Zabbix Advanced

Aula 08: Monitoramento de Rede e Performance

ICMP, TCP, Latência, SLA e Tráfego

4Linux - Curso Avançado

Agenda do Dia

1. **Monitoramento ICMP**
    - Simple checks, templates, latência, packet loss
2. **Monitoramento de Portas TCP/UDP**
    - Service checks, aplicações específicas
3. **Monitoramento de Serviços Web**
    - HTTP agent, status codes, certificados SSL

Agenda do Dia (cont.)

4. **Triggers de Rede Avançados**
    - Falha de link, degradação, flapping
5. **SLA e Métricas de Disponibilidade**
    - Cálculo de uptime, relatórios
6. **Laboratórios Práticos**

PARTE 1

Monitoramento ICMP

O Que É ICMP?

ICMP (Internet Control Message Protocol)

- Protocolo da camada de rede (Layer 3)
- Usado para diagnóstico e controle
- Implementado pelo comando `ping`

Casos de uso no Zabbix:

- ✅ Verificar se host está "vivo" (reachability)
- ✅ Medir latência (response time)
- ✅ Detectar packet loss
- ✅ Monitorar roteadores, switches, firewalls
- ✅ Dispositivos que não aceitam agent (IoT, impressoras)

Simple Checks: icmpping

Item types no Zabbix:

```
Simple check → Executado pelo Zabbix Server/Proxy
               (não precisa de agent no host)
```

Items ICMP disponíveis:

Item Key	Descrição	Retorno
icmpping	Host responde?	1 (up) ou 0 (down)
icmppingsec	Response time	Segundos (float)
icmppingloss	Packet loss	Percentual (0-100)

Criando Item ICMP: Reachability

Configuração no Zabbix 7.0 LTS:

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

Name: ICMP ping
Type: Simple check
Key: icmpping
Type of information: Numeric (unsigned)
Update interval: 1m
Host interface: (selecionar interface com IP correto)
```

Retorno:

- `1` = Host respondeu ao ping
- `0` = Host não respondeu (down ou ICMP bloqueado)

Criando Item ICMP: Response Time

Latência (tempo de resposta):

```
Name: ICMP response time
Type: Simple check
Key: icmppingsec
Type of information: Numeric (float)
Units: s
Update interval: 1m
```

Interpretação:

- `< 0.010` (10ms) = Excelente (LAN)
- `0.010 - 0.050` (10-50ms) = Bom (WAN)
- `0.050 - 0.150` (50-150ms) = Aceitável (Internacional)
- `> 0.150` (>150ms) = Degradado

Criando Item ICMP: Packet Loss

Perda de pacotes:

```
Name: ICMP packet loss
Type: Simple check
Key: icmppingloss
Type of information: Numeric (float)
Units: %
Update interval: 1m
```

Interpretação:

- `0%` = Perfeito
- `< 1%` = Aceitável
- `1-5%` = Degradado
- `> 5%` = Grave (pode causar instabilidade TCP)

Configurando Parâmetros ICMP

Ajustar quantidade e tamanho de pacotes:

```
Data collection → Hosts → [Host] → Items → [Item ICMP]

Key: icmpping[,4,,500]
       │       │ │  │
       │       │ │  └─ Timeout (ms) - padrão: 500ms
       │       │ └──── Size (bytes) - padrão: 56 bytes
       │       └────── Count - padrão: 3 pacotes
       └──────────────── (IP pode ser especificado aqui)
```

Exemplo: `icmpping[,10,,1000]`

- Envia 10 pings
- Timeout de 1000ms cada

Trigger ICMP: Host Down

Trigger básica:

```
Name: Host is unreachable via ICMP
Expression: last(/host/icmpping)=0
Severity: High
Manual close: No
```

Problema: Falso positivo em 1 pacote perdido

Trigger melhorada:

```
Expression: max(/host/icmpping,5m)=0
Description: Host não respondeu nenhum ping nos últimos 5 minutos
```

Trigger ICMP: High Latency

Detectar latência alta:

```
Name: ICMP response time is high
Expression: min(/host/icmppingsec,5m)>0.100
Severity: Warning
Description: Latência consistentemente acima de 100ms nos últimos 5min
```

Trigger de degradação progressiva:

```
Name: ICMP latency critical
Expression: min(/host/icmppingsec,5m)>0.250
Severity: High
```

Trigger ICMP: Packet Loss

Detectar perda de pacotes:

```
Name: ICMP packet loss detected
Expression: min(/host/icmppingloss,10m)>1
Severity: Warning
Description: Packet loss > 1% por 10 minutos consecutivos
```

Trigger crítica:

```
Name: ICMP packet loss critical
Expression: min(/host/icmppingloss,5m)>5
Severity: High
Description: Packet loss > 5% - link degradado
```

ICMP: Troubleshooting

Item não coleta dados (Not supported):

1. **Firewall bloqueando ICMP?**

```
# Testar manualmente do Zabbix Server
ping -c 3 192.168.1.100
```

2. **Interface configurada corretamente?**
    - Verificar IP na interface do host
3. **Zabbix Server tem permissão para ICMP?**

```
# Verificar se socket raw está permitido
getcap /usr/sbin/zabbix_server
# Deve mostrar: cap_net_raw+ep
```

ICMP: Ajustando Permissões

Se ICMP não funciona (Permission denied):

```
# Opção 1: Dar permissão de raw socket (RECOMENDADO)
sudo setcap cap_net_raw+ep /usr/sbin/zabbix_server
sudo systemctl restart zabbix-server

# Opção 2: Executar fping com setuid (MENOS SEGURO)
sudo chmod u+s /usr/bin/fping

# Opção 3: Executar Zabbix Server como root (NÃO RECOMENDADO)
# Apenas em ambientes de desenvolvimento
```

PARTE 2

Monitoramento de Portas TCP/UDP

Simple Check: net.tcp.service

Verificar se porta TCP está aberta:

```
Item Key: net.tcp.service[<service>,<ip>,<port>]
```

Serviços pré-definidos:

Service	Porta Padrão	Descrição
ssh	22	SSH
smtp	25	Mail server
http	80	Web server
https	443	Web SSL/TLS
mysql	3306	MySQL/MariaDB

net.tcp.service: Exemplos

HTTP na porta 80:

```
Name: HTTP service check
Type: Simple check
Key: net.tcp.service[http,,80]
Type of information: Numeric (unsigned)
Update interval: 1m
```

⚠️ Sintaxe Zabbix 7.0 LTS:

`net.tcp.service[http,,80]` (IP vazio = usar interface do host)

Retorno:

- `1` = Porta aberta e serviço respondendo
- `0` = Porta fechada ou sem resposta

net.tcp.service: Portas Customizadas

✅ Aplicação WEB em porta customizada (USE http):

```
Name: Tomcat HTTP check (port 8080)
Type: Simple check
Key: net.tcp.service[http,,8080]
Type of information: Numeric (unsigned)
```

Por quê usar `http`?

- Valida que é REALMENTE HTTP funcionando
- Não apenas verifica se porta está aberta
- Mais confiável para aplicações web

⚠️ Protocolo NÃO HTTP (use tcp genérico):

```
Name: PostgreSQL custom port
Type: Simple check
Key: net.tcp.service[tcp,,5433]

Name: Redis check
Type: Simple check
Key: net.tcp.service[tcp,,6379]
```

Regra: HTTP em qualquer porta → use `http`. Outros protocolos → use `tcp`

net.tcp.service.perf: Response Time

Medir tempo de resposta do serviço:

```
Name: HTTP response time
Type: Simple check
Key: net.tcp.service.perf[http,,80]
Type of information: Numeric (float)
Units: s
Update interval: 1m
```

Retorno: Tempo em segundos para estabelecer conexão TCP

Uso: Detectar degradação de performance do serviço

Trigger: Service Down

Trigger simples:

```
Name: HTTP service is down
Expression: last(/host/net.tcp.service[http,,80])=0
Severity: High
```

Trigger com hysteria (evitar flapping):

```
Name: HTTP service is down
Expression: max(/host/net.tcp.service[http,,80],3m)=0
Severity: High
Recovery expression: min(/host/net.tcp.service[http,,80],1m)=1
Description: HTTP service não respondeu por 3 minutos consecutivos
```

Trigger: Service Response Time

Degradação de performance:

```
Name: HTTP service slow
Expression: min(/host/net.tcp.service.perf[http,,80],5m)>2
Severity: Warning
Description: Tempo de resposta TCP > 2s consistentemente
```

Crítico:

```
Name: HTTP service critical slow
Expression: min(/host/net.tcp.service.perf[http,,80],5m)>5
Severity: High
```

UDP Service Check

Problema: `net.tcp.service` NÃO funciona para UDP

Solução: UserParameter com `nc` (netcat)

```
#!/bin/bash
# /usr/local/bin/check_udp.sh
HOST=$1
PORT=$2
TIMEOUT=3

nc -u -z -w $TIMEOUT $HOST $PORT 2>&1
if [ $? -eq 0 ]; then
    echo "1"
else
    echo "0"
fi
```

```
UserParameter=custom.udp.check[*],/usr/local/bin/check_udp.sh "$1" "$2"
```

Exemplo UDP: DNS Check

Verificar servidor DNS (porta 53 UDP):

```
#!/bin/bash
# /usr/local/bin/check_dns.sh
DNS_SERVER=$1

# Fazer query DNS
dig @$DNS_SERVER google.com +short +time=2 +tries=1 > /dev/null 2>&1

if [ $? -eq 0 ]; then
    echo "1"
else
    echo "0"
fi
```

```
UserParameter=custom.dns.check[*],/usr/local/bin/check_dns.sh "$1"
```

Item: `custom.dns.check[8.8.8.8]`

Monitoramento Multi-Porta

Cenário: Aplicação usa múltiplas portas

Solução: LLD (Low-Level Discovery) + Item prototype

```
{
  "data": [
    {"{#PORT}": "80", "{#SERVICE}": "HTTP"},
    {"{#PORT}": "443", "{#SERVICE}": "HTTPS"},
    {"{#PORT}": "8080", "{#SERVICE}": "Admin"}
  ]
}
```

Item prototype:

```
Key: net.tcp.service[tcp,,{#PORT}]
Name: {#SERVICE} service status
```

PARTE 3

Monitoramento de Serviços Web

HTTP Agent: Evolução do Simple Check

HTTP Agent vs Simple Check:

Recurso	Simple Check	HTTP Agent
Métodos HTTP	Apenas GET	GET, POST, PUT, DELETE, HEAD
Headers customizados	❌	✅
Authentication	Basic apenas	Basic, Bearer, NTLM, Digest
Body/Payload	❌	✅
SSL verification	Simples	Avançado
Parsing JSON	❌	✅ (com preprocessing)

Recomendação Zabbix 7.0: Usar HTTP Agent

HTTP Agent: Verificação Básica

Verificar se site está online:

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

Name: Website availability
Type: HTTP agent
URL: https://www.example.com
Request type: GET
Timeout: 3s
Request headers:
  User-Agent: Zabbix-Monitor/7.0
Type of information: Text
Update interval: 1m
```

Retorno: HTML completo da página

HTTP Agent: Status Code

Extrair apenas HTTP status code:

```
Name: Website HTTP status code
Type: HTTP agent
URL: https://www.example.com
Request type: HEAD
Retrieve mode: Headers
Type of information: Text
Update interval: 1m

Preprocessing:
  1. Regular expression: HTTP/\d\.\d\s(\d{3}) → \1
  2. Discard unchanged with heartbeat: 1h
```

Retorno: `200`, `404`, `500`, etc.

Trigger: HTTP Status Code

Detectar erros HTTP:

```
Name: Website returned HTTP error
Expression: last(/host/web.status.code)<>200
Severity: High
Description: HTTP status code: {ITEM.LASTVALUE}
```

Trigger específica para 5xx (server errors):

```
Name: Website server error (5xx)
Expression: last(/host/web.status.code)>=500 and last(/host/web.status.code)<600
Severity: High
```

HTTP Agent: Response Time

Medir tempo de resposta HTTP:

```
Name: Website response time
Type: HTTP agent
URL: https://www.example.com
Request type: HEAD
Retrieve mode: Headers
Type of information: Numeric (float)
Units: s
Update interval: 1m

Preprocessing:
  1. Check for error in JSON → (vazio)
  2. JavaScript: return value.replace(/.*/, Zabbix.getResponseTime())
```

Alternativa simples: Usar item `web.page.perf`

HTTP Agent: Verificar Conteúdo

Garantir que página tem conteúdo esperado:

```
Name: Website content check
Type: HTTP agent
URL: https://www.example.com/health
Request type: GET
Type of information: Text
Update interval: 1m

Preprocessing:
  1. Regular expression: "status":"(.*?)" → \1
```

Trigger:

```
Name: Website health check failed
Expression: find(/host/web.content.check,,"like","ok")=0
Severity: High
```

HTTP Agent: JSON API Check

Monitorar API REST:

```
Name: API health status
Type: HTTP agent
URL: https://api.example.com/v1/health
Request type: GET
Request headers:
  Authorization: Bearer {$API_TOKEN}
  Content-Type: application/json
Type of information: Text
Update interval: 30s

Preprocessing:
  1. JSONPath: $.status
```

Retorno: `healthy`, `degraded`, `down`

HTTP Agent: POST com Payload

Autenticar em API e verificar resposta:

```
Name: API login check
Type: HTTP agent
URL: https://api.example.com/auth/login
Request type: POST
Request body:
  {"username":"monitor","password":"{$API_PASSWORD}"}
Request headers:
  Content-Type: application/json
Type of information: Text

Preprocessing:
  1. JSONPath: $.token
  2. JavaScript: return (value.length > 0) ? 1 : 0
```

Retorno: `1` (sucesso) ou `0` (falha)

Monitoramento de Certificado SSL

Verificar validade do certificado:

```
#!/bin/bash
# /usr/local/bin/check_ssl_expiry.sh
DOMAIN=$1

# Obter data de expiração
EXPIRY_DATE=$(echo | openssl s_client -servername $DOMAIN -connect $DOMAIN:443 2>/dev/null | openssl x509 -noout -enddate | cut -d= -f2)

# Converter para timestamp Unix
EXPIRY_TIMESTAMP=$(date -d "$EXPIRY_DATE" +%s)
NOW=$(date +%s)

# Calcular dias restantes
DAYS_LEFT=$(( ($EXPIRY_TIMESTAMP - $NOW) / 86400 ))

echo $DAYS_LEFT
```

SSL Certificate: UserParameter

```
UserParameter=custom.ssl.expiry[*],/usr/local/bin/check_ssl_expiry.sh "$1"
```

Item:

```
Name: SSL certificate days until expiry
Type: Zabbix agent
Key: custom.ssl.expiry[example.com]
Type of information: Numeric (unsigned)
Units: days
Update interval: 1d
```

Trigger:

```
Name: SSL certificate expires soon
Expression: last(/host/custom.ssl.expiry[example.com])<30
Severity: Warning
```

Web Scenarios (Web Monitoring)

Cenário multi-step (ex: login flow):

```
Data collection → Hosts → Web → Create web scenario

Name: User login flow
Update interval: 5m
Agent: Zabbix/7.0
Variables:
  {username} = testuser
  {password} = {$WEB_PASSWORD}

Steps:
  1. Homepage: GET https://example.com/
  2. Login: POST https://example.com/login
     Body: username={username}&password={password}
  3. Dashboard: GET https://example.com/dashboard
```

Web Scenario: Validações

Para cada step:

```
Step 1: Homepage
  Required string: "Welcome"
  Required status codes: 200
  Timeout: 15s

Step 2: Login
  Required string: "Login successful"
  Required status codes: 302,200

Step 3: Dashboard
  Required string: "Dashboard"
  Required status codes: 200
```

Triggers automáticas:

- Failed step: Severity Average
- Download speed slow: Severity Warning

PARTE 4

Triggers de Rede Avançados

Flapping: O Problema

Flapping = Link oscilando entre UP e DOWN rapidamente

```
10:00 → DOWN
10:01 → UP
10:02 → DOWN
10:03 → UP
...
```

Consequências:

- 🔴 Flood de notificações
- 🔴 Actions disparando repetidamente
- 🔴 Dificulta identificar problema real

Evitando Flapping: Hysteresis

Trigger sem proteção:

```
Expression: last(/host/icmpping)=0
```

Trigger com hysteresis:

```
Name: Host is down
Problem expression: max(/host/icmpping,5m)=0
Recovery expression: min(/host/icmpping,2m)=1
Allow manual close: No
```

Significado:

- Problema: Host não respondeu NENHUM ping em 5 min
- Recovery: Host respondeu TODOS os pings em 2 min

Trigger de Degradação Progressiva

Níveis de severidade baseados na duração:

```
Trigger 1: Link degraded (Warning)
Expression: max(/host/icmpping,3m)=0

Trigger 2: Link down (Average)
Expression: max(/host/icmpping,10m)=0
Dependency: Link degraded

Trigger 3: Link down critical (High)
Expression: max(/host/icmpping,30m)=0
Dependency: Link down
```

Vantagem: Severidade aumenta com o tempo

Trigger: Latência com Threshold Duplo

Warning e Critical:

```
Trigger 1: Network latency high
Expression: avg(/host/icmppingsec,10m)>0.100
Severity: Warning

Trigger 2: Network latency critical
Expression: avg(/host/icmppingsec,10m)>0.250
Severity: High
Dependency: Network latency high
```

Uso de avg() em vez de min():

- Menos sensível a spikes isolados
- Detecta degradação sustentada

Trigger: Packet Loss com Correlação

Problema: Packet loss pode ser transitório

Solução: Correlacionar com latência

```
Name: Network quality degraded
Expression:
  avg(/host/icmppingloss,10m)>2
  and
  avg(/host/icmppingsec,10m)>0.150
Severity: Average
Description: Packet loss > 2% AND latência > 150ms
```

Reduz falsos positivos

Trigger de Mudança de Estado

Detectar quando link volta:

```
Name: Network link recovered
Expression:
  last(/host/icmpping,#1)=1
  and
  last(/host/icmpping,#2)=0
Severity: Information
Manual close: Yes
```

Uso:

- Notificar equipe que problema foi resolvido
- Auditar instabilidades frequentes

Trigger: Link Instável (Flapping Detection)

Detectar múltiplas mudanças de estado:

```
Name: Network link is unstable (flapping)
Expression:
  (last(/host/icmpping,#1)<>last(/host/icmpping,#2))
  or
  (last(/host/icmpping,#2)<>last(/host/icmpping,#3))
  or
  (last(/host/icmpping,#3)<>last(/host/icmpping,#4))
Severity: Average
Description: Link mudou de estado 3+ vezes recentemente
```

Trigger com Múltiplos Hosts (Dependência)

Cenário: Servidor web depende de gateway

```
Trigger 1 (Gateway):
Name: Gateway is unreachable
Host: gateway.example.com
Expression: max(/gateway/icmpping,5m)=0
Severity: Disaster

Trigger 2 (Web Server):
Name: Web server is unreachable
Host: web01.example.com
Expression: max(/web01/icmpping,5m)=0
Severity: High
Dependencies: Gateway is unreachable
```

Vantagem: Suprime alerta do web server se gateway está down

PARTE 5

SLA e Métricas de Disponibilidade

O Que É SLA?

SLA (Service Level Agreement)

- Acordo de nível de serviço
- Define disponibilidade mínima esperada
- Geralmente expresso em % (ex: 99.9%)

Tabela de SLA:

SLA	Downtime/ano	Downtime/mês	Downtime/semana
90%	36.5 dias	3 dias	16.8 horas
99%	3.65 dias	7.2 horas	1.68 horas
99.9%	8.76 horas	43.2 min	10.1 min
99.99%	52.6 min	4.32 min	1.01 min
99.999%	5.26 min	25.9 seg	6.05 seg

Services no Zabbix 7.0 LTS

Services = Representação lógica de um serviço de negócio

```
Services → Create service

Name: E-commerce Website
Parent: (nenhum)
SLA: 99.9%
Status calculation rule: Most critical of child services
Problem tags:
  service: ecommerce
```

Child services:

- Frontend (nginx)
- API Backend (app servers)
- Database (MySQL cluster)

Configurando Service: Frontend

```
Services → Create service (child)

Name: E-commerce Frontend
Parent: E-commerce Website
Status calculation rule: Most critical if all children have problems
Problem tags:
  component: frontend
  service: ecommerce

Tags for problem matching:
  service: ecommerce
  component: frontend
```

Link com triggers: Triggers com tag `service: ecommerce` afetam este service

Service: Status Calculation Rules

Rule	Descrição	Uso
Most critical if all children have problems	Status = worst child se TODOS tiverem problema	Redundância (N servidores)
Most critical of child services	Status = worst child sempre	Dependência (todos críticos)
Set status to OK	Sempre OK	Service informativo

Exemplo: Cluster de 3 web servers

- Rule: "Most critical if all children have problems"
- Se 1 de 3 down → Service OK (ainda há 2)
- Se 3 de 3 down → Service CRITICAL

SLA Reporting

Visualizar SLA period:

```
Services → [Seu service] → SLA

Period: Last 30 days
SLA: 99.9%
Uptime: 99.87%  ← Abaixo do SLA!
Downtime: 56 minutes
Number of downtimes: 3
```

Exportar relatório:

- Formato: PDF, CSV
- Agendamento: Semanal, mensal

Calculated SLA: Item Customizado

Calcular SLA manualmente com item:

```
#!/bin/bash
# /usr/local/bin/calculate_sla.sh
ITEM_ID=$1
DAYS=$2

# Consultar histórico do Zabbix DB
mysql -u zabbix -p'password' zabbix -N -e "
SELECT (SUM(CASE WHEN value=1 THEN 1 ELSE 0 END) / COUNT(*)) * 100
FROM history_uint
WHERE itemid=$ITEM_ID
  AND clock > UNIX_TIMESTAMP(NOW() - INTERVAL $DAYS DAY)
"
```

```
UserParameter=custom.sla[*],/usr/local/bin/calculate_sla.sh "$1" "$2"
```

Trigger: SLA Breach

Alertar quando SLA é violado:

```
Name: SLA breach - Service availability below 99.9%
Expression:
  (sum(/host/net.tcp.service[http,,80],30d) / count(/host/net.tcp.service[http,,80],30d)) < 0.999
Severity: High
Description: Service availability nos últimos 30 dias: {ITEM.LASTVALUE}%
```

⚠️ **Cuidado:** Query pesada, usar intervalo longo (1d)

PARTE 6

Laboratórios Práticos

Lab 1: ICMP Monitoring Completo

Objetivo: Monitorar roteador/gateway via ICMP

Passos:

1. Criar host para gateway:

```
Data collection → Hosts → Create host
Host name: gateway.local
Interface: IP 192.168.1.1 (sem agent)
```

2. Criar 3 items ICMP:
   - `icmpping` (reachability)
   - `icmppingsec` (latency)
   - `icmppingloss` (packet loss)

Lab 1: Triggers

3. Criar triggers:

```
Trigger 1: Gateway unreachable
Expression: max(/gateway/icmpping,5m)=0
Severity: Disaster

Trigger 2: Gateway high latency
Expression: avg(/gateway/icmppingsec,10m)>0.050
Severity: Warning

Trigger 3: Gateway packet loss
Expression: avg(/gateway/icmppingloss,10m)>1
Severity: Warning
```

Lab 1: Teste e Validação

4. Testar desconexão:

```
# Simular gateway down (no próprio gateway)
sudo iptables -A INPUT -p icmp --icmp-type echo-request -j DROP

# Aguardar 5 minutos
# Verificar trigger "Gateway unreachable" dispara

# Restaurar
sudo iptables -D INPUT -p icmp --icmp-type echo-request -j DROP
```

5. Verificar recovery automático

Lab 2: TCP Service Monitoring

Objetivo: Monitorar serviços essenciais de um servidor

1. Criar items:

```
Item 1:
  Name: SSH service check
  Key: net.tcp.service[ssh,,22]

Item 2:
  Name: HTTP service check
  Key: net.tcp.service[http,,80]

Item 3:
  Name: MySQL service check
  Key: net.tcp.service[tcp,,3306]
```

Lab 2: Triggers e Teste

2. Criar triggers:

```
Name: {#SERVICE} is down
Expression: max(/host/net.tcp.service[{#SERVICE}],3m)=0
Severity: High
```

3. Testar:

```
# Parar serviço
sudo systemctl stop nginx

# Verificar item net.tcp.service[http,,80] → 0
# Aguardar 3 minutos
# Verificar trigger dispara

# Restaurar
sudo systemctl start nginx
```

Lab 3: HTTP Agent - API Monitoring

Objetivo: Monitorar API pública e extrair métricas JSON

1. Criar item HTTP agent:

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

Name: GitHub Status API
Key: github.status.api
Type: HTTP agent
URL: https://www.githubstatus.com/api/v2/status.json
Request type: GET
Request headers:
  Accept: application/json
  User-Agent: Zabbix-Monitor
Type of information: Text
Update interval: 5m
History storage period: 7d
```

Resposta esperada:

```
{
  "page": { "id": "...", "name": "GitHub", "url": "..." },
  "status": {
    "indicator": "none",
    "description": "All Systems Operational"
  }
}
```

Lab 3: Parsing JSON

2. Criar dependent items com JSONPath:

```
Item 1 (dependent):
  Name: GitHub Status - Indicator
  Key: github.status.indicator
  Type: Dependent item
  Master item: GitHub Status API
  Type of information: Text

  Preprocessing:
    1. JSONPath: $.status.indicator

  History storage period: 30d

Item 2 (dependent):
  Name: GitHub Status - Description
  Key: github.status.description
  Type: Dependent item
  Master item: GitHub Status API
  Type of information: Text

  Preprocessing:
    1. JSONPath: $.status.description

  History storage period: 30d
```

Lab 3: Trigger

3. Criar trigger:

```
Configuration → Hosts → [Seu host] → Triggers → Create trigger

Name: GitHub Status is degraded or down
Expression:
  find(/[hostname]/github.status.indicator,,"like","none")=0
Severity: Warning
Description: GitHub Status is not operational
  Current indicator: {ITEM.LASTVALUE1}
  Description: {ITEM.LASTVALUE2:github.status.description}
```

Valores possíveis do indicator: `none` , `minor` , `major` , `critical`

Como funciona:

- `find()` procura "none" no item
- Se retornar 0 = NÃO encontrou "none" = API com problema
- Trigger dispara quando status ≠ "none"

Lab 3: Testar

4. Verificar coleta:

```
# Testar API manualmente
curl https://www.githubstatus.com/api/v2/status.json

# Verificar no Zabbix
Monitoring → Latest data → Filter: "GitHub Status"
```

Resultado esperado:

- Master item: JSON completo
- Dependent item 1: "none"
- Dependent item 2: "All Systems Operational"

Lab 4: Web Scenario - Login Flow

Objetivo: Monitorar fluxo de login de aplicação web

1. Criar web scenario:

```
Data collection → Hosts → Web → Create web scenario

Name: Admin login flow
Application: Web monitoring
Update interval: 10m
```

Lab 4: SSL Certificate Expiry

Objetivo: Alertar sobre certificados SSL expirando

1. Criar script:

```
#!/bin/bash
# /usr/local/bin/check_ssl_expiry.sh
DOMAIN=$1
EXPIRY=$(echo | openssl s_client -servername $DOMAIN -connect $DOMAIN:443 2>/dev/null | openssl x509 -noout -enddate | cut -d= -f2)
EXPIRY_TS=$(date -d "$EXPIRY" +%s)
NOW=$(date +%s)
echo $(( ($EXPIRY_TS - $NOW) / 86400 ))
```

2. Tornar executável:

```
sudo chmod +x /usr/local/bin/check_ssl_expiry.sh
sudo chown zabbix:zabbix /usr/local/bin/check_ssl_expiry.sh
```

Lab 4: UserParameter e Item

3. Criar UserParameter:

```
# /etc/zabbix/zabbix_agentd.d/ssl_check.conf
UserParameter=custom.ssl.expiry[*],/usr/local/bin/check_ssl_expiry.sh "$1"
```

4. Reiniciar agent:

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

5. Criar item:

```
Name: SSL cert expiry for example.com
Key: custom.ssl.expiry[example.com]
Units: days
Update interval: 12h
```

Lab 4: Triggers

6. Criar triggers:

```
Trigger 1: SSL certificate expires in 30 days
Expression: last(/host/custom.ssl.expiry[example.com])<30
Severity: Warning

Trigger 2: SSL certificate expires in 7 days
Expression: last(/host/custom.ssl.expiry[example.com])<7
Severity: High
Dependencies: SSL certificate expires in 30 days
```

Lab 5: SLA Reporting

Objetivo: Configurar SLA de 99.9% para serviço web

⚠️ **Pré-requisito:** Ter pelo menos 1 trigger configurada com tag

Lab 5: Passo 1 - Criar Service

Caminho:

```
Monitoring → Services
→ Click no botão "Edit" (canto superior direito)
→ Click com botão direito no espaço vazio (ou na área "Root")
→ Selecione "Add" → "Add child service"
```

OU (se já existe algum service):

```
Monitoring → Services
→ Botão direito em qualquer service existente
→ "Add child service"
```

Configuração do service:

```
Name: Production Website
Parent: Root
Status propagation rule: Increase by → Set status to OK
Additional rules: (deixar vazio)
Description: Website de produção

Aba "Tags":
  Click "Add"
  Tag name: service
  Value: website
```

[Add] para salvar

Explicação: Service vai monitorar problemas que tenham tag `service=website`

Lab 5: Passo 2 - Configurar SLA

⚠️ **MUDANÇA IMPORTANTE:** SLA agora é separado de Service!

```
Services → SLA → Create SLA

Name: Website SLA 99.9%
SLA: 99.9
Schedule: 24x7
Service: Production Website  ← Selecione o service criado

[Add]
```

Explicação: SLA define que o service "Production Website" deve ter 99.9% de uptime

Lab 5: Criar/Ajustar Trigger com Tag

2. Adicionar tag à trigger existente:

```
Data collection → Hosts → [Seu host web] → Triggers

Selecionar trigger (ex: "Website is down" ou "HTTP service is down")
→ Edit

Aba "Tags":
  Click em "Add"
  Tag: service
  Value: website

[Update]
```

Verificar se trigger tem a tag:

```
Monitoring → Problems → [Clique na trigger]
→ Deve mostrar tag "service: website"
```

Lab 5: Verificar Associação

3. Confirmar que service está vinculado:

```
Services → Services → Production Website → Action → Edit

Verificar aba "Tags for problem matching":
  ✅ service = website

Services → Services → Production Website

Clicar no service criado:
  → Deve mostrar status (OK/Problem)
  → Se houver problema com tag "service: website", aparece aqui
```

Teste: Provocar o problema (parar nginx) e verificar se aparece no service

Lab 5: Visualizar SLA Report

4. Aguardar alguns dados (opcional: simular downtime):

Simular downtime para gerar dados de SLA:

```
# Parar serviço por 5 minutos
sudo systemctl stop nginx

# Aguardar trigger disparar (pode levar 1-3 min)

# Iniciar serviço novamente
sudo systemctl start nginx
```

5. Acessar SLA report:

```
Services → Services → Production Website → SLA report

Configurações:
  Period: Last 7 days  ← Começar com período curto
  Show service level: Yes

[Apply]
```

Lab 5: Interpretar SLA Report

6. Entender o relatório:

Colunas exibidas:

- **SLA:** 99.9% (configurado)
- **SLI (Service Level Indicator):** % real de uptime
- **Uptime:** Tempo que serviço ficou OK
- **Downtime:** Tempo que serviço ficou com problema
- **Error budget:** Quanto de downtime ainda é permitido

Exemplo:

```
Period: Last 7 days (168 hours)
SLA: 99.9%
SLI: 99.85%  ← Real
Uptime: 167h 44m
Downtime: 16m  ← Teve 16 minutos de problema
Status: 🔴 SLA breach (uptime < SLA)
```

Exportar relatório:

- Click em "Export to PDF" ou "Export to CSV"

Lab 5: Troubleshooting

Problema 1: Service não mostra problemas

Causa: Tag não está correta

Solução:

1. Verificar tag na trigger: `service=website`
2. Verificar tag no service: `service=website` (sem espaços!)
3. Tags são **case-sensitive**!

Problema 2: SLA report vazio

Causa: Não há dados de histórico ainda

Solução:

1. Aguardar pelo menos 1 hora com service ativo
2. Simular um problema (parar serviço 2-3 min)
3. Usar período curto: "Last 24 hours"

Problema 3: Não consigo criar service

Causa: Permissão insuficiente

Solução:

- Usuário precisa ter role com permissão "Manage services"
- Administration → User roles → [Seu role] → Services
  - [x] Read-write

PARTE 7

Troubleshooting e Best Practices

Problema 1: ICMP Não Funciona

Sintoma: Item icmpping fica "Not supported"

Causa: Zabbix Server sem permissão para raw sockets

Solução:

```
# Verificar permissões atuais
getcap /usr/sbin/zabbix_server

# Se vazio, adicionar permissão
sudo setcap cap_net_raw+ep /usr/sbin/zabbix_server

# Reiniciar
sudo systemctl restart zabbix-server

# Verificar novamente
getcap /usr/sbin/zabbix_server
# Output: /usr/sbin/zabbix_server = cap_net_raw+ep
```

Problema 2: HTTP Agent SSL Error

Sintoma: "SSL certificate problem: unable to get local issuer certificate"

Causa: Certificado SSL auto-assinado ou CA desconhecida

Solução 1: Desabilitar SSL verify (DEV APENAS):

```
HTTP agent item:
  SSL verify peer: No
  SSL verify host: No
```

Solução 2: Adicionar CA (PRODUÇÃO):

```
# Baixar certificado CA
openssl s_client -connect example.com:443 -showcerts > cert.pem

# Adicionar ao trust store
sudo cp cert.pem /usr/local/share/ca-certificates/custom.crt
sudo update-ca-certificates
```

Problema 3: Web Scenario Falha

Sintoma: Step 2 (login) sempre falha

Debug:

1. **Verificar POST fields:**

```
Ver Developer Tools do browser (F12)
Network → Login request → Form Data
Copiar exatamente os field names
```

2. **Verificar CSRF token:**

```
Se aplicação usa CSRF:
Step 1: Extrair token com regex
Step 2: Usar token no POST
```

3. **Session cookies:**

```
Web scenario mantém cookies entre steps automaticamente
Se não funciona: verificar cookie path/domain
```

Best Practice 1: Intervalo de Coleta

Recomendações:

Tipo de Check	Intervalo	Justificativa
ICMP ping	1m	Detectar downtime rapidamente
TCP service	1m	Serviços críticos
HTTP agent	1-5m	Depende da criticidade
SNMP traffic	1m	Granularidade para gráficos
SNMP status	5m	Muda raramente
Web scenario	5-15m	Pode ser pesado
SSL expiry	12-24h	Muda lentamente

Best Practice 2: Tagging

Usar tags consistentes:

```
Host tags:
  class: network
  type: switch
  location: datacenter-1
  criticality: high

Item tags:
  component: interface
  interface_type: uplink

Trigger tags:
  service: ecommerce
  severity: critical
```

Vantagens:

- Filtros em dashboards
- Service association automática
- Relatórios segmentados

Best Practice 3: Hysteresis

SEMPRE usar hysteresis para network triggers:

❌ Ruim:

```
Expression: last(/host/icmpping)=0
```

✅ Bom:

```
Problem: max(/host/icmpping,5m)=0
Recovery: min(/host/icmpping,2m)=1
```

Evita:

- Flapping alerts
- Notification storm
- Alert fatigue

Best Practice 4: Trigger Dependencies

Evitar duplicate alerts:

```
Gateway (upstream):
  Trigger: Gateway unreachable
  Severity: Disaster
  Dependencies: (nenhuma)

Servers (downstream):
  Trigger: Server unreachable
  Severity: High
  Dependencies: Gateway unreachable ← IMPORTANTE!
```

Efeito: Se gateway cai, suprime alertas dos 50 servers downstream

Best Practice 5: Calculated Items

Reduzir carga no agent:

❌ **Ruim:** Script faz cálculo

```
#!/bin/bash
TOTAL=$(cat /proc/meminfo | grep MemTotal | awk '{print $2}')
FREE=$(cat /proc/meminfo | grep MemFree | awk '{print $2}')
echo $(( ($TOTAL - $FREE) * 100 / $TOTAL ))
```

✅ **Bom:** Server faz cálculo

```
Item 1 (agent): vm.memory.size[total]
Item 2 (agent): vm.memory.size[free]
Item 3 (calculated): 100 * (last(//total) - last(//free)) / last(//total)
```

Best Practice 6: Templates

Criar templates reutilizáveis:

```
Template: App - Web Server Standard
Items:
  - HTTP service check (port 80)
  - HTTPS service check (port 443)
  - SSL certificate expiry
  - HTTP response time
Triggers:
  - Service down
  - SSL expiring
  - Response time high
Macros:
  {$HTTP_PORT} = 80
  {$HTTPS_PORT} = 443
```

Aplicar em 100 web servers: 1 clique

Best Practice 7: Macros

Usar macros para valores variáveis:

```
Global macros:
  {$SNMP_COMMUNITY} = public
  {$ICMP_LOSS_WARN} = 2
  {$ICMP_LATENCY_WARN} = 0.100

Template macros:
  {$HTTP_PORT} = 80

Host macros (override):
  {$HTTP_PORT} = 8080  ← Para esse host específico
```

Vantagem: Atualizar macro global afeta todos os hosts

Recursos Adicionais

Documentação oficial:

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

Templates da comunidade:

- https://git.zabbix.com/projects/ZBX/repos/zabbix/browse/templates
- https://www.zabbix.com/integrations

Revisão da Aula

Aprendemos:

1. ✅ ICMP monitoring: ping, latência, packet loss
2. ✅ TCP/UDP service checks com simple checks
3. ✅ HTTP Agent avançado: APIs, JSON, SSL
4. ✅ Web Scenarios para fluxos multi-step
5. ✅ Triggers anti-flapping e dependencies
6. ✅ SLA configuration e reporting
7. ✅ 7 laboratórios práticos hands-on

Próxima Aula

Aula 09: Zabbix Proxy e Monitoramento Distribuído

Tópicos:

- Quando e por que usar Zabbix Proxy
- Instalação e configuração
- Active vs Passive proxy
- Sincronização de dados
- Estratégias para filiais e ambientes segregados
- Troubleshooting de proxies

Perguntas?

Obrigado!

4Linux - Zabbix Advanced Course