Managing Database Backup and Restore

Overview

The Deployment Packager performs back ups of the REF database as part of standard operation. If packaging processes DDL files using the CONVERT workflow or fails for any reason, the database is restored from that backup. This allows future Deployment Packager jobs to start from a known good state when errors are encountered or contributed changes are rejected by the Deployment Packager. This page will inform you on configuration options for the backup and restore features of Deployment Packager.

Using Static Backups

For large or highly complex database environments, requiring a dynamic database backup for each packaging job may significantly reduce performance. For these cases an alternative is to provide a static backup location and manage database backups directly:

- Backups are performed separately and not performed as part of packaging database changes.
- Deployment Packager uses an existing backup file rather than creating a new one for each run.
- New backups are performed manually through one of the following methods:
    - Backup is performed outside of Datical
    - Deployment Packager is run with `createDatabaseBackup=true` and `preview=true` command-line options

Additional Database Configuration Requirements

For Oracle, PostgreSQL, and DB2 databases there is no additional configuration required.

For SQL Server, enable OLE automation procedures on the target database as follows:

```
USE [database to backup]
GO

RECONFIGURE;

EXEC master.dbo.sp_configure 'show advanced options', 1
RECONFIGURE
GO

EXEC master.dbo.sp_configure 'Ole Automation Procedures', 1
RECONFIGURE
GO
```

Configuring Deployment Packager Backups

Set `databaseBackupMode` in `deployPackager.properties` to control backup behavior.

- `always` - (Default). The REF database is backed up for every packaging job.
- `on_demand` - A full backup of the REF database is made when Deployment Packager is run with the following options:
  - `pipeline=<pipelineRef>`
  - `createDatabaseBackup=true`

You can run a backup directly using the Deployment Packager command line.

- `createDatabaseBackup=true` - causes a new backup to be created. May be used in preview mode to create a new backup without creating new changesets. If done during packaging, the packaging must be done on a pipeline (using `pipeline=`), rather than on a DbDef.

Best Practice
The easiest way to get regular database backups to use a consistent naming convention is to run Deployment Packager to do them. The nightly database backup job is a copy of the Deployment Packager job. The copy runs Deployment Packager but does not create new changesets.

Schedule a recurring job that runs Deployment Packager with preview and backup modes set, as shown in the following example:

```
hammer groovy deployPackager.groovy pipeline=MY_PIPELINE_NAME scm=true createDatabaseBackup=true
preview=true
```

Packaging Behavior

Always backup

When `databaseBackupMode` is set to `always`, the database is restored from the backup file in the event of packaging failure. Deploy Packager performs these operations:

1. Back up the reference database prior to modifying the database.
2. Deploy new changes.
3. Restore from this backup if a failure occurs.

On Demand backup

When `databaseBackupMode` is set to `on_demand`, the database is restored from the backup file specified in the deployPackager.properties in the event of packaging failure. Deploy Packager performs these operations:

1. Check that the appropriate database backup file/path, privileges, and tools are present in the event a restore needs to occur.
2. Deploy new changes.
3. Restore from backup specified in the deployPackager.properies if a failure occurs.

For this process, redeployed changes are not tracked in the Deployment Monitoring Console database (DMCDB), because their deployment was already recorded. Only the new deployment creates new records in the DMCDB.

Permissions

See the roles and permissions needed for your database in Managed Databases

Oracle requires specific backup permissions, as noted in Roles and Permissions for Datical DB on Oracle Database.

Packager checks for the existence of the backup file. If checking fails on a permissions error, the responses depends on the setting of backupRestoreMode:

- ALWAYS - warn that permissions may not be set correctly and continue.
- ON_DEMAND - report an error and halt.

Impact on Packaging Time

Performance Comparison

The procedures below shows a comparison of the process in simplified form. Deploy Packager also uses the backup file internally during processing.

Normally packaging time when always backing up depends on these operations:

1. Back up the REF database
2. Package new changes
3. Restore the database from the backup file if an error occurs

When using a managed on_demand backup, packaging time is based on these operations:

1. Package new changes
2. Restore the database from the backup if an error occurs.
3. Redeploy previous changes if an error occurs - *this time grows as additional changes are deployed*.

All packaging runs should experience a lower packaging time in this mode. Successful packaging runs potentially see the most performance benefit as in some cases these may now not require a backup or a restore of the REF database.

Troubleshooting Performance

If you have implemented static/on_demand backups but have not seen a significant improvement in how long your packager jobs take to run, check your configuration. If you have set `databaseBackupMode=on_demand` but are still using `createDatabaseBackup=true` in your main packager jobs that process scripts, that is an unusual configuration. Packager will work with that configuration but it will still create a new backup file for each packager job that processes scripts, and therefore you would NOT be getting the possible performance benefit that you would have in the more typical configuration of creating a nightly backup separately with a separate packager job in preview mode and re-using that backup file when processing scripts (to avoid running backup each time). To optimize the on_demand backup, do NOT use `createDatabaseBackup=true` with your main packaging job that processes scripts (assuming that the backup file was already created and is in place).

You can also see other possible packager performance adjustments here: How To: Improve Packager Performance

Risks

Although backing up the reference database during every deployment takes time, it also assures reliability. If packaging fails, the reference database is automatically restored to exactly the state it was in when packaging started. Not taking a backup every time introduces risk to the system.

In addition, any changes made to the reference database manually (outside of Datical) are not restored when using static backup file. If you make manual changes to the reference database, you should then backup the reference database and replace the backup file that Datical uses during packaging.

Troubleshooting Backup & Restore Issues

Recovering from a Backup or Restore Failure

Occasionally an error can occur during the backup or restore phase of the Deployment Packager. Typically this happens early in the implementation process as configuration is being refined, when configuration options for backup or restore change or due to a connectivity or availability issue. Because the accurate production of new changes and the continuous automation capability of packager relies on accurate database state, the Deployment Packager should not be run until the source of the error has been addressed and the RefDB has been restored to a known good state.

As of Datical DB 7.6, execution of subsequent Deployment Packager jobs will be blocked when a backup or restore error was detected during a previous execution of the Deployment Packager.

Backup/Restore Lock File

When a failure is detected, Datical DB will create a .lock file in the root directory of your Datical Project and commit that to the Source Code Control repository for your project.

- This file follows a naming convention: `<dbdef name>.lock`
    - For example: If your DbDef name is `RefDB` the file will be named `RefDb.lock`
- The file contains the error output from the backup or restore that caused it to be created. The information will also exist in the daticaldb.log file for that Deployment Packager execution.

If you attempt to run the Deployment Packager again, you will get an error similar to the example below.

```
========================================================================
=======
DATICAL DB: Packaging SQL Script(s)
========================================================================
=======
BEGIN: 2020-08-12-13:56:22
ERROR: Run of packager blocked due to:
ERROR: Lock file exists - 'C:\Users\ddb\git\OT_SAMPLE\OT_SAMPLE\OT_REF.
lock'
ERROR: A failure has occurred during a backup or restore operation and
future runs of deployPackager.groovy will be blocked as a result.
ERROR: Please address the error and run <hammer groovy deployPackager.
groovy MAR_2020 scm=true, preview=true, createDatabaseBackup=true
clearBackupRestoreLock> to unblock deployPackager.groovy
Command 'groovy deployPackager.groovy MAR_2020 scm=true preview=true
createDatabaseBackup=true' exiting with error status 1
```

NOTE: The presence of a lock file should only block future executions of the Deployment Packager for the pipeline where the error occurred as long as each pipeline has a unique RefDB configured.

Unlocking a Datical DB Project after backup/restore issue has been resolved

Do not remove the lock until the original backup or restore issue has been investigated and resolved. Only when the issue that caused the error has been addressed and a new backup for the RefDB has been made, can the lock file be safely removed.

After the backup/restore issue is resolved, remove the lock with whichever of these methods you prefer. Note that the last option includes creating a new backup file:

1. Delete the lock file from the Datical DB project (example: `RefDb.lock)` and commit the deletion to the SCM repository for that project
2. Or, run `clearBackupRestoreLock` sub command of the `deployPackager.groovy` script with Packager's `preview` mode enabled. Here are variations of the command, the second command also creates a new backup.
    - Only clear the lock with this command:

```
hammer groovy deployPackager.groovy MAR_2020 scm=true preview=true
clearBackupRestoreLock
```

- Or, you can configure the command to clear the lock and also create a new backup of the RefDB:

```
hammer groovy deployPackager.groovy MAR_2020 scm=true preview=true
createDatabaseBackup=true clearBackupRestoreLock
```