THUNDRA™

A Comprehensive Guide on Debugging Microservices

Contents

Abstract

Debugging applications in the cloud can be challenging. Microservice applications are designed to run independently with their own databases and logging, but this poses problems for developers when attempting to trace possible bugs in the cloud. To add to this, some bugs don't show up until the developer's code is up and running in the cloud and are caused by infrastructure and environmental–rather than logic–issues. This is where developers can benefit from utilizing third-party tools to better track and target bugs that occur. In this guide, we will discuss how to effectively debug microservice applications in the cloud.

This requires implementing a system or tool that allows development teams to set up non-breaking, non-intrusive options. Some of these solutions enable developers to observe and trace issues into code as well as monitor said systems. With the right tools and systems in place, developers can better manage complex microservice architecture.

The idea behind microservices was that smaller, independent services would provide more flexibility for developers to edit code without concerning themselves that they might impact separate modules. If a developer accidentally allowed a bug in the code, it would only impact that specific microservice.

However, microservices present their own unique challenges. It is difficult to reproduce errors because requests made between various modules are asynchronous and thus difficult to track. In addition, it's not easy to detect which services interact with which, as each request might take a different path, and microservice systems gain complexity over time because each service can log differently, and be completely independent of other services. The lack of cohesiveness between different microservices poses a major problem when it comes to debugging.

This guide discusses some of the challenges in debugging microservices, as well as some methods your developers can use to reduce those challenges.

The Challenges of Debugging Microservices

TRACING REQUESTS ACROSS SERVICES IS A HASSLE

Modern-day infrastructure is quickly becoming more complex and difficult to understand from a high level, as microservices and servless models become more popular. The principles behind these design models make it difficult to track requests and data throughout the processes. This is because each module, each serverless call, and each cloud component only further obfuscates what exactly is going on beneath the hood.

In turn, this causes a problem with observability and tracing, which refers to the developer's ability to tell what the system's internal state is. When a system is developed with a microservices approach, it can be a challenge to detect the internal state based on the outputs.

This becomes apparent during debugging if a developer struggles to pinpoint the root cause of an error because all the modules are isolated.

CODE MAY ACT AND SCALE DIFFERENTLY IN PRODUCTION

How code acts in production and development is not always the same. Even if you've tested your code with unit and integration tests, it can be difficult to assess what will happen to it once millions of requests are being processed through it on distributed servers.

If the code scales poorly or the underlying database is insufficient for managing the number of requests occurring in production, this could lead to various failures.

With so many possible points of failure, it can be extremely challenging to determine which piece of code or database is causing the problem. A table could be deadlocked or a piece of code poorly written; but whatever the reason, the issue might not be apparent.

Ways To Debug Code

There are many methods developers can use to debug coding issues on microservices. Most were developed for monolithic code, where it's easy to trace and track requests through the singular code base. Here are a few examples of classic debugging techniques.

APM SOLUTIONS

Application performance management (APM) helps a development team stay on top of their application's performance. APMs can also be used to detect exactly where in your application a problem is occurring. In addition, they can often parse web server logs to calculate the number of requests coming in, monitor error rates, identify slowdowns, and track key application dependencies like your RDS or Kafka instance.

These can help pinpoint major issues from a high level so your team knows where to look when errors pop up.

DEBUGGING DUMPSTERS

Before there were log analyzers and orderly forms of error message handling, it wasn't uncommon to get a memory dump of an entire system where an error occurred. From there, developers would have to spend time reading through the messages and trying to pinpoint the source of the error. This was far from efficient and didn't always lead to the root cause of the problem, as it would show only the state of memory at that time.

BREAKPOINTS FOR STOPPING CODE

Breakpoints are a key component to active debugging. Developers can see what is changing in the state of their application at each breakpoint and track what could be going wrong. Putting breakpoints in production is not feasible. You can't stop your code as it's running live just to see what the variables are. That is why this method of debugging can't truly be used in production. It's too invasive and doesn't allow developers access to the information they need when they're trying to get to the root of their problem.

LOG ANALYZERS

There are plenty of third-party log analyzers that can help break down and process logs to reduce the amount of time developers spend parsing through them. These analyzers can help detect where bugs or other issues are occurring, provide dashboarding to track performance, and help file bugs directly into Git repositories.

This works great with monolithic systems where a request is interacting with one system, regardless of how many modules it interacts with. However, once you start working with microservices, it can become much more difficult to manage and track a request through logs.

Must Haves for Debugging Cloud Applications

While many of the methods described above can be very challenging to use in the context of microservices, that doesn't mean you can't debug your microservice code using the methods described below:

NON-BREAKING, NON-INTRUSIVE OPTIONS

This method will often require a third-party tool that provides non-breaking breakpoints. These tools allow your team to set breakpoints that do not actually pause or halt, as they would with a standard debugger.

However, you can use these breakpoints in an execution flow to view stack traces and global variables as well as individual watch variables.

Developers can use these non-intrusive breakpoints to test their hypotheses on where issues are occurring without halting code or redeploying/restarting their code base.

OBSERVABILITY INTO CODE

As discussed, one of the challenges with microservices is observability: It can be difficult to track requests as they pass through the complex systems developed over the years. Development teams would need to spend time and resources to develop their own observability platform, taking up valuable time that should be spent on new features for their company's product.

However, many modern third-party tools have been developed to help track requests and provide code observability for microservices, as well as serverless and distributed computing. For example, Thundra can help you to observe how your user requests go through your complex infrastructure in production.

AUTONOMOUS EXCEPTION CAPTURE

Whatever the issue, it's important to have a system that automatically captures these errors. However, capturing them is not enough. Your team will also need the logs and variables that show when the error occurred so that it can be easily replicated on their end. This is especially necessary for obscure bugs that occur only at odd times, like daylight savings or a leap year.

Systems that automatically track these exceptions and also help point out patterns can be very helpful for debugging in production, whether or not your system is a microservice.

Debugging Remote Doesn't Have to Be Hard

Debugging has always been an art and a discipline that requires time to master. In the modern microservice space, it has become more complex as systems grow and mature.

It is difficult to trace requests through code and to predict how code will scale once you put it into production. Add in the challenge of debugging while in production and the task becomes even more daunting–though not impossible.

Using third-party tools to help track requests and create a more observable environment provides a huge advantage for developers. Besides just creating more observable environments, many of these third-party tools, like Thundra Sidekick, can provide in-production breakpoints and help correlate logs with metrics and tracing.

Microservice architectures offer many advantages, but you shouldn't let debugging negate them. Consider employing a few of these features in your environment to improve your team's development process. It will help your team maintain their codebase and reduce the time they spend tracking down pesky bugs.

About Thundra

Thundra is a SaaS developer platform company that empowers application teams to develop, debug and deliver modern microservices on cloud. By offering everything from automated instrumentation to cloud debugging in a single platform, Thundra eliminates the need for multiple tools for pre-production environments. Developers who embrace DevOps and are motivated to solve issues in cloud applications use Thundra to eliminate problems in production. Thereby minimizing the need for on-call response, increase productivity, and deliver robust and resilient applications.

Want to see Thundra in action?

demo.thundra.io

Any questions or inquiries?

Contact us at support@thundra.io

THUNDRA