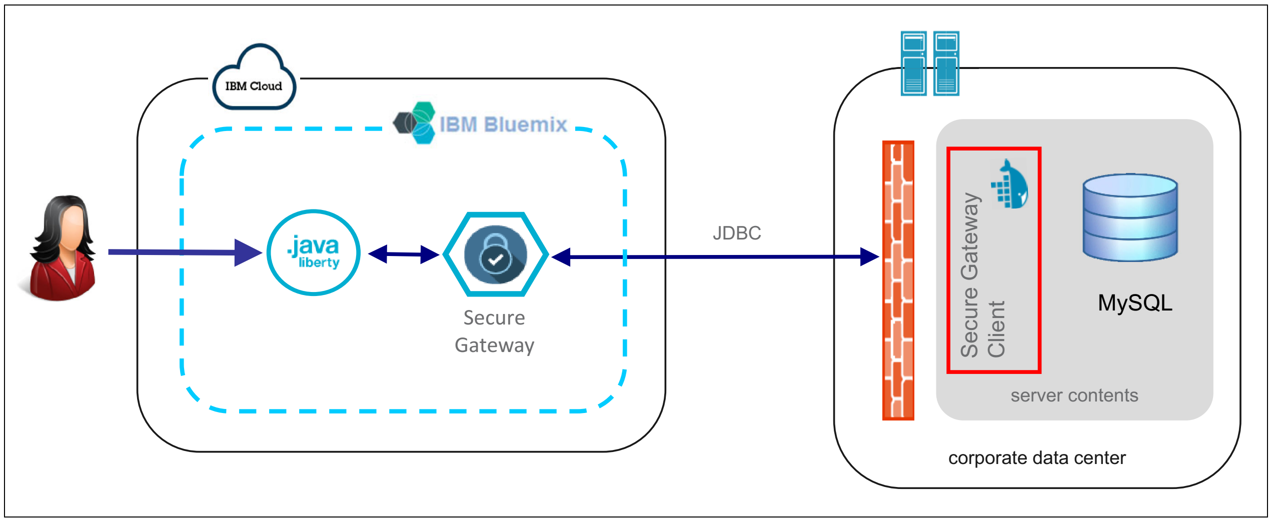


Figure 2-6   A typical simple use case for Secure Gateway

A System of Engagement web application developed in Java Enterprise Edition runs in a Java Liberty runtime in Bluemix. The web application needs to access data that is stored in a MySQL database that is hosted in a corporate data center. In this case also, the Secure Gateway service is used to securely connect from the Public Bluemix to the corporate data center to enable making JDBC calls over a secure channel.

Figure 2-7 shows a more complicated scenario.

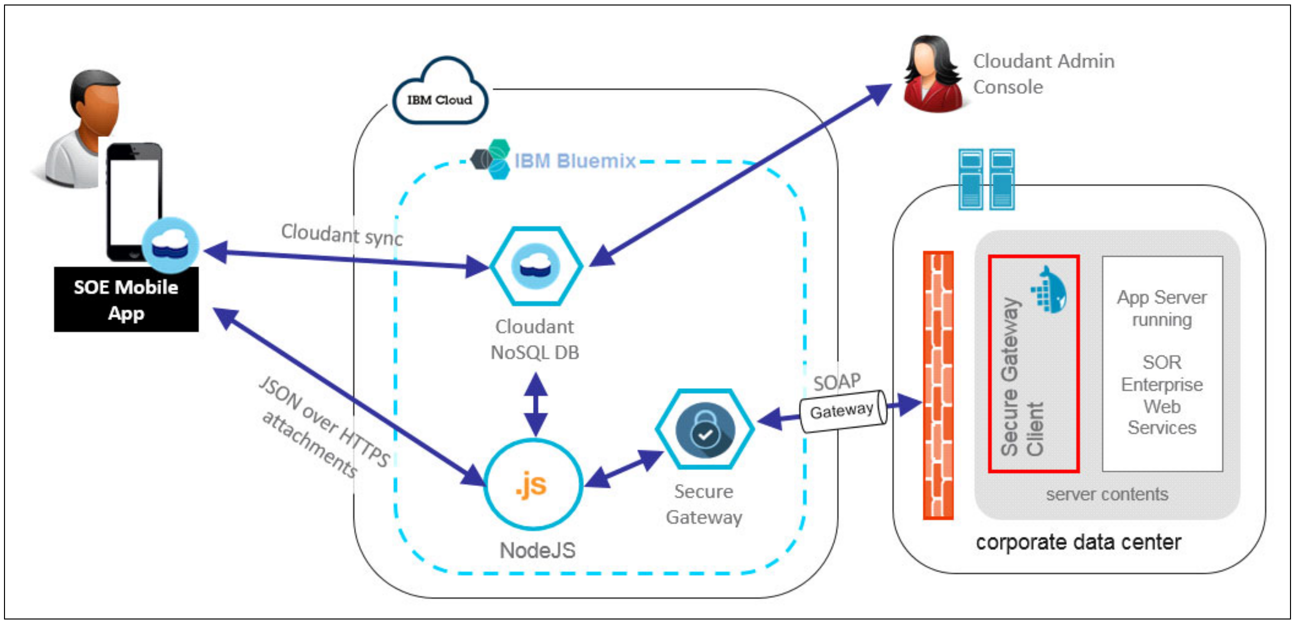


Figure 2-7 A more complex use case for Secure Gateway

A System of Engagement mobile application uses Bluemix as the MBaaS. A Cloudant NoSQL DB service provides data storage for the mobile application. A NodeJS runtime provides the necessary orchestration of service calls to fulfill the needs of user interface interactions. The mobile application needs to call a SOAP-based web service that resides in the corporate data center. The Secure Gateway service is used to securely connect from the Public Bluemix to the corporate data center. SOAP requests go through the Secure Gateway over the secured channel.

2.1.2 Commonly asked questions about Secure Gateway

Below, we list some commonly asked questions about Secure Gateway:

Q1. What protocol does Secure Gateway support? Is it similar to VPN?

Answer: Secure Gateway tunnel is similar to a secure websocket tunnel that is similar to Secure Shell (SSH). It is not similar to VPN. IPSec is not currently supported. There is IBM VPN service on Bluemix. See this website:

https://www.ng.bluemix.net/docs/services/vpn/index.html

Q2. Is Secure Gateway bidirectional?

Answer: The Secure Gateway is unidirectional in the sense that all requests have to be initiated by the Bluemix app.

Q3. Can the addressing in the destination be done by using the host name?

Answer: The addressing of the destination can be achieved by using either IP or host name. However, there is no DNS resolution in the Secure Gateway (SG) when using the host name. The host that the gateway client is running on needs to be able to resolve the host name of the endpoint. The server side does not need to know anything about the DNS.

Q4. Which ports do the Secure Gateway client use?

Answer: Secure Gateway uses Port 443 and 9000 for outbound calls only so that the client can establish a connection to the server.

Q5. What ports should be opened in your firewall?

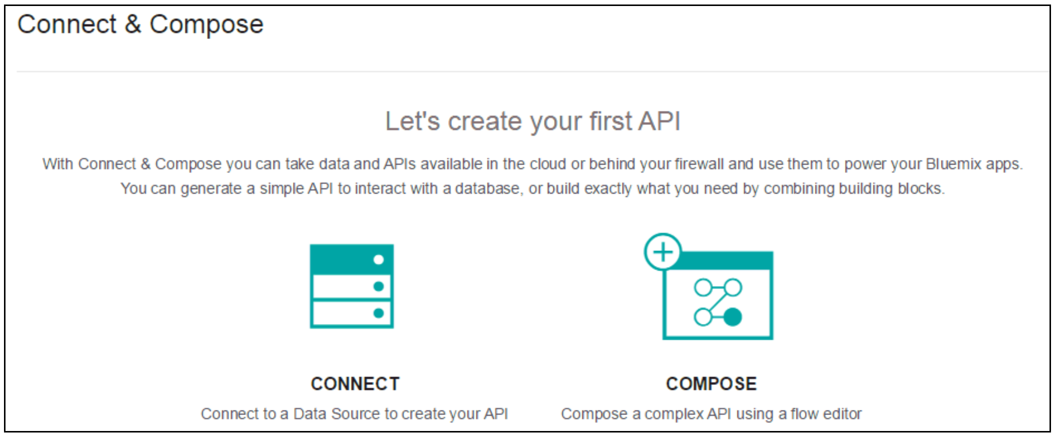
Answer: Port 443 and port 9000 should be opened in the firewall.

Secure Gateway uses port 443 and 9000 for outbound calls only so that the client can establish a connection to the server. It is the Secure Gateway servers that the gateway client is connecting out to. The first call goes to the DataPower proxy load-balancer so that needs to be allowed as well on port 443. The second call from the client goes directly to one of our server nodes on port 9000.

2.2 Connect & Compose

Connect & Compose is a new service offering in Bluemix for API creation. As shown in Figure 2-8 on page 16, the service offers to create API in two ways:

- Connect to and create a Representational State Transfer API to interact with a single source of data or service.
- Compose by using a flow editor and API that performs complex functionalities in each API call, such as relay further requests to backend services, sending email, interacting with persistence, and various third-party services.


Connect & Compose

Let's create your first API

With Connect & Compose you can take data and APIs available in the cloud or behind your firewall and use them to power your Bluemix apps. You can generate a simple API to interact with a database, or build exactly what you need by combining building blocks.

CONNECT
Connect to a Data Source to create your API
COMPOSE
Compose a complex API using a flow editor

Figure 2-8 Connect & Compose landing

Connect option

The Connect option in Connect & Compose provides a RESTful interface to interact with a growing option of data sources and services in Bluemix, on-premises, and on cloud. Figure 2-9 shows supported sources as of this publication.

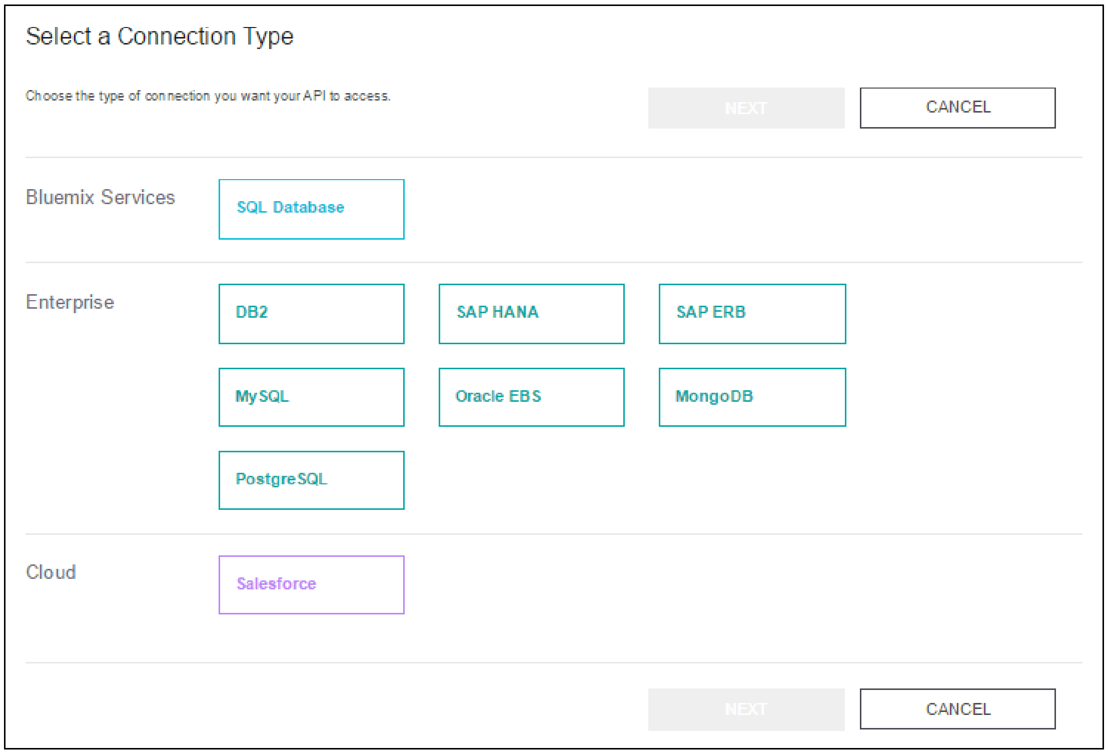


Figure 2-9 Connector options at different locations

To start, the user selects where and the type of source to connect to. For databases on Bluemix, it is required that the service be provisioned and bound to your app before Connect & Compose can connect to it. Similarly, Secure Gateway service is required to access enterprise sources behind a corporate firewall or in a secured network.

With successful connection to a data source, the user then selects the model to which the API acts on. Depending on the type of source that the user is connecting to, Connect & Compose allows users to exclude optional parameters and REST endpoints during the API creation process. This is a valuable feature when the user only wants to externalize non-destructive endpoints. A sample swagger is shown for the user to check the API formation before save. With the host and port provided by Connect & Compose after creation, the user can now interact with the API via REST calls.

Compose option

The more robust option to create an API with complex functionality is with the Compose option. API composition is delegated to a Node-RED composition interface. The interface allows user to compose an API by dragging and connecting wanted nodes. Figure 2-10 is a sample flow of how an API can be composed to perform multiple functions within one call (extra payload handling function nodes are required to wrap services nodes in most cases shown, but are eliminated to provide a generic concept). All four APIs initiate in an HTTP request, and ends with an HTTP response:

- First, the API endpoint accepts a POST request with location information, retrieves Google Places data, checks the weather, and pushes the result to email, twitter, and pinterest.
- Second, the API endpoint accepts a GET request. The function node can provide adjustments, such as putting query parameters into payload, or adjusting and reformatting input values. The adjustment is then relayed to the SAP node to invoke a function that has been configured. The returned data is then uploaded to Dropbox, and submitted for debugging at the same time.
- Third, the API endpoint accepts a GET request and passes the information to a bus schedule service, and adds the information to Google Calendar.
- Fourth, the API endpoint demonstrates a chain of HTTP requests, which ends with persisting the final result into an SAP Hana database.

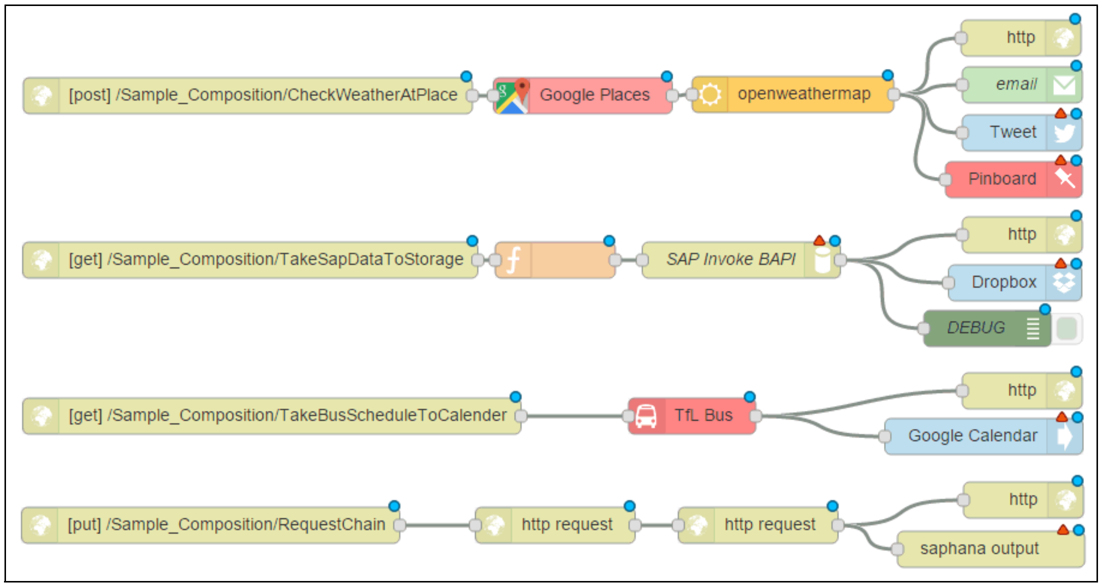
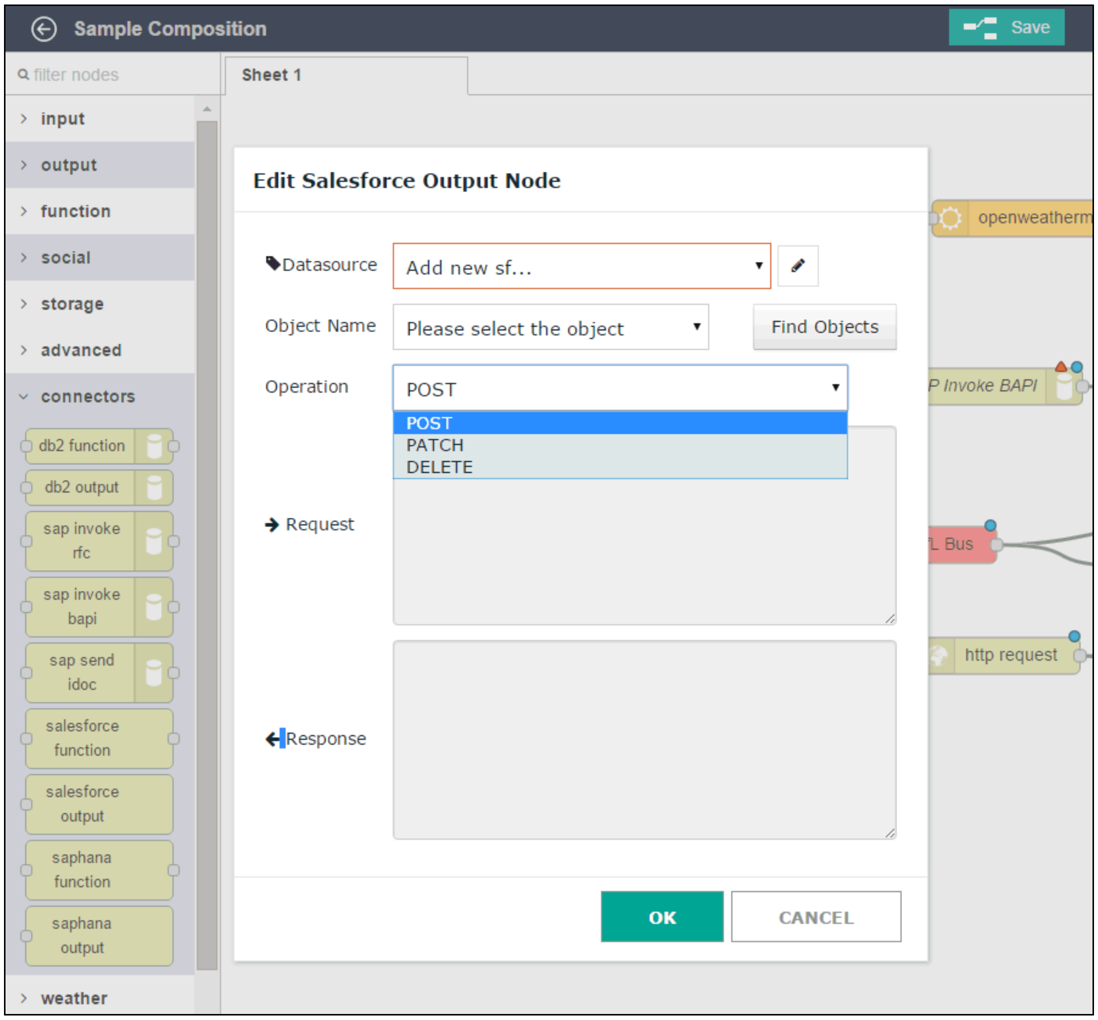

[post] /Sample_Composition/CheckWeatherAtPlace
Google Places
openweathermap
http
email
Tweet
Pinboard
[get] /Sample_Composition/TakeSapDataToStorage
SAP Invoke BAPI
http
Dropbox
DEBUG
[get] /Sample_Composition/TakeBusScheduleToCalender
TfL Bus
http
Google Calendar
[put] /Sample_Composition/RequestChain
http request
http request
http
saphana output

Figure 2-10 Sample Composition with various types of nodes

While there is a node to support it, there is no limitation on what sources to act on in the composition flow editor, which distinguishes it from the Connect counterpart. Figure 2-11 shows a sample node called *salesforce output*. This node provides the functionality to interact with data stored on the Salesforce database. Notice also in the left panel shown in Figure 2-11, Salesforce-related nodes come in two options. The *salesforce output* node performs POST, PATCH, and DELETE operations, and provides no output but does end the flow. The *salesforce function* node alternatively, performs GET operations, and provides output that can be passed on to the next node in the flow diagram.



Edit Salesforce Output Node

Datasource: Add new sf...

Object Name: Please select the object — Find Objects

Operation: POST (POST, PATCH, DELETE)

Request

Response

Figure 2-11 Configuration for salesforce output node

Table 2-2 shows the categories and their respective nodes available as of the time of this publication.

Table 2-2 Category of nodes

Category	Nodes
input	inject, catch, http, websocket
output	debug, http response, websocket
function	function, template, delay, trigger, comment, http request, switch, change, range, csv, html, json, xml, rbe

Category	Nodes
social	email, twitter, delicious, foursquare, swarm, google plus, google places, google calender, instagram, pinboard
storage	amazon s3, box, dropbox
advanced	feedparse
connectors	db2, sap, salesforce, saphana
weather	forecastio, openweathermap, wunderground
location	google geocoding, google directions
Google	google calender
transport	tfl underground, tfl bus

Using APIs

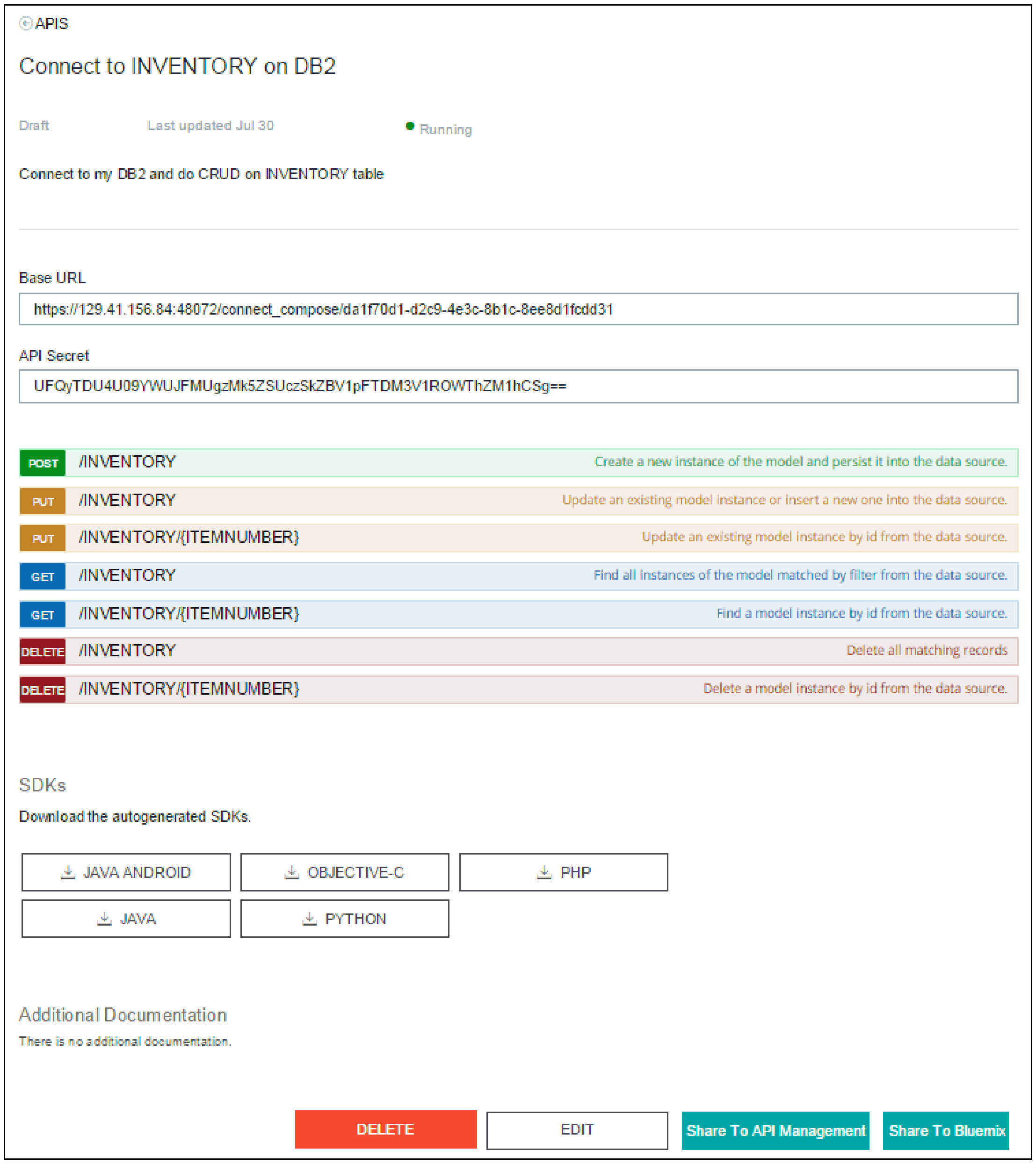
When an API is created in Connect & Compose, the API becomes immediately available on the host and port provided. Figure 2-12 on page 21 shows detailed information about the new API.

The top section of the page displays the general information about the API, including shared name, state, time stamp, and API running status. The running status is only an indicator of whether the API is available. The user is responsible to keep all connections from the API to the data source (such as Secure Gateway, virtual private network, and the data source itself) running.

The second section of the API details page shows the base Uniform Resource Locator for the API, and the access key to include in the header when sending requests to the API.

The third section includes an interactive Swagger UI that allows the user to visualize and test the API.

The fourth section provides SDK package downloads for ease of development in different languages. If the user uploaded more documentation for this API during the creation process, they are also displayed for download.


APIS

Connect to INVENTORY on DB2

Draft Last updated Jul 30 ● Running

Connect to my DB2 and do CRUD on INVENTORY table

Base URL

https://129.41.156.84:48072/connect_compose/da1f70d1-d2c9-4e3c-8b1c-8ee8d1fcdd31

API Secret

UFQyTDU4U09YWUJFMUgzMk5ZSUczSkZBV1pFTDM3V1ROWThZM1hCSg==

POST	/INVENTORY	Create a new instance of the model and persist it into the data source.
PUT	/INVENTORY	Update an existing model instance or insert a new one into the data source.
PUT	/INVENTORY/{ITEMNUMBER}	Update an existing model instance by id from the data source.
GET	/INVENTORY	Find all instances of the model matched by filter from the data source.
GET	/INVENTORY/{ITEMNUMBER}	Find a model instance by id from the data source.
DELETE	/INVENTORY	Delete all matching records
DELETE	/INVENTORY/{ITEMNUMBER}	Delete a model instance by id from the data source.

SDKs

Download the autogenerated SDKs.

Additional Documentation

There is no additional documentation.

Figure 2-12 Sample API created from connecting to IBM DB2®

For advanced utilization of the API, the user has the option to share to Bluemix, share to API Management, or both. When an API created in Connect & Compose is shared to Bluemix, the user obtains the ability to bind the API to an app in the same space. The API detail is also available for all users in the Bluemix Space to see. Alternatively, sharing the created API to API Management allows the API to be managed with more policies and all the functionalities that API Management has to offer.

2.3 IBM MQ Light

IBM MQ Light for Bluemix is a messaging service built on the Advanced Message Queuing Protocol (AMQP), available as a cloud-based service in the Bluemix platform. IBM MQ Light service provides high flexibility for applications built on the cloud, especially to the applications leveraging microservice architecture. When building applications using the microservice approach, it is fundamental to have an effective communication protocol among services that can help to ensure the core principles of a microservices architecture, which are decentralizing, independency that in turn needs a reliable, scalable, and flexible way of communicating where IBM MQ Light for Bluemix service can help.

IBM MQ Light provides support for multiple run times and various APIs that give developers the freedom to select the technologies that they are the most comfortable with. That said, IBM MQ Light can be used to enable applications that use diversified run times to communicate with each other.

When using IBM MQ Light service for building applications, the following considerations should be taken into account.

Supported runtimes

Applications developed by using the following languages: Java, JavaScript (Node.js), Python, Ruby, Perl, PHP, and C are so far natively supported by IBM MQ Light. In case you have applications developed using other languages than the ones in the list, you can still make them work with IBM MQ Light by creating a client wrapper using an AMQP messaging toolkit like Qpid Proton in the following link:

http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html

Security

Security can be viewed from two aspects: Connection to the service and the messages being sent and received.

The connection to IBM MQ Light service is enabled by default for all applications that are bound to the services and is provided by using TLS version 1.2 (with FIPS 140-2 compliant Cipher Specs). If your applications were built before the availability of the default security-enabled connection of IBM MQ Light service, the connection will continue to be decrypted until you unbound and rebound the application to the service. Python and Ruby applications are not supported with the encrypted connection to IBM MQ Light service yet.

The messages sent to the IBM MQ Light service might be persisted while being processed and the data is not encrypted so it is highly recommend doing the encryption against the data before sending it to IBM MQ Light, especially with sensitive data.

Connectivity

IBM MQ Light services provisioned in the public Bluemix environment can be connected from the applications that run either outside of Bluemix or in a Bluemix dedicated environment. This unleashes the integration capabilities of IBM MQ Light service in Bluemix to be presented anywhere in your architecture.

Messages

Four GB is the maximum storage space for each IBM MQ Light service in Bluemix to store messages data. Each message has a 256 KB maximum data allowed to be attached with, including the overhead header being added by the IBM MQ Light API. It is necessary to make sure the limitation is taken care of if your application has to process messages that reach that threshold.

Only a maximum of 10,000 messages can be persisted on a destination and messages older than 30 days are not kept.

Monitoring

IBM MQ Light services come with a web user interface, which allows the application owner to monitor the client health and the received and sent messages through IBM MQ Light service per instance.

Note: IBM MQ Light is also available as a separate product, as opposed to a Bluemix service. See 3.2, "Introduction to IBM MQ Light" on page 38 for information about IBM MQ Light as a product.

2.4 Message Hub

IBM Message Hub on Bluemix is a cloud-based scalable and high throughput message bus allowing the capability of uniting on-premises and off-premises cloud technologies. Diverse services can be wired together through Message Hub service by leveraging open wired protocols, which allows you to pick up a wide range of languages and technologies beyond the ones that IBM can initially support on day one.

Message Hub is built on top of Apache Kafka messaging engine, hence it inherits the community-proven scalability and performance, as well as the durable real-time messaging capability from the engine. Because Kafka plays a core role of Message Hub, look at its general architecture and concepts. Figure 2-13 illustrates the architectural overview of Apache Kafka.

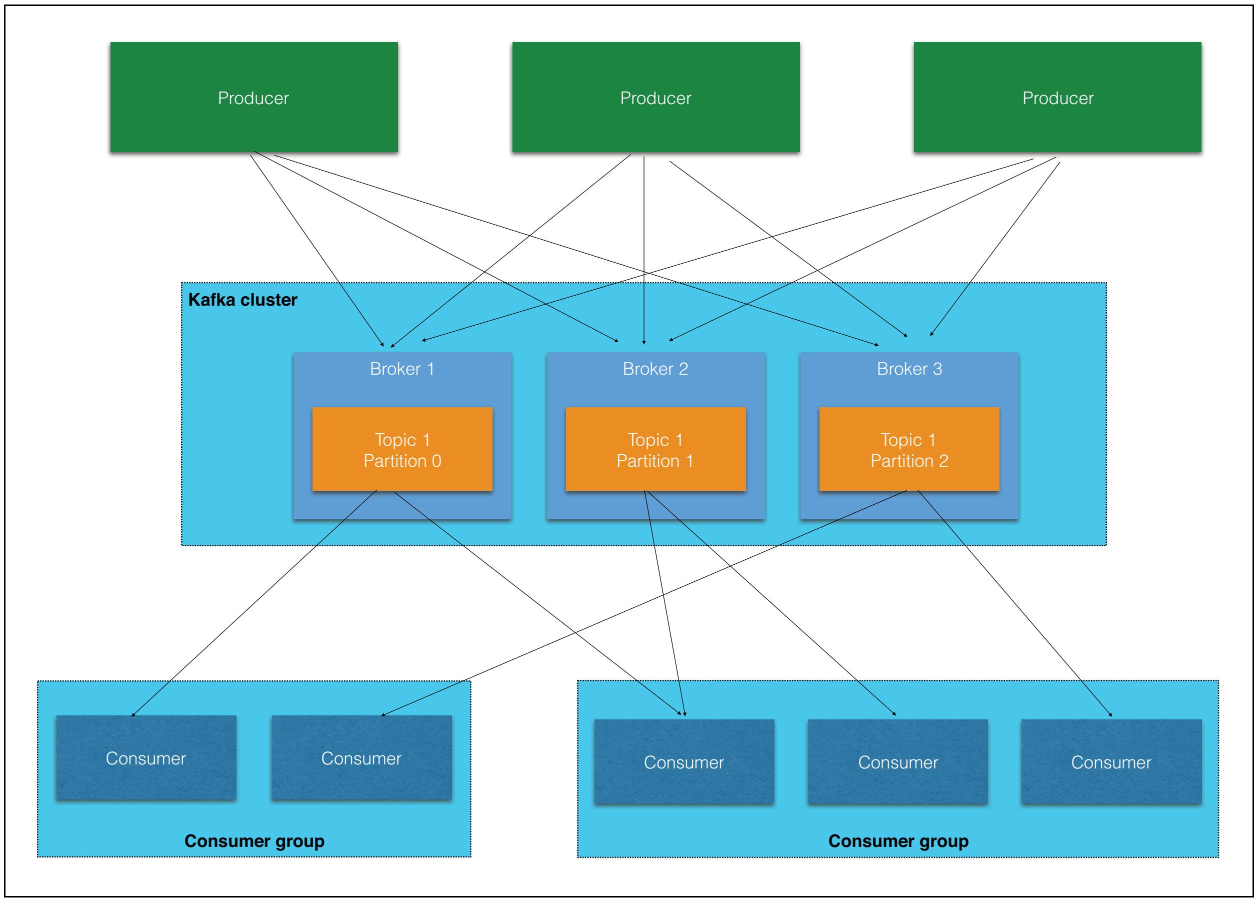


Figure 2-13 Kafka architecture

In Kafka, a topic is a category or feed name to which messages are published. Kafka is run as a cluster of one or more nodes, each of which is called a *broker*. Producers are processes that publish messages to a Kafka topic. Processes that subscribe to topics and process the feed of published messages are called *consumer*. Consumers contain in their name a consumer group name, a consumer abstraction that generalizes both queuing and publish/subscribe messaging models of consumers.

Each message published to a topic is delivered to one consumer instance within each subscribing consumer group.

Following are some highlight features initially provided by IBM Message Hub service.

Availability

IBM Message Hub architecturally inherits Kafka, which is deployed as a clustered set of message brokers. The cluster can be configured so that its brokers are not in a same failure domain when the failure occurs, which necessarily means that a failure would only affect one broker in the set. For that reason, the cluster can tolerate failure of a particular broker and continue to process messages without having to wait for the failed broker to be recovered.

The clustered model reduces the downtime to a minimum and helps Message Hub to provide a high availability to serve.

Also, messages are being replicated between the brokers, as well as effectively persisted on persistent storage for at least 24 hours, using an enhanced operating system cache mechanism, so from a message-level perspective, Message Hub reduces the possibility of a message being lost.

Scalability and performance

The clustered model also improves the throughput of messages. By design, Message Hub lets the message brokers concentrate on effectively persisting messages on storage, hence strips out other features that would potentially cause performance issues.

Security

Message Hub advanced the security features provided in Kafka by enforcing a higher level of security compliance requirement, such as:

- Connections to Kafka native and REST interface must be made by using TLS 1.2.
- Connections are restricted to a set of strong cipher suites:
  - ECDHE-RSA-AES128-GCM-SHA256
  - ECDHE-RSA-AES256-GCM-SHA384
  - DHE-RSA-AES128-GCM-SHA256
  - kEDH+AESGCM
  - ECDHE-RSA-AES128-SHA256
  - ECDHE-RSA-AES256-SHA384
  - DHE-RSA-AES128-SHA256
  - DHE-RSA-AES256-SHA256

Integration capability

Integration is one of the core capabilities of Message Hub. It provides multiple interfaces through which messages can be produced and consumed. Both a secure native Kafka interface using standard Kafka clients and a REST API are supported. It can integrate well with various other services on Bluemix.

Message Hub also leverages Apache Spark and IBM InfoSphere® streams alongside Kafka, which is suitable to build a high performance scalable streaming analytics solution.

2.5 Message Connect

IBM Message Connect is a cloud-based service in the Bluemix platform that provides a centralized, programming language independent, publish and subscribe system for messaging and other event-based data.

Note: IBM Message Connect service was formally known as the *Event Hub* service. You can see the announcement for Message Connect at the following link:

https://developer.ibm.com/bluemix/2016/02/15/new-message-connect-service/

Apart from the renaming, Message Connect also includes some minor enhancements.

Events travel along streams, which act as high-performance, high-volume pathways for data. You can configure Bluemix applications as subscribers to one or more streams to provide processing for the data or to trigger actions based on incoming messages.

Message Connect provides features that are similar to other publish and subscribe systems. At one end, one or more publishers write data onto a stream, and at the other end, one or more consumers receive a push notification of the event, along with the data packets that were produced by the publisher.

Events can be generated from various connector sources. Connectors are prebuilt to connect to third-party data sources and to flow specific data points onto a stream. For example, a connector can provide connectivity to Twitter's streaming search API. Tweets are channeled from Twitter to your stream without having to make multiple connections to Twitter itself, which Twitter limits. As a Message Connect user, all you need to do is enter Twitter authentication credentials and search criterion. Message Connect takes care of the rest, and you do not have to write a single line of code.

Message Connect provides a REST API for each stream so that applications can consume data from a stream easily by subscribing to an event and getting data in real time with minimal overhead.

The Message Connect service provides various connectors to connect to third-party data sources. At the time of writing this book, the following connectors are supported:

- Twitter connector
- Force.com connector
- IBM MQ Light connector
- Cloudant connectors

Important: Currently, Message Connect service is *experimental* in Bluemix and subject to change.

2.6 Message Hub and IBM MQ Light integration through Message Connect

The Message Hub uses the IBM MQ Light connector that is available with the Message Connect Bluemix service to connect to the IBM MQ Light service or on-premises IBM MQ Light product. This function allows applications that are built using the Apache Kafka API running on an IBM or third-party cloud to exchange messages with IBM MQ Light.

2.6.1 Configuring Message Hub and Message Connect for IBM MQ Light events

In this section, we describe how to configure Message Connect and Message Hub to consume events from an IBM MQ Light application running on-premises.

Creating Message Connect and Message Hub services

The Message Connect service requires Message Hub service to be available in your Bluemix space. You can create one Message Hub and one Message Connect service per Bluemix space. Therefore, before you begin, ensure that you have added the Message Hub service.

To add the Message Hub service, select the Message Hub tile from the Bluemix Catalog as shown in Figure 2-14.

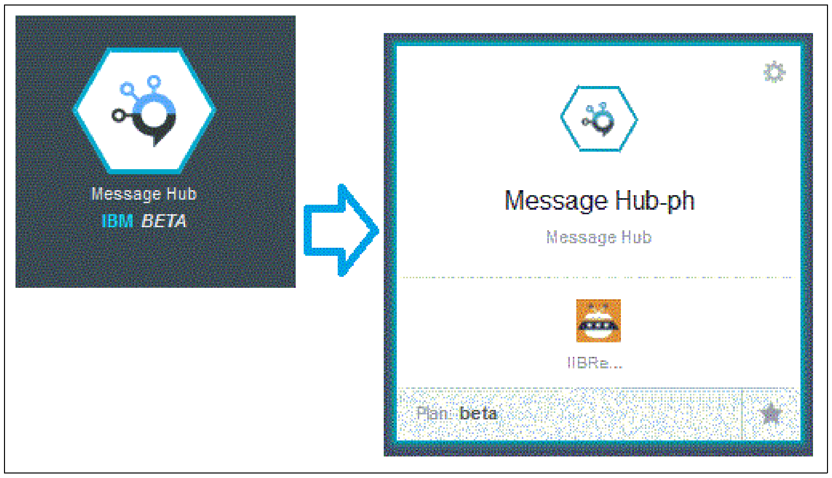


Figure 2-14 Message Hub service

To add the Message Connect service to your Bluemix org, follow these steps:

1. From the Bluemix Catalog, scroll to the end of the page and click **Bluemix Labs Catalog**.
2. Refine the search by selecting the **Integration** category.
3. Click the **Message Connect** tile. See Figure 2-15.

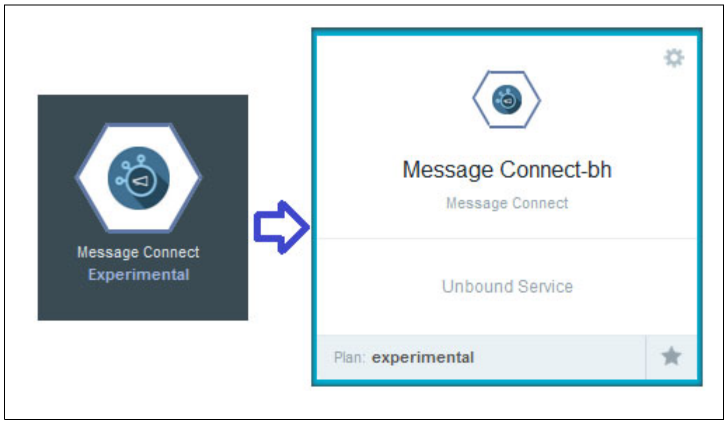
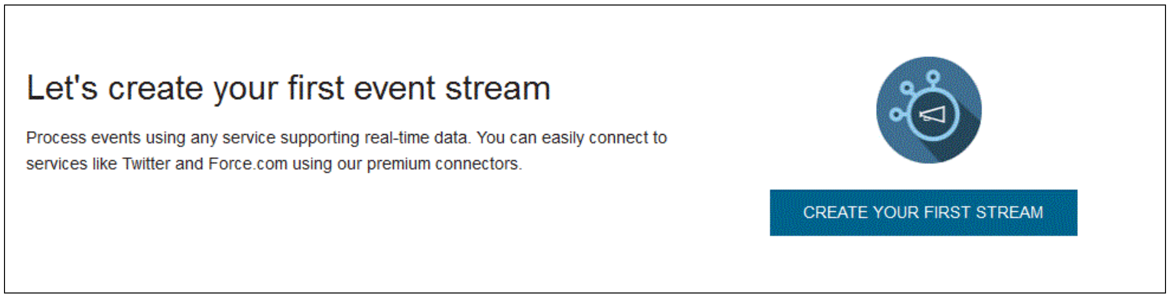


Figure 2-15 Message Connect service

   The Message Connect service details page is displayed and the plan, features, and price are listed in the “Pick a plan” section.

4. In the Add Service section, select an option for each of the following fields:
   – Space: Select the space that you want the Message Connect service to belong to. You can create different Message Connect services for each one of your Bluemix spaces.
   – Service name: This field is prepopulated. However, it can be modified or changed completely.
   – Selected Plan: Currently, this only includes the experimental plan.

5. Click **CREATE.** A new window is displayed.
6. To begin creating your first event stream, click **CREATE YOUR FIRST STREAM**, as shown in Figure 2-16.


Let's create your first event stream

Process events using any service supporting real-time data. You can easily connect to services like Twitter and Force.com using our premium connectors.

Figure 2-16 Create First Stream in Message Connect

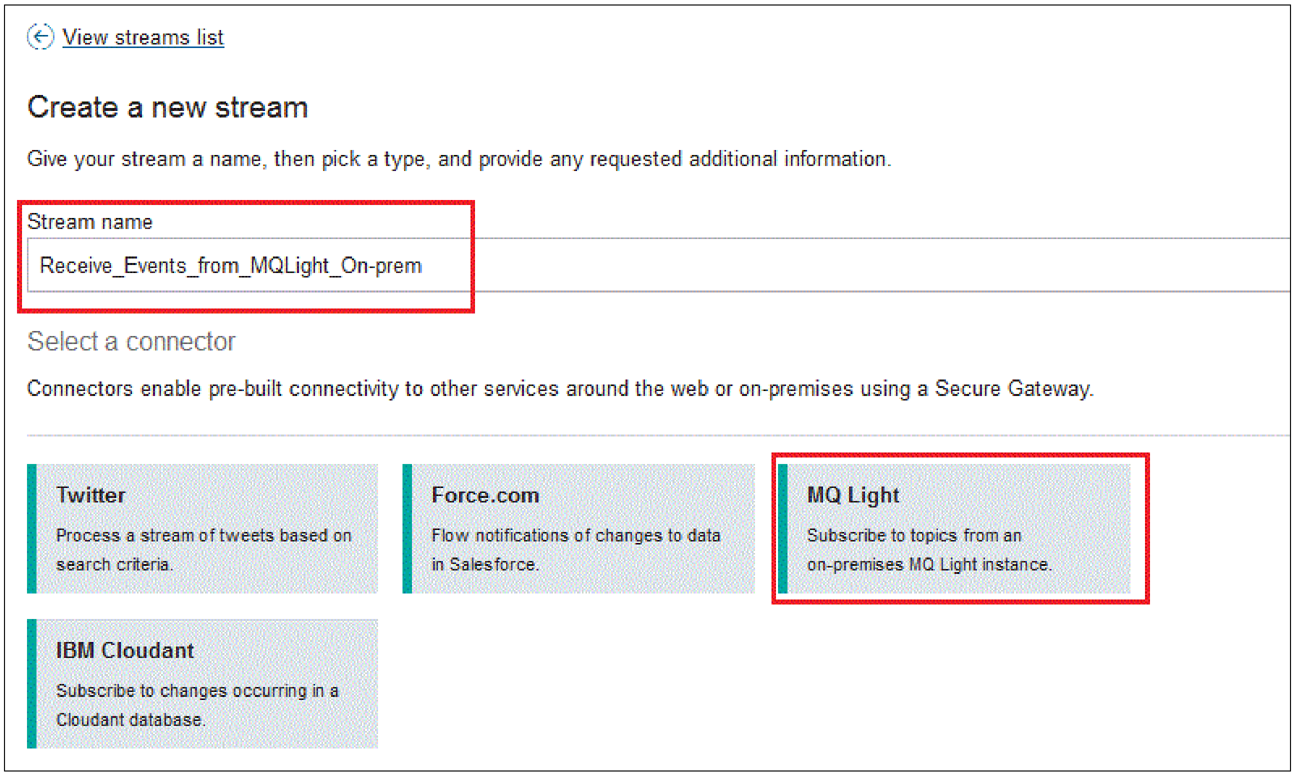
The following section describes how to create a stream to consume events from the IBM MQ Light connector.

Consuming events by connecting to a stream from an application

To enable data to flow from your chosen source to an application, you need to create a stream. Consuming events from a stream follows the same approach regardless of the connector type. Currently, events are consumed by using a REST API. To create stream and consume events, complete the following steps.

In the Message Connect dashboard:

1. Click **CREATE YOUR FIRST STREAM** as shown in Figure 2-16.
2. As shown in Figure 2-17 on page 29, in the Stream name field, provide a name for your stream. Select a connector. In this example, we select MQ Light connector.


View streams list

Create a new stream

Give your stream a name, then pick a type, and provide any requested additional information.

Stream name

Receive_Events_from_MQLight_On-prem

Select a connector

Connectors enable pre-built connectivity to other services around the web or on-premises using a Secure Gateway.

Twitter
Process a stream of tweets based on search criteria.

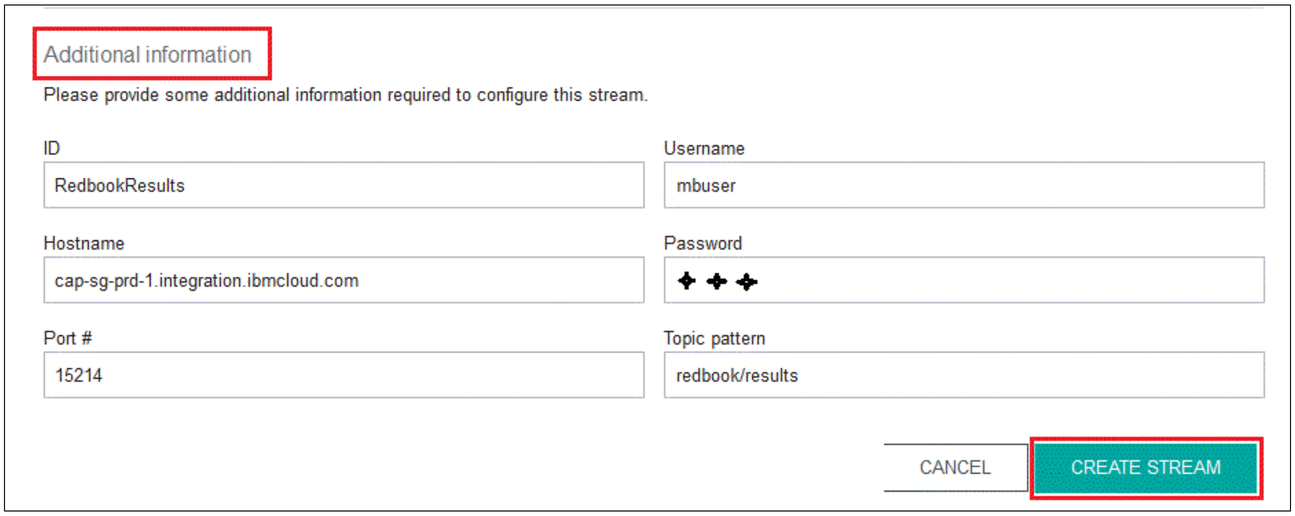
Force.com
Flow notifications of changes to data in Salesforce.

MQ Light
Subscribe to topics from an on-premises MQ Light instance.

IBM Cloudant
Subscribe to changes occurring in a Cloudant database.

Figure 2-17 Name the stream and select connector

3. As shown in Figure 2-18, provide the requested details in the Additional information section and click **CREATE STREAM**. In this example, we are showing how to connect to IBM MQ Light service running on-premises. So, we first created a Secure Gateway to create a secure connection from Bluemix to an on-premises system where IBM MQ Light service is running. This is explained in Chapter 6. The Hostname and Port number in the IBM MQ Light connector information is the cloud host and port number from the destination that is created for IBM MQ Light on-premises connectivity in Secure Gateway. The “Topic pattern” is the topic to which your on-premises IBM MQ Light application publishes the data to and that you want to consume in your Bluemix applications.

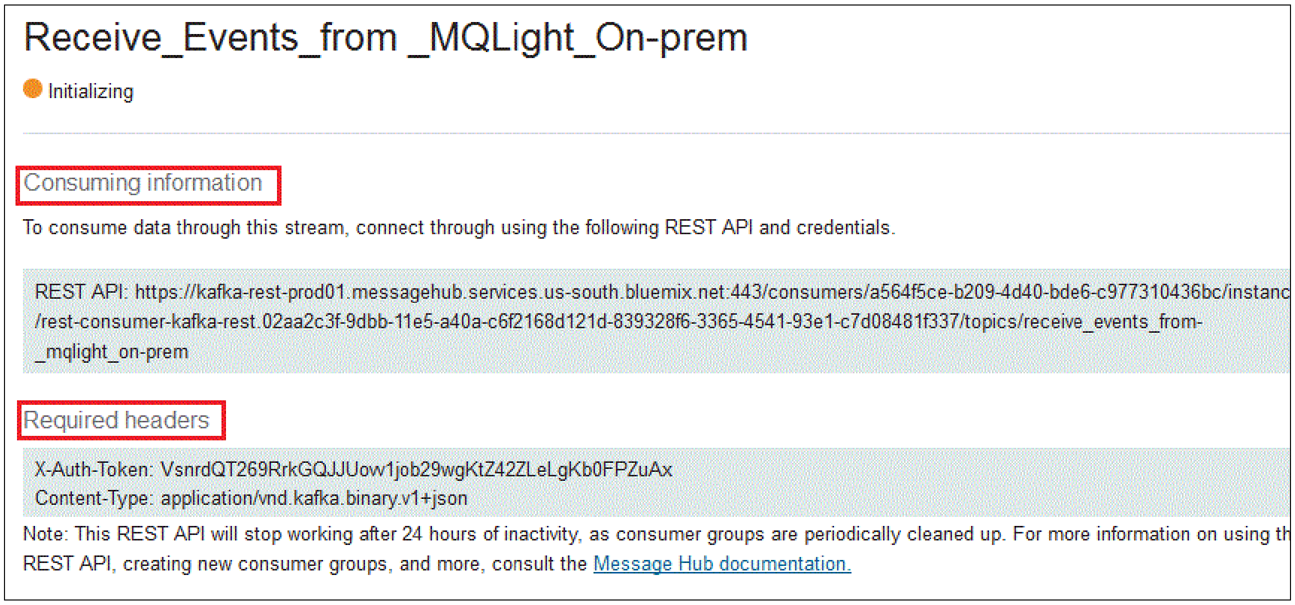

Additional information

Please provide some additional information required to configure this stream.

ID: RedbookResults	Username: mbuser
Hostname: cap-sg-prd-1.integration.ibmcloud.com	Password: ✤ ✤ ✤
Port #: 15214	Topic pattern: redbook/results

Figure 2-18 Add connection details for IBM MQ Light on-premises service

4. The stream is now ready to be used after initialization. As shown in Figure 2-19, the page is displayed with "Consuming information," which is Kafka REST API and the required headers to be used when making the REST call.



Receive_Events_from _MQLight_On-prem

Initializing

Consuming information

To consume data through this stream, connect through using the following REST API and credentials.

REST API: https://kafka-rest-prod01.messagehub.services.us-south.bluemix.net:443/consumers/a564f5ce-b209-4d40-bde6-c977310436bc/instanc /rest-consumer-kafka-rest.02aa2c3f-9dbb-11e5-a40a-c6f2168d121d-839328f6-3365-4541-93e1-c7d08481f337/topics/receive_events_from-_mqlight_on-prem

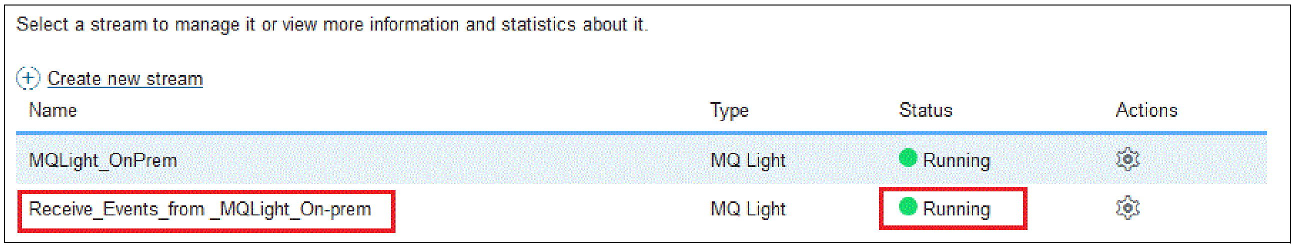
Required headers

X-Auth-Token: VsnrdQT269RrkGQJJUow1job29wgKtZ42ZLeLgKb0FPZuAx
Content-Type: application/vnd.kafka.binary.v1+json

Note: This REST API will stop working after 24 hours of inactivity, as consumer groups are periodically cleaned up. For more information on using th REST API, creating new consumer groups, and more, consult the Message Hub documentation.

Figure 2-19 REST API consuming information for the stream

5. The Streams details page is displayed where you can monitor the status of the stream to see when it enters the running state, as seen in Figure 2-20. You can choose to stop and restart a stream at any time.



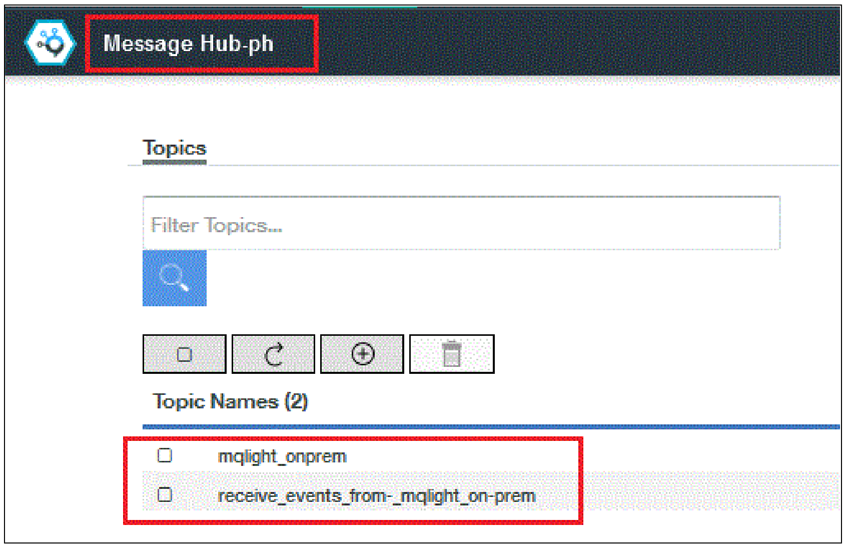
Select a stream to manage it or view more information and statistics about it.

Create new stream

Name	Type	Status	Actions
MQLight_OnPrem	MQ Light	● Running	⚙
Receive_Events_from _MQLight_On-prem	MQ Light	● Running	⚙

Figure 2-20 Status of the stream

6. In the Message Hub dashboard, you can now see the streams that we just created. This confirms that the Message Hub service is successfully registered with the streams and is ready to consume the events using REST API. See Figure 2-21.



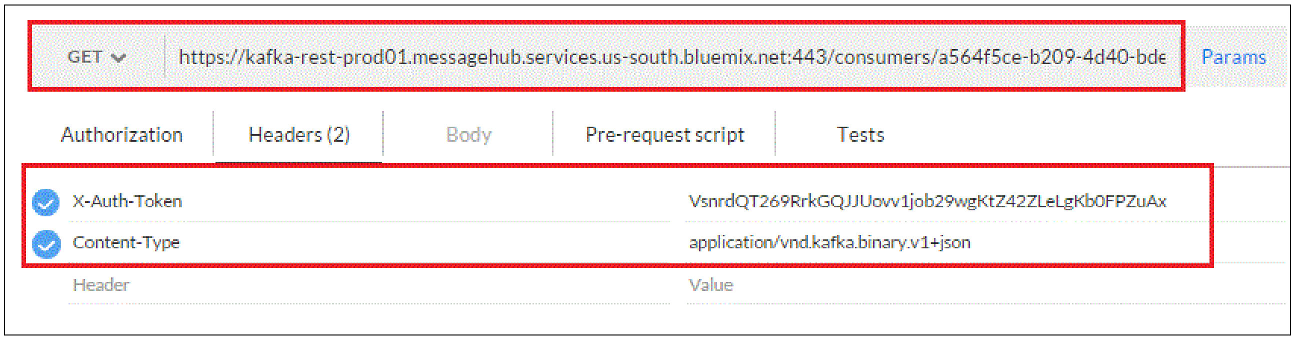
mqlight_onprem
receive_events_from-_mqlight_on-prem

Figure 2-21   Available streams in the Message Hub dashboard

After completing the above tasks, you can test the Kafka REST API to ensure that you are able to consume the events from the stream. You can try invoking the REST API that is obtained from the *Consuming information* of the stream created in the Message Connect service.

We illustrate here with an IBM MQ Light application running on-premises, which is publishing the data on a topic called *redbook/results*.

Use any REST client. Copy the REST API URL and required headers as obtained in step 4 on page 10 and enter them in the REST client, as shown in Figure 2-22.

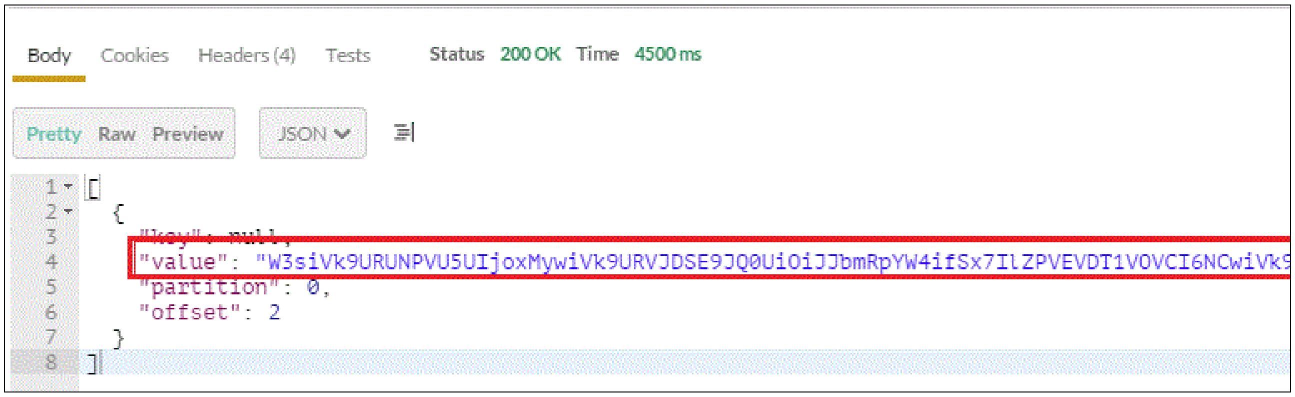


GET https://kafka-rest-prod01.messagehub.services.us-south.bluemix.net:443/consumers/a564f5ce-b209-4d40-bde

Header	Value
X-Auth-Token	VsnrdQT269RrkGQJJUovv1job29wgKtZ42ZLeLgKb0FPZuAx
Content-Type	application/vnd.kafka.binary.v1+json

Figure 2-22   Invoking Kafka API using REST Client

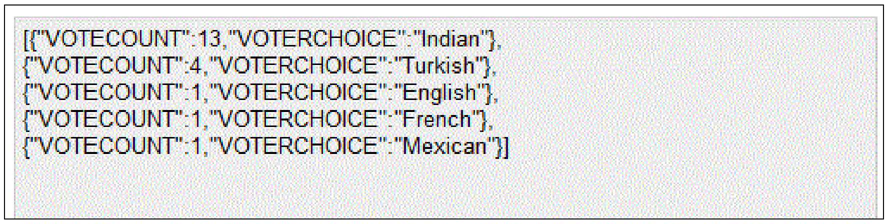
Assuming that the on-premises IBM MQ Light application is publishing data on topic 'redbook/results', by means of the REST call with the GET method, we receive the response in JSON format, as shown in Figure 2-23. The "value" field contains the actual contents of the data being published by the on-premises IBM MQ Light application. Because the Content-Type is set to *application/vnd.kafka.binary.v1+json*, the Kafka REST API returns the data in base64 encoded format.



```
[
  {
    "key": null,
    "value": "W3siVk9URUNPVU5UIjoxMywiVk9URVJDSE9JQ0UiOiJJbmRpYW4ifSx7IlZPVEVDT1VOVCI6NCwiVk9
    "partition": 0,
    "offset": 2
  }
]
```

Figure 2-23 Base64 encoded streaming data from REST API call

Decoding this data into ASCII format yields the text shown in Figure 2-24.



```
[{"VOTECOUNT":13,"VOTERCHOICE":"Indian"},
{"VOTECOUNT":4,"VOTERCHOICE":"Turkish"},
{"VOTECOUNT":1,"VOTERCHOICE":"English"},
{"VOTECOUNT":1,"VOTERCHOICE":"French"},
{"VOTECOUNT":1,"VOTERCHOICE":"Mexican"}]
```

Figure 2-24 Hex decoded data in ASCII format

The Kafka REST API is now ready for use in your Bluemix applications.

2.7 StrongLoop

StrongLoop, an IBM Company, is a leading provider of the Enterprise Node.js solution. It offers powerful API/integration capabilities and DevOps tools for Node.js.

Overview

The StrongLoop platform is an end-to-end platform for the full API lifecycle that allows you to visually develop APIs in Node and get them connected to new and legacy data. In addition, the platform provides graphical tools with DevOps features for clustering, profiling, and monitoring Node apps. See Figure 2-25 on page 33 to see StrongLoop platform capabilities.

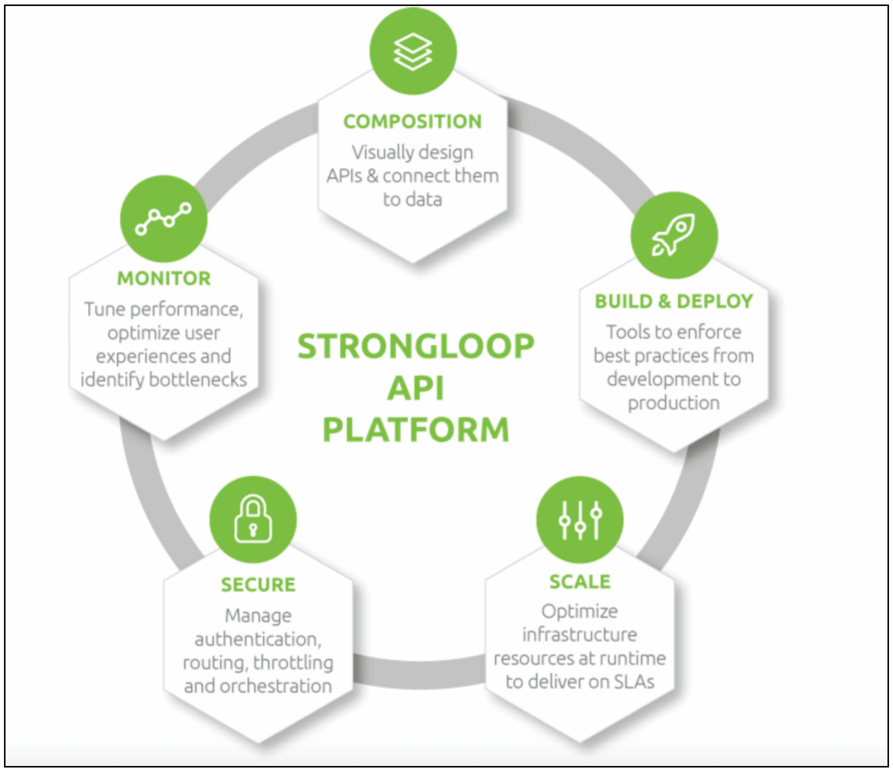

COMPOSITION
Visually design APIs & connect them to data
BUILD & DEPLOY
Tools to enforce best practices from development to production
SCALE
Optimize infrastructure resources at runtime to deliver on SLAs
SECURE
Manage authentication, routing, throttling and orchestration
MONITOR
Tune performance, optimize user experiences and identify bottlenecks
STRONGLOOP API PLATFORM

Figure 2-25 StrongLoop platform capabilities

API composition

The platform is built on top of the open source LoopBack® framework for Node.js. With LoopBack, you can easily create and compose scalable REST APIs that comply to the Swagger 2.0 specification. You can connect and access data from various databases (relational or NoSQL), SOAP, and REST APIs by leveraging the StrongLoop supported and community data connectors.

Developers can use this framework and tools provided with the platform (such as a graphical UI called *Arc* or the slc command line) to create application data models. Those models are actual source code, created without having to write the code, and are automatically exposed via a RESTful API by StrongLoop. In addition, StrongLoop automatically creates Swagger documentation for the REST APIs that expose your application data.

Build and deploy

After you create your APIs, you can use the build and deploy capabilities to automate the packaging and deployment of your Node.js application. There is built-in integration to trigger builds from a GIT repository and you can also trigger it from your local file system. The deployment automation makes it easy to deploy to one or multiple hosts that support your application.

Scale

StrongLoop provides a centralized management console for Node.js runtime environments called *Process Manager*. From the GUI console, you can easily create clusters, scale out and scale in clusters, and add additional processes to clusters.

Secure

StrongLoop and the LoopBack framework provided integrated security controls for app data models. This makes it easy to define which users and user groups can take which actions on your application data. There is also integrated support for Passport, which makes it easy to integrate your application with third-party login providers like Facebook, Twitter, Google, and so on. Passport also provides generic support for OAuth 2.0, so any OAuth 2.0-compliant service can be integrated into your app authentication process.

Monitor

StrongLoop also includes integrated monitoring and profiling that makes it easier for an administrator to understand the health and performance of their applications. Monitoring includes garbage collection, CPU and memory usage stats, event loop monitoring, response times, and slowest endpoints that are either hosted or on-premises.

StrongLoop on Bluemix

This powerful platform with all its capabilities can be hosted in the cloud, on-premises, and can also run on IBM Bluemix. It has been made available as two boilerplates on Bluemix to get you started quickly:

- A StrongLoop LoopBack Starter app: This boilerplate gets you started with a sample Node.js app that uses the StrongLoop Process Manager to start and supervise a LoopBack app. You can add your own APIs and push the changes back to the Bluemix environment.
- A StrongLoop Arc Starter app: The StrongLoop Arc boilerplate includes an app that launches an instance of StrongLoop Arc. You can use StrongLoop Arc to manage your Node.js apps that run on and are supervised by StrongLoop Process Manager.

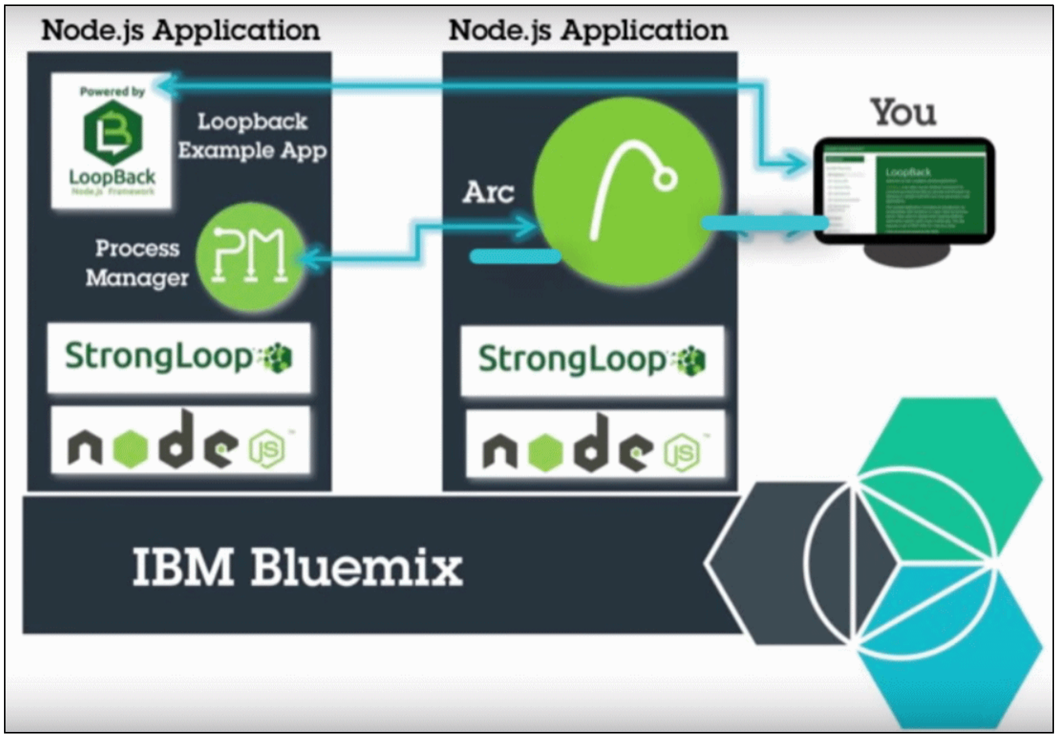


Figure 2-26 StrongLoop on Bluemix

Note: For more information about how to use the StrongLoop LoopBack starter app, see the Bluemix documentation at:

`https://www.ng.bluemix.net/docs/starters/LoopBack/index.html`

For more information about the StrongLoop Arc starter app, go to:

`https://www.ng.bluemix.net/docs/starters/StrongLoopArc/index.html`

3

Introduction to IBM messaging and integration products

This chapter describes the key concepts of IBM messaging and integration products that enable them to participate in the hybrid cloud integration.

In this chapter, we introduce you to products like IBM Integration Bus, IBM MQ, IBM MQ Light, and their features that help them interact with cloud application.

In this chapter, the following topics are covered:

3.1 Overview

Businesses are now living in a hybrid world and need to be able to tap into everything they have to create new systems of engagement.

IBM Integration Bus is an essential component of the next generation integration platform providing robust and extensive enterprise integration features that bridge between modes of IT.

IBM MQ provides a universal messaging backbone for enterprises to transport messages and data inside and outside the organization, and IBM MQ Light makes it easier for developers to incorporate asynchronous messaging into applications to help make them more responsive and easier to scale. Using IBM MQ Light, you can quickly integrate with application frameworks through easy-to-use application programming interfaces (APIs).

The following sections in this chapter describe more about each of these products.

3.2 Introduction to IBM MQ Light

IBM MQ Light is designed to allow applications to exchange discrete pieces of information in the form of messages. IBM MQ Light removes much of the complexity of TCP/IP networking, and provides a higher level set of abstractions with which to build your applications. The IBM MQ Light API is based on the OASIS Standard AMQP Version 1.0 wire protocol. AMQP is used to receive, queue, route, and deliver messages.

Some common implementations are event notification or worker offload, or generally connecting to external systems as shown in Figure 3-1.

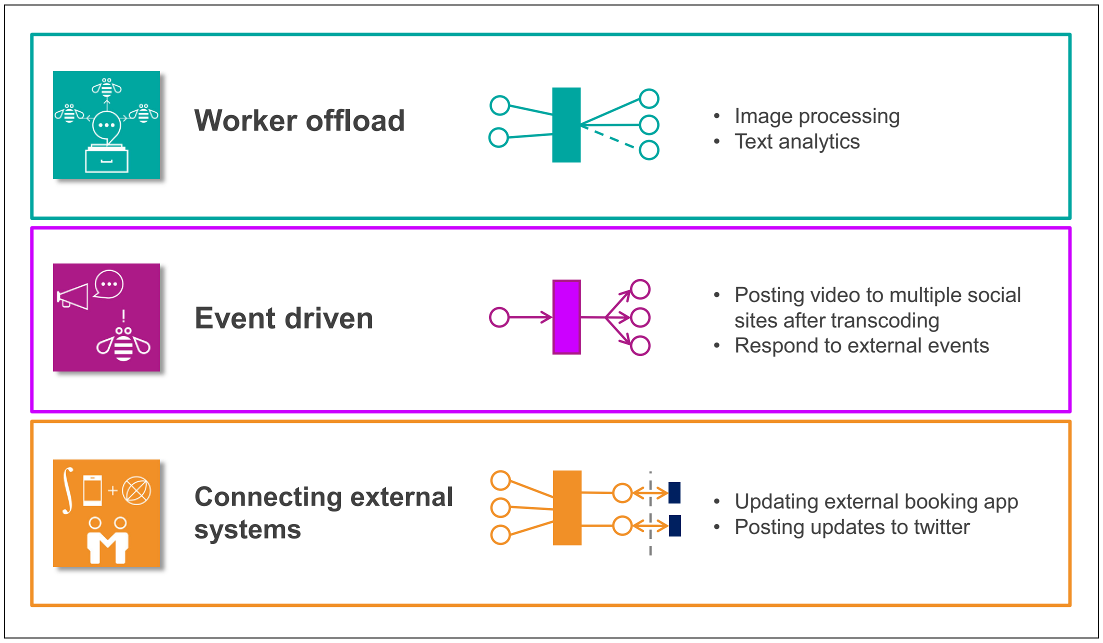

Worker offload
• Image processing
• Text analytics
Event driven
• Posting video to multiple social sites after transcoding
• Respond to external events
Connecting external systems
• Updating external booking app
• Posting updates to twitter

Figure 3-1 Common use cases for IBM MQ Light

- **Worker offload**

  A key factor in a web application's responsiveness is the amount of time that the web server takes to process an HTTP request and send a response. If a web server needs to perform CPU intensive, or high-latency processing before it can respond to a request, the number of requests that the server can process is reduced. This can lead to a situation where the server can no longer keep up with the rate at which new requests arrive, and the web application becomes unusable.

  By using IBM MQ Light, you can now offload the database and other backend work to another component of the application. Now, one component responds to web requests (referred as the "front-end" component) and one performs the database updates (which we call the "back-end" component).

  Figure 3-2 shows the interactions between our two application components, and the various other parts of the system that now make up our complete application.

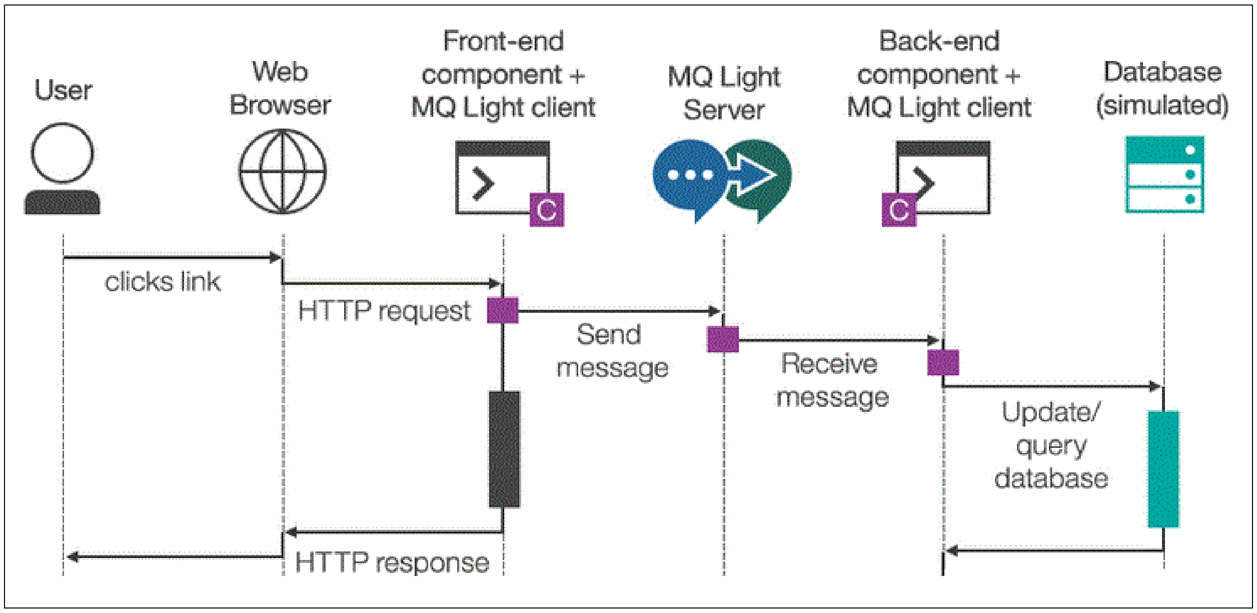


Figure 3-2 IBM MQ Light worker offload pattern

  A detailed scenario implementing this pattern is described in 6.5, "Subscenario 3: Web application saving data on IBM MQ server" on page 114.

  A tutorial for worker offload pattern is also available at the following link:

  https://developer.ibm.com/messaging/mq-light/docs/worker-offload-tutorial

- **Batch processing**

  A variant of the worker offload, but where the work is deliberately queued for processing at a later time, perhaps when off-peak, less expensive computing resources are available.

- **Event integration**

  Where an application publishes data to a group of applications. For example, publishing stock price updates to a set of trading applications. By acting as a "man in the middle," messaging allows the updates to be simply sent to the messaging service, which then manages the storage and distribution of the messages to each interested application.

3.2.1 IBM MQ Light messaging styles

IBM MQ Light is designed to allow applications to exchange discrete pieces of information in the form of messages.

The following terminology is used with IBM MQ Light messaging:

- **Messages**

  All data is carried in the form of messages. Messages can include:

  - The message payload, or the data to carry between applications.
  - Name-value properties.
  - Attributes that are interpreted by the IBM MQ Light messaging service.

- **Topics**

  Applications send messages to a topic. A topic is a string that identifies a location in the topic space that messages are routed to, and indirectly governs which applications receive a copy of the message.

- **Destinations**

  Applications receive messages from destinations. Destinations are associated with a topic, or set of topics, using a pattern.

Following are the messaging styles in which IBM MQ Light applications can communicate with each other.

Simple Receive

Applications receive messages by creating a destination with a pattern, which matches the topics they are interested in as shown in Figure 3-3.

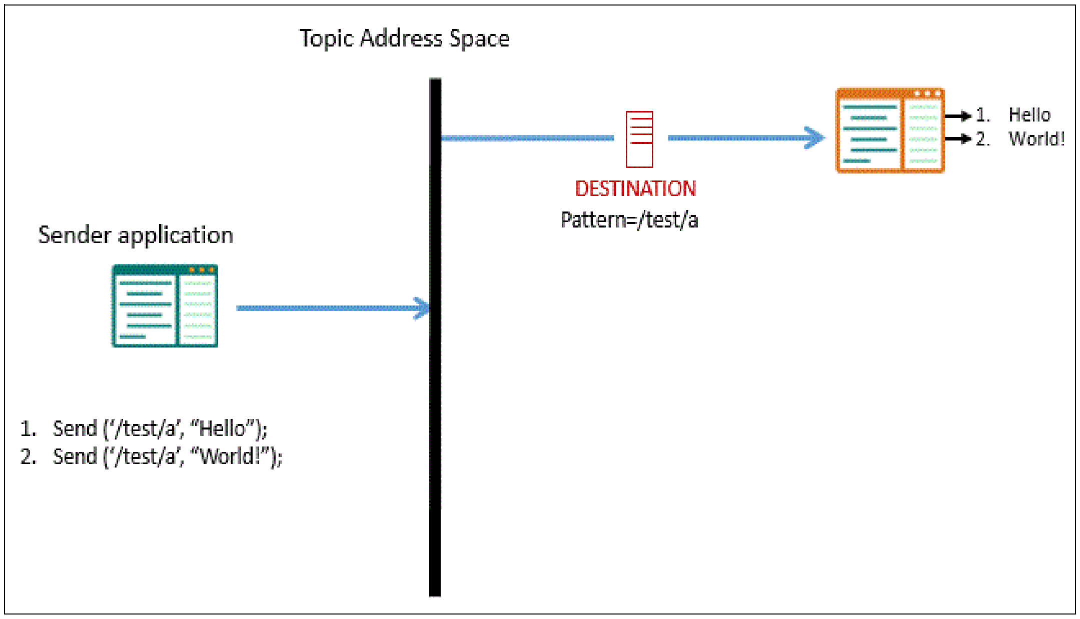


Figure 3-3 Simple receive from the destination

Destinations are associated with a topic, or set of topics, using a pattern. Destinations store a copy of each message sent to a topic that matches the destination's pattern, until it is consumed by an application. Zero, one, or many destinations can be associated with any given topic. If a message is sent to a topic without a destination, it is not delivered to any application.

Publish or Subscribe

Multiple destinations can be created that match the same topic or topic hierarchy as shown in Figure 3-4. This model is similar to publish/subscribe style messaging.

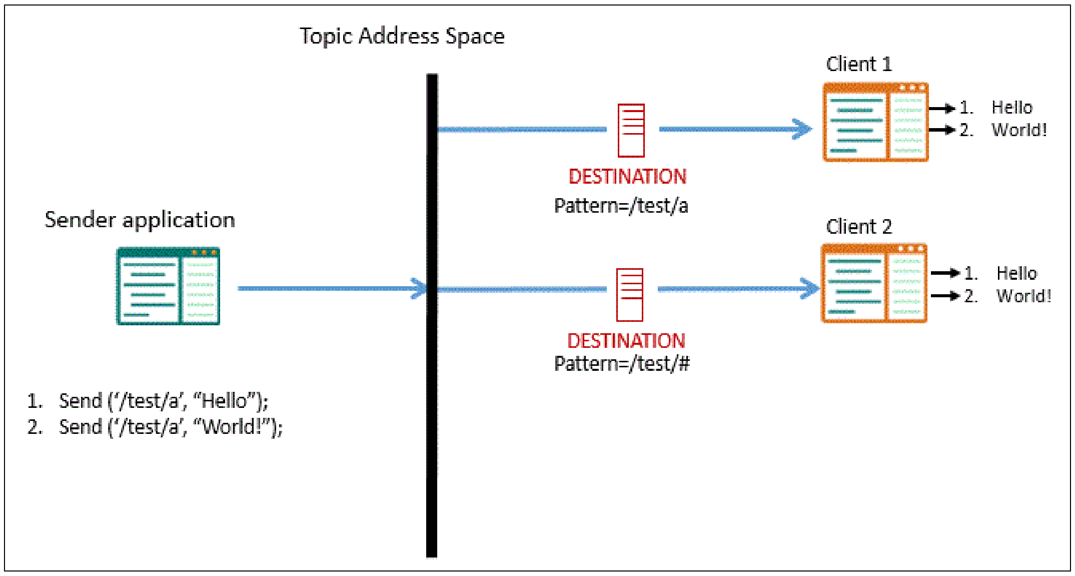


Figure 3-4 Publish/subscribe model using IBM MQ Light

Persistent destination

You can specify a "time to live" parameter when you subscribe to the destination. So even if the consuming client is not active, the message is persisted in the destination for the set amount of time. Figure 3-5 shows the messages that are published by the sender application are persisted at the destination.

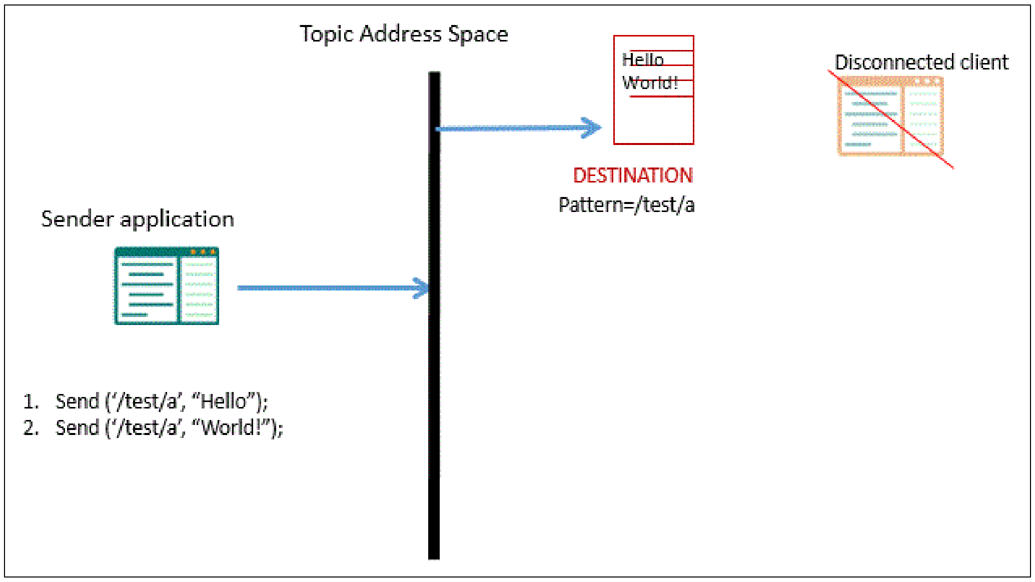


Figure 3-5 Persisting messages in the destination

Sharing

This model shows the clients attaching to the same topic pattern and share name attach to the same share destination. Applications can either exclusively use a destination, meaning that they will receive all its messages, or they can share a destination, meaning that messages arriving at the destination will be shared among the applications sharing the destination.

When more than one application is consuming messages from a destination, the messages are shared out so that each message is delivered to only one application. Figure 3-6 shows how the messages are shared between the two clients.

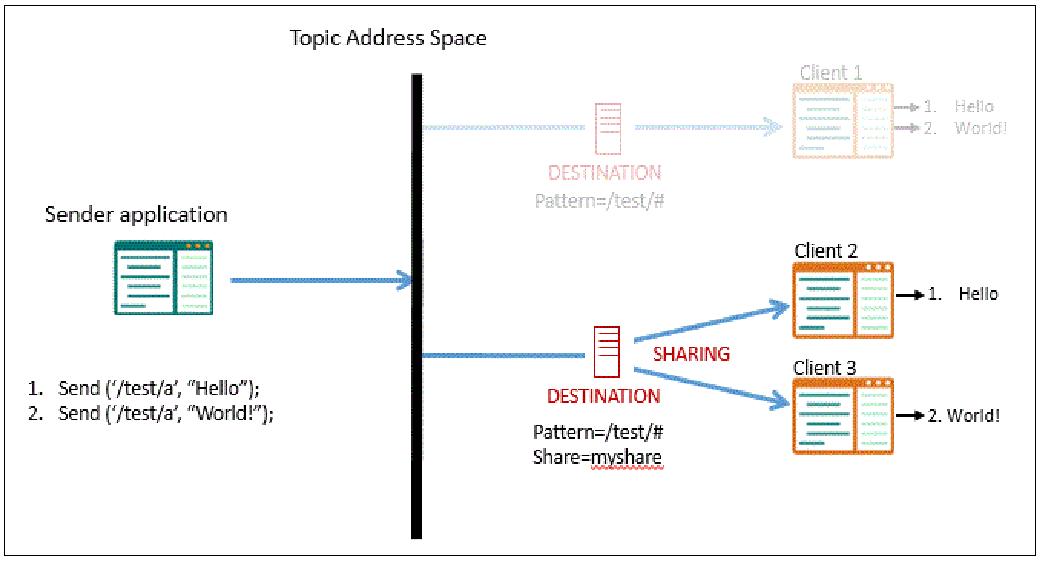


Figure 3-6 Shared destination by multiple clients

Client takeover

As shown in Figure 3-7, you can configure your client application to take over the workload from another client application that was receiving the messages before. By using the same client ID to connect to the IBM MQ Light service, you do not need to reconfigure your new client to connect and receive messages from the destination.

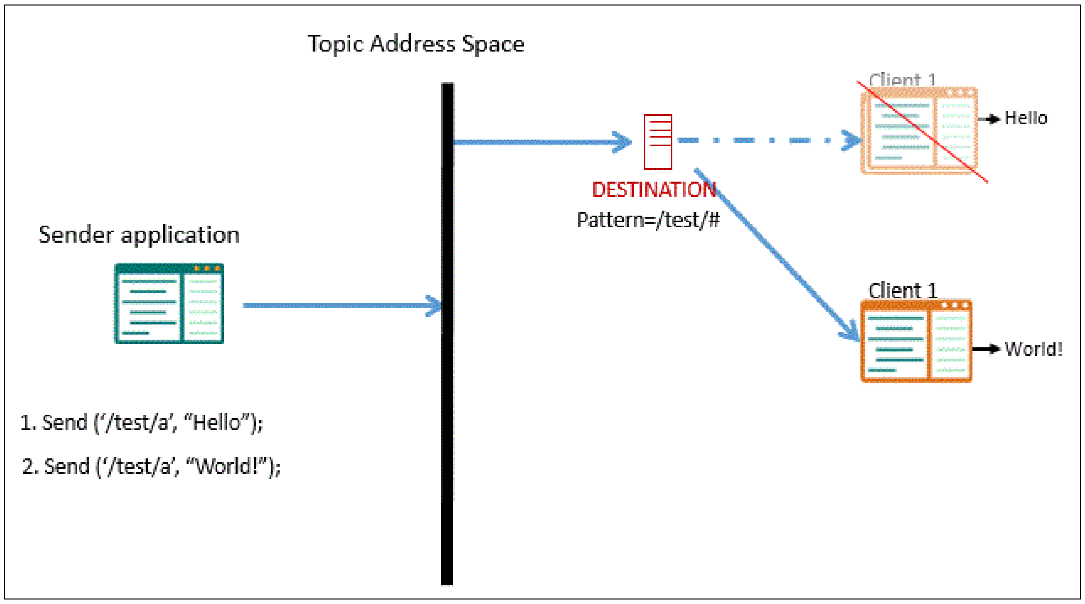


Figure 3-7 Client takeover configuration

Here are the steps of the client takeover:

1. Originally, the first application connected to IBM MQ Light Service specifying the optional ID as “*Client1*” and listening for destination “/test/#”:
   a. The sender application sent a message “Hello” to topic test/a
   b. The first application received the message “Hello”
2. A second application connected to IBM MQ Light Service specifying the optional ID as “*Client1*” and listening for destination “/test/”#:
   a. The second application preempted the original connection for the first application
   b. The sender application sent a message “World!” to topic test/a
   c. Second application received the message “World!”

The above behavior can be summarized as:

- Application connected to IBM MQ Light service specified (optional) client ID
- Another application reused the same client ID preempted the original connection

This capability allows for flexible work management where a new worker can take over the workload.

In summary, the IBM MQ Light API has the following messaging features:

- At-most-once message delivery
- At-least-once message delivery
- Topic string destination addressing
- Message and destination durability
- Shared destinations to allow multiple subscribers to share workload
- Client takeover for easy resolution of hung clients
- Configurable read ahead of messages
- Configurable acknowledgment of messages

3.2.2 Application connectivity patterns using IBM MQ Light

Following are some of the connectivity patterns for IBM MQ Light applications:

- The IBM MQ Light service enables applications that are hosted within IBM Bluemix to connect to other applications hosted inside Bluemix.
- IBM MQ Light applications that run either outside of Bluemix or in a Bluemix dedicated environment (using service syndication) can connect to the service if they connect by using an encrypted connection (AMQPS), for example, by using the IBM MQ Light Java or JavaScript API. This is not supported for Java Message Service (JMS).
- Applications that are hosted inside Bluemix can also connect to IBM MQ and IBM MQ Light servers outside of Bluemix by way of the secure connector.

3.3 Introduction to IBM MQ

Most businesses have networks of diverse hardware and software. However, related programs in different parts of a network must be able to communicate in a way unaffected by variations in hardware, in operating systems, in programming languages, and in communication protocols. Moreover, businesses need to be able to run related programs independently of each other. And all this needs to be achieved with an overall reduction in the number of sessions in the network. Complex though the problem may be, it needs a solution that works in the same way, and equally well, between programs on a single processor (in both like and unlike environments) and between programs at different nodes of a varied network. IBM MQ provides just such a solution.

Following are three key features of the messaging and queuing style of programming:

- Communicating programs can run at different times.
- There are no constraints on application structure.
- Programs are insulated from network complexities.

IBM MQ is messaging and queuing middleware. It allows application programs to use message queuing to participate in message-driven processing. IBM MQ enables programs to communicate with one another across a network of unlike components (processors, operating systems, subsystems, and communication protocols) using a consistent application programming interface known as the *message queue interface* (or MQI) wherever the applications run. This makes it easier for you to port application programs from one platform to another.

IBM MQ provides a universal messaging backbone with robust connectivity for flexible and reliable messaging for applications and the integration of existing IT assets using a service-oriented architecture (SOA).

IBM MQ provides several modes of operation: Point-to-point, publish/subscribe, and file transfer.

A few common terminologies are associated with IBM MQ messaging and are used in this book:

- Messaging

  Programs communicate by sending each other data in messages rather than by calling each other directly.

- Queuing

  Messages are placed on queues, so that programs can run independently of each other, at different speeds and times, in different locations, and without having a direct connection between them.

- Point-to-point

  Applications send messages to a queue, or to a list of queues. The sender must know the name of the destination, but not where it is.

- Publish/subscribe

  Applications publish a message on a topic, such as the result of a game played by a team. IBM MQ sends copies of the message to applications that subscribe to the results topic. They receive the message with the results of games played by the team. The publisher does not know the names of subscribers, or where they are.

3.4 IBM MQ support for IBM MQ Light APIs

Application developers who use IBM MQ Light benefit from the ease with which they can make their applications responsive and scalable, making it easier to prototype and develop business applications rapidly. IBM MQ supports the IBM MQ Light messaging API (with IBM MQ V8.0.0.4 refresh or later) so that you can use IBM MQ to deploy your IBM MQ Light application to an on-premises IBM MQ environment. You can integrate the IBM MQ Light application with other applications that are already connected to IBM MQ.

Important: Install the AMQP Service component by using the IBM MQ V8.0.0.4 manufacturing refresh, not the V8.0.0.4 Fix Pack. You cannot install the AMQP component on a version of the queue manager earlier than V8.0.0.4.

3.4.1 How to use IBM MQ Light API with IBM MQ

The IBM MQ support for IBM MQ Light APIs allows an IBM MQ administrator to define a new type of channel: An *AMQP channel*.

Note: Ensure that you select the package for AMQP when installing IBM MQ 8004 manufacturing refresh. For example, on Linux x86_64 platform, the package is named as MQSeriesAMQP-8.0.0-4.x86_64.rpm.

When the AMQP channel is started, it defines a port number that accepts connections from IBM MQ Light applications.

The AMQP channel can be managed in the same way as other IBM MQ channels. You can use MQSC commands, PCF command messages, or IBM MQ Explorer to define, start, stop, and manage the channels.

When an AMQP channel is started, you can test it by connecting an IBM MQ Light application by using any of the following methods:

- Using the IBM MQ Light client for Node.js and Java.
- Using the IBM MQ Light early access program client for Ruby and Python.
- Using another AMQP Version 1.0 client. For example, Apache Qpid Proton.

By defining and starting an AMQP channel, IBM MQ Light or AMQP 1.0 applications can publish messages that are received by existing IBM MQ applications. Messages published through an AMQP channel are all sent to IBM MQ topics, not IBM MQ queues. An IBM MQ application that has created a subscription by using the MQSUB API call receives messages published by AMQP 1.0 applications, if the topic string or topic object used by the IBM MQ application matches the topic string published by the AMQP client.

IBM MQ Light or AMQP 1.0 applications can also consume messages that are published by existing IBM MQ applications. Messages published by IBM MQ applications to an IBM MQ topic or topic string are received by an AMQP 1.0 application if the application has subscribed with a topic pattern that matches the published topic string. Figure 3-8 on page 47 shows the communication model between IBM MQ Light clients with IBM MQ and native IBM MQ applications.

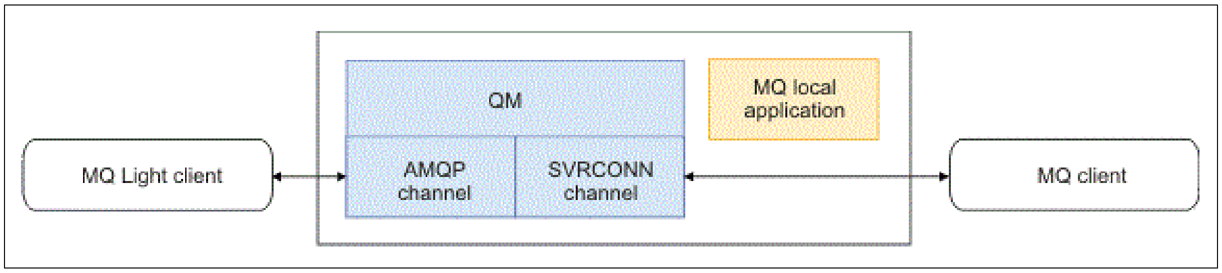


Figure 3-8 IBM MQ Light Communication channel with IBM MQ

This topology is useful when the application might be processing financial or sensitive data that must be held only on systems inside your firewall. IBM MQ Light applications can share a topic space with existing IBM MQ applications, which enables them to interact with existing enterprise systems.

The detailed steps for configuring IBM MQ Light service with IBM MQ are illustrated in the IBM MQ Knowledge Center at the following link.

`http://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.amqp.doc/tamqp_creating.htm?lang=en`

3.5 Introduction to IBM Integration Bus

Enterprise systems consist of many logical endpoints, for example, off-the-shelf applications, services, cloud apps (SaaS), web apps, devices, appliances, custom-built software. The endpoints expose a set of inputs and outputs, which comprise protocols like IBM MQ, TCP/IP, HTTP, file system, FTP, and message formats like, binary (C/COBOL), XML, industry (SWIFT, EDI), user-defined. Integration is about connecting these endpoints together in meaningful ways and some of the common uses are Route, Transform, Enrich, Filter, Monitor, Distribute, Correlate, Fire and Forget, Request/Reply, Publish/Subscribe, and Aggregation.

The IBM Integration Bus, formerly known as WebSphere® Message Broker, is an enterprise service bus that provides universal connectivity for service-oriented environments and non-service oriented environments. As shown in Figure 3-9, it connects to a range of different systems and endpoints. It also does the universal transformation and routing of messages to integrate various applications, services, and protocols.

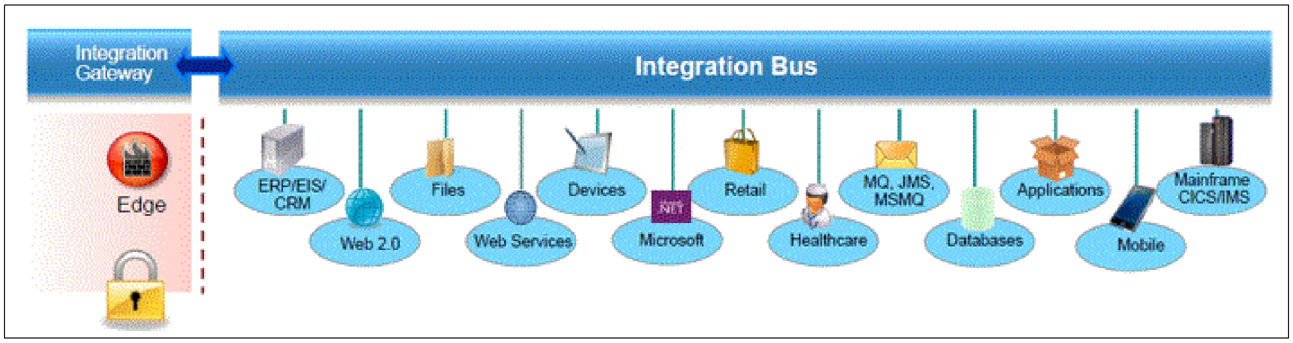


Figure 3-9 Integration Bus and the endpoints

You can use IBM Integration Bus to connect applications together, regardless of the message formats or protocols that they support. This connectivity means that your diverse applications can interact and exchange data with other applications in a flexible, dynamic, and extensible infrastructure. IBM Integration Bus routes, transforms, and enriches messages from one location to any other location.

An integration solution is easy to develop, deploy, and manage by using the Integration Bus. It is independent of any programming language, so developers can write business logic in the programming languages of their choice, including Java, Microsoft .NET, PHP, ESQL, and so on. It allows non-programmers to do the necessary message transformation by using the Mapping Editor.

The product supports a wide range of protocols: IBM MQ, JMS 1.1 and 2.0, HTTP and HTTPS, web services (SOAP and REST), File, Enterprise Information Systems (including SAP and Siebel), and TCP/IP.

3.5.1 Technical overview of IBM Integration Bus

IBM Integration Bus enables information packaged as messages to flow between different business applications, ranging from large traditional systems through to unmanned devices such as sensors.

Figure 3-10 explains the main components of IBM Integration Bus and how they interact.

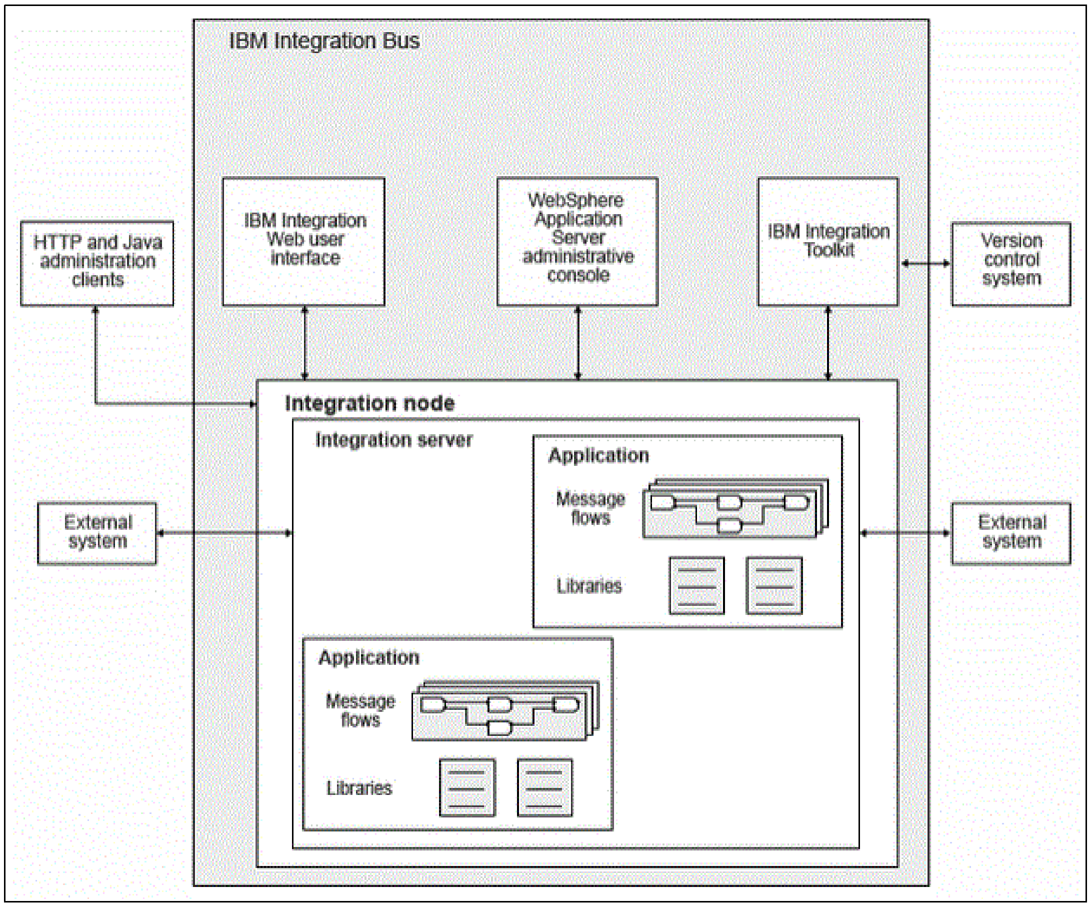


Figure 3-10 IBM Integration Bus components diagram

An *integration node* is a set of execution processes that hosts one or more message flows to route, transform, and enrich in-flight messages.

An *integration server* is a named grouping of message flows that have been assigned to an integration node. An integration server process is also known as a *DataFlowEngine* (DFE).

An *application* is a container for all the resources that are required to create a solution. An application can contain IBM Integration Bus resources, such as flows, message definitions, libraries, and Java archive (JAR) files.

A *library* is a logical grouping of related code, data, or both. You can use a library to group resources of the same type or function, and to aid the management and reuse of such resources.

A *message flow* is a sequence of processing steps that run in the integration node when an input message is received. You define a message flow in the IBM Integration Toolkit by including a number of message flow nodes, each of which represents a set of actions that define a processing step. How you join the message flow nodes together determines which processing steps are carried out, in which order, and under which conditions.

The *IBM Integration Toolkit* is an integrated development environment and graphical user interface based on the Eclipse platform. Application developers work in separate instances of the IBM Integration Toolkit to develop resources associated with message flows. The IBM Integration Toolkit connects to one or more integration nodes to which the message flows are deployed.

IBM Integration Bus processes messages in two ways: Message routing and message transformation:

- Message routing

  Messages can be routed from sender to recipient based on the content of the message. The message flows that you design control message routing. A message flow describes the operations to be performed on the incoming message, and the sequence in which they are carried out.

  IBM supplies built-in nodes and samples for many common functions. You create message flows in the IBM Integration Toolkit.

- Message transformation

  Messages can be transformed before being delivered. They can be transformed from one format to another, perhaps to accommodate the different requirements of the sender and the recipient. They can be transformed by modifying, combining, adding, or removing data fields, perhaps involving the use of information stored in a database.

In this book, we discuss features and scenarios using IBM Integration Bus v10.

Figure 3-11 on page 50 shows a quick glance at the key features of IBM Integration Bus v10.

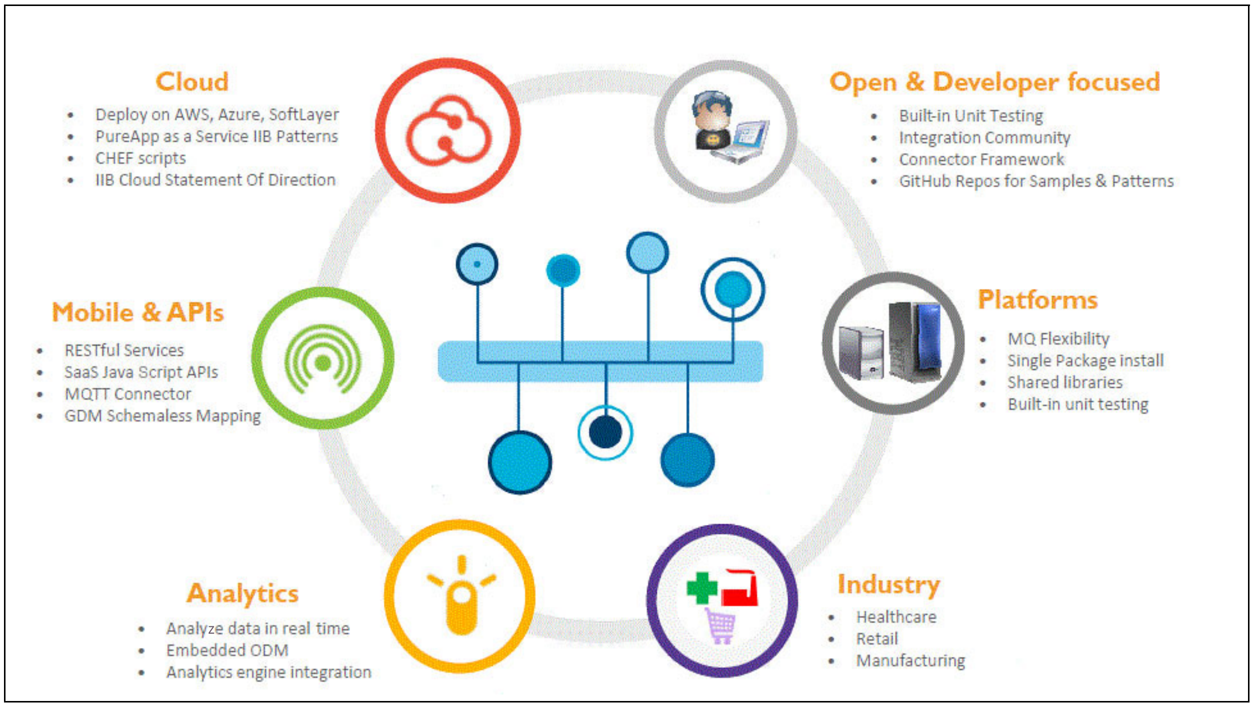


Figure 3-11 IBM Integration Bus v10 features at a glance

IBM Integration Bus v10 introduces first-class support for REST API as a specialized application. It provides a simple way to receive JSON or HTTP and expose a REST API. A REST API contains a set of resources, and a set of operations that can be called on those resources. The operations in a REST API can be called from any HTTP client, including client-side JavaScript code that is running in a web browser.

Following are features of a REST API project:

- Swagger spec provides a framework implementation for describing, producing, consuming, and visualizing RESTful APIs.
- It defines a metadata format based on JSON schema to describe the REST APIs, their parameters, and the messages that are exchanged.
- Import Swagger (v2.0) to create the REST API project.
- Original .json files are included (unchanged) in the project.
- REST APIs can use Path, Header, and Query parameters.
- As a client of an IBM Integration Bus REST API, use existing Swagger tools and projects to retrieve Swagger definitions from IBM Integration Bus.

3.5.2 Developing IBM Integration Bus message flow as a REST API

When you import a Swagger document to generate the REST API project in IBM Integration Bus Toolkit:

- REST API descriptor shows operations.
- The generated top-level message flow contains the HTTP Input node (uses Integration server listener).
- HTTP Input configured with routing table based on HTTP method and URL.
- Clicking each operation nickname generates an associated IBM Integration Bus subflow.

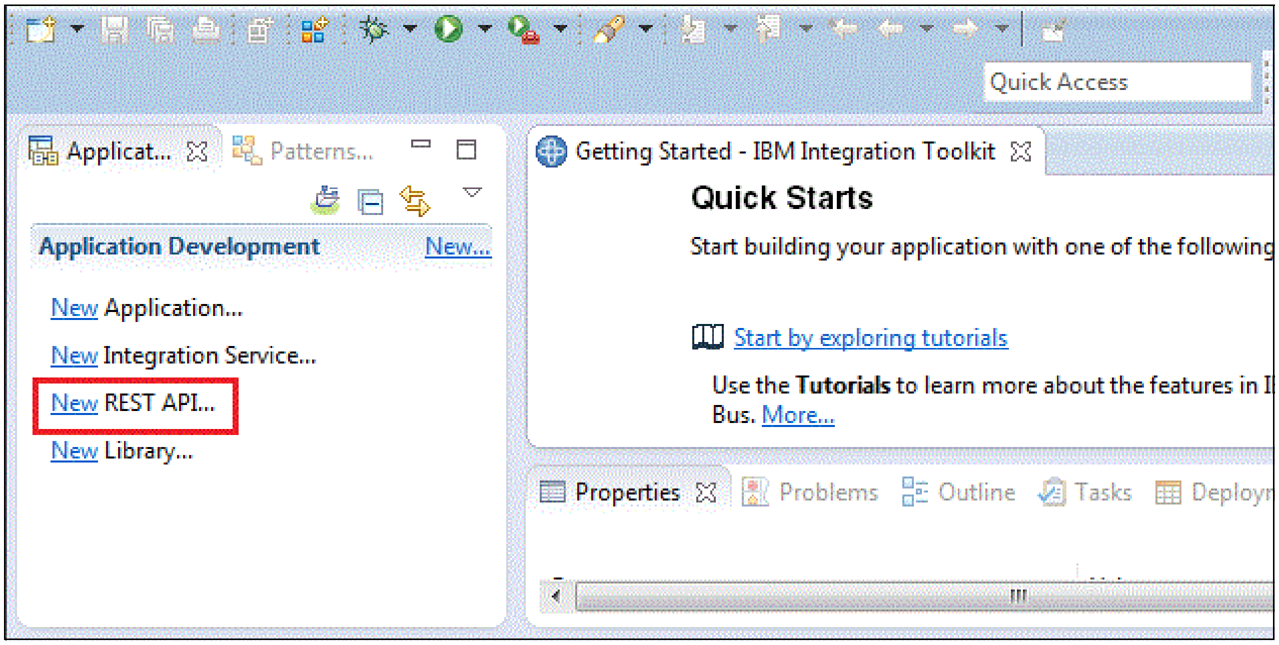
- Error handler links also created for HTTP Timeout, Failure, and Catch.
- After creation, add references to shared libs (or static libs) to aid subflow implementation.

Steps to develop and deploy a REST API project

To build a REST API in IBM Integration Bus, you must provide a Swagger document. A Swagger document is the REST API equivalent of a Web Services Description Language (WSDL) document for a SOAP-based web service. The Swagger document specifies the list of resources that are available in the REST API and the operations that can be called on those resources. The Swagger document also specifies the list of parameters to an operation, including the name and type of the parameters, whether the parameters are required or optional, and information about acceptable values for those parameters. IBM Integration Bus supports version 2.0 of the Swagger specification.

The steps to implement IBM Integration Bus message flow as a REST API are illustrated below:

1. Create a **New REST API** project in Integration Bus Toolkit as shown in Figure 3-12.



Quick Starts

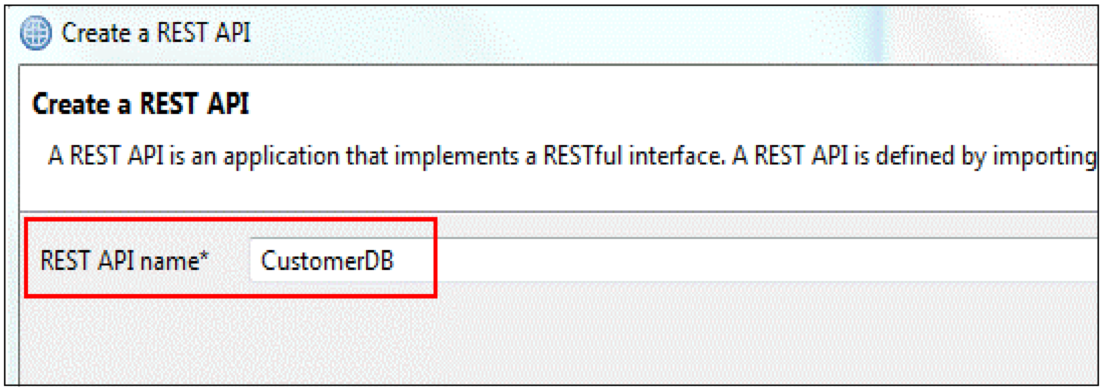
Start building your application with one of the following

Start by exploring tutorials

Use the **Tutorials** to learn more about the features in I Bus. More...

Figure 3-12 Create a new REST API project

2. Provide a name for your REST API project as shown in Figure 3-13.



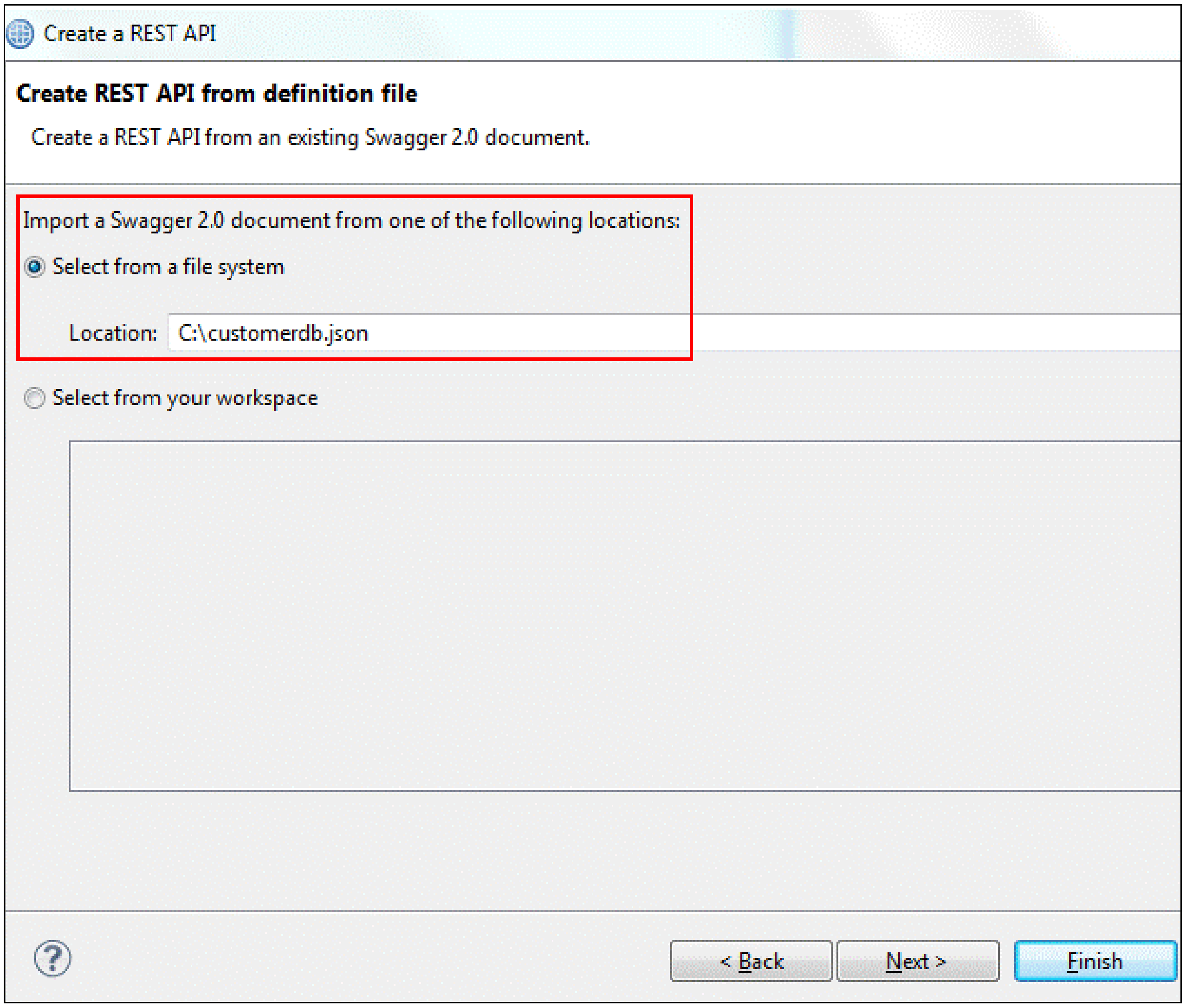
Create a REST API

A REST API is an application that implements a RESTful interface. A REST API is defined by importing

REST API name* CustomerDB

Figure 3-13 Name the REST API project

3. Select from a file system the path to the Swagger document that describes the resources and operations that you want in the REST API and click **Next**. You can import the Swagger document from the file system or from an existing project in the workspace as shown in Figure 3-14.


Create a REST API

Create REST API from definition file

Create a REST API from an existing Swagger 2.0 document.

Import a Swagger 2.0 document from one of the following locations:

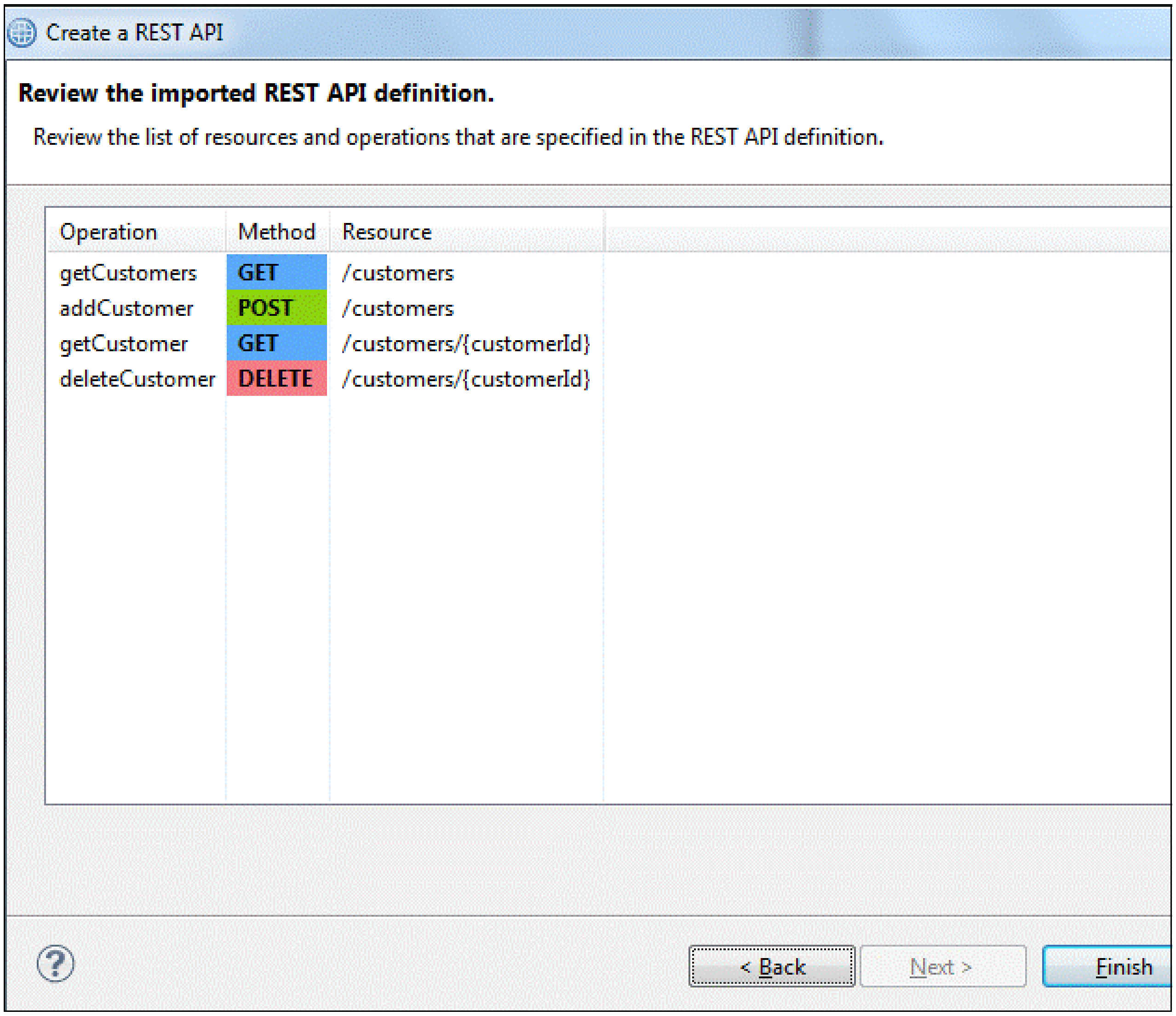
[✓ Select from a file system]

Location: C:\customerdb.json

Select from your workspace

Figure 3-14 Importing the Swagger document

4. Upon importing the .json file, the dialog box displays the API description, operations, and method type as shown in Figure 3-15.


Create a REST API

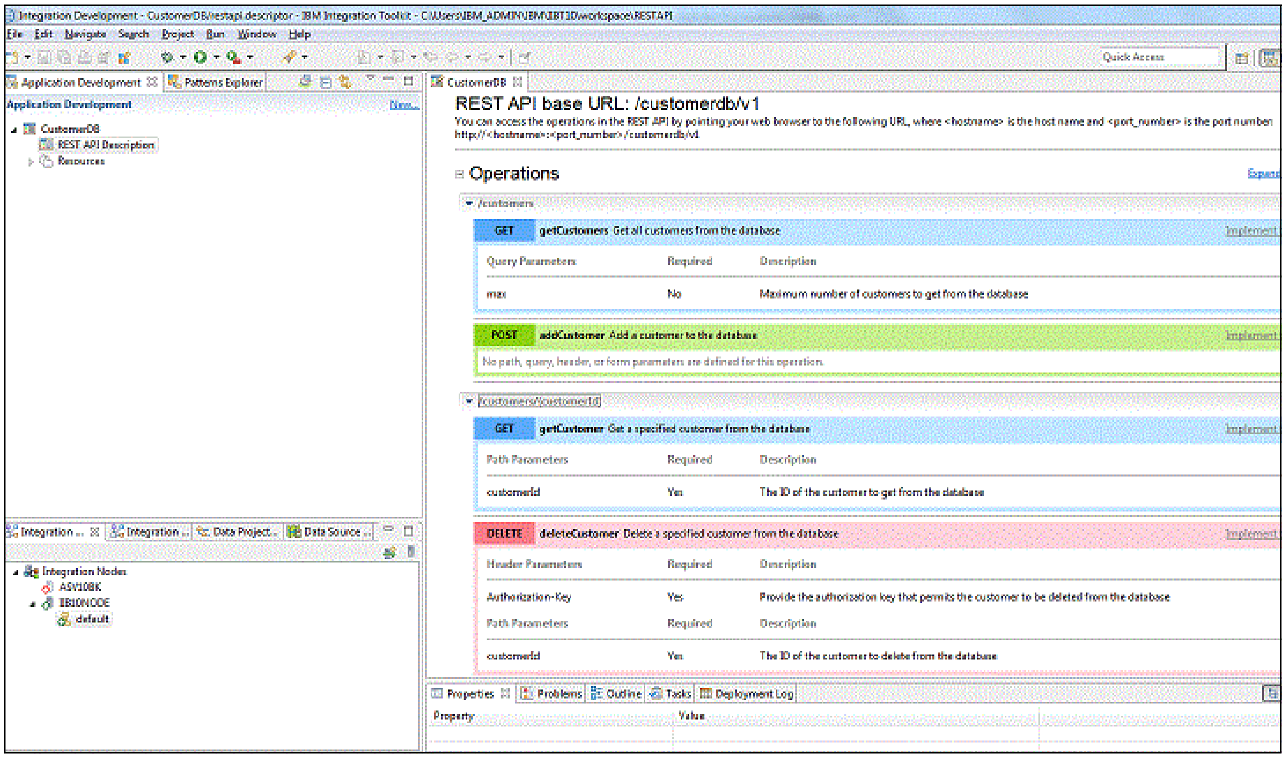
Review the imported REST API definition.

Review the list of resources and operations that are specified in the REST API definition.

Operation	Method	Resource
getCustomers	GET	/customers
addCustomer	POST	/customers
getCustomer	GET	/customers/{customerId}
deleteCustomer	DELETE	/customers/{customerId}

Figure 3-15 REST API operations

5. Click **Finish** to complete the import operation of the Swagger API document. The REST API description for the new REST API opens automatically in the IBM Integration Bus Toolkit as shown in Figure 3-16.



REST API base URL: /customerdb/v1

You can access the operations in the REST API by pointing your web browser to the following URL, where <hostname> is the host name and <port_number> is the port number:
http://<hostname>:<port_number>/customerdb/v1

Operations

/customers

GET getCustomers Get all customers from the database

Query Parameters	Required	Description
max	No	Maximum number of customers to get from the database

POST addCustomer Add a customer to the database

No path, query, header, or form parameters are defined for this operation.

/customers/{customerId}

GET getCustomer Get a specified customer from the database

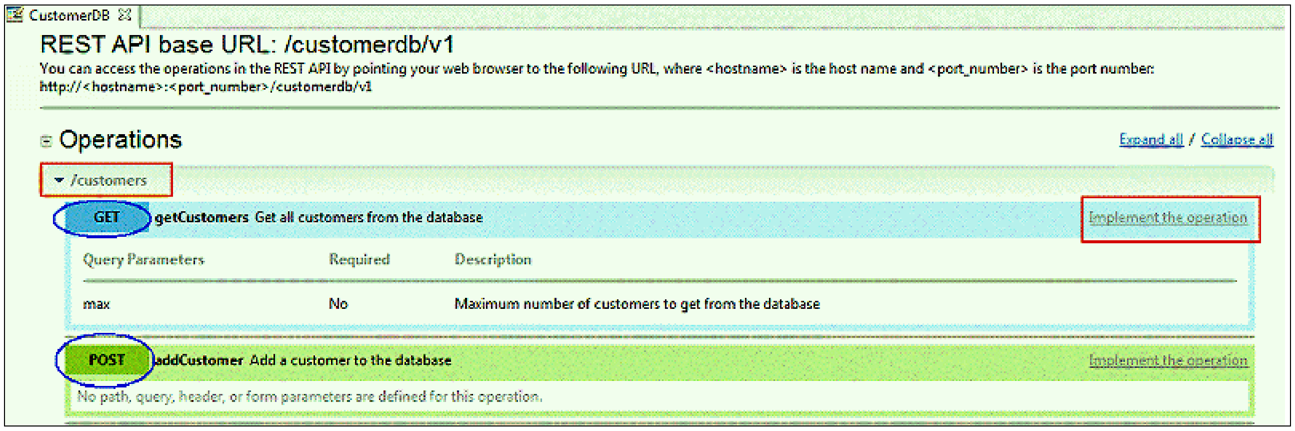
Path Parameters	Required	Description
customerId	Yes	The ID of the customer to get from the database

DELETE deleteCustomer Delete a specified customer from the database

Header Parameters	Required	Description
Authorization-Key	Yes	Provide the authorization key that permits the customer to be deleted from the database
Path Parameters	**Required**	**Description**
customerId	Yes	The ID of the customer to delete from the database

Figure 3-16 The generated REST API project in toolkit

6. You now implement the operation in a REST API. For every operation and its method type that has been imported from the Swagger document, there is an *Implement the operation* link as shown in Figure 3-17. Upon clicking this link, a new subflow is automatically created and opened.



REST API base URL: /customerdb/v1

You can access the operations in the REST API by pointing your web browser to the following URL, where <hostname> is the host name and <port_number> is the port number:
http://<hostname>:<port_number>/customerdb/v1

Operations

/customers

GET getCustomers Get all customers from the database

Query Parameters	Required	Description
max	No	Maximum number of customers to get from the database

POST addCustomer Add a customer to the database

No path, query, header, or form parameters are defined for this operation.

Figure 3-17 Implement REST operation

7. Implement the operation as shown in Figure 3-18 by adding any of the standard message flow nodes that are available with IBM Integration Bus to the subflow.

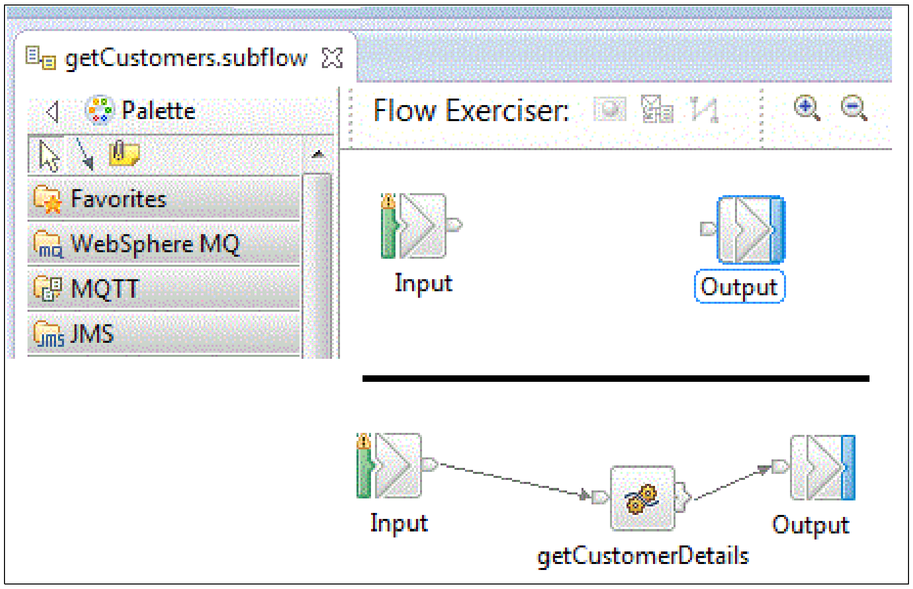
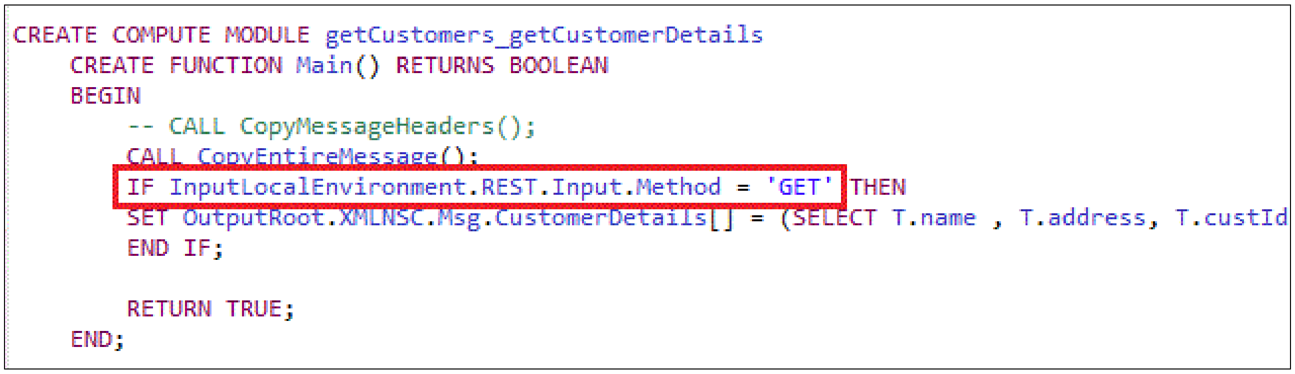


Figure 3-18 Implementing the subflow

8. Information about the current operation is automatically placed into the local environment tree as shown in Figure 3-19. You can use this information in your implementation if you want to determine which operation in the REST API was called, which HTTP method was used, the request path, or the request URI.



```
CREATE COMPUTE MODULE getCustomers_getCustomerDetails
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        -- CALL CopyMessageHeaders();
        CALL CopyEntireMessage();
        IF InputLocalEnvironment.REST.Input.Method = 'GET' THEN
        SET OutputRoot.XMLNSC.Msg.CustomerDetails[] = (SELECT T.name , T.address, T.custId
        END IF;

        RETURN TRUE;
    END;
```

Figure 3-19 LocalEnvironment for REST tree

9. When the operations are implemented and saved, they get automatically added to the main message flow as shown in Figure 3-20.

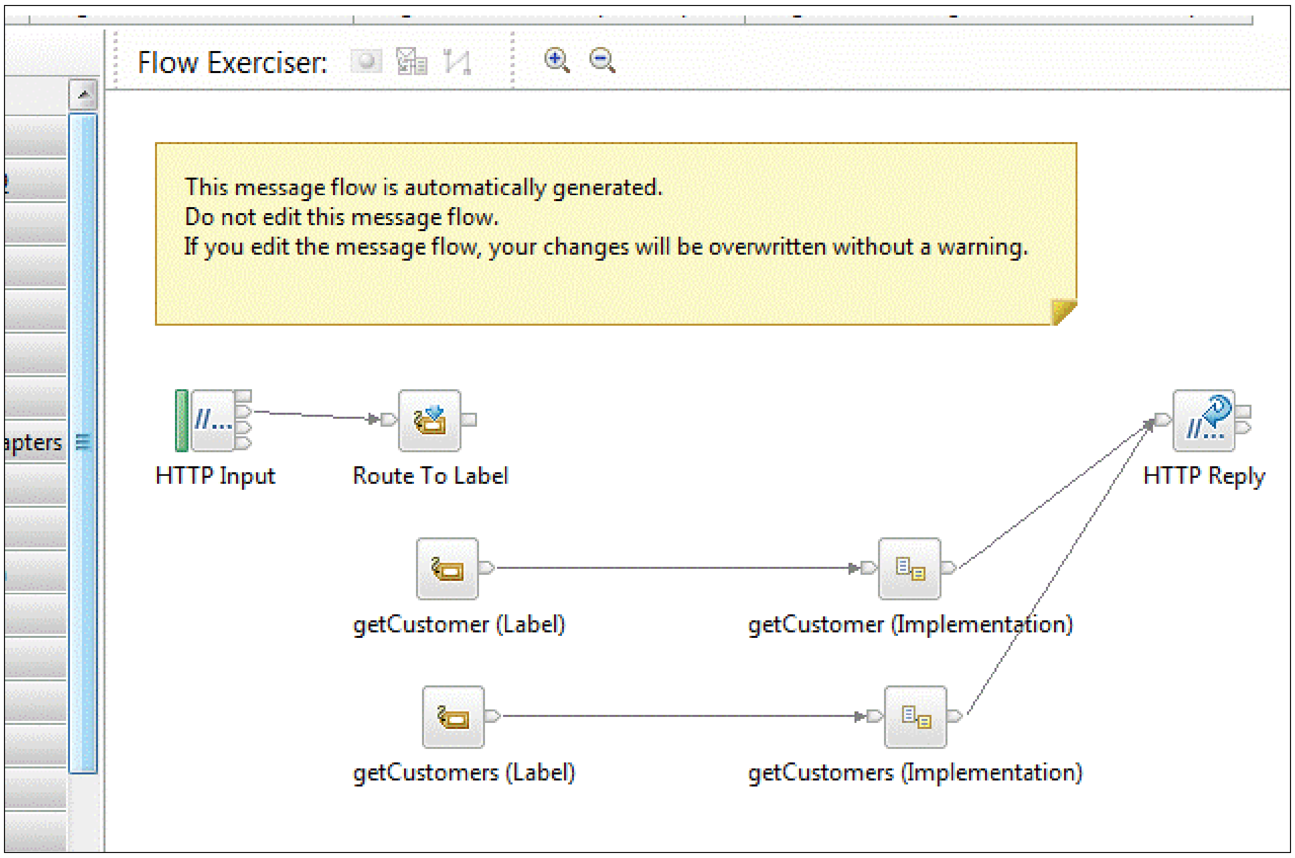
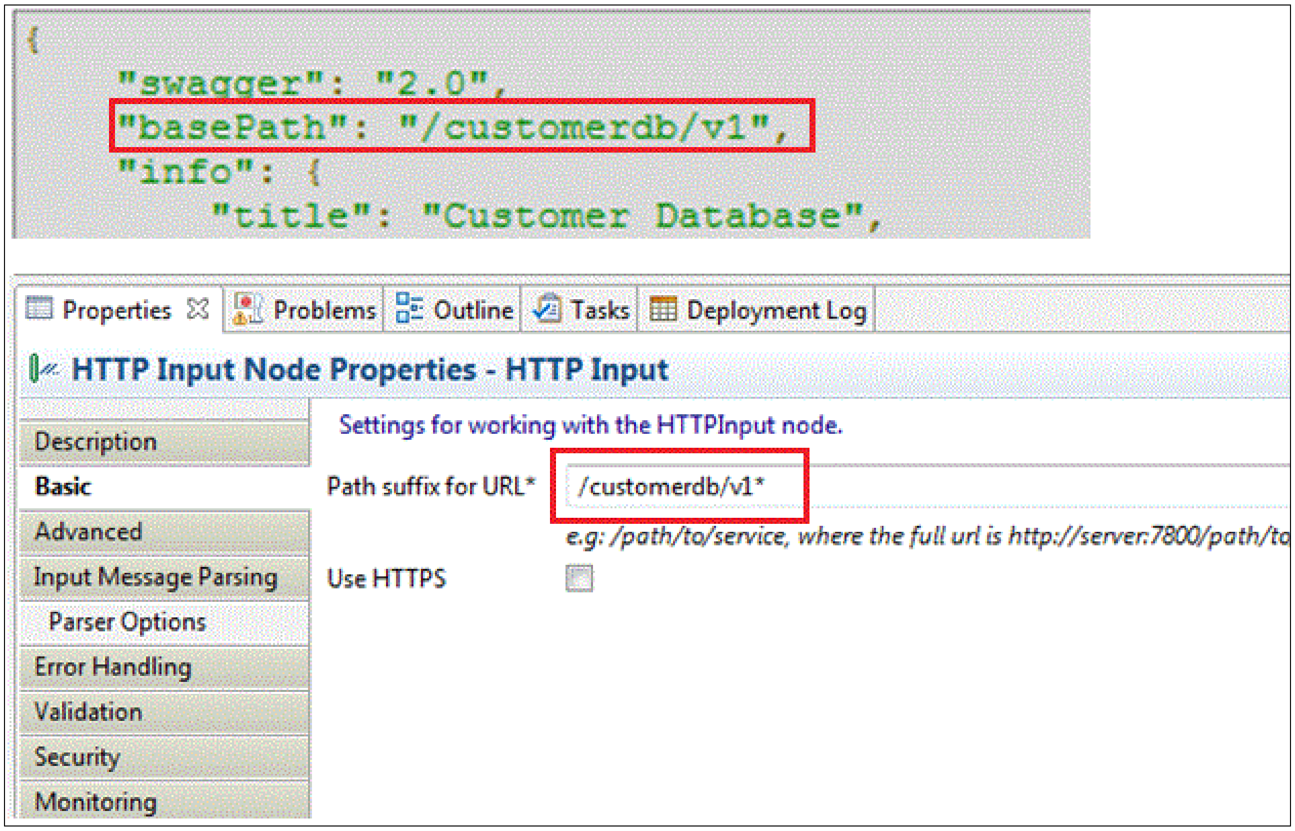

This message flow is automatically generated.
Do not edit this message flow.
If you edit the message flow, your changes will be overwritten without a warning.

Figure 3-20 Main message flow with implemented subflows

10. The node properties on the HttpInput node are automatically populated as shown in Figure 3-21 with the relative path as defined in the Swagger document.



```
{
    "swagger": "2.0",
    "basePath": "/customerdb/v1",
    "info": {
        "title": "Customer Database",
```

Settings for working with the HTTPInput node.

Path suffix for URL* /customerdb/v1*

e.g: /path/to/service, where the full url is http://server:7800/path/to

Use HTTPS

Figure 3-21 HttpInput node properties and Swagger details

11. Package your REST API into a BAR file and deploy it to an integration server. The deployed artifacts are seen in Figure 3-22.

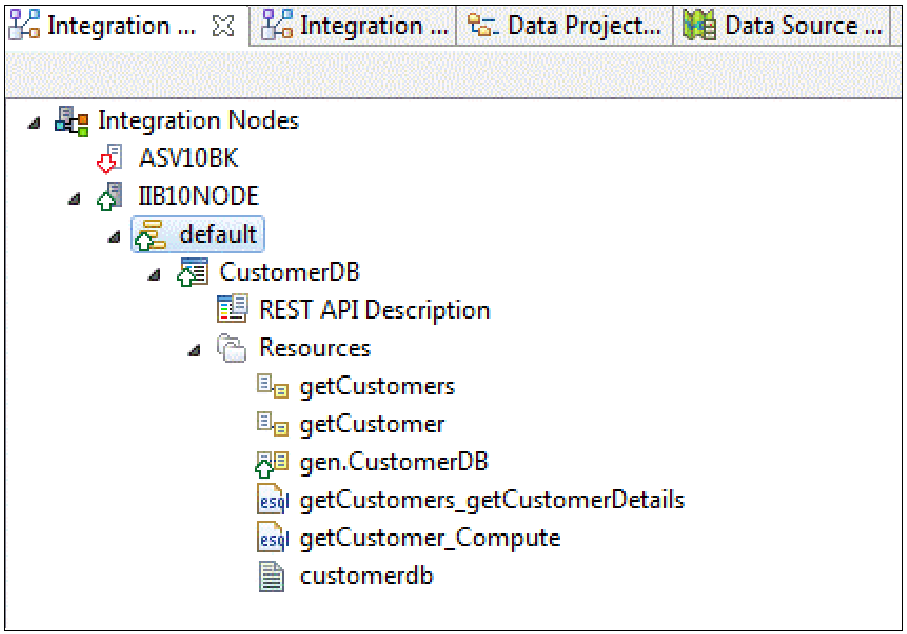


Figure 3-22 Integration server with deployed REST API project

Note: REST APIs can be deployed to integration servers that are configured to use the integration server HTTP listener only. REST APIs cannot be deployed to integration servers that are configured to use the integration node HTTP listener.

If your integration node is configured to use the integration node HTTP listener, you must do the following steps:

- Run the following command:

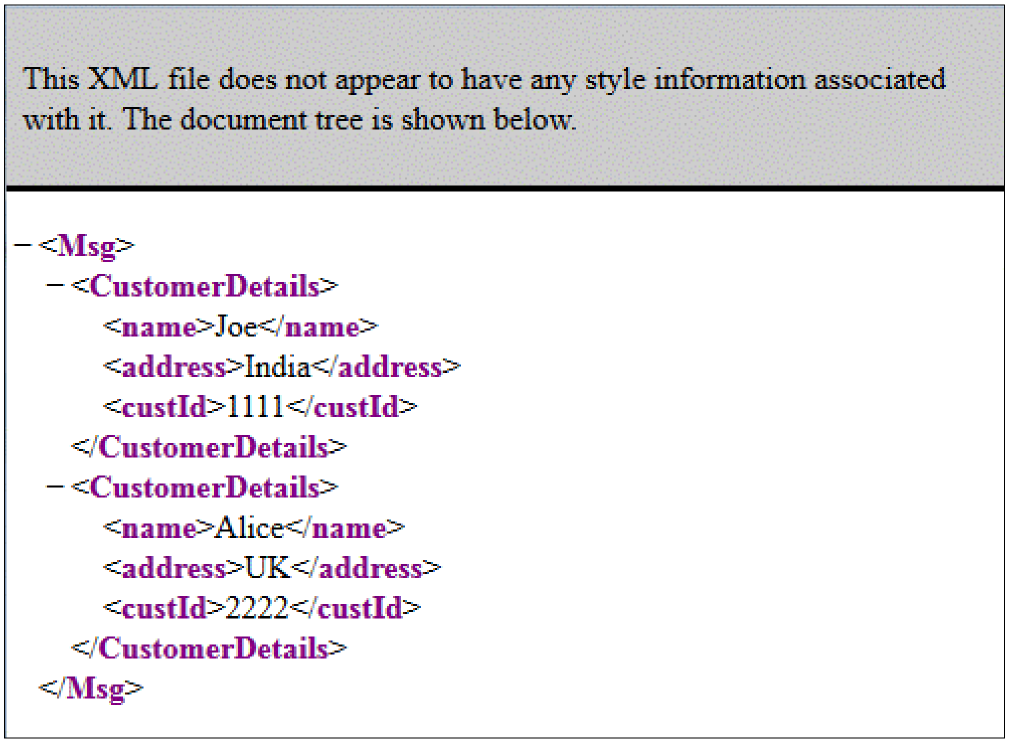
```
mqsichangeproperties integrationNodeName -e integrationServerName -o
ExecutionGroup -n httpNodesUseEmbeddedListener -v true
```

- Restart the integration server.

12. You can test your message flow API by invoking it via a web browser. For example:

```
http://localhost:7800/customerdb/v1/customers
```

This produces the output depending on the operations implemented in the message flow. In the example described here, it brings the information about all the customers from the customer database as shown in Figure 3-23.



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
– <Msg>
  – <CustomerDetails>
      <name>Joe</name>
      <address>India</address>
      <custId>1111</custId>
    </CustomerDetails>
  – <CustomerDetails>
      <name>Alice</name>
      <address>UK</address>
      <custId>2222</custId>
    </CustomerDetails>
  </Msg>
```

Figure 3-23 Output from the REST API invocation

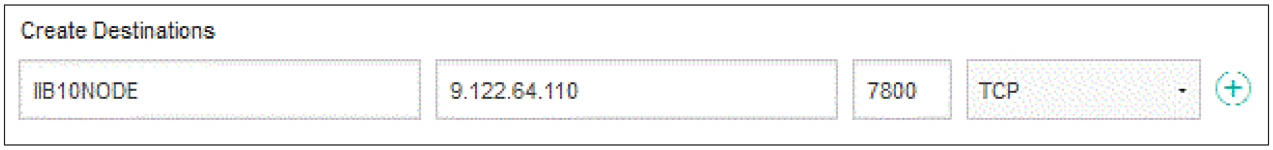
Connecting securely from Bluemix application to message flow

In the preceding section, you learned about creating a message flow as REST API. We now demonstrate how to make this message flow available to Bluemix via Secure Gateway so that applications running in Bluemix can invoke the message flow as a REST call when the application in cloud needs to work on the data that resides in the private enterprise network.

As explained in 6.11, “Implementing subscenario 4” on page 163 after you create the Secure Gateway between your Bluemix environment and on-premises enterprise system, the next task is to create a destination endpoint that points to your IBM Integration Bus node where the message flows are deployed.

Follow these steps to set up the destination in Secure Gateway:

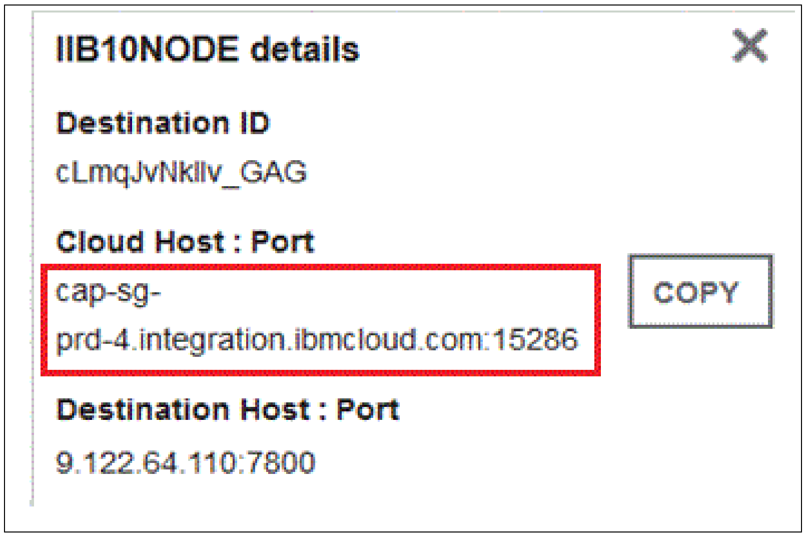
1. Provide a name to the destination, add a host name or IP address where the Integration Bus node is deployed on your enterprise system, and specify the port number on which the HttpInput node of your REST API message flow is listening on. For example, if the IP address of the host is 9.122.64.110 and the HttpInput node of REST API message flow is listening on port 7800, add the details as shown in Figure 3-24.


Create Destinations

IIB10NODE	9.122.64.110	7800	TCP

Figure 3-24 Add destination to the on-premises Integration node

2. The Secure Gateway now generates a public URL as shown in Figure 3-25.


IIB10NODE details

Destination ID

cLmqJvNkllv_GAG

Cloud Host : Port

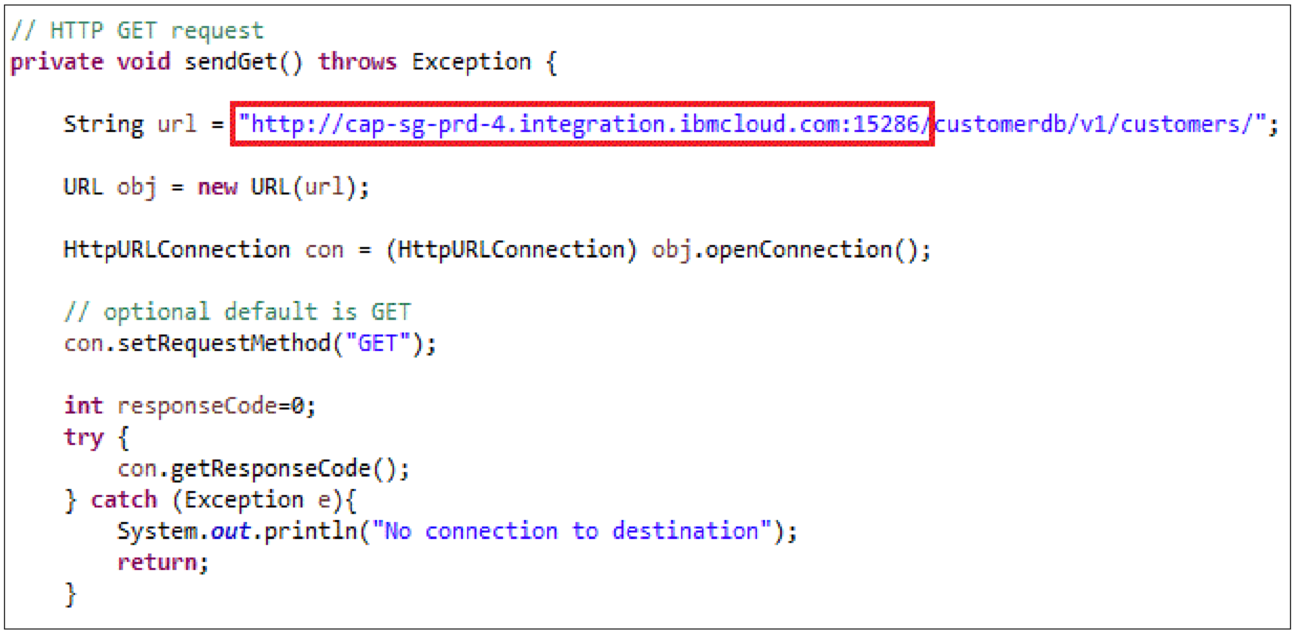
cap-sg-prd-4.integration.ibmcloud.com:15286

Destination Host : Port

9.122.64.110:7800

Figure 3-25 Public URL for on-premises REST API message flow

3. Now you can use this Secure Gateway generated cloud Host:Port pair in the applications that you develop in the Bluemix cloud environment as shown in Figure 3-26.



```
// HTTP GET request
private void sendGet() throws Exception {

    String url = "http://cap-sg-prd-4.integration.ibmcloud.com:15286/customerdb/v1/customers/";

    URL obj = new URL(url);

    HttpURLConnection con = (HttpURLConnection) obj.openConnection();

    // optional default is GET
    con.setRequestMethod("GET");

    int responseCode=0;
    try {
        con.getResponseCode();
    } catch (Exception e){
        System.out.println("No connection to destination");
        return;
    }
```

Figure 3-26 Sample web application showing use of cloud Host:Port

3.6 Introduction to IBM MessageSight

IBM MessageSight is a 2U appliance-based messaging server that is optimized to address the massive scale requirements of the machine-to-machine (M2M) and mobile use cases. It is designed to handle many connected clients and devices with the ability to process a high volume of messages securely with consistent latency.

This section provides an overview about the following key features of IBM MessageSight and how the appliance performs on each:

- Architecture overview
- Scalability and performance
- Reliability
- Security
- Integration ability
- Developer-friendly

3.6.1 Architecture overview

IBM MessageSight is designed to reside at the edge of the enterprise and can extend the existing messaging infrastructure as well as can be used as a stand-alone messaging server. It is scalable to deliver a large amount of data to analytic engines and other types of big data applications. Figure 3-27 depicts the architecture overview of IBM MessageSight, which depicts the deployment context and how the appliance can help to instrument the communication among various endpoints.

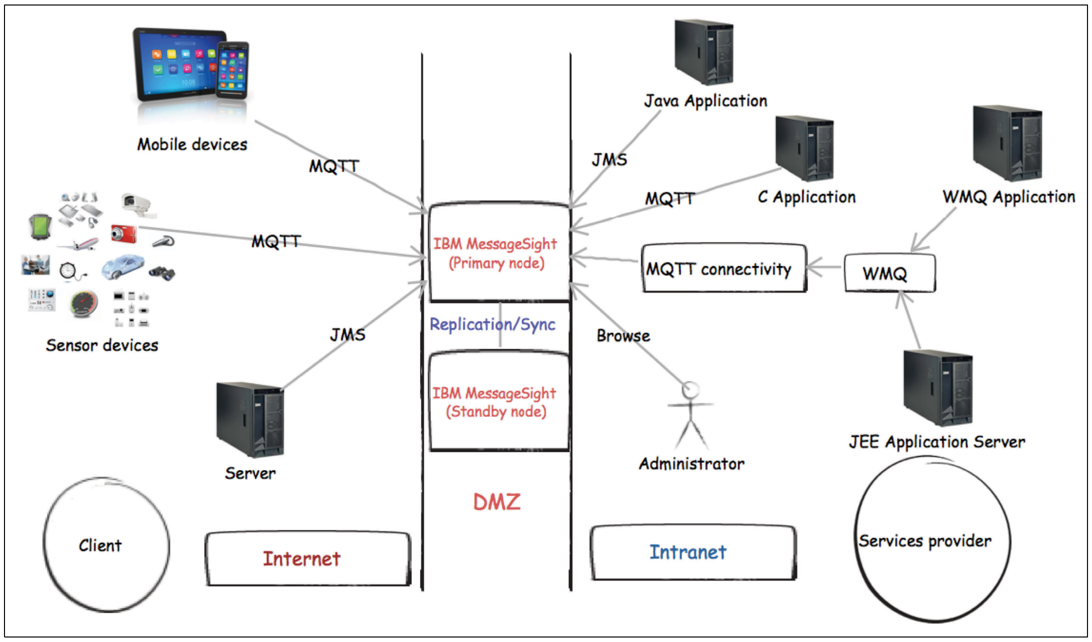


Figure 3-27 IBM MessageSight architecture overview

3.6.2 Scalability and performance

Built upon the MQ Telemetry Transport (MQTT) messaging protocol, which is faster and requires less bandwidth and power than traditional HTTPs, IBM MessageSight is well-suited with tags and sensors for mobile devices and other “things” that typically have low power and low communication bandwidth capabilities. Also, the high-scale, asynchronous publish/subscribe with event-oriented paradigm could provide responsive interaction, which turns into a better user experience and better scalability.

One MessageSight appliance can serve up to a million devices connected concurrently and handle the throughput of up to 13 million nonpersistent messages per second with predictable latency in microseconds under load. That is an impressive number.

3.6.3 Reliability

If high availability (HA) and disaster recovery ability are required, two MessageSight appliances could be easily configured to an HA-enabled mode to act as an HA pair of nodes, one to be the primary node (the appliance that is processing messages) and the other to be the standby node (the appliance to which the primary node is replicated). With HA enabled, the messaging services can withstand an outage of an appliance and continue to provide messaging services.

At the message delivery level, because MessageSight fully supports MQTT protocol, it consequently supports three qualities of service options for delivering messages between clients and servers. This allows the delivery assurance of a particular message to be flexibly achieved as needed.

3.6.4 Security

There are three main aspects to security in MessageSight: Transport level security, authentication, and authorization.

MessageSight controls transport level security and authentication settings using a security profile associated with an endpoint to define the security operations applied to a message flow. Besides protecting the message content being transferred, configuring the appropriate transport level also helps to avoid sending authentication credentials that are not encrypted.

The authentication supports both local user stores and external Lightweight Directory Access Protocol (LDAP) servers, which give users more flexibility in building up a security plan.

IBM MessageSight implements a policy-based authorization mechanism to allow clients to connect and use messaging actions (connection and messaging policies). A modern policy-based security approach is composed of a cohesive set of different policies, which helps MessageSight to efficiently achieve security compliance.

Supporting Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols that only run signed, encrypted firmware images provided by IBM, encrypted flash storage media makes MessageSight secure enough to be used for DMZ environments and placed at the edge of an enterprise to interface with the external world.

3.6.5 Integration ability

As a full-featured messaging appliance, the ability to integrate with other systems is a key feature of IBM MessageSight.

By supporting the well-known messaging protocols MQTT (MQTT over Transmission Control Protocol/Internet Protocol (TCP/IP), MQTT over WebSockets) and Java Message Service (JMS), the appliance is well-suited with both publish/subscribe (topic-based) and point-to-point (queue-based) messaging models and can be widely integrated with other systems like Java-based systems, rich HTML5-based applications, and so on.

MessageSight can extend and connect to WebSphere MQ infrastructures and supports connectivity to IBM Integration Bus by using MQ connectivity. Options appropriate for this are to use multiple queue managers to handle the messages or preserve the message order by using a single queue manager.

3.6.6 Developer-friendly

Built on the open MQTT, MessageSight supports MQTT client applications and libraries for various platforms:

- MQTT over TCP/IP: MQTT C client; MQTT client for Java, Android, and iOS.
- MQTT over WebSockets: MQTT client for JavaScript.
- JMS: IBM MessageSight JMS client.
- PhoneGap MQTT plug-ins with JavaScript application programming interface (API) for use with IBM Worklight®, Apache Cordova, and Adobe PhoneGap.

Part 2

Introduction to hybrid cloud patterns for event integration

In this part, we describe three of the common hybrid cloud patterns for event integration.

The following chapter is included in Part 2:

4 Introduction to hybrid cloud patterns for event integration

In this chapter, we introduce the patterns that can be used to integrate the events on the cloud with events occurring in the enterprise data center.

This chapter covers the following patterns:

- Patterns to provide secure connectivity between cloud and on-premises applications using Secure Gateway
- Pattern to show how a Representational State Transfer (REST) application programming interface (API) from an in-house application can be managed via API Management
- Pattern for API composition for hybrid cloud integration
- Pattern for integrating an Internet of Things (IoT) device to an on-premises asset management system
- Pattern for data analytics integration with real-time events

This chapter includes the following sections:

4.1 Events in a hybrid cloud environment

An event occurring on the cloud might need to be sent to the in-house applications in a secure fashion. An event in the cloud can take many forms, for example:

- A born-on-the cloud application capturing an event and sending data to an in-house application for processing
- Signals from a connected device sending notifications for an urgent action
- An external cloud application sending business data to an in-house application
  - Synchronizing data from a master to subordinate application
  - Transferring data, such as a purchase or sales order

Conversely, an event occurring in an in-house application might need to be exposed to the cloud in a secure fashion. An event in the in-house application can take many forms, for example:

- A born-on-the-cloud application publishing results from in-house processing
- Control events being sent to smart connected devices
- An external cloud application receiving business data from an in-house application
  - Synchronizing data from a master to subordinate application
  - Transferring data, such as a purchase or sales order

4.2 Pattern to provide secure connectivity from cloud to on-premises application using Secure Gateway

The enterprises in the current environment are reaping the benefits of very rapid innovations that are being driven by born-on-the-cloud applications. The born-on-the-cloud applications provide the opportunity to quickly develop creative solutions that use modern and evolving channels to conduct business. This is resulting in emerging organizations that challenge the market share of established businesses and business models.

An established business usually has a large inventory of core applications to support the business. Some of these applications might have been highly optimized over a period and provide the business with an edge over its competitors. These applications are traditionally in the corporate data center.

An established business needs to embrace the new channels offered by born-on-the-cloud applications to retain and grow its competitive position. In addition, it also needs to leverage the capabilities of its established applications. The enterprise has the option of implementing a hybrid cloud solution that integrates a born-on-the-web application with its in-house established applications.

A new organization with some creative ideas might want to add more rigor to their born-on-the-cloud application by supplementing its capabilities using in-house applications. For example, certain jurisdictions might have strict standards as to how and where client information can be legally stored. An in-house application could be built using an enterprise-grade database and middleware technologies. In this case, a born-on-the-cloud application might evolve into a hybrid cloud solution.

A popular choice for integrating applications on disparate systems is the use of message-oriented middleware to decouple the components. This approach can be used for hybrid cloud solutions. The hybrid cloud solutions, however, create a new challenge: How to securely connect in-house enterprise applications with a born-on-the-cloud application. The security challenge can be addressed by using the IBM Secure Gateway.

4.2.1 Using IBM Secure Gateway to connect a cloud application with an in-house application

The IBM Secure Gateway service on IBM Bluemix allows a secure connection between a cloud application and an in-house application and servers to be established.

Secure Gateway from IBM Cloud to a corporate data center

A pattern for connection is shown in Figure 4-1.

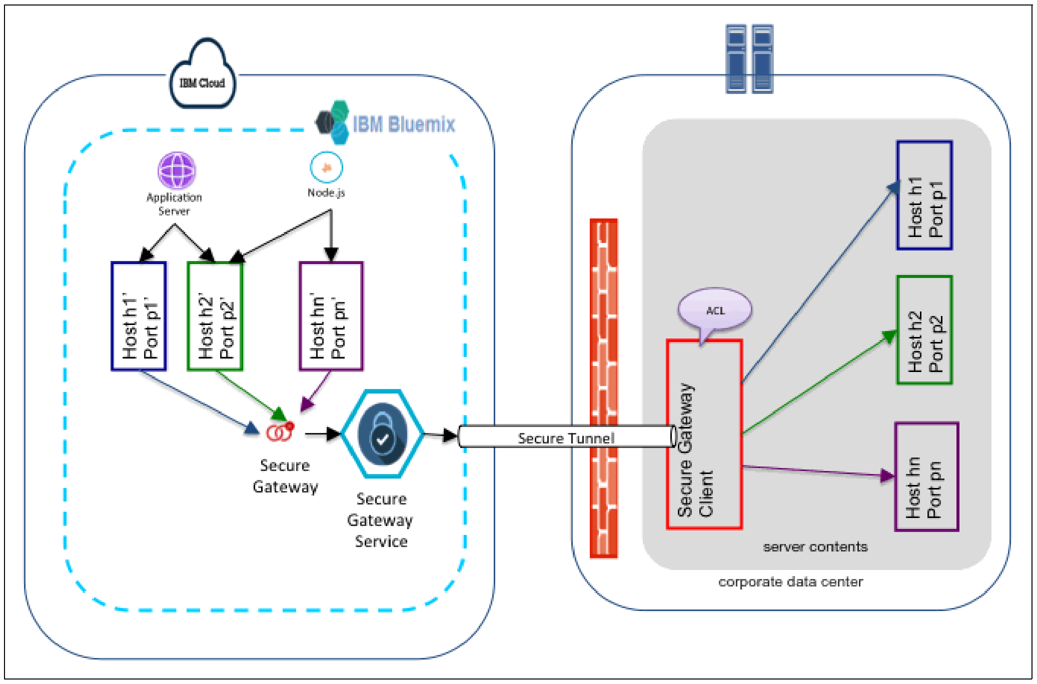


Figure 4-1 Secure Gateway from IBM Cloud to a corporate data center

The IBM Bluemix Secure Gateway service facilitates the creation of a secure tunnel to the corporate data center. Details about the IBM Secure Gateway can be found in 2.1, “Secure Gateway” on page 10.

The administrator defines a Secure Gateway for the Secure Gateway Bluemix service. A Secure Gateway has a unique “*<Gateway Id>*”:

- For a Secure Gateway, the administrator defines the following details for each host inside the corporate data center:
  - Host name or TCP/IP address inside the corporate data center, for example Host h1
  - Port number of the server inside the corporate data center, for example Port p1

- The Secure Gateway assigns the following details for each host:
  - Cloud host name, for example Host h1’
  - Cloud port number, for example Port p1’
- An application that was designed to use Host h1 and Port p1 inside the corporate data center would be configured to use Host h1’ and Port p1’ when it is running in the cloud.

The Secure Gateway client runs inside the corporate data center. The Secure Gateway client uses a local access control list (ACL). The ACL defines the host, port, or host + port combinations inside the corporate data center that can be accessed by the Secure Gateway client. The Secure Gateway client uses the unique “*<Gateway Id>*” to connect to the Secure Gateway.

Example flow of a request for data using Secure Gateway

The example in Figure 4-1 on page 67 shows a Node.js application that requires access to the following servers in the corporate data center. It is assumed that the Secure Gateway client has already been started inside the corporate firewall and the ACL has been correctly configured for the following hosts and ports:

- Host h2, Port p2
- Host hn, Port pn

The Node.js application is configured to use the following hosts and ports on the cloud:

- Host h2’, Port p2’ for Host h2, Port p2 inside the corporate data center
- Host hn’, Port pn’ for Host hn, Port pn inside the corporate data center

Following is the logical flow of control when accessing Host hn’ Port pn’ (refer to Figure 4-1 on page 67):

1. Request from Node.js is sent to Host hn’ and Port pn’.
2. The request is forwarded to IBM Secure Gateway.
3. The request travels through the secure tunnel to the Secure Gateway client.
4. The Secure Gateway client validates the request against the ACL.
5. If the access has been granted, the request is forwarded to Host hn, Port pn.

Secure Gateway from IBM and a third-party cloud to corporate data center

A pattern for connection is shown in Figure 4-2.

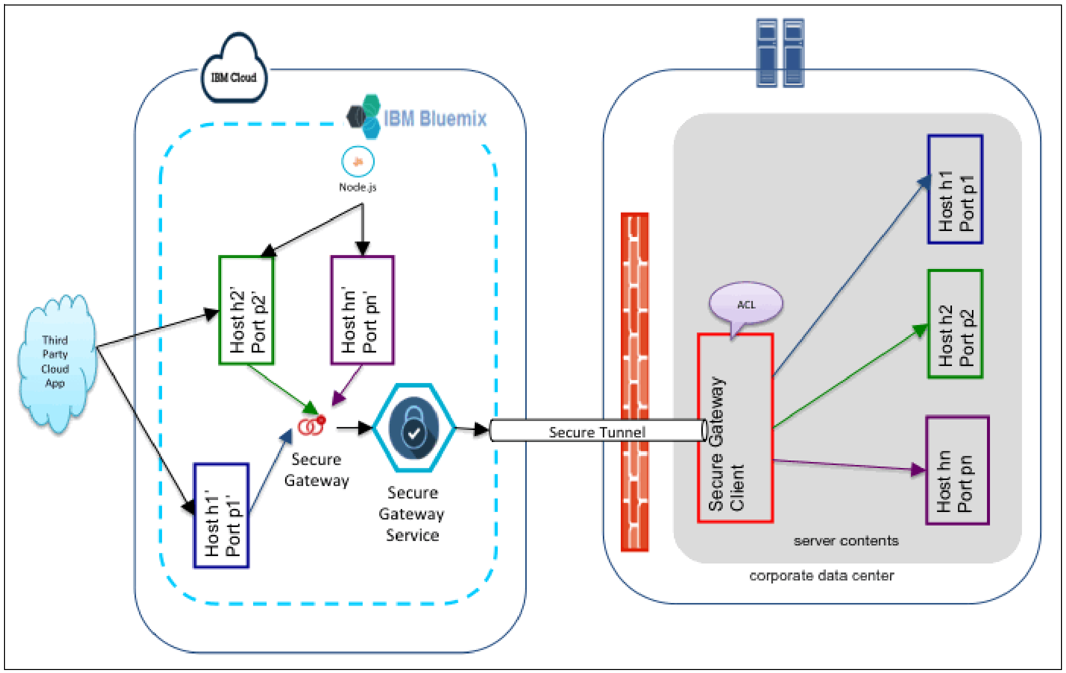


Figure 4-2 Secure Gateway from IBM and a third-party cloud to corporate data center

The third-party cloud application uses configuration that is similar to one used by an application running in the IBM Cloud. The connection between IBM Cloud to a corporate data center works the same as the previous example. The only addition is that the third-party cloud application is also able to access the servers inside the corporate data center. The third-party cloud applications need to be configured the same way as the Bluemix application. In Figure 4-2, the cloud application uses the following hosts and ports:

- Host h2’, Port p2’ for Host h2, Port p2 inside the corporate data center.
- Host h1’, Port p1’ for Host h1, Port p1 inside the corporate data center.

Secure Gateway from IBM and third-party cloud to multiple corporate data centers

A pattern for connection is shown in Figure 4-3.

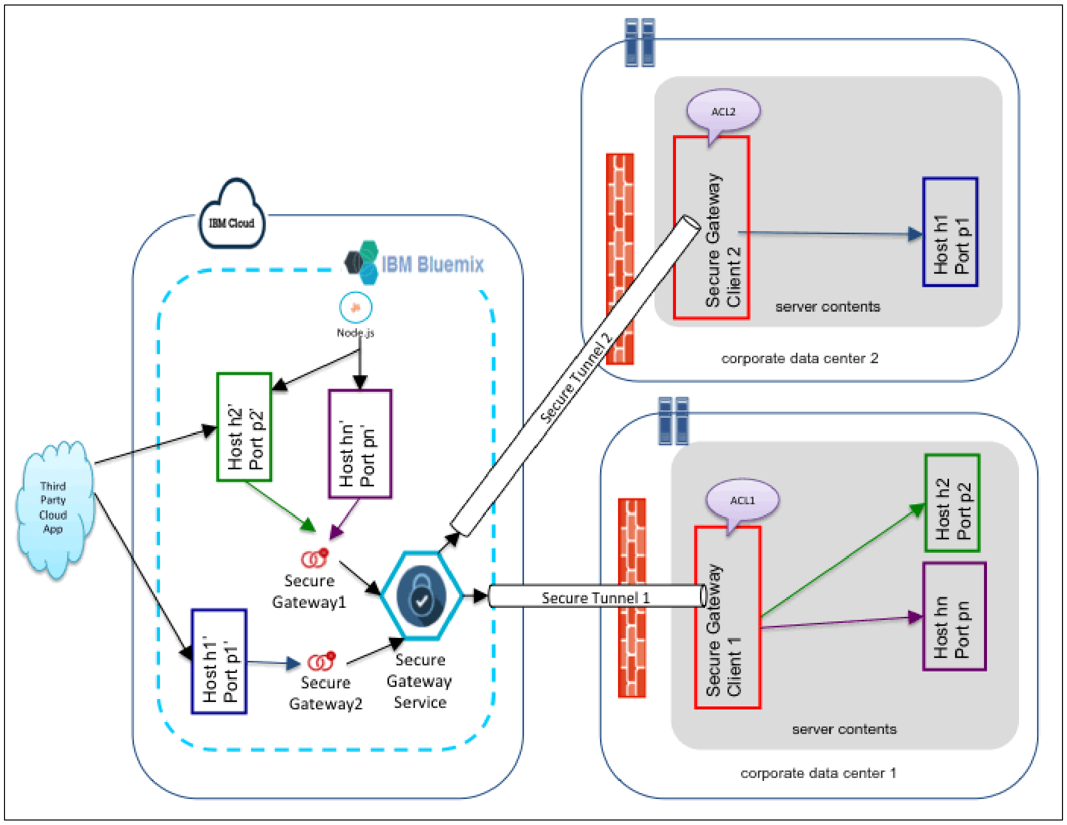


Figure 4-3 Secure Gateway from IBM and a third-party cloud to multiple corporate data centers

The Secure Gateway service can support multiple Secure Gateways. Each Secure Gateway can be configured to support one or more destinations. This allows for flexible deployment topologies. The cloud applications that run in an IBM or third-party cloud can transparently get secure access to the servers running inside multiple data centers.

Following are the differences from the previous pattern:

- There will be two "<*Gateway Id*>": One for each Secure Gateway:
  - Secure Gateway client 1 will use the "<*Gateway Id*>" for Secure Gateway 1
  - Secure Gateway client 2 will use the "<*Gateway Id*>" for Secure Gateway 2
- Secure Gateway 1:
  - Contains the definition for host h2 port p2 and exposes host h2' port p2'
  - Contains the definition for host hn port pn and exposes host hn' port pn'
- ACL 1 will contain details about allowing access to the following hosts and ports:
  - host h2 port p2
  - host hn port pn
- Secure tunnel 1 will provide secure connectivity from Secure Gateway 1 to Secure Gateway client 1.

- Secure Gateway 2:

  Contains the definition for host h1 port p1 and expose host h1' port p1'
- ACL 2 will contain details about allowing access to the following host and port:

  host h1 port p1
- Secure tunnel 2 will provide secure connectivity from Secure Gateway 2 to Secure Gateway client 2.

Secure Gateway from IBM and a third-party cloud to multiple corporate data centers example

An example is shown in Figure 4-4.

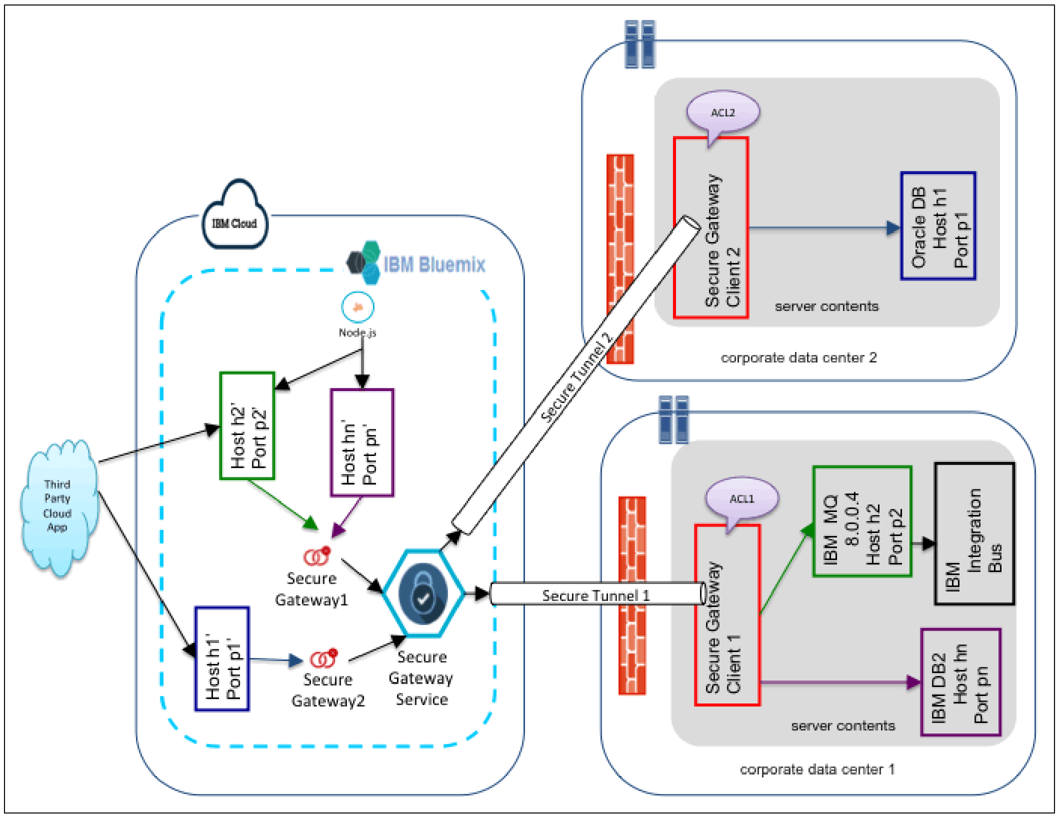


Figure 4-4 Secure Gateway from IBM and third-party cloud to multiple corporate data centers example

In the example shown in Figure 4-4, a Node.js application connects to a DB2 database and allows users to submit their application for allocation of land in a highly sought-after location. Each application is sent as a message to the IBM MQ 8.0.0.4 in data center 1.

Each message received by IBM MQ 8.0.0.4 is processed by the IBM Integration Bus in data center 1. The results of the allocation are published by the IBM Integration Bus on to IBM MQ.

A cloud application running on a third-party cloud is listening for messages on IBM MQ in data center 1. When results are published, it records the information in an Oracle database in data center 2.

4.2.2 Conclusion

In the current environment, innovation is being fueled by born-on-the-cloud applications. The business needs to leverage their applications in the corporate data center to drive the full benefits from the innovation. The Secure Gateway facilitates this connectivity. In this section, we described a number of patterns for securely connecting applications in one or more corporate data centers to the applications in IBM and third-party cloud. This capability provides an enterprise with an opportunity to flexibly manage and grow their business solutions as their business needs evolve.

4.3 Pattern to show how IBM Integration Bus flow exposed as REST API can be managed by API management

The IBM Integration Bus is frequently used as the enterprise service bus. More details about the IBM Integration Bus can be found in 3.5, “Introduction to IBM Integration Bus” on page 47. It generally reads messages that have been delivered to IBM MQ and processes the message using a flow. At the completion of a flow, the response is sent to IBM MQ.

IBM Integration Bus can also expose a flow as a REST service. In this case, a REST request is sent by a client to the Integration Bus. It processes the request using a flow. The response is sent as results from a REST call to the client.

The REST service may be consumed by applications inside the corporate data center. The REST services can be exposed to a cloud application running on IBM cloud or third-party cloud using the IBM Secure Gateway and API Management Bluemix services. For information about this service, see the Redbooks publication *Hybrid Cloud Data and API Integration: Integrate Your Enterprise and Cloud with Bluemix Integration Services*, SG24-8277.

4.3.1 Implementing the pattern

The previous section discussed the connectivity between cloud and a corporate data center using the IBM Secure Gateway. This section builds on that information and focuses on the use of API management to expose the service. A pattern for connection is shown in Figure 4-5 on page 73.

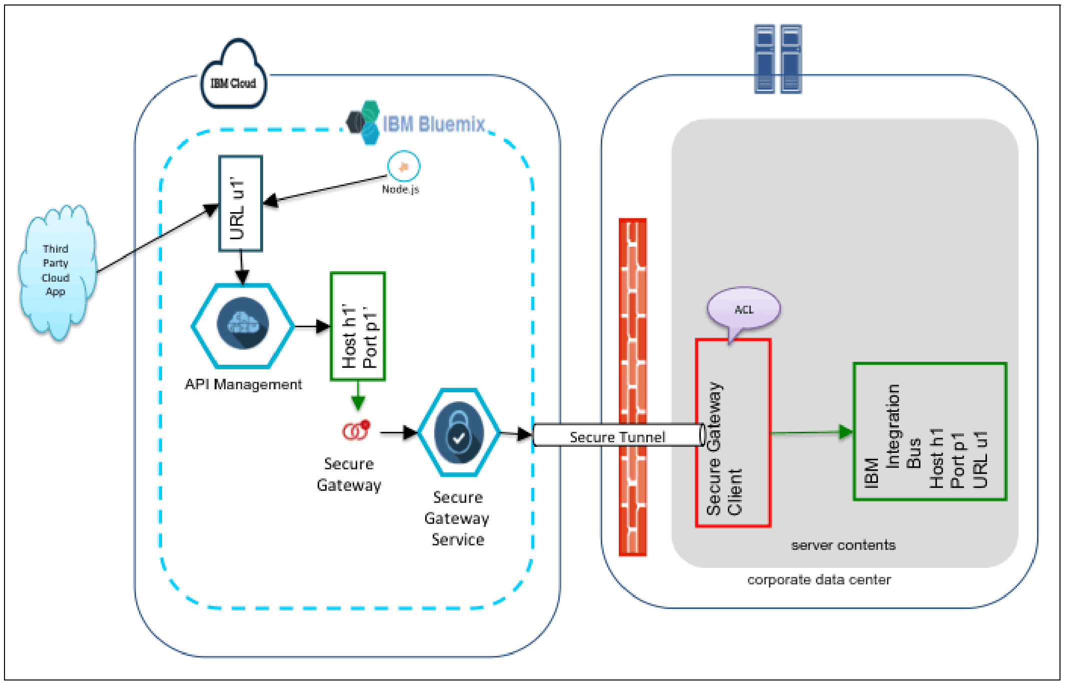


Figure 4-5 Exposing REST service on IBM Integration Bus to cloud applications

The IBM Integration Bus can expose a number of REST API services. Some of these may be for consumption by applications running in the corporate data center. A subset might be made available as REST services that are available to applications running on the cloud. The API management service on IBM Bluemix facilitates the management of REST service end points.

In the diagram shown in Figure 4-5:

- The IBM Integration Bus:
  - Exposes REST API on host h1 and port p1 and url u1 inside the corporate data center.
  - Applications inside the enterprise might invoke the REST API using these details.
- The Secure Gateway:
  - Maps host h1 as h1' and port p1 as p1'.
  - The details are as shown in the previous section.
- The API Management Bluemix service:
  - Connects to IBM Integration Bus using the Secure Gateway.
  - Maps the URL u1 on IBM Integration Bus to URL u1' on the cloud.
- The actual URL u1 for the service is not exposed to the cloud application.
- An application running inside the corporate data center would invoke the same REST service using URL u1.
- If the same application was running on IBM or a third-party cloud, it would use the URL u1'.

4.3.2 Conclusion

The IBM Integration Bus is a popular enterprise service bus. It facilitates development of flows to process messages. A flow can be exposed as a REST API. A REST API can be used internally inside the corporate data center. If a cloud application needs the REST API, it can be securely exposed to a cloud application running in the IBM or third-party cloud.

4.4 Pattern for hybrid cloud integration using API facade

A common challenge that organizations are facing is exposing existing backend systems. These systems are usually critical for organizations as they are running key aspects of their business and are fairly stable given that they have been hardened over time. However, these monolithic systems can be complex with many dependencies, which make them difficult to change and adapt quickly.

In our fast-moving world, to deliver new services and create new business opportunities, organizations need to be able to expose these systems (or a subset of functionalities) easily and quickly. There is not only one approach to do so but multiple. Depending on the requirements and objectives, organizations might choose to go through a specific path or implement multiple integration strategies.

In this section, we look at a specific approach that consists of adopting an API facade pattern to expose an existing operational system in the context of hybrid cloud.

4.4.1 Characteristics of the pattern

The primary goal of the API facade pattern is to securely expose applications to internal or external consumers through clean and simplified interfaces. It acts as a virtual interface between your backend systems and consumer applications. It can play a role of mediation to access your backend as well as altering or transforming inbound and outbound messages.

As an example, consumers might look for secured RESTful interfaces delivering content in JSON where this maps in the background to an SQL operational database within the enterprise.

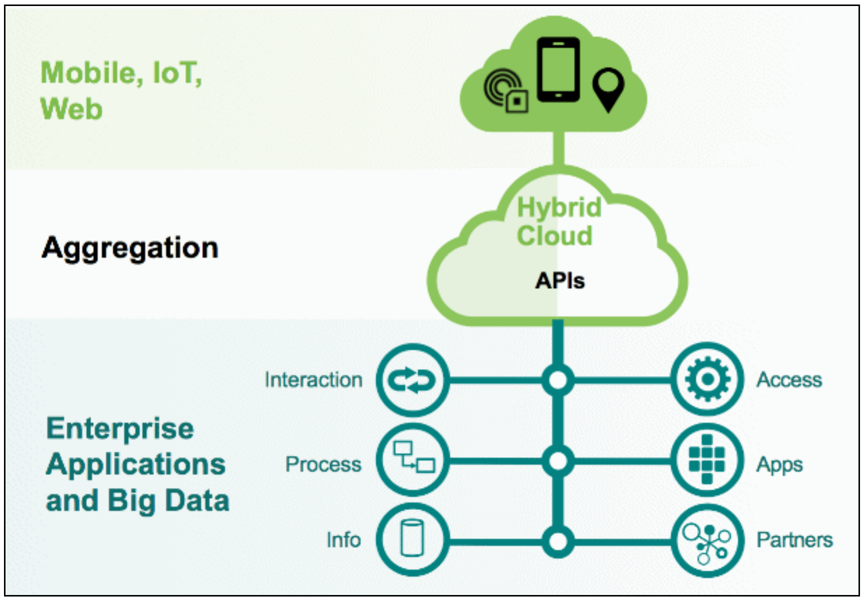


Figure 4-6 API facade pattern for hybrid cloud

From an architectural standpoint, the API facade pattern is made of interfaces that clearly define what happens when you interact with them. These interfaces can have specific characteristics, such as security, operations, and quality of service. Requests coming through these interfaces go through a mediation layer for routing, transformation, and protocol switching in order to integrate with backend systems.

From a functional perspective, this pattern can be decomposed into two major logical layers (see Figure 4-7):

- Mediation

  This is where you define and implement your APIs. It includes connecting to your backend systems leveraging adapters or other middleware. You also specify the business logic to process messages between consumers and providers. You are able to compose your APIs across multiple backends and expose simple and well-defined interfaces to consumer applications.

- Gateway

  This is where you enforce API security and control. It provides secure transport of data, request authentication and authorization, service level agreements enforcement, activity and performance management, load balancing, and caching.

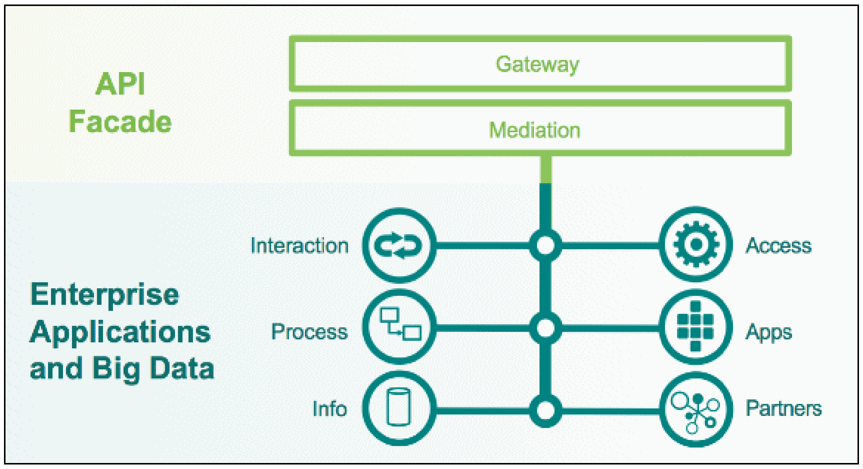


Figure 4-7 API facade logical layers

Note: To complete the overall picture, you can also introduce a third layer for API management. It can be used for API discovery, subscription management, lifecycle management and governance, monitoring, and analytics.

By using this pattern, communication is mediated through the API facade, which encapsulates the backend applications' structure. The consumer application code is simplified as all the heavy lifting is done at the middleware level. The client just has to consume the API interface that has been exposed. It also reduces the number of interactions between the consumer's application and backend systems as the service composition is done in the mediation layer.

From an implementation perspective, there are many ways to implement this pattern. Typically, the choice of implementation and technology to use depends on multiple factors, such as:

- Backend systems to expose

  Depending on the complexity and type of applications that you are trying to integrate with, you have to choose a middleware solution that is better suited for the backend integration. For example, you might have a database as a backend and consequently use a middleware that provides a database adapter that is ready for immediate use to quickly connect and expose data.

- Existing technology and skills

  You might have an existing enterprise service bus (for example, IBM Integration Bus) already in place to integrate legacy systems. Consequently, you might want to reuse it as your primary middleware layer to expose the backend. Existing skills are also important to choose the right technology because you want to leverage your pool of talents to create your API facade.

- Deployment options

  You might want to keep the integration layer close to your legacy systems or decide to leverage a cloud-based model for more flexibility and easier scalability.

- Business requirements

  You do not create APIs just for the sake of it but because it is driven by business requirements. These requirements might impact your choice of technology and approach.

In addition to that, you also have to consider usability, maintainability, flexibility, and much more. So in summary, there is no silver bullet to implement this pattern because it depends on a combination of factors.

In the next section, we look at one way to implement an API facade using the StrongLoop platform.

4.4.2 Pattern implementation with StrongLoop

In the case where developers need APIs to integrate new and legacy data to quickly deliver engaging app experiences as well as leveraging the robust Node.js ecosystem to speed up and simplify app creation, StrongLoop appears to be an ideal candidate.

StrongLoop, an IBM company provides an end-to-end platform for the full API lifecycle that allows you to visually develop APIs in Node and get them connected to new and legacy data. It provides the required capabilities to implement the API facade effectively.

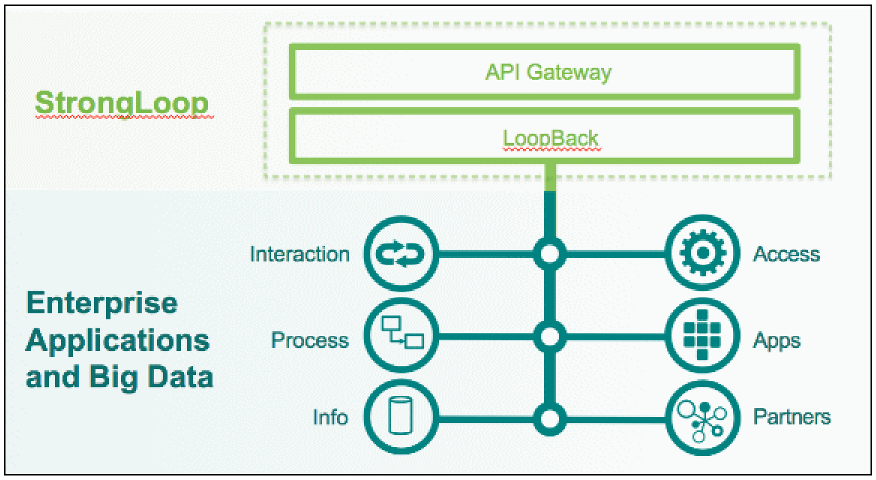


Figure 4-8 StrongLoop for API facade

The platform is built on top of the open source LoopBack framework for Node.js. LoopBack represents the mediation layer in the API facade pattern where you can create and compose scalable REST APIs.

LoopBack generalizes backend services such as databases, REST APIs, SOAP web services, and storage services as data sources. Data sources are backed by connectors that communicate directly with the database or other backend systems. On top of that, you can define models that represent backend data sources. When you define a model, it automatically comes with a predefined REST API with a full set of create, read, update, and delete operations.

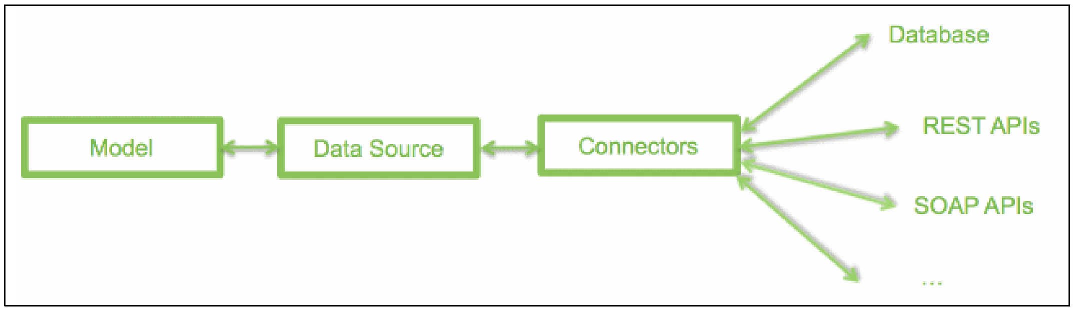


Figure 4-9 LoopBack concepts

By leveraging these capabilities, you can quickly connect to an operational database and generate a REST API to expose it. Additionally, you can add application logic in JavaScript to handle and process messages, create new routes, apply some filtering, as well as validate data.

To complete the API facade pattern, StrongLoop also provides an API Gateway that can be used to externalize, secure, and manages APIs. The gateway can act as both provider and delegator to authentication, authorization, and auditing (AAA) sources within the enterprise as the first intercept to establish identity. It can also perform infrastructure-level API consumption functions required by the client, such as pagination, throttling, caching, delivery guarantee, firewall, and so on. You can also use it to instrument APIs to fulfill service level agreements through the monitoring of APIs and also injects metadata to report on API usage, health, and other metrics. Another capability of the gateway is to compose coarse-grain APIs from fine-grained micro-APIs and can act as a reverse proxy to virtualize API endpoints.

StrongLoop provides the key capabilities to implement the API facade pattern. It can be deployed on-premises if needed because it might better fit the overall integration strategy of an organization. It can also run in the cloud for more flexibility and agility, such as on IBM Bluemix. In that case, you have to consider to use the IBM Secure Gateway to access your on-premises backend systems from Bluemix as described in 4.2, “Pattern to provide secure connectivity from cloud to on-premises application using Secure Gateway” on page 66.

Note: For more information about StrongLoop and the LoopBack framework, see the following URL:

`https://docs.strongloop.com/display/public/LB/LoopBack`

For more information about the StrongLoop API Gateway, see the following URL:

`https://docs.strongloop.com/display/LGW/StrongLoop+API+Gateway`

Additionally, you can follow the tutorial in Chapter 7, “Synchronizing data from Salesforce to a remote enterprise system” on page 189 to build and expose an API using StrongLoop.

4.4.3 Conclusion

To drive innovation and provide engaging client experiences, organizations are looking to leverage and expose existing systems and applications through APIs. In this section, we provided information about the API facade pattern that can be used in that context. We also looked at one specific way to implement this pattern using StrongLoop, which is one example out of many.

4.5 Pattern for data analytics integration with real-time events

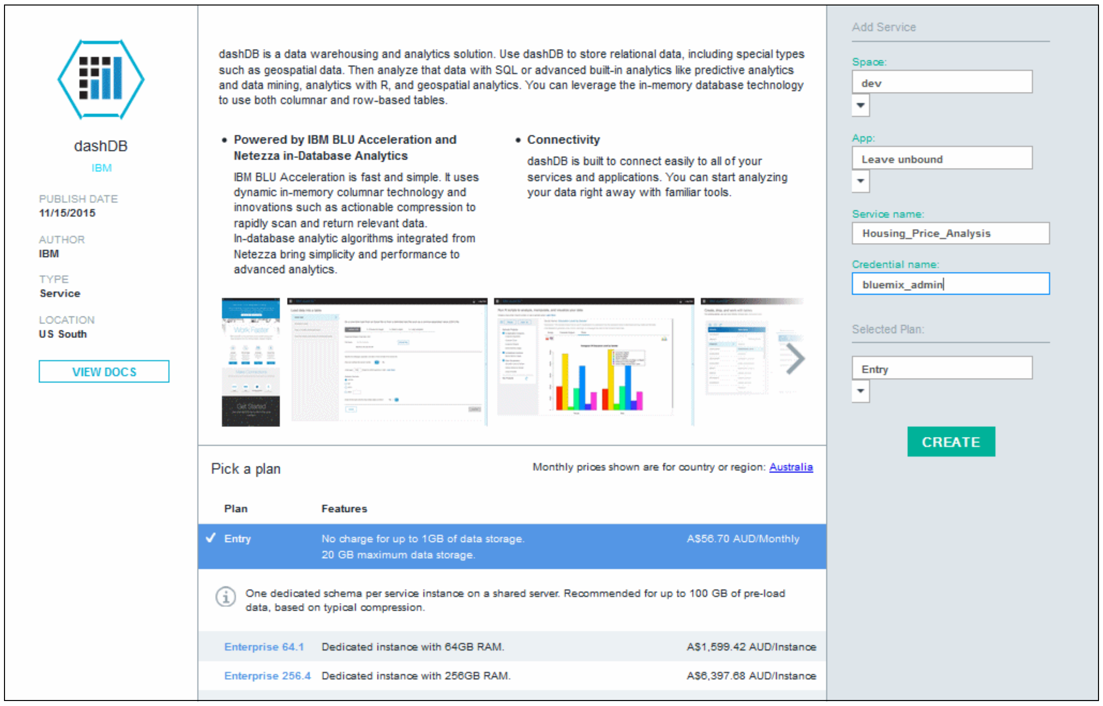
In this pattern, the integration between developing a model using data and integrating this model with a real-time event is considered. To achieve this, two different services in Bluemix will be used: The first service is IBM dashDB™, which is used for storing data in a database, and then developing a model using the data through an R script. The second service is the streaming analytics service, which is used to detect a real-time event, based on criteria set by the user.

4.5.1 dashDB service

The dashDB service has two different aspects to consider: Importing data into a dashDB database, and using this data to develop a model.

Configuring a dashDB service

The dashDB service is available through the Bluemix catalog under the data analytics section. When the service is selected, a window is brought up with some basic configuration setting regarding the name of the service and the credentials for the service user. An example of this page is shown in Figure 4-10.



PUBLISH DATE
11/15/2015

AUTHOR
IBM

TYPE
Service

LOCATION
US South

dashDB is a data warehousing and analytics solution. Use dashDB to store relational data, including special types such as geospatial data. Then analyze that data with SQL or advanced built-in analytics like predictive analytics and data mining, analytics with R, and geospatial analytics. You can leverage the in-memory database technology to use both columnar and row-based tables.

- **Powered by IBM BLU Acceleration and Netezza in-Database Analytics**

  IBM BLU Acceleration is fast and simple. It uses dynamic in-memory columnar technology and innovations such as actionable compression to rapidly scan and return relevant data.
  In-database analytic algorithms integrated from Netezza bring simplicity and performance to advanced analytics.

- **Connectivity**

  dashDB is built to connect easily to all of your services and applications. You can start analyzing your data right away with familiar tools.

Pick a plan

Monthly prices shown are for country or region: Australia

Plan	Features	
✓ Entry	No charge for up to 1GB of data storage. 20 GB maximum data storage.	A$56.70 AUD/Monthly
	One dedicated schema per service instance on a shared server. Recommended for up to 100 GB of pre-load data, based on typical compression.	
Enterprise 64.1	Dedicated instance with 64GB RAM.	A$1,599.42 AUD/Instance
Enterprise 256.4	Dedicated instance with 256GB RAM.	A$6,397.68 AUD/Instance

Space: dev

App: Leave unbound

Service name: Housing_Price_Analysis

Credential name: bluemix_admin

Selected Plan: Entry

Figure 4-10 dashDB configuration window

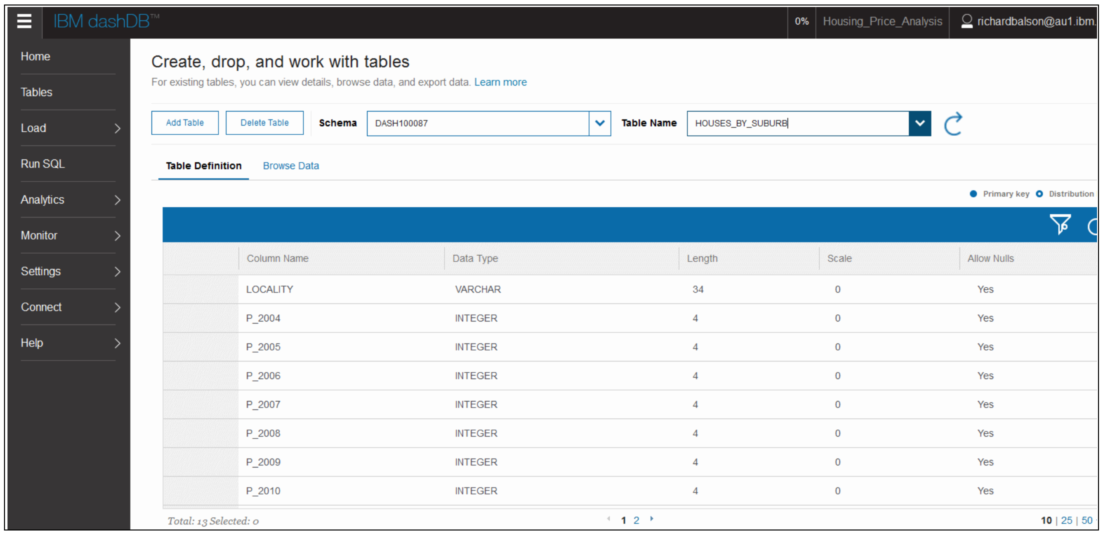
After the dashDB service is configured, it can be started through the Bluemix dashboard by first selecting the **dashDB** service and then launching the service. When the service is launched, the data stored locally needs to be uploaded to the Bluemix service.

Importing data into dashDB

On the launch page of the dashDB service, there are multiple tabs to select from. To import data the Load tab should be used, and depending on the location of the data source, an option should be selected. For example, data stored locally would require using the **Load** → **Desktop** option. This displays some options for loading the data into the database. Note that dashDB databases have requirements that the data being imported must meet, and this might require preprocessing the data before importing it into the service.

Viewing imported data in dashDB

To view data imported into dashDB, the **Table** tab can be selected. This opens a window that allows the user to access data by selecting the specific schema and table that they would like to view. An example is shown in Figure 4-11.



Create, drop, and work with tables

For existing tables, you can view details, browse data, and export data. Learn more

Schema DASH100087

Table Name HOUSES_BY_SUBURB

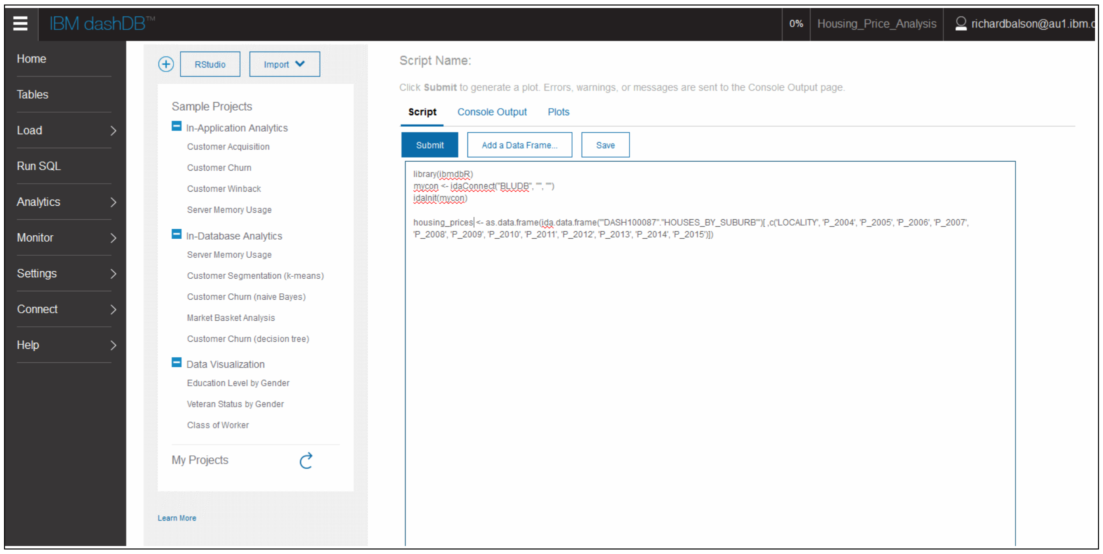
	Column Name	Data Type	Length	Scale	Allow Nulls
	LOCALITY	VARCHAR	34	0	Yes
	P_2004	INTEGER	4	0	Yes
	P_2005	INTEGER	4	0	Yes
	P_2006	INTEGER	4	0	Yes
	P_2007	INTEGER	4	0	Yes
	P_2008	INTEGER	4	0	Yes
	P_2009	INTEGER	4	0	Yes
	P_2010	INTEGER	4	0	Yes

Total: 13 Selected: 0 1 2 10 | 25 | 50

Figure 4-11 Viewing a table in dashDB

Developing a model using imported dashDB data

To develop a model in dashDB, the **Analytics** tab can be used. dashDB supports R as its analytics platform, and an R script can either be developed by using an editor or the RStudio IDE environment. Within the R script environment, new data can be imported into the R environment using the add data frame command at the top of the editor. Results can be viewed, after submitting the code, in the console and plots tabs. An example of the R script page is shown in Figure 4-12 on page 81.



```
library(ibmdbR)
mycon <- idaConnect("BLUDB", "", "")
idaInit(mycon)

housing_prices <- as.data.frame(ida.data.frame('"DASH100087"."HOUSES_BY_SUBURB"')[ ,c('LOCALITY', 'P_2004', 'P_2005', 'P_2006', 'P_2007',
'P_2008', 'P_2009', 'P_2010', 'P_2011', 'P_2012', 'P_2013', 'P_2014', 'P_2015')])
```

Figure 4-12   R script in the dashDB service

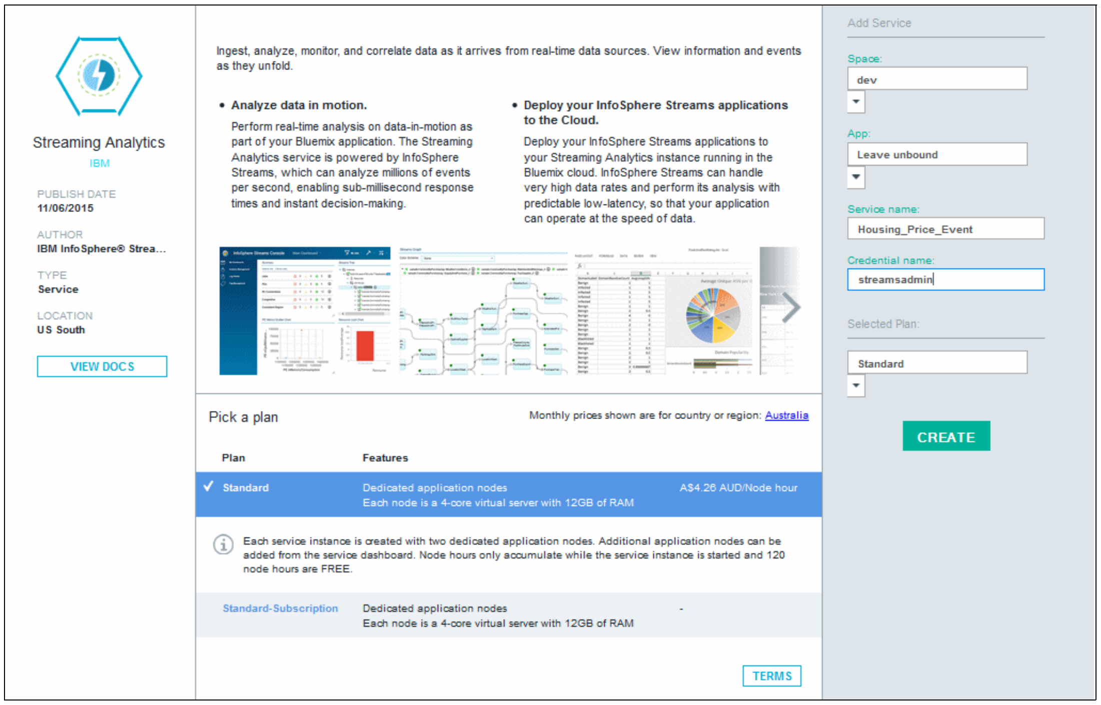
To prepare a model, some processing of the imported data might be required, and this can be achieved using R’s functionality. A model can then be built in R, and the functional description of the model can be extracted for use in the streaming analytics service.

4.5.2 Streaming analytics service

To create a real-time event detection system, the streaming analytics service in Bluemix can be used. The streaming analytics service allows the user to import a local streams application to a cloud service. To achieve this, three different aspects need to be considered: Configuring the Bluemix service, creating a local streams application, and loading the streams application to the cloud.

Configuring the streaming analytics service

The streaming analytics service can be found in the Bluemix Catalog under the data analytics section. The page to configure the service is similar to the dashDB service. An example is shown in Figure 4-13.



Streaming Analytics
IBM

PUBLISH DATE
11/06/2015

AUTHOR
IBM InfoSphere® Strea...

TYPE
Service

LOCATION
US South

VIEW DOCS

Ingest, analyze, monitor, and correlate data as it arrives from real-time data sources. View information and events as they unfold.

- **Analyze data in motion.**
  Perform real-time analysis on data-in-motion as part of your Bluemix application. The Streaming Analytics service is powered by InfoSphere Streams, which can analyze millions of events per second, enabling sub-millisecond response times and instant decision-making.
- **Deploy your InfoSphere Streams applications to the Cloud.**
  Deploy your InfoSphere Streams applications to your Streaming Analytics instance running in the Bluemix cloud. InfoSphere Streams can handle very high data rates and perform its analysis with predictable low-latency, so that your application can operate at the speed of data.

Pick a plan

Monthly prices shown are for country or region: Australia

Plan	Features	
✓ Standard	Dedicated application nodes Each node is a 4-core virtual server with 12GB of RAM	A$4.26 AUD/Node hour
	Each service instance is created with two dedicated application nodes. Additional application nodes can be added from the service dashboard. Node hours only accumulate while the service instance is started and 120 node hours are FREE.	
Standard-Subscription	Dedicated application nodes Each node is a 4-core virtual server with 12GB of RAM	-

TERMS

Add Service

Space: dev

App: Leave unbound

Service name: Housing_Price_Event

Credential name: streamsadmin

Selected Plan: Standard

CREATE

Figure 4-13 Configuring a streaming analytics service

Creating a local streams application

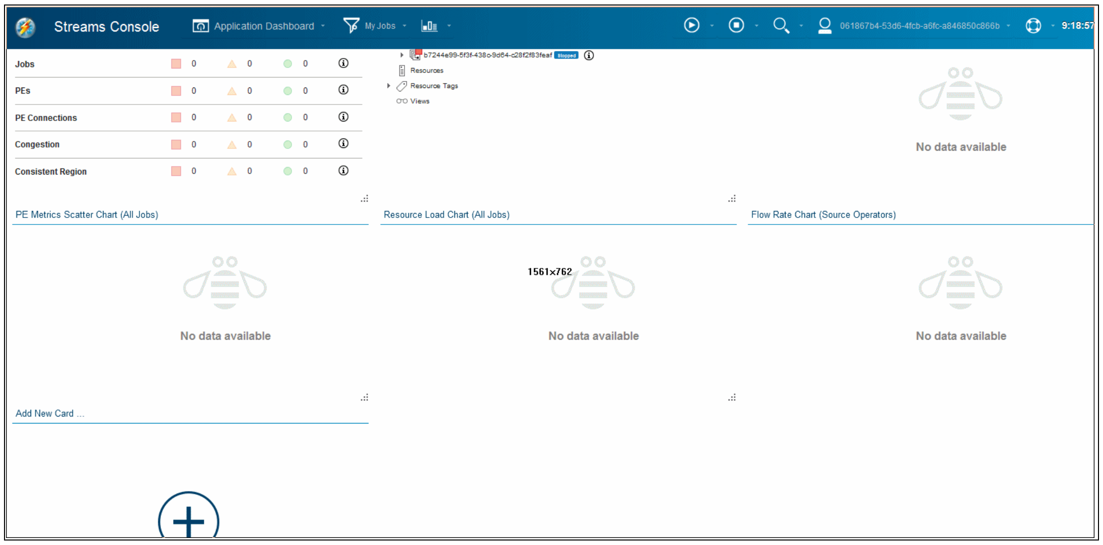
The streaming analytics service in Bluemix allows the user to import a local streams application to a cloud service. To use this service, it is necessary to first create a local streams application and submit the application to the cloud service. A streams application bundle can be created by using the streams quick start edition that is publicly available at the following site:

```
http://www.ibm.com/analytics/us/en/technology/stream-computing
```

Using this virtual machine or installing the software locally gives the user access to Streams Studio, which can be used to create a local streams application. This local streams application can be configured with the model developed in dashDB with trigger limits for real-time events, based on the user's criteria.

Importing the local streams application into Bluemix

After this streams application is created, it can be submitted to the Bluemix service and the application then runs on the cloud. To submit the application, open the streaming analytics service, and select **Start**. This starts the streams instance on which all applications are run. After starting the service, select **Launch**, which opens the streams console shown in Figure 4-14 on page 83. This figure shows a streams console where the service instance has been stopped (indicated by the red square next to the instance name).


Streams Console
Application Dashboard
My Jobs
061867b4-53d6-4fcb-a6fc-a846850c866b
9:18:57
b7244e99-5f3f-438o-9d64-c28f2f83feaf
Stopped
Resources
Resource Tags
Views
No data available
PE Metrics Scatter Chart (All Jobs)
Resource Load Chart (All Jobs)
Flow Rate Chart (Source Operators)
1561x762
Add New Card ...

Jobs	🟥	0	⚠	0	●	0	ⓘ
PEs	🟥	0	⚠	0	●	0	ⓘ
PE Connections	🟥	0	⚠	0	●	0	ⓘ
Congestion	🟥	0	⚠	0	●	0	ⓘ
Consistent Region	🟥	0	⚠	0	●	0	ⓘ

Figure 4-14 The streams console in Bluemix

To load the local stream application into the cloud service, the **Play** button on top of the streams console should be selected, and then a job should be submitted. This opens a window where the user needs to indicate the location of the stream application (the file name will end with .sab), and then click **Submit**, or click **Next** if the user wants to rename the job and specify a working directory.

See Chapter 9, “Demonstration of analytics and real-time event detection” on page 253 for the implementation of this pattern.

4.5.3 Conclusion

This pattern has indicated the process of creating a dashDB and streaming analytics service and has briefly described some of the key aspects that need to be considered when importing data, and developing a model using the dashDB service. The next aspect of the pattern described using the streaming analytics service to submit a locally created streams application bundle that is configured with the developed model and criteria for real-time event detection. The streams application then needs to be submitted to the streaming analytics service on Bluemix, and can be used to detect events in real-time and trigger further actions when an event is detected.

4.6 Pattern for integrating an Internet of Things device to an on-premises asset management system

The adoption of sensors connected to the Internet or Internet of Things devices is growing at a staggering rate. These devices produce a massive amount of data and provide intelligence and insights by a combination of other relevant technologies. There are hundreds of use cases for using these IoT devices. One important use case is to monitor systems that these sensors are attached to. This way, performance and health of the systems can be monitored in an automatic fashion. Sensor data can be sent to a server that supports lightweight MQ Telemetry Transport (MQTT) protocol and then an appropriate alert can be sent to an asset management system like IBM Maximo Asset Management. In this section, we present an architectural pattern similar to the use case above. This use case is illustrated in Figure 4-15.

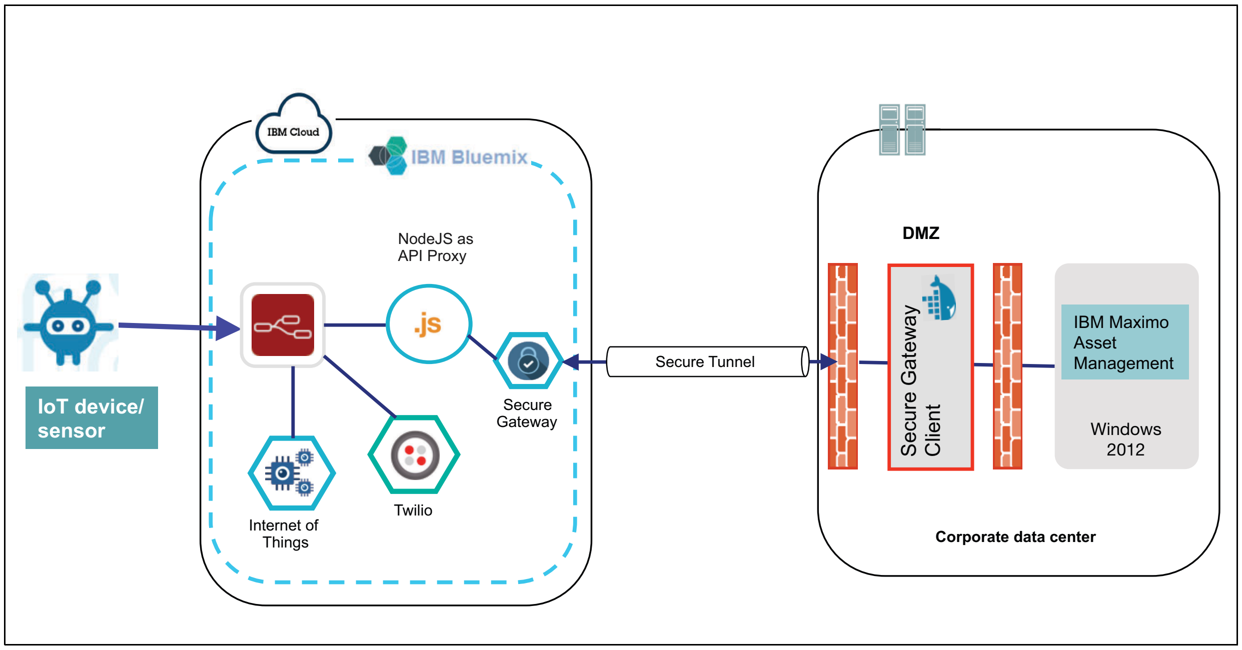


Figure 4-15 A pattern to connect an IoT device/sensor to an asset management system

In this scenario, the IBM Maximo Asset Management server is behind a corporate firewall. As such, an application running from an external cloud cannot connect to it directly. Following are some key considerations for integrating with an enterprise system of record (SOR) like IBM Maximo Asset Management server:

- Connection between the cloud application and Maximo should be secure. This is shown as the secure tunnel in the diagram.
- The data transferred through the tunnel should be encrypted by using TLS/mutual authentication.
- The Secure Gateway client that establishes a tunnel along with the Bluemix Secure Gateway service should provide high availability. That means you should be able to run a Secure Gateway client for the same tunnel/gateway in multiple locations to avoid a single point of failure.

Usually, a Secure Gateway client runs in a demilitarized zone or DMZ of the corporate network as shown in Figure 4-15. Bluemix Secure Gateway service and Secure Gateway client establishes the secure tunnel. Now a cloud application running from Bluemix can access the required data or API provided by Maximo.

Sensors or IoT devices use MQTT, a lightweight and publish/subscribe messaging protocol running on TCP/IP. An MQTT broker is required to connect IoT devices with applications connected to the broker. The IBM Internet of Things Foundation or IoTF acts as an MQTT broker. It is a fully managed and cloud-hosted service that provides functionalities, such as:

- Set up, control, and manage a device from an online dashboard.
- Secure APIs for MQTT clients (devices and applications).
- View live and historical data of devices connected.

Figure 4-16[1] demonstrates how different IBM IoT Foundation and IoT devices talk to each other. Note that a sensor needs connectivity to the Internet. In Figure 4-16, it is achieved through a smartphone that is connected to the Internet, which is acting as an Internet Gateway. The sensor is connected to the smartphone through Bluetooth.

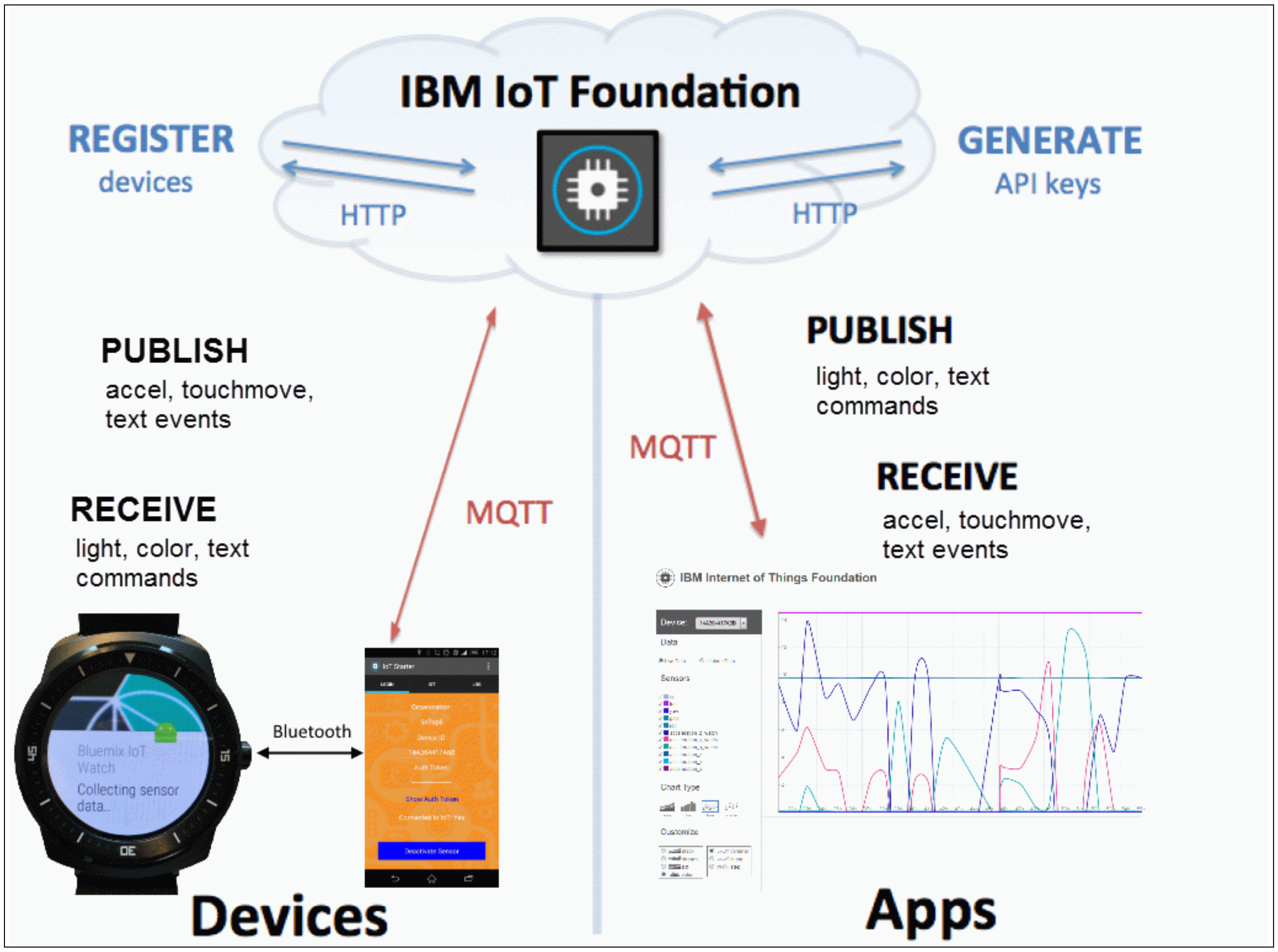


Figure 4-16   Architectural pattern showing IBM IoT Foundation and IoT devices

MQTT broker or IoTF is responsible for distributing messages to any connected clients (device or application).

Now refer to Figure 4-15 on page 84, which shows that an IoT device is talking to a Node-RED starter application that is bound to an IoTF service. In this architecture, the IoTF service is receiving messages from the IoT device using MQTT protocol. This message is parsed by a Node-RED application to see if the sensor or IoT device detected any fault with the equipment it is attached to.

[1] Source: https://developer.ibm.com/bluemix/2015/05/15/turn-android-watch-iot-device-bluemix

If the equipment that the sensor is monitoring requires maintenance, the Node-RED application invokes a RESTful API call to the asset management system through a NodeJS application acting as a proxy. The NodeJS application uses the IP address and port provided by Secure Gateway service, and certificates for TLS/mutual authentication.

This architectural pattern also demonstrates a composable cloud application model. A third-party service, Twilio, is also used here. When the NodeJS application returns the API response to the Node-RED application, an SMS with a work order number is sent to a support personnel using this Twillio service.

An implementation of this architectural pattern is presented in Chapter 8, “Integrating events from Internet of Things with Enterprise Asset Management systems” on page 217.

4.7 Pattern for integrating cloud applications using IBM Message Hub for Bluemix

IBM Message Hub for Bluemix is a scalable, distributed, high throughput message bus to unite your on-premises and off-premises cloud technologies. You can wire microservices together by using open protocols, and you can connect stream data to analytics to realize powerful insights. You can also feed event data to multiple applications to react in real time.

Message Hub is based on Apache Kafka, which is a fast, scalable, and durable real-time messaging engine that is developed by the Apache Software Foundation. For more information about Apache Kafka, see the following link:

http://kafka.apache.org/documentation.html

The Message Hub uses the Message Connect Bluemix service to connect to IBM MQ Light. This allows applications built using the Apache Kafka API running on IBM or third-party cloud to exchange messages with the IBM MQ Light server as shown in Figure 4-17.

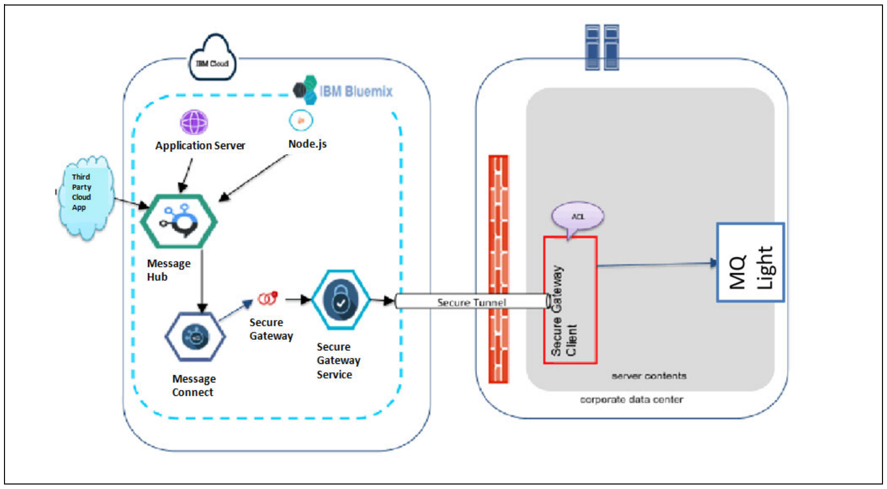


Figure 4-17 Pattern for using Message Hub to connect cloud applications with IBM MQ Light

Part 3

Hybrid cloud scenarios with IBM Bluemix

In this part, we cover several hybrid cloud scenarios with IBM Bluemix. Each scenario is described with step-by-step instructions so that you can implement similar scenarios in your environment as well.

The following chapters are included in Part 3:

- Chapter 5, “On-premises messaging middleware integration with IBM Bluemix” on page 89
- Chapter 6, “Asynchronous processing through IBM MQ Light service” on page 109
- Chapter 7, “Synchronizing data from Salesforce to a remote enterprise system” on page 189
- Chapter 8, “Integrating events from Internet of Things with Enterprise Asset Management systems” on page 217
- Chapter 9, “Demonstration of analytics and real-time event detection” on page 253

5

On-premises messaging middleware integration with IBM Bluemix

In this chapter, on-premises messaging middleware products are integrated with IBM Bluemix to achieve asynchronous non time-dependent communications between on-premises and cloud applications.

The two scenarios presented here demonstrate the concept of a message producer sending asynchronous messages from on-premises messaging middleware to an application hosted in the cloud. IBM MQ Light and ActiveMQ Apollo are used as the two on-premises messaging middleware products. The following components are needed for this solution:

- On-premises environment
  - IBM MQ Light, Message producer application built using Node.js
  - ActiveMQ Apollo, Message producer application built using Node.js
- IBM Bluemix Secure Gateway service
- Message Consumer application hosted on Bluemix
  - NodeJS application
  - MQLight API
  - Cloudant NoSQL Database

This chapter contains the following topics:

5.1 Scenario architecture

In this scenario, simulation of an event that would occur in an enterprise environment that we would like to access from the cloud is achieved. An example is a retail company that offers different access channels to customers to buy their products. They provide access on the web, in company stores, and at other stores. The sales and promotions are controlled via the marketing department residing in the company's head office. The marketing manager initiates a promotions program via the company's corporate system that is hosted on a local data center and this event then gets distributed across all the stores. Each individual store has its IT systems hosted on IBM Cloud.

IBM Bluemix Secure Gateway service is used to consume the on-premises event data from a cloud application. The Secure Gateway service provides you with a secure way to access your on-premises or cloud data from your Bluemix application through a secure passage.

The scenario architecture is shown in Figure 5-1. The company's corporate systems are hosted on the corporate data center and communicate with each other via a messaging middleware. The messaging middleware is also connected to the Secure Gateway client, which exposes selected resources on the messaging middleware (via a secure tunnel) to the outside world. The retail and online stores are hosted on IBM Cloud and these use IBM Secure Gateway service to connect to the on-premises Secure Gateway client.

The integration of cloud applications via IBM Secure Gateway service with both proprietary and open source messaging middleware products is presented. For this demonstration, we use the following middleware products:

- IBM MQ Light
- ActiveMQ Apollo

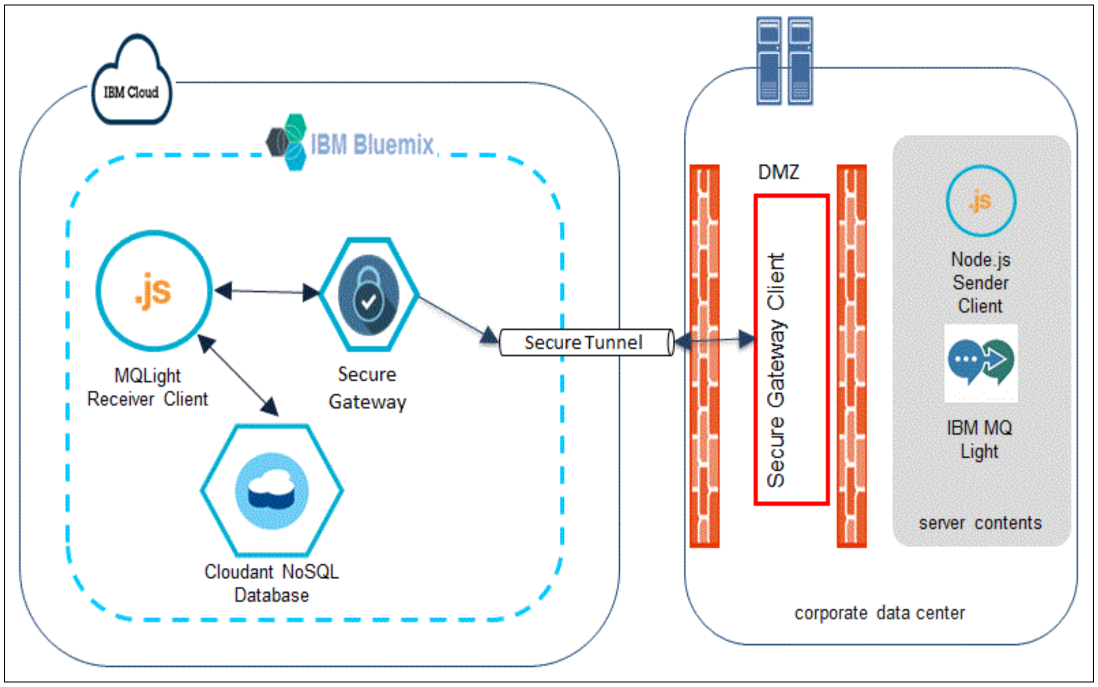


Figure 5-1 Integration between on-premises and cloud applications using IBM Secure Gateway

5.2 Setting up an on-premises environment using IBM MQ Light

In this section, the setup that is required for using the IBM MQ Light package as our on-premises message middleware is described.

Then, the writing of a simple Node.js event producer application that simulates a message sending client on-premises is shown. This sample application sends messages via the AMQP protocol.

Therefore, both the latest IBM MQ Light and Node.js packages must be downloaded and installed.

5.2.1 IBM MQ Light download and installation

First, download the relevant developer server build for your environment from the following URL. Builds for Linux, MacOS, and Microsoft Windows are available. Follow the relevant installation instructions:

`https://developer.ibm.com/messaging/ibm-mq-light-downloads`

Start your IBM MQ Light server. You are shown the following dashboard if you followed the instructions correctly. See Figure 5-2.

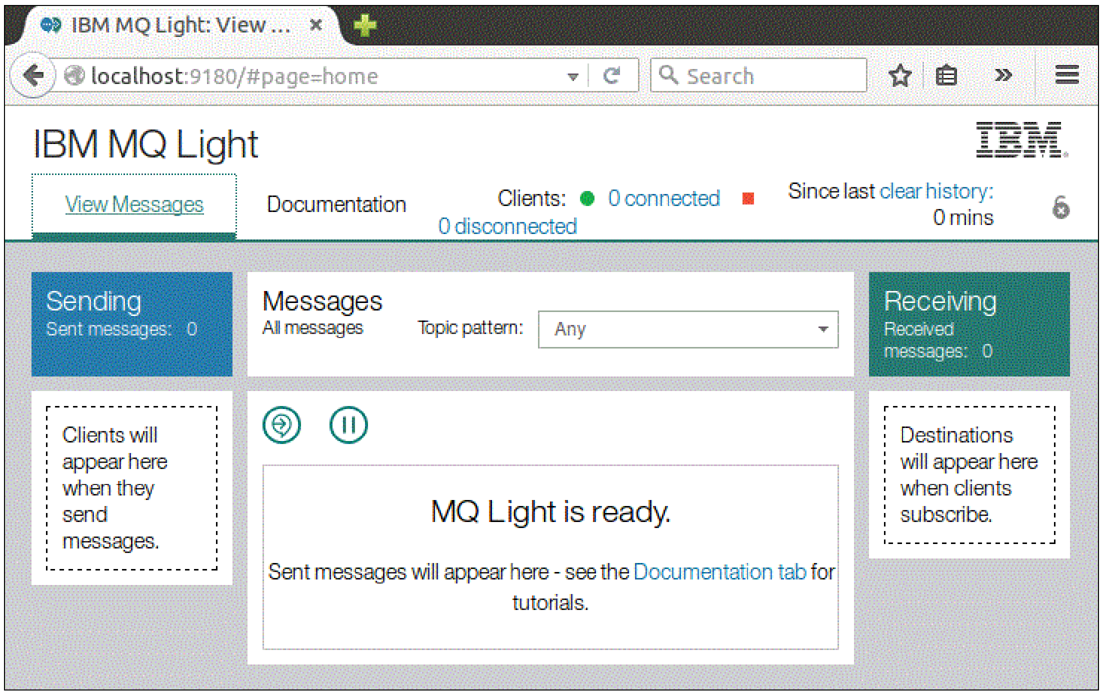

IBM MQ Light: View ...
localhost:9180/#page=home
Search
IBM MQ Light
IBM
View Messages
Documentation
Clients: ● 0 connected ■ 0 disconnected
Since last clear history: 0 mins
Sending
Sent messages: 0
Messages
All messages
Topic pattern: Any
Receiving
Received messages: 0
Clients will appear here when they send messages.
MQ Light is ready.
Sent messages will appear here - see the Documentation tab for tutorials.
Destinations will appear here when clients subscribe.

Figure 5-2 IBM MQ server dashboard

5.2.2 Node.js download and installation

Download and install the latest version of Node.js on your environment.

`https://nodejs.org/en/download`

To confirm successful installation, issue the **`node --version`** command in your console. The version installed on your machine should be shown.

5.2.3 Writing a simple Node.js IBM MQ Light message sender client

For the purpose of this scenario, a simple Node.js application is used that sends messages to our IBM MQ Light server. Both the IBM MQ server and the application are running on a machine that is on-premises.

To do this, create a new project folder. In this project folder, create a new JavaScript file called `app.js` by using a text editor of your choice:

1. Go to the filepath of the project folder and open your console or command line. The following Node.js modules are required:
   - mqlight
   - moment
   - hashmap
2. Node.js comes with a package manager pre-installation called npm (node package manager). To install these dependencies, simply type and enter the following commands while in the project directory:
   - `npm install mqlight`
   - `npm install moment`
   - `npm install hashmap`

   This installs the required dependencies in a folder called `node_modules` in the project directory.

   The mqlight node package API documentation can be found here:

   https://www.npmjs.com/package/mqlight
3. Now open the `app.js` file and add the following code, which is shown in Example 5-1.

Example 5-1 Sample Node.js message sender client

```
//reqire mqlight package
var mqlight = require(‘mqlight’);

//configuration object (options) of MQ Light client being created
var config = {
   service: ‘amqp://localhost’
};

//Create an mqlight client with options in config
var sendClient = mqlight.createClient(config);

//Define name of a topic to send message to
var topic = ‘public’;

//Send a message when client is started
sendClient.on(‘started’, function(){

   //Message to send
   var message = “Hello Red Book!”;

   //Send a message parameters: topic and message
   sendClient.send(topic, message, function(error, data){
      //Basic error checking
      if(!error)
      {
         console.log(‘Message: ‘, data);
```

```
            sendClient.stop();
         }
         else
         {
            console.log(error);
         }

      });

   });
```

4. Save the file and try to run it. To run it, type *node app.js* into your command window or console (ensure that you are in the same directory as the `app.js` file).

   The message data being sent should be logged in the console, and you should be able to see the message sitting in a topic in the IBM MQ Light dashboard. You should also be able to see a Sending client on the left side (Figure 5-3).

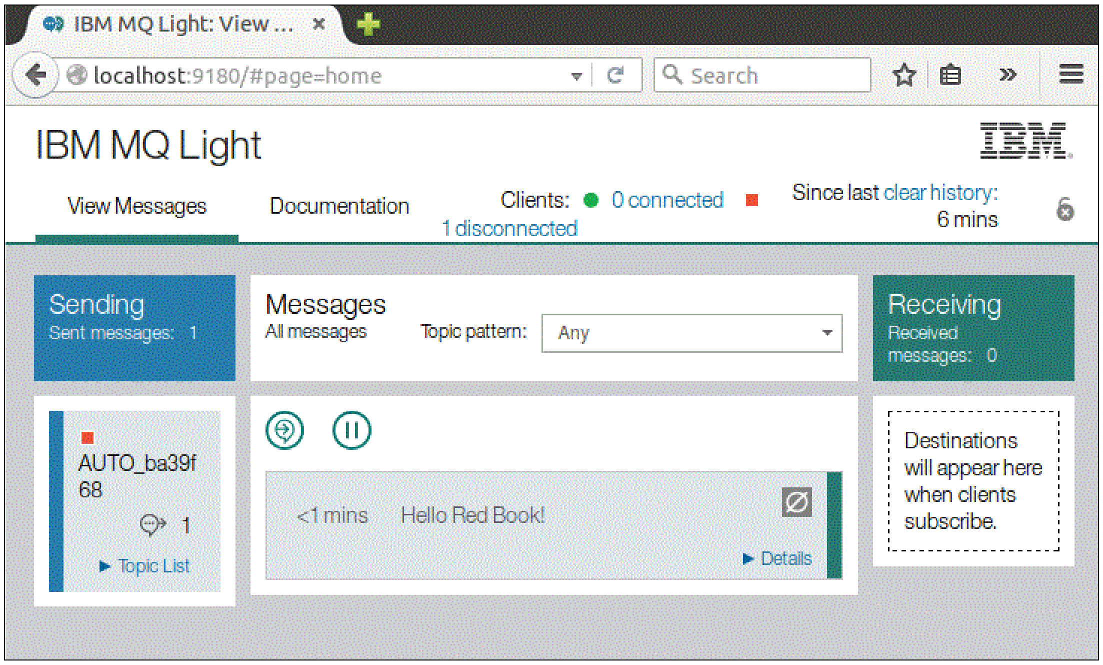


Figure 5-3 Message sitting in an IBM MQ Light topic

5.3 Setting up an on-premises environment using ActiveMQ Apollo

The setup that is required for using ActiveMQ Apollo as our on-premises message middleware is described in this section.

A simple Node.js event producer application that simulates a message sending client on-premises is also required. This sample application sends messages using the MQ Telemetry Transport (MQTT) protocol.

5.3.1 ActiveMQ Apollo

ActiveMQ Apollo is a faster, more reliable, easier to maintain messaging broker built from the foundations of the original Apache ActiveMQ. It is an open source product and is used in the scenario to demonstrate that IBM Secure Gateway service integrates well with both IBM proprietary products like MQ Light as well as open source products like Apollo.

5.3.2 Setup and Installation

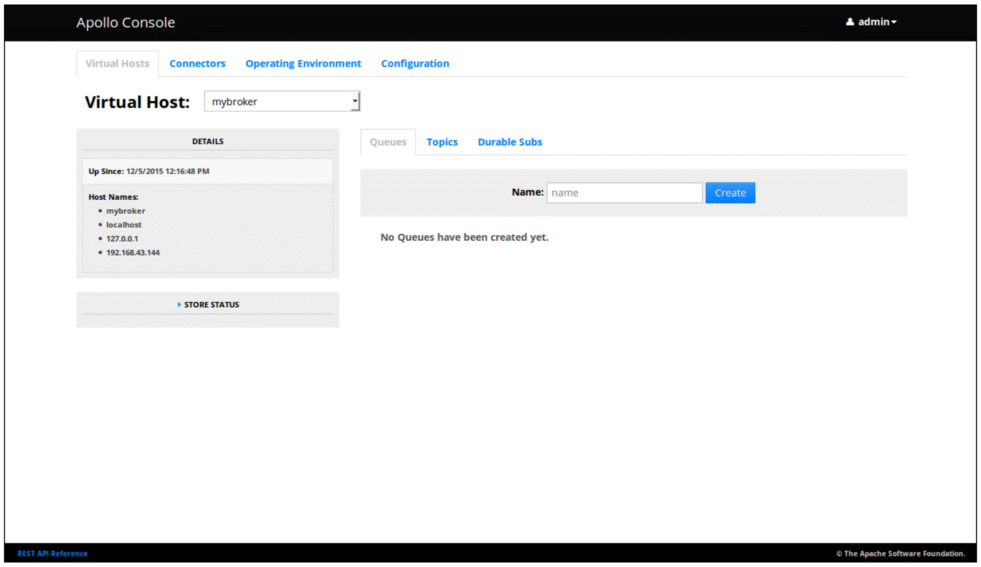
Download the ActiveMQ Apollo version that is compatible with your environment from the following URL:

https://activemq.apache.org/apollo/download.html

Builds for Linux, MacOS, and Microsoft Windows are available. Follow the relevant installation instructions from the following site:

https://activemq.apache.org/apollo/documentation/getting-started.html

When you start a broker instance, log in to the admin console to verify your installation. The admin console should look like Figure 5-4.



Up Since: 12/5/2015 12:16:48 PM

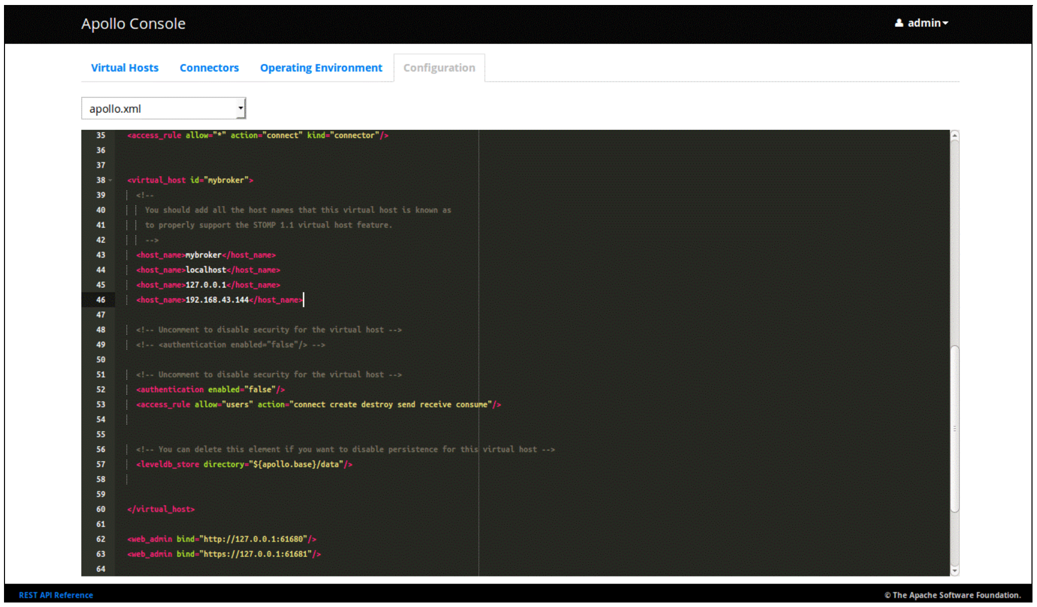
Host Names:
- mybroker
- localhost
- 127.0.0.1
- 192.168.43.144

No Queues have been created yet.

Figure 5-4 ActiveMQ Apollo Web Admin Console

By default, the new Apollo broker is bound to the loopback address. Now update the broker configuration to bound your public IP address, which is used by the IBM Secure Gateway client.

From the Web Admin Console, click **Configuration** and add your public IP address in a new <host_name> element as shown in Figure 5-5.



```
<access_rule allow="*" action="connect" kind="connector"/>


<virtual_host id="mybroker">
  <!--
    You should add all the host names that this virtual host is known as
    to properly support the STOMP 1.1 virtual host feature.
    -->
  <host_name>mybroker</host_name>
  <host_name>localhost</host_name>
  <host_name>127.0.0.1</host_name>
  <host_name>192.168.43.144</host_name>

  <!-- Uncomment to disable security for the virtual host -->
  <!-- <authentication enabled="false"/> -->

  <!-- Uncomment to disable security for the virtual host -->
  <authentication enabled="false"/>
  <access_rule allow="users" action="connect create destroy send receive consume"/>


  <!-- You can delete this element if you want to disable persistence for this virtual host -->
  <leveldb_store directory="${apollo.base}/data"/>


</virtual_host>

<web_admin bind="http://127.0.0.1:61680"/>
<web_admin bind="https://127.0.0.1:61681"/>
```

Figure 5-5 ActiveMQ Apollo broker configuration

For this demonstration, authentication on the ActiveMQ Apollo broker is disabled. This is done by setting the “enabled” attribute to “false” for authentication as shown in Figure 5-5.

5.3.3 Writing a simple Node.js ActiveMQ Apollo message sender client

For this scenario, a simple Node.js application that will act as the event producer is required. You need to first download and setup Node.js as discussed in section 5.2.2, “Node.js download and installation” on page 91.

Our consumer application sends messages to ActiveMQ Apollo by using the MQTT protocol. More details about MQTT can be found at the following link:

http://mqtt.org

Create a new project folder. In this project folder, create a new JavaScript file called `app.js` by using a text editor of your choice:

1. Go to the filepath of the project folder and open your console or command line. The following Node.js module is required:

   mqtt

2. Node.js comes with a package manager preinstallation called *npm* (node package manager). To install these dependencies, simply type and enter the following commands while in the project directory:

   ```
   npm install mqtt
   ```

   This installs the required dependencies in a folder called `node_modules` in the project directory.

   The mqtt node package API documentation can be found here:

   https://www.npmjs.com/package/mqtt

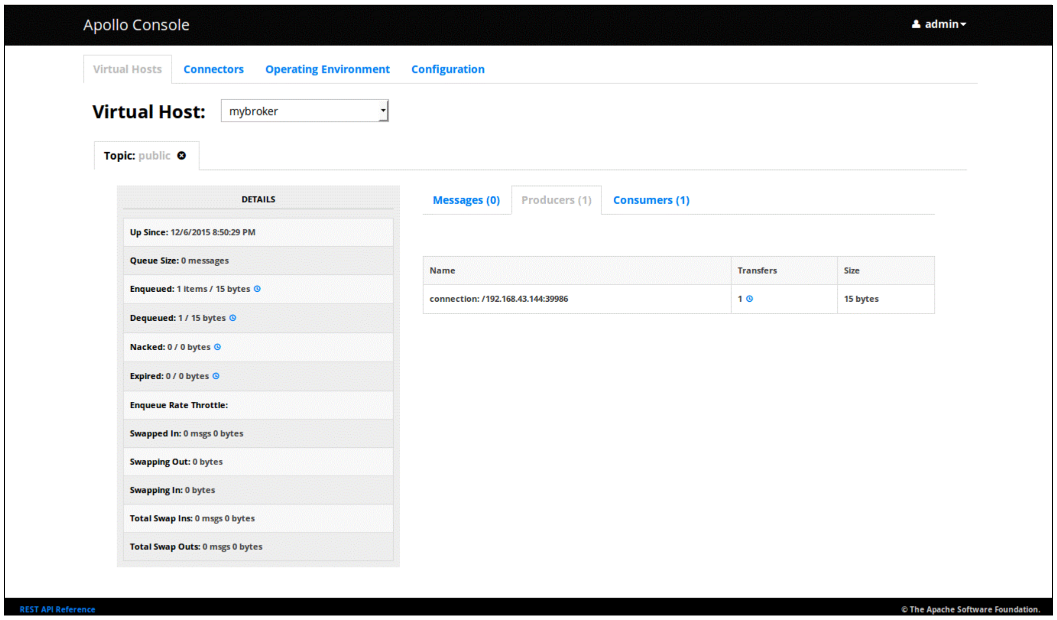
3. Now open the `app.js` file and add the following code, as shown in Example 5-2.

Example 5-2 Sample Node.js event producer using MQTT

```
//require mqtt package
var mqtt = require('mqtt');

//connection url for client
var url = 'mqtt://[YOUR_PUBLIC_IP_ADDRESS]:61613';


//Create mqlight client
var sendClient = mqtt.connect(url);

//Define topic
var topic = 'public';

var msg = "Hello Red Book!";

//Publish Message
sendClient.on('connect', function(){
   sendClient.subscribe(topic);
   sendClient.publish(topic, msg);
});
```

4. Save the file and try to run it. To run it, type *node app.js* into your command window or console (ensure that you are in the same directory as the `app.js` file).

   The message data being sent should be logged in the console, and you should be able to see the message sitting in the topic name “public” in the ActiveMQ Apollo Web Admin Console, as shown in Figure 5-6.



Virtual Host: mybroker

Topic: public

DETAILS

Up Since: 12/6/2015 8:50:29 PM
Queue Size: 0 messages
Enqueued: 1 items / 15 bytes
Dequeued: 1 / 15 bytes
Nacked: 0 / 0 bytes
Expired: 0 / 0 bytes
Enqueue Rate Throttle:
Swapped In: 0 msgs 0 bytes
Swapping Out: 0 bytes
Swapping In: 0 bytes
Total Swap Ins: 0 msgs 0 bytes
Total Swap Outs: 0 msgs 0 bytes

Name	Transfers	Size
connection: /192.168.43.144:39986	1	15 bytes

Figure 5-6 Apollo admin console message producers

5.4 Using IBM Secure Gateway to connect your on-premises sender client to the cloud application

In this section, the steps required to configure Secure Gateway are described:

- IBM Secure Gateway service on Bluemix
- IBM Secure Gateway client at an on-premises environment

These steps are identical for both IBM MQ Light and ActiveMQ Apollo.

5.4.1 Configure IBM Secure Gateway service

To configure an IBM Secure Gateway service, follow these steps:

1. Log in to IBM Bluemix:

   `https://console.ng.bluemix.net`

2. Now create an IBM Secure Gateway service by clicking the **Dashboard** and then **Use Services or APIS**. See Figure 5-7.

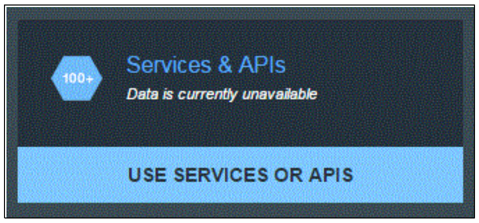


Figure 5-7 Using service and APIs

3. Click **Secure Gateway**, which can be found with the other integration services that are available. See Figure 5-8.

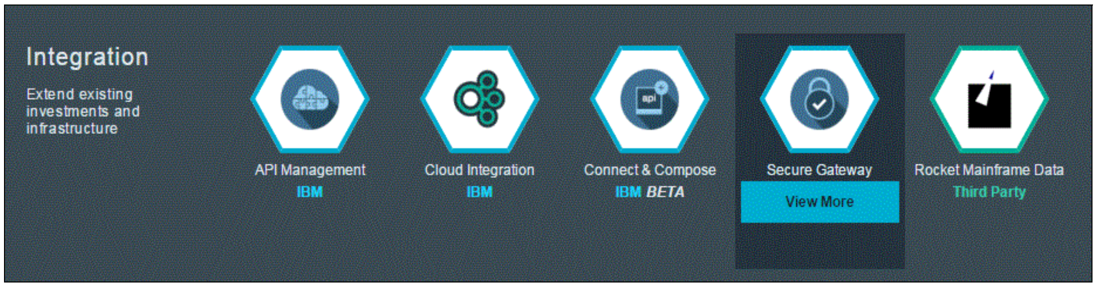
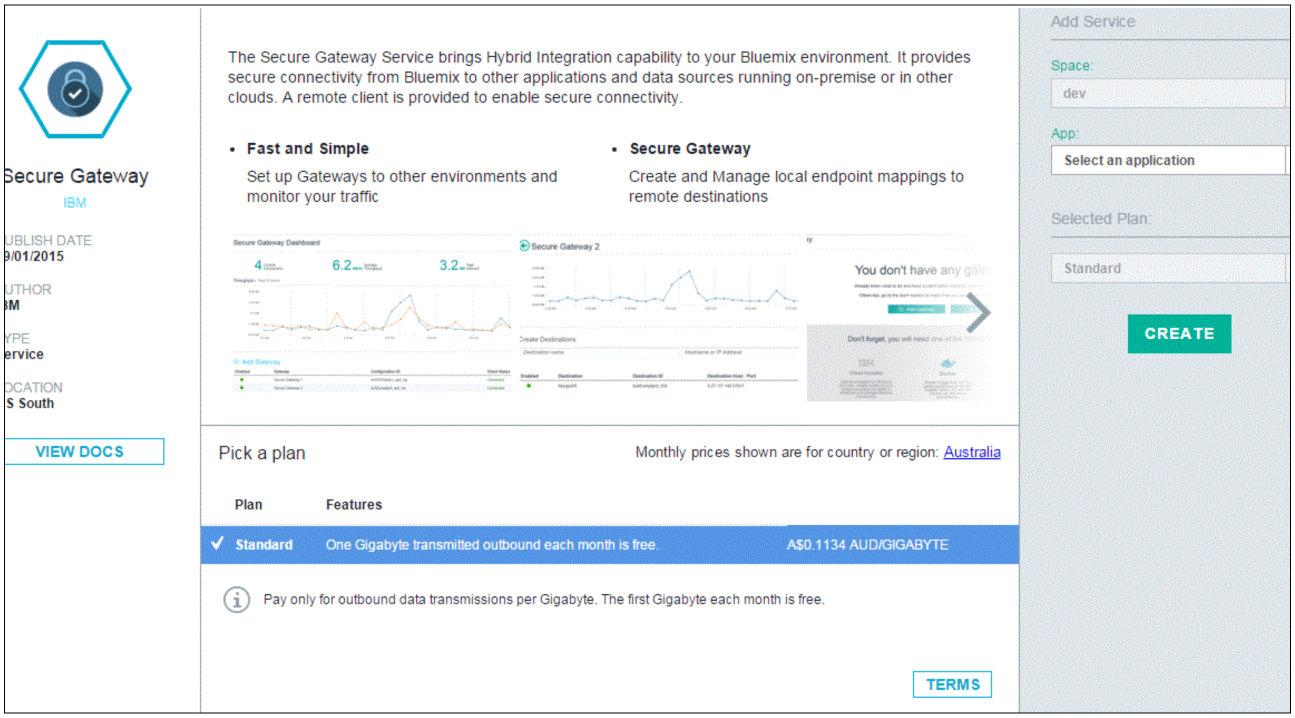


Figure 5-8 Adding a Secure Gateway service

4. Leave the service settings as default and click **CREATE**, as shown in Figure 5-9. This creates an instance of Secure Gateway for you to use.


Secure Gateway
IBM
UBLISH DATE
9/01/2015
UTHOR
BM
YPE
ervice
OCATION
S South
VIEW DOCS

The Secure Gateway Service brings Hybrid Integration capability to your Bluemix environment. It provides secure connectivity from Bluemix to other applications and data sources running on-premise or in other clouds. A remote client is provided to enable secure connectivity.

- **Fast and Simple**
  Set up Gateways to other environments and monitor your traffic
- **Secure Gateway**
  Create and Manage local endpoint mappings to remote destinations

Pick a plan

Monthly prices shown are for country or region: Australia

Plan	Features	
✓ Standard	One Gigabyte transmitted outbound each month is free.	A$0.1134 AUD/GIGABYTE

Pay only for outbound data transmissions per Gigabyte. The first Gigabyte each month is free.

TERMS

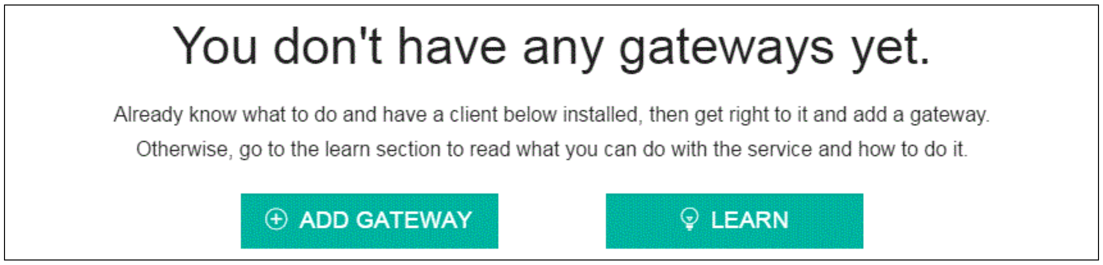
Add Service
Space: dev
App: Select an application
Selected Plan: Standard
CREATE

Figure 5-9 Creating an instance of Secure Gateway

5. Back at the dashboard, click the **Secure Gateway** icon to be taken to the Secure Gateway service dashboard.

   We need to connect the gateway to our on-premises application, which is sending messages over ampq so that the application we create and push to the cloud can receive them.

6. Click the service icon that you created in the dashboard. If this is the first time creating a gateway with the service, click **ADD GATEWAY**, as shown in Figure 5-10.

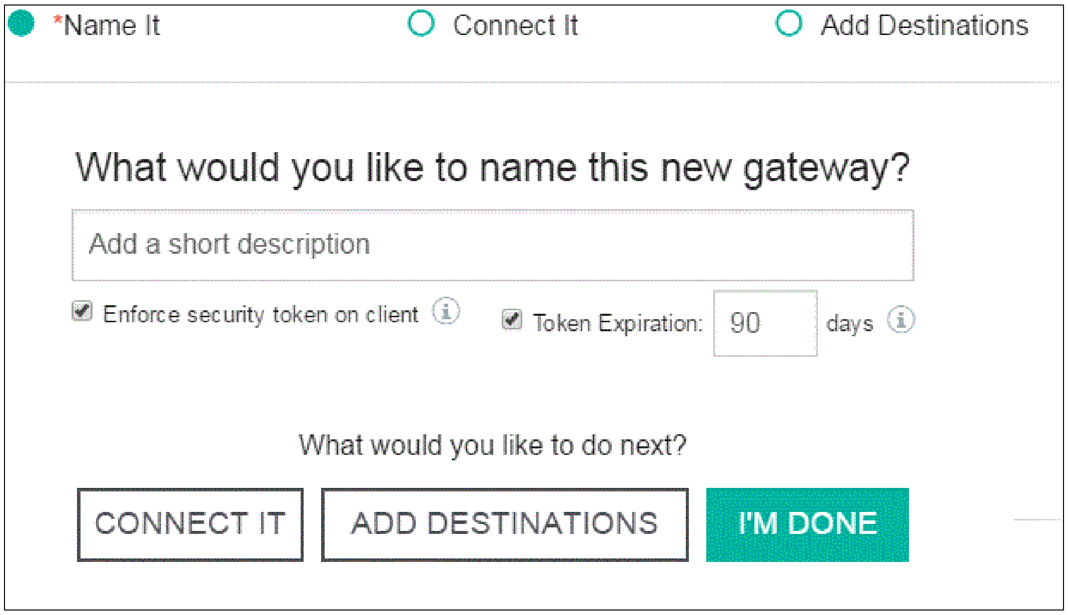

You don't have any gateways yet.

Already know what to do and have a client below installed, then get right to it and add a gateway. Otherwise, go to the learn section to read what you can do with the service and how to do it.

ADD GATEWAY   LEARN

Figure 5-10 Adding a gateway

7. Name the gateway (leave default security settings), and click **CONNECT IT**. See Figure 5-11.



What would you like to name this new gateway?

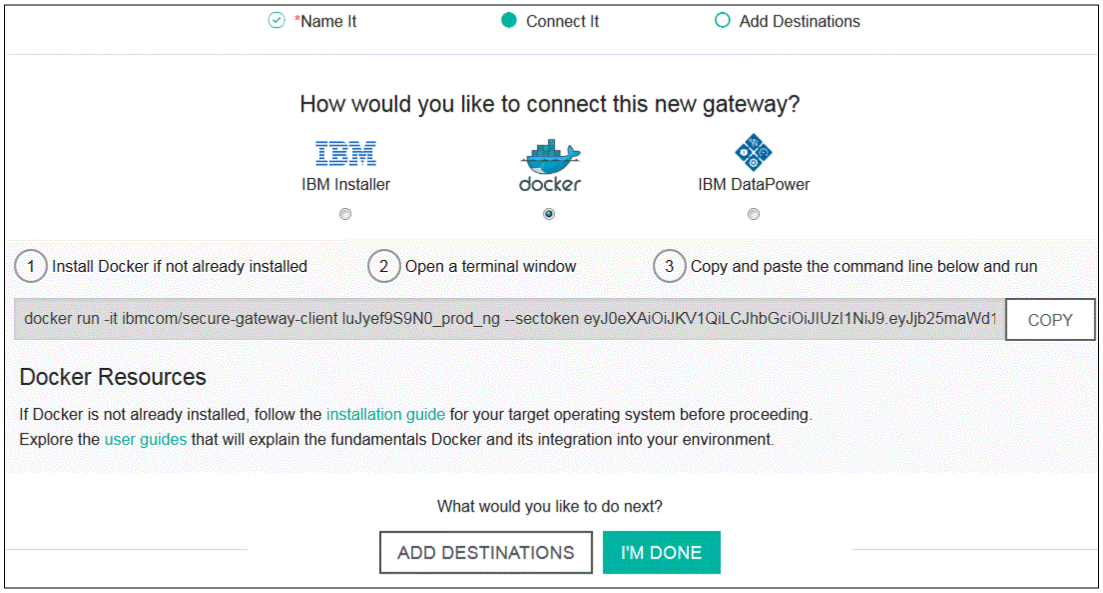
Add a short description

☑ Enforce security token on client ☑ Token Expiration: 90 days

What would you like to do next?

Figure 5-11 Naming your gateway

8. When you click **CONNECT IT**, you are presented with three options regarding how you want to connect the gateway: The IBM Installer native client, Docker, or IBM DataPower. For this example, use Docker. Select the **Docker** radio button and copy the command that you are supplied with. This command should be run from the command line of your on-premises machine whenever you want to connect the Secure Gateway tunnel. See Figure 5-12.



How would you like to connect this new gateway?

1 Install Docker if not already installed
2 Open a terminal window
3 Copy and paste the command line below and run

```
docker run -it ibmcom/secure-gateway-client luJyef9S9N0_prod_ng --sectoken eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJjb25maWd1
```

Docker Resources

If Docker is not already installed, follow the installation guide for your target operating system before proceeding.
Explore the user guides that will explain the fundamentals Docker and its integration into your environment.

What would you like to do next?

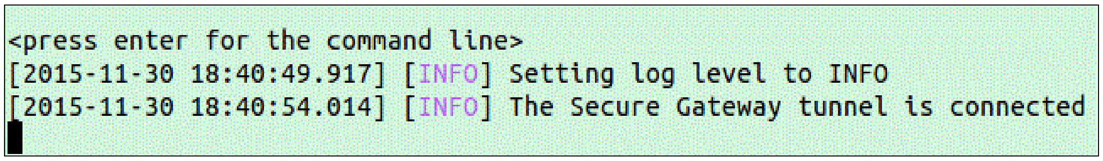
Figure 5-12 Docker command

5.4.2 Configure IBM Secure Gateway client

To configure Secure Gateway client, run the command copied in step 8.

If you have not already installed Docker, the first time that you run the command, Docker will be installed. Docker pulls an image onto your machine connecting the machine and IBM Bluemix through a secure tunnel using Secure Gateway.

Your machine should display that the Secure Gateway tunnel is connected in the command-line interface if you have not encountered any errors, as shown in Figure 5-13.



```
<press enter for the command line>
[2015-11-30 18:40:49.917] [INFO] Setting log level to INFO
[2015-11-30 18:40:54.014] [INFO] The Secure Gateway tunnel is connected
```

Figure 5-13 Secure tunnel connected

The Bluemix dashboard should also indicate that the secure tunnel is now connected, as shown by the box on the right in Figure 5-14. Red rings indicate that the tunnel is not currently connected, and green rings indicate that the tunnel is currently connected (Figure 5-14).

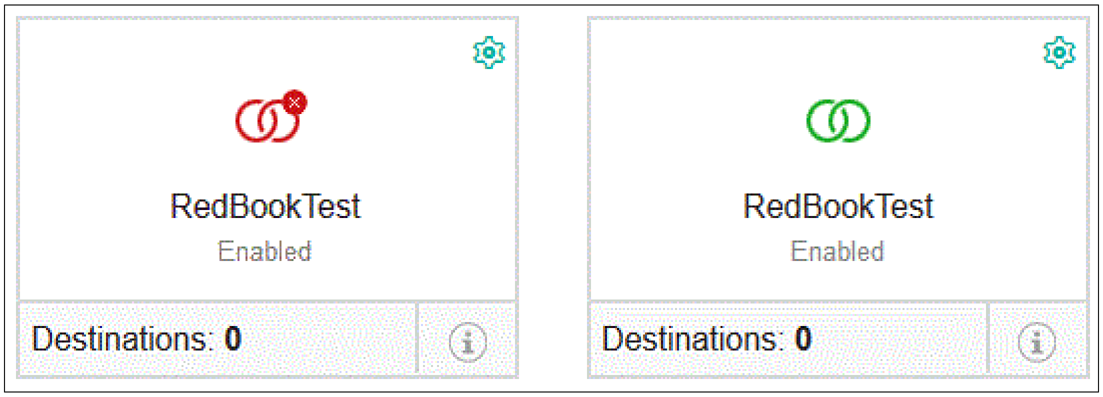


Figure 5-14 Secure tunnel connected (Bluemix)

5.4.3 Connect Secure Gateway with the on-premises environment

The secure tunnel should now be connected, but we have not defined any destinations yet. Follow these steps to do define the destinations:

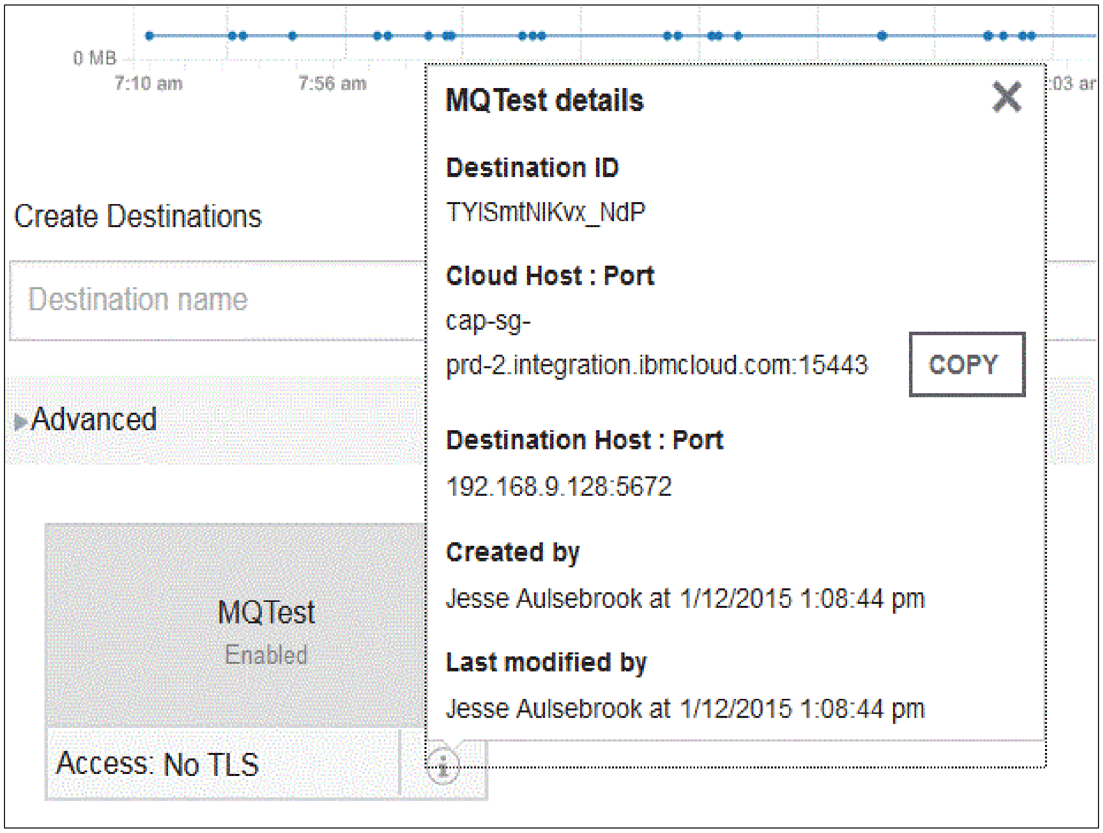
1. Click the gateway that you just created, and enter the relevant environment details of the server where your on-premises assets are hosted. Required are a destination name, the host name or IP address, and a port number.

   For this example, we are using the *ampq port 5672* for IBM MQ Light and *mqtt port 61613* for ActiveMQ Apollo.

2. When the details are entered, click the **ADD DESTINATION** icon.

   You have now connected your on-premises message service broker to IBM Bluemix.

3. A cloud host url : port to use in your receiver application is required. This can be found by clicking the information icon of the gateway. Write the **Cloud Host : Port** down because we require it when connecting our receiver application to the on-premises assets. See Figure 5-15.



MQTest details

Destination ID

TYISmtNIKvx_NdP

Cloud Host : Port

cap-sg-prd-2.integration.ibmcloud.com:15443

Destination Host : Port

192.168.9.128:5672

Created by

Jesse Aulsebrook at 1/12/2015 1:08:44 pm

Last modified by

Jesse Aulsebrook at 1/12/2015 1:08:44 pm

Figure 5-15 Cloud host url : port information

5.5 Writing a simple Node.js IBM MQ Light Message receiver client

In this section, the steps for writing a simple Node.js IBM MQ Light Message receiver client are covered. This application receives event messages from the on-premises event publisher.

5.5.1 Creating the application

Now that the secure tunnel is connecting the on-premises applications with the cloud, write a simple Node.js application receiver client, which receives any messages sent from the sender client.

The messages received are stored in a cloud database. This requires the creation of a database service in Bluemix. In this chapter, we opted for a Cloudant NoSQL database service.

Creating a Cloudant service in Bluemix is similar to how we created the Secure Gateway service in 5.2.3, “Writing a simple Node.js IBM MQ Light message sender client” on page 92. When you create the Cloudant service, begin to develop the receiver application by following these steps:

1. To begin, create a new project folder. In this folder, the `app.js` JavaScript file that was required in the sender client is also required. Two additional files are also required:

   – `manifest.yml`: Tells Bluemix how to deploy the application.

   – `package.json`: For defining any Node.js package dependencies.

   And create one additional folder inside the project folder:

   – `public`

2. First, edit the `package.json` file and add the below code. Here, any dependencies your cloud applications require are defined in the dependencies JSON object (highlighted in red below). To use the popular express framework also requires the express module. To do this, add the line “*express”: “4.12.x*”. This tells Bluemix when deploying the application to deploy the Node.js express module as well, with a version number of 4.12.x.

3. The following modules: *cfenv* module, *mqlight* module, *body-parse* module, *http* module, and the *nano* module are also required.

> **Note:** More information about Node.js packages that are available can be found at:
>
> `https://www.npmjs.com`

Example 5-3 shows the `package.json` file.

Example 5-3 The package.json file

```
{
   "name": "CloudIntegrationApp",
   "version": "0.0.1",
   "description": "A simple nodejs receiver client for Bluemix",
   "scripts": {
      "start": "node app.js"
   },
   "dependencies": {
      "express": "4.12.x",
      "cfenv": "1.0.x",
      "mqlight": "*",
      "body-parser": "*",
      "http": "*",
      "nano": "*"
   },
   "repository": {},
   "engines": {
      "node": "0.12.x"
   }
}
```

4. Also needed is the addition of the following lines to your `manifest.yml` file, as shown in Example 5-4.

Example 5-4 The manifest.yml file

```
applications:
- path: .
  memory: 256M
  instances: 1
  domain: mybluemix.net
  name: RedBookExampleCloudIntegration
  host: redbookexamplecloudintegration
  disk_quota: 1024M
  services:
  - Cloudant NoSQL DB-gl
  env:
    mqService: amqp://cap-sg-prd-2.integration.ibmcloud.com:15443
```

The important parts to note are the *services:* and the *env:* blocks. Under the "services:" line, we are binding the Cloudant service that we created earlier to our application so that our application can use it.

Attention: The Cloudant service being bound requires the name of the service you are wanting to bind as it appears in your Bluemix dashboard. In this instance, the default name of the service created was *Cloudant NoSQL Db-gl*. When creating the service, you can use a custom name if you want.

The second important part is the line under *env:*. Here, we are defining an environment variable called **`mqService`** and setting its value to the cloud host url : port that we noted earlier in section 5.4, "Using IBM Secure Gateway to connect your on-premises sender client to the cloud application" on page 97. See Figure 5-15 on page 101.

Attention: This is how to define environment variables in your cloud environment. For this example, the name of the environment variable is **`mqService`**.

Any environment variable that we define in this block can be used by the application defined in the same `manifest.yml` file.

To define a custom environment variable, you need to define the variable *name* followed by "**:**" then the *value* of the variable.

5. It is now time for the code. Open your `app.js` file and paste the below code (Example 5-5).

Example 5-5 Sample Node.js message receiver client

```
//mqService configuration object
var mqService = {
};

//database configuration object
var dbCredentials = {
   dbName : 'messages'
};
var cloudantDB;
var db;
```

```
//Module Dependencies
var express = require('express');
var cfenv = require('cfenv');
var app = express();
//Serve files out of the public folder
app.use(express.static(__dirname + '/public'));
// Get the app environment from Cloud Foundry
var appEnv = cfenv.getAppEnv();
var http = require('http');
var bodyParser = require('body-parser');
var mqlight = require('mqlight');

//Set a variable to Environment variable (Cloud Service URL : PORT)
if(process.env.mqService)
{
        mqService.url = process.env.mqService;
}

//Process Cloudant Database service credentials
if(process.env.VCAP_SERVICES)
{
      var vcapServices = JSON.parse(process.env.VCAP_SERVICES);
      if(vcapServices.cloudantNoSQLDB)
      {
         dbCredentials.host = vcapServices.cloudantNoSQLDB[0].credentials.host;
         dbCredentials.port = vcapServices.cloudantNoSQLDB[0].credentials.port;
         dbCredentials.user =
vcapServices.cloudantNoSQLDB[0].credentials.username;
         dbCredentials.password =
vcapServices.cloudantNoSQLDB[0].credentials.password;
         dbCredentials.url = vcapServices.cloudantNoSQLDB[0].credentials.url;
      }
}

//Require nano, and set up a db object wrapper
cloudantDB = require('nano')(dbCredentials.url);





//Try create Database if it doesn't exist.
nano.db.create('dbCredentials.dbName', function(err, body) {
   if (!err) {
    console.log('Database alice created!');
  }
   else {
    console.log('Database already exists!');
  }
});
db = cloudantDB.use(dbCredentials.dbName);

//Create a reciever mqlight client Note: the mqService.url we are passing is as a
parameter
var recvClient = mqlight.createClient({service: mqService.url}, function() {
  //Set client and subscribe to a topic. In this case 'public'
  //recvClient.on callback is a function that puts messages in a variable
  recvClient.on('message', processMessage);
  recvClient.subscribe('public', function(error) {
        if (error) console.error("Failed to subscribe: " + error);
        else {
          console.log("Subscribed");
        }
      });
});

//Function to parse messages sent.
var heldMsg;
var heldMsgs = [];
function processMessage(data, delivery) {
  try {
    data = JSON.parse(data);
    console.log("Received response: " + JSON.stringify(data));
   console.log(data);
  } catch (e) {
    // Expected if we're receiving a Javascript object
  }
   heldMsg = {"data" : data, "delivery" : delivery};
   //Insert messages into our Cloudant DB
   db.insert(heldMsg, function(err, body) {
   if (!err)
    console.log(body)
   })

  heldMsgs.push({"message" : data});
}

app.use(bodyParser.json());

//Define an express route to send messages to browser in json format
app.get('/messages', function(req,res) {
  var msg = heldMsgs;
  if (msg) {
    res.json(msg);
   heldMsgs = [];
  }
  else {
    res.writeHead(204);
    res.end();
  }
});

// start server on the specified port and binding host
app.listen(appEnv.port, appEnv.bind, function() {
// print a message when the server starts listening
  console.log("server starting on " + appEnv.url);
});
```

5.5.2 Pushing the application to the Bluemix cloud environment

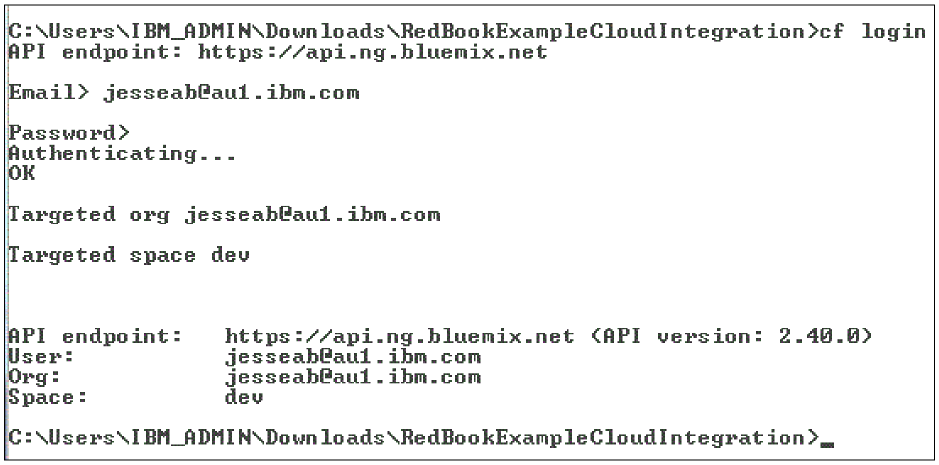
It is time to push the application to Bluemix. This can be done in multiple ways, but we use the Cloud Foundry command-line interface (CLI) by doing the following steps:

1. If you have not done so yet, install the Cloud Foundry command-line interface. Instructions can be found here:

   https://docs.cloudfoundry.org/devguide/installcf/install-go-cli.html

2. After the Cloud Foundry CLI has been installed, go to your project folder and open a command-line interface.

3. Log in to Cloud Foundry by typing **`cf login`**. Then, enter the API endpoint (if you have not logged in previously) as https://api.ng.bluemix.net. You are then prompted for your Bluemix username (email) and password. See Figure 5-16.



```
C:\Users\IBM_ADMIN\Downloads\RedBookExampleCloudIntegration>cf login
API endpoint: https://api.ng.bluemix.net

Email> jesseab@au1.ibm.com

Password>
Authenticating...
OK

Targeted org jesseab@au1.ibm.com

Targeted space dev



API endpoint:   https://api.ng.bluemix.net (API version: 2.40.0)
User:           jesseab@au1.ibm.com
Org:            jesseab@au1.ibm.com
Space:          dev

C:\Users\IBM_ADMIN\Downloads\RedBookExampleCloudIntegration>_
```

Figure 5-16 Example Cloud Foundry CLI login

4. When logged in, type the **`cf push RedBookExampleCloudIntegration`** command to push your application to the cloud.

   Important: *RedBookExampleCloudIntegration* is the host name of the application being pushed and as such in the context of the Bluemix domain, must be unique. You are required to use your own application name here. The application host and name are defined in the `manifest.yml` file and should be unique.

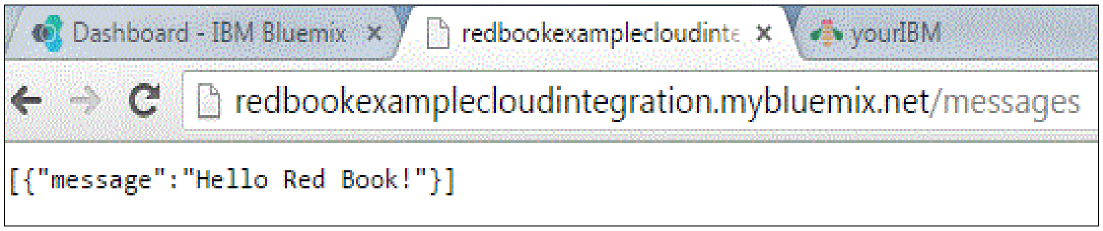
   Cloud Foundry and Bluemix take care of pushing and staging your application based on the settings and configurations you entered in the `package.json` and `manifest.yml` files. When it has completed uploading and staging your application, we have completed the scenario and now should have a working event driven messaging system that allows us to send a message from our on-premises environment and receive the message in our Bluemix cloud application.

Attention: If you are using the Docker client on your on-premises machine, you might be required to modify the access control list to allow the Bluemix Cloud Application to connect through the secure tunnel. To do this, run the Docker client, and enter the Docker CLI after running the previous Docker command. For this example, type **`acl allow :`**, which allows connections to be established from any host:port combination. This is not recommended in a production environment due to security concerns. And the host IP and port should be restricted to either known combinations or a range of combinations that will be accessing the on-premises environment.

5.6 Testing end to end

If the application is showing as down, or is crashing, most likely the issue is that the secure tunnel is not connected on the other end. Do this before restaging the application in Bluemix:

1. First, run the `app.js` file on-premises to send a message to the IBM MQ Light or ActiveMQ Apollo topic.
2. Ensure that the message has been picked up by:
   a. The IBM MQ Light topic in the IBM MQ Light dashboard. This can be accessed at l*ocalhost:9180/#page=home* if using default configurations.
   b. The ActiveMQ Apollo topic in the Apollo dashboard. This can be accessed at `http://127.0.0.1:61680/` if using default configurations.
3. Go to your Bluemix application URL and route append the api route/messages to view if any messages have been picked up by your cloud application. They should be displayed in an array of JSON objects. See Figure 5-17.



```
[{"message":"Hello Red Book!"}]
```

Figure 5-17 Sample display of messages received by cloud Node.js receiver client

If everything has been done correctly, the messages (in array of JSON objects) should be returned to your browsers. The messages are stored in the Cloudant NoSQL Database.

4. The API route that we defined in our express application is */messages.* Therefore, go to `http://redbookexamplecloudintegration.mybluemix.net/messages` to view any messages received.

   Attention: Remember that the Bluemix route defined in this example uses the host/application name that we defined in the `manifest.yml` file. The route you access will be unique and contain the application name of your choice in the place of *redbookexamplecloudintegration.*

5. We can also now check to see if the message was recorded in our Cloudant Database Service. Back at the Bluemix dashboard, click your Cloudant Service. Click the **Launch** icon to be taken to your Cloudant Service Dashboard. Go to your **messages** database, and confirm that you inserted a document with the data field being the message you sent.

6

Asynchronous processing through IBM MQ Light service

This chapter describes the business scenario for a market research company that collects data about people's preferences using a voting application. The original solution becomes a victim of its own success and starts experiencing performance problems. This chapter walks through an approach for improving the scalability and performance of the application using IBM MQ Light. The improved solution is then seamlessly integrated with the company's existing IBM MQ messaging infrastructure. This approach allows the capabilities of the IBM Integration Bus to be leveraged to deliver additional functionality to users. The users can see the current leader board as well as the list of votes by voter choice.

Recording of scenarios: You can find the recording of the scenarios described in this chapter at the following link:

`https://youtu.be/FrQz1kpoAAk`

This chapter has the following sections:

6.1 Solution background

This chapter describes the business scenario for a market research company that wants to grow its market share.

The company wants the best ideas to promote their business. They use the IBM Bluemix innovation platform and conduct a hackathon to come up with the best idea.

A voting app wins the judges' approval. The voting app has the following features:

- ► The app can be run on a browser in a desktop or mobile device.
- ► Votes can be cast by a click of a button. The participant provides their name, email address, and phone number, along with their choice from a prepopulated list.
- ► Each vote is recorded to a relational database.
- ► To encourage users to vote, all votes go into the draw for periodic prizes.

The company developed a born-on-the-cloud application using the microservices architecture, as shown in Figure 6-1.

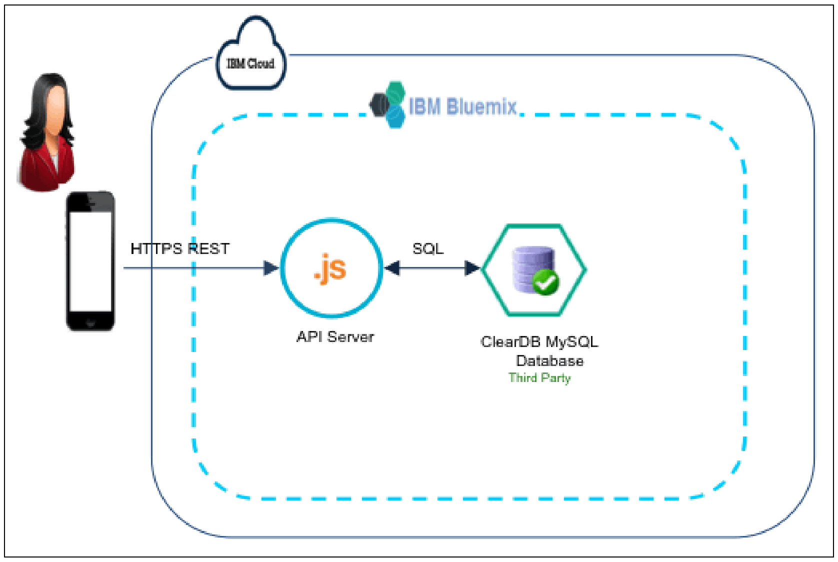


Figure 6-1 Background scenario

The company validates the concept using trusted and known parties to test the sample application. The company is now ready to transform the concept into an enterprise application.

6.2 Transformation of the sample application

This chapter walks through the various stages of transformation that enable the solution to become a hybrid enterprise application. Each subscenario enhances the capabilities delivered by the previous subscenario.

The following subscenarios are discussed:

1. The web application invokes a Representational State Transfer (REST) application programming interface (API). The REST API saves data directly on MySQL database inside the corporate data center. Secure connectivity is provided from IBM Bluemix to the corporate data center using the IBM Secure Gateway Bluemix service. This subscenario ensures that the data captured from an external event (casting of a vote) is stored on a database located inside the corporate data center.
2. The web application invokes a REST API. The REST API saves the vote to IBM MQ Light Bluemix service. A separate message processing application subscribes to messages on IBM MQ Light Bluemix service and asynchronously writes data to MySQL database. This subscenario improves the scalability and performance of the solution while retaining and improving the user experience.
3. The web application invokes the REST API to save the user vote to IBM MQ inside the corporate data center. A message processing application running inside the corporate data center subscribes to messages on IBM MQ and asynchronously writes data to MySQL database. This subscenario makes it possible to expose the details of an external event (casting of a vote) to other applications in the enterprise.
4. The web application invokes the REST API to save the user vote on to IBM MQ inside the corporate data center. An IBM Integration Bus subscribes to messages on IBM MQ. An IBM Integration Bus flow asynchronously processes each message. The flow persists the data to the Enterprise DB2 server and publishes the current leader board message on IBM MQ. Another Integration Bus flow provides REST service to support the query about a list of votes by voter choice. The web application consumes the additional services to provide the leader board as well as the list of votes by voter choice.

6.3 Subscenario 1: Web application saving data on database

The motivation for this subscenario is to securely store data captured from the user to a database inside the corporate data center.

Figure 6-2 shows the components for this scenario.

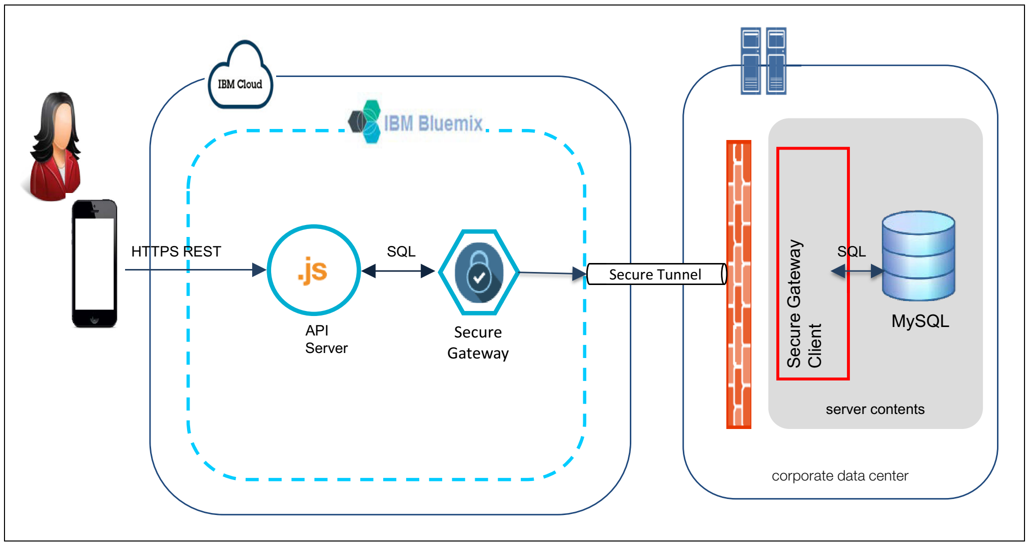


Figure 6-2 Subscenario 1 architecture: Web application saving data to database

In this subscenario, the following steps are executed:

- The web application prompts the user to enter their name, email, phone number, and their preferred candidate. The user clicks **Vote**.
- The web application invokes a REST API on the API server.
- The REST service records a time stamp for the vote, generates a unique key for the database record, and writes the record to the MySQL database.

Important: It is understood that many organizations have multiple firewalls. The Secure Gateway client may be in the DMZ. This means that there might be one or more firewalls between the Secure Gateway client and enterprise server. For the sake of brevity, the diagrams in this chapter show only one firewall.

6.4 Subscenario 2: Web application saving data on IBM MQ Light

The motivation for this subscenario is to allow the application to scale while preserving the user experience and enhancing performance. The performance is improved by offline processing of slower tasks. Figure 6-2 on page 111 shows details.

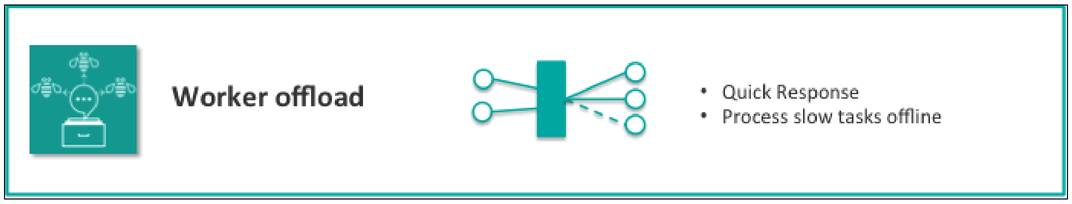


Figure 6-3 Motivation for subscenario

IBM MQ Light is selected because it is simple to use and offers flexible deployment models, as shown in Figure 6-4 on page 113. It is provisioned as an IBM Bluemix service using a subscription model.

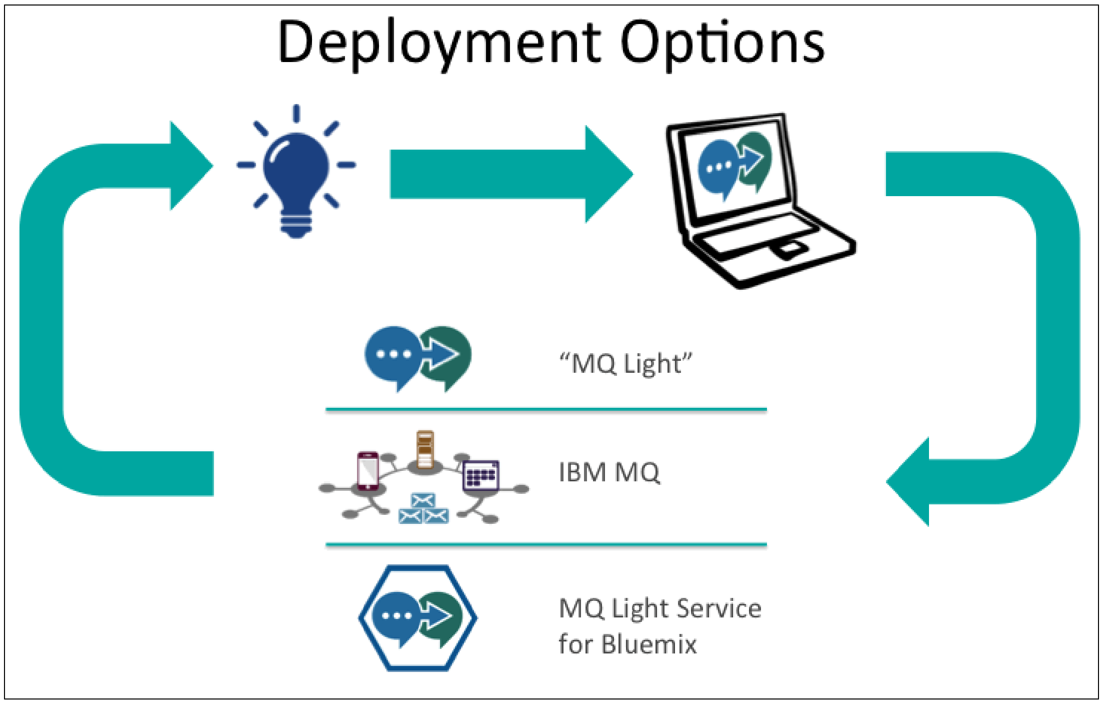


Figure 6-4   IBM MQ Light deployment options

The introduction of IBM MQ Light allows the solution to scale without changing the user interface. Only small modifications to the supporting applications are required.

Figure 6-5 shows the components for this scenario.

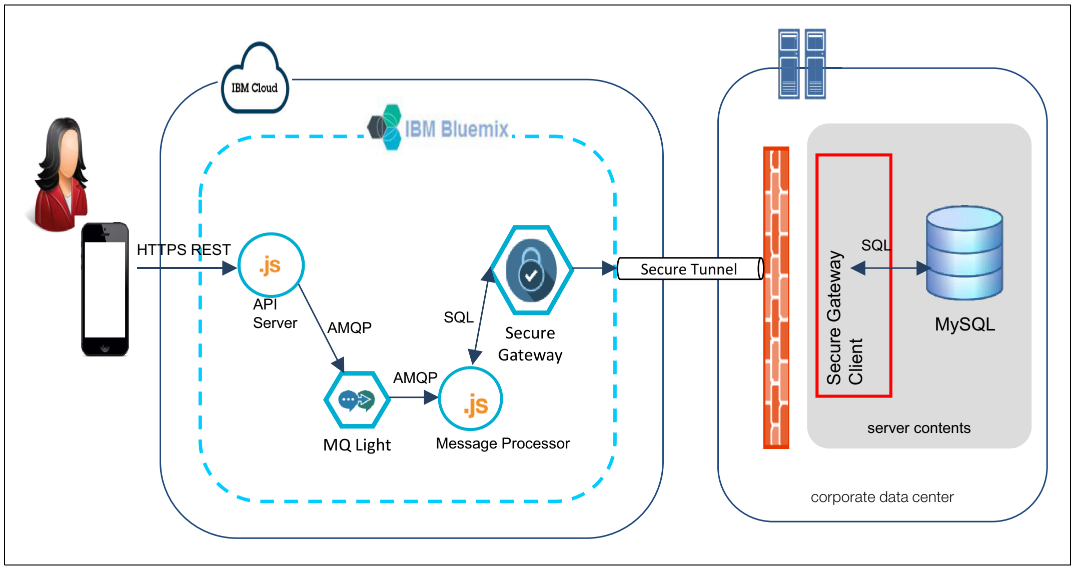


Figure 6-5   Subscenario 2 architecture: Web application saving data to IBM MQ Light

In this subscenario, the following steps are executed:

- The web application prompts the user to enter their name, email, phone number, and their preferred candidate. The user clicks **Vote**.
- The web application invokes a REST API on the API server.
- The REST service records a time stamp for the vote and writes a message to the IBM MQ Light service on Bluemix.

A message-processing application runs in the background. It reads a message from IBM MQ Light service and generates a unique key for the database record, and then writes the record to the MySQL database.

This subscenario offers improved scalability and performance because the database operation that is the slowest part of the transaction for recording a vote is removed from the data capture process. The slow process now runs as a background job.

6.5 Subscenario 3: Web application saving data on IBM MQ server

The motivation for this subscenario is similar to subscenario 2. However, this subscenario also allows the events from a new application (casting of a vote) to be made available for use by other enterprise applications.

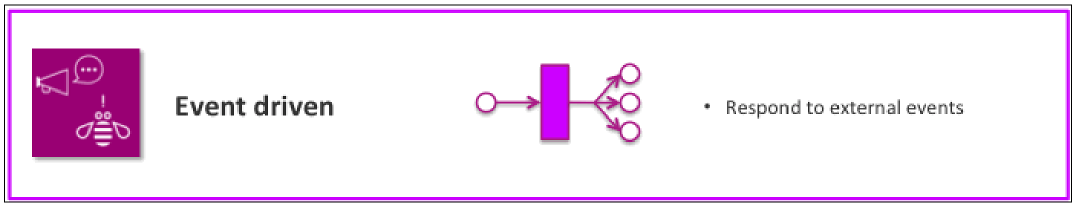


Figure 6-6 Motivation for subscenario 3

The application code developed for subscenario 2 is reused, but reconfigured. IBM MQ is used instead of the IBM MQ Light service on Bluemix. The message processor application runs in the corporate data center and connects directly to the MySQL server instead of a secure tunnel.

Figure 6-7 shows the components for this scenario.

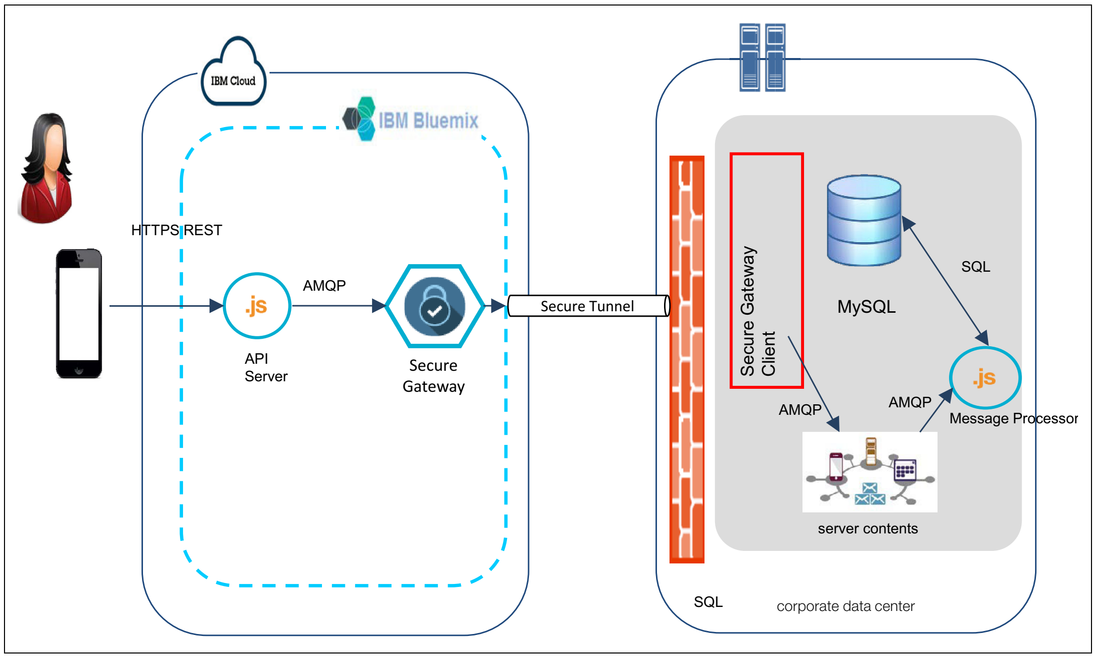


Figure 6-7   Subscenario 3 architecture: Data captured is stored in the IBM MQ server

In this subscenario, the following steps are executed:

- The web application prompts the user to enter their Name, email, phone number, and their preferred candidate. The user clicks **Vote**.
- The web application invokes a REST API on the API server.
- The REST service records a time stamp for the vote and writes a message to IBM MQ running in the corporate data center.
- A message processing application runs in the background inside the corporate data center. It reads a message from IBM MQ and generates a unique key for the database record and writes the record to the MySQL database.

This subscenario offers improved scalability, as in subscenario 2, along with the traditional enterprise-grade high availability and scalability offered by IBM MQ. In addition, the message (containing the vote) is now available to be consumed by any other enterprise application that is connected to IBM MQ.

6.6 Subscenario 4: Web application data consumed by corporate applications

The motivation for this subscenario is to offer all the capabilities of subscenario 3 and, in addition, allow the processing of external events (casting of votes) by the enterprise application. Functionality is added to securely push the internal events (new leader board) to be published back to the application running on cloud in IBM Bluemix, as shown in Figure 6-8.

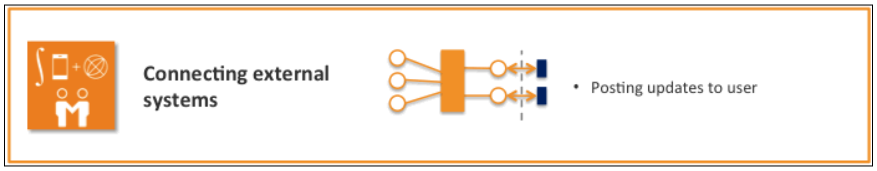


Figure 6-8 Motivation for subscenario 4

The application fulfills the needs of the developer by facilitating a flexible application design that allows events to flow smoothly between the cloud and enterprise applications. In addition, the needs of the infrastructure stake holders are satisfied as the born-on-the-web application is using the enterprise components, thus simplifying the administration as shown in Figure 6-9.

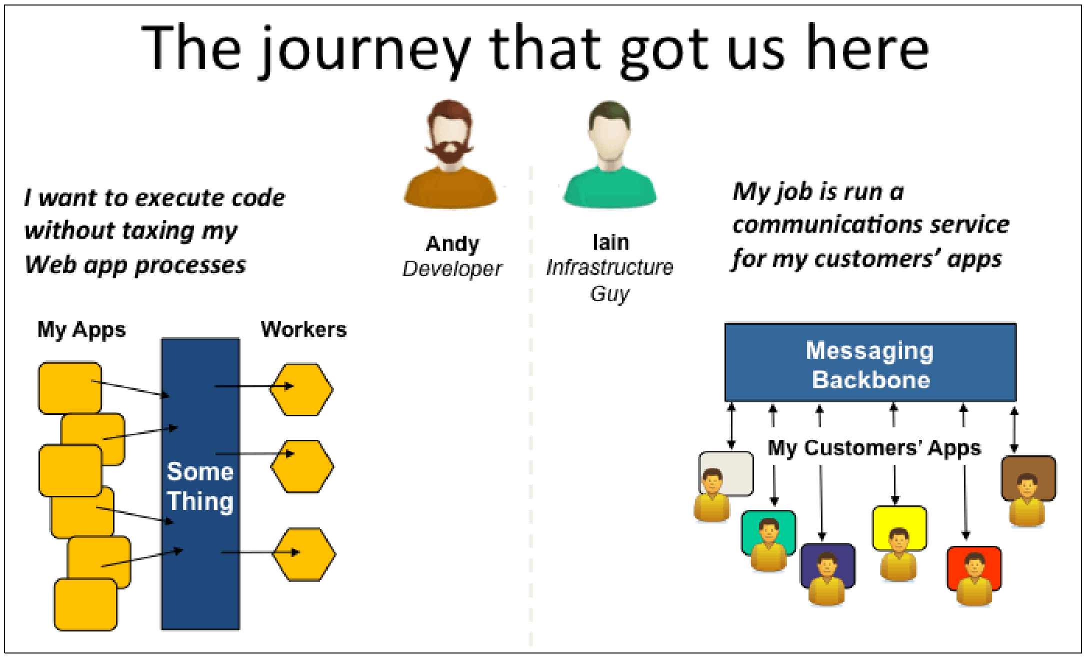

The journey that got us here
I want to execute code without taxing my Web app processes
Andy
Developer
Iain
Infrastructure Guy
My job is run a communications service for my customers' apps
My Apps
Workers
Some Thing
Messaging Backbone
My Customers' Apps

Figure 6-9 Meeting the needs of the developers and infrastructure stakeholders

This subscenario depicts a hybrid cloud application.

Figure 6-10 shows the components used by this subscenario.

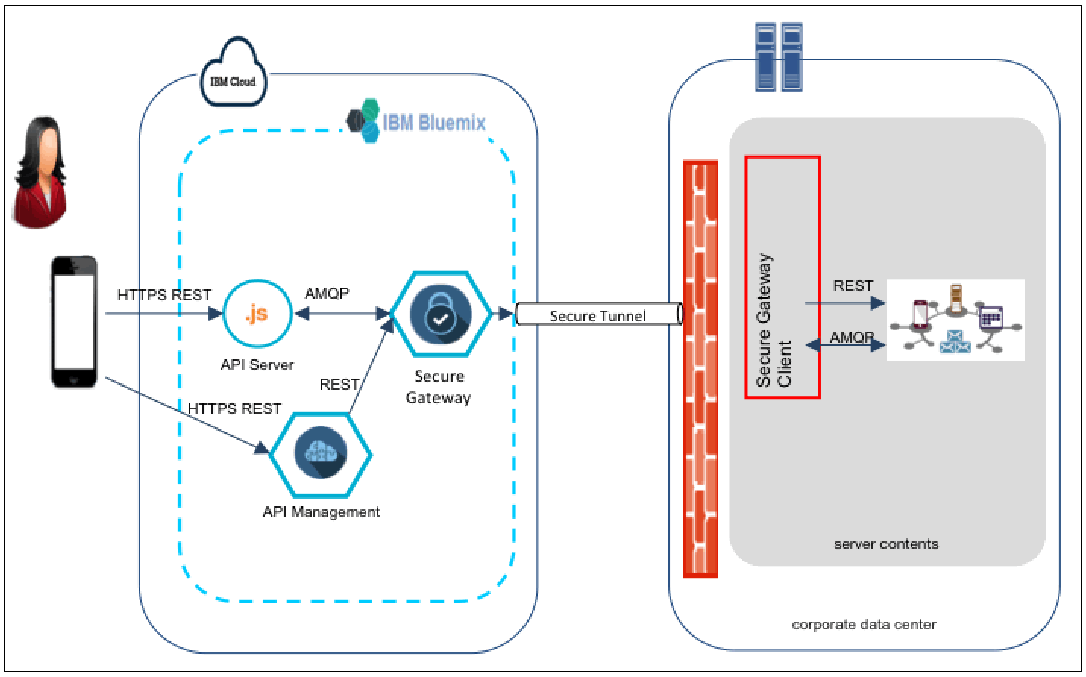


Figure 6-10 Subscenario 4 architecture: Data captured consumed by corporate applications

In this subscenario, the following steps are executed:

- The web application prompts the user to enter their name, email, phone number, and their preferred candidate. The user clicks **Vote**.
- The web application invokes a REST API on the API server.
- The REST service records a time stamp for the vote and writes a message to IBM MQ running in the corporate data center.
- The corporate application running on the IBM Integration Bus processes the messages from IBM MQ. The Integration Bus application stores the data on the corporate database. It also publishes the latest leader board.
- The IBM Integration Bus also provides a REST function that can be used to query details about all votes that have been cast at a point in time.
- The web application can now display a list of votes by voter choice using the REST services exposed using the API Management Bluemix service. The API Management service invokes the REST service on the corporate IBM Integration Bus using the Secure Gateway Bluemix service.
- The API server listens for results published from IBM MQ and caches the latest result. The cached results are returned by the Results REST API.
- The web application uses the Results REST API to display the leader board to the user.

This scenario offers improved scalability, as in subscenarios 2 and 3. Because the message is now being consumed by the enterprise applications, it also provides additional functionality to the user.

6.7 Overview of the sample application

The sample application used for this contains the following components:

- A user interface application
- An API server
- A message processing server running on Node.JS
- MySQL database server
- IBM MQ Light Server on Bluemix
- IBM MQ
- IBM Integration Bus
- IBM DB2 server

6.7.1 User interface application

This application was developed using the Ionic framework. It has been deployed as a web application running on Bluemix. The user interacts with this application to register their vote.

For subscenario 4, the sample application also provides the following additional capabilities:

- Leader board
- List of votes that have been cast for a specific option

6.7.2 API application

This server provides the REST API required by the web application. This application was developed by using the Swagger framework. The application uses the services of MySQL database and IBM MQ Light server or IBM MQ. It was implemented as a Node.JS application running on Bluemix.

In subscenario 2, the application uses the IBM MQ Light service on Bluemix. In subscenarios 3 and 4, it uses IBM MQ in the corporate data center.

The Swagger framework offers a user interface that allows a user to test the API supported by this server. For more information about the Swagger framework, see the following link:

`http://swagger.io`

6.7.3 Message processing application

This application was developed to supplement the API application. The application subscribes to messages deposited by the API application and writes data to the MySQL database. It is implemented as a Node.JS application. In subscenario 2, it runs on Bluemix and uses the IBM Bluemix MQ Light service. However, in subscenario 3, it runs in the corporate data center and uses IBM MQ in the corporate data center.

6.7.4 Enterprise IBM Integration Bus application

This application simulates the capabilities of a typical enterprise service bus in an organization. This is used only for subscenario 4. In this sample, the following capabilities have been used:

- Message flow is provided to persist messages to enterprise database
- Latest results are published
- REST service is provided to support user queries

6.7.5 Preparing for the subscenarios

The sample solution has a number of configuration settings. A configuration setting value can be derived in one step. This can be used in subsequent steps. Table 6-1 shows the parameters that are used by all the subscenarios.

These values are used in subsequent steps, so it can be useful to record them.

Table 6-1 Configuration parameters used by the sample application

Purpose	Parameter	Value
MySQL Database Server Host Name	*<virtual address of the MySQL Server>*	
MySQL database port	3306	
Host Name for API server Node.js application running on IBM Bluemix. Used in all subscenarios.	*<API Server Host Name>*	
Secure Gateway ID for MYSQL Database server. Used in subscenarios 1 and 2	*<Secure GATEWAY ID for Database Server>*	
MySQL Database Host Name when connecting using the Secure Gateway connection. Used in subscenarios 1and 2	*<Cloud Host for MySQL Server>*	
MySQL Database Port when connecting using the Secure Gateway connection. Used in subscenarios 1 and 2	*<Cloud Port for MySQL Server>*	
Host name for the web application. To start the application on a browser use URL *<Web application Host Name>/redbookApp.* Used in all subscenarios	*<Web application Host Name>*	
Secure Gateway ID for IBM MQ server. Used in subscenario 3	*<Secure GATEWAY Id for MQ Server>*	
Host Name or TCP/IP address of IBM MQ server. Used in subscenario 3 and 4	*<TCP/IP Address or Host Name for MQ Server>*	
Port Number for IBM MQ Server Used in subscenario 3 and 4	5672	
Host Name for IBM MQ server when connection using a Secure Gateway tunnel. Used in subscenario 3 and 4	*<Cloud Host for MQ Server>*	
Port number for IBM MQ server when connecting using a Secure Gateway tunnel. Used in subscenario 3 and 4	*<Cloud Port for MQ server>*	

Purpose	Parameter	Value
URL to invoke the REST API on API Gateway. This API provides details of votes by cuisine. Used in subscenario 4	<API Gateway URL for votes by cuisine service>	
Host name of the corporate server where message server is running in corporate data center. Used in subscenario 3	*<Host name or IP address of the corporate message processor>*	
Host name of message processor running on Bluemix. Used in subscenario 2	*<Message Processor Host Name Bluemix>*	
Host name of message processor running on data center. Used in subscenario 3	*<Message Processor Host Name data center>*	

6.8 Implementing subscenario 1

This subscenario requires a combination of IBM Bluemix services and components that exist in the corporate data center. For this overview, IBM Bluemix virtual machines are used to simulate the corporate data center.

The components that are used for this subscenario are shown in Figure 6-11.

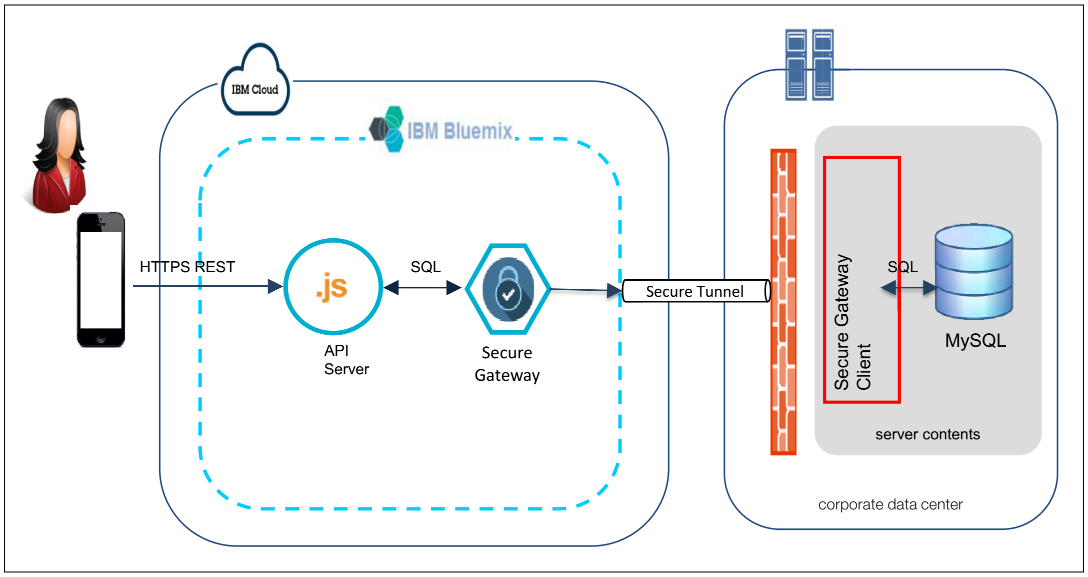


Figure 6-11 Components used for subscenario 1

6.8.1 Create simulated data center database

To demonstrate Secure Gateway, we need a *system of record* running in a private data center. One of those is difficult to download and install. So, for the purposes of these exercises, we simulate one using a MySQL database running in a virtual machine. We create a virtual machine, install Docker, install MySQL running in a Docker container, and initialize the database with some sample data that the Node.js application needs.

Create VM

We create a virtual machine to simulate a private data center. That VM can run anywhere if it has a public IP address (in any cloud provider or on your local computer). For these exercises, we use the virtual machine capability in Bluemix:

1. Create a Secure Shell (SSH) keypair, as documented in "*Creating web applications: Creating a virtual machine: Configuring an SSH security key in a VM: Creating an SSH security key to access a VM*" in the Bluemix documentation:

   https://www.ng.bluemix.net/docs/virtualmachines/vm_index.html#vm_create_ssh_key

   Specifically:

   – We call ours `mqlightkey`

   – In UNIX/Linux: Run **`ssh-keygen -t rsa -f mqlightkey`**

   – In Microsoft Windows: Use PuTTY. You can download it here:

     http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

2. Create a VM on Bluemix, as documented in "*Creating web applications: Creating a virtual machine: Creating a VM in a public cloud*" in the Bluemix documentation:

   https://www.ng.bluemix.net/docs/virtualmachines/vm_index.html#vm_create_public_cloud

   Specifically:

   – To create the VM, use the settings shown in Table 6-2. For the settings that have default values, use those values.

   – To specify the security key, select **Add Key** to import your `mqlightkey` key.

Table 6-2 Virtual machine creation settings

Property	Value	Default
VM cloud	IBM Cloud Public	default
Initial instances	1	default
Assign public IP addresses	Select (yes)	default
VM image	Ubuntu 14.04	default
VM group name	`*AAA*_MQLIGHT`	
VM size	m1.small	default
Security Key	`mqlightkey`	
Network	private	default

When the VM is created, make note of its public IP address, whose form is 129.xxx.xxx.xxx. (The other IP address, 192.168.xxx.xxx, is private.) We refer to this public IP address as <*virtual machine's IP address*>. We refer to it as the <*virtual address of the MySQL Server*>.

3. Log in to the virtual machine. The image has a user that is predefined for logging in remotely; it is `ibmcloud`. Use the authentication key specified when creating the VM:

   ```
   $ ssh -i mqlightkey ibmcloud@<virtual machine's IP address>
   ```

   You now have a running Ubuntu VM and you can log in to it.

Install Docker

You need to install a MySQL database. To simplify that installation, we use MySQL that is already installed in a Docker container. So a good reason that we need our VM to run the Docker runtime is so that it can run the MySQL container.

For information, see the following site:

"Installing Docker on Ubuntu" explains how to install Docker:

https://docs.docker.com/installation/ubuntulinux

Log in to your VM using SSH, as described above, and perform the following commands:

1. Before installing any software, ensure that your Ubuntu installation is running the latest version of all of its packages. Run this command:

   ```
   $ sudo apt-get update
   ```

2. Install the Docker package:

   ```
   $ wget -qO- https://get.docker.com/ | sh
   ```

3. Verify that Docker is installed correctly:

   ```
   $ sudo docker run hello-world
   ```

4. When hello-world runs correctly, part of the output should say:

   ```
   Hello from Docker.
   This message shows that your installation appears to be working correctly.
   ```

When you can run hello-world successfully, your VM has the Docker runtime installed and running correctly.

Install and configure MySQL

To simulate an enterprise database of record, we use a MySQL database with a small, simple data set. To initialize that database, we need a schema file and a data file.

Create the database files

Log in to your VM by using SSH, as described above, and perform the following commands:

1. Create the directory for the database initialization files:

   ```
   $ mkdir ~/mqlight-sql
   $ cd ~/mqlight-sql
   ```

2. By using your favorite Linux text editor (such as `nano` or `vi`), create the file `mqlight-schema.sql` and insert the contents shown in Example 6-1.

Example 6-1 Contents of mqlight-schema.sql

```
DROP SCHEMA IF EXISTS mqlight;
CREATE SCHEMA mqlight;
USE mqlight;
   CREATE TABLE `VOTES`
   (
   referenceNumber char(20),
   name varchar(255),
   email varchar(255),
   phone varchar(255),
   voterChoice varchar(255),
   votetimeStamp varchar(255),
   PRIMARY KEY (referenceNumber)
   ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

3. Create the file `mqlight-data.sql` and insert the contents shown in Example 6-2.

Example 6-2 Contents of mqlight-data.sql

```
USE mqlight;
INSERT INTO `VOTES` (`referenceNumber`,`name`,`email`,
`phone`,`voterChoice`,`votetimeStamp`) VALUES ("Key-001", "James", "a@a.com",
"99991234", "Indian", "2015-11-15");
INSERT INTO `VOTES` (`referenceNumber`,`name`,`email`,
`phone`,`voterChoice`,`votetimeStamp`) VALUES ("Key-002", "John", "b@b.com",
"99991235", "French", "2015-11-16");
INSERT INTO `VOTES` (`referenceNumber`,`name`,`email`,
`phone`,`voterChoice`,`votetimeStamp`) VALUES ("Key-003", "Jack", "c@c.com",
"99991236", "Turkish", "2015-11-17");
```

4. Confirm that you have the files in the correct directory. It should look like this:

```
ibmcloud@aaa-to-do-12345678:~$ ls -l /home/ibmcloud/mqlight-sql/
-rw-r--r-- 1 ibmcloud ibmcloud 227 Jan 1 12:00 mqlight-data.sql
-rw-r--r-- 1 ibmcloud ibmcloud 190 Jan 1 12:00 mqlight-schema.sql
```

Tip: You can also **`cat`** each file to ensure that its contents look correct.

You now have the schema and data file that is needed to initialize the database when creating its Docker container.

Install the database

While still logged in to your VM using SSH:

1. Create the MySQL container instance and load the sample data from the two initialization files with the command below:

```
$ sudo docker run -d --name mysql-tutum -p 3306:3306 -v
/home/ibmcloud:/home/ibmcloud -e MYSQL_PASS=passw0rd -e
STARTUP_SQL="/home/ibmcloud/mqlight-sql/mqlight-schema.sql
/home/ibmcloud/mqlight-sql/mqlight-data.sql" tutum/mysql
```

Where:

– –d runs the container in the background, not interactively
– `mysql-tutum` is the name to give the container that is created from the image
– `3306:3306` forwards the MySQL port to make it accessible from the host OSs IP address
– `/home/ibmcloud:/home/ibmcloud` binds the directory to make the directory on the host OS available within the container
– `MYSQL_PASS` sets the password of the database's main user, in this example to `passw0rd`
– `STARTUP_SQL` tells the container to run the SQL files in the order specified via the space-separated list
– `tutum/mysql` is the name of the Docker image to create the container from

You now have a running Docker container named `mysql-tutum`. That container has a MySQL database server running in it, bound to port `3306`. The database server contains a database named `mqlight` that contains a table named `VOTES` that contains the sample data for votes.

6.8.2 Provision Bluemix services

Perform the following steps to provision Bluemix services:

1. Open the following URL:

   https://console.ng.bluemix.net

2. If you do not have an IBM Bluemix ID, click **Signup** to create your account. Follow instructions provided on the page to create your new account.
3. Log in to your Bluemix account by using your IBM user ID and password.
4. Click **CATALOG**.
5. Scroll down the page and click **MQ Light**. See Figure 6-12.

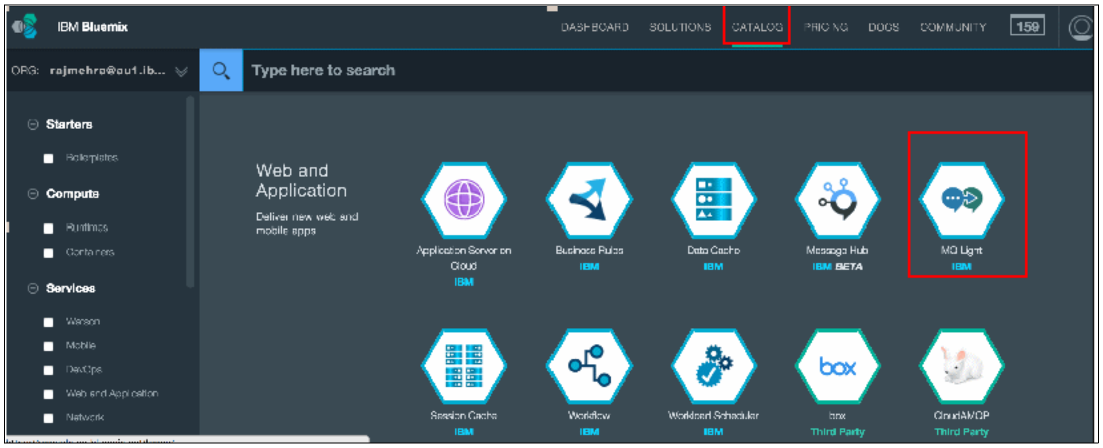
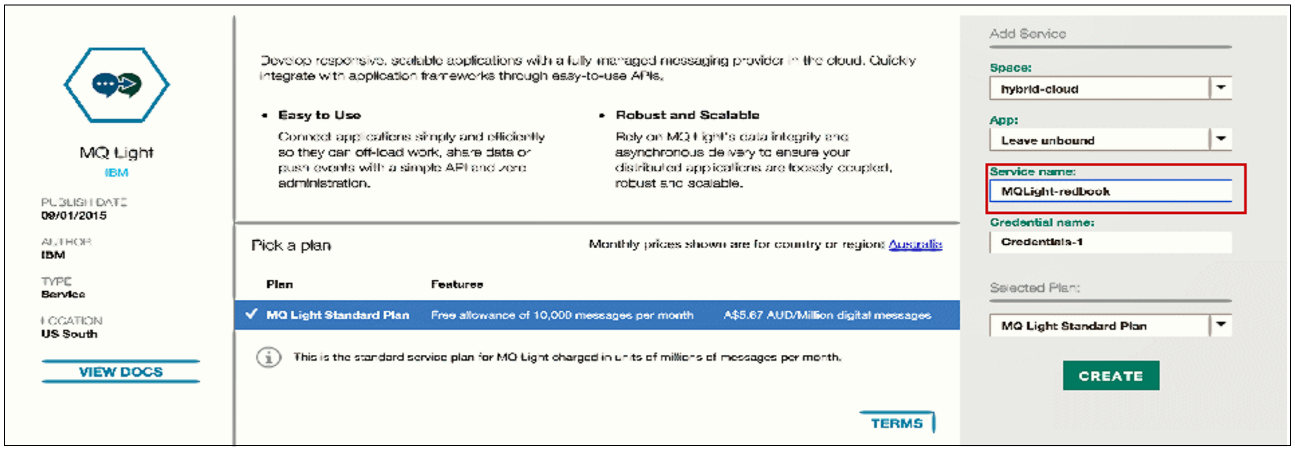


Figure 6-12 Provisioning IBM MQ Light Bluemix service

6. Enter “Service name” as MQLight-redbook. Click **CREATE**. See Figure 6-13.



MQ Light
IBM

PUBLISH DATE
09/01/2015

AUTHOR
IBM

TYPE
Service

LOCATION
US South

VIEW DOCS

Develop responsive, scalable applications with a fully managed messaging provider in the cloud. Quickly integrate with application frameworks through easy-to-use APIs.

- **Easy to Use**
  Connect applications simply and efficiently so they can off-load work, share data or push events with a simple API and zero administration.
- **Robust and Scalable**
  Rely on MQ Light's data integrity and asynchronous delivery to ensure your distributed applications are loosely coupled, robust and scalable.

Pick a plan

Monthly prices shown are for country or region: Australia

Plan	Features	
✓ MQ Light Standard Plan	Free allowance of 10,000 messages per month	A$5.67 AUD/Million digital messages

This is the standard service plan for MQ Light charged in units of millions of messages per month.

TERMS

Add Service

Space: hybrid-cloud

App: Leave unbound

Service name: MQLight-redbook

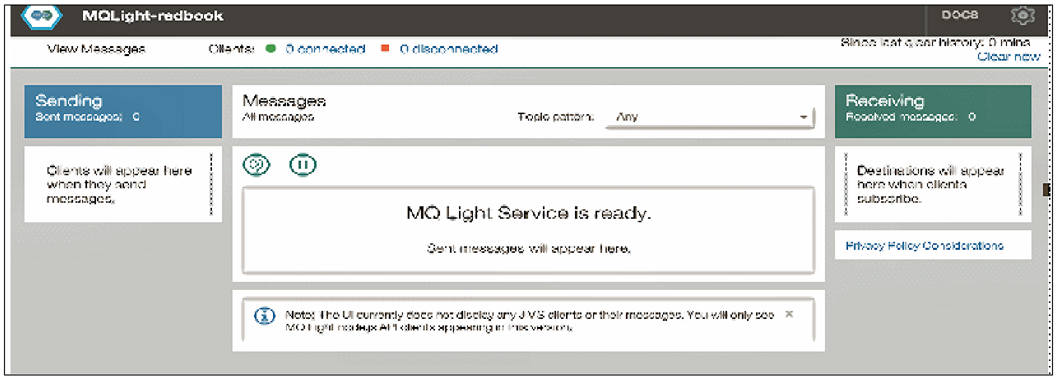
Credential name: Credentials-1

Selected Plan: MQ Light Standard Plan

CREATE

Figure 6-13 Provisioning IBM MQ Light service

The IBM MQ Light console appears as in Figure 6-14.



MQLight-redbook

DOCS

View Messages

Clients: ● 0 connected 🟥 0 disconnected

Since last clear history: 0 mins
Clear now

Sending
Sent messages: 0

Messages
All messages

Topic pattern: Any

Receiving
Received messages: 0

Clients will appear here when they send messages.

MQ Light Service is ready.

Sent messages will appear here.

Destinations will appear here when clients subscribe.

Privacy Policy Considerations

Note: The UI currently does not display any JMS clients or their messages. You will only see MQ Light node.js API clients appearing in this version.

Figure 6-14 IBM MQ Light service console

Attention: The IBM MQ Light Bluemix service is not used in subscenario 1. We just provision the service at this stage. It is used in later steps.

6.8.3 Deploy the API server

The API server is a Node.js application that is available from GIT repository along with the rest of code used by this sample. Download the sample code from this link:

https://github.com/RajMehra/hybrid-cloud-mqlight.git

This code can be downloaded as a .zip file. The .zip file can be decompressed to the local folders. The following subfolders are created:

- voting-services: API Server
- message-processing: Message Processor
- sample-app: The sample application
- IIB-code: The code that runs on IBM Integration Bus used in subscenario 4

Code from these folders is used for the subscenarios. The voting-services folder is used for deploying the API server:

1. Download and install the Cloud Foundry CLI from the following site:

   `https://www.ng.bluemix.net/docs/cli/downloads.html`

2. Follow the instructions to install it on your workstation.
3. Use the `voting-services` folder.
4. Use a text editor of your choice to edit `voting-services/manifest.yml`.
5. Replace with the value that matches your configurations. See Table 6-3.

Table 6-3 Parameters for the API server

Attribute	**Default**	**New value**	**Purpose**
DBHOST	<>	*<virtual address of the MySQL Server>*	Host Name or IP Address of MySQL server
DBPORT	3306	3306	Port number for MySQL database
DBNAME	mqlight	mqlight	MySQL database/schema name
DBUSER	admin	admin	MySQL user
DBPASSWORD	passw0rd	passw0rd	MySQL password
TOPIC	redbook/vote	redbook/vote	Topic from MQ Light/IBM MQ server
MQHOST	<>	<>	IBM MQ Server Host Name or IP server
MQPORT	5672	5672	Port number for IBM MQ server AMQP Listener
MQUSER	<>	<>	User ID to connect to IBM MQ server
MQPASSWORD	<>	<>	Password to connect to IBM MQ server
MQSERVICE	ampq://localhost	ampq://localhost	Local MQ Light server
MQUSESERVER	LOCAL	BLUEMIX	Valid Values LOCAL: Local MQ Light Server BLUEMIX: Bluemix service CLOUD: IBM MQ server
MQID	Send001	Send001	Sender Id
SAVEMETHOD	MQ	DB	Valid Values are: DB: Direct to database MQ: To MQ Light/IBM MQ Server

Attribute	Default	New value	Purpose
RESULTTOPIC	redbook/results	redbook/results	This is used only for subscenario 4 to get the leader board results.
ENABLEENTERPRISE	false	false	The valid values are “true” (only for subscenario 4) “false” for all other subscenarios.

6. Change the host name to unique name. Replace aaa with three unique characters of your choice.

Table 6-4 Parameters table

Attribute	Default	New value	Purpose
host	aaa-redbook-MQLight-service-1	**xxx**-redbook-MQLight-service-1	Unique host name for API server

7. Validate the contents of the file and save it.
8. Open a command prompt on your workstation.
9. Follow the instructions provided at the following link and log in to your Bluemix account by using the **`cf login`** command:

   `https://www.ng.bluemix.net/docs/cli/reference/cfcommands/index.html`
10. On the command prompt, change the current directory to the `voting-services` folder.
11. Enter the **`cf push`** command on the command window. This uploads the application to Bluemix and deploys it.
12. If someone else is using the same host name that you have chosen, the deployment might fail. In that instance, change the host as in step 6 and try step 11.
13. Open the following URL:

    `https://console.ng.bluemix.net`
14. Log in using your IBM user ID and password.
15. Click **DASHBOARD**. An application appears on the dashboard.

16. Click **redbook-MQLight-service-1** as shown in Figure 6-15.

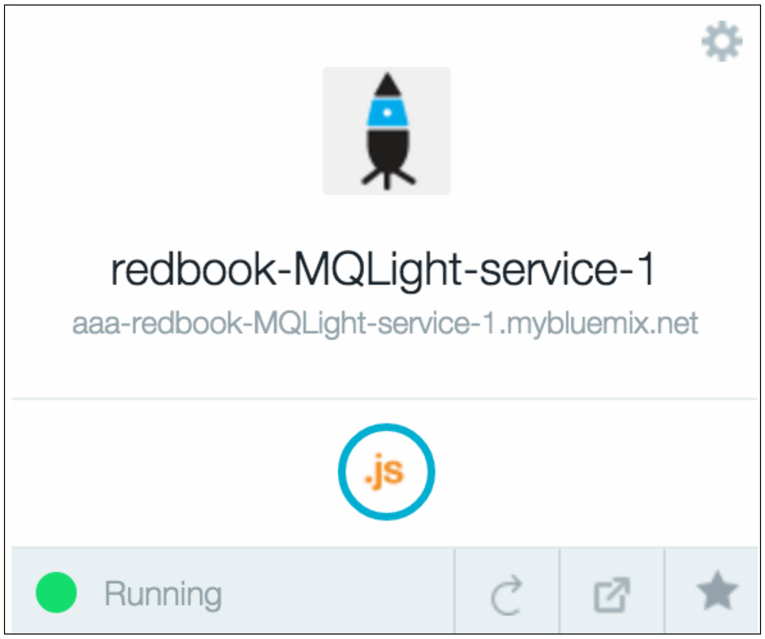

redbook-MQLight-service-1
aaa-redbook-MQLight-service-1.mybluemix.net
.js
Running

Figure 6-15 API server application deployed and running on Bluemix

17. Click **Overview**.

18. Write down the URL next to **Routes** as *<API Server Host Name>*. It will be something like http://aaa-redbook-MQLight-service-1.mybluemix.net/. See Figure 6-16.

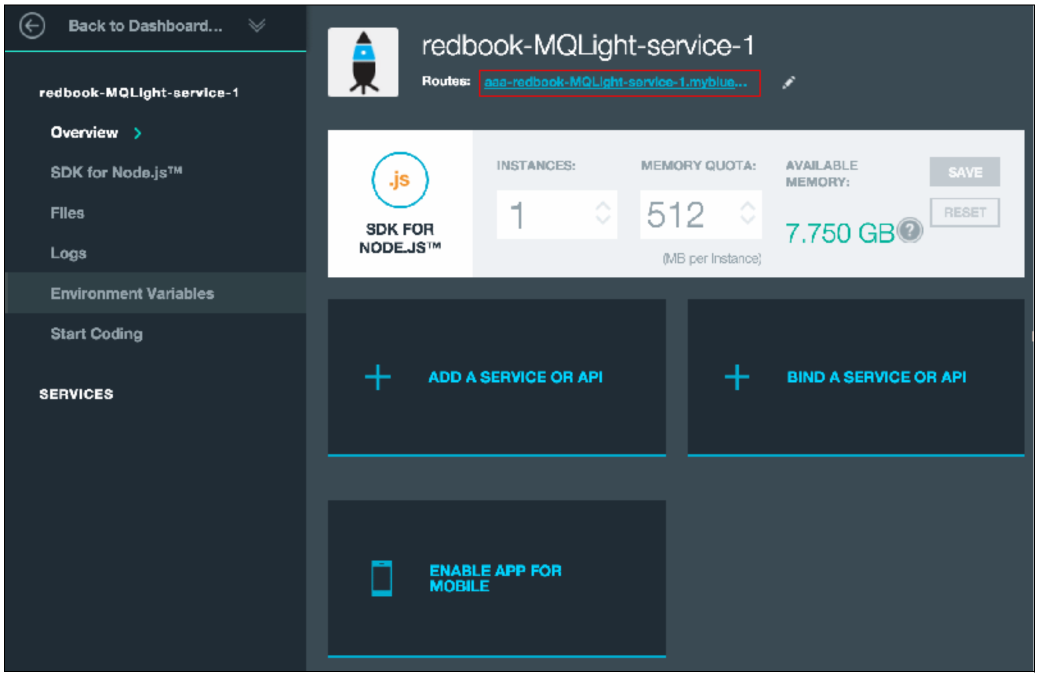
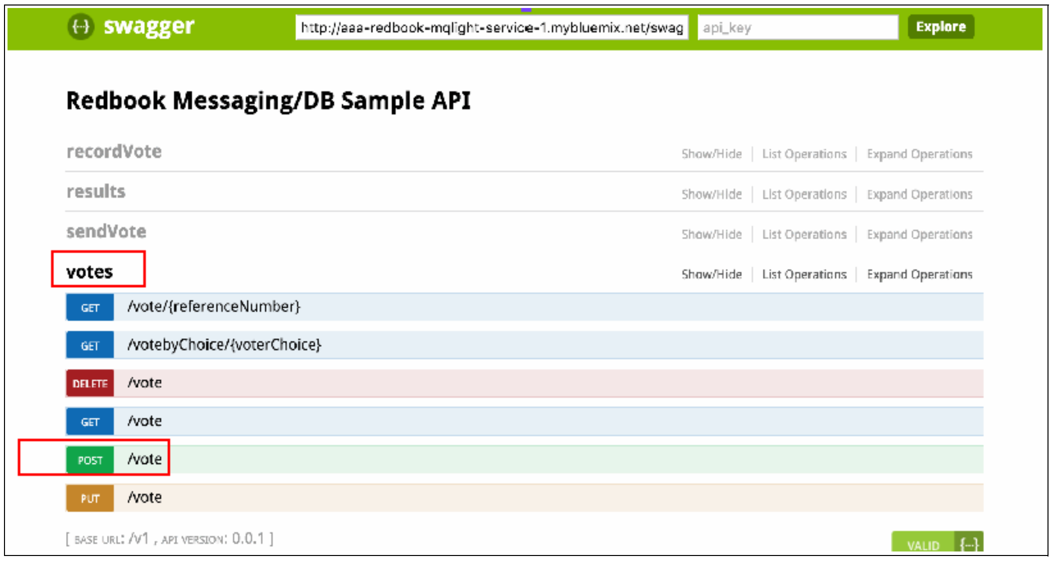


Figure 6-16 API server

19.Open a new browser window or tab and type in url *<API Server Host Name>/api-docs/#.*

API Server Test Console is shown as in Figure 6-17.

20.Click **votes.** This opens a list of APIs for votes.



Redbook Messaging/DB Sample API

recordVote	Show/Hide	List Operations	Expand Operations
results	Show/Hide	List Operations	Expand Operations
sendVote	Show/Hide	List Operations	Expand Operations
votes	Show/Hide	List Operations	Expand Operations

GET	/vote/{referenceNumber}
GET	/votebyChoice/{voterChoice}
DELETE	/vote
GET	/vote
POST	/vote
PUT	/vote

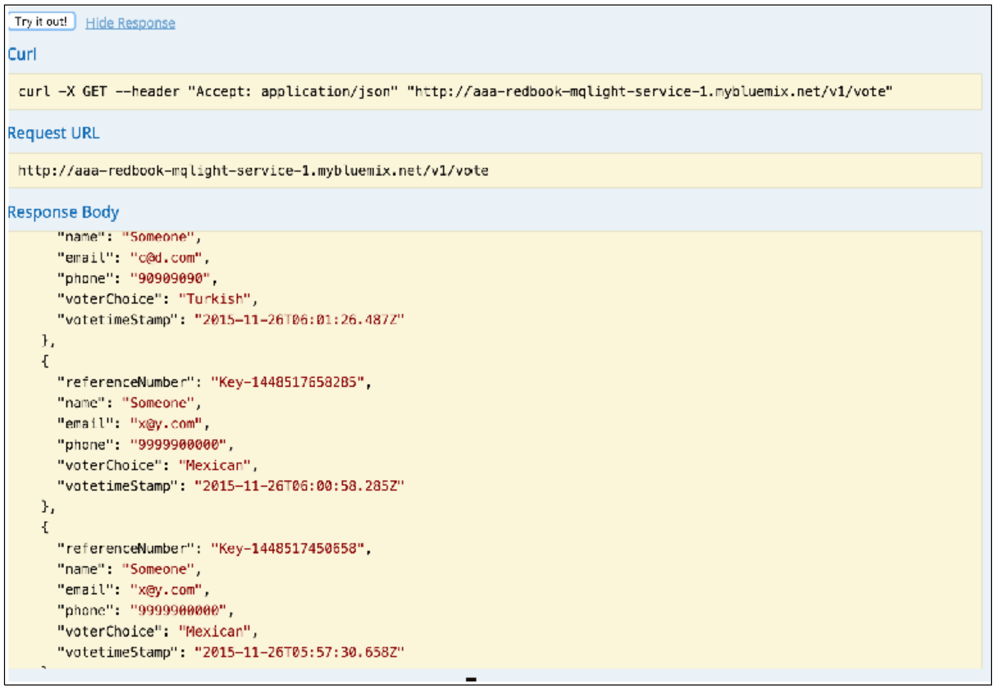
[BASE URL: /v1 , API VERSION: 0.0.1]

Figure 6-17 API Server Test Console

21.Click **GET /votes.** This opens the API details.

22.Click **Try it out!**

23.The database records from the MySQL database are shown as a JSON array in Figure 6-18. Note that the data shown will vary based on data in your database.



Curl

```
curl -X GET --header "Accept: application/json" "http://aaa-redbook-mqlight-service-1.mybluemix.net/v1/vote"
```

Request URL

```
http://aaa-redbook-mqlight-service-1.mybluemix.net/v1/vote
```

Response Body

```
      "name": "Someone",
      "email": "c@d.com",
      "phone": "90909090",
      "voterChoice": "Turkish",
      "votetimeStamp": "2015-11-26T06:01:26.487Z"
    },
    {
      "referenceNumber": "Key-1448517658285",
      "name": "Someone",
      "email": "x@y.com",
      "phone": "9999900000",
      "voterChoice": "Mexican",
      "votetimeStamp": "2015-11-26T06:00:58.285Z"
    },
    {
      "referenceNumber": "Key-1448517450658",
      "name": "Someone",
      "email": "x@y.com",
      "phone": "9999900000",
      "voterChoice": "Mexican",
      "votetimeStamp": "2015-11-26T05:57:30.658Z"
```

Figure 6-18 Results from API details

We have tested the direct connection between the API server and the MySQL database.

6.8.4 Configure a secure tunnel between the API server and database

This section describes the steps required to configure a secure connection between MySQL server and the API server running on Bluemix.

Steps to be completed to Bluemix console

The following steps need to be completed on Bluemix console. It involves setting up a Secure Gateway and adding a destination for the MySQL server.

Setting up Secure Gateway

Perform the following steps to set up the Secure Gateway:

1. Open the following URL:

   `https://console.ng.bluemix.net`

2. Log in using your IBM user ID and password.
3. Click **CATALOG**.
4. Under Services, click **Secure Gateway**. See Figure 6-19.

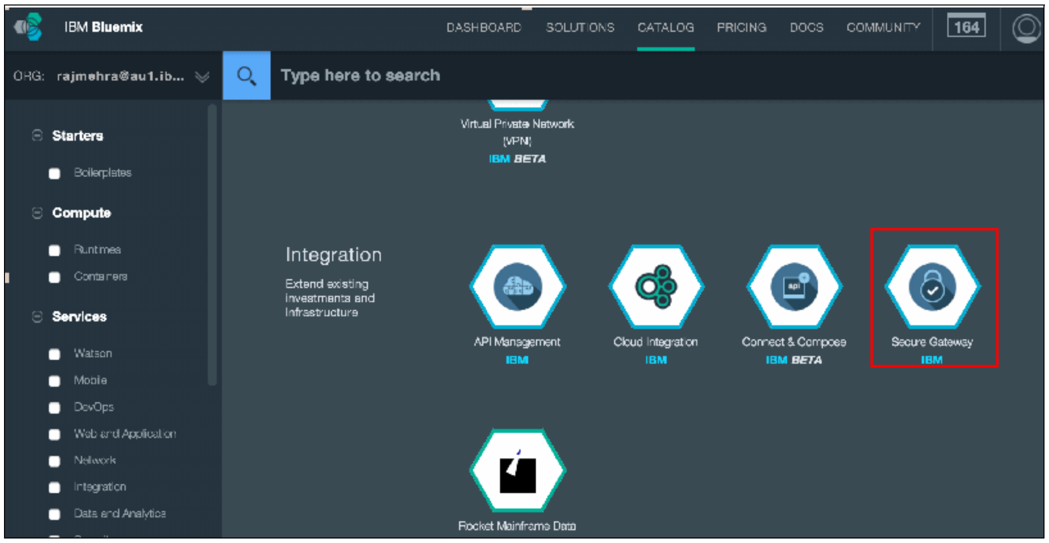
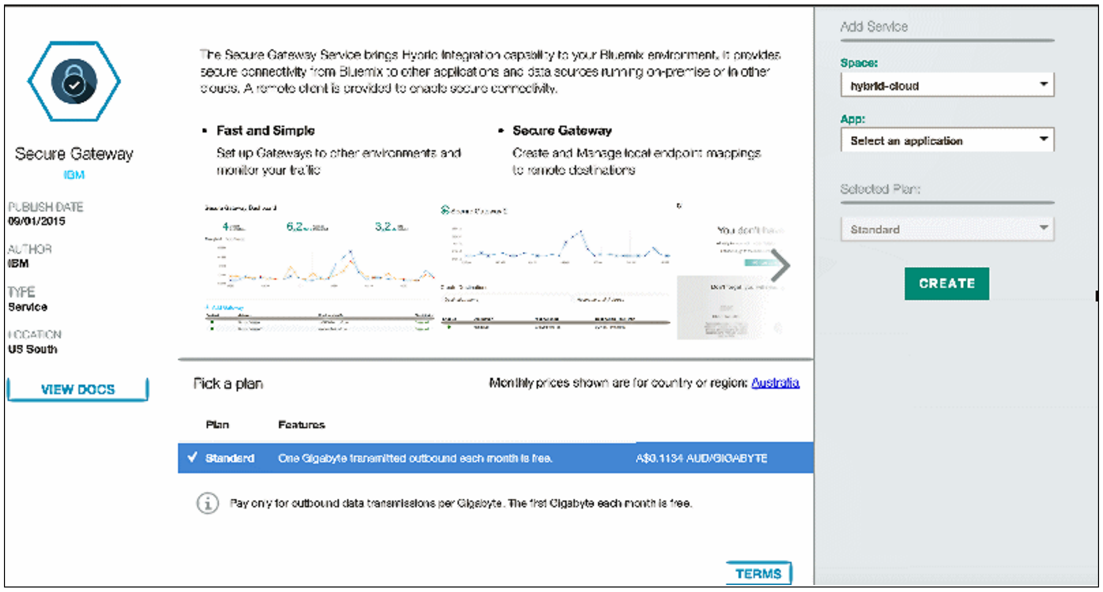


Figure 6-19 Select Secure Gateway service

5. Click **CREATE**. See Figure 6-20.



Secure Gateway
IBM
PUBLISH DATE
09/01/2015
AUTHOR
IBM
TYPE
Service
LOCATION
US South

The Secure Gateway Service brings Hybrid Integration capability to your Bluemix environment. It provides secure connectivity from Bluemix to other applications and data sources running on-premise or in other clouds. A remote client is provided to enable secure connectivity.

- **Fast and Simple**
  Set up Gateways to other environments and monitor your traffic
- **Secure Gateway**
  Create and Manage local endpoint mappings to remote destinations

Pick a plan — Monthly prices shown are for country or region: Australia

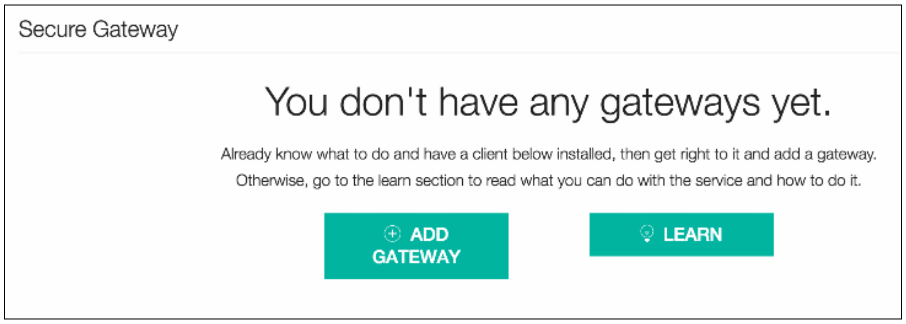
Plan	Features	
✓ Standard	One Gigabyte transmitted outbound each month is free.	A$0.1134 AUD/GIGABYTE

Pay only for outbound data transmissions per Gigabyte. The first Gigabyte each month is free.

Add Service
Space: hybrid-cloud
App: Select an application
Selected Plan: Standard

Figure 6-20 Create Secure Gateway

6. Click **ADD GATEWAY**. See Figure 6-21.



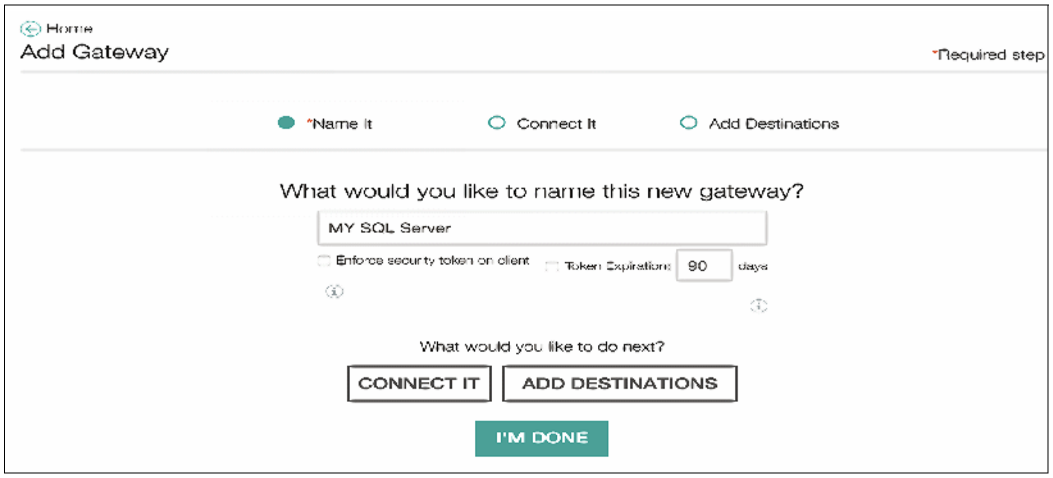
Secure Gateway

You don't have any gateways yet.

Already know what to do and have a client below installed, then get right to it and add a gateway.
Otherwise, go to the learn section to read what you can do with the service and how to do it.

Figure 6-21 Adding Gateway

7. Enter “MY SQL Server” in name, clear **Enforce security token on client** and **Token Expiration**. See Figure 6-22.



What would you like to name this new gateway?

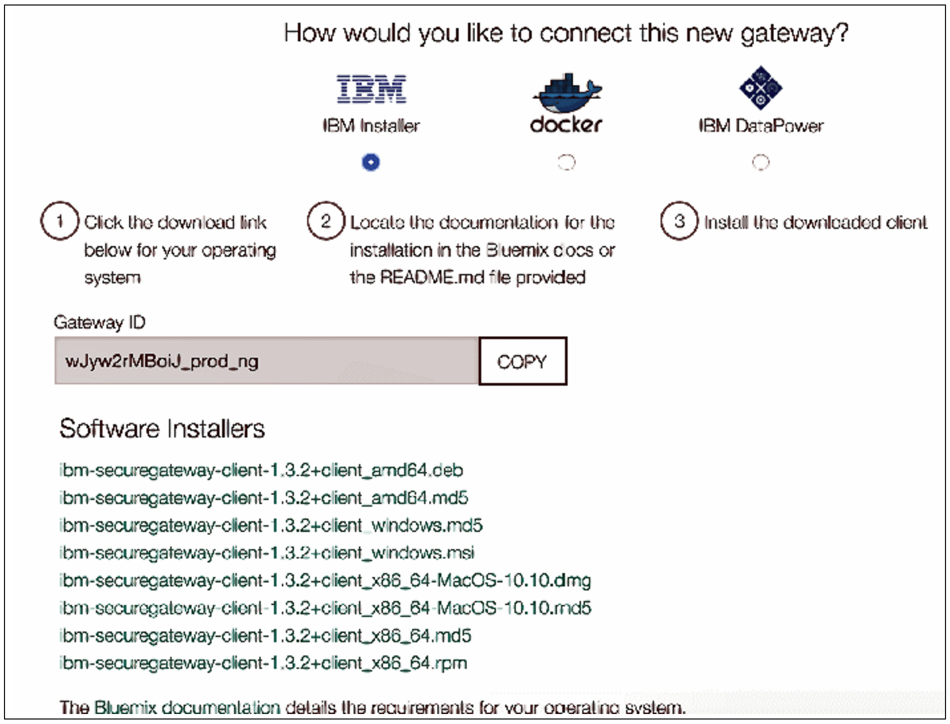
MY SQL Server

Enforce security token on client Token Expiration: 90 days

What would you like to do next?

Figure 6-22   Adding Secure Gateway connection

8. Click **CONNECT IT.**
9. Select **IBM Installer**.
10. The Gateway ID is generated and presented. Click **COPY.** This copies the Gateway ID. Store the Gateway ID as <*Secure GATEWAY ID for Database Server*>*.* See Figure 6-23.


How would you like to connect this new gateway?

1 Click the download link below for your operating system

2 Locate the documentation for the installation in the Bluemix docs or the README.md file provided

3 Install the downloaded client

Gateway ID

wJyw2rMBoiJ_prod_ng

Software Installers

ibm-securegateway-client-1.3.2+client_amd64.deb
ibm-securegateway-client-1.3.2+client_amd64.md5
ibm-securegateway-client-1.3.2+client_windows.md5
ibm-securegateway-client-1.3.2+client_windows.msi
ibm-securegateway-client-1.3.2+client_x86_64-MacOS-10.10.dmg
ibm-securegateway-client-1.3.2+client_x86_64-MacOS-10.10.md5
ibm-securegateway-client-1.3.2+client_x86_64.md5
ibm-securegateway-client-1.3.2+client_x86_64.rpm

The Bluemix documentation details the requirements for your operating system.

Figure 6-23   Generating a Gateway ID

11. Download the appropriate IBM Secure Gateway installer for your in-house server. Store as *<IBM Secure gateway client installer>*. In our example, we use the `.deb` file.
12. Follow instructions at the following URL to install and configure the secure client gateway on your MySQL database server, the one running on *<virtual address of the MySQL Server>*:

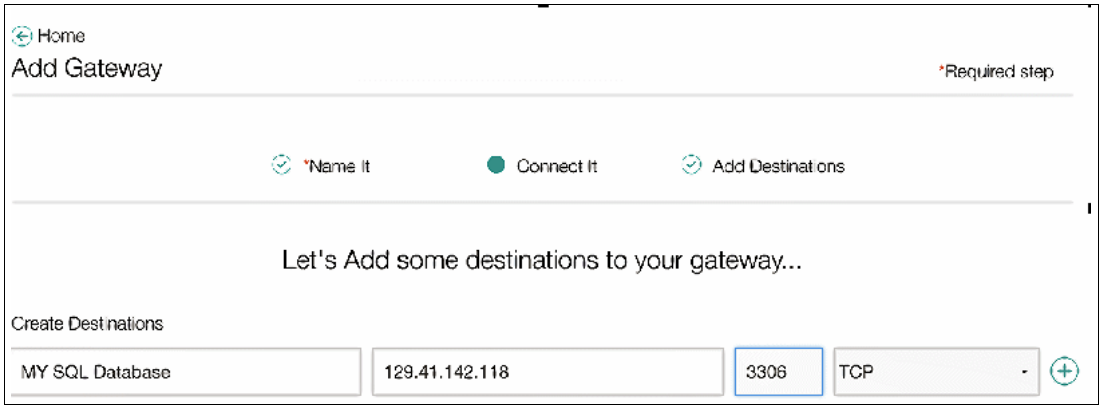
    `https://www.ng.bluemix.net/docs/services/SecureGateway/sg_021.html#sg_025`

Adding destination for MySQL server

Next, we need to add a destination for MySQL server:

1. Scroll to the bottom of the window shown in Figure 6-24 and click **ADD DESTINATIONS**.
2. Enter the Destination name, *<virtual address of the MySQL server>* as HostName, and 3306 as Port, as shown in Figure 6-24. Click **+**.

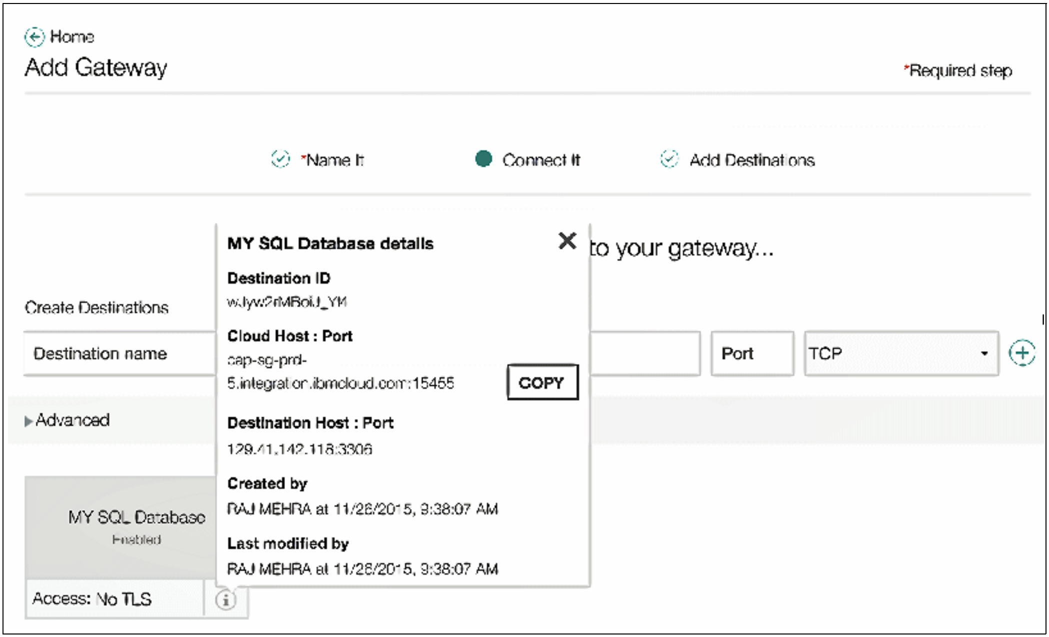
> **Important:** In this example, we are simulating the corporate data center using a public infrastructure. For this reason, the IP address used on the window is a public IP address. However, if you are using a server in your network, use the address of the server inside your network.



MY SQL Database	129.41.142.118	3306	TCP

Figure 6-24 Creating a destination for the gateway

3. The window shows the new destination at the bottom of panel. Click **i**.
4. Click **COPY** as shown in Figure 6-25. Save the details as <*Cloud Host for MySQL Server*> and <*Cloud Port for MySQL Server*>.



MY SQL Database details

Destination ID
wJyw2rMBoiJ_YM

Cloud Host : Port
cap-sg-prd-5.integration.ibmcloud.com:15455

Destination Host : Port
129.41.142.118:3306

Created by
RAJ MEHRA at 11/28/2015, 9:38:07 AM

Last modified by
RAJ MEHRA at 11/28/2015, 9:38:07 AM

Figure 6-25 Secure Gateway destination details

Note: <*Cloud Host for MySQL Server*> and <*Cloud Port for MySQL Server*> values derived in the previous step are used for configuration in later sections. It is suggested that these values be saved somewhere for later use.

Configurations for MySQL server

In this section, we follow instructions at the link below to install and configure the secure client gateway on your MySQL database server, the one running on <*virtual address of the MYSQL Server*>:

https://www.ng.bluemix.net/docs/services/SecureGateway/sg_021.html#sg_025

1. Connect to the MySQL server and open a command prompt. Type the following command:

   ```
   mkdir /home/ibmcloud/securegateway
   ```

2. From your workstation, copy ***<IBM Secure gateway client installer>*** to folder `/home/ibmcloud/securegateway`, for example, on a Linux workstation:

   `scp -i mqlightkey` ***<IBM Secure gateway client installer>*** `ibmcloud@<`*`virtual address of the MySQL Server`*`>:/home/ibmcloud/securegateway`

3. Connect to the MySQL server and open a command prompt. Type the following command:

   ```
   cd /home/ibmcloud/securegateway
   sudo dpkg -i ibm-securegateway-client-1.3.2+client_amd64.deb
   ```

   a. When prompted for the gateway ID, press Enter.

   b. When prompted, enter the configuration's ID security token (if any). Press Enter.

   c. When prompted `Do you wish to change the startup logging level, values are INFO, DEBUG, ERROR or TRACE - [INFO]:`, enter the configuration's ID security token (if any). Press Enter.

   d. When prompted `Supply an ACL File for processing [optional]:`, press Enter.
   e. The following message is displayed at the end of this process: `[postinst] Completed with SUCCESS`.
4. Connect to MySQL server and open the following command prompt:

```
cd /opt/ibm/securegateway
```

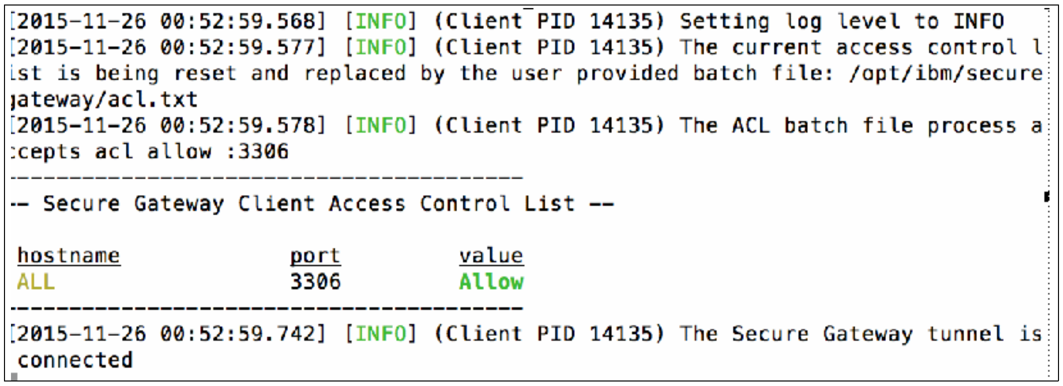
5. Create an `acl.txt` file by using your preferred editor.
6. Enter the following line:

```
acl allow :3306
```

7. Connect to MySQL server and open a command prompt. Type the following commands:

```
sudo su secgwadmin
cd /opt/ibm/securegateway
node lib/secgwclient.js <Secure GATEWAY ID for Database Server> --F
/opt/ibm/securegateway/acl.txt
```

   The gateway starts and should show details as shown in Figure 6-26.



```
[2015-11-26 00:52:59.568] [INFO] (Client PID 14135) Setting log level to INFO
[2015-11-26 00:52:59.577] [INFO] (Client PID 14135) The current access control l
ist is being reset and replaced by the user provided batch file: /opt/ibm/secure
jateway/acl.txt
[2015-11-26 00:52:59.578] [INFO] (Client PID 14135) The ACL batch file process a
:cepts acl allow :3306
---------------------------------------
-- Secure Gateway Client Access Control List --

 hostname                  port         value
 ALL                       3306         Allow
---------------------------------------
[2015-11-26 00:52:59.742] [INFO] (Client PID 14135) The Secure Gateway tunnel is
 connected
```

Figure 6-26 IBM Secure Gateway client started on MySQL database

   The API server application running on Node.js in Bluemix can now connect to the database using the secure tunnel.

6.8.5 Configure the API server to use a secure connection to the database

This section describes the steps that are required to configure a secure connection between the API server running on Bluemix and the database using the secure tunnel:

1. Open the following URL:

   https://console.ng.bluemix.net

2. Log in using your IBM user ID and password.

3. Click **DASHBOARD**.
4. Click **redbook-MQLight-service-1** under Applications as shown in Figure 6-27.

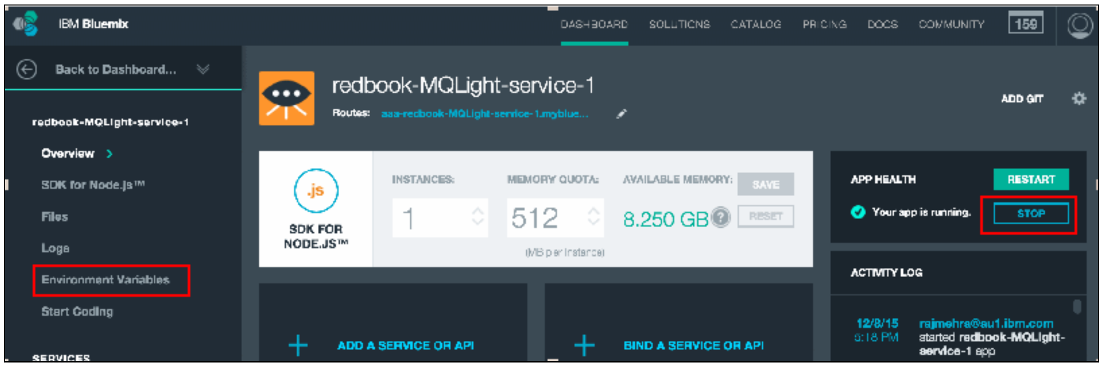
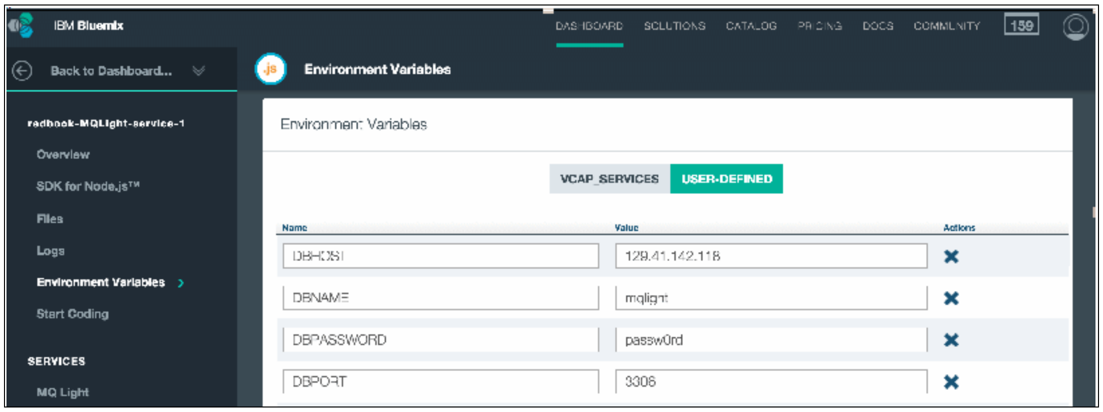


Figure 6-27   API server

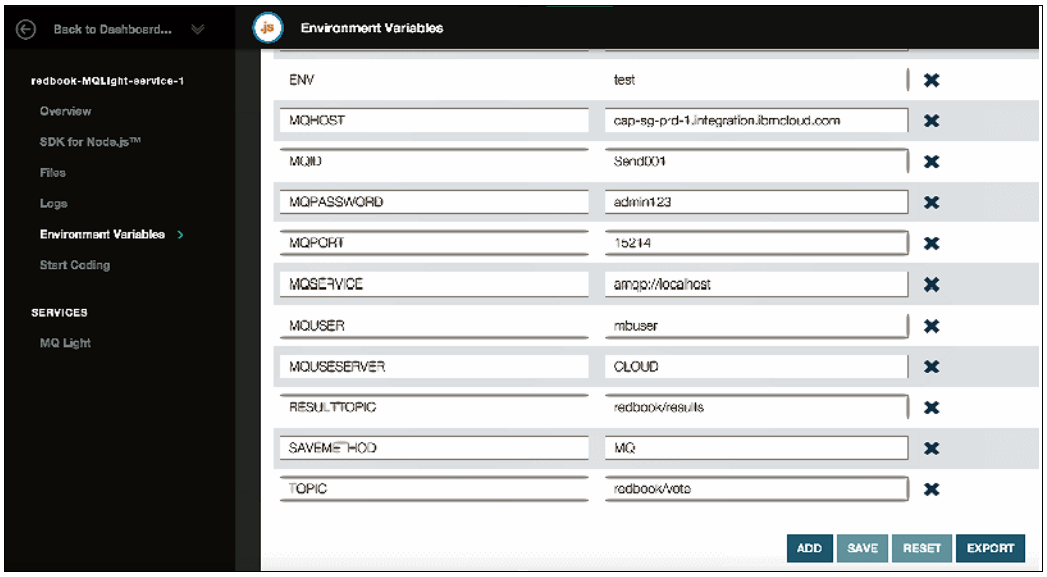
5. Click **STOP**.
6. Click **Environment Variables**. See Figure 6-28.



Name	Value	Actions
DBHOST	129.41.142.118	✖
DBNAME	mqlight	✖
DBPASSWORD	passw0rd	✖
DBPORT	3306	✖

Figure 6-28   Environment Variables window

7. Click **USER-DEFINED**. See Figure 6-29 on page 137.



ENV	test
MQHOST	cap-sg-prd-1.integration.ibmcloud.com
MQID	Send001
MQPASSWORD	admin123
MQPORT	15214
MQSERVICE	amqp://localhost
MQUSER	mbuser
MQUSESERVER	CLOUD
RESULTTOPIC	redbook/results
SAVEMETHOD	MQ
TOPIC	redbook/vote

Figure 6-29 Setting the API application user parameters

8. Replace with the value that matches your configurations, as shown in Table 6-5.

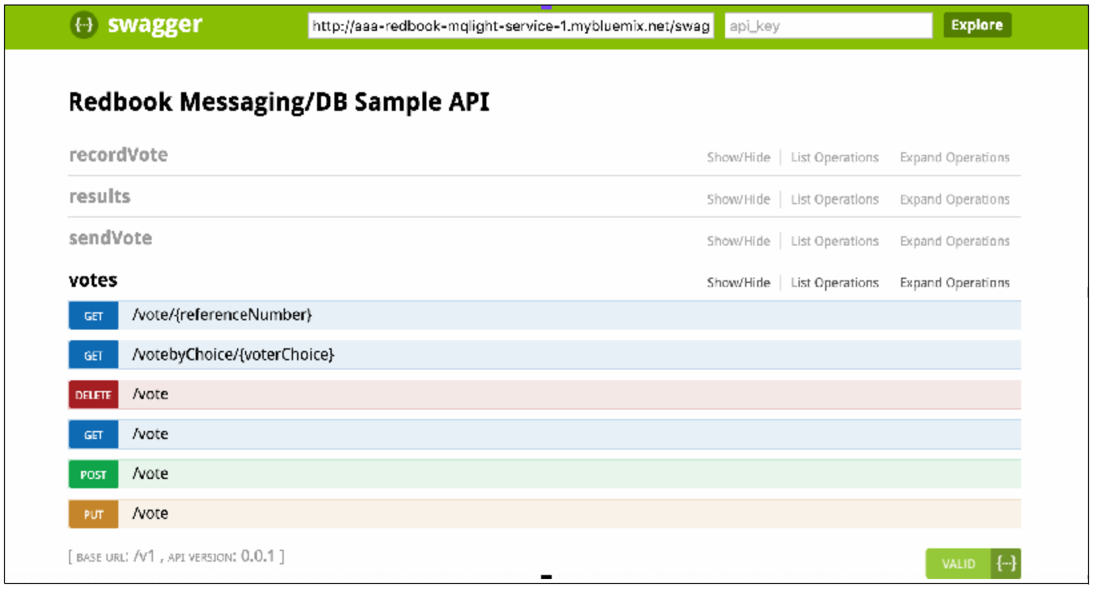
Table 6-5 Parameters for the API server

Attribute	Default/Old	New value	Purpose
DBHOST	*<virtual address of the MYSQL Server>*	*<Cloud Host for MY SQL Server>*	Host name or IP address of MySQL server
DBPORT	3306	*<Cloud Port for MY SQL Server>*	Port number for MySQL database
DBNAME	mqlight	mqlight	MySQL database/schema name
DBUSER	admin	admin	MySQL user
DBPASSWORD	passw0rd	passw0rd	MySQL password
TOPIC	redbook/vote	redbook/vote	Topic from IBM MQ Light/IBM MQ server
MQHOST	<>	<>	IBM MQ Server host name or IP server
MQPORT	5672	5672	Port number for IBM MQ Server AMQP Listener
MQUSER	<>	<>	User ID to connect to IBM MQ server
MQPASSWORD	<>	<>	Password to connect to IBM MQ server
MQSERVICE	ampq://localhost	ampq://localhost	Local MQ Light Server

Attribute	Default/Old	New value	Purpose
MQUSESERVER	LOCAL	BLUEMIX	Valid values: LOCAL: Local MQ Light server BLUEMIX: Bluemix service CLOUD: IBM MQ server
MQID	Send001	Send001	Sender ID
SAVEMETHOD	MQ	DB	Valid values: DB: Direct to database MQ: To IBM MQ Light/IBM MQ Server
RESULTTOPIC	redbook/results	redbook/results	Only used in subscenario 4 getting the results
ENABLEENTERPRISE	false	false	The valid values are “true” (only for subscenario 4) or “false” for all other subscenarios.

9. Scroll to the bottom of page and click **SAVE**.
10. Click **Overview**.
11. Click **START**.
12. The API application running on Bluemix is now connecting to the database using Secure Gateway tunnel.
13. Click **redbook-MQLight-service-1**, as shown in Figure 6-15 on page 128.
14. Click **Overview**.
15. Write down the URL next to **Routes** as *<API Server Host Name>*. It will be something like aaa-redbook-mqlight-service-1.mybluemix.net/.
16. Open a new browser window or tab and type in the URL: http://*<API Server Host Name>/api-docs/#*

    The API Server Test Console is shown as in Figure 6-30 on page 139.



Redbook Messaging/DB Sample API

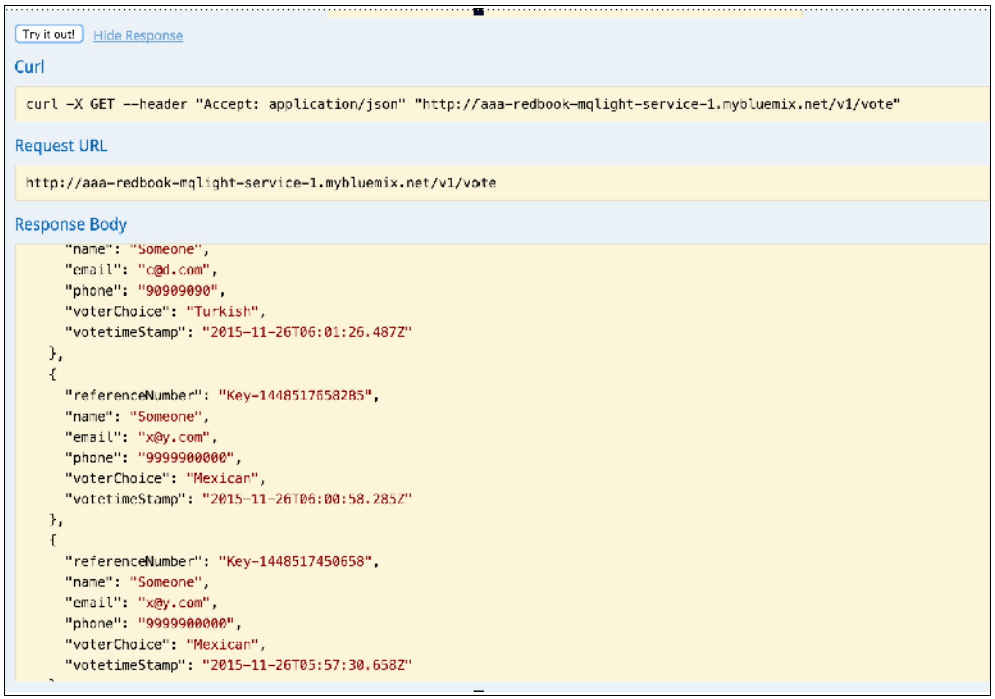
recordVote	Show/Hide | List Operations Expand Operations
results	Show/Hide | List Operations Expand Operations
sendVote	Show/Hide | List Operations Expand Operations
votes	Show/Hide | List Operations Expand Operations
GET	/vote/{referenceNumber}
GET	/votebyChoice/{voterChoice}
DELETE	/vote
GET	/vote
POST	/vote
PUT	/vote

[BASE URL: /v1 , API VERSION: 0.0.1]

Figure 6-30 API Server Test Console

17. Click **votes**. This opens a list of APIs for votes.
18. Click **GET /votes**. This opens the API details.
19. Click **Try it out!**

 The database records from the MySQL database are shown as a JSON array, as shown in Figure 6-31.



Curl

```
curl -X GET --header "Accept: application/json" "http://aaa-redbook-mqlight-service-1.mybluemix.net/v1/vote"
```

Request URL

```
http://aaa-redbook-mqlight-service-1.mybluemix.net/v1/vote
```

Response Body

```
    "name": "Someone",
    "email": "c@d.com",
    "phone": "90909090",
    "voterChoice": "Turkish",
    "votetimeStamp": "2015-11-26T06:01:26.487Z"
  },
  {
    "referenceNumber": "Key-1448517658285",
    "name": "Someone",
    "email": "x@y.com",
    "phone": "9999900000",
    "voterChoice": "Mexican",
    "votetimeStamp": "2015-11-26T06:00:58.285Z"
  },
  {
    "referenceNumber": "Key-1448517450658",
    "name": "Someone",
    "email": "x@y.com",
    "phone": "9999900000",
    "voterChoice": "Mexican",
    "votetimeStamp": "2015-11-26T05:57:30.658Z"
```

Figure 6-31 Details from the API server

The API server is connected to the MySQL server using the Secure Gateway.

6.8.6 Installing the web application

The web application allows a user to cast their vote. This application interacts with the API server to record a user's vote. Follow these steps to install the web application:

1. Use the code download in 6.8.3, "Deploy the API server" on page 125.
2. Use the code in the `sample-app` folder.
3. Use a text editor of your choice to edit `sample-app/redbookApp/connection.properties`. Table 6-6 contains the parameters as JSON. Replace with the value that matches your configurations.

Table 6-6 Parameters for the Sample-App

Attribute	Default	New value	Purpose
baseUrl	<>	*https://<API Server Host Name>/v1*	Base URL for the Rest API
serviceUrl	<>	<>	Used in subscenario 4 to store API Gateway URL for votes by cuisine.

4. Validate the contents of the file and save it.
5. Change the host name to a unique name in `sample-app/manifest.yml`. Replace aaa with three unique characters of your choice.

Table 6-7 Parameters table

Attribute	Default	New value	Purpose
host	aaa-redbook-voting-sample-app	xxx-redbook-voting-sample-app	Unique host name for Web application server

6. Open a command prompt on your workstation.
7. Follow the instructions provided at the link below and log in to the Bluemix account by using the **`cf login`** command:

   `https://www.ng.bluemix.net/docs/cli/reference/cfcommands/index.html`
8. On the command prompt, change the current directory to the `voting-service` folder.
9. Enter the **`cf push`** command in the command window. This uploads the application to Bluemix and deploys it.
10. If another application is using the same host name that you have chosen, the deployment might fail. In that instance, change the host as in step 6 and try step 11.
11. Open the following URL:

    `https://console.ng.bluemix.net`
12. Log in using your IBM user ID and password.
13. Click **DASHBOARD**.

    An application is displayed on the dashboard.

14. The web application shows up as shown in Figure 6-32.

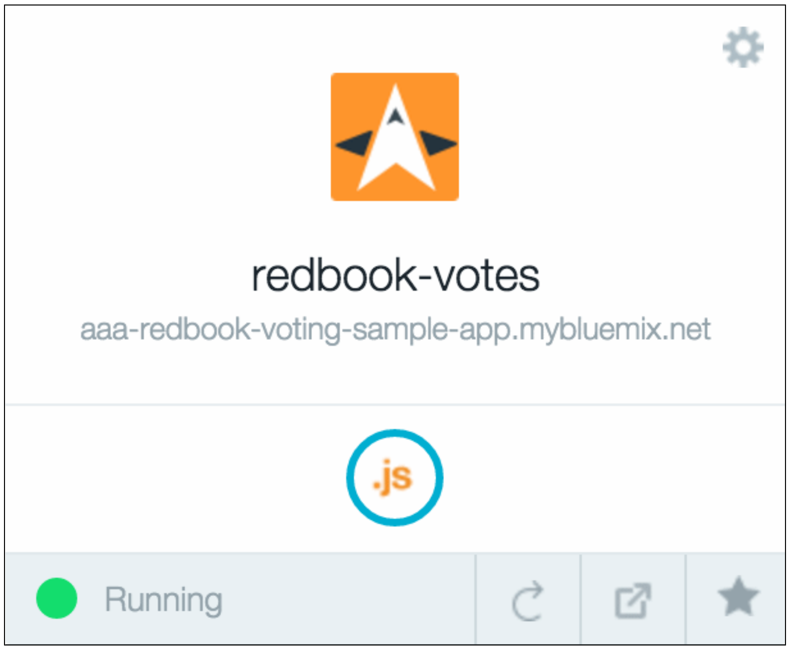
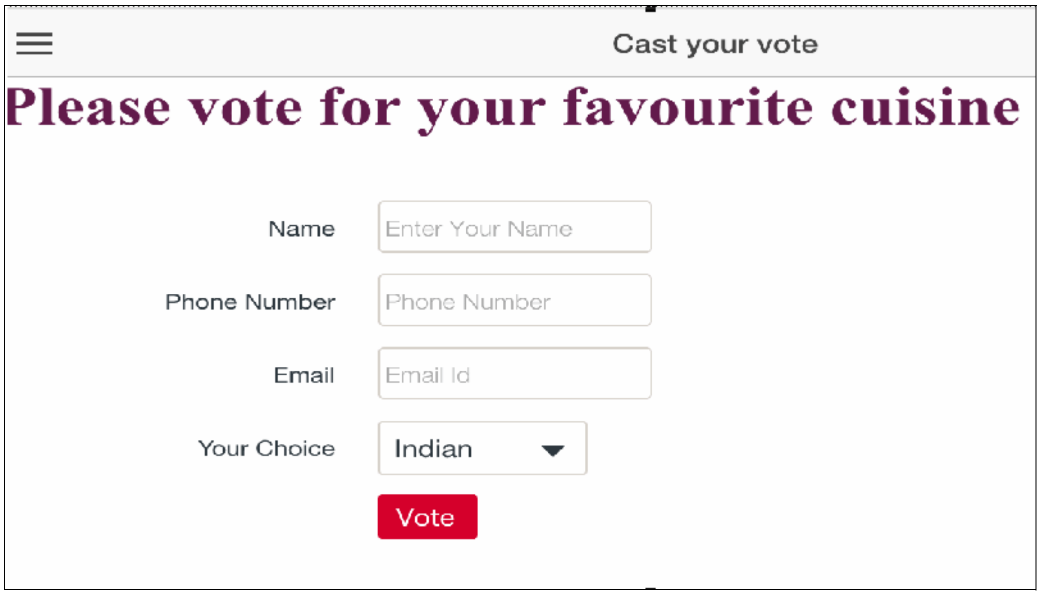


Figure 6-32 API server application deployed and running on Bluemix

15. Click **redbook-votes**.
16. Click **Overview**.
17. Write down the URL next to **Routes** as <*Web application Host Name*>. It will be something like http://aaa-redbook-voting-sample-app.mybluemix.net/.
18. Open a new browser window or tab and type in the URL: <*Web application Host Name*>*/redbookApp.*
19. Enter your name, phone number, email, and select a choice from the drop-down list, as shown in Figure 6-33.



Cast your vote

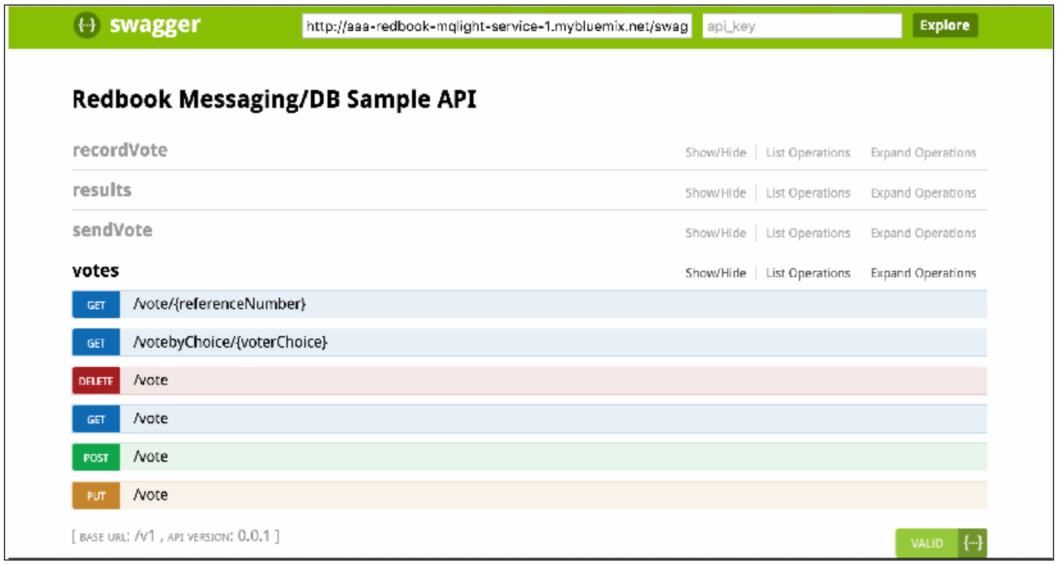
Please vote for your favourite cuisine

Name	Enter Your Name
Phone Number	Phone Number
Email	Email Id
Your Choice	Indian

Figure 6-33 Voting application: Cast your vote

20. Click **Vote**.
21. The application responds with a message of thanks and displays a time stamp for the vote.
22. Open a new browser window or tab and type in the URL: http://*<API Server Host Name>/api-docs/#*

    The API Server Test Console is displayed as in Figure 6-34.



Redbook Messaging/DB Sample API

recordVote — Show/Hide | List Operations Expand Operations
results — Show/Hide | List Operations Expand Operations
sendVote — Show/Hide | List Operations Expand Operations
votes — Show/Hide | List Operations Expand Operations

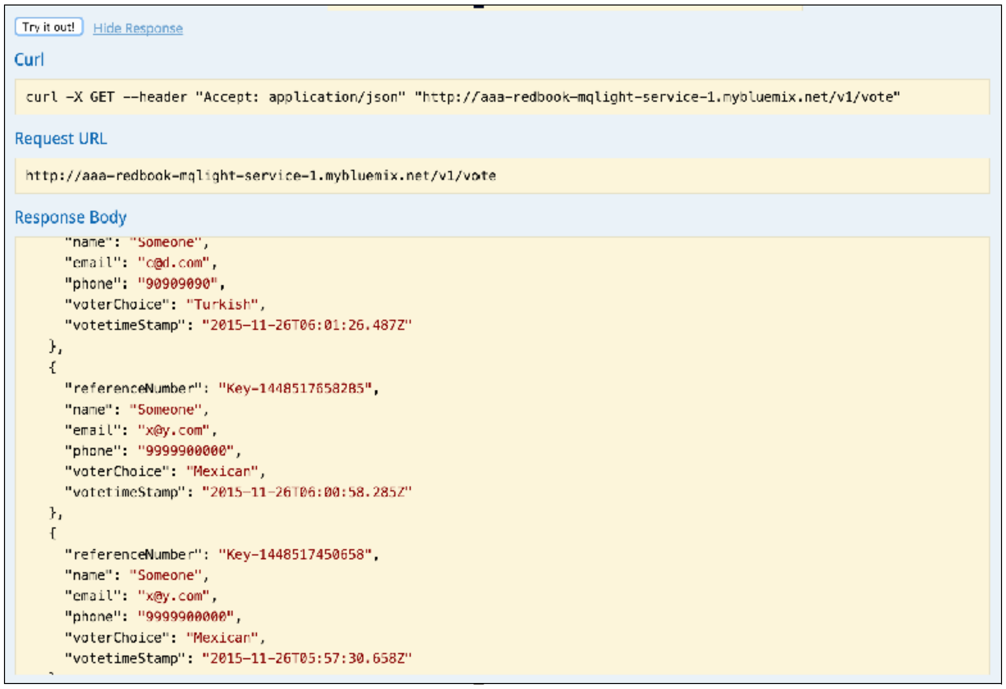
GET /vote/{referenceNumber}
GET /votebyChoice/{voterChoice}
DELETE /vote
GET /vote
POST /vote
PUT /vote

[BASE URL: /v1 , API VERSION: 0.0.1]

Figure 6-34 API Server Test Console

23. Click **votes**. This opens a list of APIs for votes.
24. Click **GET /votes**. This opens the API details.
25. Click **Try it out!**

    The database records from the MySQL database are shown as a JSON array as shown in Figure 6-35 on page 143.



```
curl -X GET --header "Accept: application/json" "http://aaa-redbook-mqlight-service-1.mybluemix.net/v1/vote"
```

```
http://aaa-redbook-mqlight-service-1.mybluemix.net/v1/vote
```

```
    "name": "Someone",
    "email": "c@d.com",
    "phone": "90909090",
    "voterChoice": "Turkish",
    "votetimeStamp": "2015-11-26T06:01:26.487Z"
  },
  {
    "referenceNumber": "Key-1448517658285",
    "name": "Someone",
    "email": "x@y.com",
    "phone": "9999900000",
    "voterChoice": "Mexican",
    "votetimeStamp": "2015-11-26T06:00:58.285Z"
  },
  {
    "referenceNumber": "Key-1448517450658",
    "name": "Someone",
    "email": "x@y.com",
    "phone": "9999900000",
    "voterChoice": "Mexican",
    "votetimeStamp": "2015-11-26T05:57:30.658Z"
```

Figure 6-35 Details from API server

Notice that the votes you cast should now appear in this list.

6.8.7 Summary of scenario 1

In this scenario, we implemented a web application that is hosted on a Node.js server on Bluemix. The web application uses REST API to persist the data. The REST API server, which is a Node.js application, connects to an on-premises database using a secure tunnel. Data is recorded into the database synchronously. This subscenario offers limited scalability for the solution.

Figure 6-36 shows the flow of information.

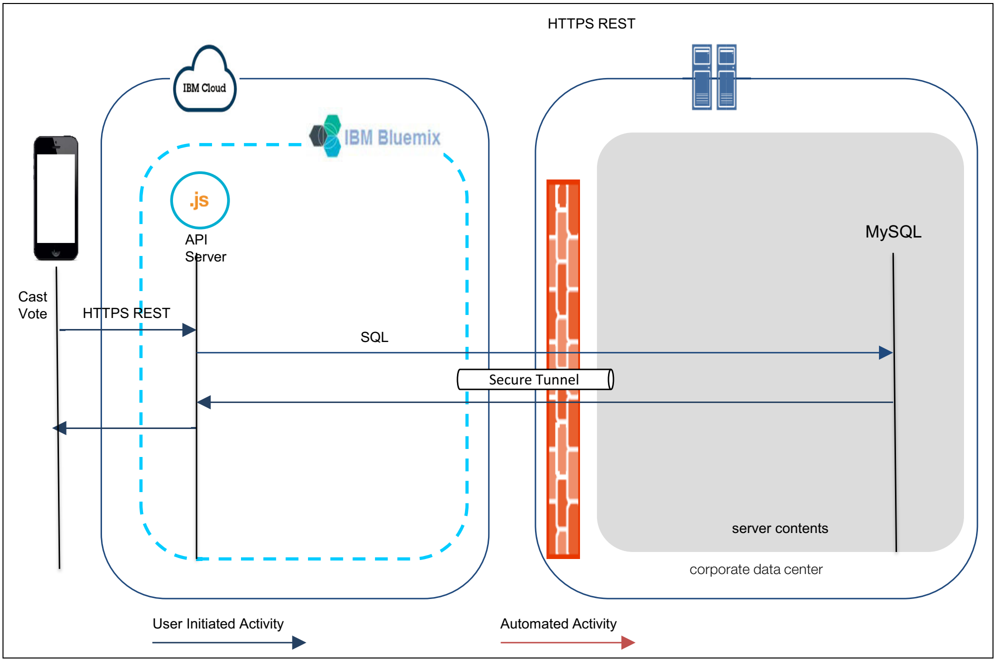


Figure 6-36 Subscenario 1: Flow of information from user to database

6.9 Implementing subscenario 2

This subscenario implements the worker offload pattern. It reuses the components used in the previous subscenario and adds a new component. The API server records a vote into an IBM MQ Light server. So the request can be processed faster, this scenario adds a message processor application to the solution. The message processor application is designed to read messages from the IBM MQ Light server and add the data to the MySQL database. This scenario allows for improved scalability for recording the votes. The slowest part of persisting data to the database has been offloaded to another application allowing the API server to process more user requests during a given period.

The components used for this subscenario are shown in Figure 6-37.

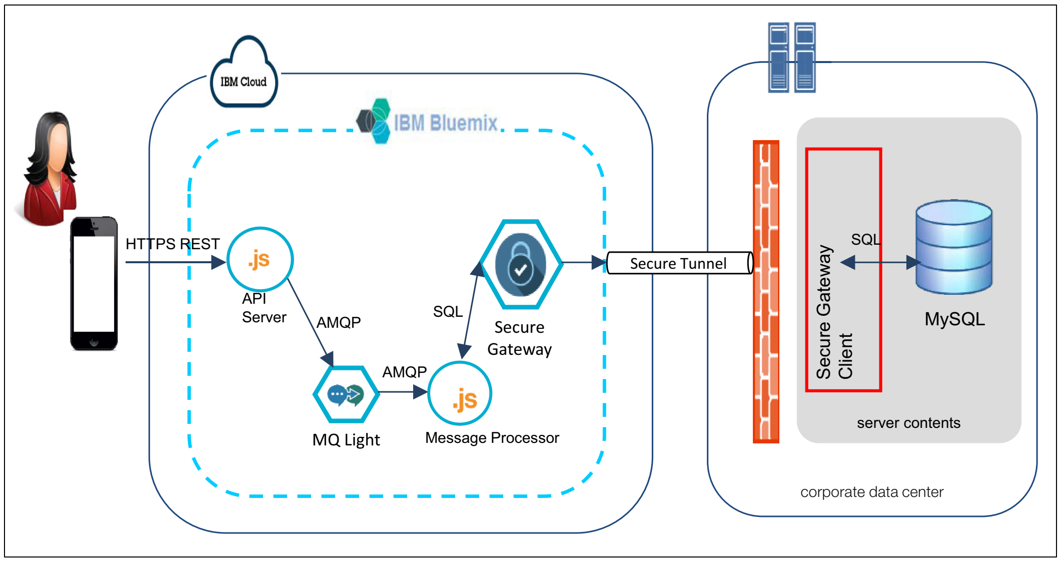


Figure 6-37 Components used for subscenario 2

6.9.1 Provision IBM MQ Light Bluemix services

Perform the following steps to provision Bluemix services:

1. Open the following URL:

   `https://console.ng.bluemix.net`

2. If you do not have an IBM Bluemix ID, click **Signup** to create your account. Follow instructions provided on the page to create your new account.
3. Log in to your Bluemix account by using your IBM user ID and password.
4. Click **Catalog**.
5. Scroll down the page and click **MQ Light**.

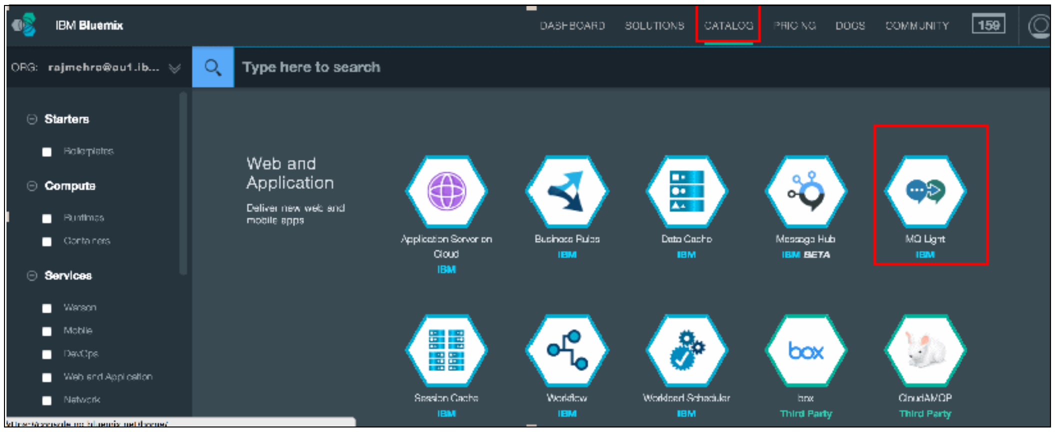
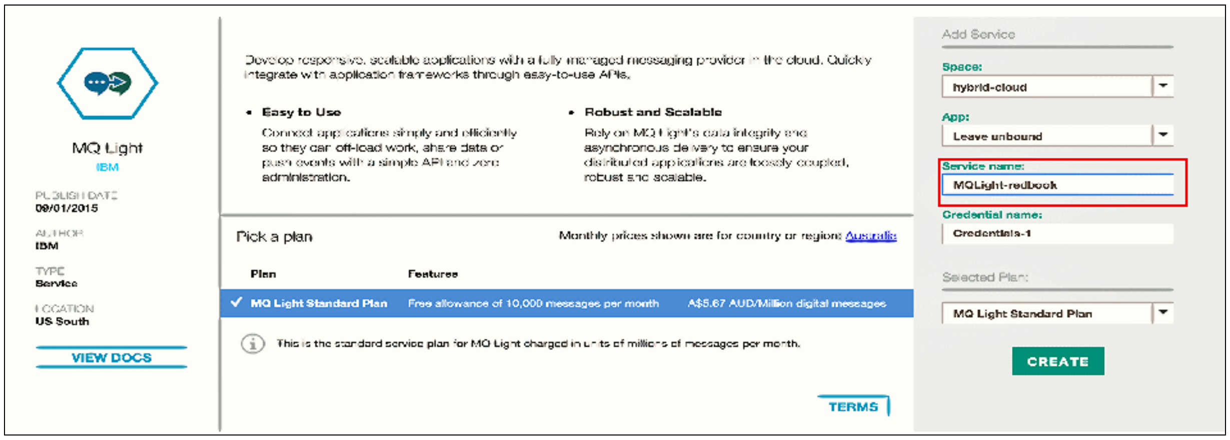


Figure 6-38 Provisioning IBM MQ Light Bluemix service

6. Enter “Service name” as MQLight-redbook. Click **Create**. See Figure 6-39.


MQ Light
IBM
PUBLISH DATE
09/01/2015
AUTHOR
IBM
TYPE
Service
LOCATION
US South

Develop responsive, scalable applications with a fully managed messaging provider in the cloud. Quickly integrate with application frameworks through easy-to-use APIs.

- **Easy to Use**
  Connect applications simply and efficiently so they can off-load work, share data or push events with a simple API and zero administration.
- **Robust and Scalable**
  Rely on MQ Light's data integrity and asynchronous delivery to ensure your distributed applications are loosely coupled, robust and scalable.

Pick a plan — Monthly prices shown are for country or region: Australia

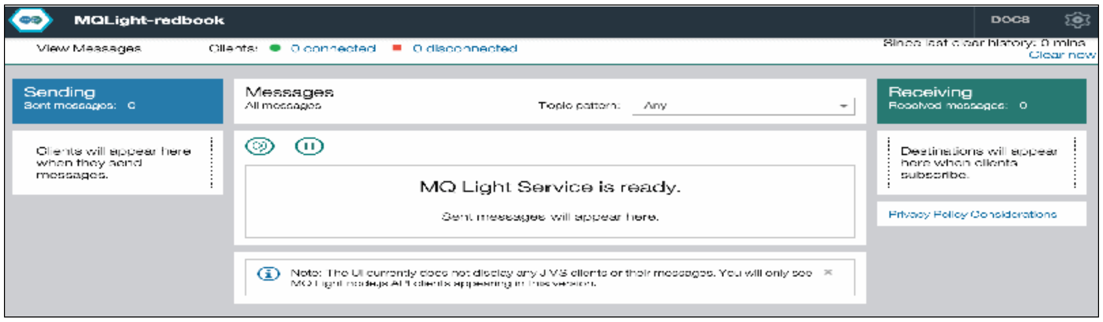
Plan	Features	
✓ MQ Light Standard Plan	Free allowance of 10,000 messages per month	A$5.67 AUD/Million digital messages

This is the standard service plan for MQ Light charged in units of millions of messages per month.

Add Service
Space: hybrid-cloud
App: Leave unbound
Service name: MQLight-redbook
Credential name: Credentials-1
Selected Plan: MQ Light Standard Plan

Figure 6-39 Provisioning IBM MQ Light service

The IBM MQ Light console appears as in Figure 6-40.


MQLight-redbook
View Messages   Clients: ● 0 connected ■ 0 disconnected   Since last clear history: 0 mins Clear now

Sending
Sent messages: 0
Clients will appear here when they send messages.

Messages
All messages   Topic pattern: Any
MQ Light Service is ready.
Sent messages will appear here.

Note: The UI currently does not display any JMS clients or their messages. You will only see MQ Light node.js API clients appearing in this version.

Receiving
Received messages: 0
Destinations will appear here when clients subscribe.

Privacy Policy Considerations

Figure 6-40 IBM MQ Light service console

6.9.2 Installing the message processor application

The message processing application listens for IBM MQ Light messages and saves them to MySQL database. Perform the following steps to install the message processor application:

1. Use the code that was downloaded in 6.8.3, “Deploy the API server” on page 125.
2. Go to the `message-processing` folder.
3. Use a text editor of your choice to edit `message-processing/manifest.yml`.
4. Replace with the value that matches your configurations, as shown in Table 6-8.

Table 6-8 Parameters for the message processor on Bluemix

Attribute	Default	New value	Purpose
DBHOST	<>	<Cloud Host for MySQL Server>	Host name or IP address of MySQL server
DBPORT	3306	<Cloud Port for MySQL Server>	Port number for MySQL database
DBNAME	mqlight	mqlight	MySQL database/schema name

Attribute	Default	New value	Purpose
DBUSER	admin	admin	MySQL user
DBPASSWORD	passw0rd	passw0rd	MySQL password
TOPIC	redbook/vote	redbook/vote	Topic from IBM MQ Light/IBM MQ server
MQHOST	<>	<>	IBM MQ server host name or IP server
MQPORT	5672	5672	Port number for IBM MQ server AMQP Listener
MQUSER	<>	<>	User ID to connect to IBM MQ server
MQPASSWORD	<>	<>	Password to connect to IBM MQ server
MQSERVICE	ampq://localhost	ampq://localhost	Local MQ Light server
MQUSESERVER	LOCAL	BLUEMIX	Valid values: LOCAL: Local IBM MQ Light server BLUEMIX: Bluemix service CLOUD: IBM MQ server
MQID	Recv001	Recv001	Receiver ID
SAVEMETHOD	MQ	MQ	Valid values: DB: Direct to database MQ: To IBM MQ Light/IBM MQ server

5. Change the host name to a unique name. Replace aaa with three unique characters of your choice.

Table 6-9 Parameters table

Attribute	Default	New value	Purpose
host	aaa-redbook-MQLight-service-1	**xxx**-redbook-MQLight-service-1	Unique host name for API server

6. Validate the contents of the file and save it.
7. Note the host of your message processor as *<Message Processor Host Name Bluemix>.*
8. Open a command prompt on your workstation.
9. Follow the instructions provided at the following link and log in to your Bluemix account by using the **`cf login`** command:

   `https://www.ng.bluemix.net/docs/cli/reference/cfcommands/index.html`
10. On the command prompt, change the current directory to the `message-processing` folder.
11. Enter the **`cf push`** command in the command window. This uploads the application to Bluemix and deploys it.

12. If another application is using the same host name that you have chosen, the deployment might fail. In that instance, change the host as in step 6 and try step 11.

13. Open the following URL:

    `https://console.ng.bluemix.net`

14. Log in using your IBM user ID and password.

15. Click **DASHBOARD**.

    An application is displayed on the dashboard. See Figure 6-41.

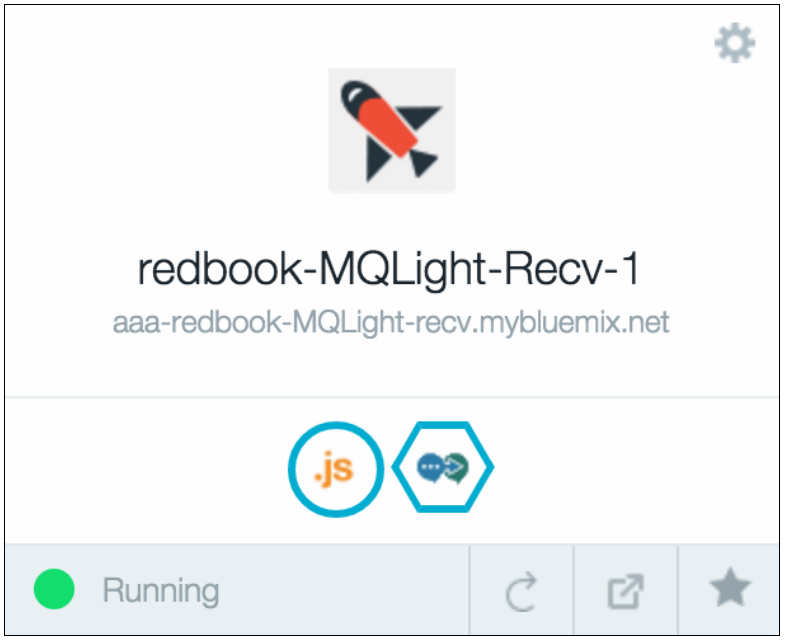


Figure 6-41   API server application deployed and running on Bluemix

The message receiver application displays as shown in Figure 6-41.

6.9.3 Configuring the API server to use IBM MQ Light service in Bluemix

This step involves changing the API application to use the IBM MQ service on Bluemix instead of persisting to the database.

This section describes the steps that are required to configure a secure connection between the API server running on Bluemix and the database using the secure tunnel:

1. Open the following URL:

   `https://console.ng.bluemix.net`

2. Log in using your IBM user ID and password.
3. Click **DASHBOARD**.
4. Click **redbook-MQLight-service-1** under Applications.

5. Click **Overview**. See Figure 6-42.

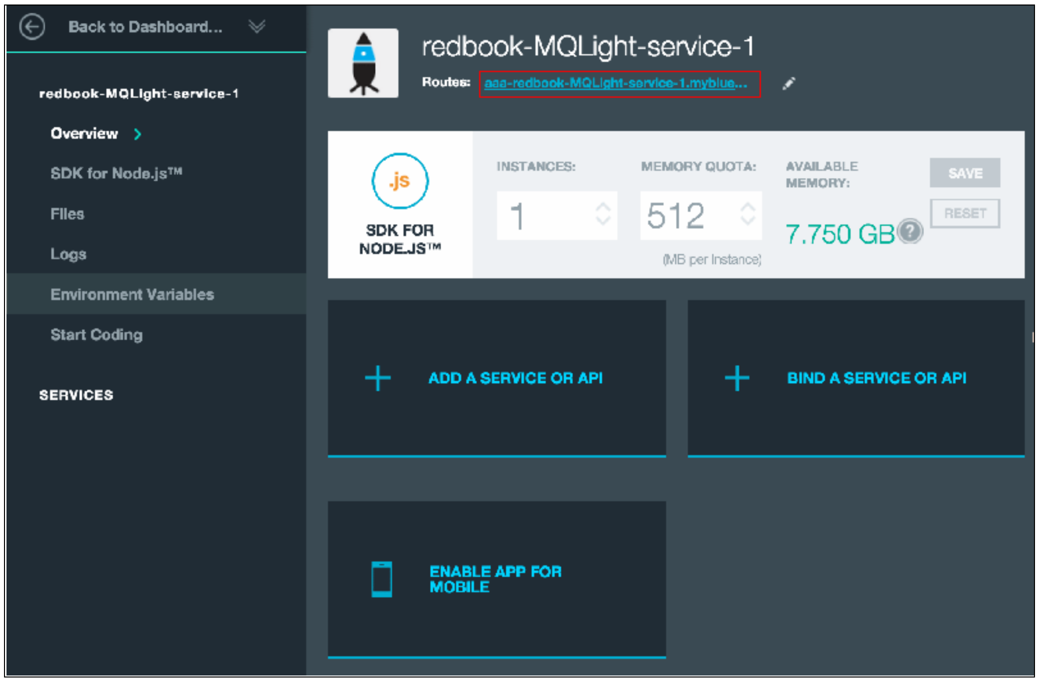


Figure 6-42 API server

6. Click **STOP**.
7. Click **BIND A SERVICE OR API**.
8. Select **MQLight-redbook** as shown in Figure 6-43.

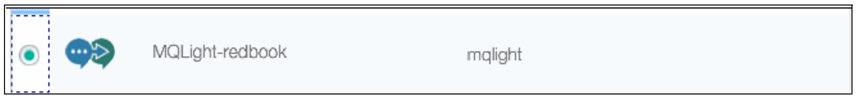
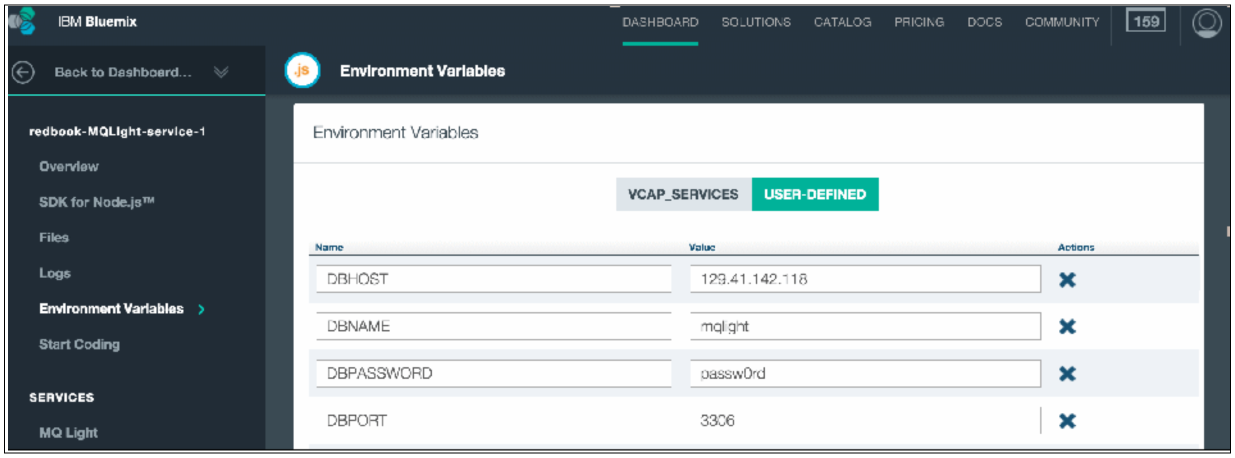


Figure 6-43 Add IBM MQ Light service

9. Click **ADD**.
10. Click **RESTAGE**.
11. Click **Environment Variables**.
12. Click **USER-DEFINED**.

13.Click **Environment Variables**. See Figure 6-44.



Name	Value	Actions
DBHOST	129.41.142.118	✖
DBNAME	mqlight	✖
DBPASSWORD	passw0rd	✖
DBPORT	3306	✖

Figure 6-44 Environment Variables window

14.Click **USER-DEFINED**.

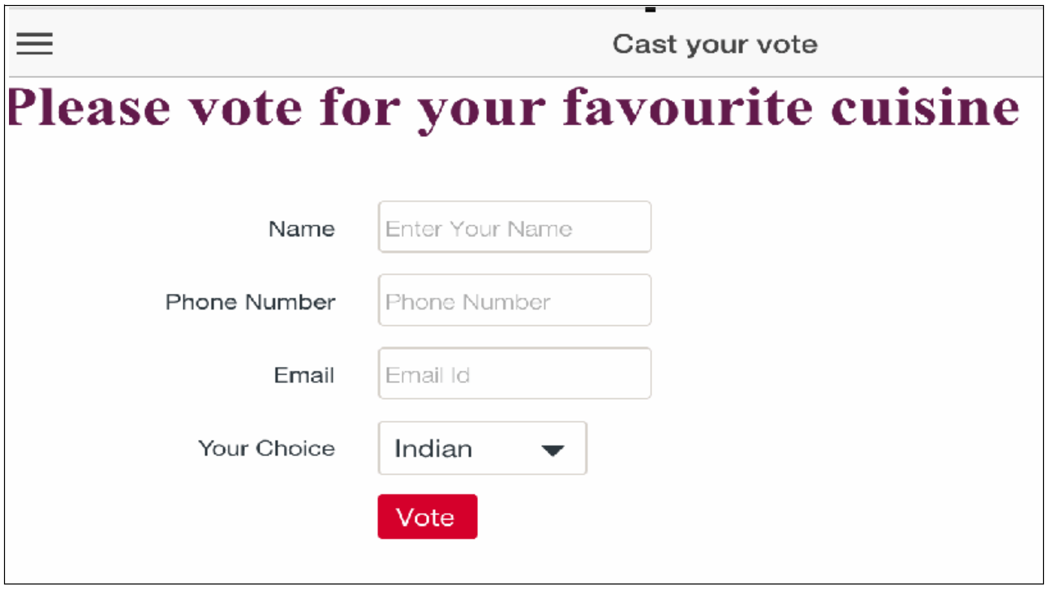
15.Replace with the value that matches your configurations, as shown in Table 6-10.

Table 6-10 Parameters for the API server

Attribute	Default/Old	New Value	Purpose
DBHOST	*<virtual address of the MySQL Server>*	*<Cloud Host for MySQL Server>*	Host name or IP address of MySQL server
DBPORT	3306	*<Cloud Port for MySQL Server>*	Port number for MySQL database
DBNAME	mqlight	mqlight	MySQL database/schema name
DBUSER	admin	admin	MySQL user
DBPASSWORD	passw0rd	passw0rd	MySQL password
TOPIC	redbook/vote	redbook/vote	Topic from IBM MQ Light/IBM MQ server
MQHOST	<>	<>	IBM MQ server host name or IP server
MQPORT	5672	5672	Port number for IBM MQ server AMQP Listener
MQUSER	<>	<>	User ID to connect to IBM MQ server
MQPASSWORD	<>	<>	Password to connect to IBM MQ server
MQSERVICE	ampq://localhost	ampq://localhost	Local MQ Light server

Attribute	Default/Old	New Value	Purpose
MQUSESERVER	LOCAL	BLUEMIX	Valid values: LOCAL: Local MQ Light server BLUEMIX: Bluemix service CLOUD: IBM MQ server
MQID	Send001	Send001	Sender ID
SAVEMETHOD	DB	MQ	Valid values: DB: Direct to database MQ: To IBM MQ Light/ IBM MQ server
RESULTTOPIC	redbook/results	redbook/results	Used in subscenario 4 to get leader board details
ENABLEENTERPRISE	false	false	The valid values are “true” (only for subscenario 4) or “false” for all other subscenarios

16. Scroll to bottom of the page and click **SAVE**.
17. Click **Overview**.
18. Click **START**.
19. The API application running on Bluemix is now recording a vote by using the IBM MQ Light server on Bluemix.
20. Open a new browser window or tab and type in the URL: <*Web application Host Name>/redbookApp.* See Figure 6-45.


Cast your vote

Please vote for your favourite cuisine

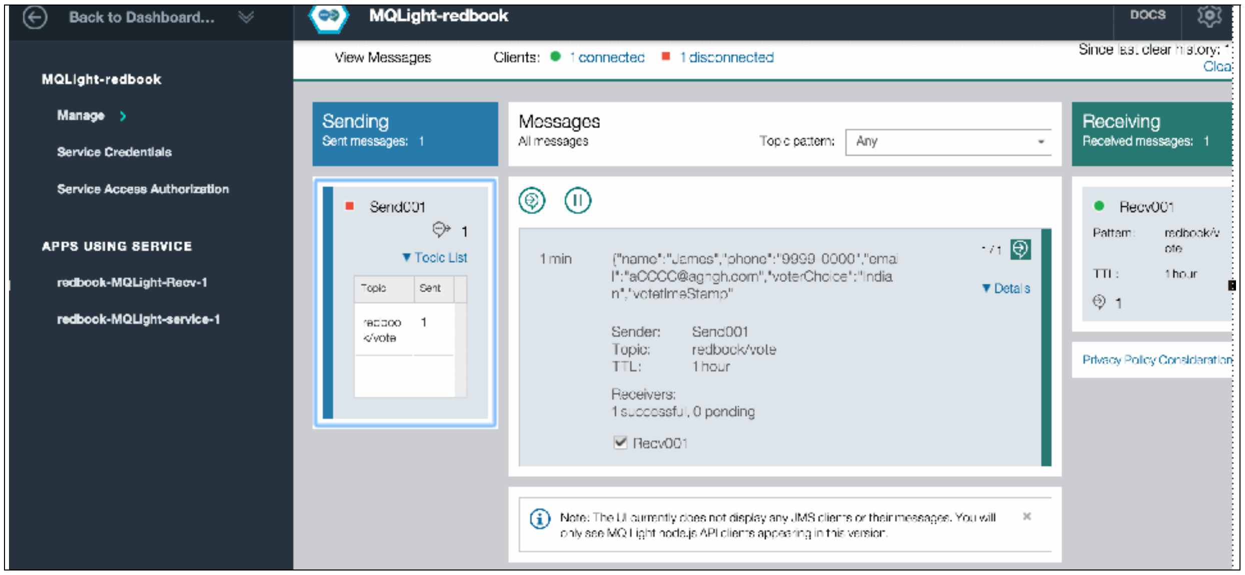
Name	Enter Your Name
Phone Number	Phone Number
Email	Email Id
Your Choice	Indian

Figure 6-45 Voting application: Cast your vote

21. Enter your name, phone number, email, and select a choice from the drop-down list.
22. Click **Vote**.
23. The application responds with a message of thanks and displays a time stamp for the vote.
24. The vote is now saved to the IBM MQ Light server by the web application. The message processing application processes the message from the IBM MQ Light server and posts it to the database using the secure tunnel.
25. Open the following URL:

    `https://console.ng.bluemix.net`

26. Log in using your IBM user ID and password.
27. Click **DASHBOARD**.
28. Under Services, click **MQLight-redbook**. See Figure 6-46.



```
1 min   {"name":"James","phone":"9999 0000","emai
        l":"aCCCC@aghgh.com","voterChoice":"India
        n","votetimeStamp"

        Sender:   Send001
        Topic:    redbook/vote
        TTL:      1 hour

        Receivers:
        1 successful, 0 pending

        ☑ Recv001
```

Note: The UI currently does not display any JMS clients or their messages. You will only see MQ Light node.js API clients appearing in this version.

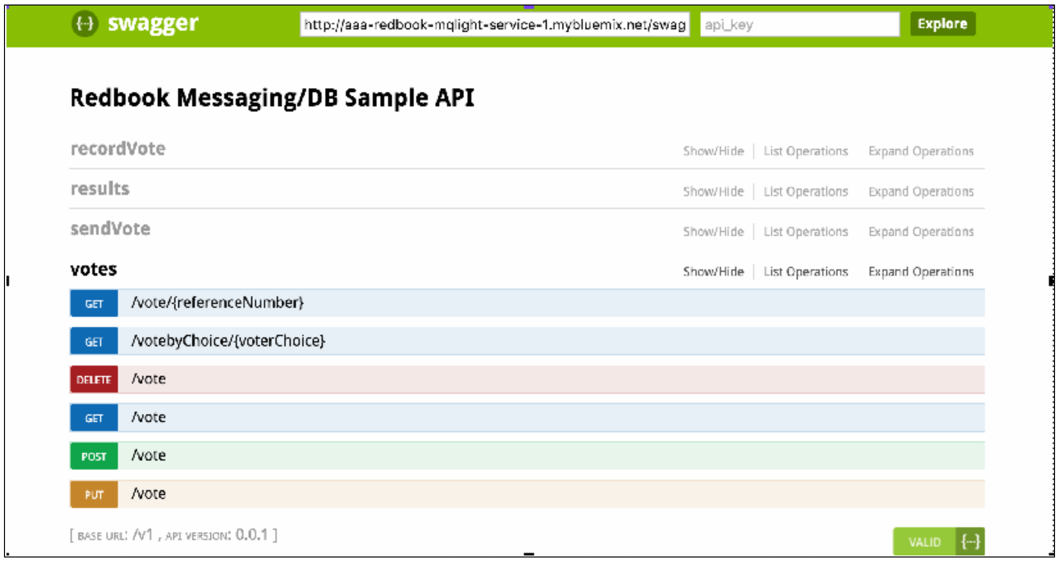
Figure 6-46 IBM MQ Light console

29. Figure 6-46 shows that the message was received from Send001 (API server). The content is the vote cast by the user. It also shows that the message was processed by Recv001, which is the message processor application.

Note: Sometimes there might be some delay in starting up the console for the IBM MQ Light service. You might not see the last message. If this occurs, cast another vote (go to step 15) and the result should show.

30. Open a new browser window or tab and type in the URL: http://*<API Server Host Name>/api-docs/#*

    The API Server Test Console is shown as in Figure 6-47.



Redbook Messaging/DB Sample API

recordVote	Show/Hide	List Operations	Expand Operations
results	Show/Hide	List Operations	Expand Operations
sendVote	Show/Hide	List Operations	Expand Operations
votes	Show/Hide	List Operations	Expand Operations

GET	/vote/{referenceNumber}
GET	/votebyChoice/{voterChoice}
DELETE	/vote
GET	/vote
POST	/vote
PUT	/vote

[BASE URL: /v1 , API VERSION: 0.0.1]

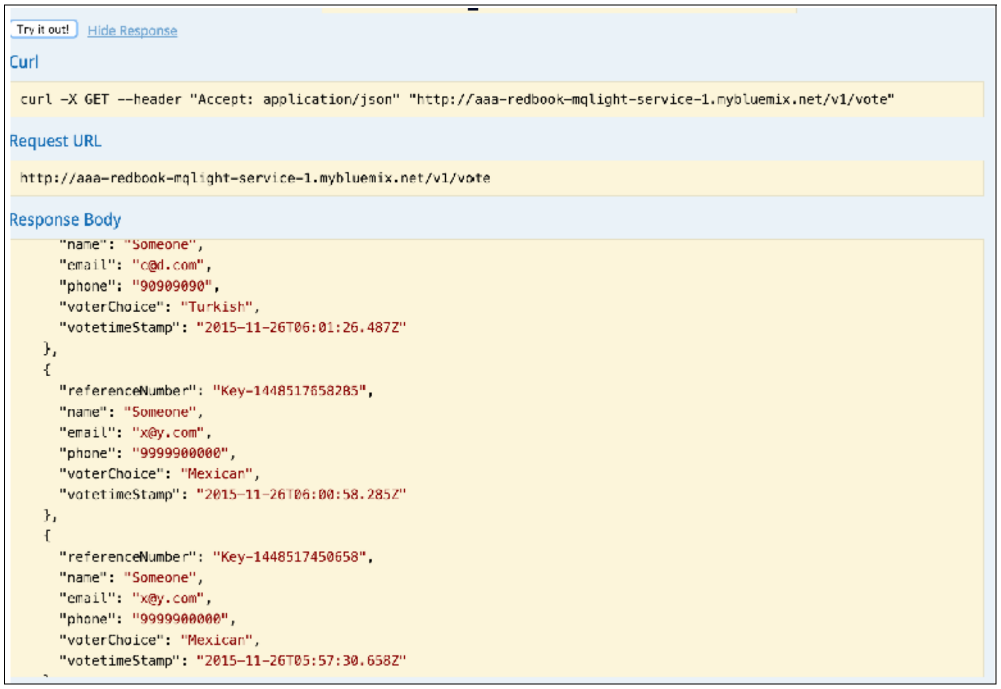
Figure 6-47   API Server Test Console

31. Click **votes**. This opens a list of APIs for votes.

32. Click **GET /votes**. This opens the API details.

33. Click **Try it out!**

    The database records from the MySQL database are shown as a JSON Array. See Figure 6-48.



Curl

```
curl -X GET --header "Accept: application/json" "http://aaa-redbook-mqlight-service-1.mybluemix.net/v1/vote"
```

Request URL

```
http://aaa-redbook-mqlight-service-1.mybluemix.net/v1/vote
```

Response Body

```
    "name": "Someone",
    "email": "c@d.com",
    "phone": "90909090",
    "voterChoice": "Turkish",
    "votetimeStamp": "2015-11-26T06:01:26.487Z"
  },
  {
    "referenceNumber": "Key-1448517658285",
    "name": "Someone",
    "email": "x@y.com",
    "phone": "9999900000",
    "voterChoice": "Mexican",
    "votetimeStamp": "2015-11-26T06:00:58.285Z"
  },
  {
    "referenceNumber": "Key-1448517450658",
    "name": "Someone",
    "email": "x@y.com",
    "phone": "9999900000",
    "voterChoice": "Mexican",
    "votetimeStamp": "2015-11-26T05:57:30.658Z"
```

Figure 6-48   Details from API server

34. Notice that the votes you cast should now appear in this list. To view the last message processed by the message processor, try the following URL on a browser: *<Message Processor Host Name Bluemix>/lastMessage*.

35. This URL displays a JSON object as shown below:

```
{"name":"John","phone":"9999-1010","email":"c@c.com","voterChoice":"French","vo
tetimeStamp":"2015-11-30T02:54:32.088Z","referenceNumber":"Key-1448852186372"}
```

6.9.4 Summary of subscenario 2

In this scenario, the data captured by the web application is temporarily stored in the IBM MQ Light server. It significantly reduces the time taken by the API server to process client requests. It improves performance and scalability of the solution. The important point to note is that no changes were required to the user interface application to achieve this result. The end result for the business is still the same. Each vote is recorded in a MySQL database.

Figure 6-49 shows the components for this scenario.

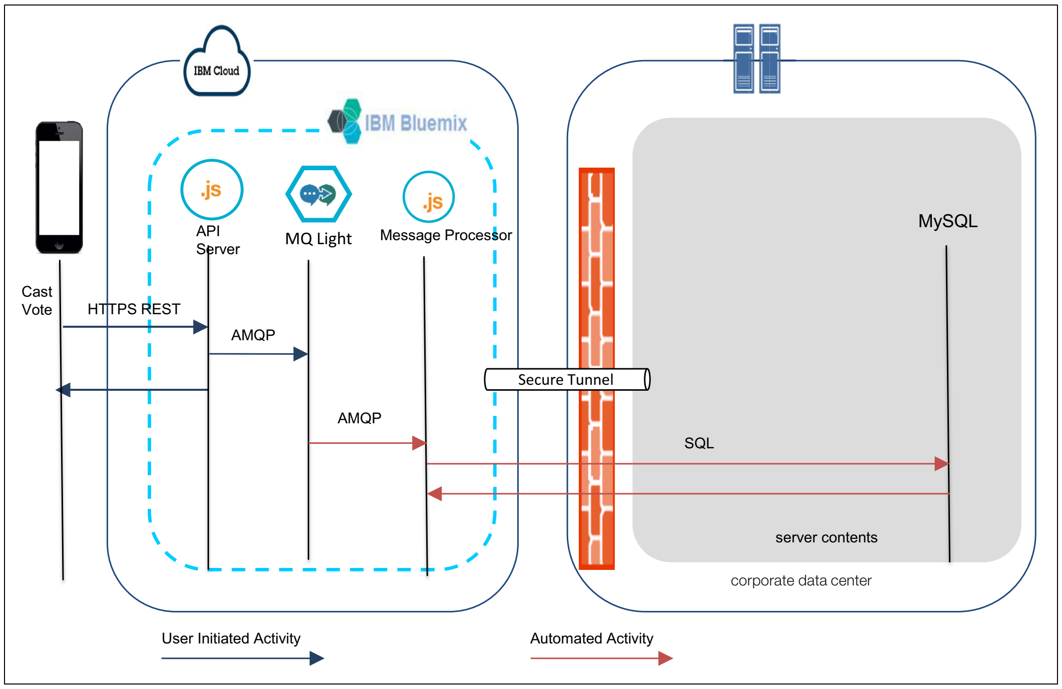


Figure 6-49 Sub scenario 2: Flow of information from user to database

This subscenario offers improved performance and scalability for the voting application. Notice that there in no adverse impact on the user interface. The performance of the operation improves and a larger number of requests can be accommodated.

6.10 Implementing subscenario 3

In this subscenario, the enterprise IBM MQ is used in place of the IBM MQ Light service. The message processing application performs the same functions by using the IBM MQ server. It reuses the components used in the previous subscenario and moves the message processing application from Bluemix to the corporate data center.

This scenario assumes that the user has an existing enterprise IBM MQ server 8.0.0.4 running on a Linux server.

The components used by subscenario 3 are shown in Figure 6-50.

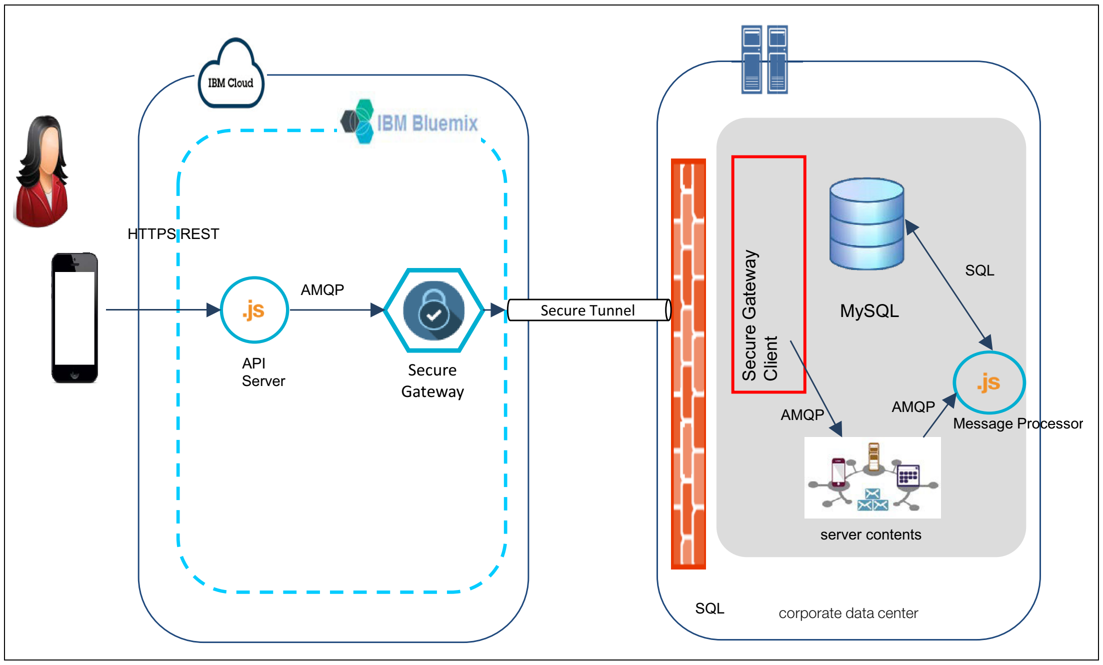


Figure 6-50 Components used by subscenario 3

6.10.1 Configuring IBM MQ for IBM MQ Light APIs

The IBM MQ Light API is based on the OASIS Standard AMQP Version 1.0 wire protocol. AMQP specifies how messages are sent between senders and receivers. An application acts as a sender when the application sends a message to a message broker, such as IBM MQ. IBM MQ acts as a sender when it sends a message to an AMQP application.

You must install the AMQP service component by using the IBM MQ V8.0.0.4 manufacturing refresh, not the V8.0.0.4 Fix Pack. You cannot install the AMQP component on a version of the queue manager earlier than V8.0.0.4.

Creating and using AMQP channels

When you install the IBM MQ support for IBM MQ Light APIs into your IBM MQ installation, you can run IBM MQSC commands (**`runmqsc`**) to define, alter, delete, start, and stop a channel. You can also view the status of a channel.

By defining and starting an AMQP channel, IBM MQ Light or AMQP 1.0 applications can publish messages that are received by existing IBM MQ applications. Messages published through an AMQP channel are all sent to IBM MQ topics, not IBM MQ queues. An IBM MQ application that has created a subscription using the MQSUB API call receives messages published by AMQP 1.0 applications. This occurs if the topic string or topic object used by the IBM MQ application matches the topic string published by the AMQP client.

The IBM MQ Knowledge Center link describes the procedure to configure and start AMQP services:

`http://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.amqp.doc/tamqp_creating.htm?lang=en`

The IBM Knowledge Center link also contains instructions about how to install the IBM MQ Light Node.js client and how to use sample sender and receiver programs to test the IBM MQ Light messages.

6.10.2 Creating a secure connection to enterprise IBM MQ

Use the instructions provided in section 6.8.4, “Configure a secure tunnel between the API server and database” on page 130 for details about how to configure a secure connection.

The following steps are different than the ones in “Setting up Secure Gateway” on page 130. For the rest of the steps, use the steps given in “Setting up Secure Gateway” on page 130.

1. In step 3, click **DASHBOARD**. We reuse the Bluemix Secure Gateway Service.
2. In step 4, click existing **Secure Gateway** service.
3. In step 9, note the <*Secure GATEWAY Id for MQ Server*>.
4. In step 13, provide Name as “MQ Server”, Host Name as <*TCP/IP Address or Host Name for MQ Server*>, and Port as 5672.
5. In step 15, note the <*Cloud Host for MQ Server*> and <*Cloud Port for MQ server*>.

Note: <*Cloud Host for MQ Server*> and <*Cloud Port for MQ Server*> values derived in the previous step will be used for configuration in later sections. It is suggested that these values be saved somewhere for later use.

Configurations for IBM MQ

In this section, we follow instructions at the URL below to install and configure the secure client gateway on your IBM MQ, the one running on <*virtual address of the MQ Server*>*.* It is assumed that the IBM MQ server was created using the same key that was used for the MySQL server. It is assumed that the user is logged in as ibmcloud and has sudo access on the server:

https://www.ng.bluemix.net/docs/services/SecureGateway/sg_021.html#sg_025

1. Connect to the IBM MQ Linux server and open a command prompt. Type the following command:

   ```
   mkdir /home/ibmcloud/securegateway
   ```

2. From your workstation, copy ***<IBM Secure gateway client installer>*** to the `/home/ibmcloud/securegateway` folder, for example, on a Linux workstation.

   `scp -i mqlightkey i`***<IBM Secure gateway client installer>*** `ibmcloud@<`*TCP/IP Address or Host Name for MQ Server*`>/home/ibmcloud/securegateway`

3. Connect to the MySQL server and open a command prompt. Type the following command:

   ```
   cd /home/ibmcloud/securegateway
   sudo dpkg -i ibm-securegateway-client-1.3.2+client_amd64.deb
   ```

   a. When prompted for the gateway ID, press Enter.

   b. When prompted, enter the configuration’s ID security token (if any): press Enter.

   c. When prompted `Do you wish to change the startup logging level, values are INFO, DEBUG, ERROR or TRACE - [INFO]:`, enter the configuration’s ID security token (if any). Press Enter.

   d. When prompted `Supply an ACL File for processing [optional]:`, press Enter.

   e. The following message is displayed at the end of this process. [*postinst] Completed with SUCCESS*.

4. Connect to MySQL server and open a command prompt:

   ```
   cd /opt/ibm/securegateway
   ```

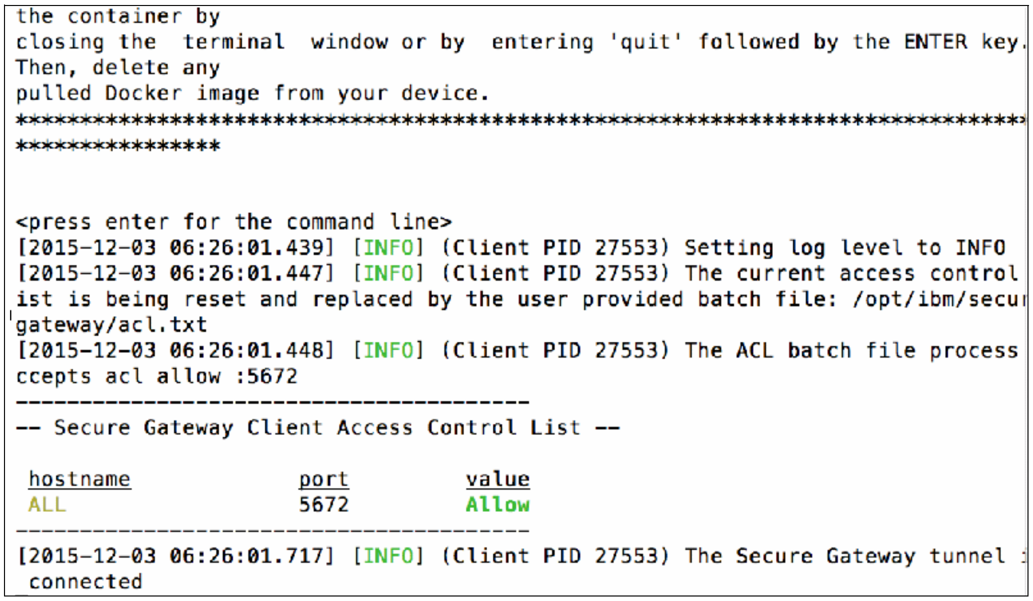
5. Create an `acl.txt` file by using your preferred editor.
6. Enter the following line:

   ```
   acl allow :5672
   ```

7. Connect to MySQL server and open a command prompt. Type the following commands:

   ```
   sudo su secgwadmin
   cd /opt/ibm/securegateway
   node lib/secgwclient.js <Secure GATEWAY ID for MQ Server> --F
   /opt/ibm/securegateway/acl.txt
   ```

8. The gateway starts and should show details as shown in Figure 6-51.



```
the container by
closing the  terminal  window or by  entering 'quit' followed by the ENTER key.
Then, delete any
pulled Docker image from your device.
*******************************************************************************
****************


<press enter for the command line>
[2015-12-03 06:26:01.439] [INFO] (Client PID 27553) Setting log level to INFO
[2015-12-03 06:26:01.447] [INFO] (Client PID 27553) The current access control
ist is being reset and replaced by the user provided batch file: /opt/ibm/secu
gateway/acl.txt
[2015-12-03 06:26:01.448] [INFO] (Client PID 27553) The ACL batch file process
ccepts acl allow :5672
----------------------------------------
-- Secure Gateway Client Access Control List --

 hostname             port         value
 ALL                  5672         Allow
----------------------------------------
[2015-12-03 06:26:01.717] [INFO] (Client PID 27553) The Secure Gateway tunnel
 connected
```

Figure 6-51   IBM Secure Gateway client started on IBM MQ server

The API server application running on Node.js in Bluemix can now connect to the corporate IBM MQ server using the secure tunnel.

6.10.3 Configuring the API server to use the enterprise IBM MQ server

Perform the following steps to configure the API server to use the enterprise IBM MQ server:

1. Open the following URL:

   https://console.ng.bluemix.net

2. Log in using your IBM user ID and password.
3. Click **DASHBOARD**.
4. Click **redbook-MQLight-service-1** under Applications.
5. Click **STOP**.
6. Click **Environment Variables**.
7. Click **USER-DEFINED**.
8. Replace with the value that matches your configurations, as shown in Table 6-11.

Table 6-11 Parameters for the API Server

Attribute	Default/Old	New Value	Purpose
DBHOST	*<virtual address of the MYSQL Server>*	*<Cloud Host for MySQL Server>*	Host name or IP address of MySQL server
DBPORT	3306	*<Cloud Port for MySQL Server>*	Port number for MySQL database
DBNAME	mqlight	mqlight	MySQL database/schema name
DBUSER	admin	admin	MySQL user
DBPASSWORD	passw0rd	passw0rd	MySQL password
TOPIC	redbook/vote	redbook/vote	Topic from IBM MQ Light/IBM MQ server
MQHOST	<>	*<Cloud Host for MQ Server>*	IBM MQ server host name or IP server
MQPORT	5672	*<Cloud Port for MQ server>*	Port number for IBM MQ server AMQP Listener
MQUSER	<>	*<User Id to connect to MQ Server>*	User ID to connect to IBM MQ server
MQPASSWORD	<>	*<Password to connect MQ server>*	Password to connect to IBM MQ server
MQSERVICE	ampq://localhost	ampq://localhost	Local MQ Light server
MQUSESERVER	BLUEMIX	CLOUD	Valid values: LOCAL: Local MQ Light server BLUEMIX: Bluemix service CLOUD: IBM MQ server
MQID	Send001	Send001	Sender ID
SAVEMETHOD	MQ	MQ	Valid values: DB: Direct to database MQ: To IBM MQ Light/ IBM MQ Server
RESULTTOPIC	redbook/results	redbook/results	Used in subscenario 4 to publish the leader board
ENABLEENTERPRIS E	false	false	The valid values are “true” (only for subscenario 4) or “false” for all other subscenarios

9. Scroll to bottom of the page and click **SAVE**.
10. Click **Overview**.

11. Click **START.**

The API application running on Bluemix is now recording a vote by using the corporate IBM MQ server on Bluemix.

6.10.4 Installing the message processor on a corporate server

This message processor is a Node.js application. It requires Node.js installation. The steps required to install Node.js in your environment are included in 5.2.2, “Node.js download and installation” on page 91.

The message processing application listens for IBM MQ Light messages and saves them to MySQL database. It assumes that Node.js is already installed on a Linux server. This is the same application that was used in subscenario 2 on Bluemix. The only changes required are the parameters that are used to call the application. We call this server as corporate message processor and the address for the server to be *<Host name or IP address of the corporate message processor>*.

Perform the following steps to install the message processor on a corporate server:

1. Use the code that was downloaded in 6.8.3, “Deploy the API server” on page 125.
2. Change the directory to the `message-processing` folder.
3. Type the following command:

```
DBHOST=<virtual address of the MySQL Server>  DBPORT=3306 DBNAME="mqlight"
DBUSER="admin" DBPASSWORD="passw0rd" TOPIC="redbook/vote" MQHOST=<TCP/IP
Address or Host Name for MQ Server> MQPORT=5672 MQUSER=<User Id to connect to MQ
Server> MQPASSWORD=<Password to connect MQ server> MQSERVICE="amqp://localhost"
MQUSESERVER="CLOUD" MQID="Recv001" SAVEMETHOD="MQ" node app.js
```

4. Table 6-12 tabulates the parameters that are used for running the message processor on the corporate server.

Table 6-12 Parameters for the receiver server in the corporate data center

Attribute	Default/Old	New Value	Purpose
DBHOST	*<virtual address of the MYSQL Server>*	*<virtual address of the MySQL Server>*	Host name or IP address of MySQL server
DBPORT	3306	3306	Port number for MySQL database
DBNAME	mqlight	mqlight	MySQL database/schema name
DBUSER	admin	admin	MySQL user
DBPASSWORD	passw0rd	passw0rd	MySQL password
TOPIC	redbook/vote	redbook/vote	Topic from IBM MQ Light/IBM MQ server
MQHOST	<>	*<TCP/IP Address or Host Name for MQ Server>*	IBM MQ server host name or IP server

Attribute	Default/Old	New Value	Purpose
MQPORT	5672	5672	Port number for IBM MQ server AMQP Listener
MQUSER	<>	*<User Id to connect to MQ Server>*	User ID to connect to IBM MQ server
MQPASSWORD	<>	*<Password to connect MQ server>*	Password to connect to IBM MQ server
MQSERVICE	ampq://localhost	ampq://localhost	Local IBM MQ Light server
MQUSESERVER	LOCAL	CLOUD	Valid values: LOCAL: Local MQ Light Server BLUEMIX: Bluemix service CLOUD: IBM MQ server
MQID	Send001	Send001	Sender ID
SAVEMETHOD	MQ	MQ	Valid values: DB: Direct to database MQ: To MQ Light/IBM MQ Server

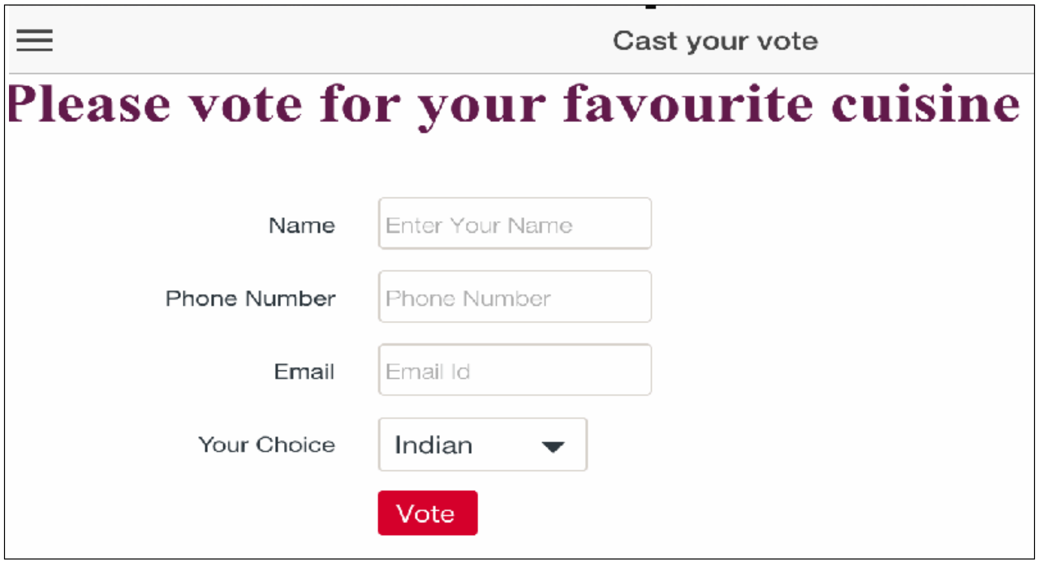
5. The message processor is not running in the corporate data center. It is reading messages by using direct connection to the corporate IBM MQ server and writing the appropriate data to the MySQL server.

6.10.5 Testing the application

The application can be tested by using the following steps:

1. Open a new browser window or tab and type in the following URL: *<Web application Host Name>/redbookApp.*

2. Enter name, phone number, email, and select a choice from the drop-down list, as shown in Figure 6-52.



Cast your vote

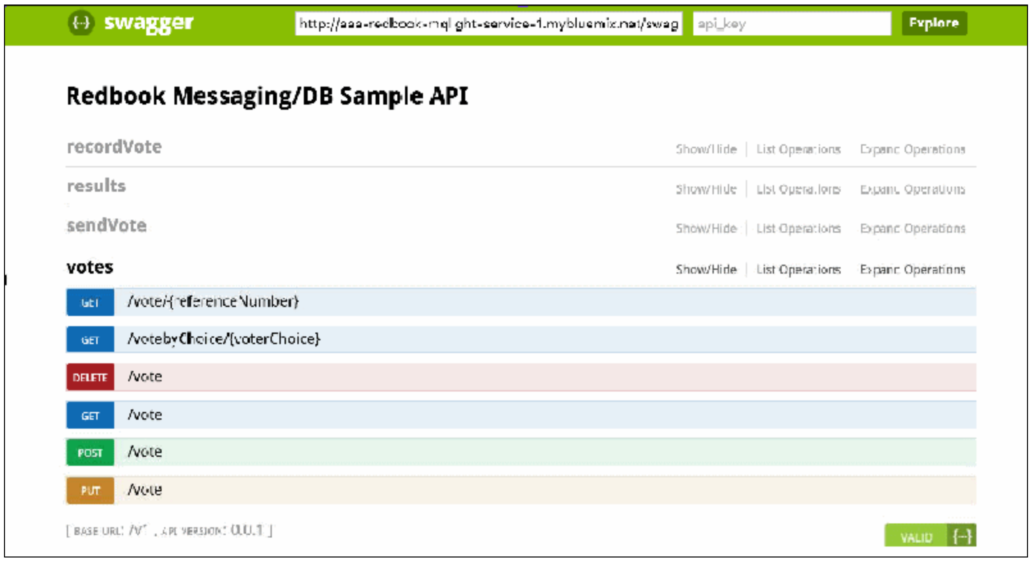
Please vote for your favourite cuisine

Name	Enter Your Name
Phone Number	Phone Number
Email	Email Id
Your Choice	Indian

Figure 6-52 Voting application: Cast your vote

3. Click **Vote**.
4. The application responds with a message of thanks and displays a time stamp for the vote.
5. The vote is now saved to the corporate IBM MQ server by the web application. The message processing application now running inside the corporate data center processes the message from the corporate IBM MQ server and posts it to the local database.
6. Open a new browser window or tab and type in the URL: http://*<API Server Host Name>/api-docs/#*

   The API Server Test Console is shown as in Figure 6-17 on page 129.



Redbook Messaging/DB Sample API

recordVote — Show/Hide | List Operations | Expand Operations

results — Show/Hide | List Operations | Expand Operations

sendVote — Show/Hide | List Operations | Expand Operations

votes — Show/Hide | List Operations | Expand Operations

GET	/vote/{referenceNumber}
GET	/votebyChoice/{voterChoice}
DELETE	/vote
GET	/vote
POST	/vote
PUT	/vote

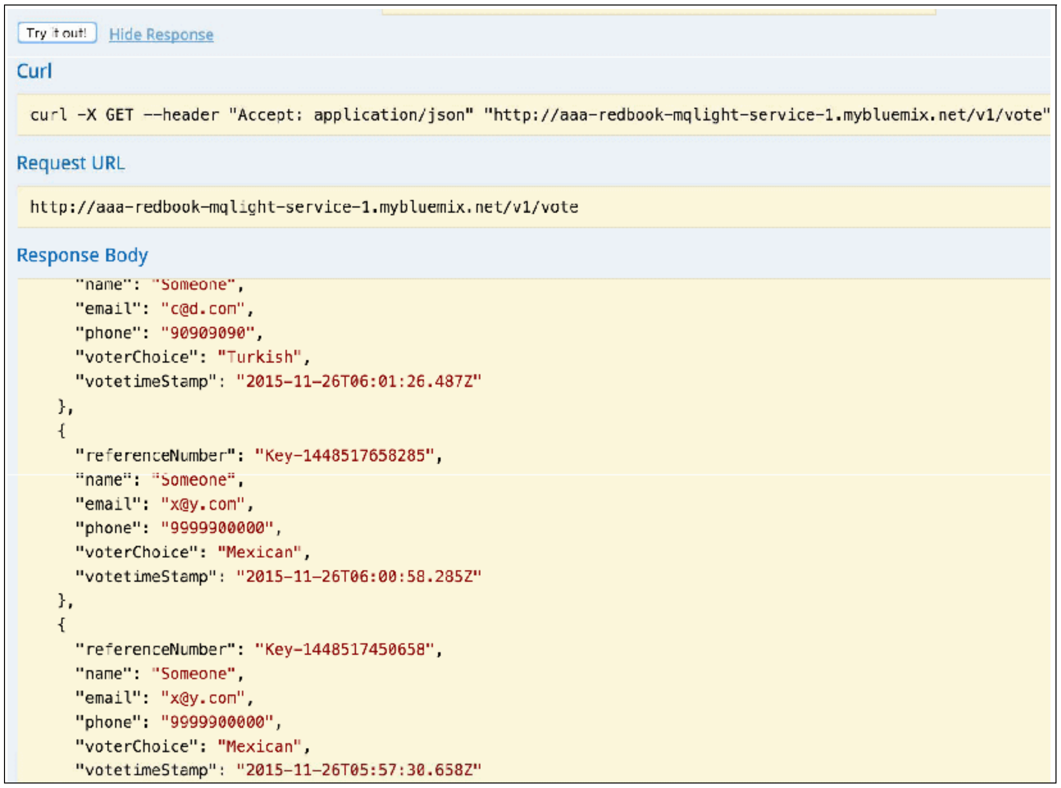
[BASE URL: /v1 , API VERSION: 0.0.1]

Figure 6-53 API Server Test Console

7. Click **votes**. This opens a list of APIs for votes.

8. Click **GET /votes**. This opens the API details.
9. Click **Try it out!**

   The database records from the MySQL database are shown as a JSON Array, as shown in Figure 6-54.


Try it out! Hide Response

Curl

```
curl -X GET --header "Accept: application/json" "http://aaa-redbook-mqlight-service-1.mybluemix.net/v1/vote"
```

Request URL

```
http://aaa-redbook-mqlight-service-1.mybluemix.net/v1/vote
```

Response Body

```
    "name": "Someone",
    "email": "c@d.com",
    "phone": "90909090",
    "voterChoice": "Turkish",
    "votetimeStamp": "2015-11-26T06:01:26.487Z"
  },
  {
    "referenceNumber": "Key-1448517658285",
    "name": "Someone",
    "email": "x@y.com",
    "phone": "9999900000",
    "voterChoice": "Mexican",
    "votetimeStamp": "2015-11-26T06:00:58.285Z"
  },
  {
    "referenceNumber": "Key-1448517450658",
    "name": "Someone",
    "email": "x@y.com",
    "phone": "9999900000",
    "voterChoice": "Mexican",
    "votetimeStamp": "2015-11-26T05:57:30.658Z"
```

Figure 6-54   Details from the API server

10. Notice that the votes you cast should now appear in this list.
11. To view the last message processed by the message processor, try the URL on a browser: *<Message Processor Host Name data center>:3000/lastMessage.*
12. This URL displays a JSON object as shown below:

```
{"name":"John","phone":"9999-1010","email":"c@c.com","voterChoice":"French","vo
tetimeStamp":"2015-11-30T02:54:32.088Z","referenceNumber":"Key-1448852186372"}
```

13. The message processor picked the vote from the IBM MQ server and saved it to the MySQL database.

6.10.6 Summary of subscenario 3

In this subscenario, the data captured by the web application is stored directly into the corporate IBM MQ. It reduces the time taken by the API server to process client requests. It improves performance and scalability of the solution. The important point to note is that no code changes were required. The same user interface application, the API server, and the message processor were reused. The API server and message provider were configured with sightly different parameters.

Following are the main differences from subscenario 2:

- The corporate IBM MQ inside the corporate data center was used instead of the IBM MQ Light service on Bluemix.
- The message processor now runs in the corporate data center.

The result for the business is still the same. Each vote is recorded in a MySQL database. The user does not see any changes to the application.

Figure 6-55 shows the flow of information from the user to the database.

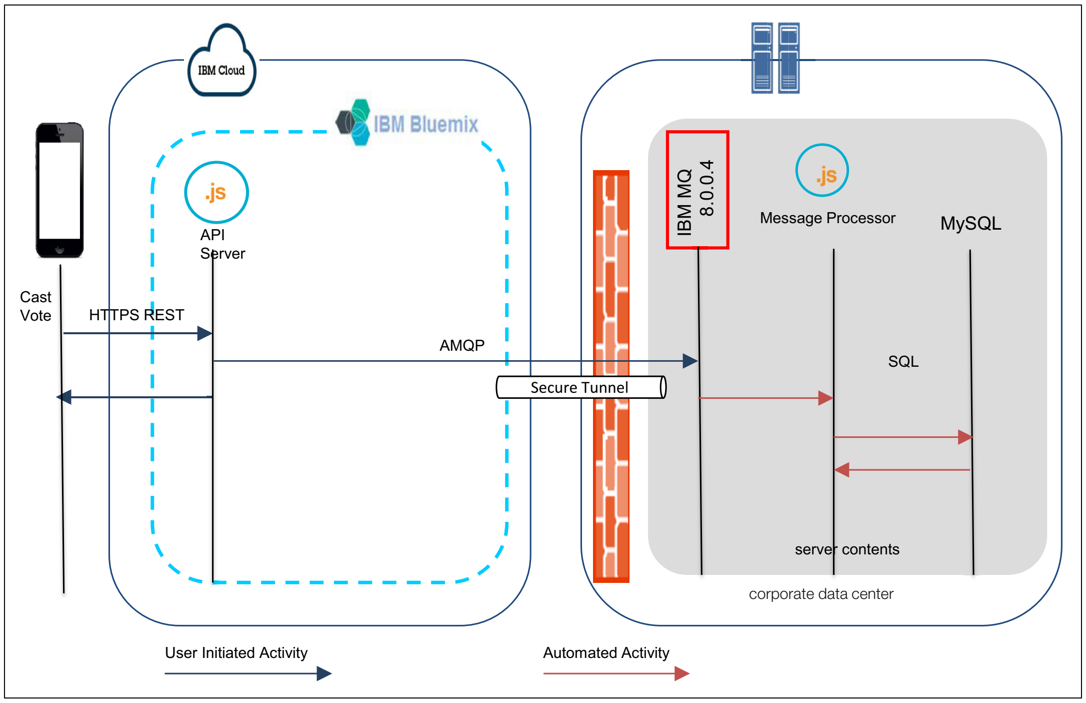


Figure 6-55 Subscenario 3: Flow of information from user to database

This subscenario offers improved performance and scalability for the voting application as compared to subscenario 1. An external event (casting of a vote) is now available as an event inside the corporate data center and can be used by the enterprise applications.

6.11 Implementing subscenario 4

In this subscenario, the API server records the user's vote on the enterprise IBM MQ. The enterprise applications' process the information received from the user. The enterprise application uses the IBM MQ, IBM Integration Bus, and IBM DB2 databases.

This section has been developed assuming that you are familiar with the concepts and operations of IBM MQ, IBM Integration Bus, and IBM DB2 databases.

The enterprise application records the data to a database and in addition:

- Publishes the latest results to IBM MQ.
- Provides a REST service that can be used to query a list of votes by voter's choice.

The API server provides the capability to publish a vote to IBM MQ and in addition:

- Listens for the latest results from IBM MQ.
- When a result is updated, it caches the latest results.
- When a user requests the latest results using the REST API, it provides the latest cached copy.

The web application allows a user to cast their vote and in addition:

- Allows a user to view the latest results
- Allows a user to see the list of votes by voter choice

Figure 6-56 shows the components that are used by this subscenario.

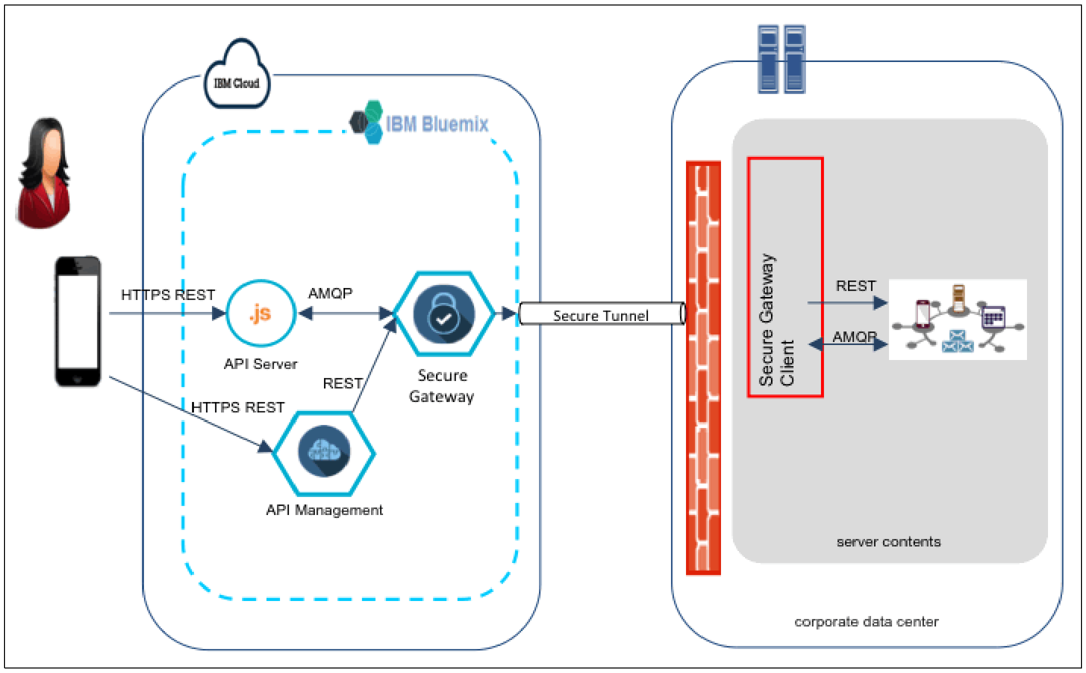


Figure 6-56 Components used by subscenario 4

6.11.1 Stopping the message processor application in a corporate gateway

The message processing server used in subscenario 3 can optionally be switched off. The message received by IBM MQ will be processed directly by the enterprise application. If the message processor server is running for this subscenario, the vote details are added to both the MySQL database and IBM DB2 database.

6.11.2 Configuring the API server to use the enterprise IBM MQ server and receive results

Perform the following steps to configure the API server to use the enterprise IBM MQ server and receive results:

1. Open the following URL:

   `https://console.ng.bluemix.net`

2. Log in using your IBM user ID and password.

3. Click **DASHBOARD**.
4. Click **redbook-MQLight-service-1** under Applications.
5. Click **STOP**.
6. Click **Environment Variables**.
7. Click **USER-DEFINED.**
8. Replace with the value that matches your configurations, as shown in Table 6-13.

Table 6-13 Parameters for the API server

Attribute	Default/Old	New value	Purpose
DBHOST	*<virtual address of the MYSQL Server>*	*<Cloud Host for MySQL Server>*	Host name or IP address of MySQL server
DBPORT	3306	*<Cloud Port for MySQL Server>*	Port number for MySQL database
DBNAME	mqlight	mqlight	MySQL database/schema name
DBUSER	admin	admin	MySQL user
DBPASSWORD	passw0rd	passw0rd	MySQL password
TOPIC	redbook/vote	redbook/vote	Topic from IBM MQ Light/IBM MQ server
MQHOST	<>	*<Cloud Host for MQ Server>*	IBM MQ server host name or IP server
MQPORT	5672	*<Cloud Port for MQ server>*	Port number for IBM MQ server AMQP Listener
MQUSER	<>	*<User Id to connect to MQ Server>*	User ID to connect to IBM MQ server
MQPASSWORD	<>	*<Password to connect MQ server>*	Password to connect to IBM MQ server
MQSERVICE	ampq://localhost	ampq://localhost	Local IBM MQ Light server
MQUSESERVER	LOCAL	CLOUD	Valid values: LOCAL: Local IBM MQ Light server BLUEMIX: Bluemix service CLOUD: IBM MQ server
MQID	Send001	Send001	Sender ID
SAVEMETHOD	MQ	MQ	Valid values: DB: Direct to database MQ: To IBM MQ Light/IBM MQ Server

Attribute	Default/Old	New value	Purpose
RESULTTOPIC	redbook/results	redbook/results	Used in subscenario 4 to publish the leader board
ENABLEENTERPRISE	false	true	The valid values are “true” (only for subscenario 4) or “false” for all other subscenarios

9. Scroll to the bottom of the page and click **SAVE**.
10. Click **Overview**.
11. Click **START**.
12. The API application running on Bluemix is now recording a vote using the corporate IBM MQ server on Bluemix. It also processes the results from the IBM MQ server that are being sent by IBM Integration Bus.

6.11.3 Configuring IBM MQ

In this subscenario, we create the subscription in IBM MQ to receive messages into a local queue as published by the IBM MQ Light application on Bluemix.

The example below shows the **runmqsc** command to create a subscription with the destination as a local queue.

Example 6-3 runmqsc command

```
runmqsc <QMGR>
def SUB (Redbook) TOPICSTR ('redbook/vote') DEST('SUBQ')
```

When an IBM MQ Light application publishes messages on the topic “redbook/vote”, the messages are put on to the IBM MQ queue “SUBQ”.

6.11.4 Developing the IBM Integration Bus message flows

This scenario requires three distinct operations that the Integration Bus needs to handle:

- Receiving the message from the IBM MQ queue and updating the backend DB2 database.
- On successful update of the record to the database, publishing the tally of votes to an IBM MQ topic.
- Providing the REST API interface to query all votes or the votes by user-defined choice.

The IBM Integration Bus code used in this subscenario has been provided in the IIB-code folder that was created in 6.8.3, “Deploy the API server” on page 125.

Developing message flow to process IBM MQ queue-based messages

The application running on Bluemix publishes messages over the AMQP channel to the on-premises IBM MQ, which is configured to listen for IBM MQ Light messages for a specified topic. As explained in the previous section, the subscription defined on IBM MQ sends the IBM MQ Light message to the destination queue “SUBQ”. Therefore, we configure our IBM MQ Input endpoint to read messages from this queue.

Figure 6-57 shows the message flow design to process this message.

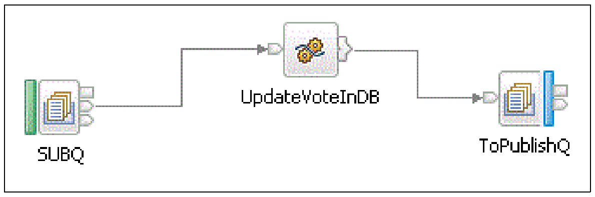
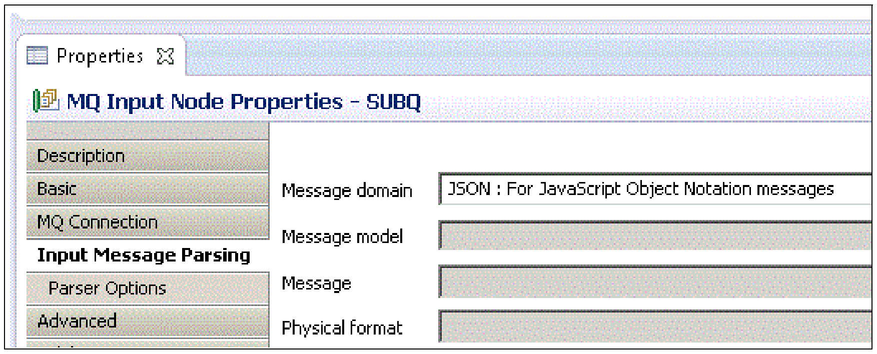


Figure 6-57 Message flow to read IBM MQ messages

Because the messages arriving from the sender application over the AMQP channel are in JSON format, we set the “Message domain” property as JSON on the MQ Input node labeled “SUBQ” as shown in Figure 6-58.



Message domain: JSON : For JavaScript Object Notation messages
Message model:
Message:
Physical format:

Figure 6-58 MQ Input Node Properties window

In the compute node labeled *Update VoteInDB,* we extract the fields from the input message and form the SQL query to update the record in the database. The excerpt of the code showing the parsing of JSON input data and SQL query to insert the data is shown in Example 6-4.

Example 6-4 ESQL code to read JSON input message and insert to database

```
SET RefNum = 'key-1001';
SET name = InputRoot.JSON.Data.name;
SET phone = InputRoot.JSON.Data.phone;
SET email = InputRoot.JSON.Data.email;
SET voterChoice = InputRoot.JSON.Data.voterChoice;
SET votertimeStamp = InputRoot.JSON.Data.votetimeStamp;

INSERT INTO Database.CustomerVotes values ( RefNum, name, email, phone,
voterChoice, votertimeStamp);
```

The message is then put to the output queue labeled as “IN_PUBLISH,” which also serves as input queue to drive another message flow that will publish the votes tally to the IBM MQ topic.

Developing message flow to publish a message to IBM MQ topic

The scenario requires that the latest tally of the votes is published when there is a new vote casted. Therefore, we create a message flow, which gets triggered when a new vote gets added to the voter’s database described in the above section.

Figure 6-59 shows the message flow design to publish the message to an IBM MQ topic.

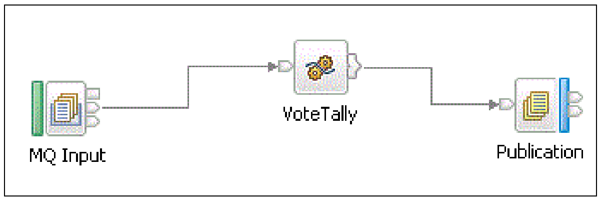
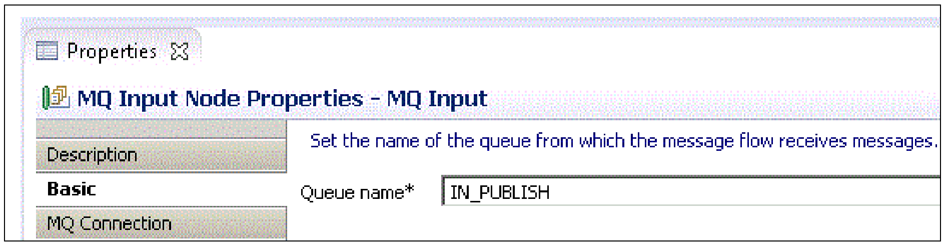


Figure 6-59 Message flow to publish to a topic

We configure the IBM MQ Input node with the properties shown in Figure 6-60.



MQ Input Node Properties - MQ Input

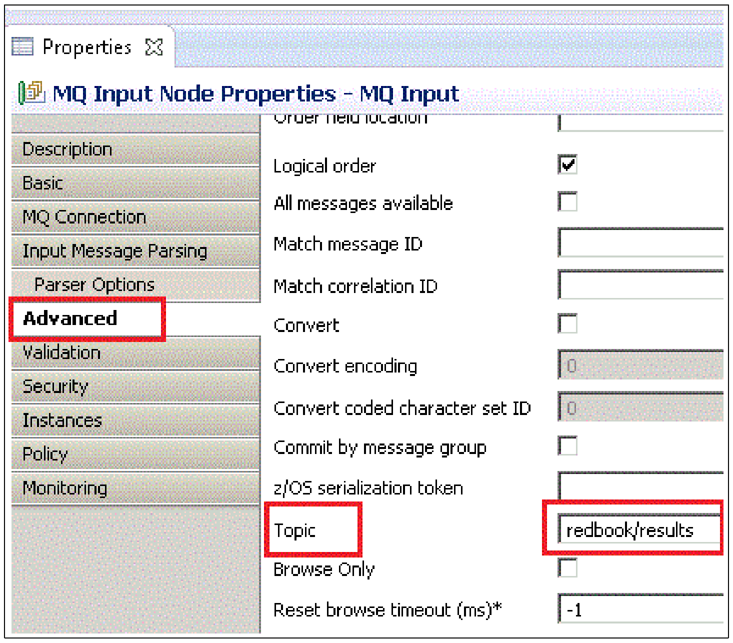
Set the name of the queue from which the message flow receives messages.

Queue name* IN_PUBLISH

Figure 6-60 Input queue for the Publish message flow

The output of the previous message flow is specified as input queue to this message flow.

In the Advanced tab of Input node, add the specific name of the topic to which the message will be published. Figure 6-61 shows the Topic is set to “redbook/results”.



MQ Input Node Properties - MQ Input

Logical order	✓
All messages available	
Match message ID	
Match correlation ID	
Convert	
Convert encoding	0
Convert coded character set ID	0
Commit by message group	
z/OS serialization token	
Topic	redbook/results
Browse Only	
Reset browse timeout (ms)*	-1

Figure 6-61 MQ Input Node Properties to set topic string

The compute node labeled as “Vote Tally” retrieves the vote tally from the database and constructs the output message in JSON format, as shown in Example 6-5.

Example 6-5 ESQL code to construct a JSON Array output message

```
CALL CopyMessageHeaders();
DECLARE myRow ROW;
SET myRow.vote[] = PASSTHRU ('SELECT count(*) as VoteCount, voterChoice FROM
Votes1 group by voterChoice order by VoteCount DESC');
SET OutputRoot.JSON.Data = myRow;
SET OutputRoot.JSON.Data TYPE = JSON.Array;
```

No configuration is required on Publication Node.

When this message flow gets triggered, the JSON message of the following format gets published to the IBM MQ topic “redbook/results” as configured on the MQInput node. A sample message is shown in Example 6-6.

Example 6-6 Sample message published by the message flow

```
[{"VOTECOUNT":3,"VOTERCHOICE":"Italian"},{"VOTECOUNT":2,"VOTERCHOICE":"Mexican"}]
```

On the IBM MQ side, we create a new subscription for the topic “redbook/results” as shown in Example 6-7.

Example 6-7 Define subscription topic on IBM MQ to receive results from Integration Bus

```
$runmqsc <Queue Manager Name>
def SUB('votesResults') TOPICSTR ('redbook/results') DESTCLAS(managed)
```

The AMQP channel configured on IBM MQ allows IBM MQ Light client applications to receive messages if they are subscribed to this topic.

Developing message flow as REST API for requesting data on demand

As an administrator of the front-end application, you might need to view the data for all the voters and also by specific choices. For this purpose, we expose the message flows as REST API, which fetches the information from the backend database and returns the results to the user.

The steps to import the Swagger 2.0 file into Integration Toolkit and implementing the subflow for each REST operation is explained in Chapter 3, “Introduction to IBM messaging and integration products” on page 37.

After implementing the REST operations (GET method), the message flow design is displayed as shown in Figure 6-62.

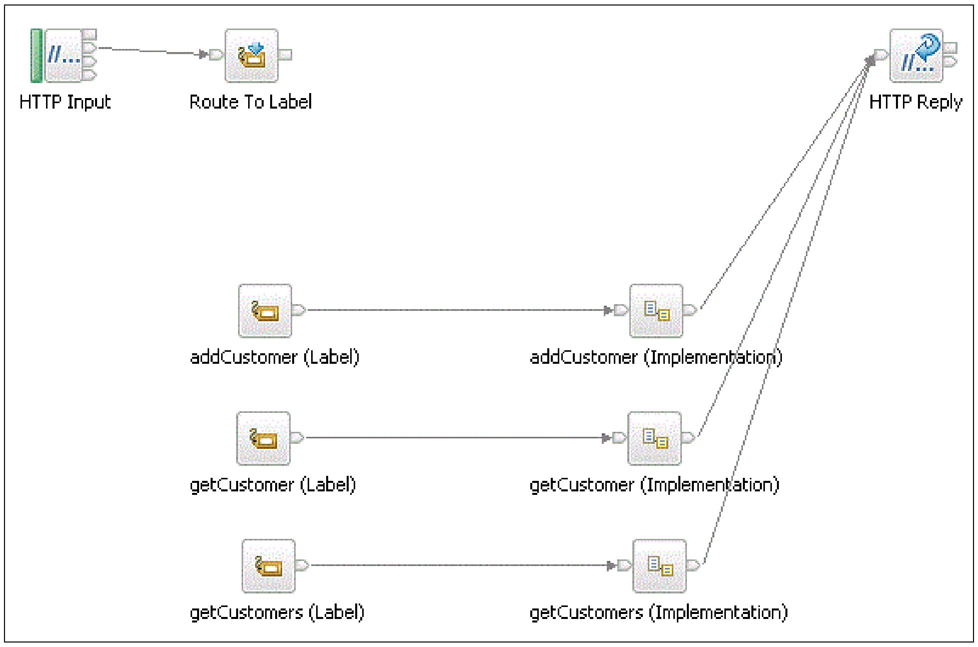


Figure 6-62   REST API project to implement customer data retrieval operations

The HTTP Input node listens for the request coming on the configured URL and routes to the respective subflow based on the operation contained in the request message.

The subflow implements a compute node to get the data from the backend database and forms an output message to be replied back to the requesting user.

For example, the getCustomer subflow has ESQL code as shown in Example 6-8 to generate the output message in JSON format.

Example 6-8   ESQL for receiving GET request and generating output message

```
DECLARE myRow ROW;
DECLARE voteChoice Char;
IF FIELDTYPE(InputLocalEnvironment.REST.Input.Parameters.customerId) IS NOT NULL
THEN
   SET voteChoice = InputLocalEnvironment.REST.Input.Parameters.customerId;
END IF;

IF InputLocalEnvironment.REST.Input.Method = 'GET' THEN
   SET myRow.vote[] = (SELECT T.RefNum as referenceNumber, T.name , T.email,
T.phone, T.voterChoice, T.votetimeStamp FROM Database.Votes1 AS T where
T.voterChoice=voteChoice );

   SET OutputRoot.JSON.Data = myRow;
   SET OutputRoot.JSON.Data TYPE = JSON.Array;
END IF;
```

Similarly, implement the operations in the other subflows according to the requirements of the user application.

Note: IBM Integration Bus V10 includes new capability that allows administrators to enable Cross-Origin Resource Sharing, or CORS. By enabling CORS support in IBM Integration Bus, web pages can make requests to services, such as REST APIs or integration services, which are hosted on IBM Integration Bus.

To enable CORS support, run the following command on your Integration Node:

HTTP listener for the integration server

```
mqsichangeproperties <IIBNODE> -e <Integration Server> -o HTTPConnector -n
corsEnabled -v true
```

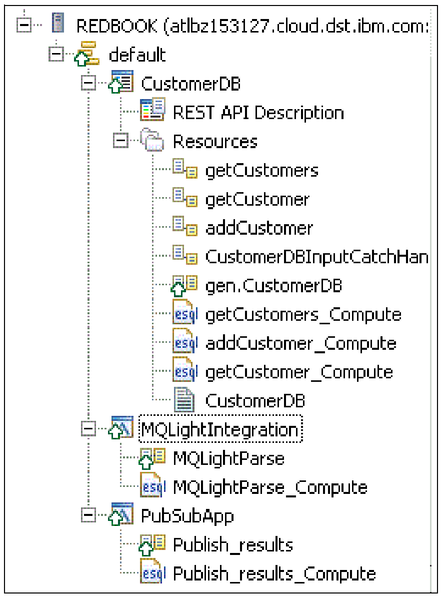
HTTP listener for the integration node (IIB v10.0.0.1 or later)

```
mqsichangeproperties <IIBNODE> -b httplistener -o HTTPConnector -n corsEnabled
-v true
```

For advanced CORS configuration options in Integration Bus, see the following link:

https://developer.ibm.com/integration/blog/2015/06/05/cross-origin-resource-sharing-cors-in-ibm-integration-bus-v10

The overall topology of the integration server after deploying all the three message flow applications looks as shown in Figure 6-63.



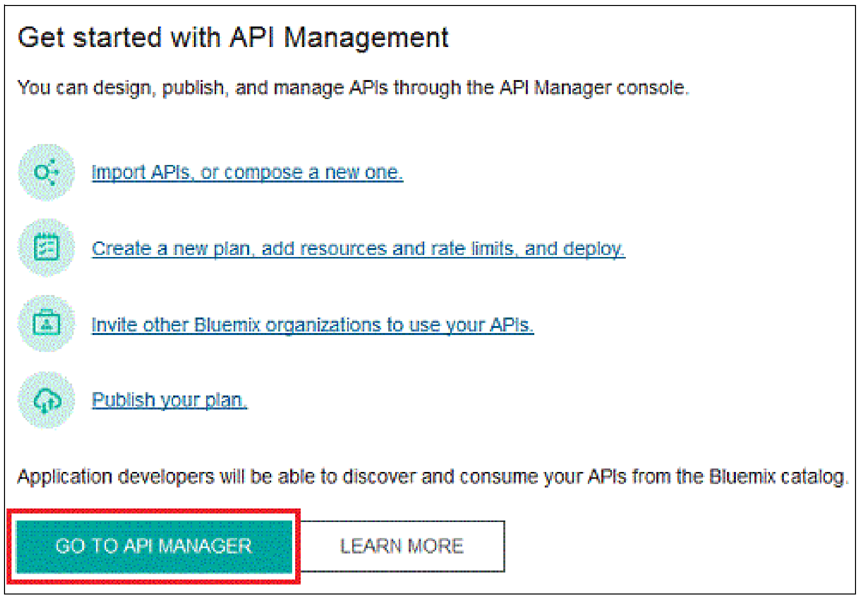
```
REDBOOK (atlbz153127.cloud.dst.ibm.com:
  default
    CustomerDB
      REST API Description
      Resources
        getCustomers
        getCustomer
        addCustomer
        CustomerDBInputCatchHan
        gen.CustomerDB
        getCustomers_Compute
        addCustomer_Compute
        getCustomer_Compute
        CustomerDB
    MQLightIntegration
      MQLightParse
      MQLightParse_Compute
    PubSubApp
      Publish_results
      Publish_results_Compute
```

Figure 6-63 Integration Server topology

6.11.5 Registering IBM Integration Bus REST API with Bluemix API Management Service

We now describe the steps to register Integration Bus REST API with API management service.

1. Create a new API management service in your Bluemix space and click **GO TO API MANAGER** as shown in Figure 6-64.



Get started with API Management

You can design, publish, and manage APIs through the API Manager console.

Import APIs, or compose a new one.

Create a new plan, add resources and rate limits, and deploy.

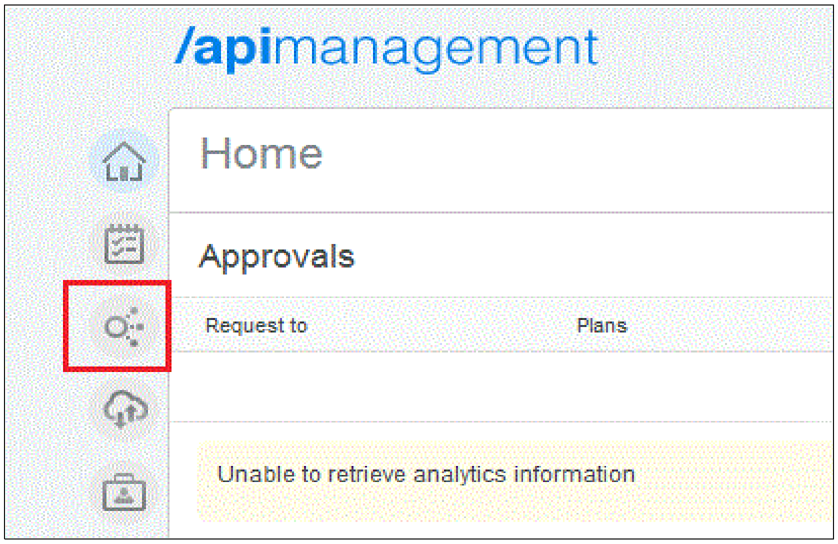
Invite other Bluemix organizations to use your APIs.

Publish your plan.

Application developers will be able to discover and consume your APIs from the Bluemix catalog.

Figure 6-64 Launch API Manager

2. Go to the APIs page by clicking the icon on the left-side menu as shown in Figure 6-65.



Unable to retrieve analytics information

Figure 6-65 Select APIs configuration tab

3. To add the API, click the plus sign (**+**) as shown in Figure 6-66.

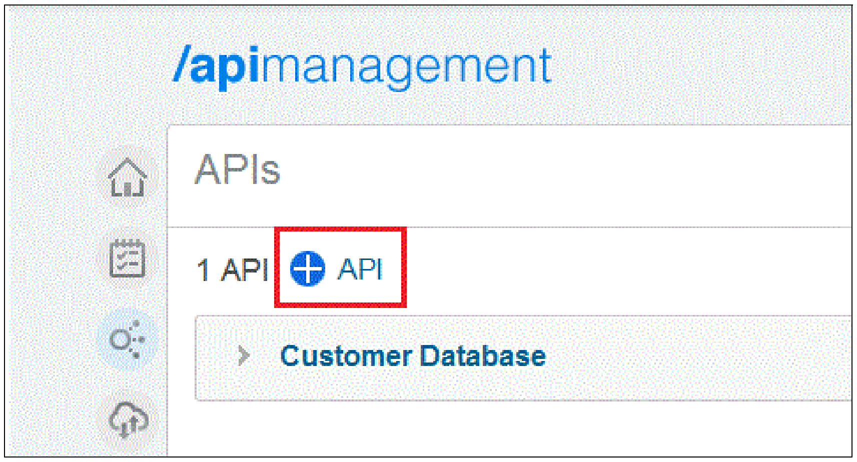


Figure 6-66 Add an API

4. Multiple options are available to add the APIs. We use the **Load a Swagger File** option to load the APIs from the Swagger file available on the local file system. We use the same Swagger definition JSON file that was used for generating the REST API message flow. Refer to Figure 6-67.

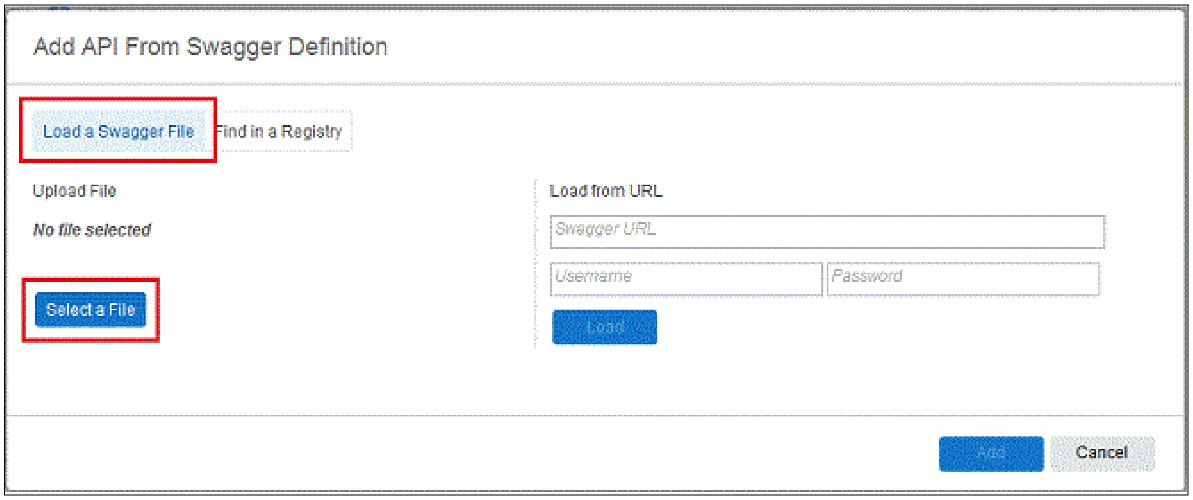
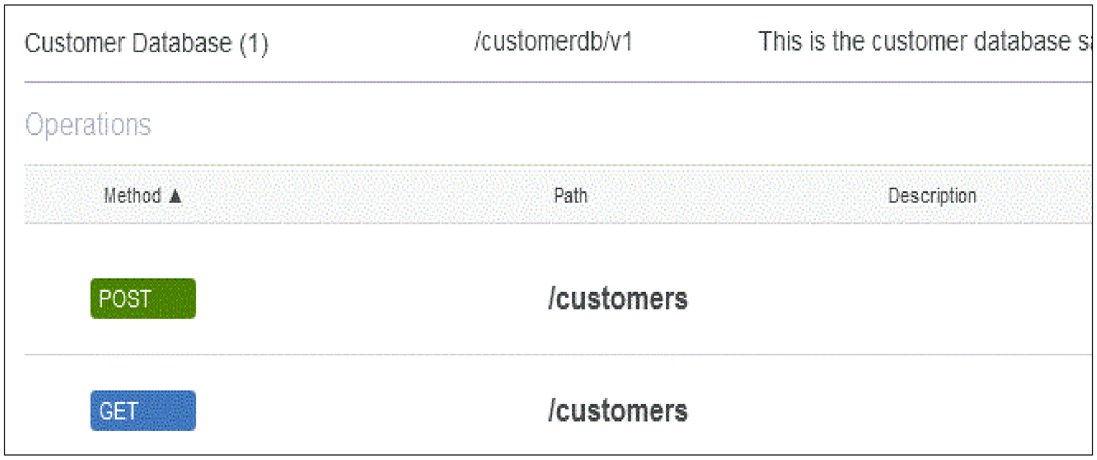


Figure 6-67 Load the Swagger JSON file

5. After the Swagger document is imported, the REST Operations appear as shown in Figure 6-68.

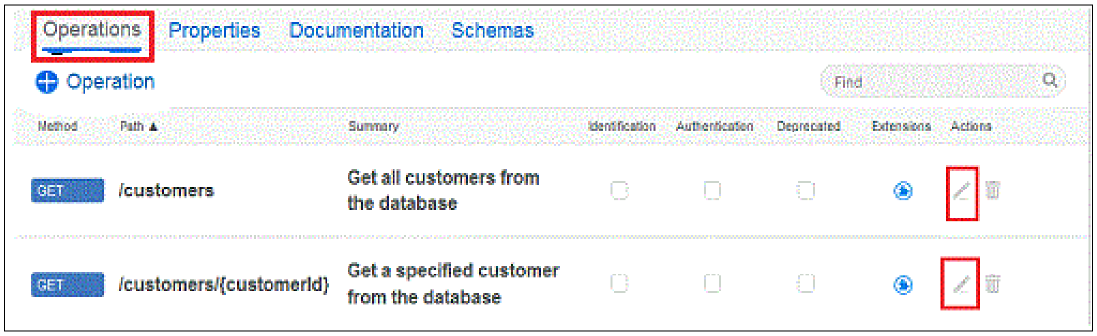

Customer Database (1) /customerdb/v1 This is the customer database s

Operations

Method ▲	Path	Description
POST	/customers	
GET	/customers	

Figure 6-68 API Operations that are loaded from the Swagger file

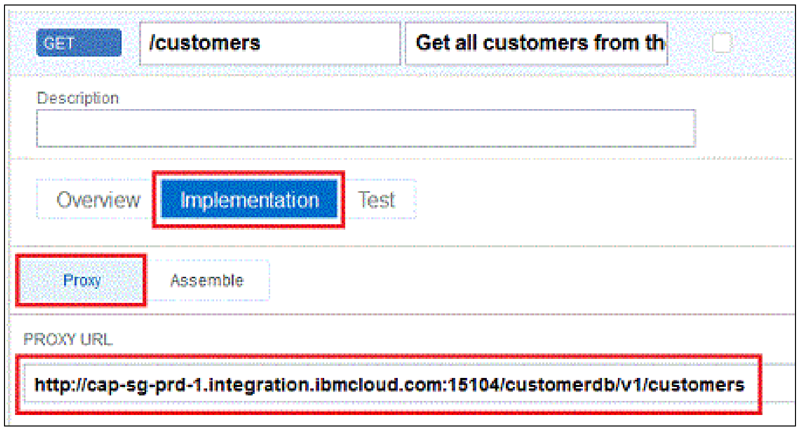
6. In the Operations section, click the **edit icon** to implement the proxy URL and click **Save**, as shown in Figure 6-69.



Method	Path ▲	Summary	Identification	Authentication	Deprecated	Extensions	Actions
GET	/customers	Get all customers from the database					
GET	/customers/{customerId}	Get a specified customer from the database					

Figure 6-69 Edit the operations to implement the Proxy URL

7. As shown in Figure 6-70, in the Proxy URL text box, replace the Integration Bus host name or IP and port with the Secure Gateway cloud host and port that is generated by the Secure Gateway when the destination was created for IBM Integration Bus REST API message flow. Similarly, perform the steps for all operations that you want to implement.



/customers Get all customers from th

Description

PROXY URL

http://cap-sg-prd-1.integration.ibmcloud.com:15104/customerdb/v1/customers

Figure 6-70 Add the Proxy URL for the REST method

8. Create a new plan to publish the above REST operations as shown in Figure 6-71.

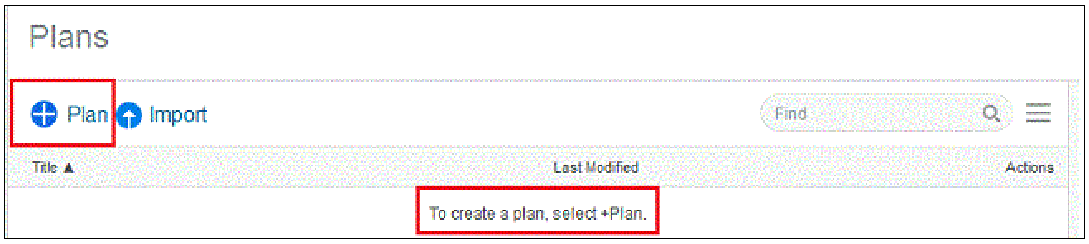


Figure 6-71 Create a plan

9. Add a title for the plan as shown in Figure 6-72.

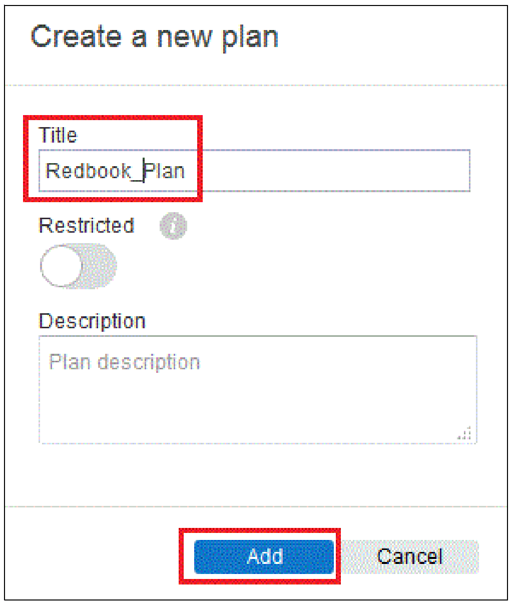


Figure 6-72 Provide a title to the plan

10. Select the operations that you want to add to the plan, as shown in Figure 6-73.

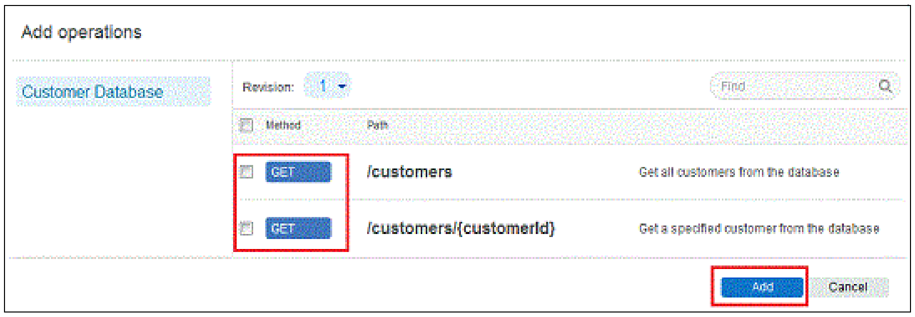


Figure 6-73 Adding operations to the plan

11.Save the plan and stage the plan to a sandbox environment as shown in Figure 6-74.

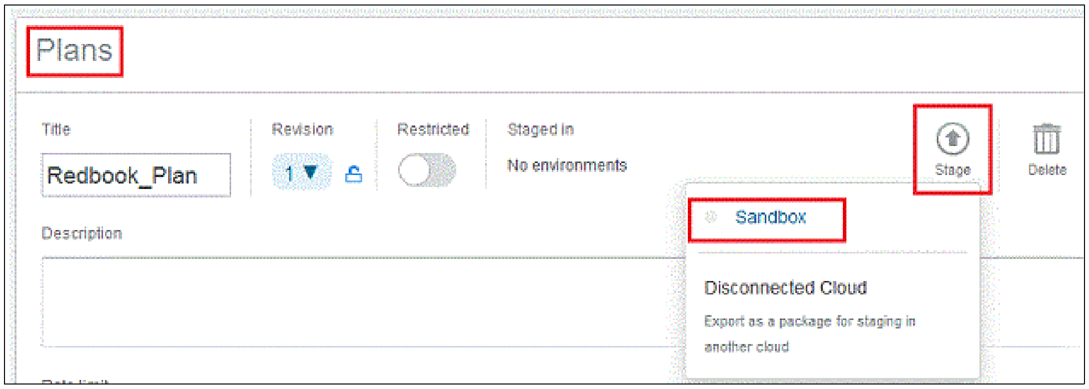


Figure 6-74   Save and stage the plan to a sandbox

12.After the plan is staged successfully, go to the **Management tab** from the left-side menu for publishing the plan. The Publish menu appears upon clicking the icon under **Actions**, as shown in Figure 6-75.

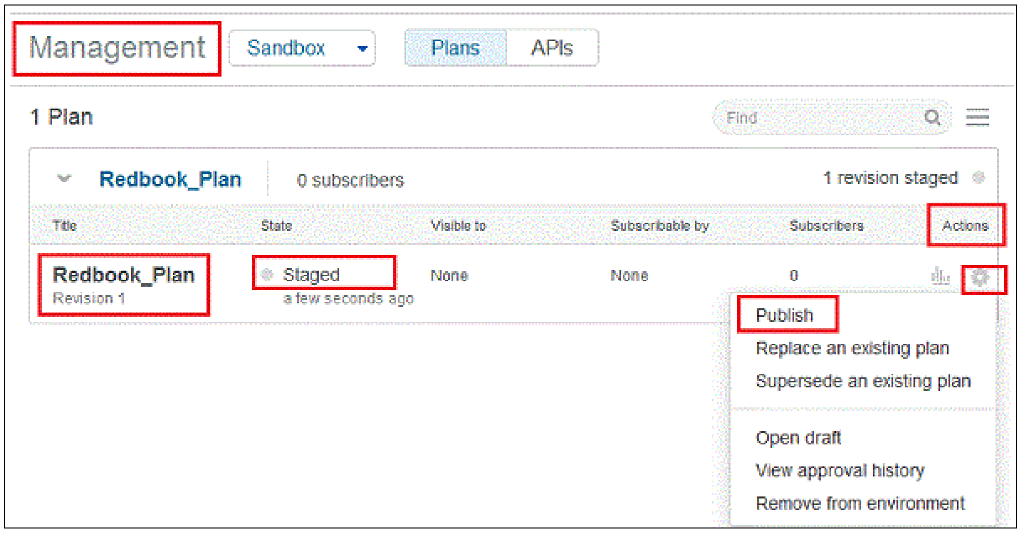
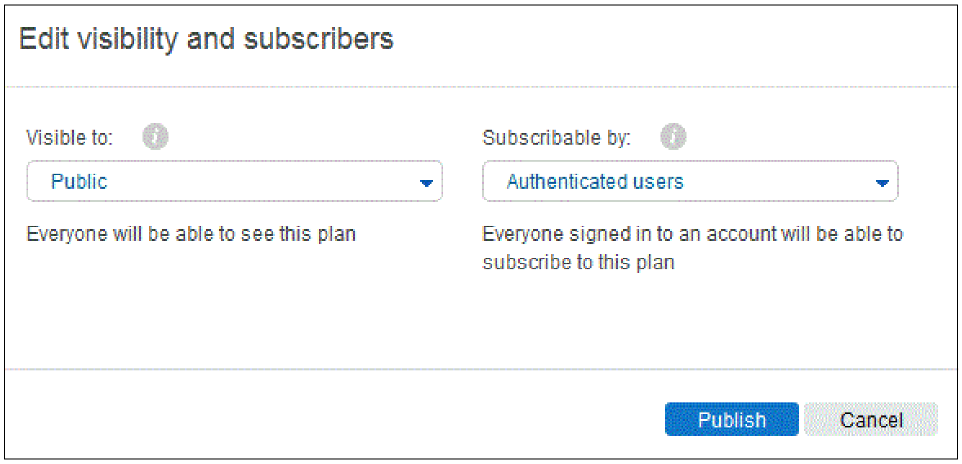


Figure 6-75   Publish the plan

13. A new dialog box appears as shown in Figure 6-76 when you click the **Publish** option in the previous step. The dialog box offers you the option to specify the visibility of the plan and the subscribers of this plan. For this scenario, we use default options.


Edit visibility and subscribers

Visible to: Public

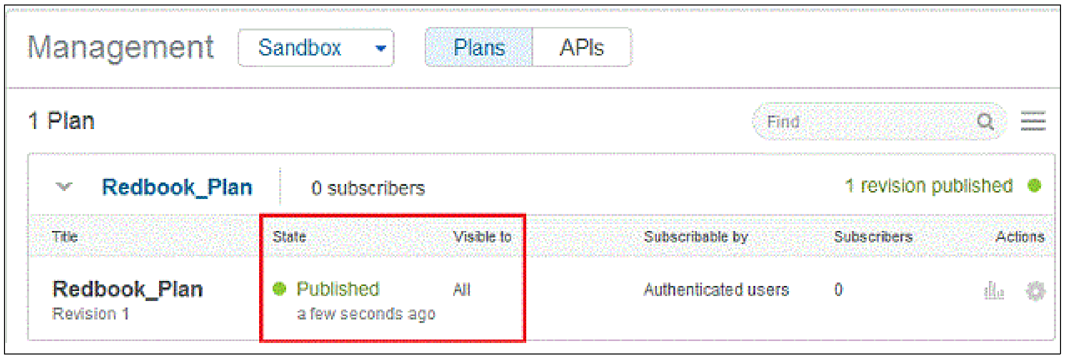
Everyone will be able to see this plan

Subscribable by: Authenticated users

Everyone signed in to an account will be able to subscribe to this plan

Figure 6-76 Visibility and subscribers of the plan

When the plan is published successfully, the status gets updated as shown in Figure 6-77.



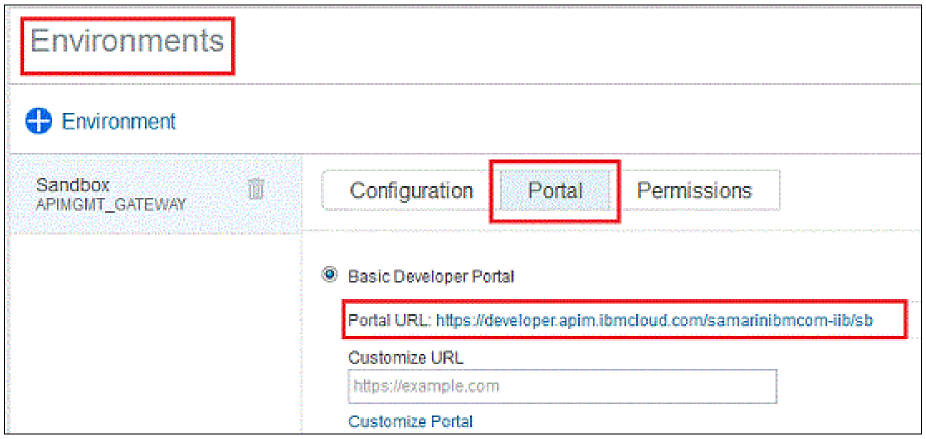
1 Plan

Redbook_Plan 0 subscribers 1 revision published

Title	State	Visible to	Subscribable by	Subscribers	Actions
Redbook_Plan Revision 1	Published a few seconds ago	All	Authenticated users	0	

Figure 6-77 Status of the plan after publish

14. Now go to the **Environments** section from the left-side menu and select the **Portal** tab and click **Portal URL** under Basic Developer Portal, as shown in Figure 6-78.



Basic Developer Portal

Portal URL: https://developer.apim.ibmcloud.com/samarinibmcom-iib/sb

Customize URL

https://example.com

Figure 6-78 Developer portal URL in Environments section

15. Sign in to the API Manager Developer portal as shown in Figure 6-79.

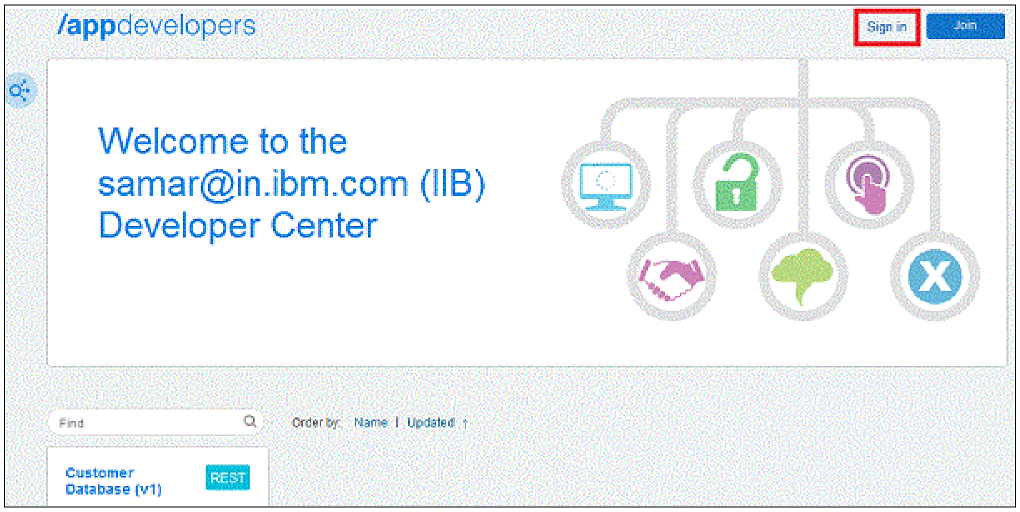
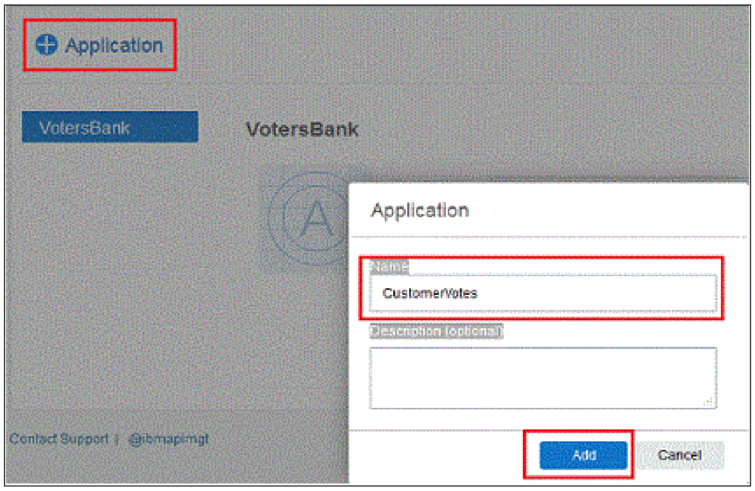


Figure 6-79 Sign in to the developer portal

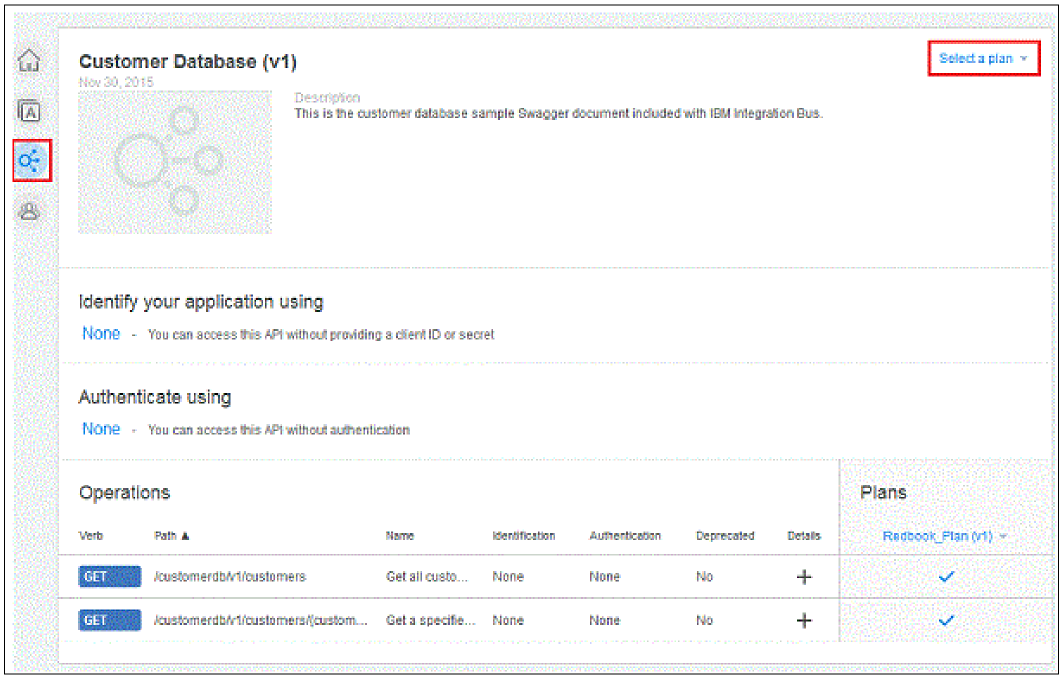
16. Go to the **Application** section from the left-side menu and click the **+** sign to add a new application, as shown in Figure 6-80.



Name
CustomerVotes
Description (optional)

Figure 6-80 Add a new application

17. After adding the application, click the **APIs** tab on the left-side menu and click the **Select a plan** drop-down list as shown in Figure 6-81.



Customer Database (v1)
Nov 30, 2015

Description
This is the customer database sample Swagger document included with IBM Integration Bus.

Identify your application using
None - You can access this API without providing a client ID or secret

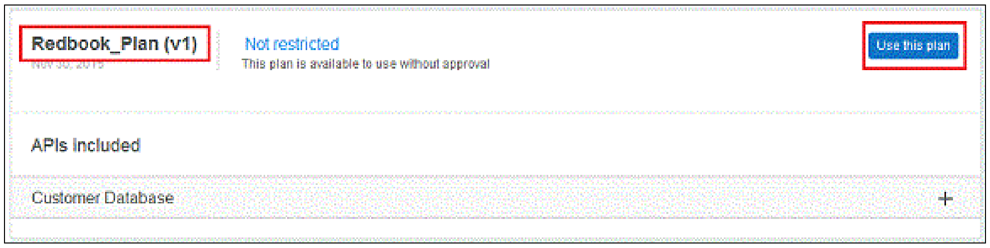
Authenticate using
None - You can access this API without authentication

Operations

Verb	Path ▲	Name	Identification	Authentication	Deprecated	Details
GET	/customerdb/v1/customers	Get all custo...	None	None	No	+
GET	/customerdb/v1/customers/{custom...	Get a specifie...	None	None	No	+

Figure 6-81 Select the plan to be used with the application

18. A list of available plans is displayed. As shown in Figure 6-82, select the plan that you want to apply and click **Use this plan**.



Redbook_Plan (v1)
Nov 30, 2015

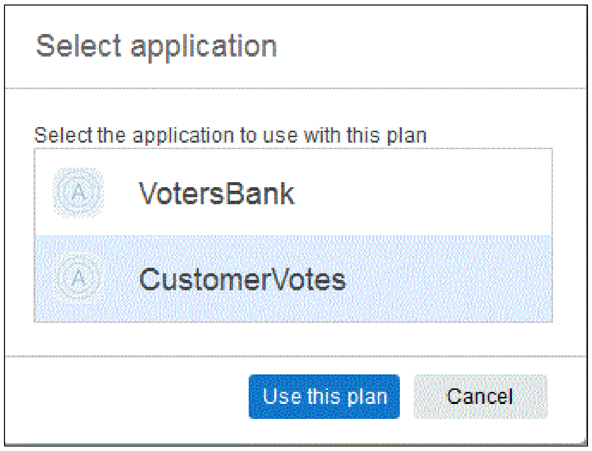
Not restricted
This plan is available to use without approval

APIs included

Customer Database

Figure 6-82 Activate the plan

19. Select the application that you want to associate this plan with as shown in Figure 6-83.



Select application

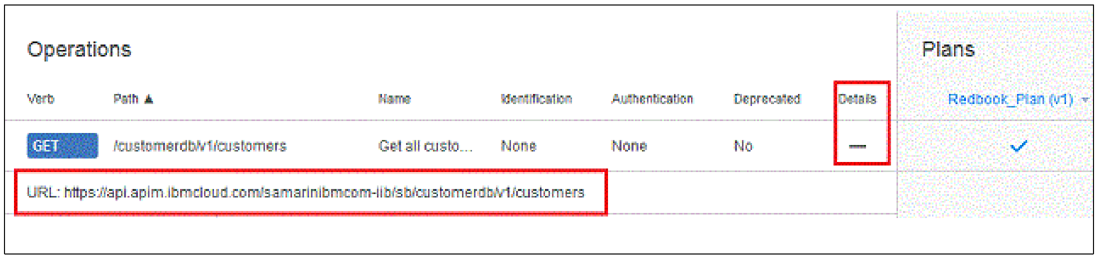
Select the application to use with this plan

VotersBank

CustomerVotes

Figure 6-83 Associate the plan with the application

20. Expand **Operations** by clicking the sign under the **Details** column to get the URL generated by API Management service, as shown in Figure 6-84. This URL can now be used in user applications deployed in Bluemix.



Operations

Verb	Path ▲	Name	Identification	Authentication	Deprecated	Details
GET	/customerdb/v1/customers	Get all custo...	None	None	No	—

URL: https://api.apim.ibmcloud.com/samarinibmcom-lib/sb/customerdb/v1/customers

Plans

Redbook_Plan (v1)
✓

Figure 6-84 URL generated by API Management service

21. Save this URL in Figure 6-84 as *<API Gateway URL for votes by cuisine service>*.

6.11.6 Update and redeploy the web application

The web application allows a user to cast their vote, and view the leader board and votes by cuisine. This application interacts with the API server to record a user's vote:

1. Use the code downloaded in 6.8.3, "Deploy the API server" on page 125.
2. Go to the `sample-app` folder.
3. Use a text editor of your choice to edit *sample-app/connection.properties*.

   Table 6-14 on page 181 contains the parameters as JSON. Replace with the value that matches your configurations.

Table 6-14 Parameters for the Sample-App

Attribute	Default	New value	Purpose
baseUrl	<>	*https://<API Server Host Name>/v1*	Base URL for the Rest API
serviceUrl	<>	*<API Gateway URL for votes by cuisine service>*	Used in subscenario 4 to store API Gateway URL for votes by cuisine

4. Validate the contents of the file and save it.
5. Change the host name to a unique name. Replace aaa with three unique characters of your choice.

Table 6-15 Parameters table

Attribute	Default	New Value	Purpose
host	aaa-redbook-voting-sample-app	**xxx**-redbook-voting-sample-app	Unique host name for web application server

6. Open a command prompt on your workstation.
7. Follow the instructions provided and log in to your Bluemix account by using the **`cf login`** command:

   `https://www.ng.bluemix.net/docs/cli/reference/cfcommands/index.html`

8. On the command prompt, change the current directory to the `voting-service` folder.
9. Enter the **`cf push`** command in the command window. This uploads the application to Bluemix and deploys it.
10. If another application is using the same host name that you have chosen, the deployment might fail. In that instance, change the host as in step 6 and try step 11.
11. Open the following URL:

    `https://console.ng.bluemix.net`

12. Log in using your IBM user ID and password.

13. Click **DASHBOARD**.

An application is displayed on the dashboard. See Figure 6-85.

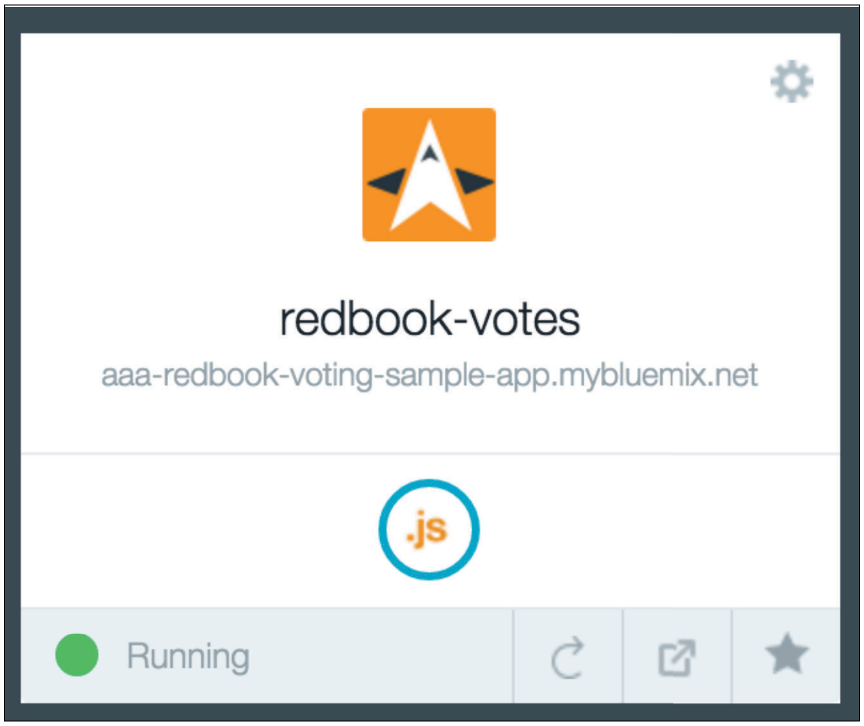


Figure 6-85 API server application deployed and running on Bluemix

14. The web application displays as shown in Figure 6-85.
15. Click **redbook-votes**.
16. Click **Overview**.
17. Write down the URL next to **Routes** as <*Web application Host Name*>. It will be something like http://aaa-redbook-voting-sample-app.mybluemix.net.

6.11.7 Testing the application

The application can be tested by using the following steps:

1. Open a new browser window or tab and type in the following URL: <*Web application Host Name>/redbookApp.*
2. Enter name, phone number, email, and select a choice from the drop-down list.
3. Click **Vote**. See Figure 6-86.

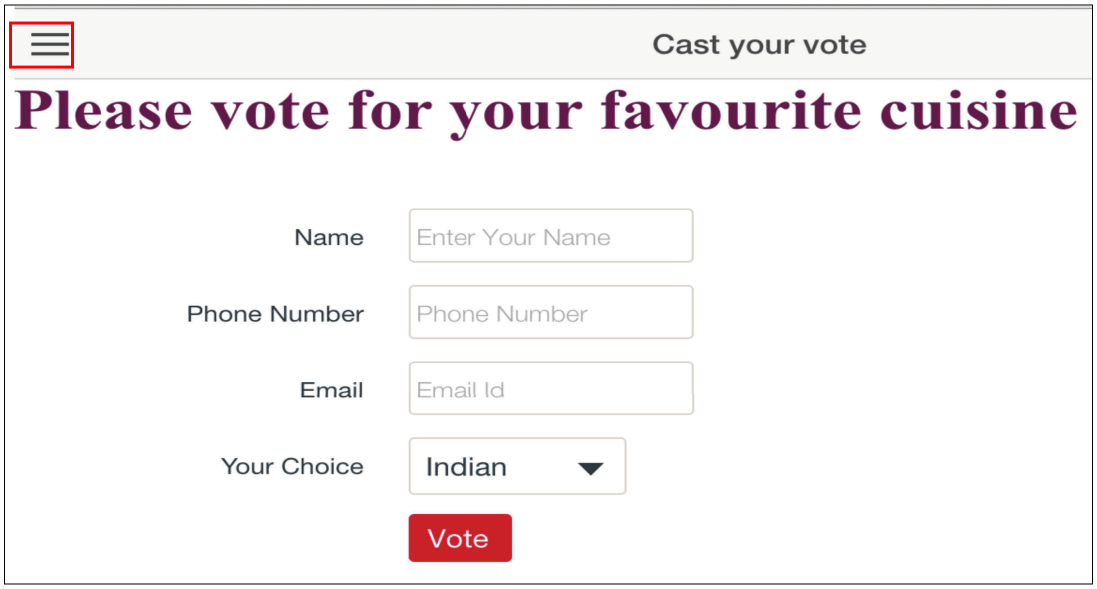


Figure 6-86 Voting application: Cast your vote

The application responds with a message of thanks and displays a time stamp for the vote.

4. Click the upper left corner of the user interface highlighted in red, as shown in Figure 6-86.
5. Click **Votes by Cuisine**. See Figure 6-87. This displays all votes sorted by latest to oldest.

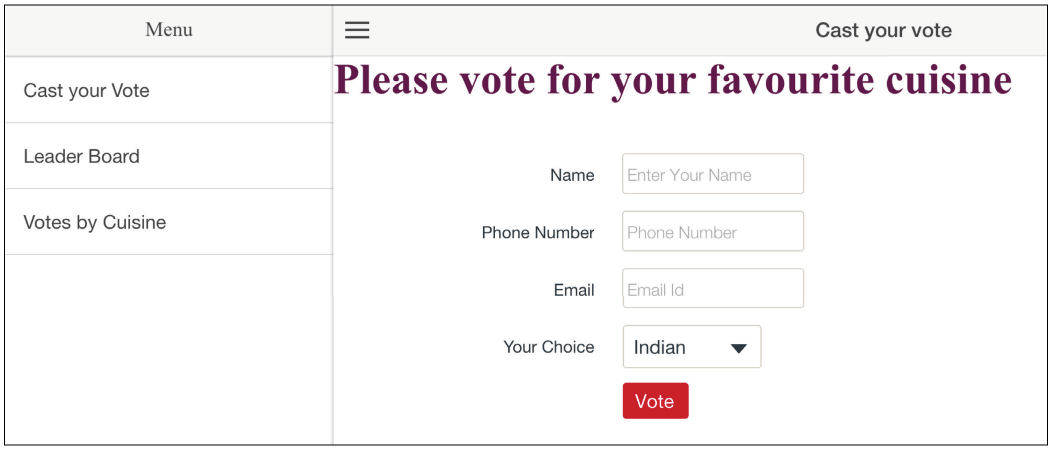
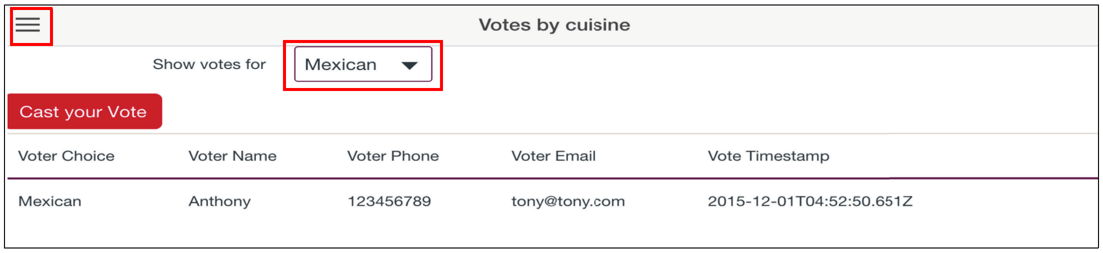


Figure 6-87 Select option from the menu

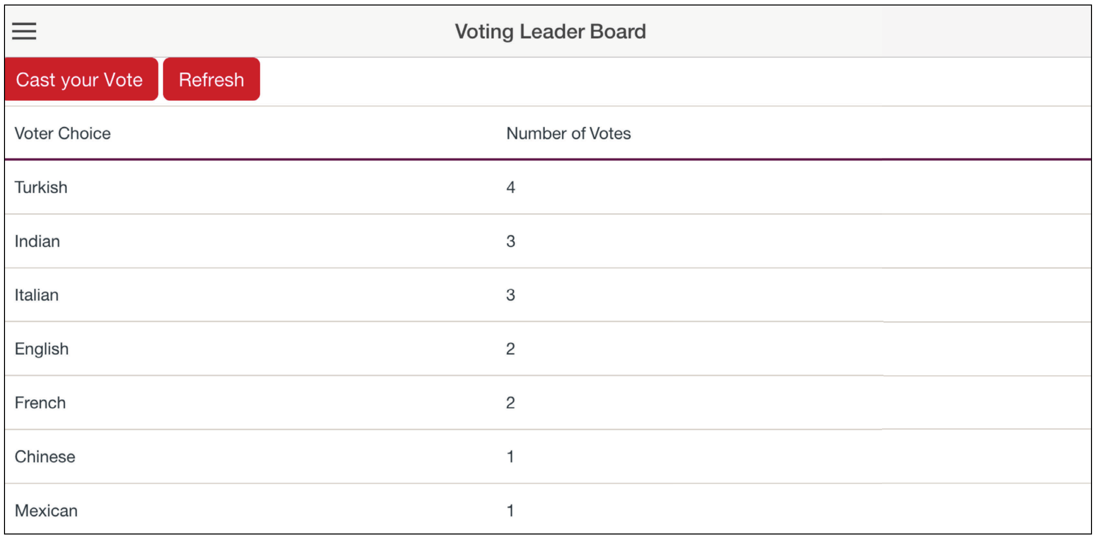
6. Select your cuisine from the drop-down list as shown in Figure 6-88.



Voter Choice	Voter Name	Voter Phone	Voter Email	Vote Timestamp
Mexican	Anthony	123456789	tony@tony.com	2015-12-01T04:52:50.651Z

Figure 6-88 Votes by cuisine

7. Click the upper-left corner as shown in Figure 6-88.
8. Click **Leader Board**. The window shown in Figure 6-89 is displayed.



Voter Choice	Number of Votes
Turkish	4
Indian	3
Italian	3
English	2
French	2
Chinese	1
Mexican	1

Figure 6-89 Voting Leader Board

9. Try casting a few more votes by clicking **Cast your Vote** and viewing the details change in the application.

6.11.8 Summary of scenario 4

In this scenario, the data was captured from the user and it is processed straight through by the enterprise application. The important point to note is that we made configuration changes to the web application and the API server.

In this subscenario, the data related to an event in a cloud application (casting of a vote) is persisted directly to the corporate IBM MQ server. This data is then processed by the corporate IBM Integration Bus. Additional functionality was provided for the latest results as well as the query about the list of votes by voter choice. An event in the internal system (a new leader board) is made available to a listener in the IBM cloud (the API server). The API server caches the result and publishes it as a REST API.

Figure 6-90 shows the information flow for the processing of a vote.

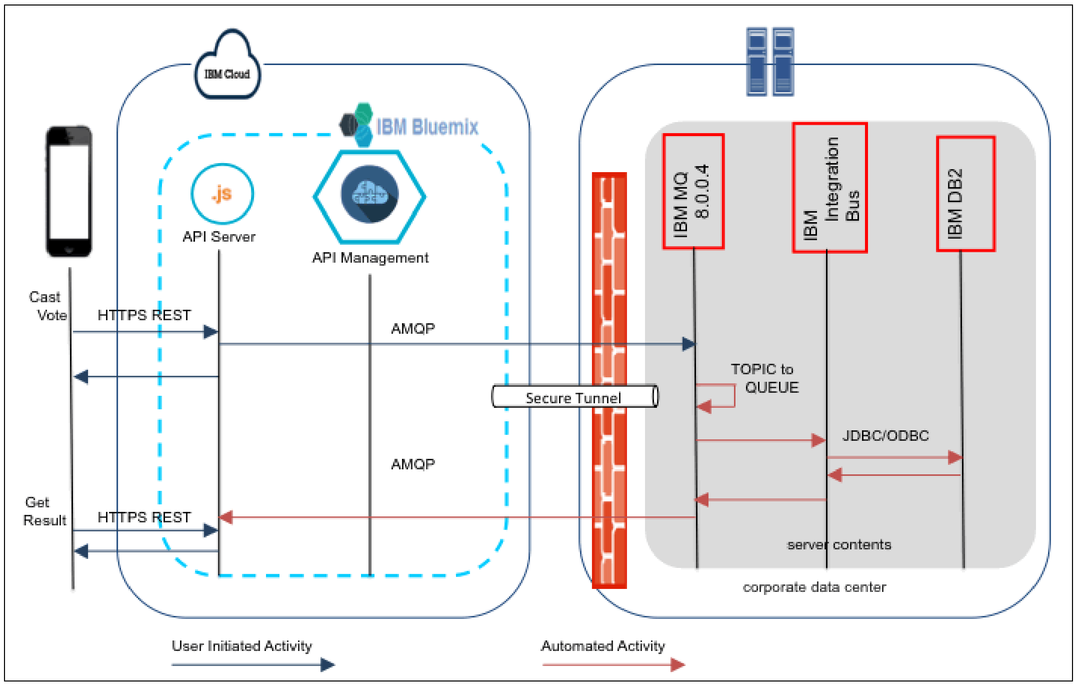


Figure 6-90   Subscenario 4: Flow of information from the user to database and publishing latest tally

As shown in Figure 6-91, the corporate Integration Bus makes available the details of the votes as a REST service. This REST API is connected to the API Management server using the secure tunnel. The User Interface application invokes the REST API running on the API Management Bluemix service so the details of the corporate Integration Bus are never exposed to the users.

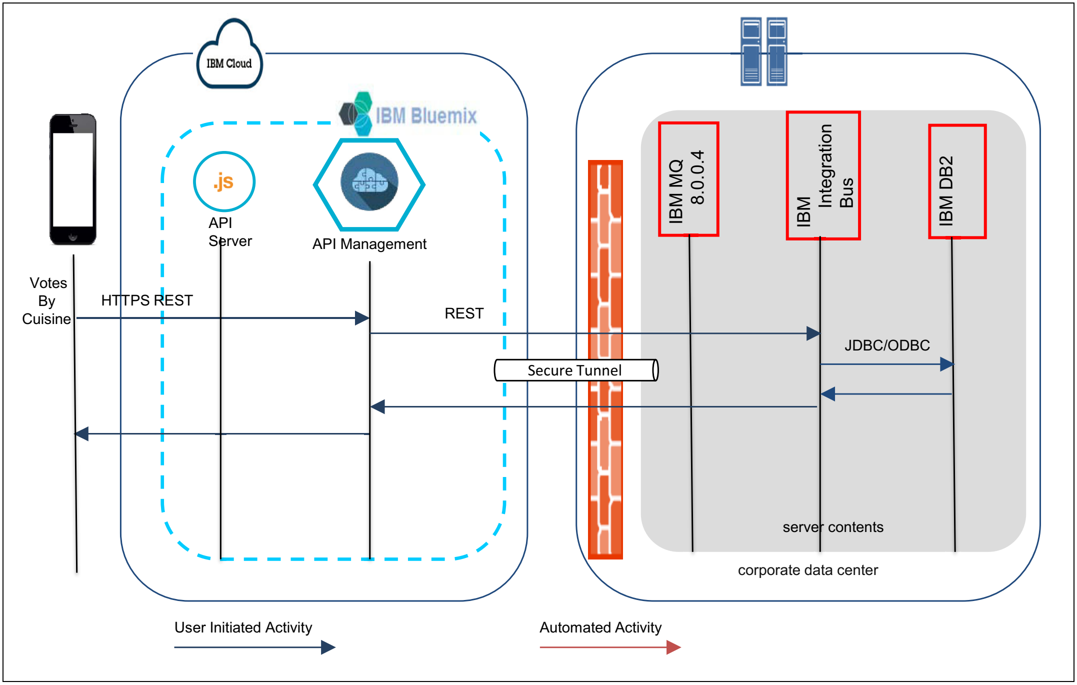


Figure 6-91   Subscenario 4: Publishing votes by cuisine

6.12 Summary

In this chapter we covered the following topics:

- We started with a born-on-the-cloud application built by using the microservices architecture. This application was developed quickly by using the IBM Bluemix innovation platform. It lived up to its promises during initial testing and was ready to be deployed into the production environment.
- Subscenario 1: We deployed the web application running on the cloud and securely connecting to a database in the corporate data center. This simulated the first stage for transformation.
- Subscenario 2: We improved the performance and scalability of the solution using Worker Offload Pattern. This was achieved by using an IBM Bluemix MQ Light service along with a Message Processor application. This simulated the second stage of transformation for the solution.
- Subscenario 3: The application components from the previous subscenario were reused. We reconfigured the solution to use an enterprise IBM MQ instead of IBM MQ Light service on IBM Bluemix. The Message Processor used in subscenario 2 ran as an in-house Node.js application. This represented the third stage of the transformation of the solution.
- Subscenario 4: This subscenario extends the solution to use the enterprise service bus to support existing capabilities as well as provide additional functionality. This subscenario represents a hybrid enterprise solution.

We transformed a born-on-the cloud application into a hybrid enterprise solution using the capabilities of IBM messaging and IBM Bluemix offerings. This transformation was done in stages. The user experience was not compromised at any stage.

The sample code used for the subscenarios was developed for application messaging and worked with configuration changes on the enterprise messaging servers offering a flexible solution and improved developer productivity. It allowed for development agility along with enterprise quality of service. See Figure 6-92.

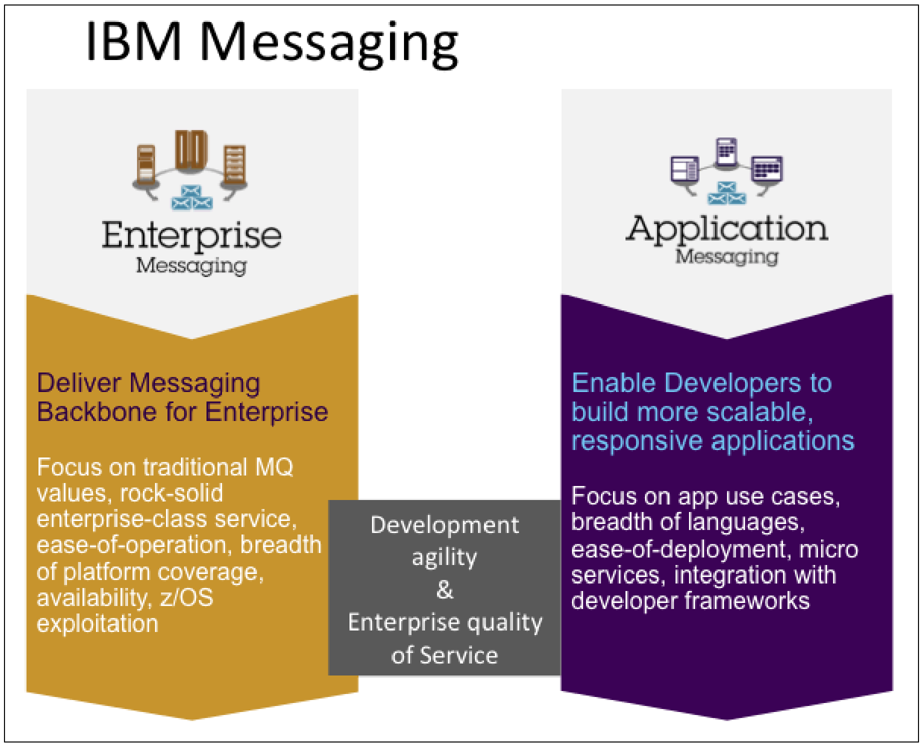

IBM Messaging
Enterprise
Messaging
Deliver Messaging Backbone for Enterprise
Focus on traditional MQ values, rock-solid enterprise-class service, ease-of-operation, breadth of platform coverage, availability, z/OS exploitation
Development agility & Enterprise quality of Service
Application
Messaging
Enable Developers to build more scalable, responsive applications
Focus on app use cases, breadth of languages, ease-of-deployment, micro services, integration with developer frameworks

Figure 6-92 Development agility and enterprise quality of service

7 Synchronizing data from Salesforce to a remote enterprise system

This chapter describes how to synchronize data one way from Salesforce to a remote enterprise system. It explains in details integration between Salesforce and an on-premises system of record through IBM Bluemix. It focuses on the following concepts:

- Remote invocation from Salesforce
- Data and services integration and application programming interface (API) composition with StrongLoop
- Secure connectivity between cloud-based application and on-premises system of record

Recording: You can find the recording of the scenario described in this chapter at the following link:

`https://youtu.be/WwpzHmmdfnA`

This chapter has the following sections:

7.1 Scenario overview

In some situations, Salesforce requires integration with remote backend systems that are running in a private data center. In this chapter, we look at a common scenario where Salesforce and a third-party system (for example, database, ERP) that are both managing account data need to be synchronized.

Salesforce will be the master system for accounts and will need to trigger events to update the backend system accordingly.

As a backend system, we use a NoSQL database (MongoDB) that will be running on-premises. In order to access the database and establish a secured tunnel, we leverage the IBM Secure Gateway on Bluemix.

Additionally, we use StrongLoop and its LoopBack framework to develop a RESTful interface and get it connected to the database. The Representational State Transfer (REST) API is exposed to Salesforce to trigger an action.

Figure 7-1 shows the overall architecture of the solution.

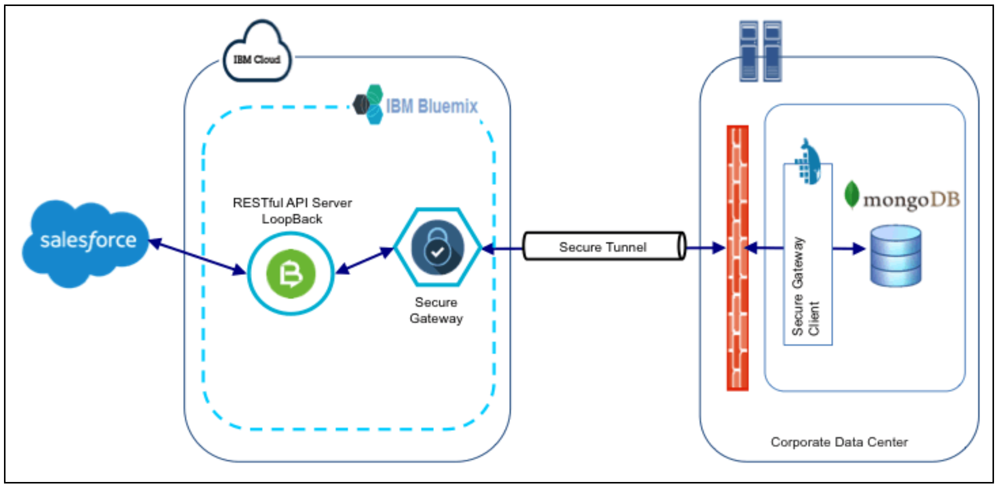


Figure 7-1 Solution architecture diagram

The end-to-end flow of this scenario is described below:

1. A new account is created or updated in Salesforce.
2. An outbound message is fired from Salesforce and does a callout to the REST service.
3. A LoopBack application on Bluemix handles the request from Salesforce and connects to MongoDB through IBM Secure Gateway.
4. The LoopBack application creates a new account document or updates an existing one in the database.
5. The LoopBack application returns an acknowledgment message to Salesforce to complete the transaction.

7.2 Setting up the database

In this scenario, we are using MongoDB to simulate our system of record. MongoDB is a cross-platform NoSQL database that is highly performant and scalable. Instead of storing your data in tables and rows in MongoDB, you store JSON-like documents with dynamic schemas, which provide greater flexibility compared to a traditional relational database.

Because we want to simulate an on-premises system of record, you install and configure MongoDB locally rather than using an instance in the cloud.

Install MongoDB

To install MongoDB, ensure that you download the binary files for the correct platform and follow the corresponding instructions:

1. Download the latest release of MongoDB. The binary files for the current version of MongoDB are available at the following site:

   https://www.mongodb.org/downloads#production

2. Go through the installation process and follow the instructions for the correct platform. Refer to the following link for further details:

   https://docs.mongodb.org/master/installation

Note: Make sure to run MongoDB by starting the **mongodb** process as specified in the installation instructions.

Configure MongoDB

The configuration of MongoDB for this scenario is fairly simple because you just need to create a new database that is used to store the account data. You use the mongo shell to create the database.

1. Start the mongo shell as shown in Example 7-1.

   Example 7-1 Start mongo shell command

   ```
   $ mongo
   ```

   For more information about the mongo shell, see the following link:

   https://docs.mongodb.org/manual/tutorial/getting-started-with-the-mongo-shell

2. Create a new database as shown in Example 7-2.

   Example 7-2 Create database command

   ```
   > use accountdb
   ```

   Note: This command creates a new database if one does not already exist. Otherwise, it returns the existing database.

You now have a database up and running that is used as a system of record in this scenario.

7.3 Exposing the database through IBM Secure Gateway

Cloud-based applications often need access to backend enterprise data or services hosted on-premises. In this scenario, you access your database instance running locally from a LoopBack application running in the cloud. The Secure Gateway service on Bluemix allows you to establish a secured tunnel between your Bluemix organization and your local network, allowing applications on Bluemix to access your database.

In this section, you configure a new Secure Gateway on Bluemix and set up the gateway client locally to enable communication between Bluemix and your local database.

Note: Ensure that you have a Bluemix account to be able to complete the rest of this tutorial. To sign up for a Bluemix account, go to the following URL:

`https://console.ng.bluemix.net/registration`

7.3.1 Configuring a Secure Gateway

In this section, you create an instance of the Secure Gateway service in Bluemix. You configure a new gateway as well as a new destination in that gateway. The destination is the MongoDB instance running locally.

Note: For more information about how to configure the Secure Gateway service on Bluemix, see the following URL:

`https://www.ng.bluemix.net/docs/services/SecureGateway/sg_022.html#sg_009`

Create a Secure Gateway instance

Once logged in to Bluemix, you have access to the service catalog and are able to create a new Secure Gateway instance:

1. In the service catalog, click **Secure Gateway**.

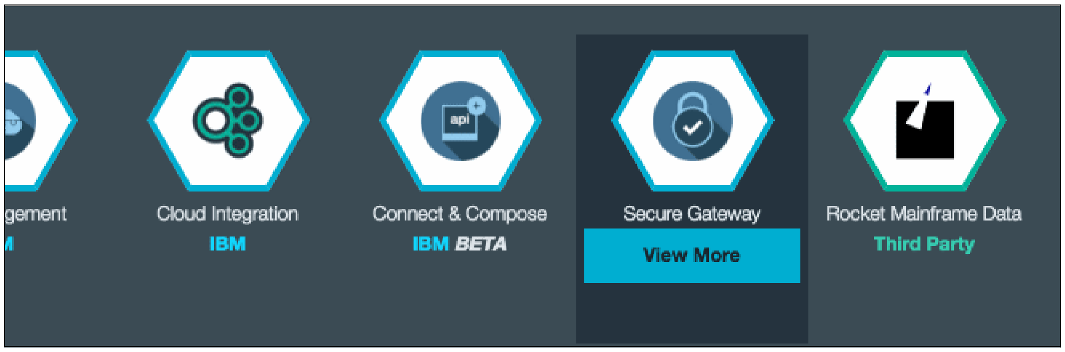
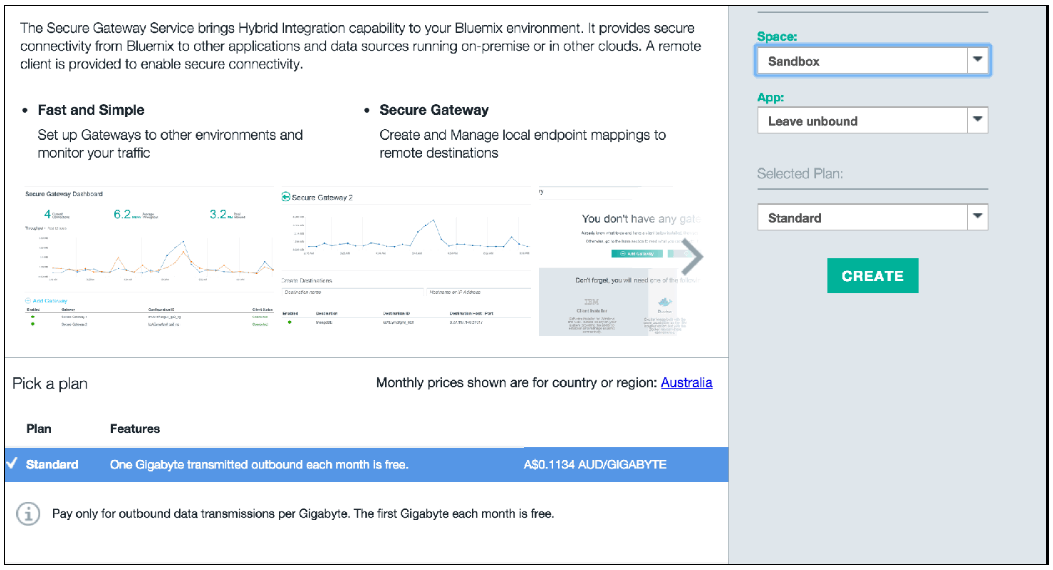


Figure 7-2 Service catalog

2. Select a space and click **Create**, as shown in Figure 7-3.



The Secure Gateway Service brings Hybrid Integration capability to your Bluemix environment. It provides secure connectivity from Bluemix to other applications and data sources running on-premise or in other clouds. A remote client is provided to enable secure connectivity.

- **Fast and Simple**
  Set up Gateways to other environments and monitor your traffic
- **Secure Gateway**
  Create and Manage local endpoint mappings to remote destinations

Pick a plan — Monthly prices shown are for country or region: Australia

Plan	Features	
✓ Standard	One Gigabyte transmitted outbound each month is free.	A$0.1134 AUD/GIGABYTE

Pay only for outbound data transmissions per Gigabyte. The first Gigabyte each month is free.

Space: Sandbox
App: Leave unbound
Selected Plan: Standard

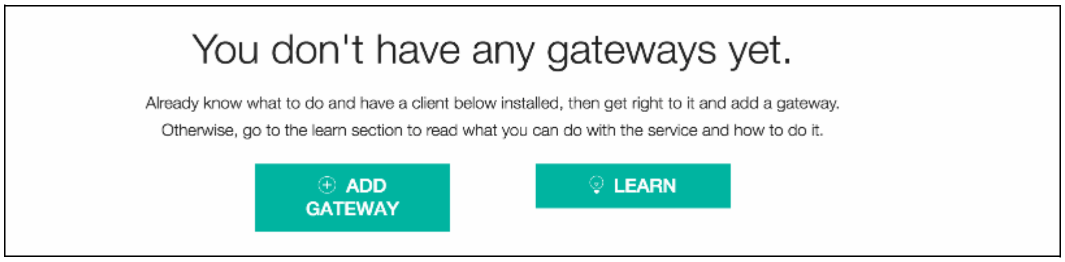
Figure 7-3 Service Gateway creation page

You have now created a new instance of the Secure Gateway service.

Add a gateway and destination

After creating a new instance of the Secure Gateway service, you add a new gateway and configure the destination:

1. In the Secure Gateway service page, click **Add Gateway.**

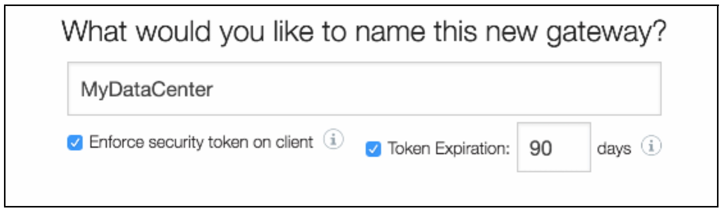


You don't have any gateways yet.

Already know what to do and have a client below installed, then get right to it and add a gateway.
Otherwise, go to the learn section to read what you can do with the service and how to do it.

Figure 7-4 Secure Gateway service page

2. In the Add Gateway page, set the name to **MyDataCenter**.



What would you like to name this new gateway?

MyDataCenter

✓ Enforce security token on client ✓ Token Expiration: 90 days

Figure 7-5 Add Gateway page

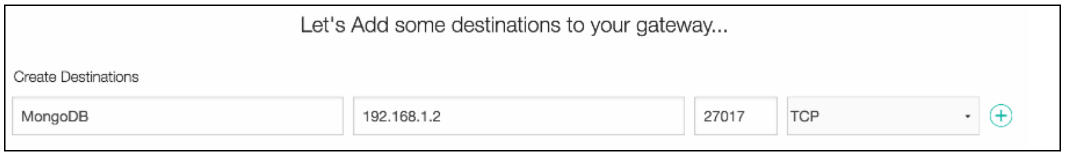
3. Click **Add Destinations**.

4. In the Add Destinations page, add a destination with the settings shown in Table 7-1.

Table 7-1 Secure Gateway destination settings

Property	Value
Destination name	MongoDB
Hostname or IP Address	*<IP address of database host>*
Port	27017
Transport	*TCP*

5. After entering the destination parameters, press the **plus sign (+)** on the right.

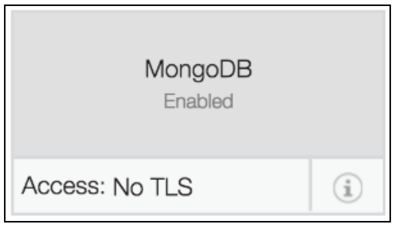

Let's Add some destinations to your gateway...

Create Destinations

MongoDB | 192.168.1.2 | 27017 | TCP

Figure 7-6 Add Destinations page

6. A destination named MongoDB is now listed.

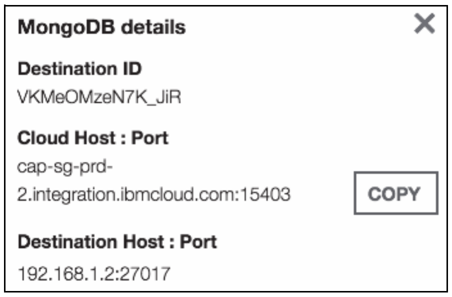

MongoDB

Enabled

Access: No TLS

Figure 7-7 New destination created

7. Click the destination **information icon** to display the configuration.


MongoDB details

Destination ID

VKMeOMzeN7K_JiR

Cloud Host : Port

cap-sg-prd-2.integration.ibmcloud.com:15403

Destination Host : Port

192.168.1.2:27017

Figure 7-8 Destination configuration

The Secure Gateway has automatically generated a **Cloud Host:Port** value. This host name and port can be used by an application to access the destination remotely. Make a note of this value because you will use it later in this tutorial.

You have now configured a gateway and destination for accessing the database.

Note: Keep the Add Gateway page open because there is still one more step to complete. We go through this step in 7.3.2, “Running the gateway client” on page 195.

7.3.2 Running the gateway client

In order to establish a secure tunnel between Bluemix and your database, you need to set up a client where the database is hosted. The gateway client requires network access to both the gateway running in Bluemix and to the database running on-premises. Typically, this means that the gateway client should be installed on a host that is connected to the Internet and also connected to the same network as the database host.

The gateway client provides native client installation and can also run in a Docker container or in IBM DataPower. In this scenario, we use the Docker container.

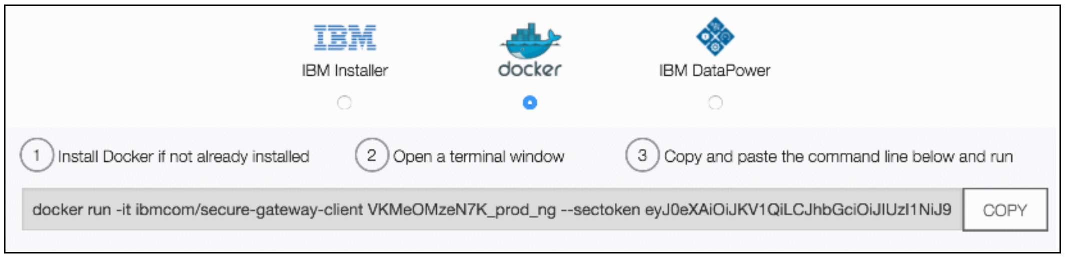
Note: If Docker is not already installed on the database host, follow the installation guide for your target operating system before proceeding:

http://docs.docker.com/engine/installation

Start the gateway client

You have to run a Docker command to start the gateway client. To do so, go through the following steps:

1. Back on the Add Gateway page, click **Connect It**.
2. Ensure that Docker is selected and click **COPY**. This is the Docker command that you need to run to start the gateway client.


IBM Installer
docker
IBM DataPower
1 Install Docker if not already installed
2 Open a terminal window
3 Copy and paste the command line below and run
COPY

```
docker run -it ibmcom/secure-gateway-client VKMeOMzeN7K_prod_ng --sectoken eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
```

Figure 7-9 Connect It page

3. Paste the command line in a terminal and run it to start the gateway client as shown in Example 7-3.

Example 7-3 Start gateway client command

```
$ docker run -it ibmcom/secure-gateway-client VKMeOMzeN7K_prod_ng --sectoken
eyJOeXAiOiJKV1QiLCJhbGciOiJIU...
```

Note: Depending on your operating system, you can use the Docker quick start terminal to run this command.

4. After the gateway client is started and successfully connected to the gateway on Bluemix, you see a message as shown in Example 7-4.

Example 7-4 Gateway client log

```
[INFO] (Client PID 1) The Secure Gateway tunnel is connected
```

Configure the gateway client

By default, access to your backend on-premises resources is denied through the Secure Gateway client. Consequently, you need to add the database host name/ip and port number to the access control list of the gateway client.

1. Press Enter to access the interactive command line and run the following command.

Example 7-5 ACL allow command

```
cli> acl allow 192.168.1.2:27017
```

> **Note:** The host name/ip address and port number should be the same as the one used in Table 7-1 on page 194.

2. You can check if the access control list has been updated successfully by running the **S** command.

Example 7-6 Access control list status

```
cli> S
-------------------------------------------------------
    -- Secure Gateway client Access Control List --

 hostname                     port              value
 192.168.1.2                 27017              Allow
-------------------------------------------------------
```

The gateway client is now configured and running. The shell used to run the client is now attached to the container, which displays when connections are open and closed.

7.4 Compose API with StrongLoop

Now that you can access the on-premises database through the Secure Gateway, you need to create an application that exposes a RESTful interface to Salesforce and connects to the database. To do so, we use the StrongLoop platform.

StrongLoop is built on top of the open source LoopBack framework and allows you to develop REST APIs in Node as well as getting them connected to your data. In this scenario, it will be the “glue” between the MongoDB database and Salesforce.

7.4.1 Getting started with StrongLoop

In this section, you go through some mandatory steps to get the StrongLoop platform up and running.

Install Node.js

StrongLoop requires Node.js to be installed. Ensure that you download the installer for your target operating system:

1. Download the latest release of Node.js available at:

   `https://nodejs.org/en/download`

2. Run the installer that you downloaded to install Node.js.

Install StrongLoop

StrongLoop can be installed via the Node Package Manager (npm) by performing the following steps:

1. Execute the following command to install StrongLoop:

   ```
   $ npm install -g strongloop
   ```

2. If you experienced any issues with the installation, see the following link:

   https://docs.strongloop.com/display/SL/Installation+troubleshooting

Now that you installed StrongLoop, you can create an application leveraging the platform.

7.4.2 Creating an application

StrongLoop features the popular open source LoopBack framework, which enables you to quickly compose scalable APIs. The application that is created in this scenario is a LoopBack application.

Additionally, StrongLoop provides an `slc` command line tool that is used for building and managing your application. This allows you to quickly connect your application to the database and generate a model to interact with it.

Generate project

You use the `slc` command line tool to generate your LoopBack application. Go through the following steps to generate the application:

1. Go to a directory where you want to create the application.
2. Run the following command to generate the application:

   ```
   $ slc loopback
   ```

3. When prompted, name your application and the directory that will contain your project as shown in Example 7-7.

Example 7-7 LoopBack application creation

```
    |       |    .--------------------------.
    |--(o)--|    |  Let's create a LoopBack |
   `---------´   |       application!       |
    ( _´U`_ )    '--------------------------'
    /___A___\
     |  ~  |
   __'.___.'__
 ´   `  |° ´ Y `

? What's the name of your application? sfToMongo
? Enter name of the directory to contain the project: sfToMongo
```

Add a database connector

LoopBack provides database connectors that implement the data exchange logic using database drivers or other client APIs. To connect to your database, you have to install the MongoDB connector. Perform the following steps:

1. Go to the root directory of your application:

   ```
   $ cd sfToMongo
   ```

2. Run the following command to install the connector using npm:

   ```
   $ npm install loopback-connector-mongodb --save
   ```

Create a data source

A LoopBack data source is a unified interface for applications to integrate with backend systems. It provides the ability to plug in various connectors as well as the necessary abstraction to interact with databases and services to decouple the business logic. Perform the following steps to create a data source:

1. Use the data source generator to add a data source to your application:

   ```
   $ slc loopback:datasource
   ```

2. Enter the data source name and select the MongoDB connector as shown in Example 7-8.

   Example 7-8 Data source creation

   ```
   ? Enter the data-source name: mongo_ds
   ? Select the connector for mongo_ds: MongoDB (supported by StrongLoop)
   ```

3. With your favorite text editor, edit */server/datasources.json* (in your sfToMongo project) to add the necessary connection properties to your data source, as shown in Table 7-2.

Table 7-2 Data source properties

Property	Value	Description
host	<cloud host name>	Use the Cloud host name as specified in Figure 7-8 on page 194
port	<cloud port number>	Use the Cloud port number as specified in Figure 7-8 on page 194
database	accountdb	Use the database name specified in 7.2, “Setting up the database” on page 191

We are using the Secure Gateway destination cloud host name and port number because the application will be running on Bluemix. Consequently, we have to access the database through Secure Gateway.

Your data source configuration should be as shown in Example 7-9 on page 199.

Example 7-9 Data source configuration

```
"mongo_ds": {
    "host": "cap-sg-prd-2.integration.ibmcloud.com",
    "port": 15403,
    "database": "accountdb",
    "name": "mongo_ds",
    "connector": "mongodb"
  }
```

Create a model

A LoopBack model represents data in backend systems such as databases. A model interacts with the database via data sources that provide create, retrieve, update, and delete operations. You create a model for the accountdb database that represents an account document:

1. Use the model generator to add a model to your application:

   ```
   $ slc loopback:model
   ```

2. Enter the model name and select the data source created previously. See Example 7-10.

   Example 7-10 Model creation

   ```
   ? Enter the model name: Account
   ? Select the data-source to attach Account to: mongo_ds (mongodb)
   ```

3. Go through the next three questions as shown in Example 7-11.

   Example 7-11 Model selection

   ```
   ? Select model's base class PersistedModel
   ? Expose Account via the REST API? No
   ? Common model or server only? common
   ```

   PersistedModel is the base class for models connected to persistent data sources, such as databases. It provides all the standard create, read, update, and delete operations. Additionally, StrongLoop allows you to automatically generate the REST endpoints for these operations. In this scenario, we create our own endpoint for Salesforce so we do not need to automatically expose our model via the REST API.

4. Add some properties to define your Account object model as shown in Example 7-12.

   Example 7-12 Model properties

   ```
   Let's add some Account properties now.
   ? Property name: account_id
   ? Property type: string
   ? Required? Yes

   Let's add another Account property.
   ? Property name: num
   ? Property type: string
   ? Required? No

   Let's add another Account property.
   ? Property name: name
   ? Property type: string
   ? Required? No

   Let's add another Account property.
   ? Property name: type
   ```

```
? Property type: string
? Required? No

Let's add another Account property.
? Property name: industry
? Property type: string
? Required? No

Let's add another Account property.
? Property name: phone
? Property type: string
? Required? No

Enter an empty property name when done.
```

A representation of the data model has been generated under */common/models/*.

5. By default, an ID is automatically generated by the database but in our scenario, we want to use `account_id`. Open and edit */common/models/account.json* as shown in Example 7-13.

Example 7-13 /common/models/account.json

```
{
  "name": "Account",
  "base": "PersistedModel",
  "idInjection": false,
  "options": {
    "validateUpsert": true
  },
  "properties": {
    "account_id": {
      "type": "string",
      "required": true,
      "id": true
    },
    "num": {
      "type": "string"
    },
    "name": {
      "type": "string"
    },
    "type": {
      "type": "string"
    },
    "industry": {
      "type": "string"
    },
    "phone": {
      "type": "string"
    }
  },
  "validations": [],
  "relations": {},
  "acls": [],
  "methods": {}
}
```

Your application is now created and configured to interact with the MongoDB database running on-premises.

Note: StrongLoop also provides a graphical UI (called Arc) that complements the slc command-line tool. StrongLoop Arc can be used to create graphically data sources and models instead of using the slc command line. See the following link for more information about StrongLoop Arc:

https://strongloop.com/node-js/arc

7.4.3 Adding application logic

In this section, you add some logic to your application to handle events coming from Salesforce and update the database accordingly.

Salesforce emits outbound messages in an XML format (see Example 7-14) to a REST endpoint. Consequently, we have to expose our application through a REST interface and process the XML message sent by Salesforce to create or update an account document in the database.

Example 7-14 Salesforce XML outbound message

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <notifications xmlns="http://soap.sforce.com/2005/09/outbound">
   <OrganizationId>00D90000000w23QEAQ</OrganizationId>
   <ActionId>04k90000000XZMcAAO</ActionId>
   <SessionId xsi:nil="true"/>

<EnterpriseUrl>https://ap1.Salesforce.com/services/Soap/c/35.0/00D90000000w23Q</En
terpriseUrl>

<PartnerUrl>https://ap1.Salesforce.com/services/Soap/u/35.0/00D90000000w23Q</Partn
erUrl>
   <Notification>
    <Id>04l9000001QWWUdAAP</Id>
    <sObject xsi:type="sf:Account"
xmlns:sf="urn:sobject.enterprise.soap.sforce.com">
     <sf:Id>0019000000y2ypEAAQ</sf:Id>
     <sf:AccountNumber>123243</sf:AccountNumber>
     <sf:Industry>Technology</sf:Industry>
     <sf:Name>IBM</sf:Name>
     <sf:Phone>0202020202</sf:Phone>
     <sf:Type>Prospect</sf:Type>
    </sObject>
   </Notification>
  </notifications>
 </soapenv:Body>
</soapenv:Envelope>
```

Add middleware

You need to include extra middleware modules to your application to be able to handle and parse the XML message coming from Salesforce:

1. In your project, edit *server/middleware.json* to register the necessary modules as shown in Example 7-15.

Example 7-15 Modules registration

```
"parse": {
    "body-parser#json": {},
    "body-parser#urlencoded": {"params": { "extended": false }},
    "express-xml-bodyparser": {}
  },
```

2. Go to the root directory of your application and install the corresponding modules using npm, as shown in Example 7-16.

Example 7-16 Modules installation

```
$ npm install body-parser --save

$ npm install express-xml-bodyparser --save
```

Add a custom route

We need to create a REST endpoint that receives messages from Salesforce and implements the logic to create or update an account object in the database. Perform the following steps to add a custom route:

1. Create a new boot script file under `/server/boot` and name it `Salesforce.js`.

Note: When a LoopBack application starts, it runs the scripts in the `/server/boot` directory. By default, LoopBack loads boot scripts in alphabetical order.

2. Edit the file that you just created and add the code as shown in Example 7-17.

Example 7-17 Salesforce.js implementation

```
module.exports = function(app) {
   var router = app.loopback.Router();
   var actModel = app.models.Account;

   //REST endpoint exposed to Salesforce
   router.post('/processSFMsg', function(req, res) {
      var account = parseMessage(req.body);

      // by default return a 'false' Ack to Salesforce
      var resMsg = '<soapenv:Envelope 
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" 
xmlns:out="http://soap.sforce.com/2005/09/outbound"><soapenv:Header/><soapenv:Body
><out:notificationsResponse><out:Ack>false</out:Ack></out:notificationsResponse></
soapenv:Body></soapenv:Envelope>';

      if (account) {
         actModel.upsert(account, function(err, acc) {
         if(!err){
            // return a 'true' Ack if data insert/updated successfully in MongoDB
            resMsg = '<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:out="http://soap.sforce.com/2005/09/outbound"><soapenv:Header/><soapenv:Body
><out:notificationsResponse><out:Ack>true</out:Ack></out:notificationsResponse></s
oapenv:Body></soapenv:Envelope>';
         }
         res.send(resMsg);
         });
      }
      else
      {
         res.send(resMsg);
      }

   });

   // parse the xml and return json object
   parseMessage = function(obj) {
     try {

      // extract attributes from XML
       var accountId =
obj['soapenv:envelope']['soapenv:body'][0].notifications[0].notification[0].sobjec
t[0]['sf:id'] ?
obj['soapenv:envelope']['soapenv:body'][0].notifications[0].notification[0].sobjec
t[0]['sf:id'][0] : '';
       var accountNumber =
obj['soapenv:envelope']['soapenv:body'][0].notifications[0].notification[0].sobjec
t[0]['sf:accountnumber'] ?
obj['soapenv:envelope']['soapenv:body'][0].notifications[0].notification[0].sobjec
t[0]['sf:accountnumber'][0] : '';
       var accountIndustry =
obj['soapenv:envelope']['soapenv:body'][0].notifications[0].notification[0].sobjec
t[0]['sf:industry'] ?
obj['soapenv:envelope']['soapenv:body'][0].notifications[0].notification[0].sobjec
t[0]['sf:industry'][0] : '';
       var accountName =
obj['soapenv:envelope']['soapenv:body'][0].notifications[0].notification[0].sobjec
t[0]['sf:name'] ?
obj['soapenv:envelope']['soapenv:body'][0].notifications[0].notification[0].sobjec
t[0]['sf:name'][0] : '';
       var accountPhone =
obj['soapenv:envelope']['soapenv:body'][0].notifications[0].notification[0].sobjec
t[0]['sf:phone'] ?
obj['soapenv:envelope']['soapenv:body'][0].notifications[0].notification[0].sobjec
t[0]['sf:phone'][0] : '';
       var accountType =
obj['soapenv:envelope']['soapenv:body'][0].notifications[0].notification[0].sobjec
t[0]['sf:type'] ?
obj['soapenv:envelope']['soapenv:body'][0].notifications[0].notification[0].sobjec
t[0]['sf:type'][0] : '';

       return {
          account_id:accountId,
         num:accountNumber,
           name:accountName,
           type:accountType,
           industry:accountIndustry,
           phone:accountPhone
       };

     } catch (e) {
       console.log('Exception parsing Salesforce XML', e);
       return null;
     }
   };

   app.use(router);
}
```

Your application is now configured to receive outbound messages from Salesforce and interact with the database accordingly.

7.4.4 Testing the application

Before deploying your application to Bluemix, you can run it locally and confirm that it correctly connects to MongoDB through the Secure Gateway. To test your application, perform the following steps:

1. To run the application, go to the root directory of your application and run the following command:

   ```
   $ slc run
   ```

   Ensure that MongoDB is running as specified in "Install MongoDB" on page 191 as well as the Secure Gateway client as described in "Start the gateway client" on page 195.

2. Check the gateway client logs because it shows new connections.

Example 7-18 Gateway client logs

```
[2015-11-30 00:03:59.752] [INFO] (Client PID 1) Connection #2 is being established
to 192.168.1.2:27017
[2015-11-30 00:03:59.755] [INFO] (Client PID 1) Connection #3 is being established
to 192.168.1.2:27017
[2015-11-30 00:03:59.758] [INFO] (Client PID 1) Connection #4 is being established
to 192.168.1.2:27017
[2015-11-30 00:03:59.766] [INFO] (Client PID 1) Connection #5 is being established
to 192.168.1.2:27017
[2015-11-30 00:03:59.972] [INFO] (Client PID 1) Connection #6 is being established
to 192.168.1.2:27017
```

3. To test the application, send messages to the REST endpoint using your preferred tool (such as cURL or Postman). Post a sample outbound message to the application using the information specified in Table 7-3.

Table 7-3 Test properties

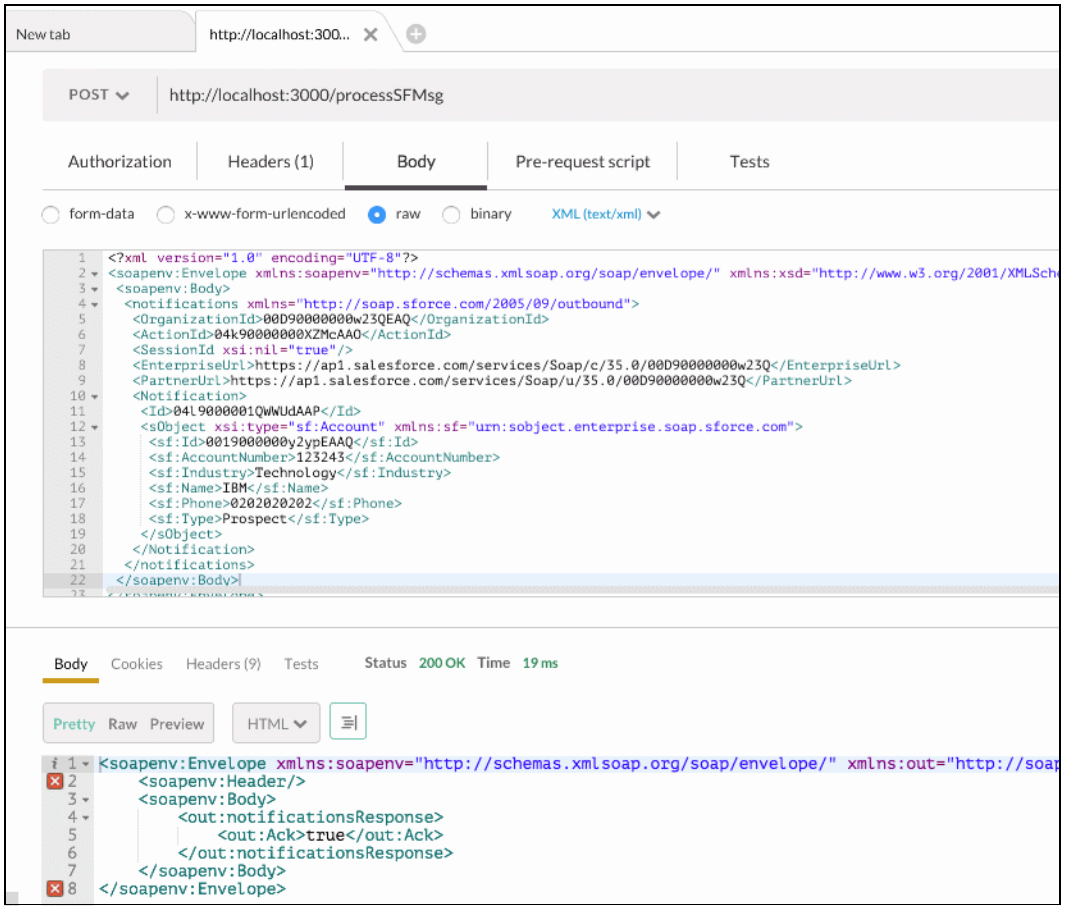
Property	**Value**
Operation	POST
Endpoint	http://localhost:3000/processSFMsg

Property	Value
Header	Content-Type=text/xml
Body	Raw XML as specified in Example 7-14 on page 201

Note: In this scenario, we use Postman to test the application. To download and install Postman, go to the following URL:

https://www.getpostman.com

4. After sending a message, you get an acknowledgment response from the application as shown in Figure 7-10.



POST http://localhost:3000/processSFMsg

Body: raw, XML (text/xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSche
 <soapenv:Body>
  <notifications xmlns="http://soap.sforce.com/2005/09/outbound">
   <OrganizationId>00D90000000w23QEAQ</OrganizationId>
   <ActionId>04k90000000XZMcAAO</ActionId>
   <SessionId xsi:nil="true"/>
   <EnterpriseUrl>https://ap1.salesforce.com/services/Soap/c/35.0/00D90000000w23Q</EnterpriseUrl>
   <PartnerUrl>https://ap1.salesforce.com/services/Soap/u/35.0/00D90000000w23Q</PartnerUrl>
   <Notification>
    <Id>04l9000001QWWUdAAP</Id>
    <sObject xsi:type="sf:Account" xmlns:sf="urn:sobject.enterprise.soap.sforce.com">
     <sf:Id>0019000000y2ypEAAQ</sf:Id>
     <sf:AccountNumber>123243</sf:AccountNumber>
     <sf:Industry>Technology</sf:Industry>
     <sf:Name>IBM</sf:Name>
     <sf:Phone>0202020202</sf:Phone>
     <sf:Type>Prospect</sf:Type>
    </sObject>
   </Notification>
  </notifications>
 </soapenv:Body>
```

Status 200 OK Time 19 ms

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:out="http://soap
    <soapenv:Header/>
    <soapenv:Body>
        <out:notificationsResponse>
            <out:Ack>true</out:Ack>
        </out:notificationsResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

Figure 7-10 Test application locally with Postman

5. Using the mongo shell as described in “Configure MongoDB” on page 191, connect to the database and run a query to verify that a new account has been successfully created.

Example 7-19 Retrieve all accounts

```
> db.Account.find()
{ "_id" : "0019000000y2ypEAAQ", "num" : "123243", "name" : "IBM", "type" :
"Prospect", "industry" : "Technology", "phone" : "0202020202" }
```

So far, your application has been successfully tested locally. You can now deploy it to Bluemix.

7.4.5 Deploying the application

In this section, you move your application to the cloud by deploying your code to Bluemix.

Note: There are various ways to deploy and manage your application on Bluemix. In this scenario, we use the Cloud Foundry (cf) command-line interface. You can download the cf tool at the following URL:

https://github.com/cloudfoundry/cli/releases

For more details about the **cf** commands, see the following link:

https://www.ng.bluemix.net/docs/cli/reference/cfcommands/index.html

1. Log in to your Bluemix account using the cf tool. Run the following command:

   ```
   $ cf login -a <API_URL> -u <username>
   ```

   Where:

   a. *<API_URL>*: The URL of the Cloud Foundry provider, which is Bluemix:

      https://api.ng.bluemix.net

   b. *<username>*: Your email address.

2. When prompted, enter your password and select the organization and space that you want to use.

3. You are now logged in but before deploying the application, you need to add a `manifest.yml` file to the root directory of your application, as shown in Example 7-20.

   Example 7-20 Manifest.yml

   ```
   applications:
   - path: .
     memory: 256M
     command: node server/server.js
     instances: 1
     domain: mybluemix.net
     name: SF2Mongo
     host: SF2Mongo
     disk_quota: 1024M
   ```

   Note: The `manifest.yml` file contains information used by Cloud Foundry to deploy the application to Bluemix. Ensure that you change the host name because it needs to be unique so that its route is also unique.

4. Go to the root directory of your application and use the cf command-line interface to push the application to Bluemix:

   ```
   $ cf push
   ```

   It typically takes 1 - 2 minutes to upload and start the application. You can track the status of the deployment through the cf logs, as shown in Example 7-21 on page 207.

Example 7-21 Deployment logs

```
...
0 of 1 instances running, 1 starting
1 of 1 instances running
App started
OK
```

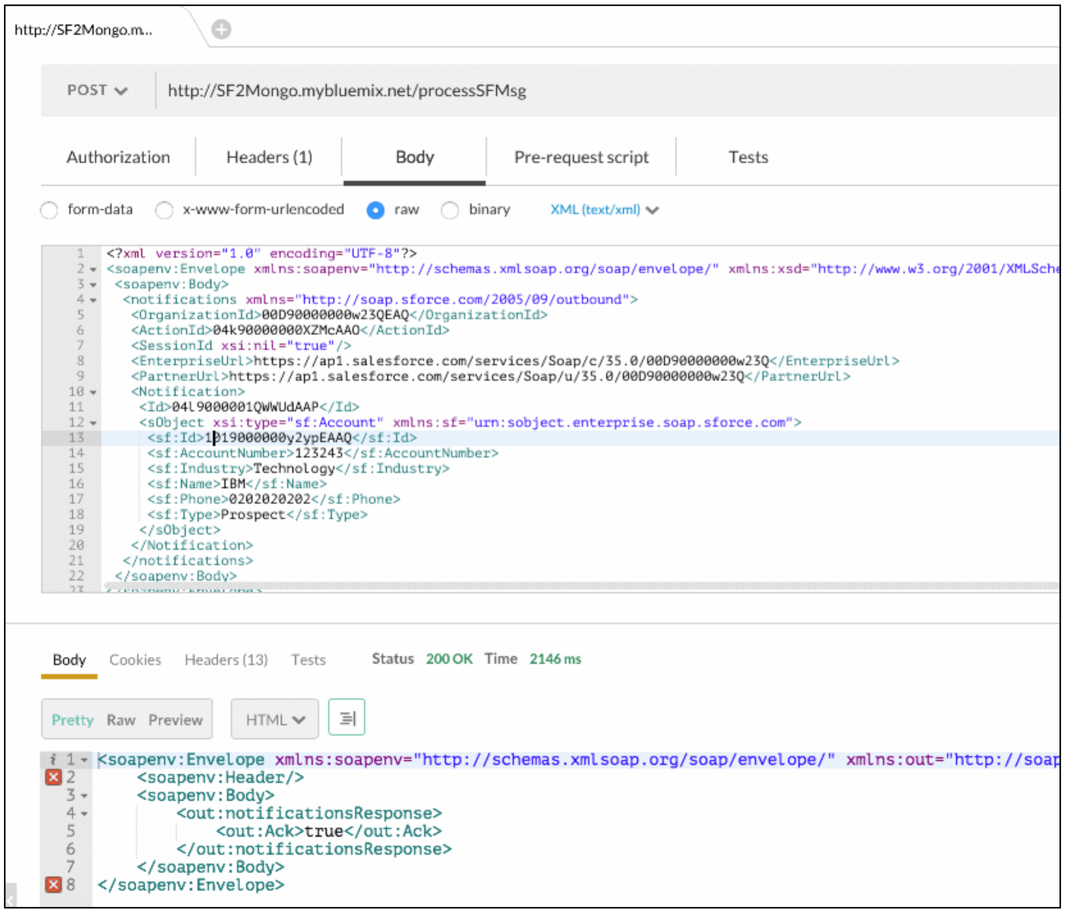
Note: You can also check the status of your application via the Bluemix dashboard in the space you selected.

5. As your application is started, check the gateway client logs as it shows new connections. It means that your application running on Bluemix is now connected to your local database through Secure Gateway.
6. You can now test your application using Postman as shown in "Testing the application" on page 204. However, this time the endpoint is pointed to Bluemix as shown below:

   `http:/SF2Mongo.mybluemix.net/processSFMsg`

Note: The URL will vary depending on the host name that you specified in the `manifest.yml` file.

7. Modify the **sf:Id field** and send a new request. You should receive an acknowledgment message from the application as shown in Figure 7-11.



```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSche
 <soapenv:Body>
  <notifications xmlns="http://soap.sforce.com/2005/09/outbound">
   <OrganizationId>00D90000000w23QEAQ</OrganizationId>
   <ActionId>04k90000000XZMcAAO</ActionId>
   <SessionId xsi:nil="true"/>
   <EnterpriseUrl>https://ap1.salesforce.com/services/Soap/c/35.0/00D90000000w23Q</EnterpriseUrl>
   <PartnerUrl>https://ap1.salesforce.com/services/Soap/u/35.0/00D90000000w23Q</PartnerUrl>
   <Notification>
    <Id>04l9000001QWWUdAAP</Id>
    <sObject xsi:type="sf:Account" xmlns:sf="urn:sobject.enterprise.soap.sforce.com">
     <sf:Id>1019000000y2ypEAAQ</sf:Id>
     <sf:AccountNumber>123243</sf:AccountNumber>
     <sf:Industry>Technology</sf:Industry>
     <sf:Name>IBM</sf:Name>
     <sf:Phone>0202020202</sf:Phone>
     <sf:Type>Prospect</sf:Type>
    </sObject>
   </Notification>
  </notifications>
 </soapenv:Body>
```

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:out="http://soap
    <soapenv:Header/>
    <soapenv:Body>
        <out:notificationsResponse>
            <out:Ack>true</out:Ack>
        </out:notificationsResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

Figure 7-11 Test application on Bluemix with Postman

8. By using the mongo shell as described in “Configure MongoDB” on page 191, connect to the database and run a query to retrieve all documents in accountdb. You will see two documents (one created when the application was tested locally and a new one created from the application running on Bluemix).

Example 7-22 Retrieve all accounts

```
> db.Account.find()
{ "_id" : "0019000000y2ypEAAQ", "num" : "123243", "name" : "IBM", "type" :
"Prospect", "industry" : "Technology", "phone" : "0202020202" }
{ "_id" : "1019000000y2ypEAAQ", "num" : "123243", "name" : "IBM", "type" :
"Prospect", "industry" : "Technology", "phone" : "0202020202" }
```

You successfully deployed and tested your application on Bluemix. You now need to configure Salesforce to emit outbound messages to this application for synchronization with your local MongoDB database.

7.5 Configuring Salesforce

When you implement Salesforce, a typical requirement is to integrate with third-party applications. Depending on the scenario, you can follow various integration patterns as described in the following document:

http://resources.docs.Salesforce.com/latest/latest/en-us/sfdc/pdf/integration_patterns_and_practices.pdf

In this scenario, you use a "Remote Process Invocation" pattern, where Salesforce makes a call to a remote system (your application running on Bluemix) to initiate a transaction on the backend side. Salesforce will be the master system for accounts and will send new or updated account data to the remote application.

From an integration standpoint, you will leverage the outbound messaging and workflow capabilities of Salesforce:

- An outbound message allows you to specify object fields to be sent to a designated remote system.
- Outbound messages are part of the workflow rule functionality in Salesforce.
- Workflow rules watch for specific kinds of field changes and trigger automatic Salesforce actions, such as sending an outbound message.

For more details about these concepts, see the following URL:

https://developer.Salesforce.com/docs/atlas.en-us.api.meta/api/sforce_api_om_outboundmessaging_understanding.htm

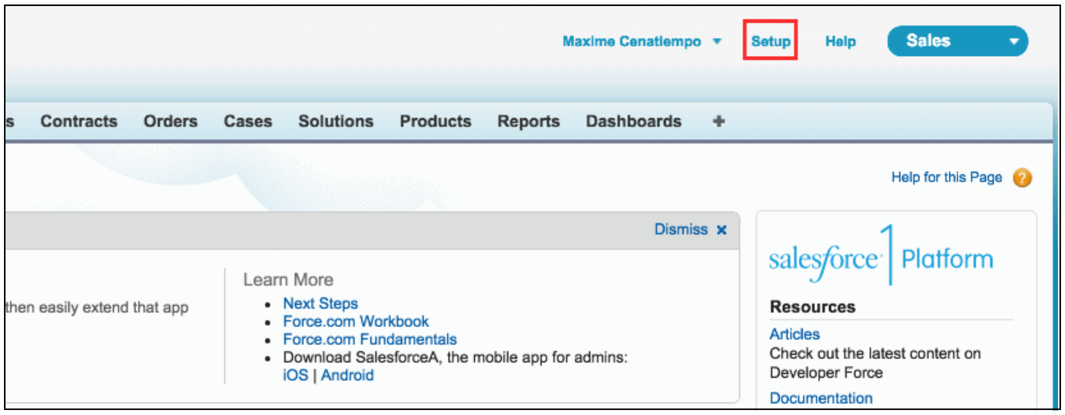
Note: Ensure that you have a Salesforce Developer account to be able to complete the rest of this tutorial. To sign up for an account, go to the following URL:

https://developer.Salesforce.com/signup

Create a workflow rule

You need to create a new workflow rule that triggers an outbound message when an account is created or updated in Salesforce. Follow these steps to create a workflow rule:

1. When logged in to Salesforce, ensure that you are on the Setup page. If not, click **Setup** at the upper right corner.



then easily extend that app

Download SalesforceA, the mobile app for admins:

Check out the latest content on Developer Force

Figure 7-12 Setup link

2. Search for **workflow** in the Quick Find box and click **Workflow Rules**, as shown in Figure 7-13.

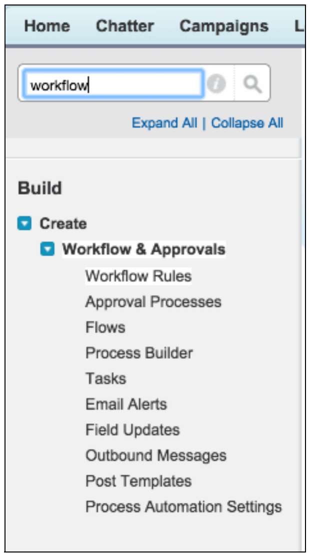
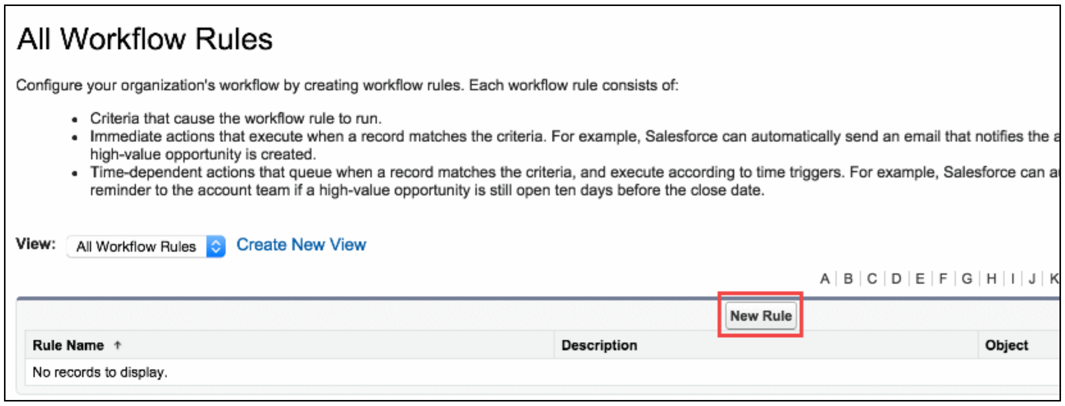


Figure 7-13 Quick Find box

3. Click **New Rule** to create a new workflow rule. Figure 7-14.



All Workflow Rules

Configure your organization's workflow by creating workflow rules. Each workflow rule consists of:

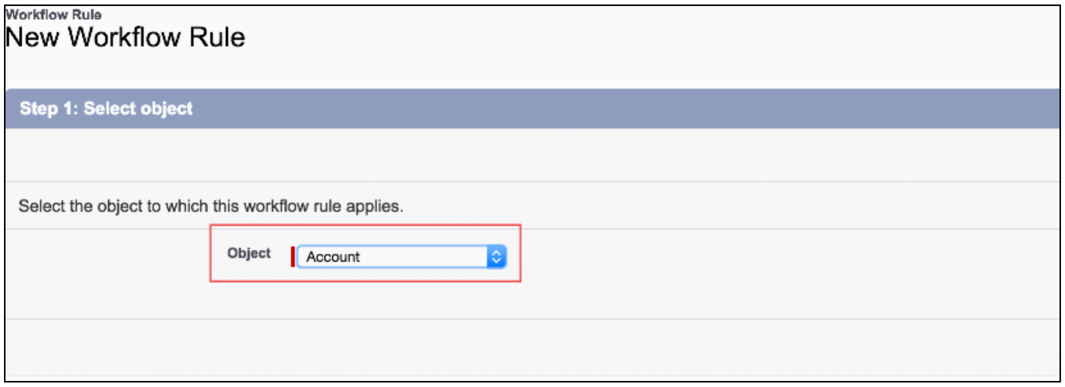
- Criteria that cause the workflow rule to run.
- Immediate actions that execute when a record matches the criteria. For example, Salesforce can automatically send an email that notifies the a high-value opportunity is created.
- Time-dependent actions that queue when a record matches the criteria, and execute according to time triggers. For example, Salesforce can a reminder to the account team if a high-value opportunity is still open ten days before the close date.

View: All Workflow Rules Create New View

Rule Name ↑	Description	Object
No records to display.		

Figure 7-14 New rule creation

4. Select the **Account** object from the list and press **Next**. See Figure 7-15.



Workflow Rule
New Workflow Rule

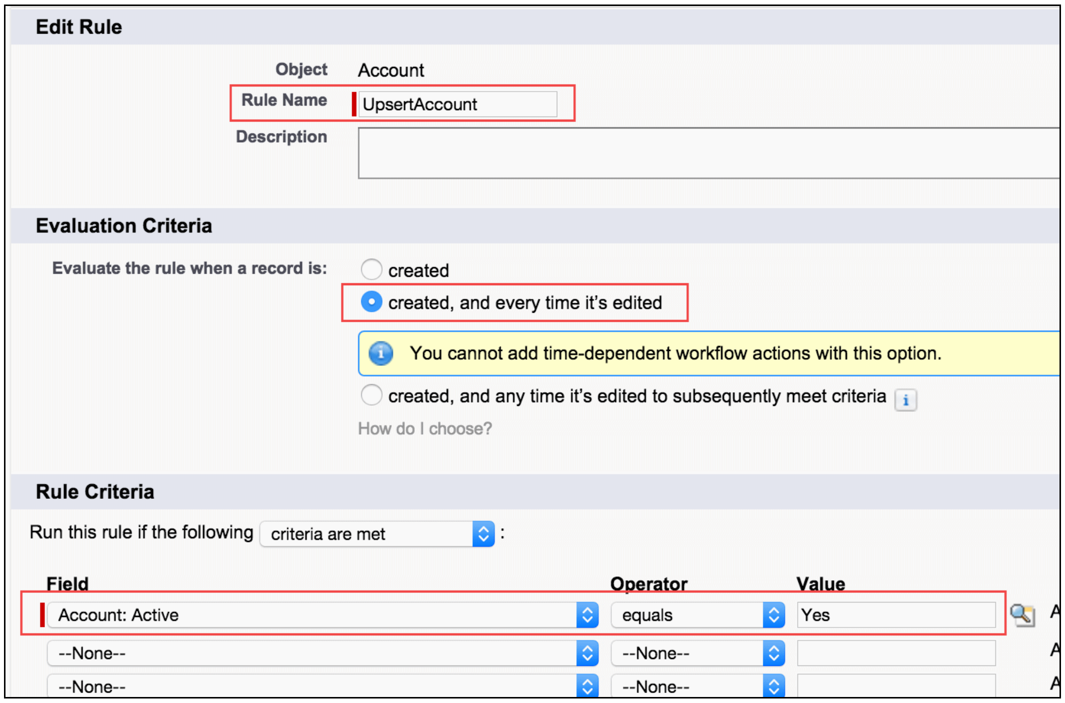
Step 1: Select object

Select the object to which this workflow rule applies.

Object: Account

Figure 7-15 Object selection

5. Enter the required information as shown in Figure 7-16 and press **Save & Next**.


Edit Rule

Object: Account
Rule Name: UpsertAccount
Description:

Evaluation Criteria

Evaluate the rule when a record is:
( ) created
(●) created, and every time it's edited
You cannot add time-dependent workflow actions with this option.
( ) created, and any time it's edited to subsequently meet criteria
How do I choose?

Rule Criteria

Run this rule if the following criteria are met :

Field	Operator	Value
Account: Active	equals	Yes
--None--	--None--	
--None--	--None--	

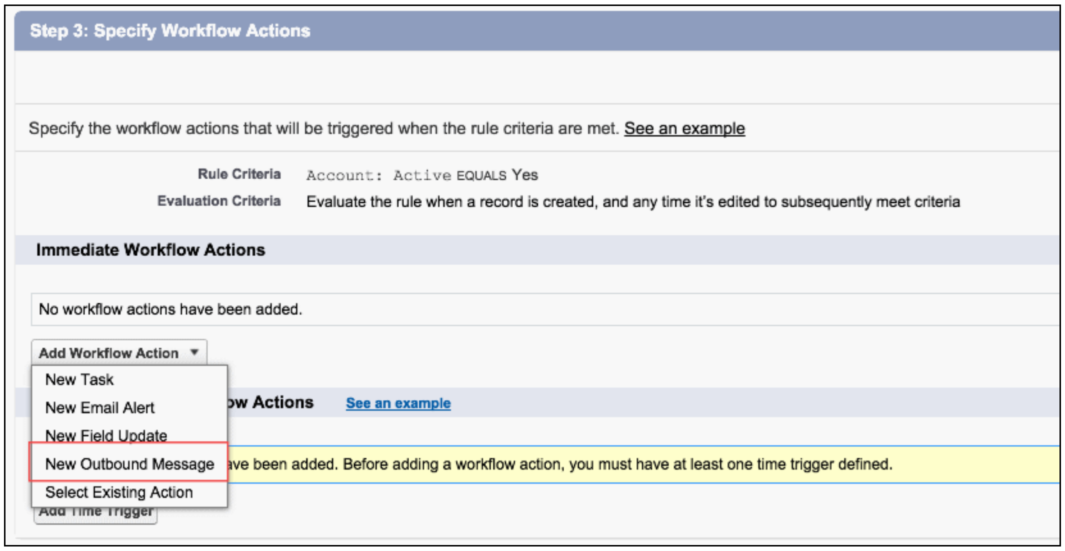
Figure 7-16 Workflow rule configuration

The criteria that you are defining allows you to trigger this workflow rule only if the account created or updated in Salesforce is active. It means that you will only synchronize active accounts from Salesforce to your MongoDB database.

Create an outbound message

You need to create an outbound message containing the fields that you want to send to the database through the application running on Bluemix. This outbound message is triggered by the workflow rule previously created. Follow these steps to create an outbound message:

1. Click **Add Workflow Action** and select **New Outbound Message** (see Figure 7-17).


Step 3: Specify Workflow Actions

Specify the workflow actions that will be triggered when the rule criteria are met. See an example

Rule Criteria: Account: Active EQUALS Yes
Evaluation Criteria: Evaluate the rule when a record is created, and any time it's edited to subsequently meet criteria

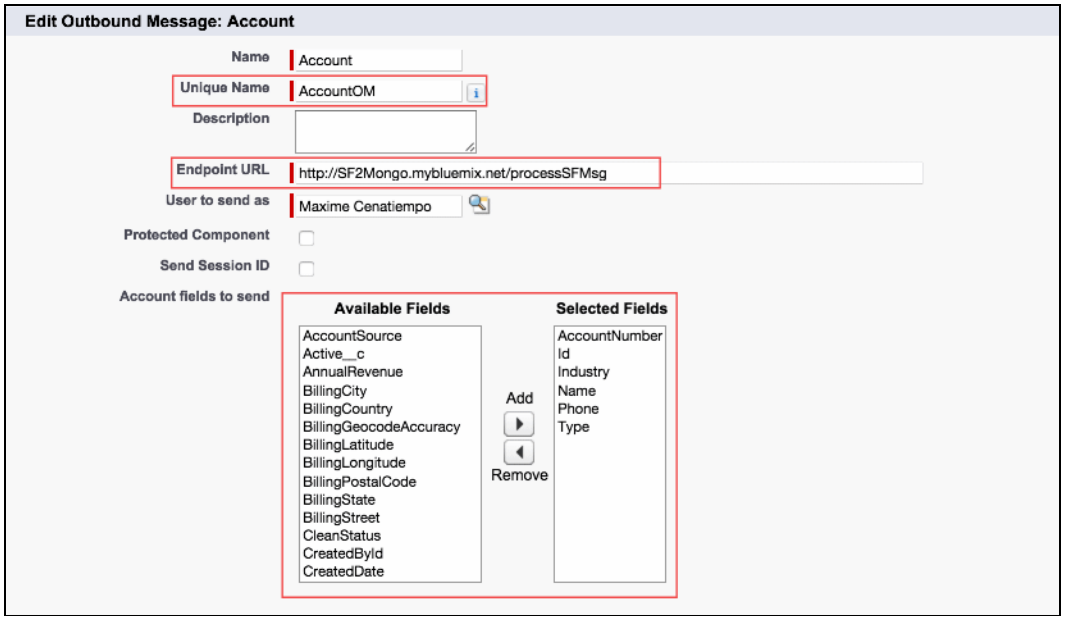
Immediate Workflow Actions

No workflow actions have been added.

ave been added. Before adding a workflow action, you must have at least one time trigger defined.

Figure 7-17 Workflow action selection

2. Enter the required information as shown in Figure 7-18 and click **Save**.



Edit Outbound Message: Account

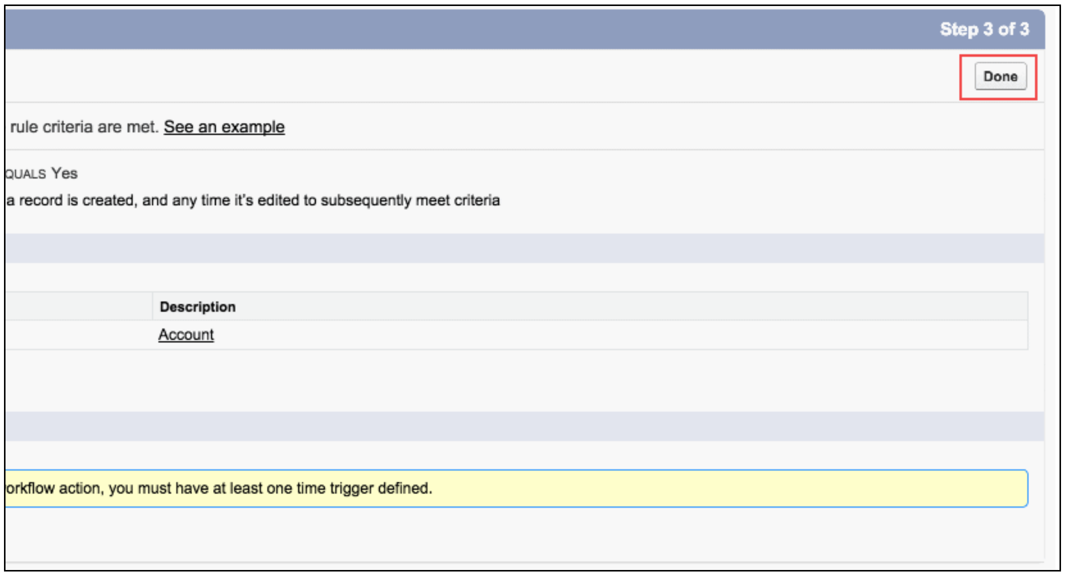
Name	Account
Unique Name	AccountOM
Description	
Endpoint URL	http://SF2Mongo.mybluemix.net/processSFMsg
User to send as	Maxime Cenatiempo
Protected Component	☐
Send Session ID	☐
Account fields to send	

Available Fields	Selected Fields
AccountSource	AccountNumber
Active__c	Id
AnnualRevenue	Industry
BillingCity	Name
BillingCountry	Phone
BillingGeocodeAccuracy	Type
BillingLatitude	
BillingLongitude	
BillingPostalCode	
BillingState	
BillingStreet	
CleanStatus	
CreatedById	
CreatedDate	

Figure 7-18 Outbound message details

Note: Ensure that you specify your own endpoint URL. It should be the same URL that you used when you tested the application on Bluemix as described in 7.4.5, “Deploying the application” on page 206.

3. Once completed, click **Done**. See Figure 7-19.



rule criteria are met. See an example

QUALS Yes

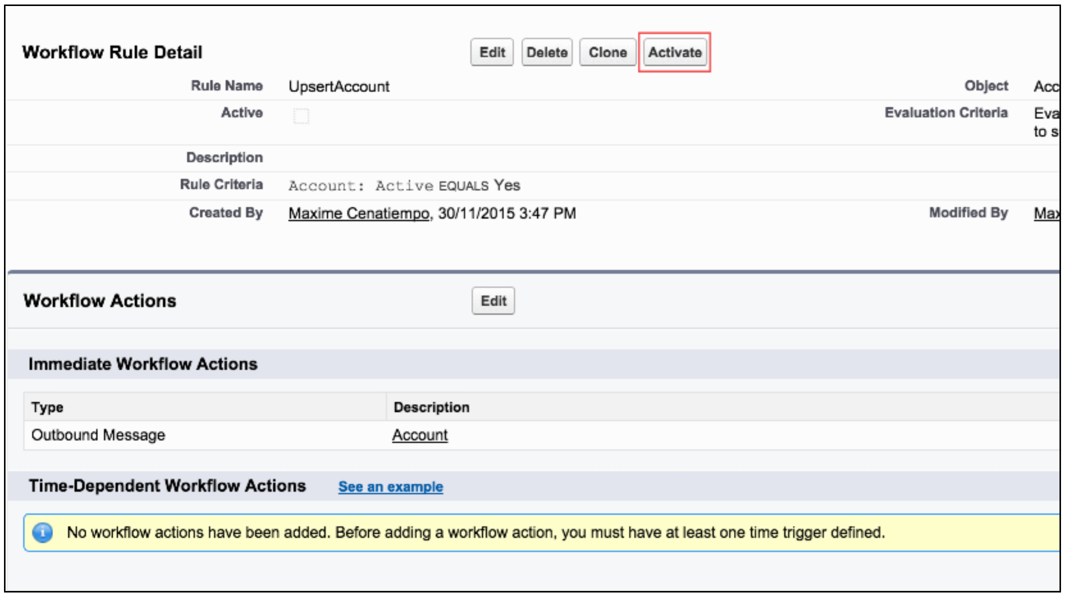
a record is created, and any time it's edited to subsequently meet criteria

Description
Account

orkflow action, you must have at least one time trigger defined.

Figure 7-19 Save workflow rule

4. Finally, you need to activate this workflow rule by clicking **Activate**, as shown in Figure 7-20.



Workflow Rule Detail

Rule Name	UpsertAccount	Object	Acc
Active	☐	Evaluation Criteria	Eva to s
Description			
Rule Criteria	Account: Active EQUALS Yes		
Created By	Maxime Cenatiempo, 30/11/2015 3:47 PM	Modified By	Max

Workflow Actions

Immediate Workflow Actions

Type	Description
Outbound Message	Account

Time-Dependent Workflow Actions See an example

No workflow actions have been added. Before adding a workflow action, you must have at least one time trigger defined.

Figure 7-20 Workflow rule activation

Your Salesforce instance is now configured to trigger an outbound message with some account information to your application running on Bluemix. This outbound message is triggered when an account is created or updated in Salesforce and if this account is active. Your application returns an acknowledgment response (true or false) to Salesforce after processing the message.

7.6 End-to-end testing

You can now test the end-to-end flow from Salesforce to MongoDB through the application running on Bluemix.

Clean up the database

Before testing the solution, you can clean the database and remove all existing documents in the account collection.

Using the mongo shell as described in “Configure MongoDB” on page 191, connect to the database and run the command as shown in Example 7-23.

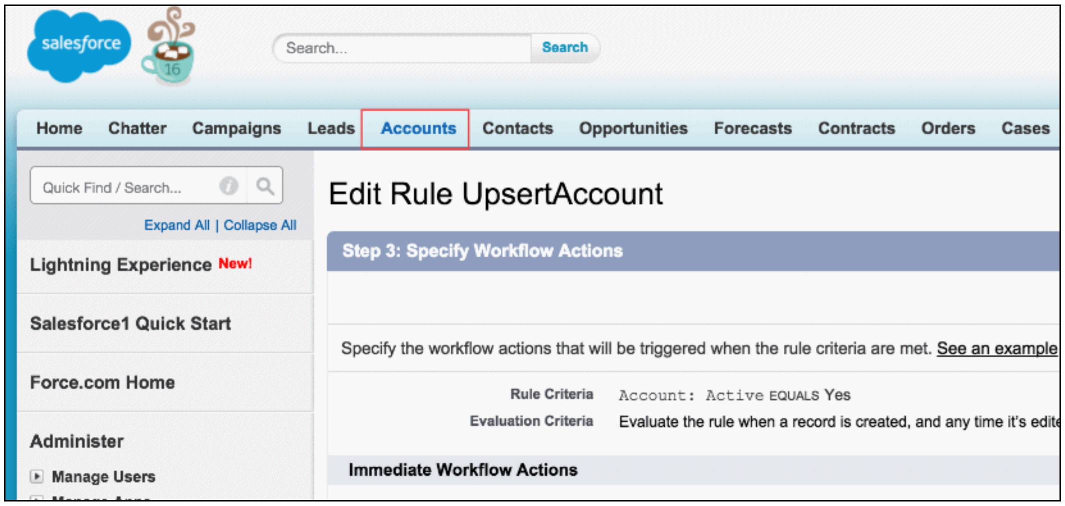
Example 7-23 Clean the database

```
> db.Account.remove({})
WriteResult({ "nRemoved" : 2 })
```

Create a new account

Create a new account in Salesforce to trigger an outbound message and test the flow between Salesforce and the database. Follow these steps to create an account:

1. In Salesforce, click **Accounts** to open the Accounts page. See Figure 7-21.



Edit Rule UpsertAccount

Step 3: Specify Workflow Actions

Specify the workflow actions that will be triggered when the rule criteria are met. See an example

Rule Criteria Account: Active EQUALS Yes

Evaluation Criteria Evaluate the rule when a record is created, and any time it's edit

Immediate Workflow Actions

Figure 7-21 Link to Accounts page

2. Click **New** to create a new account, as shown in Figure 7-22.

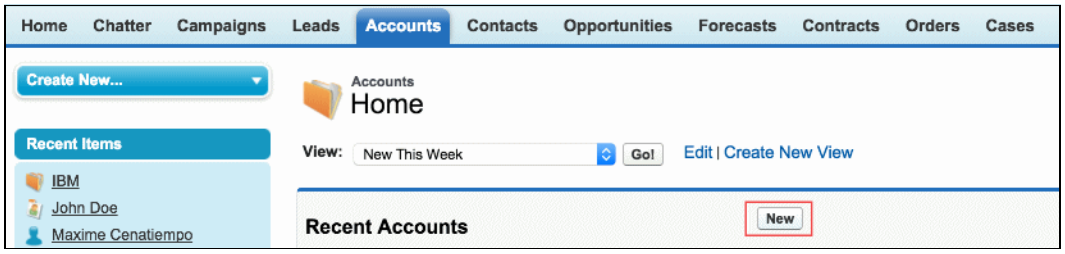
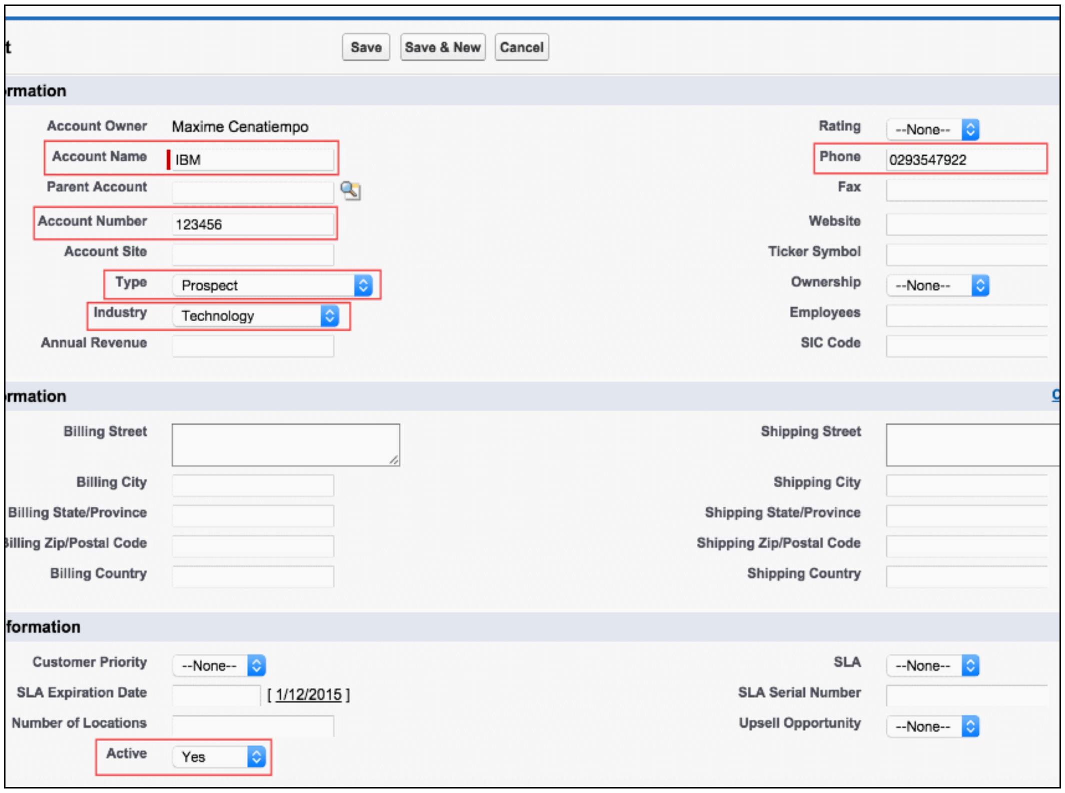


Figure 7-22 Accounts page

3. Enter the required information as shown in Figure 7-23 and click **Save**.



rmation

Account Owner	Maxime Cenatiempo	Rating	--None--
Account Name	IBM	Phone	0293547922
Parent Account		Fax	
Account Number	123456	Website	
Account Site		Ticker Symbol	
Type	Prospect	Ownership	--None--
Industry	Technology	Employees	
Annual Revenue		SIC Code	

rmation

Billing Street		Shipping Street	
Billing City		Shipping City	
Billing State/Province		Shipping State/Province	
illing Zip/Postal Code		Shipping Zip/Postal Code	
Billing Country		Shipping Country	

formation

Customer Priority	--None--	SLA	--None--
SLA Expiration Date	[1/12/2015]	SLA Serial Number	
Number of Locations		Upsell Opportunity	--None--
Active	Yes		

Figure 7-23 Account details

As soon as the account is created, the workflow rule created earlier is automatically triggered. An outbound message is sent to the application on Bluemix, which creates a new document in the database running locally.

Verify result

As an account is created in Salesforce, a new document is created in the database.

Using the mongo shell as described in “Configure MongoDB” on page 191, connect to the database and run the command as shown in Example 7-24.

Example 7-24 Retrieve accounts in database

```
> db.Account.find()
{ "_id" : "0019000001buQSWAA2", "num" : "123456", "name" : "IBM", "type" :
"Prospect", "industry" : "Technology", "phone" : "0293547922" }
```

This demonstrates that the end-to-end flow is working as you are getting a new account created in your MongoDB database as soon as an account is created in Salesforce.

You can repeat the above steps to test the creation of additional accounts. You can also edit an existing account in Salesforce (for example, you can modify the account name) and you will see that the account is updated accordingly in the database.

Note: If you want to track what's happening along the way, use one of the following methods:

- You can track the delivery status of outbound messages in Salesforce as specified at the following URL:

  `https://help.Salesforce.com/apex/HTViewHelpDoc?id=workflow_tracking_outbound_message_delivery_status.htm&language=en`

  Salesforce queues up the outbound messages if they cannot be processed by the remote application. If these messages are processed successfully, they are removed from the delivery queue.

- You can view the runtime logs of your application running on Bluemix through the Bluemix dashboard or through the command-line interface as described at the following URL:

  `https://www.ng.bluemix.net/docs/monitor_log/monitoringandlogging.html`

- You can monitor Secure Gateway on Bluemix as shown at the following URL:

  `https://www.ng.bluemix.net/docs/services/SecureGateway/sg_022.html#sg_004`

  Additionally, you can see useful information and usage statistics in the gateway client logs as described at the following URL:

  `https://www.ng.bluemix.net/docs/services/SecureGateway/sg_022.html#sg_020`

- You can check the MongoDB logs by running the **`getLog`** command as described at the following URL:

  `https://docs.mongodb.org/manual/reference/command/getLog`

7.7 Conclusion

In this chapter, we looked at various concepts:

- Expose and access a local database through IBM Secure Gateway
- Integrate database and compose API with StrongLoop
- Deploy, manage, and test the application on Bluemix
- Configure Salesforce for remote invocation

With this knowledge, you can apply the same principles to integrate Salesforce with any enterprise backend system running in your own data center through IBM Bluemix.

8

Integrating events from Internet of Things with Enterprise Asset Management systems

In this chapter, we look at how events from an Internet of Things (IoT) device can be used for alerting the asset management team and integrating the event messages automatically with an on-premises asset management system. We created a scenario here that creates a work order in Maximo Asset Management System if the sensor attached to a manufacturing equipment detects that the equipment is faulty and requires maintenance. The components for this solution include:

- Maximo Asset Management
- IBM Bluemix Internet of Things Foundation (IoTF)
- IBM Bluemix Secure Gateway service
- Node-RED Flow editor
- NodeJS application
- Twilio for sending SMS alert

This chapter has the following sections:

8.1 Scenario

This section describes the IoT scenario that we implement to demonstrate the capabilities of event integration using Secure Gateway. In later sections, we cover the basics of building a solution like this using the building blocks or services from Bluemix. IBM Maximo Asset Management is acting as an enterprise system of record (SOR). We provide an overview of IBM Maximo Asset Management, its benefit, and purpose in Section 8.5, “IBM Maximo Asset Management solution” on page 239.

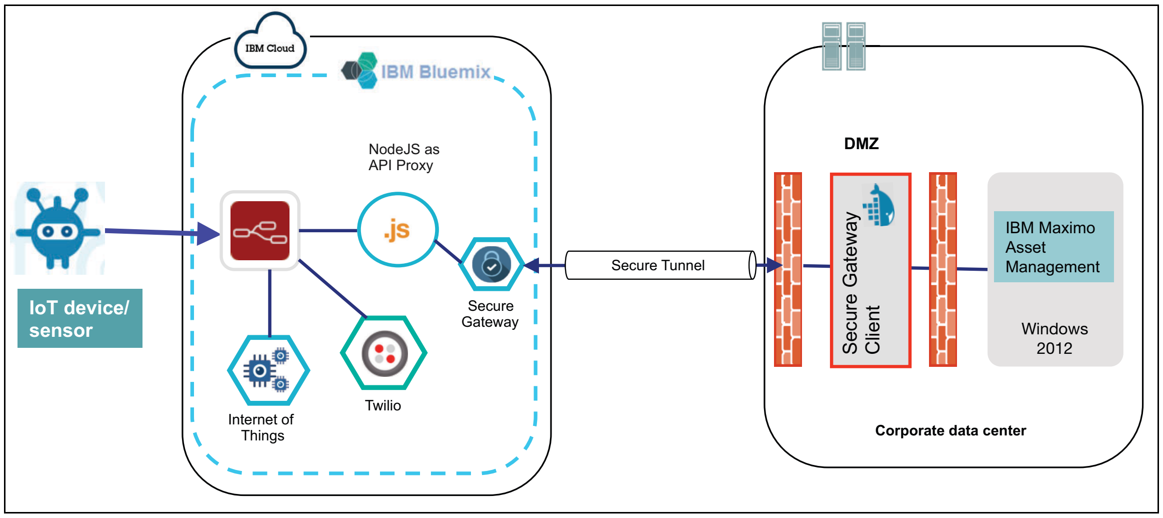


Figure 8-1 Internet of Things scenario

8.2 Introduction to IBM Internet of Things Foundation

In this section, we provide an overview of IBM Internet of Things Foundation (IoTF) and the capabilities if offers.

8.2.1 Quickstart mode

Here the *quickstart mode* refers to a flow creation in Node-RED quickly without registering the device. This mode requires entering the unique device ID of the sensor as illustrated in Figure 8-8 on page 223. There is also an IBM Internet of Things Foundation Quickstart service. This is a tool for connecting sensors using MQ Telemetry Transport (MQTT) protocols (MQTT 3.1 at the minimum) to IoTF quickly.

For more information, see the following URL:

`https://quickstart.internetofthings.ibmcloud.com`

For MQTT, see 3.6, “Introduction to IBM MessageSight” on page 60.

This section takes you through step-by-step instructions to create an IoT application in quickstart mode. This section reads the sensor data sent from a Texas Instruments (TI) SimpleLink SensorTag via the SensorTag app installed on an iPhone. SensorTag app installation and prerequisite steps to connect to IBM IoT are described in the following website:

`https://developer.ibm.com/recipes/tutorials/connect-a-cc2650-sensortag-to-the-iot-foundations-quickstart`

Therefore, this section only describes the steps needed to create the flow to read the sensor data and write to a debug node in Node-RED in JavaScript Object Notation (JSON) format. Write down the device ID that you get from your iOS SensorTag app. It is required later.

Note: Do not worry if you do not have a TI SensorTag. If you have a Raspberry Pi or Intel Galileo or any other device that talks MQTT, you can connect that too. For a list of tutorials or recipes, see the following link:

`https://developer.ibm.com/recipes`

If you do not have a device at all, you can use a simulated sensor that Bluemix provides from the following URL:

`https://quickstart.internetofthings.ibmcloud.com/iotsensor`

Follow these steps to complete this exercise:

1. Create a *space* in your Bluemix organization. Use the same space for all the steps in this chapter. We used our own organization and the space dev throughout this chapter.

   Note: For more information about how to create a space, see the following link:

   `https://www.ng.bluemix.net/docs/admin/index.html`

2. Choose **Node-RED Starter** from the Bluemix catalog as shown in Figure 8-2.

   Note: The window as shown was captured at the time of writing this book. A number of items in boilerplate sections vary from one Bluemix region to another. We created this app in Bluemix US South location.

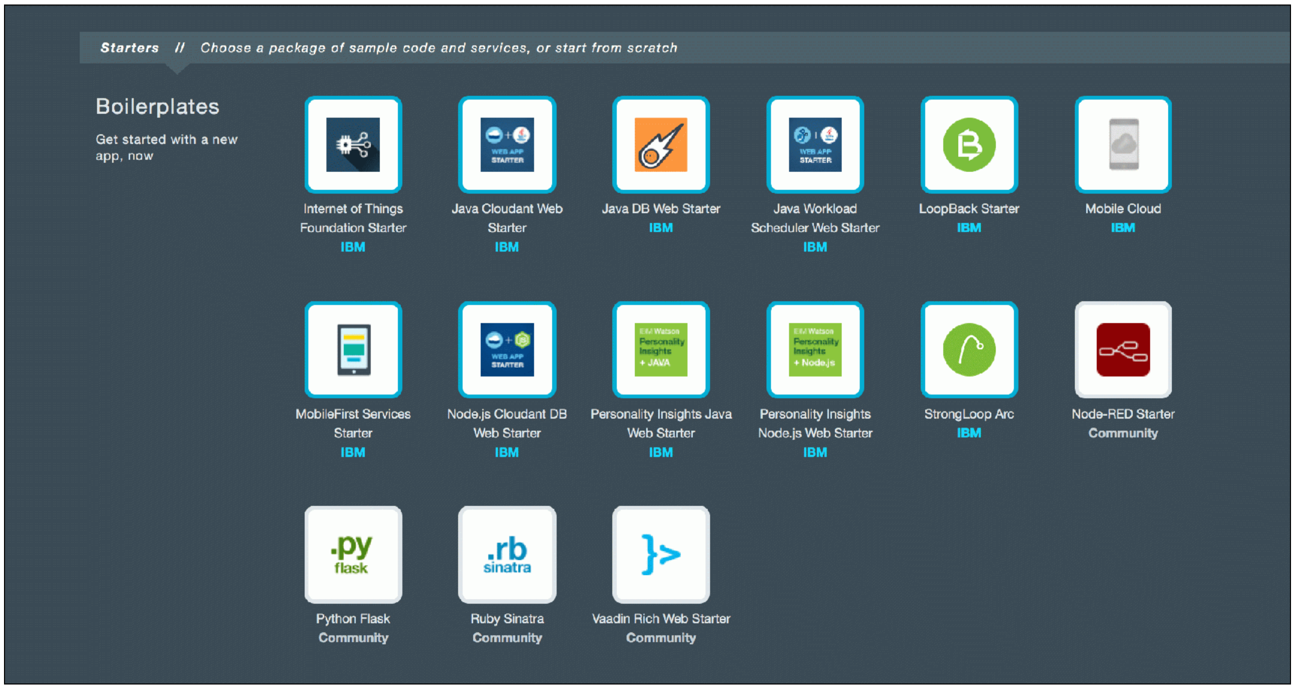
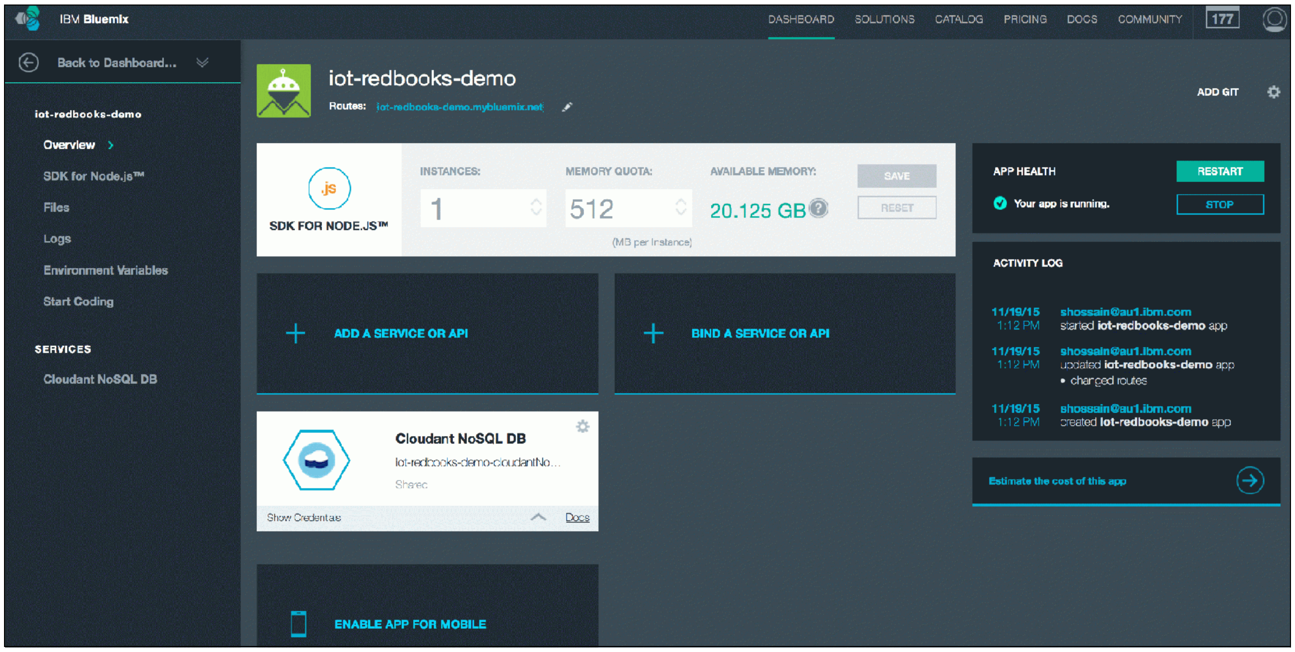

Starters // Choose a package of sample code and services, or start from scratch
Boilerplates
Get started with a new app, now
Internet of Things Foundation Starter IBM
Java Cloudant Web Starter IBM
Java DB Web Starter IBM
Java Workload Scheduler Web Starter IBM
LoopBack Starter IBM
Mobile Cloud IBM
MobileFirst Services Starter IBM
Node.js Cloudant DB Web Starter IBM
Personality Insights Java Web Starter IBM
Personality Insights Node.js Web Starter IBM
StrongLoop Arc IBM
Node-RED Starter Community
Python Flask Community
Ruby Sinatra Community
Vaadin Rich Web Starter Community

Figure 8-2 Node-RED Starter boilerplates among others from Bluemix catalog

3. The next step is to choose a name for the Starter app. We chose `iot-redbooks-demo` as the app name, all the default options, and then clicked **Create**.
4. In a minute or so, the `iot-redbooks-demo` app will be deployed and running on Bluemix. Figure 8-3 shows the application dashboard after it started. As you can see from the diagram, it is running on NodeJS runtime and using 512 MB of memory. This diagram also shows that only one instance of the app is running. Node-RED Starter app from Bluemix catalog is always bundled with a Cloudant NoSQL database. This window also shows that a Cloudant NoSQL database `iot-redbooks-demo-cloudantNoSQLDB` is bound to this Node-RED app running on NodeJS runtime. Environment Variables from the left pane direct to **`VCAP_SERVICES`** and **`USER-DEFINED`** variables. You need to add two key-value pairs in the **`USER-DEFINED`** variable section soon. Spend some time to familiarize yourself with the variables.

> **Note:** See the following URL to learn more about Cloud Foundry variables:
>
> `https://docs.cloudfoundry.org/devguide/deploy-apps/environment-variable.html`



iot-redbooks-demo
Routes: iot-redbooks-demo.mybluemix.net

INSTANCES: 1
MEMORY QUOTA: 512 (MB per instance)
AVAILABLE MEMORY: 20.125 GB

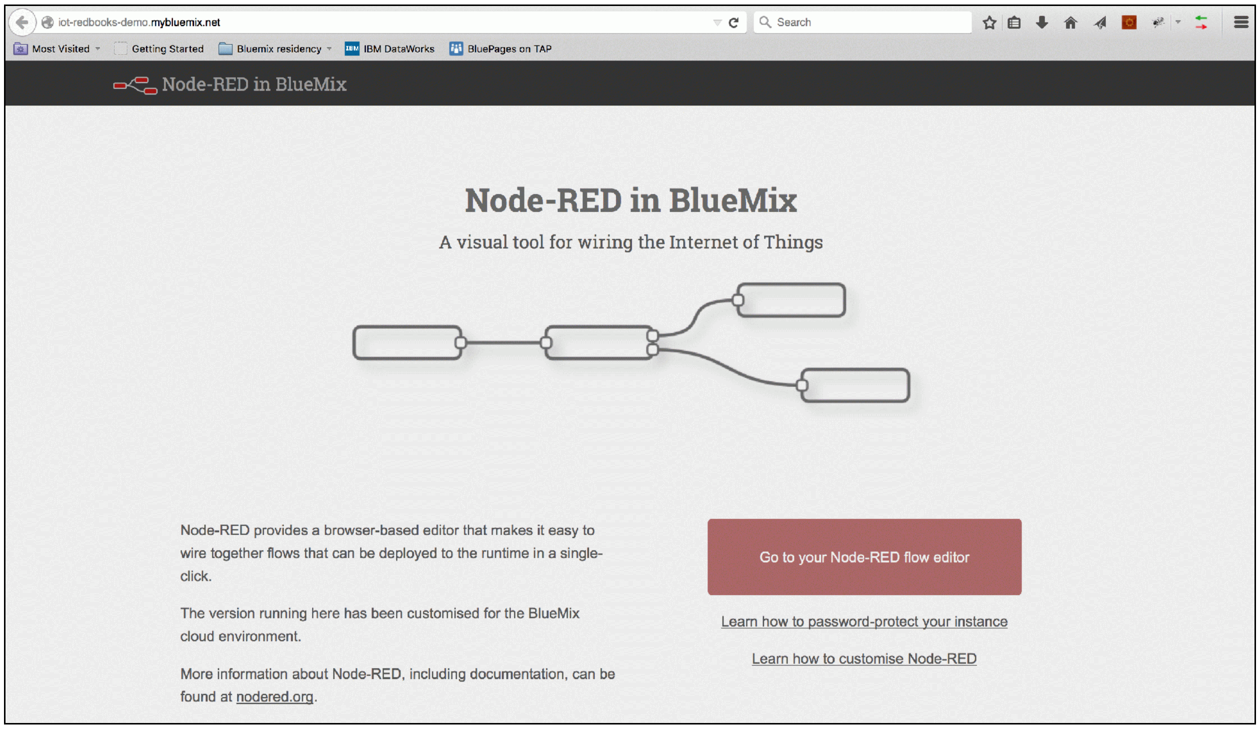
Your app is running.

11/19/15 1:12 PM shossain@au1.ibm.com started **iot-redbooks-demo** app
11/19/15 1:12 PM shossain@au1.ibm.com updated **iot-redbooks-demo** app
• changed routes
11/19/15 1:12 PM shossain@au1.ibm.com created **iot-redbooks-demo** app

Cloudant NoSQL DB
iot-redbooks-demo-cloudantNo...
Shared

Figure 8-3 Node-RED Starter

5. Click the URL from the **Routes:** label as illustrated in Figure 8-3. It opens in another browser tab. You will see a window similar to Figure 8-4 on page 221. Click **Go to your Node-RED flow editor** to go to the Node-RED flow editor.


iot-redbooks-demo.mybluemix.net
Node-RED in BlueMix
Node-RED in BlueMix
A visual tool for wiring the Internet of Things
Go to your Node-RED flow editor
Learn how to password-protect your instance
Learn how to customise Node-RED

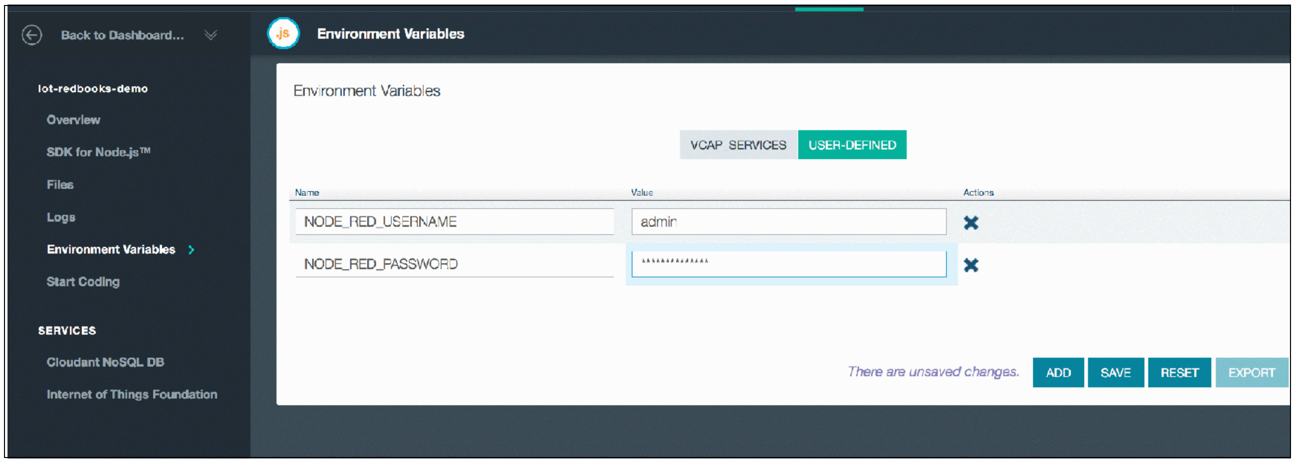
Node-RED provides a browser-based editor that makes it easy to wire together flows that can be deployed to the runtime in a single-click.

The version running here has been customised for the BlueMix cloud environment.

More information about Node-RED, including documentation, can be found at nodered.org.

Figure 8-4 Node-RED welcome page

Anyone is able to access the flow editor and modify or create new flow and deploy. This is not a secure way to manage your Node-RED based application. Bluemix allows you to make your application secure by adding environment variables. As shown in Figure 8-5, add two variables named **`NODE_RED_USERNAME`** and **`NODE_RED_PASSWORD`**. Your application needs to be restaged for this change to be effective.



Name	Value	Actions
NODE_RED_USERNAME	admin	✖
NODE_RED_PASSWORD	**************	✖

Figure 8-5 Password protecting the Node-RED editor using environment variables

6. Click **Go to your Node-RED flow editor** as shown in Figure 8-4 on page 221. This time, there is a prompt to enter credentials before proceeding to the flow editor. See Figure 8-6.

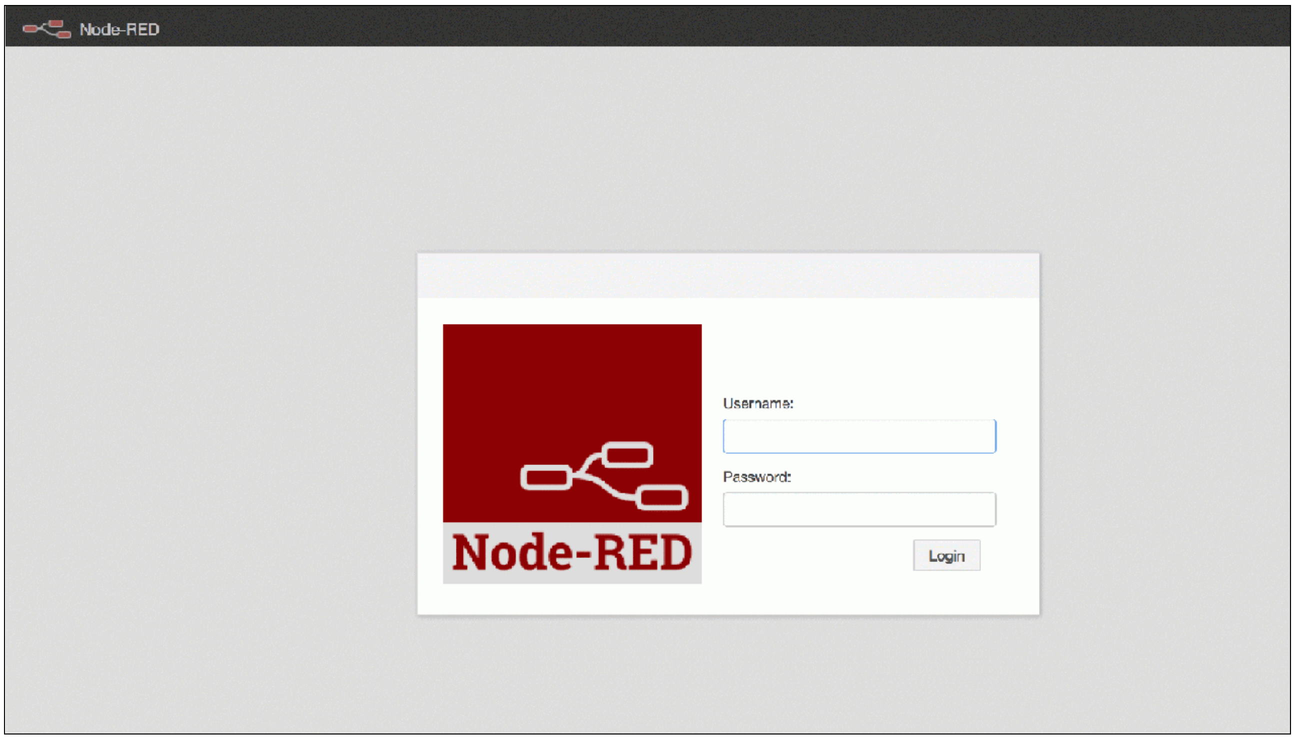
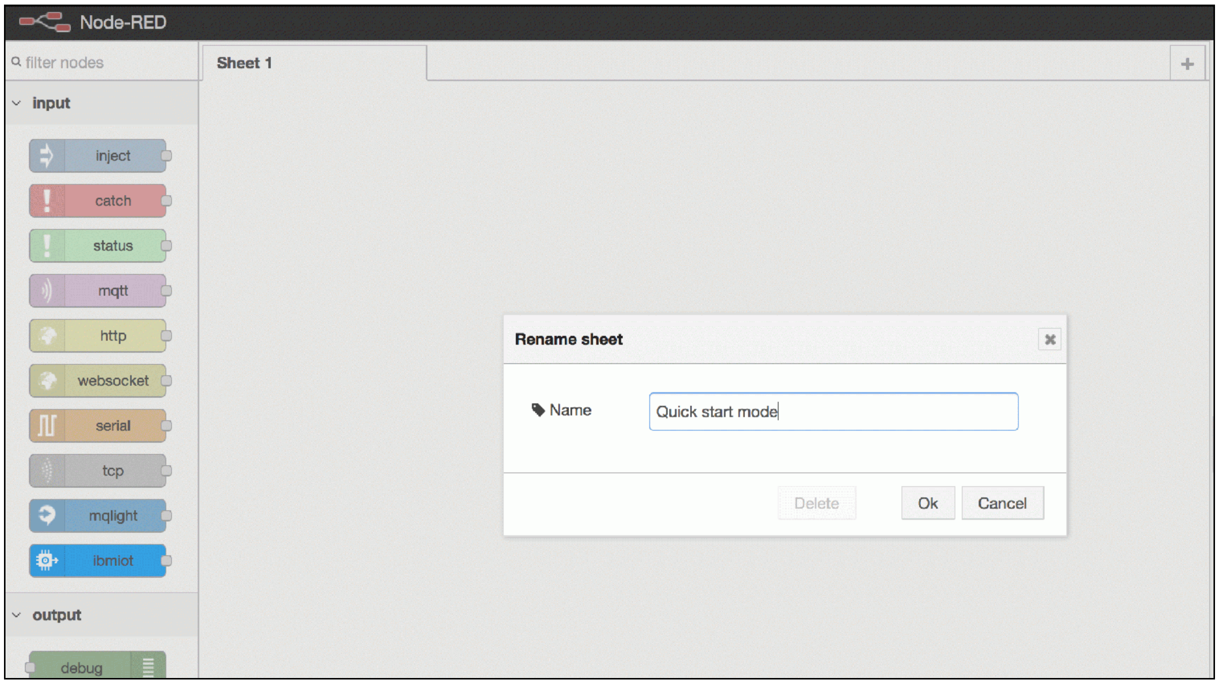


Figure 8-6 Node-RED editor prompting for credentials

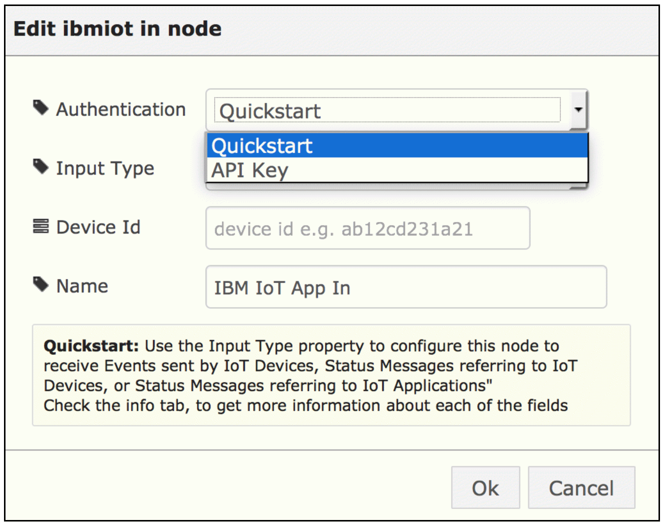
Initially, there is an empty sheet named *Sheet 1*. Double-click this tab to rename it to *Quick start mode* in the dialog box as shown in Figure 8-7. We create another tab to compose a more complex flow in the next section. Multiple tabs allow you to keep separate apps and logic organized separately for ease of management.



Name: Quick start mode

Figure 8-7 Renaming the initial sheet to Quick start mode

7. Now drag an **ibmiot** node from the left palette of the window into the `Quick start mode` sheet or tab. Double-click the **ibmiot** node and select **Quickstart** from the Authentication drop-down list as shown in Figure 8-8. Remember to enter the TI SensorTag Device ID that you noted in the prerequisite step and click **Ok**. You now have a simple IoT app that can read sensor data from the SensorTag.



Edit ibmiot in node

Authentication: Quickstart (options: Quickstart, API Key)

Input Type

Device Id: device id e.g. ab12cd231a21

Name: IBM IoT App In

Quickstart: Use the Input Type property to configure this node to receive Events sent by IoT Devices, Status Messages referring to IoT Devices, or Status Messages referring to IoT Applications"
Check the info tab, to get more information about each of the fields

Figure 8-8 Node-RED Quickstart mode

You have the option to do anything with the raw data that you get from the sensor. We extracted the raw data in JSON format and then sent it to a DEBUG node to see the content. This simple flow looks similar to the flow in Figure 8-9.

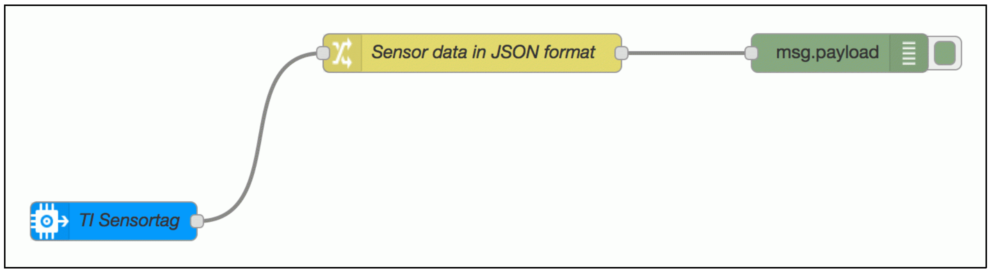


Figure 8-9 A simple flow in quickstart mode

You have the option to import the flow in JSON format by selecting **Import** → **Clipboard** as shown in Figure 8-10. Copy and paste the JSON-based flow from Example 8-1. Click anywhere in the editor to place the flow. Also, remember to change the `deviceId` in the JSON file with the actual device ID.

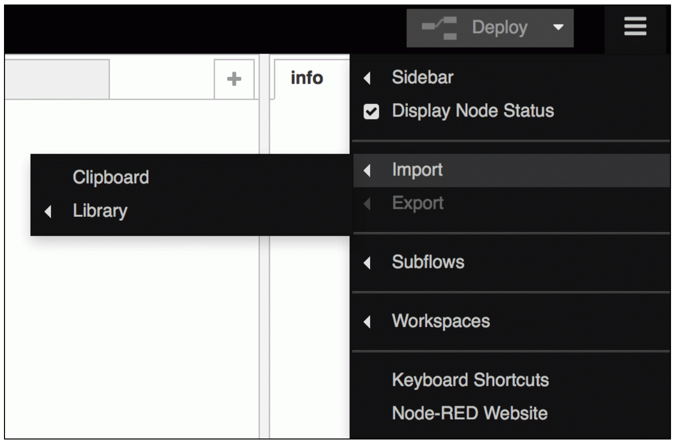


Figure 8-10 Importing a flow in JSON format

Example 8-1 JSON-based flow for the example app

```
{
      "id":"2b22eacc.20fcd6",
      "type":"ibmiot in",
      "z":"9be28101.43699",
      "authentication":"quickstart",
      "apiKey":"",
      "inputType":"evt",
      "deviceId":"EnterYourDeviceIDHere",
      "applicationId":"",
      "deviceType":"+",
      "eventType":"+",
      "commandType":"",
      "format":"json",
      "name":"TI Sensortag",
      "service":"quickstart",
      "allDevices":false,
      "allApplications":false,
      "allDeviceTypes":true,
      "allEvents":true,
      "allCommands":false,
      "allFormats":false,
      "x":110,
      "y":320,
      "wires":[
         [
            "68bbd4f1.2a2ba4"
         ]
      ]
   },
   {
      "id":"68bbd4f1.2a2ba4",
      "type":"change",
      "z":"9be28101.43699",
      "name":"Sensor data in JSON format",
      "rules":[
         {
            "t":"set",
            "p":"payload",
            "to":"msg.payload.d"
         }
      ],
      "action":"",
      "property":"",
      "from":"",
      "to":"",
      "reg":false,
      "x":390,
      "y":189,
      "wires":[
         [
            "dab1dff1.2ab71"
         ]
      ]
   },
   {
      "id":"dab1dff1.2ab71",
      "type":"debug",
      "z":"9be28101.43699",
      "name":"",
      "active":true,
      "console":"false",
      "complete":"false",
      "x":677,
      "y":189,
      "wires":[

      ]
   }
]
```

8. After all the tasks above have been completed, you will see an output similar to Example 8-2 from the debug tab from the right side.

Example 8-2 Output of the flow

```
{
   "key1":"0",
   "key2":"0",
   "AmbTemp":"24.9375",
   "IRTemp":"19.71875",
   "humidity":"74.33875",
   "accX":"0.02191162",
   "accY":"0.01220703",
   "accZ":"0.2265625",
   "gyroX":"-0.7392883",
   "gyroY":"1.750946",
   "gyroZ":"1.330719",
   "magX":"-78.69873",
   "magY":"15.43994",
   "magZ":"-132.6636",
   "optical":"7.95"
}
```

This concludes this simple exercise. In this section, you learned:

- The basics of IBM Node-RED Starter app.
- How to connect an IoT device to IoTF in a Quickstart mode.
- How to import a flow into the editor.
- Receive sensor data from a sensor and finally see the contents from the debug tab.

8.2.2 Registering an IoT device

In this section, we securely connect an IoT device to the IBM Internet of Things Foundation (IoTF). This allows you to manage the sensor or IoT device to be managed from IoTF. As explained in Section 8.2.1, "Quickstart mode" on page 218, the IBM IoT recipe website contains a list of tutorials for connecting various devices to IoTF. If you do not have a physical device to connect to IoTF, you can create an improvised IoT device. This requires some programming and modifying parameters specific to IoTF organization ID, device ID, authentication method, token, and device type. Read the following article to learn more about this and look at sample source code provided:

http://www.ibm.com/developerworks/cloud/library/cl-mqtt-bluemix-iot-node-red-app

The simulated and managed IoT device in this section has been created by using source code from the above article. It has other components, which will be discussed later.

Perform the following steps to register an IoT device:

1. Create an IOTF service in the Space dev from the `iot-redbooks-demo` app dashboard. To do this, click **Add a service or API** from the dashboard as shown in Figure 8-3 on page 220. This directs you to the Bluemix catalog. Now select the **Internet of Things** check box as shown in Figure 8-11 on page 227.

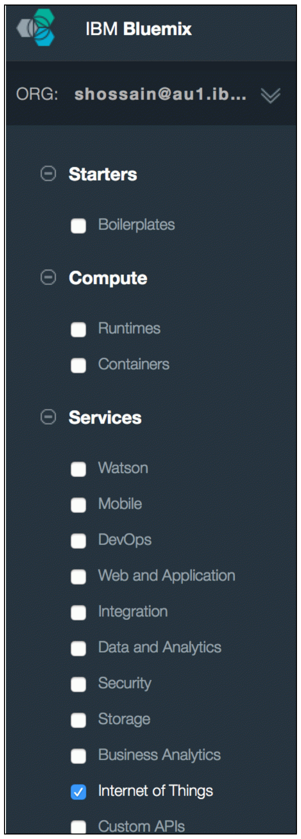


Figure 8-11 Selecting IoT service from IBM Bluemix catalog

2. After this option is selected, the Bluemix Internet of Things category and services in this category are displayed in the page that looks similar to Figure 8-12. Select **Internet of Things Foundation**.

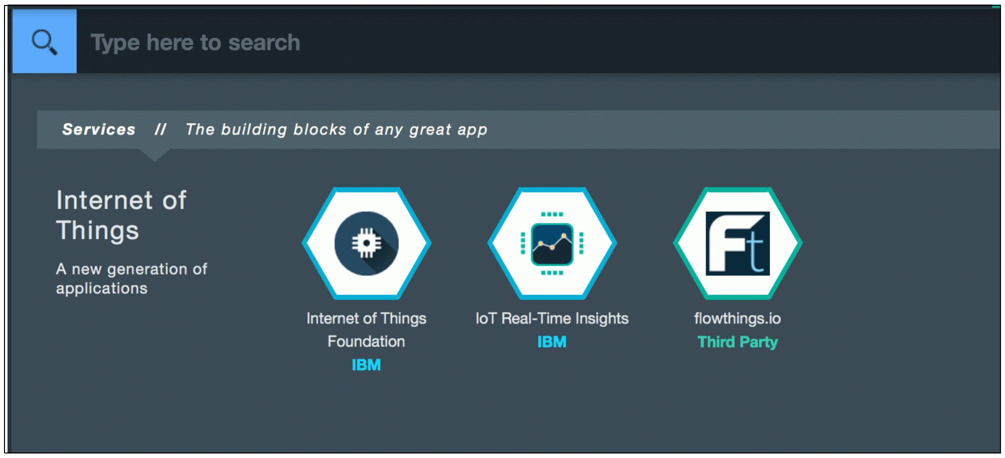
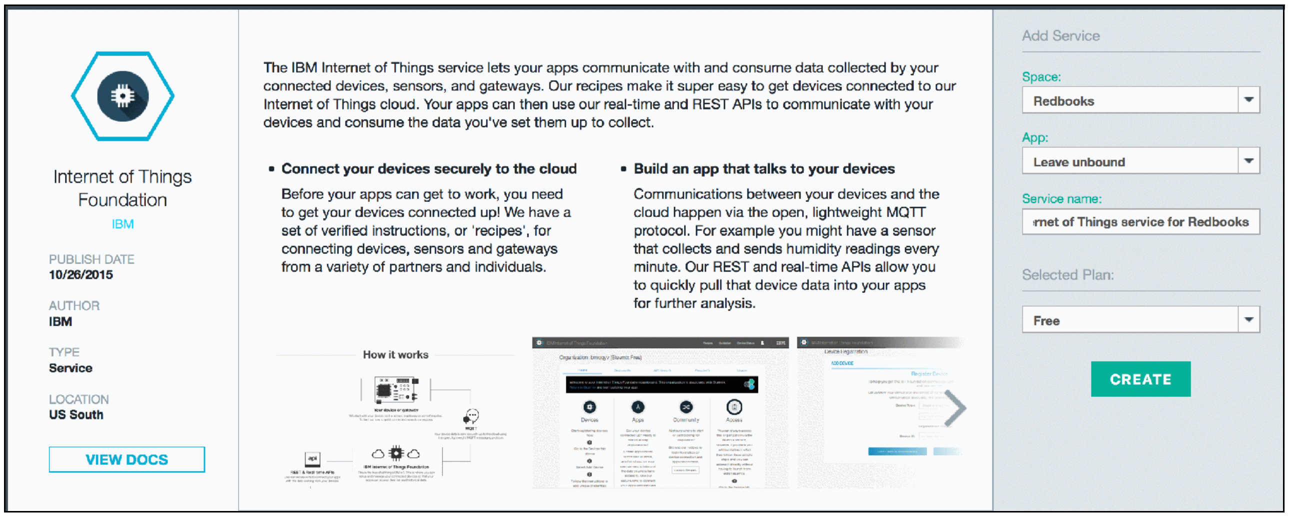

Type here to search
Services // The building blocks of any great app
Internet of Things
A new generation of applications
Internet of Things Foundation
IBM
IoT Real-Time Insights
IBM
flowthings.io
Third Party

Figure 8-12 Currently available IoT services from IBM and a third party

3. Enter **iot-redbooks** for **Service name:** and click **CREATE** with other options as shown in Figure 8-13. This new service is bound to the `iot-redbooks-demo` app.



Internet of Things Foundation
IBM

PUBLISH DATE
10/26/2015

AUTHOR
IBM

TYPE
Service

LOCATION
US South

VIEW DOCS

The IBM Internet of Things service lets your apps communicate with and consume data collected by your connected devices, sensors, and gateways. Our recipes make it super easy to get devices connected to our Internet of Things cloud. Your apps can then use our real-time and REST APIs to communicate with your devices and consume the data you've set them up to collect.

- **Connect your devices securely to the cloud**
  Before your apps can get to work, you need to get your devices connected up! We have a set of verified instructions, or 'recipes', for connecting devices, sensors and gateways from a variety of partners and individuals.
- **Build an app that talks to your devices**
  Communications between your devices and the cloud happen via the open, lightweight MQTT protocol. For example you might have a sensor that collects and sends humidity readings every minute. Our REST and real-time APIs allow you to quickly pull that device data into your apps for further analysis.

How it works

Add Service

Space: Redbooks

App: Leave unbound

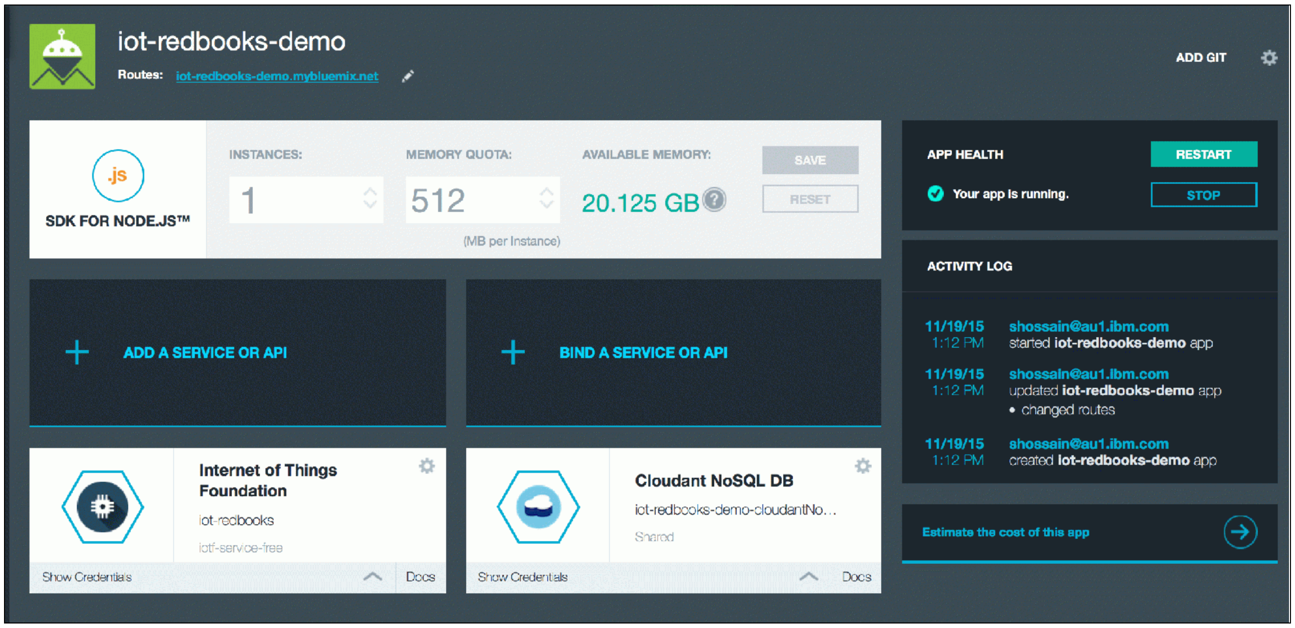
Service name: rnet of Things service for Redbooks

Selected Plan: Free

CREATE

Figure 8-13 Creating an Internet of Things Foundation service

4. `iot-redbooks-demo` is restaged for this new addition to be effective. Figure 8-14 shows the dashboard with newly added IoTF service and previous bound Cloudant NoSQL DB service.



iot-redbooks-demo

Routes: iot-redbooks-demo.mybluemix.net

ADD GIT

SDK FOR NODE.JS™

INSTANCES: 1

MEMORY QUOTA: 512 (MB per Instance)

AVAILABLE MEMORY: 20.125 GB

SAVE

RESET

APP HEALTH

RESTART

Your app is running.

STOP

ACTIVITY LOG

11/19/15 1:12 PM	shossain@au1.ibm.com started **iot-redbooks-demo** app
11/19/15 1:12 PM	shossain@au1.ibm.com updated **iot-redbooks-demo** app • changed routes
11/19/15 1:12 PM	shossain@au1.ibm.com created **iot-redbooks-demo** app

Estimate the cost of this app

ADD A SERVICE OR API

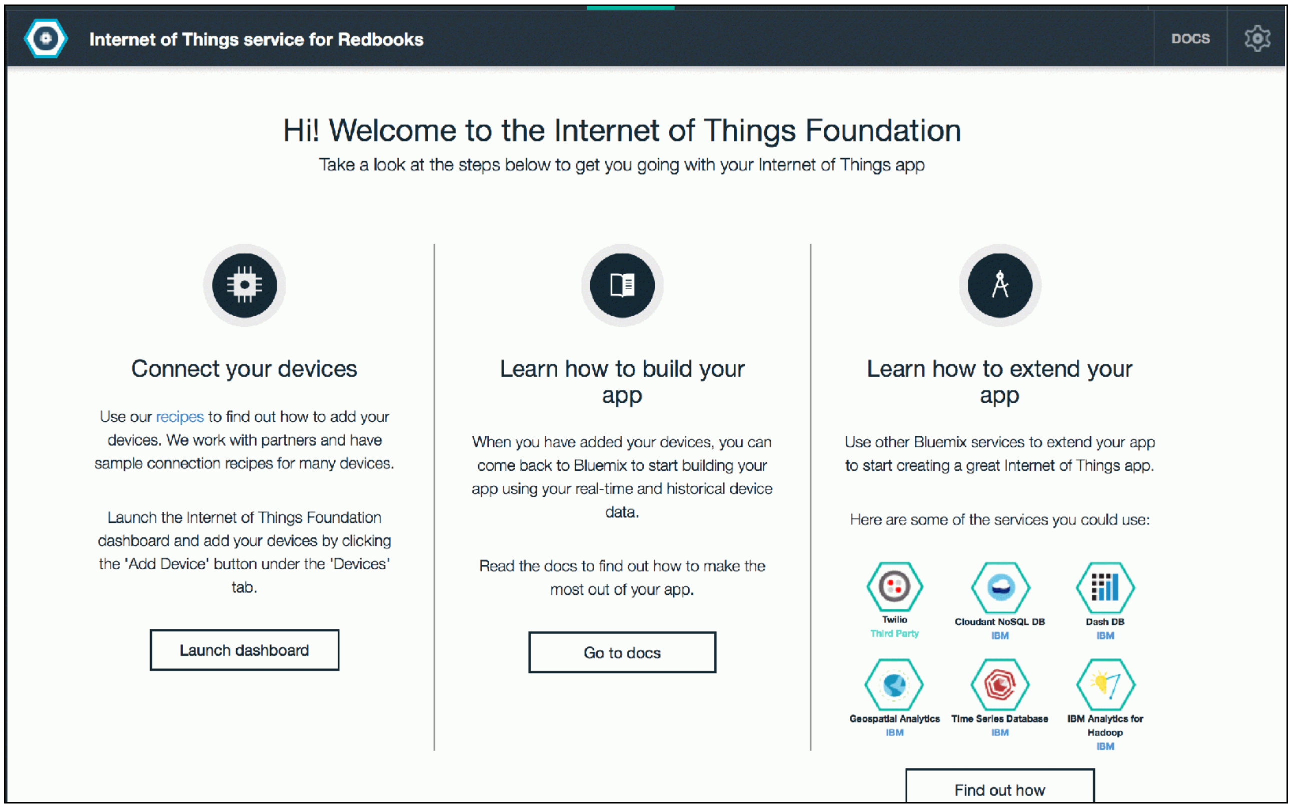
BIND A SERVICE OR API

Internet of Things Foundation
iot-redbooks
iotf-service-free
Show Credentials
Docs

Cloudant NoSQL DB
iot-redbooks-demo-cloudantNo...
Shared
Show Credentials
Docs

Figure 8-14 Node-RED Starter after addition of IoTF service

5. Click **Internet of Things Foundation** service from the dashboard. It directs you to the Internet of Things Foundation welcome page, as illustrated by Figure 8-15. This page has links to the IoTF dashboard and tutorials to get started.



Hi! Welcome to the Internet of Things Foundation

Take a look at the steps below to get you going with your Internet of Things app

Connect your devices

Use our recipes to find out how to add your devices. We work with partners and have sample connection recipes for many devices.

Launch the Internet of Things Foundation dashboard and add your devices by clicking the 'Add Device' button under the 'Devices' tab.

Learn how to build your app

When you have added your devices, you can come back to Bluemix to start building your app using your real-time and historical device data.

Read the docs to find out how to make the most out of your app.

Learn how to extend your app

Use other Bluemix services to extend your app to start creating a great Internet of Things app.

Here are some of the services you could use:

Figure 8-15 Welcome page showing links to dashboard, documentation, and tutorials

6. Click **Launch dashboard**. The OVERVIEW tab gives a summary of the number of devices registered, usage, data traffic, and access-related information. Because this is a new service, the number of devices connected in zero. As you can see from Figure 8-16, the Organization ID is `bu385r`.

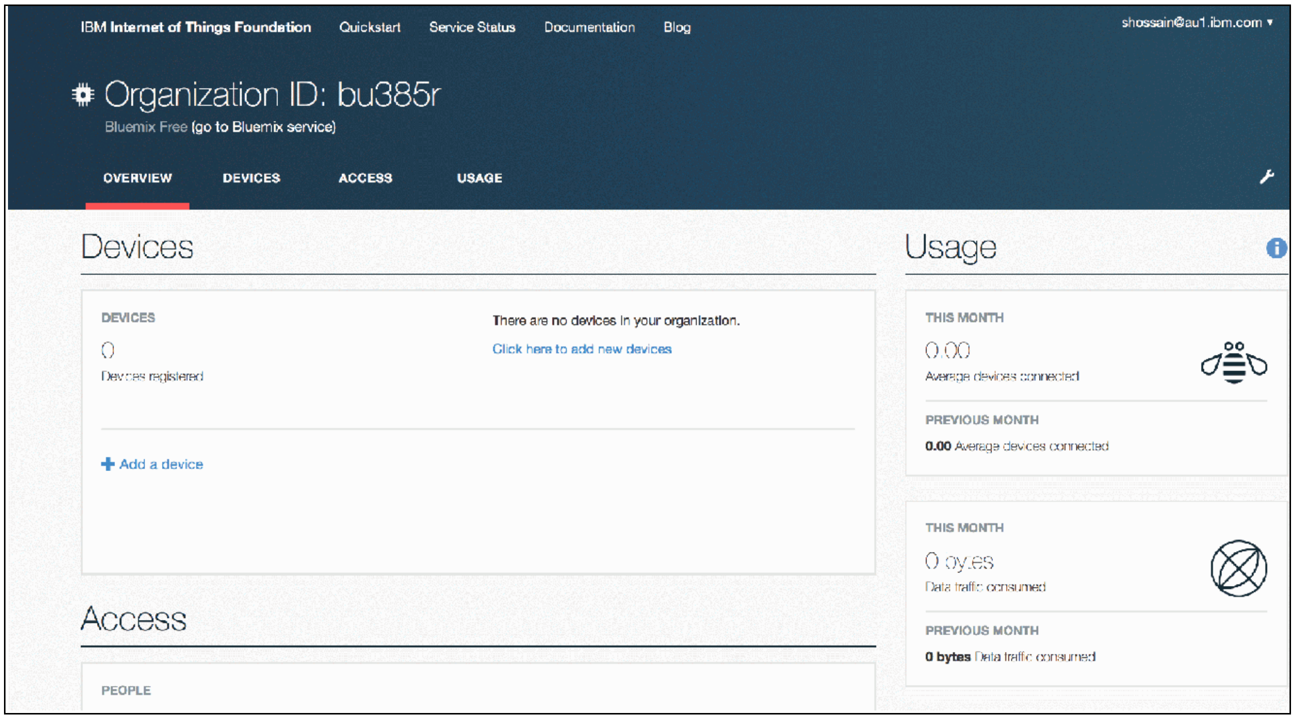


Figure 8-16 IoTF Dashboard

The Access section of the page shows information related to members in this organization, number of application programming interface (API) keys created, and number of Bluemix applications bound to this. This is illustrated in Figure 8-17. Only one Bluemix application (`iot-redbooks-demo`) has been bound to this service. Therefore, the number of Bluemix applications is one.

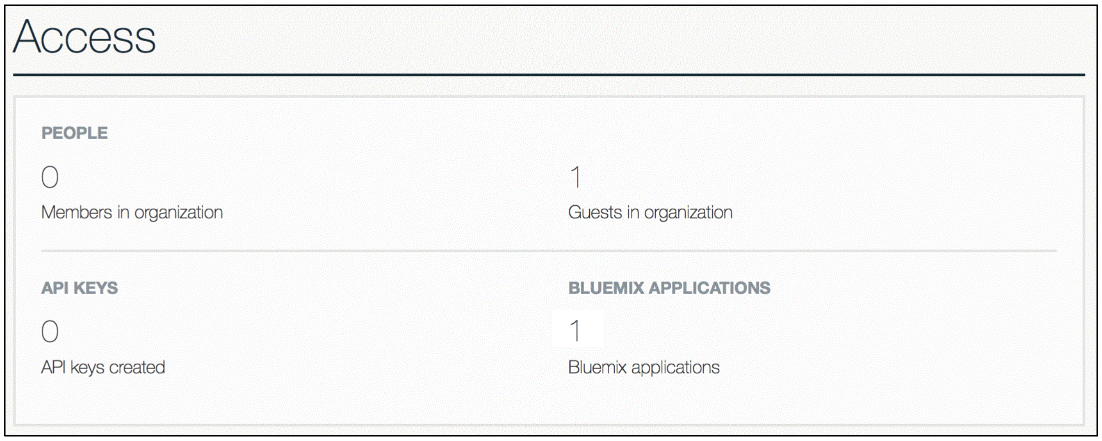


Figure 8-17 Dashboard showing access-related information for IoTF service

7. Click **+Add a device** (as shown in Figure 8-16 on page 230) to add a new device to the organization. At this stage, no device type is defined. As such, the **Choose Device Type** drop-down list is empty. Click **Create device type** to add a new device type.

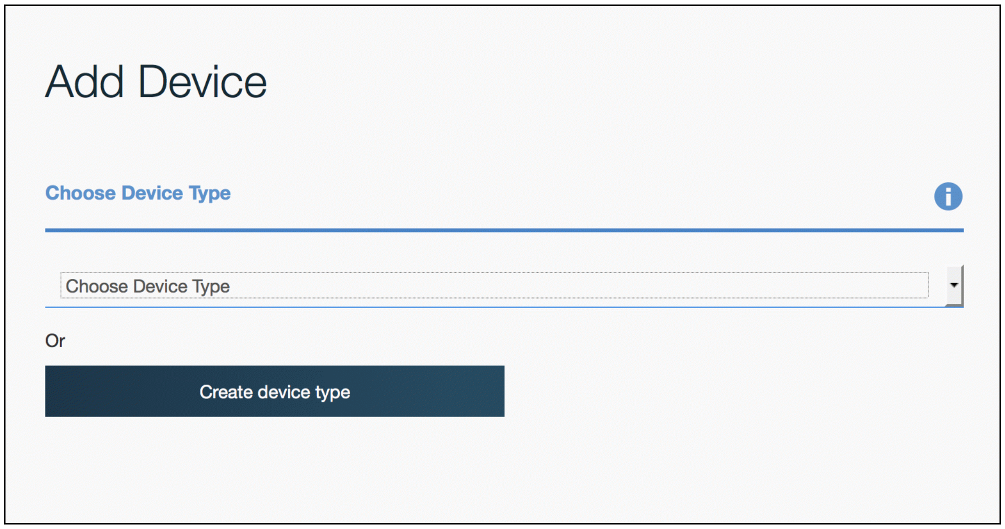
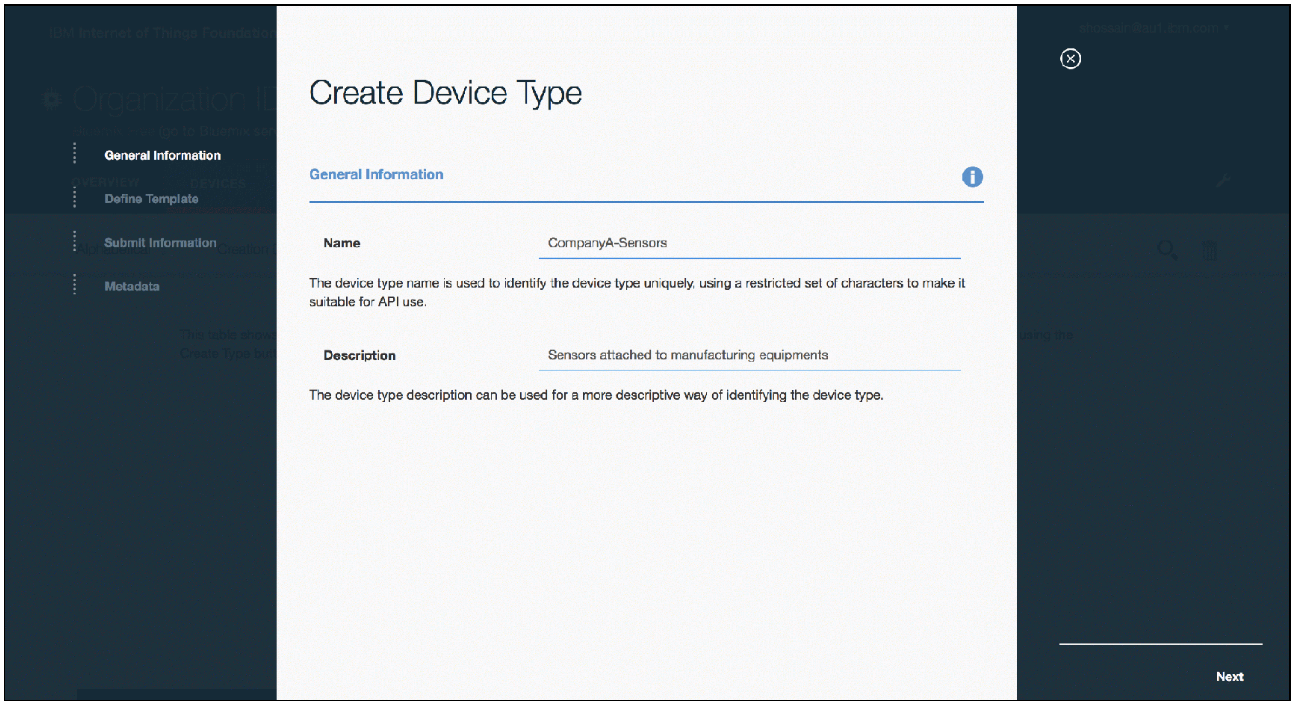


Figure 8-18 Creating a device type

8. Enter `CompanyA-Sensors` for **Name** and `Sensors attached to manufacturing equipments` for **Description** as shown Figure 8-19. Click **Next**.



Create Device Type

General Information

Name: CompanyA-Sensors

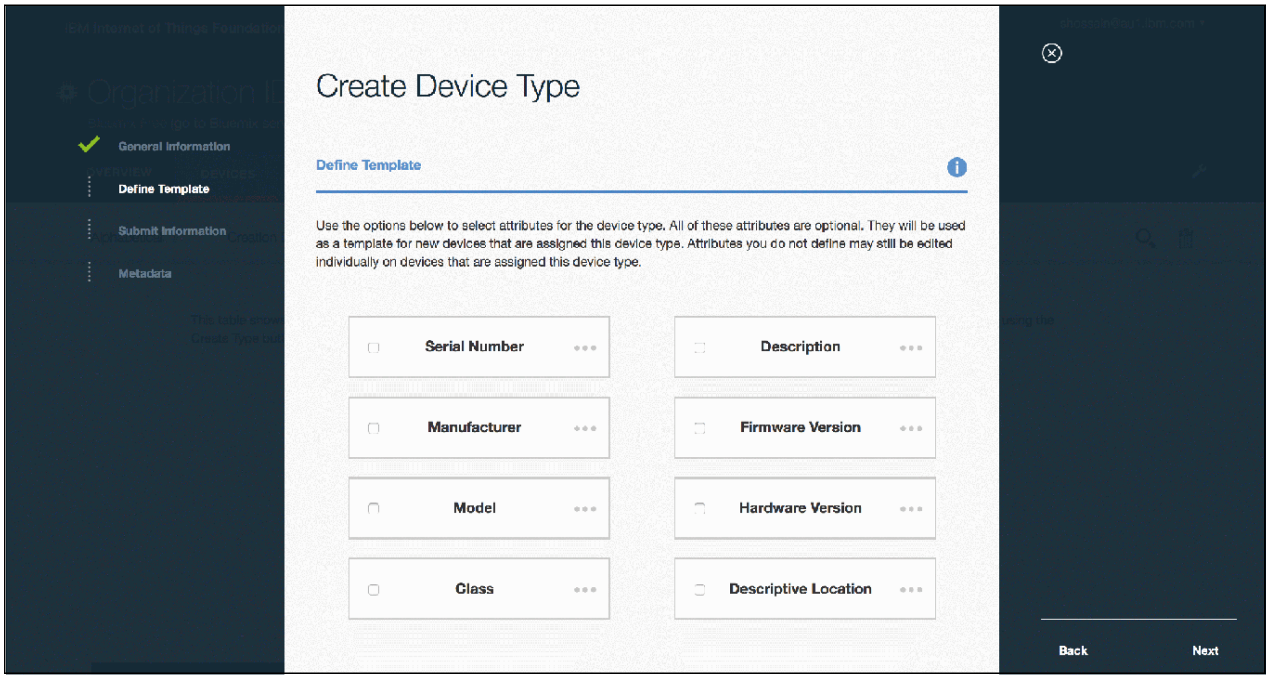
The device type name is used to identify the device type uniquely, using a restricted set of characters to make it suitable for API use.

Description: Sensors attached to manufacturing equipments

The device type description can be used for a more descriptive way of identifying the device type.

Figure 8-19 Entering details for device type

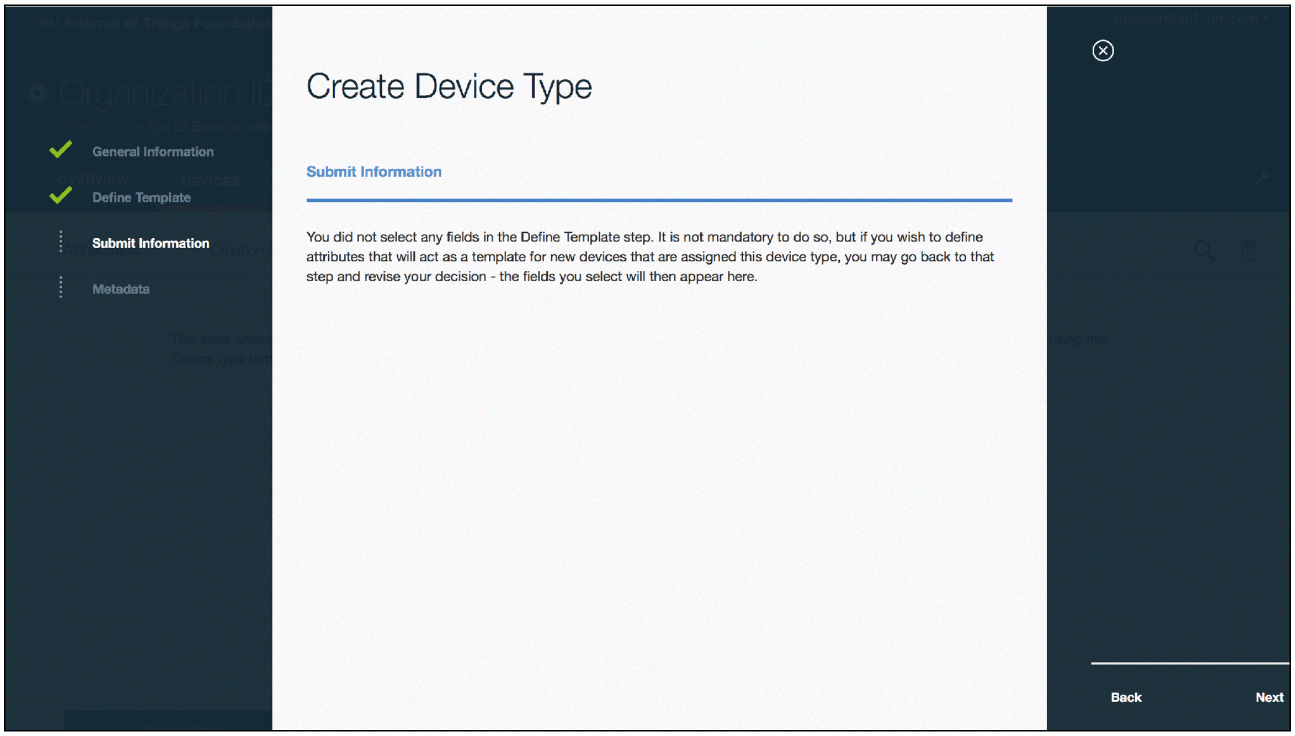
9. Parameters in the **Define Template** window as indicated in Figure 8-20 are optional. Click **Next**.



Use the options below to select attributes for the device type. All of these attributes are optional. They will be used as a template for new devices that are assigned this device type. Attributes you do not define may still be edited individually on devices that are assigned this device type.

Figure 8-20 Defining a template for the device type

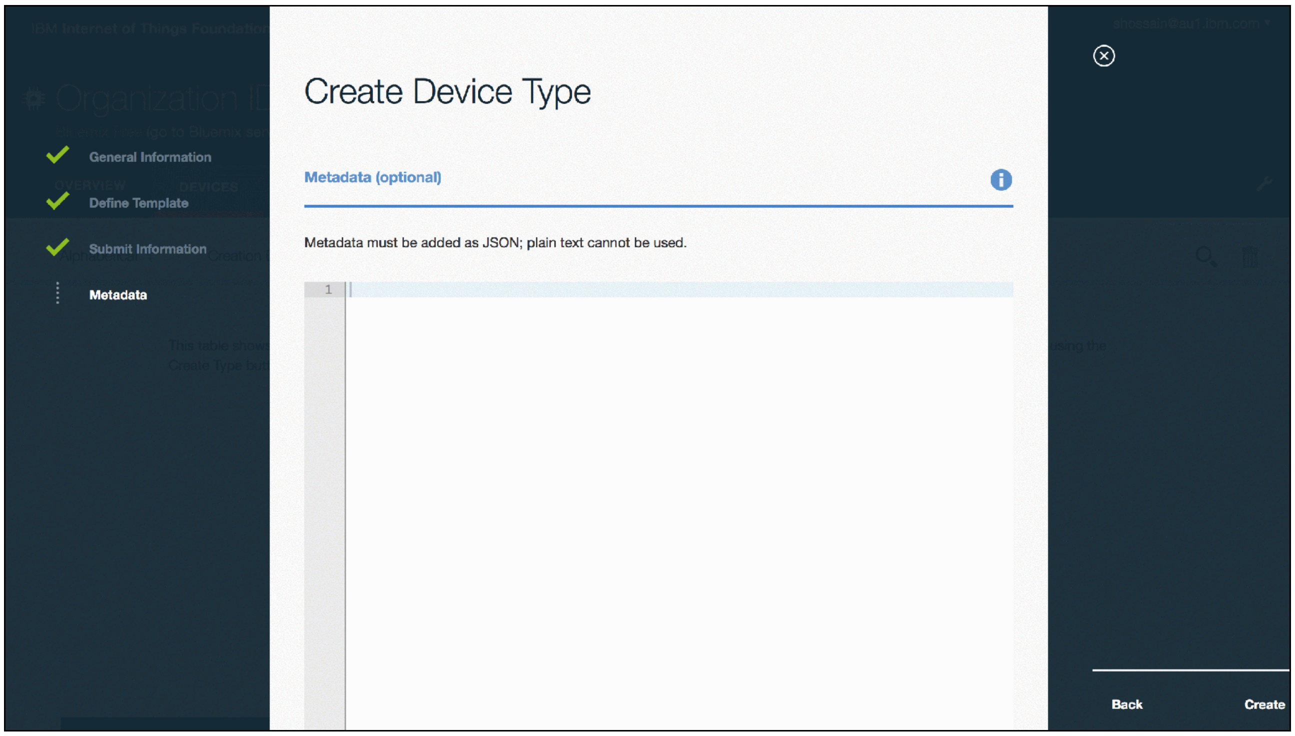
10. Click **Next** when the **Submit Information** window appears. See Figure 8-21.



You did not select any fields in the Define Template step. It is not mandatory to do so, but if you wish to define attributes that will act as a template for new devices that are assigned this device type, you may go back to that step and revise your decision - the fields you select will then appear here.

Figure 8-21 Submit information for the device type

11. The Metadata section is optional. Click **Create**. This completes all the steps to create the device type. See Figure 8-22.



Metadata must be added as JSON; plain text cannot be used.

Figure 8-22 Optional metadata for the device type

12. Now you are ready to add a device. In the Add Device window, select `CompanyA-Sensors` from the **Choose Device Type** drop-down list. Click **Next**.

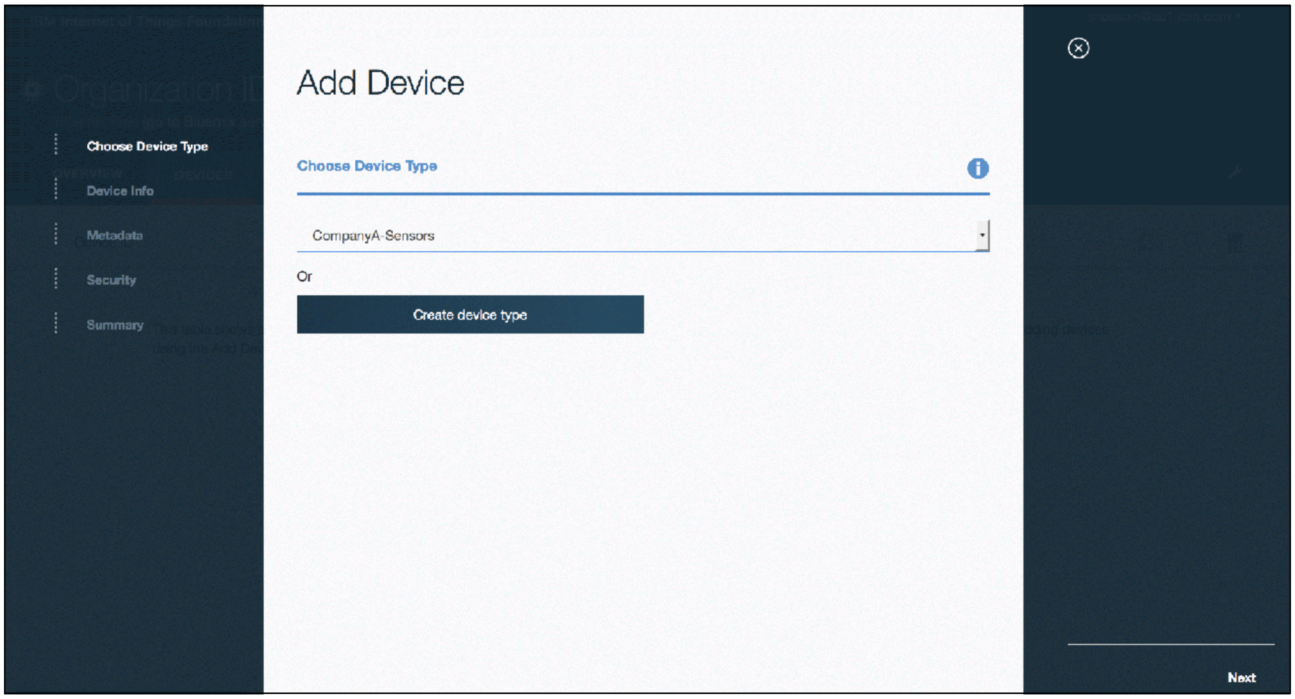
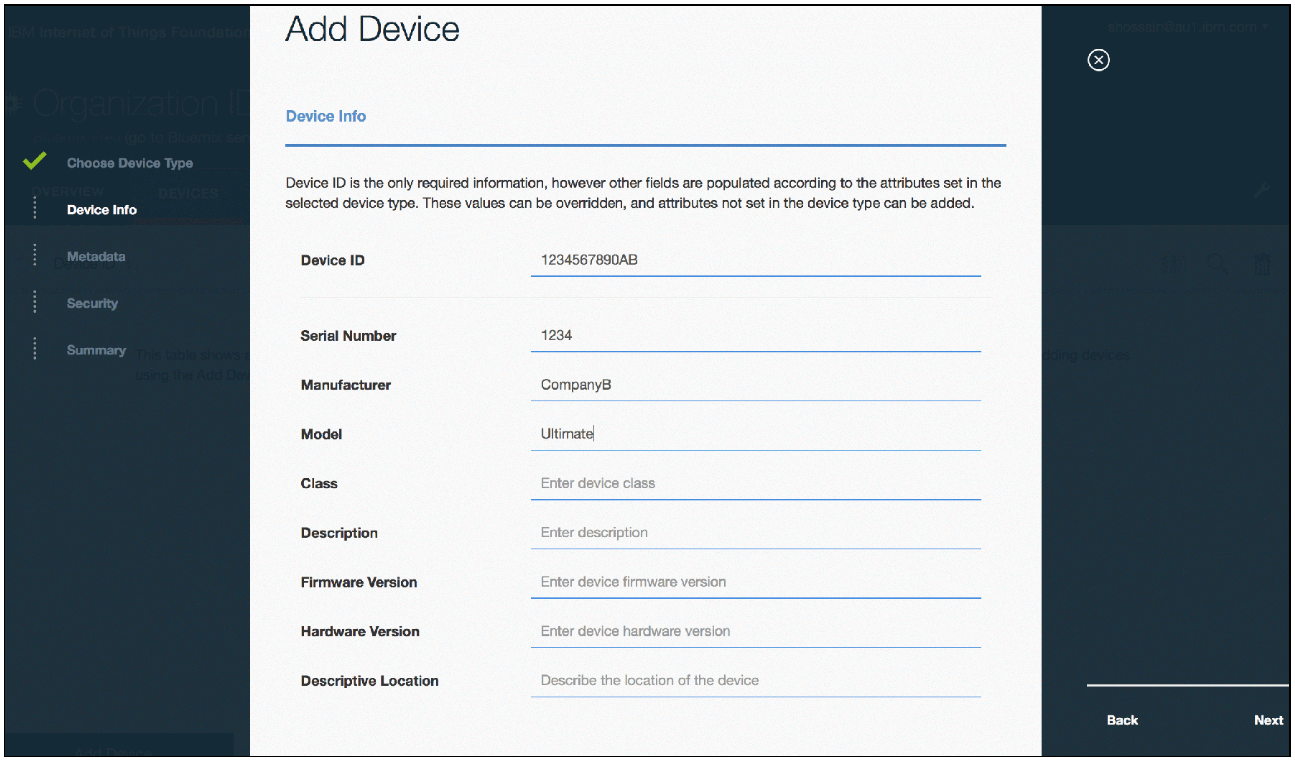


Figure 8-23 Choosing a device type for the Add Device window

13. Add Device ID, Serial Number, Manufacturer, and Model details as shown in Figure 8-24.

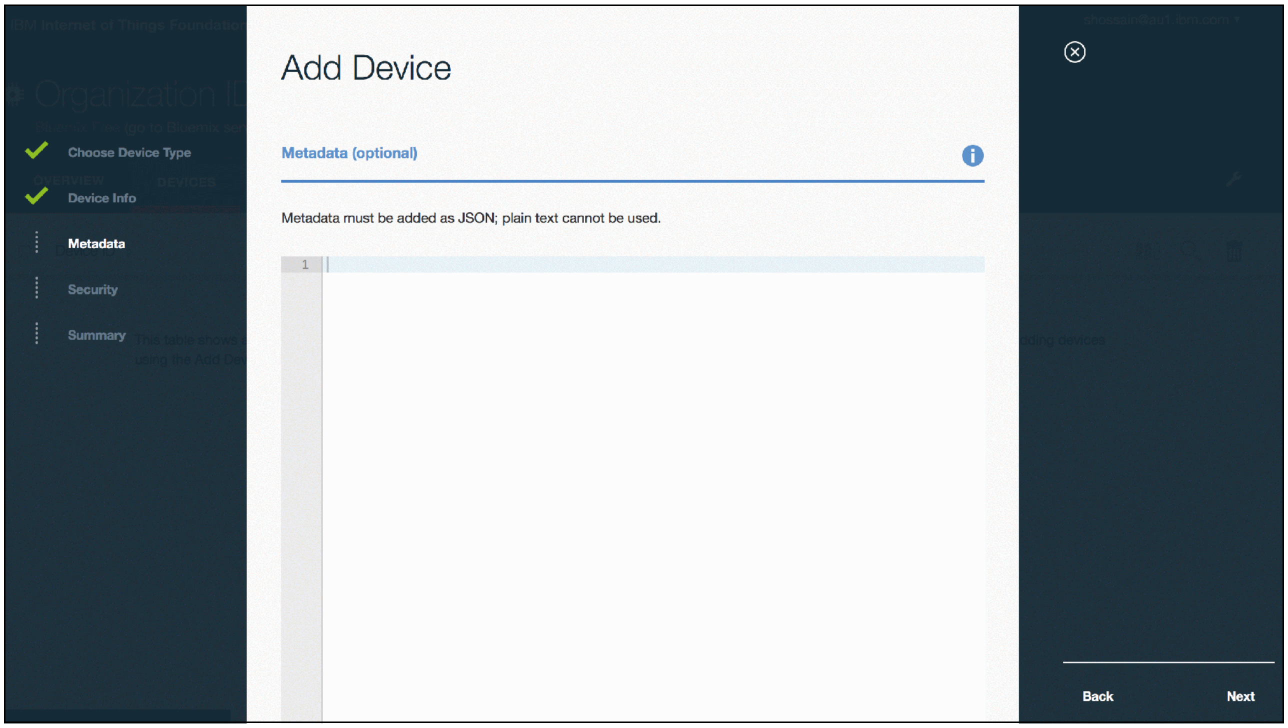


Device ID is the only required information, however other fields are populated according to the attributes set in the selected device type. These values can be overridden, and attributes not set in the device type can be added.

Device ID	1234567890AB
Serial Number	1234
Manufacturer	CompanyB
Model	Ultimate
Class	Enter device class
Description	Enter description
Firmware Version	Enter device firmware version
Hardware Version	Enter device hardware version
Descriptive Location	Describe the location of the device

Figure 8-24 Adding device details

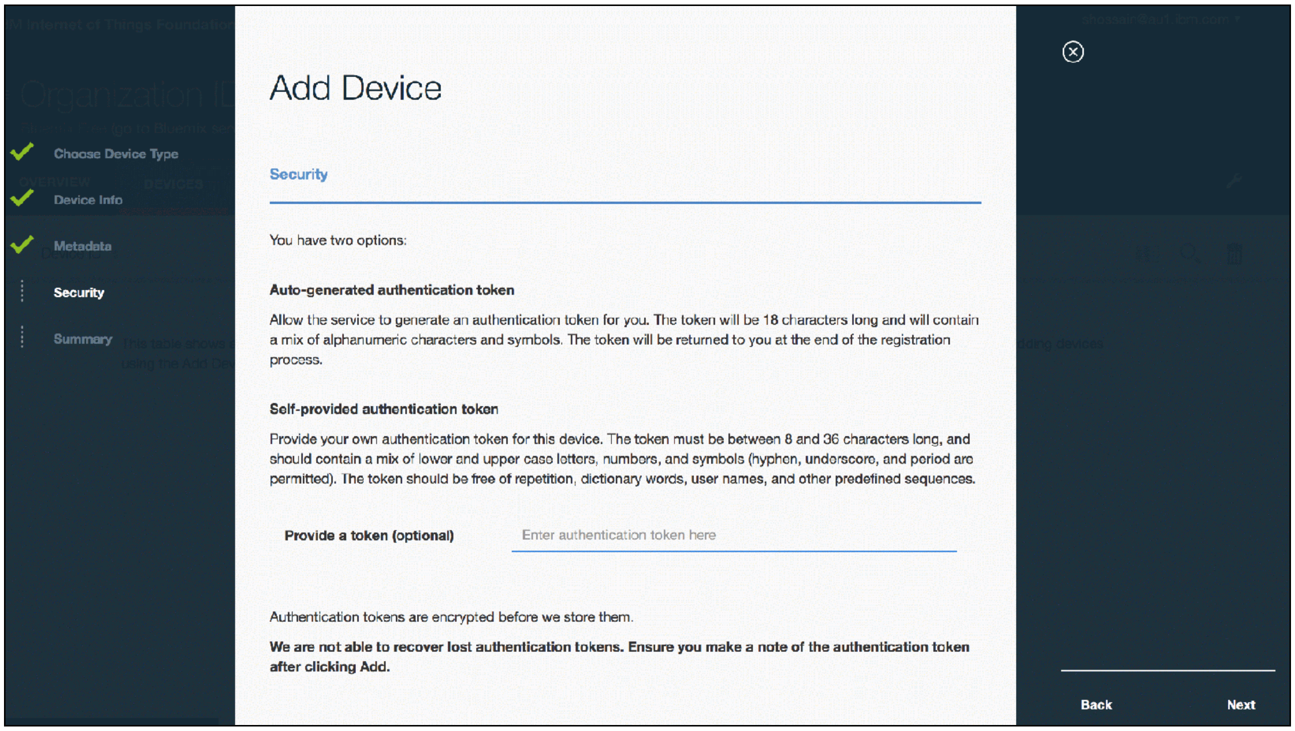
14. The **Metadata** section is optional. Click **Next**. See Figure 8-25.



Metadata must be added as JSON; plain text cannot be used.

Figure 8-25 Adding optional metadata for the device

15. In the Security section, we chose the **Auto-generated authentication token** option. If a token is not provided, this option is chosen automatically. Click **Next**. See Figure 8-26.



Add Device

Security

You have two options:

Auto-generated authentication token

Allow the service to generate an authentication token for you. The token will be 18 characters long and will contain a mix of alphanumeric characters and symbols. The token will be returned to you at the end of the registration process.

Self-provided authentication token

Provide your own authentication token for this device. The token must be between 8 and 36 characters long, and should contain a mix of lower and upper case letters, numbers, and symbols (hyphen, underscore, and period are permitted). The token should be free of repetition, dictionary words, user names, and other predefined sequences.

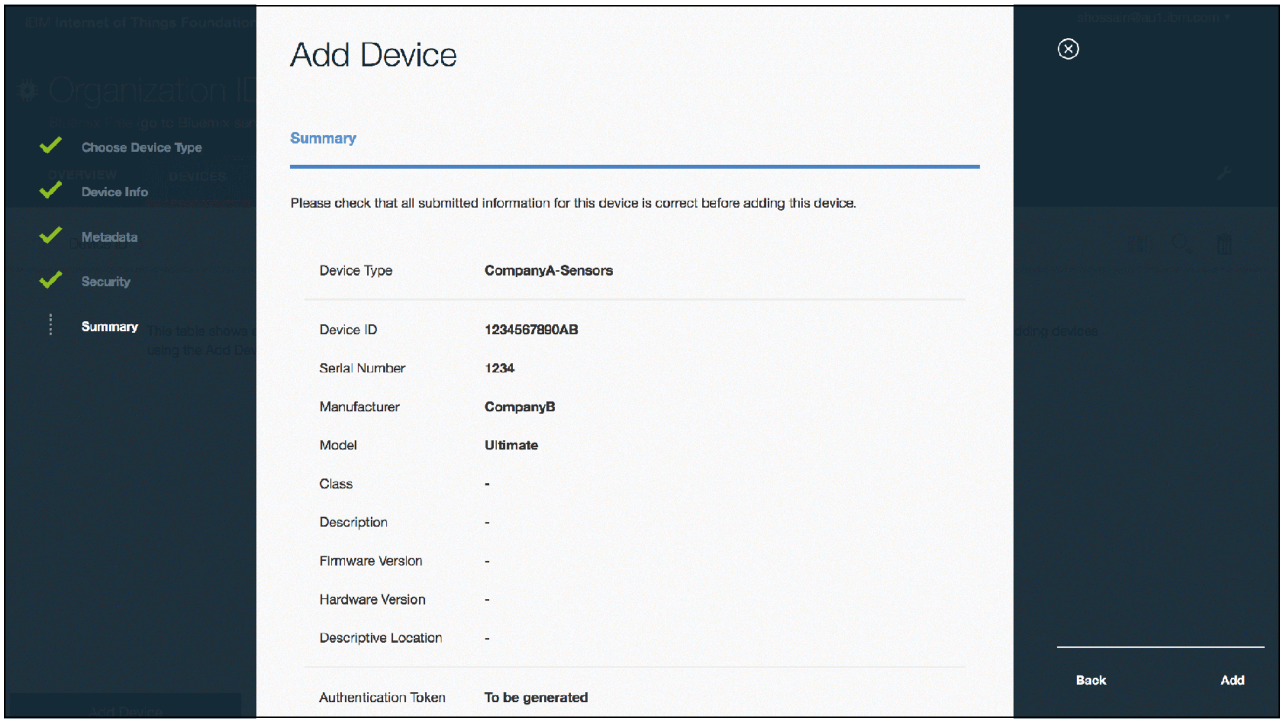
Provide a token (optional) Enter authentication token here

Authentication tokens are encrypted before we store them.

We are not able to recover lost authentication tokens. Ensure you make a note of the authentication token after clicking Add.

Figure 8-26 Generating authentication token for device

16. The Summary section summarizes the information for the device to be added. Click **Add**. See Figure 8-27.



Add Device

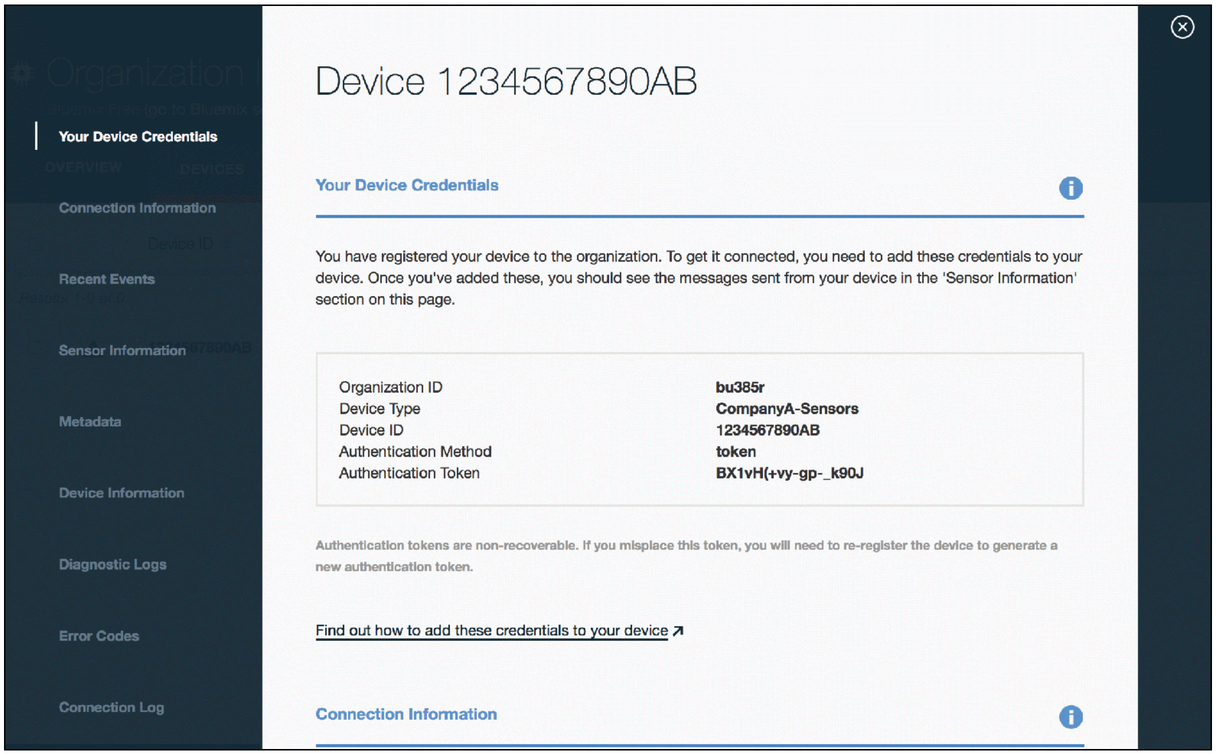
Summary

Please check that all submitted information for this device is correct before adding this device.

Device Type	CompanyA-Sensors
Device ID	1234567890AB
Serial Number	1234
Manufacturer	CompanyB
Model	Ultimate
Class	-
Description	-
Firmware Version	-
Hardware Version	-
Descriptive Location	-
Authentication Token	To be generated

Figure 8-27 Summary of device to be added

17. Now you see a window that is similar to Figure 8-28 showing device credentials. Write down these details in a secure place.



Device 1234567890AB

Your Device Credentials

You have registered your device to the organization. To get it connected, you need to add these credentials to your device. Once you've added these, you should see the messages sent from your device in the 'Sensor Information' section on this page.

Organization ID	bu385r
Device Type	CompanyA-Sensors
Device ID	1234567890AB
Authentication Method	token
Authentication Token	BX1vH(+vy-gp-_k90J

Authentication tokens are non-recoverable. If you misplace this token, you will need to re-register the device to generate a new authentication token.

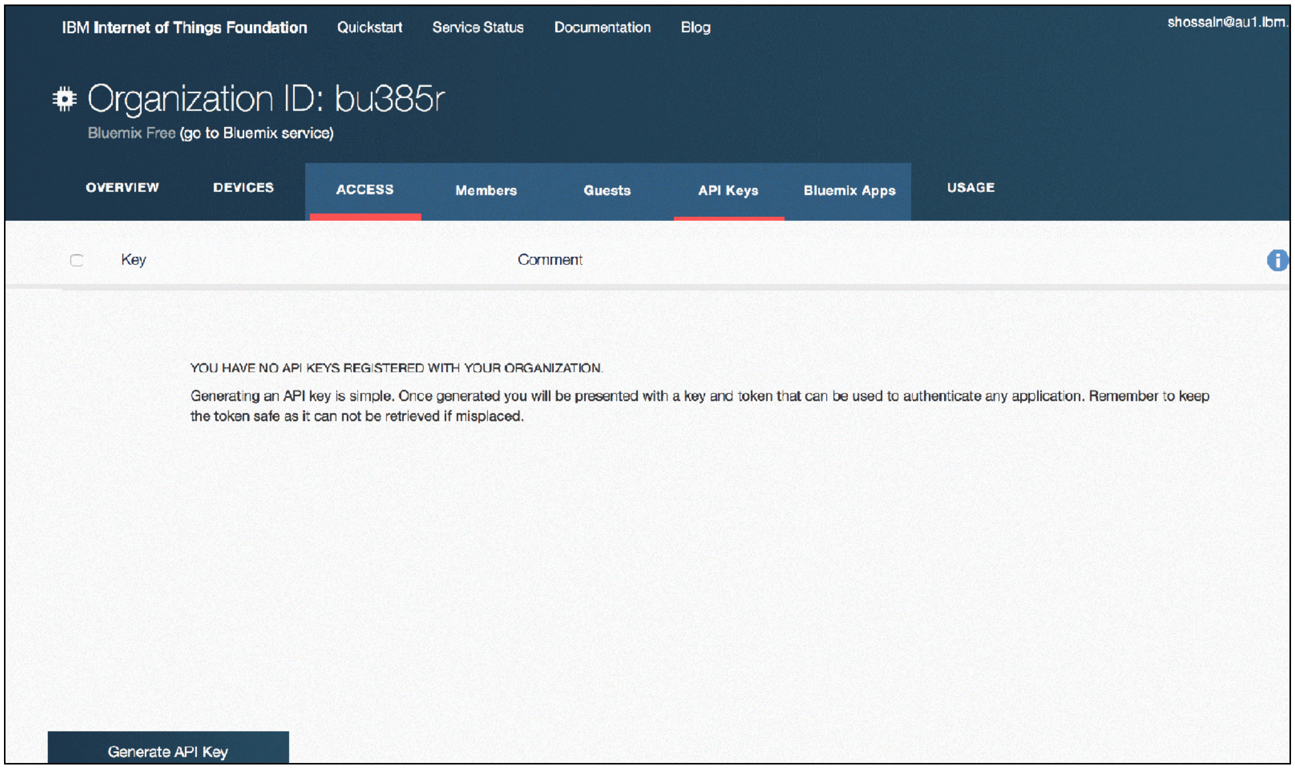
Find out how to add these credentials to your device

Connection Information

Figure 8-28 Device credentials

Note: Authentication tokens are unrecoverable. As such, you need to store these securely. If these are misplaced, you need to reregister the device to generate a new token.

18. Next, we generate an API key from the **ACCESS** tab by clicking the **API Keys** option, as seen in Figure 8-29. Click **Generate API Key**.



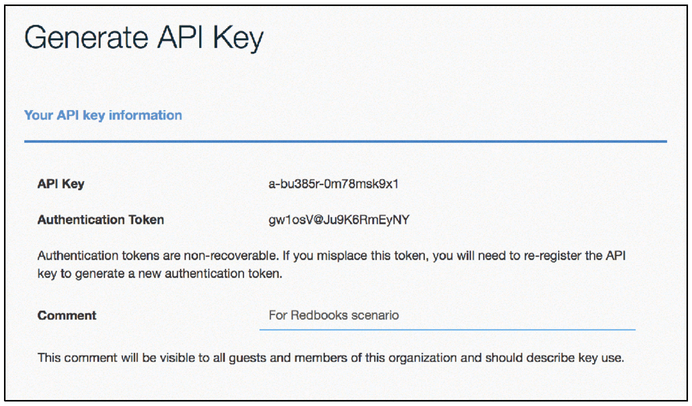
YOU HAVE NO API KEYS REGISTERED WITH YOUR ORGANIZATION.

Generating an API key is simple. Once generated you will be presented with a key and token that can be used to authenticate any application. Remember to keep the token safe as it can not be retrieved if misplaced.

Figure 8-29 Generating API key

19. Write down the API key details from "Your API key information" section as shown in Figure 8-30 and store in a secure place. These details are unrecoverable.

Note: Authentication tokens are unrecoverable. As such, you need to store these securely. If these are misplaced, you need to reregister the device to generate a new token.



Generate API Key

Your API key information

API Key	a-bu385r-0m78msk9x1
Authentication Token	gw1osV@Ju9K6RmEyNY

Authentication tokens are non-recoverable. If you misplace this token, you will need to re-register the API key to generate a new authentication token.

Comment	For Redbooks scenario

This comment will be visible to all guests and members of this organization and should describe key use.

Figure 8-30 API key information

This concludes the setup that is required to register an IoT device.

8.3 Creating the flow in Node-RED

Perform the following steps to create a flow. We gradually complete the flow to add more functionalities and integrate Maximo Asset Management:

1. As shown in Figure 8-4 on page 221, open the welcome page of **iot-redbooks-demo** and click **Go to Node-RED flow editor**. After entering the correct credentials, the flow editor is displayed.

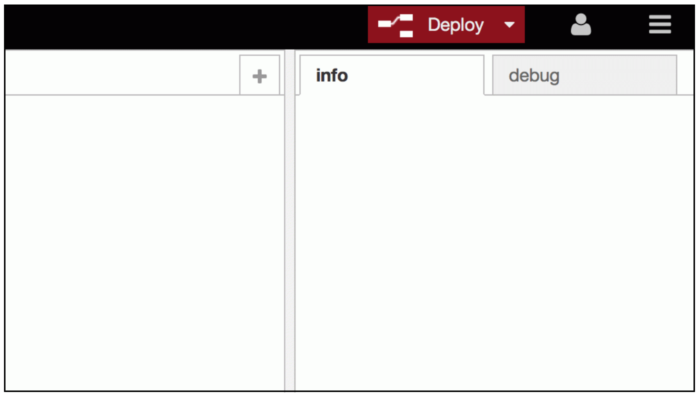
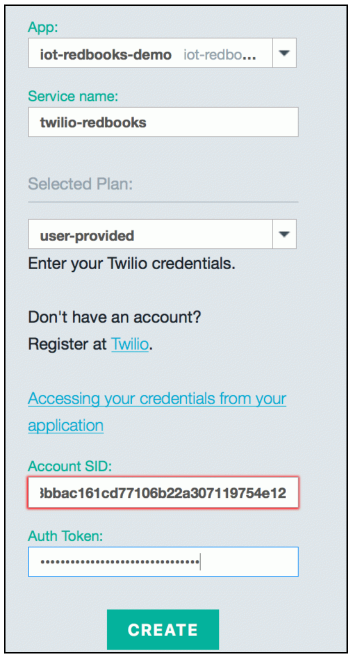


Figure 8-31 Adding a new sheet to Node-RED editor

8.4 Binding Twilio service

Twilio is a third-party service that is provided from Bluemix catalog. By now, we assume that you know how to create or bind a service to your Bluemix application. Follow these high-level steps:

1. Create a trial or paid account of Twilio from `https://www.twilio.com`.
2. Find your AccountSID and AuthToken from the Twilio Account Settings page. Write them down because you need these in the next step.
3. This step is similar to Step 1 in 8.2.1, “Quickstart mode” on page 218. Click **Add a service or API** from the dashboard of iot-redbooks-demo as shown in Figure 8-3 on page 220 and then from Catalog, choose **Twilio**.
4. While creating Twilio service, enter the values you copied earlier in the **Account SID:** and **Auth Token:** fields and click **CREATE**. This is illustrated in Figure 8-32 on page 239.
5. iot-redbooks-demo is restaged for this change to be effective.


App:
iot-redbooks-demo iot-redbo...

Service name:
twilio-redbooks

Selected Plan:
user-provided

Enter your Twilio credentials.

Don't have an account?
Register at Twilio.

Accessing your credentials from your application

Account SID:
3bbac161cd77106b22a307119754e12

Auth Token:
••••••••••••••••••••••••••••••••

Figure 8-32 Adding a Twilio service

8.5 IBM Maximo Asset Management solution

In this section, we provide a brief overview of Maximo Asset Management, its benefits, and how it is set up as an on-premises asset management solution.

8.5.1 What is it?

IBM Maximo Asset Management is Enterprise Asset Management software, which is a comprehensive solution for managing physical assets on a common platform in asset-intensive industries.

Maximo Asset Management allows organizations to share and enforce best practices, inventory, resources, and personnel. It helps manage all types of assets including plant, production, infrastructure, facilities, transportation, and communications. It includes six management modules in an enhanced service-oriented architecture:

- Asset management: Achieve the control that you need to more efficiently track and manage asset and location data throughout the asset lifecycle.
- Work management: Manage both planned and unplanned work activities, from initial request through completion and recording of actuals.
- Service management: Define service offerings, establish service level agreements (SLAs), more proactively monitor service level delivery, and implement escalation procedures.
- Contract management: Gain complete support for purchase, lease, rental, warranty, labor rate, software, master, blanket, and user-defined contracts.

- Inventory management: Know the details of asset-related inventory and its usage including what, when, where, how many, and how valuable.
- Procurement management: Support all the phases of enterprise-wide procurement, such as direct purchasing and inventory replenishment.

Maximo exposes Representational State Transfer (REST) APIs to access its services and data. Many customers are using Maximo to integrate it with on-premises systems of record and for an end-to-end automated flow.

For more information: For more information about IBM Maximo Asset Management, see the product page at the following site:

`http://www.ibm.com/software/products/en/maximoassetmanagement`

8.5.2 Maximo setup for this scenario

In this scenario, IBM Maximo Asset Management 7.6.0.1 was installed, preconfigured, and running on WebSphere Application Server version 8.5 on Microsoft Windows 2012 virtual machine. Organizations and sites were set up for Maximo. Assets related to this scenario were added as well.

8.5.3 Steps to start Maximo

The following steps are provided for references only. Steps will vary depending on your installation. We are using a preconfigured virtual machine with IBM Maximo Asset Management (will be referred as *Maximo* through out this document) software. For ease of operations, scripts have been created to start or stop the Maximo server.

Note: Steps described in this section might vary from what you will need to do to run your own Maximo Asset Management server. Follow the product documentation and instructions that shipped with your product.

Because Maximo is running on WebSphere Application Server, it needs to start the application server first. After the application server is started, Maximo is started and is ready to serve any requests:

1. Start Maximo virtual machine (VM).
2. Locate and double-click the **Maximo 7.6.0.1** shortcut from the Microsoft Windows desktop to open the shortcut to start the Maximo server, as shown in Figure 8-33 on page 241.

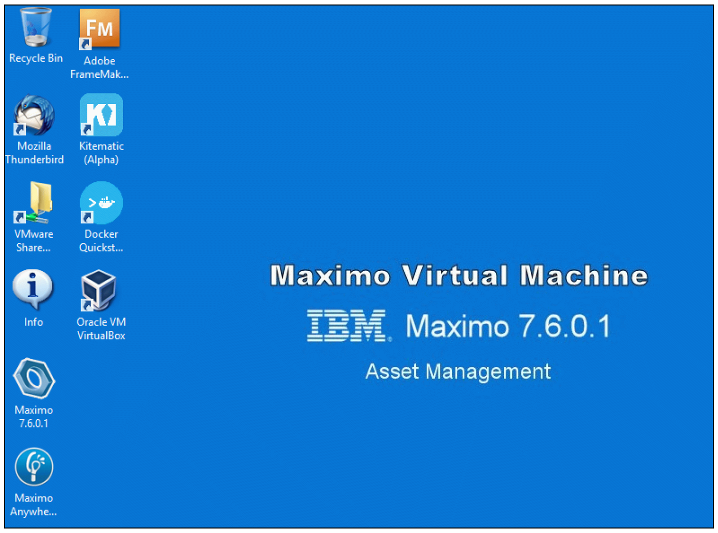
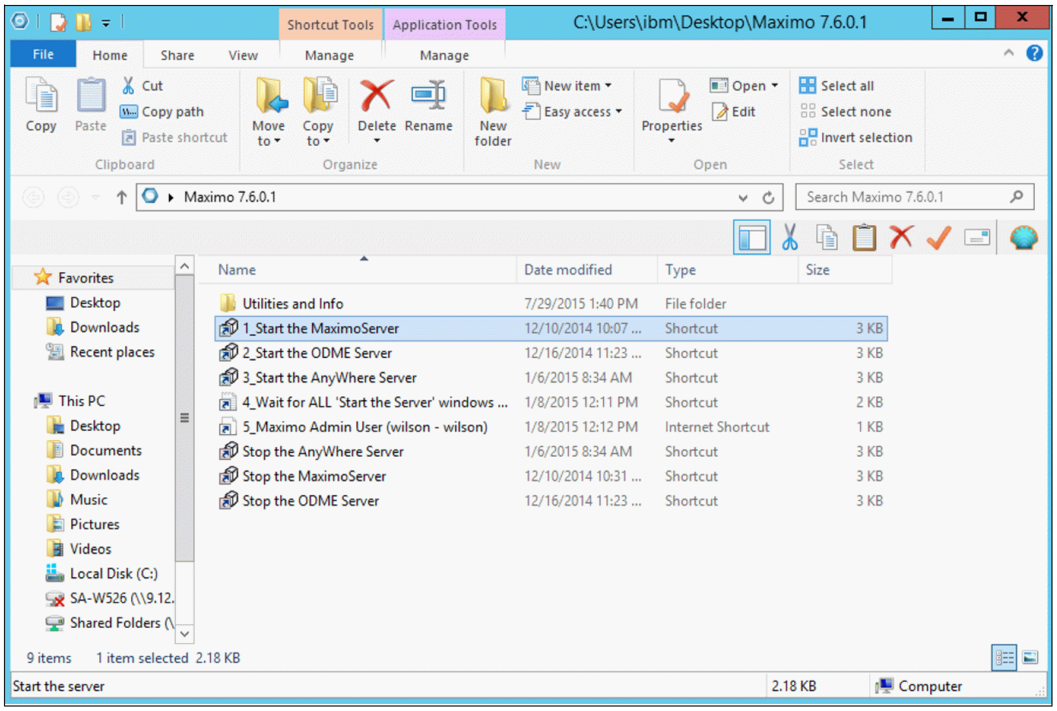


Figure 8-33 Microsoft Windows desktop showing shortcut to Maximo 7.6.0.1 scripts

3. This opens a directory containing Maximo related scripts in Microsoft Windows Explorer, as shown in Figure 8-34.

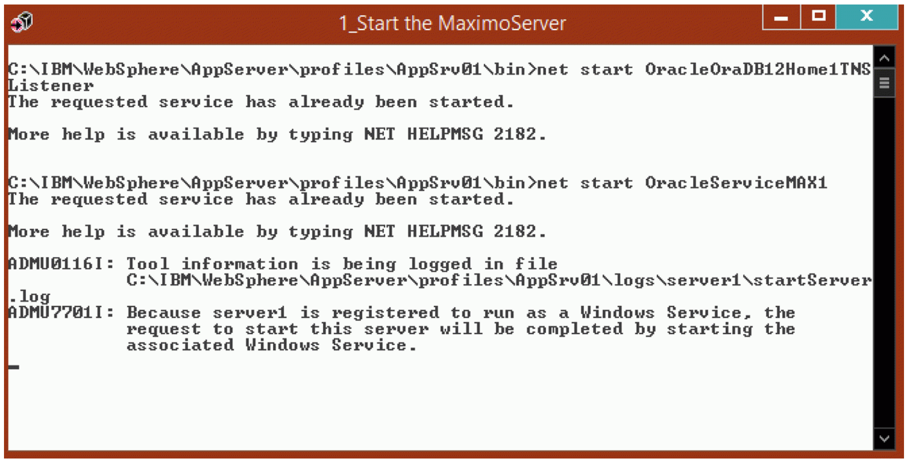


Name	Date modified	Type	Size
Utilities and Info	7/29/2015 1:40 PM	File folder	
1_Start the MaximoServer	12/10/2014 10:07 ...	Shortcut	3 KB
2_Start the ODME Server	12/16/2014 11:23 ...	Shortcut	3 KB
3_Start the AnyWhere Server	1/6/2015 8:34 AM	Shortcut	3 KB
4_Wait for ALL 'Start the Server' windows ...	1/8/2015 12:11 PM	Shortcut	2 KB
5_Maximo Admin User (wilson - wilson)	1/8/2015 12:12 PM	Internet Shortcut	1 KB
Stop the AnyWhere Server	1/6/2015 8:34 AM	Shortcut	3 KB
Stop the MaximoServer	12/10/2014 10:31 ...	Shortcut	3 KB
Stop the ODME Server	12/16/2014 11:23 ...	Shortcut	3 KB

Figure 8-34 Scripts for Maximo

4. Double-click the **1_Start the MaximoServer** shortcut.

5. A command prompt similar to Figure 8-35 opens automatically. This script starts WebSphere Application Server V8.5 and then the Maximo server. The command window is closed automatically when the tasks in the script are completed.

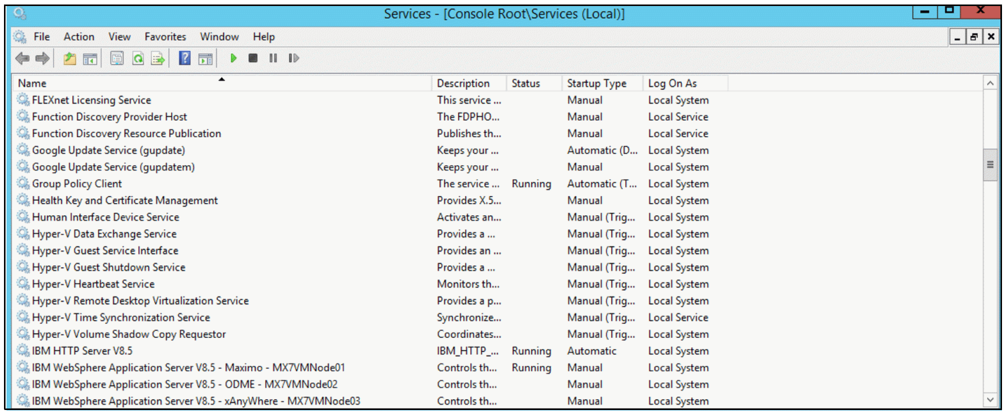


```
C:\IBM\WebSphere\AppServer\profiles\AppSrv01\bin>net start OracleOraDB12Home1TNS
Listener
The requested service has already been started.

More help is available by typing NET HELPMSG 2182.


C:\IBM\WebSphere\AppServer\profiles\AppSrv01\bin>net start OracleServiceMAX1
The requested service has already been started.

More help is available by typing NET HELPMSG 2182.

ADMU0116I: Tool information is being logged in file
           C:\IBM\WebSphere\AppServer\profiles\AppSrv01\logs\server1\startServer
.log
ADMU7701I: Because server1 is registered to run as a Windows Service, the
           request to start this server will be completed by starting the
           associated Windows Service.
_
```

Figure 8-35   Windows command prompt showing the status of Maximo startup

6. Open the services explorer to ensure that the WebSphere Application Server V8.5 service status is *Running*. This indicates that the Maximo server is now running. Refer to Figure 8-36.

> **Note:** If the service status is not *Running*, there is a problem with the startup of the server, which needs to be fixed before proceeding to the next step. These steps are specific to our Maximo setup and installation. For you, it will be different depending on your operating system and installation.



Name	Description	Status	Startup Type	Log On As
FLEXnet Licensing Service	This service ...		Manual	Local System
Function Discovery Provider Host	The FDPHO...		Manual	Local Service
Function Discovery Resource Publication	Publishes th...		Manual	Local Service
Google Update Service (gupdate)	Keeps your ...		Automatic (D...	Local System
Google Update Service (gupdatem)	Keeps your ...		Manual	Local System
Group Policy Client	The service ...	Running	Automatic (T...	Local System
Health Key and Certificate Management	Provides X.5...		Manual	Local System
Human Interface Device Service	Activates an...		Manual (Trig...	Local System
Hyper-V Data Exchange Service	Provides a ...		Manual (Trig...	Local System
Hyper-V Guest Service Interface	Provides an ...		Manual (Trig...	Local System
Hyper-V Guest Shutdown Service	Provides a ...		Manual (Trig...	Local System
Hyper-V Heartbeat Service	Monitors th...		Manual (Trig...	Local System
Hyper-V Remote Desktop Virtualization Service	Provides a p...		Manual (Trig...	Local System
Hyper-V Time Synchronization Service	Synchronize...		Manual (Trig...	Local Service
Hyper-V Volume Shadow Copy Requestor	Coordinates...		Manual (Trig...	Local System
IBM HTTP Server V8.5	IBM_HTTP_...	Running	Automatic	Local System
IBM WebSphere Application Server V8.5 - Maximo - MX7VMNode01	Controls th...	Running	Manual	Local System
IBM WebSphere Application Server V8.5 - ODME - MX7VMNode02	Controls th...		Manual	Local System
IBM WebSphere Application Server V8.5 - xAnyWhere - MX7VMNode03	Controls th...		Manual	Local System

Figure 8-36   Microsoft Windows Services Explorer showing the status of WebSphere Application Server

7. Open a browser window and enter `http://mx7vm/maximo/webclient/login/login.jsp`. The Microsoft Windows virtual machine that we are using also has this URL bookmarked in the browser. We now see a web page that is similar to Figure 8-37.

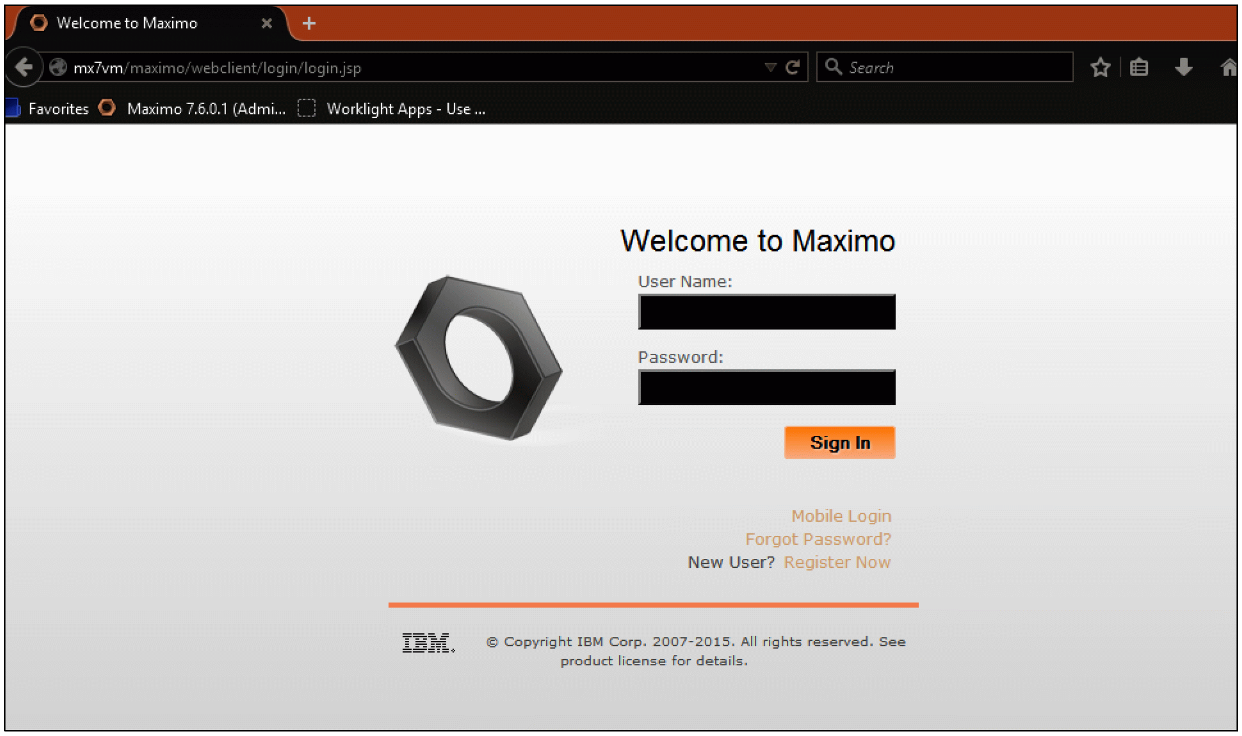
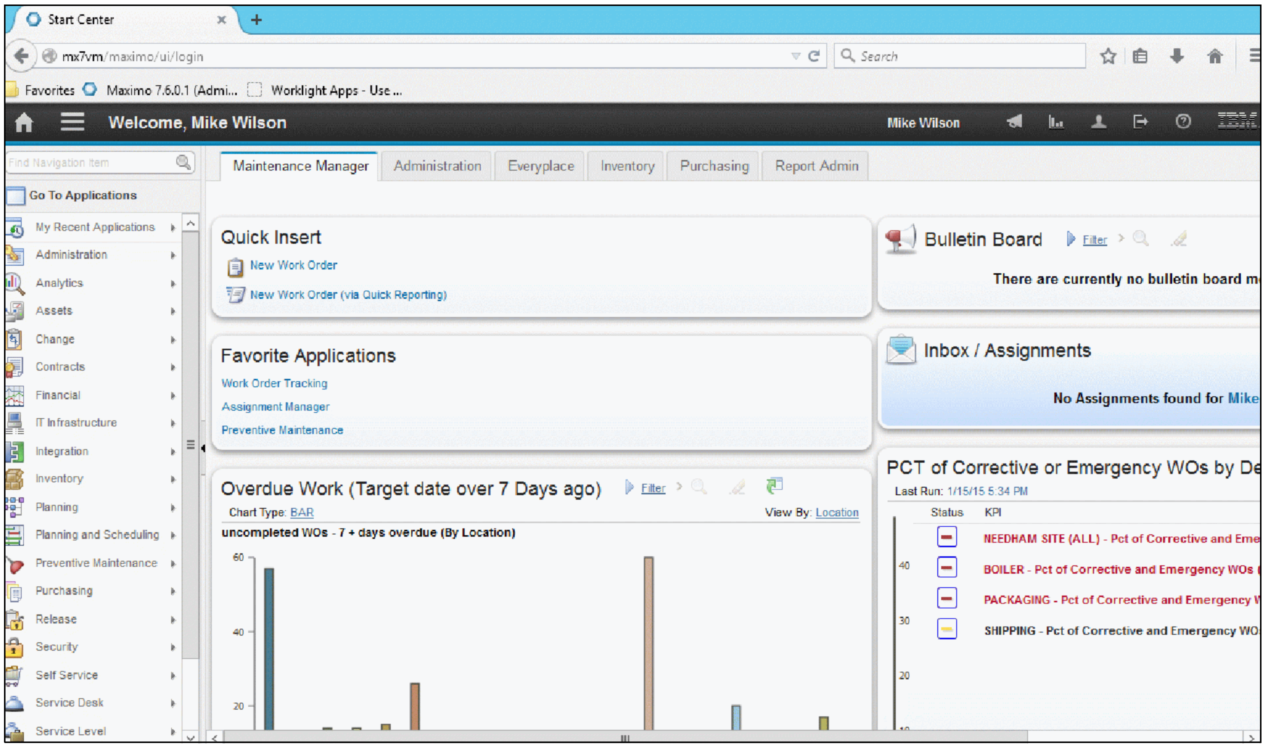


Figure 8-37 Maximo Login page

8. Enter `wilson` for both **User Name:** and **Password:** and click **Sign In**. Upon successful login, you see a page similar to Figure 8-38 on page 244.

Note: Explanation of all the menu items and functionalities of Maximo is outside the scope of this Redbooks publication. For more information, see the product documentation at the following site:

`http://www.ibm.com/support/knowledgecenter/SSLKT6_7.6.0/com.ibm.mam.doc/welcome.html`



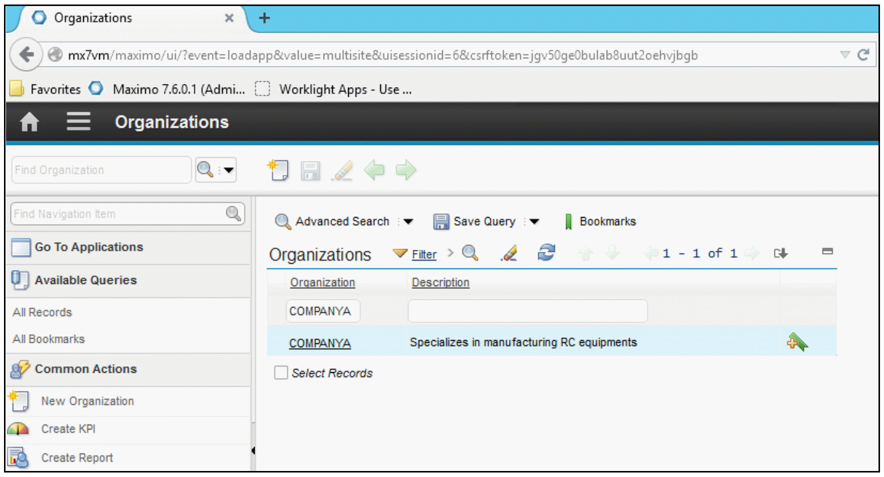
There are currently no bulletin board m

No Assignments found for Mike

NEEDHAM SITE (ALL) - Pct of Corrective and Eme
BOILER - Pct of Corrective and Emergency WOs
PACKAGING - Pct of Corrective and Emergency W
SHIPPING - Pct of Corrective and Emergency WO

Figure 8-38 Maximo main window

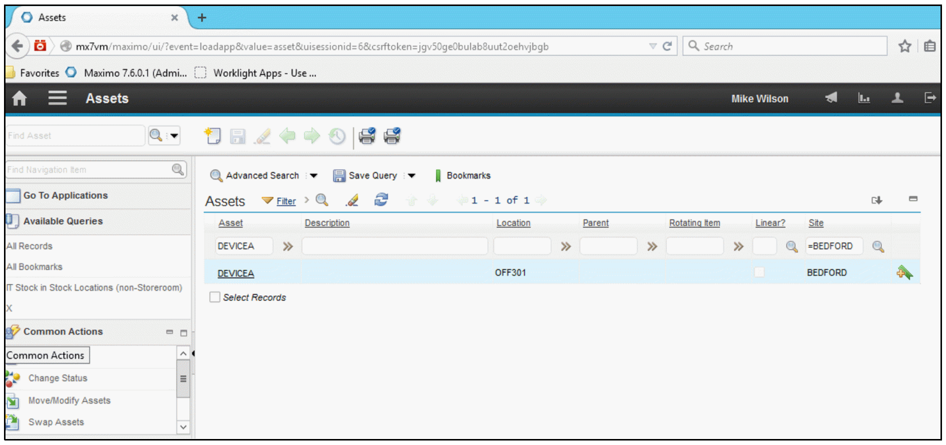
9. For this scenario, we created an organization named `COMPANYA`. This is a fictitious organization who specializes in manufacturing of remote controlled (RC) equipment. Go to the left side menu and click **Administration** → **Organization**. Type `COMPANYA` in the **Organization** text field and press Enter. The resultant window is similar to Figure 8-39.



Organization	Description
COMPANYA	
COMPANYA	Specializes in manufacturing RC equipments

Figure 8-39 Organization COMPANYA

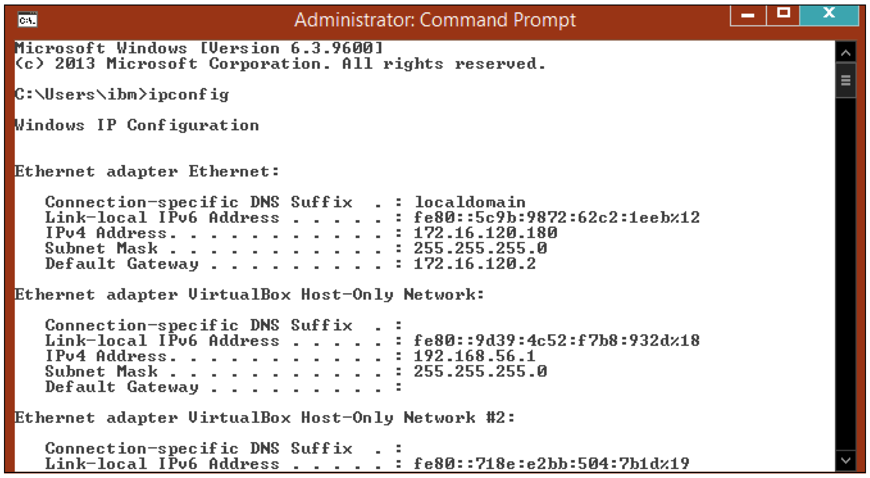
10. For this scenario, we are interested in the asset DEVICEA, which is manufacturing equipment. An IoT device or sensor attached to this asset monitors it and sends an alert to IBM IoTF when things go wrong. Finally, a Maximo work order is created for a support personnel to fix this asset. Now verify the valid asset number, location, and site before proceeding to the next section. From the right side menu, click **Assets** → **Assets**. Then, type `DEVICEA` in the **Asset** text field and press Enter. Refer to Figure 8-40.



Asset	Description	Location	Parent	Rotating Item	Linear?	Site
DEVICEA						=BEDFORD
DEVICEA		OFF301				BEDFORD

Figure 8-40   Maximo asset DEVICEA

11. Next, record the IP address of the virtual machine by opening a command prompt and entering *`ipconfig -all`*. For our case, the IP address is `172.16.123.134`.



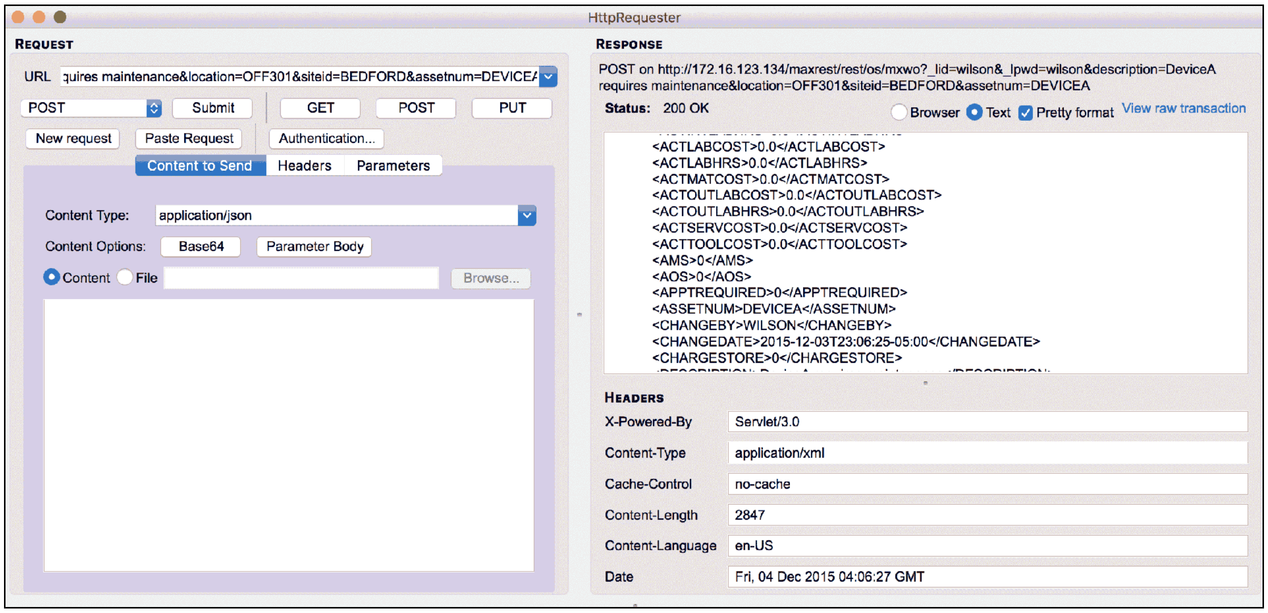
```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\ibm>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . : localdomain
   Link-local IPv6 Address . . . . . : fe80::5c9b:9872:62c2:1eeb%12
   IPv4 Address. . . . . . . . . . . : 172.16.120.180
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 172.16.120.2

Ethernet adapter VirtualBox Host-Only Network:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::9d39:4c52:f7b8:932d%18
   IPv4 Address. . . . . . . . . . . : 192.168.56.1
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :

Ethernet adapter VirtualBox Host-Only Network #2:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::718e:e2bb:504:7b1d%19
```

Figure 8-41   IP address of Maximo virtual machine

12. Use any REST API client to execute the following REST API (POST) call:

```
http://172.16.123.134/maxrest/rest/os/mxwo?_lid=wilson&_lpwd=wilson&description=De
viceA requires maintenance&location=OFF301&siteid=BEDFORD&assetnum=DEVICEA
```

We used the HTTPRequester plug-in for Firefox as shown Figure 8-42.



URL quires maintenance&location=OFF301&siteid=BEDFORD&assetnum=DEVICEA

POST [Submit]

Content Type: application/json

Content Options:

[✓ Content] File

POST on http://172.16.123.134/maxrest/rest/os/mxwo?_lid=wilson&_lpwd=wilson&description=DeviceA requires maintenance&location=OFF301&siteid=BEDFORD&assetnum=DEVICEA

Status: 200 OK

Browser [✓ Text] [✓ Pretty format]

```
<ACTLABCOST>0.0</ACTLABCOST>
<ACTLABHRS>0.0</ACTLABHRS>
<ACTMATCOST>0.0</ACTMATCOST>
<ACTOUTLABCOST>0.0</ACTOUTLABCOST>
<ACTOUTLABHRS>0.0</ACTOUTLABHRS>
<ACTSERVCOST>0.0</ACTSERVCOST>
<ACTTOOLCOST>0.0</ACTTOOLCOST>
<AMS>0</AMS>
<AOS>0</AOS>
<APPTREQUIRED>0</APPTREQUIRED>
<ASSETNUM>DEVICEA</ASSETNUM>
<CHANGEBY>WILSON</CHANGEBY>
<CHANGEDATE>2015-12-03T23:06:25-05:00</CHANGEDATE>
<CHARGESTORE>0</CHARGESTORE>
```

HEADERS

X-Powered-By	Servlet/3.0
Content-Type	application/xml
Cache-Control	no-cache
Content-Length	2847
Content-Language	en-US
Date	Fri, 04 Dec 2015 04:06:27 GMT

Figure 8-42 HTTPRequester to invoke POST API call

The output is similar to Example 8-3.

Example 8-3 Output from Maximo REST API call

```
<?xml version="1.0" encoding="UTF-8"?>
<CreateMXWOResponse xmlns="http://www.ibm.com/maximo"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
creationDateTime="2015-12-03T23:22:43-05:00" transLanguage="EN" baseLanguage="EN"
messageID="1828231.1449202963781107 57" maximoVersion="7 6 20150611-1135 V7601-65">
    <MXWOSet>
        <WORKORDER rowstamp="2868552">
            <ACTINTLABCOST>0.0</ACTINTLABCOST>
            <ACTINTLABHRS>0.0</ACTINTLABHRS>
            <ACTLABCOST>0.0</ACTLABCOST>
            <ACTLABHRS>0.0</ACTLABHRS>
            <ACTMATCOST>0.0</ACTMATCOST>
            <ACTOUTLABCOST>0.0</ACTOUTLABCOST>
            <ACTOUTLABHRS>0.0</ACTOUTLABHRS>
            <ACTSERVCOST>0.0</ACTSERVCOST>
            <ACTTOOLCOST>0.0</ACTTOOLCOST>
            <AMS>0</AMS>
            <AOS>0</AOS>
            <APPTREQUIRED>0</APPTREQUIRED>
            <ASSETNUM>DEVICEA</ASSETNUM>
            <CHANGEBY>WILSON</CHANGEBY>
            <CHANGEDATE>2015-12-03T23:22:42-05:00</CHANGEDATE>
            <CHARGESTORE>0</CHARGESTORE>
            <DESCRIPTION>DeviceA requires maintenance</DESCRIPTION>
            <DISABLED>0</DISABLED>
            <DOWNTIME>0</DOWNTIME>
            <ESTATAPPRINTLABCOST>0.0</ESTATAPPRINTLABCOST>
            <ESTATAPPRINTLABHRS>0.0</ESTATAPPRINTLABHRS>
            <ESTATAPPRLABCOST>0.0</ESTATAPPRLABCOST>
```

```
<ESTATAPPRLABHRS>0.0</ESTATAPPRLABHRS>
<ESTATAPPRMATCOST>0.0</ESTATAPPRMATCOST>
<ESTATAPPROUTLABCOST>0.0</ESTATAPPROUTLABCOST>
<ESTATAPPROUTLABHRS>0.0</ESTATAPPROUTLABHRS>
<ESTATAPPRSERVCOST>0.0</ESTATAPPRSERVCOST>
<ESTATAPPRTOOLCOST>0.0</ESTATAPPRTOOLCOST>
<ESTDUR>0.0</ESTDUR>
<ESTINTLABCOST>0.0</ESTINTLABCOST>
<ESTINTLABHRS>0.0</ESTINTLABHRS>
<ESTLABCOST>0.0</ESTLABCOST>
<ESTLABHRS>0.0</ESTLABHRS>
<ESTMATCOST>0.0</ESTMATCOST>
<ESTOUTLABCOST>0.0</ESTOUTLABCOST>
<ESTOUTLABHRS>0.0</ESTOUTLABHRS>
<ESTSERVCOST>0.0</ESTSERVCOST>
<ESTTOOLCOST>0.0</ESTTOOLCOST>
<FAILURECODE>BLDGS</FAILURECODE>
<FLOWACTIONASSIST>0</FLOWACTIONASSIST>
<FLOWCONTROLLED>0</FLOWCONTROLLED>
<GLACCOUNT>
    <VALUE>6220-300-???</VALUE>
</GLACCOUNT>
<HASCHILDREN>0</HASCHILDREN>
<HASFOLLOWUPWORK>0</HASFOLLOWUPWORK>
<HISTORYFLAG>0</HISTORYFLAG>
<IGNOREDIAVAIL>0</IGNOREDIAVAIL>
<IGNORESRMAVAIL>1</IGNORESRMAVAIL>
<INCTASKSINSCHED>1</INCTASKSINSCHED>
<INTERRUPTIBLE>0</INTERRUPTIBLE>
<ISTASK>0</ISTASK>
<LMS>0</LMS>
<LOCATION>OFF301</LOCATION>
<LOS>0</LOS>
<NESTEDJPINPROCESS>0</NESTEDJPINPROCESS>
<NEWCHILDCLASS>WORKORDER</NEWCHILDCLASS>
<ORGID>EAGLENA</ORGID>
<OUTLABCOST>0.0</OUTLABCOST>
<OUTMATCOST>0.0</OUTMATCOST>
<OUTTOOLCOST>0.0</OUTTOOLCOST>
<PARENTCHGSSTATUS>1</PARENTCHGSSTATUS>
<PHONE>(617) 555-9017</PHONE>
<PLUSCISMOBILE>0</PLUSCISMOBILE>
<PLUSCLOOP>0</PLUSCLOOP>
<REPAIRLOCFLAG>0</REPAIRLOCFLAG>
<REPORTDATE>2015-12-03T23:22:42-05:00</REPORTDATE>
<REPORTEDBY>WILSON</REPORTEDBY>
<REQASSTDWNTIME>0</REQASSTDWNTIME>
<SITEID>BEDFORD</SITEID>
<STATUS>WAPPR</STATUS>
<STATUSDATE>2015-12-03T23:22:42-05:00</STATUSDATE>
<STATUSIFACE>0</STATUSIFACE>
<SUSPENDFLOW>0</SUSPENDFLOW>
<WOACCEPTSCHARGES>1</WOACCEPTSCHARGES>
<WOCLASS>WORKORDER</WOCLASS>
<WOGROUP>1335</WOGROUP>
```

```
            <WOISSWAP>0</WOISSWAP>
            <WONUM>1335</WONUM>
            <WORKORDERID>2997</WORKORDERID>
        </WORKORDER>
    </MXWOSet>
</CreateMXWOResponse>
```

Now IBM Maximo Asset Management setup is complete and verified. In this scenario, it is acting as an on-premises system of record (SOR). Next, we create a Secure Gateway connection so that the Node-RED application on Bluemix can access this.

8.6 Integrating IoT application with IBM Asset Management system with Bluemix Secure Gateway service

This section presents the tasks to set up a secure tunnel between Bluemix and Maximo by using Secure Gateway service from Bluemix.

8.6.1 Setting up Secure Gateway

To connect a Bluemix application to on-premises Maximo, Secure Gateway service from Bluemix is required.

Secure Gateway service has two parts, which are service configuration on Bluemix and Secure Gateway client. In this scenario, the Secure Gateway client is running as a Docker image. Perform the following steps to create and configure Secure Gateway:

1. Go to **Bluemix Catalog** → **Integration** → **Secure Gateway**.

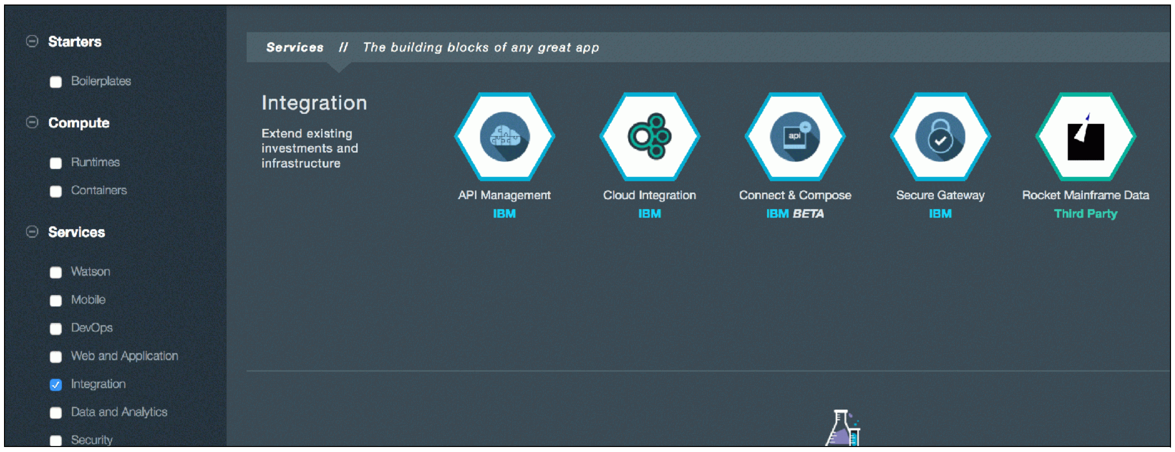
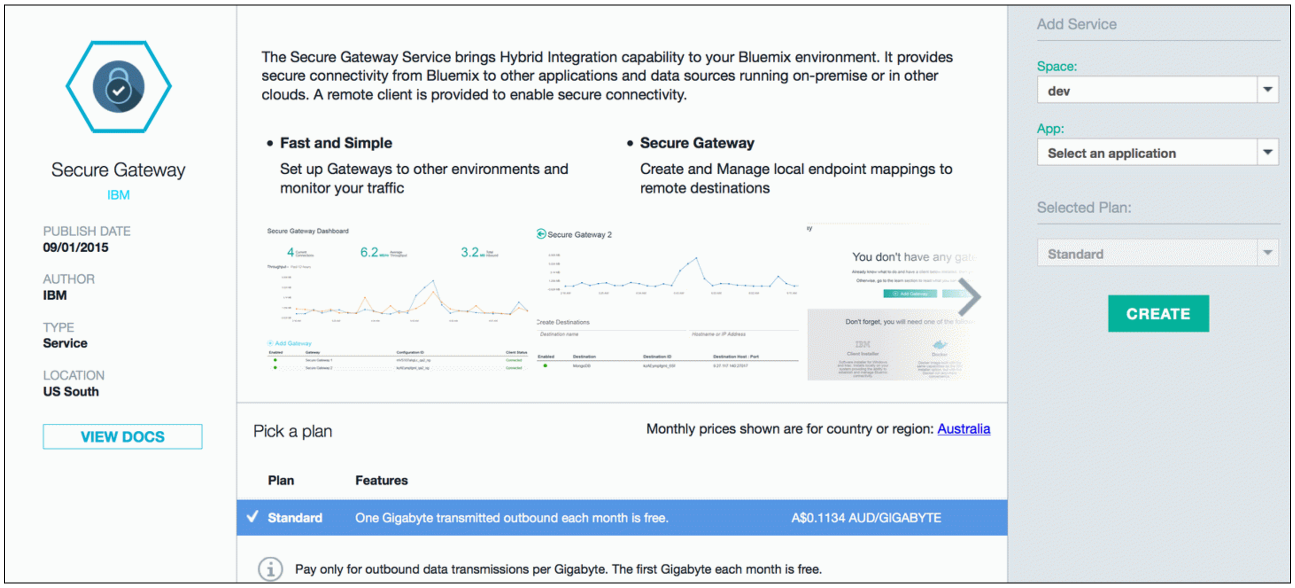


Figure 8-43 Secure Gateway in Bluemix catalog

2. Click **CREATE** to create Secure Gateway service, as shown in Figure 8-44.



Secure Gateway
IBM

PUBLISH DATE
09/01/2015

AUTHOR
IBM

TYPE
Service

LOCATION
US South

VIEW DOCS

The Secure Gateway Service brings Hybrid Integration capability to your Bluemix environment. It provides secure connectivity from Bluemix to other applications and data sources running on-premise or in other clouds. A remote client is provided to enable secure connectivity.

- **Fast and Simple**
  Set up Gateways to other environments and monitor your traffic
- **Secure Gateway**
  Create and Manage local endpoint mappings to remote destinations

Pick a plan

Monthly prices shown are for country or region: Australia

Plan	Features	
✓ Standard	One Gigabyte transmitted outbound each month is free.	A$0.1134 AUD/GIGABYTE

Pay only for outbound data transmissions per Gigabyte. The first Gigabyte each month is free.

Add Service

Space: dev

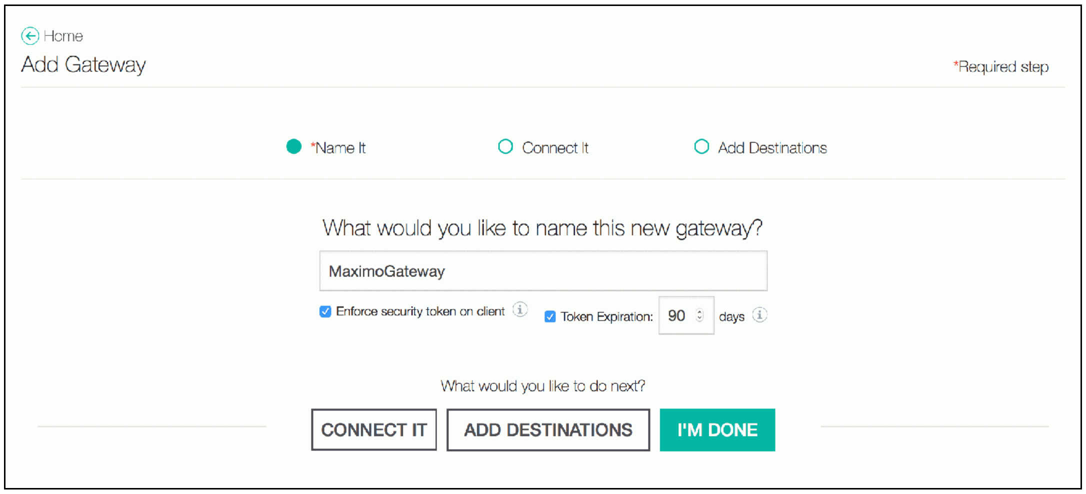
App: Select an application

Selected Plan: Standard

CREATE

Figure 8-44 Creating a Secure Gateway service

3. Enter **MaximoGateway** as the gateway name and click **CONNECT IT**. This is shown is Figure 8-45.



Home
Add Gateway

*Required step

● *Name It ○ Connect It ○ Add Destinations

What would you like to name this new gateway?

MaximoGateway

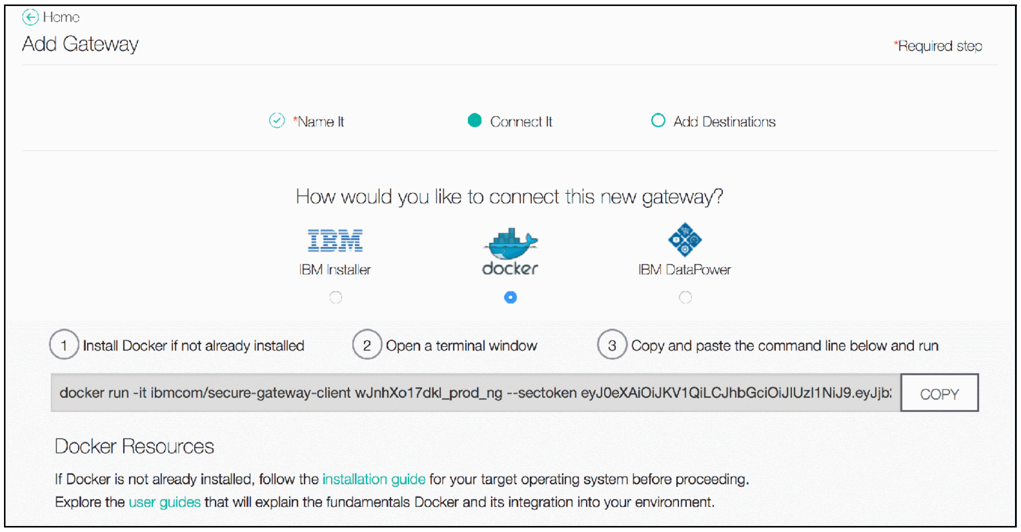
☑ Enforce security token on client ☑ Token Expiration: 90 days

What would you like to do next?

CONNECT IT ADD DESTINATIONS I'M DONE

Figure 8-45 Creating a tunnel or gateway

4. Select **docker** as the gateway connection option. Click **Copy** to copy the command that contains the Secure Gateway ID and security token for pasting into a terminal or command prompt later. Refer to Figure 8-46.

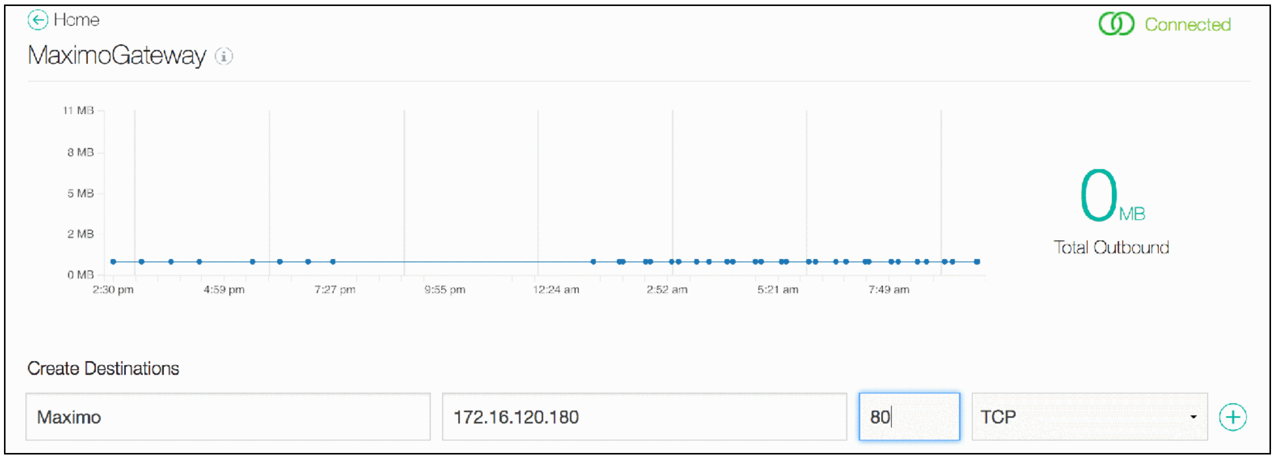


```
docker run -it ibmcom/secure-gateway-client wJnhXo17dkl_prod_ng --sectoken eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJjb
```

If Docker is not already installed, follow the installation guide for your target operating system before proceeding.
Explore the user guides that will explain the fundamentals Docker and its integration into your environment.

Figure 8-46 Secure Gateway connection options

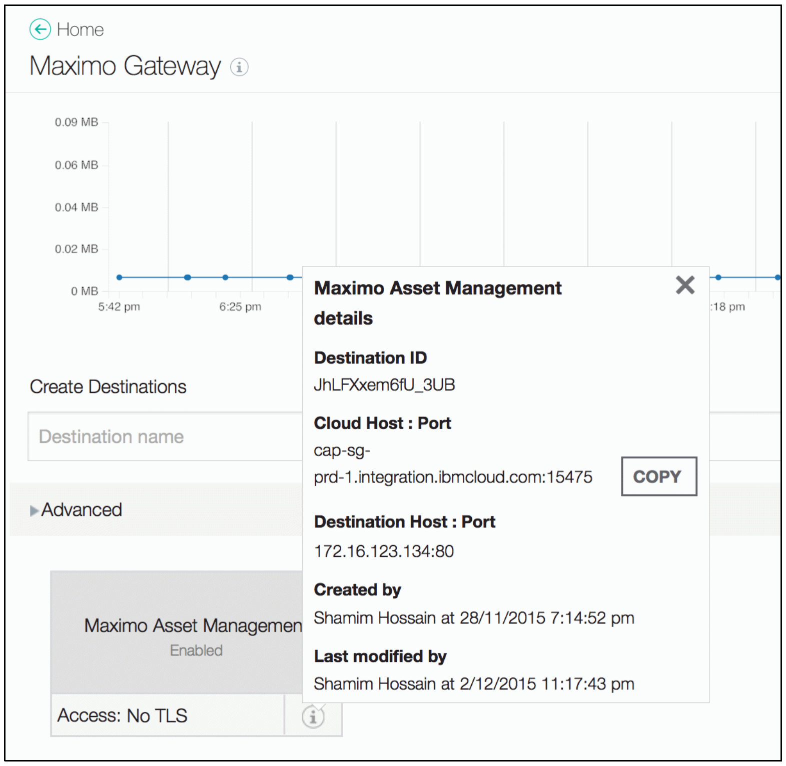
5. As noted earlier, the Maximo server IP address is 172.16.123.134 and its RESTful services are exposed at port 80. Create the destination by completing the destination name, IP address, port, and **TCP** as protocol, then click **+** as shown in Figure 8-47.



Maximo	172.16.120.180	80	TCP

Figure 8-47 Adding a destination

6. Finally, click **i** to reveal connection details as shown in Figure 8-48.



Maximo Asset Management details

Destination ID
JhLFXxem6fU_3UB

Cloud Host : Port
cap-sg-prd-1.integration.ibmcloud.com:15475

Destination Host : Port
172.16.123.134:80

Created by
Shamim Hossain at 28/11/2015 7:14:52 pm

Last modified by
Shamim Hossain at 2/12/2015 11:17:43 pm

Maximo Asset Managemen
Enabled
Access: No TLS

Figure 8-48 Summary of connection details

8.7 The complete solution for the scenario

Here is the complete Node-RED solution, which corresponds to Figure 8-49.

The workflow consists of the following steps:

1. The Node-RED app receives an MQTT message from a simulated IoT sensor.
2. This message in JSON format is parsed by the *Assign* node as shown in Figure 8-49.
3. The *switch* node determines whether maintenance is required or not.
4. If maintenance is required, a Maximo Work Order is created by using the HTTP Request function from Node-RED. As shown in the figure below, the *Request to on-prem Maximo* node will do this.
5. Finally, an SMS is sent to the support personnel using Twilio service.

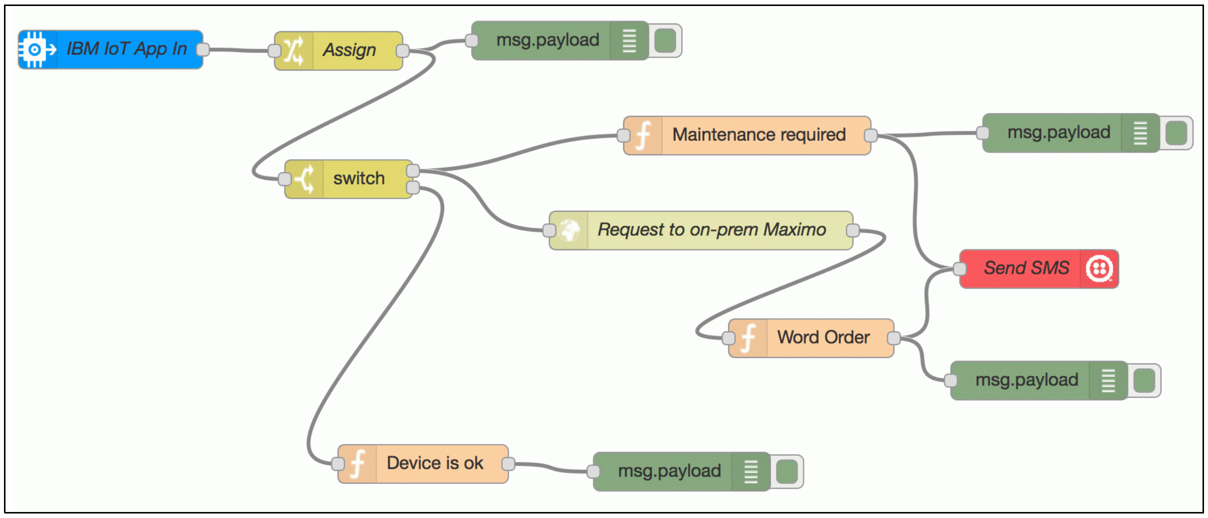


Figure 8-49 Complete Node-RED flow

9

Demonstration of analytics and real-time event detection

This chapter begins by describing the extraction of csv data into the IBM Bluemix dashDB service. This data is then prepared for analysis by using the built-in R analytics capability available in IBM dashDB. From this data, a stylized statistical model is built.

The model developed from the historical data in dash DB is then configured in a Bluemix InfoSphere Streams service. When the model is configured, a real-time data source is specified, and by using the model predictions, criteria are developed for event detection.

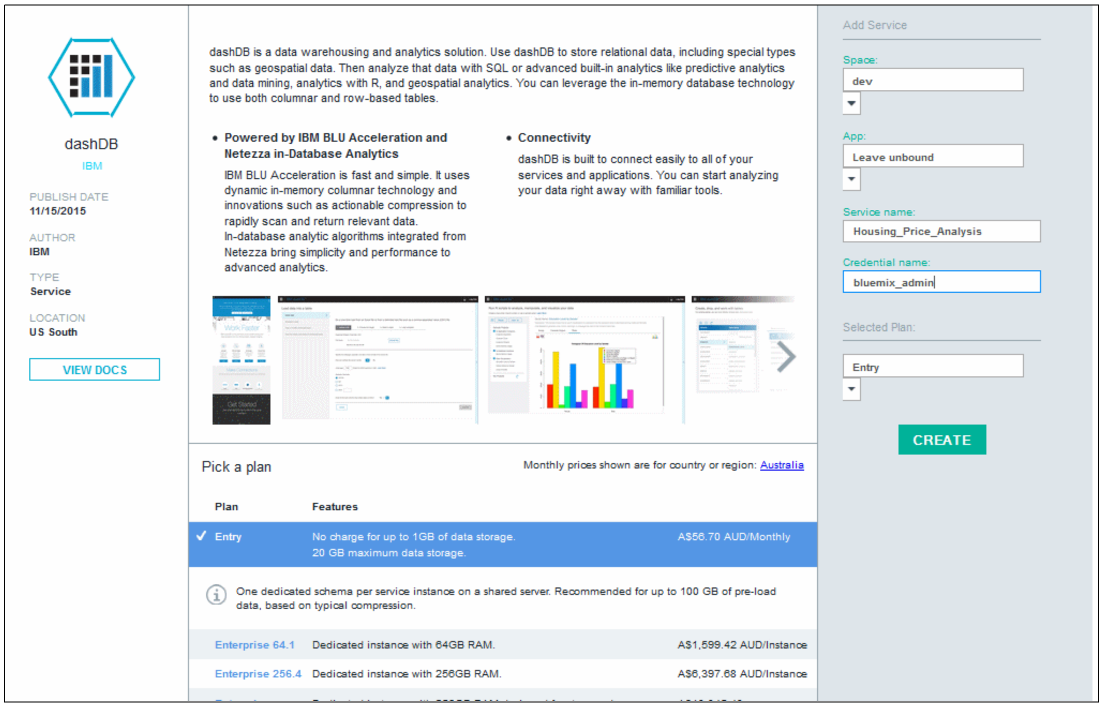
The particular scenario that is considered is the average house price in Victoria, Australia predicted by the Reserve Bank of Australia (RBA) cash rate. The model developed using R in dashDB is then created in InfoSphere Streams, and by using a simulated real-time source of the RBA cash rate, a real-time event is triggered based on some criteria.

This chapter has the following sections:

9.1 Configuring a Bluemix dashDB service

To begin, a dashDB service on Bluemix needs to be configured. The dashDB service can be found in the Bluemix Catalog under the Data and Analytics section.

After selecting the dashDB service, a more detailed description of the dashDB service comes up with some options for configuring the service on the right, as shown in Figure 9-1.


dashDB
IBM

PUBLISH DATE
11/15/2015

AUTHOR
IBM

TYPE
Service

LOCATION
US South

VIEW DOCS

dashDB is a data warehousing and analytics solution. Use dashDB to store relational data, including special types such as geospatial data. Then analyze that data with SQL or advanced built-in analytics like predictive analytics and data mining, analytics with R, and geospatial analytics. You can leverage the in-memory database technology to use both columnar and row-based tables.

- **Powered by IBM BLU Acceleration and Netezza in-Database Analytics**

  IBM BLU Acceleration is fast and simple. It uses dynamic in-memory columnar technology and innovations such as actionable compression to rapidly scan and return relevant data.
  In-database analytic algorithms integrated from Netezza bring simplicity and performance to advanced analytics.

- **Connectivity**

  dashDB is built to connect easily to all of your services and applications. You can start analyzing your data right away with familiar tools.

Pick a plan

Monthly prices shown are for country or region: Australia

Plan	Features	
✓ Entry	No charge for up to 1GB of data storage. 20 GB maximum data storage.	A$56.70 AUD/Monthly
	One dedicated schema per service instance on a shared server. Recommended for up to 100 GB of pre-load data, based on typical compression.	
Enterprise 64.1	Dedicated instance with 64GB RAM.	A$1,599.42 AUD/Instance
Enterprise 256.4	Dedicated instance with 256GB RAM.	A$6,397.68 AUD/Instance

Add Service

Space:
dev

App:
Leave unbound

Service name:
Housing_Price_Analysis

Credential name:
bluemix_admin

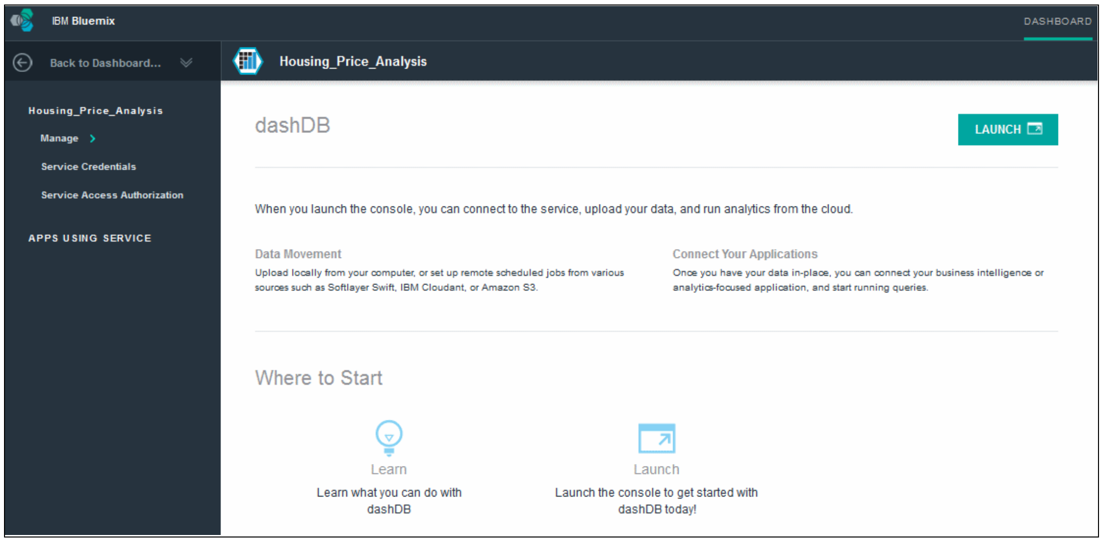
Selected Plan:
Entry

CREATE

Figure 9-1 The dashDB configure page

For this particular scenario, the service is not bound to an application. The service name is used to describe the purpose of the service. In this case, it is named ***Housing_Price_Analysis***, and the credentials name is used to specify a username, in this case ***bluemix_admin***.

The type of plan used is dependent on the size of the data set that needs analysis. In this example, the data set is small so the entry-level plan is used. After configuring all the setting, the service is created. After clicking **Create**, a page is displayed that is similar to Figure 9-2 on page 255.



When you launch the console, you can connect to the service, upload your data, and run analytics from the cloud.

Data Movement
Upload locally from your computer, or set up remote scheduled jobs from various sources such as Softlayer Swift, IBM Cloudant, or Amazon S3.

Connect Your Applications
Once you have your data in-place, you can connect your business intelligence or analytics-focused application, and start running queries.

Where to Start

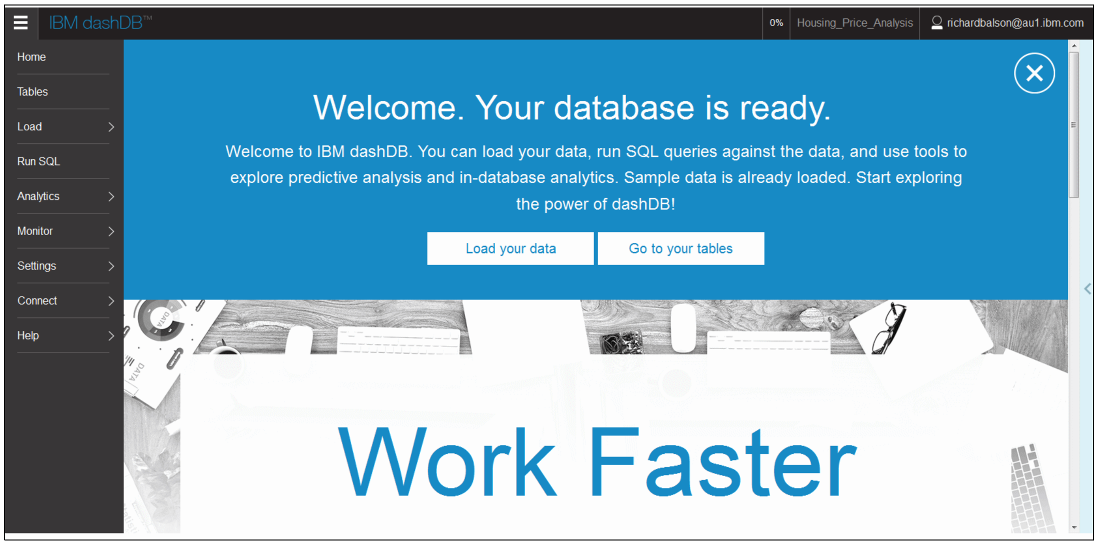
Learn
Learn what you can do with dashDB

Launch
Launch the console to get started with dashDB today!

Figure 9-2 The dashDB launch window

From this window, you can launch your dashDB service, or learn more about dashDB through the help files. On the left there are three options: Manage, Service Credentials, and Service Access Authorization. The page shown in Figure 9-3 is the manage page. In the service credentials page, you can manage who has access to the service, and in the service access authorization page, authorization can be provided for other Bluemix services to access the dashDB service configured. For this particular scenario, the application is launched as is because there is no need to integrate it with other services.

When the dashDB service is launched, a page similar to that shown in Figure 9-3 is displayed. In the next section, preparing and importing of data into the dashDB service is discussed.



Welcome. Your database is ready.

Welcome to IBM dashDB. You can load your data, run SQL queries against the data, and use tools to explore predictive analysis and in-database analytics. Sample data is already loaded. Start exploring the power of dashDB!

Figure 9-3 The dashDB service start page

9.2 Preparing and importing data into dashDB

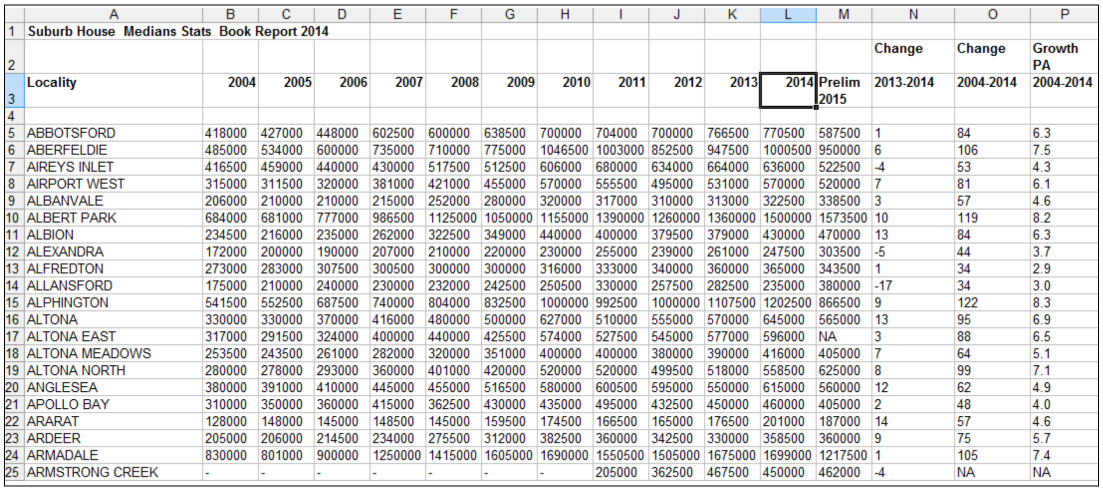
This section is split into three sub-sections. First, the data sources used for this application are described. Second, the preprocessing of the raw data is described. Third, importing the data into dashDB is explained.

9.2.1 Data sources

The first data set used is publicly available from the Victorian State Government of Australia. The data set includes the average yearly house price per suburb in Victoria, and can be found at the following site:

http://www.dtpli.vic.gov.au/property-and-land-titles/property-information/property-prices

The data is in xls format and is demonstrated in Figure 9-4.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Suburb House Medians Stats Book Report 2014															
2														Change	Change	Growth PA
3	Locality	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	Prelim 2015	2013-2014	2004-2014	2004-2014
4																
5	ABBOTSFORD	418000	427000	448000	602500	600000	638500	700000	704000	700000	766500	770500	587500	1	84	6.3
6	ABERFELDIE	485000	534000	600000	735000	710000	775000	1046500	1003000	852500	947500	1000500	950000	6	106	7.5
7	AIREYS INLET	416500	459000	440000	430000	517500	512500	606000	680000	634000	664000	636000	522500	-4	53	4.3
8	AIRPORT WEST	315000	311500	320000	381000	421000	455000	570000	555500	495000	531000	570000	520000	7	81	6.1
9	ALBANVALE	206000	210000	210000	215000	252000	280000	320000	317000	310000	313000	322500	338500	3	57	4.6
10	ALBERT PARK	684000	681000	777000	986500	1125000	1050000	1155000	1390000	1260000	1360000	1500000	1573500	10	119	8.2
11	ALBION	234500	216000	235000	262000	322500	349000	440000	400000	379500	379000	430000	470000	13	84	6.3
12	ALEXANDRA	172000	200000	190000	207000	210000	220000	230000	255000	239000	261000	247500	303500	-5	44	3.7
13	ALFREDTON	273000	283000	307500	300500	300000	300000	316000	333000	340000	360000	365000	343500	1	34	2.9
14	ALLANSFORD	175000	210000	240000	230000	232000	242500	250500	330000	257500	282500	235000	380000	-17	34	3.0
15	ALPHINGTON	541500	552500	687500	740000	804000	832500	1000000	992500	1000000	1107500	1202500	866500	9	122	8.3
16	ALTONA	330000	330000	370000	416000	480000	500000	627000	510000	555000	570000	645000	565000	13	95	6.9
17	ALTONA EAST	317000	291500	324000	400000	440000	425500	574000	527500	545000	577000	596000	NA	3	88	6.5
18	ALTONA MEADOWS	253500	243500	261000	282000	320000	351000	400000	400000	380000	390000	416000	405000	7	64	5.1
19	ALTONA NORTH	280000	278000	293000	360000	401000	420000	520000	520000	499500	518000	558500	625000	8	99	7.1
20	ANGLESEA	380000	391000	410000	445000	455000	516500	580000	600500	595000	550000	615000	560000	12	62	4.9
21	APOLLO BAY	310000	350000	360000	415000	362500	430000	435000	495000	432500	450000	460000	405000	2	48	4.0
22	ARARAT	128000	148000	145000	148500	145000	159500	174500	166500	165000	176500	201000	187000	14	57	4.6
23	ARDEER	205000	206000	214500	234000	275500	312000	382500	360000	342500	330000	358500	360000	9	75	5.7
24	ARMADALE	830000	801000	900000	1250000	1415000	1605000	1690000	1550500	1505000	1675000	1699000	1217500	1	105	7.4
25	ARMSTRONG CREEK	-	-	-	-	-	-	-	205000	362500	467500	450000	462000	-4	NA	NA

Figure 9-4 Raw unprocessed data from the Victoria State Government regarding the average house price per locality for the years 2004 - 2015

The next data set used is the RBA historical cash rate. This data is publicly accessible at the following web page:

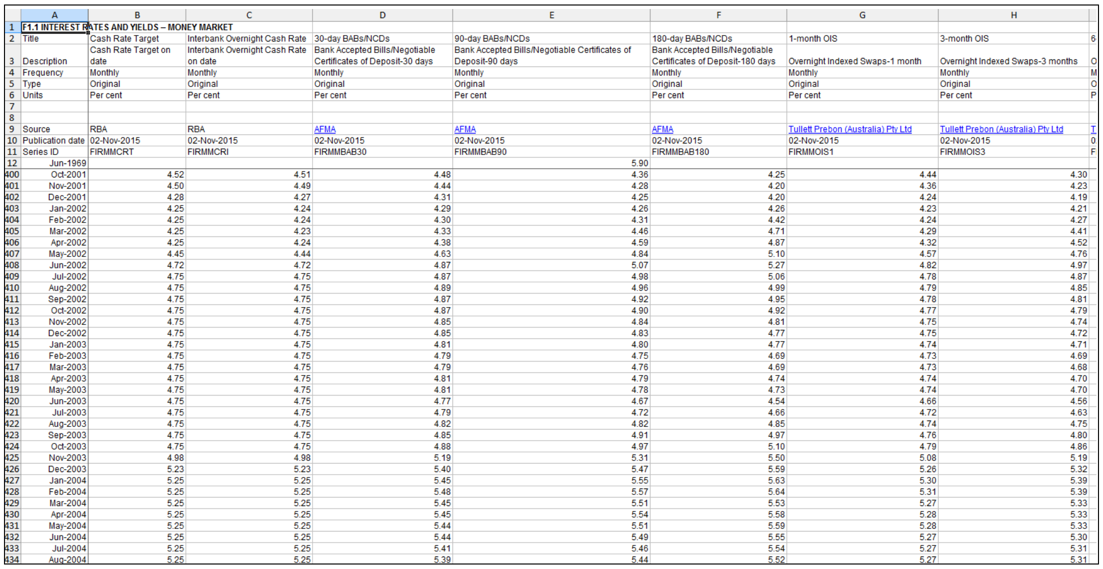
http://www.rba.gov.au/statistics/tables/index.html#interest-rates

And lastly, the historical exchange rates for different countries, also publicly available on the RBA website, can be found at the following page:

http://www.rba.gov.au/statistics/historical-data.html#exchange-rates

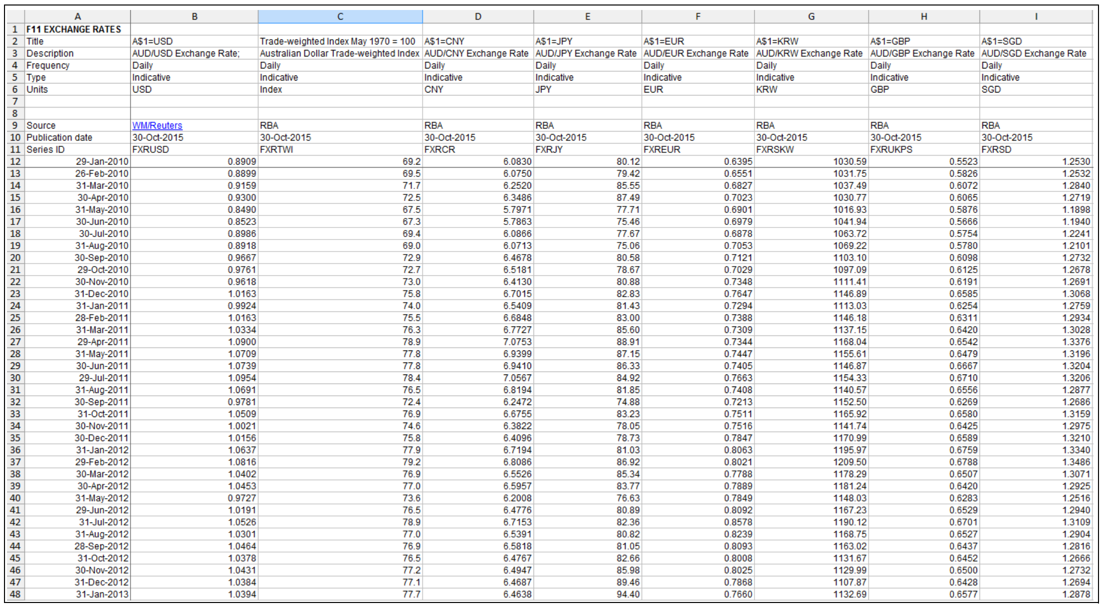
For the model, we are considering the period 2004 - 2015, and the exchange rate monthly data is split into pre 2009 and post 2009 periods. Therefore, both of these files need to be downloaded.

The raw data for historical indicator lending rates and the historical exchange rates are demonstrated in Figure 9-5 and Figure 9-6, respectively.



	A	B	C	D	E	F	G	H
1	F1.1 INTEREST RATES AND YIELDS – MONEY MARKET							
2	Title	Cash Rate Target	Interbank Overnight Cash Rate	30-day BABs/NCDs	90-day BABs/NCDs	180-day BABs/NCDs	1-month OIS	3-month OIS
3	Description	Cash Rate Target on date	Interbank Overnight Cash Rate on date	Bank Accepted Bills/Negotiable Certificates of Deposit-30 days	Bank Accepted Bills/Negotiable Certificates of Deposit-90 days	Bank Accepted Bills/Negotiable Certificates of Deposit-180 days	Overnight Indexed Swaps-1 month	Overnight Indexed Swaps-3 months
4	Frequency	Monthly	Monthly	Monthly	Monthly	Monthly	Monthly	Monthly
5	Type	Original	Original	Original	Original	Original	Original	Original
6	Units	Per cent	Per cent	Per cent	Per cent	Per cent	Per cent	Per cent
7								
8								
9	Source	RBA	RBA	AFMA	AFMA	AFMA	Tullett Prebon (Australia) Pty Ltd	Tullett Prebon (Australia) Pty Ltd
10	Publication date	02-Nov-2015	02-Nov-2015	02-Nov-2015	02-Nov-2015	02-Nov-2015	02-Nov-2015	02-Nov-2015
11	Series ID	FIRMMCRT	FIRMMCRI	FIRMMBAB30	FIRMMBAB90	FIRMMBAB180	FIRMMOIS1	FIRMMOIS3
12	Jun-1969				5.90			
400	Oct-2001	4.52	4.51	4.48	4.36	4.25	4.44	4.30
401	Nov-2001	4.50	4.49	4.44	4.28	4.20	4.36	4.23
402	Dec-2001	4.28	4.27	4.31	4.25	4.20	4.24	4.19
403	Jan-2002	4.25	4.24	4.29	4.26	4.26	4.23	4.21
404	Feb-2002	4.25	4.24	4.30	4.31	4.42	4.24	4.27
405	Mar-2002	4.25	4.23	4.33	4.46	4.71	4.29	4.41
406	Apr-2002	4.25	4.24	4.38	4.59	4.87	4.32	4.52
407	May-2002	4.45	4.44	4.63	4.84	5.10	4.57	4.76
408	Jun-2002	4.72	4.72	4.87	5.07	5.27	4.82	4.97
409	Jul-2002	4.75	4.75	4.87	4.98	5.06	4.78	4.87
410	Aug-2002	4.75	4.75	4.89	4.96	4.99	4.79	4.85
411	Sep-2002	4.75	4.75	4.87	4.92	4.95	4.78	4.81
412	Oct-2002	4.75	4.75	4.87	4.90	4.92	4.77	4.79
413	Nov-2002	4.75	4.75	4.85	4.84	4.81	4.75	4.74
414	Dec-2002	4.75	4.75	4.85	4.83	4.77	4.75	4.72
415	Jan-2003	4.75	4.75	4.81	4.80	4.77	4.74	4.71
416	Feb-2003	4.75	4.75	4.79	4.75	4.69	4.73	4.69
417	Mar-2003	4.75	4.75	4.79	4.76	4.69	4.73	4.68
418	Apr-2003	4.75	4.75	4.81	4.79	4.74	4.74	4.70
419	May-2003	4.75	4.75	4.81	4.78	4.73	4.74	4.70
420	Jun-2003	4.75	4.75	4.77	4.67	4.54	4.66	4.56
421	Jul-2003	4.75	4.75	4.79	4.72	4.66	4.72	4.63
422	Aug-2003	4.75	4.75	4.82	4.82	4.85	4.74	4.75
423	Sep-2003	4.75	4.75	4.85	4.91	4.97	4.76	4.80
424	Oct-2003	4.75	4.75	4.88	4.97	5.10	4.79	4.86
425	Nov-2003	4.98	4.98	5.19	5.31	5.50	5.08	5.19
426	Dec-2003	5.23	5.23	5.40	5.47	5.59	5.26	5.32
427	Jan-2004	5.25	5.25	5.45	5.55	5.63	5.30	5.39
428	Feb-2004	5.25	5.25	5.48	5.57	5.64	5.31	5.39
429	Mar-2004	5.25	5.25	5.45	5.51	5.53	5.27	5.33
430	Apr-2004	5.25	5.25	5.45	5.54	5.58	5.28	5.33
431	May-2004	5.25	5.25	5.44	5.51	5.59	5.28	5.33
432	Jun-2004	5.25	5.25	5.44	5.49	5.55	5.27	5.30
433	Jul-2004	5.25	5.25	5.41	5.46	5.54	5.27	5.31
434	Aug-2004	5.25	5.25	5.39	5.44	5.52	5.27	5.31

Figure 9-5 Raw interest rate data from the RBA



	A	B	C	D	E	F	G	H	I
1	F11 EXCHANGE RATES								
2	Title	A$1=USD	Trade-weighted Index May 1970 = 100	A$1=CNY	A$1=JPY	A$1=EUR	A$1=KRW	A$1=GBP	A$1=SGD
3	Description	AUD/USD Exchange Rate;	Australian Dollar Trade-weighted Index	AUD/CNY Exchange Rate	AUD/JPY Exchange Rate	AUD/EUR Exchange Rate	AUD/KRW Exchange Rate	AUD/GBP Exchange Rate	AUD/SGD Exchange Rate
4	Frequency	Daily	Daily	Daily	Daily	Daily	Daily	Daily	Daily
5	Type	Indicative	Indicative	Indicative	Indicative	Indicative	Indicative	Indicative	Indicative
6	Units	USD	Index	CNY	JPY	EUR	KRW	GBP	SGD
7									
8									
9	Source	WM/Reuters	RBA	RBA	RBA	RBA	RBA	RBA	RBA
10	Publication date	30-Oct-2015	30-Oct-2015	30-Oct-2015	30-Oct-2015	30-Oct-2015	30-Oct-2015	30-Oct-2015	30-Oct-2015
11	Series ID	FXRUSD	FXRTWI	FXRCR	FXRJY	FXREUR	FXRSKW	FXRUKPS	FXRSD
12	29-Jan-2010	0.8909	69.2	6.0830	80.12	0.6395	1030.59	0.5523	1.2530
13	26-Feb-2010	0.8899	69.5	6.0750	79.42	0.6551	1031.75	0.5826	1.2532
14	31-Mar-2010	0.9159	71.7	6.2520	85.55	0.6827	1037.49	0.6072	1.2840
15	30-Apr-2010	0.9300	72.5	6.3486	87.49	0.7023	1030.77	0.6065	1.2719
16	31-May-2010	0.8490	67.5	5.7971	77.71	0.6901	1016.93	0.5876	1.1898
17	30-Jun-2010	0.8523	67.3	5.7863	75.46	0.6979	1041.94	0.5666	1.1940
18	30-Jul-2010	0.8986	69.4	6.0866	77.67	0.6878	1063.72	0.5754	1.2241
19	31-Aug-2010	0.8918	69.0	6.0713	75.06	0.7053	1069.22	0.5780	1.2101
20	30-Sep-2010	0.9667	72.9	6.4678	80.58	0.7121	1103.10	0.6098	1.2732
21	29-Oct-2010	0.9761	72.7	6.5181	78.67	0.7029	1097.09	0.6125	1.2678
22	30-Nov-2010	0.9618	73.0	6.4130	80.88	0.7348	1111.41	0.6191	1.2691
23	31-Dec-2010	1.0163	75.8	6.7015	82.83	0.7647	1146.89	0.6585	1.3068
24	31-Jan-2011	0.9924	74.0	6.5409	81.43	0.7294	1113.03	0.6254	1.2759
25	28-Feb-2011	1.0163	75.5	6.6848	83.00	0.7388	1146.18	0.6311	1.2934
26	31-Mar-2011	1.0334	76.3	6.7727	85.60	0.7309	1137.15	0.6420	1.3028
27	29-Apr-2011	1.0900	78.9	7.0753	88.91	0.7344	1168.04	0.6542	1.3376
28	31-May-2011	1.0709	77.8	6.9399	87.15	0.7447	1155.61	0.6479	1.3196
29	30-Jun-2011	1.0739	77.8	6.9410	86.33	0.7405	1146.87	0.6667	1.3204
30	29-Jul-2011	1.0954	78.4	7.0567	84.92	0.7663	1154.33	0.6710	1.3206
31	31-Aug-2011	1.0691	76.5	6.8194	81.85	0.7408	1140.57	0.6556	1.2877
32	30-Sep-2011	0.9781	72.4	6.2472	74.88	0.7213	1152.50	0.6269	1.2686
33	31-Oct-2011	1.0509	76.9	6.6755	83.23	0.7511	1165.92	0.6580	1.3159
34	30-Nov-2011	1.0021	74.6	6.3822	78.05	0.7516	1141.74	0.6425	1.2975
35	30-Dec-2011	1.0156	75.8	6.4096	78.73	0.7847	1170.99	0.6589	1.3210
36	31-Jan-2012	1.0637	77.9	6.7194	81.03	0.8063	1195.97	0.6759	1.3340
37	29-Feb-2012	1.0816	79.2	6.8086	86.92	0.8021	1209.50	0.6788	1.3486
38	30-Mar-2012	1.0402	76.9	6.5526	85.34	0.7788	1178.29	0.6507	1.3071
39	30-Apr-2012	1.0453	77.0	6.5957	83.77	0.7889	1181.24	0.6420	1.2925
40	31-May-2012	0.9727	73.6	6.2008	76.63	0.7849	1148.03	0.6283	1.2516
41	29-Jun-2012	1.0191	76.5	6.4776	80.89	0.8092	1167.23	0.6529	1.2940
42	31-Jul-2012	1.0526	78.9	6.7153	82.36	0.8578	1190.12	0.6701	1.3109
43	31-Aug-2012	1.0301	77.0	6.5391	80.82	0.8239	1168.75	0.6527	1.2904
44	28-Sep-2012	1.0464	76.9	6.5818	81.05	0.8093	1163.02	0.6437	1.2816
45	31-Oct-2012	1.0378	76.5	6.4767	82.66	0.8008	1131.67	0.6452	1.2666
46	30-Nov-2012	1.0431	77.2	6.4947	85.98	0.8025	1129.99	0.6500	1.2732
47	31-Dec-2012	1.0384	77.1	6.4687	89.46	0.7868	1107.87	0.6428	1.2694
48	31-Jan-2013	1.0394	77.7	6.4638	94.40	0.7660	1132.69	0.6577	1.2878

Figure 9-6 Raw exchange rate data from the RBA

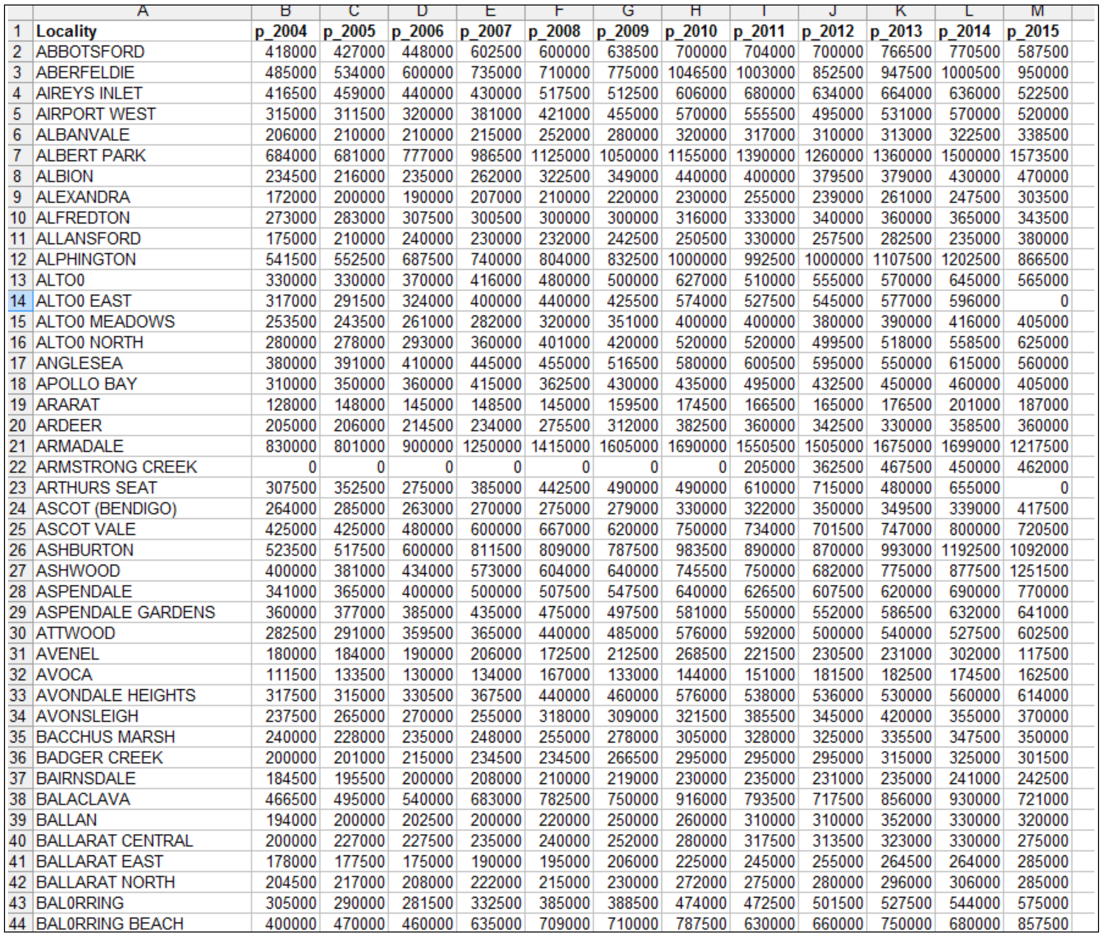
9.2.2 Data preprocessing

This is the raw data. Before importing this data into the dashDB service, the spreadsheets need to be formatted so that they are easily interpreted by the dashDB import service. For the housing price data, shown in Figure 9-4 on page 256, this is a four-step process:

1. There are a few header lines that cause confusion when importing the data. These lines need to be removed.

2. The model being developed is of the average house price and not the percentage change in house price. Therefore, all columns relating to percentage change in house prices are removed.
3. In an SQL database, headers cannot start with a number. Therefore, the year headers need to be changed.
4. The last aspect that needs to be considered is missing data. In this file, missing data is indicated by a - or an NA. For this application, all missing values are changed to zero and are removed from calculations later on while developing the model. After this has been done, the format of the pricing data is changed to numeric. The missing values are changed to zero so that when the data is imported, the column formats are recognized as integers, and not as string.

The resulting data is shown in Figure 9-7.



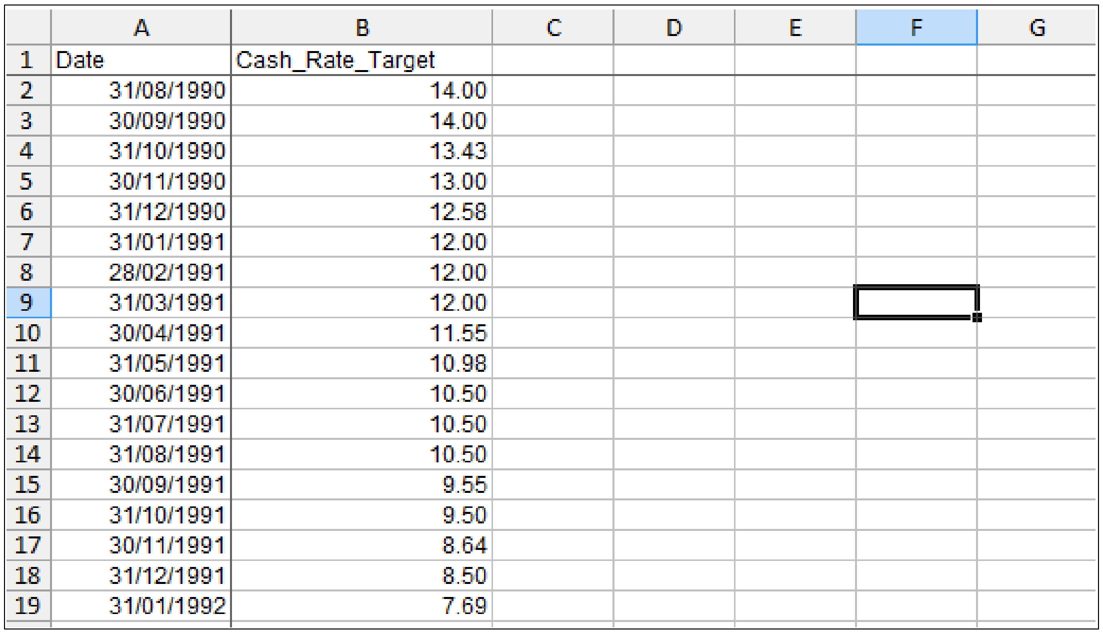
	A	B	C	D	E	F	G	H	I	J	K	L	M
1	**Locality**	**p_2004**	**p_2005**	**p_2006**	**p_2007**	**p_2008**	**p_2009**	**p_2010**	**p_2011**	**p_2012**	**p_2013**	**p_2014**	**p_2015**
2	ABBOTSFORD	418000	427000	448000	602500	600000	638500	700000	704000	700000	766500	770500	587500
3	ABERFELDIE	485000	534000	600000	735000	710000	775000	1046500	1003000	852500	947500	1000500	950000
4	AIREYS INLET	416500	459000	440000	430000	517500	512500	606000	680000	634000	664000	636000	522500
5	AIRPORT WEST	315000	311500	320000	381000	421000	455000	570000	555500	495000	531000	570000	520000
6	ALBANVALE	206000	210000	210000	215000	252000	280000	320000	317000	310000	313000	322500	338500
7	ALBERT PARK	684000	681000	777000	986500	1125000	1050000	1155000	1390000	1260000	1360000	1500000	1573500
8	ALBION	234500	216000	235000	262000	322500	349000	440000	400000	379500	379000	430000	470000
9	ALEXANDRA	172000	200000	190000	207000	210000	220000	230000	255000	239000	261000	247500	303500
10	ALFREDTON	273000	283000	307500	300500	300000	300000	316000	333000	340000	360000	365000	343500
11	ALLANSFORD	175000	210000	240000	230000	232000	242500	250500	330000	257500	282500	235000	380000
12	ALPHINGTON	541500	552500	687500	740000	804000	832500	1000000	992500	1000000	1107500	1202500	866500
13	ALTO0	330000	330000	370000	416000	480000	500000	627000	510000	555000	570000	645000	565000
14	ALTO0 EAST	317000	291500	324000	400000	440000	425500	574000	527500	545000	577000	596000	0
15	ALTO0 MEADOWS	253500	243500	261000	282000	320000	351000	400000	400000	380000	390000	416000	405000
16	ALTO0 NORTH	280000	278000	293000	360000	401000	420000	520000	520000	499500	518000	558500	625000
17	ANGLESEA	380000	391000	410000	445000	455000	516500	580000	600500	595000	550000	615000	560000
18	APOLLO BAY	310000	350000	360000	415000	362500	430000	435000	495000	432500	450000	460000	405000
19	ARARAT	128000	148000	145000	148500	145000	159500	174500	166500	165000	176500	201000	187000
20	ARDEER	205000	206000	214500	234000	275500	312000	382500	360000	342500	330000	358500	360000
21	ARMADALE	830000	801000	900000	1250000	1415000	1605000	1690000	1550500	1505000	1675000	1699000	1217500
22	ARMSTRONG CREEK	0	0	0	0	0	0	0	205000	362500	467500	450000	462000
23	ARTHURS SEAT	307500	352500	275000	385000	442500	490000	490000	610000	715000	480000	655000	0
24	ASCOT (BENDIGO)	264000	285000	263000	270000	275000	279000	330000	322000	350000	349500	339000	417500
25	ASCOT VALE	425000	425000	480000	600000	667000	620000	750000	734000	701500	747000	800000	720500
26	ASHBURTON	523500	517500	600000	811500	809000	787500	983500	890000	870000	993000	1192500	1092000
27	ASHWOOD	400000	381000	434000	573000	604000	640000	745500	750000	682000	775000	877500	1251500
28	ASPENDALE	341000	365000	400000	500000	507500	547500	640000	626500	607500	620000	690000	770000
29	ASPENDALE GARDENS	360000	377000	385000	435000	475000	497500	581000	550000	552000	586500	632000	641000
30	ATTWOOD	282500	291000	359500	365000	440000	485000	576000	592000	500000	540000	527500	602500
31	AVENEL	180000	184000	190000	206000	172500	212500	268500	221500	230500	231000	302000	117500
32	AVOCA	111500	133500	130000	134000	167000	133000	144000	151000	181500	182500	174500	162500
33	AVONDALE HEIGHTS	317500	315000	330500	367500	440000	460000	576000	538000	536000	530000	560000	614000
34	AVONSLEIGH	237500	265000	270000	255000	318000	309000	321500	385500	345000	420000	355000	370000
35	BACCHUS MARSH	240000	228000	235000	248000	255000	278000	305000	328000	325000	335500	347500	350000
36	BADGER CREEK	200000	201000	215000	234500	234500	266500	295000	295000	295000	315000	325000	301500
37	BAIRNSDALE	184500	195500	200000	208000	210000	219000	230000	235000	231000	235000	241000	242500
38	BALACLAVA	466500	495000	540000	683000	782500	750000	916000	793500	717500	856000	930000	721000
39	BALLAN	194000	200000	202500	200000	220000	250000	260000	310000	310000	352000	330000	320000
40	BALLARAT CENTRAL	200000	227000	227500	235000	240000	252000	280000	317500	313500	323000	330000	275000
41	BALLARAT EAST	178000	177500	175000	190000	195000	206000	225000	245000	255000	264500	264000	285000
42	BALLARAT NORTH	204500	217000	208000	222000	215000	230000	272000	275000	280000	296000	306000	285000
43	BAL0RRING	305000	290000	281500	332500	385000	388500	474000	472500	501500	527500	544000	575000
44	BAL0RRING BEACH	400000	470000	460000	635000	709000	710000	787500	630000	660000	750000	680000	857500

Figure 9-7 Processed housing price data for Victoria, Australia

For the RBA cash rate raw data, shown in Figure 9-5 on page 257, three steps are required to prepare the data for import:

1. The data is subset to where an RBA cash rate is first available to the most recent date where the cash rate is available.
2. To format the provided data, all header rows are removed and new headers are defined.
3. The format of the dates needs to be changed to one recognized by the SQL import option in Bluemix. In this case, the dd/mm/yyyy format is used.

The preprocessed data is shown in Figure 9-8.



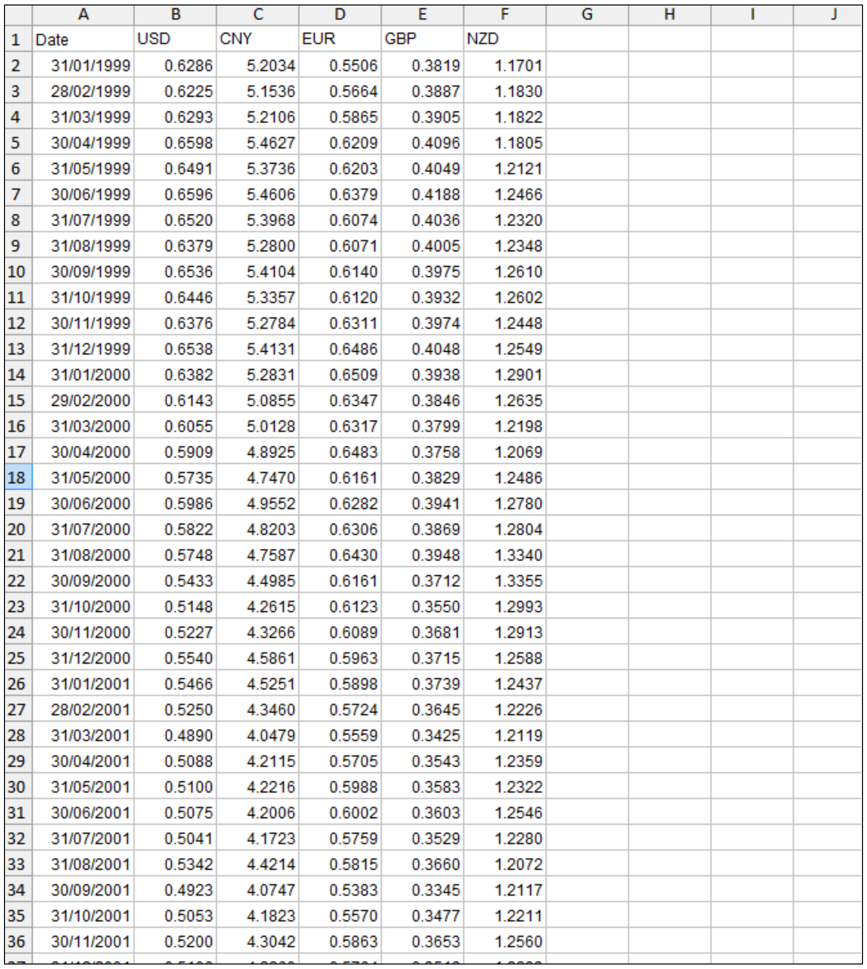
	A	B	C	D	E	F	G
1	Date	Cash_Rate_Target					
2	31/08/1990	14.00					
3	30/09/1990	14.00					
4	31/10/1990	13.43					
5	30/11/1990	13.00					
6	31/12/1990	12.58					
7	31/01/1991	12.00					
8	28/02/1991	12.00					
9	31/03/1991	12.00					
10	30/04/1991	11.55					
11	31/05/1991	10.98					
12	30/06/1991	10.50					
13	31/07/1991	10.50					
14	31/08/1991	10.50					
15	30/09/1991	9.55					
16	31/10/1991	9.50					
17	30/11/1991	8.64					
18	31/12/1991	8.50					
19	31/01/1992	7.69					

Figure 9-8 Processed RBA cash rate data

For the exchange rate data, there are two different files. The raw data for the period from 2009 - 2015 is shown in Figure 9-6 on page 257. This is a four-step process for the exchange rates:

1. The first part of preprocessing is to merge these two files into a single file. Unfortunately, the columns for these two files are not in the same order. Therefore, before merging the two files they are subset to the exchange rates of interest to decrease the amount of time required to merge the two files. For this analysis, the exchange rate with Britain, China, Europe, New Zealand, and the US is considered. This subset data is then merged.
2. The dates are subset to the start of the Euro currency.
3. Headings are renamed to the countries names.
4. The date format is changed to dd/mm/yyyy.

The preprocessed data is shown in Figure 9-9.

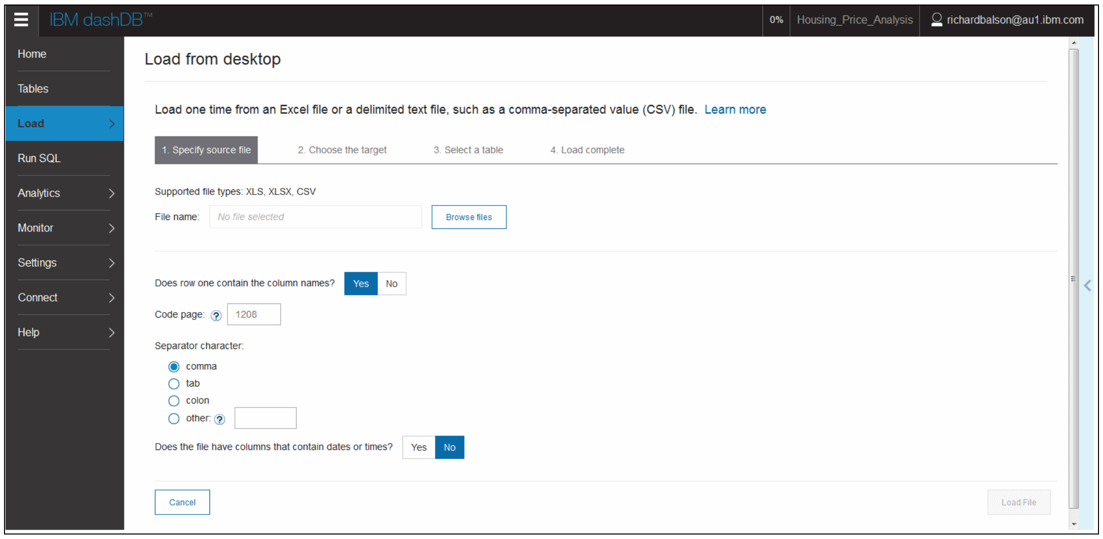


	A	B	C	D	E	F	G	H	I	J
1	Date	USD	CNY	EUR	GBP	NZD				
2	31/01/1999	0.6286	5.2034	0.5506	0.3819	1.1701				
3	28/02/1999	0.6225	5.1536	0.5664	0.3887	1.1830				
4	31/03/1999	0.6293	5.2106	0.5865	0.3905	1.1822				
5	30/04/1999	0.6598	5.4627	0.6209	0.4096	1.1805				
6	31/05/1999	0.6491	5.3736	0.6203	0.4049	1.2121				
7	30/06/1999	0.6596	5.4606	0.6379	0.4188	1.2466				
8	31/07/1999	0.6520	5.3968	0.6074	0.4036	1.2320				
9	31/08/1999	0.6379	5.2800	0.6071	0.4005	1.2348				
10	30/09/1999	0.6536	5.4104	0.6140	0.3975	1.2610				
11	31/10/1999	0.6446	5.3357	0.6120	0.3932	1.2602				
12	30/11/1999	0.6376	5.2784	0.6311	0.3974	1.2448				
13	31/12/1999	0.6538	5.4131	0.6486	0.4048	1.2549				
14	31/01/2000	0.6382	5.2831	0.6509	0.3938	1.2901				
15	29/02/2000	0.6143	5.0855	0.6347	0.3846	1.2635				
16	31/03/2000	0.6055	5.0128	0.6317	0.3799	1.2198				
17	30/04/2000	0.5909	4.8925	0.6483	0.3758	1.2069				
18	31/05/2000	0.5735	4.7470	0.6161	0.3829	1.2486				
19	30/06/2000	0.5986	4.9552	0.6282	0.3941	1.2780				
20	31/07/2000	0.5822	4.8203	0.6306	0.3869	1.2804				
21	31/08/2000	0.5748	4.7587	0.6430	0.3948	1.3340				
22	30/09/2000	0.5433	4.4985	0.6161	0.3712	1.3355				
23	31/10/2000	0.5148	4.2615	0.6123	0.3550	1.2993				
24	30/11/2000	0.5227	4.3266	0.6089	0.3681	1.2913				
25	31/12/2000	0.5540	4.5861	0.5963	0.3715	1.2588				
26	31/01/2001	0.5466	4.5251	0.5898	0.3739	1.2437				
27	28/02/2001	0.5250	4.3460	0.5724	0.3645	1.2226				
28	31/03/2001	0.4890	4.0479	0.5559	0.3425	1.2119				
29	30/04/2001	0.5088	4.2115	0.5705	0.3543	1.2359				
30	31/05/2001	0.5100	4.2216	0.5988	0.3583	1.2322				
31	30/06/2001	0.5075	4.2006	0.6002	0.3603	1.2546				
32	31/07/2001	0.5041	4.1723	0.5759	0.3529	1.2280				
33	31/08/2001	0.5342	4.4214	0.5815	0.3660	1.2072				
34	30/09/2001	0.4923	4.0747	0.5383	0.3345	1.2117				
35	31/10/2001	0.5053	4.1823	0.5570	0.3477	1.2211				
36	30/11/2001	0.5200	4.3042	0.5863	0.3653	1.2560				

Figure 9-9 Processed exchange rate data for the Australian dollar compared to the US Dollar, China Yen, the Euro, Great Britain Pound, and the New Zealand Dollar

9.2.3 Importing data into dashDB

To import the raw csv data into dashDB tables, the Load option in the dashDB service shown in Figure 9-3 on page 255 is selected. Because the data is being imported from a local source, the desktop option should be selected. Other sources can be selected depending on the location of the data source. Figure 9-10 demonstrates the insert desktop source data page in dashDB. Here, a table is created based on the raw data.



Load one time from an Excel file or a delimited text file, such as a comma-separated value (CSV) file. Learn more

Supported file types: XLS, XLSX, CSV

File name: No file selected

Does row one contain the column names? [✓ Yes] No

Code page: 1208

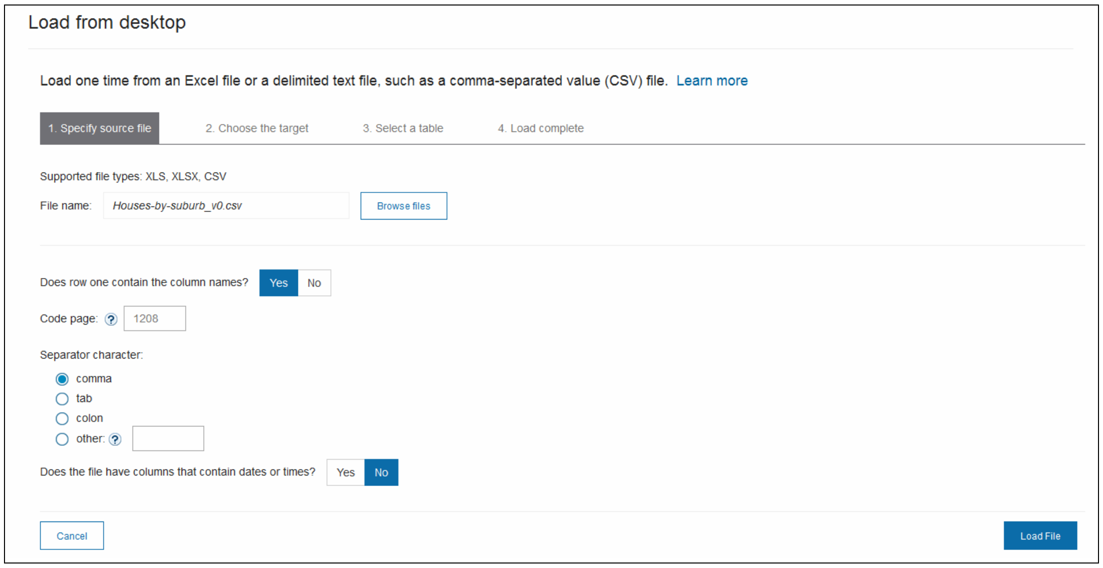
Separator character:
- [✓ comma]
- tab
- colon
- other:

Does the file have columns that contain dates or times? Yes [✓ No]

Figure 9-10 The Load from Desktop window in the dashDB service

Perform the following steps to import the data:

1. Click **Specify source file**. This is where you select the source file and the formatting of the file. For the housing prices, there are no dates and the file is comma-separated. The resulting page looks similar to Figure 9-11. For this file source, the first row is a header row. The column separator is a comma and there are no date-based columns.



Load one time from an Excel file or a delimited text file, such as a comma-separated value (CSV) file. Learn more

Supported file types: XLS, XLSX, CSV

File name: Houses-by-suburb_v0.csv

Does row one contain the column names? [✓ Yes] No

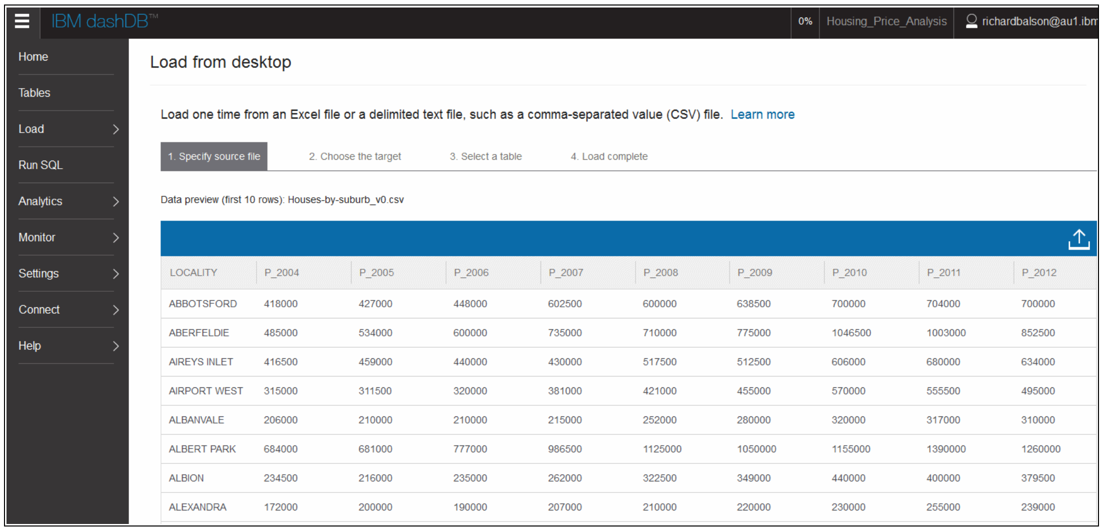
Code page: 1208

Separator character:
- [✓ comma]
- tab
- colon
- other:

Does the file have columns that contain dates or times? Yes [✓ No]

Figure 9-11 The dashDB specify file source page. Here you specify whether the first line in the file is a header file, the column separators, and whether there are any date columns

After the load is selected, a preview of the data is demonstrated. An example for the data used is shown in Figure 9-12.



Load from desktop

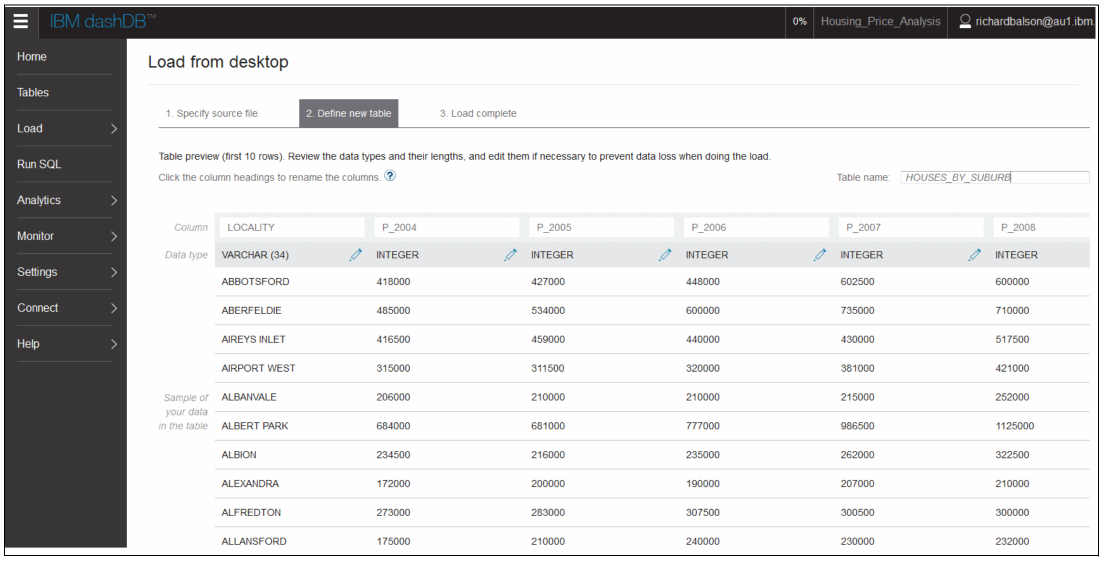
Load one time from an Excel file or a delimited text file, such as a comma-separated value (CSV) file. Learn more

Data preview (first 10 rows): Houses-by-suburb_v0.csv

LOCALITY	P_2004	P_2005	P_2006	P_2007	P_2008	P_2009	P_2010	P_2011	P_2012
ABBOTSFORD	418000	427000	448000	602500	600000	638500	700000	704000	700000
ABERFELDIE	485000	534000	600000	735000	710000	775000	1046500	1003000	852500
AIREYS INLET	416500	459000	440000	430000	517500	512500	606000	680000	634000
AIRPORT WEST	315000	311500	320000	381000	421000	455000	570000	555500	495000
ALBANVALE	206000	210000	210000	215000	252000	280000	320000	317000	310000
ALBERT PARK	684000	681000	777000	986500	1125000	1050000	1155000	1390000	1260000
ALBION	234500	216000	235000	262000	322500	349000	440000	400000	379500
ALEXANDRA	172000	200000	190000	207000	210000	220000	230000	255000	239000

Figure 9-12 Preview of the data to be imported into the dashDB database

2. Select **Next** if the data is as expected.
3. **Define new table** allows you to rename column headers and define the table name. For this example, all headers are left as is. Headers that begin with a numeric value are accepted. The table name is defined as HOUSES_BY_SUBURB. The page for this process is shown in Figure 9-13.



Load from desktop

Table preview (first 10 rows). Review the data types and their lengths, and edit them if necessary to prevent data loss when doing the load.

Click the column headings to rename the columns.

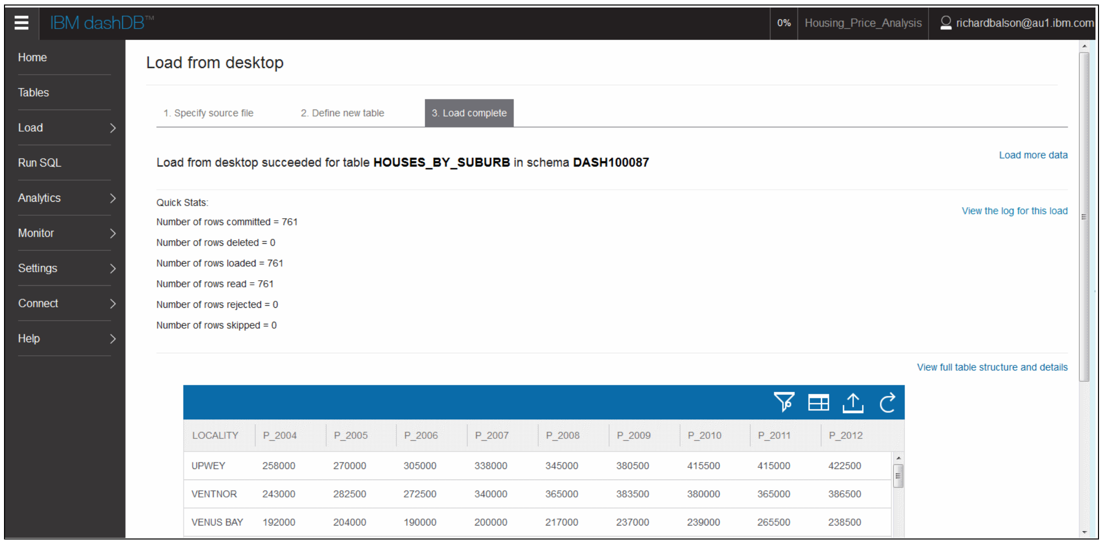
Table name: HOUSES_BY_SUBURB

Column	LOCALITY	P_2004	P_2005	P_2006	P_2007	P_2008
Data type	VARCHAR (34)	INTEGER	INTEGER	INTEGER	INTEGER	INTEGER
Sample of your data in the table	ABBOTSFORD	418000	427000	448000	602500	600000
	ABERFELDIE	485000	534000	600000	735000	710000
	AIREYS INLET	416500	459000	440000	430000	517500
	AIRPORT WEST	315000	311500	320000	381000	421000
	ALBANVALE	206000	210000	210000	215000	252000
	ALBERT PARK	684000	681000	777000	986500	1125000
	ALBION	234500	216000	235000	262000	322500
	ALEXANDRA	172000	200000	190000	207000	210000
	ALFREDTON	273000	283000	307500	300500	300000
	ALLANSFORD	175000	210000	240000	230000	232000

Figure 9-13 Defining a new table in dashDB

In Figure 9-13, the column types are also shown. This can be used to determine whether the data imported is being recognized as the correct type. If the type is incorrect, some more preprocessing of the raw data file might be necessary:

4. After the table and headers are defined, select **Finish**.
5. **Load complete** is displayed when this process is finished. The load is complete and a page summarizing the load results is displayed.



Load from desktop

1. Specify source file 2. Define new table 3. Load complete

Load from desktop succeeded for table **HOUSES_BY_SUBURB** in schema **DASH100087**

Load more data

Quick Stats:
Number of rows committed = 761
Number of rows deleted = 0
Number of rows loaded = 761
Number of rows read = 761
Number of rows rejected = 0
Number of rows skipped = 0

View the log for this load

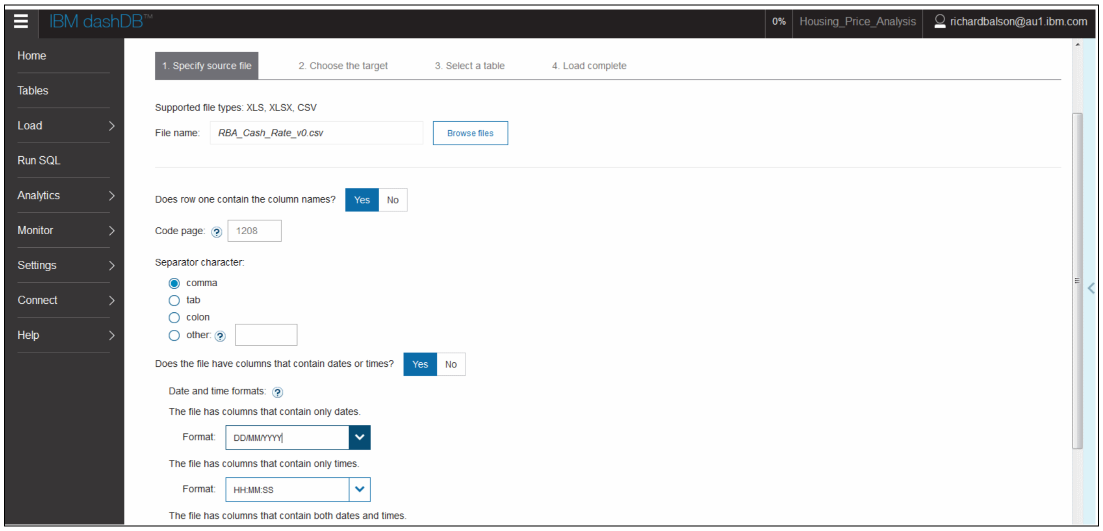
View full table structure and details

LOCALITY	P_2004	P_2005	P_2006	P_2007	P_2008	P_2009	P_2010	P_2011	P_2012
UPWEY	258000	270000	305000	338000	345000	380500	415500	415000	422500
VENTNOR	243000	282500	272500	340000	365000	383500	380000	365000	386500
VENUS BAY	192000	204000	190000	200000	217000	237000	239000	265500	238500

Figure 9-14 Load complete for a new table in dashDB

This completes the load of the housing prices into the dashDB table.

The process to import the data for the RBA cash rate and the AUD exchange rate is identical. The preceding steps 1 - 4 can be repeated with one change. Because both the RBA cash rate and interest rate data have a column with dates, the question asking if there are columns with only dates needs to be answered **Yes**. An example is shown in Figure 9-15.



1. Specify source file 2. Choose the target 3. Select a table 4. Load complete

Supported file types: XLS, XLSX, CSV

File name: RBA_Cash_Rate_v0.csv

Does row one contain the column names? [✓ Yes] No

Code page: 1208

Separator character:
[✓ comma]
tab
colon
other:

Does the file have columns that contain dates or times? [✓ Yes] No

Date and time formats:
The file has columns that contain only dates.
Format: DD/MM/YYYY
The file has columns that contain only times.
Format: HH:MM:SS
The file has columns that contain both dates and times.

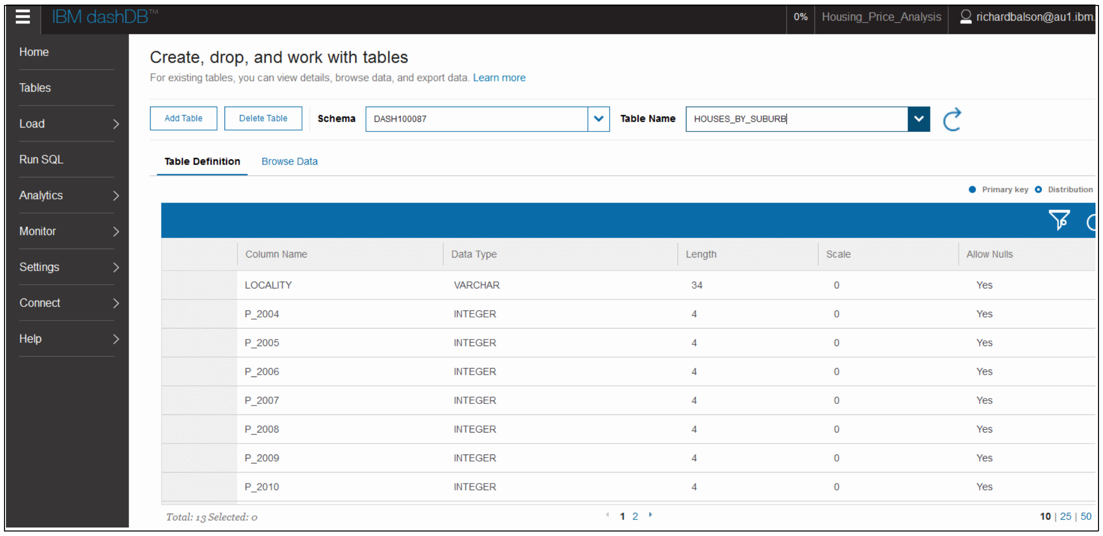
Figure 9-15 Specify source with date column

In the example shown in Figure 9-15, the date format is dd/mm/yyyy.

When creating a second table, the user is asked to specify a target. This can either be a new table in the case where new data is to be added to a new table, or an existing table when new data with the same format from a previous table is appended onto an existing table.

9.2.4 Viewing tables in dashDB

To view the tables created, select the Table option on the left side of the window. This displays a page similar to Figure 9-16. The table that should be viewed can be selected by using the down arrow next to Table Name. In this view, you can see the table definition, which shows the column types and names. The data in the tables can also be viewed by selecting the Browse Data option.



Create, drop, and work with tables

For existing tables, you can view details, browse data, and export data. Learn more

Schema: DASH100087

Table Name: HOUSES_BY_SUBURB

Column Name	Data Type	Length	Scale	Allow Nulls
LOCALITY	VARCHAR	34	0	Yes
P_2004	INTEGER	4	0	Yes
P_2005	INTEGER	4	0	Yes
P_2006	INTEGER	4	0	Yes
P_2007	INTEGER	4	0	Yes
P_2008	INTEGER	4	0	Yes
P_2009	INTEGER	4	0	Yes
P_2010	INTEGER	4	0	Yes

Total: 13 Selected: 0 1 2 10 | 25 | 50

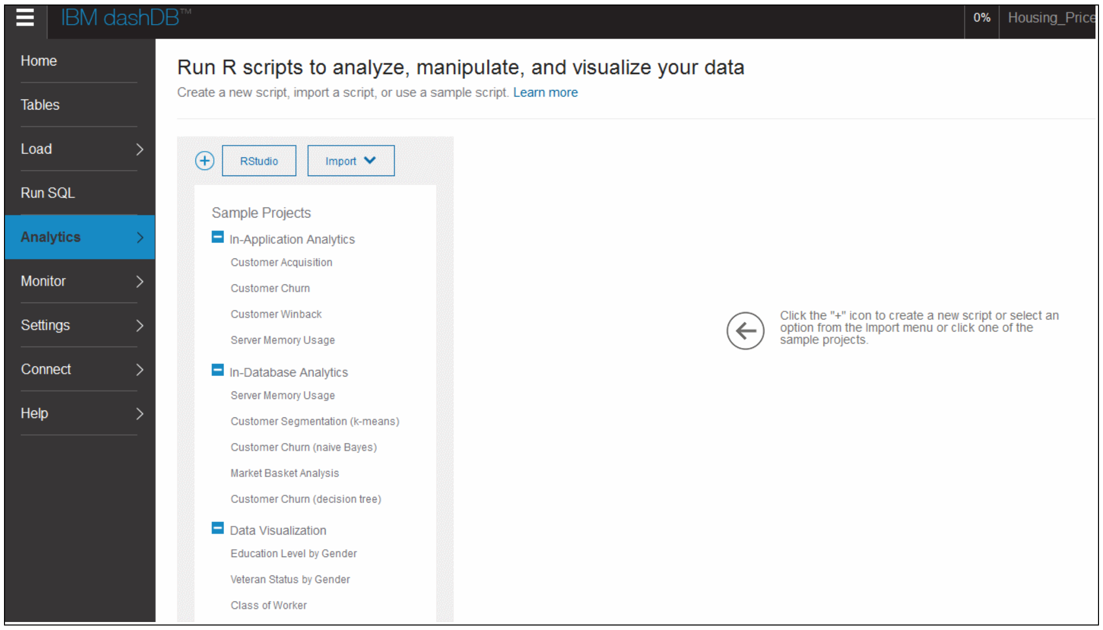
Figure 9-16 Example of viewing a table in dashDB

9.3 Cleaning and preparing data using dashDB Analytics with R

In this section, we cover the steps for cleaning and preparing data using dashDB Analytics with R.

9.3.1 Accessing R in dashDB

The data from the dashDB tables is cleaned and prepared by using the inbuilt R capability in the dashDB service. To access the R capability, select the analytics option on the left of the dashDB service page, and select R scripts. This displays a selection of sample projects that can be used, as demonstrated in Figure 9-17 on page 265.



Run R scripts to analyze, manipulate, and visualize your data

Create a new script, import a script, or use a sample script. Learn more

Sample Projects

- In-Application Analytics
  - Customer Acquisition
  - Customer Churn
  - Customer Winback
  - Server Memory Usage
- In-Database Analytics
  - Server Memory Usage
  - Customer Segmentation (k-means)
  - Customer Churn (naive Bayes)
  - Market Basket Analysis
  - Customer Churn (decision tree)
- Data Visualization
  - Education Level by Gender
  - Veteran Status by Gender
  - Class of Worker

Click the "+" icon to create a new script or select an option from the Import menu or click one of the sample projects.

Figure 9-17 R inbuilt projects and initial access

For this particular case, a new R script is created. To add a new R script, select the plus sign in the upper left corner, next to the RStudio tab.

The R script used for this example is available at the following site:

`https://github.com/RichardBalson/Redbooks/tree/master/Bluemix-IntegratingBatchAnalyticsWithRealTimeEvents`

The file is named `R_Code_CH10.R`.

9.3.2 Importing dashDB data into R

When a new script is created, a pop-up window is displayed that asks the user to select a source of data to associate with the R script. An example of this pop-up window is shown in Figure 9-18 on page 266. For this example, the housing prices are associated with the R script.

When the table source is selected, click **Apply**, and the R script is created with code similar to what is shown in Example 9-1.

Example 9-1 Code created when creating an R script and associating it with a data source

```
library(ibmdbR)
mycon <- idaConnect("BLUDB", "", "")
idaInit(mycon)
df1448855753098 <-
as.data.frame(ida.data.frame('"DASH100087"."HOUSES_BY_SUBURB"')[ ,c('LOCALITY',
'P_2004', 'P_2005', 'P_2006', 'P_2007', 'P_2008', 'P_2009', 'P_2010', 'P_2011',
'P_2012', 'P_2013', 'P_2014', 'P_2015')])
```

In this code snippet, the data is imported as a data frame in R, and is named *df1448855753098*. For simplicity, this data frame is renamed `housing_prices`. The resulting page from this process is also demonstrated in Figure 9-19. Ensure that the R script is saved before continuing, by selecting **Save** above the R script.

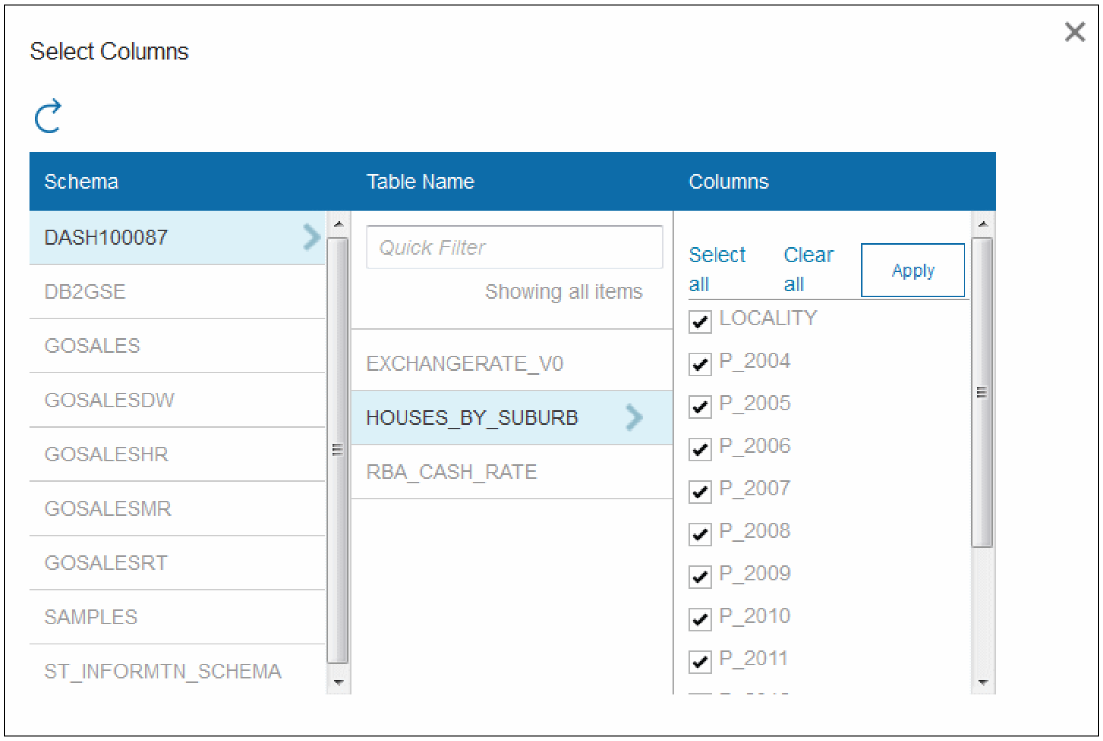
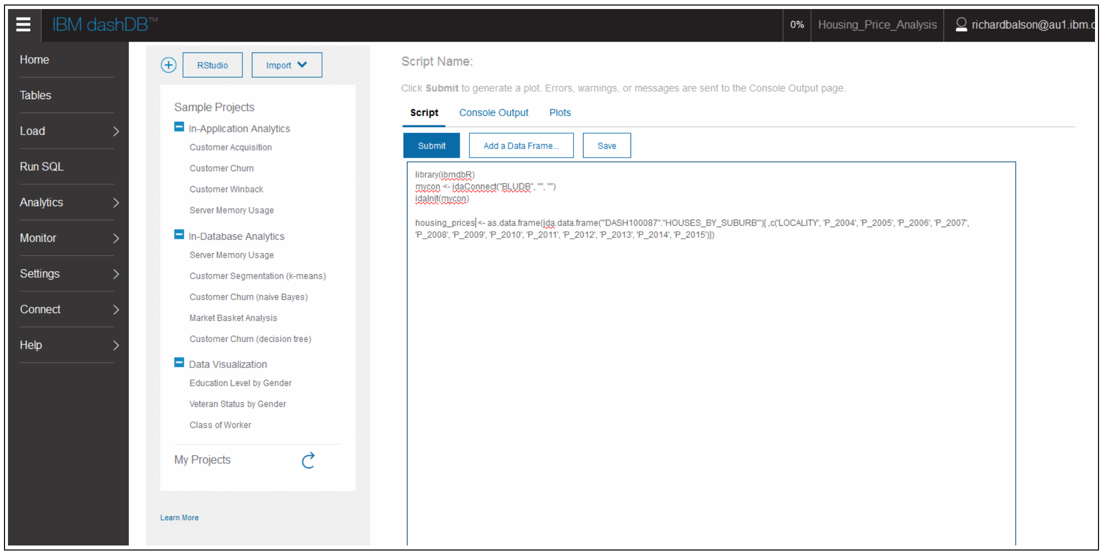


Figure 9-18 Pop-up window when creating a new R script. Here a data source can be selected to associate with the new script

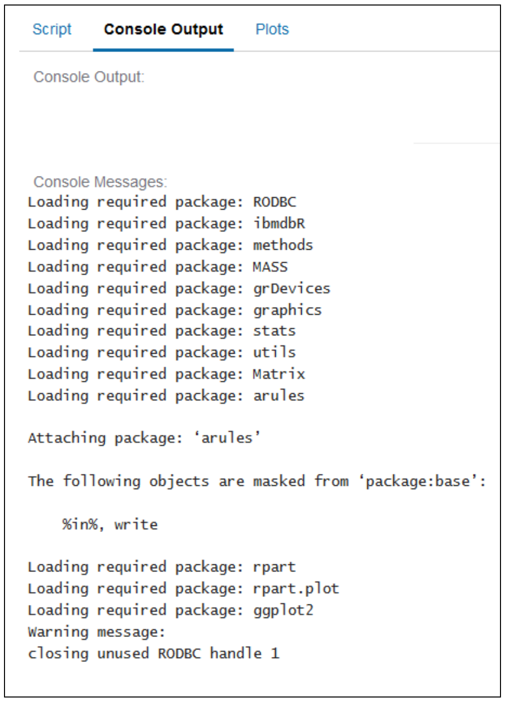


```
library(ibmdbR)
mycon <- idaConnect("BLUDB", "", "")
idaInit(mycon)

housing_prices <- as.data.frame(ida.data.frame('"DASH100087"."HOUSES_BY_SUBURB"')[ ,c('LOCALITY', 'P_2004', 'P_2005', 'P_2006', 'P_2007',
'P_2008', 'P_2009', 'P_2010', 'P_2011', 'P_2012', 'P_2013', 'P_2014', 'P_2015')])
```

Figure 9-19 Initial R script created after associating a data source

Next, the other two data sources need to be imported. This can be achieved by selecting the **Add a Data Frame** option. This opens the same pop-up window as demonstrated in Figure 9-18 on page 266. Select a new data source and click **Apply**. Repeat this process for all data sources that need to be accessible by R.

In this case, there are three different data sources that need to be accessible in R. To evaluate whether the data has been correctly imported into R, run the script by selecting **Submit**. After selecting submit, the page switches the **Console Output** tab as shown in Figure 9-20. Here no output has been assigned so there are no outputs in the console, and no plots either[1]. To switch back to the R script, select the **Script** option.



```
Loading required package: RODBC
Loading required package: ibmdbR
Loading required package: methods
Loading required package: MASS
Loading required package: grDevices
Loading required package: graphics
Loading required package: stats
Loading required package: utils
Loading required package: Matrix
Loading required package: arules

Attaching package: ‘arules’

The following objects are masked from ‘package:base’:

    %in%, write

Loading required package: rpart
Loading required package: rpart.plot
Loading required package: ggplot2
Warning message:
closing unused RODBC handle 1
```

Figure 9-20 Console output from an R script

9.3.3 Viewing Data in R

To view the data, a new variable called **view_data** is defined. This is a binary variable that lets the user decide whether or not the first 10 rows of each data set should be shown in the console output.

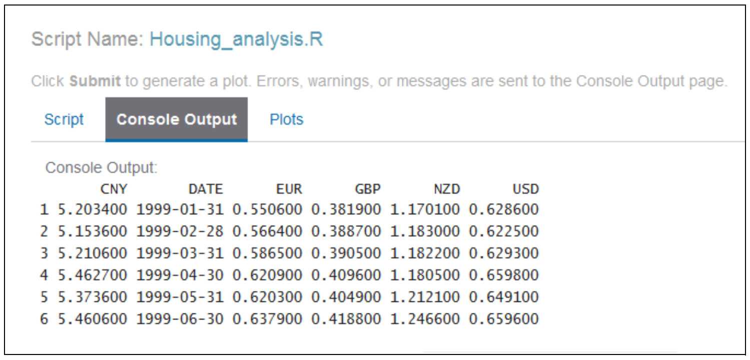
[1] If the user would like to view the data produced by the commands, the name of the data frame can be rewritten on the next line. By submitting this new script, the data is now shown in the console output.

The full script is now shown in Example 9-2. There is a section called R script variables, which is used to define all variables to control different aspects of the R script. These variables are predominately binary, such as TRUE or FALSE. The code in red in Example 9-2 is used to view the first 10 rows of each data set. The console only shows the output from the last head command, so it is necessary to comment out the data sets that are not required. In Example 9-2, the exchange rate is displayed because both the housing prices and rba cash rate head commands are commented out. If all of the head commands were in the code, the exchange rate would be shown since it is the last head command in the code.

Example 9-2 Viewing the top 10 rows of data using the head command

```
library(ibmdbR)

##----------------------------------------------Connect to database
mycon <- idaConnect("BLUDB", "", "")
idaInit(mycon)

##----------------------------------------------R Script variables

view_data = TRUE

##----------------------------------------------Import Data into R

housing_prices <- as.data.frame(ida.data.frame('"DASH100087"."HOUSES_BY_SUBURB"')[
,c('LOCALITY', 'P_2004', 'P_2005', 'P_2006', 'P_2007', 'P_2008', 'P_2009',
'P_2010', 'P_2011', 'P_2012', 'P_2013', 'P_2014', 'P_2015')])

rba_cash_rate <- as.data.frame(ida.data.frame('"DASH100087"."RBA_CASH_RATE"')[
,c('CASH_RATE_TARGET', 'DATE')])

exchange_rate <- as.data.frame(ida.data.frame('"DASH100087"."EXCHANGERATE_VO"')[
,c('CNY', 'DATE', 'EUR', 'GBP', 'NZD', 'USD')])

##--------------------------------------------View Data

if(view_data) {
##head(housing_prices)
##head(rba_cash_rate)
head(exchange_rate)
}
```

The result of submitting the code in Example 9-2 is shown in Figure 9-21 on page 269.



Script Name: Housing_analysis.R

Click **Submit** to generate a plot. Errors, warnings, or messages are sent to the Console Output page.

Script | **Console Output** | Plots

Console Output:
```
       CNY       DATE      EUR      GBP      NZD      USD
1 5.203400 1999-01-31 0.550600 0.381900 1.170100 0.628600
2 5.153600 1999-02-28 0.566400 0.388700 1.183000 0.622500
3 5.210600 1999-03-31 0.586500 0.390500 1.182200 0.629300
4 5.462700 1999-04-30 0.620900 0.409600 1.180500 0.659800
5 5.373600 1999-05-31 0.620300 0.404900 1.212100 0.649100
6 5.460600 1999-06-30 0.637900 0.418800 1.246600 0.659600
```

Figure 9-21 Viewing data in R using the head command. Here the first 10 rows of the exchange rate data are shown

9.3.4 Cleaning and preparing the data in R

In this section, we cover the steps for cleaning and preparing the data in R.

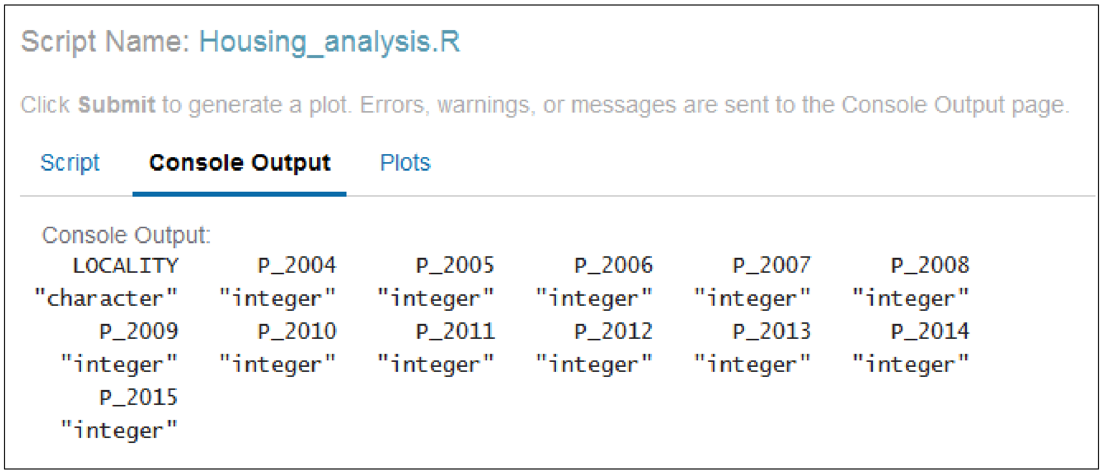
Checking data types in R

The first step to get the data ready for modeling is to check that the data types are correct. The data types can be checked by using the code shown in Example 9-3.

Example 9-3 Checking the data types of the imported R data

```
sapply(housing_prices,class)
```

The function *sapply* applies the function *class* along all columns in the data set housing_prices. This process can be repeated for all the data frames that are imported. For this case, the results are shown in Figure 9-22.



Script Name: Housing_analysis.R

Click **Submit** to generate a plot. Errors, warnings, or messages are sent to the Console Output page.

Script | **Console Output** | Plots

Console Output:
```
   LOCALITY      P_2004      P_2005      P_2006      P_2007      P_2008 
"character"   "integer"   "integer"   "integer"   "integer"   "integer" 
     P_2009      P_2010      P_2011      P_2012      P_2013      P_2014 
  "integer"   "integer"   "integer"   "integer"   "integer"   "integer" 
     P_2015 
  "integer" 
```

Figure 9-22 The result from using the sapply function with class. The results shown in this figure are for the types of the housing price data

Repeating this process for the cash rate and exchange rate data shows that the RBA cash rate data type and date are character types, which will need to be changed to numeric values and dates, respectively. For the exchange rate data, the class for all the columns is character. The exchange rate needs to be converted to numeric and the date column to an R date.

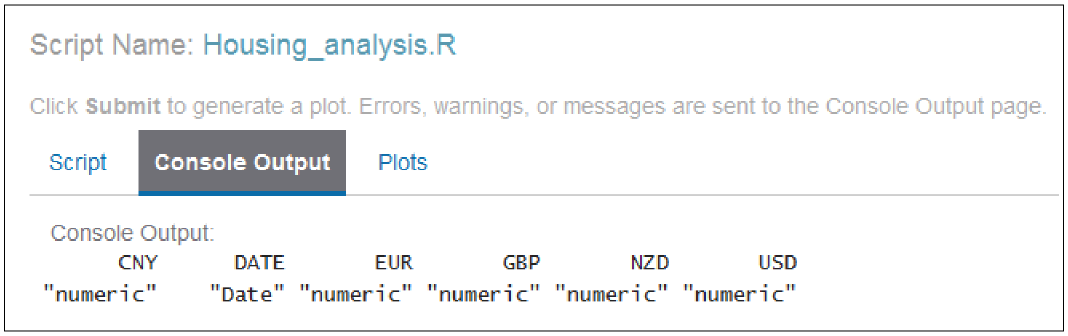
Converting data types in R

To convert data types in R, the **as** command is used. See Example 9-4. For the cash rate data, only two variables need to be converted. However, for the exchange rate six column types need to be converted. Therefore, a "for" loop is used to loop through all columns and the type of the column is changed based on the requirements. In this case, the date column, which is column two, needs to be converted to a date and all other columns need to be converted to numeric.

Example 9-4 Converting character to numeric and date in R

```
## ---------------Convert character data to numeric or date type where necessary
rba_cash_rate[,"CASH_RATE_TARGET"]<-as.numeric(rba_cash_rate[,"CASH_RATE_TARGET"])

rba_cash_rate[,'DATE']<-as.Date(rba_cash_rate[,'DATE'],"%d/%m/%Y")

for(i in 1:6){
   if(i==2){
         exchange_rate[,i] <- as.Date(exchange_rate[,2])
    } else{
         exchange_rate[, i] <- as.numeric(exchange_rate[,i])
   }
}
```

It is worthwhile ensuring that the code has correctly converted the data types. This can be achieved by repeating the code in Example 9-3 on page 269. The resulting class types for the exchange rate are shown in Figure 9-23.


Script Name: Housing_analysis.R
Click Submit to generate a plot. Errors, warnings, or messages are sent to the Console Output page.
Script
Console Output
Plots
Console Output:

```
      CNY        DATE        EUR        GBP        NZD        USD
"numeric"      "Date" "numeric" "numeric" "numeric" "numeric"
```

Figure 9-23 Result from data type conversion

Aggregating data

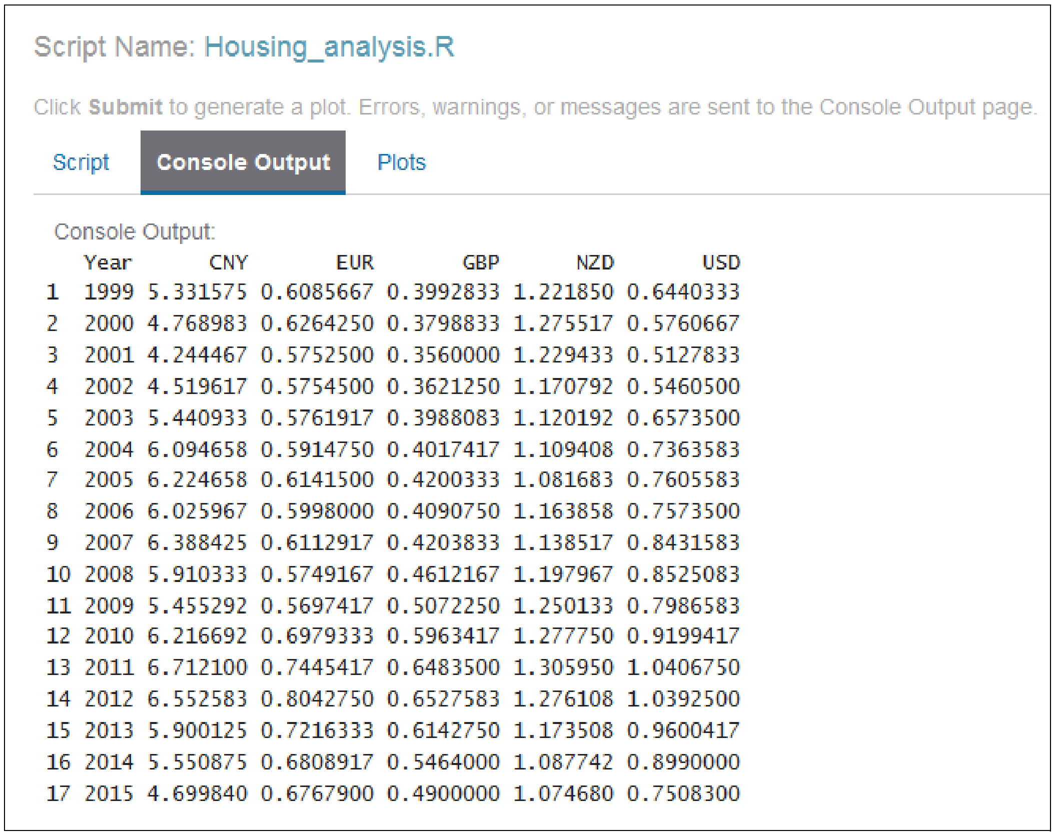
To prepare the data for the modeling phase, it is necessary to ensure that the data is compatible. For example, in this case housing prices are provided yearly, but the cash rate and exchange rates are provided monthly. For this analysis, the average of the exchange rate and interest rate for the year is used to model the average house price in Australia.

Example 9-5 on page 271 shows the process to convert monthly rates to a yearly average rate. First, the year of each date is extracted and inserted into a new variable in the data frame called **Year**. This is done by using the **format** command. The aggregate function is then used to convert the monthly rate into a yearly rate by creating a formula based on the year, and assigning it to the mean function.

Example 9-5 Aggregating monthly data to yearly data

```
##------------Housing prices are yearly, exchange rates and the cash rate are
##----------monthly. Average the monthly rates to a yearly average rate

rba_cash_rate$Year <- format(rba_cash_rate$DATE,"%Y")

rba_cash_rate_yearly<-aggregate(CASH_RATE_TARGET~Year, rba_cash_rate,mean)

exchange_rate$Year <- format(exchange_rate$DATE,"%Y")

exch_year_avg<-aggregate(cbind(CNY,  EUR, GBP, NZD, USD)~Year, exchange_rate,mean)
```

The resulting data for the exchange rate aggregation is shown in Figure 9-24. The results show the year and the corresponding average exchange rate for the year for all the considered currencies.



Script Name: Housing_analysis.R

Click **Submit** to generate a plot. Errors, warnings, or messages are sent to the Console Output page.

Script | **Console Output** | Plots

Console Output:

```
   Year      CNY       EUR       GBP      NZD       USD
1  1999 5.331575 0.6085667 0.3992833 1.221850 0.6440333
2  2000 4.768983 0.6264250 0.3798833 1.275517 0.5760667
3  2001 4.244467 0.5752500 0.3560000 1.229433 0.5127833
4  2002 4.519617 0.5754500 0.3621250 1.170792 0.5460500
5  2003 5.440933 0.5761917 0.3988083 1.120192 0.6573500
6  2004 6.094658 0.5914750 0.4017417 1.109408 0.7363583
7  2005 6.224658 0.6141500 0.4200333 1.081683 0.7605583
8  2006 6.025967 0.5998000 0.4090750 1.163858 0.7573500
9  2007 6.388425 0.6112917 0.4203833 1.138517 0.8431583
10 2008 5.910333 0.5749167 0.4612167 1.197967 0.8525083
11 2009 5.455292 0.5697417 0.5072250 1.250133 0.7986583
12 2010 6.216692 0.6979333 0.5963417 1.277750 0.9199417
13 2011 6.712100 0.7445417 0.6483500 1.305950 1.0406750
14 2012 6.552583 0.8042750 0.6527583 1.276108 1.0392500
15 2013 5.900125 0.7216333 0.6142750 1.173508 0.9600417
16 2014 5.550875 0.6808917 0.5464000 1.087742 0.8990000
17 2015 4.699840 0.6767900 0.4900000 1.074680 0.7508300
```

Figure 9-24 Aggregated exchange rate data

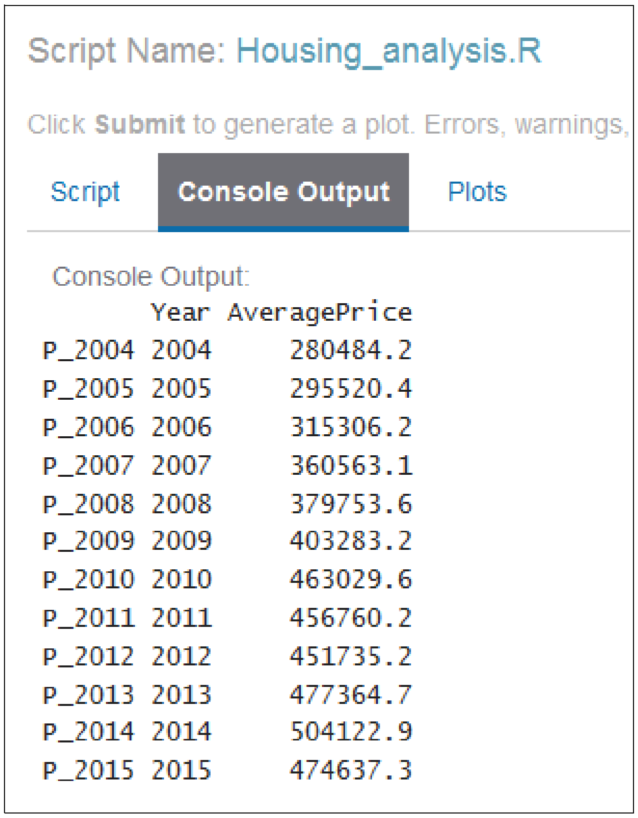
For simplicity, the model developed is for the average housing price in Victoria, and not for the average housing price per suburb. Here it is assumed that the average housing price in Victoria is the average of all the locality housing prices. This assumption is not accurate, but for a stylized model it is adequate.

To make the data for the average house price a similar format to the exchange rate and the RBA cash rate target, a new dummy data frame *df_house_sum* is created and the locality data is removed from this new data frame. The new data frame also has all zero values replaced by NAs so that they will not be used to calculate the mean of each year. A new data frame *avg_house_price* is then created with the year and average house price per year. This process is shown in Example 9-6.

Example 9-6 Aggregating house prices from per suburb to the Victoria region

```
##------Compare rates to average housing price for all suburbs in a given year

df_house_sum <- sapply(housing_prices[,-1], function(x) ifelse(x==0,NA,x))

avg_house_price <-
data.frame('Year'=2004:2015,'AveragePrice'=apply(df_house_sum,2,function(x)
mean(x, na.rm=TRUE)))

remove(df_house_sum)
```

The result from Example 9-6 is shown in Figure 9-25.


Script Name: Housing_analysis.R
Click Submit to generate a plot. Errors, warnings,
Script
Console Output
Plots

```
Console Output:
       Year AveragePrice
P_2004 2004     280484.2
P_2005 2005     295520.4
P_2006 2006     315306.2
P_2007 2007     360563.1
P_2008 2008     379753.6
P_2009 2009     403283.2
P_2010 2010     463029.6
P_2011 2011     456760.2
P_2012 2012     451735.2
P_2013 2013     477364.7
P_2014 2014     504122.9
P_2015 2015     474637.3
```

Figure 9-25 Aggregated house prices in the Victoria region

9.4 Developing a model using dashDB Analytics with R

In this section, we cover the steps for developing a model using dashDB Analytics with R.

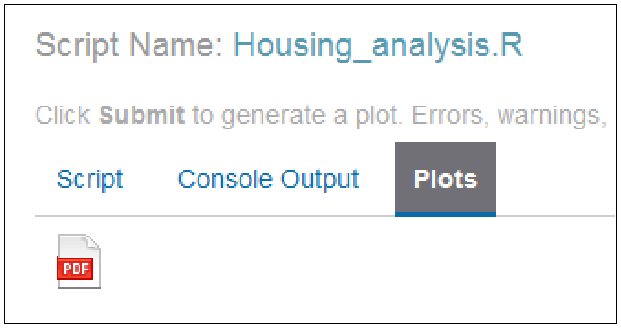
9.4.1 Plotting variables to determine validity of data

Before beginning the model build, the user should determine whether the prepared data makes sense. A good approach for doing this with housing prices and exchange and cash rates is to plot the time series data. See Example 9-7.

Example 9-7 Plotting data in R

```
plot(avg_house_price$Year,avg_house_price$AveragePrice,main="Average House Price
in Victoria per Year",xlab="Year",ylab="Average Price",type="l")


plot(rba_cash_rate_yearly$Year,rba_cash_rate_yearly$CASH_RATE_TARGET,main="Average
Yearly RBA Cash Rate Target",xlab="Year",ylab="Average RBA Cash Rate
Target",type="l")

 plot(exch_year_avg$Year,exch_year_avg$CNY,main="Average Yearly Exchange Rate
($1AUD)",xlab="Year",ylab="Average Chinese Exchange Rate",type="l")
 plot(exch_year_avg$Year,exch_year_avg$EUR,main="Average Yearly Exchange Rate
($1AUD)",xlab="Year",ylab="Average Euro Exchange Rate",type="l")
 plot(exch_year_avg$Year,exch_year_avg$GBP,main="Average Yearly Exchange Rate
($1AUD)",xlab="Year",ylab="Average Britain Exchange Rate",type="l")
 plot(exch_year_avg$Year,exch_year_avg$NZD,main="Average Yearly Exchange Rate
($1AUD)",xlab="Year",ylab="Average New Zealand Exchange Rate",type="l")
 plot(exch_year_avg$Year,exch_year_avg$USD,main="Average Yearly Exchange Rate
($1AUD)",xlab="Year",ylab="Average United States Exchange Rate",type="l")
```

Example 9-7 demonstrates some code that can be used to plot data in R. The code specifies the *x* and *y* variables, as well as the plot title, x and y label, and the plot type. When running this code in dashDB, a pdf is generated that is accessible in the Plots section. An example of the output produced is shown in Figure 9-26. The figures can be accessed by opening the pdf generated.


Script Name: Housing_analysis.R

Click **Submit** to generate a plot. Errors, warnings,

Figure 9-26 The output from generating a plot in dashDB using R

An example of the plots generated is demonstrated in Figure 9-27.

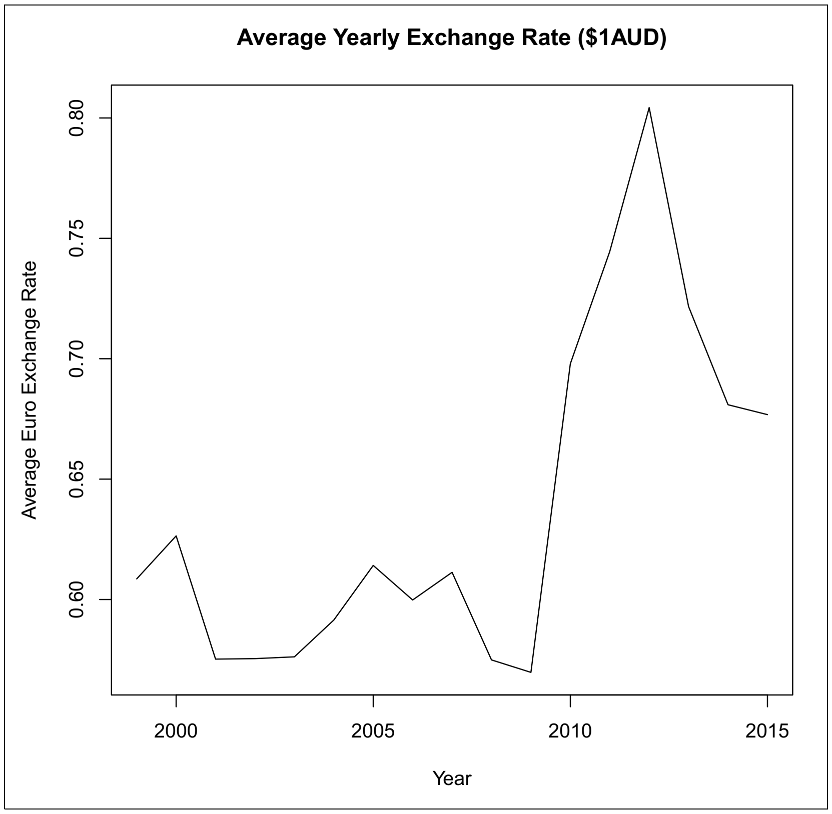


Figure 9-27 Average AUD to EURO yearly exchange rate

The data for the housing prices, exchange rate, and RBA cash rate makes sense. One aspect that is not considered, but should be used to adjust prices in a time series, is the inflation rate. However, in this case the model is stylized and is an example of the process rather than being indicative. Therefore, the affect of inflation on housing prices is ignored.

9.4.2 Determine whether the model predictors appear to be correlated to the variable being predicted

To determine which variables are predictive of housing prices, it is a good starting point to plot the variable that is thought to be predictive to the observation. In this example, the average housing price is the observation, and the RBA cash rate target and various exchange rates are thought to be predictive.

Example 9-8 on page 275 demonstrates the code to graph the relationships between the predictors and the observation. Here both the exchange rate and RBA cash rate target are subset to the period 2004 - 2015 to match the duration of housing data available.

Example 9-8 Graphing relationships between predictors and the observation

```
##--------------------Do basic graphing to see if a relationship exists

exch_year_tmp <- exch_year_avg[exch_year_avg$Year>2003,]

rba_cash_tmp <- rba_cash_rate_yearly[rba_cash_rate_yearly$Year>2003,]

plot(rba_cash_tmp$CASH_RATE_TARGET,avg_house_price$AveragePrice,main="Cash Rate
Target compared to the Average House Price in Victoria",xlab="RBA Cash Rate
Target",ylab="Average Price")

plot(exch_year_tmp$CNY,avg_house_price$AveragePrice,main="Average Victorian House
Price to the AUD to CNY Exchange Rate",xlab="AUD to CNY Exchange
Rate",ylab="Average Price")

plot(exch_year_tmp$EUR,avg_house_price$AveragePrice,main="Average Victorian House
Price to the AUD to EURO Exchange Rate",xlab="AUD to Euro Exchange
Rate",ylab="Average Price")

plot(exch_year_tmp$GBP,avg_house_price$AveragePrice,main="Average Victorian House
Price to the AUD to GBP Exchange Rate",xlab="AUD to GBP Exchange
Rate",ylab="Average Price")

plot(exch_year_tmp$NZD,avg_house_price$AveragePrice,main="Average Victorian House
Price to the AUD to NZD Exchange Rate",xlab="AUD to NZD Exchange
Rate",ylab="Average Price")

plot(exch_year_tmp$USD,avg_house_price$AveragePrice,main="Average Victorian House
Price to the AUD to USD Exchange Rate",xlab="AUD to USD Exchange
Rate",ylab="Average Price")
```

Figure 9-28 demonstrates the plot for the average house price in Victoria compared to the RBA cash rate target. Apparently, there is a negative relationship between these two variables. That is, as the RBA cash rate increases the average house price in Victoria decreases. This corresponds, in general, with the logic that as interest rates decrease housing prices increase due to increased demand.

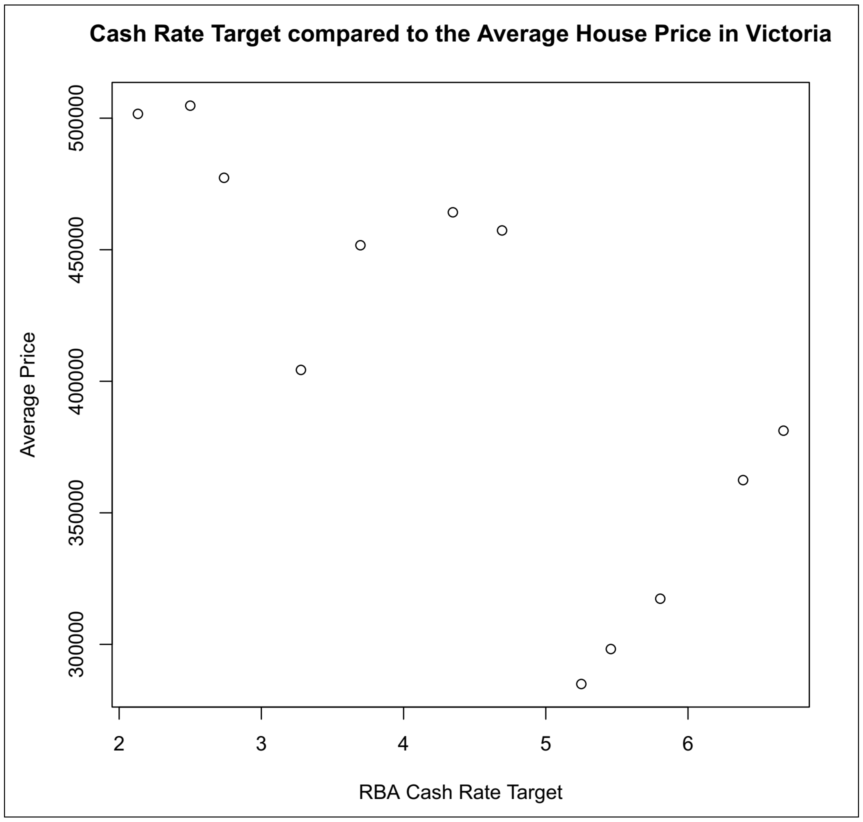


Figure 9-28 A plot of the average house price in Victoria compared to the average RBA cash rate at the same time

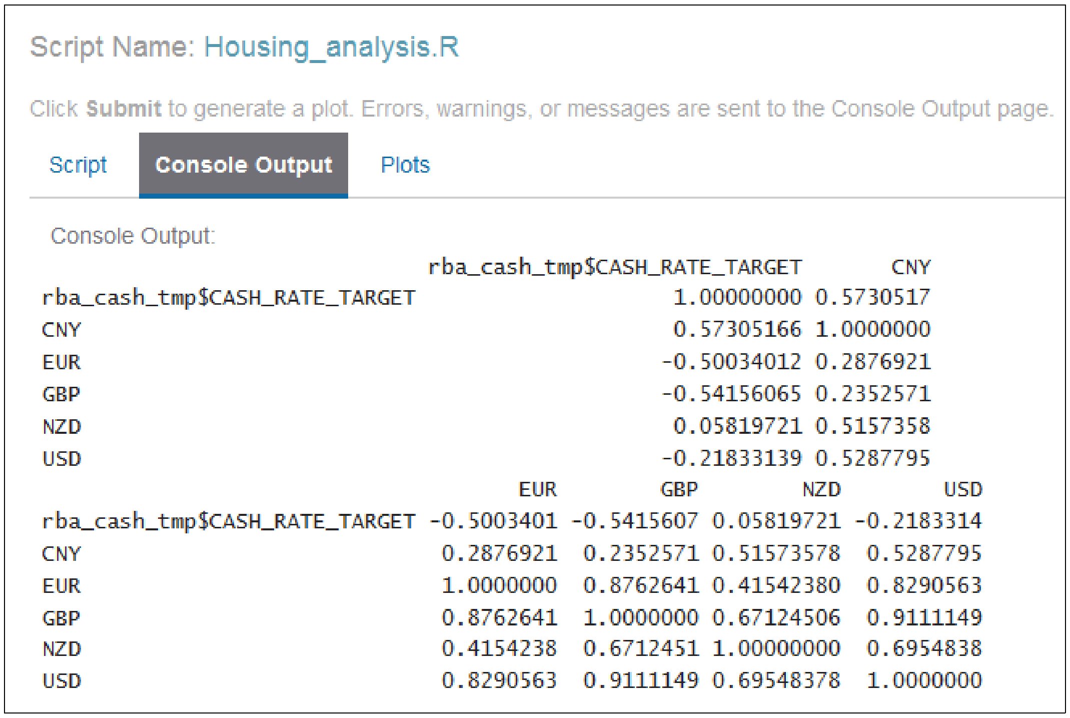
9.4.3 Determine correlation between predictors

Before using the relationships observed in the data, it is worthwhile determining whether two of the predictors considered are correlated. Correlated predictors make the model less accurate due to underestimating the effect of the critical predictor. The code to determine the correlation between all predictors is shown in Example 9-9.

Example 9-9 Code for determining correlations between predictors

```
cor(cbind(rba_cash_tmp$CASH_RATE_TARGET,exch_year_tmp[,-1]))
```

The result for this correlation test is shown in Figure 9-29. These results show that the British Pound, Euro, and US Dollar exchange rates with Australia are highly correlated. Based on the correlation results and the plots to determine relationships between variables, the model that is built will be predicting house prices based on the RBA cash rate target and the British Pound.


Script Name: Housing_analysis.R
Click Submit to generate a plot. Errors, warnings, or messages are sent to the Console Output page.
Script
Console Output
Plots
Console Output:

```
                              rba_cash_tmp$CASH_RATE_TARGET       CNY
rba_cash_tmp$CASH_RATE_TARGET                    1.00000000 0.5730517
CNY                                              0.57305166 1.0000000
EUR                                             -0.50034012 0.2876921
GBP                                             -0.54156065 0.2352571
NZD                                              0.05819721 0.5157358
USD                                             -0.21833139 0.5287795
                                     EUR        GBP        NZD        USD
rba_cash_tmp$CASH_RATE_TARGET -0.5003401 -0.5415607 0.05819721 -0.2183314
CNY                            0.2876921  0.2352571 0.51573578  0.5287795
EUR                            1.0000000  0.8762641 0.41542380  0.8290563
GBP                            0.8762641  1.0000000 0.67124506  0.9111149
NZD                            0.4154238  0.6712451 1.00000000  0.6954838
USD                            0.8290563  0.9111149 0.69548378  1.0000000
```

Figure 9-29 Correlation results for housing price predictors

9.4.4 Develop a model of housing prices

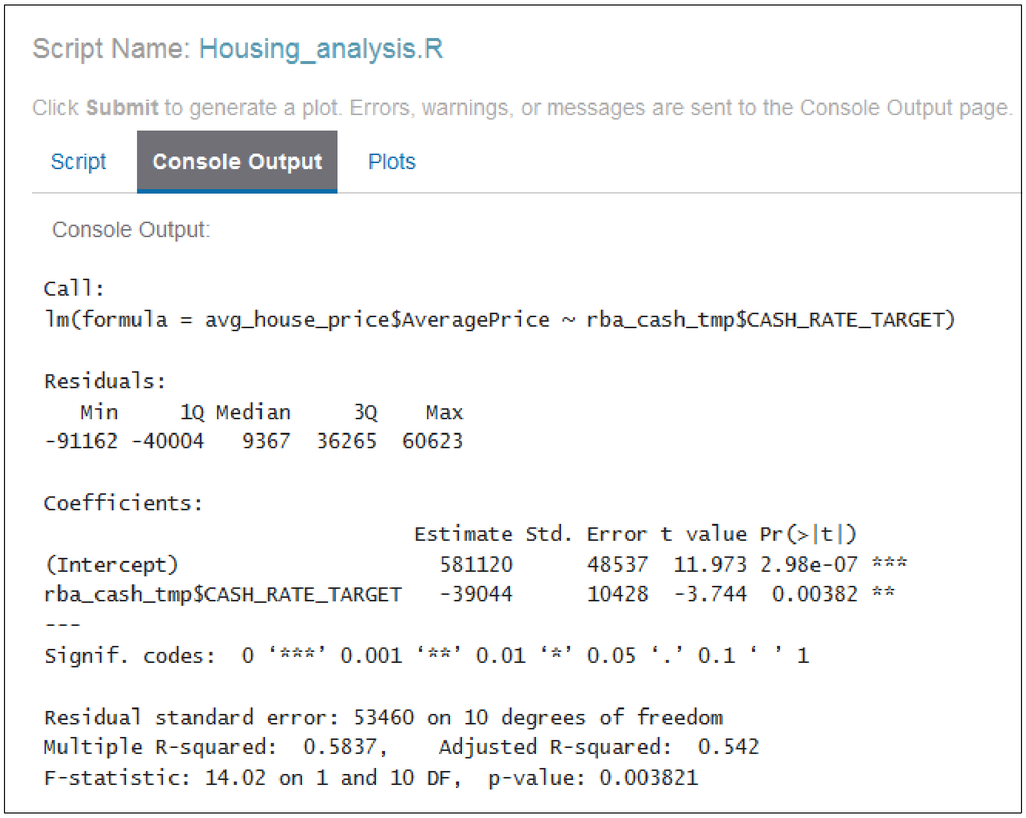
To develop a model of housing prices, a single predictor (the RBA cash rate target) is initially used. Then, the additional predictor, the British Pound, is added to the model to determine what the improvement in predictive power of the model is by adding the British Pound into the model as a predictor is.

For this example, the relationship between the RBA cash rate and housing prices appears to be linear. Therefore, initially linear regression is used, with linear predictors. This is implemented in R using the code shown in Example 9-10.

Example 9-10 R code to develop a linear model of housing prices with the RBA cash rate target as a predictor

```
##---------------------Develop model based on interest rate

housing_model <- lm(avg_house_price$AveragePrice~rba_cash_tmp$CASH_RATE_TARGET)

summary(housing_model)
```

The result of the summary command is shown in Figure 9-30. The results show that both the gradient and intercept of the predicted model were highly significant. The results also theoretically state that with an RBA cash rate target of 0%, the average house price in Victoria would be $581,120, and that with each one point increase in the cash rate target the average value of houses in Victoria decreases by $39,044.


Script Name: Housing_analysis.R
Click Submit to generate a plot. Errors, warnings, or messages are sent to the Console Output page.
Script
Console Output
Plots
Console Output:

```
Call:
lm(formula = avg_house_price$AveragePrice ~ rba_cash_tmp$CASH_RATE_TARGET)

Residuals:
   Min     1Q Median     3Q    Max
-91162 -40004   9367  36265  60623

Coefficients:
                              Estimate Std. Error t value Pr(>|t|)
(Intercept)                     581120      48537  11.973 2.98e-07 ***
rba_cash_tmp$CASH_RATE_TARGET   -39044      10428  -3.744  0.00382 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 53460 on 10 degrees of freedom
Multiple R-squared:  0.5837,    Adjusted R-squared:  0.542
F-statistic: 14.02 on 1 and 10 DF,  p-value: 0.003821
```

Figure 9-30 The output of the summary command on a linear regression model

After developing a model, check that the model fits the data. This can be done by using a plot. The code for generating this plot is in Example 9-11.

Example 9-11 Plot model on the same axis as the data used to create the model

```
##-----------------------------Check how linear mode fits the data

plot(rba_cash_tmp$CASH_RATE_TARGET,avg_house_price$AveragePrice)
abline(housing_model)
```

The resulting plot generated from Example 9-11 is demonstrated in Figure 9-31 on page 279.

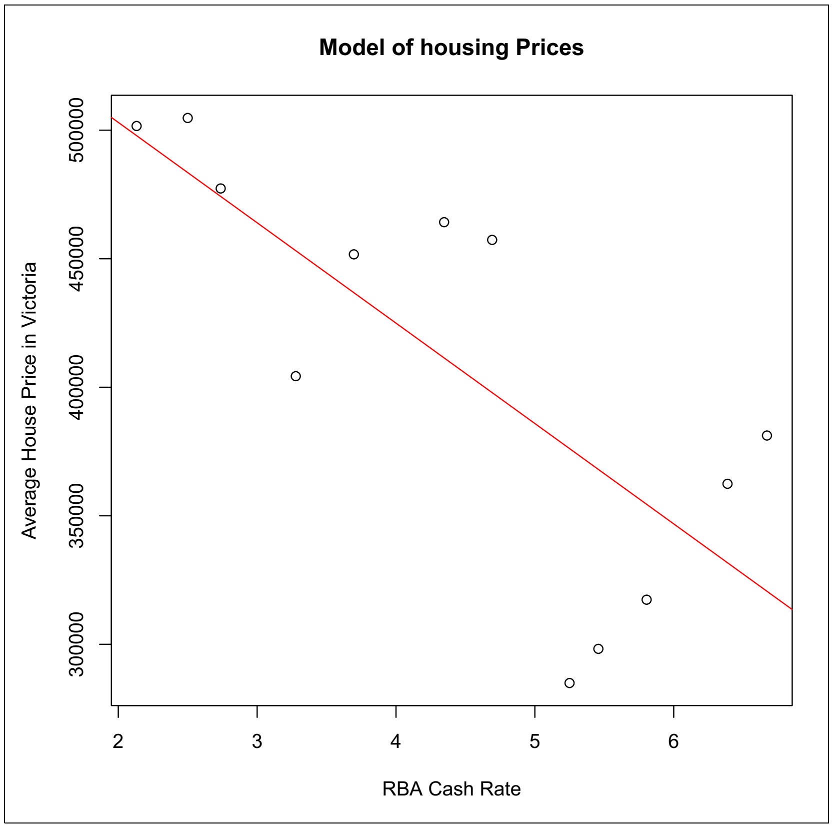


Figure 9-31 Model of the average house price in Victoria

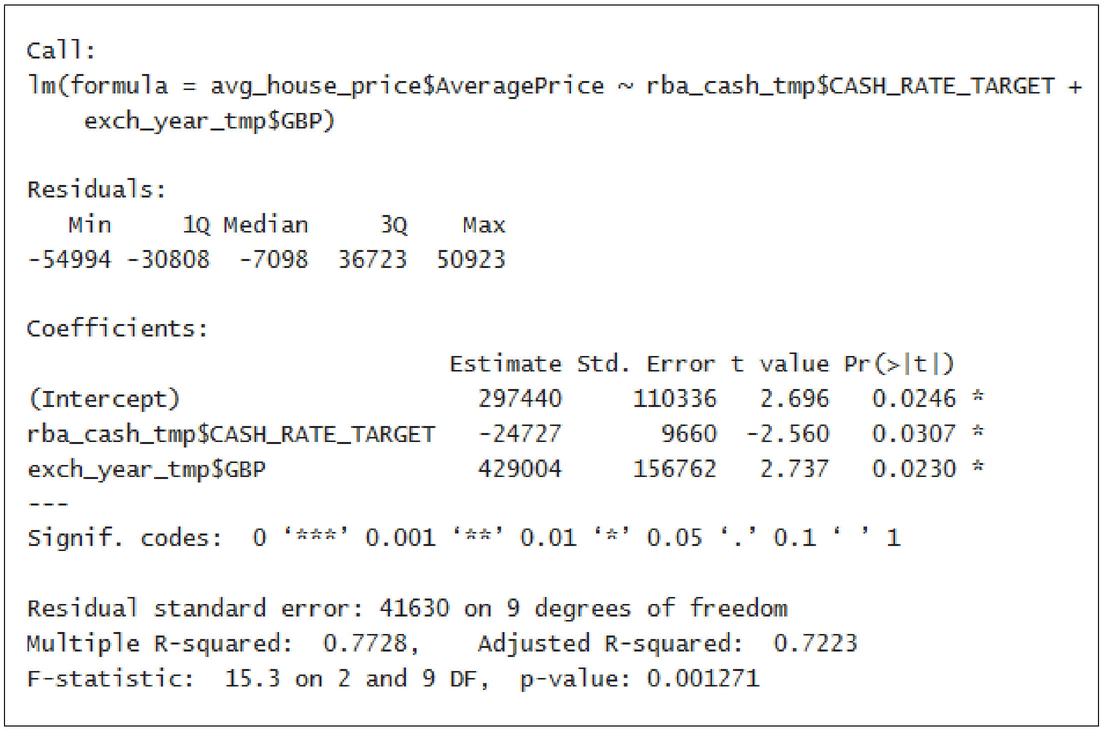
This process can be repeated by using the British Pound as the only predictor. However, for brevity the full model with both the RBA cash rate and the British Pound as predictors is considered next. The code to develop this model is shown in Example 9-12.

Example 9-12 Linear model code for housing prices based on the RBA cash rate and the British Pound to AUD exchange rate

```
##--Develop model based on interest rate and British Pound to AUD exchange rate

housing_model_2 <-
lm(avg_house_price$AveragePrice~rba_cash_tmp$CASH_RATE_TARGET+exch_year_tmp$GBP)

summary(housing_model_2)
```

The results for this model are shown in Figure 9-32. Comparing this result to the result in Figure 9-31 on page 279, it is clear that the residual error has decreased and the adjusted R squared value has increased. This indicates that the new model is better at predicting the average house price in Victoria, compared to the model where the RBA cash rate was the only predictor. Also, the results show that all predictors are significant.

Therefore, the model of average housing prices in Victoria predicts that for each point increase in the RBA cash rate, the average value of a house in Australia decreases by $25,000. And for each 0.01 increase in the British Pound exchange rate with the AUD, the average house price in Australia increases by $4290. This model is now used to trigger real-time events based on the current exchange rate for the AUD to the GBP, and the current RBA cash rate, which is far less dynamic.



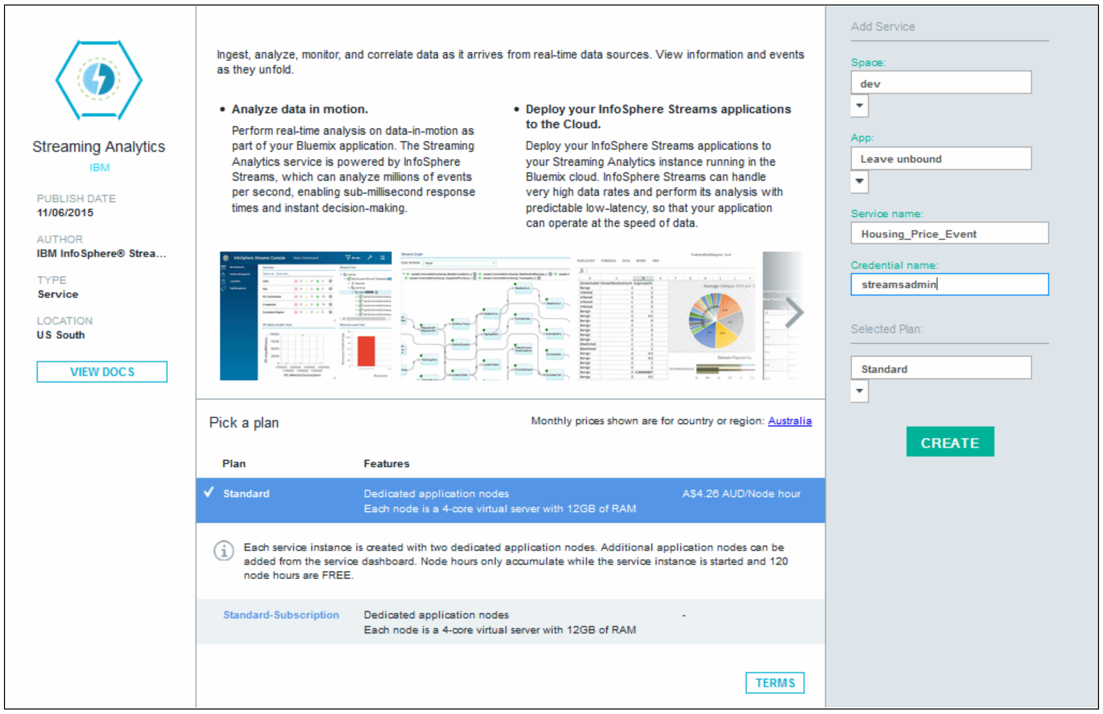
```
Call:
lm(formula = avg_house_price$AveragePrice ~ rba_cash_tmp$CASH_RATE_TARGET +
    exch_year_tmp$GBP)

Residuals:
   Min     1Q Median     3Q    Max
-54994 -30808  -7098  36723  50923

Coefficients:
                              Estimate Std. Error t value Pr(>|t|)
(Intercept)                     297440     110336   2.696   0.0246 *
rba_cash_tmp$CASH_RATE_TARGET   -24727       9660  -2.560   0.0307 *
exch_year_tmp$GBP               429004     156762   2.737   0.0230 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 41630 on 9 degrees of freedom
Multiple R-squared:  0.7728,    Adjusted R-squared:  0.7223
F-statistic:  15.3 on 2 and 9 DF,  p-value: 0.001271
```

Figure 9-32 Model of housing prices based on the RBA cash rate and AUD to British Pound exchange rate

9.5 Configuring an InfoSphere Streams Service on Bluemix

To create a Bluemix streaming analytics service, browse to the Catalog section from the Bluemix main page. In the Catalog section, select the **Data and Analytics** option on the left of the window. The service that is used for the detection of real-time events is the Streaming Analytics Service.

After the streaming analytics service is selected, it needs to be configured. This service is not bound to an application. In the configuration, the service is named Housing_Price_Event and the service is assigned to the user streamsadmin. For this application, a standard plan is used. This plan includes up to 120 hours of usage at no charge. The configuration described is demonstrated in Figure 9-33. After the configuration is complete, the service is created by selecting **Create**.


Streaming Analytics
IBM
PUBLISH DATE
11/06/2015
AUTHOR
IBM InfoSphere® Strea...
TYPE
Service
LOCATION
US South
VIEW DOCS

Ingest, analyze, monitor, and correlate data as it arrives from real-time data sources. View information and events as they unfold.

- Analyze data in motion.
  Perform real-time analysis on data-in-motion as part of your Bluemix application. The Streaming Analytics service is powered by InfoSphere Streams, which can analyze millions of events per second, enabling sub-millisecond response times and instant decision-making.
- Deploy your InfoSphere Streams applications to the Cloud.
  Deploy your InfoSphere Streams applications to your Streaming Analytics instance running in the Bluemix cloud. InfoSphere Streams can handle very high data rates and perform its analysis with predictable low-latency, so that your application can operate at the speed of data.

Pick a plan

Monthly prices shown are for country or region: Australia

Plan	Features	
✓ Standard	Dedicated application nodes Each node is a 4-core virtual server with 12GB of RAM	A$4.26 AUD/Node hour
Standard-Subscription	Dedicated application nodes Each node is a 4-core virtual server with 12GB of RAM	-

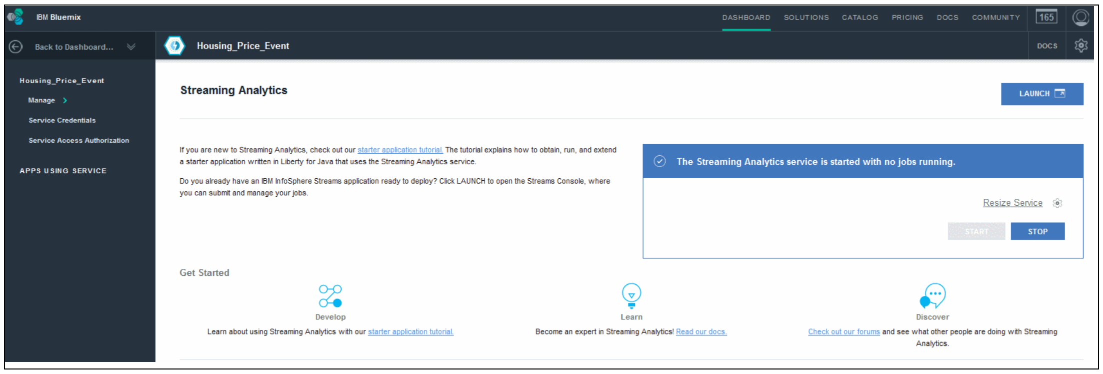
Each service instance is created with two dedicated application nodes. Additional application nodes can be added from the service dashboard. Node hours only accumulate while the service instance is started and 120 node hours are FREE.

TERMS

Add Service
Space: dev
App: Leave unbound
Service name: Housing_Price_Event
Credential name: streamsadmin
Selected Plan: Standard
CREATE

Figure 9-33 Bluemix Streaming Analytics service configuration page

After selecting CREATE, the page to manage the streamlining analytics service is displayed. This page shows that the service is started, and will be using part of the 120 no-charge hours provided by Bluemix. The service only needs to be running when a streams job is running. Because this is not the case, it is recommended that the service be stopped. Figure 9-34 shows the Manage page for the streams service. Use the **Start** and **Stop** button to run or stop the streaming analytics service. In the Service Credentials page, new users can be added to the service, and in the **Service Access Authorization** option, other services can be provided permission to access the streaming analytics service.



Streaming Analytics

If you are new to Streaming Analytics, check out our starter application tutorial. The tutorial explains how to obtain, run, and extend a starter application written in Liberty for Java that uses the Streaming Analytics service.

Do you already have an IBM InfoSphere Streams application ready to deploy? Click LAUNCH to open the Streams Console, where you can submit and manage your jobs.

The Streaming Analytics service is started with no jobs running.

Get Started

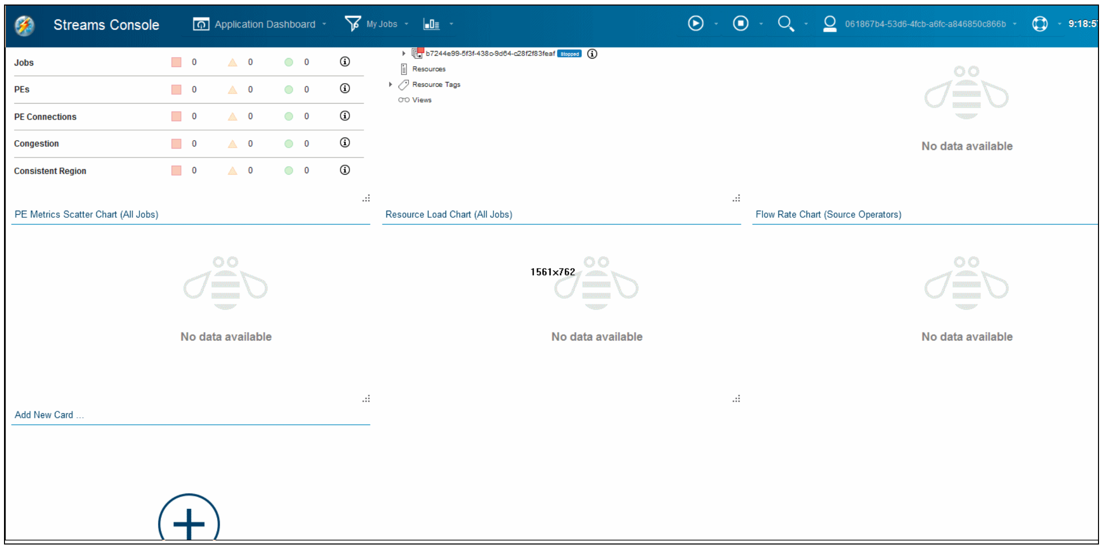
Learn about using Streaming Analytics with our starter application tutorial.

Become an expert in Streaming Analytics! Read our docs.

Check out our forums and see what other people are doing with Streaming Analytics.

Figure 9-34 Streams service manage page

Next, launch the application by clicking **Launch**. This opens the application dashboard in the Streams Console as shown in Figure 9-35. The dashboard shows the status of all jobs in the upper left, all available resources in the middle card, and has a few empty cards that are populated when a streams job is running.



Jobs	🟥 0	△ 0	● 0	ⓘ
PEs	🟥 0	△ 0	● 0	ⓘ
PE Connections	🟥 0	△ 0	● 0	ⓘ
Congestion	🟥 0	△ 0	● 0	ⓘ
Consistent Region	🟥 0	△ 0	● 0	ⓘ

Figure 9-35 Streams Console application dashboard

9.6 Running InfoSphere Streams locally using VMware

An InfoSphere Streams job comes in the form of a Streams Application Bundle. A streams application cannot be created on Bluemix. The application needs to be created locally using a local version of InfoSphere Streams. Fortunately, there is a no-charge version of InfoSphere Streams that is publicly accessible. The Quick Start edition of Streams can be found at the following site:

http://www.ibm.com/analytics/us/en/technology/stream-computing

When on this site, select **Get Started For Free**. For this example, the virtual machine is used. A virtual machine is a virtual computer that has pre-configured software on it. Select **Download the VMware Image**. To access the software, you need to log in using your IBM ID. A form detailing the purpose of your download opens. Complete the form, read the terms and agreements and accept them, and select **Next**. Download the VMware image for IBM Streams 4.0.1. At the time of writing, Streams 4.1 could not be used because the Bluemix Streaming Analytics service was running Streams 4.0.1.

To virtualize this virtual machine configuration, VMware Workstation Player is required. This software is publicly available and is available at the following site:

https://my.vmware.com/web/vmware/free#desktop_end_user_computing/vmware_workstation_player/12_0

After VMWare Workstation Player is downloaded and installed, you can open the Streams Quick Start Edition virtual machine, and power it on in the VMware Workstation Player. This opens the window shown in Figure 9-36 on page 283.

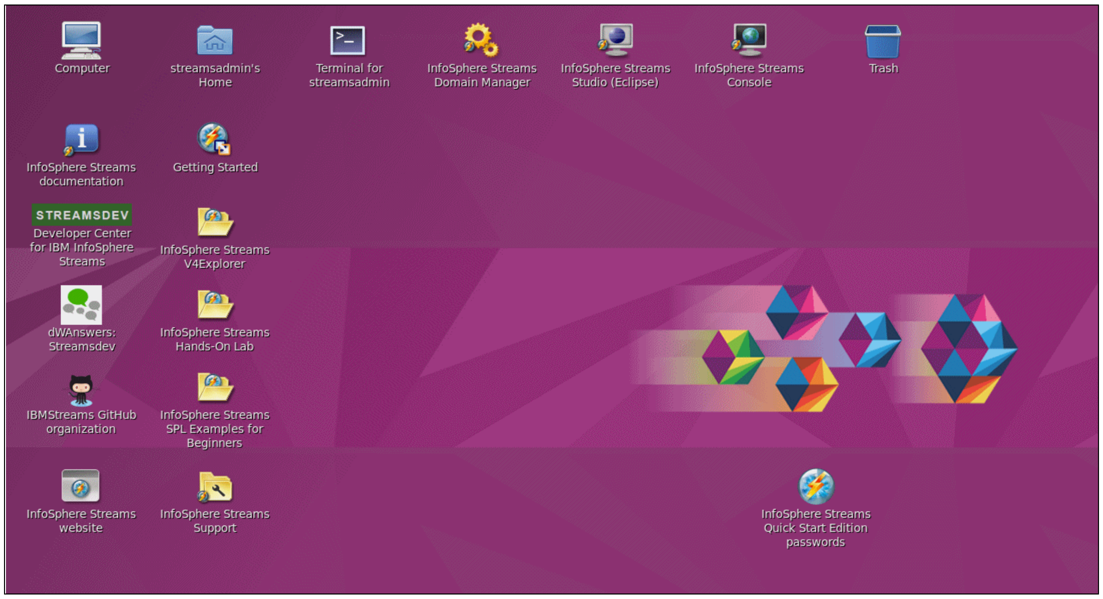

Computer
streamsadmin's Home
Terminal for streamsadmin
InfoSphere Streams Domain Manager
InfoSphere Streams Studio (Eclipse)
InfoSphere Streams Console
Trash
InfoSphere Streams documentation
Getting Started
STREAMSDEV Developer Center for IBM InfoSphere Streams
InfoSphere Streams V4Explorer
dWAnswers: Streamsdev
InfoSphere Streams Hands-On Lab
IBMStreams GitHub organization
InfoSphere Streams SPL Examples for Beginners
InfoSphere Streams website
InfoSphere Streams Support
InfoSphere Streams Quick Start Edition passwords

Figure 9-36 InfoSphere Streams Quick Start Edition virtual machine desktop

9.7 Creating a Streams Application Bundle

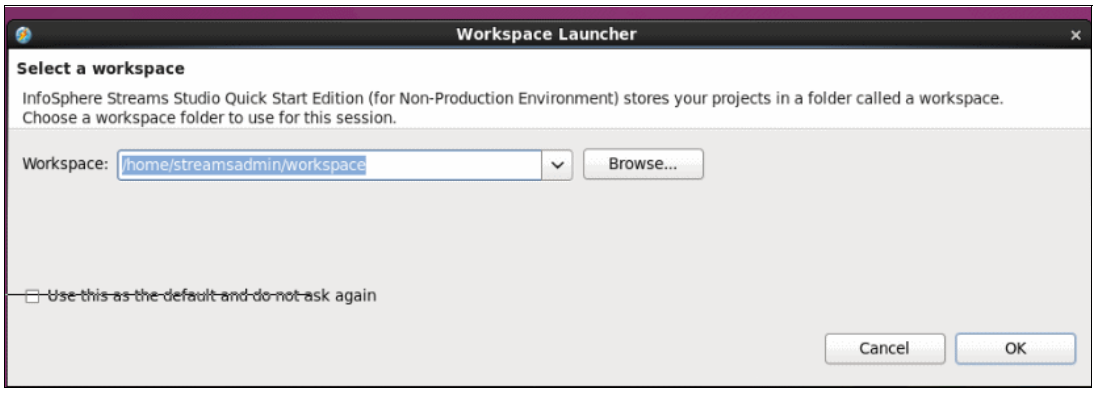
For this application, the requirement is to create a Streams job to run on Bluemix. To do this, a Streams Application Bundle needs to be created locally by using the Streams Quick Start Edition virtual machine. On the desktop of the virtual machine, there is an icon called *InfoSphere Streams Studio (Eclipse)*. This software is used to create a Streams Application Bundle.

The Streams Application Bundle used for the scenario can be found at the following site:

`https://github.com/RichardBalson/Redbooks/tree/master/Bluemix-IntegratingBatchAnalyticsWithRealTimeEvents`

The file is named `Housing.Housing_Price_Event.sab`.

1. When starting this software, a new workplace needs to be defined, as shown in Figure 9-37. For this application, the default location is used.


Workspace Launcher

Select a workspace

InfoSphere Streams Studio Quick Start Edition (for Non-Production Environment) stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: /home/streamsadmin/workspace

Figure 9-37 Streams Studio workplace definition window

2. After defining the workplace, Streams Studio opens. Within Streams Studio, a new project needs to be defined. Select **File** → **New** → **Project**. This opens the window shown in Figure 9-38. Here a new SPL Application Project is created. Select this option and click **Next**.

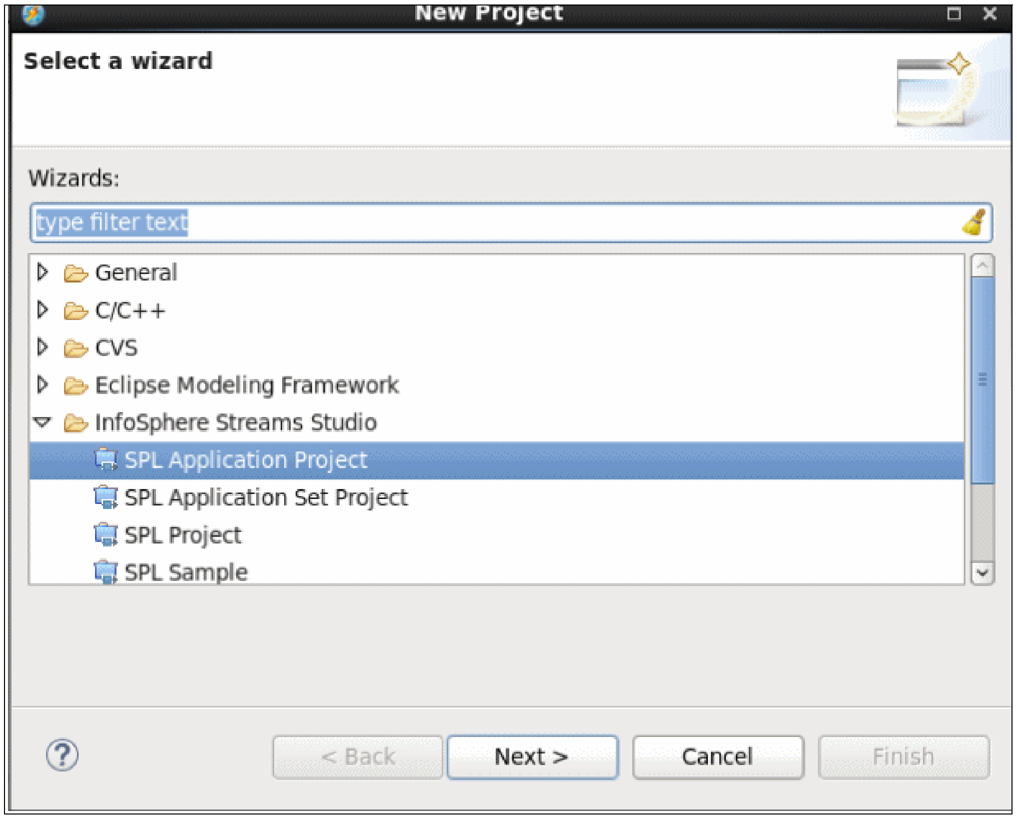
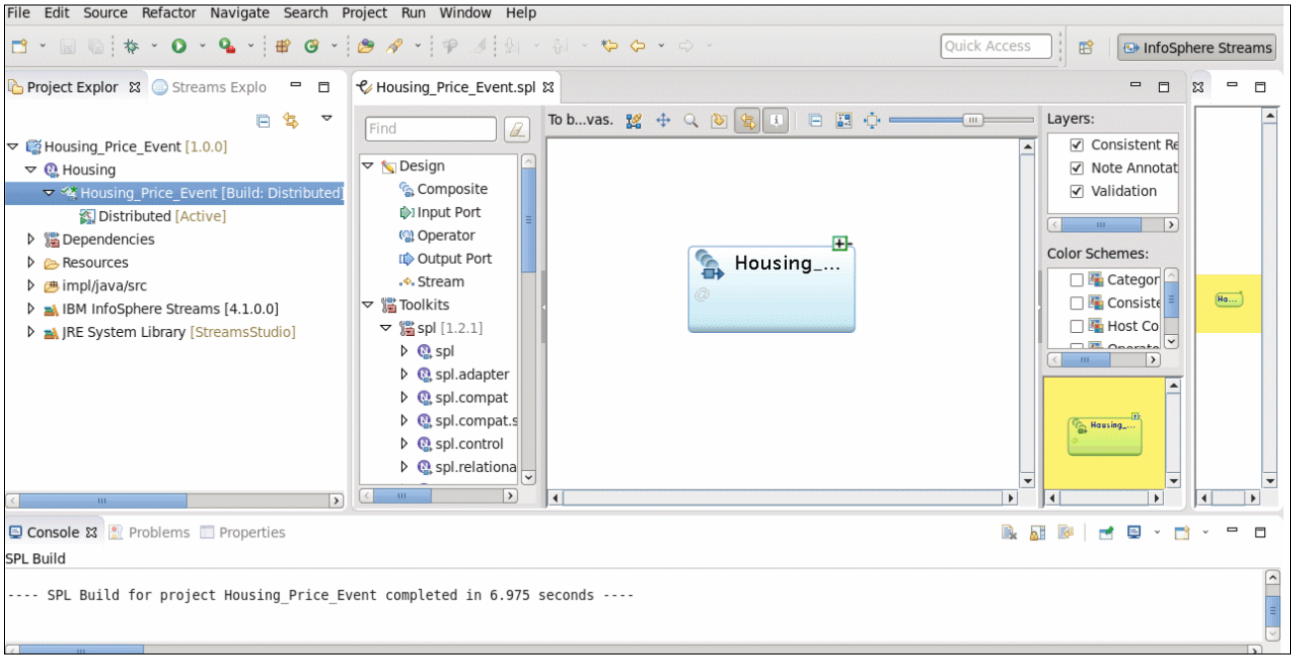


Figure 9-38 Streams Studio New Project Type selection

3. The next window asks for a project name. Name the project Housing_Price_Event. The next page asks what toolkits are necessary for this project. For this particular project, only the standard SPL toolkit is required. Clear all toolkits and select **Finish**. This opens the new project as shown in Figure 9-39 on page 285. The blue box in the middle pane is a composite, and all elements of the application are configured in this composite.

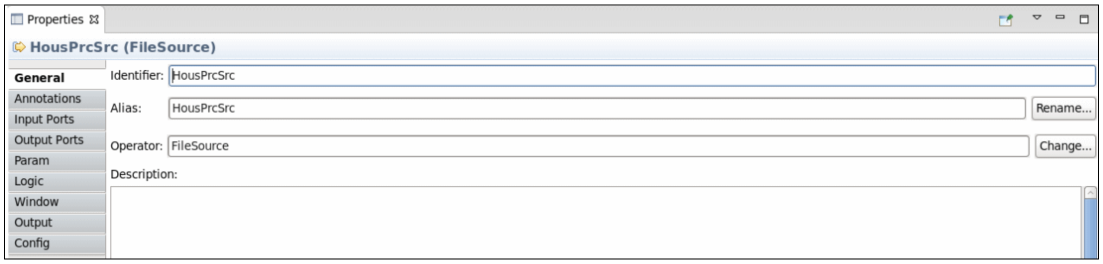


```
---- SPL Build for project Housing_Price_Event completed in 6.975 seconds ----
```

Figure 9-39 SPL project in Streams Studio

Streams file source

For the application, a simulated source is used. Here a csv file is created and used to simulate real-time data. This file needs to be imported into the Streams Application. This can be done by using a file source. The file source can be found in **Toolkit spl** → **spl.adapter** → **FileSource**. Drag a FileSource onto the Housing_Price_Event composite. Then, double-click the file source. This opens the properties page for the file source, as shown in Figure 9-40.



Identifier: HousPrcSrc
Alias: HousPrcSrc
Operator: FileSource
Description:

Figure 9-40 File Source Properties in a Streams Application

Perform the following steps to create a Streams file source:

1. In File Source Properties, under the General tab select **Rename**. Rename the file source to HousingPrcSrc.
2. In the **Output Ports** tab:
   a. Rename the output stream **PredictorData**.
   b. Create four new variables: Date, GBPExch, RBACash, and TriggerLimit. Specify the type of date as **rstring** and all other variables as **float64** (use CTRL+space to get a list of all possible formats).

3. In the **Param** tab:
   a. Select **Add**, and select file and format (these options are not available if they are already in the param section). Then, click **OK**.
   b. Specify the location of the file relative to the Streams Application Bundle in the file value (this is necessary as the Bluemix Streams Service does not have access to the local host, only the data available to the Streams Application). In this case, getApplicationDir()+"/../../etc/TestFile.csv". The **`getApplicationDir()`** function gets the location of the application toolkit directory (this is the same as the **`getThisToolkitDir()`** function in this case, which is two levels up in the folder structure from the root directory, which contains the `etc` folder.
   c. The format value should be csv.
4. Lastly in the **Config** tab:
   a. Select **Add** and select **Placement**. Then, click **OK**.
   b. Specify the value as: hostColocation("host2"). Here host1 can also be used instead of host2 depending on the hostnames that are being used in the Bluemix Streams service. hostcolocation is necessary to ensure that the Bluemix Streams server keeps the file on the same server as the application operators.

FileSource Properties can now be closed.

Creating a custom operator

To use the model developed, it is necessary to create a custom operator to specify the model. This tool can be found under **Toolkits spl** → **spl.utility** → **custom**. Drag the tool **Custom with Input and Output** into the **Housing_Price_Event** composite. The input to the Custom operator is the data extracted from the csv file specified in the file source. To enable this, click and hold the small box on the right side of the file source, then drag the mouse to the small box on the left side of the custom operator. The result of doing this should look like Figure 9-41.

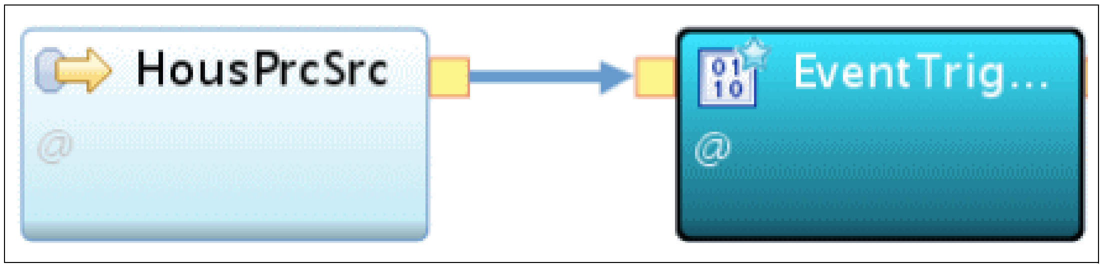


Figure 9-41 Housing_Price_Event composite

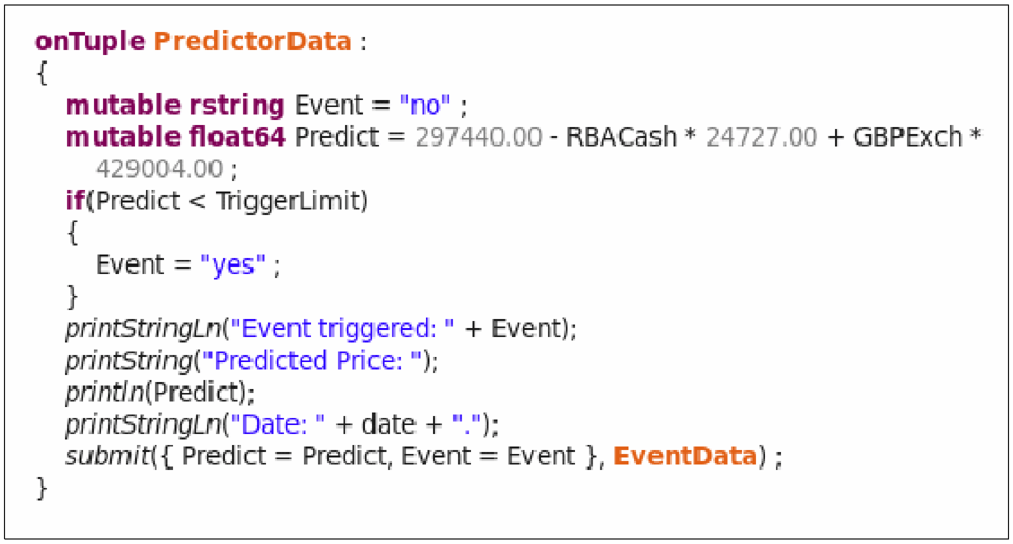
Next, double-click the **Custom** tool. This opens a Properties window that is identical to Figure 9-40 on page 285, but with the Operator specified as Custom.

To create a custom operator, perform the following steps:

1. In the **General** tab, change the name of the custom operator to **EventTrigger**.
2. Check the **Input Ports** tab to verify that the PredictorData stream is in the "Inout streams with schema" section.
3. In the **Output Ports** tab:
   a. Specify the output stream name as **EventData**.
   b. Add two variables **Predict** and **Event** with types **float64** and **rstring**.

4. In the **Logic** tab, copy the code shown in Figure 9-42.
5. Close the custom operator properties.

The code shown in Figure 9-42 predicts the average house price in Victoria based on the model developed in Figure 9-32 on page 280. The predicted house price is then compared to the trigger limit. If the predicted price is less than the trigger limit, the **`Event`** variable is changed to `Yes`. The current date, predicted price, and event status are then printed to the console. Lastly, the predicted price and event status are submitted to the output stream.



```
onTuple PredictorData :
{
  mutable rstring Event = "no" ;
  mutable float64 Predict = 297440.00 - RBACash * 24727.00 + GBPExch *
    429004.00 ;
  if(Predict < TriggerLimit)
  {
    Event = "yes" ;
  }
  printStringLn("Event triggered: " + Event);
  printString("Predicted Price: ");
  println(Predict);
  printStringLn("Date: " + date + ".");
  submit({ Predict = Predict, Event = Event }, EventData) ;
}
```

Figure 9-42 Logic in the custom operator

Streams File Sink

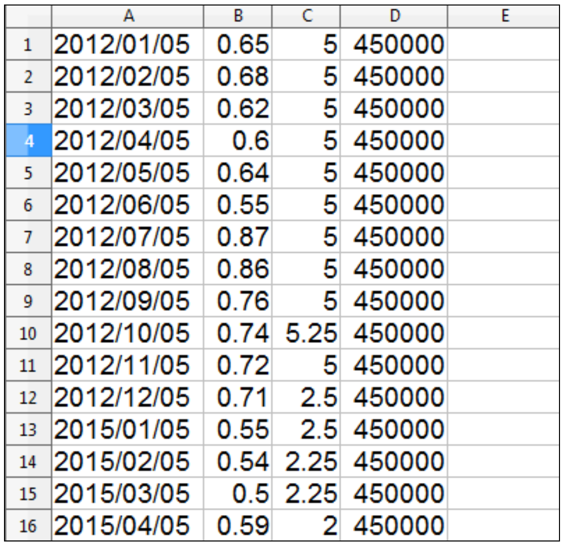
Lastly, a Streams File Sink is added to the composite. The File Sink can be found under **Toolkits spl.adapter** → **FileSink**. Drag the **FileSink** into the **Housing_Price_Event composite**, and connect the output of the custom operator to the input of the FileSink. Double-click **FileSink** to open its properties.

Perform the following steps to create a Streams File Sink:

1. In the **General** tab, rename the file sink **EventSink**.
2. In the **Input Ports** tab, check that EventData is in the “Input streams with schema section.”
3. In the **Param** tab:
   a. Add the File, flush, and format parameters.
   b. Specify the file value as *getThisToolkitDir()+“/../../etc/EventOut.csv”,* flush as 1u, and format as csv.
4. Close **FileSink** and save the project.

The Streams Application File (SAB) can be found in the workspace under `output/Housing_Price_Event/Distributed`. This is the file that needs to be submitted as a job to the streams server.

Before doing this, some test data needs to be created. An example of some test data is shown in Figure 9-43. This csv file needs to be put in the `/etc` folder within the folder where the Streams Application Bundle is located.



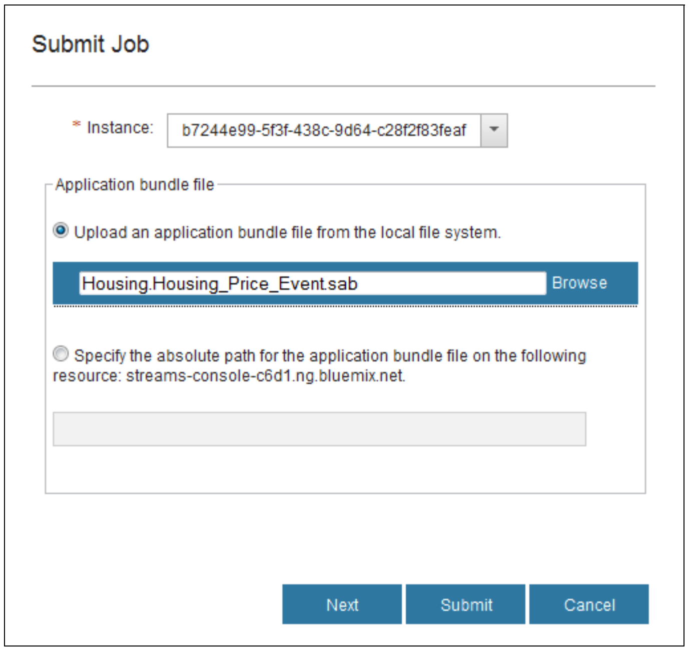
	A	B	C	D	E
1	2012/01/05	0.65	5	450000	
2	2012/02/05	0.68	5	450000	
3	2012/03/05	0.62	5	450000	
4	2012/04/05	0.6	5	450000	
5	2012/05/05	0.64	5	450000	
6	2012/06/05	0.55	5	450000	
7	2012/07/05	0.87	5	450000	
8	2012/08/05	0.86	5	450000	
9	2012/09/05	0.76	5	450000	
10	2012/10/05	0.74	5.25	450000	
11	2012/11/05	0.72	5	450000	
12	2012/12/05	0.71	2.5	450000	
13	2015/01/05	0.55	2.5	450000	
14	2015/02/05	0.54	2.25	450000	
15	2015/03/05	0.5	2.25	450000	
16	2015/04/05	0.59	2	450000	

Figure 9-43 Stream test data

9.8 Real-time event detection in InfoSphere Streams

With the Streams Application Bundle created, the last step required to run the service in real time on the Bluemix service is to submit the job to the cloud. In the Streams Service Manage page (shown in Figure 9-34 on page 281), start the Streams service. When the service has started, launch the Streams service. This opens the Streams console as shown in Figure 9-35 on page 282.

To submit a Streams Application Bundle, the **play button** at the top of the Streams Console near the middle should be selected. Then, click **Submit Job**. This opens a pop-up window where the location of the Streams Application Bundle needs to be specified. For this example, the Streams Application is available locally. Browse to the location of the application bundle and select it. This looks like Figure 9-44 on page 289.


Submit Job

* Instance: b7244e99-5f3f-438c-9d64-c28f2f83feaf

Application bundle file

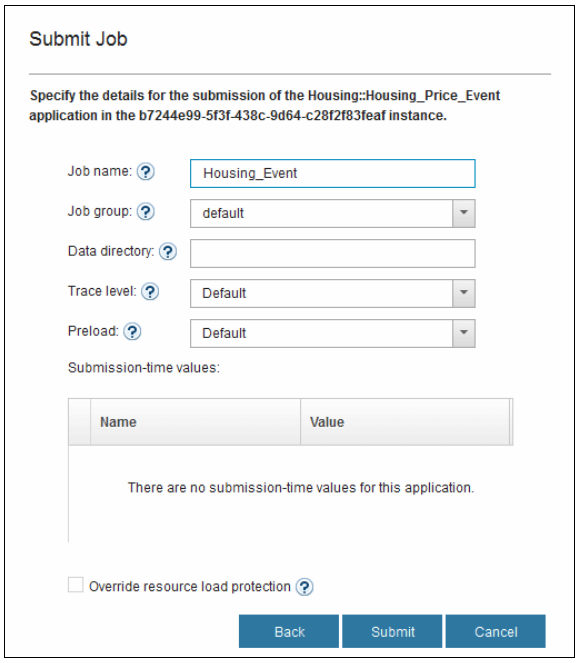
(●) Upload an application bundle file from the local file system.

Housing.Housing_Price_Event.sab

( ) Specify the absolute path for the application bundle file on the following resource: streams-console-c6d1.ng.bluemix.net.

Figure 9-44 Submitting a Streams Application job

After the Streams Application has been selected, select **Next**. This opens the configuration window for the Streams job. Here the name of the job is specified as **Housing_Event**. All other parameters are left as their default values. The job configuration window is shown in Figure 9-45.


Submit Job

Specify the details for the submission of the Housing::Housing_Price_Event application in the b7244e99-5f3f-438c-9d64-c28f2f83feaf instance.

Job name:	Housing_Event
Job group:	default
Data directory:	
Trace level:	Default
Preload:	Default

Submission-time values:

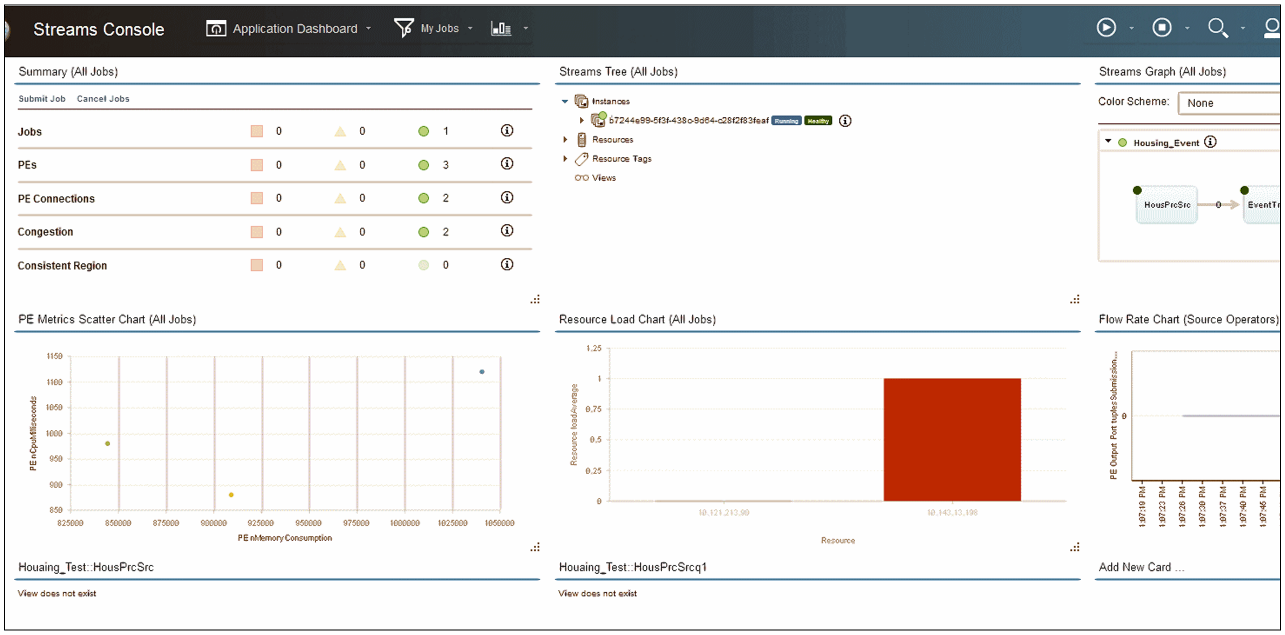
Name	Value

There are no submission-time values for this application.

☐ Override resource load protection

Figure 9-45 Job configuration window

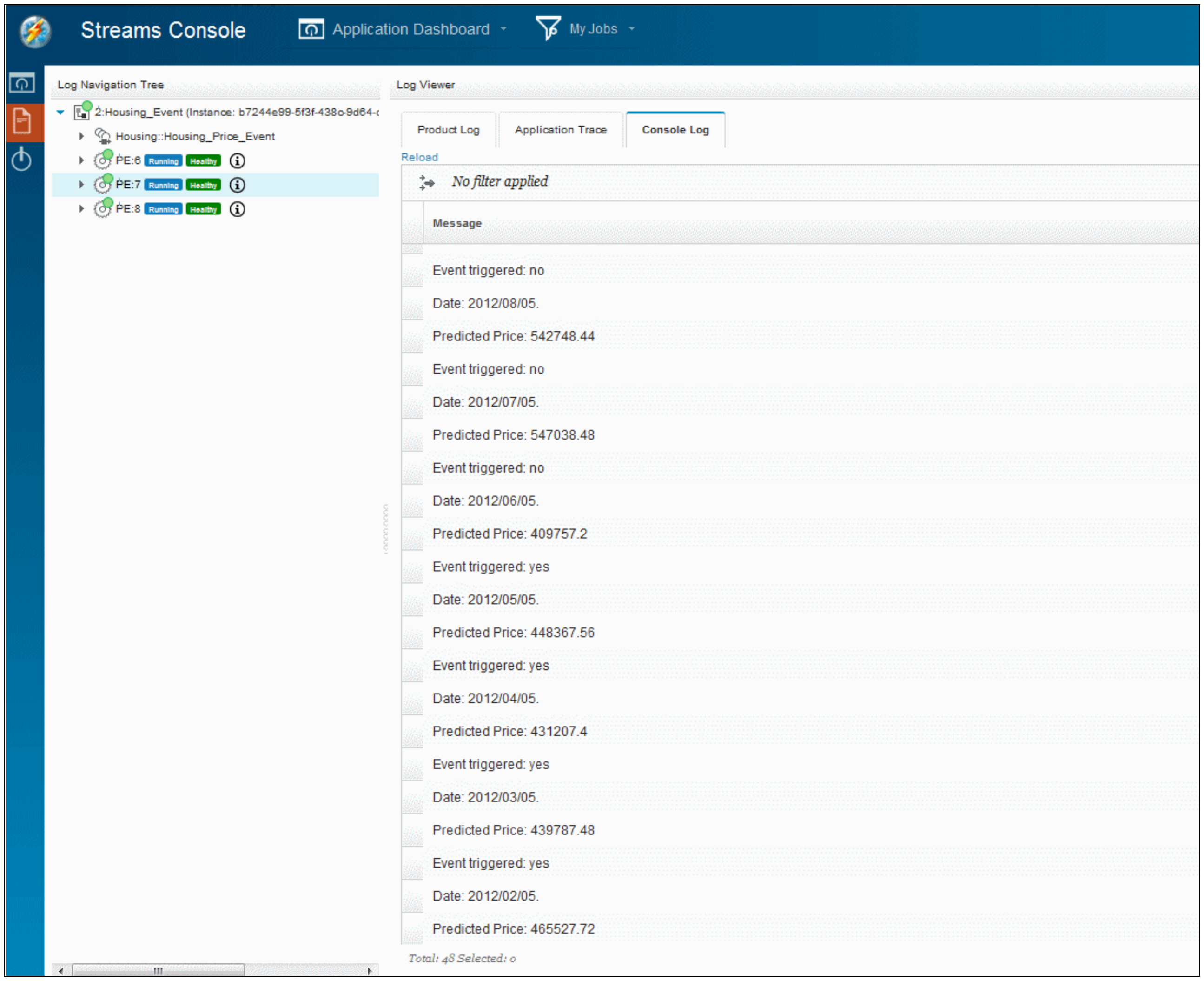
Next, the job is submitted to the streaming analytics service. This updates the Streams Console and opens a window showing the application flow, as shown in the upper right card in Figure 9-46. The green box at the upper left of each operator demonstrates that each one is functioning correctly.



Jobs	🟥	0	▲	0	●	1	ⓘ
PEs	🟥	0	▲	0	●	3	ⓘ
PE Connections	🟥	0	▲	0	●	2	ⓘ
Congestion	🟥	0	▲	0	●	2	ⓘ
Consistent Region	🟥	0	▲	0	●	0	ⓘ

Figure 9-46   Streams Application Console after submitting a job

To view the output from this job, the Streams Console needs to be used. This can be accessed by selecting the log viewer on the left of the window (the icon looks like a page). This opens a log navigation window. Expand the current job and select the processing element (PE) associated with the EventTrigger operator; in this case, it is the second PE. Then, select the **Console Log** tab, and select **load console message**. This opens the console logs for the application job as demonstrated in Figure 9-47 on page 291.



```
Event triggered: no
Date: 2012/08/05.
Predicted Price: 542748.44
Event triggered: no
Date: 2012/07/05.
Predicted Price: 547038.48
Event triggered: no
Date: 2012/06/05.
Predicted Price: 409757.2
Event triggered: yes
Date: 2012/05/05.
Predicted Price: 448367.56
Event triggered: yes
Date: 2012/04/05.
Predicted Price: 431207.4
Event triggered: yes
Date: 2012/03/05.
Predicted Price: 439787.48
Event triggered: yes
Date: 2012/02/05.
Predicted Price: 465527.72
```

Figure 9-47 Streams Console log for the event trigger processing element

The results show that when the predicted price is less than $450,000, an event is triggered. This event can then be passed onto another operator, which can be used to let a user know that an event occurred. Ensure that the service is stopped when all jobs have finished.

9.9 Conclusion

In this chapter, the process of creating a model based on data, and then using the model to detect real-time events based on user criteria using Bluemix services was demonstrated. In particular, the dashDB service was used to import data, and develop a model from this data. And the streaming analytics service was used to upload a local Streams Application to the cloud and detect events based on simulated real-time data.

The example described in this chapter is stylized, as real-time event triggers for housing prices seem a bit contradictory due to the slow change in house prices, based on the predictors used in the model. This is also clear from the methodology used to model house prices where yearly data is used. This clearly should not be the basis for a real-time model because the time scales are disparate.

However, the focus of this chapter is on the process of going from raw data extracted from a source to developing a model, and then using this model for real-time event triggers. The process shown here can be used for any data source and with any type of event trigger, with only minor changes required: First, to the methodology used to create the model (any model developed should be thoroughly considered and take into consideration the best methodology to use based on the structure of the data), second, to the local Streams Application (in particular the data source, which should be a real-time source that is accessible by the Bluemix service).

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

The following IBM Redbooks publication provides additional information about the topic in this document. Note that this publication might be available in softcopy only.

Hybrid Cloud Data and API Integration: Integrate Your Enterprise and Cloud with Bluemix Integration Services, SG24-8277

You can search for, view, download or order this document and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- IBM Maximo Asset Management product page:

  http://www.ibm.com/software/products/en/maximoassetmanagement
- IBM Maximo Asset Management V7.6 product documentation:

  http://www.ibm.com/support/knowledgecenter/SSLKT6_7.6.0/com.ibm.mam.doc/welcome.html
- Twilio website:

  https://www.twilio.com
- Swagger framework:

  http://swagger.io
- Tutorials for TI SensorTag:

  https://developer.ibm.com/recipes
- IBM Internet of Things Foundation Quickstart service:

  https://quickstart.internetofthings.ibmcloud.com
- Cloud Foundry variables:

  https://docs.cloudfoundry.org/devguide/deploy-apps/environment-variable.html
- Historical exchange rates for different countries (Data provided by Reserve Bank of Australia):

  http://www.rba.gov.au/statistics/historical-data.html#exchange-rates
- Historical cash rate (Data provided by Reserve Bank of Australia):

  http://www.rba.gov.au/statistics/tables/index.html#interest-rates

- Average yearly house price per suburb in Victoria (Data provided by Victorian State Government of Australia):

  http://www.dtpli.vic.gov.au/property-and-land-titles/property-information/property-prices

- IBM InfoSphere Streams Quick Start Guide:

  http://www.ibm.com/analytics/us/en/technology/stream-computing

- MongoDB download website:

  https://www.mongodb.org/downloads#production

- MongoDB installation manual:

  https://docs.mongodb.org/master/installation

- Configuring the Secure Gateway service on Bluemix:

  https://www.ng.bluemix.net/docs/services/SecureGateway/sg_022.html#sg_009

- Client wrapper using an AMQP messaging toolkit like Qpid Proton:

  http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Redbooks

Hybrid Cloud Event Integration: Integrate Your Enterprise and Cloud with Bluemix Integration Services

SG24-8281-00

ISBN 0738441511

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

SG24-8281-00

ISBN 0738441511

Printed in U.S.A.