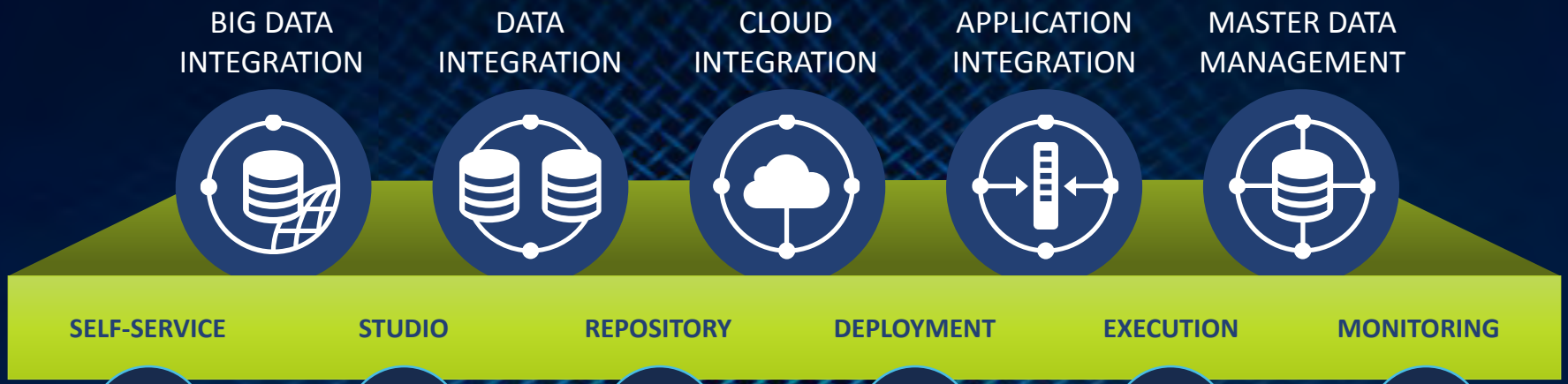
Data Integration Advanced

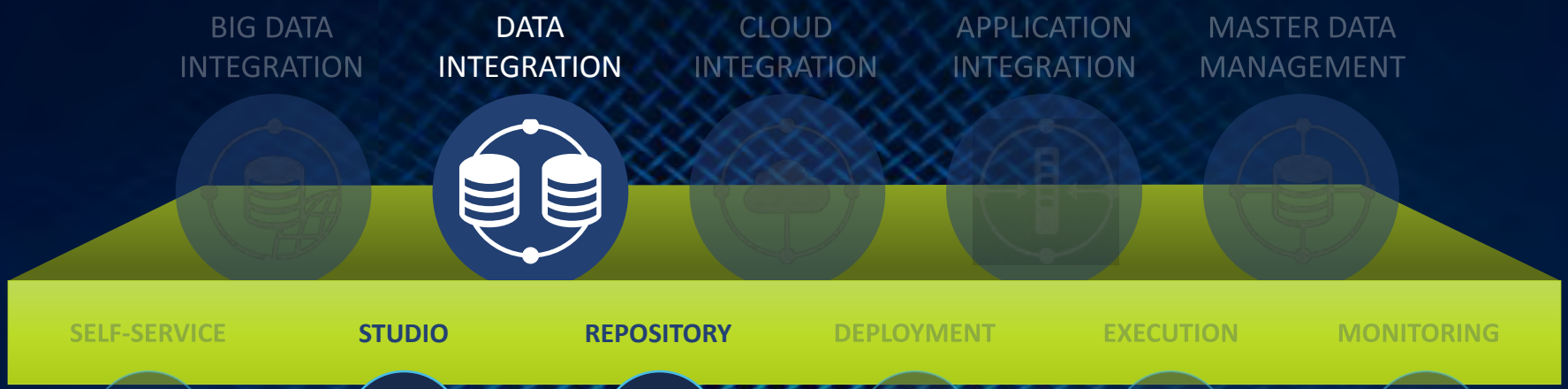
Version 6.3

talend | Data Fabric



1. Discovery and cleansing for business users
2. Comprehensive Eclipse-based user interface
3. Consolidated metadata and project information
4. Web-based deployment & scheduling
5. Same container for batch processing, message routing & services
6. Single web-based monitoring console

talend | This course focuses on:



1. Discovery and cleansing for business users
2. Comprehensive Eclipse-based user interface
3. Consolidated metadata and project information
4. Web-based deployment & scheduling
5. Same container for batch processing, message routing & services
6. Single web-based monitoring console

Course Table of Contents

1. Remote repository
2. SVN in the studio
3. Remote Job execution
4. Resource usage and basic debugging
5. Activity Monitoring Console
6. Parallel execution
7. Joblets
8. Unit test
9. Change data capture (CDC)

Connect to a Remote Repository

Objectives

- Start the Talend Administration Center service
- Configure Talend Studio to use a remote repository connection
- Start the Studio and use a project with a remote repository connection

Scenario

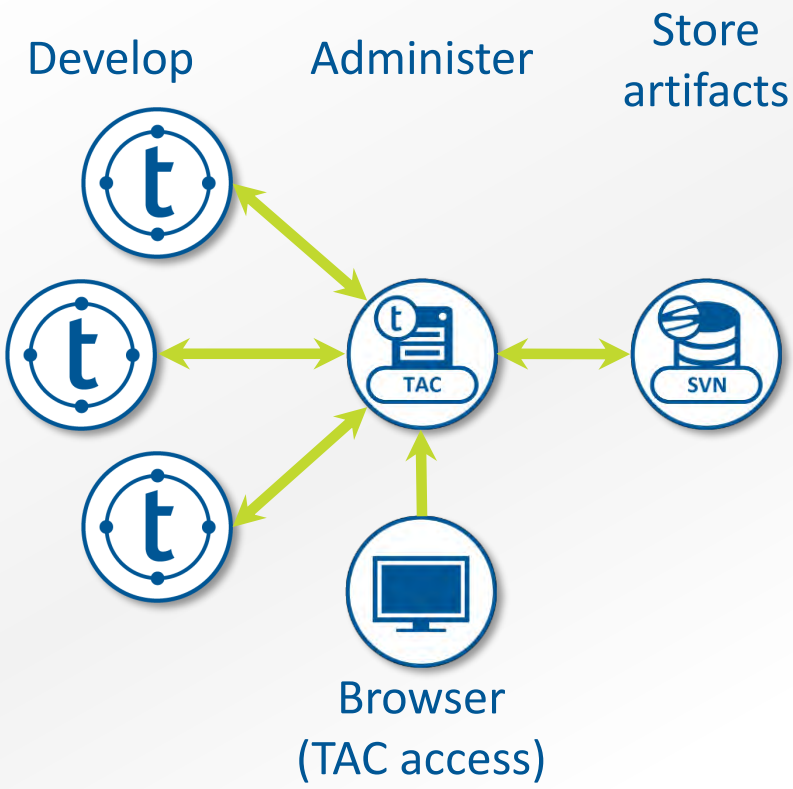
- Your company has multiple developers, each with their own instance of the Talend Studio. They all need collaborate on a single project.
- A shared repository in a central location:
  - keeps assets (metadata, Jobs, and Joblets for example) in sync
  - avoids duplication of effort
  - minimizes errors

Talend Administration Center

- The Talend Administration Center (TAC) uses an Apache Subversion (SVN) repository to store project, Job, and metadata information.
- Through the TAC, you can manage users, groups, project authorizations, and connection and license information.

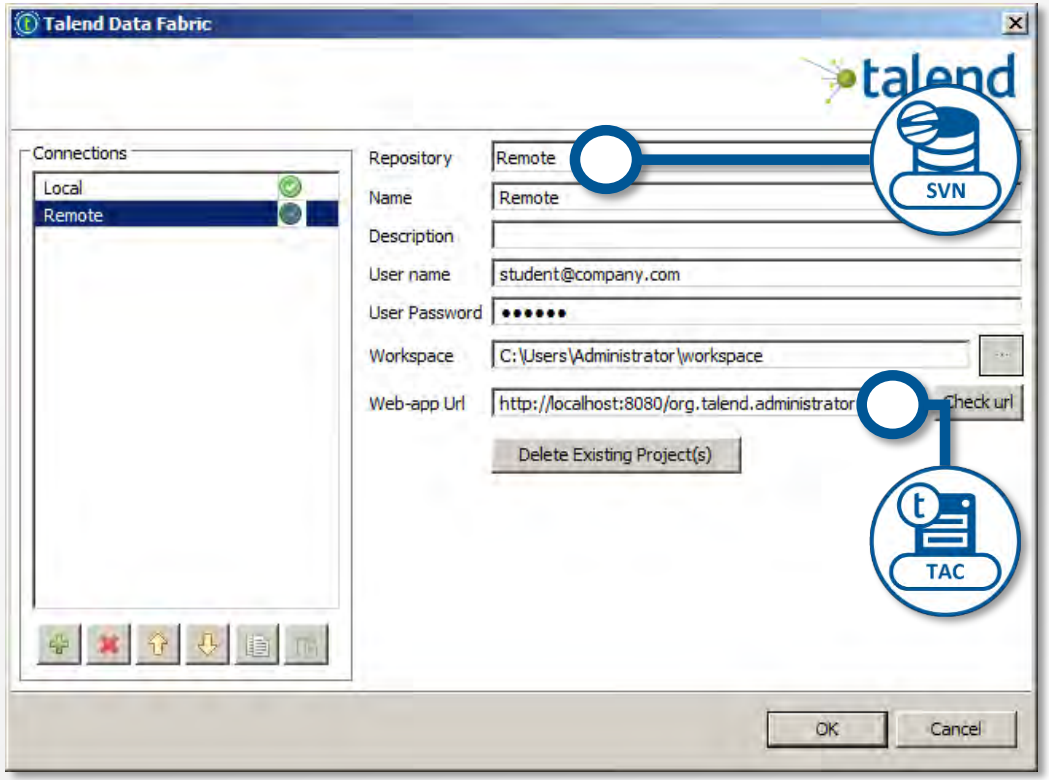
Talend Administration Center

- When developers use Talend Studio via a remote connection, they work with projects stored in the SVN repository and managed by the TAC



Project Configuration

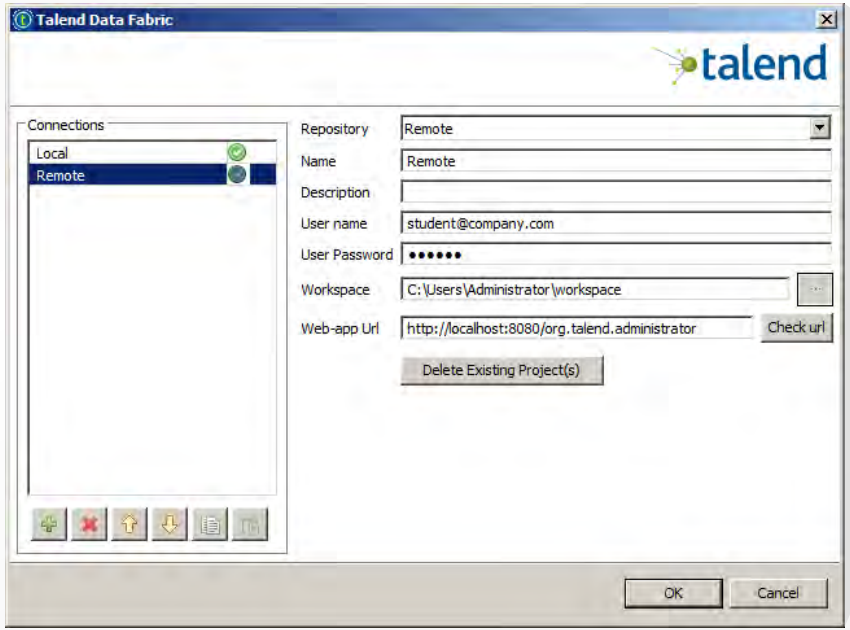
- Specify connection details when managing a project with a connection to a remote SVN repository
  - With the TAC installed, running, and communicating with SVN, it is simple to change your project from *local* to a *remote connection*



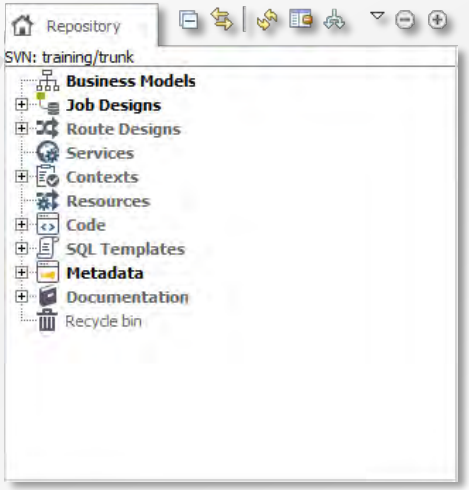
Lab Overview

Connect to a Remote Repository

- Configure a remote connection



- Develop in a remote shared repository



Lesson Summary

- TAC must be running as a service in order to configure and connect to a remote repository
- TAC connection configuration is simple
  - Set-up is typically a one-time task.
- When developers use a project configured with a remote connection, artifacts are stored on SVN, allowing team collaboration

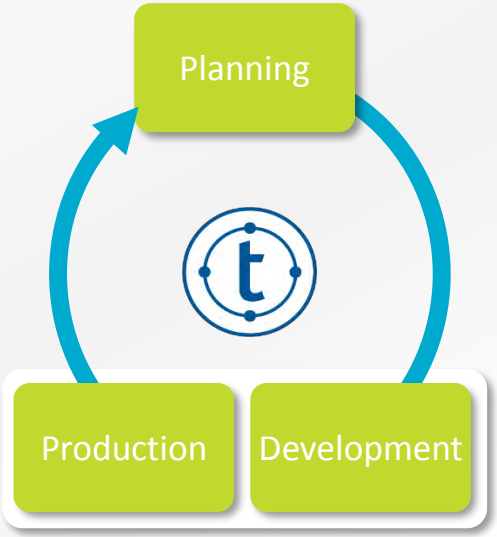
SVN in Studio

Objectives

- Identify the current SVN branch
- Switch between SVN branches
- Copy a Job between branches
- Compare Jobs between branches

Scenario

- As development progresses toward a new production release, there is a need for two separate development streams:
  - A fully tested, stable version to be released to the public
  - A developmental version containing new features for a future release
- How does this work within the Studio?
  - Solution requires source-code revision control
  - Studio supports both SVN and Git



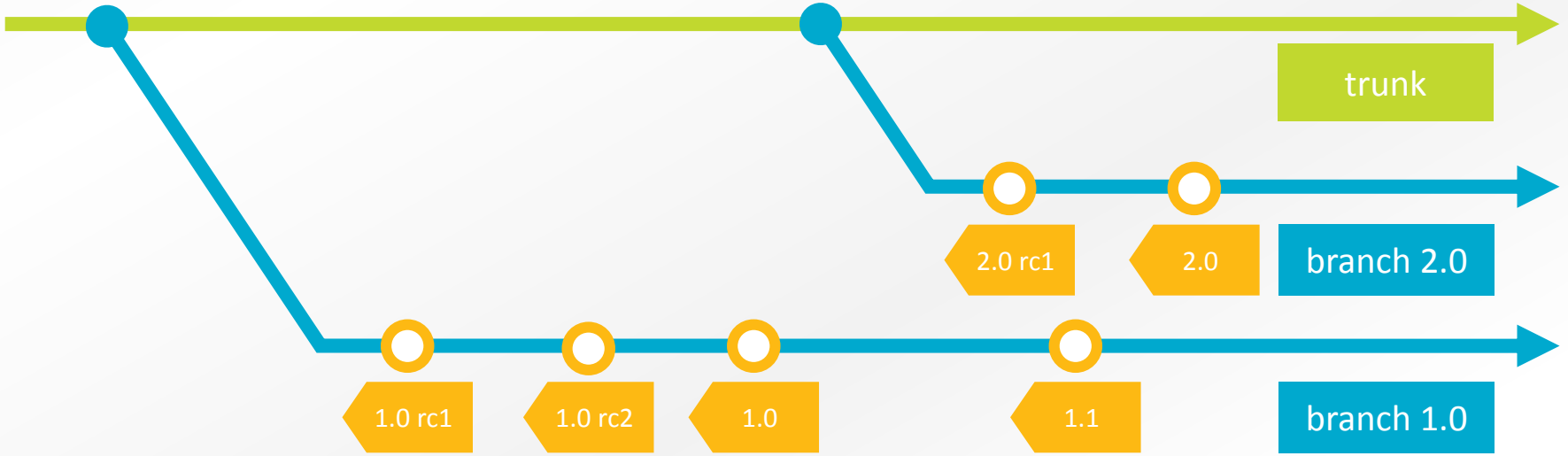
Revision Control

Common Industry Terms

Trunk: ongoing development takes place here

Branch: writeable copy of **trunk** taken at a particular point in time

Tag: read-only snapshot of a branch at a particular point in time



Revision Control

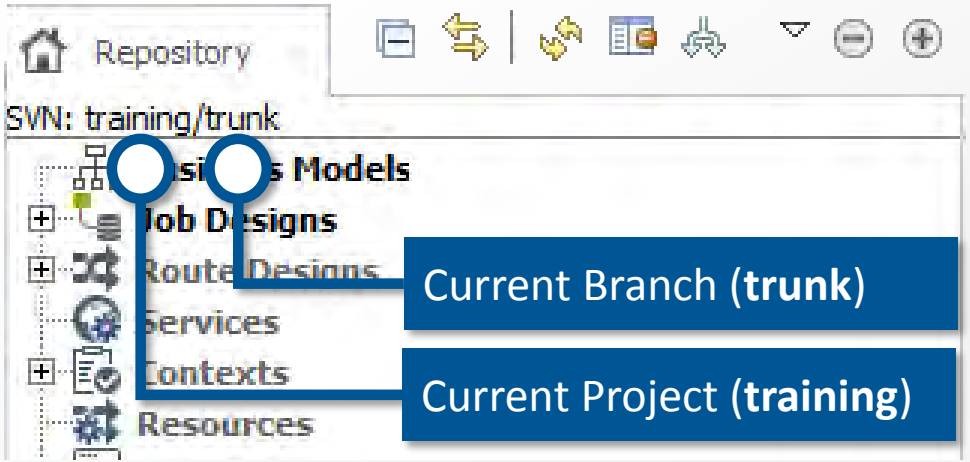
Common Operations Supported by Studio

- Branch management
  - **Switch:** change to a different branch in the repository
  - **Tag:** create a tag from a branch (create a snapshot)
- **Copy to branch:** copy selected artifacts from current repository to a branch
- **Compare Job:** highlight differences between Jobs in different branches

Revision Control

Switching Between Branches

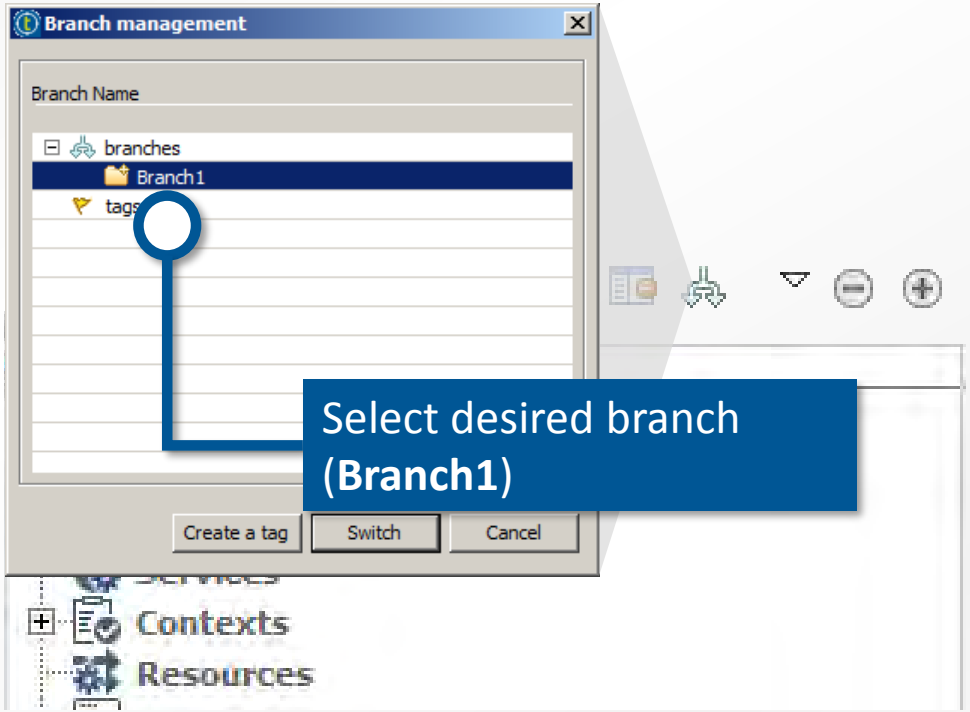
- The **Repository** view identifies the current Repository, project, and branch set in the Studio



1. Identify Current Branch
2. Select New Branch
3. Confirm New Branch

Revision Control

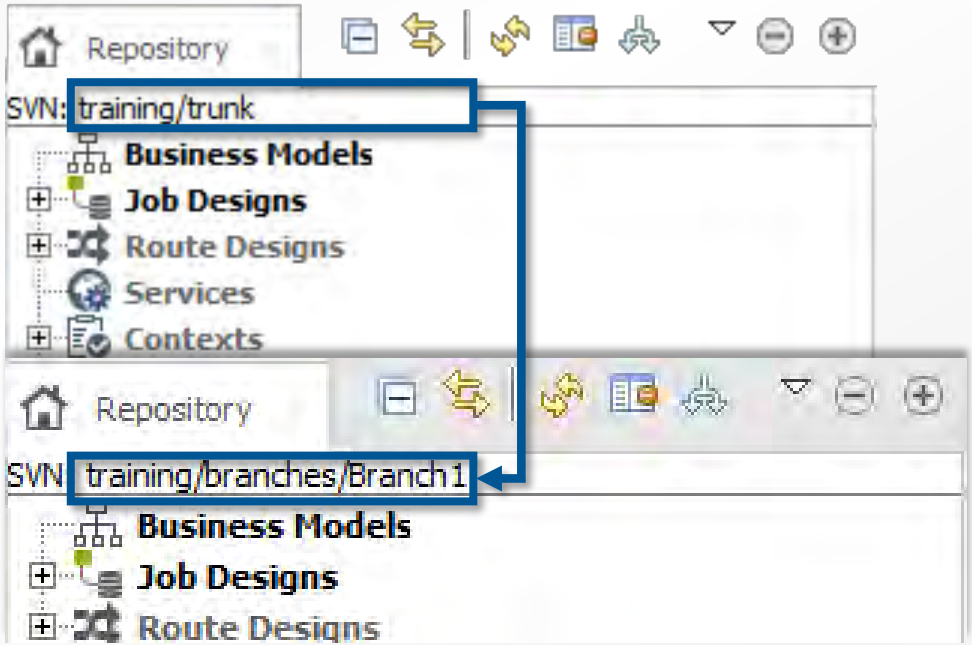
Switching Between Branches



1. Identify Current Branch
2. Select New Branch
3. Confirm New Branch

Revision Control

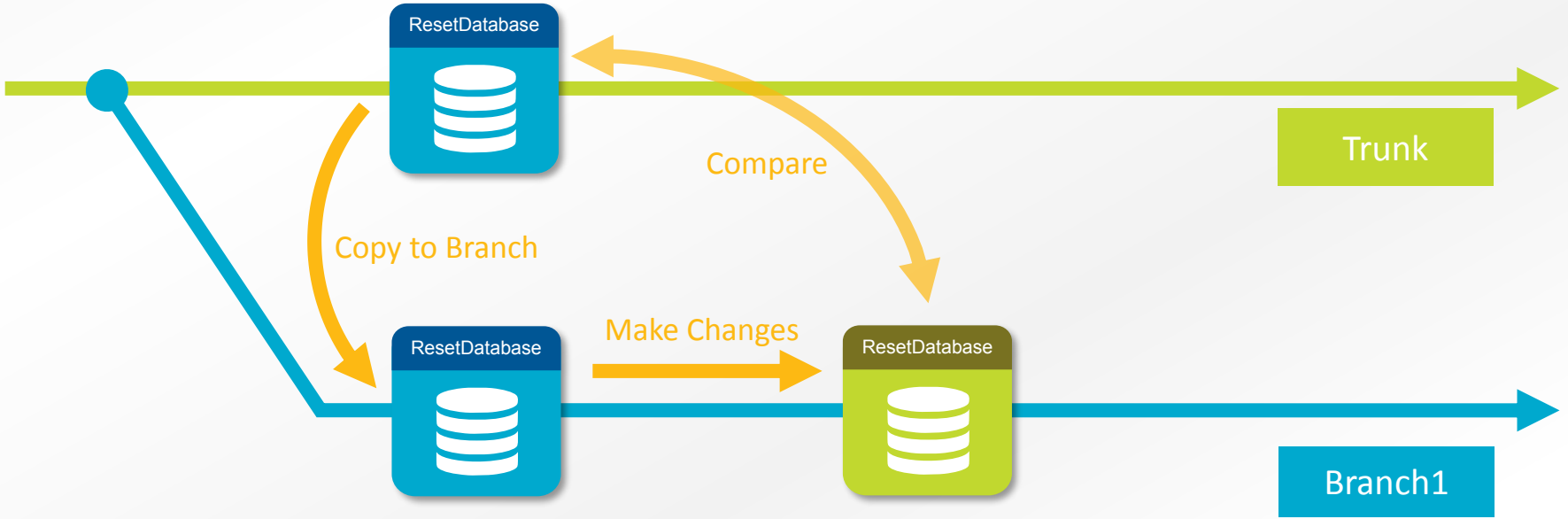
Switching Between Branches



1. Identify Current Branch
2. Select New Branch
3. Confirm New Branch

Lab Overview

- Switch branches, copy and compare Jobs between branches



Lesson Summary

- SVN is a commonly used revision control system
- SVN is integrated within Talend Studio
- For projects with remote connections, the **Repository** view identifies important information
  - Revision control system
  - Project Name
  - Current branch

Lesson Summary

- Basic revision control operations are simple to use in Studio
  - Switch between branches
  - Copy between branches
  - Compare Jobs between branches
  - Create tags

Remote Job Execution

Objectives

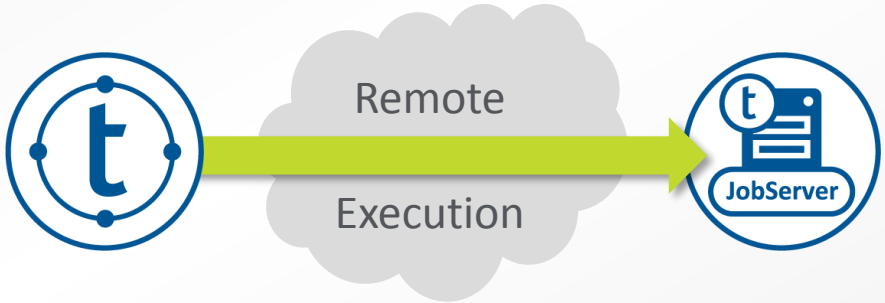
- Configure Studio to be able to run Jobs on a remote host
- Start the Talend JobServer on a remote host
- Configure and run a basic Job so it executes on a remote host

Scenario

- For performance reasons you need additional hosts on which to run Jobs
- The development and test environments need to test new Job features on hosts other than the local development machine

Talend JobServer

Overview



Remote Job execution is also referred to in the documentation as a **distant run**.

- The JobServer is a lightweight agent for running Jobs on remote hosts
- You can run as many JobServers as needed (within limits of available resources)

Talend JobServer

Requirements

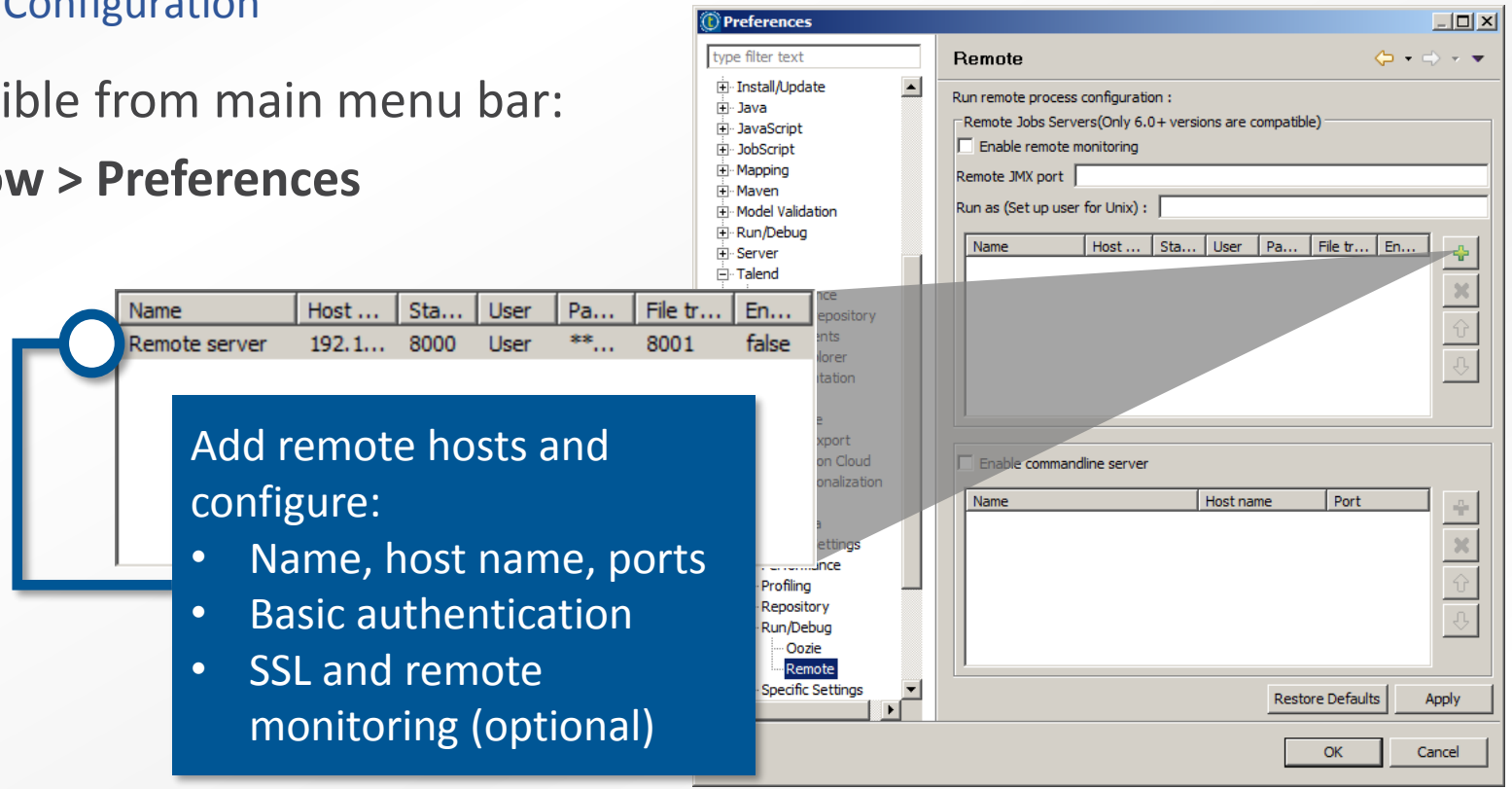
Remote Host	JobServer must run as a service on the remote host
Local Host	Configuration in Studio specifies communication details for the remote host on which the JobServer runs

Talend JobServer

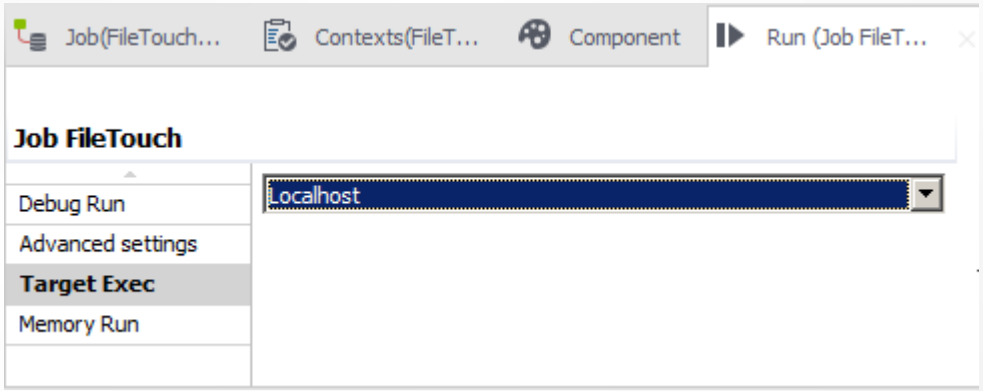
Local Host Configuration

- Accessible from main menu bar:

  Window > Preferences

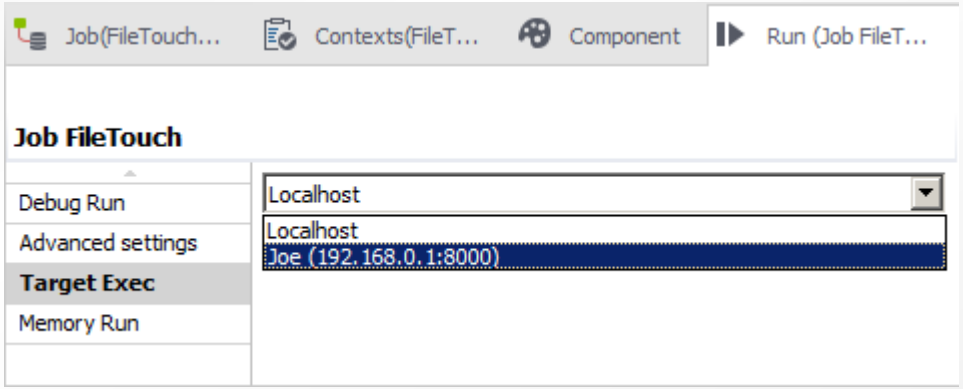


Remote Job Execution from Within Studio



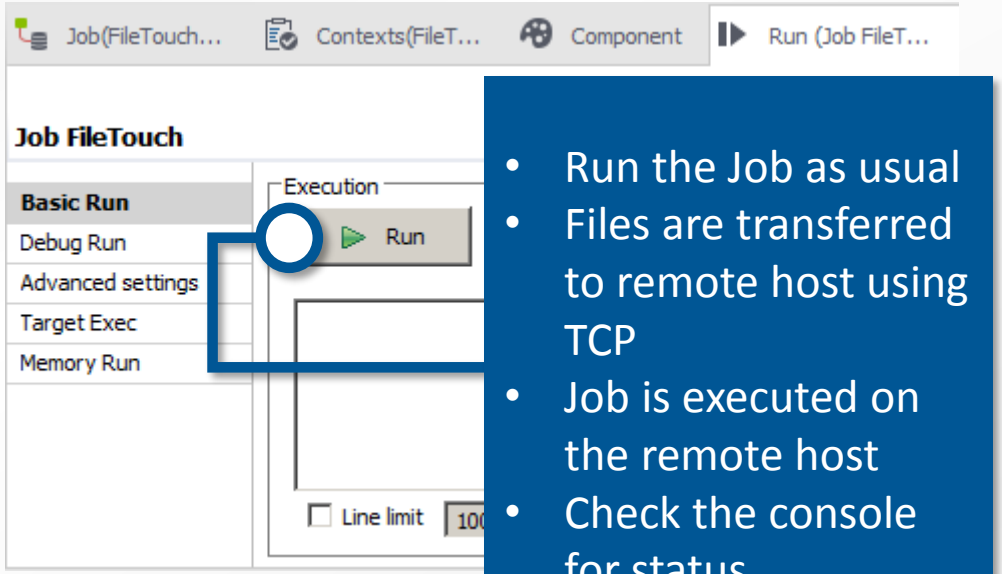
1. Select **Target Exec** tab in **Run** view
2. Specify host on which to run
3. Run the Job

Remote Job Execution from Within Studio



1. Select **Target Exec** tab in **Run** view
2. Specify host on which to run
3. Run the Job

Remote Job Execution from Within Studio



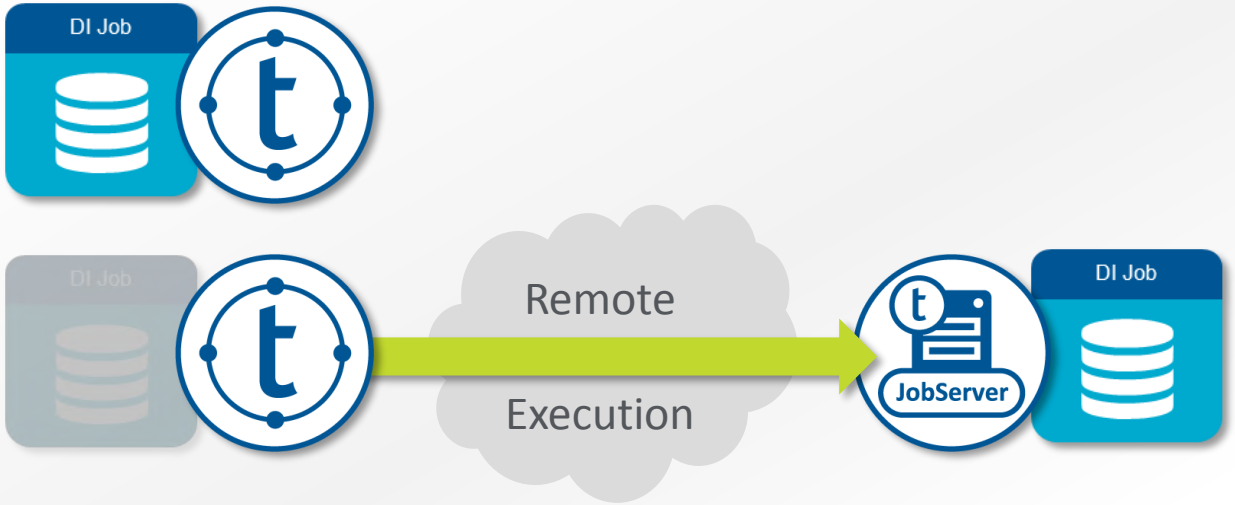
- Run the Job as usual
- Files are transferred to remote host using TCP
- Job is executed on the remote host
- Check the console for status

1. Select **Target Exec** tab in **Run** view
2. Specify host on which to run
3. Run the Job

Lab Overview

Remote Job Execution

- Configure Talend Studio and a remote host so a basic data integration Job can run:
  - Locally
  - On a remote host



Lesson Summary

- JobServer can run on multiple remote hosts to enable remote Job execution from Talend Studio
- Two requirements for Job to run on a remote host:
  - JobServer must run as a service on the remote host
  - Basic configuration in Studio (host, port, user, password)
- Change **Target Exec** in the **Run** view in order to run a Job on a remote host

Resource Usage and Basic Debugging

Objectives

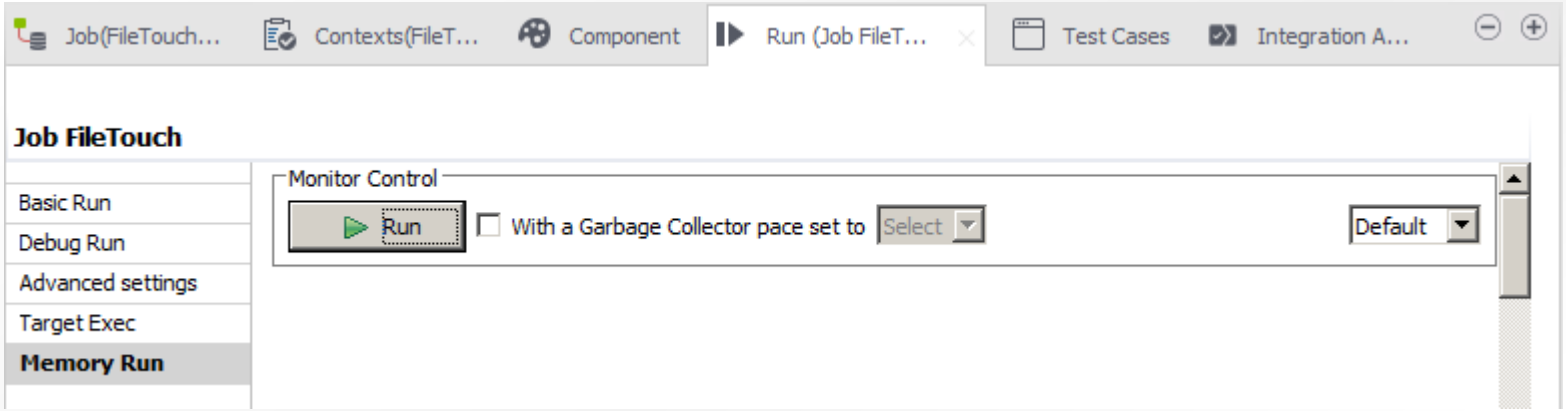
- Evaluate host CPU and real-time JVM memory usage during Job execution
- Use basic debugging features of Talend Studio
  - Turn on traces
  - Step through Job execution
  - Set breakpoints

Scenario

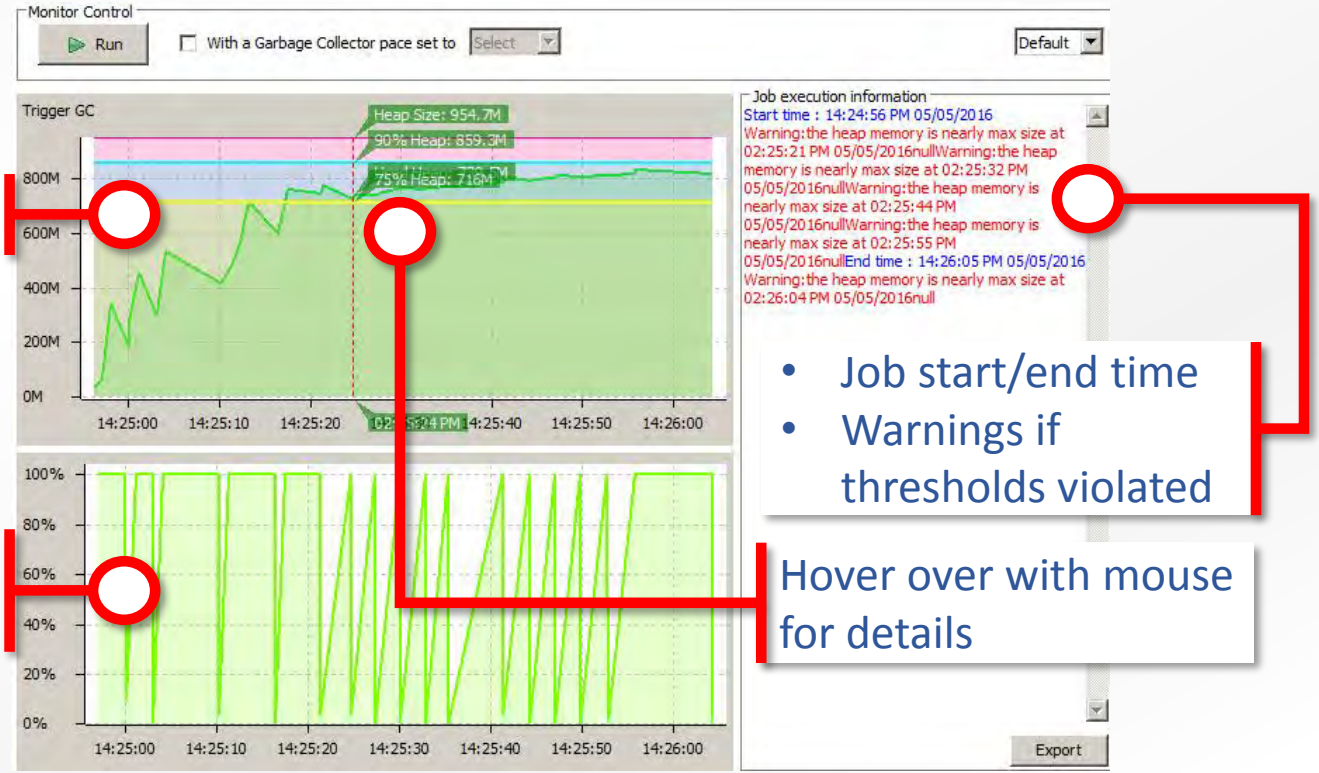
- You are concerned that the new Job you are developing is over-consuming compute resources and not performing well as a result.
  - You want to see host CPU and memory usage of the Job during runtime.
- You need additional functionality within the Studio to help with basic debugging efforts.

Memory Run

- **Memory run** is provided as a separate tab in in the **Run** view
- It reports on two key metrics:
  - **Memory:** shows Java Virtual Machine (JVM) memory usage as the Job runs
  - **CPU:** shows host CPU usage, also in real-time, as the Job runs



Memory Run



JVM memory

Host CPU Usage

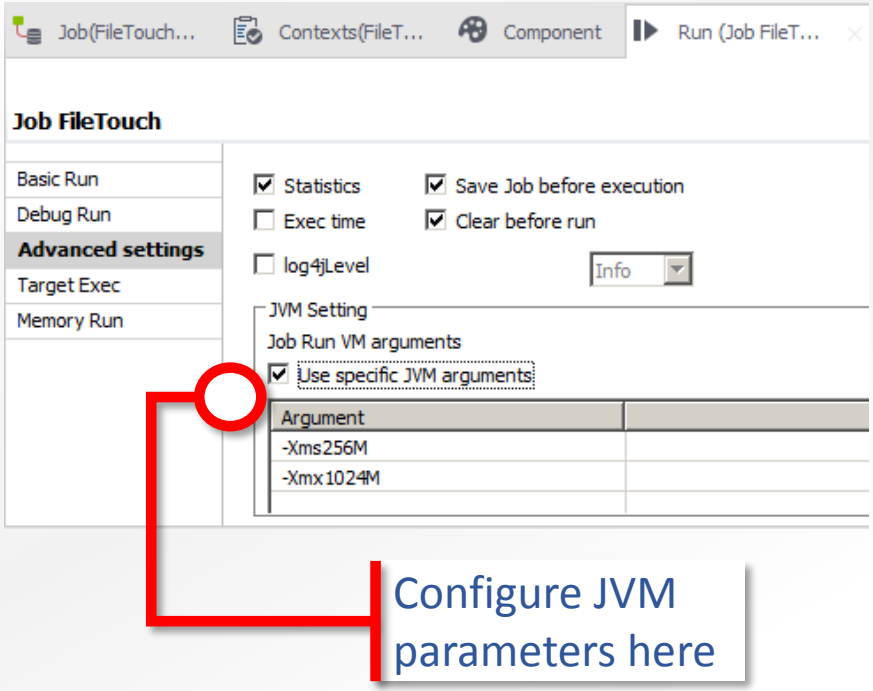
- Job start/end time
- Warnings if thresholds violated

Hover over with mouse for details

Memory Run

Resolving Issues with JVM Memory Settings

- If memory thresholds are violated, increasing the JVM memory settings might help. In particular:
  - **-Xms:** initial size of memory pool
  - **-Xmx:** maximum size of memory pool
- These parameters are passed to the JVM when it starts



Memory Run

Resolving Issues with Component Configuration Options

- Some components support options that adjust the amount of memory they consume. For example:
  - Some components honor a maximum memory usage parameter
  - Other parameters can swap working memory to disk, reducing overall memory footprint

Component	Configuration Setting
tSortRow	Sort on disk
tMap	Set max buffer size

Debug Run

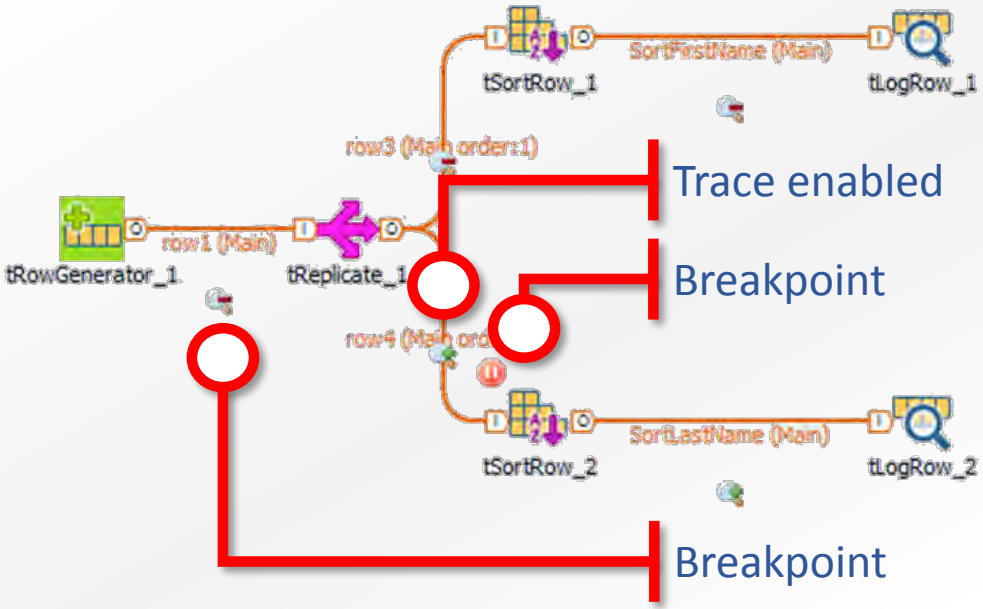
- Two debugging options are available in Talend Studio

Traces Debug	• Allows a row-by-row view of data through different flows in the Job • Does not require deep Java knowledge
Java Devug	• Based on Eclipse debugger • Advanced feature that is best suited for experienced Java developers

Debug Run

Traces Debug

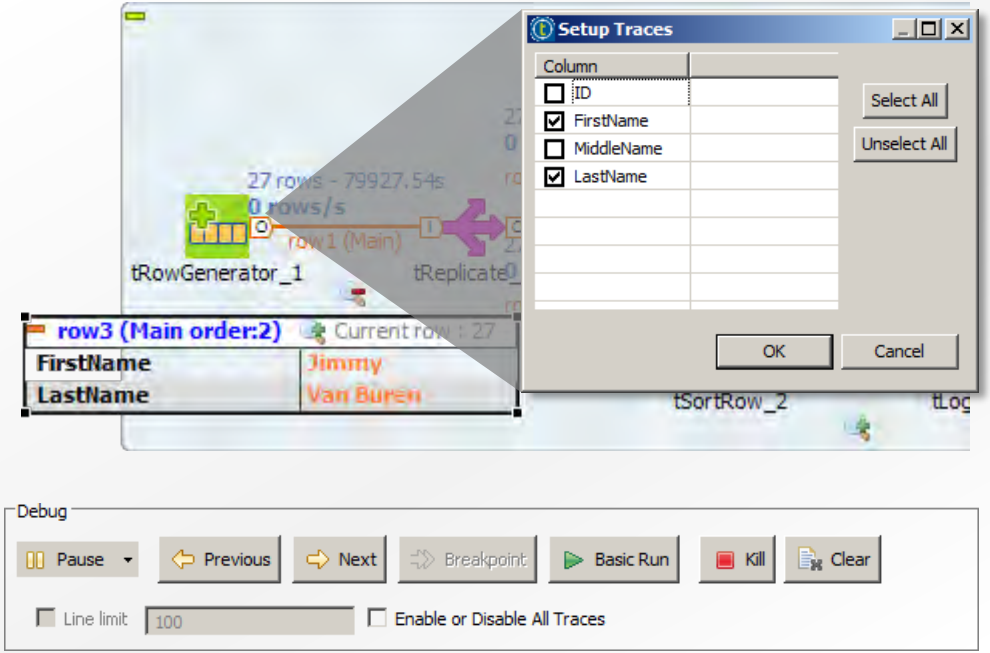
- Use **Set Traces** for a row-by-row view through the data flows
- Set **breakpoints** to pause execution when a condition is met



Debug Run

Traces Debug Configuration

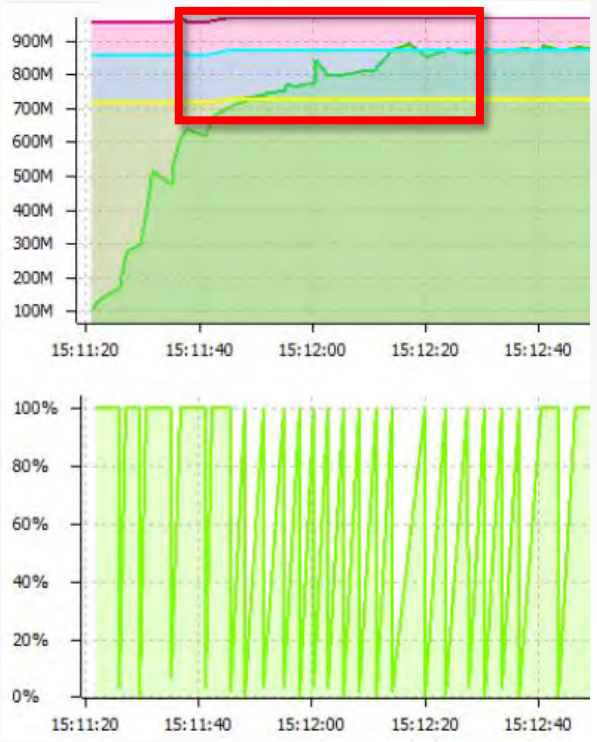
- Traces can be enabled or disabled for each data flow
- Use **Setup Traces** to configure columns to display
- Use controls to step forward or backward through Job execution



Lab Overview

Memory Run and Resource Usage

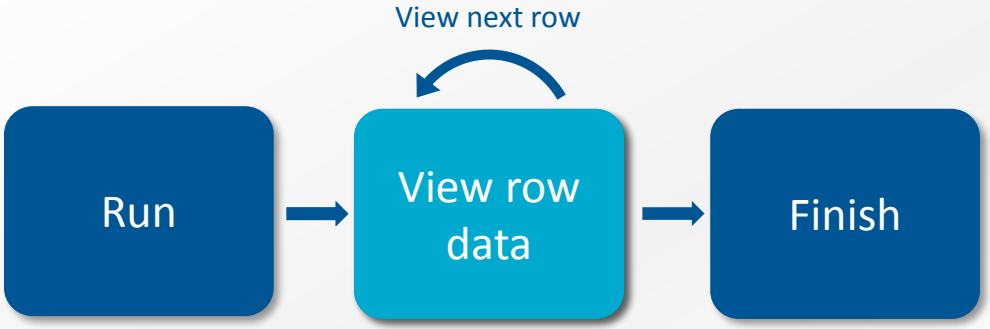
- Uses **Memory Run** to run a memory- and CPU-intensive Job while observing resource usage in real-time from Studio.
- Explore several possible ways to resolve the resource issue.



Lab Overview

Debug Run and Traces Debug

- Use **Traces Debug** to set up traces and breakpoints that let you step through the execution of your Job while viewing the processed data.



Lesson Summary

- Memory run:
  - Offers a real-time view of CPU and memory usage while Job is running
- Traces debug:
  - Enables examination of data as it flows through the Job
  - Simple-to-use VCR-style controls step you through Job execution
  - Breakpoints and conditional breakpoints can help you approach trouble spots and then step through data a row at a time

Activity Monitoring Console (AMC)

Objectives

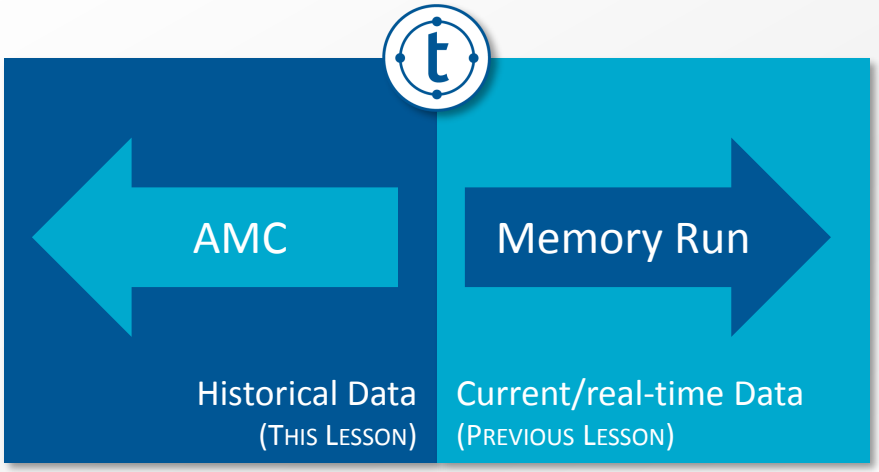
- Configure Talend Studio at the project, Job, and component level to monitor Jobs with the Activity Monitoring Console (AMC)
- Use the AMC to monitor Jobs from within Talend Studio

Scenario

- In addition to the real-time information you can obtain about your Jobs (using **Memory Run**), you need to see historical information within Studio
- Building a historical record of Job information establishes a baseline and highlights statistical trends
- The AMC perspective reveals historical statistics and log information about your Jobs

Activity Monitoring Console (AMC)

- Recall that Memory Run provides real-time data regarding Job resource consumption
- AMC complements this by providing historical insight into a Job's past performance

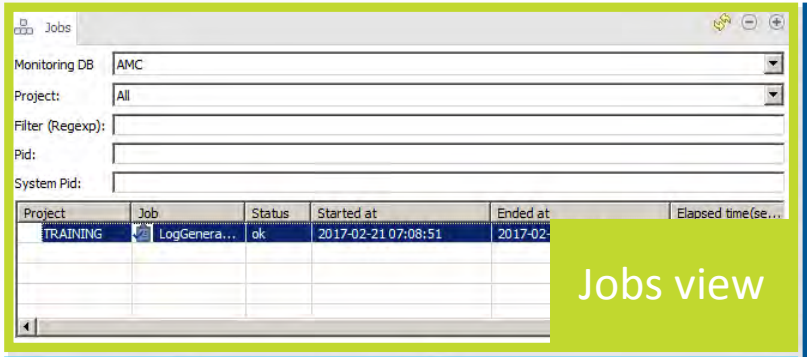


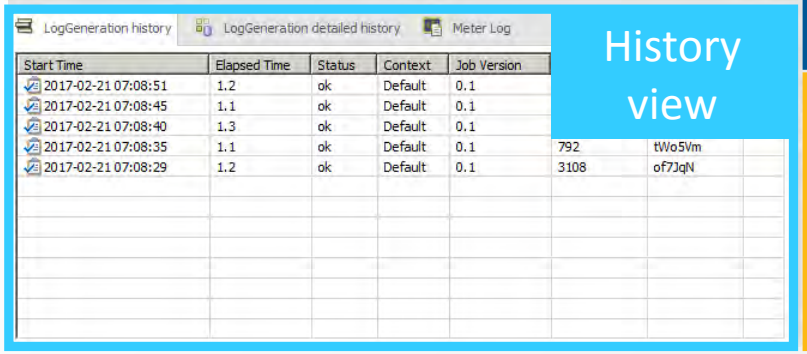
AMC Perspective

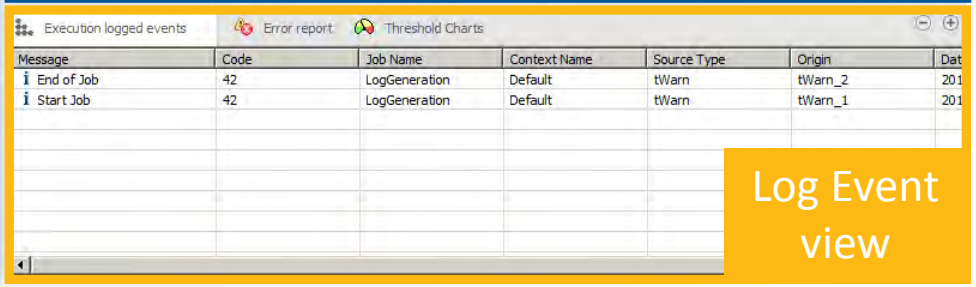
- The AMC perspective has many views. Common ones are summarized in the following table:

View	Description
Jobs	List of Jobs for which data has been collected
History	Two vantage points: • **Summary** • **Detail**
Main Chart	Graphical chart with Job execution times, component pie charts
Logged Events	Logged events are based on: • **tWarn** and **tDie** components • Java exceptions
Error Report	Errors over various time periods (hours, days, months)

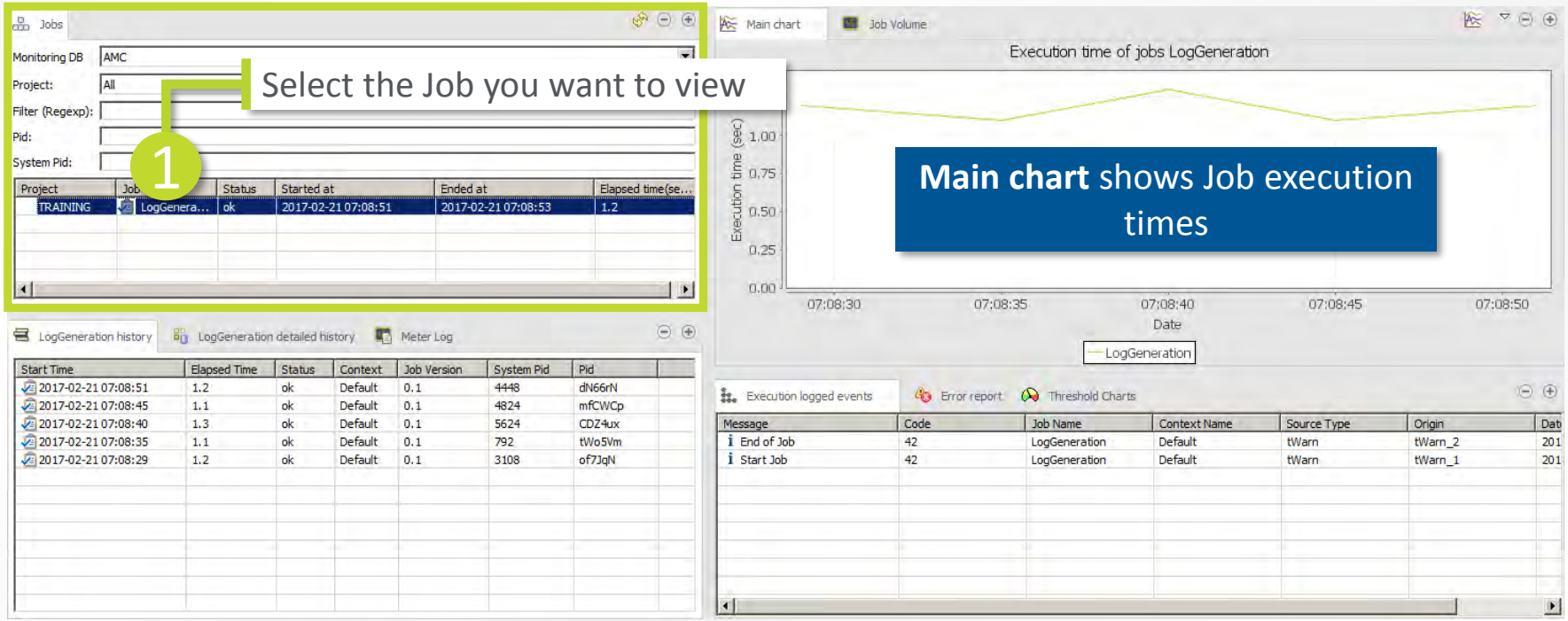
AMC Perspective



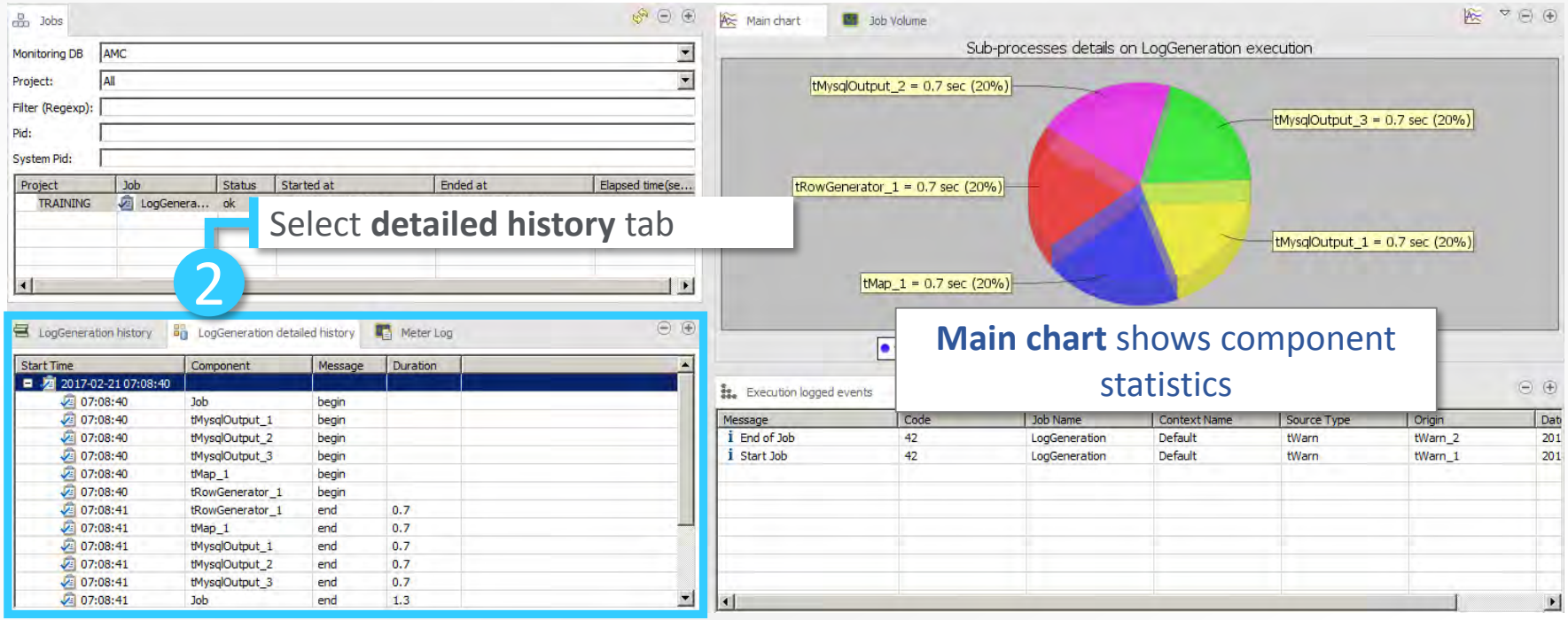




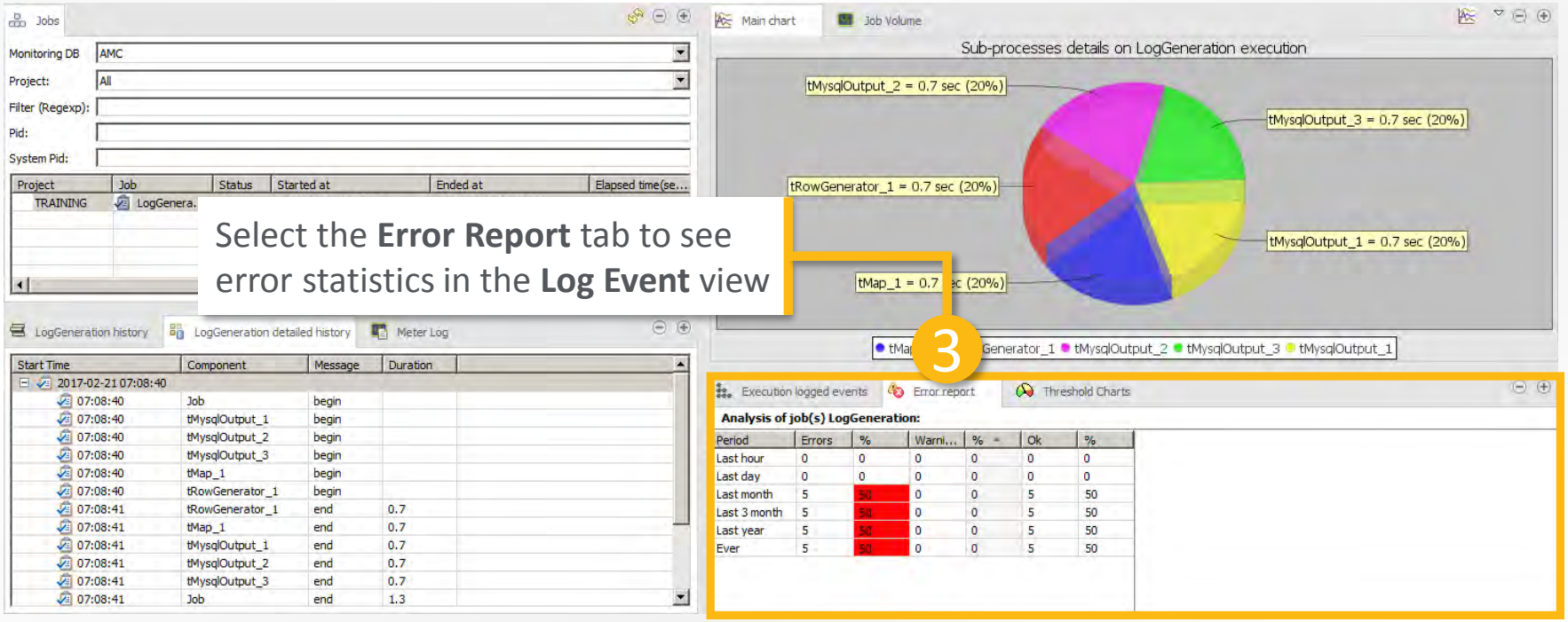
Basic AMC Usage



Basic AMC Usage



Basic AMC Usage



Configuration

Storage of Statistical Data

- By default, no historical statistics are saved for later viewing in the AMC
- Requires file or database configuration for:
  - Component statistics
  - Collection of logs
  - Data flow volumes

Configuration

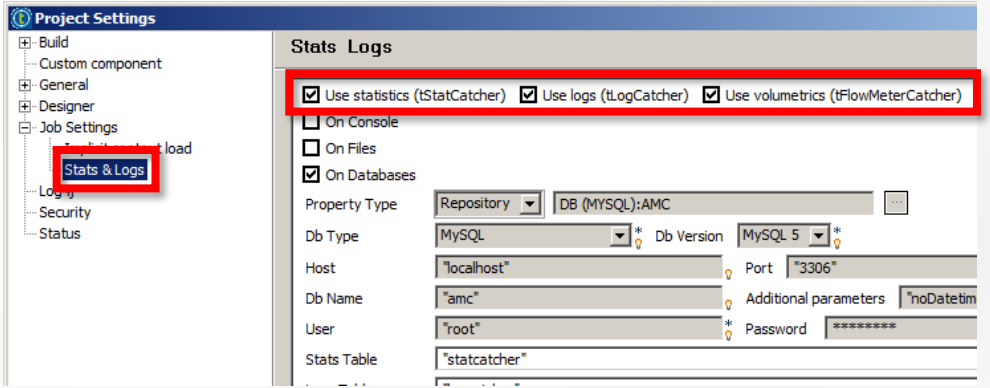
Enable Monitoring

- Monitoring must be enabled before Job statistics can be collected
- There are two different ways to enable:
  - **Stats & Logs** at the project or Job level – this is considered best practice
  - **Components** at the individual individual Job level:
    - tStatCatcher
    - tLogCatcher
    - tFlowMeterCatcher

Configuration

Project-level Configuration of Stats & Logs

- Select the type of data to collect:
  - Component statistics
  - Logs
  - Data flow volumes



1 Statistics to Use

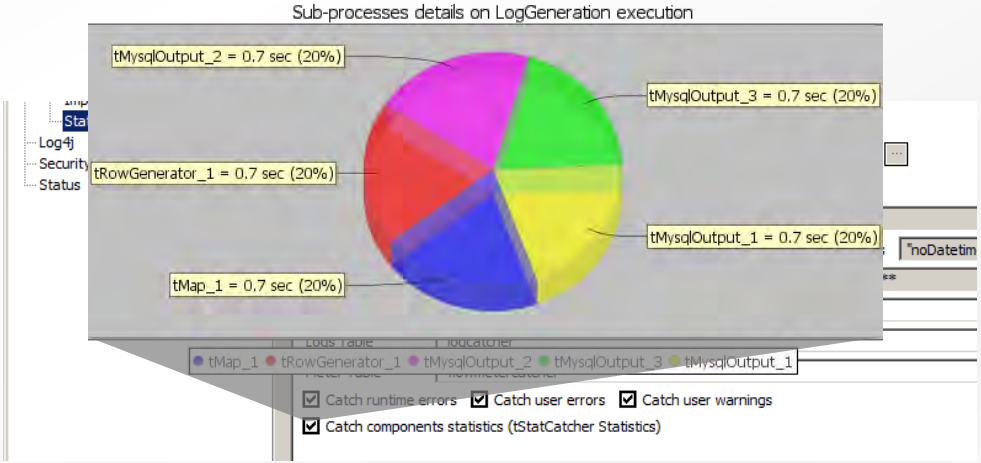
2 Components to Catch

3 Storage Information

Configuration

Project-level Configuration of Stats & Logs

- Select **Catch components statistics** to enable detailed statistics on a per-component basis

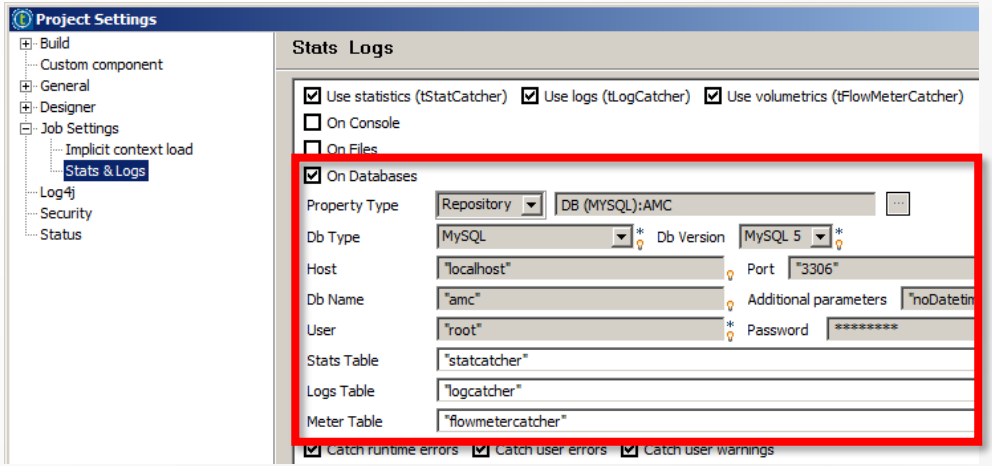


1. Statistics to Use
2. Components to Catch
3. Storage Information

Configuration

Project-level Configuration of Stats & Logs

- Specify where to record statistical data:
  - Database tables
  - Files



1 Statistics to Use

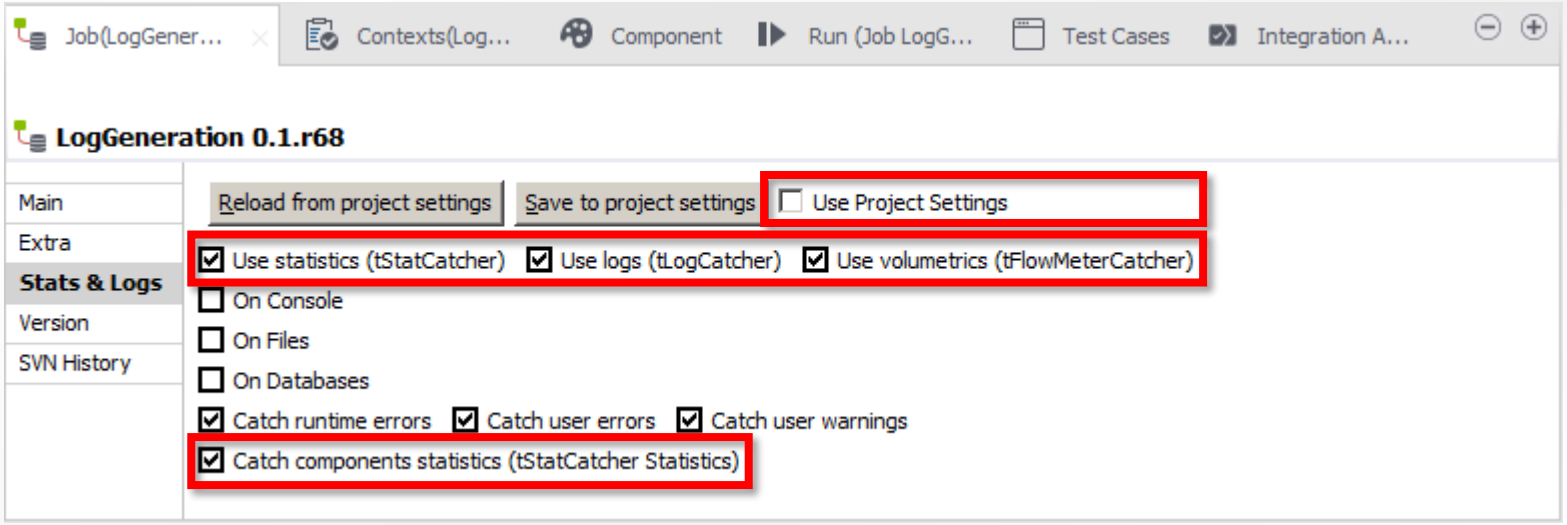
2 Components to Catch

3 Storage Information

Configuration

Job-level Configuration of Stats & Logs

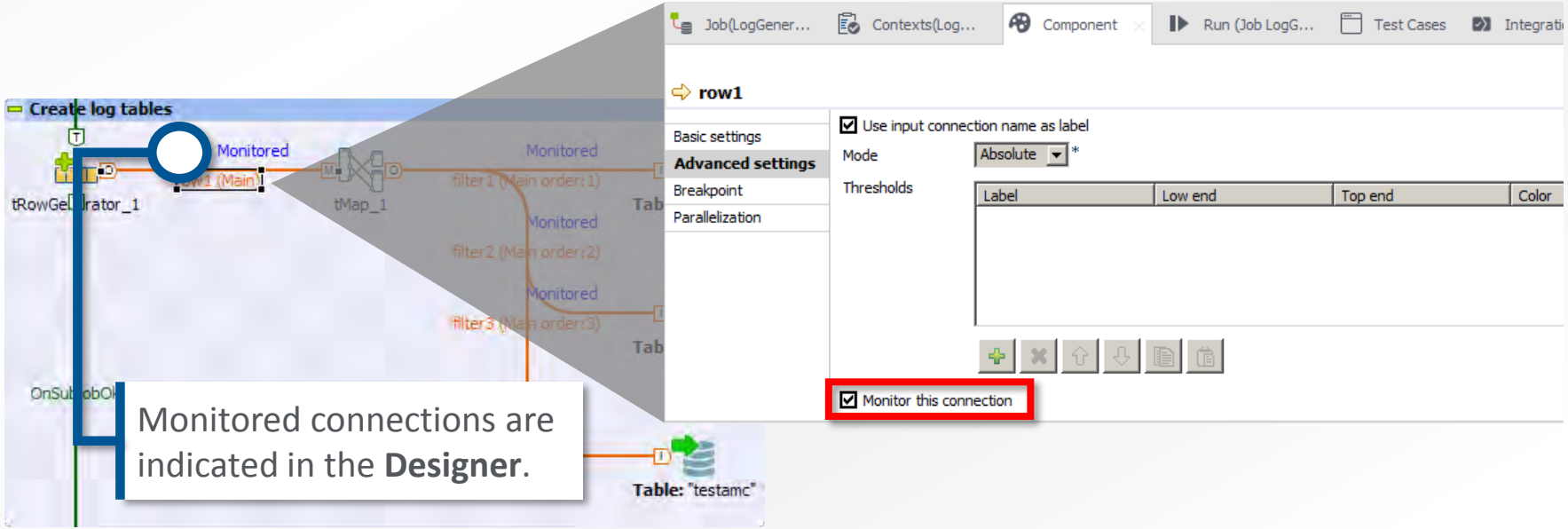
- The procedure for Job level configuration is nearly identical to the project-level configuration



Configuration

Monitoring Flows

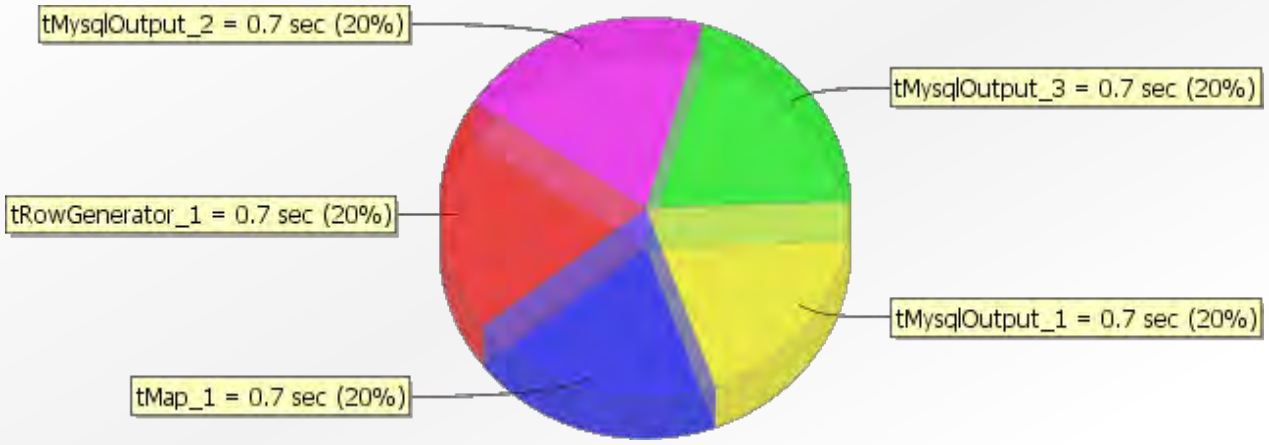
- The monitoring of data flows can be configured individually



Lab Overview

AMC

- Configure Studio so historical statistics and information can be captured for later viewing in the AMC
  - Configure the project and a Job
  - Run a test Job
  - View historical data from AMC



Lesson Summary

The AMC:

- Requires some basic configuration prior to monitoring Jobs
- Reads the collected Job execution data and displays it graphically
  - Provides a dedicated Perspective in Studio
- Can be used to supplement troubleshooting efforts after baselines are established and trends emerge

Parallel Execution

Objectives

- Implement several different methods of parallel processing in order to improve Job performance

Scenario

- One of your existing DI Jobs takes too long to complete.
  - With the continued growth in data set sizes, this has become a serious problem.
  - You are tasked with improving the performance of the Job.

Parallel Execution

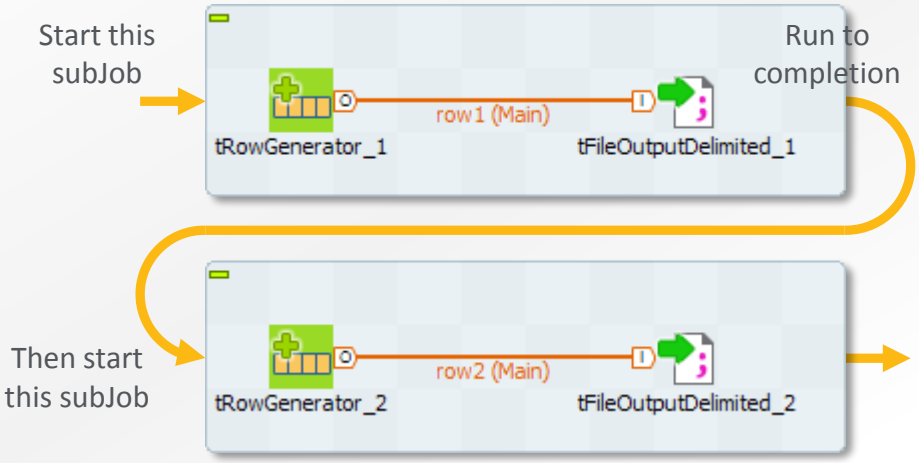
Parallelization Options Offered by Studio

- Options offered by Studio to increase Job performance through parallel processing include:
  - Enabling multithreading Job-wide
  - Using the **tParallelize** component to orchestrate subJobs
  - Leveraging parallel execution options available for specific components
  - Utilizing the automatic parallelization feature
  - Building dedicated parallelization components into your Job

Terminology

Sequential Processing

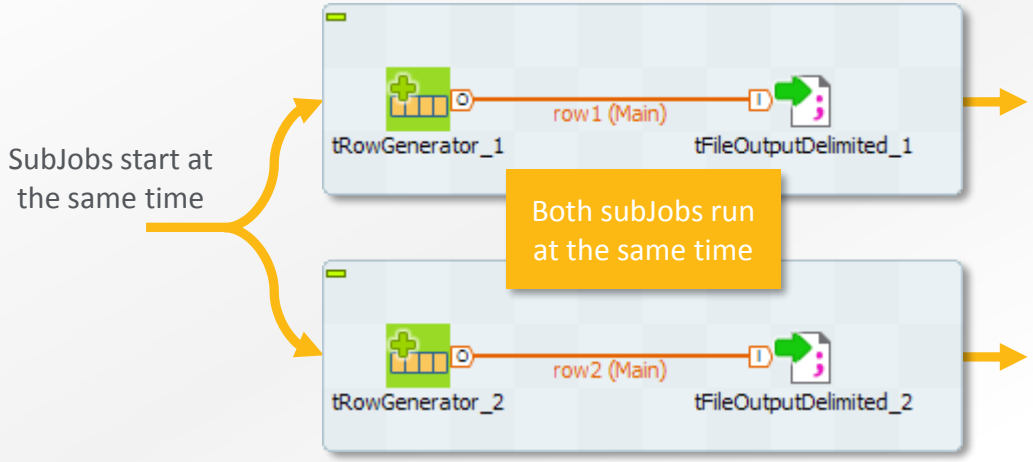
- When processing sequentially:
  - No subJob is run until the previous subJob is complete
  - Considered synchronous computing since no two subJobs are run at the same time



Terminology

Parallel Processing

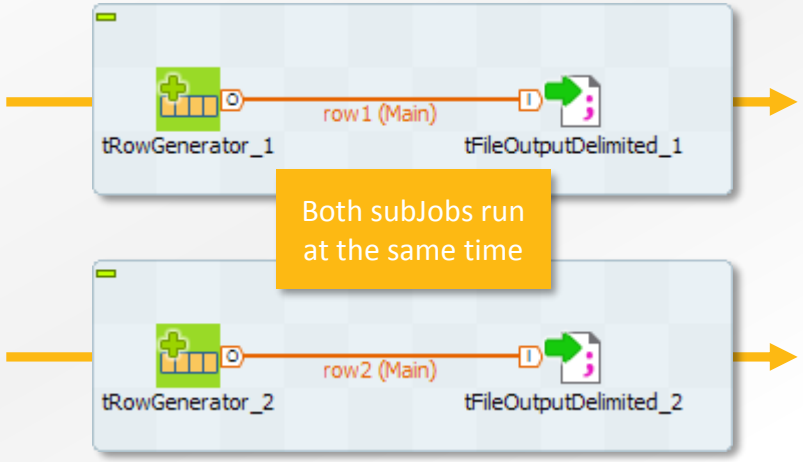
- When processing in parallel:
  - SubJobs run at the same time
  - Because both run at the same time, this is considered asynchronous computing



Terminology

Multithreading

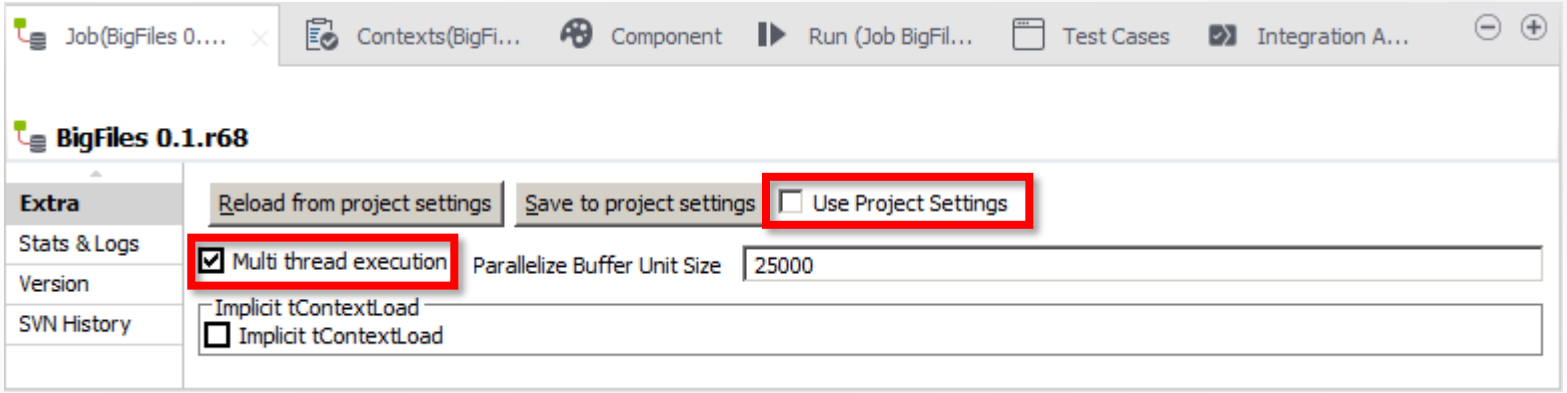
- In a multithreading scenario, all unconnected subJobs run at the same time
- Because both are running at the same time, this also is considered asynchronous computing



Parallelization Options

Multithreading

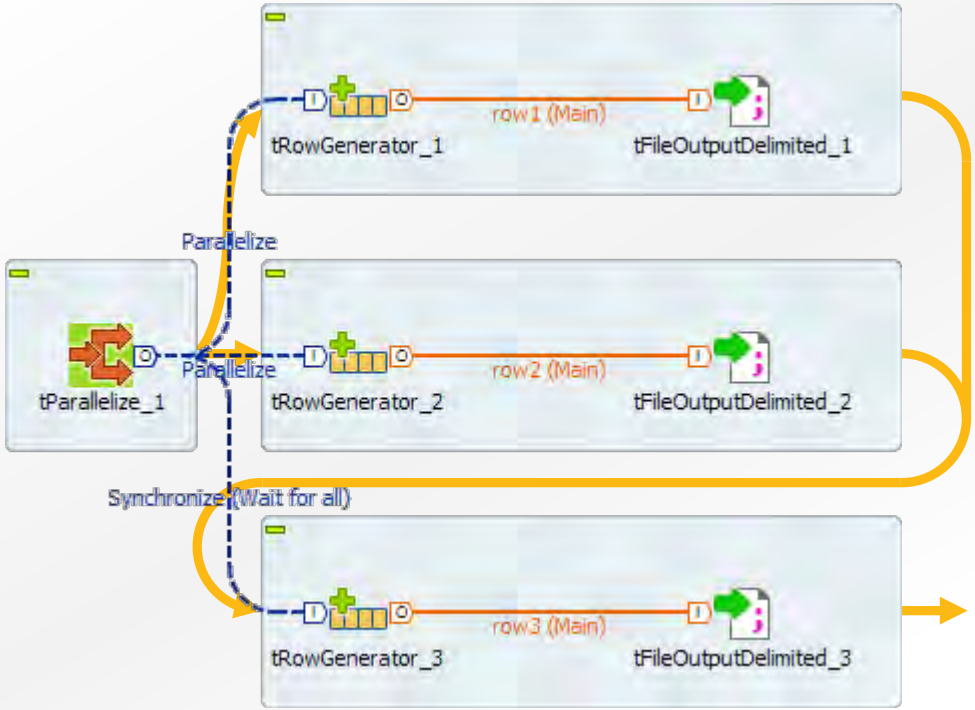
- By default, project settings are applied and need to be changed in the **Extra** tab of the **Job** view



Parallelization Options

Using the **tParallelize** Component

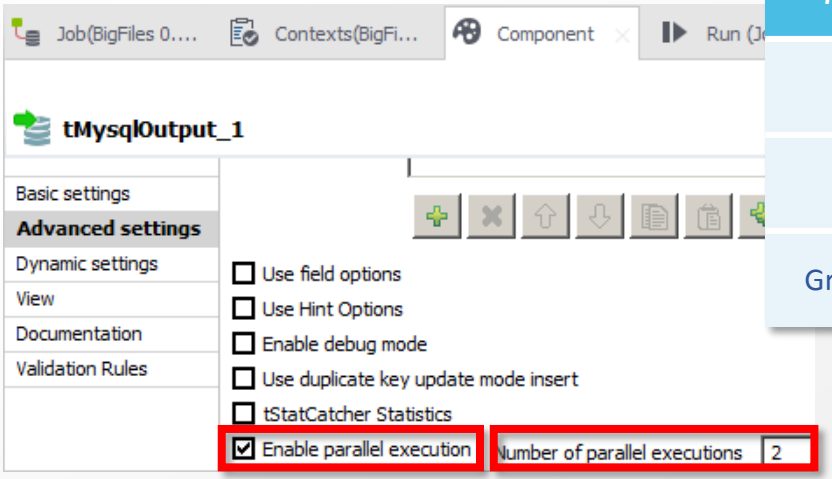
- The **tParallelize** component has triggers that allow connected subJobs to execute in parallel
  - Provides greater control as compared to multithreading the entire Job
  - Additional subJobs can run either in parallel or synchronously



Parallelization Options

Using Components that Support Parallelized Operation

- Many components (database input and output, for example) can enable parallelized operation

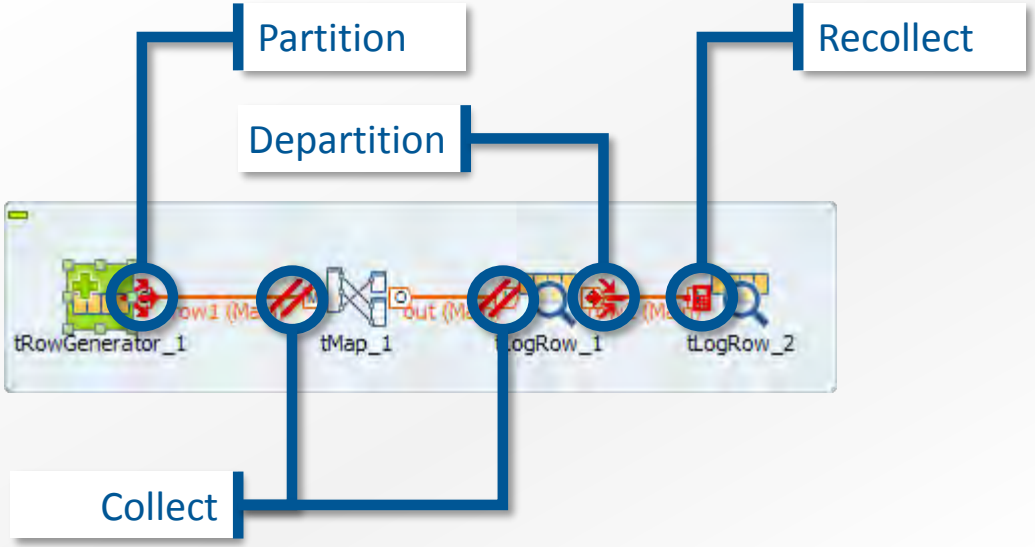


# of parallel executions	Comments
2-5	Often the best setting (default is 2).
# of available cores.	Sometimes optimal, but benchmark against other settings.
Greater than # available cores.	Typically results in degraded performance.

Parallelization Options

Automatic Parallelization

- In many situations, Talend Studio can automatic the parallelization of portions of a subJob, resulting in the following automatic operations:



Parallelization Options

Using Dedicated Components to Implement Parallelization

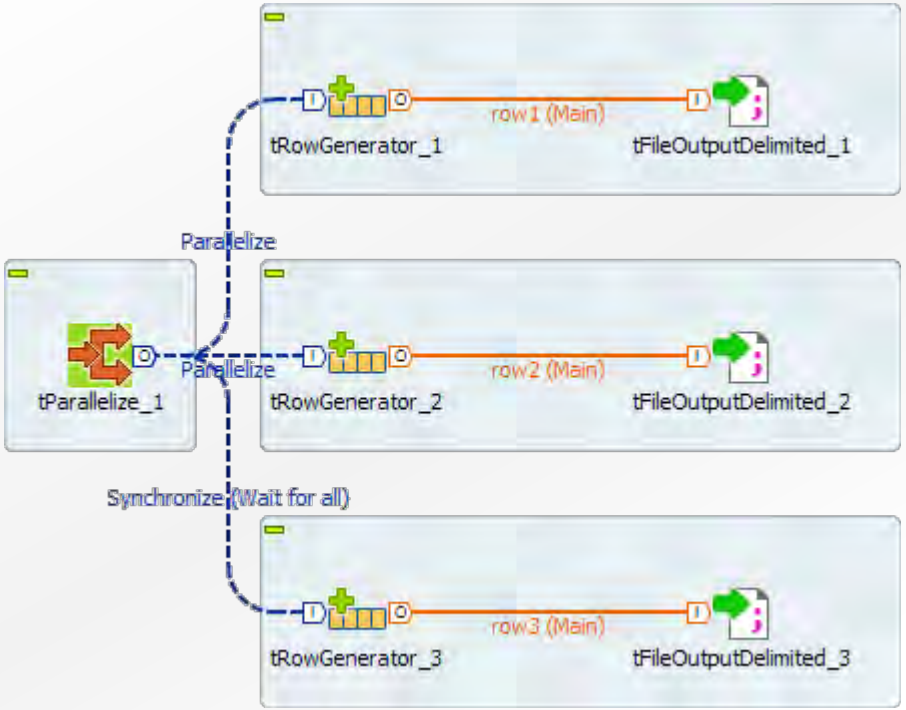
- Place and configure dedicated components to achieve parallelization
  - No longer recommended (this approach has been succeeded by the **Set Parallelization** feature)

Component	Purpose	Set Parallelization Equivalent
tPartitioner	Split data into multiple threads	
tCollector	Collect threads and send to next component.	
tDepartitioner	Group output of the threads.	
tRecollector	Capture results and output to next component.	

Lab Overview

Parallel Execution

- Build several Jobs that make use of the different parallel execution mechanisms available in Talend Studio
  - Observe the effect these have on execution flow and time



Lesson Summary

- Talend Studio supports many ways to implement parallel processing to make your Jobs run more efficiently

Joblets

Objectives

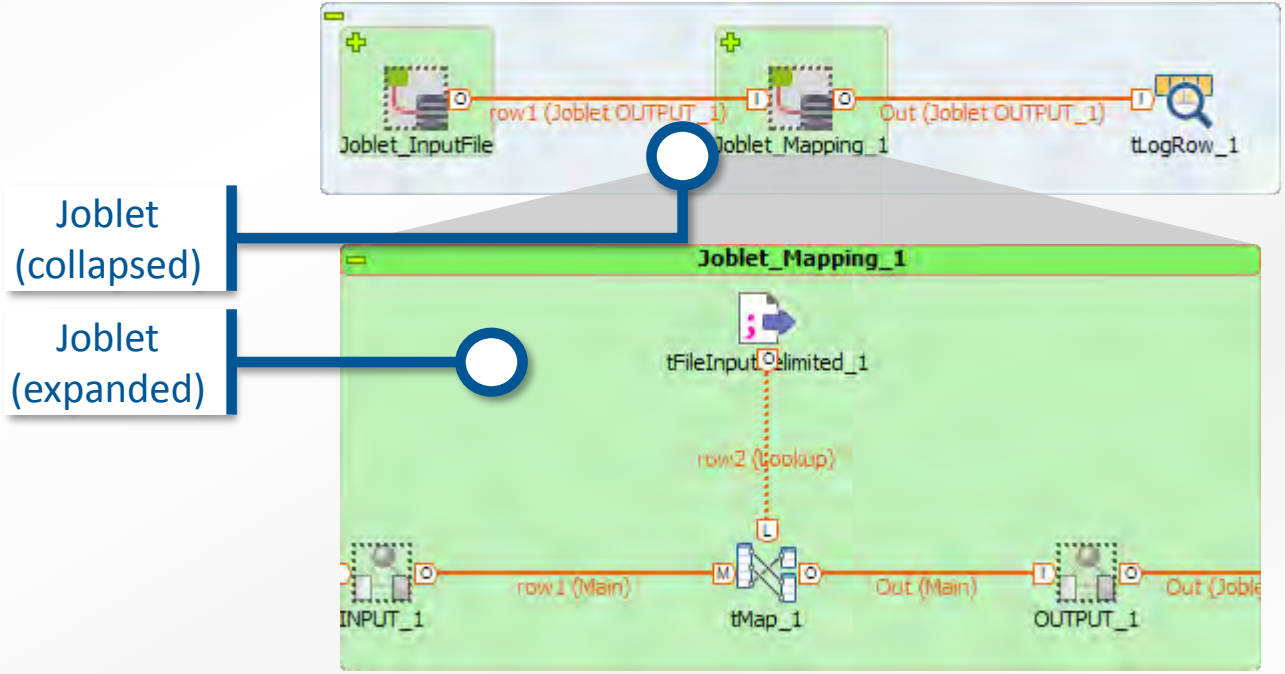
- Create a Joblet from an existing Job
- Create a Joblet from scratch
- Use a Joblet within a Job
- Use a Joblet trigger

Scenario

- You often need to reuse a portion of a Job in other Jobs
  - Repeatedly reconstructing that part of the Job wastes cycles and could introduce errors
- Other developers find a portion of a Job too complex, so abstracting it out and showing less detail makes the Job less confusing and more maintainable

What is a Joblet?

- A component that replaces a group of components

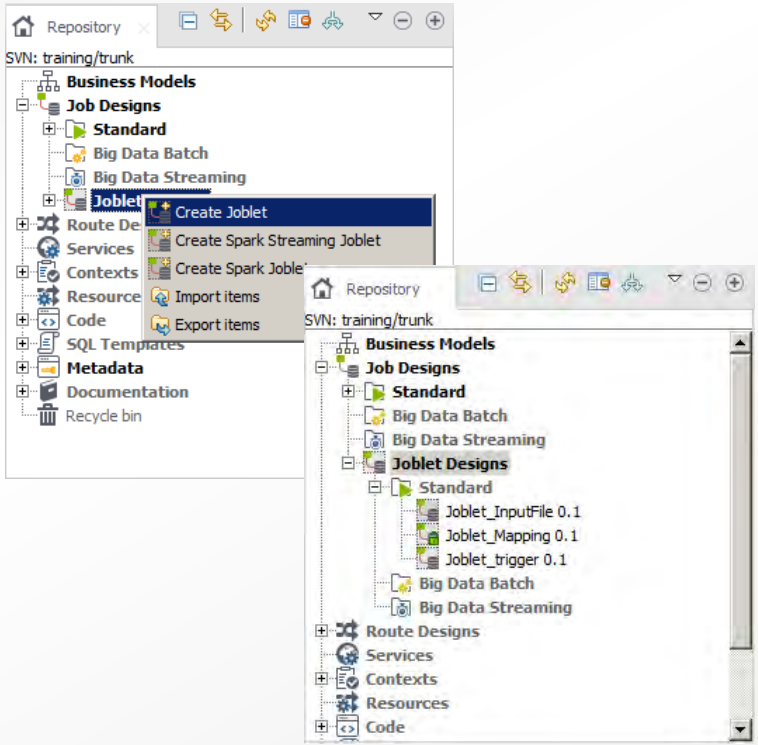


Why Use a Joblet?

- Joblets simplify the Job
- They allow for reuse
  - In other Jobs
  - In the same Job
- They are easy to monitor
  - If you monitor the Job, the Joblet is monitored, too
  - No need to use and configure log components (**tLogCatcher**, **tStatcatcher**, **tFlowCatcher**)
- They have no impact on performance
  - The Joblet code is integrated into the Job itself

How to Create a Joblet?

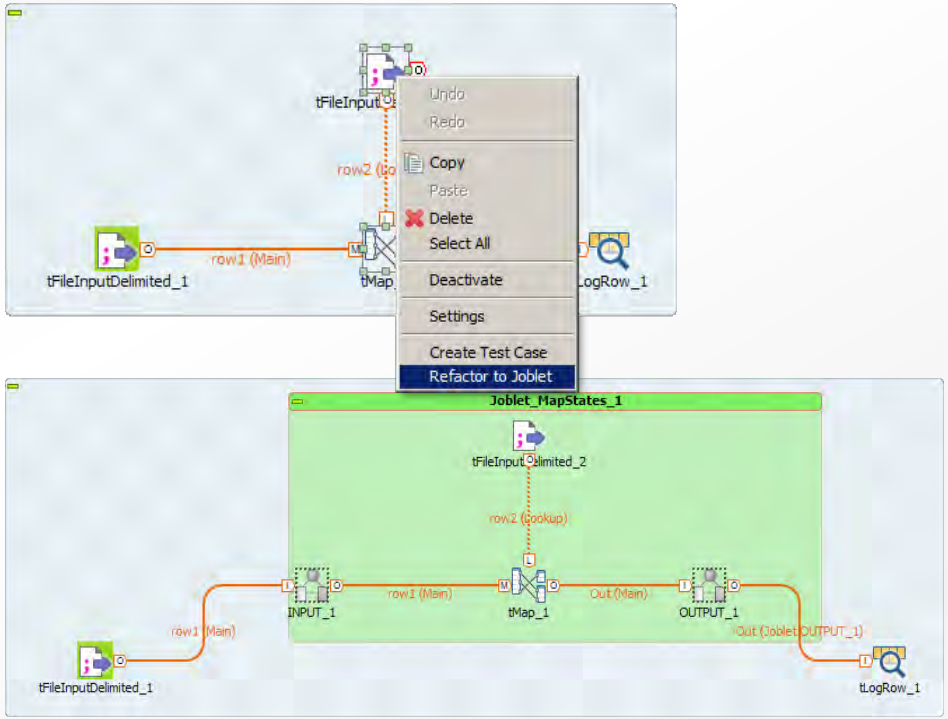
From Scratch



1. Right-click in **Repository**.
2. Fill out name and other fields.
3. Joblets appear in the **Repository** underneath **Job Designs**.
4. Add components the same way as with a standard Job.

How to Create a Joblet?

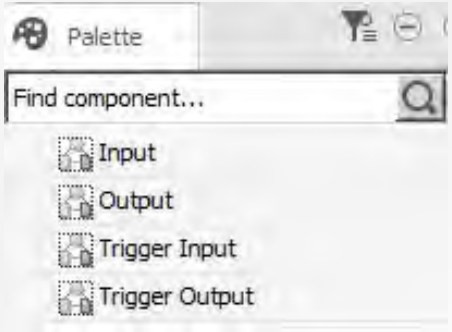
From an Existing Job



1. Select components.
2. Right-click.
3. Joblet is created with generic input and output components as the main interface.

Working with Joblets

- Four Joblet-specific components are available in the palette:
  - Generic **Input** and **Output** components direct data into and out of the Joblet
  - A **Trigger Input** can be used to start the Joblet
  - A **Trigger Output** can be used to start a subJob after a Joblet completes



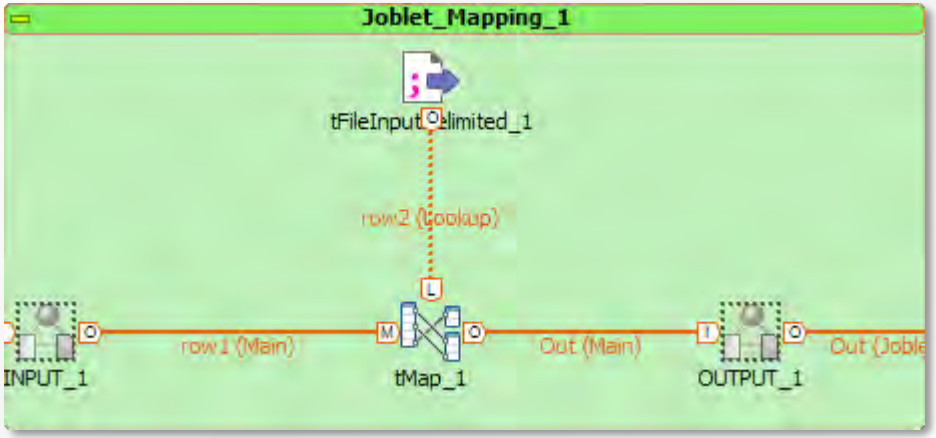
A Comparison of Joblet and tRunJob

Joblet	tRunJob
Design-time factoring	Run-time factoring
Similar to a macro that is pulled in at compile time	Calls specified Job at run-time
Use Joblets if you have a common design to share	Use tRunJob to trigger or pass data across Jobs

Lab Overview

Joblets

- Create a Joblet from an existing Job. Confirm that when it runs, it does the exact same thing as the original Job.
- Create a Joblet from scratch.
- Explore the triggering options that Joblets provide.



Lesson Summary

Joblets:

- Help simplify complex portions of a Job
- Are reusable
  - In other Jobs
  - In the same Job
- Do not impact performance
- Are placed like other components
  - Drag and drop from the palette
  - Browse or search in the palette
  - Type directly into the workspace
- Can use triggers similar to the way subJobs do

Unit Test

Objectives

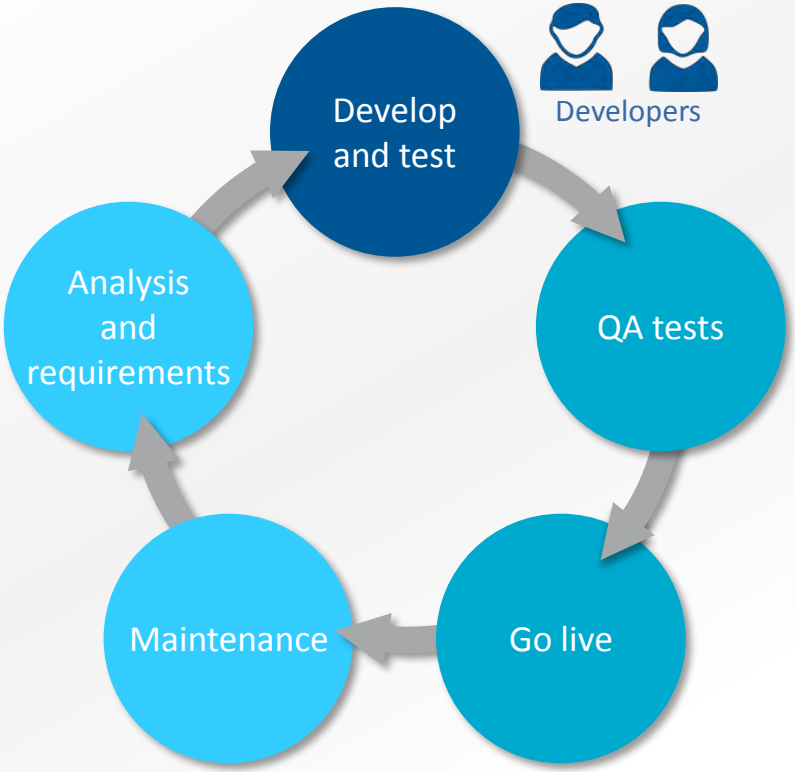
- Create a unit test case
- Understand what Talend Studio does to automate the process
- Configure a unit test case
  - Avoid common configuration pitfalls
- Run the unit test
  - From the **Test Cases** view
  - From the standard **Run** view

Scenario

- As a developer, you are tasked with developing your own unit tests
- You learn that you can use the Studio to automate the development of unit tests

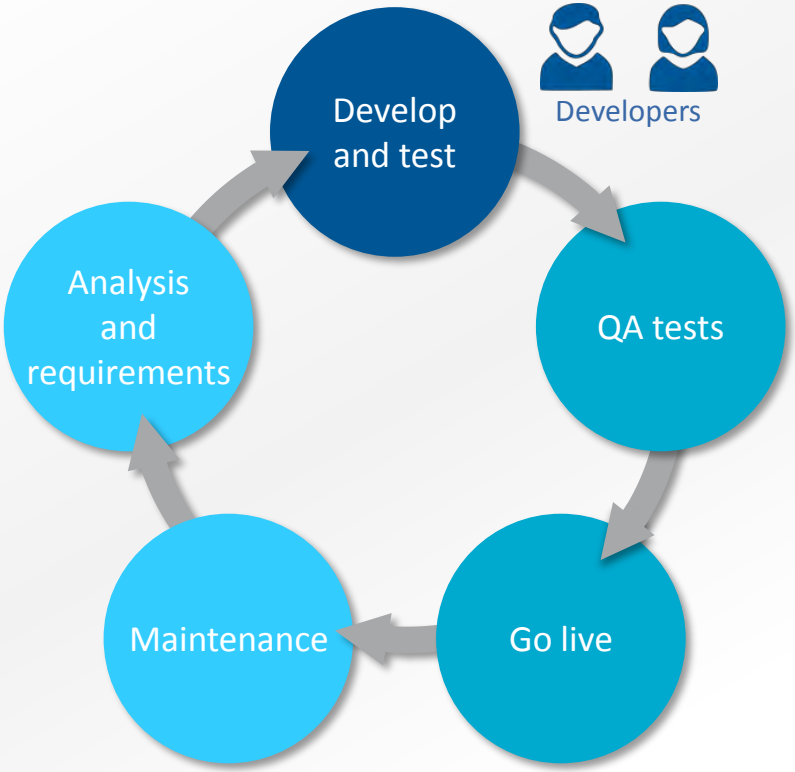
Software Development Lifecycle (SDLC)

- The **software development lifecycle** (**SDLC**) is the process of designing, developing and testing software in order to ensure its quality, efficiency, and sustainability



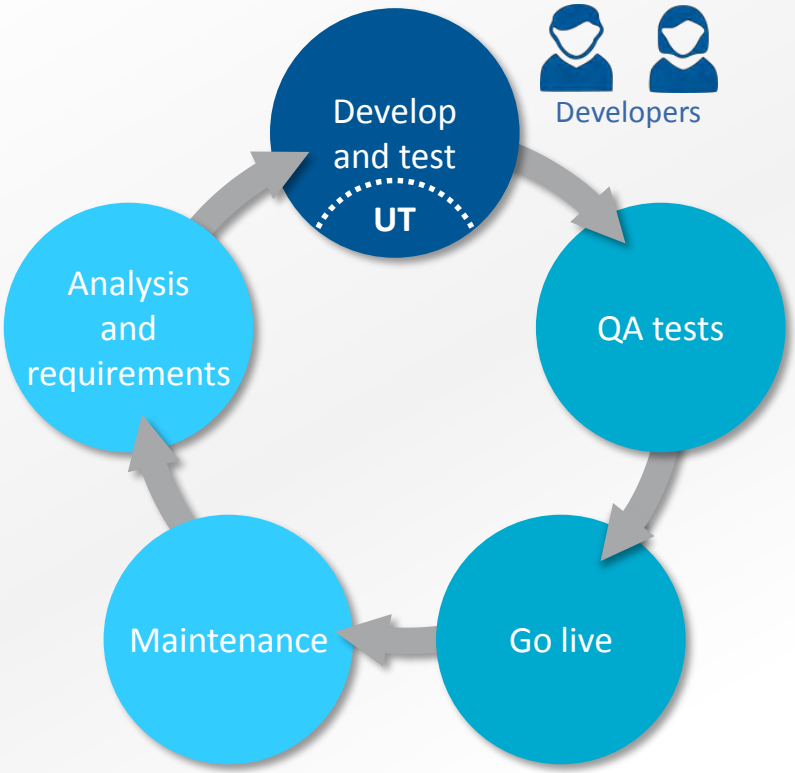
Continuous Integration

- **Continuous integration** (**CI**) refers to the process of merging all developer code into a shared mainline, often several times a day



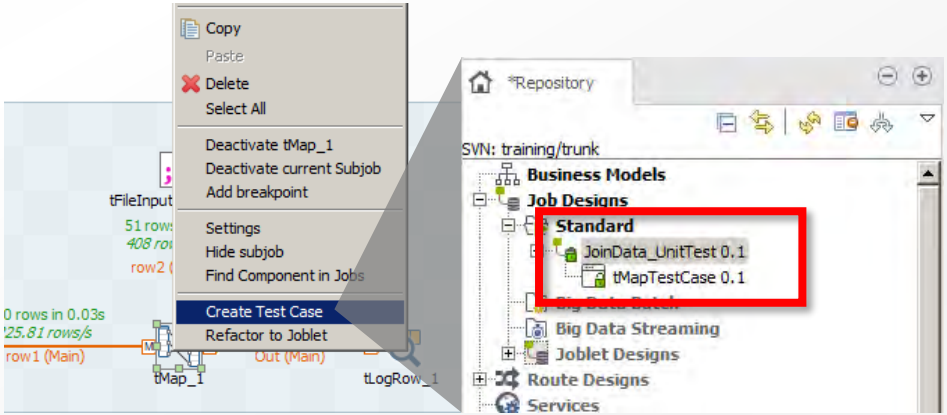
Unit Test

- Unit testing refers to the process of testing small portions of code.
  - This is considered part of the continuous integration process or software development lifecycle
  - Usually created by developers



Create a Unit Test in Talend Studio

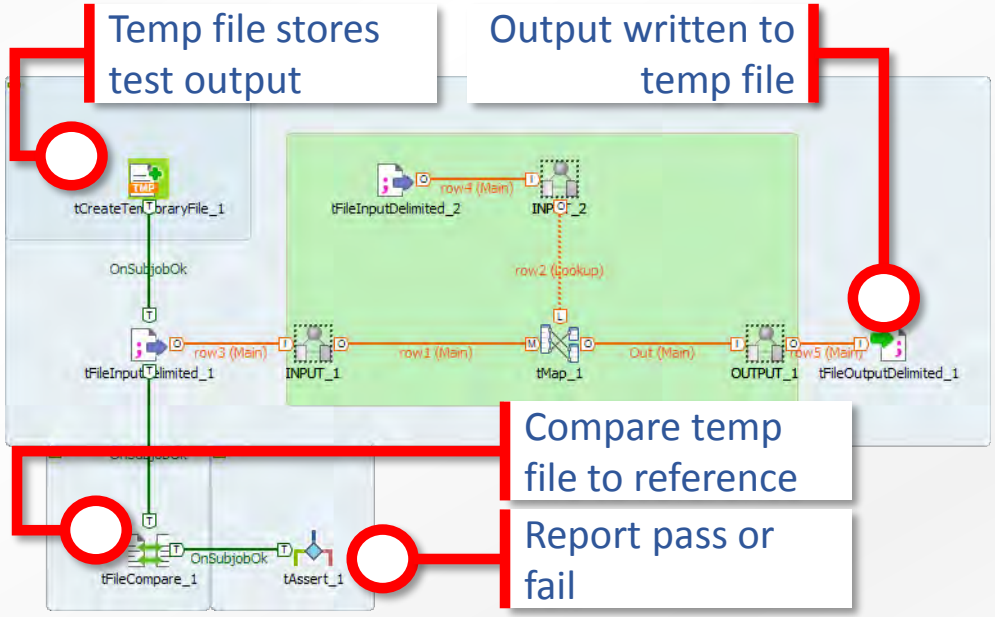
- Right-click the component for which you want to build a unit test
- The unit test is created under the Job in the **Repository**



1. Create Test Case
2. Configure
3. Run

Create a Unit Test in Talend Studio

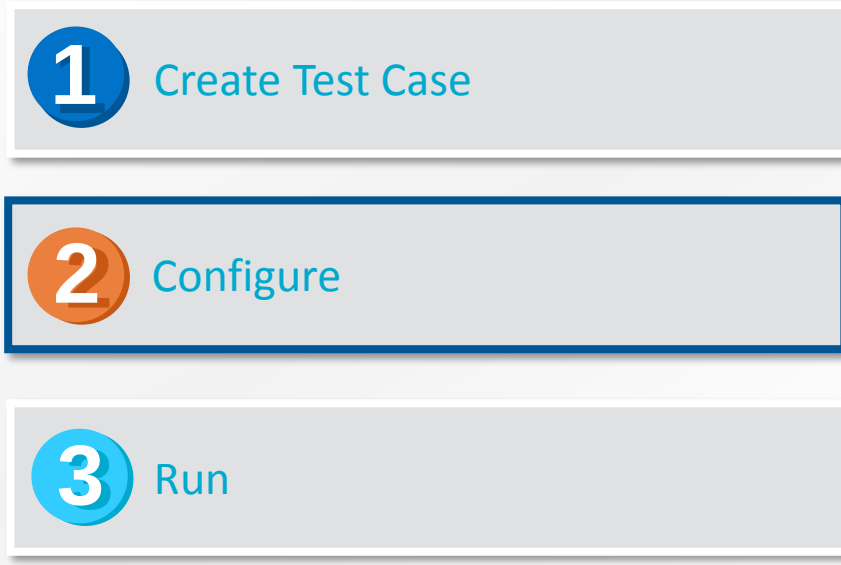
- Unit test component is wrapped with several other components



1. Create Test Case
2. Configure
3. Run

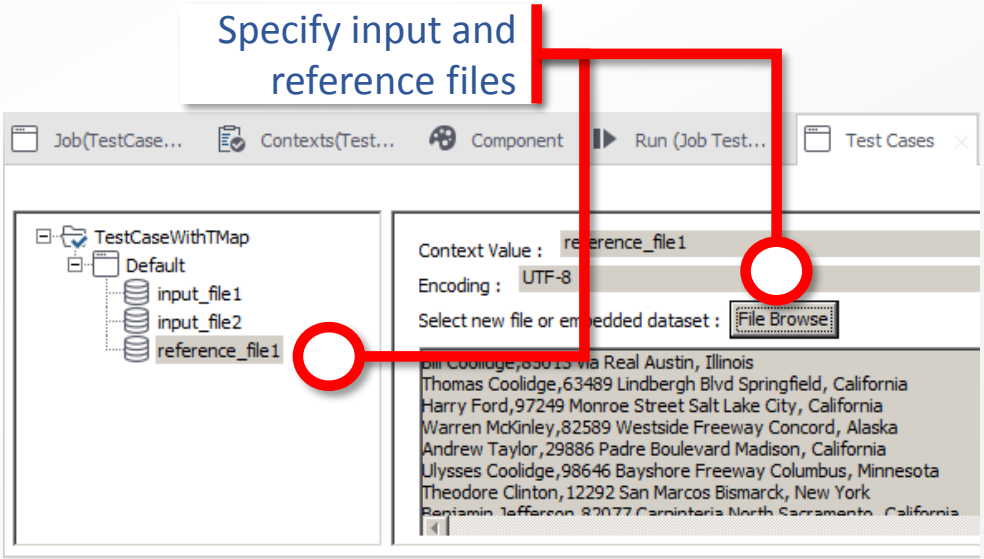
Create a Unit Test in Talend Studio

- Configure the **Input** and **Output** components
- Ensure that common settings align with the baseline output file against which you are comparing:
  - Row and field separators
  - Header values



Create a Unit Test in Talend Studio

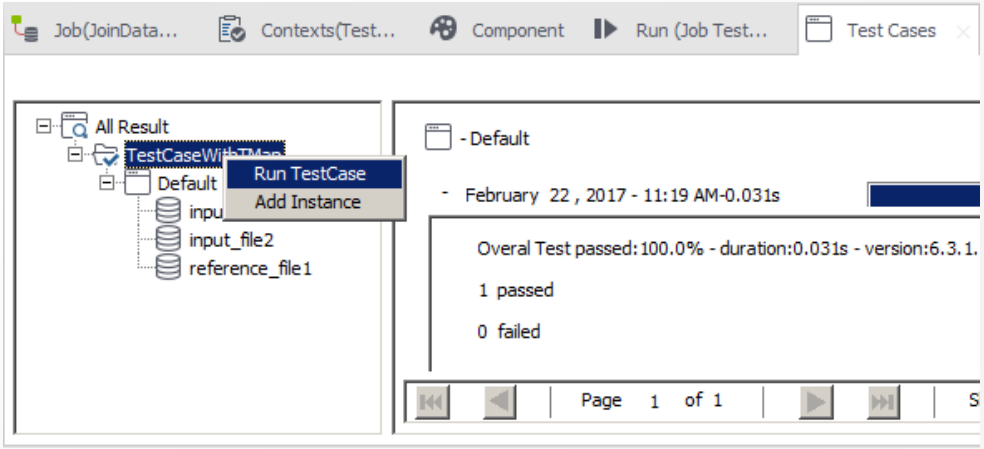
- Specify all input files and reference file in the **Test Cases** view



1. Create Test Case
2. Configure
3. Run

Create a Unit Test in Talend Studio

- Run test case and view results



1 Create Test Case

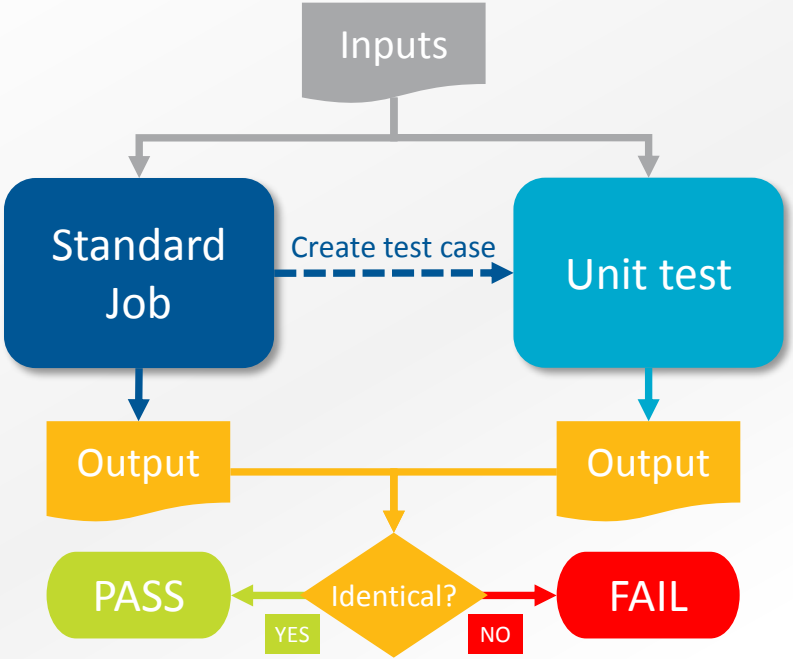
2 Configure

3 Run

Lab Overview

Unit Test

- Create a unit test around a key **tMap** component in a standard Job
- Run the unit test, which compares output from the original Job and unit test.
  - Confirm that the output is identical



Lesson Summary

- Unit tests are an important part of the development process
- The **Create Test Case** feature automates the build of a unit test in Studio
- Configuring inputs and outputs of a test is critical
- Common configuration mistakes include:
  - Incorrect row or field separators
  - Incorrect number of header rows
- You can run a unit test from the **Test Cases** or **Run** view, with appropriate configuration

Change Data Capture

Objectives

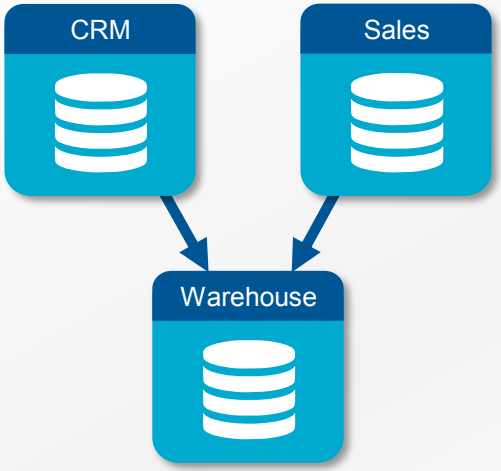
- Keep a master database in sync with changes made in transactional tables
- Configure a table to be monitored for changes
- Create a Job that uses the change information to update the master database

Scenario

- Many situations require that you keep two or more databases synchronized with each other
- Keep a data warehouse (DWH) in sync with production databases at regional offices by performing incremental updates based on changes in the Change Data Capture (CDC) table

What is Change Data Capture (CDC)?

- Monitor and track changes to tables so they can be applied to other tables in order to maintain synchronization



When is CDC Used?

- When you have so much data that it is no longer practical to dump and load the entire database
- Key benefits include saving:
  - Time
  - Compute resources
  - Memory
  - Network bandwidth

Sample Use Case

Employees

ID	First Name	Last Name	Age
1	Jane	~~Doe~~ Smith	~~24~~ 25
2	Jesse	Boardman	29
3	Julie	Yang	33

CDC

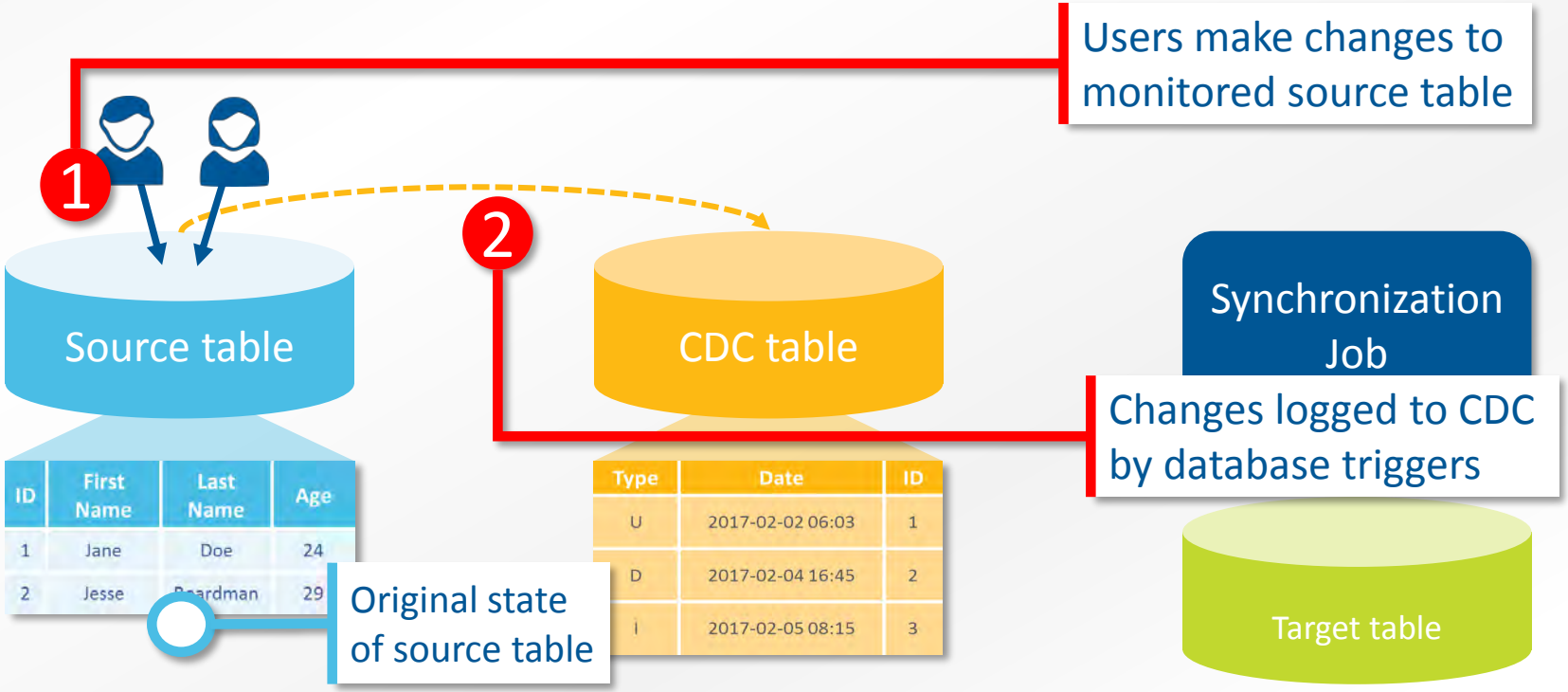
Type	Date	ID
U	2017-02-02 06:03	1
D	2017-02-04 16:45	2
I	2017-02-05 08:15	3

- HR application makes changes to **Employees** table
- Using triggers, the changes are monitored and captured in the **CDC** table
- Periodically, a Job processes the **CDC** table and commits changes to the **DWH**

Synchronize Warehouse (DWH)

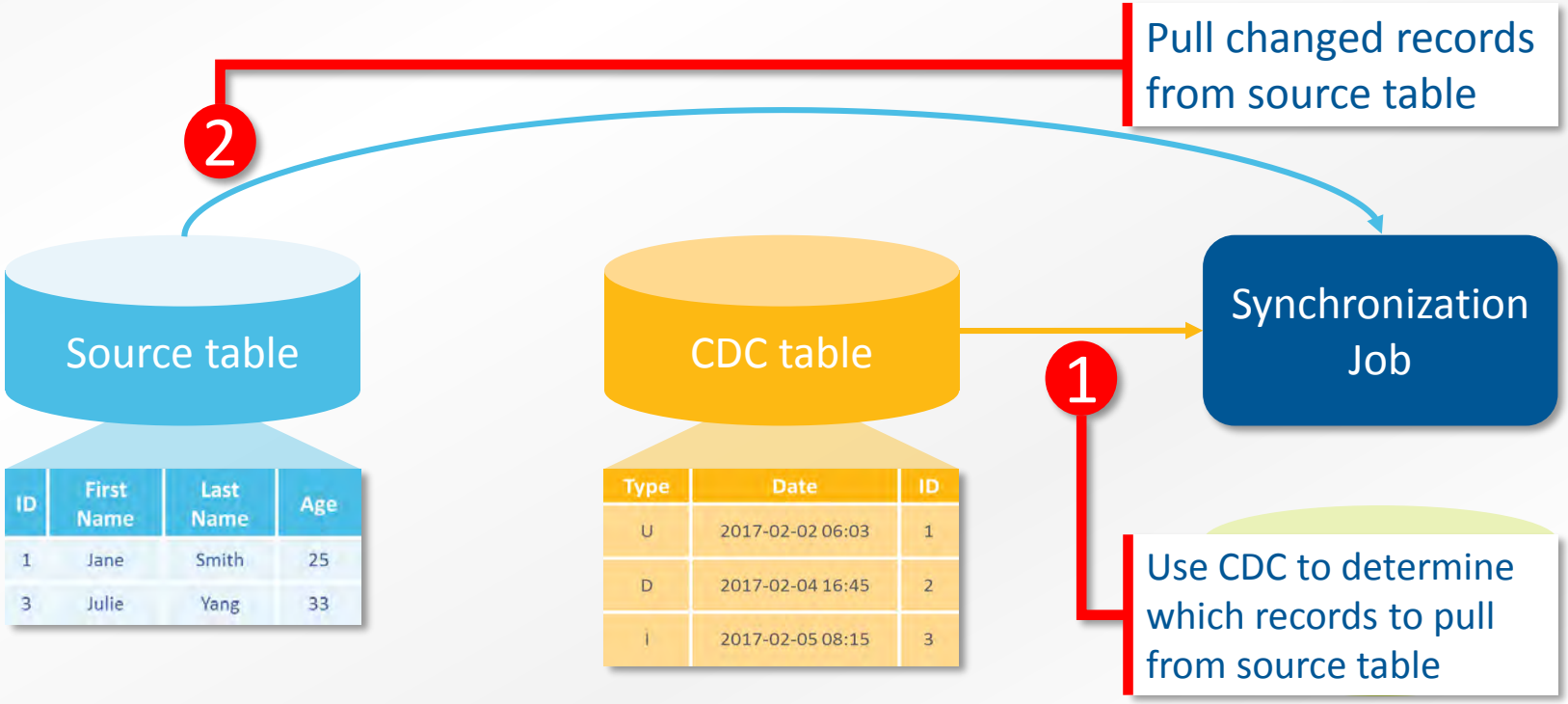
CDC Architecture

Modifications to Source Table



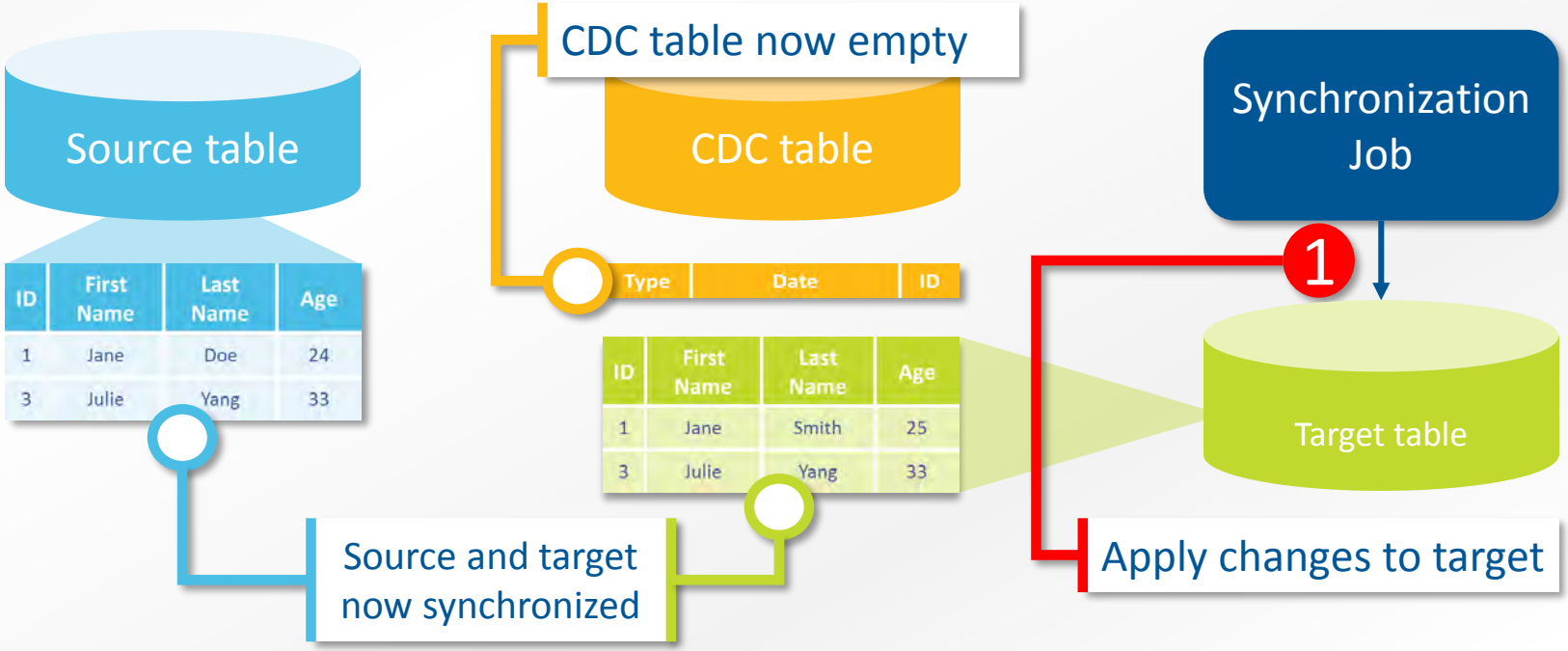
CDC Architecture

Pulling Changes from Source



CDC Architecture

Synchronizing the Target Table



CDC Database Support

Supported Databases

- Talend Studio supports many different databases:
  - Oracle
  - MySQL
  - DB2
  - PostgreSQL
  - Sybase
  - MS SQL Server
  - Informix
  - Ingres
  - Teradata
  - AS/400

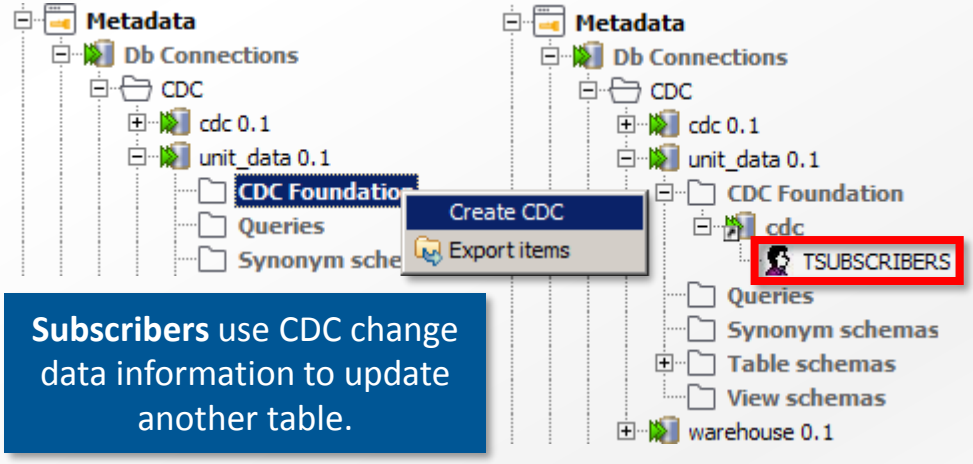
CDC Database Support

Supported Modes

- Supported CDC modes:
  - **Trigger** (default mode used by CDC components)
  - **Redo/archive log** (used with Oracle v11 and previous versions and AS/400)
  - **Xstream** (used only with Oracle v12 with OCI)
- Note that all modes are not supported by all databases

Setting Up CDC in Studio

- Use **Create CDC** to connect to the database and create a CDC table
  - The CDC table stores critical change information for a monitored table

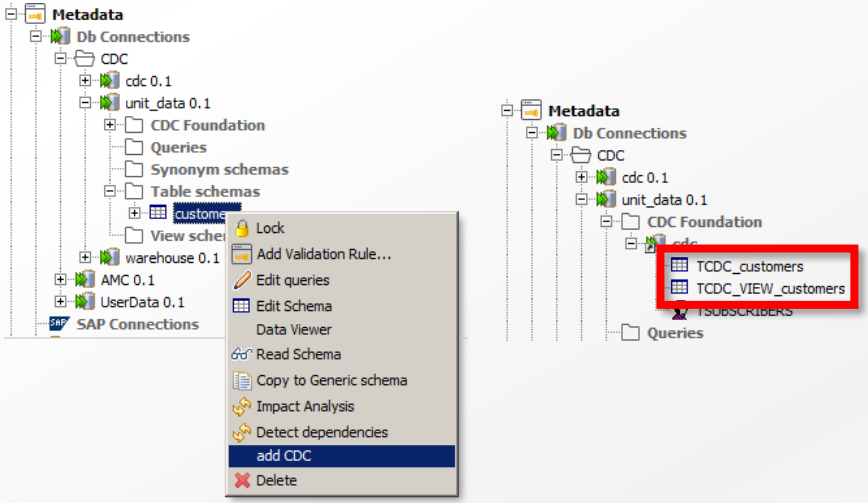


Subscribers use CDC change data information to update another table.

1. Create CDC
2. Add CDC
3. View Changes
4. Use **tMysqlCDC**

Setting Up CDC in Studio

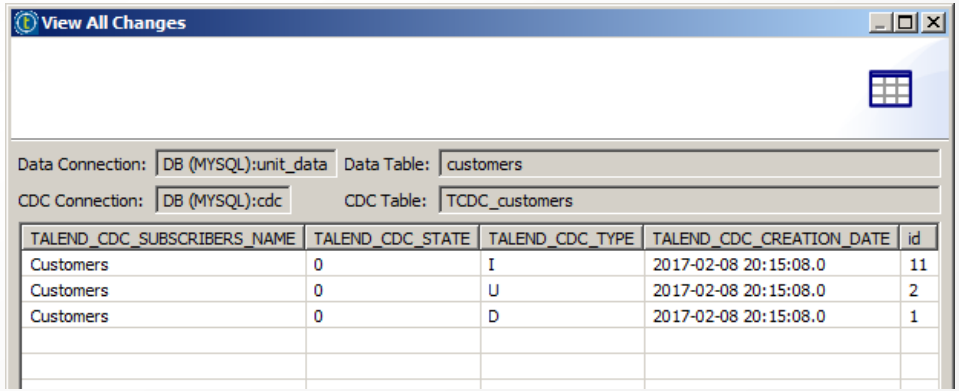
- Use **Add CDC** to specify the table to monitor
  - Tables that are monitored by CDC are labeled “CDC” in the repository



1. Create CDC
2. Add CDC
3. View Changes
4. Use **tMysqlCDC**

Setting Up CDC in Studio

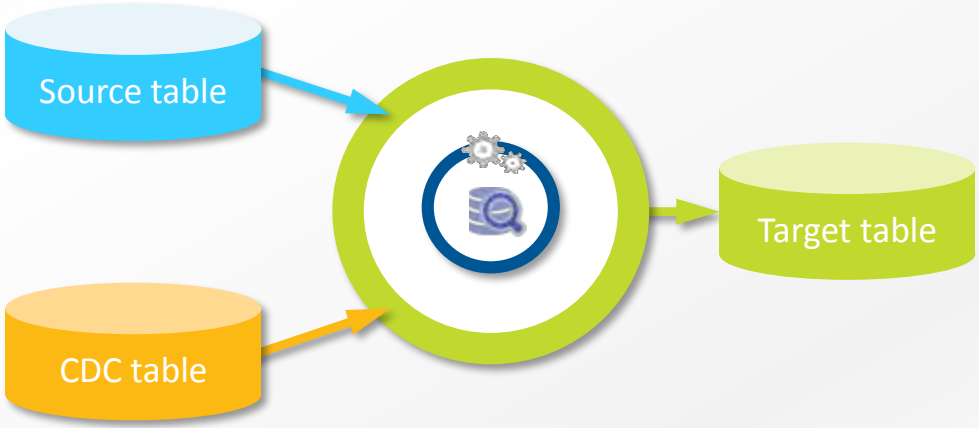
- Use **View All Changes** to see changes that have been collected
  - Good way to confirm proper CDC configuration



1. Create CDC
2. Add CDC
3. View Changes
4. Use **tMysqlCDC**

Setting Up CDC in Studio

- The **tMysqlCDC** component to the CDC table and pulls change events so they can be processed and applied to a target database table

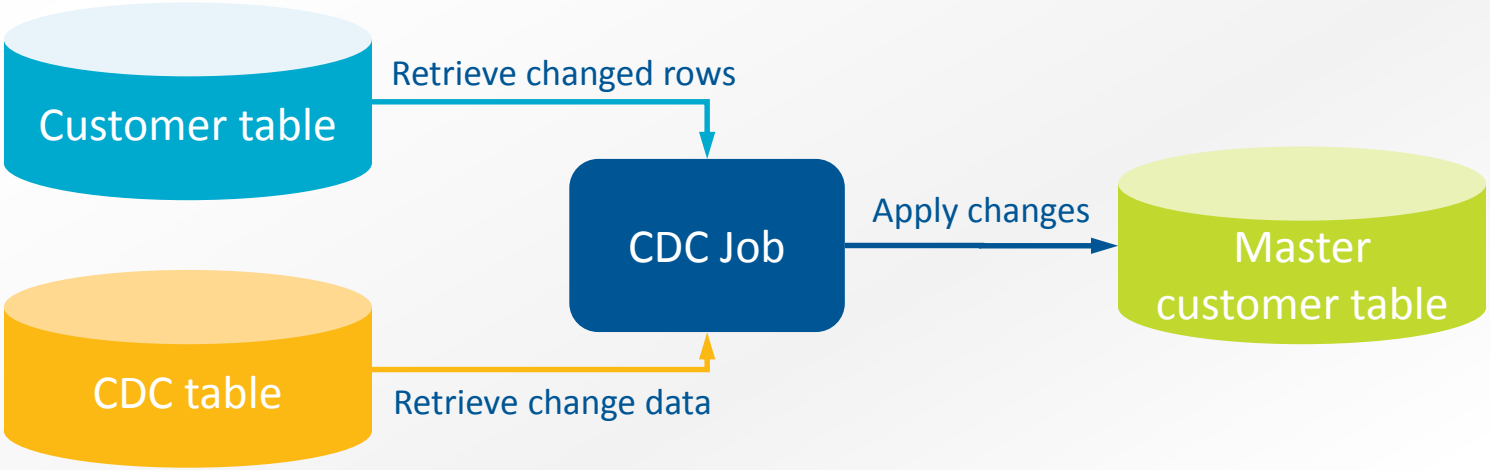


1. Create CDC
2. Add CDC
3. View Changes
4. Use **tMysqlCDC**

Lab Overview

Change Data Capture

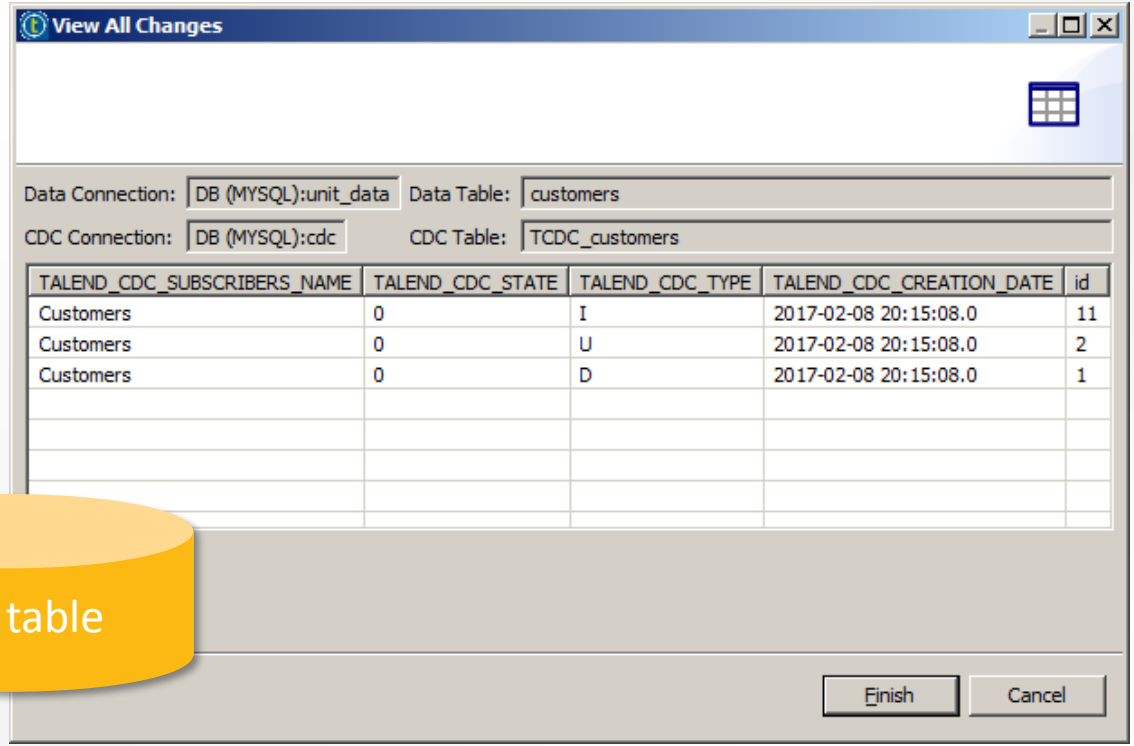
- Monitor a customer database table for changes.
  - Log all changes in a CDC table
  - Use it as input to a Job that updates a master customer table



Lab Overview

Change Data Capture

- Implement CDC in **trigger** mode within Studio to synchronize a customer database based on entries in a CDC table



Lesson Summary

- CDC is a common way for data warehouses to keep source and target databases synchronized
- Using a CDC database table with records tracking modifications (insert, update, delete) enables incremental updates to large database tables
- Implementing CDC allows you to circumvent copying massive database tables, saving valuable time and hardware resources