Debugging, Monitoring, and Performance Tuning

Developing Applications with Google Cloud Platform

STACKDRIVER LOGGING, STACKDRIVER MONITORING, STACKDRIVER TRACE

QWIKLABS DEBUGGING APPLICATION ERRORS

QWIKLABS HARNESSING STACKDRIVER TRACE AND MONITORING

Google Cloud

Version 2.0
Last modified: 2017-09-25

Google Stackdriver is a multi-cloud service

Error Reporting

Error notifications
Error dashboard

Debugger

Production debug snapshots
Conditional snapshots
IDE integration

Logging

Platform, system, and app logs
Log search/view/filter
Logs-based metrics

Monitoring

Platform, system, and app metrics
Uptime/health checks
Dashboards
Alerts

Trace

Latency Reporting
Per-URL latency sampling

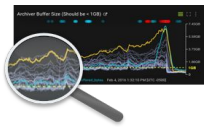
Google Cloud

Stackdriver enables debugging, monitoring, and diagnostics for applications that run on Google Cloud Platform and AWS.
Stackdriver comprises features such as Stackdriver Logging, Stackdriver Monitoring, Stackdriver Trace, Stackdriver Error Reporting, and Stackdriver Debug. These diagnostics features are well-integrated with each other. This helps you connect and correlate diagnostics data easily.

Stackdriver enables you to increase application reliability

Monitor GCP, AWS, and Multi-Cloud Environments

Get the insight that you need with minimal configuration. Monitor hosted services and cloud architectures.



Identify Trends, Prevent Issues

Visualize trends via flexible charts and dashboards. Identify risks using scoring, anomaly detection, and prediction.

Reduce Monitoring Overhead

Spend less time correlating metrics, alerts, and logs across disparate systems. Don't worry about scaling tools.

Improve Signal-to-Noise

Reduce false positives and alert fatigue with advanced alerting designed for modern distributed systems.

Fix Problems Faster

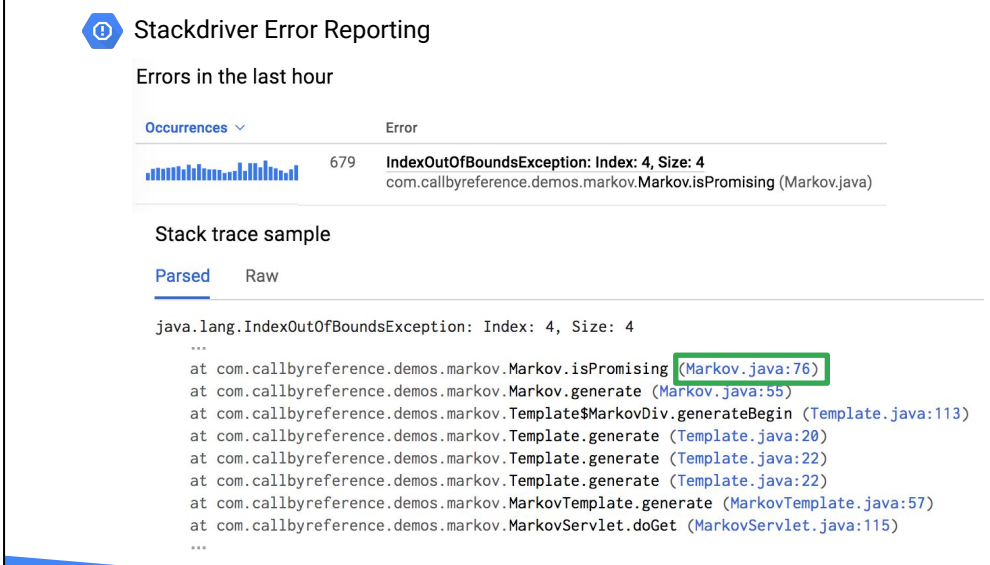
Uptime and health checks notify you quickly when endpoints become inaccessible to your users. Drill down from alerts to dashboards to logs and traces to get to the root cause quickly.

Google Cloud

Stackdriver helps increase reliability by giving users the ability to monitor GCP and multi-cloud environments and identify trends to prevent issues. With Stackdriver, you can reduce monitoring overhead and noise to fix problems faster.

Debugging Your Application

Debug your application in development and production

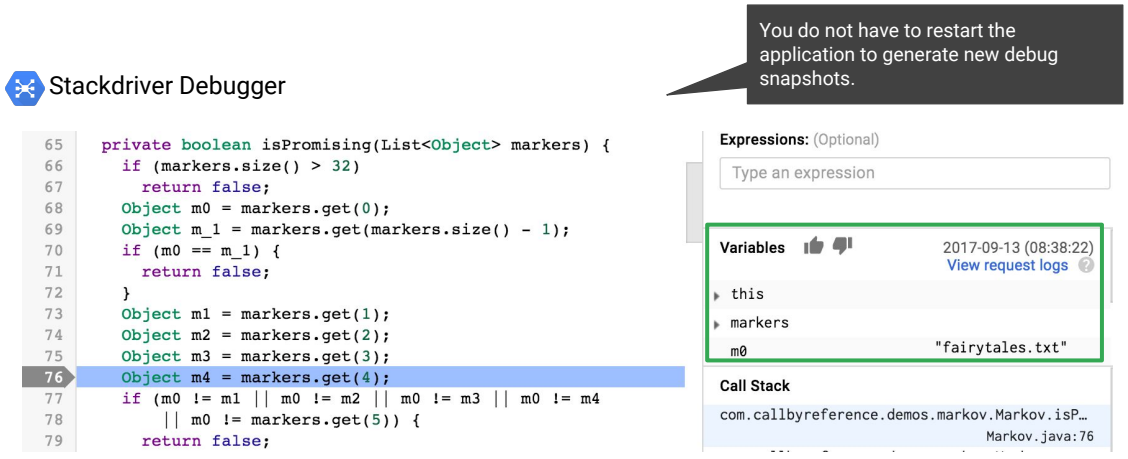


With Stackdriver Error Reporting and Debugger, you can debug your applications in development and also troubleshoot errors in production.

Error Reporting displays errors that have occurred in your applications. You can view the stack trace to determine where the error occurred. Clicking the source code file in the stack trace takes you to Stackdriver Debugger and the line of code that has a problem.

For more information about Error Reporting, see https://cloud.google.com/error-reporting/.

Debugger automatically creates debug snapshots



Google Cloud

Debugger automatically creates a debug snapshot on the line that has the error. When the application hits the line of code again, Stackdriver Debugger creates a snapshot of the application's state, including values of local variables. You can also manually select other lines of code where debug snapshots would be helpful.

You do not have to restart the application to generate new debug snapshots. This powerful feature of Stackdriver Debugger lets you inject logging information without interfering with the normal function of the application.

For more information about Debugger, see https://cloud.google.com/debugger/.

Logging

Install Stackdriver Logging Agent to capture logs

You can install Stackdriver Logging Agent on Compute Engine and Amazon EC2 instances to stream logs from third-party applications into Stackdriver Logging. The Logging agent is an application based on fluentd. When you write your logs to existing log files such as *syslog* on your VM instance, the Logging agent sends the logs to Stackdriver.

For more information, see https://cloud.google.com/logging/docs/agent/.

Stackdriver Logging is preconfigured in other compute environments

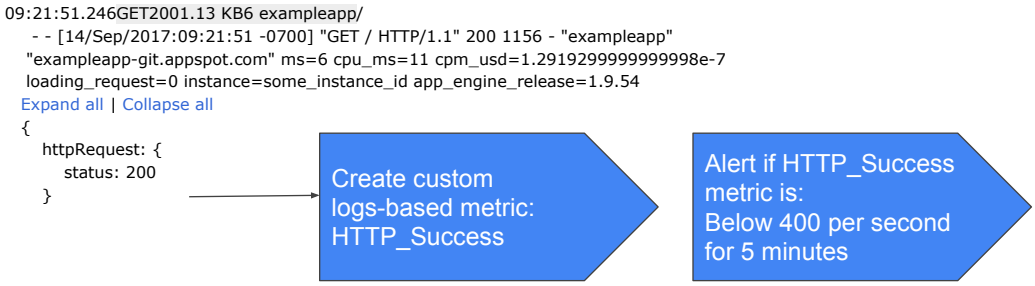
- Google Cloud Dataflow
- Google Cloud Functions
- Google App Engine Flexible and Standard Environments
- Google Container Engine

Google Cloud

Cloud Dataflow, Cloud Functions, and App Engine have built-in support for Logging. You can enable Logging on Container Engine by simply enabling a checkbox in the Cloud Platform Console when you set up a container cluster.

Set up logs-based metrics and alerts

Stackdriver Logging and Monitoring



In Stackdriver Logging, you can view your logs and search for particular types of messages. You can create custom logs-based metrics and alerts based on these metrics. The example shows the following:

- A logs-based metric called HTTP_Success that filters all log messages with an HTTP response code of 200
- An alert to notify you when the number of successful HTTP requests is below 400 per second for 5 minutes.

Monitoring and Tuning Performance

Google Cloud

In this section, you will learn the following:

- Why it is important to monitor your application
- What you need to monitor
- How to troubleshoot performance issues in development and production

Monitor to analyze long-term trends

Google Cloud

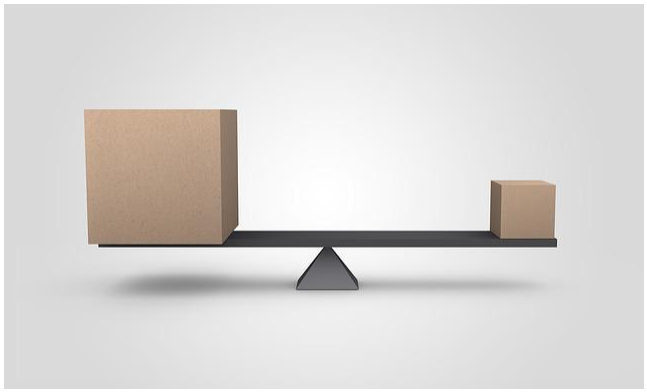
When you monitor your application and gather metrics, you can make improvements to the design of your application, increase reliability, detect and fix security issues, and reduce costs based on usage patterns.

Monitor to analyze long-term trends and answer questions such as the following:

- How fast is my database growing?
- How many users have I added in the last four quarters?

Images:
https://pixabay.com/en/stock-exchange-boom-economy-pay-2648118/

Monitor to compare results over time or between experimental configurations



Google Cloud

Monitor to compare results over time or between experimental configurations. For example:

- Is the site slower than it was last week?
- Are requests served faster with Apache or Nginx web server?

Images:
https://pixabay.com/en/balance-swing-equality-measurement-2108025/

Monitor to raise alerts when something is broken or about to be broken

Google Cloud

Monitor to raise alerts when something is broken and should be fixed urgently or something is about to be broken and can be addressed preemptively.

Images:
https://pixabay.com/en/attention-warning-sign-symbol-icon-2584482/

Monitor to perform ad hoc retrospective analysis

Monitor to perform ad hoc retrospective analysis. For example, the latency for your application just increased sharply; what else happened around the same time?

Images:
https://pixabay.com/en/analytics-charts-graphics-marketing-2618277/

Identify APIs and resources that you want to monitor

Examples

- Public and private endpoints
- Multi-cloud resources such as Compute Engine VM instances, Cloud Storage buckets, Amazon EC2 instances, and Amazon RDS databases

Google Cloud

Identify APIs and resources that you want to monitor.

Identify service-level indicators and objectives

Service-Level Indicator (SLI): Latency
Service-Level Objective (SLO): 99.9% of requests over 30 days have latency <100ms

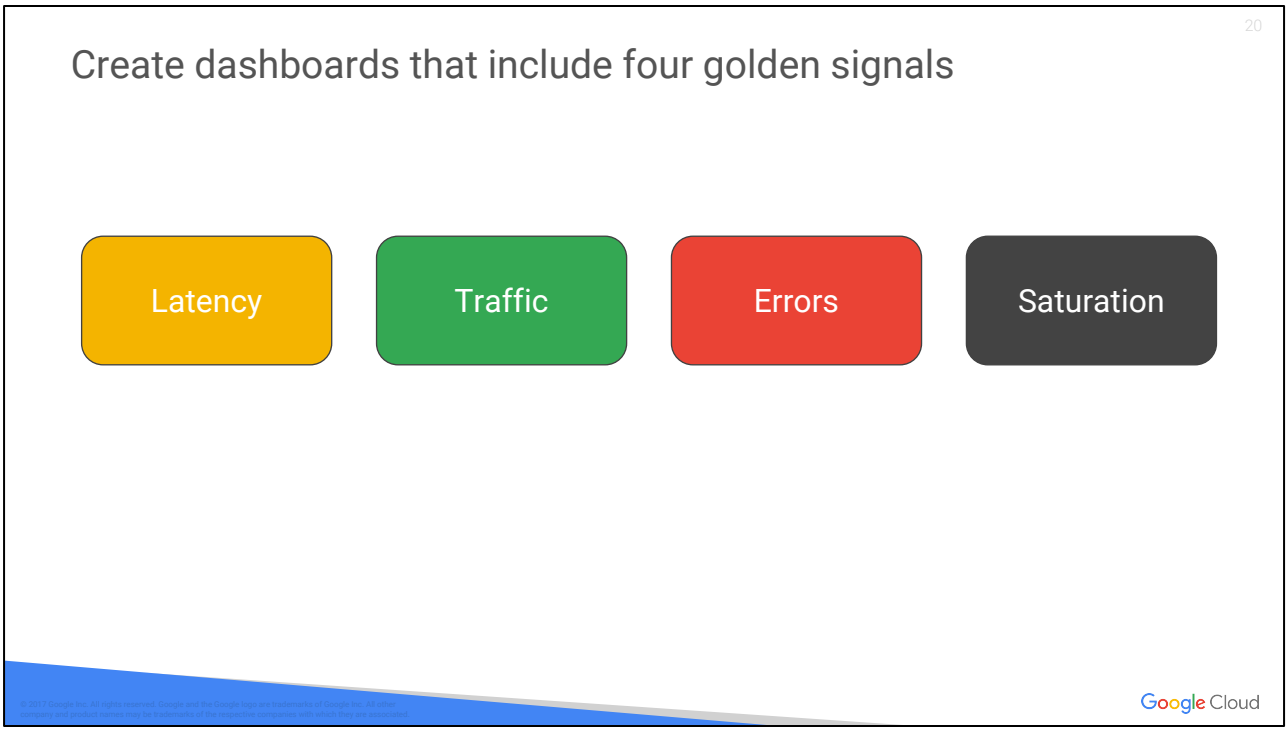
Google Cloud

A service-level indicator (SLI) is a quantitative measure of some aspect of a service. An SLI might be a predefined metric or a custom metric such as a logs-based metric.

A service-level objective (SLO) is a target value or range of values for a service level. SLOs should include tolerance for small variations. Absolute limits will result in noisy pagers and require you to constantly tweak thresholds.

For example, latency is a service-level indicator. You can set a service-level objective indicating that 99.9% of requests over 30 days have latency <100ms.

Images:
https://pixabay.com/en/gauge-icons-performance-1294568/



After you identify the resources to monitor and define service-level indicators and objectives, you can create dashboards to view metrics for your application. Create dashboards that include the following golden signals:

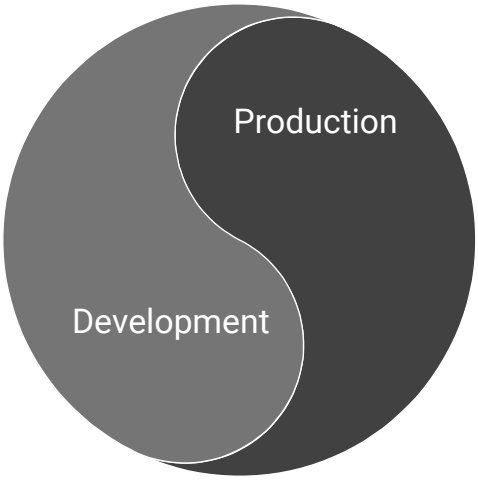
- Latency: Latency is the amount of time it takes to serve a request. Make sure to distinguish between the latency for successful and unsuccessful requests. For example, an HTTP 500 error that occurs due to a loss of connection to a database or another backend service might be served very quickly; however, because an HTTP 500 error indicates a failed request, including 500 errors in your overall latency might result in misleading metrics.
- Traffic: Traffic is a measure of how much demand is placed on your system. It is measured as a system-specific metric. For example, web server traffic is measured as the number of HTTP/S requests per second. Traffic to a NoSQL database is measured as the number of read or write operations per second.
- Errors: Errors indicate the number of failed requests. Criteria for failure might be any of the following:
    - An explicit error such as an HTTP 500 error
    - A successful HTTP 200 response but with incorrect content
    - A policy error; for example, your application promises a response time of 1 second, but some requests take over a second
- Saturation: Saturation indicates how “full” your application is or what resources are being stretched and reaching target capacity. Systems can degrade in performance before they achieve 100% utilization, so make sure to set utilization targets carefully.

Identifying and Troubleshooting Performance Issues

Google Cloud

This section describes areas that you can review to troubleshoot and resolve performance issues.

Monitor performance in development and production



Google Cloud

It is important to monitor performance in the development phase and in production.

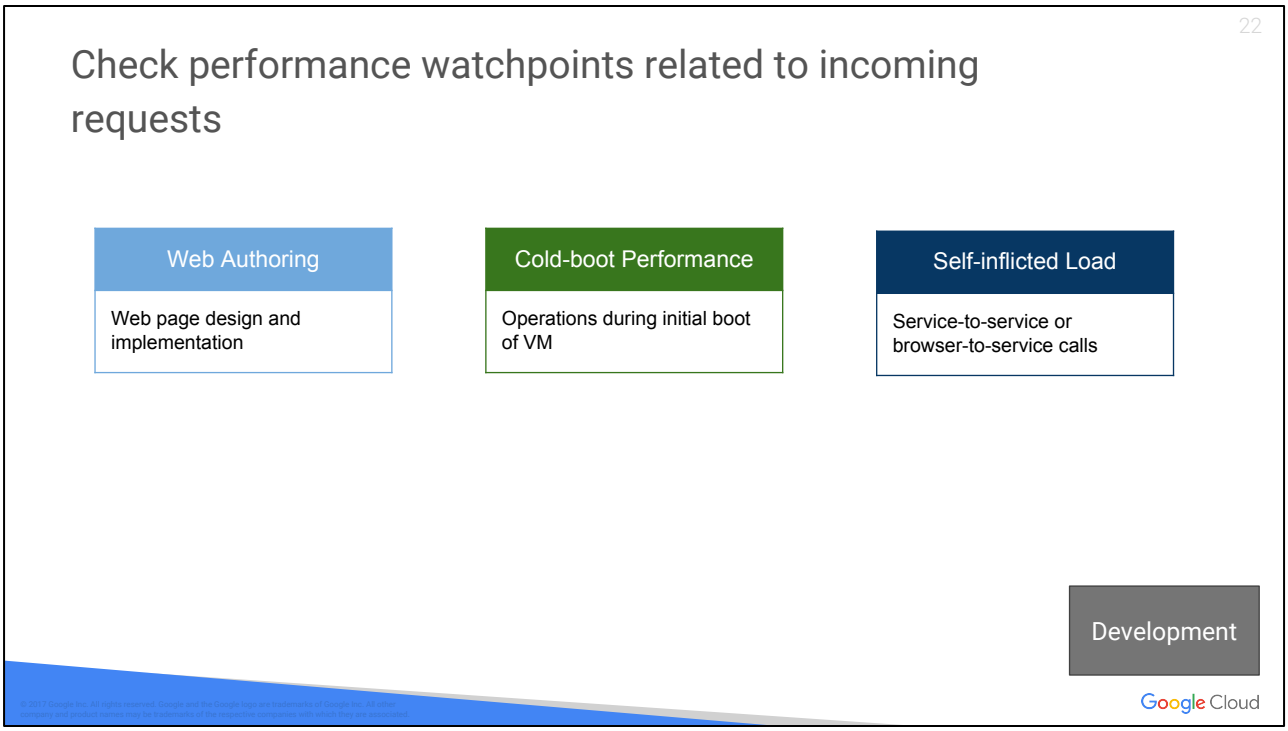
Add performance tests to your test suite

Development

Google Cloud

Add performance tests to your test suite to ensure that the performance of your application does not degrade when you fix bugs, add new features, or change underlying software. Response times and resource requirements can change significantly when you make changes to your application. With performance tests, you will be able to detect and address performance issues early in the development process.

Images
https://pixabay.com/en/running-sprinting-speed-start-run-150493/
https://pixabay.com/en/sport-train-active-fitness-1014015/
https://pixabay.com/en/gauge-icons-performance-1294568/



A watchpoint is a potential area of configuration or application code that could indicate a performance issue. Performance issues may be a result of multiple watchpoints.

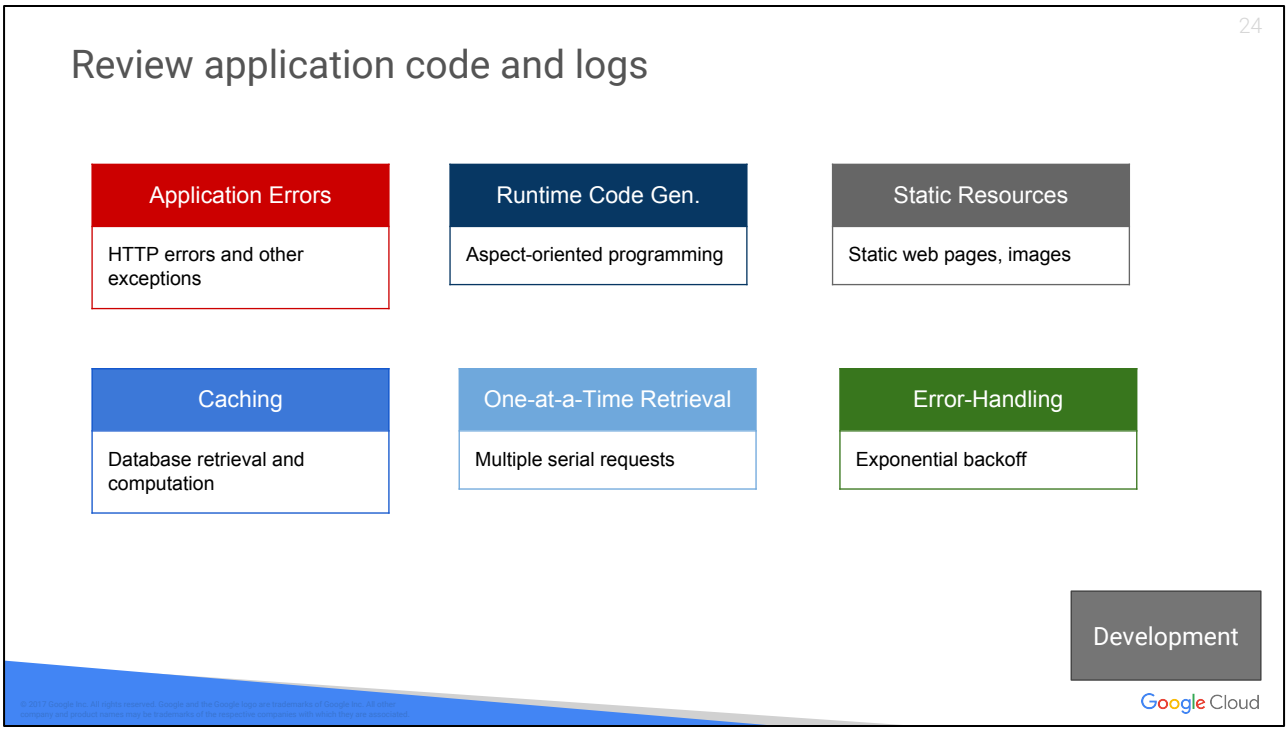
Review metrics related to incoming requests and check the following areas for performance issues:

- Web authoring: Review the design and implementation of your web pages. You can use PageSpeed Insights to view information about the lack of caching headers, the lack of compression, too many HTTP browser requests, slow DNS response, and the lack of minification. If the application is not public-facing, you can use Chrome DevTools with PageSpeed Insights to manually analyze your web pages.
- Cold-boot performance: Use tracing to understand how much time is spent on each operation. Ensure that the code only references JAR files or external libraries that are absolutely needed. Reducing binary size can help cold boot time. Optimize code paths, use previously cached information, and remove unnecessary logic in your startup code.
- Self-inflicted load: Look for load caused by the application itself, such as service-to-service or browser-to-service calls. For example, check for polling, cron jobs, batch requests, or multiple AJAX requests from browser. Consider using client-side (Chrome DevTools) and server-side load analysis tools (Stackdriver Trace).

For more information about client-side analysis tools, see:

- Chrome Dev Tools: https://developer.chrome.com/devtools

- PageSpeed Insights: https://developers.google.com/speed/pagespeed/insights/

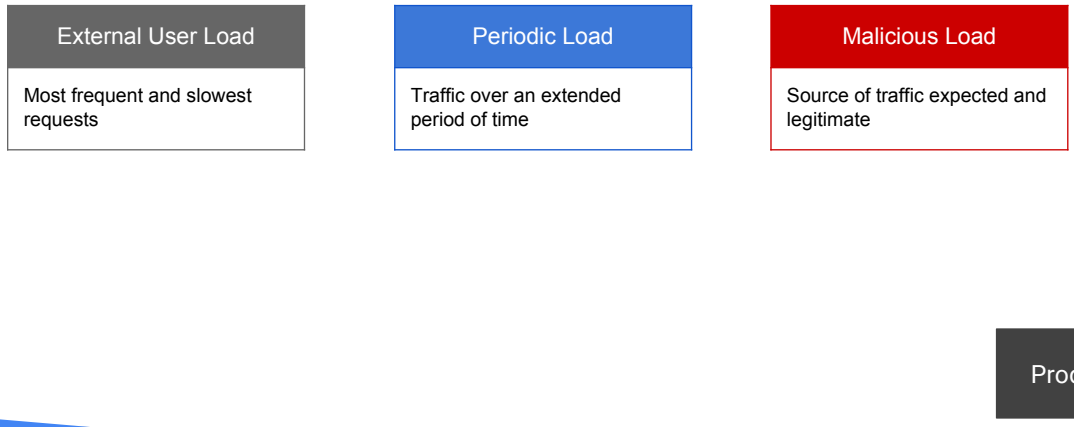


Review application code and logs as follows to check for performance issues:

- Application errors: Check logs for HTTP errors (4xx, 5xx) and application errors and exceptions. Identify the root cause of the log messages and confirm that they are not related to periodic load or performance issues. Prioritize investigation by the frequency of errors. Because this data is historical, some errors might have been intermittent or already resolved. If a log message is unreproducible, it might be better to defer the investigation.
- Runtime code generation: Aspect-oriented programming practices can sometimes cause reduced application performance. Consider compile-time code generation instead. For more information about aspect-oriented programming, see https://en.wikipedia.org/wiki/Aspect-oriented_programming.
- Static resources: Do not serve static resources from your application. Instead, use a content delivery network (CDN) such as Google Cloud CDN with Google Cloud Storage.
- Caching: Consider caching frequently accessed values that are retrieved from a database or require significant compute resources to recalculate. You can also cache generated HTML fragments for later use.
- ~~One-at-a-Time Retrieval~~ One-at-a-time retrieval: Look for areas where data is retrieved from a database or service with multiple requests in serial. Replace these individual requests with a single batch request or send the requests in parallel.
- Error handling: Do not retry constantly on errors; instead, retry with exponential backoff. Implement a circuit breaker to stop retries after a certain number of failures. Note that you should only retry in case of errors such as

- connection timeouts or too many requests. Do not retry in case of errors such as 5xx errors and malformed URL errors.

Check performance watchpoints related to incoming requests in production

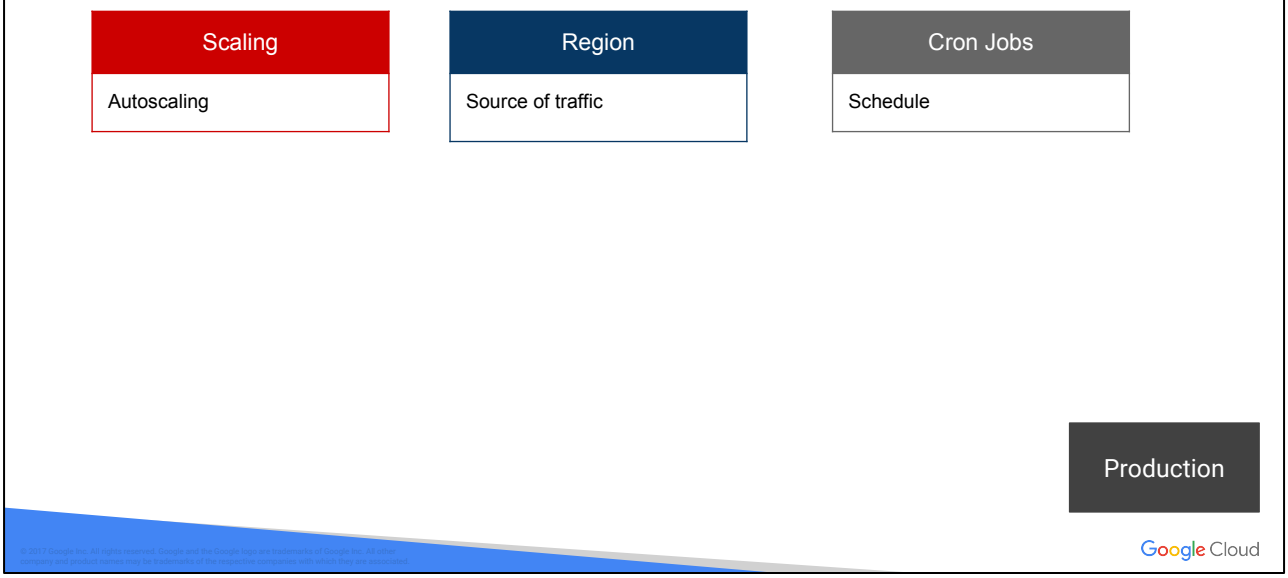


Google Cloud

Monitor incoming requests to check the following areas:

- External user load: Analyze the most frequent requests and slowest requests. Confirm that the requests are expected. Determine the cause for the slowest requests.
- Periodic load: Analyze traffic over an extended period of time to determine which periods have higher levels of usage. Confirm that there is a business reason for this load.
- Malicious load: Confirm that all load is expected and legitimate. If you have a web application, make sure that all load is coming from a web or mobile client. You can further segment the load by user to understand whether the majority of requests are coming from a small number of users.

Review deployment settings



Review deployment settings as follows:

- Scaling: Make sure that you have set up load balancing and autoscaling policies as appropriate for the traffic volumes for your application. Set target utilization levels conservatively to ensure that your application continues to handle traffic while new VM instances come online.
- Region: Determine where the bulk of your traffic is coming from. Deploy resources in the appropriate regions to reduce latency.
- Cron jobs: Make sure that your cron jobs are scheduled accurately.

Review and implement best practices for each of the services that you are using in your application.

Traditional SRE blessing

May the queries flow, and the pager stay silent.

Google Cloud

SRE stands for Site Reliability Engineering. For more information about site reliability engineering, see https://landing.google.com/sre/book.html.