An idea on how to monitor growth of database tables in an system using Bischeck

Anders Håål
anders.haal@ingby.com

Introduction

The background to this solution is a customer that has a need to monitor the growth of key tables in their ERP system. The requirement was that the tables where not allowed to grow more then 20% percent under a specific time period of 4 hours. If they did, an alarm must be generated in their Nagios based surveillance system.

In this example we will use a number of key features in Bischeck to solve the requirements and also show how Bischeck can be used when simple nagios checks is not enough. Some of the key features are:

- Dynamic thresholding
- Virtual services
- Time based scheduling
- Bischeck cache capability

After we have presented the basic solution we will elaborate how this example can be extended and what additional configuration that can be done to increase the monitoring capability.

Overview

In Bischeck we have the basic concept of host and service. This is very much in-line with the Nagios concept. If Bischeck is integrated with Nagios over NSCA the host and service name must be the same in both Bischeck and Nagios. In addition Bischeck has the concept of serviceitem. A service has one or many serviceitem and its the serviceitem that define what to execute for the service. So for a Bischeck service you define connection to a system and the schedule when to connect to the system to retrieve the monitoring data. When a connection is made all serviceitem execution statements are execute against the server that is to be monitored. When the serviceitem has retrieved the data it is stored in the Bischeck cache and then sent to the threshold class, if configured for the serviceitem, to be evaluated against the threshold rules. The result of the threshold execution is a state – OK, WARNING, CRITICAL or UNKOWN, just like Nagios likes it. The result is then sent to the Nagios as a normal passive check, with state and performance data.

The configuration files for Bischeck is located in the etc directory of the Bischeck installation. In this example we file focus on the bischeck.xml file that define host, service

and serviceitem and on the 24thresholds.xml that define dynamic thresholds over a 24 hour day.

Step 1 – Monitoring the size of the ERP table

First we start with defining a host, a service and serviceitem to query the size of the ERP systems *order* table. The SQL is of course something you must change to what ever way you determine the “size” of a table. For this example its just a count of all the rows.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<bischeck>
    <host>
        <name>erphost</name>
        <desc>Host running ERP system</desc>
        <service>
            <name>erptableSize</name>
            <desc>Service to check tables size in the ERP system</desc>
            <schedule>0 0 */4 * * ? </schedule>
            <url>jdbc:mysql://localhost/bischecktest?user=bischeck&amp;password=bischeck</url>
            <driver>com.mysql.jdbc.Driver</driver>

             <serviceitem>
                <name>erptableOrder</name>
                <desc>Check the table size of the order table</desc>
                <execstatement>select count(*) from order</execstatement>
                <serviceitemclass>SQLServiceItem</serviceitemclass>
            </serviceitem>

        </service>
    </host>
</bischeck>
```

As you see above we defined a host called erphost, a service called erptableSize and one serviceitem called erptableOrder. For the service we use a url defined as a jdbc connection to connect to the ERP server database, of course you have to change this to fit your system and database. The service also have a schedule that is define like cron to run every 4 hour. We also defined the SQL that retrieve the number of rows of the table order. We also define that the serviceitem should use a serviceitem class called SQLServiceItem that is used to do SQL.
The configuration do not have any threshold class defined since its not the absolute size we want to monitor but we want to monitor the growth of the table under a 4 hour period.

To get the data over to Nagios we must enable passive checks in Nagios and setup the NSCA daemon.

In Nagios we will now get a passive check every 4 hour with the total number of rows in the order table, but the state is always OK. If you have pnp4nagios or equivalent you get a nice graph showing how the order table size change.

Step 2 – Monitoring the growth of the ERP table

Since the growth is the ratio between the order table size with 4 hour interval we must use some way to calculate the ratio based on current order table size and what the size was 4 hours ago. This is where the Bischeck cache comes into play and a serviceitem class called CalculateOnCache. This class can do mathematical calculation based on cached data. So we add a new service called erptableGrowth to the host erphost in the existing configuration.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<bischeck>
    <host>
        <name>erphost</name>
        <desc>Host running ERP system</desc>

        <service>
            <name>erptableSize</name>
            <desc>Service to check tables size in the ERP system</desc>
            <schedule>0 0 */4 * * ? </schedule>
            <url>jdbc:mysql://localhost/bischecktest?user=bischeck&amp;password=bischeck</url>
            <driver>com.mysql.jdbc.Driver</driver>

            <serviceitem>
                <name>erptableOrder</name>
                <desc>Check the table size of the order table</desc>
                <execstatement>select count(*) from order</execstatement>
                <serviceitemclass>SQLServiceItem</serviceitemclass>
            </serviceitem>

        </service>

        <service>
            <name>erptableGrowth</name>
            <desc>Service to check the growth of the ERP tables</desc>
            <schedule>0 5 */4 * * ? </schedule>
            <url>bischeck://LastStatusCache</url>

            <serviceitem>
                <name>erptableOrder</name>
                <desc>Check the growth of order table</desc>
                <execstatement>100 * (erphost-erptableSize-erptableOrder[0] - erphost-erptableSize-
erptableOrder[1]) / erphost-erptableSize-erptableOrder[1]</execstatement>
                <thresholdclass>Twenty4HourThreshold</thresholdclass>
                <serviceitemclass>CalculateOnCache</serviceitemclass>
            </serviceitem>

        </service>
    </host>
</bischeck>
```

As you can see from above configuration we now have added the service erptableGrowth. This service has a url to make a connection to Bischeck internal cache. It will run every 4 hour, 5 minutes after the hour. The interesting part is the serviceitem added called erptableOrder that use cached data to calculate some new value that will be used for monitoring. In Bischeck we call this a virtual service since the value is not directly retrieved from a system, instead created from the cached data. The execute statement string will take the last value that was retrieved by
erphost-erptableSize-erptableOrder at index 0 and subtract that from the previous value, index 1, then divide it by the previous value, index 1, and multiple by 100. This will give the

percentage increase of the table size during the last 4 hours.
The notation **erphost-erptableSize-erptableOrder[X]** will be replaced at execution with the real data in the cache, where X is the index. Index 0 is always the last stored data.

But we still have no thresholds for this serviceitem, but we have configured it to use the threshold class Twenty4HourThreshold. So that is our next step, configure the threshold.

Step 3 – Defining thresholds on serviceitem

Earlier we stated that we wanted to have a threshold when the table growth was above 20%. Lets see how this is achieved. To do this we need to do some configuration the file 24thresholds.xml.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<twenty4threshold>
    <servicedef>
        <hostname>erphost</hostname>
        <servicename>erptableGrowth</servicename>
        <serviceitemname>erptableOrder</serviceitemname>
        <period>
            <calcmethod>&lt;</calcmethod>
            <warning>0</warning>
            <critical>10</critical>
            <hoursIDREF>1</hoursIDREF>
        </period>
    </servicedef>

    <hours hoursID="1">
        <hourinterval>
            <from>00:00</from>
            <to>23:00</to>
            <threshold>20</threshold>
        </hourinterval>
    </hours>

    <holiday year="2012">
        <dayofyear>0101</dayofyear>
        <dayofyear>0106</dayofyear>
        <dayofyear>0422</dayofyear>
        <dayofyear>0424</dayofyear>
        <dayofyear>0425</dayofyear>
        <dayofyear>0501</dayofyear>
        <dayofyear>0602</dayofyear>
        <dayofyear>0606</dayofyear>
        <dayofyear>0612</dayofyear>
        <dayofyear>0625</dayofyear>
        <dayofyear>1105</dayofyear>
        <dayofyear>1225</dayofyear>
        <dayofyear>1226</dayofyear>
    </holiday>
</twenty4threshold>
```

For the Twenty4HourThreshold class you define thresholds for each host, service and serviceitem you want to have thresholding on called. These entries are called servicedef. For each servicedef one or many periods can be defined to configure different settings for a combination of a calender like month, week, day in month and day in week. If a period do not have any calendar definitions if will used as the default period.

So in the above configuration we only have a default period that define that the threshold calculation method is <, meaning that measured values should be below threshold. We also define warning and critical level. In this case warning is 0% of threshold and warning alarm will be created if measured value is on the threshold level and up to the critical level which is set to 10% of the threshold. So what is the threshold?

For the period entry an hoursIDREF is defined. This “points” to a 24 hour profile, in this case id 1.
The hours tag has one or several hourinterval tags defining a from-to interval. The threshold set for two consecutive hours are used to calculate a liner equation for any thresholds between the consecutive hours. In our case we made it simple saying that the threshold for all hours are 20.

To summaries what we have done:

1. We retrieve the current size of the order table by using the configuration of *erphost->erptableSize->erptableOrder*
2. We calculate the growth by using the cached data in configuration of the *erphost->erptableGrowth->erptableOrder*.
3. The value from *erphost->erptableGrowth->erptableOrder* is processed through the defined threshold class where we have specified a threshold for all hours to be 20. If the growth is above 20 but below 22 (critical level was set to 10%) the state will be WARNING and if the growth is 22 and above the state will be CRITICAL. State OK is for all values below 20.

Step 4 – Multiple serviceitems

Since we probably have more tables to monitor then the order table we would like to configure monitoring for the rest of the key ERP tables. One way to do this is to make a new service and serviceitem for each of the existing tables or make additional serviceitems on the existing service. We will show last way by adding more serviceitems to the service erptableSize:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<bischeck>
    <host>
        <name>erphost</name>
        <desc>Host running ERP system</desc>

        <service>
            <name>erptableSize</name>
            <desc>Service to check tables size in the ERP system</desc>
            <schedule>0 0 */4 * * ? </schedule>
            <url>jdbc:mysql://localhost/bischecktest?user=bischeck&amp;password=bischeck</url>
            <driver>com.mysql.jdbc.Driver</driver>

            <serviceitem>
                <name>erptableOrder</name>
                <desc>Check the table size of the order table</desc>
```

```
                <execstatement>select count(*) from order</execstatement>
                <serviceitemclass>SQLServiceItem</serviceitemclass>
            </serviceitem>

             <serviceitem>
                <name>erptableCustomer</name>
                <desc>Check the table size of the order table</desc>
                <execstatement>select count(*) from customer</execstatement>
                <serviceitemclass>SQLServiceItem</serviceitemclass>
            </serviceitem>

        </service>

        <service>
            <name>erptableGrowth</name>
            <desc>Service to check the growth of the ERP tables</desc>
            <schedule>0 5 */4 * * ? </schedule>
            <url>bischeck://LastStatusCache</url>

            <serviceitem>
                <name>erptableOrder</name>
                <desc>Check the growth of order table</desc>
                <execstatement>100 * (erphost-erptableSize-erptableOrder[0] - erphost-erptableSize-
erptableOrder[1]) / erphost-erptableSize-erptableOrder[1]</execstatement>
                <thresholdclass>Twenty4HourThreshold</thresholdclass>
                <serviceitemclass>CalculateOnCache</serviceitemclass>
            </serviceitem>

        </service>
    </host>
</bischeck>
```

Now we added the serviceitem to measure table of the customer table in the existing service erptableSize. This is possible if:

- All serviceitem can used the same url connection, which for this example means that the tables are in the same database.
- Its okay to execute all serviceitems on the same scheduled time defined by the service.

Even if we have multiple serviceitem on a service all data will be sent to Nagios in the NSCA message so each individual serviceitems can be graphed. Message will for a service with multiple serviceitem look like this example:

```
OK erptableOrder = 61000 (NA) erptableCustomer = 10001 (NA)  | erptableOrder=61000;0.0;0.0;0;
threshold=0.0;0;0;0;erptableCustomer=10001;0.0;0.0;0; threshold=0.0;0;0;0; avg-exec-time=10ms
```

The (NA) means that no threshold is used for this serviceitem, which is true for the measuring of the table size.

Multiple serviceitems can of course be configured for our growth service erptableGrowth:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<bischeck>
    <host>
        <name>erphost</name>
        <desc>Host running ERP system</desc>

        <service>
            <name>erptableSize</name>
            <desc>Service to check tables size in the ERP system</desc>
            <schedule>0 0 */4 * * ? </schedule>
            <url>jdbc:mysql://localhost/bischecktest?user=bischeck&amp;password=bischeck</url>
            <driver>com.mysql.jdbc.Driver</driver>

            <serviceitem>
                <name>erptableOrder</name>
                <desc>Check the table size of the order table</desc>
                <execstatement>select count(*) from order</execstatement>
                <serviceitemclass>SQLServiceItem</serviceitemclass>
            </serviceitem>

            <serviceitem>
                <name>erptableCustomer</name>
                <desc>Check the table size of the order table</desc>
                <execstatement>select count(*) from customer</execstatement>
                <serviceitemclass>SQLServiceItem</serviceitemclass>
            </serviceitem>

        </service>

        <service>
            <name>erptableGrowth</name>
            <desc>Service to check the growth of the ERP tables</desc>
            <schedule>0 5 */4 * * ? </schedule>
            <url>bischeck://LastStatusCache</url>

            <serviceitem>
                <name>erptableOrder</name>
                <desc>Check the growth of order table</desc>
                <execstatement>100 * (erphost-erptableSize-erptableOrder[0] - erphost-erptableSize-
erptableOrder[1]) / erphost-erptableSize-erptableOrder[1]</execstatement>
                <thresholdclass>Twenty4HourThreshold</thresholdclass>
                <serviceitemclass>CalculateOnCache</serviceitemclass>
            </serviceitem>
<serviceitem>
                <name>erptableCustomer</name>
                <desc>Check the growth of order table</desc>
                <execstatement>100 * (erphost-erptableSize-erptableCustomer[0] - erphost-
erptableSize-erptableCustomer[1]) / erphost-erptableSize-erptableCustomer[1]</execstatement>
                <thresholdclass>Twenty4HourThreshold</thresholdclass>
                <serviceitemclass>CalculateOnCache</serviceitemclass>
            </serviceitem>

        </service>
    </host>
</bischeck>
```

To get the threshold for the new customer table an additional configuration in the 24thresholds.xml file must be done in the same way it was done for the threshold for the order growth serviceitem.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<twenty4threshold>
    <servicedef>
        <hostname>erphost</hostname>
        <servicename>erptableGrowth</servicename>
        <serviceitemname>erptableOrder</serviceitemname>
        <period>
            <calcmethod>&lt;</calcmethod>
            <warning>0</warning>
            <critical>10</critical>
            <hoursIDREF>1</hoursIDREF>
        </period>
    </servicedef>

    <servicedef>
        <hostname>erphost</hostname>
```

```
        <servicename>erptableGrowth</servicename>
        <serviceitemname>erptableCustomer</serviceitemname>
        <period>
            <calcmethod>&lt;</calcmethod>
            <warning>5</warning>
            <critical>15</critical>
            <hoursIDREF>1</hoursIDREF>
        </period>
    </servicedef>
```

....

As you can see we have different warning and critical level for customers but we still use the same hour profile definition where hoursID 1. This can of course be changed by creating additional hours with different profile.

When we have multiple serviceitems on a single service it important to understand that the state of the service that will be sent to Nagios is the state of the serviceitem with the highest severity level. This means if the order growth is critical and the customer growth is in an OK the state will be critical in Nagios. The benefit in Nagios is that we only have one service called erptableGrowth that is that include all tables.

To have multiple serviceitem on a single service or have separated service with just one serviceitem is depending on how you like to manage it in Nagios. If you want one alarm independent of what table that is growing above threshold and its the same group managing all tables then its probably good to have a single service with multiple serviceitems. From the data sent to Nagios it still possible to see what individual serviceitem that was above threshold.

Step 5 – More on advanced thresholding

There is some additional configuration we can do on our thresholding:

- More periods
- Different values in the 24 period

The reason to have more periods definition is that business volumes are different depending on calendar. For example we would expect that the order table grow much more if its a Friday, if its the first day of the month and if its the 5th of December because this is how the business works (This is just for example). So if this is Friday, the first day of the month and/or the 5/12 we can accept 30% growth during the 4 hour period. To achieve this we add additional period definitions to the servicedef and add additional hours.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<twenty4threshold>
    <servicedef>
        <hostname>erphost</hostname>
        <servicename>erptableGrowth</servicename>
        <serviceitemname>erptableOrder</serviceitemname>
        <period>
            <months>
```

```
                <dayofmonth>1</dayofmonth>
            </months>
            <months>
                <month>12</month>
                <dayofmonth>5</dayofmonth>
            </months>
            <weeks>
                <dayofweek>6</dayofweek>
            </weeks>
            <calcmethod>&lt;</calcmethod>
            <warning>5</warning>
            <critical>15</critical>
            <hoursIDREF>2</hoursIDREF>
        </period>

        <period>
            <calcmethod>&lt;</calcmethod>
            <warning>0</warning>
            <critical>10</critical>
            <hoursIDREF>1</hoursIDREF>
        </period>
    </servicedef>

    <hours hoursID="1">
        <hourinterval>
            <from>00:00</from>
            <to>23:00</to>
            <threshold>20</threshold>
        </hourinterval>
    </hours>

    <hours hoursID="2">
        <hourinterval>
            <from>00:00</from>
            <to>23:00</to>
            <threshold>30</threshold>
        </hourinterval>
    </hours>
```

The other thing we could configure is if we like to have a higher granularity of the growth depending on the time of the day. Currently we have set 20 (or 30) for all 24 hours. But maybe we can expect to have a higher growth during some period of the day.

```
    <hours hoursID="1">
        <hourinterval>
            <from>00:00</from>
            <to>09:00</to>
            <threshold>20</threshold>
        </hourinterval>

        <hourinterval>
            <from>10:00</from>
            <to>14:00</to>
            <threshold>30</threshold>
        </hourinterval>

        <hourinterval>
            <from>15:00</from>
            <to>23:00</to>
            <threshold>20</threshold>
        </hourinterval>

    </hours>
```

Now we have set the threshold for 10:00 and 14:00 to 30. This means between 09:00 and

10:00 we calculate the threshold as a linear equation with the starting at 20 and end at 30. This means if the threshold is check at 10:30 the threshold is calculated to 25. With mechanism we can define the dynamic in thresholds in the same way that our business is dynamic depending on time of day and by day in the calendar.

If we like to make the threshold more advanced we can instead of number use data we have in the cache. For example if there is a relation between the customer table growth and the order table growth that define that customer table growth should be less then 80% of the order growth. To configure this we can now use the cache data to create a hour threshold profile with the logic like “`0.8* erphost-erptableGrowth-erptableOrder[0]`“. Below is a hours profile that show a combination of what has been discussed that could be used by the customer growth.

```
    <hours hoursID="10">
        <hourinterval>
            <from>00:00</from>
            <to>09:00</to>
            <threshold>20</threshold>
        </hourinterval>

        <hourinterval>
            <from>10:00</from>
            <to>14:00</to>
            <threshold>0.8* erphost-erptableGrowth-erptableOrder[0]</threshold>
        </hourinterval>

        <hourinterval>
            <from>15:00</from>
            <to>23:00</to>
            <threshold>30</threshold>
        </hourinterval>

    </hours>
```

Step 6 – Just one more thing

As you remember from the introduction we where not interested to measure the size of the tables but only the growth. For that reason no thresholding was configured for the serviceitems belonging to the service erptableSize. But lets say we also like to set a threshold for the max size of the tables to get alarm when that level is reached. To do this we simple add the thresholdclass to the existing serviceitems in service erptableSize.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<bischeck>
    <host>
        <name>erphost</name>
        <desc>Host running ERP system</desc>

        <service>
            <name>erptableSize</name>
            <desc>Service to check tables size in the ERP system</desc>
            <schedule>0 0 */4 * * ? </schedule>
            <url>jdbc:mysql://localhost/bischecktest?user=bischeck&amp;password=bischeck</url>
            <driver>com.mysql.jdbc.Driver</driver>
```

```
            <serviceitem>
                <name>erptableOrder</name>
                <desc>Check the table size of the order table</desc>
                <execstatement>select count(*) from order</execstatement>
                <serviceitemclass>SQLServiceItem</serviceitemclass>
                <thresholdclass>Twenty4HourThreshold</thresholdclass>
            </serviceitem>

            <serviceitem>
                <name>erptableCustomer</name>
                <desc>Check the table size of the order table</desc>
                <execstatement>select count(*) from customer</execstatement>
                <serviceitemclass>SQLServiceItem</serviceitemclass>
                <thresholdclass>Twenty4HourThreshold</thresholdclass>
            </serviceitem>

        </service>

        . . .
</bischeck>
```

Then we create a threshold configuration for the serviceitems, like:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<twenty4threshold>
    <servicedef>
        <hostname>erphost</hostname>
        <servicename>erptableSize</servicename>
        <serviceitemname>erptableOrder</serviceitemname>
        <period>
            <calcmethod>&lt;</calcmethod>
            <warning>10</warning>
            <critical>20</critical>
            <hoursIDREF>4</hoursIDREF>
        </period>
    </servicedef>
   ...
    <hours hoursID="4">
        <hourinterval>
            <from>08:00</from>
            <to>18:00</to>
            <threshold>200000</threshold>
        </hourinterval>
    </hours>
```

So now we have defined a size threshold for the order table at 200 000 between 08:00 ans 18:00. A warning alarm will be generated when the size reach 220 000 and a critical alarm when the size reach 240 000.

Setting up Nagios

To make this example integrate with Nagios the following steps is required:

- Nagios must allow passive checks – nagios.cfg
- Nsca must be started. Nsca password and “encryption” mode must be define in nsca.cfg and in Bischeck properties.xml
- Create host erphost in Nagios
- Create service erptableSize and erptableGrowth in Nagios and enable passive check for both.

Summery

Hopefully this paper show a solution to a problem that Bischeck can fix in a simple and effective way by using the features of virtual service, dynamic thresholding and by using cached historical data. The solution to the problem as been solved without any coding, only Bischeck configuration. The principles of this solution should be applicable to any problem with the same characteristics where we want to detect growth or decline of some measurable volume.

There is also a other important aspect of this and that is maintenance. When business change it easy to change or add the Bischeck configuration without any changes or restarting requirements on the Nagios infrastructure. The Bisconf web tool support management of the Bischeck configuration, including versions of configuration files, tracking on who made the changes and for user with the right authority Bischeck can be reloaded directly from Bisconf.

Bischeck and Bisconf are licensed under GPL2. To learn more about Bischeck and Bisconf read more at www.bischeck.org.