ORACLE

DevOps with Element Manager

Automate Configuration Management in B2C Service, with Element Manager REST APIs and Jenkins

April, 2020 | Version 1.01

PURPOSE STATEMENT

This document provides an overview of features and enhancements included in Element Manager. It is intended solely to help you assess the business benefits of using Element manager in your implementation project.

DISCLAIMER

TABLE OF CONTENTS

INTRODUCTION

Oracle B2C Service customers in their quest to make changes in their IT systems available to their customers quickly, are leveraging the DevOps delivery approach to build pipelines to automate change management, configuration management and deploy management.

The process of building the pipeline for efficient configuration management in B2C Service is accomplished using Element Manager. EM enables developers and administrators to transport Oracle B2C Service artifacts developed in one site (e.g. development or test site) to another (e.g. production site) in an automated way. Customers can also customize the pipeline to have validation of imported artifacts or run some automated test scripts to have a better configuration management strategy using Element Manager in Oracle B2C Service. Thus, shortening the time to test and deploy changes into the production sites.

Configuration Management using Element Manager REST APIs and Jenkins

Element manager provides REST APIs to automate configuration management in programmatic way. Element manager REST APIs can be invoked in a sequence, to export items from a source site and import into a destination site. This process can be also automated using a custom scheduled script or by a simple Jenkins job. Customers can use any continuous integration servers of their choice and customize or extend the pipeline as per their requirement (e.g approvals, QA automation etc).

A sample Jenkins based pipeline to explain the usage of Element Manager REST APIs, is designed with the following stages:

1. Export Stage
2. Import Stage
3. Deploy Stage
4. Verify Deployment Stage
5. Rollback Stage

Element Manager REST APIs overview

APIs	Description
Authentication • POST: /elementmanager/authentication/authToken	• Fetch authentication token for export/import etc
Export • GET: /elementmanager/export/EMPackages/{id} • POST: /elementmanager/export/EMPackages • GET: /elementmanager/export/EMPackages • GET: /elementmanager/export/EMPackages/download/{id}	• Get the status of an export • Initiate an EM Package export • Get a collection of statuses of all exports • Get an exported package
Import • GET /elementmanager/import/EMPackages/{id} • DELETE /elementmanager/import/EMPackages/{id} • GET /elementmanager/import/EMPackages • PATCH /elementmanager/import/EMPackages • POST /elementmanager/import/EMPackages	• Get the status of an import • Delete an EM Package import • Get a collection of statuses of all imports • Import or rollback an EM Package • Initiate an EM Package import
Search • POST /elementmanager/search/EMElements	• Search for available EM artifacts to export

The REST APIs swagger documentation can be accessed via URL like this
https://<siteurl>/AgentWeb/elementManager/swaggerUI

The following section explains how Element Manager REST APIs can be used to build a Jenkins based CI/CD pipeline to transport Oracle B2C Service configuration elements from one site to other. Note, the sample code described below is to be treated as an example and care should be taken to write your own code on your sites/interfaces

A more detailed documentation on “How to use Element Manager REST API” is available in cx.rightnow.com (*answer ID: 10656*)

LEVERAGING JENKINS TO BUILD THE AUTOMATION PIPELINE

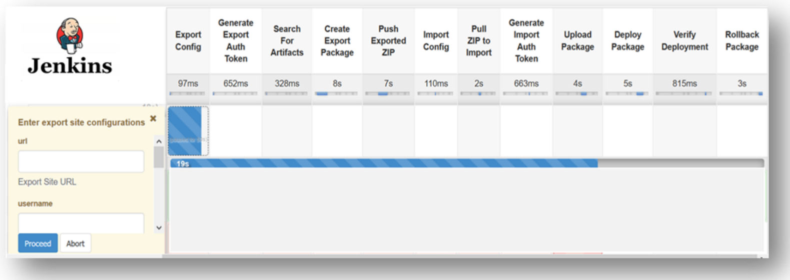
Jenkins is a self-contained, open source automation server which can be used to automate tasks related to building, testing, and delivering or deploying software. This documentation leverages Jenkins as an example to showcase how a CI pipeline could be built. Customers can leverage any other similar tools or services to develop their own pipelines.

Continuous Integration Pipeline Stages

Element Manager involves two key activities

1. Export Phase (from source site e.g. dev/test etc)
2. Import Phase (to destination sites e.g. test/production etc)

The sample pipeline shown in this document, includes following stages (note: customers can create their own pipelines too) to showcase both export and import process in Element Manager.



Export Config	Generate Export Auth Token	Search For Artifacts	Create Export Package	Push Exported ZIP	Import Config	Pull ZIP to Import	Generate Import Auth Token	Upload Package	Deploy Package	Verify Deployment	Rollback Package
97ms	652ms	328ms	8s	7s	110ms	2s	663ms	4s	5s	815ms	3s

Enter export site configurations

url

Export Site URL

username

1] Export Phase

The export process involves, identifying the source site, selecting the elements required for export and exporting out that element along with any depended elements.

a) Configure export/source site
b) Generate Auth Token and authenticate the user
c) Search/select Artifacts to be copied/exported
d) Create Export Package along with dependent elements
e) Export configuration artifacts into a zip format.

1a) The Export Config stage can be designed to accept input parameters or configurations from the pipeline user that includes input credentials like siteUrl, JWT token and search criteria based on which the export action will be performed (e.g “%emdemo%” value in the below code will be used to query artifacts having this string value)

Configure Source Site

```
stage('Export Config') {
 exportData = input(
 id: 'exportSite', message: 'Enter export site configurations ', ok: 'Proceed', parameters:
[
 string(defaultValue: exportSiteURL, description: 'Export Site URL', name: 'url'),
            string(defaultValue: '', description: 'JWT', name: 'jwt'),
            string(defaultValue: '%emdemo%', description: 'Query string to filter the
items', name: 'filter'),
            string(defaultValue: "Demo Packag ${BUILD_NUMBER}", description: 'Export
Package Name', name: 'packageName')
    ]
 )
 serverURL = exportData.url.substring(0, exportData.url.indexOf("cgi-bin")) + "AgentWeb"
}
```

1b) The Generate Auth Token stage can be designed to generate required public API authentication token, using the element-manager authentication endpoint.

Generate Auth Token

```
stage('Generate Export Auth Token') {
def response = httpRequest acceptType: 'APPLICATION_JSON', contentType:
'APPLICATION_JSON', httpMode: 'POST', url:
"${serverURL}/api/elementmanager/authentication/authToken", customHeaders: [[name:
'Authorization', value: "Bearer ${jwt}"], [name: 'interfaceUrl', value: interfaceUrl]]
def result = parseJSON(response.content)
//result .token will be the required token
}
```

1c) The artifacts search can be designed to invoke the element-manager Search API, based on the search string from the Export Config stage (e.g "%emdemo%" in the above code)

Search for Artifacts

```
stage('Search For Artifacts') {
 req = """
 {
 "limit":100,
 "searchTypes":${types},
 "searchString":"${query}",
 "startingIndex":0
 }
 """
 def response = httpRequest acceptType: 'APPLICATION_JSON', contentType: 'APPLICATION_JSON',
httpMode: 'POST', requestBody: req, url: "${serverURL}/api/elementmanager/search/EMElements",
customHeaders: [[name: 'USERSESSION', value: token]]
}
```

1d) The Create Export package stage can be designed to create a package with the required items. The prepare export, getStatus, and download APIs should be sequentially invoked at this stage to download the required package.

Create Export Package

```
stage('Create Export Package') {
 def req = """
 {
 "name" : "${pkgName}",
 "items" : ${items}
 }
 """
 def response = httpRequest acceptType: 'APPLICATION_JSON', contentType: 'APPLICATION_JSON',
httpMode: 'POST', requestBody: req, url:
"${serverURL}/api/elementmanager/export/EMPackages", customHeaders: [[name: 'USERSESSION',
value: token]]
 pakage = parseJSON(response.content)

//getStatus API can be used to track the status of the prepare export action
def response = httpRequest acceptType: 'APPLICATION_JSON', httpMode: 'GET', url:
"${serverURL}/api/elementmanager/export/EMPackages/${pkg.id}", customHeaders: [[name:
'USERSESSION', value: token]]

//download API can be used to download the package
 def package = httpRequest acceptType: 'APPLICATION_JSON', httpMode: 'GET', url:
"${serverURL}/api/elementmanager/export/EMPackages/download/${pkg.id}", customHeaders:
[[name: 'USERSESSION', value: token]]
}
```

1e) The Push Exported ZIP stage can be designed to save the exported ZIP to a GitHub repository or to some other convenient storage.

Push Exported ZIP

```
stage('Push Exported ZIP') {
    sh "rm -Rf element-manager"
    sh "git clone ${repo}"
    dir("element-manager") {
        byte[] zipData = Base64.decodeBase64(zip)
        fp = new FilePath(createFilePath(pwd()), "${exportData.packageName}.zip")
        fp.write().write(zipData)
        sh "git add .; git commit -m \"${exportData.packageName} Export. Build :
${BUILD_NUMBER} - ${BUILD_TIMESTAMP}\"; git push -u origin master"
    }
}
```

2] Import Phase

The import process is executed in a target site, where the elements are to be copied/migrated to. The process involves, importing the package, validate the package, import the elements and deploy them in the target site.

a) Configure Import/Destination Site
b) Retrieve ZIP file
c) Authenticate user credentials
d) Upload Package
e) Deploy Package
f) Rollback Package (optional)

2a) The Import Config stage could be designed to accept the input credentials for the destination site.

Configure Destination Site

```
stage('Import Config') {
 importData = input(
  id: 'importSite', message: 'Enter import site configurations ', ok: 'Proceed', parameters: [
  string(defaultValue: '', description: 'Import Site URL', name: 'url'),
             string(defaultValue: '', description: 'JWT', name: 'jwt'),
    ]
  )
  serverURL = importData.url.substring(0, importData.url.indexOf("cgi-bin")) + "AgentWeb"
}
```

2b) Retrieve ZIP to import stage could be designed to retrieve the exported package from the backed-up storage possibly a GitHub repository.

Retrieve ZIP to Import

```
stage('Pull ZIP to Import') {
 sh "git clone ${repo}"
 dir("element-manager") {
```

```
 zip = readFile(file: "${exportData.packageName}.zip", encoding: "Base64")
 }
}
```

2c) The Generate Auth Token stage could be designed to generate a public API authentication token for the destination site.

Generate Auth Token

```
stage('Generate Import Auth Token') {
 def response = httpRequest acceptType: 'APPLICATION_JSON', contentType: 'APPLICATION_JSON',
httpMode: 'POST', url: "${serverURL}/api/elementmanager/authentication/authToken",
customHeaders: [[name: 'Authorization', value: "Bearer ${jwt}"], [name: 'interfaceUrl', value:
interfaceUrl]]
 def result = parseJSON(response.content)
 //result .token will be the required token
}
```

2d) The Upload Package stage could be designed to initiate the import action using the public API.

Upload Package

```
stage('Upload Package') {
  req = """
 {"packageContent":"${zip}", "name":"${name}"}
 """
 def response = httpRequest acceptType: 'APPLICATION_JSON', contentType: 'APPLICATION_JSON',
httpMode: 'POST', requestBody: req, url: "${serverURL}/api/elementmanager/import/EMPackages",
customHeaders: [[name: 'USERSESSION', value: token]]
 //getStatus API can be used to confirm the package upload is complete
}
```

2e) The Deploy Package stage could be designed to perform the actual deploy action. We could also introduce a confirmation dialog to proceed in this stage.

Deploy Package

```
stage("Deploy Package") {
 //confirmation dialog can be displayed before deploy action
  payload = """{
 "action": "import",
 "id": "${id}",
 "permissions": []
 }
 """
 def response = httpRequest acceptType: 'APPLICATION_JSON', contentType: 'APPLICATION_JSON',
httpMode: 'PATCH', requestBody: payload, url:
"${serverURL}/api/elementmanager/import/EMPackages", customHeaders: [[name: 'USERSESSION',
value: token]]
 //wait for the deploy action to complete. GetStatus API can be used to get live status of
the import action
}
```

2f) The Rollback Package stage could be designed to roll back an already deployed package.

Rollback Package

```
stage("Rollback Package") {
  //a confirmation dialog can be shown before rollback action
  payload = """{
   "action": "rollback",
   "id": "${id}",
   "permissions": []
```

```
 }
 """
 def response = httpRequest acceptType: 'APPLICATION_JSON', contentType: 'APPLICATION_JSON',
httpMode: 'PATCH', requestBody: payload, url:
"${serverURL}/api/elementmanager/import/EMPackages", customHeaders: [[name: 'USERSESSION',
value: token]]
}
```

Exception management for REST APIs

Following are the common exception codes for Element-manager public API:

- `OSC-EM-EXC-0001: Service exception`
- `OSC-EM-EXC-0002: Unauthorized exception`
- `OSC-EM-EXC-0003: Invalid input exception`
- `OSC-EM-EXC-0004: Missing mandatory field exception`
- `OSC-EM-EXC-0005: Profile permissions not required exception`
- `OSC-EM-EXC-0006: Another import is in progress exception`
- `OSC-EM-EXC-0007: Manifest corrupted exception`
- `OSC-EM-EXC-0008: Invalid action requested exception`
- `OSC-EM-EXC-0009: Cannot proceed exception`
- `OSC-EM-EXC-0010: Export threshold exceeded exception`
- `OSC-EM-EXC-0011: API not allowed exception`
- `OSC-EM-EXC-0012: Unable to connect exception`
- `OSC-EM-EXC-0013: Invalid profile-id exception`
- `OSC-EM-EXC-0014: Invalid permission bit exception`
- `OSC-EM-EXC-0015: No privilege exception`
- `OSC-EM-EXC-0016: Package id not found exception`
- `OSC-EM-EXC-0017: Invalid element-id exception`
- `OSC-EM-EXC-0018: Invalid interface-id exception`
- `OSC-EM-EXC-0019: Profile permission not applicable exception`
- `OSC-EM-EXC-0020: Permissions not applicable exception`
- `OSC-EM-EXC-0021: Permission configuration not applicable exception`

CONCLUSION

Customers can now automate their deployment of supported artifacts from Oracle B2C Service across their Development-Testing-Production sites in an agile manner, thus cutting down on the costs and time taken to manage their configuration artifacts.

CONNECT WITH US

Call +1.800.ORACLE1 or visit oracle.com.
Outside North America, find your local office at oracle.com/contact.

blogs.oracle.com   facebook.com/oracle   twitter.com/oracle

DevOps with Element Manager
April, 2020