LOGZILLA DOCUMENTATION

Application Performance Monitoring

Correlate response-time degradation, error clustering, and service dependency failures in LogZilla using SEC threshold and trend analysis rules

Event Correlation · Generated June 12, 2026 · logzilla.ai/docs/event-correlation/application-performance-monitoring

Application Performance Monitoring Correlation

Application performance correlation detects degradation patterns, error clustering, and service dependency issues that span multiple events over time. LogZilla's pre-processing extracts performance metrics, while SEC handles complex threshold analysis and trend correlation.

Prerequisites: Ensure Event Correlation is enabled and forwarder reloading is available as shown in the Event Correlation Overview (https://www.logzilla.ai/docs/event-correlation/event-correlation-overview).

Response Time Degradation Detection

LogZilla Forwarder Configuration

```
# /etc/logzilla/forwarder.d/linux-performance.yaml
window_size: 0
type: sec
sec_name: linux-performance
match:
  - field: extra:_is_performance
    value:
      - "true"
rules:
- rewrite:
    message: >-
      [LINUX_PERFORMANCE]
      program="${:program}"
      host="${:host}"
      srcip="${:SrcIP}"
      username="${:Username}"
      service="${:service}"
      endpoint="${:endpoint}"
      status="${:HTTP Status Code}"
      response_time_ms="${:response_time_ms}"
      response_time_sec="${:response_time_sec}"
      performance="${:performance}"
      error="${:error}"
      severity="${:severity}"
      memory_usage="${:memory_usage}"
      memory_usage_pct="${:memory_usage_pct}"
      process_count="${:process_count}"
      $MESSAGE
```

SEC Rule: Response Time Degradation with Recovery

```
# /etc/logzilla/sec/linux-performance/response-time-degradation.sec
# ============================================
# Response Time Degradation Detection (Linux)
# Detects repeated slow responses for monitored services/endpoints
# Uses linux-performance message rewrite output
# ============================================

type=SingleWithThreshold
ptype=RegExp
pattern=\[LINUX_PERFORMANCE\].*host="([^\"]+)".*service="([^\"]+)".*endpoint="
([^\"]*)".*response_time_ms="([^\"]*)".*performance="(WARNING|CRITICAL)".*severity="([^\"]*)"
desc=Response time degradation detected for $1 on $3
thresh=5
window=300
action=eval %host ( "$1" ); \
       eval %service ( "$2" ); \
       eval %endpoint ( "$3" ); \
       eval %response_time_ms ( "$4" ); \
       eval %performance ( "$5" ); \
       eval %severity ( "$6" ); \
       create PERF_DEGRADATION_ACTIVE_%host_%endpoint 900; \
       write /var/log/logzilla/sec/performance.log "RESPONSE_TIME_DEGRADATION %t host=%host
service=%service endpoint=%endpoint severity=%severity response_time_ms=%response_time_ms
occurrences=%thresh window=300s"; \
       shellcmd (logger -t SEC-PERFORMANCE -p local0.warning "RESPONSE_TIME_DEGRADATION
host=\"%host\" service=\"%service\" endpoint=\"%endpoint\" severity=\"%severity\"
response_time_ms=\"%response_time_ms\" occurrences=\"%thresh\" window=\"300s\"")
```

LogZilla Trigger: Performance Degradation Response

```
name: "Application Performance Degradation"
filter:
  - field: program
    op: eq
    value: SEC-PERFORMANCE
  - field: message
    op: "=~"
    value: "RESPONSE_TIME_DEGRADATION"
actions:
  exec_script: true
  script_path: "/usr/local/bin/performance-degradation-response.sh"
  send_webhook: true
  send_webhook_template: |
    {
      "alert_type": "performance_degradation",
      "service": "{{event:ut:service}}",
      "endpoint": "{{event:ut:endpoint}}",
      "host": "{{event:ut:host}}",
```

```
      "severity": "{{event:ut:performance}}"
    }
```

Intelligent Performance Response Script

```
#!/bin/bash
# /usr/local/bin/performance-degradation-response.sh

SERVICE="$EVENT_USER_TAGS_SERVICE"
ENDPOINT="$EVENT_USER_TAGS_ENDPOINT"
HOST="$EVENT_USER_TAGS_HOST"
AVG_RESPONSE="$EVENT_USER_TAGS_AVG_RESPONSE"

# Query application metrics for additional context
CPU_USAGE=$(curl -s "http://$HOST:9090/metrics" | grep "cpu_usage" | awk '{print $2}')
MEMORY_USAGE=$(curl -s "http://$HOST:9090/metrics" | grep "memory_usage" | awk '{print $2}')
ACTIVE_CONNECTIONS=$(curl -s "http://$HOST:9090/metrics" | grep "active_connections" | awk '{print
$2}')

# Determine root cause and appropriate response
if [[ $(echo "$CPU_USAGE > 80" | bc) -eq 1 ]]; then
    # High CPU - scale horizontally if possible
    logger -t PERF-RESPONSE "High CPU detected on $HOST - attempting auto-scale"
    curl -X POST "https://orchestrator.company.com/api/scale-service" \
         -d "service=$SERVICE&action=scale_out&reason=high_cpu"

elif [[ $(echo "$MEMORY_USAGE > 90" | bc) -eq 1 ]]; then
    # Memory pressure - restart service
    logger -t PERF-RESPONSE "Memory pressure on $HOST - scheduling service restart"
    curl -X POST "https://orchestrator.company.com/api/restart-service" \
         -d "service=$SERVICE&host=$HOST&reason=memory_pressure"

elif [[ $(echo "$ACTIVE_CONNECTIONS > 1000" | bc) -eq 1 ]]; then
    # Connection overload - enable rate limiting
    logger -t PERF-RESPONSE "Connection overload on $HOST - enabling rate limiting"
    curl -X POST "https://loadbalancer.company.com/api/rate-limit" \
         -d "service=$SERVICE&limit=100&duration=300"

else
    # Unknown cause - create investigation ticket
    logger -t PERF-RESPONSE "Unknown performance issue on $HOST - creating ticket"
    curl -X POST "https://servicedesk.company.com/api/tickets" \
         -d "priority=medium&subject=Performance Investigation&service=$SERVICE&host=$HOST"
fi

# Update performance dashboard
curl -X POST "https://dashboard.company.com/api/performance-events" \
     -d "service=$SERVICE&endpoint=$ENDPOINT&host=$HOST&response_time=$AVG_RESPONSE&status=degraded"
```

Error Rate Correlation

HTTP Error Clustering Detection

Correlate HTTP error patterns to detect application issues.

SEC Rule: Error Rate Spike Detection

```
# /etc/logzilla/sec/linux-performance/error-rate-spike-detection.sec
##########################################
# HTTP Error Rate Spike Detection (Linux)
# Detects spikes of HTTP 4xx/5xx responses for monitored services
# Relies on linux-performance rewrite message format
##########################################

type=SingleWithThreshold
ptype=RegExp
pattern=\[LINUX_PERFORMANCE\].*host="([^"]*)".*service="([^"]*)".*endpoint="([^"]*)".*status="
([45]\d{2})"
desc=Spike in HTTP error responses for $1 on $3
thresh=50
window=300
action=eval %host ( "$1" ); \
       eval %service ( "$2" ); \
       eval %endpoint ( "$3" ); \
       eval %status ( "$4" ); \
       create HTTP_ERROR_RATE_ACTIVE_%host_%endpoint 600; \
       write /var/log/logzilla/sec/performance.log "HTTP_ERROR_RATE_SPIKE %t host=%host
service=%service endpoint=%endpoint status_code=%status occurrences=%thresh window=300s"
```

Service Dependency Correlation

Cascading Service Failure Detection

Detect when upstream service failures cause downstream performance issues.

SEC Rule: Service Dependency Correlation

```
# /etc/logzilla/sec/linux-performance/service-dependency-correlation.sec
# ==========================================
# Service Dependency Correlation (Linux)
# Detects upstream service degradation and tracks downstream impact
# Uses linux-performance message rewrite output
# Tracks dependencies: upstream (auth, payment, inventory) -> downstream (web, api, mobile)
```

```
# ============================================

# Rule 1: Detect upstream service degradation
type=Single
ptype=RegExp
pattern=.*\[LINUX_PERFORMANCE\].*host="([^"]*)".*service="(auth|payment|inventory)".*endpoint="
([^"]*)".*response_time_ms="([^"]+)".*performance="(WARNING|CRITICAL)".*severity="([^"]*)"
desc=Upstream service degradation detected for $2
action=eval %host ( "$1" ); \
       eval %service ( "$2" ); \
       eval %endpoint ( "$3" ); \
       eval %response_time_ms ( "$4" ); \
       eval %performance ( "$5" ); \
       eval %severity ( "$6" ); \
       create UPSTREAM_DEGRADED_%service 1800; \
       write /var/log/logzilla/sec/performance.log "UPSTREAM_SERVICE_DEGRADATION %t
upstream_service=%service host=%host endpoint=%endpoint severity=%severity
response_time_ms=%response_time_ms"; \
       shellcmd (logger -t SEC-DEPENDENCY -p local0.warning "UPSTREAM_ISSUE service=\"%service\"
host=\"%host\" endpoint=\"%endpoint\" severity=\"%severity\" response_time_ms=\"%response_time_ms\"")

# Rule 2: Detect downstream impact when auth service is degraded
type=Single
ptype=RegExp
pattern=.*\[LINUX_PERFORMANCE\].*host="([^"]*)".*service="(web|api|mobile)".*endpoint="
([^"]*)".*response_time_ms="([^"]+)".*performance="(WARNING|CRITICAL)".*severity="([^"]*)"
context=UPSTREAM_DEGRADED_auth
desc=Downstream service impact detected from auth degradation on $2
action=eval %host ( "$1" ); \
       eval %service ( "$2" ); \
       eval %endpoint ( "$3" ); \
       eval %response_time_ms ( "$4" ); \
       eval %performance ( "$5" ); \
       eval %severity ( "$6" ); \
       create CASCADING_FAILURE_%service_auth 900; \
       write /var/log/logzilla/sec/performance.log "CASCADING_FAILURE %t downstream=%service
upstream_cause=auth host=%host endpoint=%endpoint severity=%severity
response_time_ms=%response_time_ms"; \
       shellcmd (logger -t SEC-DEPENDENCY -p local0.warning "CASCADING_FAILURE
downstream=\"%service\" upstream_cause=\"auth\" host=\"%host\" endpoint=\"%endpoint\"
severity=\"%severity\" response_time_ms=\"%response_time_ms\"")

# Rule 3: Detect downstream impact when payment service is degraded
type=Single
ptype=RegExp
pattern=.*\[LINUX_PERFORMANCE\].*host="([^"]*)".*service="(web|api|mobile)".*endpoint="
([^"]*)".*response_time_ms="([^"]+)".*performance="(WARNING|CRITICAL)".*severity="([^"]*)"
context=UPSTREAM_DEGRADED_payment
desc=Downstream service impact detected from payment degradation on $2
action=eval %host ( "$1" ); \
       eval %service ( "$2" ); \
       eval %endpoint ( "$3" ); \
       eval %response_time_ms ( "$4" ); \
       eval %performance ( "$5" ); \
```

```
        eval %severity ( "$6" ); \
        create CASCADING_FAILURE_%service_payment 900; \
        write /var/log/logzilla/sec/performance.log "CASCADING_FAILURE %t downstream=%service
upstream_cause=payment host=%host endpoint=%endpoint severity=%severity
response_time_ms=%response_time_ms"; \
        shellcmd (logger -t SEC-DEPENDENCY -p local0.warning "CASCADING_FAILURE
downstream=\"%service\" upstream_cause=\"payment\" host=\"%host\" endpoint=\"%endpoint\"
severity=\"%severity\" response_time_ms=\"%response_time_ms\"")

# Rule 4: Detect downstream impact when inventory service is degraded
type=Single
ptype=RegExp
pattern=.*\[LINUX_PERFORMANCE\].*host="([^"]*)".*service="(web|api|mobile)".*endpoint="
([^"]*)".*response_time_ms="([^"]+)".*performance="(WARNING|CRITICAL)".*severity="([^"]*)"
context=UPSTREAM_DEGRADED_inventory
desc=Downstream service impact detected from inventory degradation on $2
action=eval %host ( "$1" ); \
        eval %service ( "$2" ); \
        eval %endpoint ( "$3" ); \
        eval %response_time_ms ( "$4" ); \
        eval %performance ( "$5" ); \
        eval %severity ( "$6" ); \
        create CASCADING_FAILURE_%service_inventory 900; \
        write /var/log/logzilla/sec/performance.log "CASCADING_FAILURE %t downstream=%service
upstream_cause=inventory host=%host endpoint=%endpoint severity=%severity
response_time_ms=%response_time_ms"; \
        shellcmd (logger -t SEC-DEPENDENCY -p local0.warning "CASCADING_FAILURE
downstream=\"%service\" upstream_cause=\"inventory\" host=\"%host\" endpoint=\"%endpoint\"
severity=\"%severity\" response_time_ms=\"%response_time_ms\"")
```

Application Resource Correlation

Memory Leak Detection

Correlate memory usage patterns with application performance over time.

SEC Rule: Memory Leak Detection

```
# /etc/logzilla/sec/linux-performance/memory-leak-detection.sec
# Memory Leak Detection SEC Rule for LogZilla
#
# Prerequisites:
# 1. Deploy memory-usage-logging.sh to /usr/local/bin/ on monitored hosts
# 2. Deploy check-memory-trend.sh to /usr/local/bin/ on LogZilla server
# 3. Make both scripts executable: chmod +x /usr/local/bin/*.sh
# 4. Add memory-usage-logging.sh to crontab on monitored hosts:
#    */5 * * * * /usr/local/bin/memory-usage-logging.sh
# 5. Ensure LogZilla rewrite rule (linux-performance.yaml) forwards a single-line message
```

```
#    that contains program="MEMORY_USAGE" and fields host, service, memory_usage, memory_usage_pct
#
# This rule analyzes memory usage trends over time to detect memory leaks.
# It requires at least 10 samples (50 minutes) to establish a trend.
#
# Exit code from check-memory-trend.sh:
#   0 = Memory leak detected (triggers action)
#   1 = Normal memory usage (triggers action2)
# Track memory usage trends over time per service
# Match the [LINUX_PERFORMANCE] header for host, then the original message at the end for actual data
# Pass captured values to the trend checker: host, service, memory_usage

type=SingleWithScript
ptype=RegExp
pattern=\[LINUX_PERFORMANCE\].*program="MEMORY_USAGE".*host="([^"]+)".*service=(\S+)\s+memory_usage=
([\d.]+)\s+memory_usage_pct=([\d.]+)\s+process_count=(\d+)
script=/usr/local/bin/check-memory-trend.sh %1 %2 %3
desc=Memory leak detected: %2 on %1
action=eval %host ( "$1" ); \
       eval %service ( "$2" ); \
       eval %memory_usage ( "$3" ); \
       eval %memory_usage_pct ( "$4" ); \
       eval %process_count ( "$5" ); \
       shellcmd (logger -t SEC-PERFORMANCE -p local0.alert "MEMORY_LEAK_DETECTED host=%host
service=%service memory_usage_mb=%memory_usage memory_usage_pct=%memory_usage_pct
process_count=%process_count trend=increasing")
action2=logonly  # Normal memory usage (no leak detected)
```

Memory Trend Analysis Script

```
#!/bin/bash
# /usr/local/bin/check-memory-trend.sh
# Memory Trend Analysis Script for SEC
# Analyzes memory usage trends to detect memory leaks
#
# Usage: check-memory-trend.sh <host> <service> <current_memory_mb>
# Exit 0 = Memory leak detected (triggers SEC action)
# Exit 1 = Normal memory usage (triggers SEC action2)

HOST="$1"
SERVICE="$2"
CURRENT_MEMORY="$3"

# Validate inputs
if [ -z "$HOST" ] || [ -z "$SERVICE" ] || [ -z "$CURRENT_MEMORY" ]; then
    echo "Usage: $0 <host> <service> <current_memory_mb>" >&2
    exit 1
fi

# Create history directory if it doesn't exist
HISTORY_DIR="/var/tmp/lz_memory_history"
```

```
mkdir -p "$HISTORY_DIR" 2>/dev/null

# Track memory history per service per host
MEMORY_HISTORY="${HISTORY_DIR}/memory_${HOST}_${SERVICE}"
echo "$(date +%s) $CURRENT_MEMORY" >> "$MEMORY_HISTORY"

# Keep only last hour of data (3600 seconds)
CUTOFF=$(($(date +%s) - 3600))
if [ -f "$MEMORY_HISTORY" ]; then
    awk -v cutoff="$CUTOFF" '$1 >= cutoff' "$MEMORY_HISTORY" > "${MEMORY_HISTORY}.tmp"
    mv "${MEMORY_HISTORY}.tmp" "$MEMORY_HISTORY"
fi

# Count number of samples
MEMORY_SAMPLES=$(wc -l < "$MEMORY_HISTORY" 2>/dev/null || echo "0")

# Need at least 10 samples (50 minutes of data at 5-minute intervals) for trend analysis
if [ "$MEMORY_SAMPLES" -lt 10 ]; then
    # Not enough data yet, consider normal
    exit 1
fi

# Calculate linear regression slope to detect memory growth trend
# Slope = (n * sum(xy) - sum(x) * sum(y)) / (n * sum(x^2) - sum(x)^2)
SLOPE=$(awk '{
    sum_x += NR
    sum_y += $2
    sum_xy += NR * $2
    sum_x2 += NR * NR
    n++
} END {
    if (n * sum_x2 - sum_x * sum_x == 0) {
        print 0
    } else {
        slope = (n * sum_xy - sum_x * sum_y) / (n * sum_x2 - sum_x * sum_x)
        print slope
    }
}' "$MEMORY_HISTORY")

# Memory leak thresholds (configurable)
# LEAK_THRESHOLD_MB_PER_SAMPLE: MB growth per sample interval
# At 5-minute intervals, 0.5 MB/sample = 6 MB/hour growth
LEAK_THRESHOLD_MB_PER_SAMPLE=0.5

# Check if slope exceeds leak threshold
# Use bc for floating point comparison
if command -v bc >/dev/null 2>&1; then
    IS_LEAK=$(echo "$SLOPE > $LEAK_THRESHOLD_MB_PER_SAMPLE" | bc -l)
    if [ "$IS_LEAK" -eq 1 ]; then
        # Memory leak detected
        echo "Memory leak detected: service=$SERVICE host=$HOST slope=${SLOPE}MB/sample" >&2
        exit 0
    fi
else
```

```
    # Fallback to integer comparison if bc not available
    SLOPE_INT=$(printf "%.0f" "$SLOPE")
    if [ "$SLOPE_INT" -ge 1 ]; then
        echo "Memory leak detected: service=$SERVICE host=$HOST slope=${SLOPE}MB/sample" >&2
        exit 0
    fi
fi

# Normal memory usage
exit 1
```

Generic Application Error Correlation

Note: This example uses message pattern matching since no database parsing apps exist in LogZilla by default. To parse specific database fields, you would need to create custom parsing rules.

Forwarder Configuration

```
# /etc/logzilla/forwarder.d/app-error-correlation.yaml
type: sec
sec_name: app-errors
rules:
  - match:
      - field: program
        op: "eq"
        value: ["webapp", "api-server", "microservice"]
      - field: message
        op: "=~"
        value: "(ERROR|FATAL|Exception|timeout|slow.*query)"
    rewrite:
      message: "[APP_ERROR] host=$HOST service=$PROGRAM severity=${:severity}"
```

SEC Rule: Application Error Correlation

```
# /etc/logzilla/sec/app-errors/app-error-correlation.sec
############################################
# Application Error Correlation
# Detects error spikes across application services
# Uses app-error-correlation rewrite message format
############################################

type=SingleWithThreshold
ptype=RegExp
pattern=\[APP_ERROR\] host=([^\s]+) service=([^\s]+) severity=([^\s]+)
desc=Application error spike detected on $1
thresh=10
window=300
```

```
action=eval %host ( "$1" ); \
       eval %service ( "$2" ); \
       eval %severity ( "$3" ); \
       create APP_ERROR_SPIKE_%host_%service 600; \
       write /var/log/logzilla/sec/performance.log "ERROR_SPIKE_DETECTED %t host=%host
service=%service severity=%severity occurrences=%thresh window=300s"; \
       shellcmd (logger -t SEC-ALERT -p local0.alert "ERROR_SPIKE_DETECTED host=\"%host\"
service=\"%service\" severity=\"%severity\" occurrences=\"10\" window=\"300s\""); \
       shellcmd (/usr/local/bin/app-error-response.sh "%host" "%service")
```

Related Topics

- Network Infrastructure Correlation (https://www.logzilla.ai/docs/event-correlation/network-infrastructure-correlation)
- Advanced Event Correlation Walkthrough (https://www.logzilla.ai/docs/event-correlation/advanced-event-correlation-walkthrough)
- Event Correlation Rule Types (https://www.logzilla.ai/docs/event-correlation/event-correlation-rule-types)