

IBM – CICS and DevOps Workshop



## **Lab – Git-based DevOps Pipeline for Z Application**

IBM Developer for Z

*Lab Version V1.1*

---

**October 2022**

Please send any comments on this lab exercise to:

George Ge: [ypge@au1.ibm.com](mailto:ypge@au1.ibm.com)

Lauren Li: [lauren.k.li@ibm.com](mailto:lauren.k.li@ibm.com)

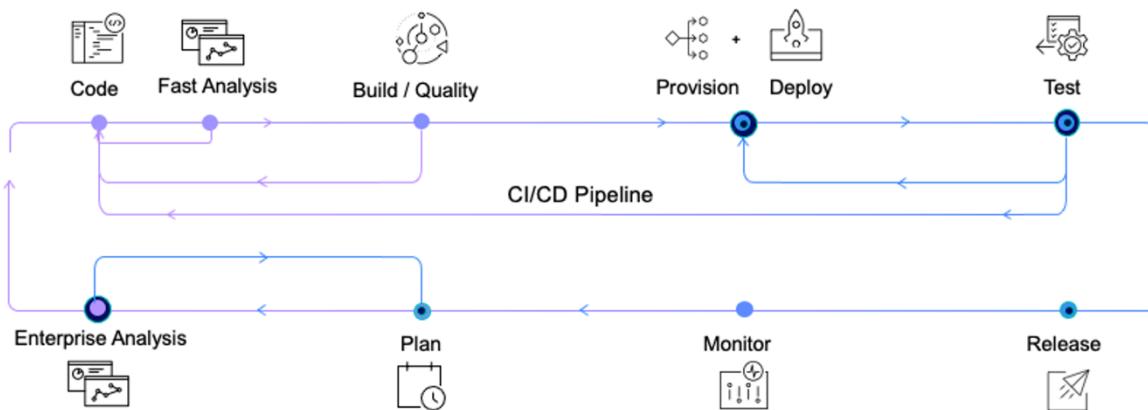
## Overview

DevOps is the enterprise capability of continuous software delivery to end-users. Its goal is to make business and IT operations more agile. It enables IT to deliver business requirements quickly and efficiently so that business operations can incorporate customer feedback and react swiftly to faster-changing market, economic and social conditions. DevOps is a union of people, processes, platform capabilities and tools that work together to make continuous software delivery possible.

The term “DevOps” is derived from the words “Development” and “Operations”. Originally, the term emphasized the importance of addressing one of the most critical impediments to application delivery process – lack of collaboration and insufficient automation in the processes between Development and Operations teams. Creating smoother integration and more efficient collaboration between Development and Operations teams are still one of the main goals of a DevOps solution, but DevOps methodology and solutions have gradually evolved into something much broader.

Enterprise digital transformation efforts look to capitalize on the hybrid cloud, the API economy, microservices and containers. This transformation is powered by software, and therefore, drives an urgent focus on innovation in the delivery of business value.

It requires software development teams to make use of agile, modern DevOps practices, and deliver applications via an integrated pipeline, the backbone of the value stream. Using the capabilities offered by the IBM Z DevOps portfolio, z/OS development teams can adopt standard software delivery practices, modernize core applications, and deliver a seamlessly integrated pipeline.



You can use a single heterogeneous pipeline to orchestrate the development, integration, and deployment of an application across multiple target platforms and environments. This pipeline must handle all stages of the process, from build tasks such as compilation and binds to artifact packaging, deployment, and larger scale integration tests. The pipeline can enforce gating mechanisms at each stage, ensuring quality.

## Advantages

Including IBM z/OS applications in DevOps transformation supports product teams in a single way of working, which simplifies the process and breaks down silos. It brings common tools, reducing the skills problem, and enables new talent to work with IBM Z assets by using tools and processes that they already know.

Focusing on a single heterogeneous pipeline allows for enterprise-wide visibility of the entire software delivery value chain. The business can deploy parts of an application to the best platform for the type of application. A single pipeline also allows the delivery organization to focus on common compliance, security, and audit controls, which can reduce cost and allow for comprehensive control across the lifecycle.

## **Lab introduction**

This lab will walk you through a sample DevOps workflow that covers the code, build, test and deploy stages using a GitLab CI/CD pipeline for a traditional CICS application. The application code is managed in a Git-based repository on the GitLab platform where the multiple developers can work in parallel.

### **Acknowledgements:**

We would like to thank Wilbert Kho for generously allowing us to use the DevOps on Z Proof of Technology (PoT) workbook as a basis for this lab document.

## **Scenario**

As a developer, you have been assigned to work on a GitLab issue that documents a defect that is causing incorrect prices for the items to be returned from the CICS Catalog Manager application for the APIs. You choose to use a modern IDE like Wazi for VS Code or IBM Developer for z/OS (IDz) to check out the code from a Git repository, create a new branch to work on, and use the debugging tool to interactively trace the execution. After identifying the root cause, you make the code change and attempt a user build in your own workspace and perform an early integration test. After it has been successful, you then commit the changes to the master branch on the remote Git repo which automatically triggers a pipeline to build the entire application and deploy the changes to the SIT environment. Lastly, you run an integration test to show the correct cost has been rectified in the API response payload.

## **Lab Requirements**

This lab uses GitLab as the source code repository and leverages GitLab CI/CD's pipeline capability to automate the different tasks that are associated with the build. However, the GitLab repository is interchangeable with other Git providers such as GitHub and Bitbucket. The GitLab CI/CD pipeline orchestrator is also interchangeable, with Jenkins being another alternative that is widely used.

The IDE we use in the lab can be **Wazi for VS Code** or the Eclipse-based **IBM Developer for z/OS**. We have two different versions of this lab document to cover each flavor of the IDE. Both documents will complete the same task and steps are equivalent between the two, just with a different user interface. You can choose either one based on your preference.

With the VS Code option, there are a few extensions that have been preinstalled for supporting Z application development.

- Git for Windows
- Zowe CLI and Zowe Explorer
- IBM Z Open Editor
- IBM Z Open Debug

On the z/OS side, we leverage IBM z/OS Debugger for interactive debugging sessions, IBM z/OS Dependency Based Build as the build framework, IBM Virtual Test Platform for early integration test, and of course, our target application Catalog Manager which is a CICS-supplied sample has been set up on CICS Transaction Server on z/OS.

We have also configured z/OS OpenSSH and RSEAPI for the integration with pipeline tasks.

We use z/OS Connect APIs to drive the CICS program for testing, but you can also use EGUI transaction which is the traditional 3270 BMS interface to run it too.

If you have any questions about the technology, please contact one of the lab instructors for assistance.

## **Lab Step Overview**

### **Part 1: Exploring the GitLab project and initiating a merge request for the issue**

Getting yourself familiar with the GitLab interface, confirm the defect with the API URL link attached. Then submit a merge request which creates a new branch to work in parallel just for this task.

### **Part 2: Create a clone of the new branch into your IDE**

In this step you will make a copy of the new branch into your IDE workspace. Navigate the project structure to understand the different components and configurations that are defined in the project as code.

### **Part 3: Debugging the application at statement level**

Enable the debugging option to intercept the program execution, then initiate another request to start the interactive debugging session, to exam the COBOL execution and identify the root cause for the issue.

### **Part 4: Make the code change**

In this part of the lab, you will learn how to explore the code editing capability in the IDE and fix the code.

### **Part 5: Run a user build to build the program in your own workspace**

In this part of the lab exercise, you will run a user build that will upload only the changed code with all its dependencies the z/OS environment and create a load module in your personal PDS dataset, with detail build result transferred back to your IDE for inspection.

### **Part 6: Early integration test of the application before deployment**

In this part of the lab exercise you will verify that your code change is now working and fixed the issue on the application level, without deploying into CICS.

### **Part 7: Commit the code and push to the remote Git repo to close the merge request**

In this step you are confident the code is ready, so you will commit it to your local Git repository and push it to the remote Git repo to submit the work you've done. This step will automatically trigger a full pipeline build.

You will dive into the different stages of the pipeline, review the build report that's published on the DBB webapp, deployment status and finally the integration test result from the API call.

### **Part 8: Summary**

This is a recap of the steps performed in this lab exercise.

## **Part 1: Exploring the GitLab project and initiating a merge request for the issue**

Getting yourself familiar with the GitLab interface, confirm the defect with the API URL link attached. Then submit a merge request which creates a new branch to work in parallel just for this task.

### **Log on to the GitLab dashboard**

1. From the desktop, double-click the Google Chrome icon to start the browser if it is not already running.



2. Click on the GitLab bookmark which opens the log in page.

A screenshot of the GitLab login page. At the top, there is a red header bar with the text "IBM CICS TS and DevOps workshop - October 2022". Below the header is the GitLab logo, a stylized orange cat icon. The main area has a light gray background. On the left, there is a sidebar with the text "A complete DevOps platform" and a brief description: "GitLab is a single application for the entire software development lifecycle. From project planning and source code management to CI/CD, monitoring, and security." Below this, it says "This is a self-managed instance of GitLab.". On the right is the login form. It contains two input fields: "Username or email" and "Password", both with placeholder text. Below the password field is a "Remember me" checkbox and a "Forgot your password?" link. At the bottom of the form is a large blue "Sign in" button. Below the form, there is a link "Don't have an account yet? Register now".

3. Use username `ibmdev` and password `Password` to login, you will see the dashboard for all the projects.

The screenshot shows the GitLab 'Projects' page. At the top, there's a navigation bar with a search bar labeled 'Search GitLab'. Below the navigation bar, the title 'Projects' is displayed, along with a 'New project' button. A filter bar allows filtering by 'Name'. The main area shows three projects:

- c** IBM Devs / **cics-catalog-manager** Owner DevOps CI/CD Pipeline Lab - CICS Catalog Manager. Status: ✓ ★ 0 ⚡ 0 Updated 1 minute ago.
- D** IBM Devs / **dbb-zappbuild** Owner. Status: ★ 0 ⚡ 0 Updated 1 month ago.
- Z** IBM Devs / **zcon-catalog-manager-api3** Owner z/OS Connect EE OpenAPI3 project for CICS Catalog Manager application. Status: ✓ ★ 1 ⚡ 0 Updated 1 week ago.

4. Click on the **cics-catalog-manager** project and you will be taken to the project homepage.

The screenshot shows the homepage of the 'cics-catalog-manager' project. The header includes the project name, a 'Project ID: 10' badge, and social sharing buttons for 'Star' (0) and 'Fork' (0). Below the header, it displays project statistics: 20 Commits, 2 Branches, 0 Tags, and 5.3 MB Project Storage. The description 'DevOps CI/CD Pipeline Lab - CICS Catalog Manager' is present. The repository navigation bar shows 'master' and a '+' button. Below the navigation bar, a commit history is shown for an update to 'COBOL/DFH0XVDS.cbl' authored by IBM Devs 1 week ago. A 'Clone' button is available. At the bottom, there are several buttons for README, Apache License 2.0, CI/CD configuration, Add CHANGELOG, Add CONTRIBUTING, Auto DevOps enabled, Add Kubernetes cluster, and Configure Integrations. A table at the bottom lists recent commits:

Name	Last commit	Last update
.vscode	Initial commit for CICS & DevOps Workshop	2 weeks ago
ASM	Initial commit for CICS & DevOps Workshop	2 weeks ago
ASMCOPY	Initial commit for CICS & DevOps Workshop	2 weeks ago

## View the project issues and confirm the defect

5. Click on **Issues** from the sidebar, you might need to expand the side bar to see the full menu.

6. On the right, click the "Incorrect item cost" issue that was created earlier.

Item	Description	Cost	Order
0010	Ball.Pens.Black.24pk.....	999.99	-
0020	Ball.Pens.Blue.24pk.....	999.99	-
0030	Ball.Pens.Red.24pk.....	999.99	-
0040	Ball.Pens.Green.24pk.....	999.99	-
0050	Pencil.with.eraser.12pk.....	999.99	-
0060	Highlighters.Assorted.5pk.....	999.99	-
0070	Laser.Paper.28-lb.108.Bright.500/ream...	999.99	-
0080	Laser.Paper.28-lb.108.Bright.2500/case..	999.99	-
0090	Blue.Laser.Paper.20lb.500/ream.....	999.99	-
0100	Green.Laser.Paper.20lb.500/ream.....	999.99	-
0110	IBM.Network.Printer.24.-.Toner.cart.....	999.99	-
0120	Standard.Diary;.Week.to.view.8.1/4x5.3/4	999.99	-
0130	Wall.Planner;.Erasable.36x24.....	999.99	-

7. Use the link included in the description to try the Catalog Manager APIs which shows all the item costs were wrongly marked as **999.99**

8. Go back to **GitLab** browser tab and scroll down the issue and click the **Create Merge Request** button to start a new branch called ***n-incorrect-item-cost*** (branch name is generated automatically and it can be a different from the screenshot) and a draft merge request that will be used to merge the work into the master branch when complete.

## New merge request

From 5-incorrect-item-cost into master [Change branches](#)

Title (required)

Draft: Resolve "Incorrect item cost"

Remove the `Draft` prefix from the title to allow this merge request to be merged when it's ready.

Add [description templates](#) to help your contributors to communicate effectively!

Description

[Write](#) [Preview](#)

---

Closes #5

B I § |≡ </> ⚡ |≡ |≡ |≡ |≡ |≡ |≡

---

Supports [Markdown](#). For [quick actions](#), type /.

Attach a file

9. Click **Assign to me** link for assignee and click the **Create merge request** button.

Assignee  
IBM Devs

Reviewer  
Unassigned

Milestone  
Milestone

Labels  
Labels

Merge options

Delete source branch when merge request is accepted.

Squash commits when merge request is accepted. (?)

**Create merge request** **Cancel**

- 10.** Now you have created a new branch in the remote Git repository, you can now start working on this branch which is isolated from any other development activities happening in the same repository. When your work is done, you will submit this merge request (which currently sits in draft state) to merge the changes to the master branch.

IBM Devs > cics-catalog-manager > Merge requests > !6

## Draft: Resolve "Incorrect item cost"

**Edit** **Code** **⋮**

**Open** IBM Devs requested to merge [5-incorrect-item-cost](#)  into [master](#) just now

**Overview** 0   **Commits** 0   **Pipelines** 1   **Changes** 0

Closes #5

This merge request contains no changes.

Use merge requests to propose changes to your project and discuss them with your team. To make changes, push a commit or edit this merge request to use a different branch. With [CI/CD](#), automatically test your changes before merging.



**Create file**

 0    0   

Oldest first **▼**

Show all activity **▼**

**Write** **Preview**

B I § |≡ </> ⚡ |≡ |≡ |≡ |≡ |≡ |≡

**Note:**

What is Git?

Git is a free and open source software for distributed version control: tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows (thousands of parallel branches running on different systems).

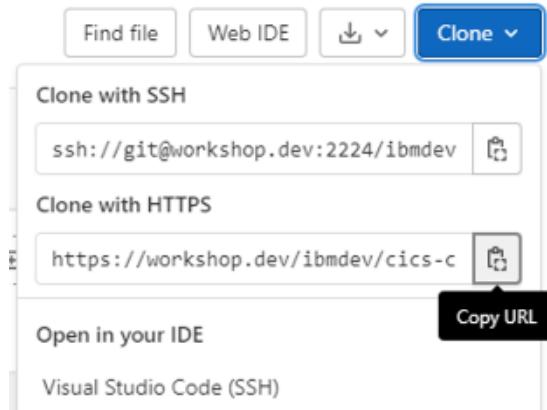
Git is very popular in the distributed world. In early 2017, Rocket Software ported Git into the mainframe – with the necessary checks to handle EBCDIC to UTF-8 conversions and vice-versa.

Git is not to be confused with GitLab or GitHub, which are commercial DevOps offerings consisting of Git-based SCM repositories with added features like bug tracking, continuous integration and team collaboration tools.

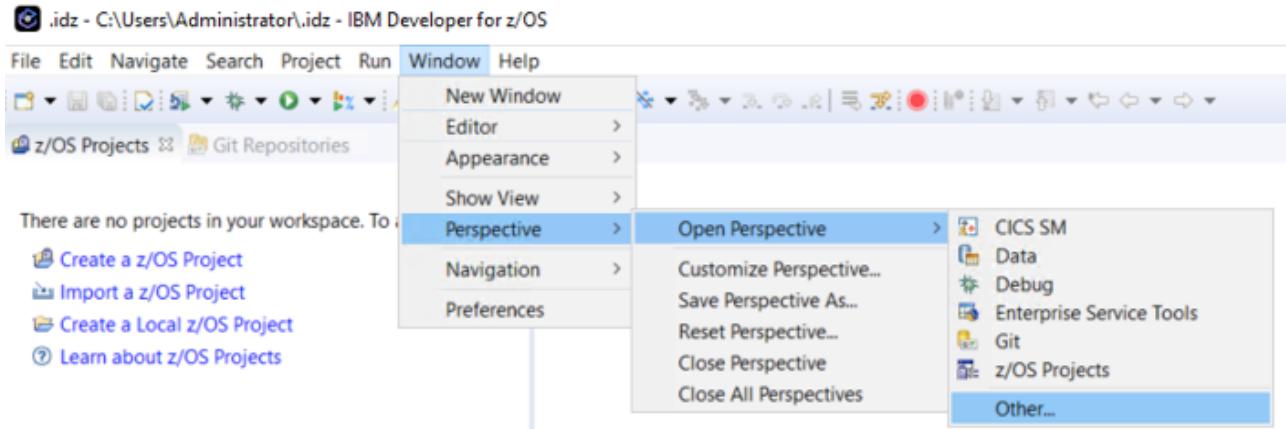
## **Part 2: Clone the git repository into your workstation using IDz**

### **Clone the Git repository into your workstation using IDz**

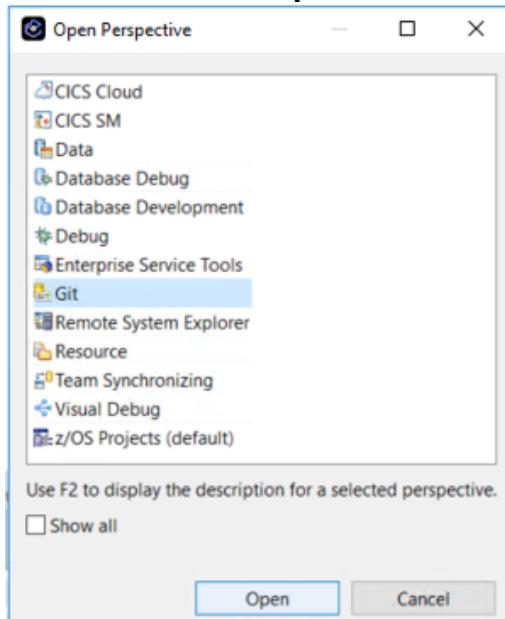
- \_\_\_ 1. In the GitLab page, click the project name on the top of the navigation bar to go back to the homepage.
- \_\_\_ 2. Click on the blue button "Clone" on the right, and then copy the URL below "Clone with HTTPS" by clicking the clipboard icon next to it.



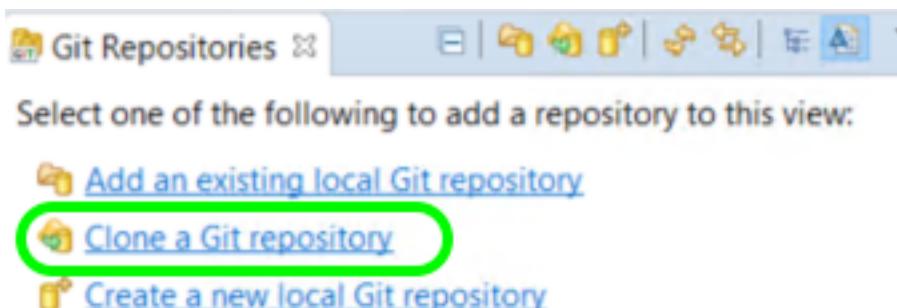
- \_\_\_ 3. Go back to IDz using the icon ( ) that is on the base of your screen.
- \_\_\_ 4. Open the Git perspective by selecting **Window > Perspective > Open Perspective > Other...**



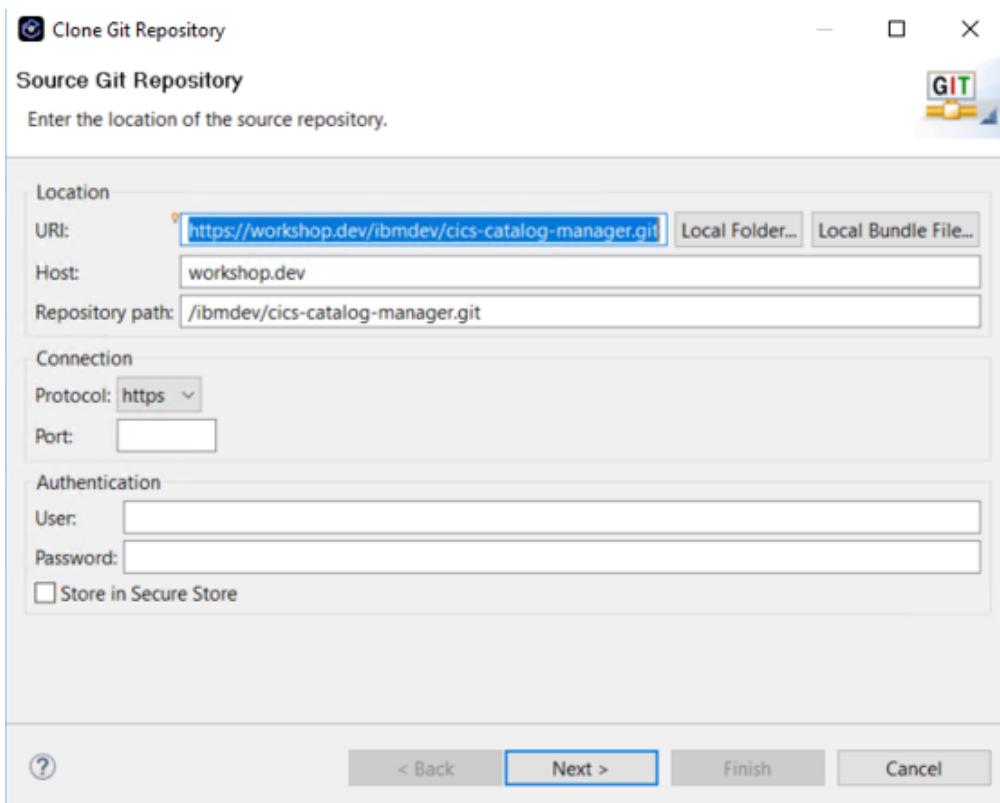
5. Select **Git** and click **Open**.



6. In the **Git Repositories** tab, click on the hyperlink to ‘Clone a Git repository’.



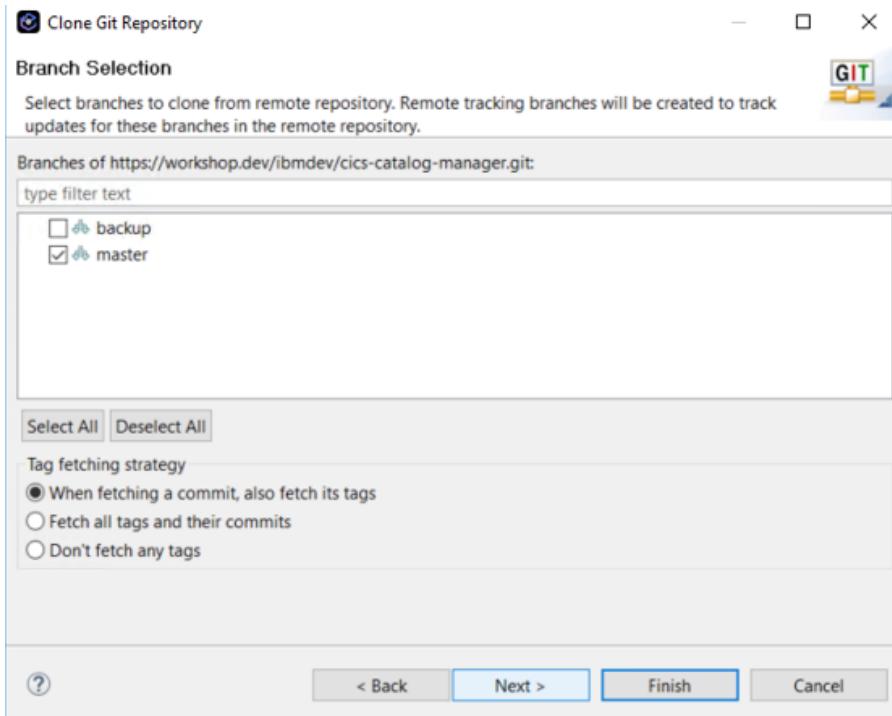
7. The values copied from the webpage (copied URL) will be shown in the Clone Git Repository window. Tip: In case you don't see it, go back to the page and copy it again (step 2b.2 above).



8. Click **Next**.

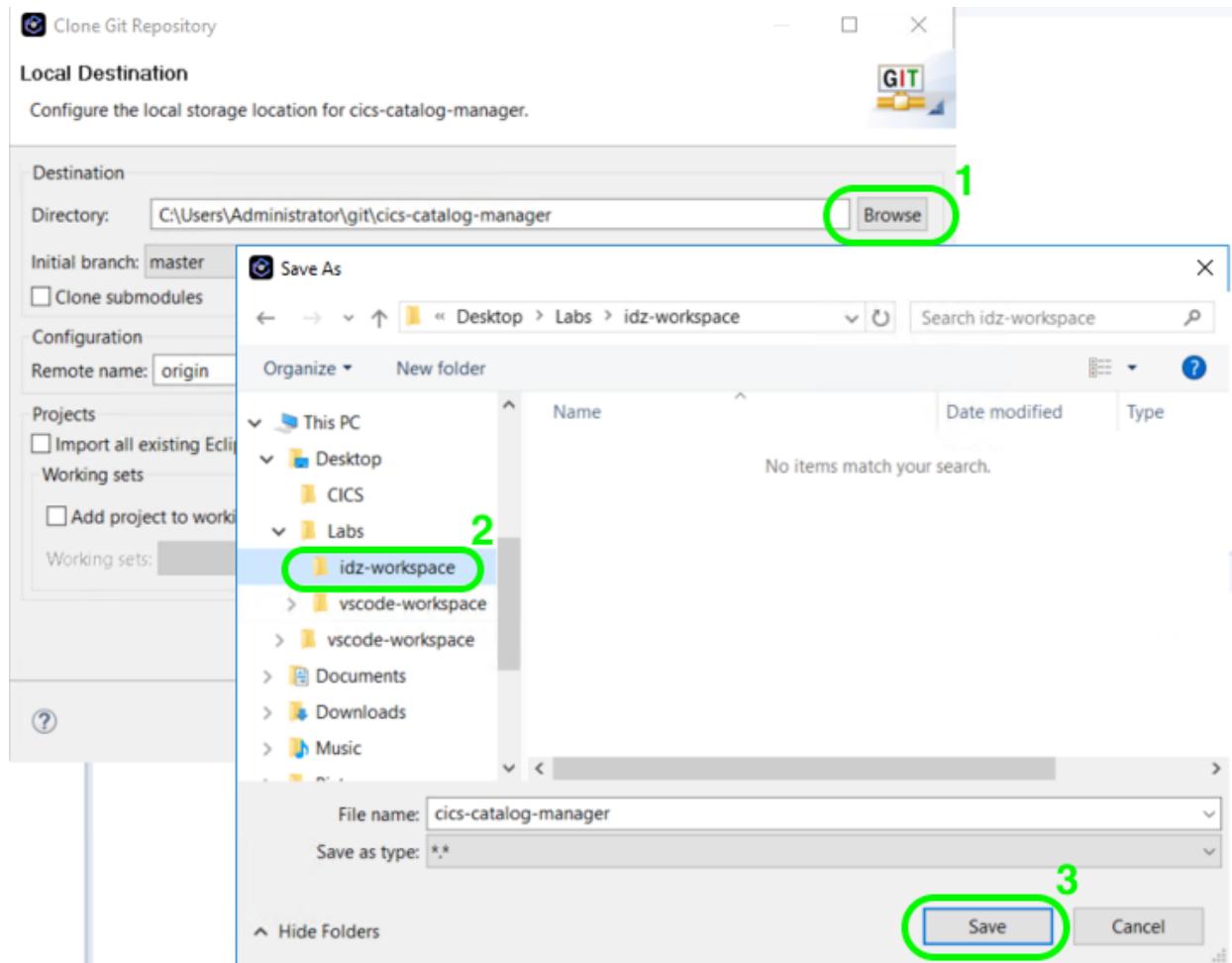
9. If a dialog asks for login, the credentials are user: **ibmdev** and password: **Passw0rd** to log in.  
Click **Log in**.

10. Ensure the branch named "master" is selected, and then click **Next**.



11. We will clone the repository on your local windows and import to be shown on IDz

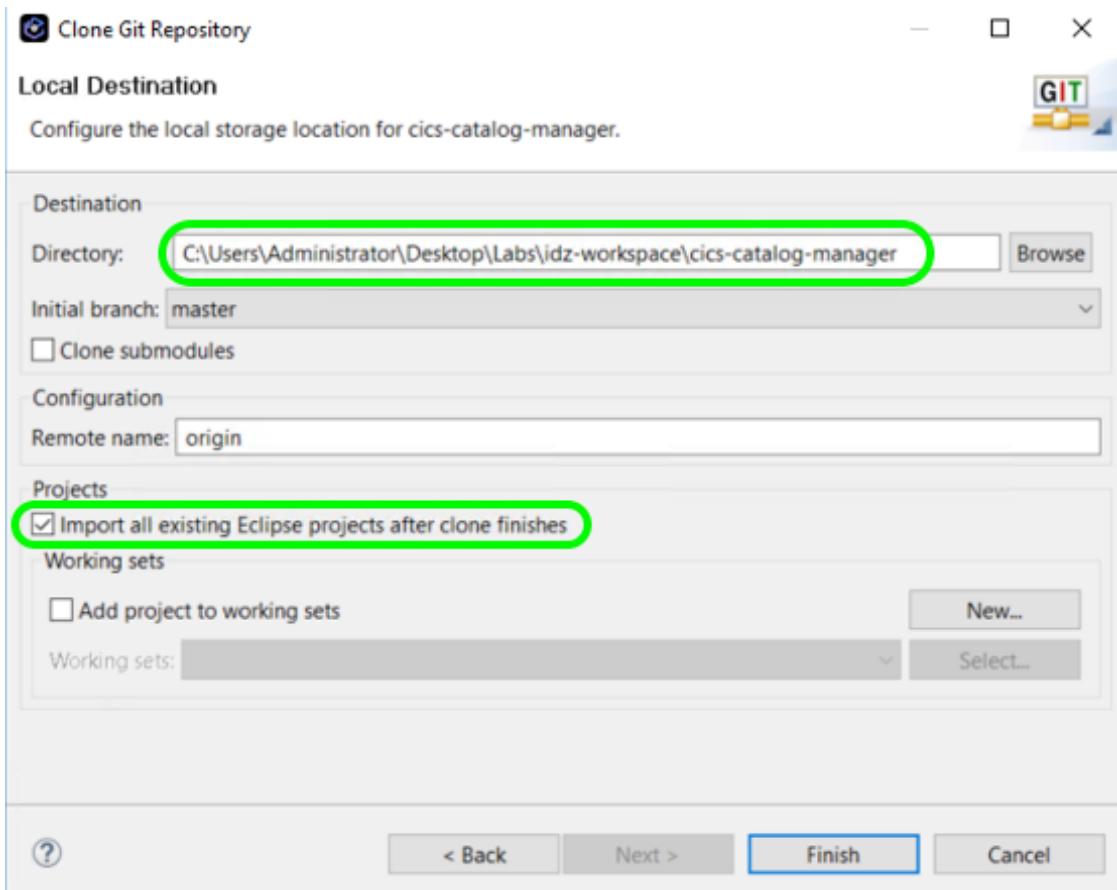
- Use **Browse** button (on the right) to select the directory to save the repository in.
- In the "Save As" pop-up window, use the file explorer in the left pane to navigate down to the folder `C:\Users\Administrator\Desktop\Lab\idz-workspace`.
- Click **Save**



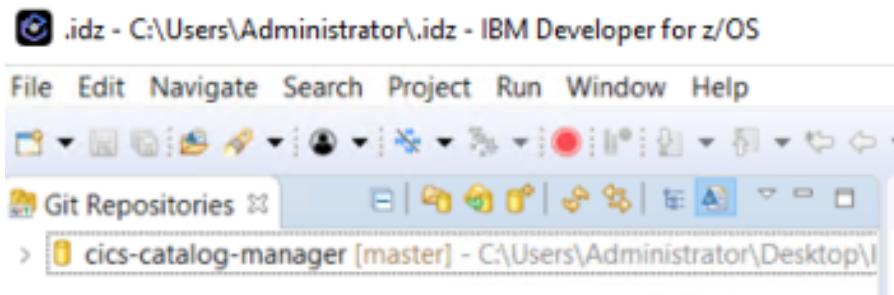
12. The Directory now should point to:

`C:\Users\Administrator\Desktop\Labs\idz-workspace\cics-catalog-manager`

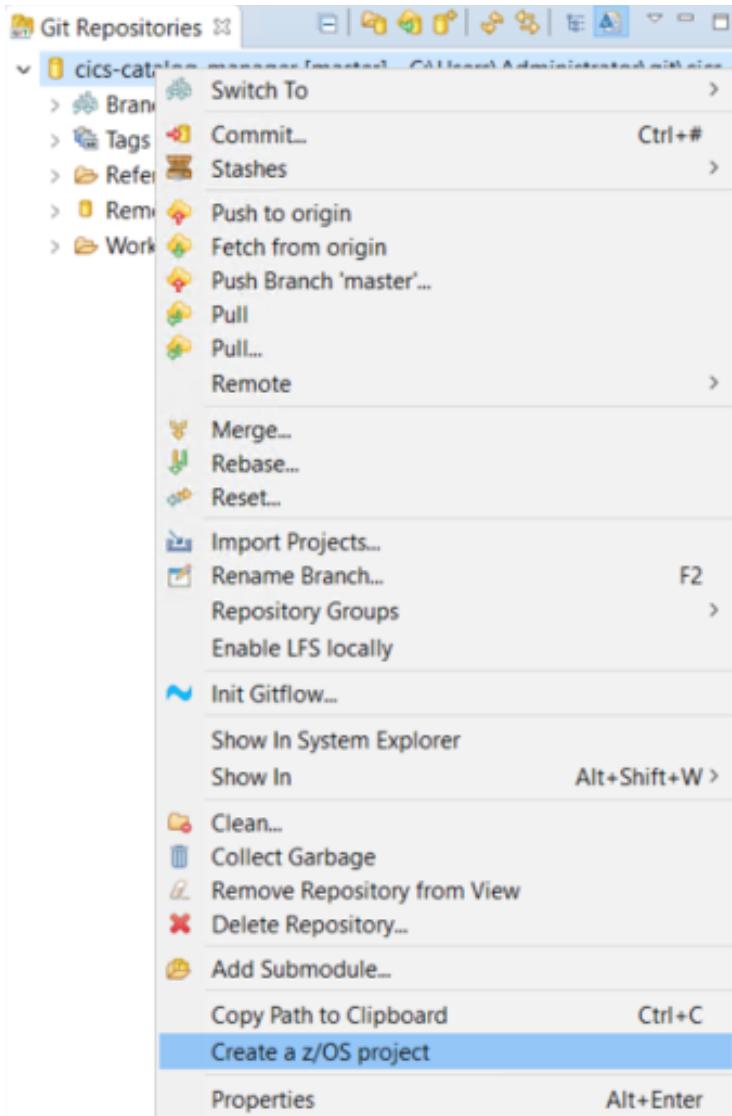
- Be sure that you selected **Import all existing Eclipse projects after clone finishes**
- Click **Finish**



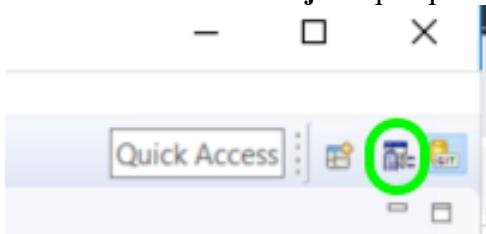
13. The Catalog Manager application will be cloned from the Remote repository and will appear in the Git Repositories view.



14. Optional: Explore the newly-cloned repository in the Git Repositories view by left-clicking the arrow icons on the left of the folder names.
15. Right-click the **cics-catalog-manager** application name in the Git Repositories view, and then select "Create a z/OS project" from the context menu.



16. Switch to the **z/OS Projects** perspective clicking on the  icon on the top right corner of IDz.

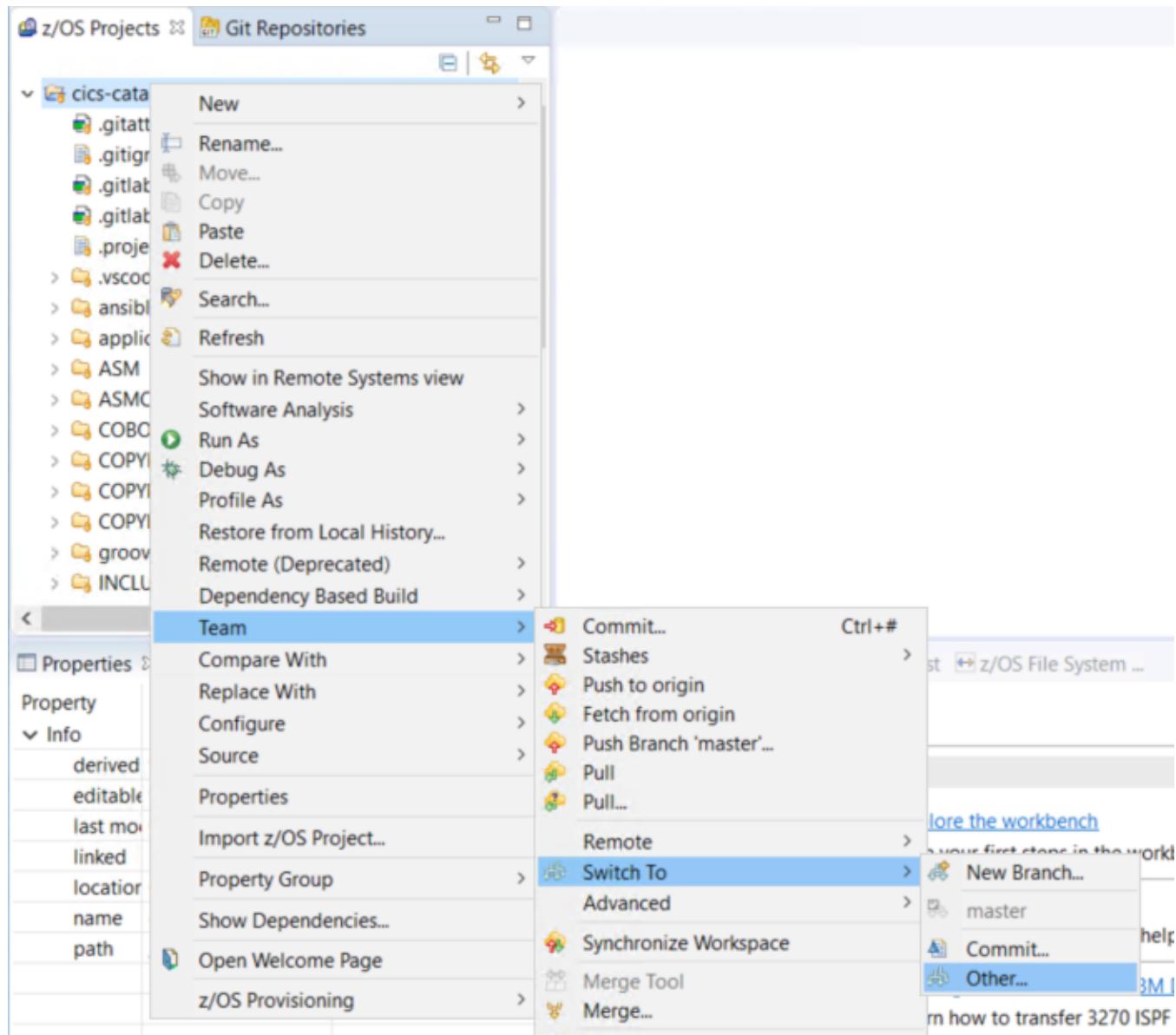


17. Expand the z/OS project "cics-catalog-manager" by clicking on the arrow icon next to the application name in the **z/OS Projects** view. You will see the source code loaded. Notice that IDz knows that this code is under a repository and the yellow decorator on the icon () indicates that.

## Switch to the issue's feature branch in IDz's z/OS Project view

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

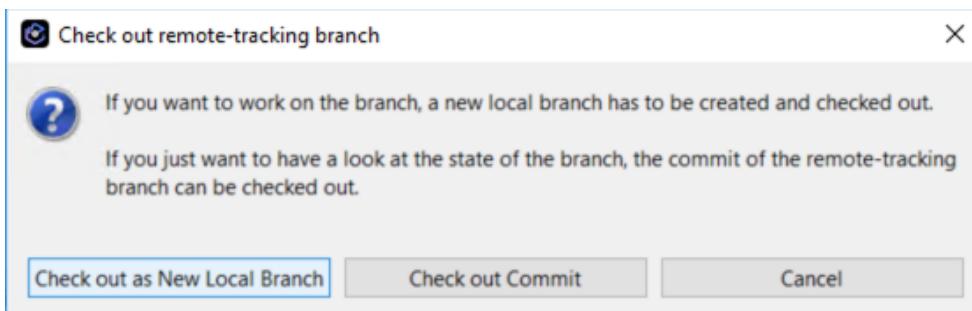
18. To switch to the new feature branch you created on GitLab to fix the defect, right-click on the application name "cics-catalog-manager" in the **z/OS Projects** view, then "Team" > "Switch To" > "Other".



**Note:**

- Switching to your issue's feature branch before beginning your code changes for the issue is important, as this is what allows you to work on the issue without having to worry about being disturbed by other development work on the repository. In other words, this feature allows you to work on the code in parallel with your team.
- To fetch the latest updates and branches from the central Git repository after the repository is already cloned into your local workstation, right click on the application name in the **z/OS Projects** view, and then select "**Team**" > "**Fetch from origin**" in the context menu.

19. In the pop-up window, expand the "**Remote Tracking**" folder, and select **origin/n-incorrect-item-cost** to checkout as the current branch to work on, where "n" is the issue number.
20. Click the "**Check Out...**" button.
21. In the new "**Checkout remote-tracking branch**" pop-up that appears, click "**Check out as New Local Branch**" to start working on a local copy of the feature branch.



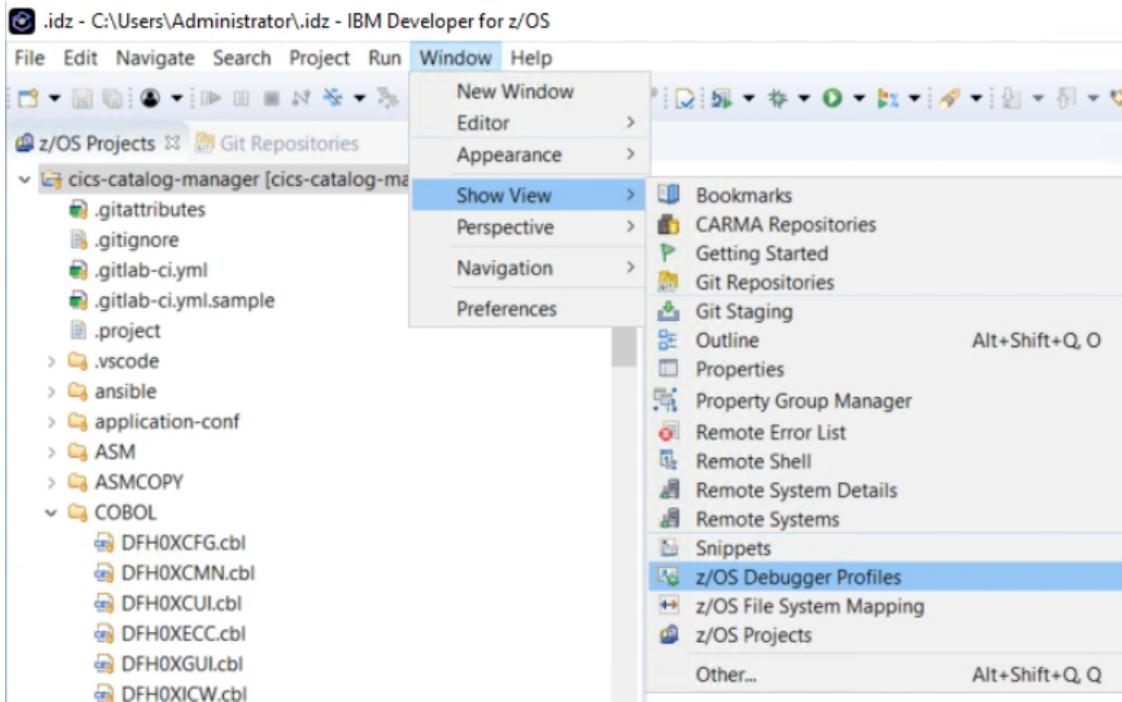
22. In the following "**Create Branch**" pop-up that appears, click "**Finish**". In the **z/OS Projects** view next to the application name, you should now also see the issue's feature branch name in parentheses.

### **Part 3: Debugging the application at statement level**

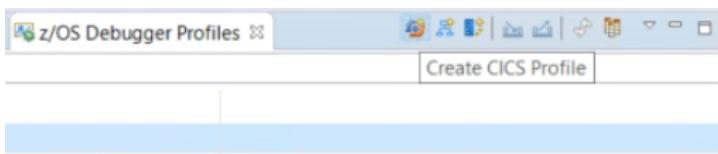
We know our program has some defects with the incorrect item cost, but the best way to identify the root cause is to use the debugger and trace the program line-by-line when in execution. In this part of the lab we will use the IBM z/OS Debugger remote debugging capability, set up a debug profile which intercept any call to the main catalog program DFH0XCMN and start the remote debugging session in the IBM Developer for z, so that developer can use graphic user interface to inspect and control the execution.

## Creating the Debug Profile

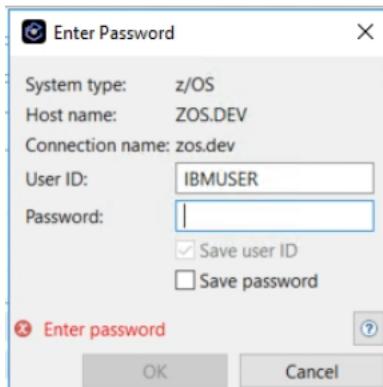
23. In the IDz window, click **Window** → **Show View** → **z/OS Debugger Profile** to open the **z/OS Debugger Profiles** view.



24. Next click the first button in the **z/OS Debugger Profiles** view which is **Create CICS Profile**.



25. If the **Enter Password** dialog popes up, use **IBMUSER** and **SYS1** to log in to the remote z/OS.



26. In the profile editor you will select what you want to debug. It's now connected to the z/OS and listing all available CICS regions, select **CICSTS56** for the Region. In the **CICS filters** section you can define a few criteria of the CICS application you want to debug, things like transaction ID, user ID, SYSID, as well as load modules or a combination.

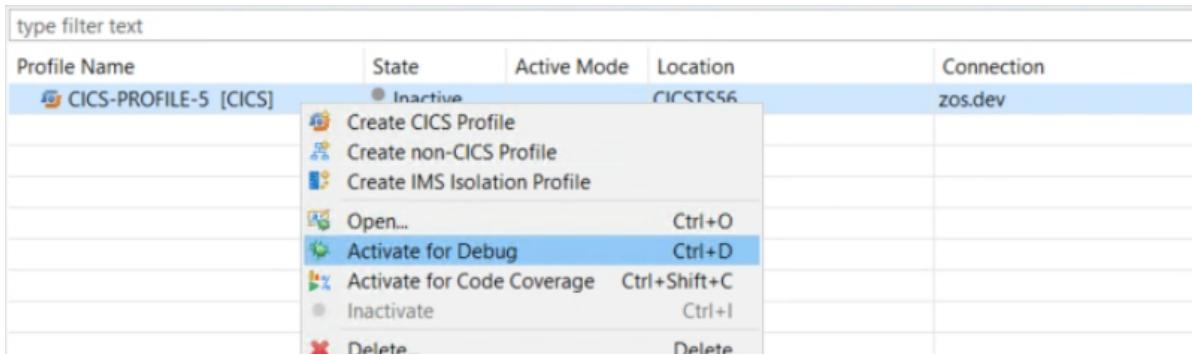
We will double-click **Add new item** button in the **Load Modules** table and specify load module name **DFH0XCMN** then press Enter. So that anytime our catalog manager program is invoked, it will be suspended and parked aside for you to start take control and debug interactively.

Load Module	Compile Unit
DFH0XCMN	*

27. Leave all the other fields as default and click the save button at the top to finish, you can see a new profile with a name **CICS-PROFILE-n** has now been defined with **inactive** status in the **z/OS Debugger Profiles** view.

z/OS Debugger Profiles					
Profile Name	State	Active Mode	Location	Connection	Description
CICS-PROFILE-5 [CICS]	<input checked="" type="radio"/> Inactive		CICSTS56	zos.dev	

28. Right-click on the profile, select **Activate for Debug** from the menu, click **Yes** to confirm and now it should be successfully activated.



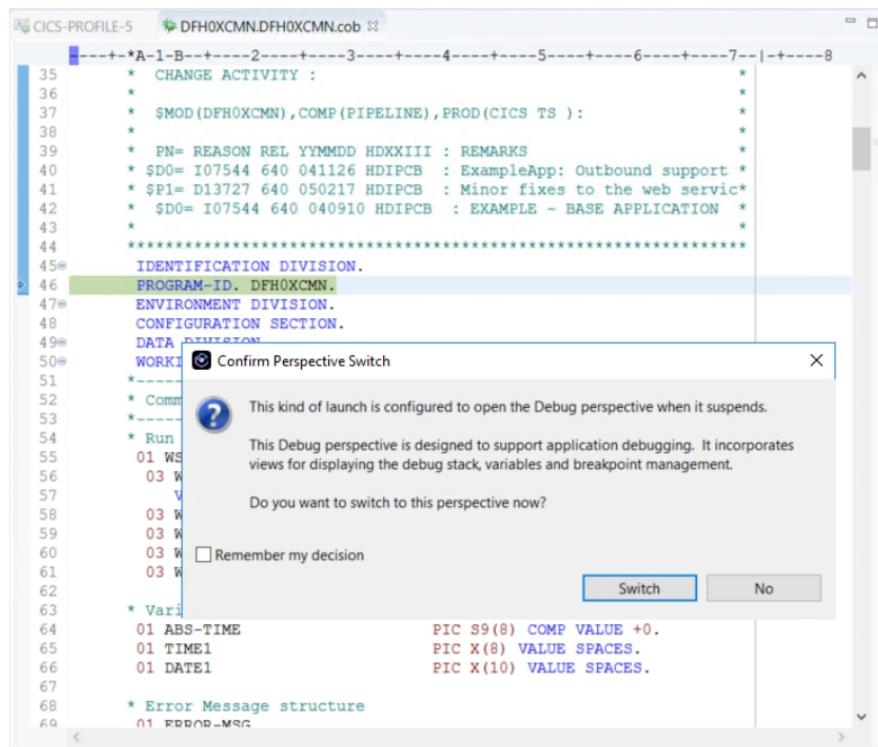
## Invoke the program to start debugging

29. Now that your debug profile for **DFH0XCMN** is activated. Go back to the browser and trigger the API call once again using the URL:

<https://workshop.dev:9443/items?startItemID=0>

You will notice the page is loading as the API is not responding this time since the backend program has been intercepted.

Go back to the IDz and you will see the pop-up dialog for switching to the **Debug Perspective**. Click **Switch** to continue.



30. The debugging session now opens, you could find source code inside the editor, see variables, call stack and watch expressions as you step thru the code. Similar to the debugging experience for other programming languages, you can step through a running program, check variable values, set breakpoints etc.

```

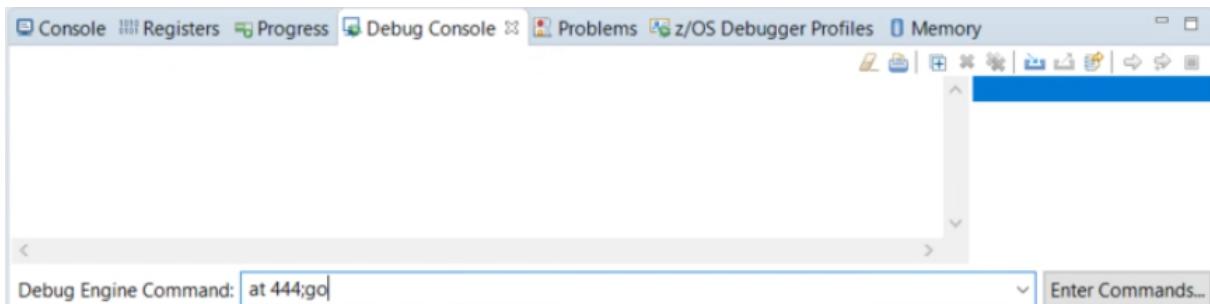
303
304  PROCEDURE DIVISION.
305
306  -----
307  MAINLINE SECTION.
308
309  -----
310  * Common code
311
312  * initialize working storage variables
313  INITIALIZE APP-CONFIG.
314  INITIALIZE WS-PROGRAM-NAMES.
315  INITIALIZE ERROR-MSG.
316
317  * set up general variable
318  MOVE EIBTRNID TO WS-TRANSID.
319  MOVE EIBTRMID TO WS-TERMINAL-ID.
320  MOVE EIBTASKN TO WS-TASKNUM.
321
322  * Check commarea and obtain required details
323
324  * If NO commarea received issue an ABEND
325  IF EIBCALEN IS EQUAL TO ZERO
326  MOVE 'NO COMMAREA RECEIVED' TO EM-DETAIL.
327  PERFORM WRITE-ERROR-MESSAGE
328  EXEC CICS ABEND ABCODE('EXCA') NODUMP
329  END-EXEC
330  END-IF
331
332
333  * Initialize commarea return code to zero
334  MOVE '00' TO CR-RETURN-CODE

```

31. You can explore the interface by:

- Try to press **F6** key multiple time to step over the program line by line.
- Hover the mouse over any variable names to monitor its real time value, or select the variable name and right-click to add into **Monitor** panel.
- Right-click in the area before the line number you can choose to set or clear breakpoint on that line.

32. Find **Debug Console** view at the bottom of the editor. type **at 444;go** and **Enter** in the Debug Engine Command text area to set a breakpoint on line 444 and continue.



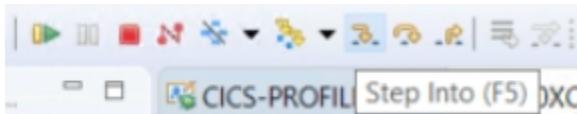
33. The program will resume execution and pause again just before linking into another CICS program that handles the catalog inquire function.

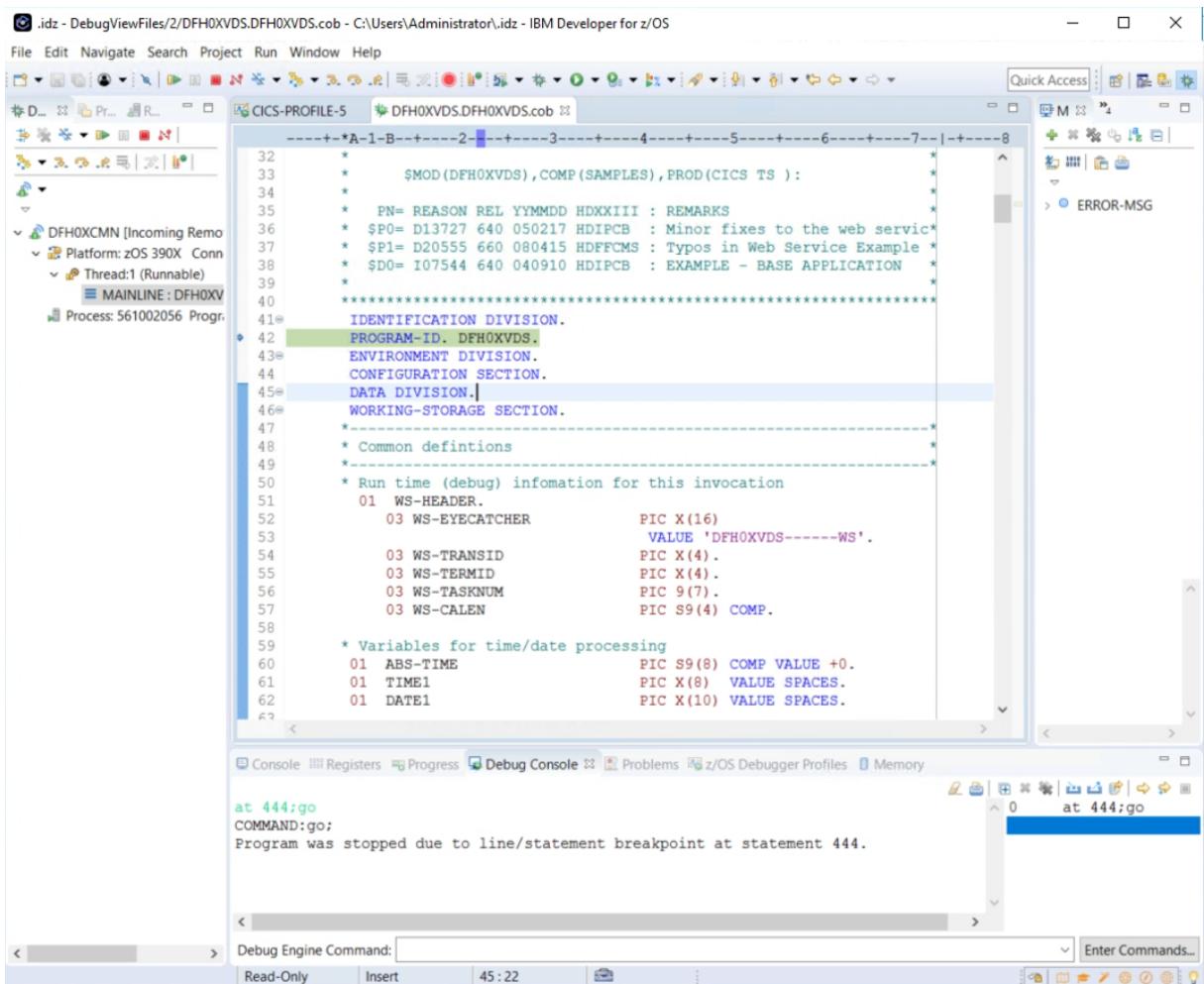
```

439      *=====
440      * Procedure to link to Datastore program to inquire
441      *   on the catalog data
442      *=====
443@     CATALOG-INQUIRE.
444      MOVE 'EXCATMAN: CATALOG-INQUIRE' TO CA-RESPONSE-MESSAGE
445      EXEC CICS LINK PROGRAM(WS-DATASTORE-PROG)
446          COMMAREA(DFHCOMMAREA)
447          END-EXEC
448          EXIT.
449

```

34. Press F5 or click Step-Into button twice from the top menu bar to trace into the subprogram, and we will see we are now debugging in the program DFH0XVDS.





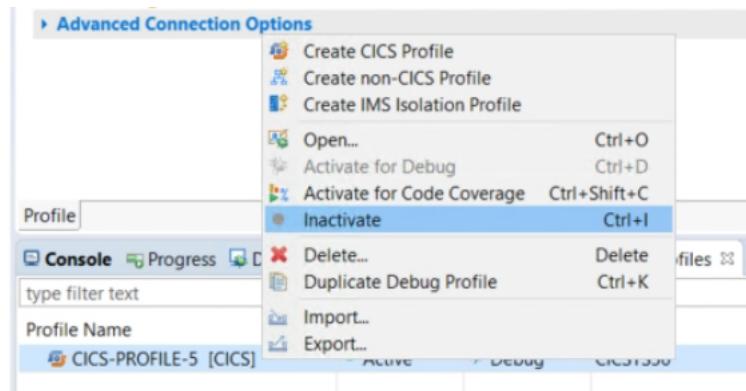
- 35.** Type `at change WS-COST` and **Enter** in the **Debug Console / Debug Engine Command** at the bottom of the screen to define a dynamic breakpoint only when the content of WS-COST variable is modified.

Press **F8** or the green **Resume** button  about 3 times until reaching **line 401** where you can clearly see the cost has been overwritten with a value 999.99 by a MOVE statement above which is the root cause of our issue here.

```
395          EVALUATE WS-RESPONSE-CODE
396              WHEN DFHRESP(NORMAL)
397                  MOVE WS-LOOP-COUNTER TO WS-RECORD-COUNT
398          *      this is a bug - start
399                  MOVE 999.99 TO WS-COST IN WS-CAT-ITEM
400          *      this is a bug - end
401          MOVE WS-CAT-ITEM TO CA-CAT-ITEM(WS-LOOP-COUNTER)
402
403          MOVE WS-RECORD-COUNT TO CA-ITEM-COUNT
404          MOVE WS-CURRENT-ITEM-REF TO CA-LAST-ITEM-REF
405
```

## Stop the debug session and deactivate the debug profile

- 36. Now we know the root cause of the defect. We now stop the debugging session by press **Ctrl-F2** or click the red stop button  , now you are disconnected the debugging.
- 37. **IMPORTANT.** Don't forget to deactivate the debug profile in the **z/OS Debugger Profiles** view. Otherwise, the program will be suspended for debugging again when we later test our program. Right-click on the profile and select **Inactivate** to prevent any future debugging session.



## Part 4: Make the code change

In this part of the lab, you will learn how to explore the code editing capability in IDz to fix the code.

### Open the source code in the editor

- 1. In the **z/OS Projects** View, left-click on the "cics-catalog-manager" application to expand it if it is not already expanded.
- 2. Navigate to the **COBOL** folder and double-click DFH0XVDS.cbl to open it. This is the program that you will update.

The screenshot shows the z/OS Projects view on the left and the DFH0XVDS.cbl editor window on the right. The editor window displays the following code:

```

-----+--A-1-B---+--2---+--3---+--4---+--5---+--6---+-
2      ****
3      *
4      * MODULE NAME = DFH0XVDS
5      *
6      * DESCRIPTIVE NAME = CICS TS (Samples) Example Application -
7      * VSAM Data Store
8      *
9      *
10     *
11     Licensed Materials - Property of IBM
12     *
13     "Restricted Materials of IBM"
14     *
15     5655-Y04
16     *
17     (C) Copyright IBM Corp. 2004, 2021"
18     *
19     DAT version
20     *
21     * STATUS = 7.2.0
22     * TRANSACTION NAME = n/a
23     *
-----+

```

3. Move the cursor to line 290 and press Ctrl-/ to comment out this line, alternatively you can also put an asterisk on column 8 which is indicated by the tab stop indicator in the editor.

The screenshot shows the COBOL editor with line 290 selected. The code on line 290 is:

```

MOVE WS-LOOP-COUNTER TO WS-RECORD-COUNT
this is a bug - start
MOVE 999.99 TO WS-COST IN WS-CAT-ITEM
this is a bug - end
MOVE WS-CAT-ITEM TO CA-CAT-ITEM(WS-LOOP-COUNTER)

```

4. Press Ctrl-S to save the file.

## Experimenting with other cool editing features

5. Browse code by using the Outline view: With the COBOL program DFH0XVDS.cbl open, use the **Outline view** to explore the structure of the program. This view provides a complete sortable and interactive "table of contents" for the program's code. You can click on the arrow icons at the left of each header to expand them, and quickly jump to any part of your code by clicking on it in the Outline view.

- In the **Outline view** (located in the lower left of the IDz window), expand the headers **PROGRAM: DFH0XVDS > PROCEDURE DIVISION > MAINLINE SECTION**.
  - (You might have to click the Outline tab to activate this view.)
- Click on the header "**CATALOG-INQUIRE**" to jump to this part of the code in the editor window.

The screenshot shows the z/OS Projects interface with the DFH0XVDS.cbl file open. The left pane displays the project structure and file outline. The right pane shows the COBOL source code. A green arrow highlights the 'CATALOG-INQUIRE.' command at line 233 in the code editor, which corresponds to the 'CATALOG-INQUIRE' command listed under the MAINLINE SECTION in the outline view.

```

-----+*A1-B-----2-----3-----4-----5-----+
      LENGTH (LENGTH OF ERROR-MSG)
226
227      END-EXEC.
228      EXIT.
229
230      * Procedure to link to Datastore program to ir
231      * on the catalog data
232
233* CATALOG-INQUIRE.
234
235
236      INITIALIZE CA-INQUIRY-RESPONSE-DATA
237      * Following 6 lines added by PM46914
238      PERFORM
239          WITH TEST AFTER
240          VARYING WS-LOOP-COUNTER FROM 1 BY 1
241          UNTIL WS-LOOP-COUNTER EQUAL 15
242          INITIALIZE CA-CAT-ITEM(WS-LOOP-COUNTER)
243      END-PERFORM
244
245      MOVE 'EXDSVSAM: CATALOG-INQUIRE' TO CA-RE
246
247      MOVE CA-LIST-START-REF TO WS-CURRENT-ITEM
248

```

6. Code completion: Use IDz's code completion feature to view and select from a list of commands, defined variable names, and code snippets when you start typing a command, variable, or paragraph name.

- At line 242 of the COBOL program DFH0XVDS.cbl, take note of the variable "CA-CAT-ITEM", and then place your cursor at the end of the line and press "Enter" to start a new line.
- Place your cursor at column 15 of the new line (if it is not already there), and beginning typing the variable name: "CA-C", then press Ctrl+Space. A selection list appears with matching commands, variable names, and code snippets, allowing you to quickly and accurately complete what you were typing by simply choosing the relevant item from the list.
  - The list dynamically narrows down if you continue typing the variable name (for example, ""CA-CAT") while it is displayed.
  - You can use the up and down arrow keys to move up and down the displayed list and view the declaration for each option.

The screenshot shows the z/OS Projects interface with the DFH0XVDS.cbl file open. The left pane displays the project structure and file outline. The right pane shows the COBOL source code. A dropdown menu is open at line 243, showing code completion options for 'CA-CAT-ITEM'. To the right, a detailed declaration for 'CA-CAT-ITEM' is shown in a yellow box.

```

INITIALIZE CA-CAT-ITEM(WS-LOOP-COUNTER)
CA-CAT
END-PERFOR
  CA-CAT-ITEM
  CA-CAT-ITEM IN CA-INQUIRE-REQUEST
MOVE 'EXDSVSAM: CATALOG-INQUIRE' TO CA-RE
MOVE CA-LIST-START-REF TO WS-CURRENT-ITEM
* Start browse

```

01 DFHCOMMAREA.  
03 CA-INQUIRE-REQUEST REDEFINES CA-REQUEST-SPECIFIC.  
05 CA-CAT-ITEM REDEFINES CA-INQUIRY-RESPONSE-DATA OCC  
07 CA-ITEM-REF PIC 9(4).  
07 CA-DESCRIPTION PIC X(40).  
07 CA-DEPARTMENT PIC 9(3).  
07 CA-COST PIC X(6).

7. Preview copybooks: Preview the contents of a copybook in the COBOL program.

- At line 129 in DFH0XVDS.cbl, hover your mouse cursor over the copybook name "DFH0XCP1" in the COPY statement to view the copybook preview.

The screenshot shows a COBOL source code editor with line numbers 127 to 142. At line 129, there is a COPY statement: COPY DFH0XCP1. A tooltip window is open over the text DFH0XCP1, displaying the following content:

```
*****
* CONTROL BLOCK NAME = DFH0XCP1
*
* DESCRIPTIVE NAME = CICS TS (Samples) Example Application -
*                   Main copybook for example application
*
```

The editor interface includes a search bar labeled "Find Text" and a "Case sensitive" checkbox. Below the search bar are several icons: magnifying glass, double arrows, a clipboard, and a pencil.

8. Hover for declaration: View a data declaration by hovering over the variable or paragraph name.

- At line 144, hover your cursor over the variable name "WS-HEADER". Note the hover pane that appears, showing you the working storage definition or DCL definition and the parent group of the variable name.
  - Note: To quickly jump to the variable's declaration, you can click the "WS-HEADER" variable to select it, and then press F3 (or right-click and select "Open declaration").

The screenshot shows a COBOL source code editor with line numbers 142 to 153. At line 144, there is an INITIALIZE statement: INITIALIZE WS-HEADER. A tooltip window is open over the text WS-HEADER, displaying the following content:

```
01 WS-HEADER.
03 WS-EYECATCHER PIC X(16) VALUE 'DFH0XVDS-----WS'.
03 WS-TRANSID PIC X(4).
03 WS-TERMID PIC X(4).
03 WS-TASKNUM PIC 9(7).
03 WS-CALEN PIC S9(4) COMP.
```

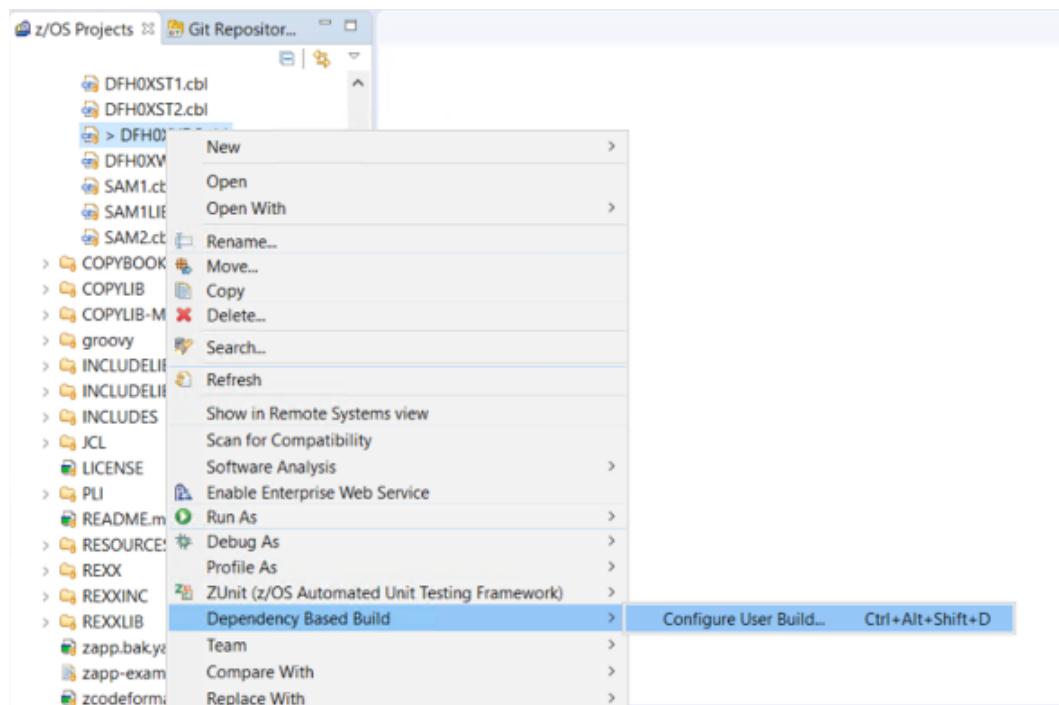
The editor interface includes a search bar and a toolbar with various icons.

9. Close the file without saving it.

## Part 5: Run a user build to build the program in your own workspace

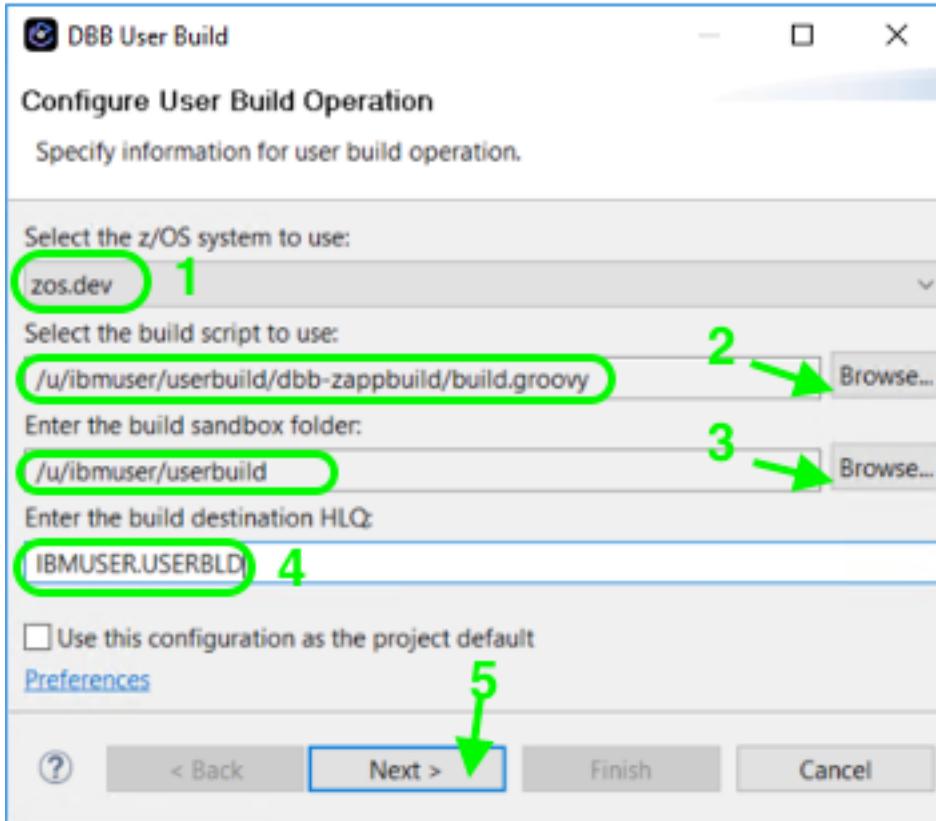
In this part of the lab exercise, you will run a user build that will upload the changed code with all its dependencies to the z/OS environment and create a load module in your personal PDS dataset, with detail build result transferred back to your IDE for inspection.

- 1. With the DFH0XVDS program saved, right click the program name "DFH0XVDS.cbl" in the z/OS Projects view, and select "Dependency Based Build" > "Configure IBM User Build".



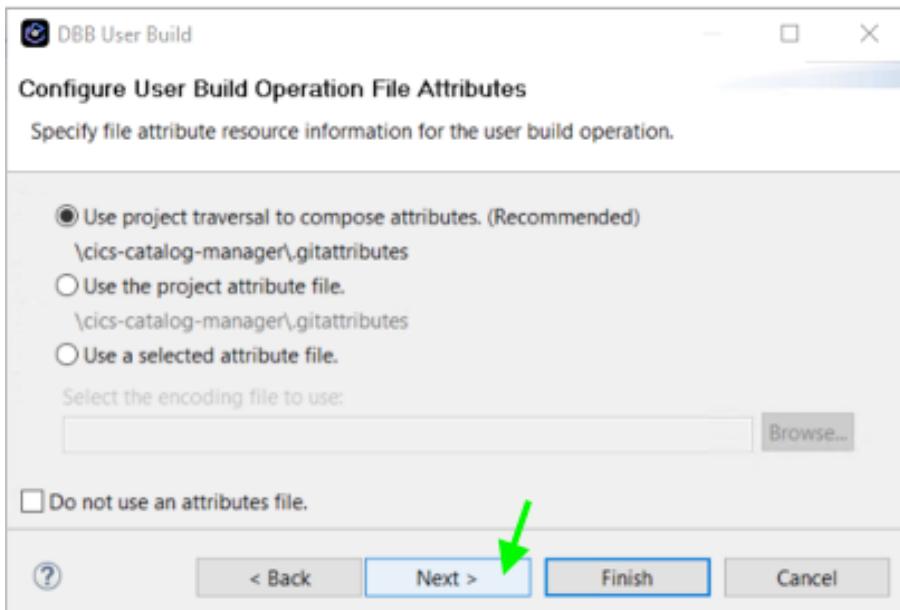
- 2. Be sure that the build values in the DBB User Build pop-up window match the values as below. (You can use the **Browse** button to select the values.)

1. z/OS system: `zos.dev`
2. Build script: `/u/ibmuser/userbuild/dbb-zappbuild/build.groovy` (use **Browse** the **remote system**)
3. Build sandbox: `/u/ibmuser/userbuild` (Use **Browse**, expand **MyHome**, click **userbuild** and **OK**)
4. Build destination HLQ: `IBMUSER.USERBLD` (must type)
5. Click **Next**

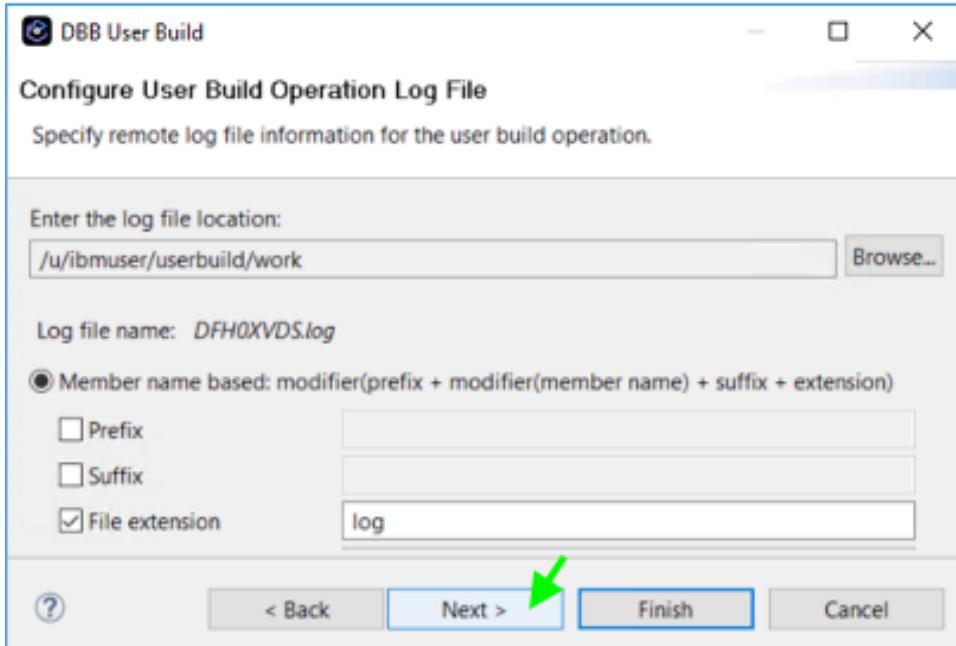


3. On Configure User Build Operation File Attributes click Next.

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

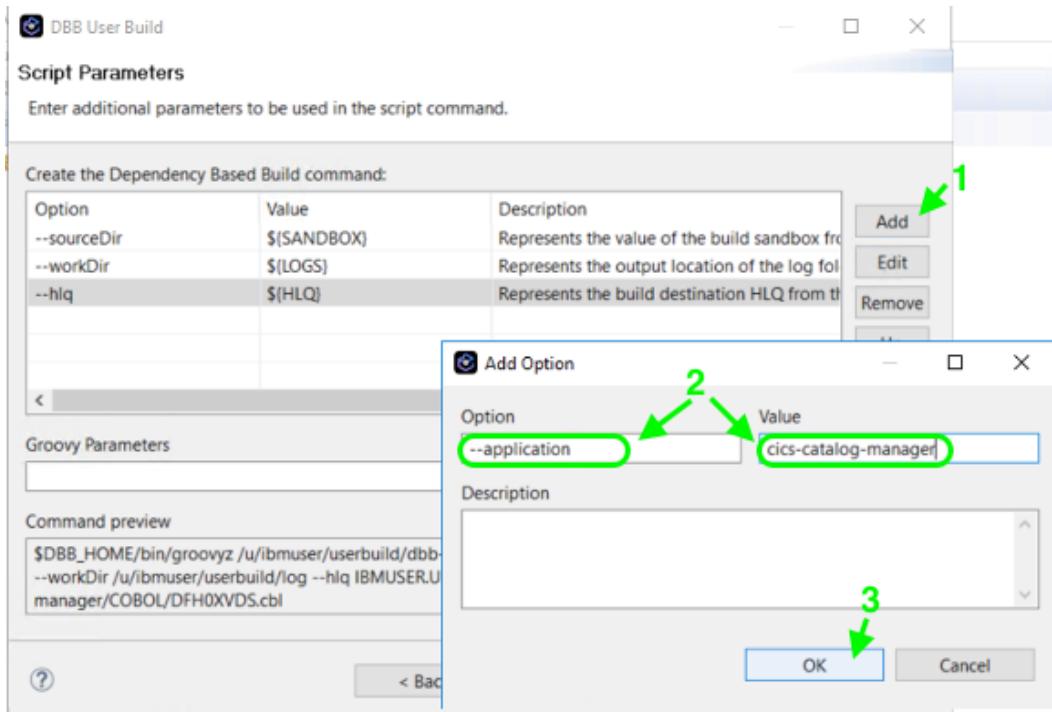


4. On Configure User Build Operation Log File, click Next. You will use the default values.



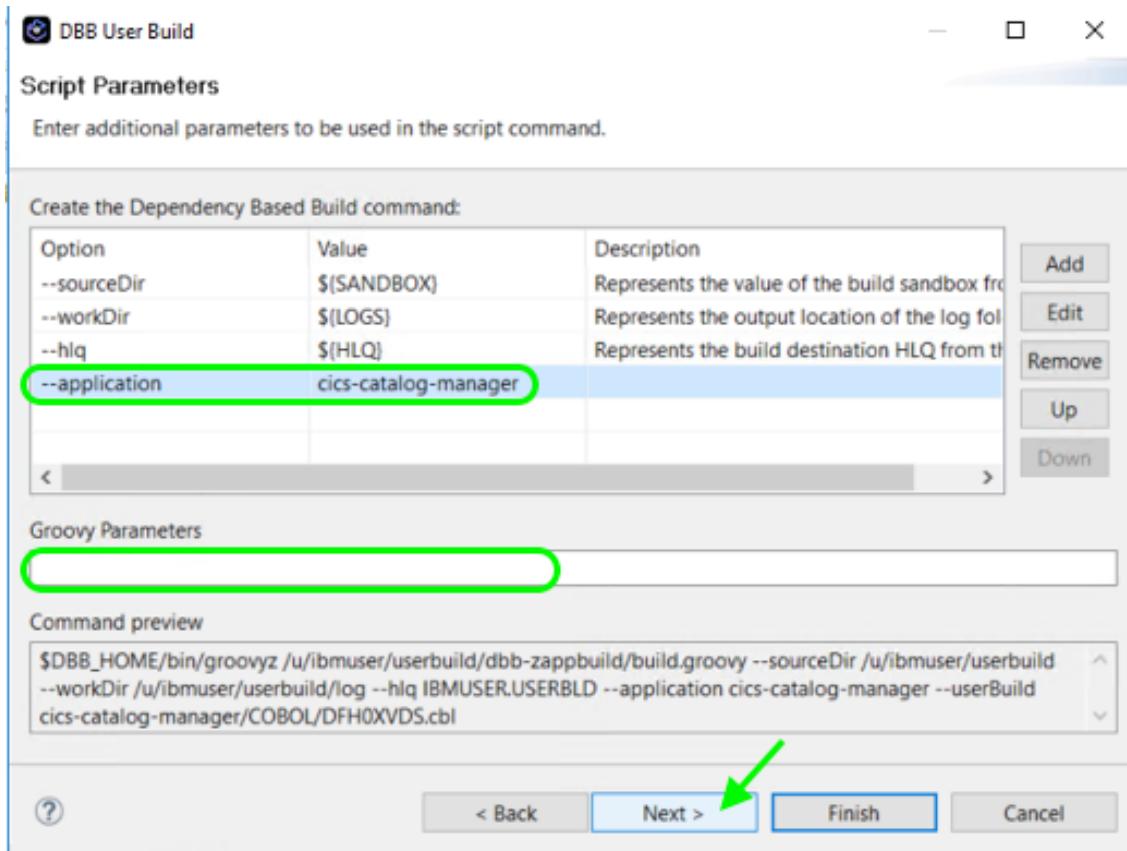
5. In *Script Parameters*, specify the application's folder:

1. Click "Add".
2. In the "Add Option" pop-up window, put "**--application**" for the **Option** field, and "**cics-catalog-manager**" for the **Value** field.
3. Click "Ok".



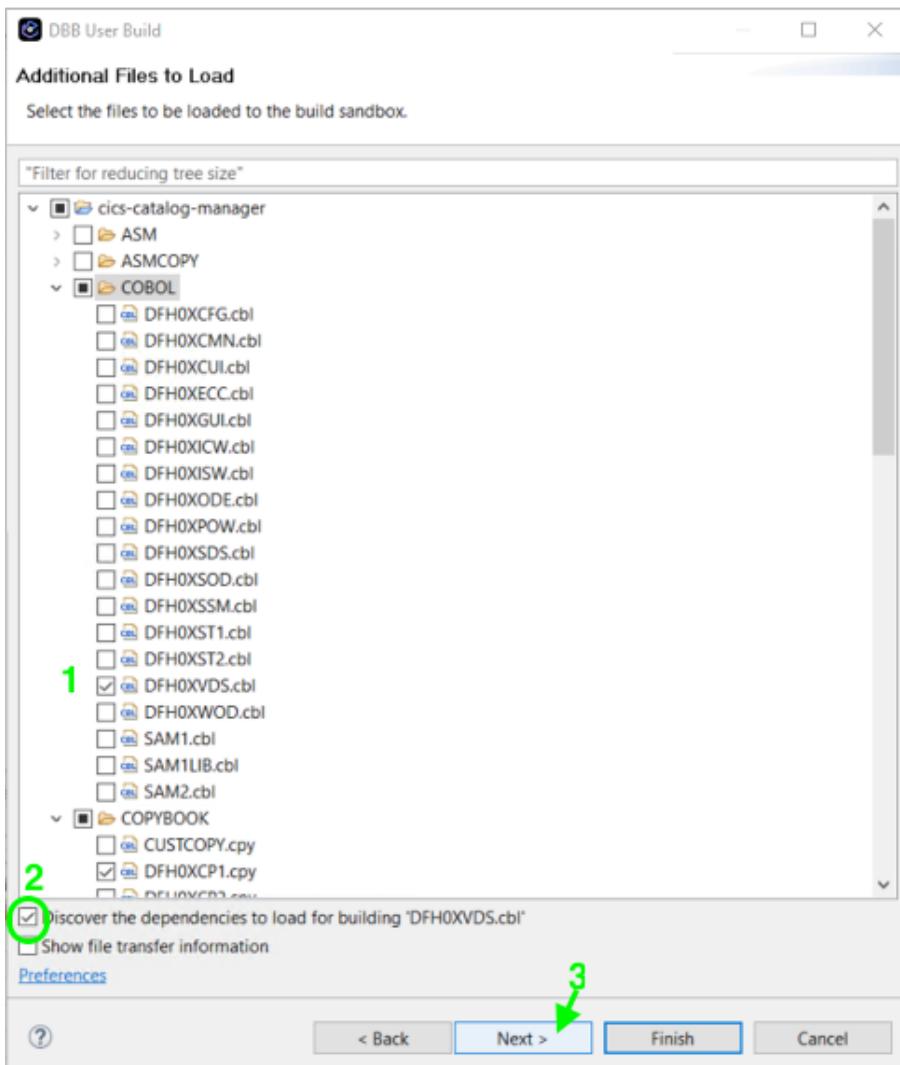
6. Back in the *Script Parameters* window, the newly-added `--application` option should now be listed.

- Be sure that the field "**Groovy Parameters**" is empty, and then click **Next**. Notice the preview of the commands that will be executed on the z/OS by the DBB toolkit.

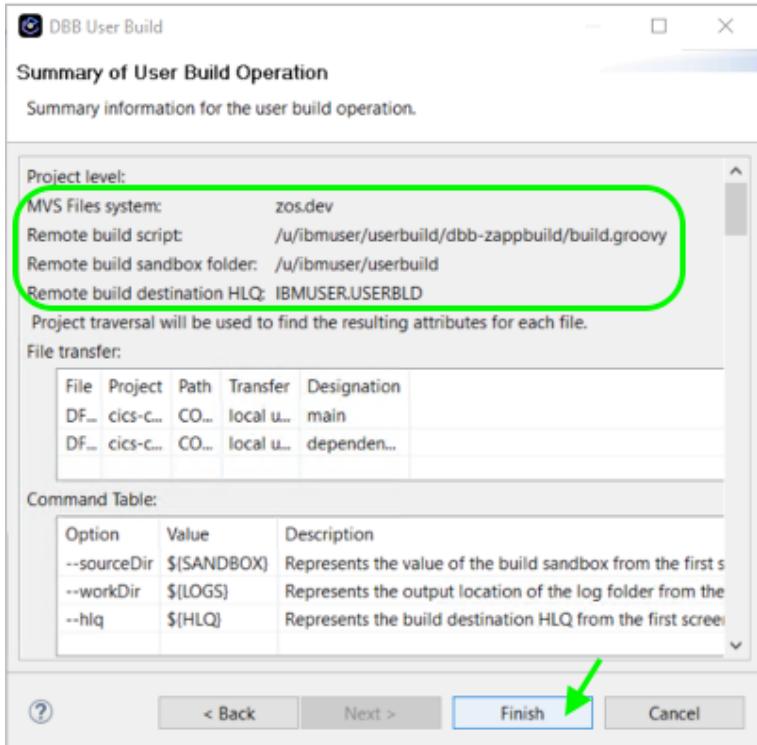


7. In Additional Files to Load, the selected files will be moved from your local workspace to a z/OS USS directory and the execution of the Groovy scripts will interact with the DBB framework that will invoke the compiler, linkage editor, etc.

1. Expand **cics-catalog-manager** and the **COBOL** and **COPYBOOK** directories.
2. Since you will also need the related copybook for a clean compile, be sure that "**Discover the dependencies to load for building DFH0XVDS.cbl**" is selected (automatically checked).
  - i. **Note:** If this is the first user build to run for this application on z/OS, you might need to also select the "application-conf" folder, to ensure those configurations will be uploaded correctly.
3. Click **Next**.



8. Verify the summary of the User Build Operation and click **Finish**.

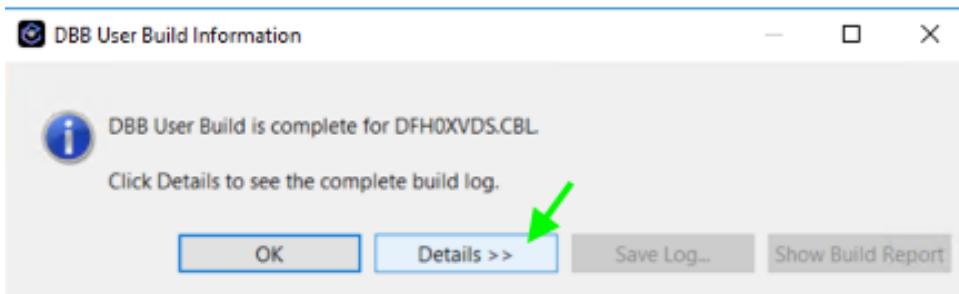


\_\_\_ 9. If the dialogs for "Duplicate Name Collision" pops up, select **Overwrite** and click **OK**

- Also select "**Overwrite All**" for the copybooks ("Copy Resources" dialog)
- The DBB User build will be running and the messages will be shown at the Console view. Click on the **Console** tab on lower right corner. The execution will start, and this will take a while (2 Minutes) .

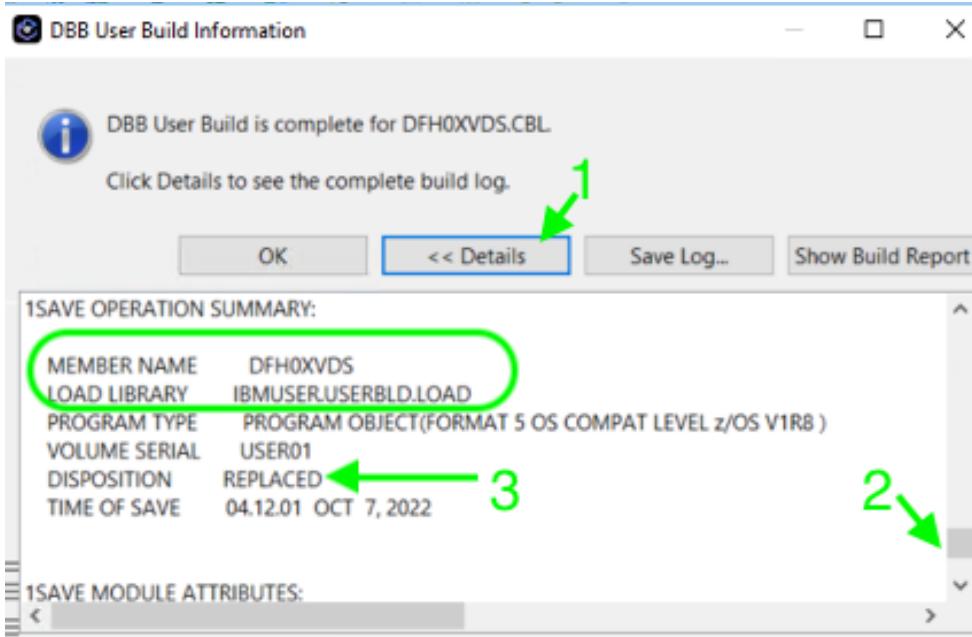
\_\_\_ 10. When complete you will have the "DBB User Build Information" message.

- Click **Details>>**

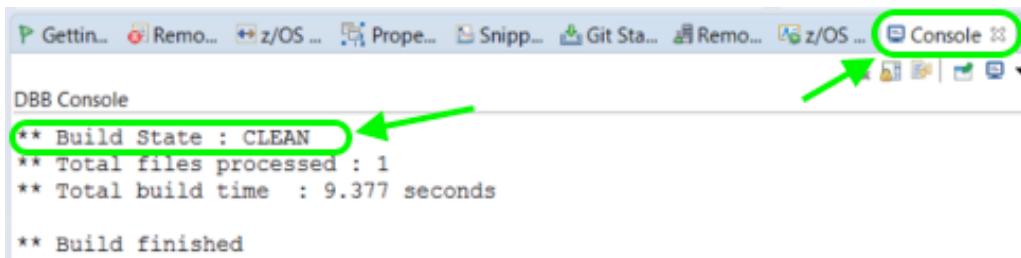


\_\_\_ 11. After clicking **Details >>**, scroll down to near the bottom of the details and verify under "SAVE OPERATION SUMMARY" that a new load module named **DFH0XVDS** was replaced on the dataset **IBMUSER.USERBLD.LOAD**.

- Click **OK** to close this dialog.



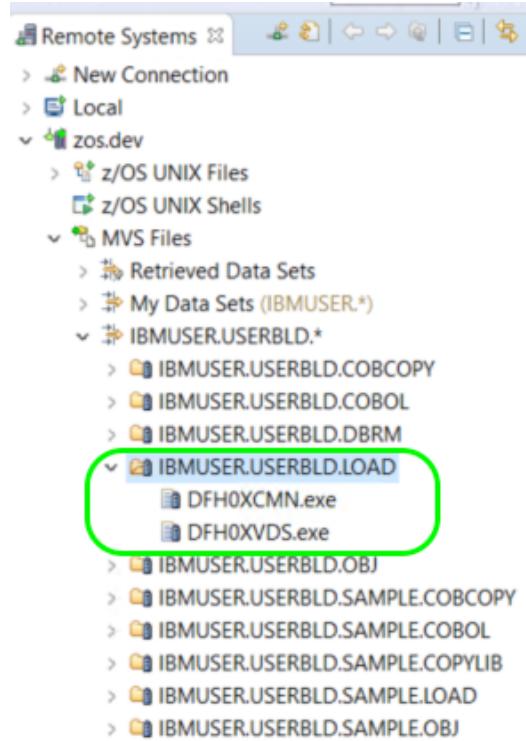
12. Verify at the **Console** tab that the Build State should show a *CLEAN* build.



13. Scroll up if you want to understand the process, take note of the HLQ of the user build location which will be used in the next step.

14. Use the Remote System Explorer (RSE) view to connect to z/OS using the pre-configured RSE profile and create a dataset filter using IBMUSER.USERBLD:

- To create a new filter in the RSE view, right click "MVS files" underneath the connection "zos.dev", and select "New" > "Filter...".
  - Enter IBMUSER.USERBLD into the Filter string field, and click "Next>", and then click "Finish". The new filter appears under zos.dev's "MVS Files".
  - Expand the new filter to see the load module has been compiled into IBMUSER.USERBLD.LOAD.



## **Part 6: Early integration test of the application before deployment**

In this part of the lab exercise, you will perform an early integration test to verify that your code change did fix the issue and it's now working and playing well with the rest of the CICS application, before deploying it into a CICS environment.

**Note:**

IBM Z Virtual Test Platform (ZVTP) provides capabilities for developers and testers to perform batch and transactional testing without the need for data or supporting subsystem (CICS, Db2, IMS, or MQ) products.

ZVTP works by intercepting calls made by an application to subsystems or other programs and recording details about those calls. After the data is recorded, those same application programs can be rerun in batch without the need for the original test environment by returning values from the recorded data. This allows users to automate the testing process of online transactions and batch programs, whether they are testing a single program, hundreds of programs, or thousands of transactions. Additionally, the application programs can be modified with required changes and then rerun to help ensure they perform without adverse effects. Subsystem responses to program calls can also be altered during the recording by user commands or during replay by using exit points. This allows users to test execution paths that programs do not normally take without having to make any program changes.

The product is composed of three parts: Dynamic Test Runner (for recording and replay), the optional ZVTP Viewer, and the optional ZVTP Server Extension. None of these parts requires any changes to user application code. There is no need for recompilation, relinking, or rebinding. The process does not require access to a debugger. All of the call interceptions are performed by using the user's original load modules, whether the application runs in batch, CICS, or IMS.

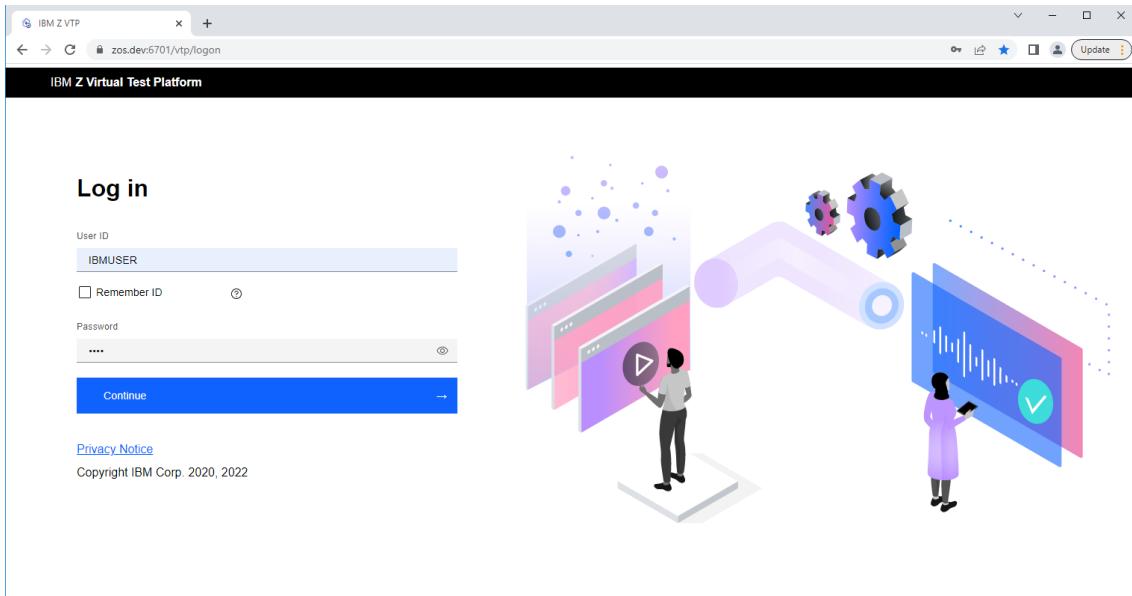
ZVTP can be used independently or in cooperation with other products. For more information, see product document. <https://www.ibm.com/docs/en/zvtp/2.0?topic=overview>

ZVTP is intended to be used by development and testing. It is not intended for a high-throughput or production environment. This product intercepts and records data contained in the arguments of various calls and this data is not "sanitized". Do not record production or sensitive data because this information can easily be viewed in the sequential file created during the recording process.

## Explore the IBM Z Virtual Test Platform web interface

1. Open a new browser tab and click **IBM VTP** from the bookmark or navigate to

<https://zos.dev:6701/vtp/logon>



2. Log in use **IBMUSER** and password **SYS1** and click **Continue** to go to the **Test cases** overview page.

Name	Type	Transactions	Programs	Recording time	Replay time	Result history	Recent result
Catalog Manager BMS 3270	CICS	EGUI	DFH0XSSM +4	9/7/2022, 8:33:14 PM	-	-	-
Catalog Manager call from API	CICS	CSMI	DFH0XCMN +1	9/19/2022, 8:31:29 PM	-	-	-

3. Click on the **Catalog Manager Call from API** test case which is a previous Virtual Test Platform recording of the execution path from API before the defect is fixed. It recorded all the CICS activity and Input/Output from all the CICS calls that's initiated when running a catalog inquire call from z/OS Connect API. By clicking on the different **Programs** name you can check the **Statement** of the EXEC CICS calls made from the program.

4. Click on the **Timeline** which show all the CICS calls involved, you can see the API invokes **DFH0XCMN** which subsequence LINK into **DFH0XVDS** where the actual VSAM **READNEXT** requests were captured before returning the control back to **DFH0XCMN**.

### Replay the test case using the new load module built by user build

5. Now we can replay the same test case using the user build load module we built in previous step. Click **Run test case** button at the top.
6. Provide the user build load module PDS name **IBMUSER.USERBLD.LOAD** in the **User library** section.

7. Click **Run** button to submit the replay of the test case, IBM Z VTP will replay the test case use updated user build load module and it will compare the test result.
- If you have been idle for too long since the log in, your user session will expire, and the run test case will fail. Just login the Virtual Test Platform again and retry the operation.

[Test cases](#)

### Catalog Manager call from API

Replay time: 10/5/2022, 6:12:24 PM

[Run test case](#) [Save changes](#) [1 result history](#)

[About test case](#) [Test results \(RC=4\)](#)

The test case was run successfully

Test case named Catalog Manager call from API was run successfully. The table and history will be refreshed.

8. When finished you will see a new entry of the **test result** is recorded with **RC=4** which indicates some discrepancies were found, click on the **test results** to look at the details.
9. Any mismatch between **recorded** vs **replayed** is highlighted with yellow icon, click on the **DFH0XVDS** program you will notice up until the (EXEC CICS) **RETURN** everything was a perfect match. Then click on the **RETURN** it then highlights the COMMAREA differences. While the recorded price was **999.99**, now during the replay it's now **002.90** with the new user build load module. It shows your code change has now fixed the defect.

Recorded	Replayed
010999...990133000020Ba1, Pens.Blue, 24pk,.....01 44444444444444FFFF4FFFFFFFFFFFC8994D9A4C9A54FF9944444444444444444444FF BBBBBBSBBBBB010995B9901330000020133B7552B2345B2472BBBBBBSBBBBBBSBBBBB01	010999...990133000020Ba1, Pens.Blue, 24pk,.....01 44444444444444FFFF4FFFFFFFFFFFC8994D9A4C9A54FF9944444444444444444444FF BBBBBBSBBBBB010002B9001330000020133B7552B2345B2472BBBBBBSBBBBBBSBBBBB01

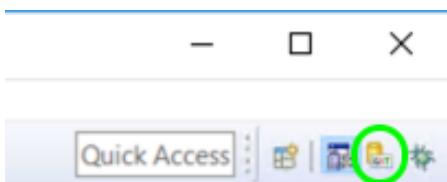
10. What happened in the background is the replay process kicks off a batch job to run your load module, without the need of CICS environment (or any other middleware your application may require, e.g. DB2, VSAM, IMS or MQ). While executing your program, VTP takes over all the external system calls and returns pre-recorded data to your program, and your program won't even know that it's not running in a real CICS region but in a virtual environment (hence the name Virtual Test Platform). VTP then capture any changes in its behavior – whether it's intended or not and report it back.
11. This is what we call an **early integration test** so that you can shift left the testing before the deployment, removing dependencies to the external resources so that you can test earlier and more often. But obviously, this is not a replacement for a full integration test which is still required at a later stage.

## Part 7: Commit the code and push to the remote Git repo to close the merge request

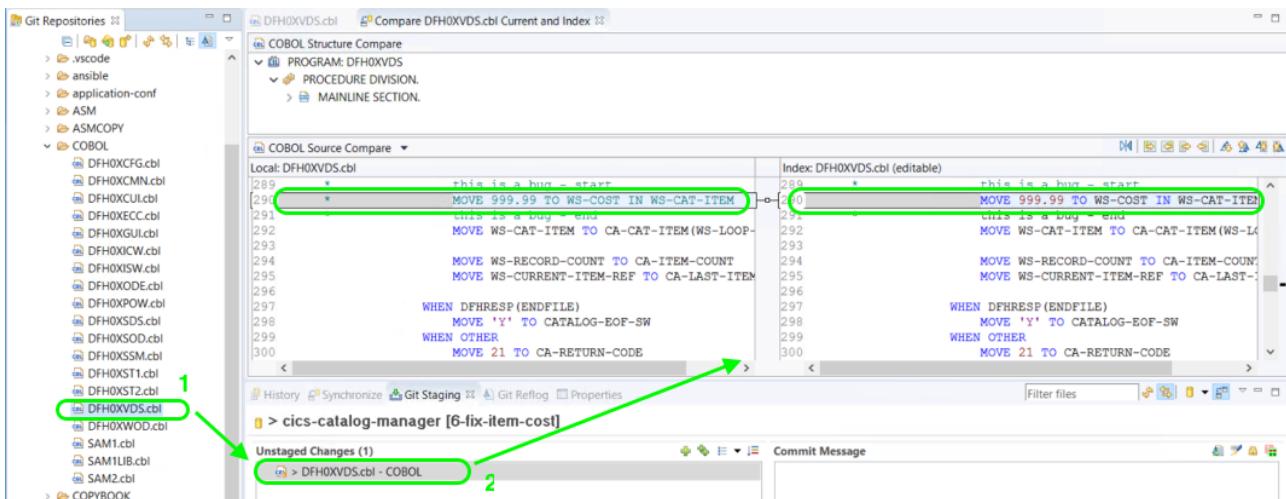
In this step you are confident the code is ready so you will commit it to your local working branch of the Git repository and push it to the remote Git repo to submit the work you've done. This step will automatically trigger the pipeline for the full build, and when the build is successful you can then safely merge the changes into the master branch.

### Commit the code change and push to the remote repository on GitLab

- \_\_\_ 1. Return to IDz Window using the icon () from the taskbar at the bottom of the screen.
- \_\_\_ 2. Switch to the **Git Perspective** selecting the icon () at the top right of the IDz window.



- \_\_\_ 3. Expand the nodes and click on the program **DFH0XVDS.cbl** that you have modified. Notice that the changed program should be listed in the Git Staging view.
  - Double click on the **DFH0XVDS.cbl** that is listed under *Unstaged Changes* and notice the comparison between the program that you updated ("Local") versus the one that is in the central Git repository ("Index").
    - You will need to commit the changes to the Git repository to begin the process of integrating them into the team's shared branch for development code.



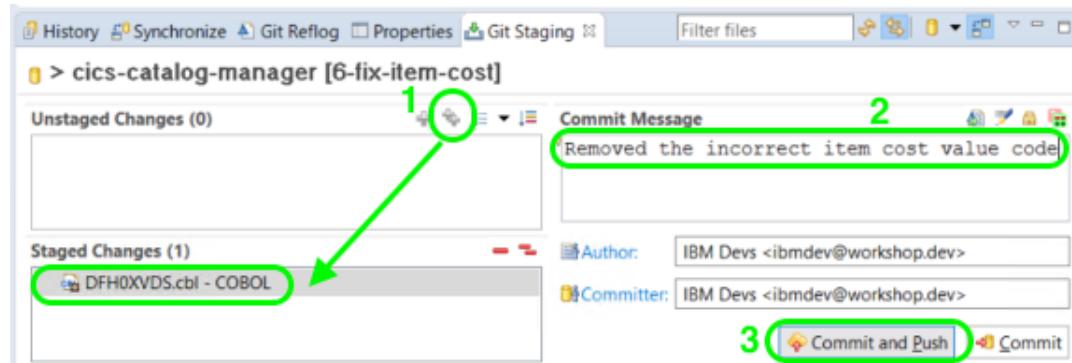
- **Note:** If the DFH0XVDS.cbl program is not listed in the Git Staging pane, then you might need to click the "switch repository" icon (dropdown arrow) and select the application (cics-catalog-manager) to switch to it.



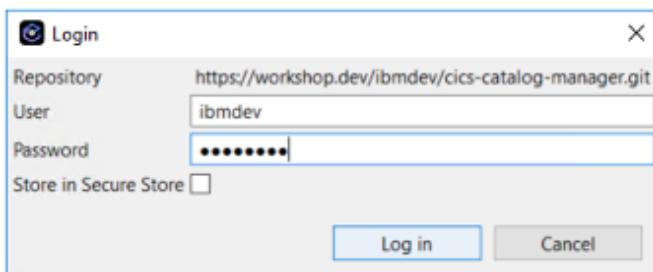
\_\_\_ 4. Use **Ctrl + Shift + F4** to close the editors.

\_\_\_ 5. In the **Git Staging** view:

1. Click on the icon
2. Write a commit message, such as: "Removed the incorrect item cost value code".
3. Click **Commit and Push ...**

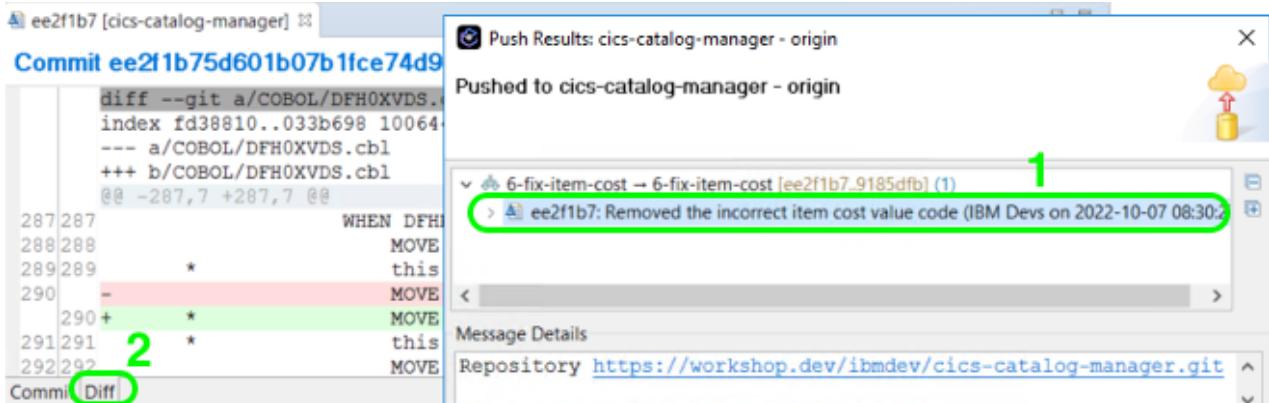


\_\_\_ 6. If it prompts for login credentials, use your GitLab credentials (user: `ibmdev` and password: `Passw0rd`) and then click **Log in**.



\_\_\_ 7. Optional: To see the changes you committed to Git when the "Push Results" window pops up:

1. Double click on Git hash string in the "Push Results" pop up. (Note: Your branch name and Git hash will likely be different from the screenshot below, and this is okay.)
2. Select the **Diff** tab.



\_\_\_8. Verify that the commit was successful and click **Close**.

- Use **Ctrl + Shift + F4** to close any opened editor.

## Review the changes on GitLab and pipeline for the full application build

\_\_\_9. Go to GitLab browser window and navigate to the project homepage by clicking the project name on the top left

c cics-catalog-manager

\_\_\_10. Switch the branch to our working branch ***n-incorrect-item-cost***

The screenshot shows the GitLab project homepage for 'cics-catalog-manager'. At the top, it shows 'IBM Devs > cics-catalog-manager'. The project details include '21 Commits', '3 Branches', '0 Tags', '5.6 MB Project Storage', 'Project ID: 10', and a 'CI/CD Pipeline Lab - CICS Catalog Manager' status bar. Below this, there's a 'Switch branch/tag' dropdown menu with 'master' selected. The main area shows a list of commits. The latest commit is highlighted with a green checkmark and the ID '0c5966b7'. The commit message is partially visible as 'Initial commit for CICS & DevOps Workshop'. Other commits listed include 'Initial commit for CICS & DevOps Workshop' and another commit from '2 weeks ago'.

\_\_\_11. Click on **commit message** of the latest commit we just made.

IBM Devs > cics-catalog-manager > Repository

The screenshot shows a repository view for 'cics-catalog-manager'. At the top, there's a navigation bar with '5-incorrect-item-c...', 'cics-catalog-manager / +', 'History', 'Find file', 'Web IDE', 'Clone', and a download icon. Below the navigation is a commit details card for a commit that removed an incorrect item cost value code. It includes a checkmark icon, the commit hash '2b7945a0', and a copy icon. A table below lists project files with their last commit details:

Name	Last commit	Last update
.vscode	updated project structure	17 minutes ago
ASM	Initial commit for CICS & DevOps Workshop	2 weeks ago
ASMCOPY	Initial commit for CICS & DevOps Workshop	2 weeks ago
COBOL	removed the incorrect item cost value code	9 minutes ago
COPYBOOK	Initial commit for CICS & DevOps Workshop	2 weeks ago
COPYLIB-MVS	Initial commit for CICS & DevOps Workshop	2 weeks ago
COPYLIB	Initial commit for CICS & DevOps Workshop	2 weeks ago

12. This screen shows the commit details and the merge request we submitted for fixing this defect, as well as the pipeline that's triggered by this commit with all the individual jobs' status.

#### removed the incorrect item cost value code

This screenshot shows a pull request or merge request details page. It includes sections for the parent commit ('parent 6fe0b464'), a merge request ('1 merge request 16 Draft: Resolve "Incorrect item cost"'), and a pipeline status ('Pipeline #101 passed with stages').

Changes 1 Pipelines 1

This screenshot shows a code diff view for the file 'COBOL/DFH0XVDS.cbl'. It highlights changes with red for deletions and green for additions. The diff shows several lines of COBOL code, with specific lines 290 and 291 highlighted in pink and green respectively, indicating they were modified.

13. Click on the Pipeline id #101 (your id will be different) to inspect the execution of the jobs in this pipeline.

IBM Devs > cics-catalog-manager > Pipelines > #101

passed Pipeline #101 triggered 20 minutes ago by IBM Devs Delete

removed the incorrect item cost value code

⌚ 4 jobs for [5-incorrect-item-cost](#) in 2 minutes and 22 seconds (queued for 3 seconds)

⟳ latest

⌚ 2b7945a0 ⌱

湓 1 related merge request: [!6 Draft: Resolve "Incorrect item cost"](#)

Pipeline Needs Jobs 4 Tests 0

Build Report Deploy Test

DBB Build Build Report CICS Deploy Integration Test

14. Click on the **DBB Build** you can review the detail log of the **IBM Dependency Build** on the z/OS. In there you can also find a link to the DBB Web App for a consolidated view of the build result.

```

16 Fetching changes with git depth set to 20...
17 Reinitialized existing Git repository in /u/ibmuser/builds/egvgArRC/0/ibmdev/cics-catalog-manager/.git/
18 Checking out 2b7945a0 as 5-incorrect-item-cost...
19 Removing builds/
20 Skipping Git submodules setup
21 Executing "step_script" stage of the job script
22 Set up environment for zos
23 /usr/lpp/cbclib/xlclang/bin/xlclang found, using CGO
24 Done!
25 $ pwd # collapsed multi-line command
26 /u/ibmuser/builds/egvgArRC/0/ibmdev/cics-catalog-manager
27 $ $DBB_HOME/bin/groovyz \ # collapsed multi-line command
28 ** Build start at 20221005.065205.052
29 ** Repository client created for https://workshop.dev:9444/dbb
30 ** Build output located at builds/egvgArRC/0/ibmdev/cics-catalog-manager/builds/101/build.20221005.065205.052
31 ** Build result created for BuildGroup:cics-catalog-manager-5-incorrect-item-cost BuildLabel:build.20221005.065205.052 at https://workshop.dev:9444/dbb/rest/buildResult/543
32 ** --fullBuild option selected. Building all programs for application cics-catalog-manager
33 ** Writing build list file to builds/egvgArRC/0/ibmdev/cics-catalog-manager/builds/101/build.20221005.065205.052/buildList.txt
34 ** Scanning source code.
35 ** Invoking build scripts according to build order: BMS.groovy,Cobol.groovy,Assembler.groovy,P
LI.groovy,LinkEdit.groovy
36 ** Building files mapped to Cobol.groovy script
37 *** Building file cics-catalog-manager/COBOL/DFH0XPOW.cbl
38 *** Building file cics-catalog-manager/COBOL/DFH0XODE.cbl
39 *** Building file cics-catalog-manager/COBOL/DFH0XECC.cbl
40 *** Building file cics-catalog-manager/COBOL/DFH0XVDS.cbl
41 *** Building file cics-catalog-manager/COBOL/DFH0XCMN.cbl
42 *** Building file cics-catalog-manager/COBOL/DFH0XGUI.cbl
43 *** Building file cics-catalog-manager/COBOL/DFH0XCUI.cbl
44 *** Building file cics-catalog-manager/COBOL/SAM2.cbl
45 *** Building file cics-catalog-manager/COBOL/DFH0XSSM.cbl
46 *** Building file cics-catalog-manager/COBOL/DFH0XST2.cbl
47 *** Building file cics-catalog-manager/COBOL/DFH0XWOD.cbl
48 *** Building file cics-catalog-manager/COBOL/DFH0XWOD.cbl
49 *** Building file cics-catalog-manager/COBOL/SAM1.cbl

```

15. You can also log on to the DBB web app to exam all the build results from all developers. Right-Click on this link in the DBB log and choose to Open link in a new tab.

<https://workshop.dev:9444/dbb>

Alternatively open a new browser tab and click the IBM DBB link from the bookmark bar.

16. When prompted use ibmdev and Passw0rd to log in and now you can inspect the build collections and detail build result.

The screenshot shows a web-based application window titled "IBM DBB". The URL in the address bar is "workshop.dev:9444/dbb/". The main content area is titled "IBM Dependency Based Build". It displays a table with two rows of data:

Collection	Collection Owner	Latest Build	Build Owner
cics-catalog-manager-5-incorrect-item-cost	ibmdev	10/5/2022	ibmdev
cics-catalog-manager-5-incorrect-item-cost-outputs	ibmdev	10/5/2022	ibmdev

At the bottom of the table, there are pagination controls: "Items per page: 15", "1–2 of 2 items", "1 of 1 page", and navigation arrows.

17. Expand the collection name with your branch name and click on one of the **Build Results**, then click **View Build Report**, you can review the details of the program with dependencies and their compile return code etc.

#### 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: 18

	File	Commands	RC	Data Sets	Outputs	Deploy Type	Logs
1	<a href="#">cics-catalog-manager/COBOL/DFH0XPOW.cbl</a> <a href="#">Show Dependencies</a>	IGYCRCTL	0	IBMUSER.EXMPCAT.COBOL(DFH0XPOW)			DFH0XPOW.cobol.log
		IEWBLINK	0		IBMUSER EXMPCAT LOAD(DFH0XPOW)	LOAD	
2	<a href="#">cics-catalog-manager/COBOL/DFH0XODE.cbl</a> <a href="#">Show Dependencies</a>	IGYCRCTL	0	IBMUSER.EXMPCAT.COBOL(DFH0XODE)			DFH0XODE.cobol.log
		IEWBLINK	0		IBMUSER EXMPCAT LOAD(DFH0XODE)	LOAD	
3	<a href="#">cics-catalog-manager/COBOL/DFH0XECC.cbl</a> <a href="#">Show Dependencies</a>	IGYCRCTL	0	IBMUSER.EXMPCAT.COBOL(DFH0XECC)			DFH0XECC.cobol.log
		IEWBLINK	0		IBMUSER EXMPCAT LOAD(DFH0XECC)	LOAD	
4	<a href="#">cics-catalog-manager/COBOL/DFH0XVDS.cbl</a> <a href="#">Show Dependencies</a>	IGYCRCTL	0	IBMUSER.EXMPCAT.COBOL(DFH0XVDS)			DFH0XVDS.cobol.log
		IEWBLINK	0		IBMUSER EXMPCAT LOAD(DFH0XVDS)	LOAD	
5	<a href="#">cics-catalog-manager/COBOL/DFH0XCMN.cbl</a> <a href="#">Show Dependencies</a>	IGYCRCTL	0	IBMUSER.EXMPCAT.COBOL(DFH0XCMN)			DFH0XCMN.cobol.log
		IEWBLINK	0		IBMUSER EXMPCAT LOAD(DFH0XCMN)	LOAD	
6	<a href="#">cics-catalog-manager/COBOL/DFH0XGUI.cbl</a> <a href="#">Show Dependencies</a>	IGYCRCTL	0	IBMUSER.EXMPCAT.COBOL(DFH0XGUI)			DFH0XGUI.cobol.log
		IEWBLINK	0		IBMUSER EXMPCAT LOAD(DFH0XGUI)	LOAD	

### Note on DBB Collection

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 is to first check if the collection with that name already exists before creating it.

18. Back to the **GitLab** page, click **Back** in the browser to the pipeline detail page and click on the **Build Report** job. On the right, you can see this job has some artifacts attached to it, click the Browse button you can download the compiler output of this build. This step is optional but used to keep a copy of the compiler output on GitLab for demonstration purposes.

The screenshot shows the GitLab Job Build Report page for a job named "Job Build Report" triggered 42 minutes ago by "IBM Devs". The build status is "passed". The build log is displayed in a dark-themed terminal window, showing the execution of a gitlab-runner script. The log includes steps like "Preparing the 'shell' executor", "Using Shell executor...", "Preparing environment", "Running on d-4782-k...", "Getting source from Git repository", "Fetching changes with git depth set to 20...", "Reinitialized existing Git repository", "Checking out 2b7945a0 as 5-incorrect-item-cost...", "Removing build\_93/", "Skipping Git submodules setup", "Executing 'step\_script' stage of the job script", and "Successfully downloaded USS file '/u/ibmuser/builds/reports-101.zip' to local file 'reports-101.zip'". The build duration was 33 seconds, and it finished 42 minutes ago. Artifacts are listed as available for download.

19. Let's navigate back to examine the third job where we do the **CICS Deploy**, this is an example that uses the zowe CLI CICS plugin to dynamically issue **NEWCOPY** command to the CICS programs so that the changes can be refreshed in CICS.

```

2fc1585 version=15.2.1
  19 Downloading artifacts from coordinator... ok      id=348 responseStatus=200 OK token=MaQz5em
    p
  ↴ 21 Executing "step_script" stage of the job script   00:08
    22 $ echo "Deploying to CICS"
    23 Deploying to CICS
    24 $ zowe cics refresh program DFH0XCMN --region-name CICSTS56
    25 The program 'DFH0XCMN' was refreshed successfully.
    26 $ zowe cics refresh program DFH0XVDS --region-name CICSTS56
    27 The program 'DFH0XVDS' was refreshed successfully.
  ↴ 29 Cleaning up project directory and file based variables   00:00
    31 Job succeeded

```

In a real-world scenario, DBB can package the load modules and other resources into a TAR file, then upload to a centralized location like Artifactory or GitLab Package Registry as the CI (continuous integration) pipeline, then there can be another closed loop manages the CD part (continuous delivery) that drives an automated delivery lifecycle for the versions and release management.

- 20. Go back to the previous pipeline screen and look at the **Integration test** job, this is another simple example that invokes **curl** command to test the program by calling an API from z/OS Connect. And you can see the item cost issue has now been fixed.

```

30 {
31   "totalItems": 15,
32   "items": [
33     {
34       "summary": {
35         "stock": "There are 131 items in stock. [KEY=QmFsbC5QZW5zLkJsYWNrLjI0cGsuLi4uLi4uLi4uL
36           i4uLi4uLi4uLg=]",
37         "orders": "0 on order at unit price of A$2.9. Total order value: $0"
38       },
39       "information": {
40         "itemReference": 10,
41         "description": "Ball Pens Black 24pk",
42         "cost": "2.9",
43         "department": 10,
44         "stock": 131,
45         "onOrder": 0
46       }
47     ],
48   }

```

**Note:**

We used zAppBuild in our lab. It is a highly customizable build solution for building z/OS applications using Apache Groovy build scripts and IBM Dependency Based Build (DBB) APIs.

The zAppBuild repository is intended to be cloned to a single location on Unix Systems Services (USS) and used to build all of your z/OS applications. This is done by simply copying the supplied application-conf folder (located in the samples folder) to the application source repository you want to build and then verify/update the contained default configuration property values to ensure they meet the build requirements of your application.

The zAppBuild sample provides the following language build scripts by default:

Assembler / BMS / Cobol / LinkEdit / PLI / DBDgen / PSBgen / MFS and ZunitConfig

All language scripts both compile and optionally link-edit programs. The language build scripts are intended to be useful out of the box but depending on the complexity of your applications' build requirements, may require modifications to meet your development team's needs. By following the examples used in the existing language build scripts of keeping all application specific references out of the build scripts and instead using configuration properties with strong default values, the zAppBuild sample can continue to be a generic build solution for all of your specific applications.

The build scope of zAppBuild is an application that is loosely defined as one or more Git repositories containing all the z/OS source files required to build the application. There are no specific rules as to the structure of the repositories except that one repository must contain the high-level application-conf folder provided by zAppBuild which contains all of the configuration properties for building the application programs.

zAppBuild supports a number of build scenarios:

Single Program - Build a single program in the application.

List of Programs - Build a list of programs provided by a text file.

Full Build - Build all programs (or buildable files) of an application.

Impact Build - Build only programs impacted by source files that have changed since the last successful build.

Impact Build with baseline reference - Build only programs impacted by source files that have changed by diffing to a previous configuration reference.

Topic Branch Build - Detects when building a topic branch for the first time and will automatically clone the dependency data collections from the main build branch in order to avoid having to rescan the entire application.

Merge Build - Build only changed programs which will be merged back into the mainBuildBranch by using a triple-dot git diff.

Scan Source - Skip the actual building and only scan source files to store dependency data in collection (migration scenario).

Scan Source + Outputs - Skip the actual building and only scan source files and existing load modules to dependency data in source and output collection (migration scenario with static linkage scenarios).

For more detail refer to <https://github.com/IBM/dbb-zappbuild>

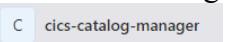
## Complete the merge request to submit work to the master branch

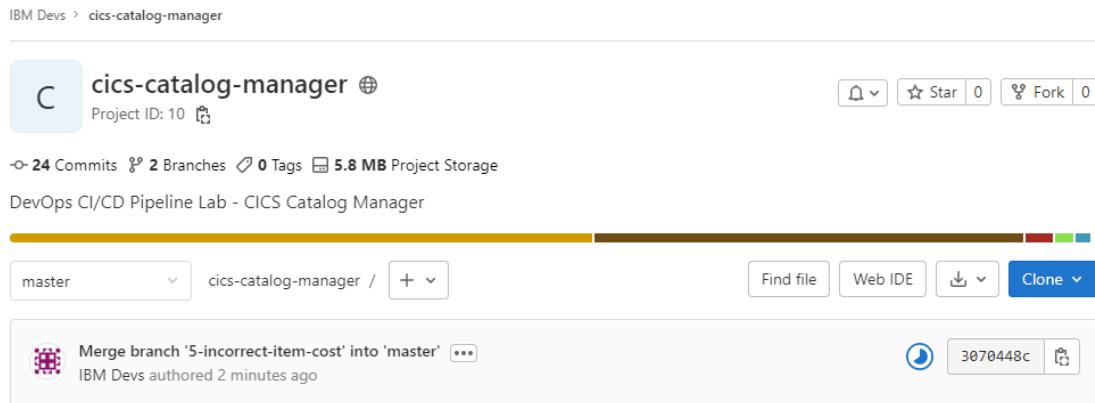
Since everything looks good and our code is ready to progress to the next stage which is to be merge with the master branch.

**21.** Click on the **Merge Request** in the GitLab project on the left, and you can find the current draft merge request.

22. Click on the **draft merge request** it shows current status is **Open**, the **pipeline** for the branch full build has passed successfully, **approval** is optional in this case, and you can click on **Mark as ready** button to prepare the merge.

23. Click **Merge** button to make it official.

- \_\_\_ 24. Congratulations, you've just merged your work into the master branch. Now your working branch is deleted, also the original GitLab **Issue** is closed automatically too.
- \_\_\_ 25. Now it triggers another build on the master branch so that your code will be built again with any other changes other people might have committed to master while you were working on the fix. Click the project name button to go back to the project home page.  And you can see the pipeline is running. For this lab, you will expect the same result.



## Part 8: Summary

**Congratulations**, you have completed all the tasks in this lab.

In this lab you performed the following steps:

- Exploring the GitLab project and review the issue and create a new branch to work on the fix.
- Create a clone of the new branch into your IDE
- Debugging the application at statement level
- Make the code change
- Run a user build to build the program in your own workspace
- Early integration test of the application before deployment
- Commit the code and push to the remote Git repo to close the merge request