LOGZILLA DOCUMENTATION

Query Examples

Worked examples of the logzilla query command for security event reports, authentication failure summaries, scheduled automation, and external integration

Command Line Tools · Generated April 29, 2026 · logzilla.ai/docs/command-line-tools/query-examples

Query Examples and Automation

Practical examples, automation scripts, and real-world use cases for the LogZilla query command. These examples demonstrate how to build reports, automate analysis, and integrate LogZilla data with external systems.

Report Generation Examples

Security Event Summary Report

Generate a security report showing authentication failures, privilege escalations, and suspicious activities.

Parameters (security-report.json):

```
{
  "field": "host",
  "limit": 20,
  "time_range": {
    "preset": "last_7_days"
  },
  "filter": [
    {
      "field": "message",
      "op": "qp",
      "value": ["\"authentication failed\" OR \"privilege escalation\" OR suspicious"]
    },
    {
      "field": "severity",
      "op": "le",
      "value": [4]
    }
  ],
  "subfields": ["program", "severity"],
  "subfields_limit": 10
}
```

Automation Script (security-report.sh):

```
#!/bin/bash

# Security Report Generation Script
REPORT_DATE=$(date +%Y%m%d)
OUTPUT_FILE="/reports/security-report-${REPORT_DATE}.xlsx"
API_TOKEN="${LOGZILLA_API_KEY}"

# Generate security report
```

```
logzilla query --type TopN \
    --params security-report.json \
    --output-file "$OUTPUT_FILE" \
    --format xlsx \
    --authtoken "$API_TOKEN"

# Email report to security team
if [ -f "$OUTPUT_FILE" ]; then
    echo "Security report generated: $OUTPUT_FILE" | \
    mail -s "Daily Security Report - $REPORT_DATE" \
         -a "$OUTPUT_FILE" \
         security-team@company.com
fi
```

Network Traffic Analysis

Analyze network device logs to identify top talkers and traffic patterns.

Parameters (network-analysis.json):

```
{
  "field": "host",
  "limit": 15,
  "time_range": {
    "preset": "last_24_hours"
  },
  "filter": [
    {
      "field": "program",
      "op": "=*",
      "value": ["*firewall*", "*router*", "*switch*"]
    }
  ],
  "with_subperiods": true,
  "subfields": ["program"],
  "subfields_limit": 5
}
```

Application Performance Report

Monitor application performance by analyzing error rates and response times.

Parameters (app-performance.json):

```
{
  "time_range": {
    "preset": "last_24_hours",
```

```
    "step": 3600
  },
  "filter": [
    {
      "field": "program",
      "op": "eq",
      "value": ["nginx", "apache2", "tomcat"]
    },
    {
      "field": "severity",
      "op": "le",
      "value": [4]
    }
  ]
}
```

Complete Script (app-performance.sh):

```
#!/bin/bash

API_TOKEN="${LOGZILLA_API_KEY}"
REPORT_DIR="/reports/$(date +%Y/%m)"
mkdir -p "$REPORT_DIR"

# Generate event rate report
logzilla query --type EventRate \
    --params app-performance.json \
    --output-file "${REPORT_DIR}/app-errors-$(date +%Y%m%d-%H%M).json" \
    --authtoken "$API_TOKEN"

# Generate top applications report
echo '{
  "field": "program",
  "limit": 10,
  "time_range": {"preset": "last_24_hours"},
  "filter": [{"field": "severity", "op": "le", "value": [4]}]
}' > /tmp/top-apps.json

logzilla query --type TopN \
    --params /tmp/top-apps.json \
    --output-file "${REPORT_DIR}/top-error-apps-$(date +%Y%m%d-%H%M).xlsx" \
    --format xlsx \
    --authtoken "$API_TOKEN"

rm /tmp/top-apps.json
```

Data Analysis Workflows

Trend Analysis Over Time

Analyze event patterns over extended periods to identify trends.

Weekly Trend Analysis (weekly-trends.sh):

```
#!/bin/bash

API_TOKEN="${LOGZILLA_API_KEY}"
WEEKS=12  # Analyze last 12 weeks

# Ensure output dir exists
mkdir -p /reports/trends

for i in $(seq 0 $((WEEKS-1))); do
    START_DATE=$(date -d "-$((i+1)) weeks" +%Y-%m-%d)
    END_DATE=$(date -d "-${i} weeks" +%Y-%m-%d)

    START_TS=$(date -d "${START_DATE} 00:00:00" +%s)
    END_TS=$(date -d "${END_DATE} 23:59:59" +%s)

    # Create parameters for this week (note: integers, not strings)
    cat > "/tmp/week-${i}.json" << EOF
{
  "time_range": {
    "ts_from": ${START_TS},
    "ts_to": ${END_TS},
    "step": 86400
  },
  "with_archive": true
}
EOF

    # Generate weekly report
    logzilla query --type EventRate \
        --params "/tmp/week-${i}.json" \
        --output-file "/reports/trends/week-${START_DATE}.json" \
        --authtoken "$API_TOKEN"

    rm "/tmp/week-${i}.json"
done

# Combine results for analysis
python3 /scripts/analyze-trends.py /reports/trends/
```

Compliance Reporting

Generate compliance reports for audit requirements.

PCI Compliance Report (pci-compliance.sh):

```
#!/bin/bash

API_TOKEN="${LOGZILLA_API_KEY}"
REPORT_DATE=$(date +%Y%m%d)
MONTH_START=$(date -d "$(date +%Y-%m-01)" +%Y-%m-%d)
MONTH_END=$(date -d "$(date +%Y-%m-01) +1 month -1 day" +%Y-%m-%d)

# Ensure output dir exists
mkdir -p /compliance

# Precompute epoch seconds
MONTH_START_TS=$(date -d "${MONTH_START} 00:00:00" +%s)
MONTH_END_TS=$(date -d "${MONTH_END} 23:59:59" +%s)

# Authentication events
cat > /tmp/auth-events.json << EOF
{
  "time_range": {
    "ts_from": ${MONTH_START_TS},
    "ts_to": ${MONTH_END_TS}
  },
  "filter": [
    {
      "field": "message",
      "op": "qp",
      "value": ["authentication OR login OR logon OR access"]
    }
  ],
  "limit": 10000,
  "sort": ["-first_occurrence"],
  "with_archive": true
}
EOF

# Generate authentication report
logzilla query --type Search \
    --params "/tmp/auth-events.json" \
    --output-file "/compliance/auth-events-${REPORT_DATE}.xlsx" \
    --format xlsx \
    --authtoken "$API_TOKEN"

# Administrative access events
cat > /tmp/admin-events.json << EOF
{
  "field": "host",
  "limit": 50,
```

```
  "time_range": {
    "ts_from": ${MONTH_START_TS},
    "ts_to": ${MONTH_END_TS}
  },
  "filter": [
    {
      "field": "message",
      "op": "qp",
      "value": ["sudo OR su OR admin OR privilege"]
    }
  ],
  "subfields": ["program", "message"],
  "with_archive": true
}
EOF

logzilla query --type TopN \
    --params "/tmp/admin-events.json" \
    --output-file "/compliance/admin-access-${REPORT_DATE}.xlsx" \
    --format xlsx \
    --authtoken "$API_TOKEN"

rm /tmp/auth-events.json /tmp/admin-events.json
```

System Monitoring Automation

Performance Monitoring Dashboard

Automated system performance data collection for dashboards.

System Health Check (health-monitor.sh):

```
#!/bin/bash
set -euo pipefail

API_TOKEN="${LOGZILLA_API_KEY}"
TIMESTAMP=$(date +%Y%m%d-%H%M%S)
METRICS_DIR="/monitoring/metrics"
mkdir -p "$METRICS_DIR"

# CPU utilization (no exporter → redirect stdout)
echo '{"time_range": {"preset": "last_1_hours"}, "cpu": "totals"}' > /tmp/cpu.json
logzilla query --type System_CPU \
    --params /tmp/cpu.json \
    --authtoken "$API_TOKEN" \
    > "${METRICS_DIR}/cpu-${TIMESTAMP}.json"

# Memory utilization
echo '{"time_range": {"preset": "last_1_hours"}}' > /tmp/memory.json
```

```
logzilla query --type System_Memory \
    --params /tmp/memory.json \
    --authtoken "$API_TOKEN" \
    > "${METRICS_DIR}/memory-${TIMESTAMP}.json"

# Disk utilization
echo '{"time_range": {"preset": "last_1_hours"}, "fs": "root"}' > /tmp/disk.json
logzilla query --type System_DF \
    --params /tmp/disk.json \
    --authtoken "$API_TOKEN" \
    > "${METRICS_DIR}/disk-${TIMESTAMP}.json"

# Event processing rate
echo '{"time_range": {"preset": "last_1_hours", "step": 300}}' > /tmp/events.json
logzilla query --type EventRate \
    --params /tmp/events.json \
    --authtoken "$API_TOKEN" \
    > "${METRICS_DIR}/events-${TIMESTAMP}.json"

rm -f /tmp/cpu.json /tmp/memory.json /tmp/disk.json /tmp/events.json

# Process metrics for alerting
python3 /scripts/process-metrics.py "${METRICS_DIR}" "$TIMESTAMP"
```

Capacity Planning Data

Collect data for capacity planning and growth analysis.

Capacity Analysis (capacity-planning.sh):

```
#!/bin/bash

API_TOKEN="${LOGZILLA_API_KEY}"
OUTPUT_DIR="/capacity-planning/$(date +%Y-%m)"
mkdir -p "$OUTPUT_DIR"

# Storage statistics for last 30 days (no exporter → redirect stdout)
echo '{"time_range": {"preset": "last_30_days"}}' > /tmp/storage.json
logzilla query --type StorageStats \
    --params /tmp/storage.json \
    --authtoken "$API_TOKEN" \
    > "${OUTPUT_DIR}/storage-stats.json"

# Event rate trends
echo '{
  "time_range": {
    "preset": "last_30_days",
    "step": 86400
  }
}' > /tmp/event-trends.json
```

```
logzilla query --type EventRate \
    --params /tmp/event-trends.json \
    --output-file "${OUTPUT_DIR}/event-trends.json" \
    --authtoken "$API_TOKEN"

# Top hosts by volume
echo '{
  "field": "host",
  "limit": 100,
  "time_range": {"preset": "last_30_days"},
  "with_archive": true
}' > /tmp/top-hosts.json

logzilla query --type TopN \
    --params /tmp/top-hosts.json \
    --output-file "${OUTPUT_DIR}/top-hosts.xlsx" \
    --format xlsx \
    --authtoken "$API_TOKEN"

rm /tmp/{storage,event-trends,top-hosts}.json
```

Integration Examples

SIEM Integration

Export LogZilla data for SIEM correlation and analysis.

SIEM Export Script (siem-export.sh):

```
#!/bin/bash

API_TOKEN="${LOGZILLA_API_KEY}"
SIEM_DIR="/siem-exports"
BATCH_SIZE=10000

# Export security events for SIEM
cat > /tmp/security-export.json << EOF
{
  "time_range": {
    "preset": "last_1_hours"
  },
  "filter": [
    {
      "field": "severity",
      "op": "le",
      "value": [4]
    }
  ],
  "limit": ${BATCH_SIZE},
```

```
  "sort": ["-first_occurrence"]
}
EOF

# Export to JSON for SIEM ingestion
EXPORT_FILE="${SIEM_DIR}/security-events-$(date +%Y%m%d-%H%M).json"
logzilla query --type Search \
    --params /tmp/security-export.json \
    --output-file "$EXPORT_FILE" \
    --authtoken "$API_TOKEN"

# Convert to SIEM format if needed
if [ -f "$EXPORT_FILE" ]; then
    python3 /scripts/convert-to-siem-format.py "$EXPORT_FILE"

    # Send to SIEM via API or file transfer
    curl -X POST \
        -H "Content-Type: application/json" \
        -H "Authorization: token ${SIEM_API_TOKEN}" \
        --data-binary "@${EXPORT_FILE}" \
        "${SIEM_ENDPOINT}/api/events"
fi

rm /tmp/security-export.json
```

Ticketing System Integration

Automatically create tickets for critical events.

Ticket Creation (create-tickets.sh):

```
#!/bin/bash

API_TOKEN="${LOGZILLA_API_KEY}"
TICKET_THRESHOLD=10  # Create ticket if more than 10 critical events

# Check for critical events in last hour
cat > /tmp/critical-check.json << EOF
{
  "time_range": {
    "preset": "last_1_hours"
  },
  "filter": [
    {
      "field": "severity",
      "op": "le",
      "value": [2]
    }
  ],
  "limit": 1000
}
```

```
EOF

# Get critical events
RESULT_FILE="/tmp/critical-events-$(date +%Y%m%d-%H%M).json"
logzilla query --type Search \
    --params /tmp/critical-check.json \
    --output-file "$RESULT_FILE" \
    --authtoken "$API_TOKEN"

# Check event count and create ticket if threshold exceeded
EVENT_COUNT=$(jq '.results.totals.count // 0' "$RESULT_FILE")

if [ "$EVENT_COUNT" -gt "$TICKET_THRESHOLD" ]; then
    # Create ticket via API
    TICKET_DATA=$(cat << EOF
{
  "title": "Critical Events Alert - $EVENT_COUNT events detected",
  "description": "LogZilla detected $EVENT_COUNT critical events in the last hour. Please
investigate.",
  "priority": "high",
  "category": "system-alert",
  "attachment": "$RESULT_FILE"
}
EOF
)

    curl -X POST \
        -H "Content-Type: application/json" \
        -H "Authorization: token ${TICKETING_API_TOKEN}" \
        --data "$TICKET_DATA" \
        "${TICKETING_ENDPOINT}/api/tickets"
fi

rm /tmp/critical-check.json
```

Advanced Automation Patterns

Multi-Query Reports

Combine multiple queries for deeper analysis.

Multi-Query Report (comprehensive-report.sh):

```
#!/bin/bash

API_TOKEN="${LOGZILLA_API_KEY}"
REPORT_DIR="/reports/comprehensive/$(date +%Y%m%d)"
mkdir -p "$REPORT_DIR"
```

```
# Function to run query with error handling
run_query() {
    local query_type="$1"
    local params_file="$2"
    local output_file="$3"
    local format="${4:-json}"

    if logzilla query --type "$query_type" \
        --params "$params_file" \
        --output-file "$output_file" \
        --format "$format" \
        --authtoken "$API_TOKEN"; then
        echo "✓ Generated: $output_file"
    else
        echo "✗ Failed: $output_file"
        return 1
    fi
}

# Top hosts by volume
echo '{
  "field": "host",
  "limit": 20,
  "time_range": {"preset": "last_24_hours"}
}' > /tmp/top-hosts.json

run_query "TopN" "/tmp/top-hosts.json" "${REPORT_DIR}/top-hosts.xlsx" "xlsx"

# Error rate trends
echo '{
  "time_range": {
    "preset": "last_24_hours",
    "step": 3600
  },
  "filter": [{"field": "severity", "op": "le", "value": [4]}]
}' > /tmp/error-trends.json

run_query "EventRate" "/tmp/error-trends.json" "${REPORT_DIR}/error-trends.json"

# System performance
echo '{"time_range": {"preset": "last_24_hours"}, "cpu": "totals"}' > /tmp/cpu-perf.json
run_query "System_CPU" "/tmp/cpu-perf.json" "${REPORT_DIR}/cpu-performance.json"

# Generate summary report
python3 /scripts/generate-summary.py "$REPORT_DIR"

# Clean up
rm /tmp/{top-hosts,error-trends,cpu-perf}.json
```

Scheduled Automation

Set up cron jobs for regular automated reporting.

Crontab Configuration:

```
# Daily security report at 6 AM
0 6 * * * /scripts/security-report.sh

# Hourly system health check
0 * * * * /scripts/health-monitor.sh

# Weekly capacity planning report (Sundays at 2 AM)
0 2 * * 0 /scripts/capacity-planning.sh

# Monthly compliance report (1st of month at 3 AM)
0 3 1 * * /scripts/pci-compliance.sh
```

Best Practices for Automation

Error Handling

- **Check command exit codes** and handle failures gracefully
- **Validate JSON parameters** before executing queries
- **Log automation activities** for troubleshooting
- **Implement retry logic** for transient failures

Performance Optimization

- **Use specific time ranges** to limit query scope
- **Schedule resource-intensive queries** during off-peak hours
- **Implement query result caching** where appropriate
- **Monitor automation performance** and optimize as needed

Security and Maintenance

- **Protect API tokens** and configuration files
- **Rotate credentials regularly** for security
- **Monitor automation logs** for suspicious activity
- **Keep scripts and parameters updated** with system changes

Documentation and Monitoring

- **Document automation purposes** and dependencies

- **Monitor automation success rates** and performance
- **Implement alerting** for automation failures
- **Maintain version control** for automation scripts

These examples provide the foundation for building sophisticated LogZilla automation workflows. Adapt these patterns to the specific requirements and integrate them with the existing monitoring and reporting systems.