

IBM Cloud Private Practical Workshop



Application
Implementation
DevOps



IBM Cloud Private Use Cases

Use Case #1

Create new cloud-native applications

Use Case #2

Modernize and optimize existing applications

Use Case #3

Opening up enterprise data centers to work with cloud services



Characteristics of a cloud-native application

Built with the cloud in mind

- Use cloud services
 - E.g. storage, queuing, cache
- Have rapid and repeatable deployments to maximize agility
- Have automated setup to minimize time and cost for new developers
- Have clean contract with underlying OS to ensure maximum portability

Loose ties into corporate IT

- Security often specified using open standards
 - E.g. OpenID, OAuth
- Data might be local to the application itself

QoS attributes are those of the cloud

- It's always expected to be there, but sometimes sites and mobile apps become unavailable, although they must be restored quickly
- Applications must scale elastically without significant changes to tooling, architecture, or development practice
- Application must be resilient to inevitable failures in the infrastructure and application

Cloud-Native Application Goals

Horizontal scaling

- Application runs in multiple runtimes spread across multiple hosts (VIII)

Immutable deployment

- A runtime is not patched, it's replaced (IX)
- A runtime is stateless (VI)
- Shared functionality in backing services (IV)

Elasticity

- Automatic scale-out and scale-in to maintain performance
- Achieved via containerization

Pay-as-you-go charging model

- Pay for what you use

12 factors for the Impatient

- I. Codebase - use version control (e.g. git)
- II. Dependencies - use a dependency manager (e.g. gradle/maven/sbt)
- III. Config - separate configuration from code (use the OS environment)
- IV. Backing Services - reference resources such as DBs by URLs in the config
- V. Build release run - separate build from run. Use versions.
- VI. Processes - run the app as one or more *stateless* processes.
- VII. Port binding - app should be self-contained. No app server.
- VIII. Concurrency - scale horizontally
- IX. Disposability - fast startup, graceful shutdown
- X. Dev/Prod parity - keep environments similar
- XI. Logs - treat logs as event streams (no FileAppenders!)
- XII. Admin Processes - treat admin processes as one-off events

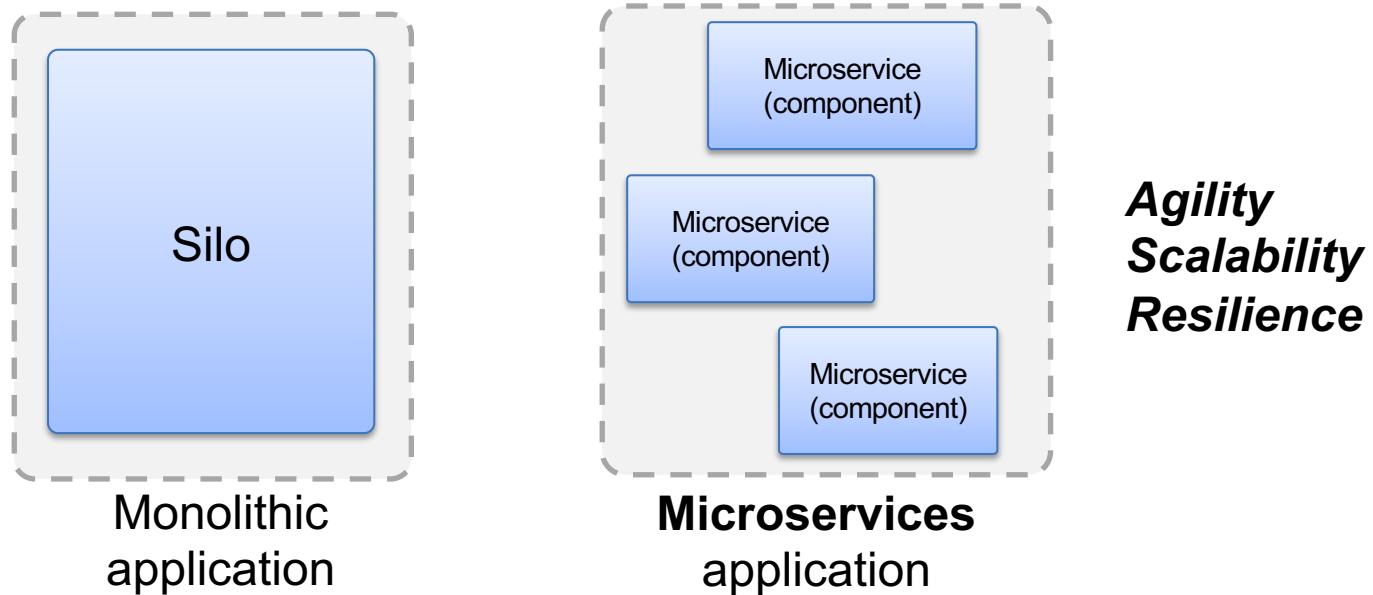
<https://12factor.net>

Microservices: Making developers more efficient

- An engineering approach that reduces an application into single-function modules
- They have well-defined interfaces that are independently deployed
- They are operated by a small team which owns the entire lifecycle of the service
- Microservices accelerate delivery by
 - Minimizing communication and coordination between people
 - Reducing the scope and risk of change

Microservices Architecture ?

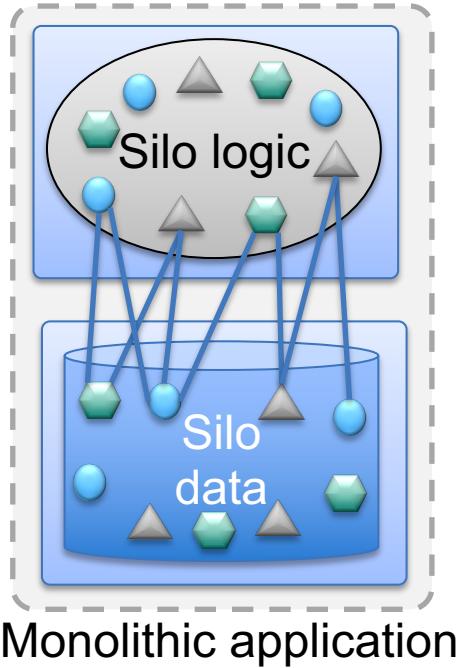
Simplistically, microservices architecture is about breaking down large silo applications into more manageable fully decoupled pieces



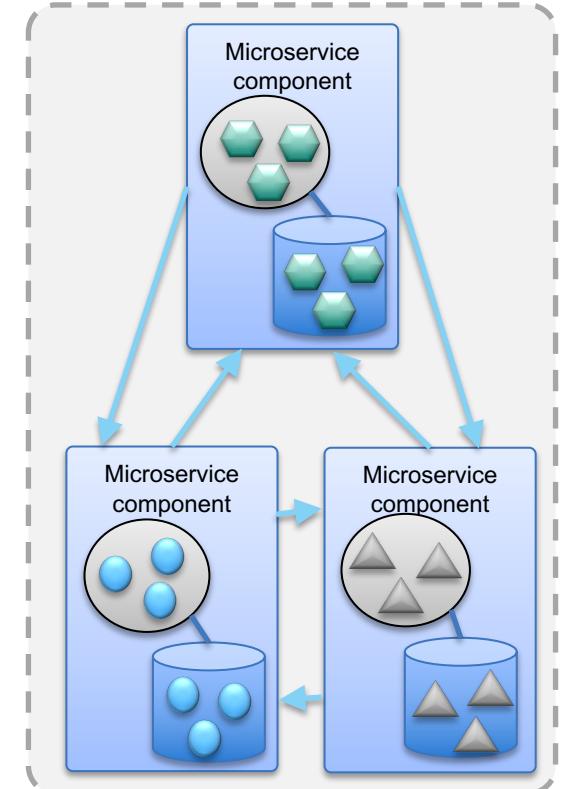
A *microservice* is a granular decoupled component within a broader application

With microservices, encapsulation is key

Related logic and data should remain together, and which means drawing strong boundaries between microservices.



Example operating
system boundaries



Microservices application

Why Microservices?

Small scoped, independent, scalable components

Scaling

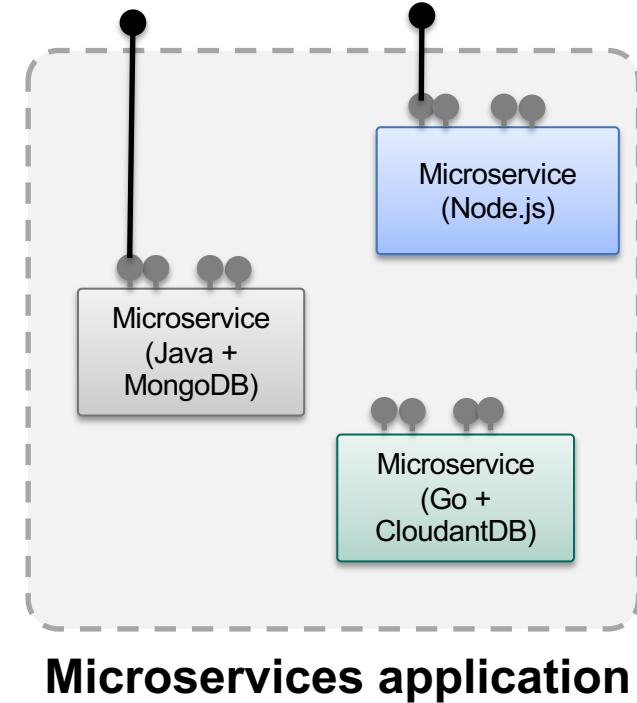
- Elastic scalability
- Workload orchestration

Agility

- Faster iteration cycles
- Bounded context (code and data)

Resilience

- Reduced dependencies
- Fail fast



Microservices application

Microservices inter-communication

Aim is decoupling for robustness

Compose a complex application using

“small”

independent (autonomous)

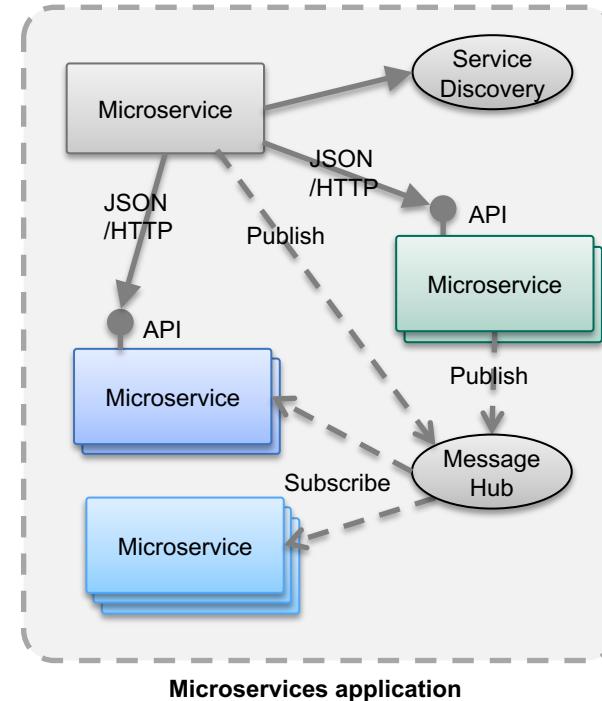
replaceable

processes

that communicate

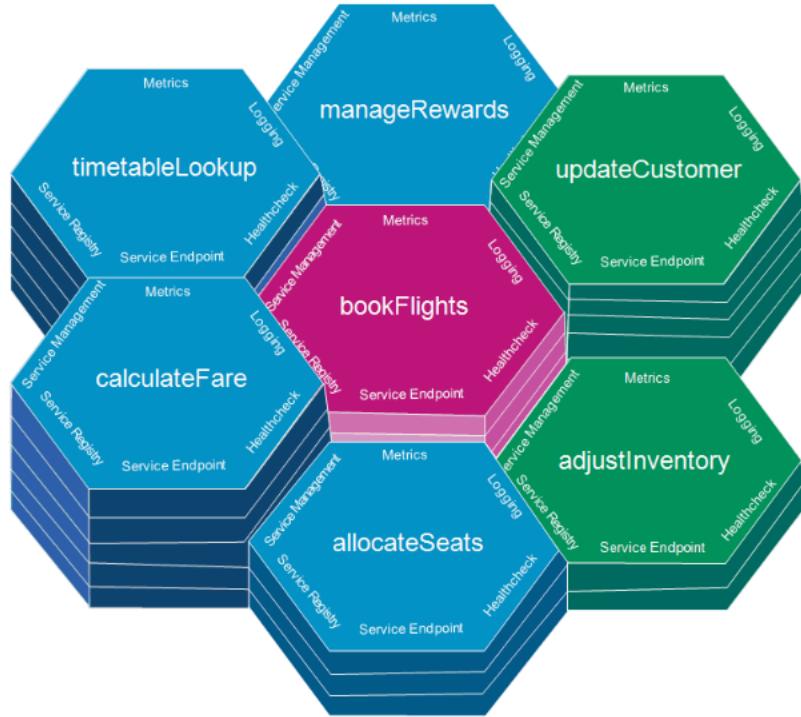
via language-agnostic APIs

synchronously and asynchronously



Sample application that uses microservices

- In this Airline reservation application, each service includes
 - Logging
 - Metrics
 - Health check
 - Service endpoint
 - Service registry
 - Service management
- Different Languages
- Sharing achieved through library sharing.
- Different number of instances



Advantages / Challenges of Microservices

Advantages

- Developed independently
- Developed by a single team
- Developed on its own timetable
- Each can be developed in a different language
- Manages its own data
- Scales and fails independently

Challenges

- Greater operational complexity
- Developers must have significant operational skills (DevOps)
- Service interfaces and versions
- Duplication of effort across service implementations
- Extra complexity of creating a distributed system:
- Designing decoupled, non-transactional systems is difficult
- Locating service instances
- Maintaining availability and consistency with partitioned data
- End-to-end testing

Comparing monolithic and microservices architectures

Category	Monolithic architecture	Microservices architecture
Architecture	Built as a single logical executable	Built as a suite of small services
Modularity	Based on language features	Based on business capabilities
Agility	Changes involve building and deploying a new version of the entire application	Changes can be applied to each service independently
Scaling	Entire application scaled when only one part is the bottleneck	Each service scaled independently when needed
Implementation	Typically entirely developed in one programming language	Each service can be developed in a different programming language
Maintainability	Large code base is intimidating to new developers	Smaller code bases easier to manage
Deployment	Complex deployments with maintenance windows and scheduled downtimes	Simple deployment as each service can be deployed individually, with minimal downtime

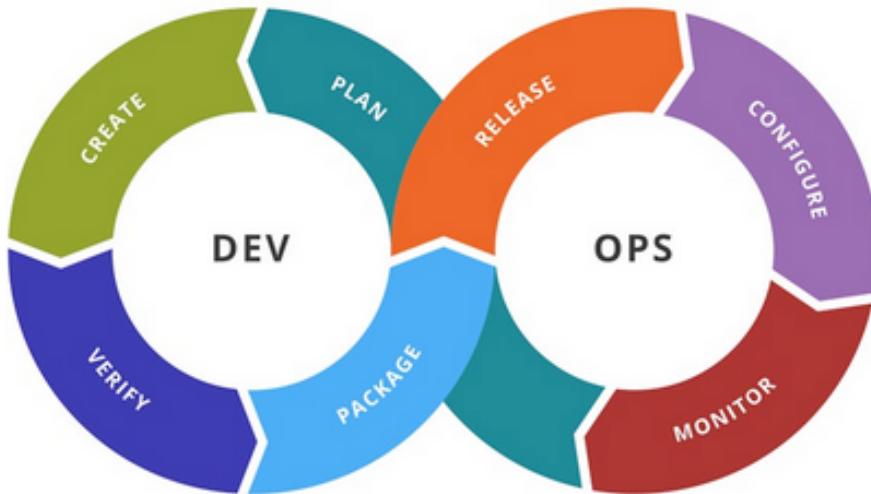
DevOps

DevOps is an approach that combines 2 major trends:

1. Agile infrastructure
2. Collaboration between development and operations

DevOps adoption drivers:

1. Agility & Speed
2. Security
3. Cohesion



.. the practice of operations and development engineers participating together in the entire service lifecycle, from design through the development process to production support.

Continuous Integration

Continuous integration is a software development practice where numbers of a team **integrate their work frequently**, usually each person integrates at least daily – leading to multiple integrations per day.

Each integration is **verified** by an automated build (including test) to detect errors as quickly as possible.

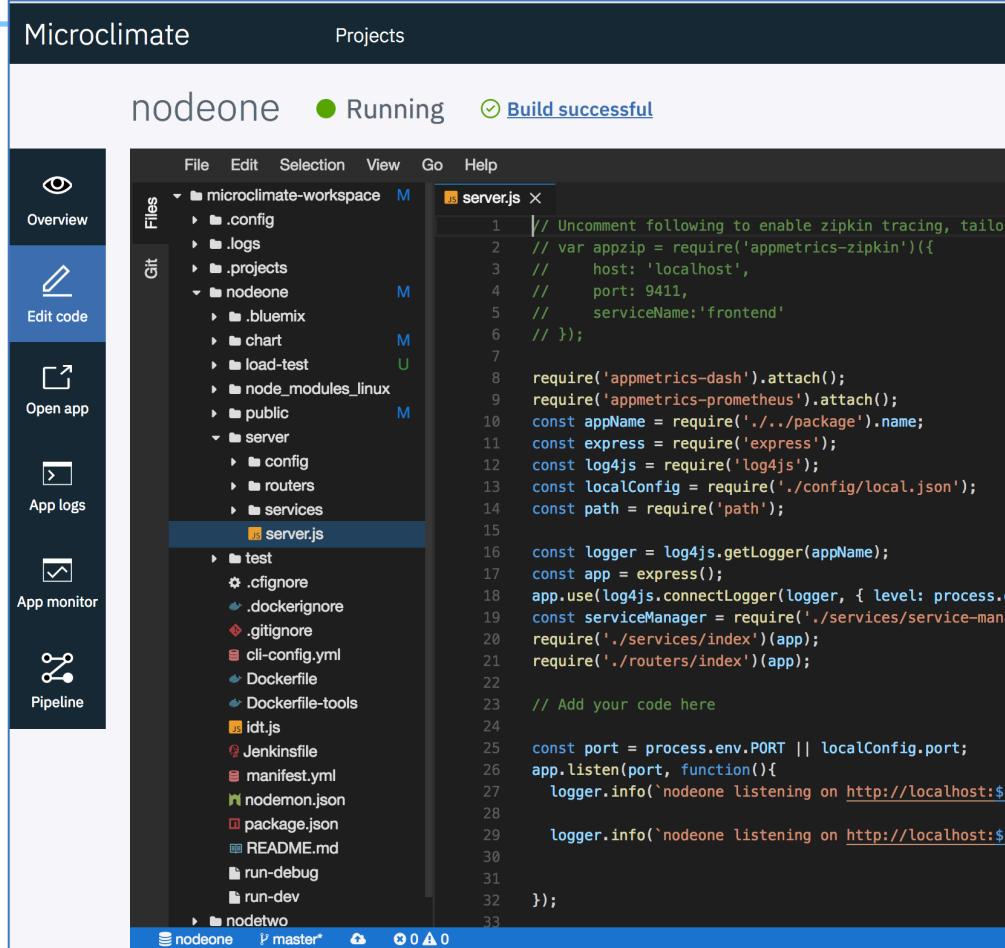
Martin Fowler

<http://martinfowler.com/articles/continuousIntegration.html>

What is Microclimate ?

Helping developers build services for IBM Cloud Private

- Service-oriented and scalable
- Unified development environment
- Cloud technology
- Built using the latest standards
- End-to-end Docker deployment
- Fast, flexible development
- Multiple IDE options
- Integrated DevOps PipeLine



The screenshot shows the Microclimate IDE interface. At the top, it displays "Microclimate" and "Projects". Below that, it shows a project named "nodeone" which is "Running" and has a "Build successful" status. The main area is a code editor with "server.js" open. The code is a Node.js script that sets up an Express app, connects to log4js, and requires various configuration files like .env, Dockerfile, and package.json. On the left, there's a sidebar with icons for Overview, Edit code, Open app, App logs, App monitor, and Pipeline.

```
// Uncomment following to enable zipkin tracing, tailo
// var appzip = require('appmetrics-zipkin')({
//   host: 'localhost',
//   port: 9411,
//   serviceName:'frontend'
// });

require('appmetrics-dash').attach();
require('appmetrics-prometheus').attach();
const appName = require('../package').name;
const express = require('express');
const log4js = require('log4js');
const localConfig = require('./config/local.json');
const path = require('path');

const logger = log4js.getLogger(appName);
const app = express();
app.use(log4js.connectLogger(logger, { level: process.
const serviceManager = require('./services/service-man
require('./services/index')(app);
require('./routers/index')(app);

// Add your code here

const port = process.env.PORT || localConfig.port;
app.listen(port, function(){
  logger.info(`nodeone listening on http://localhost:$
  logger.info(`nodeone listening on http://localhost:$

});
```

Benefits of using Microclimate

- Accelerated software delivery, from weeks to days to minutes, leveraging a **Continuous Delivery Pipeline**
- Simple and intuitive **user experience**, from development to production
- Support for **rapid** application development and testing cycles with greater agility, scalability, and security
- Reduced costs and complexity with seamless portability across popular cloud providers including public, dedicated, private, and hybrid.
- **Real-time diagnosis** and resolution of app infrastructure, minimizing downtime and maintaining SLAs
- **Easy connection** between existing applications to new cloud services (like Watson cognitive services) to discover actionable insights

What is Jenkins?



- Jenkins
 - Popular open-source framework for Continuous Integration
 - Monitors execution of repeated jobs; examples: cron jobs or builds
 - Generally used for:
 - Building/testing software projects continuously
 - Monitoring executions of externally-run jobs, such as cron jobs

Jenkins

Views

S	W	Name	Last success	Last failure	Last duration	Actions
5	W	please	15 hr (2023)	5 days 15 hr (2023)	2 min 3 sec	
5	W	please-merge	15 hr (2023)	1 mo 2 days (2023)	3 sec	
5	W	please-pull	15 hr (2023)	16 days (2023)	0.5 sec	
5	W	keystone	12 hr (2023)	13 hr (2023)	1 min 45 sec	
5	W	keystone-merge	12 hr (2023)	6 days 15 hr (2023)	4.6 sec	
5	W	keystone-pull	12 hr (2023)	6 days 15 hr (2023)	9 sec	
5	W	keystone-pull2	12 hr (2023)	6 days 15 hr (2023)	22 sec	
5	W	keystone-pull3	3 hr 44 min (2023)	10 hr (2023)	8 min 79 sec	
5	W	keystone-test	3 hr 44 min (2023)	21 hr (2023)	1 min 0 sec	
5	W	keystone-test2	3 hr 44 min (2023)	21 hr (2023)	1 min 9 sec	
5	W	quantum	9 hr 37 min (2023)	9 days 9 hr (2023)	9.1 sec	
5	W	quantum-test	9 hr 37 min (2023)	N/A	4.4 sec	
5	W	quantum-test2	9 hr 35 min (2023)	N/A	20 sec	
5	W	zabbix	21 hr (2023)	1 mo 17 days (2023)	14 sec	
5	W	math-exercise	21 hr (2023)	N/A	3.5 sec	

List of jobs being executed

Jenkins

Test Result

8 failures (1%)

All Tests

Package	Duration	Fail
com.complex.slow	0.1s	0

Continuous Delivery, Continuous Deployment

Continuous delivery is a software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time.

Continuous deployment is a similar approach that aims at automating deployments rather than having manual processes.

What is UrbanCode Deploy?



Deploy

- **What:** the deployable items - binary files, static content, MW updates, DB changes and configs
- **How:** by combining deployable items with processes to create components and designing applications that coordinate and orchestrate multicomponent deployments.
- **Where:** the targeted hosts and environments

Links about Microservices

- Building Microservices (O'REILLY)

<http://shop.oreilly.com/product/0636920033158.do>

<https://martinfowler.com/microservices/>

- Microservices Architecture (IBM)

<https://developer.ibm.com/architecture/microservices>

https://www.ibm.com/cloud/garage/architectures/microservices/2_1

<http://www.redbooks.ibm.com/abstracts/sg248275.html?Open>

<http://www.redbooks.ibm.com/redpapers/pdfs/redp5271.pdf>

- IBM Code (IBM)

<https://developer.ibm.com/code/technologies/microservices/>

Microservices from Theory to Practice

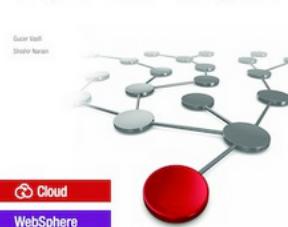
Creating Applications in IBM Bluemix Using the Microservices Approach

Shahir Daya
Nguyen Van Duy
Kameswara Eati
Carlos M Ferreira
Dejan Glotic
Vadif Guor
Manav Gupta
Sunil Joshi

Shishir Narain
Ramanan Venman



Creating Applications in Bluemix Using the Microservices Approach



Redbooks



IBM Solution Guide

