Plug-In Development mit dem Oracle Enterprise Manager 12c

Gunther Pippèrr
München

Schlüsselworte
Oracle Enterprise Manager – Plug-In Entwicklung

Einleitung

Wie kann eine eigene Lösung für das Monitoring von Applikationen mit den Enterprise Manager 12c erstellt und entwickelt werden?

Im Vortrag wird das generelle Konzept der OEM Plug-Ins im OEM 12c vorgestellt und die eigenen Erfahrungen und Grenzen beim Entwurf von Plug-Ins für eine größere Oracle NoSQL Umgebung aufgezeigt.

Das Ziel ist es über die Fallstricke in der Anfangsphase eines eigenen Projektes zu informieren, um eine schnelle Umsetzung eigener Ideen ohne allzu große Hürden zu ermöglichen.

Architektur Enterprise Manager 12c

Seit Oracle 12c setzt Cloud Control auf eine konsequente Plug-In Architektur mit der Möglichkeit der deklarativen Programmierung für die Entwicklung eigener Erweiterungen.

Der große Vorteil in dieser Architektur liegt darin, dass sich nun die Überwachungssoftware für einzelne Targets updaten lässt, ohne jedes Mal auch gesamten Enterprise Manager patchen zu müssen.

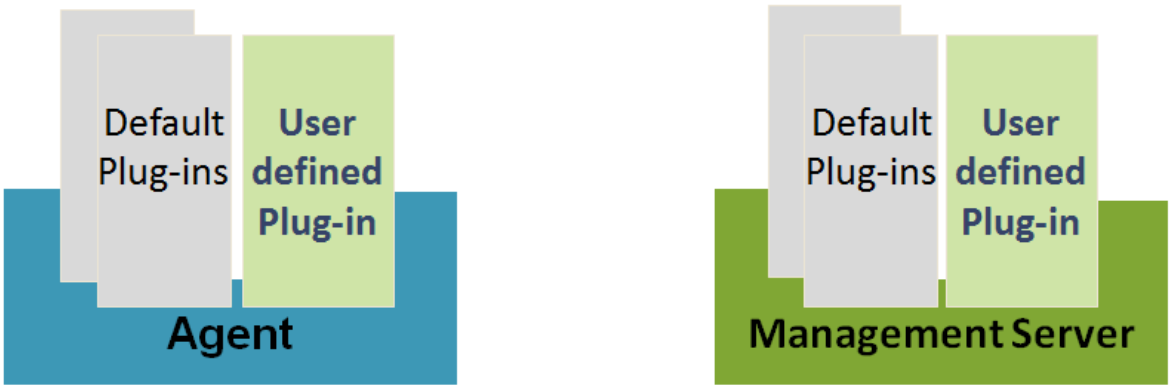

Abb. 1 : Architektur Übersicht

Das Aufsetzen einer Entwicklungsumgebung

Da die Entwicklung eines OEM Plug-Ins in weiten Teilen deklarativ über sehr viele unterschiedliche XML Dateien erfolgt, die zum Teil redundanten Informationen enthalten müssen, ist es sehr hilfreich sich zu Beginn vom Projekt eine eigene, mit Skripten automatisierte Entwicklungs-Umgebung aufzusetzen.

Auch müssen die XML Dateien des Plug-Ins in einer von Oracle festgelegten Ordnerstruktur abgelegt werden.

Ein kleiner Bug in der Namensgebung oder der Strukturierung der Daten führt leicht zu sehr schwer zu erkennenden Fehlern.

Da der OEM in meiner Umgebung unter Linux betrieben wird und auch das Plug-In für die Linux Plattform entwickelt werden soll, wird die Umgebung auf dem OEM im Oracle Home Verzeichnis in entsprechenden Unterordnern aufgebaut.

Das EDK (Enterprise Manger Extensibility Development Kit) für die Entwicklung kann vom OMS heruntergeladen werden und wird dort in ein eigenes Verzeichnis ausgepackt.

Da eine Java Umgebung für das EDK benötigt wird, wird das Java Home aus dem OMS Home referenziert, damit ist auch immer die richtige Java Version in Verwendung.

Folgende Struktur hat sich bei der Entwicklung bewährt:

- **EDK Verzeichnis** „/home/oracle/**plugin_dev**/edk**“**
- **Plug-In Development** „/home/oracle/ **plugin_dev/nosql_plugin_oms12c“**

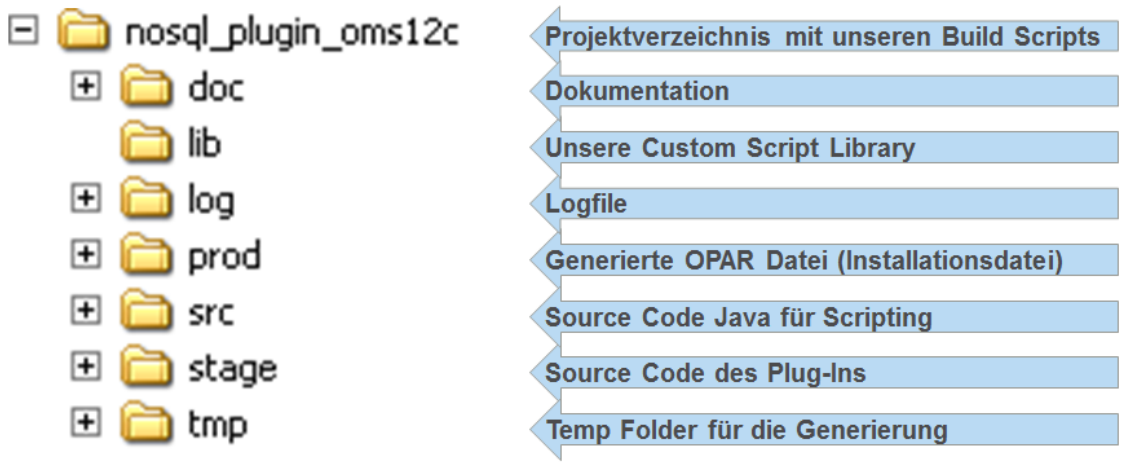

Abb. 2 : Unsere Entwicklungsumgebung

Unter dem Verzeichnis „**stage**“ erfolgt der Aufbau des Dateibaums für die XML Beschreibung des Plug-Ins.

Dies muss sehr sorgfältig erfolgen, jeder noch so kleine Fehler hat oft viele, schwer nachzuvollziehende Seiteneffekte zur Folge.

Der Baum besteht im Prinzip aus drei Ästen, dem **Agent**, dem **OMS Server Part** und dem **Discovery** Skripten.

Alles unter dem Verzeichnis „**agent**“ wird später zu dem Plug-In Bestandteil des Agenten auf dem Ziel Target übersetzt.

Alles unter dem Verzeichnis „**oms**“ definiert die Bestandteile für das zentrale Repository und die Darstellung des neuen Target Typs in der OMS Oberfläche.

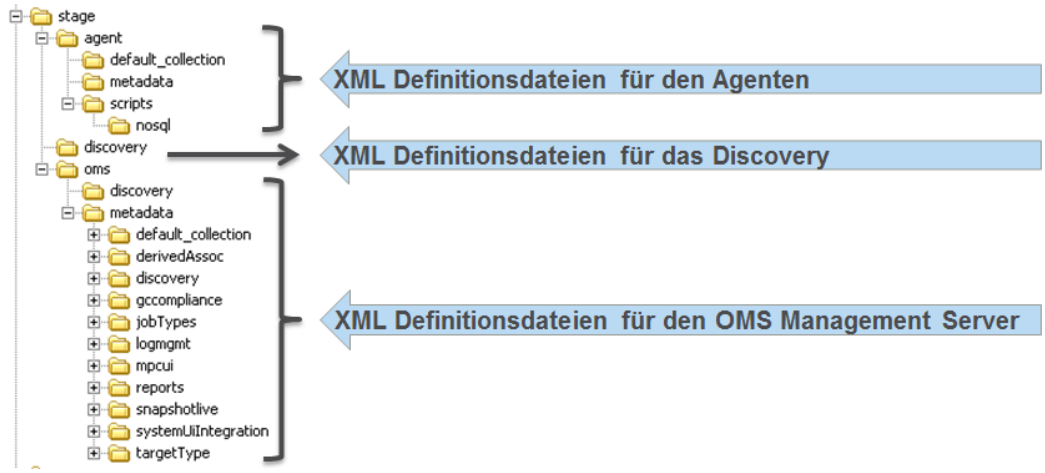

Abb. 3 : Dateistruktur für die XML Dateien eines Plug-Ins

Für das Erzeugen, Importieren und Patchen stehen verschiede Befehle des EDK zur Verfügung, es empfiehlt sich diese zu verskripten und dazu eigene Schell Skripte im Projektverzeichnis abzulegen.

Die Bedeutung der Namensgebung für die Plug-In Entwicklung

Die Namensgebung des Plug-Ins ist entscheidend für den Erfolg bei der Entwicklung und muss streng eingehalten werden, um böse Überraschungen zu vermeiden.

Es ist zum Empfehlen für den „Target Type“ einen Namen zu wählen der möglichst nicht von Oracle selbst verwendet wird. Leider vergibt Oracle hier die Namen nicht zentral, jeder Entwickler kann vergeben was immer er möchte.

Da es aber nur einen "Target Type" im gesamten System geben kann, ist die Gefahr von Kollisionen mit anderem Partner Plug-In Entwickler durchaus gegeben. Wir hatten im ersten Schritt den Namen „**NoSQL**“ gewählt, besser ist aber wohl eine Namensgebung wie „**GPIOraNoSQL**".

So habe ich zum Beispiel zu Beginn der Entwicklung für das Attribute „pluginTag“ den Wert „**v**001“ gewählt, das ging dann bis zur OEM Version 12.1.0.3 gut, ab der Version 12.1.0.4 ließ sich das Plug-In dann nicht mehr übersetzen, da in diesem Attribut ein „**x**“ als erster Buchstabe zwingend vorgeschrieben ist, d.h. das Attribut „pluginTag“ muss den Wert „**x**001“ erhalten!

Dieses kleine und auch nicht wirklich dokumentiert Feature hat dann dazu geführt, das ein Update des Plug-Ins nicht mehr möglich war und alle Targets gelöscht werden mussten, um die nächste Version einspielen zu können.

Eindeutig identifiziert wird ein „Partner“ Plug-In über die „Plug-In ID“

Die Plug-In ID besteht aus:

- Vendor ID (8 chars) = **gpiconsl**
- Product ID (8 chars) = **nosql**
- Plug-In Tag (4 chars) = **x001**

Der vollständige Plug-In Name lautet damit „**gpiconsl. nosql. x001**“

Auch muss ein Kürzel für den Target Type vergeben werden wie: **nosql**

Neben dem Namen und dem Target Type ist auch die Versions-Nummer von entscheidender Bedeutung.

Jedes "Partner Plug-In" (Plug-In von Kunden Hand) muss einer festgelegten Versionsnummer Syntax folgen, die Position ist hier besonders wichtig, bei jeden Deployment auf dem OMS muss diese um eins hochgezählt werden, damit auch die bestehende Version aktualisiert werden kann.

Plug-In Version definieren: **a.b.c.d.e**

- **a.b**= Die Version des Enterprise Manger Extensibility Development Kit (EDK), dass bei der Entwicklung verwendet wurde (12.1, 12.2, usw.).
- **c** = Muss immer eine 0 sein
- **d** = Die aktuelle Version des Plug-In, vom Entwickler vergeben
  - können 1 oder 2 digits ([1-9] | [0-9][1-9]) sein
  - Bei jedem Update muss das angepasst werden.
- **e** = Für den zukünftigen Einsatz - Default wert ist 0

Die Plug-In Version lautet damit für die erste Version: **12.1.0.01.0**

Wichtig ist es das die letzte Stelle NICHT verwendet wird!

Die wichtigsten XML Dateien erstellen

Am einfachsten ist es, ein bestehendes Sample Plug-In als Vorlage zu verwenden und von dort die entsprechende XML Dateien in die eigene Verzeichnis Strukturen zu kopieren.

Die wichtigsten Dateien sind:

- plugin.xml = > Namen und Version des Plug-Ins, die unterstützen Plattformen

Agent Part – wird auf dem Agent „ausgeführt“

- **agent**\plugin_registry.xml => Target Typen, die diese Plug-In unterstützt definieren
- **agent**\metadata\ nosql.xml => Definition der Metriken für diese Target
- **agent**\default_collection\ nosql.xml => Definition der Collections für das Target (sammeln von Metadaten wie Version etc.)
- **agent**\scripts\nosql*.* => Die notwendigen Bash, Perl Skripte und Java Klassen für die eigentlichen Metriken

OMS Part – wird im Repository und in der Oberfläche „ausgeführt“

- **oms**\Plug-In.xml = > Namen und Version des Plug-Ins, die unterstützen Plattformen
- **oms**\metadata\targetType\ nosql.xml => Kopie der Metrik Definition aus agent\metadata\
- **oms**\metadata\mpcui\ metadata_ui_homepage_charts.xml => Definition der Home Page in der OMS Oberfläche
- **oms**\metadata/default_collection\ nosql.xml => Kopie von agent/default_collection/*.xml

Discovery Part – Automatisches Erkennen des Target Typs auf einem Host

- Plug-In_discovery.xml => Aufbau des Discovery Moduls
- discover\nosql_discovery.pl => Perl Script für das Erkennen des Targets
- lib*.jar => Lib’s, zum Beispiel eigene Java Klassen

In allen Dateien sehr sorgfältig auf die richtige Versionierung und Benennung der Target Typen achten!

Für die entsprechenden Beispiele, siehe die Screenshots in der Präsentation.

Metriken für den Target Type erstellen und bekannt geben

Nach der Definition der Hülle des Plug-Ins können wir als erstes die Response Metrik definieren, ein Target sollte mindestens diese Metrik implementieren. Über die Status Metrik wird definiert ob das Target „up“ oder „down“ ist.

Dazu editieren wir die Datei „agent\metadata\ nosql.xml“ und fügen eine Metrik vom Namen „Response“ ein, bzw. editieren diese entsprechend.

Für die Implementierung einer Metrik wird ein sogenanntes **Fetchlet** verwendet. Ein Fetchlet ist ein Anweisungsblock, der definiert mit welcher Technologie und welchen Parametern die Daten eines Targets abgefragt werden sollen.

Eingesetzte Fetchlet Typen für unser NoSQL Plug-In:

- FETCHLET_ID="JMX" => JMX Abfragen (JAVA Management Interface abfragen)
- FETCHLET_ID="OSLineToken" => Script Aufrufe (mit Hilfe von frei definierbaren Bash/Perl Skripten wird die Rückgabe verarbeitet)

Für diese Response Metrik verwenden wir den Fetchlet Typ „OSLineToken“, d.h. wir rufen ein Script auf und parsen das Ergebnis.

```
<Metric NAME="Response" TYPE="TABLE">
   <Display>
      <Label NLSID="Response">Response</Label>
   </Display>
   <TableDescriptor>
      <ColumnDescriptor NAME="Status" TYPE="NUMBER"  COLUMN_NAME="Status" IS_KEY="FALSE">
         <Display>
            <Label NLSID="Status">Status</Label>
            <Description NLSID="StatusDescription">
                Status of the Oracle NoSQL Node - whether all nodes up or one is down
            </Description>
         </Display>
      </ColumnDescriptor>
   </TableDescriptor>
   <QueryDescriptor FETCHLET_ID="OSLineToken">
      <Property NAME="scriptsDir"  SCOPE="SYSTEMGLOBAL">scriptsDir</Property>
      <Property NAME="command"     SCOPE="GLOBAL">/bin/sh</Property>
      <Property NAME="MachineName" SCOPE="INSTANCE">Host</Property>
      <Property NAME="Port"        SCOPE="INSTANCE">DBPort</Property>
      <Property NAME="script"      SCOPE="GLOBAL">%scriptsDir%/nosql/getStatus.sh %Host% %Port%</Property>
      <Property NAME="startsWith"  SCOPE="GLOBAL">nosqlStatus=</Property>
      <Property NAME="delimiter"   SCOPE="GLOBAL">|</Property>
   </QueryDescriptor>
</Metric>
```

Abb. 4 : Beispiel für die Deklaration einer Response Metrik

Die eigentliche Logik verbirgt sich dann im Script „**scriptsDir%/nosql/getStatus.sh**“, in unseren Software Baum unter „agent\scripts\nosql“. Hier wird eine Java Klasse aufgerufen um den Store abzufragen und ein Output im Format „**nosqlStatus=1**“ zu erzeugen.

```
#!/bin/sh
#
# Purpose
# get the status
########## Environment ##############
SCRIPTPATH=$(cd ${0%/*} && echo $PWD/${0##*/})
SCRIPTS_DIR=`dirname "$SCRIPTPATH{}"`

if test -z "$1"
then
   export HOST=$HOSTNAME
else
   export HOST=$1
fi

if test -z "$2"
then
   export PORT=5100
else
   export PORT=$2
fi
#Debug
#echo "get the Parameter :: ${HOST} and ${Port} - use the Script Directory:: ${SCRIPTS_DIR}" >/tmp/log_nosql_agent_script_getStatus

java -classpath ${SCRIPTS_DIR}/kvclient.jar:${SCRIPTS_DIR}/kvstore.jar:${SCRIPTS_DIR}/KVCount.jar com.gpi.oranosql.ShowStatus ${HOST} ${PORT}
```

Abb. 5 : Shell Skript aus dem obigen Beispiel

Nach jeder Anpassung an den XML Dateien, diese neu überprüfen um Fehler zu finden und gleich zu beheben. Im EDK steht dazu der Befehl „empdk validate_plugin” zur Verfügung.

Beispiel:

```
${EMDK_HOME}/bin/empdk validate_plug_in -out_dir $PLUG_HOME/prod -
tmpdir $PLUG_HOME/tmp -stage_dir $PLUG_HOME/stage -conn_desc
${OEMSRV} -repos_user sysman -repos_pwd ${REPOS_PASSWORD}
```

Target Type ausrollen

Sind die ersten Metriken fertiggestellt und getestet, kann die 0'er Version des Plug-Ins auf dem Server ausgerollt werden.

Dazu wird das eigentliche Plug-In als „opar" Datei erzeugt und auf den OMS geladen.

Ablauf:

- Validieren ob alle XML den Oracle Standards entsprechen:
    - ${EMDK_HOME}/bin/empdk validate_plugin …
- OPAR Datei 12.1.0.01.0_gpiconsl.nosql.x001_2000_0.opar für das Deloyment erzeugen:
    - ${EMDK_HOME}/bin/empdk create_plugin …
- OPAR Datei deployen:
    - ${OEM_HOME}/bin/emcli import_update …

Neue Metriken Deployen

Werden neue Metriken entworfen, muss nicht jedes Mal ein neues Plug-In ausgerollt werden. Allerdings muss zwingend ein weiterer Zähler in der metric.xml Datei hochgezählt werden, damit die Änderung auch vom OMS angenommen wird.

Die XML Dateien können auch direkt in den Agenten / auf den OMS hochgeladen werden. Damit lässt sich ein direkter Test durchführen. Für den Agent ist allerdings jedes Mal auch ein Neustart notwendig.

Auch hat sich herausgestellt, dass bei größeren Änderungen nicht alles vom OMS angenommen wird und erst ein wirklich neues Deployment mit einer neuen Version alle Änderungen auch vollständig umsetzt.

Ablauf des Deployment der Metrik Metadaten mit:

1. ${OMS_HOME}/bin/emctl register oms metadata -service targetType …
2. ${OMS_HOME}/bin/emctl register oms metadata -service storeTargetType …
3. ${OMS_HOME}/bin/emctl register oms metadata -service default_collection …
4. Kopieren der Metadaten per Hand in das Plug-In Verzeichnisse mit der richtigen Version vom Agent
5. Agent neu starten

Auch hier bietet es sich an, das ganze über ein Script entsprechend zu kapseln.

Eine Homepage für das Target erstellen

Soll auch auf der OMS Seite die Darstellung des Target beeinflusst werden, zum Beispiel über eine Startseite im Dashbord Layout mit wichtigen Performance Daten im Überblick, stehen dem Entwickler zwei Optionen zur Verfügung. Das Erstellen eines eigenen Flash Plug-Ins oder die Definition einer solchen Seite über eine XML Datei.

Für die Entwicklung eines eigenen Flash Plug-In ist die Flash Software von Adobe notwendig und es müssen genau Vorgaben eingehalten werden. Die XML Variante ist dagegen deutlich einfacher in der Umsetzung und bietet viele Möglichkeiten einer komplexen DashBoard Oberfläche.

Mit Hilfe einer Definitionsdatei im Verzeichnis „oms/metadata/mpcui/“ wird diese eigene Page im OMS für das Target erzeugt.

Damit diese Seite aber auch als Homepage des Targets im OMS angezeigt wird, muss die Seiten ID „**homePg**“ lauten!

Konfigurationselemente des Targets erfassen

Über die „Metric Collections“ können Stammdaten der Targets, wie zum Beispiel die Versions-Nummer der Software, erfasst werden.

Neben dem Einsammeln der Daten werden auch für das dauerhafte Speichern Tabelle im SYSMAN Schema angelegt.

Targets automatisch erkennen lassen

Über einen eigenen Part des Plug-Ins, der „Discovery Section“ , kann definiert werden, wie der Agent erkennen kann ob auf dem aktuellen Host das Target zur Verfügung steht und wie das Target automatisch erkannt werden kann.

Auch hier ist es wieder die Kombination aus Definitionen in XML und Script Elemente, die ein funktionsfähiges Discovery ermöglichen.

Fazit

Mit dem Plug-In Konzept hat Oracle dem OEM in Richtung Kundenfreundlichkeit deutlich geöffnet. Der Kunde hat nun die Möglichkeit beliebige Applikationen und Business Prozesse auch im Enterprise Manager mit zu verwalten.

Eine eigne Lizenz ist dazu nicht notwendig, allerdings ist im Detail zu prüfen, ob der Agent auf „nicht“ Oracle Hosts entsprechend lizensiert werden muss.

Leider wird noch nicht ein „Kunden Cluster Target“ unterstützt, auch das Verknüpfen von Targets über die Topologien könnte etwas besser dokumentiert werden.

Die Entwicklung selber ist aber mit Basis Kenntnissen in der Skript Programmierung gut durchführbar.

Die Dokumentation ist leidlich gut und mit den Beispielen der andern Plug-Ins können zügig erste Erfolge erreicht werden.

Kontaktadresse:

Gunther Pippèrr
GPI Consult
Schwanthalerstr. 82
D-80336 München

Telefon: +49 (0)89 53 026 418
Mobil: +49(0)171 8065113
E-Mail gunther@pipperr.de
Internet: http://www.pipperr.de/dokuwiki/doku.php?id=dba:oms_12c_plugin_development