DBcloudbin integration in DevOps processes

WHITEPAPER

DBcloudbin

Delivered by Tecknolab. Cloud, BigData & Analytics
info@tecknolab.com

Date: 08/03/2019
Revision: rev2

Contents

Introduction

In the era of big data, our applications have to process and manage an increasingly larger amount of data. In many cases, this data is stored in our 'plain-old' relational databases, due to its capabilities and transactional requirements of many of our business-critical applications.

DBcloudbin is an innovative solution for moving contents from enterprise databases (as of today, Oracle and SQL Server are supported) to Cloud optimized object storage (both in public Cloud services or on-premises through a broad range of supported platforms). This generates huge cost reductions and simplified administration due to the drastic DB size reduction and, with it, infrastructure and backup costs to mention a few.

The key value proposition is to move the content (fundamentally binary content as documents or pictures, that are large) without any change to the application code. The content remains online and accessible, the application keeps fetching that content through the database and DBcloudbin is responsible for transparently taking this content from the object store and serving it through the database.

For doing this we need to make a few adaptations to the application data model, changes that are completely automated through a wizard-based setup tool. It is extremely simple to deploy the solution and start reducing database size.

In some scenarios our applications are in 'freeze-mode' let's say; the application is no longer enhanced or updated from a code and functional perspective; we may have no SW developers assigned to it and it is just administered from an Operations team. In this case, applying DBcloudbin is especially convenient; we don't need any specific SW dev knowledge, we just need a reasonably high-level understanding on how the data model is designed and where the data of each specific area of the application is supposed to be stored (which tables store what).

However in many other cases the application is actively being developed and enhanced adding new functional requirements in upcoming releases. The Dev team has to work with the Ops team to push new versions into production in a reliable and efficient way. In this scenario, we have to make sure that DBcloudbin specifics is integrated seamlessly into the process, avoiding any problem in production.

This whitepaper discusses the different scenarios where the adapted architecture derived from a DBcloudbin implementation may be affected by our DevOps processes.

Covered scenario

For the sake of this analysis, we will assume the following scenario:

- We have a SW Dev team (potentially external or outsourced) that has developed an existing application. This application is available in production in a defined release version with a set of known functionalities.
- We have a series of environments where the application is installed for different purposes during the DevOps processes. We will focus fundamentally on the DB layer since the application or app server layer is irrelevant for the scenario covered in this whitepaper. We may have more (or less), but we are considering a few very commonly present:
    - **Dev Environment**: It is where the SW developers creates the new code with potentially changes to the database model. This changes will be somehow collected into a 'delta-script' that defines the changes into de database to pass from version A to version A+1. Tends to be an 'unstable' environment and in some cases instantiated individually by each developer or small dev teams.
    - **Integration**: This is a controlled environment where the different development streams that are coded, potentially in parallel, are consolidated together and tested with a unified test plan. We should have a 'target-release' version for the code and database

model that is tested here. All data model changes from previous release should be collected and are clearly reproducible with a defined order of implementation through one or several DDL (data-definition-language typically a subset of SQL) scripts.
    - **Pre-production**: This is an environment mimicking the production environment from an architectural perspective with a smaller size. In many cases, there are processes for automatically or manually feeding a subset of content from the production environment with 'real-data' (potentially obscuring sensitive data). Here the final testing is done before pushing the new release into production.
    - **Production**: This is where our application is actually supporting real customers and transactions with a stable release version that has passed through the different previous environment and validation tests.
- We have a reasonably formalized process for implementing a new version of our application software 'upwards' through the different environments and potentially a data-feeding process 'backwards' for feeding data into the non-production databases.

Whitepaper Structure

During the whitepaper we will discuss on the following main topics that may impact our DevOps processes when we decide to implement DBcloudbin:

- Initial DBcloudbin implementation.
- Application update to a new release version.
- Refreshing data from production environment.
- Upgrading DBcloudbin.

DBcloudbin Overview

As we mentioned at Introduction, DBcloudbin enables the movement of database content into a specialized object store repository (large scalability, low cost) with transparency for the application.

This is the high-level technical architecture:

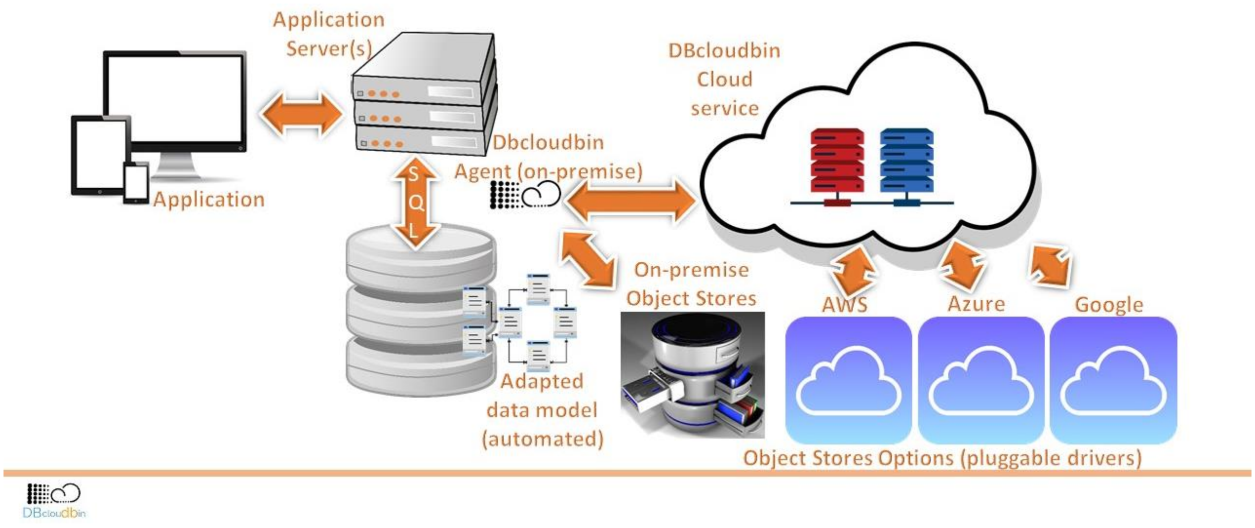


Figure 1 DBcloudbin technical architecture overview

In this scenario we have an enterprise application, potentially with a series of application servers and a enterprise database. The application uses this database to store application data in a transactionally consistent fashion. Part of this data is non-relational data (documents, pictures, ...).

Installing DBcloudbin using the wizard-based installer will allow to create a web-service interface (the DBcloudbin agent) from the database and a large list of potential object stores (both on-premise through one of our technology partners or in the most important Cloud providers, including a

DBcloudbin provided one). The installer will adapt automatically the application data model in order to allow communication from the DB with the agent and generate an attribute where the object-id (the identifier to locate the content in the object store) can be safely stored. In order to move the content to the Cloud (to reduce database size) or back to the database (e.g. for uninstalling DBcloudbin or any other reason) we have a command line interface (CLI) with simple commands where we can define the business rules for the content we would like to move (e.g. documents older than 1 month); the CLI execution can be easily scheduled through any of the existing task schedulers in the market.

Benefits

The list of benefits of implementing DBcloudbin is very compelling:

- **Immediate implementation**. Installation is automated with a simple wizard that can be executed in minutes.
- **Application transparency**. Data model adaptation is done in such a way that the application works with no modifications, no matter if the content is at the database or at the Cloud. If the latter, the content is fetched by DBcloudbin agent transparently and injected into the query so the application has the sensation that the content is locally. This is done with minimal increased latency.
- **Improved performance**. Binary data, as documents, are typically queried in human intervention scenarios (e.g. a person through the application interface pressing a button or link to read that document). These scenarios have a negligible impact on slightly higher latency reads. However, complex queries with sorts, filters and joins that any non-trivial application has to execute, gets very positively impacted by a smaller database. So, when we reduce in 80% or 90% our database size, the gain in performance is remarkable. By the other hand, writes (inserts, updates) with a DBcloudbin implemented database goes always to the database in first instance (the data transfer to Cloud is a batch, decoupled operation) so there is no difference in this scenario.
- **Big-data analytics**. In many cases we would like to do advanced analytics in a large dataset of contents; if those datasets are in our enterprise, business-critical DB, we have a serious limitation due to the potential performance impact. With the contents in a large and scalable Object Store we have the ability to freely analyze them without impacting the business-critical DB and with no need to duplicate the content (with the potential inconsistencies that may derive).
- **Database migration.** A database migration project (e.g. major version migration, HW re-platforming) is very complex, increasing exponentially with the database size; DB unavailability during migration is also very dependent on size. So, if we can move the vast majority of content while the application is online, the maintenance window can be reduced significantly.

DevOps scenarios considered

Considering the situation described in "Covered scenario" we will analyze the impact that DBcloudbin may have when an application in under active maintenance and evolution with a SW development team adding new functionality. The impact in the application SW 'code' itself is inexistent, since the application code is unaffected; however de data model has some minor changes that may affect the process of 'pushing' a new release from development into production.

Let's discuss in detail each scenario.

DBcloudbin implementation

DBcloudbin implementation is fast and simple. In this scenario, the data model in the database where the solution is being installed will be automatically adapted to support the requirements of both storing content in an external object store and keep compatible for the original SQL sentences delivered by the application(s) that uses this database.

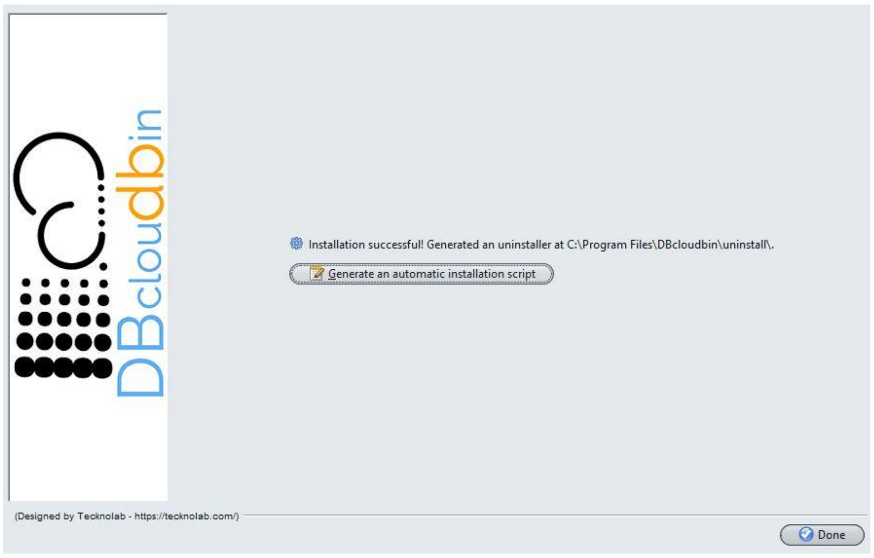
Our recommendation is to start implementing DBcloudbin in an integration or preproduction environment and test thoroughly application compatibility. In most scenarios the application works perfectly with no issues, but we cannot guarantee 100% that a quite uncommon or non-standard access to the database may be affected. So, it is important to test it in a non-production environment first.

We can consider two phases on DBcloudbin implementation:

- DBcloudbin setup itself. That is, installing the software and configuring through the database. It is detailed in this post.
- Data migration. This is an action to be performed after installation (potentially and most probably through a task scheduler.

The relevant phase from a data model compatibility perspective is the first one. Checking application compatibility with DBcloudbin can be done without moving any content outside the database, just executing the setup and testing the application afterwards. So we can concentrate the testing with actual data migration in the preproduction environment just before pushing the solution into production.

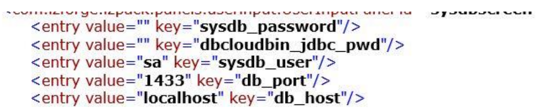
DBcloudbin setup has the very handy functionality of being able to generate an xml descriptor for automated unattended installation. So we can execute a setup with our specific parameterization and, at final stage, ask for generation of an `auto-install.xml` file.


DBcloudbin
Installation successful! Generated an uninstaller at C:\Program Files\DBcloudbin\uninstall\.
Generate an automatic installation script
(Designed by Tecknolab - https://tecknolab.com/)
Done

With this, we only have to use the 'Linux' version of the setup in jar format (in fact, it works also in Windows with no issues) and pass as an argument the xml descriptor to execute an unattended installation. That is:

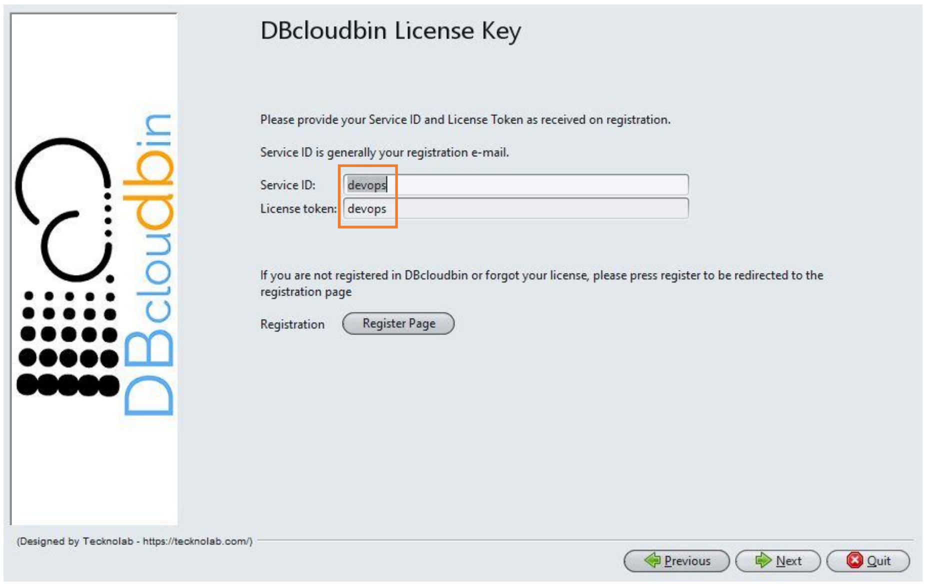
```
java –jar dbcloudbin-setup.jar auto-install.xml
```

*(**Note**: for security, DB passwords are not saved in the xml descriptor and has to be provided by modifying the descriptor or using environment variables and wrapping the descriptor in a shell script).*



```
    <entry value="" key="sysdb_password"/>
    <entry value="" key="dbcloudbin_jdbc_pwd"/>
    <entry value="sa" key="sysdb_user"/>
    <entry value="1433" key="db_port"/>
    <entry value="localhost" key="db_host"/>
```

In addition, DBcloudbin provides a special license id and user "devops" that can be used to setup the tool providing we will not do any content archive (there is no object store in the backend).


DBcloudbin License Key

Please provide your Service ID and License Token as received on registration.

Service ID is generally your registration e-mail.

Service ID: devops
License token: devops

If you are not registered in DBcloudbin or forgot your license, please press register to be redirected to the registration page

Registration: Register Page

(Designed by Tecknolab - https://tecknolab.com/)

So, we can easily create a script that setup automatically DBclodubin with the exact same configuration from a data model perspective that we are going to use in production in any pre-integration or dev environment in order to ensure we have no compatibility issues and add the xml descriptor to our configuration control.

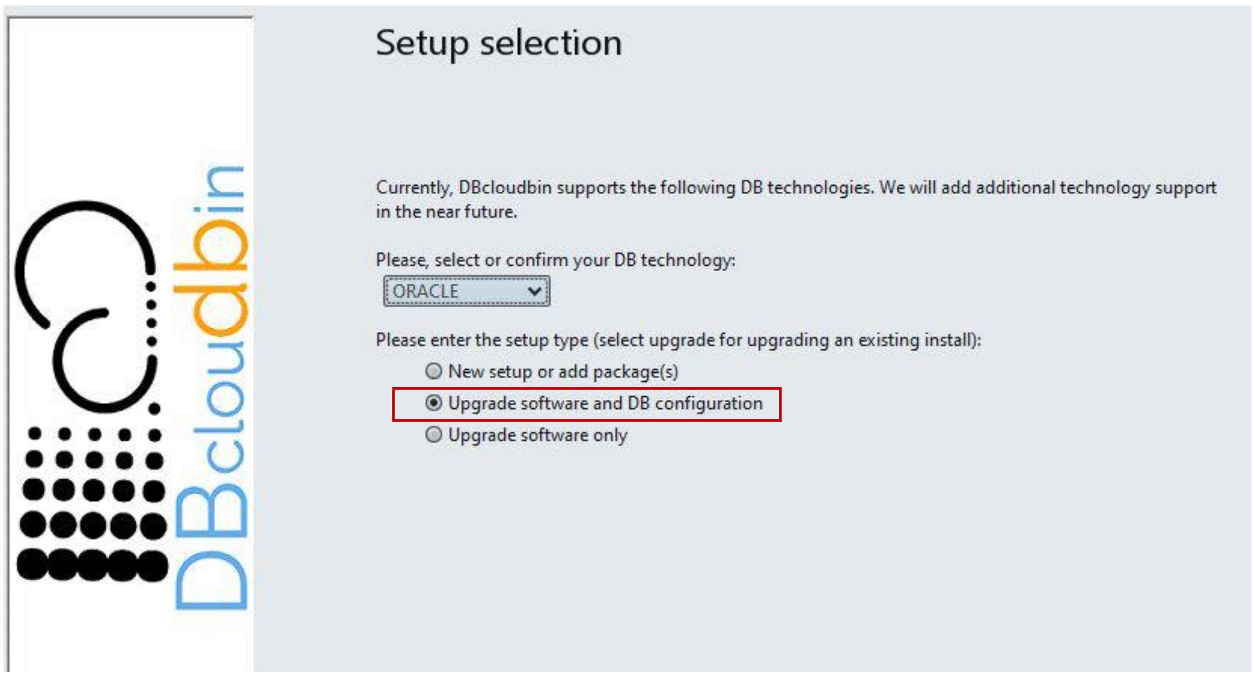
Application update to a new release version.

With the compatibility validated, our only point of attention should be the data-definition-language (DDL) SQL scripts that adds or changes the data model due to a new version of our application. Those tables that have been selected as data migration candidates to the object store (those holding BLOB-type data that we have selected during DBcloudbin setup) are renamed during DBcloudbin implementation and as such we have to modify our DDL scripts. The renamed table will typically be the original name with a suffix "_DBCLDBN"; however, due to identifier restrictions especially in Oracle, long table names have to be hashed in order to comply with naming size restrictions. The target table name (is immutable and will be the same in any environment you setup DBcloudbin for the same data model) can be retrieved with the command "dbcloudbin info" using the installed command line interface (CLI). So, e.g. in Linux, a simple "sed/<original-table>/<renamed-table>/g" throughout the DDL script would make the trick. We can keep the original DDL script file and change slightly the command that applies it to the Database (e.g. an invocation to sqlplus in Oracle), using as <renamed-table> in the previous sed command an expression that provides the renamed-table if DBcloudbin is installed and the original name if it is not; this way, the DDL script execution command is valid both in Dev environments where DBcloudbin is potentially not installed or in preproduction or production environments where it is. A 'compact' example using a bash script would be:

```
   cat <ddl-script-name.sql> | sed "/s/<orginal-table-name>/$(dbcloudbin info –table <original-table-name> 2> /dev/null || echo <original-table-name>)/g" | sqlplus <connection-credentials>
```

Any other update into the data model that is not affecting this subset of tables should have no relevance nor impact into DBcloudbin implementation.

After DDL script update to the datamodel it is recommended to execute DBcloudbin upgrade in order to refresh application datamodel objects created by DBcloudbin (required if the changes affect the table structure of BLOB containing tables managed by DBclodubin). The DBcloudbin upgrade is fast, easy, automatable and does not have any impact if executed several times over the same target version. It is as simple as generate a automation XML descriptor by executing the product setup, as described in DBcloudbin implementation (the only difference is that we are going to select DBcloudbin upgrade option when executing the setup, as shown below).


Setup selection

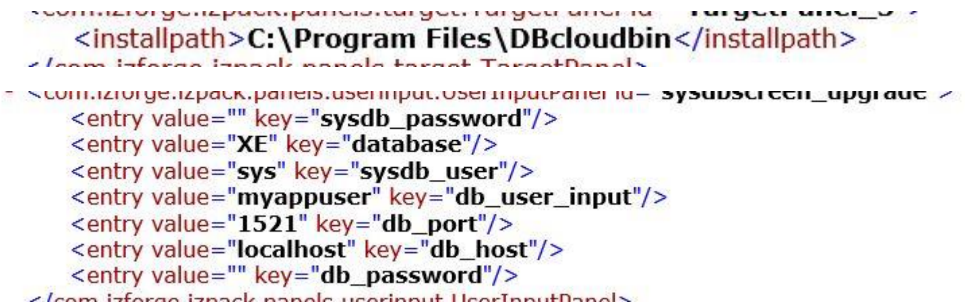
Currently, DBcloudbin supports the following DB technologies. We will add additional technology support in the near future.

Please, select or confirm your DB technology: ORACLE

Please enter the setup type (select upgrade for upgrading an existing install):

- New setup or add package(s)
- [✓ Upgrade software and DB configuration]
- Upgrade software only

We will provide DB credentials for a sysadmin and the application DB user and generate the automation xml descriptor in the last step. Then, we have to simply save this xml descriptor as template and tune it with the install path and DB credentials suitable for our different environments; this can be easily done by injecting environment variables and substitute placeholders in the template file by real values.


```
    <installpath>C:\Program Files\DBcloudbin</installpath>

- <com.izforge.izpack.panels.userinput.UserInputPanel id="sysdbscreen_upgrade">
    <entry value="" key="sysdb_password"/>
    <entry value="XE" key="database"/>
    <entry value="sys" key="sysdb_user"/>
    <entry value="myappuser" key="db_user_input"/>
    <entry value="1521" key="db_port"/>
    <entry value="localhost" key="db_host"/>
    <entry value="" key="db_password"/>
```

After generating the xml descriptor (e.g. auto-upgrade.xml), we can just integrate the automated DBcloudbin upgrade in any of our process by executing `java –jar dbcloudbin-setup.jar auto-upgrade.xml`

Refreshing data from production environment.

One typical scenario in large enterprise applications is to refresh data from production environment into test or integration environments (potentially sub-setting and/or obscuring sensitive data).

When using DBcloudbin, we have a much smaller database size so data refreshing is in fact easier and faster. However, part of the data may potentially not be at the database but at the object store, so we need to review our prepro data refreshing strategy.

In general, our goal would be to avoid any interaction of non-production environments with the production repositories, including the object store. This is done by avoiding any access from the non-production environments to the production object store. Since DBcloudbin saves all configuration environment into a specific database schema in the target database, we can easily use different credentials for the prepro environment and production environment, effectively pointing each environment to different object stores. For refreshing data, we need to ensure that the objects moved in production to the external object store, can be fetched and stored at the pre-production object store; that way, we ensure that the pre-production instance is able to use the same data. There are several strategies to solve this requirement:

- **Replicate the production object store**: we can systematically and asynchronously replicate production object store. This way we have the data in preproduction and when loading the production database into the pre-production database, the pointers to the external data will be valid (will be fetched from the pre-production object store, but the data will be the same and consistent with the loaded database).
- **Replicate a list of objects**: it would be pretty simple to create a script that receives a list of object IDs (listed from the database) and replicate those objects to the pre-production object store. In this scenario, the sequence would be the opposite of the previous one and is more suitable for situations where we move just a small subset of production into pre-production. So we first load the database from production and then execute a simple query into de database for the list of objects that have are migrated into the object store (the field where this data is stored is named as the original table attribute plus the suffix "_DBCLDBN"; so if our field with blob data is called "`content`", we have to list the non-null values in "`content_dbcldbn`"). With that list we just copy those objects from the production object store into the pre-production object store (copy command depends on the specific technology of the object store used). Once executed, the database is prepared for working as if it were the production database.

Upgrading DBcloudbin.

As with any software from time to time we will have to upgrade DBcloudbin with new released versions. In typical DevOps scenario we recommend to integrate it from a process perspective with the equivalent process of upgrading our application software.

Leveraging the capability of unattended installation of DBcloudbin setup tool, we can easily generate an XML descriptor for automating DBcloudbin upgrade and use this descriptor throughout the different application environments from development to production. We recommend to add the descriptor (and wrapping script if we use a simple script for launching the setup and injecting credentials) as an additional artifact in our configuration control system (git or similar).

Resources

- DBcloudbin website: www.dbcloudbin.com
- General overview. https://www.dbcloudbin.com/how-it-works/
- Installation procedures. https://www.dbcloudbin.com/installation/
- DBcloudbin security. https://www.dbcloudbin.com/security/
- Return-Of-Investment tool and savings. https://www.dbcloudbin.com/savings/
- Free evaluation. https://www.dbcloudbin.com/product/promotional-free-service/