

Co:Z Co-Processing Toolkit

z/OS Hybrid Batch Processing: Updating a Linux database from z/OS batch

V 1.0 Edition

Published October, 2012

Copyright © 2012 Dovetailed Technologies, LLC

Introduction

In a typical zEnterprise ecosystem, it is common to have Linux virtual servers that run customer facing web applications. In some cases, these applications use z/OS DB2 for their back end data store, but it is also typical to have this role fulfilled by an Oracle database on the Linux system itself.

Even if the database is local to the Linux system, however, it may need to be updated with data that is produced and/or held on z/OS. Traditionally, the process of applying these updates has involved a file transfer of a dataset containing the updates followed by a trigger that kicks off the utility (typically SQL *Loader) to load the database. This approach has several drawbacks:

- Coordination between the file transfer and SQL *Loader can be non-trivial
- The update data needs to exist in a file on the remote system before SQL *Loader can begin (i.e. unwanted "data at rest"). If it is sensitive, access policies need to be carefully considered.
- SQL *Loader status (error messages, logs, return codes) has to be checked on the remote system. This can be problematic for operations staff that are z/OS centric.

It would be ideal if this maintenance process could be scheduled and managed from z/OS, with just the database update program being present on the Linux system. In order to make this possible, the following considerations need to be addressed. How will SQL *Loader:

- access required input and output data, which is on z/OS?
- be scheduled, possibly so as to fit into the context of a larger job stream?
- direct status (log messages, monitoring, completion status, etc.) to z/OS so that it can be observed and acted upon?

The Co:Z Co-Processing Toolkit provides z/OS users with tools that mitigate these considerations while making it easy to exploit remote virtual servers (Linux on System z or Linux on zBX) from traditional batch workloads - an idiom we refer to as *z/OS hybrid batch processing*.

z/OS Hybrid Batch

We define *z/OS hybrid batch* as:

1. The ability to execute a program or script on a distributed (target) system from a z/OS batch job
2. The target program may already exist and should require little or no modification
3. The target program's input and output are redirected from/to z/OS spool files or data sets
4. The target program may access other z/OS resources: DD's, data sets, UNIX files and programs
5. The target program's exit code is adopted as the z/OS job step condition code

The following Co:Z Toolkit features and other programming techniques are used to implement the Hybrid Batch Processing model:

OpenSSH

The Co:Z Toolkit is built on top of OpenSSH, the industry standard secure shell implementation. OpenSSH provides the mechanism for establishing cryptographically secure connections between z/OS and a remote system.

Co:Z Launcher

The Co:Z Launcher is used to launch the target program on a remote virtual server, automatically setting up I/O redirections so that the target program standard streams (stdin/stdout/stderr) are redirected to z/OS jobstep DDs. Additionally, the launcher captures the target program's exit code and uses it to set the jobstep condition code.

Co:Z Dataset Pipes Utilities

The Co:Z Toolkit Dataset Pipes **fromdsn**, **todsn**, **fromfile** and **tofile** commands have a rich set of options, making it possible to stream data from and to z/OS from the remote process in a variety of ways. These options can greatly simplify the transfer and treatment of input and output data.

Process substitution

When the Co:Z Launcher starts a process on a target UNIX or Windows system, it runs a shell by default. On these systems, **bash** (Bourne Again SHell) is typically available. Bash has a very useful feature called "process substitution" which is a concise syntax for replacing filenames with input and output redirection. This feature can be used with Dataset Pipes commands to simplify the target program's access to z/OS file and data set resources.

Case Study: Data Administration with SQL *Loader and Co:Z

Oracle SQL *Loader is a utility that loads database tables from flat files. The command line invocation of the utility typically takes the following form:

```
sqlldr control=<control_file> data=<data_file> log=<log_file>
```

Where:

control - specifies information about the format of the data and related database field mapping information

data - the data to be loaded

log - destination for status and diagnostic messages

Upon completion, SQL *Loader will exit with one of the following return codes:

0 - success

1 - failure

2 - warning

3 - fatal error

The z/OS batch job that follows consists of two steps: The first step processes new customers and produces a temporary dataset containing customer data. The second step uses the Co:Z Launcher and SQL *Loader to load the new customer data into an Oracle database on a virtual Linux server. The annotated narrative following the JCL provides a step by step explanation of the processing that takes place.

```
//APPINT JOB ( ), 'COZ',MSGCLASS=H,NOTIFY=&SYSUID
//CUSTDATA EXEC PGM=CUSTCOB ❶
//OUTDD DD DSN=&&DATA,DISP=(NEW,PASS),
// UNIT=SYSDA,SPACE=(CYL,(20,20))
//COZLOAD EXEC PROC=COZPROC, ❷
// ARGV='myuser@linux' ❸
//CUSTCTL DD DSN=HLQ.CUST.CTL,DISP=SHR
//CUSTDATA DD DSN=&&DATA,DISP=(OLD,DELETE)
//CUSTLOG DD SYSOUT=*
//STDOUT DD SYSOUT=* ❹
//STDERR DD SYSOUT=* ❹
//STDIN DD * ❹
sqlldr control=<(fromdsn DD://CUSTCTL), \
data=<(fromdsn DD://CUSTDATA), \
log=>(todsn DD://CUSTLOG)
/*
//COZCFG DD *
ssh-tunnel=false ❺
//
```

The CUSTDATA job step ❶ runs the COBOL program CUSTCOB which processes new customers and writes their information to OUTDD, which is a temporary dataset.

The second job step, COZLOAD ❷ runs the Co:Z Launcher via the PROC COZPROC. The Launcher uses the connection information specified via the ARGS parameter ❸ to establish an OpenSSH connection to the virtual server named `linux` and authenticates with `myuser`¹.

Once the connection is established, the launcher starts a target program on the remote system. Unless otherwise specified, the target program is the user's default shell. On Linux, this is typically `bash`. As a final setup step, the launcher automatically sets up I/O redirection so that the target program's standard streams `stdin`, `stdout`, `stderr` are connected back to the controlling z/OS job step ❷.

When the shell starts on the Linux system, it reads from `stdin`. Because Co:Z has redirected `stdin`, the effect is that the shell reads the contents of the `STDIN` DD and begins executing this script which invokes the Linux command line version of SQL *Loader: `sqlldr`.

The launching process and I/O redirection illustrate the first and third principle of hybrid batch processing:

1. *The ability to execute a program or script on a distributed (target) system from a z/OS batch job*
3. *The target program's input and output are redirected from/to z/OS spool files or data sets*

It's important to emphasize that SQL *Loader is *not* running on z/OS; it's running on Linux. As such, it expects to read its control and data information from local files. However, this information is not local - it's on z/OS! To address this conundrum, the script uses *process substitution* to make the z/OS data available locally in a transparent fashion. Consider the argument to the `control` parameter: `<(fromdsn DD://CUSTCTL)` the `<()` syntax instructs `bash` to start a child process to execute the enclosed `fromdsn` command and route its output to a named pipe which can be read like a local file. `fromdsn` is a Co:Z Dataset Pipes command that reaches back into the active z/OS job step and streams the contents of a dataset (the `CUSTCTL` DD) to `stdout`. Recall that this output is written to the named pipe, making it available for reading by SQL *Loader. Once the command completes and all of the output is read, `bash` will automatically remove the named pipe. Process substitution is used in a similar fashion for the data that is to be loaded.

A slightly different form of process substitution is used to route SQL *Loader diagnostic information back to z/OS (specified as the `log` parameter on the `sqlldr` command). In this case, the syntax is of the form `>(todsn DD://CUSTLOG)`. The right angle bracket (`>`) indicates that the named pipe that `bash` creates is to be used in write, rather than read mode. Therefore, the effect is that the output from `sqlldr` is written to a named pipe whose read end is connected to the child process running the Co:Z Dataset Pipes `todsn` command. `todsn` takes its input and writes it to the dataset specified - in this case the `CUSTLOG` DD. Because this DD refers to `SYSOUT *`, the `sqlldr` progress can immediately be seen by viewing the `joblog`.

These process substitution examples illustrate the fourth principle of hybrid batch processing: *4. The target program may access other z/OS resources: DD's, data sets, UNIX files and programs* It's also very important to note that SQL *Loader executes without any modification; as far as it is concerned, it is reading and writing data to local files. This satisfies the second principle of hybrid batch processing: *2. The target program may already exist and should require little or no modification.*

Another important thing to note is that by using process substitution, there is no data at rest on the Linux system; the

¹Because batch is non-interactive, the user will typically be authenticated with an OpenSSH public/private key pair.

named pipes allow fromdsn and todsn simultaneously with sqllldr.

Finally, when sqllldr completes, it returns with one of the exit codes described above. The Co:Z launcher sets the job step condition code to this value. This satisfies the fifth principle of hybrid batch processing: 5. *The target program's exit code is adopted as the z/OS job step condition code.*

Data Security


One of the first questions that the astute observer will ask when looking at Hybrid Batch is that of z/OS data security. Namely:

- What z/OS resources can the remote target program access?
- What security measures are in place during network operations?

z/OS Resource Access

When the Co:Z Launcher launches the remote target program, there are two userids to consider: 1.) The userid of the remote system running on the Linux system and 2.) the z/OS userid that owns the batch job. The remote system userid security profile controls access to Linux programs and data, as appropriate. However, when the target program runs a Dataset Pipes command such as `fromdsn`, it is the SAF profile of the batch job owner that determines what z/OS resources can be accessed. This behavior is consistent with what z/OS operations and administration would expect.

Network security

By default, when data is transferred from z/OS to the remote system or from the remote system back to z/OS it is encrypted and tunneled back over the existing OpenSSH connection. This provides a high level of security, but at the cost of having to encrypt the data. When connecting to a remote Linux system over a non-secure network, this is the right approach. However, in the zEnterprise environment, there are two very secure network options: HiperSockets (for accessing Linux systems running on IFLs) and the Intra Ensemble Data Network (IEDN) connecting the zEnterprise to zBX blades. In this kind of environment, the Co:Z toolkit offers an option to disable data encryption and tunnelling. This can be seen in the example job in the Co:Z configuration DD, where `ssh-tunnel=false` is set . With this option specified, the data transfer performance can be extraordinarily good, in some cases coming close to co-located data.

Conclusion

This case study illustrates how Co:Z can be used to run the Oracle SQL *Loader utility on a remote system under the control of a z/OS batch job, and provides a simple but powerful demonstration of how Hybrid Batch Processing works. The z/OS hybrid batch model allows enterprises to:

- Optimize the use of heterogeneous computing resources from a z/OS batch environment.
- Extend the reach of z/OS to other platforms and application processes using existing z/OS data sets.
- Preserve investments in existing applications and processes. Changing programs to use new APIs is not required.

Appendix A. Legal Notices

- Copyright© 2012 Dovetailed Technologies, LLC. All rights reserved.
- Co:Z® is a registered trademark of Dovetailed Technologies, LLC.
- z/OS®, zEnterprise® and zBX® are trademarks of IBM Corporation.
- Oracle® and all Oracle-based trademarks and logos are trademarks or registered trademarks of Oracle Corporation.