100 XP

Perform ongoing tuning to reduce meaningless alerts

6 minutes

In this part, you learn about processes that you can implement to monitor site reliability. You also learn about ongoing tuning of your alerts to reduce meaningless alerts.

The meeting

Amita: I think they need to show us how well they can use Application Insights and Smart Detection. We may all be affected once we start getting alerts. What should alerts be about? What kinds of alerts are there?

Mara: Maybe we should come up with a sort of style guide.

Monitoring and alerting

Monitoring and alerting enables a system to tell people when it's broken, or perhaps to tell them what's about to break. If someone needs to investigate the problem, the alert should give relevant information so the person knows where to start.

When you review existing alerts or write new alert rules, consider these guidelines to keep your alerts relevant and your on-call rotation happier:

- Alerts that trigger a human's attention should be urgent, important, action-oriented, and real.
- Alerts should represent either ongoing or imminent problems with your service.
- Remove noisy alerts. Over-monitoring is a harder problem to solve than under-monitoring.
- Classify the problem into one of these categories:
  - Availability and basic functionality.
  - Latency.
  - Correctness.
  - Feature-specific problems.
- Symptoms are a better way to capture problems comprehensively and robustly with less effort.
- Include cause-based information in symptom-based pages or on dashboards but avoid alerting directly on causes.

- The further up your serving stack you go, the more distinct problems you catch in a single rule. But don't go so far that you can't sufficiently distinguish what's going on.
- If you want a quiet on-call rotation, have a system for dealing with issues that need a timely response but are not imminently critical.

Monitor for your users

Monitoring for your users is also called *symptom-based monitoring*. This is in contrast to *cause-based monitoring*. Your users don't care if your data push is failing, they care about whether their results are fresh.

In general, users care about:

- Basic availability and correctness.

  Anything that breaks the core service in some way should be categorized as unavailability.

- Latency.

  Pages should load quickly.

- Completeness, freshness, and durability.

  Your users' data should be safe, should come back promptly, and search indices should be up to date.

- Uptime.

  Even if the service is temporarily unavailable, users should have complete faith that the service will be back up soon.

- Features.

  Your users care that all the features of the service work. Monitor for anything that is an important aspect of your service, even if it's not core functionality.

There's a subtle but important difference between database servers being unavailable and user data being unavailable. The former is an immediate cause, the latter is a symptom.

Cause-based alerts have their place

Sometimes there is no symptom to alert on but you still need to be alerted to a situation. An example is running low on memory. You want rules to notify you of issues that could become a problem before they cause a symptom. In this case, you can write a rule to alert on this condition.

However, don't write cause-based rules that trigger on call alerts for symptoms you can catch otherwise.

Tickets, reports, and email

Alerts that need attention soon, but not right away are *subcritical alerts*. Here are some suggestions for logging subcritical alerts to follow up on later:

- Bug or ticket-tracking systems can be useful for this type of alert.

  An alert can open a bug, as long as the same alert gets correctly placed in a single ticket or bug. These bugs can then go through a triage process to be assigned to someone to follow up on. It's important that these types of issues be addressed before they become critical. Take into account how much of your team members' time can be devoted to fixing bug.

- A daily (or more frequent) report can be useful.

  Write subcritical alerts that are long-lived, for example, the database is over 90% full, to a report that shows all active alerts. Assign someone to triage this report daily.

- Every alert should be tracked through a workflow system to ensure that they're being seen and addressed.

In general, create a system that promotes accountability for responsiveness, but doesn't have the high cost of immediate human intervention.

Playbooks

Playbooks, sometimes referred to as runbooks, are an important part of an alerting system. Have an entry in the playbook that explains what to do for each alert or family of alerts that catch a symptom.

Tracking and accountability

If someone receives an alert and determines that there's nothing wrong, that's a sign that you need to remove the rule, demote it, or collect data in some other way. Alerts that are less than 50% accurate are considered to be broken. Even those that trigger false positives 10% of the time merit re-evaluation.

Having a weekly review of all triggered on-call alerts and analyzing quarterly alert statistics can help you to see patterns that are lost when focusing on individual alerts.

When to seek the exception from the rule?

Here are some reasons you might break the above guidelines:

- You have a known cause that actually sits below the noise in your symptoms.

  For example, if your service has 99.99% availability, but you have a common event that causes 0.001% of requests to fail, you can't alert on it as a symptom because it's in the noise, but you can catch the causative event.

- You can't monitor at the entry point because you lose data resolution.

  For example, you tolerate some endpoints being slow, like credit card validation. At your load balancers, this distinction may be lost. You will need to walk down the stack and alert from the highest place where you have the distinction.

- Your symptoms don't appear until it's too late.

  For example, you've run out of quota. You need to alert someone before it's too late, and sometimes that means finding a cause to alert on. For example, your usage is greater than 80% and will run out in less than 4 hours at the growth rate of the last 1 hour.

  However, you should also be able to find a similar cause that's less urgent. For example, your quota is greater than 90% and will run out in less than four days at the growth rate of the last day. That set of circumstances will catch most cases. You can then deal with the problem as a ticket or email alert or daily problem report, rather than the last-ditch escalation that an alert represents.

- Your alert setup is more complex than the problems they're trying to detect.

  The goal should be to move towards simple, robust, self-protecting systems.

The last skill set

Andy: That's quite a list. Is there anything else?

Mara: They need good analytic skills.

Next unit: Analyze alerts to establish a baseline

Continue >