Monitoring Guide for SAP NetWeaver CE Developers

Applies to:

SAP NetWeaver CE 7.1 EhP 1. For more information, visit the Java homepage.

This guide provides introduction to the best practices like processes and implementation specifics of doing monitoring in SAP NetWeaver CE 7.1 EhP 1

Author: Hristo Dobtchev

Company: SAP Labs Bulgaria

Created on: 01 June 2008

Author Bio

Started at SAP in 2004. A project lead of "CE Administration and Monitoring" project and one of the authors and project lead of the “Application-centric Administration” project. He provides proactive governance of SAP NetWeaver monitoring content and is also an author and early driver of advanced SAP NetWeaver monitoring dashboard: AS Java Overview. He successfully delivered the monitoring helper plug-in for NWDS in NetWeaver CE and is an author and contributor to the end-to-end traceability project. Hristo Dobtchev supported SAP leadership in Enterprise Java standards (JEE 5) through adoption of critical low level technology (byte code modification library).

Table of Contents

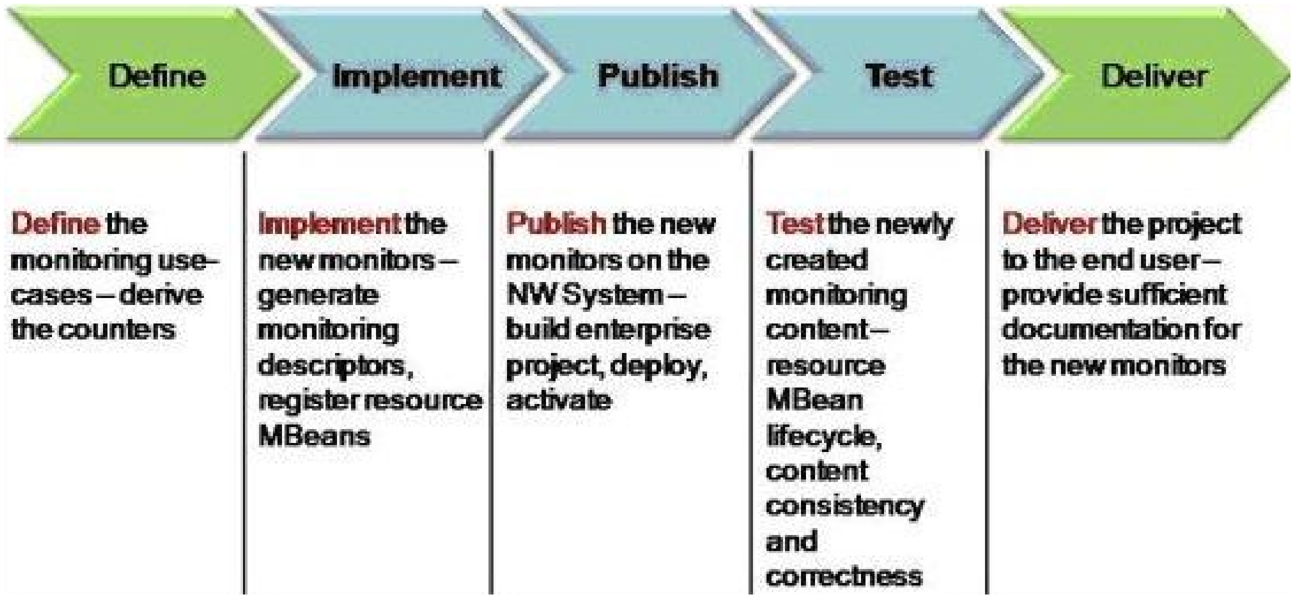
Introduction

The monitoring is very important process in the day to day work of a system administrator. It ensures him that the maintained functionality works as intended and if there is a problem it will be discovered early enough so that the affected end user base is limited. Monitoring can also help in sizing different systems by giving information about the load during specific timeframes and the scalability related to the number of end users.

The monitoring process can be reactive (alerting) or proactive. In the first case the process is triggered by the system which sends a message (an alert) to the administrator whenever a problem is detected. The second starts with the administrator looking at a dashboard or charts with monitoring data in the morning or during operations (timeframes) with heavy user load on the system and during the initial problem analysis of a known (alerted) problem.

The Java stack of NetWeaver CE provides monitoring infrastructure based on JMX that enables application developers to define and publish monitoring data for their own applications.

Monitoring Development Lifecycle


Define
Implement
Publish
Test
Deliver
Define the monitoring use-cases – derive the counters
Implement the new monitors – generate monitoring descriptors, register resource MBeans
Publish the new monitors on the NW System – build enterprise project, deploy, activate
Test the newly created monitoring content – resource MBean lifecycle, content consistency and correctness
Deliver the project to the end user – provide sufficient documentation for the new monitors

The diagram describes the five phases of a monitor metric creation. All of them are important for creating complete monitoring solution for a specific functionality (application) so it is not advisable to skip anyone. This document will look at specific aspects in different phases of the monitoring development lifecycle.

Defining monitoring content

Choosing the right monitoring content to expose to the end user is probably the most important task in the process of monitor creation and it should be done in the design phase of the project. It is very important to select such monitoring metrics (counters) that will actually be useful for the end user – a system administrator, support team or other developers.

When considering which metrics to expose it is most useful to think in use-cases or scenarios. Imagine a real-life situation with your application and try to find what type of information will improve its supportability. Often it is hard to identify this during development time and the good ideas might come in the integration testing phase when the weak spots in the architecture or some unreliable dependency are identified. Even at this stage it is very important to choose the right content to publish as monitoring information. Such monitoring metrics must conform to some common rules:

- **The exposed information should not be ambiguous -** this means that whatever content you choose to publish as monitoring it should provide consistent and correct information that leads to specific conclusions in the follow-up analysis.

- **Only “fresh fish” please** - the data exposed should be up to date. Nobody will get any benefit from an outdated value. Just the opposite – it might confuse the user even further.
- **Take care of the performance** - don’t do extensive monitoring calculations that will affect seriously the performance of the application. After all, the monitoring should be used for improved supportability and the performance of the main functionality must not be impacted by it.
- **Make sure it will be used** - creating monitors that are not clear to the end user or do not derive from a real use-case is just a waste of time for both the developer that implemented them and the end user.
- **This is no tracing** - don’t assume the monitoring process as tracing or debugging. The last have different objectives and users. Monitoring main scenarios are “alerting”, “initial problem analysis” and in some cases “server capacity planning” which usually are oriented towards the system administrator. The “tracing” and “debugging” are in the support engineers and developers capacity. These activities are used for expert error analysis and are not part of the daily activities done in productive environments by system administrators.

Monitoring Content Definition Template

The template below helps with the last two rules. It defines the main questions that everyone needs to answer for himself in order to find out if a metric makes sense to be implemented. Try to fill it out for each metric that is considered worth implementing.

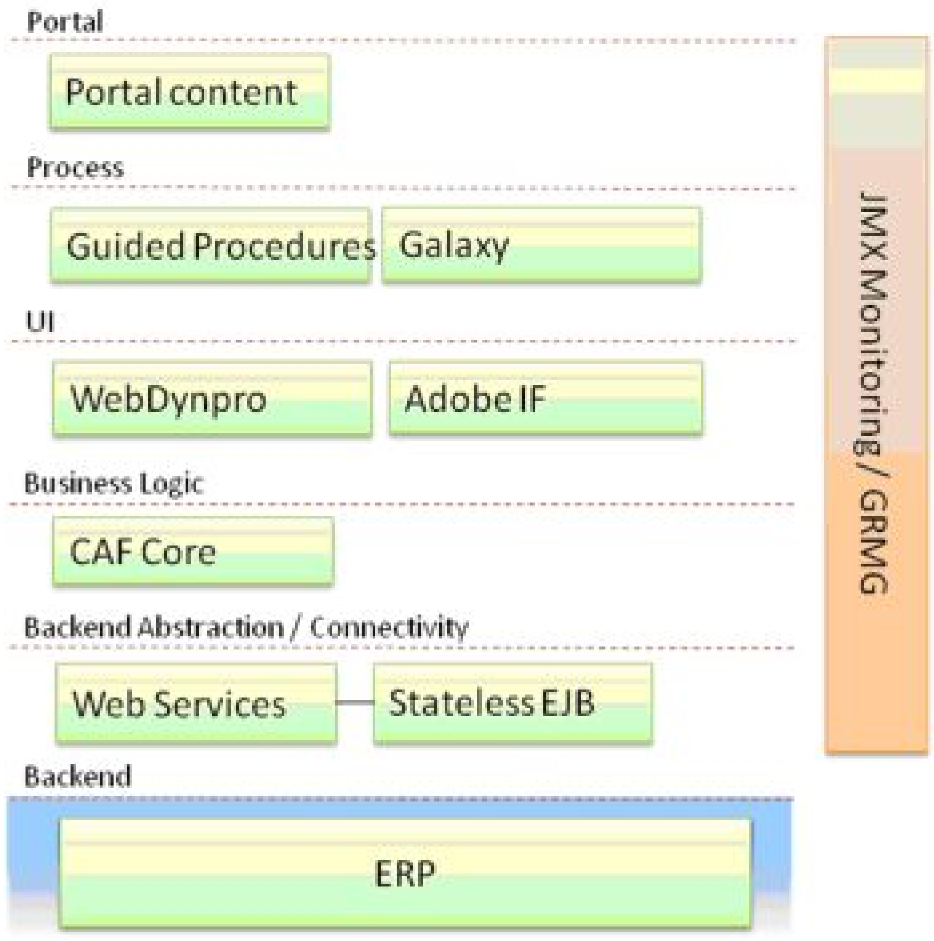
- **Description of the metric**
  Describe the metric in details: What is the source metric? How it is gathered – what is its semantic in general?
- **Target group (administrator, support, developer, etc.)**
  Provide information whom this metric is addressed to. Is it for the **system administrator** (the best option as it is monitoring primary target user) who will maintain the functionality every day at the customer side, the **support engineer** who will access the system in order to analyze a problem that the administrator cannot handle by himself or the **developer** who needs to check a specific counter during testing so that he can analyze the performance of his program. This is the worst target user for a monitor as such implementation will make no sense after the implementation is finished and some tracing or debugging tools might be much more appropriate for such a task.
- **Description of possible problems and their impact**
  What are the possible problems that can be detected with this metric? Describe the specific use cases when the metric indicates some availability or performance issues and the impact these problems might have on the overall application performance.
- **Boundary conditions for the metric (thresholds)**
  What are the boundary conditions of the monitoring metric at which an alert will be raised? There are three alert states: green (OK), yellow (warning) and red (error). Specify if such thresholds can be set and provide the initial values for them (they can be changed at later stage from the end user but it is highly advisable that you provide values that are as close to reality as possible). It is recommended that all monitoring metrics support setting of thresholds as this will mean that they can be used for configuring alerts that can be used by the end user to reactively monitor the system (the user doesn’t look at the metric unless there is a problem). Reactive monitoring is the preferred method of monitoring for the system administrators.
- **Generate alerts. Alert actions.|**
  What is the semantic of alerts raised by this metric? What actions should the user take when he sees a specific alert?
- **Recommendation how to proceed when some problem occurs**
  If there is no way to set an alert then provide information on how the user must proceed in case of detecting a problem with this metric such as how this metric should be used in a real use case scenario.
- **Raw metrics**
  What are the raw metrics (if it is a calculated value) that were used to generate this metric? By

explaining this it will be easier for the end user to understand the metric semantics and therefore it will be less likely for him to draw the wrong conclusions.

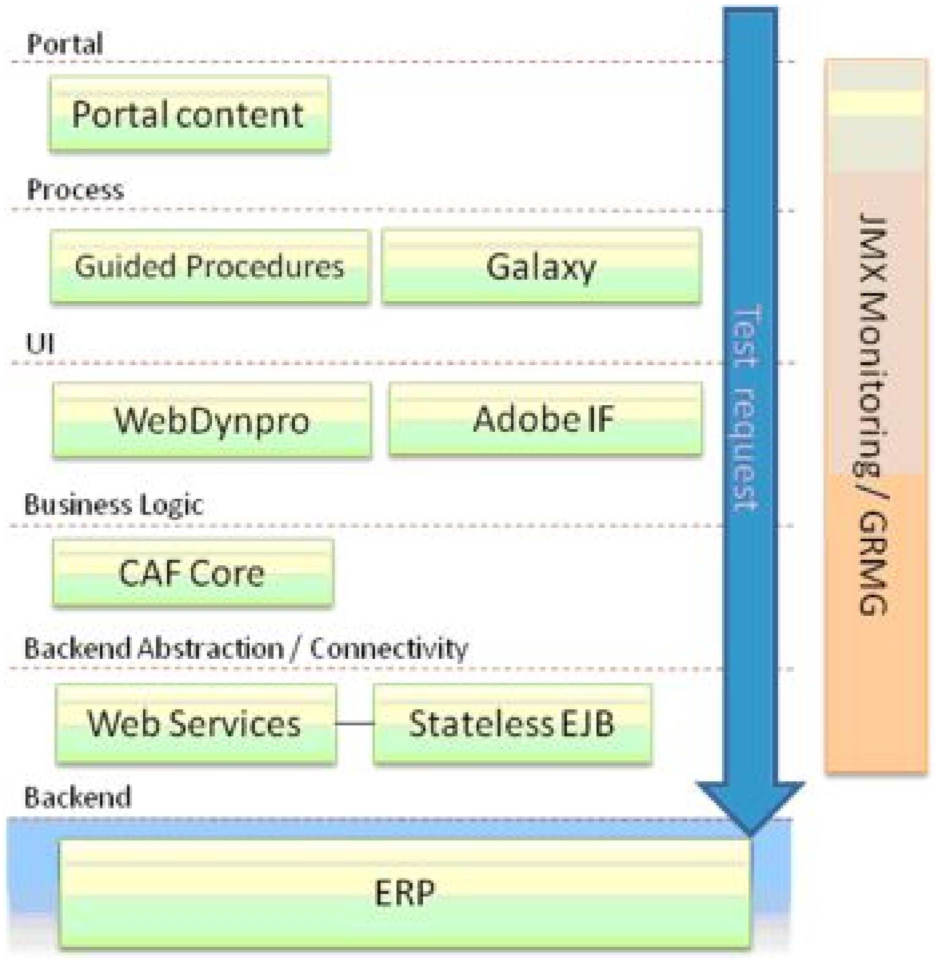
- **Units**
  What type of units this metric is measured in? KB, MB, sec, requests/min, etc.
- **Dependencies (to other metrics)**
  Very often the metric value by itself is not informative enough. In a larger scenario it is required that the end user checks other metrics so that he gets the right perception of the situation. For example, high memory consumption might be a result either of a lot of users working on the system or of a process that takes too many resources during its execution (a possible leak). In order to understand the situation the end user might need to check the active web sessions metric in addition to the memory consumption one.
- **Level (cluster/instance/node)**
  Some metrics are relevant to a single server node. Others are instance specific (e.g. “ICM incoming queue size”) or work on cluster level (e.g. “NW license expiration”).

Instrumentation Techniques

There are generally two approaches to monitoring instrumentation of an application. The first centers on monitoring the different components or layers of the application architecture. For example the typical layered architecture of a composite application is the following:



It is possible to report metrics from the different layers and components. The aggregated state of all these metrics should be sufficient to give impression on the availability and performance of the functionality. This might be true on theory but in reality it is often the case that all layer or component metrics report values that are in the normal range but the application functionality is not available or the performance is bad. This might happen as there can be a dependency that is not available or some configuration options are not set correctly (inconsistent configuration). So in this case testing the interaction between several layers might produce a better content.



Implementation of a New Monitor

The easiest way to implement a new monitor is by using the NWDS monitoring helper tool (http://help.sap.com/saphelp_nwce711/helpdata/en/48/57af6a825658d7e10000000a421937/frameset.htm) However, for some more advanced monitoring setups the tool might not be sufficient and manual coding might be required. Please, read the JMX monitoring infrastructure documentation in order to get acquaint with it before you start developing new monitor http://help.sap.com/saphelp_nwce711/helpdata/EN/65/54b24188b3e534e10000000a1550b0/frameset.htm.

Things to Consider During Implementation

There are some infrastructure specifics that are good to consider when implementing a new monitor.

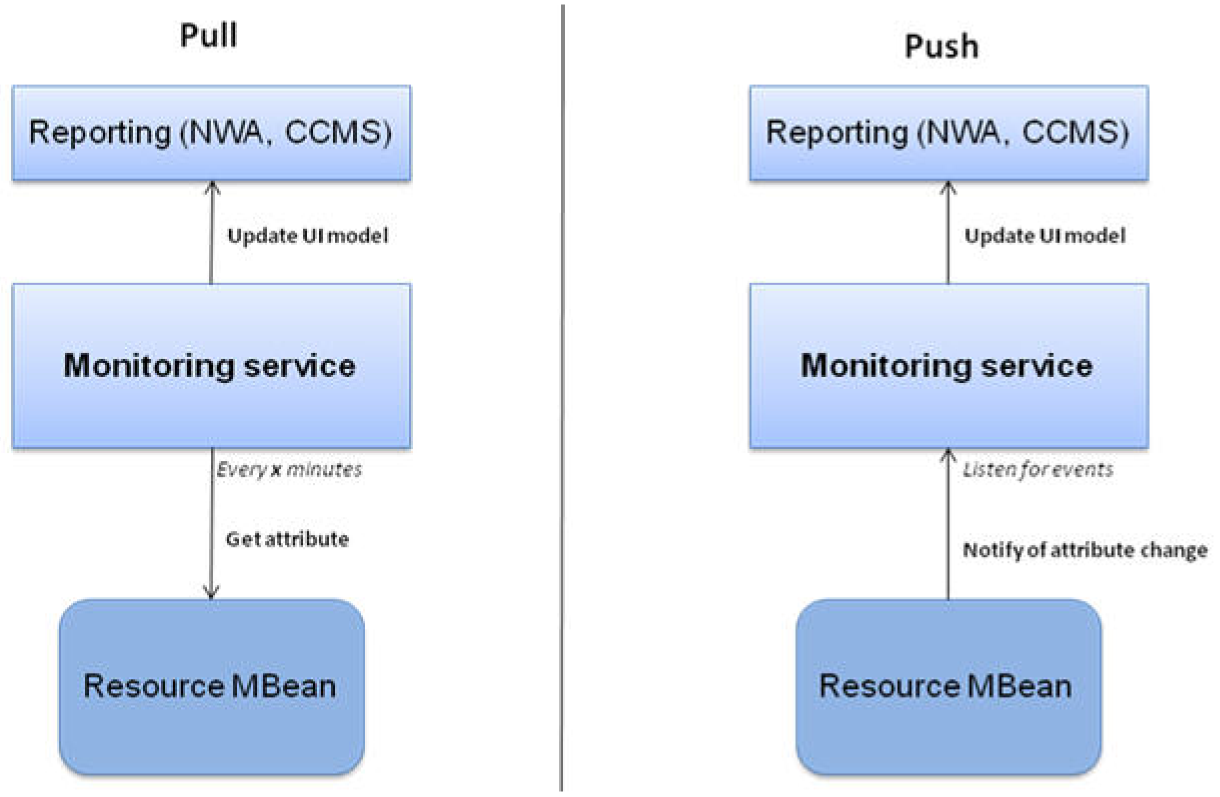
Project Setup

Note: It is imperative that the lifecycle of the monitor is the same as the resource lifecycle!

In large projects with numerous DCs (the composite applications case) it is possible to place monitoring implementation in a separate DC. The benefit is that it gives better control over the resource MBeans lifecycle (register/unregister to MBean server). At the same time it provides a context base for stateless objects that cannot be bound directly in the MBean server (e.g. stateless EJBs) <<link to stateless objects chapter>>. Another benefit is that this way an improved supportability is achieved so that the developer does not need to browse all DCs in order to find the necessary implementation and it is much clear from architecture point of view. Usually such “monitoring DC” will be an enterprise web application and there will be a servlet whose init() method is used for the MBean registration.

However in this setup it means that hard references from the monitoring DC to the resource DCs have to be set. But if the resource DCs have different lifecycle it might happen that the monitoring DC gets stopped when one of the resource DCs stops. But this might be incorrect as the other resource DC might still be up and running. So it really depends on the actual use-case what will be the best project setup.

Push or Pull



The **pull monitors** are the most widely used once as in this case the data is collected in a monitoring service thread from which the resource MBean is called on periodical intervals. Other pros of this approach are that the period of data collection is configurable in the monitoring UIs (NWA Java System Reports – Monitor Browser) and the monitor can be deactivated (will not pull data and load the system anymore). One drawback of the pulled monitors is that the data cannot be retrieved on intervals shorter than one minute and it is possible that important values are missed between the calls. Most of these problems can be resolved by aggregating the monitoring data returned from the resource but this will require additional logic in the resource implementation and can be tricky.

The **push monitor** updates the monitoring node by sending JMX notification on value change. This is very useful if immediate update is required so that important values are not lost during calls as it happens with pull monitors. However, there might be a severe performance impact on the execution as the update happens in the same thread so until it is done the execution cannot continue. That is why it is not recommended to create push monitors for metrics which are often updated. It is best to consider such monitor only if the exact time of the last metric update is very important or if such updates happen seldom. Another reason for choosing a push monitor is in the cases where it is important the resource to have control on the exact metric update timestamp.

Resource Bean Lifecycle

When creating pull monitors it is important to take care about the lifecycle of the resource MBean (the one that is invoked by the monitoring service in order to collect the metric data). The MBean lifecycle should follow the lifecycle of the monitored resource. In other words the MBean should be registered when the resource is up and running and unregistered when the resource goes down. In some cases the MBean might still be registered in the server when the resource goes down. In this case it will contain the last data returned from the resource. It is up to the developer to decide based on the business logic when to register and unregister the resource MBean. The monitoring service has two modes (configurable per monitor) for reaction on failure (resource MBean not found or throws exception) – “unregister” and “ignore”. The first option means that the monitoring node (in the tree) will be removed and the monitor will be unregistered when failure to reach it occurs. The last option means that the monitoring service will try again to reach the resource after the refresh period passes.

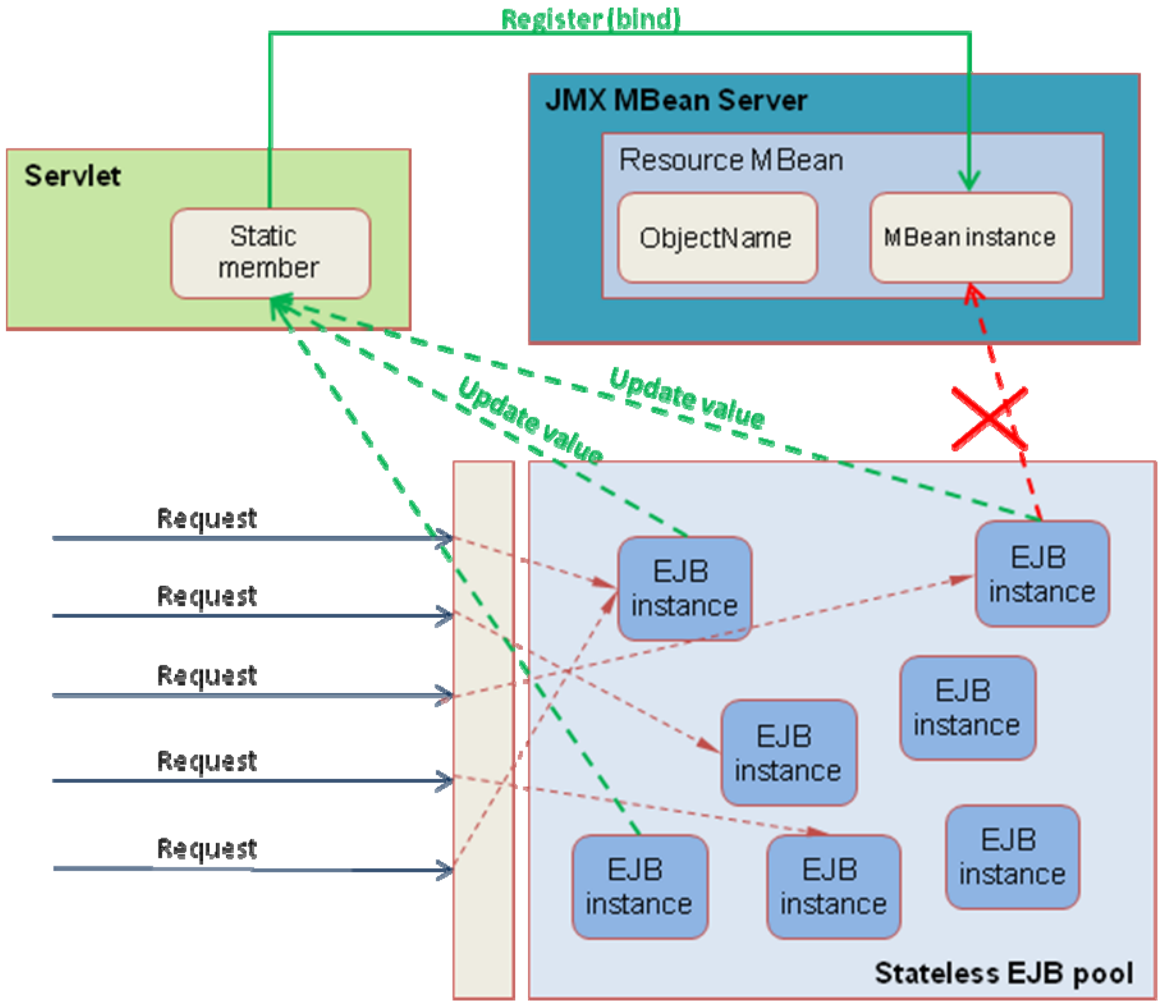
Performance Impact

Some monitoring metrics might require intensive calculations and data gathering. A developer should be very cautious when creating such metric as they can slow down the application execution significantly and even to impact on the performance of the whole system. This might happen if such metric is calculated too often and if someone configures the refresh rate of the monitor on very short intervals. That is why it is recommended for such metrics to be implemented as a push monitor so that the data collection periods are controlled by the resource rather than the administrator of the system. In this case the performance impact of the JMX notification sent in the application thread will be negligible on the scale of the overall time for metric generation.

If a pull monitor is used it is important that the call used to retrieve the metrics does not execute the metric generation code directly but rather gets a cached value which is updated when the developer finds it appropriate.

Stateless Objects (EJBs)

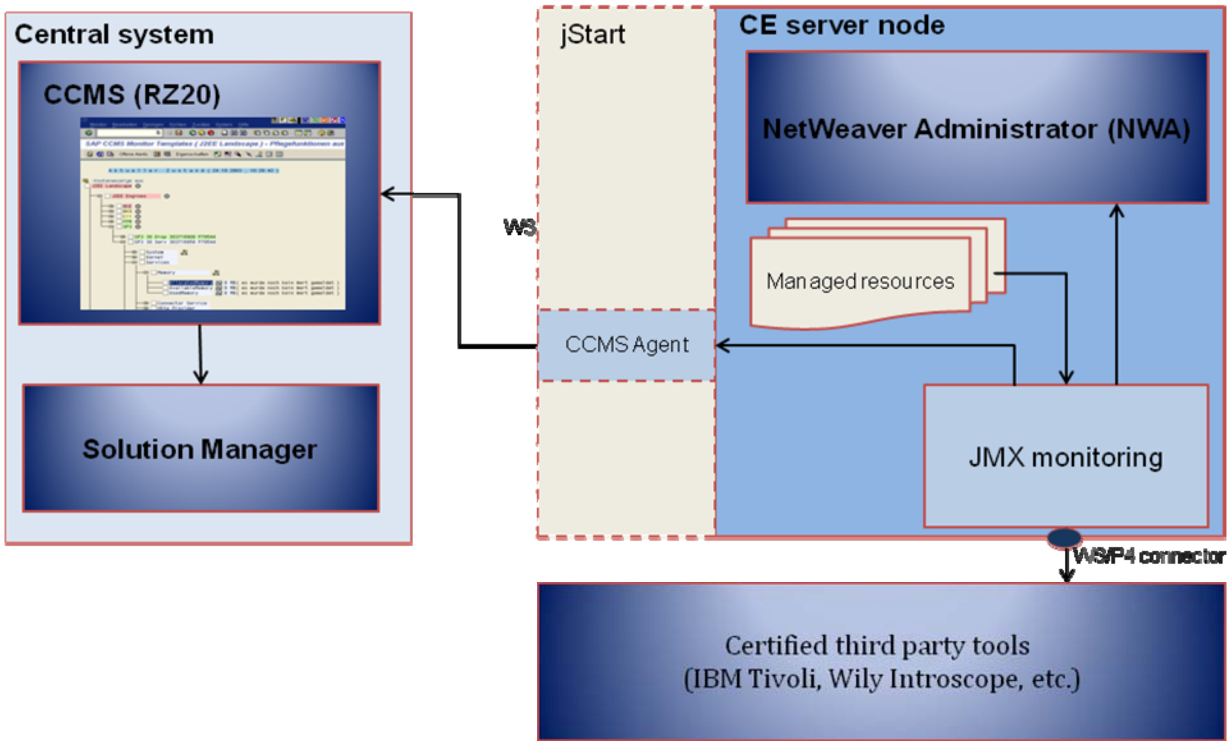
In some technologies the code is often executed in stateless objects. Such objects are not appropriate to direct binding in the JMX server as resource MBeans. For example code that executes in a stateless EJB cannot use the same object for a resource MBean implementation as the stateless EJBs come from a pool with multiple instances and are provided by the container on request. Binding such object into the MBean server will accomplish nothing as on the next request another instance might be used and the one that is registered as a resource MBean will not be updated.



In such case it is recommended to have a statefull object that can be reached (and updated) from the stateless object code. Such statefull object might be a static member of the monitoring servlet class (if you have a separate web application for monitoring). The servlet class can have static members that contain the resource MBeans objects and should expose static methods for update of their values. The code executed in the stateless EJB on its side can trigger updates using static calls to the servlet class.

Publish and Activate

Any new monitor is published into the JMX monitoring infrastructure once the deployable project is build and deployed. However, the monitor is not active by default. It means that the node is shown in the local monitoring tree but no values are pulled from the resource (this is irrelevant for push monitors though). The inactive node is not reported centrally as well. In order to activate the new metric the user should open NWA Java System Reports professional application, navigate to “Monitor browser” in the reports combo box and then find the monitor node in the displayed tree. Once selected the UI allows the user to *enable* it. Enabled (or active) monitor nodes start to collect data from the resource and are also reported to the central management system if the local system is part of a larger landscape. They can also be used in third party tools like Wily Instroscope and Tivoli.



Some monitor metrics are active by default (on fresh engine install). If you believe that your metric needs to be enabled by default you have to contact the monitoring team at SAP.

Testing and Quality

It is critical for the success of a monitor that the provided data is accurate. Once the administrator sees a metric that is wrong he will lose any thrust in it and will probably never use this monitor in his future activities. So it is very important the developer to be sure what is delivered as monitoring solution actually works.

Testing monitoring metrics is a very tricky process as it is not always possible to reproduce all the situations that might occur at customer side. It is important to create as many tests as possible where an error condition happens and the metric should alert the end user about it. In order to limit the error reporting it is possible to introduce a verification code in your resource MBean that checks whether the metric value is in a reasonable range that makes sense. Once extreme and incorrect values are detected an error log can be written and the metric value reset. That way the end user will see less wrong content and there is higher chance that he keeps trusting that content.

Once delivered an application monitoring should be backwards compatible. This means that already exposed metrics should not be removed or their semantics completely changed. It comes from the fact that once delivered the application is included in the IT processes at the customer side and it is very likely that these monitoring metrics will be used for availability reporting or even be part of SLA agreements. So any drastic change in such metric might result in bad feedback from the customers.

Delivery

The monitoring implementation should be delivered together with the monitored resources implementation. It is also very important to provide appropriate documentation for the exposed monitoring metrics so that the user at customer side can learn about them and decide whether he wants to activate them and add them to his daily reports.

Even the best metric appears useless when it is not described properly. The monitoring descriptor allows the developers to provide information (description) that will be displayed in the monitoring UIs. It is very important to provide good descriptions for all monitoring metrics that are delivered to the end user.

Related Content

Java Application Monitoring

http://help.sap.com/saphelp_nwce711/helpdata/EN/65/54b24188b3e534e10000000a1550b0/frameset.htm

NWDS monitoring helper tool (NWDS documentation)
http://help.sap.com/saphelp_nwce711/helpdata/en/48/57af6a825658d7e10000000a421937/frameset.htm

For more information, visit the Java homepage.

Copyright