Creating Applications in Bluemix using the Microservices Approach

IBM Redbooks Solution Guide

Across 2014 and into 2015, microservices became the new buzzword for application development style. So what exactly are microservices?

Microservices is an architecture style in which large complex software applications are composed of one or more services. Microservices are narrowly focused, independently deployable, loosely coupled, language-agnostic services that fulfill a business capability. These multiple microservices, communicating with each other using language-agnostic APIs such as REST, work together to fulfill the business needs of the application.

These microservices are also applications in themselves and are often owned by small teams. Unlike the normal practice of a separate application support team, the team that developed the microservices is generally also responsible for their support.

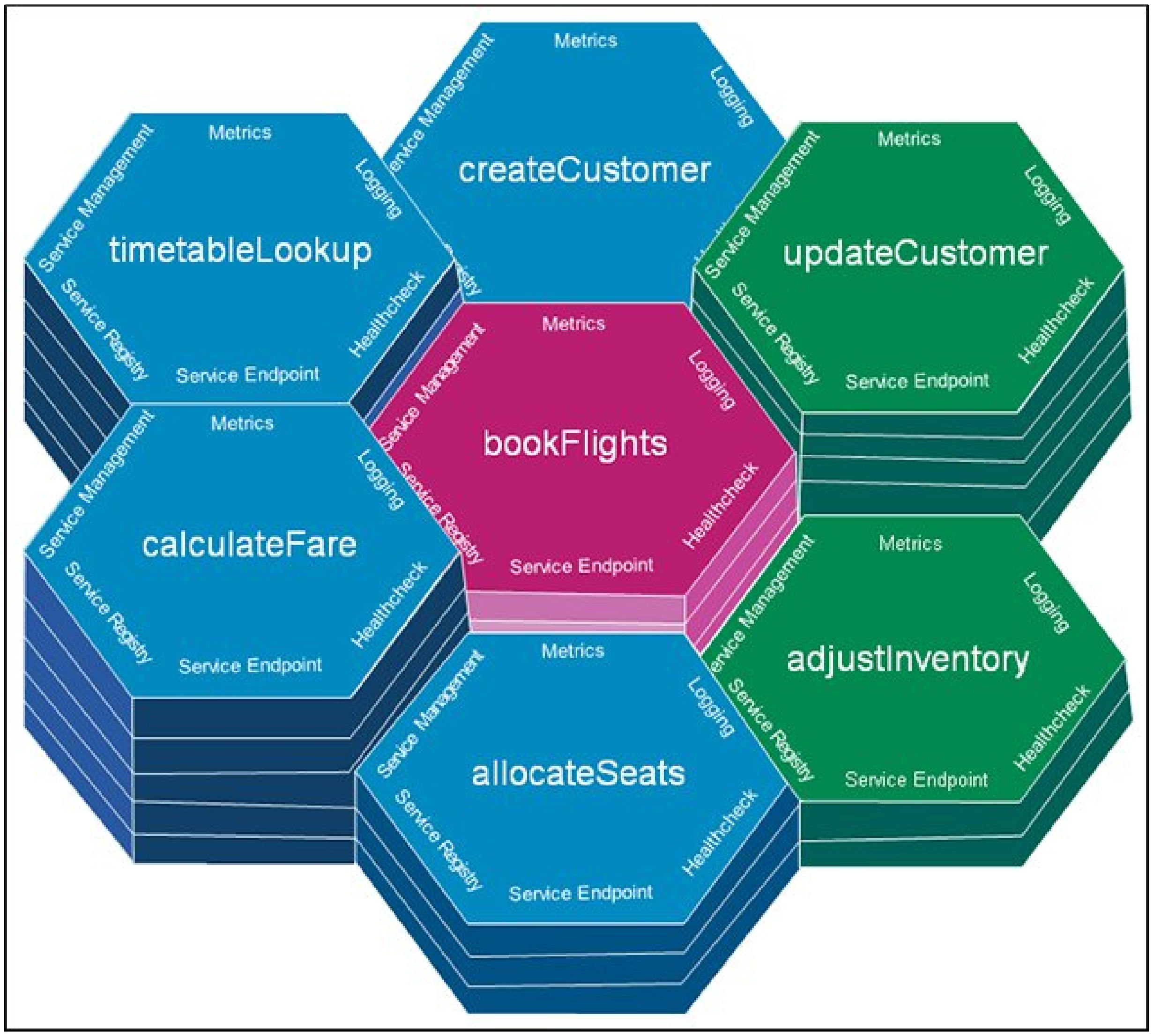

Figure 1. Microservices approach

This document describes how microservices solution addresses some of the business problems related to monolithic applications and why Bluemix is a great platform for developing applications in a microservices approach.

Did you know?

One question that is often asked is how small should these microservices be?

A good rule of thumb for sizing microservices is the two-pizza team rule: The teams that own the microservices shouldn't be larger than what two pizzas can feed! The reason for this rule of thumb is small teams make it easier to communicate more effectively among the team members.

Business value

Large monolithic applications were the de-facto standard for building large applications but practitioners noticed many problems with this approach:

- Difficult to maintain, modify, and become productive quickly, which results in long cycles of time-to-market to roll out new services
- Single deployment unit forces you to build/test/deploy the entire unit even for a small change
- Cannot scale a portion so you must scale the entire application
- Difficult to on-board new developers due to the large code base
- Managing code dependencies is a challenge
- Get stuck with a single choice of technology

To overcome this problem, a few leading architects have tried partitioning the application into small microservices, with each microservice working as an independent application and being owned by a small team.

Business owners saw the benefit of this approach quickly because they want their teams to be able to respond rapidly to new customer and market needs, but monolithic application development approach makes IT response slow. Microservices are much more aligned to business as they allow for more frequent delivery and faster delivery times. This allows business owners to get quicker feedback and adjust their investments.

Microservices also provide other business value:

- Smaller focused teams enables business owners to more easily manage resources more effectively, for example move them from low impact business areas higher impact areas.
- Enables a better user experience by allowing to scale individual microservice for removing bottlenecks.
- Easy determination and elimination of duplicate services, thus reducing development costs.

Solution overview

The microservices architecture pattern suggests partitioning the application into many small microservices with following guidelines:

- **Small and focused**: Microservices should focus on doing one task and doing it well. These microservices should be small enough so that you can rewrite the entire microservice easily if needed. Each microservice is an application in itself and should have its own source code management repository and its own delivery pipeline for builds and deployment.
- **Loosely coupled**: Loose coupling is an essential characteristic of microservices. You need to be able to deploy a single microservice on its own. There must be zero coordination necessary for the deployment with other microservices. This loose coupling enables frequent and rapid deployments.
- **Language-agnostic:** Using the correct tool for the job is important. Microservices should be built using the programming language and technology that makes most sense for the task at hand. Communication with microservices is by language-agnostic APIs, typically an HTTP-based resource API such as REST.
- **Bounded context**: Microservices do not know anything about other microservices surrounding them. This ensures that the microservices have clearly defined boundaries.

Figure 2 shows the microservices pattern. Compare it with the monolithic pattern in Figure 3.

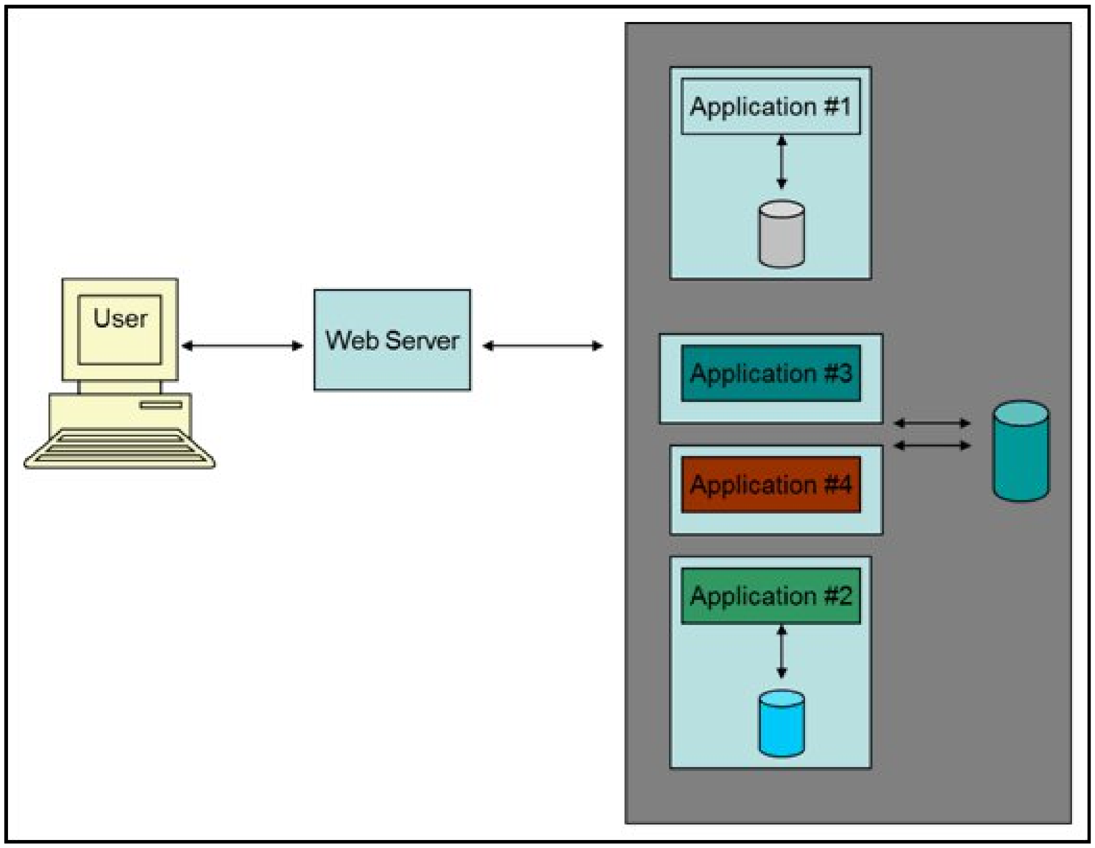

Figure 2. Microservices architecture with multiple languages and data store technologies

Figure 3 shows the monolithic pattern.

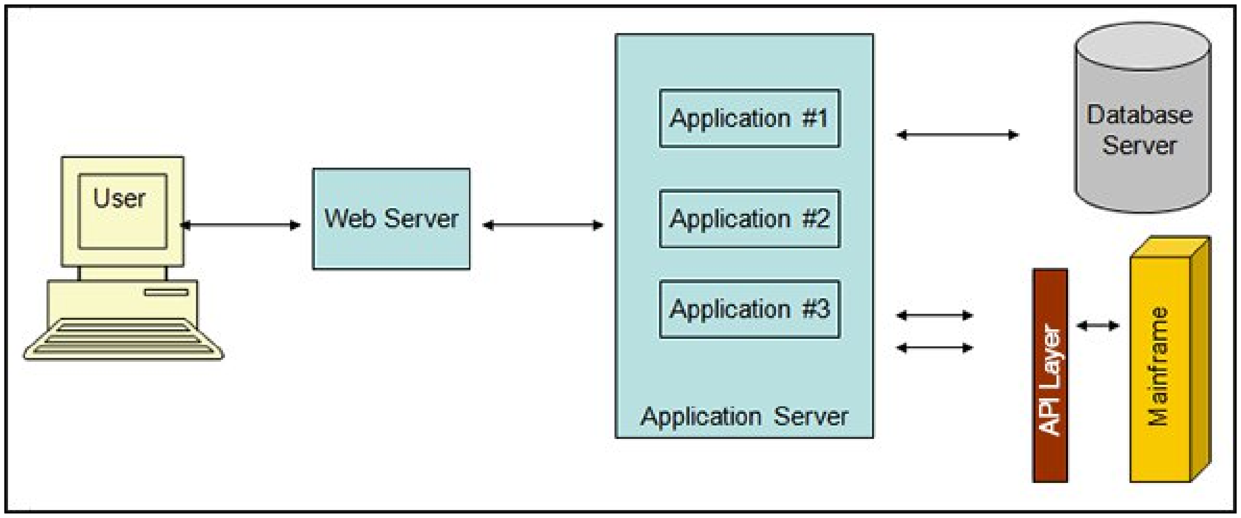

Figure 3. Typical monolithic application landscape

IBM Bluemix provides a well-rounded platform for building, running, and managing your applications that can alleviate much of the operational, networking, and other infrastructural requirements of the microservices architecture. Bluemix enables continuous delivery capabilities for application lifecycle management (ALM). Bluemix aims to simplify software development and management on the cloud by providing a complete set of flexible runtime environments, integrated services, and DevOps tools.

Solution architecture

IBM Bluemix is built on Cloud Foundry open source technology and provides a platform as a service (PaaS) environment for accelerating new application development. It also provides a DevOps toolset of concepts, practices, tooling, and team organizational structures that enable organizations to more quickly release new capabilities to their clients. Bluemix provides an array of runtime environments and pre-built services to rapidly create applications from a marketplace of IBM and third-party services.

Figure 4 shows a microservices application developed on Bluemix. IBM MQ Light for Bluemix is a cloud-based messaging service that provides flexible and easy-to-use messaging for Bluemix applications, including applications designed with the microservices approach. Applications using the REST API for inter-communication can use various runtime environments on Bluemix such as Liberty for Java or Node.js.

Note: REST and messaging do not have to be an either/or proposition. These two approaches can complement each other. In some cases, it can be optimal to combine applications doing REST calls with applications that use message transport to combine the two techniques in the context of microservices. REST can pose a problem when services depend on being up-to-date with data that they do not own or manage. Having up-to-date information requires polling, which quickly can tax a system, especially with many interconnected services. For real-time data, it is often desirable to have the data sent when it changes rather than polling to ask if it has changed. A polling interval might even mean that you miss a change. In these types of situations, a messaging protocol can be better than REST to allow real-time event updates.

Users interact with these web applications thorough browsers or mobile devices. These microservices can also communicate with other services on external systems.

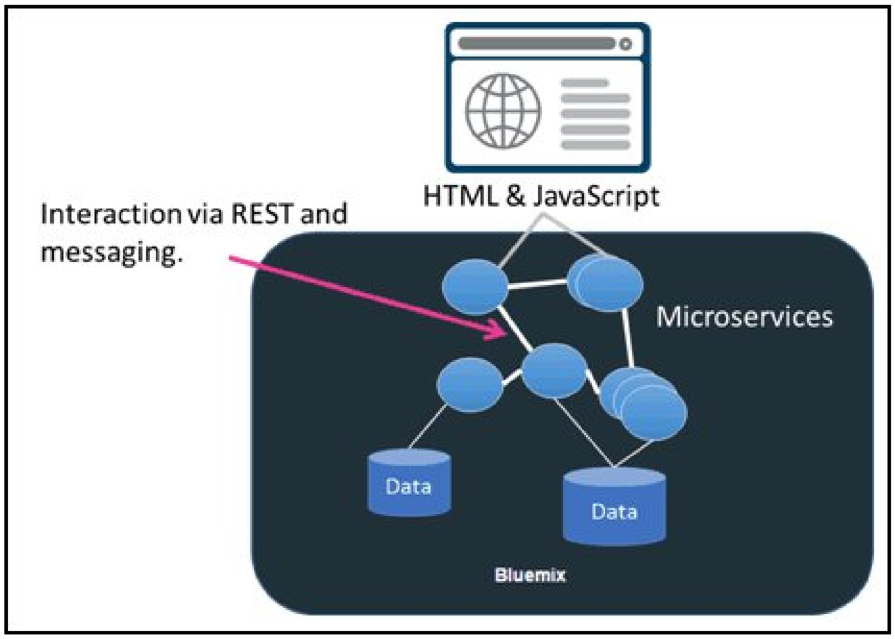

Figure 4. Developing applications using the microservices approach on Bluemix

Why IBM Bluemix DevOps Services is your best friend in a microservices implementation

When implementing a new business application in Bluemix, perform a top-down analysis and a functional decomposition to partition the application into services. These services can be created as individual microservices in Bluemix. When building microservices on Bluemix, you can treat each microservice as individual applications on Bluemix. Each application will have its own lifecycle and be managed independently. Each application typically should only have a small set of responsibilities, which usually results in a large set of applications. Usually this situation presents several operational and deployment challenges, which arewhere IBM Bluemix DevOps Services comes to the rescue. DevOps is an important set of practices and tools to improve efficiencies in software delivery. After deployment, there are additional tools for monitoring, logging, and analytics as well as the ability to easily scale up or down based on changes in the usage or load of your application.

Figure 5 shows a typical ALM cycle of a microservice application for a continuous delivery model.

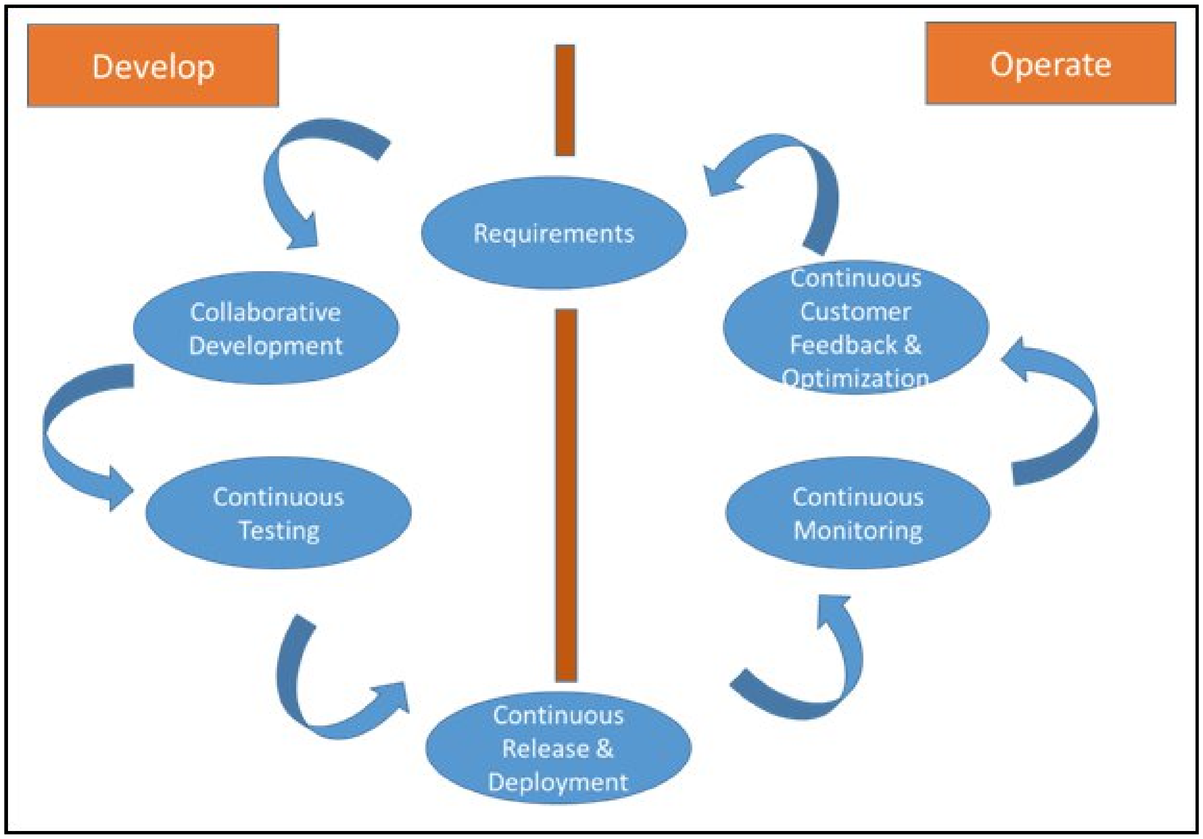
Develop
Operate
Requirements
Collaborative Development
Continuous Customer Feedback & Optimization
Continuous Testing
Continuous Monitoring
Continuous Release & Deployment

Figure 5. Management cycle of a microservice application for a continuous delivery model

For more information about how you can develop microservices applications on Bluemix, see Chapter 4 in *Microservices: From Theory to Practice*, SG24-8275.

Usage scenarios

A number of companies have adopted the microservices architectural patterns to address business challenges particular to their industries. This section describes some examples of microservices adoption.

A traditional brick and mortar retailer

A traditional brick and mortar retailer had a very large established Java services layer running on various platforms. The existing e-commerce systems were not designed to support mobile platforms nor to allow quick turnaround of changes required to adapt to business requirements. The e-commerce platform was monolithic, designed to construct a web page on the server and serve the entire experience to a browser. The legacy e-commerce layer was reaching end of life and the team decided to replace it with a platform more conducive to agile changes. The existing e-commerce platform had these issues, among others:

- High response time causing a diminished user experience
- Poor scalability
- Low reuse and high maintenance effort/cost
- High resources consumption

The IT team chose the microservices pattern and implemented the solution using newer technologies that are more amenable for microservices. If complex changes were required, the request would be passed to the legacy API until a new service in node.js could be created. This approach allowed for the in-place migration of individual services, more control over the user experience due to better ability to detect client disconnects, improved performance due to caching productivity, and faster deployments compared to traditional Java builds.

A large e-commerce company

The website for a large e-commerce company was originally built as a Ruby on Rails application, but as the company expanded, the single Rails codebase grew as well, becoming difficult to maintain and to incorporate new features. This original front-end layer exemplified the monolith pattern. The company embarked on a year-long project to migrate its US web traffic from a monolithic Ruby on Rails application to a new Node.js stack. At the initial phase of the transformation, the front-end layer was redesigned and split it into small, independent, and more manageable pieces. Each major section of the website was built as an independent application. The following are some of the benefits of this architectural transformation:

- Faster page loads across the site
- Faster release of new features with fewer dependencies on other teams
- Reuse of features in the countries where the e-commerce site is available

Integration

Bluemix platform has native integration with many IBM services such as DB2, BigInsights, and Watson. Your applications can therefore have immediate access to many leading applications from IBM and partners. In addition, you can connect your Bluemix apps to traditional IT (on-premises) systems with the IBM Cloud Integration for Bluemix service as part of a hybrid cloud solution. By using the Cloud Integration service, you can create a Cloud Integration API and publish the API as a private service for your organization.

Supported platforms

Bluemix has the following requirements:

- Hardware requirements: Accessing Bluemix requires an Internet connection and a browser.
- Software requirements: Accessing Bluemix requires an Internet connection and a browser.

Ordering information

This product is available by using Passport Advantage or IBM Cloud Services Agreement. It is not available as shrinkwrap. Ordering information is shown in Table 1.

Table 1. Ordering part numbers and feature codes

Program name	Program number	Charge unit description
IBM Bluemix Platform Subscription and Support	5725-S00	Pay Per Use, Per Month, Per Month with Support, Partial Month, Overage

Related information

For more information, see the following documents:

- *Microservices: From Theory to Practice*, SG24-8275
- *Bluemix Architecture Series: Web Application Hosting on IBM Containers*, REDP-5181
- *Bluemix Architecture Series: Web Application Hosting on Java Liberty*, REDP-5184
- *Accelerate Development of New Enterprise Solutions for the Cloud with Codename Bluemix*, REDP-5184
- *An introduction to the application lifecycle on IBM Bluemix video*
  http://www.ibm.com/developerworks/cloud/library/cl-intro-codename-bluemix-video/
- IBM Bluemix product page
  http://www-01.ibm.com/software/bluemix/
- IBM Offering Information page (announcement letters and sales manuals):
  http://www.ibm.com/common/ssi/index.wss?request_locale=en

  On this page, enter Bluemix, select the information type, and then click **Search**. On the next page, narrow your search results by geography and language.

Notices

This document was created or updated on July 29, 2015.

Trademarks