**Pentaho Data Integration (PDI) Developer Guide**

## Resources

* DevOps2 – PDI Project Setup and Lifecycle Management.pdf
* DeveloperGuide\_SanteeCooper.pdf

## GitLab

### GitLab PDI Practices

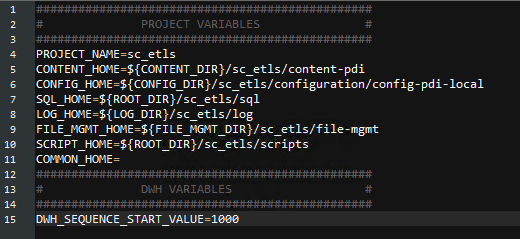
### Git Flow (See GitLab\_Flow.vsdx for a diagram)

* Production Branch
  + Is deployed on the PDI production server MCETL01P
  + Merge requests only come from the master branch after testing on the PDI dev server
* Master Branch
  + Is deployed on the PDI development server MCETL01D
  + Is used to create branches for feature creation or bug fixes
  + Can be merged to the production branch after testing on the PDI dev server
* Feature Branches
  + Can be created from the master branch for development and testing of new features like new jobs/transformations/etls, new projects, or enhancements/modifications to existing objects
  + Can be merged to the master branch for testing on the PDI dev server after testing on developer local systems
* Bug Fix Branches
  + Can be created from the master branch for fixing bugs and testing
  + Can be merged to the master branch for testing on the PDI dev server after testing on developer local systems

## Creating a New PDI Project

1. Create a new project in GitLab in the ETL subgroup
   1. Navigate to **ETL** subgroup in GitLab
   2. Click **New Project**
      1. use naming conventions below
      2. check **Initialize repository with a README**
   3. Project naming conventions
      1. Use snake case (lower case words separated by underscores)
      2. Prefix projects with categories (i.e. dwh\_, hr\_, gen\_, fleet\_, etc.)
      3. Examples:
         1. dwh\_matdis
         2. hr\_dashboard
         3. fleet\_app
2. Clone the project to your local system
   1. In GitLab
      1. Navigate to the project page in GitLab
      2. Click **Clone**
      3. Copy the **Clone with HTTP** link
   2. In GitHub Desktop
      1. Click **File -> Clone Repository …**
      2. Click on **URL**
      3. Paste the copied URL in the **URL or username/repository** box
      4. For local path, browse to **C:\etlbase\projects**
         1. The project name will be appended (C:\etlbase\projects\new\_project)
      5. Click **Clone**
3. Copy an existing project to the new project folder on your local system
   1. In an existing project (i.e. sc\_etls), copy the following folders (if they exist) to the new folder created when you cloned the new project
      1. configuration
      2. content-pdi
      3. log
      4. sql
      5. scripts
      6. file-mgt
   2. DO NOT copy the following items (maybe hidden, File Explorer -> View -> check Hidden Items to see)
      1. .git folder
      2. .gitignore
      3. (the .git folder should already exist in the new project if you followed these instructions)
4. Cleanup in the new project folders and subdirectories
   1. configuration
      1. config-pdi-dev/properties
         1. Rename the project level properties file (i.e. change sc\_etls.properties to new\_project.properties)
         2. Edit the new\_project.properties file, replacing the old project name with the new project name wherever it appears in the file
         3. Delete any job specific properties files (i.e. jb\_main\_old\_project\_job.properties)
      2. config-pdi-local/properties
         1. Rename the project level properties file (i.e. change sc\_etls.properties to new\_project.properties)
         2. Edit the new\_project.properties file, replacing the old project name with the new project name wherever it appears in the file
         3. Delete any job specific properties files (i.e. jb\_main\_old\_project\_job.properties)
      3. config-pdi-prod/properties
         1. Rename the project level properties file (i.e. change sc\_etls.properties to new\_project.properties)
         2. Edit the new\_project.properties file, replacing the old project name with the new project name wherever it appears in the file
         3. Delete any job specific properties files (i.e. jb\_main\_old\_project\_job.properties)

Example sc\_etls.properties. Need to replace sc\_etls with the new project name in all three config-pdi- folders.



* 1. content-pdi
     1. Delete all folders except util
  2. log
     1. Delete all contents within except README.md
  3. sql (if exists)
     1. Delete all contents within
  4. scripts (if exists)
     1. Delete all contents within
  5. file-mgt (if exists)
     1. Delete all contents within

1. Example of a new project’s directory structure once setup
   1. .git/
      1. Content related to the new GitLab repo created back in step 1
   2. configuration
      1. config-pdi-dev/properties
         1. project\_name.properties
      2. config-pdi-local/properties
         1. project\_name.properties
      3. config-pid-prod/properties
         1. project\_name.properties
   3. content-pdi
      1. util
         1. any utility jobs from the previous project
            1. i.e. jb\_main\_check\_db\_connections.kjb, etc.
      2. will eventually house folders for each major job
   4. log
      1. empty
   5. sql
      1. empty
   6. scripts
      1. empty
   7. file-mgt
      1. empty

## Managing Properties Files

* It’s best practice to parameterize (create variables) for any inputs that vary between local development, development server, and production server such as database connection info, fileserver info, table names, etc.
* .properties files named after projects or jobs must have names that exactly match the project or job they’re for

### Framework Level Properties

* framework/configuration/config-pdi-local(dev, prod)/properties/common.properties
* Store variables available across all projects

### Project Level Properties

* project\_name/configuration/config-pdi-local(dev, prod)/properties/project\_name.properties
* Store variables here for use with all jobs/transformations within this project

### Job Level Properties

* project\_name/configuration/config-pdi-local(dev, prod)/properties/jb\_main\_job\_name.properties
* Store variables here for use with only specific jobs
* Good idea to set this at the jb\_main level

### Changing the Properties Files that Spoon is Using

* Because .properites files can be very customized, you might need to change which .properties files your using if you begin working on a different project in a spoon session
* Follow the instructions below in Set Dev Environment in Spoon

## Launching and Configuring Spoon on the Local System

### Launching Spoon

* Launch spoon from C:\etlbase\projects\platform-configuration\config-pdi-local using one of the methods below
  + Double clicking **spoon.bat**
  + cmd prompt cd to C:\etlbase\projects\platform-configuration\config-pdi-local, run command **spoon**
* This spoon.bat file in this directory sets up paths needed for local execution of ETLs, transformations, and jobs
* **Always set the dev environment when starting spoon**

### Set Dev Environment in Spoon

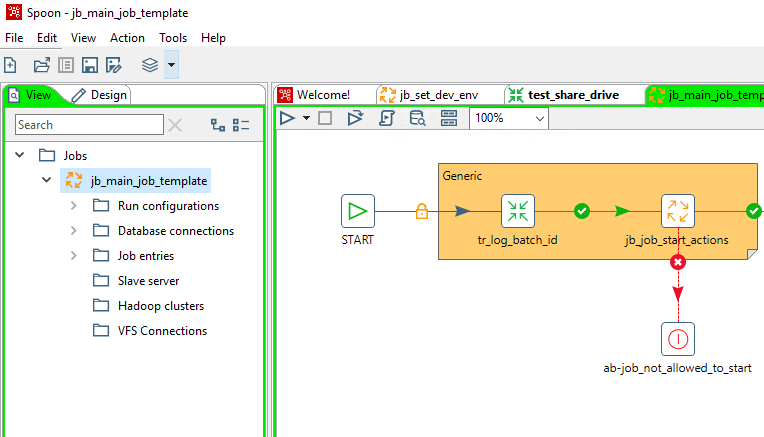
* Basic info
  + Spoon uses various .properties files, and these are highly customizable at the platform (local, dev, prod) level, project level, and even job level
  + Spoon must know which .properties files to use to access saved database connections, variables/parameters, etc.
* Every time you start spoon or want to change the .properties files that are being used at the project, job, or work unit level, follow these steps:
  1. Open C:\etlbase\projects\framework\content-pdi\developer-tools\jb\_set\_dev\_env.kjb
  2. Click Run
  3. Enter values for the parameters to use
     + P\_JOB\_NAME (optional)
       - to use any specific job .properties files
     + P\_PROJECT\_NAME (required)
       - set the project level .properties file to use
     + P\_WORK\_UNIT\_NAME (optional)
       - set any additional work unit .properties files to use
       - in most cases, work unit .properties would be included in the job level .properties file instead of a separate work unit .properties file

## Connections to Data Sources

* Naming Conventions for Connections
  1. Oracle
  2. Chronus
  3. SQL Server
     + JNDI
     + Native
* Creating New Connections
* Adding Connections to .properties Files for Saving/Sharing Connections

## Building ETL Jobs

* General Info
  + The platform-configuration, framework, and project level files work together to cooridate job/transformation execution
  + Follow this template to orchestrate execution of ETLs
* Hierarchy of ETL Files
  1. framework/content-pdi/execution/jb\_launcher.kjb
     + Controls high level settings when running jobs
  2. project\_name/content-pdi/major\_job\_name/jb\_main\_job\_name.kjb
     + Uses work units to orchestrate jobs and transformations that make up this ETL
  3. project\_name/content-pdi/major\_job\_name/ other jobs and transformations for the ETL
     + These are the actual .kjb and .ktr files that execute the ETL
* Creating ETLs
  1. Create and design transformation/jobs needed to execute the ETL process
  2. Copy framework/content-pdi/developer-tools/jb\_main\_job\_template.kjb to your project\_name/content-pdi/major\_job\_name/ folder
     + Rename to jb\_main\_job\_name.kjb
     + Open the job, then right click the jb\_main\_job\_template name in the left panel and rename to match your filename



* 1. Create Work Units for every .ktr and .kjb file created in Creating ETLs Step 1 in the jb\_main file (sample image below)
     + First transformation or job in the ETL
       - Rename jb\_work\_unit to the name of the object to run (i.e. jb\_work\_unit: tr\_extract\_transform\_load\_eamworkorders.ktr)
       - Set parameters in the work unit
         * Double click jb\_work\_unit: step
         * Job: path

${FRAMEWORK\_HOME}/execution/jb\_work\_unit.kjb

* + - * + Click the Parameters tab
        + Ensure Pass parameter values to sub job is checked
        + Fill out the values to use for all parameters listed in Static input value (most common) if they don’t change or Parameter to use if using variables

P\_PROJECT\_NAME

exact folder name where the project is

P\_WORK\_UNIT\_NAME

exact name of the .ktr or .kjb file (omit the file extension)

P\_WORK\_UNIT\_TYPE

transformation or job

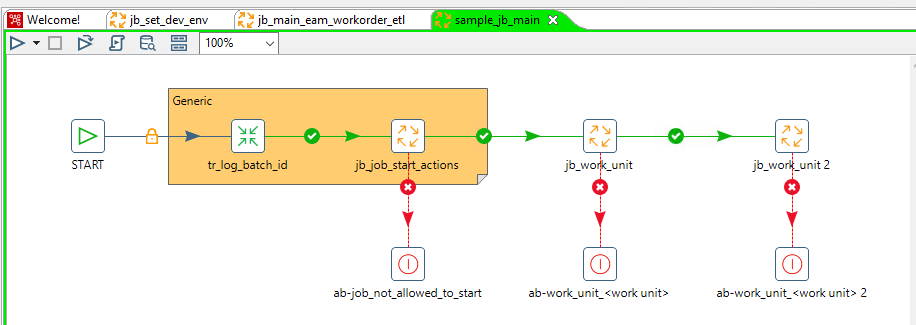
P\_IS\_LAST\_UNIT

Y or N to indicate if this is the last work unit (job/transformation) in this ETL

P\_WORK\_UNIT\_PATH

major\_job\_name folder within the project’s folder

* + - * Rename ab-work\_unit\_<work\_unit> to match the object to run (i.e. ab-work\_unit\_tr\_extract\_transform\_load\_eamworkorders)
    - Subsequent Work Units
      * Copy the jb\_work\_unit step
      * Create a hop for successful execution (right click hop -> Evaluation -> Follow When Result is True) from the previous work unit step to the new one
      * Copy the ab-work\_unit step
      * Create a hop for failed execution (right click hop -> Evaluation -> Follow When Result is False) from the new work unit step
      * Set the parameters in the work unit as above



* File Naming Conventions
  1. Keep the bold part then name accordingly
     + Main Job
       - **jb\_main**\_primary\_job\_name.kjb
     + Jobs
       - **jb**\_what\_the\_job\_does.kjb
     + Transformations
       - **tr**\_what\_the\_transformation\_does.ktr

## Running Transformations, Jobs, and ETLs on the Local System

* Follow steps in Launching Spoon and Set Dev Environment in Spoon to ensure correct use of .properties files

### Running Transformations and Jobs on the Local System

* Transformations and Jobs can be run by clicking the Run button for testing purposes
* Previewing output of a step
  1. Previewing output of a step will run the entire transformation/job, including writing to any files or databases if all hops are enabled
  2. To preview data without writing to databases or files, disable hops to output steps before previewing

### Running Complete ETL on the Local System

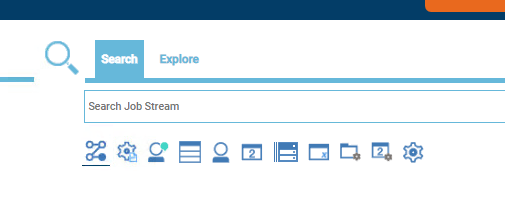
* Create the ETL script using the guidelines in Building ETL Jobs
  1. Ensure that the jb\_main file is created and configured properly
* Use C:/etlbase/projects/framework/content-pdi/execution/jb\_launcher.kjb to run complete ETL scripts
  1. Click Run in jb\_launcher.kjb
  2. Enter values for the parameters
     + P\_JOB\_NAME
       - the name of the jb\_main file for this ETL
     + P\_JOB\_PATH
       - major\_job\_name folder within the project’s folder
     + P\_PROJECT\_NAME
       - name of the project folder
  3. Click Run

## Deploying a New Project to IWS

* Submit a ticket to SIG requesting:
  1. creation of project folder (same name as your project folder) on the server (MCETL01D or MCETL01P) in the /etlbase/projects directory
  2. creation of IWS job to pull and deploy (IWS job name has \_PD\_ in it), and a schedule for which this job should be run
     1. schedules for pull/deploy jobs
        1. a good time to schedule is around the server reboots Sunday night, but other schedules can be requested as needed on a case by case basis if this doesn’t work
        2. scheduling the pull/deploy job will help keep things up to date on the server
  3. Ask for the pull deploy job names to be added to the ticket comments when complete
* Once created, make note of the IWS pull deploy job name for each server
  1. format: MCETL01D#SIMCC\_PD\_<project\_name> (MCETL01P for production)
* **Ensure GitLab is Updated**
  1. **Development**
     1. Ensure the **master** branch in GitLab has been updated
        1. Commit and push all needed changes
  2. **Production**
     1. Ensure the **production** branch in GitLab has been updated
        1. If this branch doesn’t yet exist…
           + GitHub Desktop -> choose your project in Current repository dropdown
           + Branch -> New branch…
           + Name

production

* + - * + Create branch
* Decide if you need to create a new job stream in IWS
  1. Similar jobs that are for the same process that run on the same schedule can go in the same job stream
  2. If not creating a new job stream, you can skip that step and proceed to Create a new ETL job in IWS
* Before running Pack and Deploy Jobs in IWS for PRODUCTION
  1. EMERGENCY CHANGE request
     1. Use only when updates to an existing project must be run immediately to fix something that is broken
     2. Not needed when you are first deploying a project to production in most circumstances
     3. Steps
        1. If possible, get approval from your immediate supervisor first
        2. Create the Emergency Change Request in Jira
        3. Close it upon running the Pack/Deploy job
  2. NORMAL CHANGE request
     1. Use when updating and existing project and you can wait a few days before instituting changes
     2. Not needed when you are first deploying a project to production in most circumstances unless your project has ETL’s that are replacing existing processes that will be stopped as a result of this project
     3. Steps
        1. Create a Norml Change Request in Jira
        2. Wait for approval
        3. Run the Pack and Deploy Job only after receiving approval
* Run the Pack and Deploy Job in IWS
  1. In IWS go to Planning -> Submit Predefined Job
  2. Search for the appropriate workstation
  3. Check the appropriate job SIMCC\_PD\_<project\_name>
  4. Scroll down and click OK
  5. Check that it completed successfully
     1. Monitoring & Reporting -> Monitor Jobs -> search for [/@/MCETL01D#/@/@.@](mailto:/@/MCETL01D#/@/@.@) (or P)
     2. check that it completed successfully
* **Create a new job stream in IWS**
  1. Open an existing job stream for reference if desired (use one on MCETL01P/D) using the Workload Designer
  2. Make sure the zigzag is selected to search Job Streams (the gear is for Jobs)

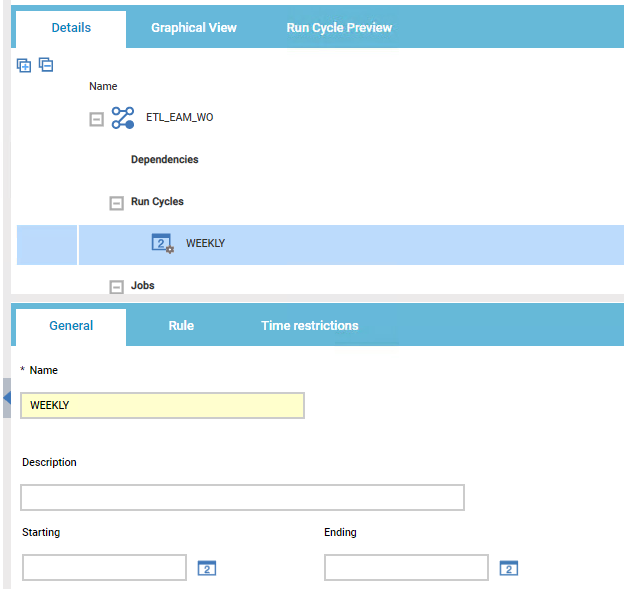


Jobs

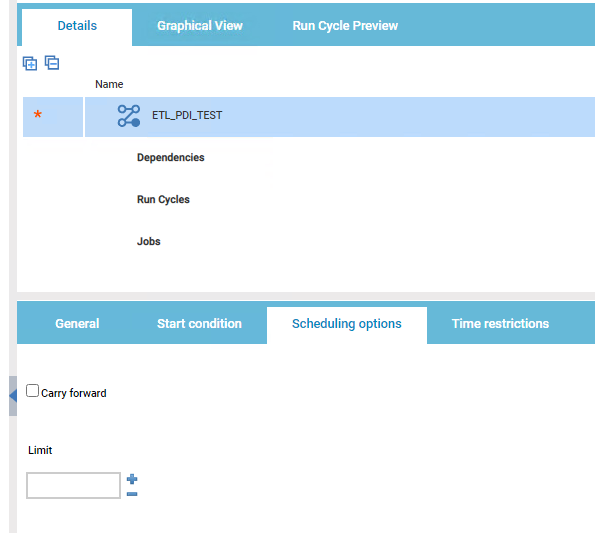
Job streams

Job streams

* 1. Search for ETL\_\* and look for one in /MCETL01D or P
  2. General Tab
     1. Folder
        1. /
     2. Name
        1. ETL\_<job\_stream\_name>
     3. Workstation
        1. /MCETL01D (or P)
     4. Description
        1. brief description
  3. Click on the Run Cycles section to get details for scheduling



* 1. Run Cycle Settings
     1. General Tab
        1. Name
           + WEEKLY (or similar)
     2. Rule Tab
        1. Repeat schedule
           + Weekly (or choose best option)
        2. Run every selected number of ……
           + Enter best option (1 for once a week)
        3. If weekly, need to check all days
     3. Time Restrictions Tab
        1. Check Use time as dependency
  2. New Job Stream Creation
     1. In Workload Designer, click Create New button at the top near the search bar
     2. Click the plus sign to expand Dependencies, Run Cycles, and Jobs



* + 1. Click on Run Cycles, right-click, and choose Add Run Cycle
       1. General Tab
          - Name

WEEKLY (or similar)

* + - 1. Rule Tab
         * For weekly, choose Weekly and pick one day to run
         * For daily, choose Weekly and check every day
         * The plan goes in at 4pm, so if you want Monday at 5 am, choose Sunday at 5am

I know it’s weird

* + - 1. FINISH AFTER FIXING IWS PERMISSIONS
* **Create a new ETL job in IWS**
  1. Open an existing ETL job in the Workload Designer for reference (has \_ETL\_ in the job name)
     1. Copy the script from the existing job (Task tab)
  2. Click Create New -> Job Definition -> UNIX
     1. General Tab
        1. Folder
           + /
        2. Name
           + Naming convention for dev

DMMMM\_ETL\_<etl\_job\_name>

* + - * + Naming convention for prod

DMPPP\_ETL\_<etl\_job\_name>

* + - 1. Workstation
         * /MCETL01D (or P)
      2. Login
         * pentaho
      3. Description
         * short description
    1. Task Tab
       1. copy the Script Name from an existing ETL
       2. check Script
          - paste the Script in the new ETL you’re creating
          - NEED TO EDIT THE PARAMETERS

P\_JOB\_NAME

jb\_main\_<job\_name>

P\_JOB\_PATH

folder in content-pdi where the job is saved

P\_PROJECT\_NAME

the primary name of the project (example: sc\_etls)

* Run a single job in IWS
  1. Planning -> Submit Predefined Job
  2. Click Go on the default engine
  3. Enter the workstation name (MCETL01D or P)
  4. Click the ellipses
  5. Choose the correct job
  6. Scroll down and click OK
  7. Copy the job name and past it in the alias box, then add your initials and maybe a time or iteration number
  8. Click OK
* Monitor jobs in IWS
  1. Monitoring & Reporting -> Monitor Workload
  2. in Query: edit it to say
     1. [/@/MCETL01D#/@/@.@](mailto:/@/MCETL01D#/@/@.@) (or P as needed)
  3. Click Run
  4. Click the refresh button (curved arrow) as needed to view the status of the job

## Error Logs and Email Rules for Forwarding Logs

* Error Logs
  1. Locating Error Logs
     1. Login to [datamgt@santeecooper.com](mailto:datamgt@santeecooper.com) using the Exchange [Santee Cooper Outlook Exchange](https://exchange.santeecooper.com/owa/auth/logon.aspx?replaceCurrent=1&url=https%3a%2f%2fexchange.santeecooper.com%2fowa%2f)
     2. Check the etld or etlp folders
* Forwarding Error Log Emails
  1. Create Email Rules to Forward Error Logs to Your Email
     1. Login to [datamgt@santeecooper.com](mailto:datamgt@santeecooper.com) using the Exchange [Santee Cooper Outlook Exchange](https://exchange.santeecooper.com/owa/auth/logon.aspx?replaceCurrent=1&url=https%3a%2f%2fexchange.santeecooper.com%2fowa%2f)
     2. In the top right of the toolbar, click the Gear icon -> Options
     3. Click on Inbox and sweep rules on the left
     4. Click the + to add a new rule
     5. Fill out the form
        1. Name
           + something descriptive like Forward Alarms PDI/ETL Errors
        2. When the message arrives, …
           + It includes these words -> in the subject…
           + jb\_main\_<job\_name>
           + click the + to add the text to the rule
           + add as many phrases as you want
           + click OK
        3. Add condition (use this to add more conditions if desired)
        4. Do all of the following
           + forward, redirect, or send -> Forward the message to…
           + type names in the Search People box and click the magnifying glass
           + click the + to add a recipient
           + add as many recipients as necessary
           + click Save above the recipients
        5. Add exception (if necessary)
        6. Stop processing more rules
           + uncheck this box to ensure you don’t mess up anyone else’s rules
           + usually not an issue, but still best practice
           + another way around this is to move your new rule to the bottom of the list once it’s created to ensure it gets processed last
        7. Click OK to save

## Redeploying an Existing Project on IWS

* Development Server (MCETL01D)

1. Ensure all changes are pushed to the Master Branch in GitLab
2. Developers run the pull deploy (\_PD\_) job in IWS for the appropriate project
   1. Required to update the project on the server using the GitLab repo
3. If a new ETL was added to the project, you need to follow the steps above to **Create a new ETL job in IWS** and decide if you can add the new job to an existing job stream or if you also need to follow the steps to **Create a new job stream in IWS**
4. If you modified an existing ETL that already has an IWS job, and you do NOT need to make changes to the schedule, you are done after running the pull/deploy job

* Production Server (MCETL01P)

1. Ensure all changes are merged to the Production Branch in GitLab
   1. If this is a new project and this branch doesn’t exist, see [Deploying a New Project to IWS](#_Deploying_a_New)
2. If it’s not an emergency change that needs to be implemented immediately, check the job stream for the pull deploy to see if it will run automatically
   1. If the pull deploy IWS job isn’t part of a scheduled job stream, and you can wait a few days to run pack/deploy…
      1. Create a normal change request
      2. Wait for approval
      3. Then run pack/deploy and create/edit any jobs/job streams that need editing
   2. If it’s an emergency change, developers create an emergency change ticket, the Data Analyst on-call should then re-run the pull deploy (\_PD\_) job on the production server
      1. Try to get Glenn’s approval first if possible
      2. Create the Emergency Change Ticket
      3. Run pack/deploy and create/edit any jobs/job streams that need editing
      4. Close the Emergency Change Ticket
3. If a new ETL was added to the project, you need to follow the steps above to **Create a new ETL job in IWS** and decide if you can add the new job to an existing job stream or if you also need to follow the steps to **Create a new job stream in IWS**
4. If you modified an existing ETL that already has an IWS job, and you do NOT need to make changes to the schedule, you are done after running the pull/deploy job

## Creating an AES Encrypted Password

* Rationale
  1. You can generate AES encrypted passwords which allows you to save credentials in the common.properties, <project\_name>.properties, or <job\_name>.properties files
  2. Saving these credentials makes using and sharing connections to databases or other sources very simple
  3. The passwords remain secure while also allowing the parameterization and sharing of saved connections
* Steps
  1. It is important to NEVER SAVE the tr\_encrypt\_password.ktr file with a clear text password
     + always inspect the framework project in GitLab when complete
     + if there are changes to the tr\_encrypt\_password.ktr file, then discard those changes
  2. Launch spoon using standard procedures
  3. Open C:\etlbase\projects\framework\content-pdi\developer-tools\tr\_encrypt\_password.ktr
  4. Double click the gv: get password step to edit it
  5. In the dialogue box, enter your unencrypted password in the Variable for clear\_password, replacing the ${P\_CLEAR\_PASSWORD} text, then click OK
  6. Right-click the d1 step, then choose Preview…
  7. Click Quick Launch
  8. Copy the contents of the encrypted\_password field to the appropriate .properties file
     + format should be (copy everything including the AES and any special characters)
       - AES kdkfja38somelongstringkdl3921=
  9. Close the tr\_encrypt\_password.ktr file when complete WITHOUT SAVING