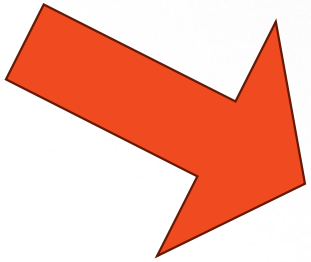


ZABBIX

Migrating Nagios to Zabbix: Lessons Learned

Opensource ICT Solutions

Who am I?

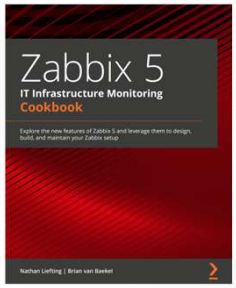
Nathan Liefting
Zabbix Consultant / Trainer

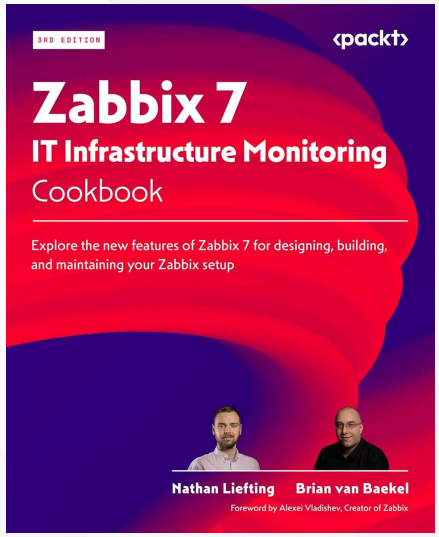
Opensource ICT Solutions

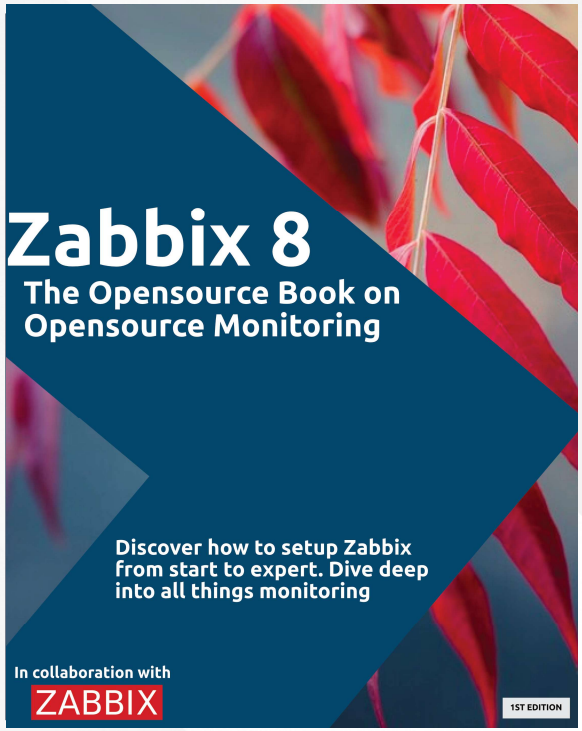
ZABBIX PREMIUM PARTNER

ZABBIX CERTIFIED TRAINER

- Zabbix support
- Zabbix training
- Zabbix consultancy
- And more…







https://www.linkedin.com/company/opensource-ict-solutions/

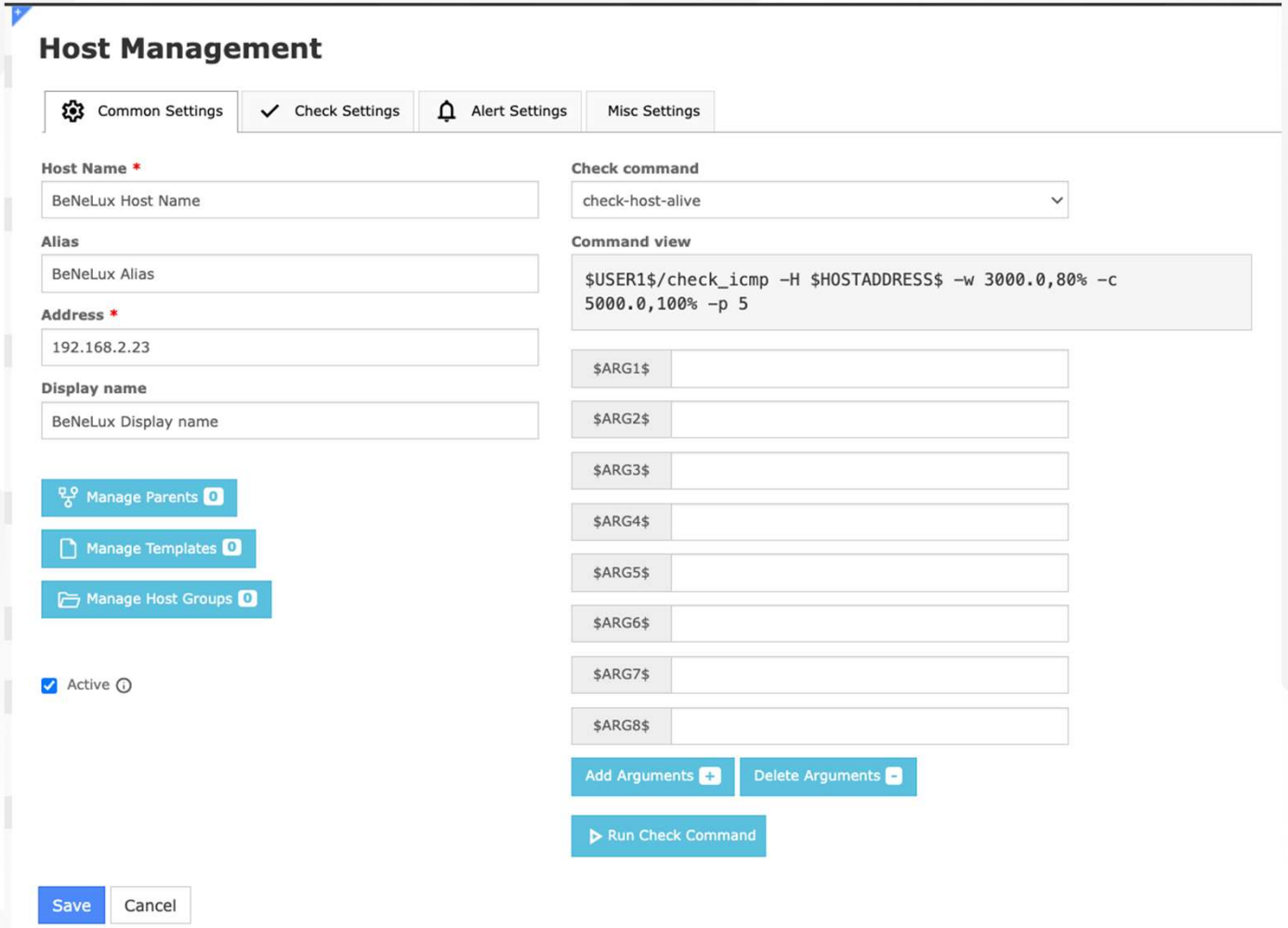
The story

- Cannot lose data
- Nagios XI has to be disabled entirely
- Historic performance data migrated to Zabbix
- Existing problems migrated from Nagios as well

Nagios hosts VS Zabbix hosts

- Nagios has hosts just like Zabbix

- Three names
  - Host Name
  - Alias = Host name
  - Display Name = Visible name



Opensource ICT Solutions

Nagios commands VS Zabbix items

- Nagios executes checks in a very different manner
- Host level check

- Service level check

Opensource ICT Solutions

Where does Nagios store data

- /usr/local/nagios/share/perfdata/BeNeLux-Host-Name/
  - ../Availability.rrd
  - ../Availability.xml

```
-rw-r--r-- 1 root root 384952 Feb  4 10:50 Availability.rrd
-rw-r--r-- 1 root root   2262 Feb  4 10:49 Availability.xml
```

- XML stores info used in Item creation
- RRD is a round-robin database file used for storing history data

- First, let's look at the XMl

- Then, let's look at a month worth of data:

rrdtool /usr/local/nagios/share/perfdata/BeNeLux-Host-Name/Availability.rrd LAST --start -30d --end now | grep -v "nan"

Where does Nagios store data - RRD

- /usr/local/nagios/share/perfdata/BeNeLux-Host-Name/Availability.rrd
- rrdtool /usr/local/nagios/share/perfdata/BeNeLux-Host-Name/Availability.rrd LAST --start -30d --end now | grep -v "nan"

```
                          1             2             3             4             5             6             7

1684669080: 7.9767186667e-02 1.5900000000e+02 1.7609813333e-02 2.0437533333e-02 5.9866666667e-06 4.1574226667e-02 4.1595200000e-02
1684669140: 7.9767186667e-02 1.5900000000e+02 1.7609813333e-02 2.0437533333e-02 5.9866666667e-06 4.1574226667e-02 4.1595200000e-02
1684669200: 7.9767186667e-02 1.5900000000e+02 1.7609813333e-02 2.0437533333e-02 5.9866666667e-06 4.1574226667e-02 4.1595200000e-02
1684669260: 7.9767186667e-02 1.5900000000e+02 1.7609813333e-02 2.0437533333e-02 5.9866666667e-06 4.1574226667e-02 4.1595200000e-02
1684669320: 7.9767186667e-02 1.5900000000e+02 1.7609813333e-02 2.0437533333e-02 5.9866666667e-06 4.1574226667e-02 4.1595200000e-02
1684669380: 7.5095020000e-02 1.5900000000e+02 1.6104420000e-02 1.9828380000e-02 6.0000000000e-06 3.9027110000e-02 3.9050090000e-02
1684669440: 7.5095020000e-02 1.5900000000e+02 1.6104420000e-02 1.9828380000e-02 6.0000000000e-06 3.9027110000e-02 3.9050090000e-02
1684669500: 7.5095020000e-02 1.5900000000e+02 1.6104420000e-02 1.9828380000e-02 6.0000000000e-06 3.9027110000e-02 3.9050090000e-02
1684669560: 7.5095020000e-02 1.5900000000e+02 1.6104420000e-02 1.9828380000e-02 6.0000000000e-06 3.9027110000e-02 3.9050090000e-02
1684669620: 7.5095020000e-02 1.5900000000e+02 1.6104420000e-02 1.9828380000e-02 6.0000000000e-06 3.9027110000e-02 3.9050090000e-02
1684669680: 7.4692786667e-02 1.5900000000e+02 1.5571000000e-02 1.8171306667e-02 6.9866666667e-06 4.0811533333e-02 4.0833546667e-02
1684669740: 7.4692786667e-02 1.5900000000e+02 1.5571000000e-02 1.8171306667e-02 6.9866666667e-06 4.0811533333e-02 4.0833546667e-02
1684669800: 7.4692786667e-02 1.5900000000e+02 1.5571000000e-02 1.8171306667e-02 6.9866666667e-06 4.0811533333e-02 4.0833546667e-02
1684669860: 7.4692786667e-02 1.5900000000e+02 1.5571000000e-02 1.8171306667e-02 6.9866666667e-06 4.0811533333e-02 4.0833546667e-02
1684669920: 7.4692786667e-02 1.5900000000e+02 1.5571000000e-02 1.8171306667e-02 6.9866666667e-06 4.0811533333e-02 4.0833546667e-02
```

Unixtime XML #1 XML #2 XML #3 XML #4 XML #5 XML #6 XML #7

Where does Nagios store data - XML

- /usr/local/nagios/share/perfdata/BeNeLux-Host-Name/Availability.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<NAGIOS>
  <DATASOURCE>
    <TEMPLATE>check_xi_service_tcp</TEMPLATE>
    <RRDFILE>/usr/local/nagios/share/perfdata/BeNeLux-Host-Name/Availability.xml</RRDFILE>
    <RRD_STORAGE_TYPE>SINGLE</RRD_STORAGE_TYPE>
    <RRD_HEARTBEAT>8460</RRD_HEARTBEAT>
    <IS_MULTI>0</IS_MULTI>
    <DS>1</DS>
    <NAME>time</NAME>
    <LABEL>time</LABEL>
    <UNIT>s</UNIT>
    <ACT>0.013125</ACT>
    <WARN></WARN>
    <WARN_MIN></WARN_MIN>
    <WARN_MAX></WARN_MAX>
    <WARN_RANGE_TYPE></WARN_RANGE_TYPE>
    <CRIT></CRIT>
    <CRIT_MIN></CRIT_MIN>
    <CRIT_MAX></CRIT_MAX>
    <CRIT_RANGE_TYPE></CRIT_RANGE_TYPE>
    <MIN>0.000000</MIN>
    <MAX>10.000000</MAX>
  </DATASOURCE>
  <RRD>
    <RC>0</RC>
    <TXT>successful updated</TXT>
  </RRD>
```

```
<NAGIOS_AUTH_HOSTNAME>BeNeLux Host Name</NAGIOS_AUTH_HOSTNAME>
<NAGIOS_AUTH_SERVICEDESC>Availability</NAGIOS_AUTH_SERVICEDESC>
<NAGIOS_CHECK_COMMAND>check_xi_service_tcp!-p 443!!!!!!!-f follow -S -I benelux.net</NAGIOS_CHECK_COMMAND>
<NAGIOS_DATATYPE>SERVICEPERFDATA</NAGIOS_DATATYPE>
<NAGIOS_DISP_HOSTNAME>BeNeLux Display name</NAGIOS_DISP_HOSTNAME>
<NAGIOS_DISP_SERVICEDESC>Availability</NAGIOS_DISP_SERVICEDESC>
<NAGIOS_HOSTNAME>BeNeLux Host Name</NAGIOS_HOSTNAME>
<NAGIOS_HOSTSTATE>UP</NAGIOS_HOSTSTATE>
<NAGIOS_HOSTSTATETYPE>HARD</NAGIOS_HOSTSTATETYPE>
<NAGIOS_MULTI_PARENT></NAGIOS_MULTI_PARENT>
<NAGIOS_PERFDATA>time=0.013125s;;;0.000000;10.000000 </NAGIOS_PERFDATA>
<NAGIOS_RRDFILE>/usr/local/nagios/share/perfdata/BeNeLux-Host-Name/Availability.rrd</NAGIOS_RRDFILE>
<NAGIOS_SERVICECHECKCOMMAND>check_xi_service_tcp!-p 443!!!!!!!-f follow -S -I benelux.net"</NAGIOS_SERVICECHECKCOMMAND>
<NAGIOS_SERVICEDESC>Availability</NAGIOS_SERVICEDESC>
<NAGIOS_SERVICEOUTPUT>TCP OK - 0.013 second response time on 145.21.234.11 port 443</NAGIOS_SERVICEOUTPUT>
<NAGIOS_SERVICEPERFDATA>time=0.013125s;;;0.000000;10.000000</NAGIOS_SERVICEPERFDATA>
<NAGIOS_SERVICESTATE>OK</NAGIOS_SERVICESTATE>
<NAGIOS_SERVICESTATETYPE>HARD</NAGIOS_SERVICESTATETYPE>
<NAGIOS_TIMET>1728890104</NAGIOS_TIMET>
<NAGIOS_XMLFILE>/usr/local/nagios/share/perfdata/BeNeLux-Host-Name/Availability.xml</NAGIOS_XMLFILE>
<XML>
```

Let's get scripting

- We create Python import tool

First, we grab our Nagios hostnames and make sure to sanitize it all for Zabbix usage

```python
# Sanitize hostnames for Zabbix
def sanitize_hostname(hostname):
    return re.sub(r'[^\w.-]', '_', hostname)

# Sanitize names for matching
def sanitize_name_for_matching(name):
    return re.sub(r'[^\w]', '_', name).lower()
```

Then we create the host in Zabbix if it doesn't exist

```python
# Create or retrieve a Zabbix host
def create_zabbix_host(api_key, alias, display_name, address):
    sanitized_host = sanitize_hostname(alias)
    if display_name == 'Egem':
        sanitized_host = 'Egem_2' if alias == 'Egem' else sanitized_host
    existing_host_id = get_existing_host_id(api_key, sanitized_host)
    if existing_host_id:
        print(f"Host {sanitized_host} already exists with ID {existing_host_id}")
        return existing_host_id, sanitized_host
    headers = {'Content-Type': 'application/json-rpc'}
    data = {
        'jsonrpc': '2.0',
        'method': 'host.create',
        'params': {
            'host': sanitized_host,
            'name': display_name,
            'interfaces': [{
                'type': 2,
                'main': 1,
                'useip': 1 if is_valid_ip(address) else 0,
                'ip': address if is_valid_ip(address) else '',
                'dns': '' if is_valid_ip(address) else address,
                'port': '161',
                'details': {'version': 2, 'community': 'public', 'bulk': 1}
            }],
            'groups': [{'groupid': '2'}],
        },
        'auth': api_key,
        'id': 1
    }
    response = requests.post(ZABBIX_URL, headers=headers, json=data)
    result = response.json()
    if 'result' in result:
        return result['result']['hostids'][0], sanitized_host
    else:
        print(f"Error creating host: {result['error']}")
        return None, None
```

Let’s get scripting

We read the XML file and create our items.

Sanitizing is important, to create a somewhat usuable Zabbix configuration and avoid Syntax errors

```
# Sanitize XML filename for Zabbix item prefixing
def sanitize_xml_filename(filename):
    return re.sub(r'[\s_\[\]]', '.', os.path.splitext(filename)[0]).lower()

# Create a Zabbix item
def create_zabbix_item(api_key, host_id, item_name, key, units, xml_filename):
    # Prefix the item key with the sanitized XML filename
    prefix = sanitize_xml_filename(xml_filename)
    prefixed_key = f"{prefix}.{sanitize_key(key)}"
    prefixed_item_name = f"{prefix}.{item_name}"

    # Check if item already exists
    existing_item_id = get_existing_item_id(api_key, host_id, prefixed_key)
    if existing_item_id:
        print(f"Item {prefixed_key} already exists on host {host_id}")
        return existing_item_id

    headers = {'Content-Type': 'application/json-rpc'}
    data = {
        'jsonrpc': '2.0',
        'method': 'item.create',
        'params': {
            'name': prefixed_item_name,
            'key_': prefixed_key,
            'hostid': host_id,
            'type': 2,
            'value_type': 0,
            'units': units if units else '',
        },
        'auth': api_key,
        'id': 1
    }
```

Let’s get scripting

We can even create the triggers straight from the XML file

```
<ACT>0.013125</ACT>
<WARN></WARN>
<WARN_MIN></WARN_MIN>
<WARN_MAX></WARN_MAX>
<WARN_RANGE_TYPE></WARN_RANGE_TYPE>
<CRIT></CRIT>
<CRIT_MIN></CRIT_MIN>
<CRIT_MAX></CRIT_MAX>
<CRIT_RANGE_TYPE></CRIT_RANGE_TYPE>
<MIN>0.000000</MIN>
<MAX>10.000000</MAX>
```

Nagios XML

```
# Create a Zabbix trigger with different severity levels
def create_zabbix_trigger(api_key, host_id, item_key, threshold, severity, sanitized_host, xml_filename):
    # Prefix the item key with the sanitized XML filename
    prefix = sanitize_xml_filename(xml_filename)
    prefixed_item_key = f"{prefix}.{sanitize_key(item_key)}"

    headers = {'Content-Type': 'application/json-rpc'}
    expression = f"last(/" + sanitized_host + f"/{prefixed_item_key},#1)>{threshold}"
    severity_mapping = {'warning': 3, 'high': 4}
    data = {
        "jsonrpc": "2.0",
        "method": "trigger.create",
        "params": {
            "description": f"Trigger for {prefixed_item_key} - {severity} severity",
            "expression": expression,
            "priority": severity_mapping[severity]
        },
        "auth": api_key,
        "id": 1
    }
    response = requests.post(ZABBIX_URL, headers=headers, json=data)
    result = response.json()
    if 'result' in result:
        return result['result']['triggerids'][0]
    else:
        print(f"Error creating {severity} trigger for {prefixed_item_key}: {result['error']}")
        return None
```

Let’s get scripting

The most important part however is mathching the XML file data to RRD.

As we saw in the RRD, we need to match XML1 to RRD column 1

```
# Process RRD data and send to Zabbix
# Update to map RRD files to prefixed item keys and send data to Zabbix
def create_and_send_zabbix_file(hostname, rrd_file, datasources, xml_filename):
    # Map RRD file name to Zabbix key prefix
    rrd_filename = os.path.splitext(os.path.basename(rrd_file))[0]

    try:
        # Use the XML filename to determine the correct sender file
        sender_file_name = f"{sanitize_xml_filename(xml_filename)}.txt"
        host_dir = os.path.join(sender_files_dir, hostname)
        if not os.path.exists(host_dir):
            print(f"Creating directory: {host_dir}")
            os.makedirs(host_dir)

        # Prepare the sender file path
        sender_file_path = os.path.join(host_dir, sender_file_name)
        print(f"Preparing sender file: {sender_file_path}")

        # Fetch data from the RRD file
        rrd_data = subprocess.check_output(
            f'rrdtool fetch {rrd_file} LAST --start -30d --end now | grep -v "nan"',
            shell=True).decode('utf-8')

        if not rrd_data.strip():
            print(f"No data found in RRD file: {rrd_file}")
            return
```

Opensource ICT Solutions

Let’s get scripting

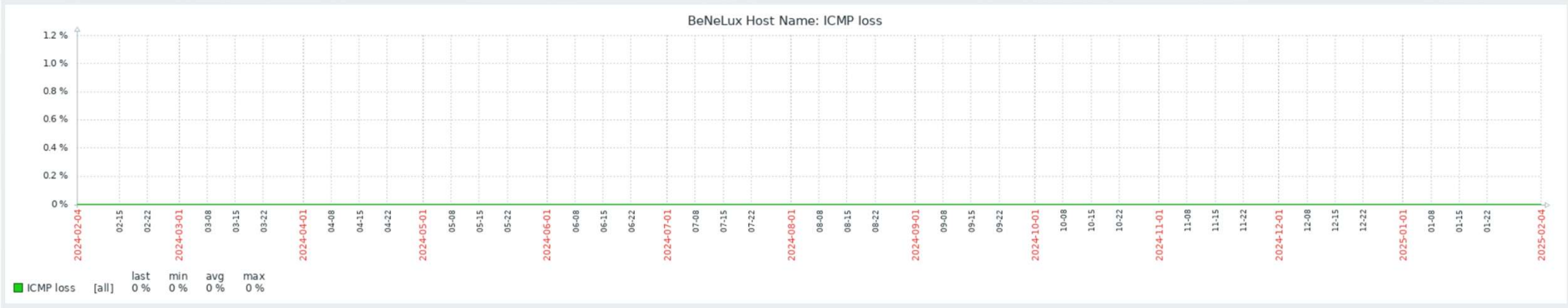
Execute script and data will be imported

For now the script makes sure the items are created as Zabbix Trapper

Name ▲	Triggers	Key	Interval	History	Trends	Type
Availability 1	Triggers 1	net.tcp.service.perf[tcp,benelux1.net,443]		31d	365d	Zabbix trapper
Availability 2	Triggers 1	net.tcp.service.perf[tcp,benelux2.net,443]		31d	365d	Zabbix trapper
Availability 3	Triggers 1	net.tcp.service.perf[tcp,benelux3.net,443]		31d	365d	Zabbix trapper
Availability 4	Triggers 1	net.tcp.service.perf[tcp,benelux4.net,443]		31d	365d	Zabbix trapper
Availability 5	Triggers 1	net.tcp.service.perf[tcp,benelux5.net,443]		31d	365d	Zabbix trapper
Availability 6	Triggers 1	net.tcp.service.perf[tcp,benelux6.net,443]		31d	365d	Zabbix trapper
ICMP loss	Triggers 1	icmppingloss		9124d	9124d	Zabbix trapper

Let’s get scripting

You will see perfect historic data



As well as historic problems are created

Time ▾	Severity	Recovery time	Status	Info	Host	Problem	Duration	Update	Actions	Tags
2024-11-14 12:26:26 PM	Average		PROBLEM		BeNeLux Host Name	No data or no response on Availability 1	2M 21d 23h	Update		

Why? Zabbix Sender data with timestamp

Opensource ICT Solutions

Let’s get scripting

Why? Zabbix Sender file is created:

```
-rw-r--r-- 1 root root 384952 Feb  4 10:50 Availability.rrd
-rw-r--r-- 1 root root   2262 Feb  4 11:01 Availability.xml
-rw-r--r-- 1 root root     67 Feb  4 12:22 _HOST_.sender
```

Data is sent with timestamp

```
cat _HOST_.sender
BeNeLux-Host-Name 1738671730 0 0
BeNeLux-Host-Name 1738671700 0 0
```

That means Zabbix knows when a problem started and when it stopped, as data passes through the normal process

Template and Item creation

Check command

check-host-alive

Command view

```
$USER1$/check_icmp -H $HOSTADDRESS$ -w 3000.0,80% -c
5000.0,100% -p 5
```

	Name ▲	Triggers	Key
•••	ICMP loss	Triggers 1	icmppingloss
•••	ICMP ping	Triggers 1	icmpping
•••	ICMP response time	Triggers 1	icmppingsec

Check command

check_xi_service_tcp

Command view

```
$USER1$/check_tcp -H $HOSTADDRESS$ $ARG1$
```

$ARG1$	-p 113 -r ok

	Name ▲	Triggers	Key
•••	{$TCP.NAME}	Triggers 1	net.tcp.service.perf

	Name ▼	Hosts	Items	Triggers
	ICMP Ping	Hosts 70	Items 3	Triggers 3
	check_xi_service_tcp	Hosts 40	Items 1	Triggers 1

Triggers

- With the data we are expecting on the items, we create normal Zabbix triggers
- We don’t need anything else to create historic problems

Severity	Name ▲	Operational data	Expression
Warning	High ICMP ping loss **Depends on:** ICMP Ping: Unavailable by ICMP ping	Loss: {ITEM.LASTVALUE1}	**min(**/ICMP Ping/icmppingloss,5m)>{$ICMP_LOSS_WARN} and **min(**/ICMP Ping/icmppingloss,5m)<100
Warning	High ICMP ping response time **Depends on:** ICMP Ping: High ICMP ping loss ICMP Ping: Unavailable by ICMP ping	Value: {ITEM.LASTVALUE1}	**avg(**/ICMP Ping/icmppingsec,5m)>{$ICMP_RESPONSE_TIME_WARN}
High	Unavailable by ICMP ping		**max(**/ICMP Ping/icmpping,#3)=0

Severity	Name ▲	Operational data	Expression
Average	No data or no response on {$TCP.NAME}		**last(**/check_xi_service_tcp/net.tcp.service.perf)=0

Opensource ICT Solutions

Questions?

Opensource ICT Solutions