Zabbix Advanced

Aula 07: Monitoramento de Sistemas Operacionais

Linux e Windows - Coleta Avançada

4Linux - Curso Avançado

Agenda do Dia

1. **UserParameters Avançados**
    - Criação, sintaxe, validação, segurança
2. **system.run e Alternatives**
    - Quando usar, riscos, melhores práticas
3. **Monitoramento de Logs**
    - Filtros, expressões regulares, parsing

Agenda do Dia (cont.)

4. **Scripts Personalizados**
    - Bash, PowerShell, Python para métricas específicas
5. **Tuning de Performance**
    - CPU, RAM, Disco: coleta otimizada
6. **Laboratórios Práticos**
    - UserParameters Linux/Windows, log monitoring, scripts

PARTE 1

UserParameters Avançados

O Que São UserParameters?

UserParameters = Extensão do Zabbix Agent para métricas customizadas

```
Item Key Padrão → system.cpu.load[percpu,avg1]
                  ↓
UserParameter   → custom.metric[param1,param2]
                  ↓
Seu Script      → /usr/local/bin/my_script.sh $1 $2
```

Por que usar?

- ✅ Métricas específicas do negócio
- ✅ Coleta de dados não nativos
- ✅ Integração com ferramentas locais
- ✅ Flexibilidade total

Anatomia de um UserParameter

Sintaxe básica:

```
UserParameter=key[*],command $1 $2 $3
```

Elementos:

- **key**: Nome do item (ex: `custom.disk.usage`)
- **[*]**: Aceita parâmetros variáveis
- **command**: Script/comando a executar
- **$1, $2, $3**: Parâmetros passados pelo item

Localização do arquivo:

- Linux: `/etc/zabbix/zabbix_agentd.d/userparameter_*.conf`

Exemplo Simples: Contagem de Processos

UserParameter:

```
# /etc/zabbix/zabbix_agentd.d/userparameter_processes.conf
UserParameter=custom.process.count[*],ps aux | grep -c "$1"
```

Uso no Zabbix:

```
Item Key: custom.process.count[nginx]
Type: Zabbix agent
Returns: 3
```

⚠️ **Problema:** Inseguro! Permite command injection

Exemplo Seguro: Validação de Parâmetros

UserParameter melhorado:

```
#!/bin/bash
# /usr/local/bin/count_process.sh

PROCESS_NAME=$1

# Validar entrada (apenas alfanumérico e hífen)
if ! [[ "$PROCESS_NAME" =~ ^[a-zA-Z0-9_-]+$ ]]; then
    echo "0"
    exit 1
fi

# Contar processos
ps aux | grep -v grep | grep -c "$PROCESS_NAME"
```

```
UserParameter=custom.process.count[*],/usr/local/bin/count_process.sh "$1"
```

UserParameter com Múltiplos Parâmetros

Cenário: Verificar uso de disco por diretório

```
#!/bin/bash
# /usr/local/bin/check_directory_size.sh

DIRECTORY=$1
UNIT=$2  # K, M, G

if [ ! -d "$DIRECTORY" ]; then
    echo "0"
    exit 1
fi

du -s${UNIT} "$DIRECTORY" 2>/dev/null | awk '{print $1}'
```

```
UserParameter=custom.dir.size[*],/usr/local/bin/check_directory_size.sh "$1" "$2"
```

Uso: `custom.dir.size[/var/log,M]` → Retorna MB

UserParameter: JSON Output

Cenário: Retornar múltiplas métricas de uma vez

```
#!/bin/bash
# /usr/local/bin/app_metrics.sh

echo '{'
echo '  "requests": '$(cat /var/app/requests.count)','
echo '  "errors": '$(cat /var/app/errors.count)','
echo '  "latency_ms": '$(cat /var/app/latency.avg)
echo '}'
```

```
UserParameter=custom.app.metrics,/usr/local/bin/app_metrics.sh
```

No Zabbix: Use Dependent Items com JSONPath

UserParameters no Windows

PowerShell script:

```
# C:\Scripts\check_service_status.ps1
param($ServiceName)

$service = Get-Service -Name $ServiceName -ErrorAction SilentlyContinue
if ($service.Status -eq 'Running') { 1 } else { 0 }
```

UserParameter:

```
# C:\Program Files\Zabbix Agent\zabbix_agentd.conf
UserParameter=custom.service.status[*],powershell.exe -NoProfile -File "C:\Scripts\check_service_status.ps1" "$1"
```

Uso: `custom.service.status[W3SVC]` → 1 (running) ou 0 (stopped)

Testando UserParameters

1. Testar diretamente no shell:

```
/usr/local/bin/count_process.sh nginx
# Output: 3
```

2. Testar via zabbix_agentd:

```
zabbix_agentd -t custom.process.count[nginx]
# Output: 3
```

3. Verificar no Zabbix Server:

```
zabbix_get -s 192.168.1.100 -k custom.process.count[nginx]
# Output: 3
```

Boas Práticas: UserParameters

✅ DO:

- Validar TODOS os parâmetros de entrada
- Usar scripts externos em vez de comandos inline
- Testar performance (timeout padrão: 3s)
- Documentar cada UserParameter
- Usar exit codes apropriados (0 = sucesso)

❌ DON'T:

- Executar comandos destrutivos (rm, dd, format)
- Confiar em entrada do usuário sem validação
- Fazer loops infinitos ou queries lentas
- Expor informações sensíveis (senhas, tokens)

Segurança: AllowKey e DenyKey

Zabbix Agent 2 (Zabbix 7.0 LTS):

```
# Permitir apenas UserParameters específicos
AllowKey=custom.process.count[*]
AllowKey=custom.dir.size[*]

# Negar system.run
DenyKey=system.run[*]
```

Prioridade: DenyKey > AllowKey

Recomendação: Sempre usar AllowKey em produção!

PARTE 2

system.run e Alternatives

O Que É system.run?

system.run = Executa comandos arbitrários no host

```
Item Key: system.run[ls -la /tmp]
Type: Zabbix agent
Returns: Output do comando
```

⚠️ **DEPRECATED no Zabbix 7.0 LTS!**

Por quê?

- 🔴 Risco extremo de segurança
- 🔴 Command injection fácil
- 🔴 Dificulta auditoria
- 🔴 Sem controle de acesso granular

system.run: Por Que NÃO Usar

Exemplo de ataque:

```
Item Key: system.run[cat /etc/passwd]
          ↓
Item Key malicioso: system.run[rm -rf /data/*]
```

Problemas:

- Qualquer usuário do Zabbix com permissão de criar items pode executar comandos root (se agent roda como root)
- Sem validação de entrada
- Sem auditoria adequada
- Sem timeout configurável

Alternatives ao system.run

Método	Segurança	Flexibilidade	Recomendado
UserParameter	✅ Alta	✅ Total	✅ SIM
External Check	✅ Média	⚠️ Limitada	✅ OK
Global Scripts	✅ Alta	✅ Alta	✅ SIM
system.run	🔴 Baixa	✅ Total	❌ NÃO

Migração recomendada:

`system.run[command]` → `UserParameter=custom.key,command`

Quando system.run Era Usado

Casos antigos:

```
# ❌ Forma antiga (Zabbix < 7.0)
system.run[df -h / | tail -1 | awk '{print $5}' | sed 's/%//']
```

✅ Forma correta (Zabbix 7.0 LTS):

```
#!/bin/bash
# /usr/local/bin/check_root_usage.sh
df -h / | tail -1 | awk '{print $5}' | sed 's/%//'
```

```
UserParameter=custom.disk.root.usage,/usr/local/bin/check_root_usage.sh
```

Habilitando system.run (NÃO RECOMENDADO)

⚠️ IMPORTANTE - Mudanças no Zabbix 7.0:

Zabbix Agent (classic):

```
# /etc/zabbix/zabbix_agentd.conf

# ❌ DEPRECATED desde Zabbix 5.0.2 (ainda funciona mas gera warning)
EnableRemoteCommands=1

# ✅ FORMA CORRETA no Zabbix 7.0
AllowKey=system.run[*]
```

Zabbix Agent 2 (recomendado no Zabbix 7.0):

```
# /etc/zabbix/zabbix_agent2.conf

# ❌ NÃO EXISTE! EnableRemoteCommands foi REMOVIDO do Agent 2
```

⚠️ CUIDADO:

- Apenas em ambientes de teste/dev
- NUNCA em produção
- Sempre com auditoria ativada
- Considere usar sudoers + UserParameters
- `DenyKey=system.run[*]` é o padrão (mais seguro)

sudoers + UserParameters: Alternativa Segura

Cenário: Reiniciar serviço (requer root)

```
# /etc/sudoers.d/zabbix
zabbix ALL=(ALL) NOPASSWD: /bin/systemctl restart nginx
zabbix ALL=(ALL) NOPASSWD: /bin/systemctl restart mysql
```

```
#!/bin/bash
# /usr/local/bin/restart_service.sh
SERVICE=$1
if [[ "$SERVICE" =~ ^(nginx|mysql|apache2)$ ]]; then
    sudo /bin/systemctl restart "$SERVICE"
fi
```

```
UserParameter=custom.service.restart[*],/usr/local/bin/restart_service.sh "$1"
```

PARTE 3

Monitoramento de Logs

Log Monitoring no Zabbix

Tipos de monitoramento:

1. **log[] / logrt[]** - Active items (agent envia dados)
2. **Zabbix Sender** - Push manual de eventos
3. **External checks** - Parse de logs externos

Casos de uso:

- Detectar erros em logs de aplicação
- Monitorar tentativas de login falhas
- Rastrear eventos de segurança
- Alertar sobre stack traces

Item: log[] vs logrt[]

log[file,<regexp>,<encoding>,<maxlines>,<mode>,<output>]

- Monitora arquivo único
- **Exemplo:** `log[/var/log/messages]`

logrt[file_pattern,<regexp>,<encoding>,<maxlines>,<mode>,<output>]

- Monitora arquivos com rotação (regex no nome)
- **Exemplo:** `logrt[/var/log/app.log.*,ERROR]`
- Funciona com `app.log.1`, `app.log.2.gz`, etc.

⚠️ **Limitação:** Não funciona com systemd journal (use script)

Exemplo: Monitorar Erros no Syslog

Item configuration:

```
Name: Syslog errors
Type: Zabbix agent (active)
Key: logrt[/var/log/syslog,ERROR|CRITICAL]
Type of information: Log
Update interval: 30s
```

Trigger:

```
Name: Errors detected in syslog
Expression: find(/host/logrt[/var/log/syslog,ERROR|CRITICAL],,"regexp","ERROR")=1
Severity: Warning
```

Expressões Regulares em Logs

Sintaxe básica:

```
logrt[/var/log/app.log,<regex>]
```

Exemplos:

```
# Qualquer erro
ERROR|CRITICAL|FATAL

# Erro específico
OOM|OutOfMemoryError

# IP suspeito (força bruta)
Failed password for .* from 192\.168\.1\.[0-9]+

# HTTP 5xx
HTTP/1\.[01]" 5[0-9]{2}
```

Monitoramento Avançado: Output Format

Capturar linha completa + contexto:

```
Key: logrt[/var/log/nginx/error.log,upstream.*failed,UTF-8,100,skip,\0]
```

Parâmetros:

- `UTF-8` : Encoding
- `100` : Máximo de linhas por intervalo
- `skip` : Modo (skip ou all)
- `\0` : Output (linha completa)

No evento: Linha completa aparece na mensagem

Parsing JSON em Logs

Cenário: Aplicação loga em JSON

```
{"timestamp":"2025-01-07T10:30:00Z","level":"ERROR","message":"Database connection timeout","user":"admin"}
```

UserParameter + jq:

```
#!/bin/bash
# /usr/local/bin/parse_app_log.sh
tail -n 50 /var/log/app.log | grep ERROR | jq -r '.message' | wc -l
```

```
UserParameter=custom.app.errors,/usr/local/bin/parse_app_log.sh
```

Monitoramento de Windows Event Log

PowerShell script:

```
# C:\Scripts\check_event_log.ps1
param($LogName, $EventID, $Minutes)

$startTime = (Get-Date).AddMinutes(-$Minutes)
$events = Get-WinEvent -FilterHashtable @{
    LogName=$LogName
    ID=$EventID
    StartTime=$startTime
} -ErrorAction SilentlyContinue

$events.Count
```

```
UserParameter=custom.eventlog[*],powershell.exe -File "C:\Scripts\check_event_log.ps1" "$1" "$2" "$3"
```

Uso: `custom.eventlog[Security,4625,60]` → Login failures últimos 60min

Troubleshooting: Logs Não Aparecem

Checklist:

1. **Agent ativo?** Verifique `zabbix_agentd.conf`:

```
ServerActive=192.168.1.10
```

2. **Permissão de leitura?**

```
sudo -u zabbix cat /var/log/app.log
```

3. **Item ativo?** Tipo deve ser "Zabbix agent (active)"

4. **Regex válida?** Teste com `grep -E`:

```
grep -E "ERROR|CRITICAL" /var/log/app.log
```

PARTE 4

Scripts Personalizados para Métricas Específicas

Por Que Scripts Personalizados?

Casos onde itens nativos não são suficientes:

- Integração com APIs proprietárias
- Métricas de negócio (vendas/hora, pedidos em fila)
- Parsing complexo de arquivos
- Cálculos específicos (ex: taxa de conversão)
- Coleta de múltiplos sistemas simultaneamente

Linguagens recomendadas:

- Bash (simples, sempre disponível)
- Python (complexidade média, boas libs)
- PowerShell (Windows)

Exemplo: Script Python - Conexões MySQL

```
#!/usr/bin/env python3
# /usr/local/bin/mysql_connections.py

import MySQLdb
import sys

try:
    db = MySQLdb.connect(host="localhost", user="zabbix", passwd="password")
    cursor = db.cursor()
    cursor.execute("SHOW STATUS LIKE 'Threads_connected'")
    result = cursor.fetchone()
    print(result[1])
except Exception as e:
    print("0", file=sys.stderr)
    sys.exit(1)
```

```
UserParameter=custom.mysql.connections,/usr/local/bin/mysql_connections.py
```

Exemplo: PowerShell - IIS App Pool Status

```
# C:\Scripts\check_apppool.ps1
param($AppPoolName)

Import-Module WebAdministration
$pool = Get-WebAppPoolState -Name $AppPoolName

if ($pool.Value -eq 'Started') { 1 } else { 0 }
```

```
UserParameter=custom.iis.apppool[*],powershell.exe -File "C:\Scripts\check_apppool.ps1" "$1"
```

Uso: `custom.iis.apppool[DefaultAppPool]`

Script com Cache Local

Problema: Query pesada em API externa (lenta)

Solução: Cache com TTL

```
#!/bin/bash
# /usr/local/bin/api_metric_cached.sh

CACHE_FILE="/tmp/api_metric_cache"
CACHE_TTL=300  # 5 minutos

if [ -f "$CACHE_FILE" ]; then
    AGE=$(($(date +%s) - $(stat -c %Y "$CACHE_FILE")))
    if [ $AGE -lt $CACHE_TTL ]; then
        cat "$CACHE_FILE"
        exit 0
    fi
fi

# Buscar novo valor
VALUE=$(curl -s https://api.example.com/metric)
echo "$VALUE" > "$CACHE_FILE"
echo "$VALUE"
```

Exemplo: Monitorar Fila RabbitMQ

```
#!/usr/bin/env python3
# /usr/local/bin/rabbitmq_queue.py

import requests
import sys

queue_name = sys.argv[1]
api_url = f"http://localhost:15672/api/queues/%2F/{queue_name}"

try:
    r = requests.get(api_url, auth=('guest', 'guest'))
    data = r.json()
    print(data['messages'])
except Exception as e:
    print("0")
    sys.exit(1)
```

```
UserParameter=custom.rabbitmq.queue[*],/usr/local/bin/rabbitmq_queue.py "$1"
```

Performance: Scripts Otimizados

❌ **Script lento (20s para executar):**

```
for i in {1..1000}; do
    curl -s http://api.com/metric/$i | jq .value
done | awk '{sum+=$1} END {print sum}'
```

✅ **Script otimizado (2s):**

```
curl -s http://api.com/metrics/batch | jq '[.[] | .value] | add'
```

Regra: Timeout padrão do Zabbix Agent = 3s

Se script demora > 3s → Item fica "Not supported"

Debugging Scripts

1. Executar manualmente:

```
sudo -u zabbix /usr/local/bin/my_script.sh param1
```

2. Verificar logs do agent:

```
tail -f /var/log/zabbix/zabbix_agentd.log
```

3. Aumentar debug level temporariamente:

```
DebugLevel=4
```

4. Testar com zabbix_get:

```
zabbix_get -s 192.168.1.100 -k custom.metric[param1]
```

PARTE 5

Coleta Otimizada: CPU, RAM, Disco

Por Que Tuning de Coleta?

Problemas comuns:

- 🔴 Coleta muito frequente → overhead no agent
- 🔴 Coleta pouco frequente → perda de picos
- 🔴 Retenção excessiva → banco de dados enorme
- 🔴 Muitos itens desnecessários → ruído

Objetivo do tuning:

✅ Coletar o necessário, na frequência correta, com retenção apropriada

Tuning: Intervalo de Coleta

Recomendações por tipo de métrica:

Métrica	Intervalo	Justificativa
CPU load	1m	Detectar picos rapidamente
Memory usage	1m	RAM muda rapidamente
Disk space	10m	Muda lentamente
Network traffic	1m	Detectar anomalias
Process count	2m	Não muda constantemente
Log files	30s	Eventos críticos

Regra: Mais crítico = menor intervalo

Tuning: History Storage Period

Configuração por item:

```
History storage period: 90d  (default)
                        ↓
                        7d   (recomendado para maioria)
```

Trend storage period:

```
365d (padrão) → Suficiente para análise anual
```

Trends = agregação horária:

- Min, max, avg de cada hora
- Usado para gráficos de longo prazo

CPU Monitoring: Otimizações

Itens essenciais:

```
system.cpu.load[percpu,avg1]     # Load médio 1min (intervalo: 1m)
system.cpu.load[percpu,avg5]     # Load médio 5min (intervalo: 5m)
system.cpu.util[,user]           # CPU user % (intervalo: 1m)
system.cpu.util[,system]         # CPU system % (intervalo: 1m)
system.cpu.util[,iowait]         # CPU iowait % (intervalo: 1m)
```

⚠️ **Evitar:**

- Coleta por core individual (a menos que necessário)
- Intervalos < 30s (overhead)

Memory Monitoring: Otimizações

Itens essenciais:

```
vm.memory.size[available]          # RAM disponível (1m)
vm.memory.size[pavailable]         # RAM disponível % (1m)
vm.memory.utilization              # Uso de memória (1m)
vm.memory.size[used]               # RAM usada (1m)
system.swap.size[,pfree]           # Swap livre % (2m)
```

Linux específico:

```
vm.memory.size[cached]             # Page cache (5m)
vm.memory.size[buffers]            # Buffers (5m)
```

Disk Monitoring: Otimizações

Itens essenciais:

```
vfs.fs.size[/,free]                 # Espaço livre em bytes (10m)
vfs.fs.size[/,pfree]                # Espaço livre em % (10m)
vfs.fs.size[/,used]                 # Espaço usado (10m)
vfs.fs.inode[/,pfree]               # Inodes livres % (30m)
```

Disco I/O (apenas se necessário):

```
vfs.dev.read[sda,operations]     # Leituras/s (1m)
vfs.dev.write[sda,operations]    # Escritas/s (1m)
```

⚠️ **Inode monitoring é crucial!** Pode acabar antes do espaço

Preprocessamento: Reduzir Carga

Exemplo: Coletar apenas mudanças

Sem preprocessamento:

- 60 valores/hora × 24h = 1440 registros/dia
- Mesmo se valor não mudar

Com preprocessamento (Simple change):

- Apenas valores diferentes são armazenados
- Redução: ~80% para métricas estáveis (disk space)

Configuração:

```
Preprocessing steps:
  1. Simple change
```

Dependent Items: Performance++

Conceito: 1 item master → N items dependentes

Exemplo: Coletar /proc/meminfo uma vez

```
#!/bin/bash
# Master item
cat /proc/meminfo
```

```
Master item:
  Key: custom.meminfo
  Type: Zabbix agent
  Update interval: 1m

Dependent items:
  MemTotal:     JSONPath $.MemTotal
  MemFree:      JSONPath $.MemFree
  MemAvailable: JSONPath $.MemAvailable
```

Calculated Items: Reduzir Agent Load

Cenário: Calcular % de uso de CPU

❌ **Forma antiga:** Script calcula %

✅ **Forma otimizada:** Calculated item

```
Item 1 (Zabbix agent):
  Key: system.cpu.util[,idle]

Item 2 (Calculated):
  Key: cpu.usage.percent
  Formula: 100 - last(//system.cpu.util[,idle])
```

Vantagem: Cálculo no server, não no agent

Passive vs Active Items: Quando Usar

Passive (Server faz polling):

- Server conecta no agent
- Melhor para poucos hosts (< 100)
- Fácil troubleshooting (zabbix_get)

Active (Agent envia dados):

- Agent conecta no server
- Melhor para muitos hosts (> 100)
- Essencial para logs e traps
- Reduz carga no server

Recomendação Zabbix 7.0: Active sempre que possível

PARTE 6

Laboratórios Práticos

Lab 1: UserParameter - Contagem de Conexões TCP

Objetivo: Criar UserParameter para contar conexões TCP em estado ESTABLISHED

```
#Instalação do netstat
apt install net-tools -y
```

Passos:

1. Criar script `/usr/local/bin/tcp_connections.sh`:

```
#!/bin/bash
netstat -an | grep ESTABLISHED | wc -l
```

2. Criar UserParameter:

```
UserParameter=custom.tcp.established,/usr/local/bin/tcp_connections.sh
```

3. Reiniciar agent e testar com `zabbix_agentd -t`

Lab 1: Criando Item e Trigger

4. Criar item no Zabbix:

```
Data collection → Hosts → [Seu host] → Items → Create item

Name: TCP connections (ESTABLISHED)
Type: Zabbix agent
Key: custom.tcp.established
Type of information: Numeric (unsigned)
Update interval: 1m
```

5. Criar trigger:

```
Name: Too many TCP connections
Expression: last(/host/custom.tcp.established)>1000
Severity: Warning
```

Lab 2: Log Monitoring - Detectar SSH Bruteforce

Objetivo: Alertar quando houver > 5 tentativas de SSH falhas em 5min

1. Criar item:

```
Name: SSH failed authentication attempts
Type: Zabbix agent (active)
Key: logrt[/var/log/auth.log,Failed password]
Type of information: Log
Update interval: 30s
```

Lab 2: Trigger com Contagem

2. Criar trigger:

```
Name: SSH bruteforce detected
Expression:
  count(/host/logrt[/var/log/auth.log,Failed password],5m)>5
Severity: High
Description: More than 5 failed SSH login attempts in 5 minutes
```

3. Testar:

```
# Gerar tentativas falhas
for i in {1..10}; do
  ssh invalid_user@localhost
done
```

Lab 3: Script Python - Monitorar Temperatura

Objetivo: Criar script Python para monitorar temperatura via lm-sensors

1. Instalar dependências:

```
sudo apt-get install lm-sensors python3-pip
sudo sensors-detect --auto
pip3 install pysensors
```

2. Criar script:

```
#!/usr/bin/env python3
# /usr/local/bin/cpu_temp.py

from sensors import sensors
sensors.init()

try:
    for chip in sensors.iter_detected_chips():
        for feature in chip:
            if 'Core 0' in feature.label:
                print(int(feature.get_value()))
                break
finally:
    sensors.cleanup()
```

Lab 3: Windows - Monitorar Serviços

Objetivo: Monitorar status de serviços Windows com PowerShell

1. Criar script:

```
# C:\Scripts\check_service.ps1
param([string]$ServiceName)

$service = Get-Service -Name $ServiceName -ErrorAction SilentlyContinue

if ($null -eq $service) {
    Write-Output "2"  # Serviço não existe
} elseif ($service.Status -eq 'Running') {
    Write-Output "1"  # Running
} else {
    Write-Output "0"  # Stopped
}
```

Lab 3: UserParameter Windows

2. Adicionar em zabbix_agentd.conf:

```
UserParameter=custom.service.status[*],powershell.exe -NoProfile -ExecutionPolicy Bypass -File "C:\Scripts\check_service.ps1" "$1"
```

3. Criar item:

```
Name: W3SVC Service Status
Key: custom.service.status[W3SVC]
Type of information: Numeric (unsigned)
```

4. Criar trigger:

```
Expression: last(/host/custom.service.status[W3SVC])=0
Severity: High
```

PARTE 7

Troubleshooting Comum

Problema 1: UserParameter Não Funciona

Sintoma: Item fica "Not supported"

Checklist:

1. **Script tem permissão de execução?**

```
chmod +x /usr/local/bin/my_script.sh
```

2. **Zabbix user pode executar?**

```
sudo -u zabbix /usr/local/bin/my_script.sh
```

3. **Script retorna valor válido?**

```
/usr/local/bin/my_script.sh
echo $?  # Deve ser 0
```

Problema 1: UserParameter (cont.)

4. UserParameter está correto?

```
grep custom.metric /etc/zabbix/zabbix_agentd.d/*.conf
```

5. Agent foi reiniciado?

```
sudo systemctl restart zabbix-agent
```

6. Teste com zabbix_agentd:

```
zabbix_agentd -t custom.metric[param]
```

7. Teste com zabbix_get:

```
zabbix_get -s 192.168.1.100 -k custom.metric[param]
```

Problema 2: Script Timeout

Sintoma: Item fica "Not supported" com timeout

Causas:

- Script demora > 3s (timeout padrão)
- Comando travado (deadlock, I/O wait)

Solução 1: Aumentar timeout:

```
# /etc/zabbix/zabbix_agentd.conf
Timeout=10
```

Solução 2: Otimizar script:

- Adicionar timeout interno
- Usar cache

Problema 3: Log Monitoring Não Funciona

Sintoma: Logs não aparecem no Zabbix

Checklist:

1. **Item é Active?**

```
Type: Zabbix agent (active)  ← Deve ser active!
```

2. **ServerActive configurado?**

```
ServerActive=192.168.1.10
```

3. **Agent pode ler o arquivo?**

```
sudo -u zabbix cat /var/log/app.log
```

Problema 3: Log Monitoring (cont.)

4. Regex está correta?

```
grep -E "ERROR|CRITICAL" /var/log/app.log
```

5. Arquivo está crescendo?

```
tail -f /var/log/app.log
```

6. Verificar agent log:

```
tail -f /var/log/zabbix/zabbix_agentd.log | grep logrt
```

Problema 4: High CPU no Zabbix Agent

Sintoma: zabbix_agentd consumindo muita CPU

Causas comuns:

1. **UserParameter pesado executando muito frequentemente**
   - Verificar scripts com `top -u zabbix`
   - Aumentar intervalo de coleta
2. **Log monitoring em arquivo gigante**
   - Usar maxlines parameter
   - Rotacionar logs com maior frequência
3. **Muitos items passive com intervalo curto**
   - Migrar para active items

Problema 5: Windows - Script Não Executa

Sintoma: UserParameter PowerShell não retorna valor

Checklist:

1. **Execution Policy:**

```
Set-ExecutionPolicy RemoteSigned
```

2. **Zabbix Agent tem permissão?**
   - Rodar serviço como user com privilégios
   - Ou ajustar ACL do script
3. **Testar script manualmente:**

```
powershell.exe -File "C:\Scripts\my_script.ps1" param1
```

Problema 6: Valor Incorreto Retornado

Sintoma: Item coleta valor mas está errado

Debug:

1. **Adicionar log ao script:**

```
echo "DEBUG: param=$1" >> /tmp/debug.log
echo "DEBUG: result=$result" >> /tmp/debug.log
```

2. **Verificar encoding:**
    - UTF-8 vs ASCII
    - Windows: UTF-16 BOM pode causar problemas
3. **Verificar type of information:**
    - Numeric (unsigned) → apenas inteiros positivos
    - Numeric (float) → decimais
    - Text → strings

Boas Práticas: Resumo

✅ DO:

- Sempre validar parâmetros de entrada
- Usar scripts externos em vez de inline commands
- Testar manualmente antes de adicionar ao Zabbix
- Documentar cada UserParameter
- Usar Active items quando possível
- Aplicar Simple change para métricas estáveis
- Monitorar performance do próprio agent

❌ DON'T:

- NUNCA usar system.run em produção
- Não executar comandos destrutivos

Recursos Adicionais

Documentação oficial:

- https://www.zabbix.com/documentation/7.0/en/manual/config/items/userparameters
- https://www.zabbix.com/documentation/7.0/en/manual/config/items/itemtypes/log_items

Templates prontos:

- https://git.zabbix.com/projects/ZBX/repos/zabbix/browse/templates
- https://github.com/zabbix/community-templates

Exemplos de scripts:

- https://github.com/hermanekt/zabbix-agent-extension-elasticsearch

Revisão da Aula

Aprendemos:

1. ✅ UserParameters avançados com validação e segurança
2. ✅ Alternativas seguras ao system.run (deprecated)
3. ✅ Log monitoring com regex e parsing complexo
4. ✅ Scripts personalizados em Bash, Python, PowerShell
5. ✅ Otimizações de coleta para CPU, RAM, Disco
6. ✅ Dependent items e calculated items
7. ✅ Troubleshooting de problemas comuns

Próxima Aula

Aula 08: Monitoramento de Rede e Performance

Tópicos:

- Templates ICMP, TCP, HTTP
- Latência e packet loss
- Triggers de degradação de rede
- Coleta de tráfego via SNMP
- SLA e métricas de disponibilidade

Perguntas?

Obrigado!

4Linux - Zabbix Advanced Course