ESA plugin

## Document info

Auteur: Wilco Koorn, Xebia IT architects

Datum: 23 aug 2013

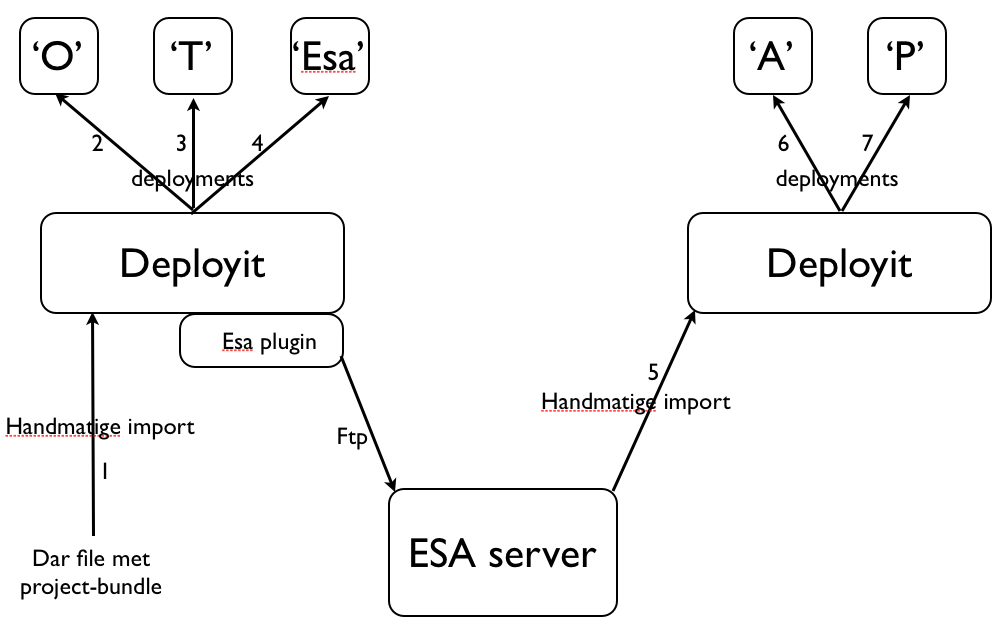
Status: Draft

# Informele Beschrijving

Omdat UWV de met BeInformed gemaakte software aan moet leveren aan IBM via “ESA” (Eenduidige Software Aanlevering) moet UWV BeInformed project-bundles aanleveren in het ESA formaat en conform ESA naamgeving standaarden. Omdat dit geen out-of-the-box Deployit functionaliteit is moet er een Deployit plugin gemaakt worden die dit regelt. Dit document beschrijft deze plugin.

De verwachting is dat BeInformed project-bundles op de ‘A’ en ‘P’ omgevingen met Deployit gedeployed gaan worden. Daarom moet het aan ESA aan te leveren pakket een file bevatten die direct in Deployit is te importeren: een zogenaamde ‘dar’ (Deployment ARchive).

Figuur 1 bevat het conceptuele overzicht hoe een project-bundle aan ESA wordt aangeleverd. Merk op dat dit wordt geïmplementeerd middels een deployment naar de dedicated ‘Esa’ omgeving.



**Figuur 1.** Overzicht van deployments met tussentijdse oplevering aan ESA.

De project-bundle wordt via een ‘dar’ file geïmporteerd in eerste Deployit instantie (links in Figuur 1) in stap -1-. Nu kan er vanuit de Deployit GUI gedeployed worden naar de Deployit Environments ‘O’ in stap -2- en ‘T’ in stap -3-. Voor oplevering naar ESA deployed men in de Deployit GUI naar de ‘Esa’ Environment in stap -4-. De Esa plugin zorgt dat het bestand dat naar de ESA server (onderin Figuur 1) wordt gekopieerd in ESA formaat is en aan de ESA standaarden voldoet.

Vervolgens pakt een TAB-er het ESA bestand op, pakt het uit, treft een ‘dar’ file aan die direct weer in de tweede Deployit instantie (rechts in Figuur 1) kan worden geïmporteerd in stap -5-. Nadat dit gebeurd is kan de project-bundle op ‘A’ of ‘P’ worden gedeployed middels de GUI van de tweede Deployit instantie in stap -6- respectievelijk -7.

Merk op dat de dar file die geïmporteerd wordt in stap -5- gelijk is aan de dar die geïmporteerd wordt in stap -1-. Deze gelijkheid wordt door de ESA plugin gegarandeerd.

# Object types

In de ESA plugin zijn de volgende object types gedefinieerd. Deze worden hieronder uitgebreid besproken.

|  |  |  |
| --- | --- | --- |
| **Deployable** | **Container** | **Deployed** |
| bi.projectbundle | uwv.esaclient | uwv.copiedprojectbundle |

**Tabel 1.** Overzicht van object types in de ESA plugin.

## Hulp types

De ESA plugin kent een hulp object type. Ook deze wordt hieronder nader besproken.

|  |  |
| --- | --- |
| **Hulp type** | **Rol** |
| esa.AbstractCopiedProjectBundle | Basis voor ‘uwv.copiedProjectBundle’ zonder enig deployment gedrag |

**Tabel 2.** Overzicht van hulp types in de ESA plugin.

# GUI

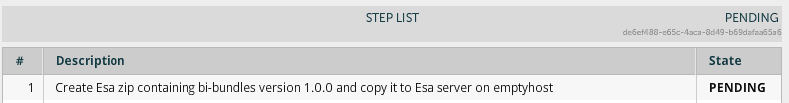
Na het creeren van een ‘uwv.esaclient’ en het plaatsen daarvan in een Environment genaamd ESA kunnen we een project-bundle naar de esaclient deployen in de GUI. De ESA plugin genereert dan een Step die het uiteindelijke werk zal doen (export van een dar, verpakken in een ESA opleverformaat en ftp-en naar de ESA server).

In de GUI ziet een dergelijke deployment er zo uit:



**Figuur 2.** Deployment van BeInformed project-bundle naar een esaclient.

Deze deployment genereert een enkele step die te zien is in Figuur 3.



**Figuur 3.** Step gegenereerd door de ESA plugin bij deployment van een project-bundle naar een esaclient.

De in Figuur 3 gepresenteerde esaclient is geconfigureerd met een ESA server die draait op een Host genaamd ‘emptyhost’. De laatste naam is wat vreemd. Het is de naam die wij gebruiken voor test doeleinden en hier verder niet relevant.

Figuur 4 laat het deployment resultaat zien na executie van de Step.



**Figuur 4.** Resultaat van executie van een deployment naar een esaclient.

In Figuur 4 is te zien dat een ‘zip’ wordt aangemaakt en wordt gekopieerd. De ‘zip’ is het door ESA voorgeschreven formaat. De inhoud voldoet aan de ESA standaard, als ook de naam van het bestand.

# Notificatie

Deze plugin notificeert *niet*. Indien bijvoorbeeld gewenst is dat deployments naar ESA worden gemeld per email, dan is gebruik van de Trigger plugin (zie <http://docs.xebialabs.com/releases/3.9/deployit/triggerPluginManual.html>) de aangewezen weg. Daarom documenteren we dat hier niet verder.

# Type beschrijvingen

Dit hoofdstuk beschrijft alle gebruikte types in detail. Aan de orde komen:

* bi.projectbundle
* uwv.esaclient
* uwv.copiedprojectbundle
* esa.AbstractCopiedProjectBundle

## Deployable: bi.projectbundle

Dit type modelleert een uit de BeInformed Studio geexporteerd bestand. We beschrijven het hier niet nader, omdat het type in de ‘synthetic.xml’ file in de ‘ext’ dirctory van de Deployit server is gedefinieerd. Het is dus reeds elders besproken.

## Container: uwv.esaclient

Dit type modelleert een client van ESA. In implementatie zin gesproken is het een FTP client. Het extend van een “generic.Container”, een out-of-de-box Deployit container type uit de generic plugin. Een ‘uwv.esaclient’ heeft een enkele property: een ‘esaServer’ van type ‘overthere.SshHost’.

We kunnen nu dus de ESA server gewoon als een Host opvoeren in de Deployit infrastructuur, en deze vervolgens in de client hangen.

De gehele definitie is te vinden in de ‘synthetic.xml’ van de ESA plugin en ziet er zo uit:

<type type=*"uwv.esaclient"* inspectable=*"true"* extends=*"generic.Container"*>

<property name=*"esaServer"* kind=*"ci"* referenced-type=*"overthere.SshHost"*/>

</type>

## Deployed: uwv.copiedprojectbundle

Dit type modelleert een project-bundle die naar ESA is gekopieerd. Dit type heeft geen enkele property en dient eigenlijk alleen maar om het aanmaken van een Step middels een zogenaamde Contributor, waarover later meer.

Ook deze definitie is te vinden in de ‘synthetic.xml’ van de ESA plugin en ziet er zo uit:

<type type=*"uwv.copiedprojectbundle"* extends=*"esa.AbstractCopiedProjectBundle"* deployable-type=*"bi.projectbundle"* container-type=*"uwv.esaclient"*/>

Merk op dat dit type een extensie is van het type ‘esa.AbstractCopiedProjectBundle’. Dat laatste is een hulp object dat we nu zullen beschrijven.

## Deployed: esa.AbstractCopiedProjectBundle (Hulp object)

Het type ‘esa.AbstractCopiedProjectBundle’ is een hulp object dat eigenlijk alleen maar dat doel heeft een Step te kunnen genereren in een zogenaamde Contributor. Daarover later meer. Het is een type zonder enig gedrag van zichzelf. Dit is niet in XML te definiëren en we moeten het daarom definiëren in een Java class.

De gehele java code ziet er als volgt uit:

package ext.deployit.esa;

import com.xebialabs.deployit.plugin.api.udm.Metadata;

import com.xebialabs.deployit.plugin.api.udm.base.BaseDeployed;

@SuppressWarnings({ "serial", "rawtypes" })

@Metadata(virtual=true)

public class AbstractCopiedProjectBundle extends BaseDeployed {

}

Bezie weer hoe we het extensie mechanisme van Deployit gebruiken, deze keer op niveau van de java plugin API: we extenden een ‘BaseDeployed’. Eigenlijk fungeert een instantie van deze klasse slechts als een ‘marker’ dat er iets op een esaclient is gedeployed.

In het navolgende hoofdstuk wordt duidelijk hoe deze marker wordt gebruikt in een Contributor die een Step genereert.

# Implementatie

Zoals hierboven al gemeld gebruiken we een Contributor en een Step als implementatie. De Contributor herkent dat er een project-bundle naar een esaclient wordt gedeployed. Als dat zo is wordt er een Step aangemaakt die bij executie het echte werk doet.

We beschrijven daarom eerst de Contributor en dan de Step.

## Contributor

Contributors zijn java methodes geannoteerd met ‘Contributor’. Bij het aanmaken van een stappenplan zoekt de Deployit server naar alle Contributors en roept deze aan met zogenaamde Deltas. Een enkele Delta is een deployment van een Deployable naar een Container.

Onze Contributor krijgt dus alle Deltas, en voegt voor alle Deltas die een deployment van een project-bundle naar een esaclient zijn een Stap toe.

Het cruciale deel van de code ziet er zo uit:

@Contributor

**public** **void** createAndCopyZipToEsa(Deltas deltas, DeploymentPlanningContext ctx) {

List<Deployed<?, ?>> projectBundles = getProjectBundlesDeployedToEsa(deltas);

**for** (Deployed<?, ?> projectBundle : projectBundles) {

ctx.addStep(**new** MakeZipAndCopyToEsaStep(projectBundle));

}

}

Bezie dat we de Deltas bekijken in ‘getProjectBundlesDeployedToEsa’ en een lijstje deployed projectBundles verkijgen. Voor elk van deze voegen we een stap toe aan het stappenplan (via de zogenaamde planningContext).

Volledige details zijn te vinden in de plugin zelf, de klasse heet ‘DistributeToEsaContributor’.

## Step

Iedere deployment van een project-bundle naar een esaclient maakt zoals we hebben gezien een Step aan. Deze is ook in java geimplementeerd en wel in de klasse ‘MakeZipAndCopyToEsaStep’.

Het cruciale deel van de code ziet er als volgt uit:

localConnection = (LocalConnection) LocalConnection.*getLocalConnection*();

connectionToEsa = createAndCheckConectionToEsaServer(ctx);

RepositoryService repoService = RepositoryServiceHolder.*getRepositoryService*();

ExporterService exportService = **new** ExporterService(repoService);

…

WorkDir workDir = **new** WorkDir((LocalFile) workingDirectory);

LocalFile exportedDar = exportService.exportDar(packageId, workDir);

String zipName = createZipFileForEsa(ctx, packageId, repoService, workingDirectory, exportedDar);

ctx.logOutput("created zip file: " + zipName);

OverthereFile src = localConnection.getFile(zipName);

OverthereFile dest = connectionToEsa.getFile(zipName);

src.copyTo(dest);

We maken dus connectie naar de lokale machine (waar de Deployit server draait) en naar de ESA server (door de property ‘esaServer’ van de esaclient te bezien). Vervolgens gebruiken we de ExporterService om de oorspronkelijke dar te hergenereren. Daarna creëren we een ‘zip’ volgens ESA afspraken. Dit bestand staat op de lokale machine. Tot slot gebruiken we de Overthere library van Deployit om de file daadwerkelijk te kopieren van de lokale machine naar de ESA server.

**NB**

Deze code is potentieel onderhoudsgevoelig omdat zowel de ‘ExporterService’ als de daarvoor benodigde ‘RepositoryService’ geen deel uitmaken van de officiële Deployit API. We hebben toch gebruik gemaakt van deze klasses omdat we een ‘dar’ moeten exporteren vanuit een Step. Het enige alternatief om in de java code van de Step een CLI commando af te trappen om deze export te implementeren. Dat is echter nog onderhoudsgevoeliger omdat het vereist dat de CLI is geïnstalleerd en altijd op dezelfde locatie staat en altijd met dezelfde credentials te benaderen is.

Om dezelfde reden is voor de opzet met een Contributor en een (java)Step gekozen en niet voor bijvoorbeeld het extenden van een generic.ExecutedScript. Ook in dat geval zouden we vanuit de context van een Shell script een export moeten doen. En dat kan alleen via de CLI met de boven reeds genoemde bezwaren.

We hebben daarom 23 aug 2013 een verzoek ingeschoten om toegang tot export functionaliteit te verkrijgen via de zogenaamde ‘ReadOnlyRepository’ die *wel* deel uitmaakt van de officiële API. Ticket is ‘#1494 Please add access to Export functionality to API’.