Zabbix Advanced

Aula 10: Monitoramento Web e Aplicações

Web Scenarios, HTTP Agent, APIs e JMX

4Linux - Curso Avançado

Agenda do Dia

1. **Web Scenarios Avançados**
   - Multi-step, autenticação, validações complexas
2. **HTTP Agent para APIs REST**
   - GET/POST/PUT/DELETE, headers, authentication
3. **Parsing de JSON e XML**
   - JSONPath, XML preprocessing, dependent items

Agenda do Dia (cont.)

4. **Monitoramento de Aplicações Java via JMX**
   - Configuração, métricas, troubleshooting
5. **Performance e Tempo de Resposta**
   - Métricas de latência, SLA web
6. **Detecção de Falhas e Alertas**
   - Triggers inteligentes, correlation
7. **Laboratórios Práticos**

PARTE 1

Web Scenarios Avançados

O Que São Web Scenarios?

Web Scenario = Sequência de requisições HTTP simulando usuário real

```
Cenário: Login na Coffee Shop (e-commerce de café)
  Step 1: GET /homepage      → Verifica se site carrega
  Step 2: POST /login        → Faz login
  Step 3: GET /products      → Acessa catálogo
  Step 4: POST /cart/add     → Adiciona ao carrinho
  Step 5: GET /checkout      → Finaliza compra
```

Por que usar?

- ✅ Monitorar jornada completa do usuário
- ✅ Detectar problemas antes dos clientes
- ✅ Medir performance end-to-end
- ✅ Validar funcionalidades críticas

Web Scenario vs Simple HTTP Check

Aspecto	Simple Check	Web Scenario
Complexidade	1 requisição	Múltiplos steps
Session/Cookies	❌	✅ Mantém entre steps
Variáveis	❌	✅ Extrai e reutiliza
Autenticação	Básica	Qualquer (form-based, OAuth)
Validação	Status code	Status + conteúdo + regex
Performance	Tempo total	Tempo por step
Use case	API health	User journey

Criando Web Scenario Básico

1. Acessar menu:

```
Data collection → Hosts → [Coffee Shop - app] → Web → Create web scenario

Name: Coffee Shop - Homepage Check
Application: Web monitoring
Update interval: 5m
Agent: Mozilla/5.0 (Zabbix)
Attempts: 3
```

Application: Grupo lógico para organizar scenarios

Attempts: Quantas tentativas antes de marcar como falha

Configurando Steps - Coffee Shop

2. Aba "Steps":

```
Step 1:
  Name: Homepage
  URL: http://172.16.1.111:8080/
  Required status codes: 200
  Required string: Coffee Shop
  Timeout: 15s
  Follow redirects: Yes

Step 2:
  Name: Product Search
  URL: http://172.16.1.111:8080/rest/products/search?q=coffee
  Required status codes: 200
  Required string: data
  Timeout: 15s
```

Variáveis em Web Scenarios

Usar macros para dados sensíveis:

```
Data collection → Hosts → [Host] → Macros

{$WEB_USERNAME} = admin
{$WEB_PASSWORD} = SecurePass123  (tipo: Secret text)
{$API_TOKEN} = Bearer eyJ...     (tipo: Secret text)
```

No step:

```
Post fields: username={$WEB_USERNAME}&password={$WEB_PASSWORD}
Headers: Authorization: {$API_TOKEN}
```

Vantagem: Alterar senha em 1 lugar, aplica a todos scenarios

Extraindo Variáveis Entre Steps

Cenário: Step 1 retorna token CSRF, Step 2 precisa usar

Step 1 (Login page):

```
Name: Get login form
URL: https://app.example.com/login
Required string: csrf_token

Preprocessing:
  Regular expression: <input name="csrf_token" value="(.*?)" → \1
  → Store to variable: {csrf}
```

Step 2 (Submit login):

```
Name: Submit login
URL: https://app.example.com/auth
Post fields: username=admin&password=pass&csrf_token={csrf}
```

Autenticação: Basic Auth

Servidor requer HTTP Basic Authentication:

```
Step 1:
  Name: API with Basic Auth
  URL: https://api.example.com/v1/health

  HTTP authentication:
    Username: monitor
    Password: {$API_PASSWORD}

  Required status codes: 200
```

Zabbix envia header automaticamente:

```
Authorization: Basic bW9uaXRvcjpwYXNzd29yZA==
```

Autenticação: Bearer Token

Exemplo real com Coffee Shop (JWT/Bearer token):

```
Step 1: Get token
  URL: http://172.16.1.111:8080/rest/user/login
  Post type: JSON data
  Post fields:
    {"email":"admin@juice-sh.op","password":"admin123"}
  Required string: authentication

  Variables:
    {AUTH_TOKEN} = regex:"token":"(.*?)"

Step 2: Use token para acessar API protegida
  URL: http://172.16.1.111:8080/api/Quantitys/
  Headers:
    Authorization: Bearer {AUTH_TOKEN}
  Required status codes: 200
```

Resposta do Step 1:

Validações Avançadas: Required String

Garantir que página tem conteúdo esperado:

```
Step: Product page
  URL: https://shop.example.com/product/123

  Required string: Add to Cart
  Required string: In Stock     ← Múltiplas strings (AND)

  Status: 200
```

Se página carrega mas sem "In Stock" → Falha detectada

Uso: Detectar páginas "quebradas" que retornam 200 mas com erro

Validações Avançadas: Regex

Validar formato de resposta:

```
Step: API response validation
  URL: https://api.example.com/status

  Required string (regex): "status":"(ok|healthy)"
  Required string (regex): "version":"[0-9]+\.[0-9]+\.[0-9]+"
```

Exemplo de resposta válida:

```
{
  "status": "ok",
  "version": "2.5.3",
  "uptime": 3600
}
```

Cookies e Session Management

Zabbix mantém cookies automaticamente entre steps:

```
Step 1: Login
  Server retorna: Set-Cookie: PHPSESSID=abc123; Path=/

Step 2: Dashboard
  Zabbix envia: Cookie: PHPSESSID=abc123

Step 3: Profile
  Zabbix envia: Cookie: PHPSESSID=abc123
```

Não precisa configurar nada!

⚠️ **Atenção:** Cada execução do scenario = nova sessão

Headers Customizados

Adicionar headers específicos:

```
Step: API with custom headers
  URL: https://api.example.com/data

  Headers:
    Content-Type: application/json
    X-API-Version: 2.0
    X-Request-ID: {HOST.HOST}-{TIME}
    User-Agent: Zabbix-Monitor/7.0
    Accept-Language: pt-BR,pt;q=0.9
```

{TIME} = Timestamp atual (macro do Zabbix)

Timeouts e Retries

Configuração global do scenario:

```
Update interval: 5m         # Executar a cada 5 minutos
Attempts: 3                 # 3 tentativas antes de falhar
```

Configuração por step:

```
Step: Slow API endpoint
  Timeout: 30s              # Máximo 30s para este step
  Follow redirects: Yes     # Seguir HTTP 301/302
```

Se step falha → Retry após 15s (automático)

Se 3 attempts falham → Scenario marcado como failed

Métricas Automáticas de Web Scenarios

Zabbix cria items automaticamente:

```
Download speed for scenario "User Login Flow"
  → Bytes/segundo do download

Download speed for step "Homepage" of scenario "User Login Flow"
  → Bytes/segundo de cada step

Failed step of scenario "User Login Flow"
  → Nome do step que falhou (ou empty se OK)

Last error message of scenario "User Login Flow"
  → Mensagem de erro detalhada

Response time for step "Login" of scenario "User Login Flow"
  → Tempo de resposta em segundos
```

Triggers Automáticas

Criadas automaticamente:

```
Trigger 1:
  Name: Failed step of scenario "User Login Flow"
  Severity: Average
  Expression: length(last(...))>0

Trigger 2:
  Name: Download speed for scenario "User Login Flow" is less than 10 KBps
  Severity: Warning
  Expression: max(...,5m)<10240
```

Pode customizar severidade e expressões

Grafana-Style Dashboard para Web Scenarios

Criar dashboard com múltiplos scenarios:

```
Dashboards → Create dashboard → Add widget

Widget type: Graph
Data set:
  - Response time Step 1 (Homepage)
  - Response time Step 2 (Login)
  - Response time Step 3 (Dashboard)
  - Response time Step 4 (Checkout)

Display: Stacked area
Legend: Bottom
```

Resultado: Visualizar qual step é mais lento

PARTE 2

HTTP Agent para APIs REST

HTTP Agent: Visão Geral

HTTP Agent = Item type para requisições HTTP avançadas

Vantagens sobre simple checks:

- ✅ Suporte completo a REST (GET/POST/PUT/DELETE/PATCH)
- ✅ Headers customizados ilimitados
- ✅ Autenticação avançada (Basic, Bearer, NTLM, Digest)
- ✅ Request body (JSON, XML, form-data)
- ✅ SSL/TLS avançado (client certs, custom CA)
- ✅ Preprocessing nativo (JSONPath, XML, regex)
- ✅ Proxy HTTP support

HTTP Agent: GET Request

Exemplo real - Coffee Shop API:

```
Data collection → Hosts → Items → Create item

Name: Coffee Shop - Application Version
Type: HTTP agent
URL: http://172.16.1.111:8080/rest/admin/application-version
Request type: GET
Timeout: 5s
Update interval: 1m

Request headers:
  Accept: application/json
  User-Agent: Zabbix/7.0

Type of information: Text
```

Retorna: `{"version":"14.5.1"}`

Parsing JSON com JSONPath

Extrair campo específico do Coffee Shop:

```
Item: Coffee Shop - Application Version (master)
  → Retorna JSON completo: {"version":"14.5.1"}

Dependent Item: Application Version Number
  Type: Dependent item
  Master item: Coffee Shop - Application Version

  Preprocessing:
    1. JSONPath: $.version

  Type of information: Text

  → Resultado: "14.5.1"
```

Exemplo com array de produtos:

JSONPath: Exemplos Comuns

JSON de exemplo:

```json
{
  "status": "ok",
  "data": {
    "users": 1523,
    "requests": 45231
  },
  "servers": [
    {"name": "web1", "cpu": 45},
    {"name": "web2", "cpu": 67}
  ]
}
```

JSONPath:

Path	Resultado
`$.status`	`"ok"`
`$.data.users`	`1523`
`$.servers[0].name`	`"web1"`
`$.servers[*].cpu`	`[45, 67]`
`$.servers[?(@.cpu>50)].name`	`"web2"`

HTTP Agent: POST Request

Enviar dados para API:

```
Name: Create user via API
Type: HTTP agent
URL: https://api.example.com/v1/users
Request type: POST
Request body type: JSON data

Request body:
{
  "username": "monitor",
  "email": "monitor@example.com",
  "role": "viewer"
}

Request headers:
  Content-Type: application/json
  Authorization: Bearer {$API_TOKEN}

Type of information: Text
```

Autenticação: Bearer Token

API moderna com token:

```
Name: API with Bearer Auth
Type: HTTP agent
URL: https://api.example.com/v1/metrics

Request headers:
  Authorization: Bearer {$API_TOKEN}
  Content-Type: application/json

HTTP authentication: None  (auth via header)
```

Macro {$API_TOKEN} configurada em:

```
Data collection → Hosts → [Host] → Macros
{$API_TOKEN} = eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
Type: Secret text
```

Autenticação: Basic Auth

API legada com Basic:

```
Name: Legacy API
Type: HTTP agent
URL: https://legacy.example.com/api/data

HTTP authentication: Basic
Username: apiuser
Password: {$API_PASSWORD}

(Zabbix gera header automaticamente)
```

Equivalente manual:

```
Request headers:
  Authorization: Basic YXBpdXNlcjpwYXNzd29yZA==
```

SSL/TLS: Verificação de Certificado

Configurações SSL:

```
Name: API HTTPS
Type: HTTP agent
URL: https://api.example.com/data

SSL verify peer: Yes       # Verificar certificado do servidor
SSL verify host: Yes       # Verificar hostname no certificado
SSL certificate file: /etc/zabbix/ssl/client.crt  # Client cert (opcional)
SSL key file: /etc/zabbix/ssl/client.key
SSL key password: {$SSL_PASSWORD}
```

Para certificados auto-assinados (DEV):

```
SSL verify peer: No
SSL verify host: No
```

Proxy HTTP Support

API acessível apenas via proxy corporativo:

```
Name: API via Proxy
Type: HTTP agent
URL: https://external-api.com/data

HTTP proxy: http://proxy.corp.local:8080

(Se proxy requer autenticação)
HTTP proxy authentication: Basic
Proxy username: proxyuser
Proxy password: {$PROXY_PASSWORD}
```

Query String Parameters

Passar parâmetros na URL:

```
Name: API with query params
Type: HTTP agent
URL: https://api.example.com/v1/search?q={$SEARCH_TERM}&limit=100&sort=desc

Request type: GET
```

Alternativa (mais legível):

```
URL: https://api.example.com/v1/search

Query fields:
  q = {$SEARCH_TERM}
  limit = 100
  sort = desc
```

Zabbix monta URL automaticamente

Preprocessing: Múltiplos Passos

Processar resposta complexa:

```
Item: API complex response
URL: https://api.example.com/stats
Returns: <xml><data><value>12345</value></data></xml>

Preprocessing:
  1. XML XPath: //data/value/text()         → 12345
  2. Discard unchanged with heartbeat: 1h   → Só armazena se mudar
  3. JavaScript: return value * 1.1         → Aplicar fator correção
  4. In range: 0 to 100000                  → Validar range
```

Ordem importa! Executado sequencialmente

Error Handling: Check for Error

Detectar erro no JSON:

```
Item: API that may return errors
Returns:
  Success: {"status":"ok","value":123}
  Error:   {"status":"error","message":"DB down"}

Preprocessing:
  1. Check for error using JSONPath: $.status
     Pattern: error
     Error output: $.message

  → Se status=error, item fica "Not supported" com mensagem "DB down"
```

Rate Limiting: Considerar

APIs com limite de requests/minuto:

```
Name: GitHub API (rate-limited)
URL: https://api.github.com/repos/zabbix/zabbix
Update interval: 5m   ← NÃO usar 30s (estoura rate limit)

Request headers:
  Accept: application/vnd.github.v3+json
  User-Agent: Zabbix-Monitor
```

GitHub API: 60 requests/hora sem auth

Solução: Usar token de autenticação → 5000 req/hora

Dependent Items: Otimização

1 request → múltiplas métricas:

```
Master Item: GitHub Repo Stats
  URL: https://api.github.com/repos/zabbix/zabbix
  Returns: {"stargazers_count":1523,"forks":450,"open_issues":89}
  Update interval: 10m

Dependent 1: Stars
  JSONPath: $.stargazers_count

Dependent 2: Forks
  JSONPath: $.forks

Dependent 3: Open Issues
  JSONPath: $.open_issues
```

Vantagem: 1 API call em vez de 3 → Respeita rate limit

PARTE 3

Parsing de JSON e XML

JSONPath: Sintaxe Completa

Operadores:

Operador	Descrição	Exemplo
`$`	Root object	`$.status`
`.`	Child operator	`$.data.value`
`[]`	Array access	`$.servers[0]`
`[*]`	All array elements	`$.servers[*].name`
`[start:end]`	Slice	`$.items[0:5]`
`?()`	Filter	`$.users[?(@.active==true)]`
`@`	Current item	`[?(@.cpu>80)]`

JSONPath: Arrays

JSON:

```
{
  "servers": [
    {"name":"web1","cpu":45,"ram":60},
    {"name":"web2","cpu":82,"ram":75},
    {"name":"web3","cpu":91,"ram":88}
  ]
}
```

Queries:

```
$.servers[*].cpu           → [45, 82, 91]
$.servers[?(@.cpu>80)]     → [{"name":"web2",...}, {"name":"web3",...}]
$.servers[?(@.cpu>80)].name → ["web2", "web3"]
```

JSONPath: Nested Objects

JSON:

```
{
  "application": {
    "name": "MyApp",
    "metrics": {
      "requests": {
        "total": 1000,
        "errors": 23
      }
    }
  }
}
```

Queries:

```
$.application.name                        → "MyApp"
$.application.metrics.requests.total      → 1000
$.application.metrics.requests.errors     → 23
```

JSONPath: Aggregation

Calcular média/soma (com preprocessing JavaScript):

```
Master Item: Cluster CPU Usage
  Returns: {"nodes":[{"cpu":45},{"cpu":67},{"cpu":52}]}

Dependent Item: Average CPU
  Preprocessing:
    1. JSONPath: $.nodes[*].cpu        → [45, 67, 52]
    2. JavaScript:
       var arr = JSON.parse(value);
       var sum = arr.reduce((a,b) => a+b, 0);
       return sum / arr.length;

  → Resultado: 54.67
```

XML Parsing: XPath

XML exemplo:

```
<?xml version="1.0"?>
<server>
  <status>online</status>
  <metrics>
    <cpu>67</cpu>
    <ram>82</ram>
  </metrics>
</server>
```

XPath queries:

```
//status/text()          → "online"
//metrics/cpu/text()     → "67"
//metrics/ram/text()     → "82"
```

XML Parsing: Namespaces

XML com namespace:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <m:Response xmlns:m="http://example.com">
      <m:Status>OK</m:Status>
    </m:Response>
  </soap:Body>
</soap:Envelope>
```

XPath com namespace:

```
//*[local-name()='Status']/text()    → "OK"
```

local-name() ignora namespace

Preprocessing: JavaScript Avançado

Transformações complexas:

```
Item: API response with timestamp
Returns: {"timestamp":"2025-01-07T10:30:00Z","value":123}

Preprocessing JavaScript:
var obj = JSON.parse(value);
var ts = new Date(obj.timestamp).getTime() / 1000; // Unix timestamp
return JSON.stringify({
  "timestamp_unix": ts,
  "value": obj.value,
  "value_doubled": obj.value * 2
});

Output: {"timestamp_unix":1736246400,"value":123,"value_doubled":246}
```

Validação de Dados: In Range

Garantir que valor está dentro do esperado:

```
Item: Temperature sensor via API
Returns: 23.5

Preprocessing:
  1. JSONPath: $.temperature
  2. In range: -40 to 85
     → Se fora do range, item fica "Not supported"
```

Uso: Detectar erros de sensor/API (valores impossíveis)

Custom Multiplier

Conversão de unidades:

```
Item: Bandwidth (API retorna em Mbps)
Returns: 125

Preprocessing:
  1. JSONPath: $.bandwidth_mbps
  2. Custom multiplier: 1048576
     → Converte Mbps para Bps (para gráfico)

Type of information: Numeric (unsigned)
Units: bps
```

125 Mbps × 1048576 = 131072000 Bps

Regular Expression: Captura

Extrair parte específica de texto:

```
Item: API version string
Returns: "Version: 2.5.3-beta (build 1234)"

Preprocessing:
  1. Regular expression: Version: ([0-9.]+)
     Output: \1

  → Resultado: "2.5.3"
```

Regex groups:

- `\1` = Primeiro grupo (entre parênteses)
- `\2` = Segundo grupo
- `\0` = Match completo

PARTE 4

Monitoramento de Aplicações Java via JMX

O Que É JMX?

JMX (Java Management Extensions)

- Tecnologia Java para monitoramento e gerenciamento
- Expõe métricas da JVM e aplicação
- Protocolo: RMI (Remote Method Invocation)

Casos de uso:

- Monitorar Tomcat, JBoss, WebLogic
- Coletar métricas de heap, threads, GC
- Monitorar connection pools (JDBC)
- Métricas customizadas da aplicação

Habilitando JMX na Aplicação

Java application startup:

```
java -Dcom.sun.management.jmxremote \
     -Dcom.sun.management.jmxremote.port=12345 \
     -Dcom.sun.management.jmxremote.authenticate=false \
     -Dcom.sun.management.jmxremote.ssl=false \
     -Dcom.sun.management.jmxremote.local.only=false \
     -jar myapp.jar
```

⚠️ **Produção:** Sempre usar authenticate=true e SSL!

Porta: 12345 (customizável)

JMX com Autenticação (Produção)

1. Criar arquivo de senha:

```
# /etc/zabbix/jmx/jmxremote.password
monitorRole readonlypassword
controlRole readwritepassword
```

```
chmod 600 /etc/zabbix/jmx/jmxremote.password
```

2. Criar arquivo de acesso:

```
# /etc/zabbix/jmx/jmxremote.access
monitorRole readonly
controlRole readwrite
```

JMX: Startup com Autenticação

3. Java startup:

```
java -Dcom.sun.management.jmxremote \
     -Dcom.sun.management.jmxremote.port=12345 \
     -Dcom.sun.management.jmxremote.authenticate=true \
     -Dcom.sun.management.jmxremote.password.file=/etc/zabbix/jmx/jmxremote.password \
     -Dcom.sun.management.jmxremote.access.file=/etc/zabbix/jmx/jmxremote.access \
     -Dcom.sun.management.jmxremote.ssl=false \
     -jar myapp.jar
```

Zabbix Java Gateway

Zabbix Server não fala JMX diretamente

Arquitetura:

```
Zabbix Server
     ↓ (solicita métricas JMX)
Zabbix Java Gateway (porta 10052)
     ↓ (conecta via JMX)
Java Application (porta 12345)
```

Java Gateway = Proxy JMX

Instalando Zabbix Java Gateway

Ubuntu 22.04:

```
sudo apt install zabbix-java-gateway -y
```

Configurar:

```
sudo nano /etc/zabbix/zabbix_java_gateway.conf
```

```
LISTEN_IP="0.0.0.0"
LISTEN_PORT=10052
START_POLLERS=5          # Processos paralelos
```

Iniciar:

```
sudo systemctl enable zabbix-java-gateway
sudo systemctl start zabbix-java-gateway
```

Configurar Zabbix Server para Java Gateway

Editar zabbix_server.conf:

```
sudo nano /etc/zabbix/zabbix_server.conf
```

Adicionar:

```
JavaGateway=192.168.1.15         # IP do Java Gateway
JavaGatewayPort=10052
StartJavaPollers=5               # Processos JMX no Server
```

Reiniciar Server:

```
sudo systemctl restart zabbix-server
```

Criar Host para JMX

No Zabbix frontend:

```
Data collection → Hosts → Create host

Host name: tomcat-app-01
Groups: Java applications

Interfaces:
  JMX:
    IP address: 192.168.1.20
    Port: 12345
    (Se auth habilitado)
    Username: monitorRole
    Password: readonlypassword

Templates:
  Apache Tomcat by JMX
```

Métricas JMX Comuns

JVM Heap:

```
jmx["java.lang:type=Memory","HeapMemoryUsage.used"]
jmx["java.lang:type=Memory","HeapMemoryUsage.max"]
jmx["java.lang:type=Memory","HeapMemoryUsage.committed"]
```

Threads:

```
jmx["java.lang:type=Threading","ThreadCount"]
jmx["java.lang:type=Threading","DaemonThreadCount"]
```

Garbage Collection:

```
jmx["java.lang:type=GarbageCollector,name=G1 Young Generation","CollectionCount"]
jmx["java.lang:type=GarbageCollector,name=G1 Young Generation","CollectionTime"]
```

Item JMX: Exemplo

Criar item manual:

```
Data collection → Hosts → tomcat-app-01 → Items → Create item

Name: JVM Heap Memory Used
Type: JMX agent
Key: jmx["java.lang:type=Memory","HeapMemoryUsage.used"]
Type of information: Numeric (unsigned)
Units: B
Update interval: 1m
JMX endpoint: service:jmx:rmi:///jndi/rmi://192.168.1.20:12345/jmxrmi
```

Descobrindo MBeans Disponíveis

Usar JConsole para explorar:

```
# Em máquina com Java JDK instalado
jconsole 192.168.1.20:12345
```

Ou usar jmxterm:

```
java -jar jmxterm.jar
open 192.168.1.20:12345
domains
beans
get -b java.lang:type=Memory HeapMemoryUsage
```

Lista todos MBeans e atributos disponíveis

Template: Apache Tomcat by JMX

Template oficial do Zabbix 7.0:

Métricas incluídas:

- JVM heap, non-heap, threads
- Tomcat connector metrics (requests/sec, bytes sent/received)
- Thread pool usage
- Session count
- HTTP status codes (200, 404, 500)
- Application deployment status

Discovery rules:

- Connectors (HTTP, AJP)

Trigger JMX: Heap Memory

Alertar quando heap > 90%:

```
Name: JVM Heap memory usage is high
Expression:
  (last(/tomcat-app-01/jmx[...HeapMemoryUsage.used]) /
   last(/tomcat-app-01/jmx[...HeapMemoryUsage.max])) > 0.9
Severity: Warning
Description: Heap usage: {ITEM.LASTVALUE1} / {ITEM.LASTVALUE2}
```

Troubleshooting JMX

Problema 1: Connection refused

```
Item: Not supported
Error: Cannot obtain JMX value
```

Verificar:

```
# Java Gateway rodando?
sudo systemctl status zabbix-java-gateway

# Java app escutando na porta JMX?
netstat -tulpn | grep 12345

# Firewall?
telnet 192.168.1.20 12345
```

Troubleshooting JMX (cont.)

Problema 2: Authentication failed

```
Error: Cannot connect to JMX: Security exception
```

Verificar:

```
# Credenciais corretas no Zabbix?
Interface JMX → Username/Password

# Arquivo de senha correto?
cat /etc/zabbix/jmx/jmxremote.password
# Permissão 600?
ls -l /etc/zabbix/jmx/jmxremote.password
```

PARTE 5

Performance e Tempo de Resposta

Métricas de Performance Web

Principais métricas:

Métrica	Descrição	Threshold Bom
TTFB	Time to First Byte	< 200ms
Page Load	Tempo total de carregamento	< 2s
DNS Lookup	Resolução DNS	< 50ms
TCP Connect	Handshake TCP	< 100ms
SSL Handshake	Negociação TLS	< 200ms
Download Speed	Velocidade de download	> 1 Mbps

Web Scenario: Tempos Detalhados

Items criados automaticamente:

```
Response time for step "Homepage" of scenario "User Flow"
  → Tempo total do step (TTFB + download)

Download speed for step "Homepage" of scenario "User Flow"
  → Velocidade de download (Bps)
```

Granularidade: Por step, não por fase (DNS, connect, etc.)

Para mais detalhe: Usar HTTP agent + preprocessing

HTTP Agent: Response Time

Item específico para medir latência:

```
Name: API Response Time
Type: HTTP agent
URL: https://api.example.com/health
Request type: HEAD   ← Mais rápido (sem body)
Retrieve mode: Headers
Type of information: Numeric (float)
Units: s
Update interval: 1m

Preprocessing:
  JavaScript:
    return Zabbix.getResponseTime();
```

Zabbix.getResponseTime() = Função nativa que retorna tempo em segundos

Calculated Item: Availability %

Calcular uptime/disponibilidade:

```
Master Item: Website availability check
  Key: net.tcp.service[http,,80]
  Returns: 1 (up) ou 0 (down)
  Update interval: 1m

Calculated Item: Website availability %
  Key: web.availability.percent
  Formula:
    avg(/host/net.tcp.service[http,,80],1d) * 100
  Type of information: Numeric (float)
  Units: %
  Update interval: 1h
```

Resultado: Percentual de uptime nas últimas 24h

SLA Tracking: Daily Report

Item para relatório diário:

```
Calculated Item: Daily Downtime (minutes)
  Formula:
    (1440 - (sum(/host/net.tcp.service[http,,80],1d) *
     count(/host/net.tcp.service[http,,80],1d))) *
    (1440 / count(/host/net.tcp.service[http,,80],1d))

  Type: Numeric (float)
  Units: minutes
  Update interval: 1d
```

1440 = minutos em 24h

Trigger:

```
Expression: last(/host/daily.downtime)>30
Severity: High
Description: Downtime hoje: {ITEM.LASTVALUE} minutos
```

Performance Baseline

Estabelecer baseline (média normal):

```
Item: API average response time (7 days)
  Type: Calculated
  Formula: avg(/host/api.response.time,7d)
  Update interval: 1h

Trigger: Response time anomaly
  Expression:
    last(/host/api.response.time) >
    last(/host/api.response.time.baseline) * 2
  Severity: Warning
  Description: Response time 2x acima da média de 7 dias
```

Detecta degradação sem threshold fixo

Percentiles: P95, P99

Zabbix não tem função nativa de percentile

Alternativa 1: Export para análise externa

```
Use Zabbix API para extrair dados
Processar com Python/R/Excel
Calcular percentiles
```

Alternativa 2: Approximação com JavaScript

```
Item: Response times (array)
  Armazena últimos 100 valores

Calculated item: Approximate P95
  JavaScript que ordena array e pega posição 95
```

PARTE 6

Detecção de Falhas e Alertas

Trigger: Step Failed

Web scenario - detectar step específico que falhou:

```
Name: Login step failed
Expression:
  find(/host/web.test.error[User Login Flow],,"like","Login")=1
Severity: High
Description: Step "Login" falhou: {ITEM.LASTVALUE}
```

Item: `web.test.error[Scenario Name]`

- Retorna nome do step que falhou
- Vazio se todos passaram

Trigger: Performance Degradation

Detectar lentidão progressiva:

```
Name: Website response time degraded
Expression:
  avg(/host/web.test.time[Homepage,Homepage],10m) >
  avg(/host/web.test.time[Homepage,Homepage],1d) * 1.5
Severity: Warning
Description: Tempo de resposta 50% acima da média diária
```

Compara: Média recente (10min) vs média histórica (1 dia)

Trigger: HTTP Error Codes

Detectar aumento de erros:

```
Item: HTTP 5xx count
  Type: HTTP agent
  URL: https://api.example.com/metrics/errors
  JSONPath: $.http_5xx_count
  Update interval: 1m

Trigger: High rate of server errors
  Expression:
    (last(/host/http.5xx.count) - last(/host/http.5xx.count,#2)) > 100
  Severity: High
  Description: > 100 erros 5xx no último minuto
```

Detecta spike (diferença entre 2 leituras consecutivas)

Trigger: Correlation com Múltiplos Itens

Problema complexo = múltiplos sintomas:

```
Name: Application degraded (multiple symptoms)
Expression:
  avg(/host/api.response.time,5m) > 2
  and
  avg(/host/http.5xx.count,5m) > 10
  and
  last(/host/jmx[...HeapMemoryUsage.used]) /
  last(/host/jmx[...HeapMemoryUsage.max]) > 0.85
Severity: High
Description: API lenta + erros 5xx + heap alto = problema grave
```

Reduz falsos positivos (só alerta se tudo ruim junto)

Event Correlation: Web Monitoring

Cenário: Web server down causa múltiplos alertas

Correlation rule:

```
Administration → Event correlation → Create correlation

Name: Web server down suppression

Conditions:
  New event tag: service = web
  Event type: Problem

Operations:
  Close old events: Yes
  Close new event: No

Tag for matching: hostname
```

Efeito: Múltiplos problemas [illegible] 1 incidente

Dependency: Infrastructure

Web app depende de DB e cache:

```
Trigger 1 (Database):
  Name: MySQL is down
  Severity: Disaster

Trigger 2 (Cache):
  Name: Redis is down
  Severity: High

Trigger 3 (App):
  Name: Web application is down
  Severity: High
  Dependencies:
    - MySQL is down
    - Redis is down
```

Se DB ou Redis caem → Suprime alerta do App

Action: Escalation por Tempo

Notificar equipes diferentes conforme duração:

```
Administration → Actions → Trigger actions → Create action

Name: Website down escalation

Conditions:
  Trigger name LIKE "Website"
  Trigger severity >= High

Operations:
  Step 1 (0-10 min):
    Send message to: On-call engineer

  Step 2 (10-30 min):
    Send message to: Team lead

  Step 3 (30+ min):
    Send message to: Director
```

Action: Auto-Remediation Web

Reiniciar serviço web automaticamente:

```
Action: Auto-restart web service

Conditions:
  Trigger = "HTTP service is down"
  Trigger value = PROBLEM

Operations:
  1. Send message (Telegram): "Tentando reiniciar..."
  2. Run script: restart_nginx.sh
     (Global script com Scope: Action operation)
  3. Wait: 60s

Recovery operations:
  Send message: "Serviço recuperado"
```

PARTE 7

Laboratórios Práticos

Lab 1: Web Scenario - Coffee Shop Login

Objetivo: Monitorar fluxo completo de compra na Coffee Shop

Aplicação: Coffee Shop em http://172.16.1.111:8080

1. Criar web scenario:

```
Data collection → Hosts → app (172.16.1.111) → Web → Create web scenario

Name: Coffee Shop - Purchase Flow
Update interval: 10m
Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Attempts: 2
```

Lab 1: Configurar Steps - Coffee Shop

2. Step 1 - Homepage:

```
Name: Homepage
URL: http://172.16.1.111:8080/
Required status codes: 200
Required string: Coffee Shop
Timeout: 15s
```

3. Step 2 - Product Search:

```
Name: Search Products
URL: http://172.16.1.111:8080/rest/products/search?q=coffee
Required status codes: 200
Required string: data
Variables:
  {PRODUCT_ID} = regex:"id":([0-9]+)
```

Lab 1: Steps com Autenticação

4. Step 3 - User Login:

```
Name: User Login
URL: http://172.16.1.111:8080/rest/user/login
Post type: JSON data
Post fields:
  {"email":"admin@juice-sh.op","password":"admin123"}
Required status codes: 200
Required string: authentication
Variables:
  {AUTH_TOKEN} = regex:"token":"(.*?)"
```

5. Step 4 - Add to Basket:

```
Name: Add to Basket
URL: http://172.16.1.111:8080/api/BasketItems/
Post type: JSON data
Post fields: {"ProductId":{PRODUCT_ID},"quantity":1}
```

Lab 1: Validar e Testar

6. Salvar scenario

7. Aguardar primeira execução (10 min)

8. Verificar resultados:

```
Monitoring → Hosts → app → Web

Scenario: Coffee Shop - Purchase Flow
  Last check: 2025-01-09 10:30
  Status: OK  (ou Failed com step name)
  Average speed: 0.8 MBps
```

9. Ver detalhes:

```
Monitoring → Latest data → Filter: "Coffee Shop"

Items:
  - Download speed for scenario "Coffee Shop - Purchase Flow"
  - Failed step of scenario "Coffee Shop - Purchase Flow"
  - Response time for step "Homepage"
  - Response time for step "User Login"
  ...
```

Lab 2: HTTP Agent - Coffee Shop API

Objetivo: Monitorar produtos da Coffee Shop via API REST e processar array JSON

1. Criar item master:

```
Data collection → Hosts → app → Items → Create item

Name: Coffee Shop - Products API (Raw)
Key: coffeeshop.products.raw
Type: HTTP agent
URL: http://172.16.1.111:8080/rest/products/search?q=
Request type: GET
Request headers:
  Accept: application/json
Type of information: Text
Update interval: 5m
```

Resposta esperada:

Lab 2: Dependent Items - Products

2. Total de Produtos:

```
Create item

Name: Coffee Shop - Total Products
Key: coffeeshop.products.total
Type: Dependent item
Master item: Coffee Shop - Products API (Raw)
Type of information: Numeric (unsigned)
Units: products

Preprocessing:
  JSONPath: $.data.length()
```

3. Preço Médio (JavaScript):

```
Name: Coffee Shop - Average Product Price
Key: coffeeshop.products.price.avg
Type: Dependent item
Master item: Coffee Shop - Products API (Raw)
Type of information: Numeric (float)
Units: $

Preprocessing:
  JavaScript:
    var data = JSON.parse(value);
    if (!data.data || data.data.length === 0) return 0;
    var total = 0;
    for (var i = 0; i < data.data.length; i++) {
        total += parseFloat(data.data[i].price || 0);
    }
    return (total / data.data.length).toFixed(2);
```

Lab 2: Triggers - Products

4. Criar trigger para poucos produtos:

```
Name: Coffee Shop - Low product count
Expression:
  last(/app/coffeeshop.products.total) < 5
Severity: Warning
Description: Product catalog has less than 5 items
```

5. Criar trigger para preço médio alto:

```
Name: Coffee Shop - High average price
Expression:
  last(/app/coffeeshop.products.price.avg) > 50
Severity: Information
Description: Average product price is above $50
```

Lab 3: HTTP Agent - Coffee Shop Quantities

Objetivo: Monitorar estoque de produtos via API REST com JSONPath filtering

1. Criar item master:

```
Data collection → Hosts → app → Items → Create item

Name: Coffee Shop - Product Quantities (Raw)
Key: coffeeshop.quantities.raw
Type: HTTP agent
URL: http://172.16.1.111:8080/api/Quantitys/
Request type: GET
Request headers:
  Accept: application/json
Type of information: Text
Update interval: 5m
```

Resposta esperada:

Lab 3: Dependent Items - Quantities

2. Total de produtos com estoque:

```
Name: Coffee Shop - Total Products in Stock
Key: coffeeshop.quantities.total
Type: Dependent item
Master item: Coffee Shop - Product Quantities (Raw)
Type of information: Numeric (unsigned)

Preprocessing:
  JSONPath: $.data.length()
```

3. Produtos com estoque baixo:

```
Name: Coffee Shop - Low Stock Count
Key: coffeeshop.quantities.lowstock
Type: Dependent item
Master item: Coffee Shop - Product Quantities (Raw)
Type of information: Numeric (unsigned)
```

Lab 3: Estoque Específico

4. Quantidade do produto 1 (Espresso Coffee):

```
Name: Coffee Shop - Product 1 Stock
Key: coffeeshop.product1.quantity
Type: Dependent item
Master item: Coffee Shop - Product Quantities (Raw)
Type of information: Numeric (unsigned)
Units: units

Preprocessing:
  JSONPath: $.data[?(@.ProductId==1)].quantity.first()
```

Lab 3: Triggers - Stock

5. Trigger para estoque baixo geral:

```
Name: Coffee Shop - Products with low stock detected
Expression:
  last(/app/coffeeshop.quantities.lowstock) > 0
Severity: Warning
Description: {ITEM.LASTVALUE} products have less than 10 units in stock
```

6. Trigger para produto 1 esgotado:

```
Name: Coffee Shop - Product 1 out of stock
Expression:
  last(/app/coffeeshop.product1.quantity) = 0
Severity: High
Description: Espresso Coffee is out of stock
```

Lab 4: JSON Parsing Avançado - Coffee Shop

Objetivo: Extrair dados complexos de arrays JSON com JSONPath filtering

A API Coffee Shop retorna array de produtos:

```json
{
  "status": "success",
  "data": [
    {"id":1,"name":"Espresso Coffee","price":2.99,"deluxePrice":1.99},
    {"id":5,"name":"Espress Machine","price":99.99,"deluxePrice":89.99},
    {"id":15,"name":"French Press","price":89.99,"deluxePrice":89.99}
  ]
}
```

Lab 4: Master Item - Product Search

1. Criar master item com filtro:

```
Name: Coffee Shop - Coffee Products (Raw)
Key: coffeeshop.products.coffee.raw
Type: HTTP agent
URL: http://172.16.1.111:8080/rest/products/search?q=coffee
Type of information: Text
Update interval: 5m
```

2. Status da API:

```
Name: Coffee Shop - API Status
Key: coffeeshop.api.status
Type: Dependent item
Master item: Coffee Shop - Coffee Products (Raw)

Preprocessing:
  JSONPath: $.status
```

Lab 4: Array Processing - Produtos Específicos

3. Extrair produto "Espresso Coffee" (primeiro com espresso no nome):

```
Name: Coffee Shop - Espresso Product Name
Key: coffeeshop.espresso.name
Type: Dependent item
Master item: Coffee Shop - Coffee Products (Raw)

Preprocessing:
  JSONPath: $.data[?(@.name=~'.*Espresso.*')].name.first()

Type: Character
```

4. Preço do produto Espresso:

```
Name: Coffee Shop - Espresso Price
Key: coffeeshop.espresso.price
Type: Dependent item
Master item: Coffee Shop - Coffee Products (Raw)
```

Lab 4: Filtering - Produtos Caros

5. Contar produtos acima de $50:

```
Name: Coffee Shop - Expensive Products Count
Key: coffeeshop.products.expensive
Type: Dependent item
Master item: Coffee Shop - Coffee Products (Raw)

Preprocessing:
  JSONPath: $.data[?(@.price>50)].length()

Type: Numeric (unsigned)
```

6. Criar trigger:

```
Name: Coffee Shop - Many expensive products
Expression:
  last(/app/coffeeshop.products.expensive) > 5
Severity: Information
```

Lab 5: Performance Baseline

Objetivo: Criar baseline e detectar anomalias

1. Item de response time:

```
Name: API Response Time
Type: HTTP agent
URL: https://api.example.com/health
Request type: HEAD

Preprocessing:
  JavaScript: return Zabbix.getResponseTime();

Type: Numeric (float)
Units: s
Update interval: 1m
```

Lab 5: Baseline Calculado

2. Baseline (média 7 dias):

```
Name: API Response Time Baseline (7d avg)
Type: Calculated
Formula: avg(/host/api.response.time,7d)
Type: Numeric (float)
Units: s
Update interval: 1h
```

3. Trigger de anomalia:

```
Name: API response time anomaly
Expression:
  last(/host/api.response.time) >
  last(/host/api.response.time.baseline) * 3
Severity: Warning
Description: Response time 3x acima da baseline de 7 dias
```

Lab 5: Teste

4. Simular lentidão:

```
# No servidor da API, adicionar delay artificial
# (método depende da aplicação)
# Ex: adicionar sleep(5) no código temporariamente
```

5. Aguardar 1-2 minutos

6. Verificar trigger dispara:

```
Monitoring → Problems

Problem: API response time anomaly
Severity: Warning
Age: 1m
```

Lab 6: Auto-Remediation Web Service

Objetivo: Reiniciar nginx automaticamente se down

1. Criar Global Script:

```
Administration → Scripts → Create script

Name: Restart Nginx
Scope: Action operation
Type: Script
Execute on: Zabbix agent
Commands:
  sudo systemctl restart nginx

Description: Auto-restart nginx service
```

Lab 6: sudoers

2. Configurar sudoers no web server:

```
sudo visudo -f /etc/sudoers.d/zabbix

# Adicionar:
zabbix ALL=(ALL) NOPASSWD: /bin/systemctl restart nginx
```

3. Criar trigger:

```
Name: Nginx is down
Expression:
  max(/webserver/net.tcp.service[http,,80],3m)=0
Severity: High
```

Lab 6: Action

4. Criar action:

```
Administration → Actions → Trigger actions → Create action

Name: Auto-restart nginx

Conditions:
  Trigger = "Nginx is down"
  Trigger value = PROBLEM

Operations:
  1. Send message to Telegram: "Nginx down, tentando restart"
  2. Run script: Restart Nginx
     On: Current host

Recovery operations:
  Send message: "Nginx recuperado"
```

Lab 6: Teste

5. Testar parando nginx:

```
# No web server
sudo systemctl stop nginx
```

6. Aguardar 3 minutos (trigger delay)

7. Verificar:

```
- Trigger dispara
- Action executa
- Script executa
- Nginx volta
- Trigger entra em recovery
- Mensagem de recovery enviada
```

8. Ver auditoria:

Lab 7: SLA Report

Objetivo: Gerar relatório de disponibilidade mensal

1. Item de availability:

```
Name: Website availability (last 30 days)
Type: Calculated
Formula: avg(/host/net.tcp.service[http,,80],30d) * 100
Type: Numeric (float)
Units: %
Update interval: 1h
History storage: 90d
```

Lab 7: Dashboard Widget

2. Criar dashboard:

```
Dashboards → Create dashboard → Add widget

Widget type: Plain text
Name: Website SLA (30 days)

Items:
  Website availability (last 30 days)

Advanced configuration:
  Show lines: 1
  Dynamic items: Off
```

3. Adicionar widget de gráfico:

```
Widget type: Graph (classic)
Name: Availability trend

Data set:
  Host: webserver
  Item: Website availability (last 30 days)

Time period: Last 90 days
```

Lab 7: Scheduled Report

4. Criar relatório agendado:

```
Reports → Scheduled reports → Create report

Name: Monthly SLA Report
Owner: Admin
Dashboards: Website SLA
Period: Previous month

Schedule:
  Start date: 01/01/2025
  End date: Never
  Period: Monthly (1st day)
  Time: 09:00

Recipients:
  User groups: IT Management
```

Será enviado automaticamente todo dia 1º do mês

PARTE 8

Troubleshooting e Best Practices

Problema 1: Web Scenario Sempre Falha

Sintoma: Step 1 OK, Step 2 (login) sempre falha

Diagnóstico:

1. Verificar POST fields:

```
# Usar browser Developer Tools (F12)
# Network → Login request → Payload
# Copiar exact field names (case-sensitive!)

username=admin   # Pode ser "user", "email", "login"
password=pass    # Pode ser "pass", "pwd", "senha"
```

2. CSRF token necessário?

```
<!-- Se página tem isso: -->
<input type="hidden" name="csrf_token" value="abc123">

<!-- Precisa extrair no Step 1 e usar no Step 2 -->
```

Problema 1: Solução CSRF

Step 1 modificado:

```
Name: Get login form
URL: https://app.com/login

Variables:
  Extract from response:
    Name: csrf
    Regular expression: name="csrf_token" value="(.*?)"
    → \1
```

Step 2 modificado:

```
Name: Submit login
Post fields: username=admin&password=pass&csrf_token={csrf}
```

Problema 2: HTTP Agent SSL Error

Sintoma: "SSL certificate problem: self signed certificate"

Solução DEV:

```
Item → HTTP agent

SSL verify peer: No
SSL verify host: No
```

Solução PRODUÇÃO:

```
# Adicionar CA ao trust store do Zabbix Server
sudo cp custom-ca.crt /usr/local/share/ca-certificates/
sudo update-ca-certificates

# Reiniciar Zabbix Server
sudo systemctl restart zabbix-server
```

Problema 3: JSONPath Retorna Vazio

Sintoma: Item dependent fica sem dados

Debug:

1. Ver resposta do master item:

```
Monitoring → Latest data → Master item → History

Ver JSON completo retornado
```

2. Testar JSONPath online:

```
https://jsonpath.com/

Colar JSON
Testar path: $.data.value
```

3. Erros comuns:

```
$.data.users    ← Case-sensitive! Deve ser exato
$.data[0].name  ← Array index correto?
$.status        ← Campo realmente existe?
```

Problema 4: JMX Connection Refused

Sintoma: "Cannot obtain JMX value"

Checklist:

1. Java Gateway rodando?

```
sudo systemctl status zabbix-java-gateway
sudo netstat -tulpn | grep 10052
```

2. Java app com JMX habilitado?

```
ps aux | grep jmxremote
# Deve mostrar -Dcom.sun.management.jmxremote.port=12345
```

3. Porta acessível?

```
telnet 192.168.1.50 12345
```

Problema 4: Java Gateway Config

4. Server configurado para usar Gateway?

```
grep JavaGateway /etc/zabbix/zabbix_server.conf
# Deve mostrar:
# JavaGateway=192.168.1.15
# StartJavaPollers=5
```

5. Interface JMX correta no host?

```
Data collection → Hosts → [Host] → Interfaces

JMX: 192.168.1.50:12345  ← IP e porta corretos?
```

Problema 5: Rate Limit Exceeded

Sintoma: API retorna 429 Too Many Requests

Solução:

1. Aumentar update interval:

```
Update interval: 1m → 15m
```

2. Usar autenticação (rate limit maior):

```
Request headers:
  Authorization: Bearer {$API_TOKEN}
```

3. Usar dependent items (1 request → N métricas):

```
Master item: Update interval 15m
Dependent items: Processam mesmo JSON
```

Best Practice 1: Web Scenario Naming

❌ **Ruim:**

```
Scenario: Test 1
  Step: Step1
  Step: Step2
```

✅ **Bom:**

```
Scenario: E-commerce Checkout Flow
  Step: Homepage
  Step: Product listing
  Step: Add to cart
  Step: Checkout
  Step: Payment
```

Facilita troubleshooting e reports

Best Practice 2: Macros para Secrets

❌ Ruim:

```
Post fields: username=admin&password=SuperSecret123
```

✅ Bom:

```
Host → Macros:
  {$WEB_USERNAME} = admin
  {$WEB_PASSWORD} = SuperSecret123  (tipo: Secret text)

Post fields: username={$WEB_USERNAME}&password={$WEB_PASSWORD}
```

Vantagens:

- Não expõe senha nos logs
- Fácil rotação de credenciais

Best Practice 3: Dependent Items

❌ Ruim (3 API calls):

```
Item 1: GET /api/stats → Extrai $.users
Item 2: GET /api/stats → Extrai $.requests
Item 3: GET /api/stats → Extrai $.errors
```

✅ Bom (1 API call):

```
Master: GET /api/stats → JSON completo
Dependent 1: JSONPath $.users
Dependent 2: JSONPath $.requests
Dependent 3: JSONPath $.errors
```

Reduz carga na API e respeita rate limits

Best Practice 4: Error Handling

❌ Ruim:

```
HTTP agent sem error handling
→ Se API retorna erro, item fica "Not supported" sem contexto
```

✅ Bom:

```
Preprocessing:
  1. Check for error using JSONPath: $.status
     Pattern: error
     Custom error: $.error_message
```

Logs mostram mensagem real do erro

Best Practice 5: Timeouts Realistas

❌ **Ruim:**

```
Timeout: 3s  (para API que demora 5-10s)
→ Falsos positivos constantes
```

✅ **Bom:**

```
Timeout: 30s  (para operações pesadas)
Timeout: 5s   (para health checks simples)
```

Ajustar conforme SLA da API

Best Practice 6: Update Intervals

Recomendações:

Tipo	Intervalo	Justificativa
Health check crítico	1m	Detectar falha rápido
API pública	5-15m	Respeitar rate limit
Web scenario	5-10m	Não sobrecarregar
JMX	1-2m	Métricas Java mudam rápido
Metrics agregadas	15m-1h	Não mudam frequentemente

Best Practice 7: Templates

Criar templates reutilizáveis:

```
Template: App - REST API Monitoring
  Items:
    - HTTP agent: /health endpoint
    - HTTP agent: /metrics endpoint
  Dependent items:
    - Status, version, uptime
  Triggers:
    - API down
    - High error rate
  Macros:
    {$API_URL}
    {$API_TOKEN}
```

Aplicar em 50 APIs: Atualizar template → Aplica em todas

Recursos Adicionais

Documentação oficial:

- https://www.zabbix.com/documentation/7.0/en/manual/web_monitoring
- https://www.zabbix.com/documentation/7.0/en/manual/config/items/itemtypes/http
- https://www.zabbix.com/documentation/7.0/en/manual/config/items/preprocessing

JSONPath tester:

- https://jsonpath.com/
- https://jsonpath.herokuapp.com/

JMX:

- https://docs.oracle.com/javase/tutorial/jmx/
- https://github.com/zabbix/zabbix/tree/master/src/zabbix_java

Revisão da Aula

Aprendemos:

1. ✅ Web Scenarios multi-step com autenticação
2. ✅ HTTP Agent para APIs REST (GET/POST/PUT/DELETE)
3. ✅ JSONPath e XML parsing avançado
4. ✅ JMX monitoring para aplicações Java
5. ✅ Métricas de performance e SLA tracking
6. ✅ Triggers inteligentes e auto-remediation
7. ✅ 7 laboratórios práticos hands-on
8. ✅ Troubleshooting e best practices

Perguntas?

Obrigado!

4Linux - Zabbix Advanced Course