Zabbix Advanced

Aula 01: Templates, Itens, Triggers e Dependências

4Linux - Curso Avançado

Agenda do Dia

1. **Conceitos Fundamentais**
    - Templates e sua importância
    - Boas práticas
2. **Itens e Pré-processamento**
    - Tipos de itens
    - Pré-processamento avançado
3. **Triggers Compostos**
    - Expressões avançadas
    - Histerese

Agenda do Dia (continuação)

4. **Regras de Dependência**
    - Conceitos
    - Implementação prática
5. **JavaScript Avançado**
    - Script items
    - Casos práticos
6. **Macros**
    - Tipos e escopo
    - Automatizações

Apresentação

Jeovany Batista da Silva

- Construtor na 4Linux
- MIT em DevOps e Cloud Computing
- Batendo cabeça com tecnologia há mais de 18 anos
- Viciado em OpenSource;
- Trabalhando com um pouco de tecnologias: Observabilidade, DevOps, Kubernetes e Infraestrutura como Código (IaC), ELK, Monitoring Tools...

PARTE 1

Conceitos Fundamentais

O que são Templates?

Templates são **"moldes"** que padronizam configurações de monitoramento

Vantagens:

- 🎯 Padronização do monitoramento
- 🔧 Facilidade de manutenção
- 📦 Reutilização entre ambientes
- ⚡ Escalabilidade
- ✅ Redução de erros

ROI de Templates Bem Estruturados

Métrica	Sem Templates	Com Templates	Ganho
Tempo config novo servidor	45-60 min	2-3 min	**~95%** ⬇️
Consistência	60-70%	100%	**40%** ⬆️
Ajustar threshold	2-4h	5 min	**~97%** ⬇️
Escalabilidade	100 hosts/admin	1.000 hosts/admin	**10x** 🚀

Economia: R$ 15.800 (83% redução de custos)

ROI anual: >400%

Estrutura de um Template

```
Template Linux by Zabbix Agent
├── Application Groups
│   ├── CPU
│   ├── Memory
│   ├── Network
│   └── Filesystem
├── Items
│   ├── system.cpu.load[all,avg1]
│   ├── vm.memory.size[available]
│   └── vfs.fs.size[/,used]
├── Triggers
│   ├── High CPU load
│   └── Low memory available
└── Macros
    ├── {$MEMORY.THRESHOLD} = 20M
    └── {$CPU.THRESHOLD} = 5
```

Boas Práticas: Nomenclatura

Templates:

- `Template [Tecnologia] by [Método]`
- Exemplo: `Template Linux by Zabbix Agent`

Items:

- `[categoria].[subcategoria][.parâmetros]`
- Exemplo: `system.cpu.load[all,avg1]`

Triggers:

- Descrição clara do problema
- Exemplo: `High CPU load on {HOST.NAME}`

Boas Práticas: Organização

Por Aplicações:

- CPU → Todos os itens de processamento
- Memory → Memória física, swap, cache
- Network → Interfaces, tráfego, erros
- Filesystem → Disco, inodes

Macros para Personalização:

```
{$CPU.UTIL.CRIT} = 90
{$MEMORY.AVAILABLE.MIN} = 20M
{$VFS.FS.PUSED.MAX.WARN} = 80
```

💬 Discussão

Pergunta para você:

> Quem já trabalhou sem templates?
> Quais problemas enfrentou?

Compartilhe:

- Tempo gasto configurando hosts manualmente
- Inconsistências encontradas
- Dificuldades em manutenção

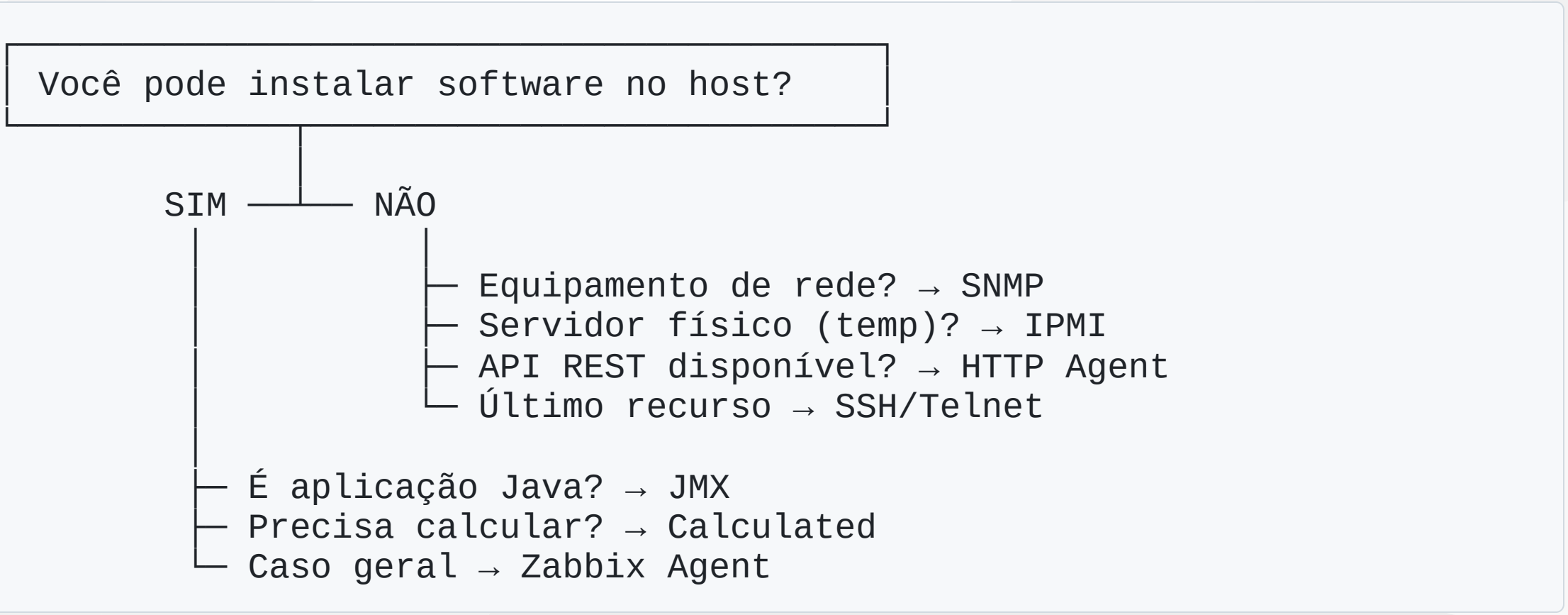
PARTE 2

Itens Personalizados e Pré-processamento

Tipos de Itens: Visão Geral

Tipo	Executa Em	Quando Usar	Performance
Zabbix Agent	Host monitorado	Sistemas com agente	⚡ Alta
SNMP	Servidor Zabbix	Equipamentos rede	🔸 Média
HTTP Agent	Servidor Zabbix	APIs REST	🔸 Média
Script	Servidor Zabbix	Lógica complexa	🔻 Baixa
Calculated	Servidor Zabbix	Cálculos	⚡ Alta

Guia de Decisão: Qual Tipo Usar?



Zabbix Agent: Versões

Versão	Linguagem	Características
Agent 1	C	Clássico, estável, modo passivo/ativo
Agent 2	Go	Plugins nativos, melhor performance, async

Chaves Principais:

- Sistema: `system.cpu.load`, `system.uptime`
- Memória: `vm.memory.size[available]`
- Processos: `proc.num[apache2]`
- Rede: `net.if.in[eth0]`
- Filesystem: `vfs.fs.size[/,used]`

Pré-processamento de Valores

Tipos Principais:

1. **Regular Expression (Regex)**

```
Pattern: Temperature: ([0-9]+)°C
Output: \1
```

2. **JSONPath**

```
$.server.cpu.usage
```

3. **JavaScript**

```
return Math.round(value * 100) / 100;
```

JSONPath: Exemplos

Entrada JSON:

```
{
  "server": {
    "cpu": {"usage": 45.7},
    "memory": {"free": 1024, "total": 8192}
  }
}
```

Extrações:

- `$.server.cpu.usage` → `45.7`
- `$.server.memory.free` → `1024`
- `$.server.memory.total` → `8192`

JavaScript Preprocessing

Exemplo: Conversão de Estado

```
var state = ['green', 'yellow', 'red'];
return state.indexOf(value.trim()) === -1
    ? 255
    : state.indexOf(value.trim());
```

Resultado:

- "green" → 0
- "yellow" → 1
- "red" → 2
- outros → 255

🔬 Laboratório Prático 1

Objetivo: Criar item HTTP com pré-processamento

Passos:

1. Criar item HTTP Agent
2. URL: `https://jsonplaceholder.typicode.com/posts/1`
3. Aplicar JSONPath: `$.title`
4. JavaScript: converter para maiúsculas
5. Testar e validar

Tempo: 30 minutos

⚠️ **Atenção:** Ordem dos pré-processamentos importa!

PARTE 3

Triggers com Expressões Compostas

Anatomia de um Trigger

```
{host:key.function(parameters)} operator value
```

Componentes:

- **host:** Nome do host ou template
- **key:** Chave do item
- **function:** avg, max, min, last, count...
- **parameters:** Período, contagem
- **operator:** >, <, =, <>
- **value:** Valor de referência ou macro

ROI de Triggers Inteligentes

Métrica	Triggers Simples	Triggers Compostos	Ganho
Falsos positivos	30-40%	5-10%	**~75%** ⬇️
Tempo investigando	15-20h/mês	2-3h/mês	**~85%** ⬇️
MTTR	30-45 min	5-10 min	**~80%** ⬇️
Precisão	60-70%	90-95%	**30%** ⬆️

Economia mensal: R$ 975

ROI anual: >35.000%

Trigger Simples vs Composto

❌ **Trigger Simples (gera falsos positivos):**

```
{host:cpu.load.last()} > 5
```

- Alerta em qualquer pico momentâneo
- 100 alertas/mês (40 falsos positivos)

✅ Trigger Composto (inteligente):

```
{host:cpu.load.avg(5m)} > 5 AND
{host:cpu.load.avg(15m)} > 3 AND
{host:cpu.load.last()} > 4
```

- 12 alertas/mês (1 falso positivo)
- **Economia:** R$ 975/mês

Funções Avançadas

Funções Temporais:

```
avg(5m)     → Média dos últimos 5 minutos
max(10m)    → Máximo dos últimos 10 minutos
min(1h)     → Mínimo da última hora
last()      → Último valor
```

Funções de Contagem:

```
count(30m, 8, "gt") → Quantas vezes foi > 8 em 30min
sum(1h)             → Soma da última hora
```

Funções de Mudança:

```
change()    → Diferença entre atual e anterior
diff()      → Houve mudança? (1/0)
delta(1h)   → Diferença entre max e min em 1h
```

Operadores Lógicos

AND (e lógico):

```
{host:cpu.load.avg(5m)} > 5 and
{host:memory.free.last()} < 500M
```

OR (ou lógico):

```
{host:disk.free.last()} < 1G or
{host:disk.pfree.last()} < 10
```

Agrupamento com Parênteses:

```
({host:cpu.load.avg(5m)} > 8 and {host:memory.free.last()} < 200M)
or
({host:disk.pfree.last()} < 5 and {host:disk.inode.pfree.last()} < 10)
```

Recovery Expression e Histerese

Problema sem Histerese:

- CPU: 89% → Alerta! 🚨
- CPU: 88% → OK ✅
- CPU: 89% → Alerta! 🚨
- **Oscilação constante** 😵

Solução com Histerese:

```
Problem:  {host:cpu.load.last()} > 90
Recovery: {host:cpu.load.last()} < 80
```

- Alerta quando > 90%
- Recupera apenas quando < 80%
- **Estável!** 😍

🔬 Laboratório Prático 2

Objetivo: Criar trigger com histerese

Requisitos:

1. Trigger dispara quando:
    - CPU média (5min) > 80%
    - CPU média (15min) > 70%
    - CPU atual > 75%

2. Recovery quando:
    - CPU média (5min) < 60%
    - CPU média (15min) < 50%

Tempo: 40 minutos

PARTE 4

Regras de Dependência entre Triggers

ROI de Dependências

Cenário: Switch core falha, afetando 50 servidores

❌ **Sem dependências:**

- 275 alertas em 5 minutos 📧📧📧
- 45 min investigando cada alerta
- MTTR: 60 min
- **Custo:** R$ 300

✅ **Com dependências:**

- 1 alerta (274 suprimidos) 🎯
- Causa raiz imediata
- MTTR: 15 min
- **Custo:** R$ 25

Economia: R$ 275/incidente (92% redução)

Tipos de Dependência

1. Dependência de Infraestrutura:

```
Internet → Router → Switch → Server → Service → Process
```

2. Dependência de Aplicação:

```
Database → App Server → Web Server → Load Balancer
```

3. Dependência de Recurso:

```
Physical Host → Hypervisor → VM → Container → App
```

Exemplo: Infraestrutura E-commerce

```
Load Balancer (nginx-lb-01)
├── Web Server 1 (web-01)
│   ├── Apache Service
│   └── PHP-FPM Service
├── Web Server 2 (web-02)
│   ├── Apache Service
│   └── PHP-FPM Service
├── Application Server (app-01)
│   ├── Java Application
│   └── Redis Cache
└── Database Server (db-01)
    ├── MySQL Service
    └── Backup Job
```

Configuração de Dependências

Nível 0 (sem dependências):

```
"Load Balancer unreachable"
```

Nível 1 (depende do LB):

```
"Web Server 01 unreachable"
  ↳ Dependência: "Load Balancer unreachable"
```

Nível 2 (depende do Web Server):

```
"Apache service down on web-01"
  ↳ Dependência: "Web Server 01 unreachable"
```

🔬 Laboratório Prático 3

Objetivo: Implementar dependências hierárquicas

Infraestrutura:

- Load Balancer → Web Servers → App Server → Database

Tarefas:

1. Criar triggers para cada camada
2. Configurar dependências hierárquicas
3. Simular falha do Load Balancer
4. Verificar supressão de alertas downstream

Tempo: 30 minutos

PARTE 5

JavaScript em Itens e Triggers

JavaScript no Zabbix

Engine: Duktape

Limitações Técnicas:

- ⏱️ Timeout: 10 segundos
- 💾 Memória: 512MB heap
- ⚠️ Falhas: 3 consecutivas reiniciam engine
- 🔄 Concorrência: Single-threaded por worker

Contextos de Uso:

- Item Preprocessing
- Script Items
- Global Scripts

Script Item: OAuth2 API

```
// Autenticação OAuth2
var req = new HttpRequest();
req.addHeader('Content-Type', 'application/x-www-form-urlencoded');

var body = 'grant_type=client_credentials' +
           '&client_id=' + '{$OAUTH.CLIENT.ID}' +
           '&client_secret=' + '{$OAUTH.CLIENT.SECRET}';

var authResponse = req.post('{$OAUTH.TOKEN.URL}', body);
var authData = JSON.parse(authResponse);
var token = authData.access_token;

// Coletar métricas com token
req.addHeader('Authorization', 'Bearer ' + token);
var response = req.get('{$API.METRICS.URL}');
return JSON.parse(response).cpu.usage;
```

Exemplo Real: Template AWS

Templates de Cloud do Zabbix = Exemplos Perfeitos!

Acesse no Zabbix:

- Configuration → Templates
- Filtrar por "AWS" / "Azure" / "Google"

Aprenda com:

- AWS EC2 by HTTP (autenticação, múltiplas APIs)
- Azure by HTTP (OAuth2, JSON complexo)
- Google Cloud by HTTP (Stackdriver)

💡 Exercício Guiado

Vamos juntos:

1. Abrir template "AWS EC2 by HTTP"
2. Examinar item de discovery
3. Analisar código JavaScript
4. Identificar:
    - Como faz autenticação?
    - Como processa JSON?
    - Como trata erros?

Adaptaremos para API customizada!

PARTE 6

Macros e Automatizações

Tipos de Macros

Built-in Macros (sistema):

- `{HOST.NAME}` → Nome do host
- `{ITEM.VALUE}` → Último valor
- `{TRIGGER.SEVERITY}` → Severidade
- `{EVENT.DATE}` → Data do evento

User Macros (usuário):

- `{$CPU.THRESHOLD}` → Threshold CPU
- `{$MEMORY.MIN}` → Memória mínima

Context Macros (contexto):

- `{$PASSWORD:MySQL}` → Senha específica MySQL
- `{$PORT:HTTP}` → 80
- `{$PORT:HTTPS}` → 443

Escopo e Precedência de Macros

```
Host Level (Highest Priority) 🥇
     ↓
Template Level (Medium Priority) 🥈
     ↓
Global Level (Lowest Priority) 🥉
```

Exemplo de Resolução:

```
Global:   {$CPU.LOAD.WARN} = 5
Template: {$CPU.LOAD.WARN} = 3  (override)
Host:     {$CPU.LOAD.WARN} = 8  (override)

Resultado final para o host: 8
```

Precedência em Ação

Global Macros:

```
{$CPU.LOAD.WARN} = 5
{$MEMORY.THRESHOLD} = 80
{$SNMP.COMMUNITY} = "public"
```

Template "Linux by Zabbix Agent":

```
{$CPU.LOAD.WARN} = 3          (overrides)
{$MEMORY.THRESHOLD} = 85       (overrides)
{$DISK.THRESHOLD} = 90         (new)
```

Host "production-web-01":

```
{$CPU.LOAD.WARN} = 8         (overrides)
{$CUSTOM.SETTING} = "prod"    (new)
```

Resultado Final

Para host "production-web-01":

```
{$CPU.LOAD.WARN} = 8          ← do host
{$MEMORY.THRESHOLD} = 85       ← do template
{$DISK.THRESHOLD} = 90         ← do template
{$SNMP.COMMUNITY} = "public" ← global
{$CUSTOM.SETTING} = "prod"    ← do host
```

Flexibilidade:

- Define padrão global
- Ajusta por tipo (template)
- Personaliza exceções (host)

Macros em Actions

Subject:

```
[{TRIGGER.SEVERITY}] {TRIGGER.NAME} on {HOST.NAME}
```

Message:

```
Problem Details:
Host: {HOST.NAME} ({HOST.IP})
Trigger: {TRIGGER.NAME}
Severity: {TRIGGER.SEVERITY}
Value: {ITEM.LASTVALUE}
Time: {EVENT.DATE} {EVENT.TIME}
Duration: {EVENT.DURATION}

Acknowledgment: {EVENT.ACK.STATUS}
{EVENT.ACK.HISTORY}
```

PARTE 7

Laboratório Prático Integrado

🎯 Exercício Final Completo

Objetivo: Criar template "Web Application Advanced"

Componentes:

1. Macros (10 min)
2. Itens com pré-processamento (30 min)
3. Triggers compostos (30 min)
4. Dependências (20 min)

Total: 1h30

Avaliação: Peer review + validação do instrutor

Etapa 1: Macros (10 min)

Criar as seguintes macros no template:

```
{$WEB.URL} = "http://localhost"
{$WEB.PORT} = "8080"
{$WEB.RESPONSE.TIME.WARN} = "5"
{$WEB.RESPONSE.TIME.CRIT} = "10"
{$API.TOKEN} = "your-token"
{$DB.CONNECTION.MAX} = "100"
{$MEMORY.APP.WARN} = "512M"
{$MEMORY.APP.CRIT} = "768M"
```

Etapa 2: Itens (30 min)

Item A: Response Time (HTTP Agent)

- URL: `{$WEB.URL}:{$WEB.PORT}/health`
- Preprocessing:
  i. JSONPath: `$.response_time_ms`
  ii. JavaScript: `return value / 1000`
  iii. Discard unchanged: 300s

Item B: Application Metrics (HTTP Agent)

- URL: `{$WEB.URL}:{$WEB.PORT}/metrics`
- Headers: `Authorization: Bearer {$API.TOKEN}`
- Preprocessing: JavaScript complexo (processar JSON)

Etapa 3: Triggers (30 min)

Trigger A: Response Time Alto

```
Problem Expression:
{Template:web.response.time.avg(5m)} > {$WEB.RESPONSE.TIME.WARN}
and
{Template:web.response.time.last()} > {$WEB.RESPONSE.TIME.WARN}

Recovery Expression:
{Template:web.response.time.avg(5m)} < {$WEB.RESPONSE.TIME.WARN} * 0.8
```

Trigger B: Database Connections Alto

```
{Template:app.metrics.last()} > {$DB.CONNECTION.MAX} * 0.8
and
{Template:app.metrics.avg(10m)} > {$DB.CONNECTION.MAX} * 0.7
```

Etapa 4: Dependências (20 min)

Criar:

1. Trigger "Application Unreachable"

```
{Template:web.response.time.nodata(5m)} = 1
```

2. Configurar dependências:
    - "Response Time Alto" depende de "Application Unreachable"
    - "DB Connections Alto" depende de "Application Unreachable"
3. Testar:
    - Simular falha de aplicação
    - Verificar supressão de alertas

Critérios de Avaliação

Templates:

- ✅ Estrutura clara e organizada
- ✅ Nomenclatura consistente

Itens:

- ✅ Tipo correto escolhido
- ✅ Pré-processamento funcional

Triggers:

- ✅ Expressões compostas corretas
- ✅ Sem falsos positivos
- ✅ Recovery expression com histerese

Dependências:

- ✅ Hierarquia lógica
- ✅ Supressão efetiva

ENCERRAMENTO

Recap dos Principais Conceitos

✅ **Templates:** Padronização e ROI de 400%+
✅ **Itens:** Escolha correta do tipo + pré-processamento
✅ **Triggers:** Expressões compostas reduzem falsos positivos em 75%
✅ **Dependências:** Supressão de 95% dos alertas em cascata
✅ **JavaScript:** Flexibilidade para casos complexos
✅ **Macros:** Host → Template → Global

Próxima Aula: Criação de Hosts e Métodos de Coleta

Recursos Úteis

📚 **Documentação Zabbix:**
https://www.zabbix.com/documentation/7.0/

🧪 **JSONPath Tester:**
https://jsonpath.com/

💻 **JavaScript Duktape:**
https://duktape.org/

🔗 **Templates da Comunidade:**
https://share.zabbix.com/

Obrigado!

Até a próxima aula! 🚀