Solution White Paper

How to Accelerate DevOps Delivery Cycles

Reduce application delivery time-to-market by optimizing automation capabilities

Table of Contents

Executive Summary

DevOps, agile development, and continuous delivery processes have attracted large followings because organizations need to accelerate service delivery to meet business goals. **Many organizations that have implemented rapid development programs still have an untapped opportunity to get new services into production faster and run them with higher quality.**

Since approximately 70 percent of all business processing is performed by application job automation (job scheduling), implementing "jobs as code" saves time during development, testing, and deployment, and results in a deliverable that is easier to operate. The alternatives increase development effort, extend test cycles, complicate deployment, and result in an application that is challenging to operate.

Enterprises can increase the pace of innovation by using automation to shorten the path that it takes to turn ideas into new services that people can use. Control-M Automation API is a set of capabilities for business services that:

- Build automation
- Increase time savings
- Enhance quality controls

This white paper explains how Control-M Automation can move services from concept to production as fast as possible, and how its features deliver value.

Seventy percent of all business processing is performed by application job automation (job scheduling). Implementing "jobs as code" saves time during development, testing, and deployment, and results in a deliverable that is easier to operate.

CONSIDER THE WHOLE LIFECYCLE

In many discussions of business transformation, there is great emphasis placed on accelerating development of new services by applying Agile, Scrum, Lean, and DevOps principles. Since DevOps comprises ″Development″ and ″Operations,″ one would expect the two parties to have relatively equal influence in organizations that are embracing this methodology. However, this is typically not the case. Choices in architecture and instrumentation are frequently made more from a development perspective since many of the most influential participants come from that side of the house. The bulk of an application's life is spent in the operations domain. **Considering the entire application lifecycle holistically can provide significant value by helping organizations achieve greater velocity in delivering and leveraging technology innovation.**

To find opportunities to streamline the application lifecycle, let's deconstruct what an application is: there is code that implements business logic (e.g., Java®, Python®); there is the infrastructure that the application will run on (e.g., Linux® server, web application server, network); there are elements that are in the middle, such as a database and the SQL statements that create tables and rebuild indexes; and there are many dependent jobs and tasks that all have to execute flawlessly for the application to work as intended.

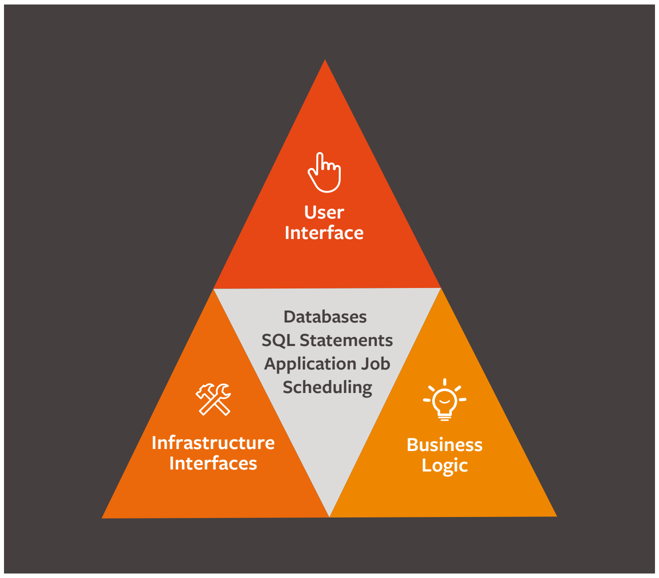


FIGURE 1: What's really in an app?

Part of the "in the middle" stuff is application job scheduling, which is a significant portion of what makes up a business application. The scheduling software itself is part of the infrastructure, such as a database or web server. However, the job definitions that define when and how jobs run are part of the application itself. These definitions are critical because they determine what to do if a job fails or what happens if a job can't start when scheduled because dependent jobs haven't been completed. The multiple jobs run every time the application runs. **In fact, one can argue jobs are more than a fundamental component—they are the application. If they don't run, the application won't run at all.**

DOING DEVOPS

If we agree application job scheduling is a fundamental component of the application, how does that impact how we "do" DevOps? It should mean we treat "scheduling" definitions like we treat Java, Python, and all other elements of the application. There are five fundamentals for accelerating development by considering scheduling during the development process:

- All application components should be architected together to eliminate or minimize waste
- All application components should be developed together
- All application components should be stored together
- All application components should be built, packaged, and tested whenever any changes are made
- All application components should travel together from phase to phase and environment to environment

By being aware of these fundamentals and their dependencies and making these activities more consistent, organizations can cut time from the development, testing, QA, and promotion processes, thus making it easier for new services to run reliably throughout their lifecycle. DevOps is a team approach, and accounting for the entire lifecycle helps the team to work together.

ARCHITECTED TOGETHER

Since an application is made up of a collection of technologies, tools, and processes, it makes sense to avoid replicating components of functions unnecessarily. For example, when including a scheduler that can initiate work at specific times on specific dates, those instrumentation capabilities shouldn't be embedded into the business logic. If the scheduler can manage flow relationships, success/failure analysis, output capture, and other functions, then those functions shouldn't be replicated in scripts, since scripts require additional effort to build, test, and support.

DEVELOPED TOGETHER

All components should be developed simultaneously in the same phase of the process as much as possible. It makes sense to use the same or similar notation and interfaces for all the components that make up the application at this stage. This consistency during development allows the components to be tested together. Even at this early stage, testing is already required.

Just like source code is syntax-checked as it's created, job definitions also should be validated at the earliest stage of construction. This is in stark contrast to traditional environments where it is common for application job scheduling definition problems to first be discovered in production. In traditional environments, new services that run fine during testing often run slowly or crash after they go live because of unforeseen resource contention or inconsistencies with other workloads in the production environment. Such inconsistencies take time to rectify and undermine the fundamental objectives of DevOps.

Forrester Research highlighted the value of early testing in its report on application and DevOps best practices:

Standardizing configurations and automating provisioning not only makes testing earlier possible, but it eliminates the "it works fine in the test environment, it will work fine in release" problem.[1]

STORED TOGETHER

A source code management (SCM) system provides a centralized repository for storing application components and for managing versions. It should be the authoritative location for enabling fallback to previous versions, "diff-ing" two versions to identify what's changed, managing potential drift in deployed systems, and providing the input to enable a quick, reliable method for building or rebuilding the application in new environments. There are many systems available (e.g., GIT, Subversion, CVS, TFS, etc.) and each has its supporters and detractors. The key is that an SCM methodology is applied to the entire application. Using a single scheduling tool will make that process more efficient and more likely to be properly adopted.

BUILT, PACKAGED, AND TESTED TOGETHER

Building an application starts with assembling all the pieces and ensuring you have all the necessary bits. This task is significantly simplified if all the objects/components are stored in a single system. Source code management systems have become the norm for managing source code and are increasingly used for software-defined infrastructure (infrastructure-as-code), too. The same approach needs to be taken for all the application components.

When any component is revised, the build tool (e.g., Jenkins®) reacts to the update by building the application and testing it to ensure the change has not caused errors or regressed any functionality, and has successfully delivered the new functionality that may have driven the change.

Having infrastructure as code is critical to provisioning a test environment that is configured as close to production as possible so tests are meaningful and truly effective.

1 Forrester Research, "DevOps Best Practices – The Path to Better Application Delivery Results," September 2015.

One characteristic of a modern delivery pipeline is that the build is immediately followed by automated tests. The tests should be as comprehensive as possible to maintain a high level of quality. If any tests fail, the entire team immediately works to fix the problem(s) so that the entire "production line" can keep chugging along. **This is where having infrastructure as code is critical to provisioning a test environment that is configured as closely to production as possible.** Consistency between the test and production environments is essential to ensuring that tests are meaningful and truly effective. The relationship between the earliest test and the eventual production environments has to be taken into account this early in the development process. The ideal test environment is identical to production throughout the entire delivery pipeline.

In the past, this ideal was completely unattainable due to the cost of infrastructure, the complexity of effort required to even approach it, and the time that would be required. With today's cloud technology and sophisticated automation leveraging everything "as code," the ideal has become not only attainable, but common for organizations.

TRAVEL TOGETHER

A typical sequence of steps during application development is for individual developers to perform unit tests immediately on their own laptops and development environments, then some lightweight tests (sometimes called "smoke tests"), followed by increasingly rigorous testing. The more complex the application is, the more comprehensive the testing will be. It is common to follow a develop-test-production sequence, but sometimes the process involves development, testing, integration, staging, pre-production testing, user acceptance testing (UAT), and more.

No matter how complex the sequence, one characteristic this pipeline should have is consistency, so that the entire application is subjected to the same hardening process. With consistency, when the application finally arrives in production, it has been as well prepared as possible.

THE PAYOFF: MORE VALUE (AND FASTER) FROM ONGOING OPERATIONS

The operational (also called production) stage is the major point of focus for the entire continuous delivery process. Until an application is operationalized, it is not delivering any value to the business. Furthermore, production is usually the longest-lived phase. However, as long as all the other stages of the software development lifecycle (SDLC) may take, there is constant pressure to compress that period to be as short as possible, whereas for production, the opposite is true. The longer an application usefully runs in production without issues or unplanned support work, the greater the value and return-on-investment to the organization.

During the long operational phase, it is critical to have insight and visibility into an application's operation and ability to meet enterprise standards for security and compliance. Because failures are still a fact of life, it is essential to build visibility into applications so that operations and support teams can quickly identify, analyze, and resolve problems to make the application available again.

Faster delivery cycles enable organizations to provide innovative solutions by quickly delivering new capabilities and reducing the time they spend waiting for feedback. They are able to try new ideas quickly, improve the ones that work, and rapidly improve or remove the ones that don't. Better, faster feedback enables organizations to cut waste, reduce cost, and improve customer experiences.[2]

"TRADITIONAL" DEVOPS

It may sound strange to describe DevOps as "traditional" since it is a relatively new approach for building and delivering applications. However, because this emerging practice is heavily influenced by the development phases of the SDLC (everything "as code" bears witness to this bias), applications are commonly operationalized using basic tools available to developers. It's common to spend significant time writing extensive scripting. The scripts are then coupled with a basic tool like cron or Jenkins that has arisen from the development environment. Both are less than ideal for managing production. **This behavior is commonly the result of existing operational tools lacking support for a DevOps approach to application delivery.** Such deficiencies manifest themselves in a variety of ways, including the requirement for using graphical interfaces for administration, operation, installation, and configuration, rather than programmatic ones.

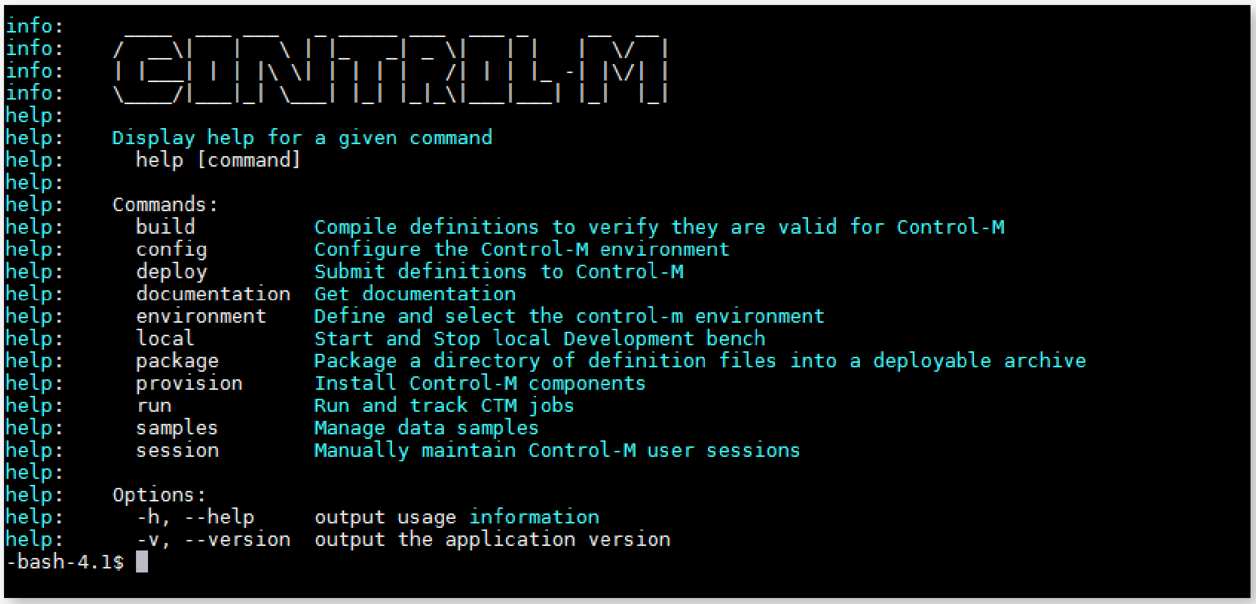
2 Forrester Research, "DevOps Best Practices – The Path to Better Application Delivery Results," September 2015.

Before DevOps, treating job flows separately towards the end of the SDLC was an acceptable compromise mandated by organizational process and tool ownership. DevOps, however, requires an automated approach that should be consistent across the entire SDLC, as discussed above. The reason is simple: including automation from the inception of the development lifecycle saves time in downstream stages, so new services can be made faster and with higher quality.

INTRODUCING CONTROL-M AUTOMATION API

DevOps architects and engineers have been given the clear mandate to accelerate application delivery to support business agility. Along with that responsibility, these teams have also been granted the right to select the tools they use to meet the goals that have been set out for them. **Control-M Automation API enables DevOps teams to access and consume the capabilities provided by Control-M, while retaining all of the benefits of speed and agility expected from DevOps methodology.**

Control-M automates application scheduling to ensure that critical business services like logistics or supply chain management operate correctly and on-time. Control-M manages data pipelines that drive modern data warehouses, analytics, and business intelligence. Control-M provides SLA management, file transfer, auditing, reporting, and version control to ensure compliance with legal, regulatory, and industry standards. It also delivers deep visibility into process flows, enabling early detection and quick remediation of application failures that may impact business service availability.



```
info:      _____  ____  ____  _  _____ ____   ____  ____  _     __  __
info:     / ___ \|    ||    \| ||_   _|  _ \ |    ||    ||    |  \/  |
info:    | |____||    || |\ \  |  | | | |_/ /|    ||    ||_ -|  |\/| |
info:     \_____/|____||_| \___|  |_| |_|_\ |____||____|____|  |_|  |_|
help:
help:     Display help for a given command
help:       help [command]
help:
help:     Commands:
help:       build          Compile definitions to verify they are valid for Control-M
help:       config         Configure the Control-M environment
help:       deploy         Submit definitions to Control-M
help:       documentation  Get documentation
help:       environment    Define and select the control-m environment
help:       local          Start and Stop local Development bench
help:       package        Package a directory of definition files into a deployable archive
help:       provision      Install Control-M components
help:       run            Run and track CTM jobs
help:       samples        Manage data samples
help:       session        Manually maintain Control-M user sessions
help:
help:     Options:
help:       -h, --help     output usage information
help:       -v, --version  output the application version
-bash-4.1$
```

FIGURE 2: Control-M Automation API Command Line Interface

Control-M Automation API is a set of programmatic interfaces that enables developers and DevOps engineers to use Control-M in a self-service manner within the modern application release process. Building job definitions in JSON and using GIT and RESTful APIs enables seamless integration of workflow scheduling artifacts with the CI/CD tools that are used to automate the application release and deployment process. **By making the entire delivery pipeline nearly identical to the target operational environment, applications run more reliably and errors are diagnosed more quickly.** SLA management, audit, and compliance are all baked-in during delivery and do not have to be bolted on at the last minute.

Control-M Automation API inverts the conventional structure of how application job automation is defined and managed. Using JSON makes it familiar and straightforward for developers and DevOps engineers to build the artifacts, which are then stored in a source code management (SCM) solution like GIT. Updates to the SCM are then used to trigger application builds via tools like Jenkins. The process can then continue with the creation of environments used for progressively more sophisticated testing. Environments can be created automatically, using Automation API services to provision and configure Control-M components. Configuration data is provided as JSON artifacts that are stored in the SCM together with the rest of the application. This approach to managing job flows implements "job workflow as code" similar to the way configuration tools like Chef® or Puppet® implement "infrastructure as code." The approach helps organizations leverage their skills and allocate more time to development and less to support.

The professionals responsible for scheduling, running, and managing workloads benefit by working with a powerful and intuitive interface, so there is little to no learning curve and no requirement for scripting or complex integration to support new workflows.

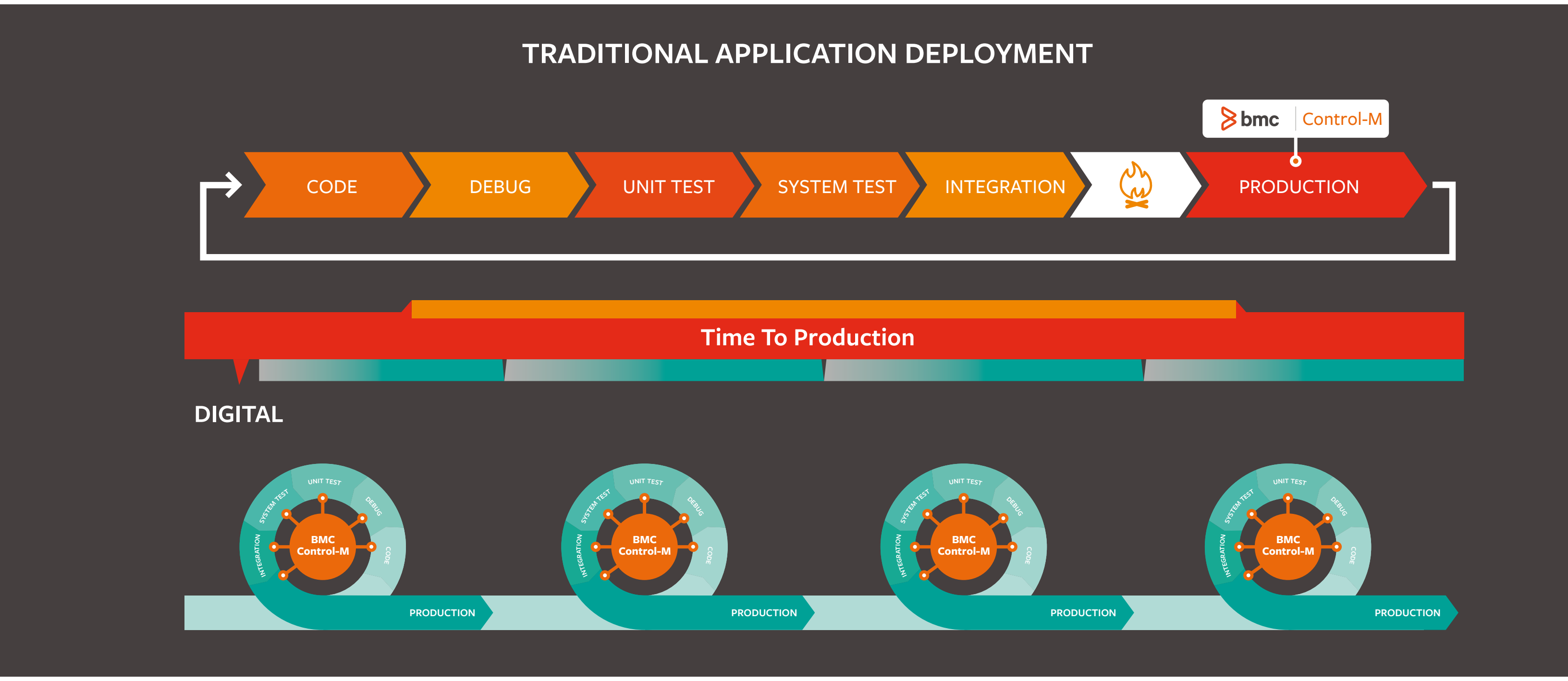


FIGURE 3: Traditional Versus Continuous Application Deployment

With Control-M Automation API, you can automate tasks throughout the lifecycle, including code, build, run, test, package, deploy, provision, and configure stages.

Some of the functions and capabilities that Control-M Automation API supports include:

- **Syntax validation**
- **Configuration management**
- **Agent provisioning**
- **Workbench**
- **Code deployment**

Developers can use these capabilities to add complex integrations into jobs early in the development process with simple workflows that do not require scripting and immediately provide success/failure analysis, relationship management, and visibility. For example, run process A and if successful, run process B. If A fails, run C and/or send an email notification and open an incident. Functionality expected in production can now be embedded as ″jobs-as-code″ into development and test environments, so tests can accurately simulate real-world conditions, eliminating surprises or rework after new services are promoted into production. Automated SLA monitoring and management and audit support can also be built into applications, so the tasks do not have to be done manually or automated through an additional software solution.

"The tool has empowered developers to own their own Control-M work without engaging another team. They can make changes that used to take weeks in a matter of minutes. It's easy to record, and there's no more hand holding by the Ops team. This is appealing to leadership because it fits in with CI/CD [continuous integration/continuous deployment] goals and the ability to represent everything as code. We were blown away."

Fortune 500® Control-M Application API beta customer

CONCLUSION

The sooner the realities of operationalizing applications are addressed in the development process, the faster enterprises can turn their ideas into functional, valuable business services. **Treating jobs as code is a powerful and often overlooked step that organizations should take to accelerate all stages of the software delivery lifecycle.** When jobs are architected, developed, stored, built, packaged, tested, and promoted together, the resulting application is optimized for the long operations stage. The "togetherness" approach not only builds-in consistency, it enables automation to be extended to more processes throughout the lifecycle. DevOps teams gain the consistency and visibility they need to complete development faster, promote jobs more easily, and run them more reliably and efficiently.

Control-M Automation API makes it possible to develop jobs as code and provides the consistency and automation that organizations need to optimize the delivery cycle. It helps unlock the full value of DevOps by providing a common environment that aligns development and production professionals. By giving developers the ability to create and manage automated workflows using their favorite tools and giving operations professionals the ability to extend their scheduling and management environments to new types of services, Control-M Automation API addresses the entire lifecycle, so organizations can introduce new services faster and operate them more efficiently to maximize business benefits.

FOR MORE INFORMATION

To learn more about how Control-M Automation API can accelerate your DevOps delivery cycles, visit **bmc.com/it-solutions/control-m-devops**

483061