DevOps on Z
Proof of Technology (PoT)

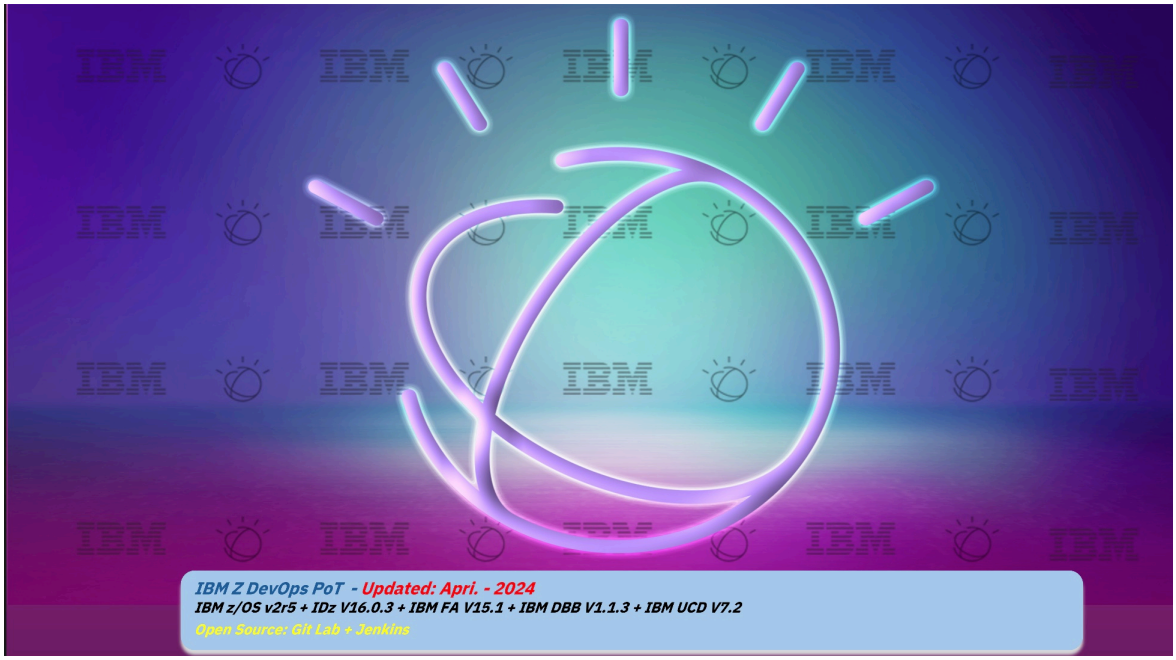
Exercises Using IBM ZDevops Tools

Updated ***2024***

Overview

- Integrated application development and problem analysis (ADFz) using Fault Analyser and z/OS Debugger
- Managing your source code using modern tools (including Git)
- Building automation (DBB) for COBOL or PL/1 without a specific source code manager or pipeline automation tool (including Groovy and Jenkins)
- Using GitLab CI along with DBB, and UrbanCode Deploy (UCD) on z/OS.
- Application deployments automation to many environments (UCD)

This material was built using a Windows on cloud that was updated on **April 2024**
Here the desktop background of this image:



The following symbols appear in this document at places where additional guidance is available.

Icon	Purpose	Explanation
	Important!	This symbol calls attention to a particular step or command. For example, it might alert you to type a command carefully because it is case sensitive.
	Information	This symbol indicates information that might not be necessary to complete a step, but is helpful or good to know.
	Trouble-shooting	This symbol indicates that you can fix a specific problem by completing the associated troubleshooting information.

LAB 1 - Working with mainframe using COBOL and DB2

Updated June,2024 (revised by Carlos Hirata/Wilbert Kho, Reviewed by Joe Huang and Frank Hernandez)

This lab will take you through the steps of using the Application Delivery Foundation for z Systems (ADFz) to work with a z/OS system. It will familiarize you with some of the capabilities of this product using a DB2 COBOL batch program that is ABENDing.

On this lab you will connect to a remote z/OS system, submit and execute a program (which ABENDs), identify the program abend, set up a MVS project, edit, compile, and debug a COBOL application. The process would be similar for a PL/I program.

Overview of development tasks

To complete this tutorial, you will perform the following tasks:

1. **Connect to a z/OS System.**
   → You will connect to the z/OS system using a provided z/OS logon userid.
2. **Execute the DB2/COBOL batch program and verify the ABEND.**
   → You will submit a COBOL/DB2 batch to execute and verify the bug.
3. **Use Fault Analyzer to identify the cause of the ABEND**
   → You will use Fault Analyzer to identify what is causing the ABEND.
4. **Use the IBM Debug for a temporary fix**
   → You will modify the field content to bypass the bug
5. **Modify the COBOL code to fix the bug.**
   → You will be able to fix the bug changing the COBOL code.
6. **Use Code Coverage**
   → You can now verify program areas covered by the test case with Code Coverage

Section 1. Connect to a z/OS System

You will connect to the z/OS. This section is like LOGON to a TSO. You will use **empot01** as userid and password.

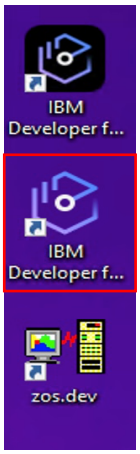
___ 1.1 Working with the IDz workspace

1.1.1 Start *IBM Developer for z Systems version 16*

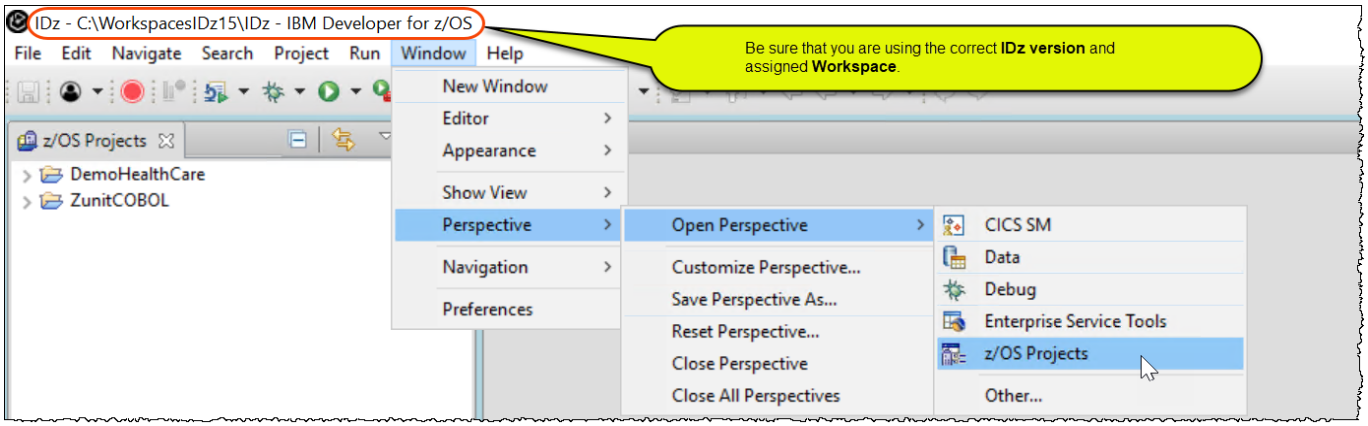
⏭ Using the windows desktop double click on **IDz V16** icon.

⏭ Verify that the message indicates that it is ***Version 16***

IMPORTANT -> This icon will start an eclipse workspace that has already some definitions required for this lab. PLEASE DO NOT start IDz using other way than this icon. The starting process might take some time .. Please be Patient.



1.1.2 ⏭ Open the z/OS Projects perspective by selecting
Window > Perspective > Open Perspective > z/OS Projects

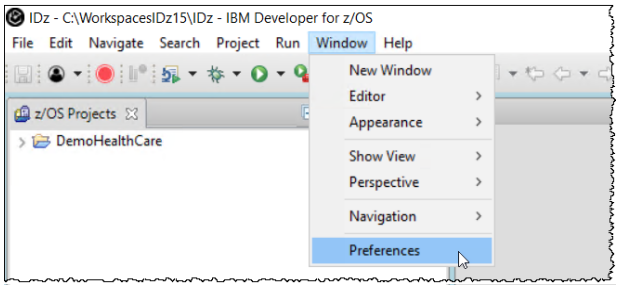


z/OS Projects perspective
Use the z/OS Projects perspective to define the connection, connect to, and work with remote systems, and to create, edit, and build projects, subprojects, and files on remote UNIX, Linux for z Systems, and z/OS systems.
The z/OS Projects perspective contains the many views like Remote Systems view, z/OS Projects view, Properties view, Outline view, Remote Error List view, z/OS File System Mapping view, Remote System Details view, Team view, Property Group Manager view and Snippets view

You can close and open views to customize the perspective.
To open one of the views that were closed:
1. In the workbench, select **Window** > **Show View**.
A menu that lists the views associated with the z/OS Projects perspective is displayed.
2. Click the name of the view you want to open.

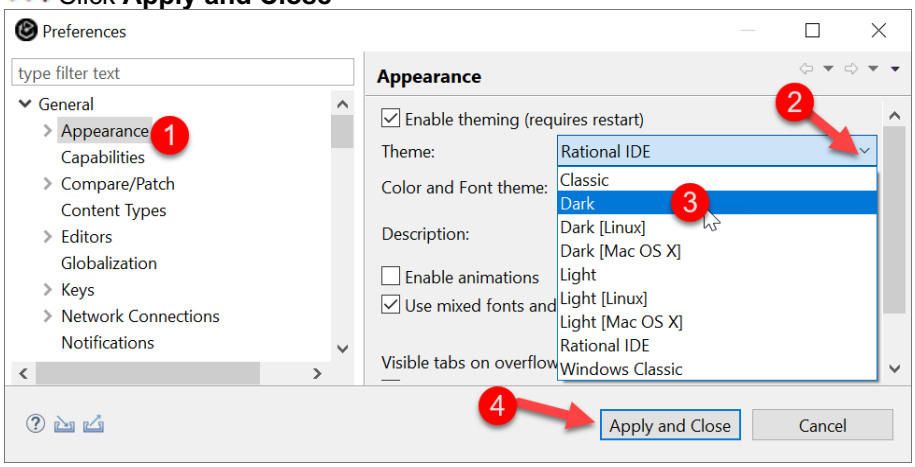
1.1.3 Starting on version 616 the developer can change the workspace appearance. If you want to try that

⏭ Click **Windows > Preferences**



1.1.4 ⏭ Under **General** select **Appearance** and on *Theme* select **Dark**.

⏭ Click **Apply and Close**

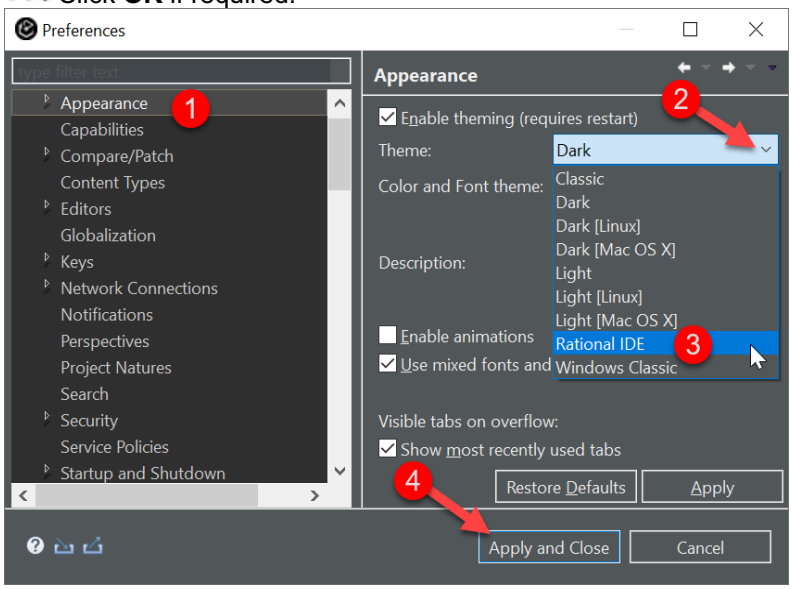


1.1.5 Since all picture here were done using “Rational IDE” theme we recommend to return to the default.

⏭ Again, click **Windows > Preferences**

⏭ Under **General** select **Appearance** and on *Theme* select **Rational IDE** and click **Apply and Close**

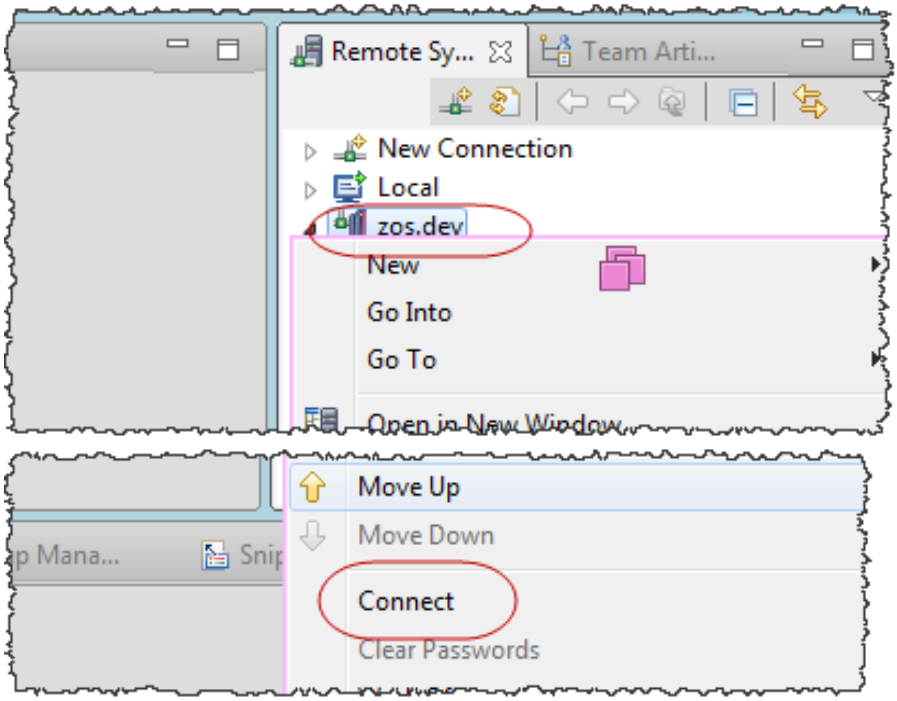
Click **OK** if required.



___ 1.2 Connecting to the z/OS Remote system

1.2.1 To Connect to the z/OS system:

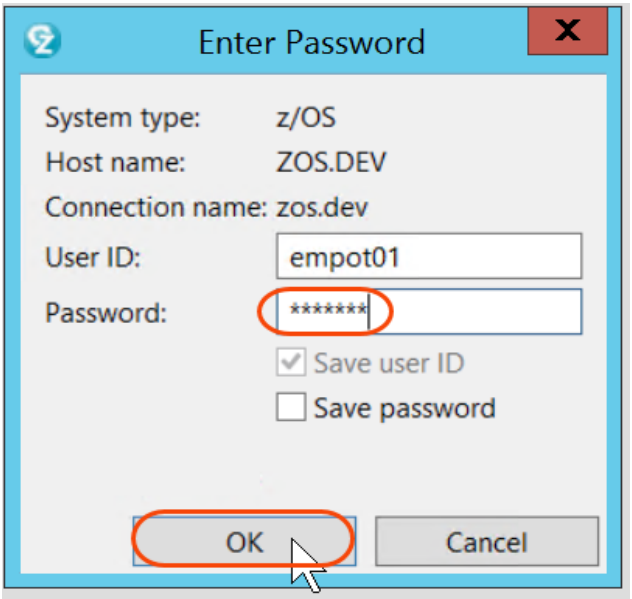
⏭ Using the *Remote Systems view* on the top and right side of the screen:
Right-click on **zos.dev** and select **Connect**
Notice: Since you are using a cloud instance the response to the right click may be delayed first time.



1.2.2 You will be are prompted for your z/OS userid and password.

⏭ Type **empot01** as userid (might be already there) and **empot01** as password.
The userid and password can be any case; don't worry about having it in UPPER case.

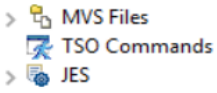
1.2.3 ⏭ Click **OK** to connect to z/OS.

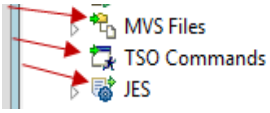


Be Patient! The connection could take a while depending on the network condition.

Wait until connection is complete (look at the bottom and left until the green bar disappears)

Notice that some folder icons changed after connected. A small **green arrow** is added.

BEFORE… 

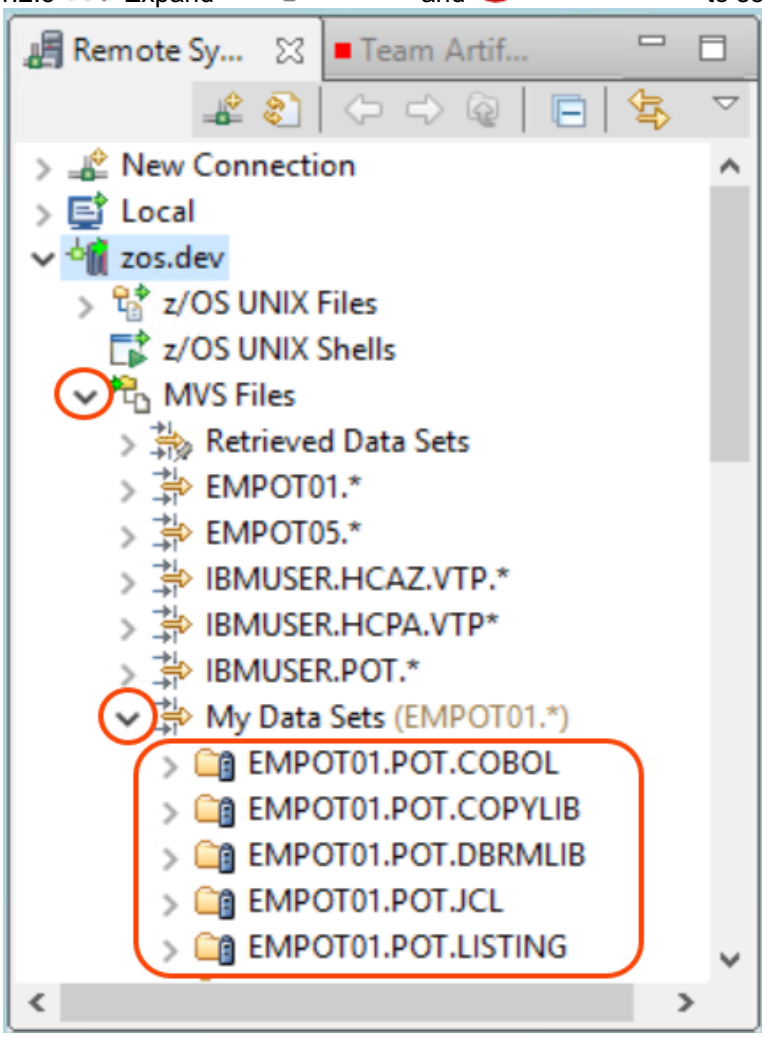
AFTER: 

In case of connection errors…

If you have errors during the connection it is because the z/OS system is not available.
You also may have network issues.

An easy way to find if z/OS is up is opening a 3270 emulation clicking on the icon that is on the base of your windows client. (zos.dev is not "pingable" .
If you do not receive a reply, Contact the instructor. Your connection is broken.

1.2.5 Expand MVS Files and My Data Sets (to see MVS data sets that starts with EMPOT01.*



Remote Systems view

The Remote Systems view shows all existing connections to remote systems.
Connections are persisted, containing the information needed to access a remote host.
The view contains a prompt to create new connections, and pop-up menu actions to rename, copy, delete, and reorder existing connections.
Connections contain attributes, or data, that are saved between sessions of the workbench. These attributes are the connection name, the remote system's host name and system type, an optional description, and a user ID that is used by default by each subordinate subsystem, at connection time.

Section 2. Execute the DB2/COBOL batch program and verify the ABEND.

To make your job easier, you will group together all the assets that you will work with. This is a new development concept for TSO/ISPF users, since TSO/ISPF does not provide such capability. To accomplish this task, you will create a **z/OS project** and select which assets will be part of this project.

What is a z/OS project?

After you define a z/OS-connection and connect to that system, you can define a z/OS project under that system. You can define the z/OS project, however, only when you are connected to the system. Application Delivery Foundation for z Systems assigns a set of default properties from the set of system properties. However, changes that you make to system properties do not affect the definition of an existing project. If you change your system properties to reference a new compiler release, for example, the reference affects only those projects that are defined subsequent to the change. This isolation of system data from existing projects is beneficial because it lets you develop your code without disruption. You can introduce changes to the project definition at a time of your choosing.

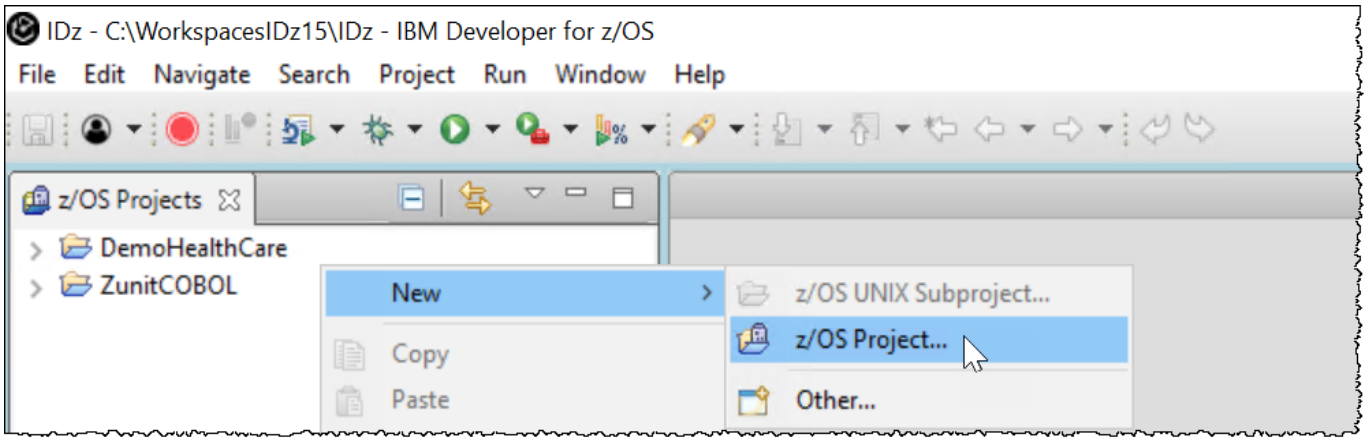
Creating a new MVS subproject

MVS subprojects contain the development resources that reside on an MVS system. You can create multiple subprojects in a z/OS project.
Before you create an MVS subproject, you need to have completed the following tasks:
→ Connecting to a remote system
→ Creating a z/OS project

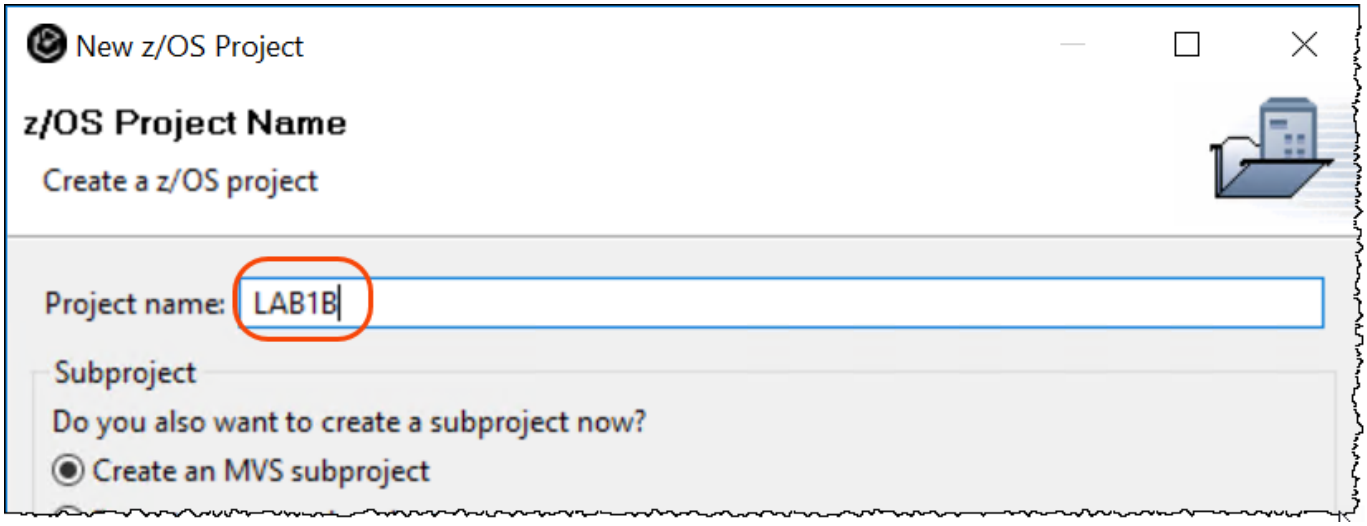
___ 2.1 Submit a COBOL/DB2 batch to execute

The advantage of creating a *z/OS Project* is that we just focus on those datasets and members that are being constructed or updated, instead of having all the mainframe datasets or members. At any time if you need to see a dataset not added to the project, just go to the z/OS Systems view and add it. In addition, at any time, you can remove from your project the datasets that you don't need anymore.

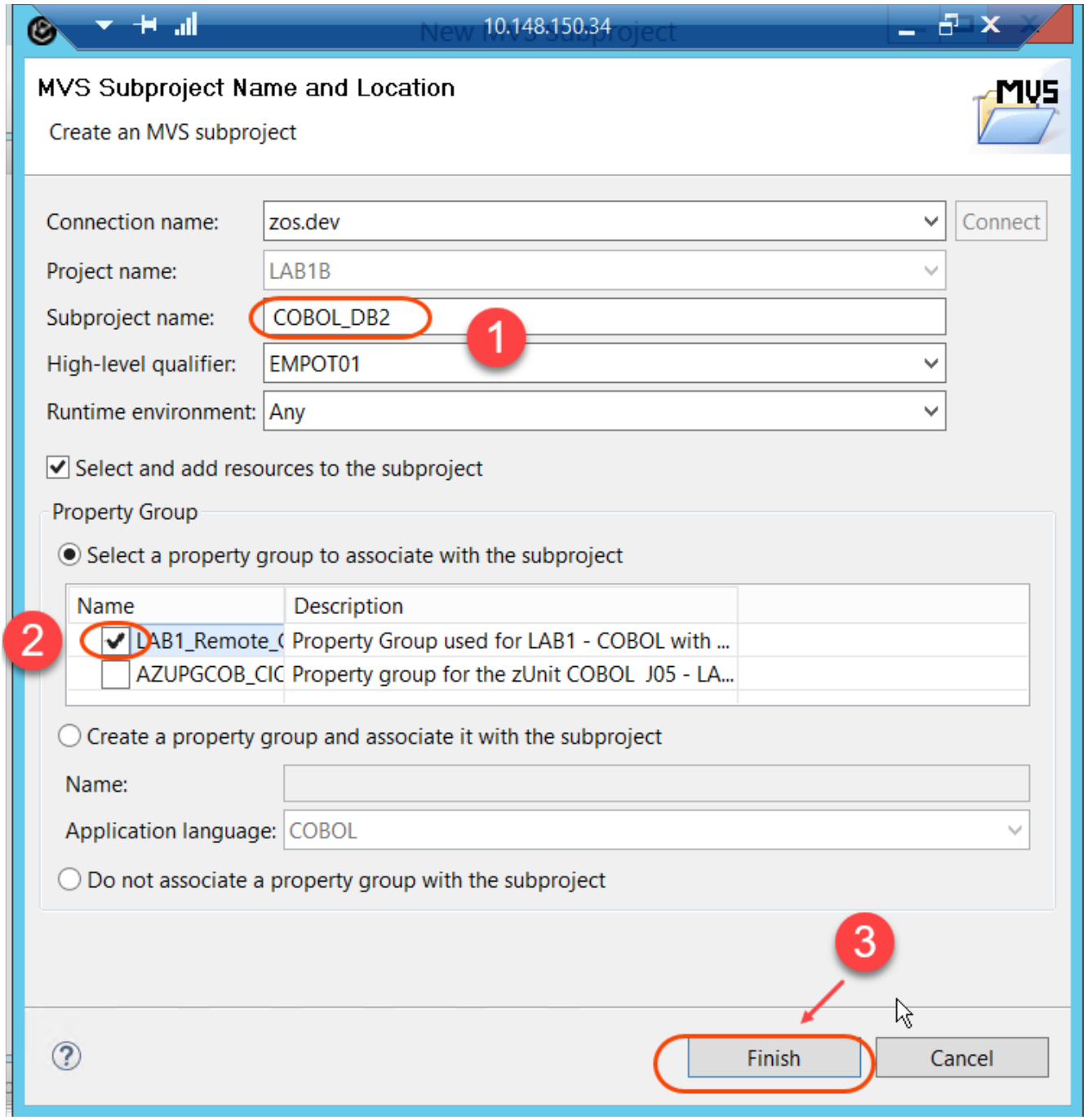
2.1.1 Using the *z/OS Projects* view (on the top and left), right click and select **New** > **z/OS Project…**



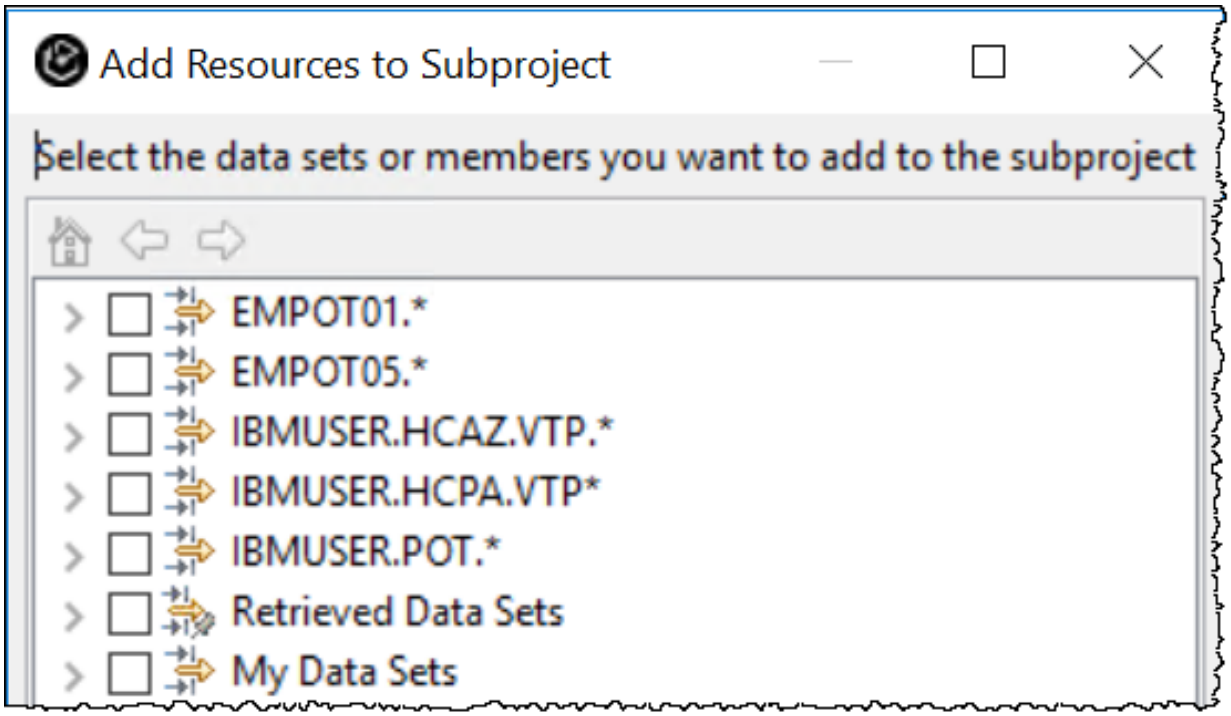
2.1.2 Type **LAB1B** as the *Project name,* select **Create an MVS subproject** and click **Finish.**



2.1.3 Type **COBOL_DB2** as *Subproject Name,* select **LAB1_Remote_Cobol..** as the property group and click **Finish .**
TIP: If using *Spanish or Brazilian* keyboard you must activate the correct language in the bottom, right corner, otherwise you will not be able to generate the "_".


10.148.150.34
MVS Subproject Name and Location
Create an MVS subproject
Connection name:
zos.dev
Connect
Project name:
LAB1B
Subproject name:
COBOL_DB2
1
High-level qualifier:
EMPOT01
Runtime environment:
Any
Select and add resources to the subproject
Property Group
Select a property group to associate with the subproject
Name
Description
2
LAB1_Remote_C
Property Group used for LAB1 - COBOL with ...
AZUPGCOB_CIC
Property group for the zUnit COBOL J05 - LA...
Create a property group and associate it with the subproject
Name:
Application language:
COBOL
Do not associate a property group with the subproject
3
Finish
Cancel

This dialog will be displayed:



Can't you find the property group above?

It is because you are using a wrong workspace. Close IDz, go back to the step 1.1.1 and be sure you start IDz from the icon that is on the windows desktop.

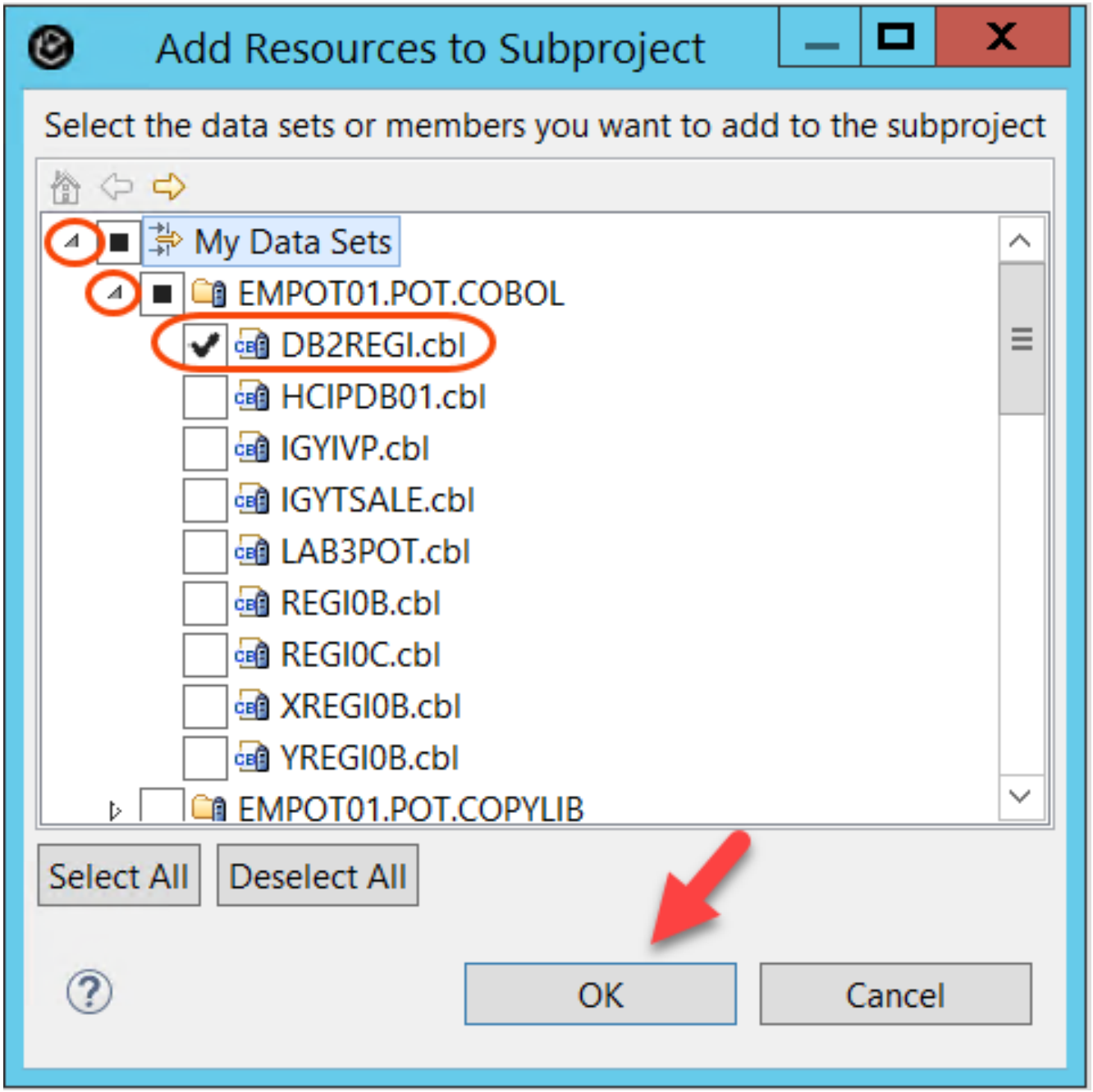
– PROPERTY GROUP

Property groups provide a way to manage resource properties, share them easily across systems, projects, resources, and users, and maintain consistency in your development and build environment.
You can, for example, define a property group that points to copybooks. This is something like `//STEPLIB`; otherwise, some of the variables used in the program are not resolved.

___ 2.2 Add z/OS resources to the MVS subproject

To make the data sets available to your remote project named *LAB1B,* you will need to add them. For this lab we just want to add one member..

2.2.1 ⏭ On the dialog below, expand **My Data Sets, EMPOT01.POT.COBOL ,** select **DB2REGI.cbl,** and click **OK**

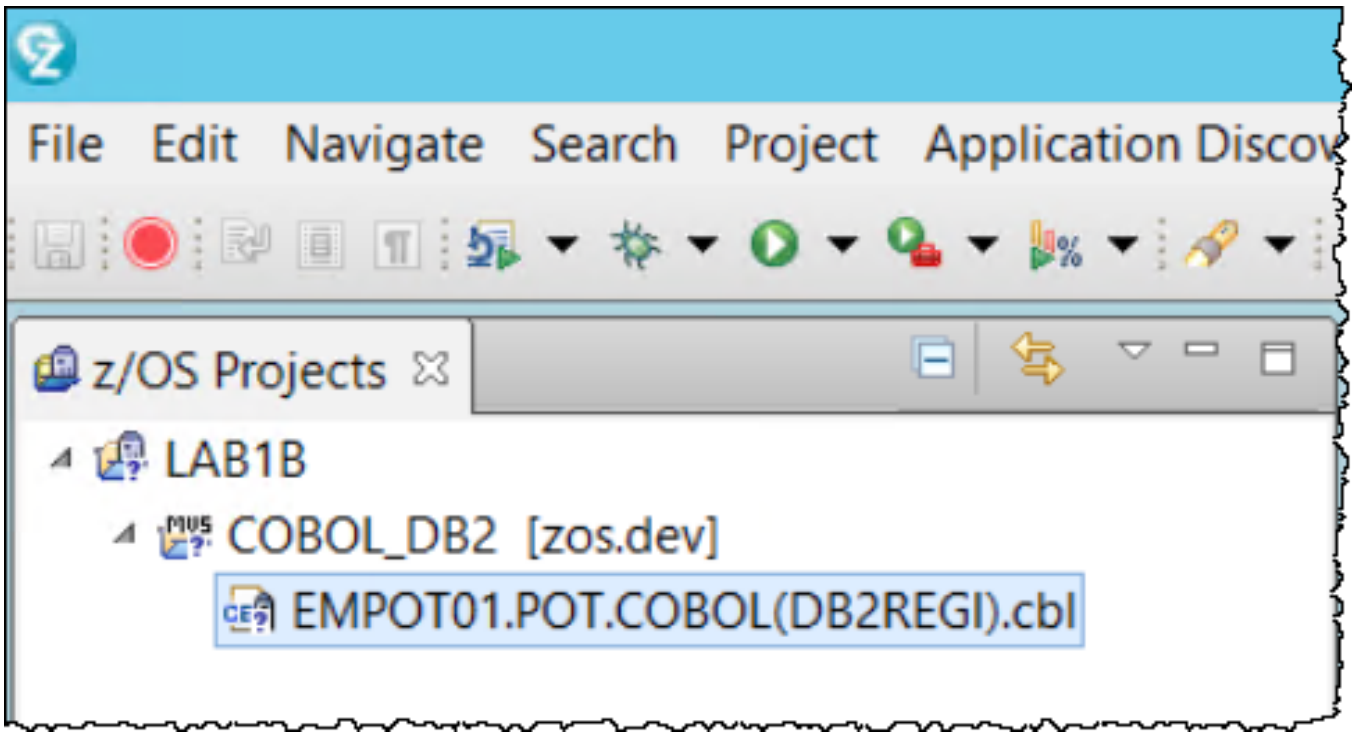

Add Resources to Subproject
Select the data sets or members you want to add to the subproject
My Data Sets
EMPOT01.POT.COBOL
DB2REGI.cbl
HCIPDB01.cbl
IGYIVP.cbl
IGYTSALE.cbl
LAB3POT.cbl
REGI0B.cbl
REGI0C.cbl
XREGI0B.cbl
YREGI0B.cbl
EMPOT01.POT.COPYLIB
Select All
Deselect All
OK
Cancel

2.2.2 You should see a z/OS Project named **LAB1B** in your *z/OS Projects view*.

Click ***z/OS Projects view*** (on left) and under **COBOL_DB2,** you will now see **DB2REGI** member added to your project.

Notice that creating a Remote project is a good practice to isolate the code you are working on, but you could work on your code without creating this project.

Our lab involves a small update so you could work directly on the *Remote System* perspective without having to create a Remote project..

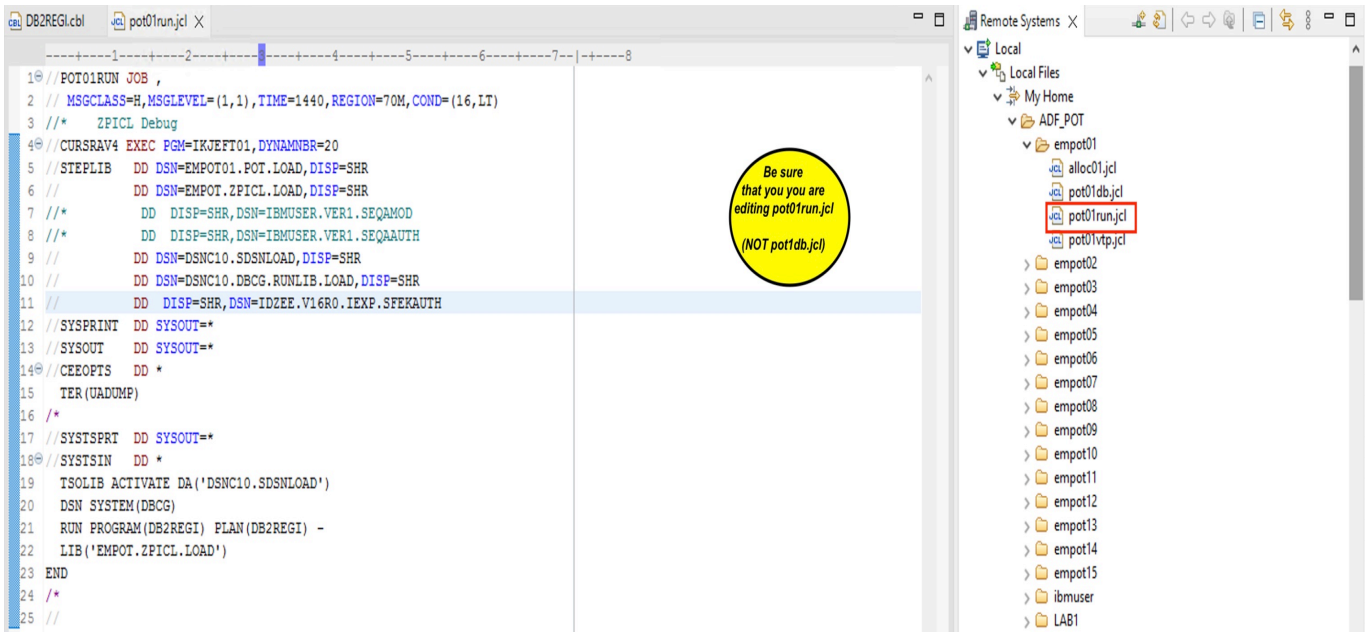


___ 2.3 Submit a JCL to execute the COBOL program

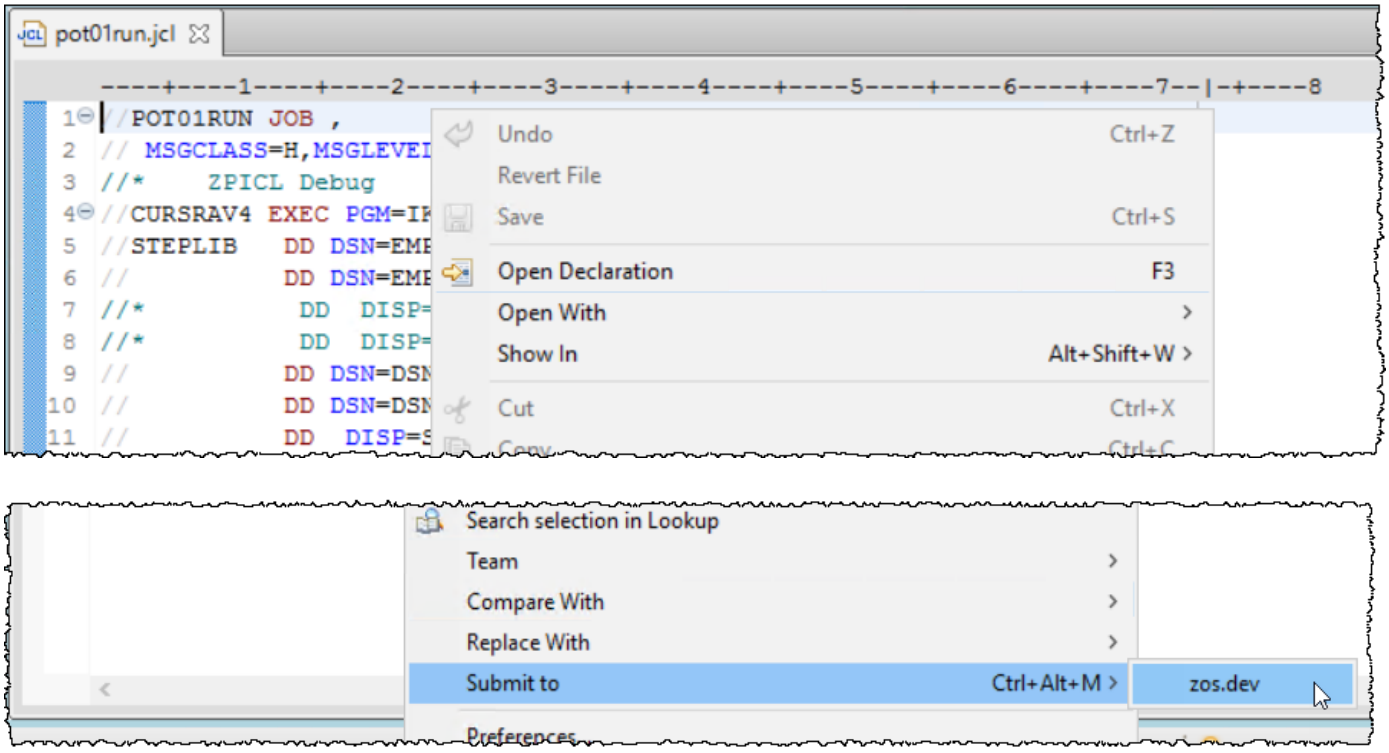
The JCL to execute the program is available on the local windows folder: "**`C:\ADF_POT\empot01`**".
You will use this JCL to execute a batch COBOL/DB2 on z/OS.

2.3.1 Using *Remote System View* (on top and right), **scroll up** to locate the file **`pot01run.jcl`** under **`Local/Local Files/ADF_POT/empot01`** and **double click** to edit.

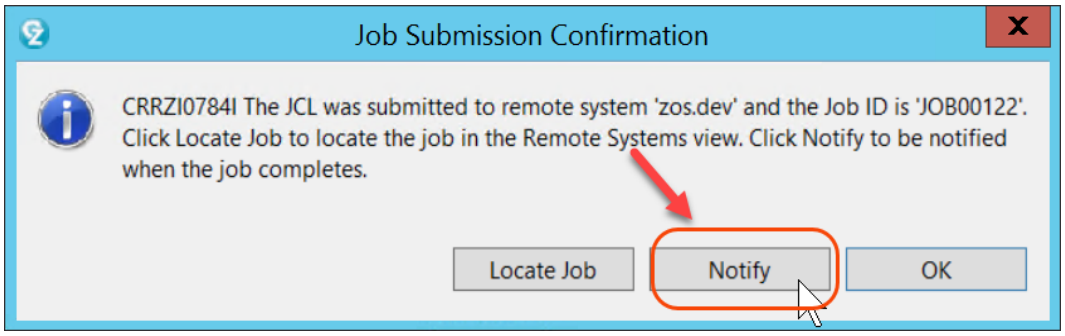
.


DB2REGI.cbl
pot01run.jcl
Be sure that you you are editing pot01run.jcl (NOT pot1db.jcl)
Remote Systems
Local
Local Files
My Home
ADF_POT
empot01
alloc01.jcl
pot01db.jcl
pot01run.jcl
pot01vtp.jcl
empot02
empot03
empot04
empot05
empot06
empot07
empot08
empot09
empot10
empot11
empot12
empot13
empot14
empot15
ibmuser
LAB1

2.3.2 **Right click** on the JCL being edited and select **Submit to → zos.dev**

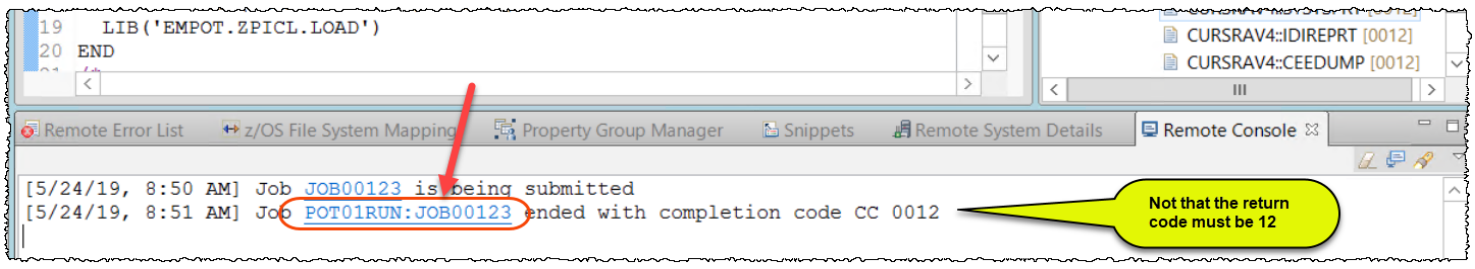


2.3.3 When the dialog pop up appears, click **Notify.** This allows you to be notified when the execution is ended.



2.3.4 Under *Remote Console*, you will be notified when execution is completed.

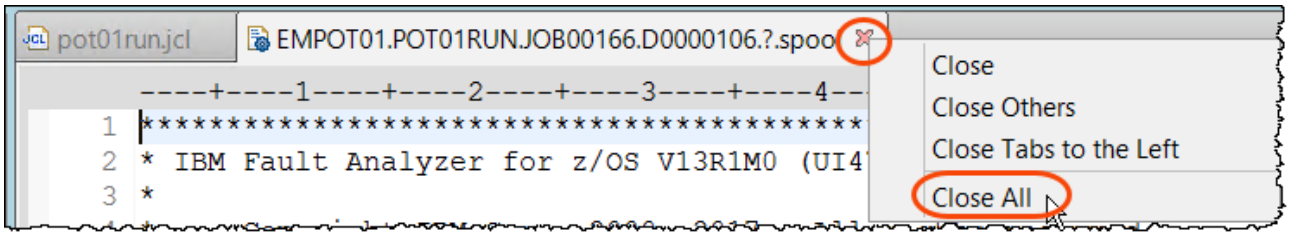
Once the execution ends, **click** on the link **POT01RUN:JOB00xxx** (where 00x**xx** is a number) to point to the Job output. This number vary depending on z/OS.



2.3.5 Under *Remote Systems* view **scroll down**, expand **JES > Retrieved Jobs > POT01RUN:JOB00xxx** and **double click CURSRAV4::IDIREPRT** step and you will see the *Fault Analyzer* report showing an ***OCB*** ABEND. Your mission is to fix that bug.

Tip: If there is no jobs under "*Retrieved Jobs"*, is because you did not click on link as stated at 2.3.4.
You can also see this output once you right click on "*My Jobs*" under *JES* and select ***Refresh***.

362.3.6 Close all the opened editors using **Ctrl + Shift + F4,**
Or right click on the . and choose **Close all.**



Section 3. Use Fault Analyzer to identify the cause of the ABEND

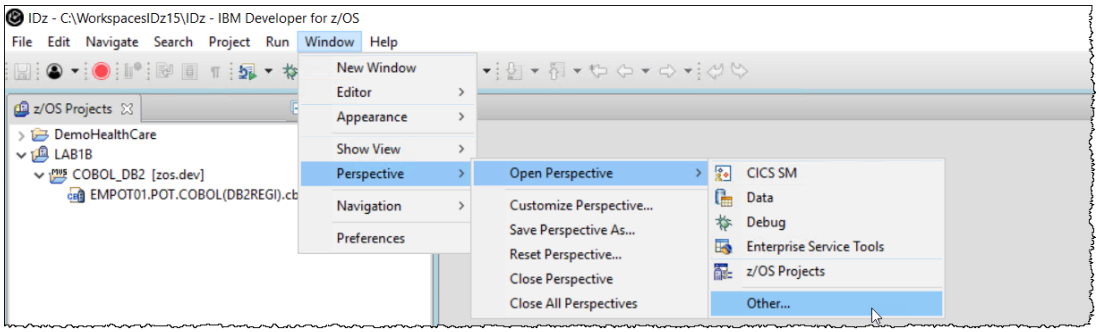
You will now take advantage of **IBM Fault Analyzer** (that is part of Application Delivery Foundation- ADF) to verify the cause of the ABEND.

What is IBM Fault Analyzer for z/OS?

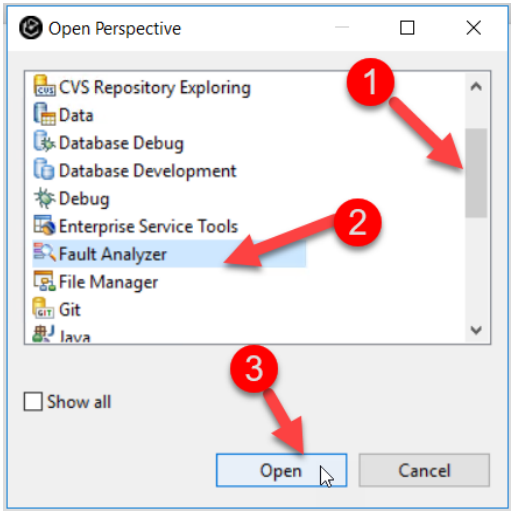
IBM Fault Analyzer for z/OS helps developers analyze and fix system and application failures for CICS, WebSphere MQ, IMS and DB2 environments. When an application ends abnormally, Fault Analyzer is automatically engaged, gathering real-time information about the event and its environment at the time of failure.
This helps the development team to identify the cause, analyze what went wrong and resolve the problem in a more timely and efficient manner to avoid costly interruptions that could jeopardize application schedules and outcomes.

___ 3.1 Using Fault Analyzer perspective

3.1.1 Open the *Fault Analyzer* perspective using **Window > Perspective > Open Perspective > Other…**

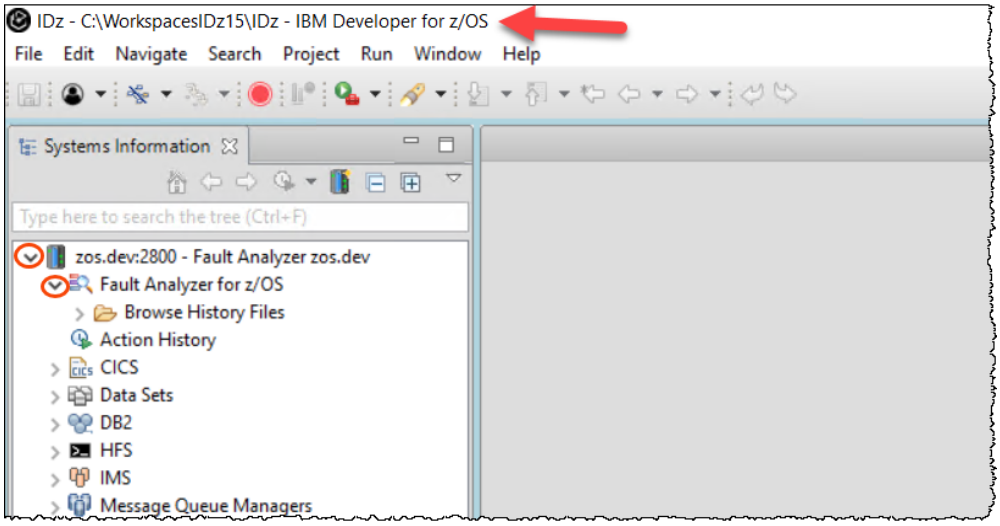


3.1.2 Scroll down, select **Fault Analyzer** and click **Open**

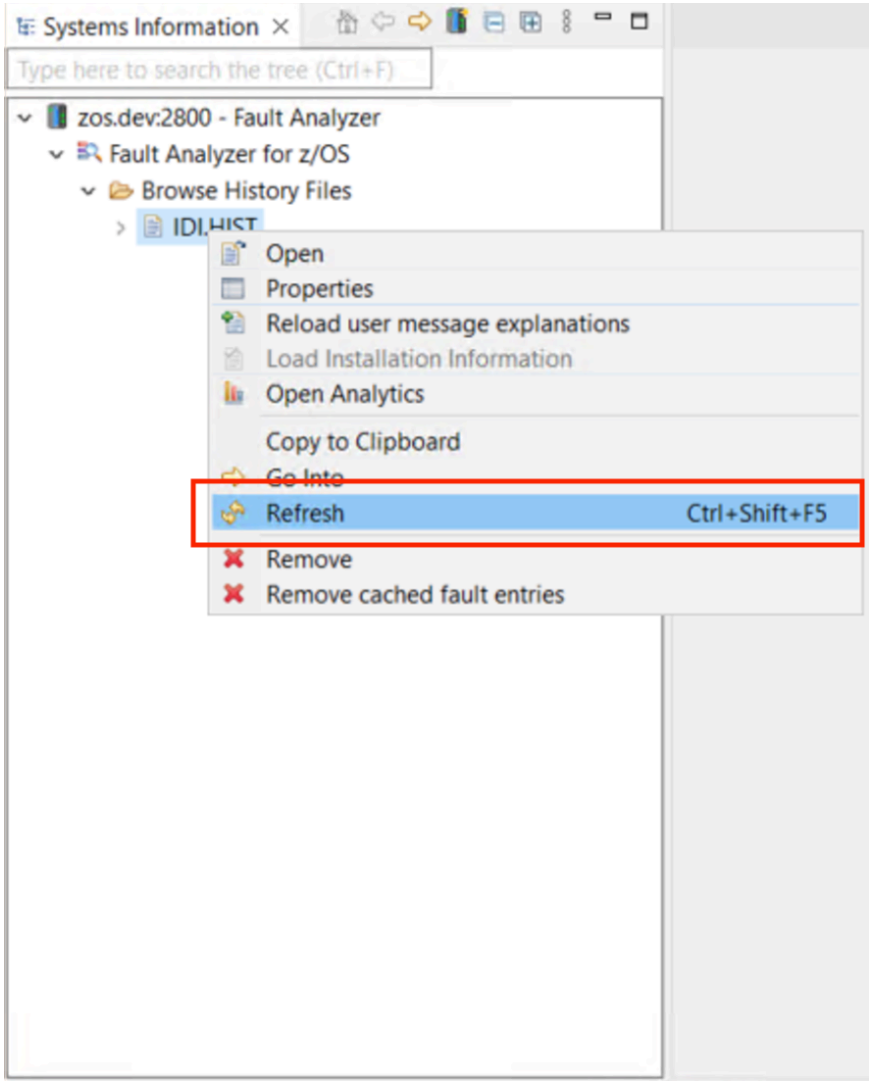


3.1.3 Under *Systems Information* view (on left) be sure that **Fault Analyzer for z/OS** is expanded

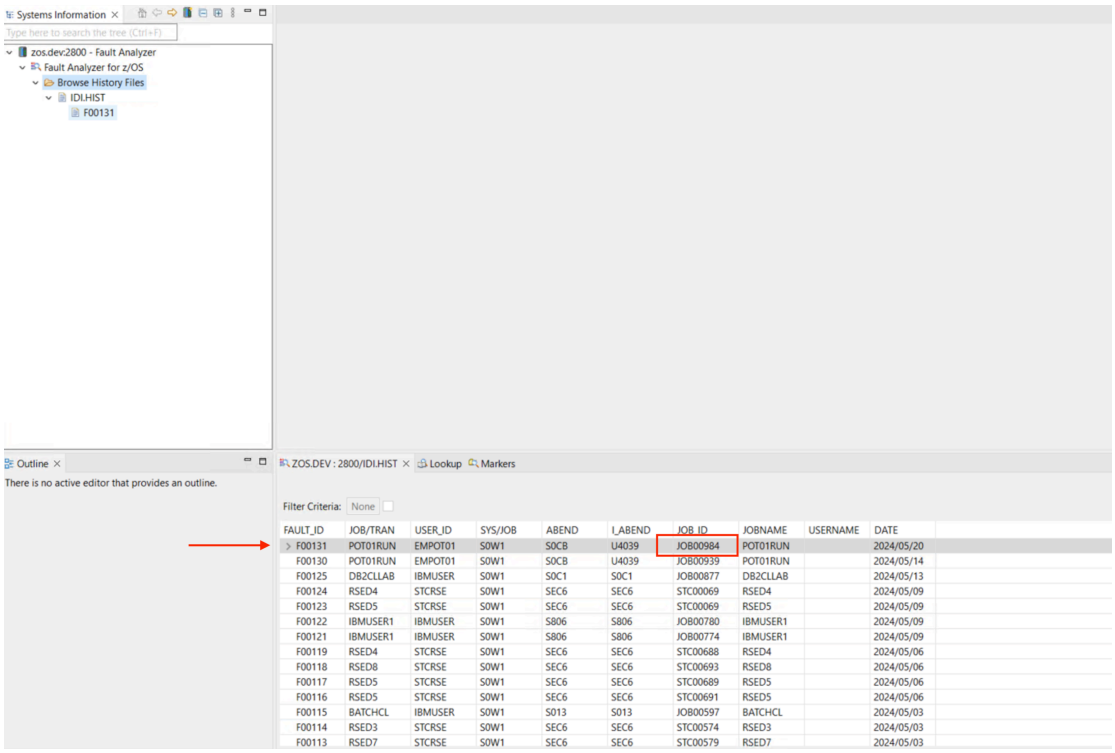
TIP: If you do not have this entry, it is because you are using a wrong IDz workspace. Please contact the instructor.



3.1.4 ⏭ Right click **Fault Analyzer for z/OS** and select **Refresh**
⏭ Expand **Browse History File**. You will see the folder ***IDI.HIST***
⏭ Right click **IDI.HIST** and select **Refresh** (or **Ctrl + Shift + F5**)

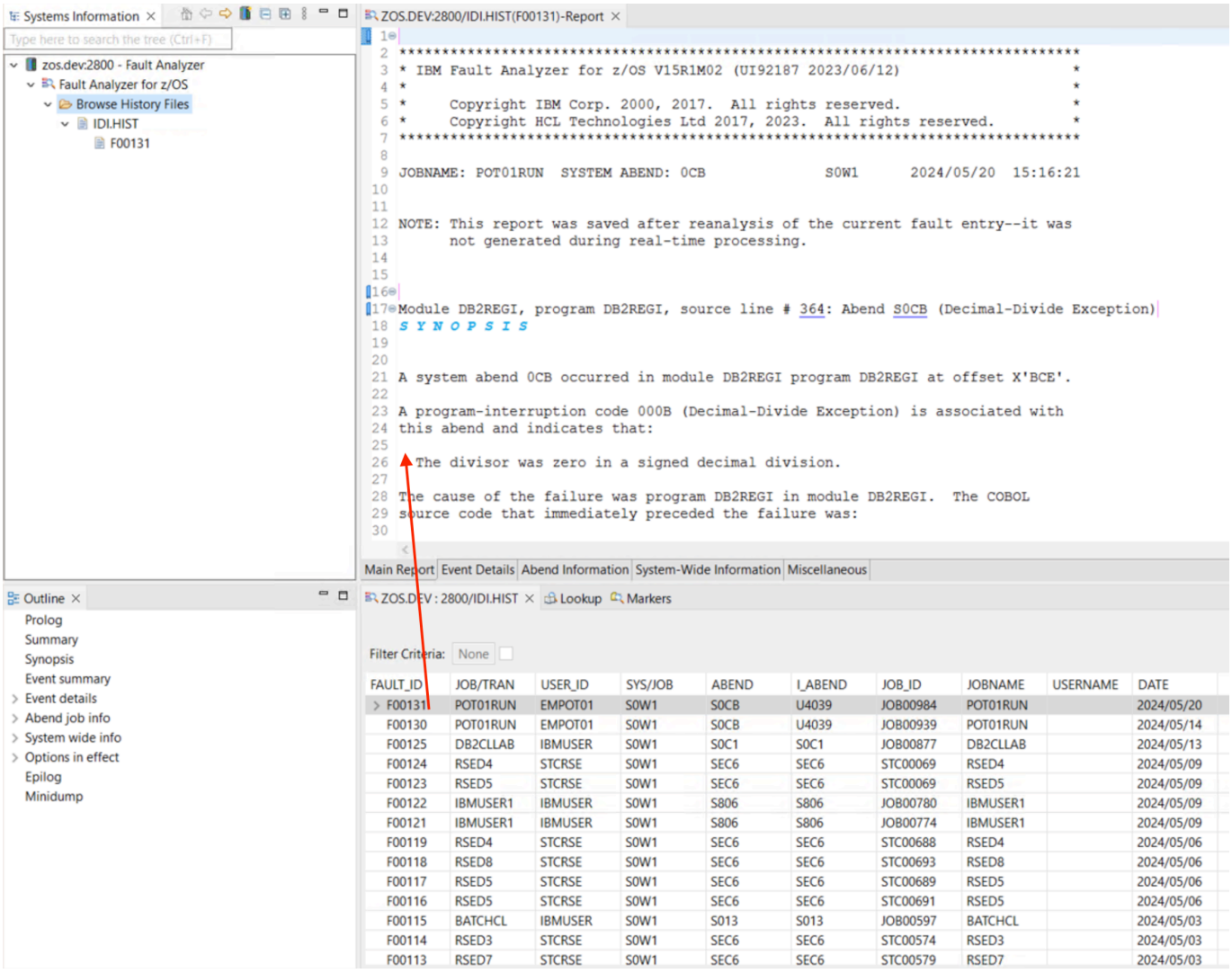


3.1.5 ⏭ If you get a *Sign on* dialog for *Fault Analyzer* use empot01 credentials and click **OK**
You see a list of *FAULT_IDs* on the z/OS system that you are connected to...



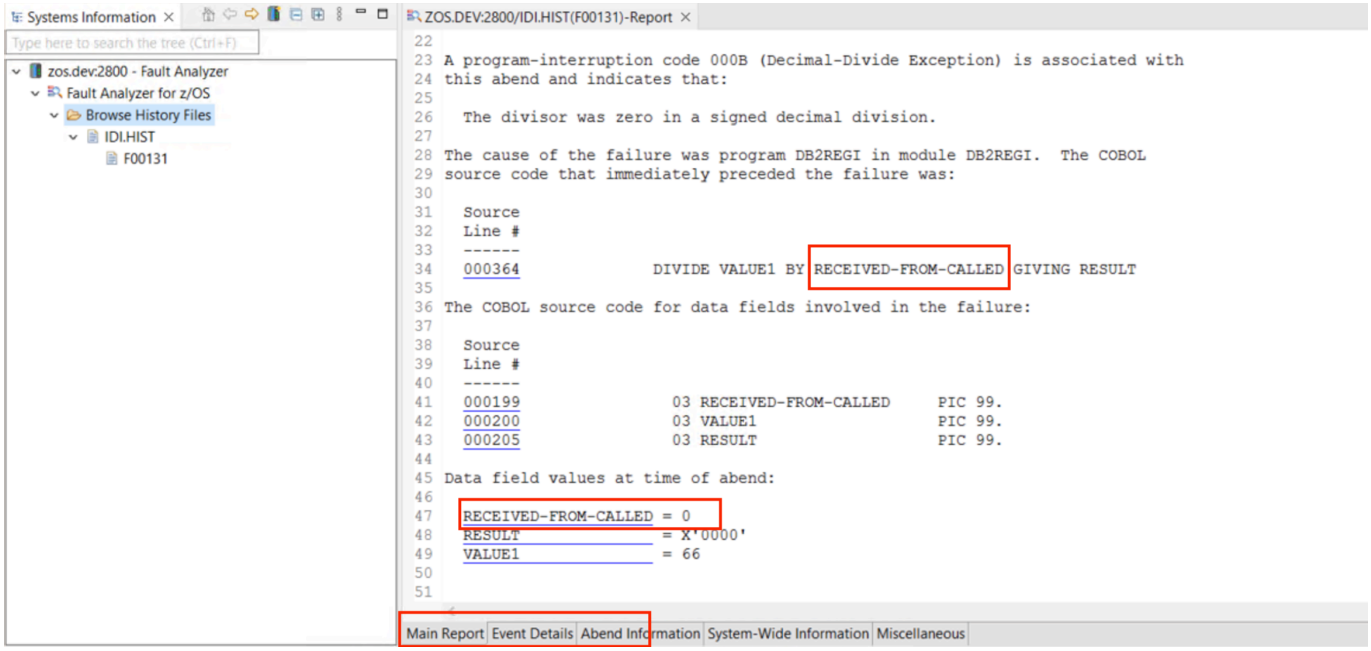
FAULT_ID	JOB/TRAN	USER_ID	SYS/JOB	ABEND	I_ABEND	JOB_ID	JOBNAME	USERNAME	DATE
F00131	POT01RUN	EMPOT01	S0W1	S0CB	U4039	JOB00984	POT01RUN		2024/05/20
F00130	POT01RUN	EMPOT01	S0W1	S0CB	U4039	JOB00939	POT01RUN		2024/05/14
F00125	DB2CLLAB	IBMUSER	S0W1	S0C1	S0C1	JOB00877	DB2CLLAB		2024/05/13
F00124	RSED4	STCRSE	S0W1	SEC6	SEC6	STC00069	RSED4		2024/05/09
F00123	RSED5	STCRSE	S0W1	SEC6	SEC6	STC00069	RSED5		2024/05/09
F00122	IBMUSER1	IBMUSER	S0W1	S806	S806	JOB00780	IBMUSER1		2024/05/09
F00121	IBMUSER1	IBMUSER	S0W1	S806	S806	JOB00774	IBMUSER1		2024/05/09
F00119	RSED4	STCRSE	S0W1	SEC6	SEC6	STC00688	RSED4		2024/05/06
F00118	RSED8	STCRSE	S0W1	SEC6	SEC6	STC00693	RSED8		2024/05/06
F00117	RSED5	STCRSE	S0W1	SEC6	SEC6	STC00689	RSED5		2024/05/06
F00116	RSED5	STCRSE	S0W1	SEC6	SEC6	STC00691	RSED5		2024/05/06
F00115	BATCHCL	IBMUSER	S0W1	S013	S013	JOB00597	BATCHCL		2024/05/03
F00114	RSED3	STCRSE	S0W1	SEC6	SEC6	STC00574	RSED3		2024/05/03
F00113	RSED7	STCRSE	S0W1	SEC6	SEC6	STC00579	RSED7		2024/05/03

3.1.6 On the ***ZOS.DEV:2800/IDI.HIST*** view locate the latest job name **POT01RUN** and **double click** on it. You will see the report being downloaded from the z/OS to your Windows client. The report below will be displayed

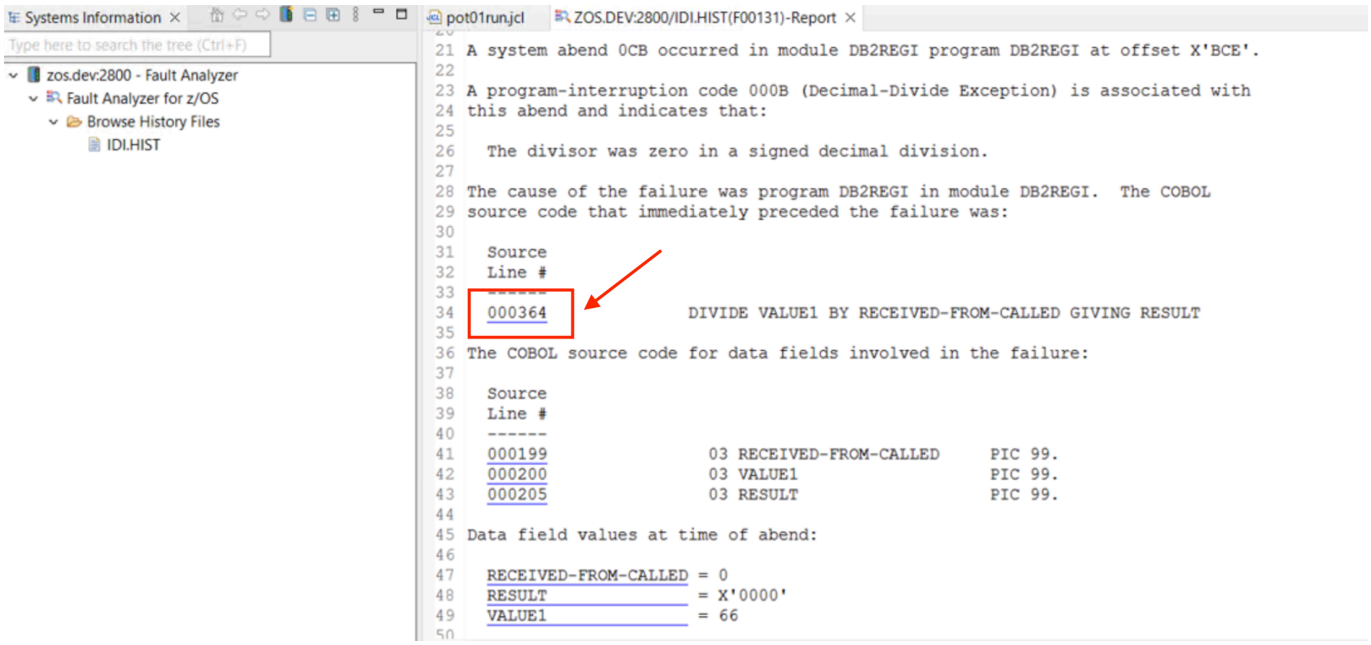


3.1.7 Scroll the report down and you will see that the field **RECEIVED-FROM-CALLED** used on the division has “0”. So the abend is because of a divide by zero.

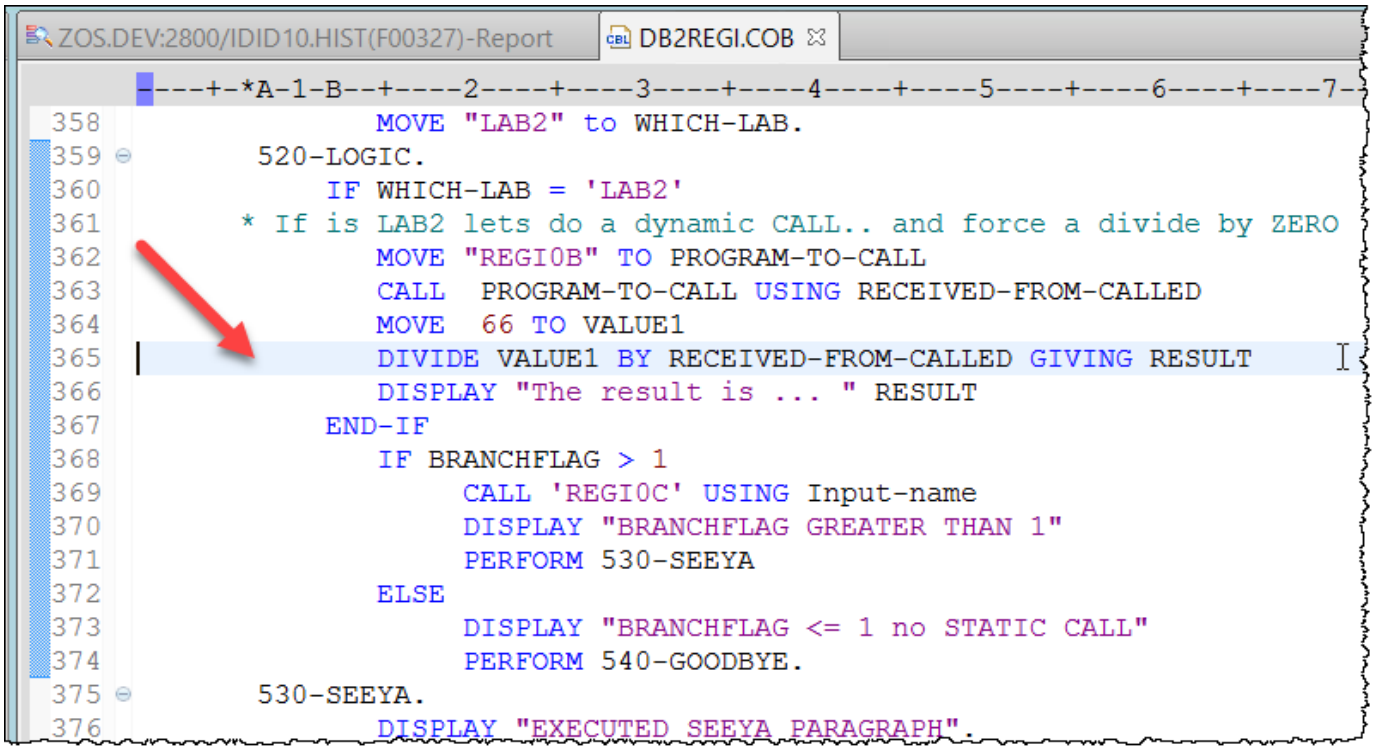
Also notice that there are other tabs on this panel (like Event Details, Abend Information, etc.). You may try those tabs to get more information about the abend.



3.1.8 Using the *Main Report* view Click on the blue link
This will show the COBOL statement that caused the *0CB abend*

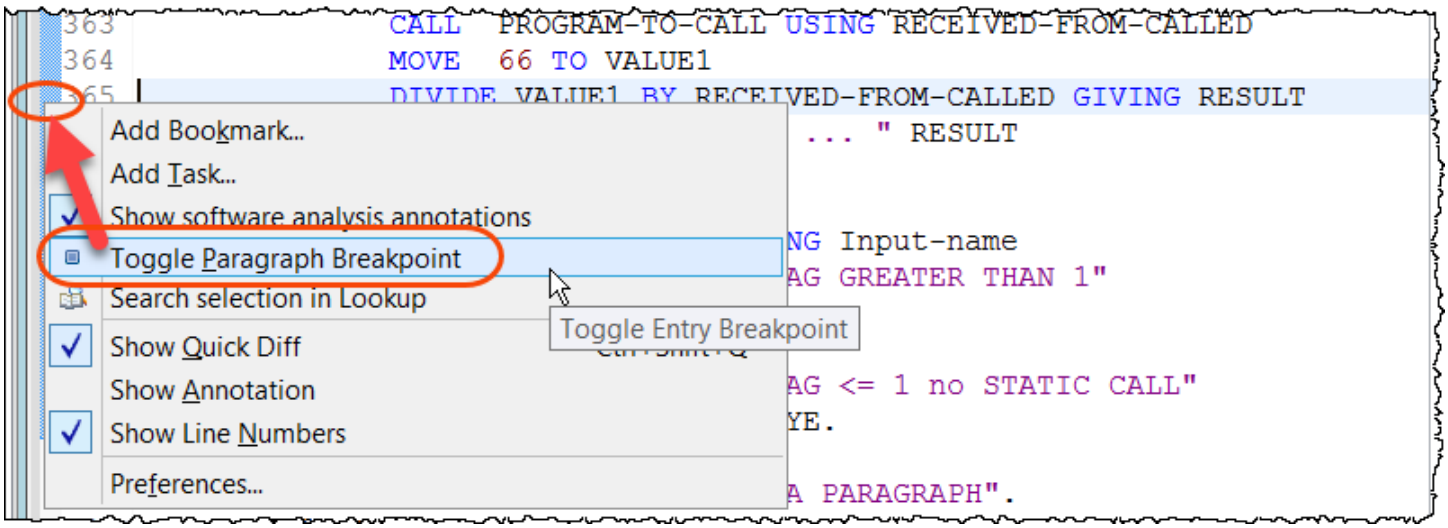


3.1.9 The program editor shows the line with the statement that is causing the abend.
Notice that this is NOT the COBOL source code. This what is on the “minidump” downloaded by Fault Analyzer. There is no sense to make changes here. But you will see what caused the abend.

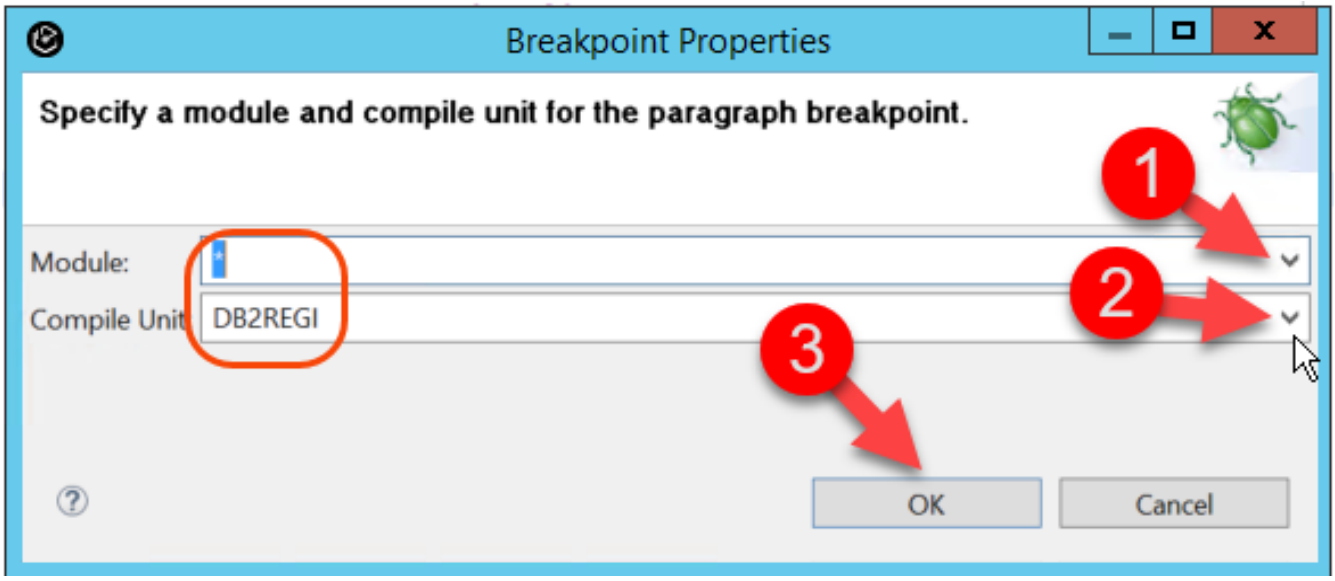


Tip: If the font is not clear on the editor, it is because we increased the font to 150%. To make this smaller start the **Control Panel** > **Display** > **set a custom scaling level** > reduce to **125%** (we did set to 150%).

3.1.10 On line " **DIVIDE VALUE1 BY RECEIVED-FROM-CALLED …"** , right-click in the ruler area (the blue line on left) and
select **Toggle Paragraph Breakpoint**. (also known as Toggle Entry Breakpoint)



3.1.11 On the Breakpoint Properties dialog use drop down to select *, and select or type **DB2REGI** as Compile Unit and click **OK**



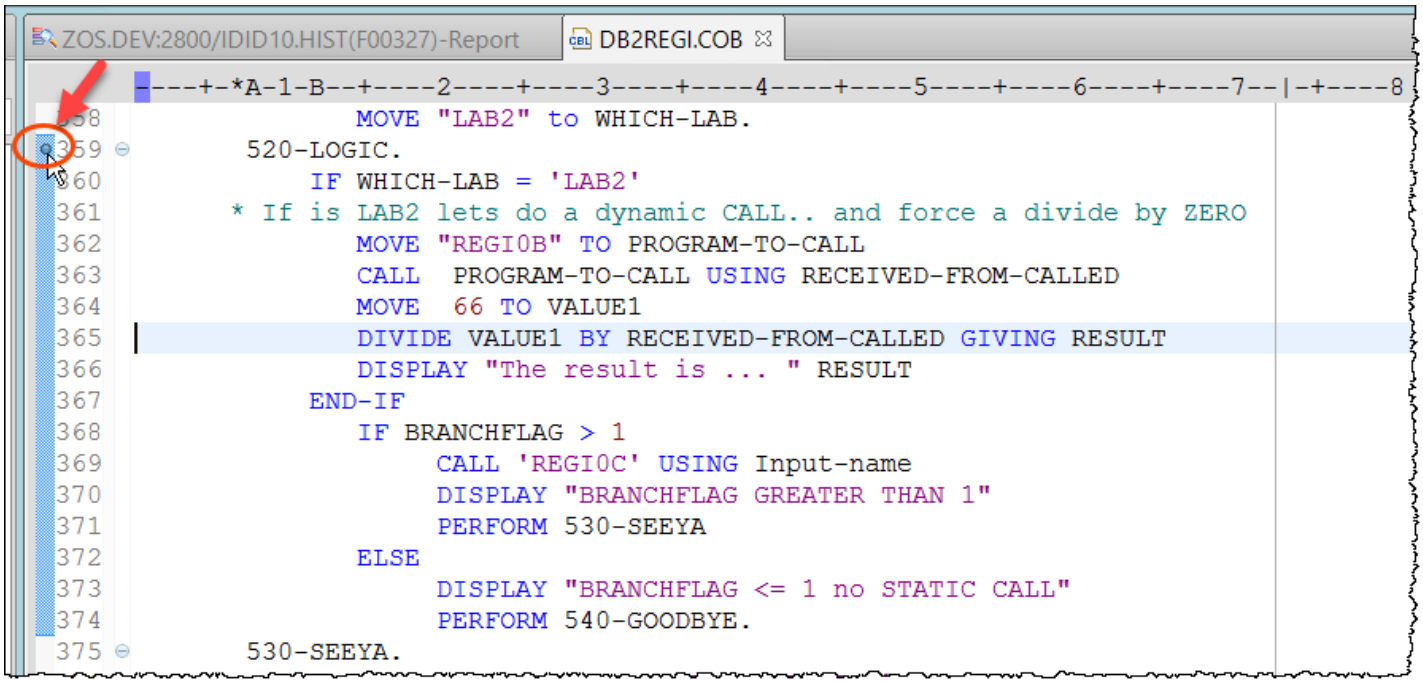
What is *Toggle Paragraph Breakpoint*?

Toggle Paragraph Breakpoint provides the ability to set paragraph breakpoints prior to starting a debug session. Because these breakpoints are set using the original source files, they persist between debug sessions.

On previous versions, breakpoints could only be set at the level of the generated program listing files.

This allows a more natural edit, compile and debug workflow on the original source files.

3.1.12 **Scroll up** a bit. The *Toggle Paragraph Breakpoint* is set at the **520-LOGIC** paragraph. This break point may optionally be used when debugging the COBOL code.



3.1.13 Close all opened editors using **CTRL+ Shift + F4**.
Or just click on the . of each opened editor.

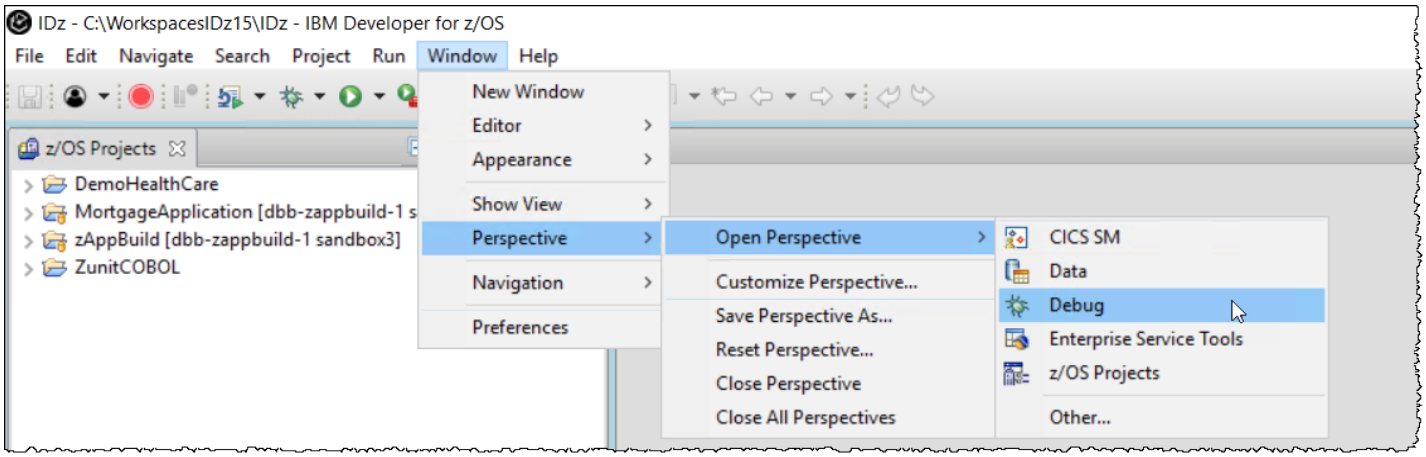
Section 4. Using the IBM z/OS Debugger for a temporary fix

You will use the Debug to verify the ABEND and make a runtime fix.

___ 4.0 Be sure that IDz client is listening on port 8001

Usually this step is NOT required, but in our cloud environment we have timeouts that force ports to be closed.

4.0.1 Go to the *Debug Perspective* by selecting **Windows > Perspective > Open perspective > Debug**



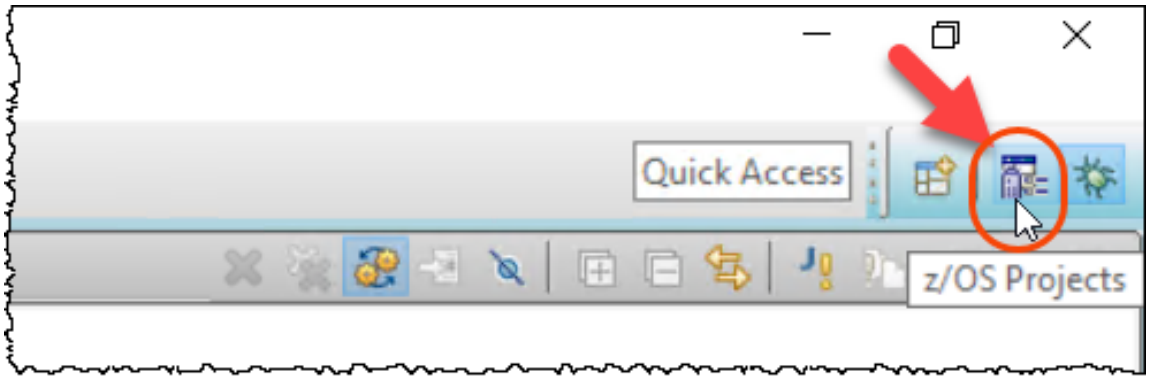
4.0.2 If the icon is **red**, click and select **Start listening on port 8001**

4.0.3 The listening icon will turn green and the IDz client is listening on port 8001.

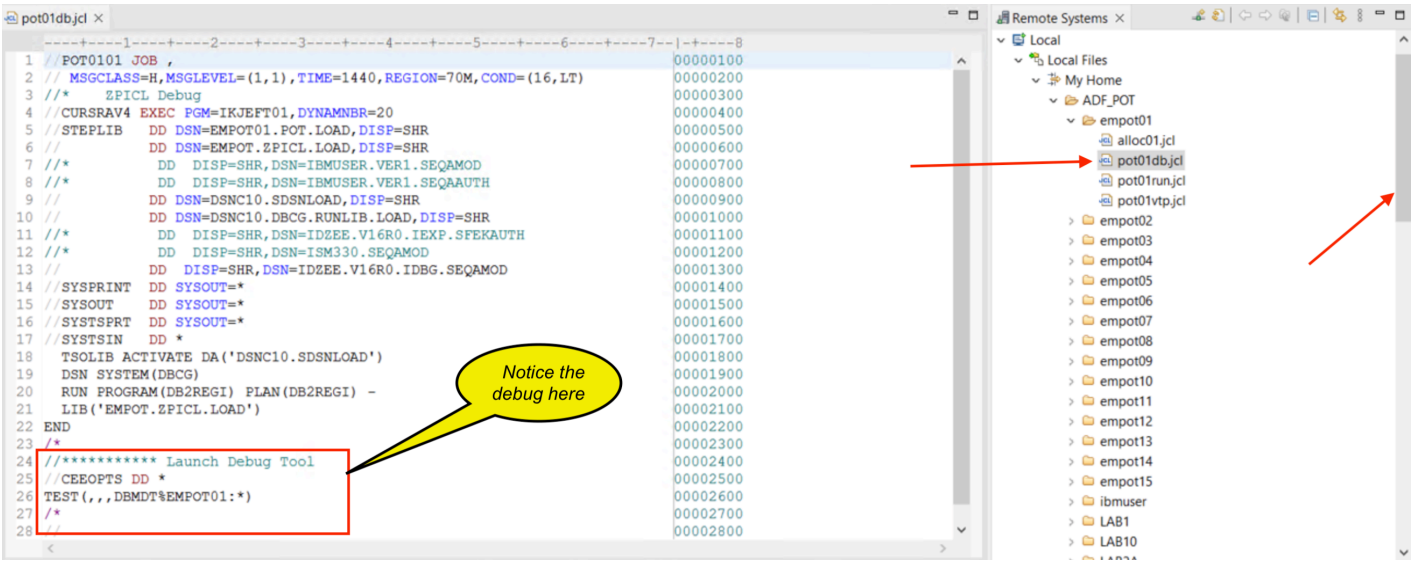
___ 4.1 Submit the JCL to invoke the Debug

You can now submit the JCL to execute again with the Debug option.

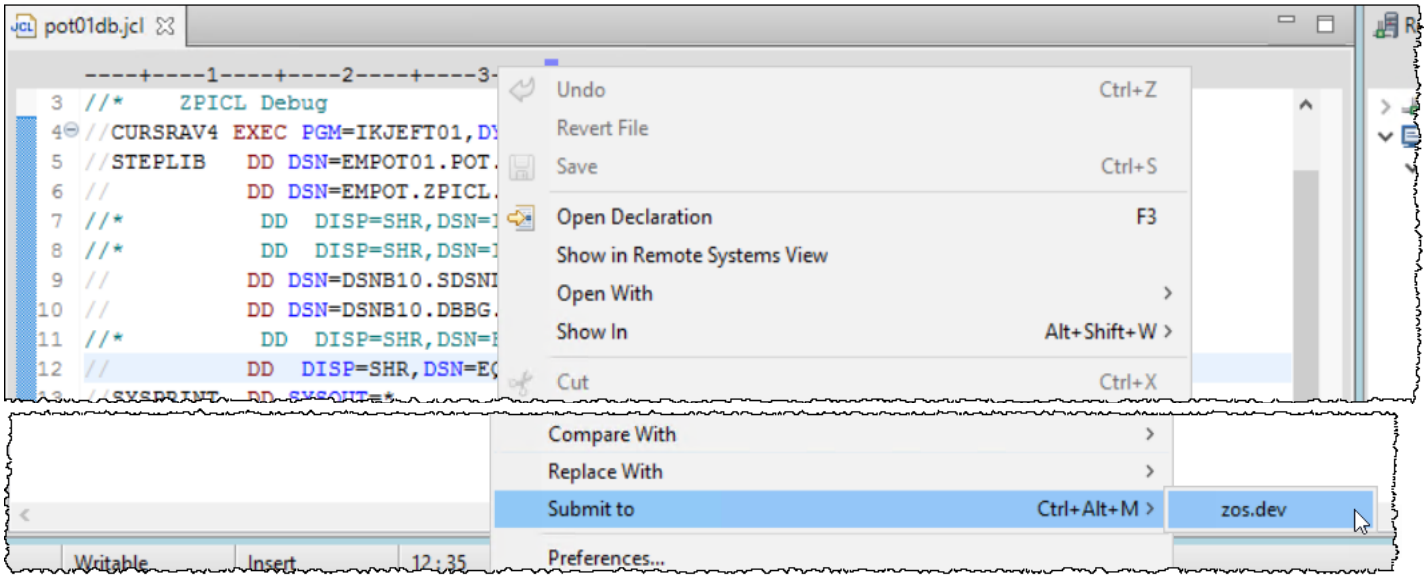
4.1.1 ⏭ Using the top right corner Go to the **z/OS Projects** Perspective by clicking on the icon below.



4.1.2 ⏭ Using *Remote System View,* **scroll up** to locate the file **pot01db.jcl** under **Local/Local Files/ADF_POT/empot01** and **double click** to edit..



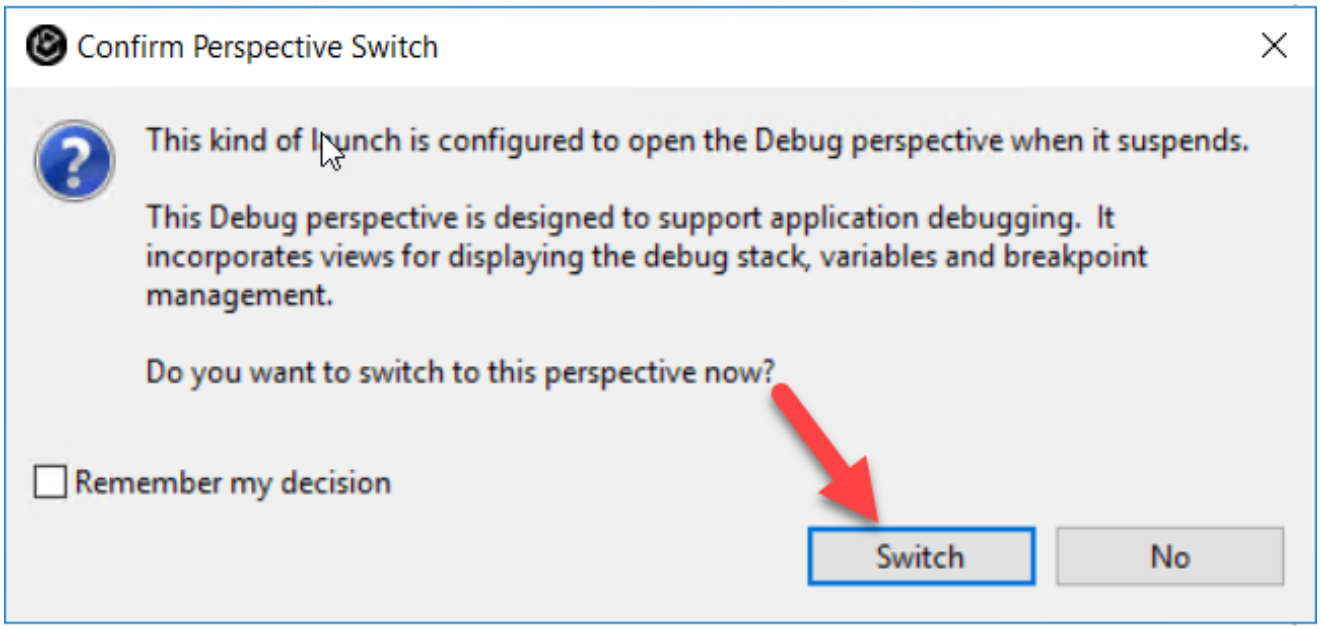
4.1.3 **Right click** on the JCL being edited and select **Submit** to > **zos.dev**



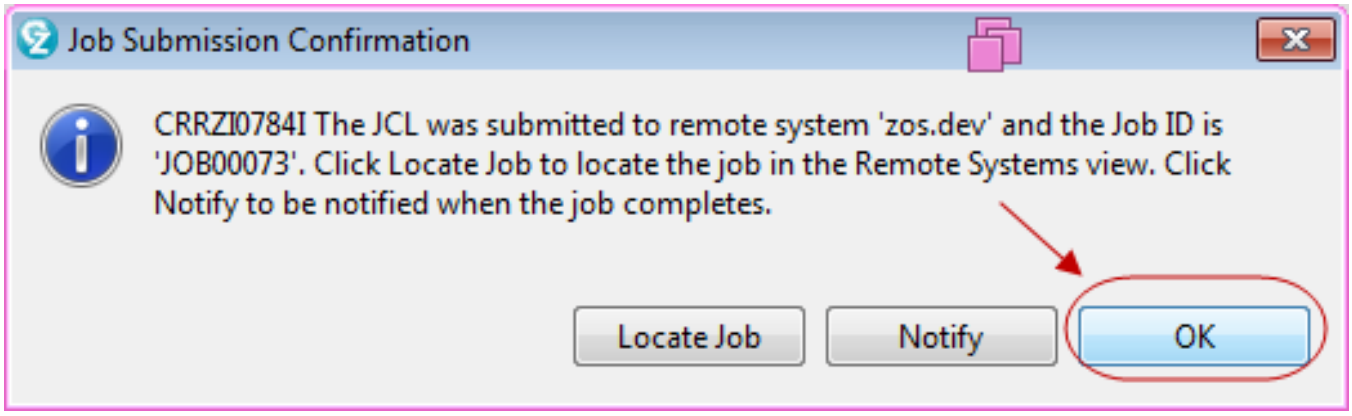
4.1.4 Once the job starts the z/OS debug will "talk" with you. Notice that the communication will be via the IDz connection (RSE), you don't need to specify IP addresses. This may work even when firewalls are in place.

Click **Switch** to open the *Debug* Perspective

Notice: Depending how fast are you the dialogs order here could be different.

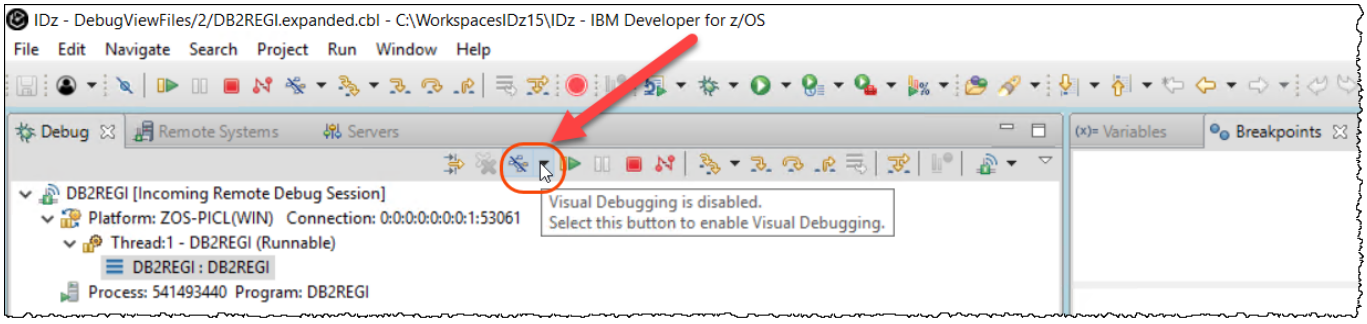


4.1.5 Also click **OK** for this dialog below



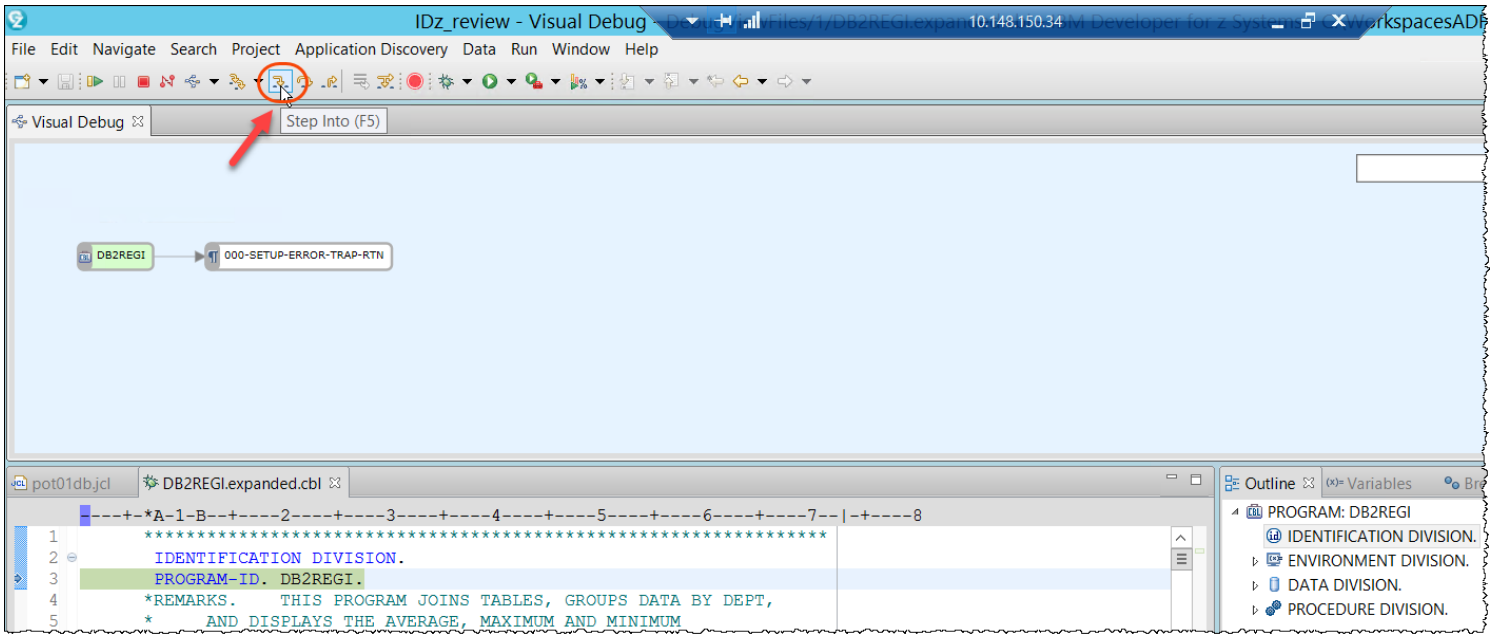
4.1.6 Using the *Debug* perspective:

Click the icon to enable Visual Debugging which shows the ***Visual Debug*** view.
If a dialog pops up click **YES**.



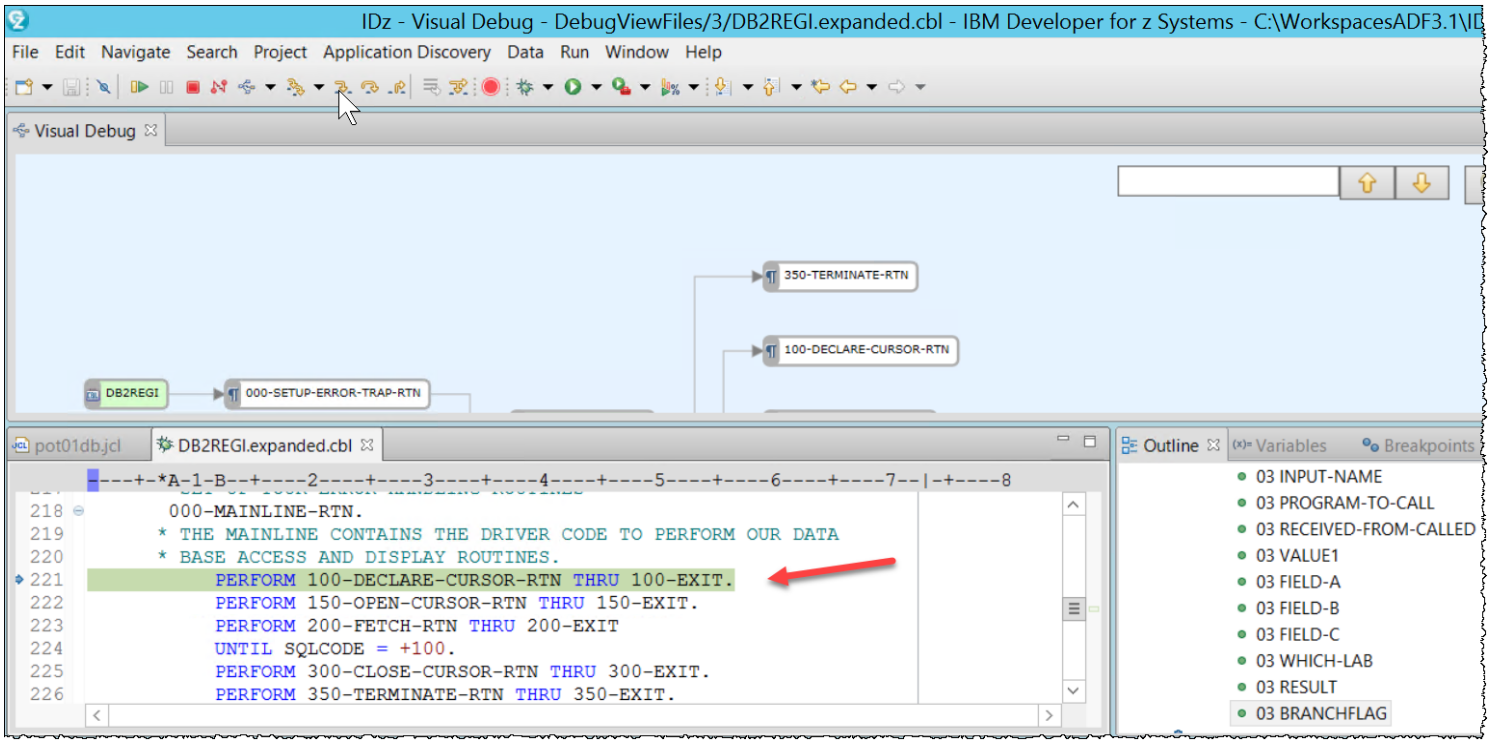
4.1.7 The *Visual Debug* view show the paragraphs being executed (in the top)

Click the icon or **F5** (*Step into*) to execute first line of code.



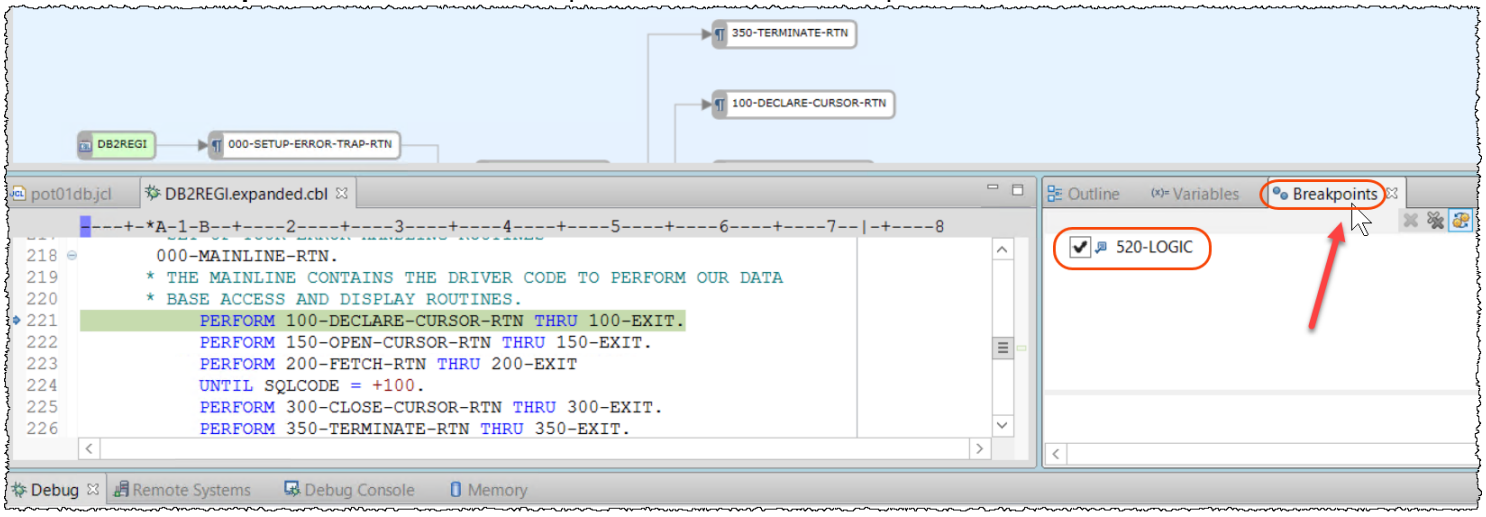
4.1.8 Press F5 or until this the statement

```
PERFORM 100-DECLARE-CURSOR-RTN THRU 100-EXIT
```



4.1.9 On the step 3.1.10 when using the *Fault Analyzer,* you created a *paragraph breakpoint* even without having the execution started. This is handy since it will show the area that needs to be debugged..

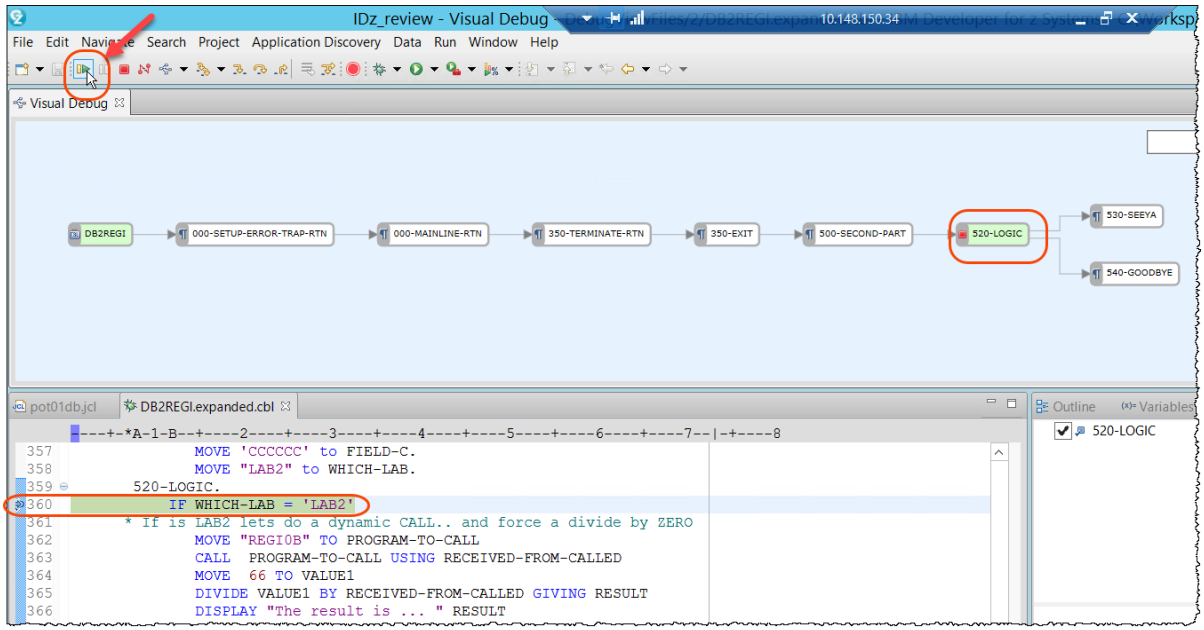
⏭ Click on **Breakpoints** tab to see this breakpoint created before on step 3.1.10.



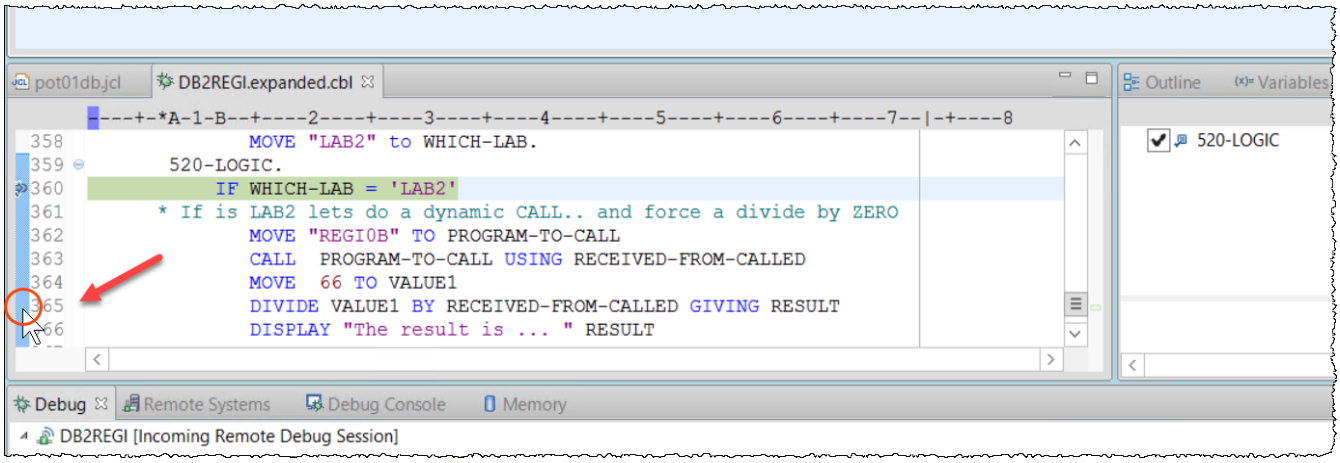
What is the Visual Debugging?

Visual debugging allows you to interact with your COBOL or PL/I debug session using the program control flow diagram.
With this diagram, you can visualize the stack trace, set breakpoints, and run to a selected call path. In COBOL, the stack trace represents the paragraph call chain.
In PL/I, the stack trace represents the procedure call chain.

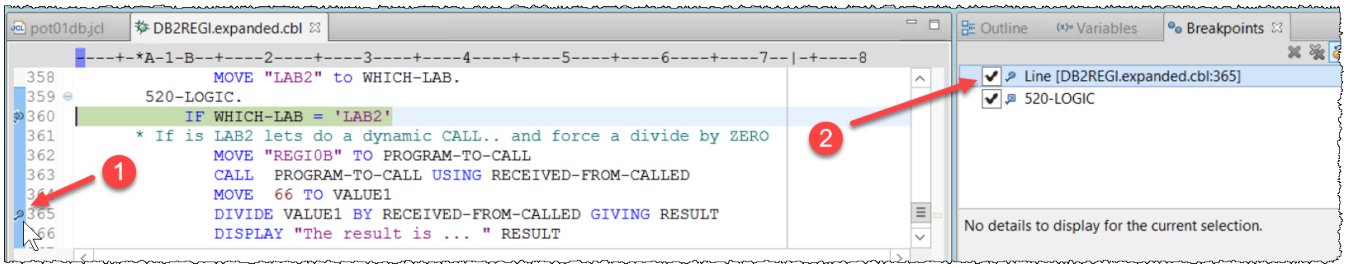
4.1.10 Click on or press **F8** and notice that the execution will stop on the line since a *Paragraph Entry Breakpoint* was created before. This line is about to be executed.



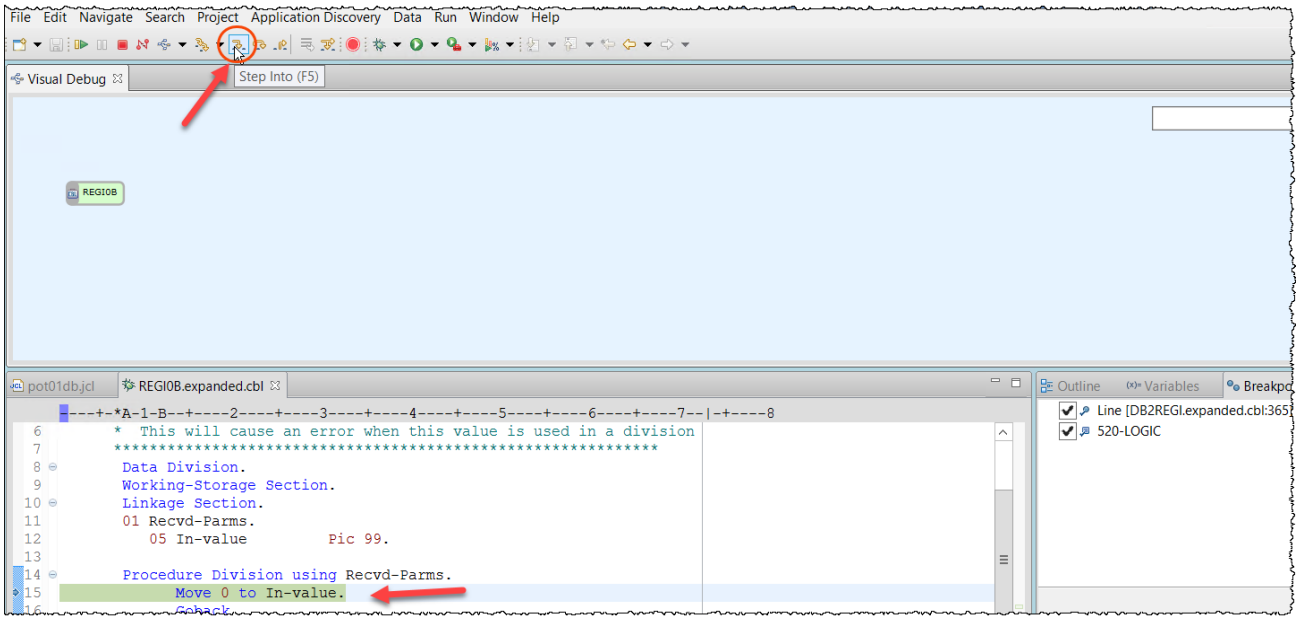
4.1.11. As you verified using Fault Analyzer, the abend occurred on line " **DIVIDE VALUE1 BY RECEIVED-FROM-CALLED ..**" where you had a divide by zero
(see step 3.1.10). Now you will add a breakpoint on this line and change the values to avoid the abend.
On the COBOL editor move the mouse to the **blue column** on left and **double click** on the line
`DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT` to create a breakpoint.



4.1.12 Notice that a small circle is shown on the left of line **" DIVIDE VALUE1 BY RECEIVED-FROM-CALLED .. "** .



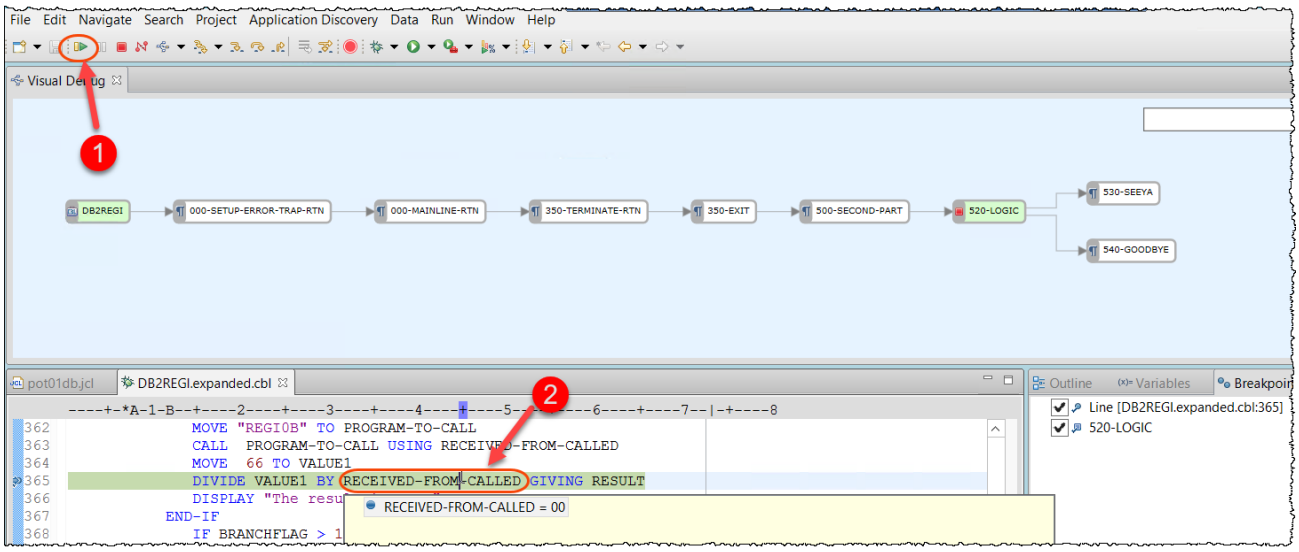
4.1.13 Continue by clicking on or using **F5** until you will see that a program named **REGI0B** is called and this program is returning a value of 0.



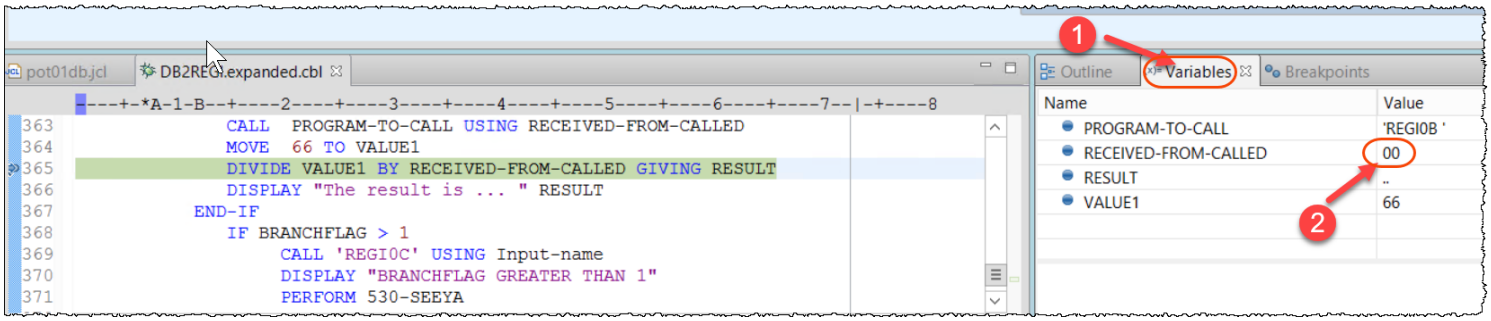
4.1.14 1 Click on or press **F8.**
The execution will stop at the breakpoint that you created (line **DIVIDE VALUE1 BY RECEIVED-FROM-CALLED**

….).

2 Move the cursor to **RECEIVED-FROM-CALLED** variable and click to see the **00** value. .
(Tip: under the cloud instance the behavior may act different and you may need to click on the editor area before moving the mouse to that field).

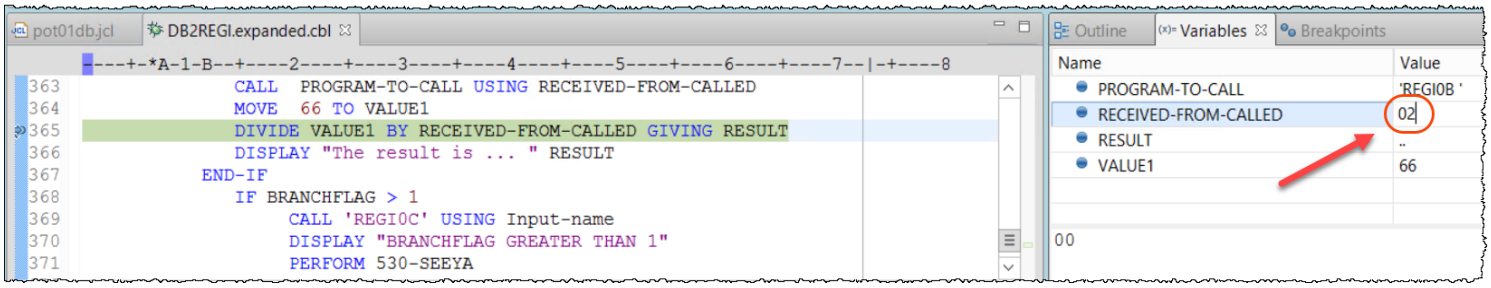


4.1.15 1 Click the **Variables** tab (right) and 2 verify that **RECEIVED-FROM-CALLED** is **00**.

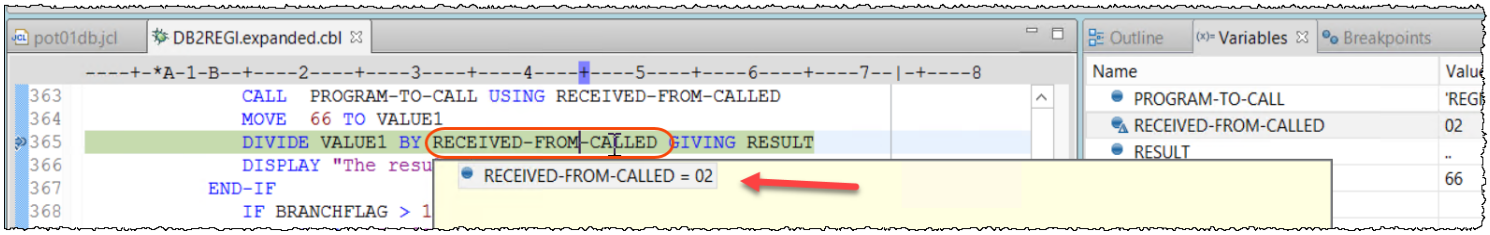


4.1.16 This is the abend cause. You must change the value to something different than 0.

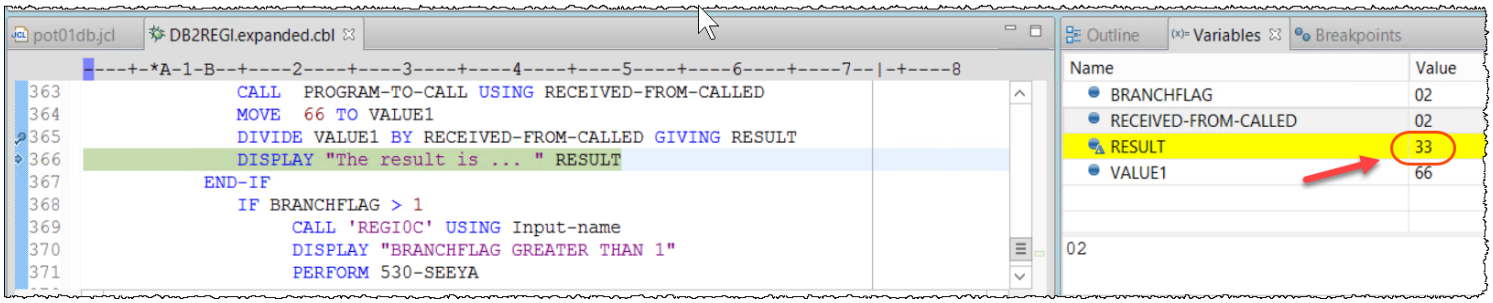
Click on **RECEIVED-FROM-CALLED** value of **00** and modify to **2**, just overtyping and press **enter.**



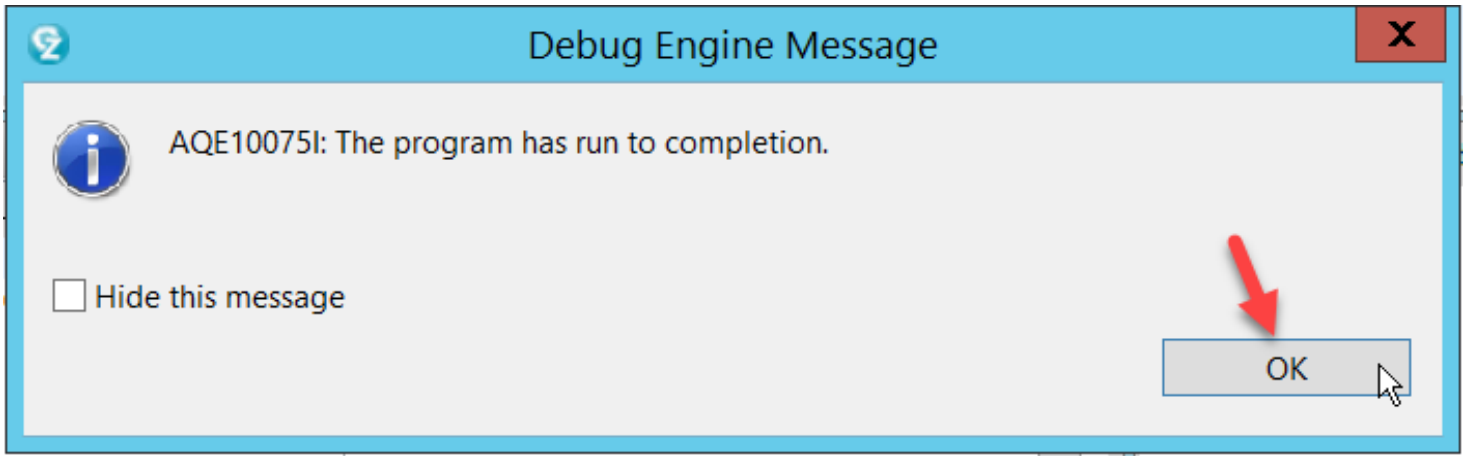
4.1.17 Again, move the cursor to **RECEIVED-FROM-CALLED** variable and now see the value **02**.
(Tip: under the cloud instance the behavior may act different and you may need to click on the editor area before moving the mouse to that field).



4.1.18 Click the icon (Step into) or press **F5** to see the next line being executed
This time the divide by 2 give you a result of **33.**



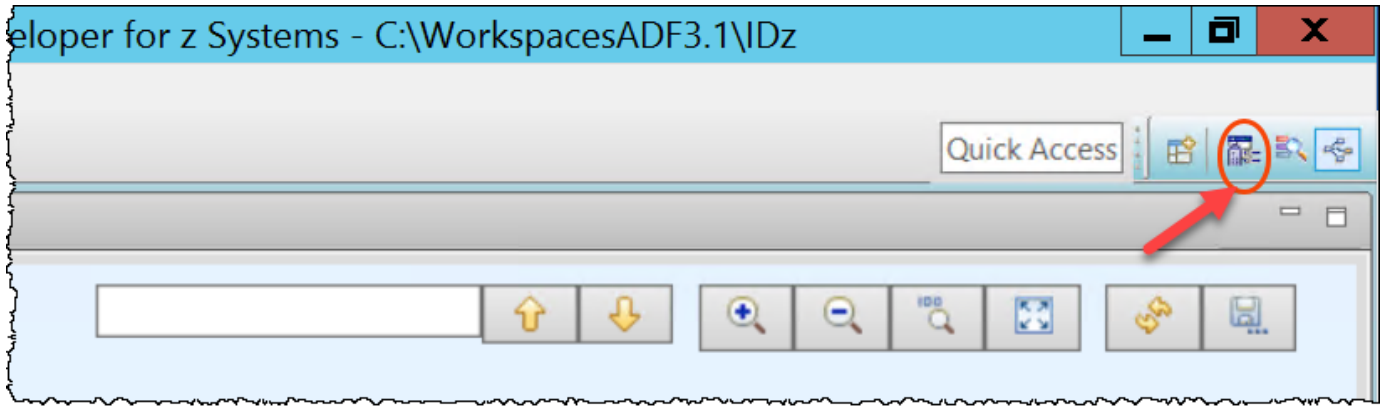
4.1.19 Click on **(or F5)** few times
and **Resume** (**F8**) when you are satisfied so the program will execute until the end.



4.1.20 Click **OK** to close the dialog above.

To fix this bug definitely you will modify the program **REGI00B** that is returning 0 to be used in a division.
You will do that later. For now, optionally you can play again with the Debugger and use the Visual Debugger.
Or continue after the optional steps.

4.1.21 Go back to the **z/OS Projects Perspective** by clicking on the icon the top right corner.



___ 4.2 Using the Visual Debugger for Stack pattern breakpoints

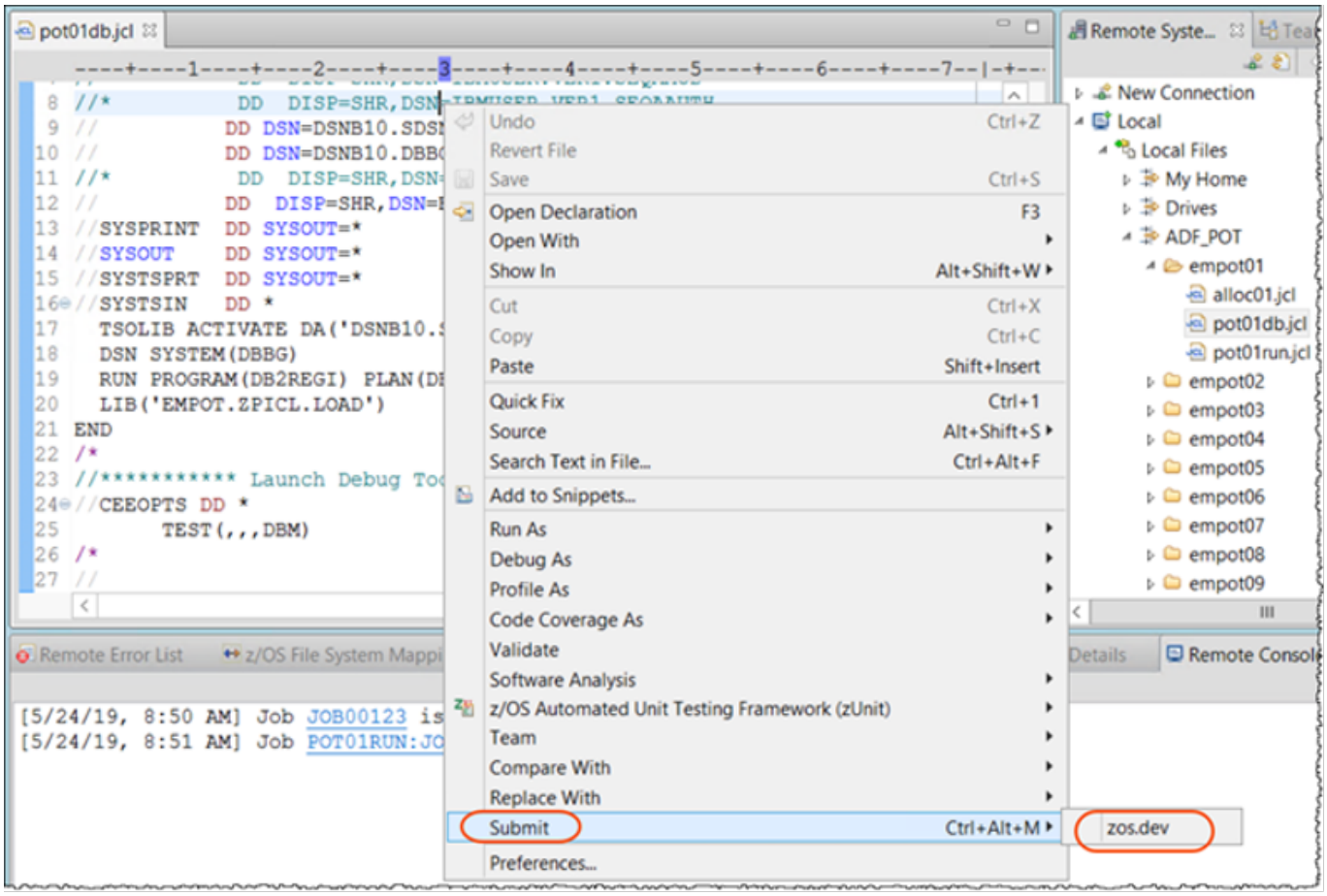
If you are not running late and interested on this subject, execute the steps below, **otherwise jump to 4.3**

Stack pattern breakpoint

A stack pattern breakpoint is a special type of conditional breakpoint. With this feature, you can specify that you only want to stop at a location when the current stack trace contains a predefined stack pattern.

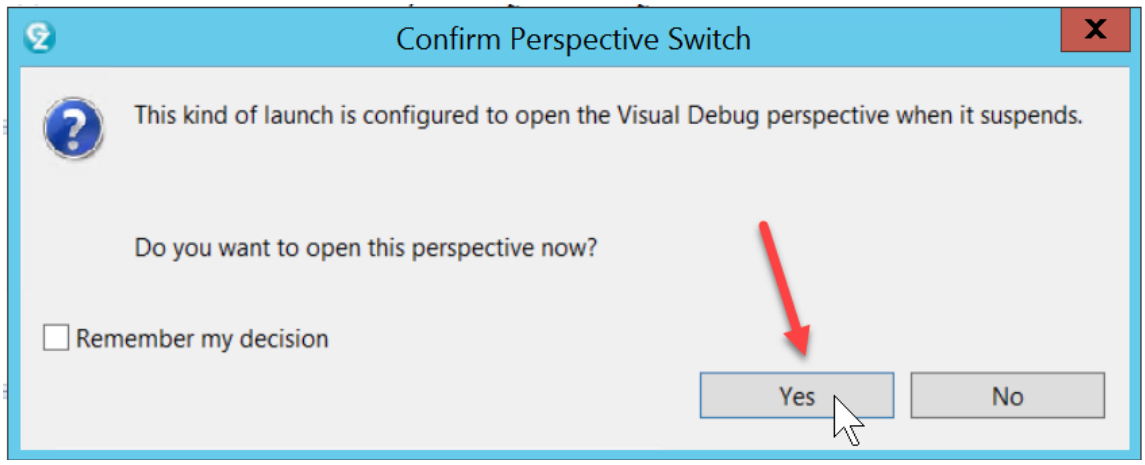
Visual debugging supports a stack pattern integration feature, which allows you to select a connected path from the program control flow diagram and set a stack pattern breakpoint using the selected path as a stack pattern.

4.2.1 **Right click** on the JCL being edited and select **Submit** > **zos.dev**

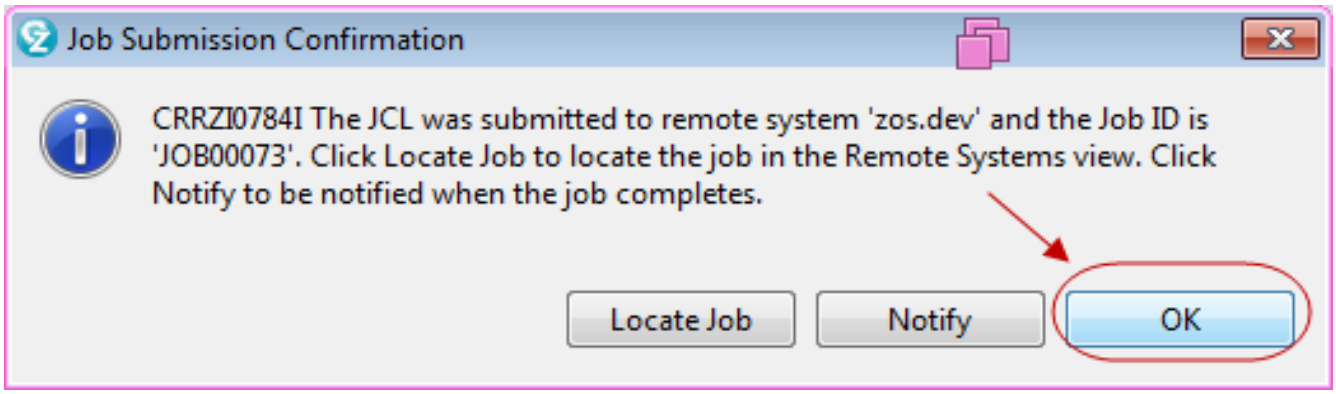


4.2.2 Once the job starts the z/OS debug will “talk” with you.

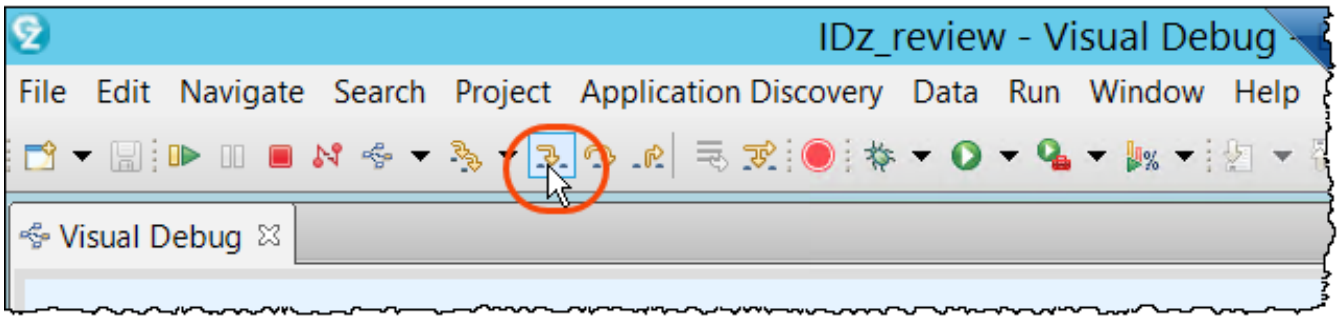
⏭ Click **Yes** to open the *Visual Debug* Perspective
(depending on the z/OS the order here could be the next step first).



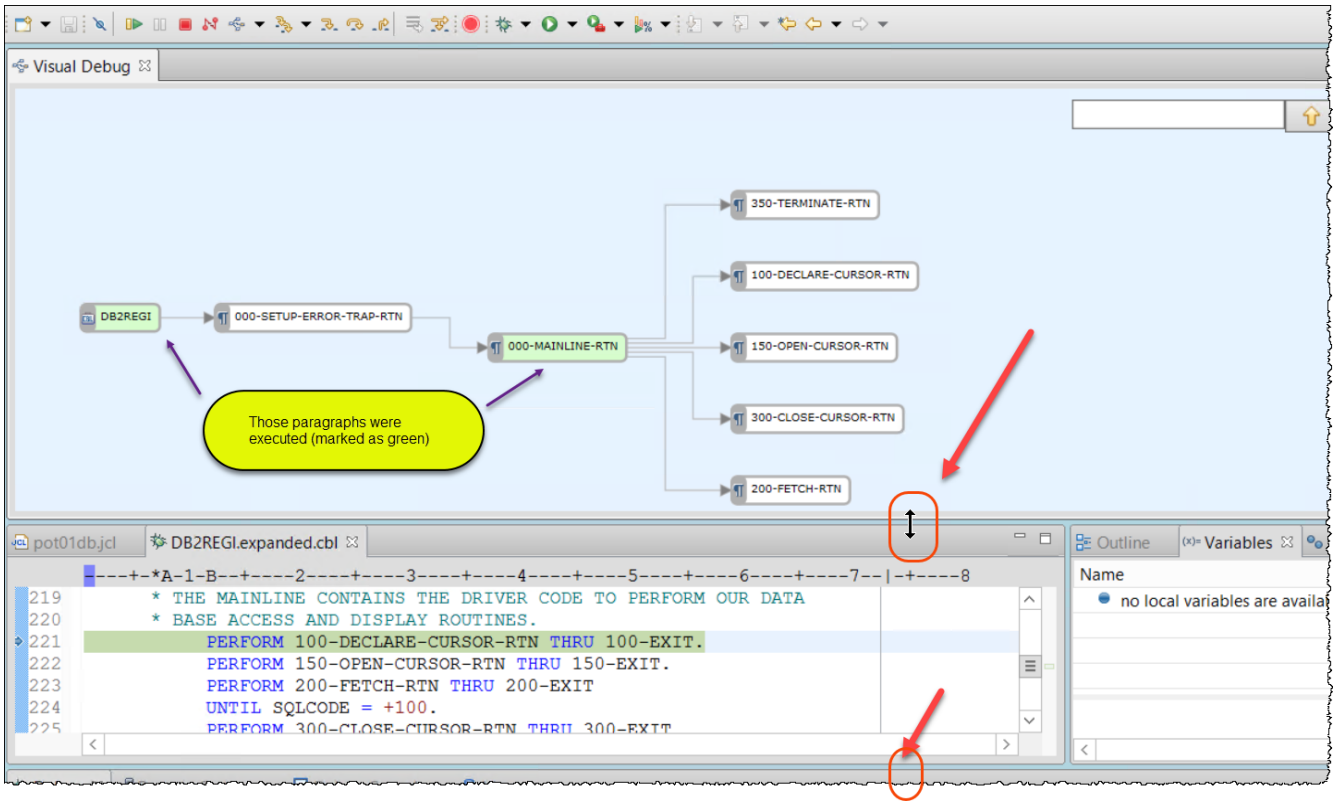
4.2.3 Also click **OK** for this dialog below



4.2.4 Using the *Visual Debug* perspective:
Click on **icon** (or Press **F5**)



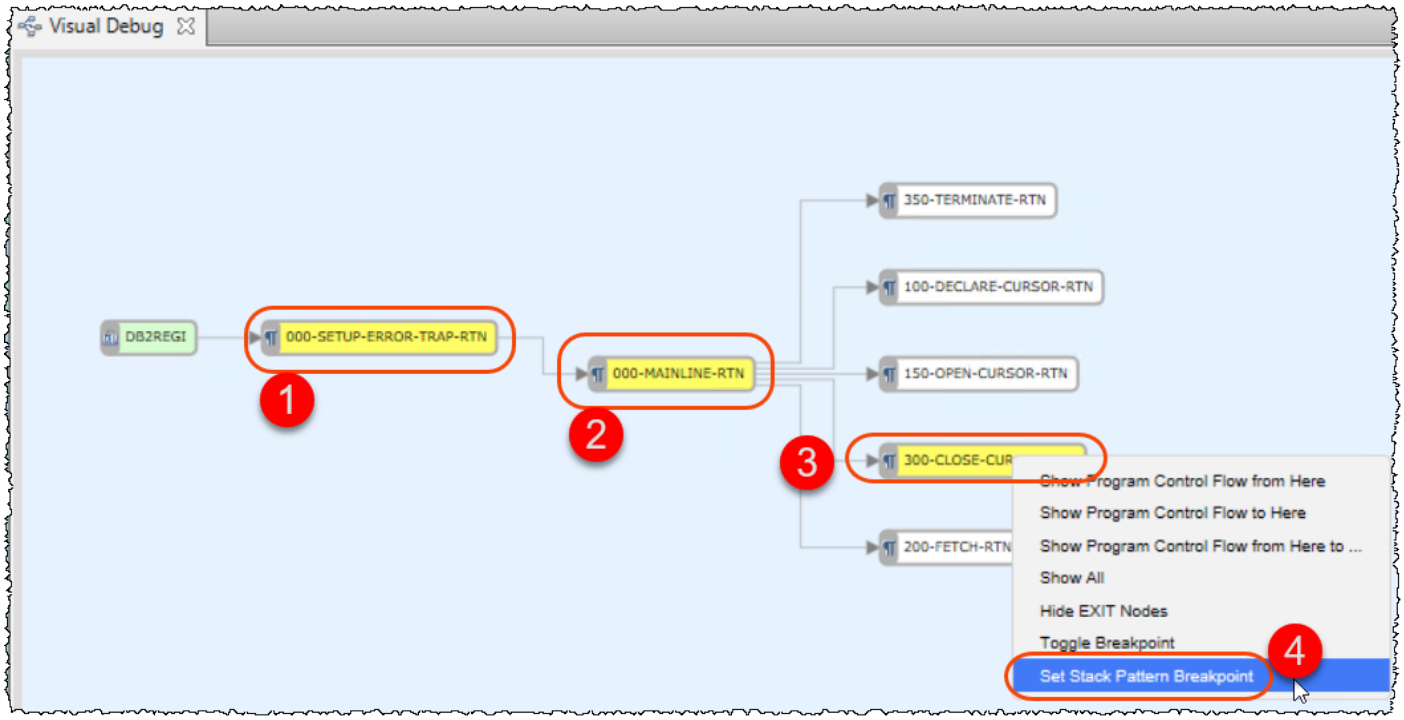
4.2.5 Resize the **Visual Debug** view and you will notice that the paragraphs executed are marked as green



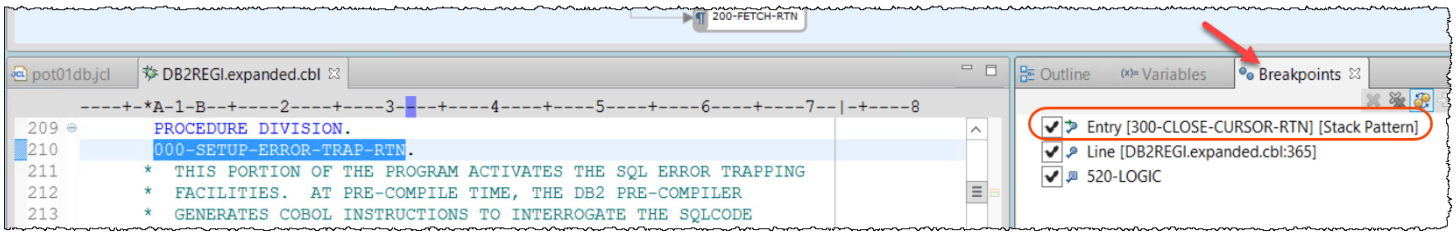
4.2.6 Using the *Visual Debug* view, **scroll down**

Press **CTRL** Key and select the 3 paragraphs as shown below, right click and choose **Set Stack Pattern Breakpoint**

This breakpoint that will happen when the path selected happens.

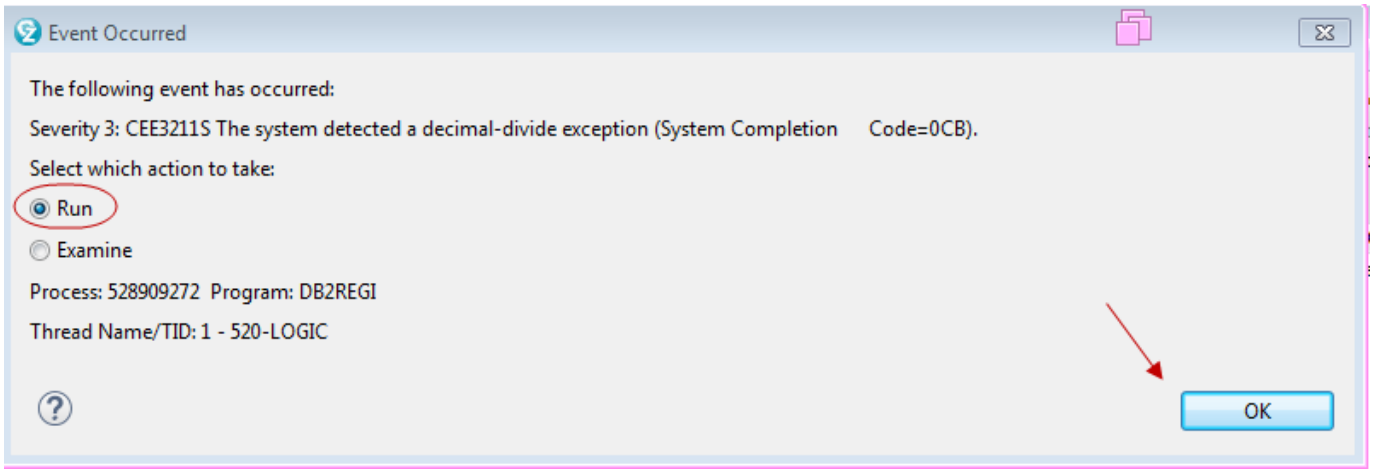


4.2.7 Click on **Breakpoints** view and you will see this breakpoint created.



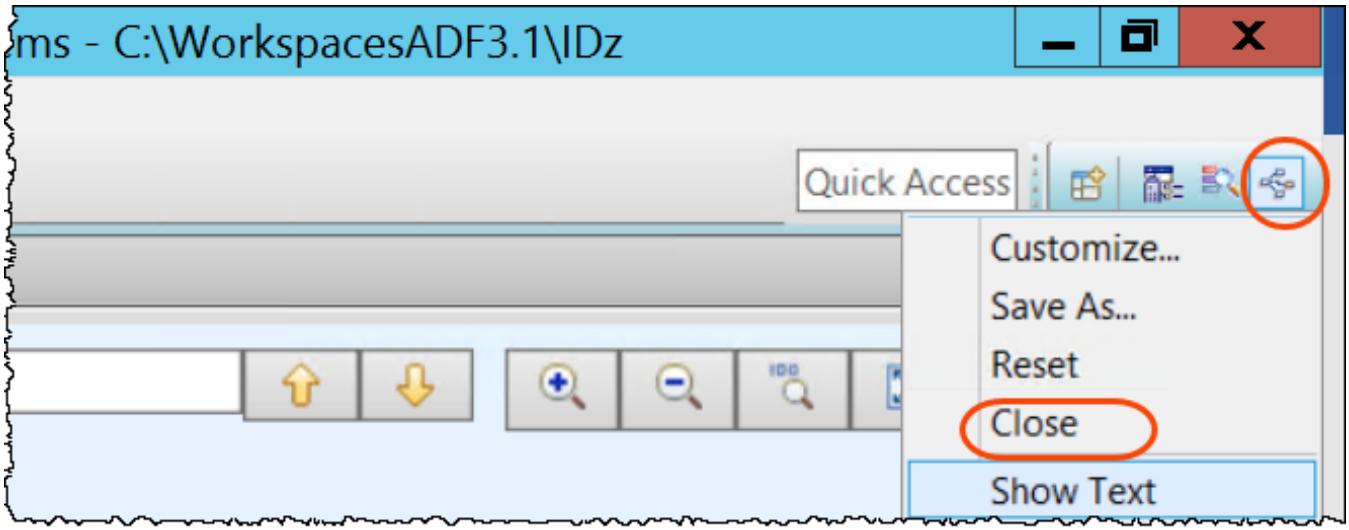
4.2.8 Click on (or press **F8)** few times to resume the execution

4.2.9 Select **Run** and click **OK** to continue

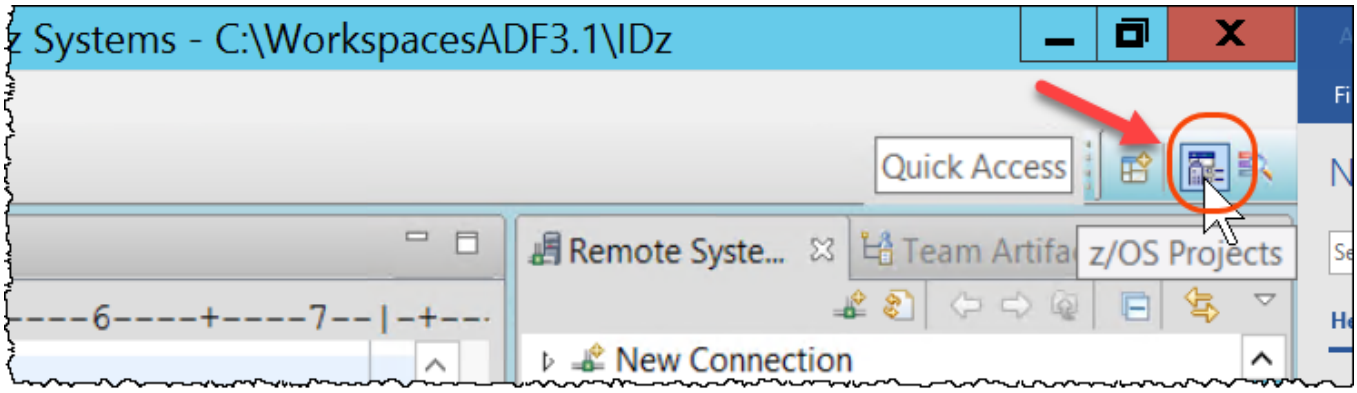


Notice – This breakpoint did not happen since the execution did not execute the 3 selected pargaraphs.

4.2.10 **Close** the *Visual Debug* perspective.



4.2.11 Go back to the **z/OS Projects Perspective** by clicking on the icon the top right corner.

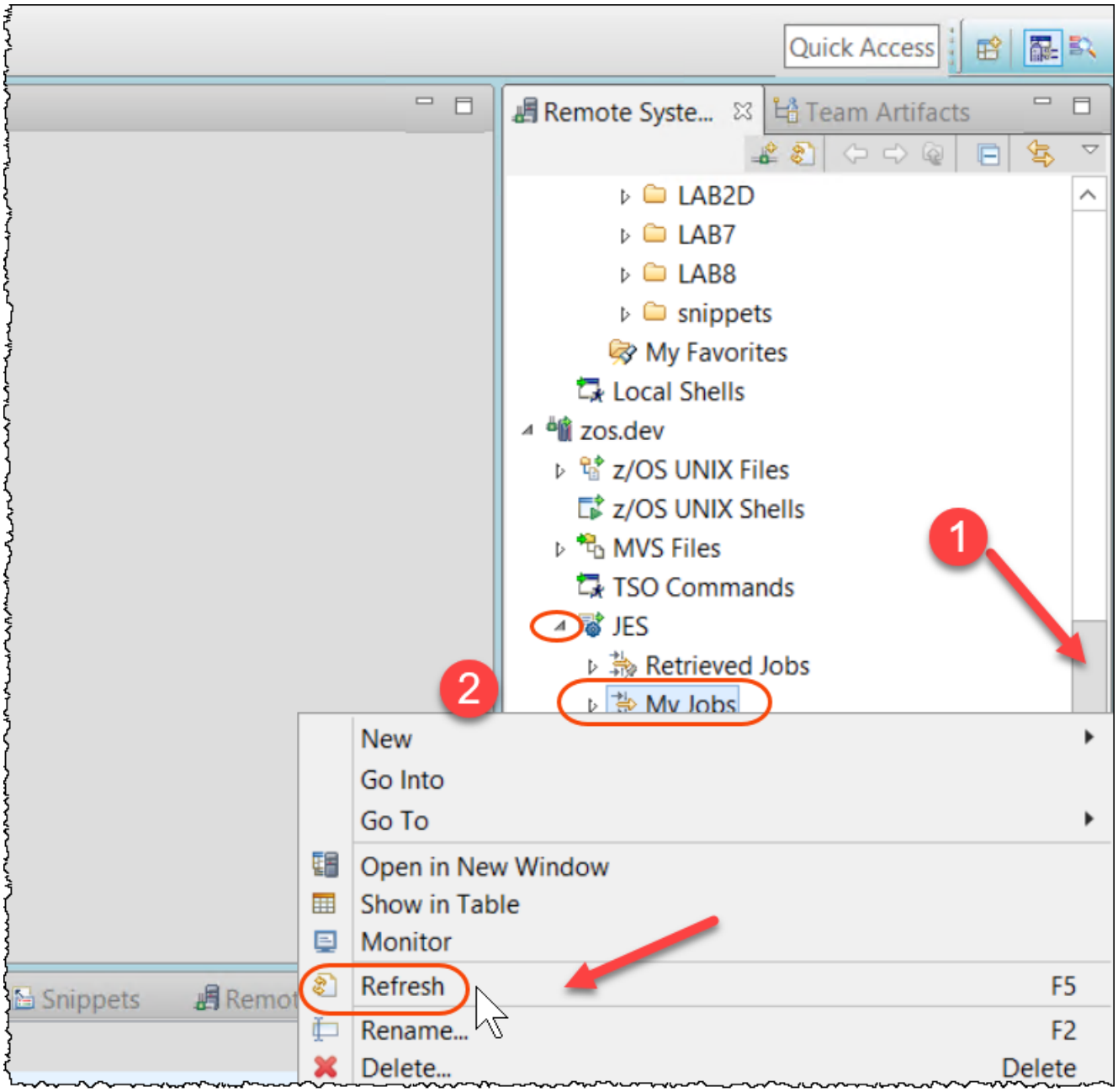


___ 4.3 Accessing the z/OS JES

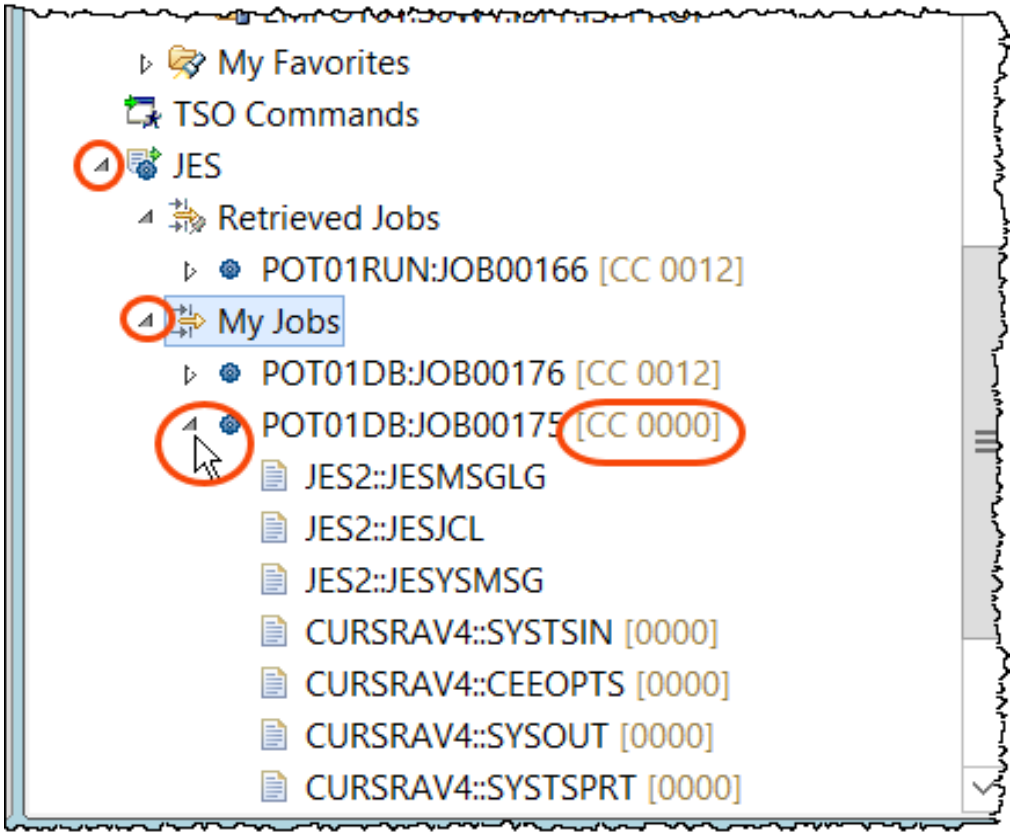
On ISPF many people use *SDSF* for this activity.
You will use the Job Monitor subsystem part of the host components.

4.3.1 Use **Ctrl + Shift + F4** to close all opened editors. Do not save any changes.
Or just click on the . of each opened editor.

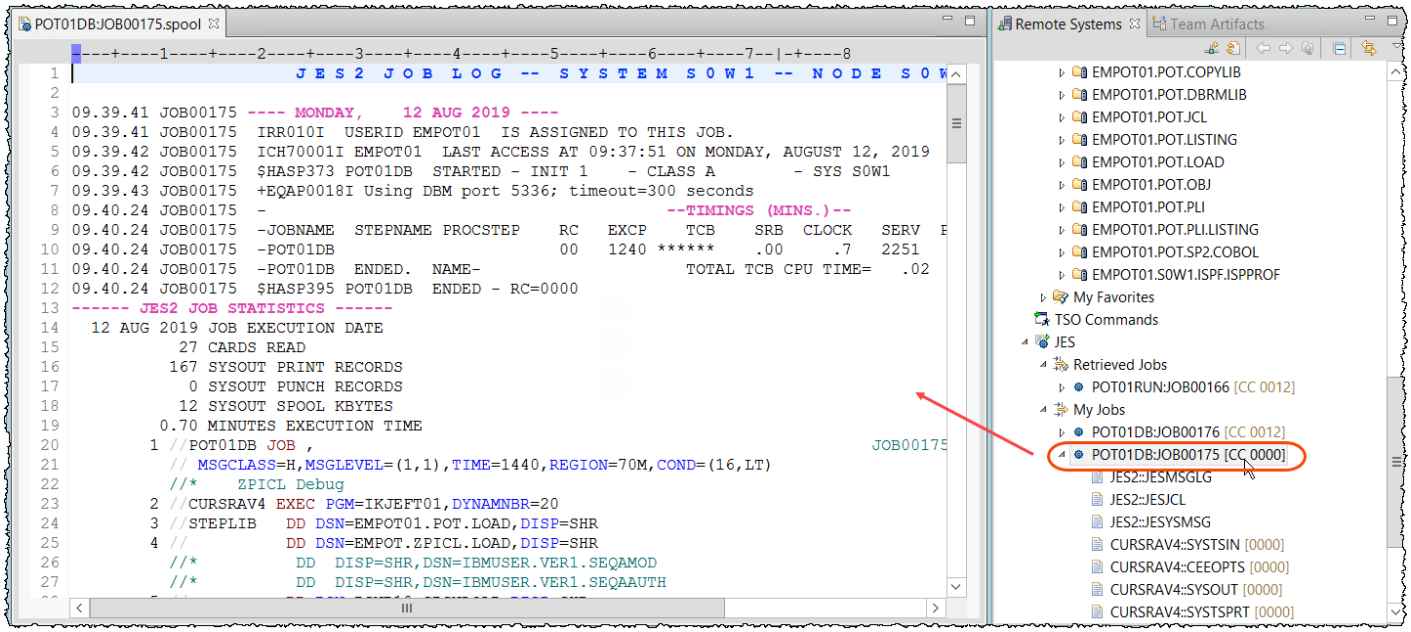
4.3.2 Using the *Remote Systems* view scroll down, expand JES right click on **My Jobs** and select **Refresh**



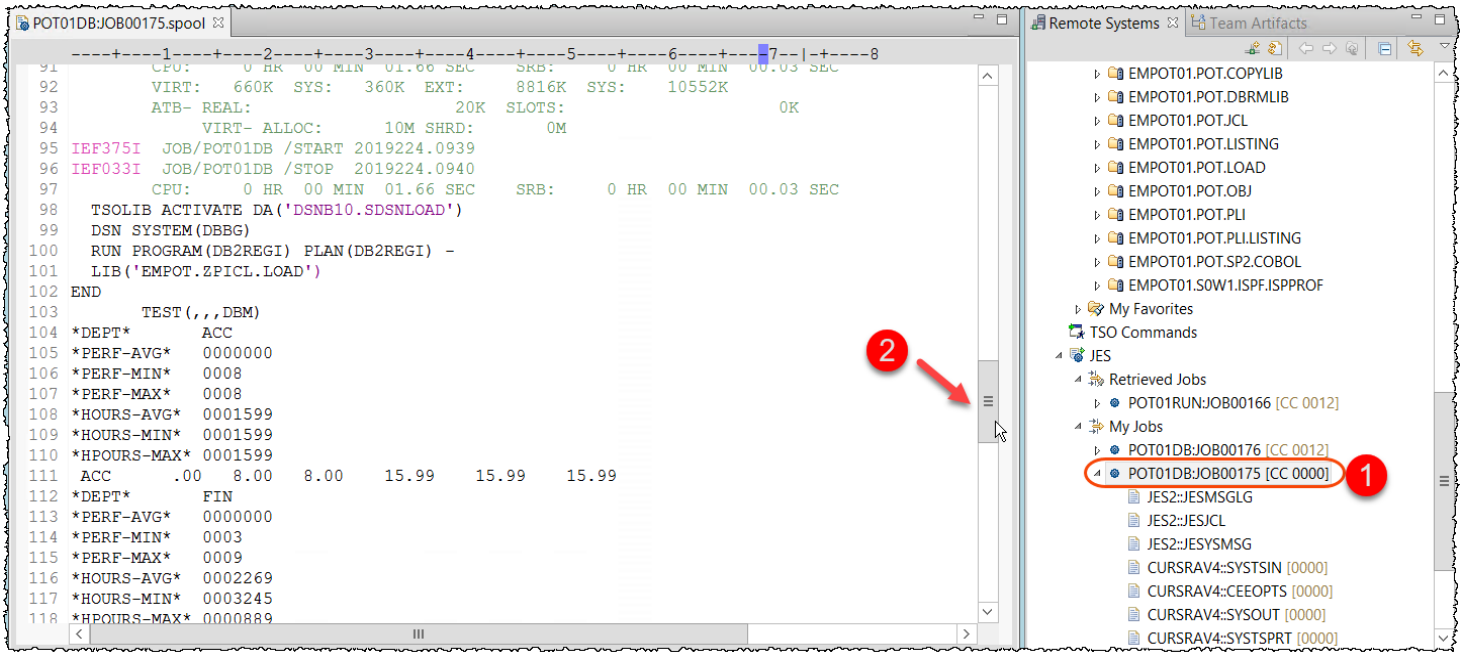
4.3.3 ⏭ Expand **My Jobs** and the Job **POT01DB:JOB00xxx** that has the CC 0000 (it will be the latest if you did not make the optional step)



4.3.4 ⏭ On *Remote Systems* view, **double click** on the **POT01DB:JOB00xxx**. that has the CC 0000



4.3.5 **Scroll down** to see the various JOB steps. Notice that a small report has been printed.



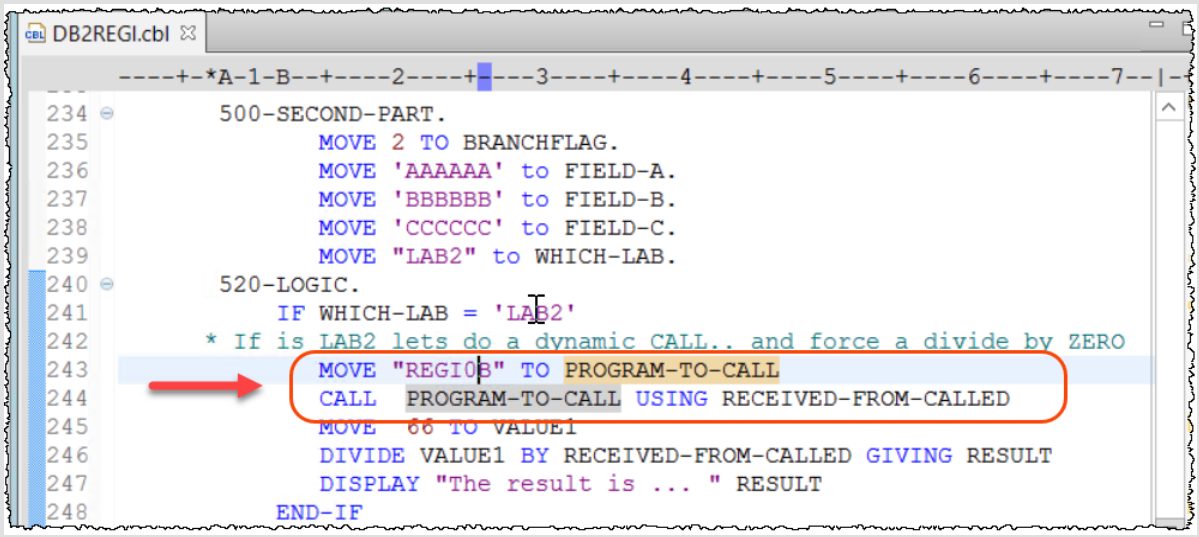
4.3.6 **Close** all opened editors. (**Ctrl + Shift + F4**). Click **No** if asked to save change changes.
Or just click on the . of each opened editor.

Section 5. Modify the COBOL code to fix the bug

You will work with a z/OS member using the editor and now we will be working with the assets located on the z/OS remote system.

What z/OS remote assets you will work with?

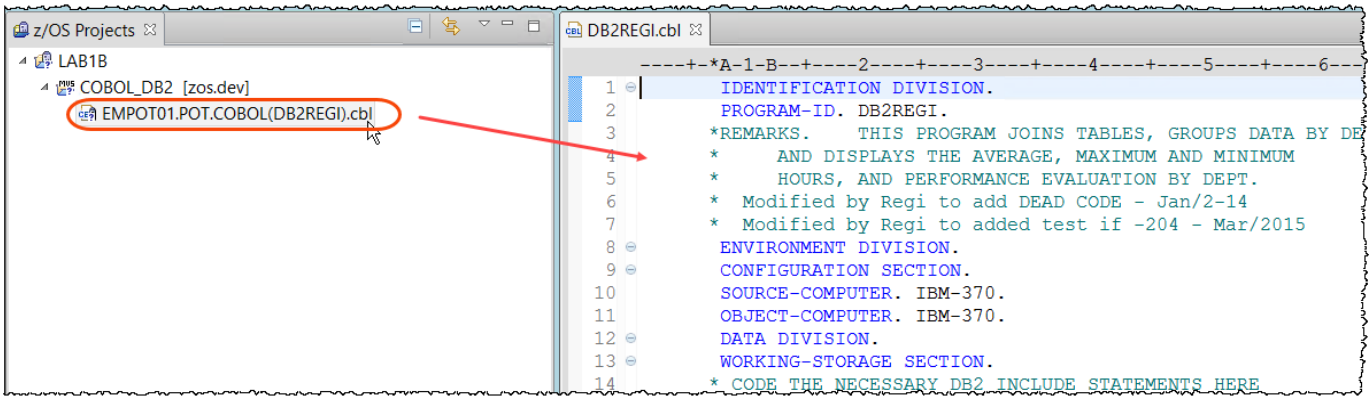
This is a batch program that reads DB2. In addition, this program does a Dynamic call to another COBOL program named **REGI0B**.



___ 5.1 Using the Editor with z/OS COBOL programs

Before fixing the code let's explore some editor capabilities on the main DB2 COBOL program.

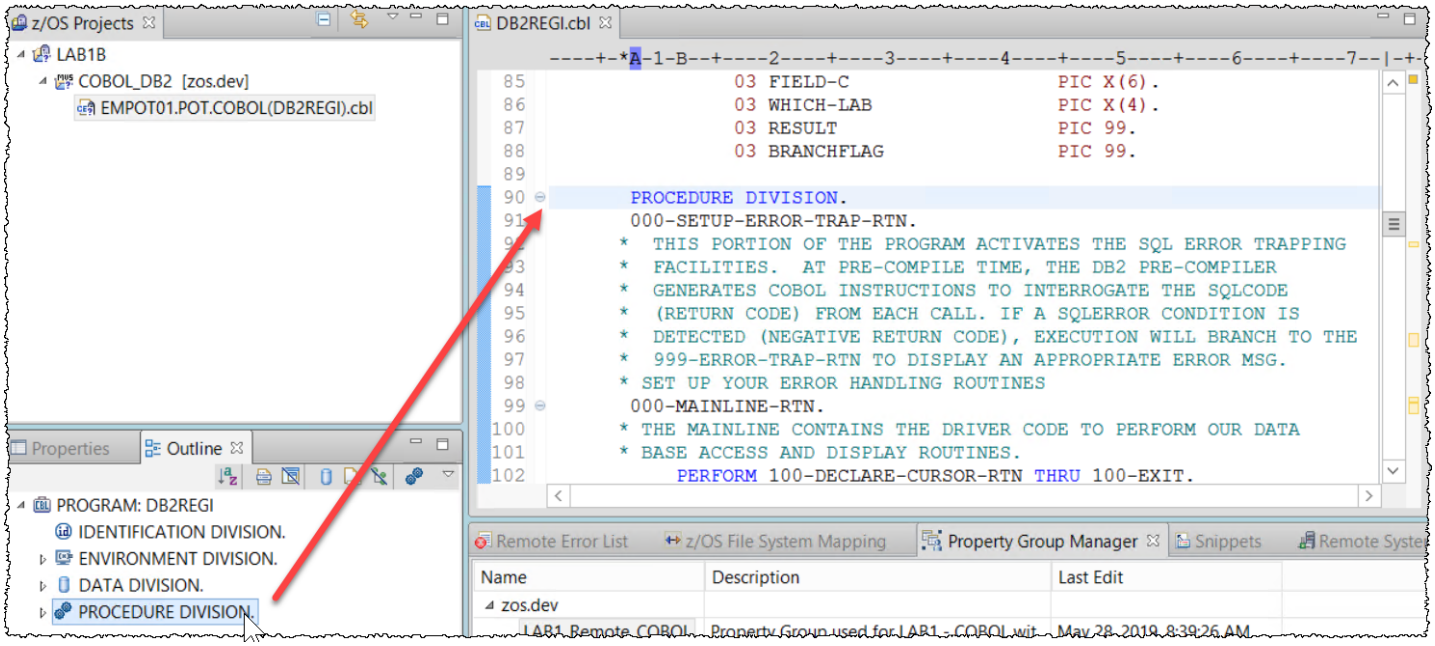
5.1.1 Using the *z/OS Projects* perspective and the *z/OS Projects* view, **double click** on **EMPOT01.POT.COBOL(DB2REGI)** to open the file using the editor.



5.1.2 ⏭ Click on the **Outline** tab (bottom and left) to see the *Outline* view.

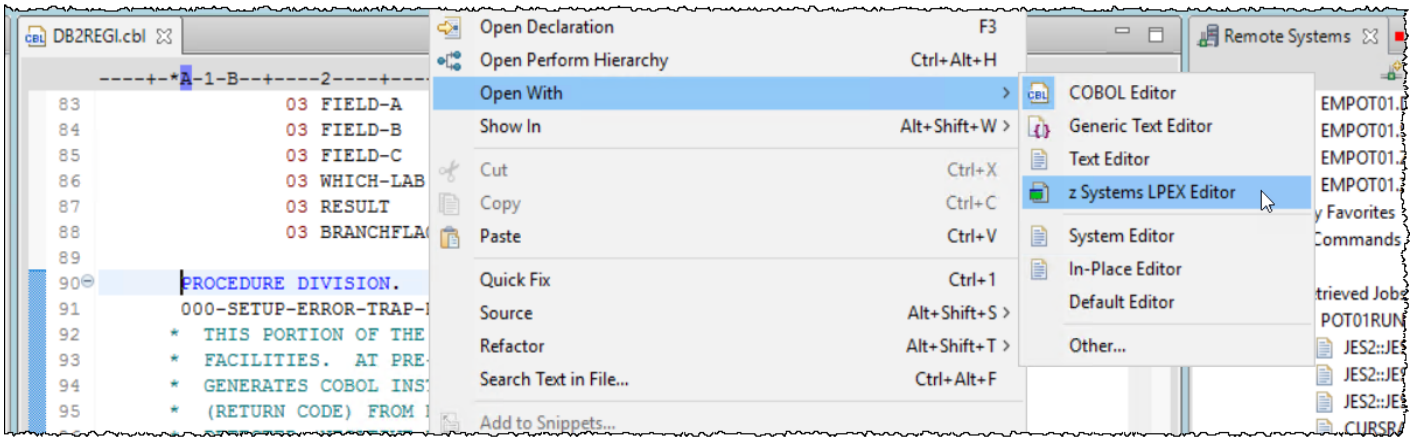
⏭ Using the editor, browse the program and notice that the contents of the *Outline view* are synchronized with the COBOL source code and vice versa.

⏭ Click on the **PROCEDURE DIVISION.** Notice that you can navigate using the *Outline* view.

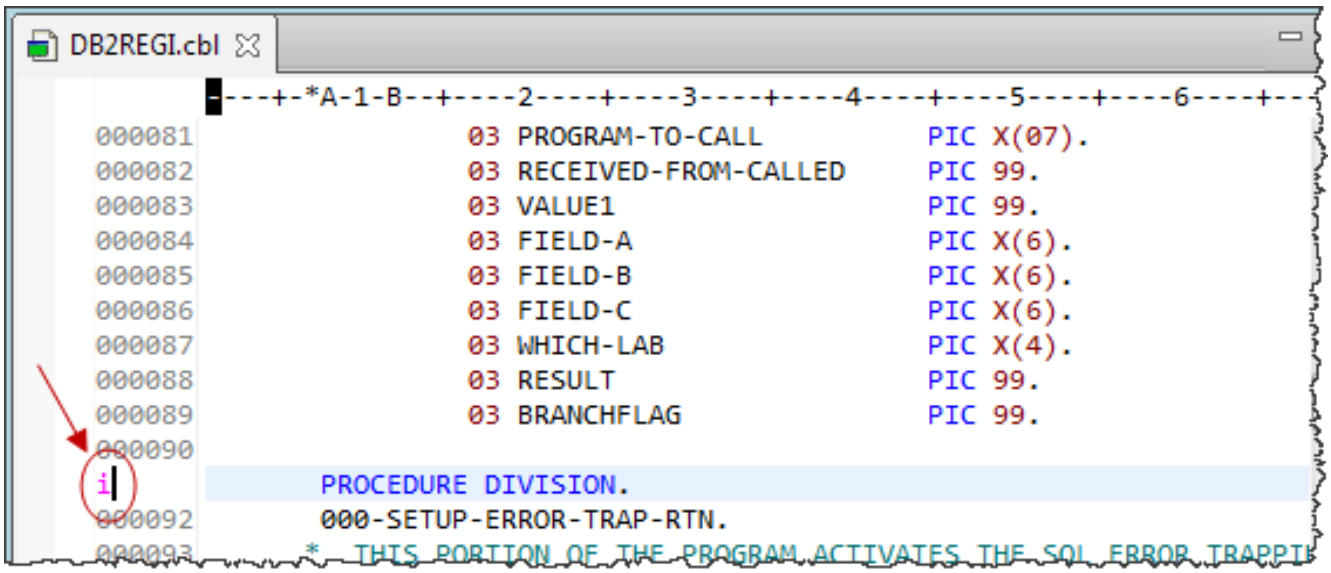


5.1.3 If you prefer the ISPF editor, you might use another IDz editor called “LPEX editor”.

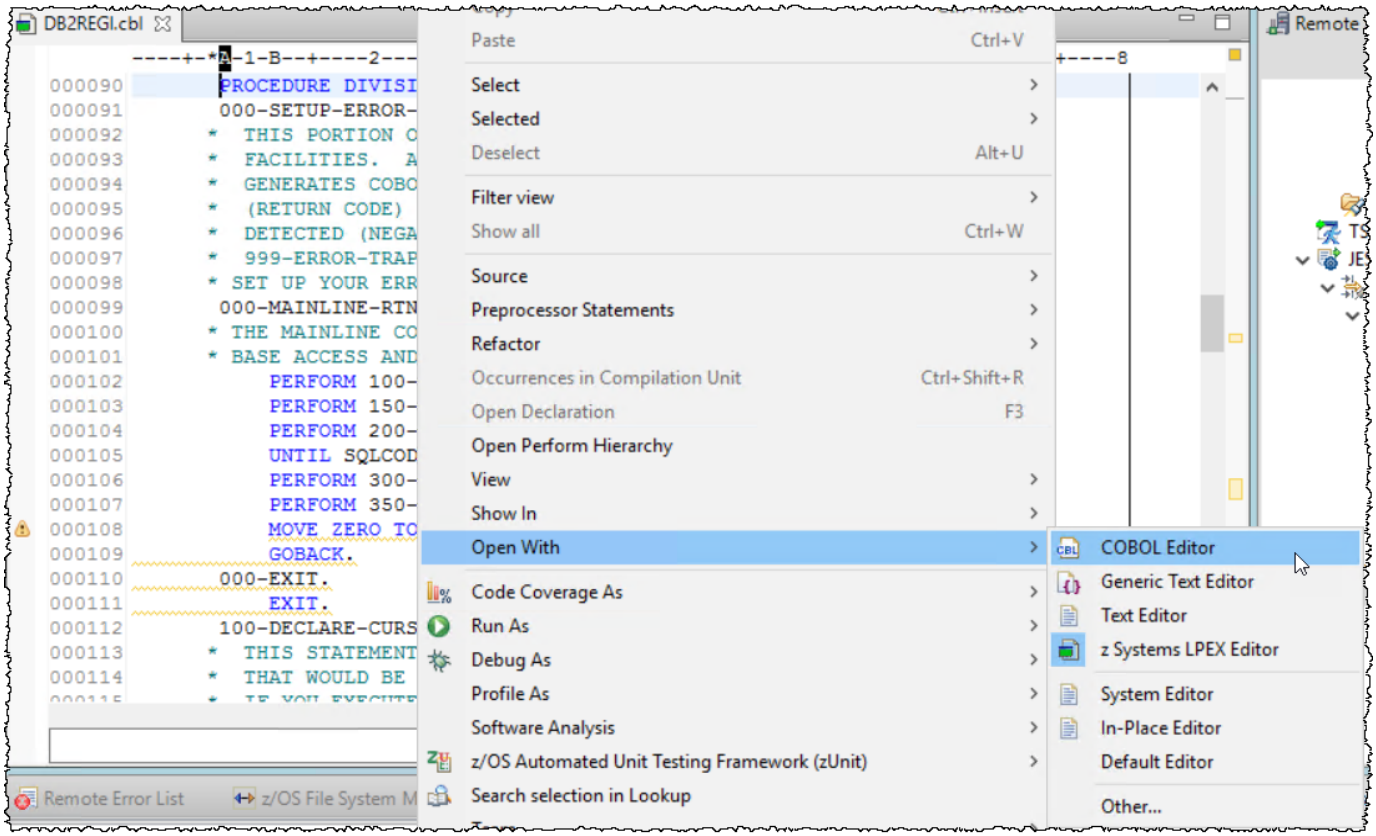
⏭ To switch the editor, right click on the program and select **Open with > z Systems LPEX Editor**



5.1.4 Note that the column on left are similar as the prefix area of the ISPF editor..
You may type commands to add lines etc..
Like **I** to insert a line, **d** to delete, **m** to move, etc. **Try it**.

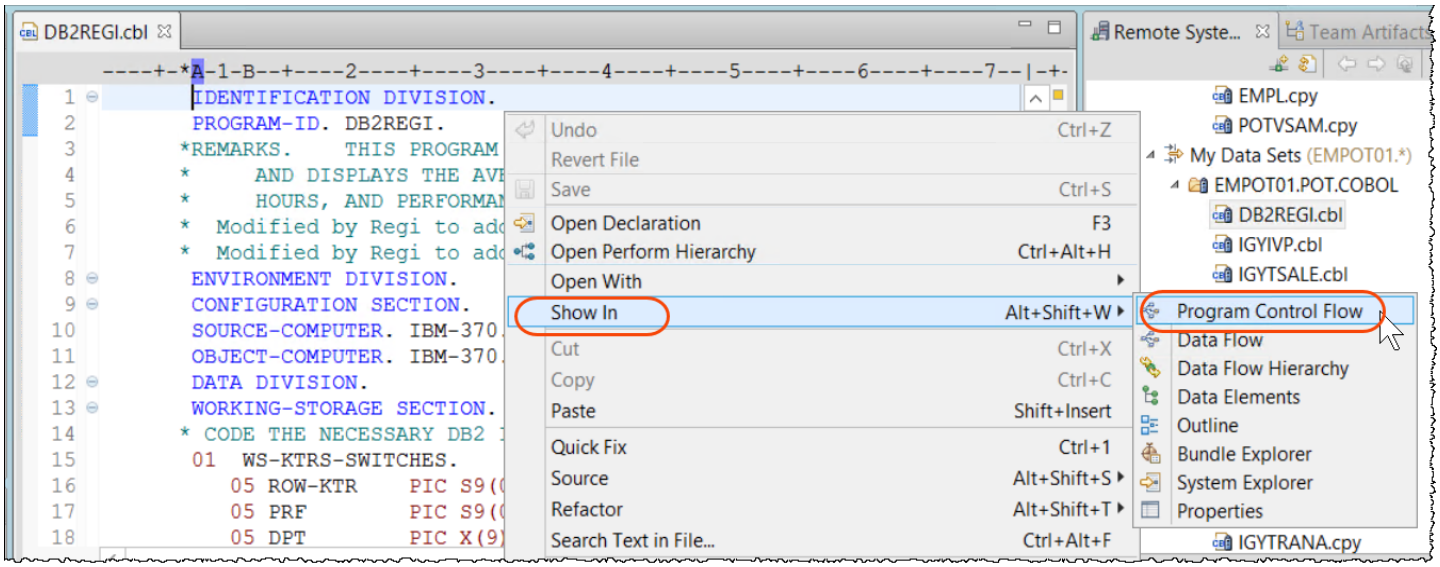


5.1.5 Switch back to **COBOL editor** (since the screen captures uses the COBOL editor)
Right click on the program and select **Open with > COBOL Editor**
Click **No** for saving changes**.**
This editor is more like the Java Editor and gives you more capabilities. When you see all advantages, probably you will prefer this editor than the ISPF like editor (LPEX)..
Please keep this program opened as we will work on it later.



5.1.6 Let's see the program in more details..

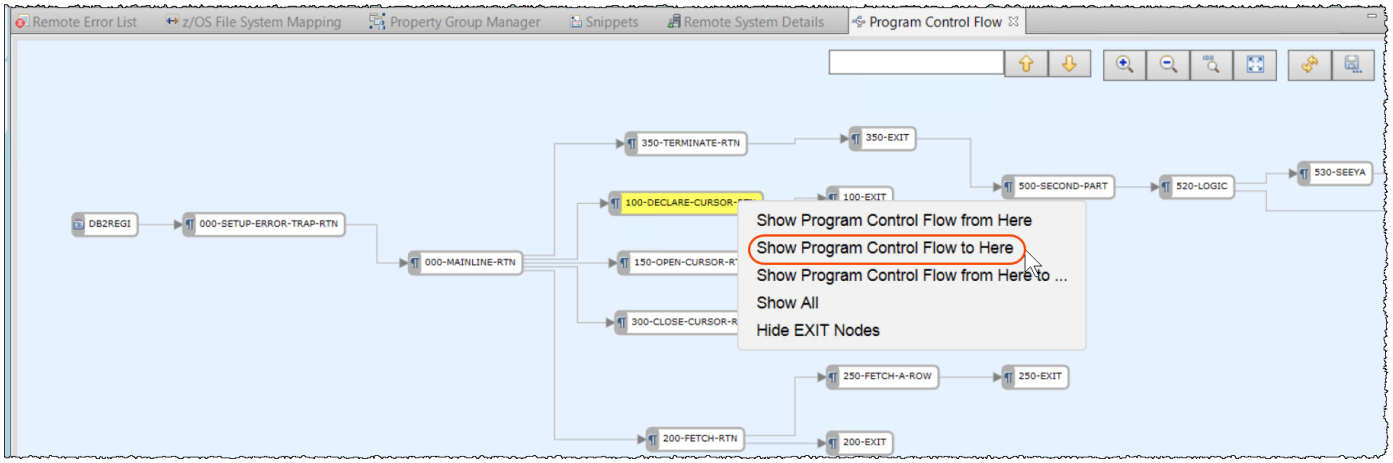
Right click on the editor and select **Show In → Program Control Flow**



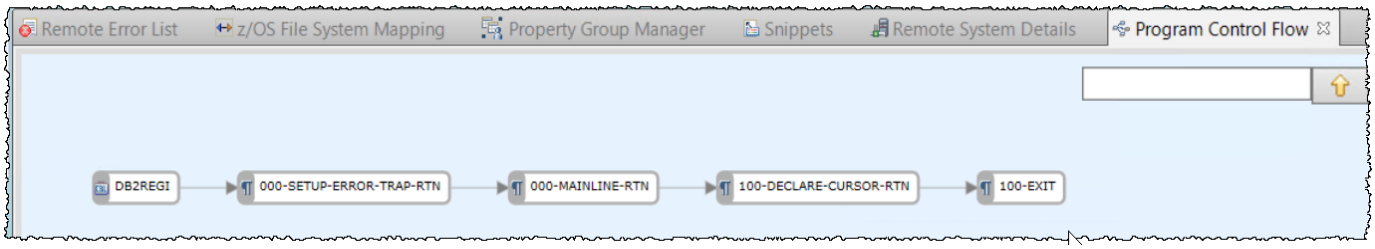
5.1.7 See the results under *Program Control Flow* view.

⏭ Double click "**Program Control Flow"** title to have a bigger diagram (full page).

⏭ Right-click on **100-DECLARE-CURSOR-RTN** box and select **Show Program Flow Control to here.**



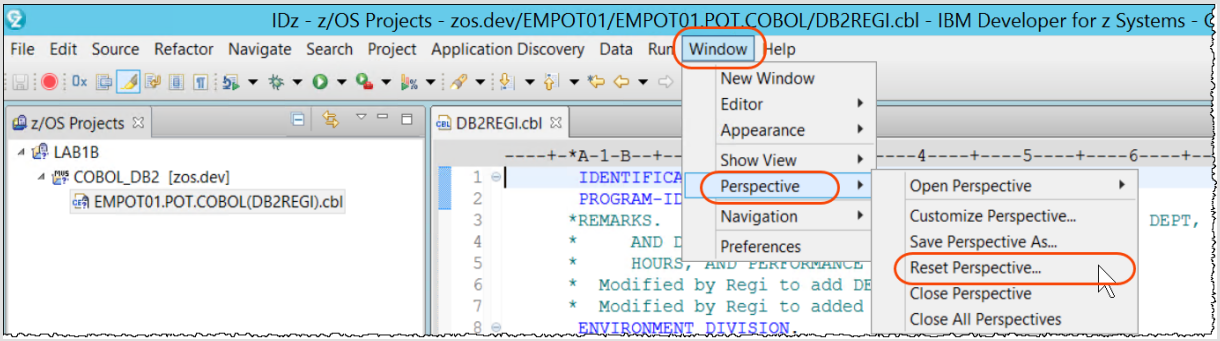
5.1.8 It shows the path to reach this paragraph.. Nice uh? That helps with extremely complex diagrams



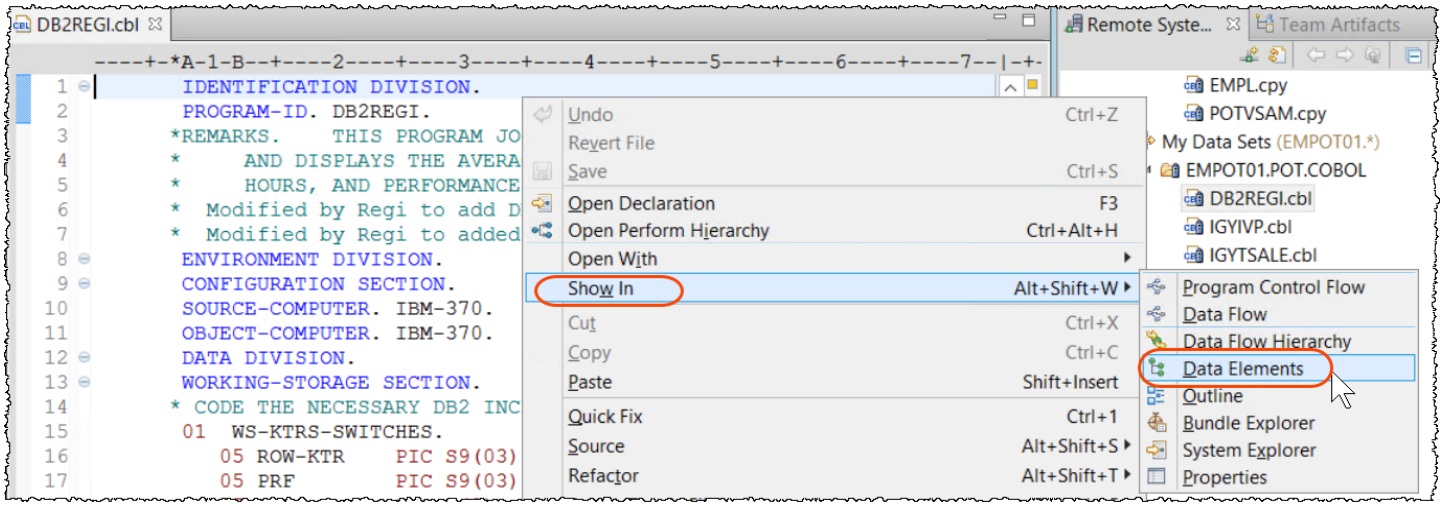
The perspectives are not the same as here? Got confused with the opened views?

At any time, you may restore the perspective to the default. You may need that when you did mistakes and closed views you should not.

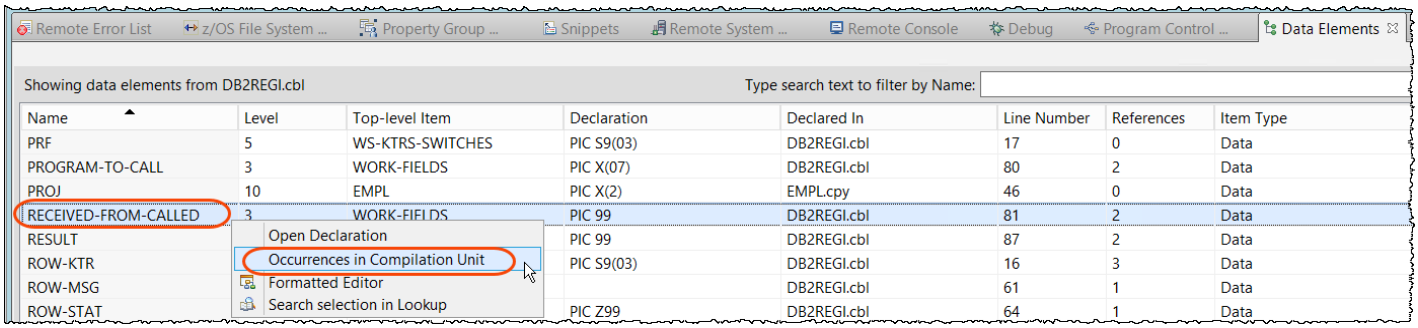
Just select **Windows > Perspective > Reset Perspective…** as below:



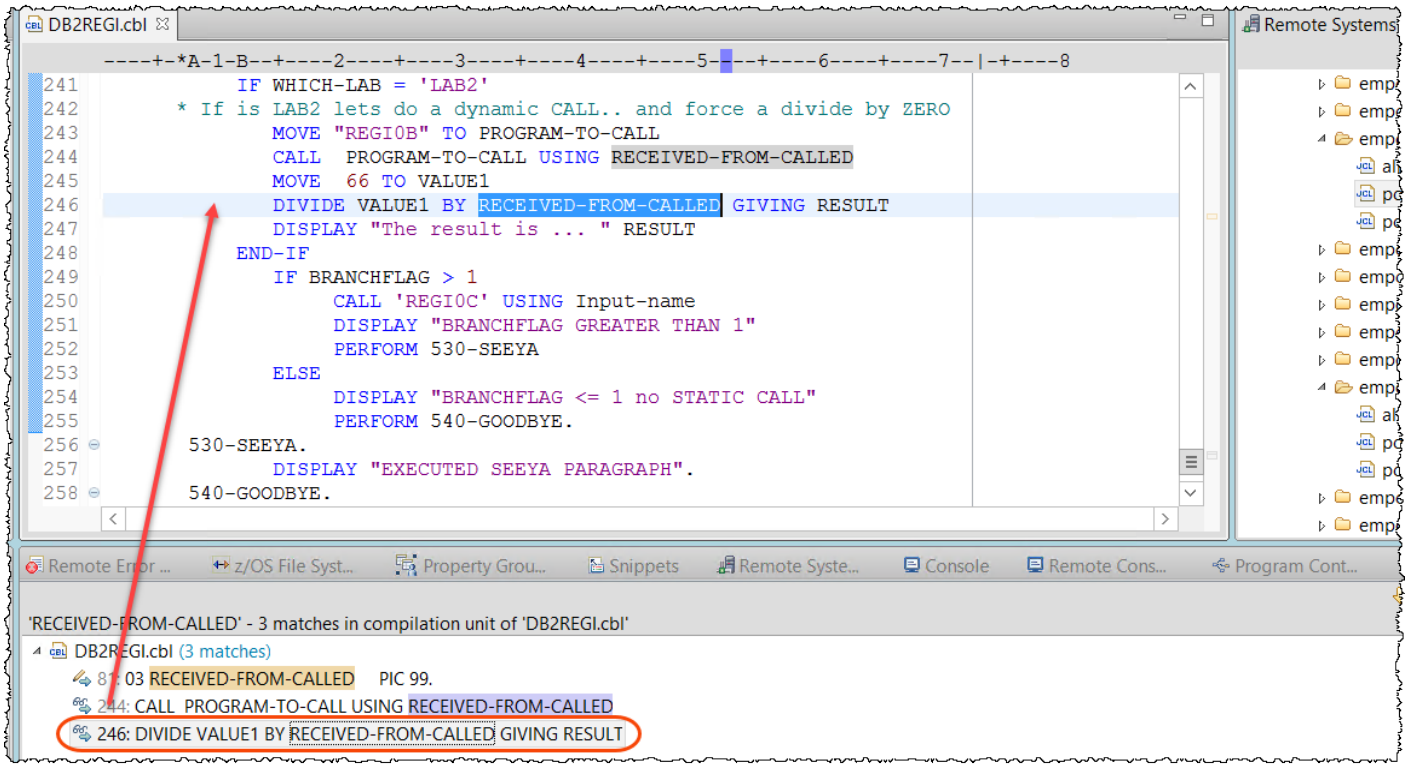
5.1.9 Double click "**Program Control Flow"** title to have the diagram smaller.
Right click again on the editor and select
Show In → Data Elements



5.1.10 On *Data Elements* view, resize the view, scroll down and right click on **RECEIVED-FROM-CALLED** and select **Occurrences in Compilation Unit**



5.1.11 Double click on line **246**. It will show in the line that is abending.
Notice the colors. When the variable is referenced, it is blue and when it is modified is brown.

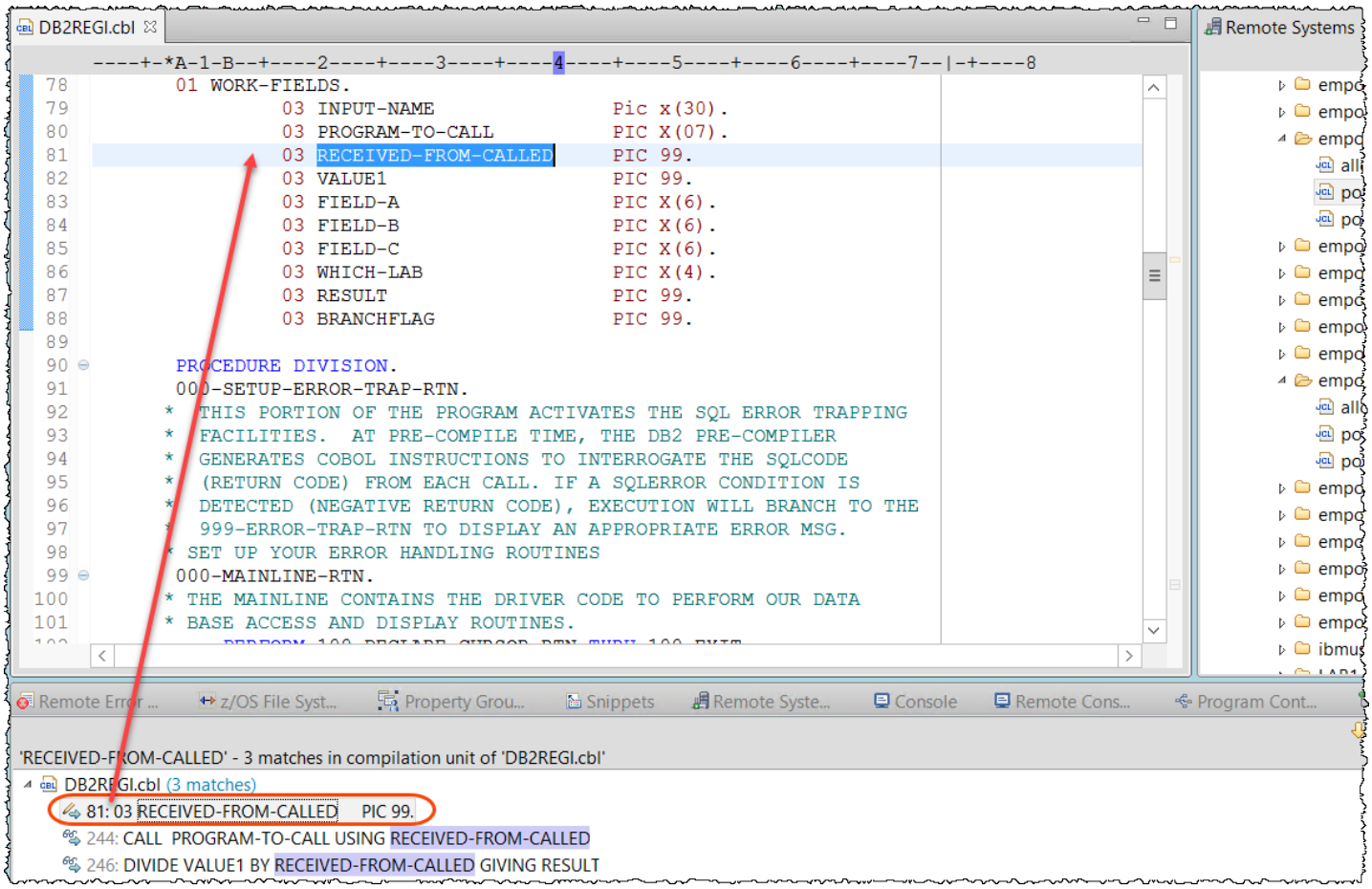


You need to fix the called program named **REGI0B** that is returning 0 on this variable as seen before with Fault Analyzer.

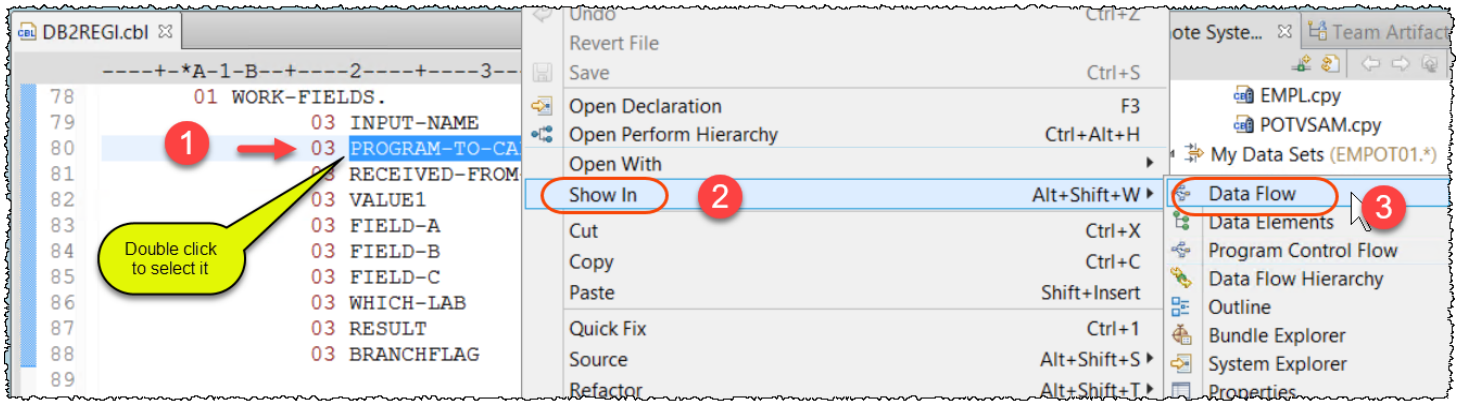
___ Exploring Data Flow

This capability is cool. *Program data flow* provides a graphical and hierarchical view of the data flow within a COBOL program. You can use this feature to examine how a data element is populated, modified, or written elsewhere.

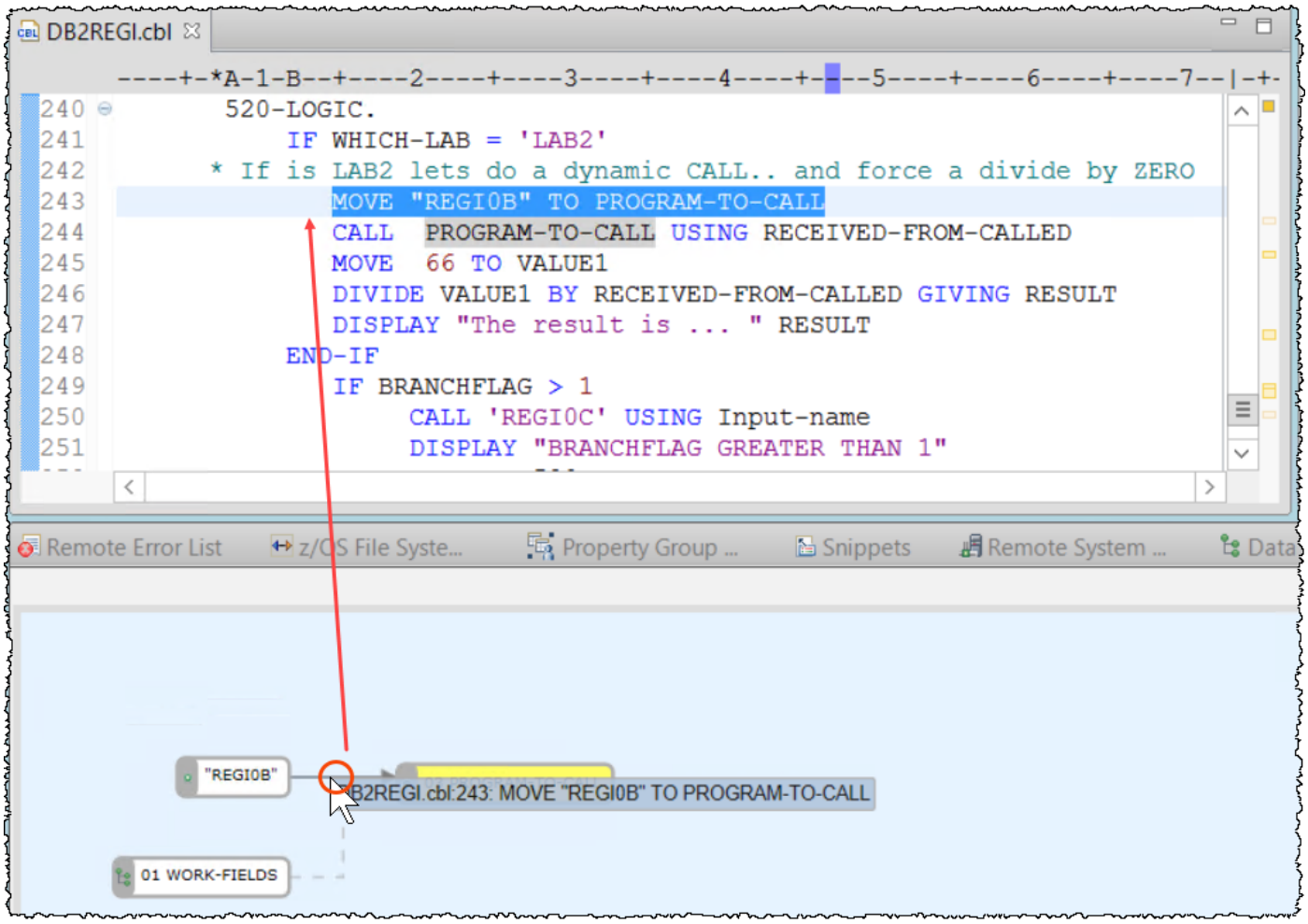
5.1.12 ⏭ Double click on line **81**. It will show where *RECEIVED-FROM-CALLED* is defined.



5.1.13 Select **PROGRAM-TO-CALL (on line 80)** double clicking on it, **right click** and select **Show In > Data Flow**. (the options order can differ from the screen capture here).



5.1.14 Move the cursor to the line between **REGI0B** and **03 PROGRAM-TO-CALL**.
Clicking in this line shows the program statement where "*REGI0B*" is moved to *PROGRAM-TO-CALL*.



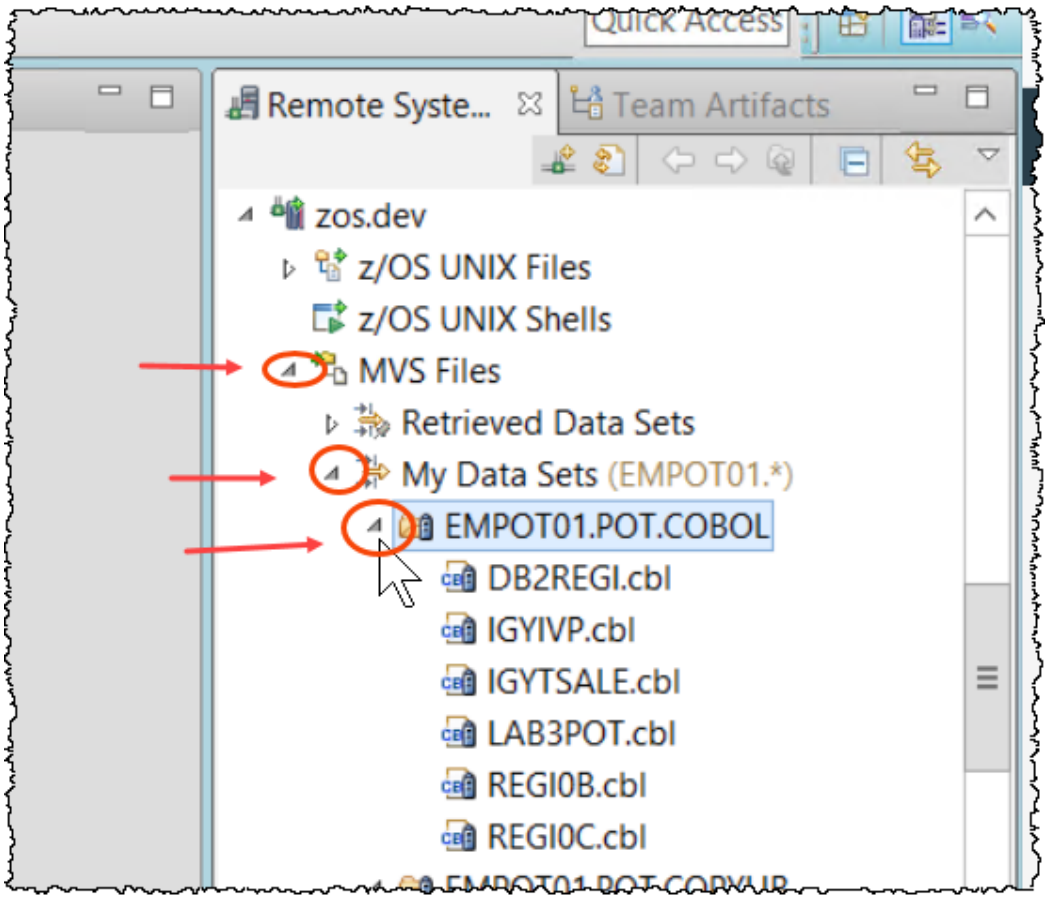
5.1.15 **Close** all opened editors if still opened. (**CTRL + Shift + F4**)
Or just click on the . of each opened editor.

___ 5.2 Fixing the program REGI0B

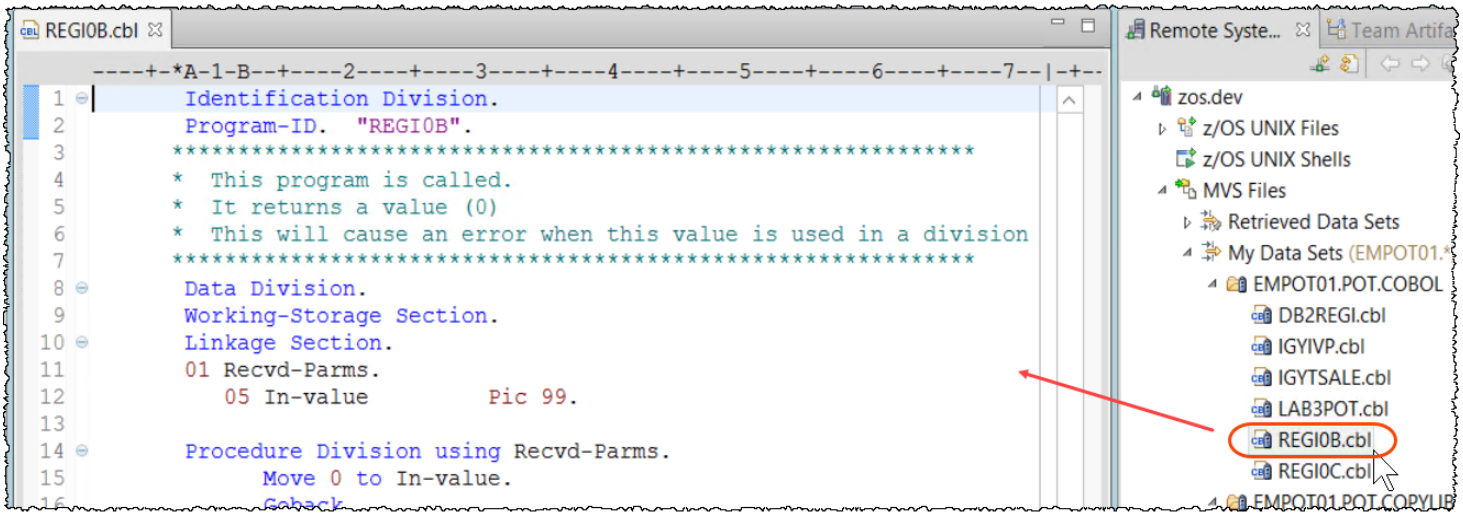
You need to fix the called program named REGI0B that is returning 0 on this variable as seen before with Fault Analyzer and the debugger.

This program is on your ***PDS EMPOT01.POT.COBOL***. .

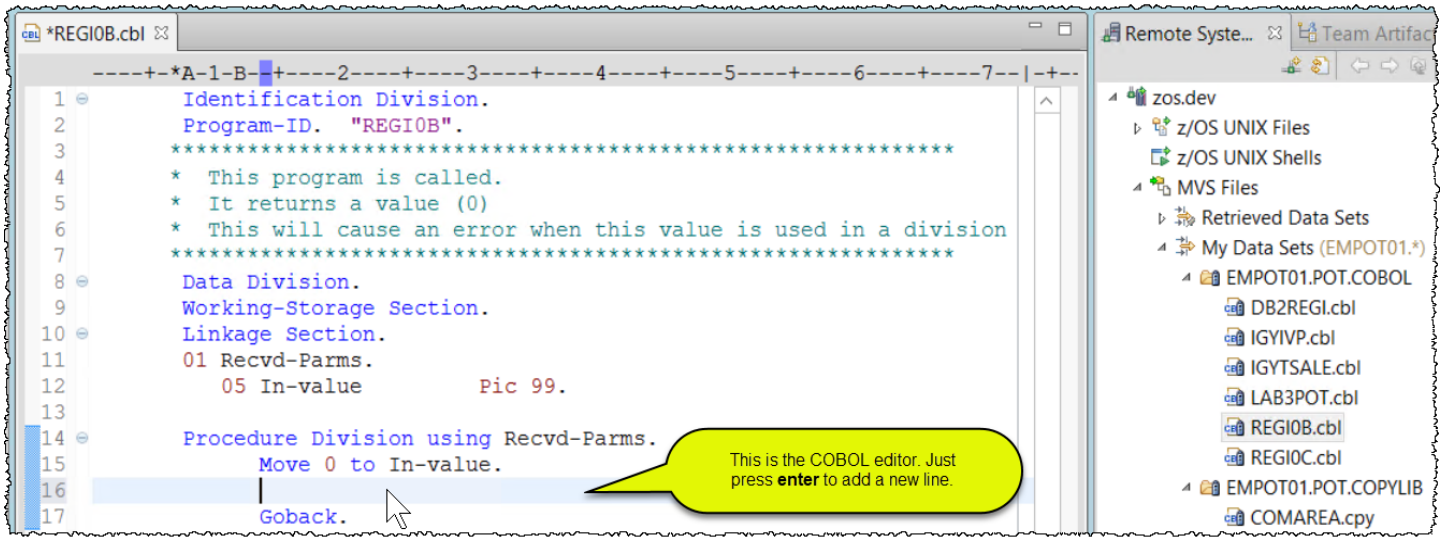
5.2.1 Using *Remote Systems* view **scroll back**, expand **MVS Files, My Data Sets** and **EMPOT01.POT.COBOL**. if not already expanded.



5.2.4 **Double** click on the program **`REGI0B.cbl`** to edit it.



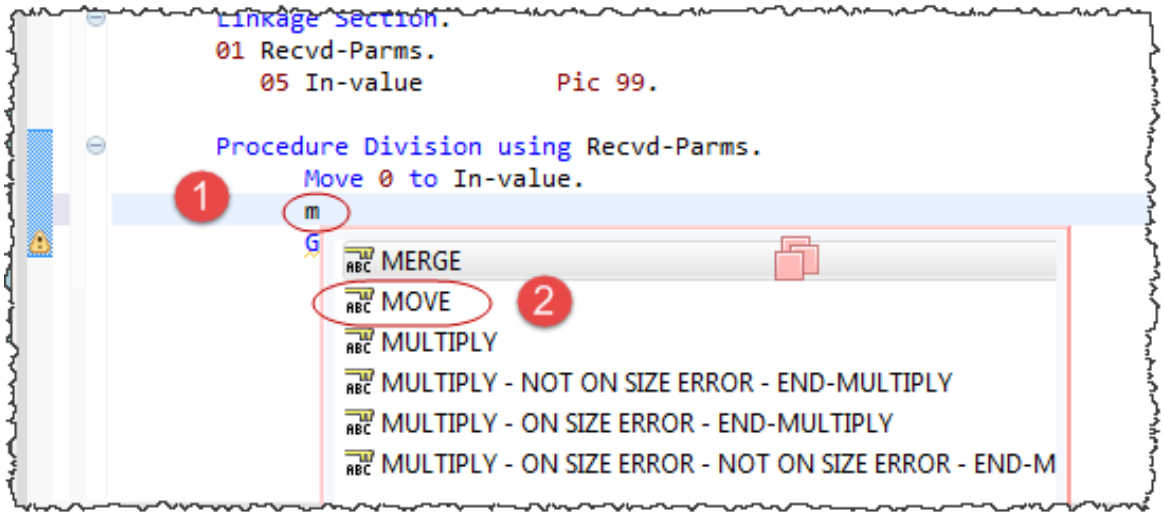
5.2.5 Move the cursor after the **In-value** and press **enter** to add a blank line



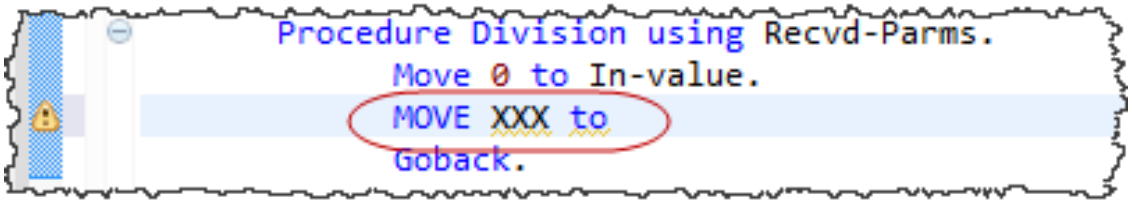
5.2.6 You can now take advantage of the *editor content assist...*

On the column 12 as shown below, type **m** and press **Ctrl + Space**

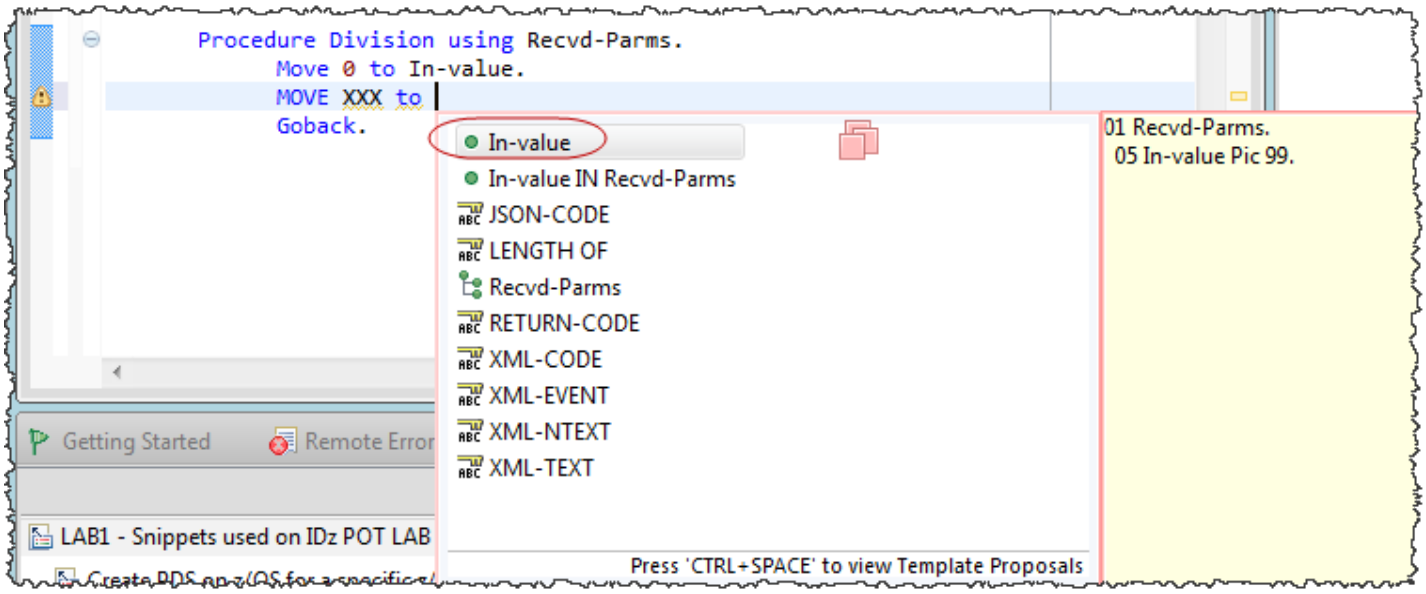
5.2.7 Select the statement **MOVE** (you can use the mouse double click or select and press enter)



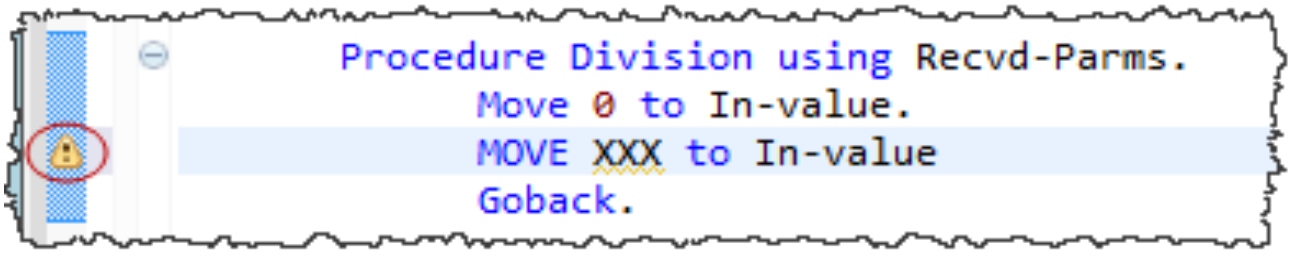
5.2.8 Type "**XXX to**"



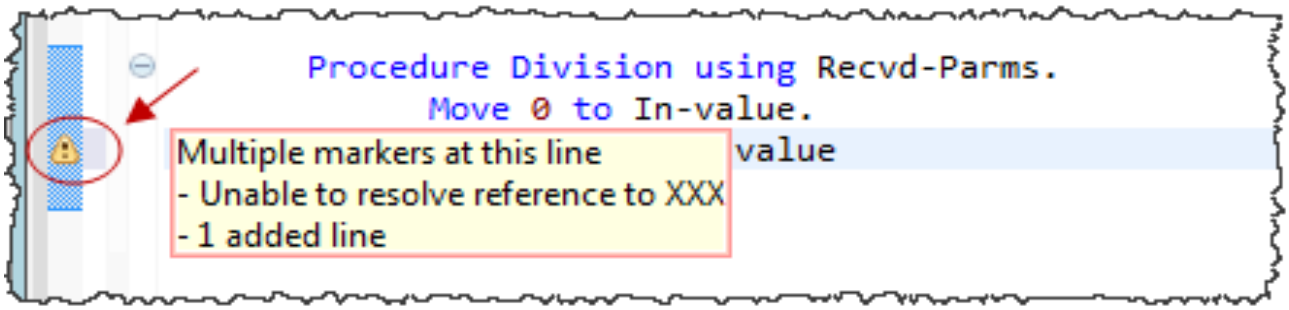
5.2.9 Press **Ctrl + Space** to list the variables and select **In-value**.
Notice that variables could be defined in copybooks.



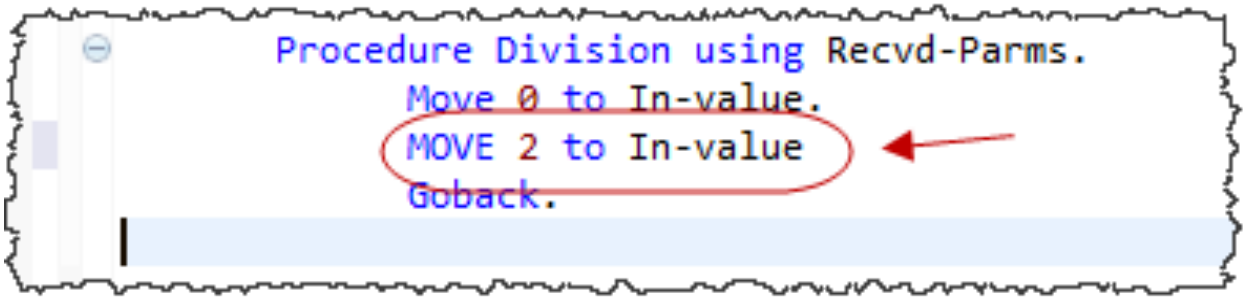
A yellow mark shows that something is wrong.



5.2.10 Move the cursor to the yellow mark to see what the error is.
You are finding this error without using the z/OS compilation. If you were using ISPF you would see that only after submitting this JOB to the z/OS COBOL compiler.
Productivity and z/OS CPU saving. You are using the power of the smart editor.



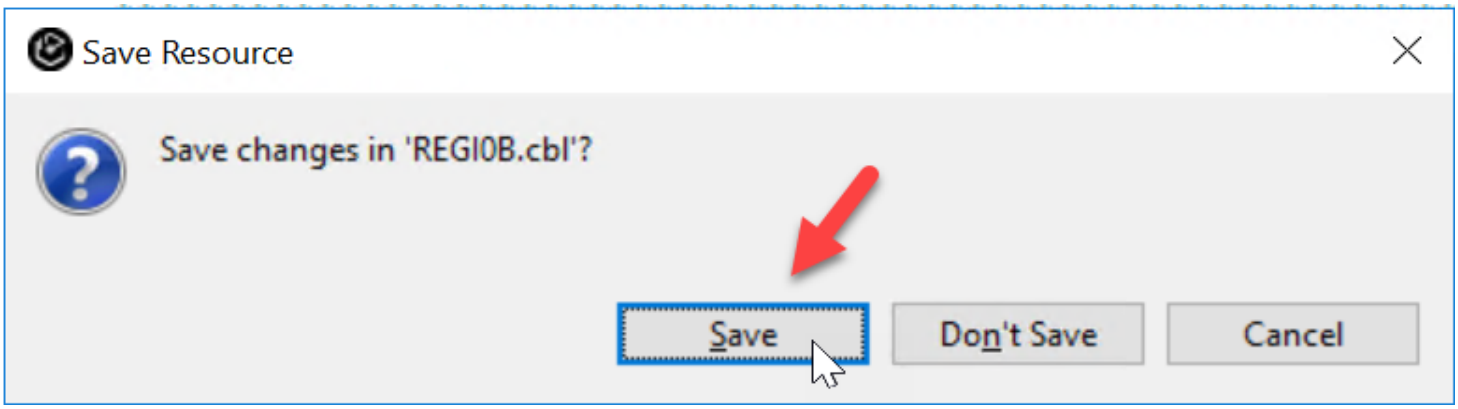
5.2.11 Change from **XXX** to **2** as below. Note that the icon will go away.



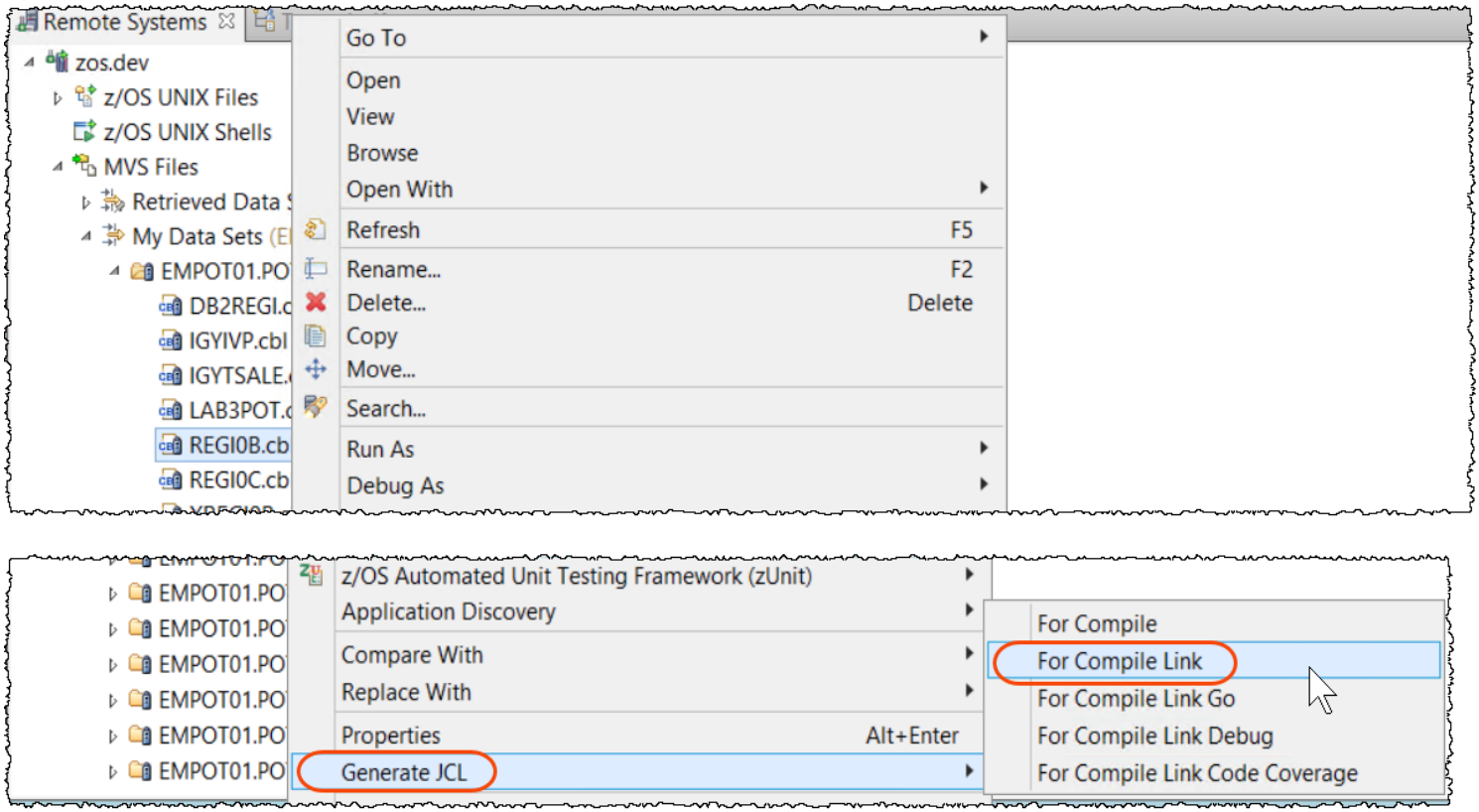
___ 5.3 Compile and link REGI0B.

You need to compile and link the program that you just changed. You could use a JCL or let IDz generate a JCL for you. Once you have a Property Group associated to the COBOL code a JCL file could be generated on the fly when you need it. You have associated the property Group at step 7.2.1 above.

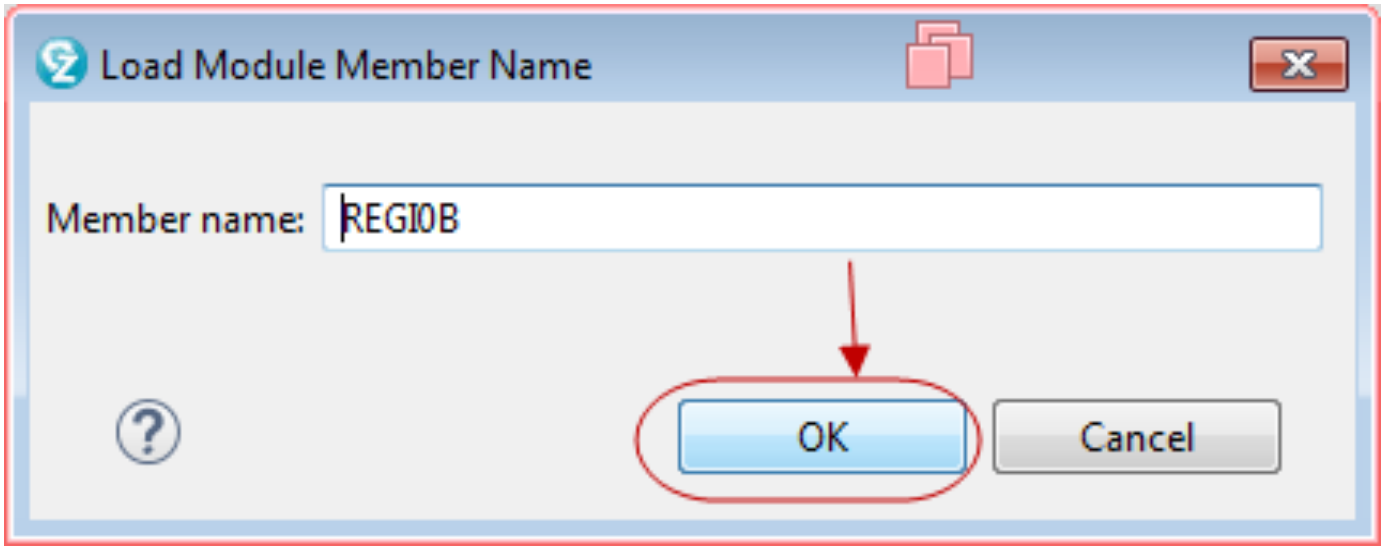
5.3.1 Click **Ctrl + Shift + F4** to close the editors. Or just left click on the . of each opened editor.
Click **Save** to save the changes.



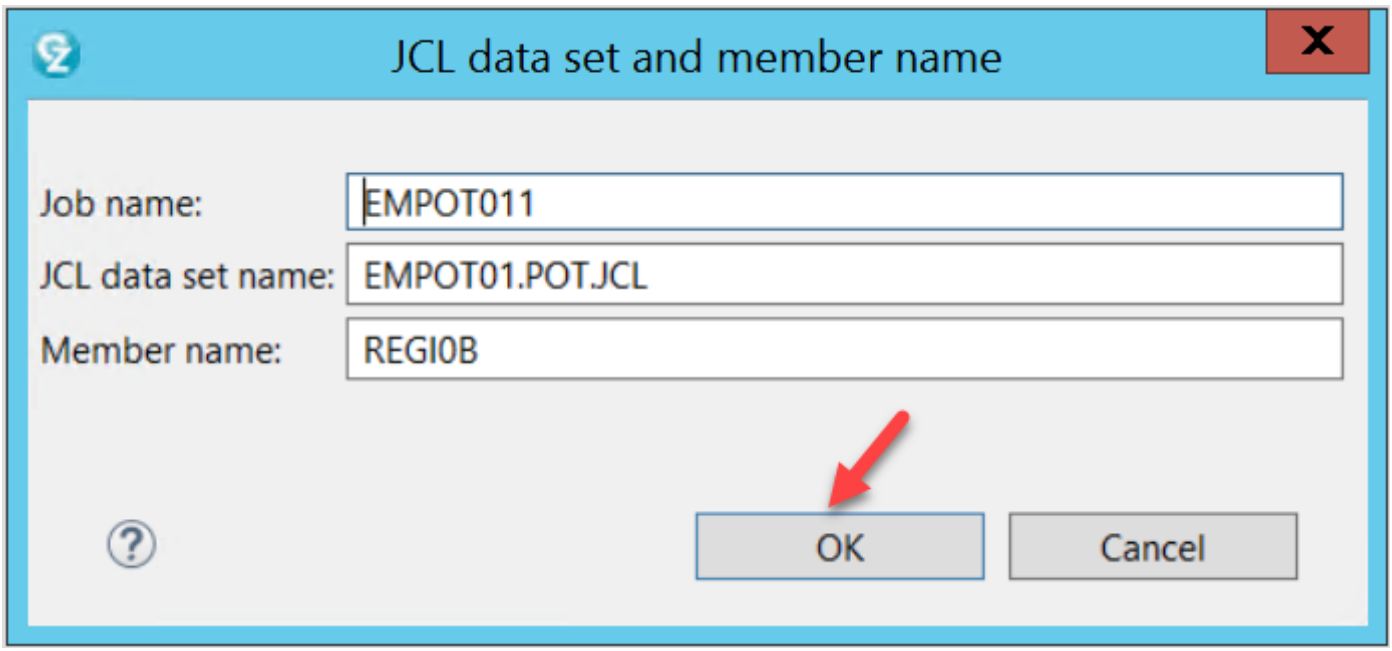
5.3.2 Using the Remote Systems view **right click** on **REGI0B.cbl** and select **Generate JCL > For Compile Link** .



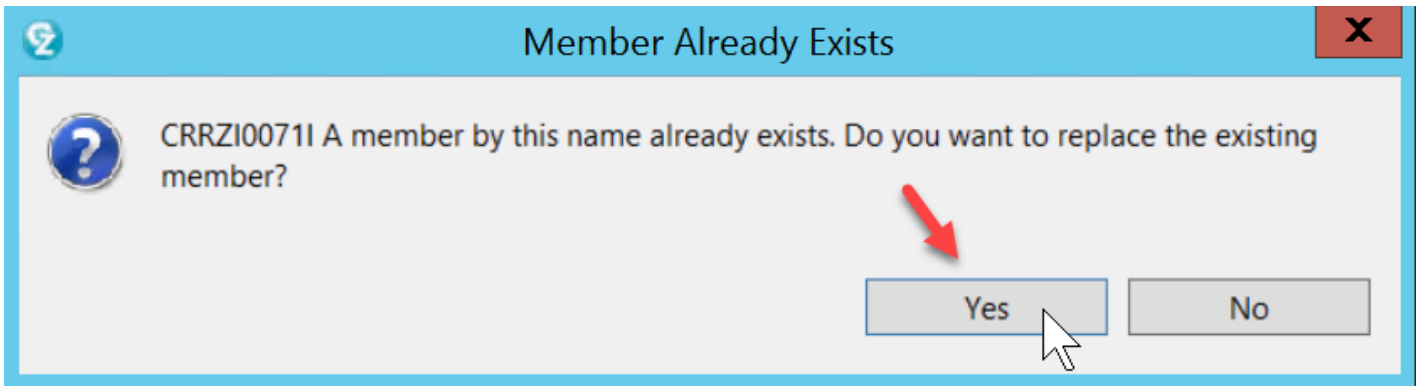
5.3.3 Accept the default member name and click **OK**



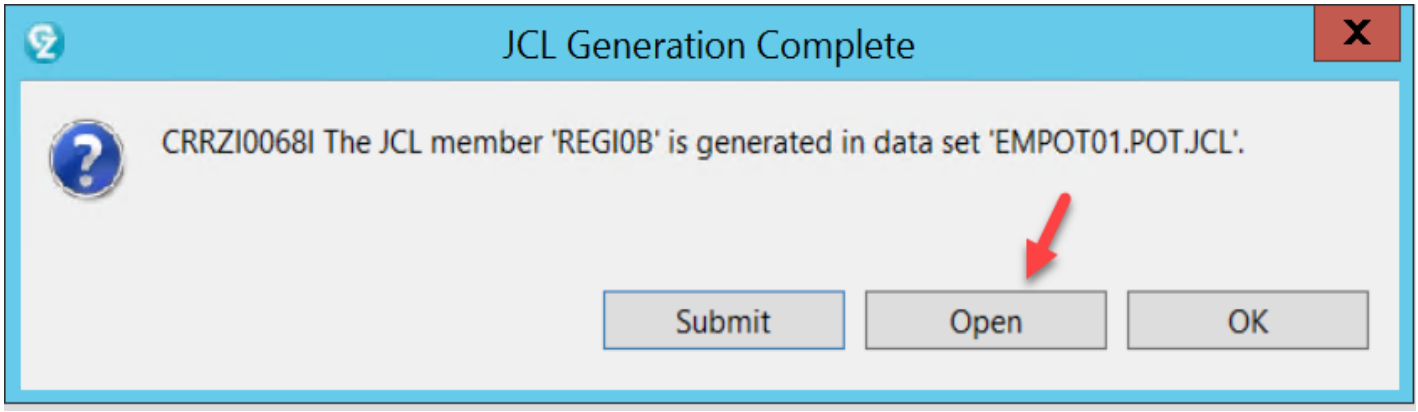
5.3.4 Accept the default and click **OK**



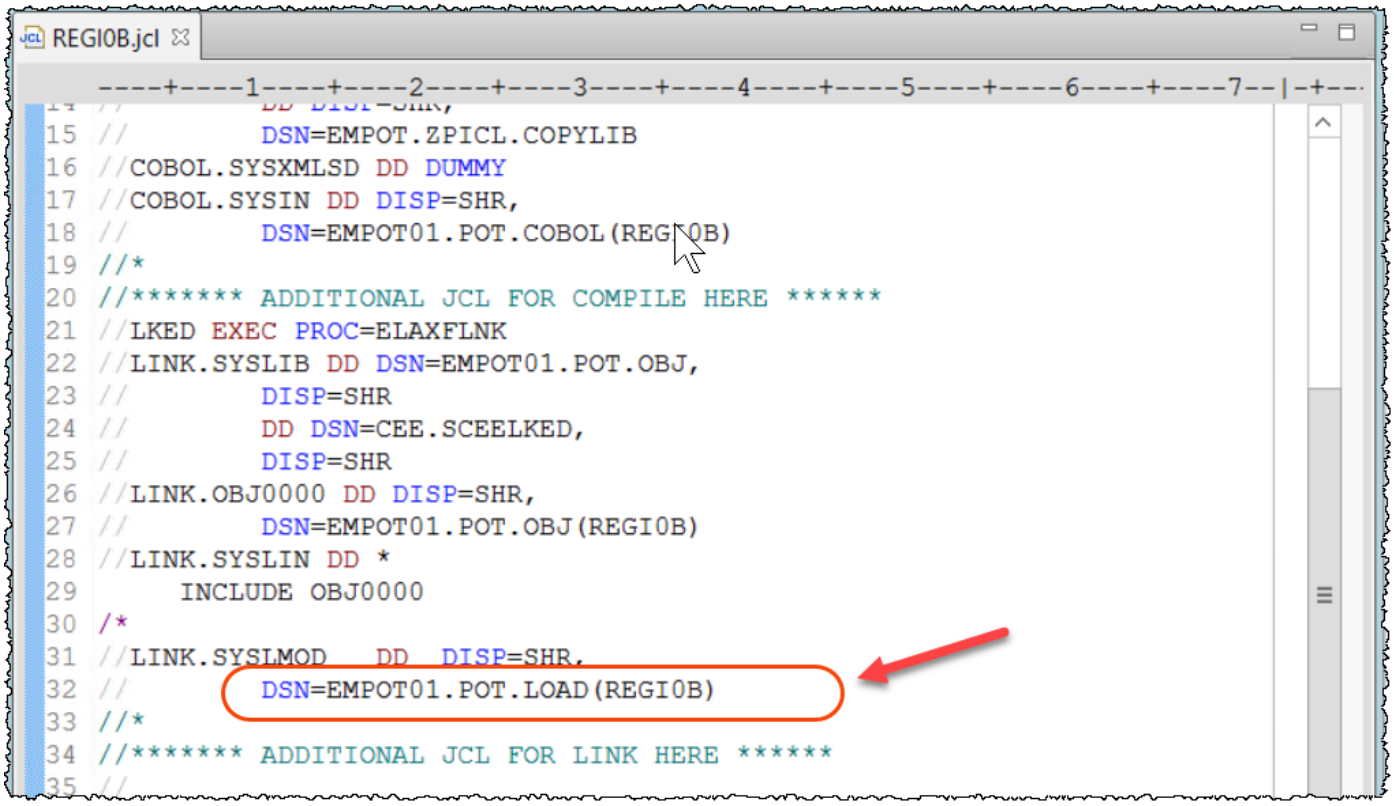
5.3.5 Click **Yes** if the member already exists



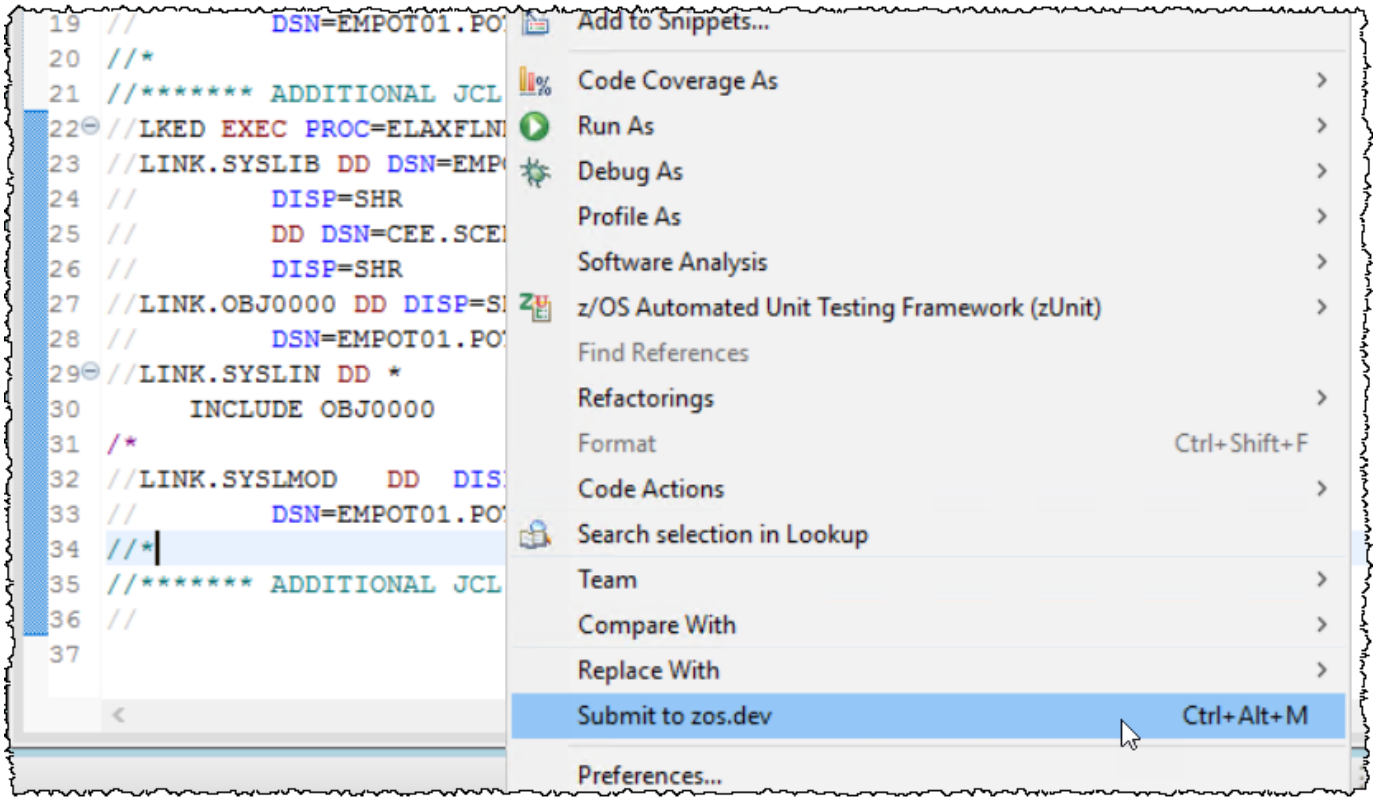
5.3.6 Click **Open** to verify the generated JCL



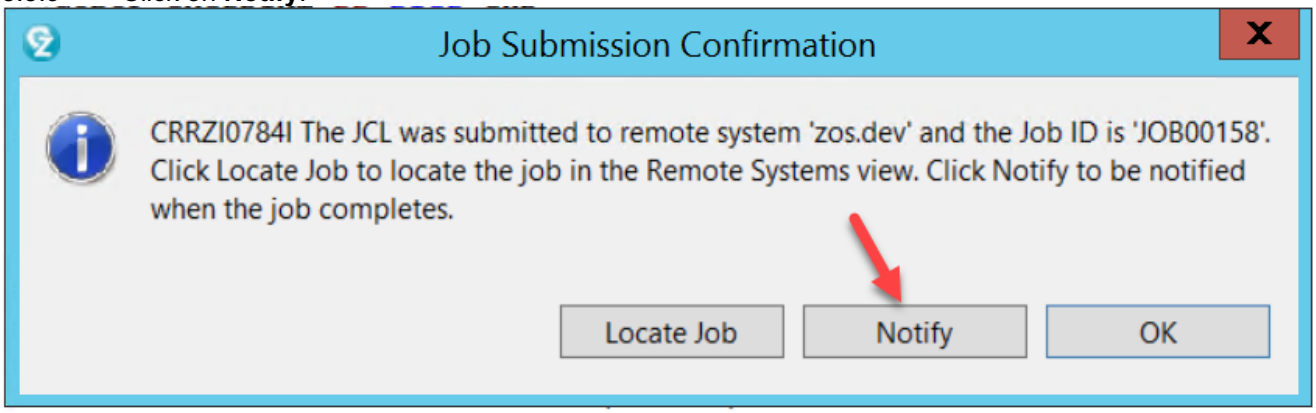
5.3.7 **Scroll down** and notice that the *Load module* will be created on the PDS: **`EMPOT01.POT.LOAD`**.



5.3.8 Right click on the JCL and click **Submit to zos.dev**

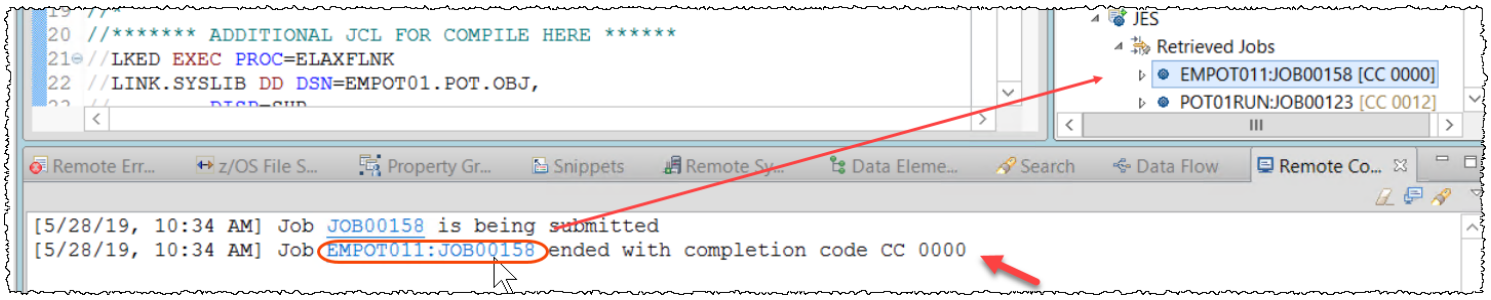


5.3.9 Click on **Notify**.



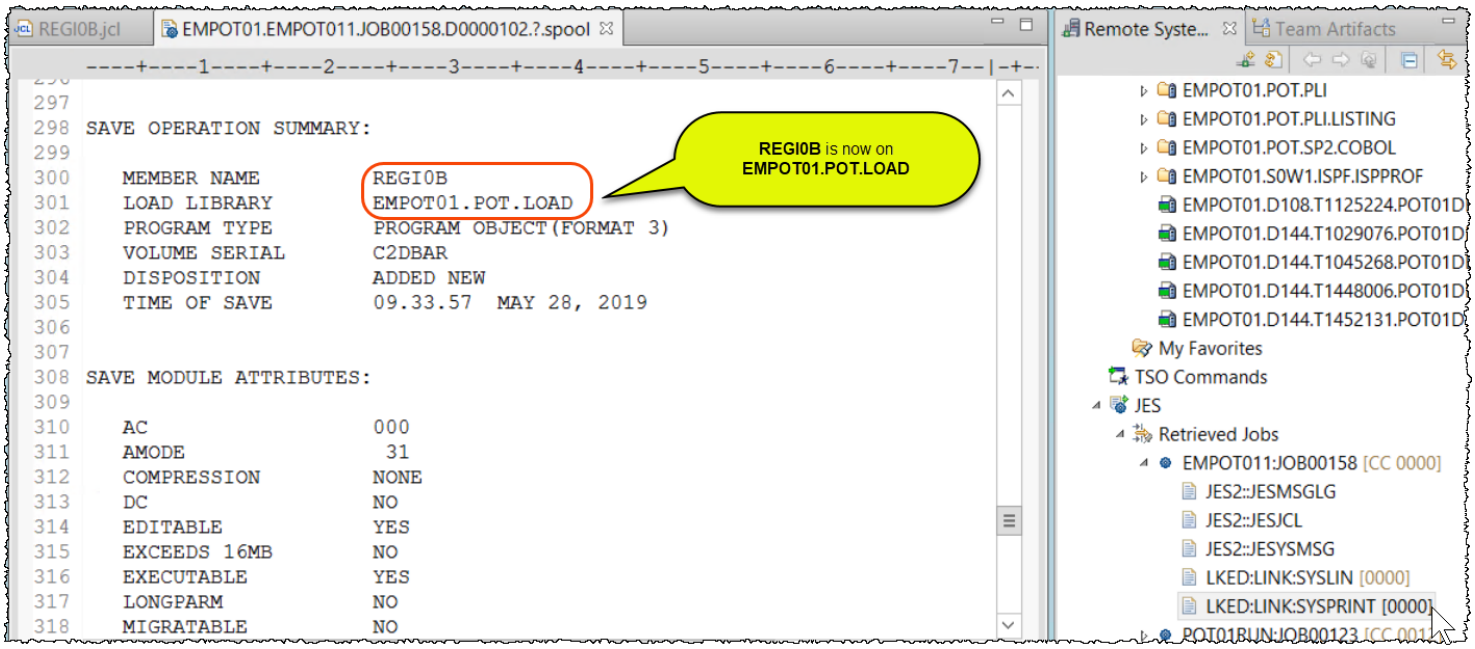
5.3.10 You MUST have **000** as return code.

⏭ Click on **EMPOT011:JOB00xxx**



5.3.11 ⏭ Expand **EMPOT01:JOB00xxx** and verify the results double clicking on **LKED:LINK:SYSPRINT.**

⏭ **Scroll down** and verify that the load module **REGI0B** invoked by your DB2 COBOL program is now created in **EMPOT01.POT.LOAD.**



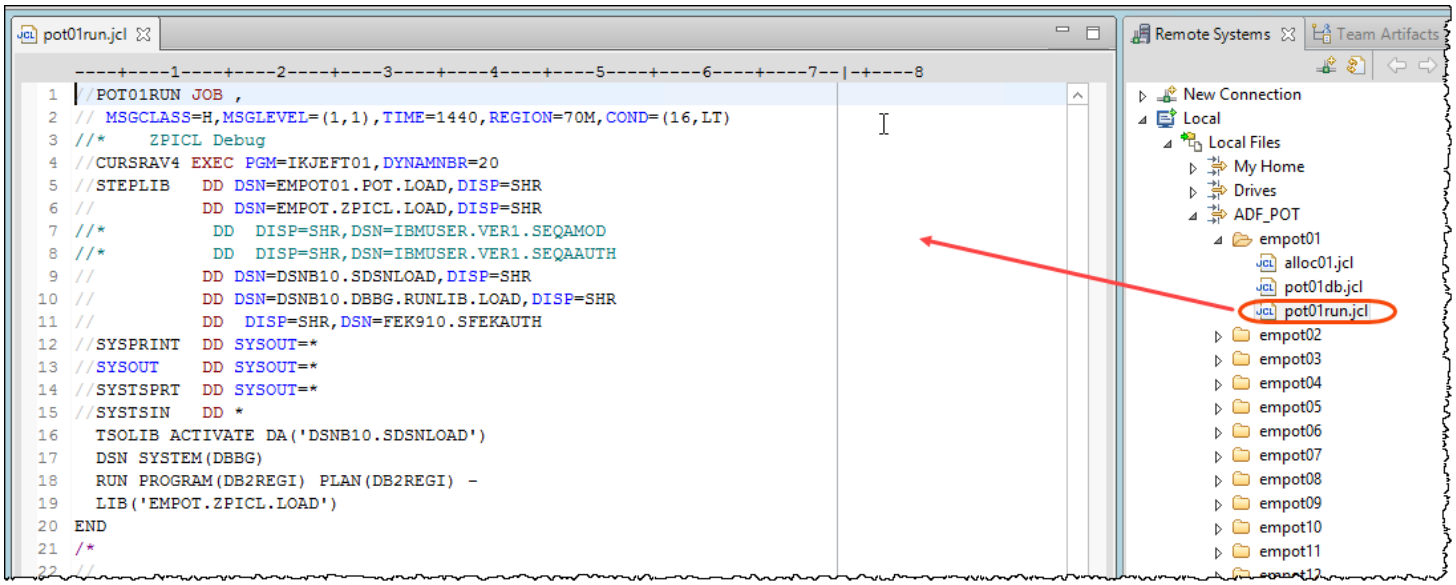
5.3.10 ⏭ Use **Ctrl + Shift + F4** to close all editors.

Or just click on the ⊠. of each opened editor

___ 5.4 Execute the COBOL/DB2 program

On this step, you will explore some capabilities of the JCL editor and modify the JCL for execution.

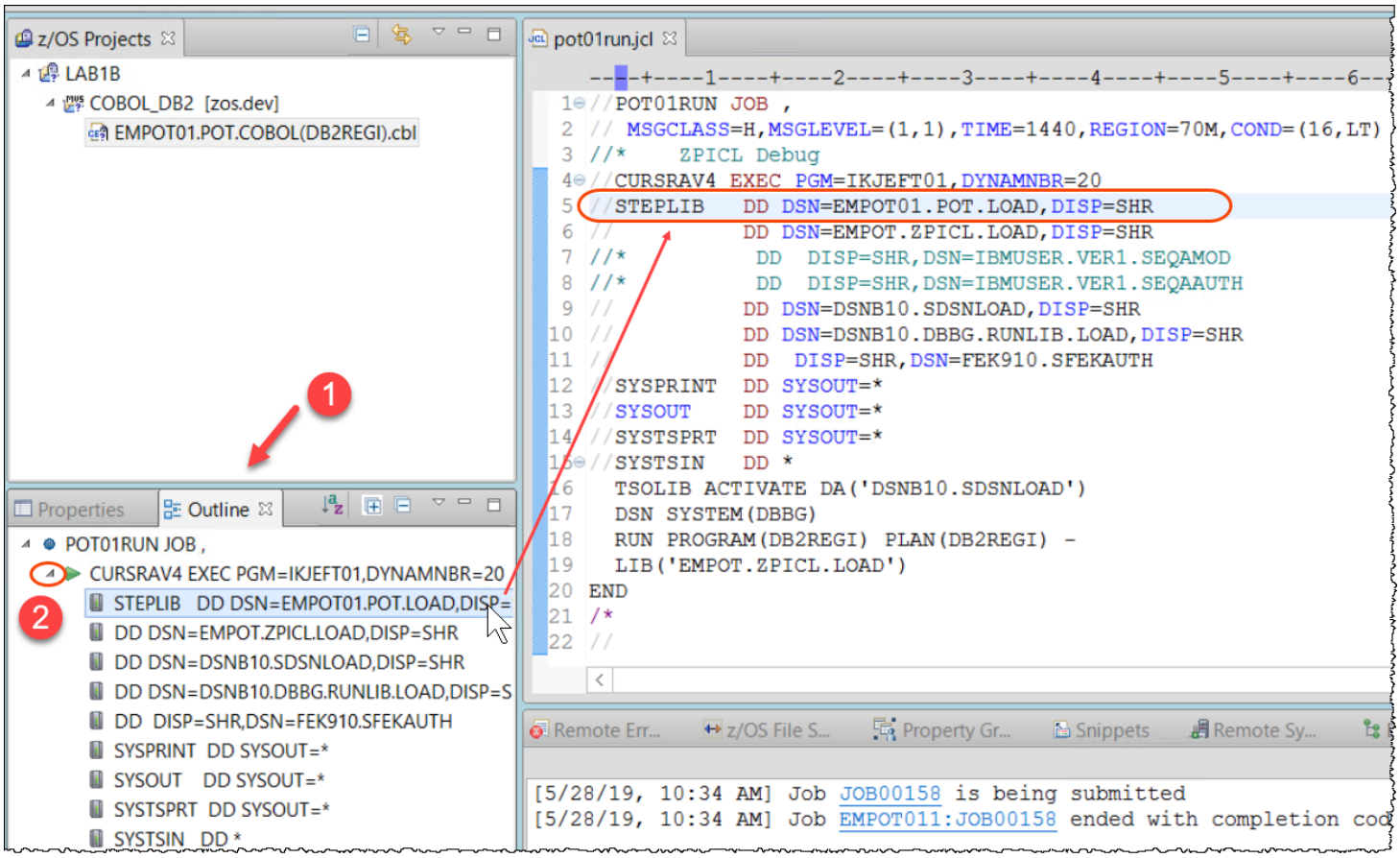
5.4.1 ⏭ Using *Remote System View,* scroll back to locate the file **`pot01run.jcl`** under the folder **`Local/Local Files/ADF_POT/empot01`** and **double click** to edit.



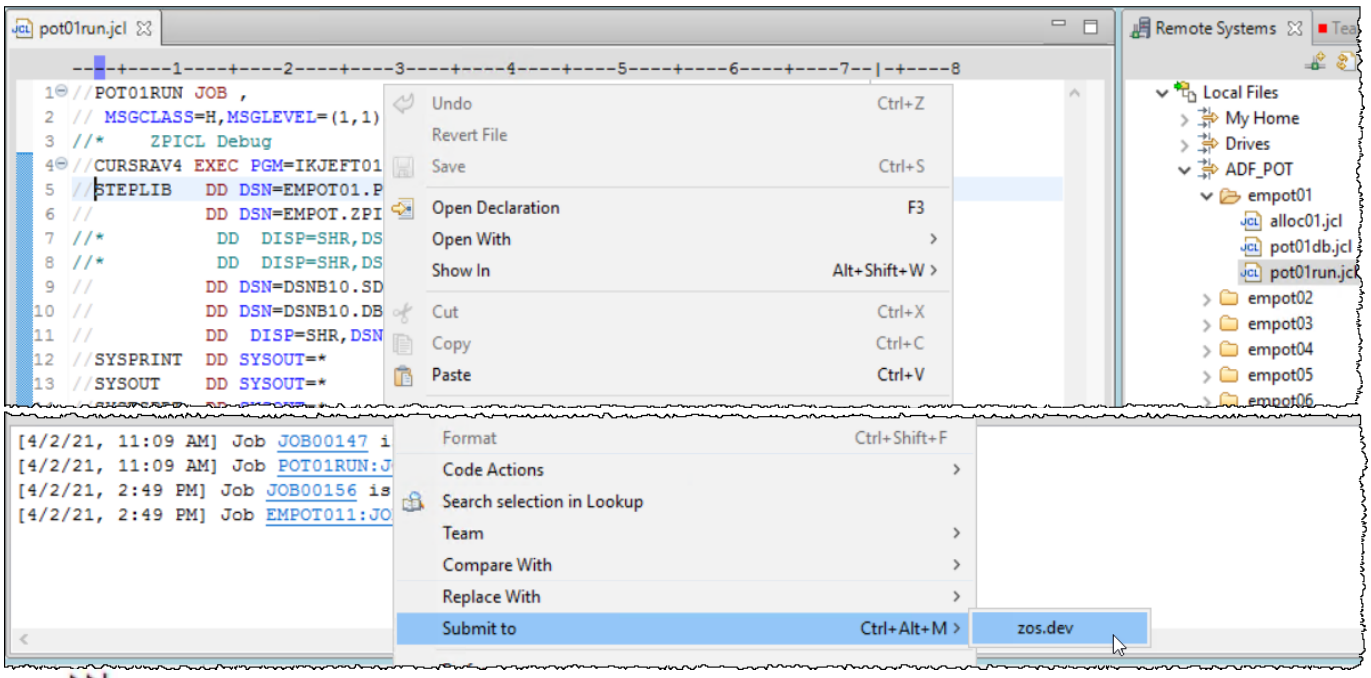
5.4.2 Notice on bottom left corner the same *Outline* view that you have for the COBOL program. It will help to navigate on large JCL members.

⏭ Expand **`CURSRAV4 EXEC PGM=`** and click on **`STEPLIB`**

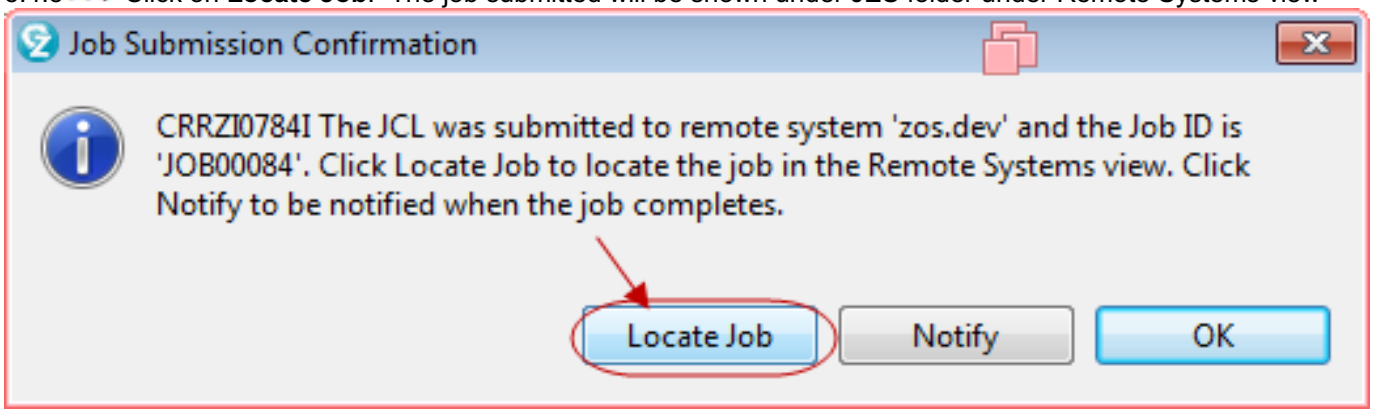
5.4.3 ⏭ Notice that your PDS where **REGI0B** is created is already on the STEPLIB.


z/OS Projects
LAB1B
COBOL_DB2 [zos.dev]
EMPOT01.POT.COBOL(DB2REGI).cbl
pot01run.jcl
1
2
Properties
Outline
POT01RUN JOB ,
CURSRAV4 EXEC PGM=IKJEFT01,DYNAMNBR=20
STEPLIB DD DSN=EMPOT01.POT.LOAD,DISP=
DD DSN=EMPOT.ZPICL.LOAD,DISP=SHR
DD DSN=DSNB10.SDSNLOAD,DISP=SHR
DD DSN=DSNB10.DBBG.RUNLIB.LOAD,DISP=S
DD DISP=SHR,DSN=FEK910.SFEKAUTH
SYSPRINT DD SYSOUT=*
SYSOUT DD SYSOUT=*
SYSTSPRT DD SYSOUT=*
SYSTSIN DD *
//POT01RUN JOB ,
// MSGCLASS=H,MSGLEVEL=(1,1),TIME=1440,REGION=70M,COND=(16,LT)
//* ZPICL Debug
//CURSRAV4 EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB DD DSN=EMPOT01.POT.LOAD,DISP=SHR
// DD DSN=EMPOT.ZPICL.LOAD,DISP=SHR
//* DD DISP=SHR,DSN=IBMUSER.VER1.SEQAMOD
//* DD DISP=SHR,DSN=IBMUSER.VER1.SEQAAUTH
// DD DSN=DSNB10.SDSNLOAD,DISP=SHR
// DD DSN=DSNB10.DBBG.RUNLIB.LOAD,DISP=SHR
// DD DISP=SHR,DSN=FEK910.SFEKAUTH
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
TSOLIB ACTIVATE DA('DSNB10.SDSNLOAD')
DSN SYSTEM(DBBG)
RUN PROGRAM(DB2REGI) PLAN(DB2REGI) -
LIB('EMPOT.ZPICL.LOAD')
END
/*
//
Remote Err...
z/OS File S...
Property Gr...
Snippets
Remote Sy...
[5/28/19, 10:34 AM] Job JOB00158 is being submitted
[5/28/19, 10:34 AM] Job EMPOT011:JOB00158 ended with completion cod

5.4.4 Right click and select **Submit to > zos.dev**



5.4.5 Click on **Locate Job**. The job submitted will be shown under JES folder under Remote Systems view



5.4.6 You MUST have the return code 0000 and the bug will be corrected.

Using *Remote Systems* view, verify the results double expanding **`POT01RUN:JOB00xxx`** (where **`00xxx`** can be any number) and clicking on **`CURSRAV4:SYSOUT`**. Now you MUST have 000 as return code

Tip: You may need to refresh Retrieved Jobs to see it go from active to finished


pot01run.jcl
EMPOT01.POT01RUN.JOB00113.D0000103.?.spool
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
33 *DEPT*   REG
34 *PERF-AVG*   0000000
35 *PERF-MIN*   0009
36 *PERF-MAX*   0009
37 *HOURS-AVG* 0002675
38 *HOURS-MIN* 0002675
39 *HPOURS-MAX* 0002675
40 REG   .00   9.00   9.00   26.75   26.75   26.75
41 *DEPT*   N/A
42 *PERF-AVG*   0000000
43 *PERF-MIN*   0000
44 *PERF-MAX*   0000
45 *HOURS-AVG* 0003545
46 *HOURS-MIN* 0003545
47 *HPOURS-MAX* 0003545
48 N/A   .00   .00   .00   35.45   35.45   35.45
49 *** END - OF - DATA ***
50 * * *   ROWS READ --> 06
51 The result is ... 33
52 Thanks to [redacted] attending this
53 BRANCHFLAG GREATER THAN 1
54 EXECUTED SEEYA PARAGRAPH
55 EXECUTED SEEYA PARAGRAPH
56 EXECUTED GOODBYE PARAGRAPH
Remote Systems
Team Artifacts
LAB2C
LAB2D
LAB7
LAB8
snippets
To_COPY_and_PASTE.txt
My Favorites
Local Shells
zos.dev
z/OS UNIX Files
z/OS UNIX Shells
MVS Files
TSO Commands
JES
Retrieved Jobs
POT01RUN:JOB00113 [CC 0000]
JES2::JESMSGLG
JES2::JESJCL
JES2::JESYSMSG
CURSRAV4::SYSTSIN [0000]
CURSRAV4::SYSOUT [0000]
CURSRAV4::SYSTSPRT [0000]
POT01CC:JOB00092 [CC 0000]
1
2
3

5.4.7 Use **Ctrl + Shift + F4** to close all editors. Or just click on the . of each opened editor

What have you done so far?
You used **Fault Analyzer** to identify why the program was abending.
You used the **Debugger** to temporarily fix the error caused by the called program REGI0B.
You used IDz to change the program REGI0B to return other value than zero.
You compiled and linked REGI0B creating another version on your load library.
You executed again the JOB but now using the REGI0B that you created and verified that the abend is gone.

Section 6. Using the Code coverage

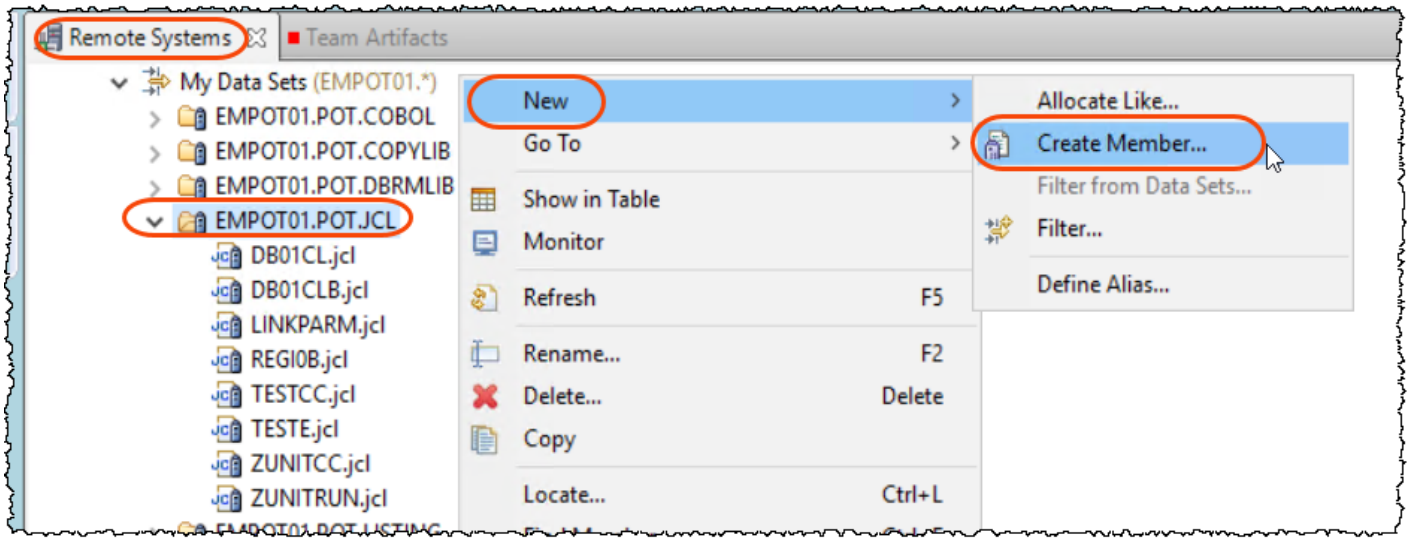
Code coverage analyzes a running program and generates a report of statements that were executed, compared to the total number of executable lines.
Developer for z Systems Host Utilities provides two ways to invoke Code coverage in batch mode, A sample JCL procedure, to process a single program run, and a set of scripts to start and stop a permanently active Code coverage collector that can process multiple program runs.

You can run code coverage for any application you can debug. You can generate code coverage reports that you can view in the workbench or save the results to your file system for future analysis. We will show a simple batch mode invocation.

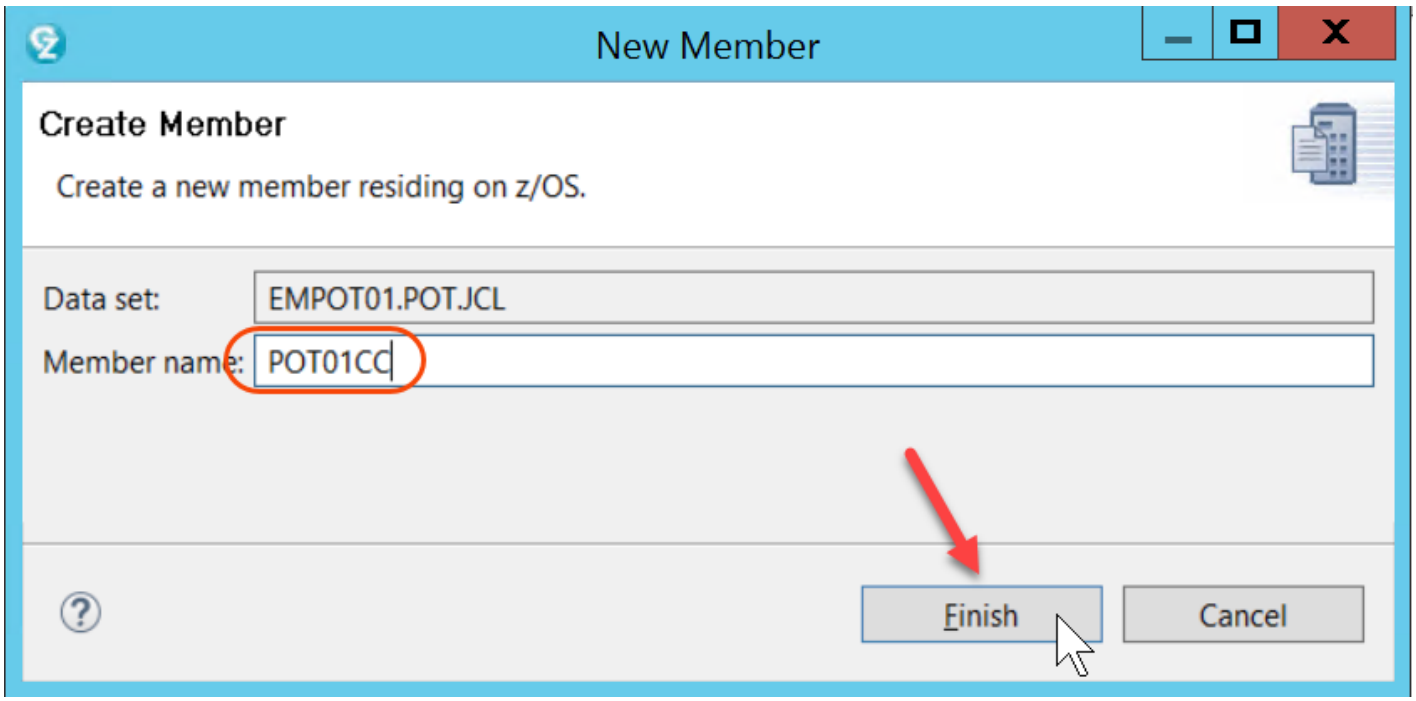
___ 6.1 Creating a JCL to run the code coverage

You will create a JCL file to run Code Coverage

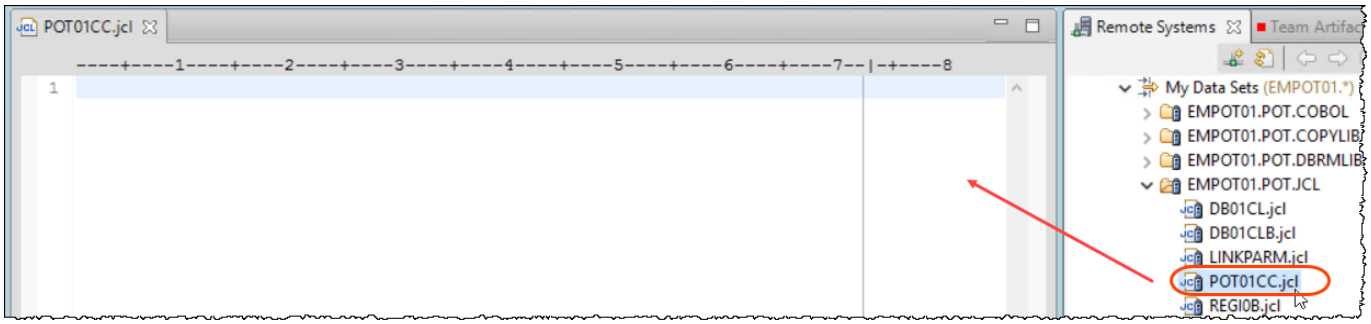
6.1.1 Using Remote Systems view, right click on **EMPOT01.POT.JCL** and select **New > Create Member…**



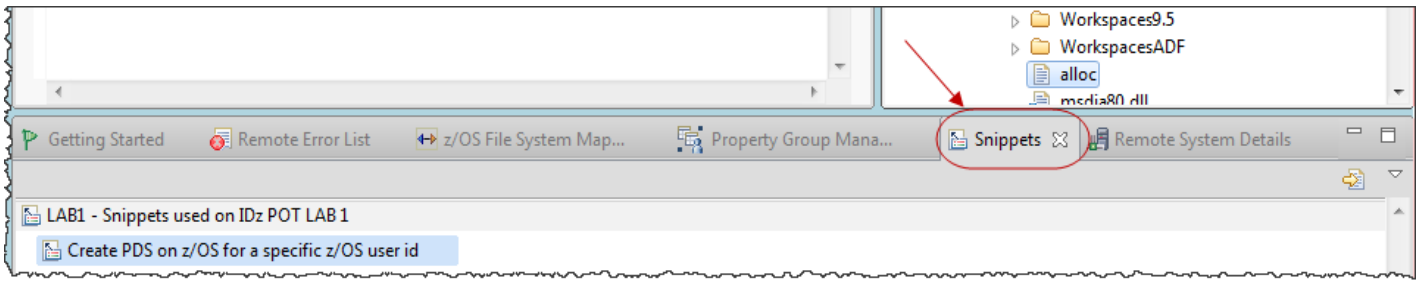
6.1.2 Name it as **POT01CC** and click **Finish**



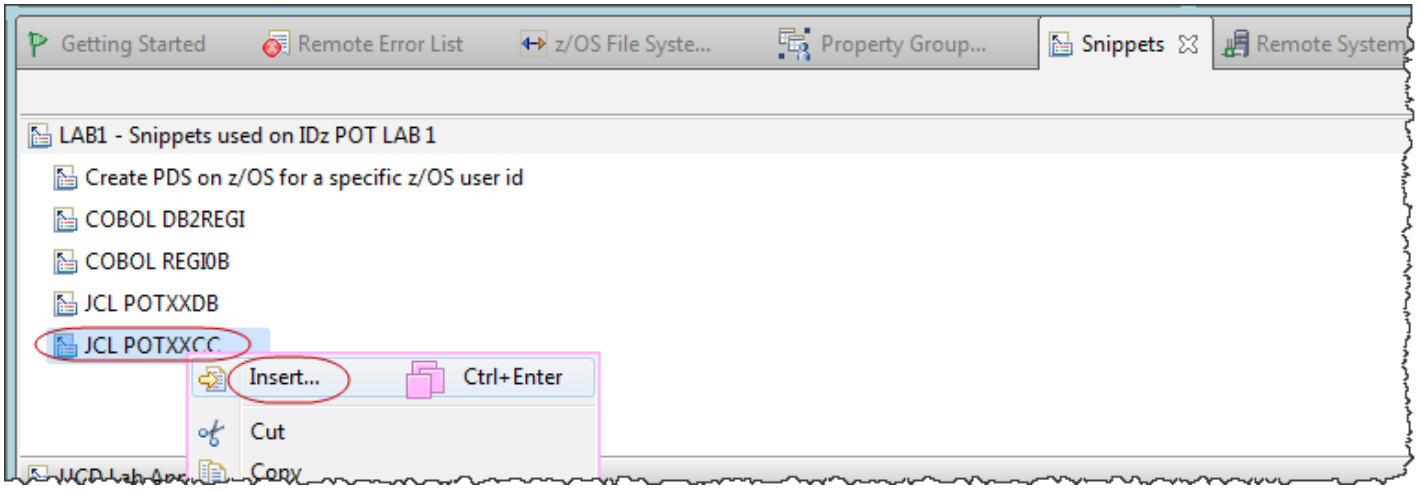
6.1.3 Double click on **POT01CC** to edit it



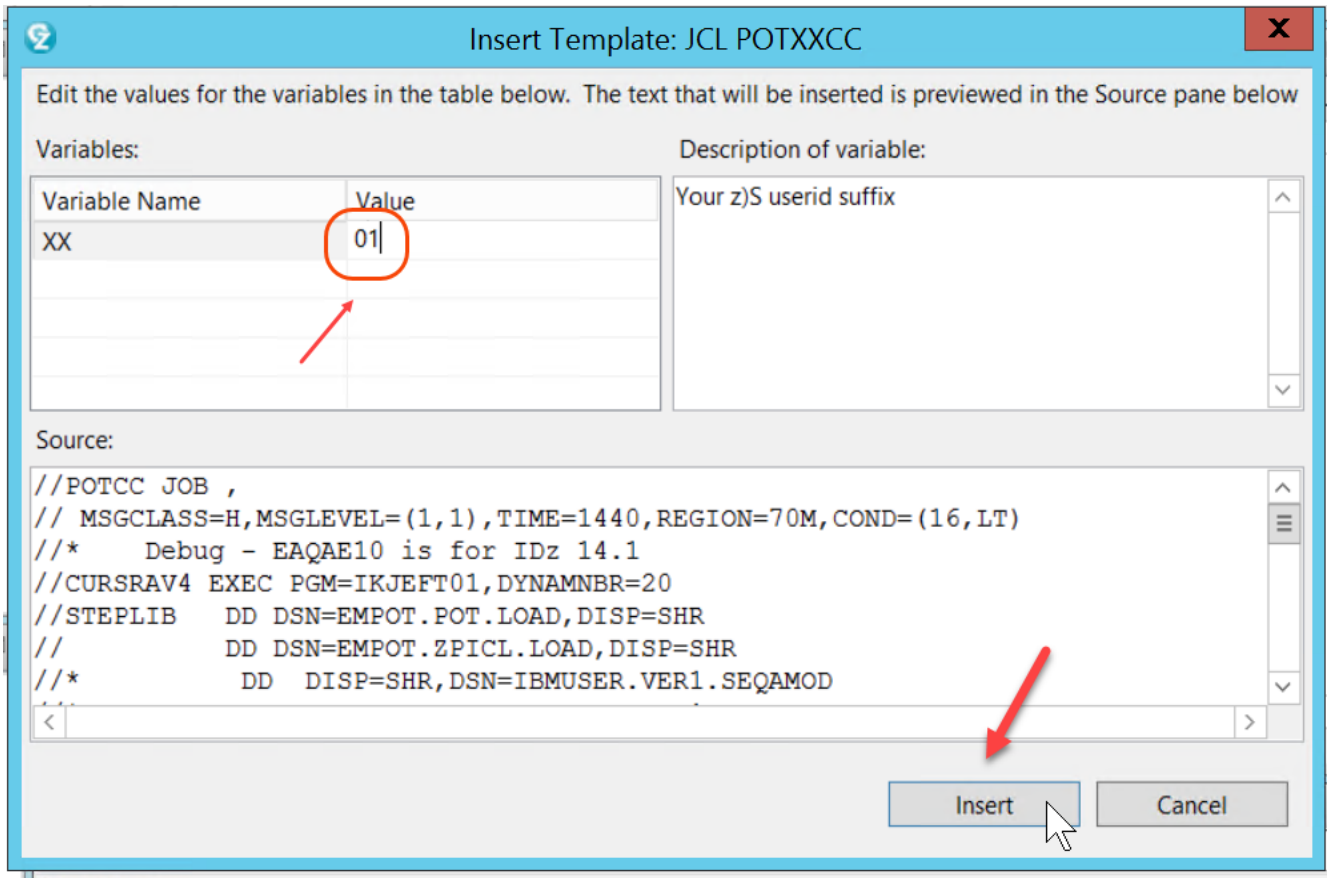
6.1.4 Click on **Snippets** tab on the bottom...



6.1.5 Right click on **JCL POTXXCC** and select **Insert…**



6.1.6 When the *Insert* dialog opens, type **01** in the *Value* and click **Insert**



6.1.7 . Notice that the Snippets variables will be replaced.

⏭ Use **Ctrl + S** to save the changes.

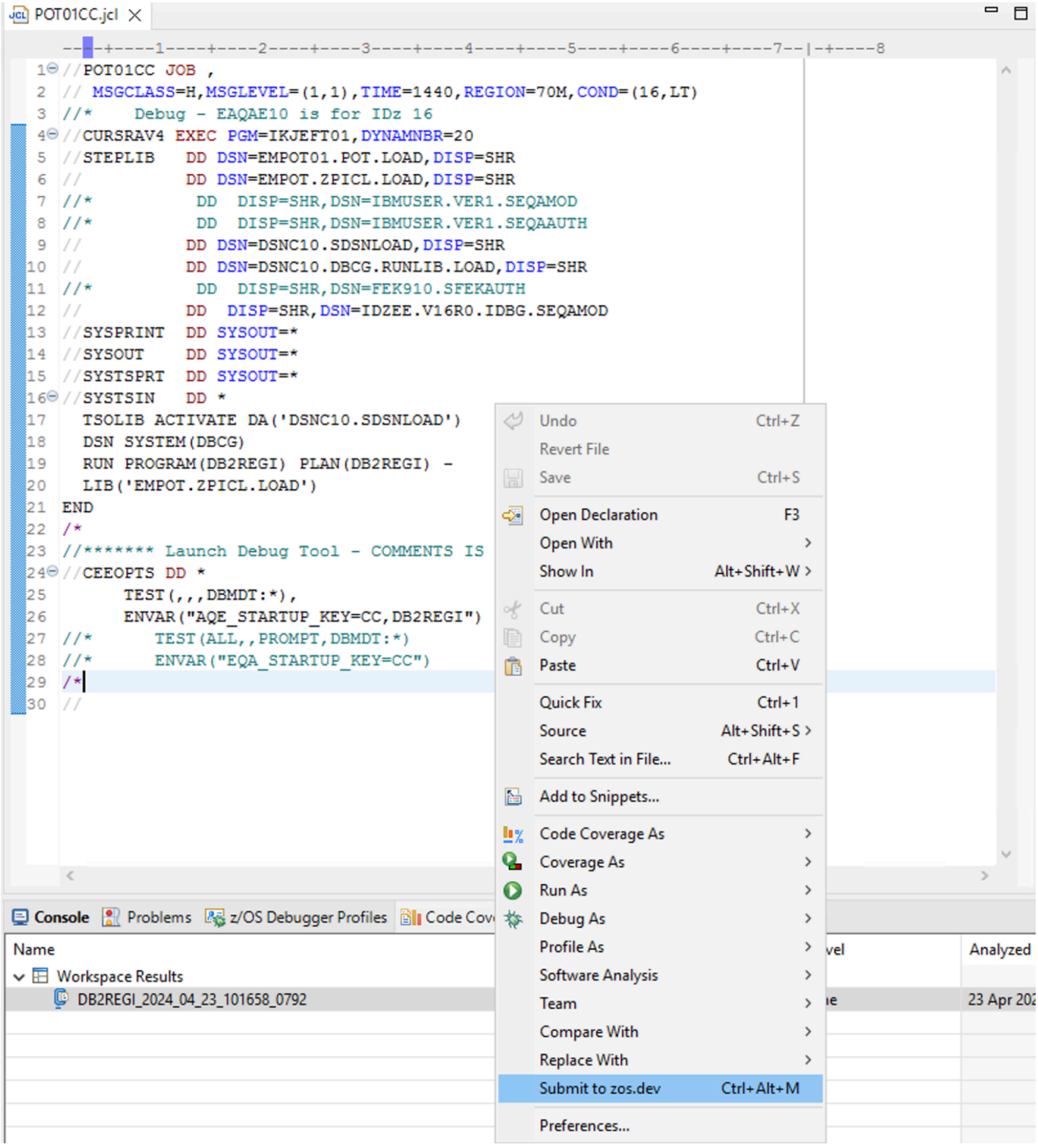
*POT01CC.jcl

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
//POT01CC JOB ,
// MSGCLASS=H,MSGLEVEL=(1,1),TIME=1440,REGION=70M,COND=(16,LT)
//*    Debug - EAQAE10 is for IDz 16
//CURSRAV4 EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB   DD DSN=EMPOT01.POT.LOAD,DISP=SHR
//          DD DSN=EMPOT.ZPICL.LOAD,DISP=SHR
//*          DD  DISP=SHR,DSN=IBMUSER.VER1.SEQAMOD
//*          DD  DISP=SHR,DSN=IBMUSER.VER1.SEQAAUTH
//          DD DSN=DSNC10.SDSNLOAD,DISP=SHR
//          DD DSN=DSNC10.DBCG.RUNLIB.LOAD,DISP=SHR
//*          DD  DISP=SHR,DSN=FEK910.SFEKAUTH
//          DD  DISP=SHR,DSN=IDZEE.V16R0.IDBG.SEQAMOD
//SYSPRINT  DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//SYSTSPRT  DD SYSOUT=*
//SYSTSIN   DD *
  TSOLIB ACTIVATE DA('DSNC10.SDSNLOAD')
  DSN SYSTEM(DBCG)
  RUN PROGRAM(DB2REGI) PLAN(DB2REGI) -
  LIB('EMPOT.ZPICL.LOAD')
END
/*
//******* Launch Debug Tool - COMMENTS IS FOR IDz 16
//CEEOPTS DD *
      TEST(,,,DBMDT:*),
      ENVAR("AQE_STARTUP_KEY=CC,DB2REGI")
//*      TEST(ALL,,PROMPT,DBMDT:*)
//*      ENVAR("EQA_STARTUP_KEY=CC")
/*
//
```

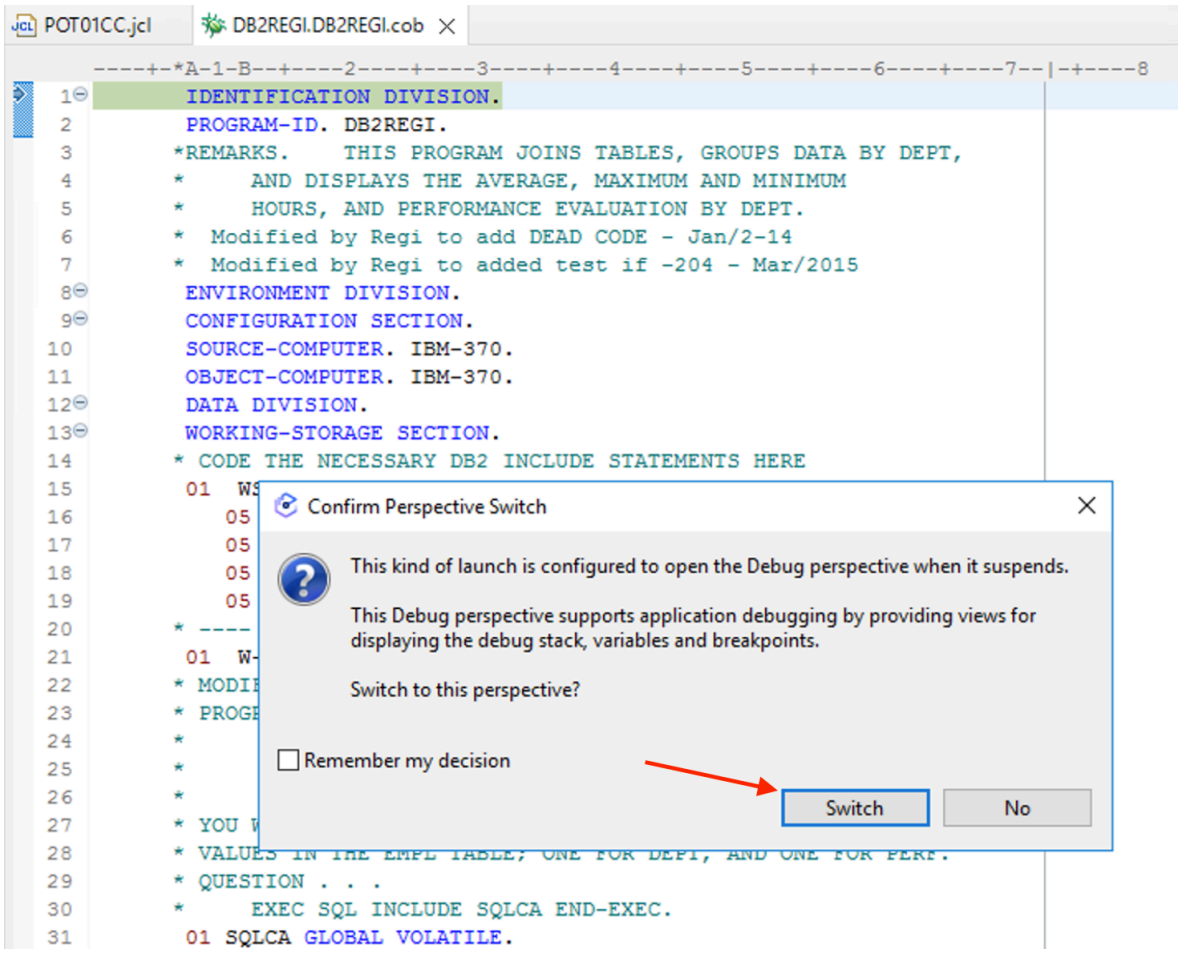
___ 6.2 Submit the JCL for z/OS execution

You will now execute the fixed batch COBOL/DB2 on z/OS

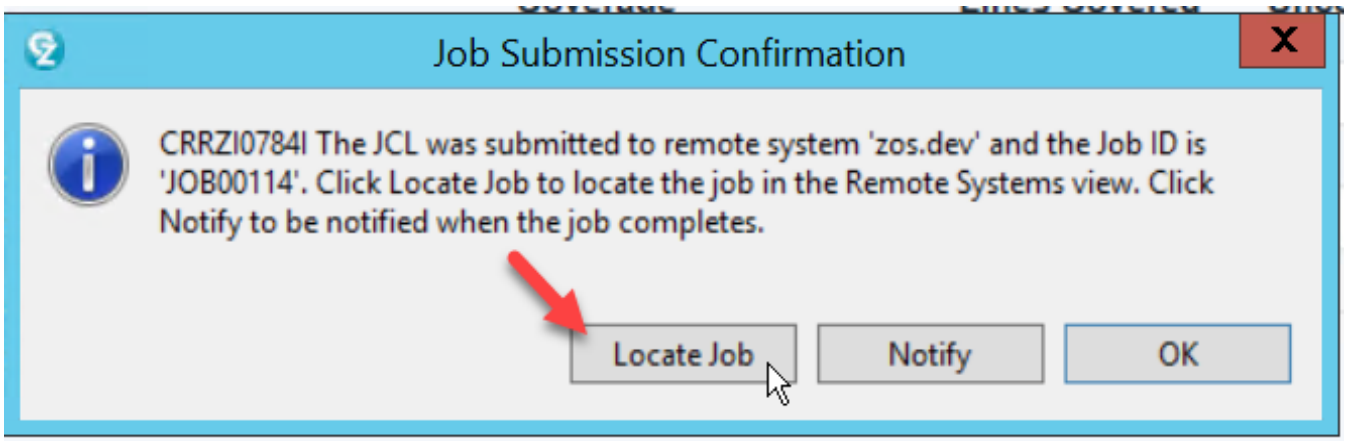
6.2.1 ⏭ **Right click** on the JCL being edited and select **Submit to zos.dev**



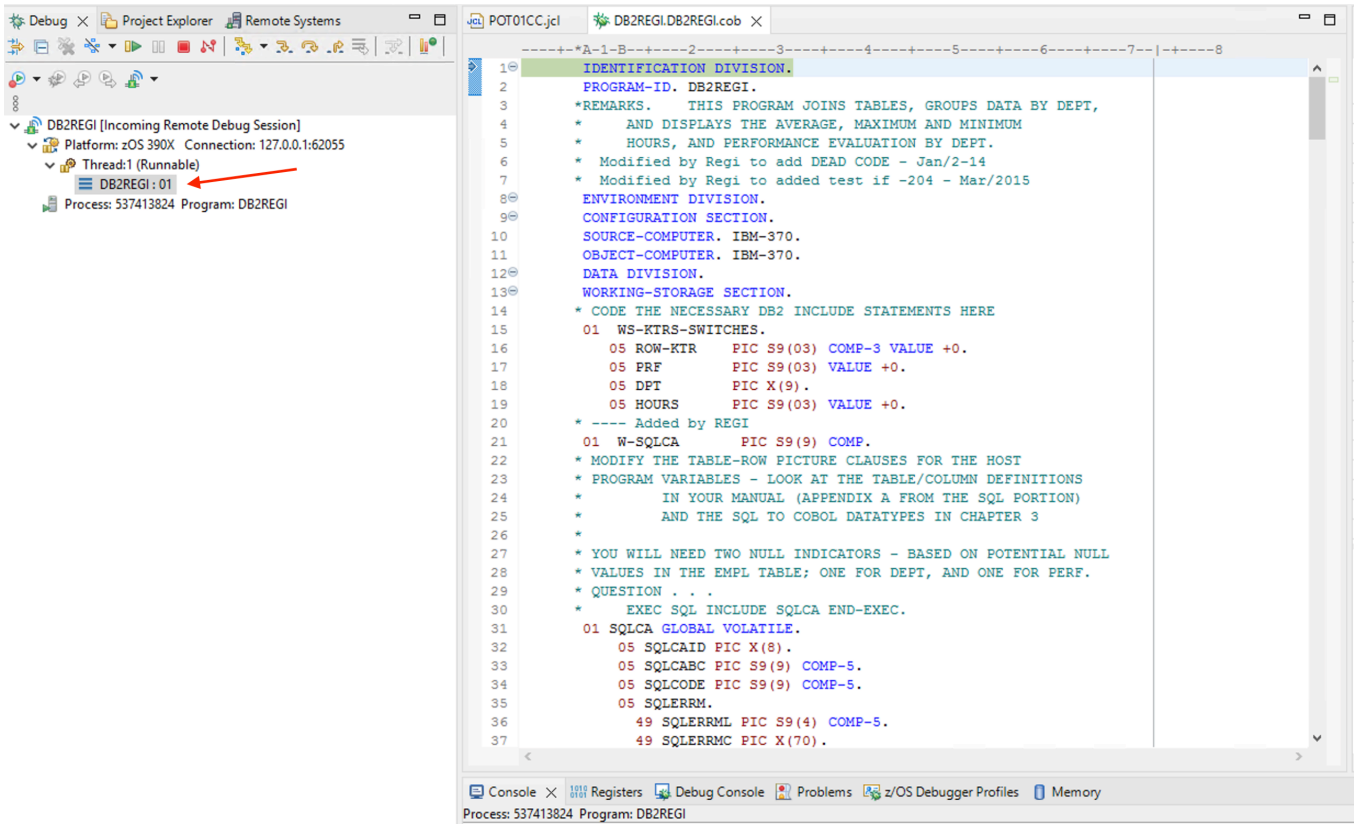
6.2.2 When the dialog pops up, select **Switch**



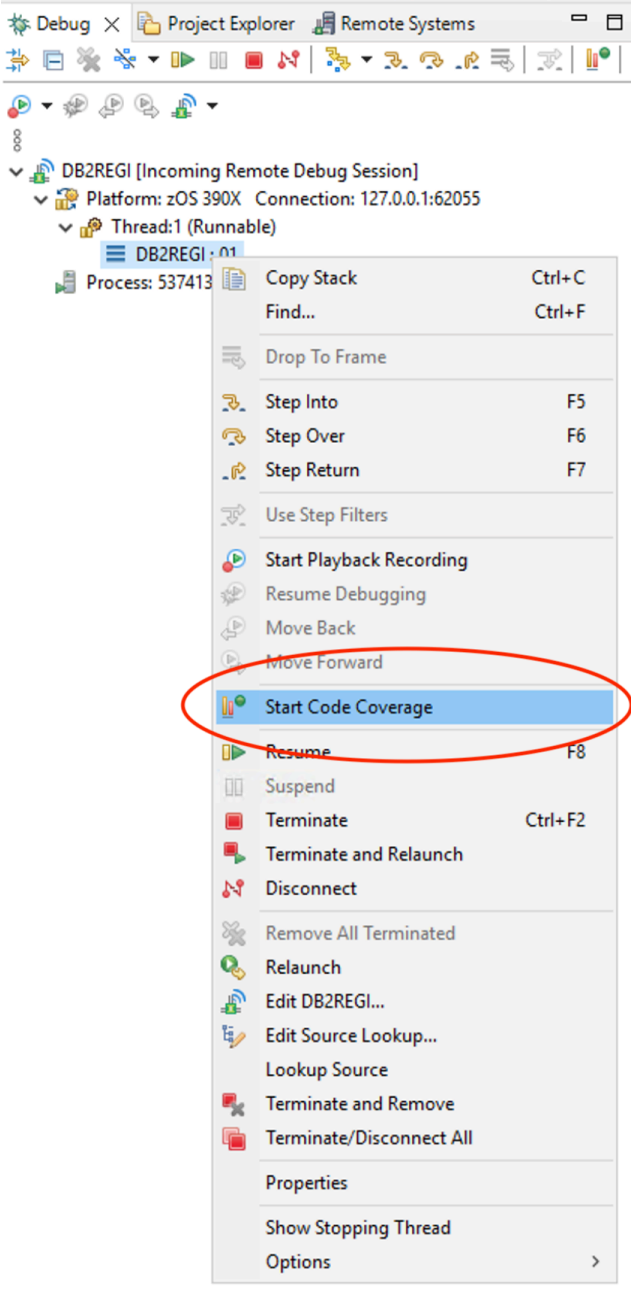
6.2.3 When the dialog pops up, select **Locate Job**



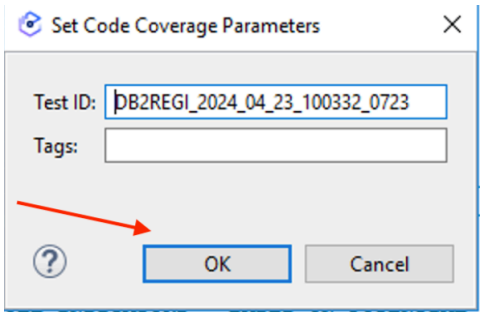
6.2.4 Debug will be start and the COBOL PGM will show up your right. See on your left **DB2REGI:01 Thread.**



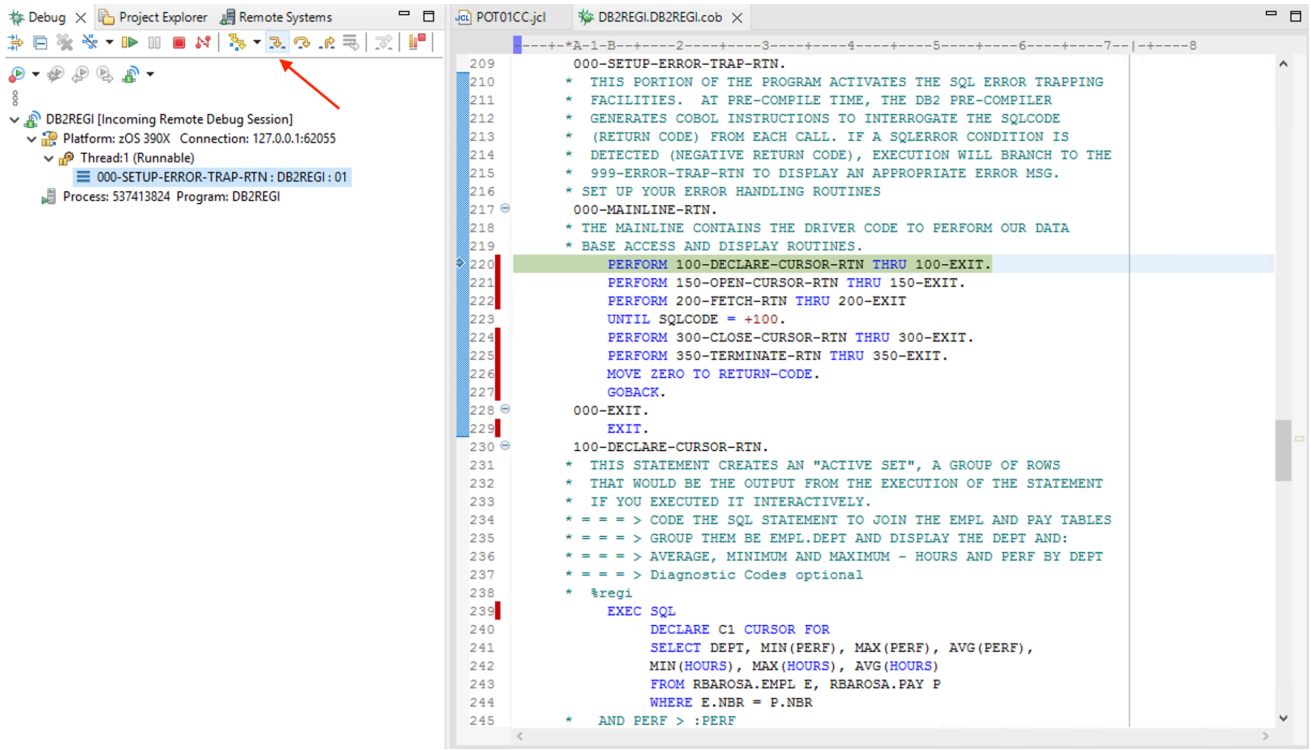
6.2.5 **Right Click** on **DB2REGI:01 Thread** and select **Start Code Coverage**.



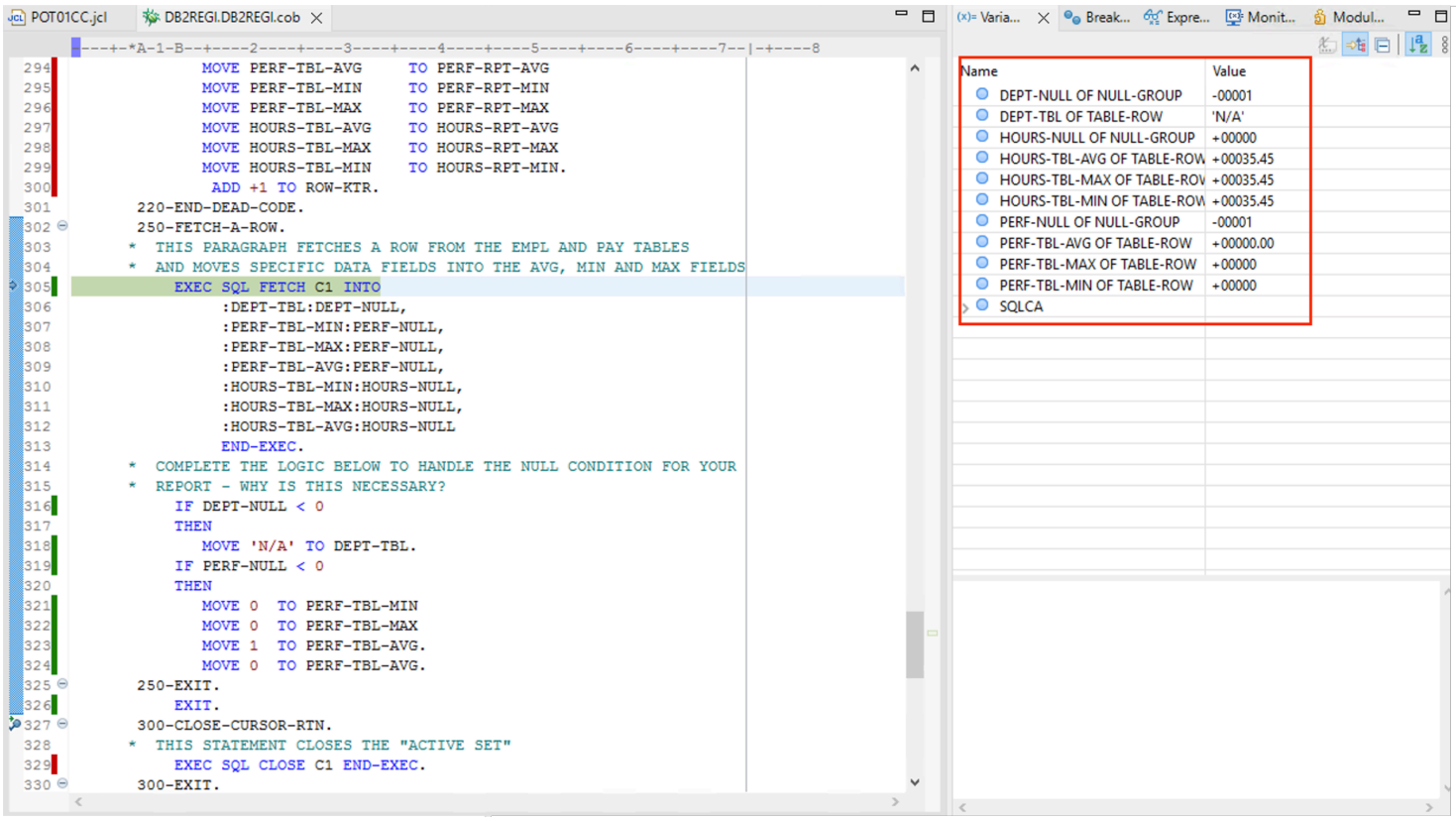
6.2.6 **Set Code Coverage Parameters** window will show up. Just Click **OK**.



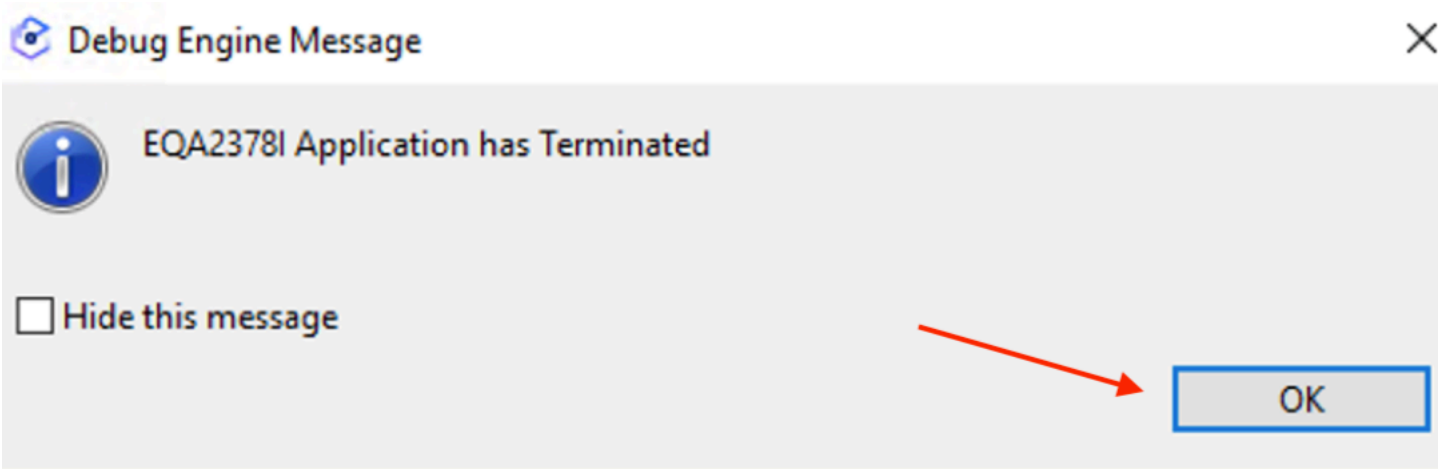
6.2.7 Click on **icon** (or Press **F5**). You will see a RED Bars changing to GREEN Bars. It means the code is coveraging. As you can see the error routine on the picture above is not tested..
Imagine how bad if you go to production and never test this condition ....



6.2.8 Take a look at on your last window (**Variables**). You can follow the values of the variables.

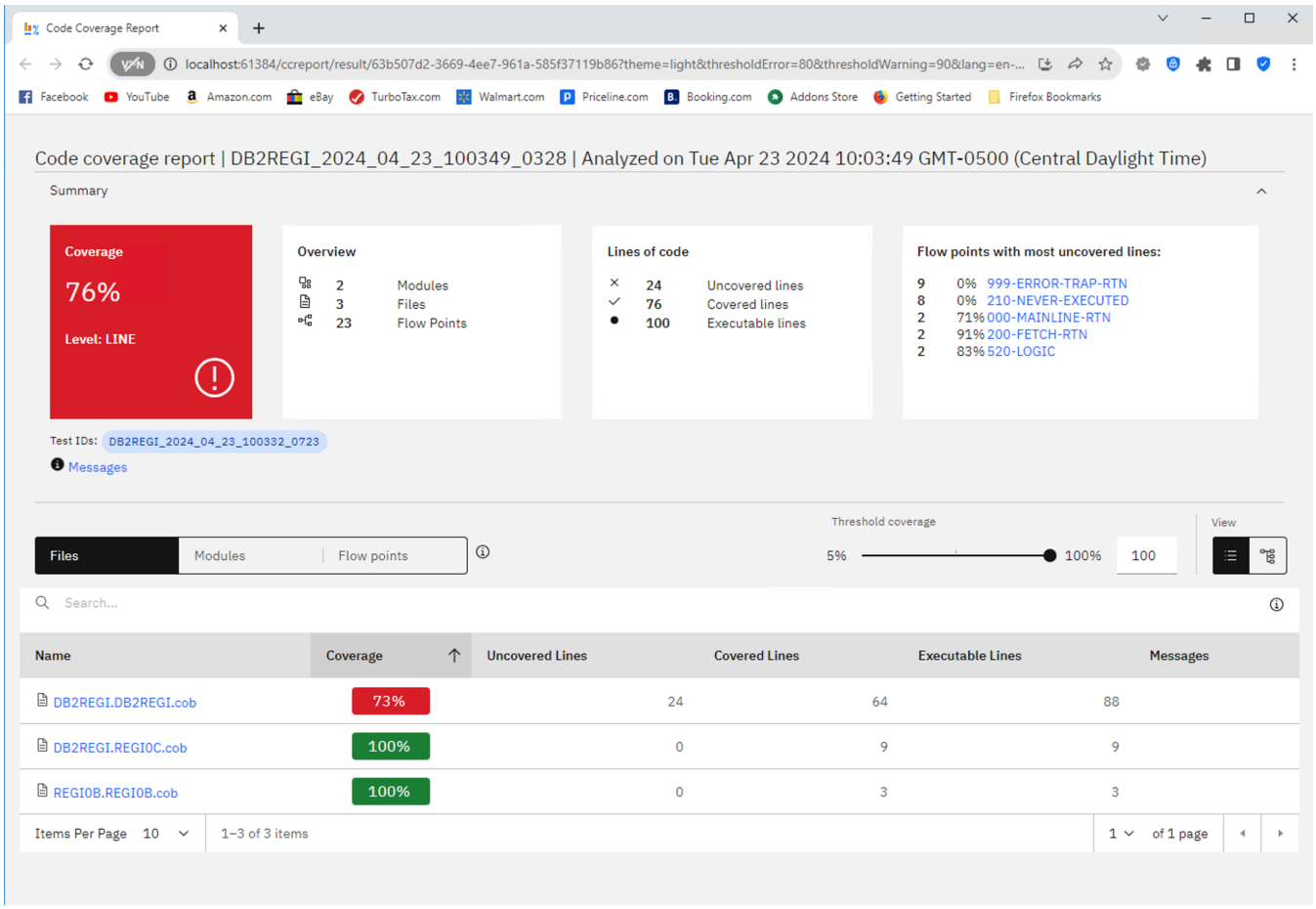


6.2.9 ⏭ Click on ▶ or press **F8** until you see **Application has Terminated** window and 9 ⏭ Click OK.



6.2.10 The code coverage report will be created.You can see that this test covered **76%.** Take a look at the programs whch have coverade. The percentage of executable lines that are covered.

- **Coverage**: The percentage of executable lines that are covered. The coverage level can either be LINE or FLOWPOINT.
- **Overview**: Shows the numbers of modules, files, and flow points.
- **Lines of code**: Shows the numbers of uncovered lines, covered lines, and executable lines
- **Flow points** with most uncovered lines: The lowest-level flow points with most uncovered lines are listed. You can click the flow point to open the source or listing in an editor.

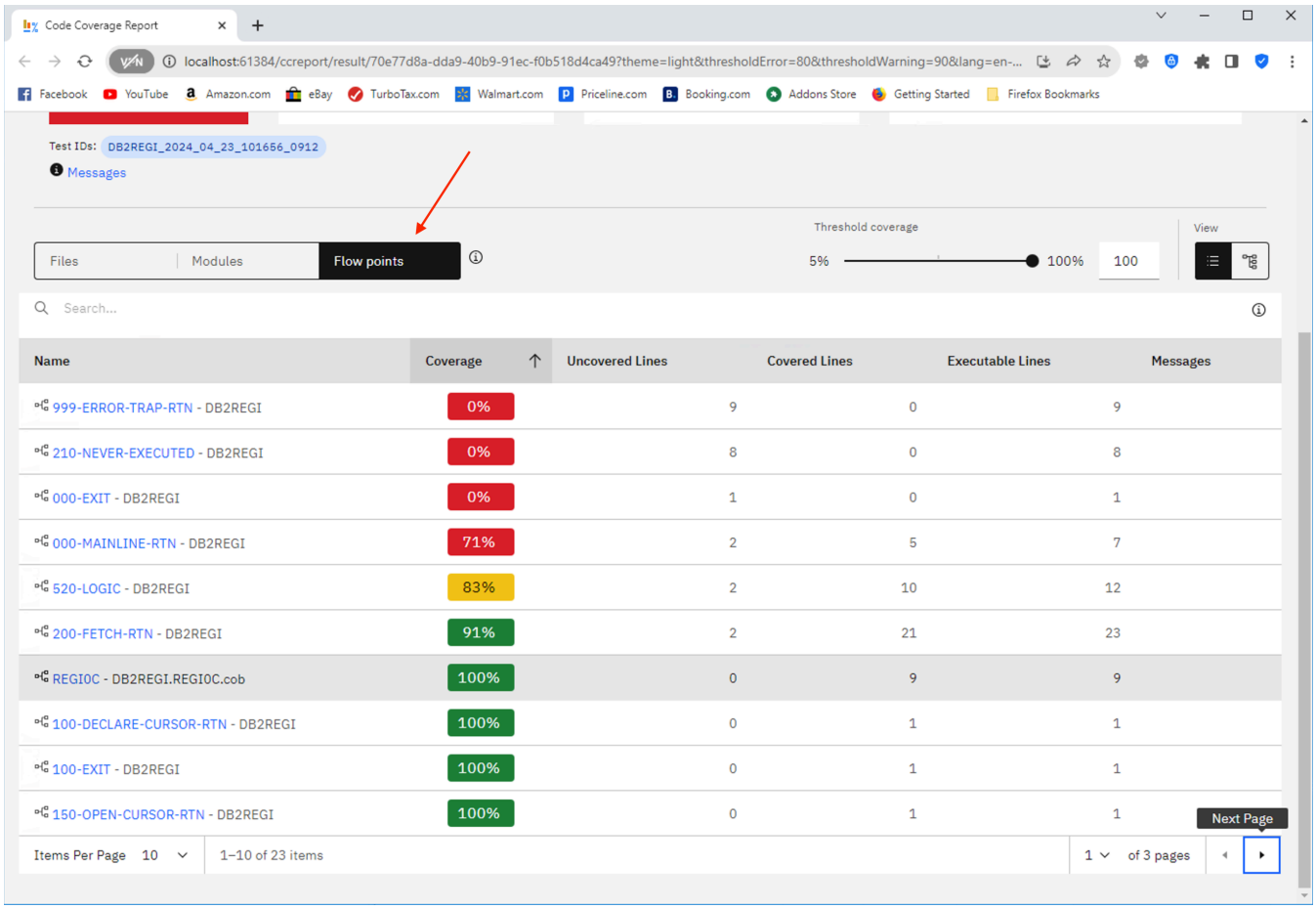


6.2.11 In the code coverage report, Click on **Flow Points** icon.

The coverage percentage is highlighted with a colored bar to indicate the status according to the thresholds you specified on the File Analysis Defaults preference page that can be opened by clicking the drop-down menu icon in the Code Coverage Results view.

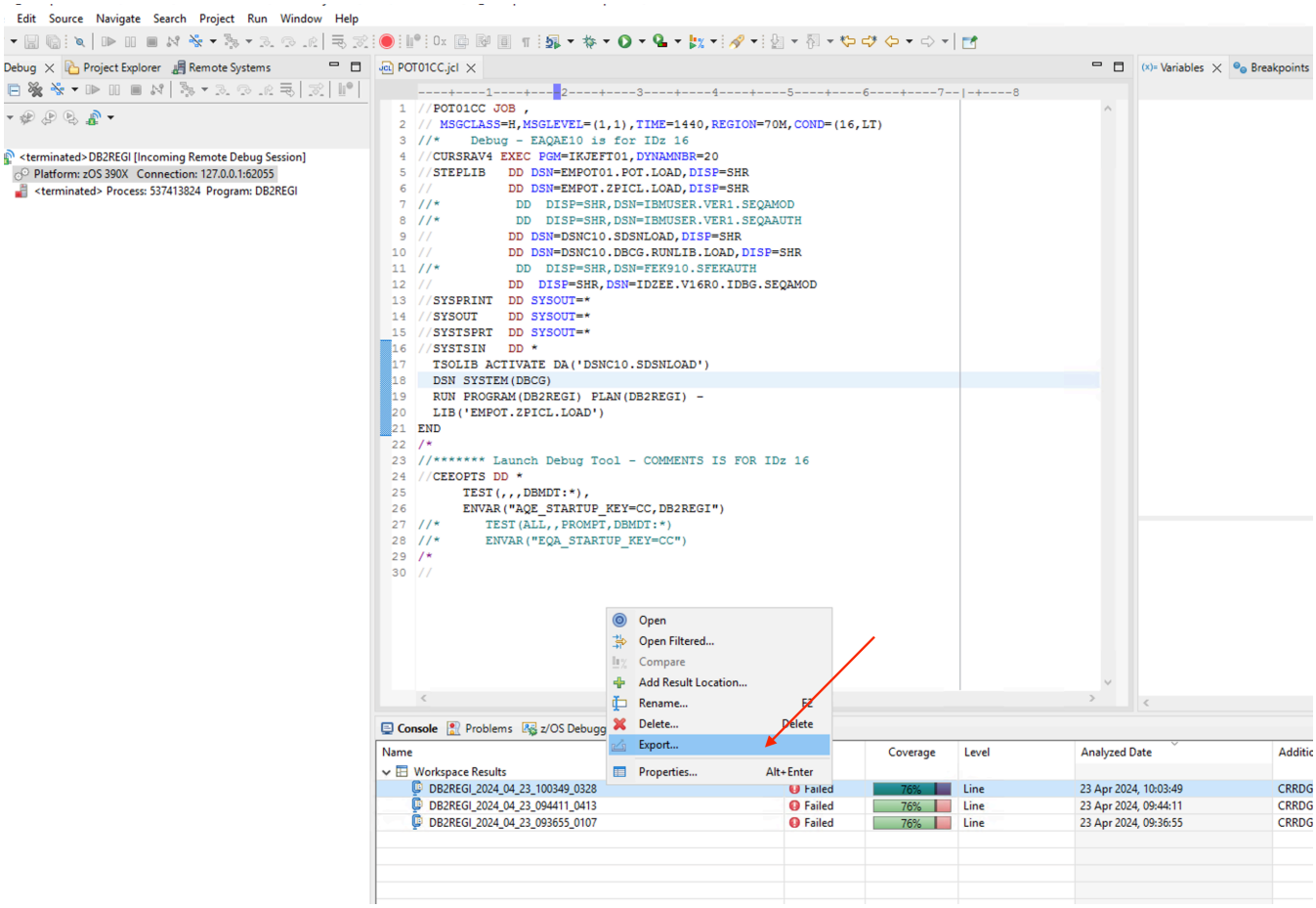
- Passed (green bar): the coverage percentage is greater than or equal to the warning threshold.
- Warning (yellow bar): the coverage percentage is smaller than the warning threshold, but larger than or equal to the failure threshold. By default, the warning threshold is 90.
- Failed (red bar): the coverage is smaller than the failure threshold. By default, the failure threshold is 80.

***Note You can explore the Coverage Report, navigatingon the page.**



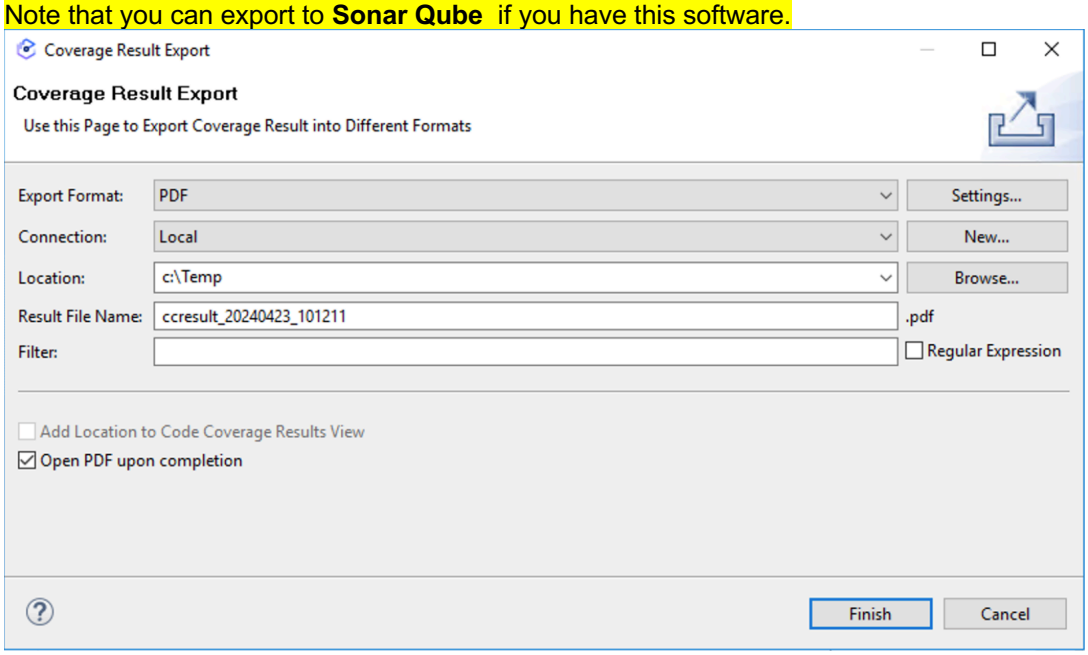
6.2.12 Back to IDZ, on **Debug Perpective**. On the **Code Coverage Results** tab, **Right Click** on the result and select **Export.** You could also create a PDF report as documentation or export this data to other products (like ADDI or SonarQube)
In the Code Coverage Results view, you can check the status of a result:

- If the coverage percentage is smaller than the failure threshold, the Failed icon is displayed in the Status column.
- If the coverage percentage level is smaller than the warning threshold, but greater than or equal to the failure threshold, the Warning icon is displayed in the Status column.
- If the coverage percentage is greater than or equal to the warning threshold, the Passed icon is displayed.

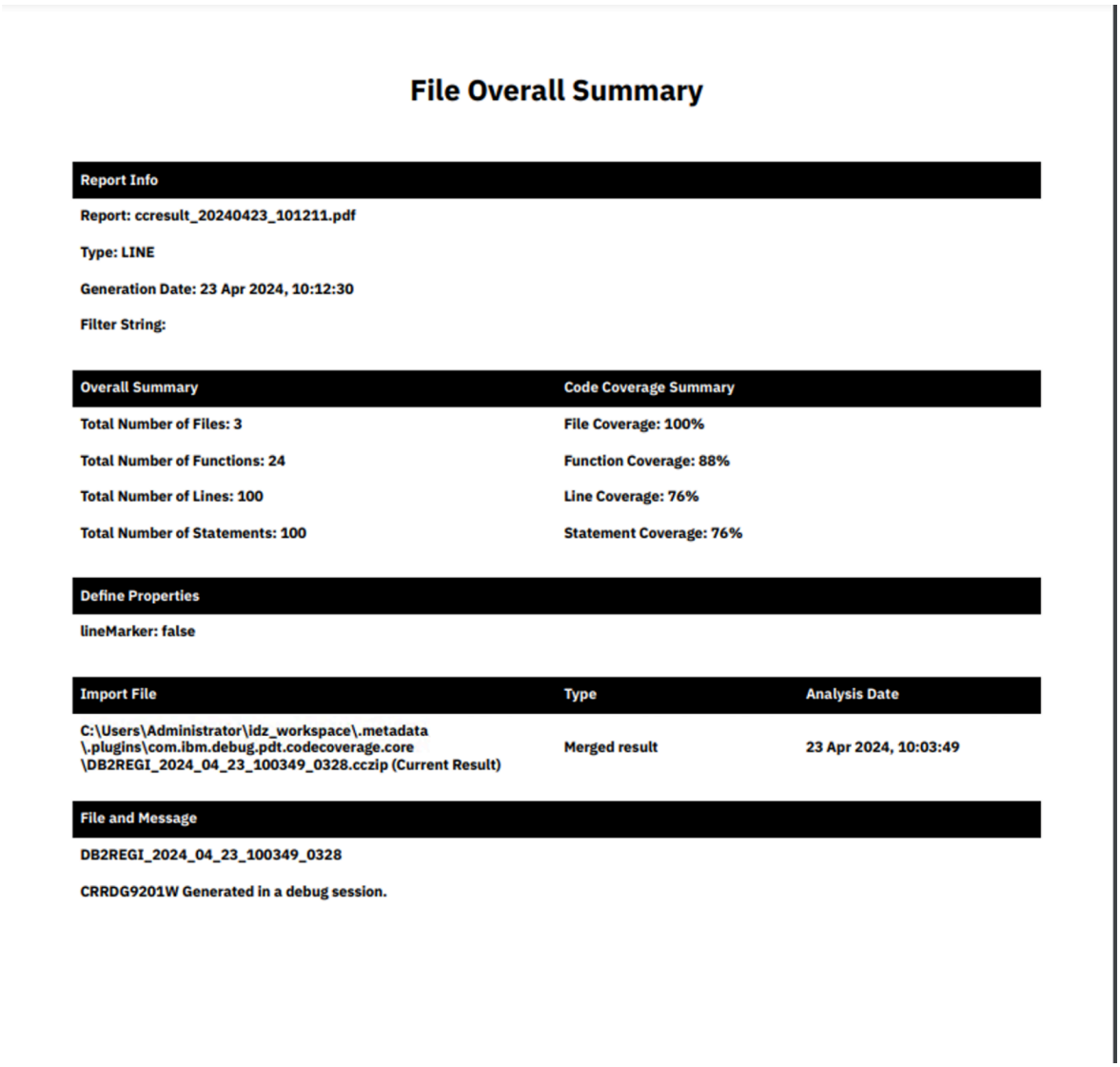


6.2.13 Select **PDF**, **C:\Temp, Open PDF upon completion** and **Finish**.

Note that you can export to **Sonar Qube** if you have this software.



6.2.14 A PDF report will be generated..



File Overall Summary

Report Info
Report: ccresult_20240423_101211.pdf
Type: LINE
Generation Date: 23 Apr 2024, 10:12:30
Filter String:

Overall Summary	Code Coverage Summary
Total Number of Files: 3	File Coverage: 100%
Total Number of Functions: 24	Function Coverage: 88%
Total Number of Lines: 100	Line Coverage: 76%
Total Number of Statements: 100	Statement Coverage: 76%

Define Properties
lineMarker: false

Import File	Type	Analysis Date
C:\Users\Administrator\idz_workspace\.metadata \.plugins\com.ibm.debug.pdt.codecoverage.core \DB2REGI_2024_04_23_100349_0328.cczip (Current Result)	Merged result	23 Apr 2024, 10:03:49

File and Message
DB2REGI_2024_04_23_100349_0328
CRRDG9201W Generated in a debug session.

6.2.15 Close all editors pressing **Ctrl + Shift + F4** Or just click on the . of each opened editor**.**

Congratulations! You have completed the Lab 1.

LAB 2 - Create UrbanCode Deploy infrastructure and deploy to z/OS

Updated June,2024 (revised by Carlos Hirata/Wilbert Kho, Reviewed by Joe Huang and Frank Hernandez)

Abstract:

You will create and deploy an existing COBOL CICS application using UrbanCode Deploy to z/OS

This Lab has 3 parts.

Part 1 – Create the UrbanCode Deploy application infrastructure.
In this section, you will design the actual deployment process; you need to prepare the application infrastructure in UrbanCode Deploy (UCD). First, you will create a component and add component version, then you will define resources and map them to an environment, and finally you will create an application and add environment and component to it.

Part 2 - Create the UrbanCode Deploy deployment processes.
On this part you will create a deployment process for your component and the application process that uses the component process to deploy the component.

Part 3 – Deploy the application to z/OS CICS
On this part you will deploy the Application to the z/OS CICS.

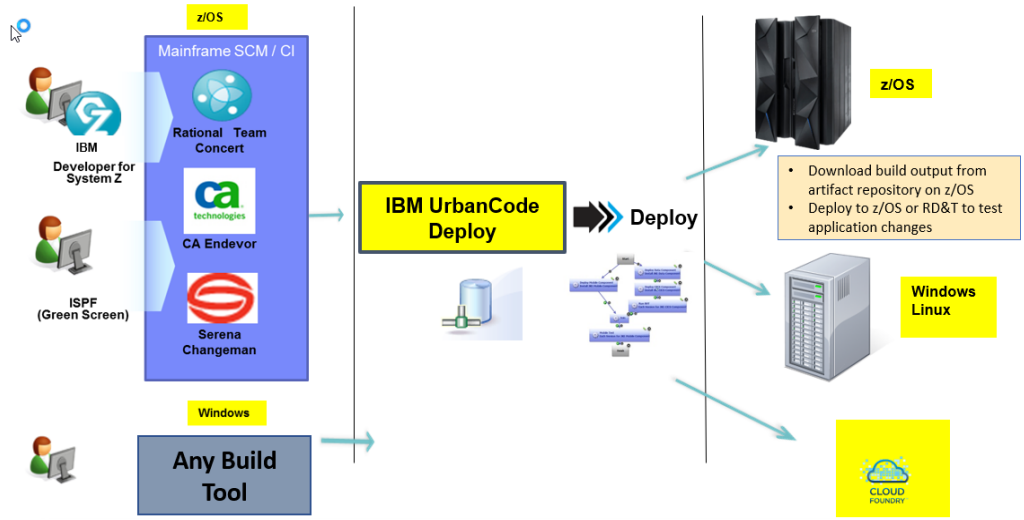
Each time you see the symbol ⏭ it means that you have to "do" something on your computer – not merely read the document.

Lab Architecture and proposed scenario

The architecture used in this lab consists of.
1. The **z/OS Server** running on the ZDT that has CICS running and the UCD z/OS agent.
2. The **UrbanCode Deploy Server** that is running on Linux and on the same machine that has the z/OS (ZDT)

Below the architecture of each cloud instance

Below is a picture that describes UCD:

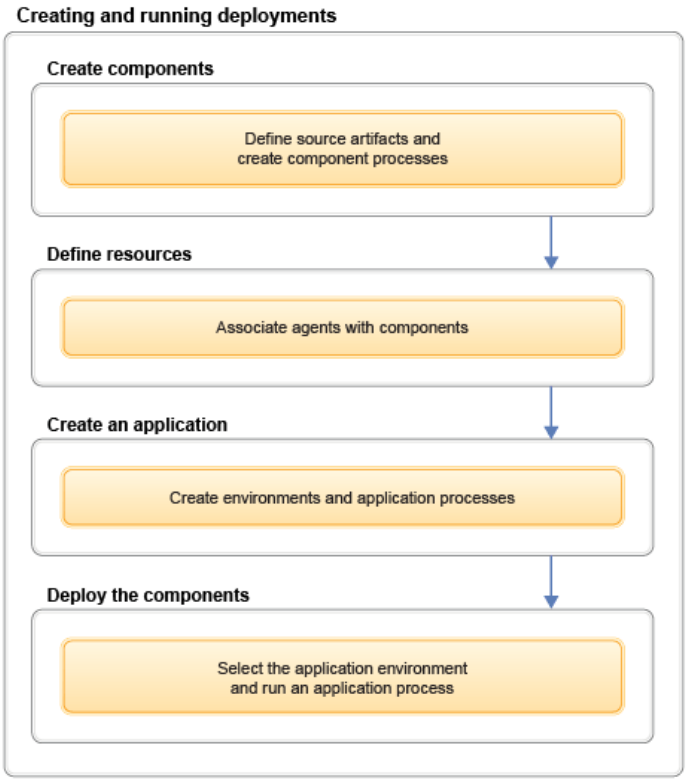


PART 1 – Create the UrbanCode application infrastructure

In this section, you will design the actual deployment process; you need to prepare the application infrastructure in UCD.

First, you will create a component and add component version, then you will define resources and map them to an environment, and finally you will create an application and add environment and component to it

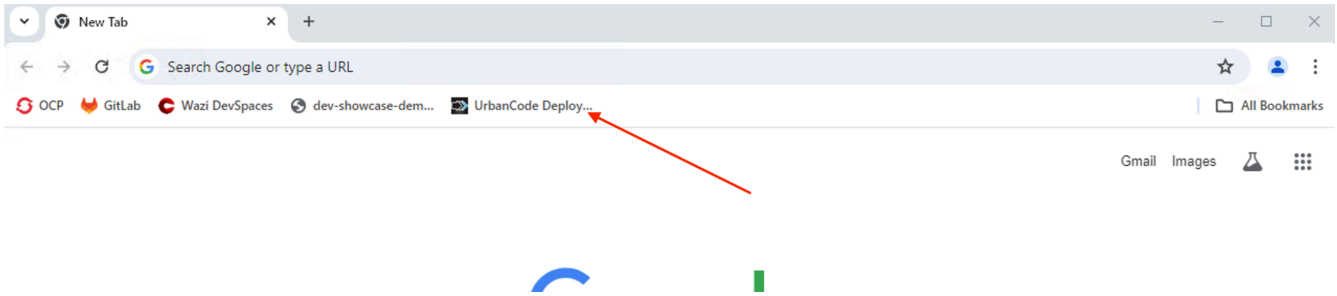
The main steps are as follows:



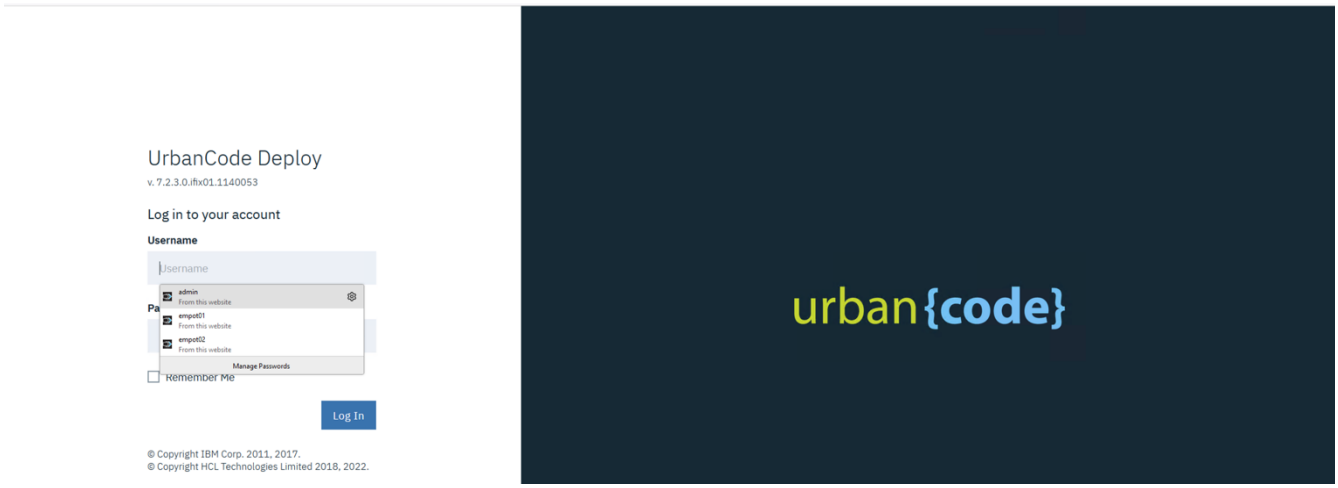
Before start UCD lab, let's verify UCD agent (zOS) is connect to UCD server (Linux Server).

0.1.1 Open firefox browser.

0.1.2 ⏭ Look at Bookmark toolbar and select **UrbanCode Deploy**

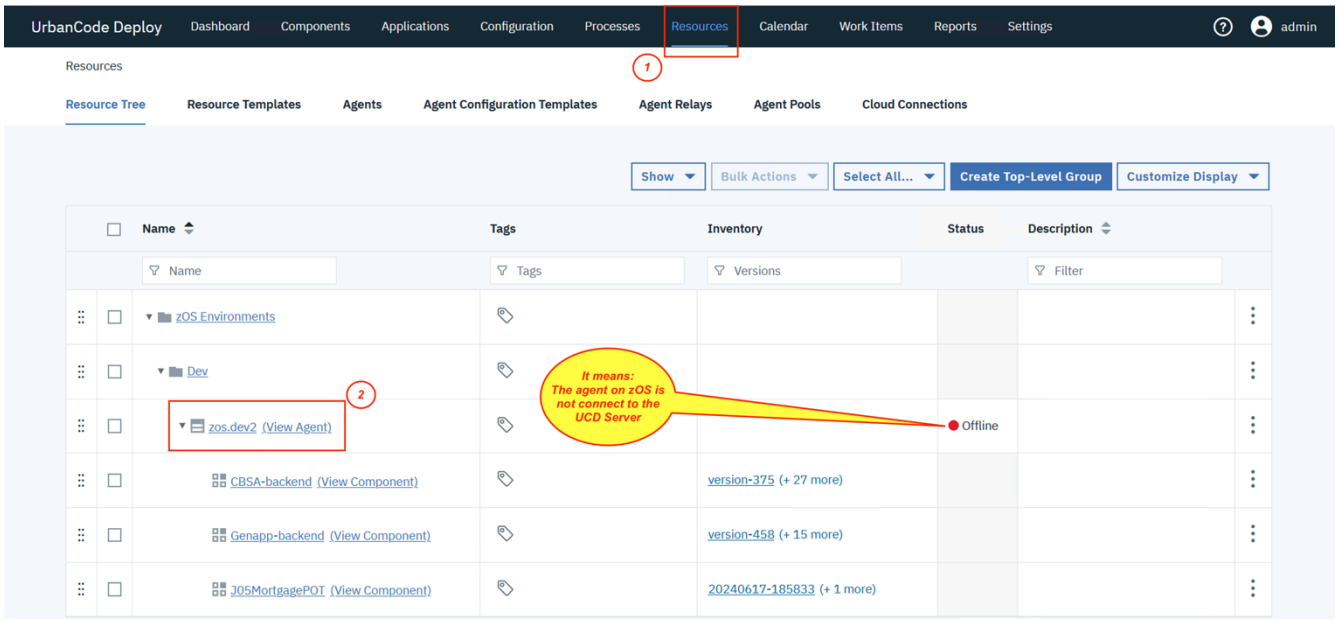


0.1.3. ⏭ Login with **`admin`** and password **`admin`**

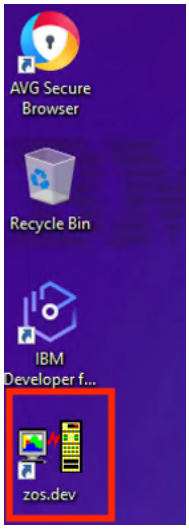


0.1.4 On Urbancode page, locate Resources tab. Look at the agent (zos.dev2) and check the status.

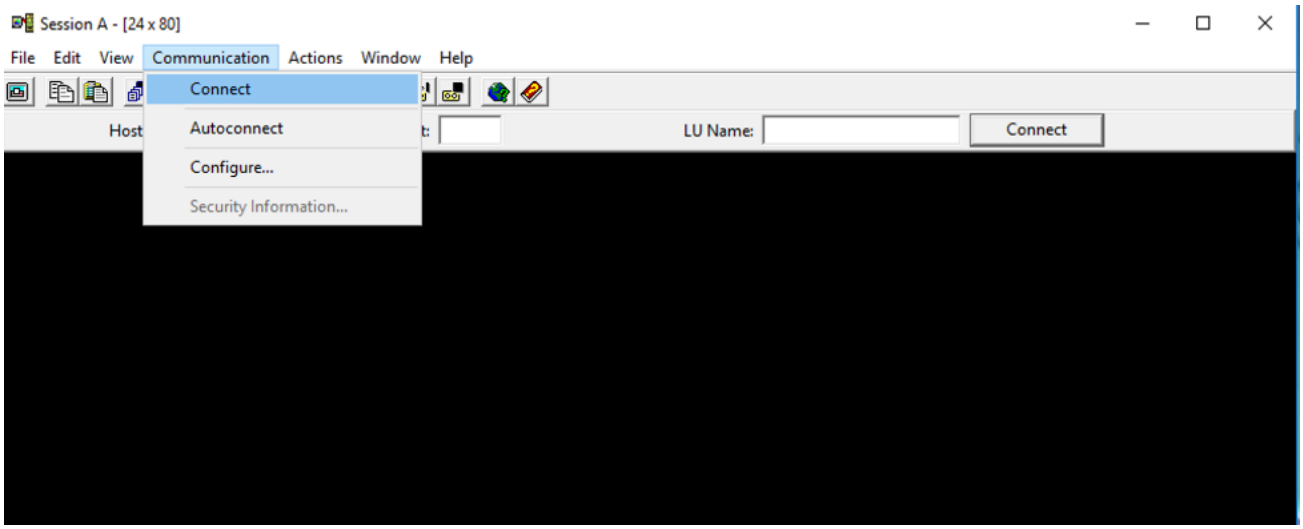
If the status is Online, it is good and skip to Task 1 in this lab. If the status is Offline. You must start the agent on zos.



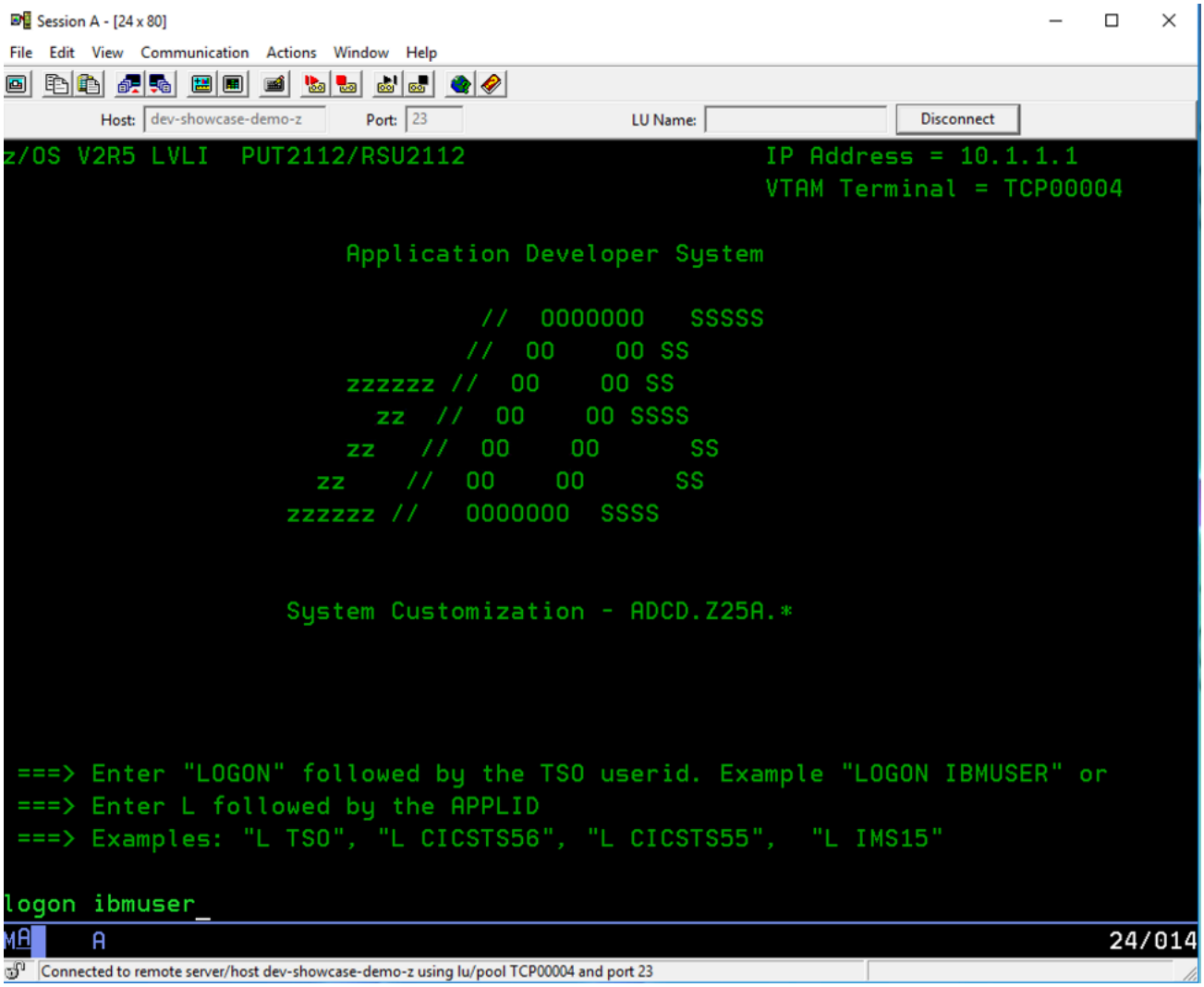
0.1.5 On Windows Desktop, click ⏭ zos.dev icon and open PCMM application.



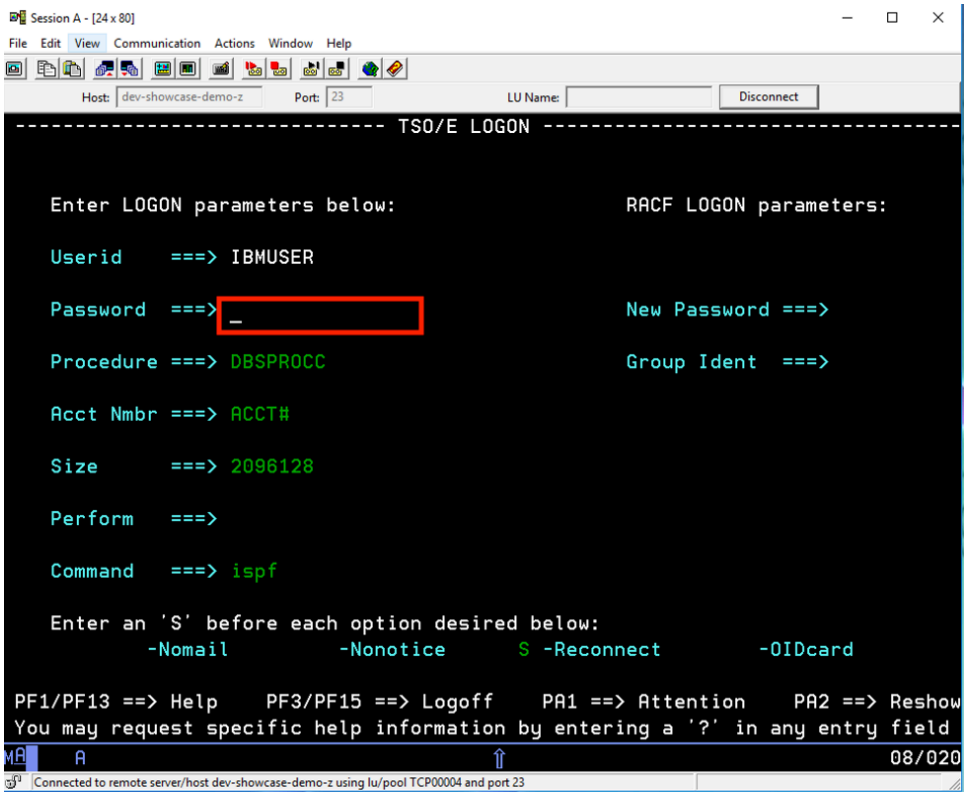
0.1.6 Click ⏭ on Communication → Connect



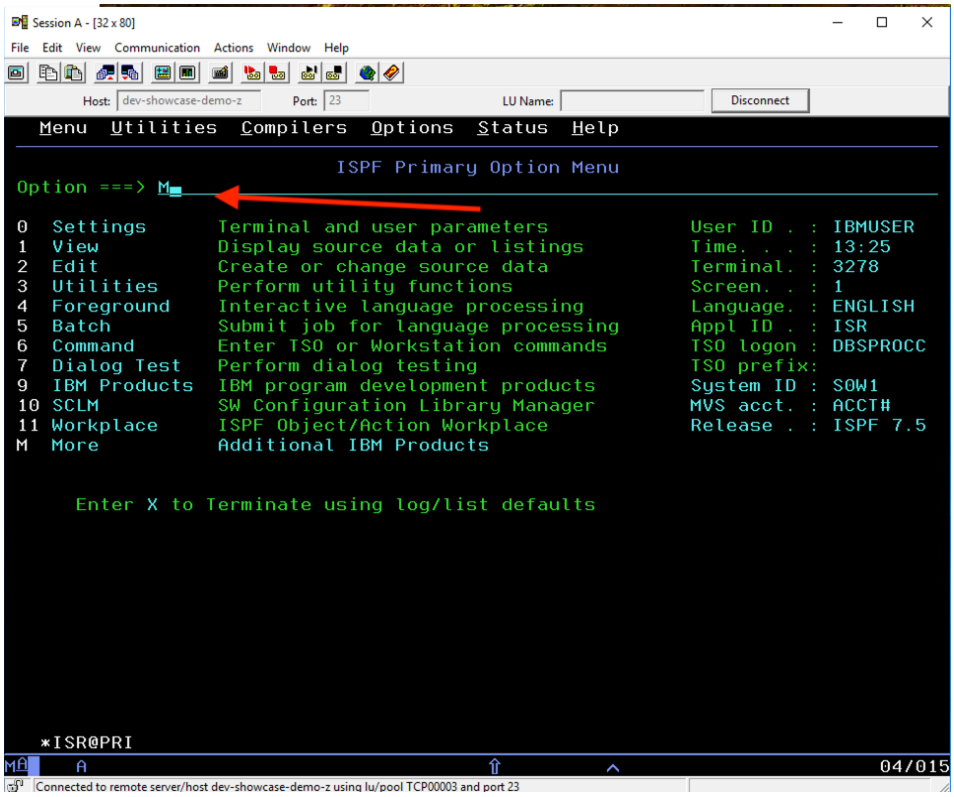
0.1.7 The z/OS main screem will show up and enter logon ibmuser (it can be lower case).



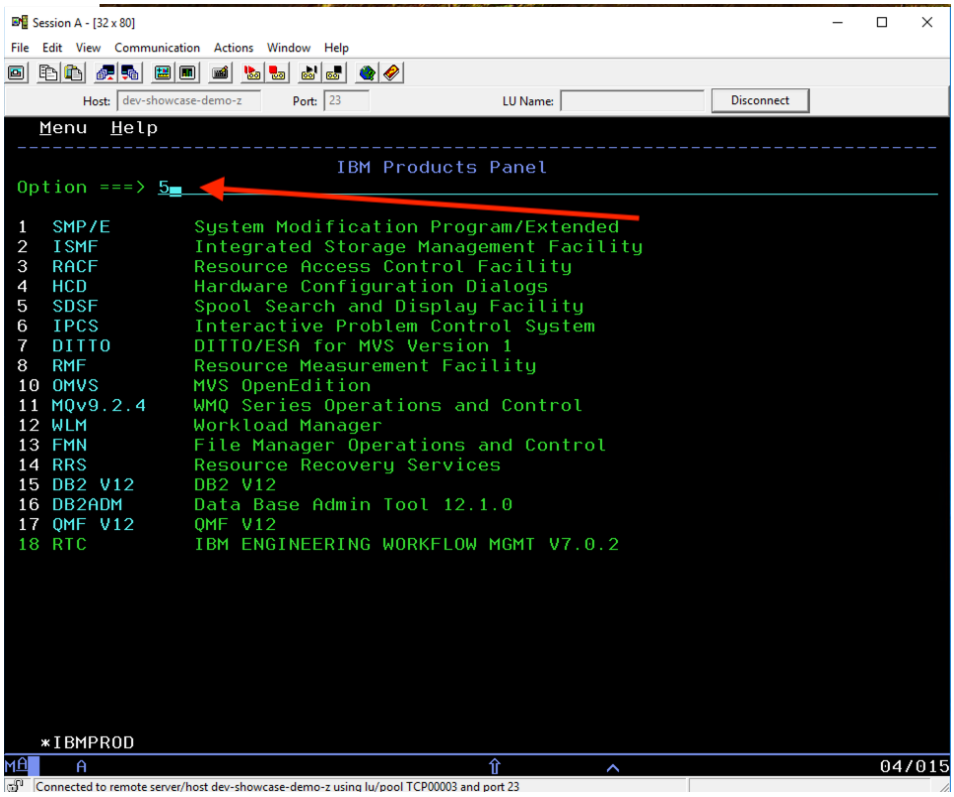
0.1.8 The userid is IBMUSER and the password is **sys1.** Press **ENTER** and **ENTER.**



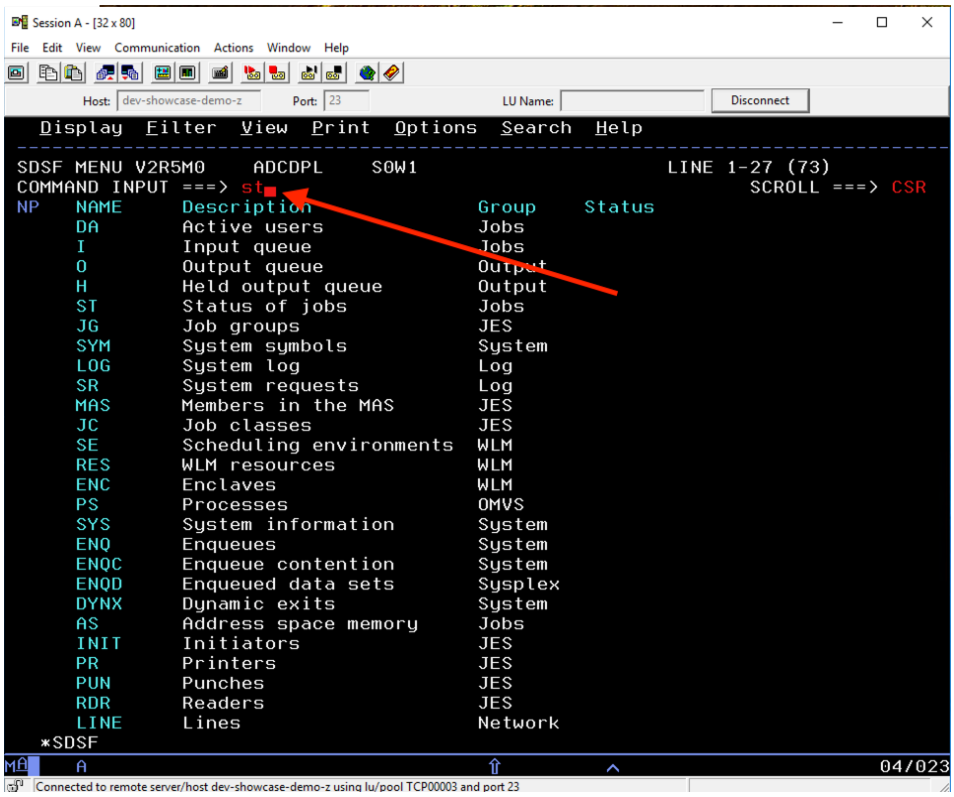
0.1.9 On ISPF Primary Menu. Enter the letter **M** on **Option** field.Press **ENTER.**



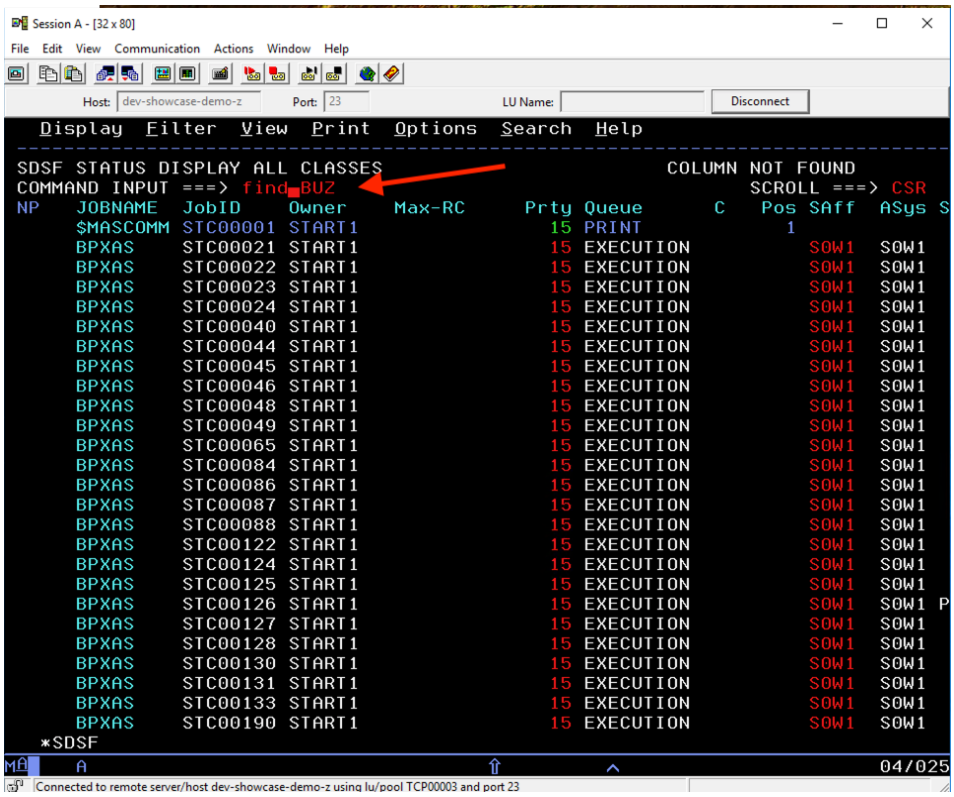
0.1.10 On IBM Products Panel. Enter the number **5** to go to **Spool Search and Display Facility.**



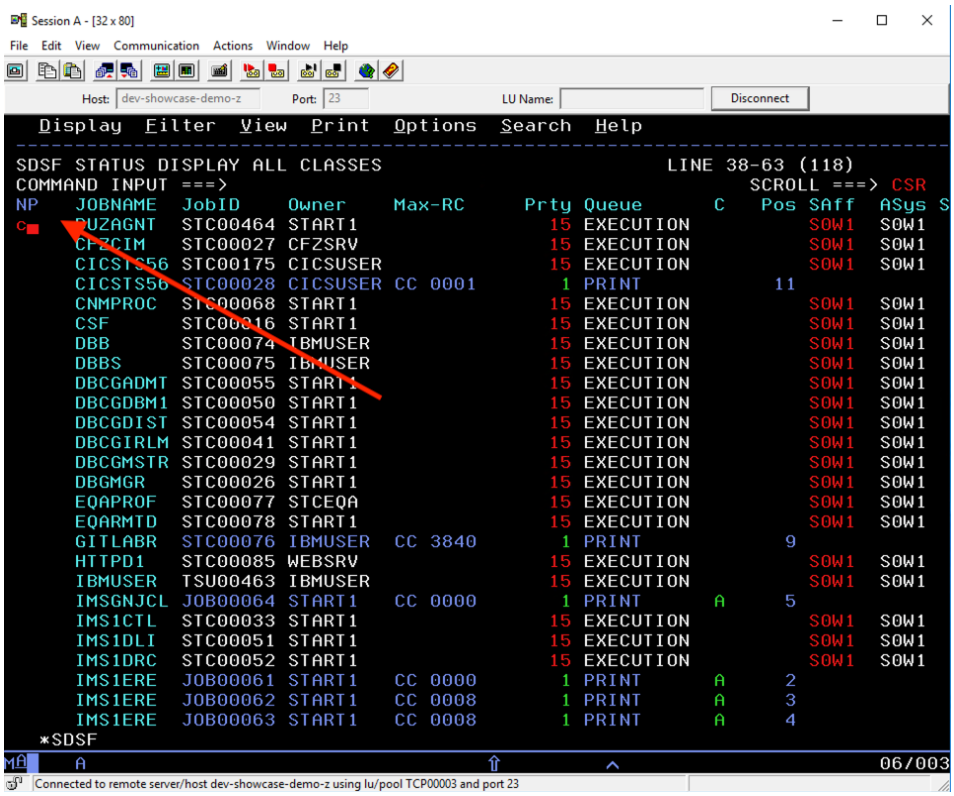
0.1.11 On **SDSF Menu.** Type **st** command on **COMMAND INPUT** and press **ENTER.**



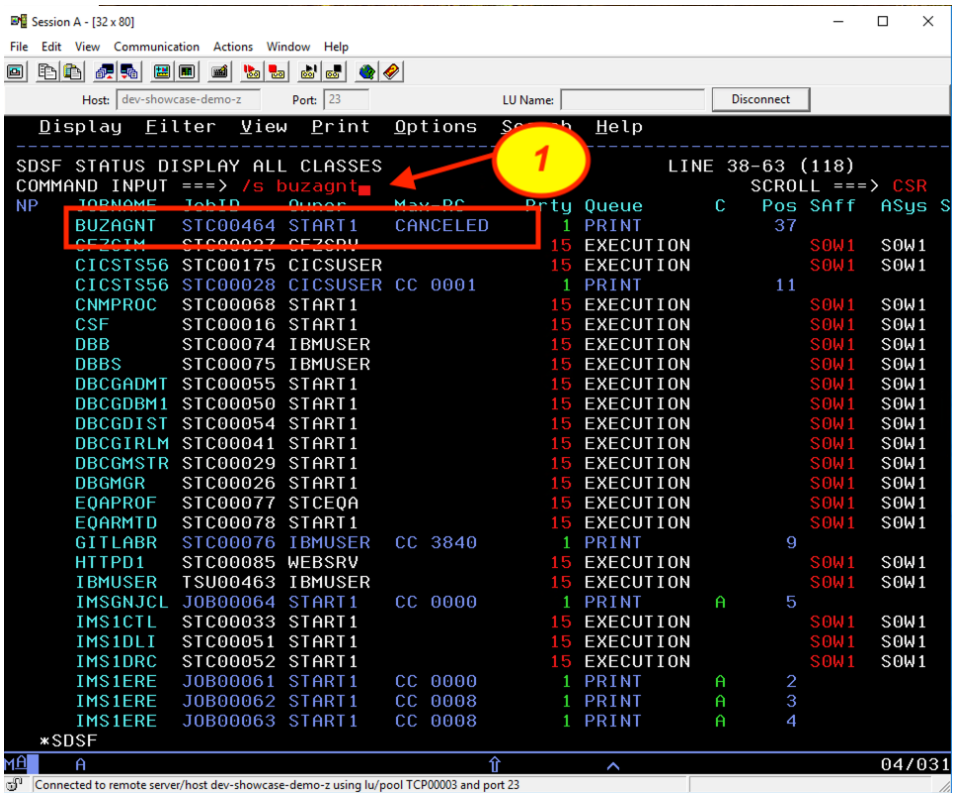
0.1.12 On **SDSF STATUS DISPLAY ALL CLASSES,** type **find buz (**3 letters is enough**)** and press **ENTER.**



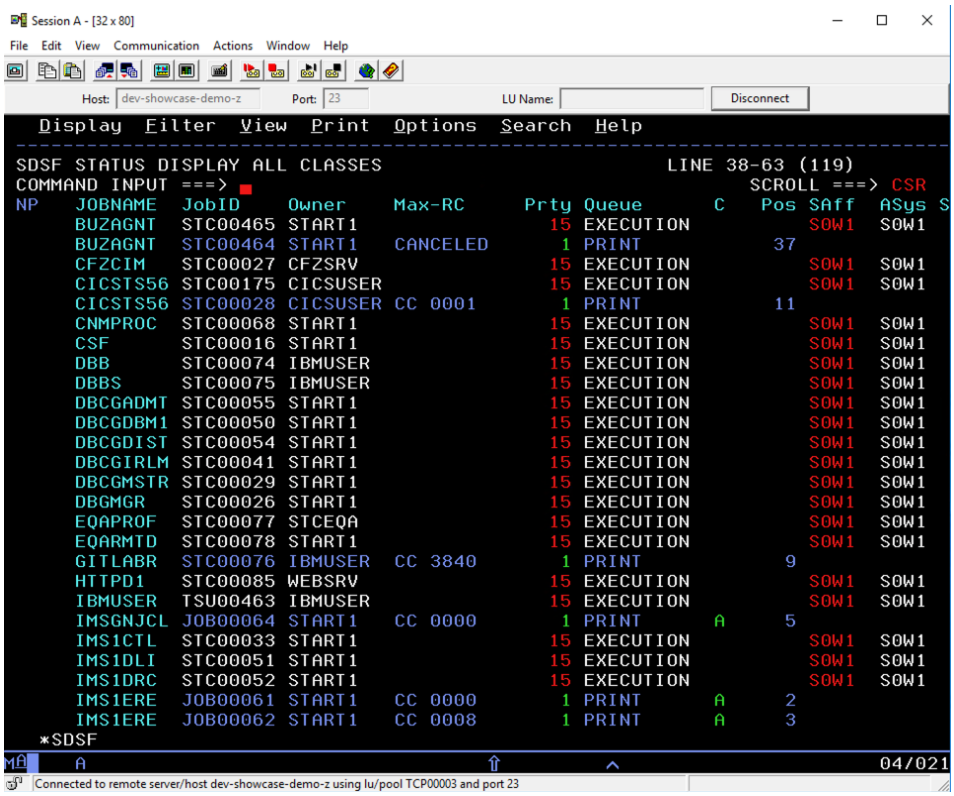
0.1.13 **BUZAGNT** agent might be started, but it is not connected to the UCD server. It is necessary to cancel this JOB and start it again. On the column **NP**, type letter **c** the front of **BUZAGNT** job and press **ENTER.**



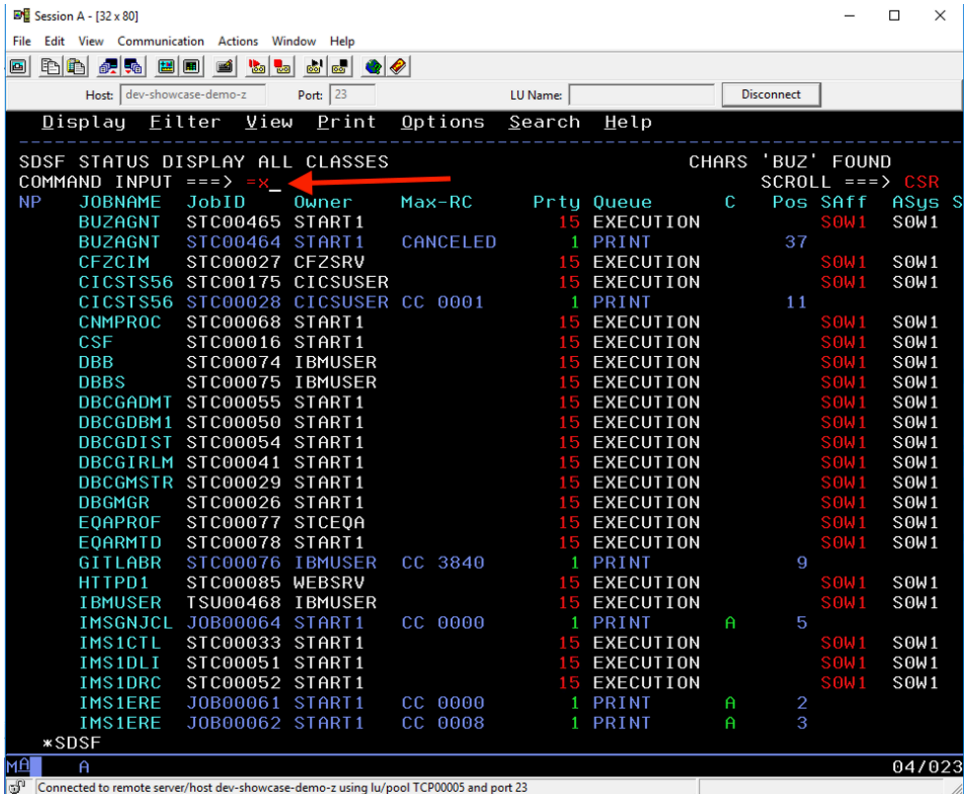
0.1.14 BUZAGNT is canceled and now start the BUZAGNT. On **COMMAND INPUT,** type **/s BUZAGNT**. Press **ENTER**.

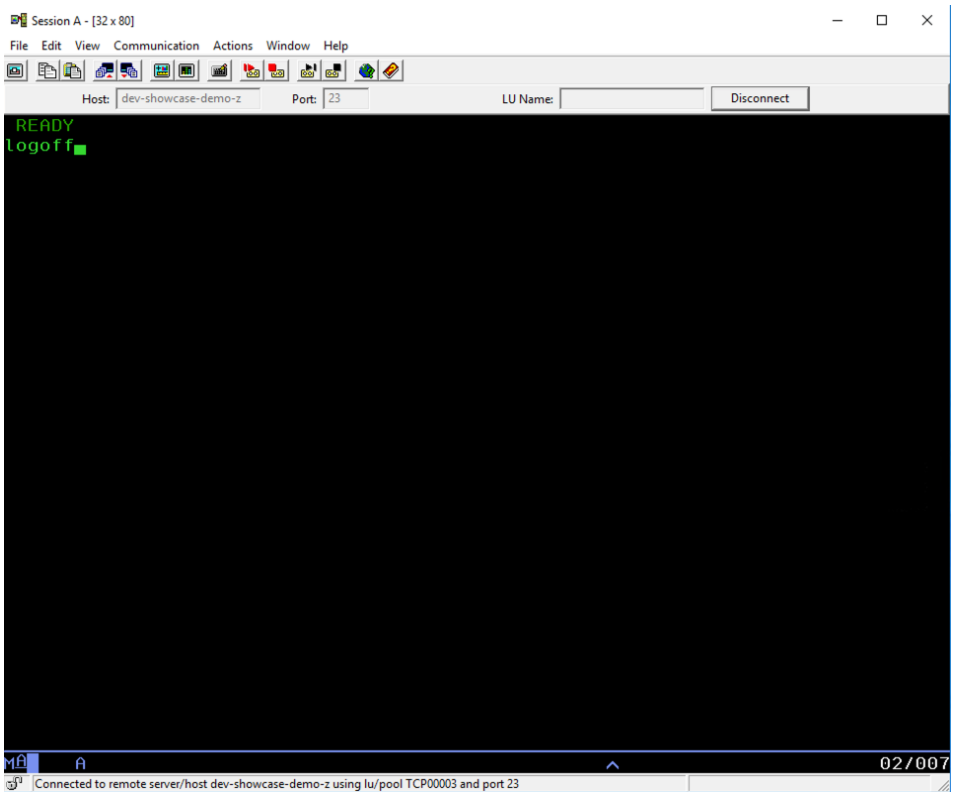


0.1.15 BUZAGNT agent is connect.

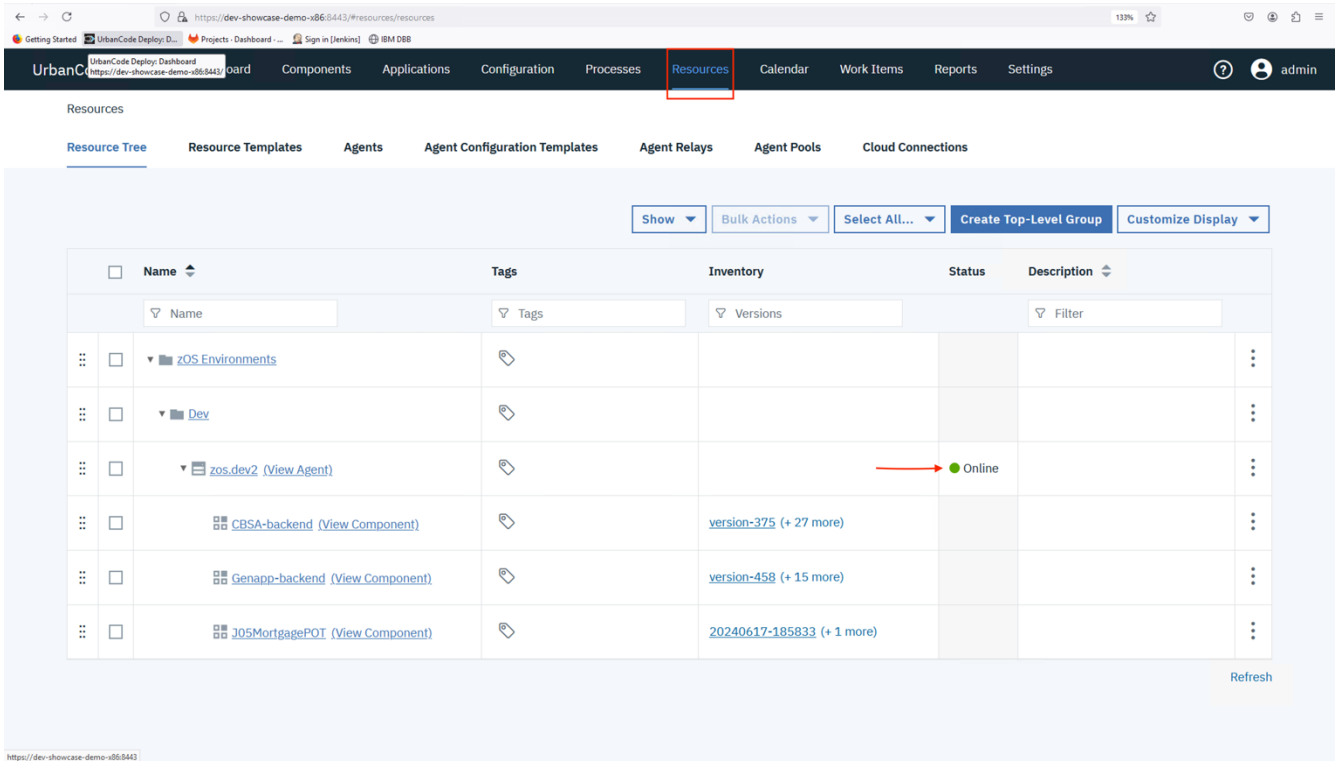


0.1.16 Logout the PCOMM session. On COMMAND INPUT, type **=x** press **ENTER.** and type **logoff** and **ENTER.**





0.1.17 Back to Firefox browser on UCD page, locate **Resource** and check **zos.dev2** is **Online**.



Now go to Task 1

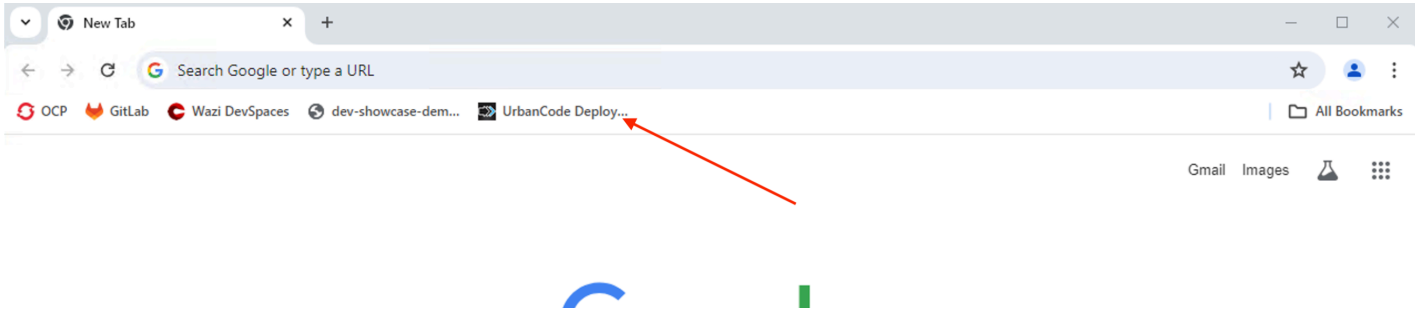
Task 1 – Logon to UCD server running on Linux and verify that the UrbanCode agents are running

You will now connect to the UrbanCode Deploy (UCD) server running on a LINUX (same Linux that ZDT is running) and that each student have access to it
Here you will verify the UCD agents that the UCD server has access to.

You will use the user id **empot02** and password **empot02**. Remember that here the Userid and password is **lower case.**

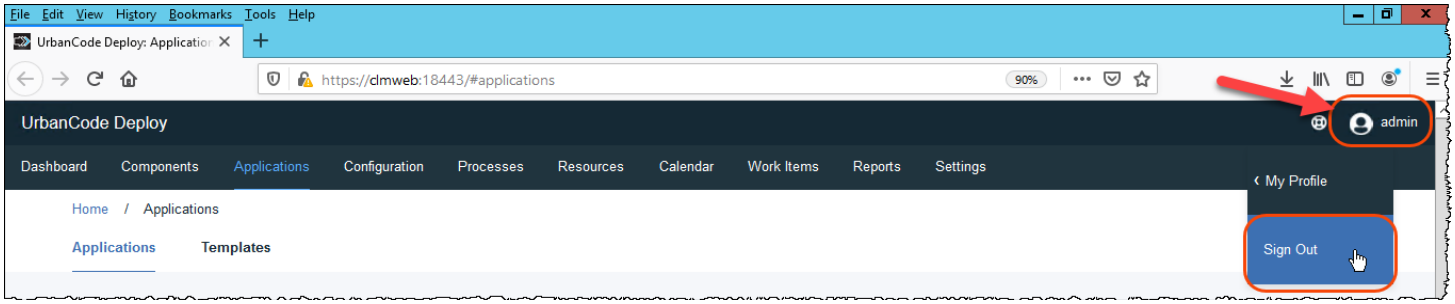
1.1 **Start the Firefox browser** to go to UCD server on Linux You can use the icon on the base of your screen Move the mouse to the base of your screen if you do not see this bar.

Look at Bookmark toolbar and select **UrbanCode Deploy**

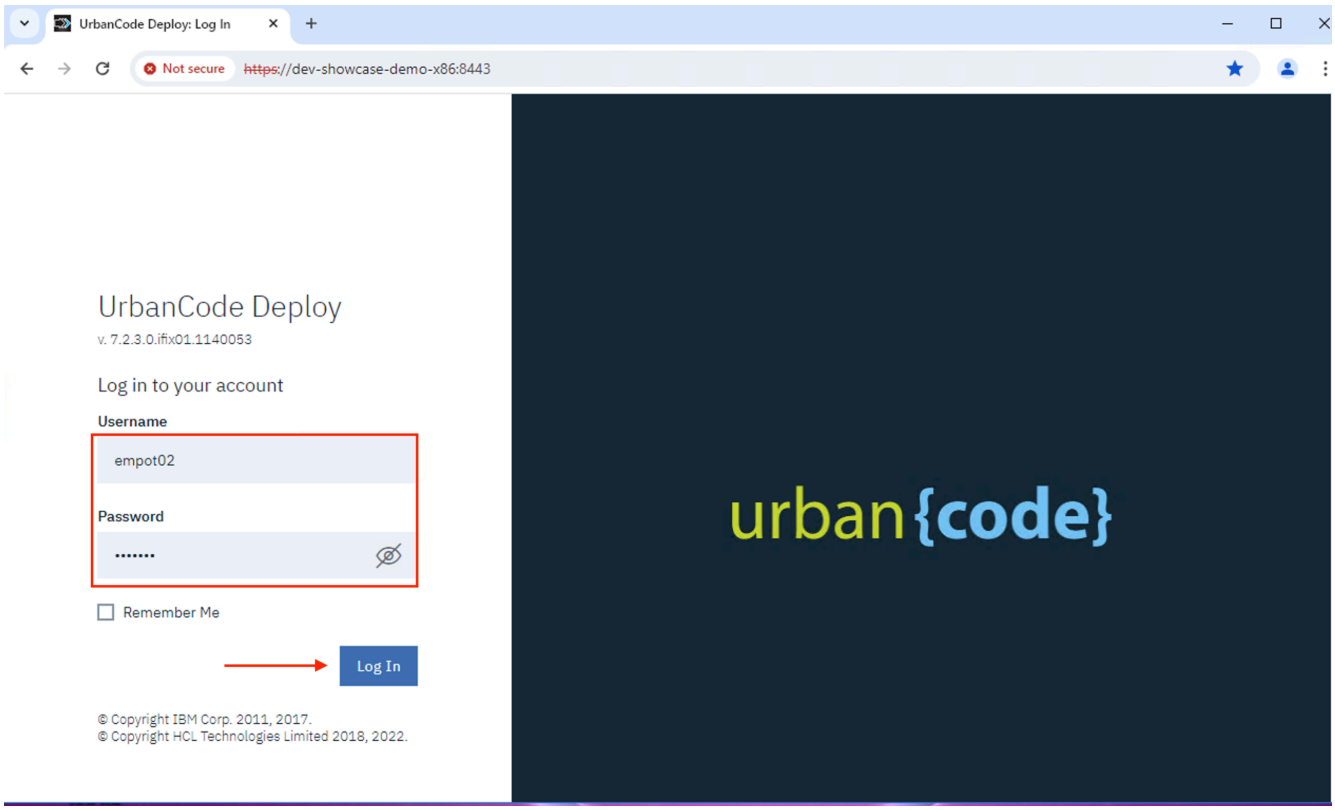


Or use: https://dev-showcase-demo-x86:8443/

1.2. If you are already logged on, you must first logoff and logon with your right user id.
Click on **admin** (or whatever user id is logged in) and select **Sign Out**



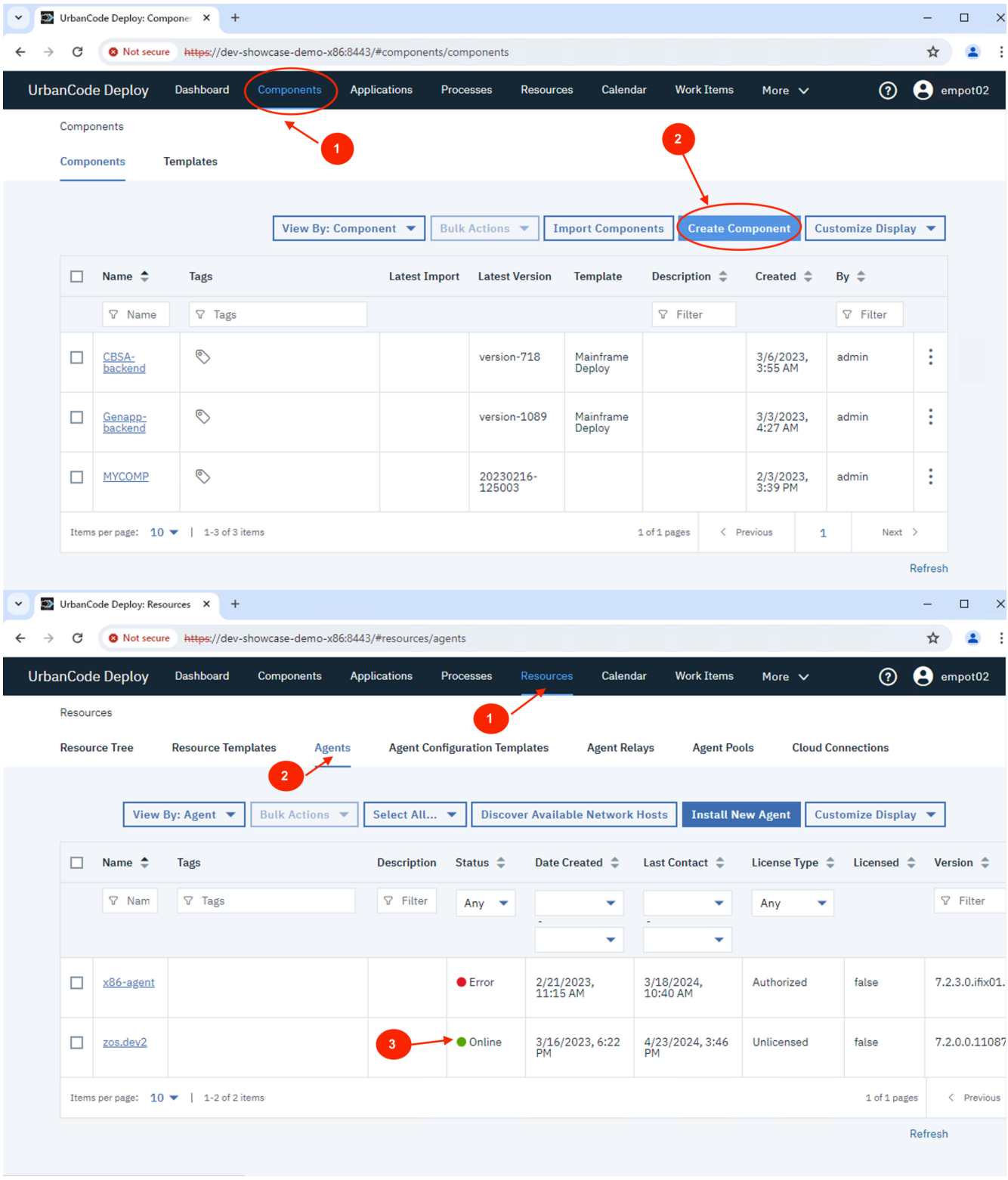
1.3. Login with **empot02** and password **empot02**
Remember that this is case sensitive here and lower case



1.4. Verify that the UCD agents are running by clicking **Resources > Agents**.

Check agent **zos.dev2** agent is running (**online**)

On this lab you will use only the z/OS V7.2 agent (**zos.dev2**)


UrbanCode Deploy: Compone
Not secure
https://dev-showcase-demo-x86:8443/#components/components
UrbanCode Deploy
Dashboard
Components
Applications
Processes
Resources
Calendar
Work Items
More
empot02
1
2
Components
Components
Templates
View By: Component
Bulk Actions
Import Components
Create Component
Customize Display
Name
Tags
Latest Import
Latest Version
Template
Description
Created
By
CBSA-backend
version-718
Mainframe Deploy
3/6/2023, 3:55 AM
admin
Genapp-backend
version-1089
Mainframe Deploy
3/3/2023, 4:27 AM
admin
MYCOMP
20230216-125003
2/3/2023, 3:39 PM
admin
Items per page: 10
1-3 of 3 items
1 of 1 pages
Previous
1
Next
Refresh
UrbanCode Deploy: Resources
Not secure
https://dev-showcase-demo-x86:8443/#resources/agents
UrbanCode Deploy
Dashboard
Components
Applications
Processes
Resources
Calendar
Work Items
More
empot02
1
2
Resources
Resource Tree
Resource Templates
Agents
Agent Configuration Templates
Agent Relays
Agent Pools
Cloud Connections
View By: Agent
Bulk Actions
Select All...
Discover Available Network Hosts
Install New Agent
Customize Display
Name
Tags
Description
Status
Date Created
Last Contact
License Type
Licensed
Version
x86-agent
Error
2/21/2023, 11:15 AM
3/18/2024, 10:40 AM
Authorized
false
7.2.3.0.ifix01.
zos.dev2
3
Online
3/16/2023, 6:22 PM
4/23/2024, 3:46 PM
Unlicensed
false
7.2.0.0.11087
Items per page: 10
1-2 of 2 items
1 of 1 pages
Previous
Refresh

Task 2 – Create a UCD component

Components are groups of deployable artifacts that make up an application. You will later create your UCD application named "J02Mortgage" that will include only one component that you will create here.

UCD: What is a Component?
A component is a logical representation of deployable artifact along with user-defined processes that operate on them. The physical artifacts are specified in a Component Version. A component has one or more component versions. Each component version has one or more files.

In UrbanCode Deploy, components represent deployable items along with user-defined processes that operate on them. Deployable items, or artifacts, can be files, PDS members, images, databases, etc. In our example, component will include **PDS** members **J02CMORT** and **J02MPMT** .

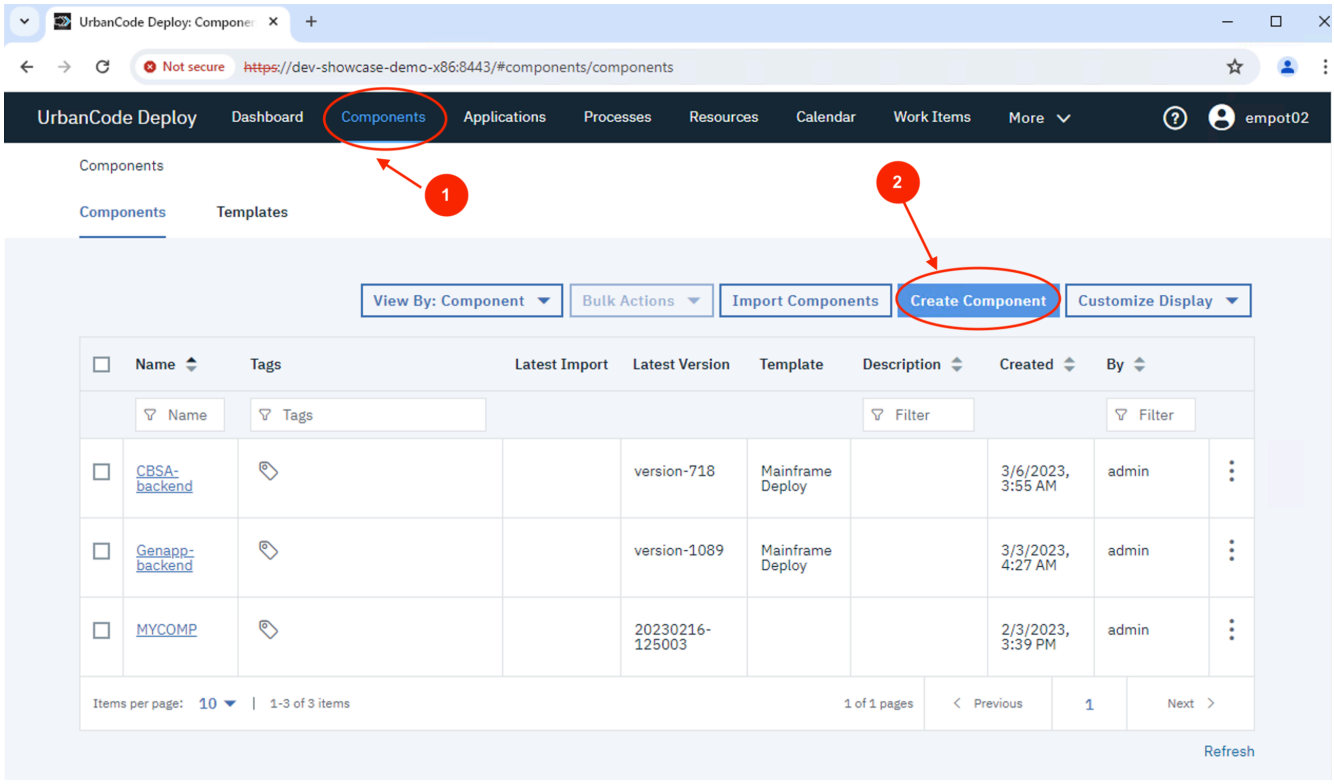
z/OS: What is a PDS?

A partitioned data set or PDS consists of a *directory* and *members*. The directory holds the address of each member, or "file", and thus makes it possible for programs or the operating system to access each member directly. Each member, however, consists of sequentially stored records.

Partitioned data sets are often called *libraries*. Programs, or other content, are stored as members of partitioned data sets. Generally, the operating system loads the members of a PDS into storage sequentially, but it can access members directly when selecting a program for execution.

A PDS also contains a directory. The directory contains an entry for each member in the PDS with a reference (or pointer) to the member. Member names are listed alphabetically in the directory, but members themselves can appear in any order in the library. The directory allows the system to retrieve a particular member in the data set.

2.1 ⏭ On the **Components** page, click **Create Component**:

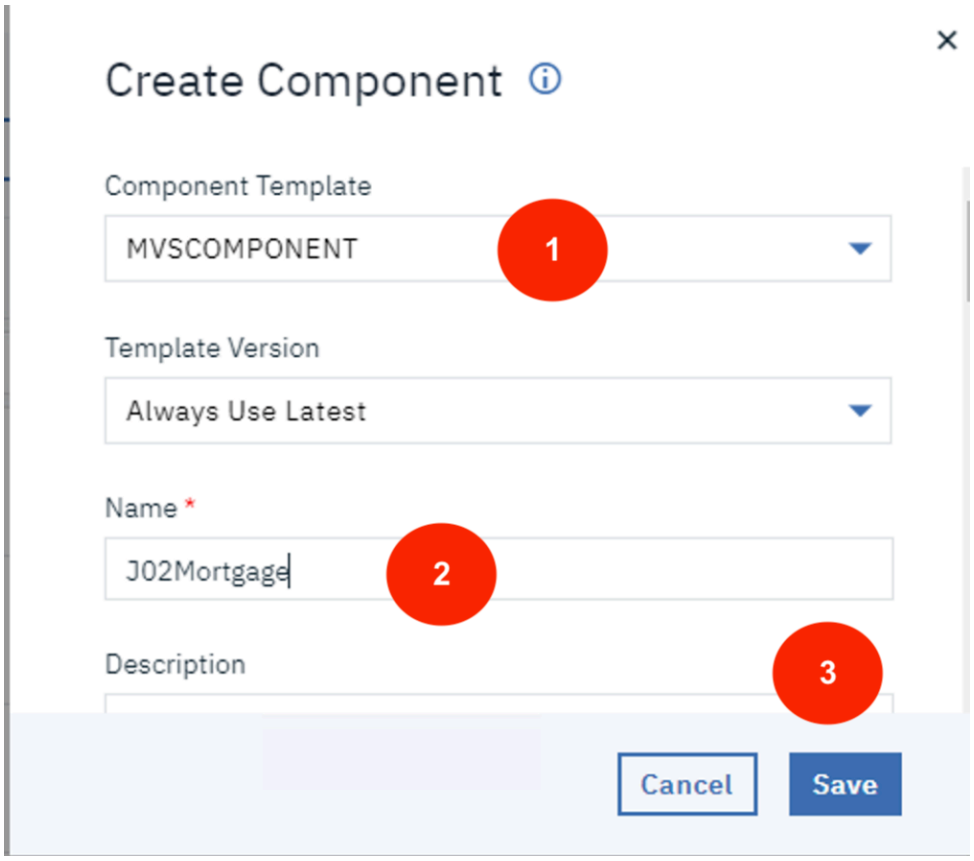


2.2 Provide values for the required fields on the **Create Component** dialog.

a. ⏭ Select **MVSCOMPONENT** in the **Template** field (Templates serve as models for various groups of resources. MVSCOMPONENT template, preinstalled with UCD, includes basic z/OS deploy processes and other z/OS related settings

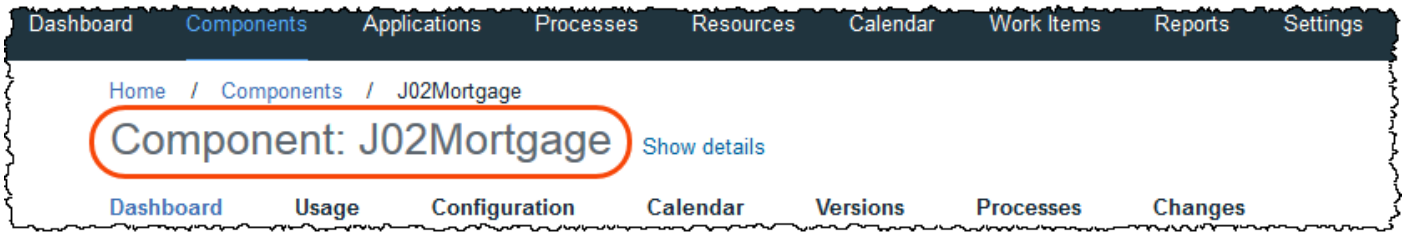
b. ⏭ Specify component Name as **J02Mortgage**
(Be careful since the name is case sensitive).

c ⏭ Scroll down and click **Save**.


Create Component
Component Template
MVSCOMPONENT
1
Template Version
Always Use Latest
Name *
J02Mortgage
2
Description
3
Cancel
Save

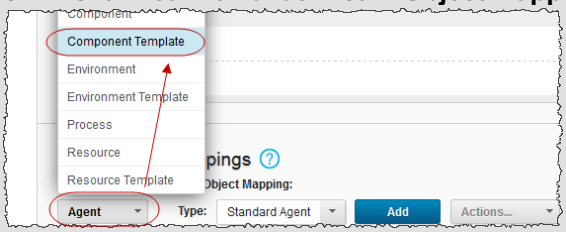
UCD: What is a Component Template?
A component template, stores component processes and properties so you can create components from them; template-based components inherit the template's properties and process.

2.3 The component **J02Mortgage** is created

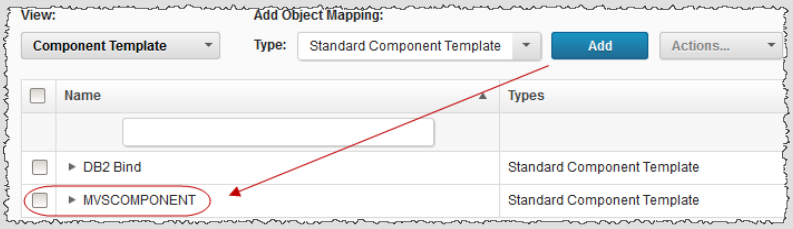


The MVSCOMPONENT is not in the drop down?.
Your UCD server must have this template defined there. If NOT call the instructor..
1. Logon ad **admin/admin**
2. Click **Settings** > **Teams** (*under Security)*
3. For EACH team perform as below..

3.1 – Click **Team 01** under **Team Object Mappings/View**: select **Component Template**



3.2 – Click **Add** and select **MVSCOMPONENT**



Task 3 – Create a ship list file and z/OS component version

The z/OS **ship list file** specifies which files from the build to include in the new component version to deploy. You must create a ship list file before you run z/OS deployment tools.

Ship list files are XML files that contain a list of files to be deployed on z/OS. For more information, refer to the UCD documentation:

http://www.ibm.com/support/knowledgecenter/SS4GSP_6.2.6/com.ibm.udeploy.doc/topics/zos_shiplistfiles.html

To create a ship list file for the component to be deployed you could use many ways, including ISPF.
To make this easy you will use IDz.

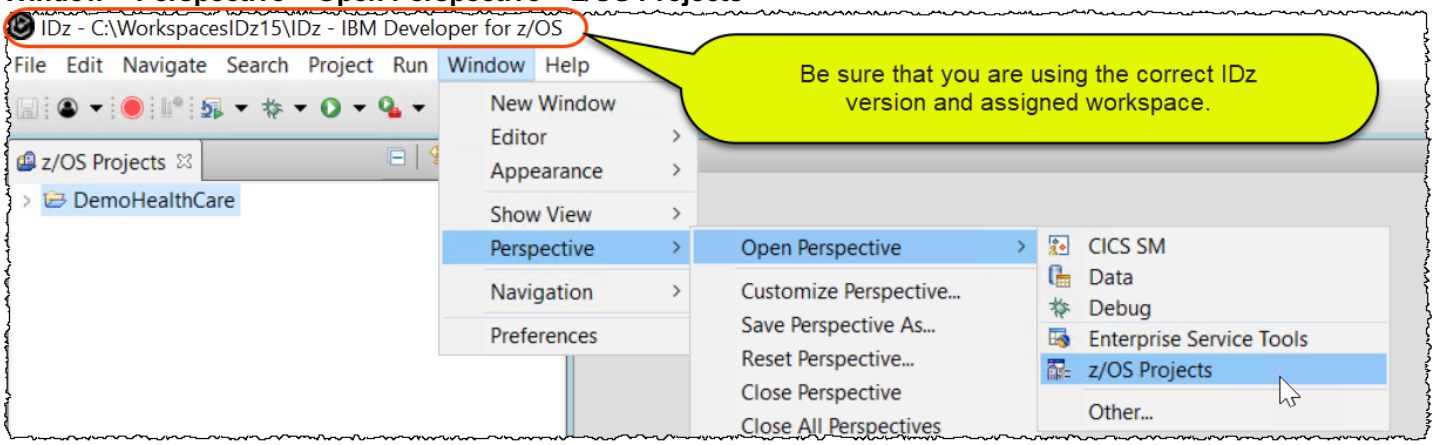
3.1 Start *IBM Developer for z Systems version 16* if it is not already started

⏭ Using the desktop double click on **IDz V16** icon.

⏭ Verify that the message indicates that it is Version **16**

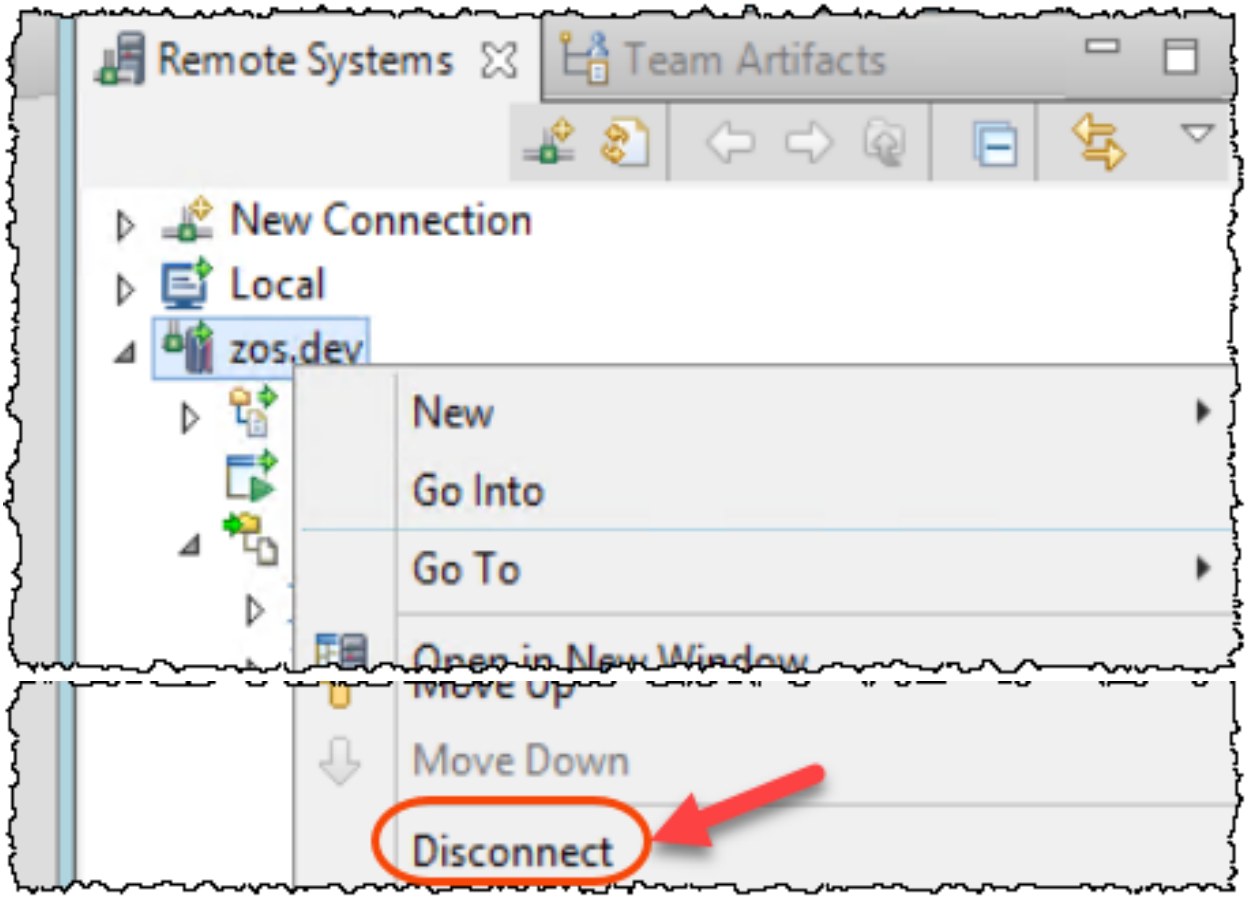
IMPORTANT -> This icon will start an eclipse workspace that has already some definitions required for this lab. PLEASE DO NOT start IDz using other way than this icon.

3.2 ⏭ Open the **z/OS Projects** perspective by selecting
Window > Perspective > Open Perspective > z/OS Projects

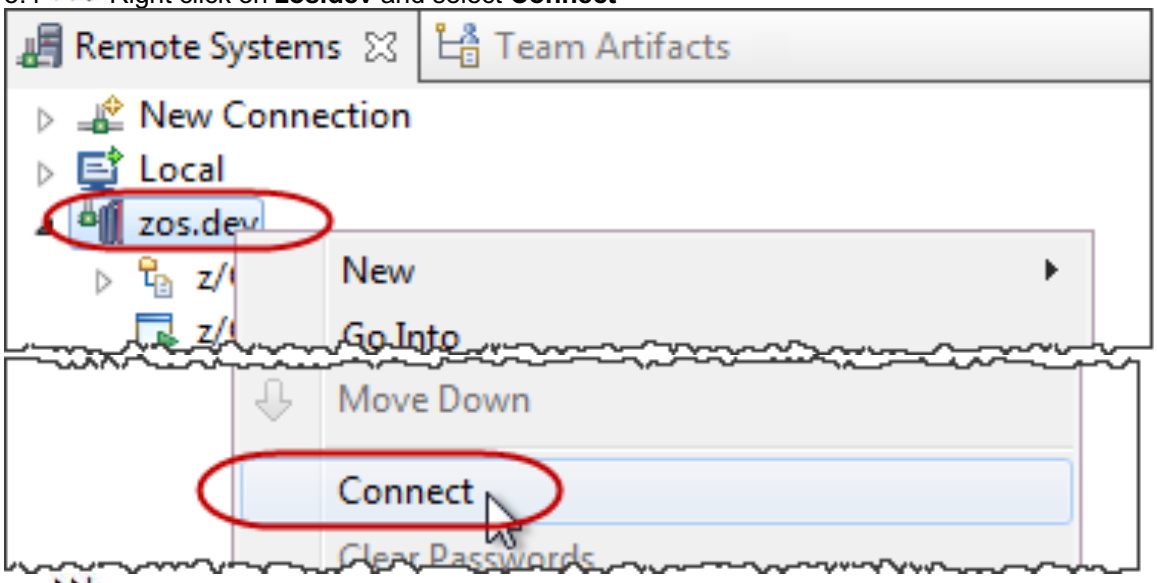


3.3 Before you used *empot01,empot05* or i*bmuser* and you need to disconnect. Your new userid now will be **empot02**.

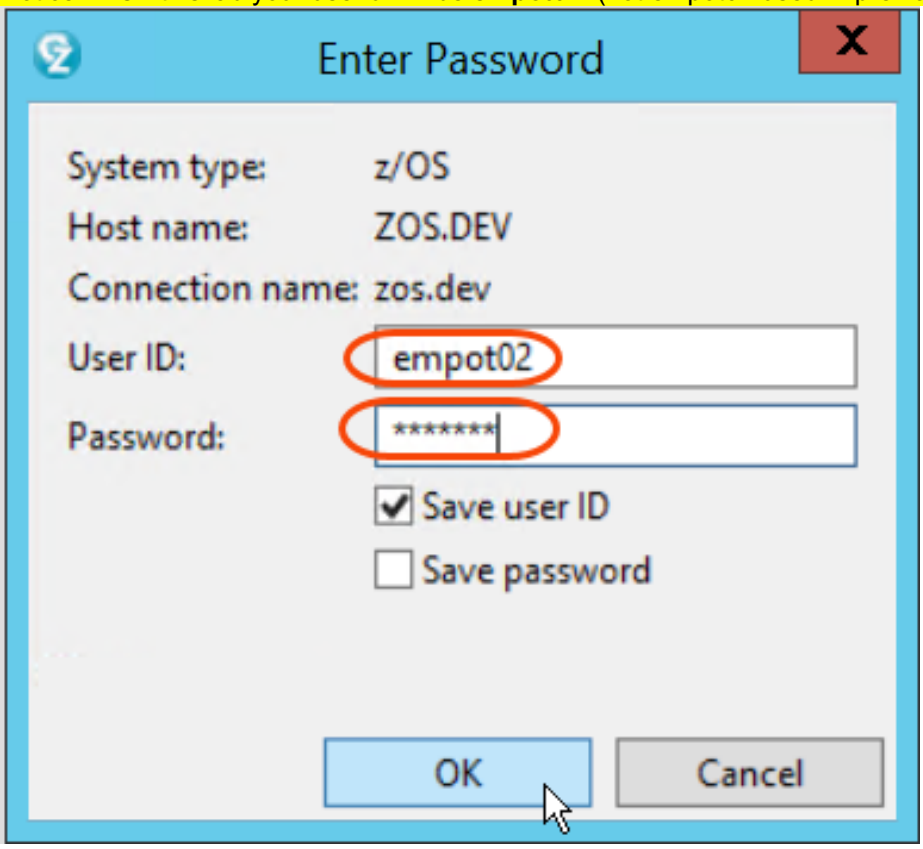
⏭ Right click on **zos.dev** and select **Disconnect**



3.4 ⏭ Right click on **zos.dev** and select **Connect**



3.5 ⏭ Type **empot02** as userid and **empot02** as password.
Click **OK** to connect to z/OS.
Notice → On this lab your userid will be **empot02** (not empot01 used in previous labs)

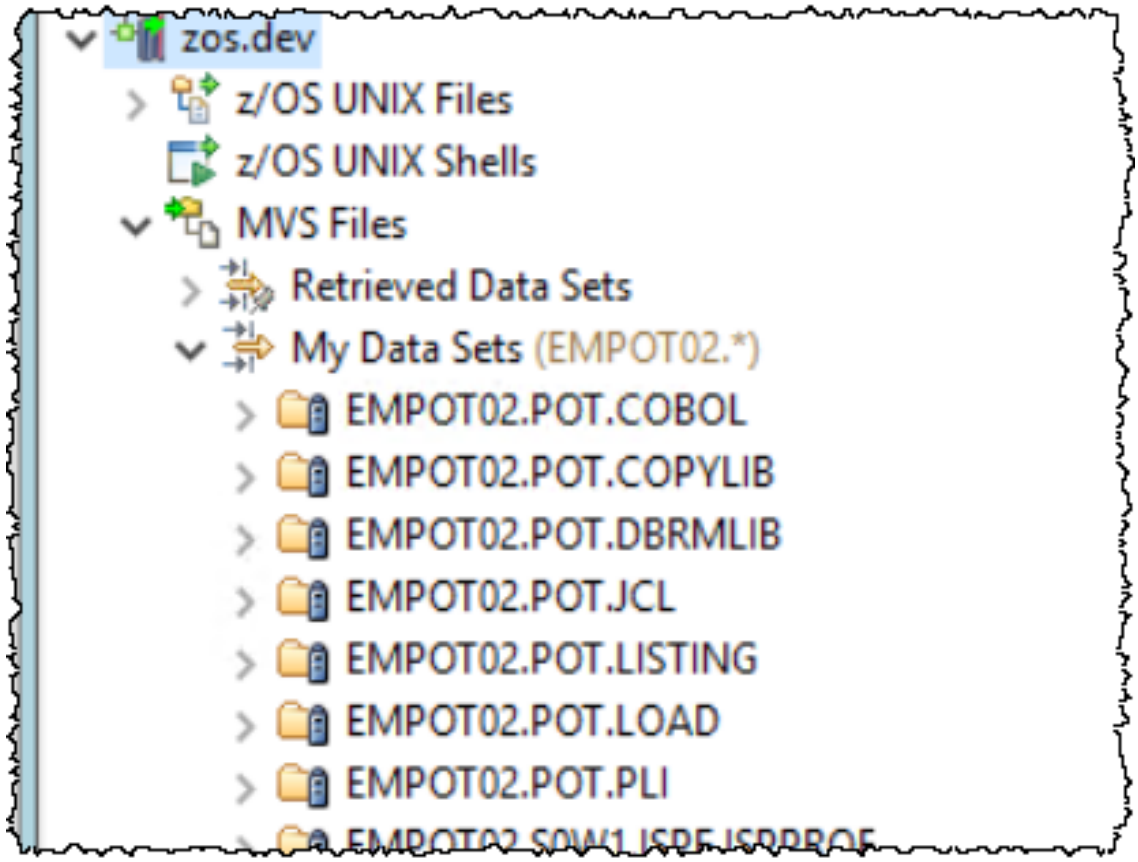


3.6 Notice that the folder icons changed once you are connected to z/OS. A small green arrow is added.,,

From… 

To: 

3.7 Expand MVS Files and My Data Sets (to see all your MVS data sets
Note that you do not have the same PDSs shown below (EMPOT02.*) this is just an example..
Depending on which ID you are using you may have no data sets, the IDs are reused and we have no control what is there.

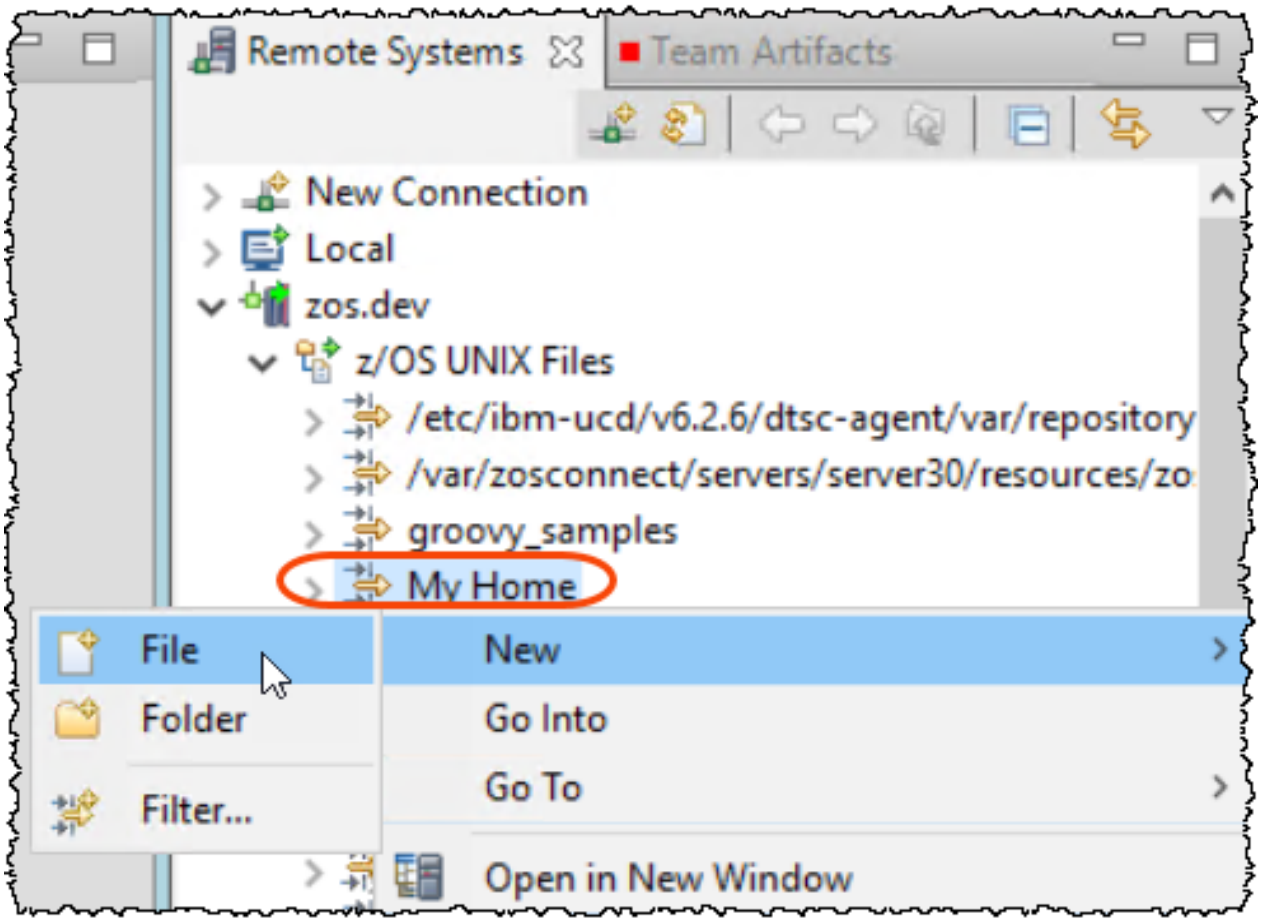


You are connected to a z/OS remote system. Now you will create a Ship List and submit JCL to create the executables to be deployed.

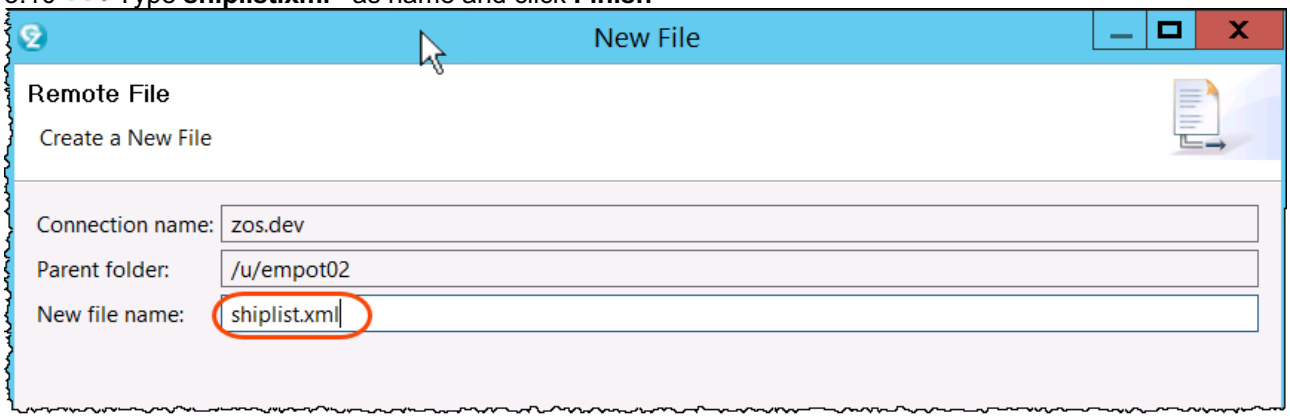
3.8 You now can create the UCD ship list. An XML file that has the list of the load modules that will be deployed.This xml could be created from an SCM build tool like RTC, Endevor etc..
For this lab we will create it manually. .

What is the Ship list file?
The ship list file specifies which files from the build to include in the new component version to deploy. You must create a ship list file before you run the IBM® z/OS® deployment tools.
Ship list files are XML files that contain a list of files. The container type that is used to identify partitioned data set (PDS) resources is PDS and the resource type is *PDSmember*. You can use an asterisk (*) as a wildcard for the resource name, if you want all members in a partitioned data set (PDS) to be included in a package. Typically, you write a script that works with your build engine to create a ship list file from the build output.

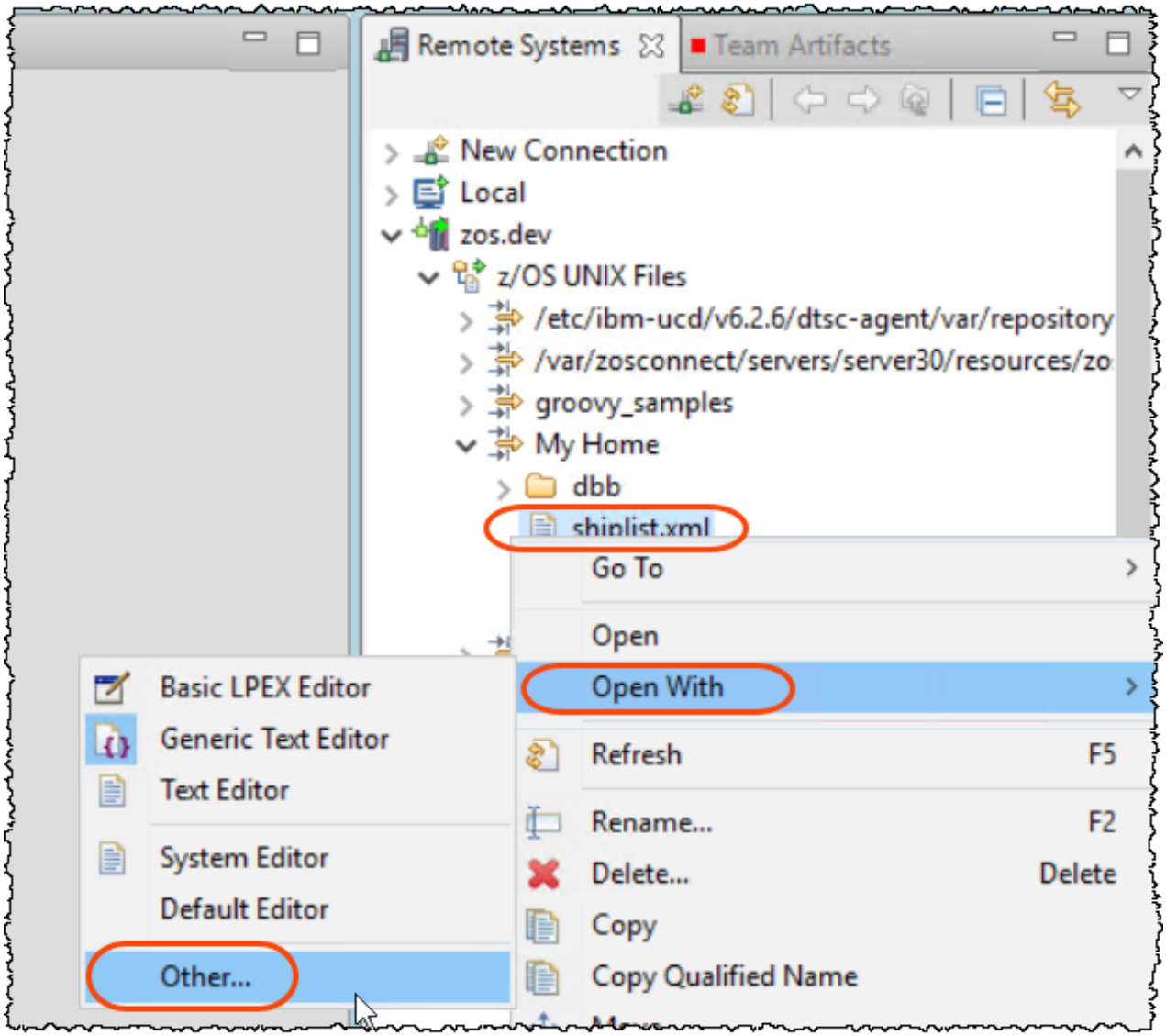
3.9 ⏭ Using the *Remote System*s view (on top right), expand **z/OS UNIX Files**, right click on **My Home → New → File**



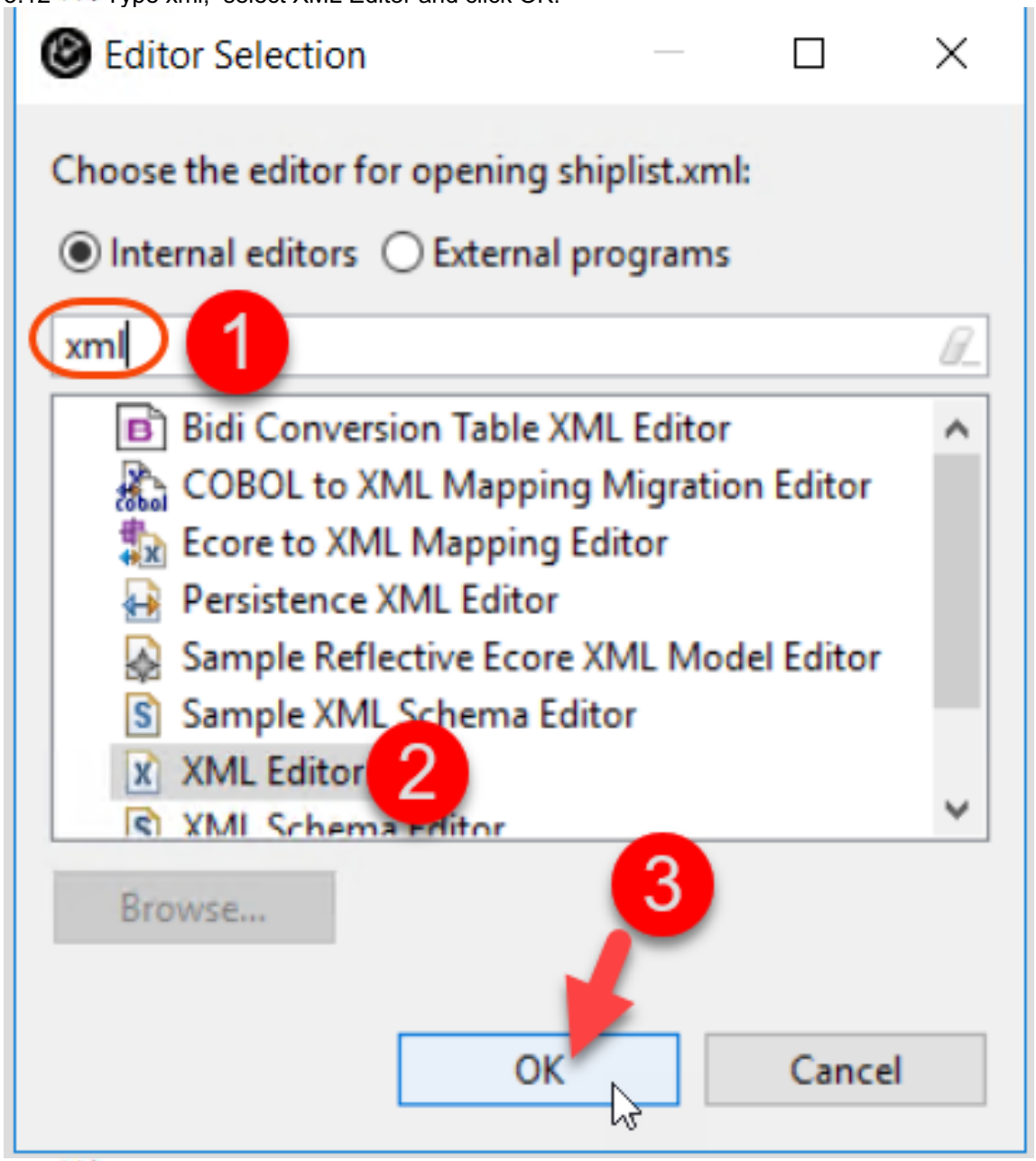
3.10 Type **shiplist.xml** as name and click **Finish**



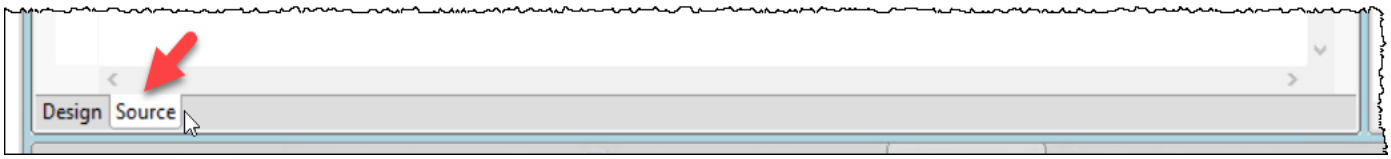
3.11 Right click on **shiplist.xml** and select it to edit, or right click and select **Open With Other…**



3.12 Type xml, select XML Editor and click OK.



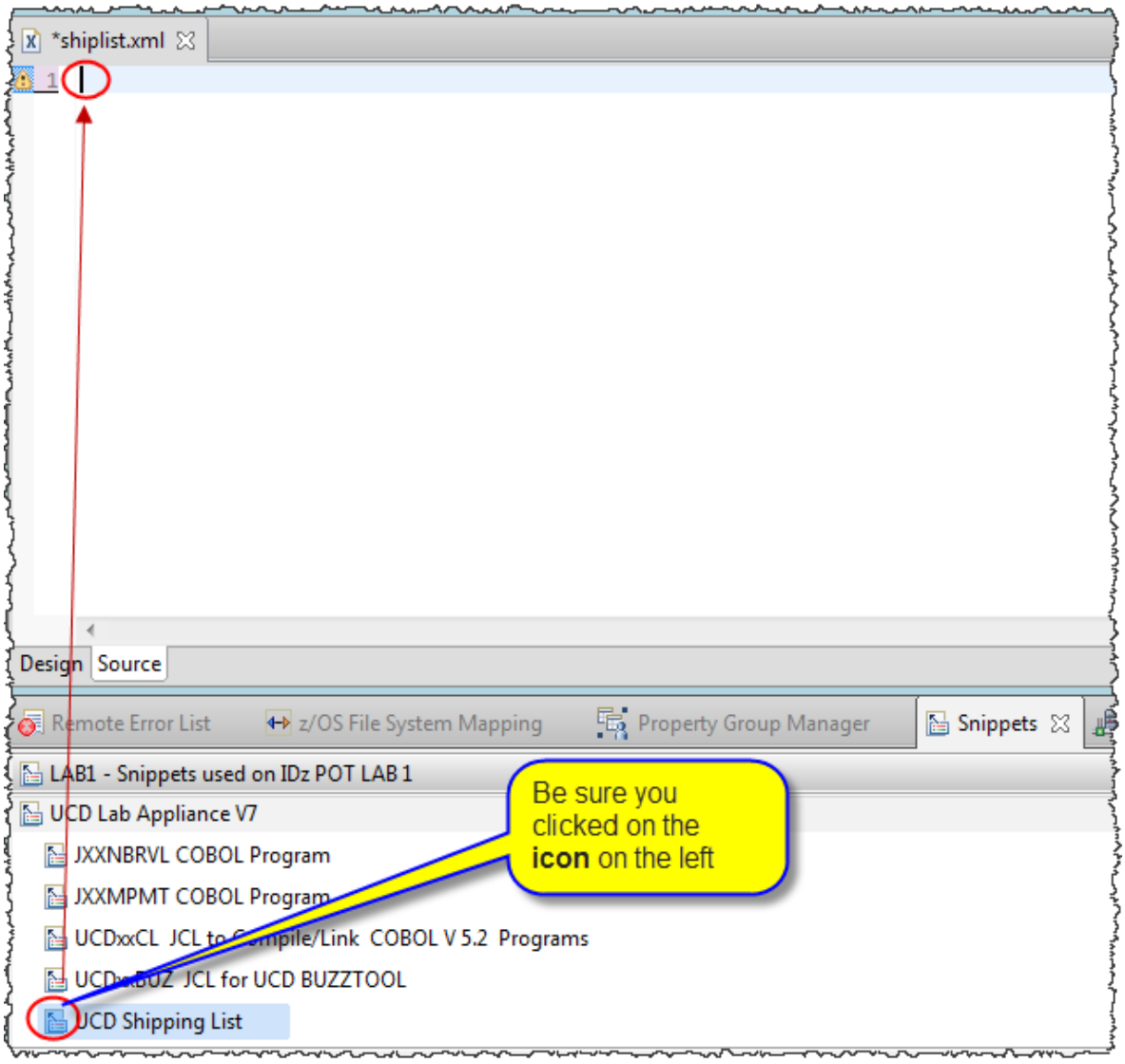
3.13 Click on **Source** tab



3.14 Click on **Snippets** tab on the bottom.

Scroll down and use the **UCD Shiping List** snippets to drag and drop to the **FIRST position** of the file

IMPORTANT: You must click on the icon to be able to drag/drop



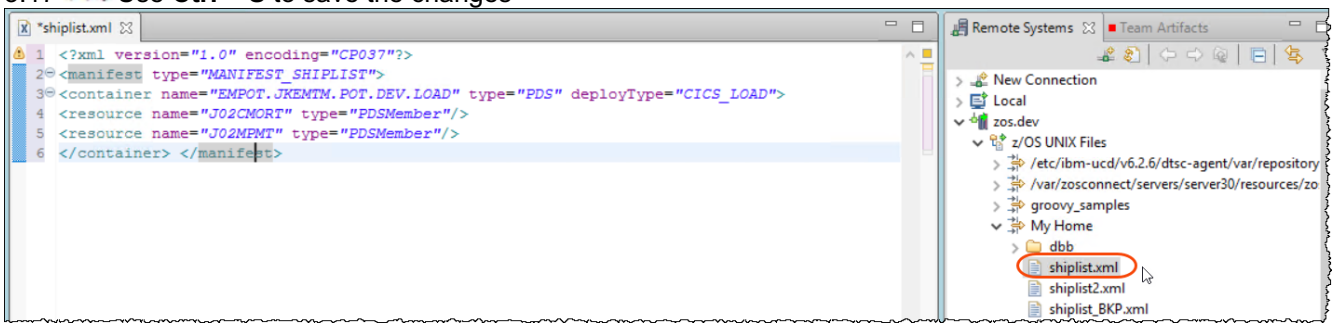
3.15 When the dialog opens type ***02***. ..

3.16 Click **Insert** to insert the lines. Notice that the variables are replaced

shiplist.xml | UCD02BUZ.jcl

```
//UCD02BUZ JOB  ,MSGLEVEL=(1,1),NOTIFY=&SYSUID,REGION=0M
//* ---------------------------------------------------------
//*   1. This job executes the BUZTOOL to create a VERSION
//*   2. Edit the Shiping List to see what will be packed
//*   on this example is  at /u/empot02/shiplist.xml
//*   3. Modify the version name  and submit to z/OS execution
//*   4. When completed verify the USS version folder is created at
//* /etc/ibm-ucd/v6.2.6/dtsc-agent/var/repository/J02Mortgage
//*   5. Go to URBAN CODE and run the DEPLOY at DEV level
//* ---------------------------------------------------------
//BUZTOOL  EXEC PGM=BPXBATCH,REGION=0M
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//STDPARM  DD *
SH /var/ucd/agent72/bin/buztool.sh createzosversion
       "-c" "J02Mortgage"
       "-s" "/u/empot02/shiplist.xml"
       "-v" "20240423"
/*
/* yyyymmdd will be the name of your version
//
```

3.17 Use **Ctrl + S** to save the changes

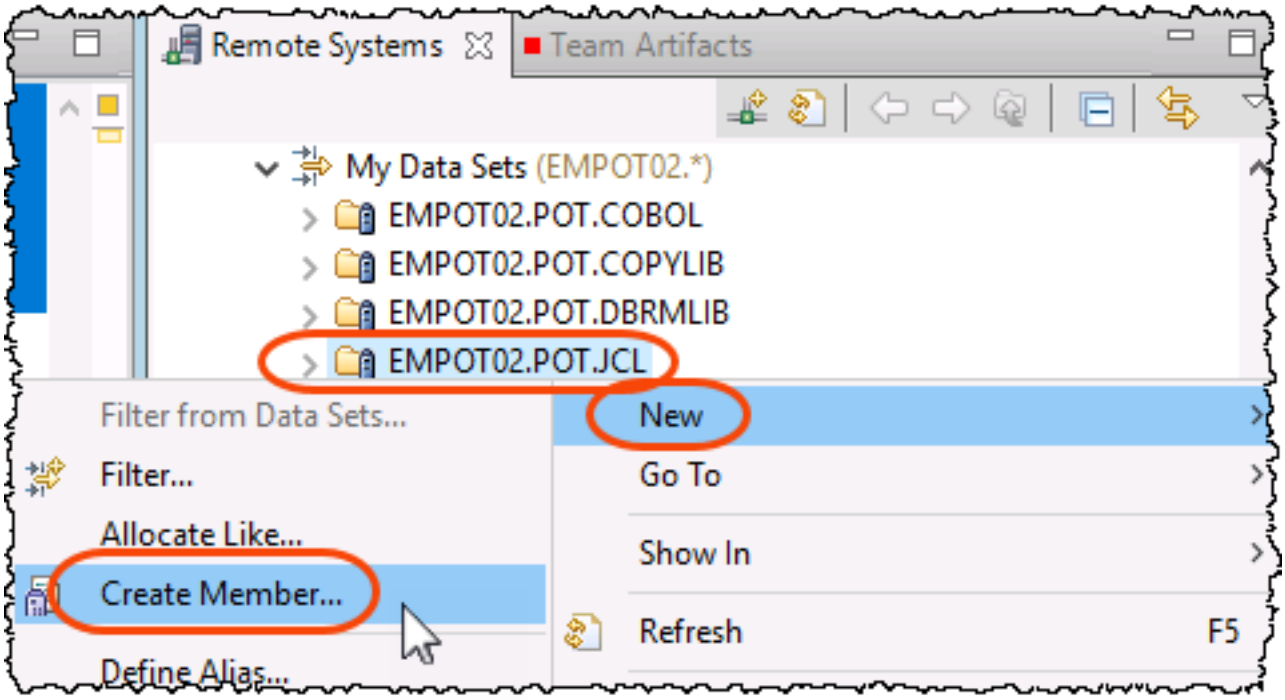


Task 4 - Create the UCD component version from JCL

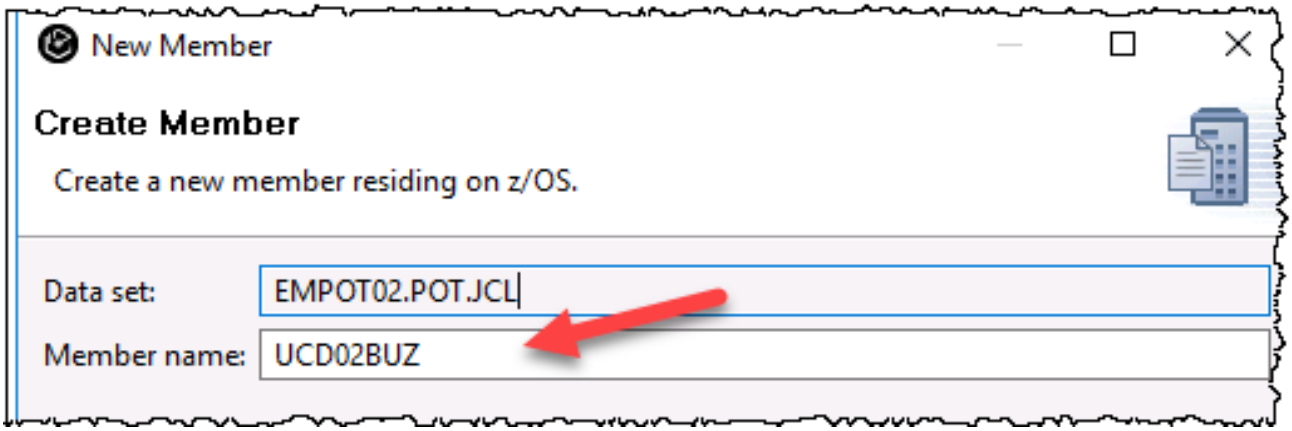
On this task, you will now use the z/OS UCD tool named "*BUZTOOL*" to create a version of the modules that need to be deployed. The UCD Toolkit installed on z/OS used the UCD Client to communicate to the UCD

server.
In this lab the UCD code station on z/OS is kept in the USS Files or on UCD server (installation decision).

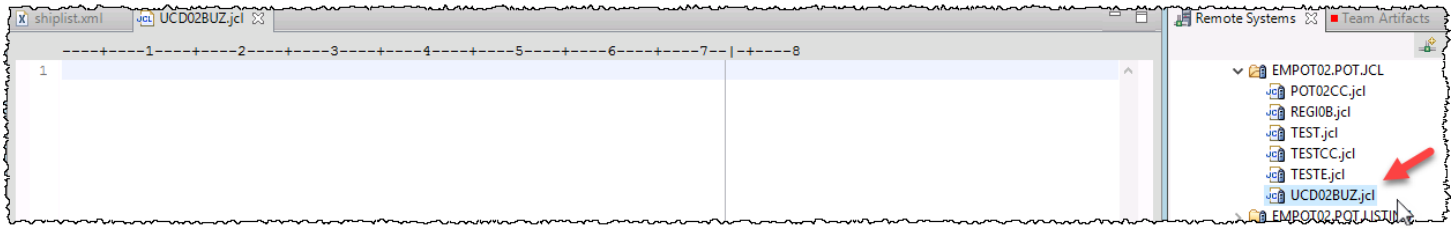
4.1 Using *Remote System* view, expand **My Data Sets, r**ight click on **EMPOT02.POT.JCL** and create a member .
If you don't have this PDS call the instructor.



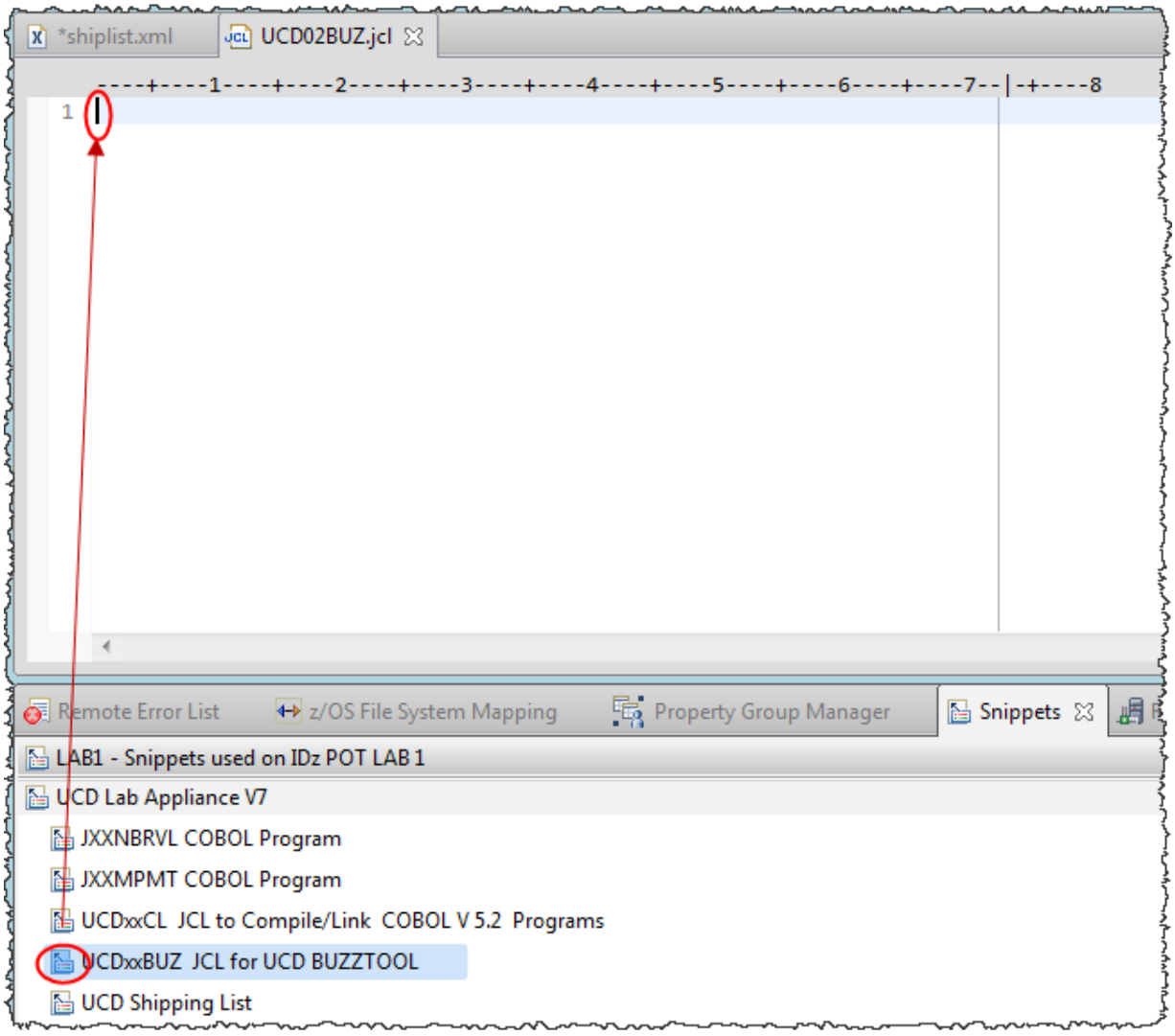
4.2 Name it **UCD02BUZ** and click **Finish**
NOTE: If you have already this member there, right click and delete it.



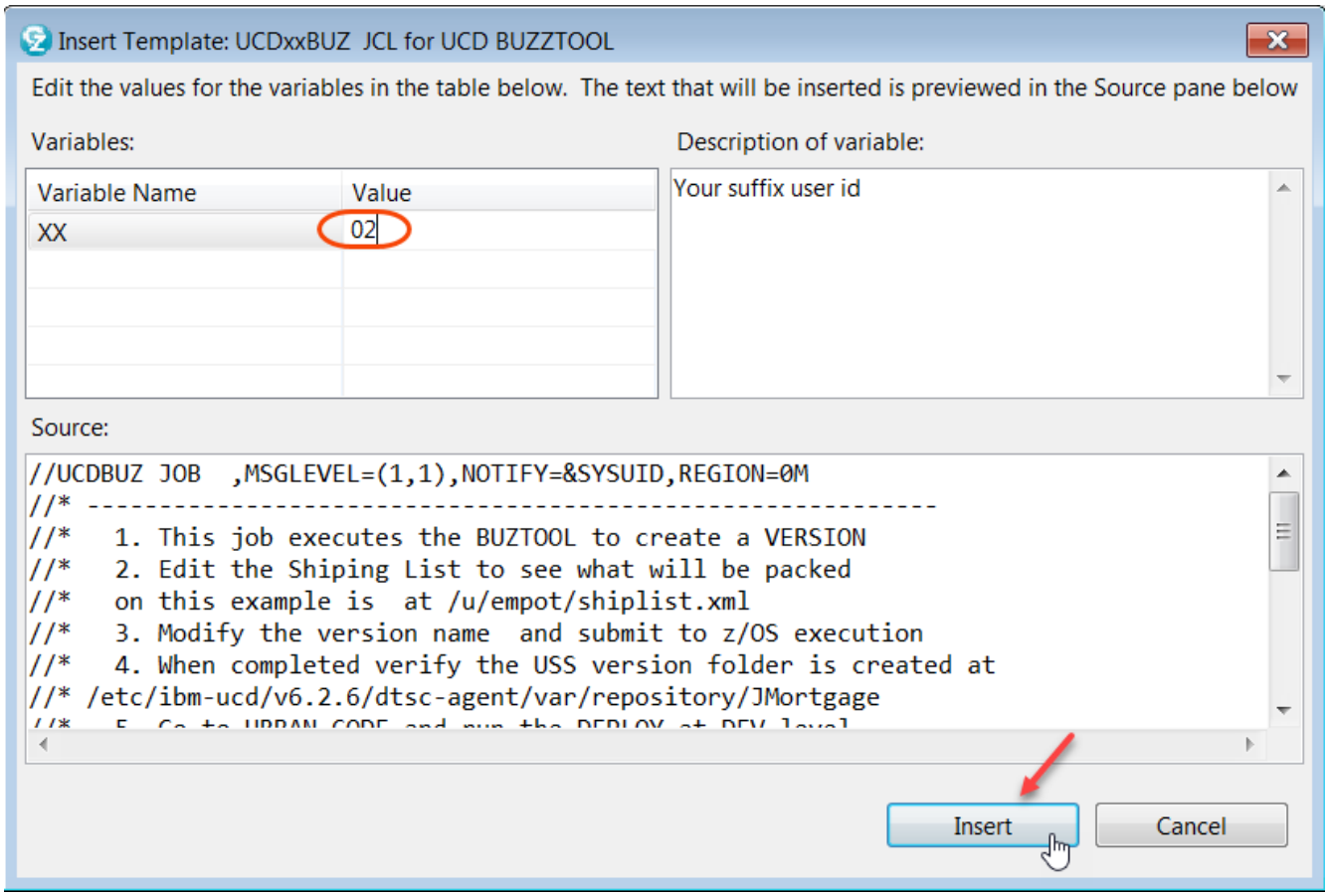
4.3 Double click on **UCD02BUZ.jcl** to edit.


shiplist.xml
UCD02BUZ.jcl
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
1
Remote Systems
Team Artifacts
EMPOT02.POT.JCL
POT02CC.jcl
REGI0B.jcl
TEST.jcl
TESTCC.jcl
TESTE.jcl
UCD02BUZ.jcl
EMPOT02.POT.LISTJL

4.4 Use the **UCDxxBUZ** snippets to drag and drop to the **FIRST POSITION** of the file
IMPORTANT: You must click on the icon to be able to drag/drop



4.5 When the dialog opens type **02**. ..
Click **Insert** to insert the lines. Note that the variables are replaced



This job uses the **buztool createzosversion** command to create the component version.

4.6 **Scroll down** and understand the parameters passed to the *BUZTOOL*

shiplist.xml | UCD02BUZ.jcl

```
//UCD02BUZ JOB  ,MSGLEVEL=(1,1),NOTIFY=&SYSUID,REGION=0M
//* ------------------------------------------------------------
//*   1. This job executes the BUZTOOL to create a VERSION
//*   2. Edit the Shiping List to see what will be packed
//*   on this example is  at /u/empot02/shiplist.xml
//*   3. Modify the version name  and submit to z/OS execution
//*   4. When completed verify the USS version folder is created at
//* /etc/ibm-ucd/v6.2.6/dtsc-agent/var/repository/J02Mortgage
//*   5. Go to URBAN CODE and run the DEPLOY at DEV level
//* ------------------------------------------------------------
//BUZTOOL  EXEC PGM=BPXBATCH,REGION=0M
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//STDPARM  DD *
SH /var/ucd/agent72/bin/buztool.sh createzosversion
       "-c" "J02Mortgage"
       "-s" "/u/empot02/shiplist.xml"
       "-v" "20240423"
/*
/* yyyymmdd will be the name of your version
//
```

*Possible arguments for the buztool **createzosversion** command*

Parameter	Required	Description
-c	true	The name of the component in IBM UrbanCode Deploy. The component name can contain only letters, numbers, and spaces.
-v	false	The name of the version to create. If a version is not specified, a version name is generated from the current time stamp. The version name can contain only letters, numbers, and spaces.

-s	true	The location of the ship list file.
-verb	false	To display a trace log, set this parameter to `true`.

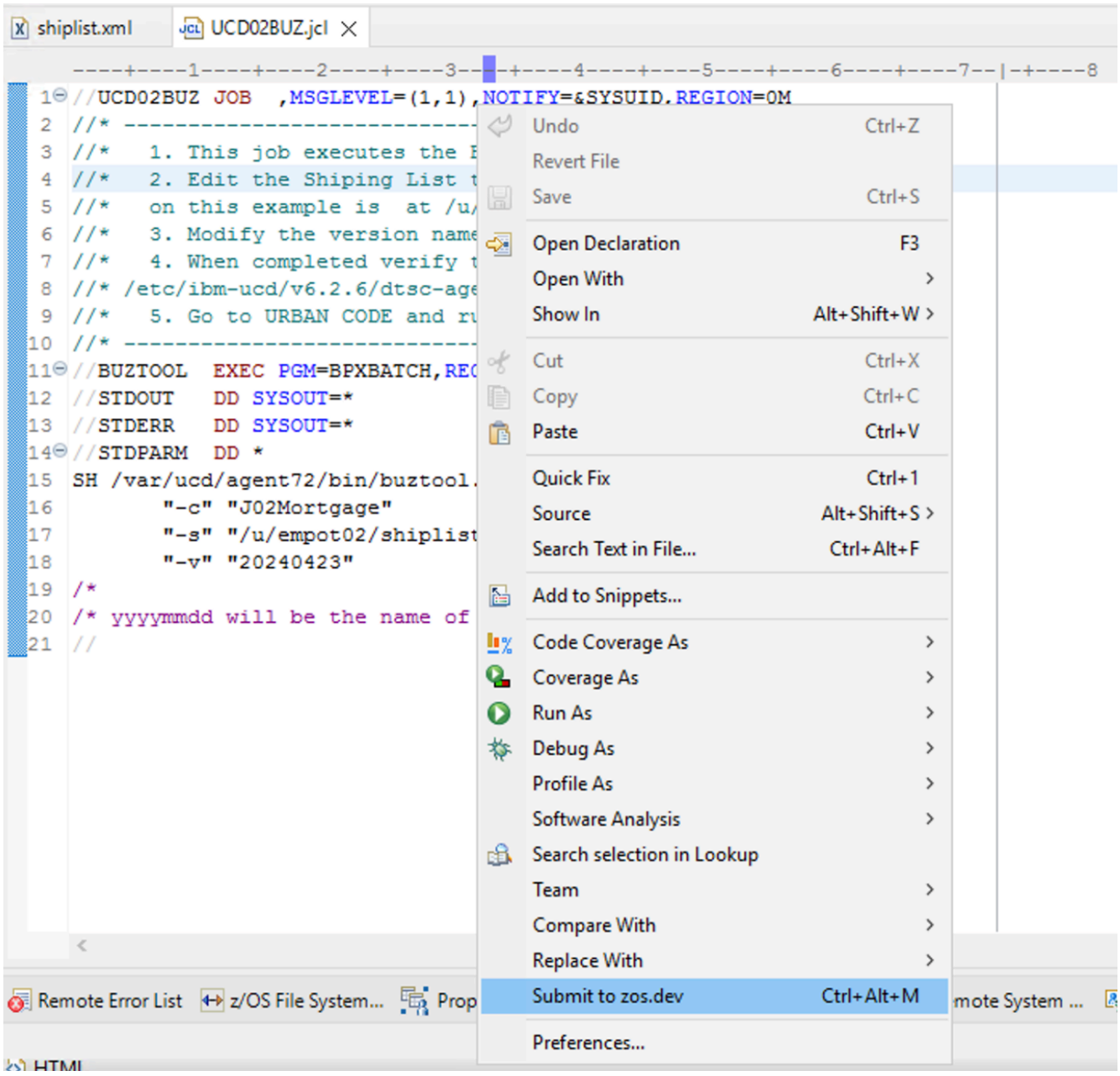
4.7 ⏭ Name a version for this component:
Modify from "yyyymmdd" to today's date in the format of **year, month** and **day**
Example: I used **20240423**. This version name must be unique for this component to avoid errors of being already created on the code station.

shiplist.xml | UCD02BUZ.jcl

```
//UCD02BUZ JOB  ,MSGLEVEL=(1,1),NOTIFY=&SYSUID,REGION=0M
//* ---------------------------------------------------------
//*   1. This job executes the BUZTOOL to create a VERSION
//*   2. Edit the Shiping List to see what will be packed
//*   on this example is  at /u/empot02/shiplist.xml
//*   3. Modify the version name  and submit to z/OS execution
//*   4. When completed verify the USS version folder is created at
//* /etc/ibm-ucd/v6.2.6/dtsc-agent/var/repository/J02Mortgage
//*   5. Go to URBAN CODE and run the DEPLOY at DEV level
//* ---------------------------------------------------------
//BUZTOOL  EXEC PGM=BPXBATCH,REGION=0M
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//STDPARM  DD *
SH /var/ucd/agent72/bin/buztool.sh createzosversion
       "-c" "J02Mortgage"
       "-s" "/u/empot02/shiplist.xml"
       "-v" "20240423"
/*
/* yyyymmdd will be the name of your version
//
```

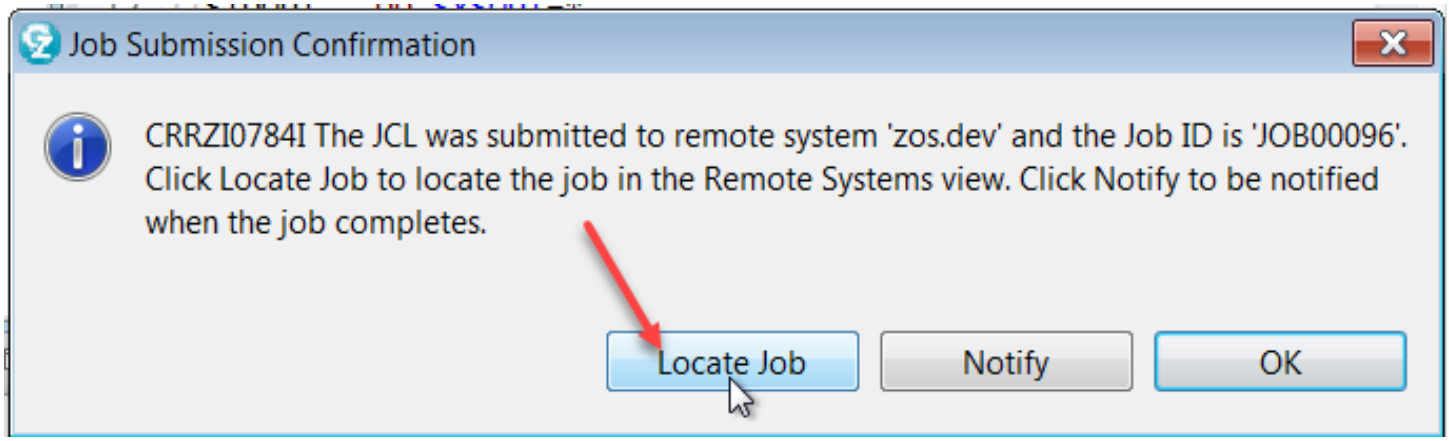
4.8 Use **Ctrl + S** to save

Right click on the editor and select **Submit to zos.dev**



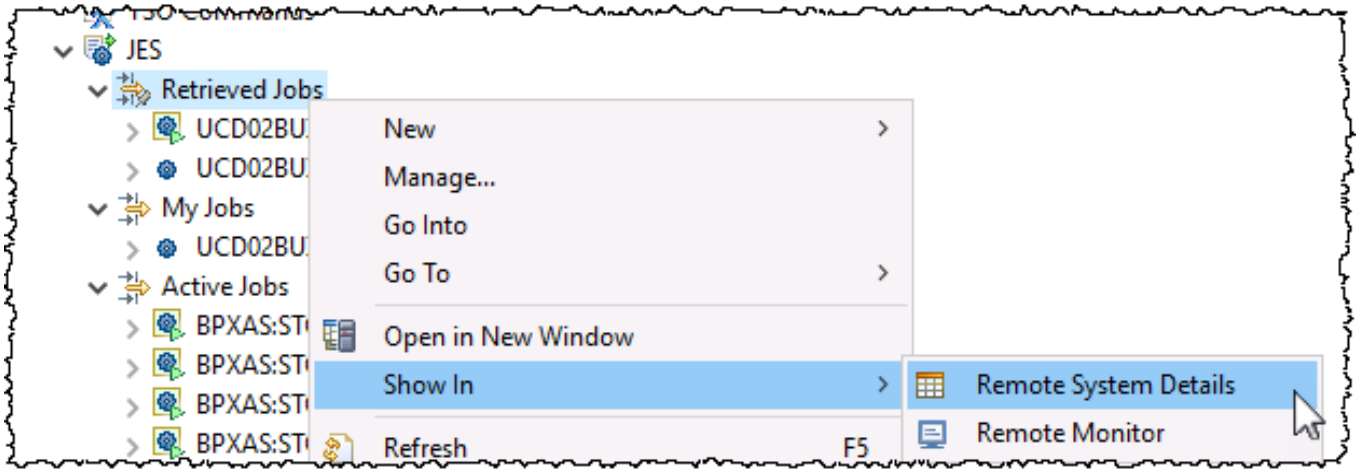
You do not find the option Submit?…
Be sure you are using the correct editor. Either LPEX or JCL editors will provide this capability when right clicking. Be sure you pasted the content on line 1 and column 1. If you still don't see, you can right click on the editor content and select ***Open with→ Other -> JCL Editor*** *and try again..*

4.9 Click **Locate Job** to get the job results

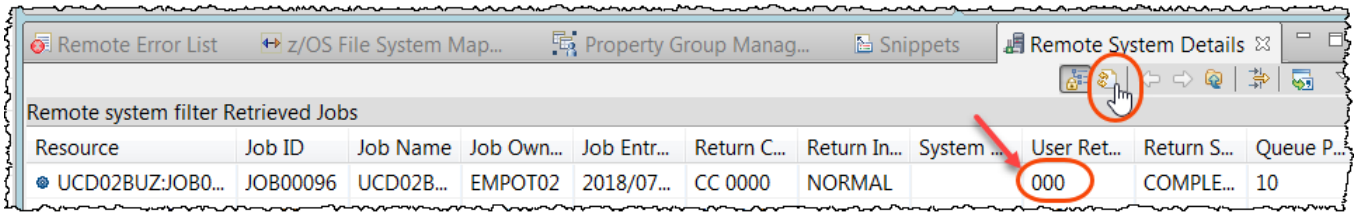


This job may take as long as 1 or 2 minutes.. Remember that your z/OS instance on cloud has limited power resources and is very slow.

4.10 On the right under *Remote System* view **right click** on **Retrieve Jobs** and select **Show in > Remote System Details**.
This is a better way to review the job execution

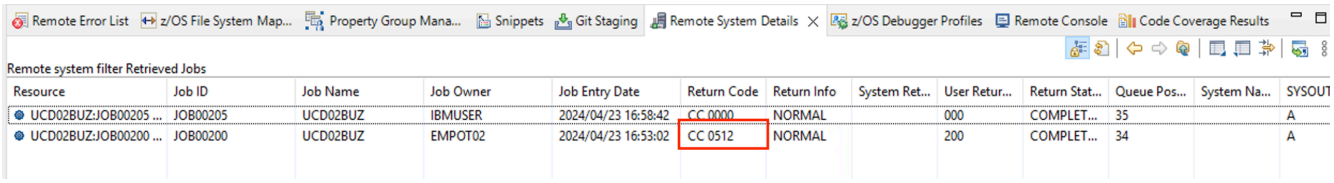


4.11 Use the **refresh icon** to see the job being executed. Once execution is completed you must have a *User Return Code*

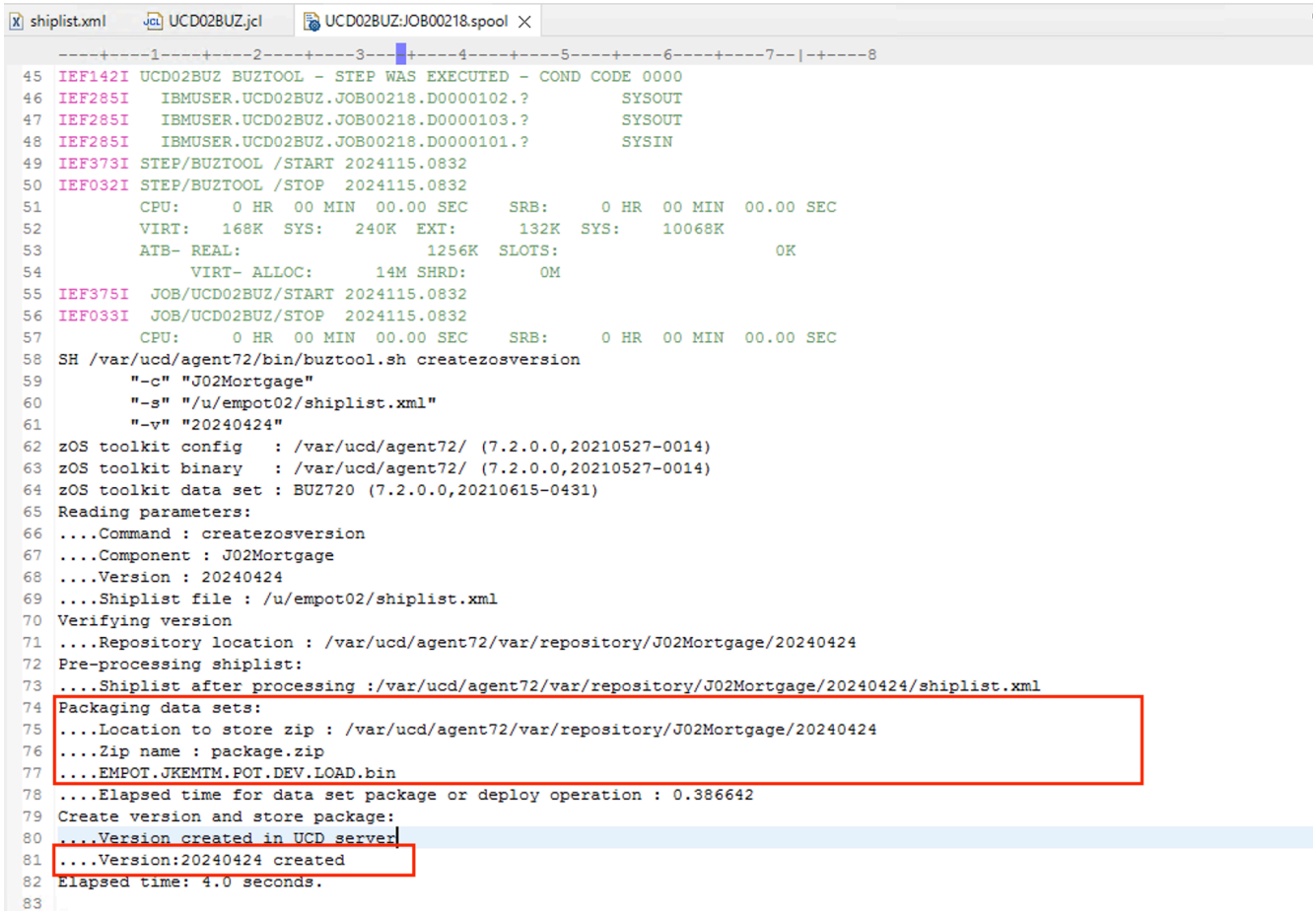


The return code **must be 0** and the version must be created on UCD Server

Note: you might receive return code 512. To continue your Lab, **disconnect and connect** change as **IBMUSER** and password is **sys1**.
On Remote Systems. Open **UCD02BUZ.jcl** on EMPOT02.POT.JCL folder, redo items **4.8,4.9** ,and **4.10**.



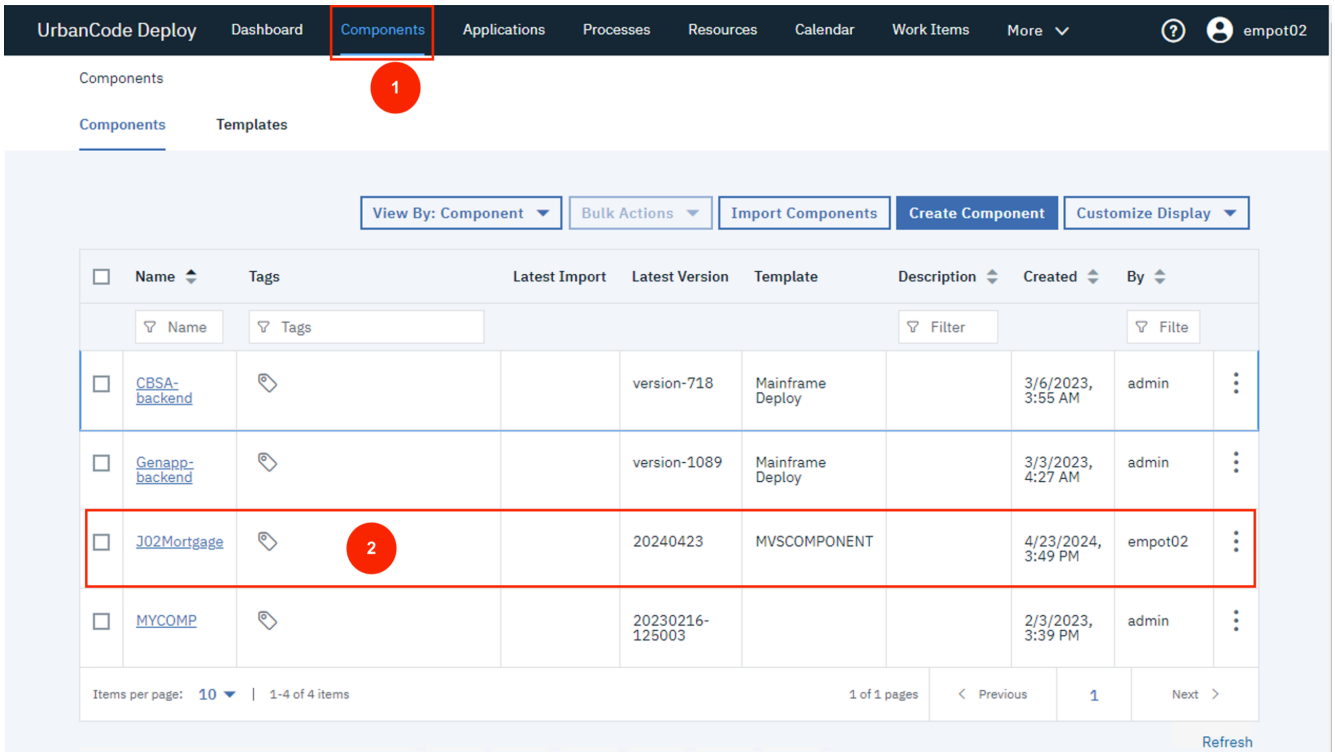
4.12 **Double click** and scroll down to see the results.



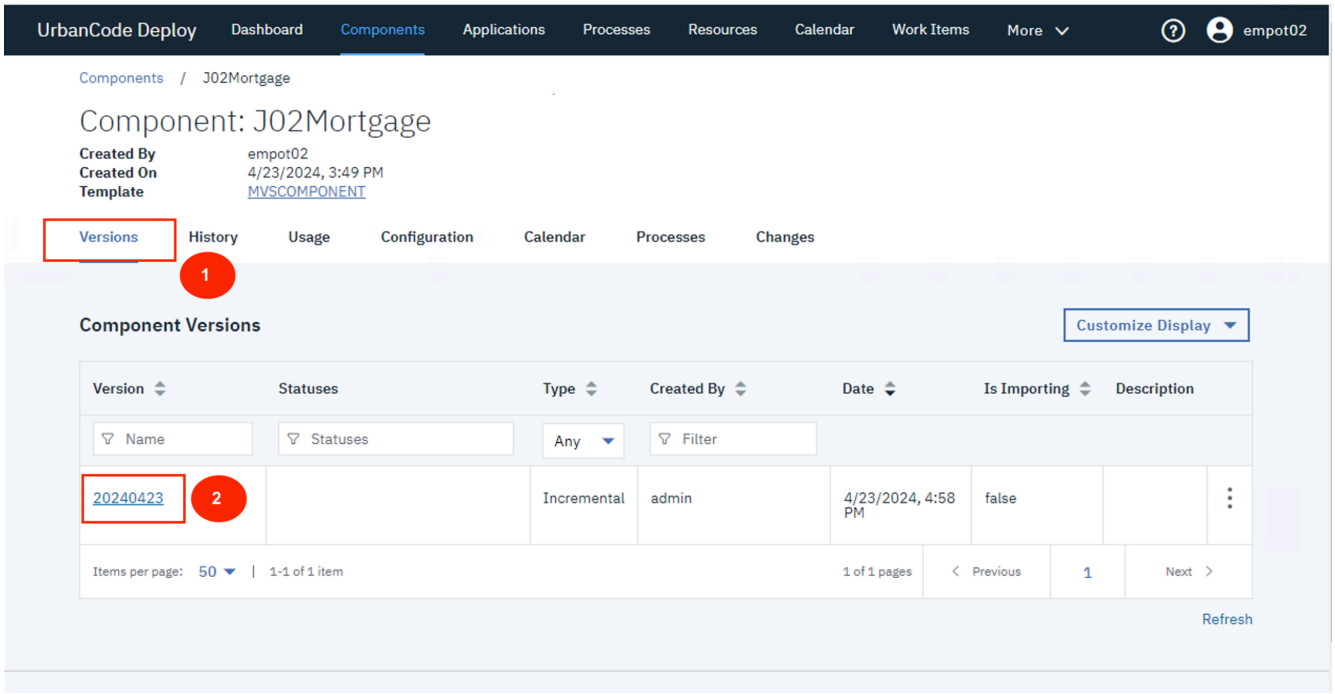
4.13 **Close** or minimize IDz, you will not need any more on this lab.

Notice that during version creating process, the *BUZTOOL* communicates with your UCD server to obtain component information and to store version artifacts. If component version was created successfully, you can verify version information using UCD server interface.

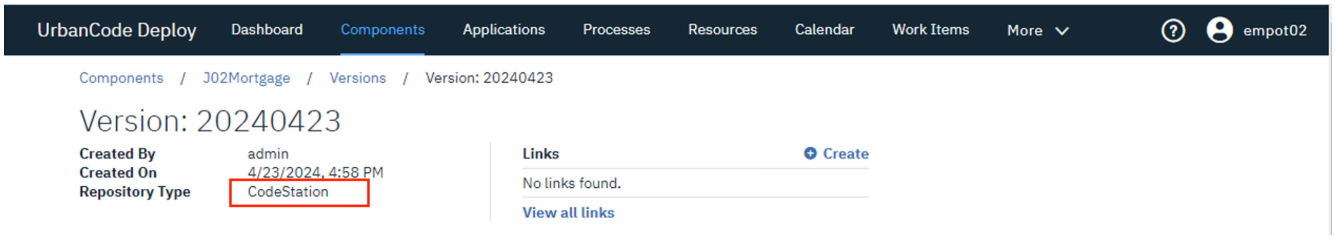
4.14 Now Using the UCD browser, click **Components -> J02Mortgage**



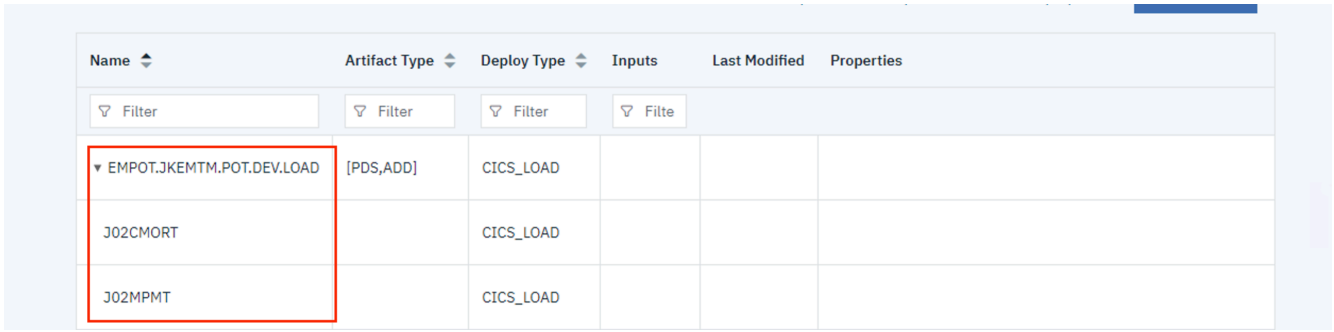
4.15 Click , 1 **Versions** and the 2 version that you created (**yyyymmdd**).



4.16 You can verify that the repository is on UCD Server Code Station (on Linux, not z/OS USS Files)



4.17 Expand **EMPOT.JKEMT.POT.DEV.LOAD** and you should be able to see the list load modules that are packaged and stored in the Code Station (from the JCL that you submitted before).
Optionally the Code Station could be on the z/OS under USS folders.
In our example the Code Station is on the UCD server (LINUX):

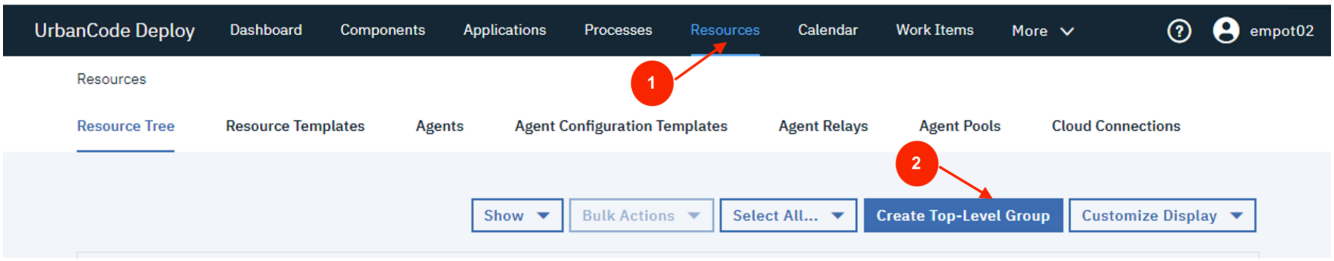


Task 5 - Create UCD Resources

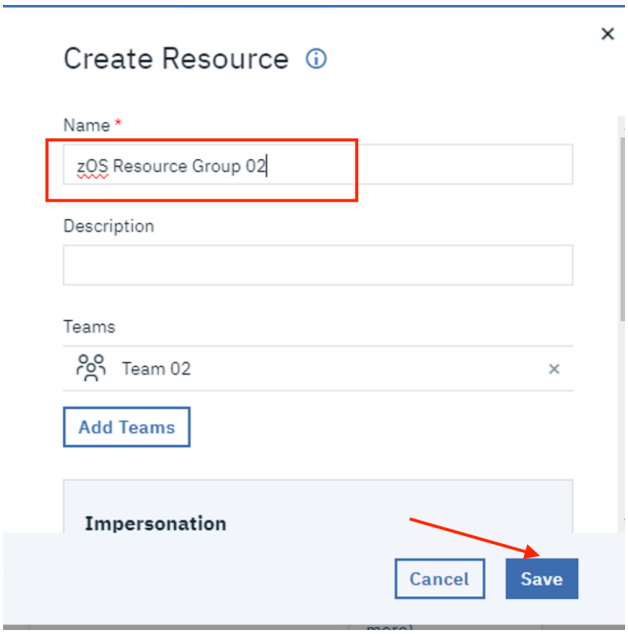
On this task you will create the UCD resources that is a user-defined construct.

UCD : What is a resource?
A resource is a logical deployment target that typically resolves to an agent and a user-defined construct that is based on the architectural model of UCD. Resources can contain other resources in a hierarchical tree structure (called also resource groups or folders) to represent complex targets. Resources are assigned to environments.

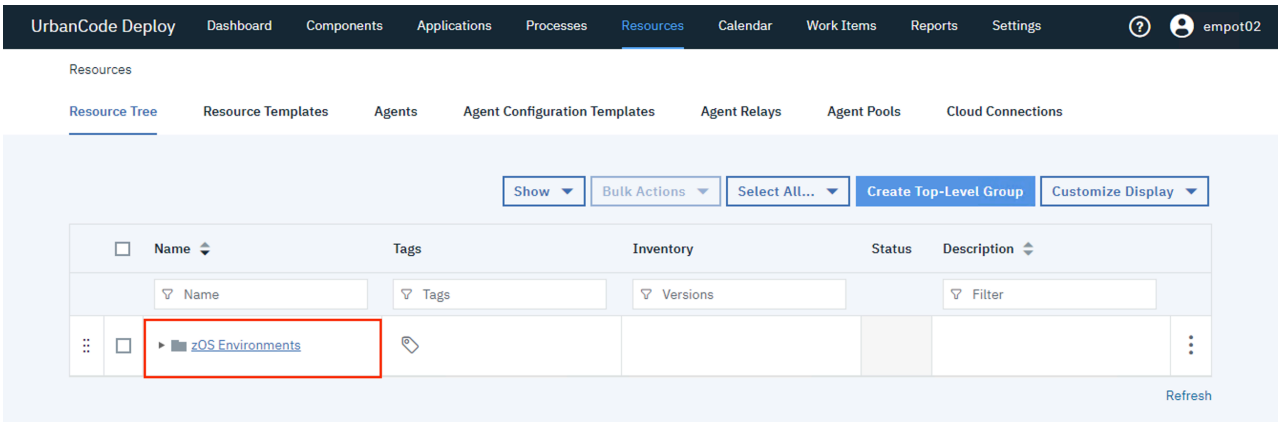
5.1 On the **Resources** tab, click **Create Top-Level Group**:



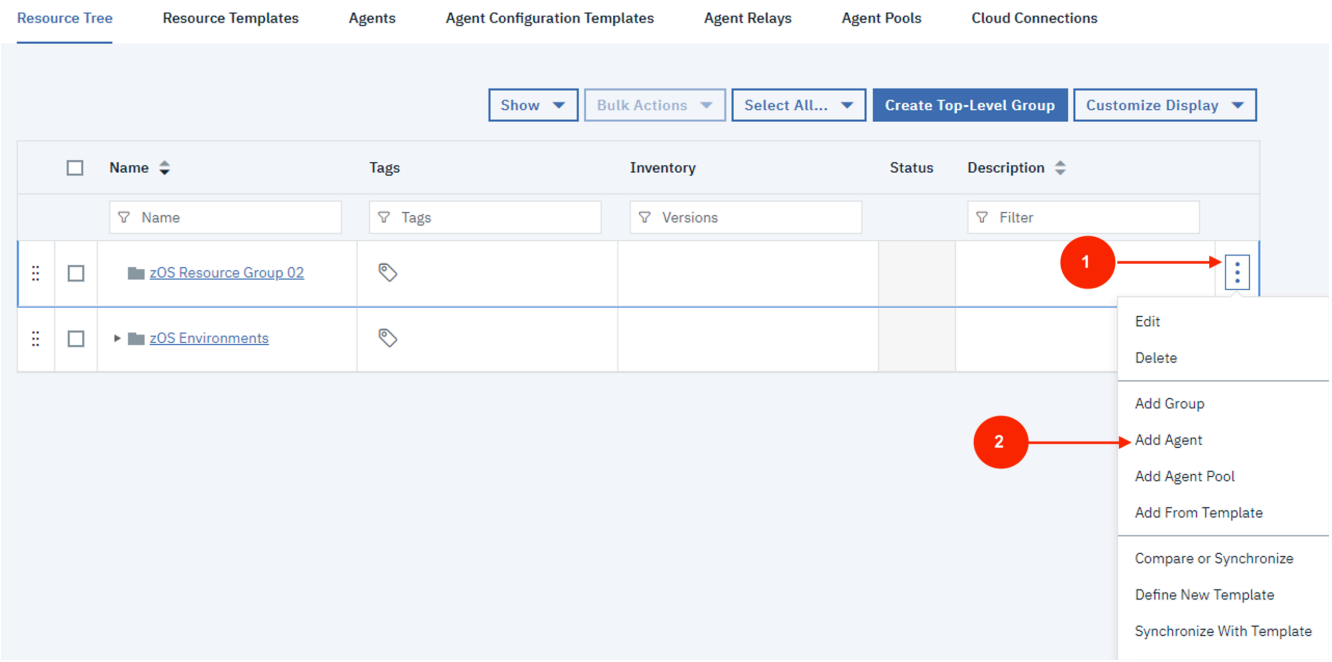
5.2 Type **zOS Resource Group 02** and click **Save.**



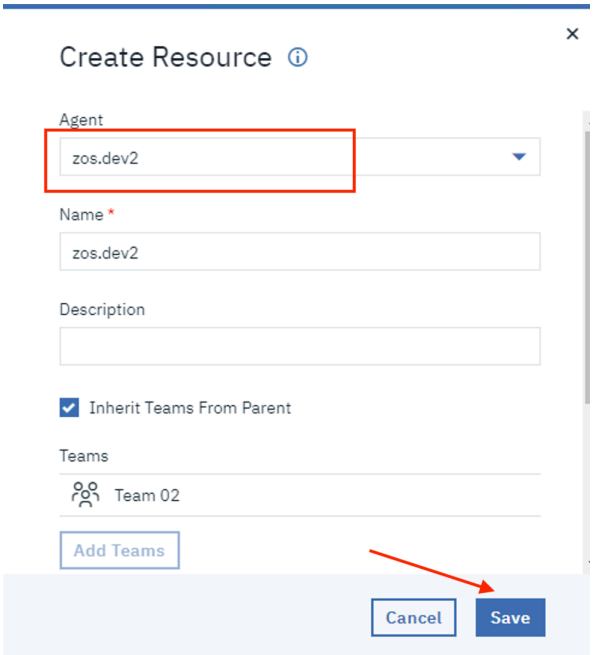
You should have:



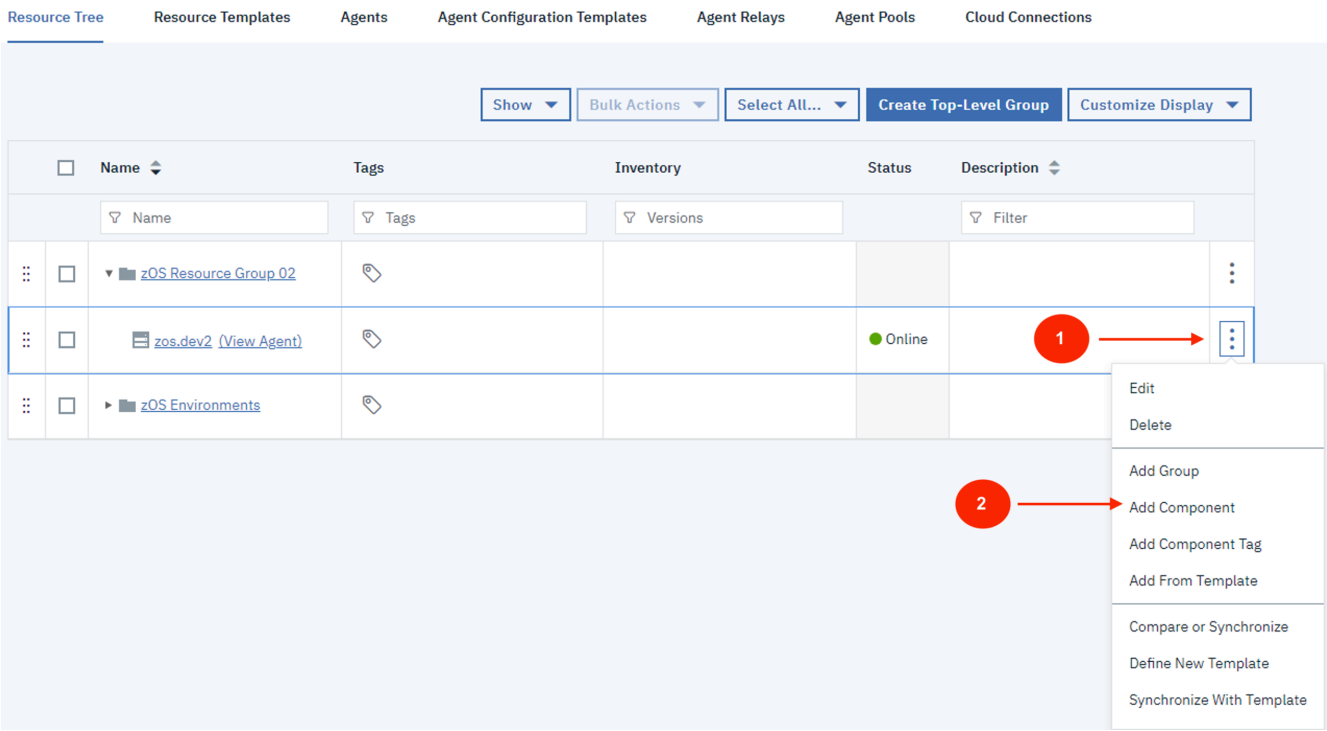
5.3 Move the mouse to **ZOS Resource Group**, click the ••• (three dots) and using the drop-down menu click **Add Agent**:



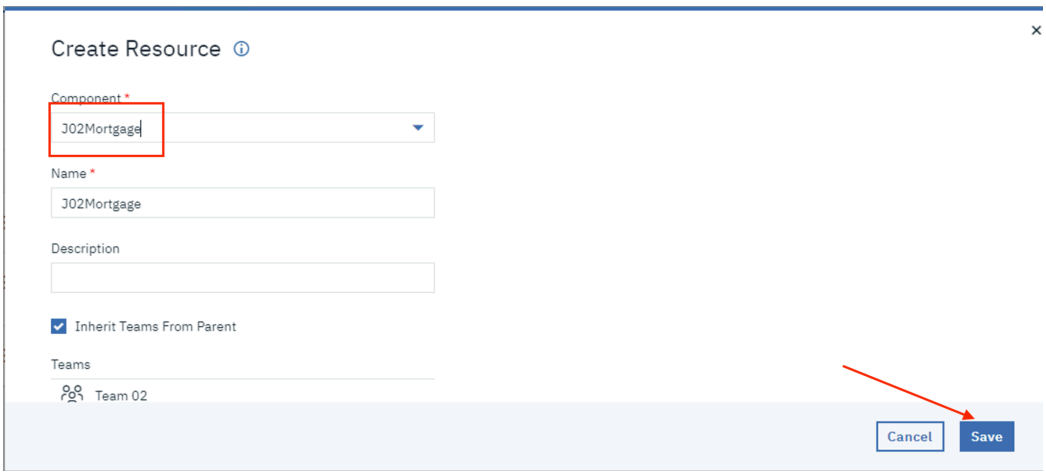
5.4 In the **Agent** drop down dialog, select **zdtagent** and click **Save**



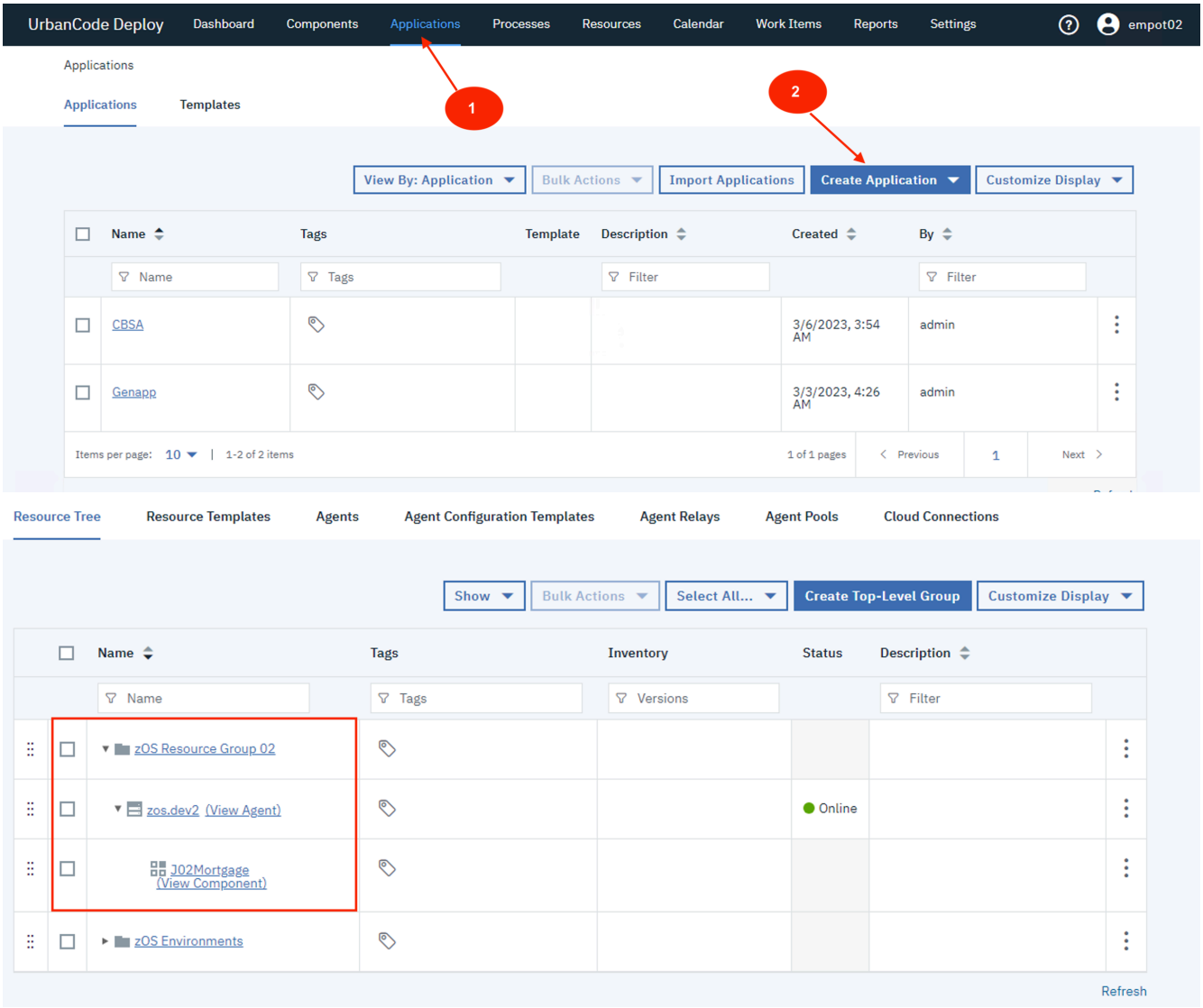
5.5 In the row **zdtagent,** , click the ••• (three dots) and using the drop-down menu click **Add Component**:



5.6 Select the component created earlier (**J02Mortgage**) and click **Save**.



You should have:



Task 6 - Create an UrbanCode Application

On this task you will create an UCD application.

Applications are responsible for bringing together all the components that must be deployed together.

UCD : What is a UCD Application?

Application is a module the contains the following elements:

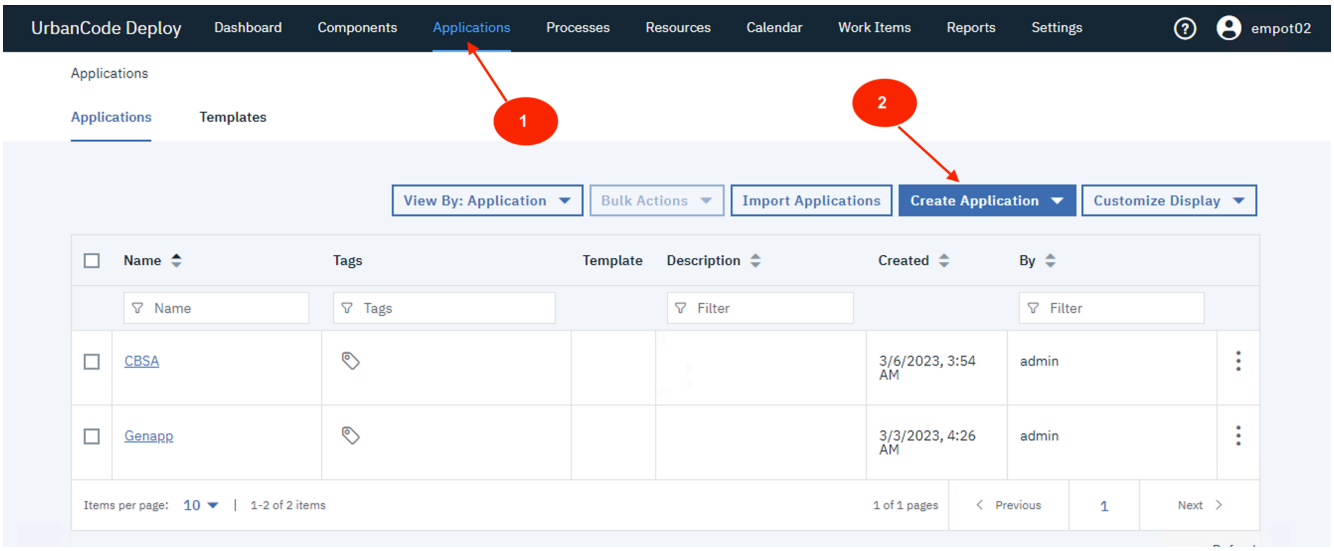
- Components to be deployed
- Environments target and the resources to deploy the application.
- A process definition (note that application process runs on the server while component process runs on agent)
- Properties

UCD: Properties

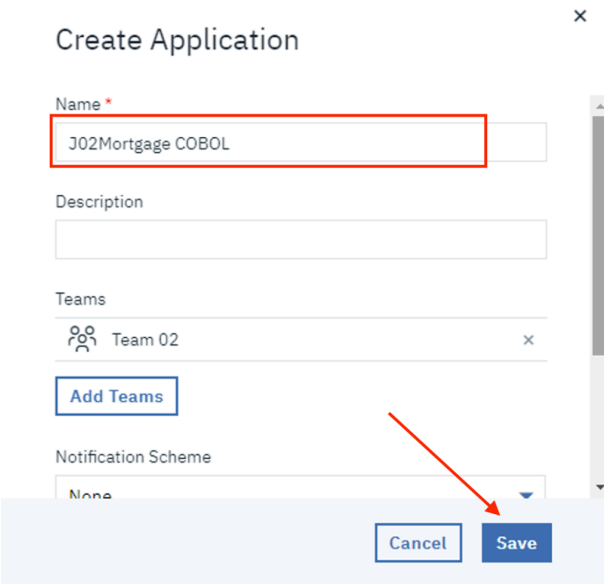
Properties can be set in UCD for many different things, including components, environments,

processes, and applications. You can also set global properties for the system.
You can refer to a property by scope:
${p:scope/propertyName} for example: ${p:environemt/DBconnect}
or without scope:
${p:propertyName}
You define objects (Application, component, process, resource, etc) properties in the object's Configuration Tab of UCD browser interface.
For more details, refer to the full documentation on Properties:
http://www-01.ibm.com/support/knowledgecenter/SS4GSP_6.2.6/com.ibm.udeploy.reference.doc/topics/ud_properties.html

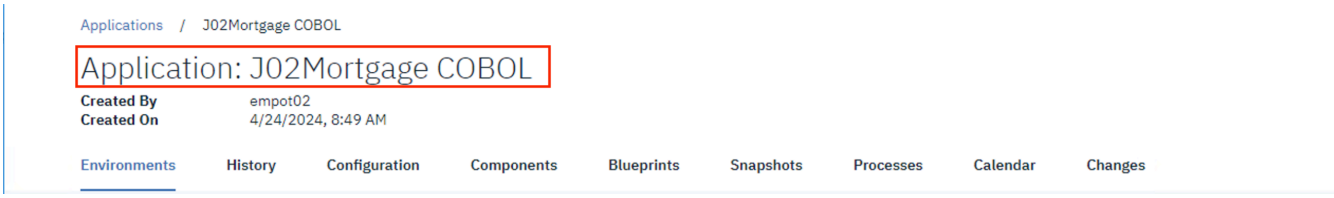
6.1 On the **Applications** tab, click **Create Application**:



6.2 Provide the value for the *Name* field in the *Create Application* dialog
Name it **J02Mortgage COBOL** and click **Save**



The result should be:

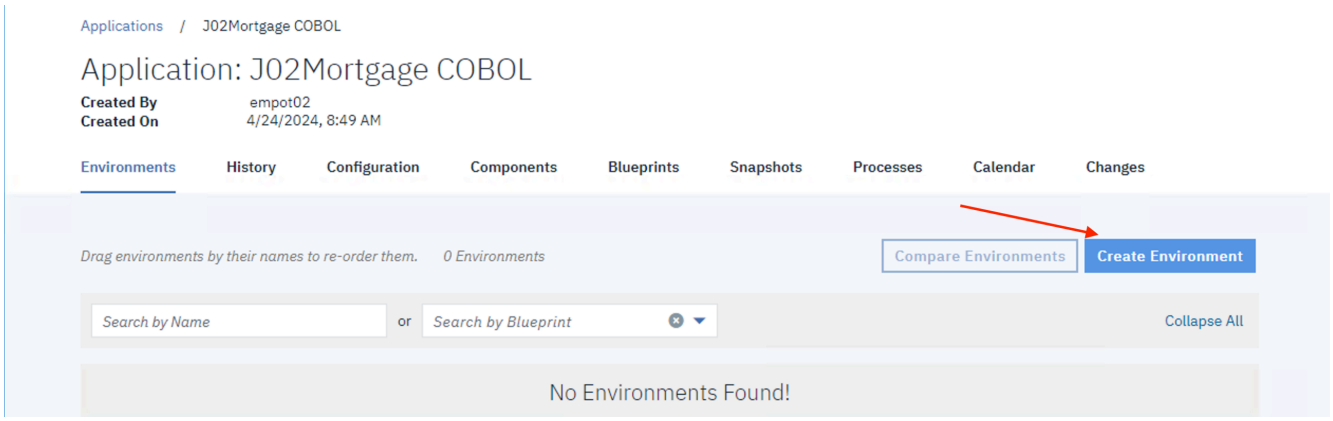


Task 7 - Create environment

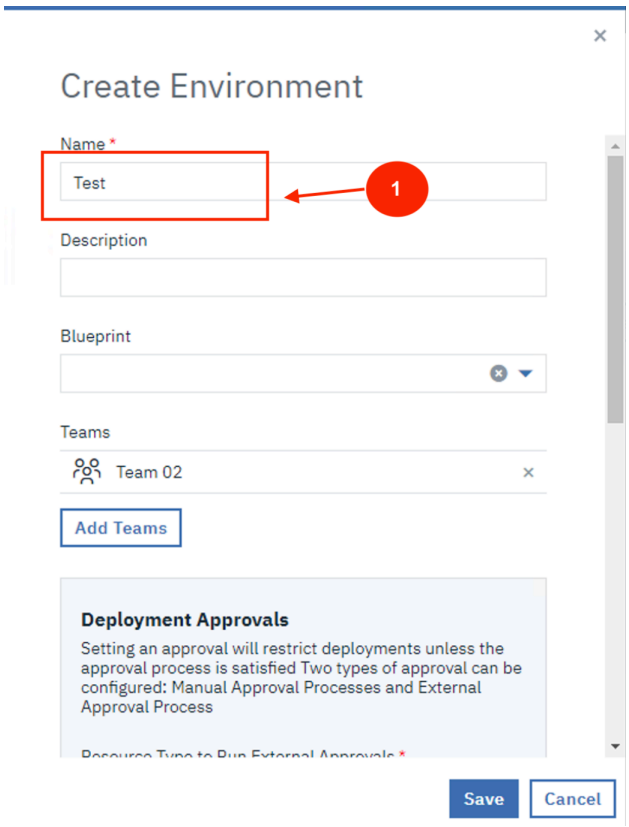
On this task you will you create The UCD environment that is a user-defined collection of resources that hosts an UCD application.
When you create an environment, you map resources to it that define where the parent application can run deployments.

UCD : What is a UCD Application Environment?
An environment is the application's mechanism for bringing together components with the agent that deploys them. Environments are typically modeled on some stage of the software project lifecycle, such as development, test, or production.
When you create an environment, you map resources to it and the application components that will be deployed.
You can also define environment specific properties.

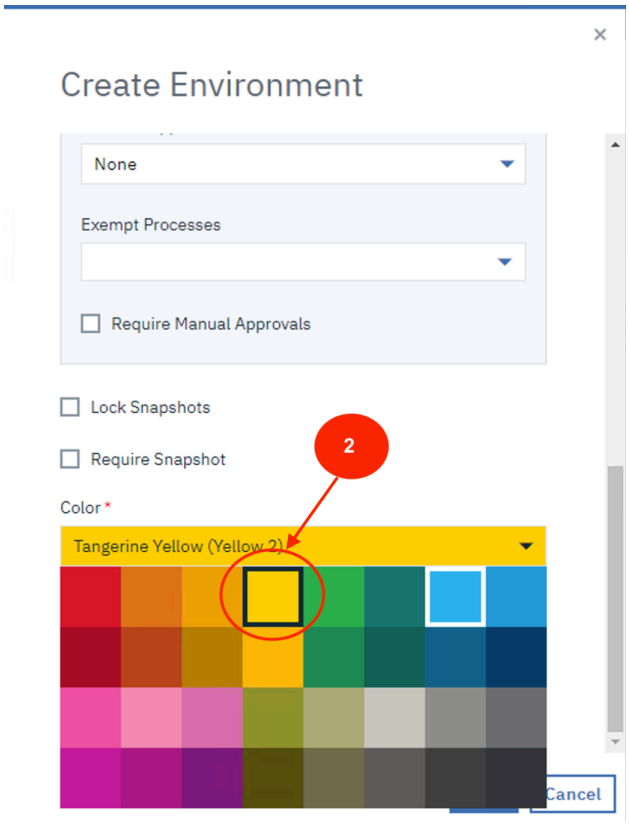
7.1 Click **Create Environment**:



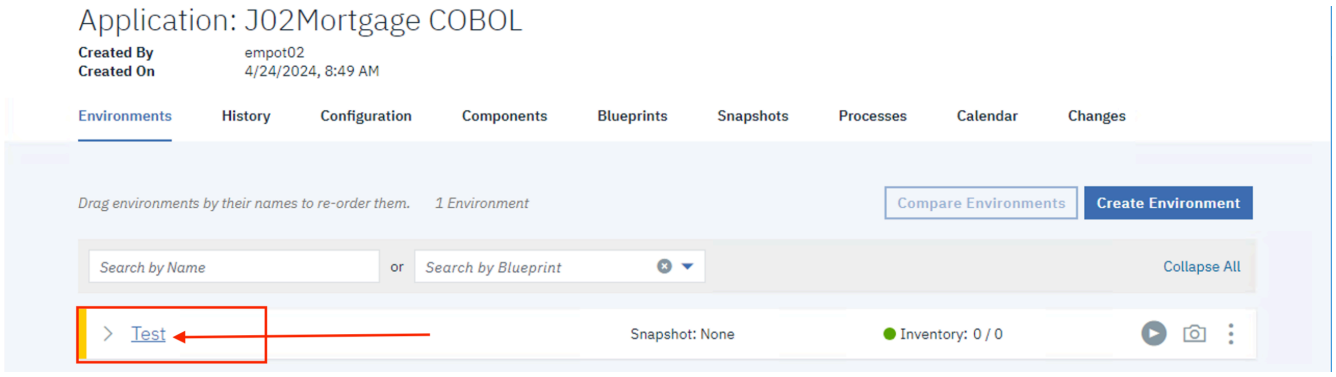
7.2 Provide the value for the Name field in the Create Environment dialog (for example, **Test**), choose a color, and click **Save**:



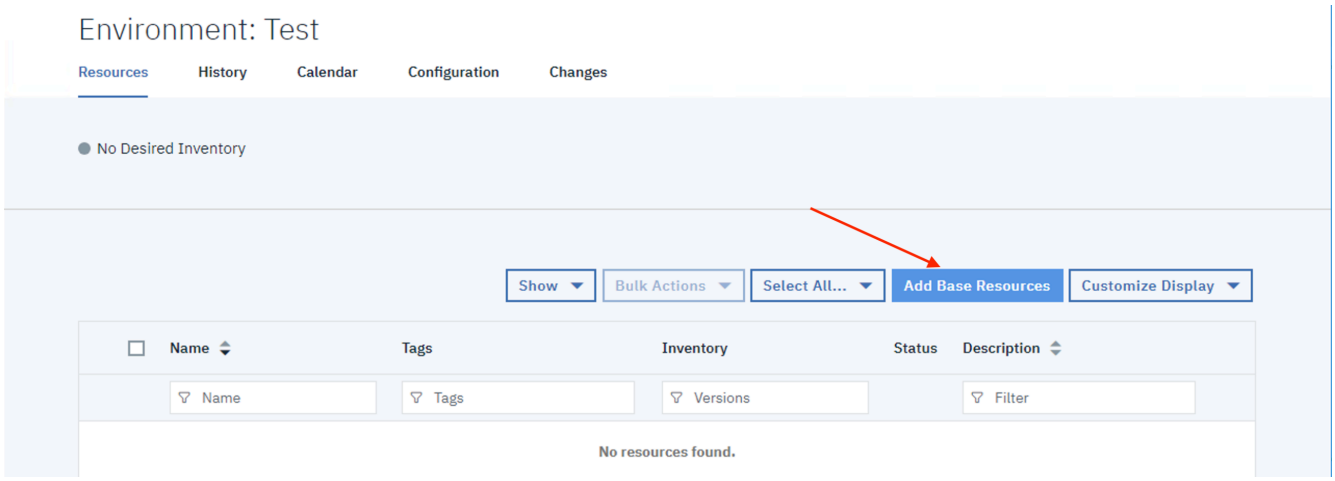
And **scroll-down** and select the color (**Orange**) for your test and Click **Save .**



7.3 Click the environment name (**Test** in our example) to open environment properties:



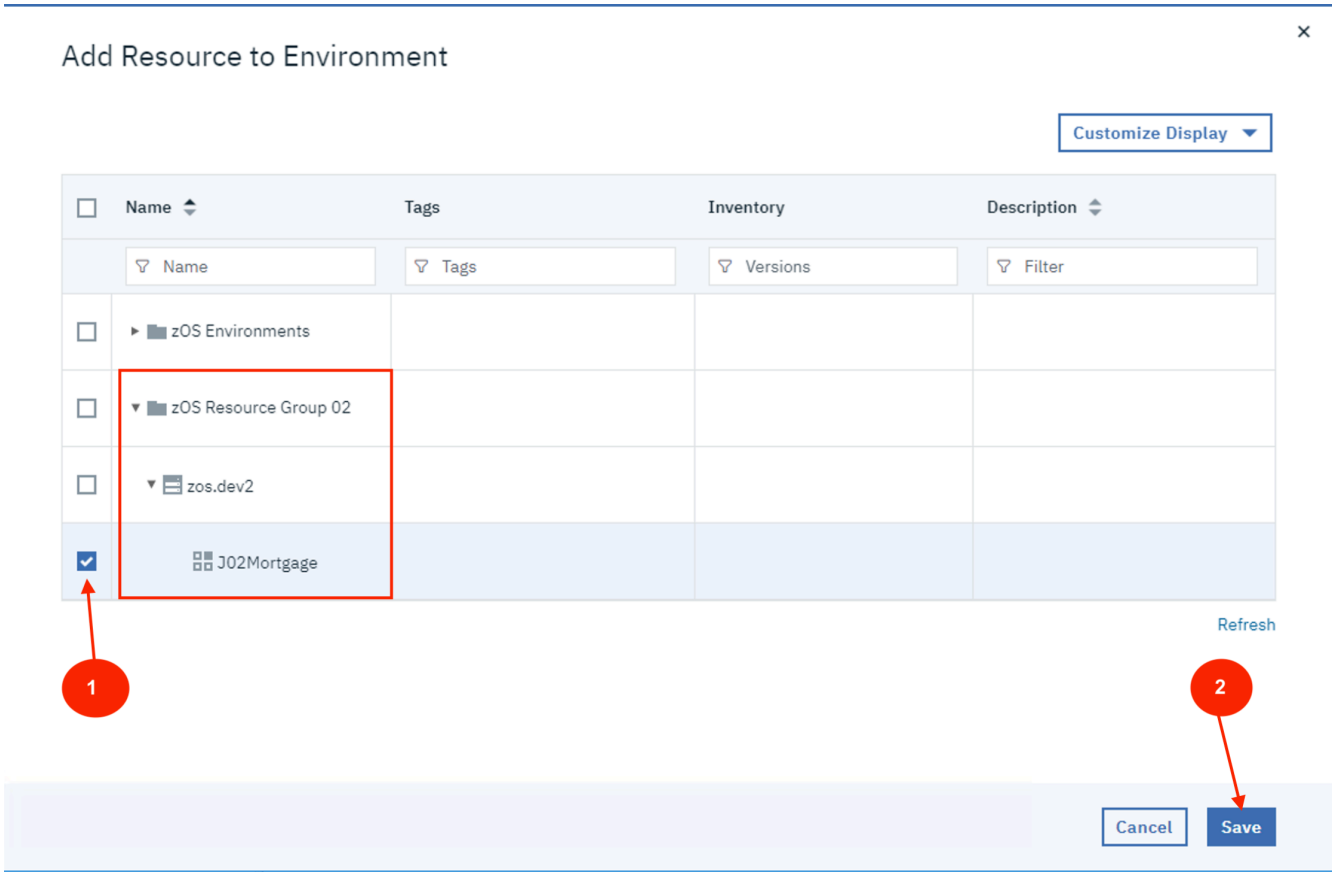
7.4 Click **Add Base Resources**:



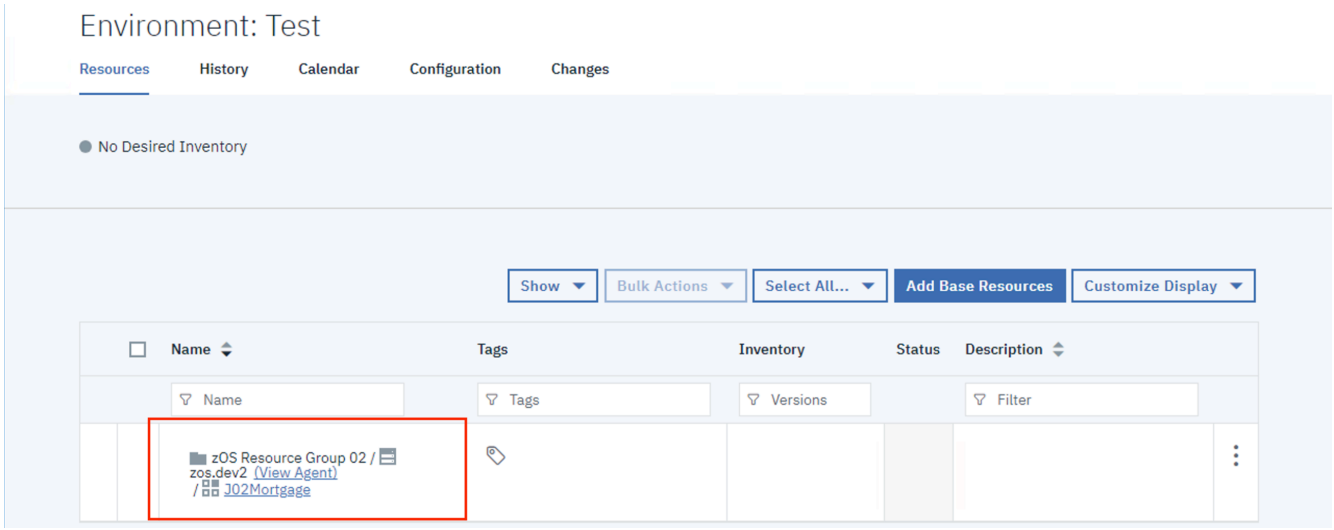
7.5 Expand **zOS Resource Group 02** and **os.dev2.**

Select Component **J02Mortgage**: and click **Save**:

Note: In order for a component to be deployed by an application, it must be added to the application and also mapped to an agent-type resource. A component that is added to an application but not mapped to an agent resource, cannot be deployed by that application.
Similarly, a component that is mapped to an agent resource but not added to an application, cannot be deployed by that application.



You should have this:



PART 2 - Create the UrbanCode deployment processes.

On this part you will you create a deployment process for your component and the application process that uses the component process to deploy the component.

Processes are automated tasks that run on agents.
There are three types of processes:

- **Generic processes** run outside the context of components or applications. Not used on this lab.
- **Application processes** run within the context of applications. In many cases, application processes call component processes. For example, an application process can call the component processes that deploy those components.
- **Component processes** run tasks on a single component, such as deploying it, uninstalling it, or running configuration tasks on it.

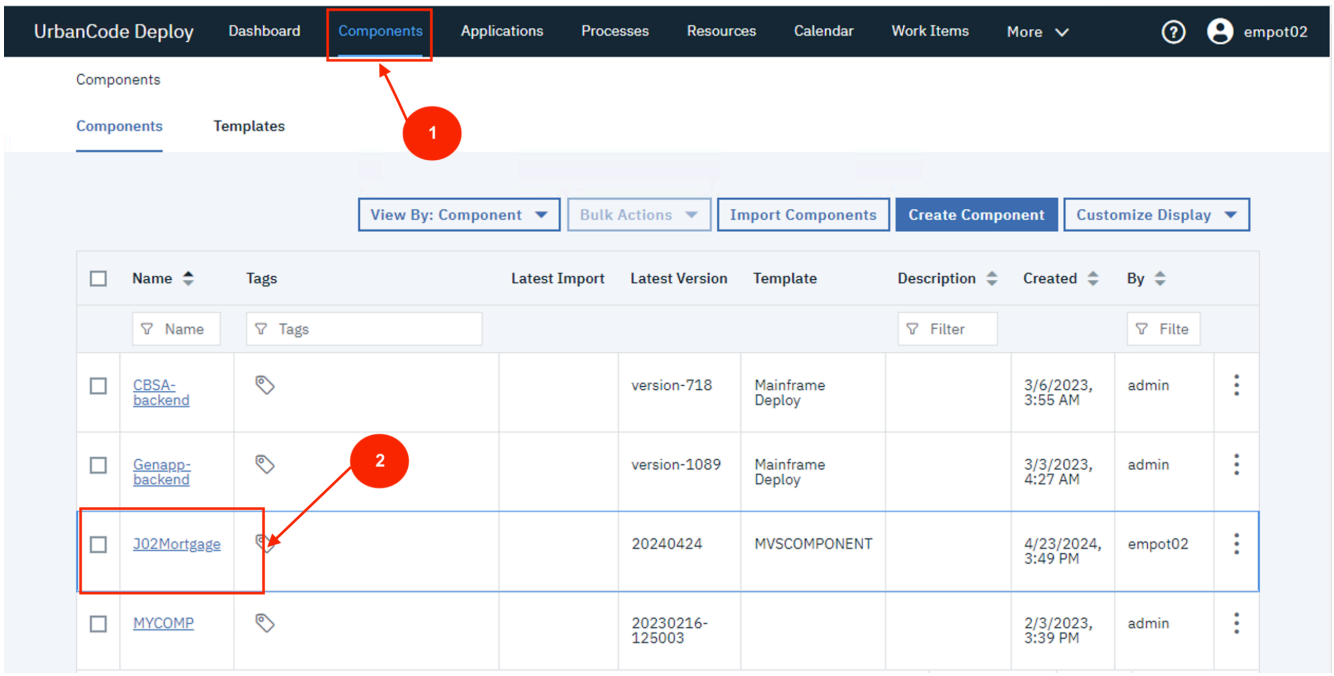
Task 8 - Create a components process

A component process is a succession of commands that are called steps. Steps can manipulate files, run system commands, set properties, pass information to other steps, and run programs. Steps are provided by automation plug-ins. Processes are designed with the drag-and-drop process editor where you drag plug-in steps onto the design editor and configure them as you go. Several plug-ins come with the product and others are available, which work with many different types of software. In this lab you use z/OS plug-ins. A component can have any number of processes defined for it, but a component must have at least one process.

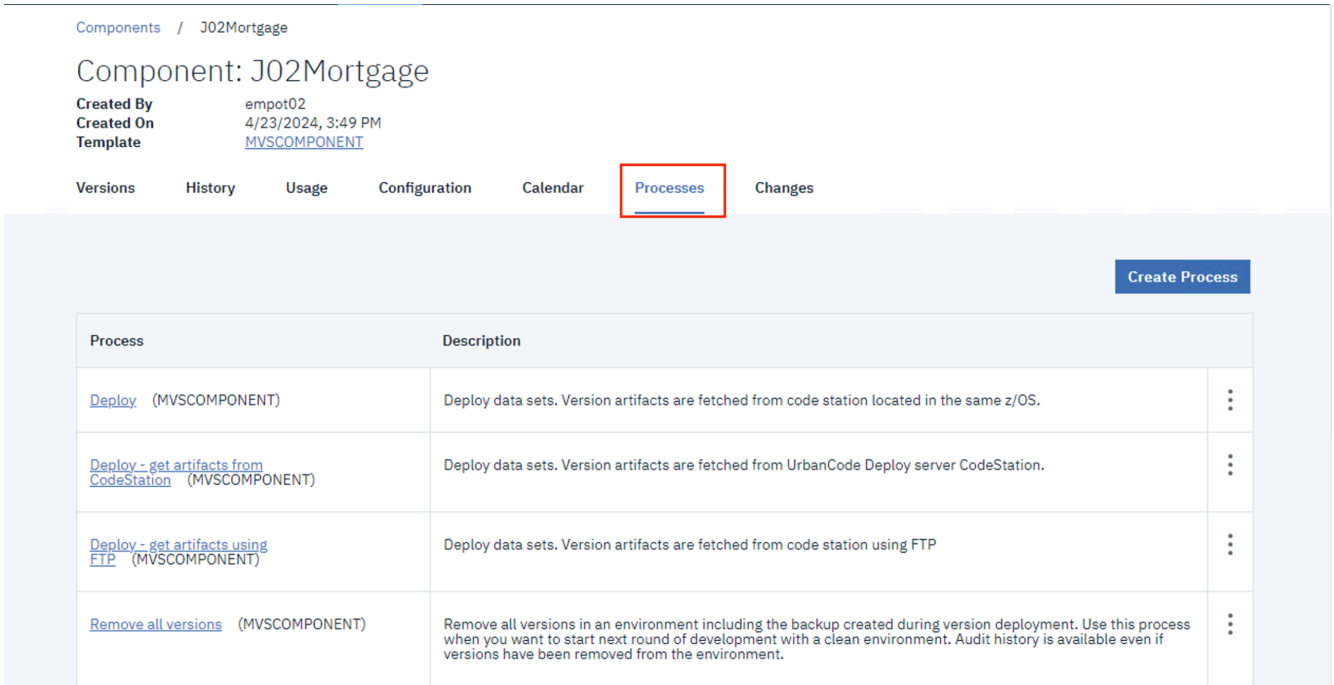
In this task you create a deployment process for your component. Later, you create an application process that uses the component process to deploy the component.

We are going to create a **component process** for this example. This process will automate deployment of the new version of our **J02Mortgage COBOL** application.

8.1 ⏭ Click on tab **Components,** find the component created earlier (**J02Mortgage**) and click on it



8.2 ⏭ Click the **Processes** tab

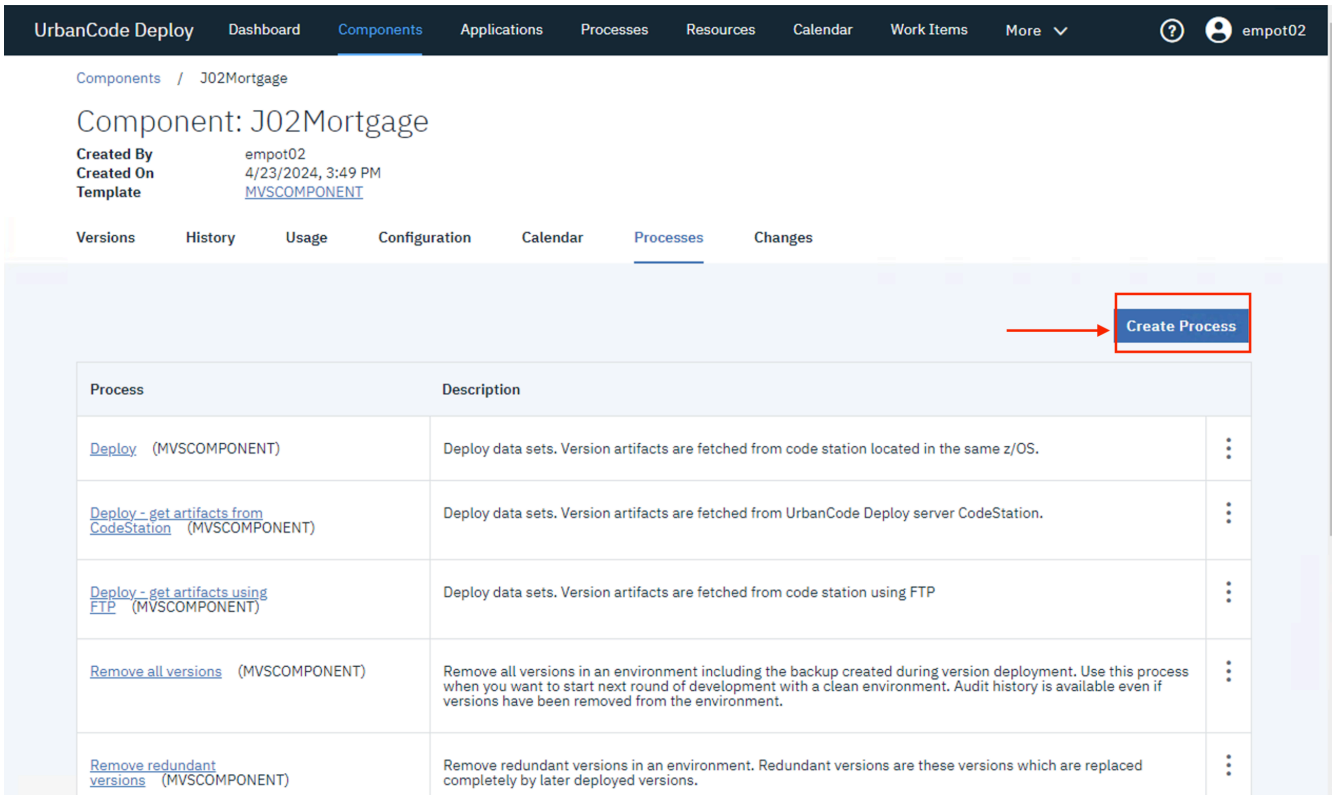


You will notice that there is a small collection of processes that was created automatically for your component (**Deploy, Remove all versions, Uninstall**, and so on). Remember that those processes were

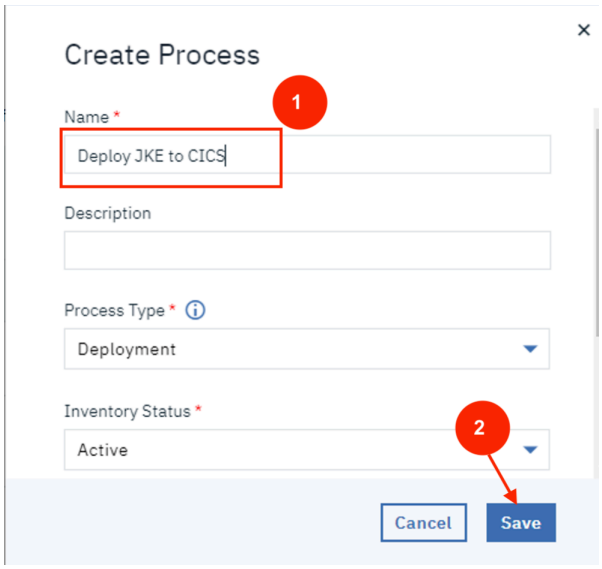
created from a template.
These processes can be used by themselves to perform simple tasks or can be used as starting points for more complex processes.

We will next create our deployment process. We could slightly simplify this task by reusing the existing process called **Deploy**. However, for the purposes of this example, we are going to create a brand new one.

8.3 Click **Create Process** button to get started.

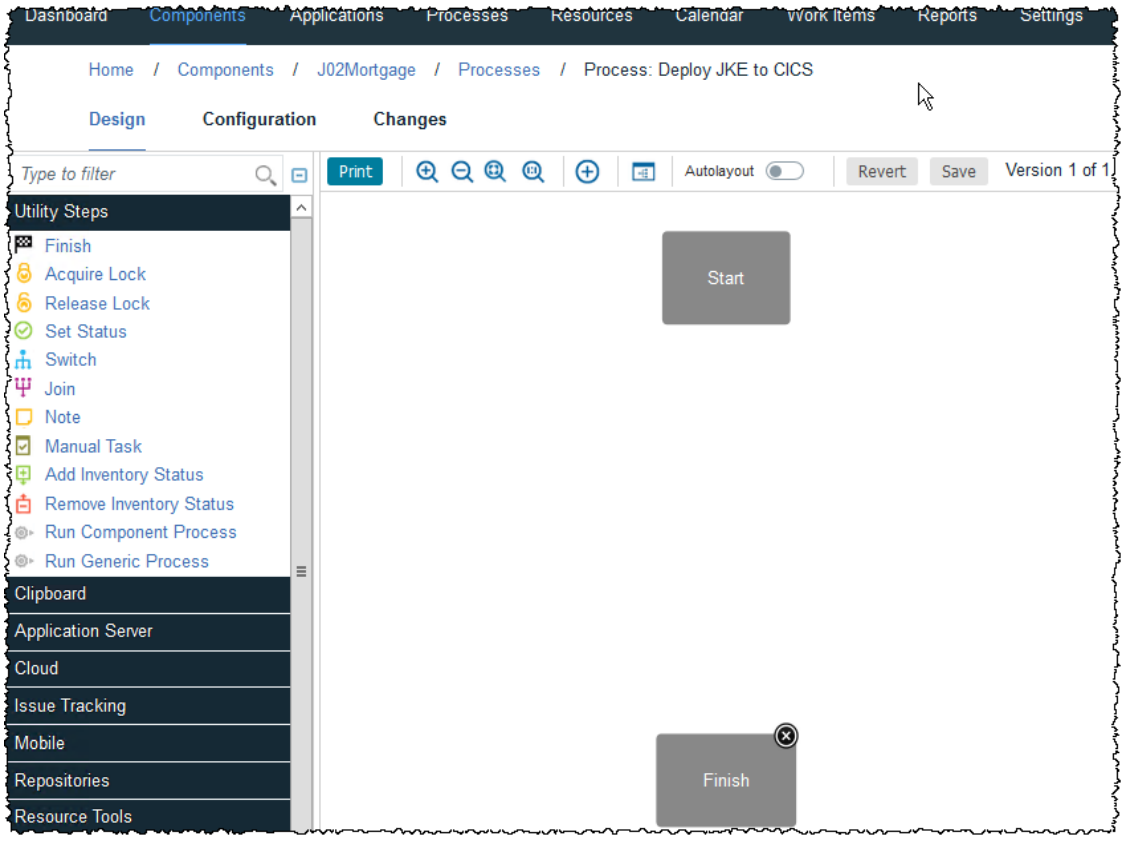


8.4 Enter Process Name (**`Deploy JKE to CICS`**)
Leave all other options at their default values and click **Save**:



8.5 The **Process Editor** will open.
Here, you can organize the steps of a process, specify their properties, and connect them to each other.
As usual, you can refer to UCD documentation for detailed information on editing processes.
If you have internet access you can go to the link below for more details::

https://www.ibm.com/support/knowledgecenter/SS4GSP_7.0.2/com.ibm.udeploy.doc/topics/comp_workflow.html

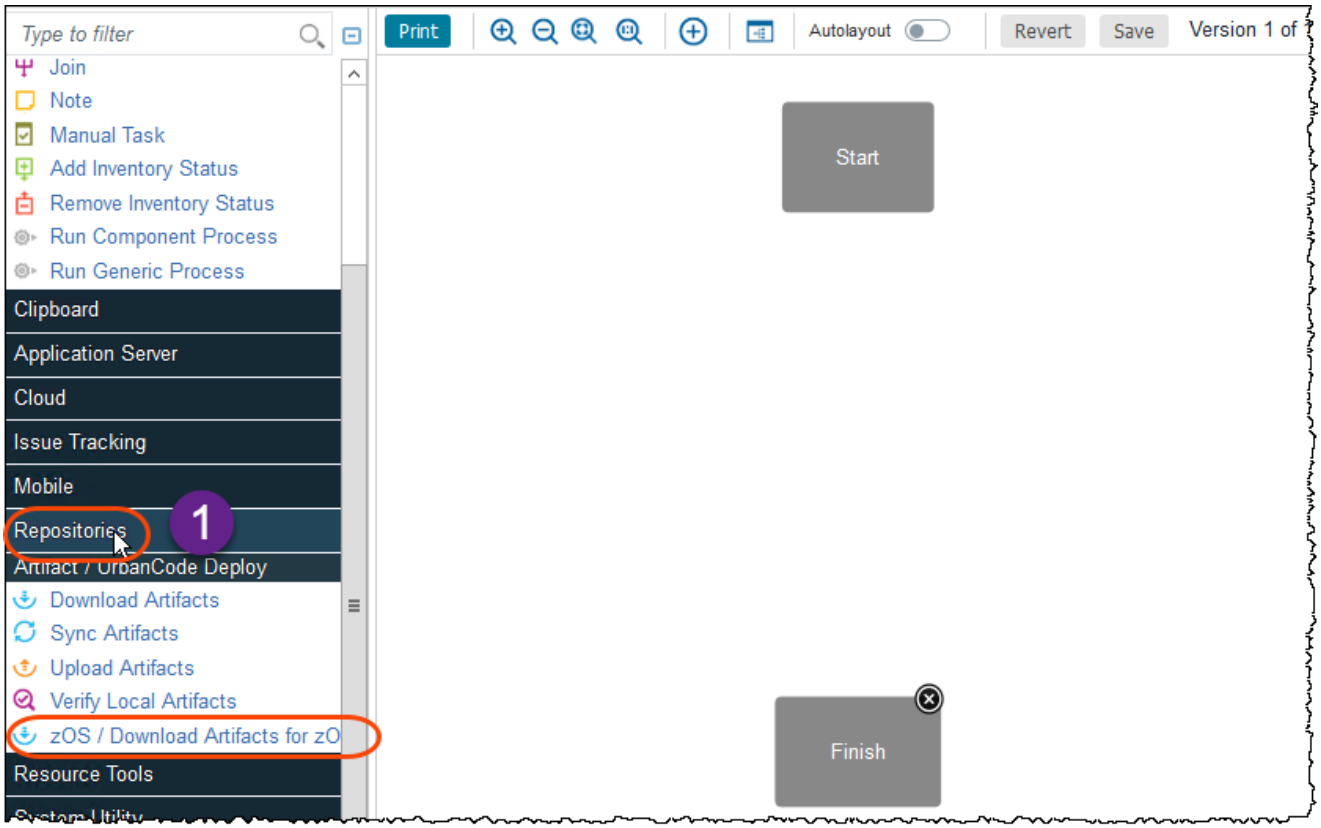

Dashboard
Components
Applications
Processes
Resources
Calendar
Work Items
Reports
Settings
Home / Components / J02Mortgage / Processes / Process: Deploy JKE to CICS
Design
Configuration
Changes
Type to filter
Print
Autolayout
Revert
Save
Version 1 of 1
Utility Steps
Finish
Acquire Lock
Release Lock
Set Status
Switch
Join
Note
Manual Task
Add Inventory Status
Remove Inventory Status
Run Component Process
Run Generic Process
Clipboard
Application Server
Cloud
Issue Tracking
Mobile
Repositories
Resource Tools
Start
Finish

8.6 We are going to create a simple process for updating an existing CICS application.

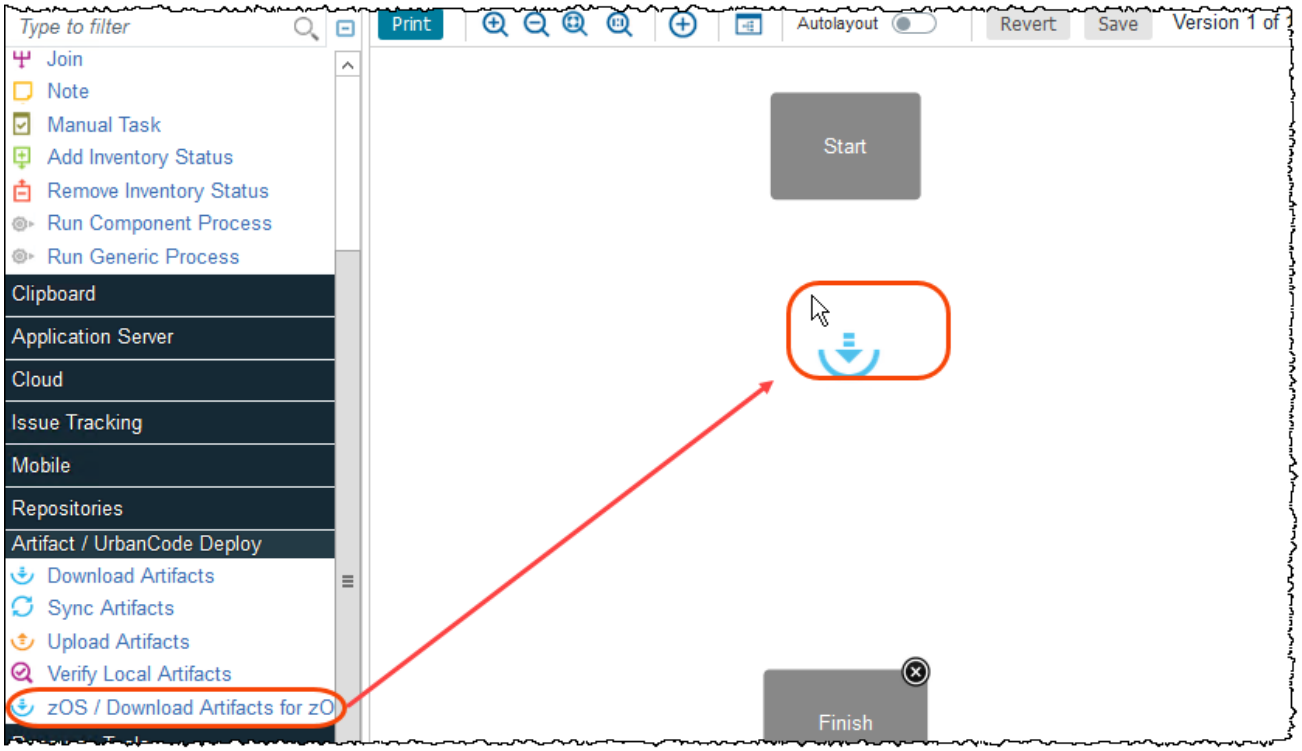
⏭ On the left-hand side of the editor you will find an extensive **Palette** of process steps.

⏭ Scroll down and expand **Repositories** .

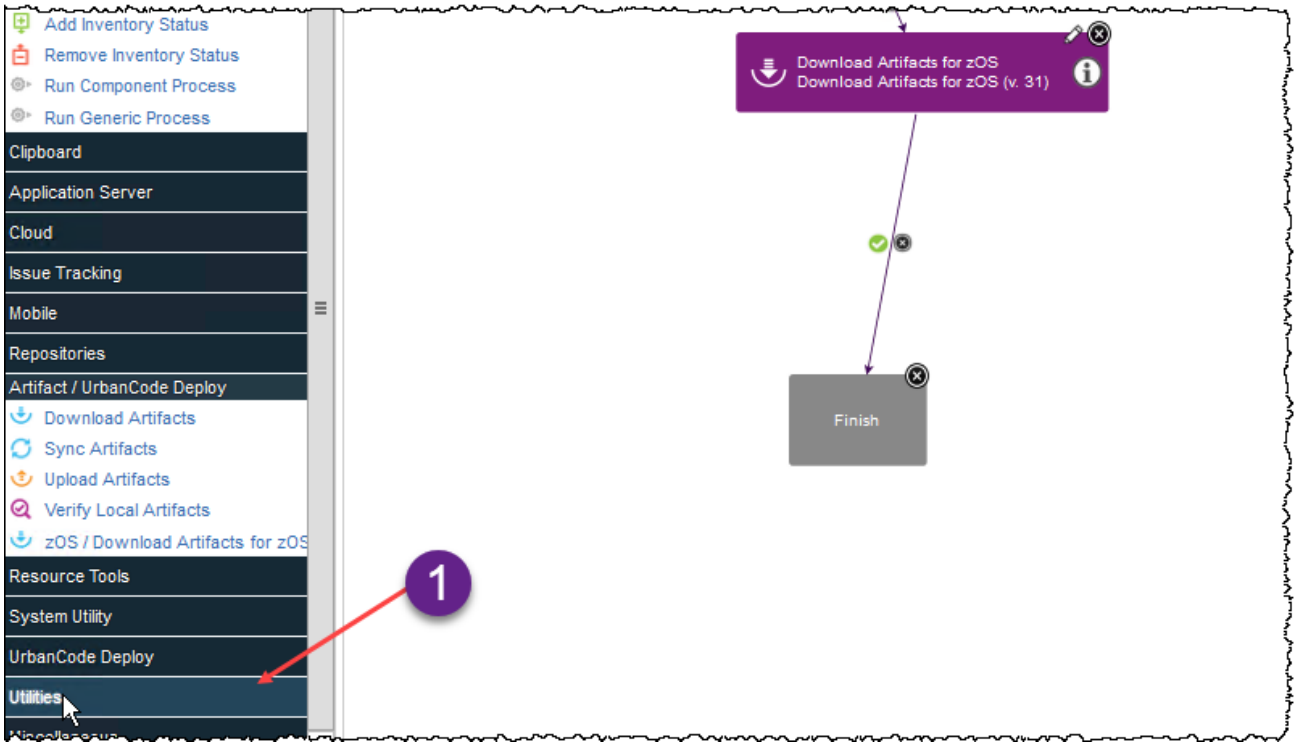
First you are going to use the step **zOS /Download Artifacts for zOS**:



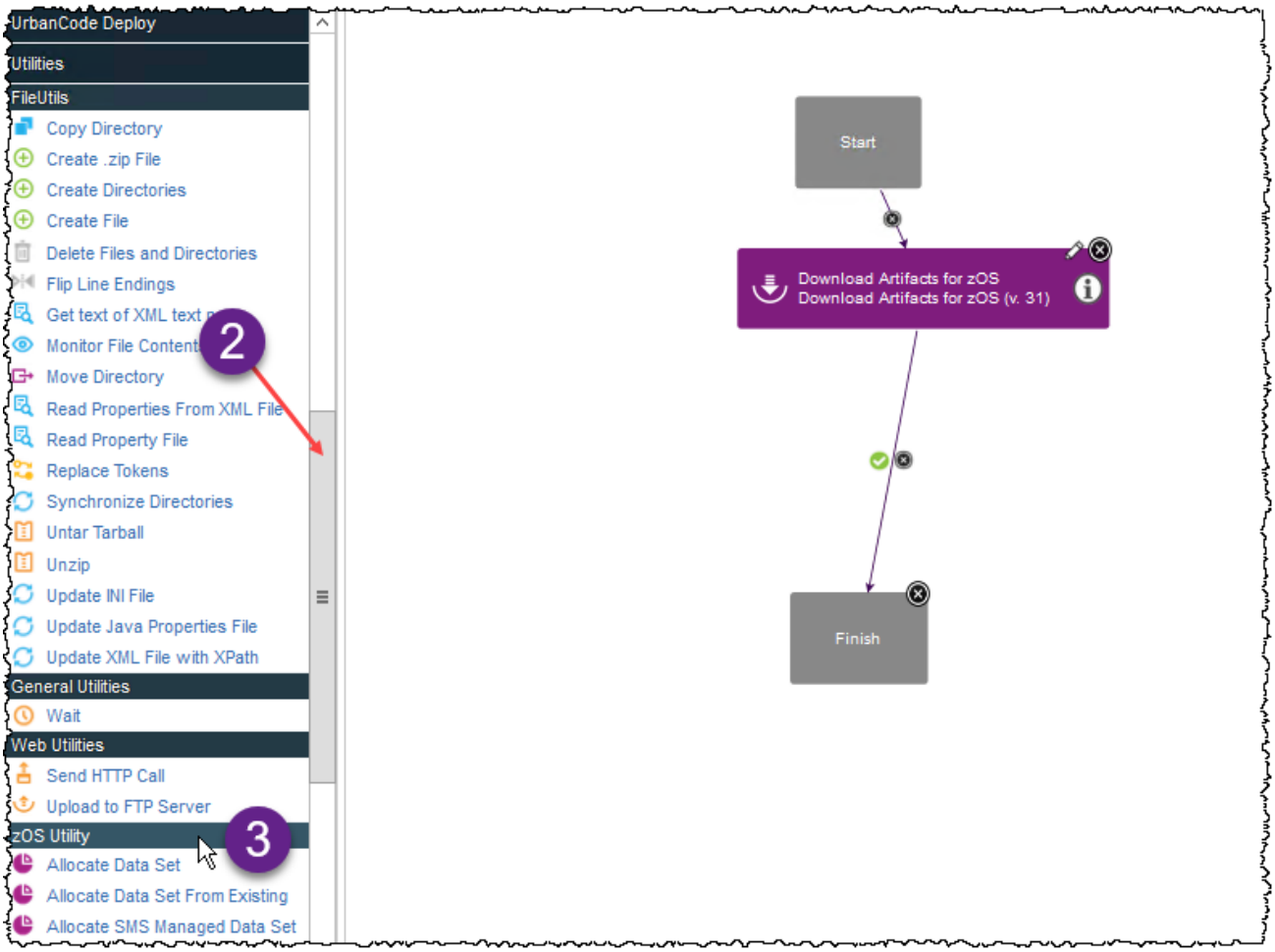
8.7 Drag and drop the first step, **zOS /Download Artifacts for zOS**, onto the design space (somewhere under the **Start** block).



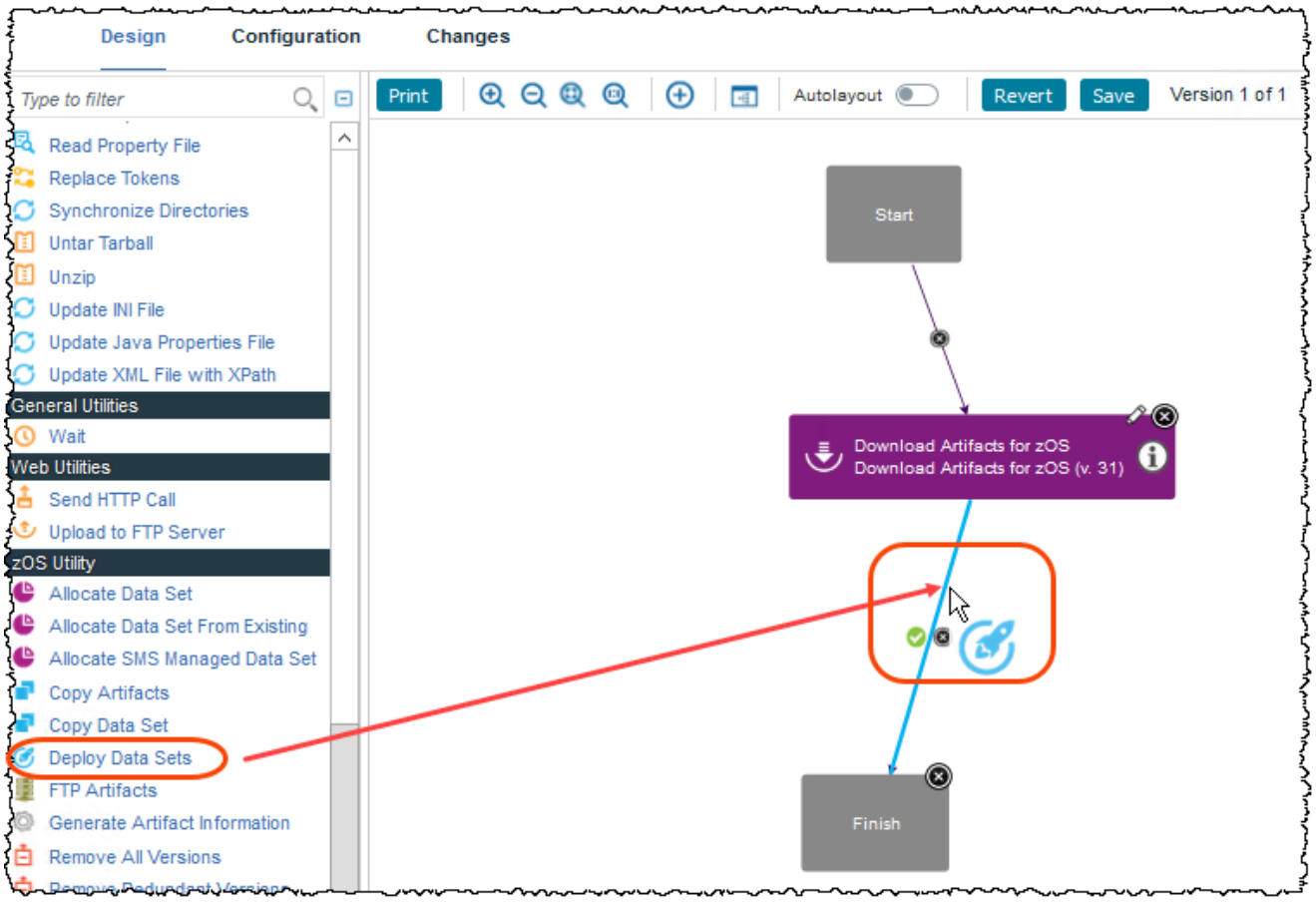
8.8 1 Using the Design palette on left expand **Utilities,** clicking on it



⏭ 2 Scroll down and 3 expand **zOS Utility**: The **Deploy Datasets** step will also be needed to load component artifacts from the z/OS repository and to bring them into z/OS work area for later deploy.

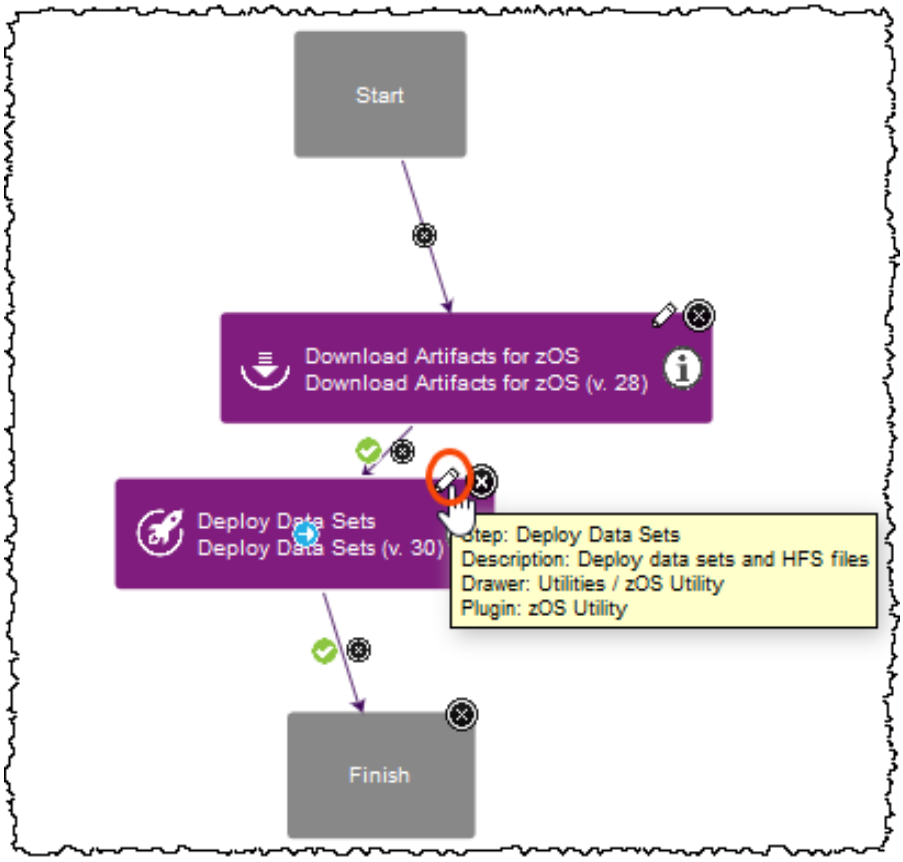


8.9 Scroll down and drag the **Deploy Data Sets** step onto the line that connects to the *Finish* as below

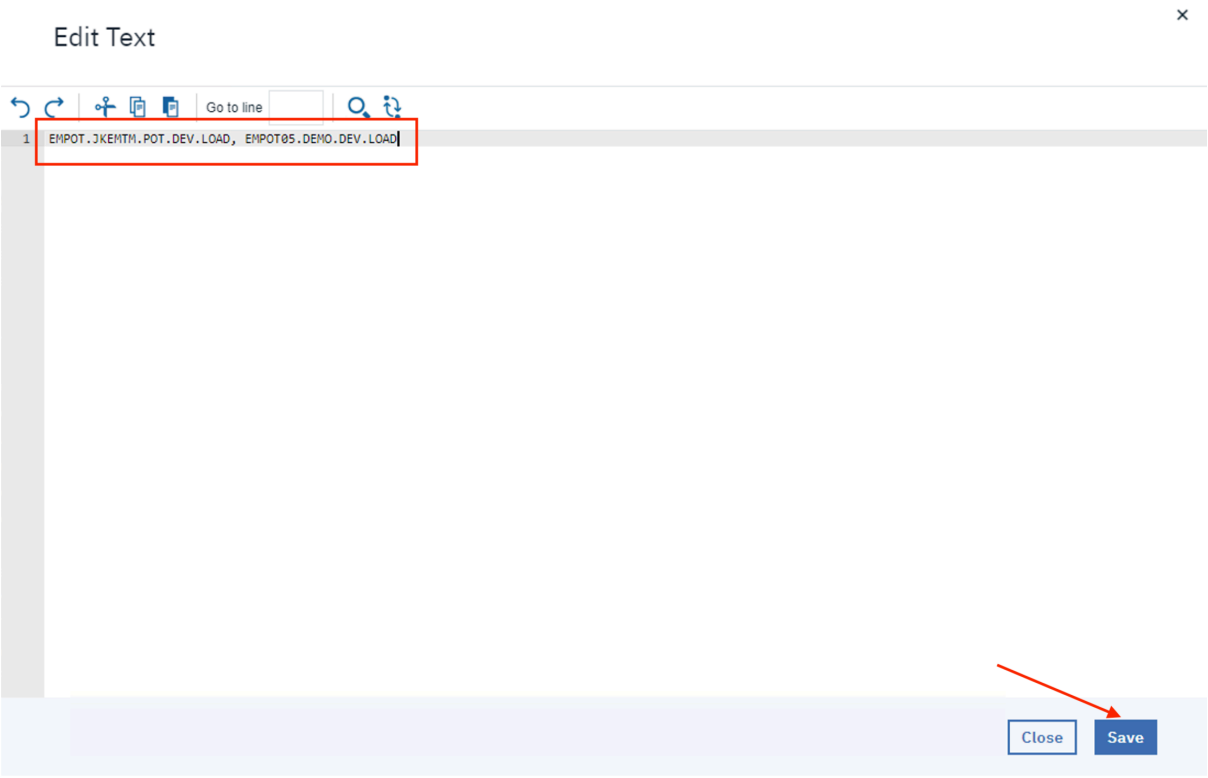


Properties
Properties can be set in UCD for many different things, including components, environments, processes, and applications. You can also set global properties for the system.
You can refer to a property by scope:
${p:scope/propertyName}
or without scope:
${p:propertyName}
For more details, refer to the full documentation on Properties:
http://www-01.ibm.com/support/knowledgecenter/SS4GSP_6.1.1/com.ibm.udeploy.reference.doc/topics/ud_properties.html

8.10 Click on the **pencil** icon on **Deploy Data Sets** to change the UCD properties for this step**.**



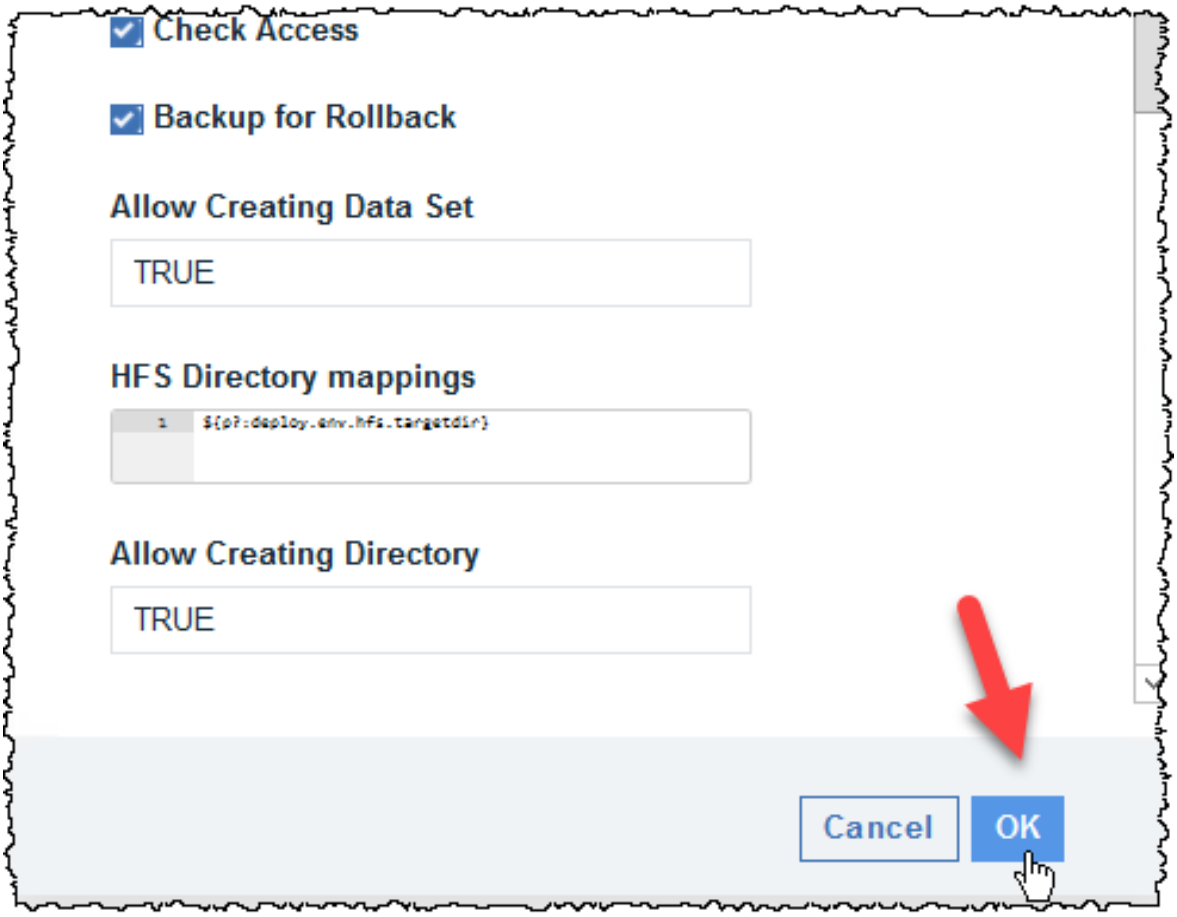
8.11 Enter the value for the *Data Set Mapping* property:
`EMPOT.JKEMTM.POT.DEV.LOAD, EMPOT05.DEMO.DEV.LOAD`



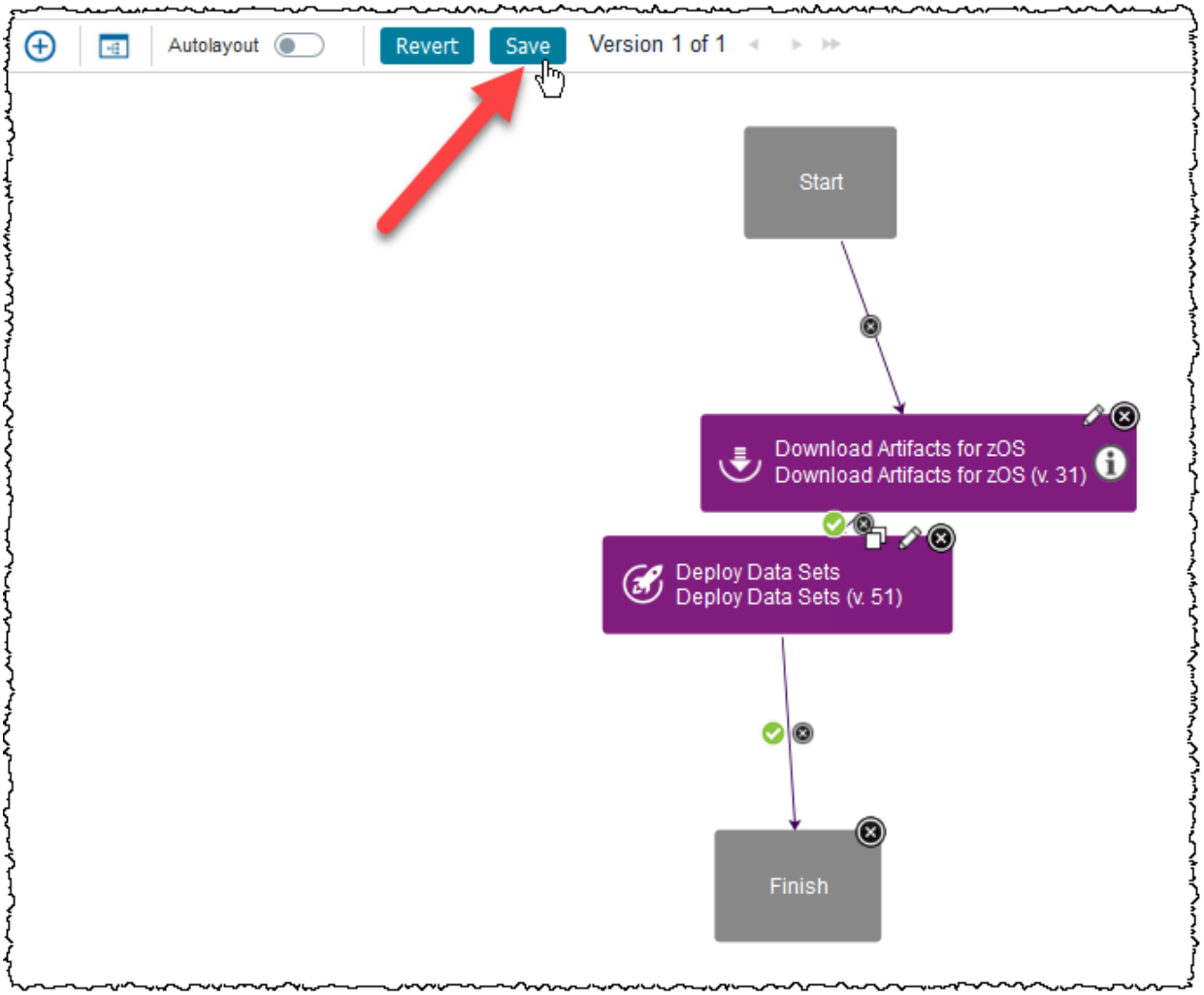
Data Set Mapping specifies which PDS members packaged with the component to deploy, and where to deploy them.
A mapping rule should follow format "From PDS, To PDS":
We could have that as variable but we will fix here to save time.

After entering the values click **Save**

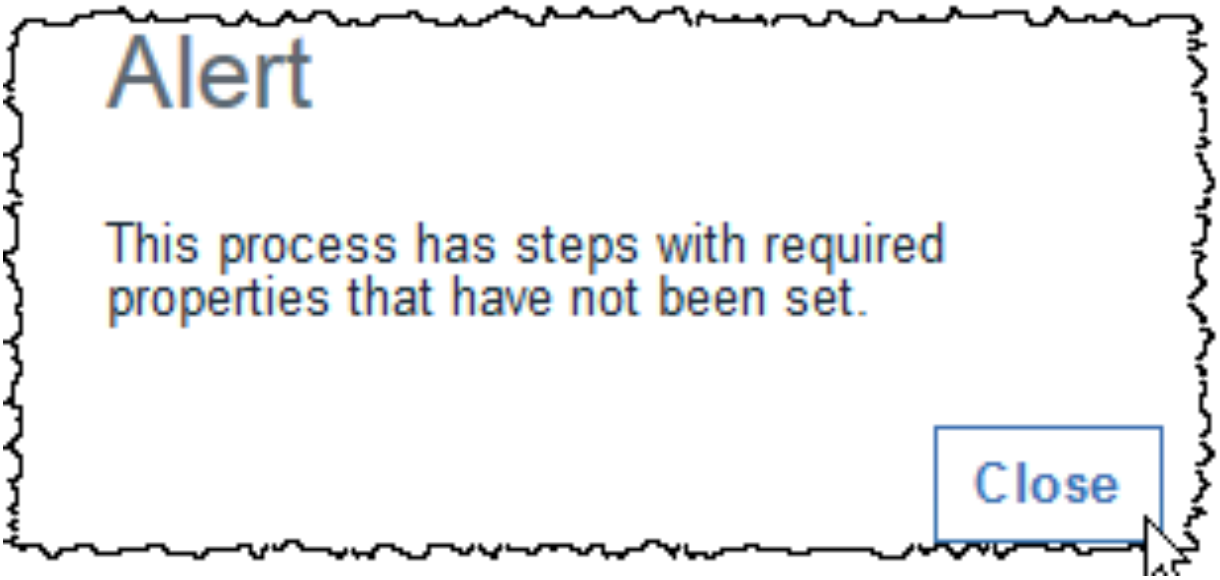
8.12 Click **OK** to save the properties updated



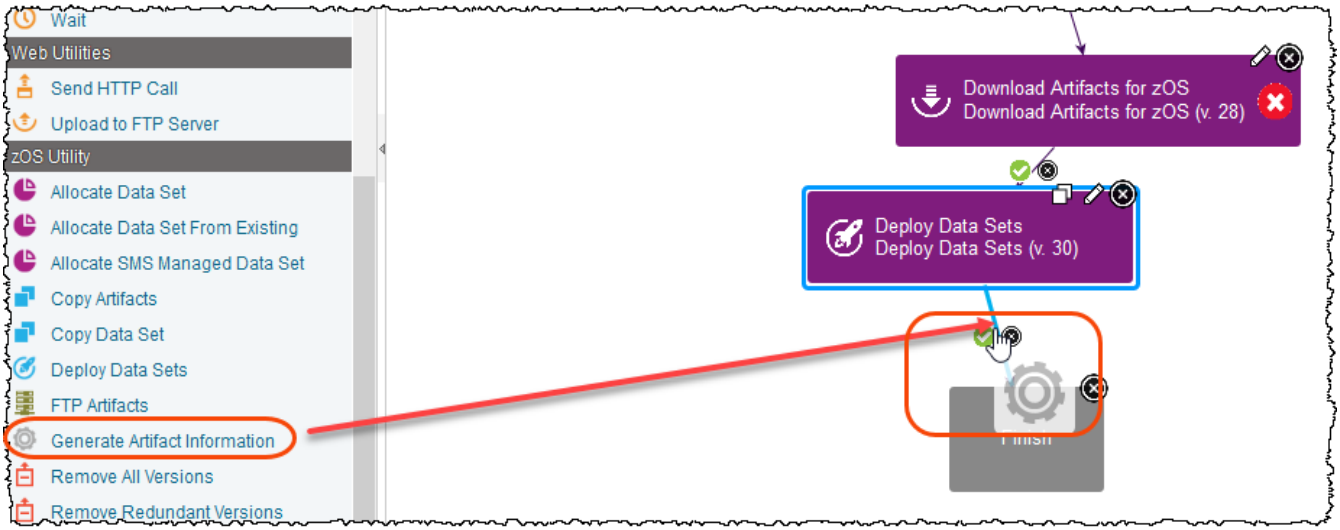
8.13 Click the **Save** button to save what you had done so far.



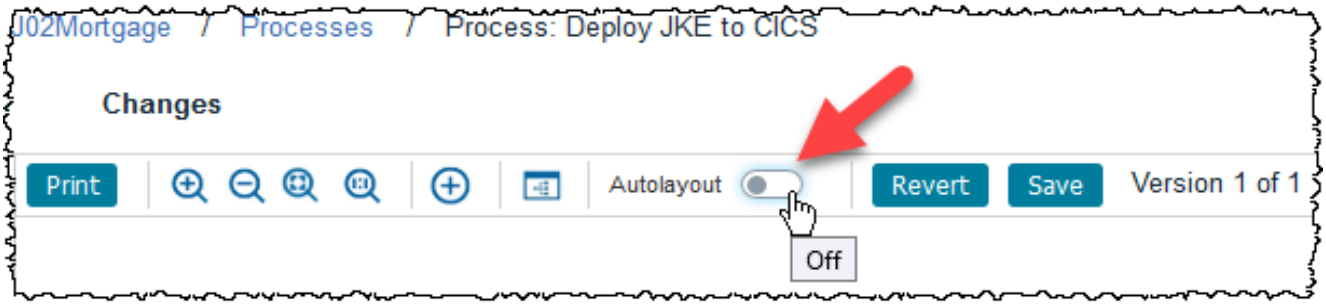
8.14 If there is a warning about missing property ignore it, clicking **Close.** You will fix that later.



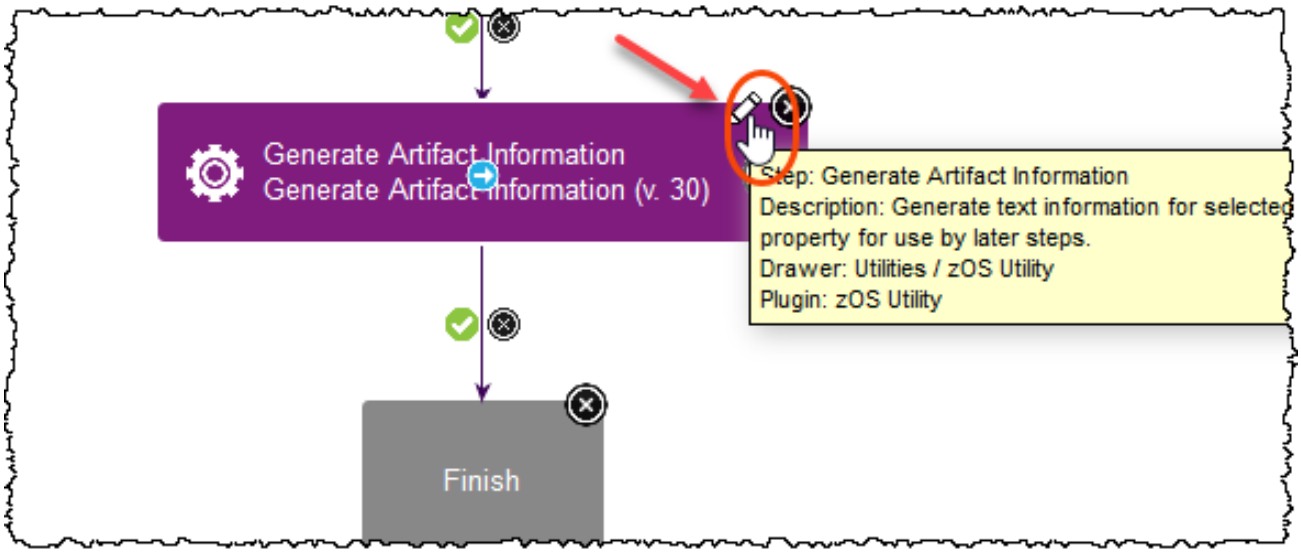
8.15 Scroll down on left pallet and drag and drop **Generate Artifact Information** on the **line** before Finish. You will need to specify a list of the load modules that CICS needs to do the *Newcopy*.
This step generate text information for selected version artifacts. Information is put to output property *'text'* to be consumed by a later step



8.16 Click on **Autolayout** to have the layout organized



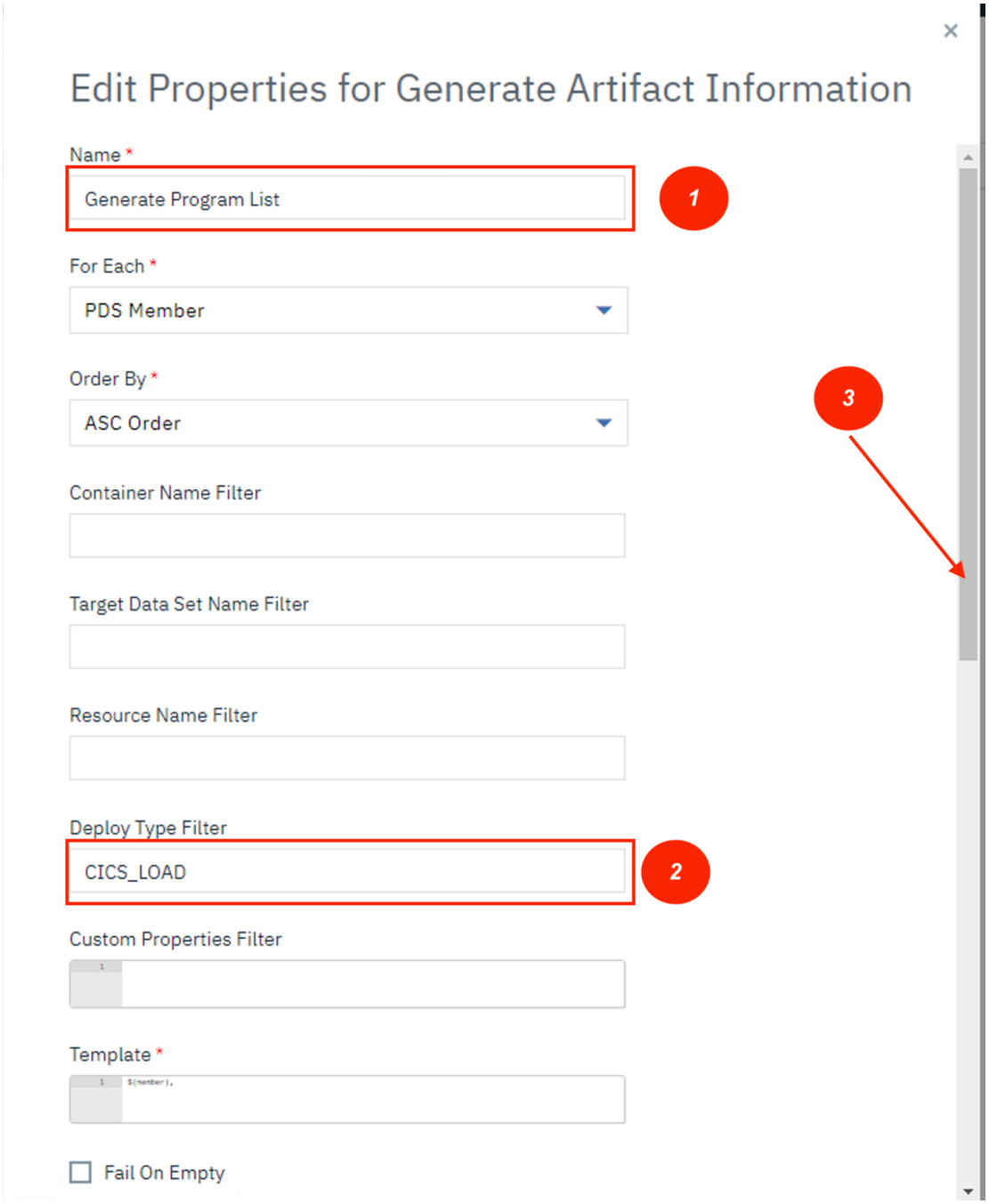
8.17 Click on the **pencil** icon on **Generate Artifact Information** to change the UCD properties for this step.



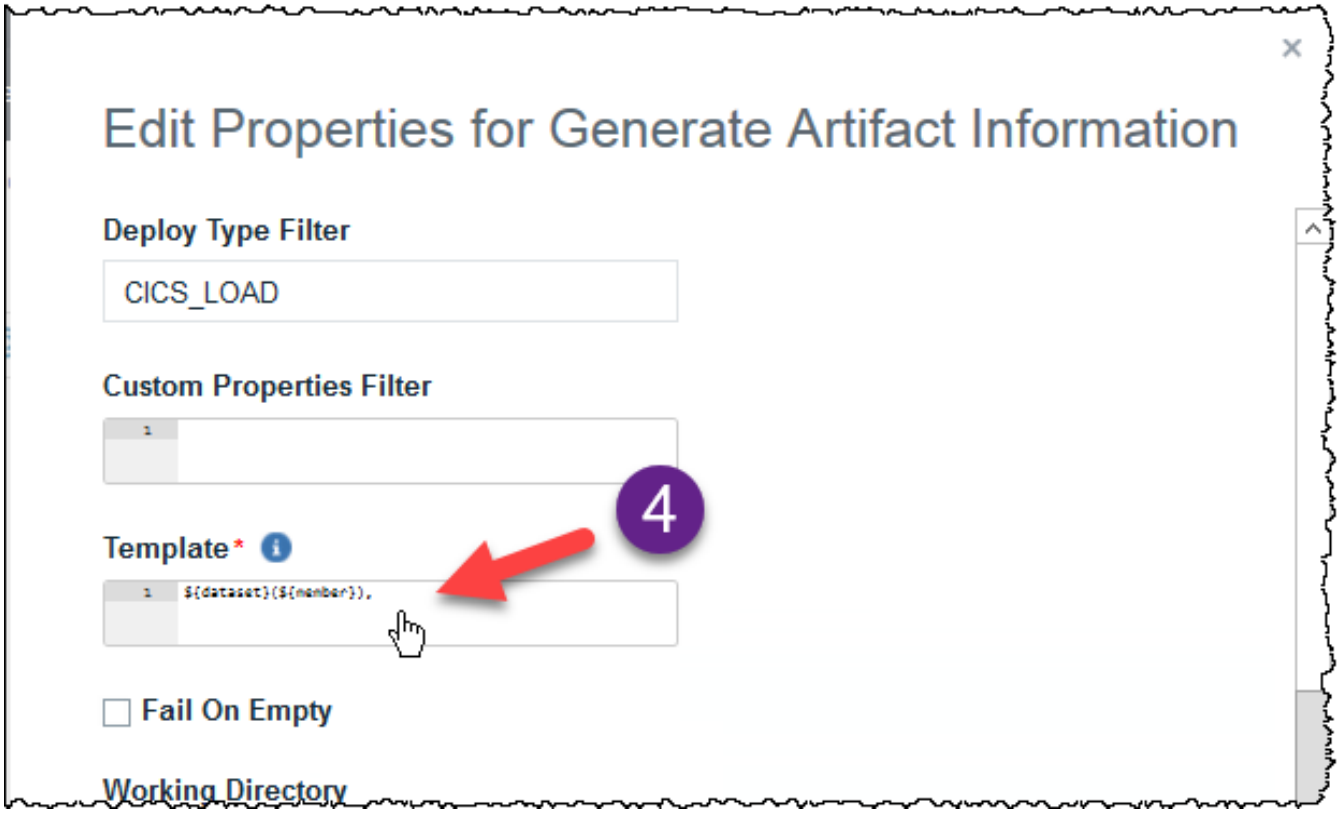
8.18 1 Change from *Generate Artifact Information* to **Generate Program List** (must respect upper/lower case and spaces)

2 Add **`CICS_LOAD`** to *Deploy Type Filter*

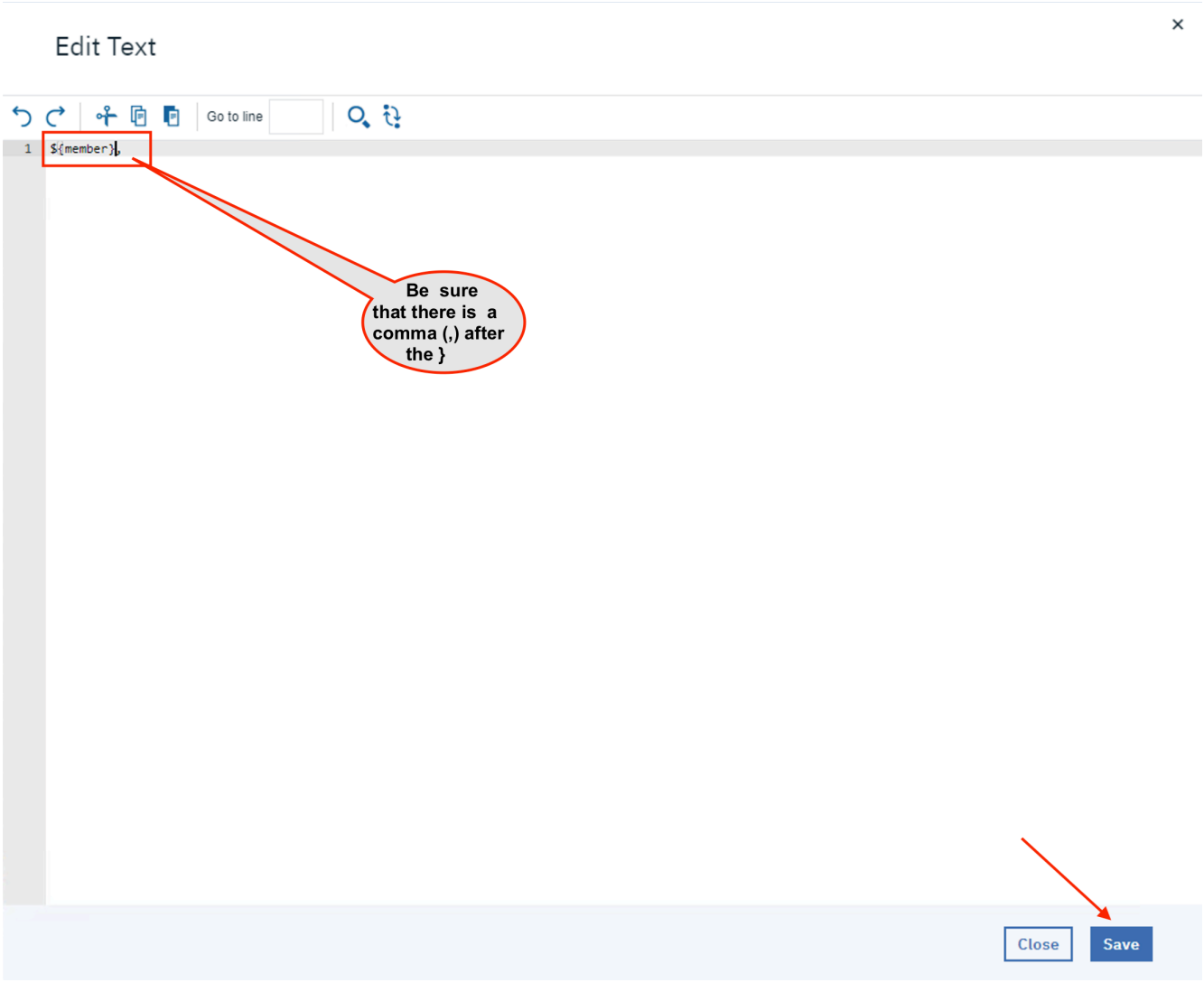
3 Scroll down to add another property


Edit Properties for Generate Artifact Information
Name *
Generate Program List
1
For Each *
PDS Member
Order By *
ASC Order
3
Container Name Filter
Target Data Set Name Filter
Resource Name Filter
Deploy Type Filter
CICS_LOAD
2
Custom Properties Filter
Template *
Fail On Empty

8.19 ⏭ 4 Click on **Template**

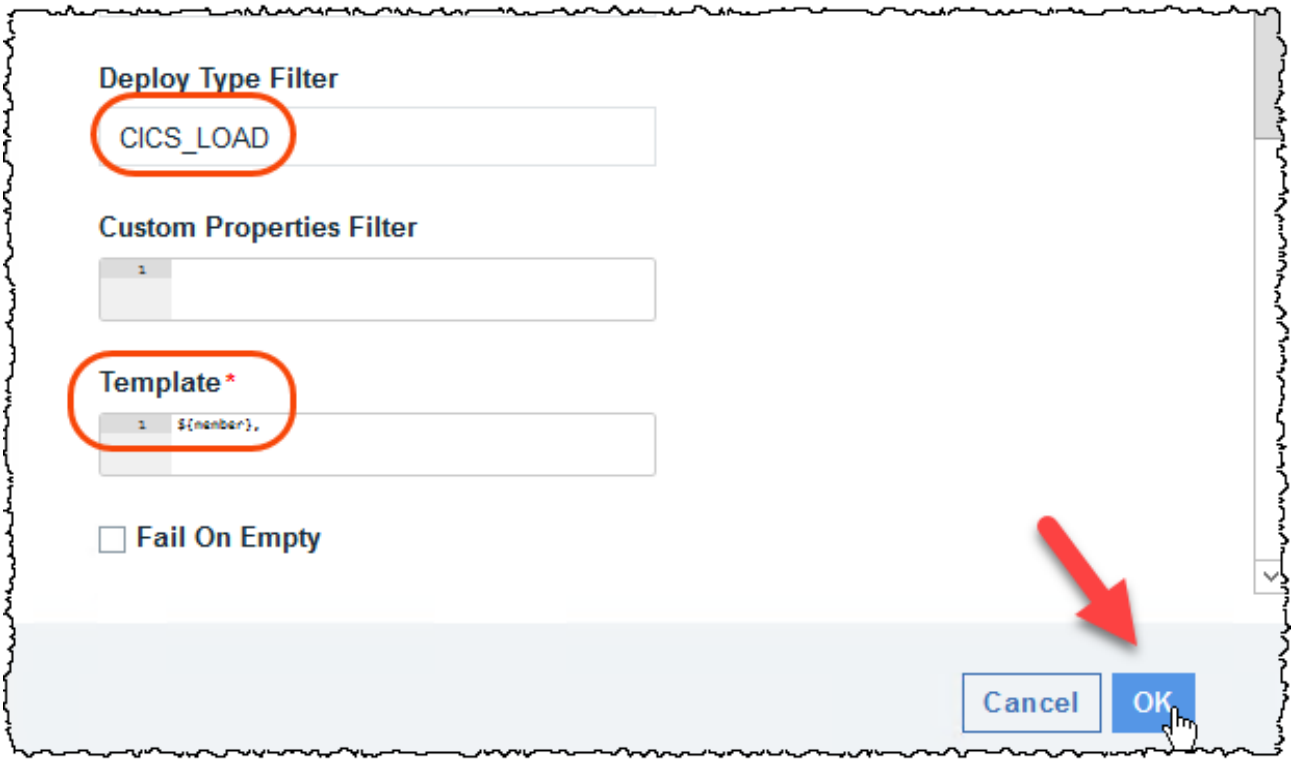


⏭ Change the value from "
`${dataset}(${member}),"` to **${member},**
IMPORTANT ➔ Be sure that you add a **, (comma)** after the **${member},**

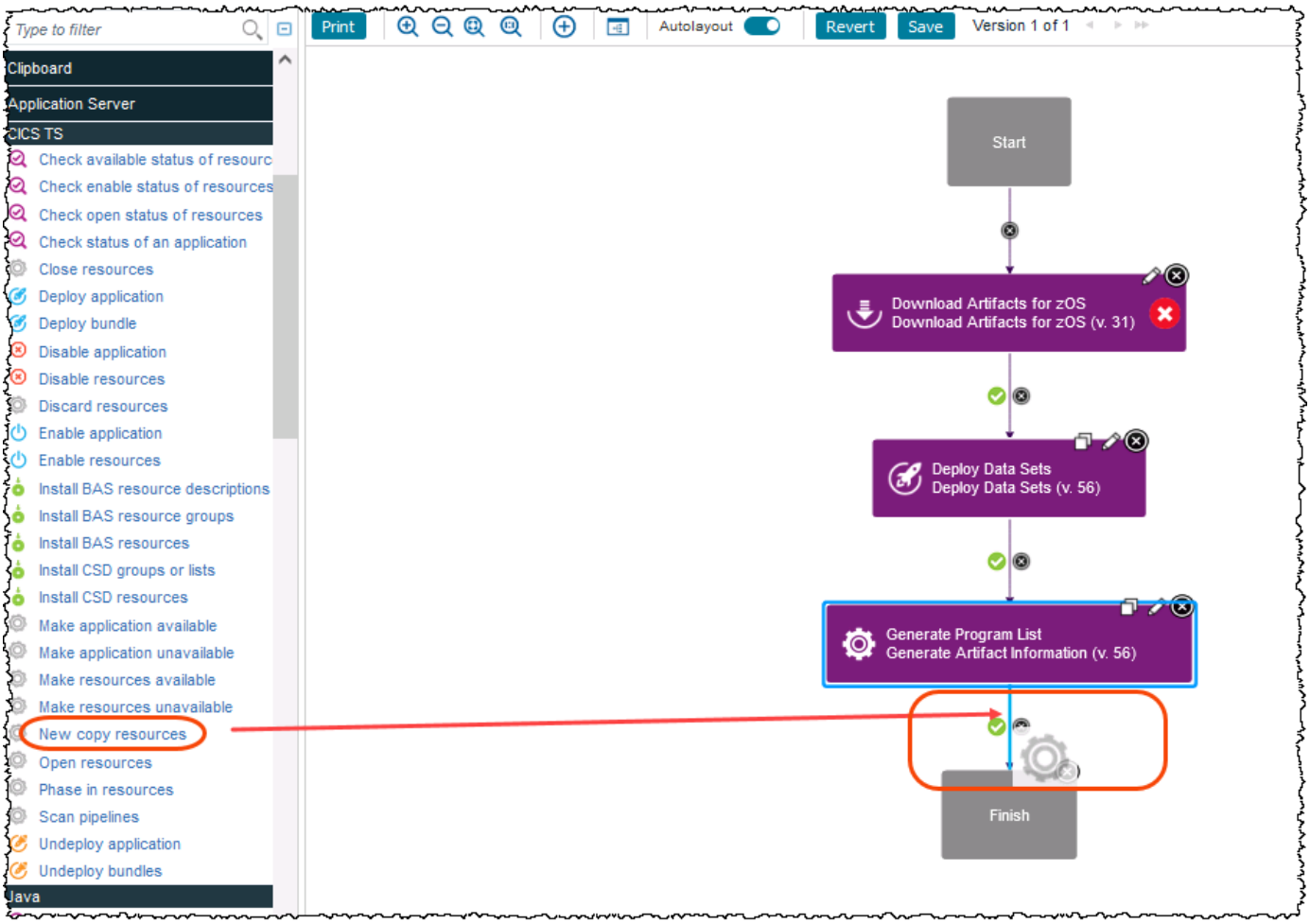


Click **Save** to save the modified value

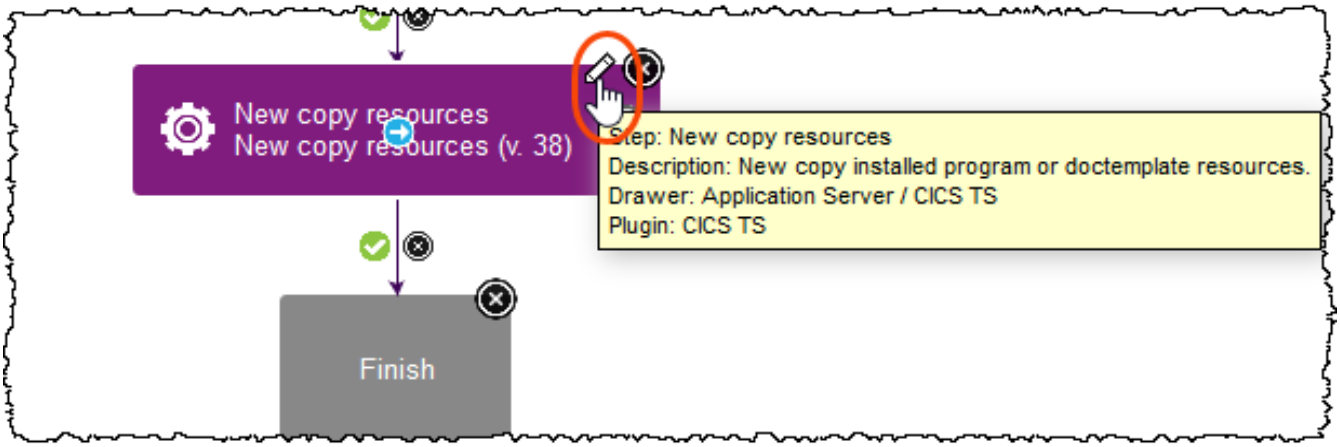
8.20 Click **OK** to close the dialog


Deploy Type Filter
CICS_LOAD
Custom Properties Filter
Template*
${member},
Fail On Empty
Cancel
OK

8.21 On the Palette (left side) expand **Application Server** (scroll up) and **CICS TS.**
Drag and drop **New copy resources** to the **line** that connects to the *Finish* box**..**

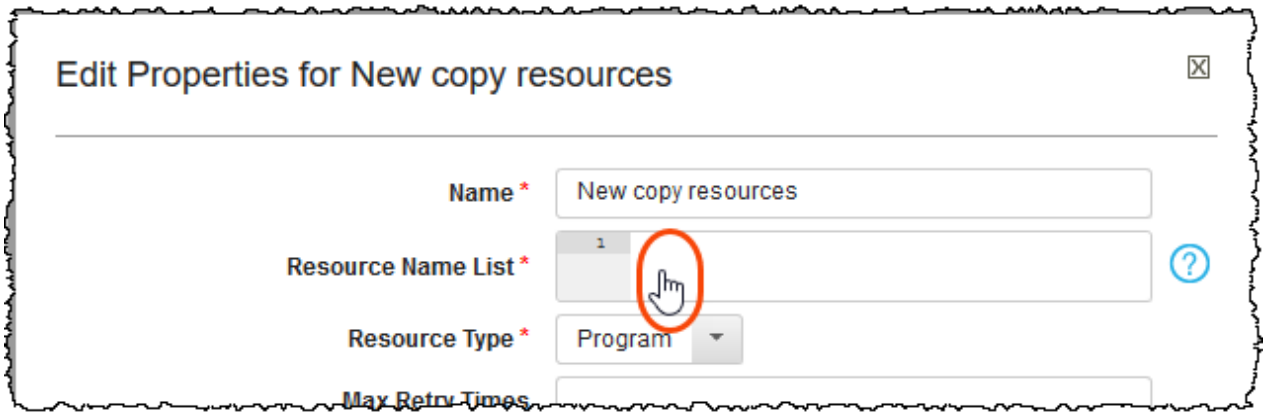


8.22 Click on the **pencil** icon on **New copy resources** to change the UCD properties for this step.

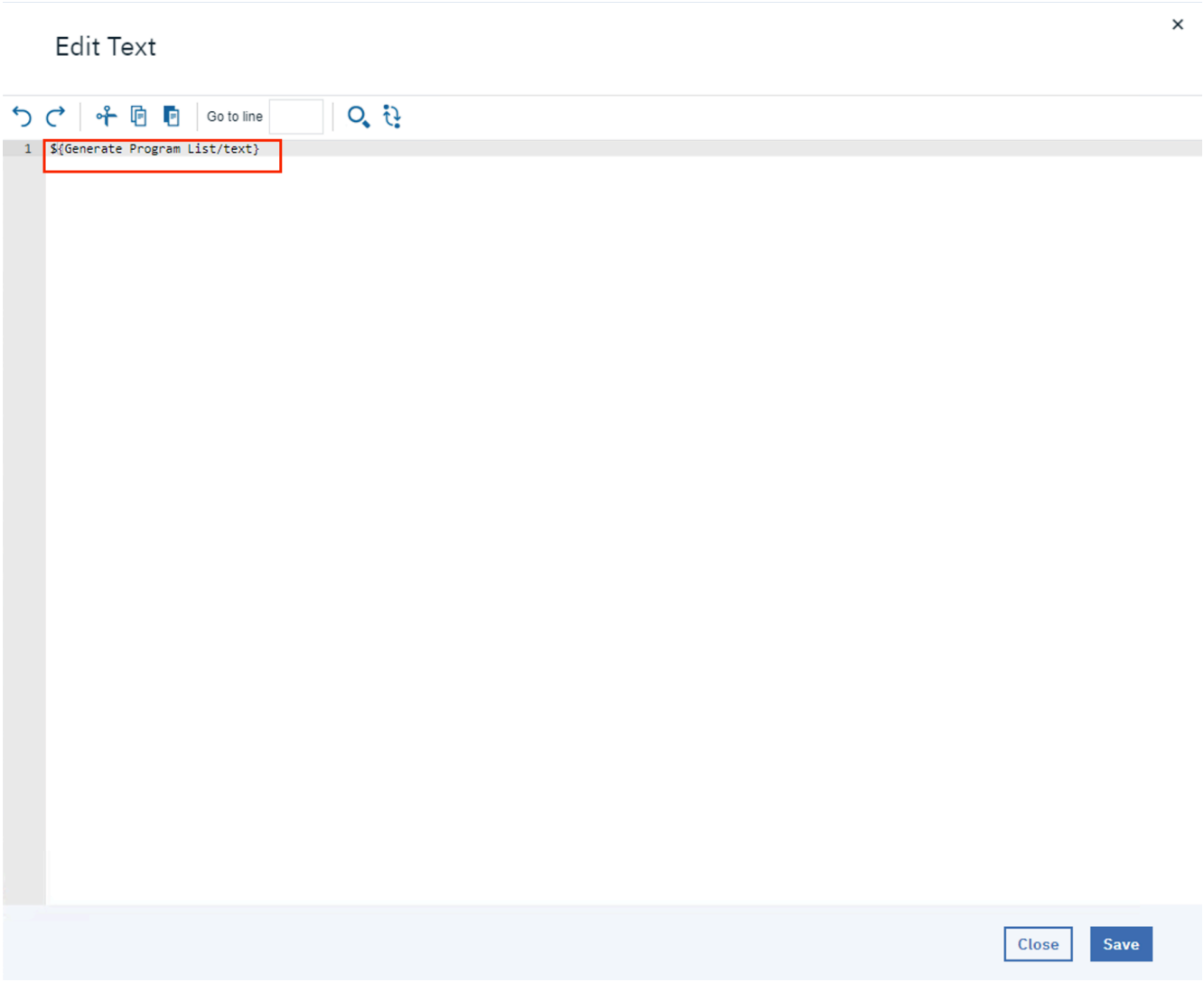


8.23 You need to specify a list of the load modules that CICS needs to do the *Newcopy*.

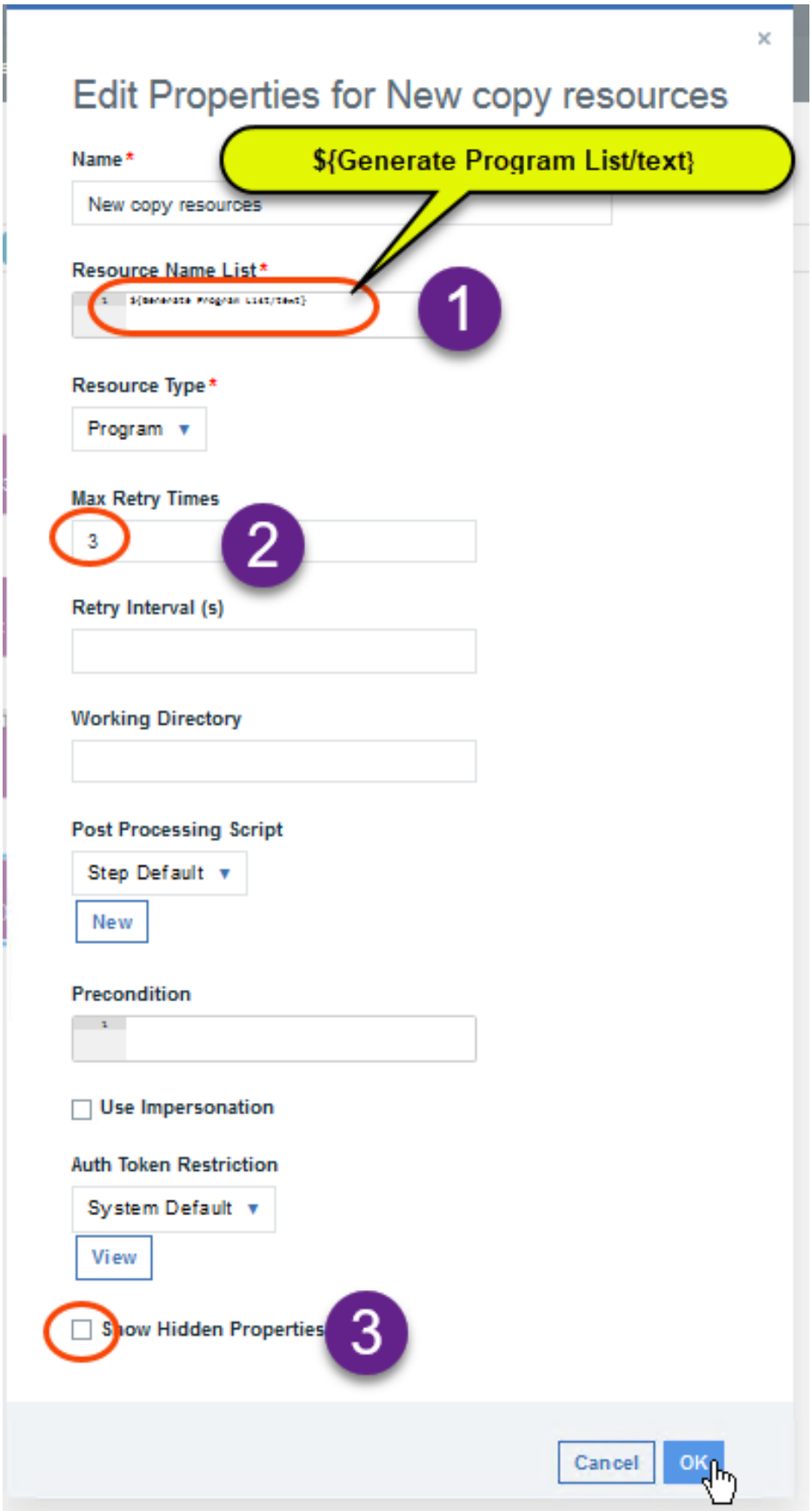
⏭ Click on **Resource Name List**



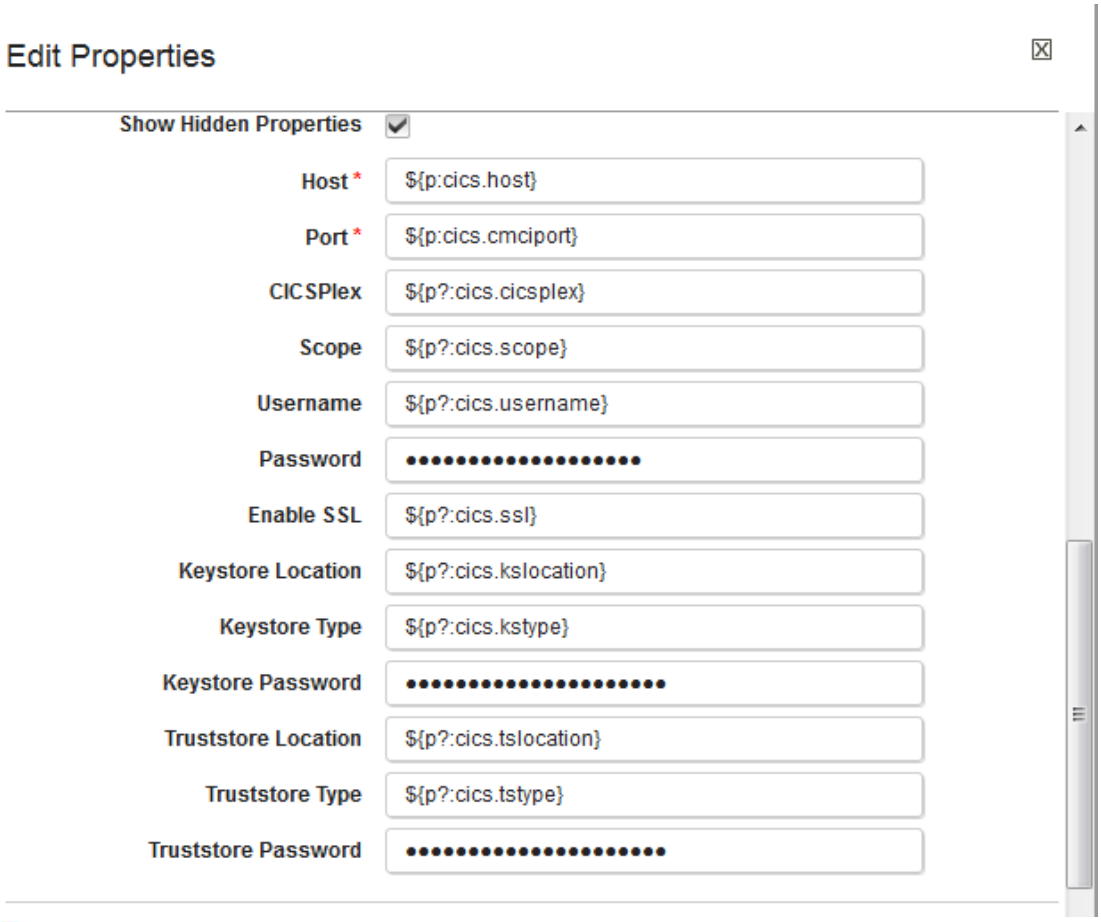
⏭ Type **${Generate Program List/text}** (This is the step name that you created on 8.18) and click **Save**



⏭ Type **3 for Max Retry Times**
⏭ Scroll down and click **Show Hidden properties**.


Edit Properties for New copy resources
Name*
New copy resources
${Generate Program List/text}
Resource Name List*
1
Resource Type*
Program
Max Retry Times
3
2
Retry Interval (s)
Working Directory
Post Processing Script
Step Default
New
Precondition
Use Impersonation
Auth Token Restriction
System Default
View
Show Hidden Properties
3
Cancel
OK

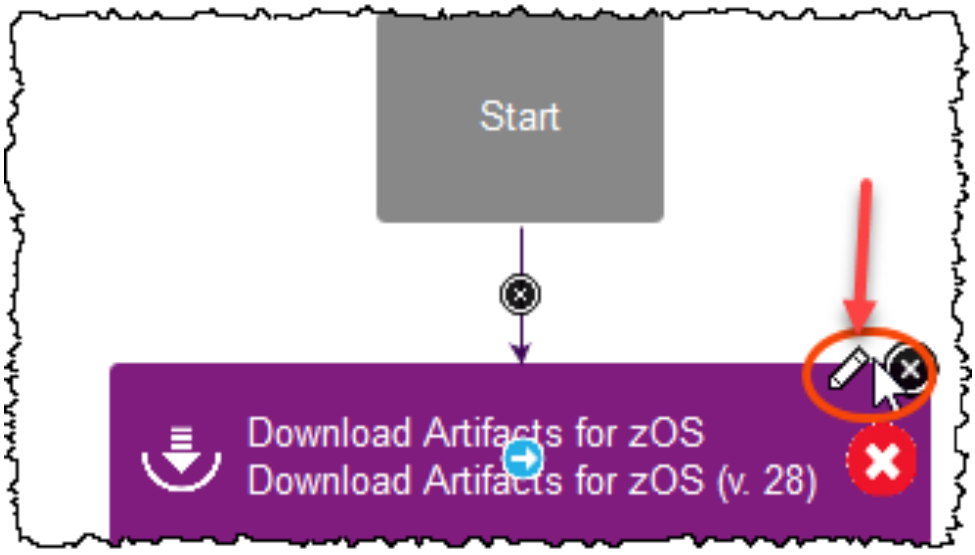
8.24 Scroll down to see those properties that need to be filled. Later we will do that at the **Test** level



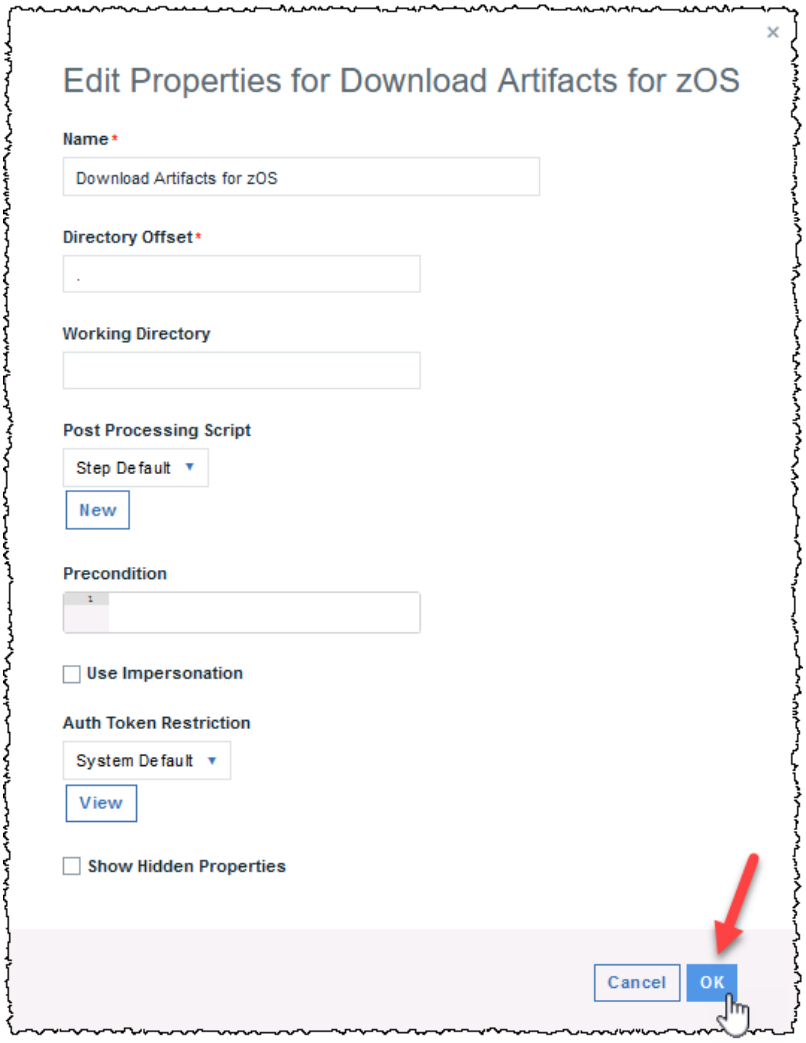
Click **OK** to save it

8.25 You still need to fix a warning on first step (see the icon).

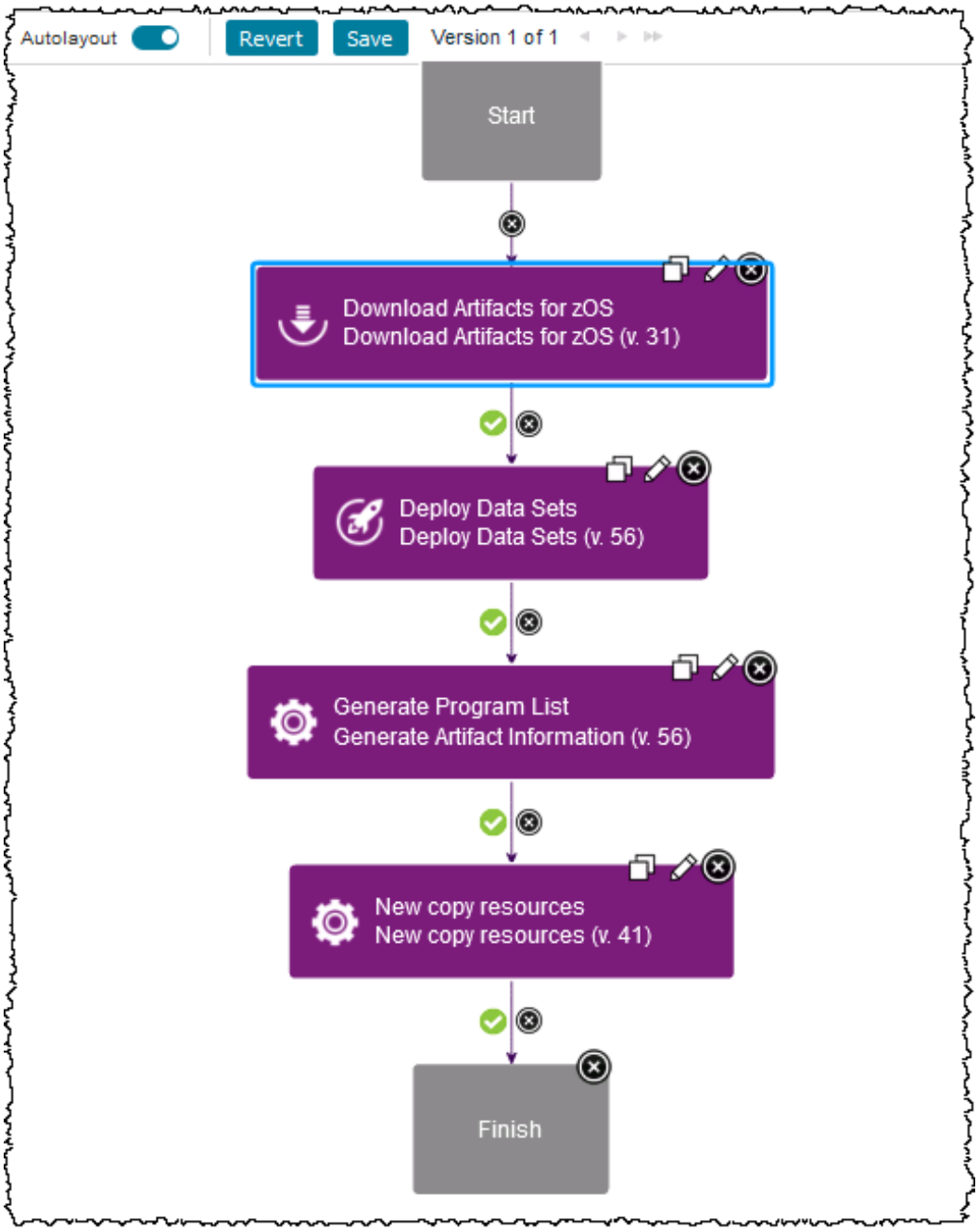
Click on the **pencil** icon on **Download Artifacts for zOS** to possibly change the properties.



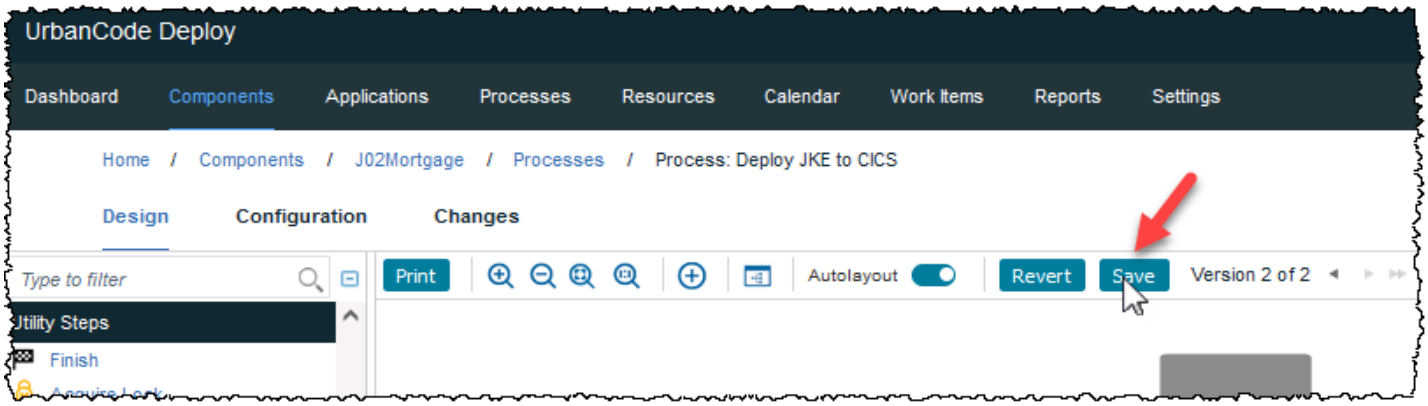
8.26 No changes are required here and click **OK**



8.27 The final result should look similar to this:



8.28 Finally, click the **Save** button located in the upper left portion of the process editor to save the process:



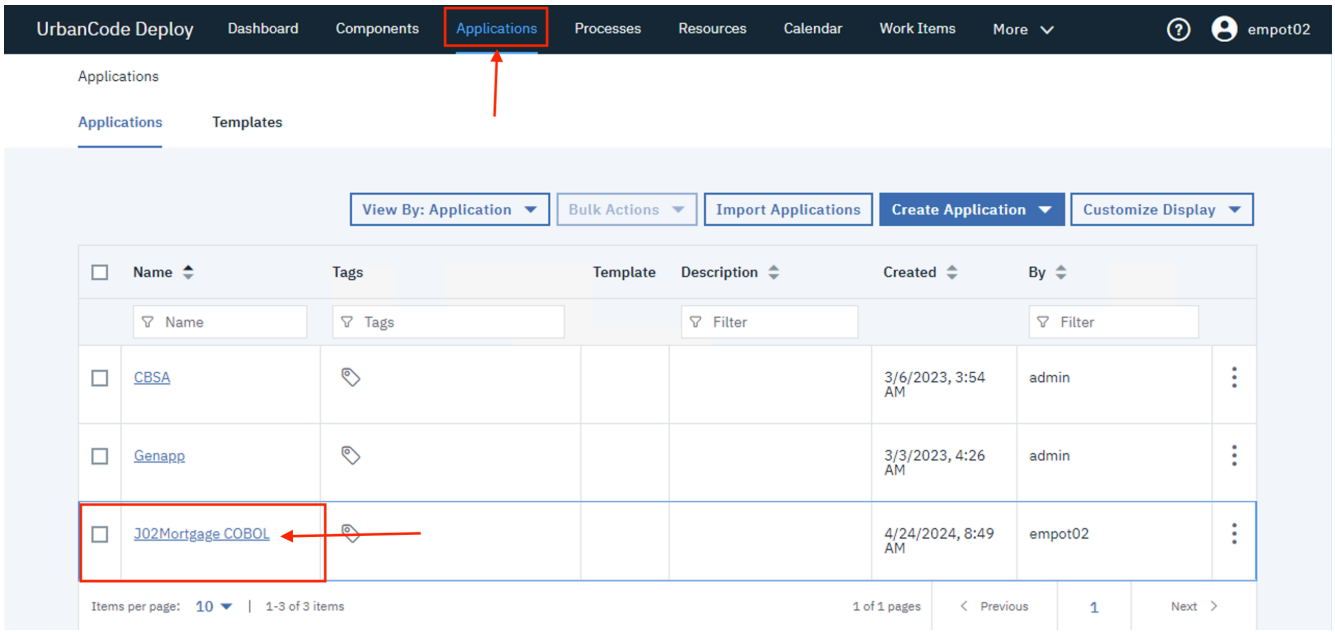
A message will indicate that the process is saved.

Task 9 - Create an application process

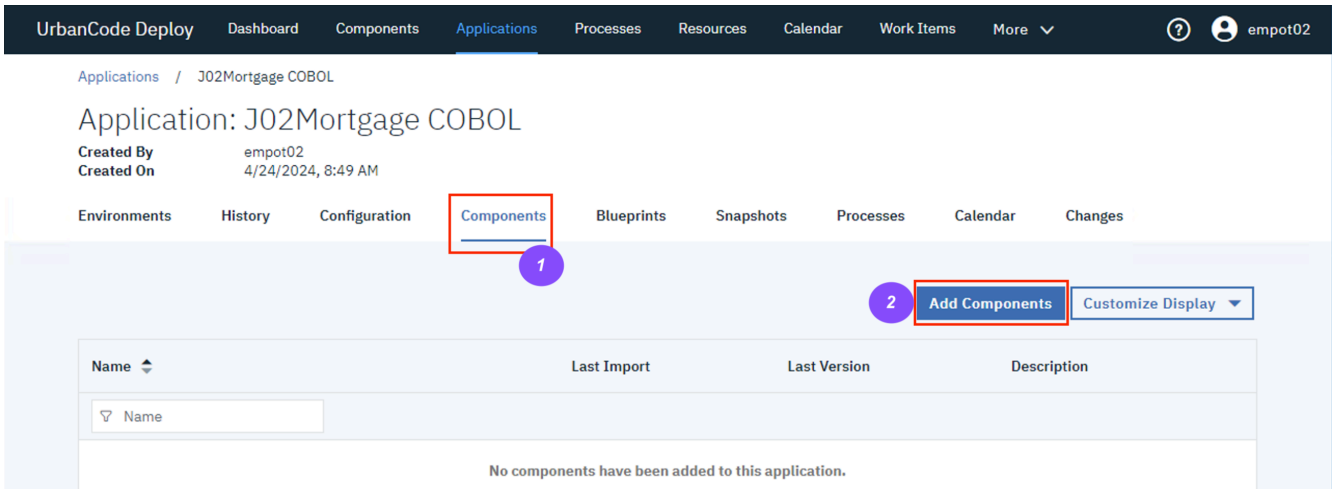
Application processes direct underlying component processes and orchestrate multi-component deployments. An application process, like a component process, consists of steps that are configured with the process editor.

In this task, you create an application process that installs the UCD application defined .

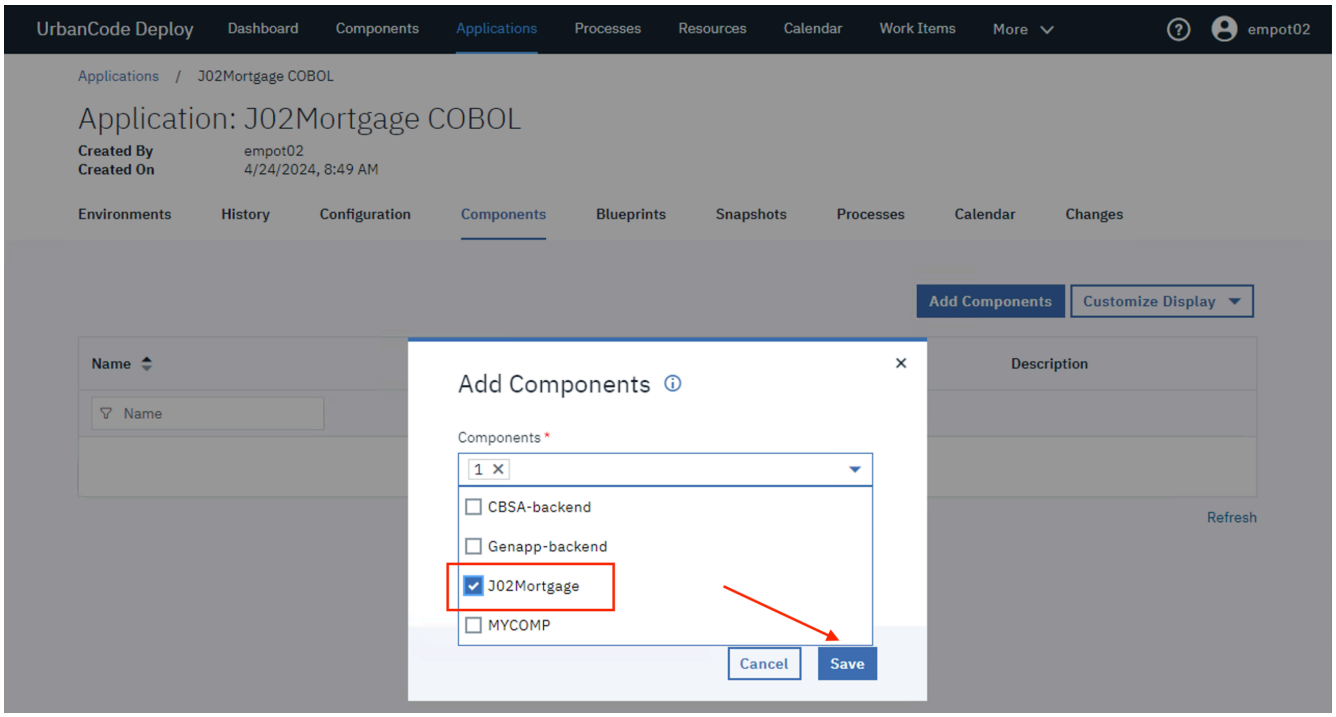
9.1 ⏭ Go to **Applications** page, select your application (**J02 Mortgage - COBOL**),



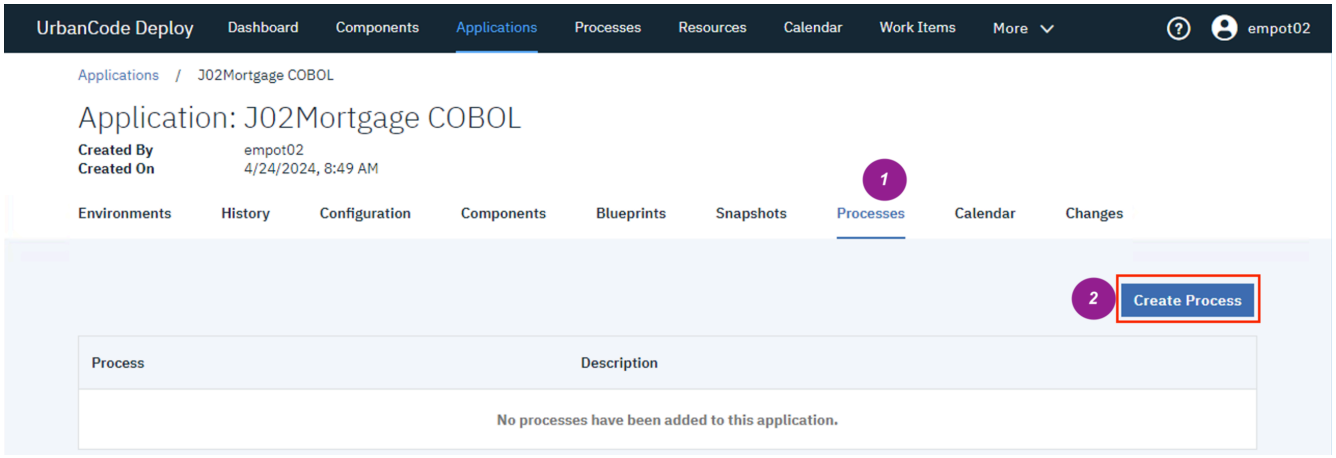
9.2 ⏭ Click **Components** tab and click **Add Component**:



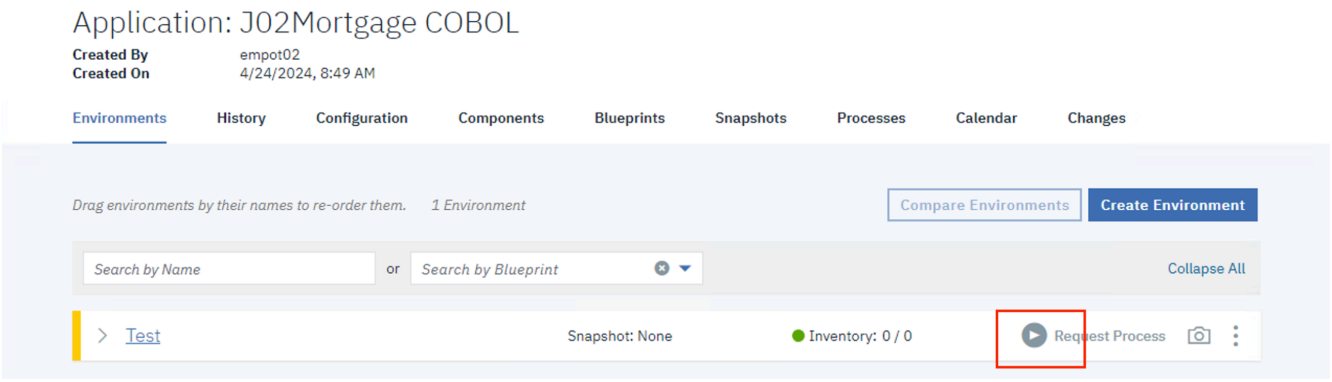
9.3 ⏭ Select your UCD component created previously (**J02Mortgage**) and click **Save**:



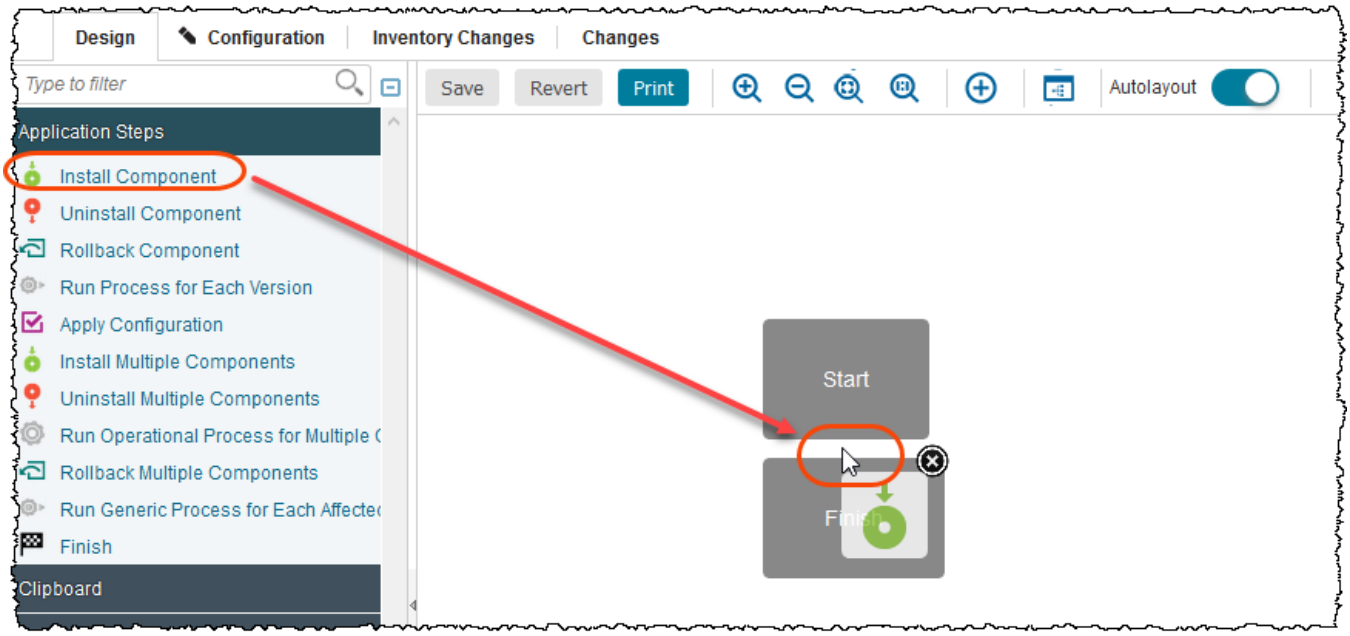
9.4 ⏭ Click the **Processes** tab and click **Create Process**:



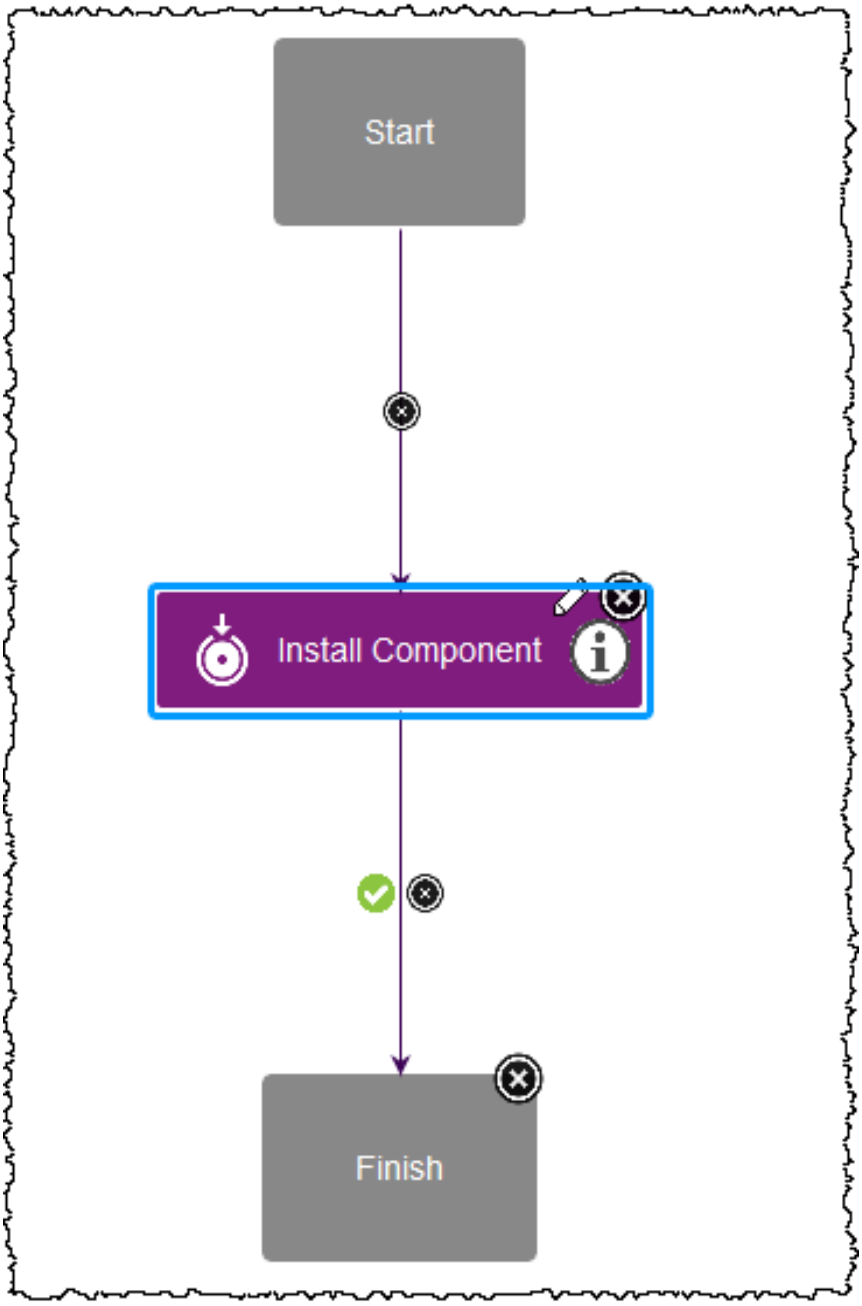
9.5 ⏭ Give a name to your process, like **Deploy J02 to CICS** and click **Save**:



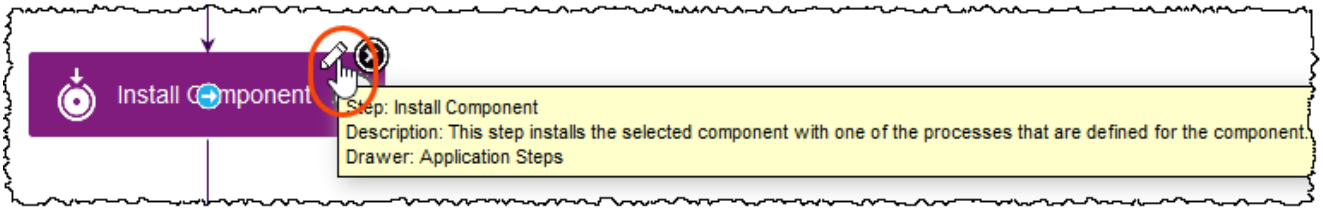
9.6 Find **Install Component…** step in the Design Palette, and drag it onto the space between *Start* and *Finish* boxes.



9.7 The result will be as below:

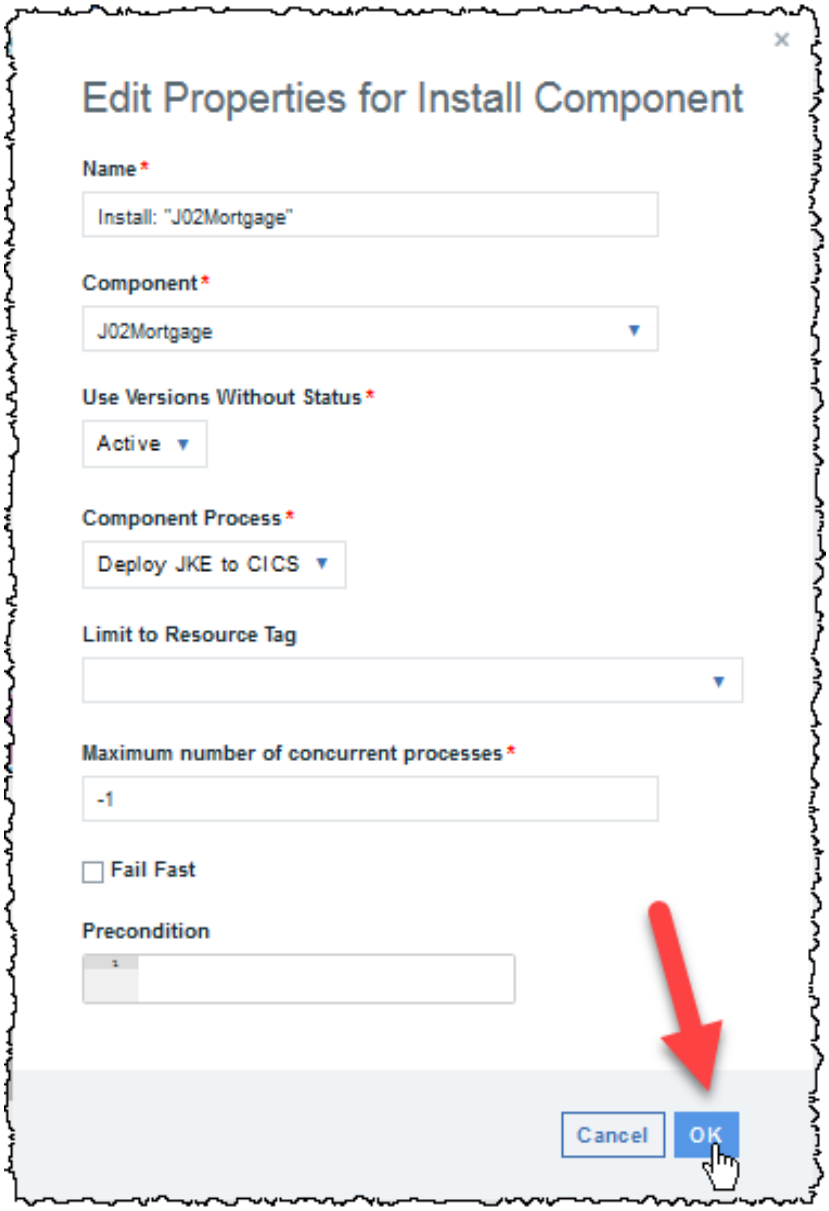


9.8 Click on the **pencil** icon on **Install Component** to possibly change the properties

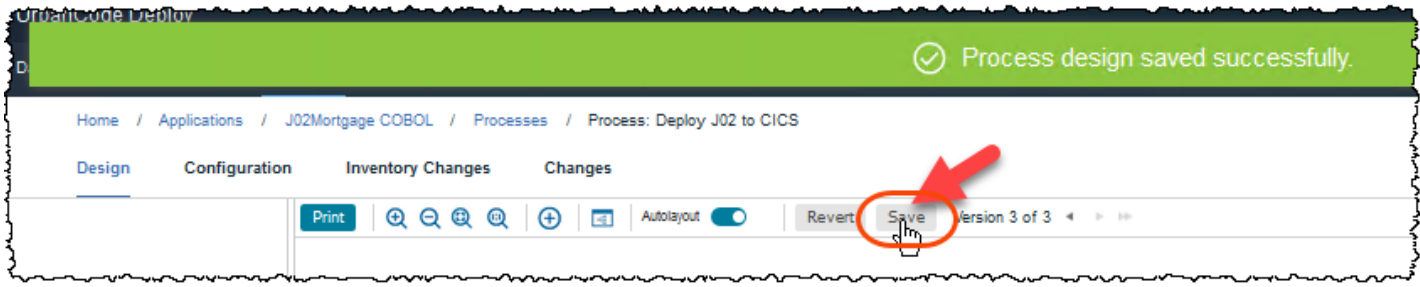


9.9 Notice that the properties of this step are pre filled for you.

⏭ Click **OK**:



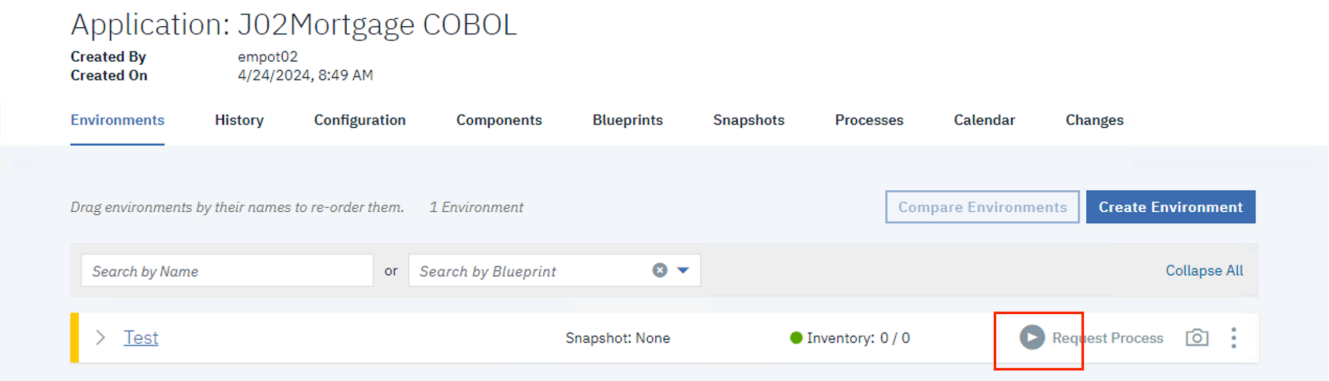
9.10 ⏭ Click **Save**



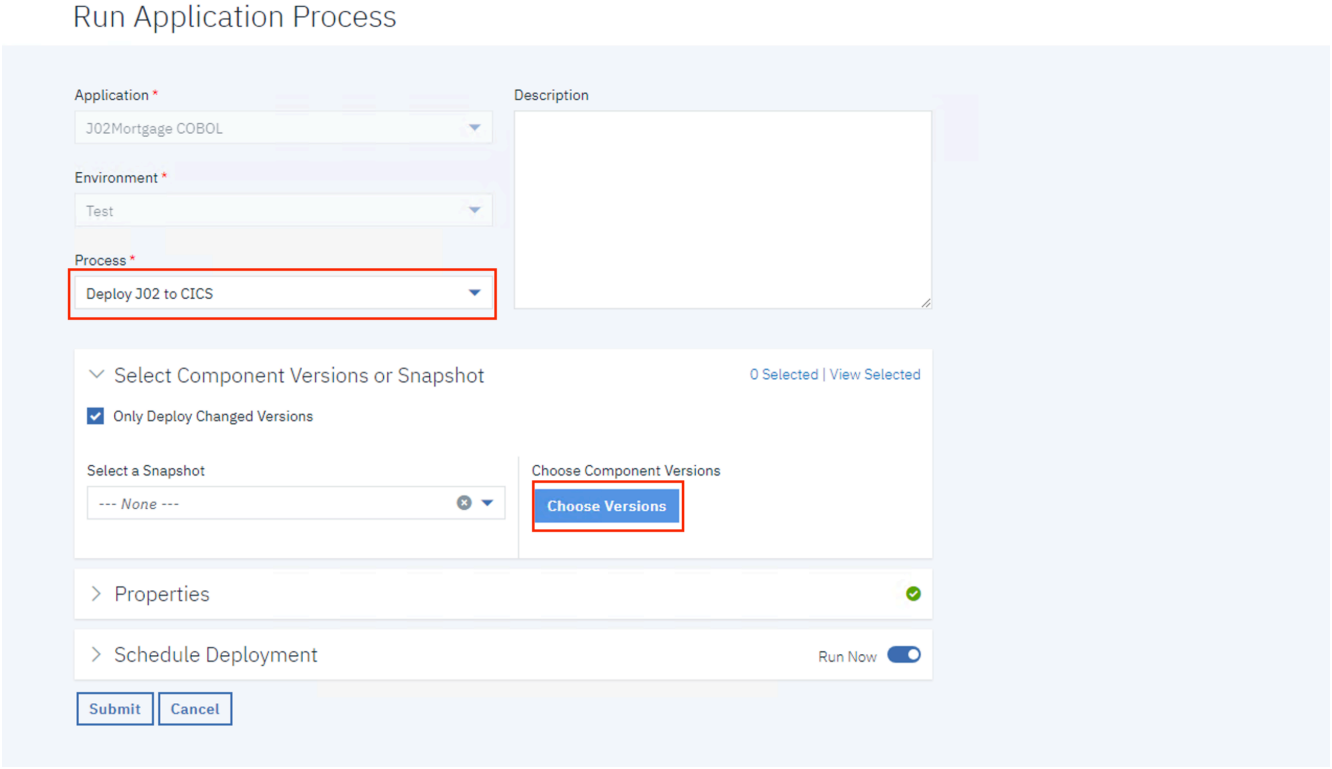
Task 10 – Environment properties configuration

On the task 7 you created the **Test** environment. For this lab we have just one environment, but usually we have few environments. Example the **Test** environment could reflect the Test LPAR and **Prod** environment that would reflect other LPAR. Each environment may have different values used on the deploy. Example the CICS test environment uses port 30090, the prod environment uses port 30091, etc..
Those values in our lab were not yet defined and you will see errors when trying to deploy the application for Test environment..

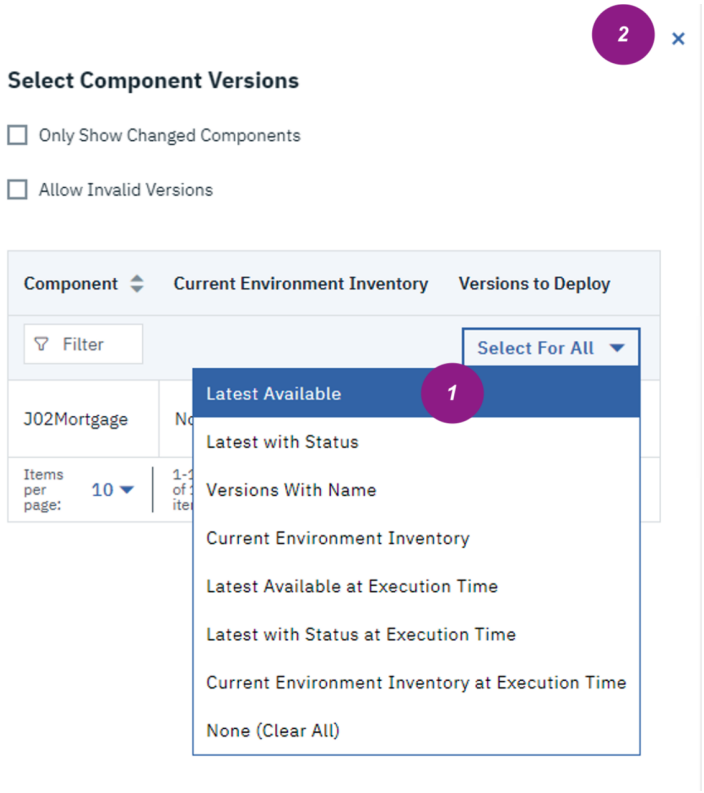
10.1 Go to **Application > J02 Mortgage COBOL** and click the icon on **Test** environmnet



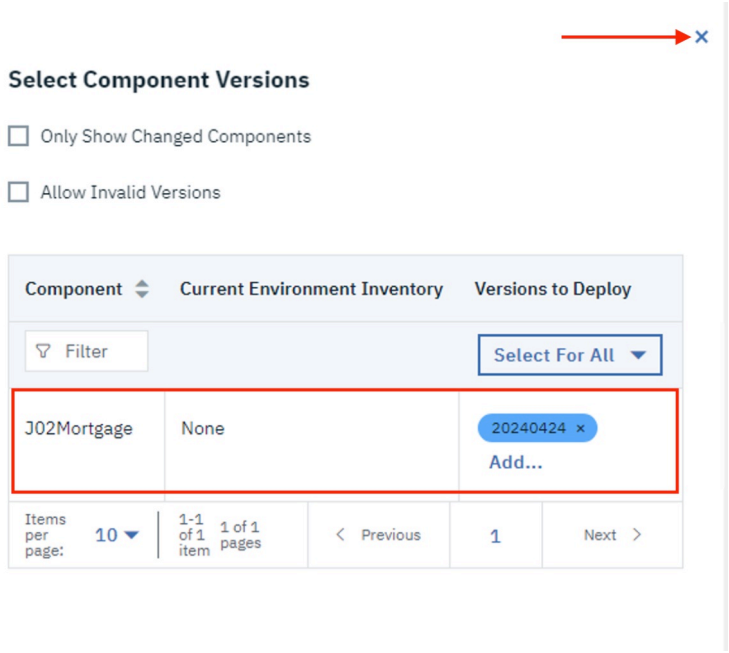
10.2 Select the Process **Deploy J02 to CICS** , click **Choose Versions,**



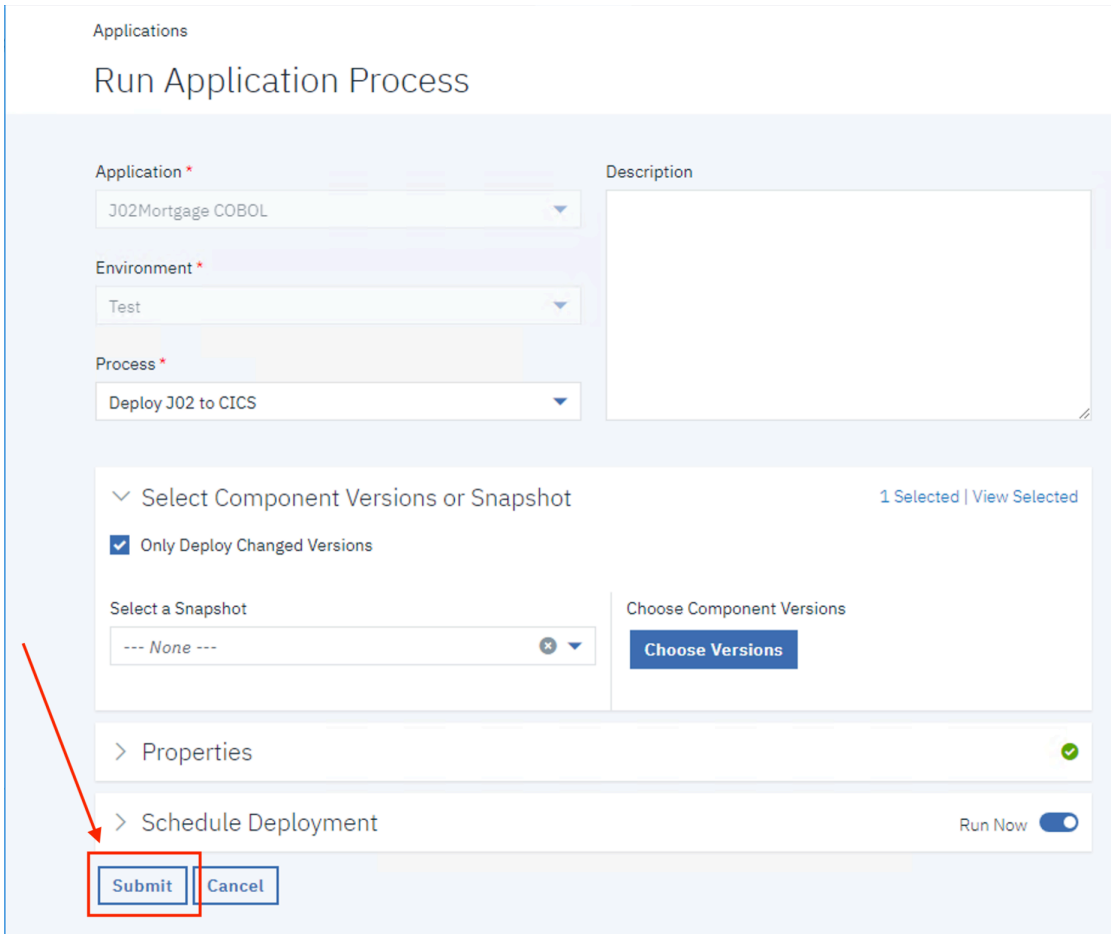
click **Select For All..** and select **Latest Available. Close window.**



You should see: and close window.

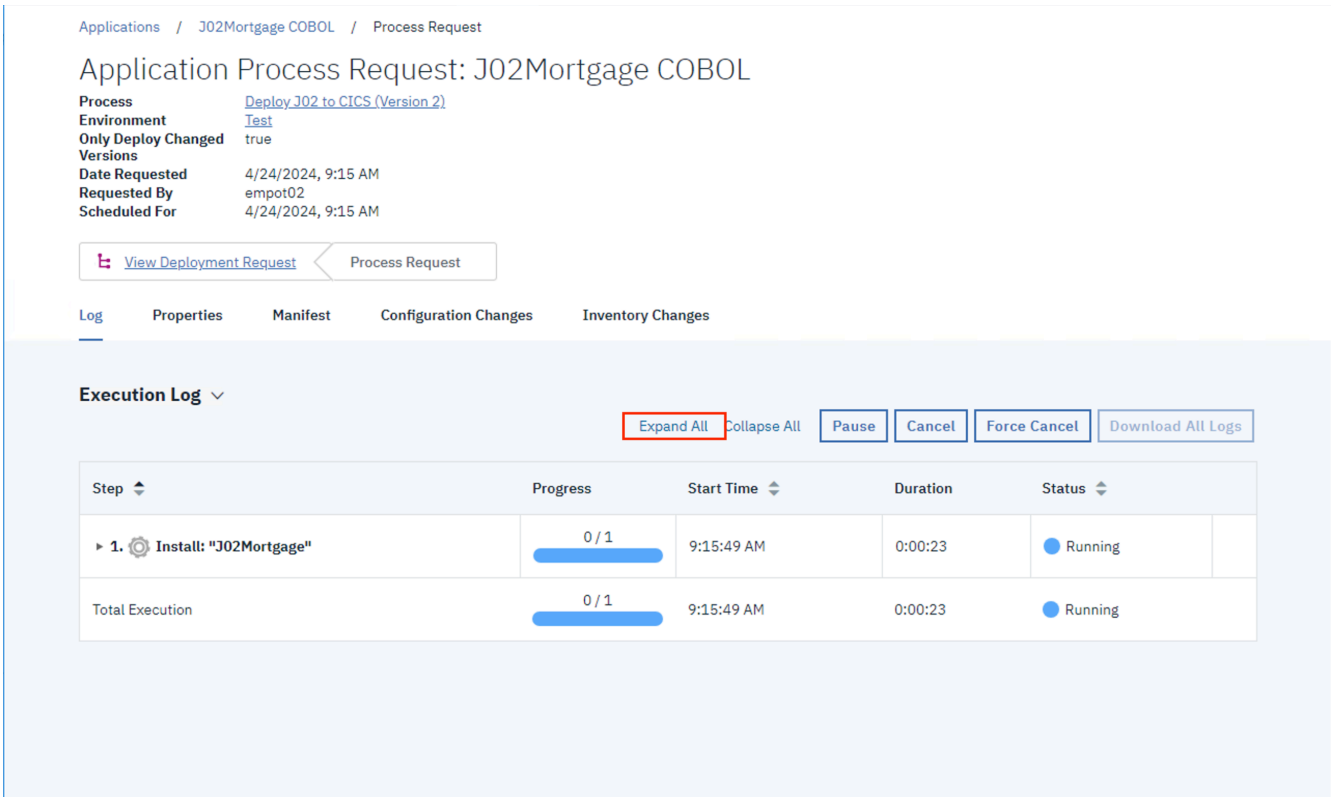


10.3 When the dialog below is shown click **Submit**



The Deploy process start, but it will fail as you see soon..

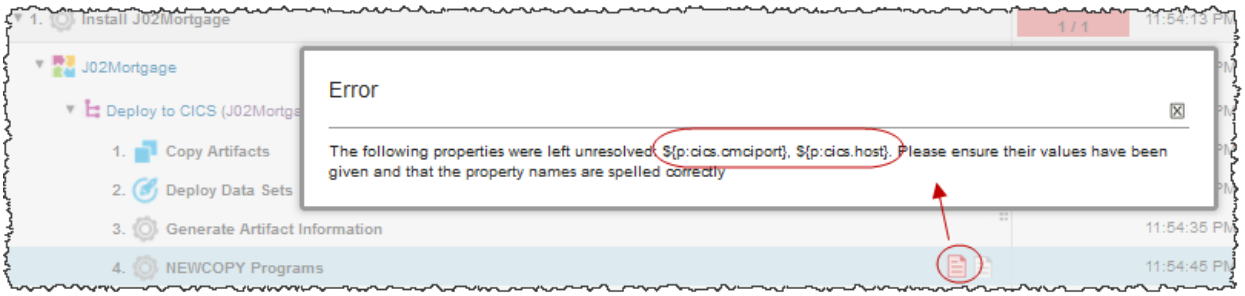
10.4 Click **Expand all** on the far right


Applications / J02Mortgage COBOL / Process Request
Application Process Request: J02Mortgage COBOL
Process Deploy J02 to CICS (Version 2)
Environment Test
Only Deploy Changed Versions true
Date Requested 4/24/2024, 9:15 AM
Requested By empot02
Scheduled For 4/24/2024, 9:15 AM
View Deployment Request
Process Request
Log
Properties
Manifest
Configuration Changes
Inventory Changes
Execution Log
Expand All
Collapse All
Pause
Cancel
Force Cancel
Download All Logs
Step
Progress
Start Time
Duration
Status
1. Install: "J02Mortgage"
0 / 1
9:15:49 AM
0:00:23
Running
Total Execution
0 / 1
9:15:49 AM
0:00:23
Running

10.5 Click icon to expand .. Be patient.. this is slow on your environment.. Three steps were successful but the last one failed.

Click on **Error Log** icon

10.6 See the *properties missing* and **close the dialog**

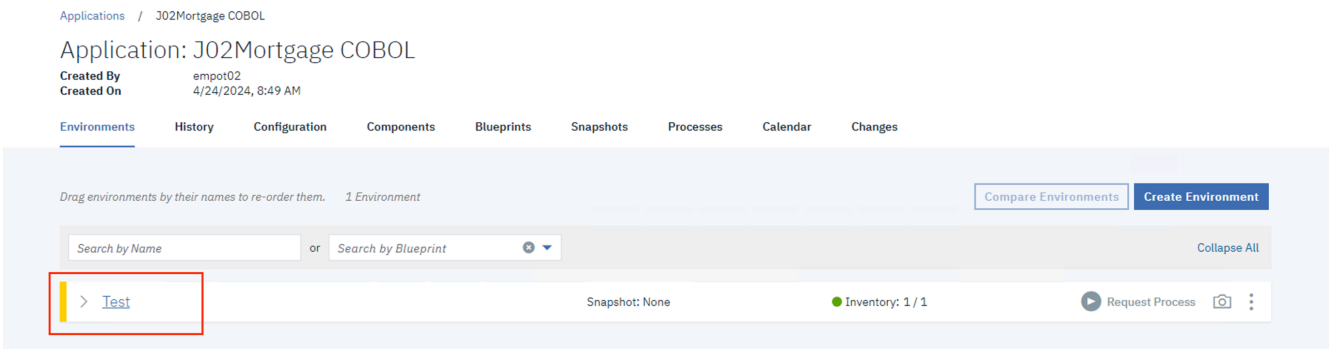


10.7 We could add properties at various levels.. In our example let's make the variables at the Environmet level. The missing properties are:

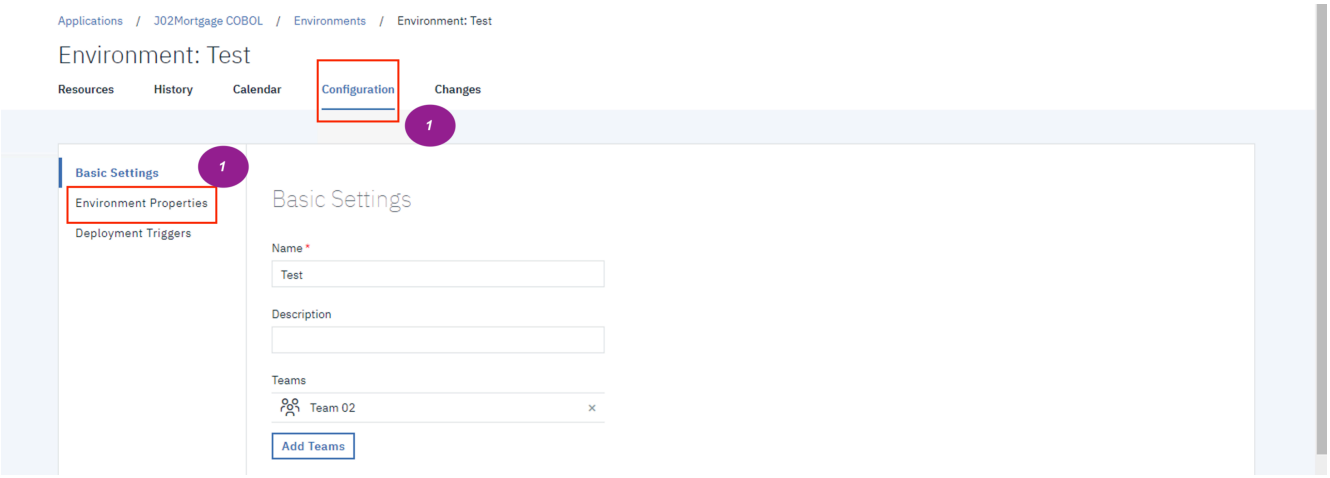
`cics.cmciport` and **`cics.host.`**

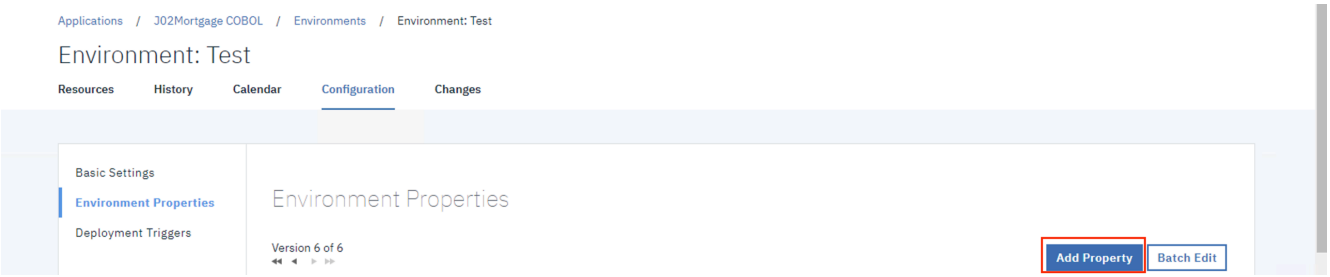
10.8 To add the properties..

Click **Applications > J02 Mortgage COBOL > Test**

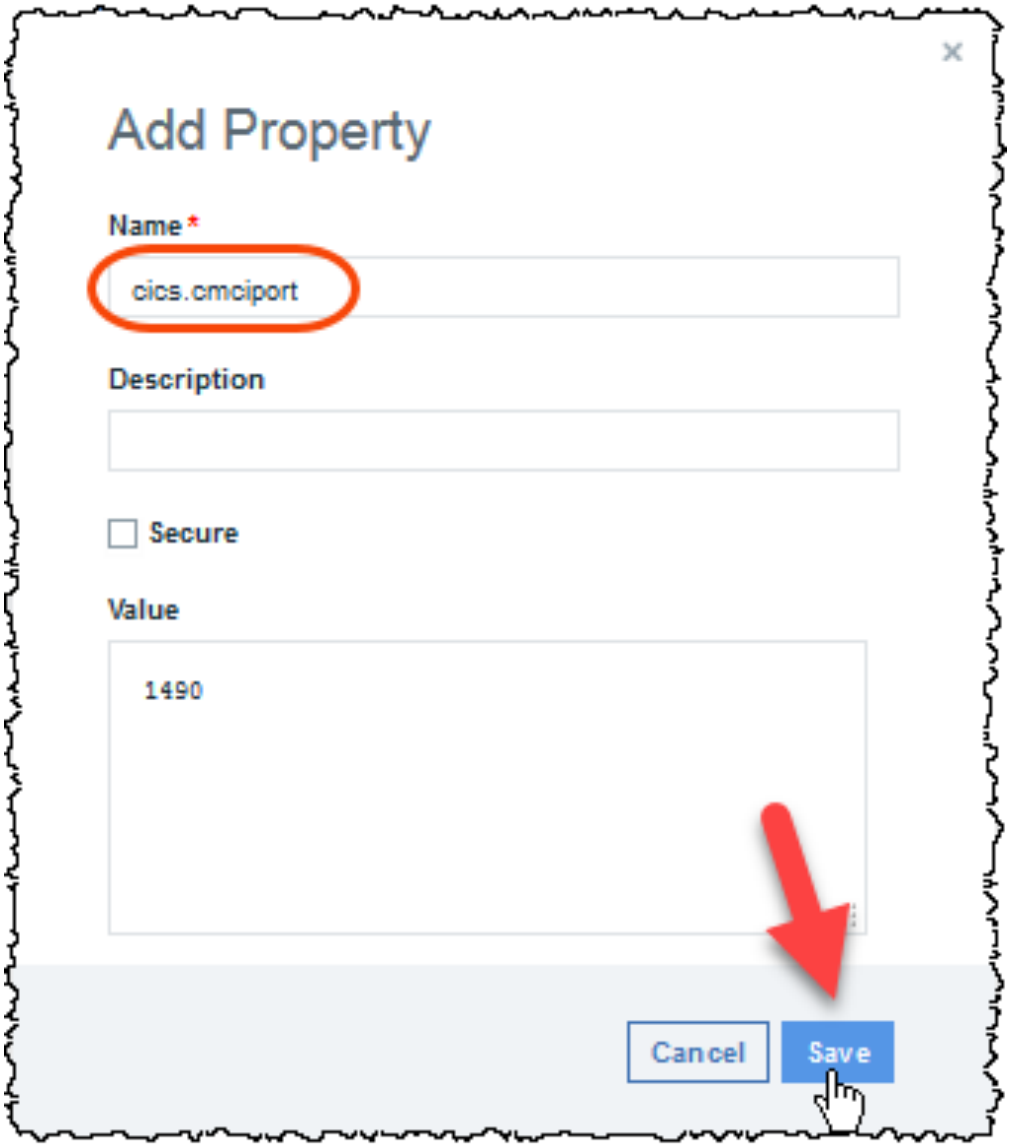


10.9 ⏭ Select **Configuration** > **Environment properties** and click **Add Property**

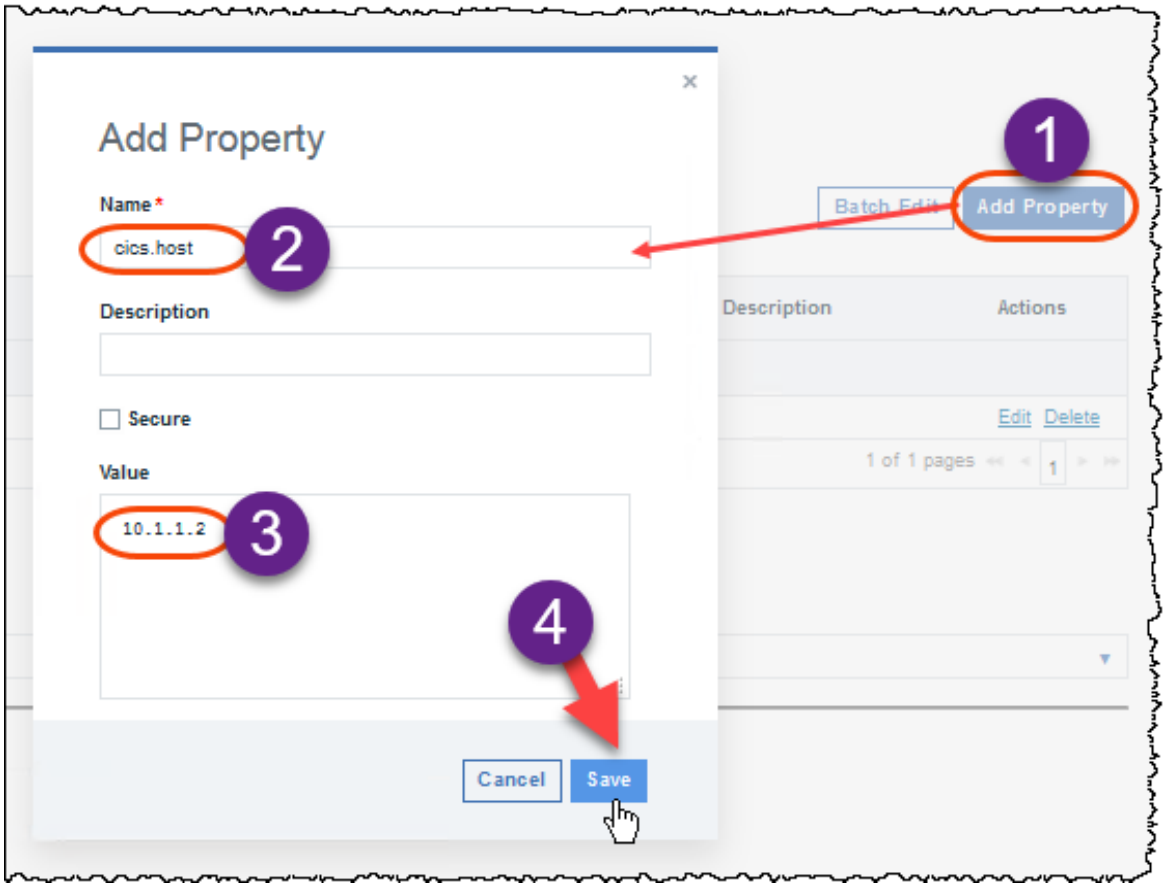




10.10 ⏭ Add the property **`cics.cmciport`** with value **`1490`** and click **Save**

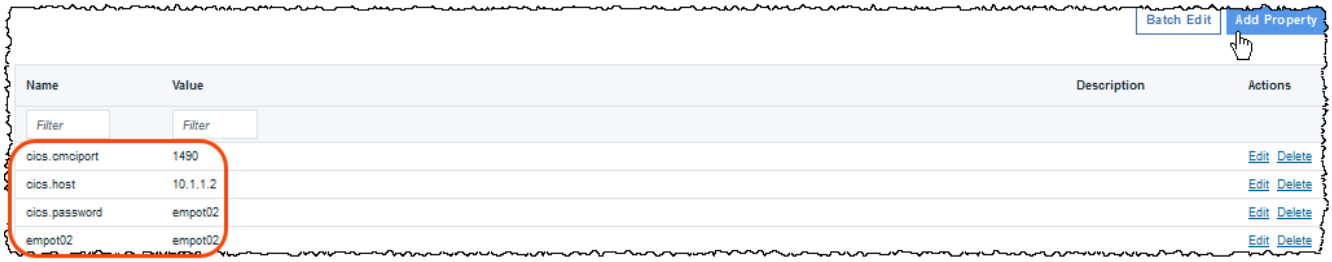


10.12 ⏭ Add the property **`cics.host`** with value **`10.1.1.2`** and click **Save**

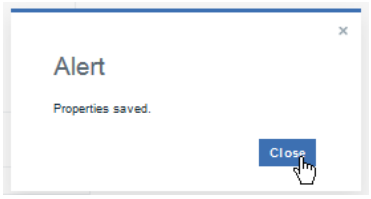


10.13 ⏭ Using the *Add Property* button
add the property **`cics.userid`** with the value **`empot02`** (your userid)

10.14 ⏭ Also add the property **`cics.password`** with the value **`empot02`**.
In the real world you will not add the password here and will use other secure way to authenticate., also In the Edit Property dialog, you can check the secure box and this will hide (secure) the password..
When complete:



10.15 Scroll down and click **Save** to save the defined properties and **Close** to close the dialog.



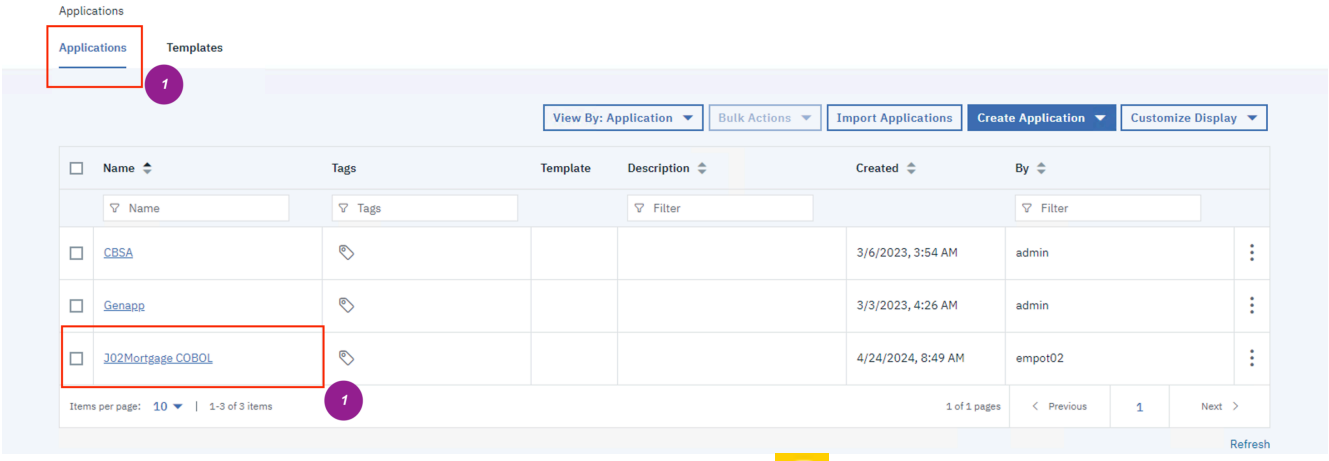
PART 3 – Deploy the application to z/OS CICS

On this part you will deploy the Application to the z/OS CICS.
To deploy the components in the application, you must run the application process on the specified environment.
We created only one environment named **Test**, you will deploy this environment.

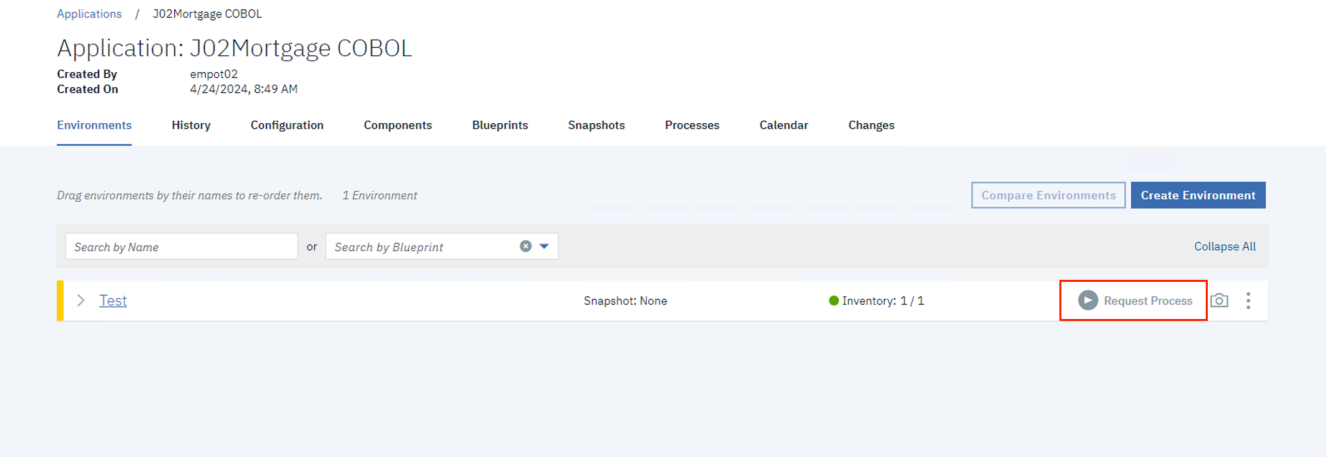
Task 11 – Run an application process

You are now ready to test our deployment process.

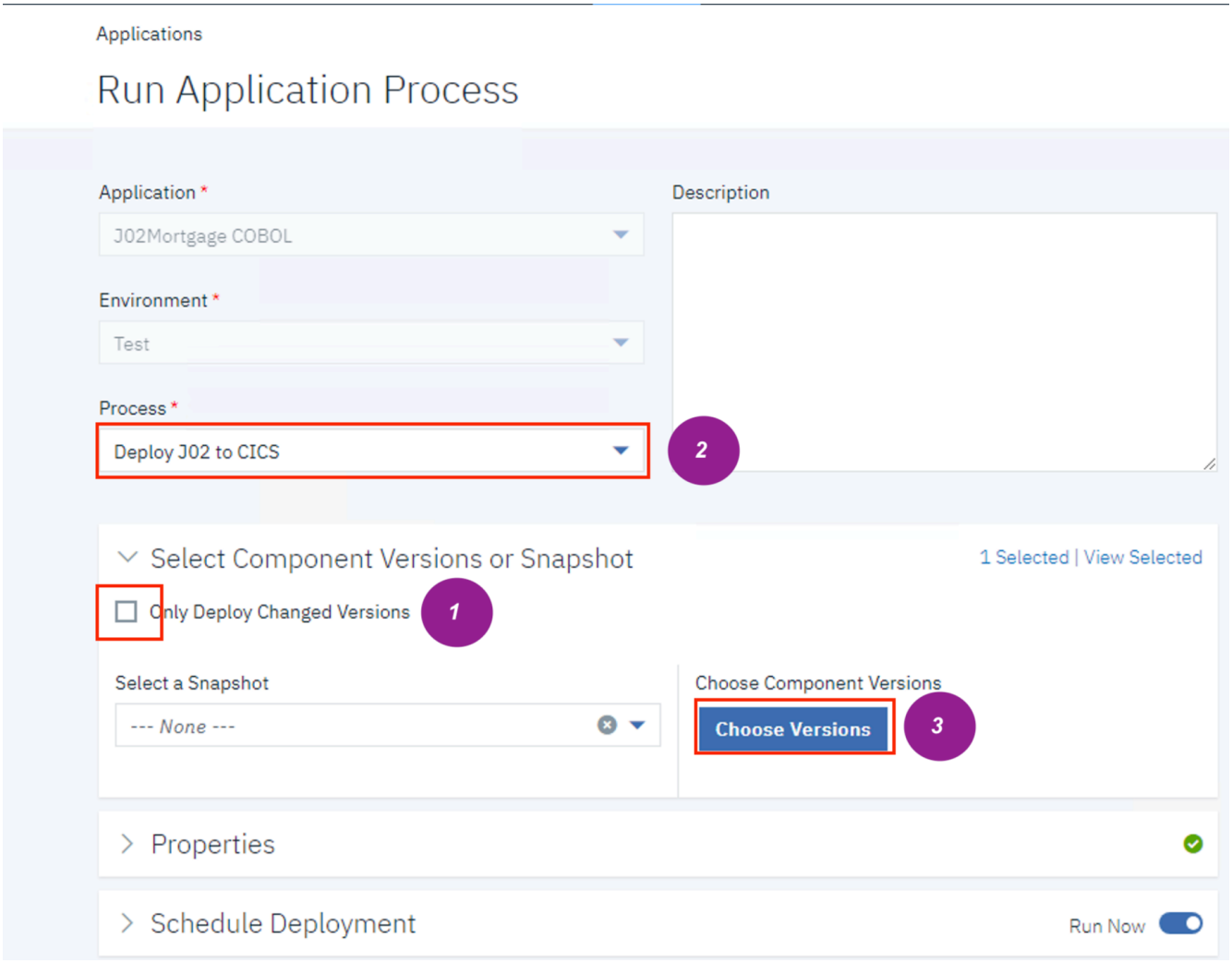
11.1 Click the **Applications** tab and then click **J02 Mortgage COBOL**



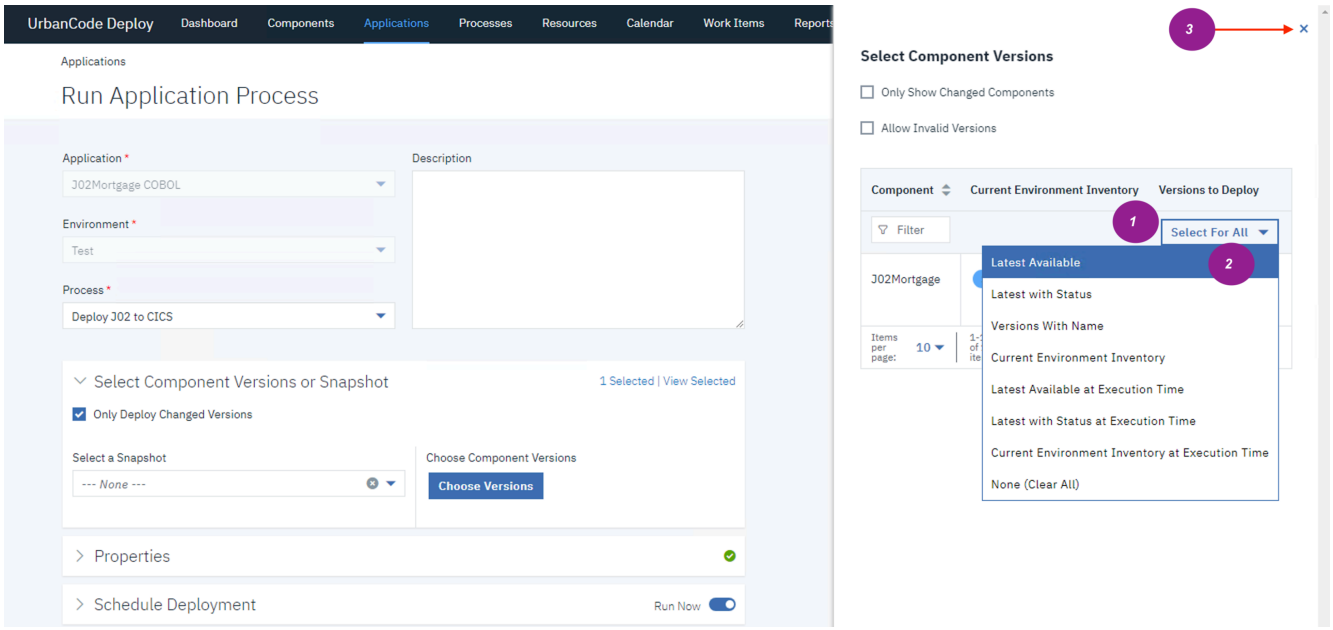
11.2 Under **Test** environment click the **Request Process** icon



11.3 In the *Run Process* window, **un-select Only Changed** Versions, in the *Process*, select the **Deploy J02 to CICS** and Under *Component Versions*, click **Choose Versions**

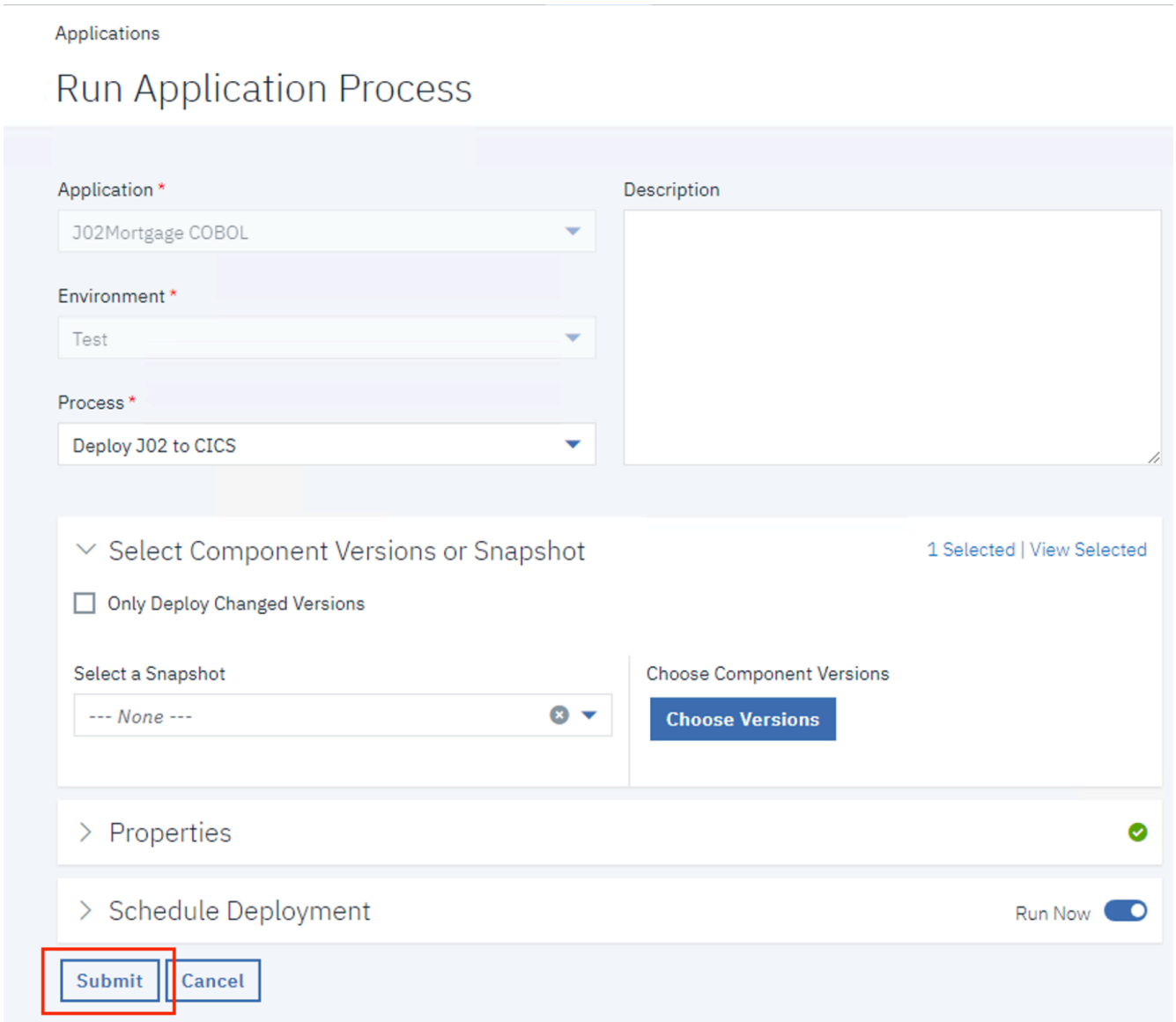


11.4 Click **Select For All** > **Latest available** click **on X**



Make sure that the version that you created on PART 1 of the lab is selected for **J02Mortgage**
If you do not select a version, that component is not deployed.

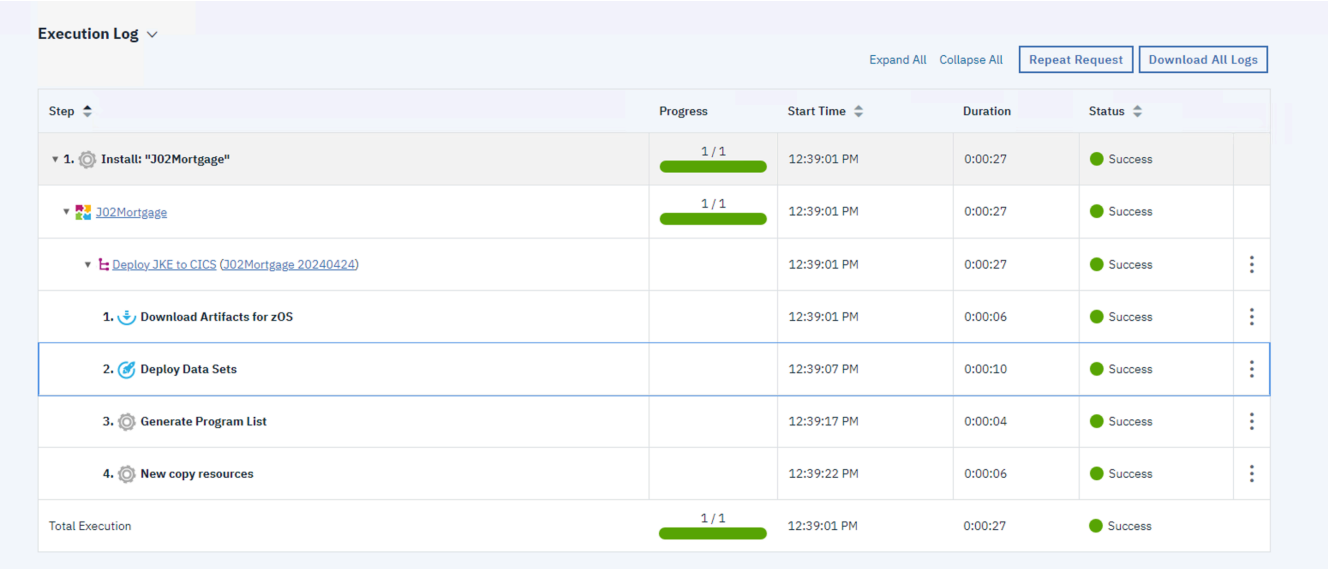
11.5 Click **OK** and **Submit**



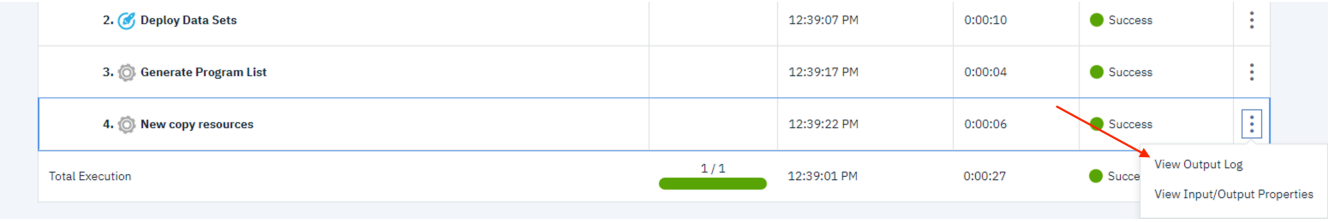
The web page shows you the progress of the application process request. From this page, you can watch as the processes run. The following figure shows the application process partially completed. The application component process is finished and the other two component processes are running.

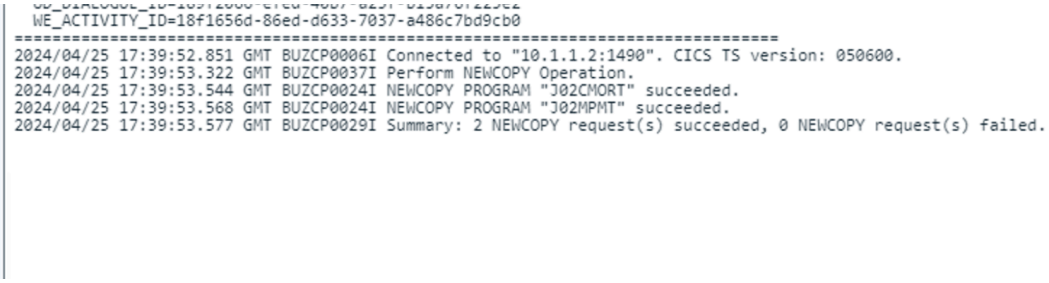
11.6 Click **Expand All** and expand **Deploy JKE to CICS** to see all steps and WAIT the deploy...

If the process runs successfully, the request shows that each component process is finished, as in the following figure:



11.7 Click on **Output Log** to see the CICS Programs NEWCOPY executed.





```
  WE_ACTIVITY_ID=18f1656d-86ed-d633-7037-a486c7bd9cb0
================================================================================
2024/04/25 17:39:52.851 GMT BUZCP0006I Connected to "10.1.1.2:1490". CICS TS version: 050600.
2024/04/25 17:39:53.322 GMT BUZCP0037I Perform NEWCOPY Operation.
2024/04/25 17:39:53.544 GMT BUZCP0024I NEWCOPY PROGRAM "J02CMORT" succeeded.
2024/04/25 17:39:53.568 GMT BUZCP0024I NEWCOPY PROGRAM "J02MPMT" succeeded.
2024/04/25 17:39:53.577 GMT BUZCP0029I Summary: 2 NEWCOPY request(s) succeeded, 0 NEWCOPY request(s) failed.
```

11.8 Close the web browser

Task 12 – Verifying the Deploy results at z/OS CICS

The z/OS components are unique for each student, so you can see the code that you deployed to CICS.

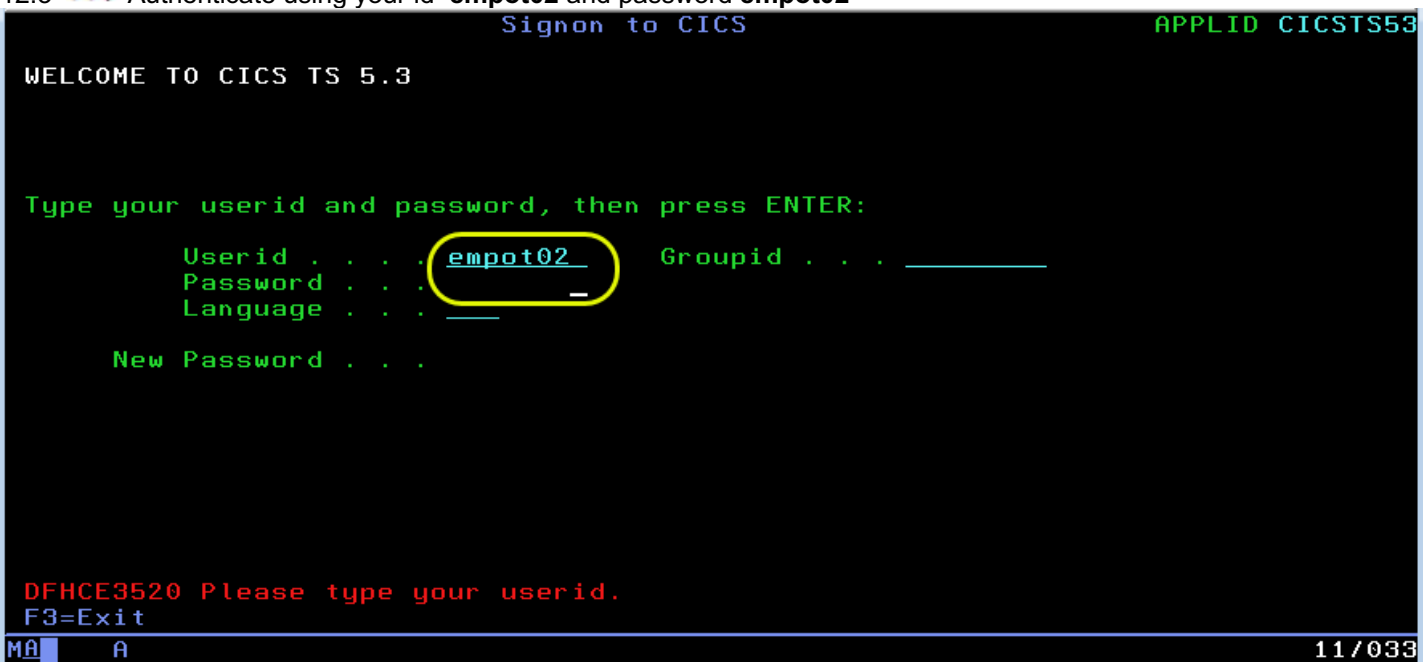
12.1 Use the 3270 emulator
Go to the windows desktop and **double click the 3270 emulation icon**

l cicsempotj02

12.2 Verify the CICS application. L cicsts56
Type **l cicsts53** (l is **L** lower case)

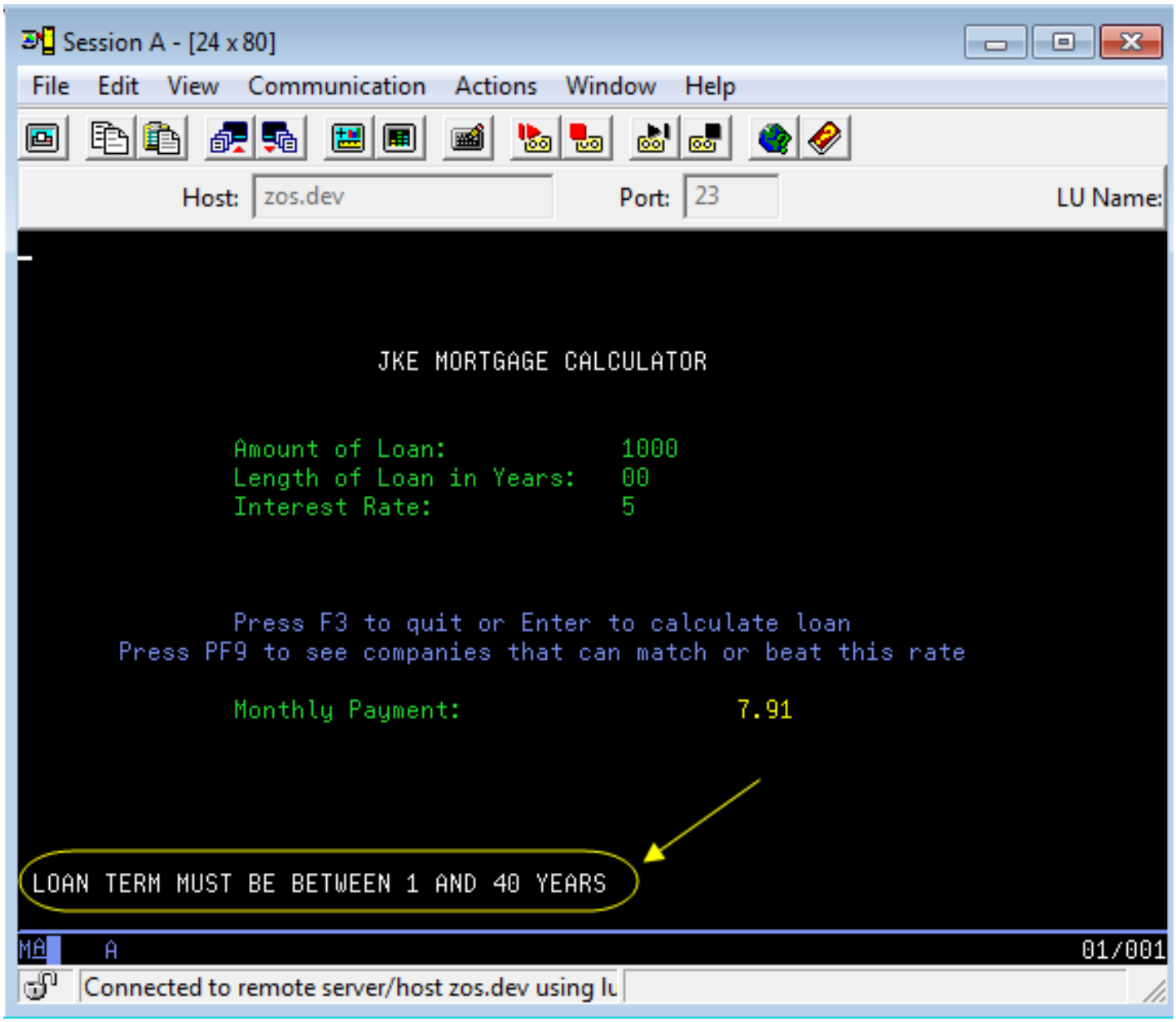
```
z/OS V2R5 LVLI  PUT2112/RSU2112                    IP Address = 10.1.1.1
                                                   VTAM Terminal = TCP00006

                    Application Developer System

                             //  OOOOOOO   SSSSS
                            //  OO    OO SS
                    zzzzzz //  OO    OO SS
                      zz  //  OO    OO SSSS
                     zz   //  OO    OO      SS
                   zz    //  OO    OO      SS
                  zzzzzz //   OOOOOOO  SSSS


                  System Customization - ADCD.Z25A.*



 ===> Enter "LOGON" followed by the TSO userid. Example "LOGON IBMUSER" or
 ===> Enter L followed by the APPLID
 ===> Examples: "L TSO", "L CICSTS56", "L CICSTS55",  "L IMS15"

l cicsts56_
MA    A                                                          24/011
```

12.3 Authenticate using your id **empot02** and password **empot02**



12.4 ⏭ Type the transaction **J02P**.

12.5 ⏭ Type **00** on Length of Loan in Years. Note that the error message.



Congratulations! You completed the lab 2

LAB 3 – Using IBM Dependency Based Build with Git, Jenkins and UCD on z/OS

Updated June,2024 (revised by Carlos Hirata/Wilbert Kho, Reviewed by Joe Huang and Frank Hernandez)

This lab will take you through the steps of using IBM Dependency Based Build (DBB) along with Git, Jenkins and UrbanCode Deploy (UCD) on z/OS.

On this lab you will modify an existing COBOL/CICS application stored in Git.
You will use **IDz** (which is part of the **ADFz** "package") to change the code and perform a personal test for later delivery and commit to *Git* and then use *Jenkins* for the final build and continuous delivery.

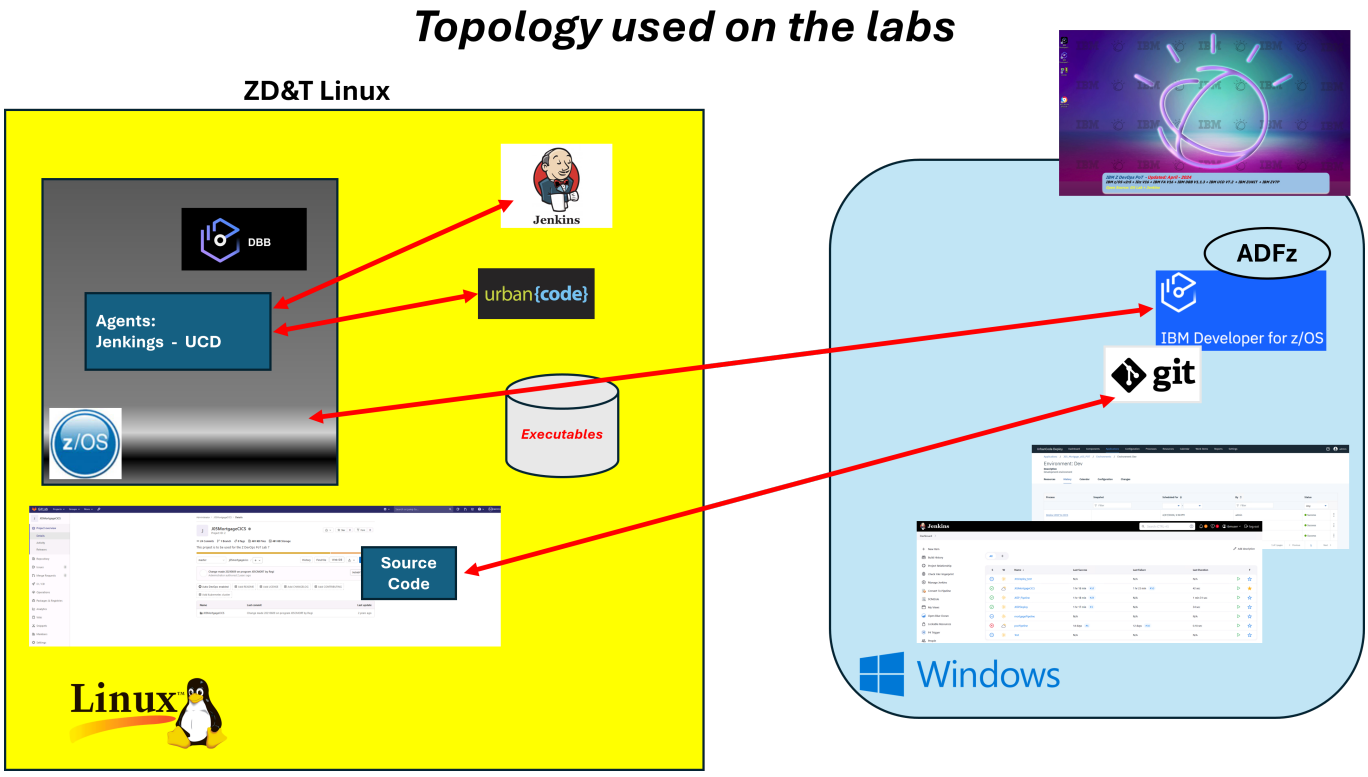
The updated code will be deployed to CICS using *UrbanCode Deploy* (UCD)

The process would be similar for a PL/I program or IMS instead of CICS.
More about DBB https://www.ibm.com/support/knowledgecenter/SS6T76_1.1.0/welcome.html

The environment used on this Lab is pictured below.

Note that we have few "servers" running on Linux like **UCD (UrbanCode Deploy)** server, **Jenkins** and also the **Git Repository (Gitlab)** is on Linux.

The z/OS has many other running tasks including **CICS, DB2, DBB server** and agents that interact with the Linux servers.



Overview of development tasks

To complete this tutorial, you will perform the following tasks:

1. **Get familiar with the application using the 3270 terminal**
   → You will start a 3270 emulation and execute a transaction named **J05P** to become familiar with the Application that you intend to modify.
2. **Load the source code from Git to the local IDz workspace**
   → You will load the *COBOL* code that is stored on Linux to your windows client to be modified.
3. **Modify the COBOL code using IDz.**
   → Using *IDz* you will modify the *COBOL* code to have a different message in a *CICS* dialog.
4. **Use IDz *DBB User Build* to compile/bind and perform personal tests.**
   → You will compile and link the modified code using the *DBB User Build* Function. When complete you will run the code using *CICS* for a personal test and verify that the change is correctly implemented.
5. **Push and Commit the changed code to Git .**
   → You will commit the changes to *Git*.
6. **Use Jenkins with Git plugin to build all the modified code committed to Git.**
   → You will use Jenkins pipeline to build the new changed code and push the executables to be deployed using *UCD*.
   Note that this step makes a build of all code committed to Git, while on step 4 only the selected COBOL program was built.
7. **Use Jenkins and UCD plugin to deploy results and test the CICS transaction again**
   → You will verify the results after the final deploy to *CICS* using *UCD*
8. **Understanding DBB Build Reports**
   → You will understand the reports generated by *DBB* during the build.

What is Git and DBB ?
Git is an open Source Code Management tool that is very popular in the distributed world.
In early 2017, Rocket Software ported Git into the mainframe – with the necessary checks to handle EBCDIC to UTF-8 conversions and vice-versa.

In Q3 2017, IBM released an Open Beta of **Dependency Based Build (DBB).** DBB provides a build tool that provides the build framework, dependency understanding, and tracking for builds run on z/OS. This build system is not dependent on any SCM or Continuous integration automation tool. In this lab, we use DBB to build our z/OS COBOL source code which resides in the distributed Git Repositories.
DBB is part of IBM Developer for Z Systems EE (Enterprise Edition) or ADFz and was announced on March 13, 2018.

GitHub *vs.* ***Bitbucket*** vs. ***GitLab*** **?**
More at: https://stackshare.io/stackups/bitbucket-vs-github-vs-gitlab

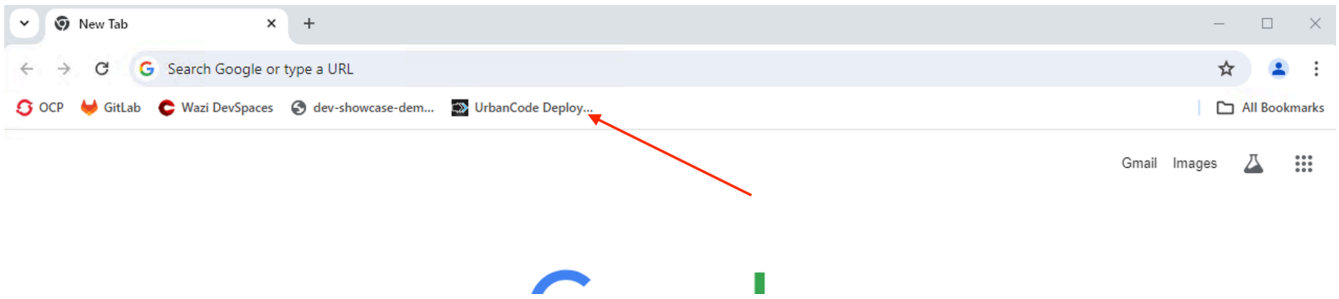
GitHub, ***Bitbucket,*** and ***GitLab*** are code collaboration and version control tools offering repository management. They each have their share of fans, though ***GitHub*** is by far the most used of the three.
Of the three, only ***GitLab*** is open source, though all three support open source projects. For that reason we are using ***GitLab*** in this lab, but the steps would be exactly the same if using other tools.
GitHub offers free public repositories; ***Bitbucket*** also offers free private repositories; ***GitLab*** offers a Community Edition which is entirely free

Section 0. Checking UCD agent is ONLINE.

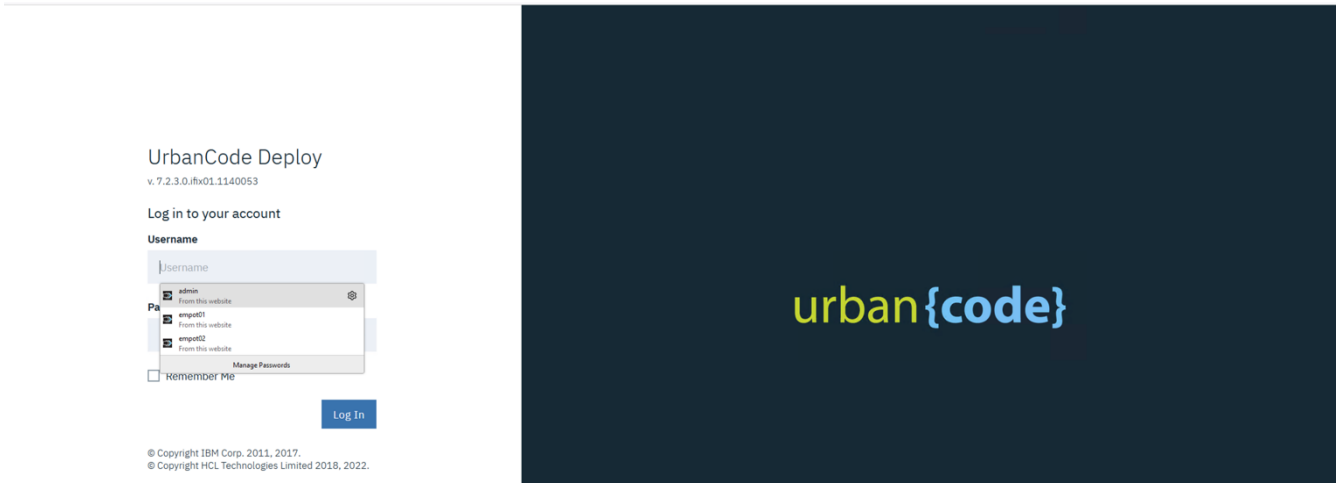
Before start UCD lab, let's verify UCD agent (zOS) is connect to UCD server (Linux Server).

0.1.1 Open firefox browser.

0.1.2 ⏭ Look at Bookmark toolbar and select **UrbanCode Deploy**

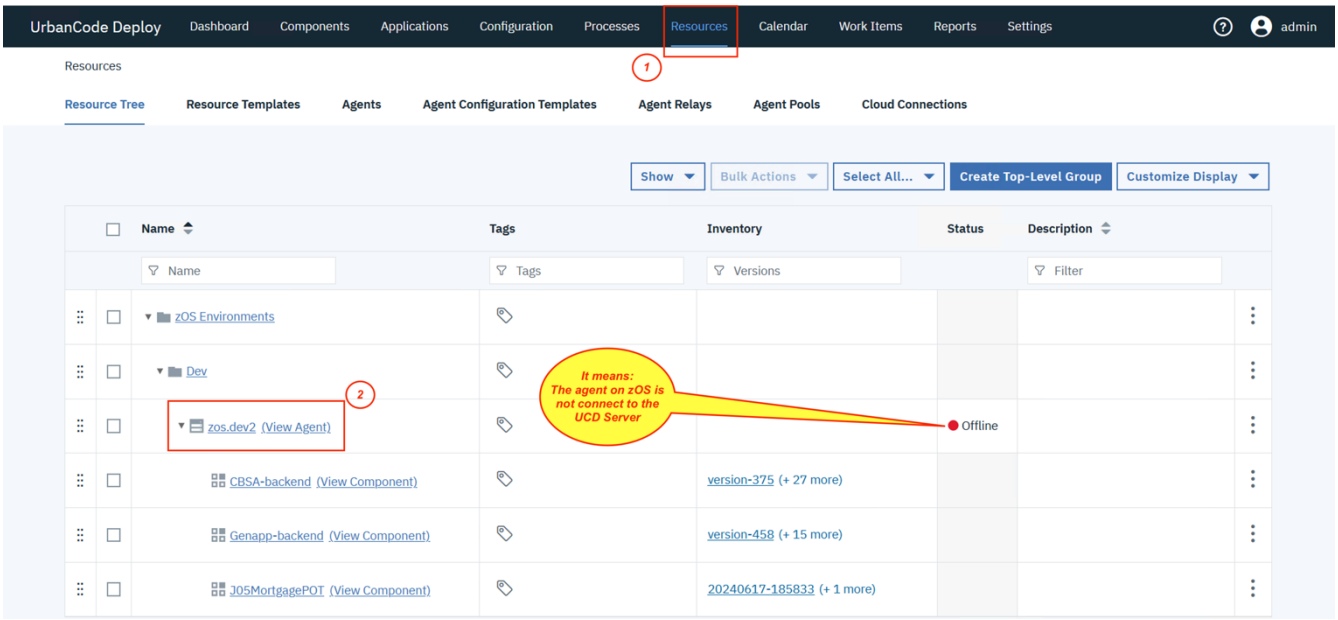


0.1.3. ⏭ Login with **`admin`** and password **`admin`**

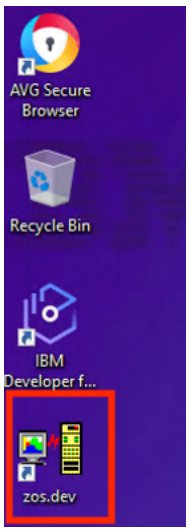


0.1.4 On Urbancode page, locate Resources tab. Look at the agent (zos.dev2) and check the status.

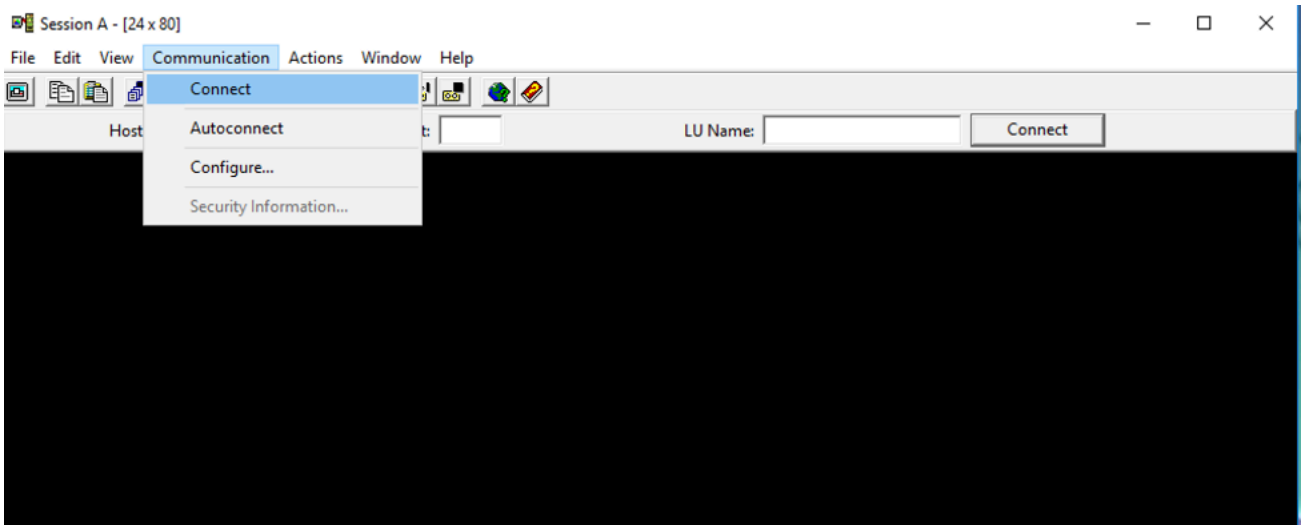
If the status is Online, it is good and skip to Task 1 in this lab. If the status is Offline. You must start the agent on zos.



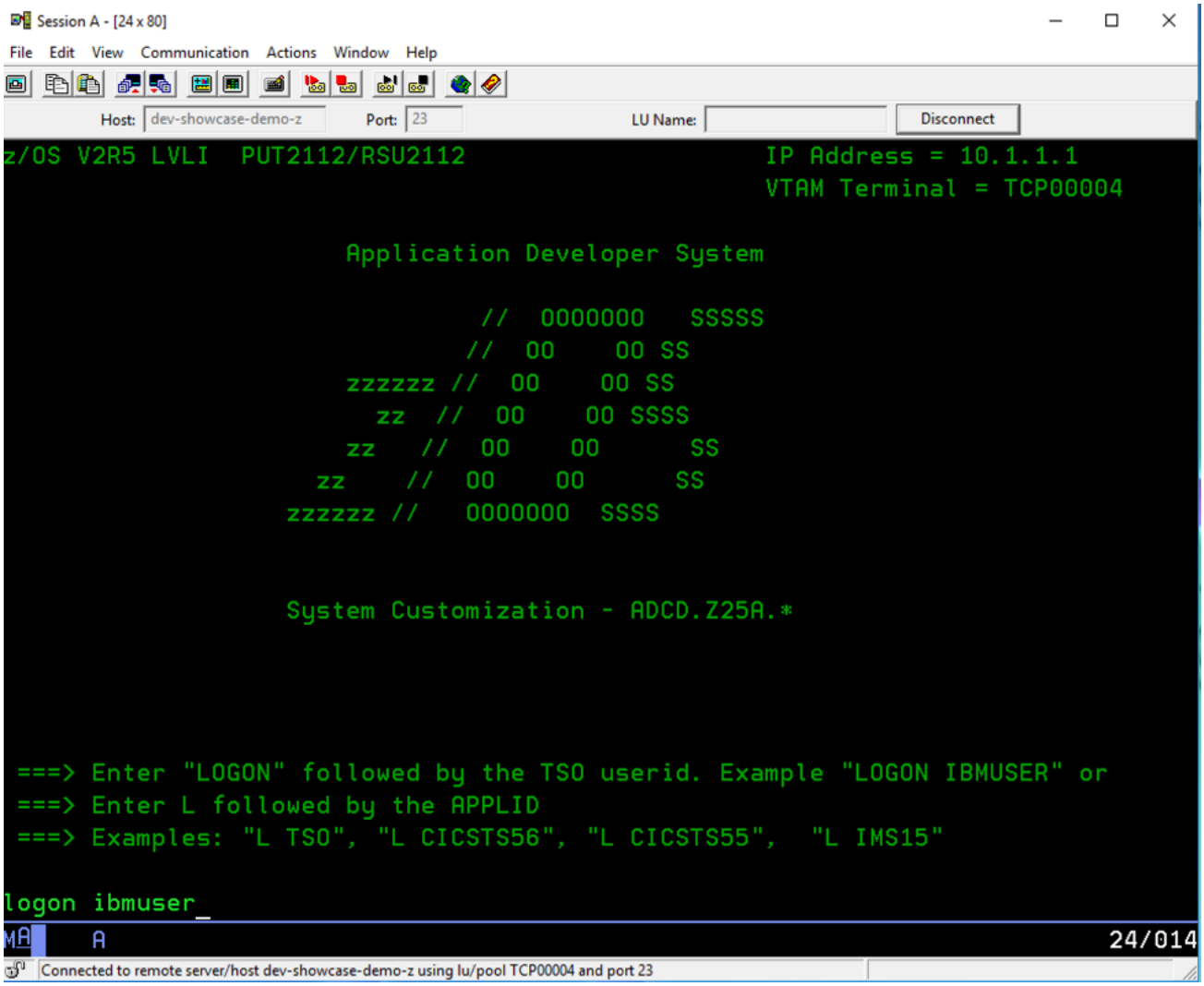
0.1.5 On Windows Desktop, click ⏭ zos.dev icon and open PCMM application.



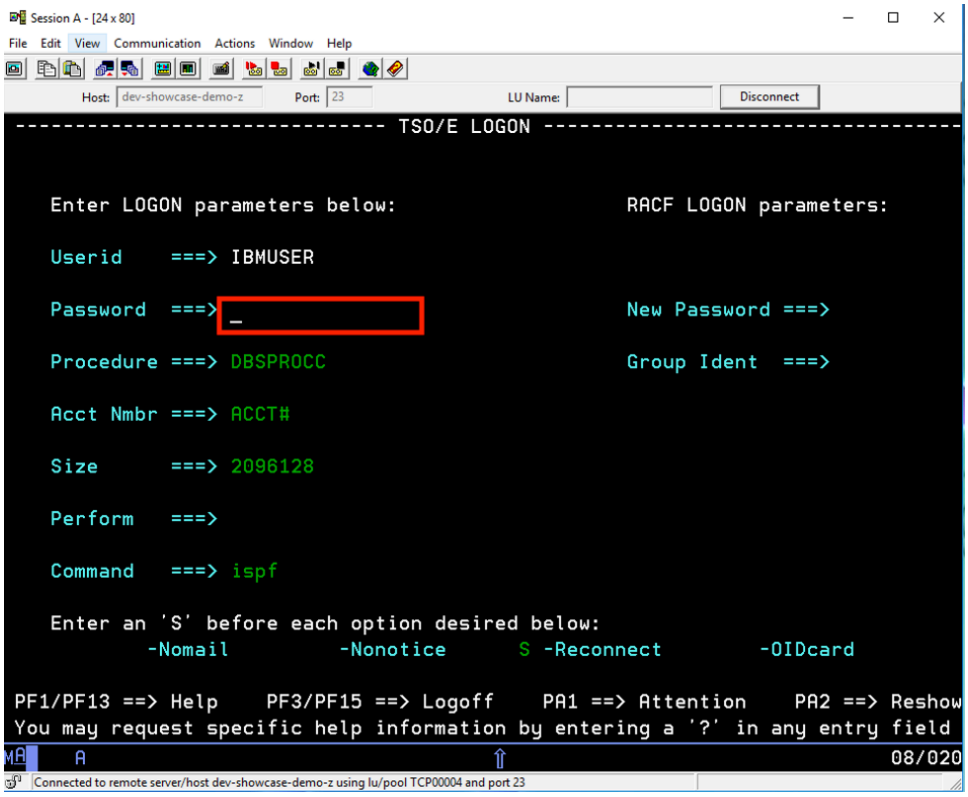
0.1.6 Click ⏭ on Communication → Connect



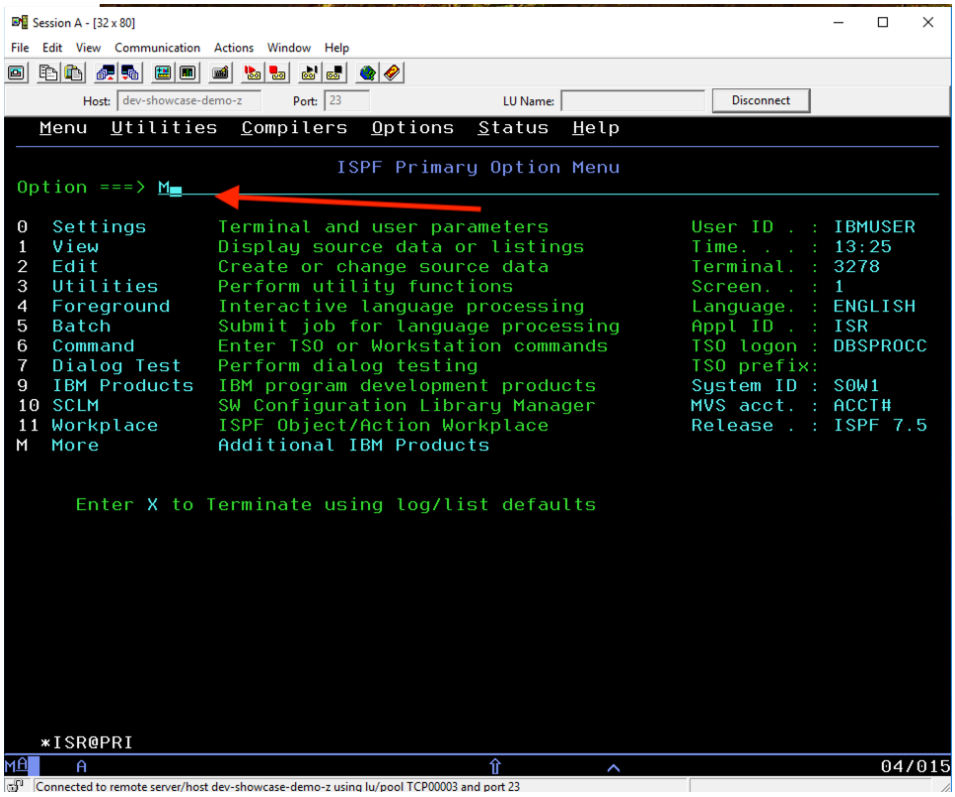
0.1.7 The z/OS main screem will show up and enter logon ibmuser (it can be lower case).



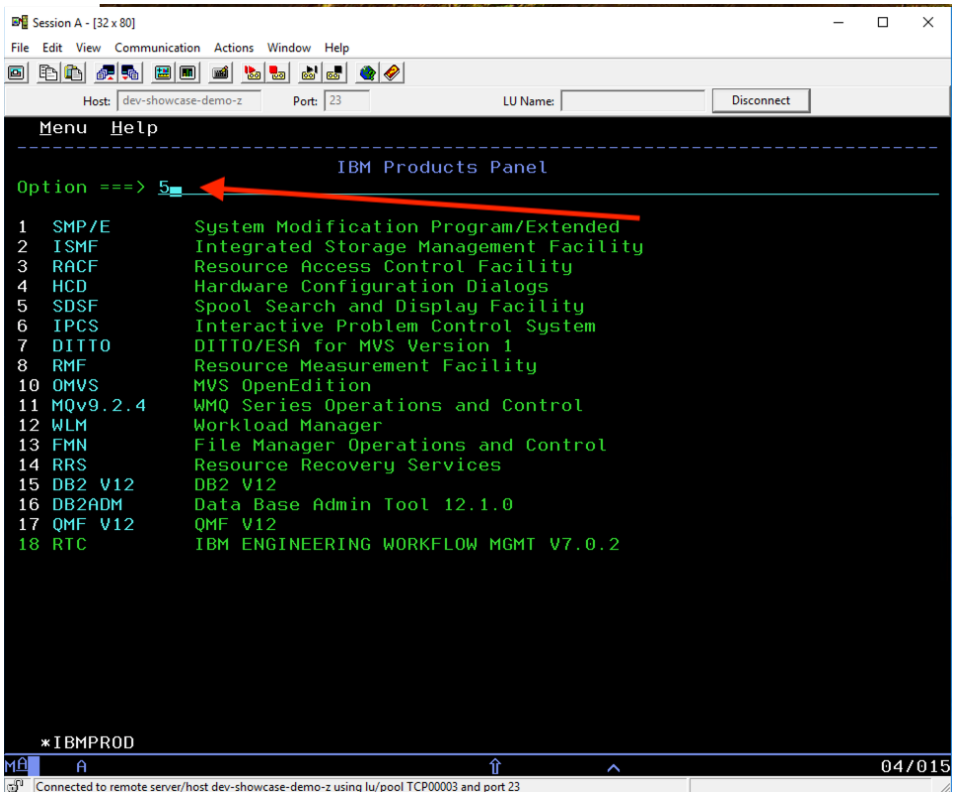
0.1.8 The userid is IBMUSER and the password is **sys1.** Press **ENTER** and **ENTER.**



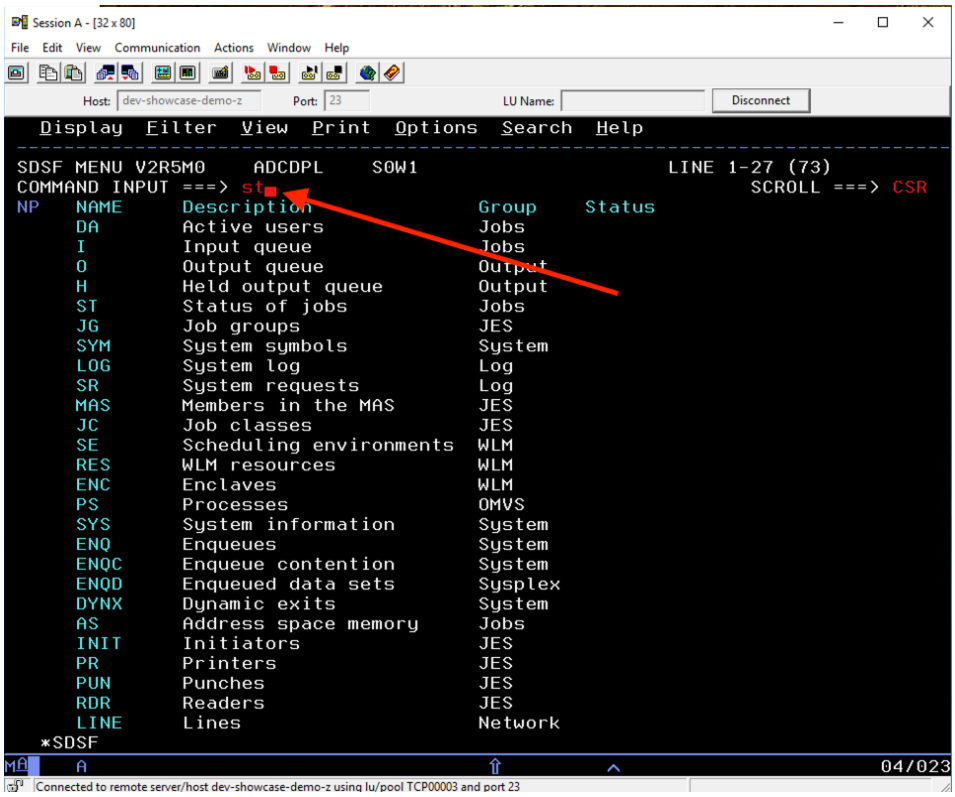
0.1.9 On ISPF Primary Menu. Enter the letter **M** on **Option** field.Press **ENTER.**



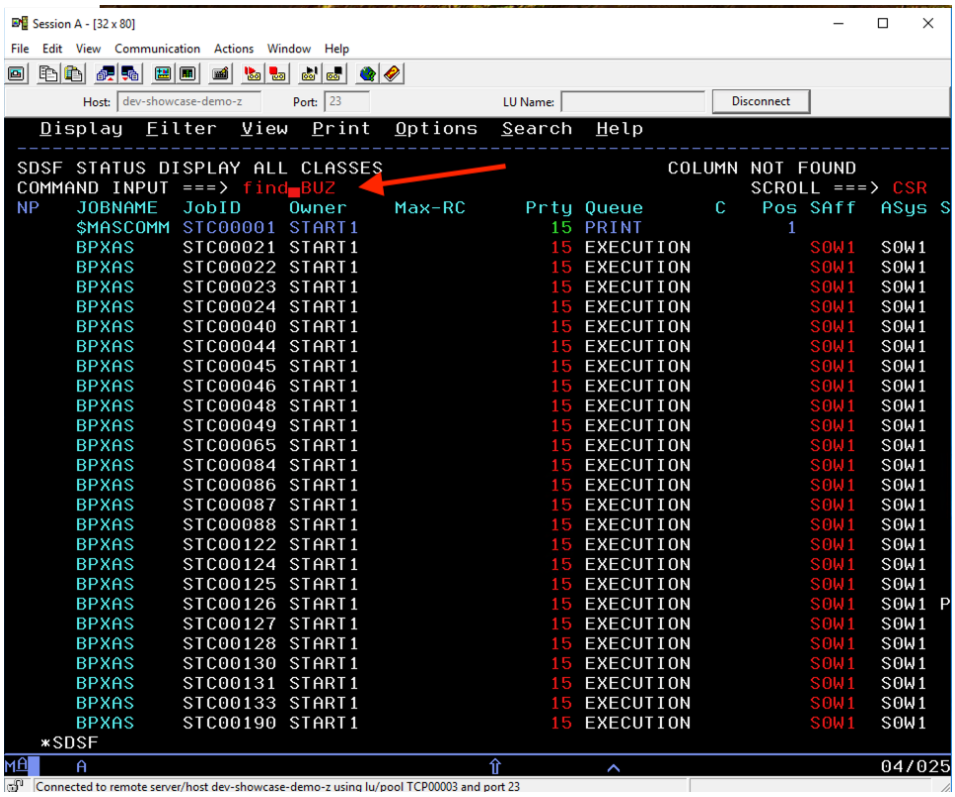
0.1.10 On IBM Products Panel. Enter the number **5** to go to **Spool Search and Display Facility.**



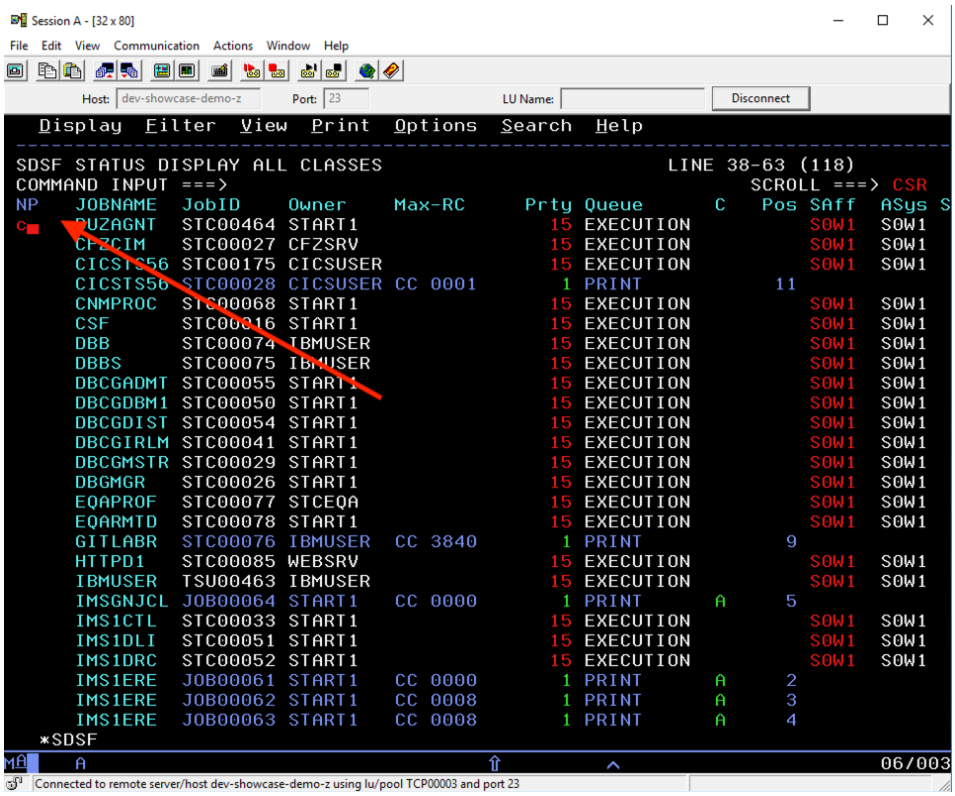
0.1.11 On **SDSF Menu.** Type **st** command on **COMMAND INPUT** and press **ENTER.**



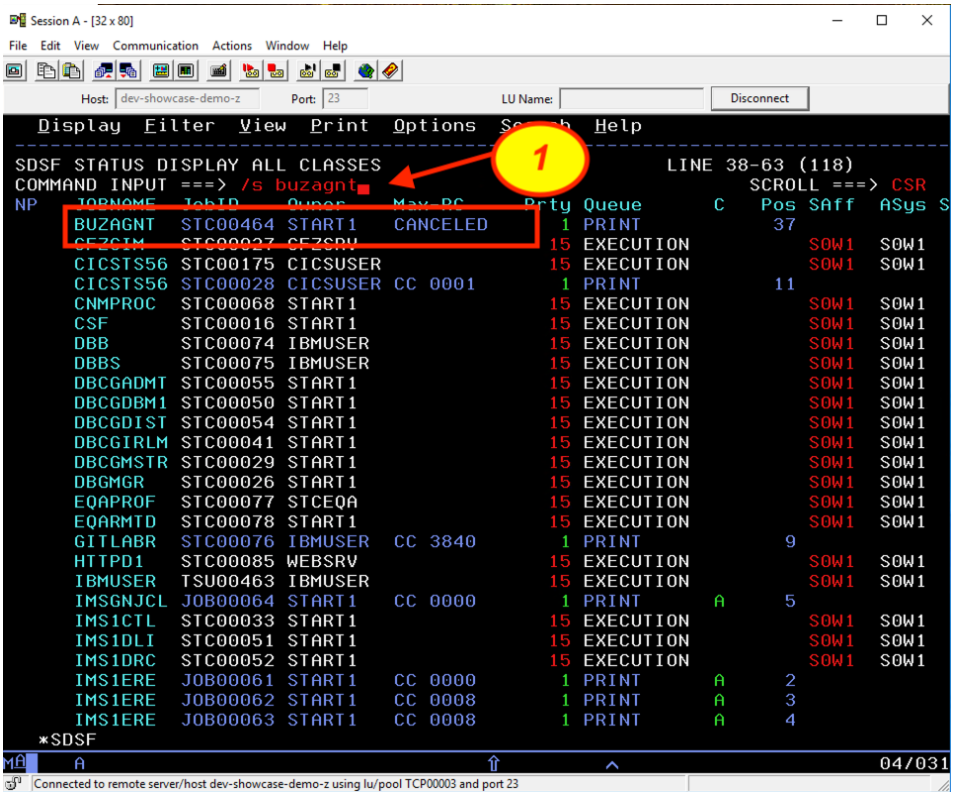
0.1.12 On **SDSF STATUS DISPLAY ALL CLASSES,** type **find buz (**3 letters is enough**)** and press **ENTER.**



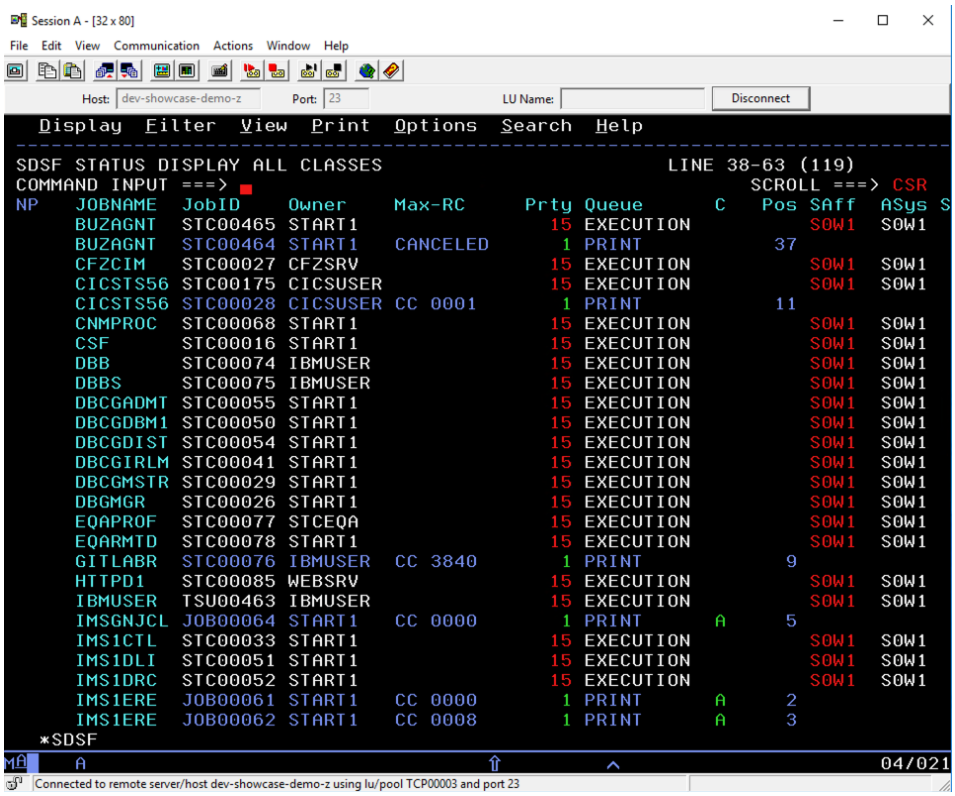
0.1.13 **BUZAGNT** agent might be started, but it is not connected to the UCD server. It is necessary to cancel this JOB and start it again. On the column **NP**, type letter **c** the front of **BUZAGNT** job and press **ENTER.**



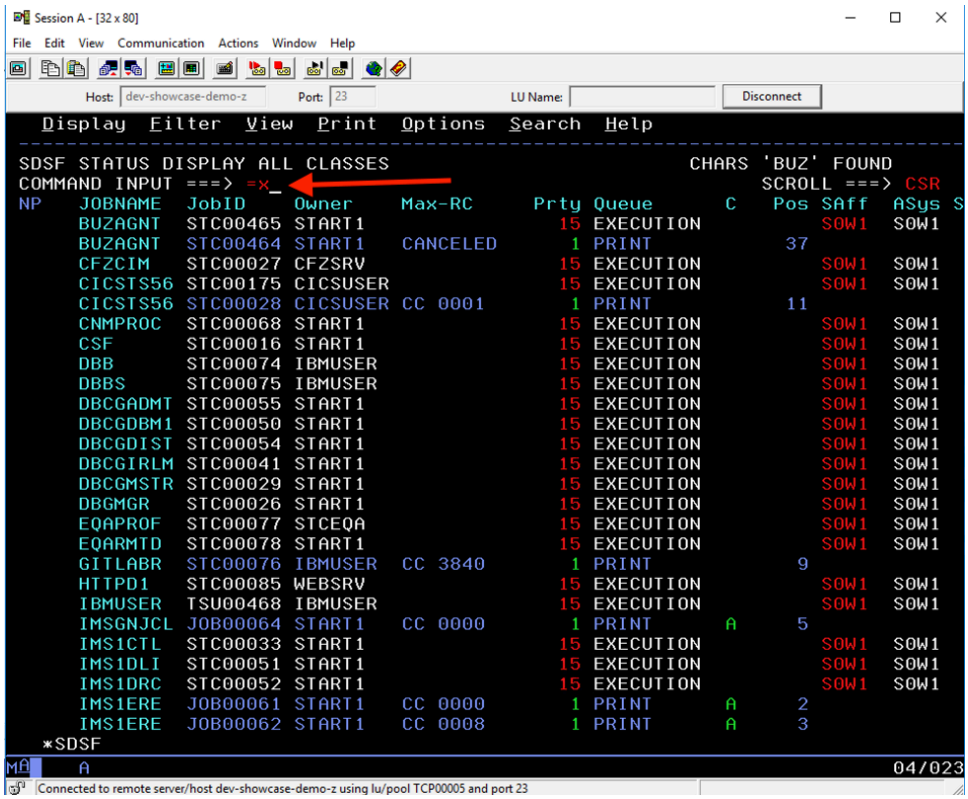
0.1.14 BUZAGNT is canceled and now start the BUZAGNT. On **COMMAND INPUT,** type **/s BUZAGNT**. Press **ENTER**.

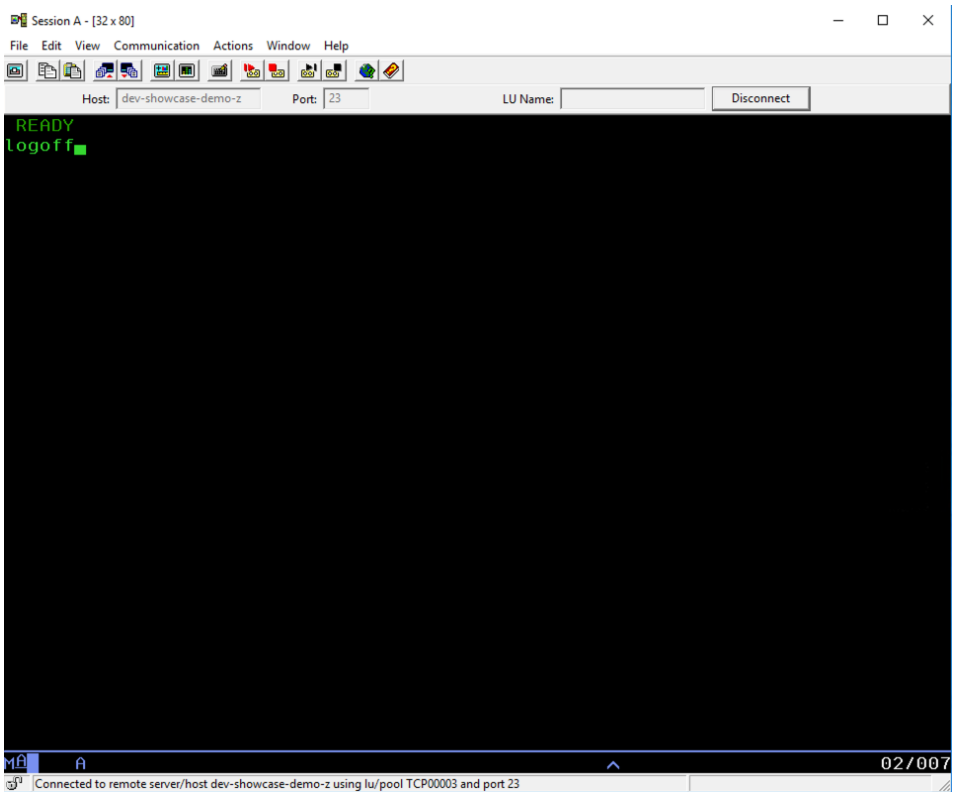


0.1.15 BUZAGNT agent is connect.

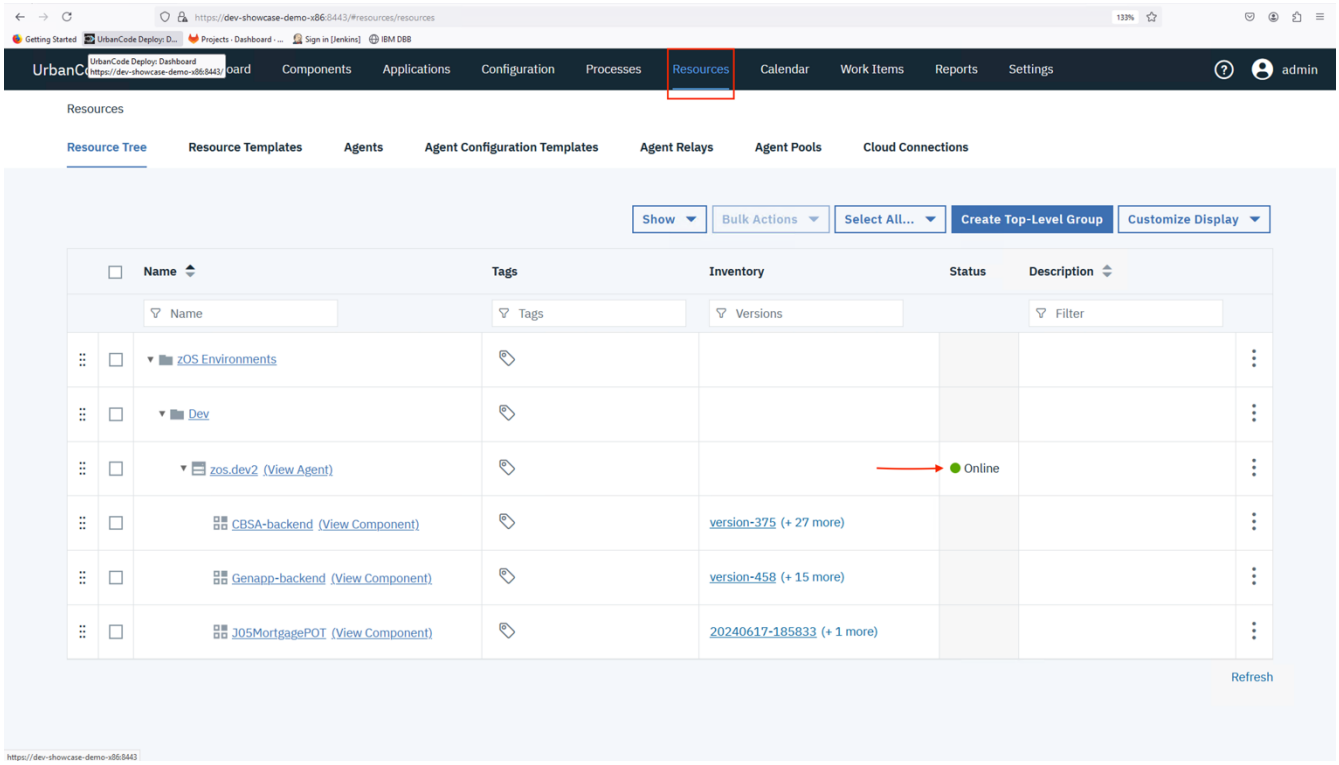


0.1.16 Logout the PCOMM session. On COMMAND INPUT, type **=x** press **ENTER.** and type **logoff** and **ENTER.**





0.1.17 Back to Firefox browser on UCD page, locate **Resource** and check **zos.dev2** is **Online**.



Section 1. Get familiar with the application using the 3270 terminal

You will start a 3270 emulation and execute a transaction named J05P to become familiar with the Application that you intend to update.

___ 1.1 Connect to z/OS and emulating a CICS 3270 terminal

1.1.1 Start *IBM Developer for z Systems version 16* if it is not already started

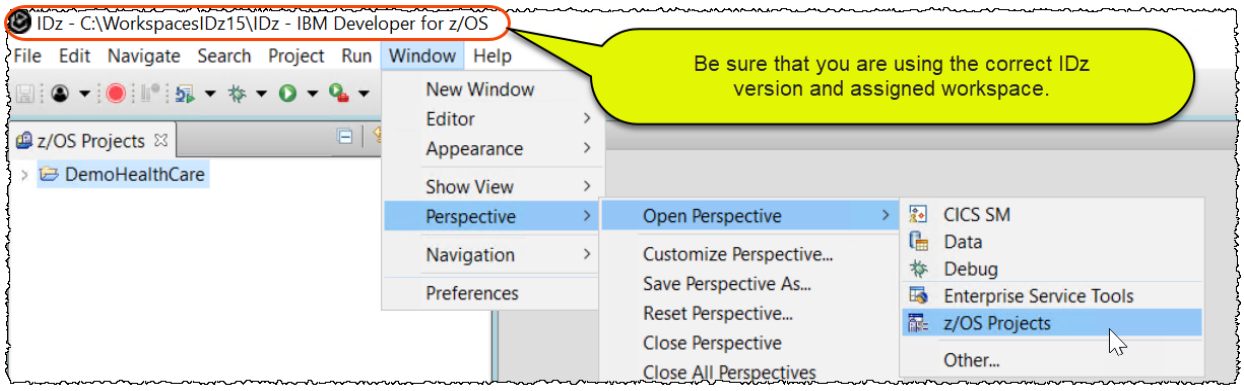
⏭ Using the desktop double click on **IDz V16** icon.

⏭ Verify that the message indicates that it is Version **16.0.3**

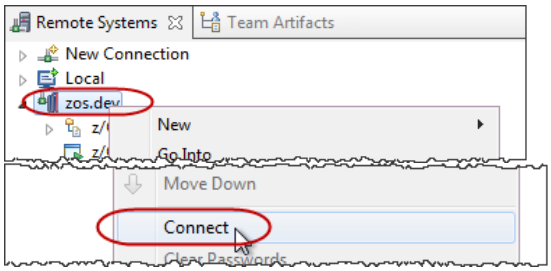
IMPORTANT -> This icon will start an eclipse workspace that has already some definitions required for this lab. PLEASE DO NOT start IDz using other way than this icon.

IBM Developer for z/OS®
Version 16.0

1.1.2 ⏭ Open the **z/OS Projects** perspective by selecting
Window > Perspective > Open Perspective > z/OS Projects

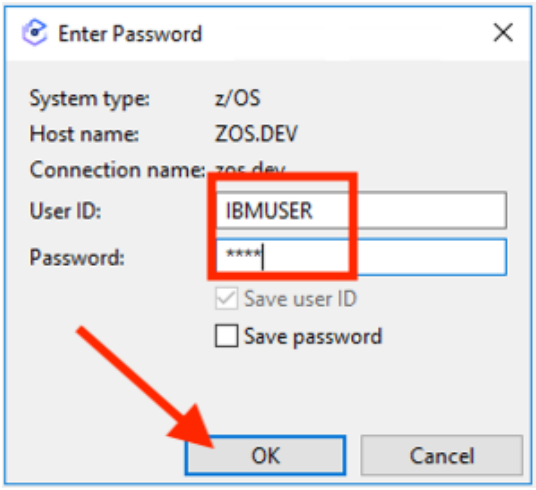


1.1.3 On other labs you used userid **empot01.**
Your new userid now will be **ibmuser**.
You need to disconnect and reconnect to the other userid.
If you are already connected, right click on **zos.dev** and select **Disconnect.** 1.1.4 Right click on **zos.dev** and select **Connect.**



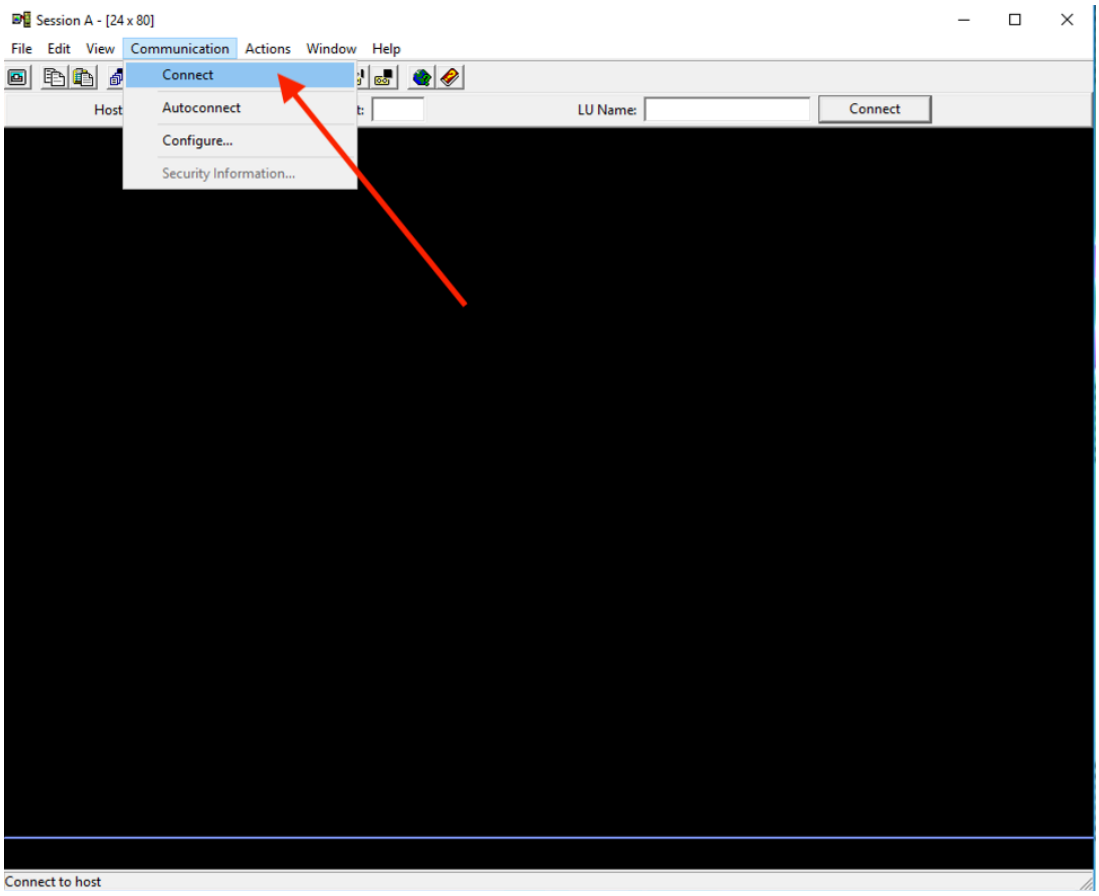
1.1.5 Type **ibmuser** as userid and **sys1** as password.
The userid and password can be any case; don't worry about having it in UPPER case.
Click **OK** to connect to z/OS.

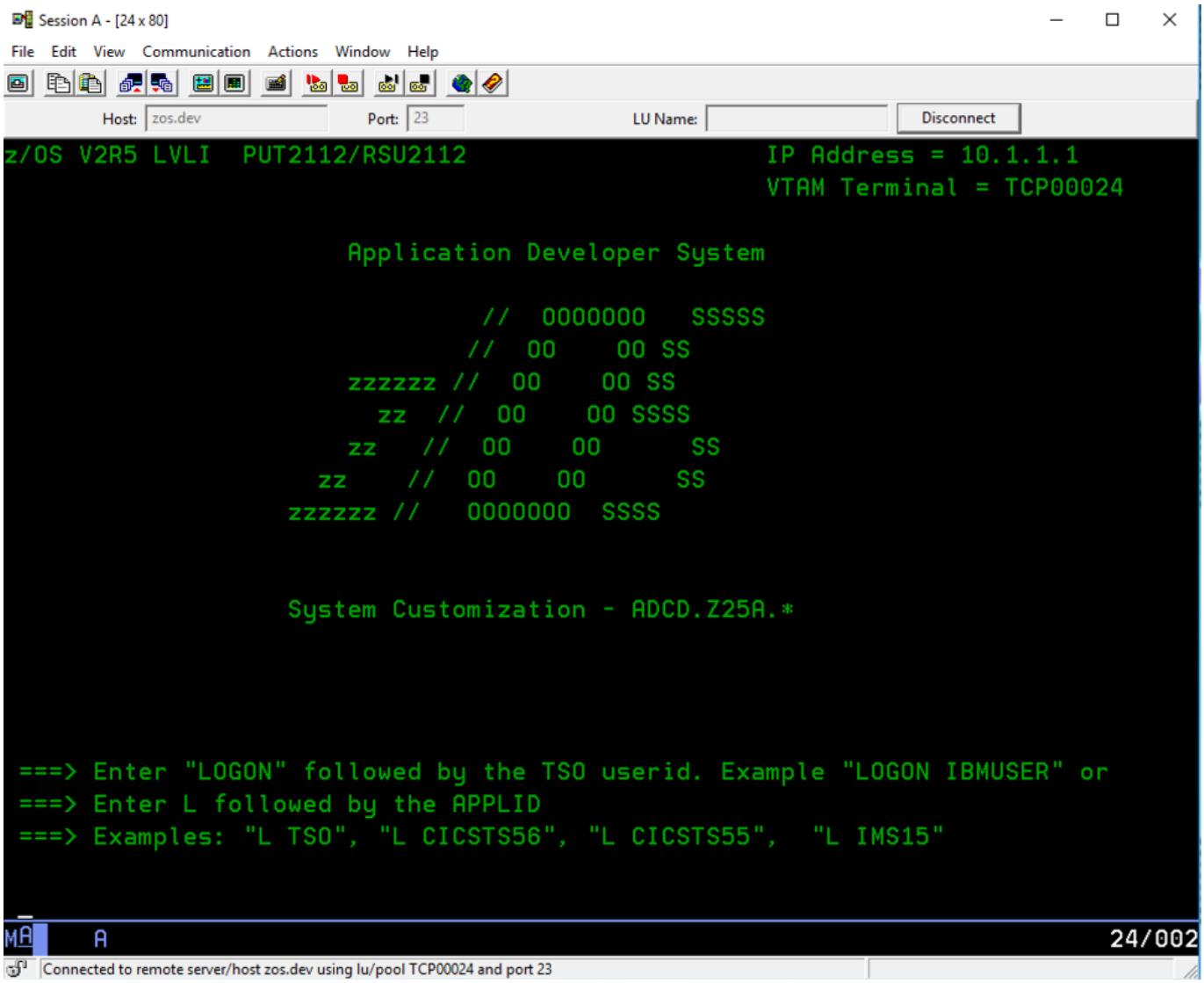
Notice → On this lab your userid will be **ibmuser** (not empot01 used in previous labs)



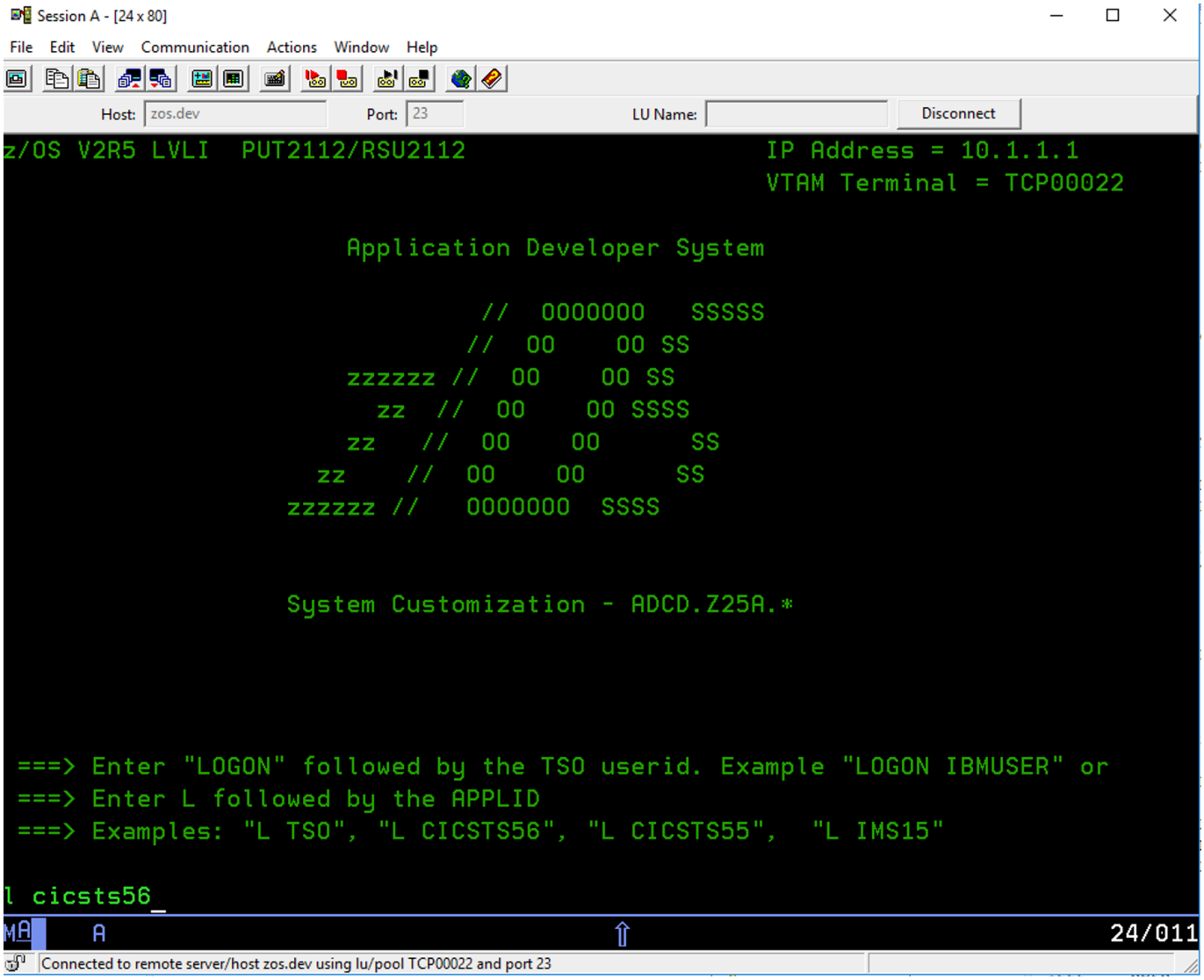
1.1.6 On Windows Destop click on **zos.dev** to access z/OS using PCOMM

1.1.7 Since you cannot connect. On the top of PCOM window, select Connect and you see VTAM welcome screen.

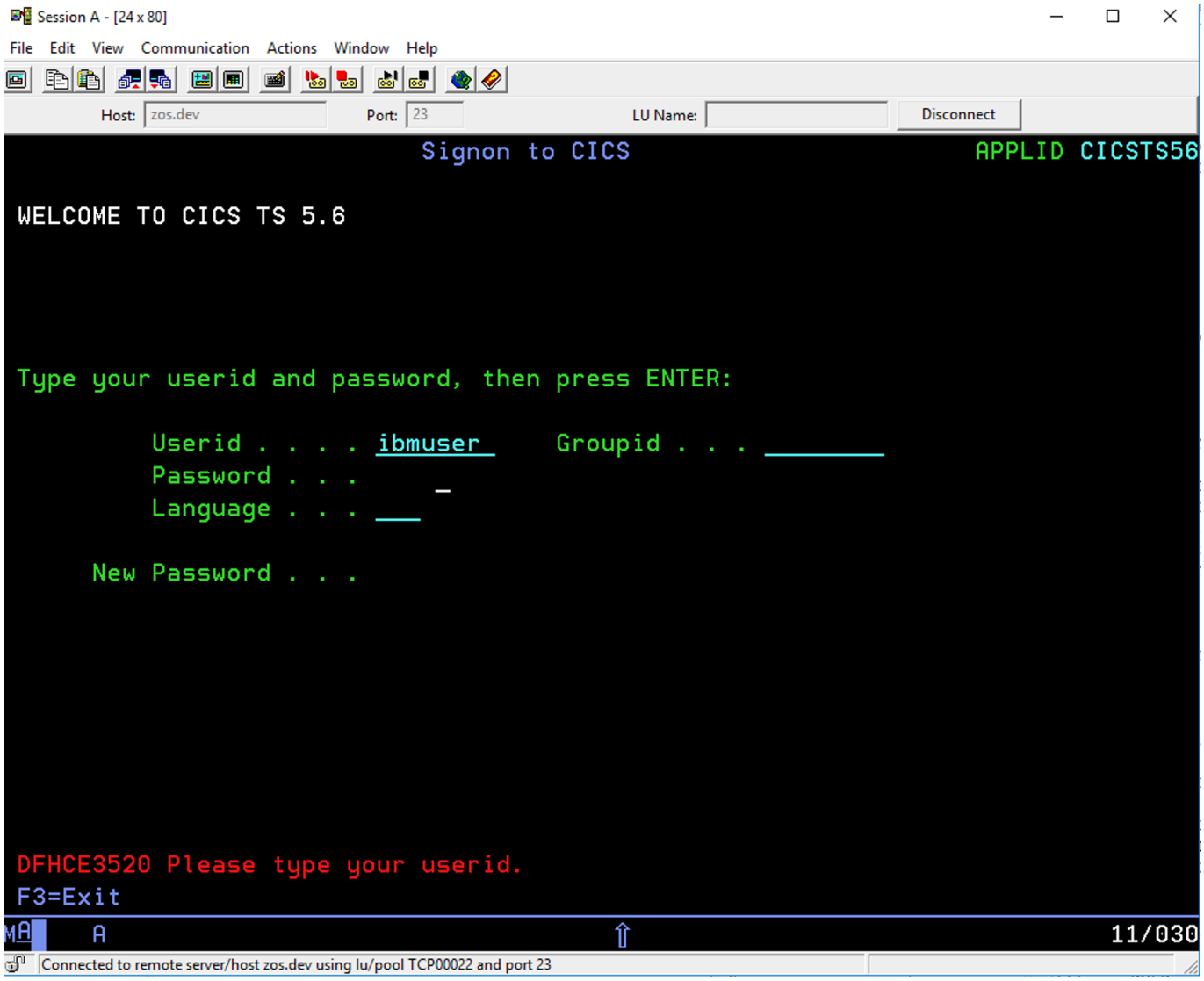



Session A - [24 x 80]
File Edit View Communication Actions Window Help
Host: zos.dev
Port: 23
LU Name:
Disconnect
z/OS V2R5 LVLI PUT2112/RSU2112
IP Address = 10.1.1.1
VTAM Terminal = TCP00024
Application Developer System
// OOOOOOO SSSSS
// OO OO SS
zzzzzz // OO OO SS
zz // OO OO SSSS
zz // OO OO SS
zz // OO OO SS
zzzzzz // OOOOOOO SSSS
System Customization - ADCD.Z25A.*
===> Enter "LOGON" followed by the TSO userid. Example "LOGON IBMUSER" or
===> Enter L followed by the APPLID
===> Examples: "L TSO", "L CICSTS56", "L CICSTS55", "L IMS15"
MA A 24/002
Connected to remote server/host zos.dev using lu/pool TCP00024 and port 23

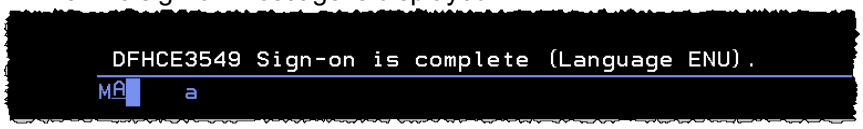
1.1.8 Type **l cicsts56**. (where “l” is the lower case of letter “L”) and press **Enter key**.



1.1.9 Logon using **ibmuser** and password **sys1** and press **Enter**.



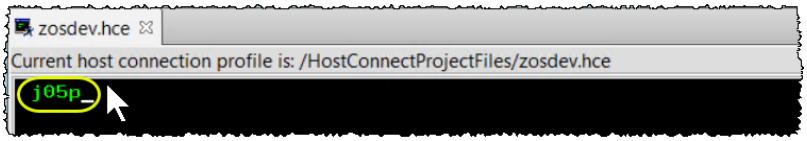
1.1.10 The sign-on message is displayed



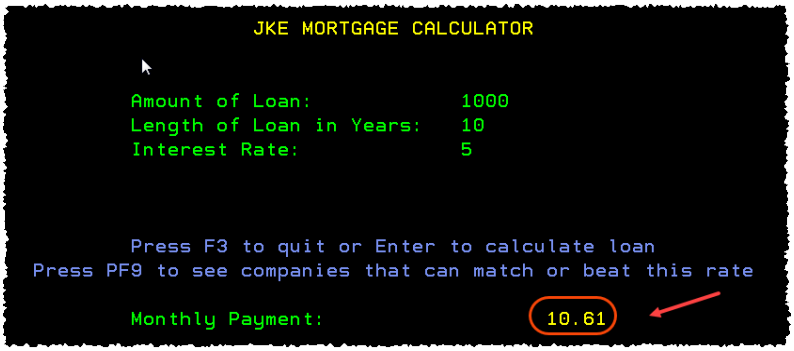
___ 1.2 Run CICS transaction J05P

You should now be in the z/OS CICS region named *CICSTS53*. This is the CICS instance where you will make the program changes.

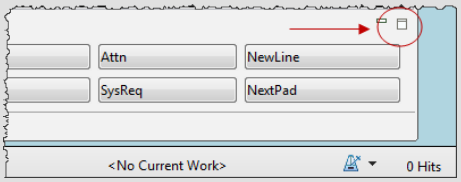
1.2.1 Type the CICS transaction **j05p** and press the **Enter** key.



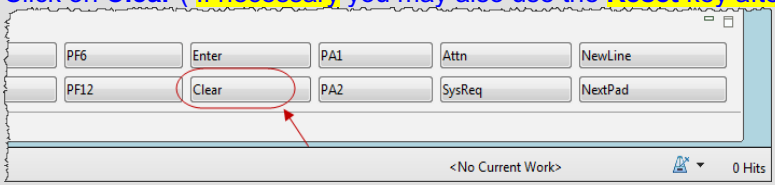
1.2.2 Press **enter** to see the monthly payment for the default data .



You may need to use the **clear** key. If the clear button is not displayed, look in the right lower corner, select this icon . This will display possible keys, including the clear button.



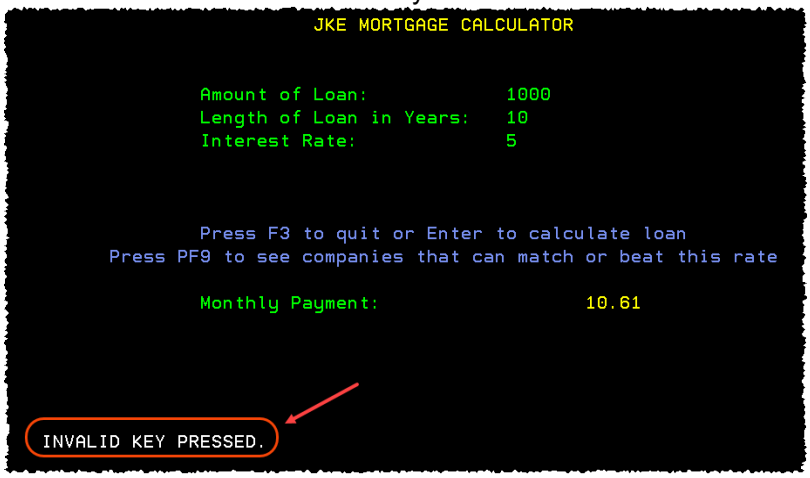
Click on **Clear** (If necessary you may also use the **Reset** key after clicking **NextPad**)



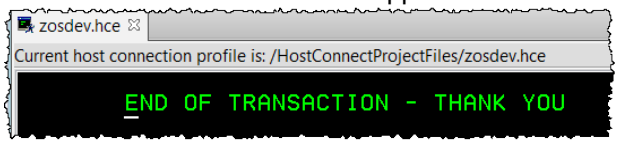
1.2.3 Press **F1** key and verify the message displayed "***INVALID KEY PRESSED***" .
Your mission will be modifying the COBOL program that send this message.
Your new message must be " **YYYYMMDD - INVALID KEY PRESSED"**.

Mac Users: Press **fn + F1**

Mac Users:
Where **YYYMMDD** will be the today's date.



1.2.4 Press **F3** to end the application.



1.2.5 **Close** the terminal emulation clicking on → .
Or pressing **CTRL + Shift + F4**.

What have you done so far?
You emulated a 3270 terminal using IBM Developer for System z.
You also executed the CICS transaction **J05P** and verified a simple interaction with the Mortgage application. The objective here was to show the code that you will update.

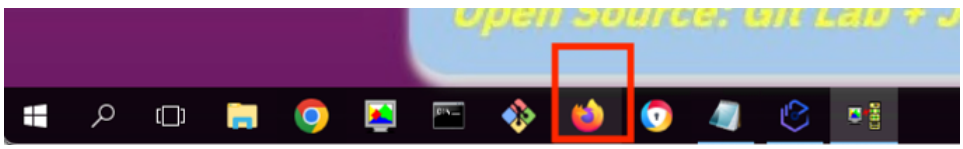
Section 2. Load the source code from Git to the local IDz workspace

You will load the COBOL code that is stored on Linux to your Windows client to be modified.

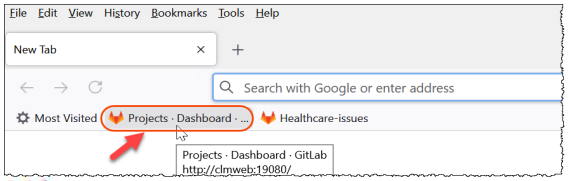
___ 2.1 Cloning the Git Repository

The Mortgage Application used in this lab has its source code in a *Git Repository* that is on Linux system. You must connect to the Git Repository and clone the Git project into the IDz. This brings the source code into your IDz workspace for edits and build. *GitLab* is open source and for that reason we are using *GitLab* in this lab. This lab would works also if using GitHub or Bitbucket.

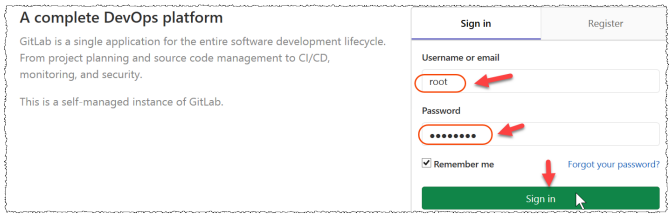
2.1.1 Before cloning the Git repository, you should visualize the code stored at GitLab.
Start a browser clicking in the icon in the bottom of your screen



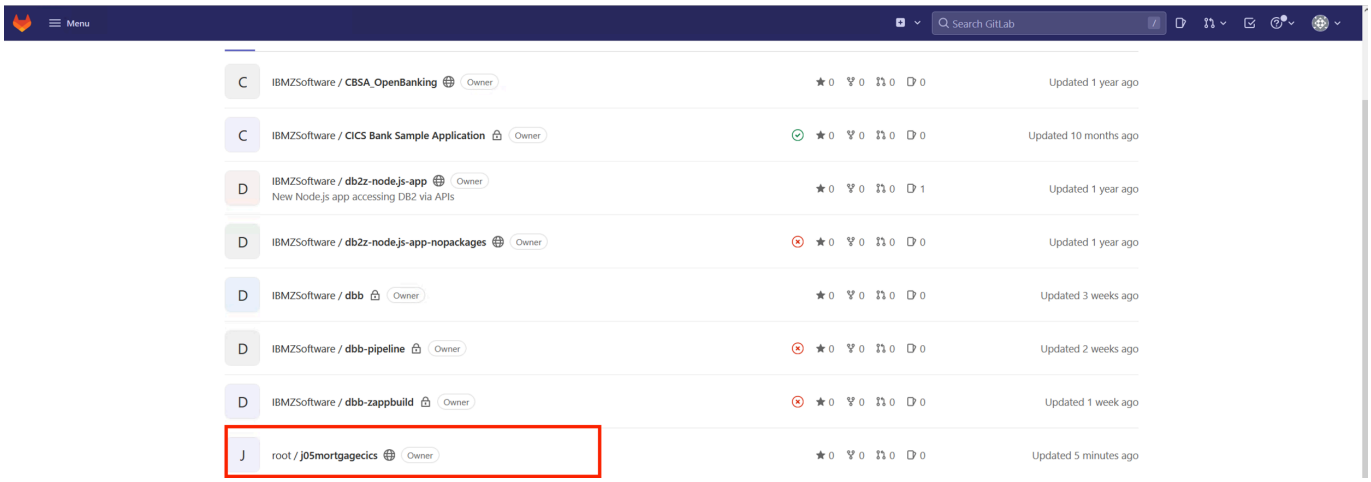
2.1.2 Click on this bookmark below to start ***GitLab***. The URL used is http://dev-showcase-demo-x86:8282/users/sign_in



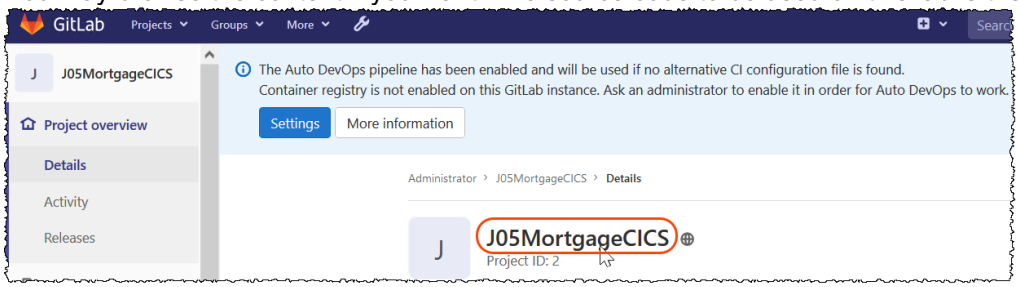
If ask for credentials use **root** and password: **passw0rd**



2.1.3 **Scroll down** and **click** on the hyperlink below to see the **j05mortgageCICS** repository.

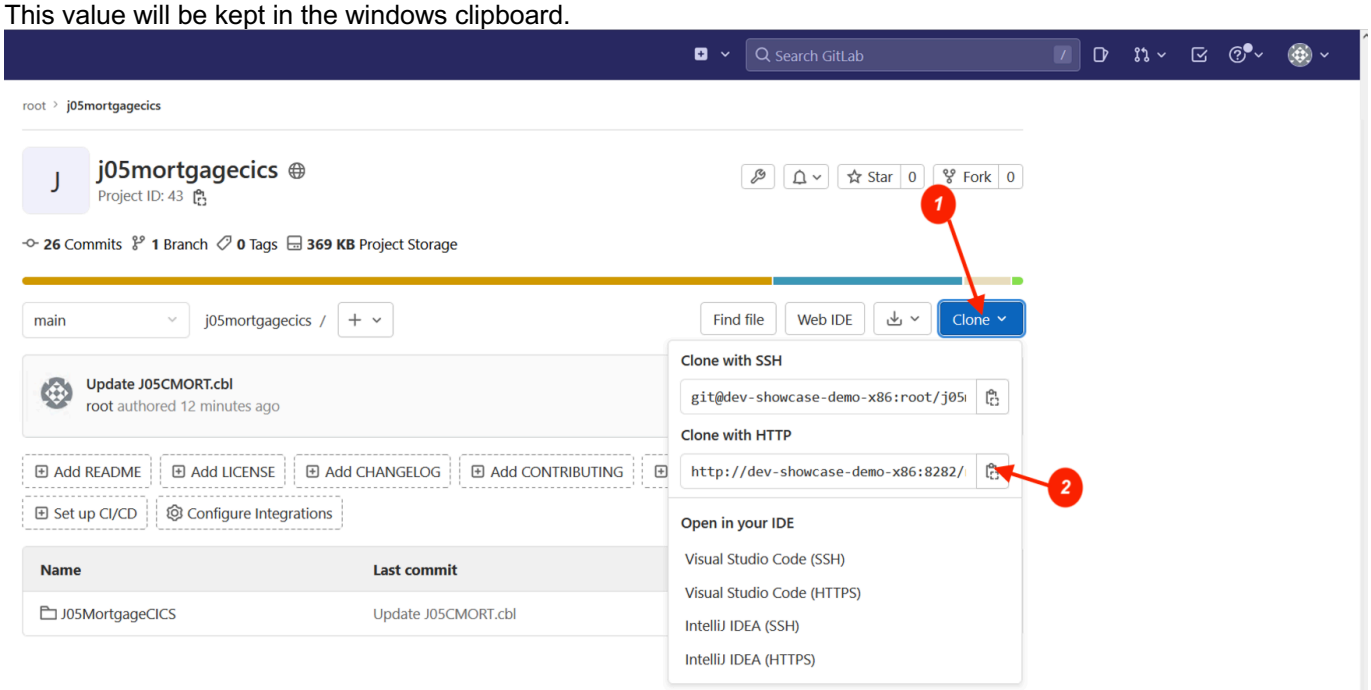


2.1.4 You can see the **J05MortgageCICS** repository.
You may browse the content if you want. The source code to be used on this lab is there

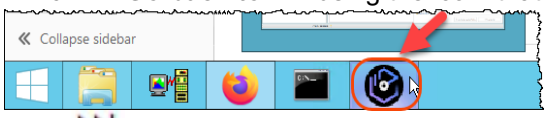


2.1.5 In order to clone it at your window desktop you could use *HTTP* or *SSH.*
Since *SSH* is blocked in our environment we need to use ***HTTP.***

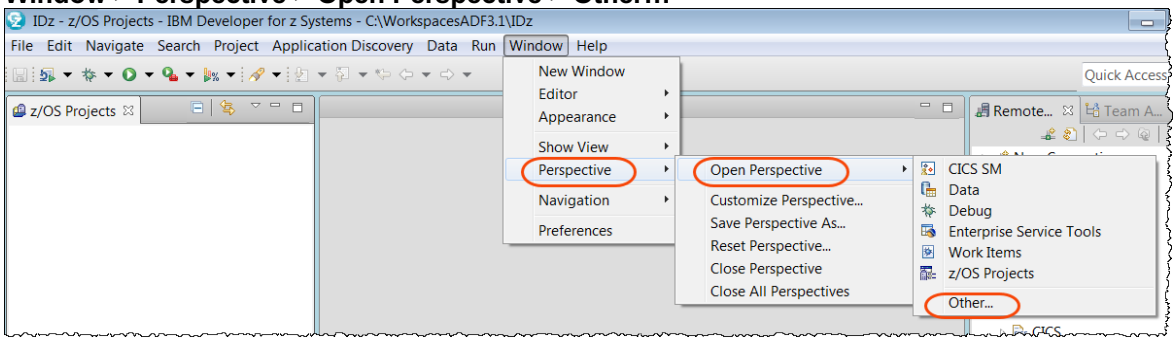
Click on **Clone** (the blue button) and in the icon to **copy the URL**
This value will be kept in the windows clipboard.



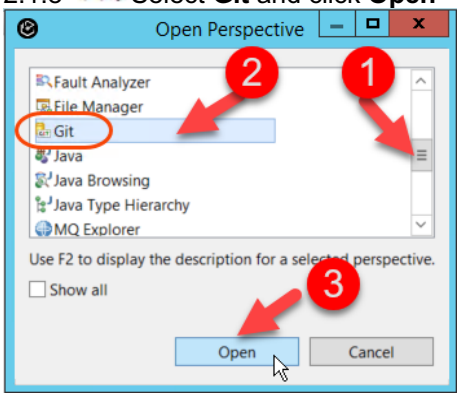
2.1.6 **Go back to IDz** using the icon that is on the base of your screen:



2.1.7 Open the **Git** perspective by selecting
Window > Perspective > Open Perspective > Other…

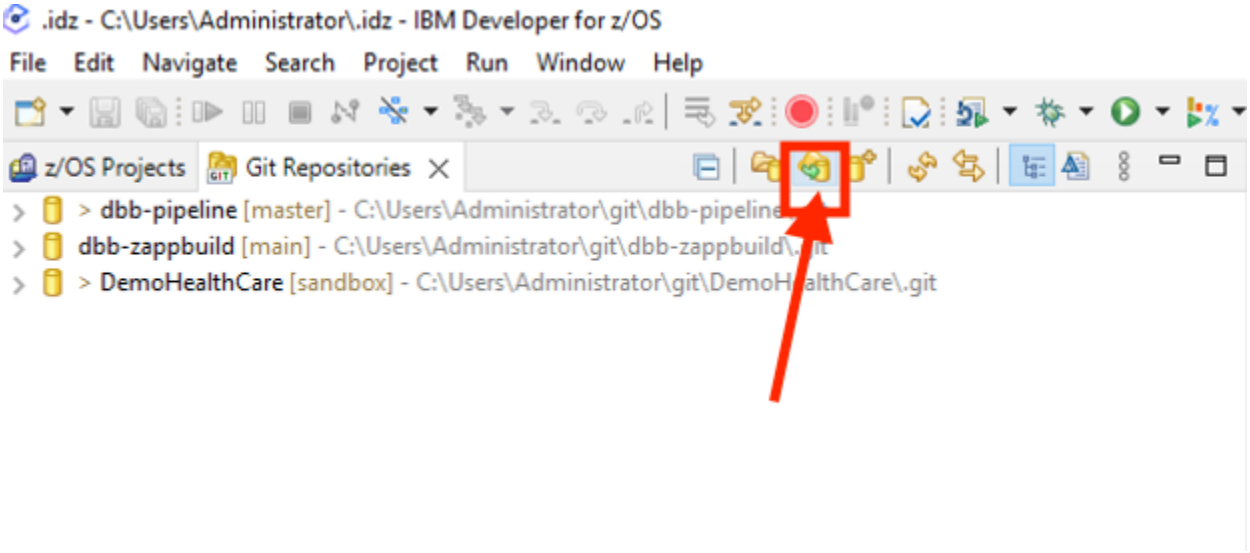


2.1.8 Select **Git** and click **Open**

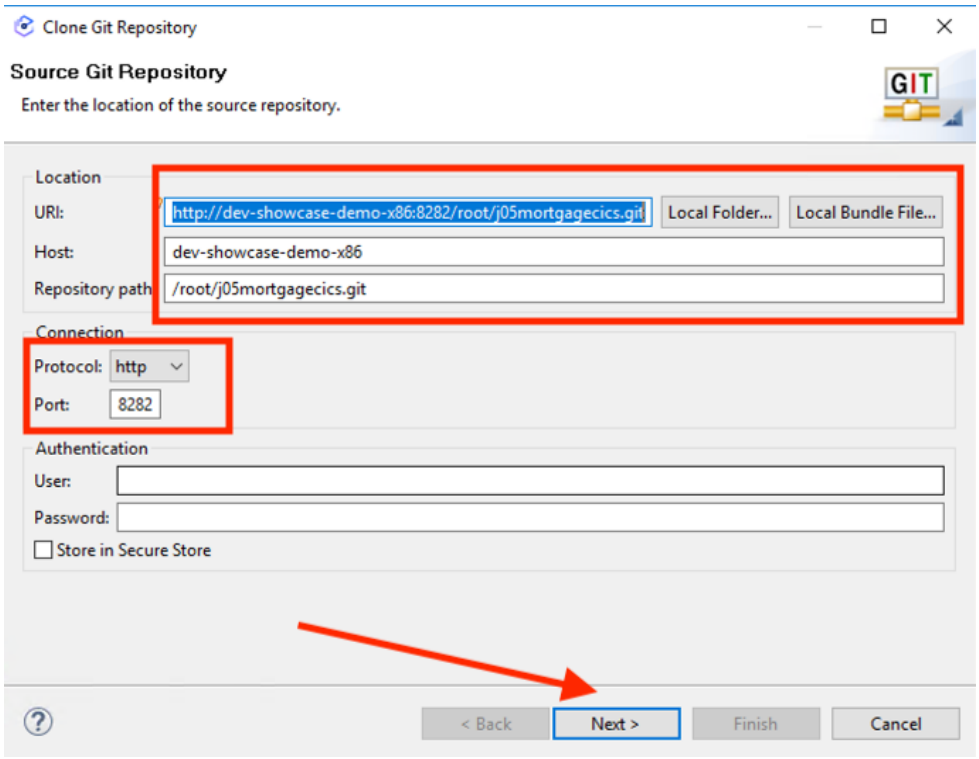


2.1.9 In the *Git Repositories* tab, click on the hyperlink to '**Clone a Git repository'**.

You might see Git Repositories with a list of projects. Click on the Icon **Clone a Git Repository …**

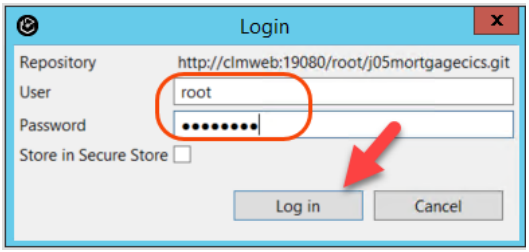


2.1.10 The values copied from the web page (copy URL) will be shown in the Clone Git Repository.
Tip: In case you don't see it, got back to the page and copy it again (steps 2.1.1-2.1.5 above).
Click **Next**



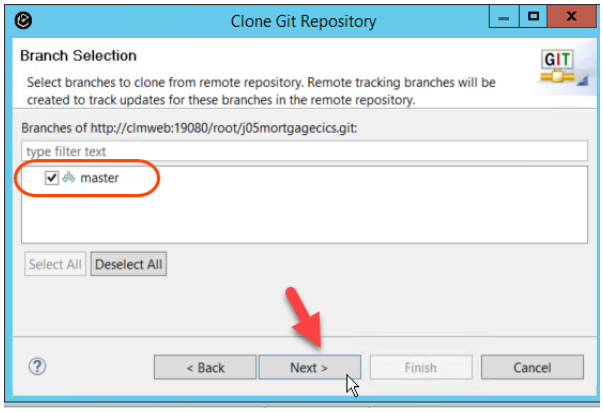
2.1.11 You might need to login using the credentials are a use user: **root** and password **passw0rd (0 is zero) . Otherwise skip to the next step.**

⏭ Click **Log in**



2.1.12 We have only one branch named master.

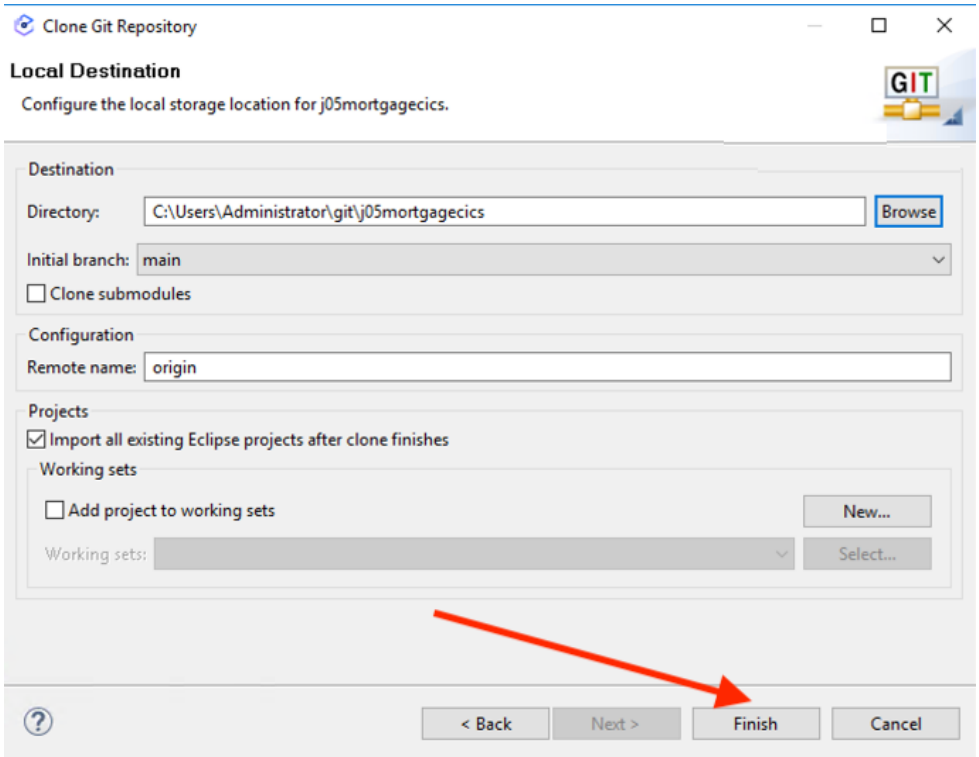
⏭ Click **Next**



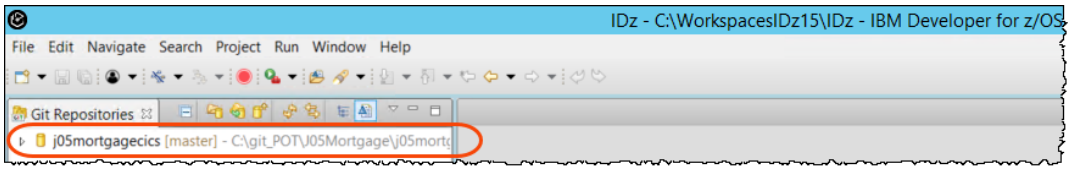
2.1.13 The *Directory* now should point to **`C:\User\Administrator\.git\j05mortgagecics`**

⏭ Be sure that you selected **Import all existing Eclipse projects after clone finishes**

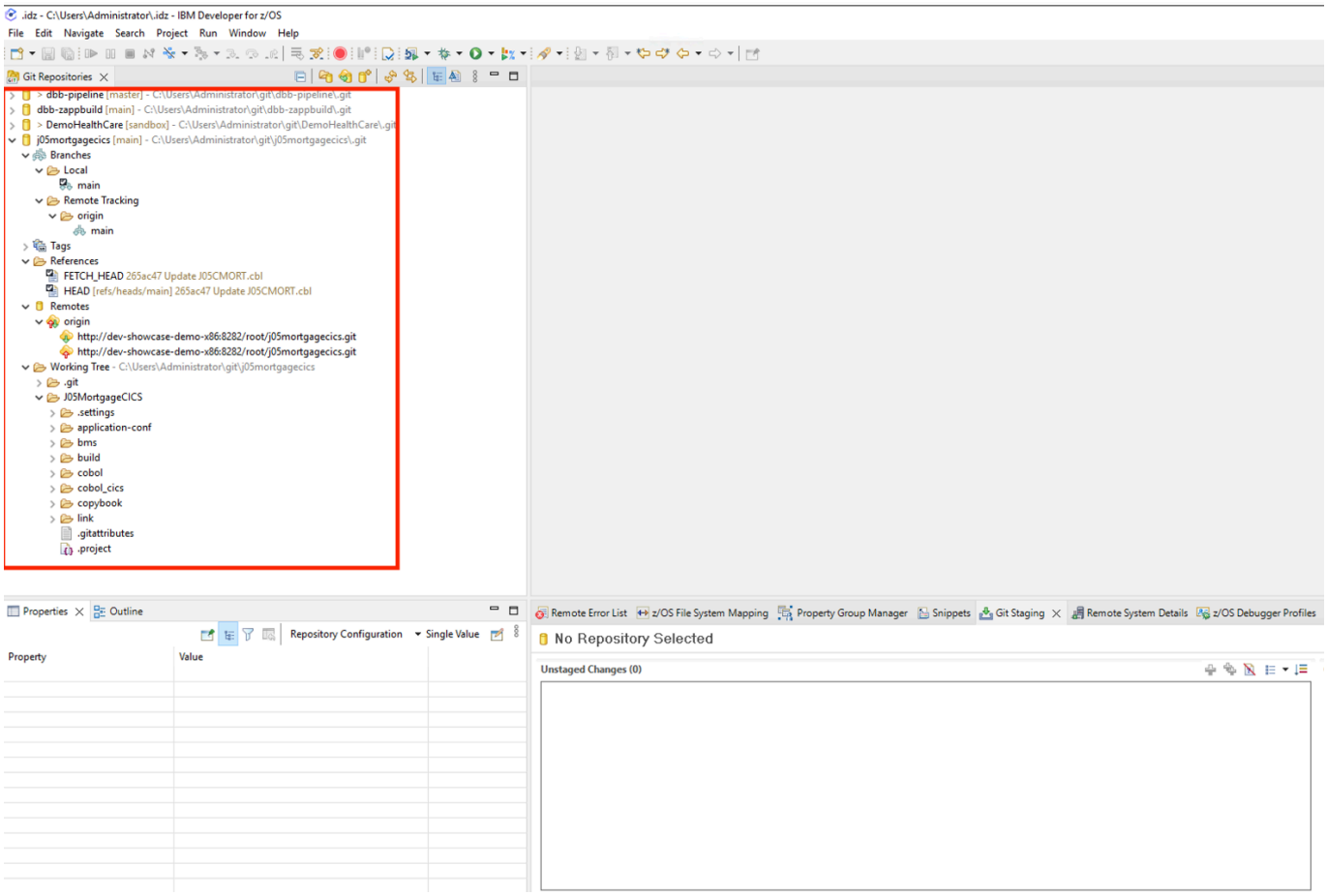
⏭ Click **Finish**



2.1.14 The Mortgage Application will be cloned from the Remote master repository and will appear in the *Git Repositories* view.



2.1.15 **Expand the nodes** by left clicking on the icon as shown below:

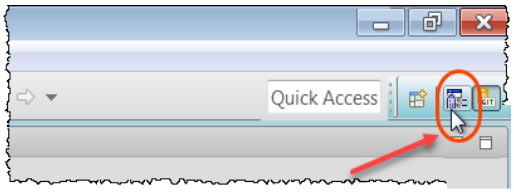


___ 2.2 Verify the code cloned using z/OS projects perspective

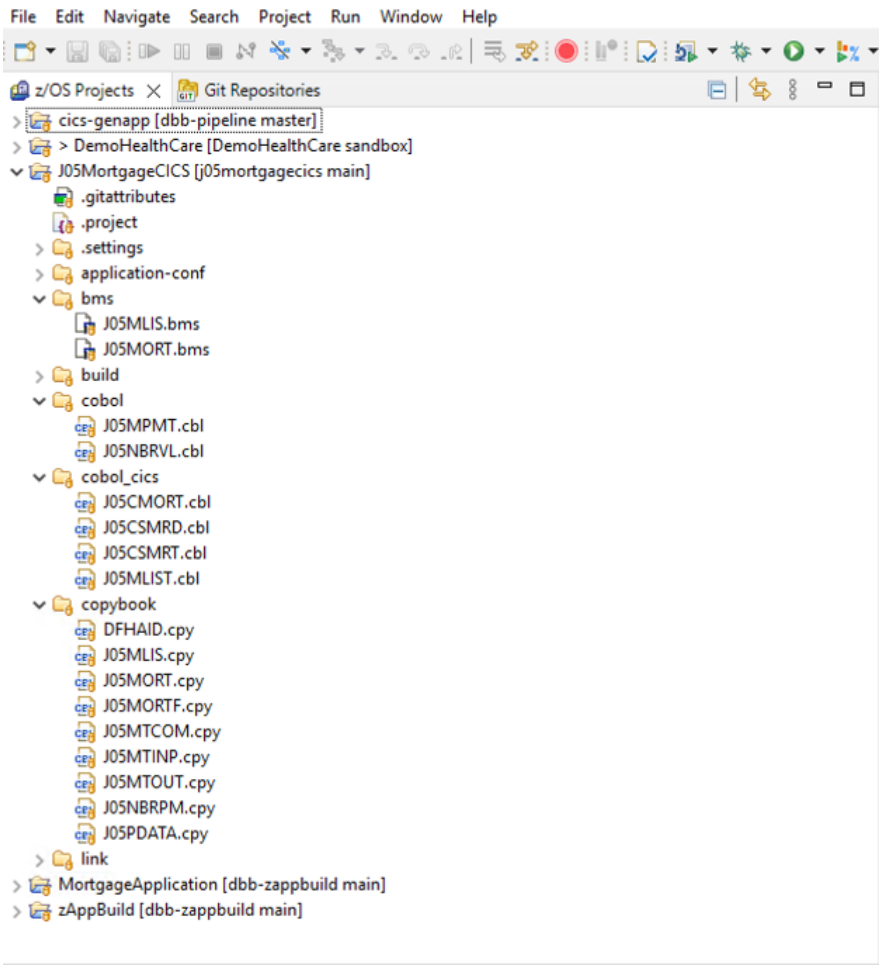
The z/OS projects perspective is the IDz perspective that developers use to work with the source code.

Notice that you have cloned the project at your local workstation but later you will need to connect to z/OS to be able to compile and build your programs.

2.2.1 Switch to the **z/OS Projects** perspective clicking on icon on the top right corner



2.2.2 Expand the project **J05MortgageCICS** clicking on icon and see the source code loaded
Notice that IDz knows that this code is under a repository and the yellow decorator on the icon indicates that.



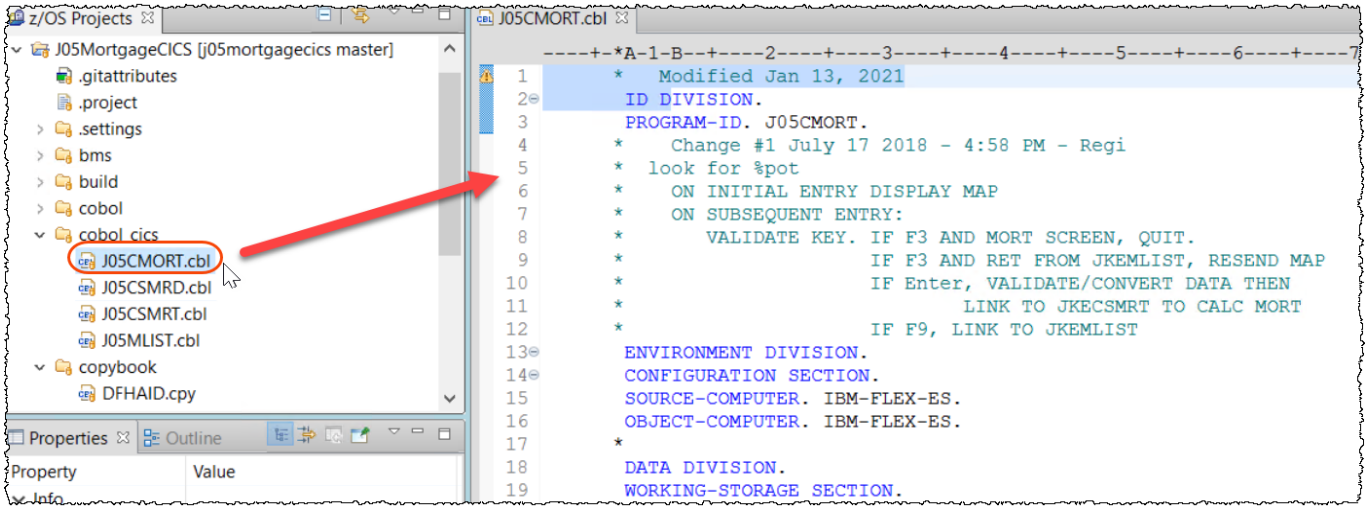
Section 3. Modify the COBOL code using IDz.

Using IDz you will modify the COBOL to have a different message in a CICS dialog.
You will replace the message "*INVALID KEY PRESSED*" when pressing F1 by another message:
"*YYYYMMDD - INVALID KEY PRESSED*".

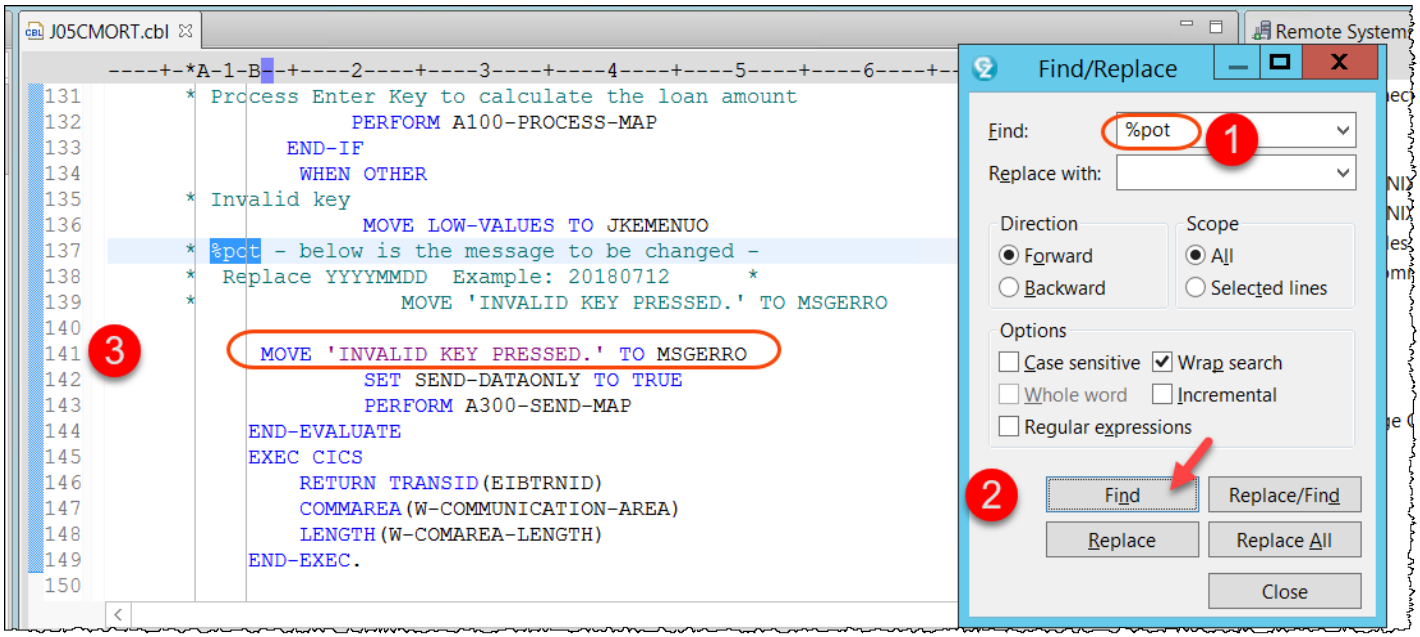
___ 3.1 Edit and modify the code that send the message

The COBOL code to be modified is the program ***J05CMORT*** that is under the folder ***cobol_cics***.

3.1.1 ⏭ Using *z/OS Projects* view double click **J05CMORT** under **cobol_cics**
This is the program that you will update



3.1.2 Use **Ctrl + f** to find the "**%pot**" on line **137 (second Find hit)**. You will modify the line **141**..



3.1.3 **Close** the find dialog and modify the line 141

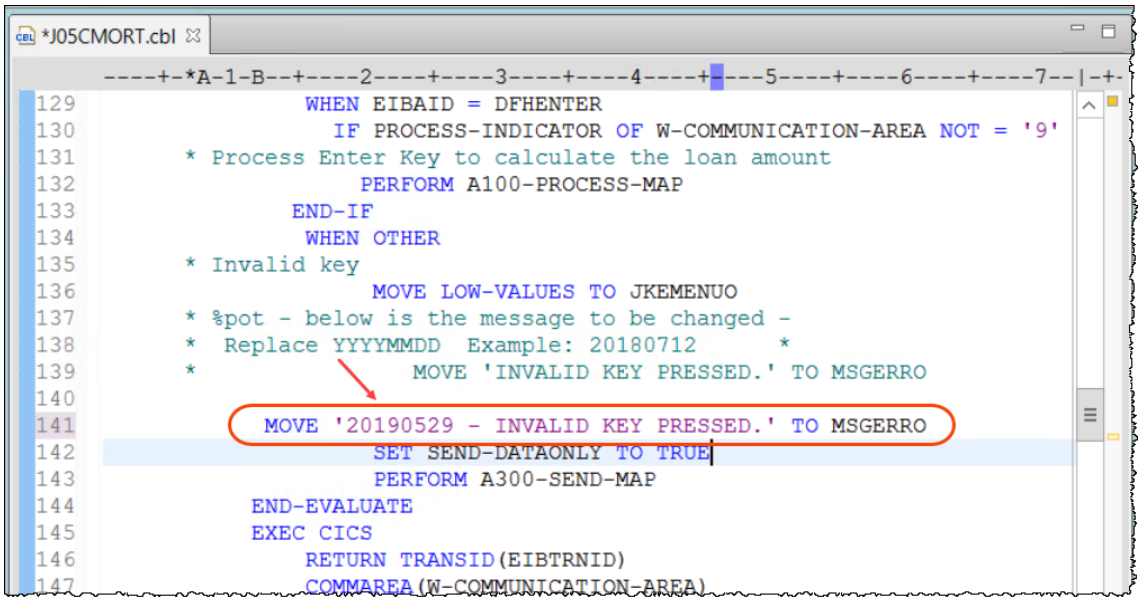
From

```
MOVE 'INVALID KEY PRESSED.' TO MSGERRO
```

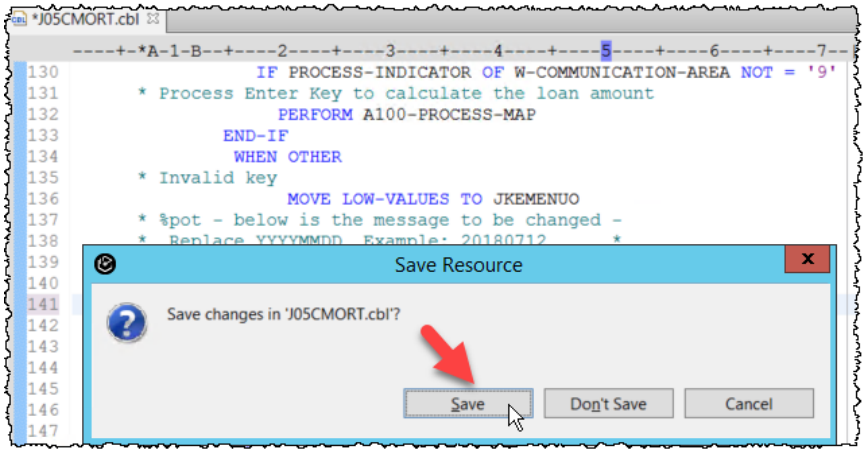
To

```
MOVE 'YYYYMMDD - INVALID KEY PRESSED.' TO MSGERRO
```

Where **YYYYYMMDD** is the today's date.. Example, I changed to **20190529..**



3.1.4 Use **Ctrl + Shift + F4** to close all code being edited. Click **Save** to save the modified code.

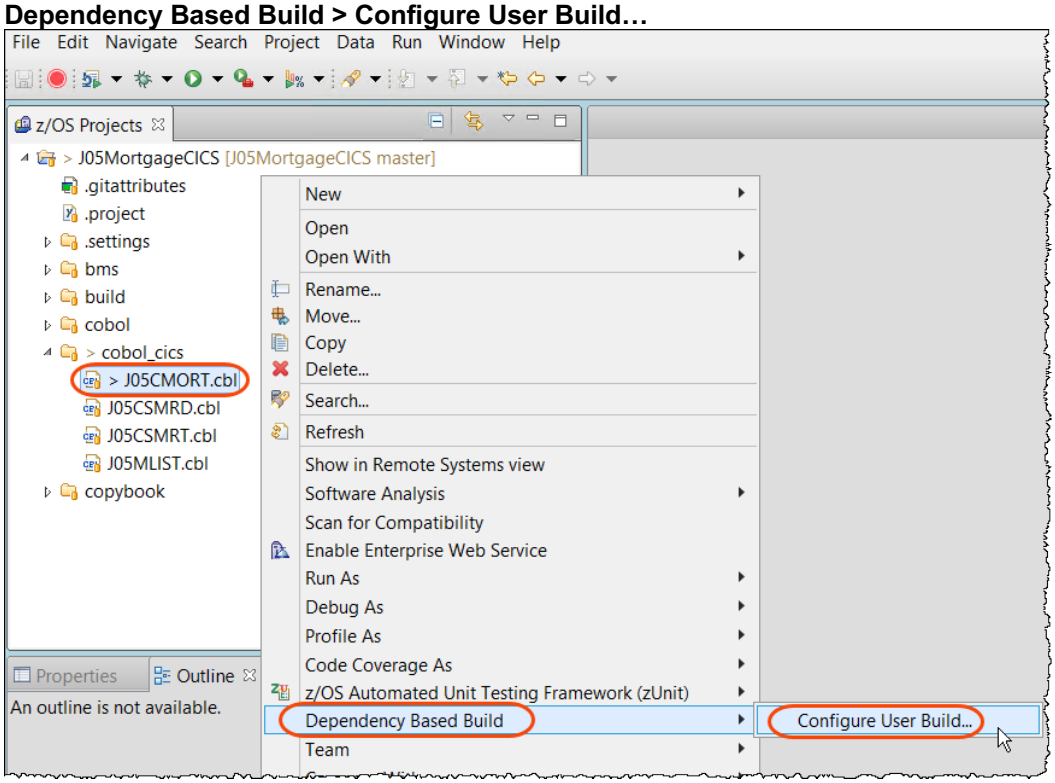


Section 4. Use IDz *DBB User Build* to compile/bind and perform personal tests.

You will compile and link the modified code using the DBB User Build function. When complete you will run the code using CICS for a personal test and verify that the change is correctly implemented.

___ 4.1 Using Dependency Based Building option

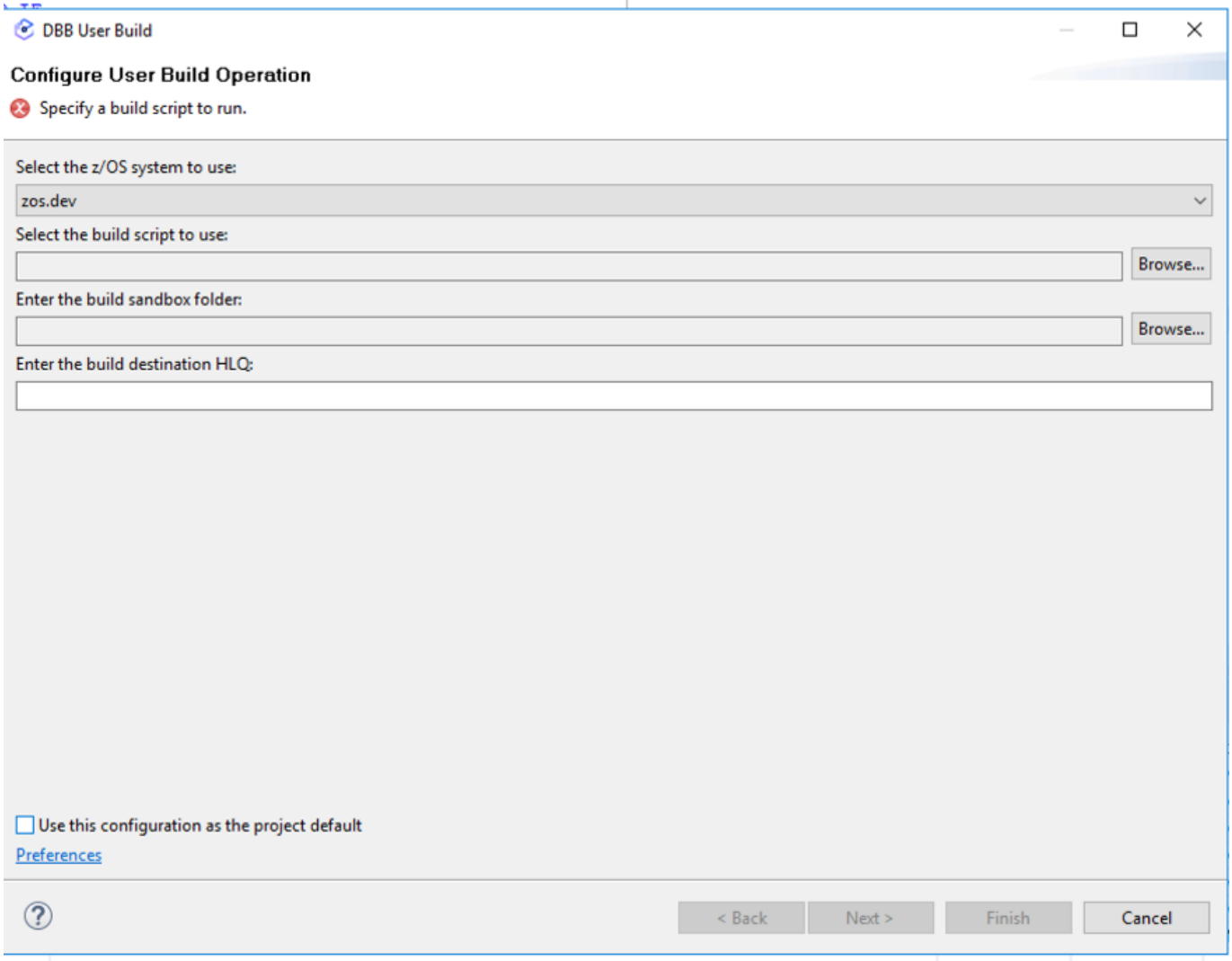
4.1.1 Under *z/OS Projects* view right click **J05CMORT** and select **Dependency Based Build > Configure User Build…**



4.1.2 Be sure that those values are already assigned for the build, otherwise use the **Browse** button and select the values as below

z/OS system: **`zos.dev`**

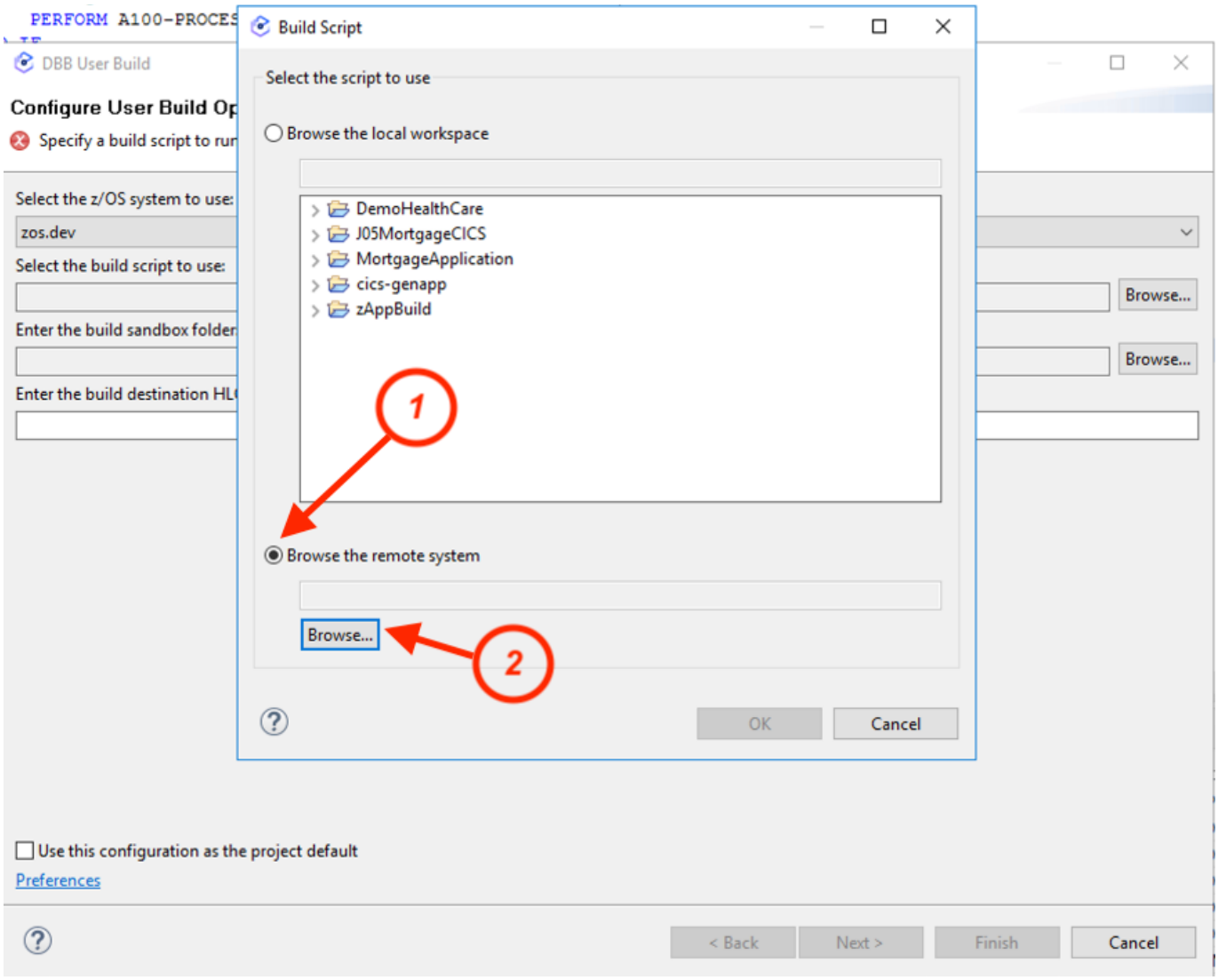
```
   END-EXEC
IEN EIBAID = DFHENTER
 IF PROCESS-INDICATOR OF W-COMMUNICATION-AREA NOT = '9'
:nter Key to calculate the loan amount
   PERFORM A100-PROCESS-MAP
```



Build script : **`We will use the latest version of build.groovy`** (use **Browse** the **local workspace**)

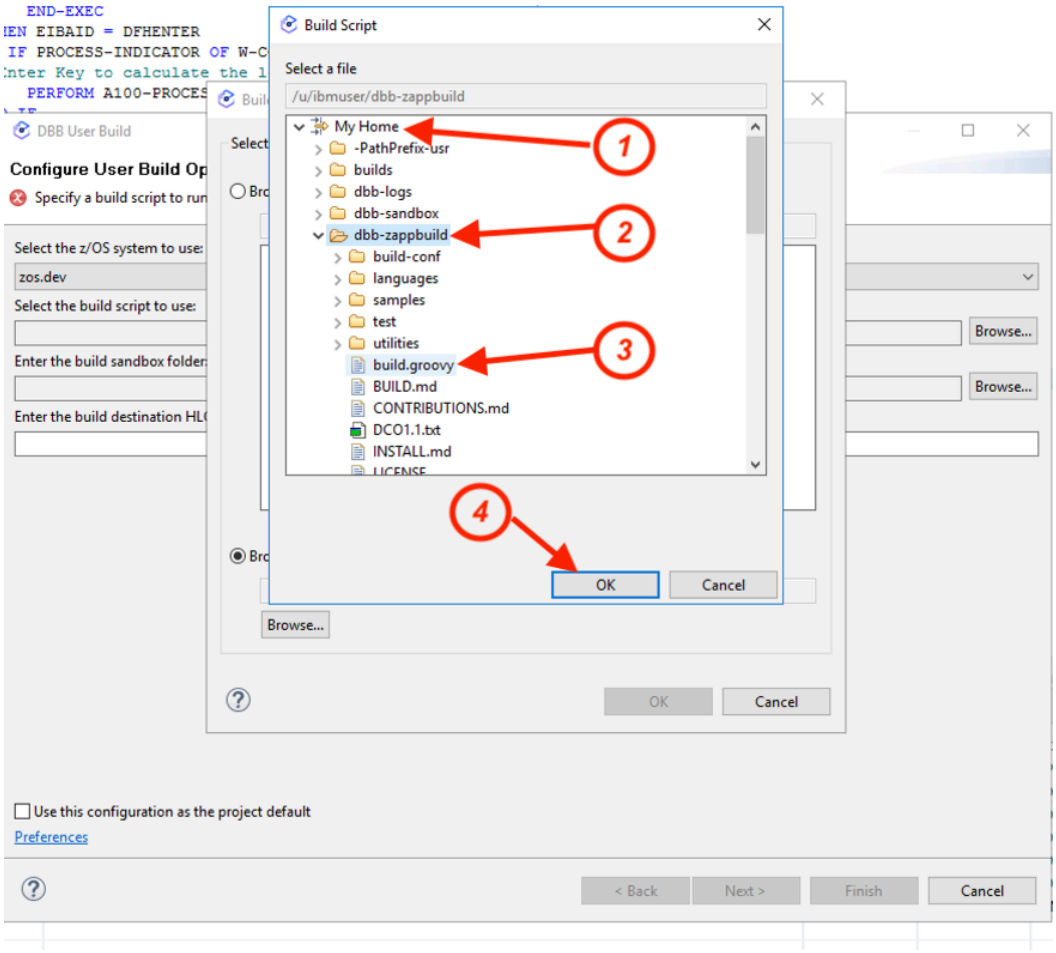
Click on Browse button on DBB User Build Window

1. On Build Script Window. Check **Browse the remote system**
2. Click **Browse** Button

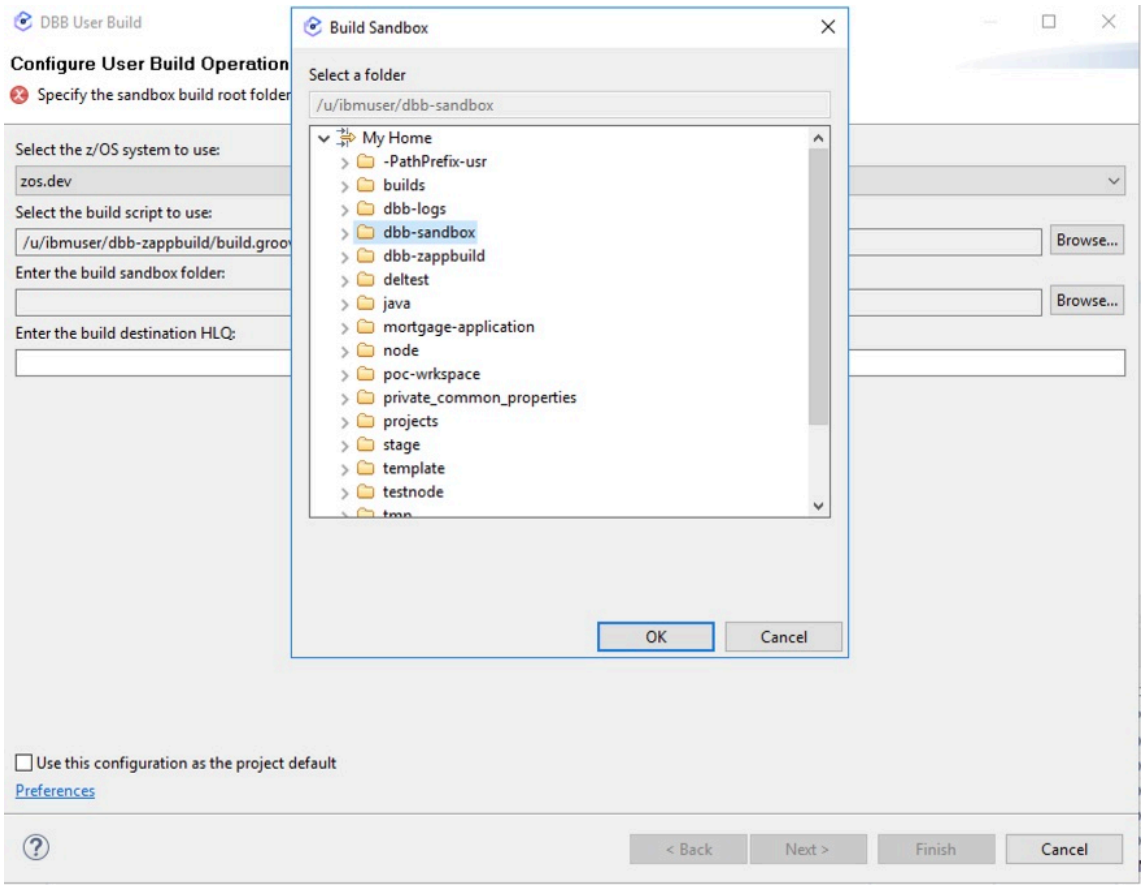


On Build Script Window:

1. Select **My Home** folder
2. Select **dbb-zappbuild** folder
3. Click **build.groovy**
4. Click **Ok**

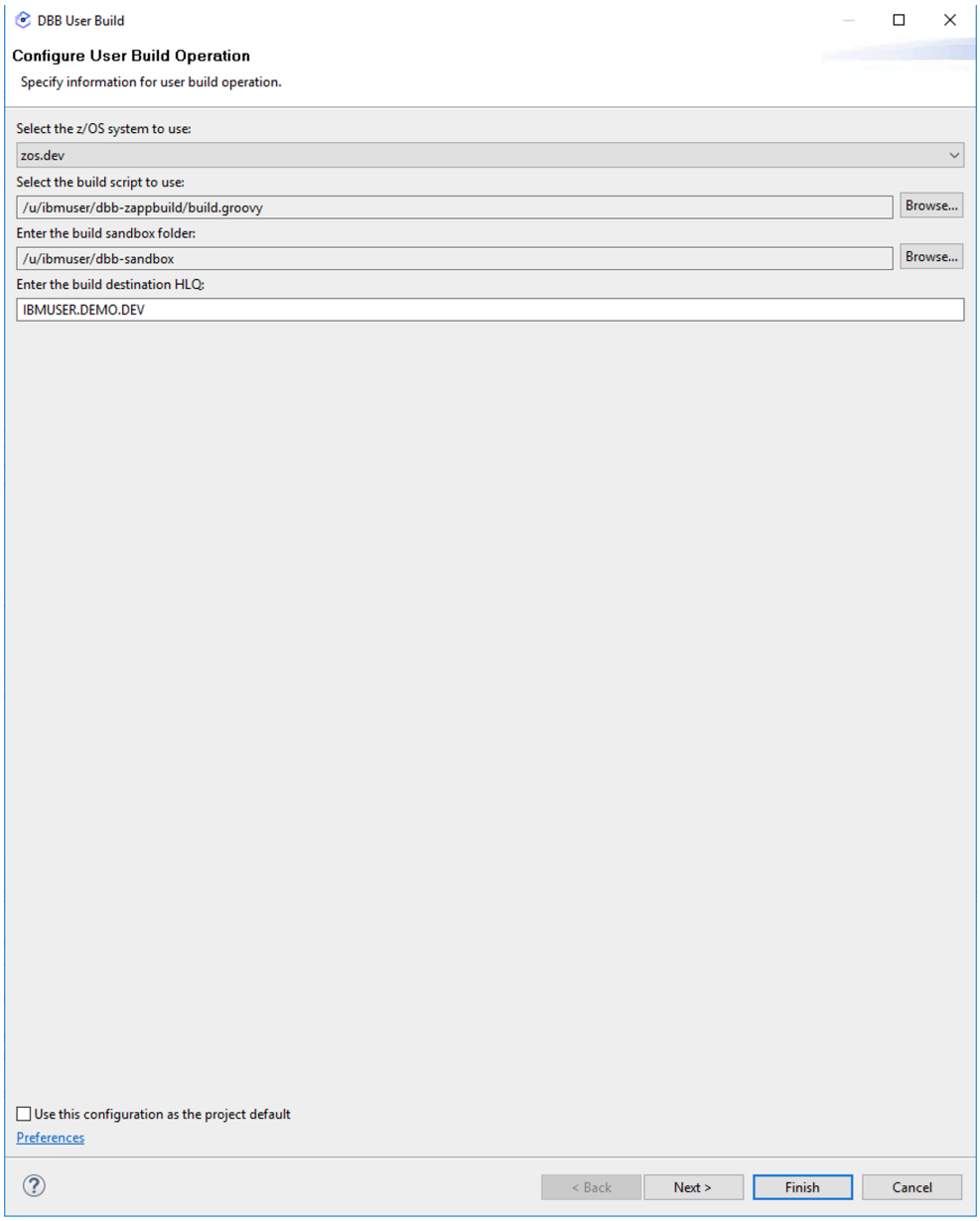


Build sandbox : **`Select the sandbox folder on remote server`** (Use **Browse,** expand **MyHome,** click **dbb-sandbox**)



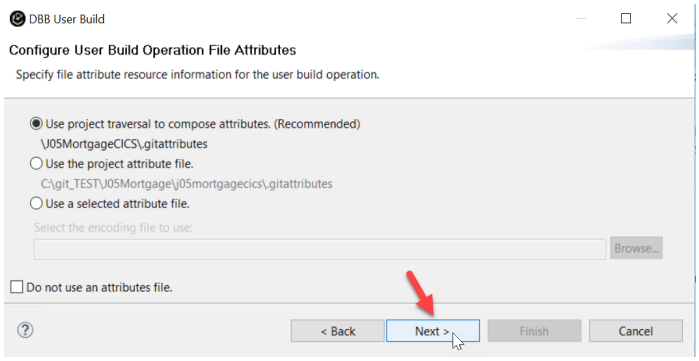
Build destination HLQ: **`IBMUSER.DEMO.DEV`** (must type)

Click **Next**

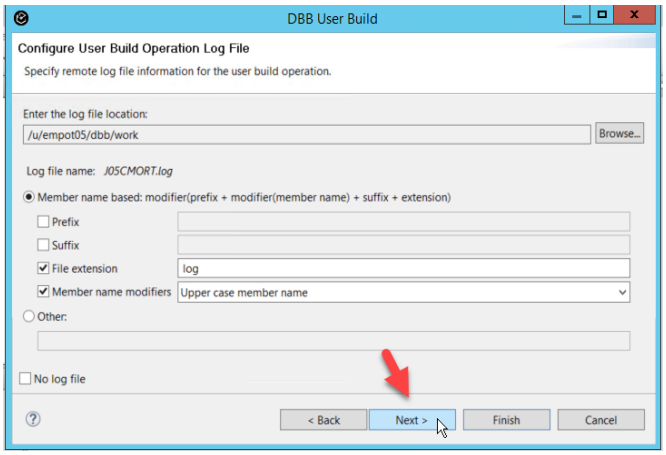


4.1.3 On *Configure User Build Operation File Attributes* click **Next**

The *.gitattributes* have the information to translate from *UTF-8* to *EBCIDIC* when moving the code to z/OS.

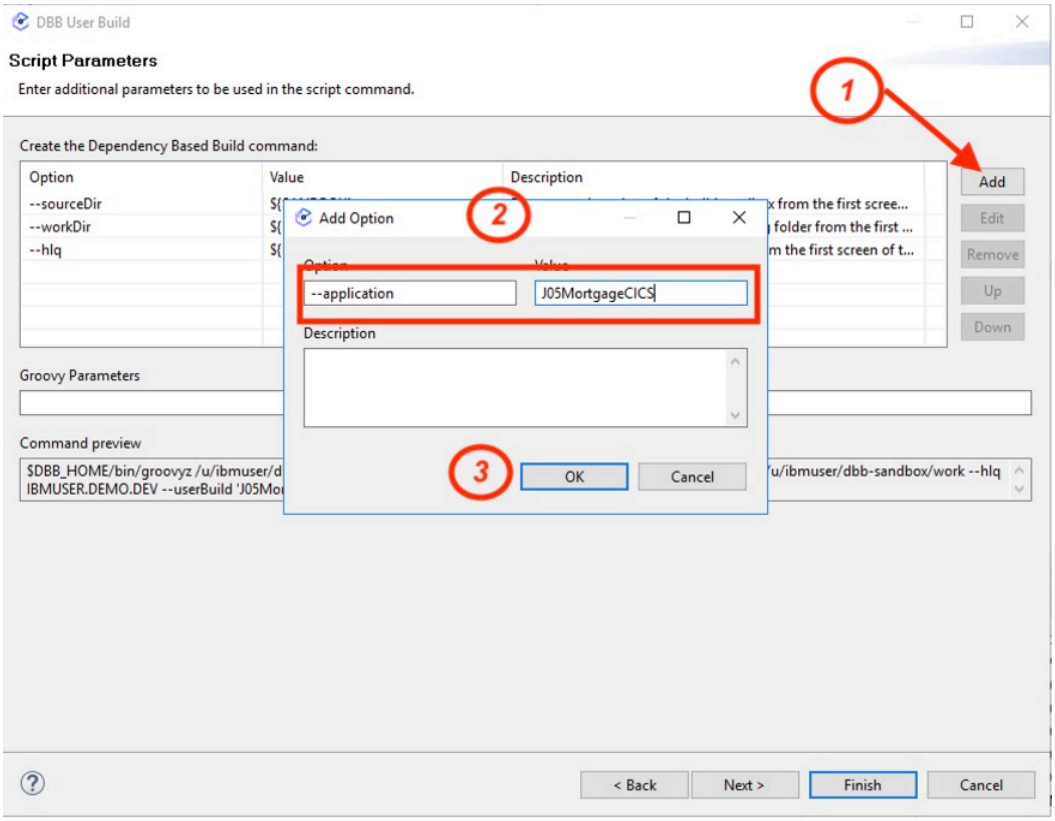


4.1.4 On *Configure User Build Operation Log File* click **Next**
You will use the default values.



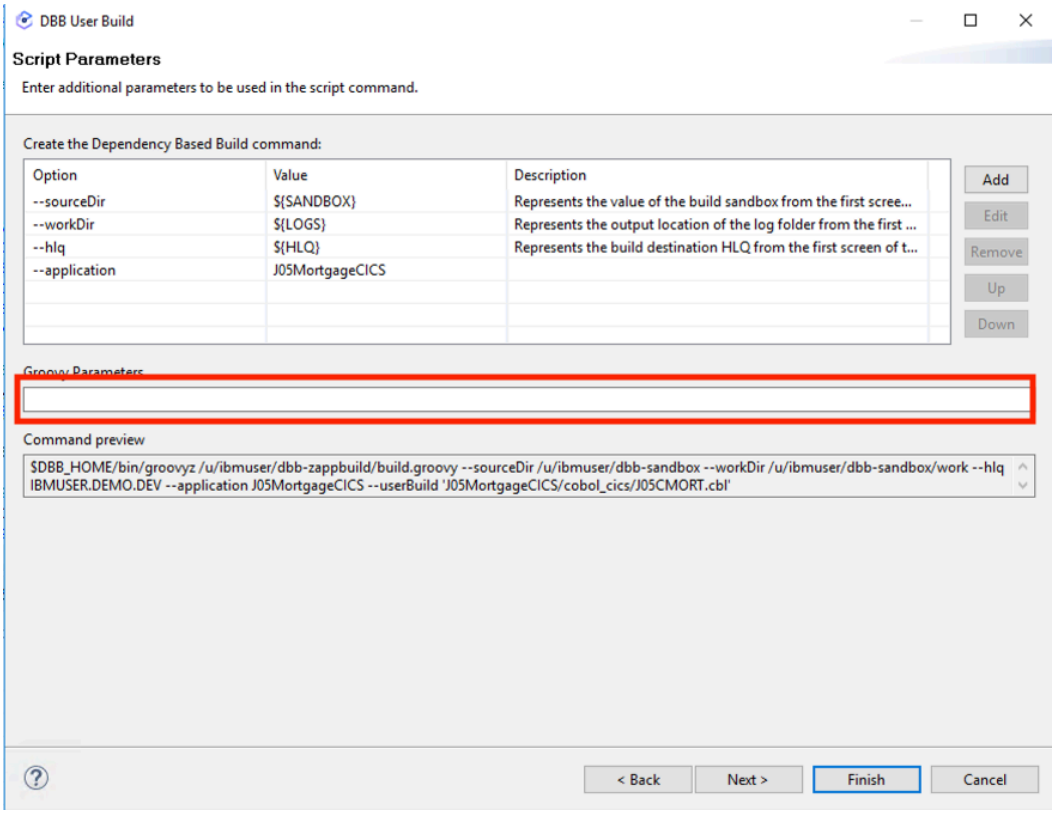
4.1.5 On Script parameters:

1. Click Add Button
2. Enter on Option **: --application**
3. Enter Value: **J05MortgageCICS**



Be sure that the field *Groovy Parameters* is empty and click **Next**

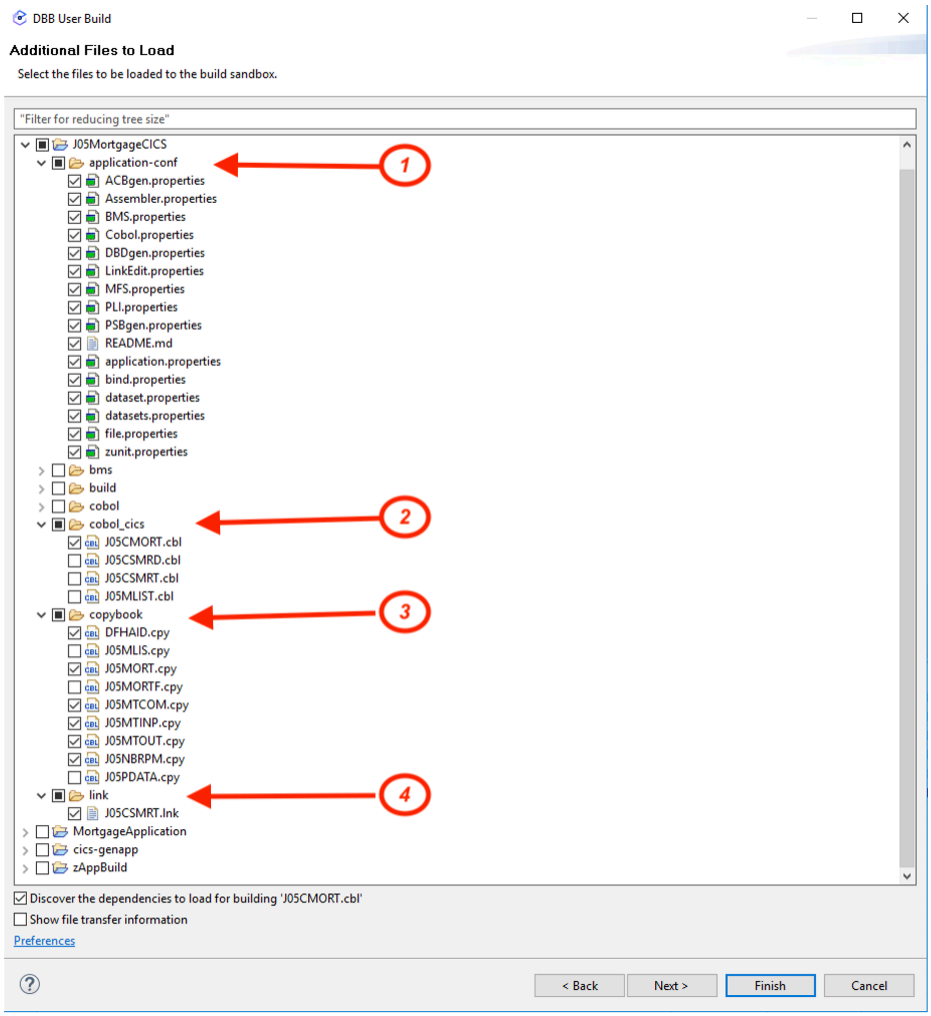
Notice the preview of the commands that will be execute at the z/OS by the DBB toolkit.



4.1.6 The selected files will be moved from your local workspace to a z/OS USS directory and the execution of the groovy scripts will interact with the DBB framework that will invoke the compiler, linkage editor, etc..

⏭ Expand folders **J05Mortgage CICS** and the ***cobol_cics*** and ***copybook*** directories

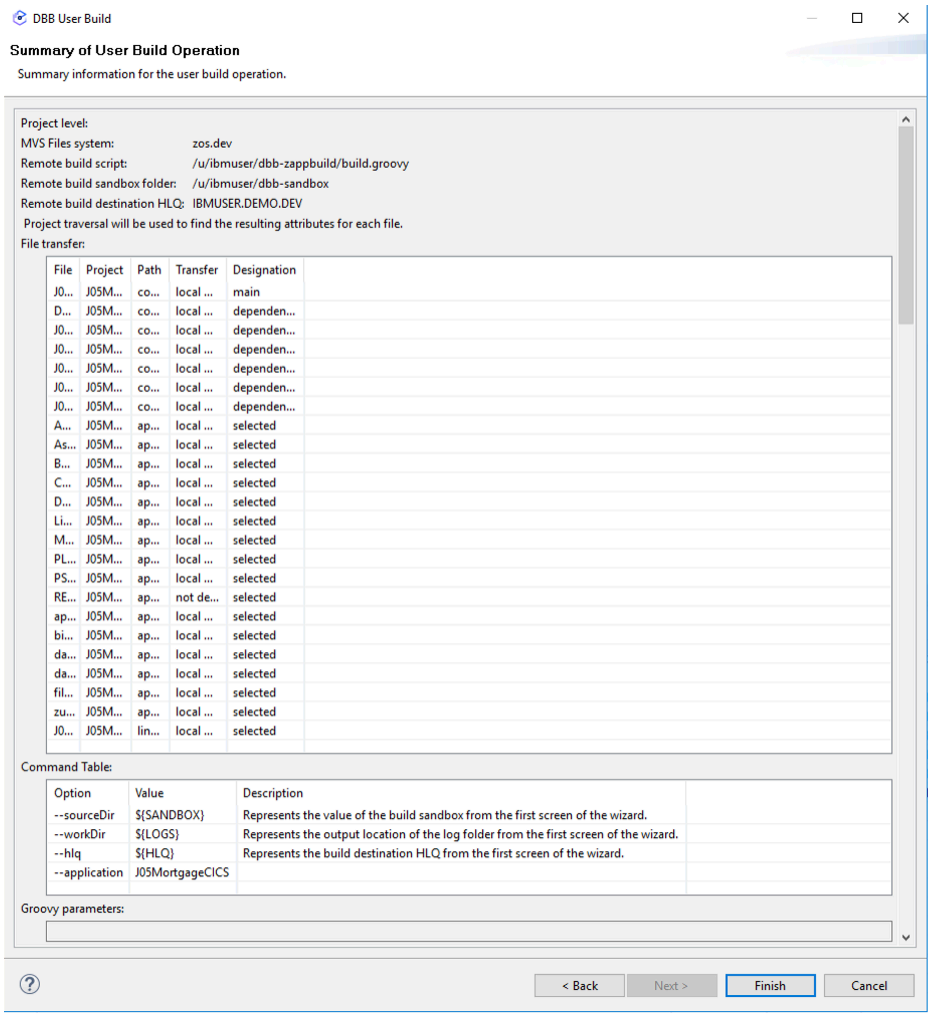
1. Check **application-conf** on J05MortgageCICS folder
2. Check J05CMORT.cbl on **cobol_cics** folder
3. Verify **copybook** folder (DBB checks all copybooks necessaries)
4. Check J05CSMRT.lnk on **link** folder



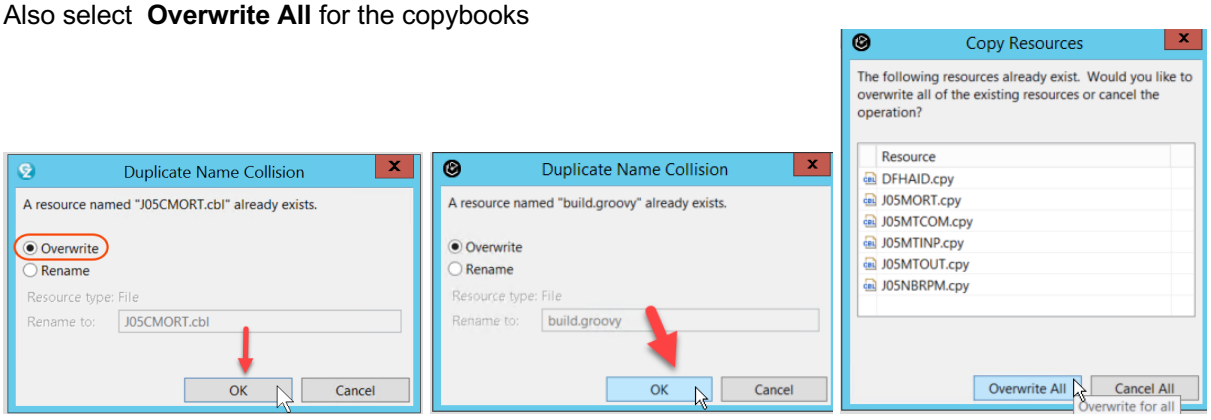
⏭ Since you will also need the related copybooks for a clean compile be sure that **"Discover the dependencies to load for building J05CMORT.cbl "** is selected (automatically checked).

⏭ Click **Next**

4.1.7 ⏭ Verify the summary of the User Build Operation and click **Finish**



4.1.8 If the dialogs below pops up select **Overwrite** and click **OK**
Also select **Overwrite All** for the copybooks

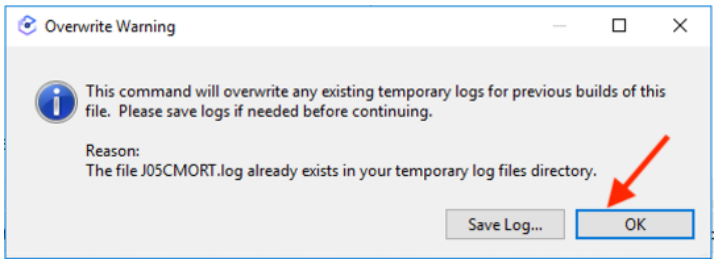


The DBB User build will be running and the messages will be shown at the Console view..
Click on **Console** tab on lower right corner The execution will start, and this will take a while .

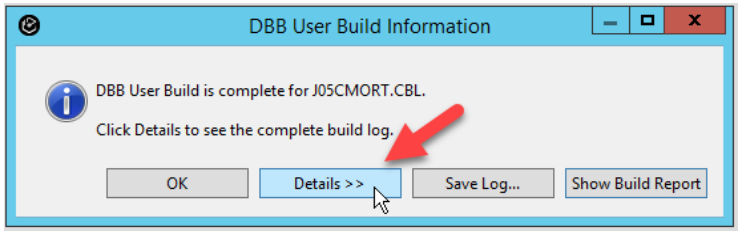
4.1.9 When complete you will have the message below:

On Overwriting Warning.

Click **OK**



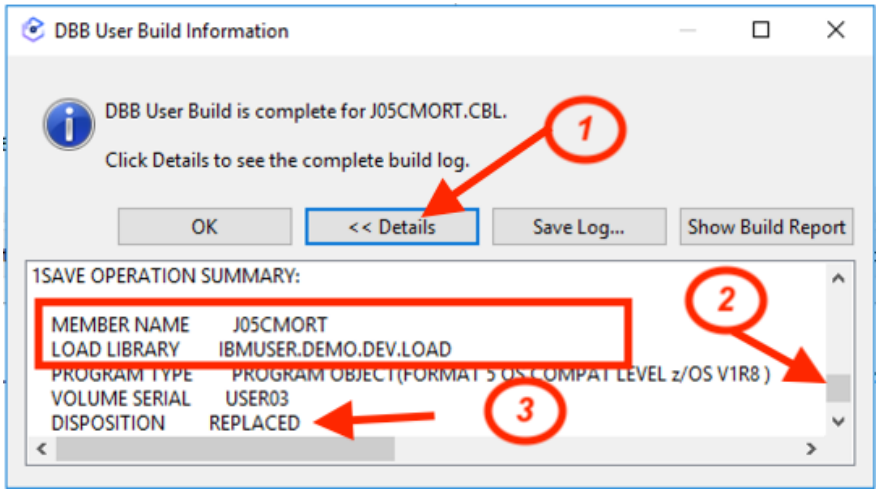
⏭ Click **Details**



___ 4.2 Verify the DBB User Build results

4.2.1 ⏭ After clicking **Details >>** scroll down and verify that a new load module name **J05CMORT** was replaced on the dataset ***IBMUSER.DEMO.DEV.LOAD***

⏭ Click **OK** to close this dialog.



4.2.2 ⏭ Verify at the Condole view that the Build State should show a *CLEAN* build.

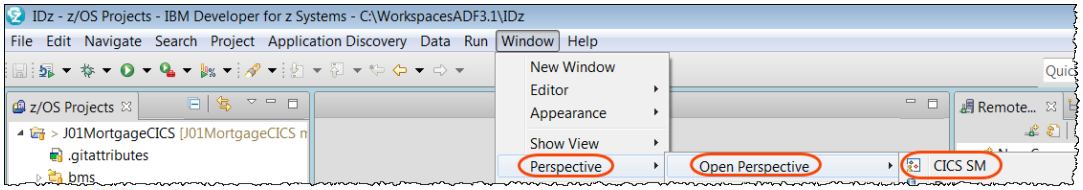
```
** Writing build report to /u/ibmuser/dbb-sandbox/work/BuildReport.html
** Build ended at Tue Jun 18 08:41:42 CDT 2024
** Build State : CLEAN
** Total files processed : 1
** Total build time   : 6.890 seconds

** Build finished

/u/ibmuser/dbb-sandbox>
```

___ 4.3 Issuing a CICS New copy

4.3.1 At this point you may test your new load module. But since this is a CICS system you will need to make a *CICS NEWCOPY*. Let's use IDz and CICS Explorer to do that.

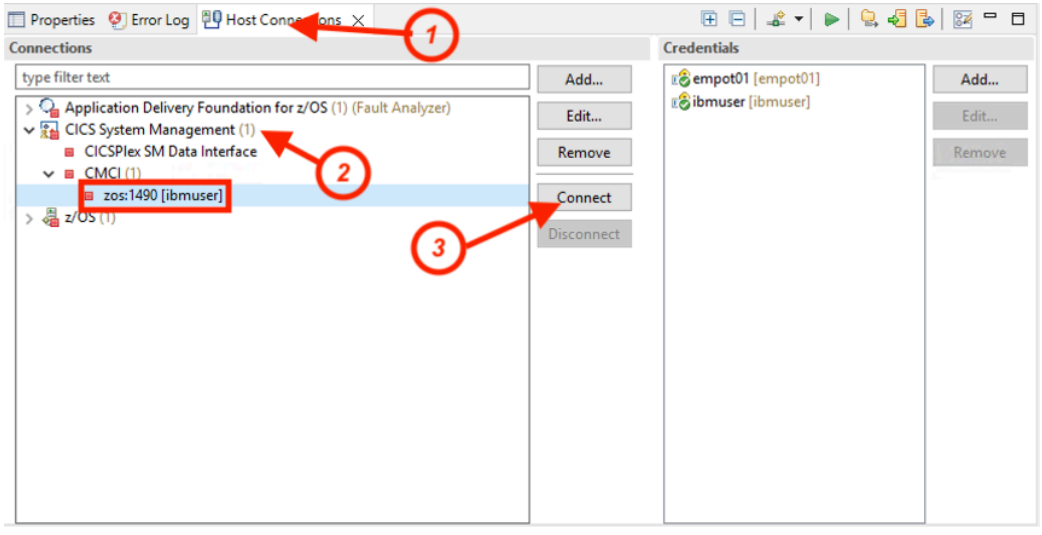
Notice that the new copy could be done by the Groovy scripts, but I want to show CICS Explorer nice capabilities here..

⏭ Open the CICS SM perspective selecting **Window > Perspective > Open Perspective > CICS SM**

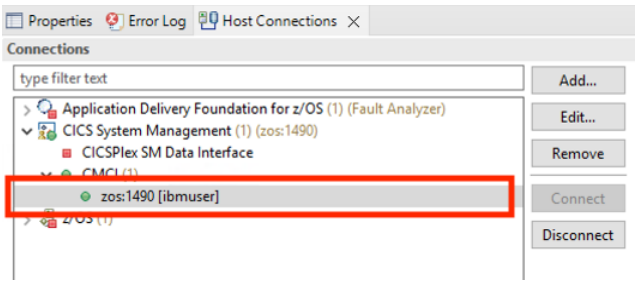


4.3.2 ⏭ Click on tab **Host Connections on the bottom**, expand **the CICS System Management, CMCI**, select **CICSTS56** and click on **Connect**

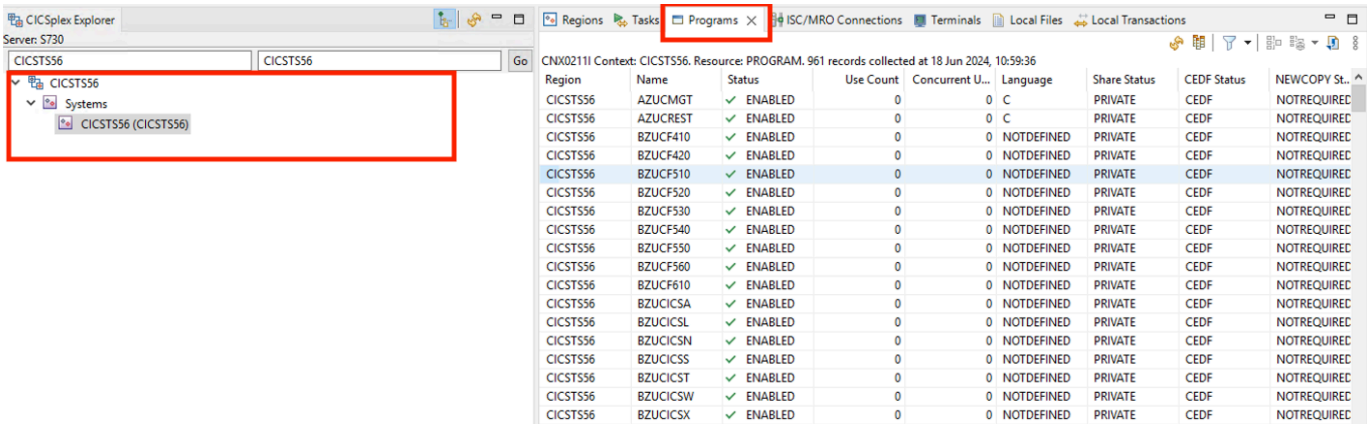
Note: If it ask you for userid and password then use **ibmuser** and **sys1** for username and password



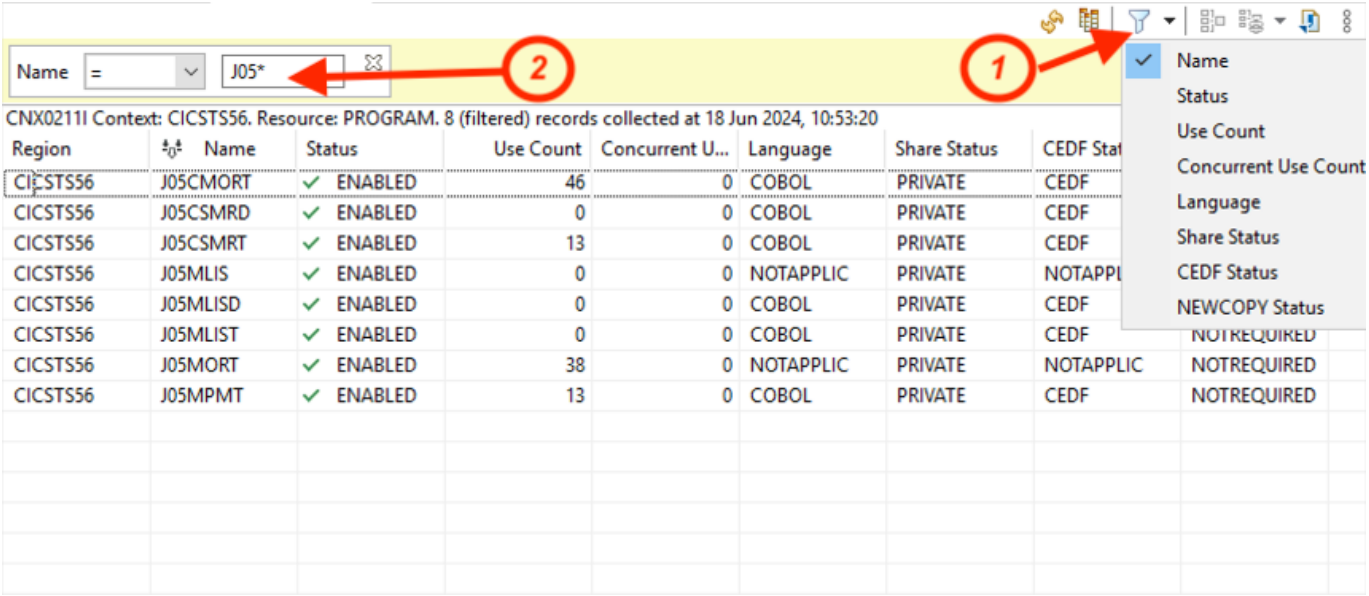
4.3.3 The red dot will turn green once is connected to CICS



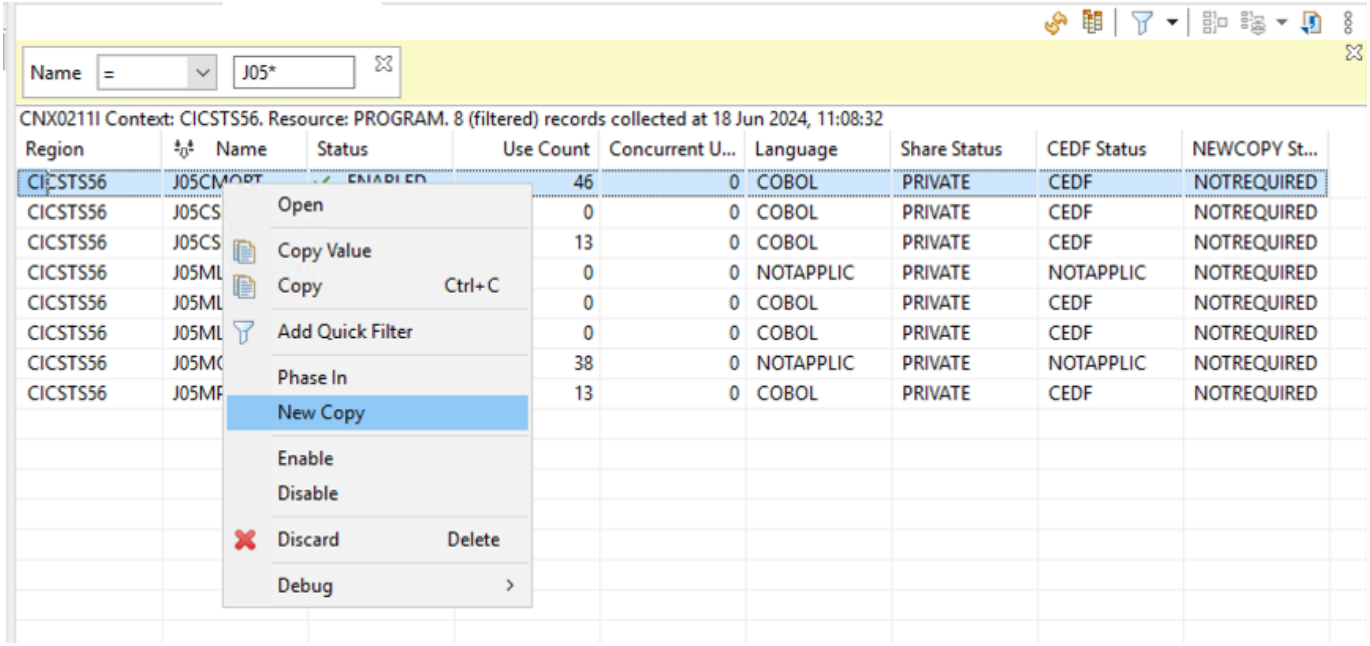
4.3.4 ⏭ On top left expand **CICSTS56** and click on **CICSTS56 (CICSTS56).**



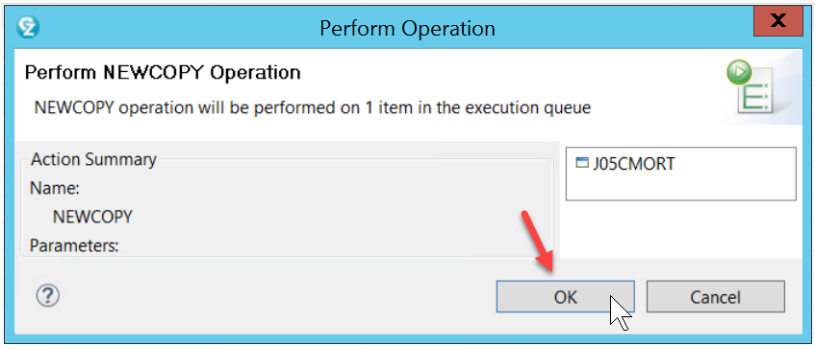
⏭ Click on the **Programs** tab.
Since the filter is already made for " **Name = J05*** " and press **ENTER** you should see the list below:



4.3.5 Right click on **J05CMORT** and select **New Copy** as the example below

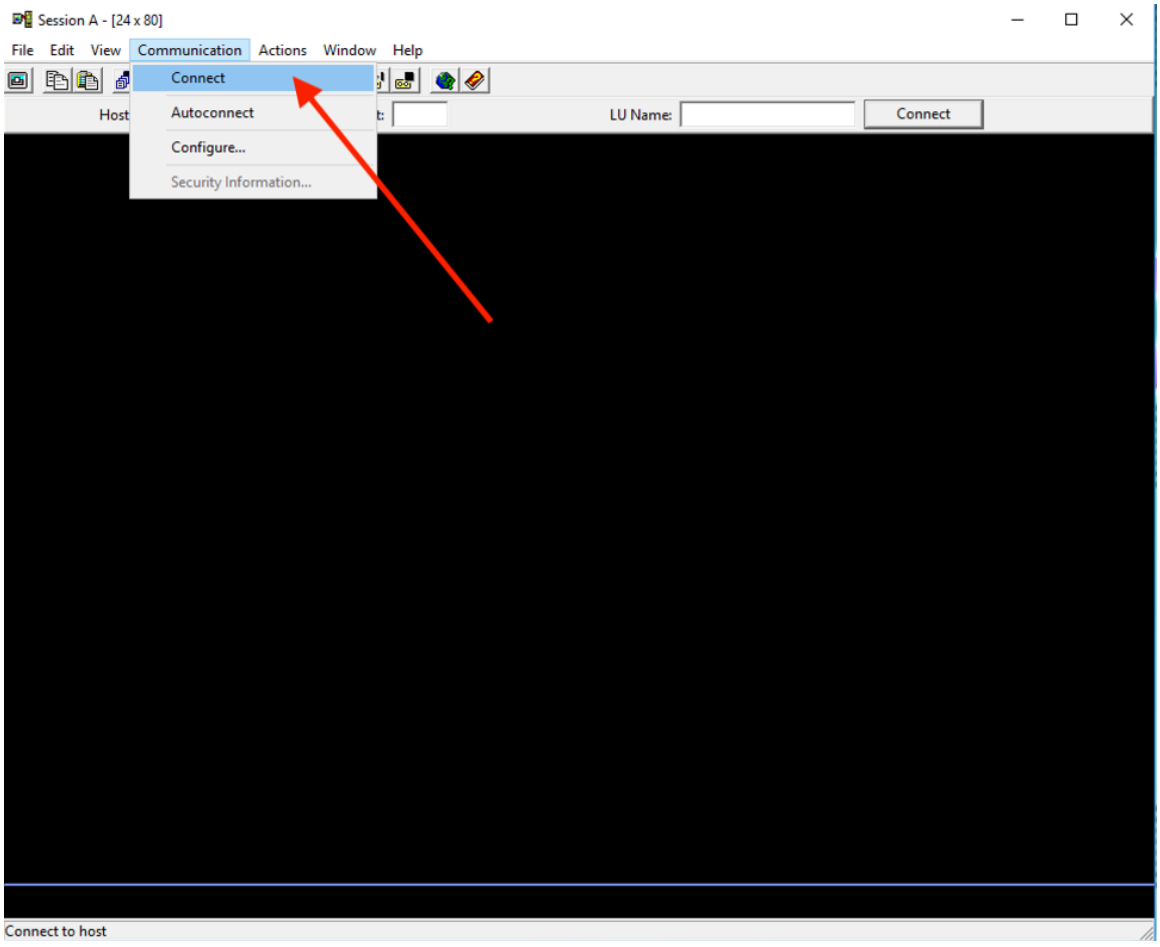


4.3.6 Select J05CMORT and click **OK**

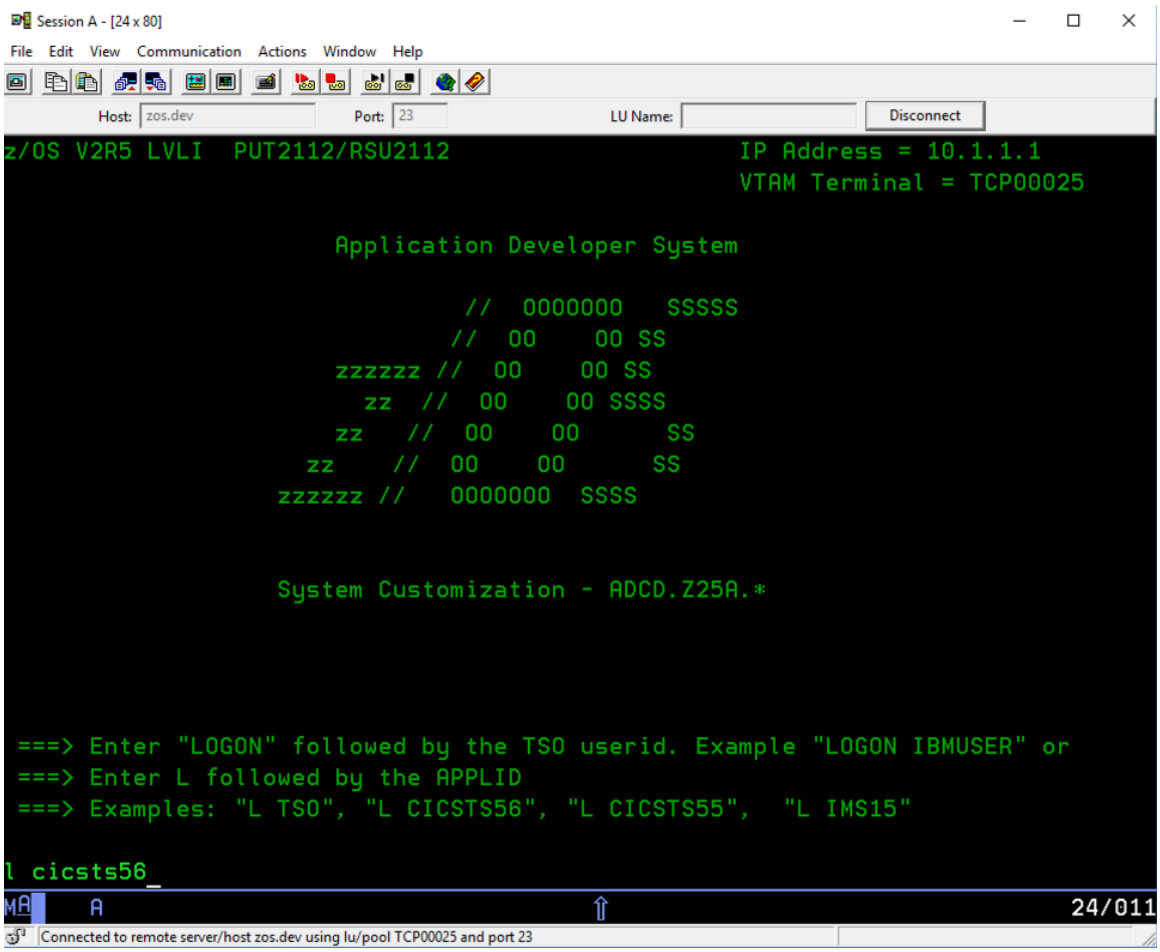


4.3.7 Let's start the 3270 emulation again ..

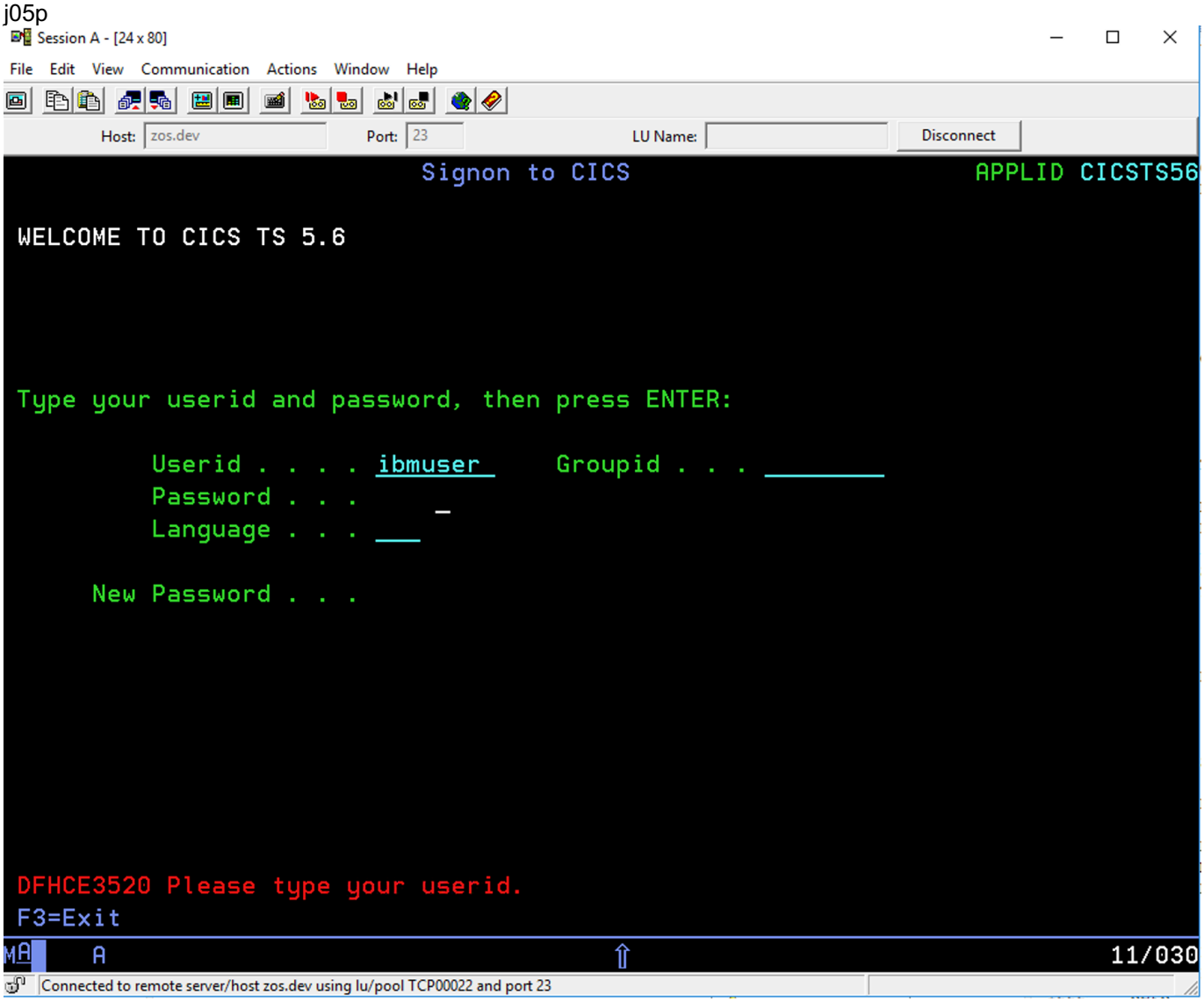
4.3.8 If a black screen shows up . Select Communications → Connect



4.3.9 Type **l cicsts56**. and press **Enter key**.



4.3.11 Logon using **ibmuser** and password **sys1** and press **Enter**.
j05p



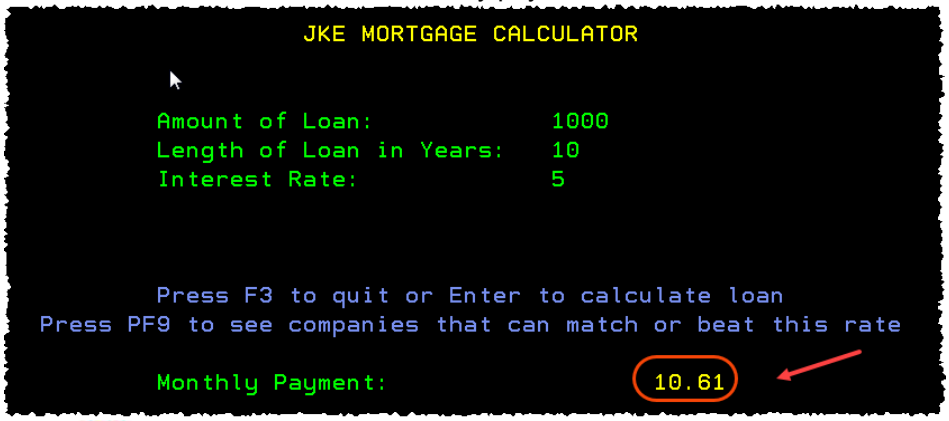
4.3.12 The sign-on message is displayed

___ 4.4 Running *j05p* CICS transaction

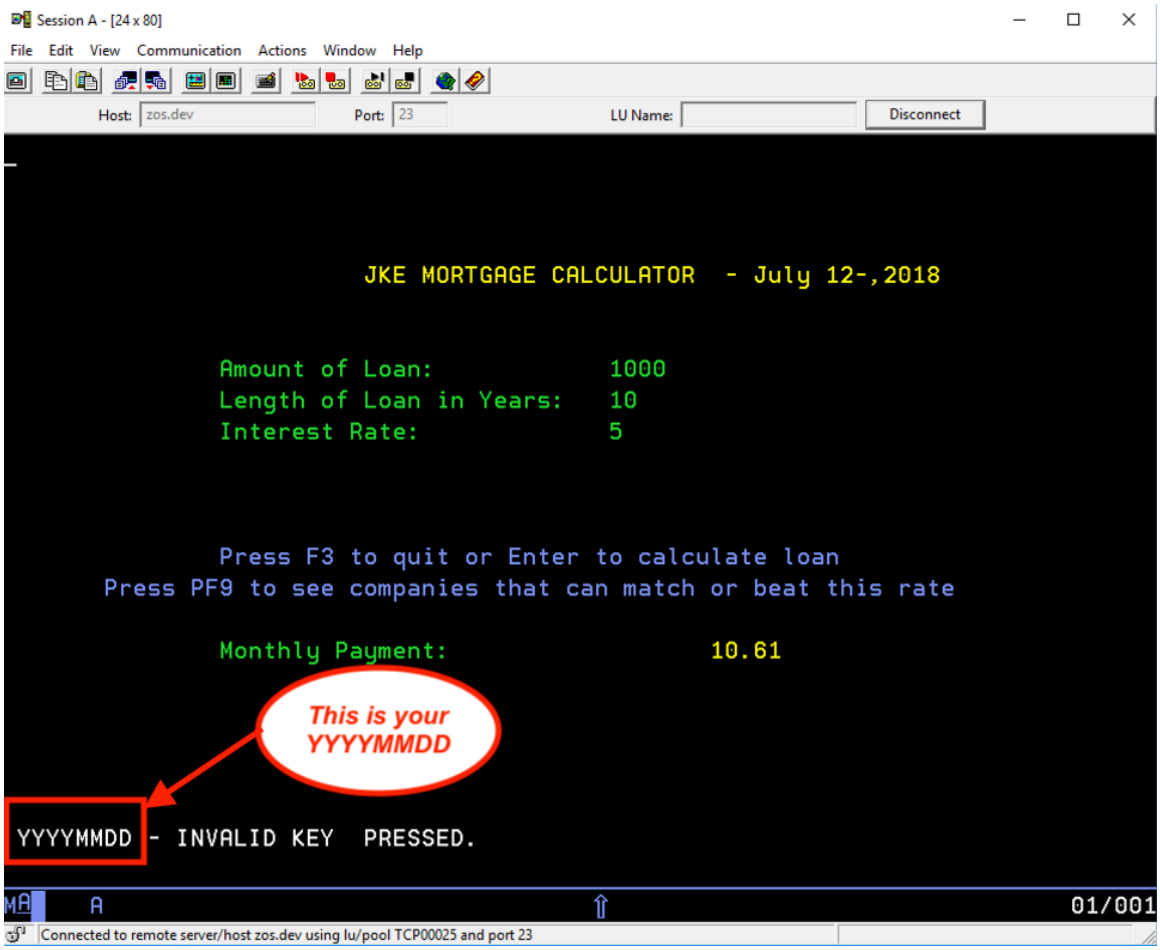
You should now be on the z/OS CICS region named *CICSTS53*. This is the CICS instance where you made the program changes.

4.4.1 Type the CICS transaction **j05p** and press the **Enter** key.

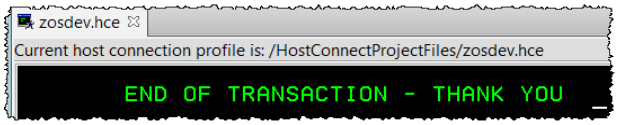
4.4.2 Press **enter** to see the monthly payment for the default data .



4.4.3 Press **F1** key and verify the message displayed
This was your mission. As you see the change has been implemented.



4.4.4 Press **F3** to end the CICS transaction.



Section 5. Push and Commit the changed code to Git.

Using IDz you will commit the changes you made to Git and later perform the final building and deploy using Jenkins and UCD.

___ 5.1 Using IDz with the Git plugin to compare the change versus original

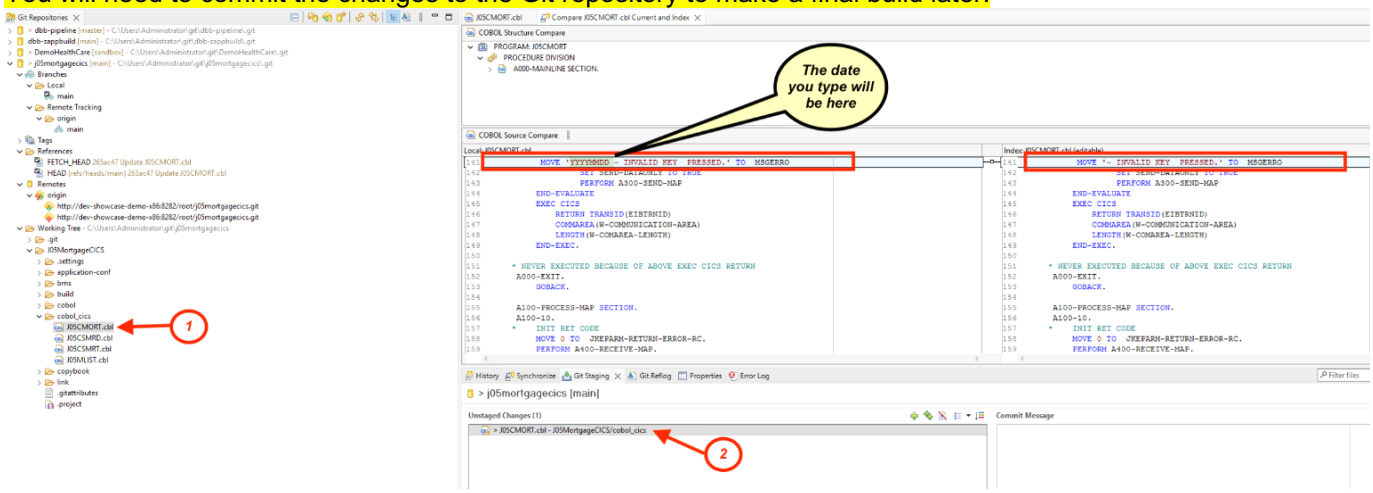
.

5.1.1 Switch to the **Git Perspective** selecting the icon on top and right

5.1.2 1 Expand the nodes and click on the program **J05CMORT.cbl** that you have modified. Notice that the changed program is listed in the *Git Staging* view..

2 Double click on the **J05CMORT.cbl** that is listed under ***Unstaged Changes*** and noticed the comparison among the program that you updated versus the one that is in the Git repository.

You will need to commit the changes to the Git repository to make a final build later.



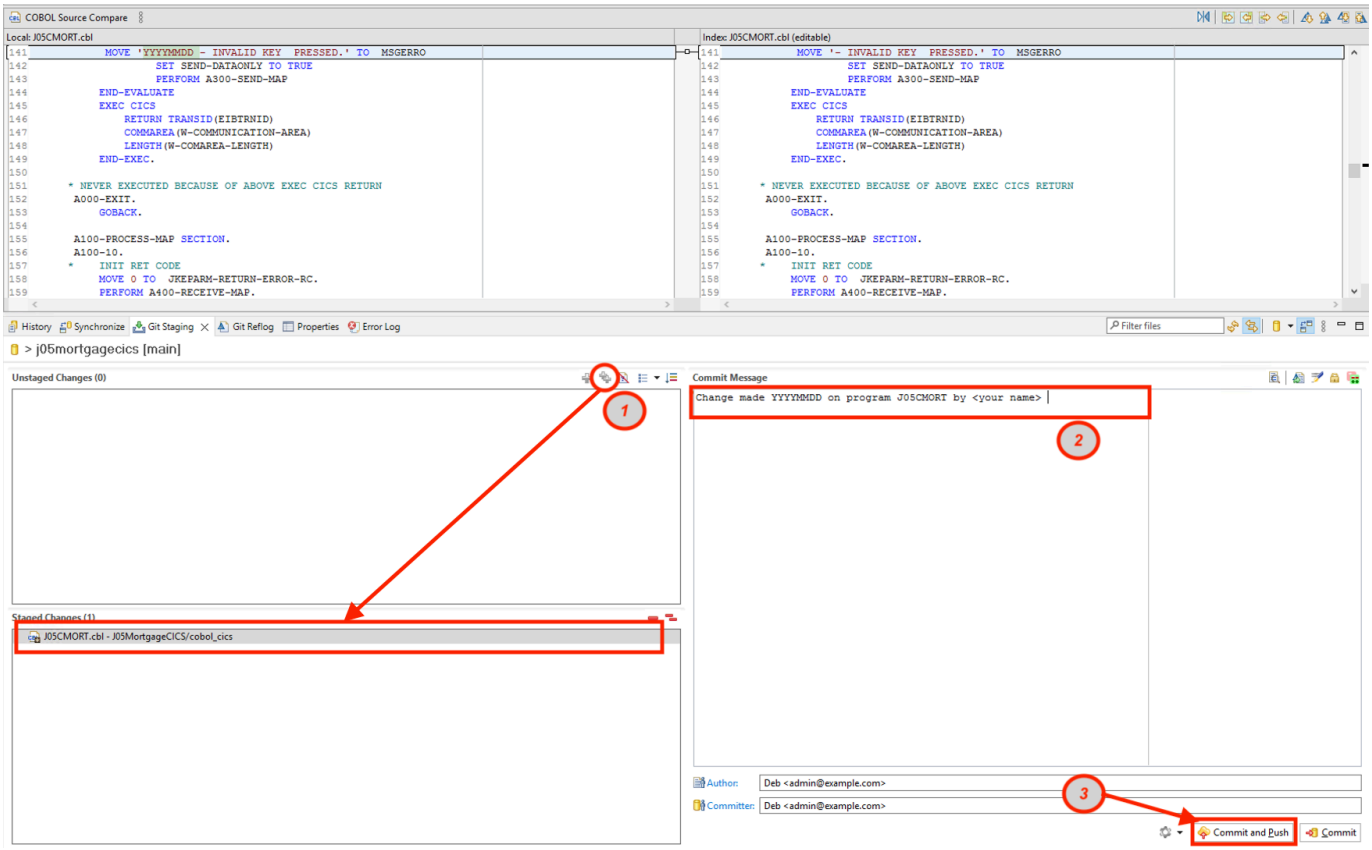
5.1.3 Use **Ctrl + Shift + F4** to close the editors.

___ 5.2 Push and Commit the changes to Git

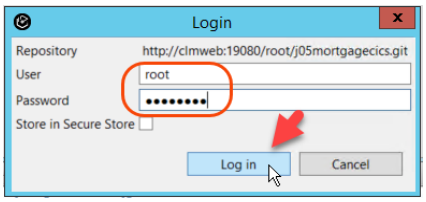
.

5.2.1 Using **Git Staging** view, ,

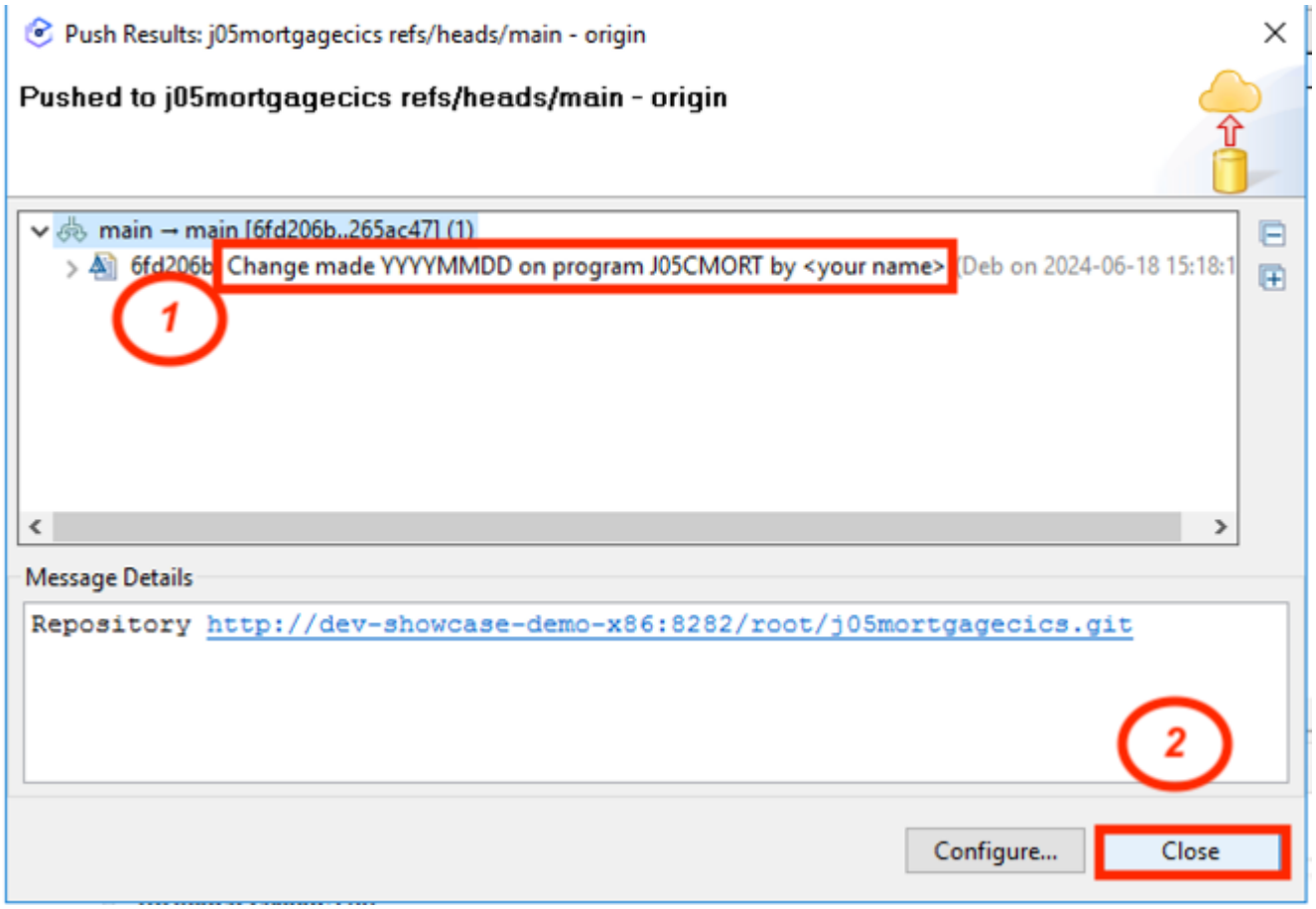
1. Click on the icon ,
2. Write a commit message like: **Change made yyyymmdd on program J05CMORT by *your name***
3. Click **Commit and Push …**



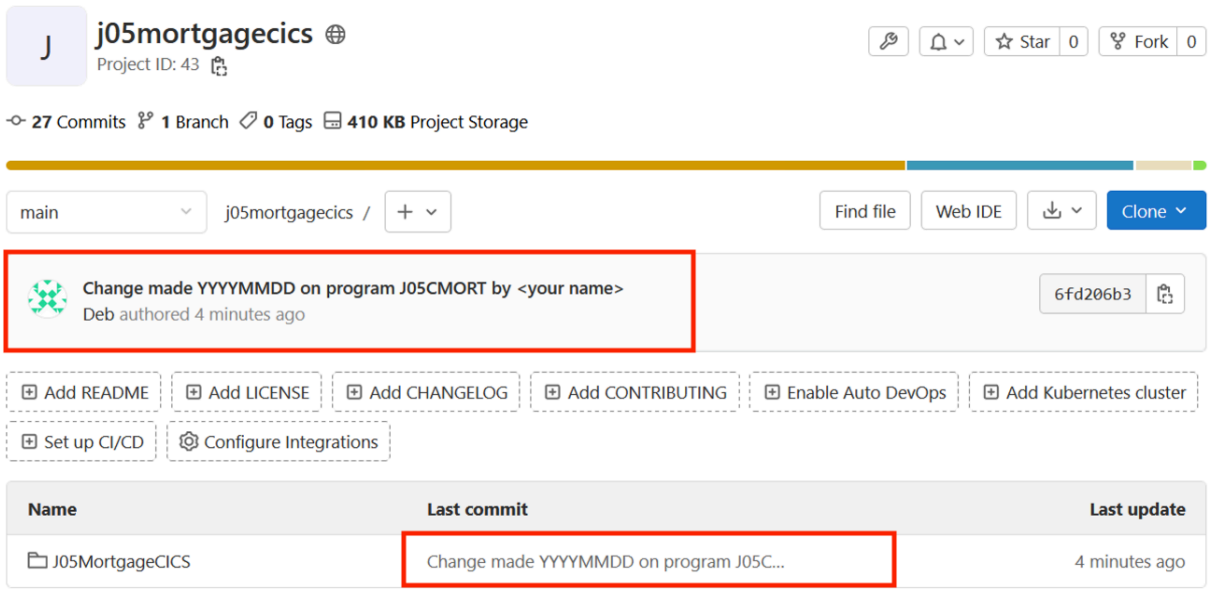
5.2.2 If it prompts for login credentials use **root** and password **passw0rd (0 is number zero)** and click **Log in**



5.2.3 ⏭ Verify that the commit was successful and click **Close**
⏭ Also use **Ctrl + Shift + F4** to close any opened editor



5.2.4 ⏭ To see this changes in *GitLab* use the **Firefox browser** and do a **Refresh** (**F5)** (details how to get the link at 2.1.1).



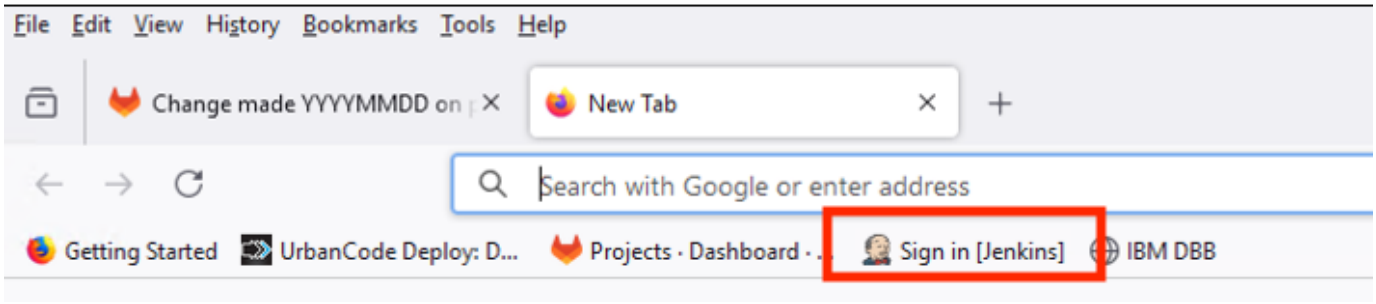
5.2.5 You also can click on **Change made yyyymmdd on program J05CMORT by xxxxx** to see the changes

Section 6. Use Jenkins with Git plugin to build all the modified code committed to Git.

At this point the changed code is delivered into the Git repository that is on Linux. You will use Jenkins pipeline to build the new changed code and push the executables to be deployed using UCD.Note that this step makes a build of all code committed to Git, while on Section 4 only the selected COBOL program was built.

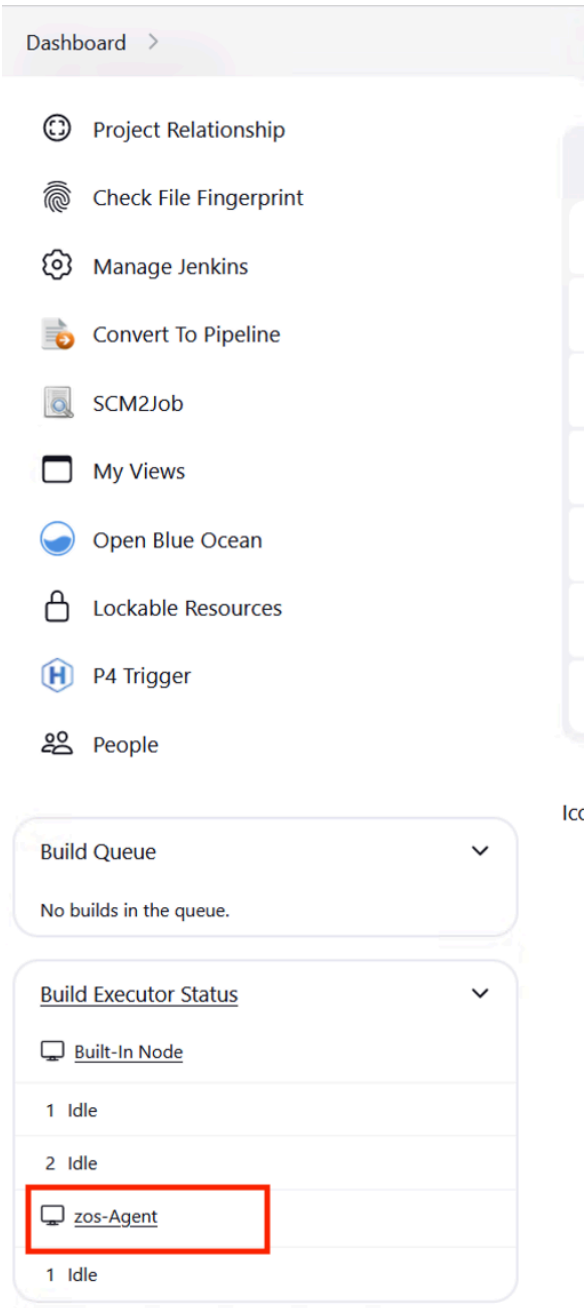
___ 6.1 Logon to Jenkins using a Web Browser and start the z/OS agent

6.1.1 ⏭ Using the **Bookmarks Tool Bar** click on **Sign in (Jenkins)**
)

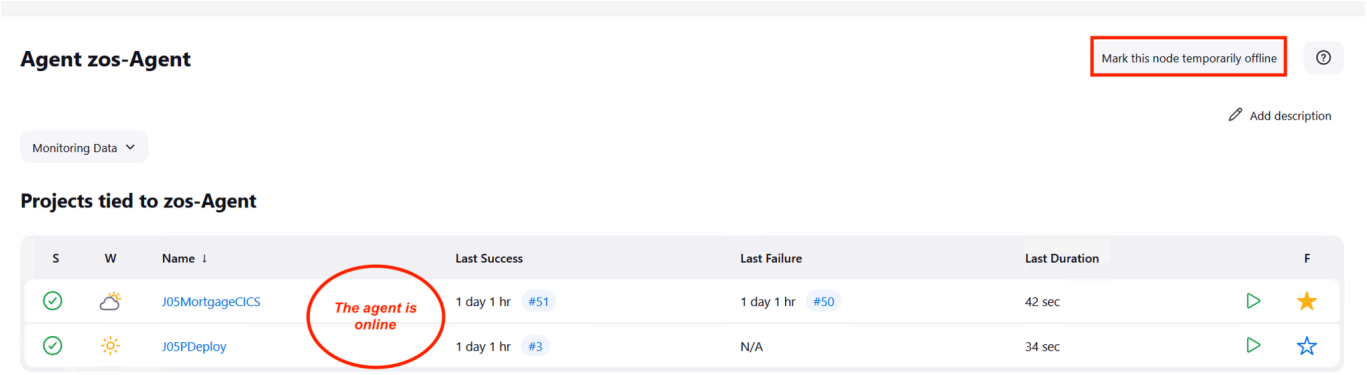


6.1.2 ⏭ If required type the user **ibmuser** and password **passw0rd** (**0 is number zero**) and click **log in**.

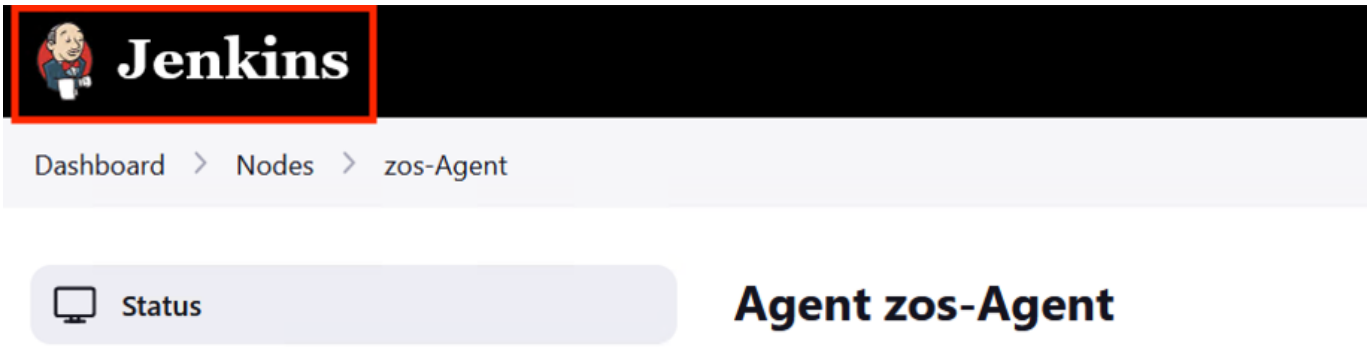
6.1.3 ⏭ Scroll down and click on **zos-Agent**



6.1.4 ⏭ If the agent zos-Agent is online go to the next step . If not On the right corner click on the **Bring this node back online** “blue button”.
This will bring the Jenkins agent online again.



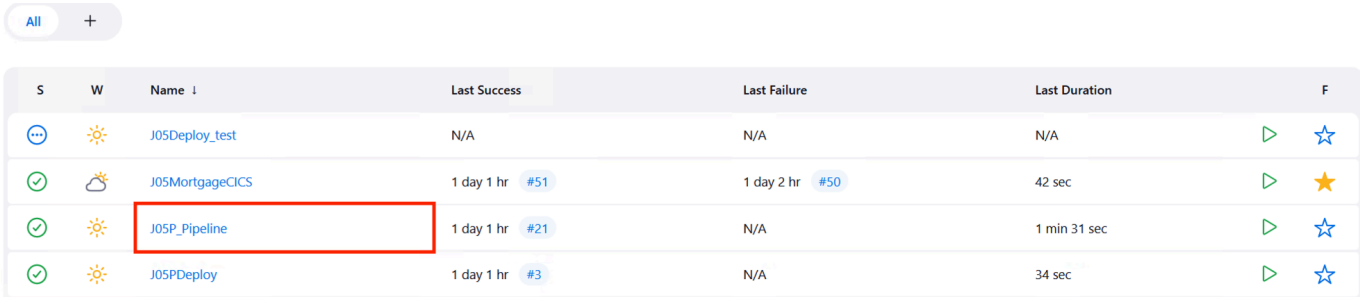
6.1.5 Click on **Jenkins** on top left:



___ 6.2 Starting the Jenkins Pipeline

6.2.1 The homepage lists all of the Jenkins jobs and pipelines.

Click on the hyperlink for the project **J05P_Pipeline** that represents your CICS application.



6.2.2 This opens the **J05P_Pipeline** view. This pipeline has been designed with two stages:

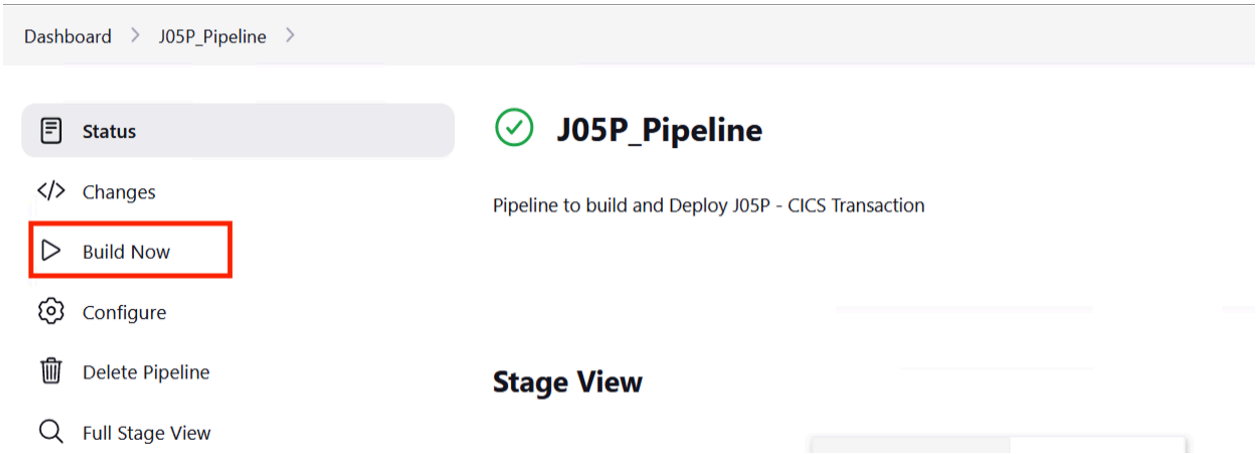
1 – Stage 1 builds ONLY the changed code from Git. This is an example of a Build pipeline.

2 – Stage 2 deploys the changes into CICS Development Region (Dev) using UrbanCode Deploy (UCD). This is an example of a Delivery pipeline.

This pipeline is a simplistic form of an actual delivery pipeline. Any other activity such as, test cases, code reviews, batch job runs, rexx scripts, table queries and other manual tasks that are currently being carried out in a shop before code is moved to QA (or any other region) can be built into a pipeline.

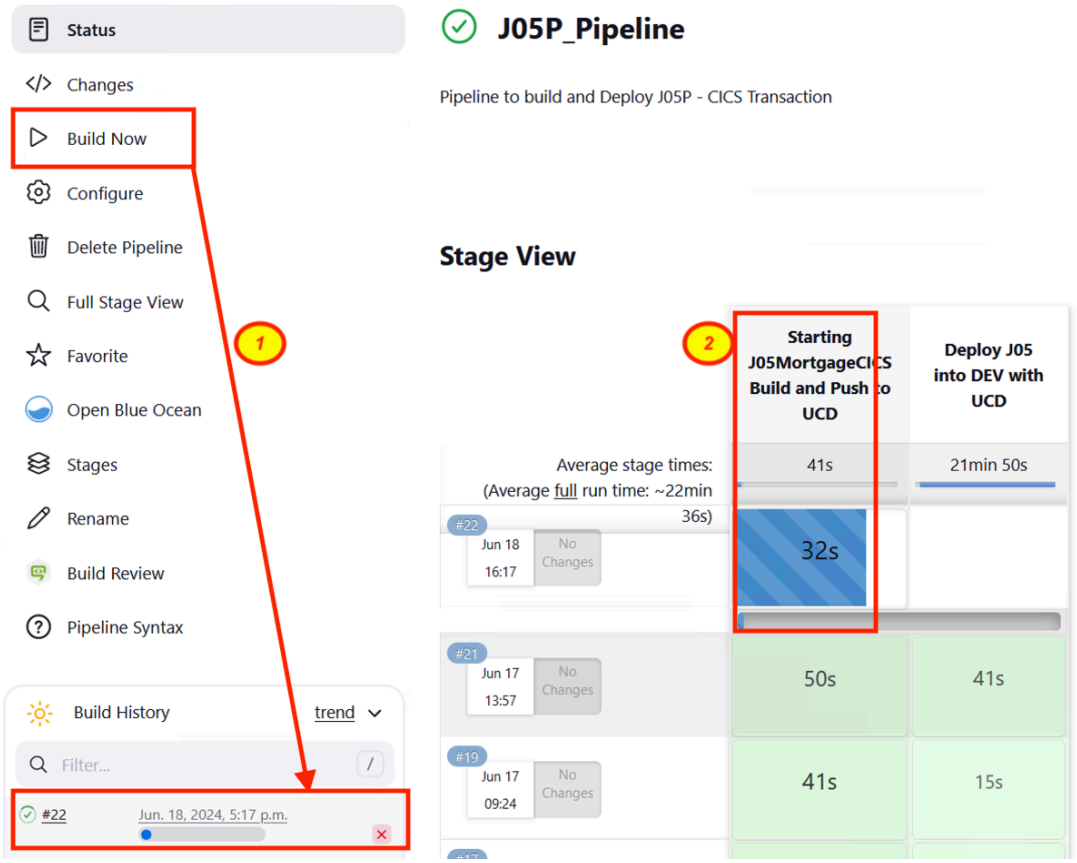
⏭ Click on **Build Now** on the left.

IMPORTANT : This *Jenkins Build* will take some **Minutes** since the cloud instances are slow compared to a real environment.



6.2.3 1 Notice that the new build has appeared under **Build History** indicating the currently started Pipeline.

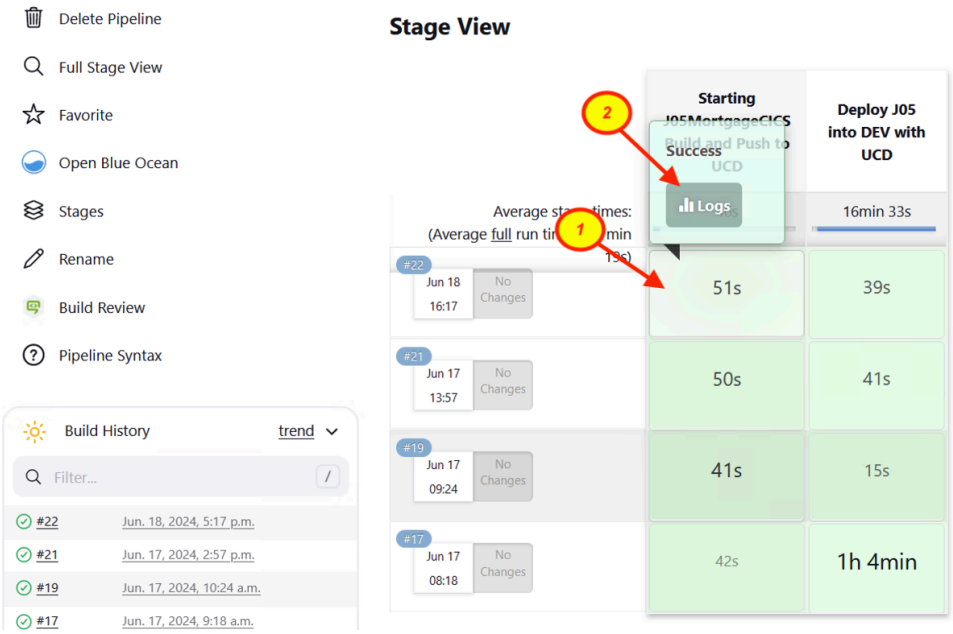
2 The **Stage View** also shows a new line with an in-progress bar.



___ 6.3 Checking the results

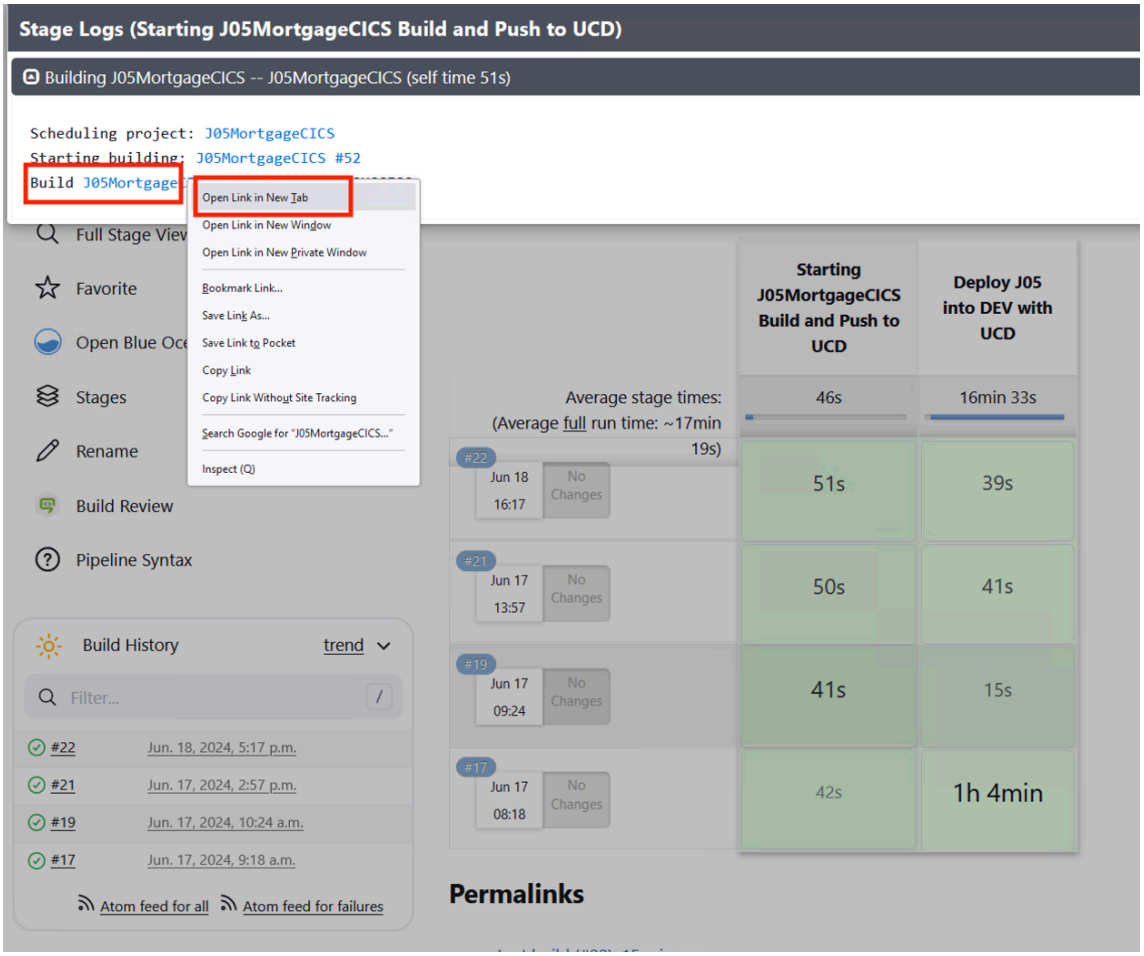
6.3.1 Since this z/OS is running in a small processor in the cloud, this activity may take some minutes, but you don't need to wait to be completed to check the results.
You can follow what is going on as described below ..

⏭ Move the cursor and click on the **green area** 1 and then click on **Logs** 2.

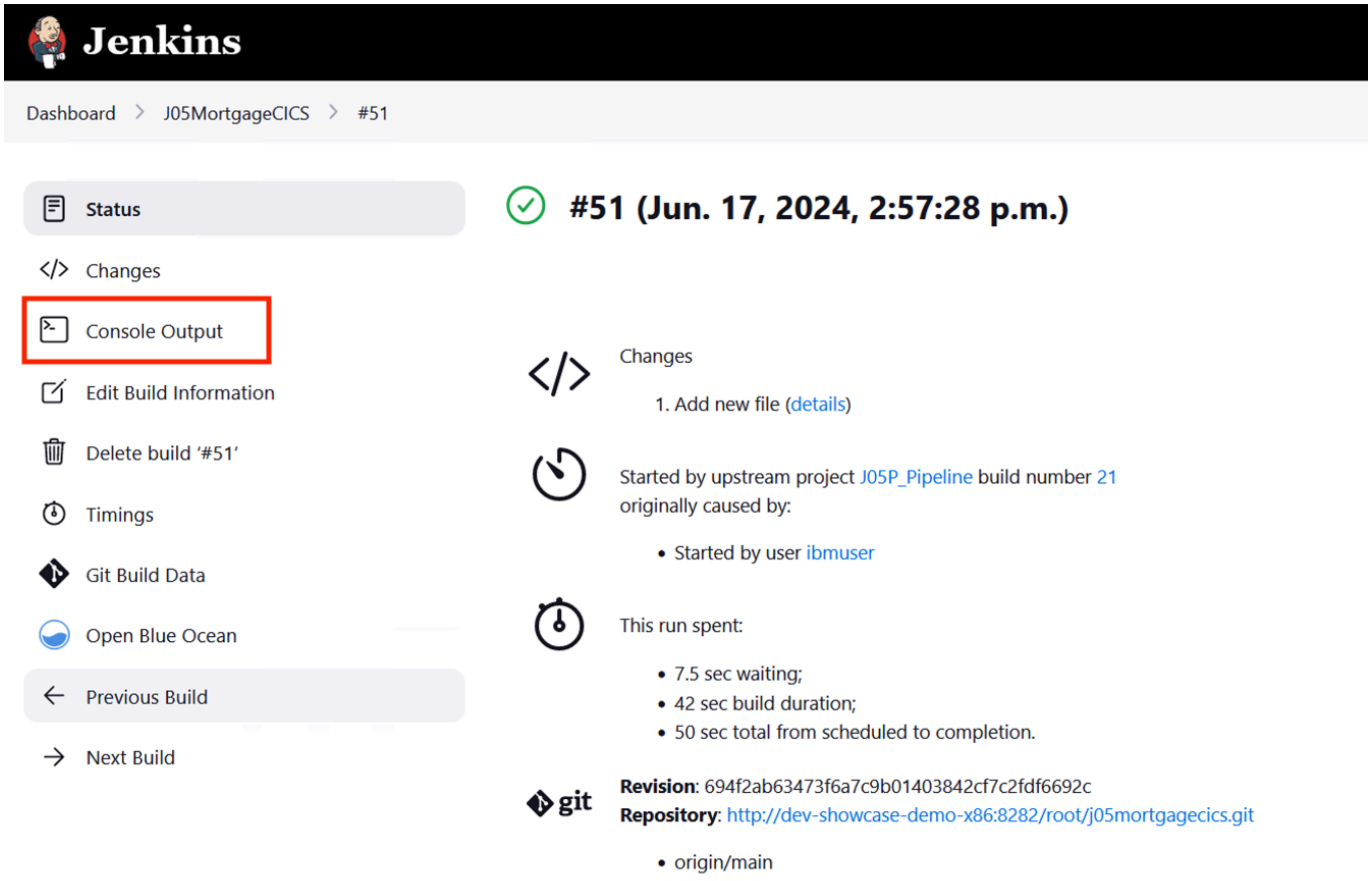


6.3.2 The dialog *Stage Logs* will open.

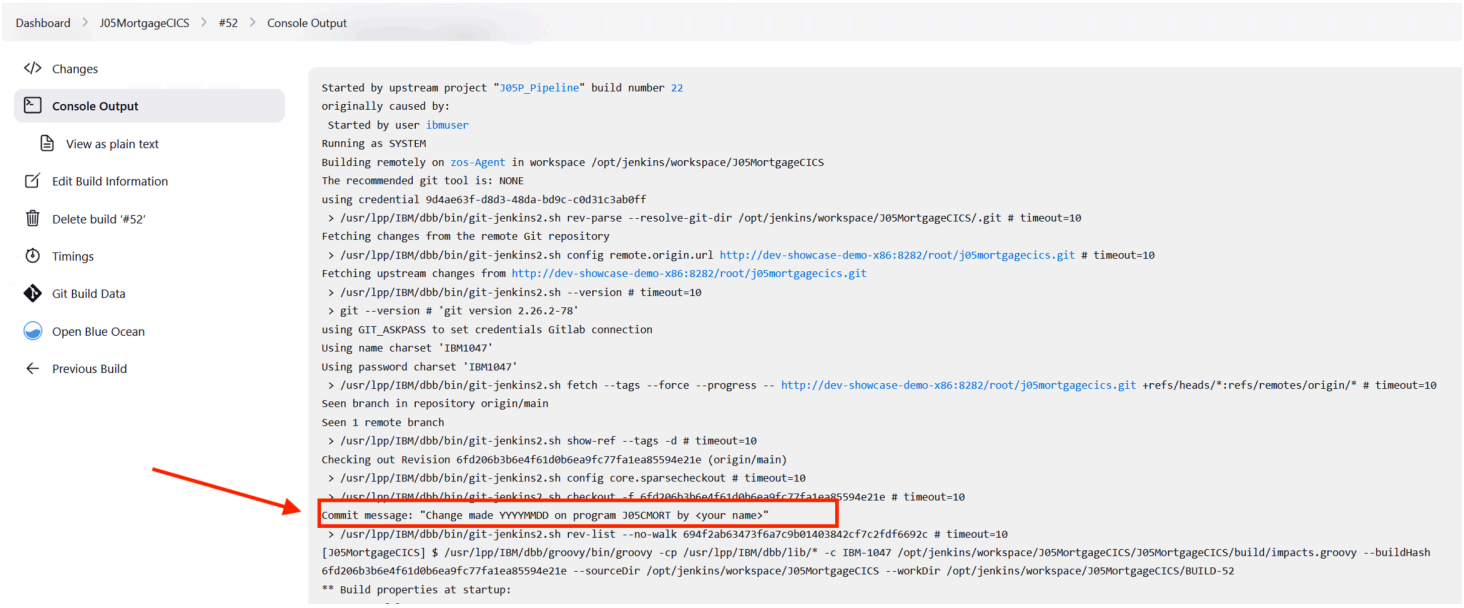
⏭ Right click on **J05MortgageCICS #nn** and select **Open Link in New Tab**



6.3.3 This opens the current running task for the first step of the J05Mortgage Application Build.
⏭ Click on the **Console Output** to view the log.



6.3.4 The Console Output log of first pipeline shows the various Groovy scripts being executed.
You need to understand what is going on !!.

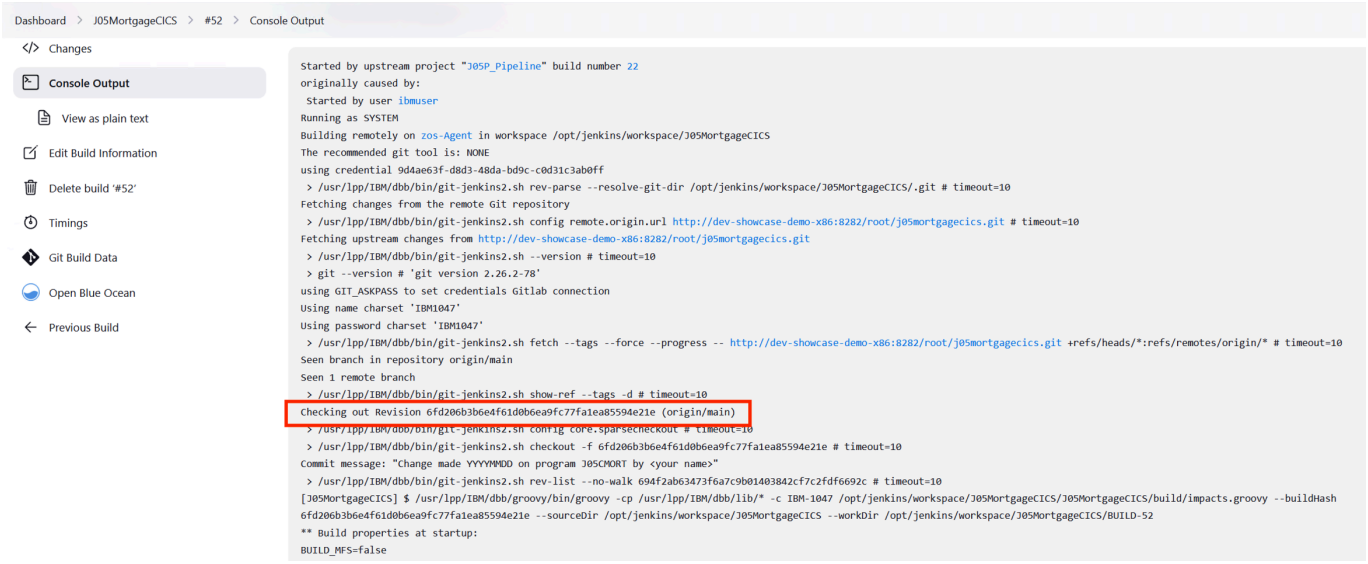


6.4 Understand the results

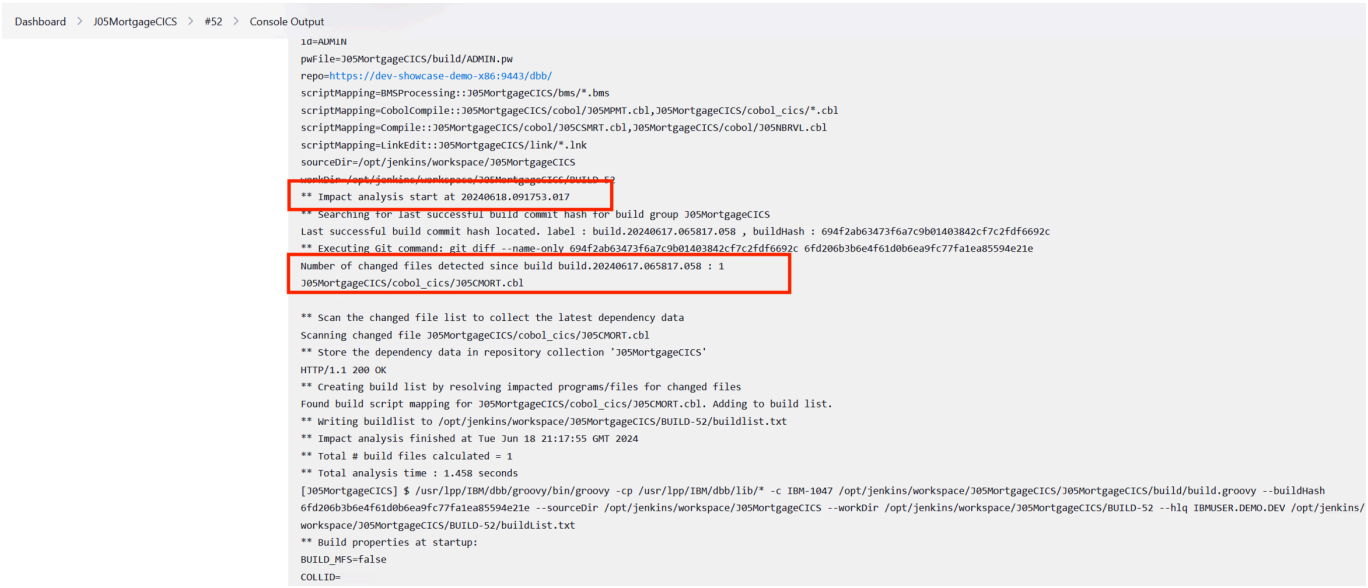
This task performs five sub-steps.

⏭ Scroll down and locate the highlighted portions shown below from the build console output to know more about the build

6.4.1 **Check out of code** from Git Repositories based on the latest commit



6.4.2 **An Impact Analysis** to identify if any programs have been changed since the last build using DBB Impact scripts. The DBB server that is installed on USS on z/OS keeps track of all builds.



Dashboard > J05MortgageCICS > #52 > Console Output

```
id=ADMIN
pwFile=J05MortgageCICS/build/ADMIN.pw
repo=https://dev-showcase-demo-x86:9443/dbb/
scriptMapping=BMSProcessing::J05MortgageCICS/bms/*.bms
scriptMapping=CobolCompile::J05MortgageCICS/cobol/J05MPMT.cbl,J05MortgageCICS/cobol_cics/*.cbl
scriptMapping=Compile::J05MortgageCICS/cobol/J05CSMRT.cbl,J05MortgageCICS/cobol/J05NBRVL.cbl
scriptMapping=LinkEdit::J05MortgageCICS/link/*.lnk
sourceDir=/opt/jenkins/workspace/J05MortgageCICS
workDir=/opt/jenkins/workspace/J05MortgageCICS/BUILD-52
** Impact analysis start at 20240618.091753.017
** Searching for last successful build commit hash for build group J05MortgageCICS
Last successful build commit hash located. label : build.20240617.065817.058 , buildHash : 694f2ab63473f6a7c9b01403842cf7c2fdf6692c
** Executing Git command: git diff --name-only 694f2ab63473f6a7c9b01403842cf7c2fdf6692c 6fd206b3b6e4f61d0b6ea9fc77fa1ea85594e21e
Number of changed files detected since build build.20240617.065817.058 : 1
J05MortgageCICS/cobol_cics/J05CMORT.cbl

** Scan the changed file list to collect the latest dependency data
Scanning changed file J05MortgageCICS/cobol_cics/J05CMORT.cbl
** Store the dependency data in repository collection 'J05MortgageCICS'
HTTP/1.1 200 OK
** Creating build list by resolving impacted programs/files for changed files
Found build script mapping for J05MortgageCICS/cobol_cics/J05CMORT.cbl. Adding to build list.
** Writing buildlist to /opt/jenkins/workspace/J05MortgageCICS/BUILD-52/buildlist.txt
** Impact analysis finished at Tue Jun 18 21:17:55 GMT 2024
** Total # build files calculated = 1
** Total analysis time : 1.458 seconds
[J05MortgageCICS] $ /usr/lpp/IBM/dbb/groovy/bin/groovy -cp /usr/lpp/IBM/dbb/lib/* -c IBM-1047 /opt/jenkins/workspace/J05MortgageCICS/J05MortgageCICS/build/build.groovy --buildHash
6fd206b3b6e4f61d0b6ea9fc77fa1ea85594e21e --sourceDir /opt/jenkins/workspace/J05MortgageCICS --workDir /opt/jenkins/workspace/J05MortgageCICS/BUILD-52 --hlq IBMUSER.DEMO.DEV /opt/jenkins/
workspace/J05MortgageCICS/BUILD-52/buildList.txt
** Build properties at startup:
BUILD_MFS=false
COLLID=
```

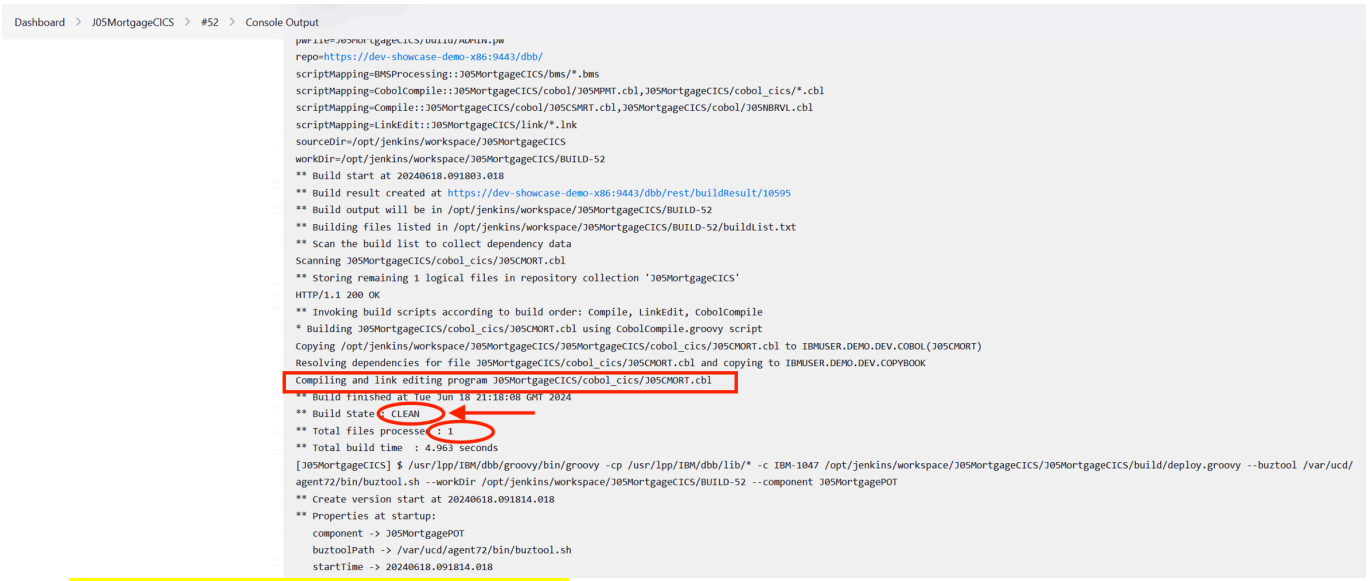
6.4.3 **A dependency scan** to pick out all the dependencies of the changed programs,
The DBB server that is installed on USS on z/OS keeps track of all dependencies.

Dashboard > J05MortgageCICS > #52 > Console Output

```
workDir=/opt/jenkins/workspace/J05MortgageCICS/BUILD-52
** Impact analysis start at 20240618.091753.017
** Searching for last successful build commit hash for build group J05MortgageCICS
Last successful build commit hash located. label : build.20240617.065817.058 , buildHash : 694f2ab63473f6a7c9b01403842cf7c2fdf6692c
** Executing Git command: git diff --name-only 694f2ab63473f6a7c9b01403842cf7c2fdf6692c 6fd206b3b6e4f61d0b6ea9fc77fa1ea85594e21e
Number of changed files detected since build build.20240617.065817.058 : 1
J05MortgageCICS/cobol_cics/J05CMORT.cbl

** Scan the changed file list to collect the latest dependency data
Scanning changed file J05MortgageCICS/cobol_cics/J05CMORT.cbl
** Store the dependency data in repository collection 'J05MortgageCICS'
HTTP/1.1 200 OK
** Creating build list by resolving impacted programs/files for changed files
Found build script mapping for J05MortgageCICS/cobol_cics/J05CMORT.cbl. Adding to build list.
** Writing buildlist to /opt/jenkins/workspace/J05MortgageCICS/BUILD-52/buildlist.txt
** Impact analysis finished at Tue Jun 18 21:17:55 GMT 2024
** Total # build files calculated = 1
** Total analysis time : 1.458 seconds
[J05MortgageCICS] $ /usr/lpp/IBM/dbb/groovy/bin/groovy -cp /usr/lpp/IBM/dbb/lib/* -c IBM-1047 /opt/jenkins/workspace/J05MortgageCICS/J05MortgageCICS/build/build.groovy --buildHash
6fd206b3b6e4f61d0b6ea9fc77fa1ea85594e21e --sourceDir /opt/jenkins/workspace/J05MortgageCICS --workDir /opt/jenkins/workspace/J05MortgageCICS/BUILD-52 --hlq IBMUSER.DEMO.DEV /opt/jenkins/
workspace/J05MortgageCICS/BUILD-52/buildList.txt
** Build properties at startup:
BUILD_MFS=false
COLLID=
CONFDIR=
MACLIB=SYS1.MACLIB
OWNER=
QUAL=
REFERAL=
RUN_DB2_BIND=false
SASMMOD1=HLA.SASMMOD1
SCEELKED=CEE.SCEELKED
```

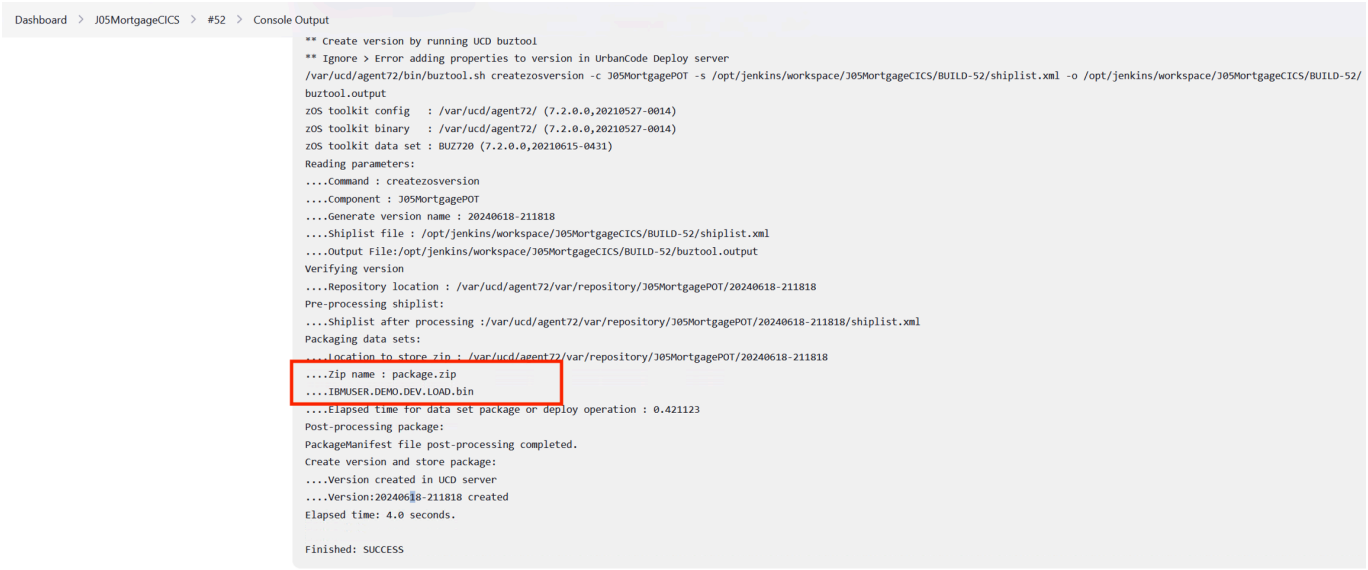
This collection is on DBB Server

6.4.4 **A compile/link-edit** of the changed programs



6.4.5 **Creation of a deployable binary version** of code that can be used by *UrbanCode Deploy* to deploy to the necessary environments. Here Jenkins is using the UCD plugin to store an executable version at UCD server for later deploy.

Ignore the error message below. This does not impact the UCD version creation..

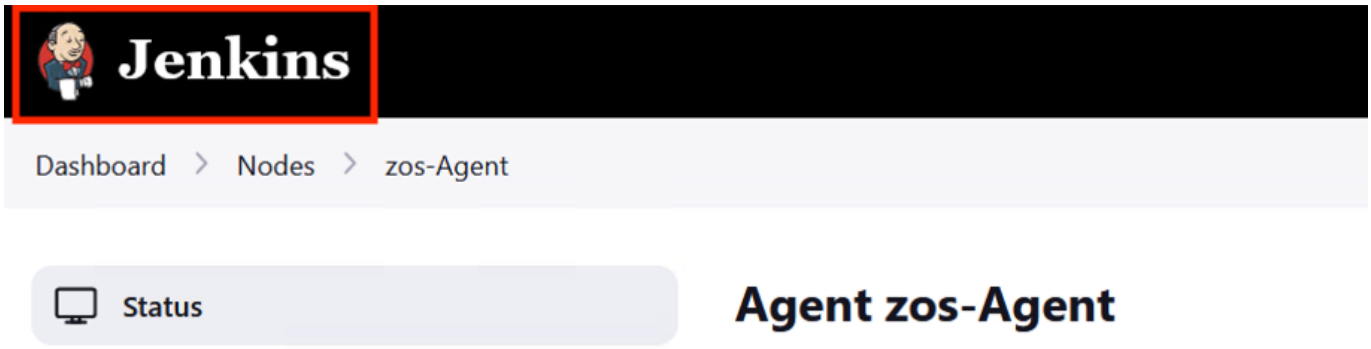

Dashboard > J05MortgageCICS > #52 > Console Output
** Create version by running UCD buztool
** Ignore > Error adding properties to version in UrbanCode Deploy server
/var/ucd/agent72/bin/buztool.sh createzosversion -c J05MortgagePOT -s /opt/jenkins/workspace/J05MortgageCICS/BUILD-52/shiplist.xml -o /opt/jenkins/workspace/J05MortgageCICS/BUILD-52/buztool.output
zOS toolkit config : /var/ucd/agent72/ (7.2.0.0,20210527-0014)
zOS toolkit binary : /var/ucd/agent72/ (7.2.0.0,20210527-0014)
zOS toolkit data set : BUZ720 (7.2.0.0,20210615-0431)
Reading parameters:
....Command : createzosversion
....Component : J05MortgagePOT
....Generate version name : 20240618-211818
....Shiplist file : /opt/jenkins/workspace/J05MortgageCICS/BUILD-52/shiplist.xml
....Output File:/opt/jenkins/workspace/J05MortgageCICS/BUILD-52/buztool.output
Verifying version
....Repository location : /var/ucd/agent72/var/repository/J05MortgagePOT/20240618-211818
Pre-processing shiplist:
....Shiplist after processing :/var/ucd/agent72/var/repository/J05MortgagePOT/20240618-211818/shiplist.xml
Packaging data sets:
....Location to store zip : /var/ucd/agent72/var/repository/J05MortgagePOT/20240618-211818
....Zip name : package.zip
....IBMUSER.DEMO.DEV.LOAD.bin
....Elapsed time for data set package or deploy operation : 0.421123
Post-processing package:
PackageManifest file post-processing completed.
Create version and store package:
....Version created in UCD server
....Version:20240618-211818 created
Elapsed time: 4.0 seconds.
Finished: SUCCESS

Section 7. Use Jenkins and UCD plugin to deploy results and test the CICS transaction again

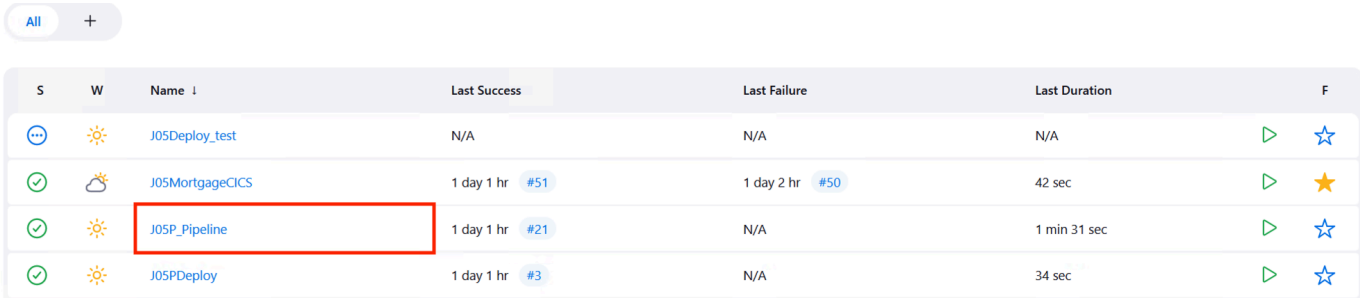
You will verify the results of the second stage (deploy using UCD) .

___ 7.1 Checking the deploy results (second stage)

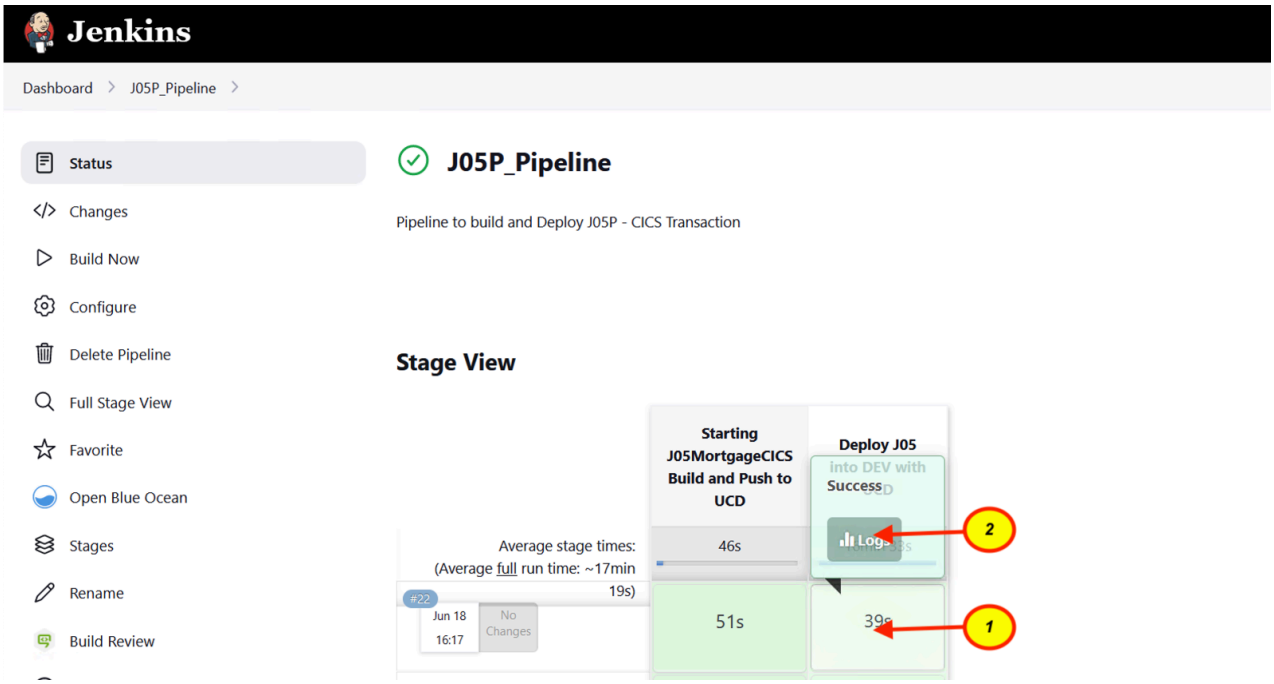
7.1.1 ⏭ Click on Jenkins to back to Dashboard.



7.1.2 ⏭ Click on **J05P_Pipeline**

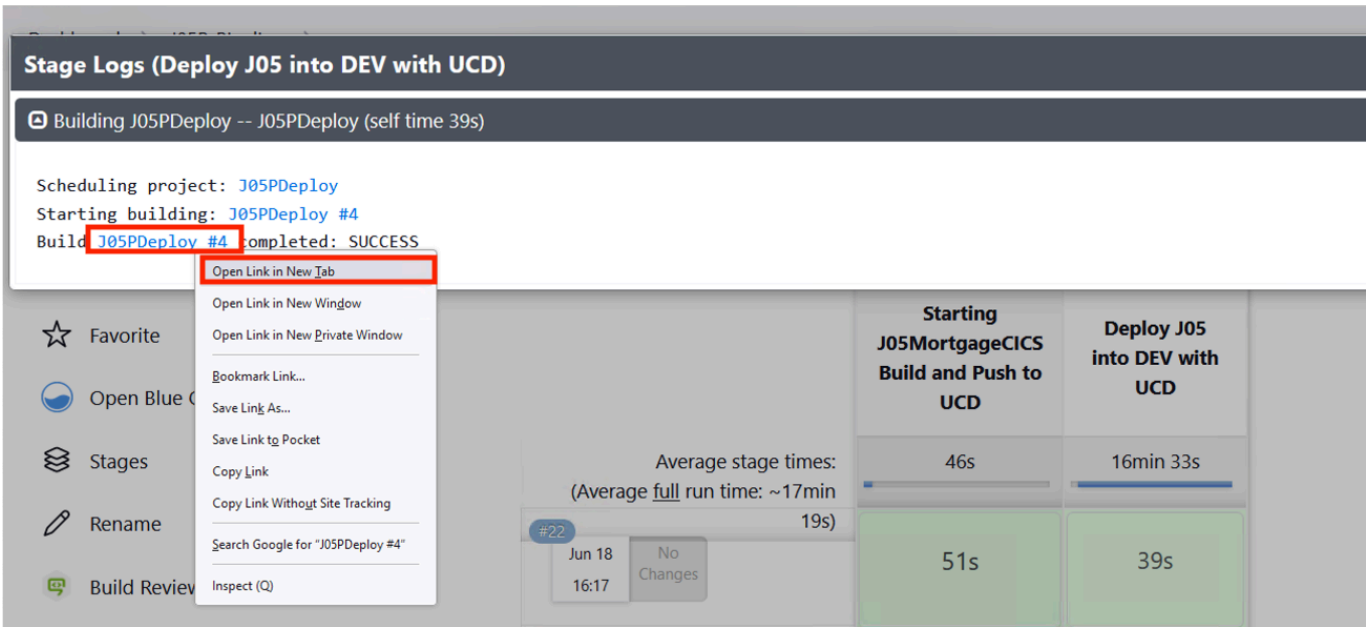


7.1.3 ⏭ Click as show in the **blue area** of the **Deploy** stage 1 and then click on **Logs** 2.

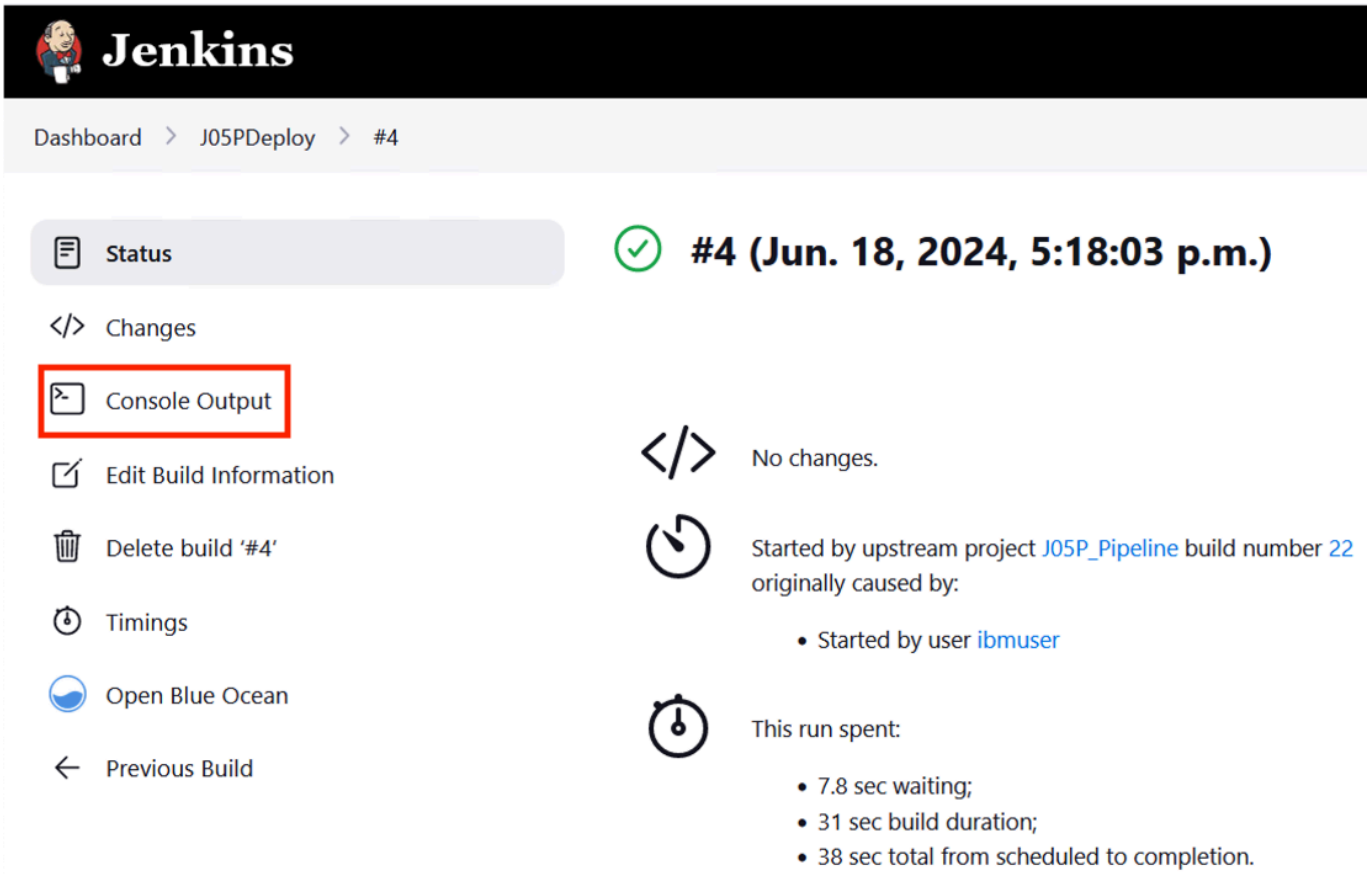


7.1.4 The dialog Stage Logs will open.

Right click on **J05PDeploy #nn** and select **Open Link in New Tab**



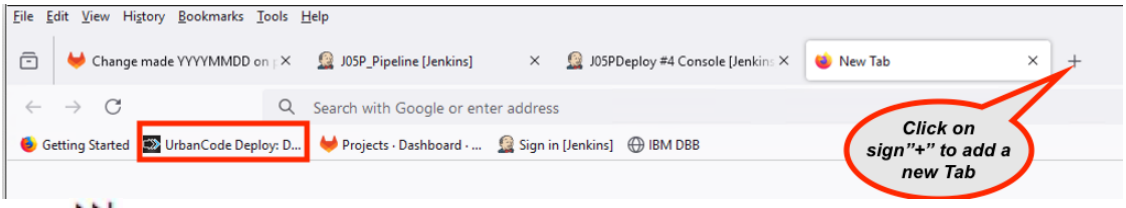
7.1.5 Click on the **Console Output** to view the logs.



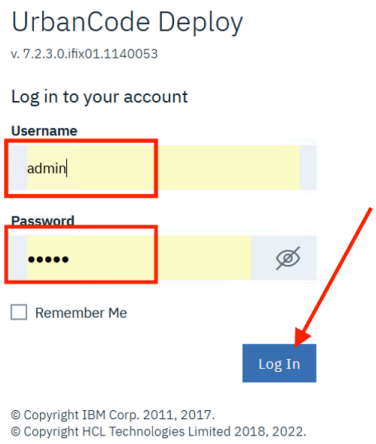
7.1.6 The Console Output log of second pipeline shows that the program that you delivered will be deployed to CICS Dev environment using UCD..

___ 7.2 Checking UrbanCode Deploy logs

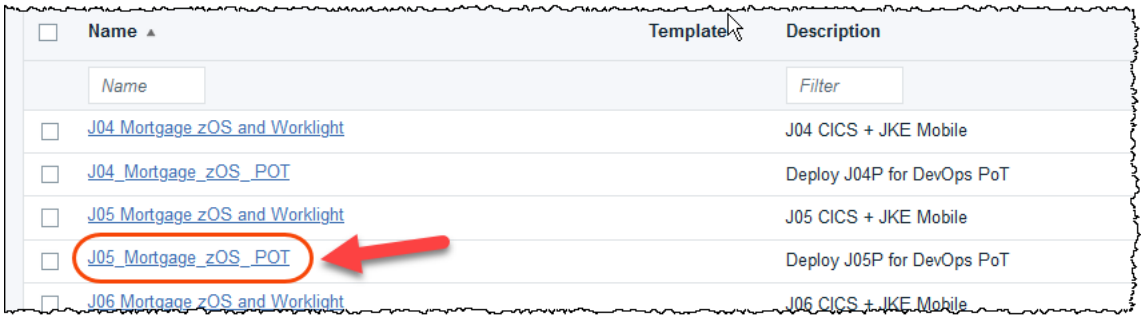
7.2.1 Open a new Tab on Firebox. And start UCD console using the Bookmarks ToolBar below:



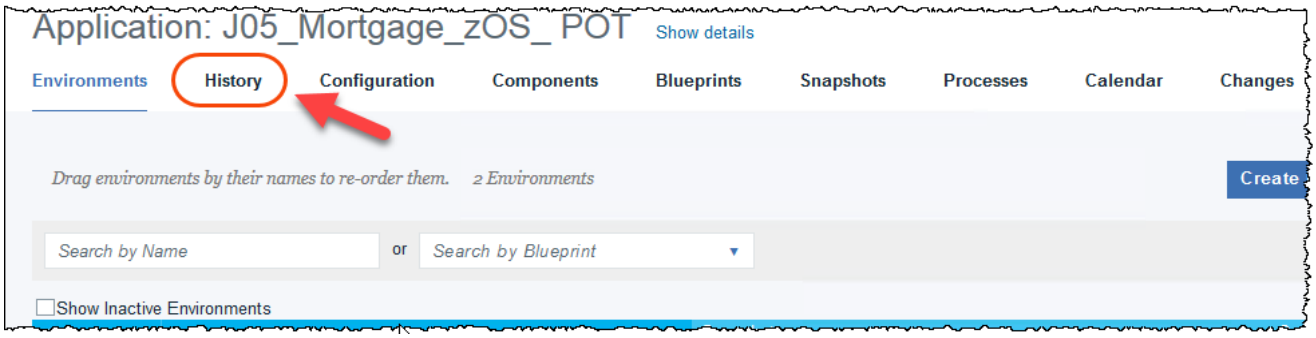
7.2.2 Using **admin** and password **admin** and click **Login**



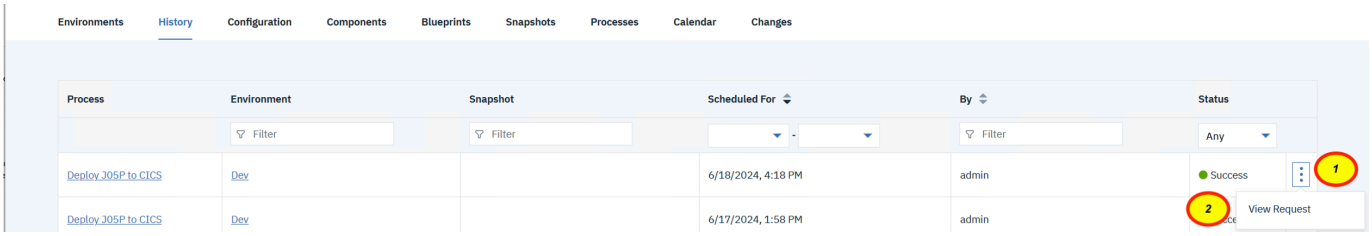
7.2.3 Click on **Applications** and find your application **`J05_Mortgage_zOS_POT`** and **click** on it



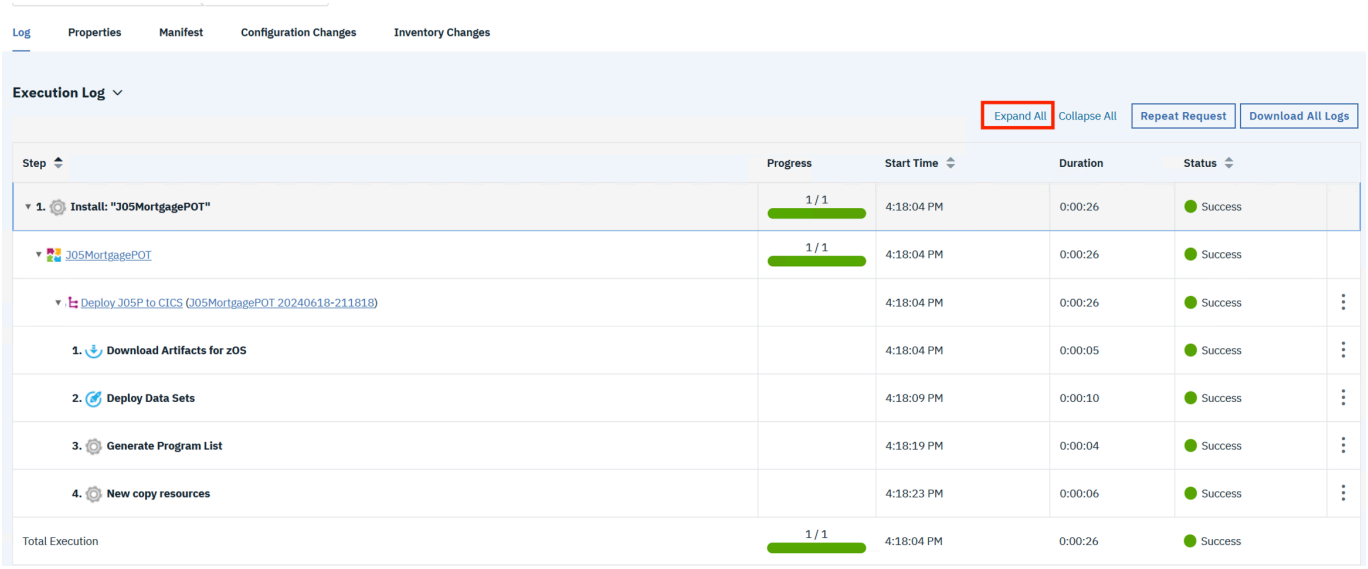
7.2.4 Click on the **History** tab



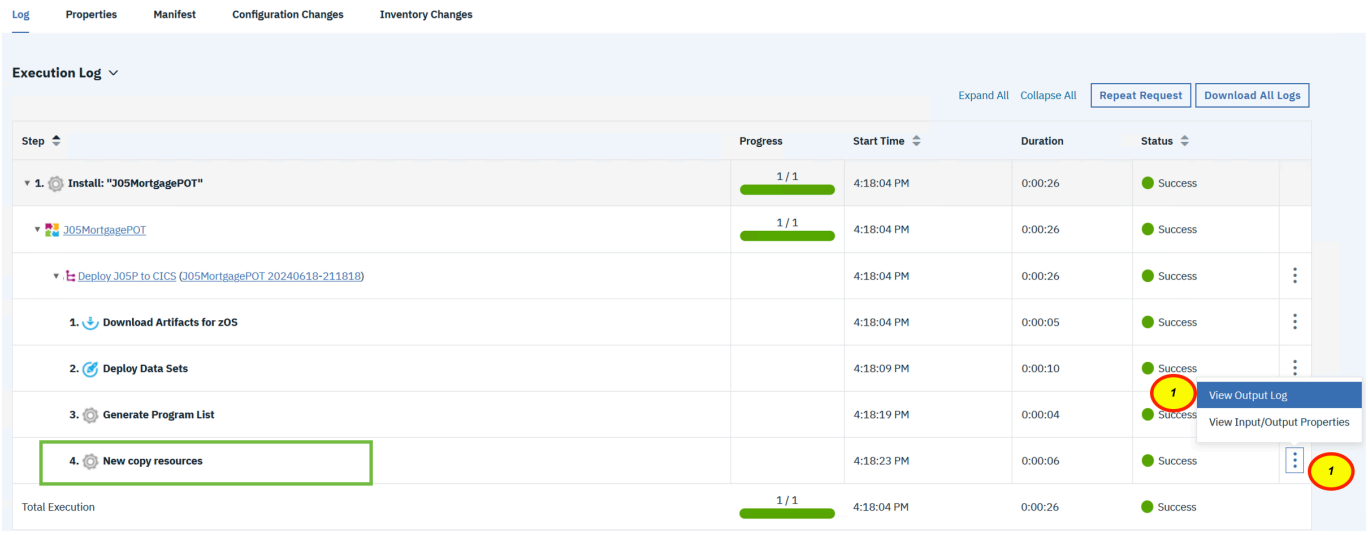
7.2.5 Click on 3 dots icon and select **View Request** entry that is related to your deploy (check the date/time)



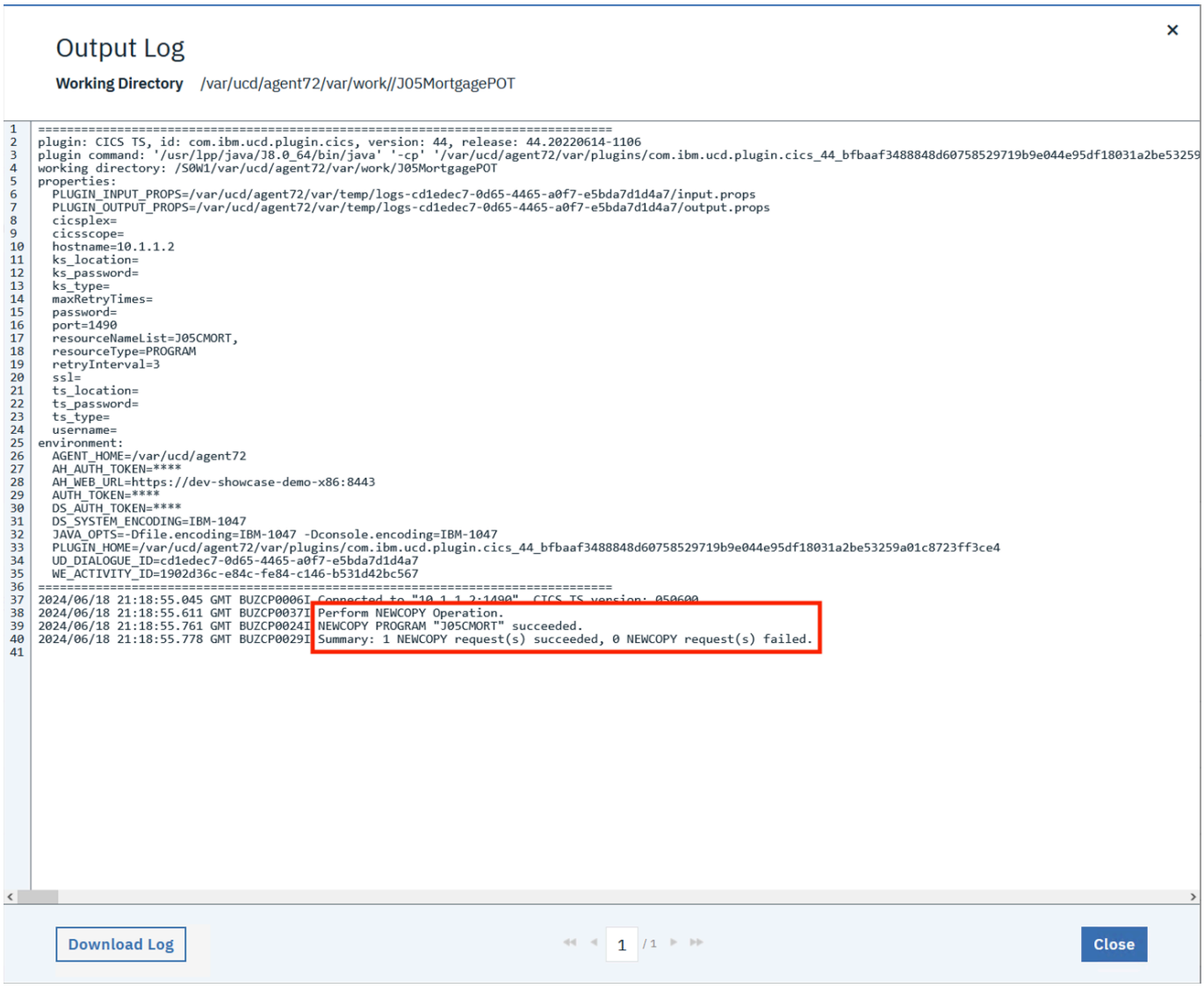
7.2.6 Click **Expand All** and **expand the node** as show in the #2 below
Once the step is green means that it is completed.



7.2.7 Once the **New copy** is green, click on 3 Dots and click **View Output log** (last UCD step)

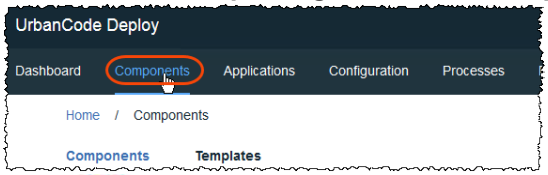


7.2.8 Notice that **J05CMORT** program was deployed to CICS Dev environment. and a CICS NEWCOPY was succeeded.

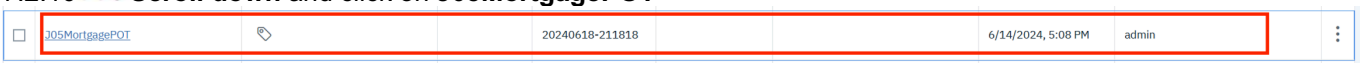


7.2.9 You could verify the binaries generated by the build (LOADLIB) at the UCD Code station. You can have versions of executables stored under UCD.

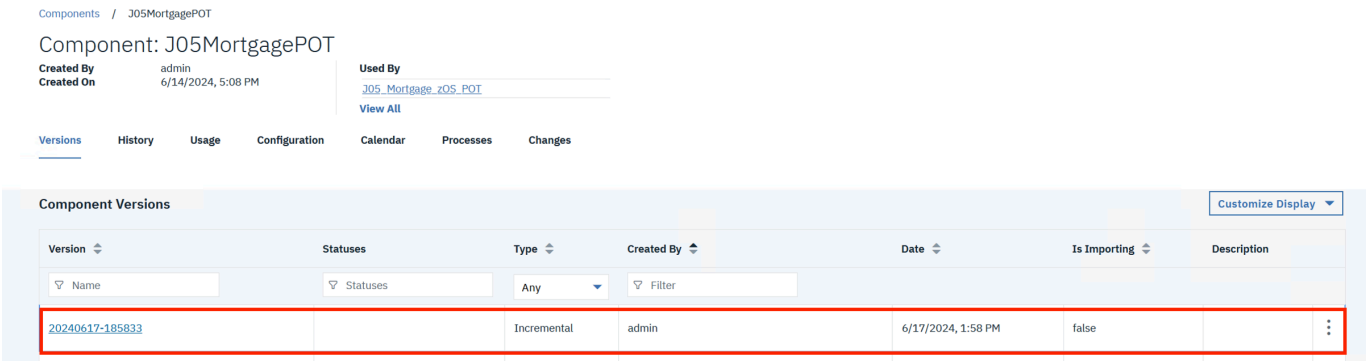
⏭ **Close the Output Log** and click on **Components**



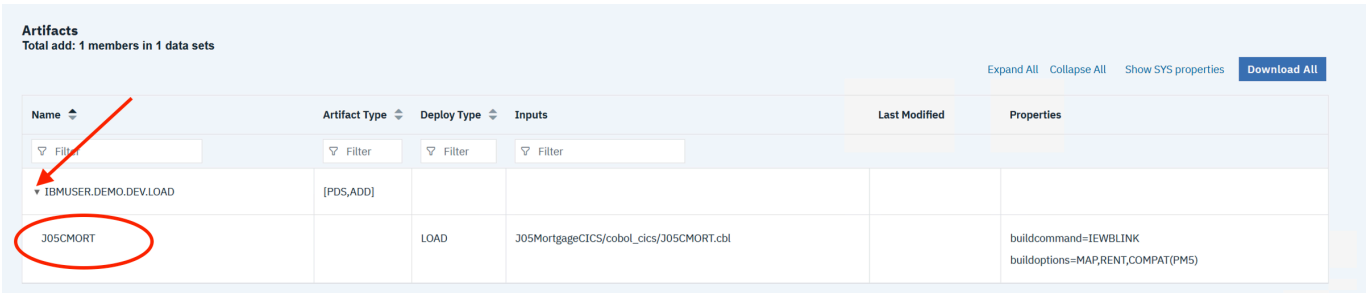
7.2.10 ⏭ **Scroll down** and click on **J05MortgagePOT**



7.2.11 ⏭ Click on **Versions** and the version created by your Jenkins run (it is a **timestamp value**)



7.2.12 Expand **IBMUSER.DEMO.DEV.LOAD** and you will see the load module created by the build on Jenkins pipeline. This is the load module that will be deployed.

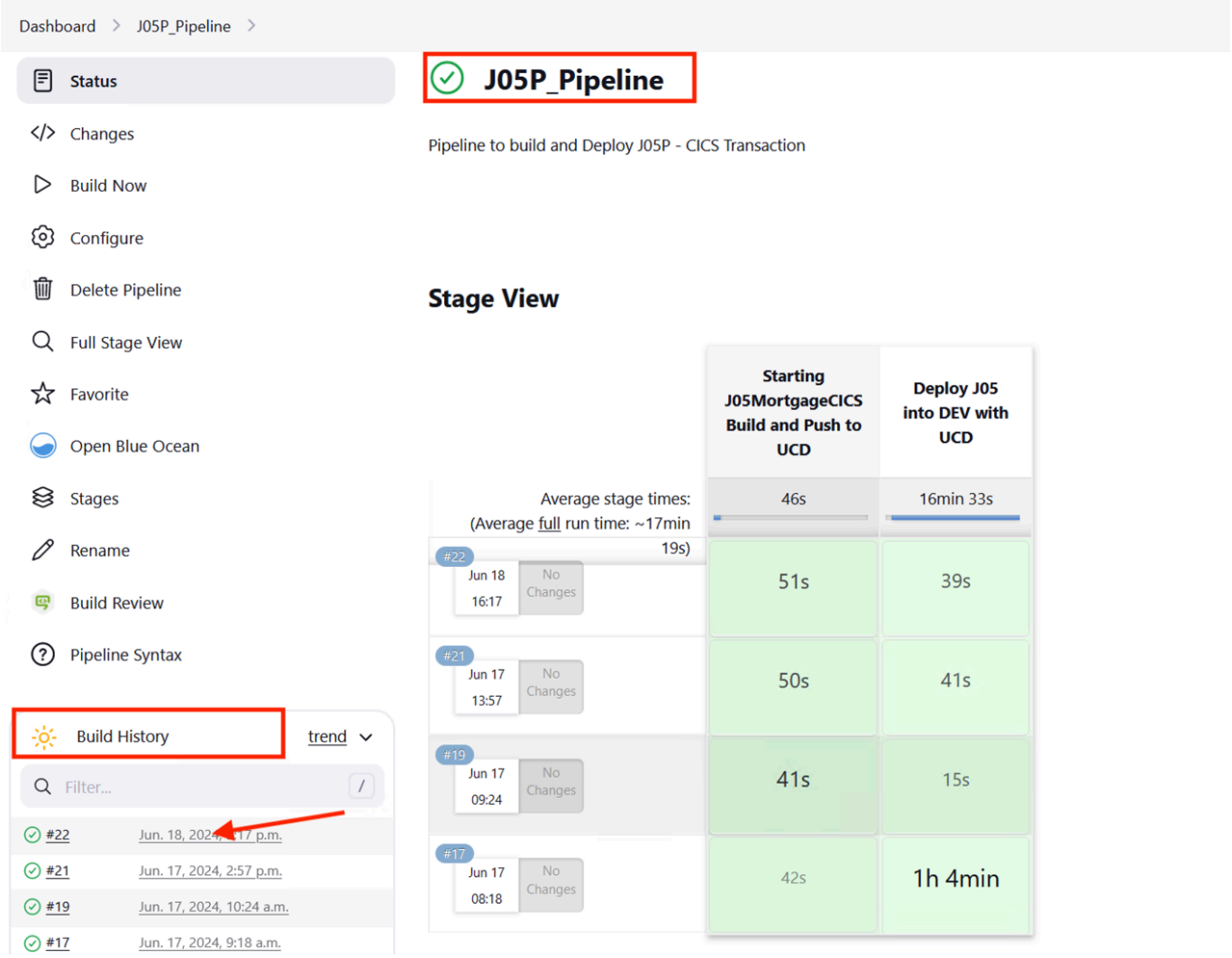


7.2.13 Minimize the browser if you will do the last optional step (#8) or close it.

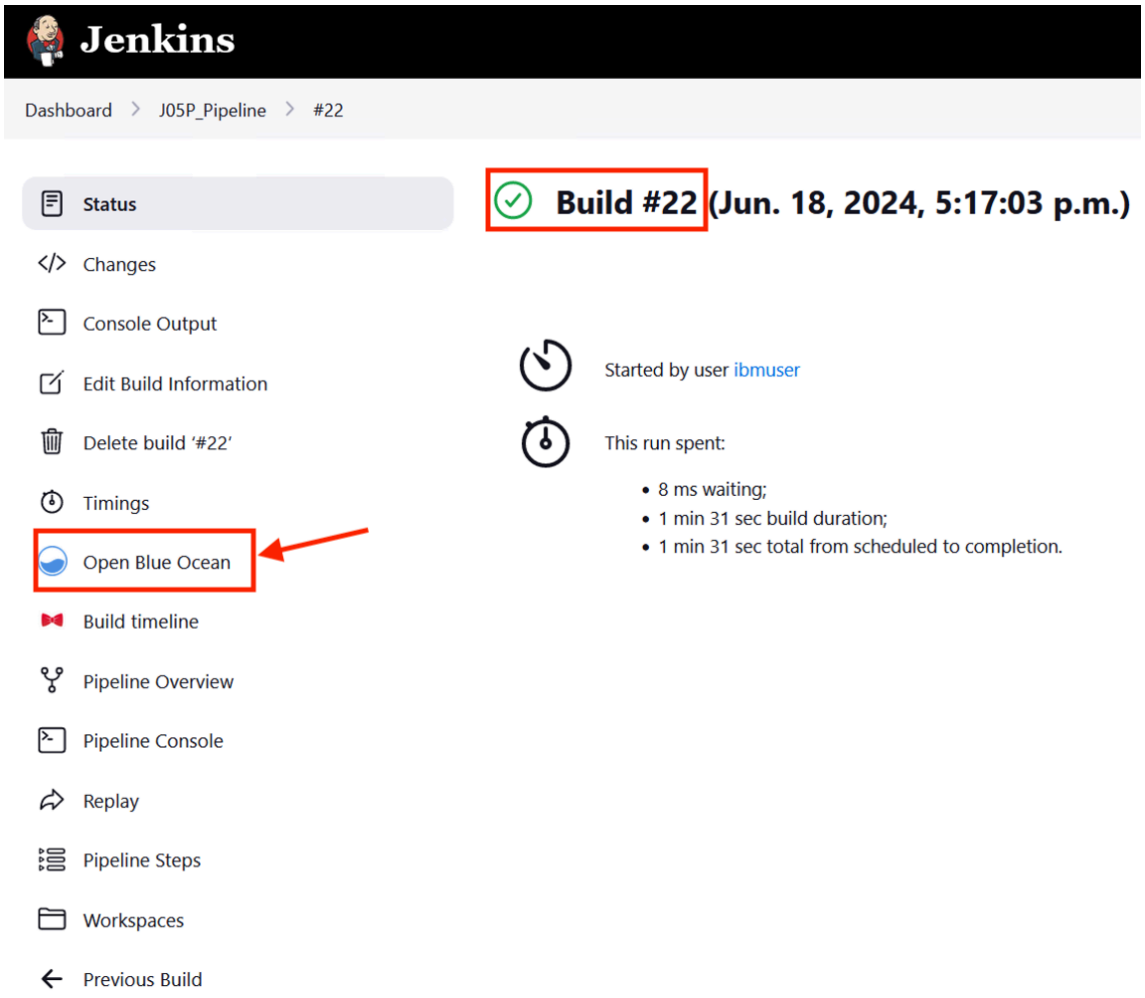
___ 7.3 Using Jenkins Blue Ocean Plugin

This is a nice plugin. We could have used this on the Jenkins activities. Log in Jenkins

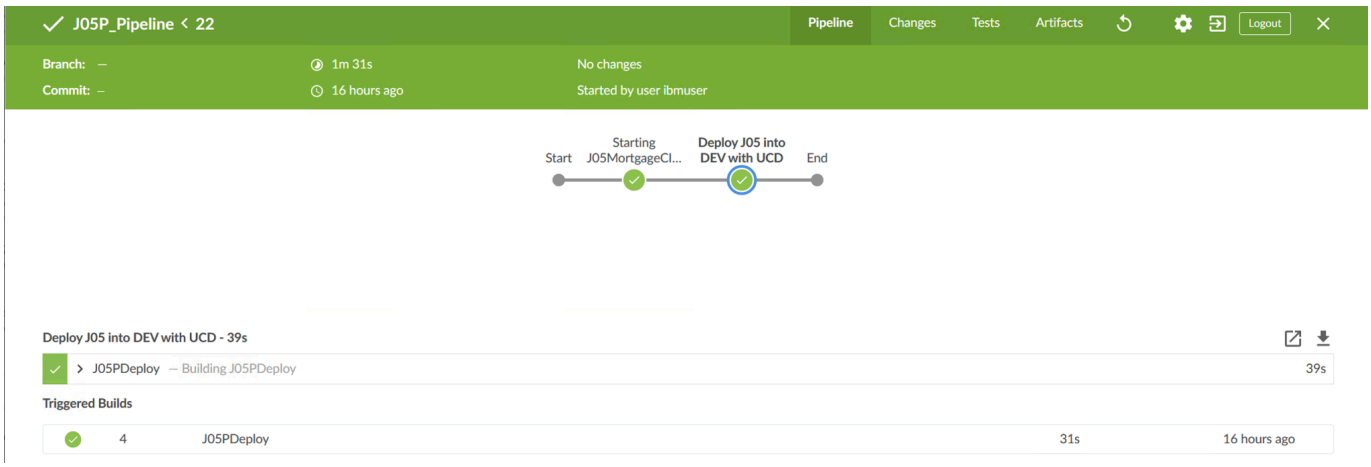
7.3.1 On J05P_Pipeline. Under Build History and **click** on the build number .



7.3.2 On the Build page you select. On the left **Click Open Blue Ocean**



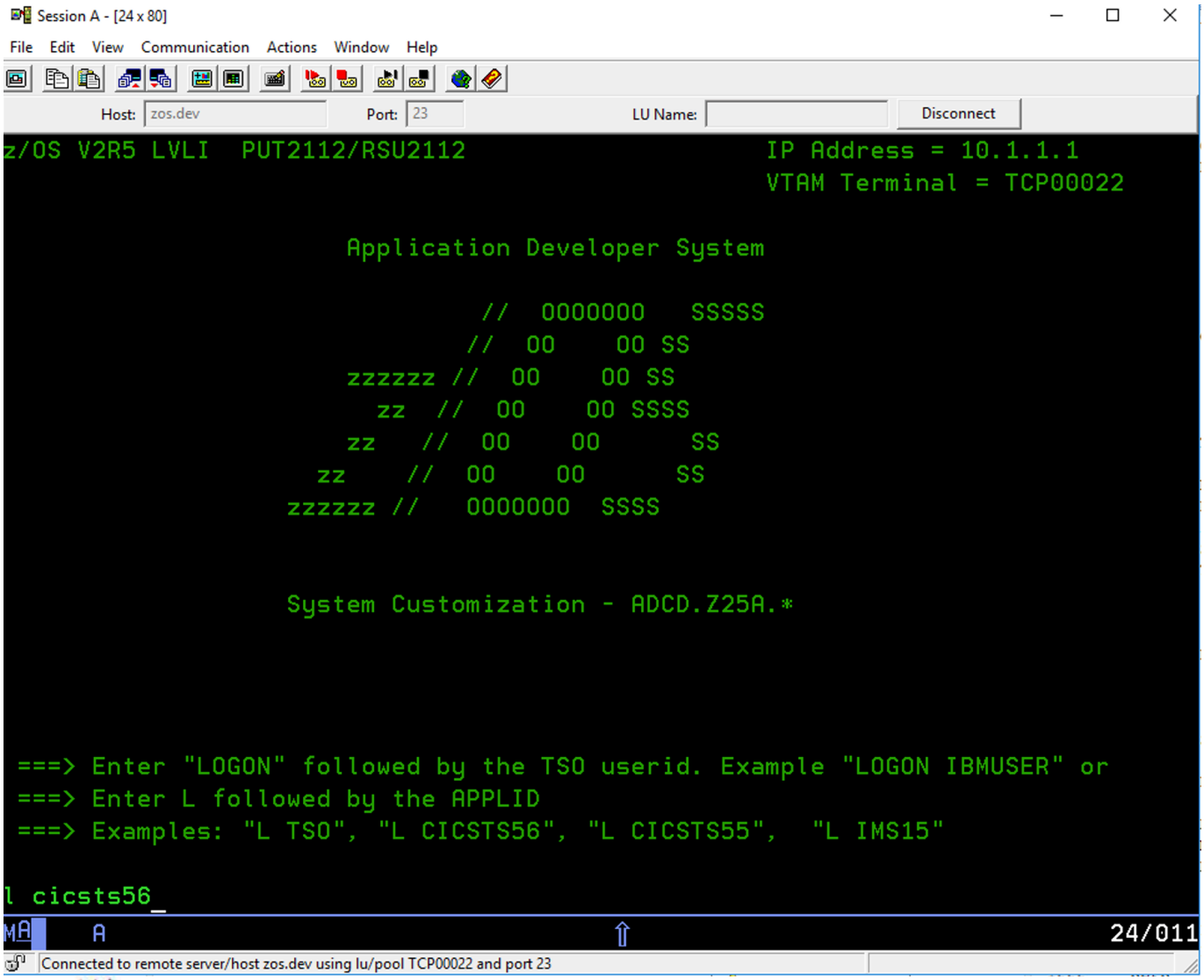
7.3.3 In this page shows the pipeline in another way. This plugin is handy when using “Jenkinsfile”



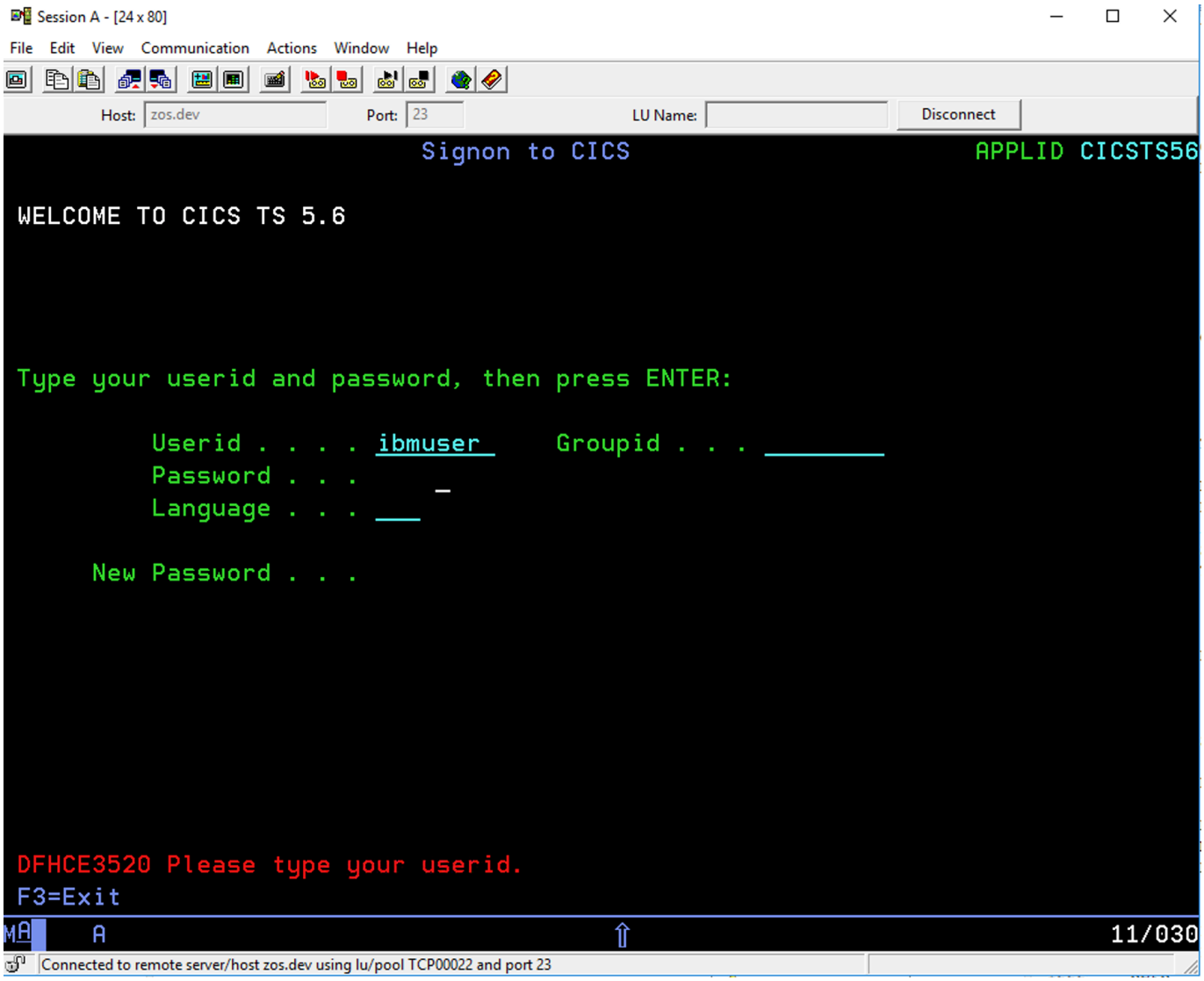
___ 7.4 Testing the CICS code deployed to Dev environment

7.4.1 Double click on the **3270 terminal emulator** that is on the Windows desktop

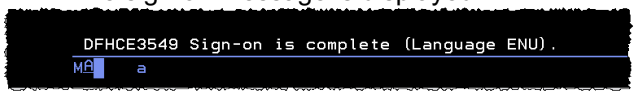
7.4.2 Type **l cicsts56**. (l as logon) and press **Enter** key. (**Ctrl** in some keyboards).



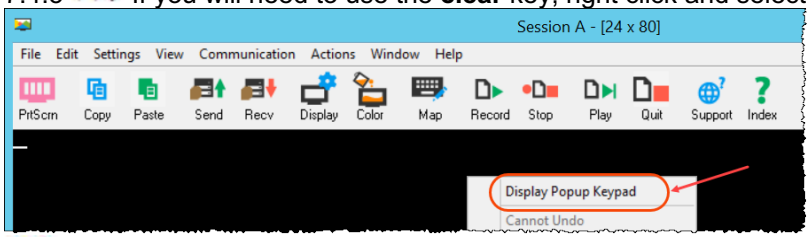
7.4.3 ⏭ Logon using your z/OS user id and password **(ibmuser and sys1)** and press **Enter**



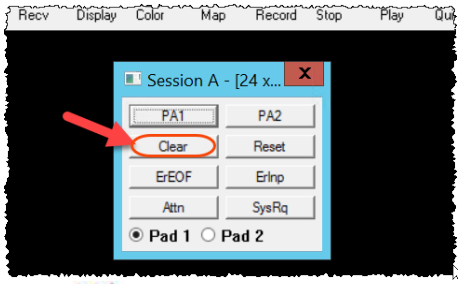
7.4.4 The sign-on message is displayed



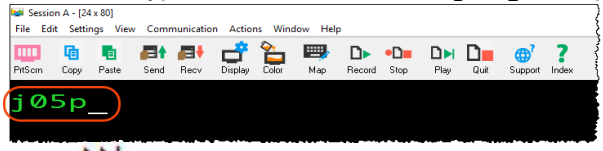
7.4.5 If you will need to use the **clear** key, right click and select **Display Popup Keypad**



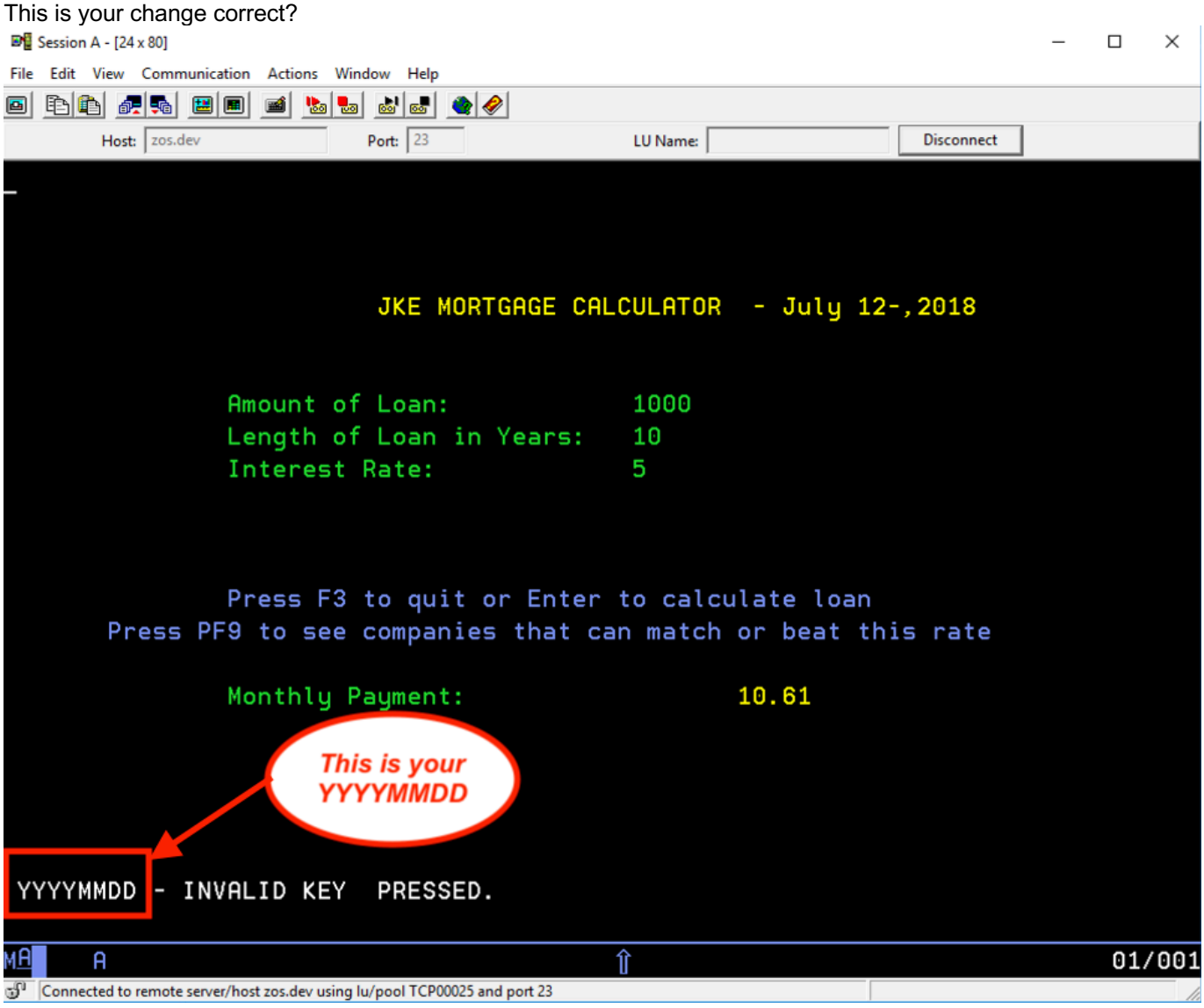
and then select **Clear**



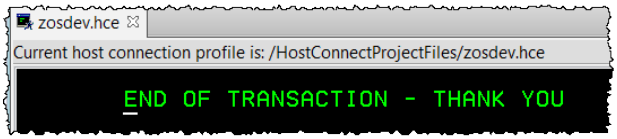
7.4.6 Type the CICS transaction **`j05p`** and press the **Enter** key.



7.4.7 Press **Enter** and then the **F1** key
and verify the message displayed "**YYYYMMDD - INVALID KEY PRESSED**" .
This is your change correct?



7.4.8 Press **F3** to end the CICS transaction.



7.4.9 Close the terminal emulator.

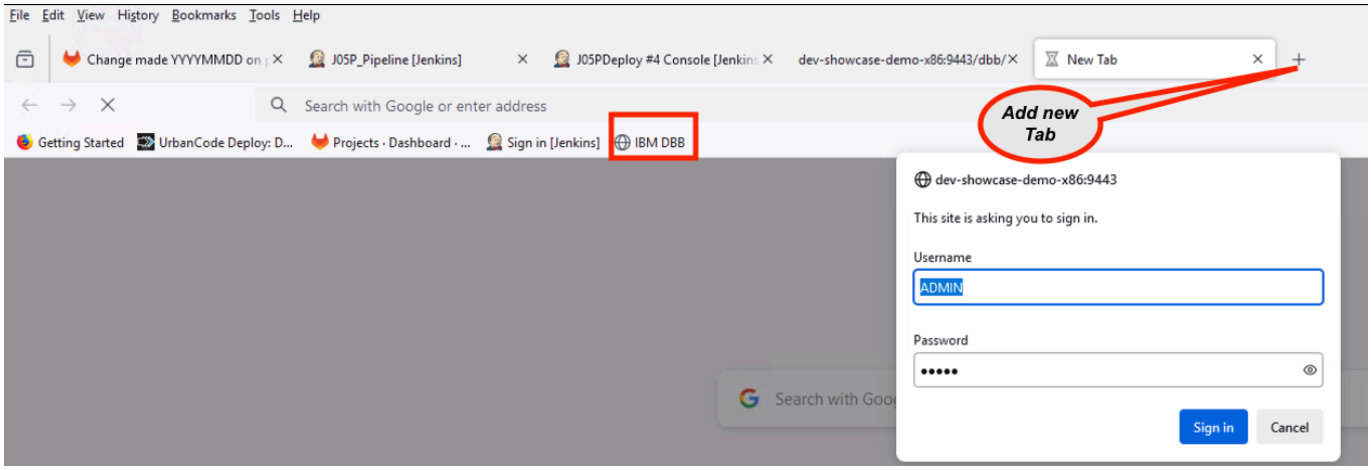
Section 8. Understanding DBB Build Reports

DBB has an application server (our is running on Linux) that capture the various build events. Here you will take a quick look of those reports.

___ 8.1 Login to the DBB server using the browser

8.1.1 ⏭ Using the Browser, add a new Tab (use "+"). On the new Tab click **Bookmarks Toolbar** click on **IBM DBB**

⏭ (If necessary) Type the user **ADMIN** and password **ADMIN** (uppercase) and click **OK**.



DBB Collections

In order to use the build dependency information collected by the *DependencyScanner* for dependency resolution and impact analysis, all scanned source files (both programs and dependency files) will need their resulting logical files stored in the DBB repository database as part of a dependency **Collection**.

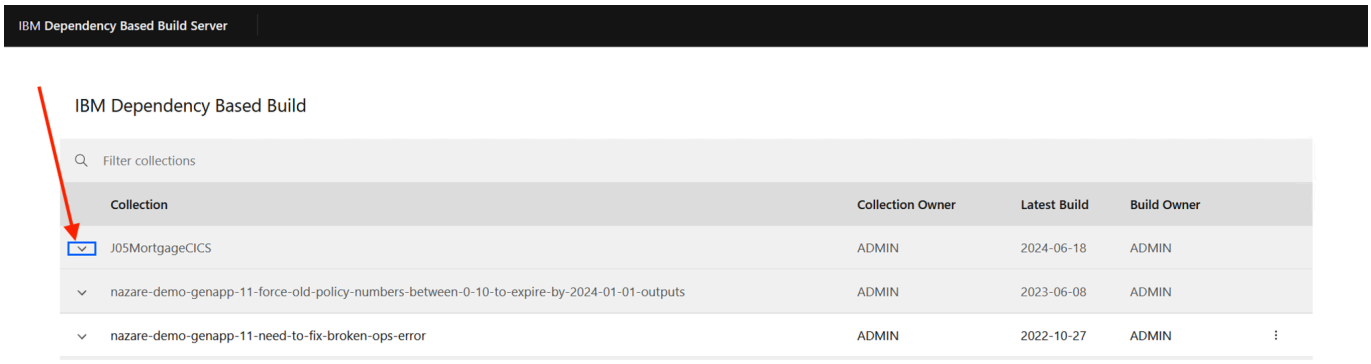
A collection is a repository container for logical files. The scope of a collection is determined by the user but generally a collection contains all the logical files of a Git branch. This way the logical files in a collection can use the scanned file's relative path from the *sourceDir* as a unique identifier in the collection.

Collections themselves can have any name but it is recommended to use the name of the Git branch of the source files being scanned

Collection names must be unique in the DBB repository database and an error will occur when trying to create a collection that already exists. A good practice it to first check if the collection with that name already exists before creating it

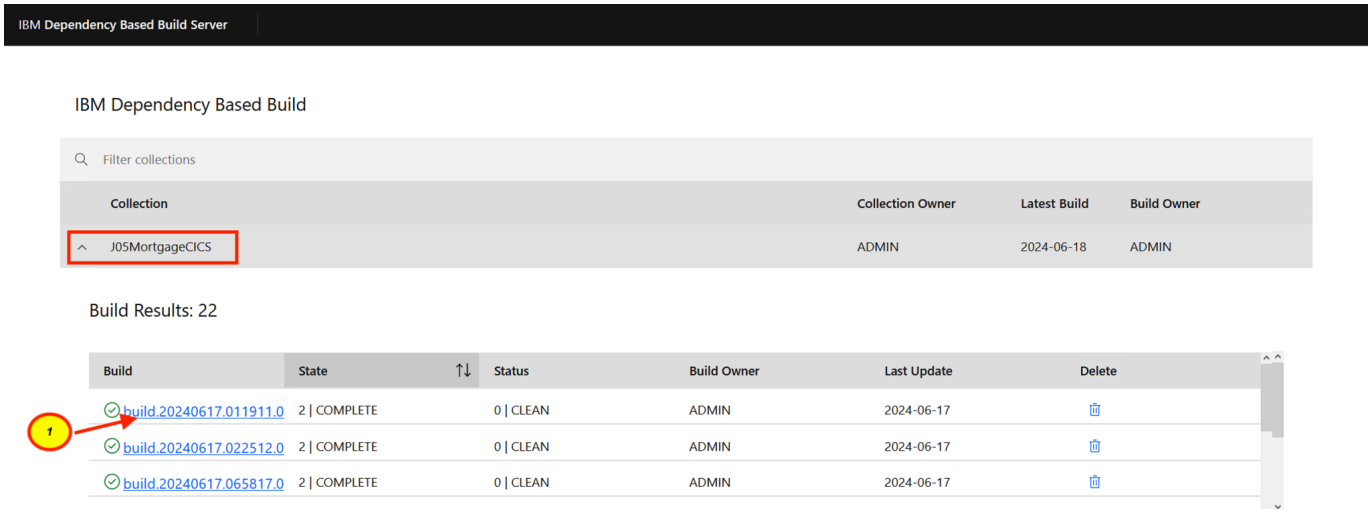
___ 8.2 Understand the DBB Collections

8.2.1 On IBM Dependency Based Build Sever. You see a list of collections ⏭ Click on the arrow of the collection name **J05MortgageCICS**



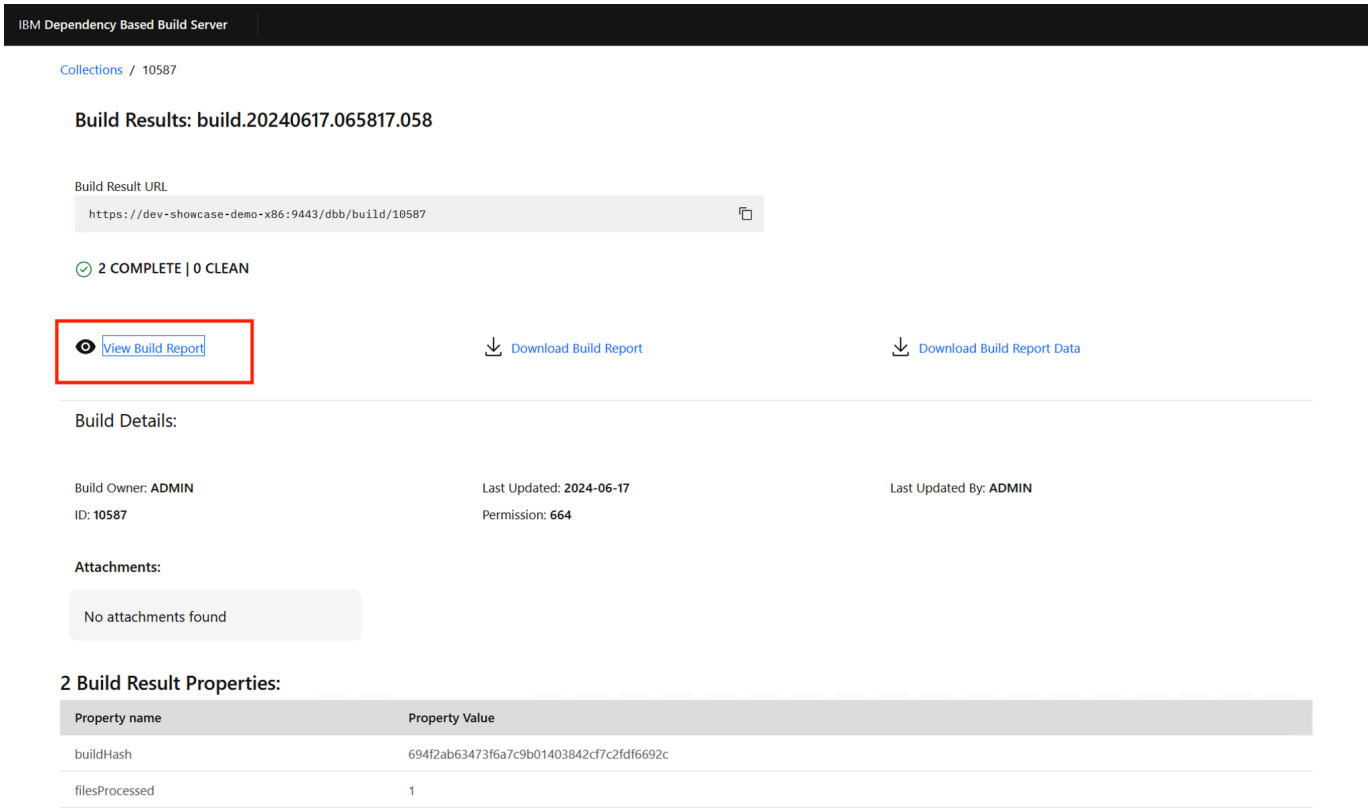
8.2.2 This opens the DBB Collection details..

⏭ Click the **link** for the recent build

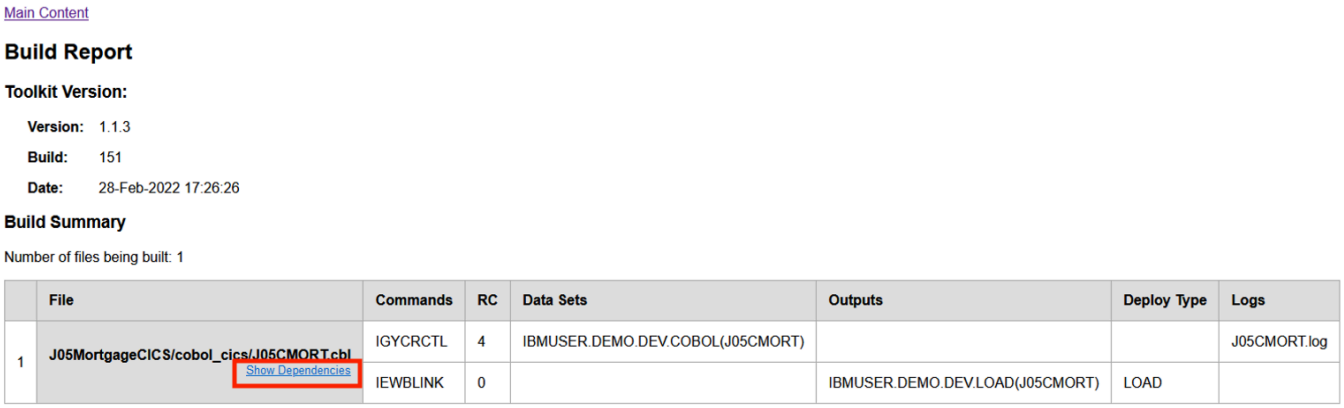


___ 8.3 Understand the Build Results

8.3.1 In this page,you see the build results. Click on **View Build Report** hyperlink,



8.3.2 This opens the *Build Report*.. Notice the load module created on this build
You also may explorer other hyperlinks. Click **Show dependencies**.



8.3.3 This opens the *Build Summary Report*.. Notice the files that are required for this build (dependencies)

.

Main Content

Build Report

Toolkit Version:

Version: 1.1.3

Build: 151

Date: 28-Feb-2022 17:26:26

Build Summary

Number of files being built: 1

	File	Commands	RC	Data Sets	Outputs	Deploy Type	Logs
1	**J05MortgageCICS/cobol_cics/J05CMORT.cbl** • J05MortgageCICS/copybook/J05NBRPM.cpy COPY • DFHAID COPY • J05MortgageCICS/copybook/J05MORT.cpy COPY • J05MortgageCICS/copybook/J05MTINP.cpy COPY • J05MortgageCICS/copybook/J05MTOUT.cpy COPY • J05MortgageCICS/copybook/J05MTCOM.cpy COPY Hide Dependencies	IGYCRCTL	4	IBMUSER.DEMO.DEV.COBOL(J05CMORT)			J05CMORT.log
		IEWBLINK	0		IBMUSER.DEMO.DEV.LOAD(J05CMORT)	LOAD	

8.3.4 ⏭ Close the browser.

Congratulations! You have completed the Lab 3. .

The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. This information is based on current IBM product plans and strategy, which are subject to change by IBM without notice. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.

Please Recycle