MEZMO EBOOK

LOGGING IN THE AGE OF DEVOPS

FROM MONOLITH TO MICROSERVICES AND BEYOND

Logging in the age of DevOps is more critical than ever. It's key to maintaining visibility and security in today's fast-moving, highly dynamic environments.

mezmo

TABLE OF CONTENTS

INTRODUCTION

Logging in today's fast-moving, highly dynamic environments.

If you work in development, IT engineering, or a similar role, you're certainly familiar with logging. For decades, logs and log data have been part and parcel of virtually every type of workflow and process in the IT industry.

The past several years, however, have witnessed a series of significant disruptions in the way workloads are deployed. These changes have brought with them important consequences for logs and logging.

One major innovation has been the widespread adoption of a DevOps culture, an approach to software delivery and management that emphasizes seamless collaboration between IT teams and developers. Using DevOps-oriented practices effectively requires leveraging log data in new ways in order to provide developers and IT engineers alike with visibility into the workloads that they collaboratively deliver and manage. This is all the more true when other stakeholders, such as security engineers, are integrated into the DevOps process.

At the same time, the way workloads are structured has changed. Many applications are now deployed as microservices with the result that there are more logs, more log data, and more varied logging structures to contend with. Microservices have also necessitated a rethinking of the way organizations handle logs.

The advent of cloud-native computing has also impacted logs and logging in profound ways. The cloud-native trend has increased the number and types of logs that organizations now need to manage. It has introduced additional challenges, such as a lack of persistent storage in some cases for the locations where log data is natively generated.

To make matters more complicated, legacy applications and software delivery practices persist among all of this change. That means organizations must not only adapt to the log management requirements of new types of workloads and architectures but must do so while continuing to manage logs effectively for legacy applications.

In short, logging in the age of DevOps has become harder than ever. And, it is more critical than ever because it is key to maintaining visibility and security in today's fast-moving, highly dynamic environments.

With these needs and challenges in mind, Mezmo has prepared this eBook to offer guidance on how best to approach the log management challenges that teams face today. It covers logging best practices for DevOps-oriented teams, microservices architectures, and cloud-native environments, while also explaining how to manage logs effectively for the legacy applications that many teams still deploy.

LOGGING FOR MONOLITHS VS. LOGGING FOR MICROSERVICES

Let's start with one of the first logging conundrums that teams must understand when they begin migrating to modern application architectures: How logging for monolithic applications is different from logging for microservices.

At first glance, microservices logging may seem simple. You just take the same logging principles you've always followed for monoliths and apply them to each microservice in your application, right?

Well, no. The differences between microservices and monolithic architectures amount to much more than a difference in the number of services involved. To create and manage logs in a way that provides true visibility into a microservices environment, developers and IT engineers must adhere to a different set of practices than they would use when logging for a monolith.

Microservices vs. Monolithic Logging

No matter how exactly you define a microservices architecture (a topic about which there is some debate), there are several distinctions between monolithic and microservices architectures that have critical ramifications for logging.

The first and most obvious is that, in a microservices app, there are more individual

components of the application. By extension, there are more logs because each microservice typically generates its own set of log data.

Another, perhaps less obvious, difference is that in a microservices architecture the various microservices interact constantly in ways that make them dependent on each other. From the perspective of logging, it's not enough to have logs that only provide visibility into the state of each individual microservice. You also need to use logs to understand how microservices are interacting and to trace the web of dependencies that link microservices together.

A third challenge with microservices logging is that the scope and types of logs generated by each microservice tend not to be the same. Each microservice performs a different function, and some do more work or handle more requests than others. As a result, it usually doesn't make sense for developers to try to have each microservice structure its logs in a uniform way. Instead, log format and structure are tailored to each microservice, leading to less consistency from the perspective of log management.

The distributed nature of microservices creates an additional challenge. Whereas a monolithic application is typically hosted in just one place—a single server or a single cloud,—microservices applications can be distributed across a variety of locations. The ability to distribute applications in this way is part of the reason why microservices are so advantageous. However, this ability also creates complexity with logging because different microservices store their logs in different places.

In many cases, microservices are deployed using technologies like containers or serverless functions, which lack built-in persistent storage. If a microservice logs data inside of a container or a serverless function execution environment, the logs will disappear when the container or serverless function shuts down unless they are moved somewhere else beforehand.

For all of these reasons, it would be simplistic and ineffective to approach microservices logging in the same way as logging for a monolith.

"To create and manage logs in a way that provides true visibility into a microservices environment, developers and IT engineers must adhere to a different set of practices than they would use when logging for a monolith"

Best Practices for Microservices Logging

Fortunately, the complexity and challenges posed by microservices logging can be tamed.
The following best practices help ensure that all of the log data generated by microservices can be leveraged effectively to provide true visibility into these applications.

Aggregate and analyze all microservices log data

Aggregate all of the log data from an application's various microservices and analyze it in one place to gain holistic visibility into the application. Knowing information such as the startup time or requests handled by an individual microservice is not very useful for maintaining overall application performance and availability if you don't know how that one service's metrics correlate with those of the rest of the application.

Make sure log aggregation is truly centralized

It's not enough to aggregate some log data into one place (such as a public cloud vendor's log manager, like CloudWatch) and aggregate other data somewhere else (like a third-party log management tool). Although this approach may seem like the way to go if some microservices run in one location and others run somewhere else, you need all of your log data in a single location if you want to analyze and store it effectively.

Use custom identifiers

Ensure that data logged by microservices includes unique identifiers, such as the name of the microservice or a unique ID for each type of message it generates. This information is immensely valuable for helping to contextualize and trace interactions between microservices. If you don't have unique identifiers, it is much harder to determine which information within your aggregated logs comes from which microservice.

In certain cases, you may need to implement unique logging identifiers within the microservices' source code. However, you can also use logging agent features, such as tagging, to associate identifiers to log data without having to modify source code.

Use custom parsing

Because microservices logs are often structured in multiple ways, trying to search through all of your logging data using generic regexes is typically not very effective. Instead, consider writing custom parsing rules that govern how your log analytics tool identifies relevant trends within log data, even if you are working with logs of varying types or structures.

Log to persistent storage

As noted above, microservices often run inside infrastructure—like a container—that lacks persistent storage. A basic and essential best practice in that case is to ensure that log data is written to somewhere where it will be stored persistently and remain available if the container shuts down.

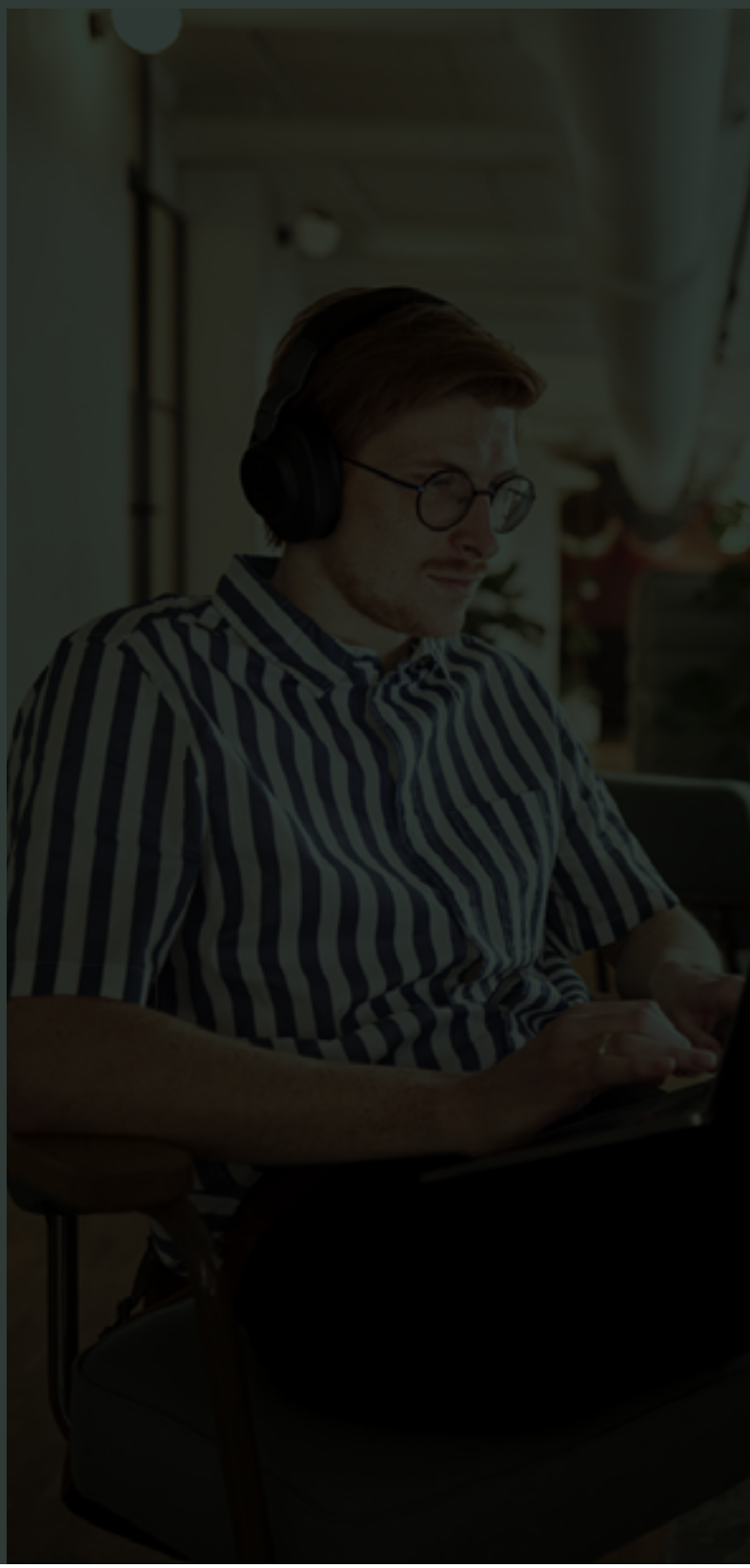
You could do this by modifying source code or your container configurations to ensure that the logs are written to an external storage volume. An easier approach, however, is to run a logging agent that will collect data from the containerized microservice in real time and aggregate it within a reliable storage location.

This way, you kill two birds with one stone (or logging agent, as it were): You aggregate logs and move log data to persistent storage, all in one step.

Chapter Summary

Logging for microservices is an entirely different game than logging for monoliths. Not only is there more log data and more logs in a microservices environment, but the lack of uniformity in log type and structure, the distributed nature of microservices hosting environments, and the complex interdependence of microservices mean that logs for microservices applications must be managed in a more sophisticated and streamlined way.

WHAT THE CLOUD-NATIVE REVOLUTION MEANS FOR LOG MANAGEMENT



Once upon a time, log management was relatively straightforward. The volume, types, and structures of logs were simple and manageable.

However, over the past few years, all of this simplicity has gone out the window. Thanks to the shift toward cloud-native technologies—such as loosely coupled services, microservices architectures, and technologies like containers and Kubernetes—the log management strategies of the past no longer suffice. Managing logs successfully in a cloud-native world requires fundamental changes to the way logs are aggregated, analyzed, and more.

Here's how the cloud-native revolution has changed the nature of log management and what IT and DevOps teams can do to continue managing logs effectively.

What Makes Cloud-Native Logging Different

At first glance, log management in a cloud-native world may not seem that different from conventional logging. Cloud-native infrastructure and applications still generate logs, and the fundamental steps of the log management process—collection, aggregation, analysis, rotation—still apply.

Yet, if you actually start trying to monitor a cloud-native environment, it quickly becomes clear that managing logs efficiently and effectively is much more difficult. There are four main reasons why.

More logs

First and foremost, there are simply more logs to contend with.

Before the cloud-native era, most applications were monoliths that ran on individual servers. Each application typically generated only one log (if it even created its own log at all. Sometimes, applications logged data to syslog instead). Each server also typically generated only a handful of logs, with syslog and auth being the main ones. Thus, to manage logs for the entire environment, you only had a few logs to contend with.

In cloud-native environments, in contrast, you typically work with microservices architectures where there could be a dozen or more different services running, each providing a different piece of the functionality required to compose the entire application. Every microservice may generate its own log.

Not only that, but there are more layers of infrastructure, too, and, by extension, more logs. You have not only the underlying host servers and the logs they generate, but also logs created by the abstraction layer—such as Docker or Kubernetes or both, depending on how you use them—that sits between the application and the underlying infrastructure.

In short, the shift to cloud-native means that IT teams have gone from contending with a handful of separate logs for each application they support to a dozen or more.

More types of logs

Not only are there more logs overall, but there are more types of logs. Instead of just having server logs and application logs, you have logs for your cloud infrastructure, logs for Kubernetes or Docker, authentication logs, logs for both Windows and Linux (because it's more common now to be using both types of operating systems in the same shop), to name a few.

This variety adds complexity not only because there are more distinct types of log data to manage, but also because these various types of logs are often formatted in different ways. As a result, it is harder to parse all logs at once using regex matching or other types of generic queries.

Diverse logging architectures

Along with the increase in the number and types of logs has come more complexity and variation with regard to the way log data is actually exposed within application environments.

Kubernetes is a prime example. Kubernetes provides some built-in functionality for collecting logs at the node level; the exact way that it does that collection depends on environment variables. For example, it logs to journald on systems with systemd installed but otherwise writes directly to log files inside /var/log.

To make matters more complicated, Kubernetes has no native support for cluster-level logging although, again, multiple approaches are possible. You could use a logging agent running on each Kubernetes node to generate log data for the cluster, or you could run a logging agent in a sidecar container. Alternatively, you could try to generate cluster-wide log data directly from the application, provided your cluster architecture and application make this practical.

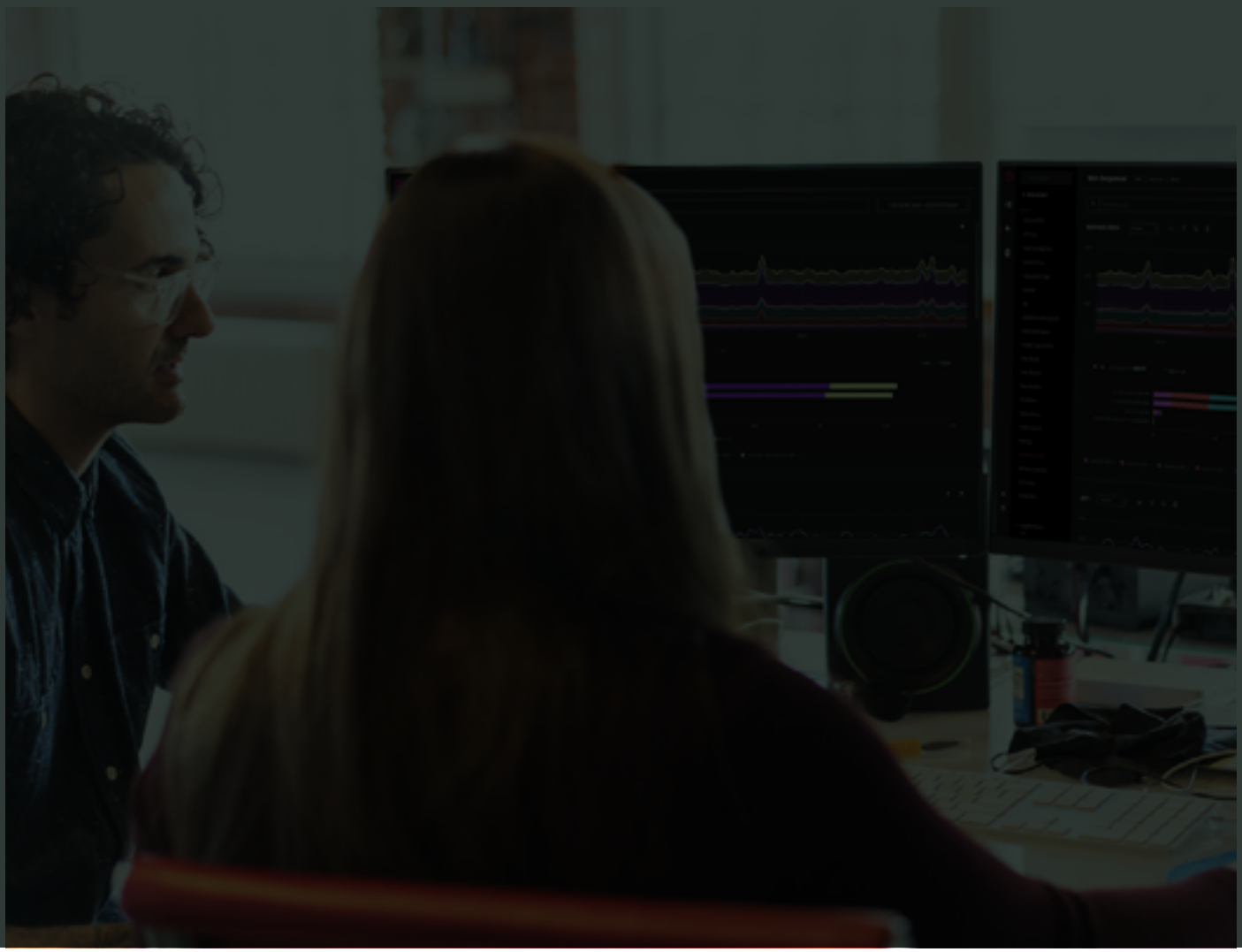
Non-persistent log storage

A final challenge in cloud-native logging arises from the fact that some cloud-native applications lack persistent data storage. Containers are the prime example.

"The variety of logs make it harder to parse all logs at once using regex matching or other types of generic queries."

When a container instance stops running, all data stored inside the container is permanently destroyed. Thus, if log data was stored inside the container (which it often is, by default), it will disappear along with the container. Because containers are ephemeral, with instances halting and being removed with new ones spinning up automatically, it's not as if admins are asked whether they want to save log data before a container shuts down. It will just shut down and be removed, taking your log data with it unless you have moved that data somewhere else beforehand.

This transience may be okay if you only care about working with log data in real time. However, if you need to keep historical logs available for a certain period of time, losing log data when containers stop running is not acceptable.



Best Practices for Cloud-Native Log Management

To respond to these challenges of logging in a cloud-native world, teams can use these guidelines.

Unify log collection and aggregation

With so many different types of log formats and architectures to support and remember, trying to manage the logs for each system separately is not feasible.

Instead, implement a unified, centralized log management solution that automatically collects data from all parts of your environment and aggregates it into a single location.

Adopt a flexible log management solution

Your log management tools and processes should be able to support any type of environment without you having to reconfigure the environment.

If you have, for example, one Kubernetes cluster that exposes log data in one way and a second cluster that logs in a different way, you should be able to collect and analyze logs from both clusters without having to change the way either cluster deals with logs. Likewise, if you have one application running on one public cloud and another one on a different cloud, you shouldn't have to modify the default logging behavior of either cloud environment in order to manage its logs from a central location.

Collect logs in real time

One way to ensure that logs from environments without persistent storage don't disappear is to collect log data in real time and aggregate it in an independent location. That way, log data is preserved in a persistent log manager as soon as it is born and will remain available even if the container shuts down.

This approach is preferable to trying to collect log data only at fixed periods from inside containers, which leaves you at risk of missing some logs if the containers shut down earlier than you expected.

Use custom log parsers

Instead of ignoring logs that are structured in ways that conventional analytics tools can't support, take advantage of custom log parsers to work with data in any format. That way, you don't risk missing out on important insights from non-standard logs.

Chapter Summary

Cloud-native log management is fundamentally different from managing log data for conventional, monolithic applications. It's not just that the scale of log data has increased (though it has), but also that there is much greater diversity when it comes to the way log data is recorded, structured, and exposed. Managing logs effectively in the face of these challenges requires a log management solution that fully centralizes and unifies log data from any and all systems that you support, while also providing the power to derive insights from non-standard log types.

HOW LOGGING AND DEVSECOPS GO HAND-IN-HAND

Today, many organizations are thinking not just in DevOps terms, but also in DevSecOps terms, a phrase that indicates a mental shift to extend DevOps principles and practices into the realm of security. And while logging is probably not the first item to come to mind when most of us think about DevSecOps, it should be.

Logging and log management play a critical role in helping to put DevSecOps principles into practice by ensuring that developers, IT operations staff, and security teams have the visibility and communication pipelines they need to prioritize security at all stages of the DevOps delivery cycle.

What is DevSecOps?

DevSecOps is a cultural shift that means making security part and parcel of all DevOps practices, from code development and integration at the start of the application delivery pipeline through to deployment and management at the end.

The term has become popular as a way to emphasize that a DevOps mindset should not just involve better integration between development and IT processes (which is the classic definition of DevOps) but should extend to security, too. Otherwise, the argument goes, security exists in a silo and ends up becoming an afterthought when code is written, tested, and deployed.

What it Takes to "Do" DevSecOps

Like other DevOps principles, DevSecOps is a philosophy rather than a specific set of practices. There are many paths that teams can take to "do" DevSecOps, and many tools that will help them get there. Thus, it would be wrong to think of DevSecOps as something that you achieve by implementing a certain process or adopting a certain tool.

At the core of any healthy DevSecOps strategy, however, are two key principles:

1. Seamless communication between the development, operations, and security teams. It's only by being able to communicate and collaborate that each of these groups can work together to improve the overall security of DevOps processes.

2. Across-the-board visibility into security considerations. This means that security issues, or potential security issues, that exist at any stage of the application delivery cycle must be easy to identify. It also means that all stakeholders—developers, IT engineers, the security team, and anyone else—have constant visibility into the state of the delivery cycle and can, therefore, identify and respond to security concerns readily.

Logging and DevSecOps

Logging and log management play an essential role in achieving both of these principles and, by extension, in enabling a DevSecOps mindset.

Communication between Development, Operations, and Security

When it comes to communication between different teams about security, log data functions as a single source of truth that all parties can use to identify, respond to, and discuss security considerations.

Whether you're a developer, an IT engineer, or a security engineer, you know how to work with log data. In fact, logging is one of the few tools and skillsets that all three of these disciplines share.

Thus, by using logs as the basis for communication, it becomes practical to operationalize DevSecOps. If a security engineer wants to discuss a potential vulnerability with development or operations, he or she can point to log data to identify the event and convey relevant information. This technique is much more effective than attempting to have developers or operations staff work through security tools they are not accustomed to using.

It's important to note that simply collecting logs is not enough to facilitate DevSecOps. Managing the logs effectively—in a way that allows all stakeholders to see relevant trends, contextualize events, and pull out the information they need to perform a certain task—is also essential.

Security visibility

Logs also form a common foundation for providing visibility into potential security considerations related to software. No matter which stage of the application delivery cycle you are working with, you can generate and analyze log data to help identify security risks.

"You can't work in a DevSecOps world if you don't manage logs effectively. If you are wondering how to put DevSecOps principles into practice, your logging strategy is a good place to start."

Logs from a CI server could be used to identify anomalous code integrations that merit further investigation. Logs from application tests and builds provide an opportunity to evaluate how software runs, and find potential vulnerabilities, before deployment. Logs from production environments, of course, offer the greatest degree of visibility into security issues that may arise once the latest version of an application is up and running.

In this way, logs make it possible to understand the security context of software at all stages of the DevOps pipeline, which is exactly what DevSecOps is all about.

Here again, of course, merely generating logs is not enough. It's also critical to have an efficient log management strategy in place so that log data from the various stages of the pipeline can be aggregated and analyzed effectively.

Chapter Summary

It would be an overstatement to say that logging and log management are the only essential ingredients in a DevSecOps team. A lot of other factors come into play in order to achieve a successful DevSec-Ops culture, such as obtaining buy-in for security by all members of the IT organization.

However, logging and log management are a critical facet of DevSecOps, and ones that have perhaps been underappreciated. You can't work in a DevSec-Ops world if you don't manage logs effectively. If you are wondering how to put DevSecOps principles into practice, your logging strategy is a good place to start.

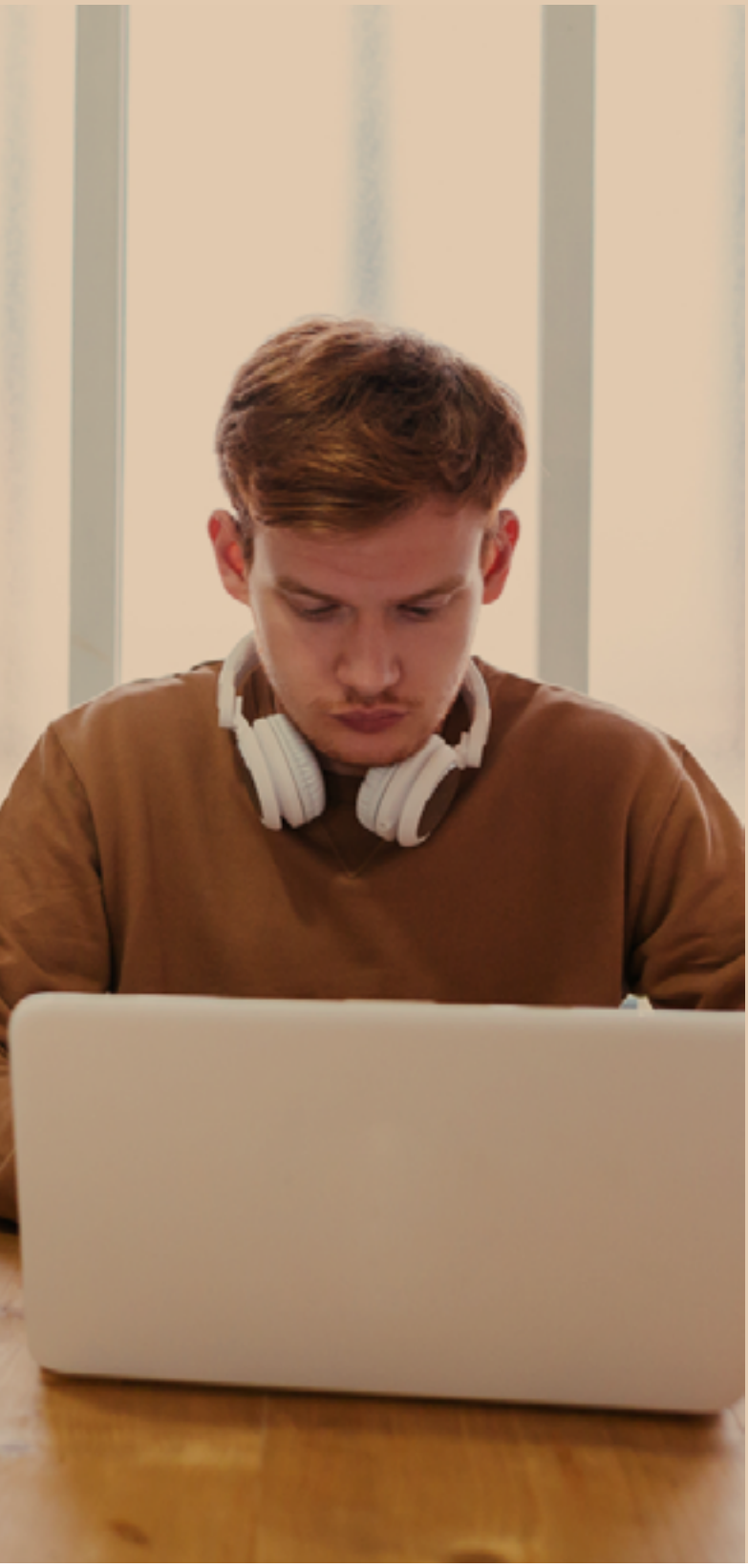
LOOKING FORWARD WITH LEGACY APPLICATION LOGGING

As the introduction noted, not all workloads have been modernized. Legacy applications remain widespread, too, and must now often be maintained alongside modern ones.

For that reason, teams should strive to develop a log management strategy that works equally well for legacy and modern apps. Ideally, their strategy will reinforce plans for evolving legacy apps over time and moving toward full modernization – even if they lack the development resources to achieve that goal overnight.
After all, any application that stands the test of time will reach a point in which it needs to be modernized, either partially or in full, if it is to remain a viable solution for its end users. The trick is to decide how to revamp the application so that the modernization effort provides great value to both the organization and end users. Log analysis and the use of a robust log analysis platform can have a large impact on a business's plan for modernizing an application.

Why is Modernizing an Application Important?

There are a host of important reasons for modernizing an application. These reasons revolve around improving application quality and reliability and revamping potentially outdated development processes used within the team. In doing so, an organization can streamline portions of the



application development process and provide themselves with the opportunity to improve and expand their business, managing logs efficiently and effectively is much more difficult. There are four main reasons why.

Decreased maintenance leads to increased innovation

Application modernization typically involves moving off of older, antiquated frameworks and technologies and onto newer ones that experience fewer failures and are easier to support. This move results in a reduction in the amount of time that development and operations teams need to spend in order to keep the system up and running. This streamlining, in turn, frees up resources that the business can then leverage to refine functionality, thereby empowering the team to provide greater value to their end users.

Improved system security and compliance

It's no secret that outdated code, frameworks, and technologies are more likely to be susceptible to both security vulnerabilities and compliance issues. For organizations collecting personal and financial information about their customers, modernizing an application represents an opportunity to build with security and compliance in mind by baking it into their application. Rebuilding that application provides developers and operations teams with the opportunity to improve adherence to industry standards for compliance and to utilize libraries, frameworks, and tools that provide an inherently increased level of awareness of security needs than those developed years ago.
This shift is representative of a more complete

6 https://aws.amazon.com/compliance/shared-responsibility-model/

solution than identifying and reacting to security and compliance shortfalls in a legacy system or attempting to bolt fixes onto an outdated codebase or infrastructure.

Improved development processes lead to an increased speed of delivery

Modernizing an application provides the opportunity for teams not only to upgrade their legacy applications but also to make upgrades to their development processes, undoubtedly leading to an increased speed of delivery. Modernization projects represent a natural entry point for the adoption of modern development practices, including continuous integration, continuous delivery, automated and continuous testing, and more. These practices help organizations deliver changes to their applications in less time and with an increased level of quality and stability.

How Can Logging in Your Legacy Application Help with Modernization?

So, keeping the advantages of application modernization in mind, how can logging in a legacy system help drive the modernization process? A few uses for legacy application log data and the resulting analysis of this data come to mind.

Identifying system components subject to frequent errors and performance problems

We all know that log analysis is of great use in determining the contributing factors of specific issues occurring within an application. However, it can also be useful when attempting to analyze the bigger picture.

One aspect lending insight to the bigger picture lies in identifying which components of a system are experiencing high error rates or great levels of latency. In the event of a gradual migration to a newer system, such insights can provide development organizations with the information they need to choose which components should be given priority in the modernization effort. Focusing on those elements that are contributing to overall application quality issues such as frequent failures or excessive slowness will help the organization increase the reliability of the product in the fastest manner possible.

Identifying areas in need of an overhaul in user experience

Just because a legacy system component isn't experiencing obvious shortfalls such as performance issues or frequent runtime errors doesn't necessarily mean it shouldn't be an area of focus in an application redesign.
For instance, consider the case of modernizing an application designed for streaming music. In this case, there could exist a feature that allows for building playlists to share with friends. If users frequently begin the process of creating a playlist but fail to follow through, it's possible this feature could be due for an overhaul to improve intuitiveness and overall user experience.

Log analysis can be utilized to provide insights into user behavior, enabling development organizations to identify features with room for improvement in these areas and thereby helping organizations make decisions about which components of a legacy system require greater attention within the modernization effort.

Chapter summary: Choosing the Right Logging Solution

There are several paths available to those teams modernizing legacy systems. There is the all-or-nothing approach of rebuilding an entire system at once, and there is the gradual modernization technique where the system is overhauled on a component-by-component basis.

With the latter, it will prove extremely valuable to choose a log analysis platform with the ability to monitor the remaining legacy components in conjunction with those that have been moved. By ensuring these logs are available in a centralized location, developers and incident response personnel will have all relevant information at their fingertips. This ability goes a long way in empowering efficient and effective incident response processes.

"By ensuring logs are available in a centralized location, developers and incident response personnel will have all relevant information at their fingertips."

CONCLUSION

The nature of workloads, as well as the entire approach to software delivery and management, have changed. So, too, have the log management strategies that yield the most effective results for today's dynamic, large-scale application environments.

Thriving in the face of these challenges—while, at the same time, managing logs effectively for the legacy applications that remain a part of many organizations' environments—requires a new approach to log management.

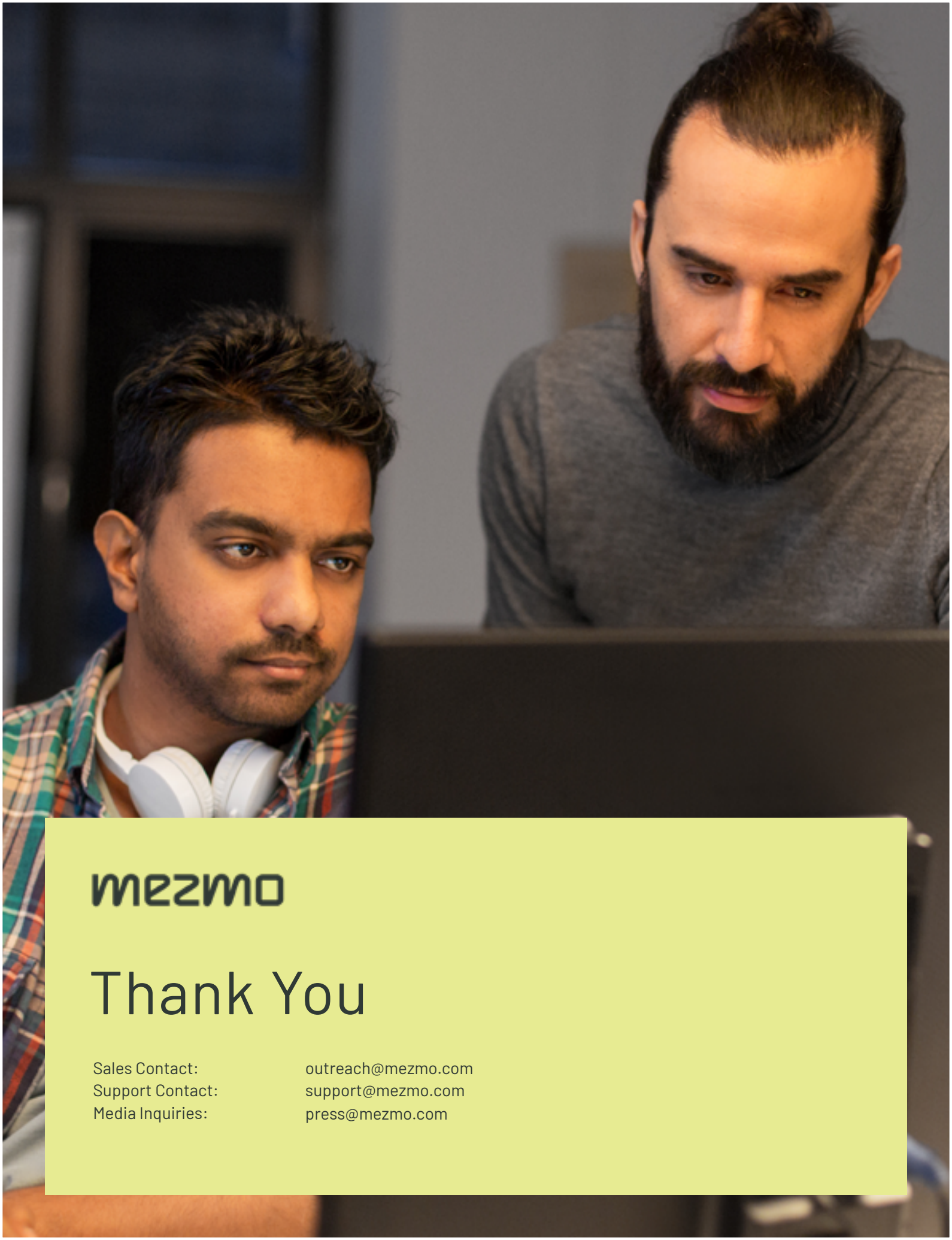
Mezmo's suite of flexible, microservices-ready log management tools empowers teams to achieve this goal. By enabling organizations of any size to manage, analyze, and derive value from logs no matter how they are structured, which programming languages generated them, or which part of an application they relate to, Mezmo provides a holistic log management solution for the age of DevOps.

See for yourself with a free, fourteen-day Mezmo trial.

ABOUT MEZMO

Mezmo is a modern log management solution that empowers DevOps teams with the insights that they need to develop and debug their applications with ease. Users can get up and running in minutes, see logs from any source instantly in Live Tail, and effortlessly search them with natural language. Custom Parsing, Views, and Alerts put users in control of their data every step of the way.

To learn more, visit mezmo.com and start your free trial today.



mezmo

Thank You

Sales Contact:	outreach@mezmo.com
Support Contact:	support@mezmo.com
Media Inquiries:	press@mezmo.com