ZABBIX 2012 Conference

JESTA digital

ZABBIX & JavaEE Platform Monitoring

A Good Match?

RIGA, 22 Sep 2012 | Marek Neumann

Company Facts

- Jesta Digital is a leading global provider of next generation entertainment content and services for the digital consumer.
- subsidiary of Jesta Group, a diversified company with holdings in real estate, manufacturing, technology and aviation.
- home to established brands - Jamba, Jamster, iLove and Mobizzo and mobile subscription, payment and ad monetization technologies

Who am I?

- more than 10 years experience in various areas of Java and JavaEE
- 6 years work for different consulting companies
- JBoss support and training pioneer
- strategy and architecture team @ Jesta Digital
  - technical guidelines, software infrastructure
  - Application Monitoring is one aspect of our work
- settled near Berlin with my family (2 kids)
- spending much of my spare time for marathon training

Agenda

- Jesta Digital application monitoring architecture
- Performance problems and how we tackled them
- Zabbix API: Automization of the monitoring configuration using an in-house application
- Zabbix API: Automization of the service monitoring within the UltraESB
- Zabbix Server monitoring in a public cloud
- Summary

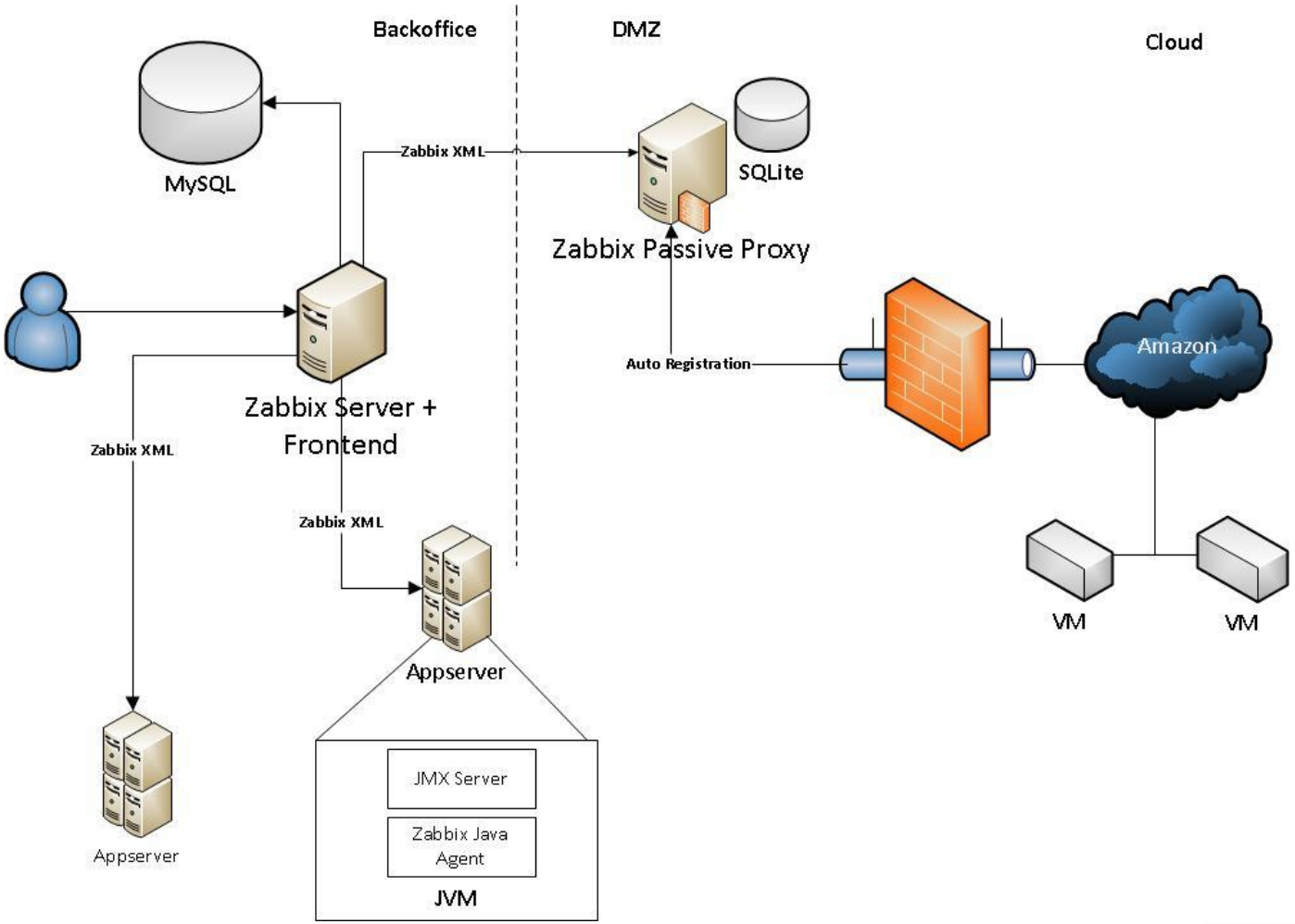
Application Monitoring Architecture

- Zabbix 1.8.5 (supported version)
- Server with passive Java agents
- Passive proxy for cloud hosts
- MySQL 5.5 backend
- separate installation for test and internal systems (CI, Staging...)
- 24x7 Monitoring team (SOC) with access to Zabbix and other monitoring tools

Status of Zabbix

Parameter	Value	Details
Zabbix server is running	Yes	localhost:11051
Number of hosts (monitored/not monitored/templates)	1054	271 / 320 / 463
Number of items (monitored/disabled/not supported)	38988	36530 / 443 / 2015
Number of triggers (enabled/disabled)[problem/unknown/ok]	45017	40532 / 4485 [8 / 0 / 40524]
Number of users (online)	34	6
Required server performance, new values per second	223.67	-

Application Monitoring Architecture

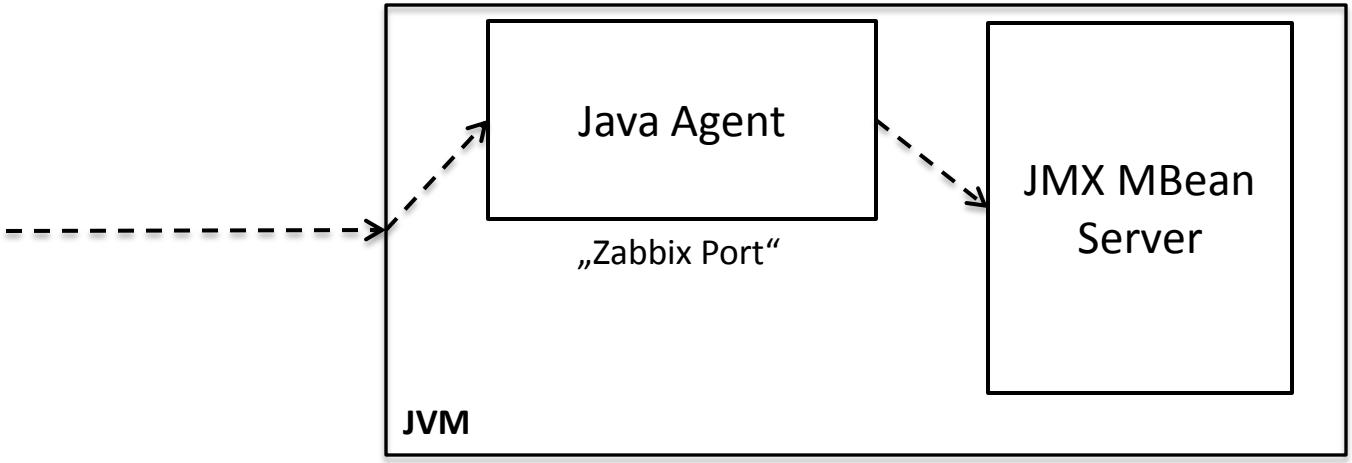


Application Monitoring Architecture

- JVMs monitored
  - JBoss Application Server 4.3.0_EAP/6.0.0_EAP
  - UltraESB 1.7.1
  - Apache Tomcat 7.x
  - (Bea Weblogic 8.x)
- WHAT is monitored
  - basic JVM metrics (heap/perm gen memory usage, garbage collection, file descriptors...)
  - business metrics (content index size, subscription reminder sms count...)
  - host availability (http port check)
  - database query executions
  - exception counts
  - log level counts (WARN, ERROR, FATAL)

Application Monitoring Architecture

- JMX-based architecture (standard way to gather metrics from a JVM by using queries and requests)
- implemented by many application server vendors
- is part of JDK since version 5
- Zabbix agent is essential part of the appserver installations



Application Monitoring Architecture

- Zabbix Agent is a modified implementation of former „Zapcat“
  - extended to support more complicated object structures and to method calls
- easily deployable in the application server (JBoss, UltraESB, Tomcat...)
- transformation of Zabbix protocol to JMX syntax and vice-versa
- local „in-VM“ calls to ensure good performance

☐	Total Collection Time	Triggers (1)	jmx[com.jamster.infra.appserver.monitoring:service=GcMonitor][CollectionTime]

Severity	Name	Expression
High	More than 60s Full Garbage Collection time within 15min	{GarbageCollection_Common_Template:jmx[com.jamster.infra.appserver.monit

method[com.jamster.infra.appserver.esb:Type=ErrorLogMonitor][getRoutingFaults=LIVE]

- JMX client is provided in Zabbix 2.0 upwards - no agent is necessary anymore

Application Monitoring Architecture

- JVM Availability Check
  - first version was implemented using noData() function
  - flood of false positives when server performance degraded
  - changed to simple TCP check with DISASTER alerts (90s interval)

```
net.tcp.service.perf[http,host1,11811]

({JVM_AVAILABLE_TEMPLATE:net.tcp.service.perf[http,host1,11811].la
st(#5)}=0) &
({JVM_AVAILABLE_TEMPLATE:net.tcp.service.perf[http,host1,11811].la
st(#4)}=0) &
({JVM_AVAILABLE_TEMPLATE:net.tcp.service.perf[http,host1,11811].la
st(#3)}=0) &
({JVM_AVAILABLE_TEMPLATE:net.tcp.service.perf[http,host1,11811].la
st(#2)}=0) &
({JVM_AVAILABLE_TEMPLATE:net.tcp.service.perf[http,host1,11811].la
st(#1)}=0)
```

Application Monitoring Architecture

- all templates are provided by S&A team to support infrastructure monitoring requirements
- developers can easily add new „business“ monitoring items by implementing JMX MBeans
  - no special knowledge of Zabbix is required
- the configuration process has a lot of manual steps right now
  - high workload for Operations team

Performance problems and how we tackled them

- Zabbix was introduced at Jesta Digital in 2008 (v1.6)
  - decision based on the architecture and the frontend capabilities
- monitoring for a big new customer was required
- existing monitoring was based on complicated custom implementations that nobody wanted to maintain
- over the last years we faced some severe performance problems
- let‘s go through our „Zabbix performance learning curve“!

Performance problems and how we tackled them

Performance Problem #1 - Virtualized server setup

- very first installation was on a virtualized server - both Zabbix Server and MySQL backend
- I/O throughputs were temporarily poor, degraded without any visible reason
- server queue was filling quickly, noData() function reported alerts due to the exhausted queue, delayed item processing

Solution:

- Zabbix database was moved to physical hardware (16 Cores, 32 GB RAM, Linux 64bit)
- Availability check was changed to simple TCP

Performance problems and how we tackled them

Performance Problem #2 - Zabbix Housekeeper

- was configured to run every hour
- concurrent write processes blocked during that time (transaction timeouts), slow frontend → queueing problems
  - template import, host deletion, mass changes

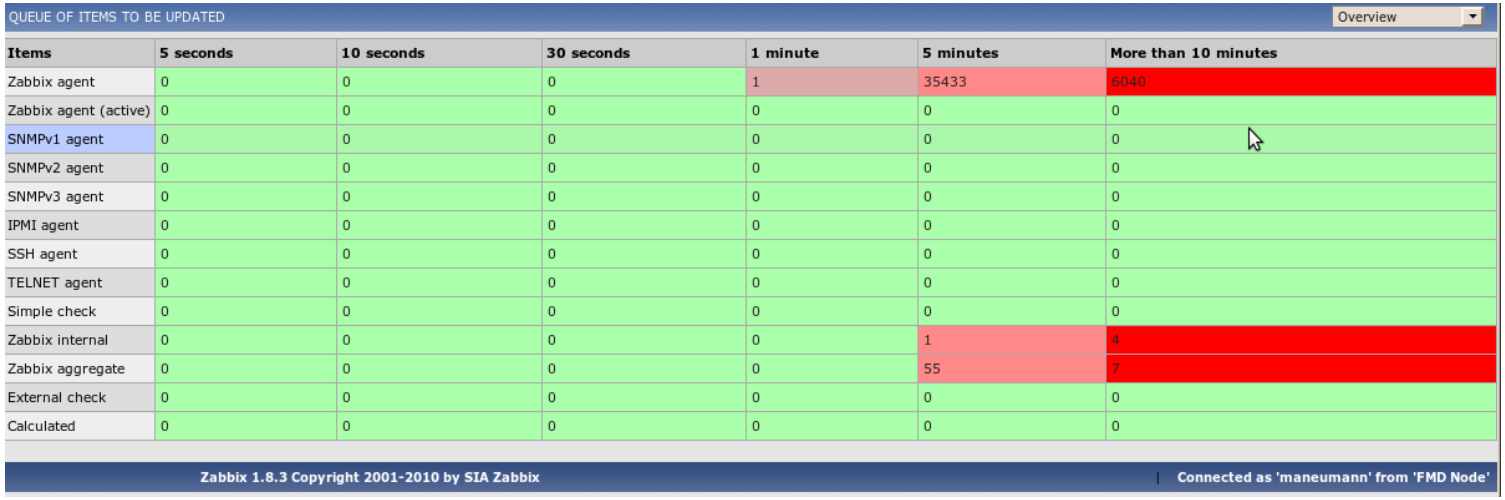
Solution:

- stop Zabbix housekeeping, introduce MySQL partitioning for history_uint and trends_uint tables
- deletion of obsolete historical data is now much quicker and has no measureable impact on the Zabbix performance
- partitions are cutted on daily and monthly base

Performance problems and how we tackled them

Performance Problem #3 - MySQL configuration (Log Size)

- Symptom: Zabbix queue filled up without any clear cause, item processing delayed, no recover *without db restart*
- from operational point of view all systems were working correctly
  - system load, cpu usage, memory, swap



QUEUE OF ITEMS TO BE UPDATED — Overview

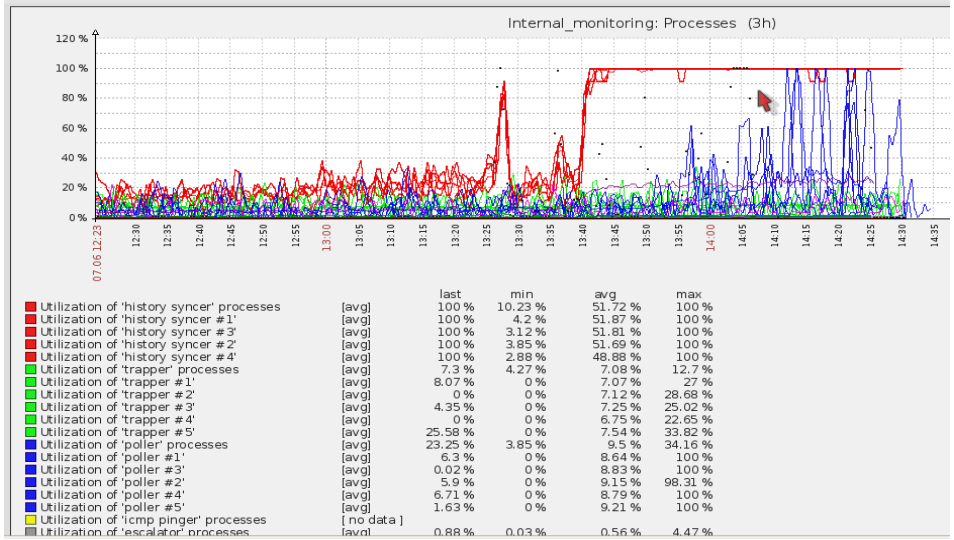
Items	5 seconds	10 seconds	30 seconds	1 minute	5 minutes	More than 10 minutes
Zabbix agent	0	0	0	1	35433	6040
Zabbix agent (active)	0	0	0	0	0	0
SNMPv1 agent	0	0	0	0	0	0
SNMPv2 agent	0	0	0	0	0	0
SNMPv3 agent	0	0	0	0	0	0
IPMI agent	0	0	0	0	0	0
SSH agent	0	0	0	0	0	0
TELNET agent	0	0	0	0	0	0
Simple check	0	0	0	0	0	0
Zabbix internal	0	0	0	0	1	4
Zabbix aggregate	0	0	0	0	55	7
External check	0	0	0	0	0	0
Calculated	0	0	0	0	0	0

Zabbix 1.8.3 Copyright 2001-2010 by SIA Zabbix | Connected as 'maneumann' from 'FMD Node'

Performance problems and how we tackled them

Solution:

- upgrade from 1.8.3 to 1.8.5
- good understanding of background processes
- introduce performance metrics to visualize Zabbix „internal“ performance (thanks to support)
- cause was related to poor syncer thread performance (persist history)



		last	min	avg	max
Utilization of 'history syncer' processes	[avg]	100 %	10.23 %	51.72 %	100 %
Utilization of 'history syncer #1'	[avg]	100 %	4.2 %	51.87 %	100 %
Utilization of 'history syncer #3'	[avg]	100 %	3.12 %	51.81 %	100 %
Utilization of 'history syncer #2'	[avg]	100 %	3.85 %	51.69 %	100 %
Utilization of 'history syncer #4'	[avg]	100 %	2.88 %	48.88 %	100 %
Utilization of 'trapper' processes	[avg]	7.3 %	4.27 %	7.08 %	12.7 %
Utilization of 'trapper #1'	[avg]	8.07 %	0 %	7.07 %	27 %
Utilization of 'trapper #2'	[avg]	0 %	0 %	7.12 %	28.68 %
Utilization of 'trapper #3'	[avg]	4.35 %	0 %	7.25 %	25.02 %
Utilization of 'trapper #4'	[avg]	0 %	0 %	6.75 %	22.65 %
Utilization of 'trapper #5'	[avg]	25.58 %	0 %	7.54 %	33.82 %
Utilization of 'poller' processes	[avg]	23.25 %	3.85 %	9.5 %	34.16 %
Utilization of 'poller #1'	[avg]	6.3 %	0 %	8.64 %	100 %
Utilization of 'poller #3'	[avg]	0.02 %	0 %	8.83 %	100 %
Utilization of 'poller #2'	[avg]	5.9 %	0 %	9.15 %	98.31 %
Utilization of 'poller #4'	[avg]	6.71 %	0 %	8.79 %	100 %
Utilization of 'poller #5'	[avg]	1.63 %	0 %	9.21 %	100 %
Utilization of 'icmp pinger' processes	[no data]				
Utilization of 'escalator' processes	[avg]	0.88 %	0.03 %	0.56 %	4.47 %

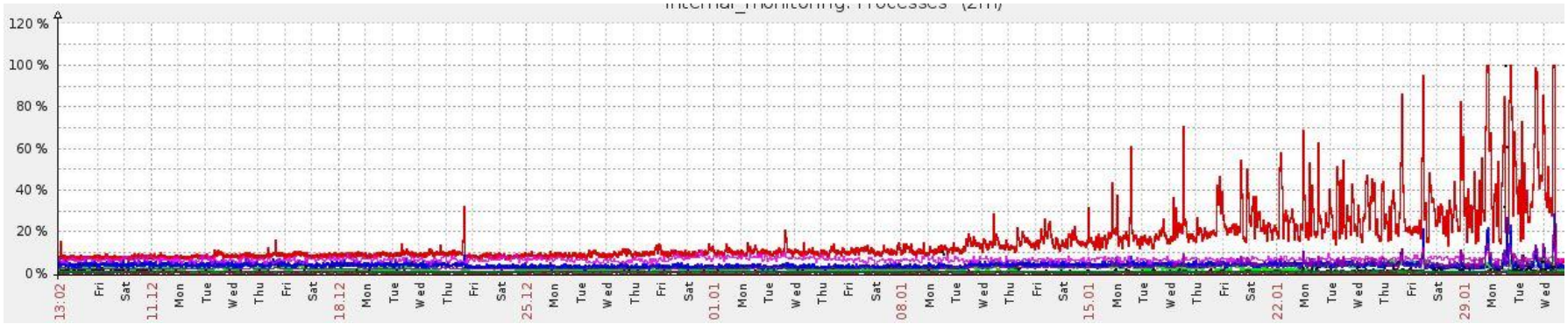
Performance problems and how we tackled them

- experimented with several MySQL options
- innodb_log_file_size: transaction log
- was set to 5MB before (causing high I/O overhead on the disc)
- a correct size can be easily calculated (depending on the current MySQL workload)
  - http://www.mysqlperformanceblog.com/2008/11/21/how-to-calculate-a-good-innodb-log-file-size/
- the log file size was then increased to 270MB
- no queue problems afterwards, normal and steady syncer thread usage

Performance problems and how we tackled them

Performance Problem #4 - MySQL configuration (Query Cache)

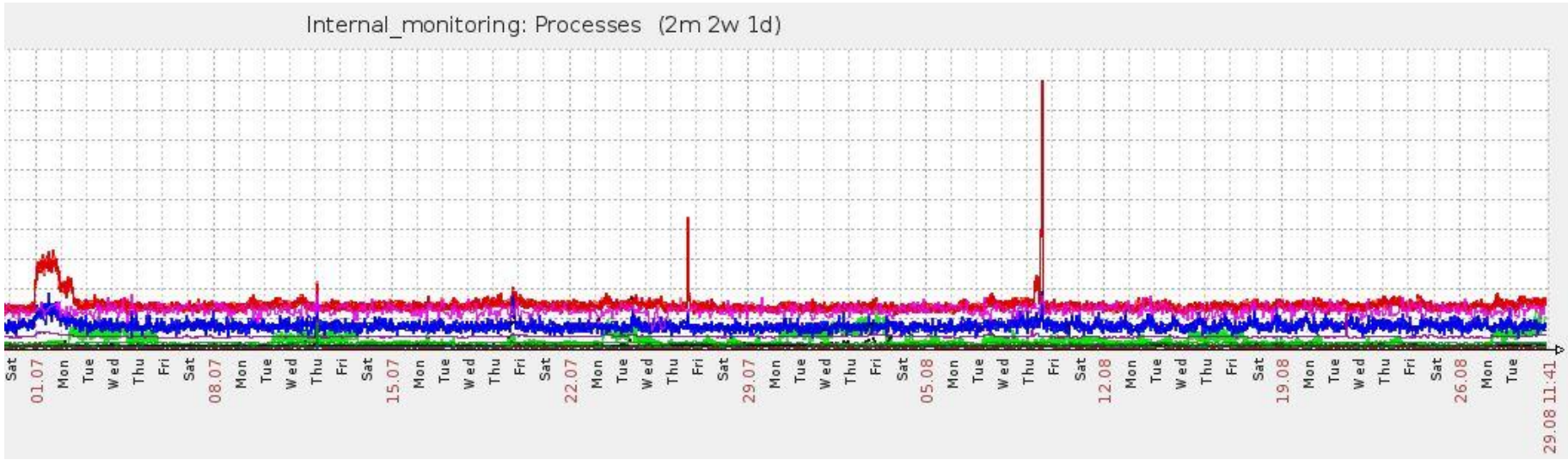
- Symptom: Zabbix queue filled up without any clear cause, slow but steady increase of syncer and poller usage, no self-recovery
- database restart needed *~every two months*
- MySQL threads: „Waiting for query cache lock”



Performance problems and how we tackled them

Solution:

- decrease the MySQL query cache size limit from 8GB! to 256MB
- when the cache size is set too high there more and more thread contention during updates
- ~400.000 results in query cache, so the limit is sufficient
- long-term graph reveals that the problem is solved:



Performance problems and how we tackled them

Our „Lessons Learnt“:

- do not virtualize the database server
- introduce Zabbix internal monitoring, esp. syncer usage
  - *zabbix[process,**history syncer**,avg,busy]*
  - Utilization of all history syncer threads more than 50%: *{Template_Internal_Monitoring:zabbix[process,history syncer,avg,busy].avg(600)}>50*
- adapt the database configuration to your requirements!
  - Zabbix server itself is quick enough for processing high througputs
- use MySQL partitioning instead of housekeeper to avoid concurrent write blocking

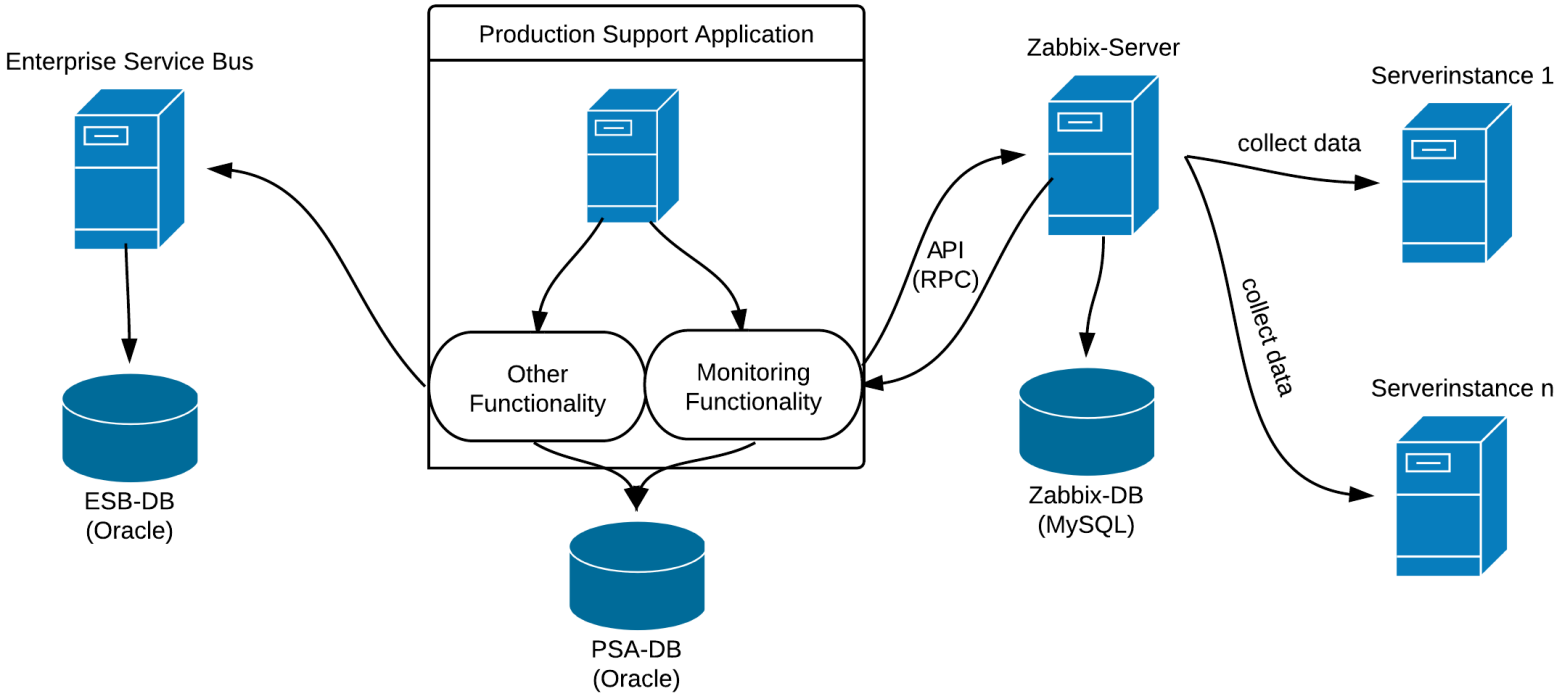
Zabbix API: Automization

- configuration changes were a manual process since we introduced Zabbix in 2008
- error-prone and time-consuming task for the operations team
- all templates must be created using a template generator
- imported into Zabbix using the frontend
- User Macros were not available at that time - a lot of templates have to be generated
- since 1.8 Zabbix API introduces more flexibility when it comes to administrative tasks
- POC: use the api to temporarily (de)activate host availibility checks

Zabbix API: Automization

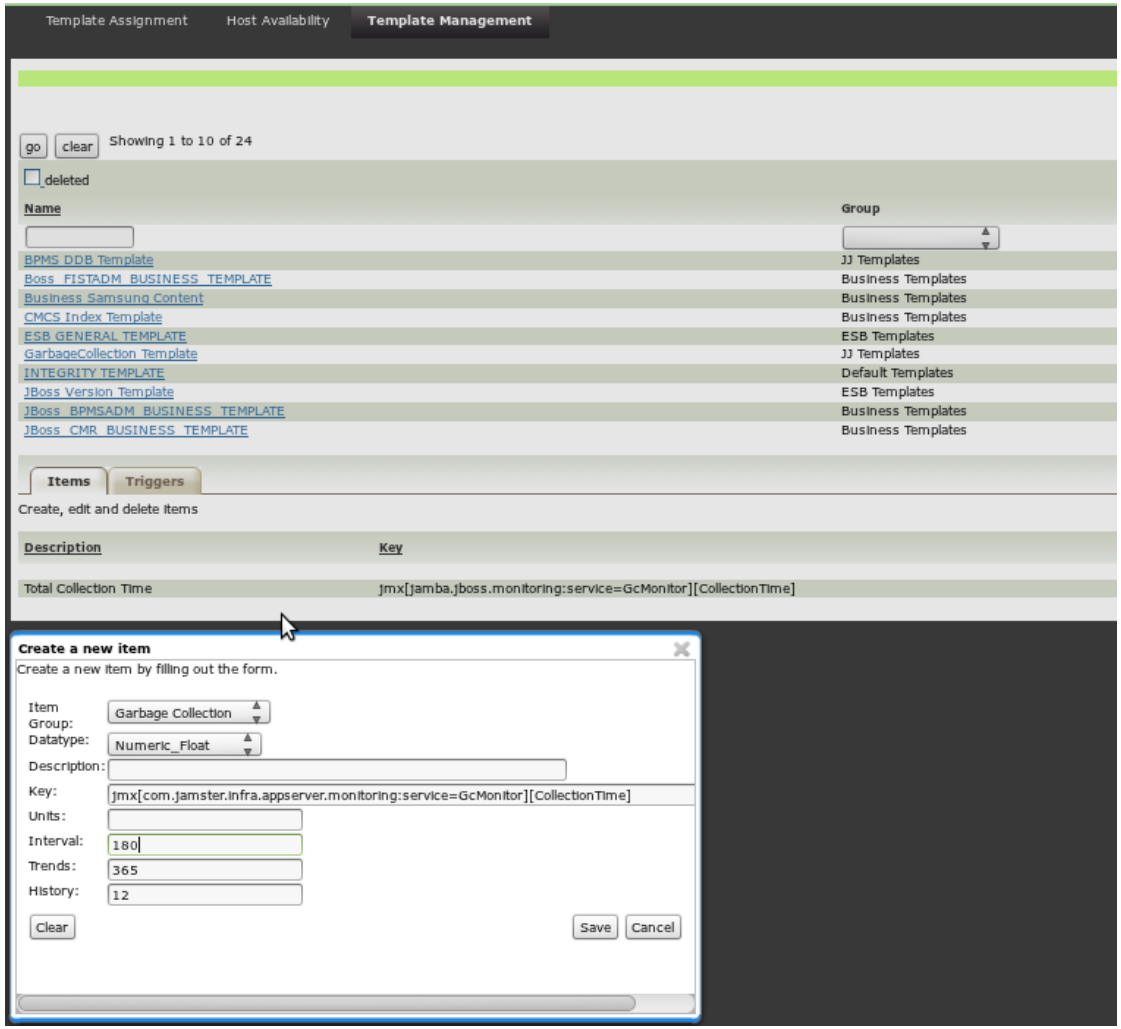
- Overall goal: all administrative tasks can be done without the Zabbix frontend (read-only access)
- reduce the maintenance efforts by ~70%
- integration is done with an existing inhouse application
  - application is managing complete server infrastructure and is the service repository for the ESB
- templates can be created and assigned to different abstraction levels
- all changes are recorded and can be rolled back
- easier to change only single values
  - change the threshold of a trigger
  - add and remove items

Zabbix API: Automization



Zabbix API: Automization

- create/update/delete templates
- create/update/delete items/triggers on templates
- (de)activate hosts and availability checks
- bulk changes supported to avoid single remote operations over the network

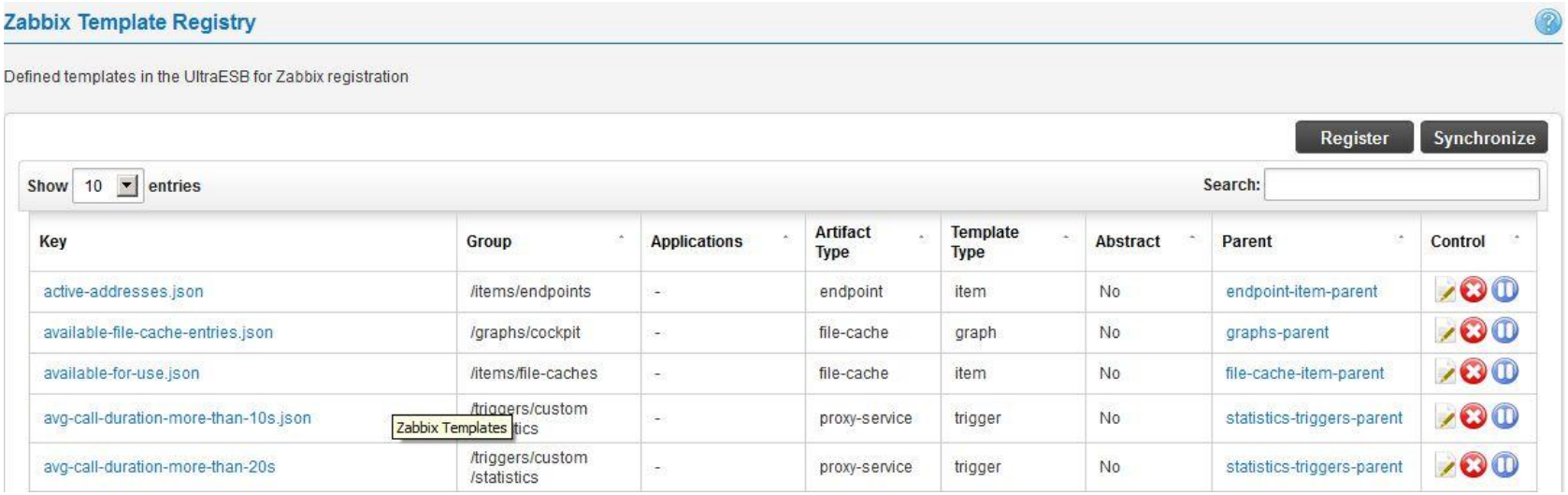


Zabbix API: ESB Monitoring Automization

- ESB is the central part of the service-oriented architecture in the platform
- remote communication between software components is done through the ESB
- ESB was integrated into the monitoring long ago
  - manual configuration process
  - huge templates (1:1 mapping template-service)
  - long-lasting configuration updates due to long template import times
  - outdated monitoring configuration
- Requirement: **update the monitoring configuration once the underlying ESB gets a new version**
  - no manual intervention should be required

Zabbix API: ESB Monitoring Automization

- Usage of the Zabbix api for integration!
- Administration can be done either using the web console or the command line client
- Templates are located on the disc (JSON structure)



Zabbix API: ESB Monitoring Automization

```
{
    "uz_meta": {
        "parent": "endpoint-item-parent"
    },
    "params":{
        "description": "${.*LIVE.*}$ endpoint - state READY address count",
        "key_":
"jmx[org.adroitlogic.ultraesb.detail:Type=Endpoints,Name=${.*LIVE.*}$][Detai
ls.readyAddressCount]"
    }
}
```

```
{
    "uz_meta": {
        "parent": "queue-item-parent"
    },
    "params":{
        "description": "Log queue defaultFault current size",
        "key_":
"jmx[com.jamster.infra.appserver.esb:Type=Queue,ConnectorName=log_queue_defa
ultFault][CurrentSize]"
    }
}
```

Zabbix API: ESB Monitoring Automization

- predefined or custom items are possible
  - calls in progress, caches, service execution times, endpoints
- cluster update is also done (Zookeeper-based)

Name	Applications	Items	Triggers	Graphs
esbusadm_esb01_infra	Applications (14)	Items (138)	Triggers (77)	Graphs (17)
esbusadm_esb02_infra	Applications (14)	Items (138)	Triggers (77)	Graphs (17)
esbusadm_esb03_infra	Applications (14)	Items (138)	Triggers (77)	Graphs (17)

- graphs and screens can be updated through the api as well!

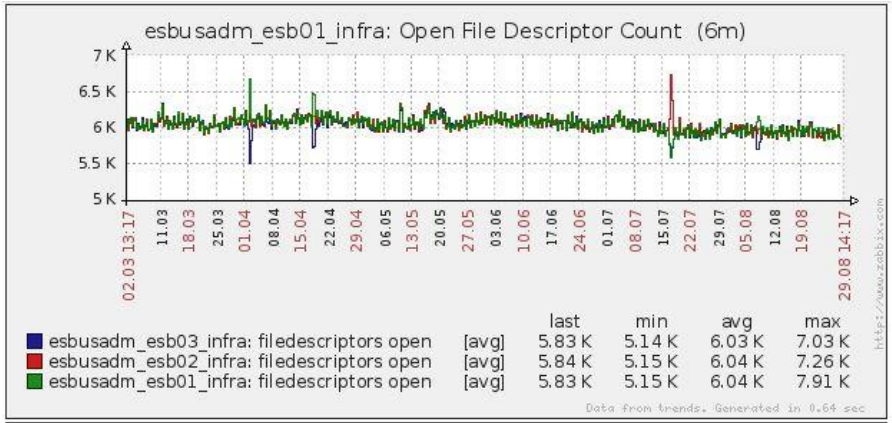
Zabbix API: ESB Monitoring Automization

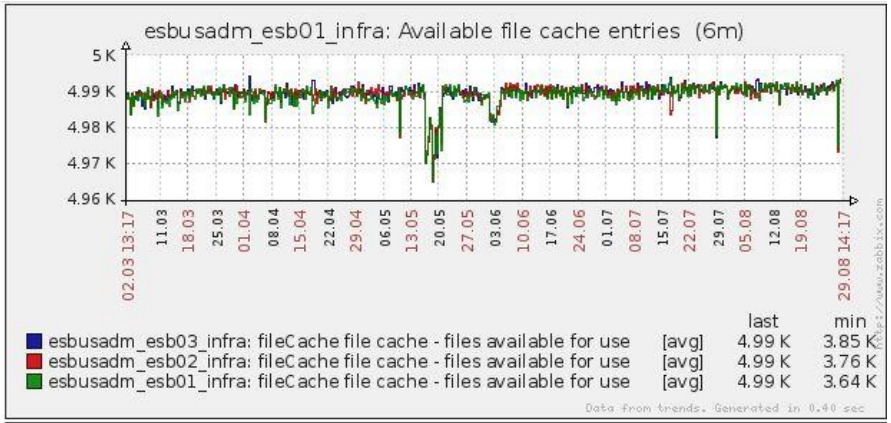
```
{
    "uz_meta": {
        "parent": "screens-parent",
        "cluster": "true"
    },
    "params": {
        "hsize":"2",
        "vsize":"7",
        "name":"UltraESB cluster cockpit for $installation$",
        "screenitems":[{
            "resourcetype":"0",
            "resourceid":"Open File Descriptor Count",
            "width":"500",
```

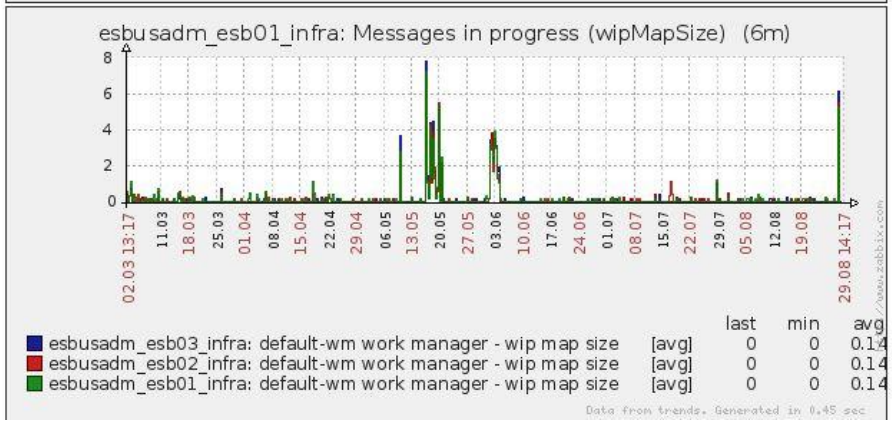
```
 ${SHARED_ESB_DIR}/bin/uterm.sh -configdir ${HOME}/${CLUSTER_DIR}/conf -c zr
-zu $ZABBIX_URL -u $ZABBIX_USER -p $ZABBIX_PASS -t
${HOME}/${CLUSTER_DIR}/conf/hosts.properties -doyw
```

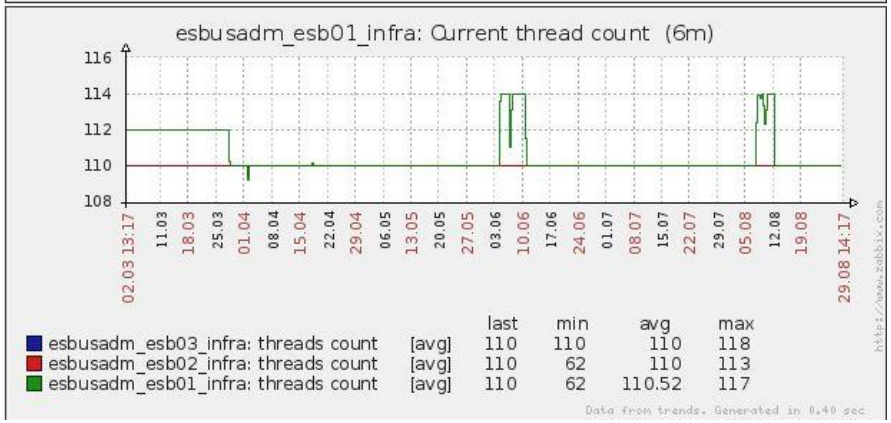
Zabbix API: ESB Monitoring Automization

Voilà: UltraESB Monitoring Cockpit







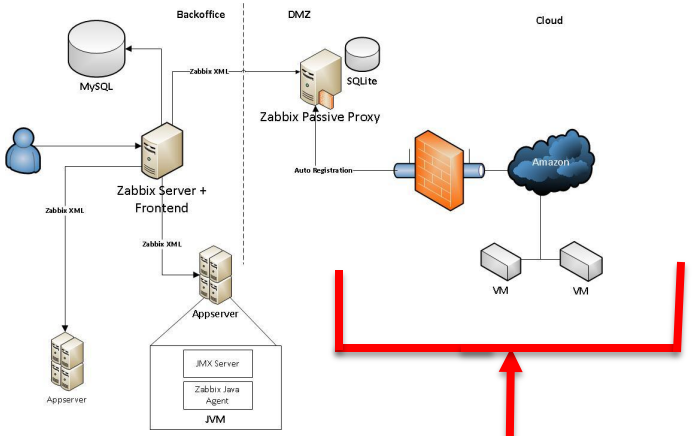


Zabbix Server monitoring in a public cloud

- some of our services hosted in a public cloud (Amazon)
- monitoring principle is similiar to the corporate one
  - firewall restrictions exist - no access to cloud hosts from corporate network
  - Zabbix server has no access to DMZ
- Zabbix developed supported feature - the passive proxy
  - Zabbix server is polling the Zabbix proxy for data
- Cloud instances (Apache Tomcat) are „equipped“ with the Zabbix Java Agents for data retrieval

Zabbix Server monitoring in a public cloud

- Auto-Registration feature:
  - during startup the instance sends a register request to the proxy („stolen“ from native agent)




```
{ZBXD01E "request":"active checks",
    "host":„10.20.30.40", "port":65535}
```

CONFIGURATION OF ACTIONS

Action

Name	Auto Registration Rule (Cloud)
Event source	Auto registration
Default subject	Cloud auto registration: new instance registered
Default message	Cloud auto registration: a new instance was registered and is now being monitored: Server: {HOSTNAME} IP: {IPADDRESS} Port: {HOST.PORT}
Status	Enabled

Save | Clone | Delete | Cancel

Action operations

Details	Action
Add host	Edit
Send message to User "SOC"	Edit
Add to group "Cloud"	Edit
Link to template "ExceptionMonitoring_Template"	Edit
Link to template "LogMonitoring_Template"	Edit
Link to template "GarbageCollection_Common_Template"	Edit
Link to template "JVM_TEMPLATE_Cloud"	Edit

New | Delete selected

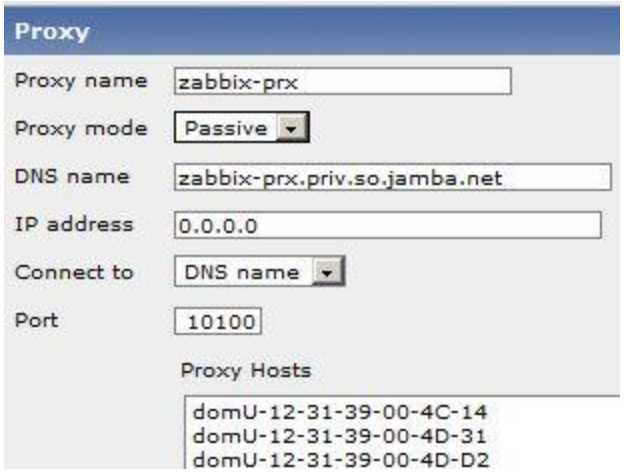
Action conditions

Conditions (A) Proxy = "zabbix-prx"

New | Delete selected

Zabbix Server monitoring in a public cloud

- the instance is created based on the registration action
- all templates are assigned
- the monitoring is started automatically
- when the instance shuts down (regularly), a shutdown signal is sent to a self-written server on the proxy machine
- using a cronjob the Zabbix server queries all unregistered instances on the proxy machine and disables them with a Zabbix API call
- Drawbacks:
  - crashed instances cannot be unregistered
  - no host availability checks
  - no historical data usage due to often VM recreation



Summary

- Zabbix can be an excellent tool for monitoring a huge Enterprise Java Platform
- just a question of the agent (native vs. Java)
  - JMX and Zabbix are a good marriage
    - easily extensible for custom checks
    - many monitoring tools cannot speak JMX
    - avoid remote calls on the same machine

- performance problems could be successfully tackled with the help of the support
  - learning curve was really long and exhausting
- the api is an appropriate way to integrate Zabbix with other systems
  - reduce the number of tools for the operations team
  - at least in 1.8 the api lets place for improvements
- cloud monitoring can be done using some simple workarounds

Q & A