LogicMonitor

Monitoring & Alerting Best Practices Guide

Best practices for smarter alerting, faster troubleshooting, and more proactive monitoring

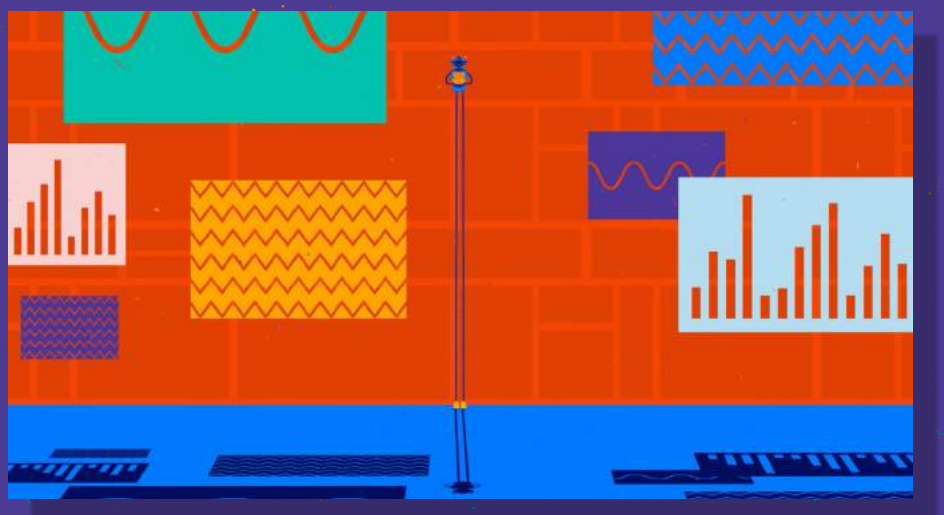


Table of Contents

Best Practices

Looking Forward

Alerts sit at the heart of a well designed monitoring system, an essential aspect of preventing downtime. But managing them can be one of the more time intensive parts of your day. How much time have you spent muting non-critical alerts—or filtering them out of your inbox entirely—trusting that you'll be notified by your manager when something truly requires your attention? Or how many times have you been woken up by an SMS alert for an issue that isn't truly critical?

By implementing an effective monitoring and alerting strategy, you can avoid these scenarios, freeing up time in your day.

This guide will provide alerting best practices to ensure three related outcomes:

1. Monitoring is in place to catch critical conditions and alert the right people
2. Noise is reduced and you or your team are not needlessly distracted
3. Time spent on alerts is reduced, creating more time for you to focus on proactive monitoring

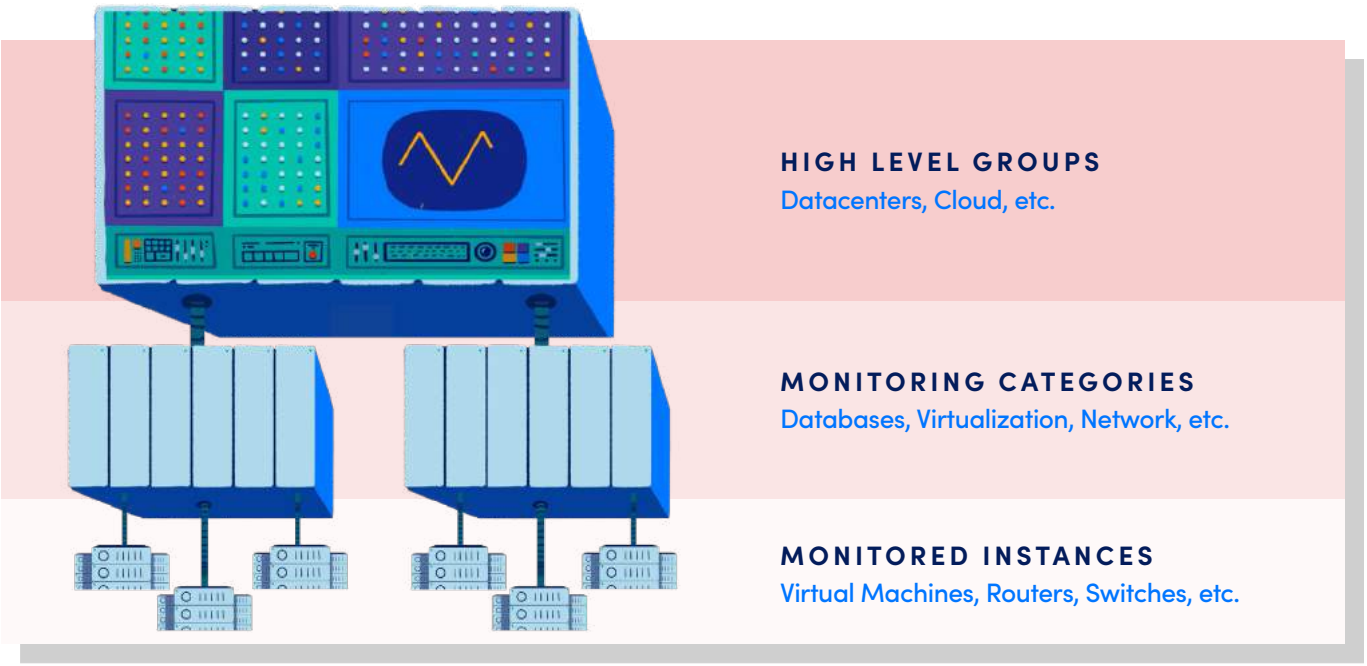
These outcomes are essential to a successful monitoring strategy, more effective troubleshooting, and more time in the day to tackle important projects that you've been meaning to get to. What follows are core best practices to consider when designing a new monitoring system or reviewing an existing system.

BEST PRACTICES

Setting Up Alerts

Alert Components

Monitoring solutions use different terminology to represent the items they monitor. Usually, there is a hierarchical system that includes high-level groups such as a datacenter, and then lower-level groups and individual hosts below that. At the host level there will be monitoring categories such as disk, CPU, etc., and below that specific instances to be monitored such as /dev/sda mounted at /. The diagram below shows an example of this:



Whatever the organizational structure of your monitoring solution, each item monitored should have the following components defined:

- **Metric**: The measure being monitored, such as CPU percent used.
- **Threshold**: The definition of when the metric is considered to be in a less than optimal state.
- **Alert Level**: The level of urgency associated with a given state—usually warning, error, or critical.
- **Action to Resolve**: The action items associated with the alert.
- **Alert Routing**: Where and how the alert is to be delivered.

Failure to consider the above components when developing a monitoring and alerting strategy will increase the likelihood of an alert system failure. Alert systems fail when they generate too much noise – which can result in staff missing or ignoring critical alerts – but they can also fail if they don't catch critical conditions due to either a lack of flexibility, insufficient monitoring capability, or bad configuration.

Alert Thresholds

Setting the correct threshold when rolling out a monitoring system can be difficult. Effective monitoring tools will have sensible defaults out-of-the-box, but these will need to be tuned over time. A warning alert on one set of systems could be a critical alert on others. For example, CPU usage of 96% is not necessarily bad for all systems, but it is for some.

Additionally, some monitoring tools provide the ability to configure alert thresholds automatically based on historic metric measurements. This offers a threshold baseline and ideally forecasts future issues based on threshold trends.

Alert Messaging

One of the most commonly overlooked requirements leading to alerts being ignored is the need to define action items. Effective monitoring solutions will have reasonable default descriptions that indicate what may be the cause of a particular condition.

Here is an example message associated with an alert on CPU usage:

"The 5 minute load average on server123 is now 99, putting it in a state of critical. This started at 2016-05-11 11:38:42 PDT. See which processes are consuming CPU (use 'ps' command to see which processes are in runnable and waiting state; 'top' command will show individual cpu core usage). Troubleshoot or adjust the threshold."

As a default message, this is pretty useful. It tells the recipient the basic steps for troubleshooting. These messages can be customized for known situations such as high traffic requiring more application servers to be added to a cluster.

Avoid implementing monitoring alerts that have no actionable responses. This adds noise and increases the frustration of those receiving the alerts.

Alert systems fail when they generate too much noise – which can result in staff missing or ignoring critical alerts – but they can also fail if they don't catch critical conditions due to either a lack of flexibility, insufficient monitoring capability, or bad configuration.

Alert Routing and Escalation Chains

Simplicity is recommended and preferred wherever possible when it comes to alert routing. Consider defining two escalation chains for each department: one for non-repeating emailed warning alerts and one for errors and critical alerts sent to pagers and email, escalating among staff.

Next, define rules that match on group or hostname pattern and severity:

- Warning rules that route warnings to email
- All rule that catches everything other than warning (error and critical) and routes to pagers

Repeat this model for each department, routing alerts to the correct destinations.

If you have non-critical or test systems within your monitoring scope, consider creating rules to route some alerts to escalation chains without destinations.

BEST PRACTICES

Handling Alerts

Once alerts are set up properly, the next thing to consider is the best strategy for handling alerts. This ensures that your team is on the same page and that alerts are used to address issues in a timely manner.

Respond

Though it seems elementary, a response is the first action to take. Respond by either acknowledging the alert, for example, replying, "ACK I am investigating," or reply that you are scheduling downtime to handle the situation, for example, by replying, "SDT 1," to indicate you are scheduling an hour of downtime for the alert. Either response will stop the escalation and notify any previously alerted contacts.

If you are indisposed and unable to deal with the alert, you can immediately escalate to the next stage by replying, "Next."

Acknowledge (ACK) or Schedule Downtime

If you believe you can resolve the condition, acknowledge the alert. Resolving the alert should include fixing the underlying cause and then tuning the alert threshold if necessary, disabling the alert, or clearing the alert. Acknowledging affects the current alert occurrence and is only in effect until the condition clears. If the alert is triggered again, another alert is sent. SDT suppresses all alert escalation for current and future events until the SDT expires.

Consider the following conditions that would warrant scheduling downtime instead of immediate attention:

- Alert triggered on a non-urgent issue late at night and you will fix the cause and tune the threshold in the AM.
- Alert will continue until you are able to address the underlying cause, such as a server needing more memory that is currently on order.

Scheduled downtime suppresses the alert until the scheduled downtime expires. Should your scheduled downtime expire before the condition is remedied, you are notified again of the continuing issue.

Another distinction to keep in mind between acknowledging and scheduling downtime is that SDT applies to all alert levels. If an alert triggers at the error level and you place the instance in SDT, the instance will not trigger another alert even if the state becomes critical. Acknowledging an alert does not affect the escalation of alerts of higher severity.

Monitor System Sprawl

You have responded. You have decided whether to acknowledge or to set scheduled downtime for the alert. Now, you need to determine whether the alert was triggered appropriately.

Does the alert fit the following criteria?

- Triggered by a real issue
- Delivered to the appropriate people
- Delivered through the correct mechanism

If so, then you can concentrate on resolving the issue and feel secure that the alert and escalation are correctly configured. For example, if the alert was on a production server triggered by a high swap rate and it was sent to the correct team members' pagers, the alerting is correct. You can concentrate on curing the memory issue on the server by restarting the process, adding memory, or transferring or retiming workloads.

Was it a real issue that was incorrectly escalated? If the wrong team members were notified or the notification mechanism was incorrect, take the steps to adjust alert escalations. Either associate the alert with the correct escalation chain and alert rules or adjust the alert rule by changing the thresholds associating severity with notification methods.

Alerts triggered during a time when the instance, host, or group was under maintenance show a need for policy change. Ensure SDT is appropriately set and policies created to the consistent use of SDT, whether recurring or one-off instances.

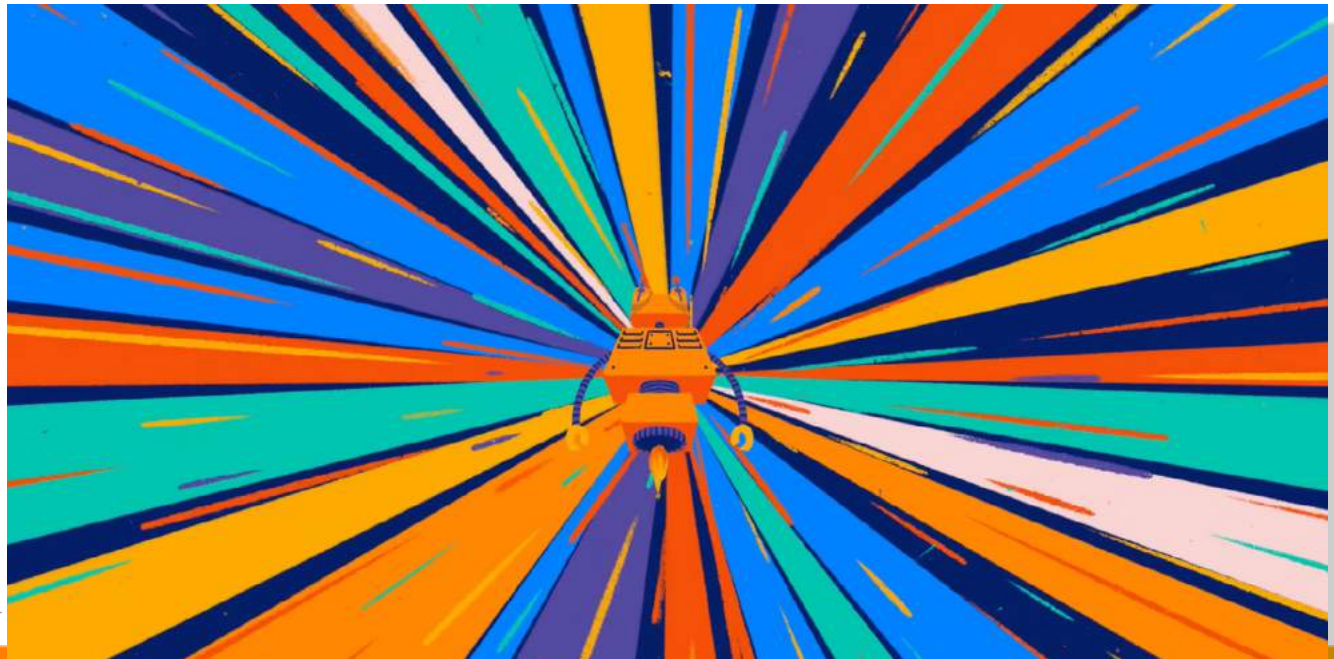
If the alert was not appropriate and noise, consider the following ways to prevent it in the future:

- **Adjust the thresholds** globally, group-wide, at the host, or at the instance level. For example, you can change the threshold for spare disk alerting to 1 for small NetApps. You can also use 5 time-based thresholds to apply different thresholds during different periods of time to prevent alerts. For example, consider adjusting the CPU load on NetApps during weekly disk scrubs.
- **Disable all alerts** for a group, host, or specific instance. For example, you might want to completely disable alerts for a group of development machines.

Add Alert Forecasting

If you want to take your alert strategy to the next level, add alert forecasting to the mix. Standalone alerts can notify you of specific changes in metrics, but alert forecasting gives you the ability to predict how that metric will behave in the future. Forecasting also enables you to predict metric trends and know when an alert will reach a designated threshold.

With this information, you can prioritize the urgency of performance issues, refine budget projections, and strategize resource allocation. Some useful examples of alert forecasting are predicting when a disk will be at 95% capacity and predicting your total monthly AWS bill. Ideally, your monitoring platform would enable you to predict alerts for several months in the future.



BEST PRACTICES

Avoiding Alert & Email Overload

Alert Storms

Too many alerts going off too frequently create alert fatigue or even what is sometimes referred to as an Alert Storm. This is extremely dangerous and can have many adverse consequences, such as a failure of system monitoring.

An overload of alerts can cause people to tune out all alerts, including those that are service-impacting. It is the classic case of the boy who cried wolf. Too many alerts can result in actual critical issues going unnoticed and can also lead to critical alerts being accidentally disabled to cut down the noise. Don't let a critical production server be put into SDT for eight hours because the admin assumed it was another false alert. Every alert must be meaningful and have an associated action. Instead, follow these best practices to prevent alert overload:

Best Practices to Prevent Alert Overload

- Adopt sensible escalation policies, distinguishing between warnings/errors or critical alerts. There is no need to wake people if NTP is out of sync, but if the primary database volume 6 is seeing 200ms latency and transaction time is 18 seconds for an end-user, that is critical and needs to be handled.
- Route the right alerts to the right people. Don't alert the DBA about network issues and don't tell the networking group about a hung transaction.
- Tune your thresholds. Every alert must be real and meaningful. Get rid of false positives or alerts triggered on test systems.
- Investigate alerts triggered when everything seems okay. If you find there was no outward issue, adjust thresholds or disable the alert.
- Ensure alerts are acknowledged, resolved, and cleared. Hundreds of unacknowledged alerts are too difficult to allow easy parsing of an immediate issue. Use alert filtering to view only the groups of systems for which you are responsible.
- Sort alerts by duration periodically and focus on clearing those uncleared for more than a day.
- Create weekly alert reports that cover the week and deliver them to your department. Meet to review top alerts by host or by alert type. Use the alert to investigate monitoring, system, or operational processes. Work to reduce the frequency of alerts.
- Consider a weekly alert summary report if a full weekly alert report is too large. List the hosts with the most alerts over the last week and resolve resource issues and tune thresholds to get to the point that weekly alert reports are manageable.
- Use trend reports to track the hosts and groups with the most alerts.

Correctly routing alerts is also important. If the action item is to call someone else, then the alert is not meaningful. Don't turn your admins into receptionists! This is particularly important when working in a DevOps culture where you have developers on-call and different teams responsible for various systems.

Take into account that certain metrics will indicate a problem at the application layer, and developers are best placed to resolve the situation. Still, problems that relate to the database should be handled by the DBA. While it makes sense for your operations people to be the first line for system failures, if their only action item is to call a developer or a DBA, this can quickly become demoralizing. Similarly, if a developer is woken up for what is a system failure, this can also be a problem. There will always be situations where one team has to escalate to another, but these should be the exceptions rather than the norm.

You can avoid alert-spamming by setting up custom escalation chains and alert rules that route alerts to the right people when an issue occurs, if available as a feature in your monitoring platform.

Email Overload

While most organizations go to great lengths to prevent false alerts that could potentially page people and wake them up at night, email alerts are often another story. Some companies will allow thousands of email alerts to go off for warning-level situations that are often self-corrected. At the same time, other warnings may be more useful, such as disk usage warnings that can be used to prevent an alert from reaching a critical level and waking someone during the night. Unfortunately, it is not realistic to manually scan a thousand emails for something that may be useful. Effective monitoring tools provide dashboards to indicate alert status across the entire infrastructure that can be monitored to anticipate issues before they occur.

Utilize Reporting Functionality

Effective reporting can help avoid email overload. One best practice is to generate a nightly report for all warning-level alerts and disable alert delivery. Admins can then check the report each morning and address any warnings to prevent them from escalating to error or critical levels.

Another helpful report is a report of all the alert thresholds. This is useful for checking the currently set thresholds, but most importantly, it will show any disabled thresholds. One of the common responses to excess noise in alerting is disabling alerting for a metric or even an entire host. This is often done to address the issue at a later time, but frequently, higher priorities can intervene, meaning that it will be forgotten until there is a related outage.

BEST PRACTICES

Learning from Missed Alerts

Even the best monitoring systems will occasionally result in system failures that go undetected. A missed issue that comes to light without the aid of monitoring must not be considered fixed until the monitoring solution can detect any recurrence of the issue. Outages occur even with proper monitoring.

Outages are bad at any time but can be particularly damaging after replacing one monitoring system with another. No one remembers the thousands of false alerts produced by your home-rolled Nagios when your shiny new SaaS system misses a production down incident.

To counter the skepticism this will inevitably introduce to your new system, it is essential to have "blameless" post-mortem practices in place for every outage. A useful checklist of questions to ask in such post-mortem include:

- What was the root cause of the failure?
- What state were the systems in prior to the failure?
- How much CPU was consumed and how much memory?
- Were we swapping?
- Were there alerts that should have triggered?
- Were the thresholds correct?

Best practices dictate that an issue not be considered resolved until monitoring is in place to detect the root cause and provide early warning, if at all possible.

The fact that things are working again does not mean the issue is closed. Only consider the issue resolved after you are content with the warning you received and the escalations that occured, or after you have adjusted monitoring to give more warning in the future. It is possible that the issue could not be detected. Sudden catastrophic failure is possible. But, evaluation of the process must be executed for each service-impacting event.

Data graphing should assist with this. Compare the graphs under normal system usage with those just before the system failure. Ideally, your monitoring platform will allow you to build dashboards that display relevant graphs and metrics across your entire infrastructure. After discussing all of the above, the next set of questions you need to answer relates to what can be done to prevent this in the future. The issue must not be marked as resolved until monitoring is in place to avoid a repeat.

For example, if a Java application experiences a service-affecting outage due to numerous users overloading the system, track the busy threads using JMX monitoring. Create an alert threshold on this metric to provide early warning before the next event. When the alert is triggered, you should have enough time to add another system to share the load or activate a means to shed load.

BEST PRACTICES

Managing Downtime

When you plan to work on a system, schedule downtime to safeguard against alert escalation during the work period.

Ideally, your monitoring platform will allow you to use scheduled downtime (SDT) on any of the following:

- **Individual instance** - for example, a single drive
- **Host data source** - for example, all the drives on the host
- **Host** - for example, all monitoring on a host
- **Group** - for example, all alerts associated with all of the hosts in a group or a data source associated with the hosts in the group

Your chosen monitoring solution should have fine-grained control for setting downtime. This can range from disabling all alerts during a full downtime deployment to controlling individual hosts and monitored instances for ongoing maintenance. When setting SDT, use the narrowest scope possible (instance, host, or group). Doing so will help limit the possibility of suppressing valid alerts during the SDT interval.

For example, a fine-grained approach to downtime will allow you to set downtime for a single hard drive on a given host. If you are inadvertently alerted for a host that is not currently in production, you may decide to disable all metrics but still want to ensure that the host is up and responding to network pings. Additionally, disabling all monitoring for a host may be desirable. For example, if a host has been removed from a production cluster but not fully decommissioned, you could decide you may not need to monitor it.

Host groups are useful for managing groups of related systems. For example, the categories dev, QA, and production group give an excellent high-level organization for managing downtime because, during QA deployments, you may decide to disable all hosts in the QA group. For automated deployments, a monitoring system will ideally provide an API to allow automation tools to specify downtime. For scheduled maintenance, the monitoring solution should allow you to schedule repeatable maintenance windows in advance.

Downtime should be scheduled at the most granular level possible. If you are increasing the size of a disk, for example, don't set the entire host to downtime. If you know it will take you 24 hours to fix the issue, schedule 24 hours of downtime. Downtime should expire automatically to avoid forgotten settings.

LOOKING FORWARD

AIOps & The Future of Alerting

Monitoring, analyzing, and fixing most issues automatically through the use of AIOps describes the future state of IT operations – a self-healing IT infrastructure. AIOps enables the monitoring system to monitor, analyze, and repair most of the issues on its own and only surface less than 20% of issues to the user. The issues that do get raised are only those for which remediation is not yet automated.

Self-healing maturity begins with automating simple infrastructure tasks based on rules to fix problems within the infrastructure proactively. This signals a significant move away from a reliance on alerts. These rules are derived from **one or more of the following sources:**

- **Organizational knowledge base**. This includes heuristic rules that operators can create using the metrics, events, and transactions collected by the monitoring system. These rules are typically documented as runbooks and are triggered automatically when a specific condition occurs in the infrastructure and is detected in the data.
- **Dynamically derived scopes**. Dynamically derived scopes are learned patterns within the collected data that depict normal versus abnormal conditions at any point in time across the entire infrastructure stack. These conditions trigger automatic actions that are configured using runbooks or IT policies.

- **Intelligently learned patterns**. Learned from historical patterns across multi-tenant environments, the self-healing system performs actions based on certain conditions happening within the infrastructure. The self-healing system uses sophisticated AI techniques like "reinforcement learning" to learn automation from historical actions performed across datasets from multiple customers.

Sophisticated monitoring platforms utilize AIOps functionality that enables you to:

- Intelligently reduce alert noise by preventing alert storms and invalid alerts from occurring.
- Make remaining alerts more meaningful and actionable by providing correlations and metadata to identify the root cause of issues more quickly and provide more intelligent context around events and alerts.

Monitoring, analyzing, and fixing most issues automatically through the use of AIOps describes the future state of IT operations – a self-healing IT infrastructure. AIOps enables the monitoring system to monitor, analyze, and repair most of the issues on its own.

The complex nature of web and SaaS platforms today means that monitoring cannot be seen as something that only applies to portions of your infrastructure. Modern monitoring solutions can provide a robust infrastructure monitoring framework that can do both – alert when necessary and help to avoid alert overload. It is important to realize, however, that even the best tools require careful implementation and ongoing tuning and improvement. As new features are designed, careful thought should be given to how they will be monitored.

- What metrics indicate the normal functioning of the application or the overall system?
- What are the acceptable thresholds for a given metric?
- At what point should we be warned and what do we do to prevent the warning escalating to a critical situation?

Ongoing optimization is not just the responsibility of the operations team maintaining the monitoring systems. It requires cross-team collaboration to ensure you are monitoring the right things and alerting the right people. This is the key to a successful monitoring strategy.

About LogicMonitor®

Monitoring unlocks new pathways to growth. At LogicMonitor®, we expand what's possible for businesses by advancing the technology behind them. LogicMonitor seamlessly monitors infrastructures, empowering companies to focus less on problem solving and more on evolution. We help customers turn on a complete view in minutes, turn the dial from optimization to innovation and turn the corner from sight to vision. Join us in shaping the information revolution by visiting LogicMonitor.com.