Automating the Conversion of IBM DataStage to AWS Native Pipelines

EXECUTIVE SUMMARY

One of the major challenges organizations face in their transformation journey and adoption of cloud technologies is migrating and moving away from legacy systems built on mainframes or IBM Informix databases. In addition to managing large volumes of data and complex data structures, these migrations involve 1000s of DataStage-based ETL jobs that handle critical functions such as data integration, processing, and management. These migration processes are particularly challenging because these ETLs implement complex workflows with intricate dependencies, custom transformations, and orchestration to handle critical functions such as data integration, processing, and management.

Ventera, with extensive experience in transitioning data infrastructures from mainframes to on-prem and cloud infrastructure for federal and state agencies, offers a disruptive but proven approach to automating the creation of AWS-native pipelines from various legacy ETLs, including DataStage.

This paper outlines Ventera's repeatable approach for rapidly and accurately converting DataStage workflows into AWS cloud-native equivalents. By automating key steps, Ventera reduces migration time, safeguards data integrity, lowers infrastructure costs, and enables organizations to fully leverage AWS's advanced tools for enhanced data transformation and deeper insights.

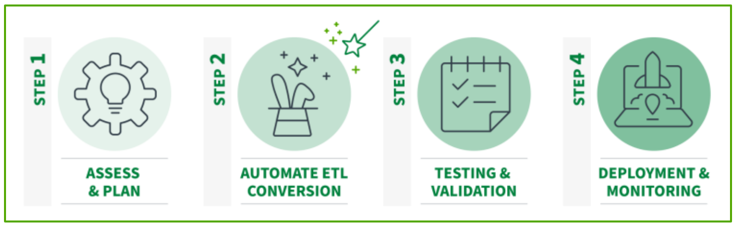
Ventera's innovation helps organizations confidently navigate this complex migration and achieve a seamless transition to a modern, cloud-native infrastructure.

Challenges in Migrating from IBM DataStage

Migrating from IBM DataStage to AWS is not without its challenges. Some of the most critical technical hurdles include:

- **Complex Workflows.** Many DataStage workflows have complex dependencies and custom logic that need to be recreated for the AWS environment.
- **Data Transformation Logic.** Transforming DataStage's proprietary transformation logic into PySpark or other AWS-native code is complicated and error-prone.
- **Minimizing Downtime.** Migration of production workloads must minimize downtime, especially for pipelines processing real-time or near-real-time data.
- **Orchestration.** DataStage provides built-in workflow orchestration, whereas, on AWS, orchestration requires custom integration with Step Functions, Glue Workflows, or Lambda triggers.

MIGRATION APPROACH



STEP 1: ASSESS & PLAN

1.1 Inventory DataStage Jobs. Ventera's first step is performing an inventory of your existing DataStage environment. This includes creating artifacts describing jobs, dependencies, and configurations – transformation logic, job schedules, input and output formats, database connections, and so on.

1.2 AWS Service Mapping. After the inventory is complete, the following step is mapping each of the DataStage jobs and the various sub-components of each job to fit with the available AWS service. For example:

- DataStage jobs for data transformation can typically be mapped to AWS Glue.
- Complex workflows or parallel jobs may map to AWS Step Functions.
- You want to set up extraction work for those data jobs, you can spin up AWS Glue Crawlers – and save the data retrieved to databases on Amazon RDS or Amazon Redshift for querying.

STEP 2: AUTOMATE JOB CONVERSION

2.1 Extract DataStage Metadata. DataStage jobs are typically stored in XML format, and job definitions (stages, connectors, transformations can be extracted programmatically. We use a mix of Python and XML parsing libraries to retrieve job metadata, which tells us the overall structure of the job (e.g., which SQL is used in which stage, in which order) – and thus, where we should begin the conversion.

2.2 Lexer-Based Code Generation for Transformers to AWS Glue PySpark. A common challenge is converting the transformation logic. DataStage "Transformers" contains custom data manipulation logic, which needs to be converted into AWS Glue PySpark scripts. Glue uses Apache Spark under the hood, which is well-suited to handle the kinds of complex transformations typically seen in DataStage.

Ventera automates this translation using a Lexer-based code generation that parses DataStage transformation logic and generates the equivalent PySpark code in AWS Glue based on a JSON based template. The Lexer works in multiple stages:

- **Lexical Analysis.** DataStage XML is tokenized into its basic components, such as filters, joins, transformations, data sources, and destinations.
- **Syntax Parsing.** Tokens are analyzed and mapped to corresponding PySpark equivalents.
- **Code Generation.** Lexer then dynamically generates the appropriate PySpark code for AWS Glue.

Advanced transformations that involve filters, joins, and aggregations can now be automatically translated into AWS Glue code. This greatly reduces the number of manual errors that may occur during these translations, which were handled manually in the past, and considerably reduces the time it takes for the conversion to be completed. For example, if a DataStage Transformer performs a filter followed by a join, a corresponding PySpark job could look like this:

```
from datalex import Lexer  # Importing the Lexer module
# Input DataStage XML or logic for water quality and climate data
transformations
datastage_job_xml = """
<DataStageJob>
<TransformStage>
<FilterCondition>temperature > 30 and water_quality ==
'good'</FilterCondition>
<JoinCondition>station_id</JoinCondition>
</TransformStage>
</DataStageJob>
"""
# Initialize the Lexer and pass the DataStage job
lexer = Lexer(datastage_job_xml)
# Parse and generate PySpark code
pyspark_code = lexer.generate_pyspark()
# The generated PySpark code (output by the Lexer) would look like:
print(pyspark_code)
# Output (Equivalent PySpark Code)
"""
df1 =
glueContext.create_dynamic_frame.from_catalog(database='climate_d
b', table_name='temperature_data')
df2 =
glueContext.create_dynamic_frame.from_catalog(database='water_db'
, table_name='water_quality_data')
# Transformation: filter and join based on temperature and water
quality
filtered_df1 = df1.filter(f.col('temperature') > 30)
filtered_df2 = df2.filter(f.col('water_quality') == 'good')
joined_df = filtered_df1.join(filtered_df2, 'station_id', 'inner')
# Write result to S3
glueContext.write_dynamic_frame.from_options(frame=joined_df,
connection_type='s3',
connection_options={'path': 's3://climate-water-analysis/output'},
format='parquet')
"""
```

In this example:

- **Lexer Initialization.** The Lexer class is initialized with the DataStage XML job definition.
- **Parsing & Code Generation.** The generate_pyspark() method automatically parses the XML logic, such as filters and joins, and dynamically generates the PySpark code required for AWS Glue.
- The Lexer outputs a **fully functional PySpark job** that replicates the same filtering and joining logic from DataStage without manual intervention.

This automation through Lexer eliminates manual coding, ensuring accurate and fast migration from DataStage to AWS Glue while maintaining data transformation integrity.

2.3 Convert DataStage Sequencers to AWS Step Functions. DataStage sequencers control the execution of ETL jobs, including conditional logic, looping, and parallelism. AWS Step Functions can be used to replicate these sequencing tasks. Ventera's approach involves automating the conversion of DataStage sequencers to AWS Step Functions using AWS SDK (Boto3), providing a streamlined and scalable approach.

- **Extract Metadata.** Export DataStage Sequencer metadata in XML format (.dsx or .isx). This contains the sequence logic, job dependencies, error handling, and parallel execution details.

```
import xml.etree.ElementTree as ET
def extract_metadata(file_path):
  """Extract metadata from DataStage export file."""
  tree = ET.parse(file_path)
  root = tree.getroot()
  jobs = []
  for job in root.findall(".//Job"):
    jobs.append({
      'name': job.find('Name').text,
      'type': job.find('Type').text,
      'next_job': job.find('NextJob').text
    })
  return jobs
# Example extraction from a water and climate data sequence file
jobs = extract_metadata('datastage_climate_water_sequence.dsx')
```

- **Parse & Analyze Metadata.** Use a custom Python script to parse the exported XML and extract job details, transitions, and conditions. This includes identifying individual jobs, types, and their respective sequence flows.

```
def parse_metadata(jobs):
  """Parse and analyze metadata for job sequencing."""
  parsed_jobs = []
  for job in jobs:
    parsed_jobs.append({
      'name': job['name'],
      'type': job['type'],
      'next_job': job['next_job']
    })
  return parsed_jobs
# Parsed metadata from climate and water jobs
parsed_jobs = parse_metadata(jobs)
```

- **Map DataStage Jobs to AWS Services.** Define a mapping strategy to associate DataStage jobs with appropriate AWS services such as Lambda, Glue, or Batch. For example, ETL jobs may translate to AWS Glue, while simple scripts can be handled by Lambda.

```
def map_jobs_to_aws(parsed_jobs):
  """Map DataStage jobs to AWS services for Step Functions."""
  state_machine = {
    "StartAt": parsed_jobs[0]['name'],
    "States": {}
  }
  for job in parsed_jobs:
    state_machine['States'][job['name']] = {
      "Type": "Task",
      "Resource": f"arn:aws:lambda:us-east-
1:123456789012:function:{job['name']}",
      "Next": job['next_job'] if job['next_job'] else "End"
    }
  return state_machine
# Map jobs from DataStage to AWS for climate and water data
processing
state_machine = map_jobs_to_aws(parsed_jobs)
```

- **Generate Step Functions JSON.** Dynamically generate a state machine definition in JSON format, replicating the sequencer logic using Step Functions states like Task, Choice, Parallel, and Catch.

```
def generate_step_function_json(state_machine):
  """Generate JSON for AWS Step Functions."""
  state_machine_json = json.dumps(state_machine, indent=4)
  return state_machine_json
# Generate AWS Step Functions JSON for water and climate data processing
workflow
step_function_json = generate_step_function_json(state_machine)
print(step_function_json)
```

Output:

```
{
  "StartAt": "IngestWaterData",
  "States": {
    "IngestWaterData": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-
1:123456789012:function:IngestWaterData",
      "Next": "ProcessClimateData"
    },
    "ProcessClimateData": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-
1:123456789012:function:ProcessClimateData",
      "Next": "End"
    }
  }
}
```

- **Deploy Step Functions Programmatically.** Utilize Boto3 to create and deploy the generated Step Function state machine in AWS, including role-based permissions and task resource definitions.

```
import boto3
stepfunctions_client = boto3.client('stepfunctions')
def deploy_step_function(state_machine_json):
  """Deploy Step Function state machine in AWS."""
  response = stepfunctions_client.create_state_machine(
    name='WaterClimateDataWorkflow',
    definition=state_machine_json,
    roleArn='arn:aws:iam::123456789012:role/StepFunctionsRole'
  )
  return response
# Deploy the Step Function for climate and water data processing
deploy_step_function(step_function_json)
```

2.3 Convert DataStage Data Connectors to AWS Glue Crawlers or Amazon RDS. DataStage has connectors that work with databases and file systems. In AWS, Glue Crawlers can be used to make schema discovery, while RDS or Redshift can be used for database integration and management. Ventera uses automation to convert DataStage data connectors to AWS Glue Crawlers, RDS, or Redshift by parsing the DataStage metadata and automatically generating the equivalent AWS infrastructure and connection configurations.

- **Extract DataStage Connector Information.** Use XML parsing to retrieve metadata from DataStage connectors (e.g., database type, connection details, tables, file systems).

- **Auto-Generate Glue Crawlers.** Based on the extracted metadata, a Python script can dynamically create Glue Crawlers to discover and catalog the schema from databases or file systems, automating the connection setup.

```
import boto3
# Initialize the Glue client
glue_client = boto3.client('glue')
# Auto-generate Glue Crawler based on DataStage
connector info for climate data
response = glue_client.create_crawler(
  Name='ClimateDataCrawler',

Role='arn:aws:iam::123456789012:role/GlueCrawlerRole',
  DatabaseName='climate_db',
  Targets={
    'S3Targets': [
      {
        'Path': 's3://climate-data/temperature/',
      }
    ]
  },
  TablePrefix='climate_'
)
print(f"Glue Crawler created: {response}")
```

- **Automate RDS/Redshift Connection Setup.** If the DataStage connector points to a relational database (e.g., Oracle, SQL Server), you can map the connection details to Amazon RDS or Redshift using Boto3 to automate database creation and connection management.

Our approach significantly reduces manual effort, ensuring accurate and consistent replication of data pipelines while also offering scalability and flexibility with AWS services.

STEP 3: TESTING & VALIDATION

3.1 Unit & Integration Testing. Testing is critical to ensure that migrated pipelines behave identically to their DataStage equivalents. Automated unit tests should be created for each AWS Glue job and Step Function workflow to verify the correctness of the data transformations and job execution flow. Integration testing can ensure that the entire pipeline—extraction, transformation, and loading—works seamlessly in AWS.

3.2 Data Validation. After ETL migration is complete, you can schedule batch jobs to auto-validate/compare your DataStage jobs in your source/legacy environment with your new pipelines in AWS. You can automatically profile your data using AWS Glue's feature in DataBrew, which is also an awesome option for consistency checking, etc.

STEP 4: DEPLOYMENT & MONITORING

4.1 Production Deployment. After rigorous testing, the AWS-native pipelines can be deployed to production. AWS Glue Workflows or Lambda triggers can be used to automate job scheduling and orchestration. The deployment process should be automated using AWS CloudFormation or Terraform to manage infrastructure as code.

4.2 Monitoring & Optimization. AWS CloudWatch can monitor job execution, and AWS CloudWatch Alarms can trigger notifications (e.g., Amazon SNS topic or Amazon Simple Email Service) in the case of job failure. AWS Glue job metrics can be ingested for visibility into how the pipeline is running.

KEY CONSIDERATIONS FOR COMPLEX MIGRATIONS

Moving from IBM DataStage to AWS comes with many challenges when dealing with large and complex datasets, such as those used for water quality monitoring, climate data analysis, or other mission-critical processes. Such a complex migration relies heavily on a deep understanding of both the source environment and AWS-native services. The following are critical success factors for such a migration to minimize business-disruption, maintain data integrity and performance while migrating to AWS.

- **Data Integrity.** Ensure data accuracy during migration using AWS Glue DataBrew for automated validation, profiling, and consistency checks across datasets like climate or water quality measurements.
- **Large-Scale Data Transfers.** When handling large datasets (e.g., water and climate records), use AWS Snowball or DataSync for secure, efficient migration, reducing the risk of data loss or corruption during transfer.
- **Minimizing Downtime.** Use AWS DMS to replicate data continuously, minimizing downtime and ensuring legacy systems remain operational until AWS-native pipelines are fully functional.
- **Security & Compliance.** Protect sensitive environmental data with AWS IAM for fine-grained access control and KMS for encryption of data in transit and at rest. Use AWS CloudTrail for logging API activity to meet regulatory compliance.
- **Performance Optimization.** Use AWS CloudWatch to monitor Glue job execution and performance. Optimize Spark configurations in Glue for large-scale transformations like climate data analytics or water quality assessments.

- **Complex Workflow Orchestration.** Replicate the complex sequencing of DataStage jobs with AWS Step Functions. Use parallel states, choice states, and custom error handling to maintain the integrity of workflows.
- **Custom Data Transformation Logic.** Automate the conversion of custom transformation logic from DataStage to AWS Glue PySpark using Lexer-based code generation. Validate with automated testing to ensure accuracy and functionality.
- **Real-Time Data Processing.** For real-time data ingestion and processing (e.g., live water monitoring or climate sensor data), leverage AWS Kinesis with Lambda for real-time data streaming and processing.
- **Disaster Recovery and Backup.** Set up AWS Backup for automatic, scheduled backups of critical data. Use Cross-Region Replication to ensure data resiliency and disaster recovery readiness for mission-critical data.

CONCLUSION

Migrating IBM DataStage ETL jobs to AWS Native Pipelines offers organizations substantial scalability, cost efficiency, and innovation benefits. By leveraging automation—through automated parsing, script-based conversion, and AWS-native orchestration- our approach reimagines this transformation, simplifies the migration process, reduces errors, and accelerates the transition to a modern cloud-based data processing framework.

Ventera (now proudly part of Cadmus) is your trusted partner, bringing the experience, expertise, and the "magic of innovation" to ensure a smooth, secure, and future-proof migration to the cloud, empowering your organization to thrive in today's rapidly evolving data landscape.

Contact us today to explore the possibilities.