

IBM MQ as a Service

A Practical Approach

Lohitashwa Thyagaraj

Cezar Eduardo Aranha

Kiran Darbha

Dirk Marski

Rob Nicholson

Jamie Squibb

David Ware



Infrastructure Solutions



International Technical Support Organization

IBM MQ as a Service: A Practical Approach

February 2016

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

First Edition (February 2016)

This edition applies to IBM MQ V7.0.0 and later (product number 5724-H72).

© 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

Notices	vii
Trademarks	viii
IBM Redbooks promotions	ix
Preface	xi
Authors	xii
Now you can become a published author, too!	xiii
Comments welcome	xiii
Stay connected to IBM Redbooks	xiv
Chapter 1. Introduction to IBM MQ and the as a service architecture	1
1.1 Overview of IBM MQ	2
1.2 What does “as a service” mean	3
1.3 What does IBM MQ as a service mean	4
1.4 Advantages of using IBM MQ as a service	5
1.5 How to use IBM MQ as a service	5
Chapter 2. Planning for IBM MQ as a service	7
2.1 Why create IBM MQ as a service	8
2.2 Discovering IBM MQ resources usage intent	8
2.2.1 Discovering developer intent	9
2.2.2 Additional information to provide	9
2.2.3 Discovering messages and ordering requirements	12
2.2.4 Publish/subscribe subscription requirements	13
2.2.5 Establishing an application scaling model	15
2.2.6 Establishing availability requirements	16
2.3 Self-service interfaces	17
2.3.1 IBM MQ administration interfaces	17
2.3.2 IBM MQ administration tools	19
2.3.3 Self-service considerations in a multi-queue manager environment	20
Chapter 3. Creating a self-service IBM MQ as a service menu for a portal	21
3.1 Artifacts to build self-service interfaces for IBM MQ	22
3.1.1 IBM MQ control commands	22
3.1.2 IBM MQ Script commands	22
3.1.3 Programmable Command Formats	22
3.1.4 Automation frameworks	23
3.2 Creating an IBM MQ as a service catalog	23
3.2.1 Creating a service for single-destination instance	23
3.2.2 Making the destination highly available	35
3.2.3 Single-destination instance for continuous availability	51
3.2.4 Multiple-destination instances	65
3.2.5 Multiple-destination instances: No stranded or marooned messages	74
3.2.6 Creating a topic destination	82
3.2.7 Setting up authorizations on IBM MQ objects	90
3.2.8 Updating the properties of IBM MQ resources	98
Chapter 4. Infrastructure view of IBM MQ topologies	107

4.1 IBM MQ hubs: An overview	108
4.2 Single availability zone: Active/Passive	108
4.3 Single availability zone: Active/Active	109
4.4 Multiple availability domains: Active/Passive	111
Chapter 5. Other factors to consider when planning IBM MQ as a service	113
5.1 Security model for IBM MQ as a service	114
5.1.1 Authentication	114
5.1.2 Authorization and access control	115
5.2 Monitoring	116
5.2.1 Resource monitoring	116
5.2.2 Monitoring performance	117
5.2.3 Monitoring application activity	118
5.2.4 Change management and auditing	118
5.2.5 Consuming events	119
5.2.6 Tracking messages	119
5.2.7 Challenges and limitations	121
5.2.8 Impact of monitoring	121
5.2.9 Configuring monitoring and consuming monitoring data	121
5.3 Multitenancy	122
5.3.1 Resource allocation	123
5.3.2 Security and administration	124
5.3.3 Maintenance	124
5.3.4 Troubleshooting	125
Chapter 6. Infrastructure choices for IBM MQ as a service	127
6.1 Virtualization	128
6.1.1 IBM MQ virtualization support	129
6.1.2 Provisioning virtual machines	130
6.2 Containers	130
6.2.1 Docker	130
6.3 IBM MQ patterns	131
6.3.1 IBM MQ Virtual System Pattern Type	131
6.4 IBM MQ Appliance	133
6.4.1 The role of IBM MQ Appliance in the IBM MQ as a service model	134
6.5 IBM Cloud Orchestrator and IBM MQ integration	134
6.5.1 Delivering pattern-based clouds	135
6.5.2 Working with IBM Cloud Orchestrator middleware patterns	136
Chapter 7. Connecting IBM MQ messaging applications to an IBM MQ as a service infrastructure	137
7.1 Using IBM MQ as a service	138
7.2 Setting connection properties	138
7.2.1 Application code	138
7.2.2 The MQSERVER environment variable	139
7.2.3 The connection factory	139
7.2.4 Client Channel Definition Table	139
7.3 Queue manager connectivity patterns	140
7.3.1 Mechanisms to achieve workload balancing and availability when connecting	141
7.3.2 Managing high availability at the application layer	145
7.3.3 Considerations when configuring workload balancing and availability	146
7.3.4 Comparison of mechanisms	148
Appendix A. IBM UrbanCode Deploy initial setup	149

Getting started	150
IBM UrbanCode Deploy concepts	150
Installing the IBM UrbanCode Deploy server components.	151
Starting the IBM UrbanCode Deploy server	156
Stopping the IBM UrbanCode Deploy server	157
Installing the IBM UrbanCode Deploy agent	157
Starting the IBM UrbanCode Deploy agent.	160
Stopping the IBM UrbanCode Deploy Agent.	161
Loading the IBM MQ plug-in for IBM UrbanCode Deploy	161
Application deployment with IBM UrbanCode Deploy	163
Creating a team	163
Adding role members to the team.	164
Creating IBM UrbanCode Deploy objects for application deployment	164
Creating an application	179
Creating the environment for the application	180
Adding the resource to the application	181
Related publications	185
IBM Redbooks	185
Online resources	185
Help from IBM	185

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 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®	First Failure Support Technology™	Redbooks®
CICS®	Global Technology Services®	Redpaper™
DataPower®	IBM®	Redbooks (logo)  ®
DB2®	IBM UrbanCode™	WebSphere®
developerWorks®	PowerHA®	z/OS®
FFST™	PureApplication®	

The following terms are trademarks of other companies:

SoftLayer, and SoftLayer device are trademarks or registered trademarks of SoftLayer, 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.

Microsoft, 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.

UNIX is a registered trademark of The Open Group in the United States and other countries.

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

Find and read thousands of IBM Redbooks publications

- ▶ Search, bookmark, save and organize favorites
- ▶ Get personalized notifications of new content
- ▶ Link to the latest Redbooks blogs and videos

Get the latest version of the Redbooks Mobile App



iOS

Download
Now

Android



Promote your business in an IBM Redbooks publication

Place a Sponsorship Promotion in an IBM® Redbooks® publication, featuring your business or solution with a link to your web site.

Qualified IBM Business Partners may place a full page promotion in the most popular Redbooks publications. Imagine the power of being seen by users who download millions of Redbooks publications each year!



ibm.com/Redbooks

About Redbooks → Business Partner Programs

THIS PAGE INTENTIONALLY LEFT BLANK

Preface

This IBM® Redpaper™ publication provides information about how to build, deploy, and use IBM MQ as a service. The information in this paper includes the key factors that must be considered while planning the use of IBM MQ as a service.

Through descriptions and examples, this paper explains how to apply as a service methodologies to an IBM MQ environment, and describes techniques and preferred practices for integrating IBM MQ into a self-service portal.

This paper explains how to create and use an IBM MQ as a service self-service menu for a portal. It includes examples that show how to use an IBM MQ as a service catalog.

This paper describes options and techniques for deploying IBM MQ as a service that is tailored to the specific enterprise messaging needs of an organization. Although these techniques can be employed in a cloud environment, they are equally applicable in an on-premises enterprise data center.

This paper includes information about the various infrastructure options that can be selected when implementing IBM MQ as a service. The information in this paper helps infrastructure administrators to define services so that you can provision IBM MQ resources quickly.

The target audiences of this paper are developers, infrastructure administrators, and line-of-business (LOB) professionals who want to provision IBM MQ resources to be accessed as services in small, medium, large, and complex implementations.

The paper consists of the following chapters and appendix:

- ▶ Chapter 1, “Introduction to IBM MQ and the as a service architecture” on page 1
- ▶ Chapter 2, “Planning for IBM MQ as a service” on page 7
- ▶ Chapter 3, “Creating a self-service IBM MQ as a service menu for a portal” on page 21
- ▶ Chapter 4, “Infrastructure view of IBM MQ topologies” on page 107
- ▶ Chapter 5, “Other factors to consider when planning IBM MQ as a service” on page 113
- ▶ Chapter 6, “Infrastructure choices for IBM MQ as a service” on page 127
- ▶ Chapter 7, “Connecting IBM MQ messaging applications to an IBM MQ as a service infrastructure” on page 137
- ▶ Appendix A, “IBM UrbanCode Deploy initial setup” on page 149

Authors

This paper was produced by a team of specialists from around the world working with the International Technical Support Organization.

Lohitashwa Thyagaraj is a Senior Technical Staff Member (STSM) and Master Inventor for IBM Messaging working at the IBM Systems Middleware organization, IBM India Software Lab. Lohit is responsible for defining the strategy and development of the IBM WebSphere® Application Server and IBM WebSphere Application Server Liberty messaging portfolio. Lohit holds a Bachelor of Engineering degree in Computer Science from Bangalore University.

Cezar Eduardo Aranha is an architect for Essential Hosting Services in the IBM Global Accounts (IGA) organization in IBM Brazil. Cezar's areas of expertise include IBM MQ, IBM Integration Bus, and IBM WebSphere Transformation Extender. Cezar is part of the Application Integration & Middleware team that supports several IBM internal customers. Cezar has 25 years of experience in the IT industry. Before his IBM career, Cezar worked for a large international bank in Brazil as project manager and technical support. He holds a Master of Business Administration degree from the Universidad de Positivo, Brazil.

Kiran Darbha is an Advisory Software Engineer in the IBM Systems Middleware organization in IBM India. He has 12 years of experience in the IT industry, and specializes in messaging services, global transactions, and Microsoft .NET. Kiran is the technical team leader for the IBM MQ Low Latency Messaging and File Transfer components. Kiran holds a Bachelor of Engineering degree in Computer Science from Visvesvaraya Technological University, India.

Dirk Marski is a manager in the IBM Integration Services organization, and is responsible for the support of the IBM API Management product. Dirk previously worked as an IBM MQ for z/OS® support engineer, and in that role he provided advanced technical support to IBM MQ clients to help them resolve the most complex technical issues with the technology. Dirk holds a MEng in General Engineering from the University of Centrale De Lyon, France, and a Master in Software Engineering (MSc) for High Performance Computing from Cranfield University, UK.

Rob Nicholson is a Distinguished Engineer and Master Inventor for IBM Messaging in the IBM Messaging development organization at Hursley, UK. Rob is responsible for the development of the IBM messaging portfolio. He holds a MEng. Electronic Systems Engineering degree from Aston University, UK.

Jamie Squibb is a Software Engineer at the IBM Hursley Laboratory in the UK. He is a team leader in the IBM MQ development organization, where he has worked since joining IBM as a graduate in 2001. Jamie has a first class degree with honors in Computer Science from the University of Warwick, UK.

David Ware is the lead architect for IBM MQ distributed platforms. David has many years of experience designing and developing messaging middleware software, spending much of his time working with and advising IBM clients. He has worked on IBM MQ and IBM WebSphere Application Server. He has a wide knowledge of messaging technology with particular expertise in distributed IBM MQ, the publish/subscribe model, and IBM MQ clustering topologies.

The project that produced this publication was managed by **Marcela Adan**, IBM Redbooks® Project Leader - International Technical Support Organization.

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

Peter Broadhurst, Jonathan Rumsey, Pete Siddall
IBM Systems, Middleware, IBM UK

Savitha Joshi
IBM Systems, Middleware, IBM India

LindaMay Patterson
International Technical Support Organization

Edson Gomes Pereira and Sergio Straessli Pinto
IBM MQ, Global Technology Services®, IBM Brazil

Now you can become a published author, too!

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

- Send your comments in an email to:

redbooks@us.ibm.com

- Mail your comments to:

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

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:
<http://www.facebook.com/IBMRedbooks>
- ▶ Follow us on Twitter:
<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>



Introduction to IBM MQ and the as a service architecture

This chapter provides an overview of IBM MQ, explains the *as a service* architecture, and describes how IBM MQ fits into the as a service paradigm.

This chapter contains the following sections:

- ▶ 1.1, “Overview of IBM MQ” on page 2
- ▶ 1.2, “What does “as a service” mean” on page 3
- ▶ 1.3, “What does IBM MQ as a service mean” on page 4
- ▶ 1.4, “Advantages of using IBM MQ as a service” on page 5
- ▶ 1.5, “How to use IBM MQ as a service” on page 5

1.1 Overview of IBM MQ

IBM MQ is a robust messaging middleware that simplifies and accelerates the integration of diverse applications and business data that might be distributed over multiple platforms and geographies. IBM MQ facilitates the assured, secure, and reliable exchange of information between applications, systems, services, and files. This exchange of information is achieved through the sending and receiving of message data through queues and topics, which simplifies the creation and maintenance of business applications. IBM MQ delivers a universal messaging solution that encompasses a broad set of offerings to meet the needs of an enterprise, in addition to providing connectivity for mobile devices and the Internet of Things.

IBM MQ provides the following capabilities:

- ▶ Rapid, flexible, and seamless connectivity of information through a single, robust, and trusted messaging backbone for dynamic heterogeneous environments.
- ▶ Secure, reliable message delivery that preserves message integrity and minimizes the risk of information loss.
- ▶ High performance and scalable message transfer to meet the demands of today's enterprise and beyond.
- ▶ Simplified management that provides control and usability.
- ▶ Lower cost of ownership by reducing the cost of integration and accelerating time to deployment.

One of the primary use cases for IBM MQ is to provide a foundation for creating service-oriented architectures (SOAs). In an SOA, some services might be shared or reused within only a single domain, and other services might be shared or reused throughout the enterprise. IBM MQ provides the means to establish communication between lines of business (LOBs) and otherwise separate business domains. IBM MQ supports a number of configuration options that can be used to scale and to achieve high availability (HA) while ensuring resistance to failure. These configuration options usually use several product features, such as IBM MQ clusters, multi-instance queue managers, and queue-sharing groups. Some configurations also depend on other hardware or software.

IBM MQ supports many APIs and tools for managing, configuring, and provisioning messaging resources:

- ▶ Control commands: Administrative command utilities
- ▶ IBM MQ Explorer: An Eclipse graphical user interface (GUI)
- ▶ Programmable Command Format (PCF) messages: A programmatic interface
- ▶ IBM MQ Script Commands (MQSC): Human-readable command interface
- ▶ Web user interface (UI): Browser-based interface for monitoring the IBM MQ Appliance

1.2 What does “as a service” mean

The advent of cloud computing introduced new terminology, such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). These terms refer to delivery models for empowering users to obtain access to infrastructure, development platforms, and software by using a *self-service interface*. The *as a service* delivery model is not limited to cloud computing; it can be equally applied elsewhere to empower users to achieve their goals without impediments and delays. Businesses that can adopt this methodology throughout their enterprise are more agile and can readily respond to changing customer and industry demands.

Underpinning *as a service* delivery is the ability for users to service themselves by using a self-service interface. For example, a customer can use a self-service interface to withdraw money by using an ATM instead of going to a bank cashier for service. The ATM provides a simple interface for the customer, eliminates the dependency on a cashier, and streamlines the process of satisfying the customer's goal. The use of ATMs also frees the cashiers to provide better service to other types of customer requests. Another example of a self-service interface is a retail store kiosk that customers can use to select and purchase items. In both of these examples, there are many processes that run to complete the requests, but whenever possible, these processes are automated and hidden from the user.

When users are empowered to use a self-service interface, it becomes harder to predict the demand for that service at any time. Most services have variations in their demand profile. Although it is necessary to provide sufficient capacity to support peak usage levels, it can be expensive or even impractical to allocate a high level of resources permanently.

To address this issue, you must become familiar with a concept known as *elastic scaling*. Elastic scaling refers to the means by which capacity is added or removed (either automatically or by human intervention) in response to the actual or predicted workload. Removing capacity when it is no longer required means that you can target underlying resources elsewhere for maximum efficiency.

Consider the online store for a large retailer. Throughout the day, week, and year, a varying numbers of customers access the online store to view the products and hopefully make purchases, track orders, and provide favorable reviews. The server capacity must elastically scale to respond to customer requests. Black Friday is one of the busiest shopping days of the year in the United States. As a result, the online store is likely to serve a much larger number of customer visits. It might be necessary to introduce additional server capacity to satisfy demand. It is also likely that far less predictable peaks in demand might occur throughout the year, for example, in response to social media trends.

Another attribute that is associated with service offerings is provisioning for *usage-based pricing*. A service is consumed to a varying extent by different users. Charging all users equally is often perceived as unfair and is not necessarily conducive to the successful adoption of a service offering. Charging consumers to use a service is also not always appropriate, but for the cases where it is appropriate, then it is preferable to charge based on consumption levels.

The self-service interface provides an appropriate point of control that can be used to track usage. Consider a company or an infrastructure team within an enterprise that offers a network backup service. Users have varying quantities of data to store and might have varying requirements, such as the speed of access. The company or infrastructure team can charge users based on the demands that are placed on the resources to accommodate their backup needs.

1.3 What does IBM MQ as a service mean

The term *IBM MQ as a service* applies the principles of an as a service delivery model to an IBM MQ deployment. Traditionally, IBM MQ is administered exclusively within an enterprise by a central messaging middleware team. LOB professionals submit change requests to this central team of experienced IBM MQ administrators who act upon these requests. As the number and complexity of requests increases, it becomes less practical for a central team to respond within a timescale that is acceptable to the business units.

Enterprises can use these capabilities to implement self-service portals that allow LOBs to request changes to the messaging infrastructure. These changes might range from the creation or deletion of a queue to the provision of a highly available and scalable topology for a new business application. The IBM MQ administrators, when providing self-service interfaces, can also use product features to ensure that stability is maintained and security, audit, and availability requirements remain satisfied. There are many challenges that are associated with accommodating both the self-service needs of business units and the availability, scalability, security, and stability requirements for which the middleware team are responsible. These challenges range from accommodating the rich configuration and security options in IBM MQ to the management of many networked systems in varying topologies.

The following use cases illustrate why it is important to make productive use of IBM MQ as a service:

► Use case 1

A large bank deployed IBM MQ over 20 years, during which time three smaller banks were acquired and absorbed into the enterprise, each with their own IBM MQ administrative practices, installations, and business applications. IBM MQ is used for every critical business transaction in the bank. Digital and mobile banking is driving the creation of new customer facing applications at a faster rate than at any other time in the history of the bank.

The bank must elastically scale and rapidly provision development and production IBM MQ infrastructures for new or updated business applications. The bank requires each LOB to use a self-service interface to provision the necessary infrastructure changes themselves within minutes or hours, not days or weeks.

The LOB professionals do not have the IBM MQ administrative skills that are necessary to accomplish this requirement. The bank must deliver continuous availability and disaster recovery (DR) across their critical tier-1 messaging infrastructure in a consistent and cost-effective way. The bank has a highly specialized central team of IBM MQ administrators that must focus on high-value architectural work instead of low-value provisioning activities.

► Use case 2

A large retail outlet has over 2000 stores that are located throughout the United States. Each store must connect to its headquarters to share information about stock levels, trades, and other business transactions. The company configured IBM MQ as the standard mechanism to interact between stores and headquarters.

The IBM MQ infrastructure at the headquarters must be resilient to failure, have a highly available configuration, and elastically scale to support the business. It also must rapidly respond to changes in the workload that is driven from the stores. When a new store is opened, it is necessary to establish connectivity with its headquarters within a working day (5 - 8 hours) with limited support from IBM MQ administrators. A self-service interface is required that store-based employees can use to achieve this goal.

1.4 Advantages of using IBM MQ as a service

It is not essential for an enterprise to offer IBM MQ as a service, but doing so can provide the following benefits:

- ▶ Increased agility

Empower LOB professionals to provision or update directly the messaging resources they require, which reduces time to value by eliminating the dependency on manual activities that traditionally are performed by a central infrastructure team. You can use automation to scale resources dynamically in response to changing business demands.

- ▶ Improved usability

LOB professionals can use self-service interfaces to focus on their core requirements, such as what destinations they need and what quality of service is necessary. The complexity of satisfying these requirements can be hidden from users.

- ▶ Improved efficiency

Automation that underpins a self-service interface allows systems and messaging resources to be dynamically provisioned for test and development purposes. The use of automation can also simplify the activity of promoting changes from development to test, quality assurance, and production environments.

- ▶ Improved consistency

Exposing a self-service interface to the LOB units provides a single entry point for changes to the messaging configuration. This interface can use common routines and workflows to deploy the necessary changes, which enforces standards and consistency.

1.5 How to use IBM MQ as a service

Here are the techniques that are necessary to use IBM MQ as a service:

- ▶ Automating the creation and management of IBM MQ resources by using scripts or programmatic interfaces. IBM MQ provides a range of control commands and script commands (MQSC) to support this technique. If you want to access resources programmatically, IBM MQ supports administration through Programmable Command Format (PCF) messages after access to a queue manager is established. IBM MQ administrators and service providers can create workflows that drive these commands to underpin a self-service interface.
- ▶ Automation frameworks, such as IBM UrbanCode™ Deploy, Chef, and Puppet, can be used to orchestrate administrative operations for IBM MQ with those for other products to create or manage entire application or server environments.
- ▶ Virtual machines (VMs), Docker containers, IBM PureApplication® System, and the IBM MQ Appliance can be used alongside automation frameworks to create a flexible and scalable infrastructure that is used to provide an IBM MQ as a service offering.



Planning for IBM MQ as a service

This chapter describes the key factors to consider when building and using IBM MQ as a service.

This chapter contains the following sections:

- ▶ 2.1, “Why create IBM MQ as a service” on page 8
- ▶ 2.2, “Discovering IBM MQ resources usage intent” on page 8
- ▶ 2.3, “Self-service interfaces” on page 17

2.1 Why create IBM MQ as a service

There are usually three categories of users who are interested with the messaging infrastructure: lines of business (LOBs), developers and testers, and the infrastructure team.

The LOB staff likely are not concerned with the details of the implementation. They need to build new solutions for the business quickly to leverage business opportunities or counter competitive threats. The LOB is looking for technology that can be provisioned quickly. Deploying IBM MQ as a service is one way to achieve that requirement.

Developers are looking to IBM MQ to provide interfaces, either for a service they are developing or a service on which they rely. Developers might not be concerned about high availability, scalability, and reliability because these requirements are usually concerns of the infrastructure team. What developers want is a way to provision the queues and topics they need for their applications as and when they need them for development and test activities. IBM MQ as a service can help developers be more effective.

The infrastructure team represents the traditional IBM MQ teams that know the product and are familiar with configuring and administering it in the various environments and topologies. In most organizations, the infrastructure team is tasked with guarding the operational integrity of the core infrastructure. It is their processes and procedures that ensure that the core infrastructure remains secure, highly available, high performing, and resilient to disasters. Typically, this task is achieved by a change control process that is rigorous, and consists of long questionnaires filled out by developers and then processed by the infrastructure team. This interaction leads many organizations to report that provisioning IBM MQ infrastructures in the organization can take as much as three months from initial request to completion.

The business pressure on infrastructure teams as a cost center is to reduce human activity to a minimum by automating their preferred practices. IBM MQ as a service can increase the speed at which the infrastructure team can deliver new IBM MQ infrastructure to developers, while ensuring that the needs of the business are met by the solution.

2.2 Discovering IBM MQ resources usage intent

Before you can automate any IBM MQ as a service solution, it is important to find a way to capture developers' functional and nonfunctional requirements. These requirements must be mapped to a set of IBM MQ resources. After this task is accomplished, the developer is given all the information that they need to access the IBM MQ resource in a reliable way from their code.

The first time an application team does something new, they are likely to need advice from the infrastructure team. In these cases, a set of questions (provided by the infrastructure team) helps provide a common language between the infrastructure team and development team, which enables them to discuss the various options. Application teams can need the same thing repeatedly. Over time, the questions (from the infrastructure team) become a way to make a fast, standardized, and fully automated request for an IBM MQ as a service infrastructure.

It is important to establish a set of questions that the infrastructure team might ask a developer to understand the developer's needs. The answers are not always mutually exclusive. Developers must make cost versus benefit decisions in how they code or change their application to meet their non-functional requirements. You must consider the IBM MQ landscape within your own enterprise and prioritize the requirements that provide the highest value to you.

2.2.1 Discovering developer intent

A developer writing a piece of code might not care about queue managers and server infrastructure directly. Typically, they care about the *destinations* their application logic requires so that they can write the code that they need to access those destinations. In a self-service infrastructure with a fast turnaround, a developer can request the resources that they need one by one, or in a group. Either way, they should end up with the same result and with the correct resources to solve their business requirements. Each requirement might need something different from the IBM MQ as a service infrastructure, making allocation at a fine-grained level a good idea.

For example, a project might need to send request/reply COBOL copybook messages to an existing IBM CICS® mainframe application by using queues and emit JSON events over publish and subscribe. The needs of these two requirements are different and might be allocated to different queue managers in an IBM MQ as a service infrastructure.

A sample question to establish developer intent

The main question that can help the infrastructure team to understand the intent of the developers from destination usage perspective is, "What will your code use the destinations for?".

The main question breaks into these parts:

- ▶ Exposing a new endpoint to other applications/components:
 - Through a queue: You plan to use queues to receive explicit requests to perform work and return data.
 - A subscription on a topic: You plan to use publish/subscribe topics to trigger one or more actions when data is emitted.
- ▶ Sending messaging to an existing endpoint:
 - Synchronous request/reply: Your code blocks waiting for responses from the remote system.
 - Asynchronous request/reply (preferred coding pattern for request/reply): Your code is run when the response arrives at any point in the future.
 - Fire-and-Forget: Your code does not expect any replies.

2.2.2 Additional information to provide

Developers seldom develop code from first principles and documentation. Instead, most developers start from a sample that performs a similar task to what they want to achieve. They use this sample and iterate on it until they have code that works correctly and they understand the code details. To achieve resilience and performance within IBM MQ as a service, it is important for the different application teams to achieve consistency in how they write their code. The starting code snippets should come from the organization.

In this case, instead of getting a queue/topic name from IBM MQ as a service, developers get a queue/topic name along with a code sample that shows how to do the job by using IBM MQ. As infrastructure teams seldom write application code themselves, it is important to work with developers to capture preferred practices from successful IBM MQ development projects.

IBM MQ provides a large set of options. Even when they start with good samples, developers sometimes must use features that meet specific requirements of the infrastructure. This situation limits the economies of scale that can be achieved with IBM MQ as a service. There are benefits to controlling how applications use those features in a prescriptive way. To obtain this control, use a simplified code library that sits in front of the IBM MQ interface and provides a subset of its features. An application team who uses the code library can be sure that they do not write any code that prevents them from using the full capabilities of your IBM MQ as a service infrastructure.

An example code library is provided by GitHub, Inc. (US), which can be found at the following website:

<https://github.com/ibm-messaging/mq-wlm-client>

Another example code library is provided by Capitalware, Inc., which can be found at the following website:

http://www.capitalware.com/mq_code_java.html

Discovering quality of service requirements

IBM MQ ensures delivery of persistent messages. After a message is placed in IBM MQ, the *in-flight* data will be delivered in all circumstances. However, making messages persistent alone does not ensure that business transactions complete exactly once. You also must consider the application infrastructure and the network that connects the applications to IBM MQ. If the application wants to use IBM MQ to complete a transaction exactly once, it must use one of the approaches that are described in the following sections.

Global (XA) transactions (exactly-once delivery)

In this particular case, applications update a database by using the same transaction that is used to send messages to IBM MQ. If the application fails or the network between the application and IBM MQ fails, the application can check the database to ensure that the message was sent. If a row was successfully inserted into the database, then the combination of the eXtended Architecture (XA) transaction coordinator and IBM MQ ensured delivery means that the message arrived at its destination. If the row is not inserted, then the same combination confirms that the message was not sent.

When an application uses XA transactions, the coordinator (such as a Java EE application server) requires a one-to-one relationship between the code creating the connection to IBM MQ and the queue manager it connects to (except for IBM z/OS queue-sharing groups). A Client Channel Definition Table (CCDT) or network load balancer cannot be used to connect the code to multiple queue managers.

Workload balancing can still be achieved, but must be performed by the code. An example for Java EE application is provided by GitHub, Inc., found at this website:

<https://github.com/ibm-messaging/mq-wlm-client>

Note: At the time of writing, a Request For Enhancement (RFE) remains open to request the removal of this limitation for cases where IBM WebSphere Application Server is the transaction coordinator. For more information, go to the following website:

http://www.ibm.com/developerworks/rfe/execute?use_case=viewRfe&CR_ID=53793

If asynchronous replication is being performed to a remote data center for disaster recovery (DR) purposes, then the transaction logs of your XA transaction manager should be in the same consistency group as the IBM MQ queue manager and database (or other XA resources) that are involved in your XA transaction. This approach ensures that entire XA transactions are restored to the same state after a failure.

Idempotence or duplicate-checking (at-least-once delivery)

Applications can be designed so that the logic that processes a message is tolerant of duplicate messages. This approach eliminates the need for an XA transaction because the sending application can resend the message if it (or a network connection) fails before getting confirmation from IBM MQ that the message is safely stored on an IBM MQ queue.

There are two approaches to tolerating duplicate messages:

► **Idempotence:**

The processing logic updates its state in a way that produces the same result no matter how many times it is processed. For example, you can use processing logic that updates a row in a database that is unique to the transaction. No matter how many times the logic runs, only one row is added to the database.

► **Duplicate checking:**

The processing logic performs a uniqueness check before it processes a message, such as by inserting a row with a uniqueness constraint in its database in the same database transaction as its main update. This approach allows state changes and additions/subtractions from totals to be performed exactly once.

In either case, it is necessary for a sending application to generate and store a unique transaction identifier before the message is sent. By placing that transaction identifier inside the message, the processing application has a unique identifier for the piece of information.

An IBM MQ CorrelationId (or GroupId) can be used for this purpose when it is used with a local (non-XA) transaction. After doing a send (MQPUT), the application can obtain a unique ID that is generated by IBM MQ. Then, it can update its database with a row that associates the business transaction with the unique identifier before calling a commit (MQCMIT).

Global uniqueness: Ensure that the first 12 characters of your queue manager names are unique throughout your IBM MQ infrastructure if you want to rely on IBM MQ Group/Correlation/Message IDs being globally unique.

A sample question to establish quality of service requirements

To understand your quality of service (QoS) requirements, ask the question, “What level of delivery assurance is required by your application code?”. This question consists of the following parts. Select the appropriate options and characteristics.

► **Exactly-once delivery**

- You have an XA transaction coordinator (such as a Java EE application server) that must coordinate database updates with each send/receive that is performed in IBM MQ.
- You include workload balancing logic within your application to manage connections to multiple IBM MQ queue managers.
- During maintenance or because of unexpected infrastructure issues, the technical design or service level agreement (SLA) of the application ensures that it can tolerate short interruptions in the availability of IBM MQ (typically up to 5 minutes).

- ▶ At-least-once delivery
 - Each transaction has a unique identifier that is sent in the IBM MQ message.
 - The processing application is either idempotent or performs duplicate checking.
 - After a message is placed into IBM MQ, you expect it to be delivered in all circumstances, including recovering messages through high availability failover after infrastructure failures.
- ▶ At-most-once delivery
 - Your application logic does not rely on IBM MQ to recover messages after unexpected failures.
 - You resend messages based on a timeout or can cope with the occasional loss of messages.

2.2.3 Discovering messages and ordering requirements

Messages often have affinities for each other. Consider a simple example of creating a customer and then updating their address. If the update to the customer's address arrives before the creation of the customer, then what does the processing do? Consider the following options:

- ▶ Store for later processing.

If an application can clearly identify when a message arrived out of order, then it can store it (in IBM MQ or a database) for later processing. This approach requires inserting sequencing information into the message when it is sent.

For this example, a per customer update counter can be incremented in a database on the sending side before sending the update message. The processing application can maintain an equivalent counter and place update messages on a holding queue until the create message arrives and the gap is filled.

If an application has not implemented exactly-once delivery, this approach can be combined with a timeout to prevent messages from being held indefinitely when a message is lost by the application/infrastructure. The action of the timeout might be to attempt processing of the message out of sequence or to start manual compensation.

- ▶ Process messages in any order.

It is possible to code the logic of a receiving application to process messages in any order, which is similar to the idempotence for at-least-once delivery. For the customer address example, the address can be stored separately from the other customer information and linked to by an ID (which is an orphan until the customer itself is created as its parent).

- ▶ Require IBM MQ to deliver the messages in the order sent.

Applications that rely on IBM MQ to deliver messages in the order that they are sent cannot easily use parallel routes through the IBM MQ infrastructure. There must be an exactly-once route from the sending application to the processing application and locking within the application (such as database locks) to prevent out-of-order processing by parallel threads within the application. This approach means that there is only one queue manager and one queue within an ordered stream. If an application wants to use multiple queue instances (for example, multiple active queue managers), it must partition its messages into ordered groups (such as by using a hashing function on an identifier) that each has a single route through the infrastructure.

- ▶ Detect occasional out-of-order delivery and drive manual/automated compensation logic.
Connections to IBM MQ and routes through IBM MQ can be prioritized so that out-of-order delivery is a rare occurrence, and is driven by recovery from temporary failures.
Applications can choose to use this approach to get continuous availability of IBM MQ through maintenance windows, but there is the risk of occasional out-of-order delivery.

A sample question to establish message ordering requirements

To establish message ordering requirements, ask the question, “Under what circumstances can your application handle out-of-order delivery of messages?”.

The question has the following parts and considerations:

- ▶ In all circumstances:
 - Your application is designed for parallel processing and can handle messages that are delivered in any order.
 - Messages can process in isolation with no affinity with other messages, or messages contain sequencing information that allows the application to store messages that arrive out-of-order for later processing.
- ▶ In exceptional circumstances:
 - Under normal processing, messages should be routed to the application in the order they are sent.
 - However, the availability to send new messages without interruption during maintenance windows or unexpected infrastructure failures is a higher business priority than ordered delivery.
- ▶ Under no circumstances:
 - The application requires a single route through the IBM MQ infrastructure from sender to receiver in all circumstances.
 - During maintenance windows or because of unexpected infrastructure issues, the technical design or SLA of the application means it can tolerate short interruptions in the availability of IBM MQ (typically up to 5 minutes).

2.2.4 Publish/subscribe subscription requirements

If an application is subscribing to a topic, there are various options for how the publish/subscribe features of IBM MQ can be configured.

Some features might be provided as options to project teams, and some might be mandated. Here are some examples:

- ▶ Uniqueness of topics - can be chosen enterprise-wide
 - You might choose to have a single topic hierarchy within your enterprise so that messages can be shared between applications, even if they are originally limited to an application.
 - In this case, you might need to enforce a topic naming convention across all project teams.

- ▶ Isolation of topics for scale - can be isolated by default
 - Even if a naming convention is in place across your enterprise, you might not want to group all topics into a single IBM MQ cluster.
 - Isolating and routing topics through different groups of queue managers can be important in a large-scale IBM MQ deployment.
- ▶ Isolation of topics for security - can be isolated by default

Business requirements might mean that messages transfer must to be limited to certain routes through your IBM MQ network.
- ▶ Subscription durability - needs input from applications
 - Applications might require messages to be kept for them when they are disconnected from IBM MQ, requiring durable subscriptions.
 - Applications might have multiple instances that must share work from a single stream of messages on a topic, requiring shared durable subscriptions.
 - Applications might want each instance to have a separate copy of each message, requiring non-durable subscriptions.
- ▶ Dynamic versus predefined subscriptions - needs input from applications
 - Applications might want to subscribe dynamically on many different topics. If so, they might or be willing to accept delays between registering a subscription and messages being routed to that subscription by publishing applications. This situation affects whether multiple queue managers can be allocated to the application or whether all applications publishing/subscribing from a set of topics must be allocated to the same queue manager.
 - Equally, applications might be willing to predefine their subscriptions to ensure that subscriptions exist on all queue managers before messages are published and to allow multiple queue managers to perform workload balancing to their application instances through an IBM MQ cluster.

The full details of the options for publish/subscribe are beyond the scope of this paper. For more information, see *IBM MQ: Using Publish/Subscribe in an IBM MQ Network*, found at:

<http://www.slideshare.net/DavidWare1/ame-2272-mq-publish-subscribe-network-pdf>

A sample question to establish publish/subscribe requirements

To establish publish/subscribe requirement, ask the following question, “If your application is creating a subscription, select all options that apply.” Here are the options:

- ▶ The topic is well-known and will not change without a new release of the application.
- ▶ The application generates topics and subscriptions dynamically in the run time:
 - It must receive all messages that are sent by publishers from the instant the subscription is created.
 - Messages should start flowing to the subscription as soon as possible after subscriptions are created, but a short gap where messages do not flow to this application is acceptable.
- ▶ The application has multiple instances and the following conditions are met:
 - The workload must be shared between these application instances (one message for the whole application).
 - When a message is published, all instances need their own copy of the message (one message per instance).

For more information about how to create a topic see 3.2.6, “Creating a topic destination” on page 82.

2.2.5 Establishing an application scaling model

In multiple active queue manager topologies, workload balancing and the sending of messages is a simpler task than ensuring all messages are consumed by applications when there are multiple destinations. There are generally two approaches:

- Route the messages to the applications.

You can use the cluster queue monitoring sample program (AMQSCLM) to route (and reroute) messages to queue instances where applications are listening if an application cannot listen to all queue instances. For more information, see 3.2.5, “Multiple-destination instances: No stranded or marooned messages” on page 74.

This approach allows multiple active queue managers to be used to provide continuous availability to receive new messages even if the application has a single instance, but does not allow the application to scale beyond the performance of an individual queue manager.

- Configure each instance of the application to listen to multiple queue managers

This approach allows you to scale beyond the performance of a single queue manager. It also avoids delays in rerouting messages that arrive on destinations that no application instance is listening while changes to applications connections are detected and propagated through the cluster through AMQSCLM. It also reduces the number of dynamic changes that occur in an IBM MQ cluster.

However, it requires the application to be configured/coded to listen to two destinations rather than one.

Within your enterprise, you might choose to provide active/passive only, or one or both of these options.

A sample question to establish the application scaling model

To establish the application scaling model, ask the following question, “How will your application scale?”.

This question has the following options:

- There is a single vertically scaled instance of the application, with active/passive high availability.
- The application is horizontally scaled with two or more instances.
 - Each instance can listen only to a single queue manager and cannot be reconfigured or changed to listen to multiple queue managers.
 - Each instance can be configured to listen to two queue managers concurrently.

For example, in Java EE applications, you can use the approach and code examples that are provided by GitHub, Inc., which can be found at the following website:

<https://github.com/ibm-messaging/mq-wlm-client>

For more information, see 3.2.4, “Multiple-destination instances” on page 65 and 3.2.5, “Multiple-destination instances: No stranded or marooned messages” on page 74.

2.2.6 Establishing availability requirements

There are multiple approaches to achieve message availability and service availability for IBM MQ. It is likely that all applications want the highest availability solution that meets their functional requirements, so some choices can be made universally. For example:

- ▶ All queue managers are configured for high availability failover by using the same technology (IBM MQ Appliance high availability, IBM MQ multi-instance with a highly available file system, high availability clustering software, or VM recovery).
- ▶ All queue managers for tier-1 applications are configured for disaster recovery replication to a secondary data center (synchronous or asynchronous with consistency groups where XA transactions are used).

Moving from single-queue manager active/passive high availability to active/active continuously available queue manager topologies places requirements on the applications. Certain choices regarding XA transaction management, absolute message ordering, and publish/subscriptions behavior prevent the use of multiple active queue managers to provide continuous service availability.

Application teams must make the choice between engineering their applications to use a continuously available active/active IBM MQ topology and accepting short planned failover delays to apply maintenance that might need to be coordinated with other teams sharing the infrastructure.

A sample question to establish availability requirements

To establish availability requirements, ask the following question, “What is the maximum acceptable time for which the IBM MQ service can be unavailable to send/receive new messages during planned maintenance or an unplanned infrastructure failure?”.

You have the following options:

- ▶ Up to 5 minutes
5 minutes is a guideline. You must test the real-world maximum failover time of your chosen high availability failover technology, in your infrastructure environment, under a representative production load.
- ▶ Less than 1 minute
The application can be engineered to use multiple active queue managers, including code-based workload management of XA connections if required, by accepting out-of-order delivery of messages and by using predefined durable subscriptions for publish/subscribe.

For more information, see 3.2.2, “Making the destination highly available” on page 35 and 3.2.3, “Single-destination instance for continuous availability” on page 51.

2.3 Self-service interfaces

Typically, *as a service* implementations include a web portal that users use to request and manage resources.

IBM MQ itself does not provide a built-in portal. IBM has two products that can be the basis of such a portal: IBM UrbanCode Deploy and IBM Cloud Orchestrator. However, many IBM MQ customers built their IBM MQ as a service provisioning portal into an enterprise-wide provisioning portal that is used to provision other kinds of middleware infrastructure, such as databases and application servers.

IBM MQ provides many interfaces that allow such portals to integrate with the messaging system. The interfaces include the control commands, IBM MQ Script (MQSC), client MQSC, IBM MQ Programmable Command Format (PCF), and IBM MQ Administration Interface (MQAI) interfaces, and existing tools, such as the IBM MQ Explorer.

The command sets are compared in the IBM MQ IBM Knowledge Center, found at:

http://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.ref.adm.doc/q083670_.htm

The following sections describe some of the high-level considerations that must be taken into account when you consider a self-service interface for IBM MQ.

2.3.1 IBM MQ administration interfaces

On platforms other than z/OS, you can use control commands to manage an IBM MQ installation, create and delete queue managers, and start and stop them. For z/OS, starting and stopping is done through MQSC commands; creating and deleting a queue manager are not functions that are provided as single commands. This capability can be achieved by using job control language (JCL).

Control commands

Control commands must be issued on the system itself. This requirement means a self-service interface must have the ability to either run or trigger commands remotely on the machine that is being used for hosting IBM MQ. Also, some mechanism of interpreting output to see whether the command was successful is necessary. One aspect that must be considered when using control commands is that the user control of these commands is system-wide and cannot be restricted without adding a higher-level control system (for example, in the self-service portal).

Queue manager administration

When considering a self-service interface for queue managers (that is, dynamic queue manager creation and deletion or reallocation in a pooling environment), consider the wider context. Elements, such as the machine on which the queue managers run and the number of queue managers per physical or virtual host, become important. If there is a single host per queue manager, then requesting a new queue manager also requires a new host and the setup that comes with it.

Creating the queue manager is one step, and can be achieved on distributed platforms by using the control commands and a series of JCL jobs on z/OS. However, after the queue manager is running, it must be customized to fit in the surrounding environment. For example, the security setup must be put in place so that users can connect to the queue manager, but other parties are kept out. Does the queue manager need access to a shared file system? If so, what restrictions should be applied? If the queue manager is joining a queue-sharing group (QSG) on z/OS, it must be carefully configured, including access to an IBM DB2® instance that is a member of the correct data sharing group.

When the queue manager must be decommissioned again, pay careful attention to identifying the correct steps to undo all of the changes that were made during setup. For example, to ensure that there are no security permissions left that somebody might exploit in the future.

In an environment where queue managers are being dynamically created and deleted, think about the amount of time that it is acceptable for the request to complete and keep a user waiting. IBM MQ Light in IBM Bluemix® pre-allocates the queue managers so that when a user requests the service, it is merely a matter of assigning them a queue manager that exists. This approach takes considerably less time than creating it dynamically and allows the request to take a few seconds at the most. The heavy lifting of deleting and re-creating a clean queue manager is done when the service is decommissioned and the queue manager is returned to the pool because the user does not need to wait for that work to complete.

Queue manager resource administration

After the queue manager is created and running, all of the resources within the queue manager can be administered by using one of the following administration tools:

- ▶ MQSC
- ▶ IBM MQ Programmable Command Format (PCF)
- ▶ IBM MQ Administration Interface (MQAI)

Choosing the preferred format for the environment depends on a number of factors:

- ▶ What is being used?
- ▶ Which platforms are supported?
- ▶ What technologies are used in the self-service portal?

Administration tools can be used to perform all the tasks that are required for a self-service portal, including creating queues, topics, channels, and altering and deleting them. The administration tools deal with IBM MQ objects meaning that security can be achieved by using IBM MQ security objects, even down to a fine-grained level. Security alternatively can be done outside of IBM MQ, for example, in the self-service portal.

Here are some of the administrative tools:

- ▶ MQSC

MQSC commands are script commands that are an effective way for individuals to control queue managers or the creation of automated scripts to run a list of commands. The output is easily consumable by humans. However, it is not ideal for programmatic processing.

MQSC commands can also be run from a client by using the `-c` flag, allowing for remote administration of IBM MQ resources. For more information, see *Bitesize Blogging: IBM MQ V8 - Client MQSC*, found at:

https://www.ibm.com/developerworks/community/blogs/messaging/entry/bitesize_blogging_mq_v8_client_mqsc

- ▶ IBM MQ Programmable Command Format (PCF)

This format uses IBM MQ messages that are put in a specific system queue in the queue manager. The queue manager monitors this queue and processes messages as soon as they arrive. A response can be sent back to a reply queue that is specified in the PCF message, allowing the administration program to retrieve the output in a well-defined format.

- ▶ IBM MQ Administration Interface (MQAI)

Available on platforms other than z/OS, this interface provides a wrapper around PCF messages, which provides a higher-level interface.

2.3.2 IBM MQ administration tools

The following tools use the IBM MQ administration interfaces to provide administration and monitoring user interfaces. These tools cannot directly be used in a centralized self-service portal, but they can be used with such a portal.

- ▶ IBM MQ Explorer

The IBM MQ Explorer is a graphical tool that users can use to view remotely and configure remotely their IBM MQ messaging system. The IBM MQ Explorer can administer local and remote queue manager resources and provide control commands for queue managers on the same machine.

In a self-service environment where users have control of the entire queue manager, a simple scenario is to use the self-service portal to provide the interface to allocate the queue manager and return the connection details to the user. The user can then use these details in their local IBM MQ Explorer installation to connect and do any further administration from there.

Using the IBM MQ Explorer as an administration tool in more complex scenarios, such as multitenancy, is also possible. However, this tool should be carefully considered, as users might be exposed to many features that can be restricted by security policies.

- ▶ IBM MQ Console appliance web UI

The IBM MQ Console is only available on the IBM MQ Appliance M2000. It provides a reduced set of IBM MQ Explorer features for administration. In addition, it provides graphical monitoring interfaces.

It can be used in a similar scenario to the IBM MQ Explorer, that is, where the users have control over an entire queue manager, and they can use a console similar to IBM MQ Explorer to manage their queue manager and the objects on it. It also provides an easy interface for creating queue managers on remote systems (that is, the IBM MQ Appliance), which IBM MQ Explorer cannot do.

In the IBM MQ Appliance M2000 announcement, IBM made the following statement of direction “IBM intends to deliver support for the new IBM MQ Console in IBM MQ¹”. In the future, this web console might form part of an as a service implementation.

¹ *IBM MQ Appliance M2000 delivers a rapid deployment option for IBM MQ V8 capability with built-in high availability*, found at <http://www.ibm.com/common/ssi/cgi-bin/ssialias?subtype=ca&infotype=an&supplier=877&letternum=ENUSZP15-0069>.

2.3.3 Self-service considerations in a multi-queue manager environment

In a messaging system, individual nodes rarely exist on their own. They exist in a wider web of connected systems and are used to route messages to their destination, applying some additional operations to them, such as fanning them out to back-end server farms.

As such, a self-service portal must cater to multiple users:

- ▶ **Developers:** Creating IBM MQ resources at a local level
- ▶ **Architect:** Connecting various local resources and establishing message routes, workload balancing, scaling, and more.
- ▶ **Administrator:** Controlling message flows to isolate parts of the messaging system for maintenance activities

How many of these features should be available through a self-service portal, and which can be done through manual intervention directly on the systems? However, having multiple ways of managing the same resources might cause issues with keeping everything synchronized.

Considering the developer perspective for a front-end application, a developer is most likely to care about where they can put the messages that are produced and possibly where to find a reply that was generated by a server as part of the processing. As such, they do not need to worry about any connectivity to the back end. The case for a server application developer is reversed because they worry only about retrieving messages from a queue and putting a reply message on the queue if one is requested.

When does the connectivity become relevant? During development, it is common to have request and reply applications running on the same queue manager. The moment this application goes into serious testing, the connectivity should be tested in a scenario similar to the production situation.

If applications are written well and obtain connectivity details of queue managers and queue names from environment variables or a similar configuration, a subset of people can manage the deployment of applications. This approach ensures that both client and server applications use the same queue name in a common cluster, taking care of any routing concerns. This approach raises the question of managing applications as a whole in the self-service environment.



Creating a self-service IBM MQ as a service menu for a portal

This chapter describes various approaches to creating a self-service IBM MQ as a service menu for use in a portal.

This chapter contains the following sections:

- ▶ 3.1, “Artifacts to build self-service interfaces for IBM MQ” on page 22
- ▶ 3.2, “Creating an IBM MQ as a service catalog” on page 23

3.1 Artifacts to build self-service interfaces for IBM MQ

IBM MQ provides several built-in capabilities, scripts, and APIs that users can use to create and administer queue managers and their related resources in IBM MQ. This section describes those capabilities.

3.1.1 IBM MQ control commands

You can use control commands to perform administrative tasks on queue managers. The usage and description of the IBM MQ control commands are documented in the IBM Knowledge Center, found at:

http://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.ref.adm.doc/q083010_.htm

3.1.2 IBM MQ Script commands

IBM MQ Script (MQSC) commands can be used to manage queue manager objects, including the queue manager itself, queues, process definitions, namelists, channels, client connection channels, listeners, services, and authentication information objects. You can issue MQSC commands to a queue manager by running **runmqsc**. You can issue commands interactively from a keyboard or you can redirect the standard input device (`stdin`) to run a sequence of commands from an ASCII text file. In both cases, the format of the commands is the same.

The usage and description of the MQSC commands are documented in the IBM Knowledge Center, found at:

http://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.adm.doc/q019950_.htm%23q019950___q019950_mqsc?lang=en

3.1.3 Programmable Command Formats

Programmable Command Formats (PCFs) define command and reply messages that can be exchanged between a program and any queue manager (that supports PCF) in a network. You can use PCF commands in a systems management application program for administration of IBM MQ objects, including authentication information objects, channels, channel listeners, namelists, process definitions, queue managers, queues, services, and storage classes. The application can operate from a single point in the network to communicate command and reply information with any queue manager, local or remote, by using the local queue manager.

The usage and description of PCF are documented in the IBM Knowledge Center, found at:

http://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.adm.doc/q019990_.htm?lang=en

3.1.4 Automation frameworks

This paper uses the IBM UrbanCode Deploy¹ and Chef² plug-ins to showcase how IBM MQ resources can be automated and exposed as services for the application teams to consume.

There are many other automation frameworks that can be used, for example, Chef alone or Puppet.³ The techniques and procedures that are described can be ported to one of these other automation frameworks.

This paper illustrates using UrbanCode Deploy because it provides tools that improve deployment speeds while simultaneously improving their reliability. The release automation tools in UrbanCode Deploy provide complete visibility into *n*-tiered deployments, enabling you to model processes that orchestrate complex deployments across every environment and approval gate.

3.2 Creating an IBM MQ as a service catalog

From an application team perspective, developers look at IBM MQ as a way to provide interfaces, either for a service they are developing or a service on which they rely. For example, a developer needs a queue to write an application to send and receive messages. In general, developers do not care about high availability (HA), scalability, and reliability, other than from the standpoint of a criteria that they need to fulfil. It is the infrastructure teams responsibility to work with the line-of-business (LOB) professionals to determine what are the other non-functional messaging requirements that must be supported.

The following sections describe how to build various self-services interfaces for IBM MQ resources that can be made available to users who want to use IBM MQ for their messaging needs.

3.2.1 Creating a service for single-destination instance

A single-destination instance corresponds to a topology where a queue or topic is created either on a new or an existing stand-alone queue manager instance without any scalability or clustering options. This section shows the steps to create a service representing a single-destination instance that the application team or any authorized user can use.

To create a process that defines the required services, complete the following steps:

1. Perform the prerequisite steps.

Before performing the steps for creating a service for a single-destination instance, see Appendix A, “IBM UrbanCode Deploy initial setup” on page 149 and complete the prerequisite steps.

¹ http://www.ibm.com/support/knowledgecenter/SS4GSP_6.0.0/com.ibm.udeploy.doc/topics/intro_ch.html?cp=SS4GSP_6.0.0

² <https://www.chef.io/>

³ <https://puppetlabs.com/>

2. Open the IBM UrbanCode Deploy web UI and click **Applications** → **Open MQaaS-Demo-App** (Figure 3-1). Open the MQaaS-Demo-App application under which the single-destination service can be created.

Note: For more information about creating the MQaaS-Demo-App, see “Creating an application” on page 179.

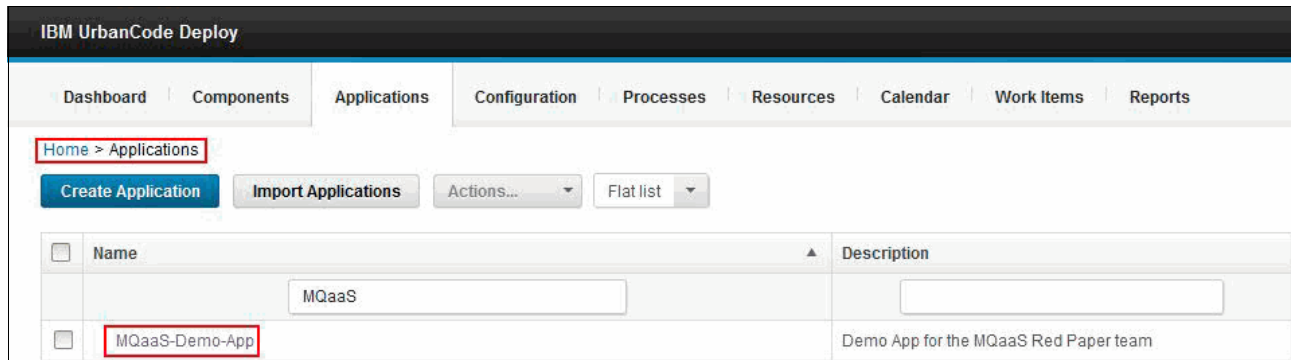


Figure 3-1 Open the MQaaS-Demo-App

3. Create an application process

One of the most important steps for automation is to create an application process. Application processes, like component processes, are created with the process editor. Application processes can run manually, automatically on some trigger condition, or on a user-defined schedule. When a component has several processes, the application determines which ones run and in which order. An application process is always associated with a target environment. When an application process runs, it interacts with a specific environment. At least one environment must be associated with the application before the process can run.

To create a process under which you can define the required services, click the **Processes** tab and click **Create Process** (Figure 3-2).

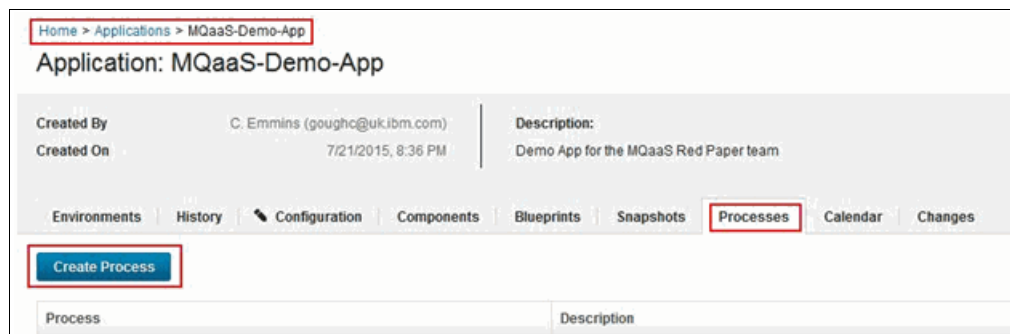
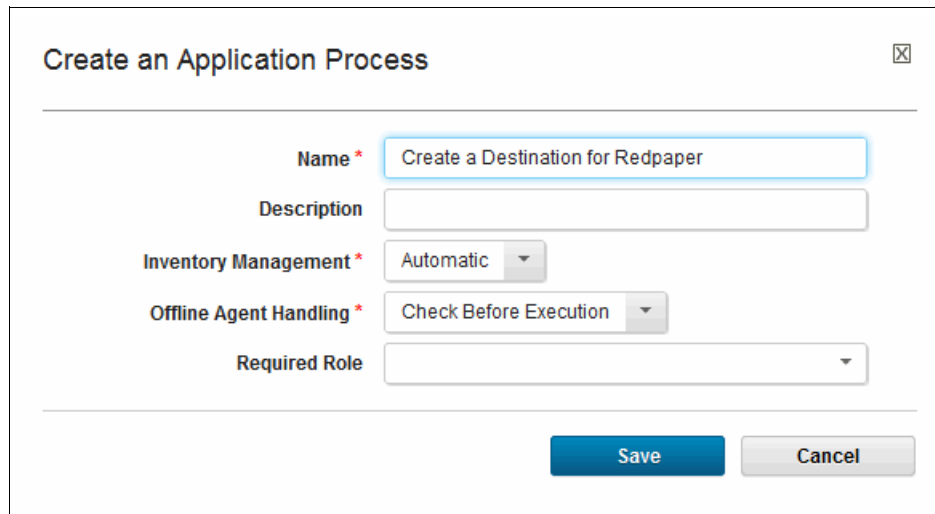


Figure 3-2 Step to create an application process

4. Type a name for the process, as shown in Figure 3-3 on page 25. In this example, the name of the process is Create a Destination for Redpaper. Click **Save**.



Create an Application Process

Name * Create a Destination for Redpaper

Description

Inventory Management * Automatic

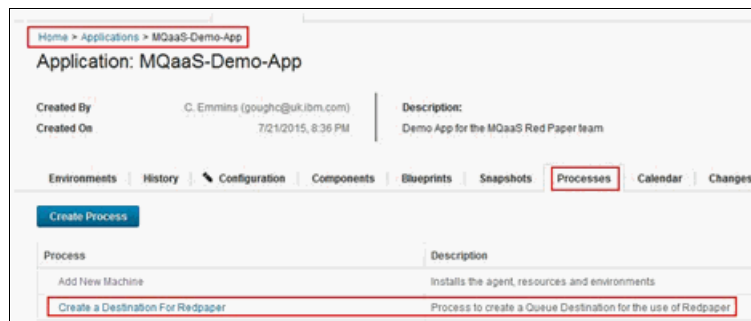
Offline Agent Handling * Check Before Execution

Required Role

Save **Cancel**

Figure 3-3 Enter the process name (Create a Destination for Redpaper)

- Figure 3-4 shows the created process. Find your created process from the list of processes and click it to open it.



Home > Applications > MQaaS-Demo-App

Application: MQaaS-Demo-App

Created By: C. Emmins (goughc@uk.ibm.com) | Description: Demo App for the MQaaS Red Paper team

Created On: 7/21/2015, 8:36 PM

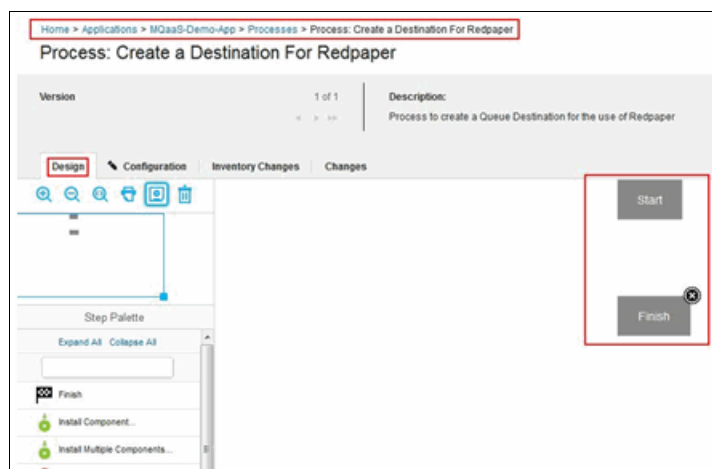
Environments | History | Configuration | Components | Blueprints | Snapshots | **Processes** | Calendar | Changes

Create Process

Process	Description
Add New Machine	Installs the agent, resources and environments
Create a Destination For Redpaper	Process to create a Queue Destination for the use of Redpaper

Figure 3-4 Shows the new process

- On the process designer palette (Figure 3-5), you see the Start and Finish buttons. You can use the process designer palette to customize how this process runs.



Home > Applications > MQaaS-Demo-App > Processes > Process: Create a Destination For Redpaper

Process: Create a Destination For Redpaper

Version: 1 of 1 | Description: Process to create a Queue Destination for the use of Redpaper

Design | Configuration | Inventory Changes | Changes

Start

Finish

Step Palette

Expand All | Collapse All

Finish

Install Component...

Install Multiple Components...

Figure 3-5 Process designer palette

7. In the Step Palette on the left (Figure 3-6), find the MQaaS folder, and you find the Add Queue Manager service under it. Drag it on to the process designer palette.

Note: For more information about creating the Add Queue Manager service, see Appendix A, “IBM UrbanCode Deploy initial setup” on page 149.

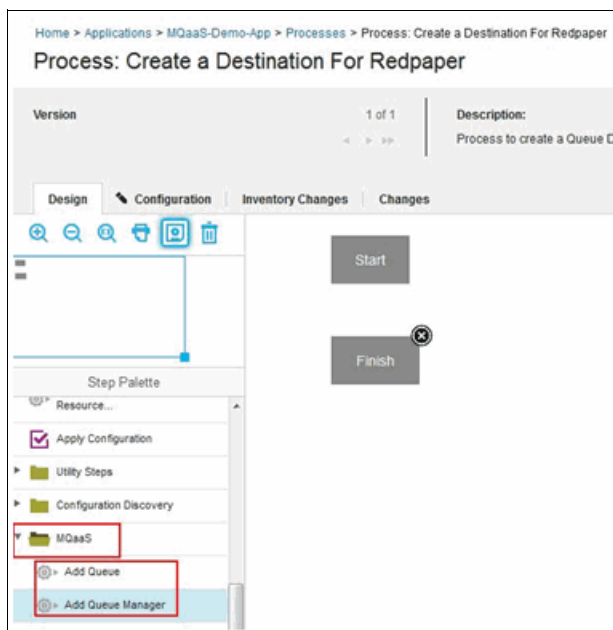


Figure 3-6 The various services under the MQaaS application

8. Dragging and dropping the Add Queue Manager service opens a wizard for input values (Figure 3-7):
 - a. Specify the name, which is **Create and Start a Queue Manager** in this example.
 - b. You can accept the default values for the other fields.
 - c. Click **Save**.

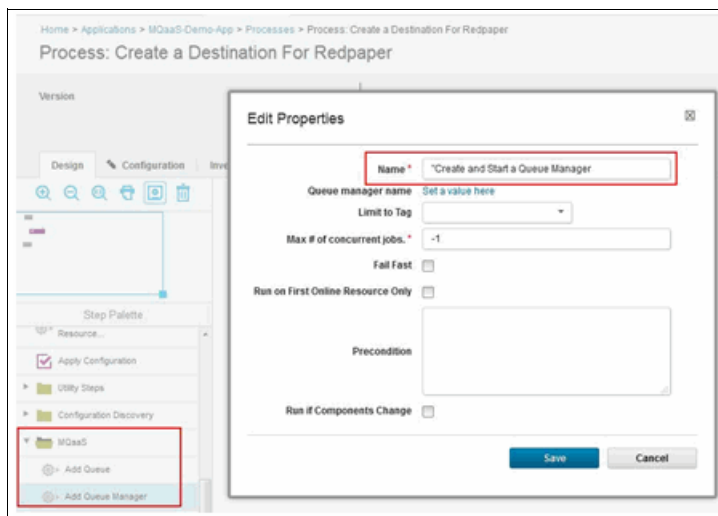


Figure 3-7 Property page to control the execution of the Queue Manager

9. When you hover your cursor over the Start box, you see a forward arrow mark. Drag it onto the Create and Start Queue Manager box (Figure 3-8).

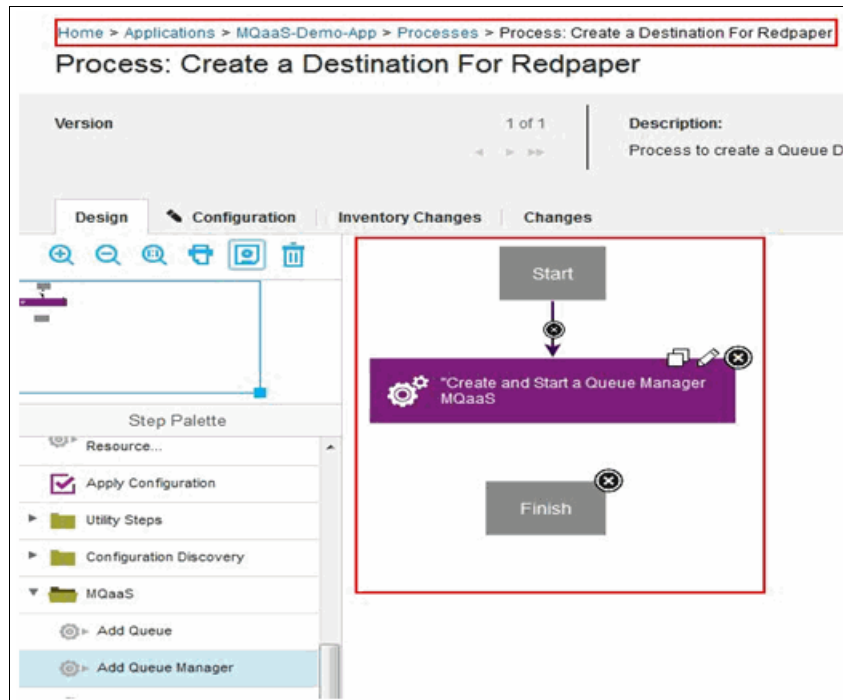


Figure 3-8 Link the resources together

10. From the menu on the left, in the MQaaS folder, you find the Add Queue service under it. Drag it on to the process designer palette (Figure 3-9).

Note: For more information about creating the Add Queue component process, see “Creating a component process: Add Queue” on page 172.

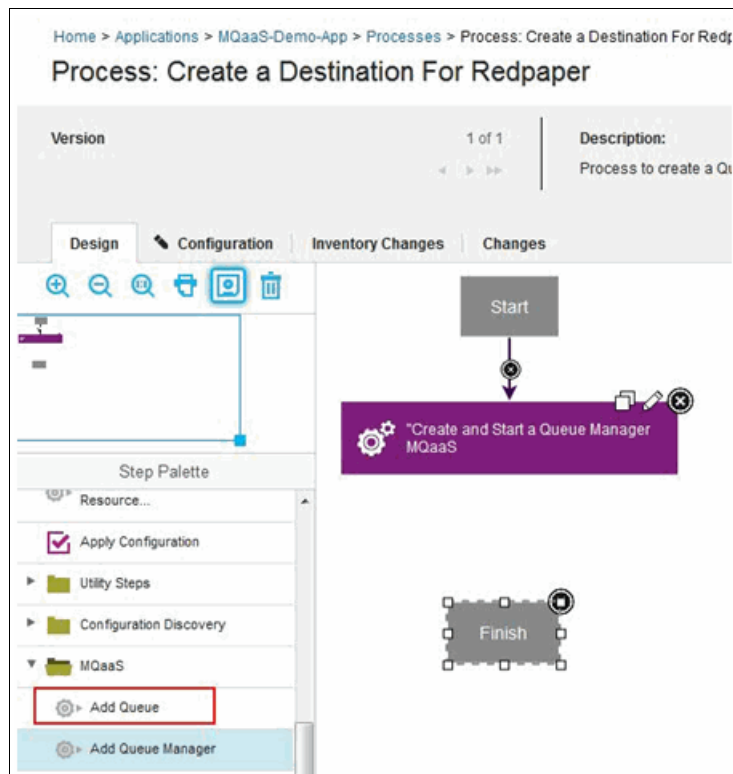


Figure 3-9 Add the Queue service to the process

11. Dragging and dropping the Add Queue service opens a wizard for input values (Figure 3-10 on page 29).
- Specify the name, which is **Define Queue** in this example.
 - Set the Queue manager name value as **\${QueueManagerName}**.
 - Accept the default values for the other fields.
 - Click **Save**.

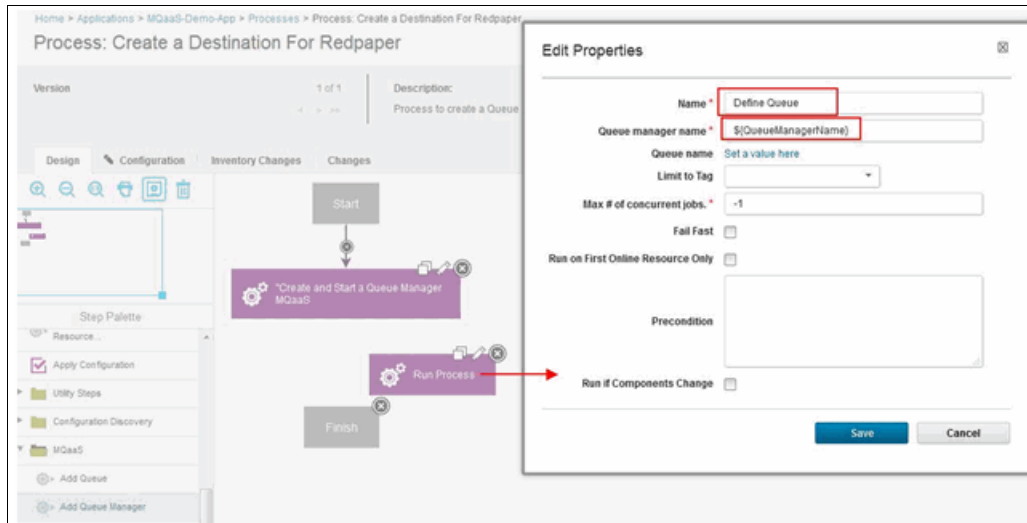


Figure 3-10 Enter the queue name for this service

12. When you hover your cursor over Create and Start a Queue Manager, you see a forward arrow mark; drag the forward arrow mark from Define Queue on to Finish (Figure 3-11).

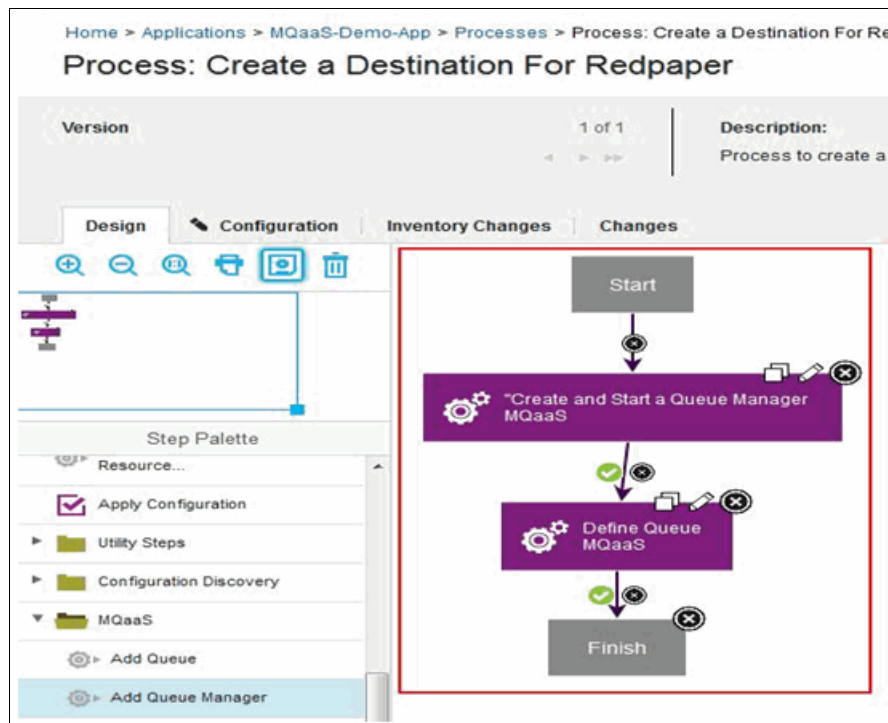


Figure 3-11 Link the sequence of the services together

13. Click **Save** to save your design, as shown in Figure 3-12.

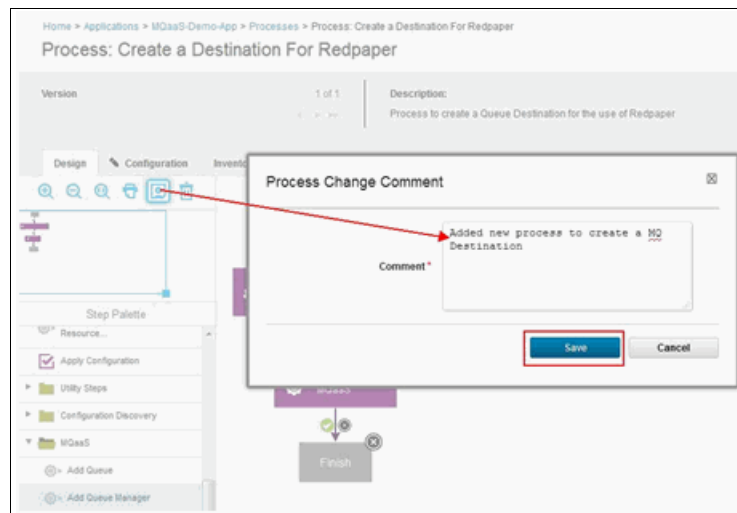


Figure 3-12 Add comment and save the created process

14. After the process is created, you must create and associate an environment for the process.

Click the **Processes** tab and then click the **Environments** tab (Figure 3-13).

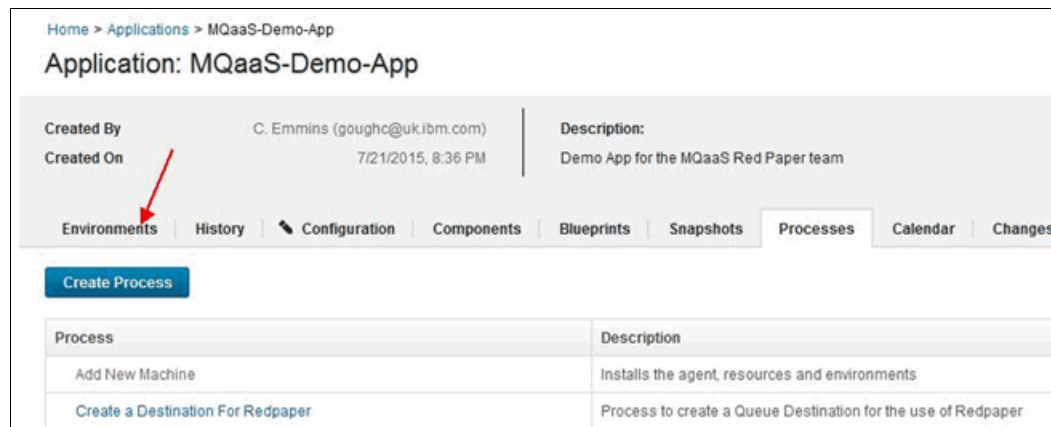


Figure 3-13 Associate the application process with the environment

15. Find the machine environment and click the play button (Figure 3-14 on page 31).

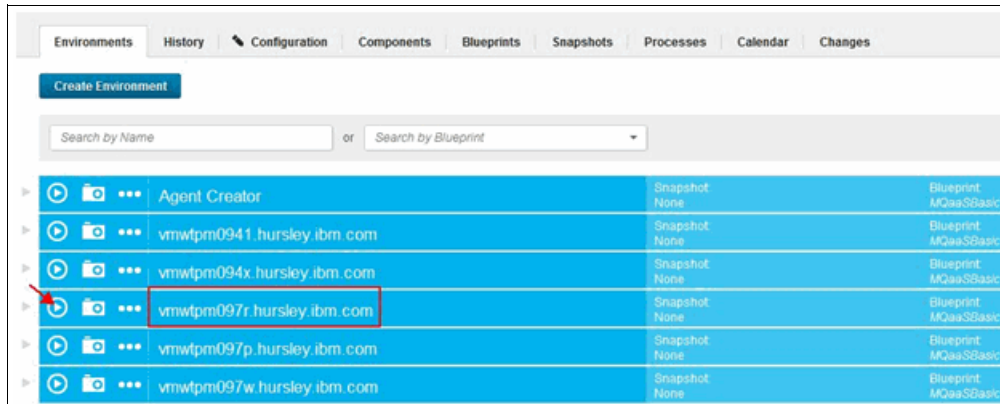


Figure 3-14 Select the appropriate environment to run the service

16. Clicking the play button opens a wizard, as shown in Figure 3-15.

- Select the process to run **Create a Destination For Redpaper**.
- Enter the Queue manager name: **MQQM-REDP**
- Enter the Queue name: **RED**
- Click **Submit**.

This action runs the process and displays the result.

Run Process on vmwtpm097r.hursley.ibm.com

Only Changed Versions ☒

Process * **Create a Destination For Redpaper**

Select a snapshot, or choose versions for individual components.

Snapshot

"Create and Start a Queue Manager / Queue manager name * **MQQM-REDP**

Define Queue / Queue name * **RED**

Schedule Deployment? ☐

Description

Submit Cancel

Figure 3-15 Specify the queue manager and queue name to be created through this process

Running the service by using REST API and JSON

Every service that is created through UrbanCode Deploy can be run by starting an appropriate REST API. This REST API can be started by any automation framework having authority in the wider PaaS environment.

There are two methods of passing the required parameters. Although this section shows a way to pass a JSON file as input to the REST API, all subsequent sections show how the REST API can be started by passing the appropriate parameters.

Example 3-1 shows passing the parameters through a JSON file as input to the REST API.

Note: Replace <userID:password> in the example with the appropriate values, for example, myuser@mycompany.com:secret.

Example 3-1 Pass a JSON file as input

```
curl -k -u '<userID:password>'
https://udeploy.in.ibm.com:8443/cli/'
https://udeploy.in.ibm.com:8443/cli/applicationProcessRequest/request -X PUT -d
@createDestination.json
```

Create the createDestination.json file and include the parameters that are shown in Example 3-2 to create the destination instance.

Example 3-2 JSON file createDestination.json

```
{
  "application": "MQaaS-Demo-App",
  "applicationProcess": "Create a Destination For Redpaper",
  "environment": "vmwtpm097w.hursley.ibm.com",
  "onlyChanged": "false",
  "properties": {
    "QueueManagerName": "QM4",
    "QueueName": "LocalQueue"
  }
}
```

CURL: CURL is a tool to transfer or pull data from the server. For more information about the CURL tool, see the CURL documentation at the following website:

<http://curl.haxx.se/docs/manpage.html>

The command that is shown in Example 3-1 submits a request to run the process to create a queue manager that is named QM4, creates a queue that is called LocalQueue for the same queue manager, and returns the unique ID of the request by using a JSON response (Example 3-3).

Example 3-3 JSON response

```
{"requestId": "c8095755-f615-46ed-bdcc-77ffa4a6fd28"}
```

You can then request the status of the response by using a command, as shown in Example 3-4.

Note: Replace <userID:password> in the example with the appropriate values, for example, myuser@mycompany.com:secret.

Example 3-4 Request status command

```
curl -k -u '<userID:password>'
'https://udeploy.in.ibm.com:8443/cli/applicationProcessRequest/requestStatus?requestId=c8095755-f615-46ed-bdcc-77ffa4a6fd28' -X GET
```


A response to this REST API call is shown in Example 3-5, which states that the queue manager and the queue were created and started successfully.

Example 3-5 Response to status request

```
{ "status": "CLOSED", "result": "SUCCEEDED" }
```

Single-destination services considerations

This section includes considerations that apply to single-destination services.

Benefits of single-destination services

Single-destination services can help in the following ways:

- ▶ An application can send and receive messages between components in the application.
- ▶ Applications can connect to an existing service that is provided by another team or application.
- ▶ Creating a destination helps applications or service to consume with or without using selectors.
- ▶ Applications can use XA transactions involving multiple resource managers.
- ▶ Applications can process messages in sequence or maintain strict message ordering.

Single-destination services considerations for developers

Table 3-1 show the advantages and disadvantages of using a single-destination service for development.

Table 3-1 Using single-destination services - considerations for developers

Advantages	Disadvantages
<ul style="list-style-type: none">▶ Provides isolation of messages.▶ Supports XA transactions for applications involving multiple resource managers.▶ Can be configured for failover and high availability.▶ Applications can perform both Put and Get messages from this destination.▶ Ordered delivery can be achieved.▶ Automatic client reconnection can be achieved with no application changes and with minimal configuration.	<ul style="list-style-type: none">▶ Messages can be stranded on the destination if the queue manager is not configured for failover or high availability.▶ Does not provide continuous availability for applications, so applications can encounter downtime.▶ Does not provide scalability for applications.

Single-destination services considerations for infrastructure teams

Table 3-2 show the advantages and disadvantages of using a single-destination service by the infrastructure team.

Table 3-2 Using single-destination services - considerations for the infrastructure team

Advantages	Disadvantages
<ul style="list-style-type: none">▶ The destination can be created on a new queue manager.▶ The destination can be created in an existing queue manager based on load and demand of the application.▶ A single destination can be provisioned to be used by multiple applications.▶ It is easier to track and monitor the movement of messages.	<ul style="list-style-type: none">▶ Difficult to provide application-level isolation when multiple applications use the same destination.▶ There is a dependency on network-attached storage devices to configure failover or high availability.▶ Requires additional skills on network file systems configuration.

Single-destination topology pattern

There are several ways an application can use the single-destination instance. This section provides multiple options on how the single destination can be used.

Figure 3-16 shows a simple topology of an application connecting to the destination that is created on the queue manager.

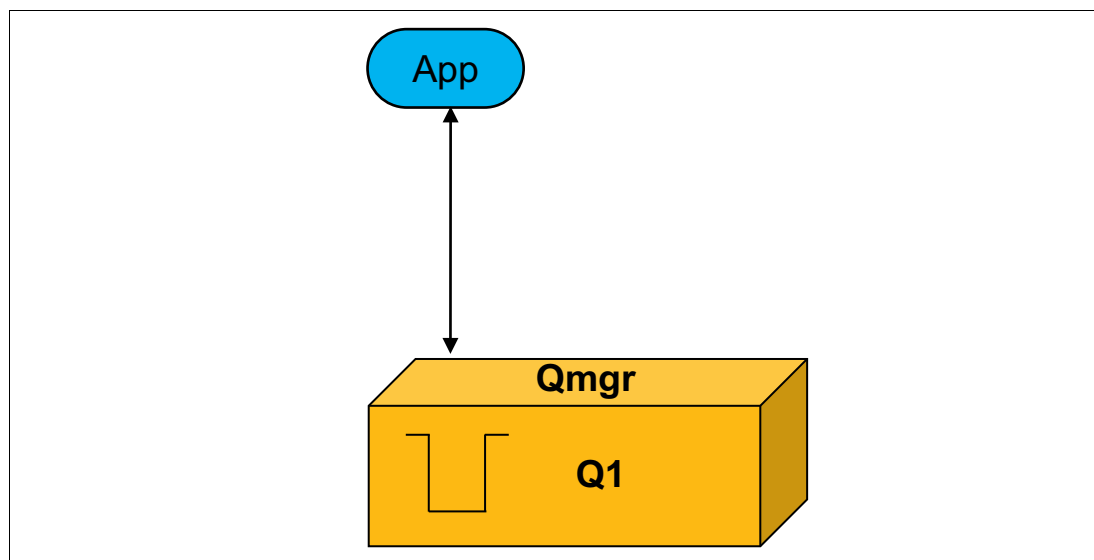


Figure 3-16 An application connecting to a single destination

Figure 3-17 on page 35 shows the usage of an active and a passive queue manager accessing a network file system. All the resources are common to both queue managers, and there is only one instance of the destination.

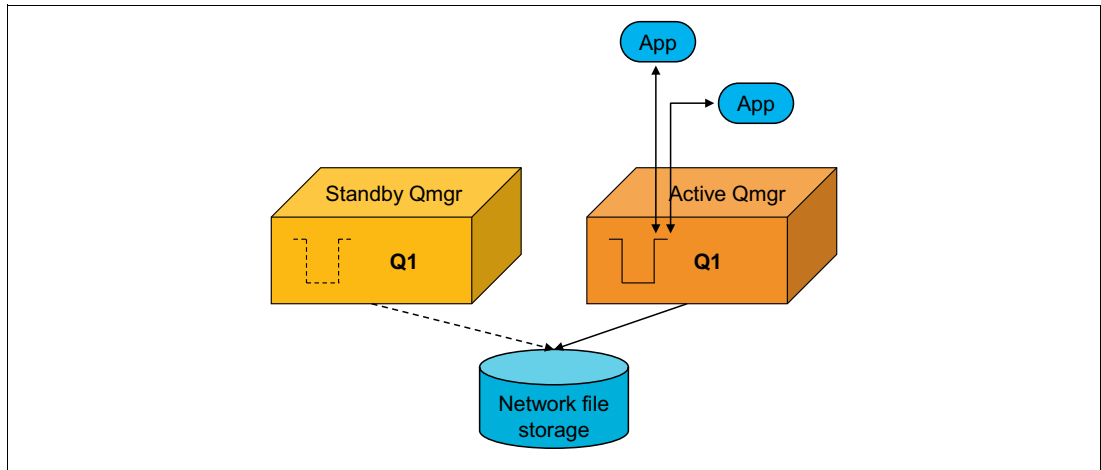


Figure 3-17 An application connecting to a single destination with HA

3.2.2 Making the destination highly available

The objective of making a destination highly available is to achieve 24x7 availability of messaging resources. Highly available destinations ensure that there is no single point of failure for messaging resources in case the system on which the destination is running fails.

The following sections explain how to set up IBM MQ to make a destination highly available. Configuring highly available destinations requires that you create multiple environments for the active/standby pair. You need two separate application processes and one component process to create an active and standby instance. For more information about creating an application and a component, see Appendix A, “IBM UrbanCode Deploy initial setup” on page 149.

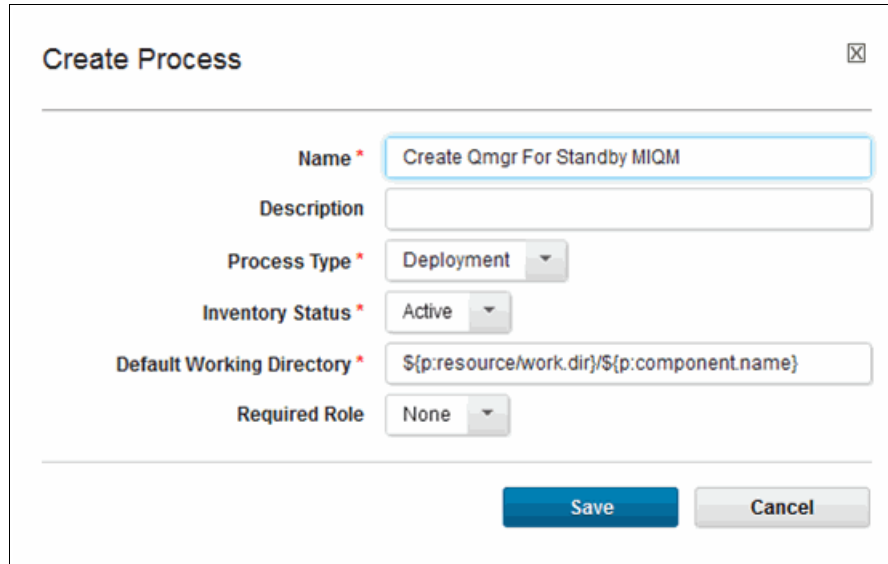
Creating a standby queue manager for the MQaaS component process

A component process is a series of user-defined steps that operate on the artifacts of a component. Each component has at least one process that is defined for it, and it can have several.

This section describes the steps to create a component process that uses the IBM MQ plug-in for IBM UrbanCode Deploy. This component process includes services that are required to create and start the queue manager instance in a standby mode.

Complete the following steps:

1. On the IBM UrbanCode Deploy web UI and from the home page, click **Components** → **MQaaS** → **Process**. Create a process called Create Qmgr For Standby MIQM (Figure 3-18).



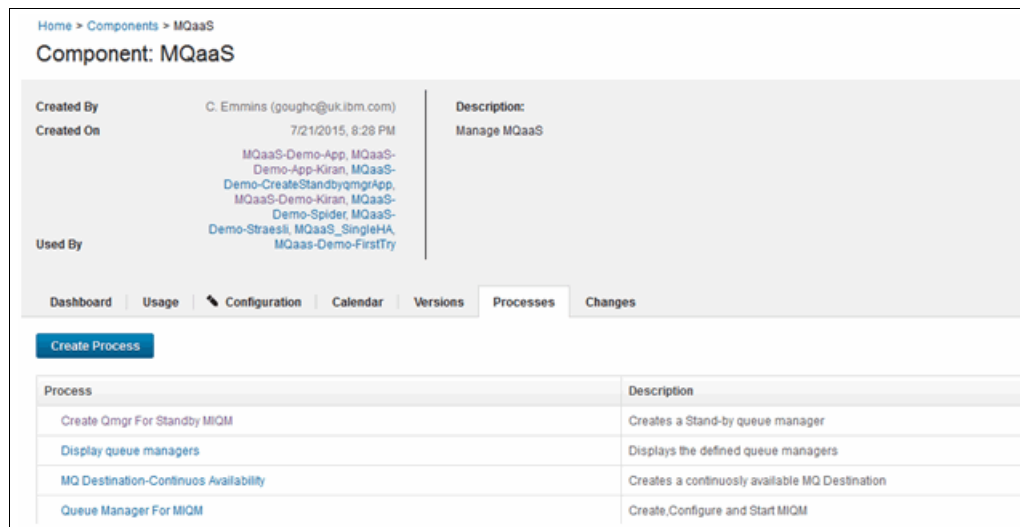
The 'Create Process' form in the IBM UrbanCode Deploy web UI. It contains the following fields and values:

- Name ***: Create Qmgr For Standby MIQM
- Description**: (empty)
- Process Type ***: Deployment
- Inventory Status ***: Active
- Default Working Directory ***: \${p:resource/work.dir}/\${p:component.name}
- Required Role**: None

Buttons: Save, Cancel

Figure 3-18 Create a component process

2. Click the **Create Qmgr For Standby MIQM** process in the list of processes (Figure 3-19).



The 'Component: MQaaS' page in the IBM UrbanCode Deploy web UI. It shows the component details and a list of processes.

Component: MQaaS

Created By: C. Emmins (goughc@uk.ibm.com)
Created On: 7/21/2015, 8:28 PM
Description: Manage MQaaS

Used By: MQaaS-Demo-App, MQaaS-Demo-App-Kiran, MQaaS-Demo-CreateStandbyqmgrApp, MQaaS-Demo-Kiran, MQaaS-Demo-Spider, MQaaS-Demo-Straesli, MQaaS_SingleHA, MQaaS-Demo-FirstTry

Navigation: Dashboard | Usage | Configuration | Calendar | Versions | **Processes** | Changes

Create Process

Process	Description
Create Qmgr For Standby MIQM	Creates a Stand-by queue manager
Display queue managers	Displays the defined queue managers
MQ Destination-Continuous Availability	Creates a continuously available MQ Destination
Queue Manager For MIQM	Create, Configure and Start MIQM

Figure 3-19 List of processes for the MQaaS component

3. Open the Create Qmgr For Standby MIQM process to display an empty process designer palette (Figure 3-20 on page 37).

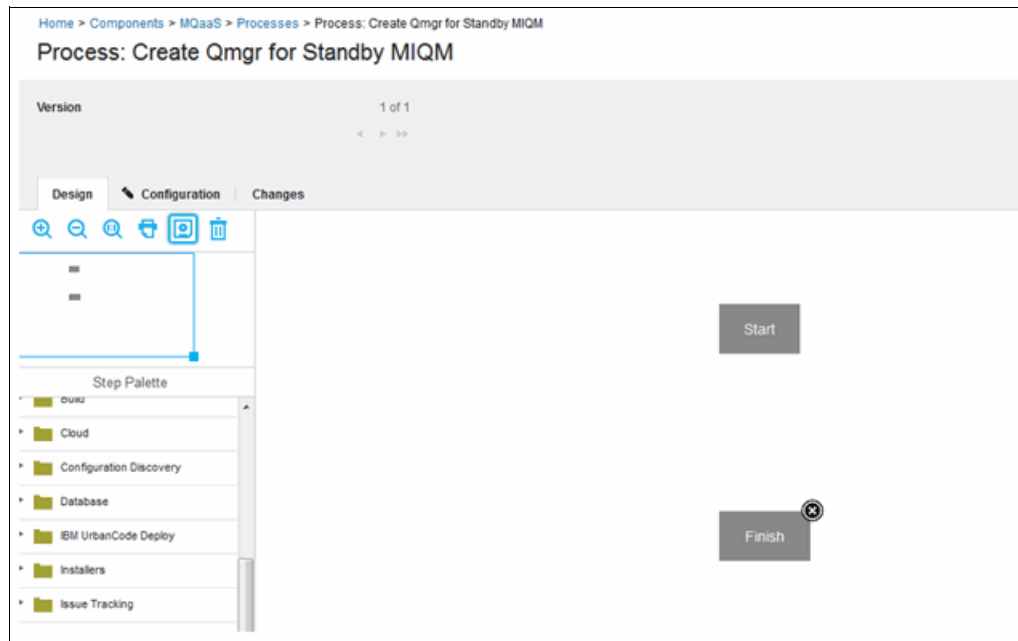


Figure 3-20 Initial designer palette for the process is created

4. To create a standby queue manager, you must run the **addmqinf** script. For more information about the **addmqinf** script, see the IBM Knowledge Center at the following website:

http://www-01.ibm.com/support/knowledgecenter/SSFKSJ_7.5.0/com.ibm.mq.ref.adm.doc/q083060_.htm

To run custom scripts, you can use the Shell feature in IBM UrbanCode Deploy.

5. As shown in Figure 3-21, in the Step Palette, click **Scripting** → **Shell**, drag Shell, and use the script in Example 3-6 to run the service.

Example 3-6 addmqinf script used in this example

```
/opt/mqm/bin/addmqinf -s QueueManager -v Name=${QueueManagerStby} -v  
Directory=${QueueManagerStby} -v Prefix=/var/mqm -v  
DataPath=/miqmda/HAUCD/qmgrs/${QueueManagerStby}
```

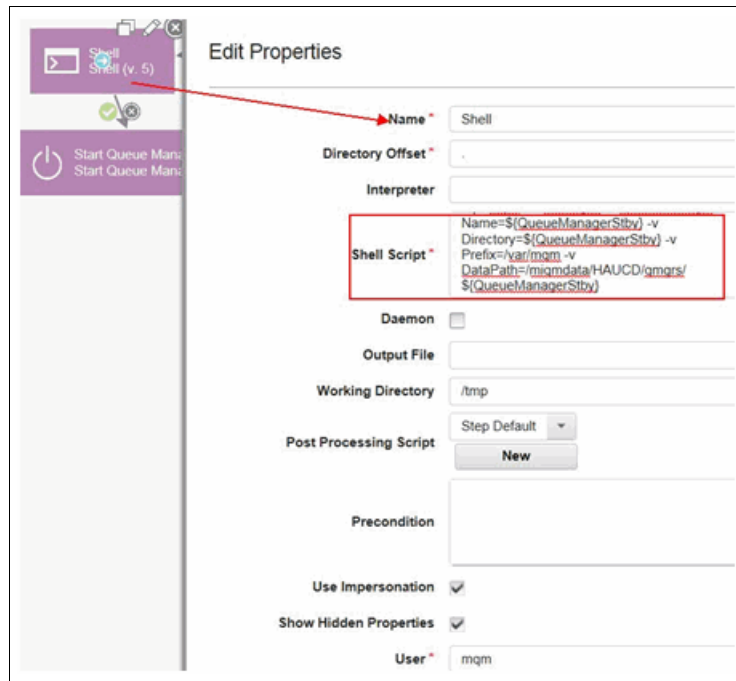


Figure 3-21 Create a custom shell script to define a standby queue manager

6. After the custom shell script is added, add the start queue manager script to start the standby queue manager.
 - a. From the Step Palette, click **Middleware** → **Messaging** → **WebSphereMQ** → **Start Queue Manager**.
 - b. Drag the Start Queue Manager service onto the process designer palette.
 - c. Complete the properties (see Figure 3-22).
 - d. Click **Save**.

Edit Properties

Name *	Start Queue Manager
Queue Manager Name *	\${QueueManagerStby}
Command Directory *	/opt/mqm/bin
Additional Arguments	-x
Working Directory	
Post Processing Script	Step Default New
Precondition	
Use Impersonation	<input checked="" type="checkbox"/>
Show Hidden Properties	<input checked="" type="checkbox"/>
User *	mqm
Group	

Figure 3-22 Properties to complete for the Start Queue Manager service

7. With both resources added, link both resources together (see Figure 3-23).

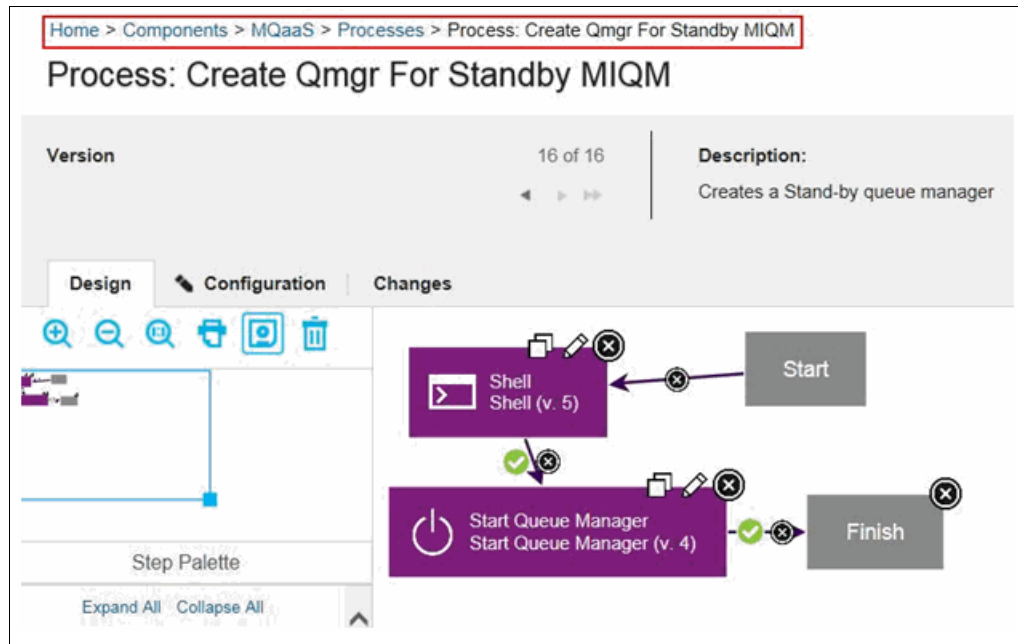


Figure 3-23 Process designer showing the completed process

Creating an active queue manager process for the MQaaS component

This section describes the steps to create a component process that uses the IBM MQ plug-in for IBM UrbanCode Deploy. This component process includes services that are required to create and start the queue manager instance in Active mode.

Complete the following steps:

1. From the IBM UrbanCode Deploy web UI home page, click **Components** → **MQaaS** → **Process**, and create a process named Queue Manager for MIQM. Use this service to create the active queue manager.

2. Click **Queue Manager for MIQM** in the list of processes to open it in Design mode. Drag the Create Queue Manager process step from the **Middleware** → **Messaging** → **WebSphere MQ** menu of the Step Palette, as shown in Figure 3-24. In this sample, the following properties are used to specify additional arguments:

```
ld /miqmdata/${QueueManagerName}/logs  
md /miqmdata/${QueueManagerName}/qmgrs
```

Edit Properties

Name * Create Queue Manager

Queue Manager Name * \${QueueManagerName}

Queue Manager Description

Command Directory * \${p.resource/WebSphereMQ.commandPath}

Additional Arguments
-ld /miqmdata/\${QueueManagerName}/logs
-md /miqmdata/\${QueueManagerName}/qmgrs

Working Directory

Post Processing Script Step Default

Precondition

Use Impersonation ☒

Show Hidden Properties ☐

User * mqm

Figure 3-24 Properties for creating a queue manager service

3. From the Step Palette, drag the **Scripting** → **Shell** resource and update it, as shown in Figure 3-25. This action gets the output from the **dspmqiinf** command so that it is used with the queue manager.

Example 3-7 shows the script that runs the **dspmqiinf** command.

Example 3-7 Shell script to run the dspmqiinf command

```
${p:resource/WebSphereMQ.commandPath}/dspmqiinf -o command ${QueueManagerName}
```

Edit Properties

Name *

Directory Offset *

Interpreter

Shell Script *

Daemon ☐

Output File

Working Directory

Post Processing Script

Precondition

Use Impersonation ☒

Show Hidden Properties ☐

Figure 3-25 Create the shell script to run the **dspmqiinf** command

4. Create the application properties to be used with the queue manager, for example, add a user or a group to be used for this queue manager. To create an application-specific property, drag **IBM UrbanCode Deploy** → **Applications** → **Create Application Property** from the Step Palette (see Figure 3-26).

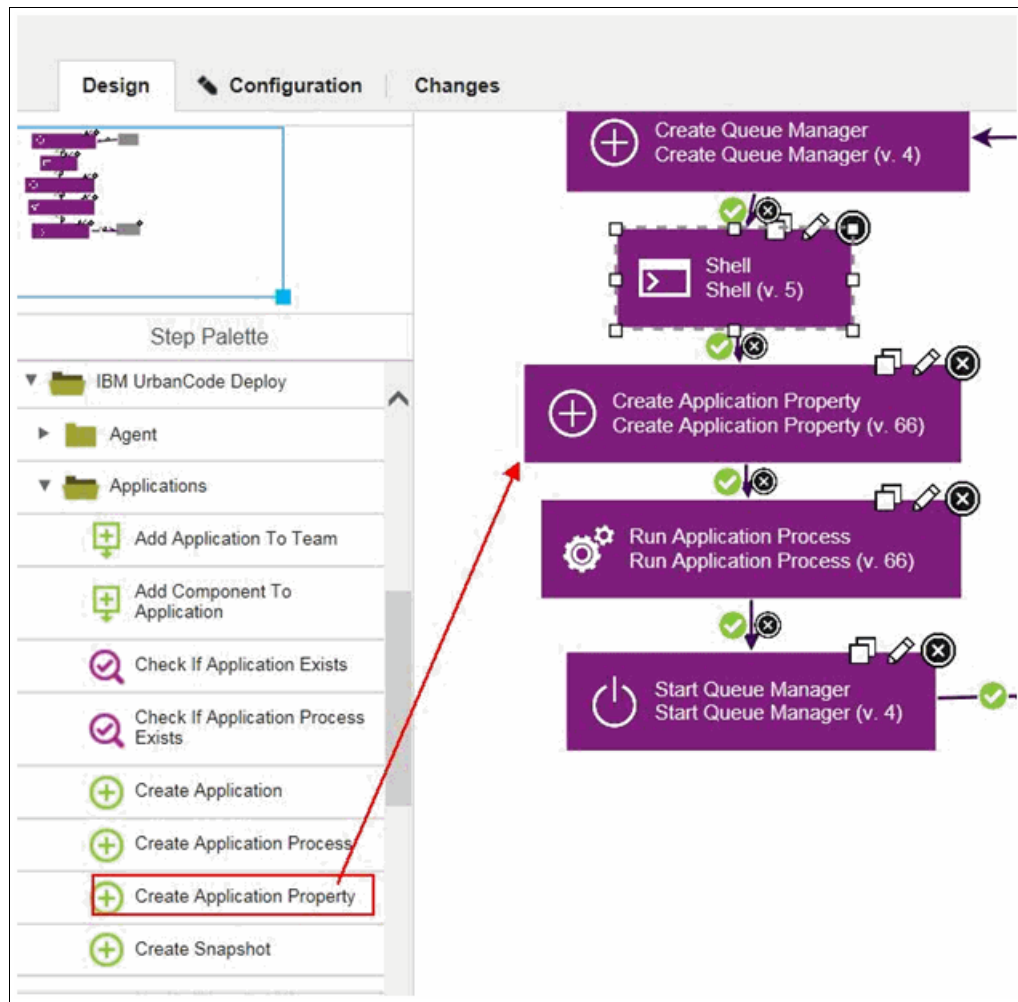


Figure 3-26 Add an application property for the process

5. Update the properties, as shown in Figure 3-27.

Edit Properties

Name * Create Application Property

Application * MQaaS-Demo-App

Property Name * QueueManagerStby

Property Value \${QueueManagerName}

Secure? ☐

Working Directory

Post Processing Script Step Default

Precondition

Use Impersonation ☒

Show Hidden Properties ☒

User * mqm

Group

Password ••••

Figure 3-27 Set the application-specific properties for the queue manager

- The queue manager name must be passed to the Create Standby Queue Manager process. For this example, keep the name of the queue manager the same for both the active and standby instances. In the Step Palette, click **IBM UrbanCode Deploy** → **Applications** → **Run Application Process** and update the properties, as shown in Figure 3-28.

Edit Properties

Name *	Run Application Process
Application Name *	MQaaS-Demo-App
Application Process Name *	Create a Stand-by QueueManager
Environment Name *	vmwtpm097p.hursley.ibm.com
Snapshot Name	
Only Changed Versions	<input type="checkbox"/>
Component Versions	<div></div>
Wait For Application Process To Finish	<input type="checkbox"/>
Working Directory	
Post Processing Script	Step Default <div>New</div>
Precondition	<div></div>
Use Impersonation	<input checked="" type="checkbox"/>

Figure 3-28 Properties set for the Running Application Process

7. Add the Start Queue Manager service from the Step Palette to start the queue manager that was created previously. To add the Start Queue Manager service, click **Middleware** → **Messaging** → **WebSphereMQ** → **Start Queue Manager** and update the properties (Figure 3-29).

The screenshot shows the 'Edit Properties' dialog for the 'Start Queue Manager' service. The dialog has a title bar with a close button. The properties are as follows:

- Name ***: Start Queue Manager
- Queue Manager Name ***: \${QueueManagerName}
- Command Directory ***: \${p:resource/WebSphereMQ.commandPath}
- Additional Arguments**: -x
- Working Directory**: (empty)
- Post Processing Script**: Step Default (dropdown), New (button)
- Precondition**: (empty)
- Use Impersonation**: ☒
- Show Hidden Properties**: (checkbox)
- User ***: mqm
- Group**: (empty)

Figure 3-29 Properties to start the queue manager process

These steps created the process to create the queue manager, set the required properties for the queue manager, and start the queue manager. The completed design should look similar to Figure 3-30.

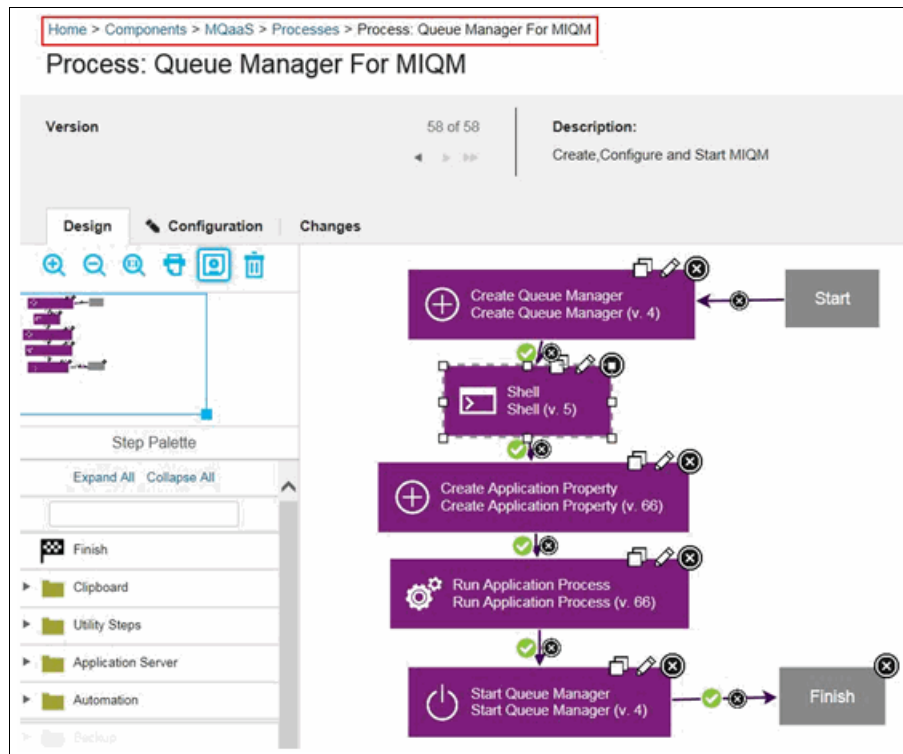


Figure 3-30 Overall flow to create a queue manager as a multi-instance queue manager

Creating an application process for an active queue manager

Now that the required component processes are created, they must be associated with the application process to be run. This section explains the steps that are required to create an application process that uses the component process that is created in “Creating an active queue manager process for the MQaaS component” on page 40.

Complete the following steps:

1. Create an application process to associate the active queue manager process previously created. From the IBM UrbanCode Deploy web UI home page, click **Applications** → **MQaaS-Demo-App** → **Process** → **Create a new process**. Name the application process Create Active QueueManager (see Figure 3-31).

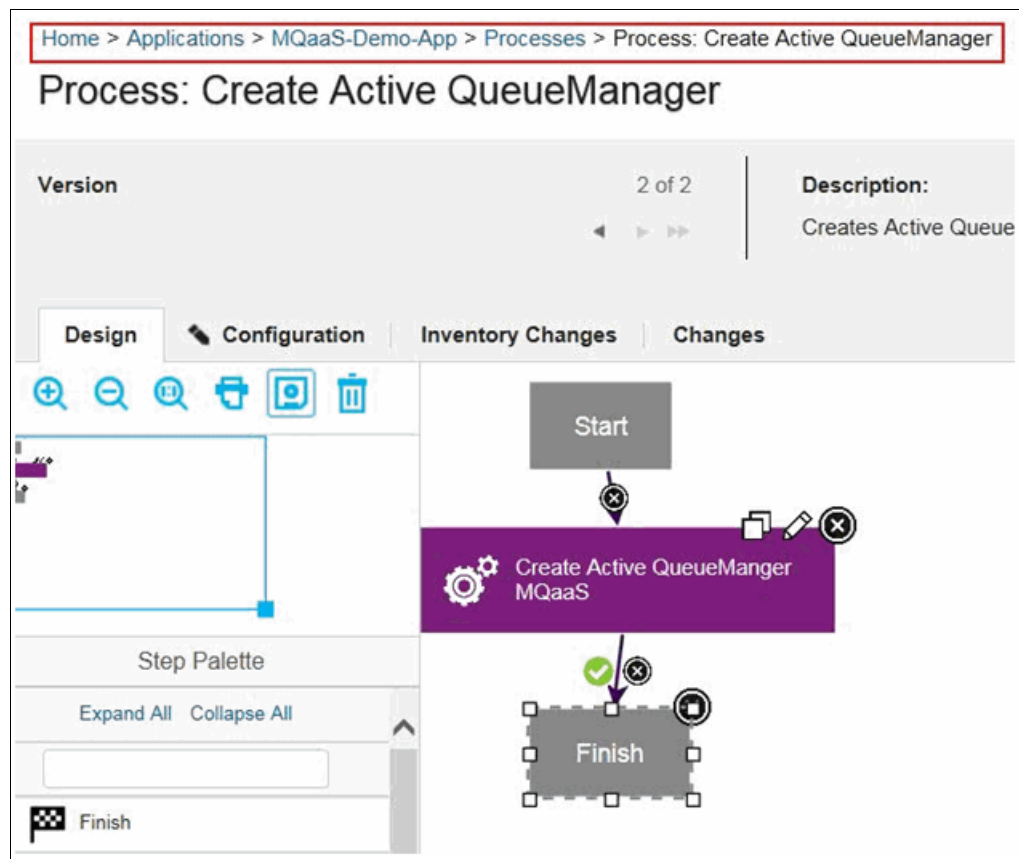
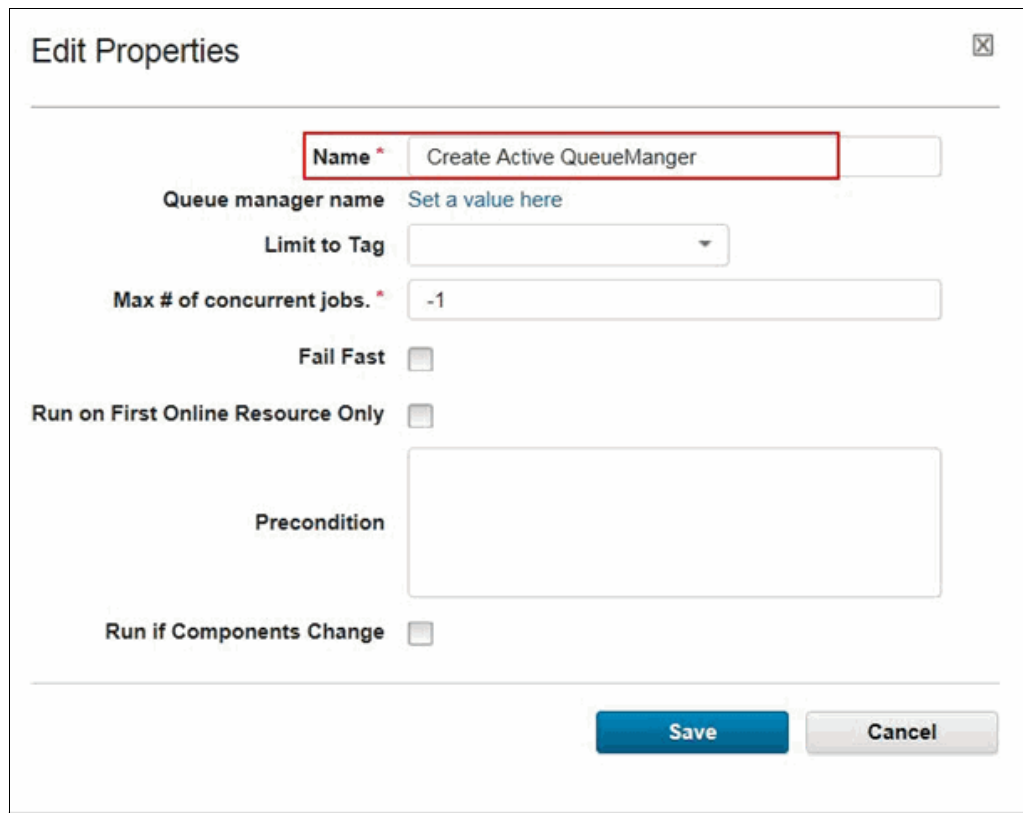


Figure 3-31 Create an active queue manager application process

2. Update the properties, as shown in Figure 3-32. Click **Save**.



The screenshot shows a dialog box titled "Edit Properties" with a close button in the top right corner. The dialog contains several configuration fields:

- Name ***: A text field containing "Create Active QueueManger", which is highlighted with a red rectangular border.
- Queue manager name**: A text field with the placeholder "Set a value here".
- Limit to Tag**: A dropdown menu.
- Max # of concurrent jobs. ***: A text field containing "-1".
- Fail Fast**: A checkbox that is currently unchecked.
- Run on First Online Resource Only**: A checkbox that is currently unchecked.
- Precondition**: A large, empty text area.
- Run if Components Change**: A checkbox that is currently unchecked.

At the bottom right of the dialog, there are two buttons: a blue "Save" button and a grey "Cancel" button.

Figure 3-32 Set the properties for the Create Active QueueManager process

Creating an application process for the standby queue manager

This section describes the steps to create an application process and associate it with the standby component process.

Complete the following steps:

1. From the IBM UrbanCode Deploy web UI home page, click **Applications** → **MQaaS-Demo-App** → **Process**. Create a process that is named Create a Stand-by QueueManager (see Figure 3-33).

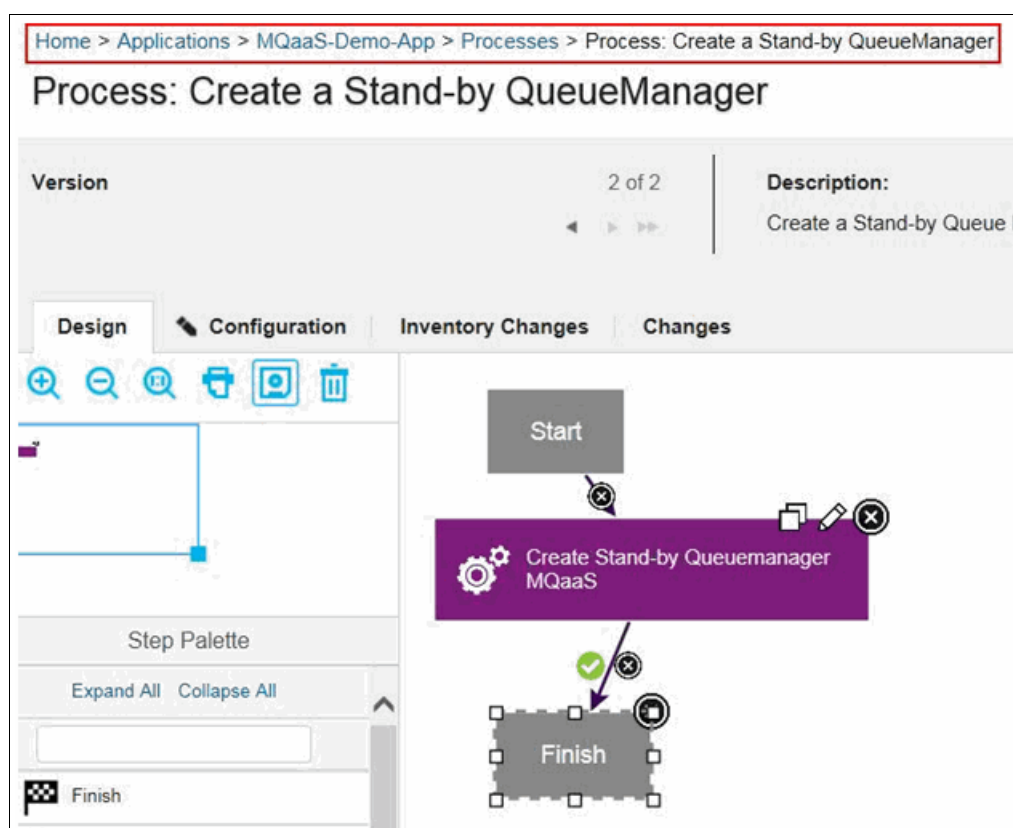


Figure 3-33 Create the standby queue manager application process

2. Update the properties, as shown in Figure 3-34 on page 51.

Edit Properties

Name * Create Stand-by QueueManager

Limit to Tag

Max # of concurrent jobs. * -1

Fail Fast ☐

Run on First Online Resource Only ☐

Precondition

Run if Components Change ☐

Save Cancel

Figure 3-34 Properties for running the standby queue manager application process

Running the service

After all the processes and services are defined, the service can now be started by using REST APIs. The REST API can be started by any automation framework that has authority in the wider PaaS environment. The section “Running the service by using REST API and JSON” on page 31 shows how to start the REST API by passing a JSON file as a parameter. The JSON file contains all the required parameters and options that must be specified for the service to run.

Example 3-8 shows another option to start the REST API by supplying the parameters directly in the URL.

Example 3-8 REST API

```
curl -k -u mqaas_user@in.ibm.com:xxxxxxx
https://udeploy.hursley.ibm.com:8443/cli/applicationProcessRequest/request -X PUT
-d {"application":"MQaaS-Demo-App","applicationProcess":"Create Active
QueueManager","environment":"vmwtpm097r.hursley.ibm.com","onlyChanged":"false","pr
operties":{"QueueManagerName":"HAUCDQMGR"}}
```

3.2.3 Single-destination instance for continuous availability

Continuous availability of the destination applies when the queue manager that hosts the destination is unavailable and applications can still connect to a different queue manager to send the messages.

When the original queue manager comes back up, the intermediate or gateway queue manager forwards the messages (also known as store and forward). This approach is applicable only for applications sending messages and not applicable for receiving messages.

IBM MQ does not support getting messages from a remote queue manager (other than the one to which it is connected). This approach can be combined with the high availability option so that the application gets both continuous and high availability of destinations.

To create the service definition for this topology, you need two different processes:

- ▶ An MQaaS component process that is created by using the IBM UrbanCode Deploy IBM MQ plug-in (for more information, see Appendix A, “IBM UrbanCode Deploy initial setup” on page 149).
- ▶ An application process that uses the newly defined MQaaS component process.

Figure 3-35 shows the complete design that deals with creating resources to achieve continuous availability of the destinations.

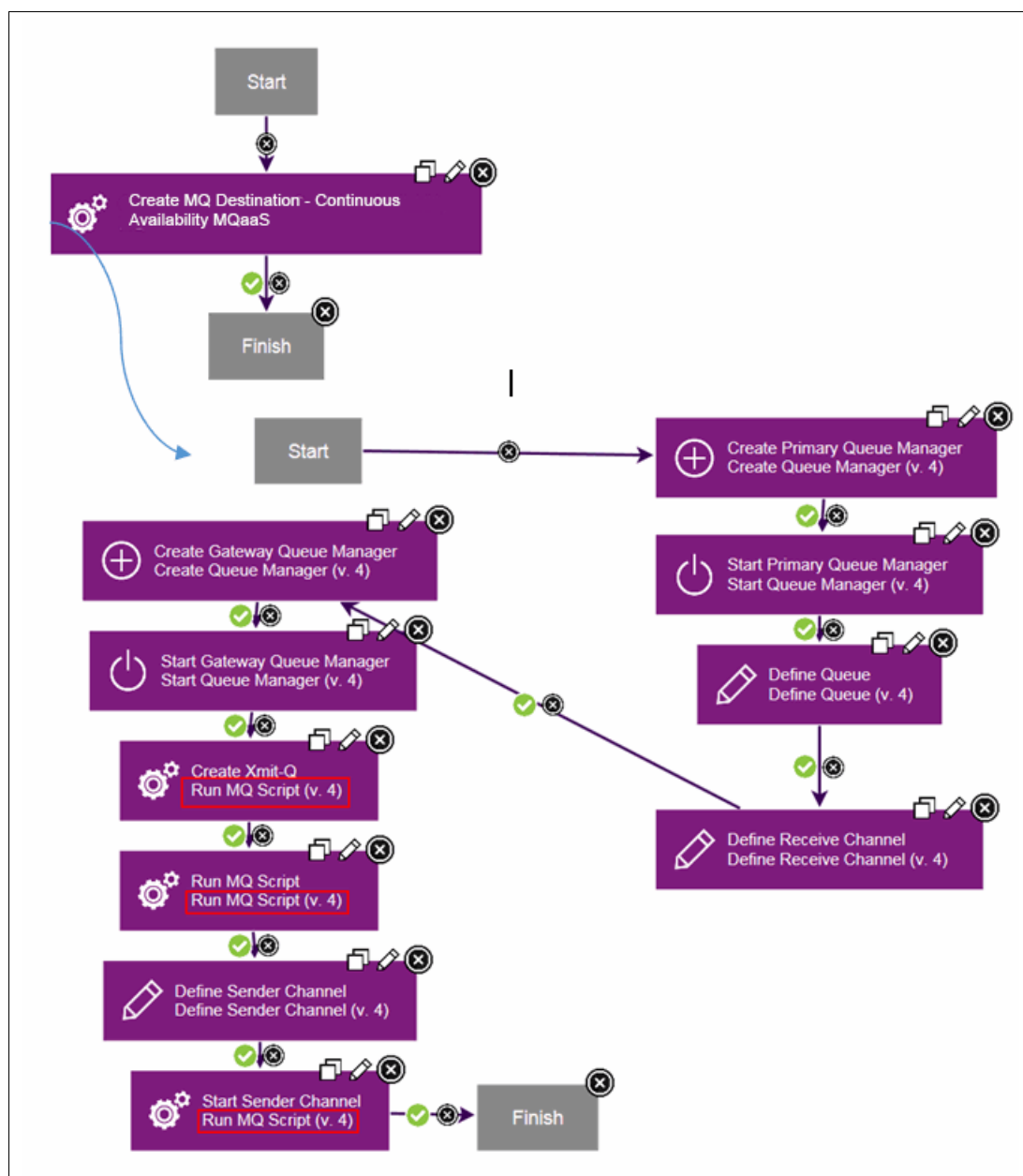


Figure 3-35 Complete design for achieving single-destination instances with continuous availability

To ensure that all the prerequisites are addressed before continuing with the following activities, see Appendix A, “IBM UrbanCode Deploy initial setup” on page 149.

Creating a process for the MQaaS component

To create the MQaaS component processes, complete the following steps:

1. From the IBM UrbanCode Deploy web UI home page, click **Components** → **MQaaS** → **Process** and create a process that is called IBM MQ Destination-Continuous Availability (Figure 3-36).

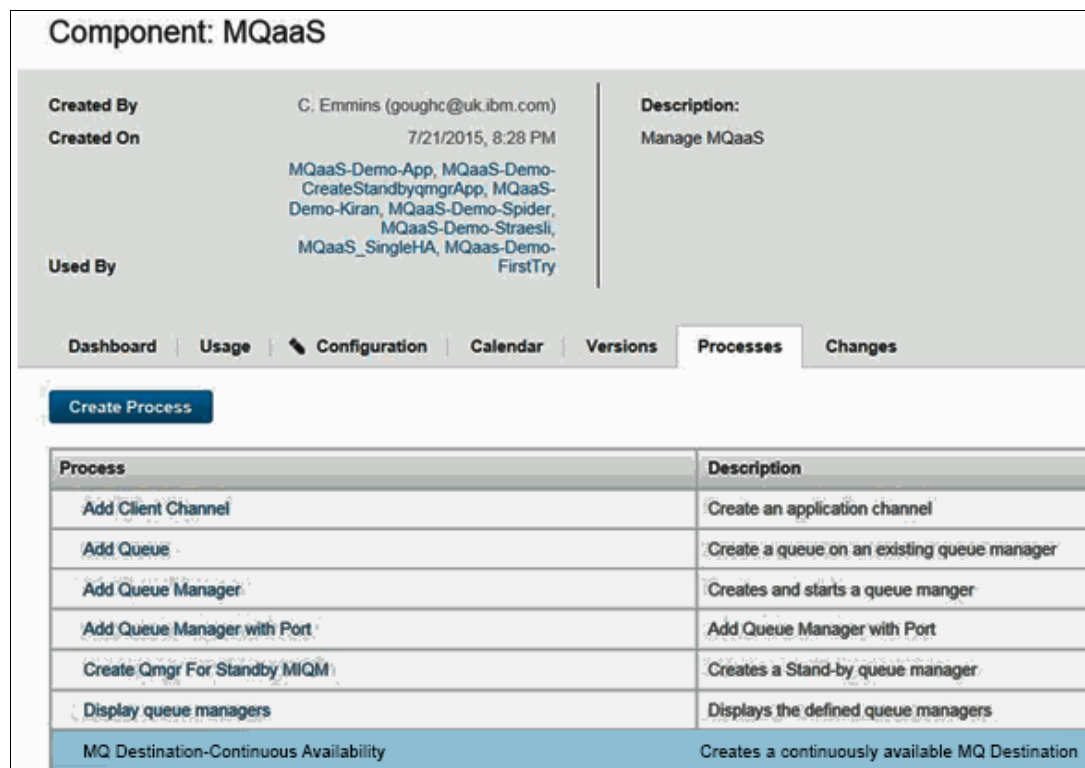


Figure 3-36 The created IBM MQ Destination Continuous Availability process

2. Select the **Configuration** tab to create the required properties. In this case, create two properties for input (Port and QueueName) (see Figure 3-37).

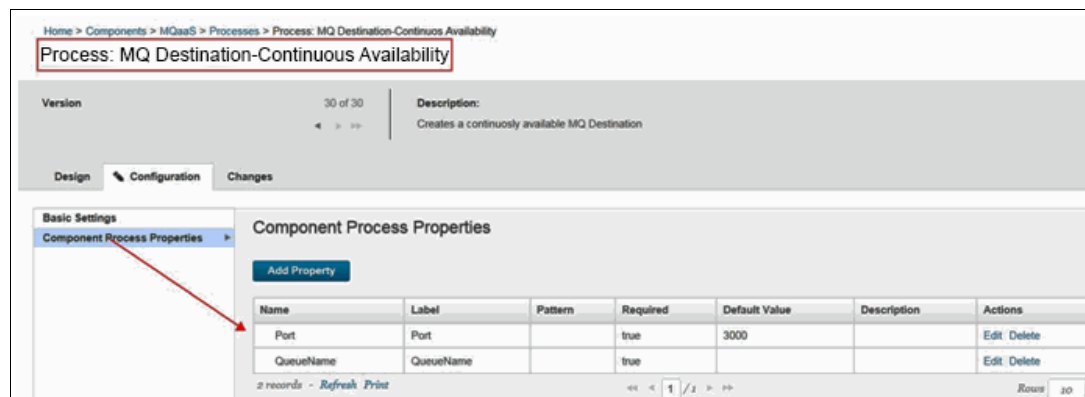


Figure 3-37 Creating the two properties for the process

To achieve the continuous availability aspects, you need two queue managers:

- ▶ A primary queue manager (where the local queue is)
- ▶ A gateway queue manager

Creating a primary queue manager where the local queue is

The detailed steps to create a queue manager and add a queue is explained in 3.2.1, “Creating a service for single-destination instance” on page 23. The main steps to create the process remain the same, but you must specify additional properties, which are explained in following steps:

1. Create the primary queue manager and provide the attributes shown in Figure 3-38. Provide the port value for the queue manager in the Additional Arguments field.

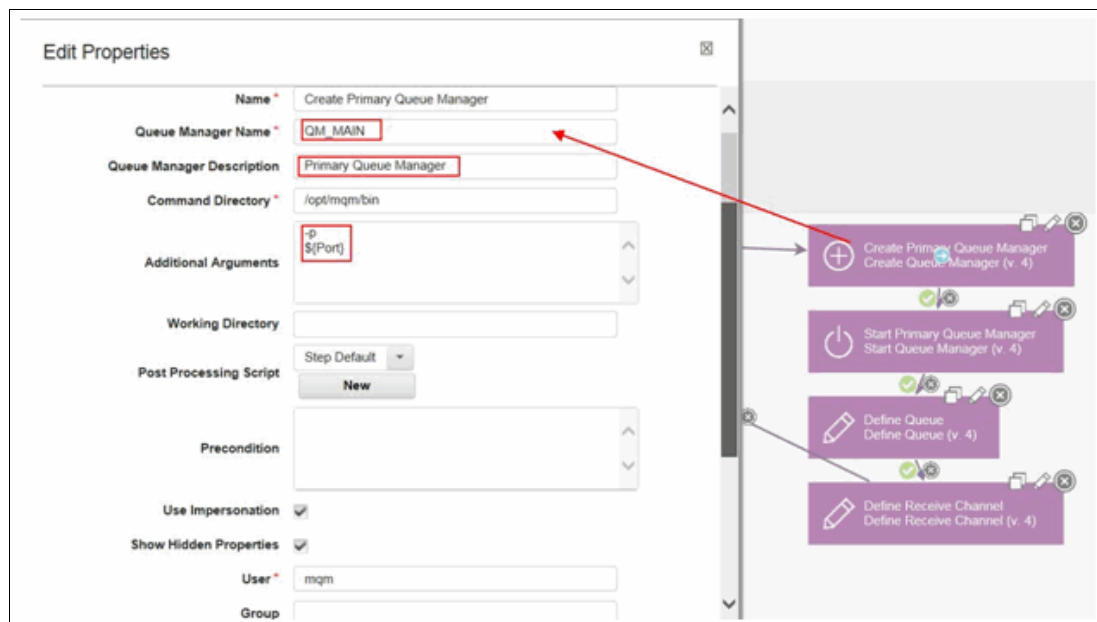


Figure 3-38 Steps to create the primary queue manager and the queue

2. Add the service to start the queue manager (Figure 3-39 on page 55). For more information about how to create the process to start the queue manager, see 3.2.2, “Making the destination highly available” on page 35.

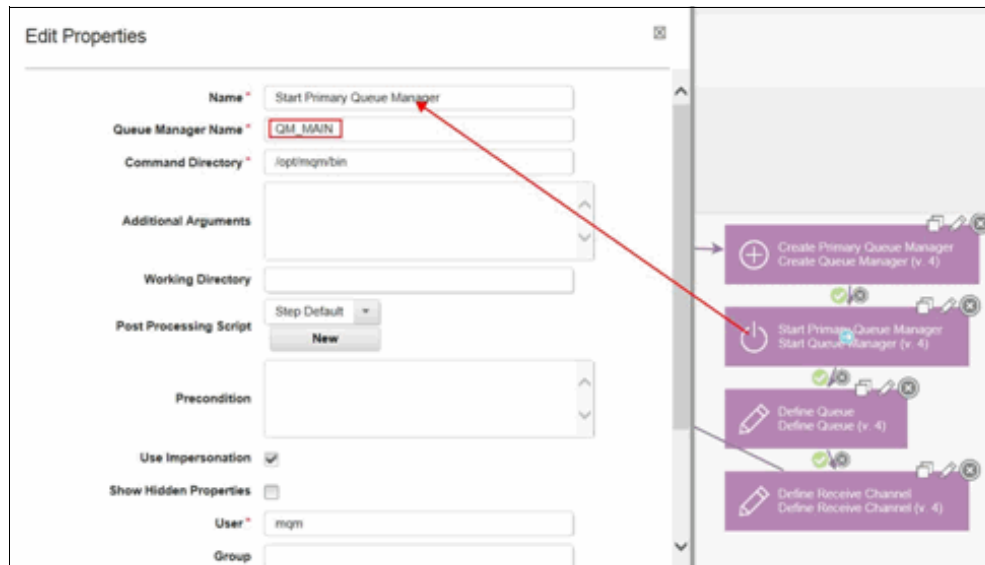


Figure 3-39 Set the properties to start the queue manager

3. Create start scripts. Add a local queue to the primary queue manager (Figure 3-40). For more information about how to create a queue, see 3.2.1, “Creating a service for single-destination instance” on page 23.

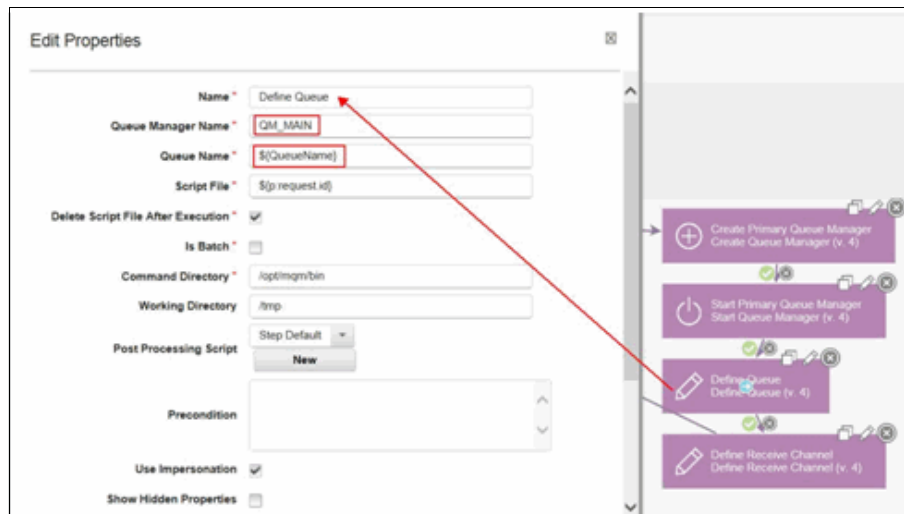


Figure 3-40 Define a queue on the primary queue manager

4. Create receiver channels.

Define the receiver channels to be paired with the sender channels that are created in the gateway queue managers:

- a. In the Step Palette, click **Middleware** → **Messaging** → **WebSphereMQ** → **Define Receiver Channel**.
- b. Drag the Define Receiver Channel resource on to the process designer palette. A dialog box opens and prompts you for details. Enter the values shown in Figure 3-41.

The image shows the 'Edit Properties' dialog for a 'Define Receiver Channel' resource. The dialog has the following fields and values:

- Name: Define Receive Channel
- Queue Manager Name: QM_MAIN
- Channel Name: QM_MAIN_RCVR
- Script File: \${p.request.id}
- Delete Script File After Execution: ☒
- Is Batch: ☒
- Command Directory: /opt/mqm/bin
- Working Directory: /tmp
- Post Processing Script: Step Default
- Precondition: (empty)
- Use Impersonation: ☒
- Show Hidden Properties: ☐

On the right side of the dialog, there is a process designer palette showing a sequence of steps: 'Create Primary Queue Manager', 'Start Primary Queue Manager', 'Define Queue', and 'Define Receiver Channel'. A red arrow points from the 'Queue Manager Name' field in the dialog to the 'Define Receiver Channel' step in the palette.

Figure 3-41 Properties for defining receiver channels

Gateway queue manager-related steps

To this point, you created the primary queue manager and its resources. This section describes the various resources that must be created for the gateway queue manager.

The detailed steps to create a queue manager are explained in 3.2.1, “Creating a service for single-destination instance” on page 23. The steps to create the process remain the same. Figure 3-42 on page 57 shows the Create Gateway Queue Manager details.

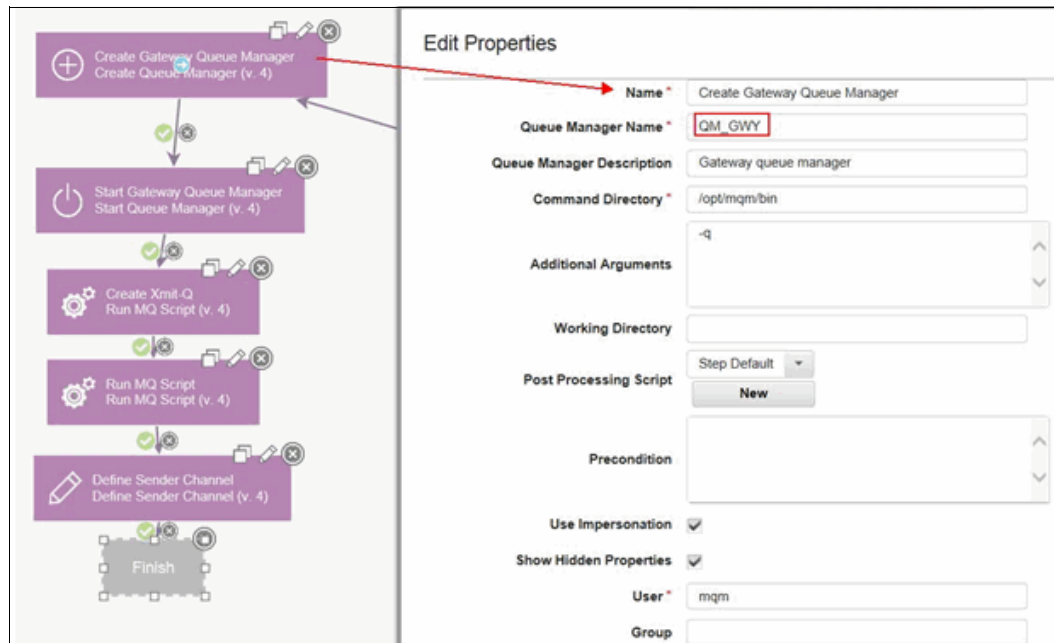


Figure 3-42 Process to configure the gateway queue manager

You must specify additional properties, as explained in the following steps:

1. With the Gateway Queue Manager resource created, add the service to start the gateway queue manager (see Figure 3-43).

For more information about how to create the process to start the queue manager, see 3.2.2, “Making the destination highly available” on page 35.

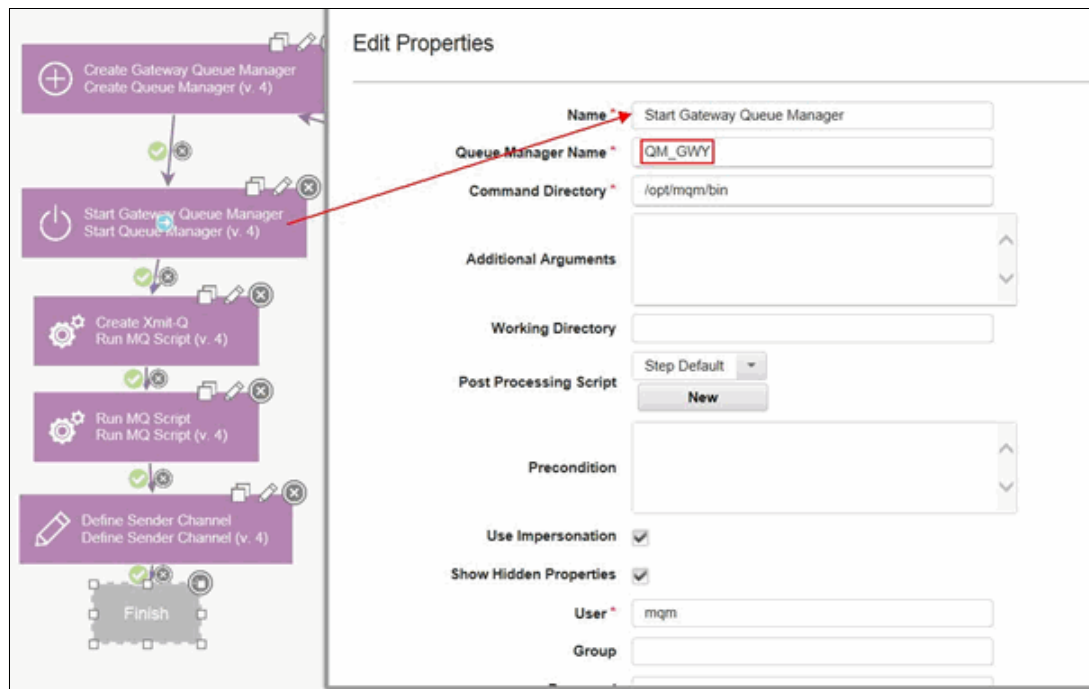


Figure 3-43 Properties to start the gateway queue manager

The gateway queue manager acts as an intermediate queue manager to store the messages that are destined for the local queue on the primary queue manager. To establish the link between the primary and gateway queue managers, you must configure the sender channel and set up the transmit queues on the gateway queue manager.

2. In the Step Palette, click **Middleware** → **Messaging** → **WebSphereMQ** → **Run MQ Script** and drag the Run MQ Script resource on to the process designer palette. Enter the details, as shown in Figure 3-44.

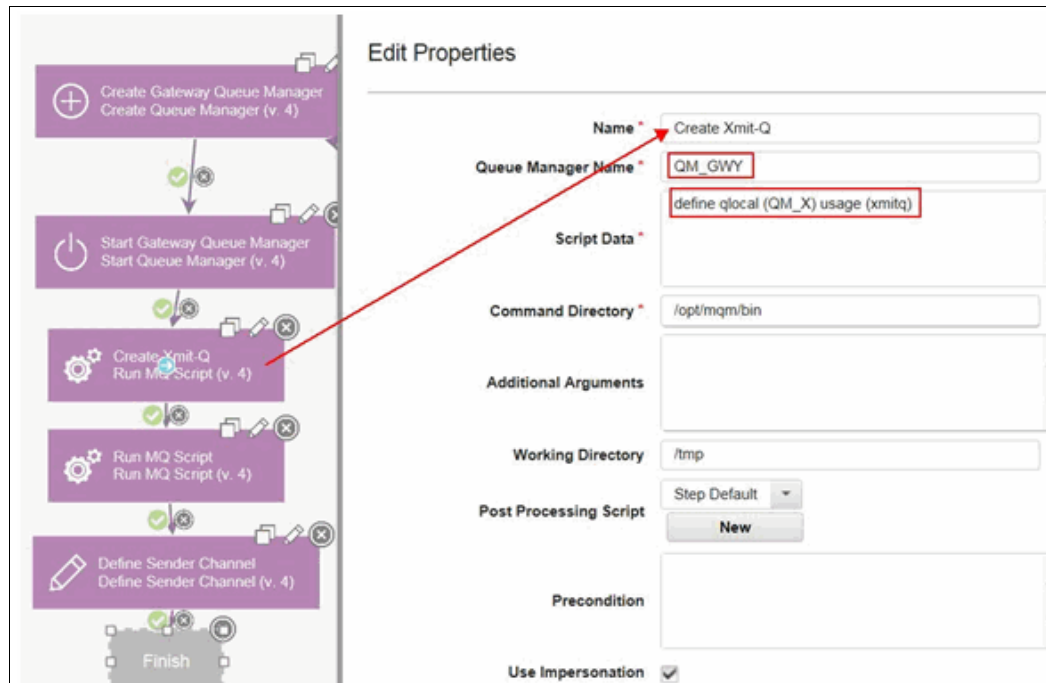


Figure 3-44 Properties for creating an IBM MQ Transmit Queue

3. The other important resource that is required to establish the link between the gateway queue manager and the primary queue manager is the IBM MQ Remote Queues.

Create the IBM MQ Remote Queues by using the Run MQ Script option from the Step Palette. Click **Middleware** → **Messaging** → **WebSphereMQ** → **Run MQ Script**. Drag the Run MQ Script resource on to the process designer palette. Enter the detail, as shown in Figure 3-45.

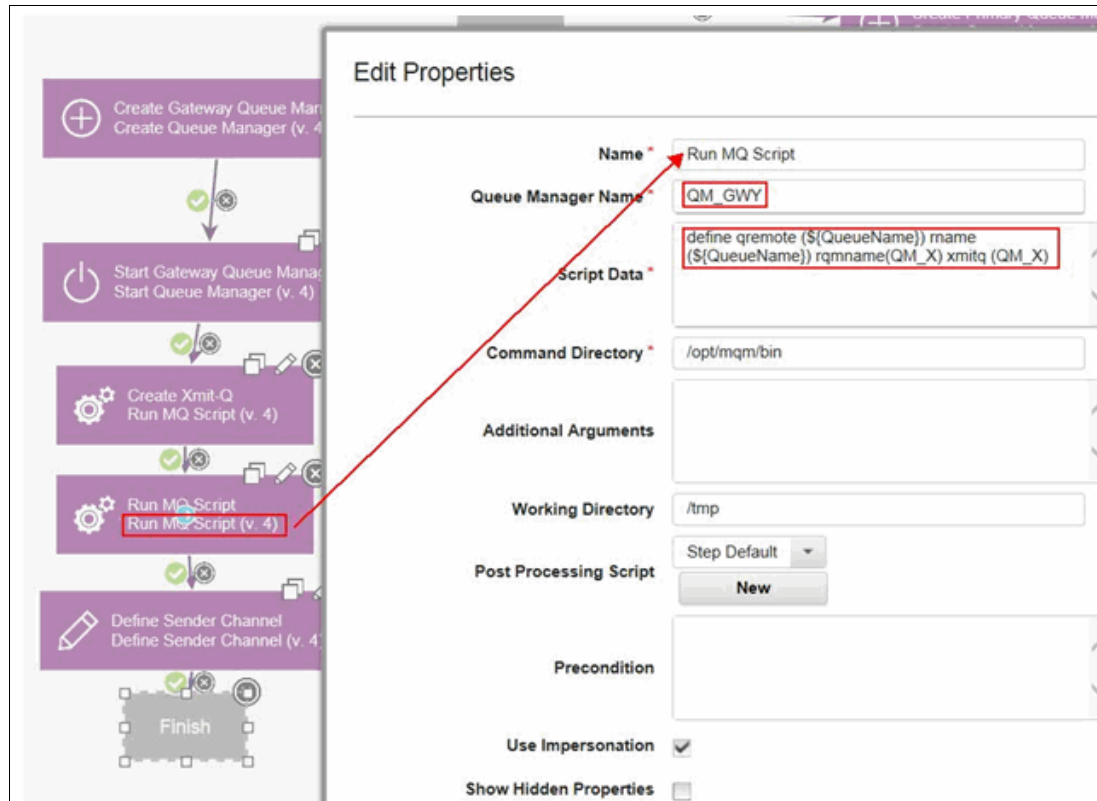


Figure 3-45 Create the remote queue on the gateway queue manager

4. Create the sender channel on the gateway queue manager that synchronizes with the receiver channel that is created on the primary queue manager.

To create a sender channel, complete the following steps:

- a. In the Step Palette, click **Middleware** → **Messaging** → **WebSphereMQ** → **Define Sender Channel**.
- b. Drag the Define Sender Channel resource on to the process designer palette. A dialog box opens and prompts for details. Enter the values shown in Figure 3-46.

Edit Properties

Name * Define Sender Channel

Queue Manager Name * QM_GWY

Channel Name * QM_GMY_SENDER

Queue Name * QM_X

Connection Name * localhost:\${Port}

Script File * \${p.request.id}

Delete Script File After Execution * ☒

Is Batch * ☐

Command Directory * /opt/mqm/bin

Working Directory /tmp

Post Processing Script Step Default
 New

Precondition

Use Impersonation ☒

Figure 3-46 Properties for creating a sender channel

5. After the sender channel is created, add the script to start the sender channel:
 - a. From the Step Palette, click **Middleware** → **Messaging** → **WebSphereMQ** → **Run MQ Script**.
 - b. Drag the Run MQ Script resource on to the process designer palette.
 - c. Enter the details, as shown in Figure 3-47 on page 61.

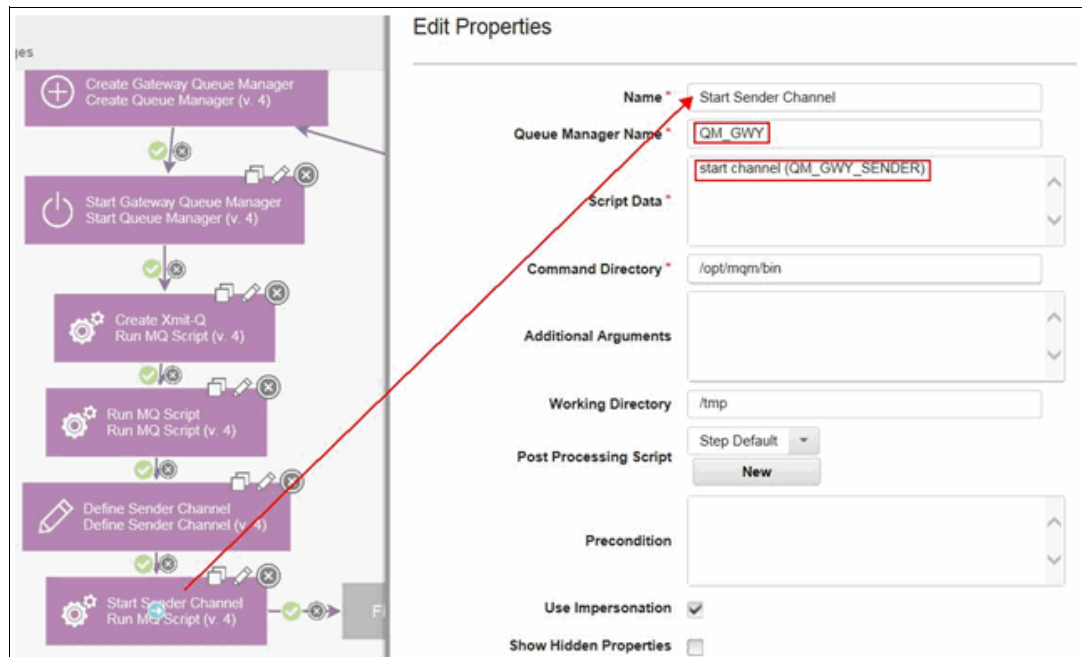


Figure 3-47 Properties for starting the sender channel.

You completed the creation of all the necessary MQaaS component processes. Now, you can create an application process that uses the component process.

Creating the application process

Create the application process by completing the following steps:

1. From the IBM UrbanCode Deploy web UI home page, click **Applications** → **MQaaS-Demo-App** → **Process** and create a process that is called Create an MQ Destination-Continuous Availability (Figure 3-48).

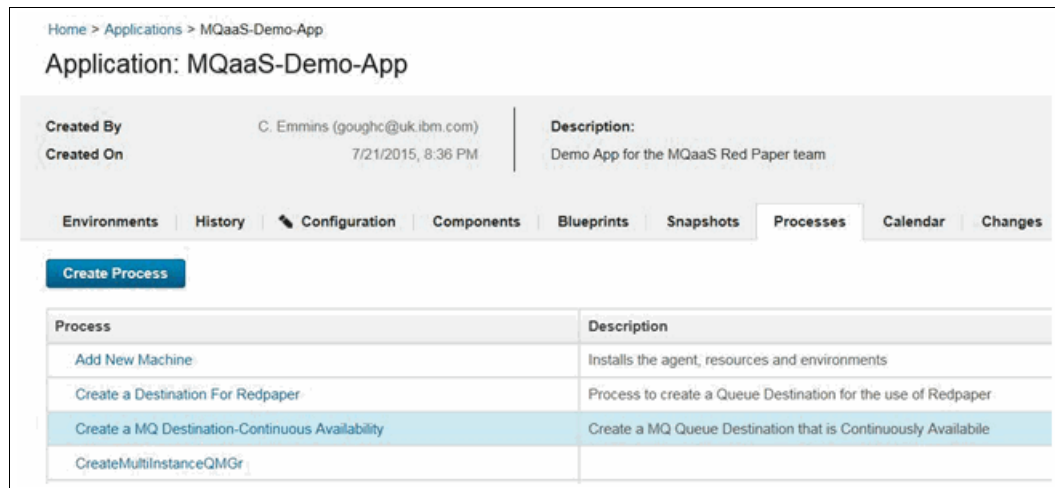


Figure 3-48 Shows the newly created application process

2. Click the process **Create an MQ Destination-Continuous Availability**.
Drag the MQ Destination-Continuous Availability resource on to the process designer palette (Figure 3-49).

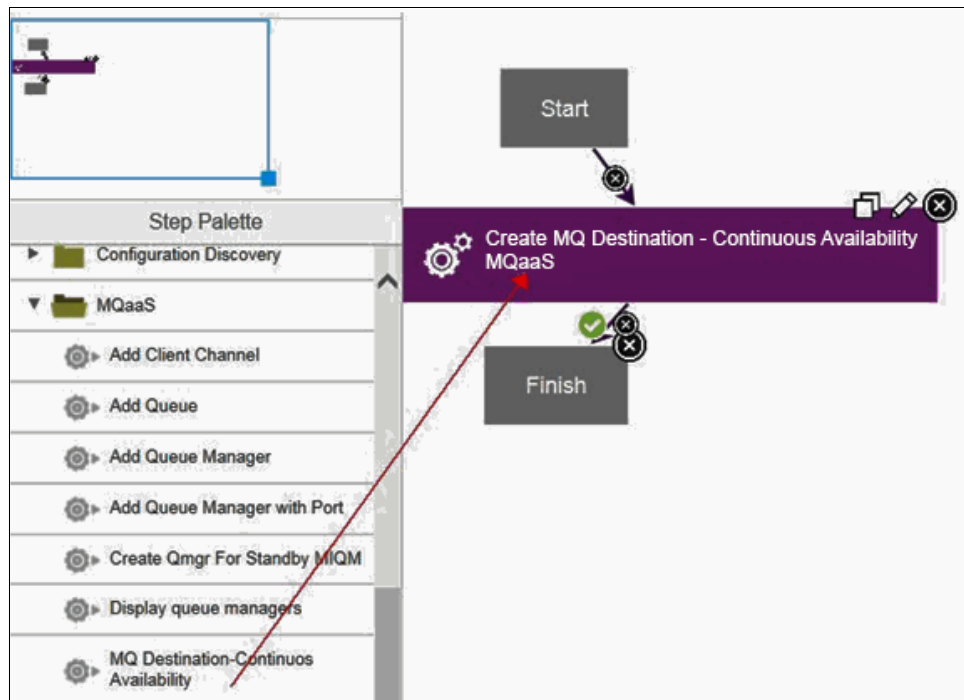


Figure 3-49 Define an application process

3. Enter the properties for Create MQ Destination - Continuous Availability, as shown in Figure 3-50.

Figure 3-50 Properties to set for creating an IBM MQ destination process

Associating an environment for the application

The next activity is to associate an environment for the application. Complete the following steps:

1. Click **Application** → **Environments** and find your preferred environment:
2. Click the play button to open a wizard.
3. Complete the information for Create an MQ Destination-Continuous Availability process, as shown in Figure 3-51.
4. Click **Submit**.

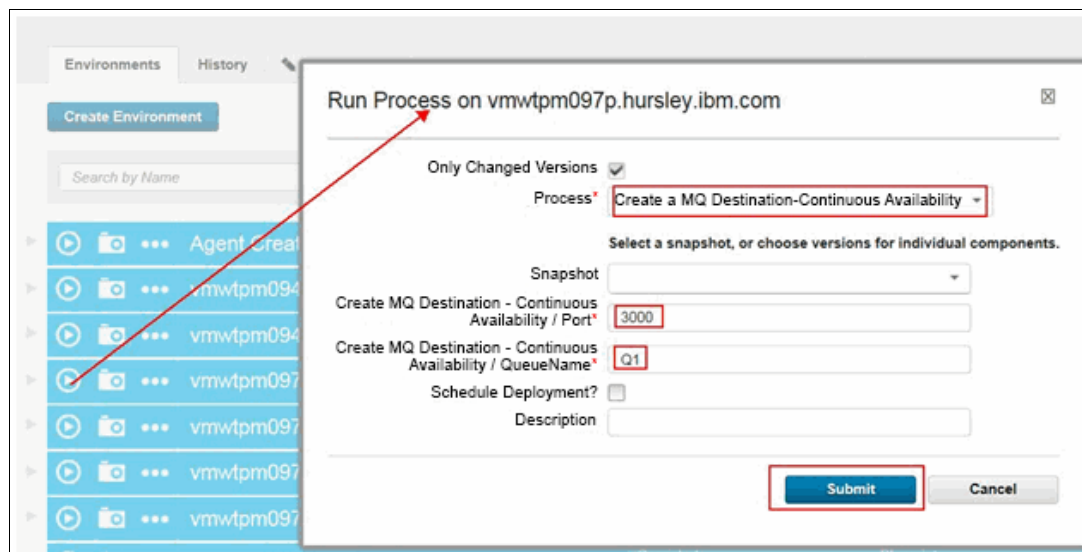


Figure 3-51 Run Process - Create MQ Destination-Continuous Availability

Starting the service

Users can start this service by running the REST command that is shown in Example 3-9.

Example 3-9 Start the Create an MQ Destination-Continuous Availability service

```
curl -k -u mqaas_user@in.ibm.com:mypassword
https://udeploy.hursley.ibm.com:8443/cli/applicationProcessRequest/request -X PUT
-d {"application":"MQaaS-Demo-App","applicationProcess":"Create an MQ
Destination-Continuous
Availability","environment":"vmwtpm097p.hursley.ibm.com","onlyChanged":"false","pr
operties":{"Port":"3000","QueueName":"Q1"}}
```

Single-destination continuous availability topology considerations

This section includes considerations that apply to the single-destination continuous availability topology.

Benefits of the single-destination continuous availability topology

The single-destination continuous availability topology helps in the following ways:

- ▶ Applications can tolerate any downtime for sending messaging by using the continuous availability of the destinations.
- ▶ Applications can tolerate minimal downtime (maybe seconds or few minutes) for receiving messages.

Single-destination continuous availability topology considerations for developers

Table 3-3 identifies the advantages and disadvantages of using the single destination with continuous availability topology for the development and application teams.

Table 3-3 *Single-destination continuous availability topology - considerations for developers*

Advantages	Disadvantages
<ul style="list-style-type: none">► Provides continuous availability for applications.► Applications can leverage the high availability and failover capabilities while achieving continuous availability.► Applications can have the flexibility to either connect directly to the queue manager that hosts the destination or connect to gateway queue managers.	<ul style="list-style-type: none">► Applications must have the logic to reconnect to different queue managers, either by using ConnectionNameList or a Client Channel Definition Table (CCDT).► eXtended Architecture (XA) transactions can be a challenge, especially during recovery.► Performance can be a bottleneck with the message being persisted across multiple queue managers (a source or gateway queue manager that holds the message temporarily, and target queue manager where the message finally is before being consumed).► Does not provide scalability for applications.

Single destination with continuous availability topology considerations for the infrastructure team

Table 3-4 identifies the advantages and disadvantages of using a single destination with a continuous availability service for infrastructure teams.

Table 3-4 *Using a single destination with a continuous availability service for infrastructure teams*

Advantages	Disadvantages
<ul style="list-style-type: none">► Provides flexibility to the infrastructure team to create a hub of queue managers that they can add as a gateway.► Queue managers can be added and removed dynamically from this gateway.► Helps in shielding the applications from connecting directly to the queue manager that hosts the local destination.► Can use built-in IBM MQ clustering capabilities for ease of administration and configuration.	<ul style="list-style-type: none">► Serviceability can be a challenge with the message being persisted across multiple queue managers.► Performance impact with the message being stored and forwarded between queue managers.► Need to ensure each of the gateway queue manager is made highly available. If not, the messages can be stranded on the gateway queue manager if the gateway queue manager goes down.

Figure 3-52 on page 65 shows a topology for continuous availability with producer applications connecting to gateway queue managers to send messages. Remote queues hold the messages and forward them to the local queue.

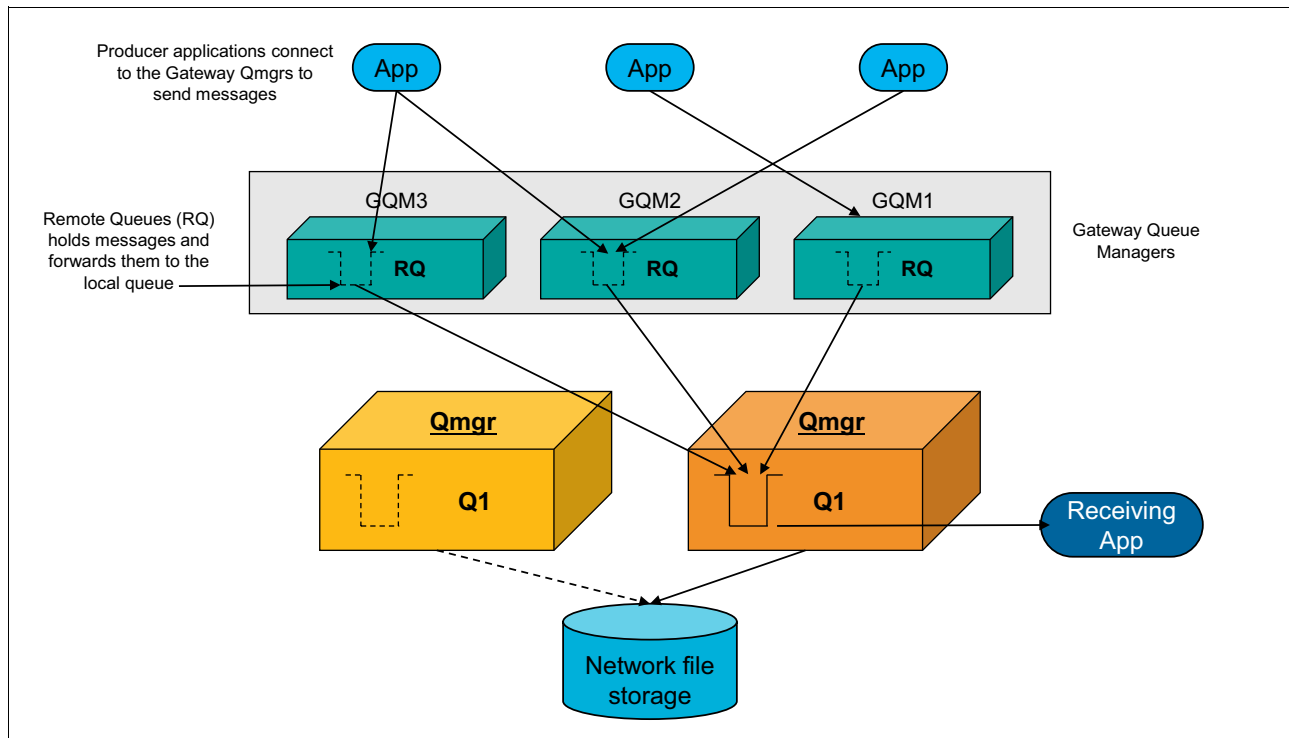


Figure 3-52 Applications connecting to gateway queue managers for continuous availability

3.2.4 Multiple-destination instances

Multiple-destination instances are necessary to support application scalability and continuous availability. In a multiple-destination scenario, there can be more than one instance of the same destination existing on multiple queue managers. Applications have the flexibility to send and receive messages from any destination.

The usage of multiple destinations enables the infrastructure team to distribute geographically the destinations. This approach enables the applications to *look the same* regardless of the queue manager instance they connect to in an IBM MQ infrastructure.

Conversely, if a particular service is down locally, the application can use a remote destination instance (even when it might be a slower than usual). Each of the destinations can also be made highly available.

All the steps up to creating an application environment remain the same for this scenario. For more information, see Appendix A, “IBM UrbanCode Deploy initial setup” on page 149.

To set up a multiple-destination scenario, two processes are needed:

1. One process as an MQaaS component process that is defined by using the IBM UrbanCode Deploy IBM MQ plug-in.
2. One process as an application process that uses the newly defined MQaaS component process.

Figure 3-53 shows the high-level design for setting up the multiple-destination instances.

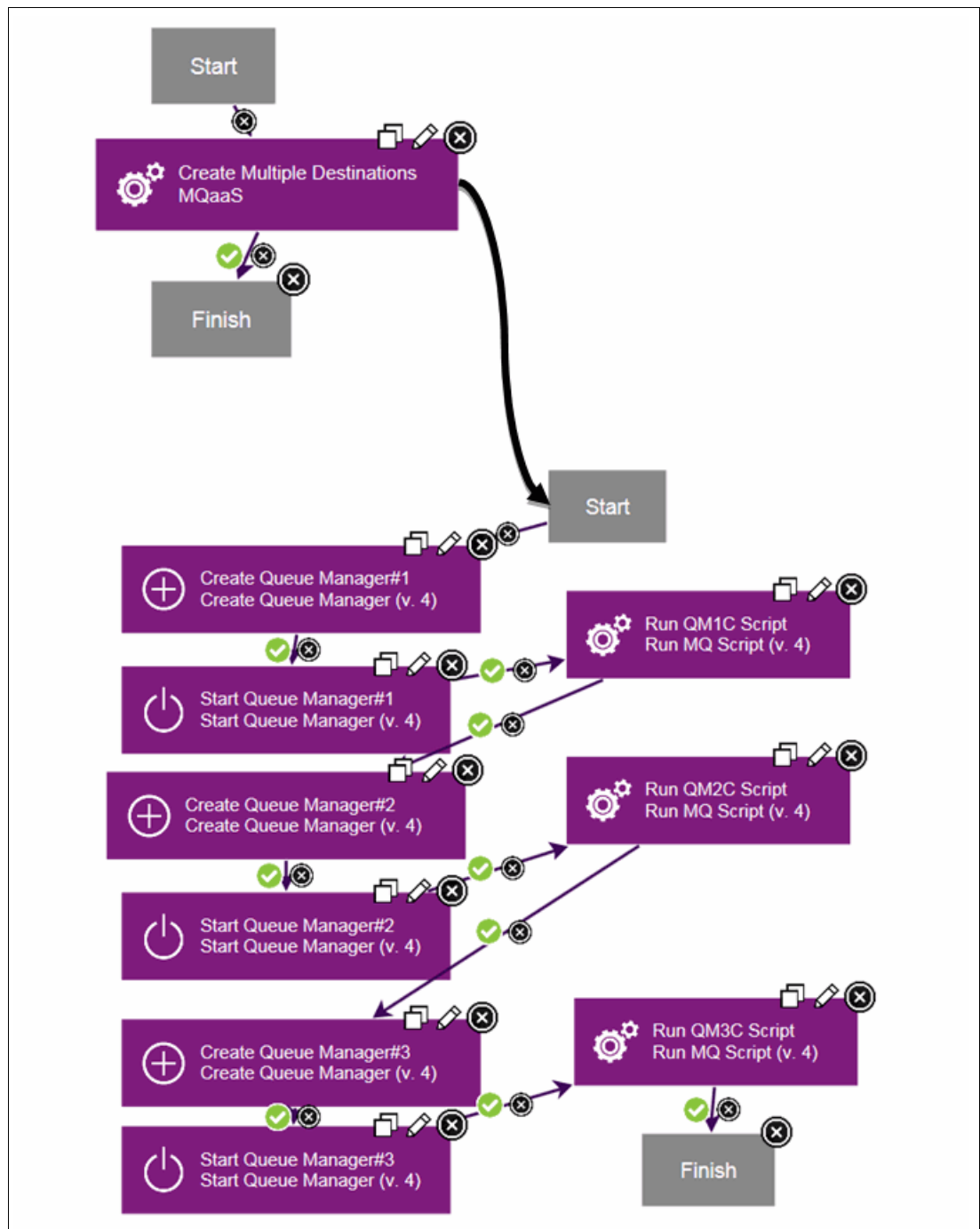


Figure 3-53 High-level design for setting up the multiple-destination instances

Creating an MQaaS component process for multiple-destination instances

The following steps define the new component process that includes all the artifacts that are required to create and run multiple-destination instances:

1. From the IBM UrbanCode Deploy web UI home page, click **Components** → **MQaaS** → **Process** and create a process that is named Create Multiple MQ Destination-Cluster Setup (Figure 3-54). For more information, see Appendix A, “IBM UrbanCode Deploy initial setup” on page 149.

The screenshot shows the 'Create Process' dialog in the IBM UrbanCode Deploy web UI. The dialog is titled 'Create Process' and has a close button (X) in the top right corner. It contains the following fields:

- Name**: Create Multile MQDestination-Cluster Setup
- Description**: Creates a Cluster setup
- Process Type**: Operational (No Version Needed) (dropdown menu)
- Default Working Directory**: \${p.resource/work.dir}/\${p.component.name}
- Required Role**: None (dropdown menu)

At the bottom right, there are two buttons: 'Save' (highlighted with a red box) and 'Cancel'.

Figure 3-54 Create an MQaaS component process

2. Define additional properties for the process. Select the process **Configuration** tab and add new properties for input, as shown in Figure 3-55.

The screenshot shows the 'Configuration' tab for the process 'Create Multiple MQDestination-Cluster Setup'. The 'Component Process Properties' table is visible, listing properties:

Name	Label	Pattern	Required	Default Value	Description
Port_QM1C	Port_QM1C		true	3001	Port for QM1 in Cluster
Port_QM2C	Port_QM2C		true	3002	Port for QM2 in Cluster
Port_QM3C	Port_QM3		true	3003	Port for QM3 in Cluster
CLUSQ	CLUSQ		false	CLUSQ	Cluster Queue

The 'Port_QM1C', 'Port_QM2C', and 'CLUSQ' rows are highlighted with red boxes.

Figure 3-55 Add component process input properties

The properties are as follows:

- Port_QM1C: Port number that is used for Cluster queue manager #1
- Port_QM2C: Port number that is used for Cluster queue manager #2
- Port_QM3C: Port number that is used for Cluster queue manager #3
- CLUSQ: Name of the Cluster queue to be defined

- Click the Create Multiple MQDestination-Cluster Setup process and open the process designer palette to add scripts and resources.
- Create the first queue manager (QM1C). For more information about creating the queue manager, see 3.2.1, “Creating a service for single-destination instance” on page 23. Enter the attributes, as shown in Figure 3-56.

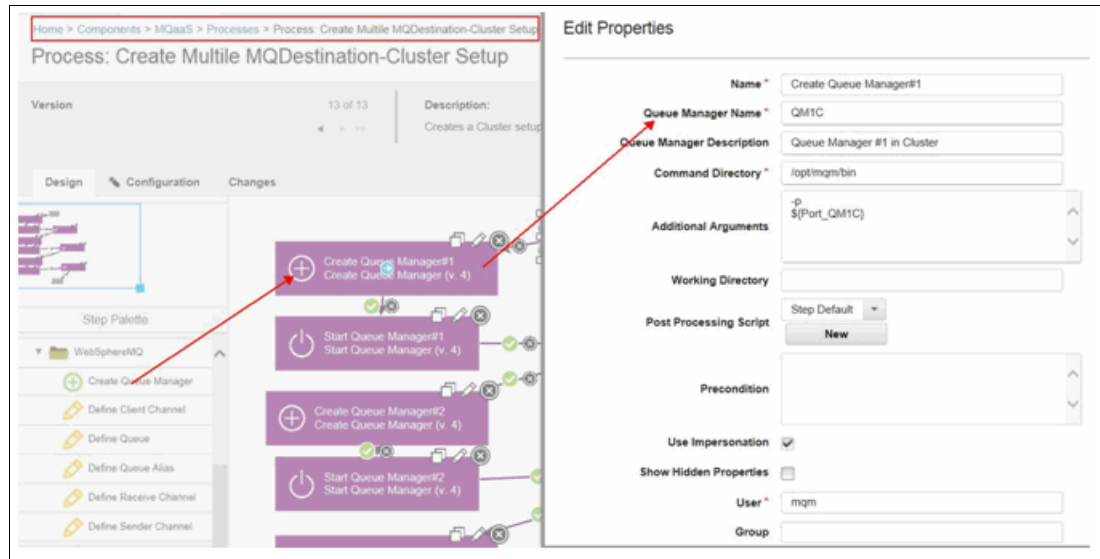


Figure 3-56 Properties for creating QM1C

- After the queue manager resource is added, add the resource to start QM1C. For more information about how to start the queue manager, see 3.2.1, “Creating a service for single-destination instance” on page 23. Enter the attributes, as shown in Figure 3-57.

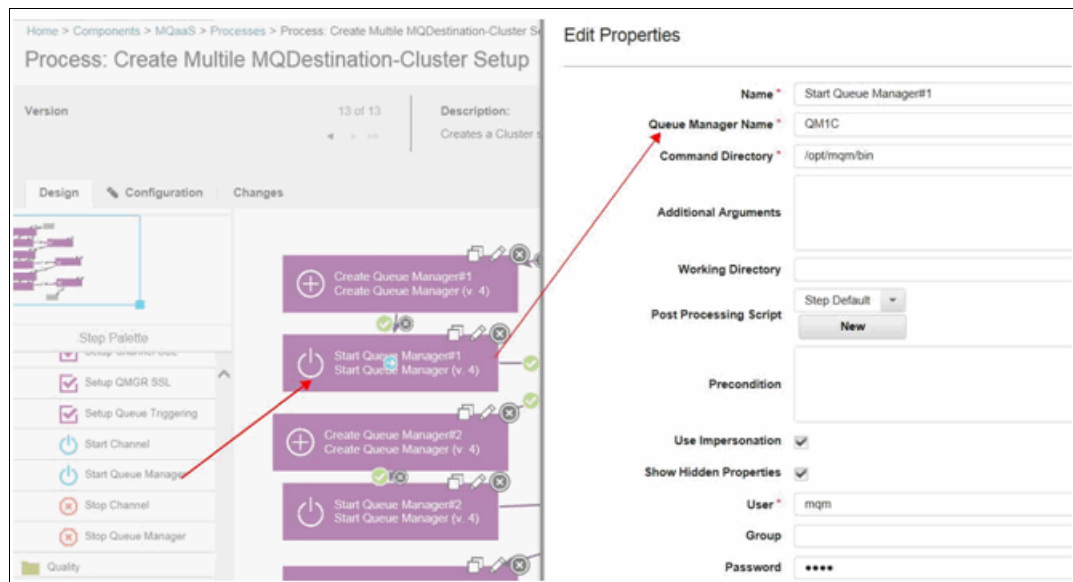


Figure 3-57 Properties for starting the QM1C queue manager

6. After the queue manager scripts are created, define the queues and channels that are required for the primary queue manager (QM1C). From the Step Palette, click **Middleware** → **Messaging** → **WebSphereMQ** → **Run MQ Script**.

Drag the Run MQ Script resource on to the process designer palette.

Enter the details into the Run MQ Script properties window, as shown in Figure 3-58. Example 3-10 shows the script.

Example 3-10 Script to define QLOCAL and CHANNEL for the queue manager

```
ALTER QMGR REPOS(MYCLUSTER)
DEFINE QLOCAL (LCQ) CLUSTER(MYCLUSTER)
DEFINE QLOCAL (${CLUSQ}) CLUSTER(MYCLUSTER) CLWLUSEQ(ANY)
DEFINE CHANNEL(MYCLUSTER.QM2C) CHLTYPE(CLUSSDR)
CONNAME('localhost(${Port_QM2C})') CLUSTER(MYCLUSTER)
DEFINE CHANNEL(MYCLUSTER.QM1C) CHLTYPE(CLUSRCVR)
CONNAME('localhost(${Port_QM1C})') CLUSTER(MYCLUSTER)
```

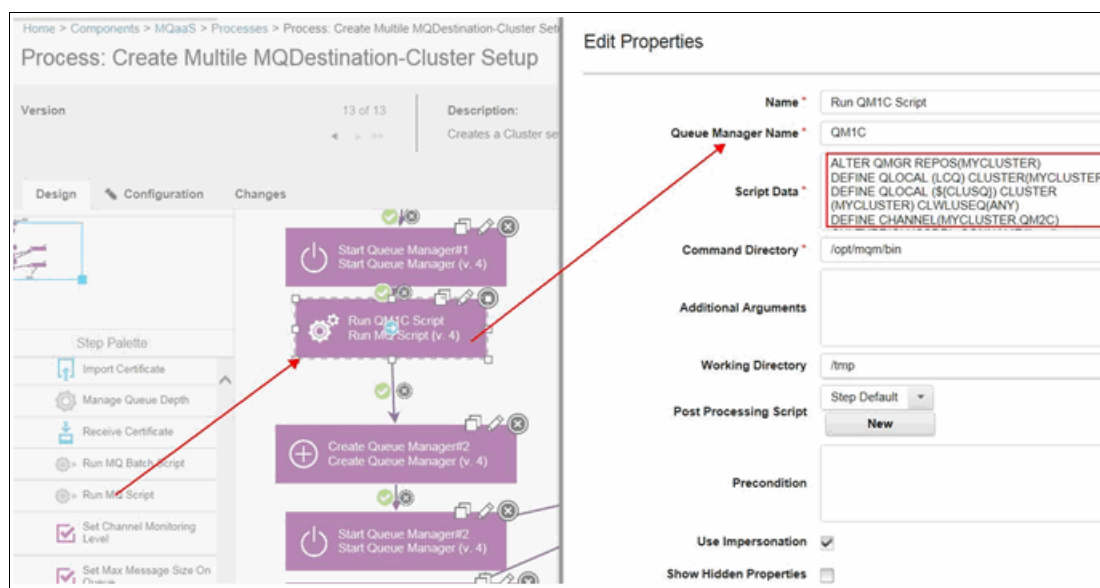


Figure 3-58 Add IBM MQ scripts to create queue, channel, and cluster information

Note: Repeat these steps for as many queue managers as needed. This example uses three queue managers (QM1C, QM2C, and QM3C).

Example 3-11 shows the scripts that are used for each queue manager.

Example 3-11 Script for each queue manager (QM1C, QM2C, and QM3C)

Qmgr#1

```
Name = QM1C
Port = 3001
Cluster Queue = CLUSQ
Local Queue = LCQ
MQSC Script
ALTER QMGR REPOS(MYCLUSTER)
DEFINE QLOCAL (LCQ) CLUSTER(MYCLUSTER)
DEFINE QLOCAL (${CLUSQ}) CLUSTER(MYCLUSTER) CLWLUSEQ(ANY)
DEFINE CHANNEL(MYCLUSTER.QM2C) CHLTYPE(CLUSSDR)
CONNNAME('localhost(${Port_QM2C})') CLUSTER(MYCLUSTER)
DEFINE CHANNEL(MYCLUSTER.QM1C) CHLTYPE(CLUSRCVR)
CONNNAME('localhost(${Port_QM1C})') CLUSTER(MYCLUSTER)
```

Qmgr#2

```
Name = QM2C
Port = 3002
Cluster Queue = CLUSQ
MQSC Script
ALTER QMGR REPOS(MYCLUSTER)
DEFINE QLOCAL(${CLUSQ}) CLUSTER(MYCLUSTER) CLWLUSEQ(ANY)
DEFINE CHANNEL(MYCLUSTER.QM1C) CHLTYPE(CLUSSDR)
CONNNAME('localhost(${Port_QM1C})') CLUSTER(MYCLUSTER)
DEFINE CHANNEL(MYCLUSTER.QM2C) CHLTYPE(CLUSRCVR)
CONNNAME('localhost(${Port_QM2C})') CLUSTER(MYCLUSTER)
```

Qmgr#3

```
Name = QM3C
Port = 3003
Cluster Queue = CLUSQ
MQSC Script
ALTER QMGR REPOS(MYCLUSTER)
DEFINE QLOCAL(${CLUSQ}) CLUSTER(MYCLUSTER) CLWLUSEQ(ANY)
DEFINE CHANNEL(MYCLUSTER.QM1C) CHLTYPE(CLUSSDR)
CONNNAME('localhost(${Port_QM1C})') CLUSTER(MYCLUSTER)
DEFINE CHANNEL(MYCLUSTER.QM2C) CHLTYPE(CLUSRCVR)
CONNNAME('localhost(${Port_QM2C})') CLUSTER(MYCLUSTER)
```

Note: All the values `${property_name}` are replaced with the actual value at run time. The input for these properties is provided by the user when the application process runs.

With these steps, you have create the MQaaS component process.

Creating the application process to associate with the Create Multiple MQDestination-Cluster Setup process

To create the application process that is associated with the Create Multiple MQDestination-Cluster Setup process, complete the following steps.

1. From the IBM UrbanCode Deploy web UI Home window, click **Applications** → **MQaaS-Demo-App** → **Process**. Create a process that is named Multiple Destinations - ClusterSetup, as shown in Figure 3-59 on page 71.

Create an Application Process

Name * Multiple Destinations - ClusterSetup

Description te Multiple destinations instances using cluster setup

Inventory Management * Automatic

Offline Agent Handling * Check Before Execution

Required Role

Save **Cancel**

Figure 3-59 Properties for creating an application process

2. Associate the component process with this application. From the Step Palette, under the MQaaS folder, find Create Multiple MQ Destination-Cluster Setup and drag it on to the process designer palette. Set the configuration as shown in Figure 3-60.

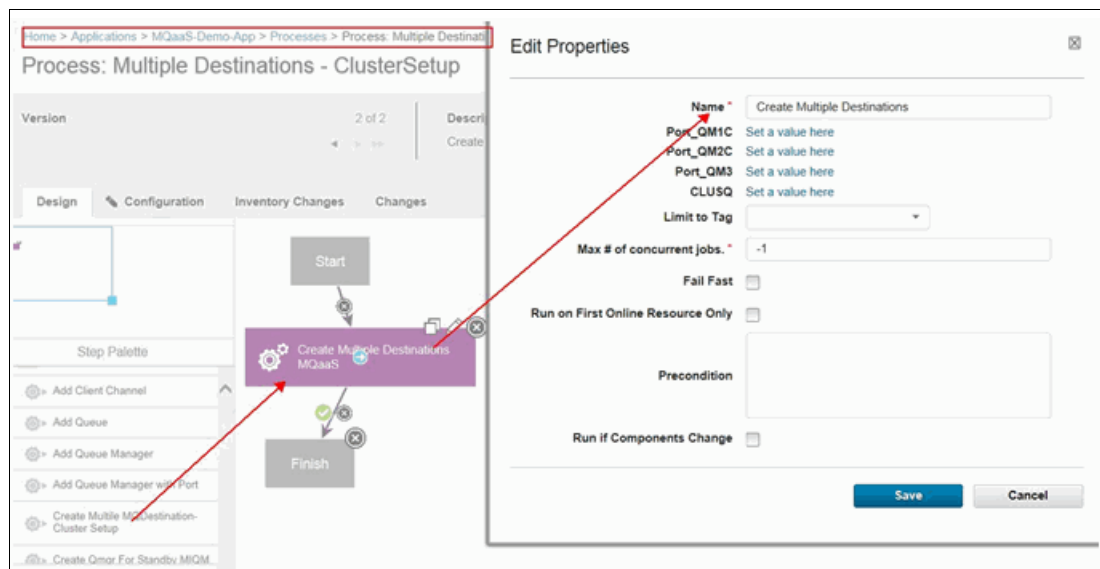


Figure 3-60 Associate the Create Multiple MQ Destination-Cluster Setup process with the application process

Verifying the process

To verify that the process definition is correct, you can run the design from the IBM UrbanCode Deploy user interface by completing the following steps:

1. Click **Application** → **Environments**, find your preferred environment, and click the play button.
2. This action opens a wizard. Complete the fields, as shown in Figure 3-61.
3. Click **Submit**.

Home > Applications > MQaaS-Demo-App

Application: MQaaS-Demo-App

Created By: C. Emmins (goughc@uk.ibm.com) | Description: Demo App for the MQaaS Red Paper team

Created On: 7/21/2015, 8:36 PM

Environments | History | Configuration

Create Environment

Search by Name

Agent Creator

vmwtpm0941.hursley.ibm.com

vmwtpm094x.hursley.ibm.com

vmwtpm097r.hursley.ibm.com

vmwtpm097p.hursley.ibm.com

vmwtpm097w.hursley.ibm.com

vmwtpm097y.hursley.ibm.com

vmwtpm098f.hursley.ibm.com

Run Process on vmwtpm097p.hursley.ibm.com

Only Changed Versions ☒

Process * Multiple Destinations - ClusterSetup

Select a snapshot, or choose versions for individual components.

Snapshot

Create Multiple Destinations / Port_QM1C * 3001

Create Multiple Destinations / Port_QM2C * 3002

Create Multiple Destinations / Port_QM3C * 3003

Create Multiple Destinations / CLUSQ CLUSQ

Schedule Deployment? ☐

Description

Submit Cancel

Figure 3-61 Run process Multiple Destinations - ClusterSetup

Starting the service through a REST URL

Example 3-12 shows the command to start the service by using a REST URL.

Example 3-12 Command for Multiple Destinations - ClusterSetup

```
curl -k -u mqaas_user@in.ibm.com:mypassword
https://udeploy.hursley.ibm.com:8443/cli/applicationProcessRequest/request -X PUT
-d {"application":"MQaaS-Demo-App","applicationProcess":"Multiple Destinations -
ClusterSetup","environment":"vmwtpm097p.hursley.ibm.com","onlyChanged":"false","pr
operties":{"Port_QM1C":"3001","Port_QM2C":"3002","Port_QM3C":"3003","CLUSQ":"CLUSQ
"}}
```


Multiple-destination topology considerations

This section includes considerations that apply to the multiple-destination topology.

Benefits of a multiple-destination topology

A multiple-destination topology helps in the following ways:

- ▶ Multiple applications can connect to a destination to send a many messages in quick succession.
- ▶ It helps address the performance requirements of the business.
- ▶ It caters to high message inflow during specific times (such as holidays, festivals, and Black Friday).

Multiple-destination topology considerations for developers

Table 3-5 shows the advantages and disadvantages of using the multiple-destination topology for developers.

Table 3-5 Multiple-destination topologies for development advantages and disadvantages

Advantages	Disadvantages
<ul style="list-style-type: none">▶ Provides continuous availability for applications.▶ Addresses the scalability needs of the applications. Provides better throughput and performance benefits for applications to send and consume messages.▶ Applications can also use the high availability and failover capabilities while achieving continuous availability and scalability.▶ Applications can have the flexibility to either connect directly to the queue manager (for sending or receiving) that hosts the destination or connect to gateway queue managers (only for sending messages).	<ul style="list-style-type: none">▶ Applications must have the logic to reconnect to different queue managers (either by using ConnectionNameList or CCDT).▶ Ordered delivery of messages is not a possibility.▶ XA transactions can be a challenge, especially during recovery.▶ If listeners (or consumers) connect to a destination that is unavailable, messages tend to pile up on this specific destination. This situation can be addressed by using AMQSCLN, as described in 3.2.5, “Multiple-destination instances: No stranded or marooned messages” on page 74.

Multiple-destination topology considerations for infrastructure teams

Table 3-6 shows the advantages and disadvantages of using the multiple-destination topology for infrastructure teams.

Table 3-6 Multiple-destination topology advantages and disadvantages for infrastructure teams

Advantages	Disadvantages
<ul style="list-style-type: none">▶ Can leverage the built-in IBM MQ clustering capabilities for automatic workload balancing of messages across multiple-destination instances.▶ Queue managers can be dynamically added or removed from the cluster to meet the application demands.▶ Allows configurations to automatically reroute messages across multiple-destination instances.▶ Ease of administration with built-in clustering networking between destinations.	<ul style="list-style-type: none">▶ Unless carefully planned, messages are persisted across multiple queue managers.▶ Must set up high availability for each queue manager to ensure failover; otherwise, messages can get stranded until the queue manager is restored.

Multiple-destination topologies examples

This section shows examples of multiple-destinations topologies.

Figure 3-62 shows applications connecting to clustered queue managers.

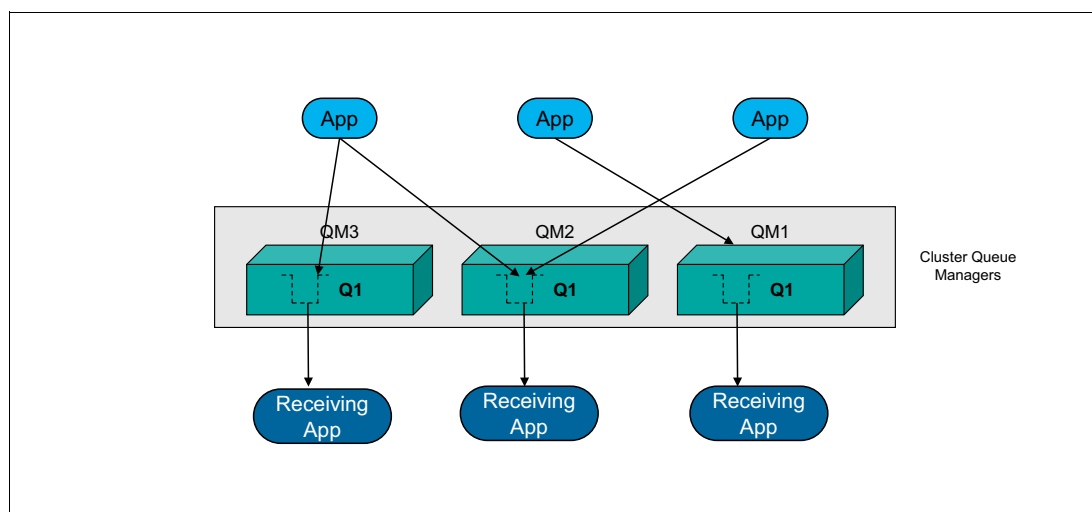


Figure 3-62 Applications connecting to clustered queues

Figure 3-63 shows applications connecting to clustered queue managers with each queue manager enabled for high availability.

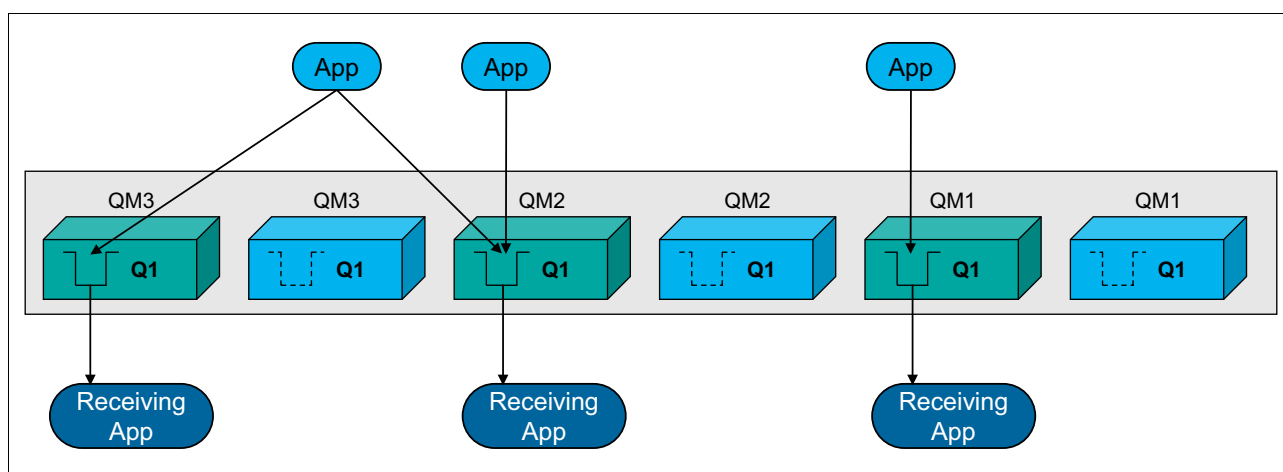


Figure 3-63 Applications connecting clustered queues with each queue manager enabled for high availability

3.2.5 Multiple-destination instances: No stranded or marooned messages

One of the key challenges with multiple-destination environments is that messages can get stranded unless you plan carefully. There are two reasons why the messages can be stranded at a destination:

- ▶ There is no highly available destination that is configured.
- ▶ There are no active listeners at a particular destination.

Although the first reason can be addressed by using a multi-instance queue manager, the second one is more challenging. In a real-life scenario, when the consumers that are connected to a destination stop, existing messages are stranded on the destination and new messages keep adding to this destination. This situation eventually causes the queue depth to be reached.

IBM MQ includes the AMQSCLM utility, which automatically reroutes the messages to another instance of the destination where an active listener is consuming. This utility ensures that the cluster priority of the queue manager (for a destination that does not have an active listener) is reduced so that no new messages are sent to this destination. After the listener is running, the cluster priority is adjusted and the new messages are sent to the destination for consumption.

For more information about the AMQSCLM utility, see the IBM Knowledge Center at the following website:

https://www.ibm.com/support/knowledgecenter/SSFKSJ_7.5.0/com.ibm.mq.dev.doc/q024640_.htm

Note: For more information about the initial IBM UrbanCode Deploy setup and the prerequisite steps to create the required artifacts (component, application, and environment), see Appendix A, “IBM UrbanCode Deploy initial setup” on page 149.

Figure 3-64 shows the high-level design for this scenario.

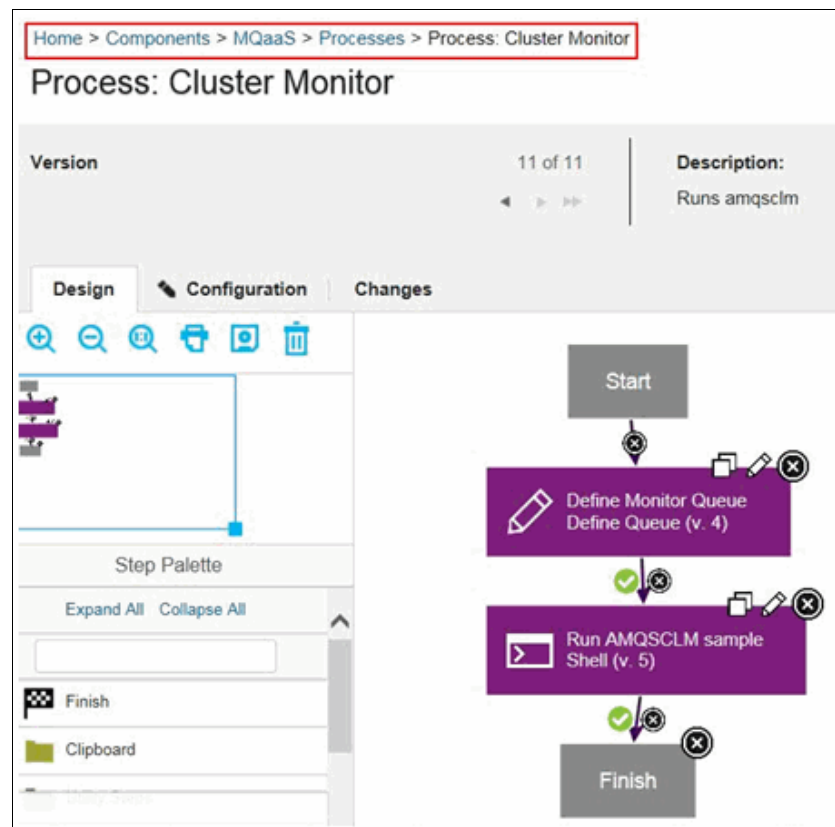


Figure 3-64 Monitor the cluster for stranded messages high-level design

Creating a cluster monitor process for the MQaaS component

This section describes how to create a process for the MQaaS component that leverages the IBM UrbanCode Deploy IBM MQ plug-in. To create the process, complete the following steps:

1. From the IBM UrbanCode Deploy web UI home page, click **Components** → **MQaaS** → **Process**. Create a process that is named Cluster Monitor and update it, as shown in Figure 3-65. Click **Save**.

Basic Settings

Name * Cluster Monitor

Description Runs amqscIm

Process Type * Operational (No Version Needed) ▼

Default Working Directory * \${p:resource/work.dir}/\${p:component.name}

Required Role None ▼

Save Cancel

Figure 3-65 Properties for creating a process Cluster Monitor

2. Open the new Cluster Monitor process, select the **Configuration** tab, and click **Component Process Properties**.
3. Add the new properties, as shown in Figure 3-66.

These properties are used as input from the user of this service.

Design Configuration Changes

Basic Settings

Component Process Properties ▶

Component Process Properties

Add Property

Name	Label	Pattern	Required
QueueManagerName	QueueManagerName		true
ClusterQueueName	ClusterQueue		true
Log_Directory	Log_Directory		false
ClusterName	ClusterName		true
MonitorQueue	MonitorQueue		true

Figure 3-66 Add the Component Process Properties

4. Create a queue to monitor messages and applications. For more information about how to define a queue, see 3.2.1, “Creating a service for single-destination instance” on page 23.

In the Step Palette, click **Middleware** → **Messaging** → **WebSphereMQ** → **Define Monitor Queue** and update Define Monitor Queue, as shown in Figure 3-67. This queue is for AMQSCLM use.

The screenshot shows the 'Edit Properties' dialog for defining a queue. The fields and their values are as follows:

- Name ***: Define Monitor Queue
- Queue Manager Name ***: \${QueueManagerName}
- Queue Name ***: \${MonitorQueue}
- Script File ***: \${p:request.id}
- Delete Script File After Execution ***: ☒
- Is Batch ***: ☐
- Command Directory ***: /opt/mqm/bin
- Working Directory**: /tmp
- Post Processing Script**: Step Default (dropdown menu) and a 'New' button
- Precondition**: (empty text area with up/down arrows)
- Use Impersonation**: ☒
- Show Hidden Properties**: ☐
- User ***: mqm
- Group**: (empty text field)

Figure 3-67 Attributes for defining the queue

5. The AMQSCLN utility must be run from within the service. To run the utility, include a shell script to run the commands for the utility.

Drag the **Shell** resource on to the process designer palette and name it **Run AMQSCLN sample**, as shown in Figure 3-68.

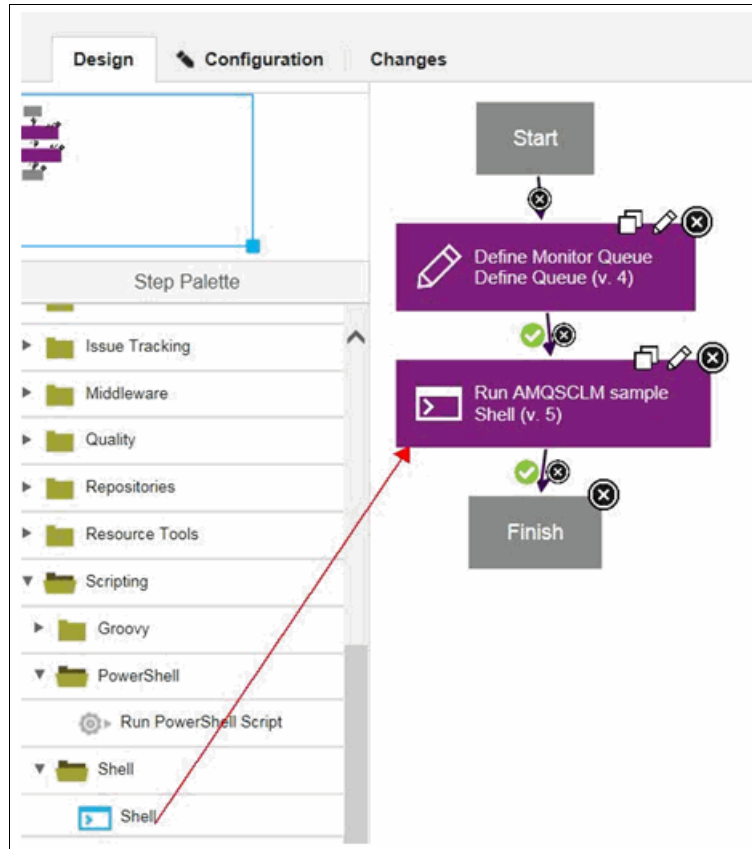


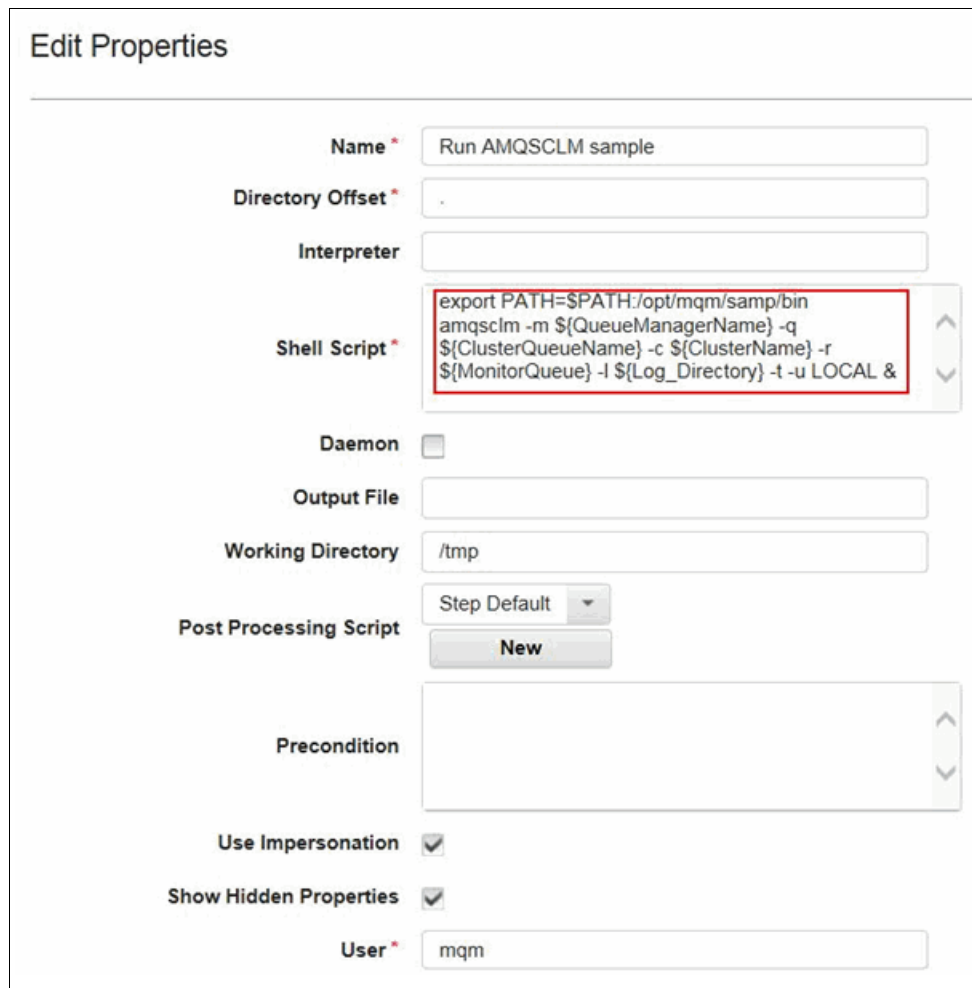
Figure 3-68 Add the shell script to run the AMQSCLN utility

Example 3-13 shows the command that starts the AMQSCLN monitor utility with the input values that are received from the user on this service.

Example 3-13 Start AMQSCLN

```
export PATH=$PATH:/opt/mqm/samp/bin
amqsc1m -m ${QueueManagerName} -q ${ClusterQueueName} -c ${ClusterName} -r
${MonitorQueue} -l /${Log_Directory} -t -u LOCAL &
```

6. Copy the script to the Shell Script field, as shown in Figure 3-69.



Edit Properties

Name * Run AMQSCLM sample

Directory Offset * .

Interpreter

Shell Script *
export PATH=\$PATH:/opt/mqm/samp/bin
amqsclm -m \${QueueManagerName} -q
\${ClusterQueueName} -c \${ClusterName} -r
\${MonitorQueue} -l \${Log_Directory} -t -u LOCAL &

Daemon ☐

Output File

Working Directory /tmp

Post Processing Script Step Default

Precondition

Use Impersonation ☒

Show Hidden Properties ☒

User * mqm

Figure 3-69 Shell script to start the AMQSCLN utility

Creating the application process

Now that you created all the required components processes to include the required resources for monitoring the queues, you now create an application process that can be associated with the component process.

Complete the following steps:

1. From the IBM UrbanCode Deploy web UI Home window, click **Application** → **MQaaS-Demo-App** → **Process**. Create a process Cluster Queue Monitoring and update it, as shown in Figure 3-70. Click **Save**.

The screenshot shows a web form titled "Basic Settings" for creating an application process. The form contains the following fields and controls:

- Name ***: A text input field containing "Cluster Queue Monitoring".
- Description**: An empty text input field.
- Inventory Management ***: A dropdown menu with "Automatic" selected.
- Offline Agent Handling ***: A dropdown menu with "Check Before Execution" selected.
- Required Role**: An empty dropdown menu.

Below the fields, there is a note: "Version presets can be configured below. Whenever a value other than 'None' is given here, the user will not be able to provide a value at runtime, and any snapshot versions for that component will be ignored."

At the bottom of the form are two buttons: a blue "Save" button and a grey "Cancel" button.

Figure 3-70 Create an application process that is named Cluster Queue Monitoring

2. From the Step Palette, click **MQaaS** → **Cluster Monitor**, as shown in Figure 3-71.

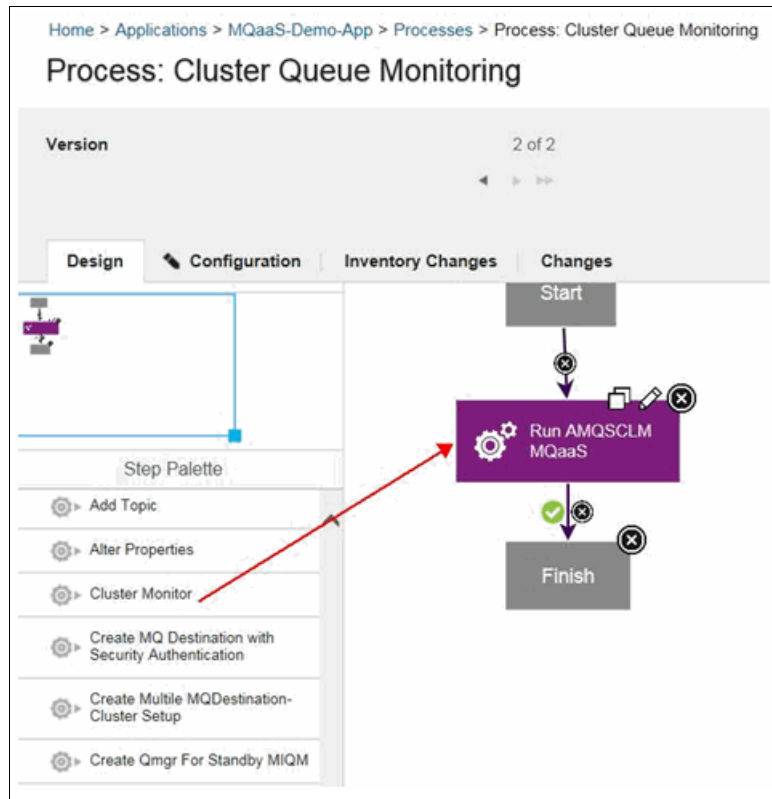


Figure 3-71 Show the cluster monitor resource that is added to the process

- On the process designer palette, click **Edit** to open the properties window for the Run AMQSCLM script and enter the properties, as shown in Figure 3-72. Click **Save**.

Figure 3-72 Properties window for configuring the AMQSCLM utility

REST API to run the service

After all the processes and services are defined, the service can be started by using a REST API, as shown in Example 3-14.

Example 3-14 Use a REST API to start the service

```
curl -k -u mqaas_user@in.ibm.com:mypassword
https://udeploy.hursley.ibm.com:8443/cli/applicationProcessRequest/request -X PUT
-d {"application":"MQaaS-Demo-App","applicationProcess":"Cluster Queue
Monitoring","environment":"vmwtpm097p.hursley.ibm.com","onlyChanged":"false","prop
erties":{"QueueManagerName":"QM1C","ClusterQueueName":"CLUSQ","ClusterName":"MYCLU
STER","MonitorQueue":"AMQSCLM.CONTROL.QUEUE","Log_Directory":"tmp"}}}
```

3.2.6 Creating a topic destination

The previous sections describe how to create a queue under different topologies. This section describes how to create a topic as a destination in a multiple instance queue manager setup.

Creating the topic destination process for the MQaaS component

For more information about how to create the component process, see “Creating a standby queue manager for the MQaaS component process” on page 35.

Figure 3-73 shows the overall design for the process after adding all the resources.

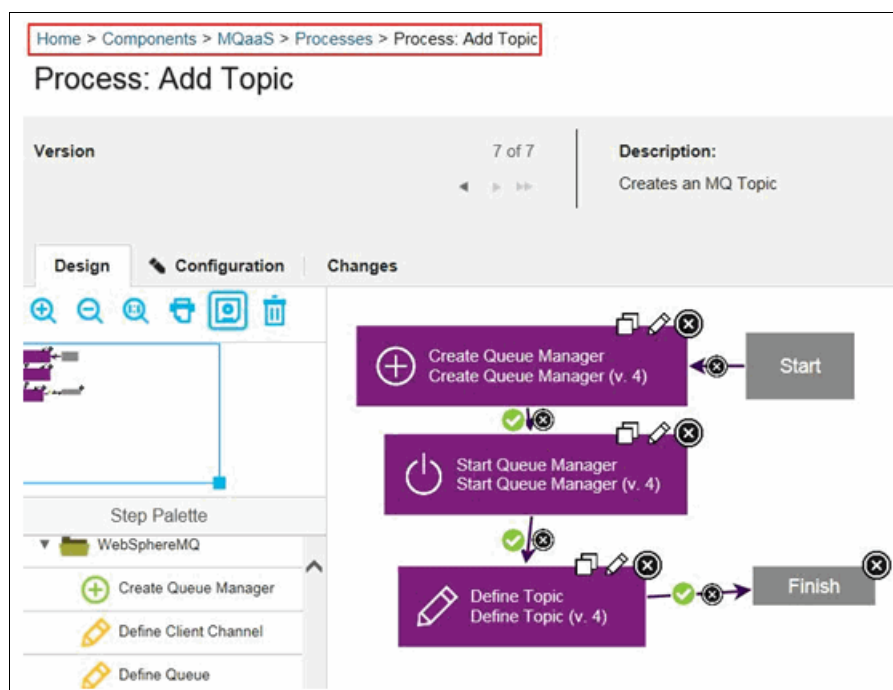


Figure 3-73 The overall design of the process

To create the topic destination process for the MQaaS component, complete the following steps:

1. Create a component process. From the IBM UrbanCode Deploy web UI Home window, click **Components** → **MQaaS**. Create a process that is named Add Topic and update it, as shown in Figure 3-74.

Figure 3-74 Create the component process Add Topic

2. In the process wizard, click the **Configuration** tab and click **Component Process Properties**. Add three new properties:
 - QueueManagerName
 - TopicName
 - TopicStringName

These properties are for input from the user of this service. See Figure 3-75.

Home > Components > MQaaS > Processes > Process: Add Topic

Process: Add Topic

Version 9 of 9 | Description: Creates an MQ Topic

Design Configuration Changes

Basic Settings
Component Process Properties

Component Process Properties

Add Property

Name	Label
TopicName	TopicName
QueueManagerName	QueueManagerName
TopicStringName	TopicStringName

Figure 3-75 Add a new Component Process Properties

3. Click the **Design** tab, from the Step Palette click **Middleware** → **Messaging** → **WebSphereMQ** → **Create Queue Manager**, and update Create Queue Manager, as shown in Figure 3-76.

Edit Properties

Name *	Create Queue Manager
Queue Manager Name *	\${QueueManagerName}
Queue Manager Description	Queue Manager for Topic Object
Command Directory *	/opt/mqm/bin
Additional Arguments	<div></div>
Working Directory	<div></div>
Post Processing Script	Step Default <div>New</div>
Precondition	<div></div>
Use Impersonation	<input checked="" type="checkbox"/>
Show Hidden Properties	<input type="checkbox"/>
User *	mqm
Group	<div></div>

Figure 3-76 Create a queue manager resource

4. From the Step Palette, click **Middleware** → **Messaging** → **WebSphereMQ** → **Start Queue Manager** and update the Start Queue Manager process, as shown in the Figure 3-77.

The 'Edit Properties' dialog for the 'Start Queue Manager' process contains the following fields and controls:

- Name ***: Start Queue Manager
- Queue Manager Name ***: \${QueueManagerName} (highlighted with a red box)
- Command Directory ***: /opt/mqm/bin
- Additional Arguments**: (empty text area with up/down arrows)
- Working Directory**: (empty text field)
- Post Processing Script**: Step Default (dropdown) and New (button)
- Precondition**: (empty text area with up/down arrows)
- Use Impersonation**: ☒
- Show Hidden Properties**: ☐
- User ***: mqm
- Group**: (empty text field)
- Password**: ****

Figure 3-77 Properties for starting the queue manager

5. Create the topic destination. From the Step Palette, click **Middleware** → **Messaging** → **WebSphereMQ** → **Create Topic**. Create Define Topic and update it, as shown in Figure 3-78.

The 'Edit Properties' dialog for the 'Define Topic' process contains the following fields and controls:

- Name ***: Define Topic
- Queue Manager Name ***: \${QueueManagerName} (indicated by a red arrow)
- Topic Name ***: \${TopicName} (indicated by a red arrow)
- Topic String ***: \${TopicStringName} (indicated by a red arrow)
- Script File ***: \${p.request.id}
- Delete Script File After Execution ***: ☒
- Is Batch ***: ☐
- Command Directory ***: /opt/mqm/bin
- Working Directory**: /tmp
- Post Processing Script**: Step Default (dropdown) and New (button)
- Precondition**: (empty text area with up/down arrows)
- Use Impersonation**: ☒
- Show Hidden Properties**: ☐

Figure 3-78 Define the properties for the topic destination

Creating an application process for a topic destination

For more information about how to create an application process, see 3.2.1, “Creating a service for single-destination instance” on page 23.

Figure 3-79 shows the complete design of the resources of this application process.

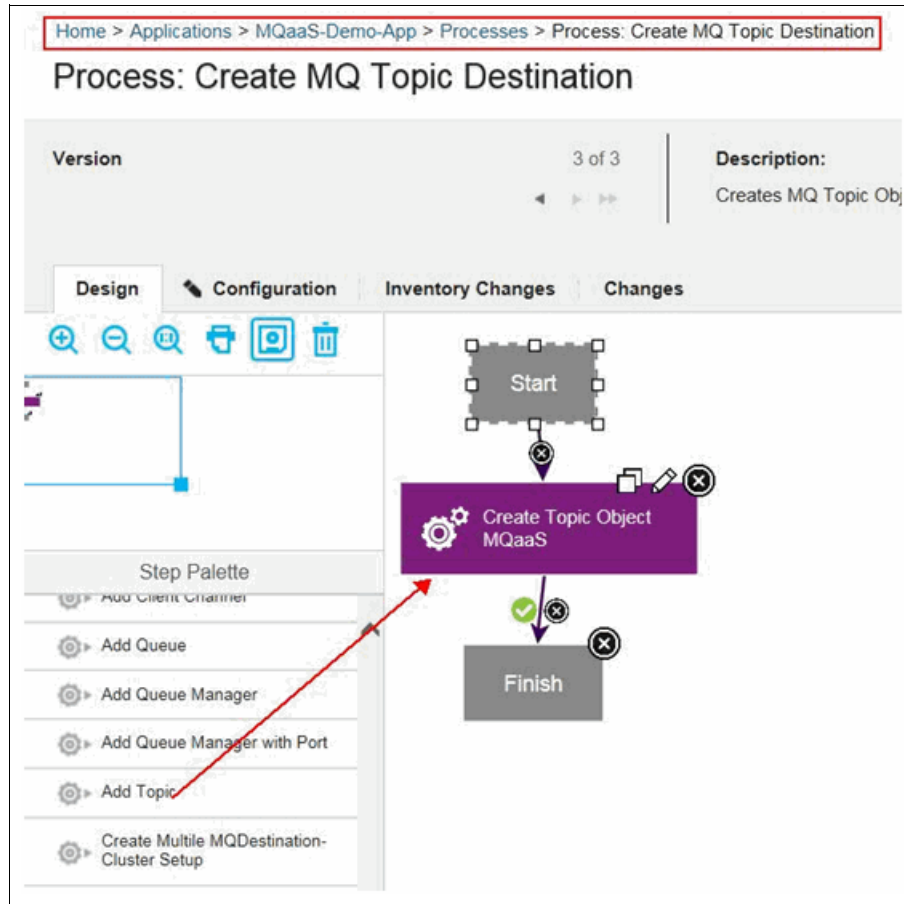


Figure 3-79 High-level design of the application process

To create an application process for a topic destination, complete the following steps:

1. Create an application process. For more information, see 3.2.1, “Creating a service for single-destination instance” on page 23.

In the IBM UrbanCode Deploy web UI Home window, click **Application** → **MQaaS-Demo-App** → **Process**. Create a process that is named Create MQ Topic Destination and update it, as shown in Figure 3-80. Click **Save**.

Home > Applications > MQaaS-Demo-App

Application: MQaaS-Demo-App

Created By
Created On

Environments

Create Process

Process

Add New Machine

Create Active Queue

Create a Destination

Create a MQ Destination

Create an Application Process

Name * Create MQ Topic Destination

Description Creates MQ Topic Object

Inventory Management * Automatic

Offline Agent Handling * Check Before Execution

Required Role

Save Cancel

Figure 3-80 Create an application process for a topic destination

2. On the Design tab, from the Step Palette, click **MQaaS** → **Add Topic**, as shown in Figure 3-81.

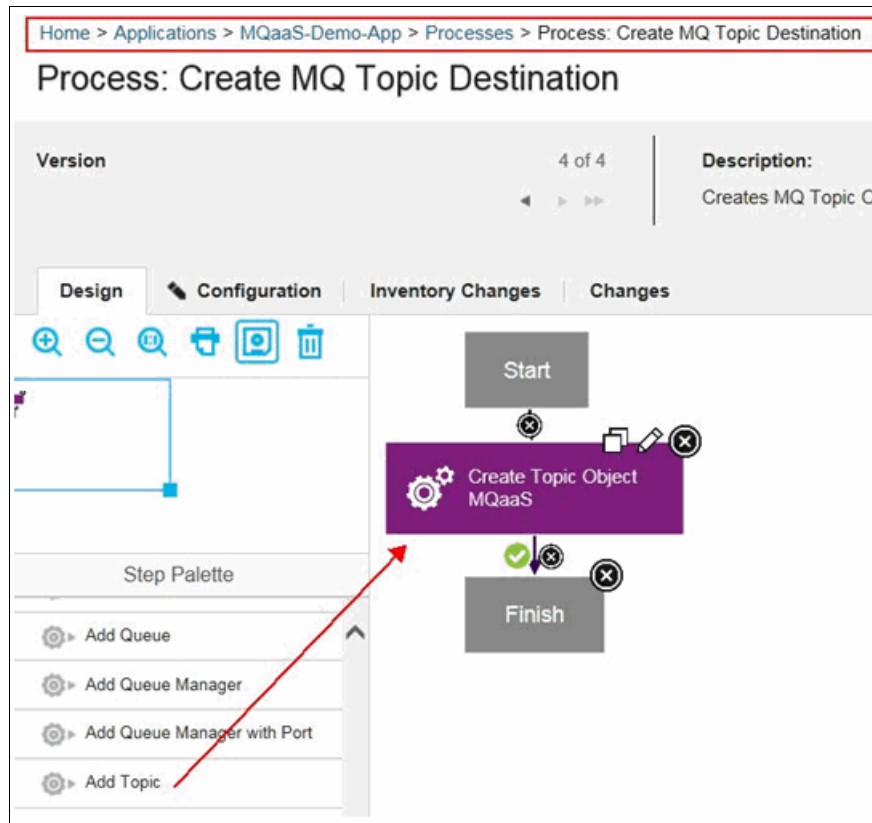


Figure 3-81 Add the topic resource from the MQaaS component

3. Update the properties for the Create Topic Object resource, as shown in Figure 3-82.

The image shows a web-based 'Edit Properties' dialog box for a resource named 'Create Topic Object'. The dialog contains several configuration fields: 'Name' is set to 'Create Topic Object'; 'TopicName', 'QueueManagerName', and 'TopicStringName' each have a 'Set a value here' link; 'Limit to Tag' is a dropdown menu; 'Max # of concurrent jobs.' is set to '-1'; 'Fail Fast' and 'Run on First Online Resource Only' are unchecked checkboxes; 'Precondition' is a large empty text area; and 'Run if Components Change' is an unchecked checkbox. At the bottom right are 'Save' and 'Cancel' buttons.

Figure 3-82 Properties for the Create Topic Object

Running the REST API

Example 3-15 shows the REST API that can be used to create the topic destination on the queue manager.

Example 3-15 Start the Create MQ topic destination

```
curl -k -u mqaas_user@in.ibm.com:mypassword
https://udeploy.hursley.ibm.com:8443/cli/applicationProcessRequest/request -X PUT
-d {"application":"MQaaS-Demo-App","applicationProcess":"Create MQ Topic
Destination","environment":"vmwtpm097p.hursley.ibm.com","onlyChanged":"false","pro
perties":{"QueueManagerName":"TOPICQMGR","TopicName":"SPORTS","TopicStringName":"S
PORTS"}}
```

3.2.7 Setting up authorizations on IBM MQ objects

This section describes how to create a service to set up authorizations for IBM MQ objects and shows an example of how to set up permissions for a queue object. The example can be replicated to set permissions on others IBM MQ objects.

Creating a component process to set up authorizations

For more information about how to create a component process, see 3.2.2, “Making the destination highly available” on page 35.

Figure 3-83 on page 91 represents the complete high-level design of the process.

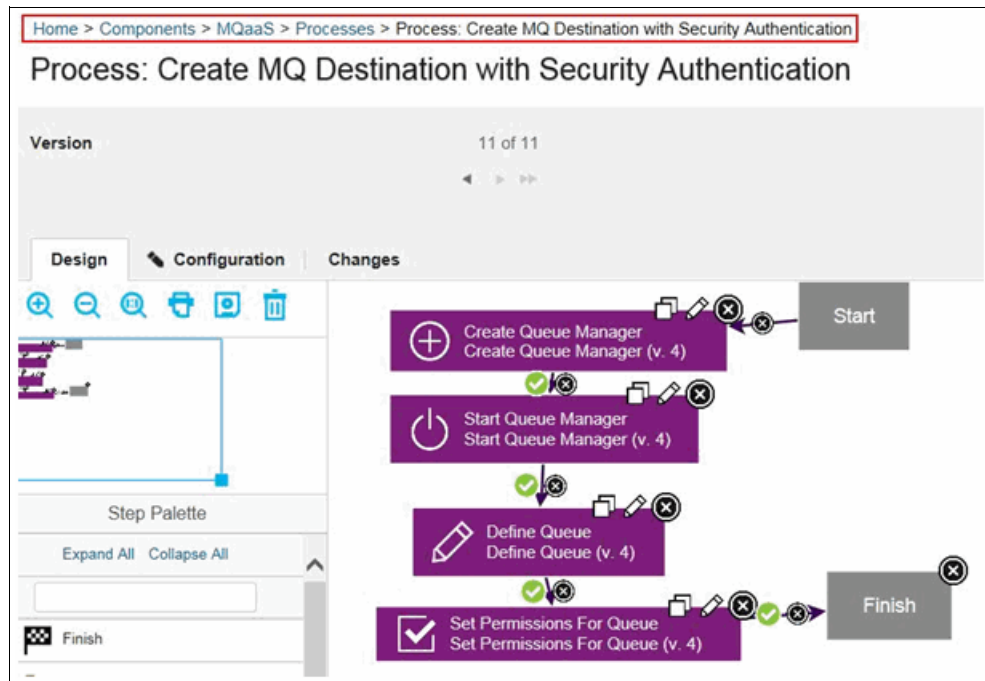


Figure 3-83 Complete design of the component process

To create a component process to set up authorizations, complete the following steps:

1. From the IBM UrbanCode Deploy web UI Home window, click **Components** → **MQaaS** → **Process**. Create a process that is named Create MQ Destination with Security Authentication and update it, as shown in Figure 3-84. Click **Save**.

Figure 3-84 Create the IBM MQ destination with security component process

- On the process wizard, click the **Configuration** tab and then click **Component Process Properties**.

Add the new properties QueueManagerName, QueueName, GroupName, Permissions, and PrincipalName as shown in Figure 3-85. These properties receive input from the user of this service.

Name	Label	Pattern	Required
QueueManagerName	QueueManagerName		true
QueueName	QueueName		true
GroupName	GroupName		true
Permissions	Permissions		true
PrincipalName	PrincipalName		true

Figure 3-85 Add new Component Process Properties

- Click the **Design** tab, and from the Step Palette, click **Middleware** → **Messaging** → **WebSphereMQ** → **Create Queue Manager**. Update Create Queue Manager, as shown in Figure 3-86. For more information, see 3.2.1, “Creating a service for single-destination instance” on page 23.

Edit Properties

Name * Create Queue Manager

Queue Manager Name * \${QueueManagerName}

Queue Manager Description

Command Directory * /opt/mqm/bin

Additional Arguments

Working Directory

Post Processing Script Step Default

New

Precondition

Use Impersonation ☒

Show Hidden Properties ☐

User * mqm

Group

Figure 3-86 Create a queue manager resource

4. From the Step Palette, click **Middleware** → **Messaging** → **WebSphereMQ** → **Start Queue Manager** and update Start Queue Manager, as shown in Figure 3-87.

The screenshot shows a web-based configuration interface titled "Edit Properties". It contains several input fields and checkboxes for configuring a step named "Start Queue Manager".

- Name ***: A text field containing "Start Queue Manager".
- Queue Manager Name ***: A text field containing "\${QueueManagerName}".
- Command Directory ***: A text field containing "/opt/mqm/bin".
- Additional Arguments**: A large text area with up and down arrow controls on the right side.
- Working Directory**: A text field.
- Post Processing Script**: A dropdown menu showing "Step Default" and a "New" button below it.
- Precondition**: A large text area with up and down arrow controls on the right side.
- Use Impersonation**: A checked checkbox.
- Show Hidden Properties**: An unchecked checkbox.
- User ***: A text field containing "mqm".
- Group**: A text field.
- Password**: A text field with four dots (password mask).

Figure 3-87 Specify the properties to start the queue manager

5. From the Step Palette, click **Middleware** → **Messaging** → **WebSphereMQ** → **Create Queue** and update Define Queue, as shown in Figure 3-88.

The screenshot shows the 'Edit Properties' dialog box for defining a queue. The dialog has a title bar 'Edit Properties' and a list of properties to be configured:

- Name ***: Define Queue
- Queue Manager Name ***: \${QueueManagerName}
- Queue Name ***: \${QueueName}
- Script File ***: \${p.request.id}
- Delete Script File After Execution ***: ☒
- Is Batch ***: ☐
- Command Directory ***: /opt/mqm/bin
- Working Directory**: (empty field)
- Post Processing Script**: Step Default (dropdown menu) and New (button)
- Precondition**: (empty text area with scrollbars)
- Use Impersonation**: ☒
- Show Hidden Properties**: ☐
- User ***: mqm

Figure 3-88 Properties for defining the queue

6. After the queue is defined, set the permission for the queue. To set the permissions, go to the Step Palette and click **Middleware** → **Messaging** → **WebSphereMQ** → **Set Permissions For Queue** and update Set Permissions For Queue, as shown in Figure 3-89 on page 95.

Home > Components > MQaaS > Process

Process: Add Setmqau

Version

Design Configuration

Step Palette

- ☒ Set Channel Monitoring Level
- ☒ Set Max Message Size On Queue
- ☒ Set Permissions For Object
- ☒ Set Permissions For Queue
- ☒ Set Permissions For Topic

Edit Properties

Name * Set Permissions For Queue

Queue Manager Name * \${QueueManagerName}

Queue Name * \${QueueName}

Principals * \${PrincipalName}

Groups * \${GroupName}

Authorities * \${Permissions}

Command Directory * /opt/mqm/bin

Working Directory /tmp

Post Processing Script Step Default New

Precondition

Figure 3-89 Properties for setting permissions on the queue destination

Creating the application process for setting permissions for queue (Create MQ Destination with Security)

For more information about how to create an application process, see 3.2.1, “Creating a service for single-destination instance” on page 23.

Figure 3-90 shows the complete design of the application process for setting the permission for the queue.

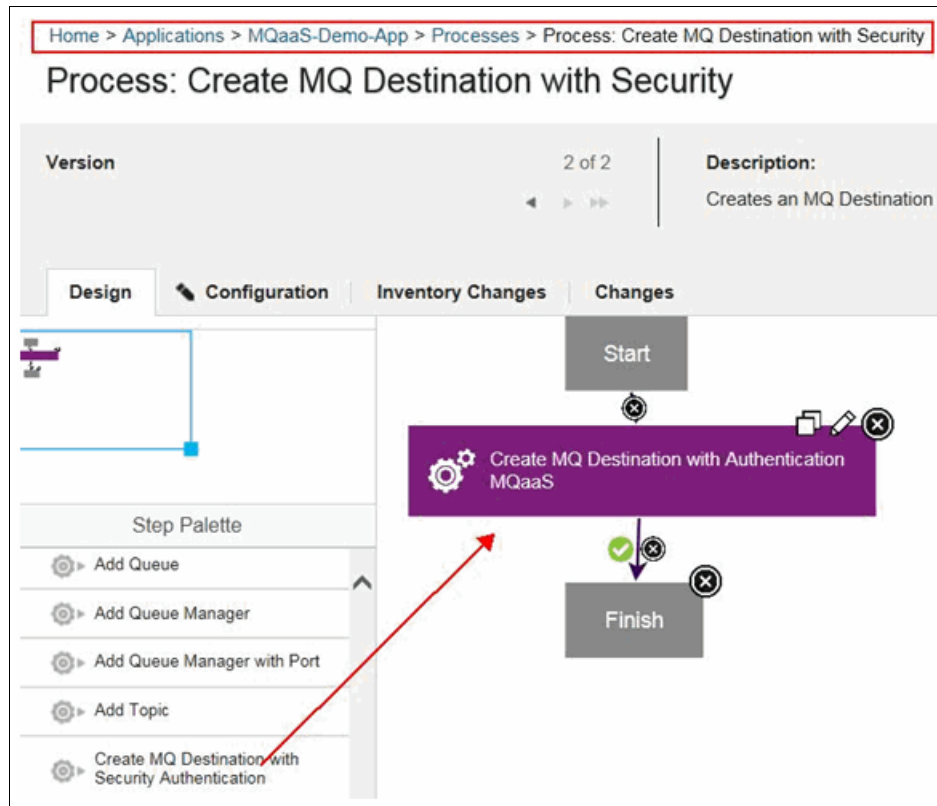


Figure 3-90 Complete design of the application process for setting permissions

To create the application process for setting permissions for queue (Create MQ Destination with Security), complete the following steps:

1. From the Step Palette, create **MQaaS** → **Create MQ Destination with Security Authentication** and drag Create MQ Destination with Security Authentication on to the process designer palette (see Figure 3-91).

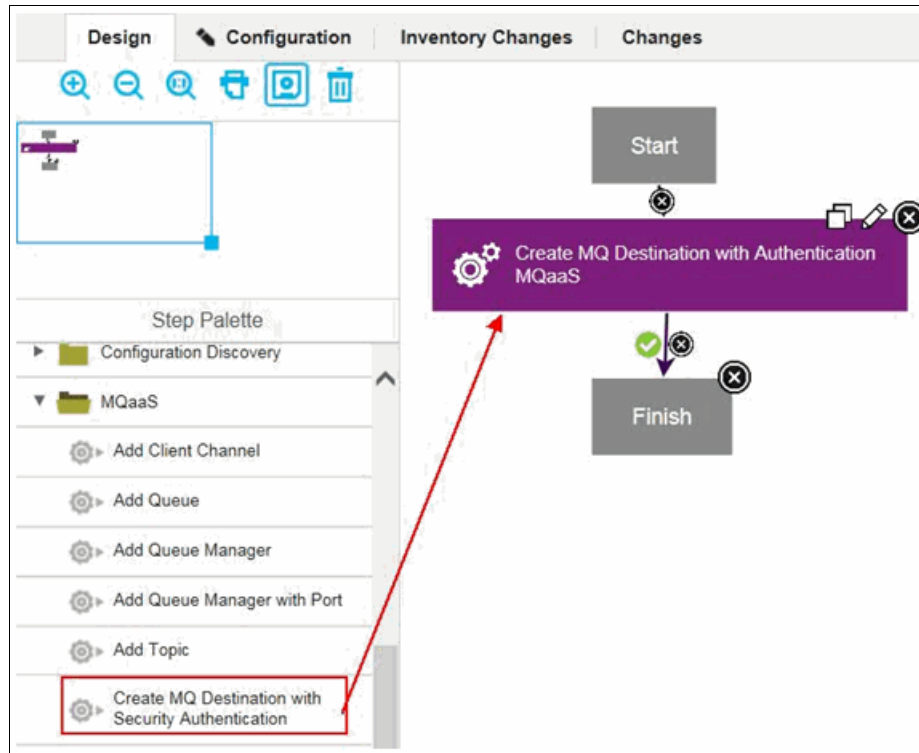


Figure 3-91 Add the Create MQ destination with Security Authentication resource

2. Update the properties for the Create MQ Destination with Security Authentication resource, as shown in Figure 3-92.

Edit Properties

Name * Create MQ Destination with Authentication

QueueManagerName Set a value here

QueueName Set a value here

GroupName Set a value here

Permissions Set a value here

PrincipalName Set a value here

Limit to Tag

Max # of concurrent jobs. * -1

Fail Fast ☐

Run on First Online Resource Only ☐

Precondition

Run if Components Change ☐

Save Cancel

Figure 3-92 Properties for the Create MQ destination with Security Authentication resource

Running the service through a REST API

Example 3-16 shows the REST API that you can use to run the service.

Example 3-16 Start Create MQ Destination with Security

```
curl -k -u mqaas_user@in.ibm.com:mypassword
https://udeploy.hursley.ibm.com:8443/cli/applicationProcessRequest/request -X PUT
-d {"application":"MQaaS-Demo-App","applicationProcess":"Create MQ Destination
with
Security","environment":"vmwtpm097p.hursley.ibm.com","onlyChanged":"false","proper
ties":{"QueueManagerName":"QMSEC","QueueName":"QSEC","GroupName":"developer","Perm
issions":"-get",
"PrincipalName":"mqaas_user"}}
```

3.2.8 Updating the properties of IBM MQ resources

Sometimes, it is necessary to update the properties of the IBM MQ resources. This section shows how to build a service that enables administrators to update the properties and attributes for IBM MQ resources.

To achieve the scenario, you need two processes.

Creating a component process to alter properties

For more information about how to create a component process, see 3.2.2, “Making the destination highly available” on page 35.

Figure 3-93 shows the high-level design of the process that is used to alter the properties of IBM MQ resources.

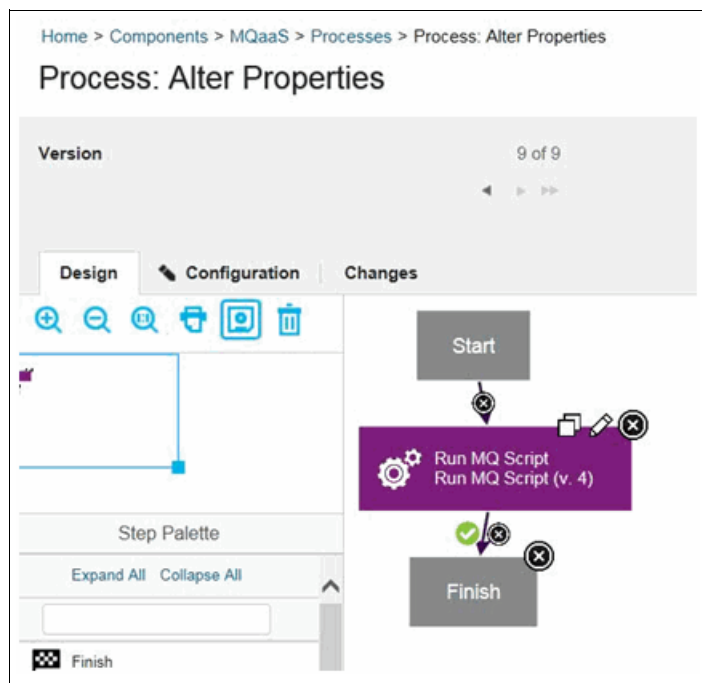


Figure 3-93 Complete process design to alter the properties of the IBM MQ resources

To create a component process to alter properties, complete the following steps:

1. From the IBM UrbanCode Deploy web UI Home window, click **Components** → **MQaaS** → **Process**. Create a process that is named **Alter Properties** and update it, as shown in the Figure 3-94. Click **Save**.

Basic Settings	
Name *	Alter Properties
Description	Alter's MQ Object Properties
Process Type *	Operational (No Version Needed)
Default Working Directory *	\$(p:resource/work.dir)/\$(p:component.name)
Required Role	None
<button>Save</button> <button>Cancel</button>	

Figure 3-94 Create the process for altering IBM MQ properties

2. Create properties for the process.

Click the **Configuration** tab and click **Component Process Properties**.

Add the properties QueueManagerName, PropertyValue, ObjectType, ObjectName, and PropertyName, as shown in Figure 3-95. These properties receive input from the user of this service.

The screenshot shows the IBM MQ Configuration console with the 'Configuration' tab selected. Under 'Basic Settings', the 'Component Process Properties' sub-tab is active. The main area displays a table titled 'Component Process Properties' with an 'Add Property' button above it. The table has four columns: Name, Label, Pattern, and Required. Five properties are listed, with the first four rows highlighted by a red border.

Name	Label	Pattern	Required
QueueManagerName	QueueManagerName		true
PropertyName	PropertyName		true
PropertyValue	PropertyValue		true
ObjectType	ObjectType		true
ObjectName	ObjectName		true

Figure 3-95 Create properties for the process

3. Create the IBM MQ script that is used to run commands to alter the properties of IBM MQ resources.

Click the **Design** tab and from the Step Palette click **Middleware** → **Messaging** → **WebSphereMQ** → **Run MQ Script**, as shown in Figure 3-96.

For this example, the following MQSC command is used. The variables are replaced by the values that are specified by the user when running the service.

```
Alter ${ObjectType}(${ObjectName}) ${PropertyName}(${PropertyValue})
```

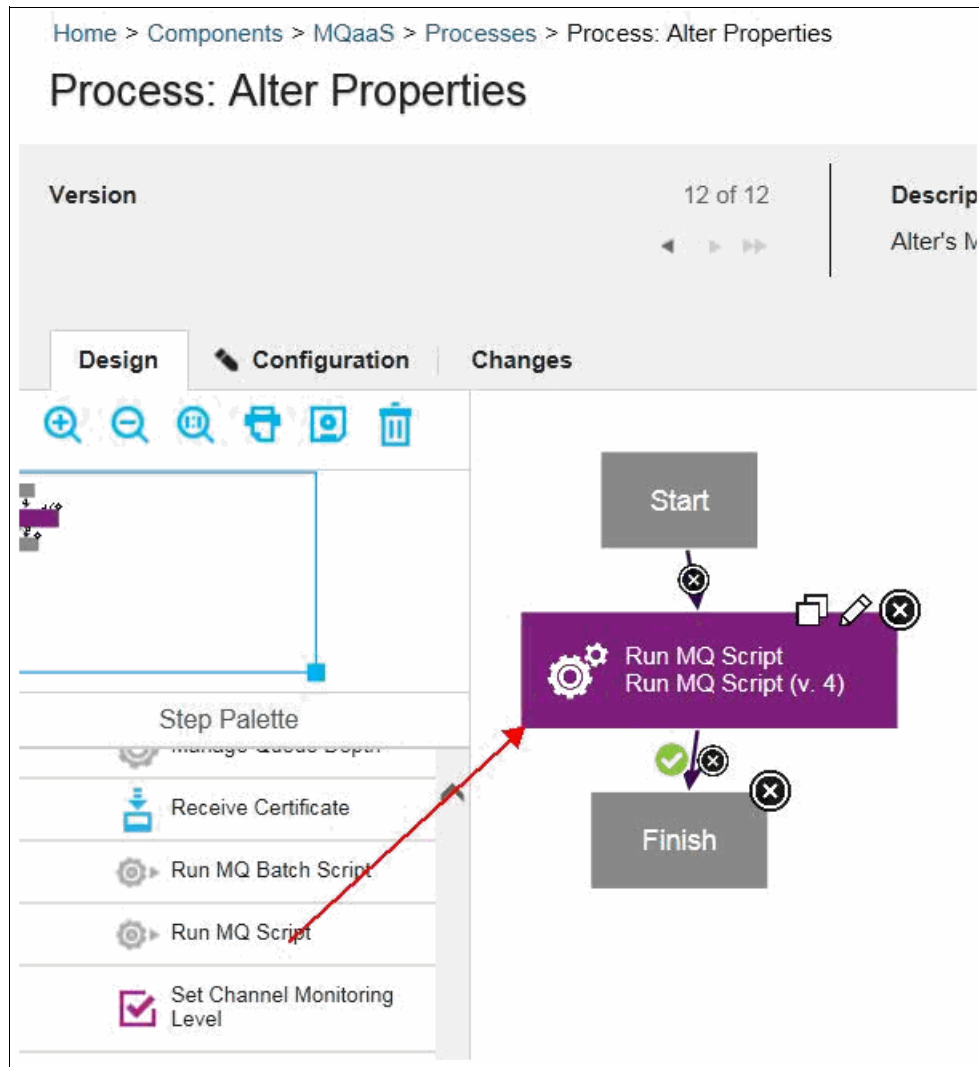


Figure 3-96 Add the Run MQ Script to run the MQSC commands

4. Drag the Run MQ Script on to the process designer palette. Enter the required MQSC commands that the service must run, as shown in Figure 3-97.

Edit Properties

Name * Run MQ Script

Queue Manager Name * \$ {QueueManagerName}

Script Data * Alter \$ {ObjectType}(\$ {ObjectName})
\$ {PropertyName}(\$ {PropertyValue})

Command Directory * /opt/mqm/bin

Additional Arguments

Working Directory /tmp

Post Processing Script Step Default
New

Precondition

Use Impersonation ☒

Show Hidden Properties ☐

Figure 3-97 Specify the MQSC command to be run by the service

Creating an application process to use the Alter Properties component process

The section “Creating a component process to alter properties” on page 99 describes the steps to create a component process that defines the required scripts to alter the properties of the IBM MQ resources. In this section, you create an application process that uses the Alter Properties component process. To create an application process, see 3.2.1, “Creating a service for single-destination instance” on page 23.

Figure 3-98 on page 103 shows the application process design to start the Alter Properties service.

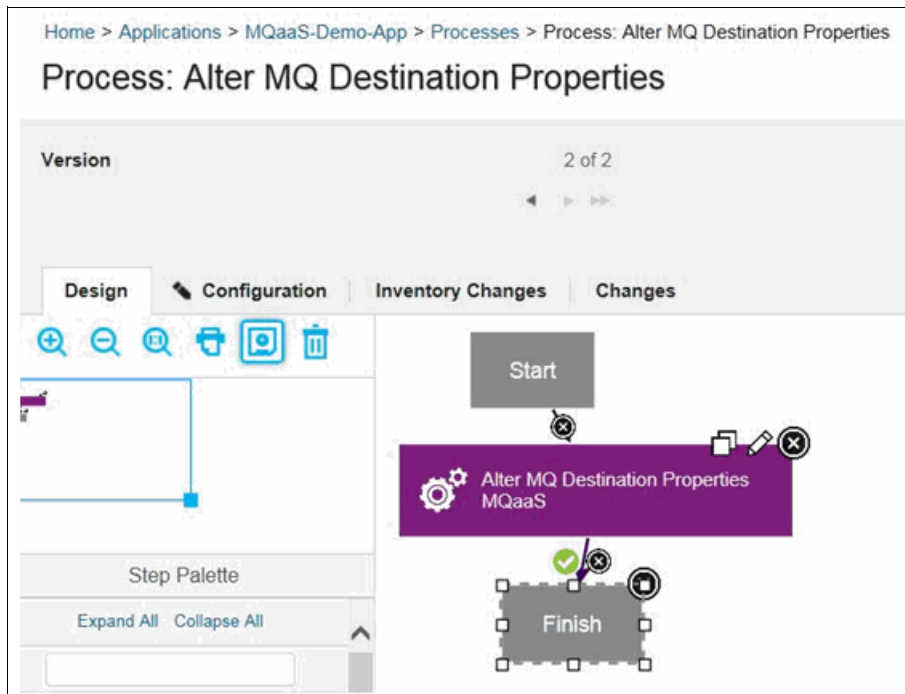


Figure 3-98 Application process design to start the Alter Properties service

To create an application process to use the Alter Properties component process, complete the following steps:

1. From the IBM UrbanCode Deploy web UI Home window, click **Applications** → **MQaaS-Demo-App** → **Create a new process**. Create the process Alter MQ Destination Properties and update it, as shown in Figure 3-99.

Figure 3-99 Properties for creating the Alter MQ Destination application process

2. From the Step Palette, click **MQaaS** → **Alter Properties** and drag Alter Properties on to the process designer palette, as shown in Figure 3-100.

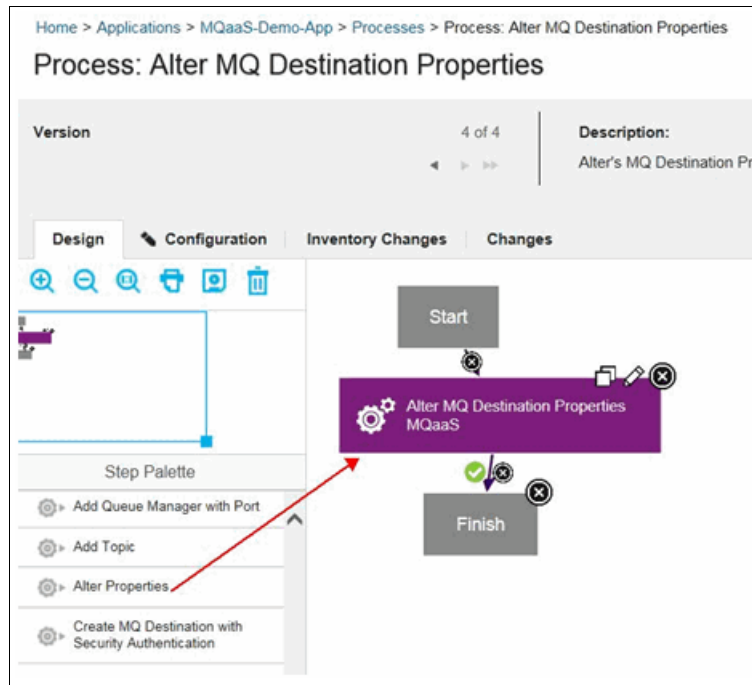


Figure 3-100 Add the Alter Properties service from the MQaaS folder

3. Set the properties, as shown in the Figure 3-101 on page 105. The values for QueueManagerName, PropertyName, PropertyValue, ObjectType, and ObjectName are replaced during run time when the service is run by using the REST API.

Edit Properties

Name *

QueueManagerName [Set a value here](#)

PropertyName [Set a value here](#)

PropertyValue [Set a value here](#)

ObjectType [Set a value here](#)

ObjectName [Set a value here](#)

Limit to Tag

Max # of concurrent jobs. * The val valid.

Fail Fast ☐

Run on First Online Resource Only ☐

Precondition

Run if Components Change ☐

Save **Cancel**

Figure 3-101 Property page for the Alter Properties service.

Running the service

Example 3-17 shows the command to run this service.

Example 3-17 Start Alter MQ Destination Properties

```
curl -k -u mqaas_user@in.ibm.com:password
https://udeploy.hursley.ibm.com:8443/cli/applicationProcessRequest/request -X PUT
-d {"application":"MQaaS-Demo-App","applicationProcess":"Alter MQ Destination
Properties","environment":"vmwtpm097p.hursley.ibm.com","onlyChanged":"false","prop
erties":{"QueueManagerName":"QM_MAIN","ObjectType":"QL","ObjectName":"QA","Propert
yName":"MAXDEPTH","PropertyValue":"500"}}
```

This procedure can update any attributes of IBM MQ resources, such as channel, topic, and queues.



Infrastructure view of IBM MQ topologies

Common requirements in new enterprise projects include continuous availability to send IBM MQ messages, high availability for individual persistent messages, linear scale of throughput, consistency between different IBM MQ installations, and reduced complexity for administrators. This chapter provides a foundational IBM MQ topology that meets these non-functional requirements for a wide range of use cases, and facilitates building IBM MQ hubs or even a shared multitenancy IBM MQ infrastructure (an IBM MQ cloud).

This chapter contains the following sections:

- ▶ 4.1, “IBM MQ hubs: An overview” on page 108
- ▶ 4.2, “Single availability zone: Active/Passive” on page 108
- ▶ 4.3, “Single availability zone: Active/Active” on page 109
- ▶ 4.4, “Multiple availability domains: Active/Passive” on page 111

4.1 IBM MQ hubs: An overview

In large-scale IBM MQ deployments, it is possible to have several hundreds to thousands of IBM MQ queue managers and their resources participating together to provide business scalability, high availability, and continuous availability. In such topologies, you must have a centralized IBM MQ infrastructure that can be scaled, managed, and administered independently from the application. Typically, applications connect to a queue manager to send and receive messages (known as the *gateway queue manager*). A single queue manager can work as a sender and a receiver gateway queue manager. IBM MQ installations have queue managers acting as the sending and receiving gateways. The sending and receiving applications attach to these queue managers as clients. *Gateway* in this instance indicates that these queue managers are the way that the application gets messages into or out of the IBM MQ network. Each application is assigned a set of queue managers to use in the sending and receiving gateway roles. A group of queue managers that a set of applications connect to is called an *IBM MQ hub*.

4.2 Single availability zone: Active/Passive

In a single availability zone, there is only one instance of the queue manager running. The queue manager instance can be made highly available either by the using the built-in multi-instance queue manager, by using external clustering software (for example, IBM PowerHA® and Veritas Cluster), or by using queue-sharing groups on z/OS for accessing individual persistent messages. Applications can be made highly available or scalable (horizontal or vertical) by connecting to the single instance of the queue manager.

Figure 4-1 shows a topology that uses IBM MQ hub.

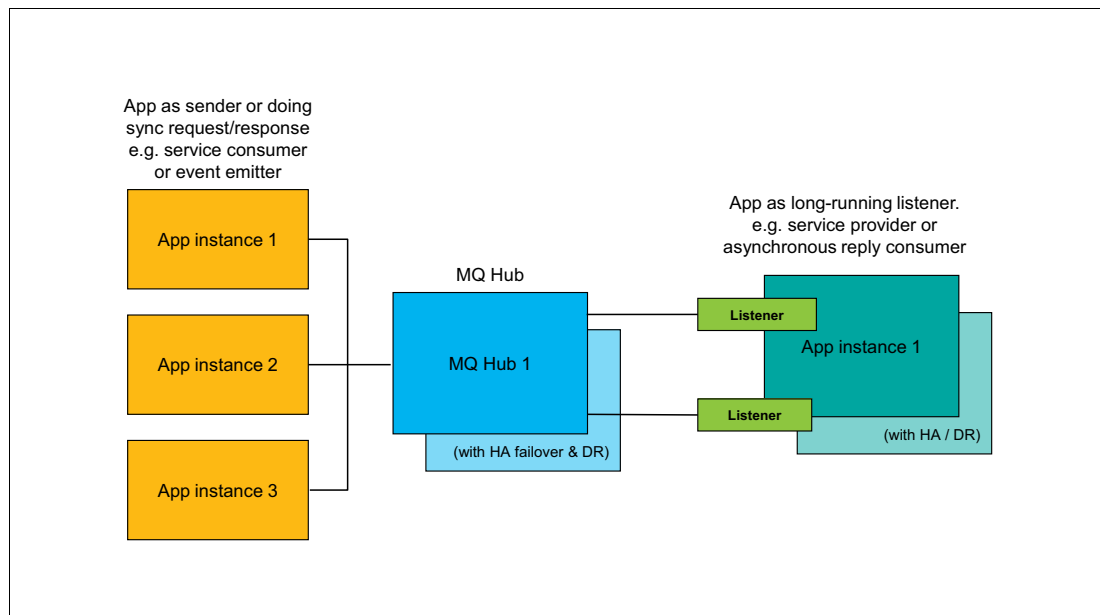


Figure 4-1 Topology that uses IBM MQ hub and a predictable load

This topology is best applicable when the load is predictable and there are a limited number of applications connecting to the IBM MQ hub. In a single IBM MQ hub instance, applications connect directly either in BIND mode (shared memory, which requires both the application and queue manager to be on the same host) or CLIENT mode (over TCP/IP) to the single queue manager instance to send and receive messages.

Table 4-1 shows some of the advantages and disadvantages of using this topology.

Table 4-1 Advantages and disadvantages of using a single IBM MQ hub

Advantages	Disadvantages
<ul style="list-style-type: none"> ▶ Everything is in one place, which makes it easier to manage and monitor. ▶ Aids performance because all messages and applications connections are on the same queue manager. ▶ Aids manageability because all messages and application connections are on the same queue manager. ▶ Serviceability and tracking of messages is simpler with fewer moving parts. ▶ Queue managers can be made highly available by using built-in multi-instance queue manager or external clustering software (PowerHA and Veritas Cluster). ▶ Provides scalability of applications. 	<ul style="list-style-type: none"> ▶ Continuous availability issues. When the active queue manager fails, the passive queue manager can take time to start, causing the applications to be unavailable during the failover. ▶ There are scalability and performance messaging issues because only one instance of the queue manager is available.

4.3 Single availability zone: Active/Active

In this topology, there can be multiple IBM MQ hubs to which applications can connect to send and receive messages. One of the many advantages of using the active/active topology is that applications can be linearly scaled (to achieve performance throughputs) and achieve continuous availability.

Figure 4-2 shows various options of how applications can connect to any queue manager to achieve scalability and continuous availability.

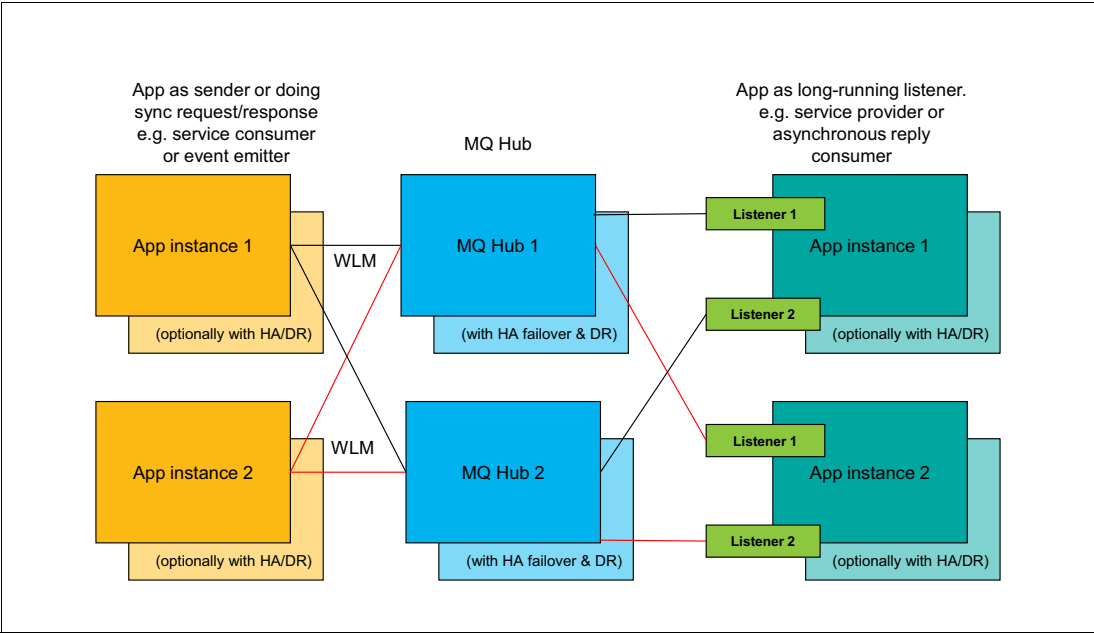


Figure 4-2 Active/Active application connecting to active/active IBM MQ hub

Figure 4-2 shows multiple applications that are configured to connect and send messages to the IBM MQ Hub. This topology uses the built-in cluster workload manager (WLM) to route automatically the messages across multiple active queue manager instances. Each queue manager is configured for high availability to address issues with stranded messages if there are failures. Similarly, each receiving application instance creates a listener to each queue manager. This method provides continuous availability for applications to process the messages.

Table 4-2 shows the advantages and disadvantages of this topology.

Table 4-2 Advantages and disadvantages of an active/active IBM MQ hub

Advantages	Disadvantages
<ul style="list-style-type: none"> ► Scalability, high availability, and continuous availability for the applications and the IBM MQ hub is achieved. ► Messages are automatically workload balanced among the available queue managers. ► Queue managers can be dynamically added or removed from the cluster to address the needs of scalability. ► Higher performance throughputs. 	<p>Serviceability and tracking of messages can be hard because messages might traverse across multiple queue managers.</p>

Figure 4-3 on page 111 shows a topology that is similar to the one that is shown in Figure 4-2, except that each application instance connects to only one IBM MQ hub.

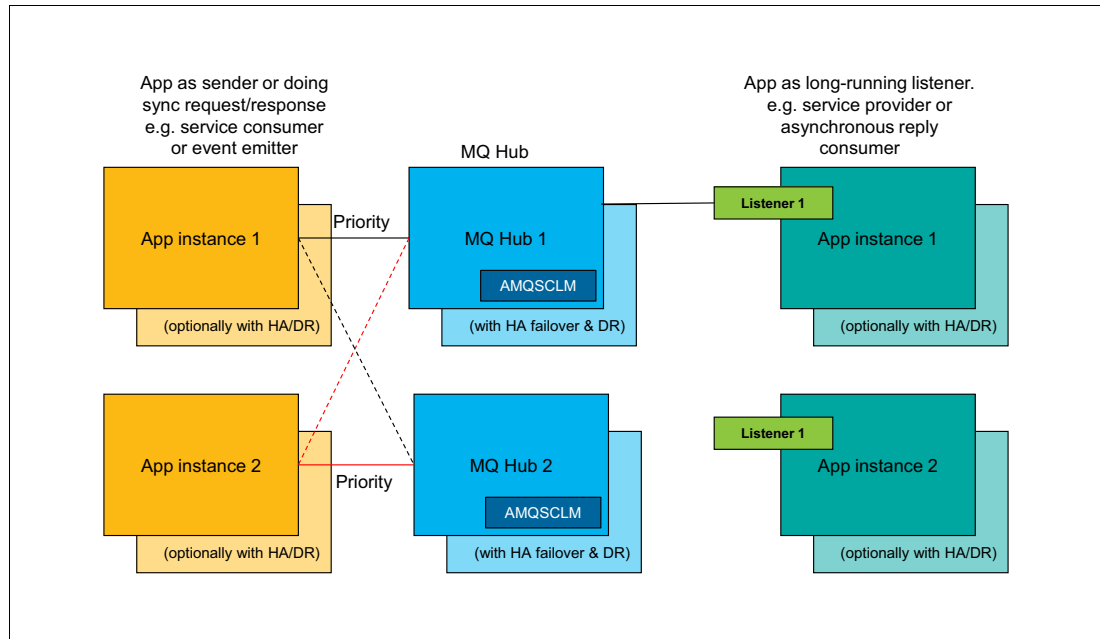


Figure 4-3 Active/Active application connecting to an active/active IBM MQ hub by using priority and the AMQSCLN utility

In this topology, the application instances connect to only one IBM MQ hub. For example, sender application instance 1 always tries to connect to IBM MQ Hub1 if it is available; otherwise, it connects to IBM MQ Hub 2. Similarly, each receiving application instance creates only one listener pointing to the queue manager of the individual IBM MQ hub.

It is possible that one of the listeners is down and the application can tolerate some delays. If any of the listeners is not available for some duration, then you can use the AMQSCLM utility along with the cluster management to divert the stranded messages to an active listener instance.

4.4 Multiple availability domains: Active/Passive

A multiple availability topology primarily refers to addressing the needs of disaster recovery (DR). In today's *always-on* global business environment, organization leaders need the fastest possible failover and fallback of their critical business applications to guard against disasters of all shapes and sizes.

Every organization should put processes and procedures in place to ensure that mission-critical business functions can continue during and after a disaster. The emphasis is more on maintaining business operations rather than IT operations. In a typical active/passive scenario, the systems at the DR site are not doing any work. The data, logs, and configurations at the primary data site center must be periodically replicated (either synchronously or asynchronously) to the backup data center. The replication in itself is outside the scope of IBM MQ, and customers must rely on external disk replication techniques to ensure that data consistency is maintained between the primary and the backup data centers.

Figure 4-4 shows a topology for an active/passive scenario.

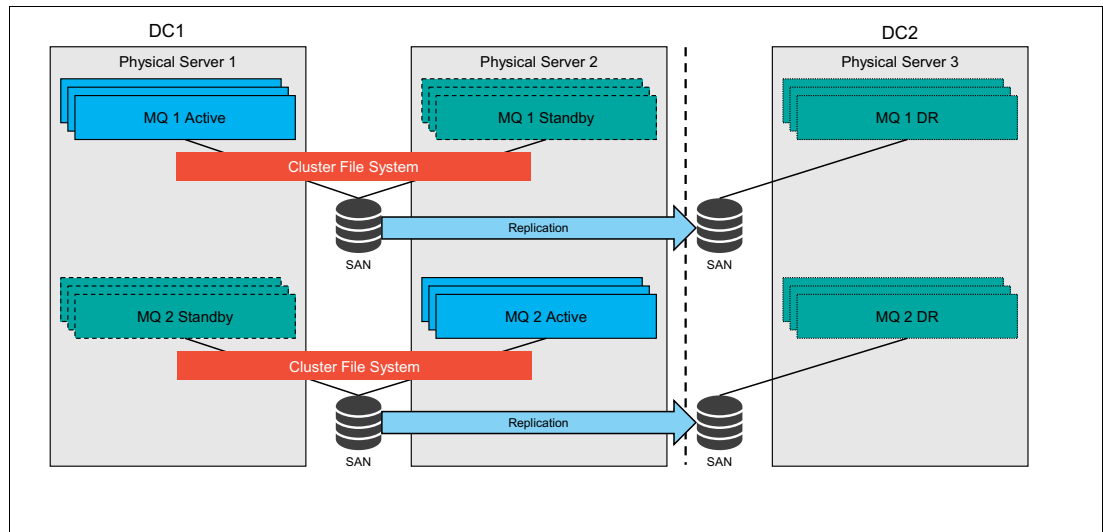


Figure 4-4 Active/Passive disaster recovery site



Other factors to consider when planning IBM MQ as a service

This chapter describes additional requirements that the infrastructure team must be aware of and that can impact the usage of IBM MQ resources.

This chapter contains the following sections:

- ▶ 5.1, “Security model for IBM MQ as a service” on page 114
- ▶ 5.2, “Monitoring” on page 116
- ▶ 5.3, “Multitenancy” on page 122

5.1 Security model for IBM MQ as a service

The security model that is used is an important factor in all deployments of IBM MQ. The three *As*, authentication, access control, and auditing, are especially important to think about when IBM MQ is deployed as a service, given that typically these requirements and responsibilities are likely to be used for self-service. In addition, the requirements and responsibilities for integrity and privacy of message data is likely to be an important factor, particularly in multitenancy environments.

5.1.1 Authentication

The strength of the best security models comes from having multiple layers of defense. For example, a car that is left unlocked parked on a street has less security to prevent it from being stolen than a locked car parked in a locked garage. Authentication should be considered as the first line of security, and therefore is important. If you cannot identify who is accessing IBM MQ resources, other security measures can be undermined, such as access control.

Authentication can be single or multifactor in nature. Identification of a user to IBM MQ can be based on a single type of credential, such as the correct password for a user ID, or multiple credentials, such as the correct password and a trusted digital certificate. IBM MQ supports various forms of authentication in the core product function. The authentication functions can be further extended through the use of channel security exits and pluggable authority manager.

Authentication in IBM MQ can be broken down into local and network access vectors, that is, programs that are running on the same local operating system as the queue manager and connections that are made over a network connection. Authentication over a network connection can be more challenging than local access. A program that connects locally to IBM MQ already performed OS-level authentication, and a user ID that is known locally can be asserted as a credential. With network-based connections, there is no such implicit trust in any user IDs that are asserted across a network connection, so user ID values alone should not be used for authentication.

User ID and password

IBM MQ V8 can authenticate user ID and passwords against either local OS or LDAP repositories through core product capabilities, which are implemented by the connection authentication (CONNAUTH) attribute. Authentication against other repositories is possible through the use of security exits for client applications or through a custom object authority manager (OAM) component.

User ID and password checks are valid for both local and network-originated connections. LDAP user ID and password checks are one way in which a centralized repository can be used to validate the identity of a user requesting access to IBM MQ resources.

In a self-service environment, it is likely that a centralized repository is a preferable choice to avoid the need to create and delete local OS user accounts.

IBM MQ V8 can be configured to use access control that is based on local or LDAP users and groups or to authenticate against LDAP and map to a local user account for access control checks.

Digital certificates

For authentication of remotely connected client applications, X.509 digital certificates provide a strong reliable form of authentication.

Client channels that use secure communications protocols such as TLS 1.0 or TLS 1.2 can be configured to require client authentication. In fact, the default setting for such channels is to require SSL authentication (SSLCAUTH(REQUIRED)). In this configuration, the client must provide an X.509 digital certificate that is signed by a trusted certificate authority (CA), and it must pass any optional Distinguished Name (DN) matching rules.

In addition to authentication, configuring TLS for a client connection channel also provides privacy and integrity characteristics so that any data that is sent in the network has protection against being read or altered by man-in-the-middle (MITM) attacks.

Other credentials

In addition to user ID and password and digital certificates, there are other credentials or characteristics that can be used to authenticate a connection. However, these credentials tend to be weaker forms of authentication.

For example, the source IP address of a TCP/IP connection can be used as a form of identity. However, it is possible for a determined attacker to impersonate or *spoof* a source IP address, and so it is not preferable to use an IP address as the only factor in authentication checks. However, IP address as one authentication factor can be useful to augment the stronger authentication checks.

After the authentication of credentials is proven, IBM MQ can assign or map the credentials to an identity that is used by IBM MQ to perform further security checks.

IBM WebSphere MQ V7.1 and later releases provide a flexible rules-based system to map credentials to an identity from IBM MQ client connections. The rules-based system is known as channel authentication (CHLAUTH). This system supports multi-factor authentication of channel connections by using user names, digital certificates, and IP addresses.

5.1.2 Authorization and access control

After authentication is performed and an identity is proven, an administrator can assign different roles and access to IBM MQ resources that are based on the identity.

Access control is granted either to a user (or principal) or to a group. The decision to opt for a user or group model is determined by many different factors. A group model is a good model for a self-service environment, that is, the same access rights are granted to all members of an OS or LDAP group.

Typically, access control is divided into two distinct vectors: access that is required for applications and access that is required for administration and monitoring purposes. IBM MQ provides a granular set of authorities that can control activities down to message queue interface (MQI) calls, such as allowing browsing, but not destructive removal of messages.

Members of the overall IBM MQ administrators group can create, start, end, and delete queue managers. They have full administrative and API privileges to a queue manager. In all environments, membership of the IBM MQ administrators group should be highly restricted and limited. In a self-service environment, this recommendation is especially true given that access to queue manager resources cannot be revoked for users in these groups.

Access control can be managed for IBM MQ resources that exist on a queue manager or that might come into existence either through specific or generic profiles. The latter approach is interesting when a queue manager is clustered, providing multitenancy, or is in some other way provisioned for self-service.

The naming scheme that is adopted for IBM MQ resources can have an impact on the administration effort that is required to configure access control correctly. A well-defined naming scheme where object names, such as queues, follow a strict set of rules with regards to name prefixes can simplify the setup of generic profile rules. For example, if a queue manager hosts queues for sales, accounts, payroll, and HR departments, and if the names of the queues that are used by these departments are prefixed with the department names, then authorities to the queues need to be set up only once.

5.2 Monitoring

To provide stable performance and highly available services, you must be able to monitor effectively the operation of each service. Identifying potential issues as early as possible ensures that a service provider can quickly respond and take remedial action before users are impacted.

However, monitoring is more than just identifying problems as they occur; it is also fundamental for effective change management, aiding application development, and understanding the ongoing operational needs of a service. This section describes the monitoring challenges and requirements for a messaging service and the features that are provided by IBM MQ to address them.

5.2.1 Resource monitoring

Monitoring the resources that underpin a messaging service is essential. These resources comprise the operating environment resources, such as real storage (memory), processing capacity (CPU), and auxiliary storage (disks). Also included in the resources are the internal resources of the messaging system, such as message queues and recovery logs.

Tools to manage the operating environment and the messaging resources are provided by the operating system and the messaging system, but increasingly users are looking for a single portal where everything can be monitored together.

The following interfaces are provided by IBM MQ to facilitate resource monitoring:

- ▶ Queue manager output, such as the error log files AMQERR01.LOG on distributed systems and the job log on z/OS
- ▶ Online status commands
- ▶ Offline accounting and statistics
- ▶ Event notifications
- ▶ IBM MQ Explorer (Eclipse tools)
- ▶ Web UI (IBM MQ Appliance)

IBM MQ queue managers output operational messages to their log that can be used by administrators to monitor messaging activity. These messages can be informational, such as reporting that a channel started or stopped, or they can indicate resource constraints or problems to which an administrator must respond.

Each message has a unique identifier and action code that can be used to filter output when monitoring an active system. Filtering messages allows an administrator to focus quickly on those messages that are important, while maintaining a record of other activities.

Online commands can be issued to IBM MQ to obtain near real-time status information for messaging resources, such as queues, topics, and channels.

IBM MQ also provides commands to query the active application connections and the resources that each connection is using. These commands can be issued interactively by users or programmatically by IBM MQ Explorer, SupportPacs, homegrown tools, or vendor products by using a documented API called Programmable Command Formats (PCF), or a REST API that is introduced for the web UI that is provided with the IBM MQ Appliance.

The ability to obtain near real-time information about the depth of message queues, or the status of channel connections that are used for network communication, is important when problems are encountered or an administrator must verify normal operation. These commands can be issued locally or from a remote system for central administration. IBM MQ Explorer and the web UI provide a graphical environment for issuing commands and displaying their output.

As messaging topologies grow, it becomes less feasible to monitor resources by using near real-time queries. In these environments, it is far more consumable for notifications to be emitted by the messaging system when important events occur.

IBM MQ can be configured to output various event notifications. For example, event notifications can be generated when channels start or stop or in response to errors that require remedial action, such as a Secure Sockets Layer (SSL) handshake failure. Event notifications can also be generated in response to resource errors that are encountered by applications, such as attempting to open a queue that is not defined.

5.2.2 Monitoring performance

Services are provided with an associated quality of service (QoS), which often covers both availability and performance. It is necessary for service providers to monitor the services they offer to ensure that this criteria is met and to verify that the services are running efficiently. IBM MQ generates event notifications that can be processed by administrators so they can respond to performance issues in near real-time fashion. Statistics data can also be captured that can be used for offline analysis, both for problem determination and capacity planning for the future.

Messaging is fundamentally about delivering data from source to destination. The value of the business data that is represented in messages can be realized only when those messages are consumed. Messages can pile up on the destination for several reasons. For example, if messages are generated faster than they are consumed, they accumulate in the messaging system. Messages might accumulate because of application constraints, insufficient processing capacity, or the availability of or connectivity to the destination to which they are sent. IBM MQ can generate the following performance event notifications:

- ▶ Queue depth high, low, and full alerts
- ▶ Queue service interval alerts

A queue depth high event is generated by IBM MQ when the number of messages on a queue reaches a configured threshold, which is defined as a percentage of the configured maximum queue depth. This threshold can be configured independently for each queue.

A queue depth high event indicates that messages are not being processed quickly enough, so action is required to either increase the number of application threads processing them, divert messages to an alternative location, or take some other remedial action.

Similarly, a queue depth low event is generated when the depth of a queue reduces to a separately configurable threshold, indicating that the accumulation of messages is remedied.

A queue full event is generated when a queue reaches its maximum permitted depth.

Queue service interval events are generated to indicate whether an operation is performed on a queue within a user-defined time interval. These events can be used to determine whether messages are consumed at the required rate.

IBM MQ can be configured to record statistics data about queues, channels, internal IBM MQ resources, and the number of MQI calls made by applications. Data is captured during a configurable interval, which is written to System Management Facilities (SMF) on z/OS, or as a message on other platforms.

Accumulating and analyzing statistics data can provide valuable insight into the level of messaging activity. Statistics data can also be used to identify resource constraints or suboptimal configurations.

5.2.3 Monitoring application activity

Monitoring application activity can be beneficial both for system administration and application development. IBM MQ can be configured to record accounting data for applications, such as statistics.

Accounting data captures the MQI operations that are performed by each application. This data can be used by service providers to charge consumers based on their service usage. For example, users can be charged based on the number or size of messages. Service providers can also use this information to understand application messaging patterns or to record an audit trail of activity.

Accounting data is captured at user-defined intervals, so it is primarily designed for offline analysis. It can be helpful to trace the messaging activity of an application to aid debugging by examining individual API parameters, or just to understand what activity an application is performing. This information is captured in near real time, so it can be consumed immediately for timely analysis.

5.2.4 Change management and auditing

One of the most important aspects of monitoring is tracking changes. Understanding what recently changed, who made the changes, and why the changes were made, provides vital insight when diagnosing the cause of an issue.

External documents or tools are often used to track why a set of changes were made, but relying on this mechanism to detail the specific updates that are made by administrators can be prone to error. The most reliable way to track changes that are made to a system is for the system to track the changes automatically.

IBM MQ can generate configuration event notifications to record administrative actions, including:

- ▶ Administrative commands that were issued, who issued the command, and how the command was provided.
- ▶ Administrative changes that are made to object definitions, such as queues and topics, which can be correlated with the administrative commands that instigated them.
- ▶ Authority failures. For more information, see 5.1, “Security model for IBM MQ as a service” on page 114.

5.2.5 Consuming events

The previous sections describe various types of event notifications that can be generated by IBM MQ. Each notification is represented by a message that is written to a system event queue on the local queue manager. Each type of event is written to a different system queue so the different types of event can be processed separately. For example, performance events are written to a different queue to configuration events because it is likely that timely action is required to respond to a growing queue depth, and configuration changes are typically captured for auditability.

As the number of queue managers increases, it can become complex for a service provider to manage. Therefore, it is common to route event notifications to a central location where they can be processed and reported collectively. For example, a portal or dashboard can be used to report event notifications for an entire messaging topology or a set of provisioned environments.

IBM MQ can be configured to route event messages to a central location by redefining the system queues to which event notifications are put. Each event message contains information that can be used to identify uniquely its origin, so changing these system queues to remote queue definitions allows the messages to be routed to a remote queue manager for centralized collection. Each event also uniquely identifies its type so all notifications can be routed to a single queue for processing if segregation of events by their type is not wanted.

It is sometimes useful or even necessary for event notifications to be sent to multiple parties. This action is especially relevant for service providers who might want to retain a record of events for themselves, while also notifying users of the messaging service. To help with this goal, IBM MQ allows each system event queue to be redefined as an alias to a topic. If IBM MQ is configured in this way, when an event is generated, the notification is published to each subscriber for the associated topic. This technique allows event notifications to be duplicated and consumed by multiple recipients. The target of a subscription can be a remote queue definition, so configuring events for publish/subscribe does not prevent centralized collection of these notifications. This technique can be useful for service providers who do not readily know what events users want to consume; users can subscribe to the events they require.

5.2.6 Tracking messages

One of the most common messaging problems is messages not arriving at their expected destination. In this scenario, users might complain that their messages are missing, but this situation is rarely true. It is far more likely that a problem was encountered when routing the messages to their destination, so they are being held on a queue within the messaging topology.

It is also possible that a configuration error routed the messages to the wrong location. When faced with this problem, where should you start? Fortunately, there are many techniques that can be employed to find the missing messages, and there are features of IBM MQ that can assist in this process.

Before you start to look for messages, consider where they might be located and the reason for them being there. The following locations should always be considered:

- ▶ **Transmission queues:** Messages are held on a transmission queue if the channel that should process them is not active. The channel might be inactive because of a connectivity issue with the destination, it is disabled, or a problem was encountered delivering a message to a target queue at the destination and a dead-letter queue was not configured.
- ▶ **Dead-letter queues:** Messages are routed to a dead-letter queue if they cannot be delivered to a target queue so they are not lost. There are many reasons why this situation might occur, but the most common reasons are because the target queue does not exist, the target queue is full, or access to the target queue is denied.
- ▶ **Destination queues:** If multiple instances of a destination queue are defined, such as in a cluster configuration, it is worth checking all instances to ensure that they are being serviced by applications. If an application instance is not running, messages can still be routed to the queue, but they are not processed. It is also worthwhile checking for uncommitted messages at the destination, which is unavailable until the unit of work in which they were put is completed.

A useful technique is to replay the steps a message takes from source to destination. A message might have to traverse a number of hops to reach its destination; verifying that each hop was successfully traversed in turn reduces the problem scope.

IBM MQ includes a feature that is called Trace Route that can be used to identify the route that a message might take to its destination. This feature allows special tracer messages to be injected that generate activity reports as they are routed through an IBM MQ topology. They are destroyed when they reach their destination so that they cannot be unexpectedly processed by an application.

The activity reports are routed back to the source so that each hop a trace message takes can be reported in the same way as by tools such as traceroute. This information can quickly identify where a message is being routed to the wrong location or where its traversal through the topology unexpectedly halts.

You can then immediately focus on the problem area. If a message is being routed to the wrong location, verify the target of remote queues or aliases. When multiple routes are possible, multiple trace messages are likely to be required to identify all the paths. It is possible that only a subset of the available routes encounters a problem. The most common scenarios where multiple routes are used are when a cluster configuration is used or messages are routed within a queue-sharing group by using either channels or the Intra-Group Queuing agent.

The two most common tools that are used to take advantage of trace route messages are the **dspmqrte** command-line utility that is provided with IBM MQ and the MS0P SupportPac extension to IBM MQ Explorer. Trace route messages can also be a useful technique to verify that the cluster workload balancing is configured correctly.

5.2.7 Challenges and limitations

The previous sections describe monitoring requirements and the capabilities of IBM MQ that can be used to satisfy them. This section summarizes the monitoring challenges when providing a messaging service and some of the limitations that exist for IBM MQ.

5.2.8 Impact of monitoring

IBM MQ provides a range of monitoring capabilities for service providers and users. However, the cost of monitoring a system must be considered.

Some features, such as status commands, are relatively cheap from a performance perspective unless they are issued at a high frequency. Other features, such as capturing accountancy data, are more expensive from a performance perspective.

Service providers must assess what benefits each capability offers, how the capabilities help satisfy operational requirements, and what impacts are incurred. For example, on systems where there is only one customer, capturing accounting data might not be necessary all of the time. It might be preferable to capture it only for short intervals instead. However, on systems where there are multiple tenants sharing access to the same queue manager, then accounting information might be necessary for accurate chargeback.

The impact of monitoring is not limited to the performance cost that is associated with capturing the data. You must also consider the cost of processing or even not processing this information. For example, if many event messages are generated, but they are not consumed, then the depth of the queues on which the event notification messages are held grows. In the short term, this situation might not be a problem, but eventually the cost of storing this data might become significant. Similarly, for systems that are resource-constrained, the cost of processing the monitoring output might potentially impact the ability to meet committed response times.

5.2.9 Configuring monitoring and consuming monitoring data

Service providers must consider what access they are willing to provide to users and how user access might constrain the monitoring options that can be employed.

If users require access to queue manager output, such as the log file AMQERR01.LOG or the z/OS job log, then access to log on directly to the systems might be necessary. Enabling many monitoring options requires administrative authority within IBM MQ. Is granting this level of authority acceptable? Service providers might want control over the wider configuration, so they might not want to allow some monitoring options to be enabled by users without their consent. This situation is particularly pertinent when a queue manager is servicing multiple tenants.

IBM MQ is supported on a wide range of platforms, but the offering on z/OS has a different heritage. Therefore, service providers might need to consider how the monitoring capabilities on this platform differ from Windows, Linux, and the other UNIX variants. These differences might become significant when exposing the captured information to users. For example, security events, accounting data, and statistics data are all captured as PCF event messages on platforms other than z/OS. On z/OS, this information is captured elsewhere. The external security manager records security events and the System Management Facility (SMF) captures accounting and statistics information. Capturing this information in these external resources is highly beneficial for the z/OS platform. However, for users that want to consume this information in a consistent format for all platforms, this difference might be an issue, especially when using a generic messaging service.

5.3 Multitenancy

Service providers might want to achieve economies of scale by hosting a messaging system for multiple users or tenants. This approach can be implemented by using multiple virtual hosts on the same physical server, hosting multiple queue managers in the same server instance, or even hosting the queues for multiple users in the same queue manager. Each option has its own challenges and considerations, from security and isolation to monitoring.

The previous sections describe the impact of monitoring and considerations for how monitoring can be configured and its output consumed. These items are important in a multitenancy environment because the needs of one user might be inconsistent with the needs of other users. It is important for service providers to consider these needs when allocating messaging resources. When users share a queue manager or have queue managers on the same server instance, there are some additional challenges:

- ▶ Tenants might have different requirements regarding the version of IBM MQ that they use. This requirement might originate from the need to take advantage of newly added monitoring or other administrative capabilities. From Version 7.1 onward, IBM MQ supports multiple installations that can be used to satisfy this requirement.
- ▶ If tenants require local access to administer their queue managers by using control commands or they require other administrative authority, it might be problematic to implement a shared server instance on platforms other than z/OS. The reason for this situation is that the administrative authority that is granted by the mqm group on these platforms has a server scope that applies to all queue managers on a server instance.

Many authorities can be granted explicitly to avoid the need to grant access to mqm, but certain operations still require this access. One reason a tenant might require this level of access is if they need to start or stop a queue manager after making changes to its configuration, such as editing `mqat.ini` to configure defaults for activity trace.

- ▶ Commands are issued to a queue manager, so if a queue manager services multiple users, then access to resources must be restricted to prevent them accessing each other's queues or messages. IBM MQ provides various levels of access control that can be used to achieve this effect, including command, queue, and even message-level protection. If a user requires administrative access to a queue manager, such as the ability to start, stop, or make global changes to it, then it is likely that they require a dedicated queue manager to ensure the service that is offered to other users is isolated.
- ▶ Event notifications that are generated by IBM MQ are put to system queues that have a queue-manager scope. If a queue manager services multiple tenants, then notifications for all users are put to the same named queues. Using topic aliases and publish/subscribe to duplicate each event message allows tenants to receive their own copy of the notification. However, it is challenging to filter the events so that each tenant receives only those events that are applicable to them. This situation is because IBM MQ is not tracking which resources are used by a given user or application, which is necessary to achieve the required filtering.

A similar problem to filtering events exists for messages sent to a dead-letter queue. Each queue manager has a single dead-letter queue. If a queue manager is shared and one user requires access to messages that were routed to the dead-letter queue, then they also have access to the messages of other users. To address this problem, access to the dead-letter queue should be restricted, and a dead-letter queue handler is used to process the messages. The dead-letter queue handler should route each message to an equivalent queue that is created for each tenant, which should be protected. However, the same challenge of mapping each message to a tenant exists.

5.3.1 Resource allocation

Service providers must consider how the available resources should be partitioned for each tenant. It is important to consider carefully resource allocation because it has a significant impact on the ability to satisfy user requirements and system administration goals.

Users might demand a high level of isolation, but this requirement must be balanced with the complexity and inflexibility that this isolation might introduce. Allocating separate server instances to each tenant, either physical or virtual, provides greater opportunities for isolation, but is likely to introduce greater maintenance and management impacts.

Here are the resources that should be considered:

- ▶ Hardware resources, such as real storage (memory), processing capacity (CPU), and auxiliary storage (disks)
- ▶ Operating system resources, such as processes, file systems, and user accounts
- ▶ Networking resources, such as bandwidth, ports, and IP addresses
- ▶ Messaging resources, such as queues, topics, and channels

Allocation of hardware resources must be carefully managed to avoid over committing the available capacity. It is important to understand the peak usage requirements for each tenant, not just their typical processing needs. Is there sufficient capacity to maintain QoS if all tenants peak at the same time? If not, what are the implications, and which tenants should have priority? Some tenants might demand complete isolation to avoid being impacted by the workload of others. Consideration is necessary to determine whether these tenants should be allocated a dedicated proportion of the available capacity, and the other consumers share the remaining resource.

Service providers that choose to host multiple tenants on the same server instance must consider how the operating system resources should be shared. Separate file systems might be required to prevent one tenant exhausting the available disk capacity.

Operating system resources, such as shared memory and user accounts, are also important. It might be necessary to consider what process limits should be configured for each tenant, especially if they can run local applications. If many processes are created for one tenant, how might this impact the available processing (CPU) time awarded to others?

It is also necessary to consider how networking resources are allocated. Planning the use of the available bandwidth is important to ensure that the messaging service can function efficiently. Careful planning should also be given to the allocation of ports and IP addresses. Allocating a virtual IP address to each tenant provides flexibility to move resources from one system to another with fewer configuration changes required to applications. Coordinating the allocation of TCP/IP ports is necessary to prevent conflicts. If tenants require specific ports to be allocated, this situation might impact how their messaging service can be provisioned. Tenants that require the same port numbers must be hosted on different servers.

A service provider might consider hosting multiple tenants on the same queue manager, especially if their messaging requirements are limited. In this case, it is important to employ a naming convention for IBM MQ objects, such as queues and channels. Allocating a namespace to each tenant, such as a high-level qualifier for object names, prevents conflicts and simplifies administration.

It is also important to consider the topic tree for tenants that want to use publish/subscribe. Creating a high-level topic object for each tenant is a simple way to allocate them a unique portion of the topic tree. If one tenant requires many topics or subscriptions, it might be advisable to host their messaging service on a separate queue manager. Similarly, it is important to consider the wider messaging topology of each tenant.

5.3.2 Security and administration

In a multitenancy environment, service providers must consider how the data of each tenant and the resources that are allocated to them are protected. IBM MQ provides a comprehensive set of security profiles that can be configured to restrict access to messaging resources, ranging from administrative commands to the ability to access queues and the encryption of data. However, how resources are allocated to each tenant can greatly impact the complexity of enforcing access control. Using a high-level qualifier for the messaging objects of each tenant simplifies the security profiles that are required and makes it easier to identify which tenant is affected if an issue is encountered.

Service providers also must consider how the appropriate security profiles are established when provisioning resources for each tenant. This situation is especially important for shared resources. If users require local access for administration, there is additional complexity to ensure that one tenant cannot inadvertently, or even maliciously, interfere with another tenant. For example, access to each queue manager must be restricted independently to allow a tenant to administer their queue managers, but not others. This approach might become complicated to manage if privileged access, such as the ability to start or stop queue managers, is required.

If users are allowed to start resources, such as TCP/IP listeners for a queue manager, it might be necessary to prevent a tenant from using a port that is allocated to someone else. It is also important to consider what access is permitted to the queue manager configuration and data files. If tenants have data that is especially sensitive, then using Advanced Message Security (AMS) might be necessary to prevent others from being able to view data at rest, which is important when complying with government or international standards, such as PCI or HIPAA. If AMS is required, it is also necessary to consider how the security certificates are going to be managed to prevent them from being compromised.

5.3.3 Maintenance

Service providers must consider how to maintain their messaging service and the systems on which it runs. In a multitenancy environment, there is additional complexity to coordinate maintenance with each user. The fewer queue managers there are to manage, the less maintenance activity is required, but it might be more difficult to schedule.

You can use IBM MQ to start and stop queue managers independently, which can be used to stagger outages. You can use features such as queue-sharing groups on z/OS and clusters to ensure continuous availability by routing messages elsewhere while a queue manager is unavailable. From IBM MQ V7.1 onward, multiple versions of IBM MQ can coexist on the same server, regardless of the operating system that is being used. Before this version, coexistence was supported only on z/OS.

In a multi-installation environment, queue managers can be migrated from one installation to another independently, which supports staggered upgrades from one release to another. This technique can also be used to upgrade from one maintenance level to another one. Capabilities such as this one were added to IBM MQ to reduce the outage window when migrating a queue manager.

Support for multiple installations can be used by service providers to upgrade the messaging service for one tenant while preserving the environment that is used by others. Without this ability, it is necessary to relocate the tenant's resources to a different system, which introduces additional complexity and risk.

In environments where the messaging service for multiple tenants is hosted by a single queue manager, then it remains necessary to coordinate maintenance to an agreed time and duration. In this environment, service providers must establish a procedure for quiescing queue managers. If a queue manager does not quiesce in a timely manner, how long should an administrator wait for applications to end before terminating them? If an application does not disconnect, how easy is it to identify the tenant that is affected?

Provision for backup and recovery is another aspect of maintenance that it is important for service providers to consider. Tenants likely have different backup and recovery requirements that might impact how a messaging service is provided.

Service providers should ensure that they establish policies for managing backups for queue manager configuration files and queue data, plus any associated system resources. Providers also must plan how these resources can be restored if there is failure. Restoring a queue manager from a backup because of a tenant's requirement might affect multiple tenants. If one of the tenants is unwilling to revert to a previous backup, there must be a process for managing this situation.

5.3.4 Troubleshooting

Service providers must plan how tenants can troubleshoot problems that they might encounter. For simpler problems, limited administrative access is necessary, such as the ability to connect to a queue manager and display or manage messaging resources. An example might be to inquire into the status of, or manage, queues or channels that are owned by the tenant. Providing access to resources with a queue-manager scope, such as the dead-letter queue, is likely to be contentious if they are shared with other tenants.

If it is necessary for tenants to enable diagnostic tools such as trace to obtain support from IBM, determine how this activity might affect other tenants. Enabling diagnostic tools is likely to cause a degradation in performance and can consume some of the available storage capacity for the output. If tenants are able to enable diagnostics themselves, decide how to manage it in a way that other tenants are protected from this activity. If access to log on to a server is not available to a tenant, plan in advance how they diagnose problems that require this level of access, such as the ability to view queue manager logs and IBM First Failure Support Technology™ (IBM FFST™).



Infrastructure choices for IBM MQ as a service

To create an efficient IBM MQ as a service implementation, the IBM MQ queue managers must be deployed on an infrastructure that is flexible and meets the needs of the enterprise. This chapter introduces various deployment and topology options for IBM MQ.

This chapter contains the following sections:

- ▶ 6.1, “Virtualization” on page 128
- ▶ 6.2, “Containers” on page 130
- ▶ 6.3, “IBM MQ patterns” on page 131
- ▶ 6.4, “IBM MQ Appliance” on page 133
- ▶ 6.5, “IBM Cloud Orchestrator and IBM MQ integration” on page 134

6.1 Virtualization

Virtualization is key to building cloud-based architectures because it allows greater flexibility and utilization of the underlying resources (such as servers, network, and storage). Instead of requiring a separate physical resource for each tenant, multiple tenants can be separated logically on a single underlying infrastructure. This concept is known as *multitenancy*. Depending on the infrastructure, a tenant can be an individual application, internal team/department, or an external customer. A virtual machine (VM) or a hypervisor is the foundation for realizing the virtualization concepts. A VM can be viewed as an operating system (OS) or an application environment that is installed on software that represents the underlying hardware. From a user perspective, VMs provide the same experience as dedicated hardware.

There are several benefits to using VMs instead of dedicated hardware for applications:

- Isolation from the host operating system

You can deploy and run applications within the VM environment without risking the stability of the underlying host platform. The user has the flexibility to package multiple applications together in a single VM and can deploy multiple instances of the VM on the same hardware while each of the applications is isolated from each other.

- Replication

You can replicate the VM contents (that is, the applications, configurations, and installed applications) to another server. The infrastructure team can create an image consisting of IBM WebSphere Application Server, IBM MQ, and IBM DB2 products together. The team can do the connections between these products (for example, setting up queues and topics on IBM MQ, creating the required connection factories in the WebSphere Application Server product, and creating the required tables in the DB2 product). You can give this VM image to a developer who can readily use its contents for their application development knowing that it contains the correct setup with no external dependencies.

- Recovery

You can perform a recovery when a failure happens. Taking snapshots at regular interval supports recovery from failure. If there is a failure, the administrator can revert to the previous snapshot of the VM.

- Efficiency and ease of use

VMs save expenses by reducing the need for multiple physical servers. Instead of dedicating entire physical machines to certain applications or tasks, VMs can more efficiently use hardware through multitenancy without risking cross contamination. This approach lowers the quantities of hardware and associated maintenance costs and reduces power and cooling demand.

Figure 6-1 on page 129 shows a cloud that contains virtual systems.

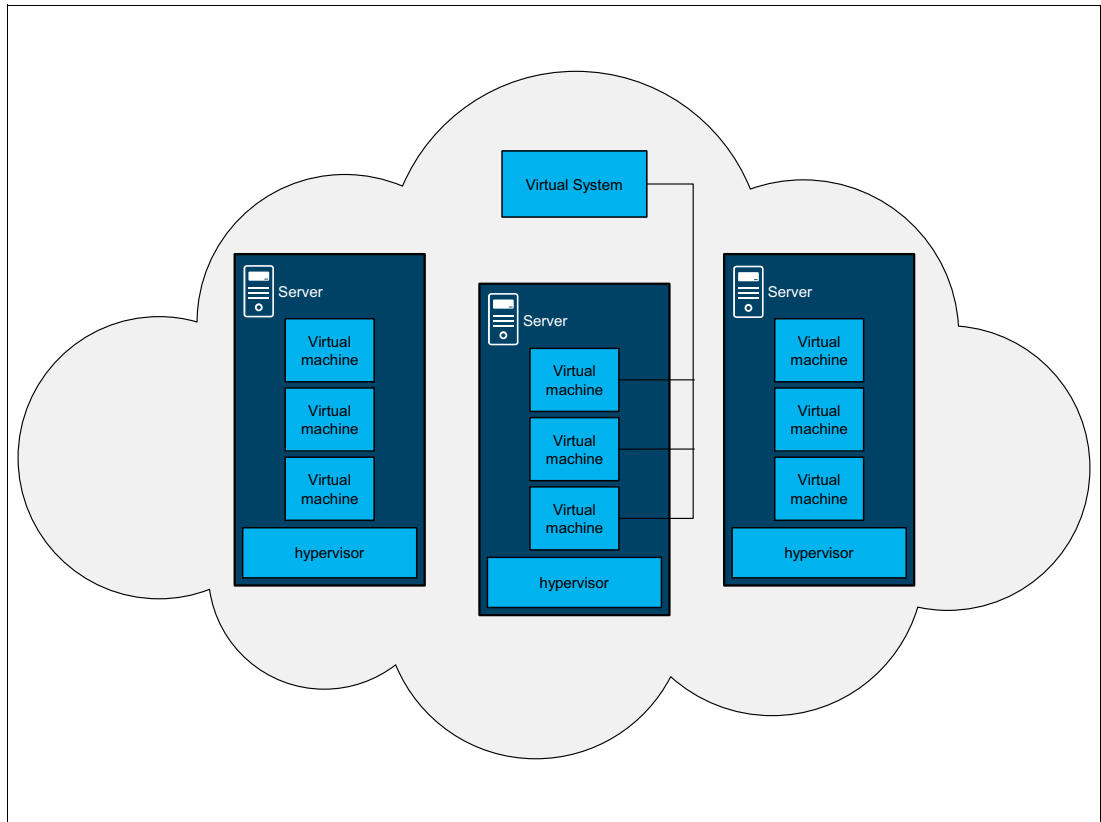


Figure 6-1 A cloud that contains virtual systems

6.1.1 IBM MQ virtualization support

By creating virtual images, businesses can ensure that systems are built with a known, stable, and tested configuration, reducing the potential for errors and ensuring rapid deployment of a working system, whether for development, test, or production.

You can build virtual images that can run IBM MQ queue managers. These VMs can be created for many platforms. In fact, IBM supports IBM MQ running in a VM based on any of the platforms that support a particular version of IBM MQ. This action is described in *IBM MQ's support position on Virtualization, low-level hardware, file systems on networks and high availability*, found at the following website:

<http://www.ibm.com/support/docview.wss?uid=swg21392025>

With this support, it is a simple case of creating a virtual image and installing and configuring the required OS. Then, you install IBM MQ onto the virtual image along with any other necessary components or products. Now, you have a virtual image that can be used multiple times to create reliable IBM MQ environments. You can use these images in successive phases of deployment, such as development, test, and production, which reduces the chances of environment-based issues when transferring from one phase to the next.

In the past, IBM MQ provided pre-packaged hypervisor editions that are pre-built VMs that contain a specific base OS and a particular version of IBM MQ. This approach is initially convenient, but can prove to be restrictive in the versions and platforms that are supported and might not fit your particular demands. Building your own VMs might be more suitable.

6.1.2 Provisioning virtual machines

After creating virtual images with IBM MQ correctly installed, configured, and tested, these images can be incorporated into the provisioning infrastructure of your choice to enable IBM MQ as a service. This image might be a system running in your own data center, that is, a private cloud, or even a public cloud-based solution.

Integrating the provisioning of VMs with the IBM MQ resource configurations that are defined on them (such as queue managers and their queues, and channels) results in a true *as a service* solution. For example, it is possible to extend the IBM UrbanCode Deploy flows that are described in 3.2, “Creating an IBM MQ as a service catalog” on page 23 to include the provisioning of new VMs to host the queue managers being deployed.

6.2 Containers

Similar to the more established virtualization techniques, containers automate the deployment of applications and infrastructure by providing an additional layer of abstraction and automation of the OS-level virtualization.

Containers offer many of the benefits of virtualization, but typically at a lower impact because of the way they abstract the underlying OS into a container.

One such container technology is Docker¹. IBM recently announced support for running IBM MQ inside Docker containers. The rest of this section uses Docker as a reference to explain the container concepts from the IBM MQ point of view.

6.2.1 Docker

Docker is a tool that can package an application and its dependencies in a virtual container that can run on any Linux server. You can use this approach to enable flexibility and portability on where the application can run, such as on-premises, on a public cloud, on a private cloud, and on bare metal. Multiple containers share a kernel, but each container can be constrained to use only a defined amount of resources, such as CPU, memory, and I/O. The container can have its own process ID space, file system structure, and network interfaces.

Using Docker to create and manage containers might simplify the creation of highly distributed systems by allowing multiple applications, worker tasks, and other processes to run autonomously on a single physical machine or across multiple VMs. You can use this approach to perform the deployment of nodes as the resources become available or when more nodes are needed. You can use this approach to provision a Platform as a Service (PaaS) style of deployment and scaling for systems.

In summary, Docker/container functions fall into several categories:

- ▶ Portable deployment of applications as a single object versus process sandboxing
- ▶ Application-centric versus machine/server-centric
- ▶ Supports automatic container builds
- ▶ Built-in version tracking
- ▶ Reusable components
- ▶ Public registry for sharing containers
- ▶ A growing tools infrastructure from the published APIs

¹ <https://www.docker.com/>

For more information about how to use IBM MQ with Docker, see the GitHub, Inc. website:

<https://github.com/ibm-messaging/mq-docker>

Here are some use cases that involved IBM MQ on Docker:

- ▶ DevOps
 - As a DevOps team, you want to provide IBM MQ dynamically.
 - As a DevOps team, you want to provide an IBM MQ server where all dependencies are managed, whether it is a development environment, test environment, or pre-production environment. Each of these environments gets the same IBM MQ server with all the prerequisite software, so you eliminate the risk of each environment having different dependencies.
- ▶ Infrastructure as a service (IaaS)

As an IaaS offering, you can make IBM MQ available as a container, and multiple containers can be used. This approach addresses the load and performance aspects of IBM MQ when it is on IBM SoftLayer®.
- ▶ Enterprises might want to use IBM MQ in Docker containers to address resource management, high availability, and disaster recovery.
- ▶ On the IBM Bluemix environment (PaaS):
 - The availability of IBM MQ in Docker containers, where containers can be deployed on Bluemix, enables IBM MQ accessibility on Bluemix.
 - When a third party uses IBM MQ and Docker containers setup, you can use this approach to use IBM MQ Docker containers and deploy the containers on Bluemix as a SaaS offering.

6.3 IBM MQ patterns

Patterns are solutions to problems that infrastructure and development teams face during the development, test, and production phases. Patterns include a set of configuration steps and scripts that can be run multiple times to create specific topologies with minimal human intervention. The benefit of using patterns is that they can be automated, you can use them to predict what comes next, and you can solve problems that are tedious to solve otherwise.

In a typical scenario, the infrastructure team can create a set of IBM MQ related patterns either based on a line-of-business (LOB) requirement or application needs. These services can be started by the development or application teams. The following sections describe some of the patterns that are available for IBM MQ.

6.3.1 IBM MQ Virtual System Pattern Type

IBM MQ Advanced Virtual System Pattern Type for IBM PureApplication System (referred to in this documentation as IBM MQ Virtual System Pattern Type) contains an IBM MQ plug-in (the virtual system software component) and associated script packages, which you can use to create a virtual system pattern for your IBM MQ environment.

You can use the IBM PureApplication System product to manage virtual applications in a cloud-computing environment, where data and services are in data centers. The PureApplication System product can be placed in a data center to dispense applications and topologies into a pool or cloud of virtualized hardware and to manage these resources. The data and services can then be accessed from any connected devices over the internet. By using the PureApplication System product, you can access the resources in your cloud and you can manage multiple environments from a single system and remote interface.

The PureApplication System product installs and configures your IBM MQ software and manages the application run time by using policies that you define.

Virtual system patterns enable efficient and repeatable deployments of systems that include one or more VM instances and the applications that run on them. You can automate the deployment and eliminate the need to perform multiple time-consuming manual tasks. IBM MQ Virtual System Pattern Type contains IBM MQ software. You can install IBM MQ on a system image as part of a virtual system deployment, which managed by the PureApplication System pattern engine. Included in the virtual system pattern are a number of script packages (compressed files in .zip format) used for automation of the product lifecycle management. Script packages run when the pattern is deployed as a virtual system, when the virtual system is deleted, or whenever you choose to run the scripts manually. You can add these artifacts to a blank template to build a customized virtual system pattern for your IBM MQ environment (Figure 6-2).

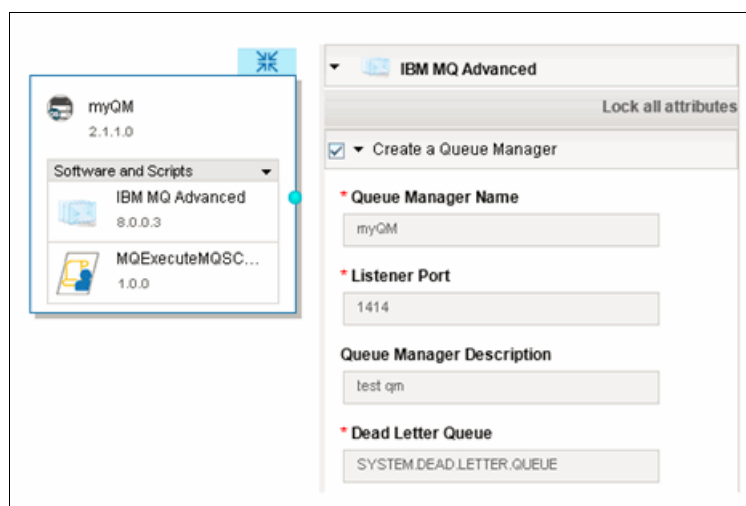


Figure 6-2 A template to build a customized virtual system pattern for an IBM MQ environment

This pattern offers a number of features:

- ▶ The ability to apply a fix pack to deployed pattern instances through the PureApplication System management interfaces.
- ▶ Support for multiple queue managers on the same VM.
- ▶ Queue managers are automatically stopped/started when appropriate by the PureApplication System.
- ▶ Separate file systems can be used for IBM MQ logs, data, and errors.

For more information about the IBM MQ Virtual Pattern Type, see the IBM Knowledge Center at the following website:

https://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.mqp.doc/mqp2000_.htm

6.4 IBM MQ Appliance

IBM MQ Appliance is a hardware appliance that hosts queue managers. IBM MQ Appliances are suited to the IBM MQ hub architecture that is described in Chapter 4, “Infrastructure view of IBM MQ topologies” on page 107.

There are several other benefits that the IBM MQ Appliance provides:

- ▶ The appliance is easy to deploy, has familiar IBM MQ administration interfaces, and supports existing IBM MQ definitions and security.
- ▶ It has easier administration with separate roles and a browser-based IBM MQ Console Tool.
- ▶ High availability is provided with the appliance.
- ▶ There are lower data center costs, space usage, and power requirements.
- ▶ Downtime is reduced by separating applications and middleware.
- ▶ An appliance contains only firmware, which removes the management costs of maintaining the currency and inter-dependencies of the operating system and middleware.

Figure 6-3 shows an IBM MQ hub that is created by using IBM MQ Appliance.

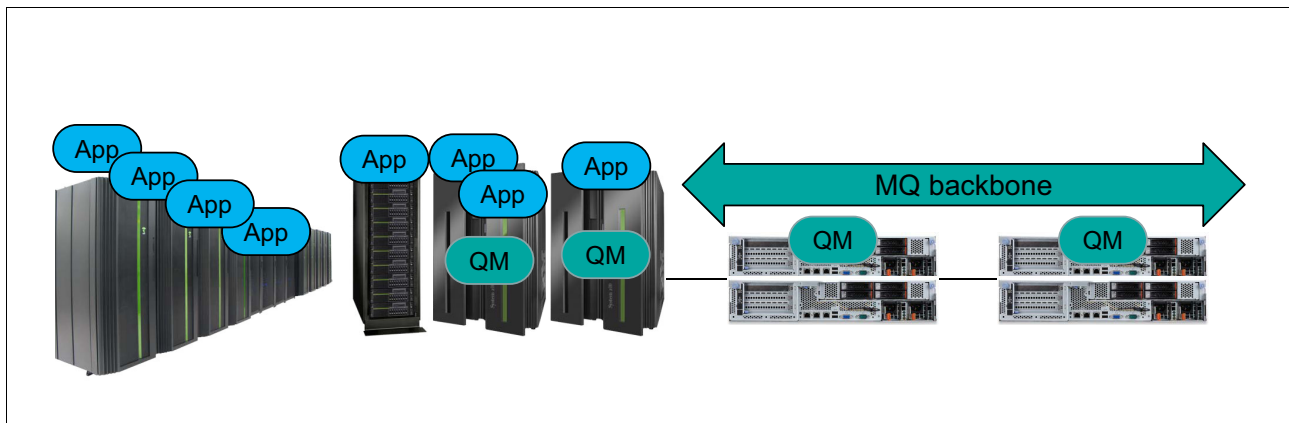


Figure 6-3 An IBM MQ hub created by using IBM MQ Appliance

Here are the key differences between the IBM MQ Appliance and installed IBM MQ software:

- ▶ The hub pattern, where no applications are deployed to the appliance. Applications must connect as remote clients.
- ▶ No user exits can be run on the appliance, such as the channel security feature, Channel Authentication Records (CHLAUTH), and application activity trace.
- ▶ There is appliance-specific high availability technology with no shared file system or shared disk.
- ▶ Authentication and authorization are performed through an onboard or central repository.
- ▶ The command-line interface on the appliance is not a general-purpose shell.

6.4.1 The role of IBM MQ Appliance in the IBM MQ as a service model

The IBM MQ Appliance contains a IBM MQ Version 8.0 image delivered as a state-of-art appliance built by using the latest IBM DataPower® Gateway Appliance hardware and OS. IBM MQ Appliance comes with a pre-tuned configuration that provides maximum performance, which implies the infrastructure team does not have to worry about configuration parameters across servers and operating systems. With the appliance, system administrators can create repeatable deployment patterns with minimum configuration and performance tuning. This approach helps in building IBM MQ as a service from the appliance perspective, which is similar to the standard IBM MQ software installation. System administrators can reuse the same patterns or catalog of services that are created for a standard IBM MQ installation along with the IBM MQ Appliance. IBM MQ Appliance uses administrator concepts and syntax that are similar to the standard IBM MQ software, making it interoperable with either of the deployment models.

6.5 IBM Cloud Orchestrator and IBM MQ integration

The IBM Cloud Orchestrator product helps you with end-to-end service deployment across infrastructure and platform layers. It also provides integrated IT workflow capabilities for process automation and IT governance, resource monitoring, and cost management. The product offers you an extensible approach to integration with existing environments, such as network management tools. It facilitates integration with customer-specific service management processes, such as those defined in the IT Infrastructure Library (ITIL).

By using the IBM Cloud Orchestrator product, you have a consistent, flexible, and automated way of integrating the cloud with customer data center policies, processes, and infrastructures across various IT domains, such as backup, monitoring, and security. Use the intuitive, graphical tool in IBM Cloud Orchestrator to define and implement business rules and IT policies. You can connect the aspects of different domains into a consistent orchestration of automated and manual tasks to achieve your business goals.

The IBM Cloud Orchestrator product is based on a common cloud platform that is shared across the IBM Cloud offerings. This common cloud stack provides a common realization of the core technologies for comprehensive and efficient management of cloud systems.

Figure 6-4 on page 135 shows a high-level IBM Cloud Orchestrator architectural overview.

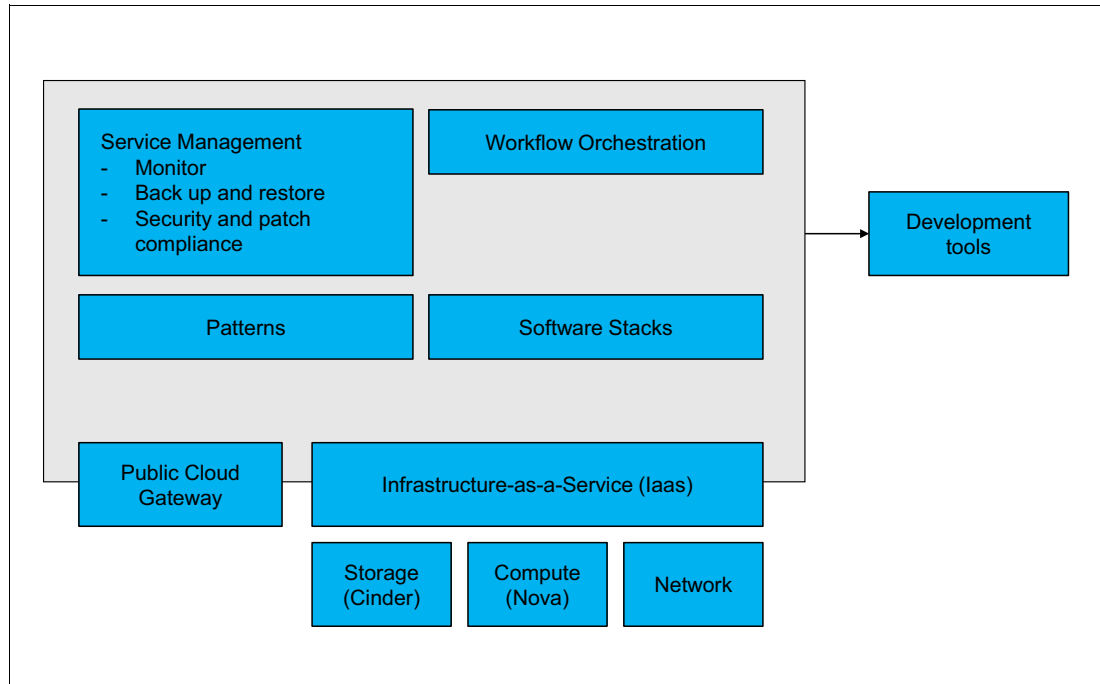


Figure 6-4 High-level product architecture overview of the IBM Cloud Orchestrator product

6.5.1 Delivering pattern-based clouds

The IBM Cloud Orchestrator product is integrated with IBM Workload Deployer, an advanced pattern engine. You can use the IBM Cloud Orchestrator product to design, deploy, and reuse software solutions (virtual systems, virtual applications, and OpenStack Heat infrastructural stacks) that are spread across multiple virtual instances with automatic scalability policies. The IBM Cloud Orchestrator product gives you greater flexibility by removing the need to develop specific interfaces for different cloud services. You can quickly combine and deploy various cloud services onto the cloud infrastructure by assigning the compute, storage, and network resources with an easy-to-use graphical interface. The IBM Cloud Orchestrator product is a private cloud offering that simplifies the task of managing an enterprise-grade cloud. You use a core set of open, source-based technologies to build enterprise class cloud services that can be ported across hybrid cloud environments. You can use the Public Cloud Gateway to communicate with SoftLayer, Amazon Elastic Compute Cloud (EC2), and non-IBM supplied OpenStack.

The IBM Cloud Orchestrator product provides an intuitive self-service user interface, where you can use the Self-Service Catalog to request resources. For example, you can deploy VMs, add volume, or manage key pairs. From this interface, you can also monitor your requests and manage your resources. The IBM Cloud Orchestrator product provides a rich set of predefined offerings in the Self-Service Catalog. In addition, Service Designer can create offerings with IBM Process Designer, and populate the Self-Service Catalog with additional offerings.

6.5.2 Working with IBM Cloud Orchestrator middleware patterns

The IBM Cloud Orchestrator product provides a virtual application pattern that contains components that represent middleware services that are required by the virtual appliance instance. You can connect components in a virtual application pattern to indicate dependencies. Optionally, you can apply a policy to configure middleware services during deployment to configure specific behavior or define a QoS level. Components, links, and policies can have required and optional attributes.

Components, links, and policies are defined by plug-ins. When you create a virtual application pattern, the available components, links, policies, and configuration options are determined by the plug-ins that are included with the selected pattern type.

There are several message components to choose from when building a virtual application pattern.

An existing messaging service

An existing message service component represents a connection to an external messaging system, such as IBM MQ. The presence of a messaging system allows an enterprise application running on WebSphere Application Server to connect to the external messaging resource, such as IBM MQ. The messaging service component represents a connection to an instance of IBM MQ. The component can be configured to create a connection to your IBM MQ installation.

An existing topic

A topic represents a message destination on an external IBM MQ messaging service through which messages are published and subscribed.

An existing queue

An existing message queue is a message queue on an external IBM MQ service from which messages are sent and received.

The IBM Cloud Orchestrator product enables infrastructure teams to build and deploy quickly and easily an IBM MQ queue manager and its resources for applications usage. The IBM Cloud Orchestrator product uses a similar virtual system pattern, as described in 6.3.1, “IBM MQ Virtual System Pattern Type” on page 131. Using these patterns, infrastructure teams can automate the deployment and provisioning of IBM MQ resources in their cloud environment.



Connecting IBM MQ messaging applications to an IBM MQ as a service infrastructure

This chapter describes the various ways that you can configure your IBM MQ applications to connect to an IBM MQ environment to provide *as a service* offerings.

This chapter contains the following sections:

- ▶ 7.1, “Using IBM MQ as a service” on page 138
- ▶ 7.2, “Setting connection properties” on page 138
- ▶ 7.3, “Queue manager connectivity patterns” on page 140

7.1 Using IBM MQ as a service

To use IBM MQ as a service, it is important that the application layer is decoupled as far as possible from the underlying IBM MQ architecture. By using this approach, you can develop applications with minimal knowledge of the system in which they are running, which simplifies their development and speeds up their deployment.

Applications should not be *aware* of the physical resources within the infrastructure, and they should not include any IBM MQ specific endpoint configuration, such as the host name, port, channel, and authentication information. This requirement makes sense when you want to virtualize the infrastructure where queue managers are running because of the likelihood of moving those resources in the future.

Ideally, the IBM MQ resources, such as queue managers, should also be abstracted, which provides flexibility to the infrastructure team to manage the IBM MQ infrastructure without worrying about dependencies from individual application teams. By using this approach, the infrastructure team can dynamically add/remove queue managers or move the queue managers to run on different nodes.

Making the IBM MQ resources portable depends on providing a suitable IBM MQ architecture that supports such a model. If the IBM MQ architecture is highly tuned to different applications, providing the configuration of resources to each of them, it is likely that the applications must be aware of this situation and written so. This limitation increases the cost of any modification to that architecture and reduces the chances of providing a self-service capability. If an *as a service* architecture, such as the IBM MQ hub model that is described in Chapter 4, “Infrastructure view of IBM MQ topologies” on page 107, is used, you might find it easier to auto-generate the connection details, which enable self-service.

A connectivity pattern that lends itself to an infrastructure of this type is to use IBM MQ client-connection (CLNTCONN) channels (over TCP/IP) rather than locally bound applications that must be running in the same system as a queue manager. By using this level of separation, you can separate the provisioning of applications from that of the queue managers themselves.

7.2 Setting connection properties

IBM MQ provides several mechanisms for providing connection details to clients, some more powerful than others, and some more suitable to an as a service infrastructure than others.

7.2.1 Application code

It is possible to code connection details into an application. For as a service deployments, the application should never have hardcoded values, but it can retrieve the details from an external source. This capability can be provided as a component of the self-service layer, for example, program libraries that are provided by the as a service infrastructure team that retrieve the connection properties from a managed source at connection time.

7.2.2 The MQSERVER environment variable

Many client applications (possibly written in C or C++) can use the simple MQSERVER environment variable to obtain a basic level of connection information at run time. This variable can be configured automatically and set as part of the application's runtime environment when provisioned by an as a service infrastructure layer, simplifying the application deployment. However, the MQSERVER environment variable does not support the configuration of secure channels that use Transport Layer Security (TLS), so it is limited in its use.

7.2.3 The connection factory

When using Java Message Service (JMS) or IBM Message Service Client (XMS), setting the connection details in connection factories and storing them in a centrally managed repository, such as Java Naming and Directory Interface (JNDI), provides an excellent way of abstracting and managing the connection properties for each application.

These definitions can be automatically generated as part of the provisioning of new IBM MQ resources by building their generation into processes, such as with the IBM UrbanCode Deploy product.

Connection factories do not suffer from the same limitation as the MQSERVER environment variable, allowing configuration of all aspects of a connection, including TLS on the channel.

7.2.4 Client Channel Definition Table

Connection properties can also be abstracted in to a Client Channel Definition Table (CCDT). A CCDT is a file that is generated through IBM MQ configuration and tools. The file contains all the details that are necessary to connect an application to a queue manager, in a similar way to a connection factory, but can be used across a wider set of programming languages. For this reason, they work well in a provisioned as a service solution.

A CCDT is built by defining the client-facing side of each server-connection (SVRCONN) channel that is used by applications to connect to a queue manager. These channels are CLNTCONN channels. These channels can be defined through the configuration tool of your choice, such as MQSC commands (**runmqsc**).

A CCDT can contain the CLNTCONN definitions for many channels, potentially for many different queue managers. For that reason, it is not necessary or indeed possible in many cases to define the CLNTCONN channels on the same queue manager where the SVRCONN channels are defined.

Until recently, you defined all the CLNTCONN channels on a single queue manager of your choice and obtained the CCDT file from that queue manager's data directory. With IBM MQ V8, it is now possible to run the **runmqsc** command disconnected from any queue manager (by using the **-n** argument) when configuring the CLNTCONN channels. This action generates a CCDT that is local to where you run the tool, rather than using a particular queue manager.

It is now simpler to embed a step into your provisioning layer that automatically generates the correct CCDT to match the SVRCONN channels that are available to an application. For example, you can build an IBM UrbanCode Deploy flow that creates a new queue manager with its SVRCONN channel. A **runmqsc** script can be run as part of the same flow to define a matching CLNTCONN channel and to generate the CCDT file

After you generate the CCDT, the next step is to make this file available to the application. This action can be completed either by distributing a copy of it to the application's local environment or through centrally hosting the file and configuring the application's system to use it. When using connection factories with JMS or XMS, a URL can be specified (the CCDTURL property) to define the location of a remote CCDT file, which is accessible over HTTP. Alternatively, when using the client, the environment can be configured to map a network drive hosting the CCDT file and by setting the MQCHLLIB and MQCHTAB environment variables to point to that file.

Centrally hosting the CCDT files is preferable because it removes the need to redistribute them if they change. In an as a service world, changes to the location and set of queue managers in the system might be frequent.

As already mentioned, a CCDT file can contain any set of queue managers and their channel configurations. This approach enables various connectivity patterns, but it also means that a single file can contain the details of channels that are used by multiple distinct applications, with different applications providing different queue manager names to identify the ones that apply to them.

Either a single CCDT file containing all channel definitions can be created or each application can be provided with their own CCDT file that contains only the channels that apply to them, even with a single entry. Each has its own benefits and drawbacks.

A single CCDT means that there is only a single file to maintain and potentially distribute, simplifying the process. However, every new channel that is added to your as a service messaging infrastructure results in a change to that single file, which means it is necessary to rebuild frequently and potentially redistribute the file to all the applications, whether the change is of any importance or not. For large systems, a single large CCDT might slow down the IBM MQ connectivity performance, although minimally.

Creating a separate CCDT for each application, containing only the channels that are necessary for it, means that there might be many files to maintain and manage. However, any change to support a new queue manager or channel affects only a small subset of applications, reducing the need to redistribute the file.

If you integrated the automatic generation of the CCDT into a provisioning system, then separate CCDT files are probably better because much of the complexity of the management is hidden from you.

7.3 Queue manager connectivity patterns

The mechanisms that are described in 7.2, "Setting connection properties" on page 138 defines how an application obtains its connection properties, but it does not define to what it connects. The *what* might be a single network endpoint, a set of endpoints, or queue managers.

Connecting an application to a single endpoint is simple, but there are multiple reasons why limiting an application might be inadequate:

- ▶ If that queue manager is highly available, for example, as a multi-instance queue manager, it might be active at one of multiple network addresses.
- ▶ Even if the queue manager is highly available, if there is a failover, there might be a period (no matter how brief) when it cannot be connected to. In this situation, it might be preferable for the application to connect to an alternative queue manager while the other one is restarting.

- If the messaging infrastructure supports large-scale applications or many applications that share an infrastructure, rather than directing each application to a specific queue manager, it might be preferable to workload balance those applications and their connections across a set of queue managers.

7.3.1 Mechanisms to achieve workload balancing and availability when connecting

Many of the methods for setting connection properties that are described in 7.2, “Setting connection properties” on page 138 can be used to support workload balancing and high availability.

Comma-separated connection list

The simplest way of removing the restriction of a connection to a single address is to use a comma-separated list of host names and their ports. This list can be set in connection factories or in the MQSERVER environment variable.

This approach can be used simply to find a single queue manager that might be running at one of multiple possible network locations (supporting the situation that is described in the first bullet in 7.3, “Queue manager connectivity patterns” on page 140).

It is also possible to use this mechanism to support the situation that is described in the second and third bullets of 7.3, “Queue manager connectivity patterns” on page 140. However, there are certain limitations:

- If the endpoint list defines the location of different queue managers, they must all have the same named SVRCONN channels defined, each with a matching security configuration.
- Only a list of network locations is provided, not channel names and other information.
- The application itself must not specify a queue manager name. This name must be left blank or a successful connection is made only to the queue manager of that specific name.

The list is used by the client starting with the first location in the list and working down the list until it successfully connects to a queue manager. Two things to consider:

- The further down the list the client must go, the longer the connection takes, so the list should be ordered with the most likely available locations first.
- The first available queue manager in the list receives all the connections, not making it a good match for workload balancing. It is possible to provide lists in different orders to different applications to have some level of balancing of connections, but there are better mechanisms.

An example of an MQSERVER variable that uses this mechanism is shown in Example 7-1.

Example 7-1 MQSERVER variable

```
export MQSERVER=SVRCONN.CHANNEL/TCP/machine1.ibm.com(1414),machine2.ibm.com(1414)
```

Client Channel Definition Tables details

CCDTs are suitable for an as a service solution because they abstract the low-level details of an IBM MQ endpoint (an SVRCONN channel) from the application itself. A CCDT file can contain multiple entries, for different channels on different queue managers.

When a CCDT is configured through the definition of the CLNTCONN channels, each CLNTCONN channel is associated with a queue manager name. For an application that is configured to use a CCDT (see 7.2.4, “Client Channel Definition Table” on page 139), when it specifies a specific queue manager name on the connection call, the matching channel entry in the CCDT is used.

However, it is possible to define multiple entries within a CCDT with the same queue manager name. The queue manager name that is provided does not need to match the name of the queue manager to which the CLNTCONN channel points. It is used as a way to group certain channels and to allow an application to identify that group. This collection is a queue manager group, and an application identifies that it wants to connect to any member of the group by specifying the queue manager name that is set for all the channels, prefixed with an asterisk character (*), for example, '*QMGR_GROUPA'.

By defining multiple channels in a single queue manager group within a CCDT, it is possible to satisfy the queue manager connectivity patterns, whether it is for availability or for workload balancing. The CCDT can be configured so that channels in a group are either sequentially tried (for availability), or randomly tried based on specified weightings (for workload balancing). For more information about this approach, go to the IBM Knowledge Center at the following website:

http://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.dev.doc/q027490_.htm

Such capabilities make CCDTs ideal as an abstraction layer for an as a service solution, where applications connect to a group of queue managers rather than any one in particular. The IBM MQ infrastructure can change without requiring changes to the applications themselves. For example, when an application grows in scale, it is possible to provision an additional queue manager and add it to the queue manager group in the CCDT being used by the application.

You can adopt the asterisk (*) group name notation even when a single queue manager is used to allow the underlying queue manager's name to be abstracted from the application. This approach is a preferred practice in an as a service environment, and you have greater freedom in naming your underlying IBM MQ resources.

Network load balancers

Another method for providing abstraction of the underlying IBM MQ infrastructure in addition to availability and workload balancing is by using *network load balancers*.

Load balancing is a technique that you can use to divide the incoming requests across multiple available resources. Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid the overloading of any single resource. Using multiple components with load balancing instead of a single component might increase reliability and availability through redundancy. Load balancing usually involves dedicated software or hardware, such as a multilayer switch or a Domain Name System (DNS) server process.

External load balancers (such as F5 or Alteo) can be configured with IBM MQ to federate and workload balance the incoming requests across multiple IBM MQ queue managers. The load balancers provide high availability, Secure Sockets Layer (SSL) offloading, and Transmission Control Protocol (TCP) optimizations to the IBM MQ solutions. A prime way of using load balancers is in front of IBM MQ for connection balancing. Although IBM MQ already provides connection balancing through the CCDT, using external load balancers can bring certain advantages.

When using a load balancer, the client is configured to use a virtual server address when connecting to a queue manager, and the load balancer can be made aware of the existence of the queue manager and routes the connect request to one of the available queue manager instances.

A useful pattern is to configure queue managers with multiple listeners. One queue manager listens on an address and port that provide the queue manager resource to the network load balancer. The other queue manager listens on an address and port that provide direct connectivity to a particular queue manager for the applications that need it (for example, administrative tools and point-to-point channels). Remote connections can then be configured to connect by using either the load balancing address or the direct address.

There are two different modes on how the load balancers can be configured to integrate with IBM MQ solutions.

In mode 1 (Figure 7-1), the load balancers are configured to include information about the available queue managers. The load balancers continuously monitor the availability of the queue managers and their states. If the queue manager fails or is unavailable, the entry is automatically removed from the load balancers repository. The new client requests are automatically balanced with the available list of queue managers.

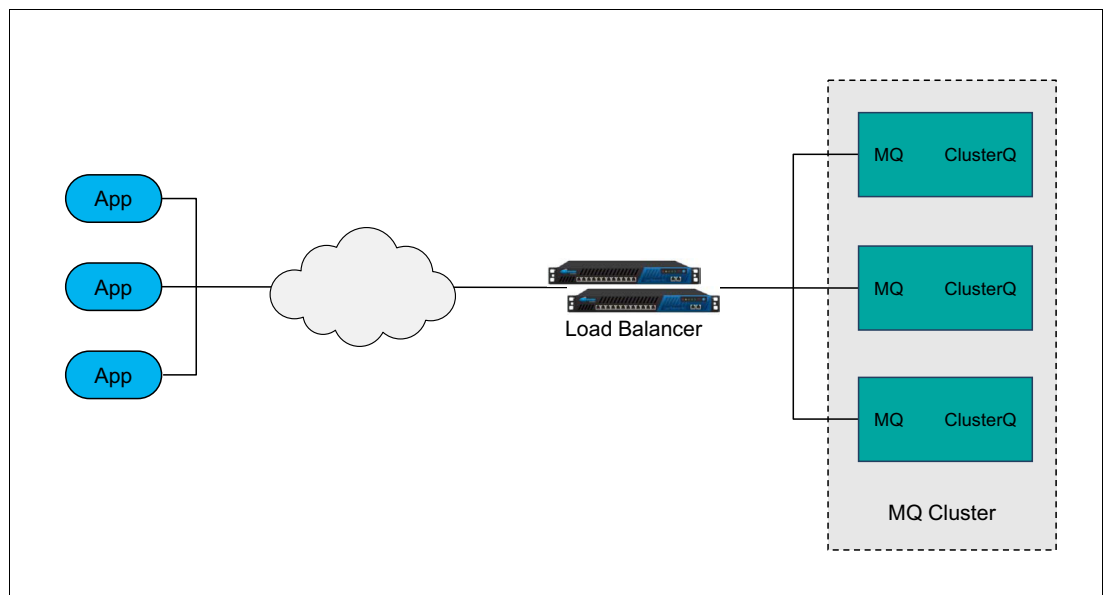


Figure 7-1 Mode 1

In mode 2 (Figure 7-2), the load balancers do not directly interact with IBM MQ, but connect to highly available IBM DataPower Gateway appliances. The load balancers receive all the incoming requests and forward these requests to one of the available IBM DataPower Gateway services. The appliance performs validations and then sends the requests to one of the available queue managers.

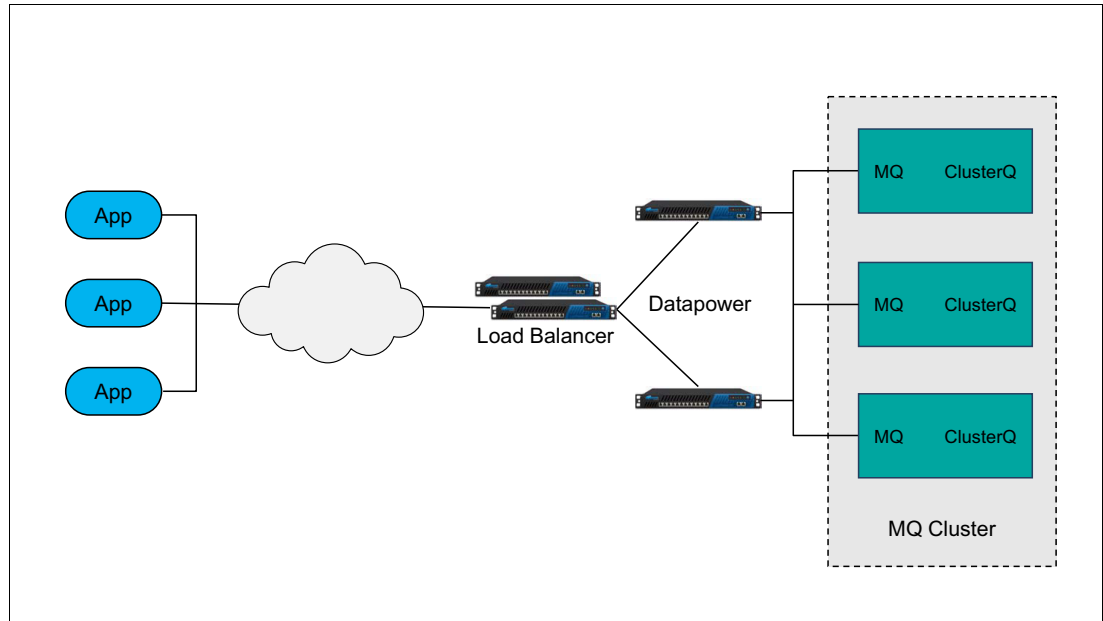


Figure 7-2 Mode 2

Custom logic

Despite all the mechanisms that are available, it is possible that your system requires logic to achieve the exact connectivity pattern that you need. It also might be necessary to overcome some of the considerations.

Custom logic can be provided in various forms, either as a wrapper layer to the IBM MQ client logic that performs the connection or in some environments as an IBM MQ exit.

IBM developerWorks® has a three-part series. The first part is *A flexible and scalable WebSphere MQ topology pattern: Part 1: Building a multi-purpose WebSphere MQ infrastructure with scalability and high availability*. The series covers an example of a wrapper, and it can be found at the following website:

http://www.ibm.com/developerworks/views/websphere/libraryview.jsp?search_by=A+flexible+and+scalable+WebSphere+MQ+topology+pattern

For applications that are written in C, it is possible to write and deploy a preconnect exit that is started every time that an application connects to IBM MQ. You then can inject your own custom logic to control how and what an application connects to. For example, you can retrieve the connection details from a central repository or define your own workload balancing logic. For more information about this concept, see the IBM Knowledge Center at the following website:

http://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.dev.doc/q028260_.htm

7.3.2 Managing high availability at the application layer

In some situations, infrastructure teams provide a layer of abstraction between the IBM MQ resources and the applications. The application connectivity is always routed through this abstraction layer, which acts as the *service interface* to provide continuous availability and high availability of IBM MQ resources to the application teams.

A typical use is when the destination is partitioned across multiple queue managers. To achieve continuous and high availability, the infrastructure team creates a destination with the same name across multiple queue managers or adds a queue as a clustered queue in a queue manager cluster. It might be a design principle that in a given IBM MQ hub (that is, a farm of queue managers working together), every destination must be present across all the queue managers in the hub.

The service interface knows all the queue managers that are present in a given hub. When the service is started, the service creates a connection to each queue manager in the hub. For example, if there are 10 queue managers in the hub, the service interface creates a connection to each queue manager in the hub.

The service interface then manages the lifecycle of these connections. If the connection to one of the queue managers fails, it re-creates the connection to that queue manager. If the queue manager fails or is down for a long time, the service interface removes the connection from its list and re-creates the connection after the queue manager restarts. This solution can be achieved by using the client-triggering capability that IBM MQ provides. For more information, see the IBM Knowledge Center at the following website:

https://www.ibm.com/support/knowledgecenter/SSFKSJ_7.5.0/com.ibm.mq.dev.doc/q026910_.htm

Figure 7-3 shows a service interface that creates and holds connections to each queue manager.

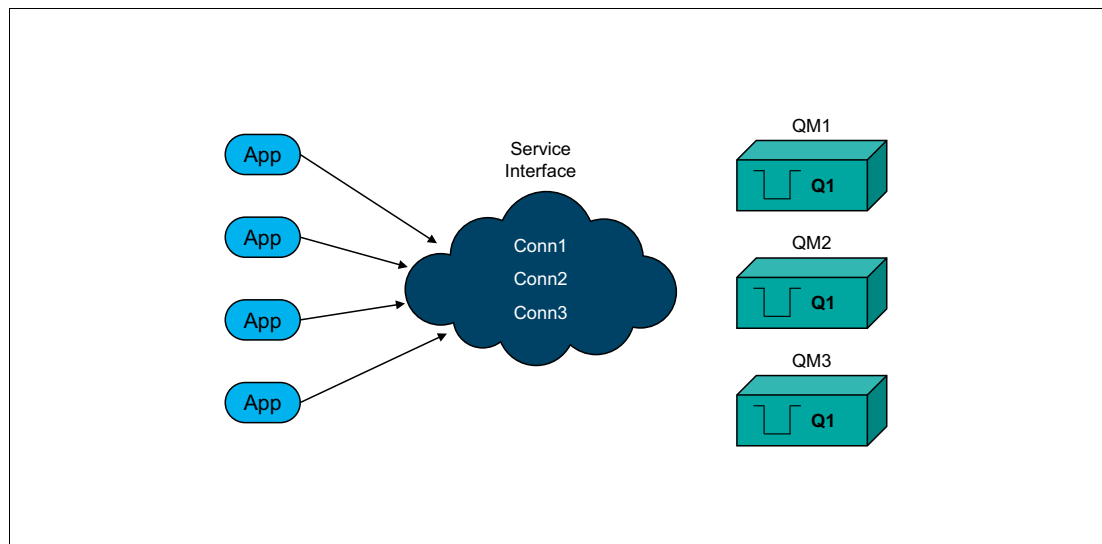


Figure 7-3 Service interface that creates and holds connections to each queue manager

In Figure 7-3 on page 145, the applications do not directly connect to a queue manager, but rather call or start an API that is exposed by the service interface. The service interface is responsible for providing an appropriate connection to the application. This service interface can implement several logics to provide a suitable connection, for example, a round-robin or least-recently-used connection. Alternatively, the service interface can start a destination monitoring service or queue manager monitoring service to determine which queue manager is busy or free and then decide to return a particular connection to the application.

7.3.3 Considerations when configuring workload balancing and availability

Building high availability and workload balancing into an as a service infrastructure has its obvious benefits, but it can pose problems for certain application types and patterns. Any developer of an as a service infrastructure should be aware of these considerations and ensure that applications are developed.

Global transactions

When using global (XA) transactions, the transaction coordinator must know which queue manager an application was connected to in order to provide transaction recovery support. When mechanisms, such as CCDTs and network load balancers, are used to provide multiple potential target queue managers, the ability to resolve correctly transactions is lost. This situation can result in incorrect recovery of transactions in failure scenarios.

For this reason, custom logic might be a more suitable mechanism to use when global transactions are required in combination with workload balancing of high availability.

Queue manager names

An application should not rely on hardcoding a queue manager name within the application logic. Queue manager groups regarding CCDTs is one way to remove an affinity within an application to a particular queue manager name.

Applications should not be setting queue manager names elsewhere in their logic, for example, specifying a queue manager ReplyToQMGr in a C language application's message queuing message descriptor (MQMD). In practice, it should not be necessary for an application to set queue manager names within an as a service messaging system. The applications should be designed to expect any reply messages to be sent to a reply queue on the queue manager to which they connect, not a specific queue manager in the system. By leaving the value blank in the application, the currently connected queue manager uses their own name, ensuring reply messages are routed back to the correct queue manager.

Durable subscriptions

When using publish/subscribe, an application might dynamically create a durable subscription (if one does not exist) on the queue manager to which it is connected. JMS is one API that has such a feature. If the application is configured to connect to a set of different queue managers for workload balancing or availability reasons, there is a strong possibility that it might create multiple durable subscriptions, one for each queue manager that it connects to over time. This action poses multiple problems. This action results in multiple copies of the same subscription, with all except one of them building up a growing number of copies of publications, potentially consuming valuable resources. The application might see duplicate copies of the same publication or miss some altogether as it moves between queue managers, which is probably not the wanted behavior for the application; otherwise, a non-durable subscription would have been used.

When using durable subscriptions, the application might have an affinity to a particular queue manager, so it is hard to move that application between queue managers. Instead, every effort must be made to make that queue manager highly available by using mechanisms, such as multi-instance queue managers or the IBM MQ Appliance high availability support.

JMS and XMS

JMS and XMS support multiple sessions that are managed by a single connection. However, each connection and session creates a separate IBM MQ client/server conversation. To support the correct semantics of JMS and XMS, each of the session conversations must be directed to the same queue manager as its managing connection. Therefore, except for a few limited situations, workload balancing mechanisms such as network load balancers and connection lists cannot be used.

When using a network load balancer, it is possible for the load balancer to route the individual sessions (that is, the underlying IBM MQ connections) to different network endpoints and queue managers. This approach breaks the semantics of JMS and XMS and makes the use of network load balancers with JMS or XMS difficult.

One way to alleviate such a problem is to rely on the shared conversation capability of IBM MQ channels. If a channel supports multiple IBM MQ connections on a single instance of the SVRCONN channel, then it might be possible for the separate IBM MQ connections that represent the connection and sessions from a JMS or XMS application to use the same channel instance and a TCP/IP connection. This approach prevents the network load balancer from routing the connections and sessions to separate queue managers.

However, there is no control that ensures that the correct connection and sessions use the same TCP/IP connection. It depends on the number of sessions within a single application process and the shared conversation setting on the channel. For this reason, it is suitable only if the applications are running in separate processes (for example, not in an application server) and that the combined number of connections and sessions is less than the shared conversation setting on the channel.

Ultimately, this approach results in a fragile setup that is not suited to an as a service model where you are trying to decouple the application logic from the infrastructure, so this approach should be avoided.

Another point to consider when using load balancers with JMS in a Java Platform, Enterprise Edition environment is that you should load balance only outbound connection factory connections (known as *outbound* even if you receive calls to gather responses in request/reply). Activation specifications for message-driven beans (MDB) message listeners should bypass F5 and connect directly to the queue managers. If you have multiple queue managers, then you should configure multiple endpoints, one per queue manager. Load balancers can cause the child sessions of an MDB connection to connect to a different queue manager than the one to which the parent connects. Also, it does not make sense to load balance the inbound connections because you can end up with stranded messages if you have queue instances to which no application instances are listening.

7.3.4 Comparison of mechanisms

This chapter described multiple ways of achieving multi-queue manager client attachment. Although each of the options provides a means to virtualize application connectivity to IBM MQ, you should understand the various merits and demerits of using each of these options. Table 7-1 summarizes the differences when using these options.

Table 7-1 Summary of options

Option	Comma-separated list	CCDT	Load balancer	Custom logic
Scale of code change required for an existing application connecting to a single queue manager (QM).	Good: <ul style="list-style-type: none"> ► MQCONN("QMNAME") to MQCONN("**QMNAME"). ► QMName might be in JNDI configuration for Java EE applications. Otherwise, it requires a one-character code change. 			Poor: Replace the existing JMS/MQI connection logic with a code stub.
Support for different WLM strategies.	Poor: Prioritized only.	Neutral: Prioritized and random.	Good: Any, including per-connect round robin.	Good: Any, including per-connect round robin.
Performance impact while the primary QM is down.	Poor: Always tries the first in the list.	Good: Remembers the last good QM.	Good: Port monitoring avoids bad QMs.	Good: Can remember the last good QM and retry intelligently.
XA transaction support	Poor: The transaction manager must store recovery information that reconnects it to the same QM resource. An MQCONN that resolves to different QMs generally invalidates this situation. For example, in Java EE, a single connection factory should resolve to a single QM when using XA.			Good: A code stub can meet the XA transaction manager's requirements, for example, multiple connection factories.
Connection rebalancing on failback (for example, when a QM restarts after a failure/planned outage, how long does it take until the applications can use it again).	Poor: Connection pooling in Java EE holds on to connections indefinitely, unless connections are configured with an aged timeout. Using an aged timeout might drive exceptions in some cases. An aged timeout also introduces a performance impact during normal operation. Conversation sharing might need to be disabled (SHARCNV=1) with an aged timeout to ensure reconnects always establish a new socket. The "remembers last good" CCDT behavior might also delay failback.			Good: A code stub can handle failback flexibly with little or no performance impact.
Administration flexibility to hide infrastructure changes from applications.	Poor: DNS only.	Neutral: DNS and shared file system/CCDT file push.	Good: Dynamic Virtual IP address (VIP).	Neutral: DNS or single QMGR CCDT entries.



IBM UrbanCode Deploy initial setup

To simplify the creation of IBM MQ resources, use IBM UrbanCode Deploy to automate and create services for IBM MQ resources. This appendix describes the steps that are necessary to use IBM UrbanCode Deploy to create IBM MQ resources.

This appendix includes the following sections:

- ▶ “Getting started” on page 150
- ▶ “IBM UrbanCode Deploy concepts” on page 150
- ▶ “Installing the IBM UrbanCode Deploy server components” on page 151
- ▶ “Installing the IBM UrbanCode Deploy agent” on page 157
- ▶ “Loading the IBM MQ plug-in for IBM UrbanCode Deploy” on page 161
- ▶ “Application deployment with IBM UrbanCode Deploy” on page 163

Getting started

Before you use the information in this appendix, review the following points:

- ▶ All the examples in this appendix are based on the Microsoft Windows operating system.
- ▶ The examples were developed by using the evaluation version of the IBM UrbanCode Deploy software. You can download the IBM UrbanCode Deploy evaluation code from the following website:

https://www.ibm.com/marketing/iwm/iwm/web/preLogin.do?source=RATLe-UCDeploy.install.doc/topics/server_install_interactive.html

- ▶ The examples use the IBM MQ plug-in for IBM UrbanCode Deploy. You can download the plug-in from the following website:

<https://developer.ibm.com/urbandcode/plugin/websphere-mq/>

- ▶ In the examples in this appendix, both the IBM UrbanCode Deploy server and the IBM UrbanCode Deploy agent run on the same machine.

- ▶ IBM MQ must be preinstalled on the machine where the IBM UrbanCode Deploy agent runs.

- ▶ Before you start, ensure that the JAVA_HOME system variable is set to <Java-InstallPath>\jre.

- ▶ For IBM UrbanCode Deploy general information, see the IBM UrbanCode Deploy IBM Knowledge Center at the following website:

http://www.ibm.com/support/knowledgecenter/SS4GSP_6.1.3/com.ibm.udeploy.doc/ucd_version_welcome.html

- ▶ The IBM UrbanCode Deploy server runs as user admin in the examples. A Windows user that is called admin with password admin was created for this purpose.
- ▶ IBM UrbanCode Deploy requires Java 1.6 or above.
- ▶ All the examples in this paper use all the default components, such as the Apache Derby database, that are included with the IBM UrbanCode Deploy evaluation copy. If you want to use non-default components, see the IBM UrbanCode Deploy IBM Knowledge Center for instructions.

IBM UrbanCode Deploy concepts

To understand IBM UrbanCode Deploy, view the YouTube video *Basic concepts and overview of IBM UrbanCode Deploy 6.0*, found at the following website:

<https://www.youtube.com/watch?v=oVv6a8mUwXo>

For more information about IBM UrbanCode Deploy, see the IBM Knowledge Center, found at the following website:

http://www.ibm.com/support/knowledgecenter/SS4GSP_6.1.3/com.ibm.udeploy.doc/ucd_version_welcome.html

Installing the IBM UrbanCode Deploy server components

The section provides detailed information about installing the IBM UrbanCode Deploy server components. For more information about installing the server in interactive mode, see IBM Knowledge Center at the following address:

http://www.ibm.com/support/knowledgecenter/SS4GSP_6.1.3/com.ibm.udeploy.install.doc/topics/server_install_interactive.html

To install the IBM UrbanCode Deploy server components, complete the following steps:

1. Download the IBM UrbanCode Deploy evaluation code from the following website:

https://www.ibm.com/marketing/iwm/iwm/web/preLogin.do?source=RATLe-UCDeploy-EVAL&S_CMP=web_dw_rt_sw

Download the compressed file that is named URBANCODE_DEPLOY_6.x.x.x_EN_EVAL.zip (.x.x.x denotes the product version) in your Downloads folder.

2. Extract the contents of the file into the ucd-server-install directory.
3. Open a Command Prompt window and make sure that the JAVA_HOME system variable is set to <Java-InstallPath>\jre.
4. Go to the ucd-server-install directory.
5. Run **install-server.bat**. The IBM UrbanCode Deploy server installation starts.
6. Specify the information as the installation program prompts you. Accept the default values (displayed within brackets) by pressing Enter.

Note: Accept the defaults for the prompts that provide default values during the installation.

Example A-1 shows the console of the IBM UrbanCode Deploy server installation example.

Example A-1 IBM UrbanCode Deploy server installation in interactive mode

```
C:\Work\MQaaS\ibm-ucd-install>install-server.bat
```

```
Buildfile: install.with.groovy.xml
```

```
compile:
```

```
[groovyc] Compiling 1 source file to C:\Work\MQaaS\ibm-ucd-install\compiled
```

```
install:
```

```
[unzip] Expanding: C:\Work\MQaaS\ibm-ucd-install\conf.zip into
```

```
C:\Users\admin\AppData\Local\Temp\install-3856131630055644552.tmp
```

```
[echo] Found sub-installer UCDeployInstaller.groovy
```

```
International License Agreement for Evaluation of Programs
```

```
Part 1 - General Terms
```

```
BY DOWNLOADING, INSTALLING, COPYING, ACCESSING, CLICKING ON AN "ACCEPT" BUTTON, OR OTHERWISE USING THE PROGRAM, LICENSEE AGREES TO THE TERMS OF THIS AGREEMENT. IF YOU ARE ACCEPTING THESE TERMS ON BEHALF OF LICENSEE, YOU REPRESENT AND WARRANT THAT YOU HAVE FULL AUTHORITY TO BIND LICENSEE TO THESE TERMS. IF YOU DO NOT AGREE TO THESE TERMS,
```

```
* DO NOT DOWNLOAD, INSTALL, COPY, ACCESS, CLICK ON AN "ACCEPT" BUTTON, OR USE THE PROGRAM; AND
```

```
* PROMPTLY RETURN THE UNUSED MEDIA AND DOCUMENTATION TO THE PARTY FROM WHOM IT WAS OBTAINED. IF THE PROGRAM WAS DOWNLOADED, DESTROY ALL COPIES OF THE PROGRAM.
```

1. Definitions

"Authorized Use" - the specified level at which Licensee is authorized to execute or run the Program. That level may be measured by number of users, millions of service units ("MSUs"), Processor Value Units ("PVUs"), or other level of use specified by IBM.

"IBM" - International Business Machines Corporation or one of its subsidiaries.

"License Information" ("LI") - a document that provides information and any additional terms specific to a Program. The Program's LI can be found in the Program's directory, by the use of a system command, or as a booklet included with the Program.

"Program" - the following, including the original and all whole or partial copies: 1) machine-readable instructions and data, 2) components, files, and

(1/36) "Enter" for the next page, or "F" to print the full license: F

Complete License Agreement will be printed as Text on Console.....

[echo] Do you accept the license? [y,n]

y

[echo]

[echo] Installing IBM UrbanCode Deploy version 6.1.1.5.672018

[echo] Enter the directory where the IBM UrbanCode Deploy should be installed.

[Default: C:\Program Files\ibm-ucd\server]

Press <enter> ==>Use Default

[echo] The specified directory does not exist. Do you want to create it? Y,n

[Default: Y]

Press <enter> ==>Use Default

[echo]

[echo] Installing IBM UrbanCode Deploy to: C:\Program Files\ibm-ucd\server

[echo] Enter the home directory of the JRE/JDK used to run the server.

[Default: C:\Program Files\IBM\java\jre]

Press <enter> ==>Use Default

[echo] JVM Version detected: 1.7.0

[echo] JAVA_HOME: C:\Program Files\IBM\java\jre

[echo] Will this server be used as a node in a high-availability cluster? y,N [Default: N]

Press <enter> ==>Use Default

[echo] Where should the server store application data such as logs, plug-ins , and keystores?

[Default: C:\Program Files\ibm-ucd\server\appdata]

Press <enter> ==>Use Default

[echo] The specified directory for application data (C:\Program Files\ibm-ucd\server\appdata) doesn't exist. Do you want to create it? Y,n [Default Y]

Press <enter> ==>Use Default

[echo] What host name will users access the web UI at? [Default: MyComputerNamw]

Enter the value as "localhost" or any host name and

Press <enter>

[echo] Do you want the web UI to always use secure connections by using SSL? Y,n [Default: Y]

Enter 'n' ==> For the demo, we will not use SSL.

Press <enter>

[echo] Enter the port on which the web UI should listen for HTTP requests. [Default: 8080]

Press <enter> ==>Use Default

[echo] Enter the initial password for the admin user.
Enter your preferred password

Press <enter>

[echo] Type password again.
Enter your preferred password again

Press <enter>

[echo] Enter the port to use for agent communication. [Default: 7918]

Press <enter> ==>Use Default

[echo] Do you want the Server and Agent communication to require mutual authentication? This requires a manual key exchange between the server and each agent. See the documentation for more details. y,N
[Default: N]

Press <enter> ==>Use Default

[echo] Enter the port and host name of a Rational License Key Server containing product licenses for IBM UrbanCode Deploy, in the form of port@hostname. (for example, 27000@licenses.example.com)
Alternatively, you may leave this blank to begin a 60-day evaluation period. [Default: none]

Press <enter> ==>Use Default

[echo] When installing a server as a part of an existing cluster or when using a pre-populated database, it is not necessary to create the database schema.

This step must be performed when installing a stand-alone server to a fresh database or when installing the first server in a cluster.

[echo] Create database schema? (For high availability servers, this should be done only for the first server in the cluster) Y,n [Default: Y]

Press <enter> ==>Use Default

[echo] The following database types are supported: derby, mysql, oracle, sqlserver, postgres, db2.
[echo] Enter the database type to use. [Default: derby]

Press <enter> ==>Use Default

[echo] Enter the database user name. [Default: ibm_ucs]

Press <enter> ==>Use Default

[echo] Enter the database password. [Default: password]

Press <enter> ==>Use Default

[mkdir] Created dir: C:\Program Files\ibm-ucs\server\bin
[mkdir] Created dir: C:\Program Files\ibm-ucs\server\conf\server
[mkdir] Created dir: C:\Program Files\ibm-ucs\server\endorsed

```

[mkdir] Created dir: C:\Program Files\ibm-ucd\server\lib
[mkdir] Created dir: C:\Program Files\ibm-ucd\server\licenses
[mkdir] Created dir: C:\Program Files\ibm-ucd\server\native
[mkdir] Created dir: C:\Program Files\ibm-ucd\server\extensions
[mkdir] Created dir: C:\Program Files\ibm-ucd\server\var\log
[mkdir] Created dir: C:\Program Files\ibm-ucd\server\var\temp
[mkdir] Created dir: C:\Program Files\ibm-ucd\server\appdata\patches
[mkdir] Created dir: C:\Program Files\ibm-ucd\server\appdata\var
[mkdir] Created dir: C:\Program Files\ibm-ucd\server\appdata\conf\server
[copy] Copying 1 file to C:\Program Files\ibm-ucd\server\conf
[sync] Copying 201 files to C:\Program Files\ibm-ucd\server\lib
[sync] Copying 4 files to C:\Program Files\ibm-ucd\server\native
[sync] Copying 8722 files to C:\Program Files\ibm-ucd\server\opt\tomcat
[copy] Copying 1 file to C:\Program Files\ibm-ucd\server\bin
[copy] Copying 1 file to C:\Program Files\ibm-ucd\server\opt\tomcat\conf
[mkdir] Created dir: C:\Program Files\ibm-ucd\server\bin\service
[copy] Copying 6 files to C:\Program Files\ibm-ucd\server\bin
[copy] Copying 1 file to C:\Program Files\ibm-ucd\server\bin
[copy] Copying 1 file to C:\Program Files\ibm-ucd\server\bin
[copy] Copying 3 files to C:\Program Files\ibm-ucd\server\bin
[copy] Copying 2 files to C:\Program Files\ibm-ucd\server\bin\service
[unzip] Expanding: C:\Work\MQaaS\ibm-ucd-install\udconf.zip into
C:\Users\admin\AppData\Local\Temp\install-3385571869548379440.tmp
[copy] Copying 18 files to C:\Program Files\ibm-ucd\server\conf
[copy] Copying 2 files to C:\Program Files\ibm-ucd\server\appdata\conf
[copy] Copying 8 files to C:\Program Files\ibm-ucd\server
[copy] Copying 27 files to C:\Program Files\ibm-ucd\server\appdata
[copy] Copying 4 files to C:\Program Files\ibm-ucd\server\bin
[propertyfile] Updating property file: C:\Program Files\ibm-ucd\server\conf\server\installed.properties
[echo] OS: windows
[echo] Architecture: x64
[delete] Deleting directory C:\Users\admin\AppData\Local\Temp\install-7744486556928995982.tmp
[echo] Creating new AES encryption key.
[propertyfile] Updating property file: C:\Program Files\ibm-ucd\server\conf\server\installed.properties
[echo] Starting embedded database ...
[echo]     waiting for db to start - 60 seconds remaining
[echo]     waiting for db to start - 57 seconds remaining
[echo] Database Started
[echo] Creating IBM UrbanCode Deploy Database Schema ...
[sql] Executing resource: C:\Work\MQaaS\ibm-ucd-install\database\deploy\derby\ud-schema.ddl
[sql] 78 of 78 SQL statements executed successfully
[sql] Executing resource: C:\Work\MQaaS\ibm-ucd-install\database\deploy\ud-foreign-keys.ddl
[sql] 231 of 231 SQL statements executed successfully
[echo] Creating Versioned Configuration Database Schema ...
[sql] Executing resource: C:\Work\MQaaS\ibm-ucd-install\database\vc\derby\vc-schema.ddl
[sql] 8 of 8 SQL statements executed successfully
[echo] Creating Versioned Configuration Foreign Keys ...
[sql] Executing resource: C:\Work\MQaaS\ibm-ucd-install\database\vc\vc-foreign-keys.ddl
[sql] 13 of 13 SQL statements executed successfully
[echo] Seeding Versioned Configuration Database ...
[sql] Executing resource: C:\Work\MQaaS\ibm-ucd-install\database\vc\vc-seed-data.sql
[sql] 2 of 2 SQL statements executed successfully
[echo] Creating Property Database Schema ...
[sql] Executing resource:
C:\Work\MQaaS\ibm-ucd-install\database\property-sheets\derby\property-schema.ddl
[sql] 10 of 10 SQL statements executed successfully
[echo] Creating Property Database Foreign Keys ...
[sql] Executing resource: C:\Work\MQaaS\ibm-ucd-install\database\property-sheets\ps-foreign-keys.ddl
[sql] 21 of 21 SQL statements executed successfully
[echo] Seeding Property Database ...

```

```

[sql] Executing resource: C:\Work\MQaaS\ibm-ucd-install\database\property-sheets\ps-seed-data.sql
[sql] 1 of 1 SQL statements executed successfully
[echo] Creating Inventory Database Schema ...
[sql] Executing resource: C:\Work\MQaaS\ibm-ucd-install\database\inventory\derby\inv-schema.ddl
[sql] 5 of 5 SQL statements executed successfully
[echo] Creating Inventory Foreign Keys ...
[sql] Executing resource: C:\Work\MQaaS\ibm-ucd-install\database\inventory\inv-foreign-keys.ddl
[sql] 14 of 14 SQL statements executed successfully
[echo] Seeding Inventory Database ...
[sql] Executing resource: C:\Work\MQaaS\ibm-ucd-install\database\inventory\inv-seed-data.sql
[sql] 1 of 1 SQL statements executed successfully
[echo] Creating Workflow Engine Database Schema ...
[sql] Executing resource: C:\Work\MQaaS\ibm-ucd-install\database\workflow\derby\wf-schema.ddl
[sql] 6 of 6 SQL statements executed successfully
[echo] Creating Workflow Engine Foreign Keys ...
[sql] Executing resource: C:\Work\MQaaS\ibm-ucd-install\database\workflow\wf-foreign-keys.ddl
[sql] 7 of 7 SQL statements executed successfully
[echo] Creating Security Database Schema ...
[sql] Executing resource: C:\Work\MQaaS\ibm-ucd-install\database\security\derby\security-schema.ddl
[sql] 27 of 27 SQL statements executed successfully
[echo] Creating Security Database Foreign Keys ...
[sql] Executing resource: C:\Work\MQaaS\ibm-ucd-install\database\security\foreign-keys.ddl
[sql] 27 of 27 SQL statements executed successfully
[echo] Creating Security Database Indexes ...
[sql] Executing resource: C:\Work\MQaaS\ibm-ucd-install\database\security\indexes.ddl
[sql] 23 of 23 SQL statements executed successfully
[echo] Seeding Workflow Engine Database ...
[sql] Executing resource: C:\Work\MQaaS\ibm-ucd-install\database\workflow\wf-seed-data.sql
[sql] 1 of 1 SQL statements executed successfully
[echo] Creating Agent Topology Database Schema ...
[sql] Executing resource: C:\Work\MQaaS\ibm-ucd-install\database\agent-topology\derby\top-schema.ddl
[sql] 6 of 6 SQL statements executed successfully
[echo] Creating Agent Topology Foreign Keys ...
[sql] Executing resource: C:\Work\MQaaS\ibm-ucd-install\database\agent-topology\top-foreign-keys.ddl
[sql] 8 of 8 SQL statements executed successfully
[echo] Seeding Agent Topology Database ...
[sql] Executing resource: C:\Work\MQaaS\ibm-ucd-install\database\agent-topology\top-seed-data.sql
[sql] 2 of 2 SQL statements executed successfully
[echo] Creating HA Lock Database Schema ...
[sql] Executing resource: C:\Work\MQaaS\ibm-ucd-install\database\halock\derby\h1k-schema.ddl
[sql] 2 of 2 SQL statements executed successfully
[echo] Creating HA Lock Foreign Keys ...
[sql] Executing resource: C:\Work\MQaaS\ibm-ucd-install\database\halock\h1k-foreign-keys.ddl
[sql] 0 of 0 SQL statements executed successfully
[echo] Seeding HA Lock Database ...
[sql] Executing resource: C:\Work\MQaaS\ibm-ucd-install\database\halock\h1k-seed-data.sql
[sql] 2 of 2 SQL statements executed successfully
[echo] Seeding IBM UrbanCode Deploy Database ...
[sql] Executing resource: C:\Work\MQaaS\ibm-ucd-install\database\deploy\ud-seed-data.sql
[sql] 193 of 193 SQL statements executed successfully
[echo] Updating External URLs ...
[sql] Executing commands
[sql] 2 of 2 SQL statements executed successfully
[echo] Updating License Server URL ...
[sql] Executing commands
[sql] 1 of 1 SQL statements executed successfully
[echo] Setting up admin login credentials ...
[sql] Executing commands
[sql] 1 of 1 SQL statements executed successfully
[echo] Stopping embedded database ...

```

```
[java] Wed Dec 23 01:30:28 IST 2015 : Apache Derby Network Server - 10.8.3.1 - (1476465) shutdown
[propertyfile] Updating property file: C:\Program Files\ibm-ucd\server\conf\server\installed.properties
[copy] Copying 1 file to C:\Program Files\ibm-ucd\server
[echo] Do you want to install the Server as Windows service? y,N [Default:N]
```

Press <enter> ==>Use Default

```
[echo]
[echo] You can install service manually (see documentation).
[echo]
[propertyfile] Updating property file: C:\Program Files\ibm-ucd\server\conf\server\installed.properties
[propertyfile] Creating new property file: C:\Program Files\ibm-ucd\server\conf\installed.version
[echo] After starting the server, you may access the web UI by pointing your web-browser at
[echo] http://localhost:8080 to complete the Installation.
[echo] Installer Complete. (press return to exit installer)
```

```
[delete] Deleting directory C:\Users\admin\AppData\Local\Temp\install-7445003796587733363.tmp
```

BUILD SUCCESSFUL

Total time: 3 minutes 21 seconds

C:\Work\MQaaS\ibm-ucd-install>

```
[echo]
[echo] You can install service manually (see documentation).
[echo]
[propertyfile] Updating property file: C:\Program Files\ibm-ucd\server\conf\server\installed.properties
[propertyfile] Creating new property file: C:\Program Files\ibm-ucd\server\conf\installed.version
[echo] After starting the server, you may access the web UI by pointing your web-browser at
[echo] http://localhost:8080 to complete the Installation.
[echo] Installer Complete. (press return to exit installer)
```

```
[delete] Deleting directory C:\Users\admin\AppData\Local\Temp\install-3703141844035877983.tmp
```

BUILD SUCCESSFUL

C:\Work\MQaaS\ibm-ucd-install>

The IBM UrbanCode Deploy server is now installed.

Starting the IBM UrbanCode Deploy server

After the IBM UrbanCode Deploy server component is installed, start the server by completing the following steps:

1. Open a Command Prompt window.
2. Go to the IBM UrbanCode Deploy server installation directory, which in this example is c:\Program Files\ibm-ucd\server\bin.
3. Run **run_server**. This command runs the server in the same Command Prompt window.

Note: Alternatively, you can run **server start** as shown in Example A-2. This command starts a new applet window to start the server.

Example A-2 Run the server start command

```
C:\Program Files\ibm-ucd\server\bin>server start
2015-12-23 01:33:25,684 INFO  main com.urbancode.ds.persistence.database.DerbyHelper - Starting Derby...
2015-12-23 01:33:27,567 INFO  main com.urbancode.ds.persistence.database.DerbyHelper - Waiting for Derby to
start...
2015-12-23 01:33:27,884 INFO  main com.urbancode.ds.persistence.database.DerbyHelper - The Derby instance
on localhost:11377 is started

..... Output Start execution output.....
.....

2015-12-23 01:33:45,287 INFO  codestation.upgrader
com.urbancode.ds.repl.codestation.upgrade.CodestationUpgrader - Finished upgrading versions.
2015-12-23 01:33:45,573 INFO  main com.urbancode.ds.UDeployServer - Configuring Agent Network System for
single-server setup...
2015-12-23 01:33:49,248 INFO  main com.urbancode.ds.UDeployServer - done
2015-12-23 01:33:49,260 INFO  main com.urbancode.ds.UDeployServer - IBM UrbanCode Deploy server version
6.1.1.5.672018 started.
```

4. To confirm that the IBM UrbanCode Deploy server is started, open a web browser and enter the IBM UrbanCode Deploy server URL, which is `http://localhost:8080` in this example.

Stopping the IBM UrbanCode Deploy server


To stop the IBM UrbanCode Deploy server, complete the following steps:

1. Open a Command Prompt window.
2. Go to the IBM UrbanCode Deploy server installation directory, in this example,
`c:\Program Files\ibm-ucd\server\bin`
3. Run the command **server stop**.

Installing the IBM UrbanCode Deploy agent

Download and extract the agent installer on to the system where the agent is installed. You use the agent installer to install the IBM UrbanCode Deploy agent.

Complete the following steps:

1. Start the IBM UrbanCode Deploy web UI. Open a web browser and enter the IBM UrbanCode Deploy server URL, which is `http://localhost:8080` in this example.
2. From the web UI top menu, click the Help drop-down menu  and select **Tools**.
3. In the next window, click **IBM UrbanCode Deploy Agent** and download the agent installer compressed file `ibm-ucd-agent.zip`.

Note: In this example, both the IBM UrbanCode Deploy server and the IBM UrbanCode Deploy agent run on the same system. If you choose to run the agent on a different system, download the `ibm-ucd-agent.zip` file on to the system where you want to run the agent.

After the download completes, the compressed file `ibm-ucd-agent.zip` is in the Downloads directory.

4. Extract the contents of the `ibm-ucd-agent.zip` file into the `ibm-ucd-agent-install` directory.
5. Make sure that the `JAVA_HOME` system variable is set correctly to `<Java-InstallPath>\jre`.
6. Open a Command Prompt window and go to the directory where the `ibm-ucd-agent.zip` file is extracted, which is `\ibm-ucd-agent-install` in this example.
7. Run **Install-agent.bat** to install the agent.
8. Specify the information as the installation program prompts you. Accept the default values (displayed within brackets) by pressing Enter.

Example A-3 shows the console of the IBM UrbanCode Deploy agent installation in this example.

Example A-3 Install the IBM UrbanCode Deploy agent

```
C:\Work\MQaaS\ibm-ucd-agent-install>install-agent.bat
```

```
Buildfile: C:\Work\MQaaS\ibm-ucd-agent-install\install.with.groovy.xml
```

```
check-classpath-includes:
```

```
define-groovy-classpath-include:
```

```
define-groovy-classpath-no-include:
```

```
taskdef-groovy:
```

```
install-agent:
```

```
  [unzip] Expanding: C:\Work\MQaaS\ibm-ucd-agent-install\overlay.zip into C:\Users\admin\AppData\Local\Temp\agent-install-6060137285723659729.tmp
```

```
  [echo]
```

```
  [echo] Installing IBM UrbanCode Deploy Agent
```

```
  [echo] Enter the directory where agent should be installed.
```

```
[Default: C:\Program Files\agent]
```

```
Press <enter> ==>Use Default
```

```
  [echo] The specified directory does not exist. Do you want to create it? Y,n [Default: Y]
```

```
Press <enter> ==>Use Default
```

```
  [echo]
```

```
  [echo] Installing Agent to: C:\Program Files\agent
```

```
  [echo] Enter the home directory of the JRE/JDK used to run this agent.
```

```
[Default: C:\Program Files\IBM\java\jre]
```

```
Press <enter> ==>Use Default
```

```
  [echo] JAVA_HOME: C:\Program Files\IBM\java\jre
```

```
  [echo] Will the agent connect to an agent relay instead of directly to the server? y,N  
[Default: N]
```

Press <enter> ==>Use Default

[echo] Enter the host name or address of the server the agent will connect to.
[Default: localhost]

Press <enter> ==>Use Default

[echo] Enter the agent communication port for the server. [Default: 7918]

Press <enter> ==>Use Default

[echo] Do you want to configure another failover server connection? y,N [Default: N]

Press <enter> ==>Use Default

[echo] Does the server agent communication use mutual authentication with SSL? y,N
[Default: N]

Press <enter> ==>Use Default

[echo] Enter the name for this agent. [Default: MyComputerName]

You can choose a different name than computer name.

Press <enter>

[echo] The agent can be added to one or more teams when it first connects to the server.
Changing this setting after initial connection to the server will not have any effect.

[echo] Enter teams to add this agent to, separated by commas. [Default: None]

Specify a Team name (can be any noun). This demo uses team as "MQaaSRedPaper"

Press <enter>

[copy] Copying 2 files to C:\Program Files\agent\conf

[copy] Copying 1 file to C:\Program Files\agent\properties

[propertyfile] Creating new property file:

C:\Program Files\agent\conf\agent\installed.properties

[propertyfile] Updating property file: C:\Program Files\agent\conf\agent\installed.properties

[propertyfile] Updating property file: C:\Program Files\agent\conf\agent\installed.properties

[copy] Copying 56 files to C:\Program Files\agent\lib

[copy] Copying 4 files to C:\Program Files\agent\monitor

[copy] Copying 4 files to C:\Program Files\agent\native

[copy] Copying 1621 files to C:\Program Files\agent\opt

[copy] Copying 2 files to C:\Program Files\agent\opt\udclient

[mkdir] Created dir: C:\Program Files\agent\bin\service

[copy] Copying 1 file to C:\Program Files\agent\bin

[copy] Copying 2 files to C:\Program Files\agent\bin

[copy] Copying 2 files to C:\Program Files\agent\bin

[copy] Copying 3 files to C:\Program Files\agent\bin

[copy] Copying 1 file to C:\Program Files\agent\bin\service

[copy] Copying 1 file to C:\Program Files\agent\bin

[copy] Copying 3 files to C:\Program Files\agent\bin

[echo] Installed version 6.1.1.5.671426

[echo] Do you want to install the Agent as Windows service? y,N [Default: N]

Press <enter> ==>Use Default

```
[echo]
[echo] You can install service manually (see documentation).
[echo]
[echo] Installer Complete. (press return to exit installer)
```

```
[delete] Deleting directory
C:\Users\admin\AppData\Local\Temp\agent-install-6060137285723659729.tmp
```

BUILD SUCCESSFUL
Total time: 1 minutes 19 seconds

Starting the IBM UrbanCode Deploy agent

After the IBM UrbanCode Deploy agent is installed, start the agent by completing the following steps:

1. Open a Command Prompt window and go to the IBM UrbanCode Deploy agent installation directory, which is `\Program Files\agent\bin` in this example.
2. Run `run_agent.cmd`, as shown in Example A-4. This command runs the agent in the same Command Prompt window.

Note: Alternatively, you can run `agent start`. This command starts a new applet window to start the agent.

Example A-4 Start the IBM UrbanCode Deploy agent

```
C:\Program Files\agent\bin>run_agent.cmd
2015-12-23 01:48:03,631 INFO com.urbancode.air.agent.AgentWorker - Logging configured
2015-12-23 01:48:03,662 INFO com.urbancode.air.agent.AgentWorker - Agent version: 6.1.1.5.671426
2015-12-23 01:48:09,912 INFO com.urbancode.air.devilfish.common.SaContainer - Starting SaContainer...
2015-12-23 01:48:10,006 INFO com.urbancode.air.devilfish.apps.plugin.PluginRuntimeServer - Plugin Recovery ...
2015-12-23 01:48:10,022 INFO com.urbancode.air.devilfish.common.SaContainer - Started SaContainer
2015-12-23 01:48:10,022 INFO com.urbancode.air.agent.AgentWorker - Agent started
2015-12-23 01:48:11,272 INFO com.urbancode.air.devilfish.apps.agentcontrol.PropertiesCollector - Checking connection to localhost:8080...
2015-12-23 01:48:11,272 INFO com.urbancode.air.devilfish.apps.agentcontrol.PropertiesCollector - HTTP connection successful.
```

3. Verify that the agent is started.

From the IBM UrbanCode Deploy web UI home window, click the **Resources** tab and then click the **Agents** tab. The agents status should be *Online*, as shown in Figure A-1 on page 161.

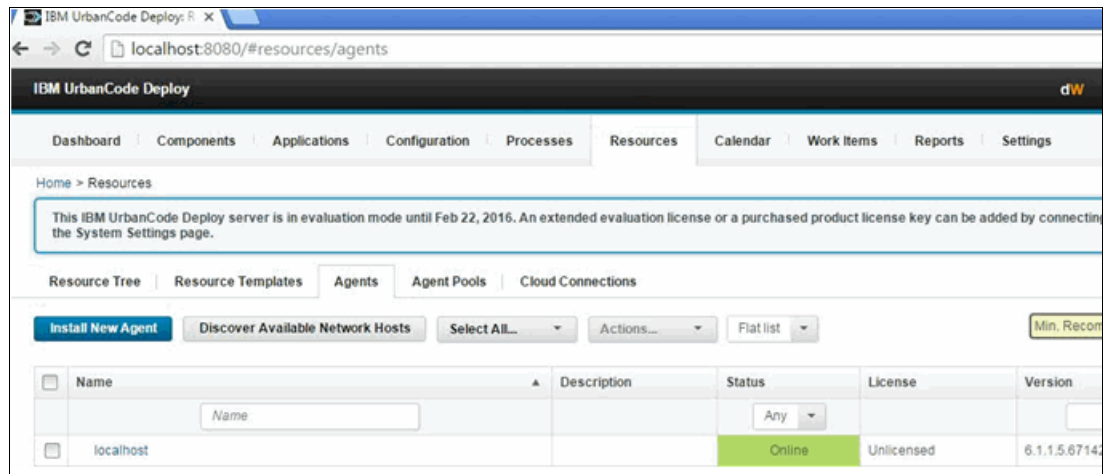


Figure A-1 Agent view and status in the IBM UrbanCode Deploy web UI

Stopping the IBM UrbanCode Deploy Agent

To stop the IBM UrbanCode Deploy agent, complete the following steps:

1. Open a Command Prompt window and go to the IBM UrbanCode Deploy agent installation directory, which is `\Program Files\agent\bin` in this example.
2. Run `agent.cmd stop` to stop the agent.

Loading the IBM MQ plug-in for IBM UrbanCode Deploy

Before using IBM UrbanCode Deploy to create IBM MQ services, you must create certain IBM UrbanCode Deploy artifacts that are common to the scenarios that are presented in this paper. One of the artifacts is the IBM MQ plug-in for IBM UrbanCode Deploy.

Complete the following steps:

1. Download the IBM MQ plug-in for IBM UrbanCode Deploy from IBM developerWorks from the following website:

<https://developer.ibm.com/urbancode/plugin/websphere-mq/>

After the download is complete, the `WebSphereMQ-4.664802.zip` file is in the Downloads directory.

2. Load the IBM MQ plug-in for IBM UrbanCode Deploy; this plug-in is an automation plug-in.

From the IBM UrbanCode Deploy web UI window, click **Settings** → **Automation** → **Automation Plugin** and click **Load Plugin**, as shown in Figure A-2.

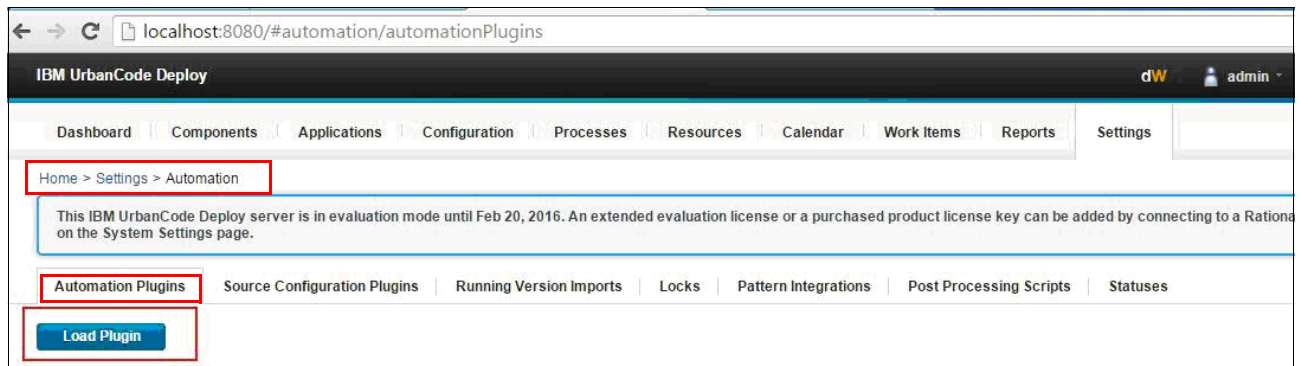


Figure A-2 Load the IBM MQ plug-in for IBM UrbanCode Deploy from the IBM UrbanCode Deploy web UI

3. Browse to the location where the plug-in's compressed file is, select the file (WebSphereMQ-4.664802.zip in this example), and click **Submit**, as shown in Figure A-3.

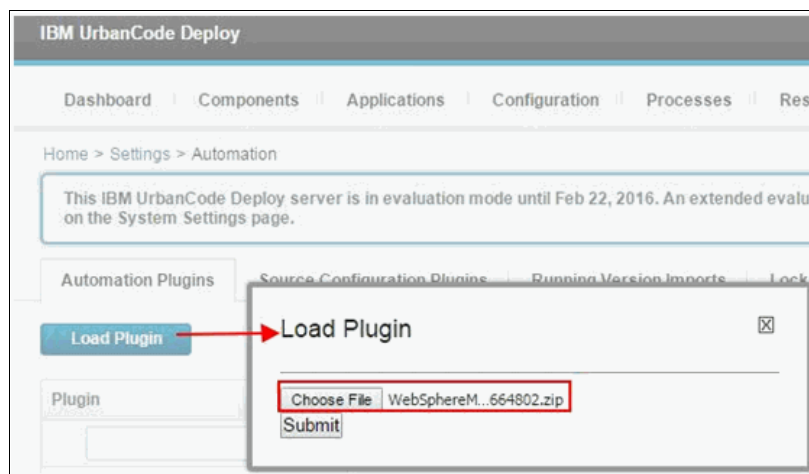


Figure A-3 Load the IBM MQ plug-in for IBM UrbanCode Deploy from the web UI

4. After the plug-in is loaded, the new plug-in is listed with the name WebSphereMQ, as shown in Figure A-4 on page 163.

The screenshot shows the IBM UrbanCode Deploy web interface. At the top, there's a navigation bar with tabs like Dashboard, Components, Applications, Configuration, Processes, Resources, Calendar, Work Items, Reports, and Settings. Below this, a breadcrumb trail reads 'Home > Settings > Automation'. A message box states: 'This IBM UrbanCode Deploy server is in evaluation mode until Feb 20, 2016. An extended evaluation license or a purchased product license key can be added by connecting to a Rational License Server on the System Settings page.' Below the message, there's a sub-navigation bar with tabs: Automation Plugins, Source Configuration Plugins, Running Version Imports, Locks, Pattern Integrations, Post Processing Scripts, and Statuses. A 'Load Plugin' button is visible. The main content area displays a table of installed plugins.

Plugin	Description	Version
IBM UrbanCode Deploy Versions	Plugin for editing Component Versions in IBM UrbanCode Deploy.	61.671775
Shell	The Shell plugin allows users to run custom shell scripts during the deployment process.	6.671801
System Information	The System Information plugin includes a variety of checks to perform against the operating system. These steps can be used to verify that a deployment can succeed or has succeeded.	3.671787
WebSphereMQ	Plugin to interact directly with WebSphere MQ	4.664802
WinRS Agent Install	Plugin for installing agents to remote machine using WinRS	3.621653

Figure A-4 WebSphereMQ plug-in loaded and available in the Plugin list

Application deployment with IBM UrbanCode Deploy

This section takes a step-by-step approach to application deployment with IBM UrbanCode Deploy.

Creating a team

Team is a logical namespace that groups a set of related IBM UrbanCode Deploy objects, such as agents, resource trees, components, and environments, for application deployment. This paper uses a team that is named MQaaSRedPaper to group all these objects. Creating this team is an important step because associating the IBM UrbanCode Deploy objects with this team logically groups the IBM UrbanCode Deploy objects that used in the examples of this paper.

To create a team, complete the following steps:

1. From the IBM UrbanCode Deploy web UI, click **Settings** → **Security** → **Teams**.
2. Click **Create Team** in the left menu.

3. In the Name field, enter the team name, which is MQaaSRedPaper for the examples in this paper, and click **Save**, as shown in figure Figure A-5.

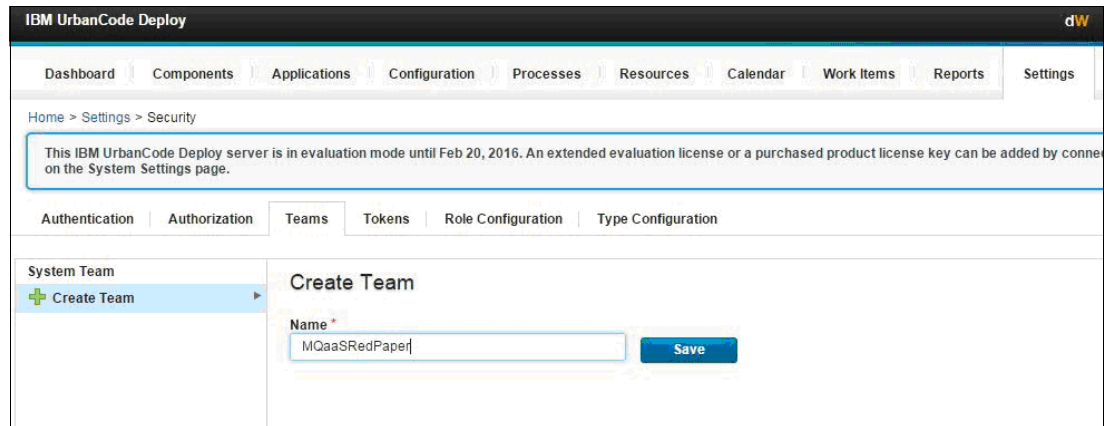


Figure A-5 Create a team

Adding role members to the team

After the team is created, you can add role members to the team. Complete the following steps:

1. Open the team by clicking it.
2. On the Teams page (MQaaSRedPaper in this example), click **Add Users and Groups** to add the role members to the team.
3. From the Users & Groups window, drag the user and group objects into the corresponding fields in the Role Members window, as shown in Figure A-6.

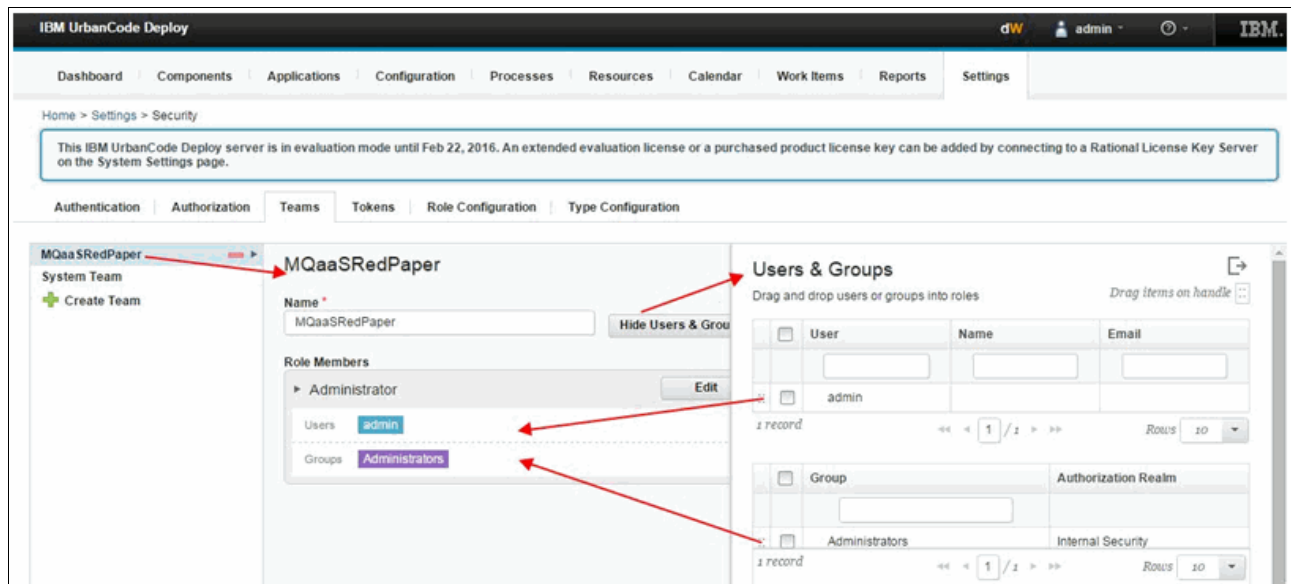


Figure A-6 Add role members to the team

Creating IBM UrbanCode Deploy objects for application deployment

IBM UrbanCode Deploy objects, such as agents, components, and a resource tree, must be created for application deployment.

Adding the team to the IBM UrbanCode Deploy agent

Using the IBM UrbanCode Deploy agent that was created in “Installing the IBM UrbanCode Deploy agent” on page 157, set the team on the agent by completing the following steps.

Note: It is a preferred practice to give the agent the same name as the server where it runs. In the examples in this appendix, both the IBM UrbanCode Deploy server and the agent run on the same computer, so the agent is referred to as `localhost`. In other chapters of this paper, the name might be different, for example, `xxx.hursley.ibm.com` because the agent is running on a different computer.

Agent names can be changed by editing the agent's `/conf/agent/installed.properties` file and restarting the agent.

1. From the IBM UrbanCode Deploy web UI home page, click **Resources** → **Agents**.
2. Select the agent (**localhost** in this example) and click the Edit icon (pen symbol), as shown in Figure A-7.

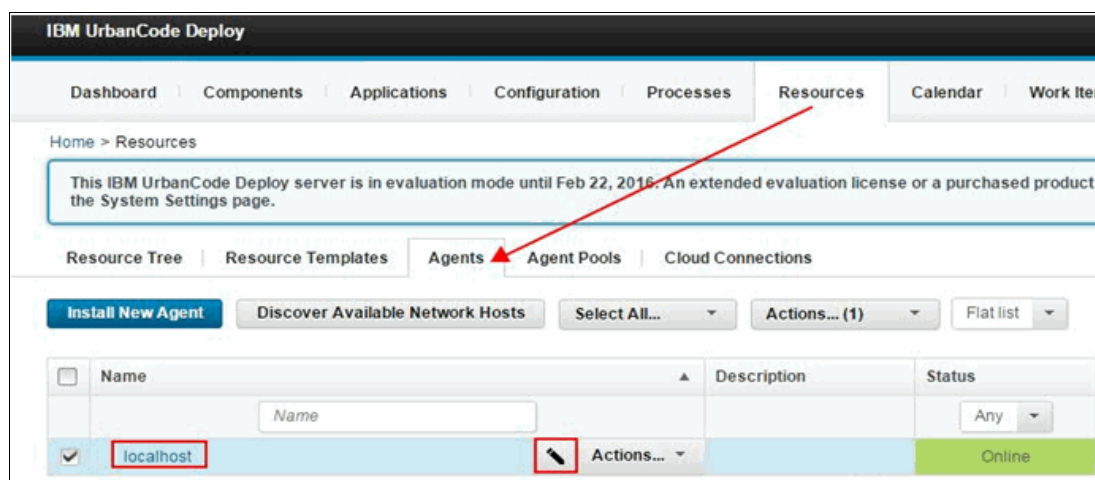


Figure A-7 Edit the agent from the IBM UrbanCode Deploy web UI

3. In the Edit Agent wizard, click the plus sign (+) for Team and select the team **MQaaSRedPaper**, which was created in “Creating a team” on page 163. Then, click **Add**, as shown in Figure A-8.

The screenshot shows the 'Edit Agent' wizard. At the top, the 'Name' field contains 'localhost'. Below it, a green plus sign is visible. The 'Teams' section is expanded, showing a dropdown menu with '1 Selected' and a 'Clear Selection' button. Below the dropdown, the team 'MQaaSRedPaper' is selected and highlighted with a red box. The 'Type' dropdown is set to 'Standard Agent'. At the bottom, the 'Add' button is highlighted with a red box, and a 'Cancel' button is also visible. A note at the bottom states: 'Agent names can be changed by editing the agent's conf/agent/installed.properties file and restarting the agent.'

Figure A-8 Add a team to the agent

Creating IBM UrbanCode Deploy components

Components represent deployable items along with user-defined processes that operate on them, usually by deploying them.

In this paper, components represent deployable items such as queue managers, destinations (topics and queues), and other IBM MQ objects. A user-defined process is an activity in which you design the deployment flow for the IBM MQ objects.

To create a component, complete the following steps:

1. From the IBM UrbanCode Deploy web UI home page, click the **Components** tab and click **Create Component**.
2. In the Create Component wizard, enter a name for the component. In the examples in this paper, the component name is MQaaS.
3. In Teams, click the plus sign (+) and select the team, which is **MQaaSRedPaper** in this example.
4. Accept the default values for the other fields and click **Save**.

Figure A-9 shows the steps to create a component.

The screenshot displays the IBM UrbanCode Deploy interface. On the left, the 'Components' tab is active, and a red arrow points to the 'Create Component' button. The main area shows the 'Create Component' wizard. The 'Name' field is 'MQaaS'. The 'Teams' field contains 'MQaaSRedPaper' with a plus sign to add more. The 'Template' is set to 'None' and the 'Component Type' is 'Standard'. The 'Version Source Configuration' section includes 'Source Configuration Type', 'Import Versions Automatically' (unchecked), 'Copy to Code Station' (checked), and 'Default Version Type' set to 'Full'. There are three radio button options for version import: 'Use the system's default version import agent/tag.' (selected), 'Import new component versions using a single agent.', and 'Import new component versions using any agent with the specified tag.'. The 'Cleanup Configuration' section has 'Inherit Cleanup Settings' checked.

Figure A-9 Create an IBM UrbanCode Deploy component

Creating a component process: Add Queue Manager

Each component has at least one process. A process defines the deployment flow by using the process steps that are provided by the IBM UrbanCode Deploy plug-ins. The examples in this paper use the process steps of the IBM MQ plug-in for IBM UrbanCode Deploy.

To create the Add Queue Manager process for the MQaaS component, complete the following steps:

1. In the MQaaS component, click the **Process** tab and click **Create Process**.
2. In the Create Process wizard, give the process a meaningful name. In this example, the process name is **Add Queue Manager**.
3. For Process Type, click **Operational(No Version Needed)**.
4. Click **Save**.

Figure A-10 shows the steps to create a component process.

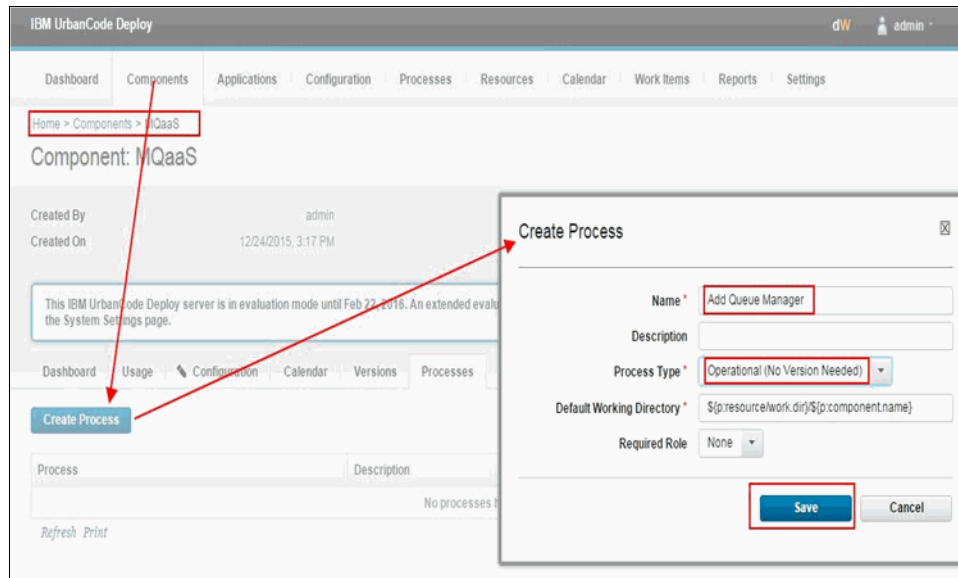


Figure A-10 Create a process

After the process is created, it is displayed in the process list of the component.

5. Open the process by clicking it. A process designer opens; use the process designer to design the flow of the process steps.
 - Process steps on the left, under the Step Palette, help design the process.
 - Process steps are the list of actions that are used to form a deployable item. The examples in this paper use process steps from the IBM MQ plug-in for IBM UrbanCode Deploy. For example, a process step can be a deployable item from the IBM MQ plug-in, such as Create Queue Manager or Create Queue. A process step can also be a set of process steps designed as a single flow, such as Create Queue Manager and Create Queue.
 - The process designer includes Start and Finish steps, as shown in Figure A-11 on page 169.

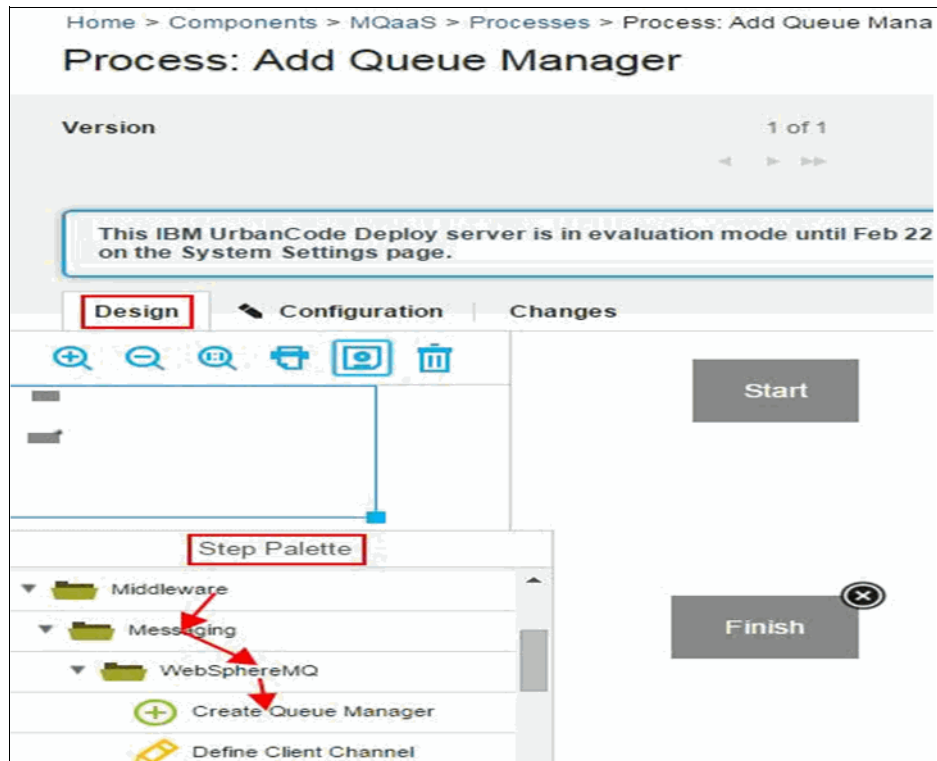


Figure A-11 Process designer view

6. In the Step Palette, find the IBM MQ process steps by clicking **Middleware** → **Messaging** → **WebSphereMQ**, as shown in Figure A-11.

The purpose of the Add Queue Manager process is to create an IBM MQ queue manager and to start the queue manager, so two IBM MQ process steps, Create Queue Manager and Start Queue Manager, are needed for this process.

Adding the Create Queue Manager process step

To add the Create Queue Manager process step, complete the following steps:

1. Drag the Create Queue Manager process step from the list of process steps to the process designer. The window that is shown in Figure A-12 opens.

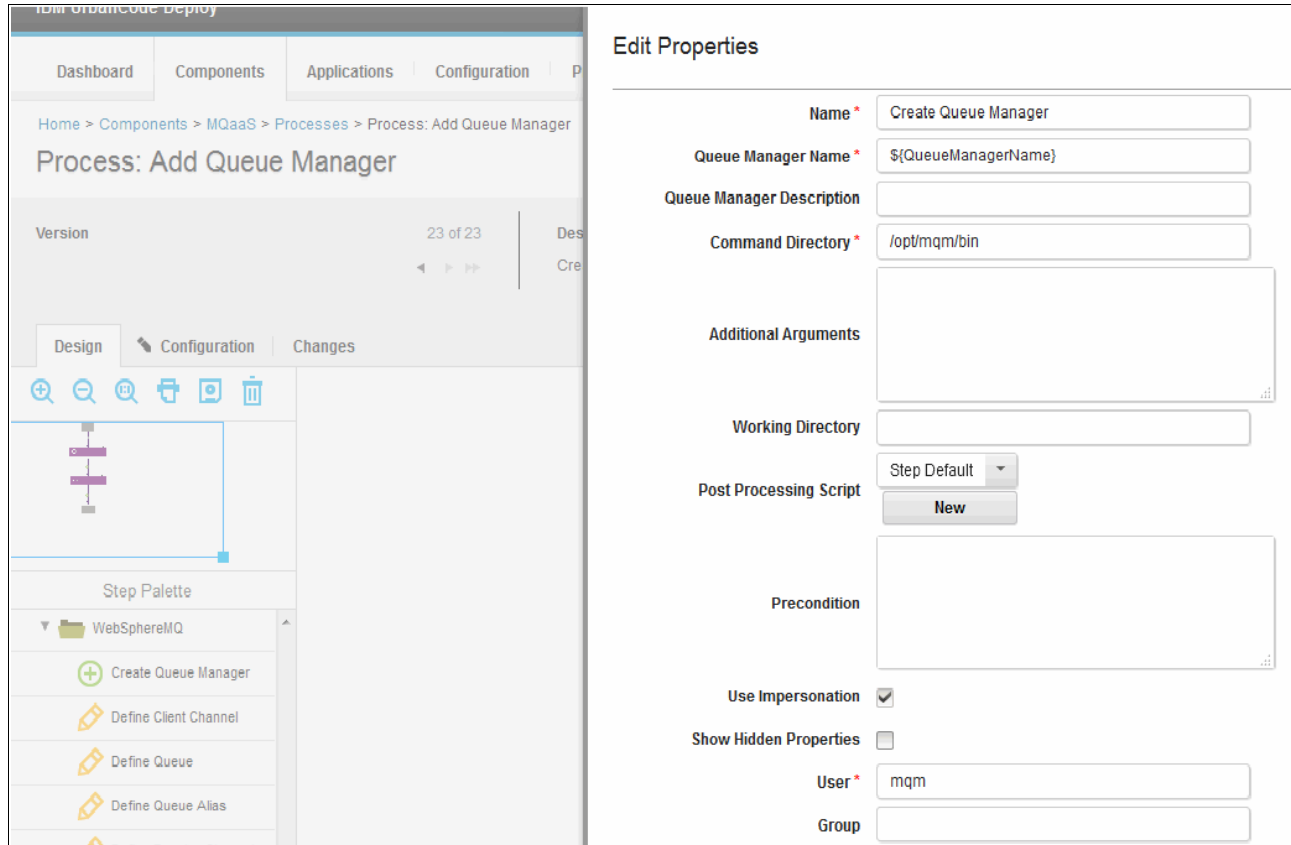


Figure A-12 Add the Create Queue Manager process step

2. Enter the queue manager name in the format `${QueueManagerName}`, which helps to supply the queue manager name as input during run time. You also must add a property to the process configuration. To add the new property, complete the following steps:
 - a. Save the process design.
 - b. Click the process' **Configuration** tab.
 - c. Click **Add Property**. The Edit Property wizard opens.
 - d. Enter the name and the label of the property, in this example, QueueManagerName.
 - e. Click **Required** to require an input value from the user for this property.
 - f. Click **Save**.

Figure A-13 on page 171 shows the Add Property window.

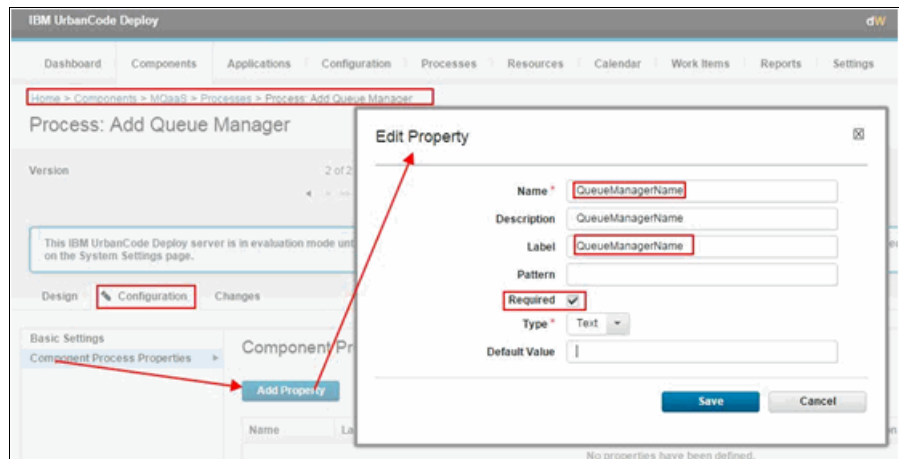


Figure A-13 Add a property

Adding the Start Queue Manager process step

To add the Start Queue Manager process step, complete the following steps:

1. From the Step Palette, click **Middleware** → **Messaging** → **WebSphereMQ**.
2. From the list of process steps, drag the **Start Queue Manager** process step to the process designer, as shown in Figure A-14.

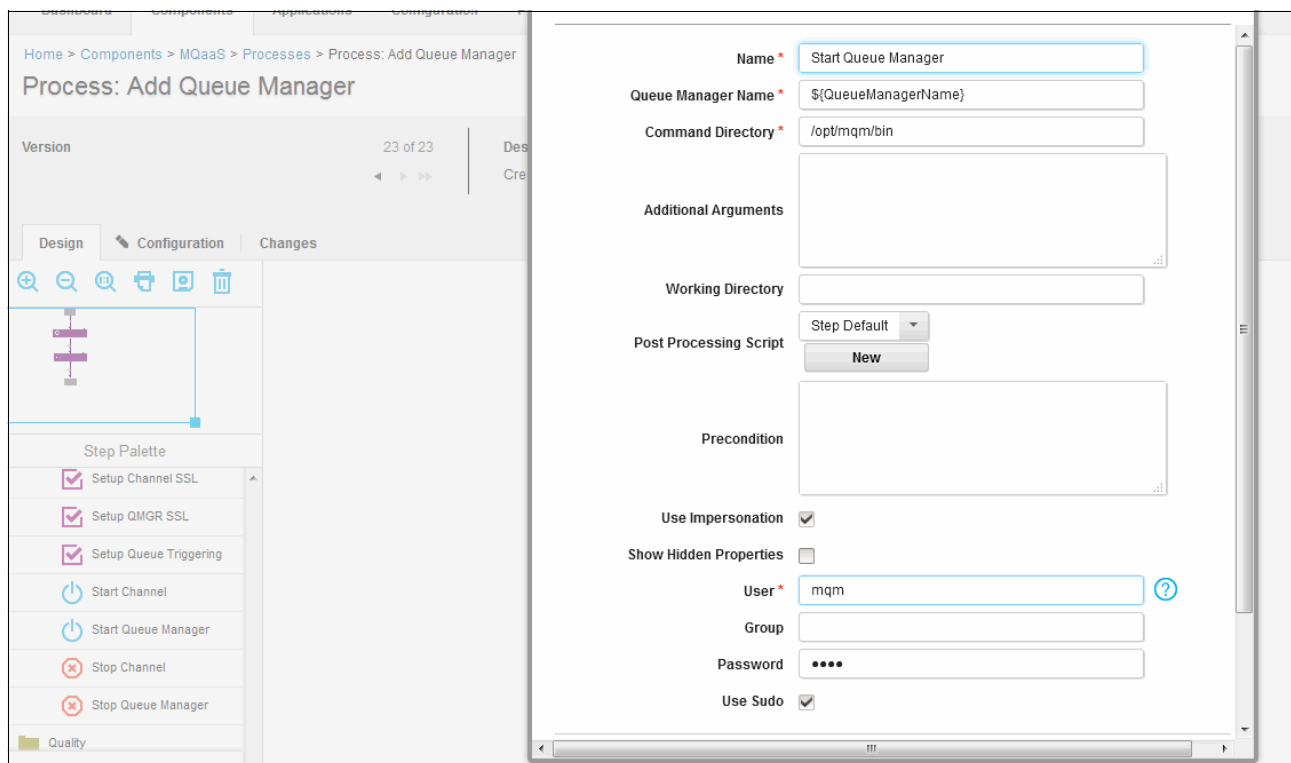


Figure A-14 Adding the Start Queue Manager process step

3. Enter the queue manager name in the format `{$QueueManagerName}`. You use this variable to specify the queue manager name as an input value during run time.

Note: You do not have to add a property because the property QueueManagerName was added to the Add Queue Manager process in step 2 on page 170.

4. Link all the process steps as shown in Figure A-15. Save the changes.

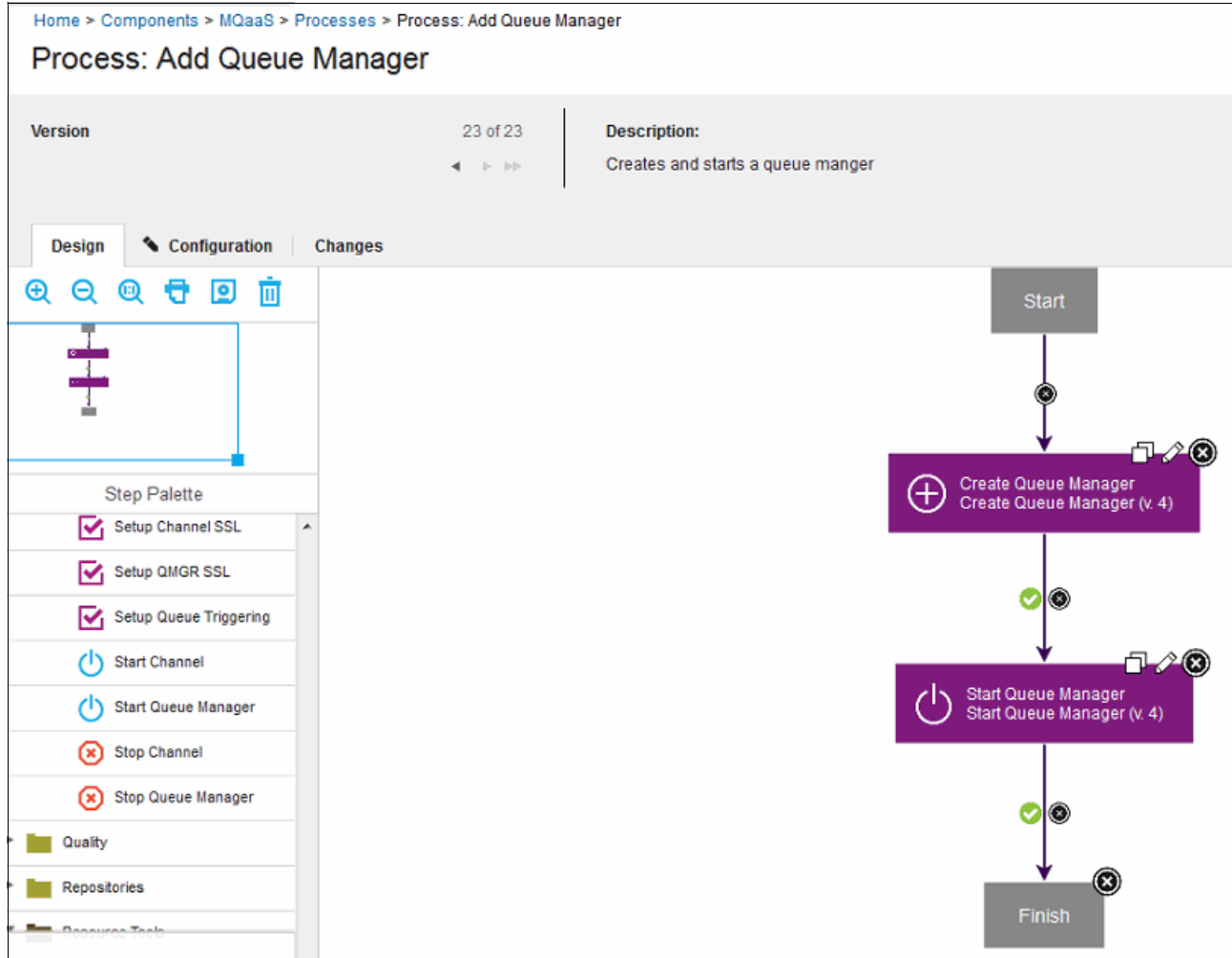


Figure A-15 Complete design of the Add Queue Manager process

You completed the creation of a component and the associated process for adding a queue manager. Similarly, you can create another process to create a queue destination to send messages to and receive messages from.

Creating a component process: Add Queue

This section describes how to create the Add Queue component process. The Add Queue process includes artifacts to create a queue for a given queue manager based on user input.

To create the Add Queue process for the MQaaS component, complete the following steps:

1. From the IBM UrbanCode Deploy web UI, click **Components** → **MQaaS** and click the **Process** tab to create a component process, as shown in Figure A-16 on page 173.

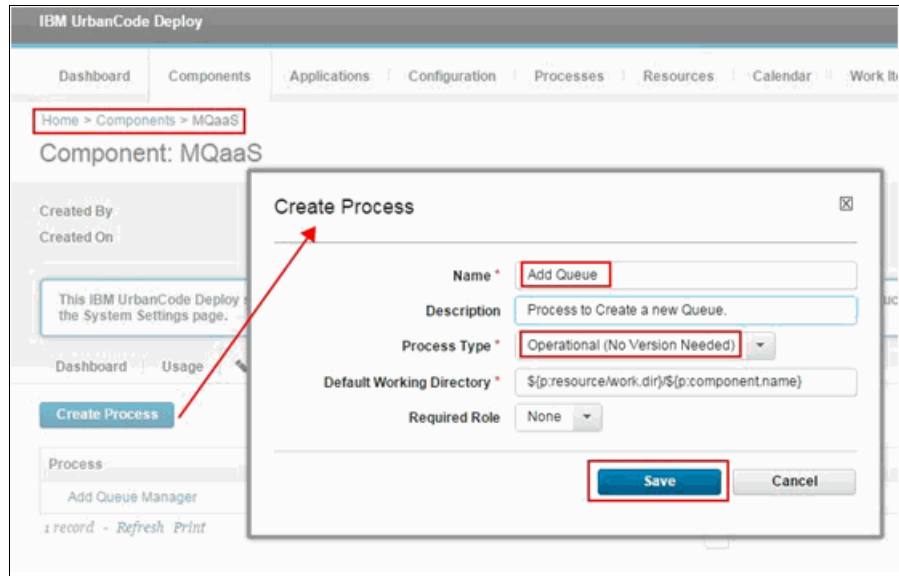


Figure A-16 Create a process to add a queue

2. Click **Create Process**. The Create Process wizard opens. Enter the process name Add Queue.
3. Make sure that the Process Type is set to **Operational (No Version Needed)**.
4. Click **Save**.
5. The new process (Add Queue) is included in the process list. To access the process list, from the IBM UrbanCode Deploy web UI home page, click **Components** → **MQaaS** → **Process**.

6. Click the **Add Queue** process to open it in the designer view, as shown in Figure A-17.

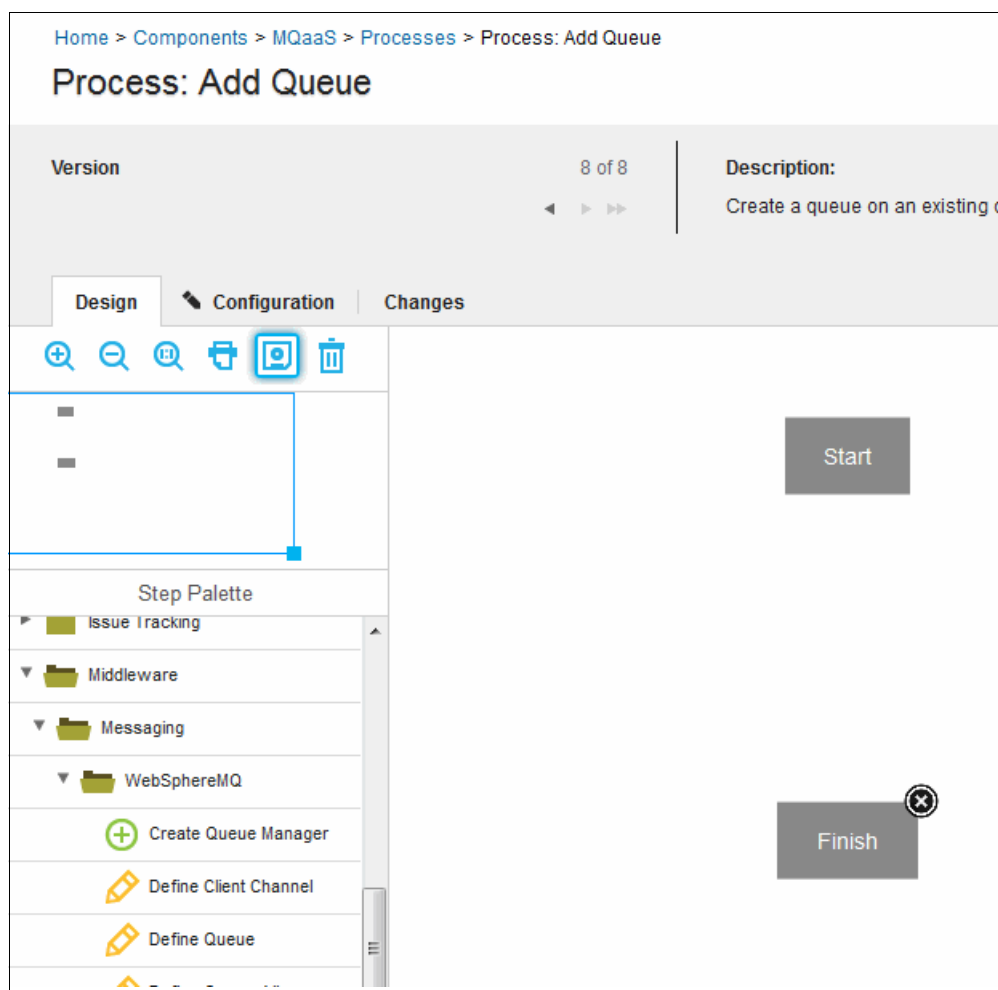


Figure A-17 Process designer palette for the Add Queue process

7. To add steps to this process, use the process steps that are provided by the IBM MQ plug-in for IBM UrbanCode Deploy by clicking **Middleware** → **Messaging** → **WebSphereMQ**.
8. You can use the Add Queue process to create queues on a specific queue manager. Drag the Define Queue process step from the list of IBM MQ process steps to the process designer.
9. A configuration wizard opens (Figure A-18 on page 175). Enter the queue manager name in the format `${QueueManagerName}` and Queue Name as `${QueueName}`. You can use these variables to specify the queue manager and queue names input values during run time.

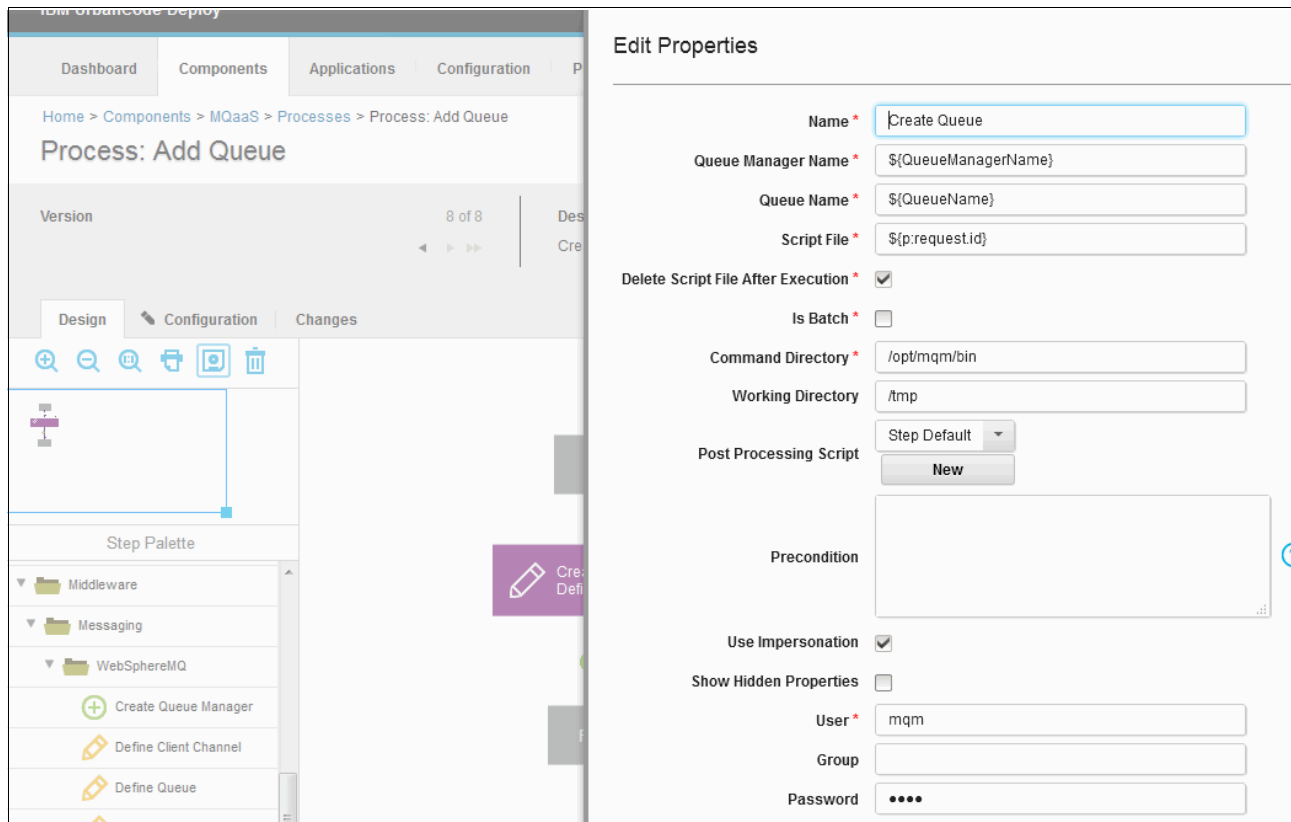


Figure A-18 Create the Create Queue process

10. You also must add two properties to this process, QueueManagerName and QueueName, so that their values can be received as user input during run time.
 - a. Add the QueueManagerName property, as described in step 2 on page 170.
 - b. Add the QueueName property, as shown in Figure A-19.

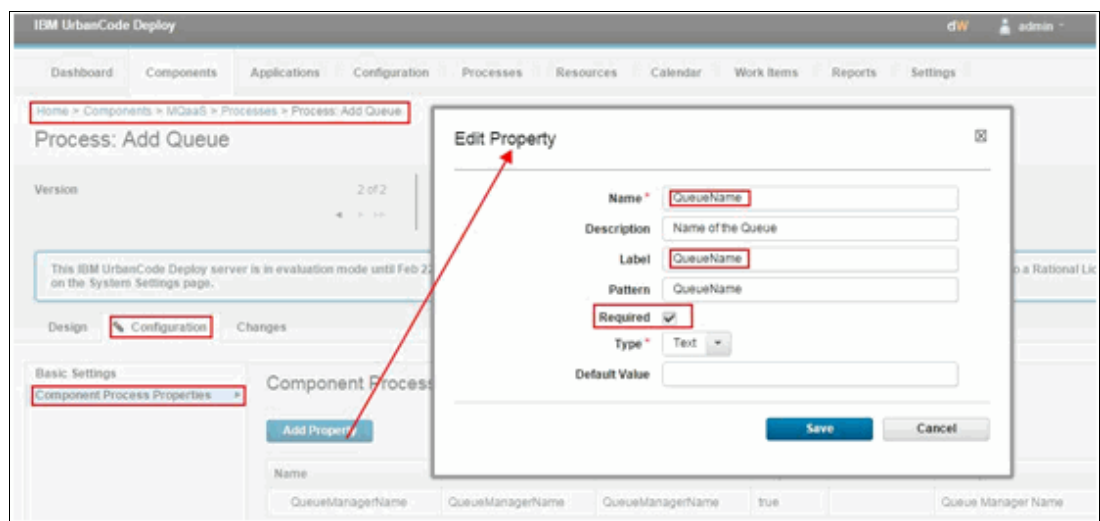


Figure A-19 Add the QueueName property to the Add Queue component process

11. Link all the process steps as shown in Figure A-20. Save the changes.

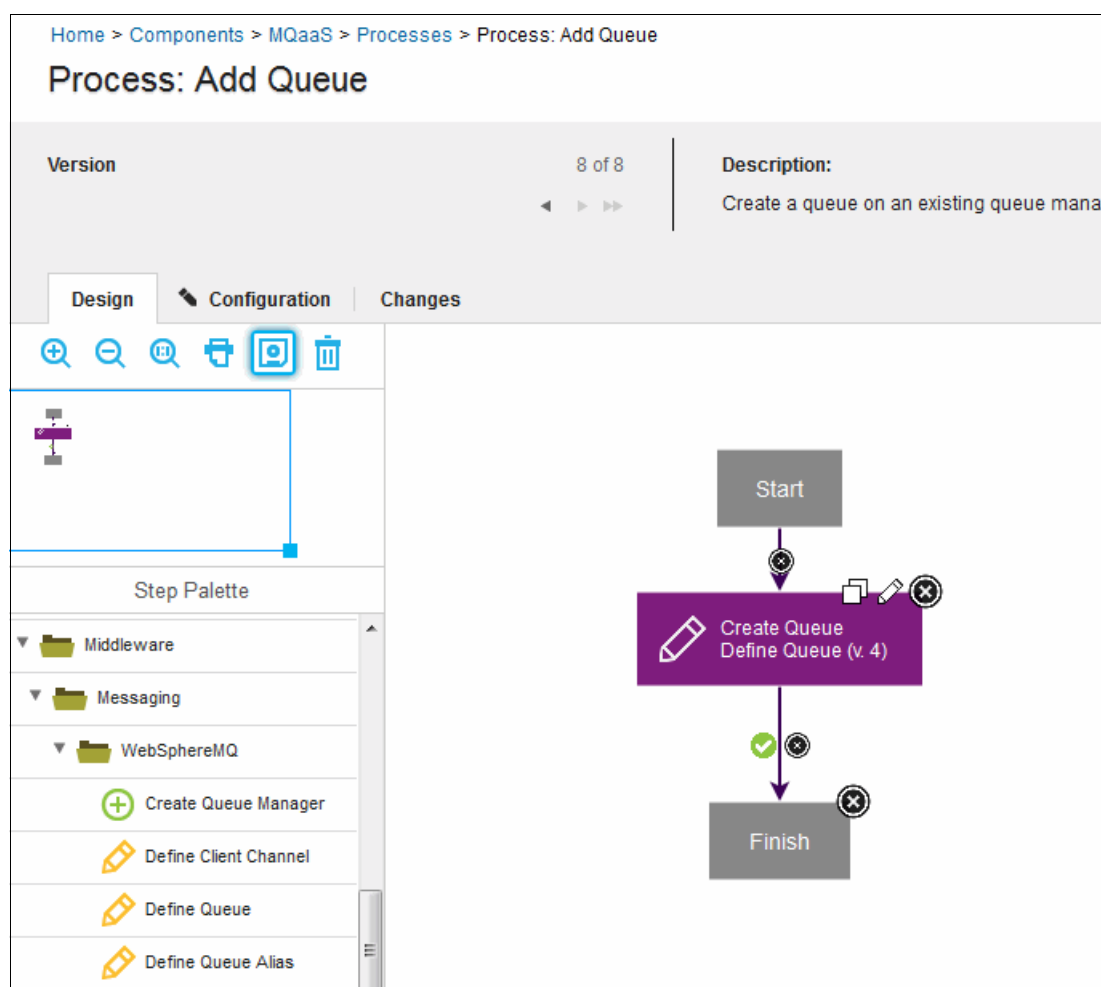


Figure A-20 Complete design for the Add Queue process

Creating a resource

A *resource* is a logical deployment target that typically resolves to an agent. A resource can represent an agent, agent pool, component, or an organizational entity that is used to group other resources. A component-type resource can point directly to the agent or agent pool that deploys it, or point to another resource in a hierarchical chain of resources. If a resource is part of a hierarchy, it delegates its automation, if any, up the chain until it finds an agent.

To create a resource, complete the following steps:

1. From the IBM UrbanCode Deploy web UI home page, click the **Resources** tab and click **Create Top-Level Group**.
2. The resource configuration wizard opens. Enter a name for the resource, which is MQaaS in this example (see Figure A-21 on page 177).

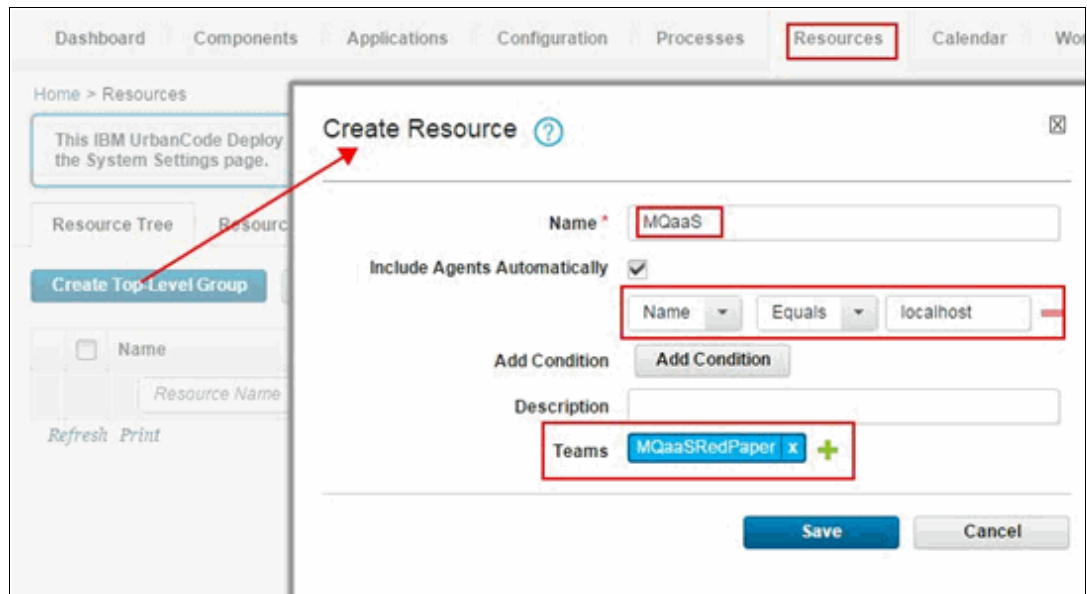


Figure A-21 Create the MQaaS resource

3. Select the **Include Agents Automatically** check box. This example uses a name conditional match, which finds agents by matching a name and loads the agent automatically.

Note: The agent and the resource must belong to same team. If they do not belong to same team, even if the name matches, the agent is not included.

4. For Teams, click **MQaaSRedPaper**.
5. Click **Save**.
6. In the Resources page, the resources list includes the new resource MQaaS as a tree structure with the agent as a child of the resource (localhost in this example) included automatically (see Figure A-22).

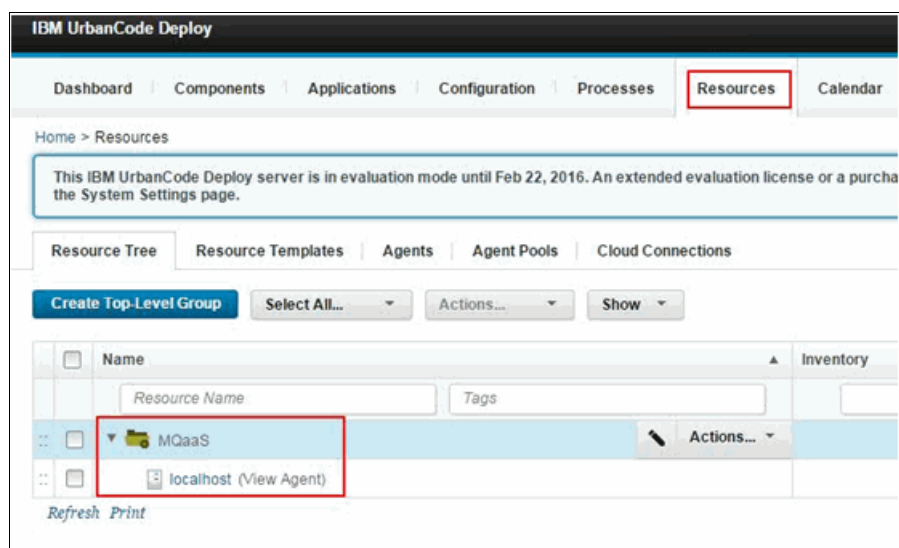


Figure A-22 Resource tree view with agent included automatically

Note: If the agent is not included automatically in the resource tree, it indicates that the resource is not finding the agent. The most common reason for this problem is that the team name is not set correctly for either the resource or the agent, resulting in them being associated with different teams.

You can confirm that the team name is correct by opening the resource in edit mode. From the IBM UrbanCode Deploy web UI home page, click **Resources** → **MQaaS**. To open the agent in edit mode from the IBM UrbanCode Deploy web UI, click **Resources** → **Agents** → **localhost**.

Adding the component to the resource tree

You must add the component that was created in “Creating IBM UrbanCode Deploy components” on page 166 to this resource tree (MQaaS).

To add the component to the resource tree, complete the following steps:

1. From the MQaaS resource tree view (see Figure A-22 on page 177), select the agent and click **Actions** → **Add Component**.
2. In the Add Component wizard, select the **MQaaS** component and click **Add**, as shown in Figure A-23.

The screenshot shows the IBM UrbanCode Deploy web interface. On the left, the 'Resource Tree' is visible with 'MQaaS' selected. On the right, the 'Create Resource' wizard is open. The 'Component' dropdown is set to 'MQaaS'. The 'Name' field is 'MQaaS'. The 'Teams' dropdown is set to 'MQaaS@StoolPigeon'. The 'Inherit Teams From Parent' checkbox is checked. The 'Default Impersonation' checkbox is unchecked. Below the form, there is a section for 'Role Properties: MQaaS' which contains an empty table with columns 'Property', 'Description', 'Value', and 'Actions'. A 'Refresh' button is at the bottom of this section.

Figure A-23 Add the MQaaS component to the resource tree

3. After the component is added, the resource tree is complete, as shown in Figure A-24 on page 179.

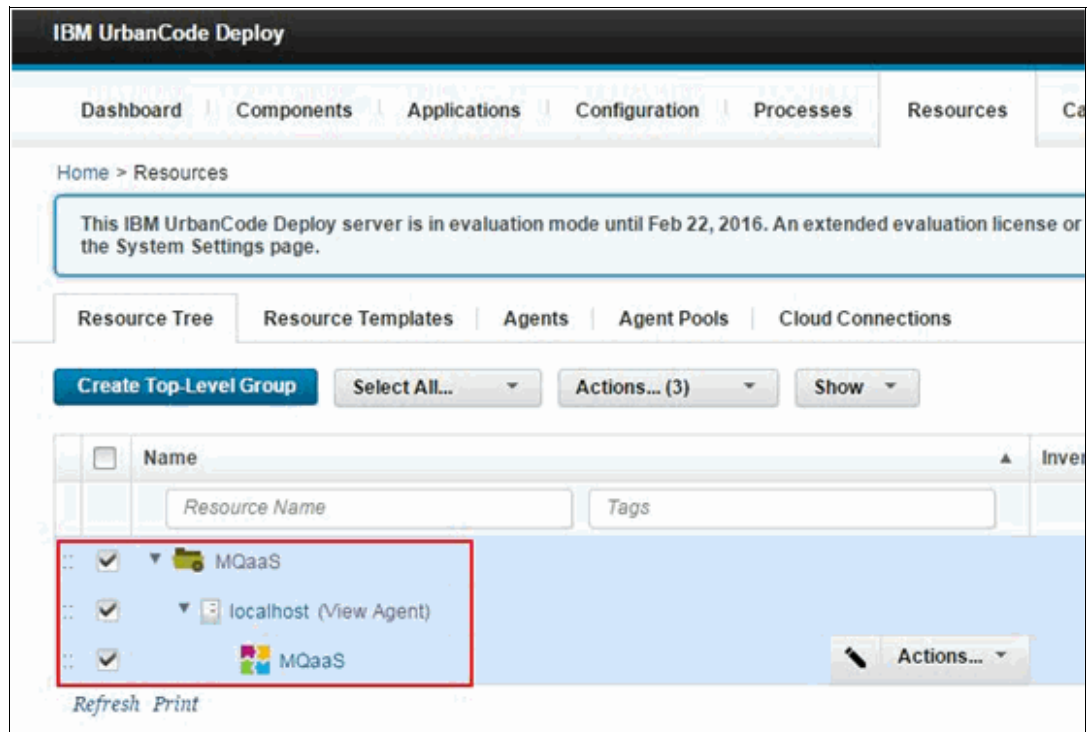


Figure A-24 Complete resource tree view

You now have a resource tree that groups agents and components. The next step is to create an application that uses this resource tree for deployment.

Creating an application

To create the MQaaS-Demo-Application, complete the following steps:

1. From the IBM UrbanCode Deploy web UI home page, click **Applications** → **Create Application**, as shown in Figure A-25.

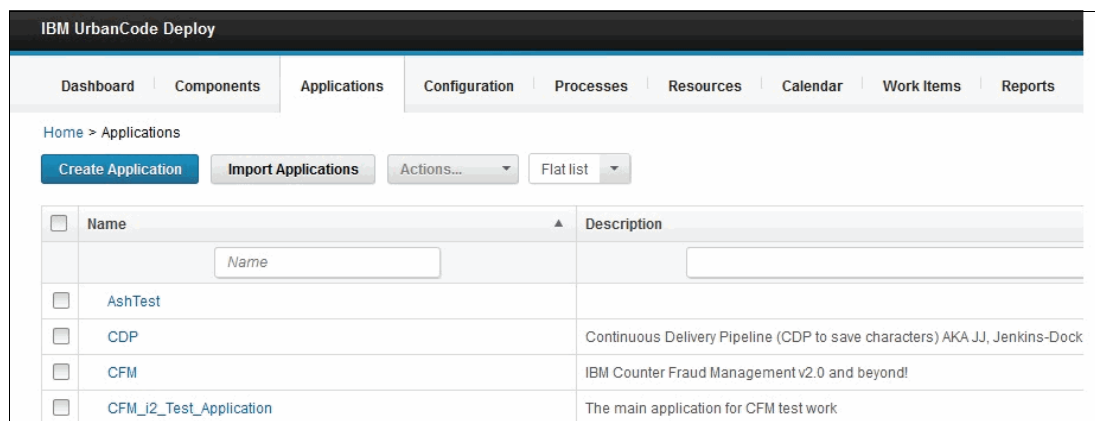


Figure A-25 Create an application

2. In the Create Application window, complete the following steps:
 - a. Enter the application name, which is **MQaaS-Demo-App** in this paper (in your own environment, you can use any name of your choice).
 - b. For Teams, select the **MQaaSRedPaper** team.
 - c. Enter the remaining information, as shown in Figure A-26.
 - d. Click **Save**.

Figure A-26 Create the MQaaS-Demo-App application

3. The new application appears in the application list, as shown in Figure A-27.

IBM UrbanCode Deploy			
Dashboard	Components	Applications	Configuration Processes Resources Calendar Work Items Reports
Home > Applications			
Create Application Import Applications Actions... Flat list			
<input type="checkbox"/>	Name	Description	Created
<input type="checkbox"/>	<input type="text" value="Name"/>	<input type="text"/>	
<input type="checkbox"/>	MQ-Databases	Contains the DB2 installs needed for MQ stacks	12/11/2014, 8:09 PM
<input type="checkbox"/>	MQ-Stacks	Creates build & test stacks for MQ	7/9/2014, 3:57 PM
<input type="checkbox"/>	MQaaS-Demo-App	Demo App for the MQaaS Red Paper team	7/21/2015, 8:36 PM

Figure A-27 List of applications created

Creating the environment for the application

After you created the application, you must set up the appropriate environment for the application by completing the following steps:

1. From the IBM UrbanCode Deploy web UI home page, click **Applications** → **MQaaS-Demo-App** → **Environment** and click **Create Environment**, as shown in Figure A-28 on page 181.

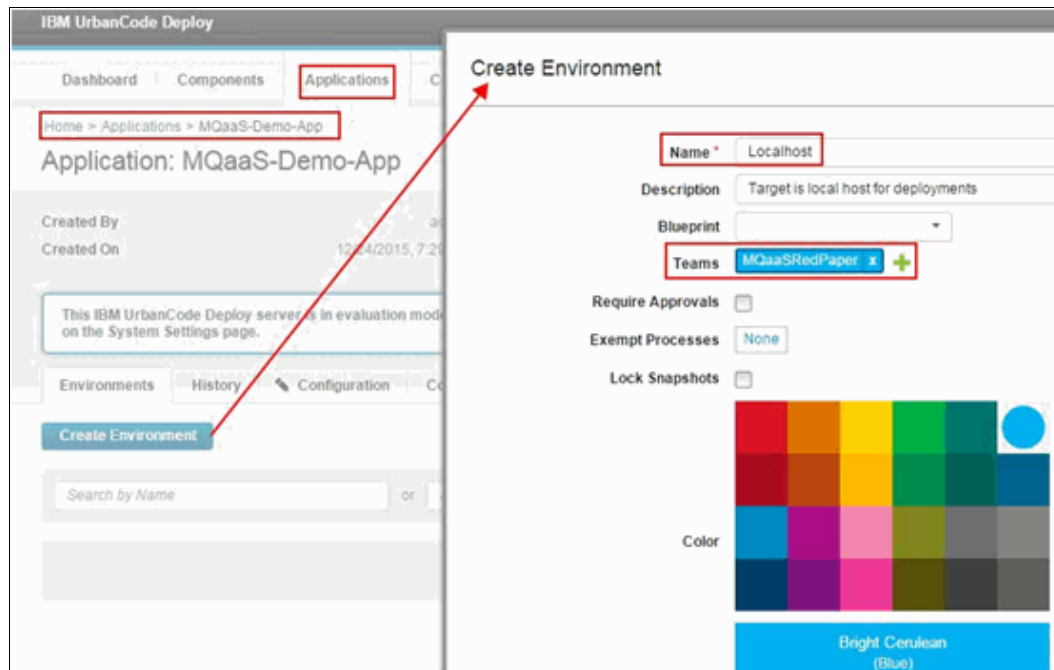


Figure A-28 Create the environment for the application

Note: It is a preferred practice to give the environment the same name as the server to which it maps, which is the server on which the application is. This approach is similar to the practice to name agents.

2. For Teams, select the **MQaaSRedPaper** team.
3. Enter the remaining information, as shown in Figure A-28.

Note: The environment name is the name of the target computer where the application is deployed. You can create several environments, one per computer, where the application will be deployed. Examples in other chapters of this paper show different environments for this reason.

Adding the resource to the application

After the environment is created, it appears in the list of environments. To add the resource to the application, complete the following steps:

1. From the environment list in the IBM UrbanCode Deploy web UI, click the new environment, which is **Localhost** in this example.

2. In the environment window, click **Add Base Resources**, as shown in Figure A-29.

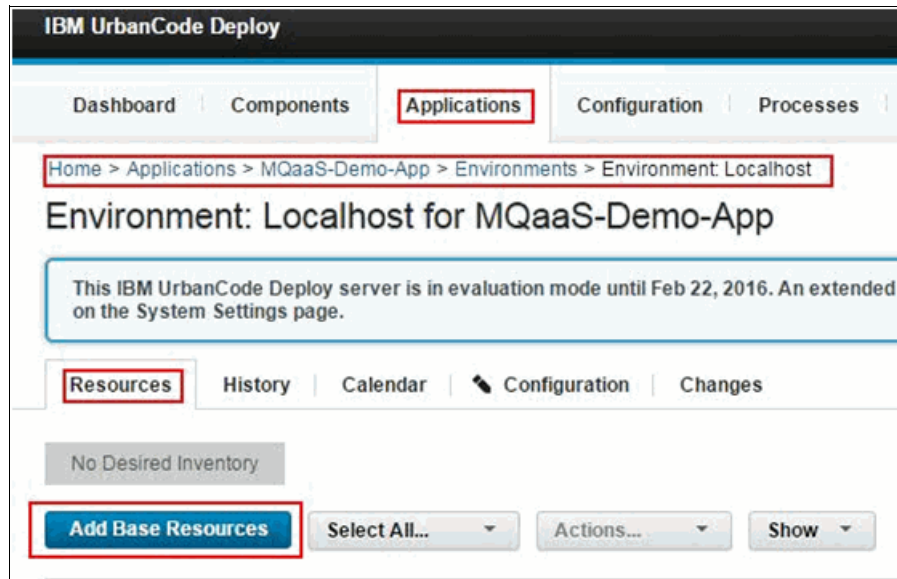


Figure A-29 Add a base resource to the environment of the application

3. The Add Base Resources wizard opens. Select all the items of the resource tree. In this example, select the **MQaaS** resource, which includes the Localhost agent and the MQaaS component. Select all the items and click **OK**, as shown in Figure A-30.

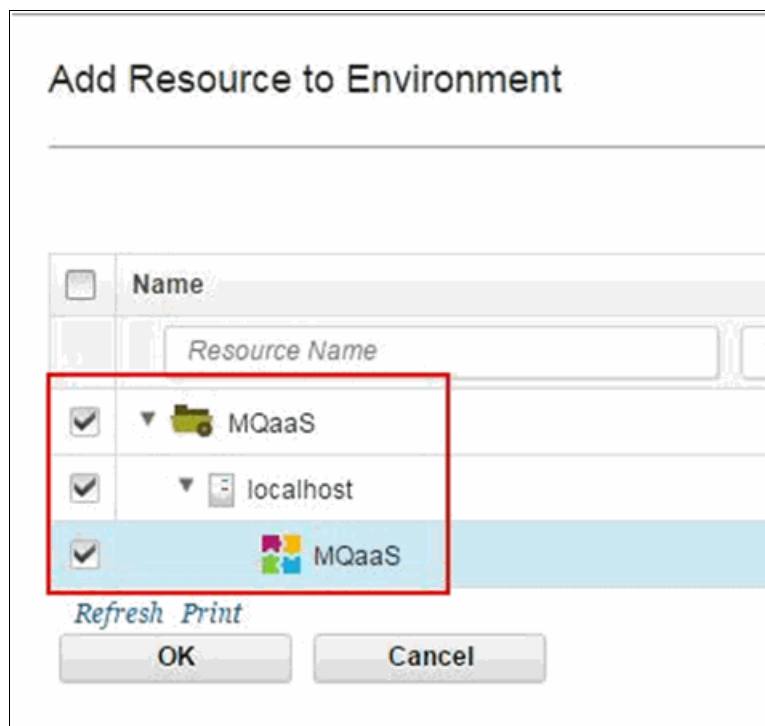


Figure A-30 Add the resource tree as the base resource to the application environment

4. Figure A-31 on page 183 shows the resources that are added to the environment.

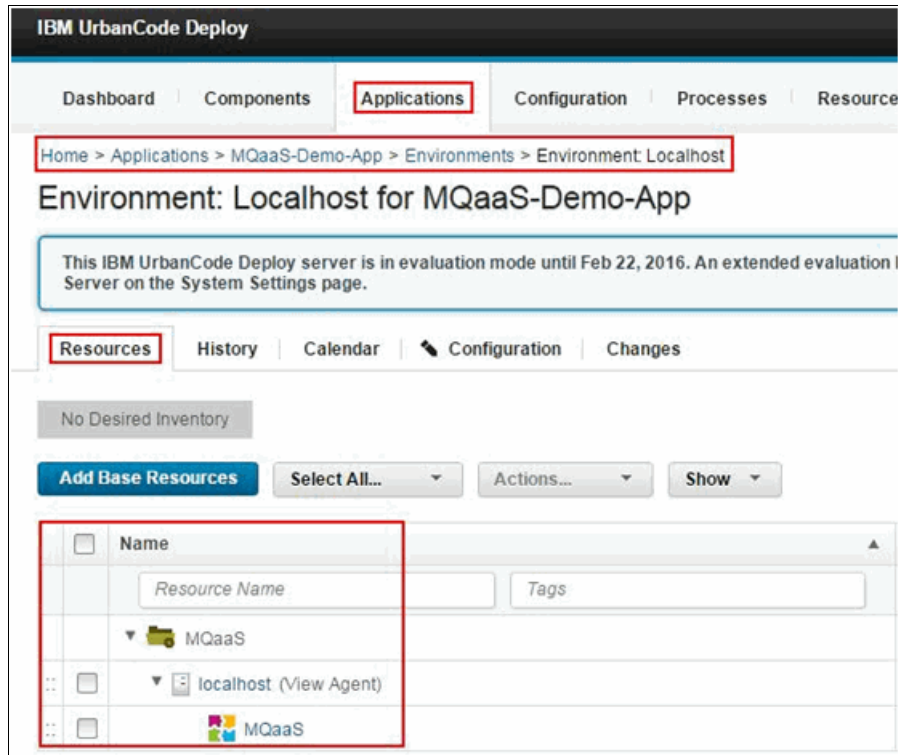


Figure A-31 Resources added to the application environment

Now, you can create individual application processes to help with the deployment of IBM MQ as a service, as described in Chapter 3, “Creating a self-service IBM MQ as a service menu for a portal” on page 21.

Related publications

The publications that are listed in this section are considered suitable for a more detailed description of the topics covered in this paper.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Some publications that are referenced in this list might be available in softcopy only.

- ▶ *IBM MQ V8 Features and Enhancements*, SG24-8218
- ▶ *IBM WebSphere MQ V7.1 and V7.5 Features and Enhancements*, SG24-8087
- ▶ *Integrating the IBM MQ Appliance into your IBM MQ Infrastructure*, SG24-8283
- ▶ *WebSphere MQ Primer: An Introduction to Messaging and WebSphere MQ*, REDP-0021

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

Online resources

These websites are also relevant as further information sources:

- ▶ Docker:
<https://www.docker.com/>
- ▶ IBM MQ IBM Knowledge Center:
https://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.helphome.v80.doc/WelcomePagev8r0.htm
- ▶ IBM UrbanCode Deploy V6.1.3 documentation:
http://www.ibm.com/support/knowledgecenter/SS4GSP_6.1.3/com.ibm.udeploy.doc/ucd_version_welcome.html

Help from IBM

IBM Support and downloads

ibm.com/support

IBM GLOBal Services

ibm.com/services



REDP-5209-00

ISBN 0738441457

Printed in U.S.A.

Get connected

