Cymbal Direct’s warehouse and

inventory system was written in Java.

The system uses a microservices

architecture in GKE and is

instrumented with Zipkin. Seemingly

at random, a request will be 5-10

times slower than others. The

development team tried to reproduce

the problem in testing, but failed to

determine the cause of the issue.

What should you do?

A. Create metrics in Cloud Monitoring for your microservices to

test whether they are intermittently unavailable or slow to

respond to HTTPS requests. Use Cloud Profiler to determine

which functions/methods in your application’s code use the most

system resources. Use Cloud Trace to identify slow requests and

determine which microservices/calls take the most time to respond.

B. Create metrics in Cloud Monitoring for your microservices to test whether they are intermittently

unavailable or slow to respond to HTTPS requests. Use Cloud Trace to determine which

functions/methods in your application’s code use the most system resources. Use Cloud Profiler to

identify slow requests and determine which microservices/calls take the most time to respond.

C. Use Error Reporting to test whether your microservices are intermittently unavailable or slow to

respond to HTTPS requests. Use Cloud Profiler to determine which functions/methods in your

application’s code use the most system resources. Use Cloud Trace to identify slow requests and

determine which microservices/calls take the most time to respond.

D. Use Error Reporting to test whether your microservices are intermittently unavailable or slow to

respond to HTTPS requests. Use Cloud Trace to determine which functions/methods in your

application’s code Use the most system resources. Use Cloud Profiler to identify slow requests and

determine which microservices/calls take the most time to respond.

**A. Create metrics in Cloud Monitoring for your microservices to test whether they are intermittently unavailable or slow to respond to HTTPS requests. Use Cloud Profiler to determine which functions/methods in your application’s code use the most system resources. Use Cloud Trace to identify slow requests and determine which microservices/calls take the most time to respond.**

**Explanation:**

This option provides a comprehensive approach to diagnosing the intermittent performance issue:

* **Cloud Monitoring:** Tracks the overall health and performance of microservices, identifying potential bottlenecks or outages.
* **Cloud Profiler:** Provides insights into code-level performance, helping pinpoint resource-intensive functions or methods.
* **Cloud Trace:** Offers visibility into request flow and latency, enabling identification of slow-performing microservices or API calls.

By combining these tools, you can effectively isolate the root cause of the intermittent slowdowns, whether it's a specific microservice, a code-level issue, or an underlying infrastructure problem.

**Why not the other options:**

* **B:** While Cloud Monitoring and Cloud Trace are essential, Cloud Profiler is better suited for analyzing code-level performance than Cloud Trace.
* **C and D:** Error Reporting is primarily for tracking errors, not performance issues. Using it to monitor availability or response times is less effective than Cloud Monitoring.

By implementing option A, Cymbal Direct can gain valuable insights into the performance of its microservices and take corrective actions to improve system reliability.

Cymbal Direct releases new versions of

its drone delivery software every 1.5 to

2 months. Although most releases are

successful, you have experienced

three problematic releases that made

drone delivery unavailable while

software developers rolled back the

release. You want to increase the

reliability of software releases and

prevent similar problems in the future.

What should you do?

A. Adopt a “waterfall” development process. Maintain the current

release schedule. Ensure that documentation explains how all

the features interact. Ensure that the entire application is tested

in a staging environment before the release. Ensure that the process

to roll back the release is documented. Use Cloud Monitoring,

Cloud Logging, and Cloud Alerting to ensure visibility.

B. Adopt a “waterfall” development process. Maintain the current release schedule. Ensure that

documentation explains how all the features interact. Automate testing of the application. Ensure that

the process to roll back the release is well documented. Use Cloud Monitoring, Cloud Logging, and

Cloud Alerting to ensure visibility.

C. Adopt an “agile” development process. Maintain the current release schedule. Automate build

processes from a source repository. Automate testing after the build process. Use Cloud Monitoring,

Cloud Logging, and Cloud Alerting to ensure visibility. Deploy the previous version if problems are

detected and you need to roll back.

D. Adopt an “agile” development process. Reduce the time between releases as much as possible.

Automate the build process from a source repository, which includes versioning and self-testing. Use

Cloud Monitoring, Cloud Logging, and Cloud Alerting to ensure visibility. Use a canary deployment to

detect issues that could cause rollback.

**D. Adopt an “agile” development process. Reduce the time between releases as much as possible. Automate the build process from a source repository, which includes versioning and self-testing. Use Cloud Monitoring, Cloud Logging, and Cloud Alerting to ensure visibility. Use a canary deployment to detect issues that could cause rollback.**

[1. What is CI/CD? - GitLab](https://about.gitlab.com/topics/ci-cd/" \l ":~:text=CI%2FCD%20automates%20much%20or,%2C%20encompassing%20the%20build%2C%20test%20(" \t "_blank)

[[Icône de la source](https://about.gitlab.com/topics/ci-cd/#:~:text=CI%2FCD%20automates%20much%20or,%2C%20encompassing%20the%20build%2C%20test%20()](https://about.gitlab.com/topics/ci-cd/" \l ":~:text=CI%2FCD%20automates%20much%20or,%2C%20encompassing%20the%20build%2C%20test%20(" \t "_blank)

[about.gitlab.com](https://about.gitlab.com/topics/ci-cd/" \l ":~:text=CI%2FCD%20automates%20much%20or,%2C%20encompassing%20the%20build%2C%20test%20(" \t "_blank)

[2. Canary Deployment Explained: Minimizing Risks & Optimizing Releases - Enov8](https://www.enov8.com/blog/canary-deployment-explained/" \l ":~:text=Faster%20Rollback%20and%20Recovery%3A%20In,for%20swift%20rollback%20to%20the" \t "_blank)

[[Icône de la source](https://www.enov8.com/blog/canary-deployment-explained/#:~:text=Faster%20Rollback%20and%20Recovery%3A%20In,for%20swift%20rollback%20to%20the)](https://www.enov8.com/blog/canary-deployment-explained/" \l ":~:text=Faster%20Rollback%20and%20Recovery%3A%20In,for%20swift%20rollback%20to%20the" \t "_blank)

[www.enov8.com](https://www.enov8.com/blog/canary-deployment-explained/" \l ":~:text=Faster%20Rollback%20and%20Recovery%3A%20In,for%20swift%20rollback%20to%20the" \t "_blank)

**Explanation:**

This option addresses the core issues of infrequent, high-risk releases by incorporating:

* **Agile development:** Encourages iterative development, faster feedback loops, and adaptability.
* **Frequent releases:** Reduces the impact of issues by limiting the scope of each release.
* **Automation:** Streamlines the build and testing process, reducing human error.
* **Self-testing:** Improves code quality and reduces defects.
* **Monitoring and alerting:** Provides visibility into system behavior and enables timely responses.
* **Canary deployment:** Minimizes the risk of widespread issues by gradually rolling out the new version to a subset of users.

By combining these elements, Cymbal Direct can significantly increase the reliability of its software releases and reduce the frequency of problematic deployments.

**Breakdown of other options:**

* **A and B:** Waterfall methodologies are generally slower and less adaptable to change, which is counterproductive in this scenario.
* **C:** While agile development is a good choice, maintaining the current release schedule and relying on manual rollbacks doesn't address the core issue of reducing release risk.

Option D offers a comprehensive approach to improving software release reliability and mitigating the impact of potential issues.

**Would you like to explore any of these options in more detail or discuss specific implementation strategies?**