

LEARNING MADE EASY

BMC Special Edition

Accelerating DevOps Delivery Cycles

for
dummies[®]
A Wiley Brand



Apply a Jobs-as-Code approach

Go beyond automation

Save time and resources

Compliments
of



About BMC

BMC helps customers run and reinvent their businesses with open, scalable, and modular solutions to complex IT problems. Bringing both unmatched experience in optimization and limitless passion for innovation to technologies from mainframe to mobile to cloud and beyond, BMC helps more than 10,000 customers worldwide reinvent, grow, and build for the future success of their enterprises, including 92 of the Forbes Global 100.

Accelerating DevOps Delivery Cycles

**for
dummies**[®]
A Wiley Brand



Accelerating DevOps Delivery Cycles

BMC Special Edition

for
dummies[®]
A Wiley Brand

Accelerating DevOps Delivery Cycles For Dummies®, BMC Special Edition

Published by
John Wiley & Sons, Inc.
111 River St.
Hoboken, NJ 07030-5774
www.wiley.com

Copyright © 2019 by John Wiley & Sons, Inc., Hoboken, New Jersey

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without the prior written permission of the Publisher. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permissions>.

Trademarks: Wiley, For Dummies, the Dummies Man logo, Dummies.com, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates in the United States and other countries, and may not be used without written permission. BMC and the BMC logo are trademarks or registered trademarks of BMC Software, Inc. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc., is not associated with any product or vendor mentioned in this book.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND THE AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS WORK AND SPECIFICALLY DISCLAIM ALL WARRANTIES, INCLUDING WITHOUT LIMITATION WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES OR PROMOTIONAL MATERIALS. THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY SITUATION. THIS WORK IS SOLD WITH THE UNDERSTANDING THAT THE PUBLISHER IS NOT ENGAGED IN RENDERING LEGAL, ACCOUNTING, OR OTHER PROFESSIONAL SERVICES. IF PROFESSIONAL ASSISTANCE IS REQUIRED, THE SERVICES OF A COMPETENT PROFESSIONAL PERSON SHOULD BE SOUGHT. NEITHER THE PUBLISHER NOR THE AUTHOR SHALL BE LIABLE FOR DAMAGES ARISING HEREFROM. THE FACT THAT AN ORGANIZATION OR WEBSITE IS REFERRED TO IN THIS WORK AS A CITATION AND/OR A POTENTIAL SOURCE OF FURTHER INFORMATION DOES NOT MEAN THAT THE AUTHOR OR THE PUBLISHER ENDORSES THE INFORMATION THE ORGANIZATION OR WEBSITE MAY PROVIDE OR RECOMMENDATIONS IT MAY MAKE. FURTHER, READERS SHOULD BE AWARE THAT INTERNET WEBSITES LISTED IN THIS WORK MAY HAVE CHANGED OR DISAPPEARED BETWEEN WHEN THIS WORK WAS WRITTEN AND WHEN IT IS READ.

For general information on our other products and services, or how to create a custom *For Dummies* book for your business or organization, please contact our Business Development Department in the U.S. at 877-409-4177, contact info@dummies.biz, or visit www.wiley.com/go/custompub. For information about licensing the *For Dummies* brand for products or services, contact BrandedRights&Licenses@Wiley.com.

ISBN 978-1-119-56968-8 (pbk); ISBN 978-1-119-56967-1 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Publisher's Acknowledgments

We're proud of this book and of the people who worked on it. Some of the people who helped bring this book to market include the following:

Project Editor: Martin V. Minner

Editorial Manager: Rev Mengle

Executive Editor: Steve Hayes

Business Development

Representative: Molly Daugherty

Production Editor:

Tamilmani Varadharaj

Contents at a Glance

Introduction	1
CHAPTER 1: Exploring DevOps 101	5
CHAPTER 2: Jobs-as-Code in DevOps	13
CHAPTER 3: Application Workflow Orchestration in a CI/CD Pipeline	19
CHAPTER 4: Application Workflow Orchestration in a Containerized World	27
CHAPTER 5: Looking at Real-World Examples.....	31
CHAPTER 6: Ten Benefits of Application Workflow Orchestration and Jobs-as-Code.....	37

Table of Contents

INTRODUCTION	1
About This Book	1
Foolish Assumptions	2
Icons Used in This Book	2
Beyond the Book	3
Where to Go from Here	3
CHAPTER 1: Exploring DevOps 101	5
What Is DevOps?	5
What Is CI/CD?	6
What Is Application Workflow Orchestration?	9
CHAPTER 2: Jobs-as-Code in DevOps	13
Defining Jobs	13
Thinking of Jobs as More than Business Logic	14
Optimizing the Job Request Process	17
CHAPTER 3: Application Workflow Orchestration in a CI/CD Pipeline	19
Examining the CI/CD Pipeline	19
Adding Jobs-as-Code	24
Scaling Up Application Workflow Orchestration	25
CHAPTER 4: Application Workflow Orchestration in a Containerized World	27
Looking at the Container Pipeline	27
Developing Container Pipelines	28
Adding Application Workflow Orchestration	29
CHAPTER 5: Looking at Real-World Examples	31
Developing a Payment Processing Application	31
From support tickets to CI/CD	31
Adding Jobs-as-Code	32
Leveraging Big Data and DevOps to Fight Malware	33
Big data processing	33
Adding application workflow orchestration	33
Advanced integrations	34

A Leading Travel Technology Company Leverages	
DevOps and the Cloud.....	34
Batch jobs at scale	34
DevOps and cloud.....	35
CHAPTER 6: Ten Benefits of Application Workflow	
Orchestration and Jobs-as-Code.....	37
Faster Software Delivery at Scale	37
Increased Software Quality	38
Automated Testing.....	38
Repeatable Processes	38
Better Communication	39
Increased Agility	39
Greater Visibility	39
CI/CD Integration.....	40
Containerization	40
Native Languages	40

Introduction

DevOps is all about improving collaboration so businesses can speed application time-to-market without sacrificing quality. However, application workflow orchestration is often left out of the DevOps equation. Instead, developers use various basic tools to code jobs as they build apps.

What's wrong with this approach? The absence of consistent dev standards often leads to a failure to meet production standards. When something breaks, it's usually hard to find and fix.

Jobs-as-Code standardizes and automates job scheduling by embedding operational instrumentation, written using code-like notation like JavaScript Object Notation (JSON), YAML Ain't Markup Language (YAML), or Python, that uses an application programming interface (API) to invoke the services of a workflow orchestration solution. That code is managed throughout the continuous integration/continuous delivery (CI/CD) pipeline, the same way all other code that implements the application business logic is managed.

Application workflow orchestration determines when jobs run and what to do if a job fails. If developers are spending too much of their time defining this administrative functionality, they are re-creating the wheel, only to create a future maintenance burden for themselves and headaches for operations.

This book shows you how to apply a Jobs-as-Code approach to workflow orchestration to save time, conserve resources, reduce errors, and ensure consistency to accelerate DevOps in your organization.

About This Book

Accelerating DevOps Delivery Cycles For Dummies, BMC Special Edition, consists of six chapters that explore

- » The basics of DevOps, including its evolution, workflow orchestration, and Jobs-as-Code (Chapter 1)

- »» What comprises the jobs in Jobs-as-Code, how the approach works, and its benefits (Chapter 2)
- »» The CI/CD pipeline and application workflow orchestration in depth (Chapter 3)
- »» Containers and workflow orchestration (Chapter 4)
- »» Real-world use cases and success stories using workflow orchestration and Jobs-as-Code (Chapter 5)
- »» Ten key benefits of application workflow orchestration and Jobs-as-Code in DevOps (Chapter 6)

Foolish Assumptions

It's been said that most assumptions have outlived their usefulness, but we assume a few things nonetheless.

Mainly we assume that you are somewhat familiar with DevOps practices and some of the challenges of DevOps. You may be responsible for DevOps in your organization or looking to enhance DevOps and CI/CD practices in your organization.

If any of these assumptions describe you, then this book is for you! If none of these assumptions describe you, keep reading anyway. It's a great book and it'll accelerate your knowledge of DevOps.

Icons Used in This Book

Throughout this book, we occasionally use special icons to call attention to important information. Here's what to expect:



REMEMBER

This icon points out information you should commit to your nonvolatile memory, your gray matter, or your noggin — along with anniversaries and birthdays!



TECHNICAL
STUFF

You won't find a map of the human genome here, but if you seek to attain the seventh level of NERD-vana, perk up! This icon explains the jargon beneath the jargon.



TIP

Tips are appreciated, never expected — and we sure hope you'll appreciate these tips. This icon points out useful nuggets of information.

Beyond the Book

There's only so much we can cover in this book, so if you find yourself at the end, thinking, "Where can I learn more?" just go to www.bmc.com/control-m.

Where to Go from Here

If you don't know where you're going, any chapter will get you there — but Chapter 1 might be a good place to start! However, if you see a particular topic that piques your interest, feel free to jump ahead to that chapter. Each chapter is written to stand on its own, so you can read this book in any order that suits you (though we don't recommend upside down or backward).

- » Getting started with DevOps
- » Introducing continuous integration (CI) and continuous delivery (CD)
- » Enhancing DevOps with application workflow orchestration

Chapter 1

Exploring DevOps 101

In this chapter, you learn the basics of DevOps and how application workflow orchestration helps accelerate your organization's DevOps journey by ensuring the applications you deliver to production can run efficiently and meet business demands like governance and service levels.

What Is DevOps?

DevOps is a set of practices dedicated to building, delivering, and operating rapidly-evolving systems in close alignment with business objectives. Some key practices are prolific communication and collaboration among all participants in the software development life cycle (SDLC). But DevOps isn't about holding hands and singing "Kumbaya." DevOps focuses on creating an ongoing feedback loop of analyzing, building, and testing while leveraging automation to speed the entire software delivery process. To achieve this kind of seamless and constant loop of software building and testing, you need to create cross-functional teams that can work together effectively.



TIP

The key to DevOps functioning at optimal levels is engendering a culture of communication in which teams can coordinate among themselves and with other teams in an effortless manner.

An environment strongly committed to DevOps tends to produce applications that are modular and insulated by an application programming interface (API) layer. Each component can be developed, revised, and deployed on its own, and any issues within an individual component have only a minor impact on the entire software project. This approach allows new features and releases to be implemented more easily and simplifies rollbacks if necessary. Keeping each deliverable component to a smaller, more manageable size helps to maintain the quality of work while accelerating the speed at which changes can be made.

The SDLC is a process for producing software applications. Although opinions vary about the number and specifics of each step, this book focuses on the following phases:

- » **Code:** Includes code development and review, source code management tools, and code merging.
- » **Build:** Includes continuous integration tools and build status.
- » **Test:** Consists of automated testing tools that help to identify bugs and avoid unintended regression from previous versions.
- » **Deploy:** Includes continuous delivery tools and deployment status.



REMEMBER

DevOps is neither a silver bullet that will make all your problems go away nor a tool you can install and easily turn on like a light switch. DevOps helps to optimize your IT organization's process for software development along each step of the SDLC.

What Is CI/CD?

In a world where every company is a software company, competitive advantage comes through innovation in the improvements and changes to software-enabled business services delivered to end-users and customers. To increase the rate of innovation, organizations must ensure that the business and technical challenges in releasing new or enhanced services are mitigated. Close integration among the development, testing, and operations roles, as well as key business decision makers, is critical to achieving

these goals. IT has traditionally been tasked with “keeping the lights on,” but progressive organizations are driving unprecedented business opportunities by delivering the custom application features, services, and innovations that their end-users and customers demand. At the same time, these organizations recognize that traditional development and delivery methodologies fail to deliver low-risk, software-enabled business services at a rapid pace.



REMEMBER

Continuous integration (CI) and continuous delivery (CD) are software engineering practices aimed at developing and delivering software as quickly as possible, with the highest levels of quality.

CI involves the process of merging all coding works of a software development project on an ongoing basis. Typically, all code changes are committed to a centralized version control repository, such as Git. A process referred to as “build” is then initiated by tooling, such as Jenkins, either automatically, as a result of the commit action, or on some pre-defined basis (for example, hourly or once a day). The build process collects all the application components and ensures that requirements are satisfied and that references are resolved for the construction of the application. Some minimal testing may also be performed to ensure that the build results in a functional product.

Today, CI relies on automation tools and is coupled with a culture that drives rapid integration of iterative code development. If an error is introduced or a failure is detected, the entire product team pauses its normal work and focuses on identifying and correcting the problem. This cultural component is an integral part of any CI strategy — collaborating, communicating, and learning how to perform and merge small code changes faster requires a cultural shift at an individual and collective level within the organization. CI strategies encourage small and frequent code commits that can be integrated faster without breaking the resulting software functionality. Build tools run automated tests on the merged code to identify bugs early in the SDLC and ensure that application changes are structurally correct.

As defined at <https://continuousdelivery.com>, CD is “the ability to get changes of all types — including new features, configuration changes, bug fixes and experiments — into production, or

into the hands of users, safely and quickly in a sustainable way.” As an application proceeds along the SDLC, practices that contribute to achieving this goal include:

- » Dynamic infrastructure that can support comprehensive and realistic testing
- » Automated testing frameworks that can subject even a minor change to extensive regression and acceptance testing
- » Automated, rule-based customization of environment-dependent properties

An automated CI/CD process helps to accelerate innovation through a fast and efficient software release process. Secure and functional software updates are ensured through automated build and testing. Development, testing, and operations teams work together to resolve issues that arise within the delivery pipeline. IT shops are freed from manual tasks like solving complex bug fixes and resolving code dependencies that appear late in the software delivery process. Any code change that introduces a bug is identified immediately, and developers can collaborate to make changes accordingly. As a result, correct, functional, secure and improved software updates are delivered to end-users and customers faster. Organizations can quickly respond to market changes, cybersecurity issues, or new business opportunities. Unlike traditional methodologies that focus on delivering software updates to end-users and customers in weeks or months, automated CI/CD strategies aim to deliver working updates in a matter of hours or days.

To achieve these goals, an effective CI/CD strategy should include the following best practices:

- » **Leverage version control.** To ensure that every change to the software build is recorded and tracked, a version control system (VCS) should be used to track changes and provide a quick method for reverting to earlier, stable versions. An automated CI process can be achieved by triggering software integration and testing processes as the VCS is updated with a new code commit. Changes are documented accordingly to maintain a single version of truth as the build progresses through the development phase.

- » **Operate the infrastructure as code.** An effective CI/CD strategy requires comprehensive, highly iterative and automated testing in an environment that mirrors the target production environment as much as possible. Operating infrastructure as code enables infrastructure to be created easily, automatically, and consistently, and to then be discarded as soon as it is no longer required. CI/CD is almost impossible to achieve through any other management approach.
- » **Test constantly.** Perform comprehensive tests after every change, no matter how minor, to identify problems as early as possible. Early problem identification significantly reduces the time and effort required to apply fixes and directly contributes to higher overall quality of the final product.
- » **Keep it secure.** Take necessary measures to ensure optimum security of the CI/CD infrastructure, especially because the pipeline contains valuable data and may have access to production systems that are targets of deployment. Use advanced identity and access management (IAM) capabilities such as multi-factor authentication (MFA), virtual private networks (VPNs), and a layered approach to security, depending upon the risk of exposure.

What Is Application Workflow Orchestration?

So far, this chapter examines how modern applications are being built. Next, consider the individual components — what makes up an application:

- » **Business logic:** This is the major component that has the “instructions” that implement an application. It may be written in Java, Python, or another programming language.
- » **Infrastructure:** This includes the servers and networking on which the application executes.
- » **Configuration:** This may include firewall rules, memory and CPU requirements, and locations of databases.

» **Operational logic:** This includes dependencies and relationships with other applications or files, restrictions (for example, the application is required to only run together with another application or only if the other application is not running) and service-level commitments. This is referred to as *application workflow orchestration*.

Not all applications require workflow orchestration. For example, a web application that interacts with users and is expected to be always operational may only require some monitoring to ensure the application is healthy and available. Other applications, however, such as those in a data analytics pipeline, may require input from outside sources or data extracted from other systems. Before analytics are performed, it may be necessary to subject incoming data to validation or cleansing after the required data components have been assembled. After results have been computed, the output may have to be pushed to another application. Such a sequence of processing will likely require orchestration that controls the various processes and invokes the correct ones when their prerequisites have been satisfied.

An application workflow orchestration product should provide comprehensive capabilities, including the following:

- » **Arrangement:** How processes and tasks relate to one another within the business service they collectively deliver.
- » **Coordination:** How one application relates to other services that may have some indirect relationships with one another.
- » **Management:** The monitoring required for successful operation, such as ensuring that a predecessor process or event has completed successfully before invoking the next one in the sequence of steps, notifying when errors occur, and ensuring that service levels demanded by business requirements are attained.
- » **Oversight:** Ensuring proper authorization to perform functions, accumulating audit information about who did what and when, and collecting logs and output for both governance and problem analysis.

Organizations embracing a DevOps approach to application development and delivery should have the goal of managing all application components as code, not just business logic or infrastructure. Application workflow orchestration, for those

applications that require it, is an equally critical set of application artifacts and should be included in the CI/CD pipeline as code just like all other application components. Because workflow orchestration may have been the responsibility of IT operations teams in the past, it is particularly important to marshal organizational commitment to support this shift in mindset for developers and engineers to embrace workflow orchestration as code (also referred to as Jobs-as-Code).



REMEMBER

Developers and engineers may not consider workflow orchestration to be their responsibility. However, remember that DevOps includes an “Ops” component. Well-designed and well-running applications must include operational instrumentation and workflow orchestration to deliver effective and efficient operation that meets the requirements of the business.

Another measure of DevOps maturity is whether or not an organization manages jobs “as-code.” Jobs-as-Code is an approach to managing workflow orchestration within the SDLC just like any other code component of an application. Source code, or business logic, is managed with source code management (SCM) solutions. Infrastructure-as-Code is stored in the same SCM and managed with configuration management solutions. It’s a logical next step in embracing DevOps to include workflow orchestration in your SCM and manage that operational instrumentation as code. Doing so brings all elements of software delivery into one single workflow, as illustrated in Figure 1-1.

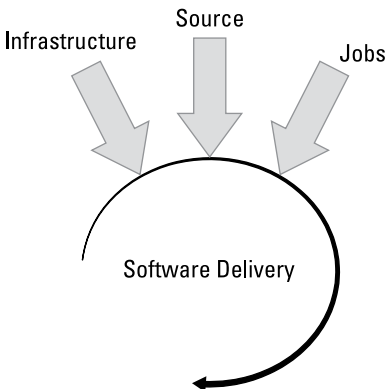


FIGURE 1-1: A software delivery cycle with infrastructure, source, and jobs all managed as code.



REMEMBER

Managing jobs as code means that developers can work on them earlier in the life cycle. Developers can manage jobs as just another code component of an application to be added to testing and deployment. With the right application workflow orchestration product, developers can test jobs and workflows with their testing frameworks, ensuring that deployment to production can be done with confidence.

- » Looking at the jobs in Jobs-as-Code
- » Seeing Jobs-as-Code in action
- » Exploring the benefits of Jobs-as-Code in DevOps

Chapter 2

Jobs-as-Code in DevOps

This chapter explores Jobs-as-Code: what it is and how it helps accelerate DevOps.

Defining Jobs

Jobs-as-Code is a DevOps approach that treats application automation management in the software development life cycle (SDLC) just like any other code components of an application. More specifically, it's a way of standardizing and automating job scheduling by embedding code, using a simple notation that makes application programming interface (API) calls to a scheduling engine. Calls are managed throughout the continuous delivery (CD) pipeline the same way Java or Python code is managed. As such, Jobs-as-Code is fundamentally no different from other code management practices.

To understand Jobs-as-Code, you need to know what a job is. *Jobs* are sets of commands or programs that are executed, usually on a schedule, or triggered by another activity, such as the completion

of a process. Jobs can be as simple or as complex as needed. Some examples include:

- » **Data pipelines:** Data movement and processing through the four main stages of any data pipeline: ingestion, storage, processing, and analytics.
- » **Extract, transform, load (ETL):** Moving data to a data warehouse by extracting and transforming it to the necessary schemas for the warehouse.
- » **Reconciliation:** Accounting reconciliation at the close of business for the day, or for another period, such as a month or quarter.
- » **Reporting:** Executing reports that run periodically, such as nightly or quarterly. Some reports place a burden on database resources and are thus run overnight or during off-peak hours.
- » **Billing:** Processes involving the generation of a large number of invoices or bills to send to clients.



TIP

When jobs are developed at the same time as the rest of the code, they can be tested with the main codebase. If jobs are created late in the process, changes to those jobs due to testing failures can affect the delivery schedule. Sometimes testing failures found late in the process can require extensive, costly modifications. By testing earlier and incorporating jobs as part of the regular testing process for software development, taking a Jobs-as-Code approach delivers benefits throughout the SDLC.

Thinking of Jobs as More than Business Logic

Jobs that manage the workflows of business applications are simply instrumentation. Traditionally, developers have relied on scripting to orchestrate their application workflows. Consider the following example:

1. A data pipeline ingests data from systems of record like ERPs or CRMs and from devices or social media streams.
2. The data is then stored in a data lake.

3. Based on different business group needs, the data is processed.
4. Artificial intelligence and machine learning models are applied to the refined data to provide insights to relevant business groups.

Here is a snippet of a data pipeline workflow in code format.

```
"IOT_Pipeline": {
  "Type": "Folder",
  "Comment" : "Folder for RX2 to create EMR
cluster and run Predictive maintenance analysis
based on vehicle sensor data",

  "IOT_Create_Cluster": {"Type" :
"Job:Script",
  "Description": "Launch EMR cluster
and deploy a Control-M agent",
  "FileName" : "launchEMR.bat",
  "FilePath" : "C:\\\\Prod_Stuff\\",
  "Host" : "controlm",
  "RunAs" : "Administrator"
},

  "IOT_CPSSetup": {"Type" : "Job:Script",
  "Description" : "Customize and deploy
connection profiles for dynamic EMR agent",
  "FileName" : "FY19_CPDeploy.sh",
  "FilePath" : "/home/BMC_Stuff",
  "Host" : "controlm",
  "RunAs" : "Administrator"
},

  "IOT_JAR_Setup": {
  "Type" : "Job:FileTransfer",
  "ConnectionProfileSrc" : "controlm",
  "ConnectionProfileDest" : "EMRhostSFTP",
  "Host" : "controlm",
  "FileTransfers" :
    [
      {
```

```

        "Src" : "C:\\BMC_Stuff\\
maintenance_data.csv",
        "Dest" : "/home/hadoop/
maintenance_data.csv",
        "TransferType": "Ascii",
        "TransferOption":
"SrcToDest"
    },
    {
        "Src" : "C:\\BMC_Stuff\\
lr-assembly-1.0.jar",
        "Dest" : "/home/hadoop/",
        "TransferOption":
"SrcToDest"
    }
]
},

```

This entire flow will pass through multiple applications and may run entirely in the cloud or in a hybrid-cloud/on-premises architecture. Developers may write the code for such a pipeline in Python, Java, or some other language, and then use scripting languages to orchestrate and run the jobs in this workflow based on an event trigger, a time-based schedule, or a combination of both.

Typically, operational scripting ends up taking more time (and arguably more code) than the business logic itself. When an application like this is delivered to operations to run in production, the scripts are examined and strung together in a more robust orchestration system that gives operations visibility to the flow of jobs for better error handling. They need precise alerting and notification when there is a failure or delay so they can recover quickly. There may be operational requirements to ensure completion of the workflows within a given business service-level agreement (SLA) as well. All this operational instrumentation usually happens at the end of the release cycle. This can be chaotic, and to meet go-live deadlines, some operational shortcuts may have to be taken. This is likely to create costly technical debt and service failures in the future. An application workflow orchestration solution can abstract this complexity.



TIP

In the preceding example, business logic is the code that ultimately performs the business function the application was intended to deliver.

The application workflow orchestration solution used in production should support a Jobs-as-Code approach and have the following operational capabilities:

- »» Easily support sophisticated workflow relationships
- »» Provide extensive application integration to support diverse platforms and technologies
- »» Enable operational insight into execution status and progress
- »» Display output and log collection
- »» Support service-level management, business-level abstraction, and full security
- »» Ensure audit and governance compliance
- »» Provide comprehensive end-to-end support of a fully automated release/delivery pipeline



TIP

For an application to deliver the greatest return on investment (ROI), the development process must be as streamlined and efficient as possible. The application should run in production with the fewest possible issues ensuring a positive customer experience. Taking a Jobs-as-Code approach is a key element in achieving both of these goals.

Optimizing the Job Request Process

Traditionally, developers aren't given access to the system used to orchestrate jobs in production. When they need to request a new workflow (or make changes to an existing job), they usually have to submit a ticket in the service desk system detailing the changes needed. This process often creates friction and delays in getting changes to production. For example, operations may send the request back to the developers directing them to modify the request to adhere to operational standards such as production naming conventions.

In an implementation where a Jobs-as-Code approach was adopted and has been used for a good length of time, developers can define jobs as code and have access to customizable templates that define the production standards. They can then store these jobs in a source code management system and CI/CD toolchain to build and deploy the job flow in the same manner as the rest of the application code.

A key concept in DevOps is to ensure that development and operations teams work together through the entire life cycle of the product. Everything required for an application to be production-ready should be baked into the SDLC. Terms like *DevSecOps*, which simply means security should be considered early in the development life cycle instead of at the end, are starting to be used as a result of this concept. The same applies to the orchestration and operational readiness of the jobs that will orchestrate the workflow. Jobs-as-Code is modeled after a prominent concept called Infrastructure-as-Code. The focus of Infrastructure-as-Code is defining, running, and managing jobs through machine-readable definition files, rather than interactive tools.



REMEMBER

Keep in mind that even the most rigorous development practices occasionally fail. Jobs-as-Code adds visibility to applications so that, when necessary, operations and support teams can more quickly identify, analyze, and resolve problems to make the application available again.

- » Looking at the CI/CD pipeline
- » Going beyond automation with application workflow orchestration

Chapter 3

Application Workflow Orchestration in a CI/CD Pipeline

Continuous integration (CI) and continuous delivery (CD) are processes within the software development life cycle (SDLC) designed to enable rapid and robust software development. Both processes follow the same direction within the SDLC pipeline but end at different intervals. In this chapter, you learn about the CI/CD pipeline and application workflow orchestration.

Examining the CI/CD Pipeline

Continuous integration involves merging all coding works of a software development project on an ongoing basis. For example, committing all code changes to a centralized repository can be considered as a simplified version of CI. The concept is further enhanced using automation tools and processes in a DevOps culture that drives rapid integration of iterative code developments.

The build is therefore available at a single, accessible machine location for further testing.

The cultural component is an integral part of a CI strategy — collaborating, communicating, and learning how to perform and merge small code changes faster requires a cultural shift at the individual and collective levels within the organization. CI strategies encourage small and frequent code commits that can be integrated faster without breaking the resulting software functionality. The build servers run automated tests on the merged code to identify bugs early in the SDLC pipeline, as well as validating and delivering new application changes to end-users.

Continuous delivery extends CI to incorporate automated software releases within the SDLC pipeline. The builds with continuously integrated code changes are automatically released to production after initial testing (such as automated unit tests). At the production stage, the software build is available for in-depth testing and therefore ready for production, although a release may require further manual approval for business or technical reasons. If the release process is also automated, the process is called *continuous deployment* (rather than continuous delivery).

Several common components exist across the CI/CD pipeline, including:

- » **Source control:** Developers check out code from a source control system, make changes, and check the code back into the source control system. In this way, developers can collaborate on code and track changes to it.
- » **Build tools:** Build tools automatically kick off processes after code changes are committed to a particular branch, such as “testing” or “integration.” The tools and commands to merge and compile the code vary across programming languages. Some languages do not require a compile step.
- » **Testing tools:** Automated test tools come in many forms, from those that accept plain-language test definitions to more complex back-end testing frameworks. These tests include internal acceptance testing, which may also have non-functional requirements, such as performance testing.

- » **Deployment system:** Deployment tools come in many forms. And, just as some programming languages may not require a compile step, deployment tools are dependent on the type of application being deployed.

The CI/CD pipeline doesn't typically begin with an end-to-end, fully integrated process. Instead, organizations move incrementally toward the goal of CI/CD. The following best practices will help you implement an effective CI/CD strategy for your organization:

- » **Operate the infrastructure as code.** An effective CI/CD pipeline requires the infrastructure to be adaptable and consistent with the production environment while preserving the integrity of configurations, as resources are provisioned dynamically and automatically. Any configuration drift affects the repeatability of the testing and deployment process and therefore prevents true continuity within the SDLC pipeline.
- » **Maximize version control.** To ensure that every change to the software build is meaningful and successful, a version control system (VCS) can be used to track the changes and revert to earlier deployments as necessary. An automated CI process can be achieved by triggering software integration and testing processes as the VCS is updated with a new code commit. The changes can be documented accordingly to maintain a single version of truth as the build progresses through the development phase. Additionally, it is beneficial to limit the branching in the VCS to reduce the possibility of a branch not being tracked for code updates and testing.
- » **Maintain a consistent deployment process.** A cultural, as well as tooling, change may be necessary to ensure that developers adhere to a standardized process for code commits. The build process should also be consistent throughout the pipeline. For example, build unique binary artifacts and reuse the result throughout the SDLC pipeline. By avoiding packaging software multiple times in different versions simultaneously between disparate teams, you can ensure inconsistency will not be injected into the final software product delivered to end-users.

WHAT IS CONTINUOUS TESTING?

Cutting corners isn't an option for enterprises that value the quality of their services and the customer's experience. Slowing down isn't, either. It defeats the purpose of DevOps. The only way to achieve CI/CD while putting out healthy and stable services is through continuous testing.

Continuous testing takes advantage of automated tests as a core piece of the software delivery pipeline to get feedback throughout the SDLC. Continuous testing is an automated end-to-end testing solution that integrates into your existing development processes. Modern application development and delivery pipelines have resulted in deployment cycles with less time between deliveries as well as more complex software releases. Continuous testing is implemented into the development process so that quality assurance (QA) happens every step of the way to ensure risk is constantly measured and mitigated.

The primary goal of continuous testing is assessing business risk coverage by providing instant insight into the overall health of each release candidate. Embedding testing into the software development process from beginning to end ensures that issues are found sooner and are more readily manageable. Continuous testing is seamlessly integrated into the software delivery pipeline and DevOps toolchain. The pursuit of continuous testing is to eradicate bottlenecks completely by performing the right tests at the right stages of each development cycle.

Continuous testing delivers actionable feedback for each step of the development process. Realistically assessing the end-user's experience provides invaluable feedback that helps DevOps teams ensure that the user experience is protected without slowing down the SDLC. Achieving all of this requires the implementation of automated tests that work together with DevOps development tools to integrate directly into each stage of the process.

Continuous testing is a cultural shift from testing at the end, to testing early, often, and at all stages of development, with the utilization of automation wherever possible. The approach to testing should be systematic and, as with all processes in DevOps, should be in a constant state of improvement. The beauty of speeding up the SDLC is that it gives DevOps teams more experience dealing with each stage

of the process, which enables them to have deeper insight into ways the process might be improved. The goal of DevOps teams and continuous testing is to improve constantly while looking for ways to optimize each step of the process.

The primary goal of continuous testing is assessing business risk coverage. In practice, this means that the information you glean from your automated tests must be actionable data that is meaningful enough to inform deployment decisions. Risk assessment requires low-level details as well as high-level information that can be used as data for supporting deployment decisions. If your tests aren't affecting the business decisions you make, then your tests aren't telling you enough.

Continuous testing should focus on the user's experience and whether or not changes have affected not only performance, but also functionality. Protecting the end-user experience is paramount when it comes to DevOps deployment schedules because changes should be released rapidly. Quick deployments can result in quickly breaking your services if you don't implement continuous testing throughout the process from start to finish. Tests should be broad enough to detect the impact of changes made on the user's experience and the functionality of the application.

Risk assessment performed by continuous testing practices should cover risk mitigation tasks, technical debt, quality assessment, and test coverage optimization. This ensures builds are ready for the next step of the process before they move on. Continuous testing should also test for policy compliance. The information provided by continuous testing should be actionable and relevant to the software's current stage in the SDLC. This allows fixes to be performed right away without allowing issues to corrupt steps later in the process and require additional backtracking to address.

Various tools that are available for DevOps practitioners can be invaluable additions to your technology stack. This is true for continuous testing systems as well. Tools like GitHub and Selenium are open-source options that help with testing the functionality of code every step of the way. DevOps teams can utilize the same automated tests regardless of where the changes are within the development life cycle. Continuous testing provides unparalleled access to actionable data that helps prevent the customer experience from degrading while also providing detailed information for business decisions.

- » **Test early and often.** Perform small and faster testing procedures early during the SDLC pipeline to identify problematic changes before it's too late. This means that the SDLC teams must prioritize testing, usually starting with unit tests, followed by integration tests, system tests, and acceptance tests. Developers may run some tests locally before applying code changes and therefore detect issues before the code is integrated within the centralized repository. Run the tests in containers to standardize the test environment and enhance portability of the testing infrastructure.
- » **Security.** Ensure optimum security of the CI/CD infrastructure, especially because the pipeline contains valuable data and access to deploy code changes to a centralized repository. Using advanced identity and access management capabilities, virtual private networks (VPNs) for access, and multiple layers of security may be necessary, depending upon the risk exposure.

Adding Jobs-as-Code

Event-driven workflows including batch and micro-batch jobs can have elements from numerous external systems. The Jobs-as-Code approach allows jobs to be processed through a CI/CD pipeline.

As the CI/CD pipeline in an organization matures, tasks naturally move toward development and become part of a developer's task list or backlog. However, because batch jobs have so many different parts from numerous systems, creating jobs as code requires a mature DevOps organization. In such an organization, developers and operations work together to create the batch job.

Developers may create the code for the batch job using their knowledge of the business requirements and the application being developed. Operations supplies the environment, possibly including data elements, credentials, and other system-level items.



TIP

Jobs-as-Code (discussed in Chapter 2) enables job scheduling to be standardized and automated by embedding code using a simple notation that makes application programming interface (API) calls to a scheduling engine, then manages these calls throughout the CI/CD pipeline.

Scaling Up Application Workflow Orchestration

With application workflow orchestration in place, facilitating the repeatability of processes within the orchestration becomes the next challenge. As more tasks are added to the workflow, coordination becomes a challenge.

Workflow orchestrations become cumbersome when more time is spent managing the orchestration than developing the application. Therefore, a tool that can scale to meet the needs of many complex tasks is vital to success.

Some characteristics to look for in an application workflow orchestration solution include:

- » **Flexible control:** The tool should enable multiple members of the DevOps team to work with the tool.
- » **Customization:** It should enable customizations to fit the organization rather than the organization changing its processes to fit the solution.
- » **Visibility:** Task and process status, including both success and failure, should always be visible.
- » **Scalability:** The application workflow orchestration solution should be able to scale up to support complex and simultaneous workflows.



REMEMBER

Finding application workflow orchestration solutions that support the complexity needed by a mature DevOps organization is difficult. Evaluating products means putting them into real-world situations through proof-of-concept and parallel processing, all of which is time consuming. Ideally, a solution should have a proven track record so that setup is well documented and supported.

IN THIS CHAPTER

- » Exploring the container pipeline
- » Building standard container pipeline processes
- » Bringing application workflow orchestration into the Jobs-as-Code pipeline

Chapter 4

Application Workflow Orchestration in a Containerized World

Containers have revolutionized the virtualization market by segmenting processes, drastically increasing portability, and saving space and time. These lightweight, safe spaces coincide beautifully with the DevOps methodology by bringing cohesion and clarity to all stages of the build, test, and deploy model of computer coding. In this chapter, you learn how to bring application workflow orchestration to the container pipeline.

Looking at the Container Pipeline

A pipeline is a workflow strategy that uses automation to produce software in an effective and timely process. First, automated tests are generated with every code change or check-in. Next, a code analysis runs. If the code makes it through the quality control gates and testing, then automatic deployment is initiated. Finally, acceptance tests run against the code. When applied optimally, new automation begins at every stage of code check-in. Only one

build is tested at a time. If a segment goes red or status updates occur, developers are notified from source control to end-user. This process prevents compound errors that can occur through integration and testing of the application's smallest components.



REMEMBER

Pipeline or “pipelining” breaks continuous integration and continuous delivery (CI/CD) down into the individual stages of the build. This allows developers to incrementally test, assess, and redeploy computer code without impeding workflow throughout the project.

Developing Container Pipelines

Standard processes for developing container pipelines allow you to implement a pipeline that's effective and useful. The details of your processes will depend upon the size of your team, your coding language, the platforms used, and the individual projects themselves. However, by establishing standard processes, you can keep your pipeline clear, managed, and stable. A typical four-stage process includes the following stages:

- » **Commit and build:** Newer versions of the code are submitted into the CI/CD system. The code is containerized and tested. When the code has passed all tests, it moves to the next stage.
- » **Automated acceptance:** Tests for basic functionality are executed. These tests should be done in an environment similar to the production environment. Issues found should be addressed before moving on.
- » **Continuous deployment:** Additional tests, such as functional, regression, and stress, occur.
- » **Production release:** Fully tested code is released into production.



REMEMBER

The timeframe of the development stage often depends on the business and its goals. By establishing standard processes, robust software can be distributed to the end-user more efficiently.

Adding Application Workflow Orchestration

Workflows should be treated the same as all other code components of an application, meaning they should be written early on, together with all other coding tasks. This is a fundamental component of a Jobs-as-Code approach.

When applications are containerized, the application workflow orchestration handles some additional technical considerations. For example, an application might consist of three images called A, B, and C. When running an instance of the application, the workflow requires that container A1 is instantiated from Image A. When that instantiation completes successfully, containers B1 and C1 should be launched, based on images B and C, respectively. In addition to conventional sequencing and dependency information, workflow definitions must be able to express that the preceding steps are performed by launching a container using Docker or some other container runtime. The workflow engine must be able to start and track the progress of containers, and the logs and output generated by the processes running inside a container — whose lifetime is likely very short — must be extracted and saved. Application workflow orchestration provides all these capabilities.

With these capabilities, a layer of abstraction for application workflows is provided such that the underlying technology platform is not a factor to be considered. If the application component, or *microservice*, that runs as container C1 must move to a different platform for some reason, this can easily be accomplished without changing the workflow logic. With this approach, application workflows can be managed in any pipeline using the best technology fit for the purpose.

In Figure 4-1, Jobs-as-Code is applied early in the development stage, just as it would be in a non-containerized environment. The application workflow orchestration solution monitors workflow execution and is included as part of the infrastructure in which the application is tested and operated.

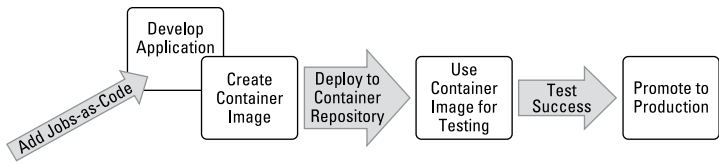


FIGURE 4-1: Adding Jobs-as-Code in a container architecture.



TIP

Application workflow orchestration must be included as a component of the infrastructure in which applications are tested and operated. In automated and virtualized or containerized environments, the infrastructure is usually built by tools that implement Infrastructure-as-Code. An important part of Jobs-as-Code is to provide services that can be invoked programmatically to build application workflow infrastructure. This means that the steps required to build and configure the components to execute and monitor application workflows can be performed automatically with no human intervention.

IN THIS CHAPTER

- » Developing a payment processing application
- » Applying DevOps and big data to fight malware
- » Pairing DevOps and cloud for a leading travel technology company

Chapter 5

Looking at Real-World Examples

This chapter provides real-world examples of organizations using application workflow orchestration with Jobs-as-Code in their DevOps processes and continuous integration/continuous delivery (CI/CD) pipelines.

Developing a Payment Processing Application

A hugely successful payments processor with active users all over the world has more than 4,500 developers working to maintain and enhance software used by millions of customers every day.

From support tickets to CI/CD

In just a few years, the payments processor has gone from requiring developers to open support tickets for everything to a process where developers are in control. They previously had to open tickets for tasks such as test environment creation, application testing, release planning, capacity planning, and other common tasks. Completion of these processes took anywhere from days to months.

The company began implementing DevOps practices using a custom tool that enabled developers to gain more control over the software development life cycle (SDLC). The company also integrated Control-M, an application workflow orchestration product from BMC, into the DevOps process.

Application delivery is now faster, and developers can manage tasks themselves. As the self-service paradigm matured, the company started seeing the benefits of DevOps and CI/CD.

Adding Jobs-as-Code

This company is all about helping its customers pay for online purchases. There are no jobs in that process, and usually it's a web or mobile application you interact with. But at its core, the company is a payment processor that requires moving money around. That movement happens in the background and is largely managed by Control-M jobs.

Taking a Jobs-as-Code approach has enabled the company's developers to build orchestration for business-critical money management applications at the same time they write new or updated business logic. By accessing the capabilities of Control-M via its Automation Application Programming Interface (API) from the Developers' Self-Service Portal, developers can work with workflow orchestration, further enhancing the self-service nature of the company's DevOps processes. For example, developers can create a complex workflow, perform a dry run to ensure that it is working as planned, and then push the entire application to production.

Being able to leverage Control-M hasn't increased the learning curve for developers, as can sometimes happen when new tools or processes are added to a DevOps process. Because Control-M uses de-facto industry standards of JavaScript Object Notation (JSON) for its configuration and Representational State Transfer (REST) APIs, it was simple to embed its functionality into the technology stack used by developers. Doing so enables the entire application, including operational workflows, to receive the additional testing benefits that come from the Jobs-as-Code paradigm.



REMEMBER

DevOps practices have shown that moving additional development tasks nearer to the developers really works. When developers learn the benefits associated with an enhanced self-service workflow, productivity increases.

Leveraging Big Data and DevOps to Fight Malware

A popular cybersecurity firm that provides protection for numerous operating systems from malware and other threats processes a serious amount of data.

Big data processing

The data processed by this cybersecurity firm meets the criteria traditionally associated with big data:

- » **Volume:** The sheer amount of data collected about potential cybersecurity threats
- » **Variety:** Numerous sources, each reporting in different ways and with different levels of information density
- » **Velocity:** The high speed with which data is generated and consumed

Data received is typically low quality, requiring a large amount of processing to gain value from it. The data is received quickly. It requires a system with little downtime and with continual movement into more structured forms for downstream analysis.

Adding application workflow orchestration

Orchestration is necessary because of the complexity of the processing involved. Incoming data is placed into a large data lake, and application workflow orchestration begins through a series of managed workflows. The first step involves extract, transform, load (ETL) processing to clean and aggregate the data. Control-M works with various back-end elements such as Redis Enterprise and several Amazon Web Services (AWS) cloud services.



TECHNICAL
STUFF

At a very high level, *ETL* refers to processes that move data from one system to another system while potentially altering the data to fit into the new system.

Once the data has gone through the ETL processes, the company sends the data into multiple streams for further processing. The data moves through several platforms such as those for machine

learning and predictions and eventually into data marts for consumption by end-user tools such as Tableau.

Control-M orchestrates the complex tasks involved in data processing and monitors the orchestration status. Monitoring is important because it enables the company to meet its service-level agreements (SLAs).

Advanced integrations

Control-M integrates well with back-end data stores and cloud services. It makes it easy to integrate with popular services and products via a powerful REST API.

Developers are not required to learn new programming languages to get immediate value from Control-M. The code to create jobs is written in standard JSON.

A Leading Travel Technology Company Leverages DevOps and the Cloud

A leading travel technology company processes the vast majority of travel-related transactions for hundreds of airlines and hundreds of thousands of hotels worldwide. They are also responsible for operations at airports, cruise lines, railroads, and ferries. If you've booked a flight online, chances are good that this company was behind it.

The company processes an incredible amount of data, with nearly 4 million bookings per day and more than 50,000 transactions per second during peak times. This is done with more than 220 million lines of code written and maintained by thousands of developers.

The company utilizes Control-M to provide critical services in support of the vital and time-sensitive technology solutions necessary for travel-related operations.

Batch jobs at scale

In addition, the company manages more than 300,000 jobs every day with Control-M. Many of the jobs are mission-critical to smooth and timely travel operations around the globe. A delay in

processing a batch job can mean a real impact on flight schedules and other travel plans.

A significant part of the work is done through file transfers. These transfers are received from multiple airlines with numerous flights and other data elements that must be processed quickly and accurately.

An example workflow involves receiving updated pricing information from airlines. This information must be updated within a related database while meeting the requirements of the SLA. The impact of not meeting the SLA is huge because tickets may be sold with the wrong pricing information.

These SLAs are sometimes difficult to meet with batch jobs. Control-M manages the jobs and provides robust monitoring for the workflow. If a problem occurs, a team member can address it immediately.

DevOps and cloud

In the past, the company used an in-house tool that enabled developers to create jobs. Those jobs were then handed off to another team for scheduling. The tool evolved but was eventually orphaned in favor of Control-M. This migration enabled developers to create the job as well as manage its end-to-end deployment.

The company has moved its thousands of development and operations staff to a DevOps model with Control-M as an important piece of the DevOps processes. The company leverages its internal cloud running Docker, Kubernetes, OpenShift, and Control-M to speed development.

With the internal cloud deployment, the company includes Control-M as part of the Docker container image, thus saving deployment and management time. Control-M can manage both the container-based jobs and those jobs still using external cloud and virtual machine-based operating environments.

With DevOps processes, an internal cloud, and Control-M, the company eliminates manual processing for requests such as job scheduling. The result is that developers can use the Jobs-as-Code approach within the CI/CD pipeline.

The company provides 99.99 percent availability on average, with more than 5,000 IT-related changes and more than 500 software deployments per month.

- » Looking at the benefits of application workflow orchestration
- » Considering the advantages of Jobs-as-Code

Chapter 6

Ten Benefits of Application Workflow Orchestration and Jobs-as-Code

This chapter describes some of the benefits of integrating application workflow orchestration and Jobs-as-Code into DevOps and the software development life cycle (SDLC).

Faster Software Delivery at Scale

When an organization applies the same methods and processes for managing application code and configurations to application workflows, business services can be developed and deployed more quickly. Over time, developers become more familiar with the process of creating and managing jobs and workflows.

In-house, proprietary tools can be replaced with an application workflow orchestration product. Importantly, processes that previously would have required manual actions from operations

or IT staff can be automated within the workflow. For example, developers can perform complex end-to-end tests of an entire application, including the execution of application workflows, without needing to open a ticket or contact the operations team.

Increased Software Quality

Batch jobs can be created and managed by the same developers who are creating the application code. This increases accuracy in relation to the business requirements because the developers are closest to those requirements. Overall software quality increases when the organization implements application workflow orchestration and Jobs-as-Code.

Automated Testing

By automating production scheduling, similar notation and interfaces for all jobs can be used at the earliest stages of the SDLC, allowing for early and accurate testing. This is even more effective when Infrastructure-as-Code is used to provision a test environment that is as close to the production environment as possible. That makes it possible to anticipate and eliminate resource contention and inconsistencies with other workloads.

Repeatable Processes

Automated testing and other elements involved in application workflow orchestration are repeatable. If something goes wrong with a test or other part of the managed processes, changes can be rolled back, the code fixed, and the test executed again.

Jobs frequently require a significant amount of setup and configuration prior to testing, so being able to repeat the test easily is important.

Better Communication

Moving Jobs-as-Code earlier in the development life cycle gives developers and operations teams a chance to discuss what must happen for the jobs to run correctly. As DevOps practices mature within an organization, better communication about how jobs are executed enables both teams to learn more about the behind-the-scenes processes and requirements for the jobs.

Increased Agility

With job creation occurring earlier in the development process, requirements can be refined and changed earlier in the process as well. Without Jobs-as-Code, the prerequisites for a job or even its requirements might not be discovered until late in the process, when operations begins creating the job.

Having developers create the job and communicate with operations staff gives the organization a chance to learn the requirements and seek optimizations in the process. For example, development staff and operations may not have the same view of how data is processed through an extract, transform, load (ETL) job. By working on the job earlier in the process, the organization can optimize previously hidden elements.

Greater Visibility

Application workflow orchestration and DevOps increase visibility. The DevOps processes are typically tied closely with agile processes, and this approach increases visibility into the entire SDLC. An application workflow orchestration product makes it easy to see where in the process a given change resides and, in the event of failure, where the process failed.

CI/CD Integration

Jobs-as-Code in application workflow orchestration integrates with popular continuous integration/continuous delivery (CI/CD) tools. At a process level, Jobs-as-Code fits well within existing practices already taking place in a DevOps organization.

Job definitions can be retrieved from the source code repository and deployed as part of the CI testing process and eventually as part of the production environment.

Containerization

Application workflow orchestration supports both non-containerized solutions and containerized solutions. The agent for the workflow orchestration product is deployed onto the container image at build time and is automatically deployed when the container is deployed.

Native Languages

Application workflow orchestration uses JavaScript Object Notation (JSON), which is lightweight and doesn't require document type definitions (DTDs). It has a consistent syntax that looks a lot like key/value pairs familiar to operations staff.

Simplicity. Visibility. Agility.

Innovate with application
workflow orchestration

Get started >



Apply a Jobs-as-Code approach to accelerate DevOps

DevOps is all about improving collaboration so businesses can speed application time-to-market without sacrificing quality. However, application workflow orchestration is often left out of the DevOps equation. This book shows you how to apply a Jobs-as-Code approach to workflow orchestration to save time, conserve resources, reduce errors, and ensure consistency to accelerate DevOps in your organization.

Inside...


- Get started with DevOps
- Explore continuous integration (CI) and continuous delivery (CD)
- See Jobs-as-Code in action
- Develop container pipelines
- Add application workflow orchestration
- Look at real-world examples



Go to **Dummies.com**[®]
for videos, step-by-step photos,
how-to articles, or to shop!

ISBN: 978-1-119-56968-8
Not For Resale

for
dummies[®]
A Wiley Brand

 Also available
as an e-book



WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley's ebook EULA.