WHITE PAPER

Automating Compose Deployments

Best Practices for Compose for Data Warehouses

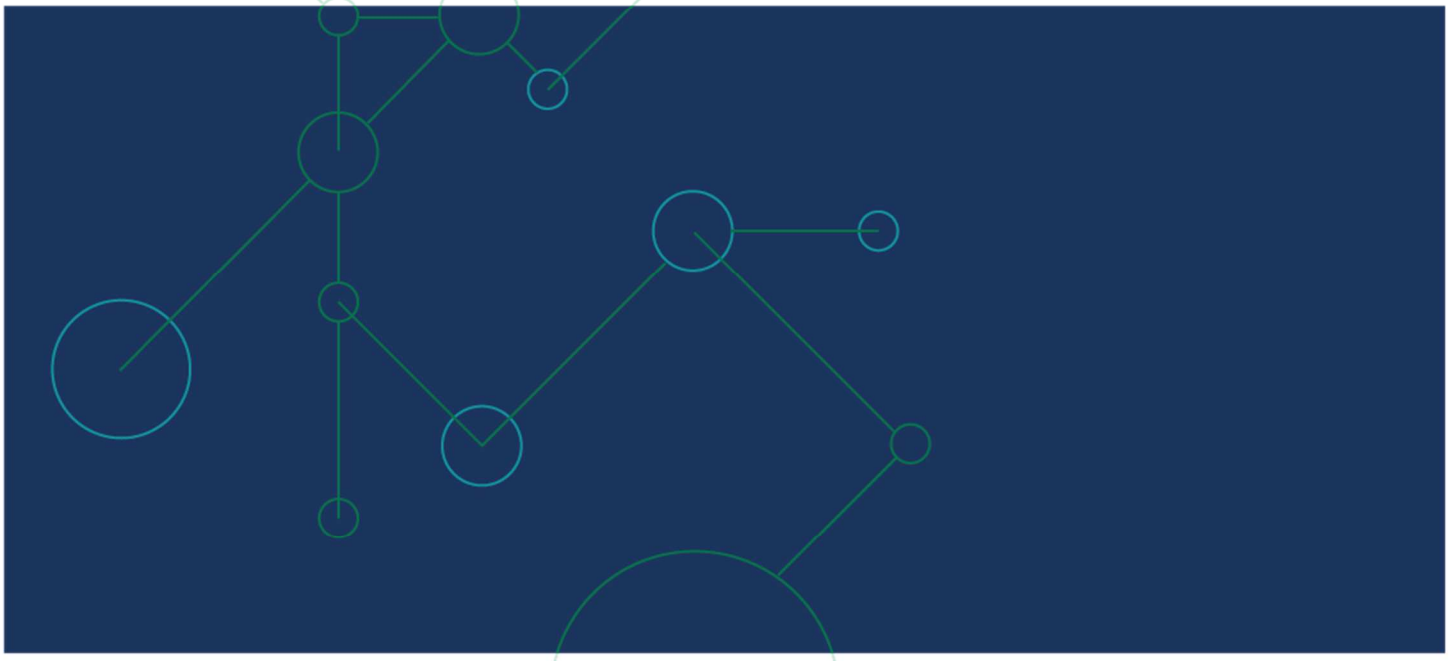


TABLE OF CONTENTS

SUMMARY

- Compose deployment allows the migration of a Compose project to other environments in support of Development, Test, Acceptance and Production (DTAP).
- Compose provides two methods for deploying projects. One is the UI based method for user interactive migration. The other is the Command Line Interface (CLI) method for scripted and automated migration
- This paper will explore using the CLI interface and related artifacts to build an automated pipeline for Compose deployments in a Continuous Integration/Continuous Delivery (CI/CD) fashion

INTRODUCTION

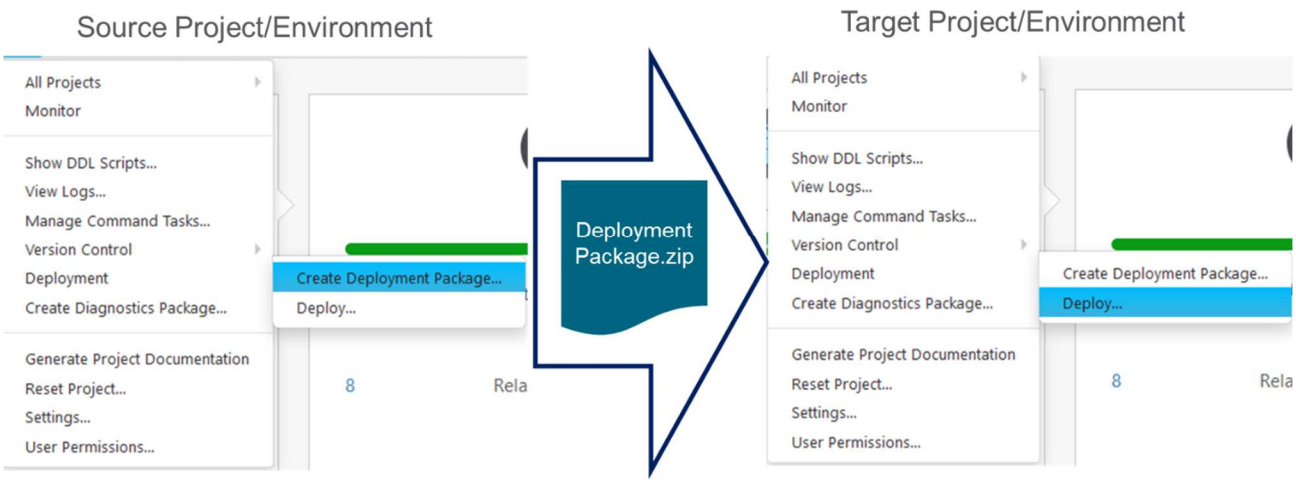
Agility is becoming a foundational aspect of data management implementations. This requires the implementation process to incorporate DataOps concepts to seamlessly move data products from design to operations. Key to DataOps is the ability to establish a CI/CD pipeline to promote finished data products.

Deployment automation is at the heart of any CI/CD pipeline. Qlik Compose supports deployment automation via its Command Line Interface (CLI) and project metadata JSON file.

In this paper we will demonstrate the use of the Qlik Compose CLI commands, project metadata JSON file, Power Shell scripting, and Jenkins CI/CD tool to automate the deployment of projects.

Standard Compose Deployment

The standard Compose deployment uses the UI to export (Create Deployment Package) the project from the source environment and import this project (Deploy) into the target environment. This process is shown below.



In addition, after deploying the project to the target environment the user must perform the following manual steps

- Update the Database connections (if initial deployment into target environment)
- Validate the model
- Create (if initial deployment) or adjust (on subsequent deployments) the data warehouse tables
- Generate task instructions, one by one, for all data warehouse tasks
- Create or adjust Data mart(s)
- Generate the tasks for all data marts

The process outlined requires many manual steps that if executed in this fashion over time present increasing opportunities for error. Deployment automation takes these steps and places them in an automated flow significantly reducing the effort and input needed into the process

Deployment Automation

Automating the Compose deployment requires usage of three components. These are the Compose Command Line Interface (CLI), scripting for processing externalized environment specific variables (PowerShell), and a build automation tool (Jenkins) to manage the deployment. We will first outline the functions performed by each of the components and then provide a sample flow using them.

Compose CLI

The Compose CLI provides three key commands we need to build as building blocks to our deployment automation:

1. Connect - establishes a connection to the Qlik Compose Server. Must be run before running any other command.
2. Export Project – exports the project to a JSON file.
3. Import Project – imports the JSON file to am existing Compose project. Creates the project in the process if it doesn't exist

Command Notes

Export and import commands should be run locally in the Compose server host. Shared drives between servers are recommend for transferring of project export.

Export project uses the is_without_credentials option to not populate password fields. The build process needs to populate these with the ones corresponding to the target environment

Import project uses the override_configuration to have Compose accept the externalized variables for the environment

For additional information refer to the Compose online user guide at help.qlik.com

PowerShell

Between the export project and import project step we need to edit the JSON file of the project to replace environment variables the differ between environments. Among these we have:

1. Host – server name and port for the target environment
2. Database names
3. Credentials – Username & Password

4. Schemas - Data Warehouse, Data Mart, and landing schemas

We will use a Power Shell script to update the values for these parameters with the ones corresponding to the target deployment environment. PowerShell provides two key built-in functions for this process: JSON processing (`ConvertFrom-Json`, `ConvertTo-Json`) and CSV file processing for environment variables (`Import-Csv`)

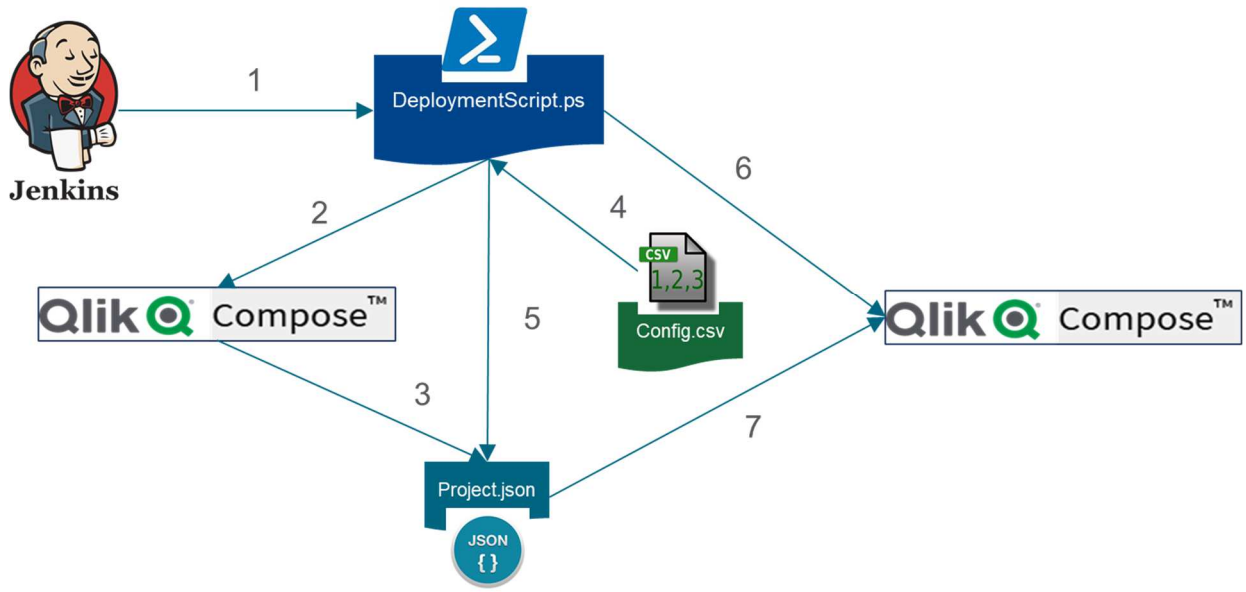
Build automation tool

A build automation tool is needed to serve as the orchestrator for the deployment process. It can be configured to build on demand, on schedule, or by other specified triggers such as a commit to a git repository. In this White Paper, we'll use Jenkins for build automation.

CI/CD Pipeline

End-to-end deployment automation flow

The following flow diagram illustrates the steps and sequence of execution for the automated deployment flow



The steps are as follows:

1. Jenkins triggers the deployment script
2. Deployment script calls Compose CLI in source environment to export project
3. Compose CLI process generates project metadata JSON file
4. Deployment script reads configuration CSV file for target environment
5. Deployment script applies changes for project metadata JSON file
6. Deployment script call Compose CLI to read updated project JSON file in target environment
7. Compose CLI read in project JSON file, applies it, and validates, adjusts, and generates tasks

We will now detail the CI/CD pipeline by putting the automation concepts to work. First, we need to define a parameter file for the target environment. The file is defined as follows:

Config file format

name	server_name	database_name	user_name	password	included_schemas	dma_schema	port
Data Warehouse	Db.host.com	DW_DB	Compose	Password	DW	DM	1234
Sales Src	Db.host.com				*		0
Sales Src_Landing	Db.host.com	SRC_DB	Source	Password	Landing	Landing	1234

Notice the name field must match the name of the databases defined in the Compose project. The key databases for higher environment are the Data Warehouse and those with _Landing in the name. The others are blanked out because they are only used for metadata at design phase.

Script

We now illustrate the key sections of the PowerShell script. First the connect and export commands to connect to the Compose server and fetch the project as a JSON file

```
$connectProc = Start-Process -NoNewWindow -FilePath $composeCLIPath -ArgumentList "connect"
-PassThru -Wait
if ($connectProc.ExitCode -ne 0) {
    exit $connectProc.ExitCode
}
```

```
$exportProc = Start-Process -NoNewWindow  -FilePath $composeCLIPath -ArgumentList
"export_project_repository  --project ""$projectToExport"" --outfile=""$file"" --
is_without_credentials" -PassThru -Wait
if ($exportProc.ExitCode -ne 0) {
    exit $exportProc.ExitCode
}
```

Once executed, these commands fetch the JSON export for the source environment project. We then need to use the defined parameters file to replace the corresponding parameters in the JSON file. The following snippet reads the CSV file. Note that this command by default uses the first row of the CSV as header and allows addressing the values in the columns by the names in the header.

```
[string]$InputTableListCSVFile=$csvfile
$InputTableListCSV = Import-Csv -Path $InputTableListCSVFile
```

Once we have the CSV loaded, we can use the following command to read the exported project JSON.

```
$jTree = Get-Content -Raw -Path $file|ConvertFrom-Json
```

This command reads the entire JSON document with the Compose project metadata and instantiates a corresponding document tree object in memory. Since our focus for deployment are those items that change across environments, namely the databases section, this is what we'll read and modify. However, note the same method could be used to change other aspects of the project programmatically.

The following snippet reads and takes stock of the databases for the project.

```
$databases = $jTree.objects|where type -EQ "Database"
$databaseHash = @{ }
foreach($database in $databases) {
    $databaseHash.Add($database.name, $database)
    $database.inner_item.name
}
```

The next step is to iterate through the list of databases present in the configuration file and update the corresponding database in the project metadata JSON. Since we are using the CSV configuration to drive the project update, only those databases included in the file will be modified. The following snippet updates the configuration:

```
foreach ($line in $InputTableListCSV) {
    $databaseHash.Item($line.name).inner_item.server_name = $line.server_name
    $databaseHash.Item($line.name).inner_item.database_name = $line.database_name
    $databaseHash.Item($line.name).inner_item.user_name = $line.user_name
    $databaseHash.Item($line.name).inner_item.password = $line.password
    $databaseHash.Item($line.name).inner_item.included_schemas = $line.included_schemas
    $databaseHash.Item($line.name).inner_item.dma_schema = $line.dma_schema
    $databaseHash.Item($line.name).inner_item.port = $line.port
}
```

Once the changes are made to the project metadata, the last step is to create a new JSON with the updates and to invoke Compose import project command to deploy it. Note that this example is using the same Compose server for both environments (source of migration and target of migration). It would need to be modified to accommodate for different environments for source and target.

First write the new file:

```
$new_json_string = ConvertTo-Json $jTree -depth 99
$newFile = $file.Replace('.json','_UPDATED.json')
$new_json_string|Out-File -FilePath $newFile
```

Then call Compose to import the project.

```
$importProc = Start-Process -NoNewWindow  -FilePath $composeCLIPath -ArgumentList
"import_project_repository  --project ""$projectToImport"" --infile=""$newfile"" --
override_configuration --autogen" -PassThru -Wait
```

The import command due to the –autogen parameter is going to execute all the steps outlined in the standard deployment section of this document automatically. The one caveat to keep in mind is that any warehouse changes that require manual execution of a SQL script would make the import command fail at that step and require the use to manually complete the script and remaining steps. IT is advisable to execute any such changes in advance of calling the deployment command to have this process complete without manual intervention.

Deployment automation with Jenkins

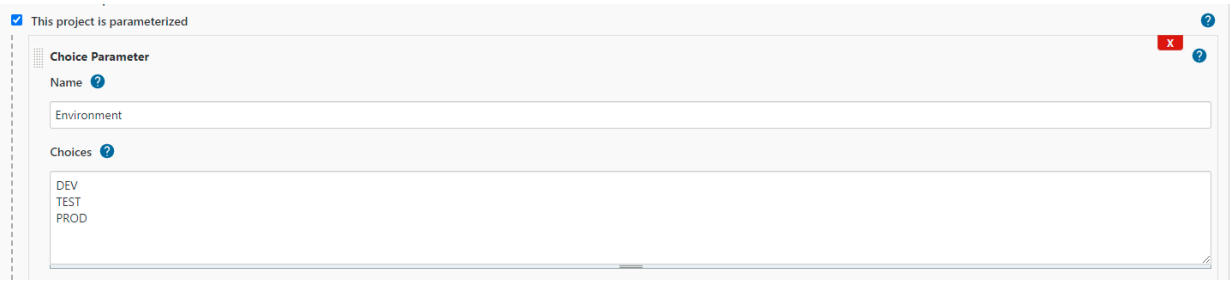
The orchestration of the deployment script is best managed using an automation server. For this exercise we will utilize the Jenkins open-source tool, however any similar purpose tool would work. The process will use a simple same server for build automation and Compose. In most cases the user will need to use a distributed build where the automation server and Compose, potentially both source and target of deployment are all in separate hosts. The particulars of that distributed build configuration are beyond the scope of this paper.

The basic steps to setup the Compose deployment on Jenkins are as follows:

1. Create a new item of type: Freestyle project
2. Add build step of type: Execute Windows batch command
3. Inside the command text box, paste the command to run the PowerShell script to execute the deployment. E.g.


```
powershell -ExecutionPolicy Unrestricted C:\code\powershell_samples\C4DW_Migrate_Project.ps1
DW_DEV2 DW_TEST C:\temp\testscript C:\DevTools\Qlik\Compose\bin
C:\temp\ComposeConfig_values_%Environment%.csv
```

Note that in this case, we are using a build parameter for the environment to build to, using the parameterized project option



Similarly additional parameters could be added and passed to the script.

4. Add a trigger (optional) to have the build run on a schedule or using a GitHub hook trigger, if Compose is configured for remote commits, to have the build automatically triggered after a push.

Conclusion

Qlik Compose has built-in support for a CI/CD pipeline leveraging deployment automation via it's command line interface (CLI). We have demonstrated how to create this pipeline using the following components:

	ComposeCLI	PowerShell	Jenkins	CSV configuration file
What It is	Programmatic interface to execute Compose commands	Scripting language	Build automation tool	Text file with comma separated values
Role	Project metadata export and import using a JSON file	Provides native facilities to handle JSON, CSV, and Windows commands needed to process the deployment	Orchestrates the automated build	Externalizes environment specific parameters

Organizations can leverage this method to integrate Compose deployments into their DataOps environments in an automated fashion in support of their CI/CD pipeline.