

# An Architectural and Practical Guide to IBM Hybrid Integration Platform

Carsten Börnert

Amar Shah

Kim Clark

Johan Thole

Shahir Daya

Matthieu Debeaux

Gerd Diederichs

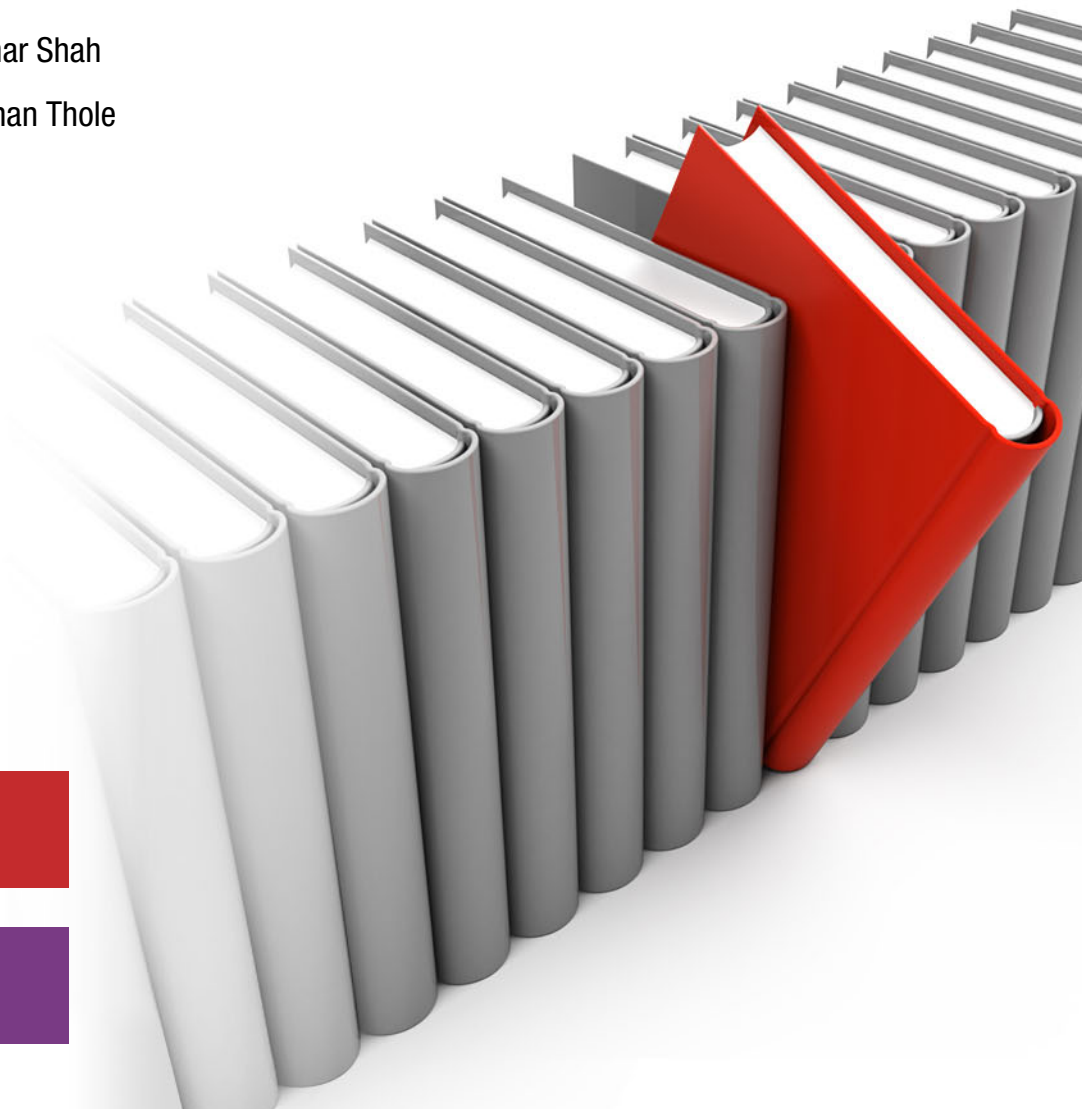
Vasfi Gucer

Shamim Hossain

Gary Kean

Carlo Marcoli

Shohei Matsumoto



 **Cloud**

**WebSphere**





International Technical Support Organization

**A Practical Guide for IBM Hybrid Integration Platform**

December 2016

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

**First Edition (December 2016)**

This edition applies to the following products:

- ▶ IBM API Connect Version 5.0
- ▶ IBM MQ Version 9.0
- ▶ IBM Integration Bus Version 10.0.0.6

© Copyright International Business Machines Corporation 2016. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Figures</b> .....	vii
<b>Tables</b> .....	xiii
<b>Examples</b> .....	xv
<b>Notices</b> .....	xvii
Trademarks .....	xviii
<b>Preface</b> .....	xix
Authors .....	xix
Now you can become a published author, too! .....	xxii
Comments welcome. ....	xxiii
Stay connected to IBM Redbooks .....	xxiii
<b>Part 1. Concepts and architecture</b> .....	1
<b>Chapter 1. Introduction to hybrid integration</b> .....	3
1.1 What is hybrid integration .....	4
1.1.1 Hybrid user communities .....	4
1.1.2 Hybrid connectivity .....	5
1.1.3 Hybrid integration styles .....	6
1.1.4 Hybrid deployment .....	6
1.2 Hybrid integration reference architecture .....	7
1.2.1 Hybrid communities .....	8
1.2.2 Hybrid connectivity .....	9
1.2.3 Hybrid styles .....	9
1.2.4 Hybrid deployment .....	9
1.3 How is IBM addressing hybrid requirements in its integration portfolio? .....	9
<b>Chapter 2. Hybrid integration use cases</b> .....	11
2.1 Introduction to hybrid Integration use cases .....	12
2.2 Use case A: Joining the API economy .....	12
2.2.1 Unlocking systems of record .....	13
2.2.2 Securely surfacing APIs .....	14
2.2.3 Extending the use case: Creating more valuable “interaction APIs” .....	15
2.2.4 A hybrid landscape for API components .....	15
2.3 Use case B: Improving productivity .....	17
2.3.1 Extending the use case: Cloud-to-ground (and ground-to-cloud) .....	20
2.4 Use case C: Refactoring for innovation .....	21
2.4.1 Modernizing applications for composability .....	21
2.4.2 Synchronizing systems of record .....	22
2.4.3 Embracing external sources .....	24
2.4.4 Innovating new solutions .....	24
<b>Chapter 3. IBM Hybrid Cloud and Integration Portfolio</b> .....	25
3.1 Introduction .....	26
3.2 IBM Application Integration Suite .....	26
3.2.1 IBM Integration Bus .....	28
3.2.2 IBM API Connect .....	32

3.2.3 IBM App Connect .....	34
3.3 IBM Messaging Portfolio .....	40
3.3.1 IBM MQ .....	40
3.3.2 IBM Message Hub .....	43
3.3.3 IBM Watson Internet of Things Platform .....	46
3.4 IBM DataPower .....	49
<b>Part 2. Hybrid integration scenarios .....</b>	<b>53</b>
<b>Chapter 4. Introduction to the scenarios .....</b>	<b>55</b>
4.1 Introducing CompanyA .....	56
4.1.1 The business .....	56
4.1.2 Hybrid integration scenarios .....	57
4.2 Architecture overview .....	60
4.2.1 Components .....	61
4.3 Re-creating the scenarios .....	62
4.3.1 Prerequisites .....	62
4.3.2 The GitHub repositories .....	63
4.3.3 Environment configuration for the scenarios .....	64
<b>Chapter 5. Exposing APIs externally .....</b>	<b>69</b>
5.1 Solution outline .....	70
5.1.1 Overview of the CompanyA hybrid integration landscape .....	70
5.1.2 Expose a System API for product information from a catalog .....	71
5.1.3 Create an Interaction API for order information .....	71
5.2 Implementation .....	71
5.2.1 Create an API Connect instance on Bluemix .....	72
5.2.2 Expose a SOAP Web Service as an API .....	77
5.2.3 Create an Interaction API for order information .....	90
5.2.4 Create a Production catalog with an IBM Developer Portal Site .....	110
5.3 Resources .....	117
5.3.1 Sign up for an IBM Bluemix account .....	117
5.3.2 Setting up a Salesforce developer edition account .....	117
5.3.3 Importing the Accounts data into the Salesforce developer organization .....	117
<b>Chapter 6. Automation for business users .....</b>	<b>119</b>
6.1 Solution outline .....	120
6.1.1 Functional overview .....	120
6.1.2 Technical overview .....	122
6.2 Implementation .....	123
6.2.1 Preparation .....	123
6.2.2 Creating the AppConnect flow to keep the campaign overview spreadsheet up to date .....	124
6.2.3 Exposing the order update events as triggers into AppConnect .....	128
6.2.4 Creating an AppConnect flow to process campaign-related orders .....	142
6.2.5 Creating an AppConnect flow to process canceled orders .....	147
6.3 Resources .....	150
6.3.1 AppConnect pattern for IBM Integration Bus .....	150
6.3.2 Installing the trigger pattern in IBM Integration Bus Toolkit .....	151
<b>Chapter 7. Kick-start digital teams .....</b>	<b>153</b>
7.1 Solution overview .....	154
7.1.1 Variant A .....	155
7.1.2 Variant B .....	155

7.1.3 Variant C . . . . .	156
7.1.4 Implementation considerations . . . . .	156
7.2 Technical implementation of variant B . . . . .	157
7.2.1 IBM Integration Bus: Exposing enterprise order events . . . . .	157
7.2.2 Message Hub . . . . .	170
7.2.3 MongoDB . . . . .	172
7.2.4 API Connect . . . . .	177
7.2.5 Dashboard app . . . . .	185
7.3 Technical implementation of variant C . . . . .	190
7.3.1 Secure Gateway . . . . .	191
7.3.2 Docker Compose (IBM Integration Bus, IBM MQ, Secure Gateway) . . . . .	197
7.3.3 Message Hub . . . . .	199
7.3.4 Message Connect . . . . .	200
7.3.5 MongoDB . . . . .	204
7.3.6 API Connect . . . . .	204
7.3.7 Microservice . . . . .	204
7.3.8 Dashboard app . . . . .	204
7.4 Technical implementation of variant A . . . . .	205
7.4.1 IBM Integration Bus: Exposing enterprise order events . . . . .	205
7.4.2 MongoDB . . . . .	207
7.4.3 API Connect . . . . .	207
7.4.4 Microservice . . . . .	207
7.4.5 Dashboard app . . . . .	207
<b>Related publications . . . . .</b>	<b>209</b>
IBM Redbooks . . . . .	209
Online resources . . . . .	209
Help from IBM . . . . .	210





# Figures

1-1	Different people in more diverse roles are now involved in performing integration. . . . .	4
1-2	Surface area over which integration applications have expanded dramatically . . . . .	5
1-3	Different design styles of application integration, API exposure, and data integration . .	6
1-4	Options for how and where to deploy . . . . .	7
1-5	Hybrid integration reference architecture . . . . .	8
2-1	Most common use cases found in enterprises that use hybrid integration . . . . .	12
2-2	Use case A: Joining the API economy . . . . .	13
2-3	Two layered models . . . . .	15
2-4	Bi-modal IT . . . . .	16
2-5	Use case B: Improving productivity. . . . .	17
2-6	Enabling the business users to perform some of their own integration automation . . .	18
2-7	Cloud-to-cloud integration. . . . .	19
2-8	Ground-to-cloud integration . . . . .	20
2-9	Microservices applications . . . . .	21
2-10	New era of application architectures. . . . .	23
2-11	Integrating external sources . . . . .	24
3-1	Application Integration Suite offerings . . . . .	26
3-2	Hybrid integration personas . . . . .	27
3-3	API Connect capabilities overview . . . . .	32
3-4	IBM API Connect logical components. . . . .	33
3-5	Select applications to integrate. . . . .	35
3-6	Connect application accounts. . . . .	36
3-7	Select/specify any data. . . . .	37
3-8	Run the flow . . . . .	38
3-9	Add a custom application . . . . .	39
3-10	Connect to your network by using the Secure Gateway . . . . .	40
3-11	Connectivity options with Message Hub . . . . .	43
3-12	Message Hub use cases. . . . .	45
3-13	Kafka architecture . . . . .	46
3-14	Watson Internet of Things service from Bluemix. . . . .	47
3-15	IBM Watson IoT architecture . . . . .	48
3-16	Internet of Things Quickstart mode . . . . .	49
4-1	Functional capabilities. . . . .	56
4-2	Long tail of integration candidates . . . . .	58
4-3	High-level architecture overview . . . . .	60
4-4	GitHub repositories for scenarios . . . . .	63
4-5	Sample content of scenario repository . . . . .	63
4-6	Sample docker-compose.yml file . . . . .	64
4-7	Required environment variables . . . . .	65
4-8	Starting the containers for scenario 1 . . . . .	65
4-9	Checking the state of containers with docker-compose . . . . .	66
5-1	Logical view of the CompanyA hybrid environment . . . . .	70
5-2	Bluemix categories . . . . .	72
5-3	Create an API Connect service instance . . . . .	72
5-4	Define an API Connect service. . . . .	73
5-5	API Connect service pricing plans . . . . .	73
5-6	Draft space in the API Manager interface . . . . .	74
5-7	Pinning the Navigation pane to the user interface . . . . .	74

5-8	Catalog settings page . . . . .	75
5-9	API Endpoint Base URL . . . . .	75
5-10	Developer portal settings . . . . .	76
5-11	Developer portal notification email . . . . .	76
5-12	Developer Portal main site . . . . .	77
5-13	Implementation components for exposing a SOAP Web service . . . . .	77
5-14	IBM Bluemix service categories . . . . .	78
5-15	Secure Gateway service creation . . . . .	79
5-16	Secure Gateway definition . . . . .	79
5-17	Secure Gateway ID and secret . . . . .	80
5-18	Add a Secure Gateway destination . . . . .	81
5-19	Setting the Secure Gateway Client variables . . . . .	82
5-20	Secure Gateway connection status . . . . .	82
5-21	Running docker container services . . . . .	82
5-22	Secure Gateway destination settings . . . . .	83
5-23	Secure Gateway connection details . . . . .	83
5-24	Retrieve the WSDL from the Cloud endpoint . . . . .	84
5-25	CompanyA API Connect instance . . . . .	84
5-26	API Manager Design view . . . . .	85
5-27	JSON Schema definition for the Products API . . . . .	85
5-28	Definitions for GET /product operation . . . . .	86
5-29	Import CatalogServiceV1 SOAP web service . . . . .	87
5-30	SOAP service invocation through an Invoke policy . . . . .	87
5-31	Edit inputs of the getProducts: input map policy . . . . .	87
5-32	Configure the input mapping policy . . . . .	88
5-33	Configure the output mapping policy . . . . .	89
5-34	Start the API Manager test tool . . . . .	89
5-35	Implementation components for exposing an API from IBM Integration Bus . . . . .	90
5-36	Imported projects . . . . .	93
5-37	REST operations . . . . .	94
5-38	Add parameters to operations . . . . .	94
5-39	Main message flow for Orders API . . . . .	94
5-40	Implemented operations details . . . . .	95
5-41	Implementing a subflow . . . . .	95
5-42	Aggregation flow . . . . .	96
5-43	Aggregation Fanout flow . . . . .	97
5-44	flow1_salesforce.msgflow . . . . .	97
5-45	Salesforce Request node property . . . . .	97
5-46	flow2_catalogservice.msgflow . . . . .	98
5-47	SOAP Request Node Properties . . . . .	98
5-48	Aggregate two response messages . . . . .	98
5-49	Monitoring event configuration . . . . .	99
5-50	Integration Node details in WebUI . . . . .	101
5-51	Deploy to integration Server . . . . .	101
5-52	Deploy BAR File . . . . .	102
5-53	Deployed resources . . . . .	102
5-54	Cloud host and port . . . . .	104
5-55	Push REST APIs to API Connect . . . . .	104
5-56	Configure connection to an API Connect cloud . . . . .	105
5-57	Successful connection . . . . .	105
5-58	Specify target details for pushing REST API . . . . .	106
5-59	Select the API to be pushed . . . . .	106
5-60	Override host and port for the APIs on IBM Integration Bus . . . . .	106

5-61	Success message after Pushing API	107
5-62	Go to the API Connect Dashboard	107
5-63	Publish a staged product in a catalog	108
5-64	Add a catalog to an API Connect instance	110
5-65	Add a Production catalog to API Connect	110
5-66	Portal Configuration window	111
5-67	Add a Developer organization to the catalog	111
5-68	Enter your Bluemix details	111
5-69	Bluemix Developer organization confirmation email	112
5-70	Confirm your Bluemix organization	112
5-71	Stage a product to a catalog	112
5-72	API Connect IBM Developer Portal site	113
5-73	Creating the application	113
5-74	API Products published in the Production catalog	114
5-75	Subscribe an application to a Plan	114
5-76	Select the Products API that was created earlier	115
5-77	Products API details in the Developer Portal	115
5-78	Show the GET /product operation	116
5-79	Use the Developer Portal test tool	116
6-1	Functional overview diagram	120
6-2	Technical overview	122
6-3	Simplified configuration	123
6-4	Salesforce trigger options	125
6-5	Select application accounts	125
6-6	Select Google spreadsheet and worksheet	126
6-7	Use of functions for the date fields	126
6-8	Finalize and name the AppConnect flow	127
6-9	Create a Campaign in Salesforce	127
6-10	Added row for Winter Sale Campaign to Google sheet	128
6-11	Publishing an order update event	128
6-12	Flow overview	129
6-13	IBM Integration Bus menu	130
6-14	Select endpoint type	130
6-15	IBM MQ Endpoint configuration	130
6-16	Synchronize IBM MQ Endpoint	131
6-17	Download agent configuration	131
6-18	Setting up an agent for Callable Flows	131
6-19	Download configuration	132
6-20	Scenario2 directory and content	132
6-21	Status of docker-compose environment	133
6-22	Agent connected	134
6-23	EVENTQM endpoint connected	134
6-24	Importing the Project Interchange file	135
6-25	Finalize Project Interchange import	135
6-26	Flow skeleton	136
6-27	WebhookOutput subflow	136
6-28	Wiring the Route node terminals	137
6-29	Subflow node properties for cancelOrder	137
6-30	New BAR file	138
6-31	Build and Save the BAR file	138
6-32	Add an integration	139
6-33	Upload BAR file	139
6-34	Switching off Basic Authentication	139

6-35	Details of the integration . . . . .	140
6-36	Start integration. . . . .	140
6-37	Host name for the application on IBM Integration Bus on Cloud . . . . .	141
6-38	Directory contents . . . . .	141
6-39	Cancel and Campaign messages received . . . . .	142
6-40	Output from the iib-listTriggers command. . . . .	142
6-41	Replacing the host name in the application definition yaml file. . . . .	143
6-42	Add an application to IBM App Connect . . . . .	143
6-43	Add Application window . . . . .	144
6-44	Orders application. . . . .	144
6-45	Select the campaignOrder event trigger . . . . .	145
6-46	Select accounts for the applications used in the flow . . . . .	145
6-47	Select the worksheet, and auto match fields . . . . .	146
6-48	Completed flow . . . . .	146
6-49	One event has been routed to the orderCampaign trigger . . . . .	147
6-50	Save + Continue button . . . . .	147
6-51	Cancel order trigger . . . . .	147
6-52	Create Case trigger. . . . .	148
6-53	Cancelled order follow-up flow . . . . .	149
6-54	Registered AppConnect flow . . . . .	149
6-55	Command line output . . . . .	149
6-56	New Case in Salesforce . . . . .	150
6-57	Integration topology that uses callable flows . . . . .	151
6-58	Import trigger pattern . . . . .	152
6-59	Application and Libraries for Trigger pattern. . . . .	152
7-1	Scenario 3 - Logical diagram - All variants . . . . .	154
7-2	Scenario 3: Technical diagram for all variants . . . . .	156
7-3	Scenario 3: Technical diagram for variant B. . . . .	157
7-4	Flow overview . . . . .	158
7-5	Docker compose environment . . . . .	159
7-6	New application. . . . .	160
7-7	Import from File System . . . . .	160
7-8	Selecting resources to be imported . . . . .	161
7-9	Creating a new Message Flow . . . . .	162
7-10	Select inputs and outputs for the mapping . . . . .	163
7-11	Adding a user-defined field . . . . .	164
7-12	JSON output object. . . . .	165
7-13	Creating a Move mapping action . . . . .	166
7-14	Custom ESQL mapping action . . . . .	166
7-15	Configuring the Custom ESQL properties. . . . .	167
7-16	Current date function . . . . .	167
7-17	Complete mapping . . . . .	168
7-18	Port number . . . . .	169
7-19	Deployed IBM Integration Bus application . . . . .	169
7-20	Output from test script. . . . .	170
7-21	Searching Message Hub in Bluemix catalog . . . . .	170
7-22	Configure the Message Hub service. . . . .	171
7-23	Create the Message Hub service . . . . .	171
7-24	Add a topic . . . . .	171
7-25	Configure the Message Hub topic. . . . .	172
7-26	Message Hub topic created . . . . .	172
7-27	Compose website . . . . .	173
7-28	Compose account creation form . . . . .	173

7-29	Compose account creation form completed	174
7-30	Compose address form	175
7-31	Compose payment form	175
7-32	Compose MongoDB creation form	176
7-33	Compose MongoDB deployment	177
7-34	Microservice architecture	177
7-35	IBM API Connect order model creation	179
7-36	IBM API Connect order event model creation	179
7-37	IBM API Connect Message Hub creation	180
7-38	IBM API Connect MongoDB data source creation	180
7-39	Microservice directory structure	181
7-40	Microservice message-hub-proxy.js	182
7-41	Microservice message-hub-proxy function	182
7-42	Microservice message-consumer.js	183
7-43	Microservice message-consumer method	183
7-44	IBM API Connect Designer	184
7-45	IBM API Connect catalog	185
7-46	Microservice management in Bluemix	185
7-47	Application overview diagram	186
7-48	Ionic application file structure	187
7-49	Order tracker application settings	188
7-50	Order tracker application settings filled	189
7-51	List of orders	189
7-52	Order details	190
7-53	Scenario 3 technical diagram, variant C	191
7-54	Search the Secure Gateway service in Bluemix catalog	191
7-55	Secure Gateway service details page	192
7-56	Secure Gateway notification if the service exists	192
7-57	Secure Gateway service creation	192
7-58	Secure Gateway configuration	193
7-59	Create a gateway	193
7-60	Secure Gateway Dashboard	194
7-61	Security Token and Gateway ID	194
7-62	Add a destination to Secure Gateway	195
7-63	Add destination configuration	195
7-64	Destination host and port setting	196
7-65	Protocol setting	196
7-66	Authentication setting	196
7-67	IP table rules setting	196
7-68	Name the destination	196
7-69	Created destination	197
7-70	Flow overview for this variant	197
7-71	Docker port mapping	198
7-72	Test subscription	199
7-73	Test event	199
7-74	Bluemix Experimental Services	200
7-75	Message Connect service	200
7-76	Message Connect service instance creation	201
7-77	Message Connect dashboard	201
7-78	Stream name	201
7-79	Select MQ Light connector	202
7-80	Secure Gateway configuration import	202
7-81	Additional information	202

7-82	Created stream . . . . .	203
7-83	Streams list . . . . .	203
7-84	Topic on Message Hub . . . . .	203
7-85	Scenario 3 - Technical diagram - Variant A . . . . .	205
7-86	Flow overview . . . . .	205
7-87	Installing the mongodb connector . . . . .	206
7-88	Output of testme.sh script . . . . .	207

# Tables

5-1	Advanced setup values . . . . .	81
5-2	Properties table . . . . .	85
5-3	Properties table . . . . .	88
5-4	Properties table . . . . .	88
5-5	ORDER table . . . . .	96
5-6	PRODUCTLIST table . . . . .	96
5-7	Parameters table . . . . .	100
5-8	Configuration table . . . . .	103
5-9	API Connect server host . . . . .	105
6-1	Google spreadsheet layout . . . . .	124
6-2	Campaign values . . . . .	127
6-3	Properties of the new node . . . . .	136
6-4	Application name and Description . . . . .	144
6-5	Mapping and values on the set up Salesforce action window . . . . .	148
7-1	Attributes . . . . .	164
7-2	Target attributes . . . . .	165
7-3	Node properties . . . . .	168
7-4	Property values . . . . .	169
7-5	Properties table . . . . .	198
7-6	Properties table . . . . .	207





# Examples

5-1 Temporary text file . . . . .	80
5-2 Accessing parameter values in various languages . . . . .	92
5-3 The docker-compose ps command . . . . .	100
5-4 Operation successful message . . . . .	103
5-5 Output of the getOrder operation . . . . .	103
5-6 Test date (orderdata01.txt) . . . . .	108
5-7 Create order . . . . .	108
5-8 Get order . . . . .	109
5-9 Cancel order . . . . .	109
5-10 Get order again . . . . .	109
7-1 Result of display channel status command . . . . .	204



# Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US*

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

## COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

# Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

Bluemix®	IBM UrbanCode™	Redbooks (logo)  ®
Cast Iron®	IBM Watson™	UrbanCode™
DataPower®	IBM Watson IoT™	Watson IoT™
DB2®	InfoSphere®	WebSphere®
IBM®	MQSeries®	z/OS®
IBM API Connect™	Redbooks®	

The following terms are trademarks of other companies:

SoftLayer, and SoftLayer device are trademarks or registered trademarks of SoftLayer, Inc., an IBM Company.

LoopBack, and the StrongLoop logo are trademarks of StrongLoop, Inc., an IBM Company.

ITIL is a Registered Trade Mark of AXELOS Limited.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other company, product, or service names may be trademarks or service marks of others.

# Preface

In order to remain competitive in today's world, companies need to be able to integrate internally and externally by connecting sensors, customers and partners with the information in their systems of record. In short, they need to *integrate with everything*.

This IBM® Redbooks® publication describes how IBM Application Integration Suite and IBM Messaging portfolio can be used to satisfy the needs of core hybrid integration use cases, accelerating companies in their digital transformation journey.

All concepts are explained within the context of these use cases:

- ▶ Joining the API economy
- ▶ Improving productivity
- ▶ Refactoring for innovation

The target audience for this book is cloud and integration architects and specialists who are implementing hybrid integration solutions.

## Authors

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Austin Center.



**Carsten Börnert** is an experienced IT Specialist working in Cloud Adoption and Deployment Services, which is part of IBM Cloud. He has over 12 years of experience in Application Integration, specifically using IBM Integration and Messaging products like IBM Integration Bus, IBM MQ, and more recently IBM API Connect™. His area of expertise in IBM Cloud is Hybrid Integration. Carsten has been living and working for IBM in a number of countries around the world. He has worked with customers from a wide range of industries in roles ranging from support over development to solution architecture.



**Kim Clark** has spent the last couple of decades working in the field helping customers implement the integration aspects of solutions, tackling concepts such as SOA, BPM, and APIs and microservices. He has a broad and deep knowledge of the architecture and design issues that are involved and of the related IBM products. As a result, he writes and presents regularly on design and architecture best practices in this space. In his current role, Kim works as a strategist on the IBM integration portfolio shaping the future direction of the products, but still spends much of his time assisting customers improving their integration architecture. Recent articles include “The evolving hybrid integration reference architecture” <https://ibm.biz/HybridIntRefArch> and “Microservices, SOA, and APIs: Friends or enemies?” <https://ibm.biz/MicroservicesVsSoa>. You can find him on LinkedIn: <https://www.linkedin.com/in/kimjulianclark>.



**Shahir Daya** is an IBM Executive Architect in the GBS Global Cloud Center of Competence. He is an IBM Senior Certified Architect and an Open Group Distinguished Chief/Lead IT Architect. Shahir has over 20 years at IBM with the last 15 focused on application architecture assignments. He has experience with complex high volume transactional web applications and systems integration. Shahir has led teams of practitioners to help IBM and its customers with application architecture for several years. His industry experience includes retail, banking, financial services, public sector, and telecommunications. Shahir is currently focused on Cloud application development services and in particular platform as a service (PaaS) such as IBM Bluemix®, containerization technology such as Docker, design, and development of Systems of Engagement (SOE), and Cloud-based DevOps including IBM Bluemix DevOps Services.



**Matthieu Debeaux** is an IBM Certified IT Specialist with IBM Cloud in France. He has 5 years of experience working in cloud computing, virtualization, and IT process management both on sales and delivery projects. Matthieu is currently focused on Cloud technologies such as OpenStack and Docker, and DevOps solutions, in particular IBM UrbanCode™ Deploy and Release. He holds a Master’s degree in Computer Science from Grenoble Institute of Technology.



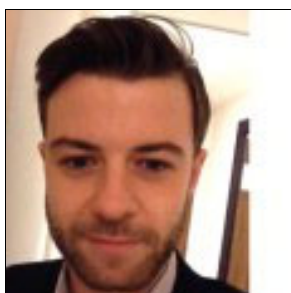
**Gerd Diederichs** is Technical Consultant in South Africa. He has been in IT for 47 years, about 28 of which he spent developing a wide variety of application systems and some early middleware. He has been with IBM in South Africa for 17 years and currently works as a Technical Consultant for the worldwide team for Business Partner Enablement, presenting, teaching, and proposing solution architectures to partners and their customers.



**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 three 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 fiber 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.



**Gary Kean** is an API Connect Solutions architect working in the worldwide IBM professional services team. He is experienced in delivering SOA and Microservices architectures for customers, particularly in the banking sector. Gary has a keen interest in Node.js and other JavaScript-based frameworks such as AngularJS and Ionic.



**Carlo Marcoli** is an Enterprise Architect specialized in API and Digital Channels. He has a long experience helping customers adopt leading-edge technology solutions to transform their business. He currently works in the worldwide API Connect pre-sales team. His main area of interests include the API economy, Node.js, and micro services.



**Shohei Matsumoto** is an IT Specialist with IBM MQ and IBM Integration Bus in Japan. He has 15 years of experience designing and implementing in enterprise systems integration on both distributed and mainframe platforms, and has been in technical support for messaging products. Shohei is currently focused on digital messaging technologies such as MQTT, AMQP, and Apache Kafka.



**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 a designated Lab Advocate for key clients and provides 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.



**Johan Thole** is a certified IT Specialist working in the European Tech Practice for Cloud Adoption and Deployment Services, which is part of IBM Cloud. He has over 15 years of experience in Security, and Application Integration. His area of expertise in IBM Cloud is Hybrid Integration. Johan's technical experience includes products like IBM DataPower® Gateways, IBM API Connect, and IBM Cast Iron®. Johan holds a Bachelor in Software Engineering from the Saxion University of Applied Sciences (formerly HIO Enschede).

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

Aarti D Borkar  
Tony Curcio  
Jack Carnes Jr  
IBM US

Rob Nicholson  
Andrew Schofield  
Mickael Maison  
Andrew Coleman  
Lee Gavin  
IBM UK

Francois van der Merwe  
IBM South Africa

## 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](http://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](http://ibm.com/redbooks)

- ▶ Send your comments in an email to:

[redbooks@us.ibm.com](mailto: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:

<http://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

## Concepts and architecture

This part introduces hybrid integration concepts and describes the most commonly used use cases for hybrid integration. It also introduces several IBM hybrid integration products that are used later in Part 2, “Hybrid integration scenarios” on page 53.

This part includes the following chapters:

- ▶ Introduction to hybrid integration
- ▶ Hybrid integration use cases
- ▶ IBM Hybrid Cloud and Integration Portfolio





# Introduction to hybrid integration

The computing world is now fundamentally hybrid in nature. Devices, systems, and people are spread across the globe. Achieving integration across this ever changing surface area at the pace of modern digital initiatives while still being able to manage the increasingly complex landscape is a significant challenge.

This book explores how IBM Application Integration Suite and the IBM Messaging portfolio can be used to satisfy the needs of core hybrid integration use cases, accelerating companies in their digital transformation journey.

This chapter explores what is meant by the term *hybrid integration* and how IBM is aligning its portfolio to meet these needs. It has the following sections:

- ▶ What is hybrid integration
- ▶ Hybrid integration reference architecture
- ▶ How is IBM addressing hybrid requirements in its integration portfolio?

## 1.1 What is hybrid integration

The face of integration is changing, and that change is being driven by the needs of the business. One such example is the shift to cloud, which is driven by cost optimization to improve the bottom line or recognizing the opportunity to build more agile business solutions. Although cloud solutions have many of the same integration needs as their predecessors on-premises, including connectivity, transformation, and cleansing, they also introduce a new set of requirements.

If businesses are to be successful in the face of these shifts, the enterprise must be equipped to adapt to these changing conditions. IBM has defined this agenda as “hybrid integration.” We strongly believe that hybrid is the future of integration.

In this context, hybrid has four primary dimensions:

- ▶ Hybrid user communities: Integration is needed by both IT and line of business (LOB) users who are adopting integration tools to automate application interactions.
- ▶ Hybrid connectivity: Integration reaches across secure connections to access data in the cloud or access data from the cloud.
- ▶ Hybrid integration styles: Combining application, API, and data integration to apply the optimal technology at the best time for any cloud solution.
- ▶ Hybrid deployment: Cloud based solutions rely on components being available on-premises and in cloud data centers.

This section looks at each of these dimensions in more detail.

### 1.1.1 Hybrid user communities

Integration is no longer the preserve of a central dedicated team of integration specialists. Different people in more diverse roles are now involved in performing integration, and there is a significant shift towards the lines of business solving their own integration challenges (Figure 1-1).

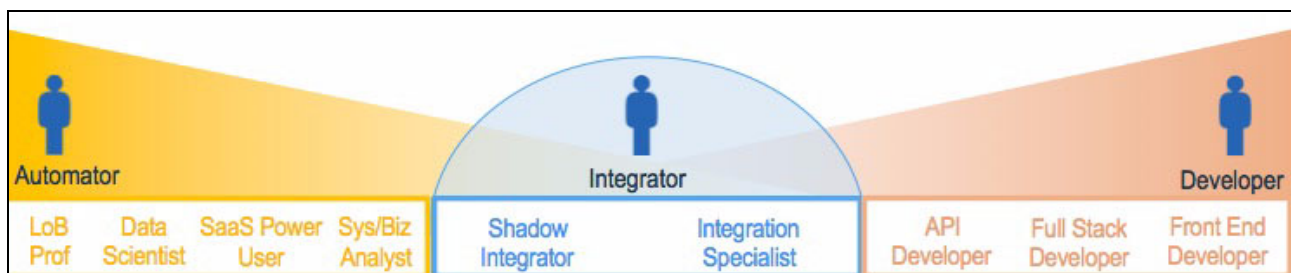


Figure 1-1 Different people in more diverse roles are now involved in performing integration

To describe all these roles in detail is beyond the scope of this book, but in brief Figure 1-1 shows three primary roles:

- ▶ Integration specialist: The traditional role that usually involves a dedicated team whose full-time job is integration. The members of this team use deep specialist tools, and attain a high degree of knowledge of how to interact with the unique systems that are owned by the enterprise.
- ▶ Developers: These members have deep technical skills in user interface and application runtime technologies. Their job is to create and enhance new business applications. They understand the areas of the business that are related to projects they work on, and the

applications they create might require access to a number of systems of record. They might understand how to connect to local systems such as application databases, but would prefer not to spend time writing code to integrate with complex systems because this process takes time away from implementing business requirements in their applications.

- ▶ Automators: Business users who want to focus on directly building the business. They are typically users of a number of systems of record. They might not be from a technical background. The fact that systems need to be integrated directly affects to the amount of manual work they need to do across any two or more systems. At the extreme end, these users are often referred to as Citizen Integrators.

This book covers how the use cases often involve more than one role and discuss how the tools should enable seamless collaboration between these different user roles.

### 1.1.2 Hybrid connectivity

As shown in Figure 1-2, the *surface area* over which applications are integrated has expanded dramatically. In addition to the vast array of traditional on-premises systems, there is an ever-growing ecosystem of cloud-based systems available over modern APIs, and a host of other diverse partner connectivity patterns.

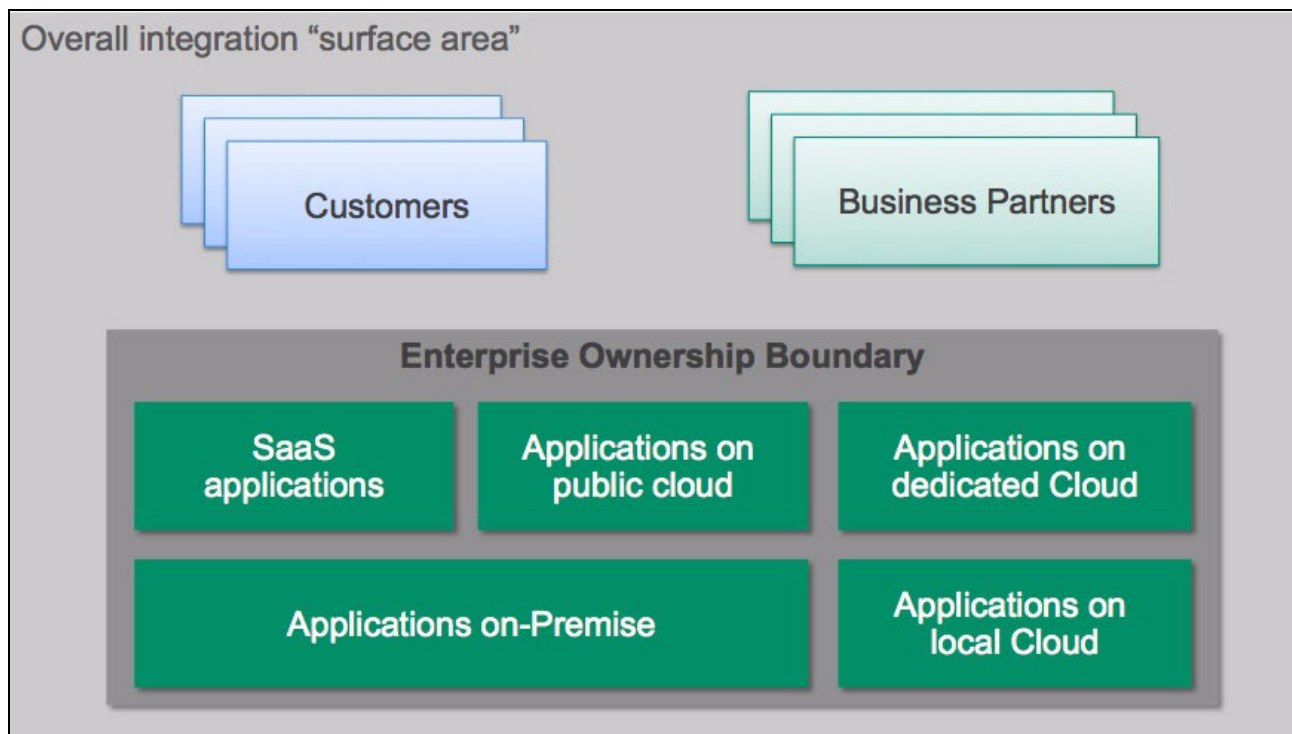


Figure 1-2 Surface area over which integration applications have expanded dramatically

It is also important to recognize how an enterprise must now recognize a broader "ownership boundary" that extends far beyond the physical walls of the company and its data centers. This boundary includes infrastructure, platform, and software "as a service" that contains the core data and functions critical to the day to day running of the business.

### 1.1.3 Hybrid integration styles

No single integration style is sufficient to resolve the needs of a complex hybrid solution. As shown in Figure 1-3, a hybrid integration platform needs to be able to combine the different design styles of application integration, API exposure, and data integration as required.

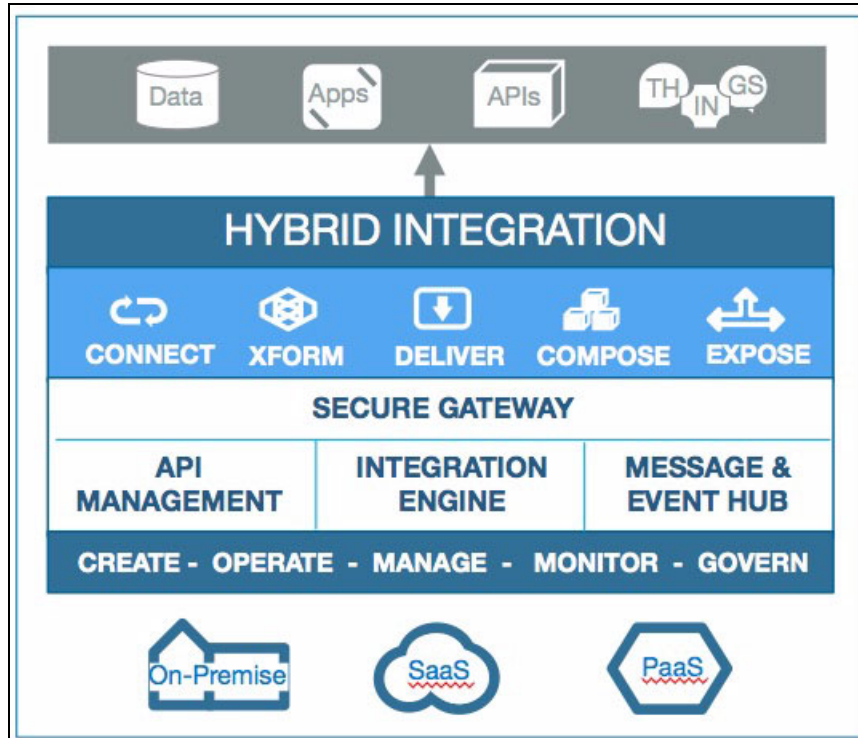


Figure 1-3 Different design styles of application integration, API exposure, and data integration

A hybrid integration platform must be able to combine the different design styles of application integration, API exposure, and data integration as required. It should be possible to apply the optimal patterns and runtimes for any given hybrid solution.

### 1.1.4 Hybrid deployment

The extension of the enterprise ownership boundary not only affects the breadth of things you need to connect to, it also has an effect on where you need to deploy your integration software components.



As shown in Figure 1-4, you need options when it comes to how and where you deploy.

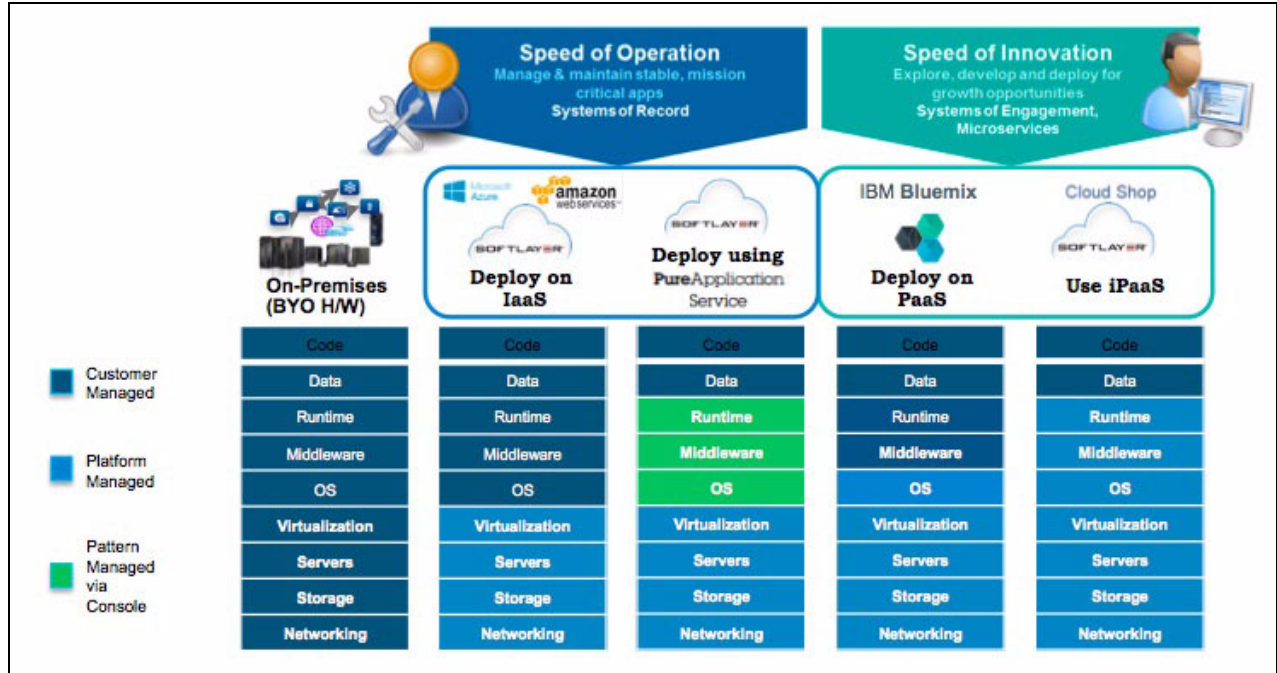


Figure 1-4 Options for how and where to deploy

You need to have the flexibility to deploy on bare metal, in virtual machines, and in containers, and deploy to a good breadth of different vendor infrastructure as a service. You also need to offer different levels of managed services from self-installed to fully managed platform as a service, or even software as a service.

**Note:** Although SoftLayer® is shown as a separate brand in Figure 1-4, IBM made an announcement that says that SoftLayer's products and tools will be merged under the Bluemix brand. For more information, see:

<https://www.ibm.com/blogs/bluemix/2016/10/bluemix-softlayer-unified-cloud-platform/>

With this integration, you will be able to manage all your Bluemix and SoftLayer assets from a single console, and be billed in a single invoice.

## 1.2 Hybrid integration reference architecture

IBM has proposed a *hybrid integration reference architecture*, catering to common patterns seen in enterprises to tackle these issues. It is described in full in a detailed article here:

<http://ibm.biz/HybridIntRefArch>

The core concepts are summarized in the following online presentation video:

<http://ibm.biz/HybridIntRefArchYouTube>

This section covers some of the most relevant core concepts as shown in Figure 1-5.

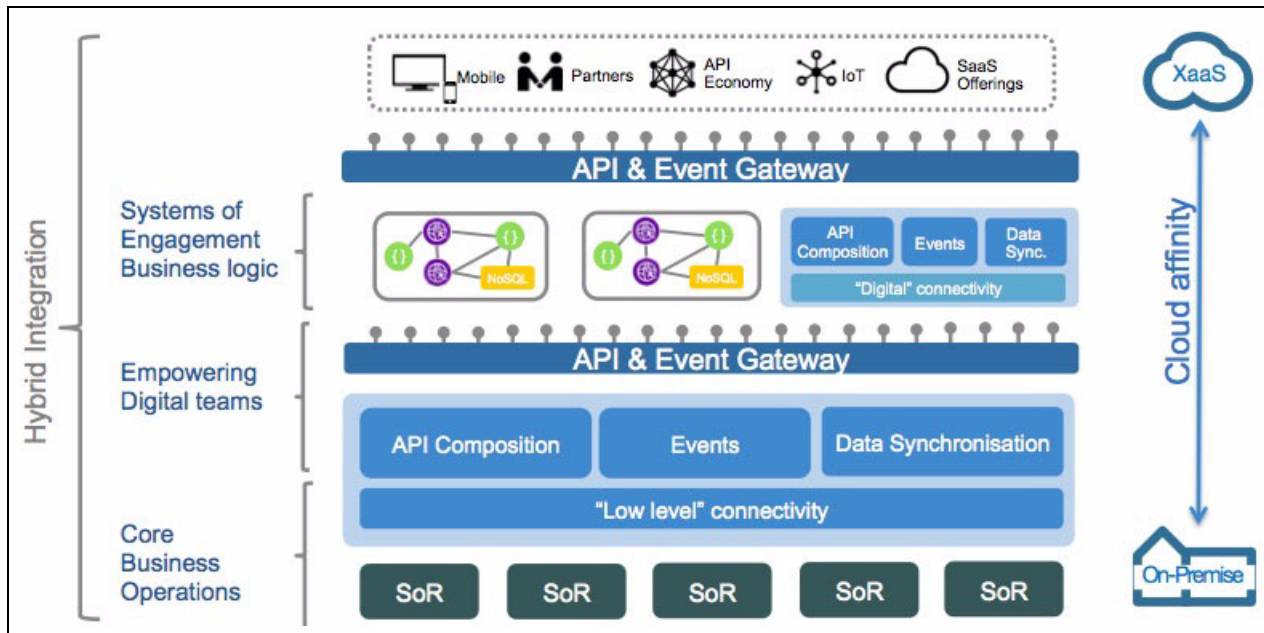


Figure 1-5 Hybrid integration reference architecture

The goal is to enable teams to become self-sufficient in their integration needs to the highest degree possible, while the IT department still maintains control and the ability to manage and refine the more business critical integrations.

The reference architecture addresses the previously mentioned four *dimensions* of hybrid with the architecture: Communities, connectivity, styles, and deployment.

### 1.2.1 Hybrid communities

Modern IT is often described as bi-modal, and sometimes multi-modal when describing the fact that there are different independent teams working at different velocities. Integration requirements come from people across the business. The change in hybrid integration is that it is no longer only the integration specialists who can resolve those requirements.

The reference architecture recognizes this aspect. In Figure 1-5 two gateways are shown, with a repeated structure of implementation capabilities beneath each. This configuration represents the fact that integration software must be modular and repeatable so that teams can take on their own integration, using just the components of the integration architecture that they need.

Fully managed tools should be available that enable straightforward integrations to be performed directly by business users and shadow IT departments aligned with the line of business. Complex integrations are often collaborative, and the tools should enable specialists to surface events or APIs into the business users integration environment.

Therefore, integration tools need to be available to everyone, at a level that is appropriate to their job role, and in such a way that it makes it easy to collaborate on integrations with other teams.

## 1.2.2 Hybrid connectivity

Integration is fundamentally about connecting systems and devices with other systems and devices. The target for hybrid integration is that it can connect to anything, anywhere. The connectivity components within the architecture must connect to anything from low-level connectivity to systems of record, yet equally easily take advantage of the simplicity of modern interfaces on cloud native systems software as a service (SaaS). In short, it must enable any system to become fully integrated into the integration landscape.

To achieve this goal, connectivity needs to be built in a modular fashion such that common connectors can be reused across the architecture. However, these connectors need to be provided in different types depending on the type of user. As an experienced integration specialist, you should be able to connect to a SaaS-based CRM system from the lowest level integration and use the full richness of the API. Yet it should also be possible to simply wire connections to that same CRM systems as a business user. The connectivity layer is common in each repeated instance of the implementation, but it is presented differently to different users based on their experience and job role requirements.

## 1.2.3 Hybrid styles

Previously discrete patterns of integration often need to be blended together to resolve complex solutions. Enterprise application integration might be used to create a data synchronization between systems, yet those systems might also be accessed in real time by APIs. Events being passed between systems might need to be addressed individually for latency during online periods, yet in bulk to manage high volume scenarios. The tools and runtimes need to be able to span across real-time APIs, events, and bulk data patterns to ensure that all scenarios can be covered.

## 1.2.4 Hybrid deployment

Modern applications are deployed across a broad landscape, so the accompanying integration components must have equally flexible installation options on metal, in virtual machines, or in containers. The components should be as easy to run on-premises, as in common cloud infrastructures. Patternized deployment should enable automation such that ad hoc environment creation becomes simple and mechanical. The components should have simple ways to integrate within, and across network and security boundaries. Finally, there should be different levels of managed services from self-managed infrastructure right through to fully managed software-as-a-service.

# 1.3 How is IBM addressing hybrid requirements in its integration portfolio?

IBM has dramatically simplified the way customers acquire the range of modular integration components they need.

*IBM Application Integration Suite* offers application integration, cloud connectivity, and API management features under one part number. This offering is well-aligned to customers who are pursuing API economy and cloud applications and need integration tools at the heart of their solution. This offering is among the simplest yet most complete and cost effective offerings in the market to help customers achieve their digital transformation goals.

The suite integrates well with the IBM Messaging portfolio, which provides industry-leading enterprise and platform messaging capabilities with the broadest array of deployment options on the market.

These offerings provide the following advantages:

- ▶ Enable integration by the full range of different roles within the organization
- ▶ Enable connectivity to the broadest range of systems
- ▶ Cater to every type of integration styles
- ▶ Can be deployed flexibly across all major platforms or purchased as fully managed services

The products are described in detail in Chapter 3, “IBM Hybrid Cloud and Integration Portfolio” on page 25. The later chapters (Part 2, “Hybrid integration scenarios” on page 53) describe how to build specific scenarios by using the components of the suite across common hybrid integration use cases.



## Hybrid integration use cases

This chapter provides an introduction to hybrid integration and present the three most common use cases. This chapter has the following sections:

- ▶ Introduction to hybrid Integration use cases
- ▶ Use case A: Joining the API economy
- ▶ Use case B: Improving productivity
- ▶ Use case C: Refactoring for innovation

## 2.1 Introduction to hybrid Integration use cases

Figure 2-1 shows the three most common use cases found in enterprises that use hybrid integration to further their digital transformation programs.

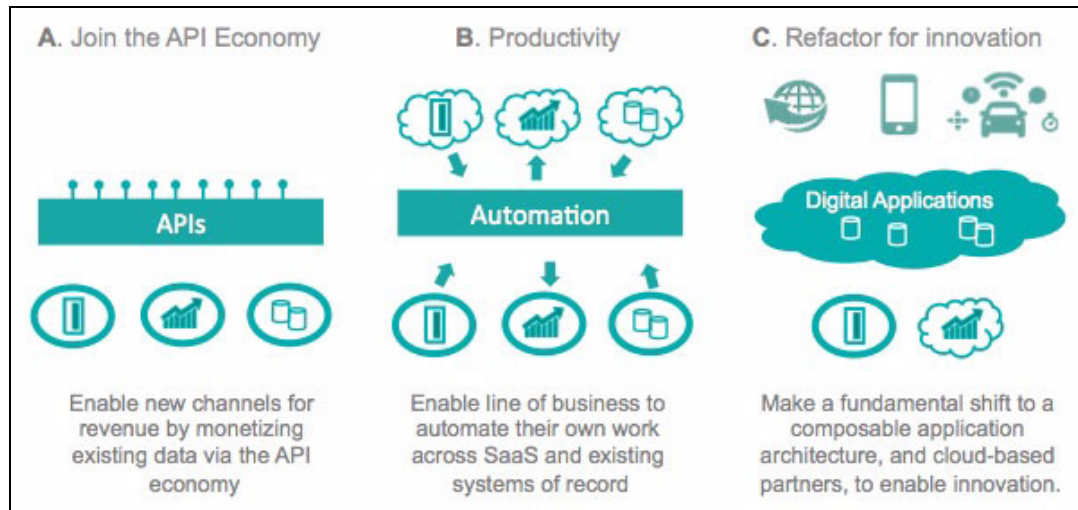


Figure 2-1 Most common use cases found in enterprises that use hybrid integration

This section covers each of these use cases. In the scenarios later, we choose an example of each use case to build using products from the IBM hybrid integration portfolio.

## 2.2 Use case A: Joining the API economy

This is a core use case to hybrid integration because the business initiative falls almost entirely into the integration domain. For this reason, the hybrid integration platform is important across the entire initiative.

As a first example, a company wants to introduce a new revenue opportunity by bringing their current data and functions into the API economy. Commonplace examples would include APIs making product catalogs available, APIs to enable ordering goods, and APIs to enable payments and more. What unique data or function does your company perform? Today it might be something you only use internally, but if it could be made available for anyone to include in their applications by using an API, would it provide sufficiently significant value so they would pay for it? Would it increase your market share? Would simply making it available to external developers further your reach? By enabling developers beyond the walls of the enterprise to explore your APIs, might new business models evolve that you would never have imagined on your own?

Entering the API economy requires, as a minimum, the ability to expose data and functions beyond the walls of the enterprise by using APIs, but what is really happening beneath the surface? The company must either hold unique data, or they must be able to perform a unique function that they can expose as an API. The simplest scenario is that this data or function already exists in one or more of their existing systems of record and all they need to do is unlock it as shown in Figure 2-2.

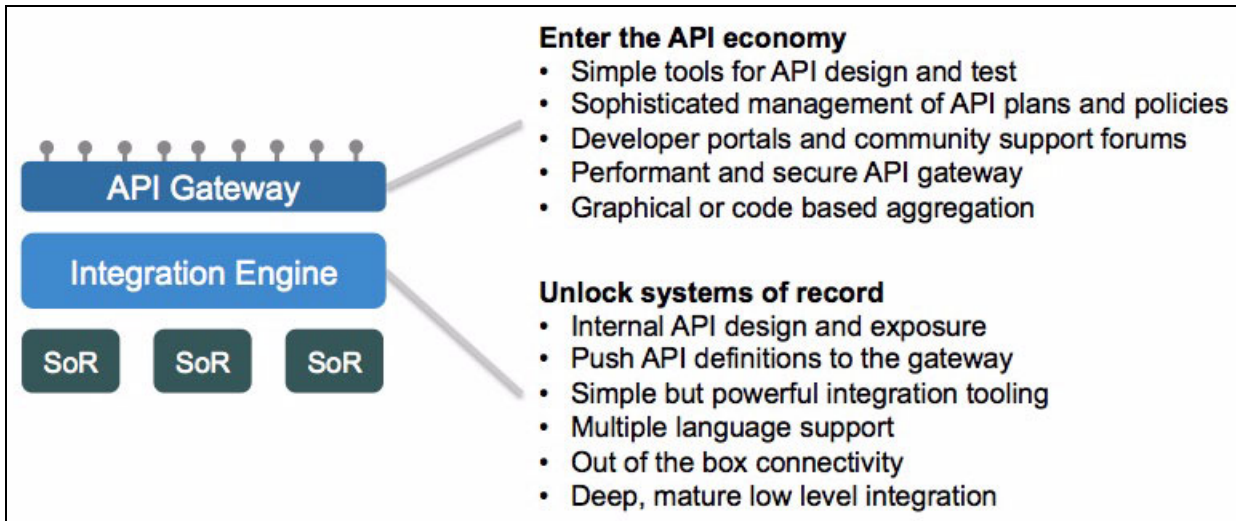


Figure 2-2 Use case A: Joining the API economy

In this scenario, IBM Integration Bus would typically be used as the integration engine to unlock the data in the systems of record. Then API Connect would surface the APIs by using its gateway, and provide policy management and developer portal capabilities to safely and simply bring the APIs into the public domain. This section explores those steps in more detail.

## 2.2.1 Unlocking systems of record

IBM Integration Bus has been evolving to fulfill this role for decades. It comes with a vast array of pre-built configurable connectivity to common systems that enable you to integrate to almost any system within an enterprise whether modern or legacy, with support for a huge range of ready for use protocols and transports.

Surprisingly, its large number of capabilities does not prevent it from being quick to install and simple to use. Patterns and tutorials are available for common needs, and a vibrant community exists for exchanging common practices through blogs, forums, and repositories such as GitHub. In fact, there is no license cost for developer use, which further encourages an active community.

Most integration needs can be satisfied by using powerful intuitive graphical tools for mapping, data formatting and parsing, aggregation, routing, and so on. At the other end of the scale, there is a suite of different programming languages available for more complex tasks.

A vast array of synchronous protocols is provided from modern REST or SOAP-based HTTP communication down to low-level support for databases, TCP/IP, CORBA, and many more besides. It has deep native asynchronous messaging support based on open standards such as JMS, MQTT, and Kafka. IBM MQ is used as a de-facto messaging capability by most enterprises for robust asynchronous communication.

IBM Integration Bus also provides pre-built connectors for enterprise applications such as SAP, Siebel, and also cloud based applications such as Salesforce. It also uses a common connector framework enabling new connectors to be quickly created, or sourced from the partner community.

IBM Integration Bus can make integrations available through an equally broad variety of protocols, although in this use case, we would most likely use RESTful APIs, or SOAP web services. These can be defined bottom-up, taking existing integrations and creating REST APIs from them; or by importing existing definitions (for example in Swagger format), and implementing the underlying integrations to that specification. These integrations are then available to use directly within the enterprise (a common pattern for IBM Integration Bus) or expose these APIs to developers beyond the enterprise (as in this use case), which requires a further API exposure layer as described in the next section.

This book focuses on the use of IBM Integration Bus to connect to deep back end systems. However, if a significant portion of your integration is to business assets and data on the IBM z/OS® platform, you should also consider z/OS Connect Enterprise Edition (z/OS Connect EE). IBM Integration Bus and z/OS Connect EE play complementary roles. z/OS Connect EE enables granular exposure and control over the low level services and APIs from the z/OS platform. Due to the Open API specification that both honor, these can then be seamlessly brought into IBM Integration Bus and composed together and combined with other sources to create the rich, high value APIs and services required by the enterprise and beyond. For information about on z/OS Connect EE's capabilities, and guidance on when and where to use it based on the same hybrid integration reference architecture, see *IBM z Systems Integration Guide for the Mobile and API Economy*, REDP-5319.

## 2.2.2 Securely surfacing APIs

IBM API Connect provides a simple way for developers to discover, sign up for, and use APIs. Developers are provided with an intuitive portal within which they can explore the catalog of APIs, their usage, and what plans are available to subscribe to.

The definitions of APIs created by IBM Integration Bus can be easily pushed into IBM API Connect, and then configured for exposure to a broad audience of external developers.

Although IBM API Connect comes with its own lightweight gateway, it is often preferred to use IBM DataPower for external API exposure because it provides a robust, secure, and performant gateway. IBM DataPower is typically placed at the edge of enterprise (in the DMZ), and is often already in place performing other gateway functions.

IBM API Connect is also available in a completely cloud-based form factor, as a software as a service (SaaS) offering. This offering can reach into the enterprise through a secure gateway to expose APIs from within the enterprise. This structure removes any need for an infrastructure initiative to get the software in place, and enables an immediate start.

When speaking of the API economy, we tend to think exclusively of monetized public APIs. However, recognize that IBM API Connect is used just as often to control access to APIs within an enterprise to ensure that dependencies between business units are clear. Enterprises that in the past might have looked to a service registry for SOA services, often choose internal API exposure as a simpler way of retaining control of relationships between applications. Equally, external exposure of APIs is often used to provide simpler sign-up processes for partners, without necessarily making the APIs available to the general public.



## 2.2.3 Extending the use case: Creating more valuable “interaction APIs”

The section has covered how the use case would be implemented in its simplest form, unlocking data from the systems of record using IBM Integration Bus to making them available as internal APIs, and then exposing them in their current state externally with IBM API Connect. What if an enterprise has more complex requirements?

What is often created is a two-layered model as shown in Figure 2-3:

1. **System APIs:** At the lowest level, “system APIs” need to be exposed robustly from the Systems of Record by using deeper integration specialist skills and overcoming complex protocol and data format challenges. As discussed earlier, IBM Integration Bus remains the core engine for performing this complex connectivity to systems of record, and surfacing them as rich System APIs.
2. **Interaction APIs:** These APIs are then composed together to form “interaction APIs” that are more useful to the typical applications and channels they serve. So now you start to see the developer role involved not just in using the System APIs for their applications, but also in building new applications that surface more business and channel-centric valuable “interaction APIs.” IBM API Connect is more than just the gateway and portal for APIs. It includes API creation capabilities as well. Interaction APIs can be composed by using a powerful graphical flow editor, or by using community grown patterns such as the Node.js LoopBack® framework, to provide maximum developer freedom and productivity. This feature plays well into the microservice architectures commonly chosen by the “digital teams” mentioned in the introduction.

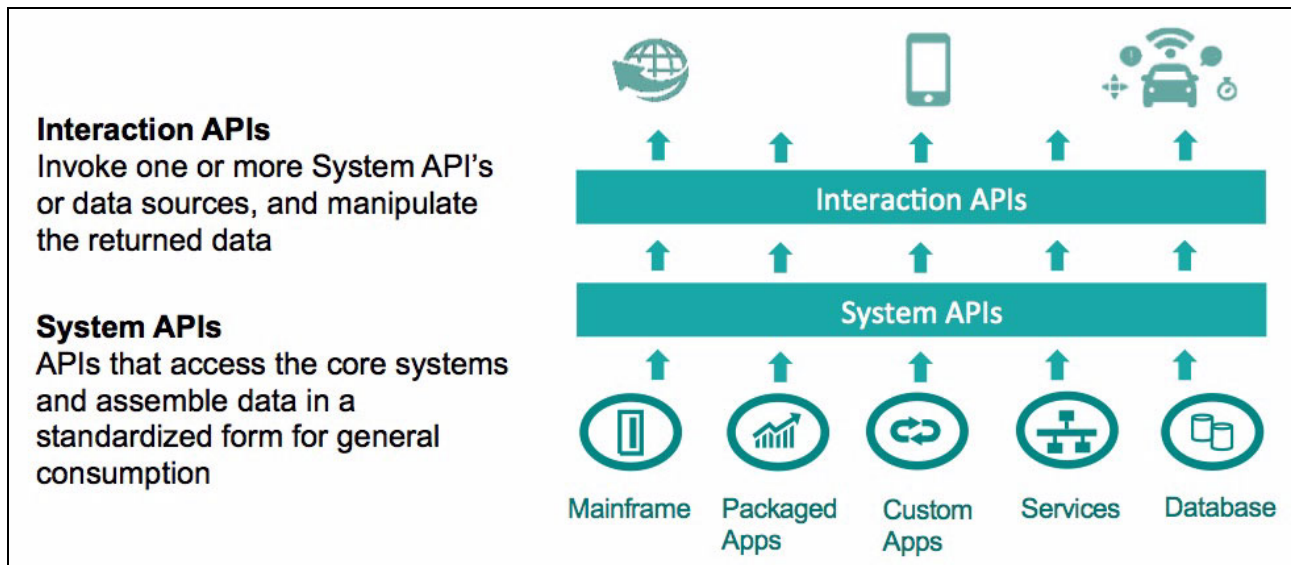


Figure 2-3 Two layered models

## 2.2.4 A hybrid landscape for API components

It is important to recognize just how flexible this architecture can be in relation to where the components are located, and how many different levels of sophistication can be achieved.

The simplest configuration back is shown in Figure 2-2 on page 13. However, you commonly see a split in roles. Often, integration specialists from the enterprise IT team create the lower-level integrations that result in System APIs. This process is typically done by using IBM Integration Bus, which although traditionally is often placed on premises, can also now be leveraged as a managed cloud offering.

The more business-focused interaction APIs are then often created by separate IT teams that are owned by the line of business. These teams have in the past been described as “shadow IT,” but are now often more commonly recognized as “digital IT.” This split between enterprise IT and digital IT is typically described as “bi-modal IT,” which inevitably works its way into the architecture as well, as shown in Figure 2-4.

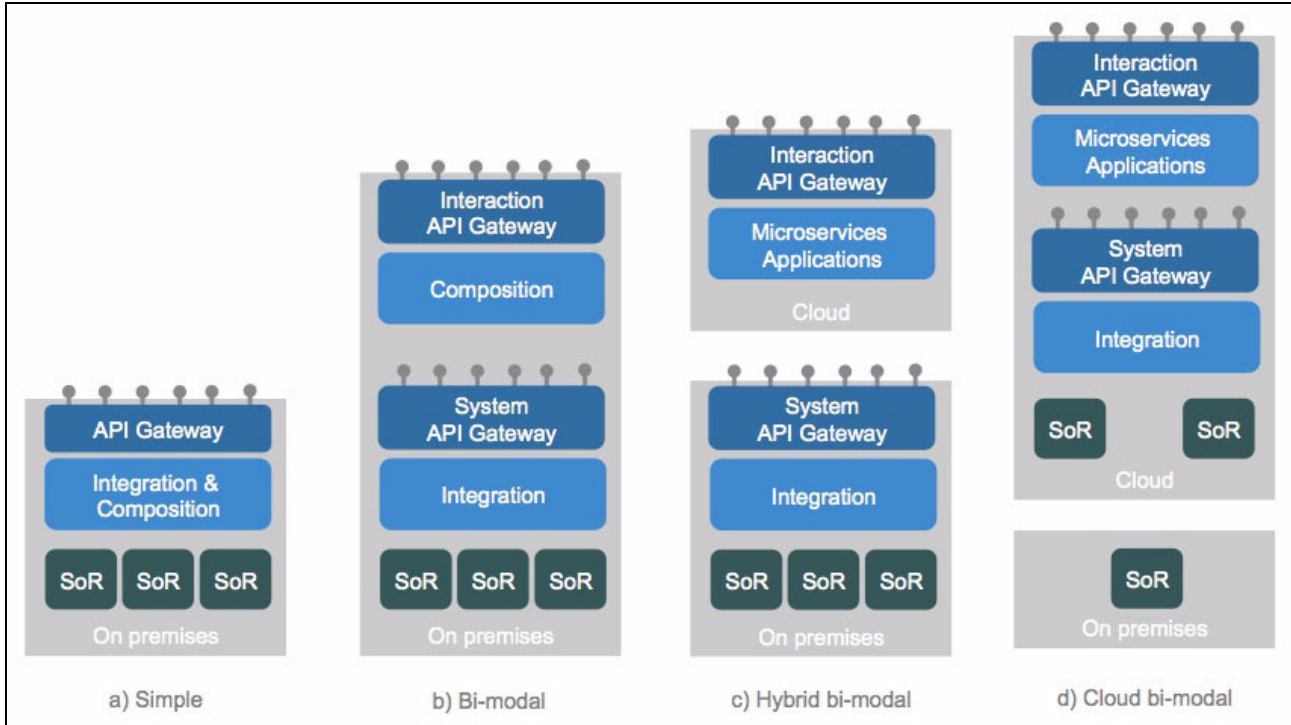


Figure 2-4 Bi-modal IT

To enable this flexibility in the architecture, all the products within IBM Application Integration Suite have both on-premises license and managed cloud offerings. For example, IBM Integration Bus can be installed traditionally on premises, or in virtual images, or in Docker containers created on a cloud-based infrastructure as a service offering such as IBM SoftLayer, or any of many other supported providers. There is also a managed service, IBM Integration Bus on Cloud, that IBM uses to look after the deployment and maintenance of the containers. The customer simply requests capacity when they need it.

For the most sophisticated APIs, you might need more than simple composition, and might need to write new applications. Shown as c) and d) in Figure 2-4, microservices represent a common architecture for complex APIs, enabling the rapid creation of more composable, scalable solutions. Microservices are described further in use case C.

Lightweight runtimes suitable for creation of microservice components in Java and Node.js are provided within IBM API Connect, for more sophisticated API implementations. In addition to the on-premises licensed product, there is also a fully managed cloud native version of IBM API Connect to enable flexible deployment options. IBM API Connect is also the API capability within IBM Bluemix. IBM Bluemix provides a ready made, managed platform as a service for building modern applications. It is suited to hosting microservice architectures. It also intelligently incorporates a catalog of capabilities for enhancing your applications that include data, analytics, cognitive, Internet of Things, and many more. Now the only limit to the capability of your APIs is your imagination.

## 2.3 Use case B: Improving productivity

This use case focuses on how to enable line of business to automate their work across SaaS applications and even to existing systems of record. It uses IBM App Connect to enable business users to automate work between common SaaS applications. More complex enterprise applications can also then be made available to IBM App Connect by exposing their triggers and actions through IBM Integration Bus.

Although there is inevitably a lot of focus around the API economy, a significant amount of integration is behind the scenes. Lines of business need to focus on innovation rather than on day-to-day processes. In today's culture of demanding immediate results, the ability of a business to streamline its processes is a key differentiator.

Background integration is everywhere. Few business processes occur only with a single system, so something somewhere needs to move data from one system to another for the next step in the process to occur. This type of integration is happening all the time. How do the new sales that you log in the call center become orders in the order management system? How do those orders become shipments in the logistics systems?

Traditional enterprise integrations were costly and challenging because they joined complex and old systems together that had not originally been designed around this type of interaction. They would typically require dedicated IT projects to unpick the differences between complex data formats, interaction patterns, and security models. These are the costly enterprise integration requirements towards the left of Figure 2-5. Clearly only the integration projects with high business value are worth embarking on.

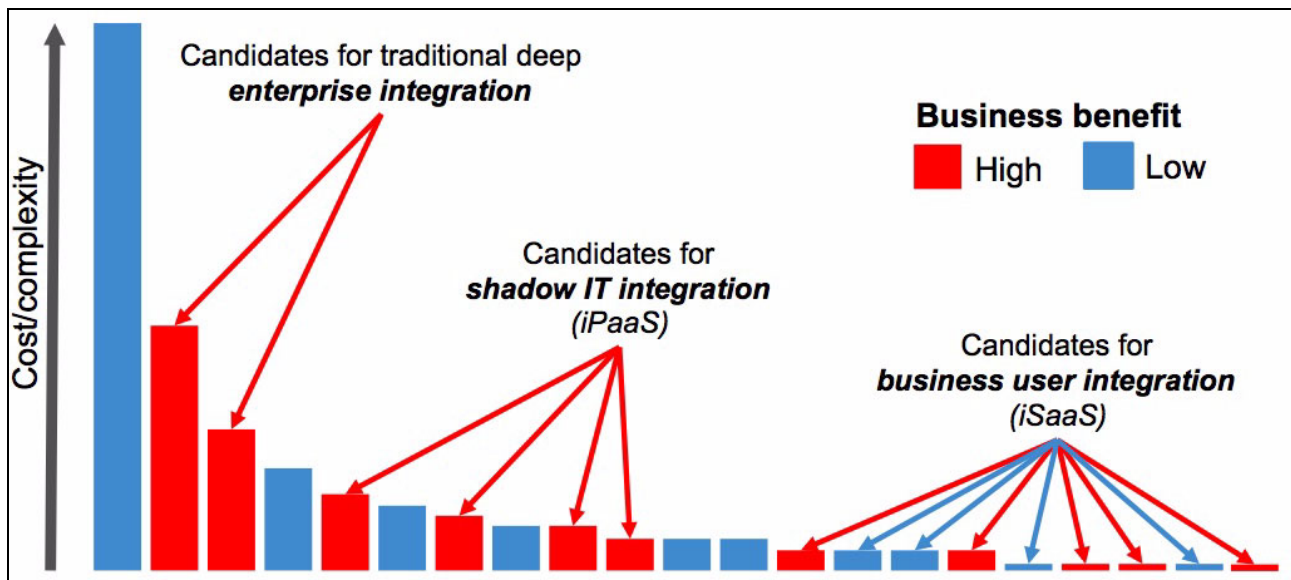


Figure 2-5 Use case B. Improving productivity

But if a company is to compete in a digital economy, its day-to-day tasks and processes need to be as slick and automated as possible. You need to ensure that you are only involving people to do added value tasks and ensure they can do those tasks efficiently. Users should not spend significant time moving data from one system to another.

However, with the increasing number of systems that businesses must work with, both on premises and in the cloud, more individualized unique integrations are required. It would be far too expensive to perform full scale integration projects for each of these integrations. In the past, these opportunities for improved efficiency would continue to be addressed only by more expensive human processes/efforts.

With modern tools and techniques, this “long tail” of unique and simpler integration requirements can finally be addressed. At the far right of the graph in Figure 2-5 on page 17 are many simple day-to-day tasks that could be simplified or automated by the business users themselves, with no need to get IT involved. This process is often described as *citizen integration*, and the tools that are used to do it are sometimes referred to as *integration software-as-a-service* (iSaaS). This is the primary subject of this use case (B). The middle ground of shadow IT integration and iPaaS is addressed in the final use case (C).

This use case focuses on how to enable the business users to perform some of their own integration automation as shown in Figure 2-6. Interestingly, the simplest route into this feature is for them to integrate cloud based SaaS applications, helping the enterprise to embrace external sources of data and function. This process works because SaaS applications have been built in an era where providing a standardized API is considered mandatory, which leads to a generalized and simplified integration approach. After this “cloud-to-cloud” integration has been introduced, you can then perform the more complex job of unlocking systems of record within the enterprise. You should be able to enable controlled access to a select set of enterprise applications such that the business can perform some of their own “cloud-to-ground” integrations as well.

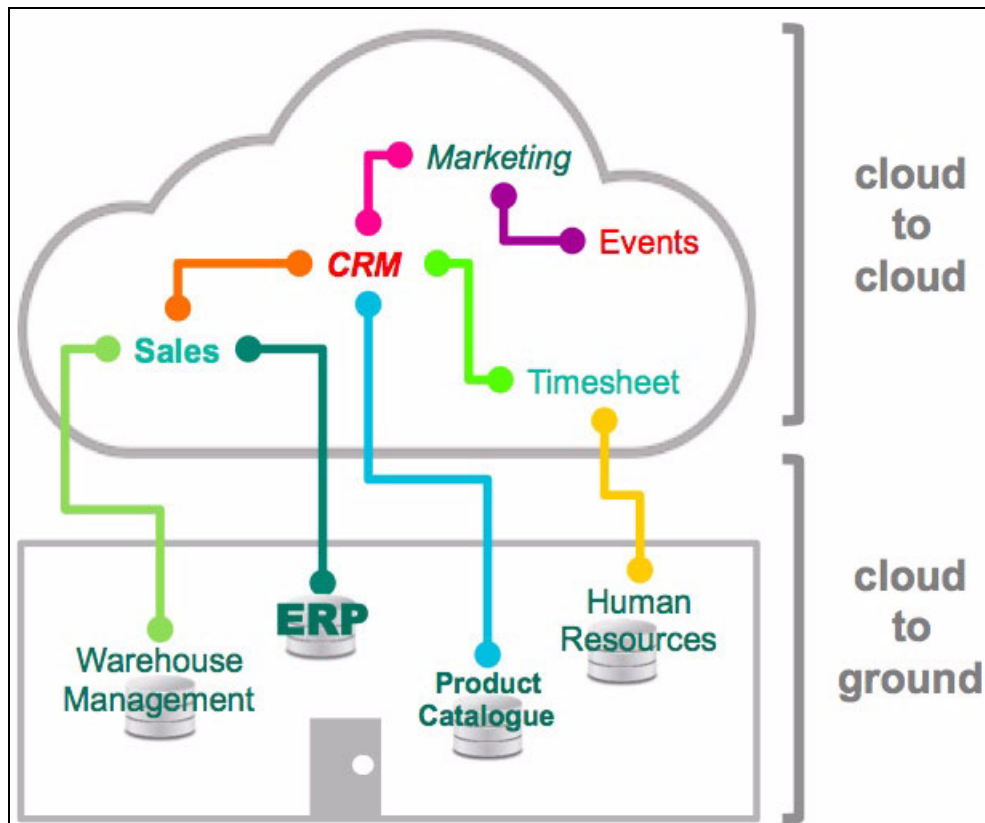


Figure 2-6 Enabling the business users to perform some of their own integration automation

As an example of cloud-to-cloud integration, take a marketing use case where you want a new SaaS application to organize corporate events for customers. This application will allow customers to register online in the corporate events management system. In addition, you want to make a note of those customers in your SaaS based marketing system so that you can keep in touch with them after the event as shown in Figure 2-7.

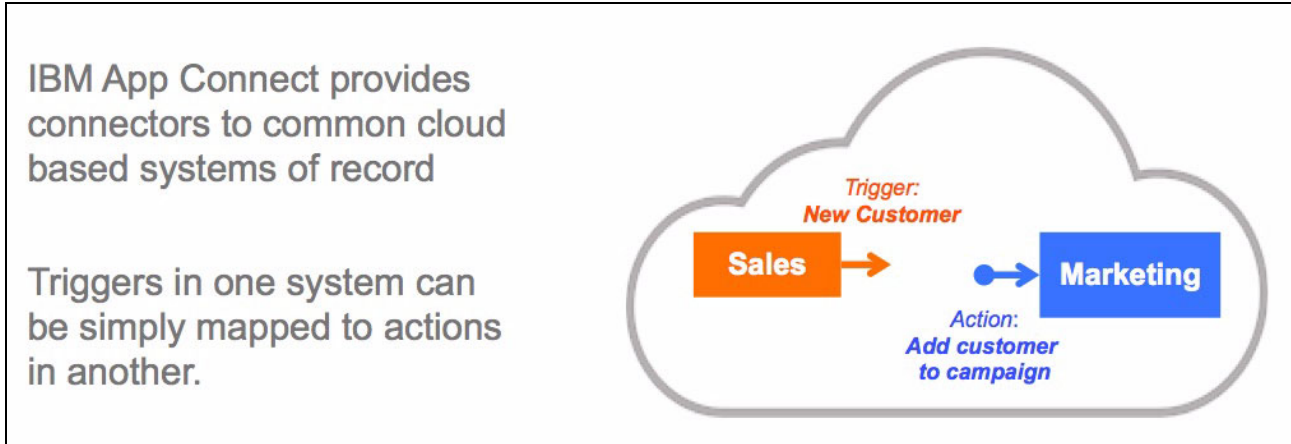


Figure 2-7 Cloud-to-cloud integration

It is probably not worth the cost of an IT project to set up that integration. However, it would be a terrible waste of resources to have one of your staff members do it manually, not to mention the potential for errors. It would be great if people in the line of business, or tech-savvy power users in your department could set up that integration themselves.

What kind of person are we talking about here? This is the automator from Figure 1-1 on page 4, who is a business person who needs to integrate the systems they use, but probably would not even talk of it as “integration.” They are probably not programmers. They are almost certainly not familiar with complex integration tools. This person wants to perform integration using tools no more difficult than those they use in their daily work such as email, spreadsheets, social media, and the SaaS applications they use regularly. This role is sometimes known as a “citizen integrator.”

Citizen integrators need a tool to create these connections between the applications they use. Those tools needs to be instantly available online, with no installation required. They should be engaging and simple to explore and to use, but still enable powerful integration capabilities.

This type of capability is sometimes referred to as iSaaS. At a minimum it should have these capabilities:

- ▶ Pre-built connectivity to common cloud-based systems of record.
- ▶ A native understanding of the events (triggers) that those systems create, and the actions that can be performed on them.
- ▶ Rapid ways to join those systems together, including choosing triggers/actions, setting up login credentials, mapping the data, and so on.
- ▶ Runtime management of integrations.
- ▶ Monitoring of what events have been processed.
- ▶ Ability to switch the flow of events on and off.

IBM App Connect is designed for exactly this purpose, providing a simple but powerful experience for business users to automate simple tasks. It enables users to join triggers from one system to actions in another. For example, the creation of new customers in Salesforce results in the customer data being automatically appended to a new row in a Google Spreadsheet. New connectors to common systems are being created regularly by IBM based on market demands. Over time, users are able to perform more sophisticated actions such as copying and synchronizing chosen data across systems.

### 2.3.1 Extending the use case: Cloud-to-ground (and ground-to-cloud)

Connecting between common cloud-based systems is already a huge productivity improvement for business users, but what if they could also connect to the unique systems of record within the enterprise? What about if you could listen for new products added to the enterprise product catalog, or perform updates to the inventory in the stock control system as shown in Figure 2-8? We need to find a way to bring the core data and the capabilities of those critical enterprise systems of record up to the business user.

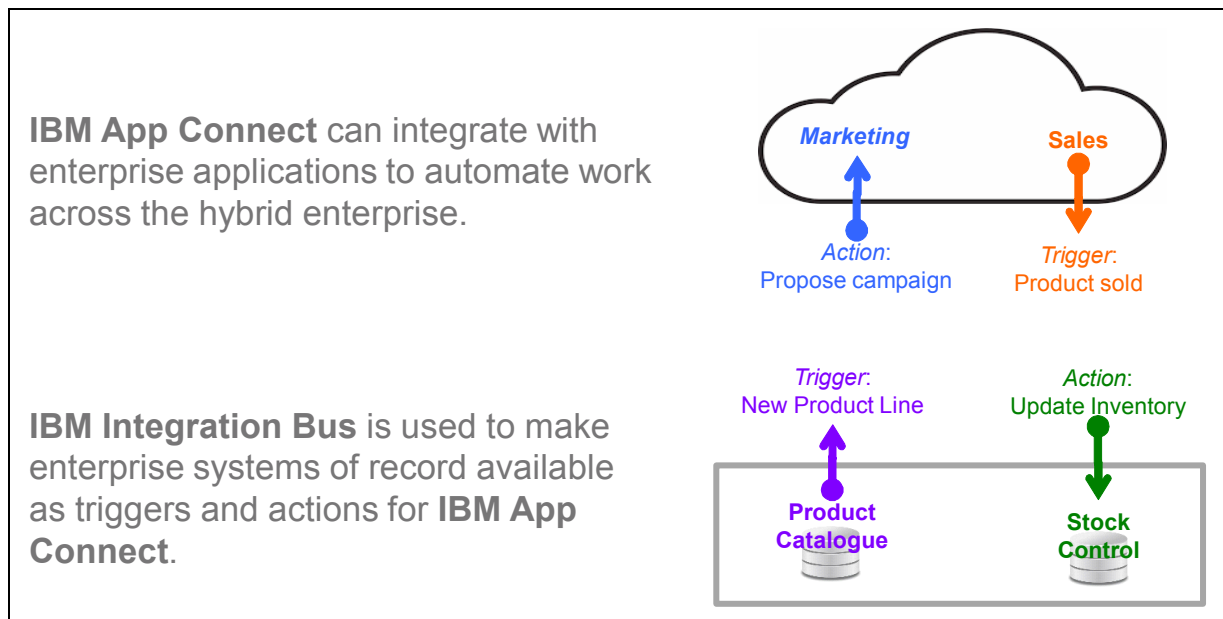


Figure 2-8 Ground-to-cloud integration

Because the complex enterprise systems might not have existing interfaces, they might need the assistance of the integration specialist to make the systems of record available in a form that the iSaaS can consume. The challenge is to make this process as easy as possible for both the business user and the integration specialist.

One of the simplest ways to make a system of record available to IBM App Connect is to surface it using IBM Integration Bus. In just a few simple steps, IBM Integration Bus flows can be made available to IBM App Connect users, either as triggers or as actions. The IBM Integration Bus flow might be performing complex integration, but after they are configured, they appear just like any other system, with the same simplicity of integration to the IBM App Connect user. Once they are configured, the business user (in the role of automator) can create all the integrations that they want without requiring the help of an integration specialist.

## 2.4 Use case C: Refactoring for innovation

This use case focuses on the new application architectures such as microservices that companies are using to enable faster innovation by building composable applications. Microservice applications are not built in isolation. They need access to data, and they need to be made available to other applications through APIs. This use case covers the components of the IBM hybrid integration portfolio that can be used to provide foundational capabilities around a microservices architecture.

Many businesses have successfully moved all or part of their operations to a completely online, self-service model, including banks, shopping, insurance, and travel booking. These businesses now have self-service capabilities that enable the consumer to transfer money, purchase items, get quotes, and book trips.

Moving a business online is just the first step in modern e-commerce. What makes a business stand out is the pace and creativity of their innovation in online products. For example, consider the now well-known disruption to the taxi industry. To be able to order a taxi online is not interesting in itself. Enabling a new demographic of drivers by providing a simple application that combines location awareness, satellite navigation, and cashless payment is a true disruption to an industry. Being able to rapidly add features like driver and passenger rating systems, surge pricing, ride sharing, and multiple taxi types provide advantages as competitors move into the new market.

So what does the underlying IT look like to enable innovation at this pace? A business must be able to rapidly compose new applications from what they already have. Equally important, they must be able to embrace partner capabilities and add them into their solutions easily. You need to look to modern application architectures, and at ways to embrace external sources of data and capabilities into your solutions. With all of these competing sources of potentially duplicate data, you also must consider how to synchronize your systems of record.

This use case is noticeably different in scale to use cases A and B. An initiative with as broad a scope as this involves many of the activities that are mentioned in the descriptions of the other use cases. Innovations might include an API channel and might require automation, so this use case is a superset of the other use cases combined with additional capabilities, such as event streaming.

### 2.4.1 Modernizing applications for composability

New application architectures such as microservices have emerged to enable the creation of more composable applications. Complex applications are broken down into self-contained microservice components, which are each focused on doing just one thing well. This technique enables new ideas to be implemented in isolation, providing greater agility, scalability, and resilience (Figure 2-9).

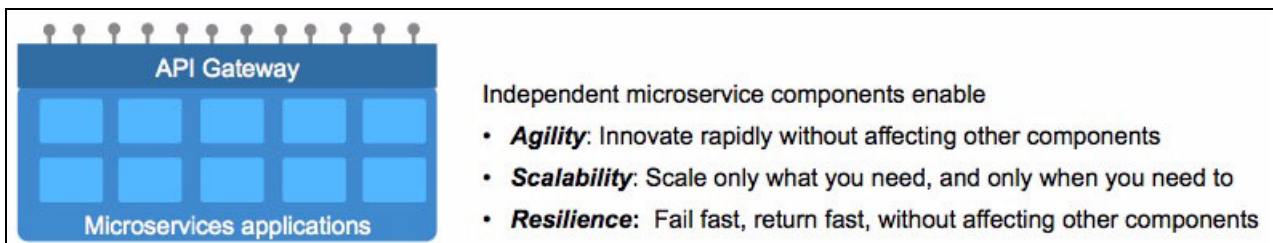


Figure 2-9 Microservices applications

These components can then be joined into solutions with more elastic scalability, and more flexible resilience models. This process allows you to innovate faster, scale the ideas that take off, and ensure that they meet the levels of service that your reputation depends on.

Microservices architecture's relationship with integration is complex. The distinction between it and initiatives such as service-oriented architecture can be hard to understand as described in “Microservices, SOA, and APIs: Friends or enemies?,” but at least two clear integration requirements emerge:

- ▶ The functions that are provided by a microservice is typically exposed as an API. This configuration implies the use of an API management capability such as IBM API Connect to surface the APIs in a rich and self-administrable way. This decentralization of the administration of APIs is critical for agility both from the perspective of the microservices implementers, and the business partners wanting to use those APIs. This aspect was already covered when discussing Interaction APIs in use case A.
- ▶ In order for microservices to be truly independent, they might need copies of data from other sources. This issue is discussed in the following two sections.

To read “Microservices, SOA, and APIs: Friends or enemies?,” use this link:

<https://ibm.biz/MicroservicesVsSoa>

## 2.4.2 Synchronizing systems of record

Applications are unlikely to provide much value if they do not have access to data, and the same is true for microservice components. It might be tempting to say that because most sources are exposed with APIs, microservices can easily access all the data that they want. However, this assumption ignores one of the most important design aspects of microservices: They must be independent. If they call APIs at run time, they depend on them. If the API is not available, then so is the microservice. So in fact, although APIs are sometimes used by microservices, other more decoupled integration patterns are often more appropriate.



For more sophisticated microservice components to be truly independent, they need a separate, yet up-to-date copy of the data to work with. A method is needed to capture and distribute events happening in the systems of record to ensure that the microservices can build and maintain their local data stores. These are traditionally well known integration patterns, such as “data sync” and “event streams,” but they are returning in subtle new forms for this new era of application architectures as shown in Figure 2-10.

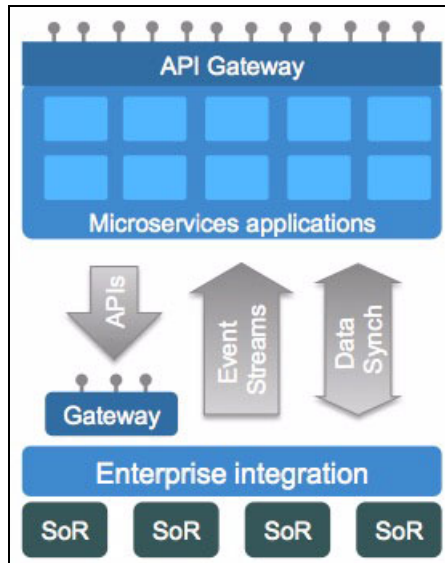


Figure 2-10 New era of application architectures

Although IBM Integration Bus enables real-time integration as described in use case A, do not forget that event driven integration has been at the core of IBM Integration Bus usage since its inception. It has the low-level connectivity required to listen to events occurring in a huge range of enterprise systems, transforming and routing them to their intended destination systems. This role is now reinvigorated as a mechanism to deliver the event streams required by microservice style applications. IBM Integration Bus is compatible with a host of different messaging platforms, and has close compatibility with IBM MQ. Messages can also be pushed up onto the cloud native application platforms used to host microservice components such as IBM Bluemix, and their native messaging capabilities such as IBM Message Hub, which is a product ionized version of the popular open source messaging standard Kafka.

**Note:** See the following links for the recently available KafkaProducer and KafkaConsumer nodes that have been provided in IBM Integration Bus 10.0.0.7:

<https://developer.ibm.com/integration/blog/2016/11/25/using-the-new-kafka-nodes-in-ibm-integration-bus-10-0-0-7/>

[http://www.ibm.com/support/knowledgecenter/SSMKHH\\_10.0.0/com.ibm.etools.mft.doc/bz91040\\_.htm](http://www.ibm.com/support/knowledgecenter/SSMKHH_10.0.0/com.ibm.etools.mft.doc/bz91040_.htm)

## 2.4.3 Embracing external sources

Ensuring that you can write your applications in a composable fashion and use data from your current systems of record is only half the process. To truly innovate, you need to embrace external sources of data and function. The example taxi company would not be successful if it is not able to incorporate location services, traffic routing, maps, and payment systems into its application. Attempting to build any of those for itself would have taken time away from the core value proposition. It is by combining forces with creativity from outside of the organization that you are able to turn your existing assets into truly innovative solutions (Figure 2-11).

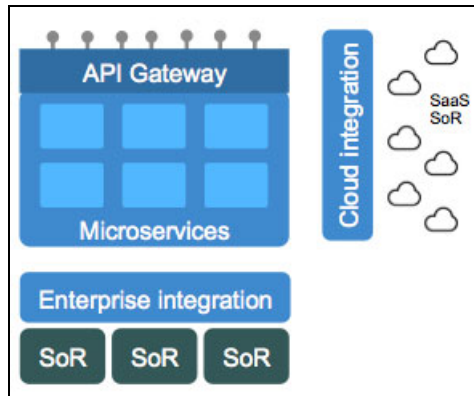


Figure 2-11 Integrating external sources

Composable applications need simple ways to integrate with data and functions from partner systems. Most modern internet-accessible partners provide APIs as a core part of their offering. However, the existence of an API to connect to is only the beginning of the journey. You still need to handle security credentials, map data, manage reliable updates, aggregate API calls to make them more usable, and so on. In addition, APIs are all implemented in subtly different ways, despite conforming to the same high-level standards. This concern is especially relevant when listening to events happening in external systems because the ways to represent events are many and varied. You could write code for this integration, but it would make more sense to use pre-built connectors. Using common integration tools provides the same experience regardless of the system connected to, and provides the necessary operational monitoring immediately.

IBM Application Integration Suite contains purpose-built cloud connectors to a vast range of common cloud-based services, and a clean, simple visual environment. External cloud-based systems can be easily connected together, or bound into enterprise systems by using configuration rather than code.

## 2.4.4 Innovating new solutions

A microservice architecture without the surrounding integration infrastructure would be limited in scope. The integration capabilities within IBM Application Integration Suite have enabled applications to accomplish these goals:

- ▶ Expose capabilities as secure yet easily configurable APIs
- ▶ Synchronize data with systems of record
- ▶ Gain access to external sources of data to enrich their functions

We now have everything in place to enable composition of more far reaching innovative solutions.



# IBM Hybrid Cloud and Integration Portfolio

IBM offers a spectrum of enterprise grade Hybrid Cloud capabilities. This chapter describes the IBM Hybrid Cloud and Integration Portfolio and has the following sections:

- ▶ Introduction
- ▶ IBM Application Integration Suite
- ▶ IBM Messaging Portfolio
- ▶ IBM DataPower

## 3.1 Introduction

For most enterprises, the cloud deployment model is not the only form of IT that they have. They might have an established on premises IT operation that has been running for many years that encapsulates the business knowledge and processes that make their business unique in their market. It might be that regulatory pressure forces them to maintain certain parts of their business relevant information on their own premises and in-country. Many other motivations can exist for not moving to a full rollout in a single public cloud footprint, or not doing it now.

Another potentially interesting aspect of introducing cloud-style deployment strategies, even where they are not yet needed is the ability to clone system instances from standardized patterns that are more robust and easier to maintain. In addition, such systems can be machine generated in hours rather than weeks, requiring fewer skilled staff.

Therefore, most enterprises realize a need for a hybrid model that combines on-premises, off-premises, and traditional models as well as secure connectivity between systems of engagement and systems of record. IBM offers a variety of enterprise grade Hybrid Cloud capabilities. This chapter introduces the products that make up the IBM Hybrid Cloud and Integration Portfolio.

## 3.2 IBM Application Integration Suite

As shown in Figure 3-1, IBM Application Integration Suite provides the tools that enterprises need to connect cloud and on-premises applications, and to build, expose, and manage APIs. These are the all essential ingredients in digital transformation.

Enterprise Integration	API Management	Cloud Connectivity
✓ Deep mature connectivity	✓ Policy based traffic management	✓ Broad native cloud connectors
✓ Rich flow language support	✓ Integrated developer portal	✓ Non-specialist composition
✓ Advanced mediation capabilities	✓ Lifecycle management	✓ Pre-defined templates

Figure 3-1 Application Integration Suite offerings

Three established products are included in this offering:

- ▶ IBM Integration Bus: For deep enterprise connectivity
- ▶ IBM API Connect: For API exposure and management
- ▶ IBM App Connect: Bringing cloud connectivity to the non-integration specialist

These products cover the multiple dimensions of hybrid integration:

- ▶ Provide targeted experiences for the diversity of roles involved
- ▶ Enable multiple integration patterns and styles
- ▶ Provide rich enterprise and cloud connectivity
- ▶ Provide a breadth of deployment options

Figure 3-2 shows the various roles in Application Integration Suite.

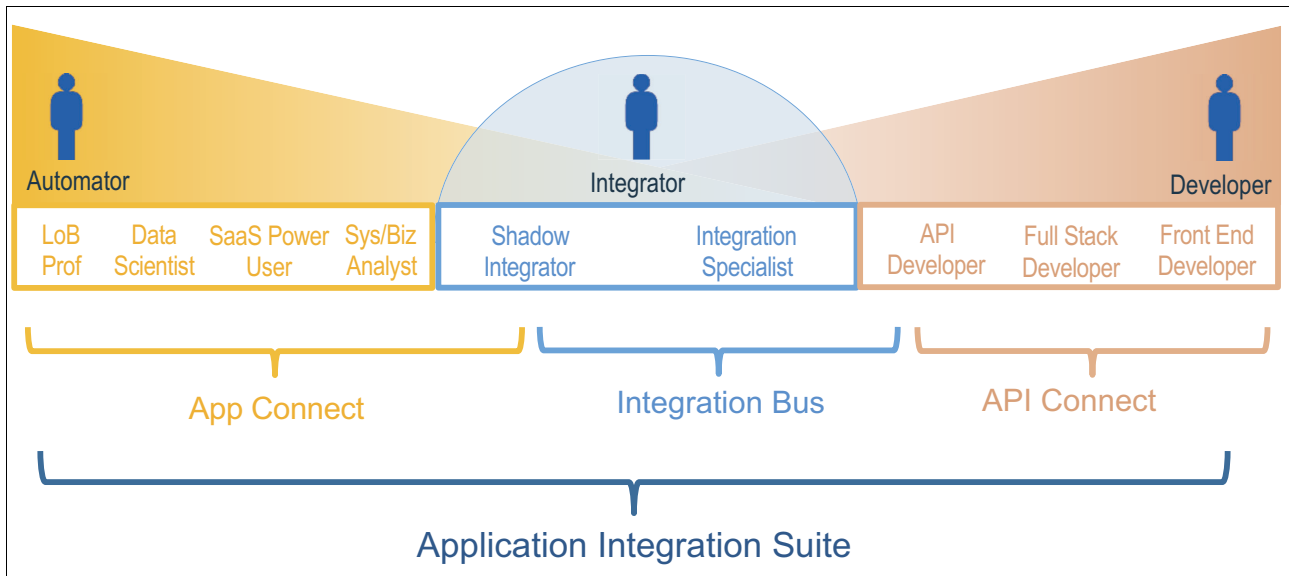


Figure 3-2 Hybrid integration personas

The suite provides the following capabilities and benefits:

- ▶ Securely integrate on-premises and cloud applications, enabling you to optimize resources and productivity in on-premises, software as a service (SaaS), and cloud models.
- ▶ Use the enterprise service bus (ESB) architectural pattern to eliminate costly point-to-point connections, which are expensive to maintain and resistant to change.
- ▶ Rapidly design and build APIs with essential security policies and share APIs across organizational boundaries and environments, while providing deep insight into API usage.
- ▶ Create microservice components and expose them as APIs by using Node.js Loopback and Express frameworks.
- ▶ Accelerate your integration by using pre-built patterns, rich connectors, and easy-to-use tools.
- ▶ Cohesion across the included components through adherence to common standards such as Swagger definitions for APIs and the introduction of a common connector framework.
- ▶ Various deployment options to meet hybrid needs from bare metal installations to virtual machines and containers.
- ▶ Available both as self-administered and as fully managed services.
- ▶ Freedom to move between on-premises and cloud offerings.
- ▶ Elastically scalable to enable costs to match your business model.

IBM Application Integration Suite provides the most flexible and cost effective way to encompass the broad set of integration requirements on all levels of a digital transformation program.

### 3.2.1 IBM Integration Bus

When the product we now know as IBM Integration Bus was first released in 2000 (under the name of IBM MQSeries® Integrator Version 2.0) it was positioned to become the all-encompassing integration tool for a then entirely on-premises corporate IT landscape. It was eventually able to connect *anything you can think of* to *anything else you can think of*. Today, this includes anything in the cloud and in the Internet of Things.

However, in the beginning it was limited to MQSeries (now IBM MQ) as supported transport technology. It made it easy and quick to build integration logic between applications that used IBM MQ but needed adjustments to their message formats or routing. Typical usage patterns would intercept request and response messages, perform transformations and possible enrichment with database contents, and pass the resulting messages on to their destinations.

Messages could also be routed to different destinations based on their contents or even distributed to multiple destinations simultaneously, even with a different format for each. Messages could also be saved in database tables, which allowed the creation of “utility” type services, such as database inquiries, updates, inserts, and deletions. By using IBM MQ’s support for global transactions, complex transactions of this nature could be protected by the eXtended Architecture (XA) protocol for two phase commits.

In general, IBM Integration Bus inherited all the strong features that IBM MQ was already renowned for, such as its speed, ease of use, and legendary reliability.

Integration developers use the Eclipse-based toolkit to create integration logic in the form of graphically built message flows. Depending on their specific function, these flows might need extra constructs to run:

- ▶ Programming code in Java, Extended Structured Query Language (ESQL), PHP Hypertext Preprocessor (PHP) or any .NET language
- ▶ XML Schema Definitions (XSDs) and Web Services Description Language (WSDL) files to describe message formats or services

All development constructs are collected in applications or libraries and eventually bundled into broker archives (*bar files*) that are then deployed to a runtime environment to be run, either to test or to run in production. Management of IBM Integration Bus runtime environments is shared between the toolkit and a browser-based management console. In addition, an API and command-line implementations are provided that support all administrative functions, so users can create their own administration tools and scripts.

Message flows are built by wiring together functional modules that are graphically represented by processing nodes through connection terminals with specific uses. Most nodes have an input terminal where they receive the message under processing and a number of output terminals that pass the message on to the next step in the flow. Terminals are also provided to connect to catch logic for exceptions or failure handling.

It is typical for IBM Integration Bus that for the key functions, such as transformation and routing, it provides a number of alternative ways to do them. Transformations, for instance, can be accomplished by using graphical mapping, stylesheets (XSLT), or a number of programming languages, such as Java, ESQL, PHP, or any of the .NET languages (under Windows.) A plug-in for IBM WebSphere® Transformation Extender is also available as an add-on.

A message flow always starts with an input node for a message source:

- ▶ A queue in IBM MQ or Java Message Service (JMS)
- ▶ An HTTP/S port
- ▶ A SOAP message source
- ▶ A file
- ▶ A TCP/IP socket
- ▶ A database trigger
- ▶ Many others

The input node parses the incoming message based on the configured format specifications. It then makes the result available as a message tree, which is an internal representation of the message contents without the specifics of the physical representation as Extensible Markup Language (XML), JavaScript Object Notation (JSON), tagged/delimited, fixed file, or any other data structure. During the flow's processing, this "logical" representation of a message is in force. Only at the end points where messages are put on an external transport again (such as IBM MQ, JMS, or file) will a physical representation be rendered again.

This output might not be the same format as used for the input message. Conversions between formats or, for instance, a change in XSD are common forms of message transformation. It is also normal that message fields are dropped or added, that logical encodings are altered (for example, date and time formats, state or country codes). Because it is possible to include external code libraries (for example, JAR files), more complex transformations such as compressing information into a compressed file format are likely.

All the assets that are created for an application or library are then collected in *broker archive* files (bar files in analogy to JAR files, war files, ear files, and so on). These files can be archived in version control systems and are sent to the runtime system for execution in test or production mode. The typical developer workstation runs the IBM Integration Bus toolkit and a runtime instance for local testing. However, deployments and all other functions can be performed against any runtime anywhere in the network, if the permissions allow. Thus, suitably authorized users can deploy bar files onto production servers. Bar files can be manipulated in certain ways by administrators without having to revert to the contained source objects to accommodate environment specifics, such as diverging naming conventions for names of queues, data sources, and so on.

You can use several methods to scale IBM Integration Bus throughput to impressive levels. The most easily implemented one is the *additional instances* parameter that can be configured to allow up to 256 extra threads running a flow simultaneously, processing additional messages if any are queued. Another option is to run extra operating system processes. If the applicable IBM Integration Bus license is of the "advanced" or "scale" type, then additional integration servers can be run under the integration node. Flows can then be deployed to one or more of them, increasing the number of parallel threads even further. Lastly, multiple brokers can be installed on separate hardware and can be coordinated through IBM MQ clusters or network load distributors.

The functions of IBM Integration Bus can fairly easily be extended by adding custom-made, possibly user-developed, parsers or processing nodes. For instance, WebSphere Transformation Extender is available to run as a plug-in node and a parser for IBM Integration Bus.

The graphical development environment leads to an easy and fast development cycle. The graphically represented message flows are to a large extent self-documenting, which leads to better team development dynamics and easier support. Comprehensive built-in testing facilities, such as tracing, step-by-step execution, and message recording, further improve developer productivity.

These productivity tools are available:

- ▶ An ever-growing library of executable samples
- ▶ Numerous wizards that fast track the start of a project
- ▶ A pattern capability that allows developers to generate message flows from IBM-provided or customer-authored patterns that represent common flow designs or architectures
- ▶ A comprehensive monitoring feature that can collect and emit information at any point in a message flow
- ▶ Message flow statistics, which are optionally extremely fine-grained
- ▶ Message record and replay

By the time that service-oriented architectures (SOAs) became popular and the need for a mediation tool called ESB was postulated (around 2005), it was discovered that IBM Integration Bus had been just that, and more, all along.

IBM Integration Bus now supports these transport 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)
- ▶ TCP/IP

Supported message formats include binary formats (C and COBOL), XML, and industry standards (including SWIFT, EDI, and HIPAA). Custom formats can also be defined.

IBM Integration Bus was an early adopter of the standard message description language DFDL. A public repository of DFDL definitions of data formats can be found on GitHub.

IBM Integration Bus's event handling features include the techniques that are used by the various input nodes to detect changes in the environment. From messaging protocols, which by their nature automatically alerts the broker of arriving messages to the polling of FTP servers or file directories or database triggers employed by the Database Input node, the most appropriate method is implemented for each transport medium. Another key feature is the ability to publish information in a number of publish/subscribe environments (such as IBM MQ and MQTT). Complex events can be configured by using the Collector node, where a set of rules can describe sets of incoming events that will then be combined to create one outgoing "complex" event.

High availability configurations of IBM Integration Bus can be created in a number of ways, most easily by using the high availability capability of an underlying multi-instance IBM MQ queue manager.

A built-in global cache facility is available to keep frequently used data readily available or to share information between components of a broker network. Alternatively, a separately available WebSphere eXtreme Scale data grid can be accessed and shared with non-IBM Integration Bus components. The use of a global cache can help overcome certain restrictions and affinity issues that result from the essentially stateless conceptual architecture of IBM Integration Bus. This restriction is routinely overcome in many popular use cases today.

Over the years, IBM Integration Bus has been enhanced by many added capabilities and usability improvements, while also becoming much more compact, agile, and performant.



Because it is the central integration hub in many large enterprise systems, IBM Integration Bus is typically found in on-premises deployments, where such enterprise systems are located. But in a hybrid integration context, IBM Integration Bus is also often found in private cloud implementations, such as on PureApp servers.

Lately, though, IBM has also been offering *IBM Integration Bus on Cloud* in a public cloud format on SoftLayer, running inside Docker containers and positioned for easy access from the Bluemix environment. This implementation is still evolving, and at this time does not fully support all of the capabilities of IBM Integration Bus on-premises. The main reason for this is that many of those capabilities are directed at accessing resources of the on-premises systems on which IBM Integration Bus is usually found.

Instead, IBM Integration Bus has been (and will continue to be) enhanced by a number of new capabilities that facilitate linking between IBM Integration Bus and various other systems that are typically found in the cloud, including many popular cloud native applications.

The list of these enhancements is already long and still growing. Here are some worth noting:

- ▶ Processing nodes are provided for a “callable flow” interface that makes it easy to switch processing between IBM Integration Bus nodes, whether they be on-premises or on cloud, using secure connectivity where required. For example, a flow running in the cloud might need access to an on-premises database or enterprise information system server. The best place to host these interfaces might be an on-premises integration node right next to the required resources in a callable flow. Another useful scenario might be an on-premises flow that needs to “farm out” some CPU-heavy workload to maintain its own responsiveness. It could do so by starting a callable flow in a cloud-based IBM Integration Bus instance.
- ▶ JSON message formats, REST APIs, and a quick and simple way to push APIs to API Connect.
- ▶ A Loopback Request node to access data through a Loopback connector. Available connectors include MongoDB, PostgreSQL, and MySQL.
- ▶ A Salesforce Request node to access data on your Salesforce.com account (only available in Application Integration Suite operation mode.)

### 3.2.2 IBM API Connect

IBM API Connect is an integrated offering that provides an API management solution and a microservices run time that address all the critical aspects of the API lifecycle for both on-premises and cloud environments. As shown in Figure 3-3, IBM API Connect offers capabilities to create, run, manage, and secure APIs and microservices. It also enables you to rapidly deploy APIs and simplify and centralize their administration across your organization.



Figure 3-3 API Connect capabilities overview

The development tools of IBM API Connect go beyond API Management and provide capabilities to define and test not only the secure access point, but also microservices that are implementing new business logic.

In particular, an API developer can use:

- ▶ A *Swagger Editor* and a *Policy Assembler* to define the interface of the API, specify security constraints, invocation rate limits, routing, data, and protocol transformations.
- ▶ *LoopBack Models* to implement the business logic of the APIs. LoopBack is a Node.js open source framework that is effective for the development of APIs because it embraces from the ground up the core concepts of REST.

The definition of all of these artifacts is supported by both a graphical user interface and command line to suit both novice and advanced users and be easily integrated in your existing DevOps tool chain.

The run time of IBM API Connect includes the four high-level logical components as shown in Figure 3-4 and described in this section.

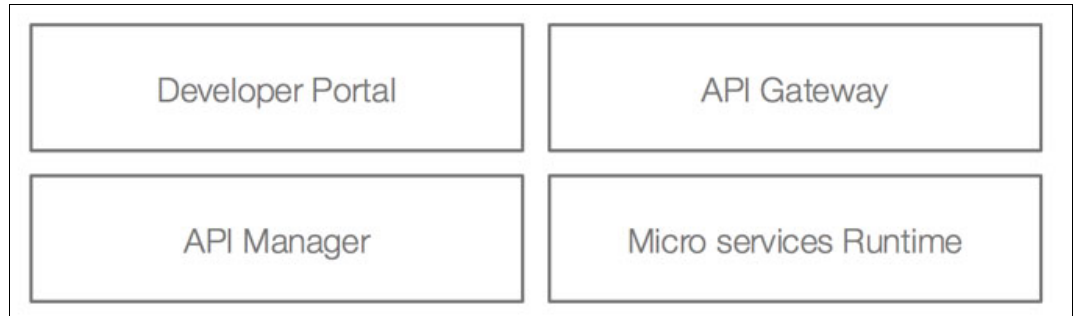


Figure 3-4 IBM API Connect logical components

### Developer Portal

The Developer Portal enables developers, who can be internal or external to the organization exposing the API, to discover and consume APIs in a *self-service* fashion. This process includes finding an interface, reviewing its documentation, subscribing to it, and obtaining the required access credentials. After the developer starts using the API, he can continue to refer to the Developer Portal to have statistics on his own API usage rate, receive updates on any API version change, communicate with his peers through forums, and open support tickets.

### API Manager

The API Manager is the component enabling an API Product Owner to be in control of the relationship with the API consumers. This process includes configuring who should be able to see and start an API, managing its subscription process, determining whether any money flows and under what circumstances, handling the migration between versions, running analytics on the actual usage metrics, and so on.

### API Gateway

Any API call goes through the API Gateway, the point of enforcement of access control policies. These policies include authentication and authorization of the caller, but also rate limiting to prevent a client to go beyond the entitlements that person has subscribed to. The Gateway is also responsible for collecting runtime metrics that are fed into the API Manager. After handling an API request, the gateway redirects the request to the components that implement its logic. Typically, this is a back end system or a microservice.

IBM API Connect includes two types of gateways:

- ▶ DataPower: An enterprise scale secure gateway that provides a public access point to your APIs
- ▶ The Micro Gateway: A lightweight Node.js gateway that is useful to support testing or the enforcement of policies for a single application

### Microservices run time

API Connect supports two types of Microservices run times:

- ▶ Node.js: Runs LoopBack applications
- ▶ Java Liberty: Deploys Java components

API Connect includes a Node.js run time and a “collective controller,” which a management component responsible for deploying and scaling LoopBack applications.

Although Node.js is successful for the implementation of *mobile back ends* (a use case for interaction APIs), many organizations will want to continue to use the wealth of Java assets they own or can access, both in terms of technical libraries and development skills. A choice of run times also gives you the ability to place workloads with different characteristics on the most appropriate technology. It is beyond the scope of this section to provide an in-depth analysis of the characteristics of Node.js versus Java. However, it is worth noting that the asynchronous nature of Node.js makes it best suited for applications that coordinate access to many external resources, while the threading model of Java makes it efficient for CPU-intensive workloads.

All of these components can be installed on premises, on a third-party cloud, or consumed as a managed service on IBM Bluemix (Public or Dedicated), providing the most flexible deployment and scalability model.

### 3.2.3 IBM App Connect

IBM App Connect is a web-based tool that enables non-technical business users to integrate applications to automate tedious and repetitive tasks that are typically done manually in many cases by *swivel chairing*, such as looking at data in one system and manually entering the data into another system. App Connect can be used to integrate cloud-based applications and custom application on-premises using a Secure Gateway to connect App Connect to the on-premises system. Here are a couple of examples of using App Connect:

- ▶ Send an email notifying people when a new sheet has been created in Google Sheets.
- ▶ Add a row in a Google Sheet when a new lead has been added to Salesforce.

You integrate applications by creating flows.

#### Trigger and action flows

A trigger is an event that occurs in one application that causes an action in another application. In the example mentioned earlier, a sheet being created in Google Sheets is an event that is a trigger for an action to occur in another application. In that example, the action is sending an email notification about the new sheet.

The flow is what links a trigger to an action and automates the execution of the action when the trigger occurs.

#### Integration in four steps

You can integrate your applications in four steps with App Connect:

1. Select applications to integrate
2. Connect to your accounts
3. Select the data that you want
4. Run it

### Select applications to integrate

Creating a flow starts by selecting the applications that you want to integrate. The first application is where the trigger will occur and the second is where you want the action to occur. Figure 3-5 shows the window in App Connect where you select applications to integrate.

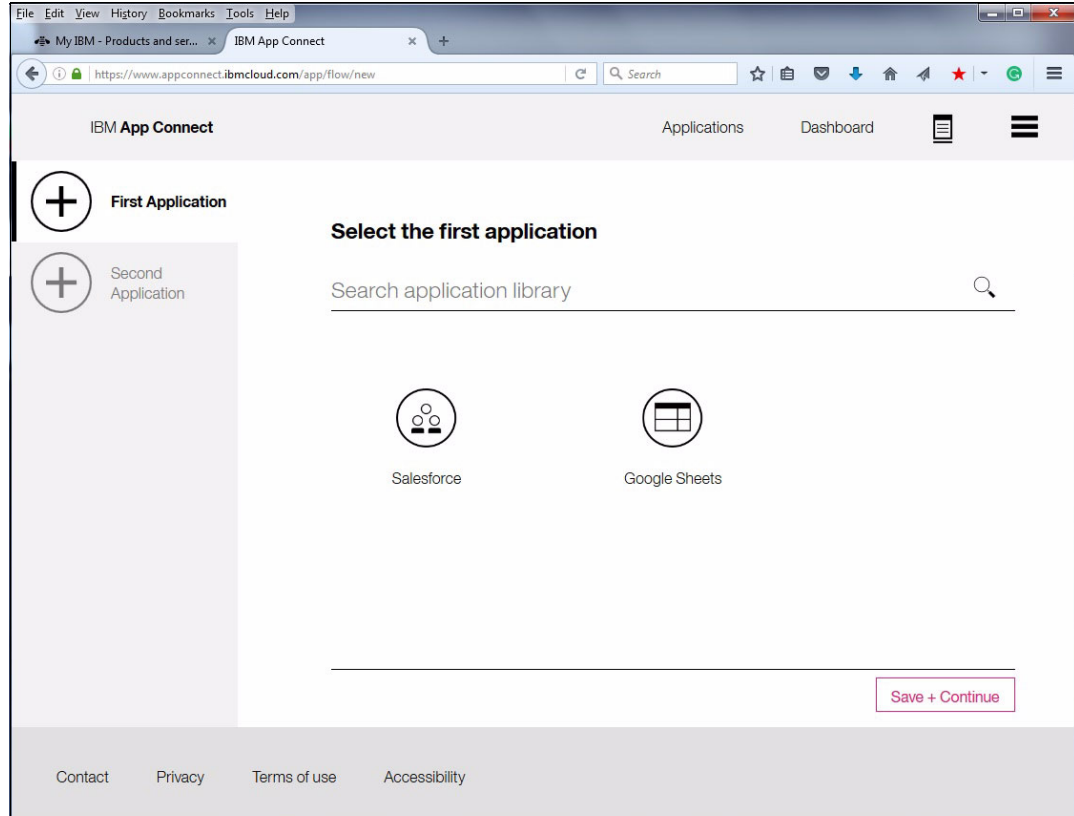


Figure 3-5 Select applications to integrate

### Connect to your accounts

For App Connect to automate tasks for you, it must be able to connect to the applications that you are trying to integrate with your account/credentials. You must connect an account for each application by authorizing App Connect to use the account by using OAuth Version 2.0.

Figure 3-6 shows the window in App Connect that enables you to connect an account with the applications in your flow.

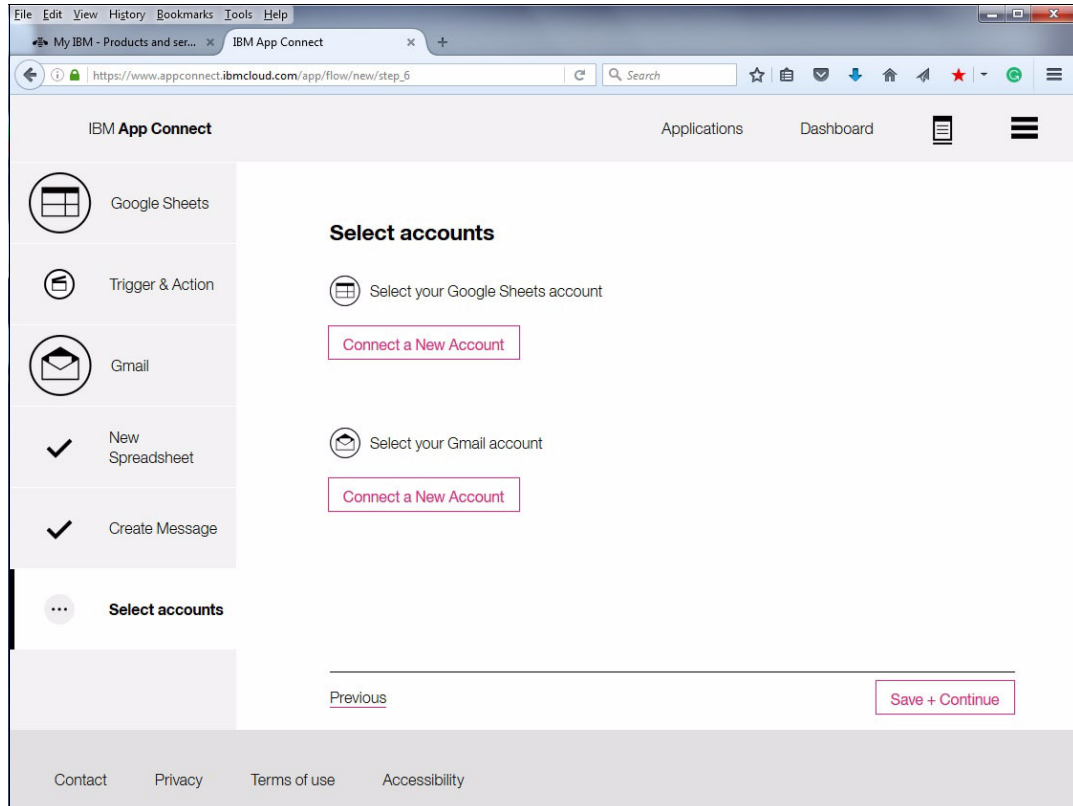


Figure 3-6 Connect application accounts

### Select the data that you want

Based on the application and trigger selected for that application, you might need to select and map data from your first application to your second application. In the Google Sheets example, the trigger is when a spreadsheet is created, and the action in Gmail is to send an email notification. As you can see in Figure 3-7, you need to specify the fields that are required to send the email, which are the **To**, **Subject**, and **Body** fields. You can insert fields from the new spreadsheet by entering a # and selecting the field from the list.

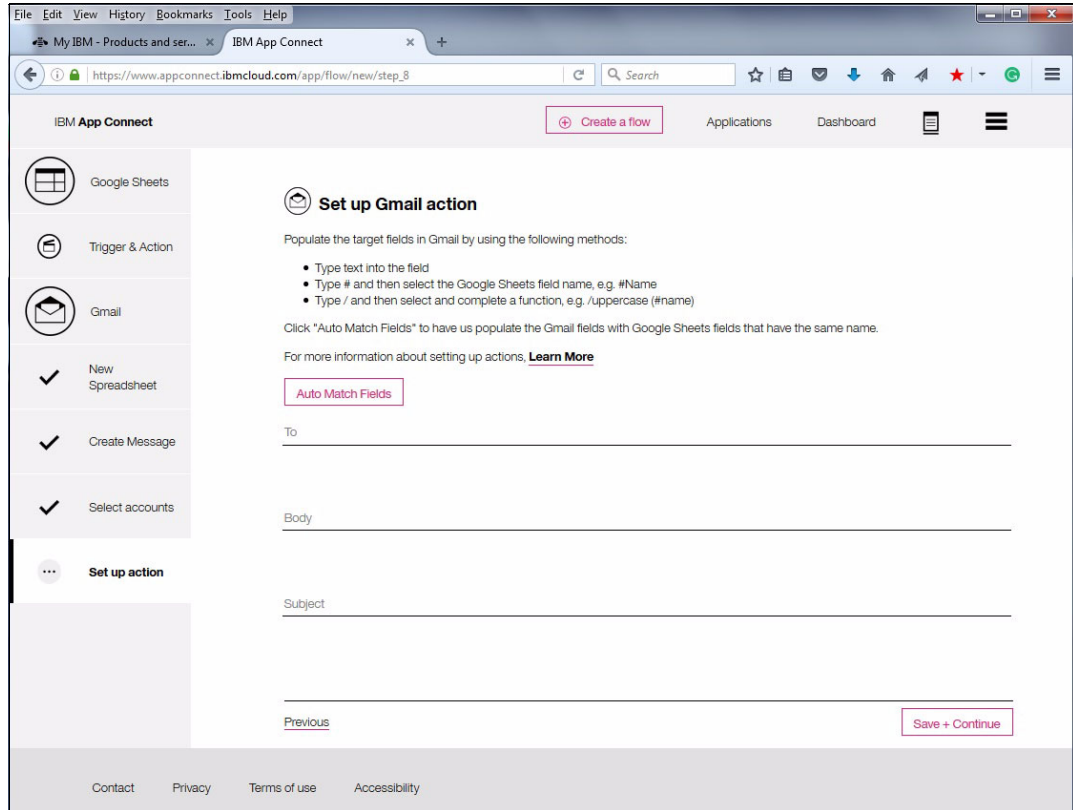


Figure 3-7 Select/specify any data

## Run it

Flows run in the cloud in App Connect. You do not have to run anything. You simply turn on the flow. Figure 3-8 shows the App Connect Dashboard that displays all of your flows. You can toggle a flow on or off from this dashboard.

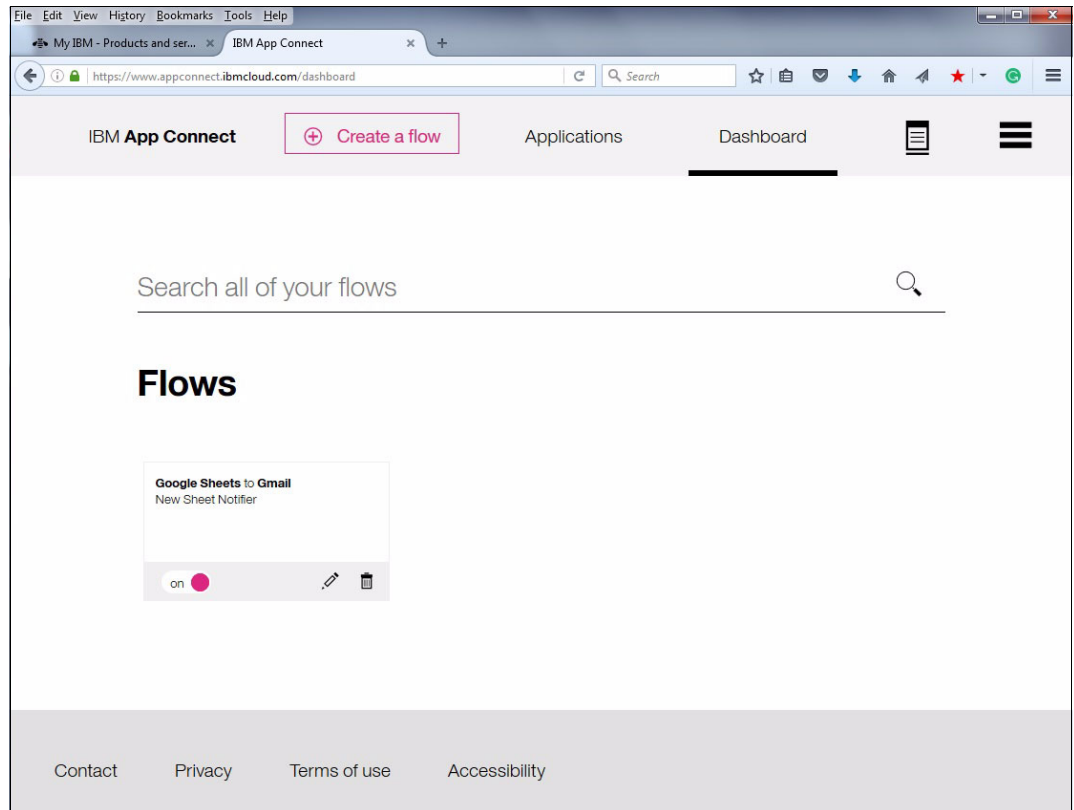


Figure 3-8 Run the flow

## Extending App Connect to include your custom applications

In most organizations, existing enterprise applications representing systems of record contain data that business teams need access to. Securely exposing these enterprise applications inside App Connect and empowering business users to create flows to automate their workflow is a significant benefit.

At the time of writing, the custom applications that you add must be based on an IBM Integration Bus message flow. You can find a sample application here:

<https://github.com/ot4i/iib-app-connect-trigger-pattern/blob/master/doc/runwarehouse.md>



Figure 3-9 shows the window in App Connect to add a custom application. You can select a network from the drop-down list if you have previously created one. If you have not created one, you can do so by clicking the **Connect a new Network** link.

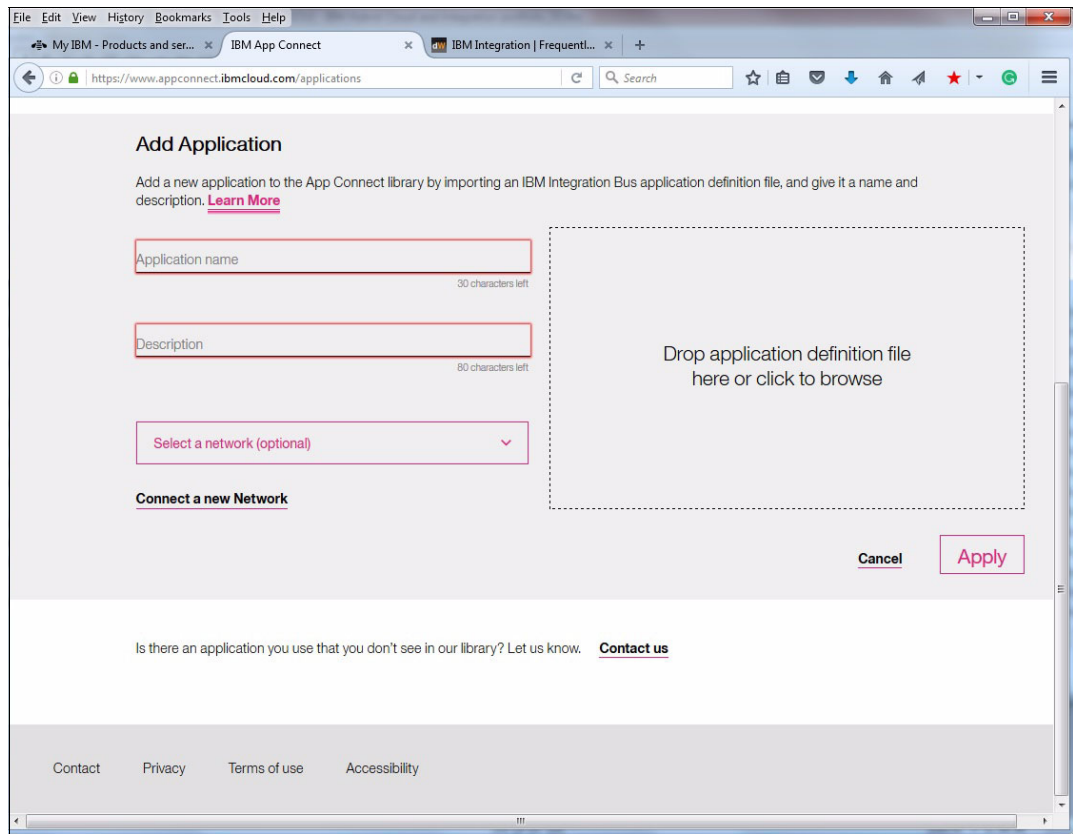


Figure 3-9 Add a custom application

App Connect uses the IBM Secure Gateway. You must download and install a Secure Gateway client on a computer on the private network. Figure 3-10 shows the window in App Connect from where you can download the Secure Gateway client and create a connection to your on-premises private network.

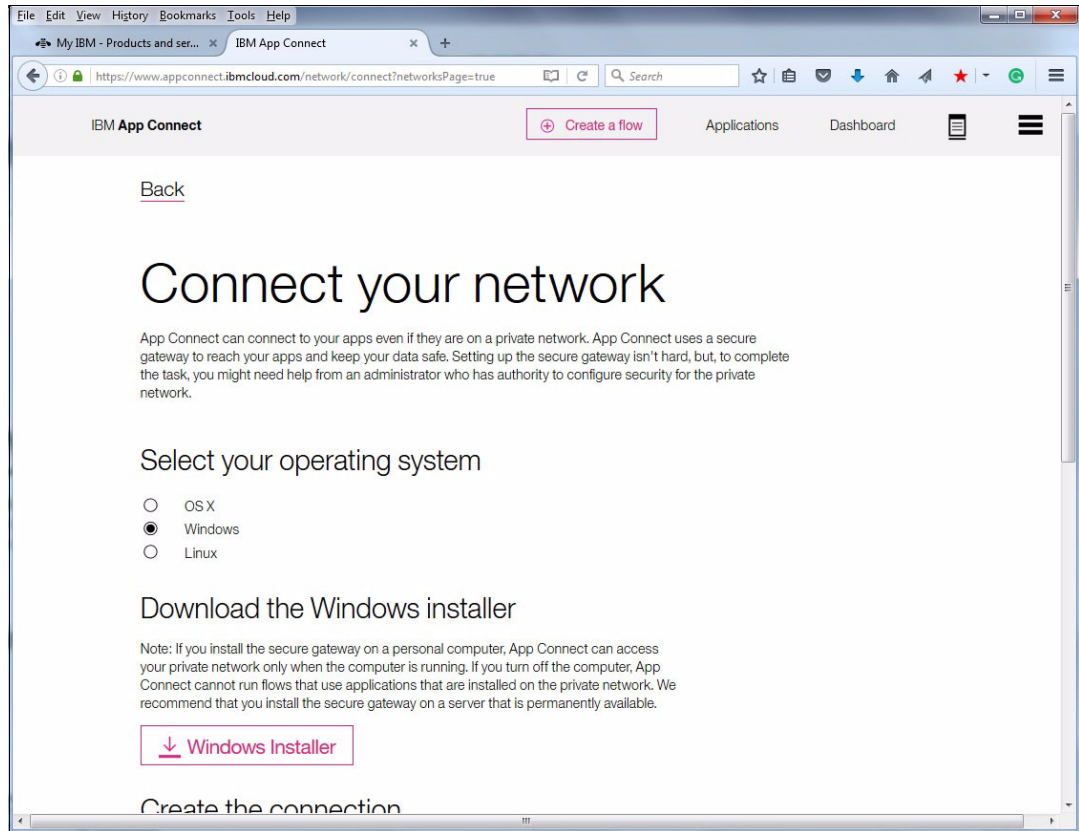


Figure 3-10 Connect to your network by using the Secure Gateway

You can also configure an access control list (ACL) to restrict what resources on your private network are accessible over this secure gateway.

## 3.3 IBM Messaging Portfolio

This section described the IBM Messaging Portfolio.

### 3.3.1 IBM MQ

IBM MQ (formerly WebSphere MQ and IBM MQSeries, where “MQ” stands for “message queuing”) is arguably, of all the products mentioned in this chapter, the one with the longest history. At its inception, it solved a great need of the IT community of the day, which was to facilitate the exchange of data between independently running applications across vastly diverse platforms, programming models, and network protocols, often across rather unreliable wide area networks.

Its universal support for platforms, networks, and programming models and its ease of use made IBM MQ an instant success at the time. Its reliability and security features were critical for it to become the de facto standard for application-to-application messaging in corporate

environments. In fact, the “once and once only” delivery of messages by IBM MQ has become legendary. Its ability to cooperate with many other transaction managers in distributed transactions based on the XA protocol for two phase commits (for example, database management systems) ensures that atomicity, consistency, isolation, durability (ACID) properties of complex transactions are maintained.

The architectural concept of IBM MQ is based on queues in which applications leave information for other applications in the form of messages that the recipient can retrieve and process at its own time and speed. This form of messaging is said to be “asynchronous” and has advantages over the more restrictive synchronous mode. IBM MQ can implement synchronous communication patterns simply by application logic. Any replies are sent as messages as well. The API functions of IBM MQ are non-blocking, so the sender of a request message is not forced by the API to wait for a response. Rather, the sender can perform any other functions before receiving a reply message. If appropriate, the sender of a request and the recipient of the reply could even be separate programs, and even on separate systems.

Queues are maintained inside IBM MQ server instances that are called *queue managers*, which can be connected across network links called channels. Typically, IBM MQ is configured so that it is not apparent to an application whether the queue that it is sending a message to is hosted on the local queue manager or on a remote one connected through a channel.

Conceptually, IBM MQ does not try to understand or interpret the message contents (conversion capabilities for character codes or numeric representation between platforms notwithstanding.) That fact means that applications that are exchanging messages must have an agreement between them on the meaning of the bitstreams exchanged. Where messages need to be interpreted, reformatted, enriched with additional information, or routed based on information in them, another type of software is required. The industry refers to this type of software as ESB. The IBM ESB tool of choice is IBM Integration Bus. For more information, see 3.2.1, “IBM Integration Bus” on page 28.

Several features were added to IBM MQ over the years, growing it into IBM Universal Messaging Backbone.

A publish/subscribe messaging model allows applications to “publish” information without knowing the recipients. This goal is achieved by using a brokering function that maintains a list of “topics” to which interested parties can subscribe. When a message is published to a topic, the broker forwards a copy of it to the designated queue of each subscriber.

The IBM MQ Clustering capability allows an IBM MQ infrastructure to easily scale beyond the capacity of individual servers. It can combine groups of queue managers into clusters, in which a logical queue can have multiple physical representations on separate systems, each being processed by a separate instance of the application. A built-in load balancing feature distributes incoming message traffic evenly across the available application instances. If one system fails, traffic is routed to the remaining instances.

These other features are included:

- ▶ Support for Java Messaging Service (JMS)
- ▶ Support for Windows Communication Foundation (WCF)
- ▶ An integrated automatic failover feature that implements high availability for queue managers
- ▶ Support for IBM MQ Telemetry Transport (MQTT)

Extra features are provided as separately licensed selectable options:

- ▶ **Advanced Message Security:** Allows messages to be signed and encrypted end-to-end between sending and receiving applications
- ▶ **Managed File Transfer:** Uses a distributed IBM MQ infrastructure to administer, perform control, and audit trail file transfers across a corporate environment

Traditionally, IBM MQ has been available on practically all relevant operating systems. In addition to that, IBM MQ is now also available as an appliance, providing the software on a dedicated and locked-up hardware platform. This new implementation allows much easier administration, for instance on remotely deployed systems, and new, powerful implementations of high availability and disaster recovery scenarios. IBM MQ Appliances can be freely integrated in existing server-based IBM MQ networks or can be used as stand-alone units.

To facilitate the use of IBM MQ in cloud environments, a number of supporting components have been introduced. Some of them are discussed in more detail in other sections of this chapter, while others are discussed here.

IBM has two main paradigms for messaging in the cloud:

- ▶ Messaging in enterprise IT, based on IBM MQ
- ▶ Messaging in the cloud native digital IT, which is implemented in IBM Message Hub, based on Apache Kafka and discussed in detail in 3.3.2, “IBM Message Hub” on page 43

As a common factor, both products support the IBM MQ Light API, which was created to facilitate the use of IBM MQ based messaging in cloud native applications. Thus, the data resources held in enterprise IT can be accessed as easily as the information exchanged with other cloud native apps through the Message Hub. Other components that facilitate the exchange of information across the realms of a hybrid cloud architecture are IBM Message Connect and Secure Gateway. The latter provides an easy way to gain secure access to enterprise IT resources for knowledge workers and mobile developers by using a self-service process. Message Connect facilitates the exchange of information between the two messaging environments, for example to allow event information flow from the enterprise realm to the Message Hub.

**Important:** Message Connect was an experimental service at the time of writing this book and has now been withdrawn. The intention is to roll this functionality into Message Hub itself in the near future. This is described in more detail along with an interim option in the following Bluemix blog post:

<https://www.ibm.com/blogs/bluemix/2016/12/bridging-mq-message-hub/>

Keep an eye on <https://developer.ibm.com/messaging/message-hub> for updates on the changes.

The core of IBM MQ has not been changed to accommodate cloud deployments. Many users are running IBM MQ in cloud hosted virtual machines, which is essentially no different from running IBM MQ anywhere else. However, because the cloud can bring many advantages over a conventional deployment, several options can be used to deploy IBM MQ in a more cloud-centric way. Use the hypervisor edition, where IBM MQ is preinstalled in a Red Hat Enterprise Linux virtual machine, which provides a standardized runtime environment for IBM MQ that can easily be replicated. Another option is the use of patterns that include IBM MQ in a PureApp environment. This configuration allows the rapid deployment of systems that, even though they are still standardized by the pattern logic, can be customized in certain useful ways. For example, you can automatically scale the number of application server instances

based on real-time demand. High Availability or Disaster Recovery configurations are easily built using PureApp pattern technology.

Another way to manage deployments is provided by container solutions. Notably, IBM MQ is being deployed in Docker containers to great effect. One of the most important advantages of Docker containers is that, compared to virtual machines, they require much less overhead for the platform part of the service. However, they still effectively isolate the hosted system from its surroundings. In a Docker deployment, you can run many software components in their own containers, but under a single operating system instance. This configuration requires a lot less computer power than having to run a separate OS footprint for each component.

Thus, IBM MQ continues to be an essential component of most IT architectures.

### 3.3.2 IBM Message Hub

IBM Message Hub on Bluemix is a cloud-based scalable and high throughput message bus that provides the capability of uniting on-premises and off-premises cloud technologies. Diverse services can be wired together through the Message Hub service by using open wired protocols, which allows you to use a wide range of languages and technologies.

Message Hub is built on top of the Apache Kafka messaging engine. Therefore, it inherits the community-proven scalability and performance as well as the durable real-time messaging capability from the engine.

This section describes some feature highlights provided by the IBM Message Hub service.

#### Integration capability

Integration is one of the core capabilities of Message Hub. It provides multiple interfaces through which messages can be produced and consumed. Message Hub service supports these APIs:

- ▶ REST API
- ▶ Kafka native API
- ▶ IBM MQ Light API

Figure 3-11 illustrates the connectivity options with Message Hub service.

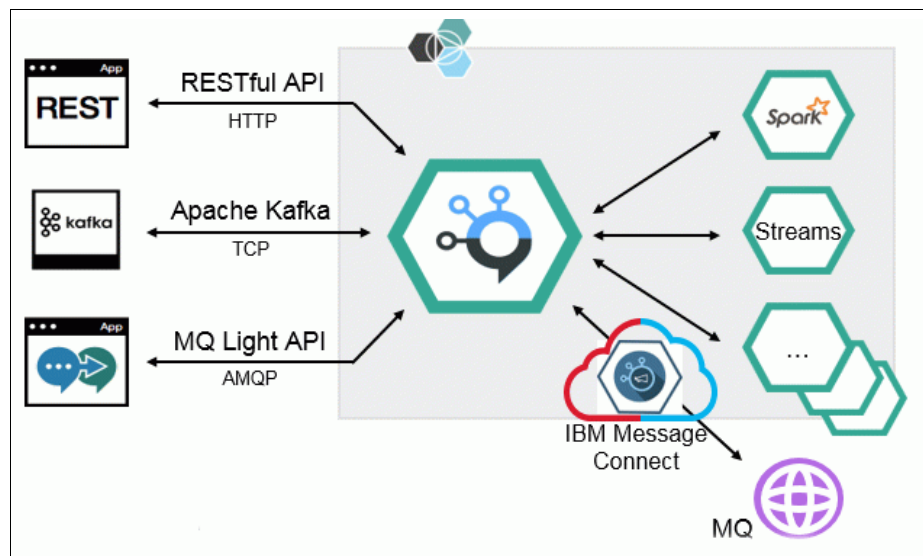


Figure 3-11 Connectivity options with Message Hub

IBM MQ Light provides a simple yet powerful Advanced Message Queuing Protocol (AMQP)-based messaging API for Java, Node.js, Python, and Ruby. You can use IBM MQ Light to quickly create scalable and responsive applications. It can integrate well with various other services on Bluemix.

The IBM MQ Light API provides a higher level of abstraction than the Kafka API. IBM MQ Light enables apps to be written quickly and with great portability in a unified messaging model that supports both point-to-point and publish/subscribe messaging patterns. Apps exchange messages using dynamically created destinations, which you can hierarchically structure (for example `/sports/football`), group by using wildcards (for example `/sports/#`), and have simple controls for delivery assurance and message expiry. This feature enables you to implement scenarios such as worker offload, event notification, and batch processing straightforwardly.

As well as sending messages between other apps using the IBM MQ Light API, you can also exchange messages with apps that use the Kafka REST or Kafka APIs. Alternatively, you can also use the IBM MQ Light API directly with an on-premises IBM MQ queue manager and thus communicate with applications by using the IBM MQ API.

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

### ***Message Hub use cases***

The following are some of the common use cases for using the Message Hub service:

1. Hub for asynchronously connecting services inside Bluemix or beyond:
  - Applications connected to events happening in other Bluemix services, or from beyond the cloud
  - Connects with enterprise IBM MQ on-premises, providing all the benefits of working in a hybrid environment
2. Microservices allow applications to evolve rapidly:
  - Open protocols support several runtimes
  - Remove the interdependency between microservices
  - Work in a range of languages that suit you
  - Deploy and scale microservices independently
3. Insights from the data you already have:
  - Data needs to be streamed from anywhere to one or many analytics engines
  - React to changing trends as they happen
  - Acts as a buffer between your data and the analytics engine
  - Run real time and batch analytics on the same data

These use cases are illustrated in Figure 3-12.

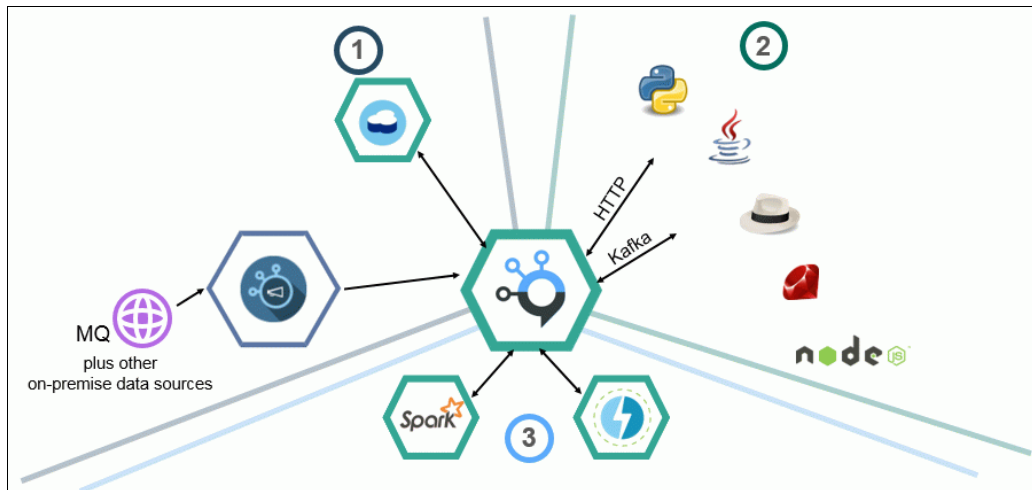


Figure 3-12 Message Hub use cases

Sinatra is an open source software web application library and domain-specific language written in Ruby that is available for no fee and helps in writing applications in Microservices style. It is available at:

<http://www.sinatrarb.com/about.html>

### 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 the same failure domain when a failure occurs, which means that a failure only affects 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 high availability of its service.

Figure 3-13 illustrates the architectural overview of Apache Kafka.

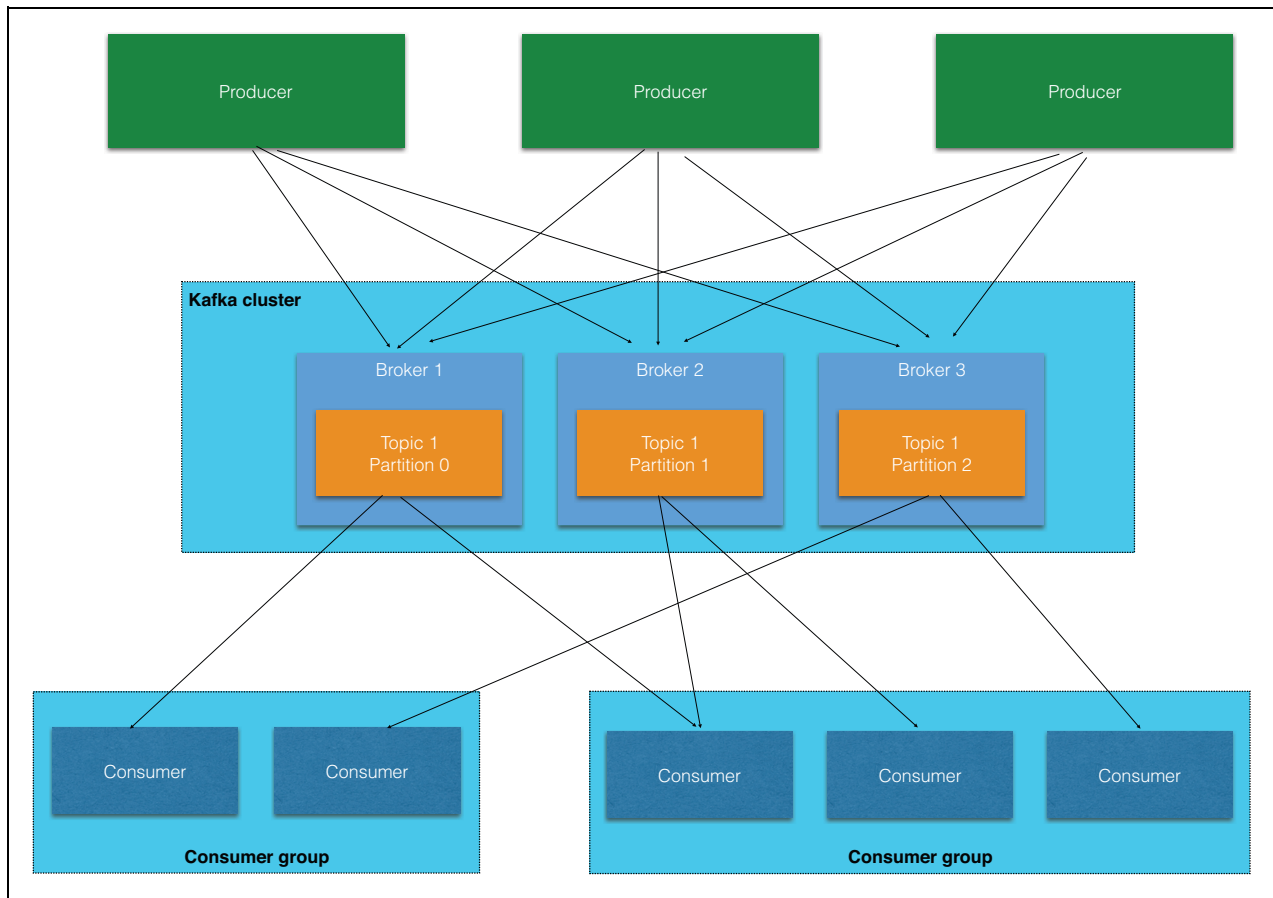


Figure 3-13 Kafka architecture

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

### 3.3.3 IBM Watson Internet of Things Platform

It is undeniable that the Internet of Things (IoT) is becoming mainstream as devices are becoming intelligent, instrumented, and interconnected. Billions of internet-connected devices are adding a tremendous amount of data to global data traffic and are being used across various industries. Businesses can unlock opportunities and gain insights from this data. It becomes clear that managing millions of devices and analyzing the data generated from these devices is not an easy task. It demands a highly scalable, fault tolerant, and robust



platform to manage these IoT devices. IBM Watson™ Internet of Things Foundation (IoT Foundation) platform is such a platform.

The IBM Watson IoT™ platform (Figure 3-14) is a cloud-based service for managing and connecting IoT devices, and composing and extending applications that use data and analytics from connected devices, sensors, and gateways. It is available as a catalog item or service from the IBM Bluemix Platform or through the IBM Watson Internet of Things portal. The IoT portal is available at:

<https://internetofthings.ibmcloud.com>

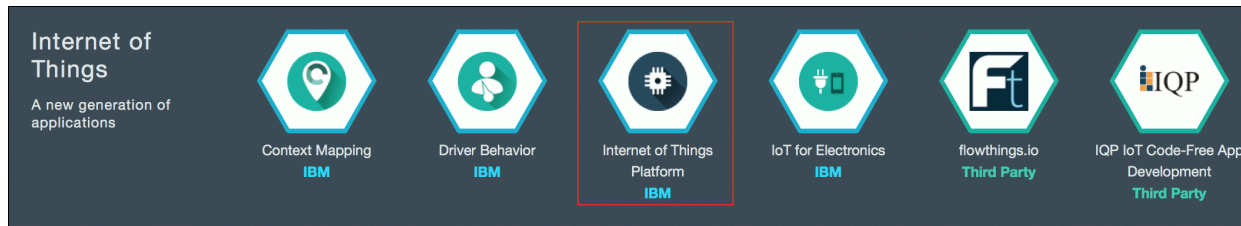


Figure 3-14 Watson Internet of Things service from Bluemix

Four key areas of the IBM Watson IoT Platform can be distinguished:

- ▶ **Connect:** It is concerned with connecting and managing devices and integrations with third-party services fall under this category. It provides a powerful web dashboard to add and manage devices, control access to the IoT service, and monitor usage and other key parameters at a glance. Functions also include device management actions like rebooting or updating firmware, receive device diagnostics and metadata, and perform bulk device addition and removal.
- ▶ **Information Management:** It is used for data storage and transformation. IoT service provides access to real-time data from sensors, and devices as well as options to store data to different types of Bluemix hosted databases. This way, users can have access to real-time and historical data.
- ▶ **Analytics:** This function is useful for visualizing real-time device data by using Dashboard cards.
- ▶ **Risk Management:** This area is concerned with security, authentication, and prevention of fraud by providing a powerful authentication mechanism, and connectivity over Transport Layer Security (TLS).

Figure 3-15 illustrates the architecture of the IoT platform and how applications, devices, and gateways connect to the platform.

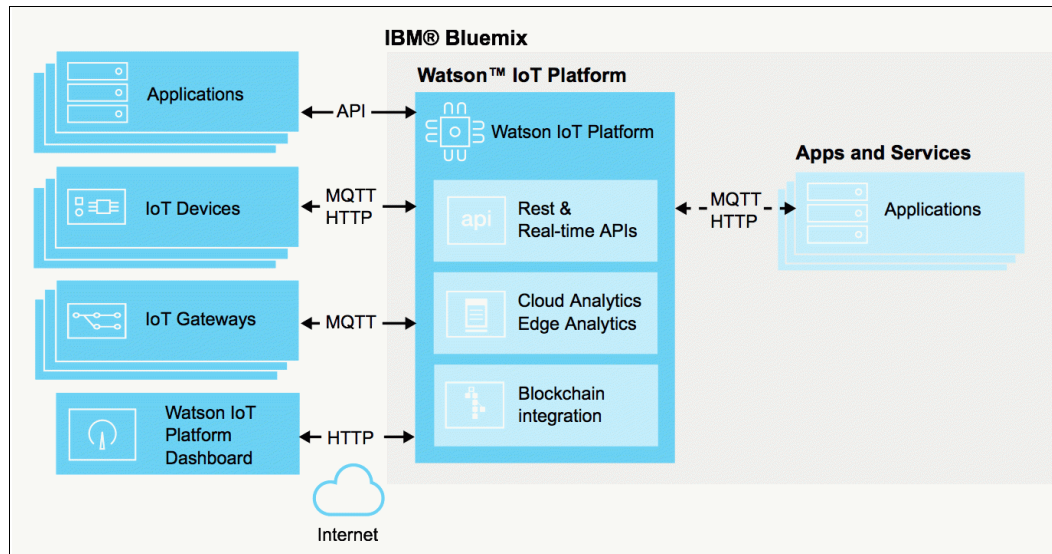


Figure 3-15 IBM Watson IoT architecture

The IoT service is based on the industry-standard MQTT protocol (OASIS ratified) to connect devices and applications. Support for MQTT with TLS is available to receive data from and send commands to devices securely. It is important to be familiar with the following key terminologies for the IoT service:

- ▶ **Organization:** An organization groups a number of devices and applications for security reasons. Each organization has a 6-character unique ID. Devices and applications that are registered within an organization get unique device and API keys that makes sure data from devices and applications are only sourced from registered entities. No device and application is allowed to connect to IoTF without the appropriate keys.
- ▶ **Devices:** Anything that has a connection to the internet and has data to publish to the cloud is defined as a device. Devices are not allowed to communicate with each other directly. Devices can accept commands from applications. They identify themselves to Watson IoT by unique authentication keys.
- ▶ **Applications:** Anything that has a connection to the internet and can interact with data from devices and control the behavior of those devices in some manner. An application identifies itself to Watson IoT by a unique API key and application ID.
- ▶ **Gateway devices:** These are a special type of device that possesses the combined capabilities of a device and an application. Devices that cannot connect directly to IoTF can use gateway devices as access points to connect to the service. These devices must be registered in IoTF first before they make any connection. Gateway devices can register new devices, and can send and receive data on behalf of devices that are connected to them.
- ▶ **Events:** An event is a mechanism by which a device publishes data to IoTF.
- ▶ **Commands:** A command is a mechanism by which an application communicates with a device.

Devices can be managed or unmanaged. Managed devices are those that contain a management agent. A management agent allows the device to interact with the IoT Device Management service by using the Device Management protocol. Managed devices can

perform device management operations including location updates, firmware download and updates, and reboot and factory reset.

Unmanaged devices do not contain a management agent. They can connect to IoT and send and receive data. They cannot perform device management operations.

Quickstart mode allows a physical device or IoT simulator to connect to IBM IoT without having to sign up for the service, register, and add the device, as opposed to managed mode. It is available in the open sandbox to quickly and easily connect any device that can run an MQTT client to Watson IoT. Quickstart is available at:

<https://quickstart.internetofthings.ibmcloud.com>

A unique device ID must be entered in the Text box as shown in Figure 3-16. The device data is then displayed in the window.

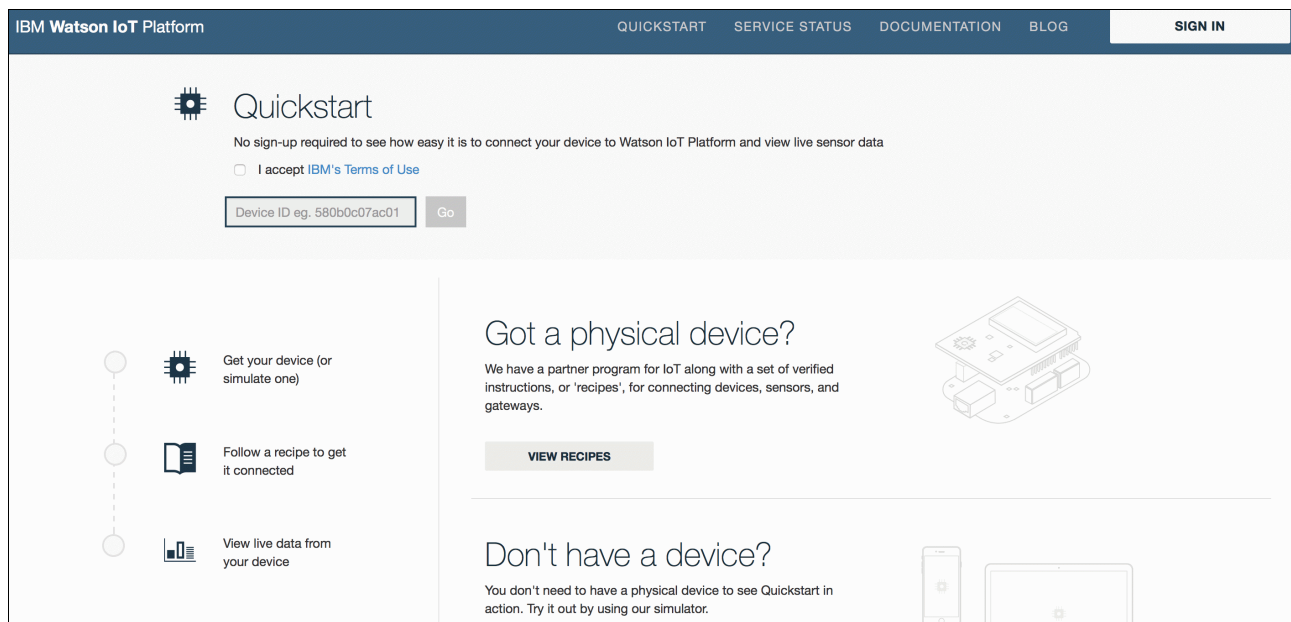


Figure 3-16 Internet of Things Quickstart mode

In summary, Watson Internet of Things is a platform for device connection, management, and integration with third party services, applications, analytics, and security.

### 3.4 IBM DataPower

At its inception, the first DataPower product was designed to help accelerate XML processing, which had turned out to be excessively CPU intensive. Optimized algorithms on specially tuned hardware were the strategic direction. However, it soon became apparent that the high-speed XML processing and the appliance approach to providing the function suggested a different usage pattern: The XML Security Appliance.

Not only was XML an expensive format to process, it also soon became the universal language for data exchange on the internet, meaning that data was being exchanged between organizations much more frequently than ever before. This situation immediately brought on numerous security concerns that needed to be addressed, and needed to be addressed in a consistent and comprehensive fashion by a central instance. It had become clear by then that firewall technology could address many but not all challenges. The most

notable “loophole” was the fact that to communicate through the internet at all, at least the HTTP(S) ports needed to be open. But there was no scrutiny of the new traffic. Typically, the servers that were running the enterprise applications were expected to provide their own security, custom configured to their needs. But too often neither server capacity nor administrator or developer skills were up to the task. And if server-based security was configured, it turned out to require a lot of resources and was maintenance intensive.

However, a DataPower appliance was ideally suited to the task due to these characteristics:

- ▶ XML-oriented processing power
- ▶ Ample network capacity
- ▶ Minimal and purpose-built operating platform that did not allow installation of third-party software of any kind
- ▶ Hardened physical implementation

A single appliance would easily support the traffic for many servers, thus relieving much more costly hardware from the security-related processing and often from expensive parsing of complex XML messages.

Even today, several generations of hardware and software later, most DataPower appliances are found in a DMZ, with one set of network ports outbound to the internet, another to the trusted zone inside the corporate firewall, and a third one isolated just for use by administrators. All exposed services or other communication endpoints are implemented on the outbound ports. Incoming requests are scrutinized by the defined and configured security policies, messages are decrypted, signatures and credentials are verified, and messages are transformed into other formats where required and routed to the internal endpoints of the serving applications. Responses are passed through matching processing sequences and handed back to the external requesters.

In addition to generic XML messages, SOAP messages addressing WSDL-based web services can be handled. In addition to HTTP(S), a range of FTP-style interfaces are supported by some models and IBM MQ and JMS traffic. There is also a feature that specializes in business-to-business traffic, supporting AS1/AS2/AS3 protocols.

Current versions of the appliance also support REST and JSON, and offer a scripting facility now called GatewayScript that is derived from JavaScript.

DataPower appliances are typically partitioned into multiple domains. These domains are usually isolated from each other in terms of the configured assets, such as policies, but need to coordinate the use of unique resources, such as port numbers.

A number of interfaces are available that allow developers and administrators to interact with a DataPower appliance, such as a command-line interface (through the built-in serial port or an SSH connection) and an XML interface that can be used to send configuration messages from external sources, such as software tools. But probably the most popular interface is a built-in interactive web GUI.

Some use cases for the XML interface are the ability of IBM Integration Bus and API Connect to remotely configure a DataPower appliance. There is also an Eclipse-based external administration and development toolkit that allows the joint management and configuration of multiple appliances from one location. The recently released DataPower Operations Dashboard also uses this interface.

In addition, API Connect offers two API Gateway implementations:

- ▶ A micro gateway implemented in Node.js
- ▶ An enterprise grade gateway implemented by DataPower appliances

These implementations are configured transparently to the user by the API Management console.

Originally, DataPower was only available as a physical appliance. With the growing popularity of cloud deployments, a virtual edition is now offered. However, a physical appliance still provides the best throughput. For some production and most test and development use, including training, the virtual DataPower is quite adequate. It can be deployed on most virtualization platforms, including Docker containers.

IBM DataPower remains the strategic integration gateway for IBM in the age of the cloud.





## Part 2

# Hybrid integration scenarios

This part shows how to implement several hybrid integration scenarios to show you the powerful integration capabilities of the IBM hybrid integration portfolio discussed in Part 1, “Concepts and architecture” on page 1.

This part includes the following chapters:

- ▶ Introduction to the scenarios
- ▶ Exposing APIs externally
- ▶ Automation for business users
- ▶ Kick-start digital teams







## Introduction to the scenarios

This chapter bridges the gap between the generic concepts discussed in chapters 1-3 and a practical application example. The text introduces a fictional company and three hybrid integration scenarios.

This chapter has the following sections:

- ▶ Introducing CompanyA
- ▶ Architecture overview
- ▶ Re-creating the scenarios

## 4.1 Introducing CompanyA

This section introduces CompanyA.

### 4.1.1 The business

CompanyA is a company from the retail sector that sells and distributes electronic goods. The company's sales are predominantly made through a network of partners that have both online and physical stores.

Digital transformation is a global process that affects all kinds of industries and enterprises of all sizes. Apart from being left behind while other companies (including competitors) embrace digital transformation, CompanyA realized that this change also represents an opportunity namely around the following aspects:

- ▶ Increase revenue and outreach by using the API Economy.
- ▶ Increase efficiency and productivity by using automation.
- ▶ Disrupt and innovate using agile approaches and modern application architectures.

CompanyA identified a number of functional capabilities and business scenarios with which to capitalize on those opportunities. Figure 4-1 provides an overview of the functional capabilities that are required to enable the digital transformation.

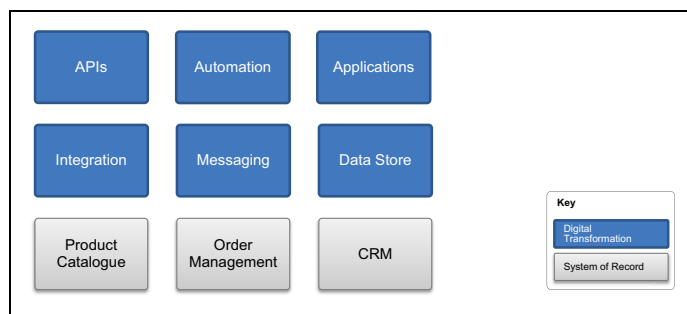


Figure 4-1 Functional capabilities

Figure 4-1 involves these concepts:

<b>APIs</b>	Making enterprise assets accessible either externally or for internal reuse.
<b>Automation</b>	Increased productivity through automation at the level of non-technical users.
<b>Applications</b>	Applications are a way for CompanyA to accomplish digital transformation and realize its benefits. Better applications constructed in a more focused way enable CompanyA to keep up with industry and customer changes.
<b>Data Store</b>	A facility that provides applications with data independently from access to the systems of record. This facility supports functional changes and non-functional aspects, such as performance.

<b>Integration</b>	The glue between the systems of engagement and the systems of record, which keeps the company running. In the context of digital transformation, this concept represents the access point to functions that can be exposed from the systems of record.
<b>Messaging</b>	Messaging is an enabling capability for APIs, automation, and applications.

## 4.1.2 Hybrid integration scenarios

The scenarios that the company identified relate 1:1 to the use cases described in Chapter 2, “Hybrid integration use cases” on page 11. Putting them in the context of CompanyA, a number of different departments feature in the hybrid integration scenarios, and are introduced here briefly:

<b>Integration Team</b>	This is the team of skilled IT Specialists with subject matter experience in system integration who support the on-premises infrastructure of CompanyA.
<b>Sales Team</b>	The business partner sales team is organized in regions, and each member looks after a set of partners and their orders. The members of this team do not have a technical background, but use technology and various systems every day as part of their job.
<b>Digital Team</b>	Originally, this team started with only two graduates who were given a short-term project to create a web page for the BP Sales team. Now it has become a group of permanent employees who are funded by the BP Sales Team. The team uses the agile methodology and modern, mobile-friendly programming languages, frameworks, and tools to do their work.
<b>Marketing Team</b>	The marketing team co-ordinates with the sales team and manages marketing campaigns with the product manufacturers. Like the Sales Team, the people in this team do not have a technical background, but are avid technology users who work with cloud-based systems and occasionally working remotely.

In line with the use case described in 2.2, “Use case A: Joining the API economy” on page 12, the first scenario is about joining the API Economy.

### Exposing APIs externally

CompanyA has decided to make some of the information and functions in the systems of record available to external parties. Specifically, the company decided to start with product and order information. These items are exposed as APIs that allow external developers to accomplish these tasks:

- ▶ Retrieve information about the products sold by CompanyA. This data includes product names, descriptions, photos, and other details.
- ▶ Manage purchase orders all the way from creation over change to cancellation. Purchase orders contain information about the company that placed the order (i.e. the partner), the products sold, and the total order amount.

Exposing these functions through APIs, CompanyA aims to provide more up-to-date information that is easier to consume and integrate for its partners, ultimately leading to more revenue. The scenario is split into two main parts:

- ▶ Exposing an existing SOAP API externally as REST
- ▶ Creating an exposing API by integrating information from multiple systems

The main actor in all parts of the scenario is the Integration Team.

Monetization is another important aspect of the use case, as is the distinction of system and interaction APIs. Although both are important considerations, this book goes into the most detail about the API exposure.

### Automation for business users

The second scenario that CompanyA has chosen in its digital transformation initiative is about empowering its business users with automation, in the first instance specifically the marketing team.

One of the team’s activities is looking after campaigns. Originally, campaigns were managed manually with a spreadsheet. Nowadays, the team works with Salesforce CRM, which is also used by the Sales team for customer relationship management.

The scenario explores how the Marketing team can use automation to become more productive and spend less time on routine tasks. Automation is used for three parts of the scenario:

- ▶ Keep the campaign details in the spreadsheet up to date by synchronizing it with campaign information from Salesforce CRM.
- ▶ Triggering a follow-up action in Salesforce CRM for any orders that were canceled.
- ▶ Tracking the success of campaigns by correlating the campaign information with order volume data.

Although the first part of the scenario can be done by the Marketing team themselves, the other parts are supported by the Integration team.

This scenario relates directly to the hybrid integration use case described in 2.3, “Use case B: Improving productivity” on page 17. It specifically addresses the business user integration scope of the use case, which is shown in Figure 4-2.

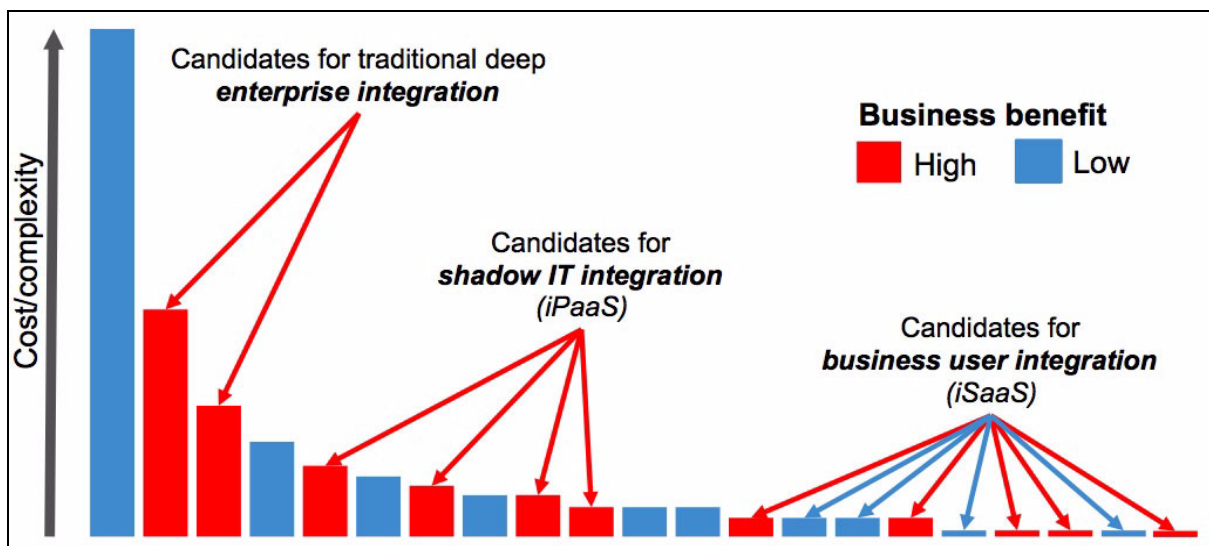


Figure 4-2 Long tail of integration candidates

The starting point of the scenario is therefore a cloud-to-cloud integration flow from Salesforce CRM to Google Sheets. The integration flows in this area are in general low in complexity. They are created by users who, although they are comfortable with using technology, do not have an integration background. While keeping the flows simple, the

scenario is improved by triggers from on-premises systems (events from the order management system) showing the ground-to-cloud integration aspect of the use case.

## Kick-starting digital teams

Last but not least, the third scenario addresses the scope described in 2.4, “Use case C: Refactoring for innovation” on page 21. The main driver behind this innovation is microservices, and therefore these are also at the heart of the scenario.

CompanyA has decided to use microservices to create an application to be used by its sales team. The aim is to make it easier for the team to stay up-to-date with changes to the orders placed by its partners. They need to get an overview quickly about order status, changes, and other updates so that they can react quickly to problems or even proactively avoid them. Initially the updates will only come from the order management system. However, it is easy to imagine how shipment events or even external information like weather data can be added for an even richer solution.

As such, the scenario has two main aspects:

- ▶ **Creating the stream of order events**

In this part, the raw events from the order management system are enriched to make sure that they are meaningful by themselves. They are then republished as business events.

There are several technical implementation options for sending those events and making them available to the microservices, all of which are discussed as part of this scenario.

- ▶ **Implementing the microservices and exposing them for the business application**

In this part, the actual microservices are constructed that will drive the business application. The sales team user might want to use filters and views on the data that are addressed in this chapter.

The first aspect, which is related to the events and communication, is usually handled by the Integration Team. However, after the event stream is established and it comes to microservices and their local data, the digital team can continue to work independently.

This scenario also shows the use of a data store, which enables the microservices to be independent from the actual system of record in the backend. In this context, it is important to understand that these data stores are not replicas or a replacement of the system of record. Rather, they contribute towards the three goals for composable applications:

<b>Agility</b>	The developers can make changes to the data store without affecting anybody else. It contains only the data necessary for that particular microservice or set of microservices, and only for a limited lifespan.
<b>Scalability</b>	The data store and microservices instances can be (cloud-)scaled to support the non-functional requirements of the application.
<b>Resilience</b>	A dedicated data store helps make the microservices independent, and therefore resilient to failures or outages of other components.

Finally, the scenario also includes an actual business application that uses the exposed microservices. Although it is included in the scenario, the design and development of the front-end application are outside the scope of this book.

## 4.2 Architecture overview

Figure 4-3 shows a high-level solution architecture for CompanyA. Apart from the components related to hybrid integration (highlighted in blue), there are a number of systems of record, business applications, and external parties. Each of the systems is explained in more detail below.

Another important aspect of the diagram is the notion of hybrid integration with components that are distributed in different environments. Although the diagram shows APIC in the Bluemix layer, this does not mean that it can only run on Bluemix. In fact, a number of deployment options ranging from on-premises to cloud-dedicated or cloud-shared are available for many of the components. This section briefly describes the deployment options in general, and explains the concrete deployment decisions taken for the implementation of the scenarios in their respective chapters.

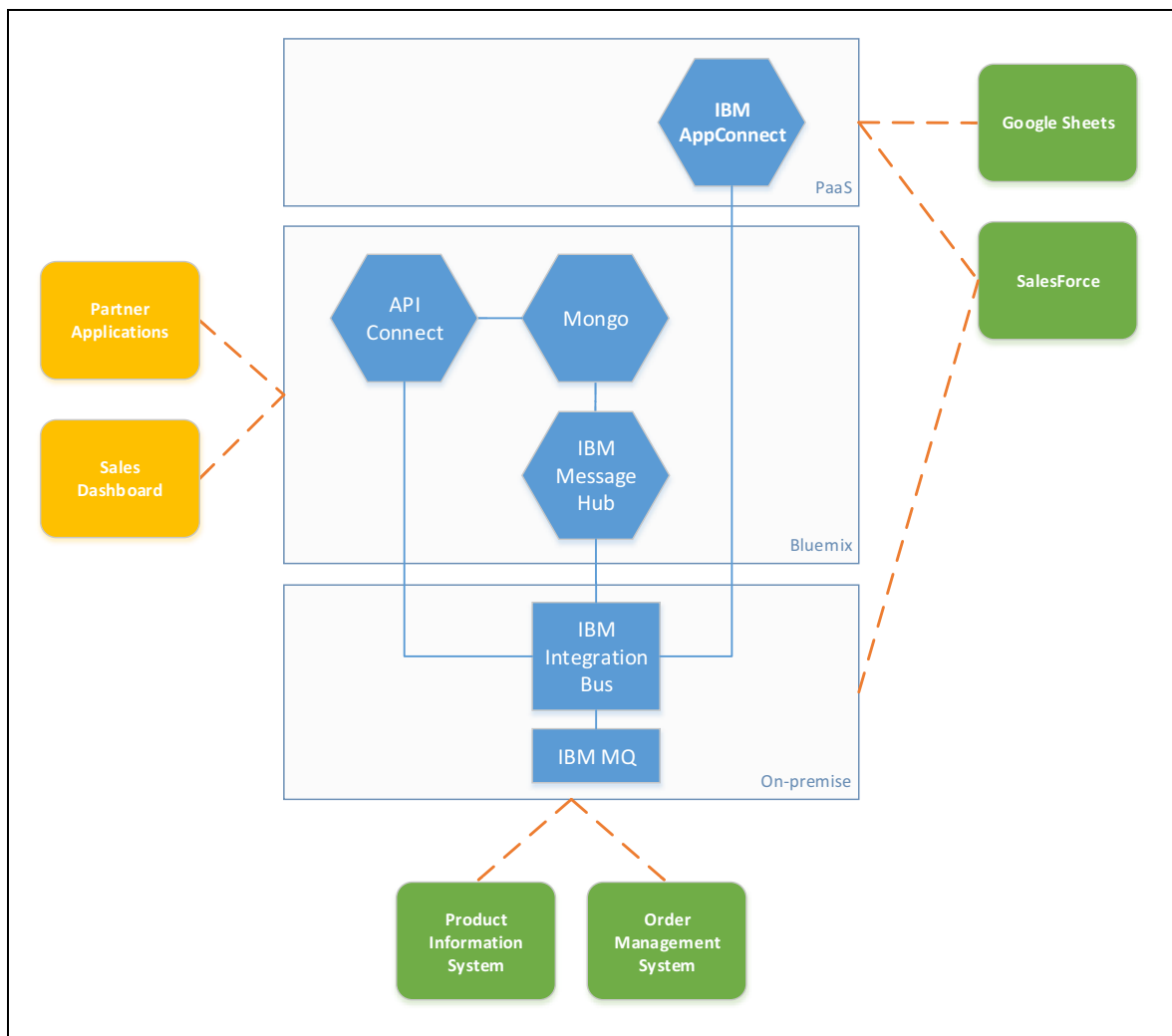


Figure 4-3 High-level architecture overview

## 4.2.1 Components

Figure 4-3 on page 60 provides a high-level overview of the solution components in scope for the digital transformation initiative for Company A. The components can be internal or external to Company A, and are a mix of on-premises and cloud-based systems. The following list provides a brief introduction for each component:

<b>Partner Applications</b>	These are the consumers of the APIs exposed externally as part of scenario 1.
<b>Sales dashboard</b>	The dashboard is an internal business application used by CompanyA to display and manage order updates. Scenario 3 explains how the dashboard is built on top of a microservices architecture. The dashboard application has multiple deployment options, but these are outside the scope of this book.
<b>Google Sheets</b>	<i>Google Sheets</i> is a business application that is used by CompanyA's employees internally. As a software as a service (SaaS) application, it is outside of the company's sphere of influence.
<b>SalesForce CRM</b>	SalesForce CRM is an SaaS application that is considered a system of record for CompanyA. It is used by the sales team to manage the details of the company's partners. It is also used by the marketing team to manage marketing campaigns. In contrast to the other systems of record, it is not located and run on-premises, showcasing a typical example for the need of hybrid integration.
<b>Product Information System</b>	A typical on-premises system of record that in this scenario has existed in CompanyA since the beginning. It manages the information about the products that CompanyA sells and distributes, and exposes this information through SOAP web services.
<b>Order Management System</b>	In technical terms, this is an on-premises database. It is used to store all information about purchase orders in CompanyA. The Order Management System is another typical example of an on-premises system of record.
<b>IBM App Connect</b>	IBM App Connect is one of the components from the hybrid integration portfolio to provide integration capabilities for business users. In this scenario, it is used by the marketing team.
<b>IBM API Connect</b>	IBM API Connect is all about APIs. In these scenarios, the component is used to expose APIs to external partners in scenario 1 and to internal applications in scenario 3. It also includes a runtime environment for microservices.
<b>MongoDB</b>	MongoDB is the noSQL data store supporting the microservices.
<b>IBM Message Hub</b>	IBM Message Hub is a cloud-based messaging service. It is a supporting component for the microservices for which it provides the event stream (in this scenario of order events).

## IBM Integration Bus

This is the integration backbone for CompanyA. It provides deep integration and connectivity capabilities that are used to implement various integration techniques and patterns ranging from covering point-to-point use cases like *request/response* or *fire and forget*, and *publish/subscribe* APIs.

## IBM MQ

The main messaging backbone for Company A. It facilitates messaging support point-to-point integrations and providing a publish/subscribe platform for events.

## 4.3 Re-creating the scenarios

The content of the following chapters was designed so that you can easily reproduce it. Towards that goal, the scenarios have these characteristics:

1. Make a number of simplifications in the solution for each chapter.

A realistic (or at least more real-world-like) solution outline is still shown at the beginning of each scenario to help you transfer the example to your own environment. In addition, the simplifications themselves are documented.

2. Provide a set of resources to help you with routine tasks.

These resources are stored in a GitHub repository, and range from stubs to code snippets. They are there to help you to focus on the interesting aspects of the solution rather than having to worry about, for example, product installations.

### 4.3.1 Prerequisites

To re-create the scenarios and work with the provided resources and images, you need to install the following software packages on your system:

- ▶ Docker

Installation instructions can be found here:

<https://docs.docker.com/engine/installation/>

- ▶ Docker Compose

Installation instructions can be found here:

<https://docs.docker.com/compose/install/>

- ▶ Git

Installation instructions can be found here:

<https://git-scm.com/>

- ▶ IBM Integration Bus Toolkit v10

Download and installation instructions can be found here:

<https://developer.ibm.com/integration/docs/ibm-integration-bus/get-started/get-started-with-ibm-integration-bus-for-developers/>

- ▶ IBM MQ Explorer v9

Download and installation instructions can be found here:

<http://www.ibm.com/support/docview.wss?uid=swg24021041>



## 4.3.2 The GitHub repositories

The accompanying GitHub repositories can be found under <http://github.com/sg248351> as shown in Figure 4-4.

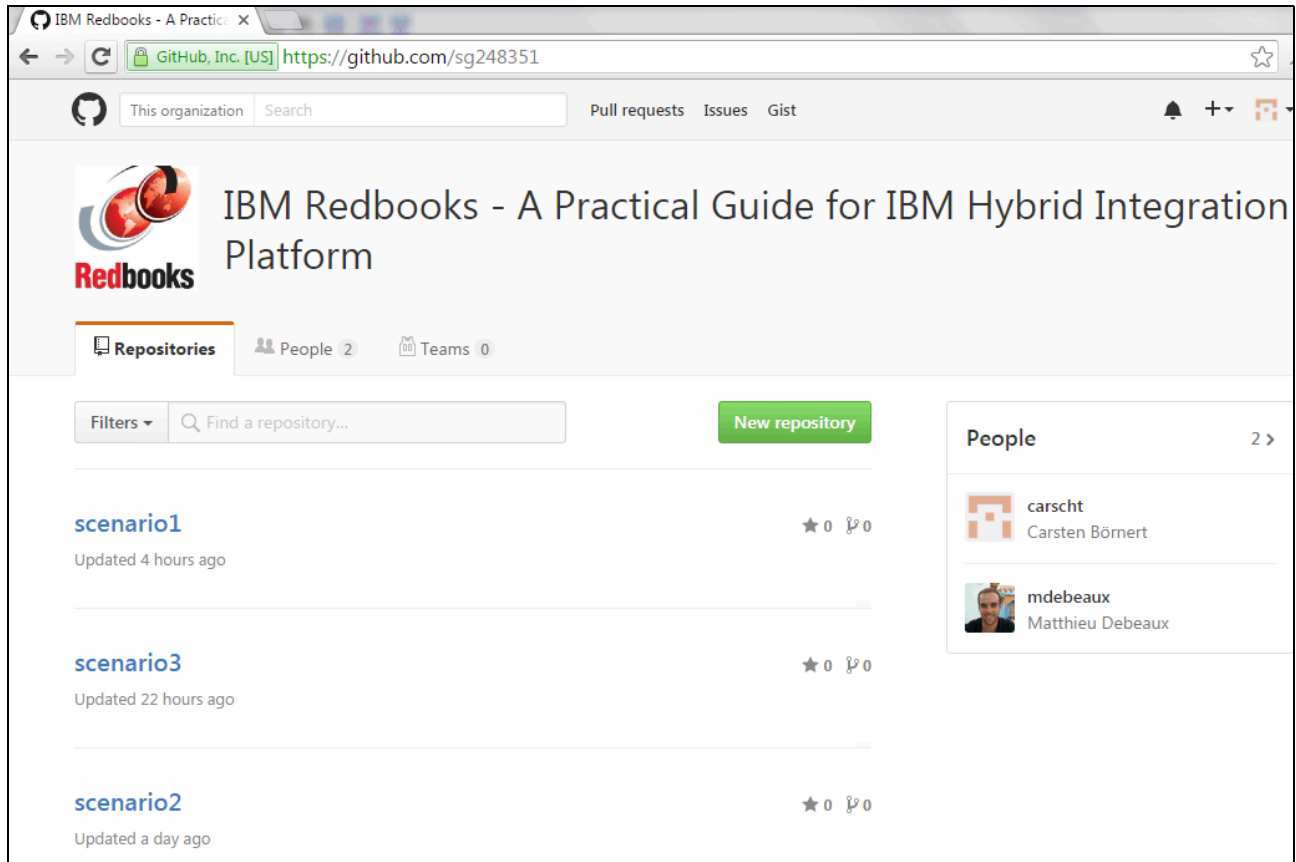


Figure 4-4 GitHub repositories for scenarios

There is a GitHub repository for each scenario. All of them have the same basic structure as shown in the example in Figure 4-5.

1_coding	initial set of files	2 days ago
2_testing	initial set of files	2 days ago
99_docker/sgc	restructured for multiple docker builds	14 minutes ago
LICENSE	initial set of files	2 days ago
README.md	Update README.md	2 days ago
docker-compose.yml	restructured for multiple docker builds	14 minutes ago

Figure 4-5 Sample content of scenario repository

The following elements are shown in Figure 4-5:

- LICENSE** File containing licensing information for the files in this repository.
- README.md** Documentation for the repository. This file also provides information about the licensing of IBM products that are used as part of the scenario, such as a link to the licensing information about IBM Integration Bus for Developers.

<b>docker-compose.yml</b>	Definition file that is used with the docker-compose command to set up all the containers necessary for the environment in this scenario. 4.3.3, “Environment configuration for the scenarios” provides more information about how to work with this file.
<b>1_coding</b>	This directory contains resources to help you with the coding or configuration aspects as part of the steps described in the scenario.
<b>2_testing</b>	This directory contains resources to help you perform any testing aspects of the scenario, for example test files.
<b>99_docker</b>	This directory contains resources that are used with the docker-compose.yml file.

### 4.3.3 Environment configuration for the scenarios

Because each scenario addresses a different use case, only a subset of all the components from the overall solution are required. In terms of re-creating the scenarios, this means that for each scenario, different systems and stubs are required. Use docker-compose to provide environment-specific configurations (docker-compose.yml files) that are kept in the respective scenario’s GitHub repository.

Figure 4-6 provides an example of such a configuration, showing that for this scenario, two docker containers (or services as docker-compose calls them) are created:

- ▶ One container with IBM MQ (eventqm)
- ▶ Another container that represents a stub

```

1  version: '2'
2  services:
3    stub:
4      build: 99_docker/stub/
5      image: sg248351/order-api-stub
6      depends_on:
7        - mq
8      ports:
9        - "4414"
10       - "7800"
11     environment:
12       - LICENSE=${LICENSE}
13   mq:
14     image: sg248351/eventqm
15     ports:
16       - "1414"
17     environment:
18       - LICENSE=${LICENSE}

```

Figure 4-6 Sample docker-compose.yml file

To start the environment for a specific scenario, complete the following steps. Note that all of the commands need to be run on the machine where you set up Docker. The following steps describe the process for scenario 1:

1. Create a directory to hold all the configuration data, for example sg248351, and open a shell or command prompt there.
2. Clone the GitHub repository for the respective scenario by using the following command:

```
git clone https://github.com/sg248351/scenario1
```

**Tip for Windows users:** If you are using Windows, you need to install and start Git bash on Windows because Windows does not have its own bash shell available. See the link:

<https://git-for-windows.github.io/>

All Linux operation systems have bash available, so Git bash is not needed.

3. Navigate to the `scenario1` directory by running this command:

```
cd scenario1
```

4. An environment can consist of multiple containers running multiple IBM products. To accept all license agreements, run the following command:

```
export LICENSE=accept
```

5. Depending on the scenario, you might need to configure other environment variables. You can see which ones by running any of the `docker-compose` commands as shown in Figure 4-7.

```
vmuser@ubuntu:~/dev/sg248351/scenario1$ docker-compose ps
WARNING: The SC1_GWID variable is not set. Defaulting to a blank string.
WARNING: The SC1_SECTOKEN variable is not set. Defaulting to a blank string.
Name      Command      State      Ports
-----
vmuser@ubuntu:~/dev/sg248351/scenario1$
```

Figure 4-7 Required environment variables

The scenario chapter itself includes a description of the properties and which values should be used.

6. Create and start all containers that are required for this scenario by running this command:

```
docker-compose up -d
```

**Note:** Depending on your environment, you might need to run the following commands before starting the `docker-compose` environment:

- ▶ `$ docker-machine start` to start the virtual machine for docker
- ▶ `$ eval "$(docker-machine env default)"` to set environment variables

The first time that you run this command, it will take long time to download all docker images from the internet. Subsequent scenarios reuse many of the images, so the command runs much faster. The output should look similar to what is shown in Figure 4-8.

```
vmuser@ubuntu:~/dev/sg248351/scenario1$ docker-compose up -d
WARNING: The SC1_GWID variable is not set. Defaulting to a blank string.
WARNING: The SC1_SECTOKEN variable is not set. Defaulting to a blank string.
Creating network "scenario1_sc1-net" with driver "bridge"
Creating scenario1_orderdb_1
Creating scenario1_stub_1
Creating scenario1_iib-op_1
Creating scenario1_sgc_1
vmuser@ubuntu:~/dev/sg248351/scenario1$ █
```

Figure 4-8 Starting the containers for scenario 1

- Verify that the containers have started successfully by running `docker-compose ps` as shown in Figure 4-9.

```

vmuser@ubuntu:~/dev/sg248351/scenario1$ docker-compose ps
-----
Name                Command                State                Ports
-----
scenario1_lib-op_1  itb_mq_manage.sh      Up                  0.0.0.0:32860->1414/tcp, 0.0.0.0:32859->4414/tcp, 0.0.0.0:32858->7800/tcp
scenario1_orderdb_1 /entrypoint.sh dbzstart Up                  22/tcp, 0.0.0.0:32855->50000/tcp
scenario1_sgc_1     node itb/secgwclient.js - ... Up
scenario1_stub_1   itb_manage.sh         Up                  0.0.0.0:32857->4414/tcp, 0.0.0.0:32856->7800/tcp
vmuser@ubuntu:~/dev/sg248351/scenario1$

```

Figure 4-9 Checking the state of containers with docker-compose

**Note:** The ports are unique for each deployment, so use the ones that are displayed for your machine.

There are a couple of things to note from the output of the command, namely hosts, names, state and ports:

- The first column shows the name of each of the containers in the scenario environment. When using docker commands to interact with a specific container, you use this name.
- The third column shows the state of the container.
- In the last column, you can see entries for ports like this `0.0.0.0:32860->1414/tcp`. This means that port 1414 from inside the container has been mapped to port 32860 on the docker host.

**Note:** The docker host is normally `localhost`. For Windows users who have installed the Docker Toolbox, however, it is the IP of the embedded Virtualbox Linux VM that acts as the docker host. Docker Toolbox users can determine the IP of the docker host by running the following command:

```
docker-machine default ip
```

For example, you would point IBM MQ Explorer to `localhost:32860` in order to interact with the queue manager listening on port 1414 in the container. The allocation of the ports on the docker host is dynamic and can change every time that you start the container.

- Here is a list of additional docker commands and options that are useful when re-creating the scenarios:
  - **docker-compose down:** Stops and removes the containers that are defined in a `docker-compose.yml` file.
  - **docker-compose start:** Starts all previously created docker containers in this compose environment.
  - **docker exec -it <container\_name> /bin/bash:** Starts a bash shell inside the container. The shell is started in interactive mode, which means that it stays open, allowing you to issue more commands from inside the container until you terminate the shell with the `exit` command.
  - **docker stop <container name>:** Stops a single running container.
  - **docker rm <container name>:** Deletes a stopped container.
  - **docker images:** Shows the docker images known locally on that docker host.
  - **docker pull <image name>:** Pulls (or updates) a docker image from docker hub and stores it locally.

For complete details of these and other related commands, see the documentation for Docker and Docker Compose at:

<https://docs.docker.com/engine/reference/commandline/cli/>

<https://docs.docker.com/v1.5/compose/cli/>





## Exposing APIs externally

This chapter describes the implementation for the first scenario described in Chapter 4, “Introduction to the scenarios” on page 55. It focuses on these tasks:

- ▶ Exposing APIs by using IBM API Connect on Bluemix
- ▶ Creating a secure connection between Bluemix and on-premises systems
- ▶ Using IBM App Connect to integrate on-premises and cloud based systems

This chapter has the following sections:

- ▶ Solution outline
- ▶ Implementation
- ▶ Resources

## 5.1 Solution outline

The business case for this scenario is described in “Exposing APIs externally” on page 57. The following section summarizes the solution outline for this scenario.

### 5.1.1 Overview of the CompanyA hybrid integration landscape

CompanyA is going to use a hybrid integration scenario to create and expose their ordering service to external partners. Figure 5-1 gives an overview of what such a landscape might look like in a production scenario.

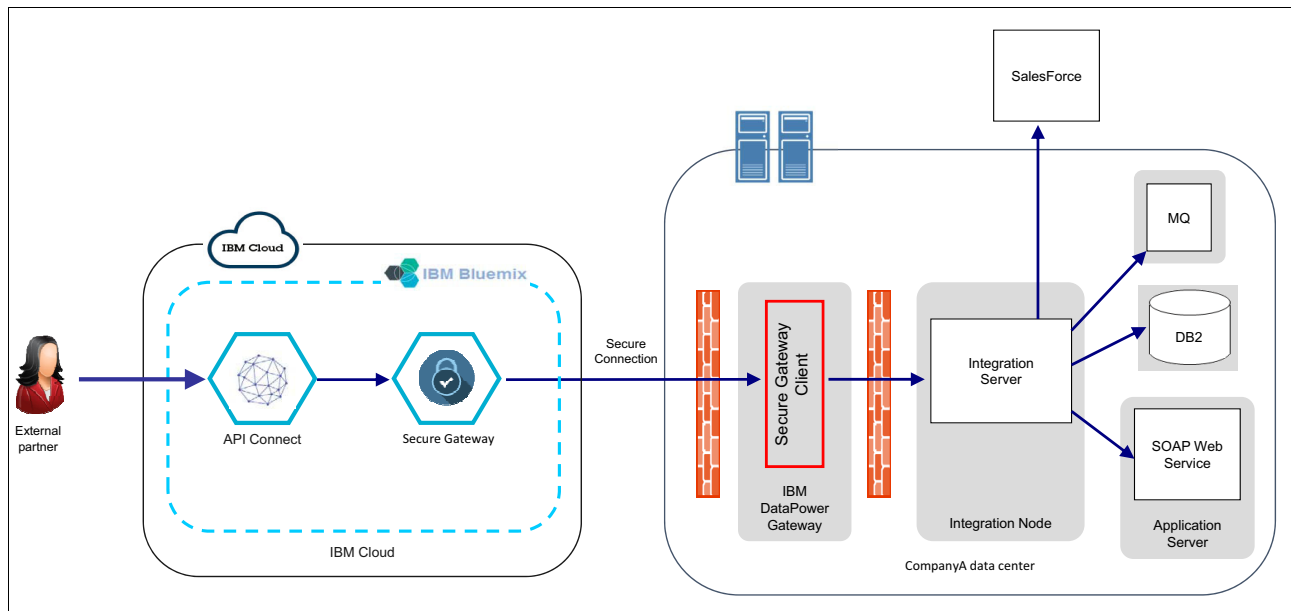


Figure 5-1 Logical view of the CompanyA hybrid environment

CompanyA wants to expose their ordering and product catalog services to external partners. To enable them to do that, they decided to use IBM API Connect on Bluemix. This way, they can easily integrate APIs with other Bluemix services, and can give developers and consumers access to their APIs without having to maintain the infrastructure themselves.

API Connect is running in Bluemix Public. Because APIs in API Connect must be able to connect to CompanyA's on-premises applications, a secure connection to their data center is needed. For this purpose, they will use a Bluemix Secure Gateway. The Secure Gateway provides secure connectivity from Bluemix to other applications and data sources running on-premises or in other clouds. Secure Gateway clients are provided for Linux, Windows, and Mac OS X. You can also run a Secure Gateway client in IBM DataPower 7.2 or later, or in a Docker container. In a production scenario, the Secure Gateway client could run on a DataPower Gateway appliance in the DMZ.

For their product catalog data, CompanyA use an existing SOAP service. Because they want to give external partners access to their product data, they want to expose this SOAP service as a REST API through the Bluemix API Connect instance.

Additional enterprise systems and data sources in their landscape, all placed in the protected zone, are IBM DB2® and IBM MQ. CompanyA will use a message flow on IBM Integration Bus to implement customer order logic, which will also interface with the cloud-based



Salesforce CRM platform. In their production landscape, the DataPower gateway is likely to be used as a secure proxy for outgoing connections from the on-premises data center.

## 5.1.2 Expose a System API for product information from a catalog

As a first step, CompanyA wants to expose their existing product catalog to external partners. API Connect is used to create a REST API that exposes the catalog SOAP service. API Connect takes care of the conversion from JSON to XML by using the built-in mapping policy. Also, create a Secure Gateway definition to allow API Connect to interface with the SOAP service in a secure way.

**Note:** To create a connection from the Bluemix Secure Gateway, the machine that you use as the on-premises server must be visible on the internet.

The API that CompanyA wants to create to expose their product catalog can be classified as a *system API*. A system API passes data from a system of record unchanged to the consumer. An API that starts one or more System APIs or data sources, and manipulates the returned data with new logic is called an *interaction API*.

## 5.1.3 Create an Interaction API for order information

This step involves exposing an API that runs on IBM Integration Bus. In API Connect, create a new API and Product to expose an orders API, which is implemented in IBM Integration Bus.

This scenario showcases caching in API Connect (for the GET operations), and transactionality in IBM Integration Bus (for the POST, PUT, and DELETE operations).

This section introduces API Security, which determines that order information is confidential, and which needs authentication and authorization for the app user. Creating an actual security definition that uses Basic Auth or OAuth is beyond the scope of the scenario.

Testing the API in this scenario is done by using the test tool in the developer portal.

The IBM Integration Bus flow gets catalog data from the SOAP service, gets and updates order information from a DB2 database, and pushes order events to IBM MQ. We need to explain why transactionality is a differentiator that positions IBM Integration Bus above the other integration options.

The customer Account data is stored in Salesforce. The IBM Integration Bus message flow queries Salesforce for the Account data, by using the Account number as key.

## 5.2 Implementation

This section covers the step-by-step implementation for the “Exposing APIs externally” scenario.

## 5.2.1 Create an API Connect instance on Bluemix

For the implementation of the scenarios, use API Connect on Bluemix. This section describes how to create and configure an API Connect instance in a Bluemix space. It assumes that you already have a Bluemix organization and space defined. If not, follow the instructions in 5.3, “Resources” on page 117.

1. Log in into your Bluemix organization and space with your IBM ID. An overview of the available service categories is displayed as shown in Figure 5-2.

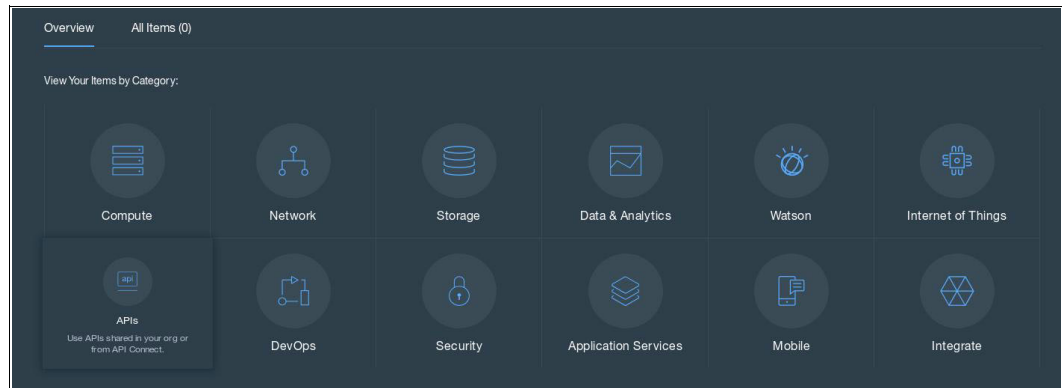


Figure 5-2 Bluemix categories

2. Navigate to the APIs category, and click **API Connect Manager** as shown in Figure 5-3. Here you can create and configure a Bluemix API Connect service instance.

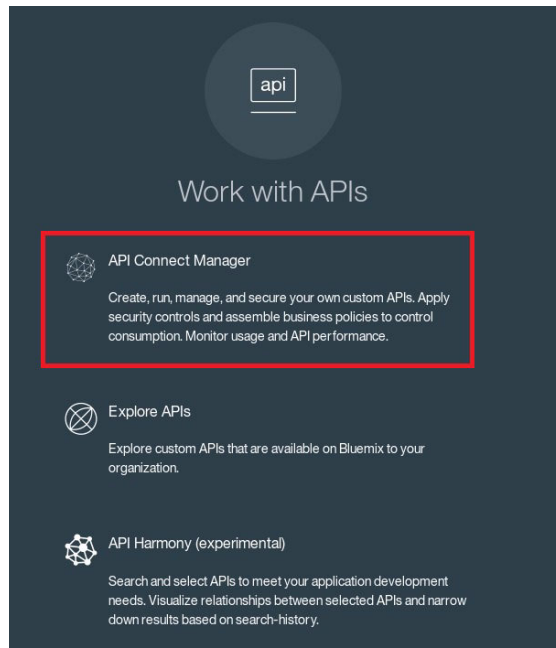


Figure 5-3 Create an API Connect service instance

**Tip:** You can only create one API Connect instance within the same Bluemix space.

3. Click the plus sign (+) to add an API Connect service.

4. Click **API Connect**, and define a service name. Enter CompanyA as shown in Figure 5-4.

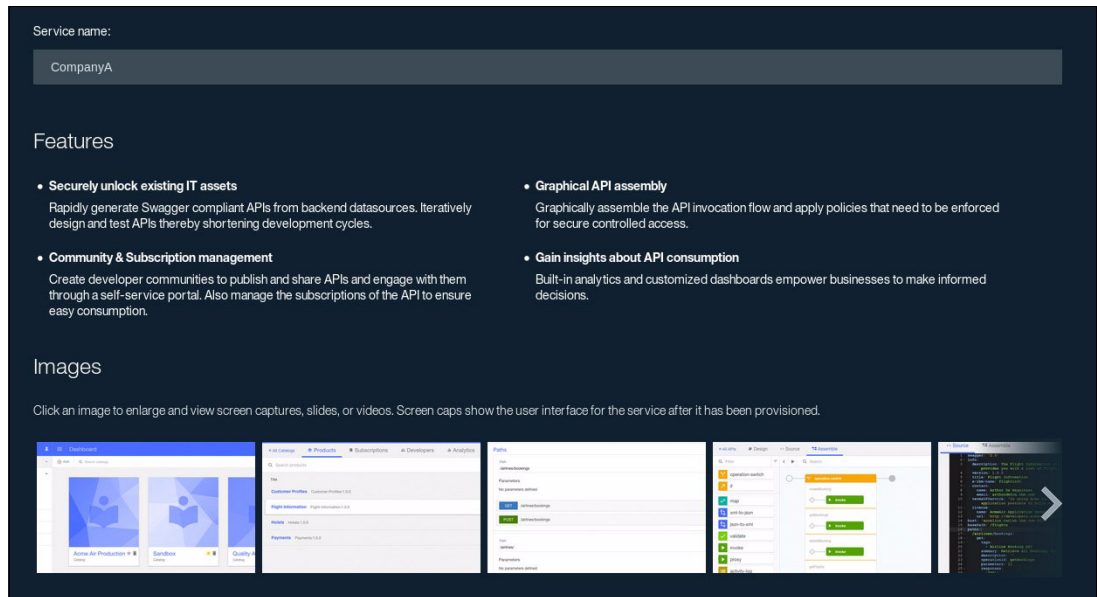


Figure 5-4 Define an API Connect service

5. For the scenarios you build and run in this book, use the Essentials pricing plan. This is a pricing plan that allows you 50K API calls per month for no fee. The Essentials plan is selected by default, as shown in Figure 5-5.

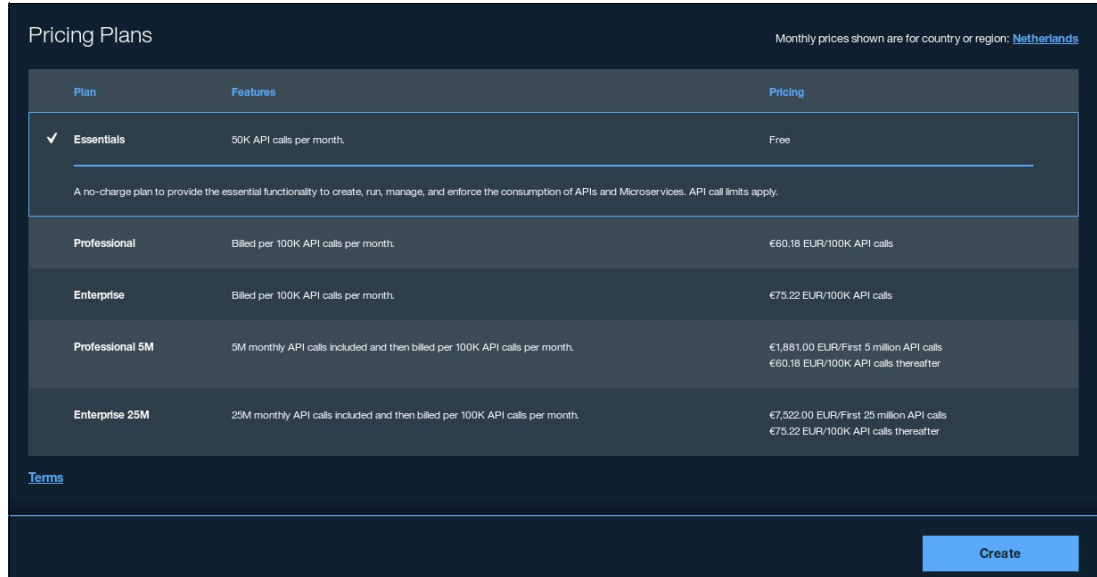


Figure 5-5 API Connect service pricing plans

- Click **Create** to confirm your choices, and create the API Connect service. The API Connect Manager interface is launched, as soon as the service is created, and you will automatically be logged in with your IBM ID. In the API Manager interface, you can create and manage your Catalogs, and manage your APIs and Products. By default, the API Manager interface shows the Draft space, where you can edit API and Product definitions as shown in Figure 5-6.

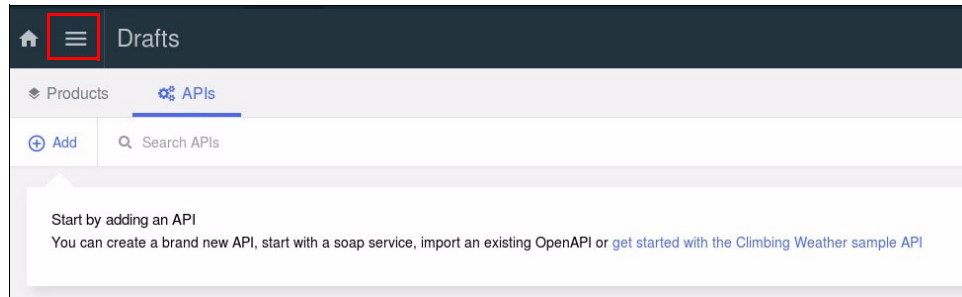


Figure 5-6 Draft space in the API Manager interface

- The push pin symbol in the navigation pane allows you to pin the pane to the interface so that it is always accessible as shown in Figure 5-7.

**Note:** To pin the UI navigation pane, click the **Navigate to** icon shown in a red rectangle in Figure 5-7. To pin the UI navigation pane, click the **Pin menu** icon.

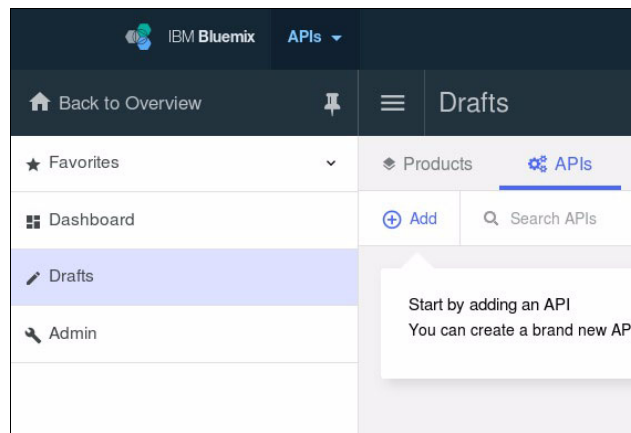


Figure 5-7 Pinning the Navigation pane to the user interface

- Navigate to the Dashboard, where you can manage API Connect catalogs. In the scenarios within this book, the Sandbox development catalog is used. In the dashboard, click the Sandbox catalog to manage this catalog.

An API Connect catalog is a staging target, and behaves as a logical partition of the gateway and the Developer Portal. The URLs for API calls and for the Developer Portal are specific to a particular catalog.

- To allow app developers to explore and use the APIs published to the Sandbox catalog, a developer portal site must be published. To make calls to the API operations, app developers need to know the Base Endpoint URL for the catalog.

Navigate to **Settings** to modify the catalog configuration as shown in Figure 5-8. Here you are able to configure the IBM Developer Portal settings. Record the API Endpoint Base URL of the catalog for later use.

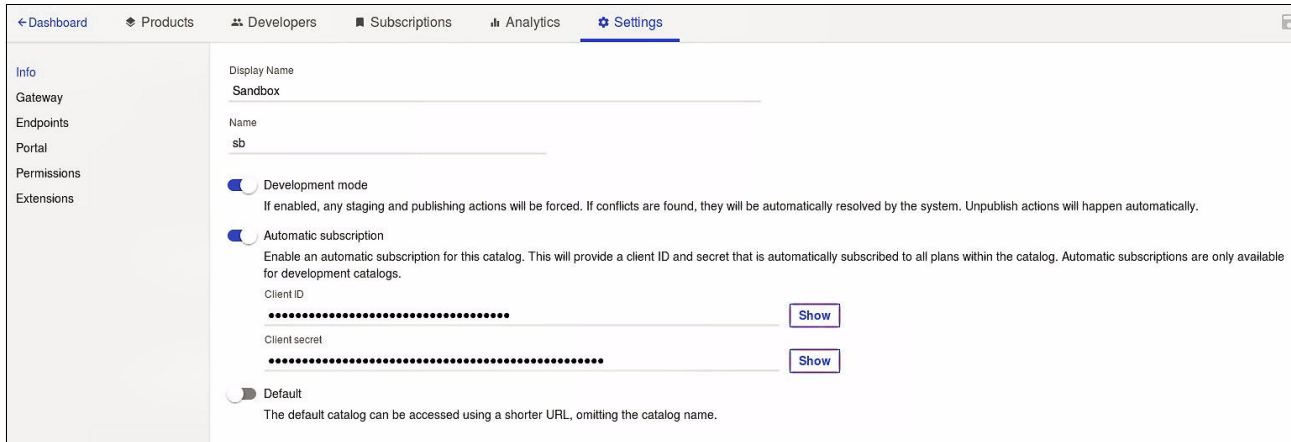


Figure 5-8 Catalog settings page

10. Within the Settings tab, click **Endpoints** to find the API Endpoint Base URL. This is the API Connect gateway URL for this catalog. You need it to call API operations externally. Figure 5-9 shows the API Endpoint Base URL for a catalog.

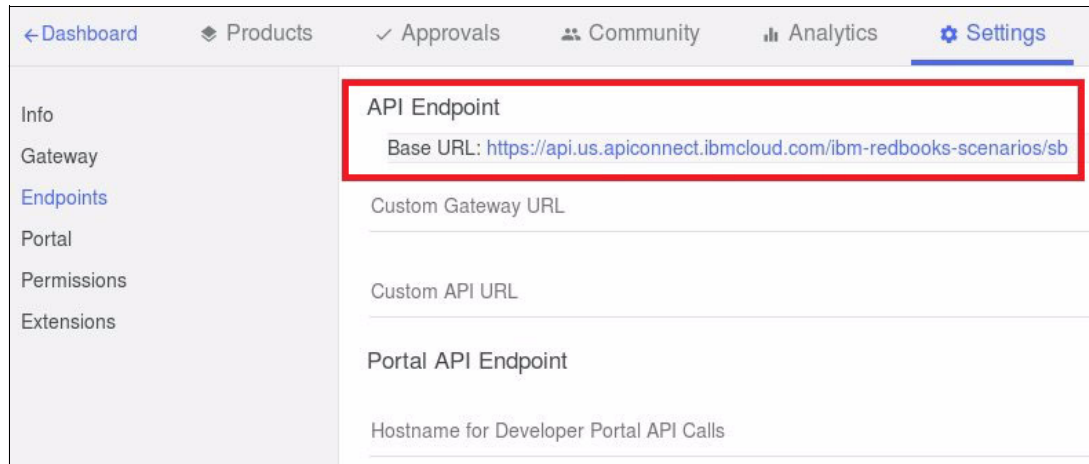


Figure 5-9 API Endpoint Base URL

11. On the Settings tab, click **Portal** to change the developer portal settings. Click **IBM Developer Portal** and accept the default portal URL. Leave all other portal settings unchanged. See Figure 5-10.

Portal Configuration

None

IBM Developer Portal

Portal URL: <https://sb-ibm-redbooks-scenarios.developer.us.apiconnect.ibmcloud.com>

Other

URL

.....

User Registration and Invitation

User Registry

Sandbox

Developers can invite collaborators and assign the following roles:

Viewer  
Viewers can only view applications and application activity.

App Developer  
App developers can create and edit applications, manage client keys and subscribe to plans.

Viewer and App Developer

Self-service onboarding

Figure 5-10 Developer portal settings

12. Click **Save** to apply and save the portal settings. A window appears indicating that you will receive an email (see Figure 5-11) after the developer portal site has been created. Note that you might need a while for the DNS records to be updated.

```
Administrator (admin),

Your Developer Portal site has been created. You can log in as the 'admin' user by using the
following one-time log in link. After you have logged in with the following link, you must change
the password for the 'admin' user account immediately.

https://sb-ibm-redbooks-scenarios.developer.us.apiconnect.ibmcloud.com/user/reset/1/1473769680/
BpQpPhy7DwDKXkLot64KM2HubFmSvYtN67b0Z10jY

For more information on setting up and customizing your new site see:

http://www.ibm.com/support/knowledgecenter/SSMNED_5.0.0/com.ibm.apic.devportal.doc/
capim_devportal_admin.html
```

Figure 5-11 Developer portal notification email

13. When the portal site has been created, and DNS records across the internet have been updated, the portal site is reachable on the Portal URL shown for the catalog. Use the one-time login link in the notification email to log in into the portal site and change the admin password. Clicking the **Home** link takes you to the Main site (Figure 5-12).

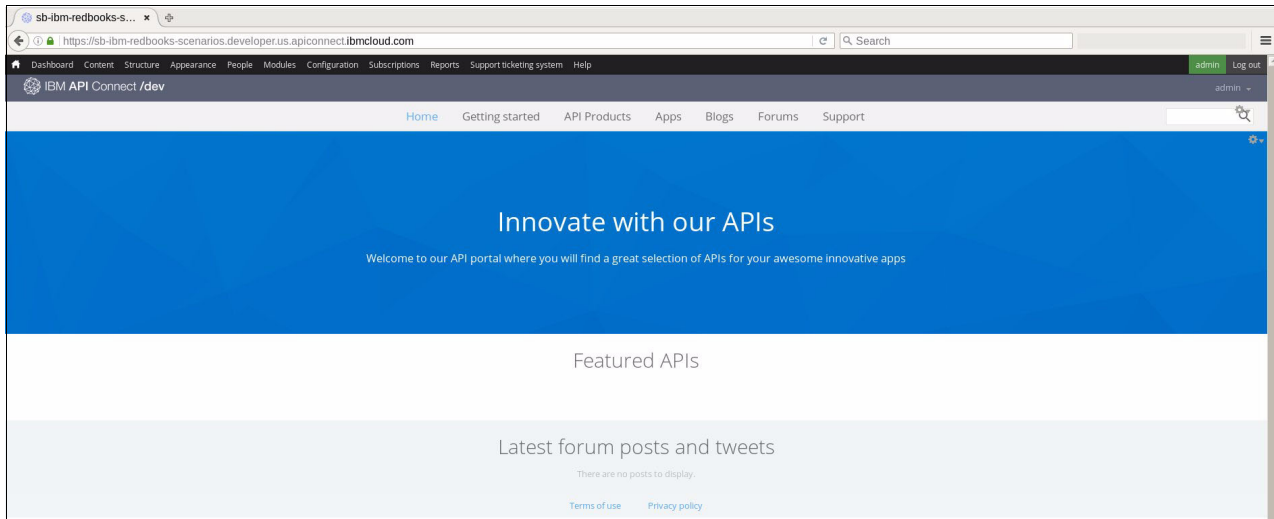


Figure 5-12 Developer Portal main site

## 5.2.2 Expose a SOAP Web Service as an API

This section covers the steps for exposing a SOAP Web Service as an API.

### Simplified landscape for exposing a SOAP Web Service as an API

For the implementation of the CompanyA scenario in this section, we use a simplified landscape, compared to the hybrid integration landscape introduced earlier in this chapter. Figure 5-13 gives an overview of the implementation components for this section.

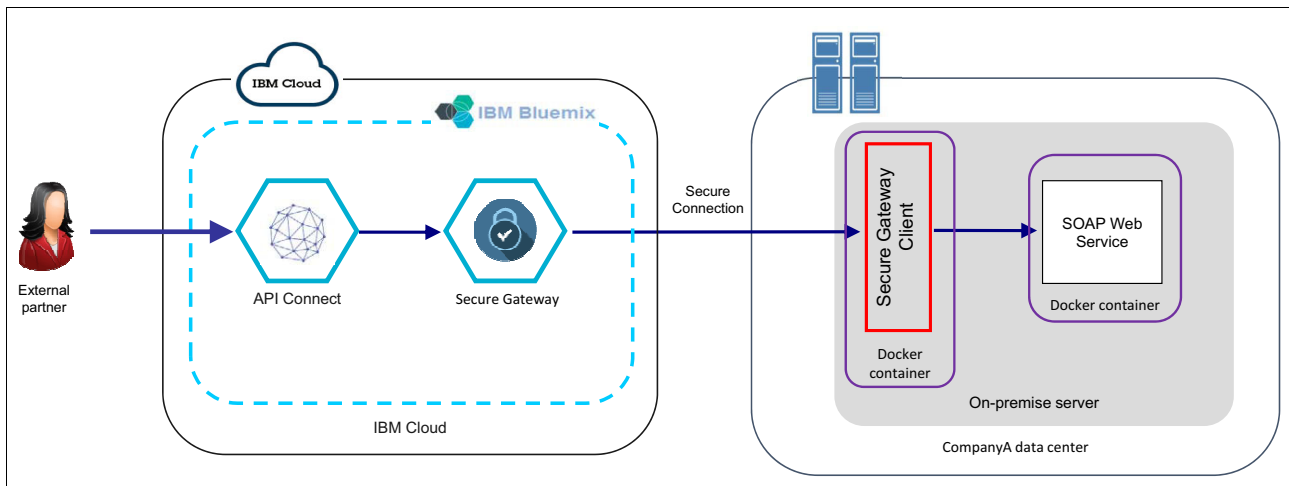


Figure 5-13 Implementation components for exposing a SOAP Web service

In this scenario implementation, some simplifications are made to the on-premises infrastructure:

- ▶ No firewall services are defined. The machine on which the docker images run can be behind one or more firewalls.
- ▶ The Catalog SOAP Service and Secure Gateway Client run in separate docker images, but on the same host machine.

The instructions for downloading and configuring the docker containers that are used in this scenario are found in Chapter 4, “Introduction to the scenarios” on page 55.

## Configure the Bluemix Secure Gateway service

The catalog SOAP web service that CompanyA wants to expose is running in their data center, and, for the scope of this scenario, in a Docker container that runs on your machine. To enable services running on Bluemix to securely connect to this web service, a secure gateway must be defined. This configuration creates a secure connection between a Secure Gateway client running on-premises, and a Secure Gateway service running in Bluemix. To define the gateway, complete these steps:

1. Return to the Bluemix console by clicking the **IBM Bluemix** icon in the API Connect management interface. Alternatively, you can browse to this URL:

<https://new-console.ng.bluemix.net/#overview>

The Bluemix console displays the various service categories, as shown Figure 5-14.

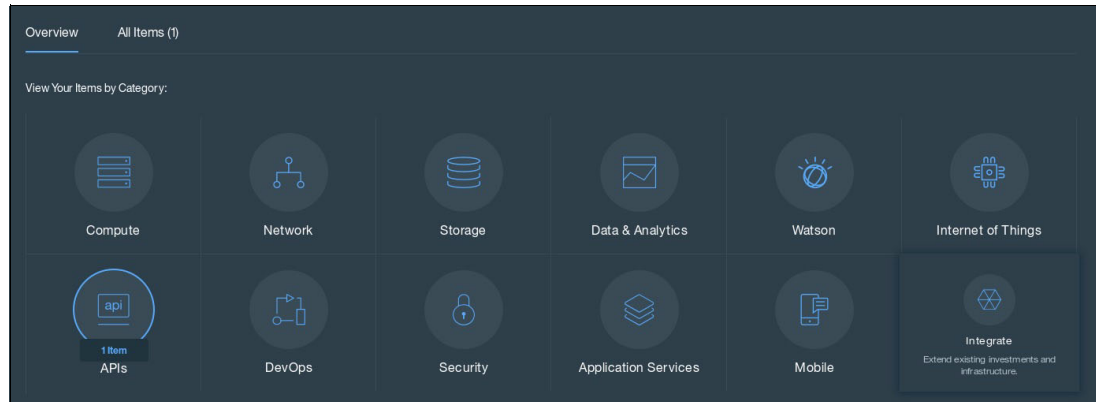


Figure 5-14 IBM Bluemix service categories

2. Click **Integrate** and then the **Plus** icon to create a new service.
3. Select the **Secure Gateway** service.
4. Select the **Standard pricing plan**, which is the only available pricing plan at the time of writing.
5. Click **Create** to create the Secure Gateway service so that Gateways can be added to the configuration.



6. Click the **ADD GATEWAY** to configure the first gateway as shown in Figure 5-15.

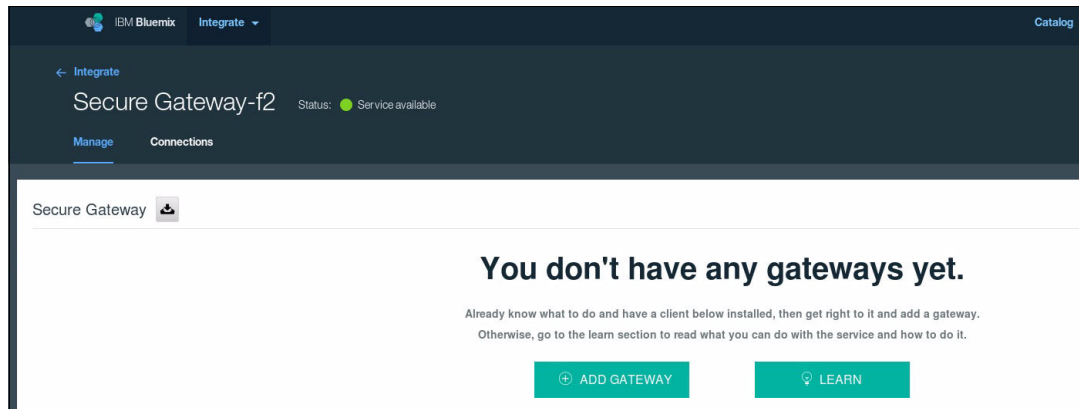


Figure 5-15 Secure Gateway service creation

7. Enter a name for the gateway, such as CompanyA\_Gateway. Leave the **Require security token to connect clients** option enabled, but disable the **Token Expiration** option.
8. Click **Add Gateway** to create the gateway definition. The window should look like Figure 5-16.

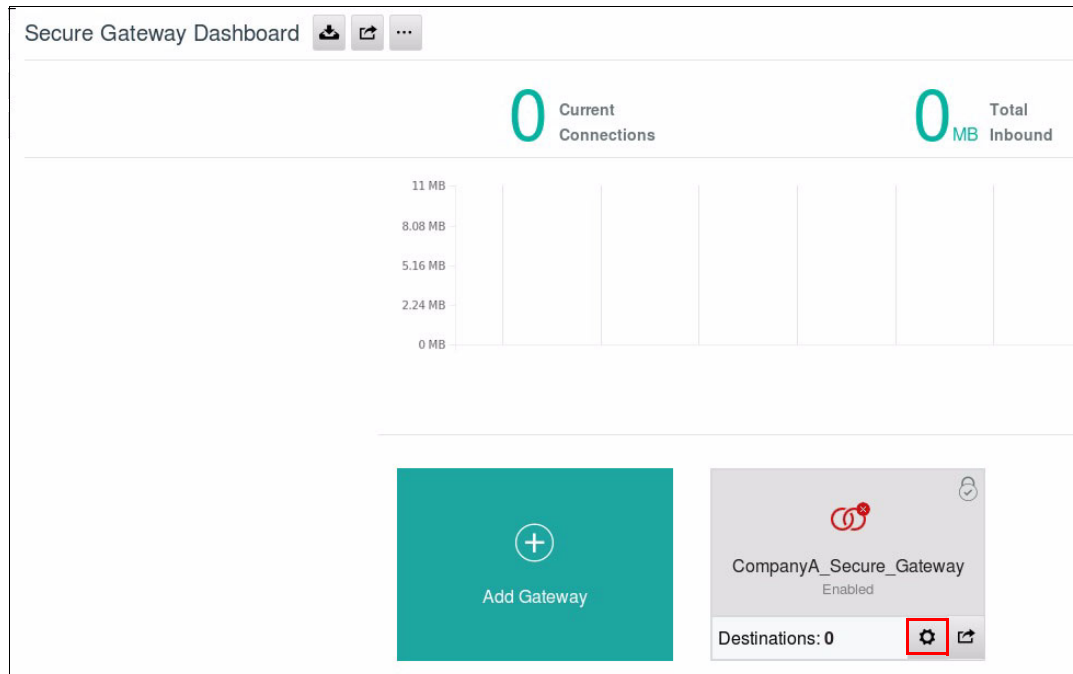


Figure 5-16 Secure Gateway definition

To enable the secure gateway connection, a destination must be added and the secure gateway client must be configured. To configure the secure gateway client, you will need the following parameters:

- The secure gateway ID
- The secure gateway secret

- Click the **Cogwheel** icon (see the box in Figure 5-16 on page 79) in the secure gateway definition to display the settings for this gateway, as shown in Figure 5-17.

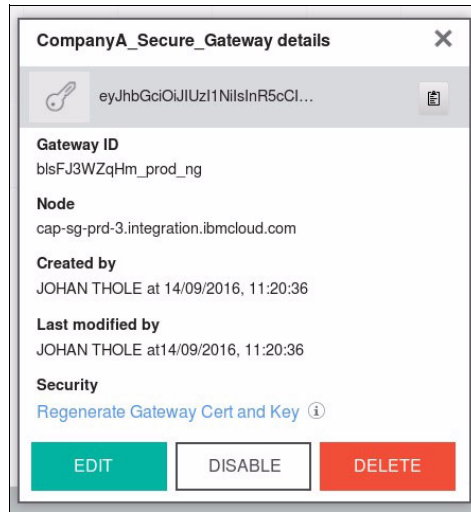


Figure 5-17 Secure Gateway ID and secret

- Click the **Copy** button next to the key to export the secret value and save it in a temporary text file. Also, copy the **Gateway ID** value to the same text file.

The temporary text file should look like Example 5-1.

*Example 5-1 Temporary text file*

---

```

SC1_GWID=blsFJ3WZqHm_prod_ng
SC1_SECTOKEN=
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjb25maWd1cmF0aW9uX2lkIjoieYmxzRkozV1pxSG
1fcHJvZGF9uZyIsInJlZ2lvd2I6IiInVzLXNvdXR5IiwiaWF0IjoxNDczODQ0ODM2fQ.t0zYREXUX9UmzB
C2jYhdonDGtC51dZOG8vARw7mc2Ks

```

---

These values will be needed for configuring the secure gateway client environment variables, so save the file for future use.

11. In the Secure Gateway, a destination must be configured for the on-premises services. Close the Settings window to return to the gateway definition. Click the **gateway name** (CompanyA\_Secure\_Gateway) and click **Add Destination**. Select that the resource is located on-premises, as shown in Figure 5-18, and click **Next**.

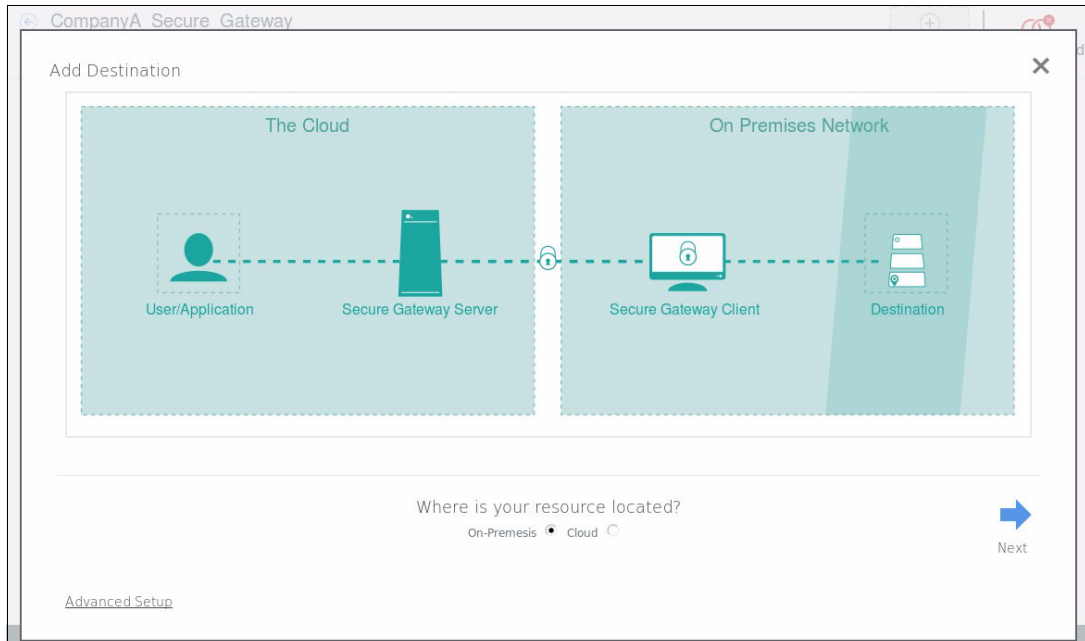


Figure 5-18 Add a Secure Gateway destination

12. Click **Advanced Setup**, and enter the values that are shown in Table 5-1.

Table 5-1 Advanced setup values

Field	Value
Name	Catalog Service
Resource Hostname	Hostname or IP address of your computer
Resource Port	7800
Protocol	TCP

13. Click **Add Destination** to add the destination to the gateway configuration.

14. After the destination is added, the Secure Gateway Client must be configured. The Secure Gateway Client runs in a docker container. Complete the docker setup in Chapter 4, “Introduction to the scenarios” on page 55 to download and prepare the containers that you need for these scenarios. Because the Gateway ID and Secret Token are unique for each gateway, they must be passed in environment variables to the Secure Gateway Client when starting the Docker container.

Ensure that the Secure Gateway Client was configured and started correctly by looking at the Bluemix Secure Gateway definition as shown in Figure 5-19.

```

File Edit View Search Terminal Help
scenario1$ export LICENSE=accept
scenario1$ export SC1_GWID=3lbCuQT45x4_prod_ng
scenario1$ export SC1_SECTOKEN=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJVrJJc7CdwUDyyq1dMQ
scenario1$ docker restart scenario1_sgc_1
scenario1_sgc_1
scenario1$

```

Figure 5-19 Setting the Secure Gateway Client variables

The status of the Gateway should be Connected as shown in Figure 5-20.

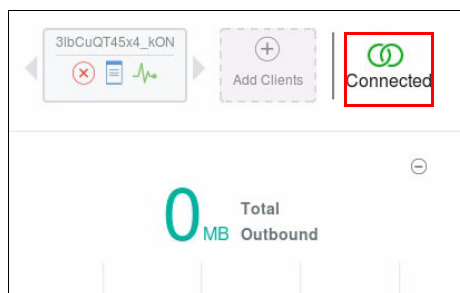


Figure 5-20 Secure Gateway connection status

## Start the Catalog service on the on-premises server

Complete the following steps to start the Catalog service on the on-premises server:

1. On the system where the docker images are installed, check whether the Catalog service is running by issuing the **docker-compose ps** command. The output of this command should look like Figure 5-21 showing the running services. Note that the local ports of the docker containers (32775 and 32776) might be different from those shown in this figure.

```

$ docker-compose ps
-----
Name                Command                                State          Ports
-----
scenario1master_iib-op_1  iib_mgmt_manage.sh                    up            0.0.0.0:1414->1414/tcp, 0.0.0.0:4414->4414/tcp, 0.0.0.0:7800->7800/tcp
scenario1master_orderdb_1 /entrypoint.sh db2start                up            22/tcp, 0.0.0.0:32774->50000/tcp
scenario1master_sgc_1    node lib/secgwclient.js 31 ...         up            0.0.0.0:32776->4414/tcp, 0.0.0.0:32775->7800/tcp
scenario1master_stub_1   iib_manage.sh                          up

```

Figure 5-21 Running docker container services

2. In a web browser, open the WSDL of the local service by using the following URL:

<http://localhost:32775/CatalogServices?wsdl>

**Note:** This command assumes that you are working on Linux. For Windows, you need to issue a **docker-machine default ip** command to get the IP of the Docker VM. Rather than using the localhost, you need to use this IP address.

The WSDL for the Catalog service is returned. This result confirms that the application is running.

**Note:** Because the Secure Gateway client was configured and connected, it is also possible to retrieve the WSDL from an external destination. In a production scenario, address filtering would be set up to ensure that only authorized Bluemix services can connect through the gateway.

3. To determine the external endpoint, go to the **Secure Gateway definition** for the **CompanyA\_Secure\_Gateway** in the Bluemix console, select the destination that was configured, and click the **Cogwheel** icon as shown in Figure 5-22.

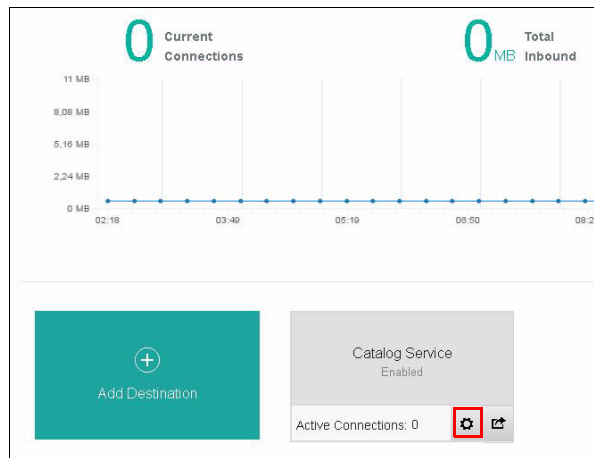


Figure 5-22 Secure Gateway destination settings

This action opens the settings window, from which the Cloud host endpoint address and port can be copied. See Figure 5-23.

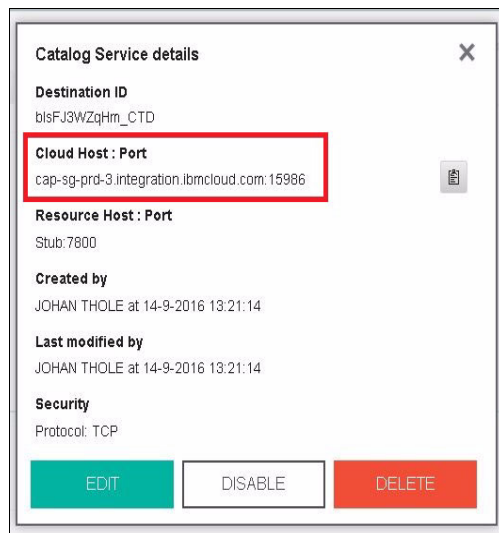


Figure 5-23 Secure Gateway connection details

4. In a web browser, enter the Cloud endpoint URL to retrieve the Catalog service WSDL as shown in Figure 5-24.

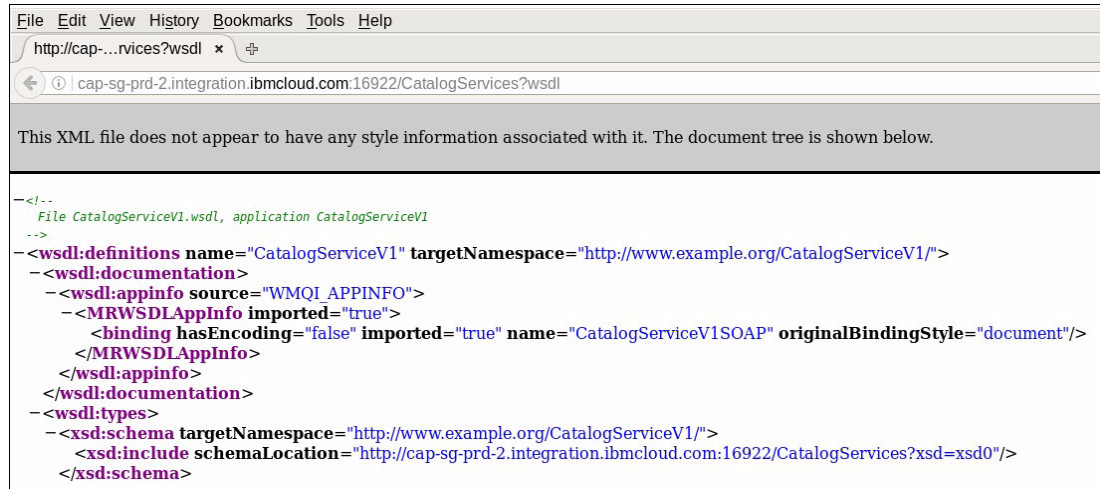


Figure 5-24 Retrieve the WSDL from the Cloud endpoint

## Create the Products API in the API Designer

Next, create the Products API in the API Designer using these steps:

1. Return to the Bluemix API Management instance you created earlier, by clicking the **Instance** icon (Figure 5-25).

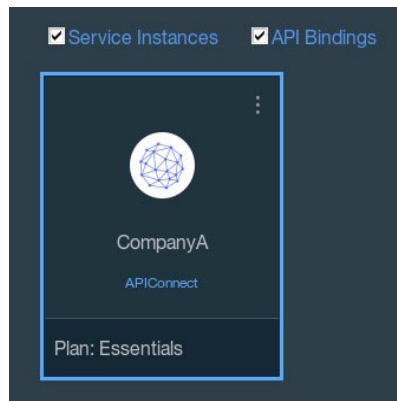


Figure 5-25 CompanyA API Connect instance

2. This action opens the API Manager interface for the API Connect instance in your Bluemix organization. In the navigation pane, click **Drafts**. Within the Drafts window, click the **APIs** tab, and click **+ Add** → **API**. This action creates a REST API definition.
3. In the dialog box, provide the following values:
  - Title: Products
  - Name: products
  - Version: 1.0.0
4. Click **Add to a new product** and enter the product name. For this scenario, use CompanyA Products.

- Click **Add** to create the Product and API definitions in the Draft space. The API Design view opens as shown in Figure 5-26.

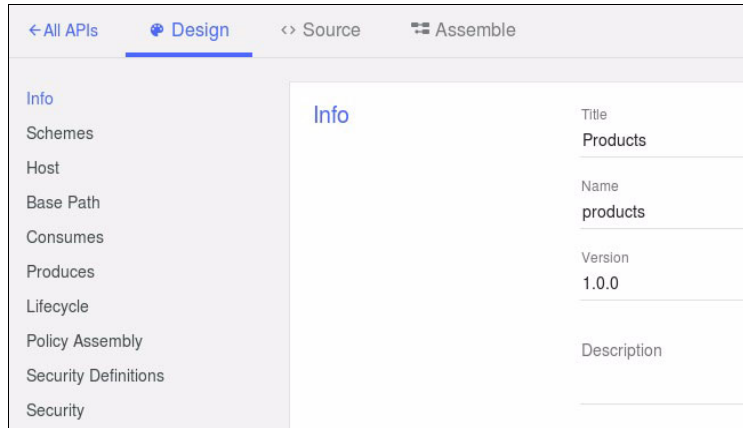


Figure 5-26 API Manager Design view

- In the Base Path field, enter /Products.
- As a next step, a JSON schema definition for the output of the API is created. Select the Definitions section and create the definitions as shown in Figure 5-27.

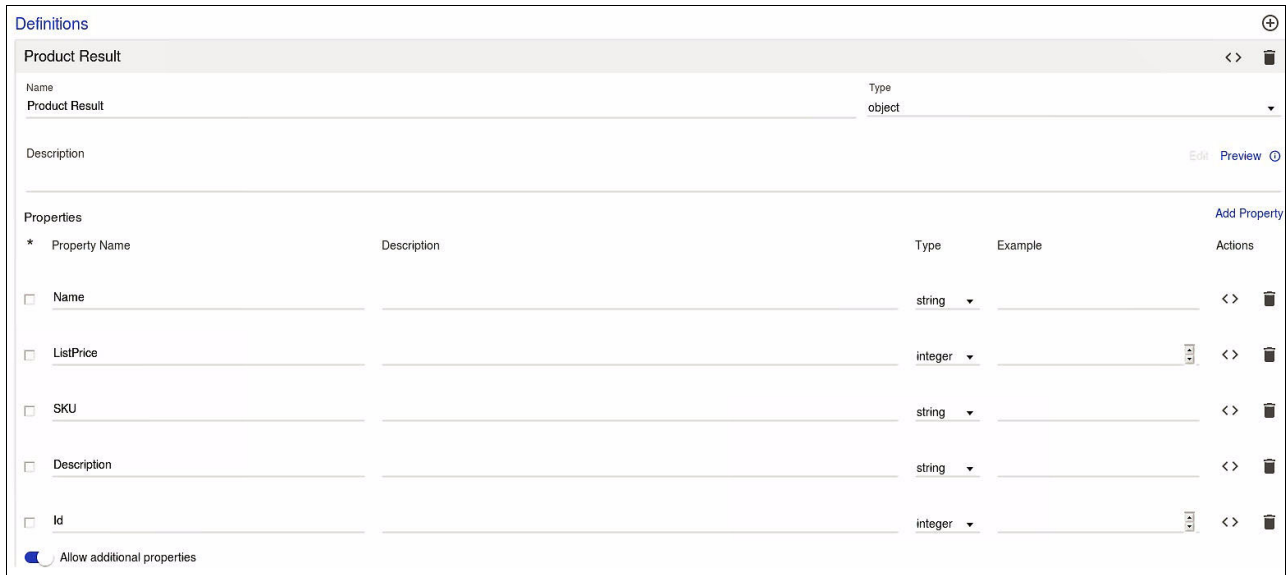


Figure 5-27 JSON Schema definition for the Products API

- Define the JSON Schema object name as Product Result. For the properties, use the values in Table 5-2.

Table 5-2 Properties table

Property name	Property type
Name	String
ListPrice	Integer
SKU	String

Property name	Property type
Description	String
Id	Integer

9. In the Paths section, click the **Add Path** icon. Add the /product path, and click the **GET /product** operation to expand it.
10. In the input fields for the GET /products operation, click **Add parameter**, and on **Add new parameter**. Add an input parameter with name product\_id and type Integer. Specify that it is a Query parameter, and that it is required. Marking a parameter as required is indicated to users in the OpenAPI (Swagger 2.0) definition of the API, is enforced by validate policies, and will result in the test tool always generating the parameter as part of a sample API call.
11. Select the **Product Result** definition as the response schema for the **200 OK** response. An example of the configured operation is shown in Figure 5-28.

Parameters					Add Parameter
Name	Located In	Description	Required	Type	
product_id	Query		<input checked="" type="checkbox"/>	integer	

Responses				Add Response
Status Code	Description	Schema		
200	200 OK	Product Result		

Figure 5-28 Definitions for GET /product operation

12. Click **Save** to save the API definition.
 

Now that you have defined the input and output parameters for the products API, the invocation for the Catalog SOAP service can be configured.
13. In the **Services** section of the API Design view, click the **Add service** icon. The **Import web service from WSDL** window opens, and presents you with three choices:
  - Upload file
  - Load from URL
  - Find in registry
14. Click **Load from URL** and enter the WSDL URL of the Cloud host endpoint. In the example, the URL is:

`http://cap-sg-prd-2.integration.ibmcloud.com:16922/CatalogServices?wsdl`

**Tip:** This is the Cloud host endpoint that you copied from the destination settings (see Figure 5-23 on page 83).



- Click **Next** and wait until the WSDL loads. Select the **CatalogServiceV1** web service and click **Done** as shown in Figure 5-29.

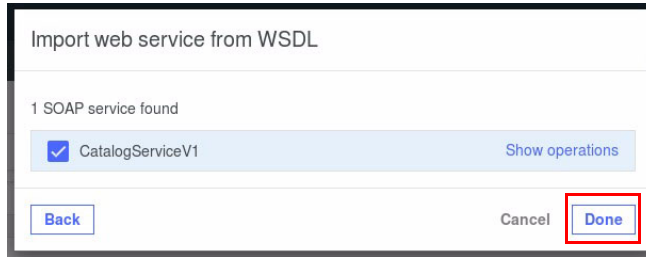


Figure 5-29 Import CatalogServiceV1 SOAP web service

- Click the **Assemble** tab in the API Management interface, and select **DataPower Gateway policies**. From the predefined assembly, delete the existing invoke policy on the canvas by hovering your cursor over the policy and then clicking the **Delete policy** trash can icon.
- From the palette, drag the **getProduct web service** operation onto the dashed box that is displayed on the canvas.

An invoke policy and two map policies are placed in the assembly. The first map policy assigns variables to the input of your web service invocation, and the second policy assigns outputs of your web service invocation to variables. The outputs of the first map and the inputs of the second map are generated from the WSDL provided when you imported the web service. The resulting assembly is shown in Figure 5-30.



Figure 5-30 SOAP service invocation through an Invoke policy

- Click the **getProduct: input map policy** and then click the **Edit inputs** pencil icon in the inputs column of the property sheet. An example is shown in Figure 5-31.

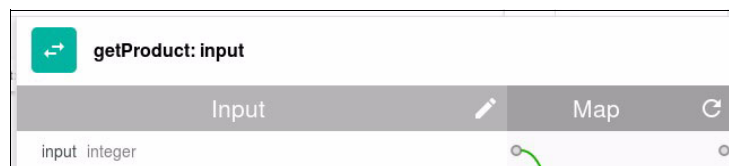


Figure 5-31 Edit inputs of the getProducts: input map policy

19. Configure the input for the mapping as shown in Table 5-3.

Table 5-3 Properties table

Property	Value
Context variable	request.parameters.product_id
Name	input
Content type	none
Definition	Integer

20. Click **Done**, and map the input of the mapping policy to the output, as shown in Figure 5-32.

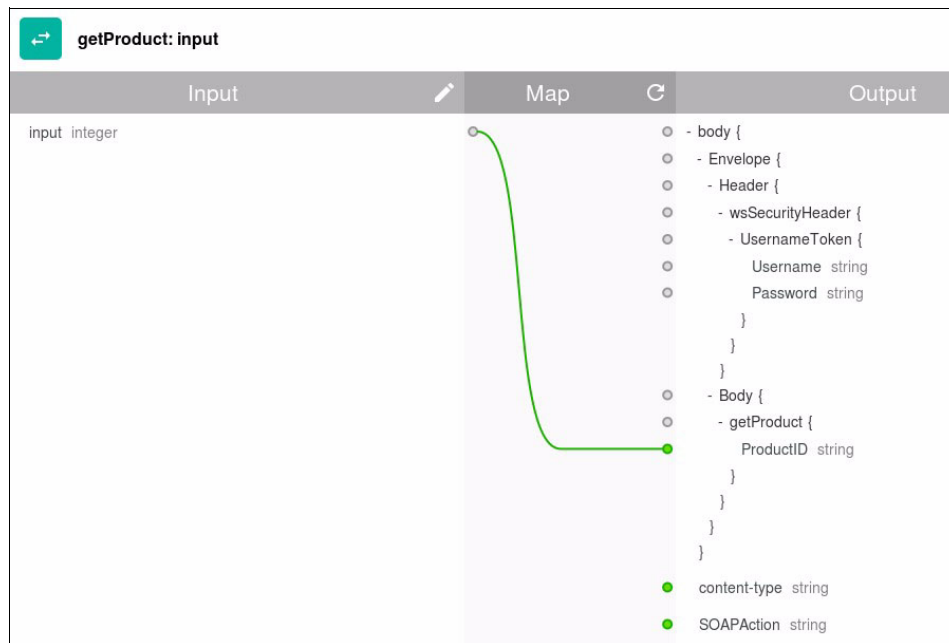


Figure 5-32 Configure the input mapping policy

21. Click the **(X)** icon on the property sheet to close the input mapping policy.

The `getProduct:input` policy maps the input from the API call (the `product_id` query parameter) to the input message for the service invocation. The `getProduct:output` policy maps the response from the invocation to the response for the API call.

22. Click the `getProduct:output` map policy and click the **Edit outputs** icon. Define the web service output as shown in Table 5-4.

Table 5-4 Properties table

Property	Value
Context variable	message.body
Name	output
Content type	none
Definition	#/definitions/Product Result

23. Map the inputs to the outputs as shown in Figure 5-33.

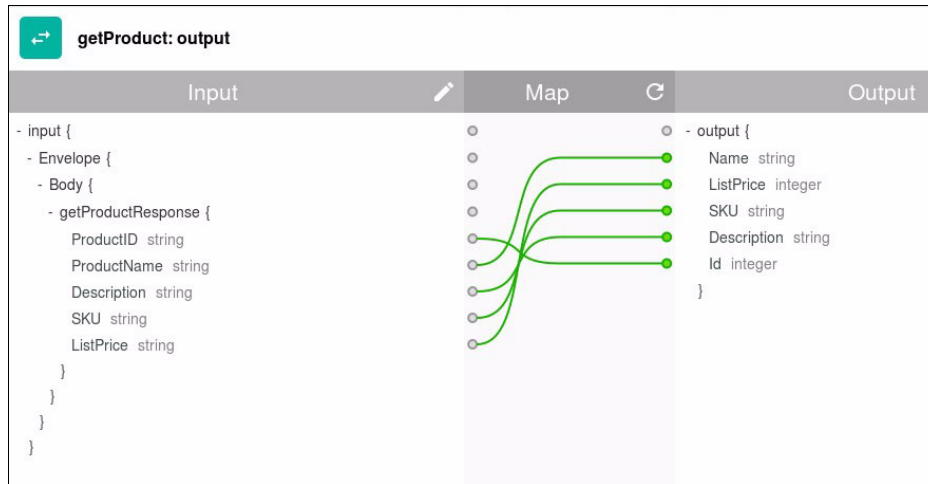


Figure 5-33 Configure the output mapping policy

You have included the web service invocation in your assembly, mapped an input parameter to the appropriate part of the SOAP request, and mapped the appropriate part of the SOAP response to a JSON output.

24. Save the API definition before continuing.

### Test the API definition in the API Manager interface

Next, test the API definition in the API Manager interface by using these steps:

1. The API Manager interface has a built-in test tool. To start the test tool, click the indicated icon in the Assembly tab, as shown in Figure 5-34.

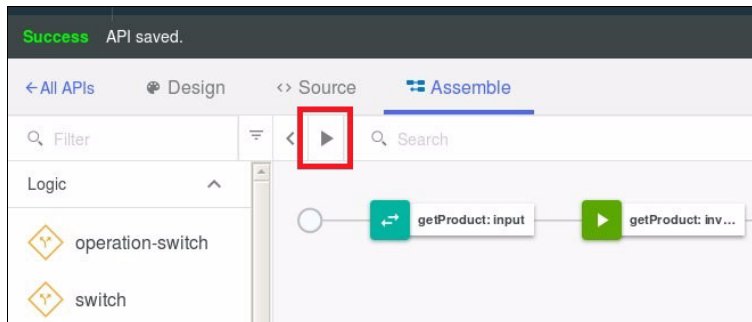


Figure 5-34 Start the API Manager test tool

2. Perform the following steps in the test tool:
  - a. If you have used the test tool before, click **Change setup**.
  - b. In the Catalog field, select the **Sandbox** catalog.
  - c. In the Product field, select the **CompanyA Products** Product and then click **Republish product** to publish your Product so that it can be tested. The catalog that you created earlier has the “development mode” and “automatic subscription” settings enabled by default. Whenever you publish or republish a product to this catalog, you can test the included APIs immediately.
  - d. Click **Next**.

- e. In the Operation field, select **get /product**.
  - f. In the **product\_id** field, enter 12345.
3. Click **Invoke**. The response is displayed.

This completes the first part of this scenario. In this section, you performed the following activities:

- ▶ Configured an IBM Bluemix API Connect instance
- ▶ Configured an IBM Bluemix Secure Gateway Service instance
- ▶ Created a REST API definition
- ▶ Started an on-premises SOAP service from API Connect on Bluemix
- ▶ Tested the API definition

### 5.2.3 Create an Interaction API for order information

In this scenario, a predefined message flow is imported in IBM Integration Bus. Salesforce connectivity is configured, and you push an API to API Connect. In API Connect, you make the API available to application developers.

#### Simplified landscape for the order information API

For the implementation of this scenario, a simplified landscape is presented again, as shown in Figure 5-35.

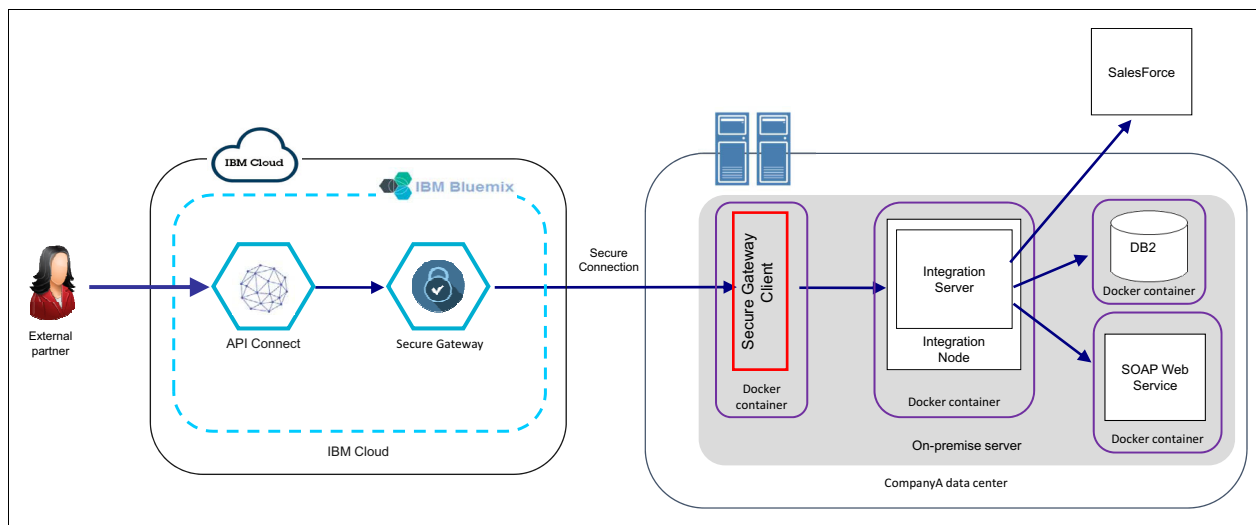


Figure 5-35 Implementation components for exposing an API from IBM Integration Bus

The following simplifications were made:

- ▶ There are no firewall services defined.
- ▶ The Docker images all run on the same computer, albeit in separate docker containers.

#### Introduction to REST API project in IBM Integration Bus

In IBM Integration Bus, a REST API is a specialized application that can be used to expose integrations as a RESTful web service that can be called by HTTP clients.

IBM Integration Bus also provides a set of REST nodes that you can use to interact either synchronously or asynchronously with external REST APIs.

Integration Bus V10 introduces a new type of project, or container, called a REST API. A REST API in Integration Bus is a specialized application that allows you to expose a set of integrations as a RESTful web service. It has these characteristics:

- ▶ Operations that are defined in the REST API are implemented as normal subflows.
- ▶ The REST API container automatically takes care of the routing of inbound HTTP requests to the correct subflow for the operation being called.
- ▶ You simply need to connect the dots between the Input and Output nodes in each subflow.
- ▶ REST APIs support all of the Integration Bus features that you can use with applications (such as shared libraries, monitoring, activity log). All message flow nodes can be used within a REST API.
- ▶ A REST API describes a set of resources, and a set of operations that can be performed against those resources.

### **Resources**

A REST API has a base path, which is the root from which all of the resources and operations are available. An example base path might be:

```
http://mycompany.com:7843/customerdb/v1
```

Each resource in a REST API has a path, relative to the base path that identifies that resource. Example resources might be:

```
/customers - all of the customers in the database  
/customers/12345 - customer #12345  
/customers/12345/orders - all orders for customer #12345  
/customers/12345/orders/67890 - order #67890 for customer #12345
```

### **Operations**

Each resource in a REST API has a set of operations. An operation in a REST API has a name, and an HTTP method, such as GET, POST, PUT, or DELETE.

The combination of the path of the HTTP request and the HTTP method identifies which resource and operation are being called.

Example operations on the resource `/customers/12345` might be:

- ▶ **GET `getCustomer`**: Retrieve the customer details from the database
- ▶ **PUT `updateCustomer`**: Update the customer details in the database
- ▶ **DELETE `deleteCustomer`**: Delete the customer from the database

To call the `updateCustomer` operation, the HTTP client must make an HTTP PUT request to:

```
http://mycompany.com:7843/customerdb/v1/customers/12345
```

### **Parameters**

Each operation that is defined in a REST API can also specify a set of parameters. Parameters can be used to pass information in to the operation. These parameters are in addition to the body passed in the HTTP request.

Integration Bus supports three different types of parameters:

- ▶ **Path parameters**: One or more parts of the path for a resource can be defined as a variable. For example, the customer ID in the previous examples is a path parameter:

```
/customers/{customerId}/orders/{orderId}  
/customers/12345/orders/56789
```

- ▶ Query parameters: One or more parameters can be specified in the URL following the path:  
/customers?maxResults=5&name=2
- ▶ Header parameters: One or more parameters can be specified in the headers of the HTTP request:  
Api-Client-Id: ff6e2c5d-42d5-4026-8f7f-d1e56da7f777

### **Swagger**

Swagger is an open standard for defining a REST API. For more information, see this website:

<http://swagger.io/>

Along with the specification, Swagger has a set of open source tools that can be used to interact with Swagger documents and the REST APIs that they describe.

A Swagger document includes definitions of the resources, operations, and parameters in a REST API. It can also include JSON schema that describes the structure of the request and response bodies to an operation. A Swagger document can be thought of as the REST API equivalent of a WSDL document for a SOAP web service.

Integration Bus supports Swagger 2.0. The specification for Swagger 2.0 can be found at:

<https://github.com/swagger-api/swagger-spec/blob/master/versions/2.0.md>

REST APIs in Integration Bus are described by a Swagger 2.0 document. You can build a REST API by importing a Swagger document. Alternatively, you can now build a REST API from scratch by using the toolkit.

### **Cross-Origin Resource Sharing (CORS)**

If you are making an HTTP request from a web page to a REST API or other HTTP service that is deployed to Integration Bus, it is likely that a cross-origin request will need to be made.

Integration Bus V10 now includes built in support for Cross-Origin Resource Sharing (CORS). At V10 GA, this support was limited to the HTTP listener for the integration server, but as of V10 FP1 it is available for the HTTP listener for the integration node as well.

The support is disabled by default, but can be easily enabled with the following commands:

```
mqschangeproperties IB10NODE -e default -o HTTPConnector -n corsEnabled -v true
mqschangeproperties IB10NODE -b httpListener -o HTTPConnector -n corsEnabled -v true
```

The default settings for CORS should be sufficient for most needs, but more configuration options are available for fine level control. When CORS settings are modified, the changes are effective immediately, so you do not have to restart the integration server or node.

### **Accessing parameter values**

Example 5-2 shows some examples of how to access parameter values in the various transformation languages supported by Integration Bus.

*Example 5-2 Accessing parameter values in various languages*

---

```
ESQL:
DECLARE max INTEGER -1;
IF FIELDTYPE(InputLocalEnvironment.REST.Input.Parameters.max) IS NOT NULL THEN
  SET max = InputLocalEnvironment.REST.Input.Parameters.max;
```

```
END IF;
```

```
Java:
```

```
MbElement maxElement =  
inLocalEnvironment.getRootElement().getFirstElementByPath("/REST/Input/Parameters/  
max");  
int max = -1;  
if (maxElement != null) {  
    max = Integer.valueOf(maxElement.getValueAsString());  
}
```

```
.NET:
```

```
NBElement maxElement =  
inLocalEnvironment.RootElement["REST"]["Input"]["Parameters"]["max"];  
int max = -1;  
if (max != null) {  
    max = (int) maxElement;  
}
```

---

## IBM Integration Bus artifacts for the scenario

The IBM Integration Bus artifacts discussed in this scenario are available for download from GitHub repository:

[https://github.com/sg248351/scenario1/tree/master/1\\_coding](https://github.com/sg248351/scenario1/tree/master/1_coding)

To download and install the artifacts, complete these steps:

1. Download the `iib_scenario1.zip` and `SCENARIO01.bar` files on your computer in which IBM Integration Bus toolkit is installed from the GitHub repository.
2. Start the Integration Bus toolkit. You can import the artifacts in your workspace by using **File** → **Import** → **Project Interchange**.
3. On the Import Projects window, specify the `iib_scenario1.zip` file that you just downloaded. Select all projects, then click **Finish**. Now you can see the projects as shown in Figure 5-36.

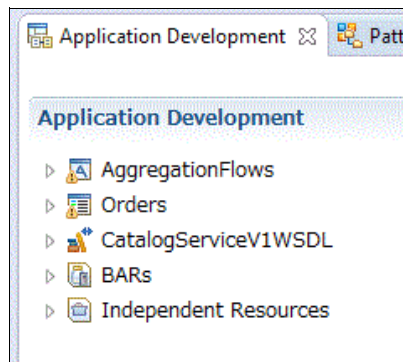


Figure 5-36 Imported projects

- By opening **Orders** → **REST API Description**, you can see that three operations are implemented (postOrder, deleteOrder, and getOrder) as shown in Figure 5-37. You can see the path of operations, http methods of each operation, and request parameters.

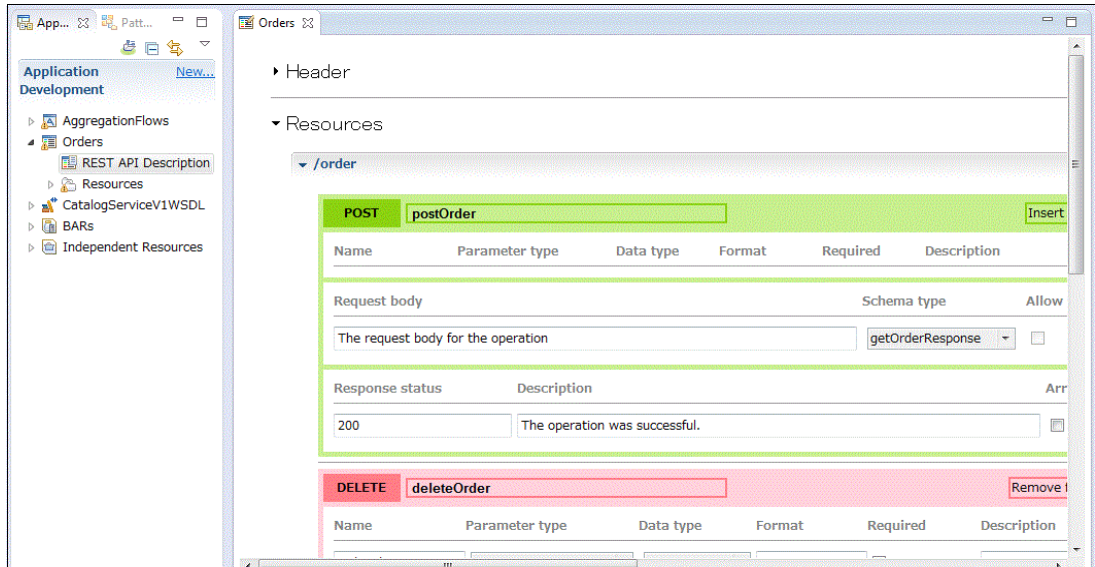


Figure 5-37 REST operations

- If your operations accept query or header parameters, then add them in using the (+) icon on the operation. See Figure 5-38.

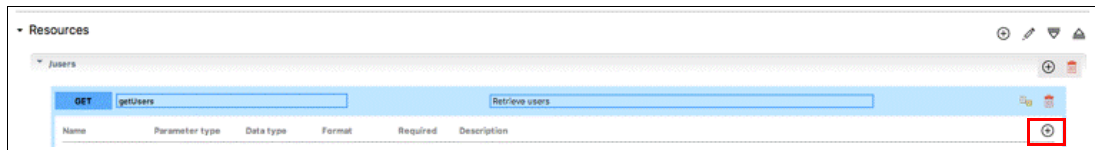


Figure 5-38 Add parameters to operations

- Each of these operations is implemented as a subflow, and reference gets added into the main message flow as shown in Figure 5-39.

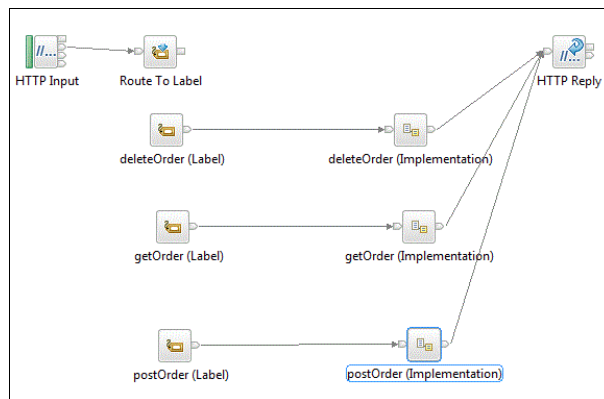


Figure 5-39 Main message flow for Orders API

- You can see this flow by opening it with the Message Flow Editor. This view can be found at **Orders** → **Resources** → **Flows** → **gen** → **Orders.msgflow**, then right-click the flow, and clicking **Open with** → **Message Flow Editor**.



8. Now look at the getOrder operation. This operation is requested with http GET method, which has one query parameter, **orderId** as shown in Figure 5-40.

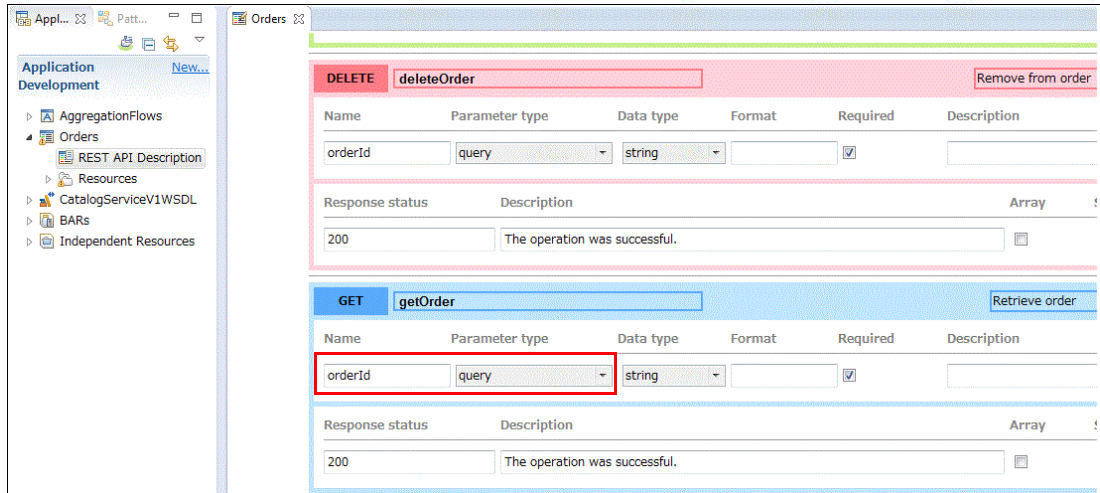


Figure 5-40 Implemented operations details

9. Scroll to the right side and click the button marked by red square in Figure 5-41.

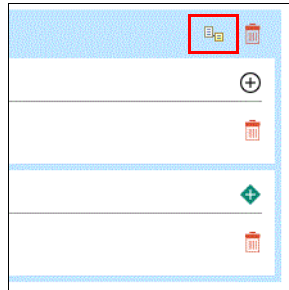


Figure 5-41 Implementing a subflow

### Implementation of getOrder subflow

The getOrder operation is implemented by using an aggregation technique.

IBM Integration Bus provides three message flow nodes that support aggregation:

- ▶ The AggregateControl node
- ▶ The AggregateRequest node
- ▶ The AggregateReply node

You can also use these aggregation nodes to issue requests to applications outside the integration node environment. Messages can be sent asynchronously to external applications or services. The responses are then retrieved from those applications, and the responses are combined to provide a single response to the original request message.

These nodes can help to improve response time because slow requests can be performed in parallel, and they do not need to follow each other sequentially. If the subtasks can be processed independently, and they do not need to be handled as part of a single unit of work, they can be processed by separate message flows.

The design of the subflow for getOrder operation is as shown in Figure 5-42.

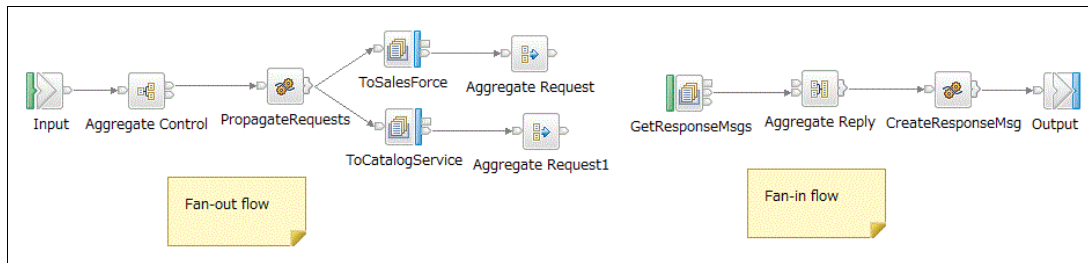


Figure 5-42 Aggregation flow

The sequence of steps involved in processing of the getOrder operation is as follows:

1. Receive an order information request from clients.
2. Get the order ID from the query parameter, **orderId**, in the Compute node.
3. Retrieve the order information from the tables ORDER and PRODUCTLIST by using the **orderId** as the key. The ORDER table stores order information. The structure of the ORDER table is shown in Table 5-5.

Table 5-5 ORDER table

Parameter	Meaning
ORDERID	Order ID (Primary key)
ACCOUNTNAME	Account name
TOTAL	The amount of order
STATUS	The status of order
CREATIONDATE	The creation date of order
CAMPAIGNID	Campaign ID (option)

The PRODUCTLIST table stores the line items list that is related to each order. The structure of the PRODUCTLIST table is shown in Table 5-6.

Table 5-6 PRODUCTLIST table

Parameter	Meaning
PRODUCTLISTID	Product list ID (Primary key)
ORDERID	Order ID (Foreign key)
PRODUCTID	Product ID
QUANTITY	The quantity of the product
PRICE	The price of the product

- Now fan out the two requests to retrieve the account details and the product details as shown in Figure 5-43.

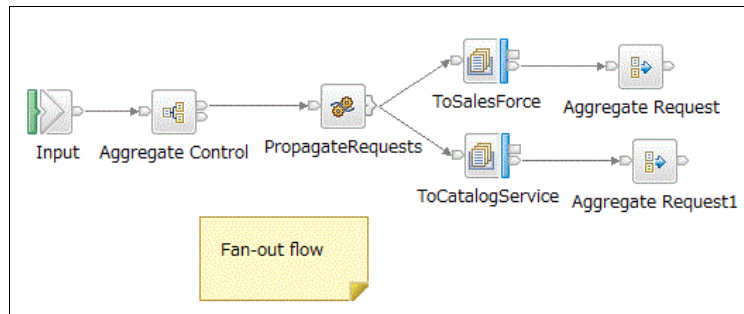


Figure 5-43 Aggregation Fanout flow

- Account details are obtained from the Salesforce CRM system. Hence, one of the fan-out messages has the account name that will be used to query the account details from the Salesforce CRM. This part is implemented at the `flow1_salesforce.msgflow` in the `AggregationFlows` application as shown in Figure 5-44. You can see the **Salesforce Request** node that is used to access the Salesforce system. See 5.3, “Resources” on page 117 for the prerequisites for Salesforce and IBM Integration Bus for the integration.

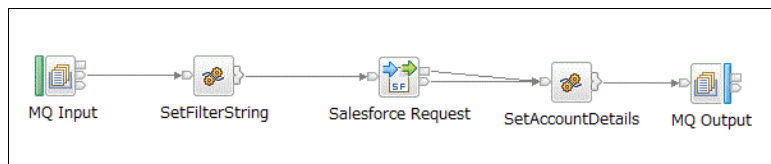


Figure 5-44 `flow1_salesforce.msgflow`

- The Salesforce Request node window provides the settings for the Salesforce system as shown in Figure 5-45. You can see the value of the **Security Identity** property is `sf1`. This identity is related to the credentials for the Salesforce system. You are going to configure it in the next step. Select the **Account** object from the dropdown list and select the operation type as **Retrieve**.

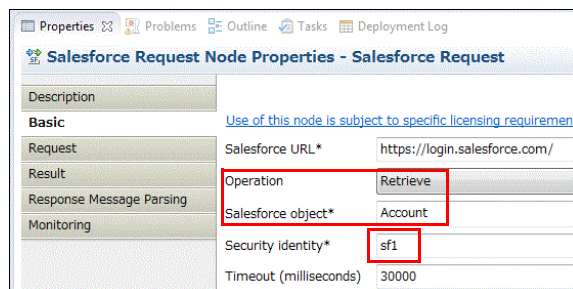


Figure 5-45 Salesforce Request node property

- The second message in the fan-out flow has the **productID** of line items and is used to query the product details in the Product Information System. This part is implemented at the `flow2_catalogservice.msgflow` in the `AggregationFlows` application (Figure 5-46).

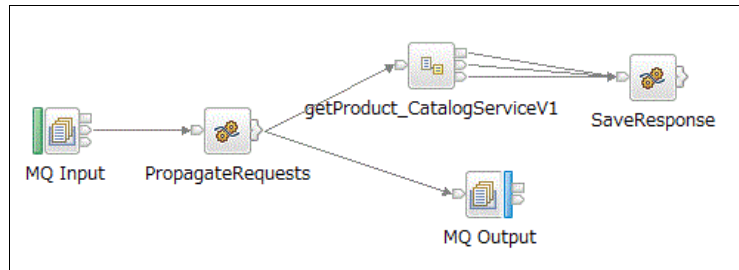


Figure 5-46 `flow2_catalogservice.msgflow`

- For the example scenario, the Product Information System is implemented as SOAP service within IBM Integration Bus (Figure 5-47), but it could be any external system that provides a SOAP-based service.

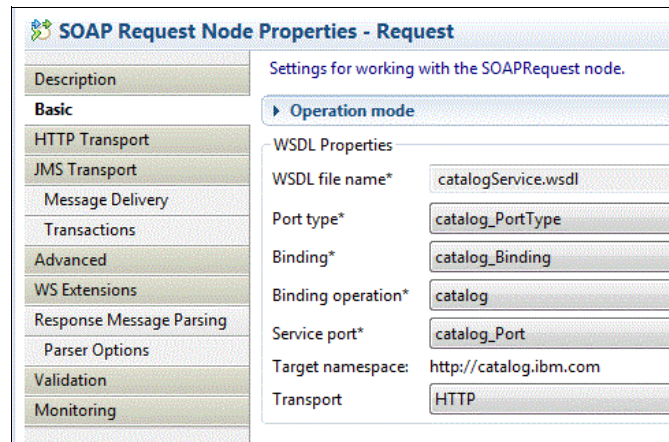


Figure 5-47 SOAP Request Node Properties

- Aggregate two response messages and combine them into one response message, then return it to the client. See Figure 5-48.

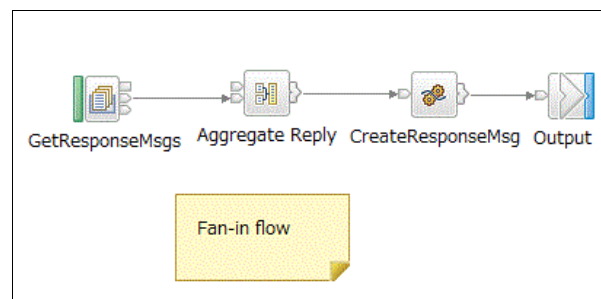


Figure 5-48 Aggregate two response messages

Similarly, you can review the `postOrder` and `deleteOrder` operations in the provided artifacts.

## Generating monitoring events for transactions

You can configure IBM Integration Bus to emit a monitoring event (an XML document) when something interesting happens. Events are typically emitted to support transaction monitoring, transaction auditing, and business process monitoring.

A monitoring event can also contain the following items:

- ▶ Application data that is extracted from the message.
- ▶ Part or all of the message bit stream. All nodes can produce bit streams, which can be included in monitoring events.

Events are published to a topic, where they can be read by multiple subscribers. The topic name has the following structure for events published by the IBM MQ publish/subscribe broker:

```
$SYS/Broker/integrationNodeName/Monitoring/integrationServerName/flow_name
```

In this scenario, configure the flow to emit the events when a request is made for *new order*, *cancel order*, or *retrieve the details of existing order*. These are three operations for this service. Therefore, configure the compute node in each of the subflow operations as shown in Figure 5-49.

Data location	
\$Environment/Variables/orderId	
\$Environment/Variables/OrderType	
\$Environment/Variables/campaignId	

Figure 5-49 Monitoring event configuration

The custom information that you would like to include in your event is defined under the Event Payload section. This scenario includes information regarding OrderId, OrderType, and campaignId as part of the event payload.

## Deploy the IBM Integration Bus BAR file to the Integration Server

You can deploy the IBM Integration Bus BAR file, SCENARIO1.bar, to the Integration Server by using the IBM Integration Bus web user interface (web admin GUI) on your web browser. The web user interface allows you to view and administer your integration node resources and perform monitoring tasks. To deploy it, complete these steps:

1. Before deploying the BAR file, set the credentials for Salesforce on the Integration node. Issue the following commands at your scenario1 directory on your Docker host computer. Run the following command in the Docker command window:

```
docker exec -it scenario1_iib-op_1 /bin/bash
```

2. Now issue the following IBM Integration Bus command:

```
mqsisetdbparms NODE1 -n salesforce::sf1 -u <UserId> -p <Password> -c <ClientIdentity> -s <ClientSecret>
```

The parameters are defined in Table 5-7.

**Note:** You can refer to 5.3.3, “Importing the Accounts data into the Salesforce developer organization” on page 117 for instructions. Note that when you create a connected app on Salesforce.com, you can see the ClientIdentity and ClientSecret as the Consumer Key and Consumer Secret.

Table 5-7 Parameters table

Parameter	Meaning
UserId	Your Salesforce user ID
Password	Your Salesforce password, suffixed with the security token that was sent to you by Salesforce
ClientIdentity	The name of the consumer key of your Connected App
ClientSecret	The consumer secret of your Connected App

3. Restart the Integration node by running these commands:

```
mqsistop NODE1
mqsistart NODE1
```

4. Run the **exit** command.

5. To deploy the BAR file to the IBM Integration Bus integration server, access the web user interface from your web browser:

```
http://<Docker host>:<Port>/
```

where:

- Docker host is the host name or IP address of your Docker host computer.
- Port is the Docker proxy port, which is configured as tcp/4414. You can check it by running the Docker command **docker-compose ps**. In Example 5-3, you can see that 32864 is the port that was used.

Example 5-3 The docker-compose ps command

```
[root@rb-docker-host-01 scenario1]# docker-compose ps
-----
Name                Command             State      Ports
-----
scenario1_iib-      iib_mq_manage.sh   Up         0.0.0.0:32865->1
op_1                414/tcp, 0.0.0.0
                   :32864->4414/tcp
```

```

, 0.0.0.0:32863-
>7800/tcp
scenario1_orderd /entrypoint.sh Up 0.0.0.0:32862->4
b_1 db2start 22/tcp, 0.0.0.0:
scenario1_sgc_1 node lib/secgwcl Up 32860->50000/tcp
ient.js 1X ...
scenario1_stub_1 iib_manage.sh Up 0.0.0.0:32862->4
414/tcp, 0.0.0.0
:32861->7800/tcp

```

You can see the integration node and integration server as shown in Figure 5-50.

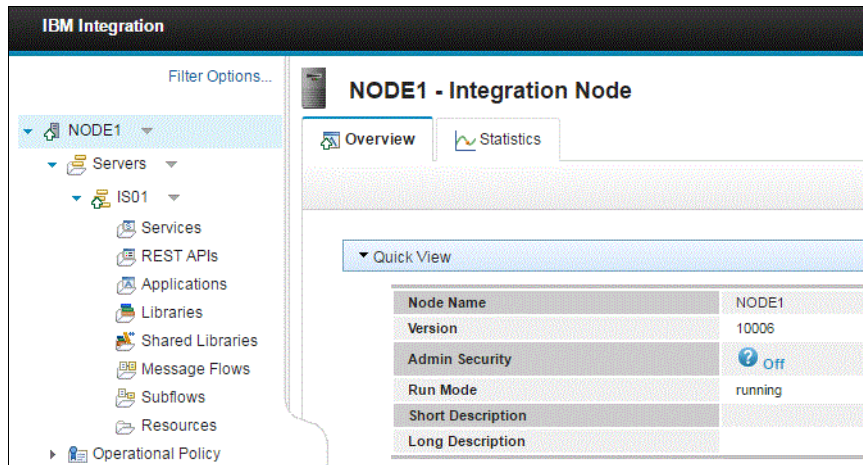


Figure 5-50 Integration Node details in WebUI

- Click the drop down menu icon next to the IS01 integration server and then click **Deploy** to deploy the BAR file to the server as shown in Figure 5-51.

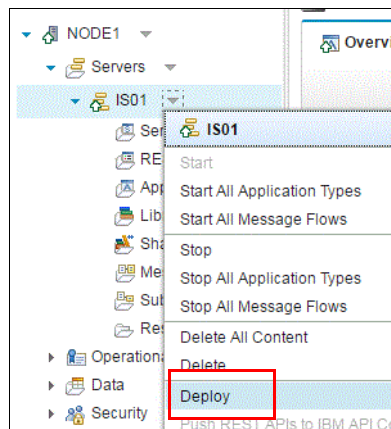


Figure 5-51 Deploy to integration Server

- The Deploy BAR File window is displayed. To select a BAR file, click **Browse**, navigate to your BAR file, and click **Open**. The BAR file properties and the associated values are displayed in the window as shown in Figure 5-52. Click **Deploy**.

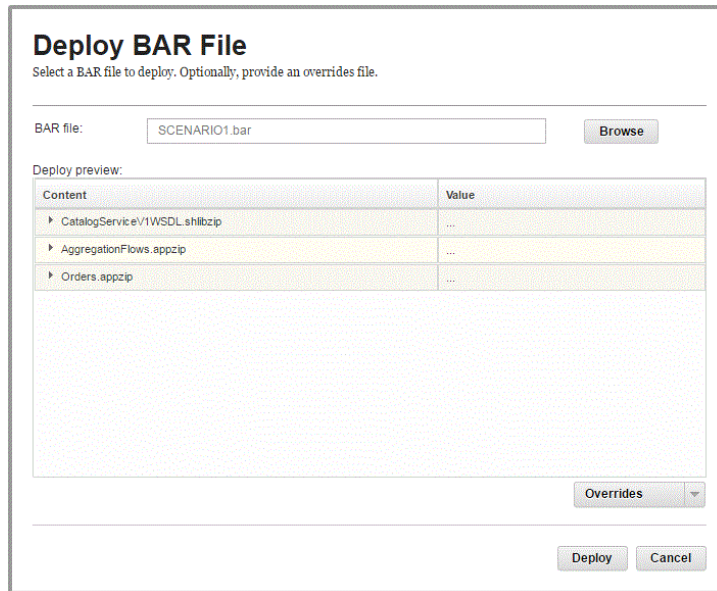


Figure 5-52 Deploy BAR File

In the web user interface navigator, the deployed resources are listed under the integration server as shown in Figure 5-53.

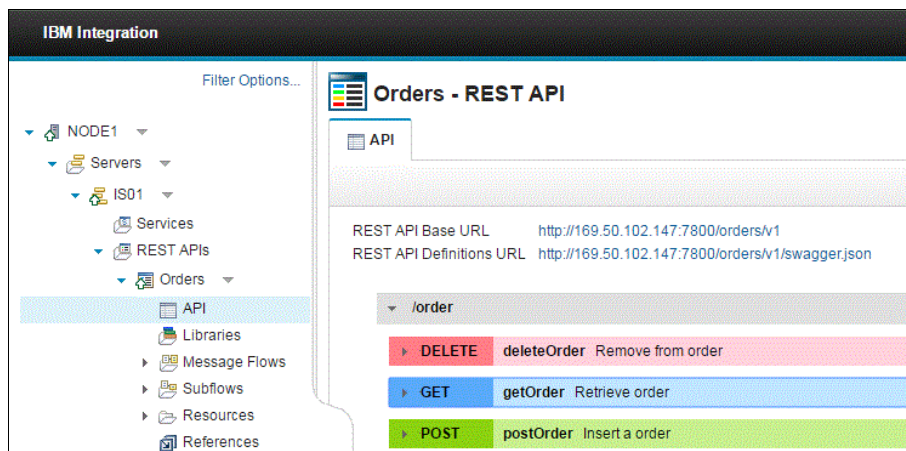


Figure 5-53 Deployed resources

- You can confirm the deployment by using a `curl` command. The following test data is provided in the GitHub repository:

[https://github.com/sg248351/scenario1/tree/master/2\\_testing](https://github.com/sg248351/scenario1/tree/master/2_testing)

- orderdata01.txt
- orderdata02.txt
- orderdata03.txt

- Download these files into your current directory.



10. To confirm the postOrder operation, issue the following command:

```
curl -X POST http://<Docker host>:<Port>/orders/v1/order -H  
"Content-Type:application/json" --data-binary @orderdata01.txt
```

where:

- <Docker host> is the host name or IP address of your docker host computer.
- <Port> is the docker proxy port, which is configured as tcp/7800. Example 5-3 on page 100 shows that 32863 is the port that was used.

If it succeeds, the result should look like Example 5-4.

*Example 5-4 Operation successful message*

```
{"result":"OK","orderId":1}
```

11. Confirm the getOrder operation by issuing the following command:

```
curl -X GET http://<Docker host>:<Port>/orders/v1/order?orderId=1
```

If it succeeds, the result should look like Example 5-5.

*Example 5-5 Output of the getOrder operation*

```
{"ORDERID":1,"ACCOUNTNAME":"AccountA","TOTAL":5.7E+2,"STATUS":"NEW","CREATIONDATE":"2016-10-01",  
"CAMPAIGNID":"Winter Sale","accountDetails":{"accountName":"AccountA","salesforceId":"0012800000  
qqyIcAAI","billingCountry":"Japan","billingPostalCode":"999999"},"lineItems":[{"PRODUCTID":12345  
,"QUANTITY":10,"PRICE":1.2E+1,"productName":"Drugstore Special","description":"Fever  
Pills","SKU":"1234-567-8","listPrice":"12"},{"PRODUCTID":12346,"QUANTITY":10,"PRICE":4.5E+1,"pro  
ductName":"Pharmacists'Delight","description":"Anti-Inflammatory","SKU":"1236-889-7","listPrice  
":"45"}]}
```

**Tip:** In Example 5-5, the salesforceId, billingCountry, and billingPostalCode fields come from Salesforce. If your result does not have these data, check your Salesforce credentials settings.

Now the CompanyA's order system is created by integrating information from multiple systems, and exposed as REST APIs in the IBM Integration Bus.

## Push the API definitions to API Connect

The integration between IBM Integration Bus and API Connect means that you can push APIs from IBM Integration Bus to API Connect. This configuration allows application developers to easily consume them, while making it easy for the API provider to secure and monitor them.

Complete the following steps to push the REST APIs from IBM Integration Bus to IBM API Connect on Bluemix:

1. Create a destination on the Secure Gateway to enable IBM API Connect on Bluemix to access IBM Integration Bus on-premises. You can use the gateway *CompanyA\_Secure\_Gateway* that you created in "Configure the Bluemix Secure Gateway service" on page 78, and create a destination in the same way. At the **Advanced Setup** stage, enter the values as shown in Table 5-8.

*Table 5-8 Configuration table*

Field	Value
Name	Order API
Resource Hostname	iib-op

Field	Value
Resource Port	7800
Protocol	TCP

- Find the cloud host and port by clicking the **Cogwheel** icon of the destination. See Figure 5-54.

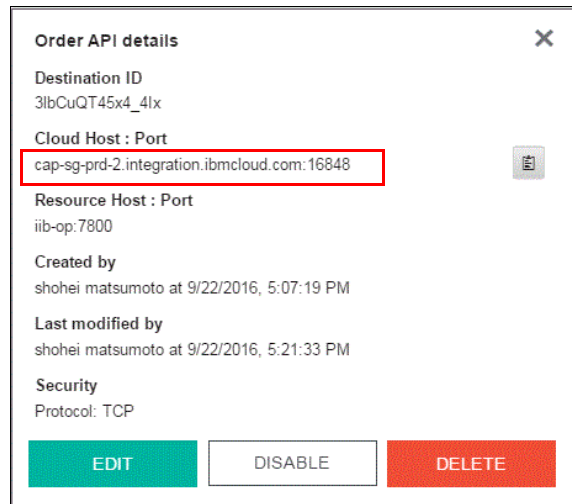


Figure 5-54 Cloud host and port

- In the web user interface, click the arrow next to the IS01 integration server, and then click **Push REST APIs to IBM API Connect** (Figure 5-55).

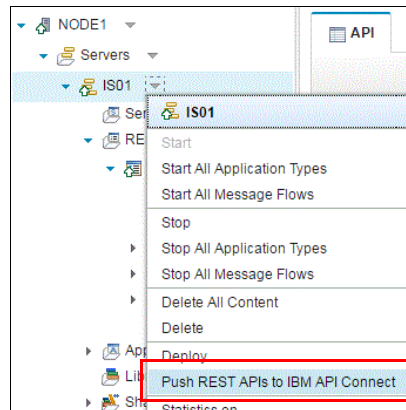


Figure 5-55 Push REST APIs to API Connect

- The Push REST APIs to IBM API Connect window is displayed, in which you define your connection to the API Connect instance in your Bluemix organization. Enter the connection details for your IBM API Connect server in the **Host** and **Port** fields. The Host depends on which region of Bluemix you are using (Table 5-9). The port is 443.

Table 5-9 API Connect server host

Bluemix Region	Host
US South	us.apiconnect.ibmcloud.com
Sydney	au.apiconnect.ibmcloud.com
United Kingdom	eu.apiconnect.ibmcloud.com

- Enter your Bluemix credentials in the **User ID** and **Password** fields, and click **Connect to IBM API Connect**. See Figure 5-56.

Figure 5-56 Configure connection to an API Connect cloud

- When the connection is established, you can see the message shown in Figure 5-57. Click **Next**.

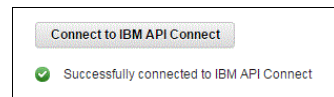


Figure 5-57 Successful connection

7. Select the target organization, and specify the name and version of the Product that you want to create. Select the **Sandbox** catalog. See Figure 5-58.

Figure 5-58 Specify target details for pushing REST API

8. Click **Next**. A window is displayed (see Figure 5-59) in which all available APIs are listed. Select the **Orders** REST API and then click **Next**.

REST API Name	Title	Version
Orders	Orders	1.0.0

Figure 5-59 Select the API to be pushed

9. You need to override the host name and port that are used by IBM API Connect to start the APIs (Figure 5-60). The host name and port are provided by the Secure Gateway service.

Figure 5-60 Override host and port for the APIs on IBM Integration Bus

10. Click **Push to IBM API Connect**. If it is successful, you can see the window shown in Figure 5-61. Click **Close**.

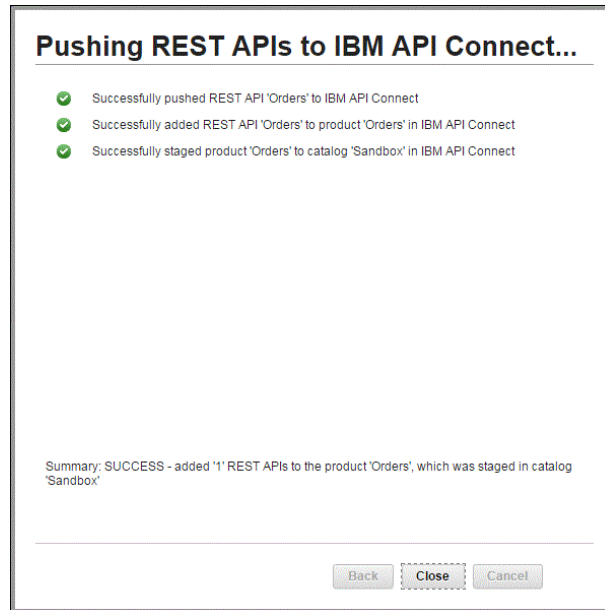


Figure 5-61 Success message after Pushing API

Pushing the API Product and the API stages the product in the **Sandbox** catalog. Staging means that the product is deployed to a catalog. However, it is not yet visible in the developer portal and cannot be subscribed yet. To make it visible, the product must be published.

11. Log in to the API Manager interface for your Bluemix API Connect instance again, and go to the dashboard (see Figure 5-62).

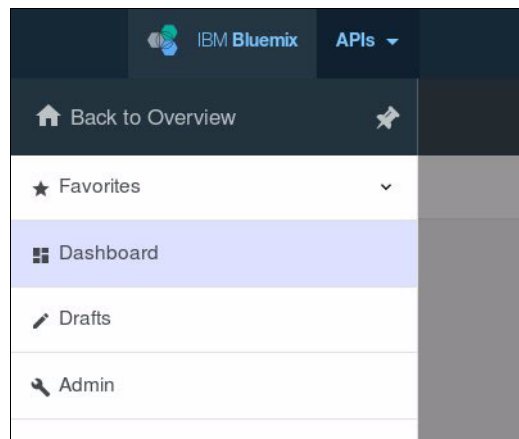


Figure 5-62 Go to the API Connect Dashboard

12. Click the **Sandbox** catalog, and click the Products tab. You will see that the **Orders** product, which was pushed from IBM Integration Bus, is in the Staged state.

- Click the **Manage** icon (...) next to the product and select **Publish**, as shown in Figure 5-63.

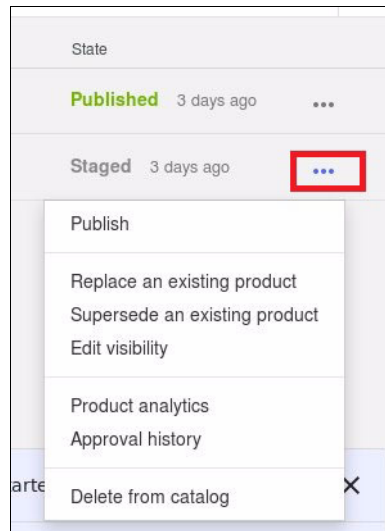


Figure 5-63 Publish a staged product in a catalog

- Accept the default visibility settings, and click **Publish**. The product is now published in the **Sandbox** catalog. Developers are able to see the product, and can subscribe to it, to call the included API from their applications.
- Test the solution by using the open source tool cURL. Example 5-6 through Example 5-10 on page 109 show testing the Create order, Get order, Cancel order functions.

*Example 5-6 Test date (orderdata01.txt)*

---

```
{
  "accountName": "AccountA",
  "lineItems": [
    {"productId": 12345, "quantity": 10, "price": 12},
    {"productId": 12346, "quantity": 10, "price": 45}
  ],
  "total": 570,
  "status": "NEW",
  "creationDate": "2016-10-01",
  "campaignId": "Winter Sale"
}
```

---

Example 5-7 shows testing Create order.

*Example 5-7 Create order*

---

```
curl -X POST <API Endpoint Base URL>/orders/v1/order -H
"Content-Type:application/json" --data-binary @orderdata01.txt

result:
{"result": "OK", "orderId": 1}
```

---

**Note:** You can find <API Endpoint Base URL> in Figure 5-9 on page 75.

Example 5-8 shows testing Get order.

*Example 5-8 Get order*

---

```
curl -X GET <API Endpoint Base URL>/orders/v1/order?orderId=1
```

```
result:
{"ORDERID":1,"ACCOUNTNAME":"AccountA","TOTAL":5.7E+2,"STATUS":"NEW","CREATIONDATE":"2016-10-01","CAMPAIGNID":"Winter Sale","accountDetails":{"accountName":"AccountA","salesforceId":"0012800000qqyIcAAI","billingCountry":"Japan","billingPostalCode":"999999"},"lineItems":[{"PRODUCTID":12345,"QUANTITY":10,"PRICE":1.2E+1,"productName":"Drugstore Special","description":"Fever Pills","SKU":"1234-567-8","listPrice":"12"},{"PRODUCTID":12346,"QUANTITY":10,"PRICE":4.5E+1,"productName":"Pharmacists' Delight","description":"Anti-Inflammatory","SKU":"1236-889-7","listPrice":"45"}]}
```

---

Example 5-9 shows testing Cancel order.

*Example 5-9 Cancel order*

---

```
curl -X DELETE <API Endpoint Base URL>/orders/v1/order?orderId=1
```

```
result:
{"result":"OK"}
```

---

The cancel order operation changes the status of the order. You can see that the status has changed by getting the order again as shown in Example 5-10.

*Example 5-10 Get order again*

---

```
curl -X GET <API Endpoint Base URL>/orders/v1/order?orderId=1
```

```
result:
{"ORDERID":1,"ACCOUNTNAME":"AccountA","TOTAL":5.7E+2,"STATUS":"CANCELED","CREATIONDATE":"2016-10-01","CAMPAIGNID":"Winter Sale","accountDetails":{"accountName":"AccountA","salesforceId":"0012800000qqyIcAAI","billingCountry":"Japan","billingPostalCode":"999999"},"lineItems":[{"PRODUCTID":12345,"QUANTITY":10,"PRICE":1.2E+1,"productName":"Drugstore Special","description":"Fever Pills","SKU":"1234-567-8","listPrice":"12"},{"PRODUCTID":12346,"QUANTITY":10,"PRICE":4.5E+1,"productName":"Pharmacists' Delight","description":"Anti-Inflammatory","SKU":"1236-889-7","listPrice":"45"}]}
```

---

In this section, you performed the following activities:

- ▶ IBM Integration Bus Step 1
- ▶ IBM Integration Bus Step 2
- ▶ Published the API pushed from IBM Integration Bus to an API Connect catalog

## 5.2.4 Create a Production catalog with an IBM Developer Portal Site

In this section, you create a catalog and configure a Developer Portal Site, from which application developers can browse the API products that are published to this catalog. To do so, complete these steps:

1. Go to the dashboard again in the API Manager interface, and click **Add** to for a new catalog as shown in Figure 5-64.

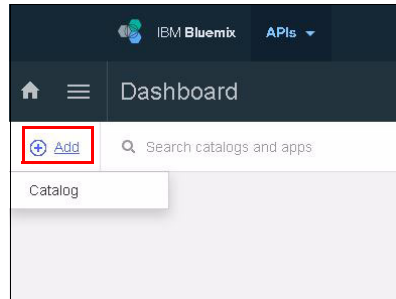


Figure 5-64 Add a catalog to an API Connect instance

2. Name the catalog **Production**, and click **Add**. See Figure 5-65.

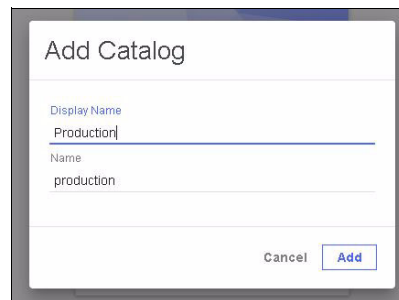
The image shows a modal window titled 'Add Catalog'. It contains two text input fields. The first field is labeled 'Display Name' and contains the text 'Production'. The second field is labeled 'Name' and contains the text 'production'. At the bottom right of the form, there are two buttons: 'Cancel' and 'Add'.

Figure 5-65 Add a Production catalog to API Connect



3. In the new **Production** catalog, click the **Settings** tab, and click the **Portal** settings. Select the **IBM Developer Portal** and select **IBM ID** as the User Registry. An example is shown in Figure 5-66.

Portal Configuration

None

IBM Developer Portal

Portal URL:

Thank you for enabling the IBM Developer Portal. Please note that due to the time taken for the DNS definition

Other

URL

User Registration and Invitation

User Registry

IBM ID

Figure 5-66 Portal Configuration window

4. Click **Save**, and wait for a confirmation email. It will take some time to set up the Developer Portal site, and to propagate the DNS change across the internet.
5. In the API Manager interface, click the Developers tab, and click **Add Bluemix Organization**. See Figure 5-67.

Figure 5-67 Add a Developer organization to the catalog

6. Enter your Bluemix email address and click **Add**, as shown in Figure 5-68.

Add Organization

Bluemix user email address

Cancel Add

Figure 5-68 Enter your Bluemix details

7. Wait for a confirmation email, which looks like Figure 5-69, and click the link in the email. If you were not logged in to Bluemix yet, you must provide your Bluemix credentials to continue.



Figure 5-69 Bluemix Developer organization confirmation email

8. Select your Bluemix organization, and click **CONFIRM** as shown in Figure 5-70.

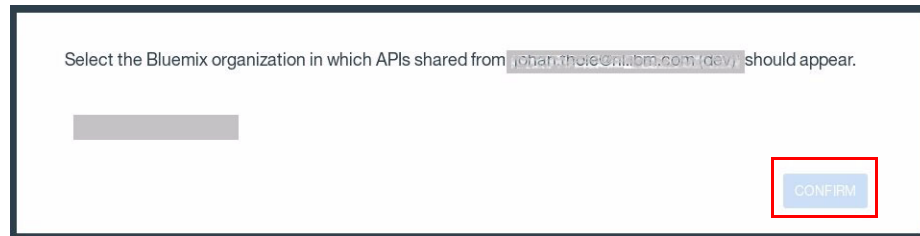


Figure 5-70 Confirm your Bluemix organization

Now that the Developer Portal site is configured, and a Developer organization created, you can subscribe to products, which are published in the catalog. From within the API Manager interface, you can stage and publish products from the Drafts space.

9. Go to the Drafts space in the API Manager interface. Select the **CompanyA Products product** and click the **Stage** icon, as shown in Figure 5-71. Select the **Production catalog**.

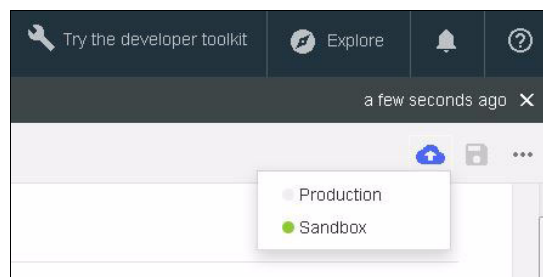


Figure 5-71 Stage a product to a catalog

10. In the navigation pane of the API Management interface, go to the Dashboard and click the **Production** catalog. Make sure that you are in the Products tab, and click the (...) icon behind the Staged **CompanyA Products** product. Click **Publish** and leave the visibility settings at the default. Click **Publish** to publish the product to the **Production** catalog.

11. In your web browser, go to the **Developer Portal** site. You can find the link in the Portal settings of the **Production** catalog. The Developer Portal site home page looks like Figure 5-72.

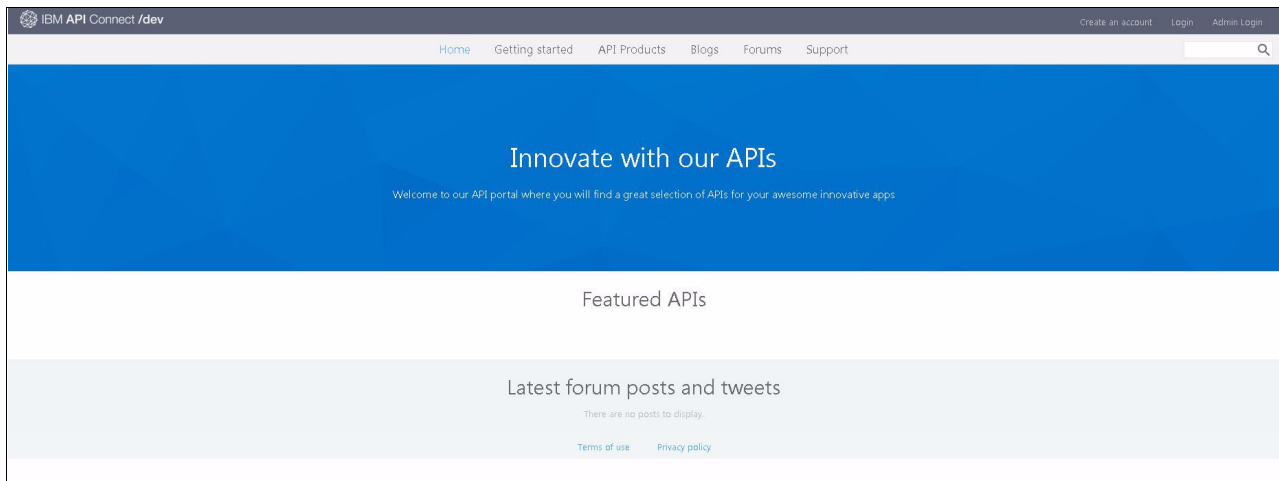


Figure 5-72 API Connect IBM Developer Portal site

12. Log in to the Developer Portal with your Bluemix credentials.
13. Click the Apps tab to create an application, and select **Register new application**.
14. In the example shown in Figure 5-73, OrderViewer was entered as the title for the app. Click **Submit** to create the application.

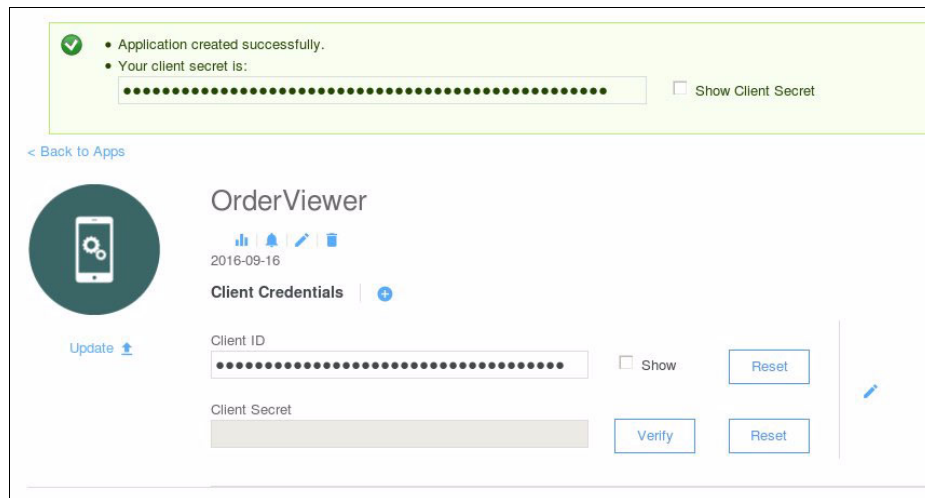


Figure 5-73 Creating the application

15. In the Developer Portal site, click the API Products tab to show which products were published to the catalog. You can use the APIs included in these products within your apps after you subscribe to them. The list of products should look similar to Figure 5-74.

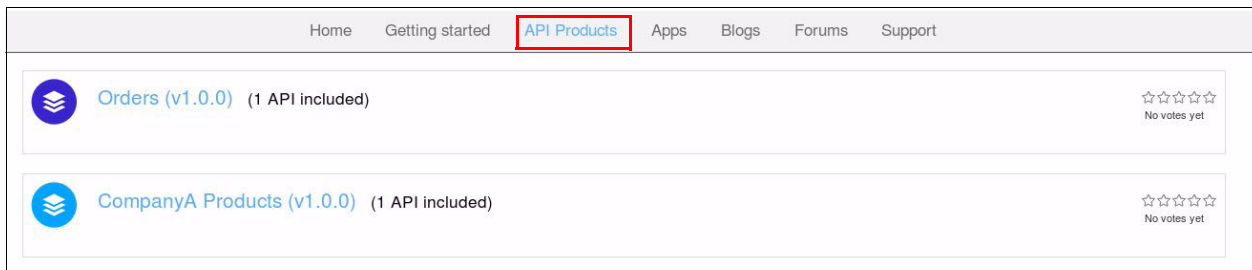


Figure 5-74 API Products published in the Production catalog

16. Select the **CompanyA Products API** product from the list of products. A window opens with the details about the product, including the contained APIs and Plans. Use the default plan in this scenario. Click **Subscribe** in the **Default** plan. Select the **OrderViewer** application as shown in Figure 5-75.

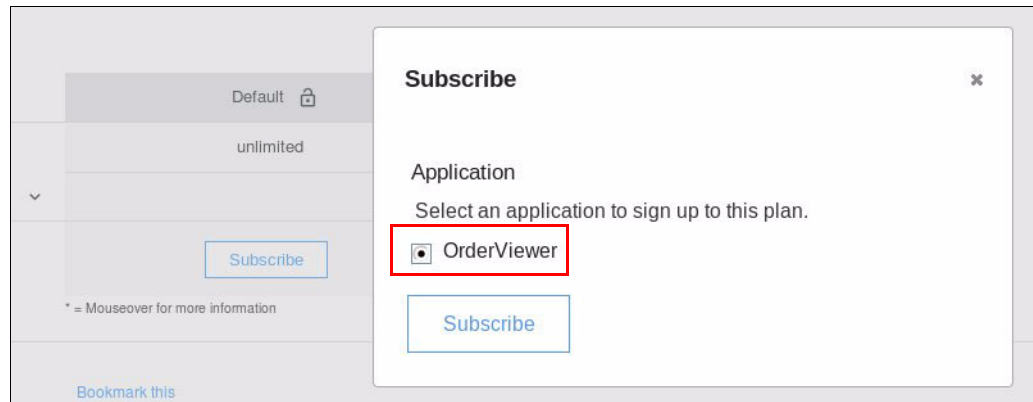


Figure 5-75 Subscribe an application to a Plan

17. In the same page, click the **products** API within the **CompanyA Orders** product, as shown in Figure 5-76.

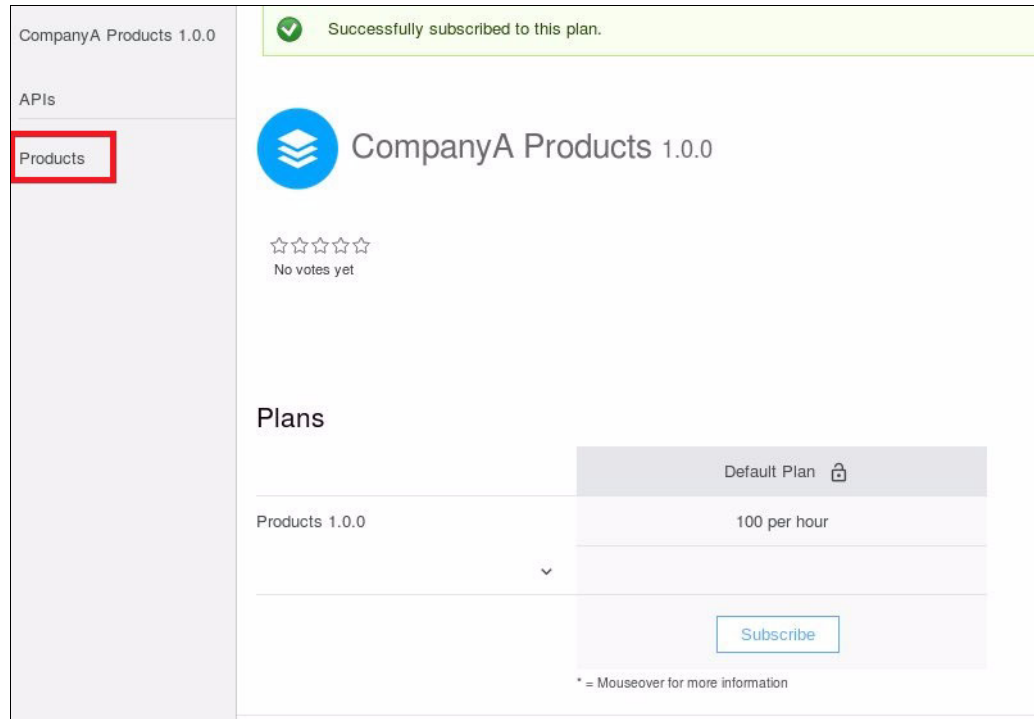


Figure 5-76 Select the Products API that was created earlier

Information about the API, including code examples, is displayed as shown in Figure 5-77.

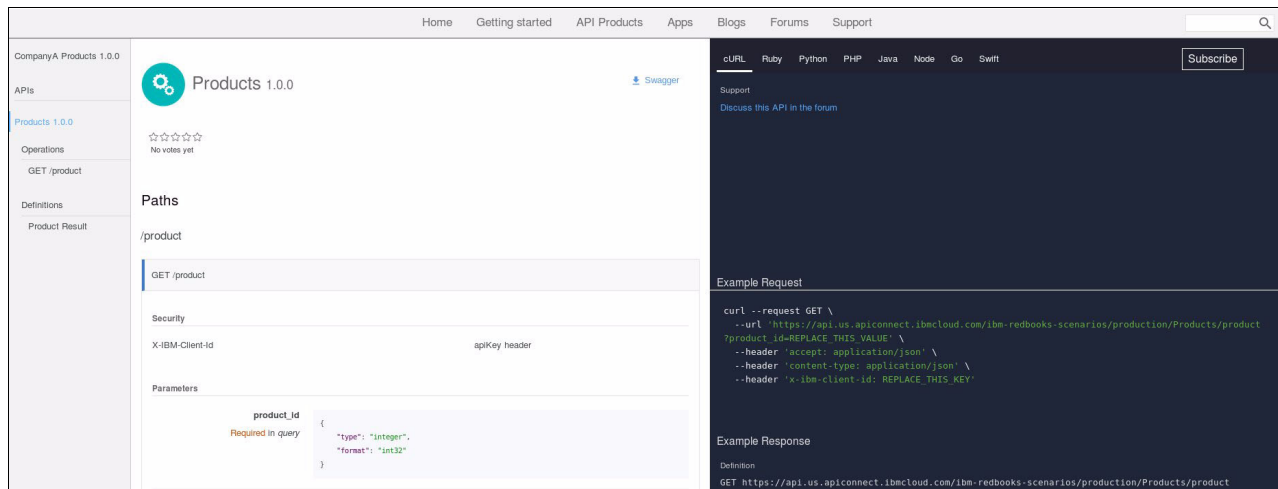


Figure 5-77 Products API details in the Developer Portal

18. Select the **GET /product** REST operation, as shown in Figure 5-78.

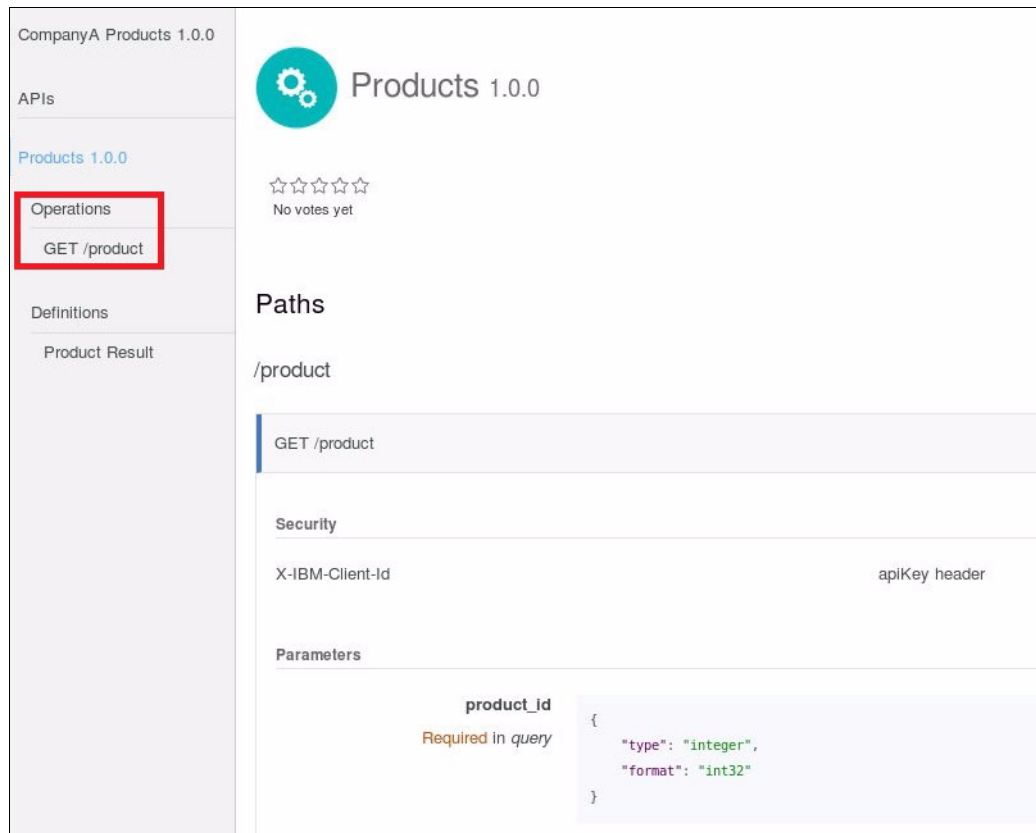


Figure 5-78 Show the GET /product operation

19. Scroll down to the Developer Portal test tool, labeled **Try this operation**, as shown in Figure 5-79.

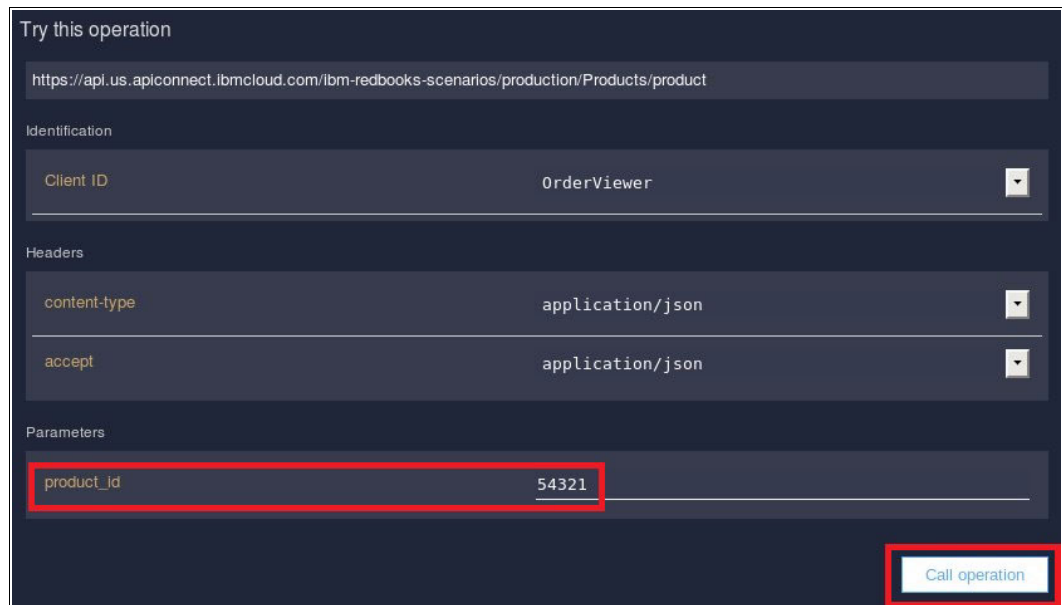


Figure 5-79 Use the Developer Portal test tool

20. Provide a value for the **product\_id** operation, and click **Call operation**. The response, including the protocol headers, is displayed in the window.

In this section, you performed the following activities:

- ▶ Configured a new catalog in API Connect.
- ▶ Configured the Developer Portal settings in the catalog.
- ▶ Published an API Product to the catalog.
- ▶ Used the Developer Portal to subscribe an app to a published API.
- ▶ Tested the API definition from the Developer Portal.

## 5.3 Resources

This section lists several resources that can help when implementing this scenario.

### 5.3.1 Sign up for an IBM Bluemix account

If you do not have an IBM Bluemix account, you can sign up for a 30 days trial by using the following link:

<https://console.ng.bluemix.net/registration/>

Your 30-day trial is available at no charge, with no credit card required. You get access to 2 GB of runtime and container memory to run apps, unlimited IBM services and APIs, and complimentary support.

To start the IBM Bluemix console and log in, use the following link:

<https://new-console.ng.bluemix.net/#overview>

### 5.3.2 Setting up a Salesforce developer edition account

Because this scenario explores using the Salesforce request node in IBM Integration Bus to interface with Salesforce, you need a Salesforce Developer edition account to complete the scenario. You can sign up for the Salesforce Developer edition by using the following link:

<https://developer.salesforce.com/signup>

Before you can connect to Salesforce from an IBM Integration Bus node, you must set up the environment. The required steps are documented in the IBM Knowledge Center for IBM Integration Bus 10:

[https://www.ibm.com/support/knowledgecenter/SSMKHH\\_10.0.0/com.ibm.etools.mft.doc/bz90640\\_.htm](https://www.ibm.com/support/knowledgecenter/SSMKHH_10.0.0/com.ibm.etools.mft.doc/bz90640_.htm)

### 5.3.3 Importing the Accounts data into the Salesforce developer organization

For the Salesforce integration, you must create the Salesforce Account records from a CSV file. You can retrieve the `customers.csv` file from the GitHub repository for this book:

<http://github.com/sg248351>

If you prefer to create the CSV file yourself, you can use the following values:

```
accountName,billingCountry,billingPostalCode
AccountB,India,887777
AccountA,Japan,999999
AccountC,USA,888888
AccountD,Germany,22335
```

To import the data into Salesforce from the customers.csv file, follow the instructions that are provided by Salesforce.com:

[https://help.salesforce.com/apex/HTViewHelpDoc?id=import\\_with\\_data\\_import\\_wizard.htm&language=en\\_US](https://help.salesforce.com/apex/HTViewHelpDoc?id=import_with_data_import_wizard.htm&language=en_US)

Feel free to create additional Account records in Salesforce as well.





## Automation for business users

This chapter describes the implementation for the scenario outlined in “Automation for business users” on page 58.

The chapter has the following main sections:

- ▶ Solution outline
- ▶ Implementation
- ▶ Resources

## 6.1 Solution outline

The main focus of the scenario is on the automation component and how it can be used to allow business users, in this example CompanyA's Marketing team, to realize tangible benefits from integration use cases that they can implement themselves or with limited support from CompanyA's integration team.

### 6.1.1 Functional overview

Figure 6-1 provides an overview of the logical components and the functions that are implemented in this scenario.

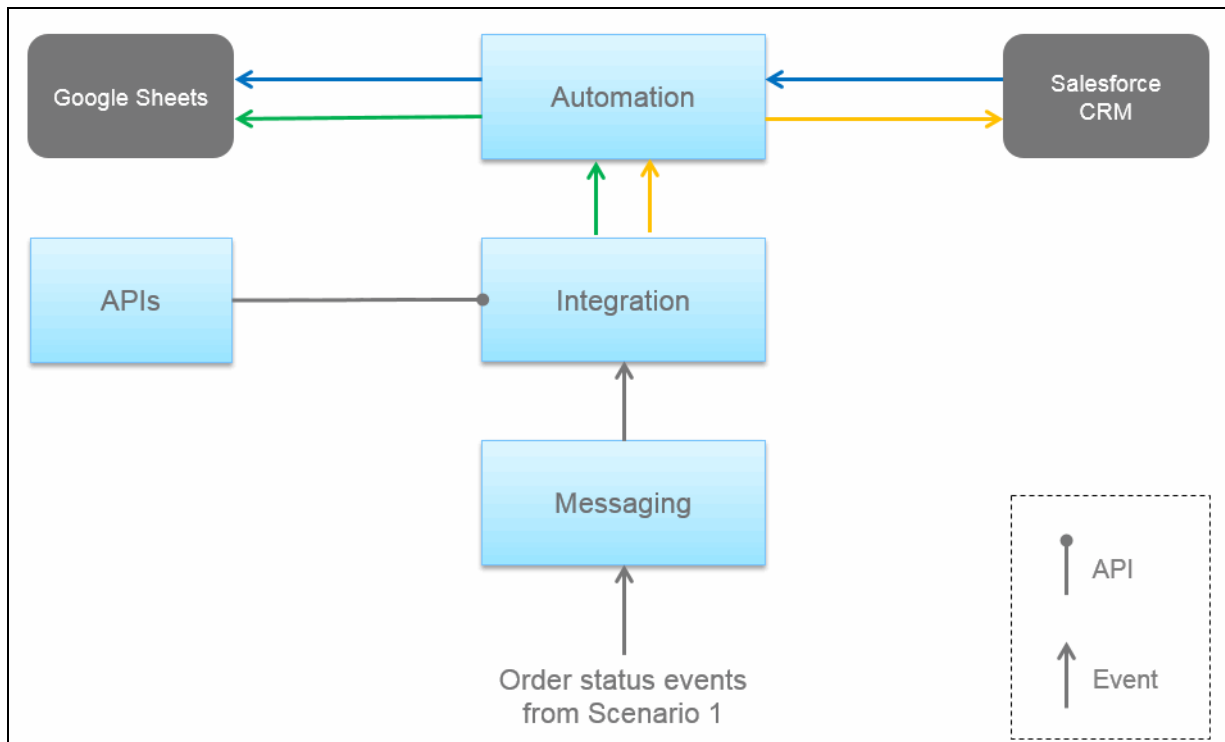


Figure 6-1 Functional overview diagram

From a high level, there are two main parts to this scenario:

- ▶ Synchronize campaign information between Salesforce and Google Sheets (as illustrated by the blue arrows).
- ▶ Enable additional business value by using custom triggers based on events from in-house (backend) systems (as illustrated by the yellow and green arrows).

The business context for each of these parts is described in more detail below.

#### Keeping the campaign overview spreadsheet up to date

As described earlier, one of the activities that the marketing team are responsible for are campaigns. In the past, all campaigns were managed manually with a spreadsheet. Although the team now uses Salesforce CRM to create campaigns, they still use the spreadsheet to keep an overview and history of all marketing campaigns. At the moment, this spreadsheet is maintained manually which, while not much work at any one time, is a tedious task and prone to error. In short, it is a perfect candidate for automation.

This part of the scenario describes the steps that a non-technical professional would perform to create an integration between Salesforce CRM and Google Sheets so that for each new campaign that is created in Salesforce, a line with the basic campaign data gets added to the Google Sheets spreadsheet. As a result, the spreadsheet is always up to date with the correct information straight out of Salesforce CRM.

### **Tracking the success of a campaign**

Following on from the integration between Salesforce CRM and Google Sheets described above, the Marketing team also needs an easy way of tracking the success of marketing campaigns and specifically the order value that got generated in response to each campaign. Each campaign has a unique ID that the partners need to use to receive any special campaign-related conditions such as discounts. This ID can be used to correlate individual purchase orders to the different marketing campaigns.

To make the necessary purchase order details available for the Marketing team, an integration needs to be set up with the order backend system. The respective tasks for this part of the scenario fall to the Integration team. They use IBM Integration Bus to expose campaign-related order updates as a trigger for use by the Marketing team. The Marketing team can then use this trigger in the same way as they have already used a trigger for a new campaign in Salesforce CRM. Any technical details like protocol conversion, enrichment, or connectivity settings necessary to source the information from the backend are hidden from the Marketing team.

The Marketing team then integrates the campaign-order updates into the same Google Sheets spreadsheet that already contains the campaign data. For every campaign-related order, a row is inserted in a separate tab in the spreadsheet. A formula automatically sums up all orders that are related to a specific campaign, providing the team with a quick and up-to-date overview without having to query the information in the order management system or run reports.

### **Automated follow-up for canceled orders**

Because the scenario already deals with purchase order updates, CompanyA decided to capitalize on those events and improve the follow-up on cancellations. Whenever a purchase order is canceled, a follow-up action in form of a case is created in Salesforce CRM. The intention is that the Sales team will then use these cases to determine what the reason for the cancellation is and whether there is a problem that could potentially be fixed.

This integration is another good example of a low-complexity integration candidate that would not have received enough attention for a formal development project because its business benefits would be seen as either too small or too uncertain to warrant the effort.

## 6.1.2 Technical overview

Figure 6-2 illustrates a possible technical implementation of this scenario in a production environment.

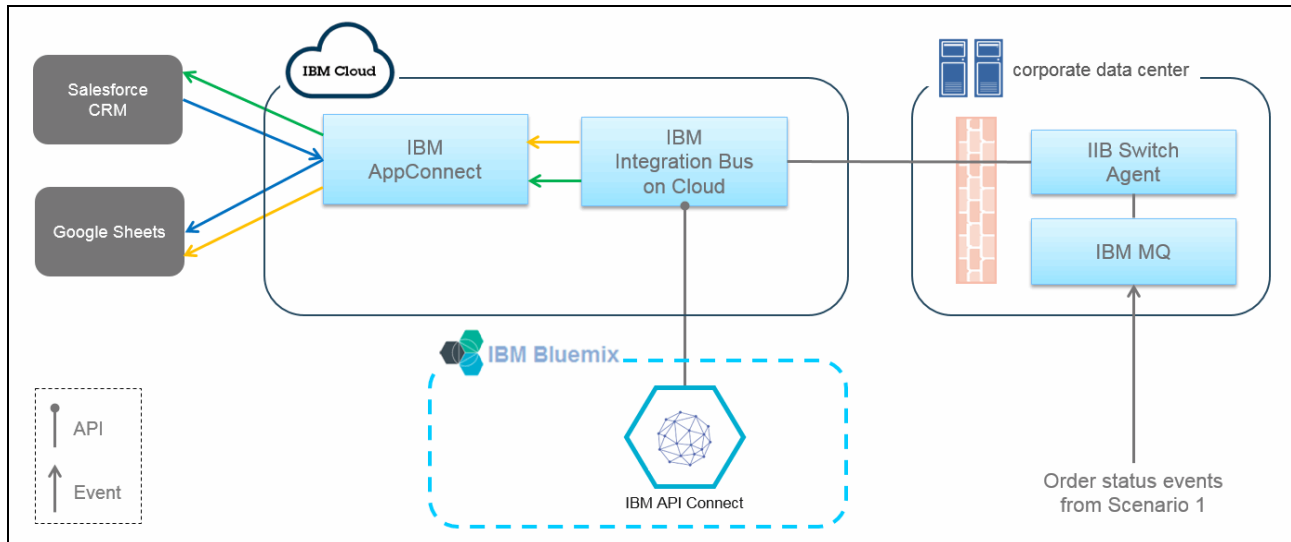


Figure 6-2 Technical overview

### Technical components

This section describes the technical components shown in Figure 6-2.

The Automation capability is realized with IBM App Connect, which is used in all parts of the scenario.

IBM Integration Bus on Cloud realizes the Integration capability and is used to perform the following activities as part of this scenario:

1. Enriching and formatting the raw order update events

The data portion of the event message varies. For example, a cancel event merely has the order number and the order status as canceled. IBM Integration Bus uses the Orders API discussed in scenario 1 to retrieve any missing order details before passing the data on to IBM App Connect.

2. Exposing the order update events as triggers into IBM App Connect

IBM App Connect realizes integrations by using triggers that are linked with actions. The arrival of an order update message is exposed by IBM Integration Bus as a trigger. As part of the scenario, two triggers are created. The first trigger is for campaign-related orders and is linked to an action to update Google Sheets (as illustrated by the yellow arrow in Figure 6-3 on page 123). The second trigger is for canceled orders, which then cause a new case to be created in Salesforce CRM (as illustrated by the green arrow).

3. Filtering and routing the order update events

Not all of the raw order events are usable for all triggers. Therefore, IBM Integration Bus is used to make sure that only the campaign-related order updates are passed to that trigger. Some events need to be routed to both triggers (the cancellation of a campaign order), and other unrelated event messages can be ignored (for example, order retrieval (read) events).

IBM API Connect exposes the Orders API described in scenario 1.

IBM MQ provides the Messaging capability and is the source of the raw order update events.

The IBM Integration Bus Switch Agent is a component of the on-premises installation of IBM Integration Bus. Its responsibility allows secure communication from IBM Integration Bus on Cloud to the IBM MQ Endpoint behind the firewall of the corporate data center.

## Simplifications

For this book, the setup for this scenario has been simplified as shown in Figure 6-3.

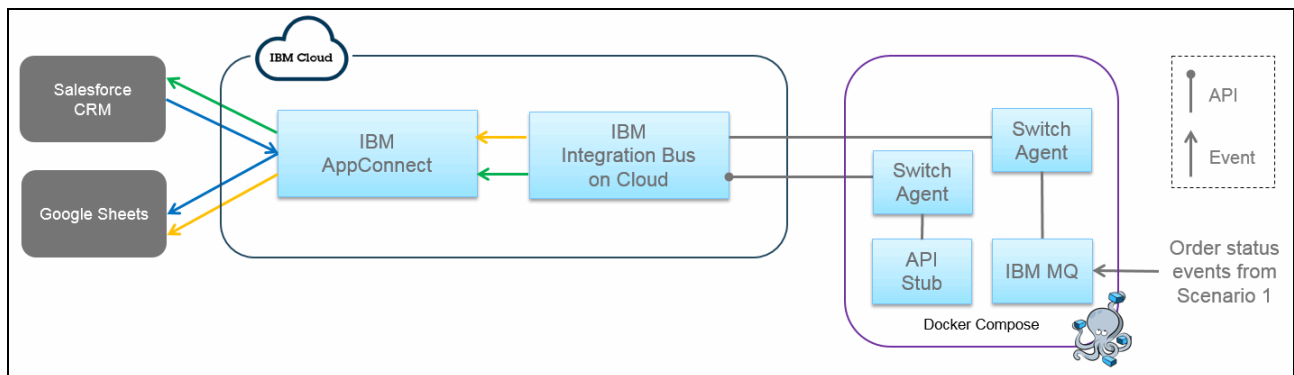


Figure 6-3 Simplified configuration

The first major simplification is that all the components from the corporate data center are simulated with a single docker compose configuration. This change allows you to concentrate on the automation aspects that are the focus of this scenario.

The other simplification is that instead of calling the Orders API exposed by using API Connect, that is included in the Docker compose setup. If you have already built the Orders API in scenario 1, feel free to reuse that. Otherwise, the instructions below use a stub that is part of the Docker compose configuration. The stub is exposed as a Callable Flow, which is an easy way to bridge the gap between IBM Integration Bus on-premises and in the cloud. This mechanism also uses a Switch agent configuration, just as the IBM MQ Endpoint configuration.

## 6.2 Implementation

This section covers the step-by-step implementation of this scenario.

### 6.2.1 Preparation

To re-create the implementation of the scenario, complete the following steps:

1. Download the resources that are used to run the Docker compose components.
2. Set up the accounts for the cloud-based components of the solution:
  - a. IBM App Connect
  - b. IBM Integration Bus on Cloud
  - c. Salesforce
  - d. Google

See 6.3, “Resources” on page 150 for the creation of a Salesforce.com Developer account.

To create a Google account, follow the instructions at this website:

<https://accounts.google.com/SignUp>

## 6.2.2 Creating the AppConnect flow to keep the campaign overview spreadsheet up to date

The instructions in this section allow you to create an IBM App Connect flow that creates a row in a Google sheet every time a new Campaign object is created in Salesforce.

### Build

To build an IBM App Connect flow, complete these steps:

1. Create a Google Sheet for the campaign data. You can create any spreadsheet layout that you want, but to enable auto matching field names from IBM App Connect, the structure in Table 6-1 is suggested.

Table 6-1 Google spreadsheet layout

Worksheets	Columns
Campaigns	Name Type Status Start Date End Date Expected Revenue Budgeted Costs Order Volume
Orders	Order Id Campaign Order Value

2. Create the IBM App Connect flow. For registration to IBM App Connect, see 6.3, “Resources” on page 150. Log in to IBM App Connect on <https://www.appconnect.ibmcloud.com>, and sign up for the service. After signing up, you will be able to start the IBM App Connect Free product.

To create a flow in IBM App Connect for updating a Google Sheet from Salesforce.com, you will need the following prerequisites:

- Your Salesforce.com developer account details
- Your Google account details
- An existing spreadsheet in Google Sheets

3. Click **Create a flow** to start creating a AppConnect flow.
4. Select **Salesforce** as the first application, and click **Save + Continue**.
5. Select **Google Sheets** as the second application, and click **Save + Continue** again.
6. Select **Trigger & Action** as the flow type. This flow causes an event in the first application to trigger a specific response in the second application.

7. After you click **Save + Continue**, your window should look like Figure 6-4. The Salesforce trigger that must be defined is for the **New Campaign** trigger.

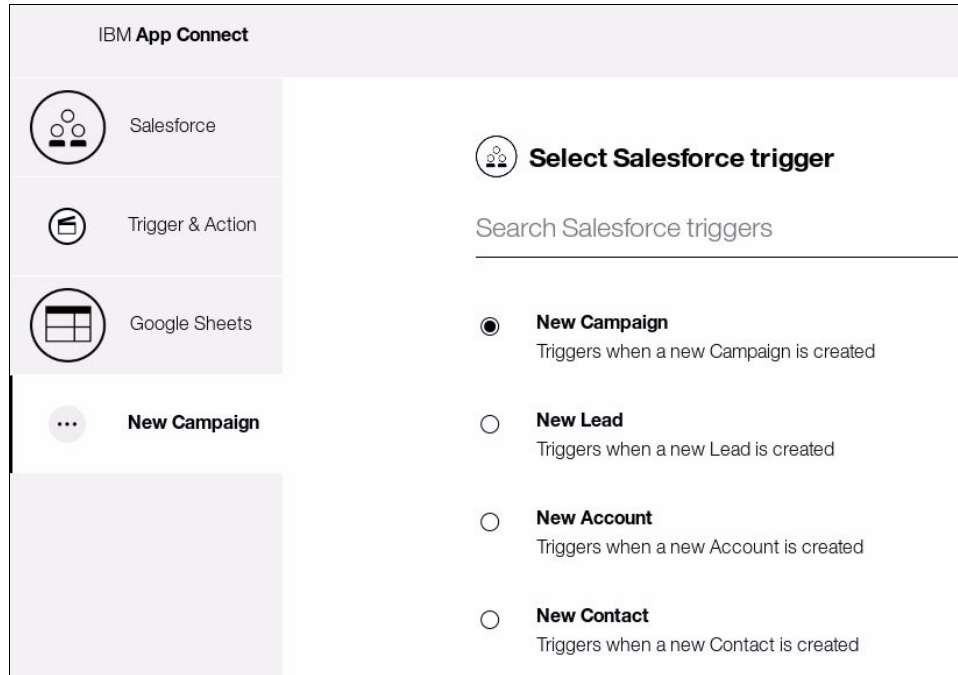


Figure 6-4 Salesforce trigger options

8. Click **Save + Continue** again, and select **Create Row** as the Google Sheets action. Click **Save + Continue**, and provide your Salesforce developer and Google accounts on the next window. If you have not configured your accounts in AppConnect yet, you can do so on the window that is shown in Figure 6-5.

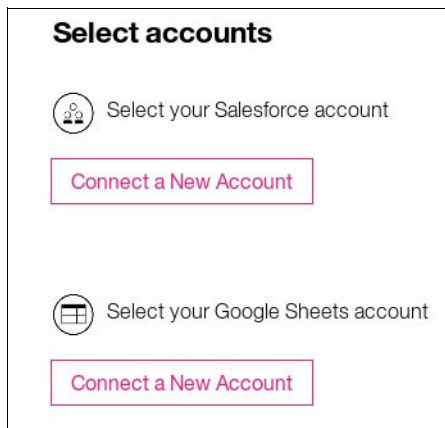
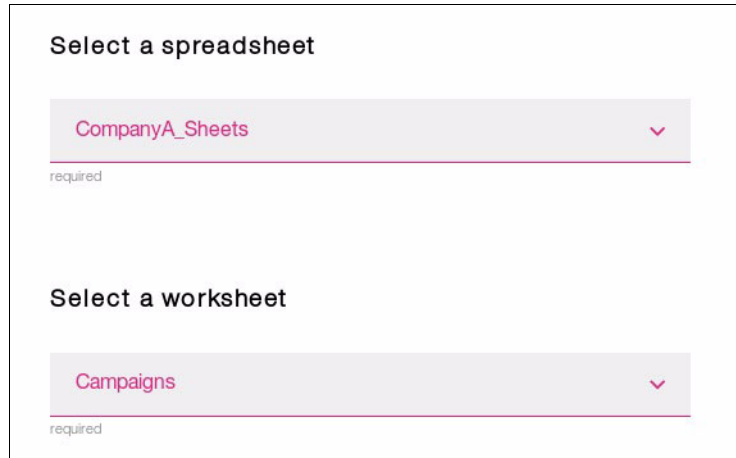


Figure 6-5 Select application accounts

9. When you click **Connect a New Account**, you have the opportunity to authorize AppConnect to use your existing Salesforce and Google accounts.

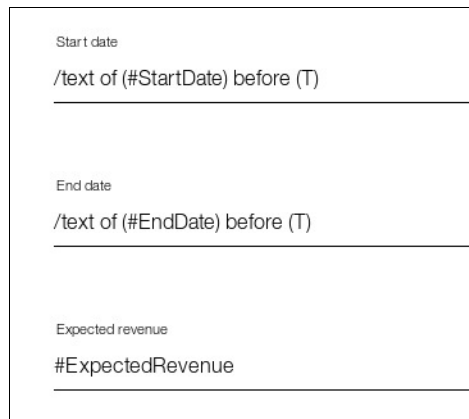
10. Click **Save + Continue** to set up a Google Sheets action. Select the spreadsheet and worksheet that you want to use as shown in Figure 6-6.



The screenshot shows a configuration interface with two sections. The first section, titled "Select a spreadsheet", contains a dropdown menu with "CompanyA\_Sheets" selected and a small downward arrow on the right. Below the dropdown is the word "required". The second section, titled "Select a worksheet", contains a dropdown menu with "Campaigns" selected and a small downward arrow on the right. Below this dropdown is also the word "required".

Figure 6-6 Select Google spreadsheet and worksheet

11. Click **Auto Match Fields** to automatically populate the Google spreadsheet column names with the matching Salesforce fields. However, truncate the date and time strings that are sent from Salesforce to only contain the dates. Therefore, modify the Start and End date columns as shown in Figure 6-7.



The screenshot displays three configuration rows. The first row is labeled "Start date" and contains the formula `/text of (#StartDate) before (T)`. The second row is labeled "End date" and contains the formula `/text of (#EndDate) before (T)`. The third row is labeled "Expected revenue" and contains the formula `#ExpectedRevenue`.

Figure 6-7 Use of functions for the date fields

12. For each row that is added to the spreadsheet, insert a formula in the **Order Value** column. To enable this process, insert the following text in the **OrderVolume** mapping field:

```
=SUMIF(Orders!B:B, INDIRECT(ADDRESS(ROW(), COLUMN()-7)), Orders!C:C)
```



- Click **Save + Continue** to finalize the flow, and to give it a name. For this scenario, use Salesforce Campaigns to Google, as shown in Figure 6-8.

**Finalize your flow**

Use the bar to the left to review your decisions. When satisfied, name and switch on your flow.

Name your flow

Salesforce Campaigns to Google

optional

Figure 6-8 Finalize and name the AppConnect flow

- Click **Switch on Flow** to save and activate the flow. The AppConnect dashboard is displayed, with the **Salesforce Campaigns to Google** flow switched to ON.

## Test

Testing the IBM App Connect flow that you just created consists of the following steps:

- Open the Google spreadsheet that you created, and leave it on the Campaigns worksheet. Because the spreadsheet is dynamically updated, any new rows are automatically added.
- Log in to Salesforce, and go to the Campaigns object. Click **New** to create a campaign. See Figure 6-9.

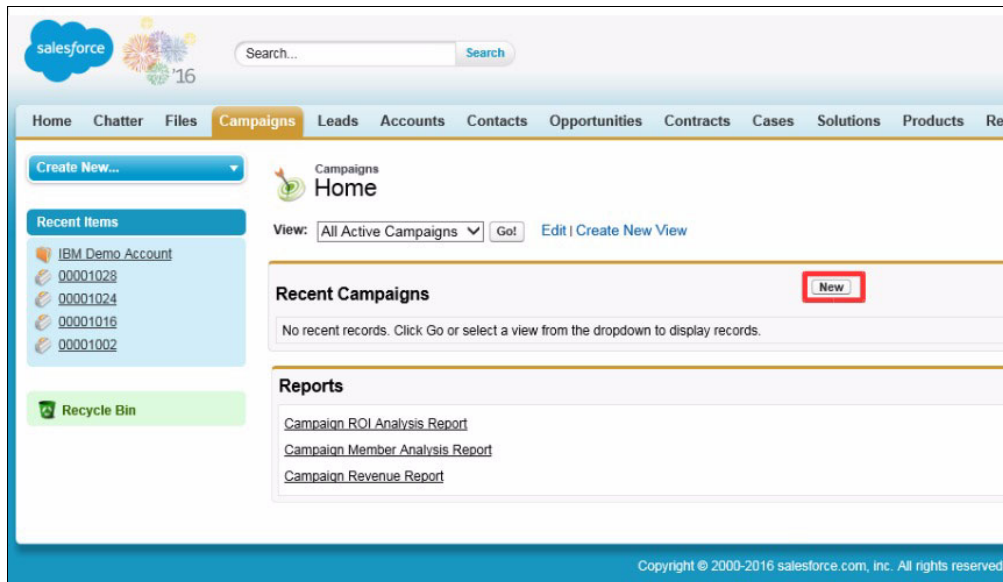


Figure 6-9 Create a Campaign in Salesforce

- Enter the values shown in Table 6-2.

Table 6-2 Campaign values

Field name	Value
Campaign name	Winter Sale
Type	Direct Mail
Status	Planned

Field name	Value
Start Date	November 1, current year
End Date	January 1, next year
Expected Revenue	10000
Budgeted Cost	100

- Click **Save** to add a new row to the Campaigns tab of the Google spreadsheet. See Figure 6-10.

	A	B	C	D	E	F	G	H
1	Name	Type	Status	Start date	End date	Expected revenue	Budgeted cost	Order Volume
2	Winter Sale	Direct Mail	Planned	2016-09-09	2017-01-01	1000	100	0

Figure 6-10 Added row for Winter Sale Campaign to Google sheet

### 6.2.3 Exposing the order update events as triggers into AppConnect

The diagram in Figure 6-11 provides an overview of the sequence of interactions that happens when an order update event is published from the Order Management system.

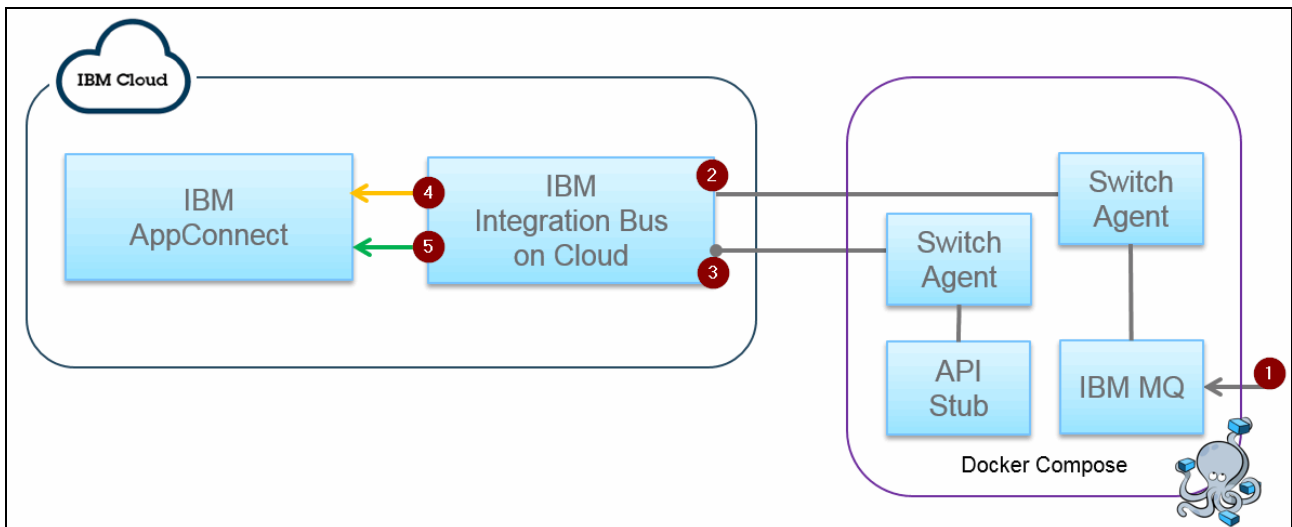


Figure 6-11 Publishing an order update event

The process involves these steps:

- An order update event arrives as a message in the IBM MQ container of the Docker compose environment.
- A message flow running on IBM Integration Bus on Cloud pulls the message from the queue.

The communication from the IBM Cloud to the container running on your notebook computer (or on-premises) is facilitated by an IBM Integration Bus Switch Agent.

- The message flow enriches the information from the event message by calling the Orders API stub running in another container in the docker-compose environment.

As before, the communication from the IBM Cloud to the local environment/containers is facilitated by an IBM Integration Bus Switch Agent.

4. Depending on the contents of the event, it is then published to one or both of the following triggers:
  - a. cancelOrder
  - b. campaignOrder

Apart from the on-premises connectivity, which is set up as part of the preparation steps in the next section, the bulk of the work is in the IBM Integration Bus message flow deployed on IBM Integration Bus on Cloud. The final flow looks like Figure 6-12.

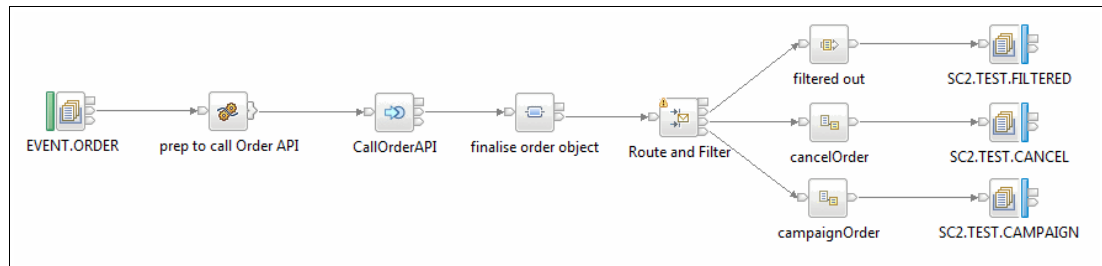


Figure 6-12 Flow overview

The process shown involves the following steps:

1. The raw order update event is read by using the `EVENT.ORDER` MQ Input node.
2. The `prep to call Order API` Compute node extracts all attributes necessary to call the Order API. Because this example simplifies the API call and uses a stub instead, those attributes are simply saved in the Environment for later use.
3. The `CallOrderAPI` is a Callable Flow Request node that is used to call the stub.
4. The `finalise order object` Mapping node processes the response from the API call and creates the JSON object to be published to the IBM App Connect triggers.
5. The `Route and Filter` Routing node makes sure that the correct events go to the correct triggers and unused events are filtered out.
6. The `cancelOrder` and `campaignOrder` nodes are subflows that implement the logic to publish to the IBM App Connect triggers.
7. The three `SC2.TEST.*` nodes are only there so that during testing you can see which messages have been processed and where they were sent. They would not be part of the production flow implementation.

## Prepare

Perform the following steps to prepare your environment:

1. If you have not already done so, create a directory to hold all the configuration data, for example `sg248351`, and open a shell or command prompt there.
2. Clone the GitHub repository for the scenario 2 by using the following command:

```
git clone https://github.com/sg248351/scenario2
```

As shown in Figure 6-11 on page 128, the communication to the IBM Cloud is facilitated through the IBM Integration Bus Switch Agents. This is the case for both the IBM MQ Endpoint and the stubbed API. Both sets of configurations settings are stored within the `agentp.json` and `agentx.json` configuration files. Examples are stored within the `99_docker\iib-sc2\agents` subdirectory.

## Setting up the IBM Integration Bus switch agents

These steps provide guidance on how to replace the examples with your own configuration:

1. Log in to IBM Integration Bus on Cloud.
2. Click **Endpoint Connectivity** from the left menu as shown in Figure 6-13.

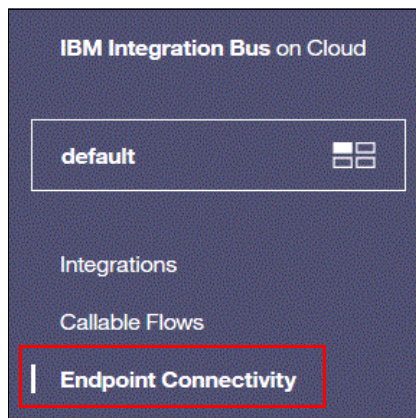


Figure 6-13 IBM Integration Bus menu

3. Select the type of endpoint that you want to connect to. For this scenario, select **MQEndpoint** as shown in Figure 6-14.

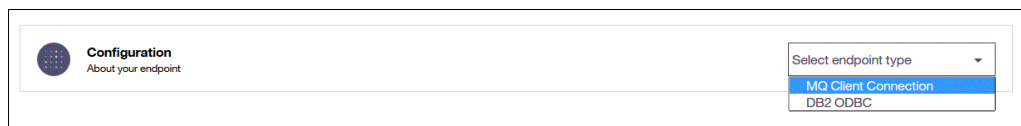
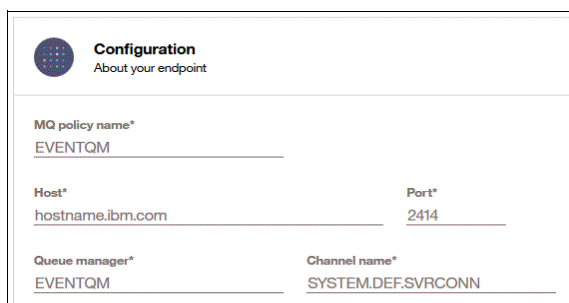


Figure 6-14 Select endpoint type

4. Configure the queue manager details as shown in Figure 6-15. To simplify the steps, IBM MQ Security has been disabled in the Docker container. Therefore, no user name or password are necessary.

The image shows the 'Configuration' page with the subtitle 'About your endpoint'. It contains several input fields for configuration:

- MQ policy name\***: EVENTQM
- Host\***: hostname.ibm.com
- Port\***: 2414
- Queue manager\***: EVENTQM
- Channel name\***: SYSTEM.DEF.SVRCONN

Figure 6-15 IBM MQ Endpoint configuration

- The queue manager entry is displayed as shown in Figure 6-16. To enable connectivity between your endpoint configurations and the on-premises agent, click **Enable** or **Synchronize**.

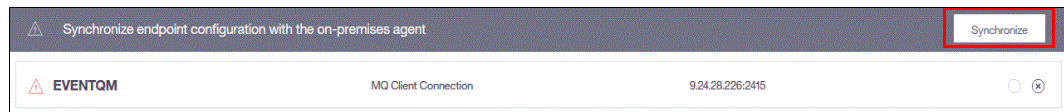


Figure 6-16 Synchronize IBM MQ Endpoint

- Click **Download Configuration** to save the agent configuration and close the window by clicking **Do this later** as shown in Figure 6-17.

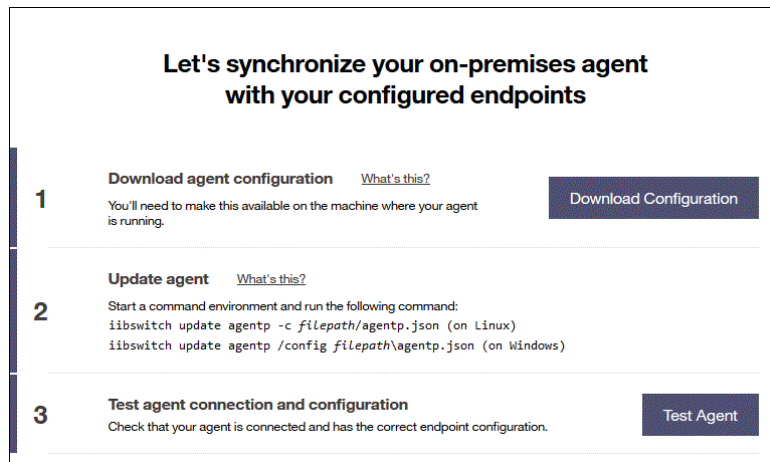


Figure 6-17 Download agent configuration

- Use the downloaded `agentp.json` file to replace the example configuration in the `99_docker\iib-sc2\agents` subdirectory.

This concludes the steps for the IBM MQ Endpoint. The following steps focus on the stubbed API that is implemented as a Callable flow in IBM Integration Bus.

- Click **Callable Flows** and then on **Set up an agent** as shown in Figure 6-18.

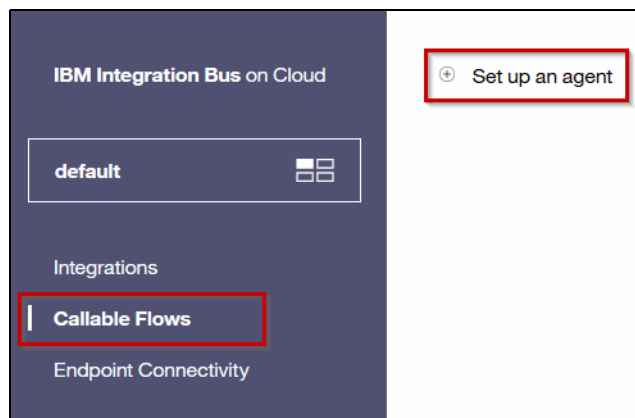


Figure 6-18 Setting up an agent for Callable Flows

9. Similar to the previous agent, download the configuration file as shown in Figure 6-19.

**Let's set up an agent**

- 1 Install the on-premises agent in your network** [Get It Here](#)  
The agent, which enables secure network connectivity, is included in IBM Integration Bus v10.0.0.4 or later. Install Integration Bus if you do not already have it installed.
- 2 Download agent configuration** [What's this?](#) [Download Configuration](#)  
You'll need to make the agent configuration available on the machine where you installed Integration Bus.
- 3 Start agent in an existing integration server** [What's this?](#)  
The agent needs to run in the integration server where you've deployed your spilt flow. Start a command environment and run the following command:  
`mqsichangeproperties <NODE_NAME> -e <SERVER_NAME> -o ComIbmIIBSwitchManager -n agentXConfigFile -p filepath/agentx.json (Linux)`  
`mqsichangeproperties <NODE_NAME> -e <SERVER_NAME> -o ComIbmIIBSwitchManager -n agentXConfigFile -p filepath\agentx.json (Windows)`
- 4 Test agent connection** [Test Agent](#)  
Check that your agent is connected.

[Do this later](#)

Figure 6-19 Download configuration

Later on, you will use the same window to test the agent connectivity.

10. Use the downloaded `agentx.json` file to replace the example configuration in the `99_docker\iib-sc2\agents` subdirectory.

### Starting the docker compose environment

Perform the following steps to star the docker compose environment:

1. Open a command line window and go to the directory where you cloned the GitHub repository. The contents of the directory should look similar to Figure 6-20.

```
vmuser@ubuntu:~/dev/sg248351/scenario2$ ls
1_coding 2_testing 99_docker docker-compose.yml LICENSE README.md
vmuser@ubuntu:~/dev/sg248351/scenario2$
```

Figure 6-20 Scenario2 directory and content

2. Run the following command to indicate that you accept the license agreements for the products that are used in this scenario:

```
export LICENSE=accept
```

**Note:** The details about the respective licenses can be found in the readme file in the scenario2 GitHub repository.

3. Start the docker-compose environment by running this command:

```
docker-compose up -d
```

**Tip:** The first time you that run the command takes some time as Docker is downloading the container images from the internet.

4. After the command has completed, run **docker-compose ps** to check the status of the environment. The output should look similar to Figure 6-21.

```
vmuser@ubuntu:~/dev/sg248351/scenario2$ docker-compose ps
-----
      Name            Command             State      Ports
-----
scenario2_mq_1      mq.sh              Up         0.0.0.0:32810->1414/tcp, 5672/tcp
scenario2_stub_1   iib_manage.sh     Up         0.0.0.0:32816->4414/tcp, 0.0.0.0:32815->7800/tcp
vmuser@ubuntu:~/dev/sg248351/scenario2$
```

Figure 6-21 Status of docker-compose environment

Two containers run in this environment: One for IBM MQ and one for the stub.

### Testing agent connectivity

As part of the procedure that was run when the docker-compose environment was created and started, the agentx and agentp configuration files were used, and inside the container the agents were automatically started for you. These steps verify that the agents are successfully connecting to IBM Integration Bus on Cloud:

1. Return to the browser window where you are logged in to IBM Integration Bus on Cloud.
2. If not already open, click the **Callable Flows** menu on the left side. As before, click **Set up an agent**, but this time, when the window opens, click **Test Agent**.

**Note:** You do not need to run any of the other steps that the window specifies. You have already downloaded the configuration file previously, and the commands shown were included in the container setup and have been run during the startup of the containers.

The window should now show you a green confirmation message Agent in sync as shown in Figure 6-22.

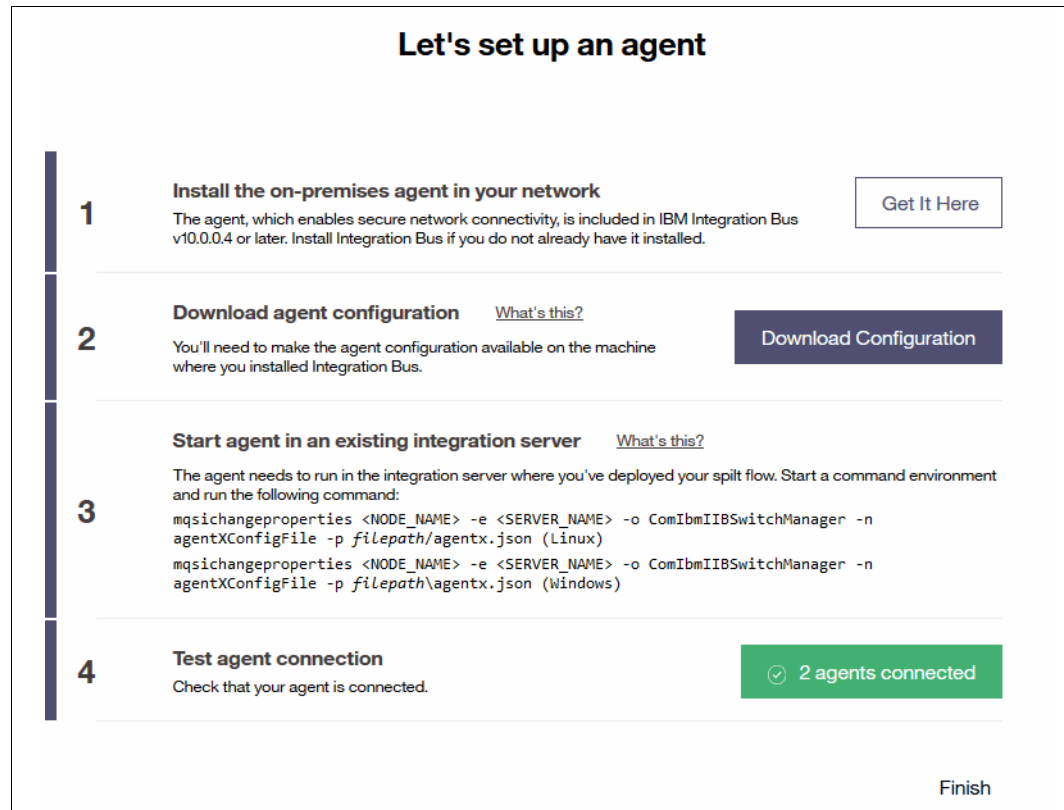


Figure 6-22 Agent connected

3. Next, click the **Endpoint Connectivity** menu and click the **Synchronize** button shown above the entry for EVENTQM. You should see another green confirmation message. Afterward, the IBM MQ Endpoint should be listed as shown in Figure 6-23.



Figure 6-23 EVENTQM endpoint connected

This concludes the preparation for this scenario. The steps in the next section focus on the message flow that will be deployed to IBM Integration Bus on Cloud and that will make the different order update events available to IBM App Connect.

## Build

Complete the following build steps:

1. Start IBM Integration Bus Toolkit and select or create a workspace.
2. Some IBM Integration Bus resources have been provided as a starting point for this scenario. Import the project interchange file by choosing **File** → **Import** and select Project Interchange.



3. Click **Next** as shown in Figure 6-24.

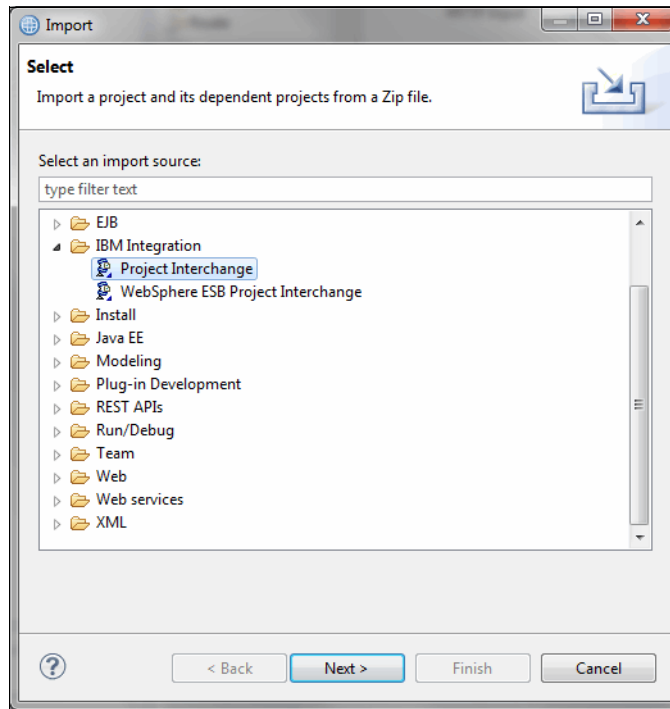


Figure 6-24 Importing the Project Interchange file

4. Click **Browse** and go to the `iib_scenario2.zip` file located in the `1_coding` subdirectory. Then, select all resources from the project interchange file and click **Finish** as shown in Figure 6-25.

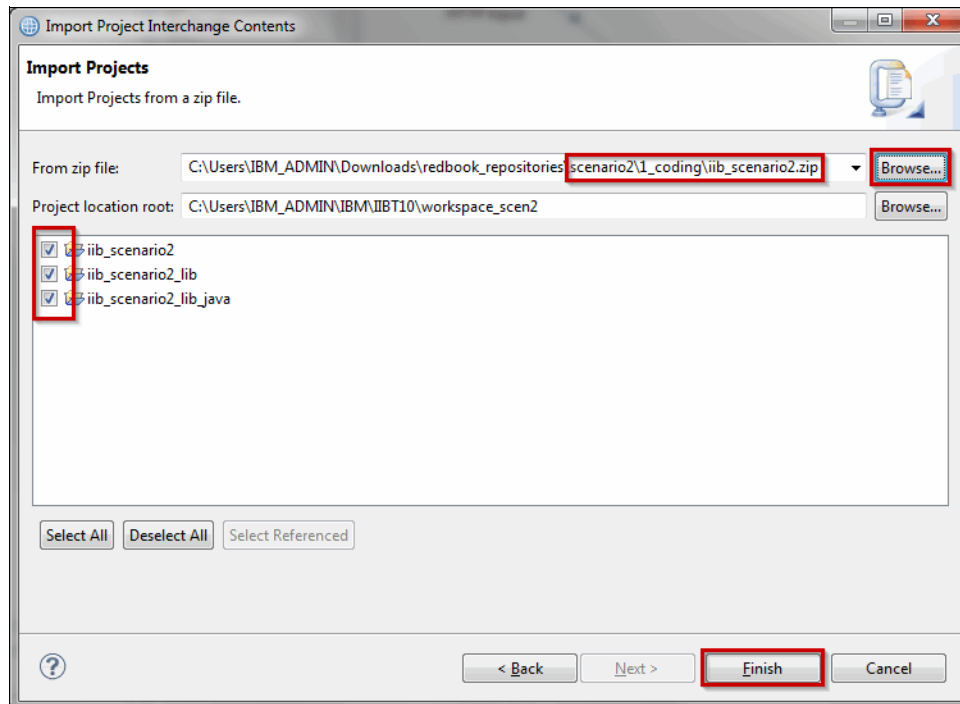


Figure 6-25 Finalize Project Interchange import

This action provides you with a starting point from which to complete the build activities.

- In the `iib_scenario2` application, double-click the **PublishTriggerFromEvent** flow to open it. It should look similar to Figure 6-26.

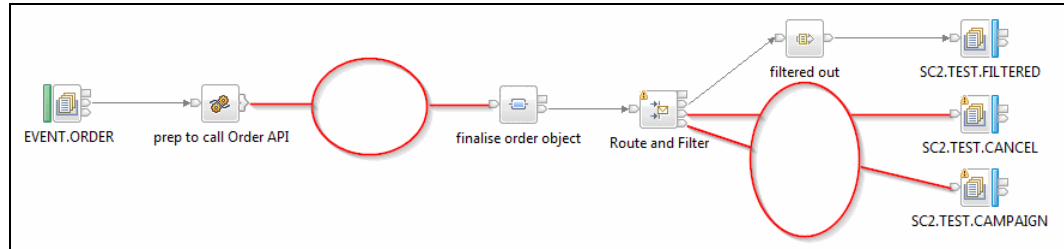


Figure 6-26 Flow skeleton

The gaps in the flow that are highlighted in red are completed with these items:

- The `CallableFlowInvoke` node to mimic the Order API stub call
- The nodes that emit the `campaignOrder` and `cancelOrder` events towards IBM App Connect.

This configuration is described in the steps below.

- From the `CallableFlow` drawer in the palette, add a `CallableFlowInvoke` node and wire it between the Out terminal of the `Compute` node and the `Mapping` node.
- Configure the properties of the new node as shown in Table 6-3.

Table 6-3 Properties of the new node

Drawer name	Property name	Property value	Comment
Description	Node name	<code>CallOrderAPI</code>	
Basic	Target Application	<code>iib_scenario2_orderapi_stub</code>	The spelling needs to be exact.
Basic	Target endpoint name	<code>getOrder</code>	The spelling needs to be exact.
Basic	Request timeout (sec)	15	

- The `iib_scenario2_lib`, which you imported earlier, contains two subflows: `InnerWebhook` and `WebhookOutput` as shown in Figure 6-27.

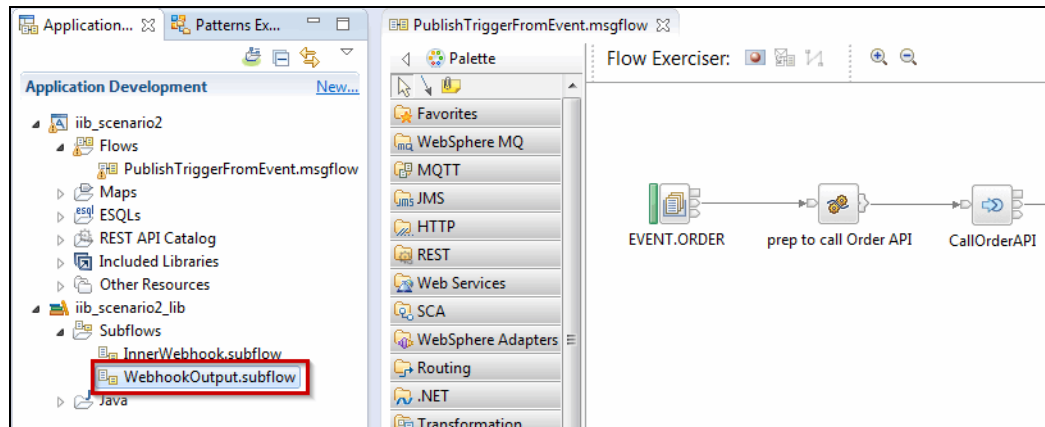


Figure 6-27 WebhookOutput subflow

9. Drag the WebhookOutput subflow onto the canvas twice so that you have two copies of the same subflow. Wire them between the Routing node and the IBM MQ Output nodes. The terminals on the Routing node that need to be connected are named cancelOrder and campaignOrder. Use the same names to label the subflow nodes themselves as shown in Figure 6-28.

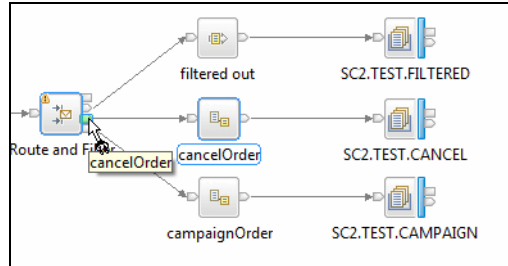


Figure 6-28 Wiring the Route node terminals

10. Configure the properties on the subflow nodes as follows:
  - a. The WebhookBaseUrl and WebhookStarUrl properties on the Basic drawer are configured identically for both nodes. Use the following values:  
 WebhookBaseUrl: /triggers/orders-updates  
 WebhookStarUrl: /triggers/orders-updates/\*
  - b. The EventType on the same drawer has different values for each node. They are the same ones that were used earlier for the names of the subflow nodes: cancelOrder and campaignOrder. Figure 6-29 shows the properties that are configured for the cancelOrder subflow as an example.

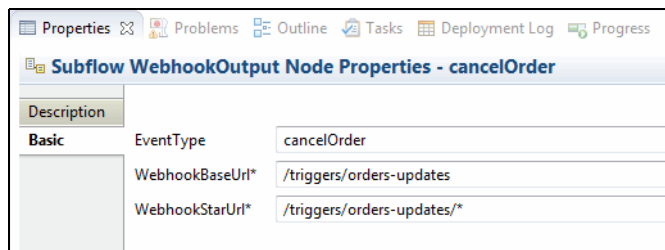


Figure 6-29 Subflow node properties for cancelOrder

11. This concludes the build steps for this section. To deploy the application to IBM Integration Bus on Cloud, create a BAR file.

In the **Toolkit** menu, click **File** → **New** → **BAR file** as shown in Figure 6-30.

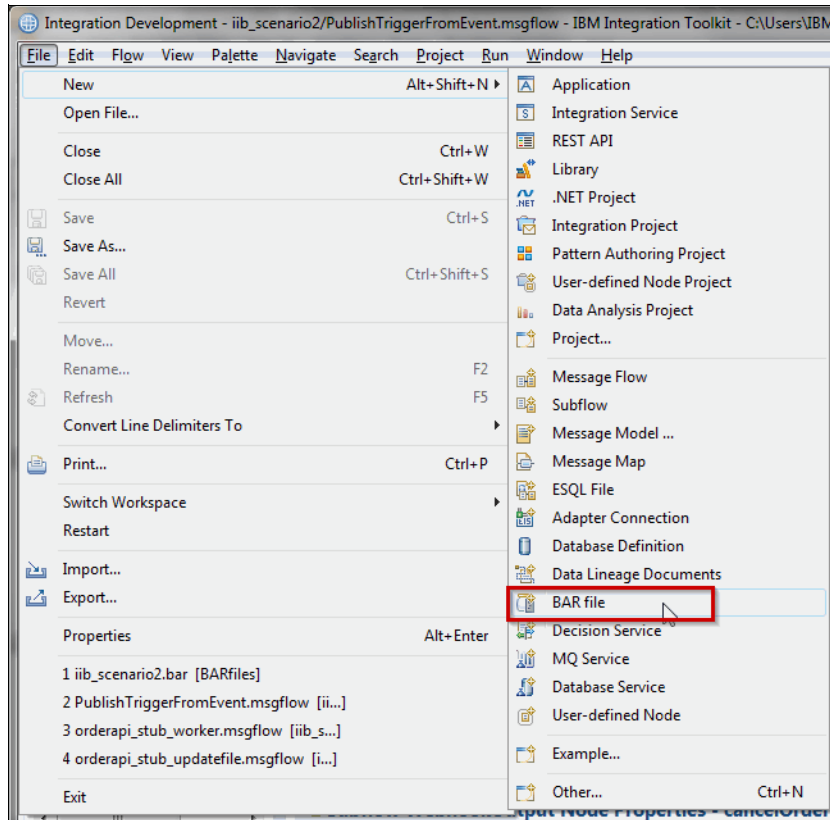


Figure 6-30 New BAR file

12. Change the name of the BAR file to `iib_scenario2`. Leave the other properties at the default values and click **Finish**.
13. A new tab opens to allow you to configure the content of the BAR file. Choose the `iib_scenario2` application on the left, then click the **Build and Save** button and **OK** after the build process is complete. Figure 6-31 illustrates this process.

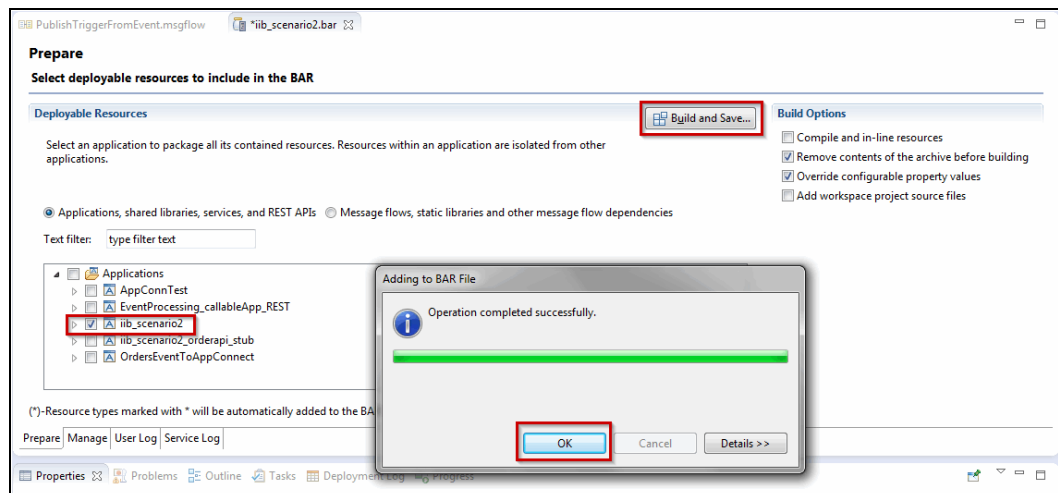


Figure 6-31 Build and Save the BAR file

14. Now switch to a web browser and log in to IBM Integration Bus on Cloud at <https://ibm-cloud-ui.ibmintegrationbus.ibmcloud.com> to deploy your application.
15. You can upload your integration flows to IBM Integration Bus on Cloud by using the **Add Integration** option as shown in Figure 6-32.

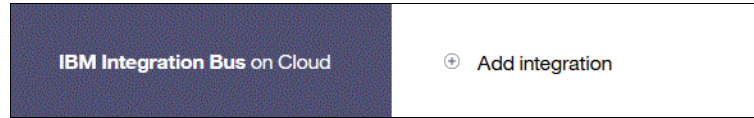


Figure 6-32 Add an integration

16. Upload the BAR file that contains your integration flow as shown in Figure 6-33.

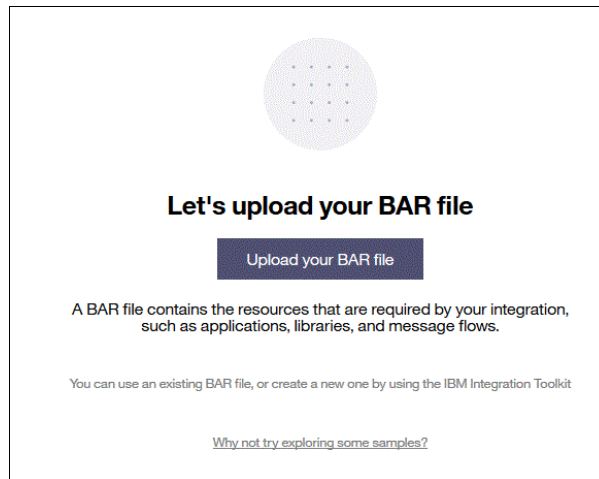


Figure 6-33 Upload BAR file

17. When the BAR file is successfully imported into the IBM Integration Bus on Cloud, the contents of the BAR file are displayed. Switch off **Basic Authentication** as shown in Figure 6-34.

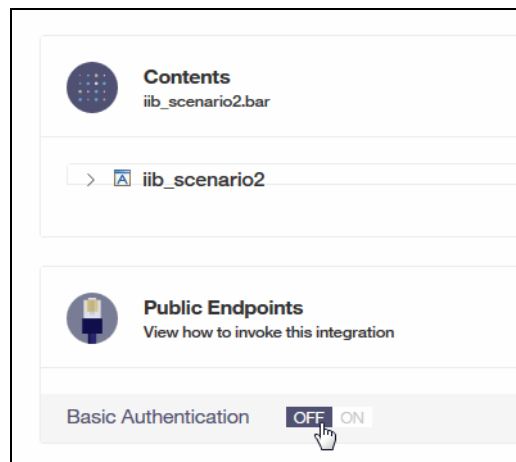


Figure 6-34 Switching off Basic Authentication

18. Click **Save** to save the integration. The integration flow then gets deployed and has the default state of stopped. The deployed integration becomes visible on the **Integrations** menu. Click the item itself, as shown in Figure 6-35, to see the details of the integration.

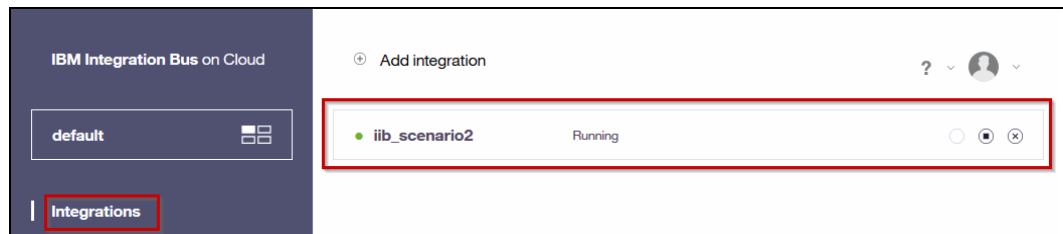


Figure 6-35 Details of the integration

19. Start this integration by clicking **Actions** → **Start** as shown in Figure 6-36.

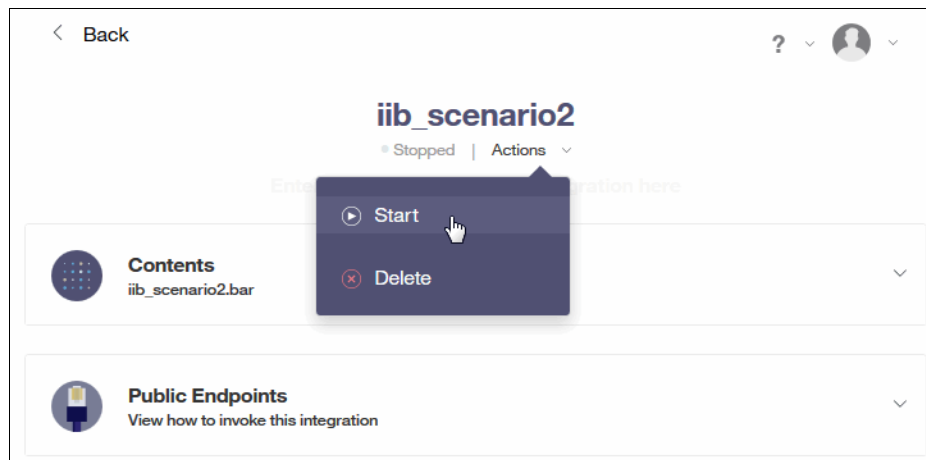


Figure 6-36 Start integration

20. An information window might be displayed after you click **Start**. Confirm the information by clicking **OK**.

21. After the application is running, click **Public Endpoints** to expand that section. Highlight and copy or note down the **Host** as shown in Figure 6-37. You will need that information in the testing steps in the next section.

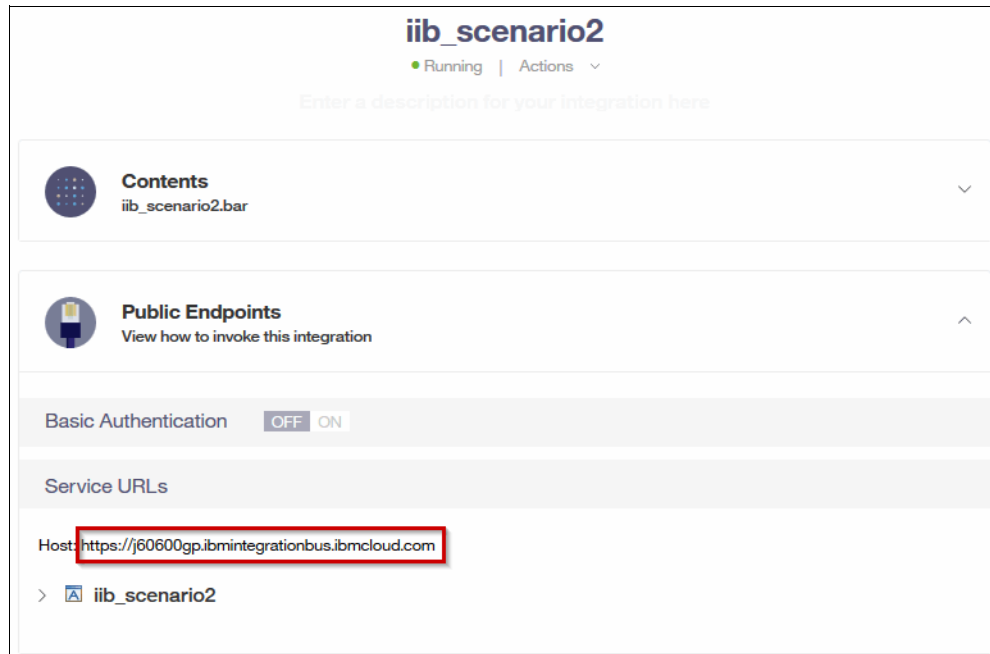


Figure 6-37 Host name for the application on IBM Integration Bus on Cloud

This concludes the build steps for the IBM Integration Bus application and message flow.

## Test

This part of the testing verifies that the application is correctly deployed to IBM Integration Bus on Cloud. In particular, the testing covers the connectivity to on-premises, the processing and mapping of messages, and the ability of the flow to receive webhook subscriptions. Complete these steps:

1. Return to the command line where you started your docker-compose environment.
2. Open a shell inside the IBM MQ container by running the following command:  

```
docker exec -it scenario2_mq_1 /bin/bash
```

You are now inside the IBM MQ container and are logged in as the user root.
3. Change to the home directory by running `cd` and list the files in the directory with the `ls` command. The output should look like Figure 6-38.

```
(mq:9.0)root@6f493e634c5e:/# cd
(mq:9.0)root@6f493e634c5e:~# ls
iib-listTriggers.sh  qs-clear.sh  qs-show.sh  stub-updateFiles.sh  testfiles  testme.sh
(mq:9.0)root@6f493e634c5e:~#
```

Figure 6-38 Directory contents

4. There a number of small scripts that have been provided to reduce the amount of typing that you need to do during the testing steps.

The first script is the `testme.sh` script. Its purpose is to put one of the files from the `testfiles` directory on the `EVENT.ORDER` queue. In this test, the IBM Integration Bus application picks up the message, processes it, and routes it to both the `campaignOrder` and `cancelOrder` targets. Therefore, you should see two messages on the queues after the test is complete.

5. Run the following command:

```
./testme.sh 02
```

The output from the script should look like in Figure 6-39.

```
(mq:9.0)root@e91135304914:~# ./testme.sh 02
Sample AMQSPUT0 start
target queue is EVENT.ORDER
Sample AMQSPUT0 end

QUEUE(SC2.TEST.CANCEL)          TYPE(Queue)
CURDEPTH(1)
QUEUE(SC2.TEST.FILTERED)       TYPE(Queue)
CURDEPTH(0)
QUEUE(SC2.TEST.CAMPAIGN)       TYPE(Queue)
CURDEPTH(1)
QUEUE(EVENT.ORDER)             TYPE(Queue)
CURDEPTH(0)
```

Figure 6-39 Cancel and Campaign messages received

**Note:** You can use the `./qs-show.sh` command to display the contents of the queues again in case the messages did not arrive in time as expected.

Because you have not configured any of the IBM App Connect functions yet, those events were not published to IBM App Connect. After it is configured, IBM App Connect will register triggers for these events. The script called `iib-listTriggers.sh` allows you to list those registered triggers.

6. Use the `iib-listTriggers.sh` script with the host name of your deployed application on IBM Integration Bus on Cloud. For example:

```
./iib-listTriggers.sh https://ch4gine4.ibmintegrationbus.ibmcloud.com
```

The output should be an empty JSON array `[]` as shown in Figure 6-40.

```
(mq:9.0)root@6f493e634c5e:~# ./iib-listTriggers.sh https://ch4gine4.ibmintegrationbus.ibmcloud.com
[]
```

Figure 6-40 Output from the `iib-listTriggers` command

This concludes the testing steps for this section.

## 6.2.4 Creating an AppConnect flow to process campaign-related orders

CompanyA wants to track the success of their campaigns, and decided to relate this to the order value generated for each campaign. Build an IBM App Connect flow that processes Campaign Order creation and cancellation events, and updates the Orders worksheet in the Google spreadsheet you used before.



## Prepare

In the previous section, you deployed the IBM Integration Bus application that is now available under a specific host name on IBM Integration Bus on Cloud. Use the same host name again to update the configuration file used in this section. Complete these steps:

1. Locate the 01\_coding subfolder for this scenario. The folder is part of the GitHub repository that you downloaded earlier.
2. Open the Orders-app-definition.yaml file in a text editor and replace the example host name in the file with the actual host name from your IBM Integration Bus on Cloud application as shown in Figure 6-41.

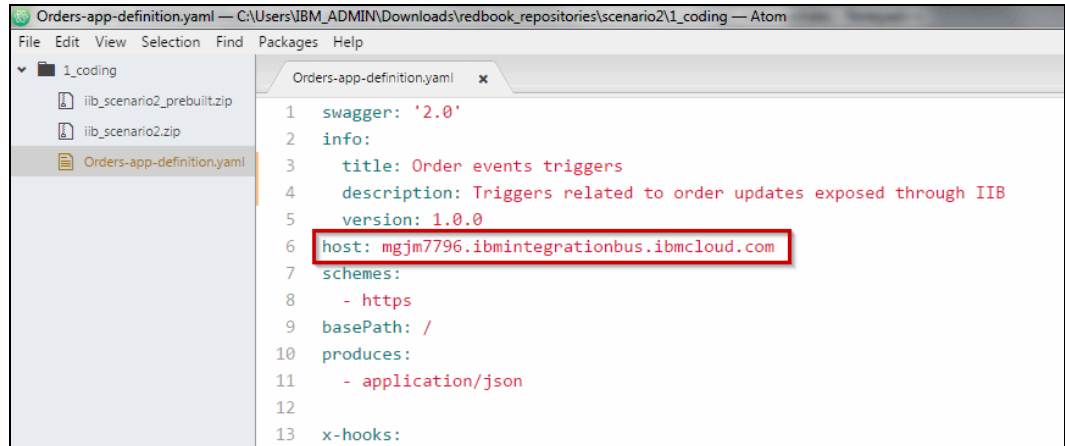


Figure 6-41 Replacing the host name in the application definition yaml file

This concludes the preparation activities for this section.

## Build

1. Log in to IBM App Connect, and from the dashboard, click **Applications**. To add the application that processes the Campaign Order events from IBM Integration Bus, click **Add your application now** as shown in Figure 6-42.

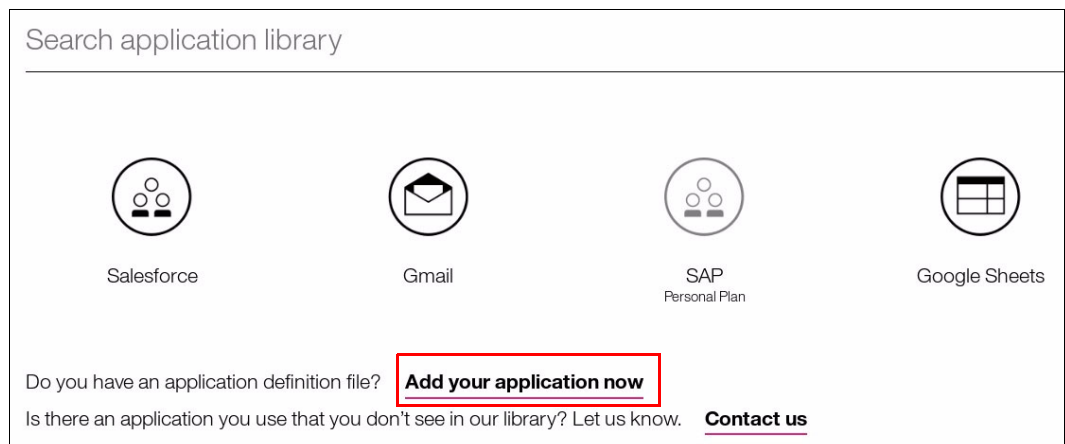


Figure 6-42 Add an application to IBM App Connect

2. Provide the **Application name** and **Description**, as shown in Table 6-4.

Table 6-4 Application name and Description

Field	Value
Application name	Orders
Description	Triggers related to order update events from the Order Management system

3. Click the area that specifies the drop application definition file here, or browse and select the Orders-app-definition.yaml file that you prepared earlier.
4. Click **Apply** to add the application to App Connect as shown in Figure 6-43.

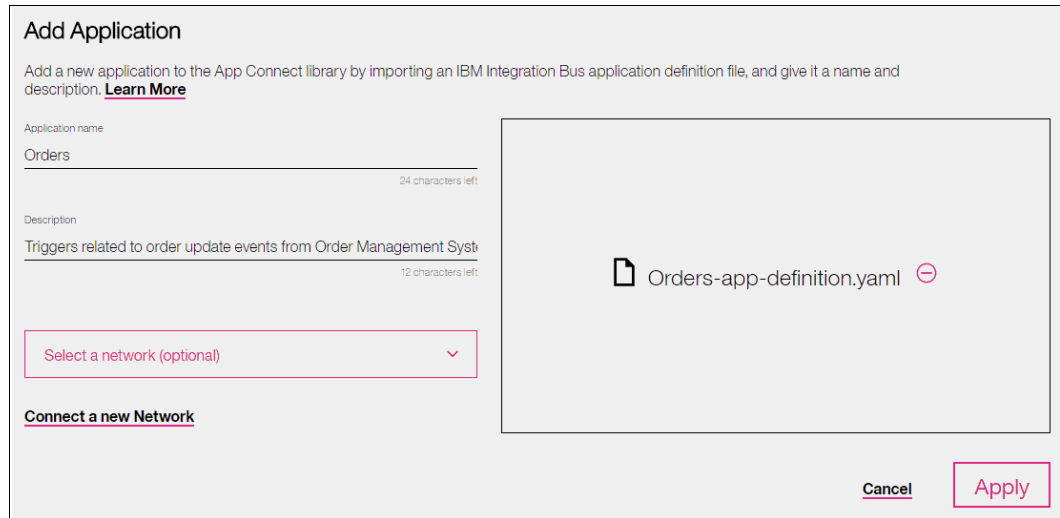


Figure 6-43 Add Application window

5. The new application is added to the application library as shown in Figure 6-44.

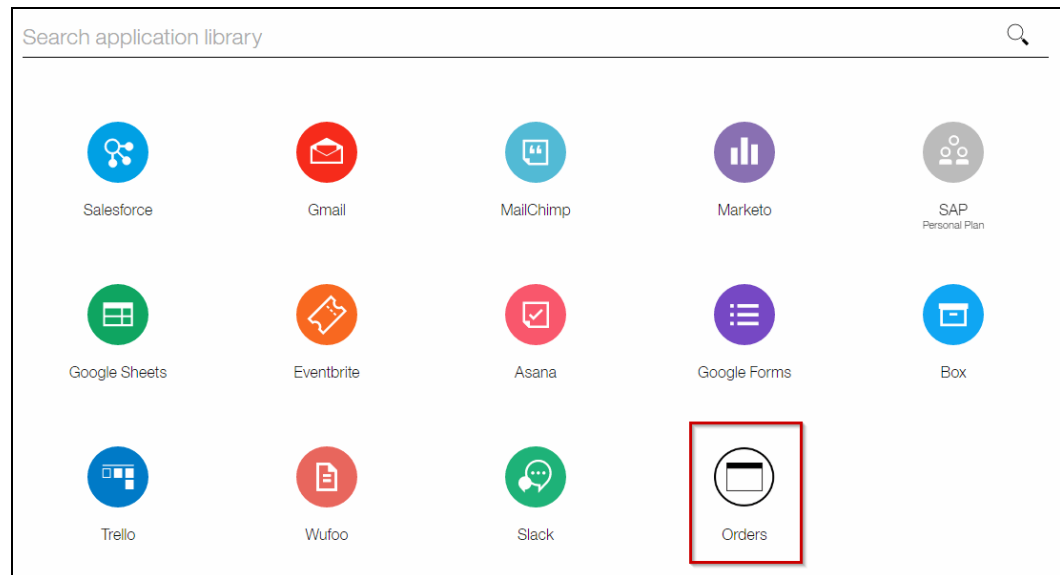


Figure 6-44 Orders application

6. Click **Save + Continue** to save the current selection.

7. Click **Create a flow** and select **CampaignOrderEvents** as the first application.
8. Go to the next window, and select **Google Sheets** as the second application.
9. In the next window, select the default **Trigger & Action** flow type.
10. Go to the next window and select the **campaignOrder** trigger. The draft flow now looks like Figure 6-45.

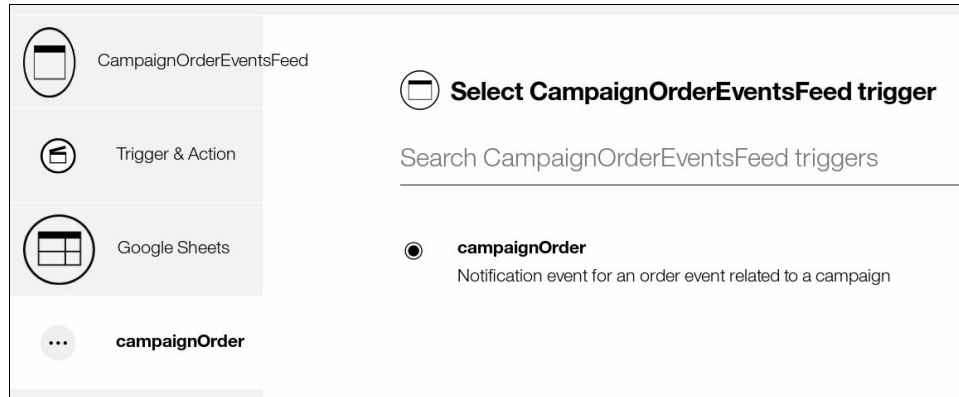


Figure 6-45 Select the campaignOrder event trigger

11. Click **Save + Continue** to switch to the next window, where you can select the **Google Sheets** action. The only available action is **Create Row**. Save the flow configuration, and continue to the next window. Here you select the **Google** account that is used for updating the spreadsheet. See Figure 6-46.

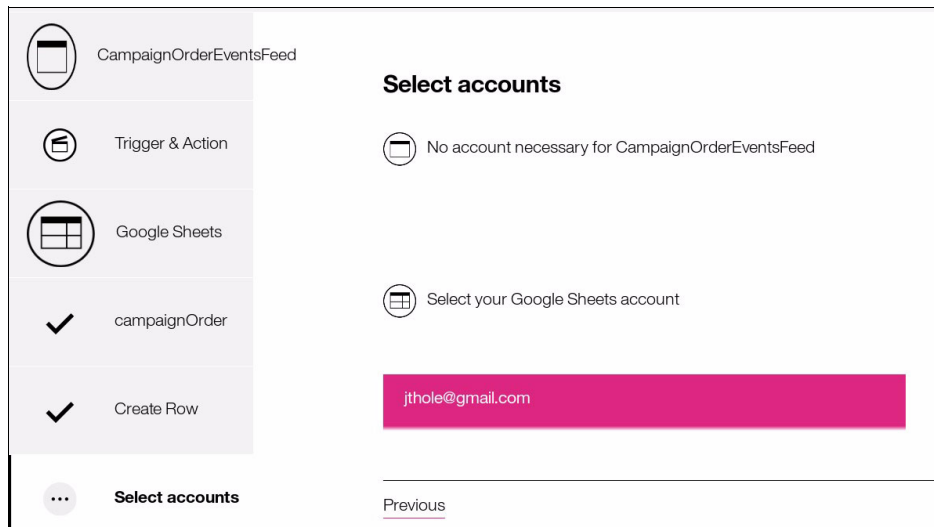


Figure 6-46 Select accounts for the applications used in the flow

12. Click **Save + Continue** and select the Google spreadsheet and worksheet from the list. Use the values shown in Figure 6-47. Click **Auto Match Fields** to map the fields from the **CampaignOrderEvents** application on the spreadsheet columns. Type #campaignId in the mapping field for the **Campaign** column.

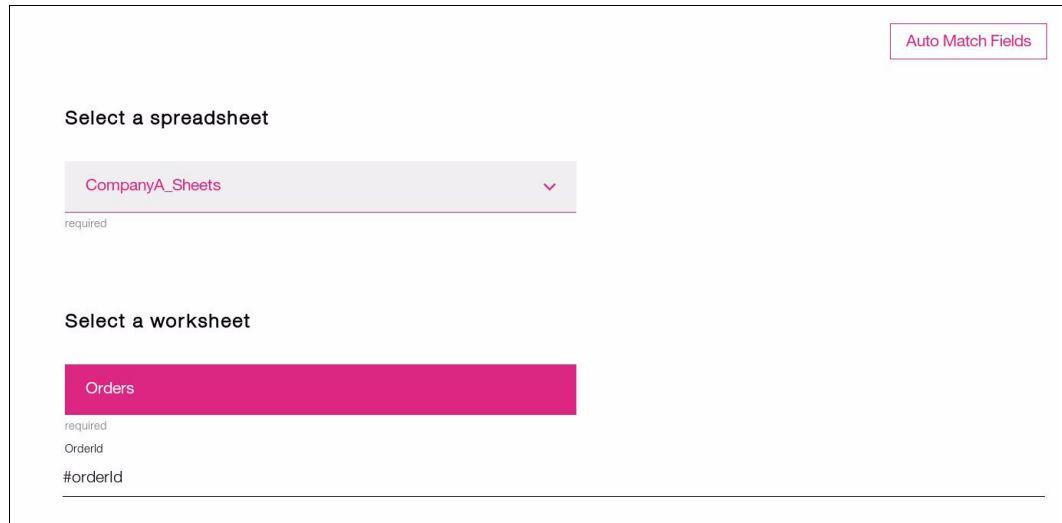


Figure 6-47 Select the worksheet, and auto match fields

13. Finalize your flow, and save it as CampaignOrderEvents. Click **Switch on Flow** to start the flow. As you can see in Figure 6-48, two flows are now running in your AppConnect space.

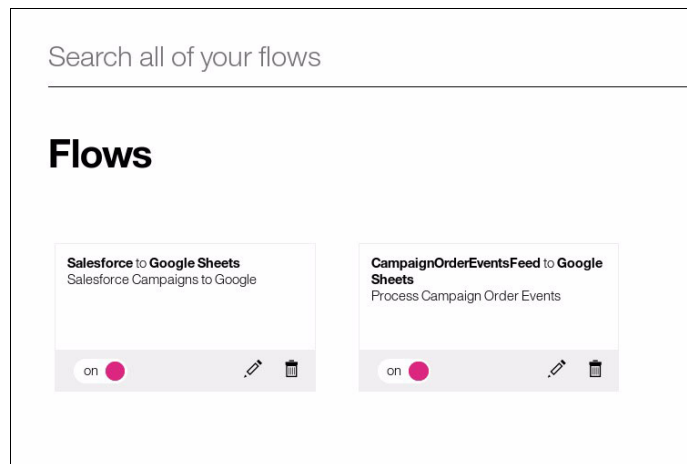


Figure 6-48 Completed flow

14. Test your flow by generating a Campaign order creation or cancellation event in Integration Bus and inspecting the Orders and Campaign worksheets in your Google spreadsheet.

## Test

Perform the following steps to test this part of the scenario:

1. Return to the command line where you started your docker-compose environment.
2. Open a shell inside the mq container by running the following command:

```
docker exec -it scenario2_mq_1 /bin/bash
```

You are now inside the mq container and are logged in as user root.

3. Change to the home directory with the `cd` command and use the `test.me` script to send an event about a new campaign-related order event. Run the following command:  
`./testme.sh 01`
4. The response on the command line should look like Figure 6-49. You should also see an additional line in the Google Sheets spreadsheet.

```
(mq:9.0)root@e91135304914:~# ./qs-show.sh
QUEUE(SC2.TEST.CANCEL)           TYPE(Queue)
CURDEPTH(0)
QUEUE(SC2.TEST.FILTERED)        TYPE(Queue)
CURDEPTH(0)
QUEUE(SC2.TEST.CAMPAIGN)        TYPE(Queue)
CURDEPTH(1)
QUEUE(EVENT.ORDER)              TYPE(Queue)
CURDEPTH(0)
```

Figure 6-49 One event has been routed to the orderCampaign trigger

This concludes the testing steps for this section.

## 6.2.5 Creating an AppConnect flow to process canceled orders

This section covers creating an AppConnect flow to process canceled orders.

### Build

Complete the following build steps:

1. Log in to IBM App Connect.
2. On the dashboard view, start by clicking **Create a flow**.

Confirm each of the following steps by clicking **Save + Continue** as shown in Figure 6-50.



Figure 6-50 Save + Continue button

3. Select **Orders** as the first and **Salesforce** as the second application.
4. Select **Trigger & Action** as the type of flow.
5. Select the **cancelOrder** trigger as shown in Figure 6-51.

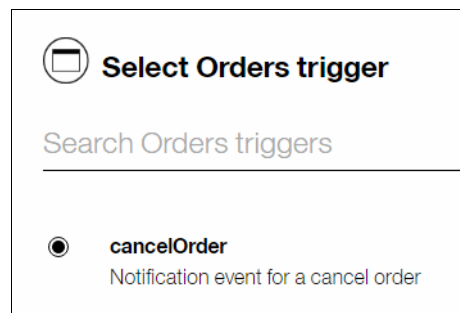


Figure 6-51 Cancel order trigger

6. For the Salesforce action, select **Create Case** as shown in Figure 6-52.

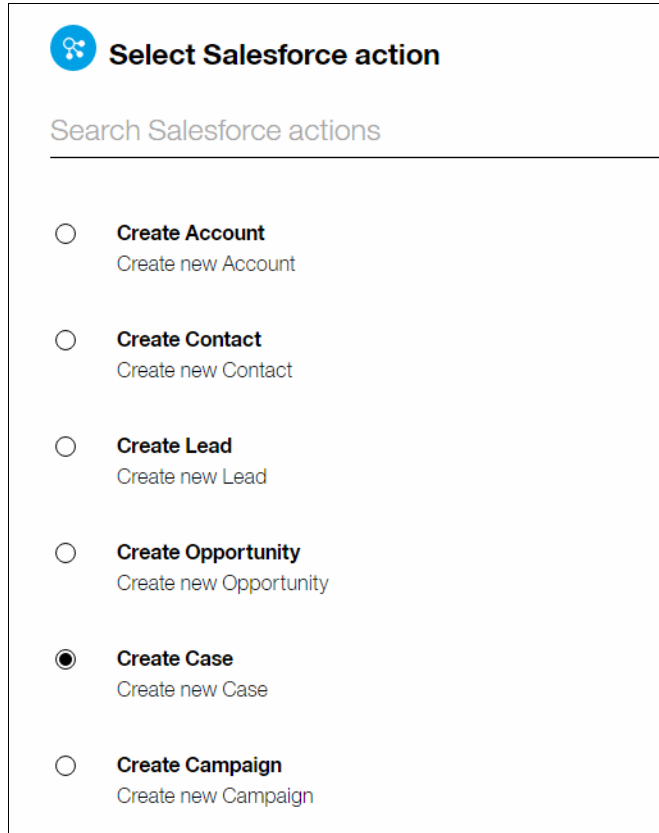


Figure 6-52 Create Case trigger

7. Select the already configured Salesforce account details (or configure new ones).
8. In the Set up Salesforce action window, configure mapping and values for the fields as shown in Table 6-5.

Table 6-5 Mapping and values on the set up Salesforce action window

Target field name	Target field value	Comment
Account ID	#salesforceId	
Status	New	
Case Origin	Web	
Subject	#accountName: Cancelled order	
Priority	Low	
Description	Order number #orderId was canceled. This follow-up was created automatically.	

- Name the flow **Cancelled order follow-up** and click **Switch on Flow** to finish the process as shown in Figure 6-53.

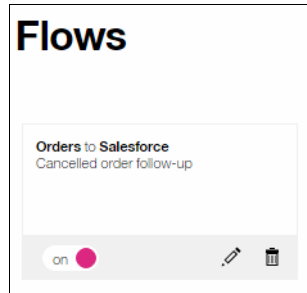


Figure 6-53 Cancelled order follow-up flow

The flow is now active. This concludes the build steps in this section.

## Test

Complete the following steps to test this part of the scenario:

- Return to the command line where you started your docker-compose environment.
- Open a shell inside the mq container by running the following command:  

```
docker exec -it scenario2_mq_1 /bin/bash
```

 You are now inside the mq container and are logged in as user root.
- Change to the home directory with the `cd` command and use the `iib-listTriggers.sh` script to see whether the newly created IBM App Connect flow has registered a webhook trigger with the IBM Integration Bus application. You must replace the host name shown in Figure 6-54 with the one from your actual IBM Integration Bus application.

```
(mq:9.0)root@e91135304914:~# ./iib-listTriggers.sh https://ch4g1ne4.ibmintegrationbus.ibmcloud.com [{"id":1,"callback":{"url":"https://webhook-connector-provider-prod.appconnect.ibmcloud.com/webhooks/270007JT57/webhook/cancelOrder/28f03e10-97a5-11e6-8c55-c9ffc41a06a4"},"event_types":["cancelOrder"]}]
```

Figure 6-54 Registered AppConnect flow

- Use the `test.me` script to send an event about a new campaign-related order event. Run the following command:

```
./testme.sh 04
```

The output of the script is shown in Figure 6-55.

```
(mq:9.0)root@e91135304914:~# ./testme.sh 04
Sample AMQSPUT0 start
target queue is EVENT.ORDER
Sample AMQSPUT0 end

QUEUE(SC2.TEST.CANCEL)          TYPE(Queue)
CURDEPTH(1)
QUEUE(SC2.TEST.FILTERED)        TYPE(Queue)
CURDEPTH(0)
QUEUE(SC2.TEST.CAMPAIGN)        TYPE(Queue)
CURDEPTH(0)
QUEUE(EVENT.ORDER)              TYPE(Queue)
CURDEPTH(0)
```

Figure 6-55 Command line output

You should find a new case in Salesforce as shown in Figure 6-56.

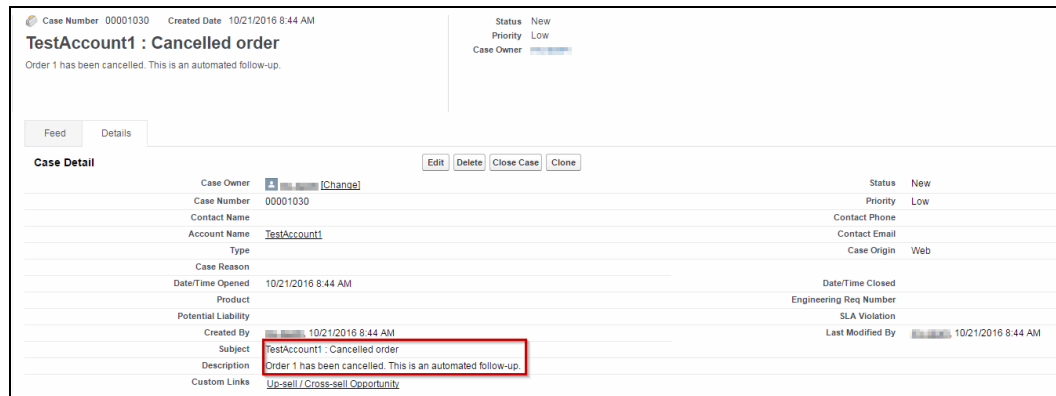


Figure 6-56 New Case in Salesforce

This concludes the testing steps for this section.

## 6.3 Resources

A number of additional resources are relevant to this scenario.

### 6.3.1 AppConnect pattern for IBM Integration Bus

The integration developer community provides a tutorial on custom application development using the AppConnect pattern for IBM Integration Bus. The tutorial can be obtained from this website:

<https://developer.ibm.com/integration/docs/app-connect/tutorials-for-ibm-app-connect/#pane12>

This tutorial provides a basic background on how to use a custom application (built by using IBM Integration Bus) with IBM App Connect. You can download the trigger pattern from this website:

<https://github.com/ot4i/iib-app-connect-trigger-pattern>

Although the sample included in this tutorial is based on the message flow that listens on http endpoints, it is possible to modify the sample to use your own endpoint, for example, IBM MQ. The instructions on how to modify the existing sample to use MQ Endpoints is available from another article available on GitHub community at:

<https://github.com/ot4i/iib-app-connect-trigger-pattern/blob/master/doc/modwarehouse.md>



As the example scenario uses IBM MQ as an endpoint to listen to the events, the design of the integration flow follows a similar construct as described in the GitHub article mentioned above. In addition to that, the integration flow uses the callable flow feature to start another integration flow running on-premises. The topology as shown in Figure 6-57 is used in the scenario described in this chapter to split the message flow processing between IBM Integration Bus and IBM Integration Bus on Cloud.

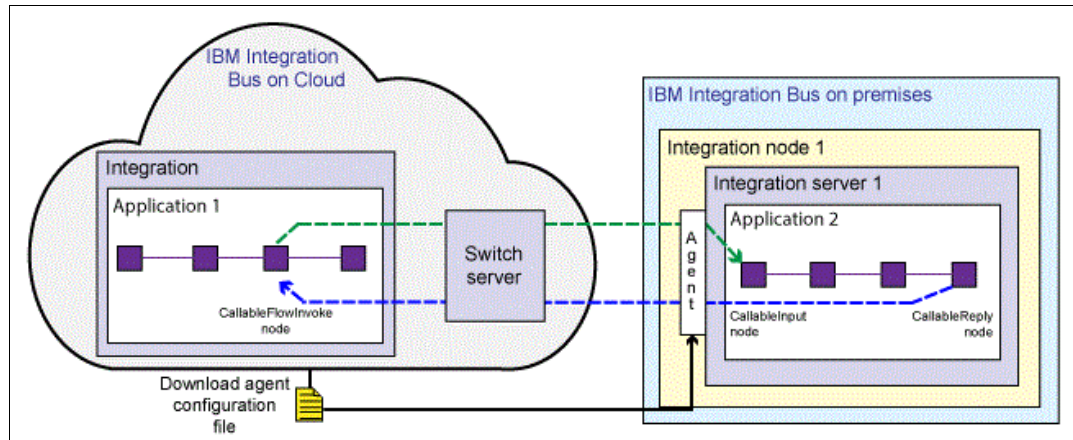


Figure 6-57 Integration topology that uses callable flows

See the IBM Integration Bus Knowledge Center topic on *Preparing the environment for callable flows* at:

[https://www.ibm.com/support/knowledgecenter/SSMKHH\\_10.0.0/com.ibm.etools.mft.doc/c123148\\_.htm](https://www.ibm.com/support/knowledgecenter/SSMKHH_10.0.0/com.ibm.etools.mft.doc/c123148_.htm)

*Preparing the environment to split processing between IBM Integration Bus and IBM Integration Bus on Cloud* describes the steps to set up the connectivity agent.

The integrations that you create in IBM Integration Bus on Cloud might need to interact with private endpoints on premises, such as a database or IBM MQ. You provide connection details in IBM Integration Bus on Cloud for each of your private endpoints. The concept on Secure connectivity to private endpoints is explained in the IBM Integration Bus Knowledge Center at:

[https://www.ibm.com/support/knowledgecenter/SSMKHH\\_10.0.0/com.ibm.iib.cloud.doc/c100017\\_.htm](https://www.ibm.com/support/knowledgecenter/SSMKHH_10.0.0/com.ibm.iib.cloud.doc/c100017_.htm)

### 6.3.2 Installing the trigger pattern in IBM Integration Bus Toolkit

A pattern for publishing events from IBM Integration Bus to be consumed as triggers for IBM App Connect is available at the following GitHub repository:

<https://github.com/ot4i/iib-app-connect-trigger-pattern>

For the scenario, provide the respective resources as a library. However, there is also a pattern available to help with the creation of those subflows. If you want to use the pattern, follow these instructions:

1. Download the compressed file by using the **Clone or Download** option. Extract the file into a separate directory and import the projects in IBM Integration Bus toolkit by clicking **File** → **Import** → **General** → **Existing Projects into Workspace**. Figure 6-58 shows the list of projects that have been imported into the Integration Bus Toolkit.

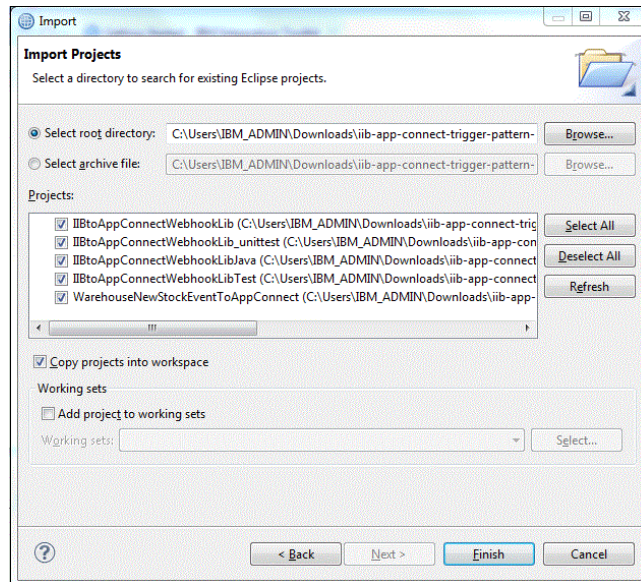


Figure 6-58 Import trigger pattern

2. After successful import, the Toolkit workspace displays the projects as shown in Figure 6-59.

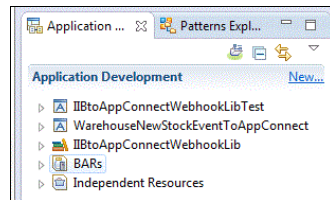


Figure 6-59 Application and Libraries for Trigger pattern



## Kick-start digital teams

This chapter describes how easily you can enable digital teams with design applications displaying events coming from the enterprise world. It covers how to set up the enterprise backend to stream events and create a digital application to consume those events.

This chapter includes the following sections:

- ▶ Solution overview
- ▶ Technical implementation of variant B
- ▶ Technical implementation of variant C
- ▶ Technical implementation of variant A

## 7.1 Solution overview

The solution for this chapter can be divided in two parts:

- ▶ The event stream creation: In this part, you use the work you have done in the previous chapters to generate a stream of events based on the status changes of the orders.

**Note:** Even though this chapter relies on the status events that were created in Scenario 1, this chapter is fully independent.

- ▶ The event stream consumption: In this part, you build an application composed of these elements:
  - A backend microservice that consumes the event stream
  - An API gateway that exposes the backend microservice
  - A front end user interface to display the stream of events

This chapter details three different methods of implementation. Each method is called a *variant*. Each variant includes the pros and cons to help you easily select the alternative that best matches your use case.

The logical diagram Figure 7-1 describes the three variants to implement the event stream creation and consumption.

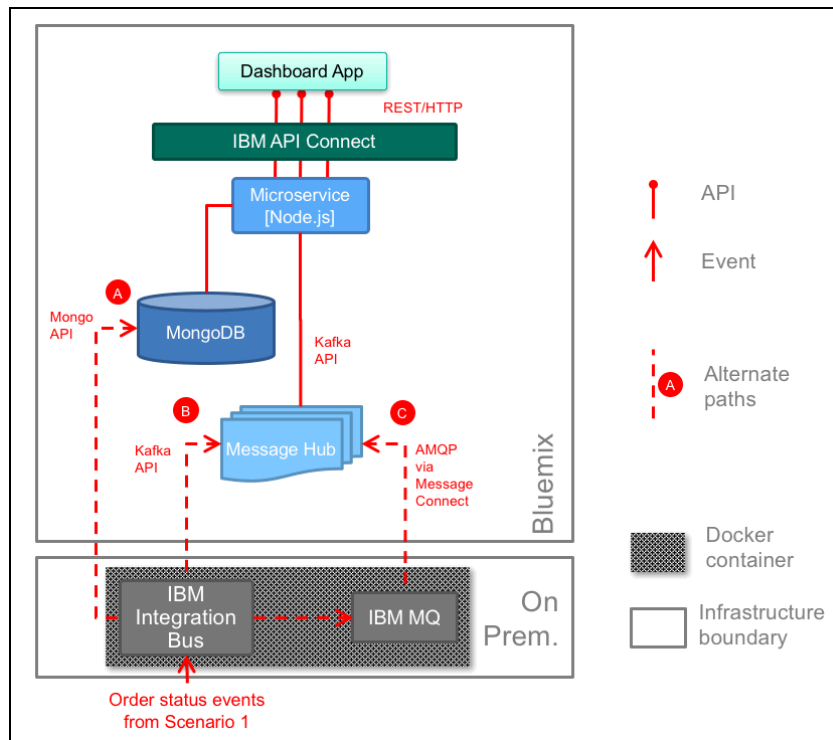


Figure 7-1 Scenario 3 - Logical diagram - All variants

## 7.1.1 Variant A

The variant A is the simplest method of implementation. In this case, the implementation process is straightforward:

1. IBM Integration Bus retrieves the status events from the orders.
2. IBM Integration Bus enriches and formats the events before storing them in MongoDB.
3. The IBM API Connect microservice exposes the events that it finds stored in MongoDB.
4. The dashboard application displays the stream of events.

Although this variant is the easiest to implement, it has some drawbacks. The most important one is that there is a *direct dependency* between the event stream creation and its consumption.

In most companies, the creation and the consumption of the event stream would be done by two different teams. In this variant, a change to the MongoDB database has a direct effect on both teams. In fact, a change to MongoDB requires an update to both the flow in the IBM Integration Bus and to the microservice in the IBM API Connect microservice.

To avoid this problem, the next variant implements *loose coupling* between the event stream creation and its consumption.

## 7.1.2 Variant B

The next option, variant B, is an evolution of variant A, which implements a form of loose coupling. In this case, the process involves these steps:

1. IBM Integration Bus retrieves the status events from the orders.
2. IBM Integration Bus enriches the events and publishes them to Message Hub.
3. The IBM API Connect microservice subscribes to the events from Message Hub.
4. The IBM API Connect microservice formats the events and stores them in MongoDB.
5. The dashboard application displays the stream of events.

Steps 1 and 5 have not changed from variant A.

In this variant, loose coupling has been implemented between the event stream creation and its consumption by using Message Hub.

Message Hub is built on Apache Kafka and uses a publication/subscription mechanism. This mechanism allows IBM Integration Bus to push events data in Message Hub, and the application backend microservice to read events data in Message Hub with no dependency between the two actions.

When IBM Integration Bus pushes events, it uses a standard format and allows the microservice to handle the formatting and the storage in MongoDB. Moreover, after IBM Integration Bus has pushed the data to Message Hub, any microservices can read the data from Message Hub.

Loose coupling has been achieved in this variant. However, there is still a drawback to this variant that can be an issue in some cases. IBM Integration Bus is the direct producer of the event stream and it needs to be able to scale with the number of updates.

To fix this scaling issue in variant C, we will use IBM MQ features in IBM Integration Bus.

### 7.1.3 Variant C

The variant C is an evolution of variant B, where message queuing is implemented.

In this case, the process involves these steps:

1. IBM Integration Bus retrieves the status events from the orders.
2. IBM Integration Bus enriches the events and sends them to a message queue in MQ, where they will wait until they are retrieved by Message Connect.
3. Message Connect pulls the events from IBM MQ and publishes them to Message Hub.
4. The IBM API Connect microservice subscribes to the events from Message Hub.
5. The IBM API Connect microservice formats the events and stores them in MongoDB.
6. The dashboard application displays the stream of events.

Only steps 2 and 3 have changed from variant B.

In this variant, message queuing has been implemented between IBM Integration Bus and Message Hub thanks to the use of Message Connect.

### 7.1.4 Implementation considerations

Now that the three variants have been detailed, consider the technical implementation of each variant. The technical diagram in Figure 7-2 is the technical equivalent of the logical diagram in Figure 7-1 on page 154.

The technical diagram describes on a technical level the components that are required to implement each variant. To better understand the changes between each variant, the alternative flows are represented as dotted lines and are color-coded by variant.

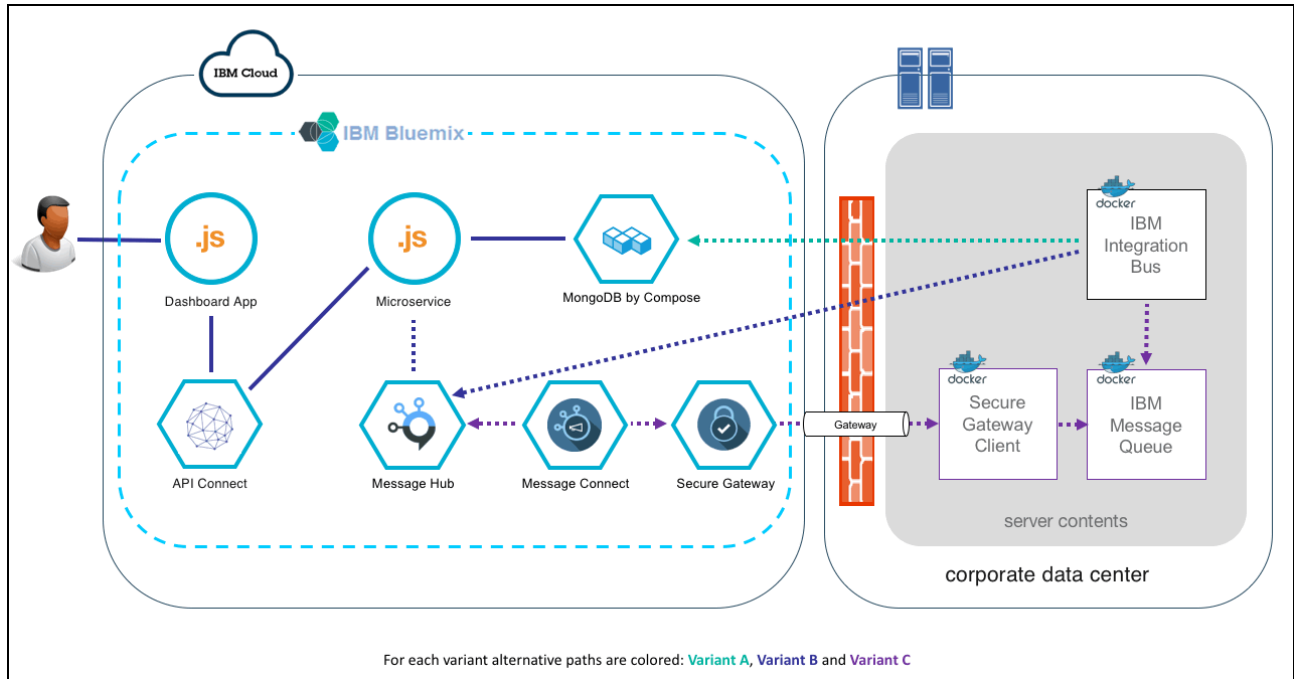


Figure 7-2 Scenario 3: Technical diagram for all variants

To facilitate the technical implementation of each variant, we decided to fully describe variant B as the preferred variant to implement in most cases.

For variants C and A, only the steps that differ from variant B are described.

Each variant starts with a diagram that displays only the components that are needed for this option, with a numbered sequence to follow for the implementation.

## 7.2 Technical implementation of variant B

In this section, variant B is implemented by going through six steps in the order that is displayed in Figure 7-3.

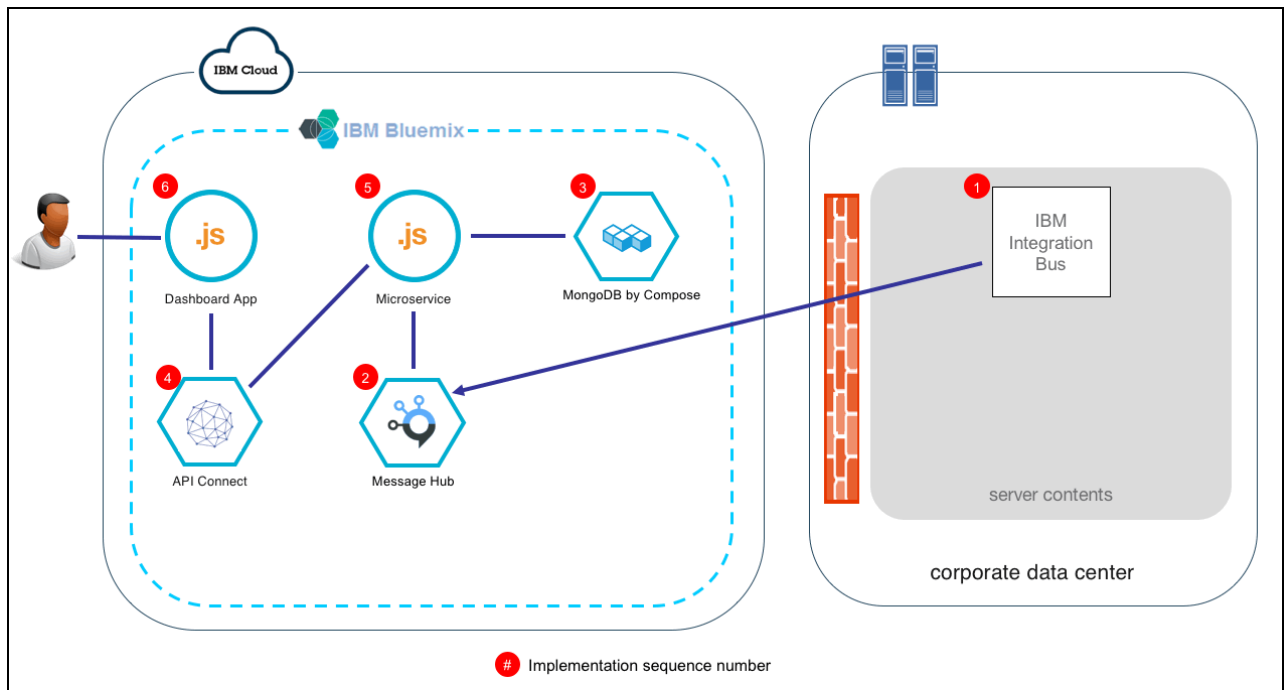


Figure 7-3 Scenario 3: Technical diagram for variant B

### 7.2.1 IBM Integration Bus: Exposing enterprise order events

The basis for the event stream that the digital team uses for the dashboard app are the order update events that come from the Order Management system. All processing with regards to these events is implemented in IBM Integration Bus and so far included the following requirements:

- ▶ The raw events are published as part of scenario 1.
- ▶ Some of the raw events also need enrichment, which is demonstrated in scenario 2.

The last aspect of feeding this data into the event stream is explained here as part of scenario 3. The first option of doing this is to publish the events directly from IBM Integration Bus into Message Hub. The message flow shown in Figure 7-4 is used for this purpose.



Figure 7-4 Flow overview

The flow performs the following actions:

1. A subflow is used to receive the order update event message.
2. The event is enriched (for example, in a cancel order event) and filtered if necessary.
3. A Mapping node is used to transform the message so that it matches the expected format of the microservice.
4. A Kafka Publisher node is used to send the event message to IBM Message Hub.
5. If you received the event from an HTTP input node, then the HTTP reply node is required by the protocol and is expected by the sender of the HTTP message so that it will not time out. HTTP reply node is expected to send a reply message, which in this case might be an acknowledgement of having received the message, which can also be taken as confirmation that the event has been published.

As already mentioned, the implementation has been simplified for this book in the following ways:

- ▶ As explained in the architecture overview, IBM MQ is used for messaging including event publication. To make it easier to test the flow at the end of the chapter, HTTP is used here as input to the flow instead of an IBM MQ queue.

**Note:** If you did not receive the message through HTTP, you cannot send an HTTP reply.

- ▶ Instead of enriching the raw event messages in the flow, testing is with already prepared messages. Enrichment is demonstrated in scenario 2.
- ▶ User and password for the Message Hub connection are configured for you.

The following sections take you through the building and testing of this flow.

## Prepare

To re-create the solution described in this scenario, read the introduction in Chapter 4, “Introduction to the scenarios” on page 55 and more specifically complete the prerequisites described in 4.3, “Re-creating the scenarios” on page 62.

In addition, the following preparation steps are specific to this scenario:

1. If you have not already done so, create a directory to hold all the configuration data for this IBM Redbooks, for example: sg248351, and open a shell or command prompt there.
2. Clone the GitHub repository for the scenario 3 by using the following command:  

```
git clone https://github.com/sg248351/scenario3
```



3. Change to the scenario3 directory by running this command:

```
cd scenario3
```

**Note:** A number of files have been provided to help you in the Build steps described later in this chapter. These files are located in the subdirectory called 1\_coding. The steps direct you back to this location when appropriate.

4. The pre-configured environment for this scenario consists of three containers. To use them, accept the license agreements for the products involved. To accept the license agreements, run the following command:

```
export LICENSE=accept
```

5. Run **docker-compose up -d** to start the containers in this environment.
6. Downloading the images and starting the containers might take some time. After it completes, check the status of the environment by running the following command:

```
docker-compose ps
```

The output should look similar to Figure 7-5.

```
vmuser@ubuntu:~/dev/sg248351/scenario3$ docker-compose ps
WARNING: The SC3_SGMID variable is not set. Defaulting to a blank string.
WARNING: The SC3_SECTOKEN variable is not set. Defaulting to a blank string.
-----
Name                                Command                                State                                Ports
-----
scenario3_iib-op_1                   iib_mq_manage.sh                      Up                                0.0.0.0:32777->1414/tcp, 0.0.0.0:32776->4414/tcp, 0.0.0.0:32775->7800/tcp
scenario3_mq_1                       mq.sh                                  Up                                0.0.0.0:32774->1414/tcp, 0.0.0.0:32773->5672/tcp
scenario3_sgC_1                      node lib/secgwclient.js - ...         Up
vmuser@ubuntu:~/dev/sg248351/scenario3$
```

Figure 7-5 Docker compose environment

The focus of the next steps is the container scenario3\_iib-op\_1, which represents the on-premises instance of IBM Integration Bus.

## Build

To build the scenario, complete the following steps:

1. Start IBM Integration Bus Toolkit and select or create a new workspace.
2. Create an application called iib\_scenario3 by select **File** → **New** → **Application** from the toolkit menu as shown in Figure 7-6 on page 160.

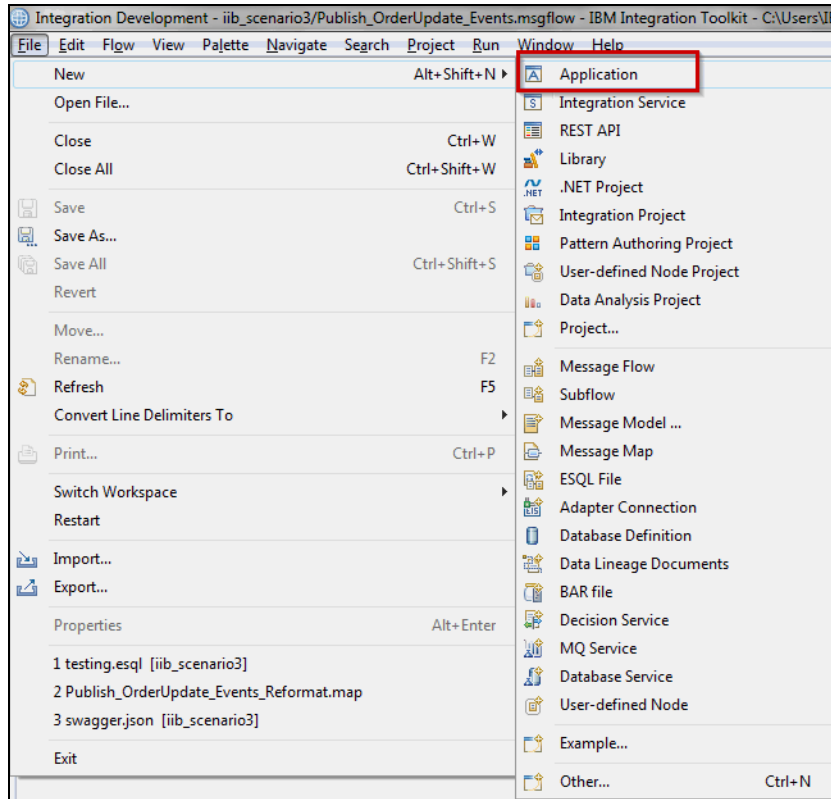


Figure 7-6 New application

3. Begin importing files into the newly created application by right-clicking **iib\_scenario3** and selecting **Import** as shown in Figure 7-7, then click **Next**.

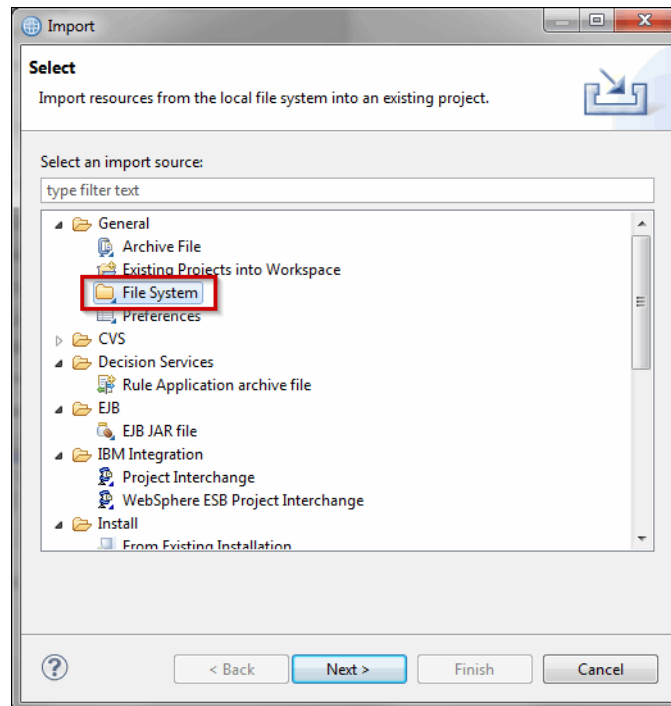


Figure 7-7 Import from File System

- To configure the import from file system, click **Browse** and go to the location to where you cloned the GitHub repository, specifically to the subdirectory called `1_coding` as shown in Figure 7-8.

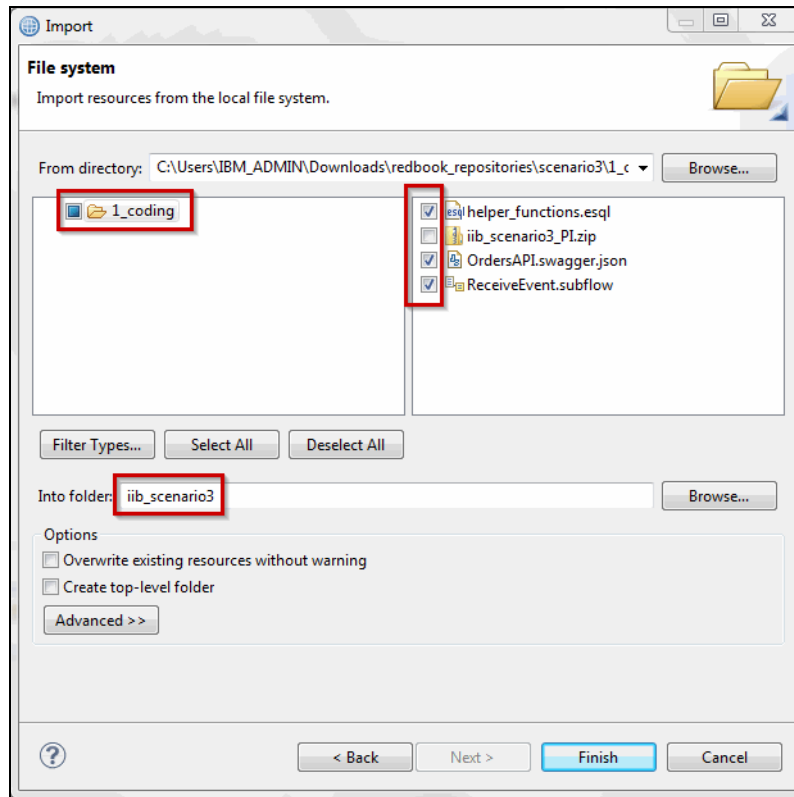


Figure 7-8 Selecting resources to be imported

- Select the subflow, esql, and swagger files as shown, and ensure `iib_scenario` folder is selected as the destination folder.
- Click **Finish**.

7. Create new message flow by right-clicking the `iib_scenario3` application and selecting **New** → **Message Flow** as shown in Figure 7-9.

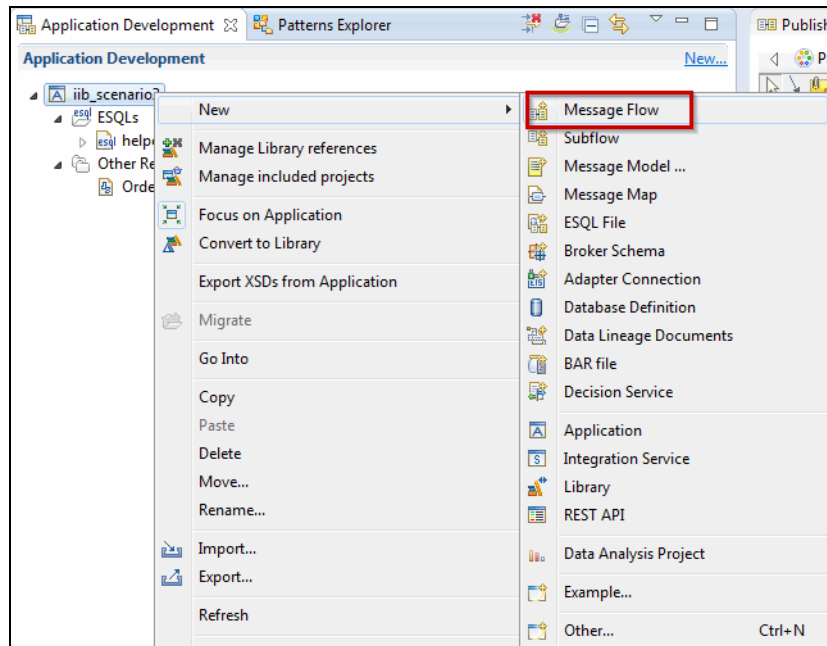


Figure 7-9 Creating a new Message Flow

8. Use `Publish_OrderUpdate_Events` as the name of the flow and click **Finish**.
9. Locate the `ReceiveTestEvent` subflow in the application on the left and drag it onto the flow canvas.
10. Add the `Passthrough` node from the `Construction` category in the palette and set the node name as `Enrich`.  
This is where the enrichment of certain events would normally happen through the `Orders` API. As a simplification, the implementation of this step is skipped, and already enriched test messages are used instead.
11. Connect the `Out` terminal of the `HTTP Input` node with the `Passthrough` node.
12. Add a `Mapping` node from the `Transformation` palette, name it `Reformat`, and connect the `Passthrough` node to it.

13. Double-click the **Mapping node** to configure the actual reformatting/mapping. In the window that opens, click **Next** and then configure the map inputs and outputs as shown in Figure 7-10.

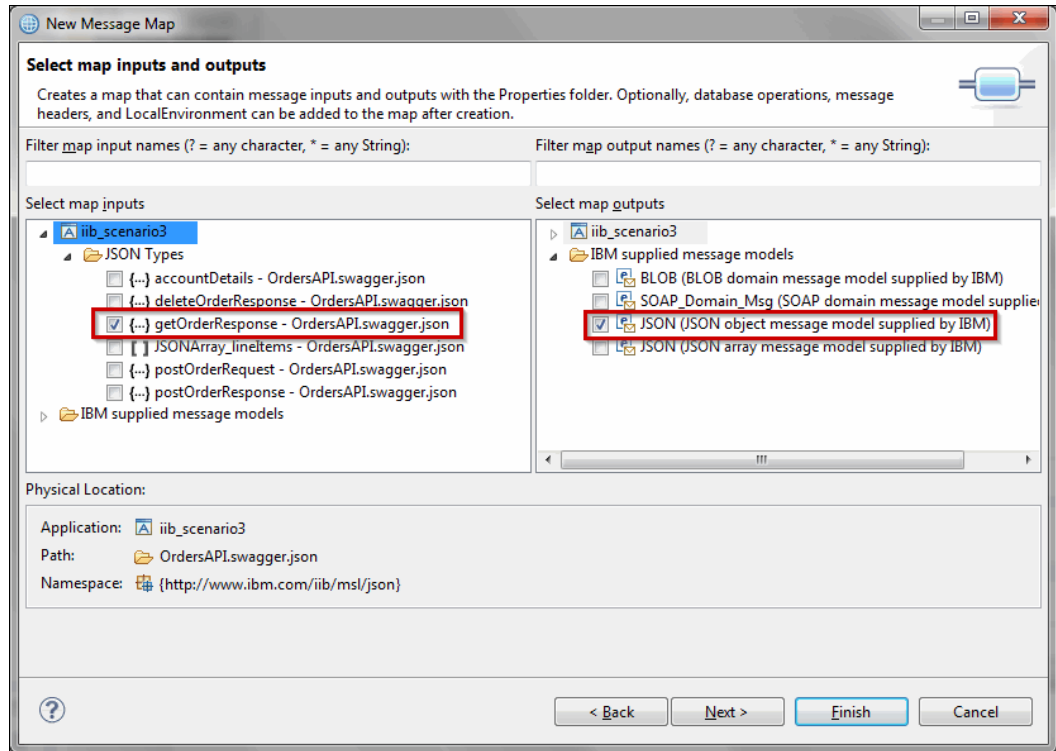


Figure 7-10 Select inputs and outputs for the mapping

14. Click **Finish**.

The JSON object on the right side of the mapping is what will be published to Message Hub for use by the microservices. Because we do not have a JSON schema that defines the JSON object, create the attributes from scratch in the mapping editor.

15. Expand the **JSONMsgType** object on the right to the lowest level and select the **any** element.

16. Right-click on the **any** element to create a new User-defined attribute called eventId as shown in Figure 7-11.

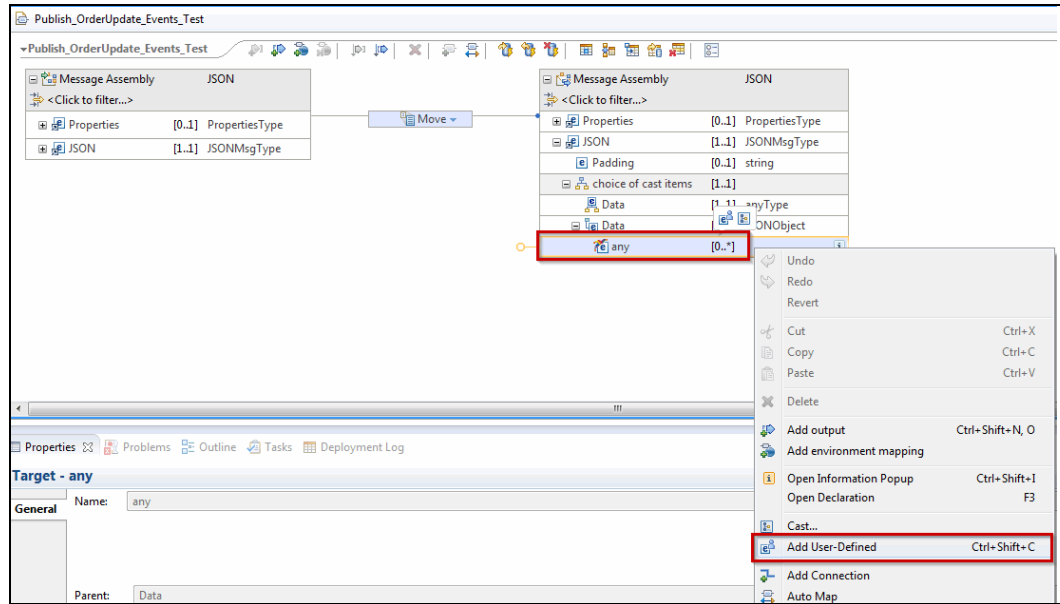


Figure 7-11 Adding a user-defined field

Instead of using the menu, you can also use the key combination **Ctrl+Shift+C** to create a new attribute or the small **Add User-Defined** icon, which appears on the any element after you have it selected/highlighted.

17. Use the technique described in the previous step to create the five more attributes with the names and types shown in Table 7-1.

Table 7-1 Attributes

Attribute name	Attribute type
time	string
type	string
orderId	string
customerId	string
description	string

Afterward, the structure in the mapping editor should look like Figure 7-12.

Message Assembly		JSON
<Click to filter...>		
Properties	[0..1]	PropertiesType
JSON	[1..1]	JSONMsgType
Padding	[0..1]	string
choice of cast items	[1..1]	
Data	[1..1]	anyType
Data	[1..1]	JSONObject
choice of cast items	[0..*]	
any	[1..1]	
eventId	[1..1]	string
time	[1..1]	date
type	[1..1]	string
orderId	[1..1]	string
customerId	[1..1]	string
description	[1..1]	string

Figure 7-12 JSON output object

18. Now you are ready to configure the mapping for the target attributes. Table 7-2 summarizes the mappings for all fields followed by more detailed steps to implement each one of them.

Table 7-2 Target attributes

Target element	Source element	Mapping description
eventId	n/a	The mapping creates a new unique eventId every time.
time	n/a	The current time when the event is published, generated in the mapping.
type	STATUS	The content of the source element is transformed to lowercase, but is otherwise taken unchanged.
orderId	ORDERID	Straight move from the ORDERID field.
customerId	ACCOUNTNAME	Straight move from the ACCOUNTNAME field.
description	lineItems	This field contain a textual description that details how many line items are in this order.

19. Start by configuring a Move action for the orderId and customerId elements by dragging the source element (from the left) to the target element (on the right) as shown in Figure 7-13.

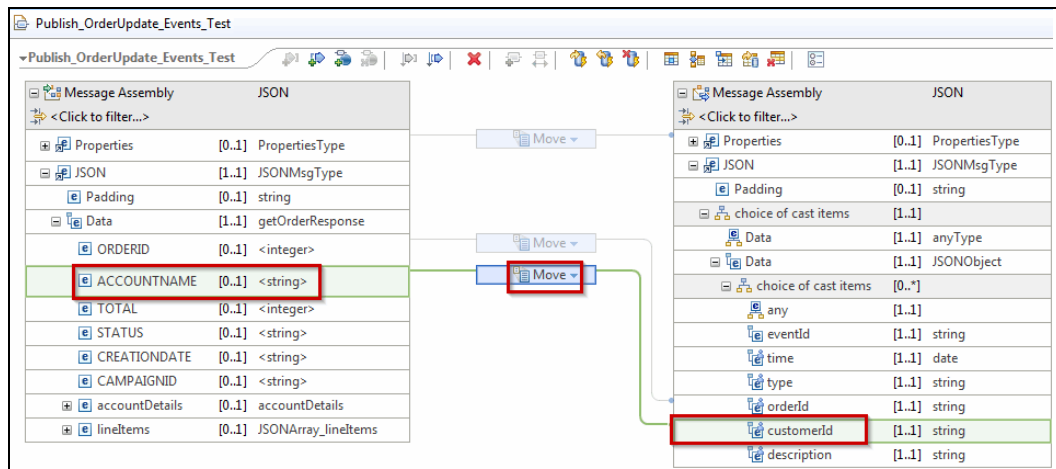


Figure 7-13 Creating a Move mapping action

20. The eventId and time attributes are both generated in the flow/mapping. Configure the eventId by using these steps:
  - a. Highlight the **eventId** field, right-click it, and select **Add Assign**.
  - b. Click the **twisty (triangle)** on the **Assign** action to show the list of possible mapping actions and change it to **Custom ESQL** as shown in Figure 7-14.

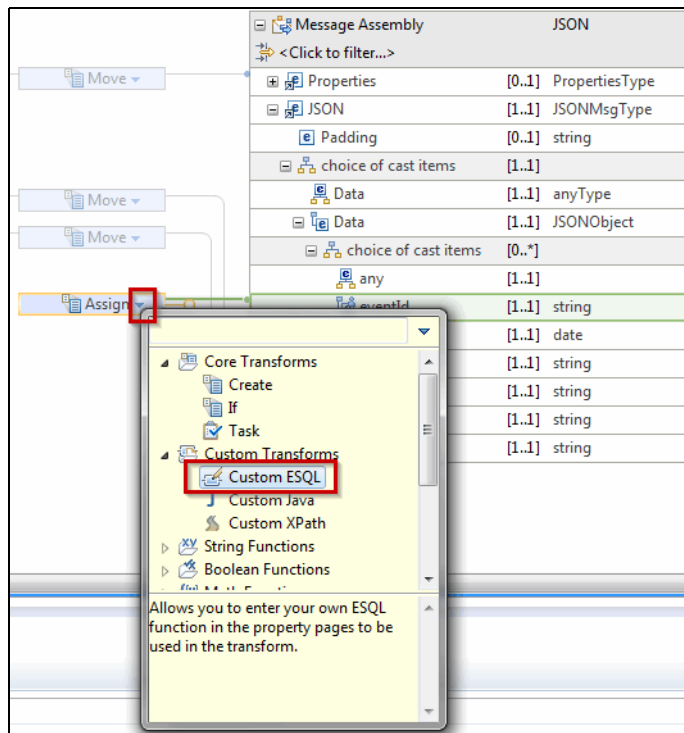


Figure 7-14 Custom ESQL mapping action

- c. After the **Custom ESQL** action is chosen, specify the specific esql routine that you want to use. This change is done in the properties below the mapping editor.



First, specify which file the routine is in by clicking **Browse** and selecting the `helper_functions.esql` file. Then select the `esqlUIDASCHAR` routine from the menu. The final configuration of the properties is shown in Figure 7-15.

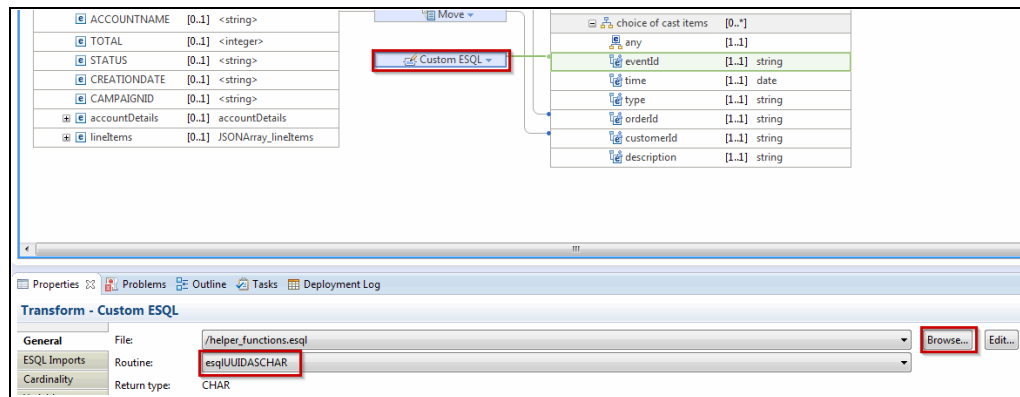


Figure 7-15 Configuring the Custom ESQL properties

21. The **time** field also starts off as an Assign action, but instead of using a Custom ESQL function, the field value is assigned by using a XPath function:
  - a. Change the Assign action to the `fn:current-date` function from the Date and Time Functions category as shown in Figure 7-16.

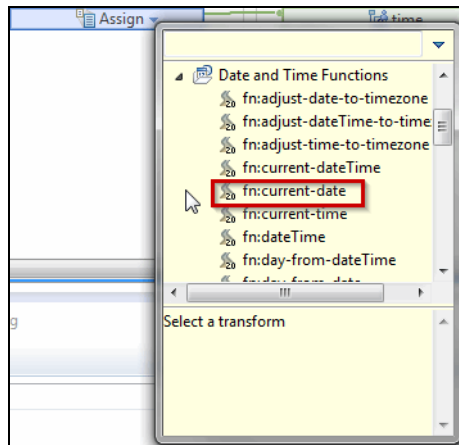


Figure 7-16 Current date function

22. To configure the mapping for the **type** field, drag the **STATUS** field onto the **type** field and change the transformation action to be the `fn:lower-case` function.
23. Drag the **lineItems** field onto the **description** field and enter the following custom XPath expression in the **General** field in the properties section below the mapping editor:
 

```
fn:concat("Order contains ", fn:count($lineItems/Item), " line items.")
```

The final mapping configuration should look like Figure 7-17.

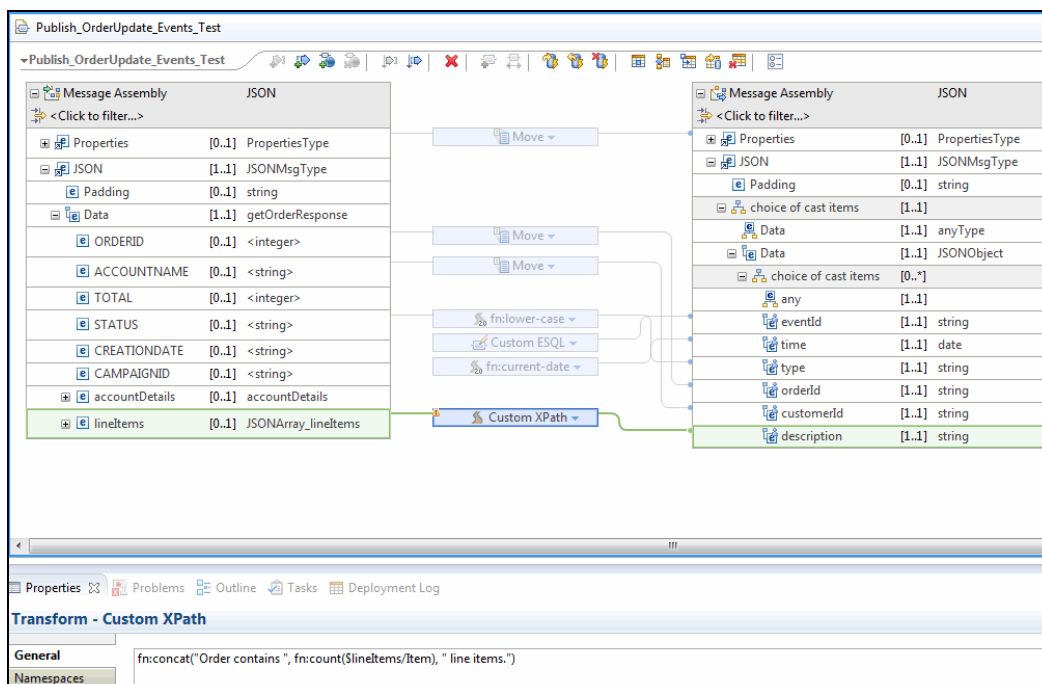


Figure 7-17 Complete mapping

24. Create credentials in Message Hub.

25. Add Kafka Producer node to the flow. Connect and configure the node as follows:

- a. Wire the Out terminal of the Mapping node to in terminal of the Kafka Producer node.
- b. Configure the following properties of the node listed in Table 7-3.

Table 7-3 Node properties

Tab name	Property name	Property value	Comment
Description	Node name	Publish to Message Hub	
Basic	Topic name	order-events	
Basic	Bootstrap servers	kafka01-prod01.messagehub.services.us-south-bluemix.net:9093	This is a sample value. Use the actual server from the credentials section of your Message Hub service configuration.
Basic	Client Id	iibsc3	
Basic	Acks	1	

26. From the HTTP category of the palette, add a HTTP Reply node to the flow. As mentioned previously, this is only necessary because we use HTTP to test the flow and need a way to return a response after invoking the flow.

- a. Wire the Out terminal of the Kafka Producer node to the in terminal of the HTTP Reply node.
- b. Configure the name of the HTTP Reply node as Response for testing.

This concludes the build steps for the message flow.

## Deploy and test

The application with the flow built in the previous section can now be deployed to the IBM Integration Bus instance running in the Docker container:

1. Connect the Toolkit where you developed the flow to the running IBM Integration Bus instance in Docker:
  - a. Determine which port the IBM Integration Bus admin port 4414 has been mapped to on the docker host by reviewing the output of the **docker-compose ps** command. In the example shown in Figure 7-18, this is port 32771.

```

vuser@ubuntu:~/dev/sg248351/scenario3$ docker-compose ps
WARNING: The SC3_SGMID variable is not set. Defaulting to a blank string.
WARNING: The SC3_SECTOKEN variable is not set. Defaulting to a blank string.
-----
Name                Command                State                Ports
-----
scenario3_ibb-op_1  iib_mq_manage.sh      Up                  0.0.0.0:32772->1414/tcp, 0.0.0.0:32771->4414/tcp, 0.0.0.0:32770->7800/tcp
scenario3_mq_1      mq.sh                  Up                  0.0.0.0:32769->1414/tcp, 0.0.0.0:32768->5672/tcp
scenario3_sgc_1     node lib/segwclient.js - ... Up
  
```

Figure 7-18 Port number

- b. Use the values in Table 7-4 to connect to the IBM Integration Bus instance from the IBM Integration Bus Toolkit.

Table 7-4 Property values

Property name	Property value	Comment
Host name	192.168.225.140	This is a sample value. Replace it with the host name of your own docker host.
Port	32771	This is a sample value. Replace it with the port number determined in the previous step.
Integration node name	NODE1	The integration node has already been set up for you with this name.
User name	anything	Administrative security has been disabled. Use any string as user name or password.
Password	anything	Administrative security has been disabled. Use any string as user name or password.

- c. Click **Finish**. The toolkit then connects and you should see that an integration server called IS01 has already been created for you.
2. Deploy the `iib_scenario3` application by dragging it down to IS01. After the deployment operation is complete, the integration server should look similar to Figure 7-19.

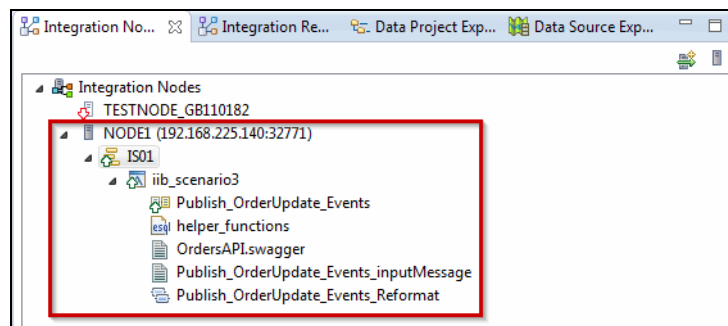


Figure 7-19 Deployed IBM Integration Bus application

3. To start and test the flow, complete the following steps:
  - a. Open a command prompt inside the docker container by running this command:

```
docker exec -it scenario3_iib-op_1 /bin/bash
```
  - b. To change to the iibusers home directory, run the **cd** command.
  - c. To save you the typing, the **curl** command to test the flow has been placed in a small script for you. Start it by typing **./testme** and pressing **Enter**. The result is shown in Figure 7-20.

```
(IIB_10:)iibuser@18a47fa981cb:/$ cd
(IIB_10:)iibuser@18a47fa981cb:~$ ./testme.sh
{"eventId":"20740446-8c84-11e6-b435-ac1500040000","time":"2016-10-07","type":"submitted","orderId":2,
"customerId":"AccountA","description":"Order contains 3 line items."}
```

Figure 7-20 Output from test script

This concludes the testing portion of this section.

## 7.2.2 Message Hub

To create the Message Hub service on IBM Bluemix, complete the following steps:

1. Go to the IBM Bluemix console at <https://new-console.ng.bluemix.net>, click **Catalog** at the top of the window and enter Message Hub in the **Search** field as shown in Figure 7-21.

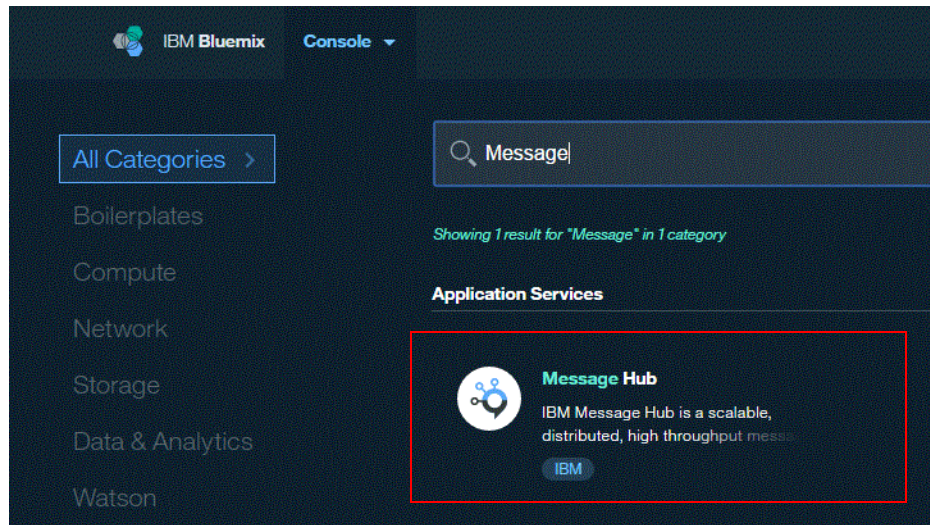


Figure 7-21 Searching Message Hub in Bluemix catalog

2. Click the **Message Hub** tile and enter Message Hub as the name of the service as shown in Figure 7-22.

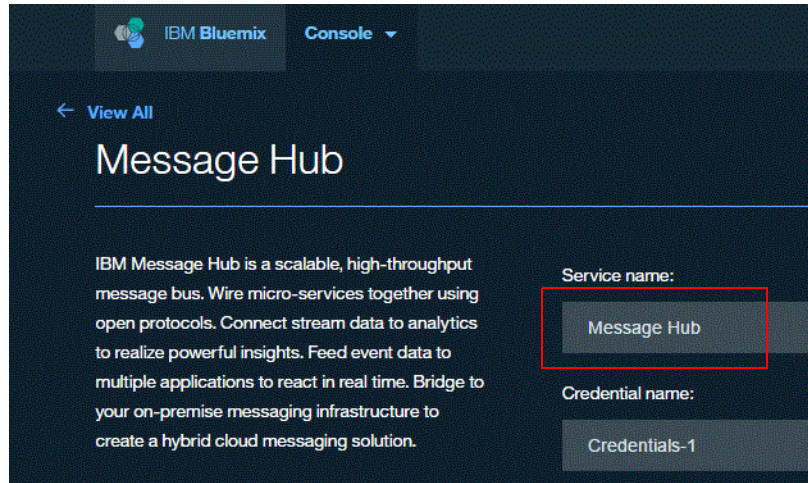


Figure 7-22 Configure the Message Hub service

3. Click **Create** to start the creation of the service as shown in Figure 7-23.

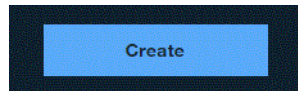


Figure 7-23 Create the Message Hub service

4. Wait for the service to be created and click the plus sign (+) to add a topic as shown in Figure 7-24.

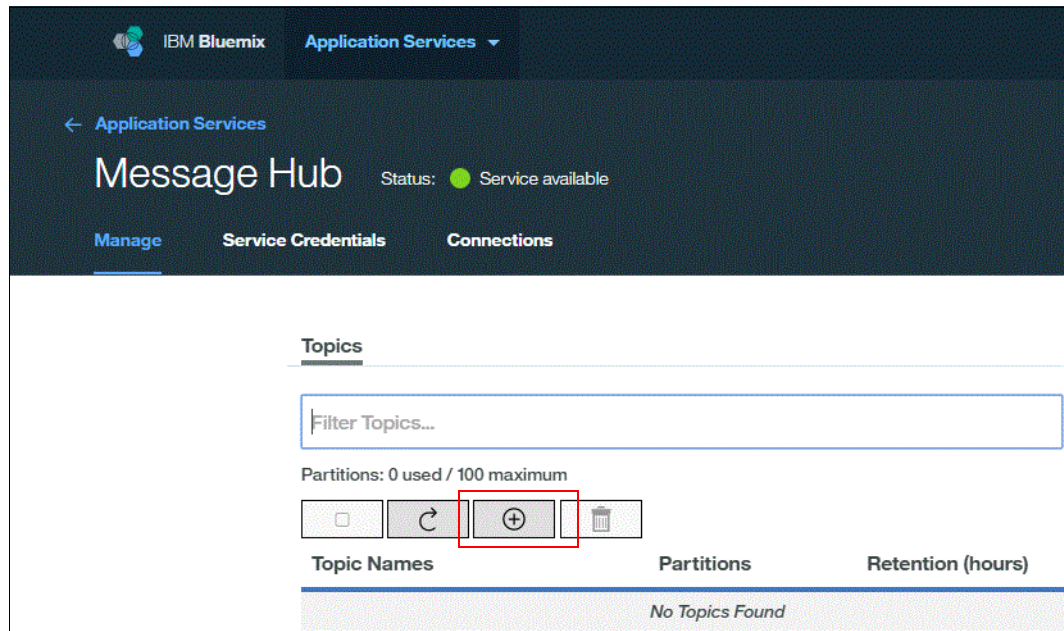


Figure 7-24 Add a topic

5. To configure the topic, enter the following values:

- mhtopic in the **Topic Name** field
- 1 in the **Partition** field
- 24 in the **Retention** field

Then click **Save** as shown in Figure 7-25.

Topic Names	Partitions	Retention (hours)	
<input type="text" value="mhtopic"/>	<input type="text" value="1"/>	<input type="text" value="24"/>	<input type="button" value="Save"/> <input type="button" value="X"/>

Figure 7-25 Configure the Message Hub topic

6. Check that the topic is properly created as shown in Figure 7-26.

Topic Names	Partitions	Retention (hours)	
Topic 'mhtopic' created. <input type="button" value="X"/>			
<input type="checkbox"/> mhtopic	1	24	

Figure 7-26 Message Hub topic created

## 7.2.3 MongoDB

This section covers the installation and configuration of MongoDB.

### Installation of MongoDB

To install MongoDB, use the MongoDB by Compose service in Bluemix and complete the following steps:

1. Go to the Compose website to create a trial account:

<https://www.compose.com/>

**Note:** If you already have a Compose account, jump to step 7 on page 176 to deploy a new MongoDB database.

2. Click **Register for a Free 30-Day Trial** as shown in Figure 7-27.

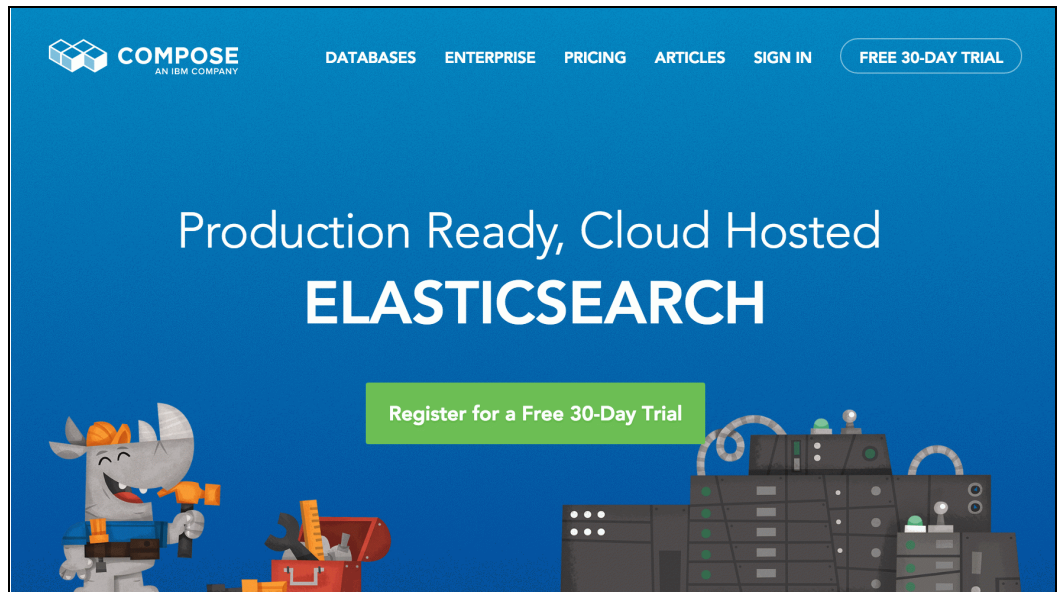


Figure 7-27 Compose website

3. Enter your personal information for the trial account creation as shown in Figure 7-28.

Enter Your Information	Choose a Database
Full Name <input type="text" value="Barney Rubble"/>	MongoDB
Company/Account Name <input type="text" value="Slaterock Gravel Co."/>	Elasticsearch
Email <input type="text" value="barney@slate.rock"/>	RethinkDB
Password <input type="text" value="Minimum of 8 characters"/>	etcd <b>BETA</b>
By clicking the button below, you agree to Compose's <a href="#">terms of service</a> .	Redis
	PostgreSQL
	RabbitMQ <b>BETA</b>
<input type="button" value="Enter Payment Information"/>	

Figure 7-28 Compose account creation form

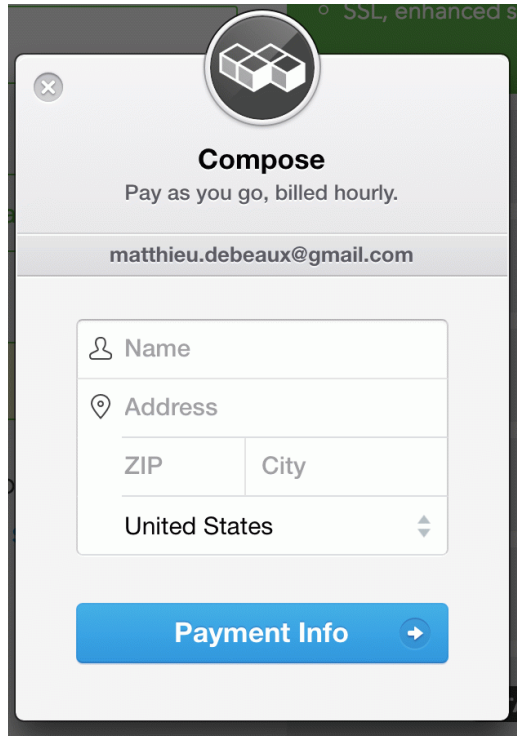
4. Select **MongoDB** in the **Choose a Database** column as shown in Figure 7-29.

Enter Your Information	Choose a Database
<p>Full Name</p> <input type="text" value="Matthieu Debeaux"/>	<p><b>MongoDB</b> ✓</p> <ul style="list-style-type: none"><li>◦ \$31/mo for the first 1GB, then \$18/GB</li><li>◦ Multiple databases per deployment</li><li>◦ Nightly backups included</li><li>◦ SSL, enhanced security features</li></ul>
<p>Company/Account Name</p> <input type="text" value="mdebeaux"/>	<p>Elasticsearch ✓</p>
<p>Email</p> <input type="text" value="matthieu.debeaux@gmail.com"/>	<p>RethinkDB ✓</p>
<p>Password</p> <input type="password" value="....."/>	<p>etcd <b>BETA</b> ✓</p>
<p>By clicking the button below, you agree to Compose's <a href="#">terms of service</a>.</p>	<p>Redis ✓</p>
	<p>PostgreSQL ✓</p>
	<p>RabbitMQ <b>BETA</b> ✓</p>

Figure 7-29 Compose account creation form completed



5. Click **Enter Payment Information** and enter your address in the window as shown in Figure 7-30.



**Compose**  
Pay as you go, billed hourly.

matthieu.debeaux@gmail.com

Name

Address

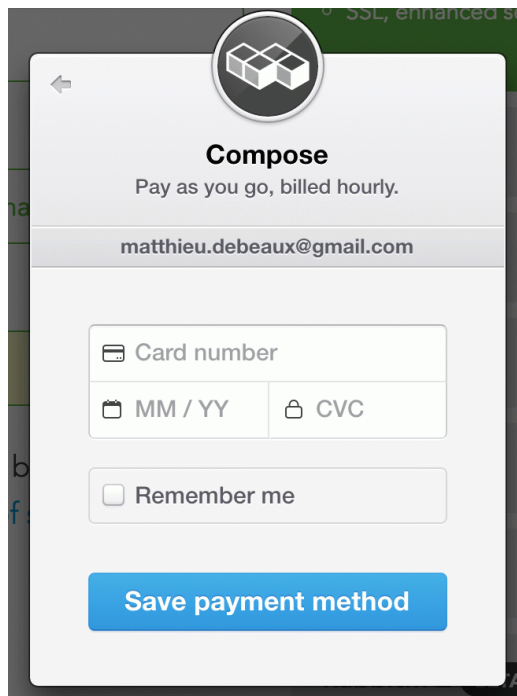
ZIP City

United States

**Payment Info** →

Figure 7-30 Compose address form

6. Click **Payment Info**, enter your credit card credentials, and click **Save payment method** as shown in Figure 7-31.



**Compose**  
Pay as you go, billed hourly.

matthieu.debeaux@gmail.com

Card number

MM / YY CVC

Remember me

**Save payment method**

Figure 7-31 Compose payment form

7. In the **New MongoDB Deployment** window, enter the following information as shown in Figure 7-32:
  - a. Enter OrderTrackerDB as **Deployment Name**.
  - b. Select **US Dallas 9** as **Location**.
  - c. Ensure that the **Enable SSL access** check box is selected.
  - d. Ensure that the **WiredTiger Storage Engine** check box is cleared.
  - e. Ensure that the **Allocate initial deployment resources** slider is showing 1 GB.

Please select your mongodb build options. ✕

### New MongoDB Deployment

Deployment Name  
OrderTrackerDB ↗

Location  
US Dallas 9 ▾

Enable SSL access

WiredTiger Storage Engine

Allocate initial deployment resources

1GB ○ ○ ○ ○ ○ 125GB ○

1GB Storage

Start with **1GB Storage / 102MB RAM** at **\$31.00/mo.**

[Create Deployment](#)

Figure 7-32 Compose MongoDB creation form

- Click **Create Deployment** and wait for the deployment to complete as shown in Figure 7-33.

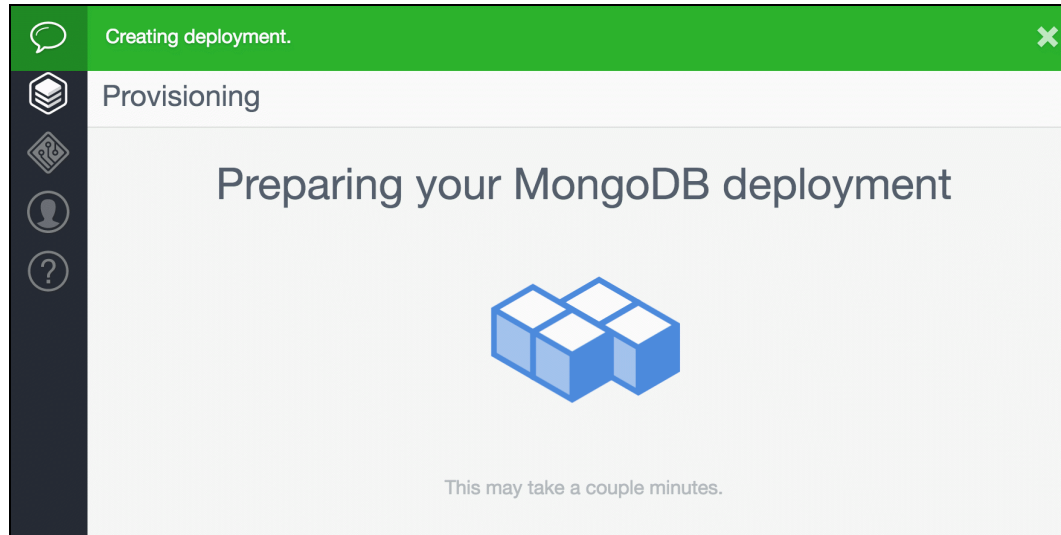


Figure 7-33 Compose MongoDB deployment

**Tip:** The deployment might take a while. However, this window can be closed without affecting the deployment.

## 7.2.4 API Connect

This section shows you how to create the Order Tracker application in API Connect.

### Application overview

The Order Tracker application is built with API Connect using the LoopBack Framework. Loopback is an open source Node.js framework built around the concept of *models*. Models are descriptions of the core business entities that you want to expose through REST interfaces as shown in Figure 7-34.

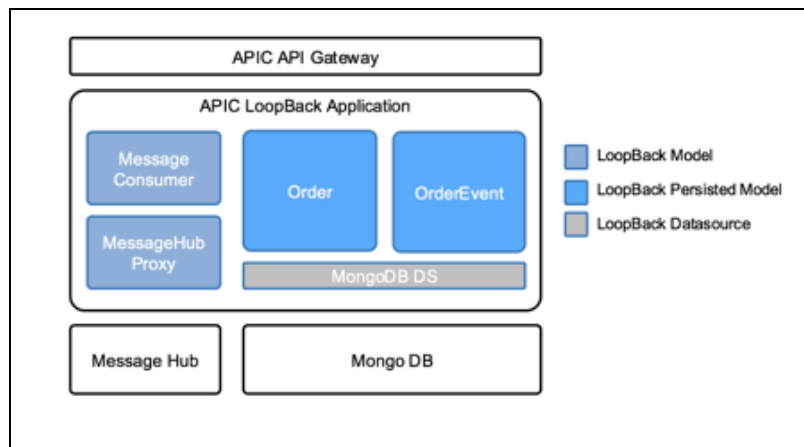


Figure 7-34 Microservice architecture

Models have properties, which are the data associated to the entity. In this case, for example, “Order” has an “orderId,” “customerId,” and “description.” LoopBack *connectors* make persisting these data in a database, like Mongo DB, transparent to the application developer. A connector is configured to point to a specific database instance by defining a *DataSource*.

A key benefit of using LoopBack is the automatic generation of the APIs exposing the data of the models. These APIs are documented in Swagger files and can be deployed and secured on the API Connect Gateway.

Business entities are often related to each other. For example an “Order” can have many “Order Events.” For this reason, LoopBack supports the concept of “relationship” and creates APIs to navigate from one entity to the other.

Not all the models are persisted, however. For this application, the “Message Consumer” model only exposes APIs to start the consumption of messages from Message Hub and does not need to maintain a persisted state. However, even in this case, LoopBack provides easy constructs to create APIs and their Node.js implementation.

Working with Node.js gives you access to an extensive library of pre-tested packages that you can download from the public registry () and reuse in your application. The registry is available at this URL:

<https://www.npmjs.com/>

For example, this scenario uses a preexisting package (message-hub-rest) to access the message hub with a few JavaScript line of code. This package is calling, behind the scenes, Kafka REST APIs to interact with Message Hub. For this book we chose to speed the implementation of the scenario. However, for production implementations, use Node.js packages that use the native Kafka protocol to optimize performance and reliability. JavaScript clients for Kafka are maturing at the time of writing, but examples are available at these websites:

<https://github.com/Blizzard/node-rdkafka>

<https://github.com/oleksiyk/kafka>

## Creating the Order Tracker application

This section walks you through the key steps you need to complete to build the Order-Tracker application. Note that this is application development, not just configuration, so for brevity some of the steps are described only at the high level. If you need more details, you can refer to the application source code published here.

This chapter also assumes that you have installed the API Connect toolkit on your developer workstation. See 4.3.1, “Prerequisites” on page 62 for instructions on how to set it up.

For clarity, the development steps are organized in four phases:

- ▶ Application scaffolding and dependencies installation
- ▶ Models and data sources configuration
- ▶ Business logic development
- ▶ Application deployment and test

Each phase contains the coverage of all the application elements so that you can have an overview of what needs to be done. However, during actual development, you must follow a much more iterative approach in which you deploy and test a new component or piece of logic as soon as you have created it. Also, because the focus of this section is the micro-service implementation, more than securing the APIs on the gateway, leave the API exposure configuration settings of the application at their defaults. In an enterprise-level implementation, you will probably want to group the APIs that create and modify Orders and

OrderEvents in a dedicated API product accessible only to users with admin privileges, while mobile developers who work on the Order Tracker mobile app can subscribe only to endpoints that read, but cannot modify, data.

### **Application scaffolding and dependencies installation**

Use the API Connect command line to create the structure of the application and install the required npm packages. In particular, complete these steps:

1. Run **\$apic loopback** to create the basic structure of the application. On the command line, specify application name: order-tracker and application type: empty server.
2. Within the application directory, run the following commands to install the required dependencies:

```
$npm install --save loopback-connector-mongodb  
$npm install --save message-hub-rest
```

To get more information about how AP Connect command line works, you can run the **\$apic -h** command and access its documentation.

### **Models and data sources configuration**

You can continue and create models and data sources with the command line, but this section show how you can do that with API Designer to have a visual representation of your entities:

1. Start API Designer by running **\$apic edit**.
2. Create an Order model with the properties that are shown in Figure 7-35.

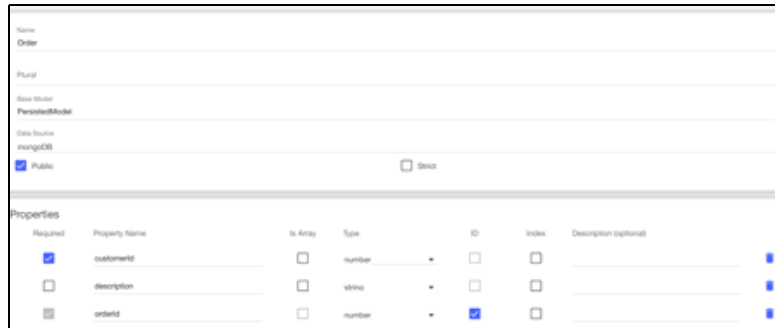


Figure 7-35 IBM API Connect order model creation

3. Create an OrderEvent model with the properties that are shown in Figure 7-36.

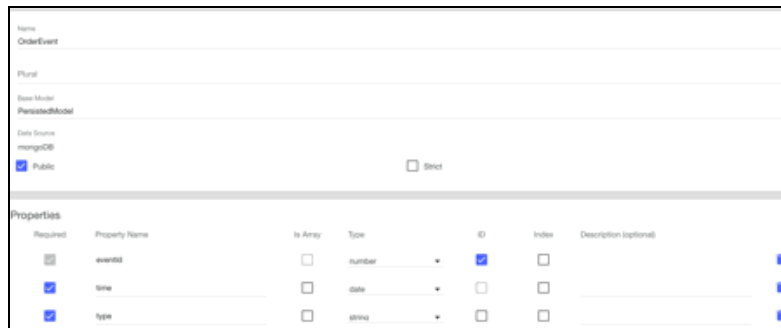


Figure 7-36 IBM API Connect order event model creation

4. Create the MessageConsumer and MessageHub Proxy models. For both of them, set the **Base Model** property to **Model** (you do not need to persist them) as shown in Figure 7-37.

The screenshot shows a configuration window for a model. The 'Name' field is set to 'MessageHubProxy'. The 'Plural' field is empty. The 'Base Model' dropdown menu is set to 'Model'.

Figure 7-37 IBM API Connect Message Hub creation

5. Create a data source for your Mongo database. In the properties, specify the following as shown in Figure 7-38.

- **Name:** mogo
- **Connector:** MongoDB
- **URL:**  
`mongodb://<user>:<password>@<server>:<portnumber>/<databasename>?ssl=true`  
 Replace the values in brackets with the ones relevant for your environment.

The screenshot shows a configuration window for a data source. The 'Name' field is set to 'mogo'. The 'Connector' dropdown menu is set to 'MongoDB'. The 'url' field contains the string: `mongodb://sampleuser:samplepwd@server:portnumber/databasename?ssl=true`.

Figure 7-38 IBM API Connect MongoDB data source creation

6. Go back to the Order and OrderEvent models, and associate them to the data source that you have just created.
7. Use the API Connect command line to create a relationship between Order and OrderEvents by issuing this command:

```
$apic loopback:relation
```

Then, select the following options:

- Model to create a relationship from: Order
- Relationship type: has many
- Model to create a relationship with: OrderEvent
- Press **Enter** on all the other options and leave them to their defaults

## Business logic development

Now you need to write the custom logic to read events from Message Hub, map them to the Order and OrderEvent models (the structures exposed by the REST API of the microservice) and persist the data in Mongo. You also must create an additional API endpoint to start the consumption of the messages.

While you were working with the API Connect command line and user interface in “Models and data sources configuration” on page 179, the tool created a Node.js app behind the scenes. You are now going to extend its behavior.

Look at the folder structure of the directory in which you have been working. You should find the artifacts that are shown in Figure 7-39.

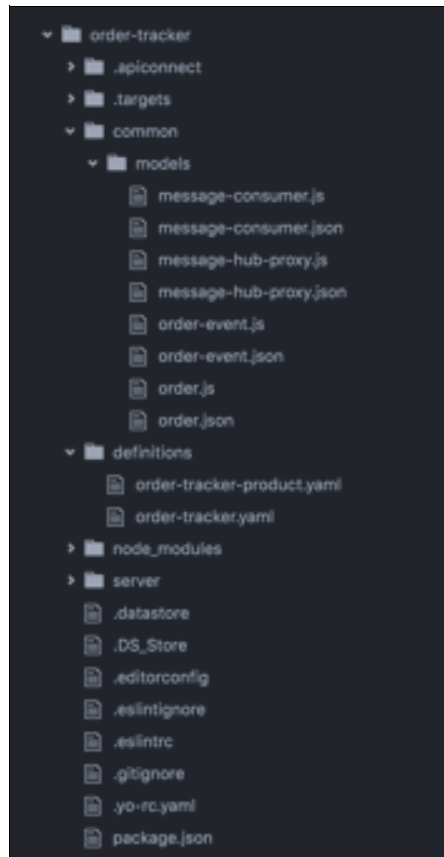


Figure 7-39 Microservice directory structure

Your API and models are contained in the `definitions` and `models` folders. The first one includes the Swagger documentation of the APIs, and the second has the definition of the models in `.json` files and the code customizing their behavior in the `.js` files.

187

**Tip:** If you are not familiar with Node.js development, you can import the version of these two files from GitHub and then run `$apic loopback:refresh` to align the interfaces with the customized models.

Extend `message-hub-proxy.js` and `message-consumer.js` as explained below:

1. In `message-hub-proxy.js`, reference the `message-hub-rest` package (line 2 in Figure 7-40 on page 182) and then initialize a Message Hub client, passing the connection details of the Message Hub instance you want to reach in Figure 7-40.

```
var MessageHub = require('message-hub-rest');
var events = require('events');

module.exports = function(Messagehubproxy) {

  var defaultProperties =
  {
    "messagehub": [
      {
        "name": "Message Hub-hk",
        "label": "messagehub",
        "plan": "standard",
        "credentials": {
          "mqlight_lookup_url": "https://mqlight-lookup-prod02.messagehub.services.eu-gb.ibm.com/lookup?serviceId=",
          "api_key": "176NACf78WCnpWlp3owfezyiYmAnF916XWpTz2HVcxy1FQ",
          "kafka_admin_url": "https://kafka-admin-prod02.messagehub.services.eu-gb.ibm.com:443",
          "kafka_rest_url": "https://kafka-rest-prod02.messagehub.services.eu-gb.ibm.com:443",
          "kafka_brokers_sasl": [
            "kafka01-prod02.messagehub.services.eu-gb.ibm.com:9093",
            "kafka02-prod02.messagehub.services.eu-gb.ibm.com:9093",
            "kafka03-prod02.messagehub.services.eu-gb.ibm.com:9093",
            "kafka04-prod02.messagehub.services.eu-gb.ibm.com:9093",
            "kafka05-prod02.messagehub.services.eu-gb.ibm.com:9093"
          ],
          "user": "xxxx",
          "password": "xxxx"
        }
      }
    ]
  }

  var services = process.env.VCAP_SERVICES || defaultProperties;
  this.hubInstance = new MessageHub(services);
}
```

Figure 7-40 Microservice `message-hub-proxy.js`

2. Then, create a function that polls the messaging system and reads new messages as shown in Figure 7-41.

```
Messagehubproxy.startConsumer = function (topic, cb){
  var consumer;
  var receivedMessage = 0;
  var mhEventEmitter = new events.EventEmitter();
  mhEventEmitter.on('error',function(err){
    console.error(err);
  });
  cb(null,mhEventEmitter);
  hubInstance.consume('order-tracker-group', 'order-tracker-cl', {'auto.offset.reset':'largest'})
  .then(function(response) {
    consumer = response[0];
    var consumeInterval = setInterval(function () {
      consumer.get(topic)
      .then(function(data){
        receivedMessage++;
        mhEventEmitter.emit('order-event',data);
      })
      .fail(function(error){
        console.error(error);
      })
    }, 10000);
  })
};
```

Figure 7-41 Microservice `message-hub-proxy` function



3. In message-consumer, add the logic that starts the polling, receive the data read from Message Hub, and save it to the database as shown in Figure 7-42. Note the following characteristics:
  - The communication between this component and the previous one happens asynchronously, through the Node.js event framework.
  - To create records in the database, you just need to call the `create()` function of a `PersistedModel`, LoopBack does the rest for you behind the scenes, by using the data source you have associated with the model.

```

Messageconsumer.startPolling = function(callback) {
  Messageconsumer.app.models.MessageHubProxy.startConsumer("order_events",function(err, emitter){
    callback(err, "polling about to start...");
    emitter.on('order-event',function(data){
      console.log("data received by the emitter: " + data);
      for(var i=0; i<data.length; i++){
        myevent = JSON.parse(data[i]);
        newOrder = {
          customerId : myevent.customerId,
          description : myevent.description,
          orderId : myevent.orderId,
        };
        newEvent = {
          orderId : myevent.orderId,
          eventId : myevent.eventId,
          time : myevent.time,
          type : myevent.type,
        };
        Messageconsumer.app.models.Order.findById(myevent.orderId,function(err,instance){
          if (err){
            emitter.emit('error',err);
          } else {
            if (!instance){
              Messageconsumer.app.models.Order.create(newOrder, function(err, object){
                console.log("new order created: " + JSON.stringify(object));
              });
            }
            Messageconsumer.app.models.OrderEvent.create(newEvent, function(err, object){
              console.log("new order-event created: " + JSON.stringify(object));
            });
          }
        });
      }
    });
  });
};

```

Figure 7-42 Microservice message-consumer.js

4. In this component, you also define a “remote method” exposing the `startPolling` function as an API as shown in Figure 7-43.

```

1
2
3  Messageconsumer.remoteMethod('startPolling',
4    {
5      http: {path:'/start', verb:'post'},
6      returns: {arg:'state',vtype:'object'}
7    }
8  );
9

```

Figure 7-43 Microservice message-consumer method

**Note:** To surface this new operation in the Swagger files that are published on the API gateway, you must run the command `$apic loopback:refresh`. This command makes sure that the interfaces are updated with what is defined in the models.

## Application deployment and test

Use API Designer to test and then deploy the order-tracker microservice:

1. Start API designer with `$apic edit`.
2. Start the server by clicking the **arrow button** at the lower left of the screen. This action starts two Node.js processes on your computer:
  - One for the micro-gateway
  - One for your LoopBack application
3. Click the Explore tab at the top of the screen to test the interface of the microservice as shown in Figure 7-44.

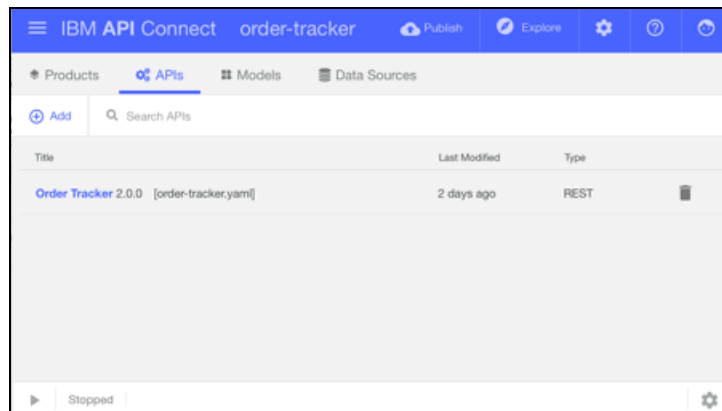


Figure 7-44 IBM API Connect Designer

4. You can now deploy the solution to Bluemix:
  - a. Click **Publish** at the top of the screen and select **Add and manage targets** to specify where you want to deploy your code.
  - b. Select **Add IBM Bluemix Target** and then specify the Bluemix region, organization, and catalog where you want to deploy the APIs of the order-tracker app.
  - c. Click **Next**.
  - d. Enter a new application name (order-tracker), click the (+) button to add it to the list, and then click **Save**.
  - e. Back on the main API Designer UI, click **Publish** again and select the deployment target that you just defined. On the window that opens select **Publish Application** and then click **Publish**. API Designer then publishes the LoopBack application to a Cloud Foundry run time on Bluemix and updates the properties of the Swagger files to point to that instance.
  - f. Repeat the steps of the previous bullet point, but now select **Stage and Publish** and then click **Publish** to deploy the API definition on the API Gateway on Bluemix.

If you browse to your API Manager UI in Bluemix, notice that an order-tracker app has been added to your dashboard as shown in Figure 7-45.

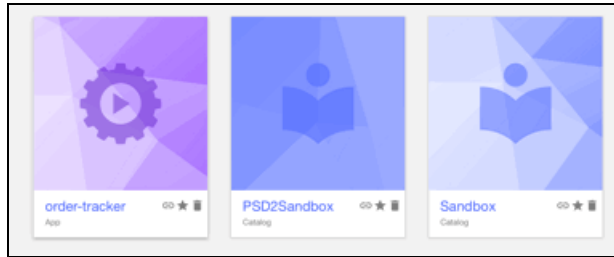


Figure 7-45 IBM API Connect catalog

5. Click the tile of the app and then the **Manage this app** link to go to the Node.js management interface of the Node.js run time where the Loopback application is running as shown in Figure 7-46.



Figure 7-46 Microservice management in Bluemix

6. Back on the API Manager UI, you can select the Explore tab and test the APIs on Bluemix, or go to the Developer Portal of the catalog and subscribe to them.

## 7.2.5 Dashboard app

This section describes the implementation of the dashboard application.

## Application overview

The order-tracker mobile application is built with the Ionic framework, an open source SDK for hybrid mobile app development built on top of Apache Cordova and AngularJS. The mobile app accesses the REST endpoints exposed by API Connect to obtain details of orders and order events as shown in Figure 7-47.

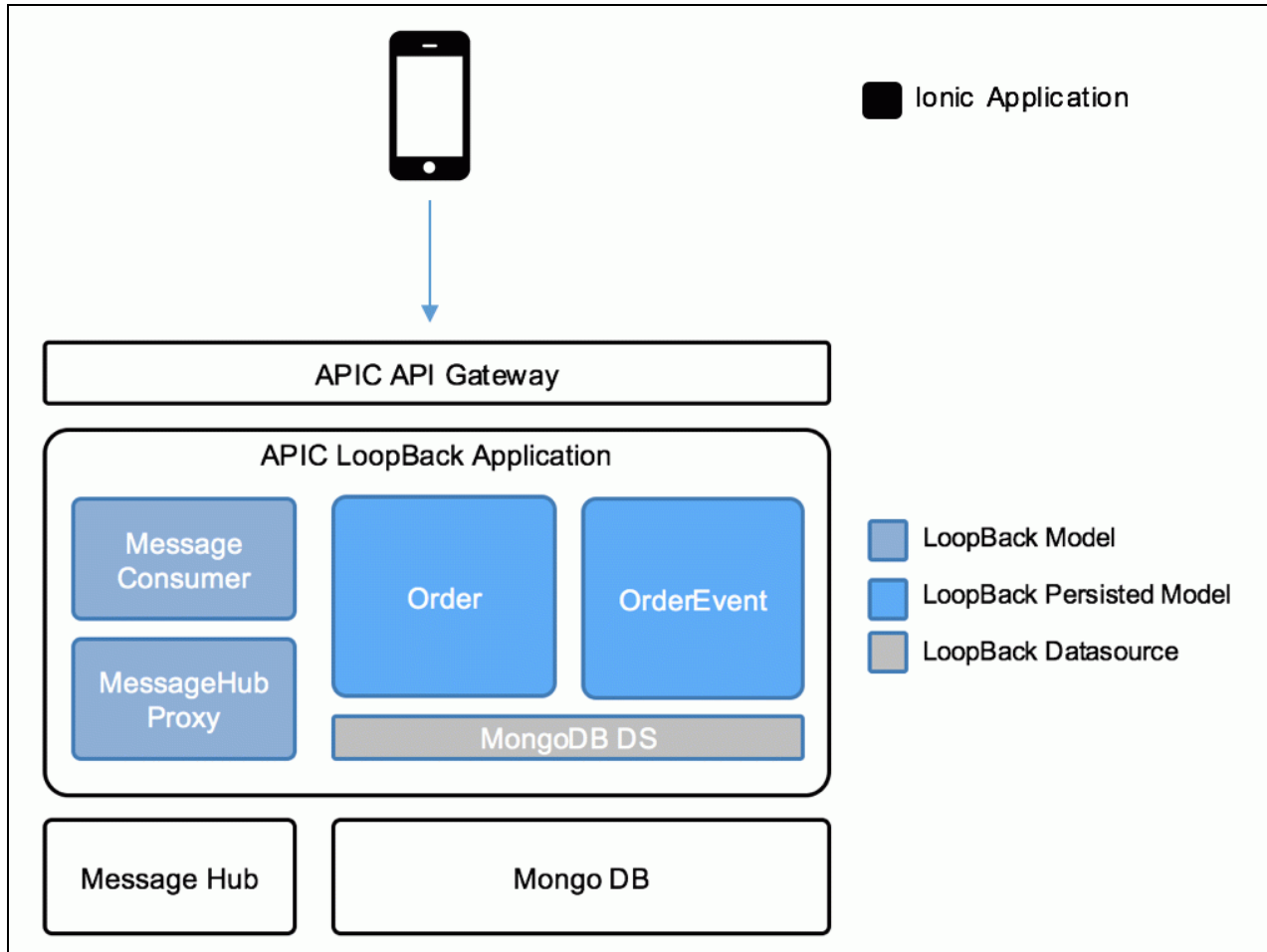


Figure 7-47 Application overview diagram

## Installing Ionic

Ionic requires Cordova. Install both by using the Node Package Manager (npm). Therefore, it is a prerequisite to have Node.js and npm installed on your developer computer. You can download Node.js from this locations:

<https://nodejs.org/en/download/>

After Node.js and npm are installed, run the following command to install Cordova:

```
sudo npm install -g cordova
```

**Note:** Remove **sudo** from the commands if you use Windows.

To install the Ionic framework, run this command:

```
sudo npm install -g ionic
```

## Creating an Ionic application

This Ionic application that is used in this example has a side-menu type design. To create a new application from a pre-made “side-menu” template, go to a base directory of your choice and run this command:

```
ionic start myApp sidemenu
```

This process gives you a basic Ionic application that can be run by the following command:

```
ionic serve
```

This command runs the application in the default web browser. We also look at running the application using an iOS or Android emulator and also how to deploy it to a stand-alone Android device later in this chapter.

The previous step created the Ionic project scaffolding for you, the key folders and files of which are shown in Figure 7-48.

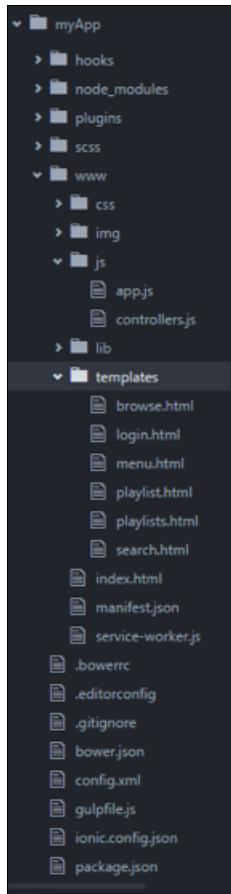


Figure 7-48 Ionic application file structure

The key files and folders are listed here:

- ▶ `index.html` (file): Where the view of the application is initialized and the JavaScript and stylesheets that are used by the application are set.
- ▶ `www > js` (folder): Contains the following files:
  - `app.js` (file): The location where the states of the app are controlled and what templates are associated with what logic.
  - `controllers.js` (file): Where the main logic for the app is written.
- ▶ `templates` (folder): Where the HTML templates for each of the application screens are stored.
- ▶ `css` (folder): Where the stylesheets that are referenced by the app are stored.

## Running the Order Tracker Ionic application

For this chapter, an Ionic application has been created. The source code is available in the GitHub repository `scenario3`.

To run the application, complete the following steps:

1. Go to `scenario3` folder that you cloned from GitHub in 7.2.1, “IBM Integration Bus: Exposing enterprise order events” on page 157.
2. Go to `3_ionic` folder to find the Ionic project content.
3. Start the application by running `ionic serve`.
4. Open a web browser and go to `localhost:8100` to find the running application as shown in Figure 7-49.

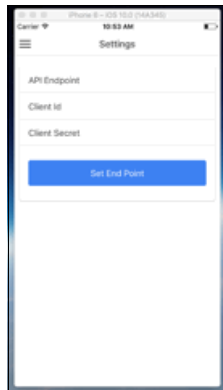


Figure 7-49 Order tracker application settings

5. Fill the required information as displayed in Figure 7-50 as follows:
  - a. API Endpoint using the IBM API Connect endpoint created in 7.2.4, “API Connect” on page 177.
  - b. Client ID with the ID generated by IBM API Connect.
  - c. Client Secret with the secret generated by IBM API Connect.

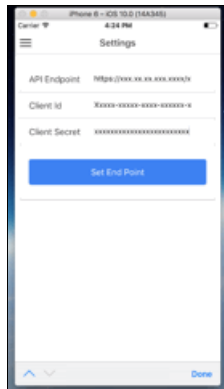


Figure 7-50 Order tracker application settings filled

6. Click **Set End Point** to validate the settings.
7. After it is configured, the Orders screen is displayed by using the REST API exposed by the Loopback application to retrieve all orders as shown in Figure 7-51.

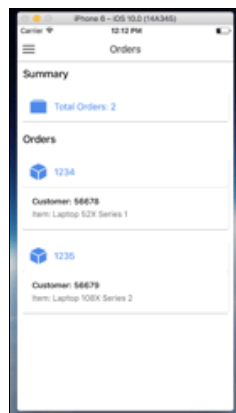


Figure 7-51 List of orders

8. Select an order to display the Order Details screen.

9. The Orders screen uses the REST API exposed by the Loopback application to retrieve all events related to the order selected as shown in Figure 7-52. Specialist CSS styling is used to display the order events in a timeline with specific icons for each event type.

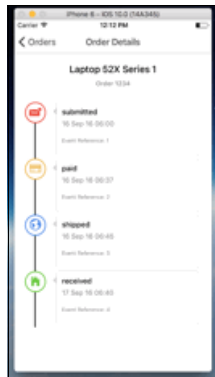


Figure 7-52 Order details

### Testing the Ionic application

The quickest way to test the application is to run the following command and test all functions from the web browser:

```
ionic serve
```

If you are running on a Mac and have Xcode installed, you can install the iOS dependencies and run the app in the iOS emulator by using the following commands:

```
ionic platform add ios
ionic build ios
ionic emulate ios
```

If you are running on another platform such as Windows and have Android Studio and SDKs installed, then replace `ios` in these commands with `android`.

If you have an Android device attached to your computer and the device is in USB debugging mode, then you can run the app on your device by running this command:

```
ionic run android
```

The final option is to install the apk provided on your Android device directly.

## 7.3 Technical implementation of variant C

This section shows how to implement variant C by going through eight steps in the order that is displayed in Figure 7-53 on page 191.

**Note:** Some steps of this variant are identical or similar to steps in variant B. For those instructions, a reference points you to the correct step in 7.2, “Technical implementation of variant B” on page 157.



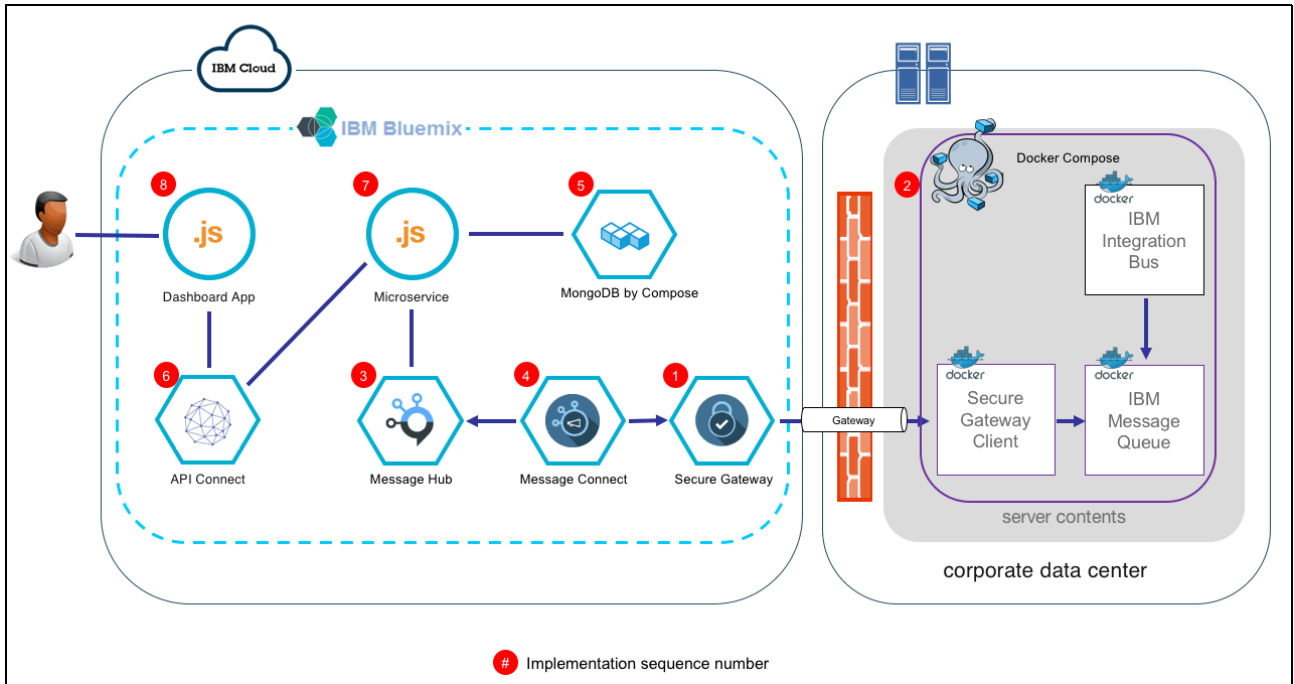


Figure 7-53 Scenario 3 technical diagram, variant C

### 7.3.1 Secure Gateway

The Secure Gateway creates a secure tunnel between the Message Hub service on Bluemix and the on-premises IBM MQ. To use it, complete these steps:

1. To create the service, go to IBM Bluemix console (<https://new-console.ng.bluemix.net>), click **Catalog**, and search for the Secure Gateway service in the **Search** field as shown in Figure 7-54.

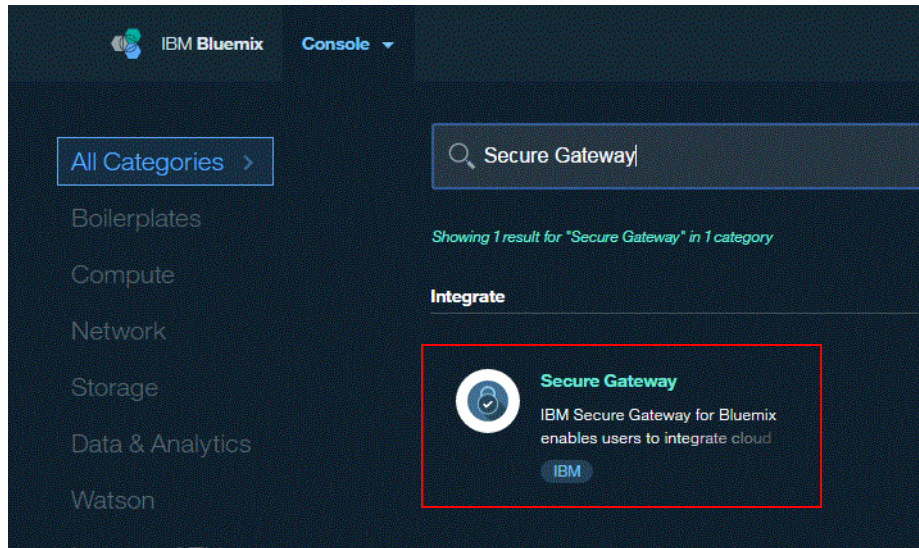


Figure 7-54 Search the Secure Gateway service in Bluemix catalog

2. Click the **Secure Gateway** tile to display the service creation page as shown in Figure 7-55.

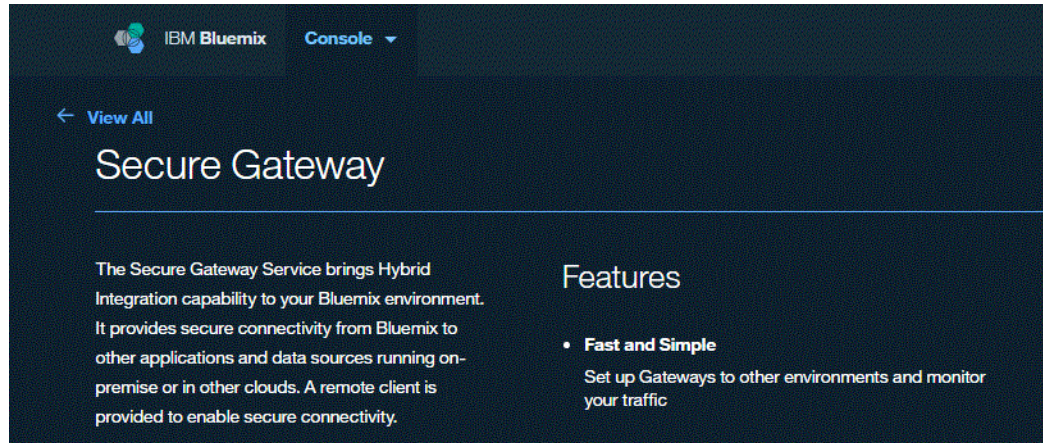


Figure 7-55 Secure Gateway service details page

**Note:** If you already have a Secure Gateway service installed, Bluemix indicates that you are limited to one Secure Gateway per space as shown in Figure 7-56.

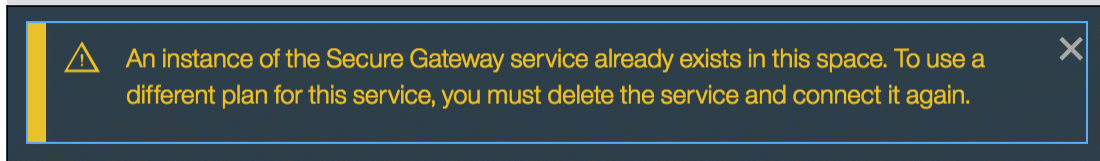


Figure 7-56 Secure Gateway notification if the service exists

In that case, skip the Secure Gateway creation steps and continue with the Secure Gateway configuration at Step 4 on page 193.

3. Click **Create** to start the Secure Gateway service creation as shown in Figure 7-57.

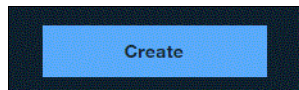


Figure 7-57 Secure Gateway service creation

4. After the service is created, configure it by creating a gateway. To do so, click **ADD GATEWAY** as shown in Figure 7-58.

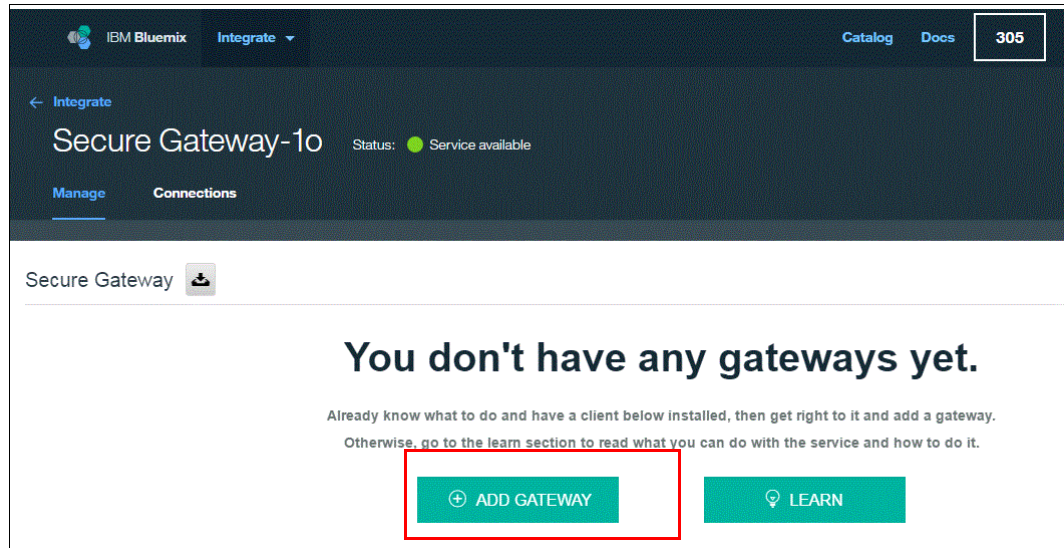


Figure 7-58 Secure Gateway configuration

5. Enter Scenario3 as the Gateway name and click **ADD GATEWAY** as shown in Figure 7-59.

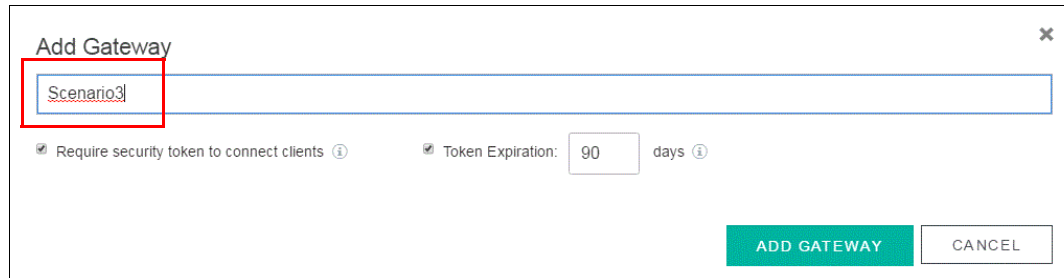


Figure 7-59 Create a gateway

- After it is created, the gateway is displayed on the Secure Gateway Dashboard. Click the **Cog** button to access the details of the gateway as shown in Figure 7-60.

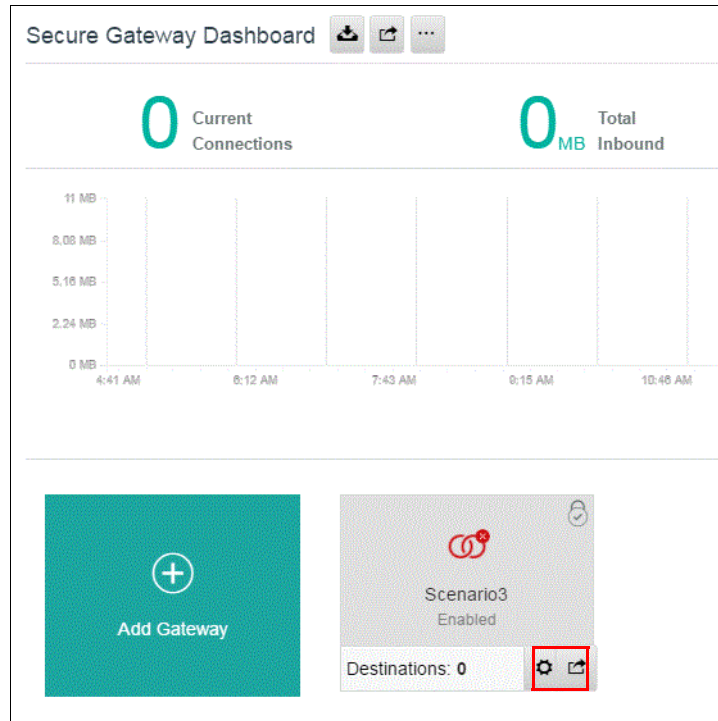


Figure 7-60 Secure Gateway Dashboard

- Copy the **Security token** and **Gateway ID** displayed on the Gateway details window. These items are used to connect to this Gateway from the Secure Gateway Client as shown in Figure 7-61.

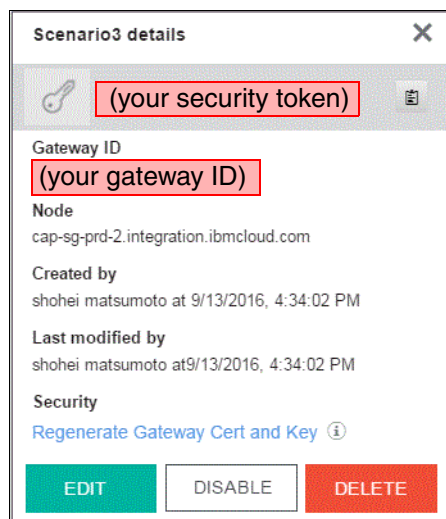


Figure 7-61 Security Token and Gateway ID

8. Close the details window, then click the **Scenario3** gateway that you just created, and click **Add Destination** as shown in Figure 7-62.

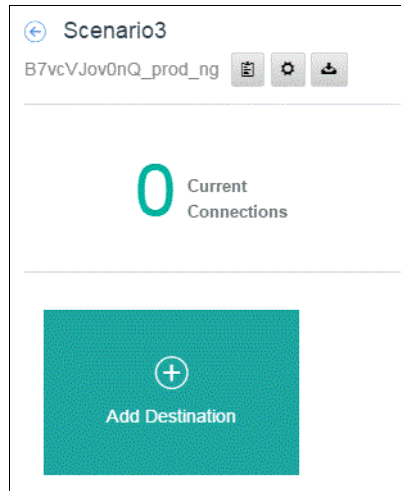


Figure 7-62 Add a destination to Secure Gateway

9. For this variant, enable the Message Connect service in Bluemix to connect to the IBM MQ on premises. Therefore, on the **Add Destination** window, select **On-Premises**, and click **Next** as shown in Figure 7-63.

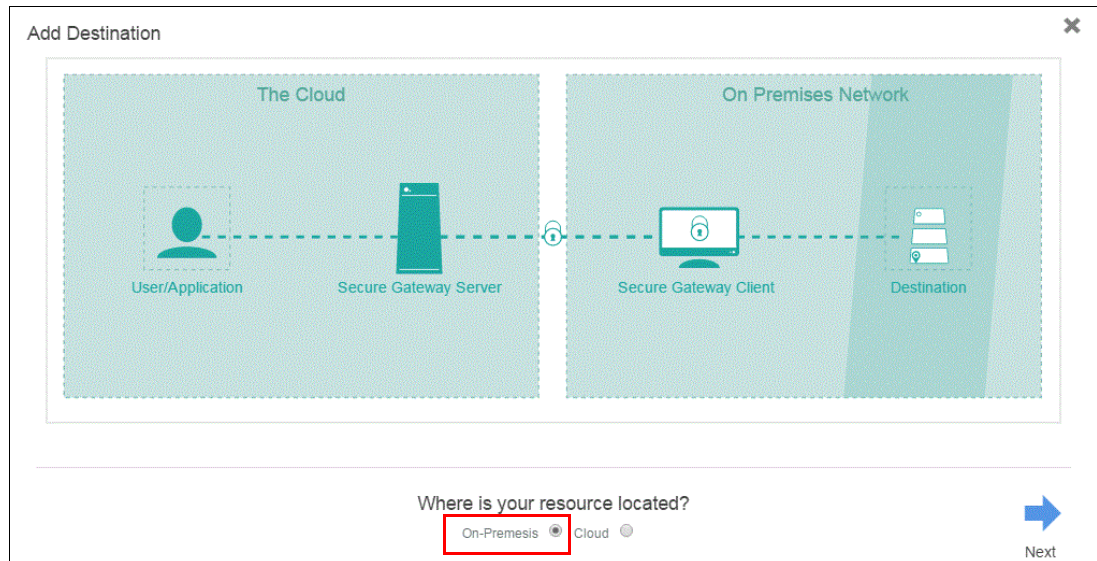
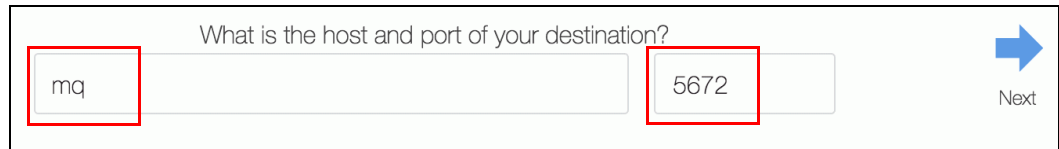


Figure 7-63 Add destination configuration

10. Specify the **host** and **port** of your destination. This information is used by the Secure Gateway Client to connect to the destination, the IBM MQ queue manager. IBM MQ runs in a Docker container which has the host name `mq` and the exposed port 5672. Fill in the **host** and **port** fields with the appropriate values as shown in Figure 7-64 and click **Next**.



What is the host and port of your destination?

Next

Figure 7-64 Destination host and port setting

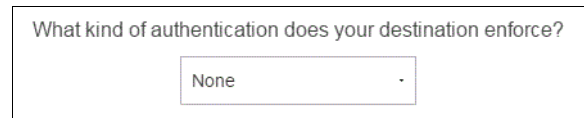
11. The flow between Message Connect and IBM MQ uses the AMQP protocol, which is based on TCP. Therefore, select TCP for the protocol to connect to the destination as shown on Figure 7-65 and click **Next**.



What protocol will the User/Application use to connect to your destination?

Figure 7-65 Protocol setting

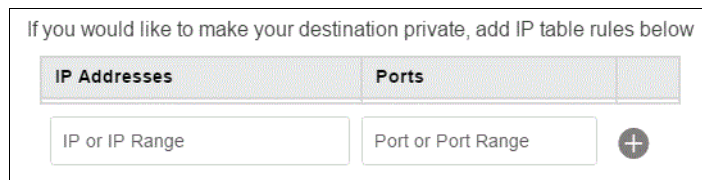
12. Leave the default **Authentication** setting at None as shown in Figure 7-66 and click **Next**.



What kind of authentication does your destination enforce?

Figure 7-66 Authentication setting

13. Leave the IP table rules settings empty and click **Next** (Figure 7-67).



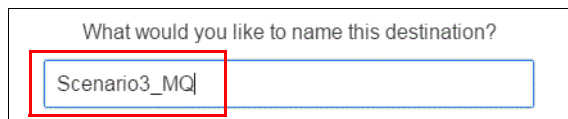
If you would like to make your destination private, add IP table rules below

IP Addresses	Ports
<input type="text" value="IP or IP Range"/>	<input type="text" value="Port or Port Range"/>

+

Figure 7-67 IP table rules setting

14. Enter `Scenario3_MQ` as the **name** of the destination as shown in Figure 7-68 and click **Finish**.



What would you like to name this destination?

Figure 7-68 Name the destination

15. After the new destination is created, it is displayed in the Gateway Dashboard as shown in Figure 7-69.

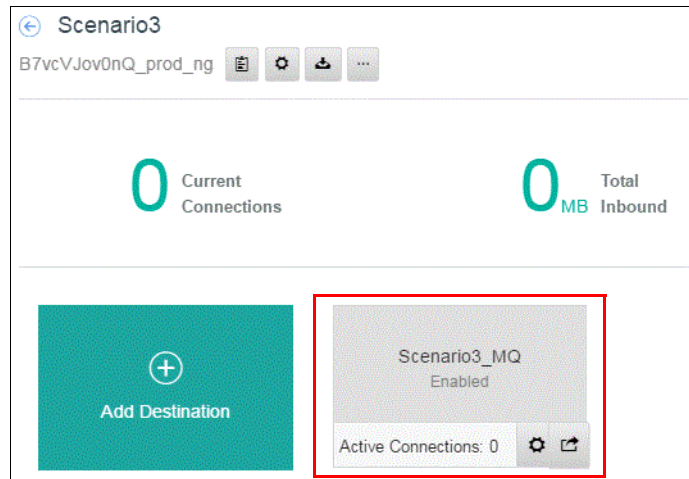


Figure 7-69 Created destination

### 7.3.2 Docker Compose (IBM Integration Bus, IBM MQ, Secure Gateway)

For IBM Integration Bus, IBM MQ and the Secure Gateway Client creation, use Docker Compose to create every component and the networking between each component.

The basis of the configuration for the variant is the implementation described in 7.2.1, “IBM Integration Bus: Exposing enterprise order events” on page 157. The instructions assume that you have already completed those steps.

The main difference in this variant is to publish the order update event through IBM MQ rather than directly into Message Hub as shown in Figure 7-70.

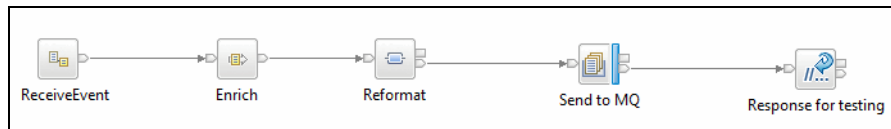


Figure 7-70 Flow overview for this variant

The required IBM MQ configuration to receive the event from IBM Integration Bus, namely an alias queue pointing to a topic, has already been prepared as part of the Docker container image. The necessary changes to the message flow that processes the event are described in the steps below.

#### Build

Complete the following steps:

1. Open the `Publish_OrderUpdate_Events` flow in the IBM Integration Bus Toolkit and remove the Kafka node.
2. Add the MQ Output node and wire it between the Mapping node and the HTTP Reply node in the same way as you wired the Kafka Producer node.

3. Configure the properties as shown in the Table 7-5.

Table 7-5 Properties table

Tab name	Property name	Property value	Comment
Description	Node name	Send to MQ	
Basic	Queue name	ORDER.UPDATES.TOPIC.ALIAS	Alias queue pointing to the existing topic called MQ.ORDER.UPDATES
MQ Connection	Connection	MQ client connection properties	Select from the menu.
MQ Connection	Destination queue manager name	EVENTQM	The Docker-based queue manager, preconfigured and part of the environment.
MQ Connection	Queue manager host name	mq	
MQ Connection	Listener port	1414	MQ security has been disabled and the listener configured and started.
MQ Connection	Channel name	SYSTEM.DEF.SVRCONN	

## Test

The testing steps are the same as already documented for the preferred solution.

In addition to the command line output of the test script, MQ Explorer can also be used to create a test subscription for the topic. This technique provides an easy way to debug potential problems in the Message Connect configuration because it shows whether messages are actually published and to which IBM MQ topic.

In the container, version 9 of IBM MQ is used. MQ Explorer needs to be at least the same version to connect.

The following steps describe the configuration for this optional activity:

1. The output of the **docker-compose ps** command on the Docker host provides the necessary information for connecting with IBM MQ Explorer. In particular, it shows the port number to which mq port 1414 has been mapped on the docker host. Figure 7-71 shows an example.

```

vmuser@ubuntu:~/dev/sgz48351/scenario3$ docker-compose ps
WARNING: The SC3_SQMID variable is not set. Defaulting to a blank string.
WARNING: The SC3_SECTOKEN variable is not set. Defaulting to a blank string.
-----
Name                                Command                                State                                Ports
-----
scenario3_iib-op_1                    iib_mq_manage.sh                       Up                                0.0.0.0:32799->1414/tcp, 0.0.0.0:32798->4414/tcp, 0.0.0.0:32797->7800/tcp
scenario3_mq_1                        mq.Sh                                   Up                                0.0.0.0:32796->1414/tcp, 0.0.0.0:32795->5672/tcp
scenario3_sgc_1                        node lib/secgwclient.js - ...           Up

```

Figure 7-71 Docker port mapping



2. Connect to a remote queue manager in MQ Explorer. Apart from the host name and port, the following settings should be used:
  - Queue manager name: EVENTQM
  - Server-connection channel: SYSTEM.DEF.SVRCONN
  - No user identification or password are required. The respective settings have been disabled in the container.
3. After connecting, right-click the MQ.ORDER.UPDATES topic and select **Test subscription** as shown in Figure 7-72.

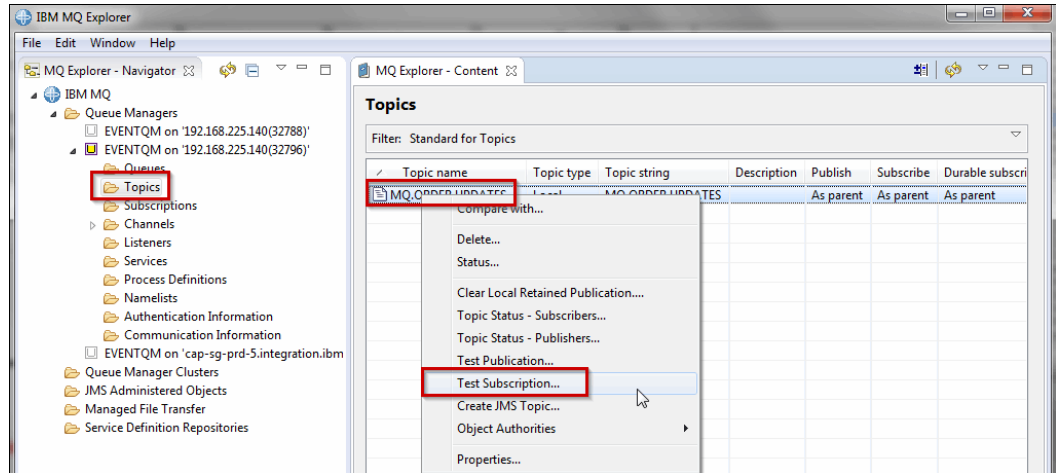


Figure 7-72 Test subscription

4. When using the `testme.sh` script, you can control when an event is published and what the contents of the message are exactly. The output should look similar to Figure 7-73.

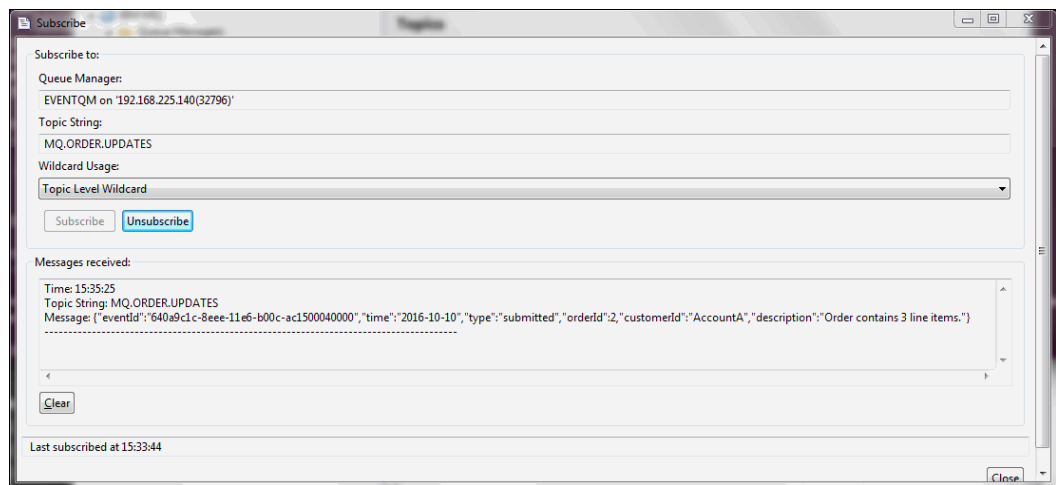


Figure 7-73 Test event

### 7.3.3 Message Hub

For Message Hub, follow the same steps as for variant B in 7.2.2, “Message Hub” on page 170.

## 7.3.4 Message Connect

The Message Connect service pulls the events from MQ and publishes them in Message Hub. To create the service, complete these steps:

1. Go to IBM Bluemix console at <https://new-console.ng.bluemix.net>, click **Catalog**, and scroll to the bottom of the window to click **Bluemix Experimental Services** as shown in Figure 7-74.

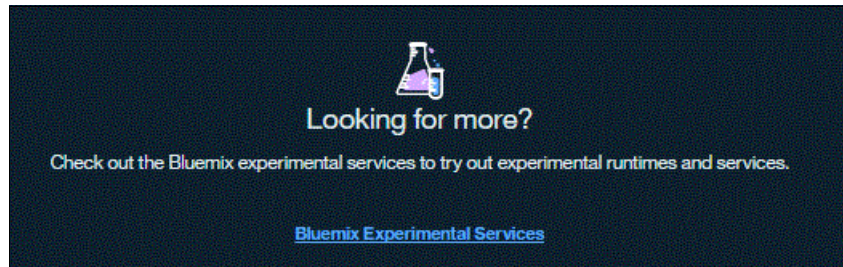


Figure 7-74 Bluemix Experimental Services

2. Enter Message Connect in the **Search** field as shown in Figure 7-75 and click the **Service** tile.

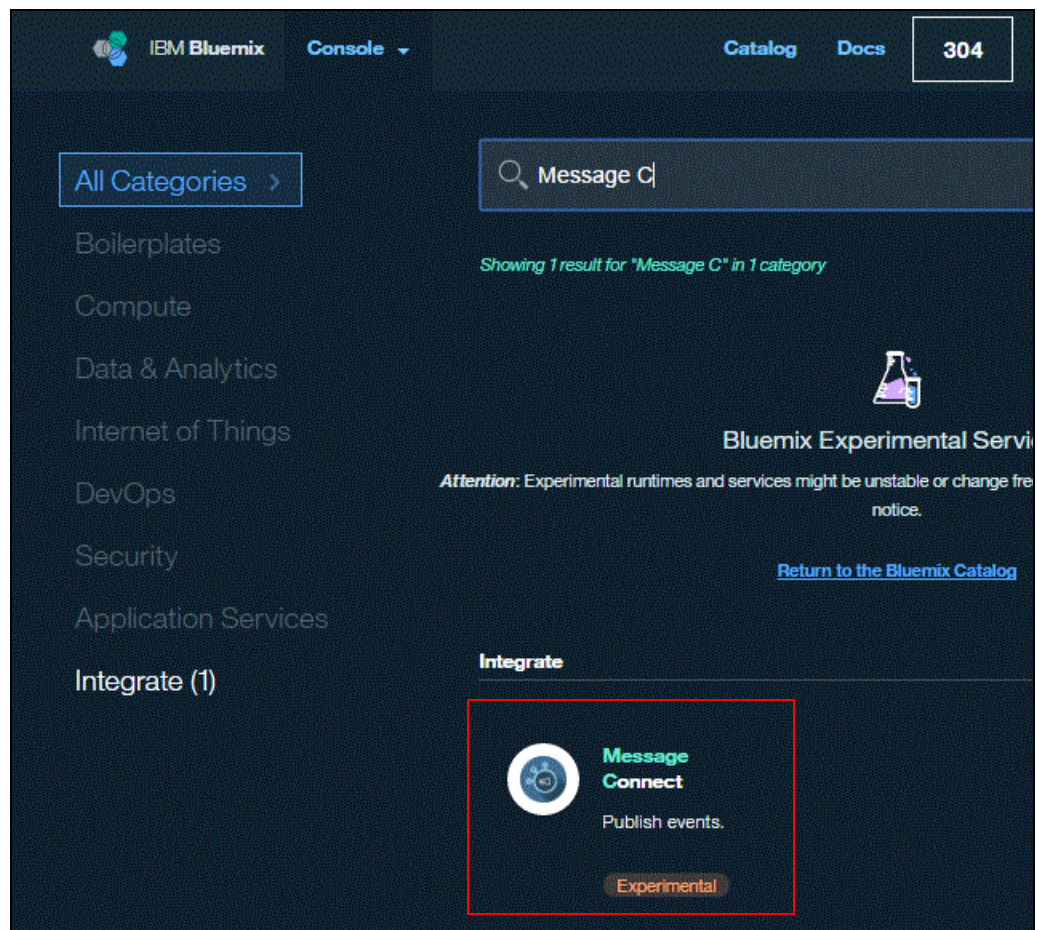


Figure 7-75 Message Connect service

3. Enter Message Connect as the name of your service instance and click **Create** as shown in Figure 7-76.

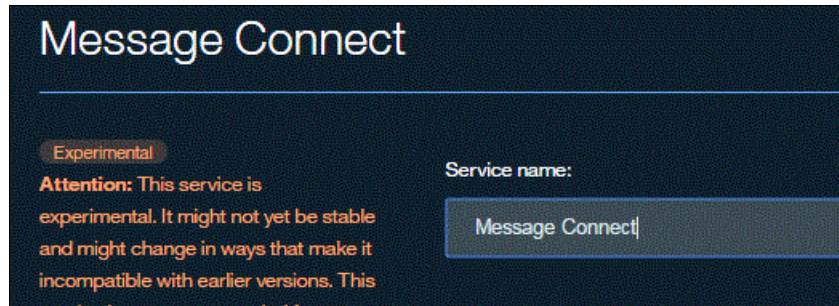


Figure 7-76 Message Connect service instance creation

4. After the Message Connect dashboard is displayed, click **CREATE YOUR FIRST STREAM** as shown in Figure 7-77.

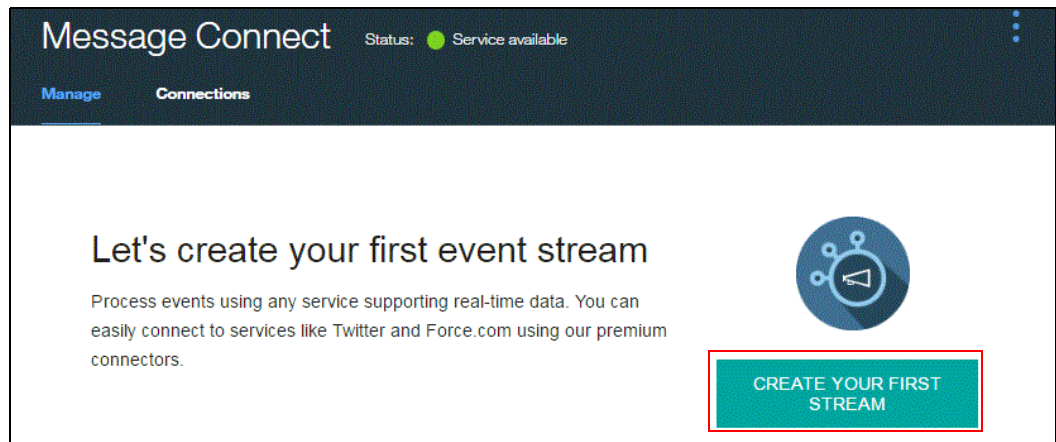


Figure 7-77 Message Connect dashboard

5. Enter mqstream as the **Stream name** as shown in Figure 7-78. This will also be the name of the topic in Message Hub.

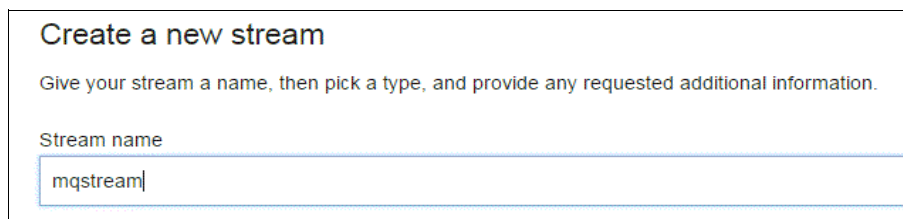
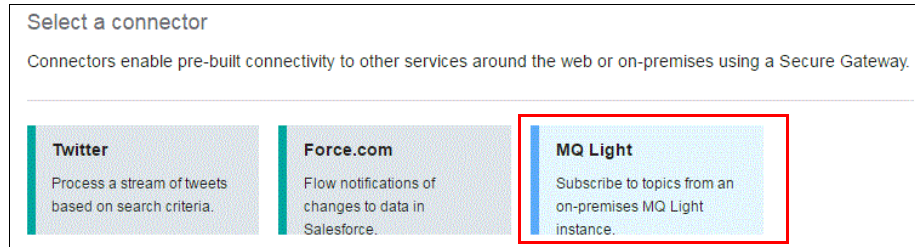


Figure 7-78 Stream name

6. Select **MQ Light** as a connector, which is the IBM implementation of the AMQP 1.0 (Figure 7-79).



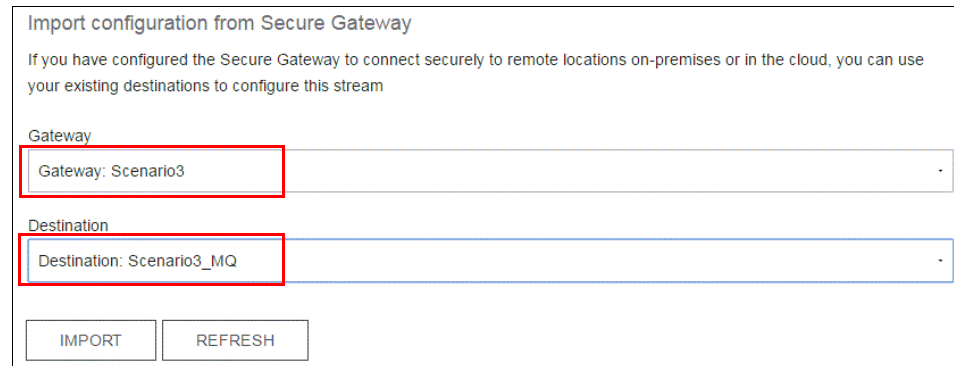
Select a connector

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

<b>Twitter</b> Process a stream of tweets based on search criteria.	<b>Force.com</b> Flow notifications of changes to data in Salesforce.	<b>MQ Light</b> Subscribe to topics from an on-premises MQ Light instance.
--	--	---

Figure 7-79 Select MQ Light connector

7. Select the **Gateway** and **Destination** that you created in 7.3.1, “Secure Gateway” on page 191, then click **IMPORT** to import configuration from the Secure Gateway as shown in Figure 7-80.



Import configuration from Secure Gateway

If you have configured the Secure Gateway to connect securely to remote locations on-premises or in the cloud, you can use your existing destinations to configure this stream

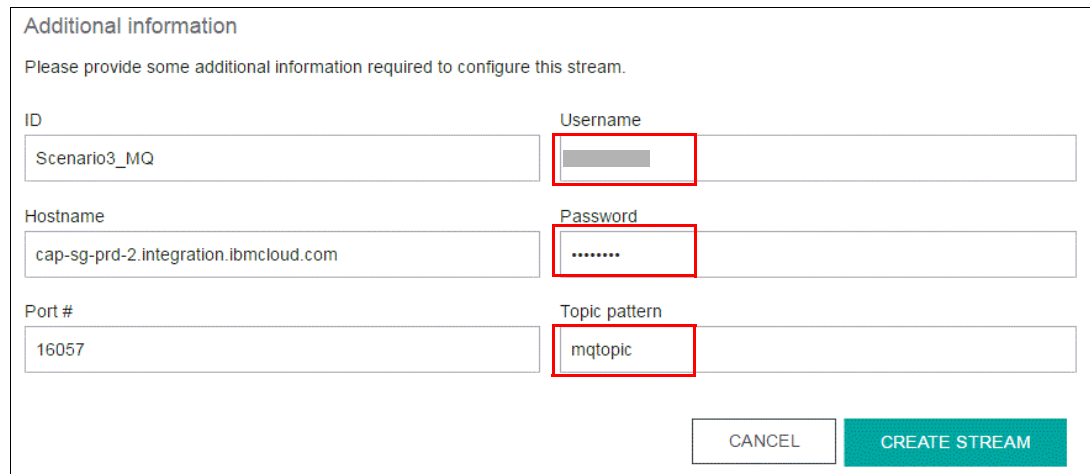
Gateway  
Gateway: Scenario3

Destination  
Destination: Scenario3\_MQ

IMPORT REFRESH

Figure 7-80 Secure Gateway configuration import

8. Now you can see that the **ID**, **Hostname**, and **Port** fields have been automatically filled. Enter `mqm` as the **Username**, password as the **Password** and `mqtopic` as the **Topic pattern** as shown in Figure 7-81. Then, click **CREATE STREAM**.



Additional information

Please provide some additional information required to configure this stream.

ID Scenario3_MQ	Username mqm
Hostname cap-sg-prd-2.integration.ibmcloud.com	Password *****
Port # 16057	Topic pattern mqtopic

CANCEL CREATE STREAM

Figure 7-81 Additional information

9. When the stream is displayed, it might be in the Stopped status. Click **View streams list** as shown in Figure 7-82.

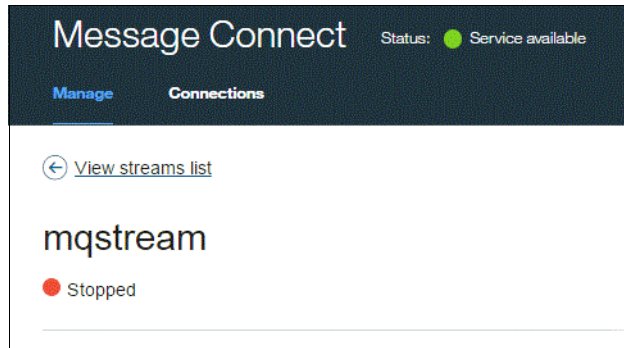


Figure 7-82 Created stream

10. Wait for the stream to change to the Running status as shown in Figure 7-83.

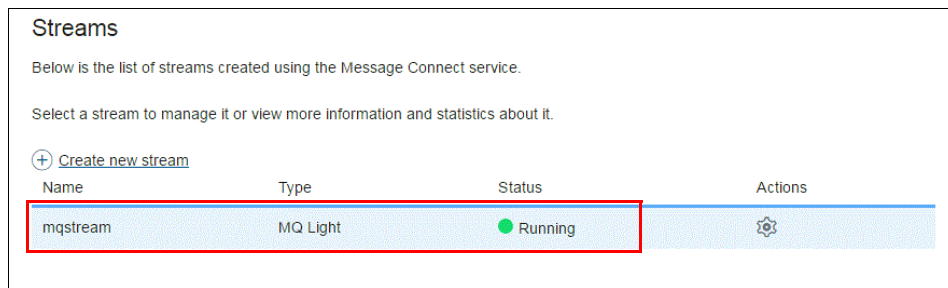


Figure 7-83 Streams list

11. Go to the Message Hub service dashboard and make sure that the mqstream topic you just created as a stream is displayed (Figure 7-84).

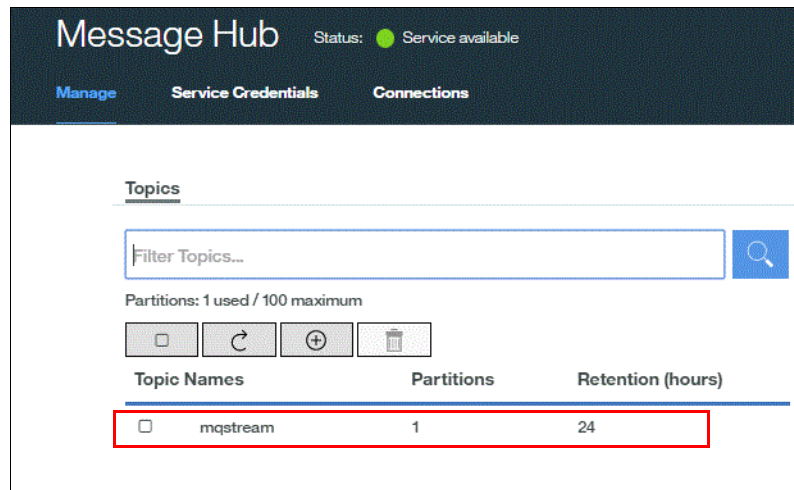


Figure 7-84 Topic on Message Hub

12. To check that MQ is properly configured, connect to your IBM MQ Docker container by issuing the following command:

```
docker exec -it scenario3_mq_1 /bin/bash
```

13. Display the status of IBM MQ by entering `display chstatus(*) chltype(AMQP) clientid(*) all`. Your console should display the status shown in Example 7-1.

*Example 7-1 Result of display channel status command*

---

```
display chstatus(*) chltype(AMQP) clientid(*) all
7 : dis chstatus(*) chltype(AMQP) clientid(*) all
AMQ8417: Display Channel Status details.
CHANNEL(SYSTEM.AMQP.CHL)                CHLTYPE(AMQP)
CLIENTID(Scenario3_MQ)                   STATUS(RUNNING)
CONNNAME(127.0.0.1)                       AMQPKA(0)
MCAUSER(mqm)                              CLNTUSER(mqm)
MSGSENT(0)                                 MSGRCVD(0)
INDOUBTIN(0)                              INDOUBTOUT(0)
PENDING(0)                                 LSTMSGDA( )
LSTMSGTI( )                               CHSTADA(2016-09-13)
CHSTATI(17.02.01)                         PROTOCOL(AMQP)
```

---

### 7.3.5 MongoDB

For MongoDB, follow the same steps as for variant B in 7.2.3, “MongoDB” on page 172.

### 7.3.6 API Connect

For API Connect, follow the same steps as for variant B in 7.2.4, “API Connect” on page 177.

### 7.3.7 Microservice

For the microservice, follow the same steps as for variant B in “Business logic development” on page 180.

### 7.3.8 Dashboard app

For the dashboard app, follow the same steps as for variant B in “Application deployment and test” on page 184.

## 7.4 Technical implementation of variant A

This section covers how to implement variant A by going through five steps in the order displayed in Figure 7-85.

**Note:** Some steps of this variant are identical or similar to steps in variant B. For those instructions, a reference points you to the correct step in 7.2, “Technical implementation of variant B” on page 157.

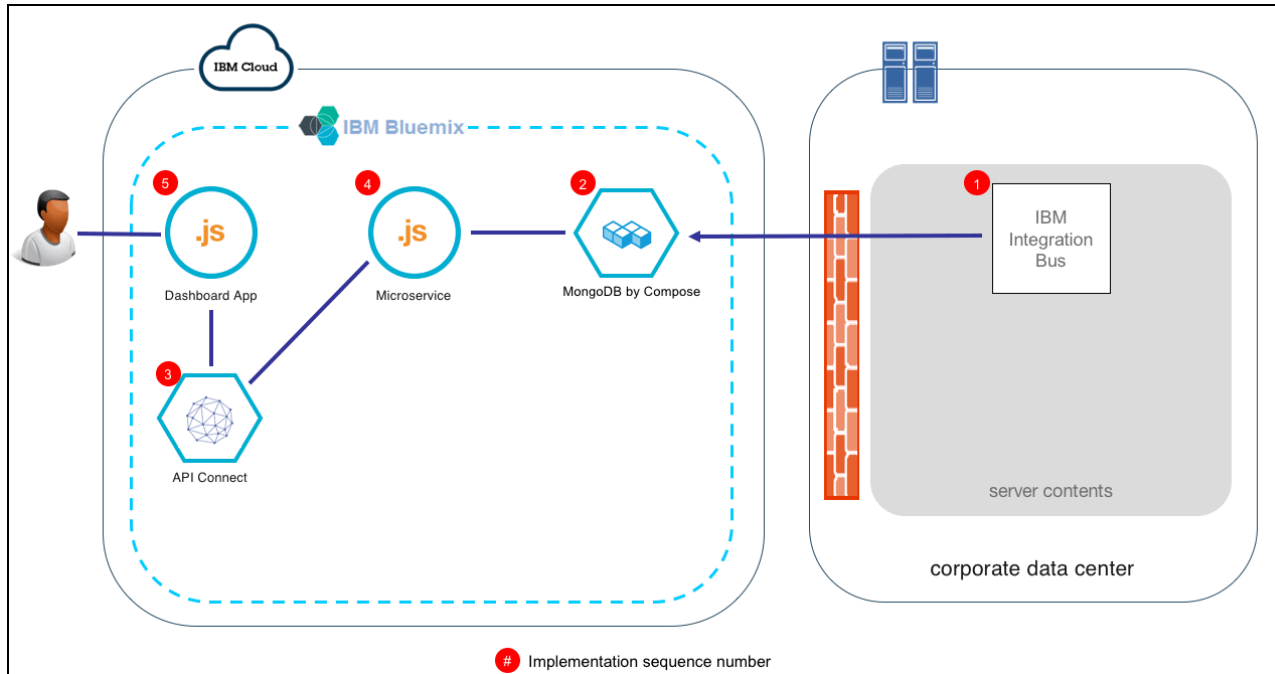


Figure 7-85 Scenario 3 - Technical diagram - Variant A

### 7.4.1 IBM Integration Bus: Exposing enterprise order events

The basis of the configuration for the variant is the implementation described in 7.2.1, “IBM Integration Bus: Exposing enterprise order events” on page 157. The instructions assume that you have completed those steps.

The main difference in this variant is publishing the order update event directly into the MongoDB service. The updated flow should look like similar to Figure 7-86.

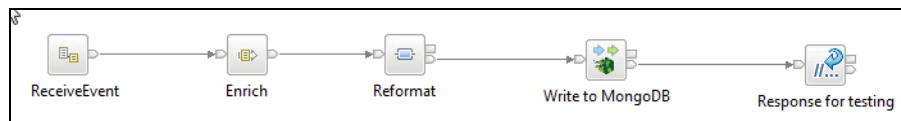


Figure 7-86 Flow overview

Instead of the Kafka node that is used in the original implementation, this variant uses a LoopBack Request node. Using the LoopBack framework, the node is configured to use a MongoDB connector.

## Prepare

The main preparation for the steps in this section is preparing the MongoDB database and configuring user credentials to access the service. Because the main variant of scenario 3 has the MongoDB setup in it already, the instructions here assume that the credentials are known. No further preparation steps are necessary.

## Build

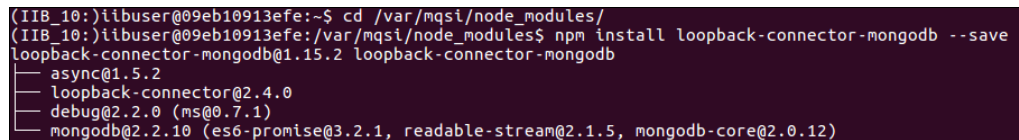
Complete the following steps:

1. Before changing the message flow itself, install the MongoDB connector that is then used by the LoopBack Request node. Open a shell inside the container by issuing the following command on your docker host:

```
docker exec -it scenario3_iib-op_1 /bin/bash
```

2. Install the mongodb connector into IBM Integration Bus by completing these steps:
  - a. In the shell session, change to the connector directory by issuing these commands:

```
cd /var/mqsi/node_modules
```
  - b. Run the command **npm install loopback-connector-mongodb --save** to install the mongodb connector. The output should look similar to Figure 7-87.



```
(IIB_10:)iibuser@09eb10913efe:~$ cd /var/mqsi/node_modules/
(IIB_10:)iibuser@09eb10913efe:/var/mqsi/node_modules$ npm install loopback-connector-mongodb --save
loopback-connector-mongodb@1.15.2 loopback-connector-mongodb
├── async@1.5.2
├── loopback-connector@2.4.0
├── debug@2.2.0 (ms@0.7.1)
└── mongodb@2.2.10 (es6-promise@3.2.1, readable-stream@2.1.5, mongodb-core@2.0.12)
```

Figure 7-87 Installing the mongodb connector

3. The next step is to configure the connection parameters for the Mongo database by completing the following steps:
  - a. Change to the `/var/mqsi/connectors` directory by issuing **cd /var/mqsi/connectors**.
  - b. Create the loopback subdirectory by issuing **mkdir loopback** and changing the directory with **cd loopback**.
  - c. The connection parameters and the definition for the mongodb data source are kept in a file called `datasources.json`. Copy the sample file into the newly created directory by issuing this command:

```
cp /home/iibuser/datasources.json
```
  - d. Use your preferred command line editor (for example, vi or emacs) to replace the `url` connection string in the `datasources.json` file with the one that you used in 7.2.4, “API Connect” on page 177.
4. Return to the IBM Integration Bus Toolkit and open the `Publish_OrderUpdate_Events` flow.
5. Remove the Kafka node from the flow and replace it with a LoopbackRequest node. Wire the node as before.



6. Configure the properties shown in Table 7-6 in the LoopbackRequest node.

Table 7-6 Properties table

Property tab	Property name	Property value	Comment
Description	Node name	Write to MongoDB	
Basic	Data source name	mongoDB	As configured previously in the datasources.json file.
Basic	Loopback object	OrderEvent	
Basic	Operation	Create	Select from the menu.

7. Deploy the flow.

## Test

Test the flow as before using the testme.sh script. The output should look like Figure 7-88.

```
(IIB_10:)libuser@09eb10913efe:~$ ./testme.sh
Test command used:
curl -X POST -d @test_event1.json -H Content-Type: application/json http://localhost:7800/orderupdates
{"id":"58009805dd0bcec39f41601c","eventId":"b009eb34-91e8-11e6-ad3e-ac1500040000","time":"2016-10-14Z",
"type":"submitted","orderId":2,"customerId":"AccountA","description":"Order contains 3 line items."}
```

Figure 7-88 Output of testme.sh script

## 7.4.2 MongoDB

For MongoDB, follow the same steps as for variant B in 7.2.3, “MongoDB” on page 172.

## 7.4.3 API Connect

For API Connect, follow the same steps as for variant B in 7.2.4, “API Connect” on page 177.

## 7.4.4 Microservice

For the microservice, follow the same steps as for variant B in “Business logic development” on page 180.

## 7.4.5 Dashboard app

For the dashboard app, follow the same steps as for variant B in “Application deployment and test” on page 184.



# 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 publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- ▶ *Hybrid Cloud Data and API Integration: Integrate Your Enterprise and Cloud with Bluemix Integration Services*, SG24-8277
- ▶ *Hybrid Cloud Event Integration: Integrate Your Enterprise and Cloud with Bluemix Integration Services*, SG24-8281
- ▶ *Getting Started with IBM API Connect: Concepts and Architecture Guide*, REDP-5349
- ▶ *Getting Started with IBM API Connect: Scenarios Guide*, REDP-5350

You can search for, view, download or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

[ibm.com/redbooks](http://ibm.com/redbooks)

## Online resources

These websites are also relevant as further information sources:

- ▶ IBM hybrid integration reference architecture white paper  
<http://ibm.biz/HybridIntRefArch>
- ▶ IBM hybrid integration reference architecture video  
<http://ibm.biz/HybridIntRefArchYouTube>
- ▶ “Microservices, SOA, and APIs: Friends or enemies?” article  
<https://ibm.biz/MicroservicesVsSoa>
- ▶ IBM hybrid integration reference architecture video  
<http://ibm.biz/HybridIntRefArchYouTube>
- ▶ IBM Watson Internet of Things portal  
<https://internetofthings.ibmcloud.com>

## Help from IBM

IBM Support and downloads

[ibm.com/support](https://ibm.com/support)

IBM Global Services

[ibm.com/services](https://ibm.com/services)

**Redbooks**

**A Practical Guide for IBM Hybrid Integration Platform**

(0.2"spine)  
0.17"->0.473"  
90->249 pages







SG24-8351-00

ISBN 0738442267

Printed in U.S.A.

Get connected

