mezmo

MEZMO EBOOK

Log Management in the Age of Digital Transformation

"Digital Transformation" has been bouncing around as a buzzword for years. But it was perhaps not until 2020 that many teams realized the full meaning and urgency of the term.

INTRODUCTION

"Digital Transformation" has been bouncing around as a buzzword for years. But it was perhaps not until 2020 that many teams realized the full meaning and urgency of the term. Faced with unprecedented demand to deliver high-performing, ultra-reliable user experiences in a chaotic environment, organizations over the past year have learned that they can no longer ignore digital transformation.

In practice, embracing digital transformation requires a number of changes. Siloed teams must evolve into DevOps squads. Monoliths need refactoring into microservices. Release velocity must change from once a year to once or more a day.

If you follow conversations about DevOps and digital transformation, you already know these things. You know that you should be updating your practices and tools to ensure that your teams have what they need, when they need it.

But how do you actually do that? How can operations teams quickly identify and alert developers of production issues before they become Twitter hashtags? How can developers use data to assess how their code changes may impact the larger pipeline before they ever commit changes? How can business leaders leverage massive amounts of data to better understand the health of their organization as a whole?

For many companies, these are the real questions that need to be answered in the Age of Digital Transformation.

TABLE OF CONTENTS

LOG MANAGEMENT AND DIGITAL TRANSFORMATION

The solution to these challenges lies, in part, in effective log management. Although it would, admittedly, be an overstatement to suggest that log management alone is the key to successful digital transformation, leveraging logs effectively is one key ingredient to ensuring that teams can move quickly, fail forward and keep their end-users continually delighted in an age of rapid change.

To prove the point, this eBook walks through the role that log management plays in digital transformation initiatives. By focusing on the steps organizations typically go through along their digital transformation journeys – which often start with a move toward microservices, followed by adoption of other DevOps practices – the following chapters explain which special challenges developers and operations teams face as they modernize their IT environments, how log management addresses these challenges and where log management fits within the DevOps journey.

THE POWER OF MICROSERVICES LOGGING

Let's start by discussing what log management means when you begin to modernize your IT estate by moving from monoliths to microservices-based applications.

The microservices-oriented architectural style is the culmination of longstanding efforts to promote a heterogeneous yet orchestrated way of modeling complex software systems. As business use cases become more advanced, this style has grown in adoption. Logging microservices is a powerful way for developers to extract valuable information from the massive amount of data that modern applications process and produce.

Modern Systems = Complex Systems

In his *Introduction to the Modeling and Analysis of Complex Systems*, Professor Hirorki Sayama defines complex systems as "networks made of a number of components that interact with each other, typically in a nonlinear fashion." If we acknowledge that a microservices architecture can be viewed as a network of semi-independent systems that communicate with each other, we can say that it is a complex system rather than a monolith. But what is the reason for this?

The first reason is the environment: microservices architectures respond to the requirements of interconnected systems, which are much more common than isolated systems these days. Meanwhile, monolithic architectures respond to simpler business requirements. Monoliths are useful for ideation, proof of concepts, and focused work like user stories and one-off projects. While complex applications can be built with a monolithic architecture, as the business

case expands it's often better to be prepared with a distributed architecture.

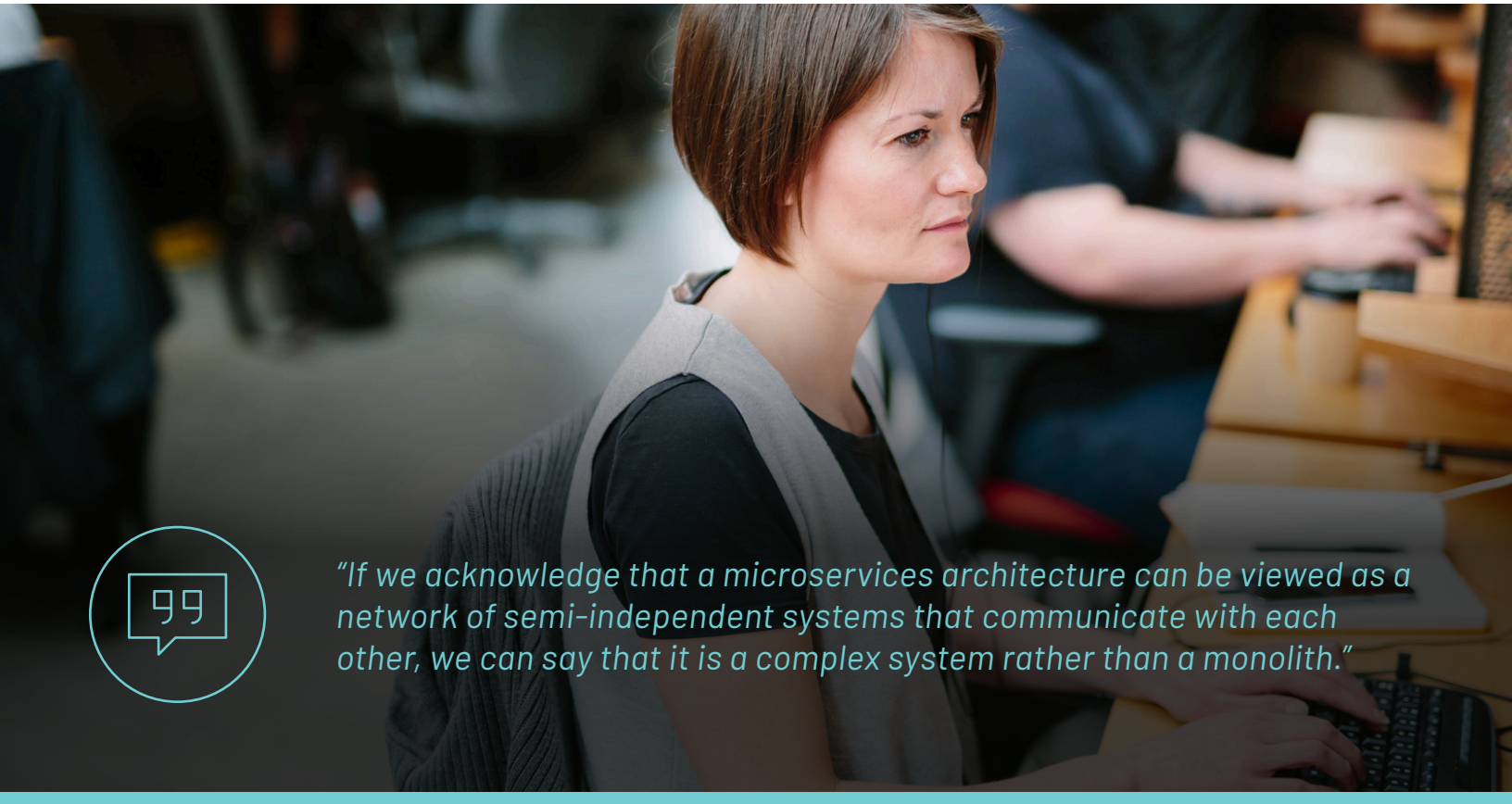
Other benefits of using a distributed architecture are that critical components can be isolated and fault tolerance pathways can be defined, teams can be laser focused on delivering business value, and the entire system can more fluidly scale to meet the needs of a dynamic customer base.

Designing a distributed system is not as simple as creating a federation of monoliths. Microservices architectures create new challenges in areas that were not as difficult to design for a single system (such as consistency, communication, persistence, security, and deployment). In addition, microservices instances are usually considered disposable in order to allow scale-in and scale-out depending on the system load. Under the traditional logging scheme, this leads to numerous, ephemeral log files that must be accessed one at a time. Centralizing your logs into one single source of truth is essential to understand how these connected systems interact and diagnose errors with the context you need.

The Power of Logs

In his book, I Heart Logs, Jay Kreps presents the concept of logs as a data structure that solves problems like consensus in order to determine what happened if there was an issue in a distributed system. This means that logging can be used as a tool for data integration by making all of a system's data available even if it comes from heterogeneous sources. Logging is also a useful way to collect data that is produced in real time, such as data that is gathered by sensors that are connected to the internet and deployed in hundreds or thousands of instances at the same time.

By leaning on this view of logs as power tools for processing and storing data, developers move towards what Kreps calls "log-centric architectures." In this type of architecture, small and specialized microservices can run in large numbers simultaneously



"If we acknowledge that a microservices architecture can be viewed as a network of semi-independent systems that communicate with each other, we can say that it is a complex system rather than a monolith."

in order to focus on their internal logic and delegate tasks such as consistency, data flow, and recovery. With so many systems interacting at once, centralized log management is the only way to easily find and make sense of the information that you need.

Microservices also provide data about their own operational status, meaning that a centralized logging platform can help developers find and address other operational problems using the power of logging. According to the principles of chaos engineering, organizations need to ask themselves how much confidence they can have in the complex systems they put into production. This will depend entirely on their techniques for gathering data and their ability to understand and take action on it.

The heterogeneous nature of microservices contributes to the chaos that can result from this type of architecture. Microservices are usually not implemented by the same team, so there can be several different ways to log data. Even if that's not the case, business requirements are different for each microservice, so logs may not represent data in a consistent way. All of this means that the platform used to centralize the logging for microservices must be flexible enough to transform and aggregate the data in order to provide useful information, no matter if the format is standard or custom.

Centralizing log data is not a simple task, given that microservices architectures must comply with higher levels of scalability, traceability, flexibility, and security. A carefully-chosen platform that will connect the logs from your microservices will allow you to harness the power of the information that you can derive from this kind of architecture.

IMPROVING YOUR RELEASE CYCLE WITH LOG MANAGEMENT

Making the shift to microservices is only one of the key steps toward Digital Transformation. Equally important is embracing CI/CD to enable a high rate of release velocity – which is one of the pillars of DevOps.

The DevOps Release Cycle

In the past, the traditional software development life cycle would begin with the development team working on a solution in its entirety. Once development was complete, they would pass it on to the quality assurance (QA) team for testing. Assuming the product got approval from QA, it would then be passed to an operations team to deploy and manage the solution in the production environment. While this process worked, it wasn't efficient and would often result in adversarial relationships between the teams.

Most modern development teams have adopted the DevOps approach for developing and releasing software into a production environment. The objective of DevOps is to help development, operations, and security teams work more harmoniously and, in some cases, even centralize their work into a single team. When practiced effectively, this has the effect of shortening the release cycle and improving quality. Often, a single team is responsible for managing the entire lifecycle of a product, increasing a sense of ownership and reducing friction. DevOps can be incredibly effective, but it requires a paradigm shift in the way teams work; and teams need access to new and more efficient tools.

Here's an example of a DevOps release cycle:

1. A small unit of work is selected, and the team designs and builds a solution.
2. The solution includes additions to the test suite, such as unit tests, integration tests and performance tests.
3. The code is reviewed and merged into the main code base.
4. The code merge triggers a continuous integration (CI) pipeline, which executes all tests within the test suite, and may also run static analysis and security scans against the updated code base.
5. If successful, a continuous delivery or deployment (CD) pipeline builds and packages the code, and deploys it into either a test or a production environment.
6. The team monitors the new deployment to determine its effectiveness and stability, and takes any additional action to ensure that it works as expected.

Steps 4, 5 and 6 are often combined into a single utility, known as a CI/CD pipeline, and the effectiveness of this pipeline has a massive bearing on the team's effectiveness and success.

The Power of Automation

We use automation with software development to streamline processes and ensure that repeatable steps are always performed, without relying on human input or intervention. An effective CI/CD pipeline leverages the power of automation to take changes to the code base, and move them through a carefully designed series of tests and validations until they are either rejected or deployed into an environment.

The automated processes within the pipeline rely on feedback to validate their results. Feedback during the testing process comes from the results of each of the tests. Once the product is deployed, the pipeline needs access to information that validates the performance of the application or service itself. An effective log management solution is critical at this point.

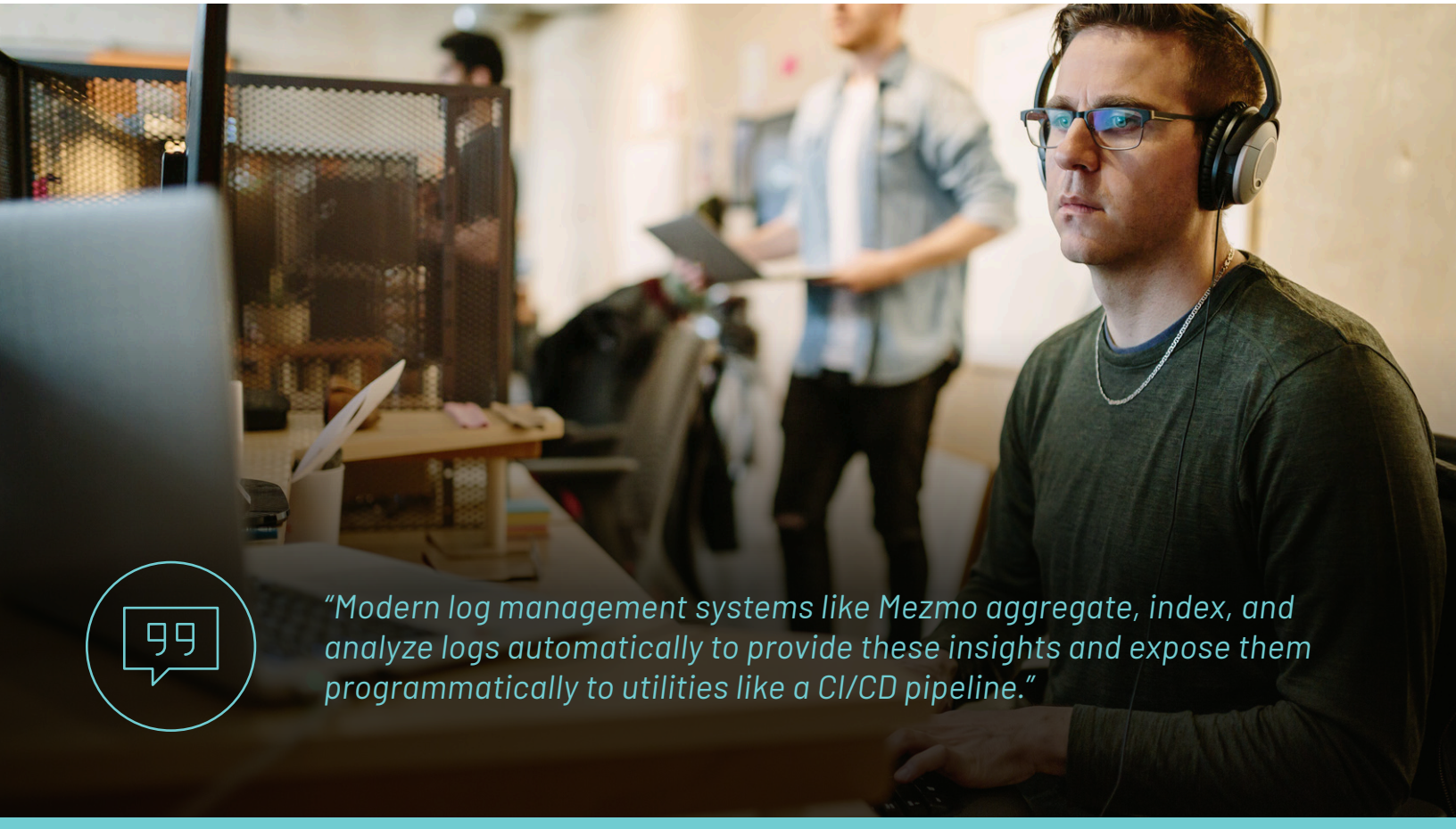
Using Logs to Drive Automation

Traditionally, software engineers have used logs to troubleshoot applications and gain insights into the performance of the services they support. Modern log management systems like Mezmo aggregate, index, and analyze logs automatically to provide these insights and expose them programmatically to utilities like a CI/CD pipeline.

The CI/CD pipeline can use data from the log management system in a couple of ways. One way is as a validation source for a canary deployment within a microservice architecture. The team might deploy a specific service in a high-availability configuration in a microservices architecture consisting of several containers or instances running behind a load balancer.

Let's consider an example of a cluster of five instances running a particular service. Instead of replacing all the instances at once, the pipeline deploys a single instance. Once successful, it configures the load balancer to route 5% of the traffic to the new instance. The log management system collects the logs from the new instance. The pipeline can query the system to get the new instance's logs and determine error rates, success rates, and metrics such as latency and processing times.

The pipeline compares these results to those from the instances running the previous version of the software. Suppose the results fall within an acceptable range or show an improvement. In that case, the release is determined to be successful, and the remaining instances are updated or replaced with new instances executing the new code. If the results show an increase in errors or performance degradation, the pipeline fails the deployment and removes the canary instance.



"Modern log management systems like Mezmo aggregate, index, and analyze logs automatically to provide these insights and expose them programmatically to utilities like a CI/CD pipeline."

WHY DEVOPS TOOLS ARE ESSENTIAL FOR DIGITAL TRANSFORMATION

Although the journey toward DevOps often begins with microservices and CI/CD, it doesn't stop there. As this chapter explains, a variety of other DevOps tools and practices must come into play to enable complete digital transformation.

Why Digital Transformations Succeed and Fail

Successful digital transformations take dedication, hard work, and discipline. The organization needs to commit to the process and believe in future success. The most effective transformations start with a clear vision and commitment from the executive leadership team, who select transformation leaders and empower them to enact change, introduce new practices, and adapt as needed. The strategy can involve selecting high-performers in each area to investigate and implement new processes and then disseminate them and change their teams' culture. Over time, these processes evolve alongside a culture of experimentation, agility, and adoption of continual improvement.

Digital transformations generally fail due to a lack of commitment and a lack of flexibility in implementing the changes. An organization might make the changes in some parts of their organization and not in others. Or, they might attempt to change processes without addressing the corporate culture, and failing to change the collective mindset of those who support the day-to-day operations.

A digital transformation involves a lot of risk and a lot of coordinated changes. It requires flexibility and agility. One path that many organizations have used to succeed in their endeavors is using the Agile software movement and DevOps principles.

Using DevOps to Drive Digital Transformation

DevOps is a new way of thinking about developing and supporting software. DevOps breaks down the traditional silos of development, quality assurance and operations, and forms new teams. These teams then own applications and services – from the design process to deployment, monitoring, and support. DevOps teams often operate within an agile framework, whereby small units of work are identified and developed within short timeframes. This approach allows teams to deploy new changes rapidly and easily pivot to new business requirements as needed.

Because of the unique combination of skills required to practice DevOps , these engineering teams must have a reliable set of tools and utilities to support their efforts. This support infrastructure utilizes automation and various data sources to test and validate software through the development process. It continues as the applications are deployed and used by consumers, within the organization and externally.

Ensuring DevOps Success with the Right Toolset

At the core of a successful DevOps practice are tools that gather and analyze data programmatically. The DevOps process produces data in many forms. Data sources can include, but aren't limited to:

1. The results of automated testing; unit tests, integration tests, performance tests, and others.
2. Static code analysis, security and vulnerability scans, and code quality assessments.
3. Application and system logging.
4. Application Performance Metrics (APM).

The testing and code analysis results are essential during the build, package, and deployment phases of an application or service. After deployment, the application and system logs and the performance metrics are critical for a concept known as observability. Observability is the external analysis of metrics from a system that provides insights into the health and the performance of that system. Log aggregation and management is an invaluable component that makes up one pillar of observability, alongside tracing and metrics. Especially in distributed systems, managing logs from a central control plane is critical.

Teams should be able to adopt and use these tools with ease. Adopting a self-service model, whereby teams have access to on-demand training and documentation, can easily integrate the tools into the DevOps process. The guiding principle is to provide tools and reduce any friction related to DevOps teams' use.

Learning from DevOps Tools

The DevOps tools' features that make them effective can be applied to your organization as well. Just as the tools use data as a basis for their decisions, your teams use data to inform their decisions. Data-based decision-making is a crucial part of any digital transformation.

DevOps tools also reduce friction by automating repeatable processes to make them consistent and more efficient. The end goal of a digital transformation is to automate repeatable tasks and improve their efficiency and speed. Automating processes frees your most valuable asset – your people – and allows them to focus on exploring creative and unique ways of solving problems and continuing to revolutionize your business practices.

UNDERSTANDING THE IMPACT CODE CHANGES HAVE ON YOUR PIPELINE

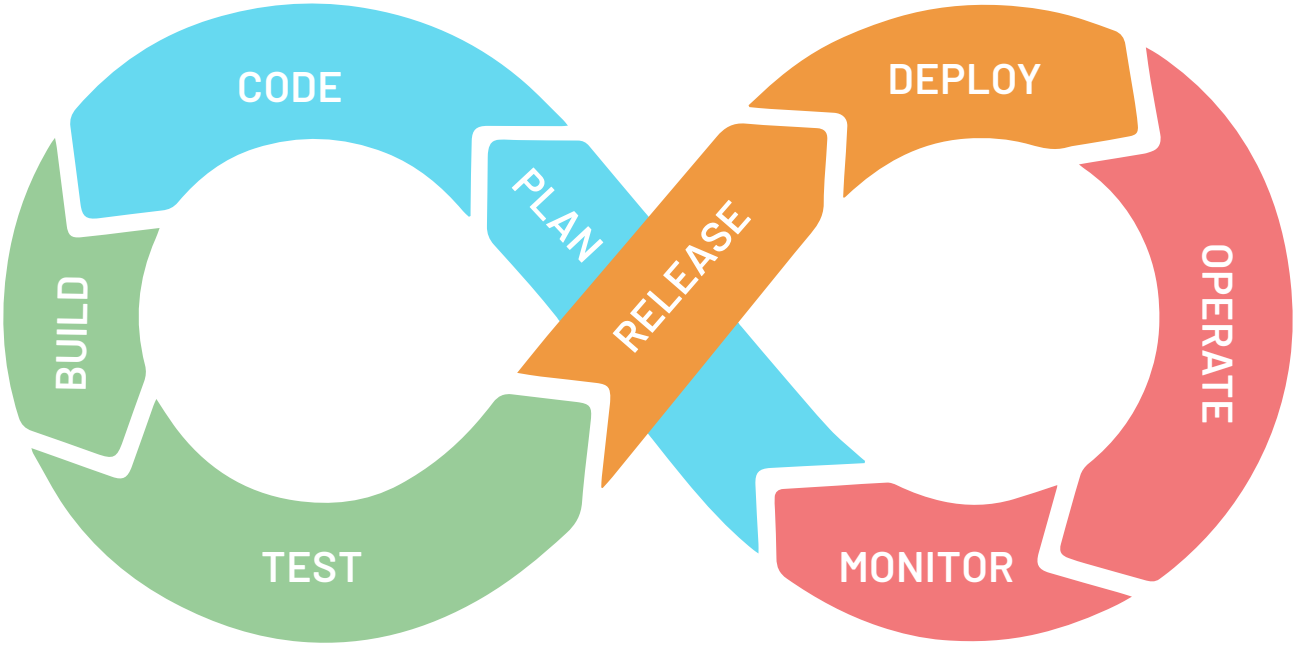
In previous chapters, we explored the importance of logging to monitor distributed applications or systems built using a microservices architecture. We've also discussed how a comprehensive log management solution is invaluable in continually improving your release cycle. And, we looked at the importance of DevOps toolsets in accomplishing a digital transformation within your organization.

This chapter will take a closer look at how a log management system can provide insights into the impact that code changes have at each stage in the development lifecycle, including within your continuous integration / continuous deployment (CI/CD) pipeline. We'll discuss the different effects that code changes have and how to observe, identify, and mitigate those effects.

The DevOps Lifecycle

The DevOps lifecycle is a cycle of identifying a need, designing and implementing a solution, and then deploying it through an automated pipeline into a production environment. Once deployed, various systems monitor the code. Using this information, the team can identify additional features or modifications and repeat the process. Within the lifecycle, a core philosophy at play is that of feedback loops. At each step, the team uses procedures to validate the code and determine if it can proceed. If validation fails, the team rectifies the problem and resubmits the modified code. Rapid and frequent feedback gives the engineers time to react, prioritize, and address issues before they can significantly impact the process.

Let's walk through each step within the lifecycle and identify how code changes impact the process and how teams can use tools and automation to ensure problems



are identified and mitigated. When this lifecycle is automated and proven by the team, it establishes a sense of trust with the team. The certainty that the process works increases a team's confidence in their code, accelerating the development process.

Planning and Coding

The planning and coding phase of the lifecycle is the most difficult to automate. This phase is essential to establish a firm foundation for the remainder of the process. It includes the process of identifying and refining the requirements and building the solution. Building the solution involves coding and adding tests to a test suite that validates the new or modified functionality. Engineers typically execute these tests locally. It is typical for teams to have a process run the test suite and other analytics tools when introducing changes to the code repository.

Techniques like API-first development and test-driven development help ensure that the logic within the new or modified code is sound and meets the requirements established during the planning phase. In addition to executing the test suite,utilities like static code analysis, mutation testing, and vulnerability scanning complement any checks that are part of a compiler or preprocessor.

Finally, in this part of the process, the engineers instrument their code to support observability and monitoring later in the lifecycle. This instrumentation takes the form of adding log statements and implementing libraries and frameworks that support distributed tracing and performance metric reporting.

Build and Test

The build and test actions are the initial steps that the pipeline performs on newly submitted code changes. Ideally, the actions performed should mirror those already performed manually by the engineer. Performing them validates that the compilation, test suites, and packaging of the application or service are completed as expected.

The pipeline relies on the build and test processes' results to validate this part of the lifecycle. If failures occur, the pipeline utility typically sends an alert to the engineering team, and they research and rectify the problem before submitting the change again.

Release and Deploy

Having validated that the code passed all tests, and packaging the code into a new deployment unit, the pipeline proceeds to the release and deploy phases. This pipeline stage is when the instrumentation and logging are of the utmost importance. The pipeline deploys the new code to validate the new release's performance without a significant impact on users of the system. Some examples of different approaches that teams use are canary deployment, blue/green deployment, or red/black deployments.

The pipeline relies on data from the newly deployed code to validate that the deployment was successful and that consumer interactions execute as expected. System and application logs are the core atomic unit for gathering the necessary information to make these decisions. A log management system such as Mezmo is essential to perform this task. The log management system aggregates, analyzes, and then provides data to the pipeline to make appropriate decisions based on the results.

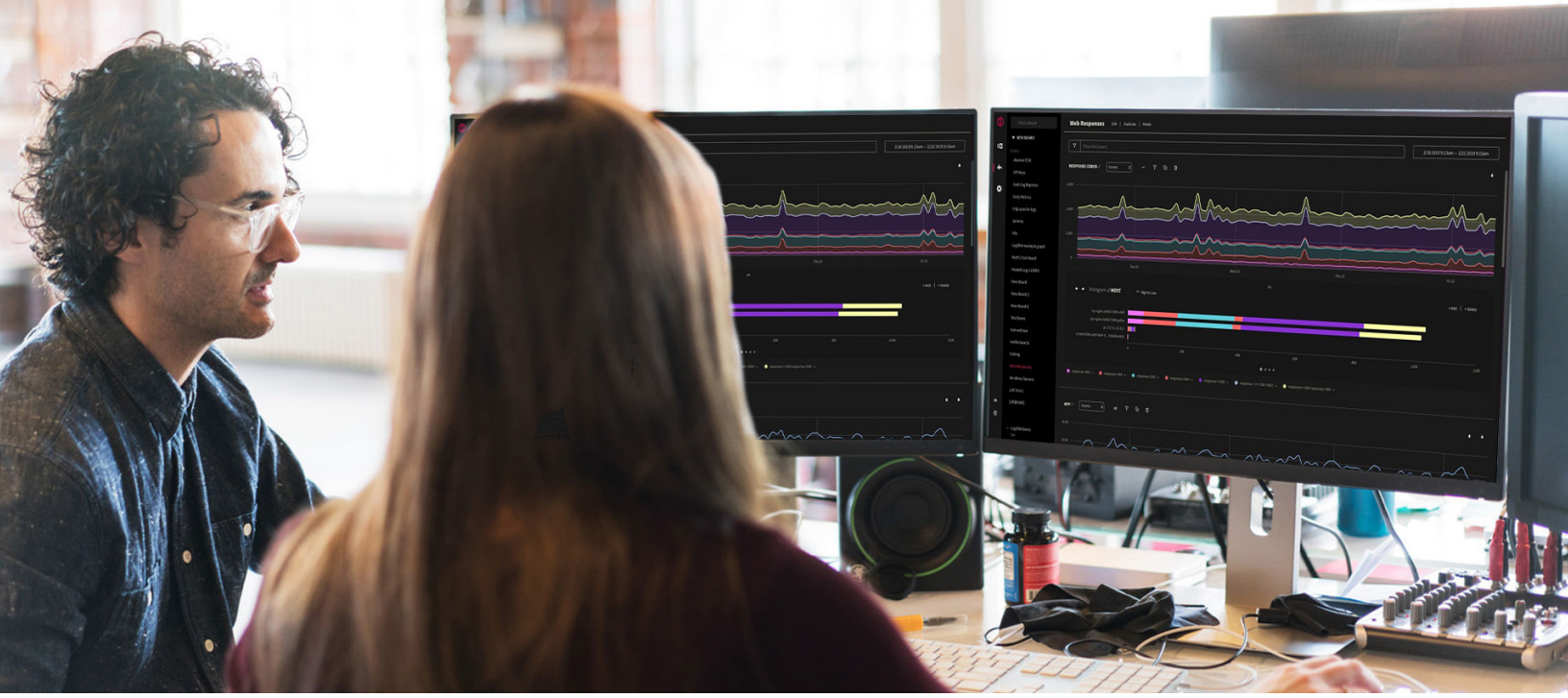
If the pipeline determines that the deployment is successful, it orchestrates the complete replacement of previous versions of the application or service and annotates the pipeline as having completed successfully.

Monitor and Operate

Once the pipeline deploys the application, we enter the final phase of the DevOps lifecycle. As with the previous stage, system and application logs and performance metrics are central to this phase. In the past, operations personnel would routinely review logs and manually check each server's performance metrics as part of their role. In the DevOps age, and with modern distributed architectures, this is no longer feasible or necessary.

Log management systems do more than collect and aggregate logs. Systems like Mezmo perform complex analytics on the logs to help engineers detect anomalies and identify potential problems. DevOps teams can establish baseline metrics for performance, error rates, and latency. Modern log management systems can consider these baseline metrics and alert users when they act outside of the expected behavior.

Automating these processes is more efficient. In many cases, even the incident response can be automated, resulting in fewer interruptions for the engineering team.



SAML allows them to leverage existing identities and roles within your organization to control access to your data that's stored in their systems.

The basic process is quick and fairly seamless:

1. A client connects to the service provider.
2. The service provider redirects to the client organization's identity provider with a basic SAML token (such as an Active Directory).
3. The identity provider recognizes the SAML token and asks the client for their login credentials.
4. After a successful login, the identity provider redirects the client back to the service provider with a fully populated SAML token, which can include client information and authorized roles.
5. The service provider receives the SAML token and processes its data; then, it grants the client access according to the roles that it has been assigned.
6. The client uses the service provider.

By leveraging SAML and taking advantage of its single sign-on and federation features, your organization's identity provider can have full control over both the accounts and the roles that are used by any solution providers. This use allows existing processes to handle account management, and any centralized logging that you have in place will gain immediate visibility into logins and access requests on those external systems. Many third-party services also allow events and other activities to be exported as logs, either in real-time or as a scheduled activity. These logs can then be incorporated securely into your centralized logging to improve visibility into all ongoing activities across your organization.

CONCLUSION

In short, although log management may seem to be a rather mundane process, it plays a vital role in any digital transformation initiative. By allowing organizations to retain visibility into their applications as they embrace complex, microservices-based architectures, log management serves as the linchpin of a successful shift to digital.

Businesses can't thrive today if they lack the visibility and actionability that log management brings to their software-delivery and management processes. They need a tool like Mezmo, which offers real-time aggregation, monitoring, and analysis, as well as real-time alerting to tools like PagerDuty and Slack, to complement digital transformation initiatives.

To learn more about how Mezmo's developer-friendly log management platform can empower your team with the insights it needs to complete a successful digital transformation, request a free, full-featured product trial.

About Mezmo

Mezmo is a centralized log management solution that helps modern engineering teams be more productive in a DevOps-oriented world. It enables frictionless consumption and actionability of log data so developers can monitor, debug, and troubleshoot their systems with ease.

mezmo

Thank You

PR & marketing inquiries:	marketing@mezmo.com
Sales inquiries:	sales@mezmo.com
Technical inquiries:	tech@mezmo.com
Customer service:	outreach@mezmo.com