ROOKOUT

Remote Debugging For Developer Team Managers

The recent shift to remote work has brought its own unique set of challenges to developer team workflows. These challenges increase exponentially when it comes to debugging a remote application.

Remote Debugging For Developer Team Managers

New times bring new challenges. As senior software engineers, system architects, or team leaders, we know that the challenge of finding and resolving bugs will always be a part of our daily work process. We also know that when debugging a remote application, one that is running on a cloud machine or in the customer's environment, the challenge increases exponentially.

The recent Covid-19 epidemic and its impact on remote development and debugging practices seems to further emphasize these challenges. Software engineers work away from the office, sometimes without direct access to the servers and tools that may help them get the data they need. Additionally, developers work in an isolated manner, without direct contact with their teammates or managers, which makes collaboration difficult and information sharing near impossible.

This whitepaper aims to provide an overview of challenges that developer teams and managers are facing today and solutions they can implement to overcome them.

KEY TAKEAWAYS:

The recent shift to remote work has brought its own unique set of challenges to developer team workflows. These challenges increase exponentially when it comes to debugging a remote application.

The latest disruption to remote debugging has come in the form of working remotely.

There are three classic approaches to remote debugging- such as a debugger in the developer's favorite IDE, logging, or APMs. We took a look at their pros and cons.

To keep up with the changing times, new tools are needed to complement these classic choices: 1. Collaboration, project management, and software development cycle tools 2. Tools that track and document tasks and code changes 3. Live debugging tools

Live debugging tools are aimed at remote debugging and help to improve the effectiveness of remote teams struggling with remote debugging tools.

Choosing the right tool to let your team dynamically connect to serverless functions will significantly improve their velocity and greatly reduce resolution time.

Classic debugging challenges

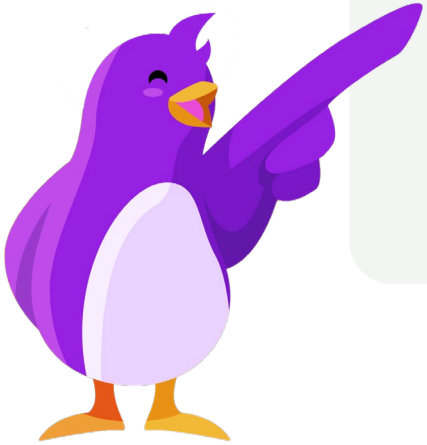
Remote debugging has been a challenge almost since the dawn of the software industry. Having direct access to the server where an application is running, knowing which version of the code is running it, having visibility into the log lines and variable values have been with us for decades now.

In the classic sense of the term, remote debugging means debugging a machine other than the laptop a developer is using to write their code locally. The remote application could be running on another server in the same room, on a cloud machine, on a virtual machine or docker container to which we may not have direct access, or it could be running as a kubernetes pod or a serverless function - a dynamic, transient and almost unreachable host which we would need to attach to in order to debug our application.

Classic debugging challenges entail discerning where the issue has happened, finding a way to connect to that location and fetch data from it, as well as doing all of this within a reasonable time and without halting the application. The traditional methods for achieving this are either by using a classic debugger and attaching it to a remote process (which requires an intimate familiarity with its location, and the right access permissions, and the ability to halt it), or by printing a significant number of log lines, assuring they are pipelined to a centralized logging server and later sieving through them to understand what happened as the remote issue happened.

A new generation of challenges

In recent years, technology and software development methods have been going through a continuous revolution, aiming to increase velocity and improve the quality of applications being developed. DevOps has become a prominent practice, motivating development teams to automate their testing, building, deployment and data collection flows. When applied effectively, DevOps could dramatically increase the rate in which an application changes, as new components, could be released and deployed several times a day.



> *The challenge of finding and resolving bugs will always be a part of our daily work process.*

Microservices have been adopted as a common practice, leading software engineers to break the logic of their applications into byte size, well-siloed components, that can be dynamically scaled up or down as necessary. Also, serverless technology has provided IT organizations with a complete abstraction layer between the software being written and the hardware it requires, along with a cost effective way to scale out and deploy virtual functions without worrying about hardware cost.

These newly adopted methods and tool sets have indeed increased the speed and agility of development teams across the industry. However, when it comes to remote debugging, it has also raised new challenges, and made classic challenges better felt by the teams.

> ***When debugging a remote application the challenge increases exponentially.***

DevOps means that software versions keep changing, and that there is no human interference with the deployment cycle. When an issue is discovered in a remote environment, this raises new unknowns. Which version is deployed in the machine where the issue was encountered? Which configuration was used by the automated deployment cycle? When has the change occurred?

Microservices and Serverless deployments take this challenge one step further. Is the "machine" where the incident occurred even there? If it was scaled up and scaled out dynamically, do I have a machine to connect to remotely? If not, how can I identify what was the cause for the issue?

The latest disruption

Working remotely from the office, or working in self isolation, brings its own set of challenges to remote debugging. One of the most painful aspects of this disruption is the lack of direct access to a developer's teammates or manager. For example: not being able to quickly ask a question about an unidentified piece of code or not being able to find out who changed it and why.
Specifically regarding remote debugging, that means not being able to share information about remote debugging tools and practices, or not being able to quickly get more information about a new incident being investigated. Which debugger should I use? What configuration should I use to gain access to a remote server? What is the most effective way for me to add a log line, or to find the relevant log line in our logging system?

Remote debugging in days of remote location brings a new spin to classic challenges. To address these challenges, senior developers and managers need to consider which tools and practices will help their teams debug efficiently. To assist those decisions, we have listed the available tools and methods and highlighted their advantages and disadvantages in assisting remote debugging.

> ***The new challenges brought by forced remote working and self isolation, require a new set of tools to complement the classic remote debugging tool choices.***

A closer look at the classics

Traditional remote debugging means using the debugger in a developer's favorite IDE, in remote debugging mode. This entails running the remote application in a way that allows debugging to begin with (including debug information and exposing the server to a remote connection that will take over the running process and allow halting it for debugging purposes). The advantage of this approach is that a developer has a familiar, robust debugging experience. They get to see the code as it runs and step-step-step through the unexpected behavior, thus learning the value of each variable as the issue is reproduced. However, when it comes to debugging live applications, this approach has a significant disadvantage. Namely, debugging the application causes it to halt. On top of that, in cloud-based applications, and specifically those who run in a production configuration or on the customer's environment, running with debug information or giving a developer access to connect remotely is often considered risky from a performance and security perspective.

Another classic approach to remote debugging is using logs. This is done by printing log statements, pipelining them to a centralized logging server, and searching through them when an issue occurs. The advantage of this approach is that it does not halt the application and it does not require granting developers access to a remote server. However, printing log lines requires additional coding effort. Adding a log statement means writing more code, going through a build and release cycle, and waiting for the new version of the code to be in place before getting access to the new debug information. When an urgent issue occurs in a remote server, this "wasted" time could be critical and delay the resolution of the problem. On top of that, collecting and pipelining logs has a performance impact on the running application, and storing logs incurs a monetary cost that many IT organizations work hard to minimize.

The third classic approach to remote debugging is using alerting, monitoring and exception management solutions. Application Performance Monitoring tools, or APM as they are usually known, allow software development teams to collect information about running

> *Live debugging tools are specifically designed and built to let engineers debug remote applications, without stopping the application.*

applications, alert the developers when an exception occurs, and collect server behavior metrics and profiling data without halting the application. Such tools are often quite robust and comprehensive, and allow developers to get a deep understanding of their application without waiting for log lines and without the need to connect to a remote server. However, these tools are often quite expensive and tricky to deploy and manage. Furthermore, in most cases they don't allow actual debugging. Namely, they don't collect a full variable trace, instead opting to collect a set of predefined metrics that hint at the source of the problem.

Disrupting the disruption

The changing times, and specifically the new challenges brought by forced remote working and self isolation, require a new set of tools to complement the classic remote debugging tool choices. In some cases, these are familiar tools that have become prominent in the new work circumstances. In other cases, these are brand new tools that disrupt the world of software development and remote debugging. Let us dig into those tool families.

The first group of repurposed tools is the family of collaboration and conferencing tools. These are tools that let developers communicate interactively and discuss upcoming code changes, document changes made, and track changes that may have led to an issue in a remote application. Chat and video conferencing tools such as Slack, Teams, Zoom, and Webex allow developers to communicate quickly and efficiently, even when they are not in the same room. These tools also adapt to become a de-facto knowledge sharing and change tracking tools, as discussions and decision made and information shared remain documented for search and auditing after an incident is identified. While not specifically designed to be used for remote debugging, these tools have become a prominent fixture of remote debugging flows in recent years.

A natural complement to chat and conversation tools are tools aimed specifically at tracking and documenting tasks and code changes. Project management tools like Jira and TFS let software engineers document the intended design for an upcoming code change and track the identification, investigation, and eventual resolution of remote bugs that required a code change to be fixed. A step further in this direction is taken by source control management tools such as Github, Perforce, and others. These tools track changes made to the code, and correlate them to a specific point in time, often to a specific feature or bug fix that prompted a code change. This form of code change auditing is often used by software engineers when trying to understand what changes made to a remote application may have caused an incident.

A third group of tools that emerged in recent years is directly aimed at remote debugging and can be used to significantly improve the effectiveness of remote teams struggling with remote debugging tools. Live debugging tools are specifically designed and built to let engineers debug remote applications, without stopping the application. These tools also solve the problems that some classic debugging tools struggle with: they are able to dynamically connect to running applications without stopping them and without requiring intimate access to specific servers. They let developers add and remove log lines without requiring a full development and release cycle and they are tailored to remote teams attempting to collaborate, as they are online and collaborative in nature.

Looking to the future

The world of software development in general, and remote debugging specifically, will keep changing around us at an ever increasing pace. The changes made to it in recent years raised the need for efficient collaboration and remote debugging tools. When choosing a remote debugging strategy, a development leader will have to choose between a mix of classic and next generation tools to keep remote debugging efficient and safe.

Picking a tool that will let your team dynamically connect to serverless functions or microservices, without halting the application and without requiring a code change, will significantly improve the efficiency of the engineers and will reduce resolution time dramatically.
Choosing an online tool that naturally integrates with collaboration and source control management tools will take this impact one step further, allowing engineers to be as efficient as they are when working together from the same room.

BOOK A DEMO