DSXchange™

White Paper

What Is DataStage?

White Paper: What Is DataStage?

Sometimes DataStage is sold to and installed in an organization and its IT support staff are expected to maintain it and to solve DataStage users' problems. In some cases IT support is outsourced and may not become aware of DataStage until it has been installed. Then two questions immediately arise: "what is DataStage?" and "how do we support DataStage?".

This white paper addresses the first of those questions, from the point of view of the IT support provider. Manuals, web-based resources and instructor-led training are available to help to answer the second.

DataStage is actually two separate things.

- In production (and, of course, in development and test environments) DataStage is just another application on the server, an application which connects to data sources and targets and processes ("transforms") the data as they move through the application. Therefore DataStage is classed as an "ETL tool", the initials standing for extract, transform and load respectively. DataStage "jobs", as they are known, can execute on a single server or on multiple machines in a cluster or grid environment. Like all applications, DataStage jobs consume resources: CPU, memory, disk space, I/O bandwidth and network bandwidth.
- DataStage also has a set of Windows-based graphical tools that allow ETL processes to be designed, the metadata associated with them managed, and the ETL processes monitored. These client tools connect to the DataStage server because all of the design information and metadata are stored on the server.

On the DataStage server, work is organized into one or more "projects". There are also two DataStage engines, the "server engine" and the "parallel engine".

- The server engine is located in a directory called DSEngine whose location is recorded in a hidden file called /.dshome (that is, a hidden file called .dshome in the root directory) and/or as the value of the environment variable DSHOME. (On Windows-based DataStage servers the folder name is Engine, not DSEngine, and its location is recorded in the Windows registry rather than in /.dshome.)
-
-

- The parallel engine is located in a sibling directory called PXEngine whose location is recorded in the environment variable APT_ORCHHOME and/or in the environment variable PXHOME.

DataStage Engines

The server engine is the original DataStage engine and, as its name suggests, is restricted to running jobs on the server. The parallel engine results from acquisition of Orchestrate, a parallel execution technology developed by Torrent Systems, in 2003. This technology enables work (and data) to be distributed over multiple logical "processing nodes" whether these are in a single machine or multiple machines in a cluster or grid configuration. It also allows the degree of parallelism to be changed without change to the design of the job.

Design-Time Architecture

Let us take a look at the design-time infrastructure. At its simplest, there is a DataStage server and a local area network on which one or more DataStage client machines may be connected. When clients are remote from the server, a wide area network may be used or some form of tunnelling protocol (such as Citrix MetaFrame) may be used instead.

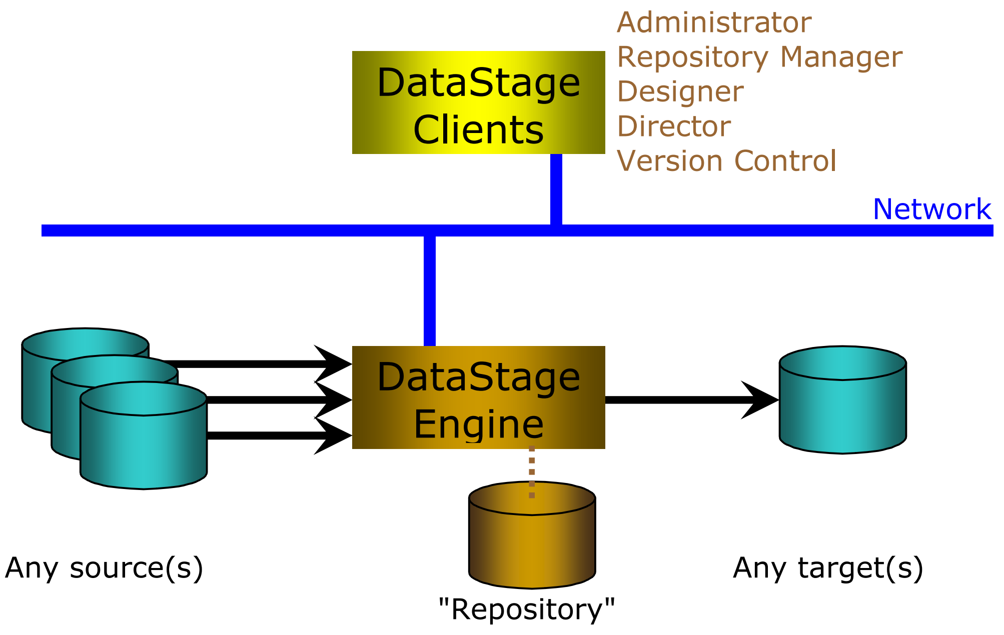

Figure 1 – DataStage Design-Time Architecture

Note that the Repository Manager and Version Control clients do not exist in version 8.0 and later. Different mechanisms exist, which will be discussed later.

Connection to data sources and targets can use many different techniques, primarily direct access (for example directly reading/writing text files), industry-standard protocols such as ODBC, and vendor specific APIs for connecting to databases and packages such as Siebel, SAP, Oracle Financials, etc.

DataStage Client/Server Connectivity

Connection from a DataStage client to a DataStage server is managed through a mechanism based upon the UNIX remote procedure call mechanism. DataStage uses a proprietary protocol called DataStage RPC which consists of an RPC daemon (dsrpcd) listening on TCP port number 31538 for connection requests from DataStage clients.

Before dsrpcd gets involved, the connection request goes through an authentication process. Prior to version 8.0, this was the standard operating system authentication based on a supplied user ID and password (an option existed on Windows-based DataStage servers to authenticate using Windows LAN Manager, supplying the same credentials as being used on the DataStage client machine – this option was removed for version 8.0). With effect from version 8.0 authentication is handled by the Information Server through its login and security service.

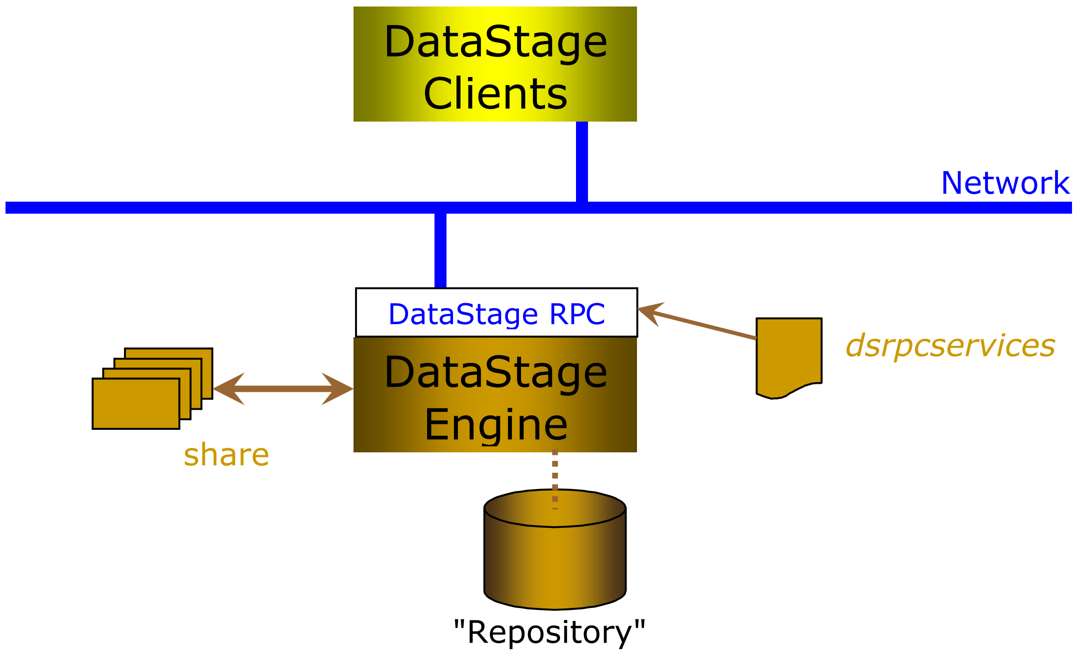

Figure 2 – Connecting to DataStage server from DataStage client

Each connection request from a DataStage client asks for connection to the dscs (DataStage Common Server) service.

The dsrpcd (the DataStage RPC daemon) checks its *dsrpcservices* file to determine whether there is an entry for that service and, if there is, to establish whether the requesting machine's IP address is authorized to request the service. If all is well, then the executable associated with the dscs service (dsapi_server) is invoked.

DataStage Processes and Shared Memory

Each dsapi_server process acts as the "agent" on the DataStage server for its own particular client connection, among other things managing traffic and the inactivity timeout. If the client requests access to the Repository, then the dsapi_server process will fork a child process called dsapi_slave to perform that work.

Typically, therefore, one would expect to see one dsapi_server and one dsapi_slave process for each connected DataStage client. Processes may be viewed with the **ps -ef** command (UNIX) or with Windows Task Manager.

Every DataStage process attaches to a shared memory segment that contains lock tables and various other inter-process communication structures. Further each DataStage process is allocated its own private shared memory segment. At the discretion of the DataStage administrator there may also be shared memory segments for routines written in the DataStage BASIC language and for character maps used for National Language Support (NLS). Shared memory allocation may be viewed using the **ipcs** command (UNIX) or the **shrdump** command (Windows). The shrdump command ships with DataStage; it is not a native Windows command.

DataStage Projects

Talking about *the* Repository is a little misleading. As noted earlier, DataStage is organized into a number of work areas called "projects". Each project has its own individual local Repository in which its own designs and technical and process metadata are stored.

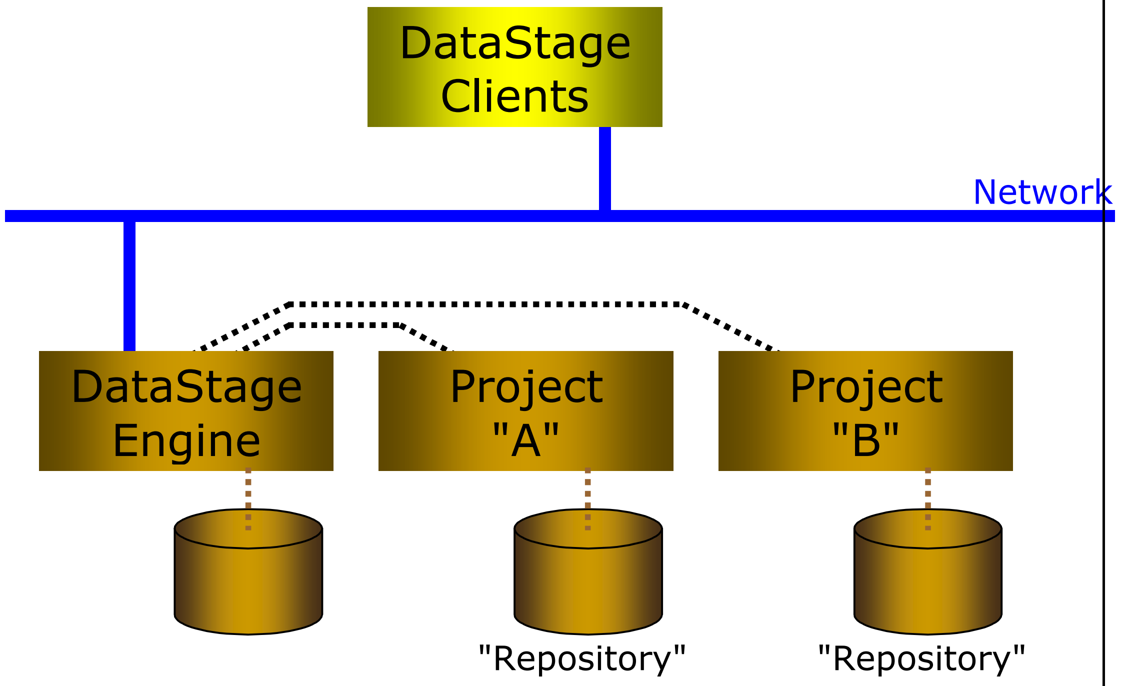

Figure 3 – DataStage Projects

Each project has its own directory on the server, and its local Repository is a separate instance of the database associated with the DataStage server engine. The name of the project and the schema name of the database instance are the same. System tables in the DataStage engine record the existence and location of each project.

Location of any particular project may be determined through the Administrator client, by selecting that project from the list of available projects. The pathname of the project directory is displayed in the status bar.

When there are no connected DataStage clients, dsrpcd may be the only DataStage process running on the DataStage server. In practice, however, there are one or two more.

The DataStage deadlock daemon (dsdlockd) wakes periodically to check for deadlocks in the DataStage database and, secondarily, to clean up locks held by defunct processes – usually improperly disconnected DataStage clients.

Job monitor is a Java application that captures "performance" data (row counts and times) from running DataStage jobs. This runs as a process called JobMonApp.

Changes in Version 8.0

In version 8.0 all of the above still exists, but is layered on top of a set of services called collectively IBM Information Server. Among other things, this stores metadata centrally so that the metadata are accessible to many products, not just DataStage, and exposes a number of services including the metadata delivery service, parallel execution services, connectivity services, administration services and the already-mentioned login/security service. These services are managed through an instance of a WebSphere Application Server.

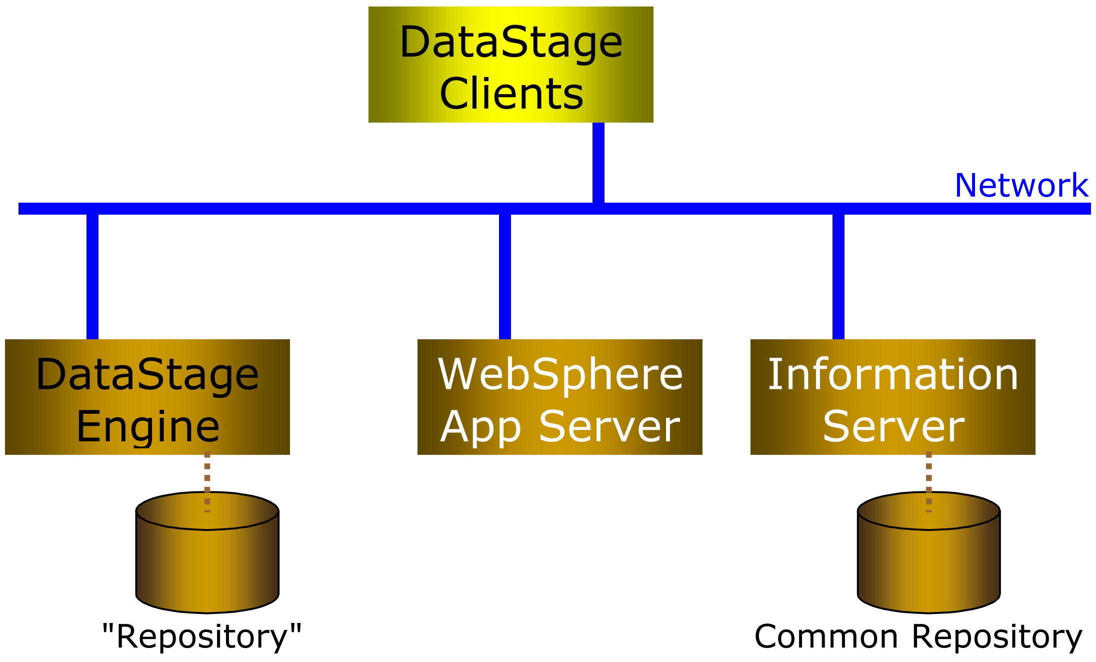

Figure 4 – Version 8.0 and later Infrastructure

The common repository, by default called XMETA, may be installed in DB2 version 9 or later, Oracle version 10g or later, or Microsoft SQL Server 2003 or later.

The DataStage server, WebSphere Application server and Information Server may be installed on separate machines, or all on the same machine, or some combination of machines. The main difference noticed by DataStage users when they move to version 8 is that the initial connection request names the Information Server (not the DataStage server) and must specify the port number (default 9080); on successful authentication they are then presented with a list of DataStage server and project combinations being managed by that particular Information Server.

Run-Time Architecture

Now let us turn our attention to run-time, when DataStage jobs are executing. The concept is a straightforward one; DataStage jobs can run even though there are not clients connected (there is a command line interface (dsjob) for requesting job execution and for performing various other tasks).

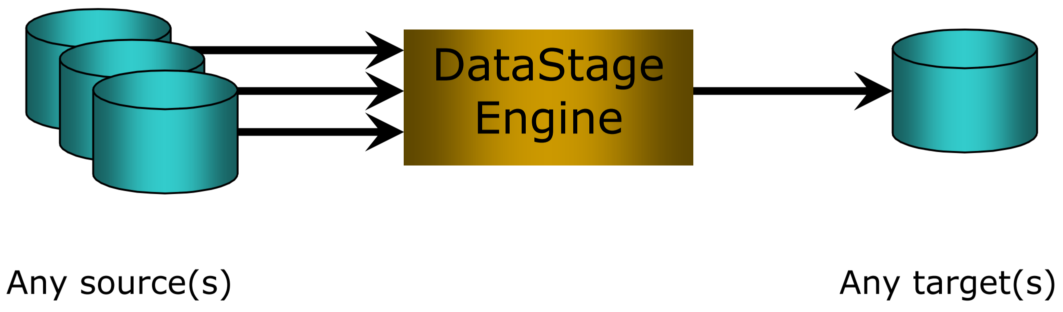

Figure 5 – DataStage at Run Time

However, server jobs and parallel jobs execute totally differently. A job sequence is a special case of a server job, and executes in the same way as a server job.

Server Job Execution

Server jobs execute on the DataStage server (only) and execute in a shell called uvsh (or dssh, a synonym). The main process that runs the job executes a DataStage BASIC routine called DSD.RUN – the name of this program shows in a ps –ef listing (UNIX). This program interrogates the local Repository to determine the runtime configuration of the job, what stages are to be run and their interdependencies. When a server job includes a Transformer stage, a child process is forked from uvsh also running uvsh but this time executing a DataStage BASIC routine called DSD.StageRun. Server jobs only ever have uvsh processes at run time, except where the job design specifies opening a new shell (for example sh in UNIX or DOS in Windows) to perform some specific task; these will be child processes of uvsh.

Parallel Job Execution

Parallel job execution is rather more complex. When the job is initiated the primary process (called the "conductor") reads the job design, which is a generated Orchestrate shell (osh) script. The conductor also reads the parallel execution configuration file specified by the current setting of the APT_CONFIG_FILE environment variable. Based on these two inputs, the conductor process composes the "score", another osh script that specifies what will actually be executed. (Note that parallelism is not determined until run time – the same job might run in 12 nodes in one run and 16 nodes in another run. This automatic scalability is one of the features of the parallel execution technology underpinning Information Server (and therefore DataStage).

Once the execution nodes are known (from the configuration file) the conductor causes a coordinating process called a "section leader" to be started on each; by forking a child process if the node is on the same machine as the conductor or by remote shell execution if the node is on a different machine from the conductor (things are a little more dynamic in a grid configuration, but essentially this is what happens). Each section leader process is passed the score and executes it on its own node, and is visible as a process running osh. Section leaders' stdout and stderr are redirected to the conductor, which is solely responsible for logging entries from the job.

The score contains a number of Orchestrate operators. Each of these runs in a separate process, called a "player" (the metaphor clearly is one of an orchestra). Player processes' stdout and stderr are redirected to their parent section leader. Player processes also run the osh executable.

Communication between the conductor, section leaders and player processes in a parallel job is effected via TCP. The port numbers are configurable using environment variables. By default, communication between conductor and section leader processes uses port number 10000 (APT_PM_STARTUP_PORT) and communication between player processes and player processes on other nodes uses port number 11000 (APT_PLAYER_CONNECTION_PORT).

To find all the processes involved in executing a parallel job (they all run osh) you need to know the configuration file that was used. This can be found from the job's log, which is viewable using the Director client or the dsjob command line interface.

About the Author

Ray Wurlod

Ray is a self-employed trainer and consultant for the IBM DataStage®, IBM UniVerse and IBM Red Brick Warehouse and DataStage® XE suites of products.

Ray has taught advanced classes in the USA, the UK and Germany, and has been used frequently as a training consultant by IBM to conduct advanced in-house training classes. Additionally, Ray has presented training classes in almost every country in the Asia-Pacific region, and has been involved in technical presentations and implementations throughout the region.

Ray joined Prime Computer of Australia in 1986. He later joined VMARK Software (original developers of DataStage®) after Prime Computer sold its database businesses to VMARK. Ray's principal role with VMARK, and subsequently with Ardent, Informix and IBM was as a DataStage® trainer, but was also actively involved in technical support. While with VMARK and Ardent he was actively involved in the development of DataStage®, creating a complete training curriculum for use in the Asia-Pacific region. He has also developed training curriculum and train-the-trainer programs for the UniVerse® RDBMS, including its NLS (national language support) implementation. When Ardent sold its database businesses to Informix Software, Ray continued his involvement in Data Warehouse technology by becoming expert with the Red Brick Warehouse product; a database designed specifically for Data Warehouse (star schema) implementations.

When Informix was acquired by IBM Ray continued his concentration on training, while additionally focusing on Data Warehousing applications.