WHITEPAPER

Best Practices for Alert Management

The following white paper provides best practices on preventing erroneous alerts, alert handling and routing, and ensuring you are alerted when a service impacting event occurs.

Two Common Alert System Failures

Alert systems fail because they generate so many meaningless alerts as to create an overload situation with staff, ultimately leading to people ignoring important alerts. Or, they fail to alert you to real issues in your infrastructure due to a lack of flexibility, monitoring capability, or system configuration. Often, both of these situations occur within the same team and the same system.

Your alerting needs to sufficiently catch all conditions needing attention and only alert on conditions that need attention, focus must be on awareness and removing alert noise. Achieving these goals requires tuning the system to do away with non-meaningful alerts, and never closing an incident until you create alerts for previously undetected events.

The following steps help you achieve this balanced state with LogicMonitor.

Best Practices for Alert Management

Preventing Alerts During Planned Downtime

When you plan to work on a system, schedule downtime to safeguard against alert escalation during the work period. Within LogicMonitor, you can use scheduled downtime (SDT) on any of the following:

• Individual instance - for example, a single drive

• Host datasource - 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 datasource associated with the hosts in the group

Take advantage of scheduling within LogicMonitor, use it to set recurring SDTs for daily,

weekly, or monthly intervals.

When setting an 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.

Routing and Escalating Alert Notification

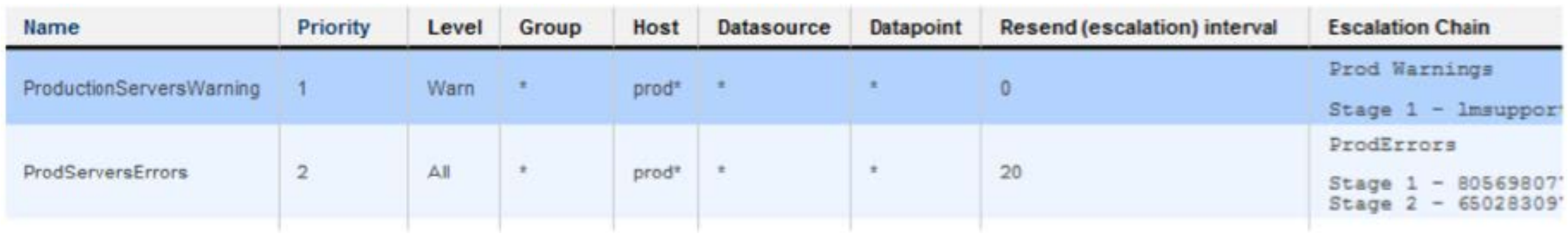
LogicMonitor provides any amount of flexibility you need, but simplicity is recommended and preferred wherever possible. Consider defining two escalation chains for each department: one for non-repeating emailed warning alerts and one for errors and critical alerts sent to pagers and email, escalating among staff.

"LogicMonitor provides any amount of flexibility you need, but simplicity is recommended and preferred wherever possible."

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



Name	Priority	Level	Group	Host	Datasource	Datapoint	Resend (escalation) interval	Escalation Chain
ProductionServersWarning	1	Warn	*	prod*	*	*	0	Prod Warnings Stage 1 - lmsuppor
ProdServersErrors	2	All	*	prod*	*	*	20	ProdErrors Stage 1 - 80569807 Stage 2 - 65028309

Repeat this model for each department, routing alerts to the correct destinations. Keep in mind that these rules are 'first-match' rules. That is, the alerting logic works from top to bottom and the first rule matched by an event is the one executed. They do not continue to cascade after a match.

If you have non-critical or test systems within your monitoring scope, consider creating rules to route some alerts to escalation chains without destinations. These alerts will only be visible in the web console.

Handling Alerts

Consider the following recommendations in reference to incoming alerts.

"Consider the following recommendations in reference to incoming alerts:

• First Respond,

• Then Acknowledge (ACK) or Schedule Downtime,

• Lastly Analyze."

Respond

Though it seems elementary, 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, replying "SDT 1" to indicate you are scheduling an hour of downtime for the alert. Either response will stop 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 are able to 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 in the middle of the night and you will fix the cause and tune the threshold in the morning.
- Alert will continue until you are able to address an underlying cause, for example, 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 to 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.

Monitoring System Sprawl

You have responded. You have decided whether to acknowledge or to set scheduled downtime for the alert. Now, you need to decide if 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 people's pagers, the alerting is correct. You can concentrate on curing the memory issue on the server: restarting the process, adding memory, or transferring or retiming workloads.

Was it a real issue that was incorrectly escalated? If the wrong people were notified or the notification mechanism was incorrect, take the next 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 point to a need for policy alteration. 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.

• Disable a datasource globally, group-wide, or on a specific host or instance. Again, you might specifically disable the collection of Apache data on development machines.

• Eliminate the alerting datasource instance from discovery or create a cloned datasource to discover and classify it with a different set of alert thresholds. For example, if you have a set of instances that should not be discovered, like NFS mounted filesystems on linux hosts, that you want to set different thresholds on than other instances, like test VIPs on load balancers, you can filter your Active Discovery.

Not Alerted

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 good monitoring. 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. Use graphs to investigate behaviors that can be used to alert about the event. For example, if a Java application experiences a service-affecting outage due to a 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 load or activate a means to shed load.

The need for analysis of outages for which you were not alerted is of utmost importance. Because things are working again does not mean the issue is closed. Only after you are content with the warning you received, the escalations that occurred, or have adjusted monitoring to give more warning should you consider the issue resolved. 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.

Avoiding Alert Overload

Too many alerts going off too frequently creates overload. People tune out all of them. When you receive real service-impacting alerts, they go unnoticed, also. It is the classic case of the boy who cried wolf. Don't let a critical production server get put into SDT for 8 hours because the admin assumed it was another false alert:

• Adopt sensible escalation policies, distinguishing between warnings and error or critical alert levels. 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. You need to be on it, no matter the time.

• 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. Tune the monitoring to 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. You can also reach out to LogicMonitor to ask about ramifications of

disabling an alert.

• Ensure alerts are acknowledged, resolved, and cleared. Hundreds of unacknowledged alerts are too difficult to allow easy parsing of an immediate issue. User 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 hos 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.

Never hesitate to contact LogicMonitor support if you have any questions.