IBM – CICS and DevOps Workshop

Lab – Git-based DevOps Pipeline for Z Application

Wazi Developer for VS Code

Lab Version V1.1

October 2022

Please send any comments on this lab exercise to:
George Ge: ypge@au1.ibm.com
Lauren Li: lauren.k.li@ibm.com

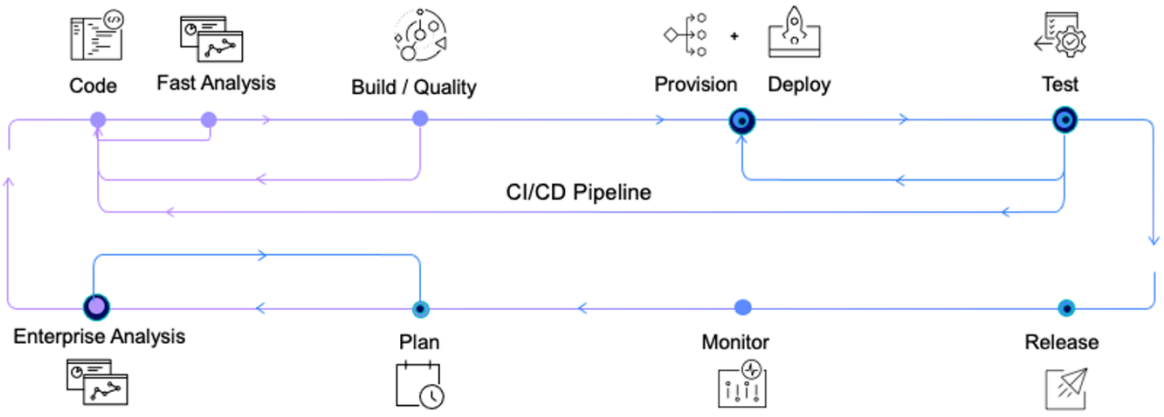
Overview

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 IBM 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.

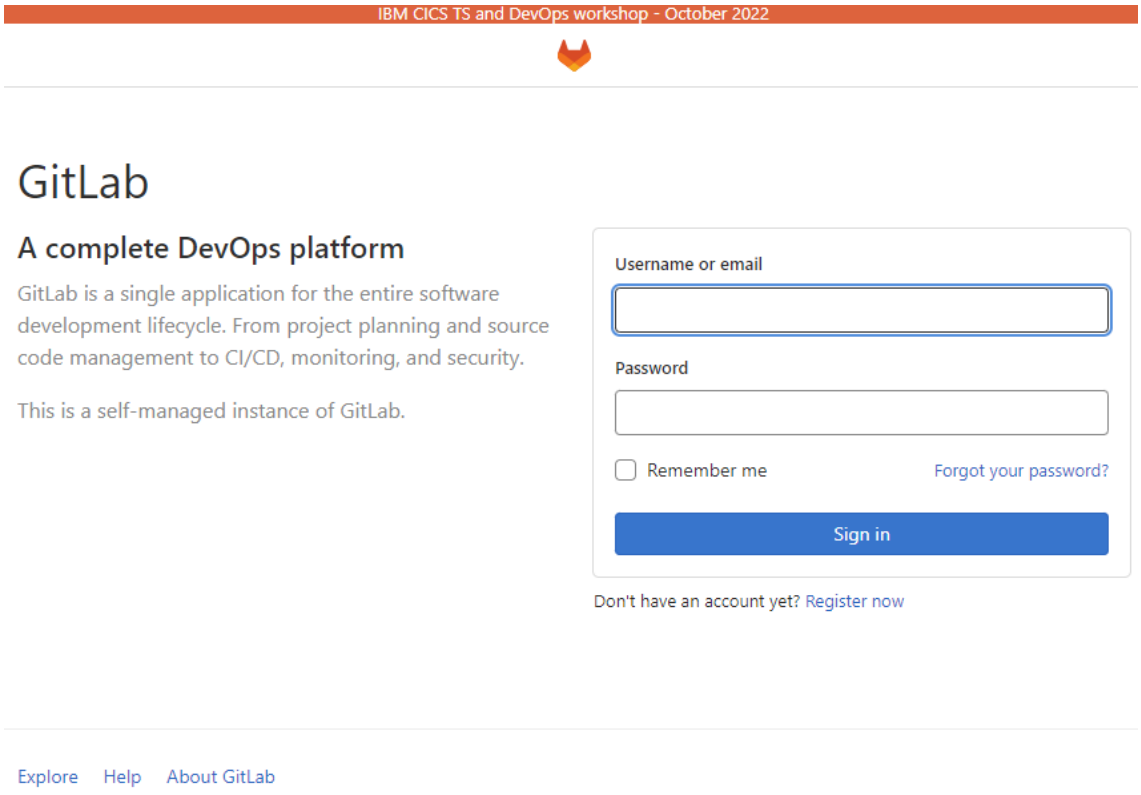
Part 1: Exploring the GitLab project and initiating a merge request for the issue

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.

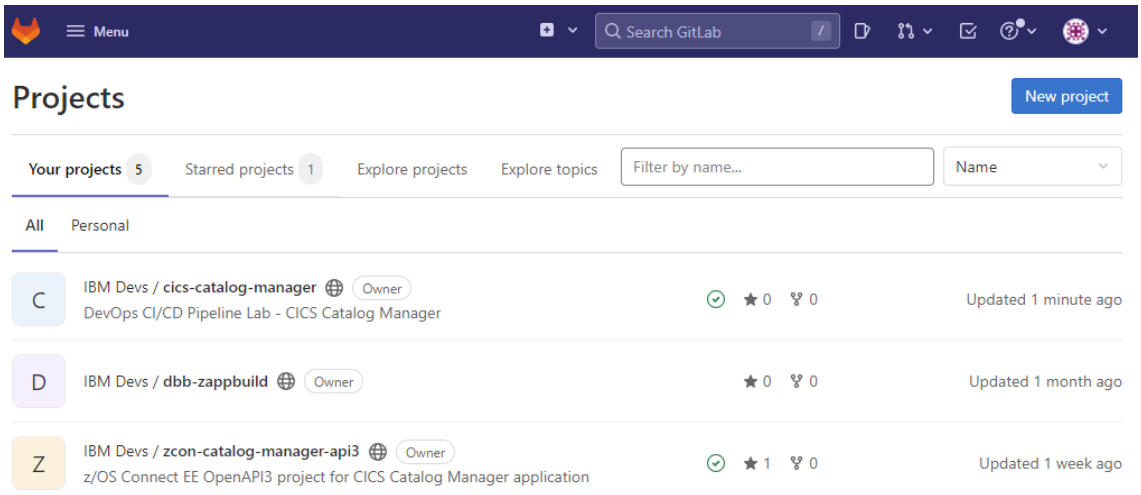
Log on to the GitLab dashboard

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

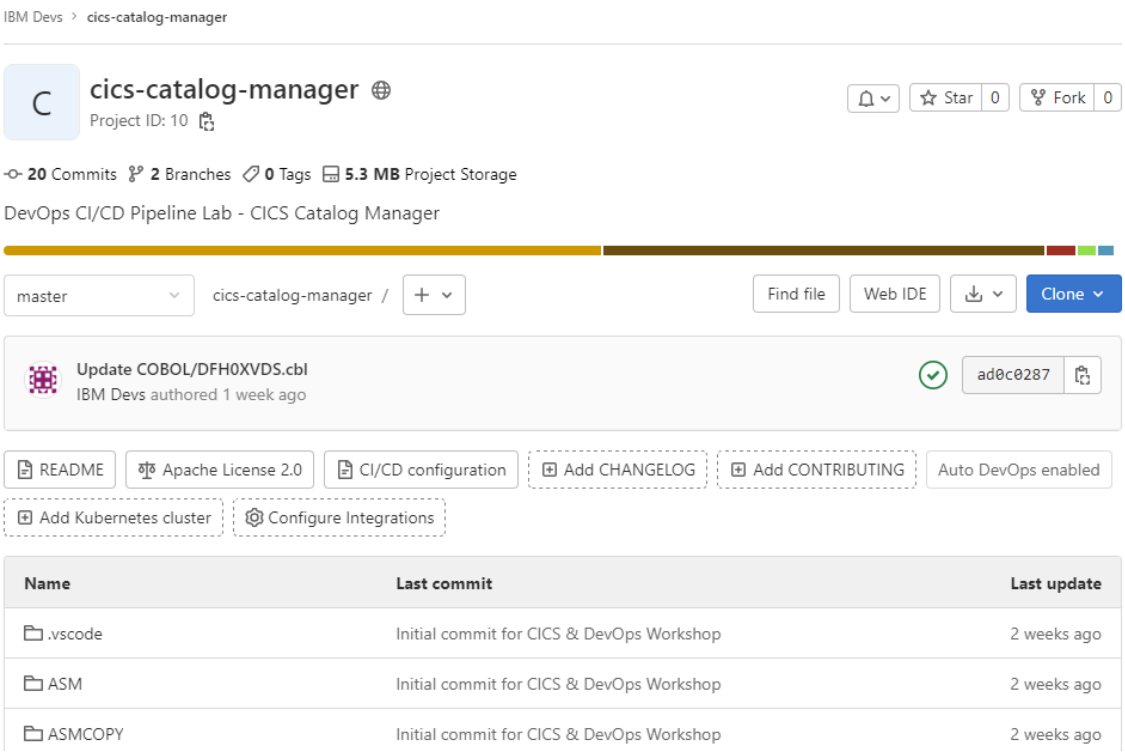
___**2.** Click on the GitLab bookmark which opens the log in page.



___**3.** Use username `ibmdev` and password `PassW0rd` to login, you will see the dashboard for all the projects.

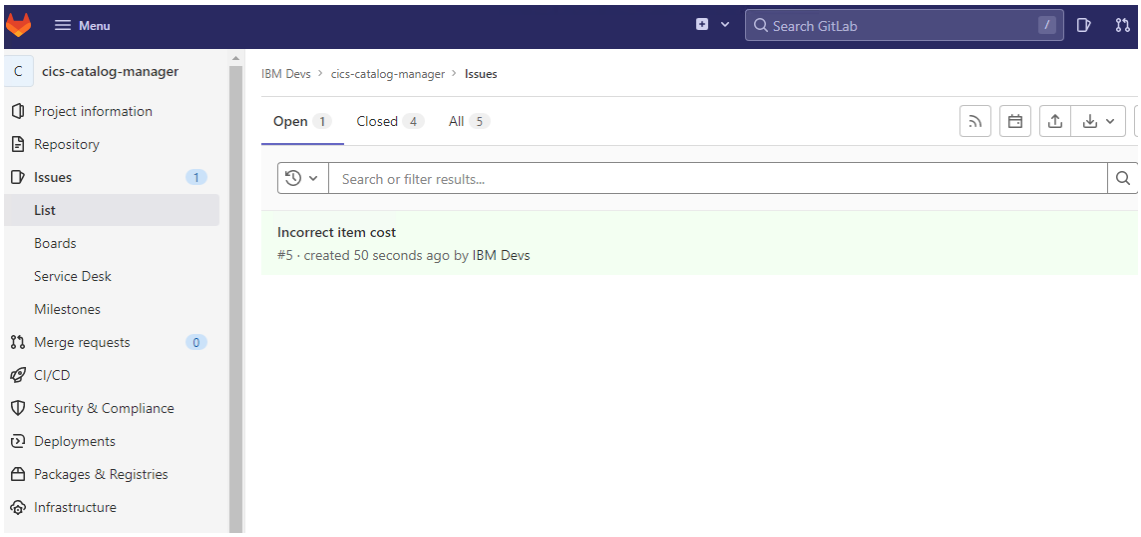


___**4.** Click on the **cics-catalog-manager** project and you will be taken to the project homepage.

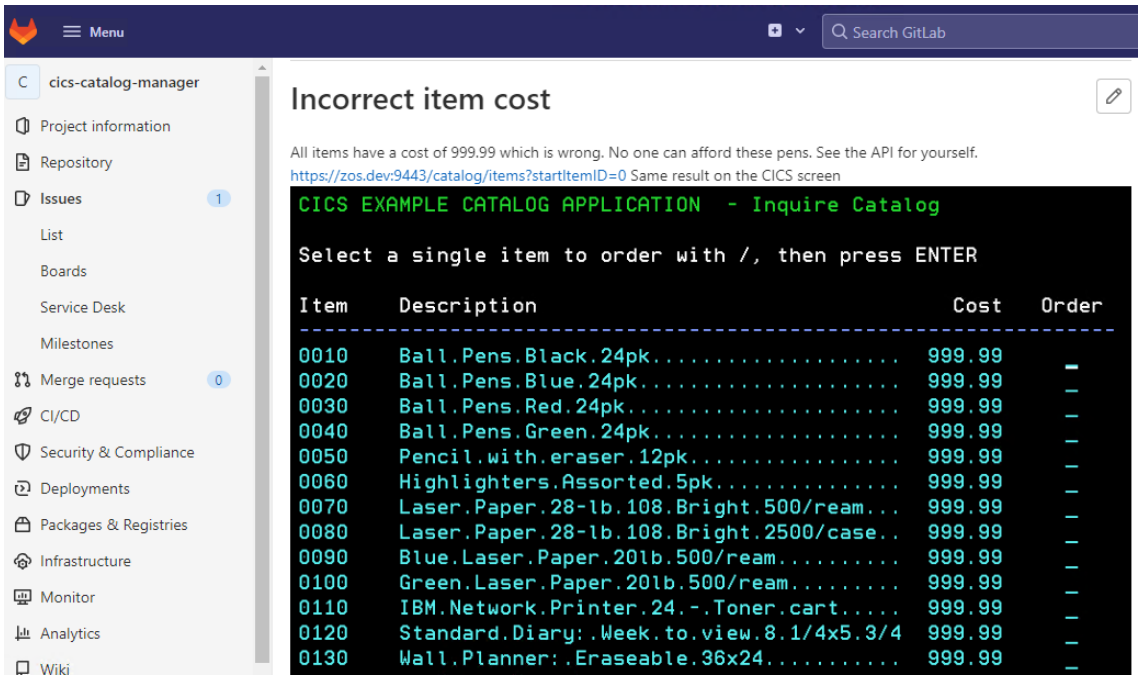


View the project issues and confirm the defect

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



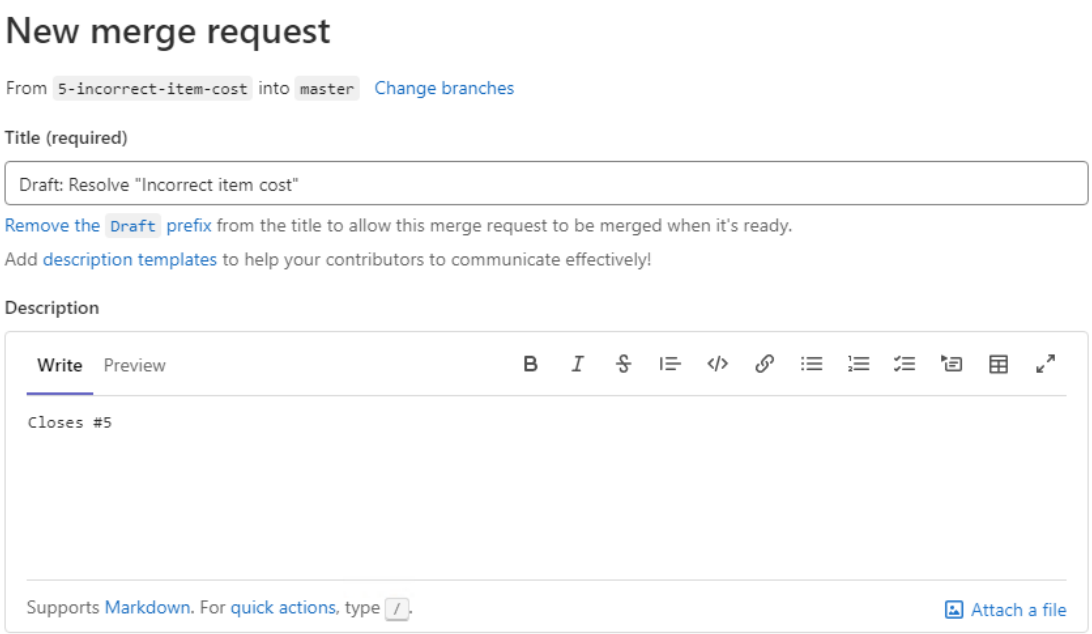
___**6.** On the right, click the "Incorrect item cost" issue that was created earlier.



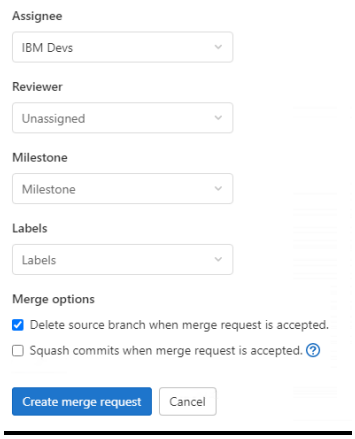
___**7.** Use the link included in the description to try the Catalog Manager APIs which shows all the item costs were wrongly marked as **999.99**

```
"totalItems":15,"items":[{"summary":{"stock":"There are 131 items in stock. [KEY=QmFsbC5QZW5zLkJsYWNrLjI0cGsuLi4uLi4uLi4uLi4uLi4uLg==]",
otal order value: $0"},"information":{"itemReference":10,"description":"Ball Pens Black 24pk","cost":"999.99","department":10,"stock":131,"o
tems in stock. [KEY=QmFsbC5QZW5zLkJsdWUuMjRway4uLi4uLi4uLi4uLi4uLi4uLg==]","orders":"50 on order at unit price of A$999.99. Total order
"itemReference":20,"description":"Ball Pens Blue 24pk","cost":"999.99","department":10,"stock":6,"onOrder":50}},{"summary":{"stock":"There a
KEY=QmFsbC5QZW5zLlJlZC4yNHBrLi4uLi4uLi4uLi4uLi4uLg==]","orders":"0 on order at unit price of A$999.99. Total order value: $0"},"info
"itemReference":30,"description":"Ball Pens Red 24pk","cost":"999.99","department":10,"stock":106,"onOrder":0}},{"summary":{"stock":"There a
KEY=QmFsbC5QZW5zLkdyZWVuLjI0cGsuLi4uLi4uLi4uLi4uLi4uLg==]","orders":"0 on order at unit price of A$999.99. Total order value: $0"},"info
"itemReference":40,"description":"Ball Pens Green 24pk","cost":"999.99","department":10,"stock":80,"onOrder":0}},{"summary":{"stock":"There
KEY=UGVuY2lsLndpdGguZXJhc2VyLjEycGsuLi4uLi4uLi4uLi4uLi4uLg==]","orders":"0 on order at unit price of A$999.99. Total order value: $0"},"info
"itemReference":50,"description":"Pencil with eraser 12pk","cost":"999.99","department":10,"stock":83,"onOrder":0}},{"summary":{"stock":"The
KEY=SGlnaGxpZ2h0ZXJzLkFzc29ydGVkLjVway4uLi4uLi4uLi4uLi4uLg==]","orders":"40 on order at unit price of A$999.99. Total order value: $39999.6"
"itemReference":60,"description":"Highlighters Assorted 5pk","cost":"999.99","department":10,"stock":13,"onOrder":40}},{"summary":{"stock":"
KEY=TGFzZXIuUGFwZXIuMjgtbGIuMTA4LkJyaWdodC41MDAvcmVhbS4uLg==]","orders":"20 on order at unit price of A$999.99. Total order value: $19999.8"
"itemReference":70,"description":"Laser Paper 28-lb 108 Bright 500/ream","cost":"999.99","department":10,"stock":102,"onOrder":20}},{"summar
KEY=TGFzZXIuUGFwZXIuMjgtbGIuMTA4LkJyaWdodC4yNTAwL2Nhc2UuLg==]","orders":"0 on order at unit price of A$999.99. Total order value: $0"},"info
"itemReference":80,"description":"Laser Paper 28-lb 108 Bright 2500/case","cost":"999.99","department":10,"stock":25,"onOrder":0}},{"summary
KEY=Qmx1ZS5MYXNlci5QYXBlci4yMGxiLjUwMC9yZWFtLi4uLi4uLi4uLi4uLg==]","orders":"0 on order at unit price of A$999.99. Total order value: $0"},"info
"itemReference":90,"description":"Blue Laser Paper 20lb 500/ream","cost":"999.99","department":10,"stock":22,"onOrder":0}},{"summary":{"stoc
KEY=R3JlZW4uTGFzZXIuUGFwZXIuMjBsYi41MDAvcmVhbS4uLi4uLi4uLg==]","orders":"20 on order at unit price of A$999.99. Total order value: $19999.8"
"itemReference":100,"description":"Green Laser Paper 20lb 500/ream","cost":"999.99","department":10,"stock":3,"onOrder":20}},{"summary":{"st
KEY=SUJNLk5ldHdvcmsuUHJpbnRlci4yNC4tLlRvbmVyLmNhcnQuLi4uLg==]","orders":"0 on order at unit price of A$999.99. Total order value: $0"},"info
"itemReference":110,"description":"IBM Network Printer 24 - Toner cart","cost":"999.99","department":10,"stock":12,"onOrder":0}},{"summary":
KEY=U3RhbmRhcmQuRGlhcnk6LldlZWsudG8udmlldy44LjEvNHg1LjMvNA==]","orders":"0 on order at unit price of A$999.99. Total order value: $0"},"info
"itemReference":120,"description":"Standard Diary: Week to view 8 1/4x5 3/4","cost":"999.99","department":10,"stock":7,"onOrder":0}},{"summa
KEY=V2FsbC5QbGFubmVyOi5FcmFzZWFibGUuMzZ4MjQuLi4uLi4uLi4uLi4uLg==]","orders":"0 on order at unit price of A$999.99. Total order value: $0"},"info
"itemReference":130,"description":"Wall Planner: Eraseable 36x24","cost":"999.99","department":10,"stock":3,"onOrder":0}},{"summary":{"stock
KEY=NzAuU2hlZXQuSGFyZC5CYWNrLndpcmUuYm91bmQubm90ZXBhZC4uLg==]","orders":"0 on order at unit price of A$999.99. Total order value: $0"},"info
heet Hard Back wire bound notepad","cost":"999.99","department":10,"stock":84,"onOrder":0}},{"summary":{"stock":"There are 36 items in stock
KEY=U3RpY2t5Lk5vdGVzLjN4My5Bc3NvcnRlZC5Db2xvcnMuNXBrLi4uLg==]","orders":"45 on order at unit price of A$999.99. Total order value: $44999.55
"itemReference":150,"description":"Sticky Notes 3x3 Assorted Colors 5pk","cost":"999.99","department":10,"stock":36,"onOrder":45}}]}
```

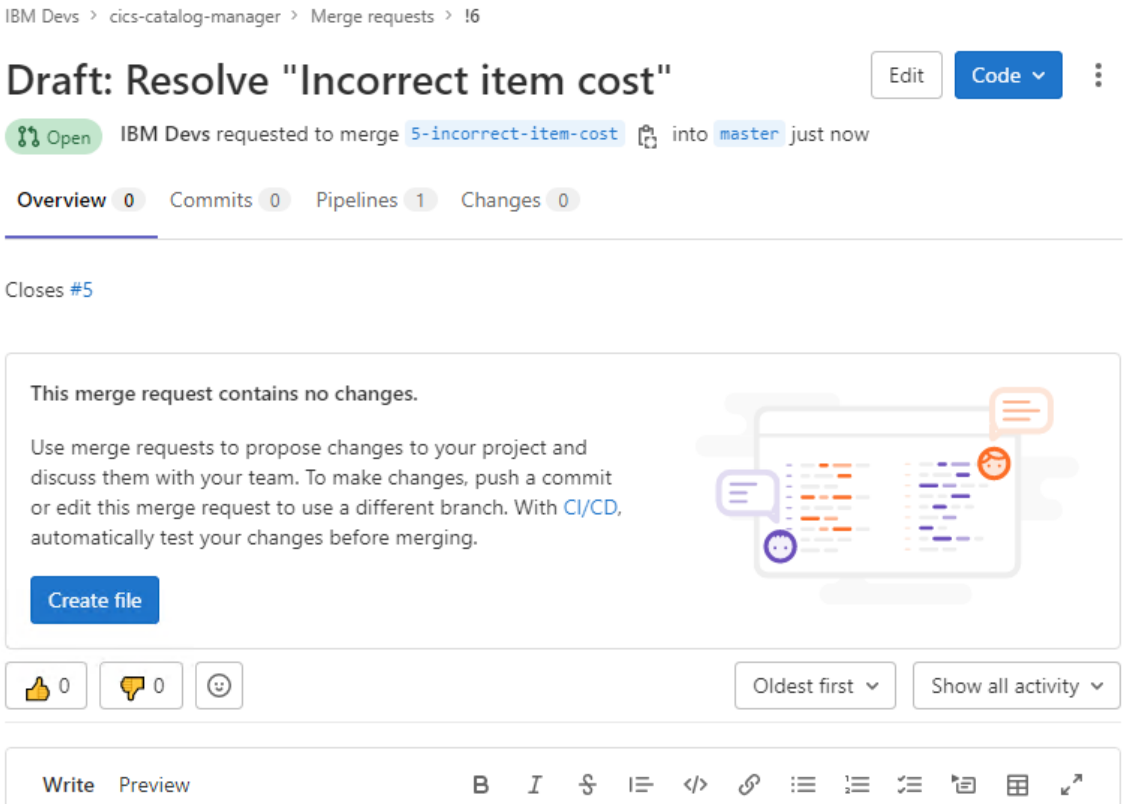

___**8.** Go back to **GitLab** browser tab and scroll down the issue and click the **Create Merge Request** button to start a new branch called ***n*-incorrect-item-cost** (branch name is generated automatically and it can be a different from the screenshot) and a draft merge request that will be used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



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



___**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.



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.

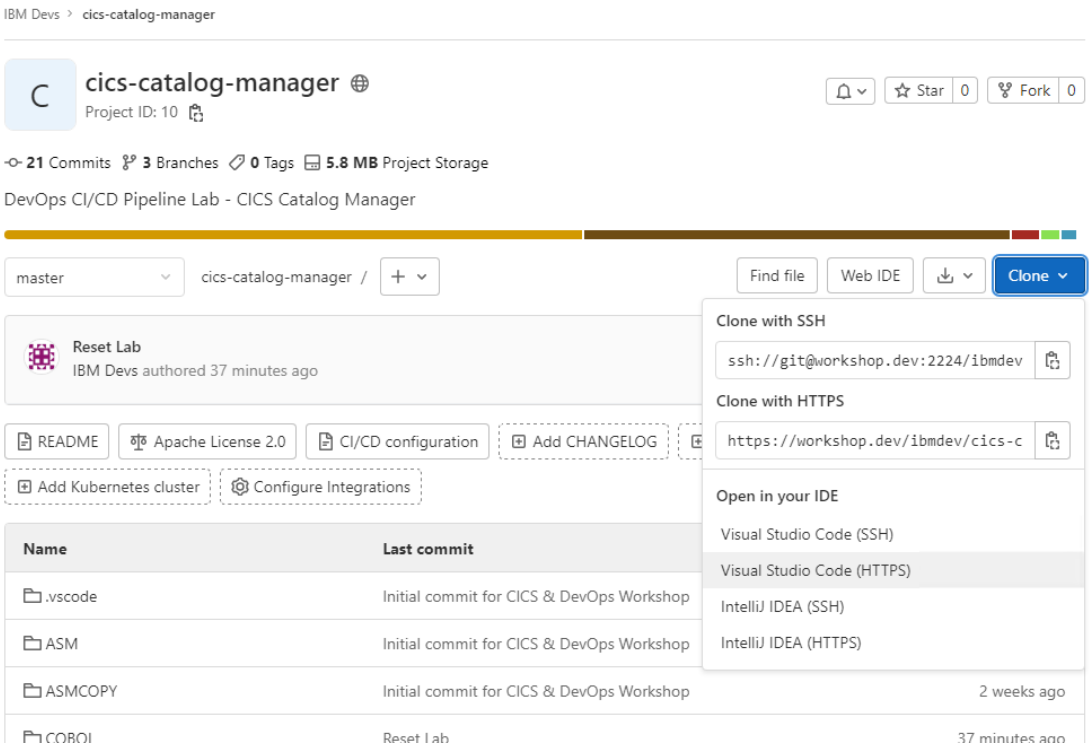
Part 2: Create a clone of the new branch into your IDE

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.

Clone the git repository into your workstation using VS Code

___**1.** Go back to the project homepage in **GitLab** by clicking on the icon at the top of the navbar.

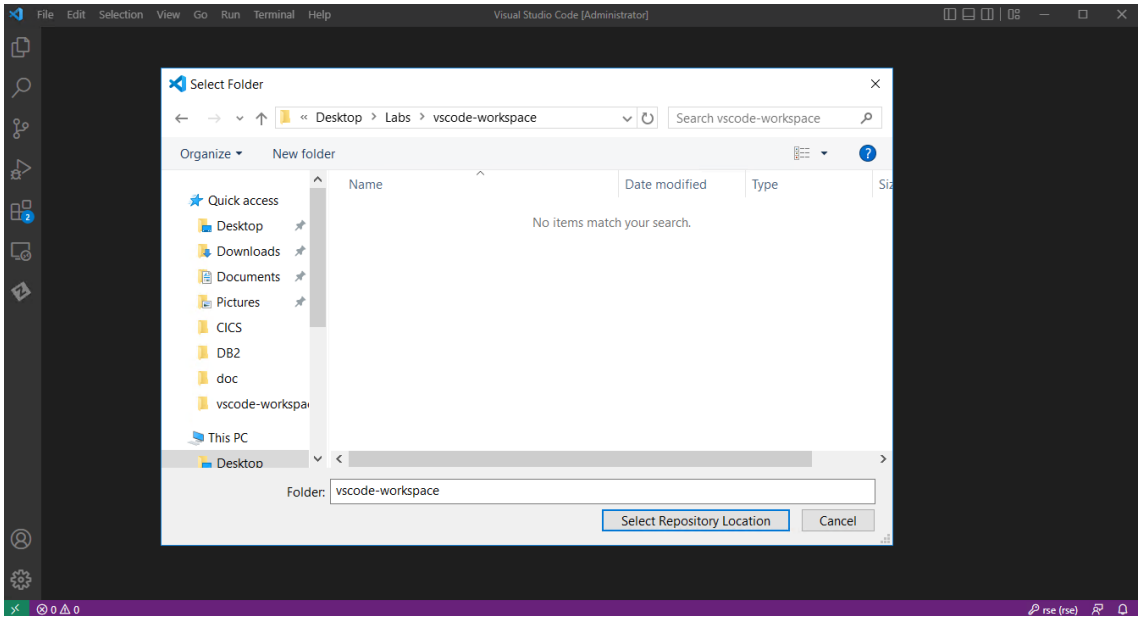
___**2.** Click on the **Clone** button on the right and select **Open in your IDE**, **Visual Studio Code (HTTPS)** option.



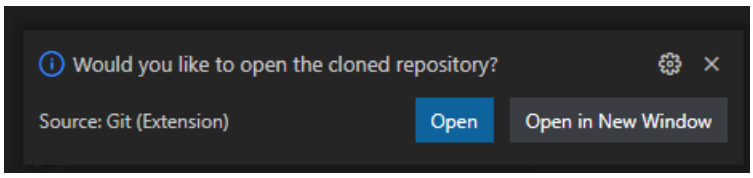
___**3.** VS Code should automatically open (If prompted, click **Open** to confirm the operation).

___**4.** In the **Select Folder** window, Open the **Desktop → Labs → vscode-workspace** folder on and click **Select Repository Location** button. This should be an empty directory.

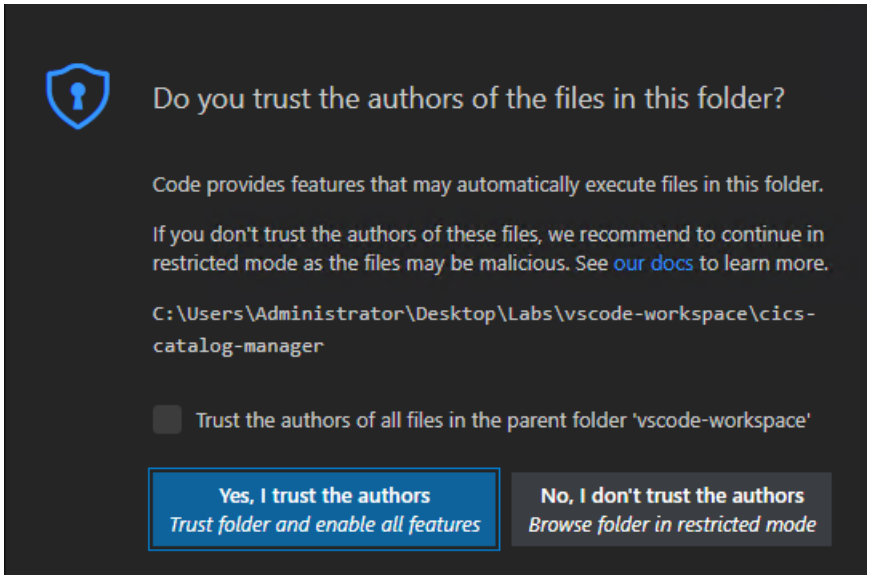
(The full path is `C:\Users\Administrator\Desktop\Labs\vscode-workspace`)



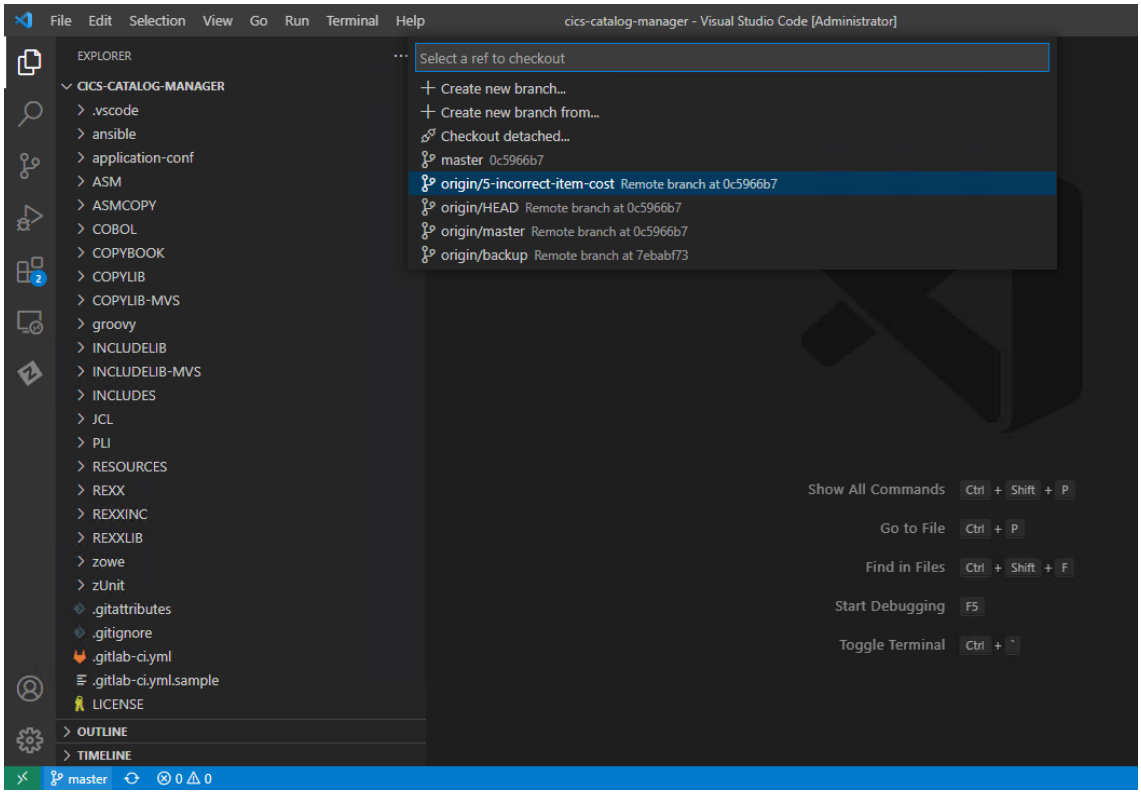
___**5.** Click **Open** button at the bottom pop-up dialog to open the local cloned project directory.



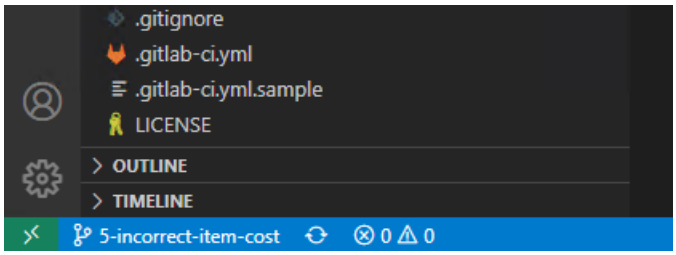
___**6.** Click **Trust** button to trust the source in the project.



___**7.** VS Code has now created a local copy of the Git repository, now very **importantly** as you want to work on the new branch rather than the master, select **origin/*n*-incorrect-item-cost** to checkout as the current branch to work on.



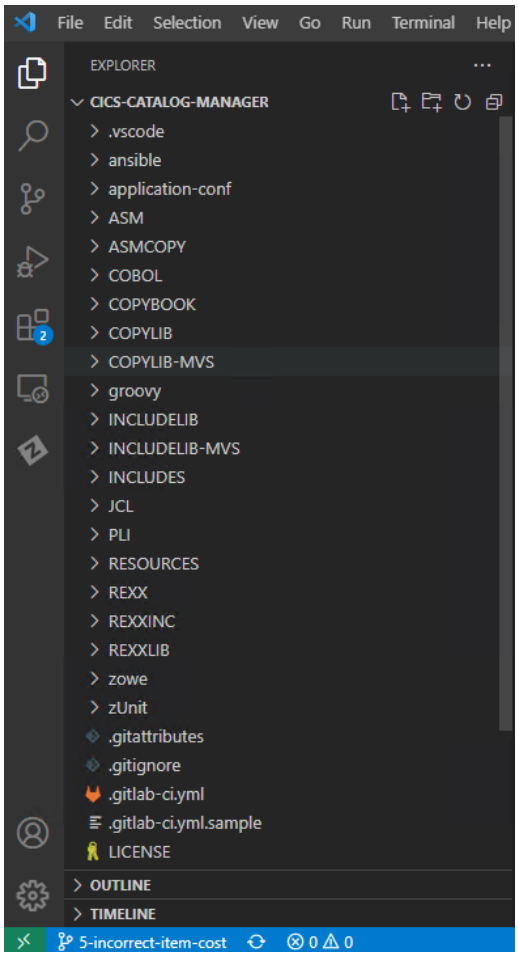
___**8.** **Make sure** the bottom status bar shows the new branch name which means you are now working in an isolated code branch now rather than the main development stream.



Understanding the structure of the z application project

___**9.** Feel free to navigate and explore the content of the project in VS Code and get familiar with the way it's structured.

___**10.** Some of the key elements in the project includes:



- **.vscode** contains the project specified settings for the VS Code environment so that all the team members will automatically inherit IDE settings and project specific configurations e.g. copybook resolving search order, file type mapping, also environmental settings like remote z/OS debugging port numbers and user-build configurations.

- **application-conf** contains the project-specific setting for zAppBuild framework, which leverages IBM Dependency Build for z/OS for the build, it includes configuration on how to build the project, more specifically for our project it defines things like the folders of the COBOL source code, copybooks etc. And impact resolution rules, and last but not least the compiler options for each type of the program as well as the link editor options.

- The application source is located in **language-specific folders** e.g. Assembler, COBOL and PLI, JCL, REXX. In our lab, we only focus on COBOL and COPYBOOK, but you can see how they can be organized for any real project.

- **.gitlab-ci.yml** file is the main CI/CD pipeline definition file for GitLab, which will be actioned whenever changes are committed. This is a YAML document that defines the different stages of the DevOps pipeline and will be executed automatically by the runner when triggered. Similarly, if you use Jenkins for pipeline orchestration, you will have a **Jenkinsfile** located in the project repository managed by the SCM too.

___**11.** As you can see the whole concept for the way the project is structured is to have all the configuration managed in the repository like source code (configuration-as-code) so that it becomes portable, and the moment developer clone the project into the IDE they will have a workspace ready for all the tasks they need.

___**12.** Without making any changes, you can close any open files now.

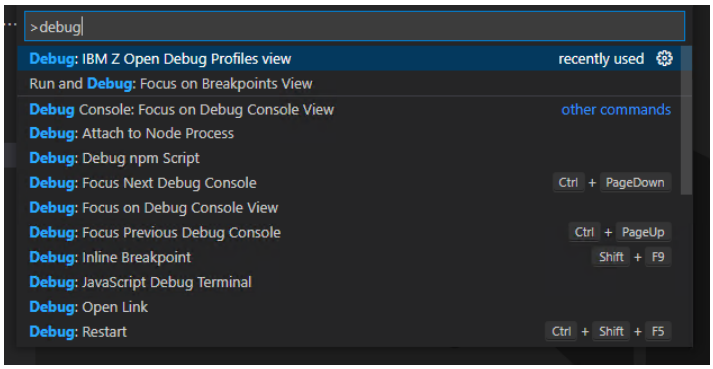
> **Note:**
>
> IBM Dependency Based Build (DBB) is an intelligent build system for traditional z/OS applications written in languages such as COBOL and PL/I that allows the analysis of build dependencies between objects.
>
> The goal of DBB is to provide automation capabilities that can be used on z/OS
> IBM DBB is a standalone framework (it does not require a specific source code manager or automation tool) to simplify the process of building code on z/OS based on a modern scripting language.
>
> z/OS development teams have the freedom to choose a modern software configuration management (SCM) tool, such as Git, and continuous integration tools, such as Jenkins or GitLab, to build traditional z/OS applications written in COBOL or PL/I.
>
> DBB allows you to standardize DevOps processes and practices across multiple platforms

Part 3: Debugging the application at statement level

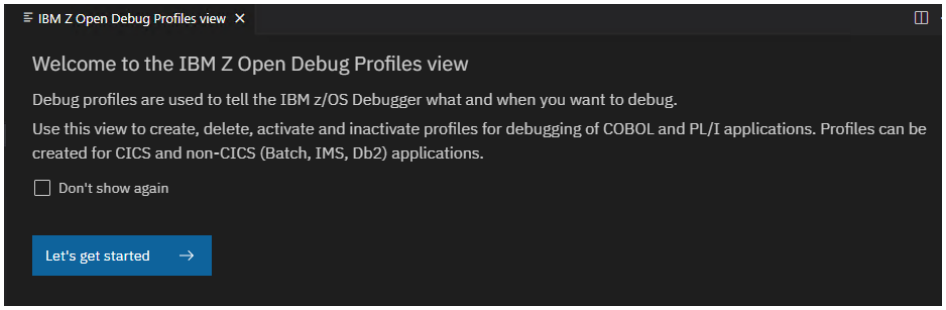
We know our program has some defects with the incorrect item cost, but the best way to identify the root cause is to use the debugger and trace the program line-by-line when in execution. In this part of the lab we will use the z Open Debug to connect to IBM z/OS Debugger, set up a debug profile which intercept any call to the main catalog program DFH0XCMN and “park” its execution, so that developer can use VS Code to connect to the "parked" debug session to inspect and control the execution.

Creating the Debug Profile

___**13.** In the VS Code window, press `Control-Shift-p` to bring up the **command palette** and start typing `debug profile` and select **Debug: IBM Z Open Debug Profiles view** from the list.

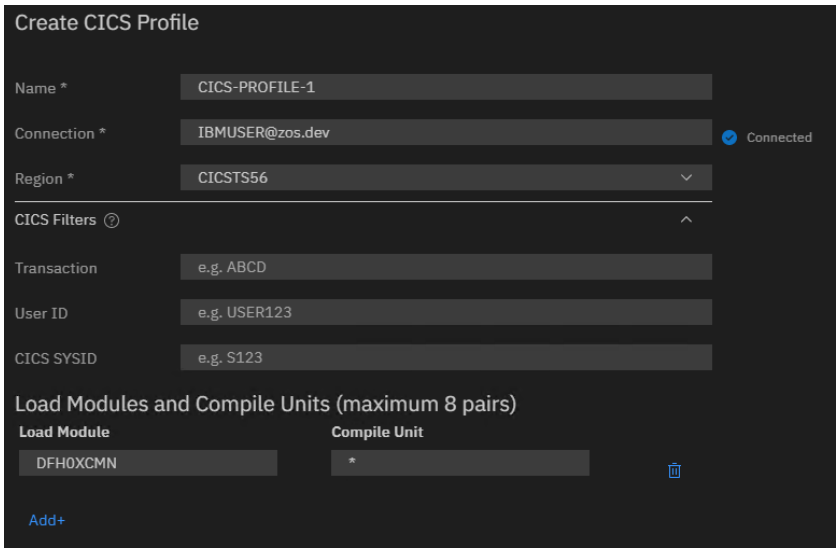


___**14.** In the welcome screen read the description and click **Let's get started** button.



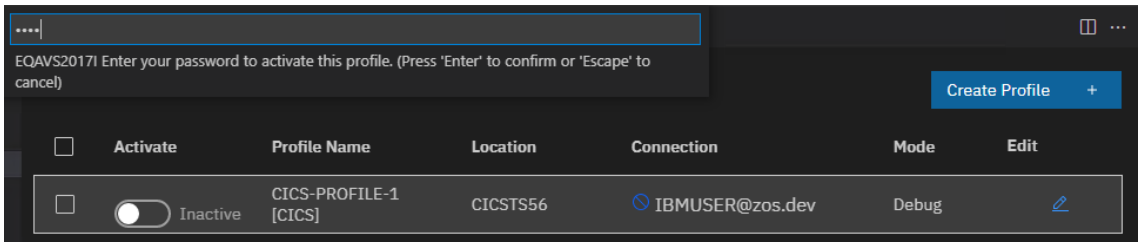
___**15.** Next step click **Create Profile** button and select **CICS**

___**16.** On the next screen you will be able to select what you want to debug. you will notice that it's now connected to the z/OS and listing all available CICS regions, select **CICSTS56.** 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 click **Add+** button under **Load Modules and Compile Units** and specify load module name `DFH0XCMN` 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.



___**17.** Leave all the other fields as default and click **Create** button to finish, you will be taken back to the previous page with **CICS-PROFILE-1** defined with **inactive** status.

___**18.** Click on the **Activate** toggle switch, you will be asked to type your password, type `SYS1` in the pop-up window and press **Enter** and now it should be successfully activated.



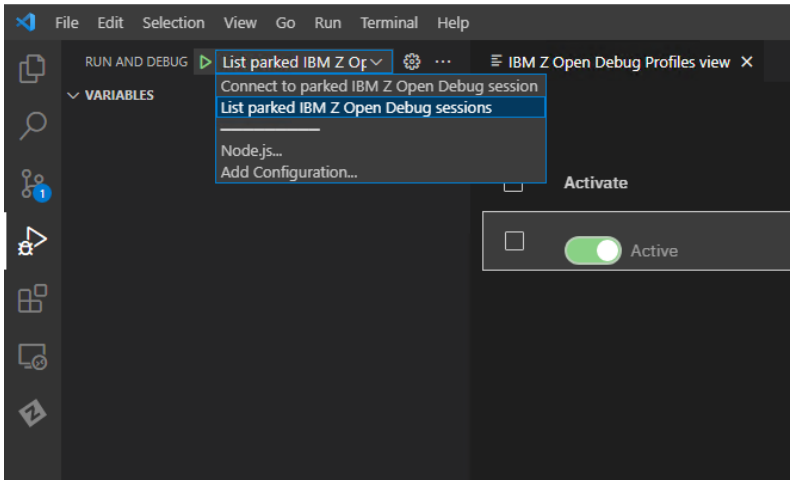
Invoke the program to start debugging

___**19.** Now that your debug profile for **DFH0XCMN** is activated. Go back to the browser and trigger the API call once again using the URL:

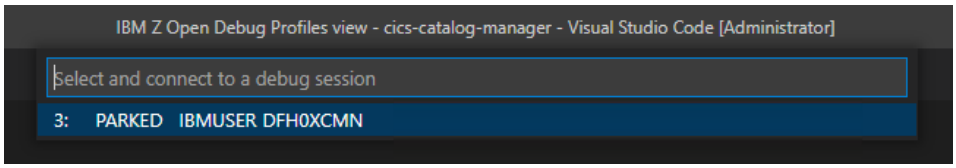
```
https://workshop.dev:9443/items?startItemID=0
```

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

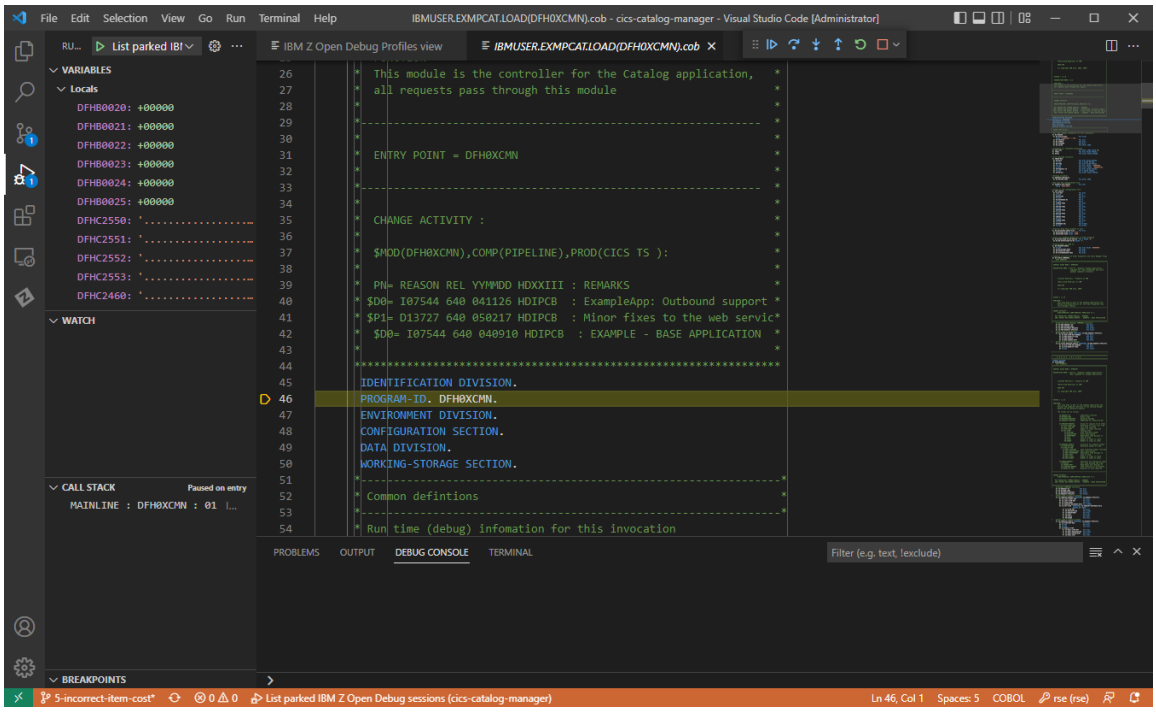
___**20.** Go back to the VS Code and click **Debug** icon on the navbar, and at the top dropdown list choose **List parked IBM Z Open Debug sessions** and click the **green go** button.



___**21.** There should be a pop-up window list the current debug session for **DFH0XCMN**, click on the session name to start the debug session.



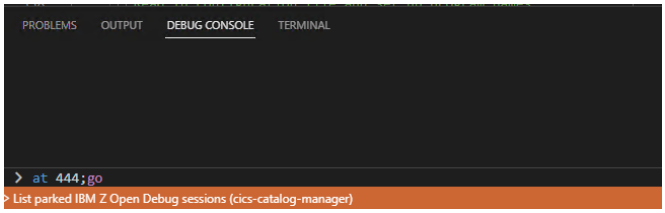
___**22.** In the debugging session, you could find source code inside the editor, see variables, call stack and watch expressions as you step. Similar to the debugging experience for other programming languages, you can step through a running program, check variable values, set breakpoints etc.



___**23.** You can explore the interface by:

- Try to press **F10** 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 right-click to add into watch window.
- Click in the area before the line number you can set or clear breakpoint on that line.

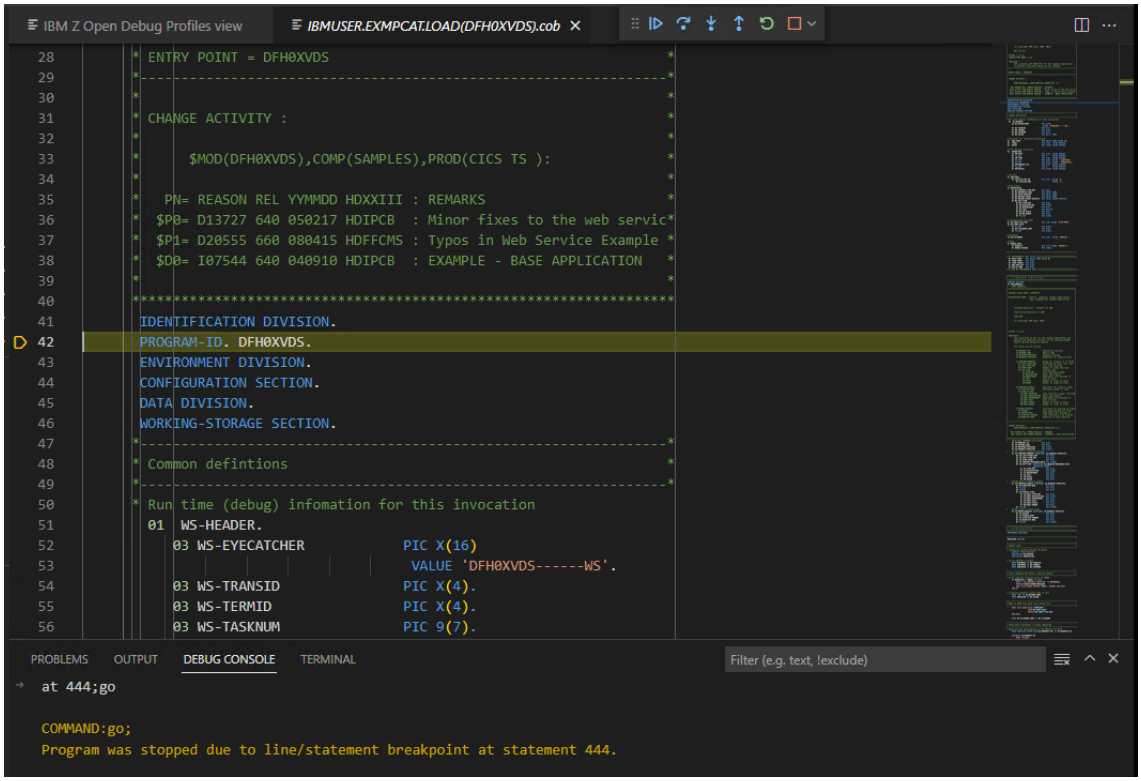
___**24.** type `at 444;go` and **Enter** in the **Debug Console** at the bottom of the screen to set a breakpoint on line 444 and continue.



___**25.** 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.
```

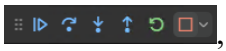
___**26.** Press **F11** or click **Step-Into** button from the top bar  to trace into the subprogram, and we will see we are now debugging in the program **DFH0XVDS**.



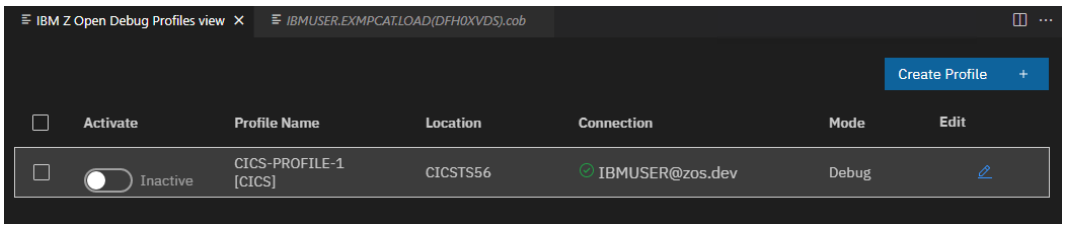
___**27.** Type `at change WS-COST` and **Enter** in the Debug Console at the bottom of the screen to define a dynamic breakpoint only when the content of WS-COST variable is modified.
Press **F5** or the **continue** button  about 3 times until reaching **line 401**.
Where you can clearly see the **WS-COST** has been overwritten with a value 999.99 by a **MOVE** statement. This the root cause of the defect which we will fix later.

```
394
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

___**28.** Now we know the root cause of the defect. We now stop the debugging session by press **Shift-F5** or click the stop button  , now you are disconnected the debugging session and you should also be able to see the API response now returned.

___**29.** IMPORTANT. Don't forget to deactivate the debug profile in the **IBM Z Open Debug Profile view**. Otherwise, the program will be suspended for debugging again when we later test our program. When deactivating the profile, type `SYS1` in the password window.



> **Note:**
> IBM Z Open Debug is a remote debug interface to be used with z/OS Debugger to debug z/OS COBOL, PL/I, and assembler applications in Wazi for VS Code and Wazi for Dev Spaces IDEs.
>
> IBM z/OS Debugger is a host component that supports various debug interfaces, like the Eclipse and Visual Studio Code IDEs.

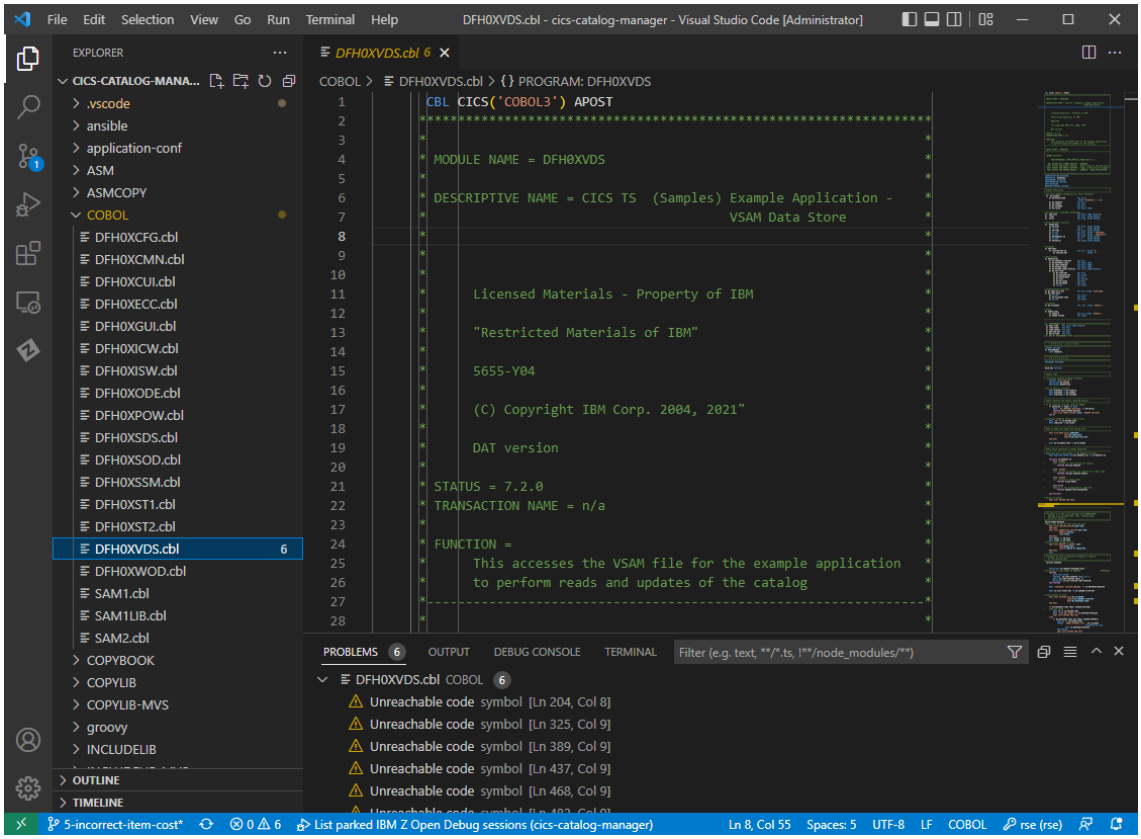
Part 4: Make the code change

In this part of the lab, you will learn how to leverage the code editing capability in VS Code to fix the code.

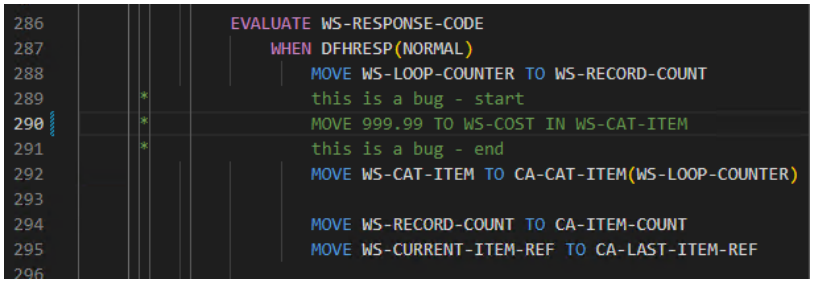
Open the source code in the editor

___**1.** Click on the top **Explorer** button  in the left navigation bar to go back to the local project we just cloned.

___**2.** Expand the **COBOL** folder and click **DFH0XVDS.cbl** to open the source code in the editor.



___**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.



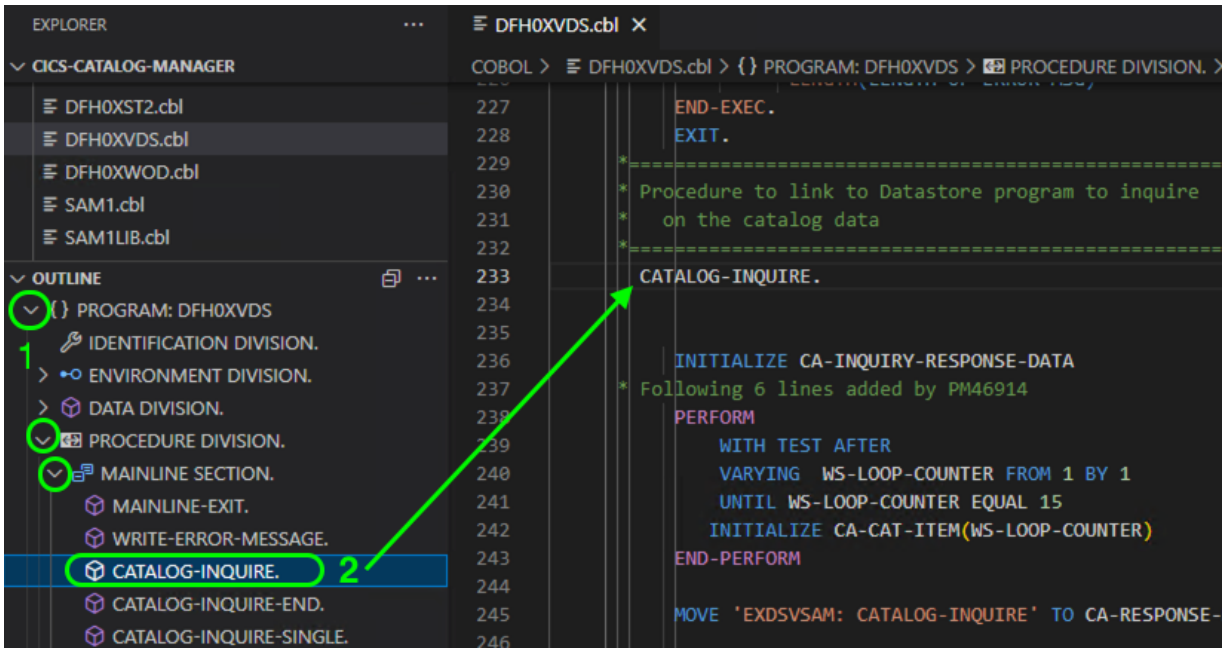
___**4.** Press Ctrl-S to save the file to complete the bug fix.

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

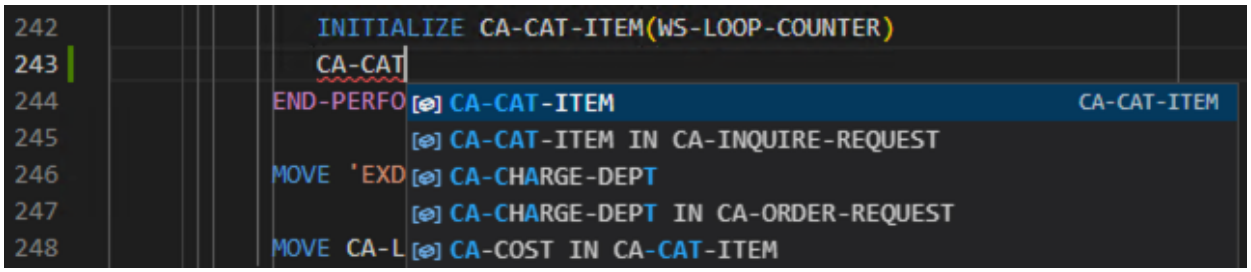
left of each header to expand them, and quickly jump to any part of your code by clicking on it in the Outline view.

- In the **Outline view** (located in the lower left of the VS Code 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.



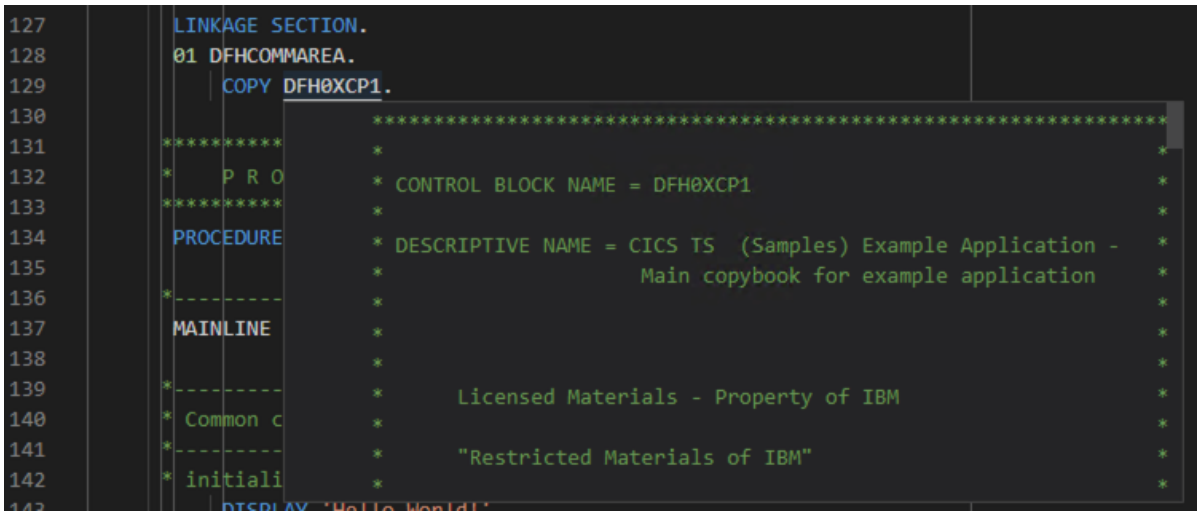
___**6.** Code completion: Use IBM Z Open Editor code completion feature to view and select from a list of commands, defined variable names, and code snippets when you start typing a command, variable, or paragraph name.

- 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 beginCCA typing the variable name: "`CA-C`". A selection list automatically 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.



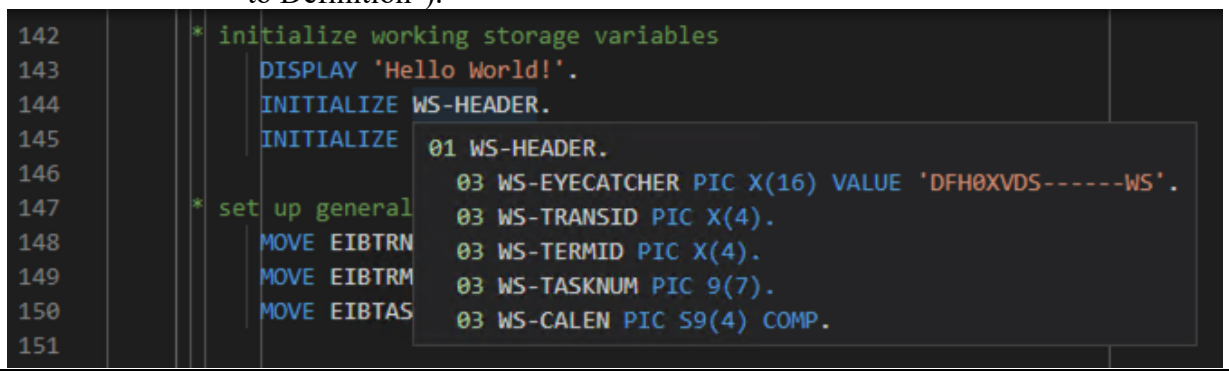
___**7.** Preview copybooks: Preview the contents of a copybook in the COBOL program.

- At line 129 in DFH0XVDS.cbl, hover your mouse cursor over the copybook name "`DFH0XCP1`" in the COPY statement to view the copybook preview.
- You can also open the copybook in a separate editor by pressing **Ctrl+Click** (Windows) or **Cmd+Click** (Mac) on the hover pop-up. (You can also do this by scrolling to the bottom of the hover pop-up and clicking "Follow link".)



___**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 "Go to Definition").



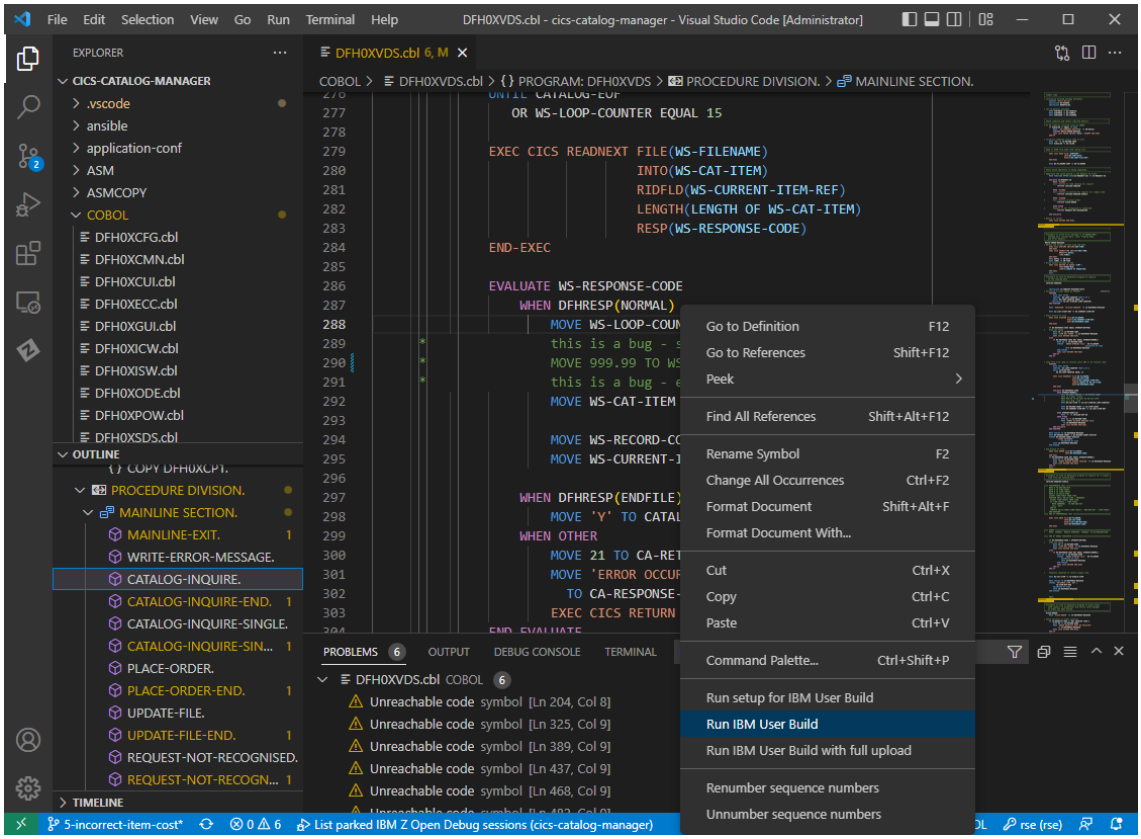
___**9.** Close the file without saving it.

> **Tech-Tip:**
> For additional editing features, you can check out the IBM Z Open Editor documentation: https://www.ibm.com/docs/en/cloud-paks/z-modernization-stack/2022.3?topic=files-making-cobol-pli-code-changes

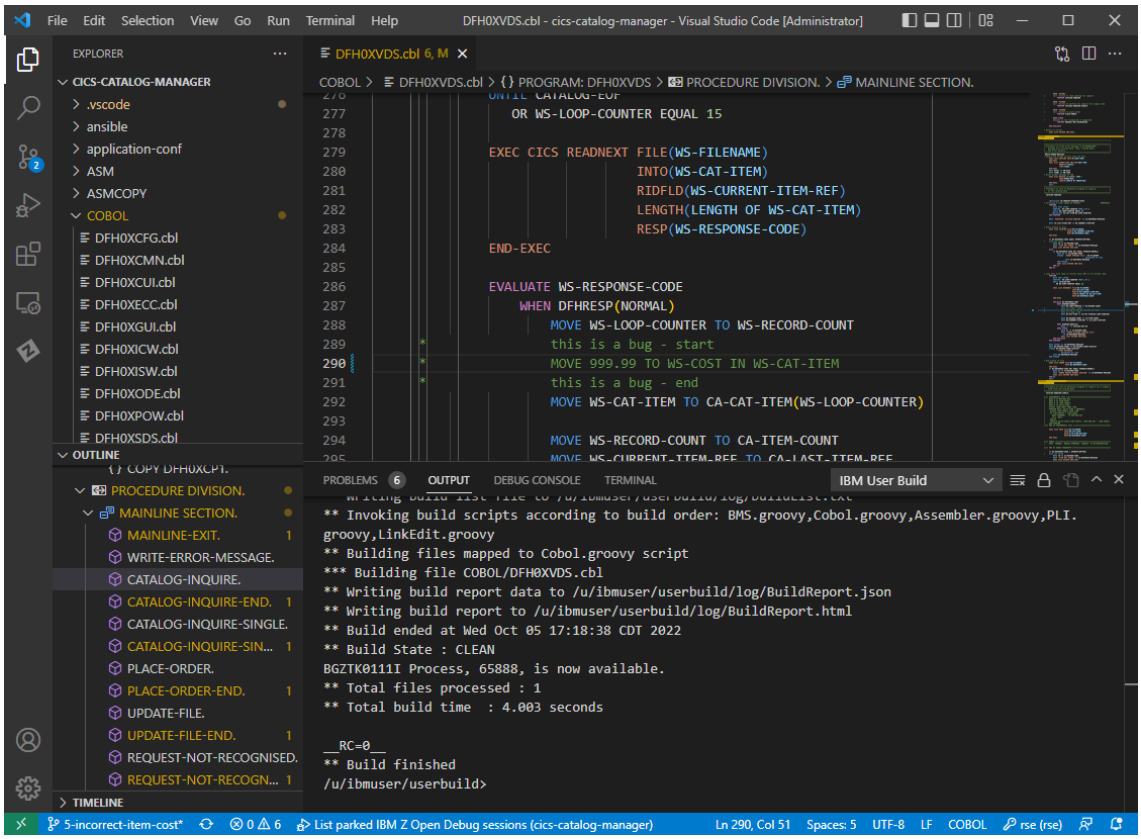
Part 5: Run a user build to build the program in your own workspace

In this part of the lab exercise, you will run a user build that will upload only the changed code with all its dependencies to the z/OS environment and use IBM DBB to build a load module in your personal PDS load library dataset. And detail build report is then transferred back to your workstation IDE for review.

___**1.** With the **DFH0XVDS.cbl** program saved, right-click within the editor area and choose "Run IBM User Build"



___**2.** Switch to the **Output** panel down below and monitor the output information for **IBM User Build process** until you see the **Build State: CLEAN**

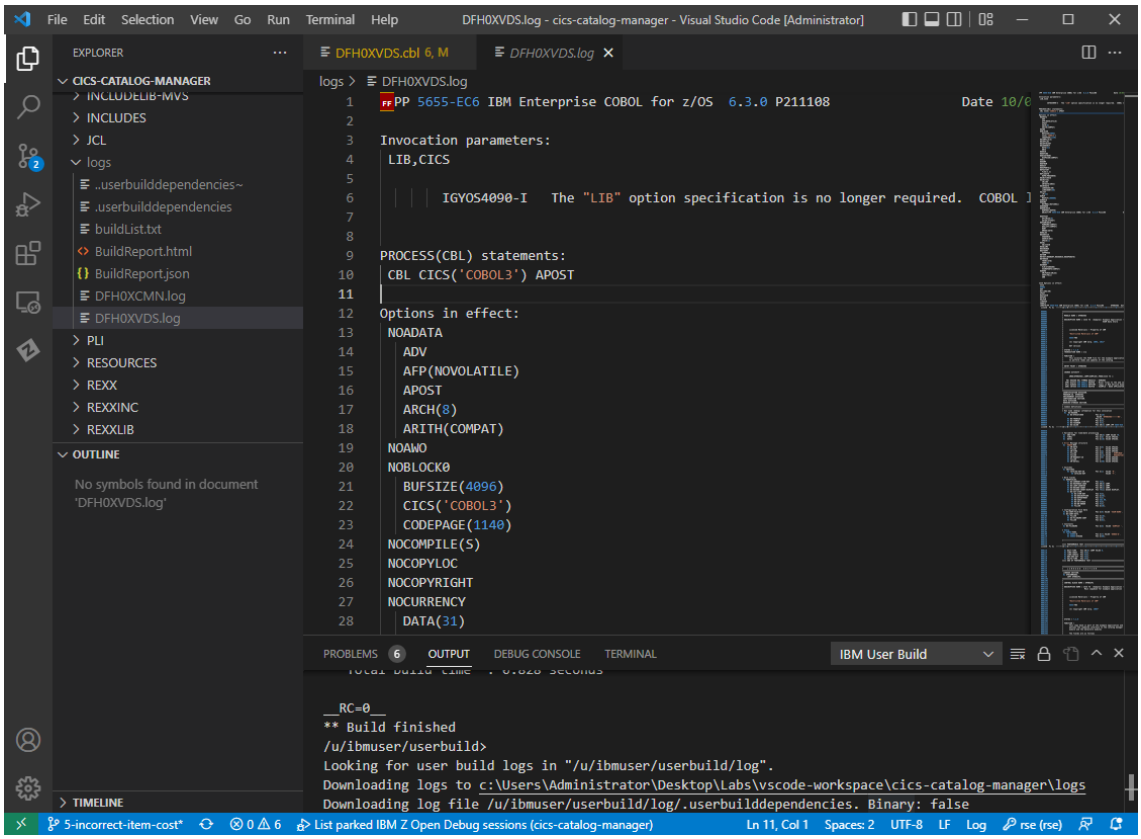


___**3.** Scroll up if you want to understand the process, take note of the high-level qualifier of the user build dataset library which will be used in the next step which is **IBMUSER.USERBLD**

```
BGZTK0108I Process, 65888: /u/ibmuser/userbuild/dbb-zappbuild/build.groovy --
userBuild --workspace /u/ibmuser/userbuild --application . --hlq
IBMUSER.USERBLD --outDir /u/ibmuser/userbuild/log --dependencyFile
/u/ibmuser/userbuild/log/.userbuilddependencies
/u/ibmuser/userbuild/COBOL/DFH0XVDS.cbl
```

___**4.** Optionally you can review the build output log files including the compiler listing under the project directory `/logs`

- It is a known issue that sometimes you might need to rerun the user build to see the `logs` directory.

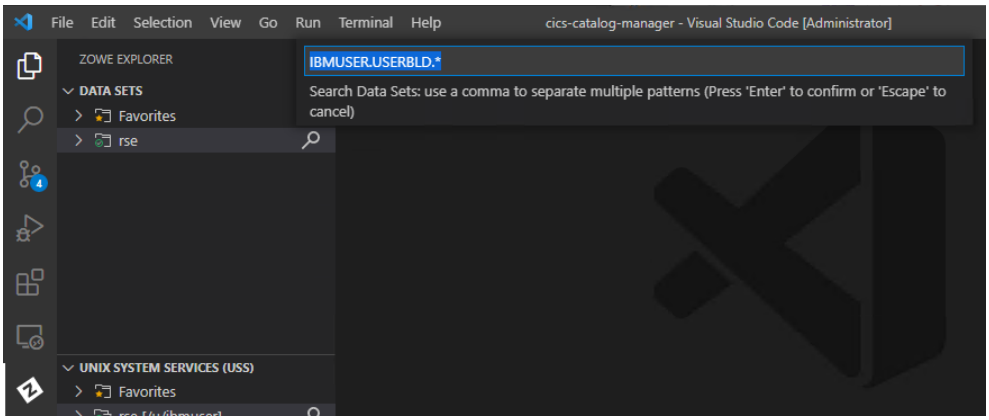


___**5.** As you can see in the log, while we ran the user build for program **DFH0XVDS.cbl**, the **DFH0XCP1.cpy** is also uploaded to z/OS for the build because it has been identified as a dependent include file that's required for building the program.

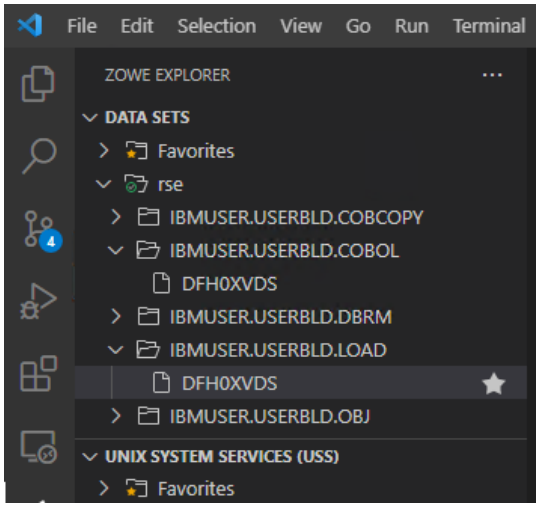
```
Checking if include files were updated since the last build...
Updating 1/1 local include files.
Uploading file C:\Users\Administrator\Desktop\Labs\vscode-workspace\cics-catalog-manager\COPYBOOK\DFH0XCP1.cpy to /u/ibmuser/userbuild/COPYBOOK/DFH0XCP1.cpy with encoding ibm-1047
```

___**6.** Optionally use the **Zowe Explorer** icon  in the navbar to connect to the remote z/OS to check the load module from the user build.

___**7.** Under the **DATA SETS** section, click the pre-configured connection profile **rse** and click the magnifier icon to create a dataset filter called `IBMUSER.USERBLD.*` and hit **Enter** TWICE**.**



___**8.** You can see the load module has been compiled into **IBMUSER.USERBLD.LOAD**. In the next part of the lab we will run early integration test against this load module.



Note:
The User Build capability of the IBM Z Open Editor VS Code extension helps COBOL developers to leverage IBM Dependency Based Build (DBB) toolkit right from their local VS Code or Eclipse Che development environment. A developer who is working on a COBOL application can run a user build to compile and link programs before the code is ready to be exposed to the repository for others to use. With user build, you can compile your program without having to perform commits or pushes.

IBM User Build automatically does the following when you click to run a build:
- Uploads only the necessary files and folders to z/OS
- Resolves and uploads COBOL application's copybooks
- Executes DBB user build script on z/OS
- Downloads log folders to your local git directory so you don't have to navigate to the remote z/OS system to view them.

IBM User Build allows developers to do their COBOL development locally using modern tools without context switching.

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 re-run 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.

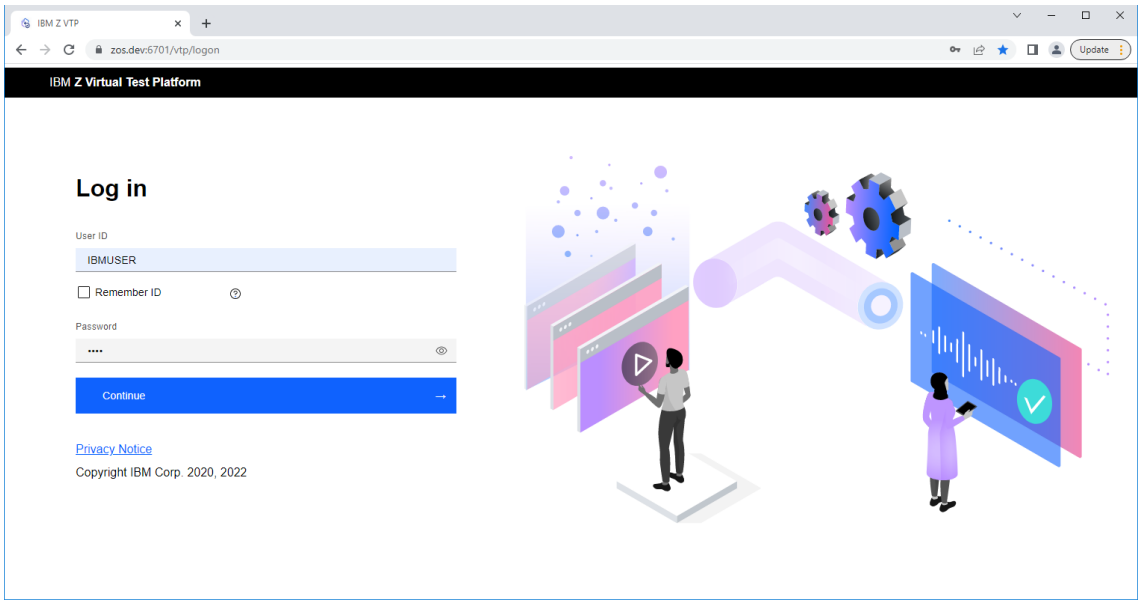
ZVTP can be used independently or in cooperation with other products. For more information, see product document. https://www.ibm.com/docs/en/zvtp/2.0?topic=overview

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.

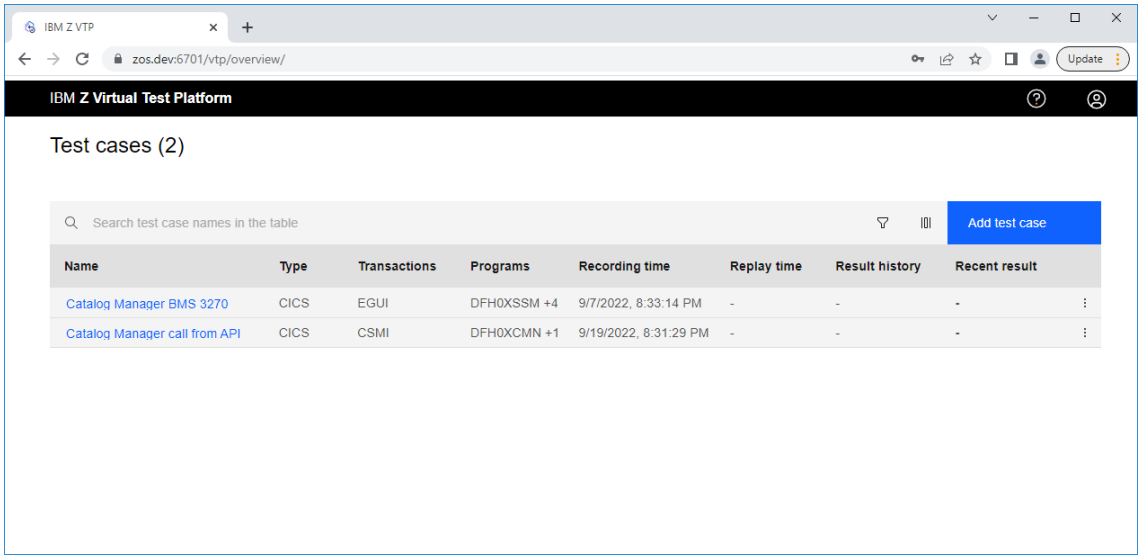
Explore the IBM Z Virtual Test Platform web interface

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

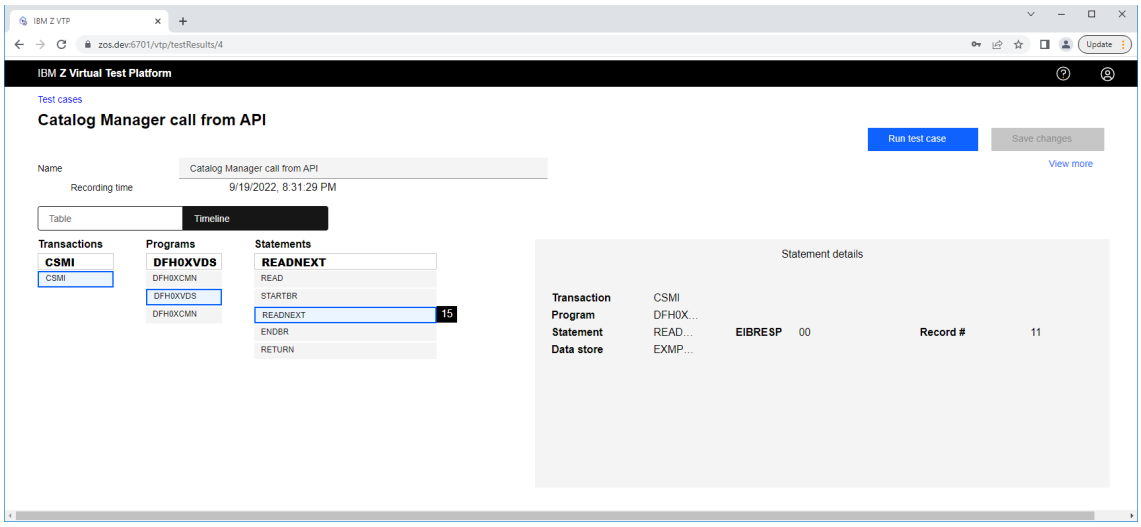
`https://zos.dev:6701/vtp/logon`



___2. Log in use `IBMUSER` and password `SYS1` and click **Continue** to go to the **Test cases** overview page.



___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.

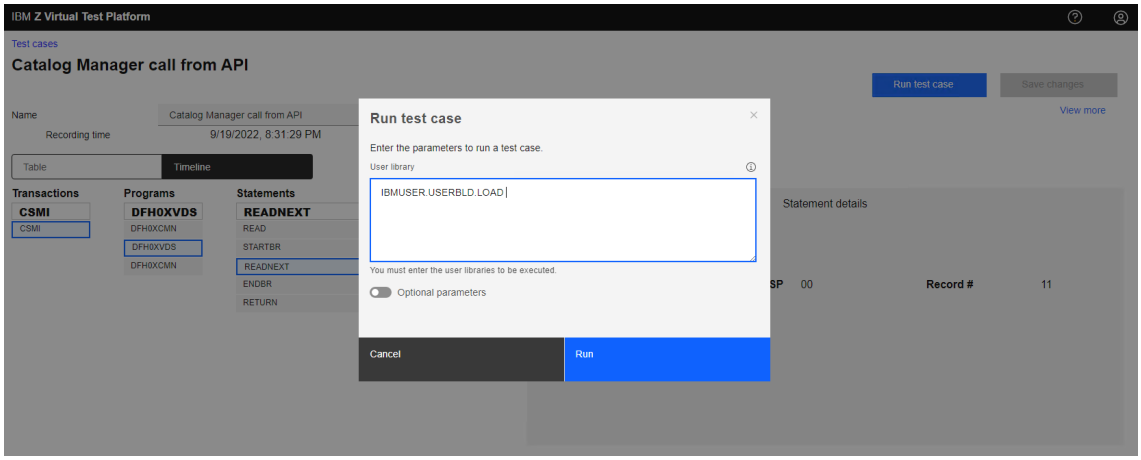


___**4.** Click on the **Timeline** which show all the CICS calls involved, you can see the API invokes **DFH0XCMN** which subsequence LINK into **DFH0XVDS** where the actual VSAM **READNEXT** requests were captured before returning the control back to **DFH0XCMN**.

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

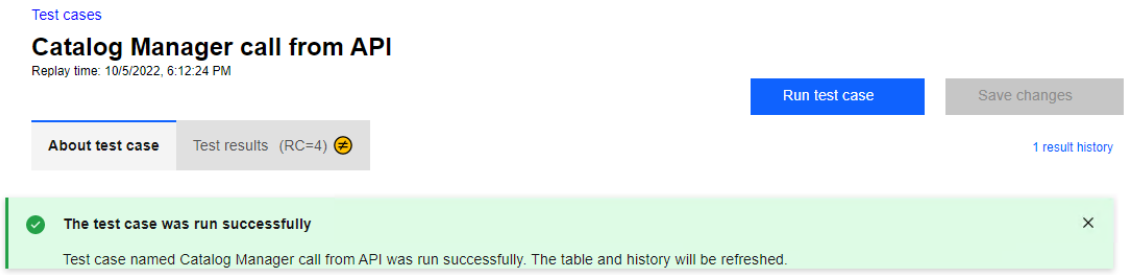
___**5.** Now we can replay the same test case using the user build load module we built in previous step. Click **Run test case** button at the top.

___**6.** Provide the user build load module PDS name `IBMUSER.USERBLD.LOAD` in the **User library** section.



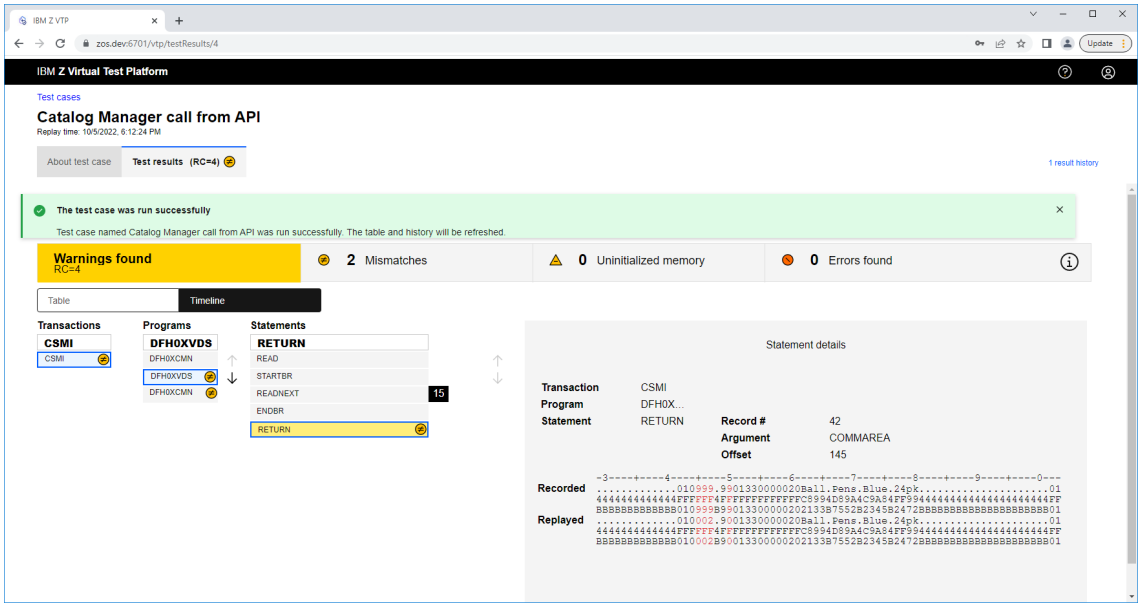
___**7.** Click **Run** button to submit the replay of the test case, IBM Z VTP will replay the test case use updated user build load module and it will compare the test result.

- 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.



___**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.



___**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.

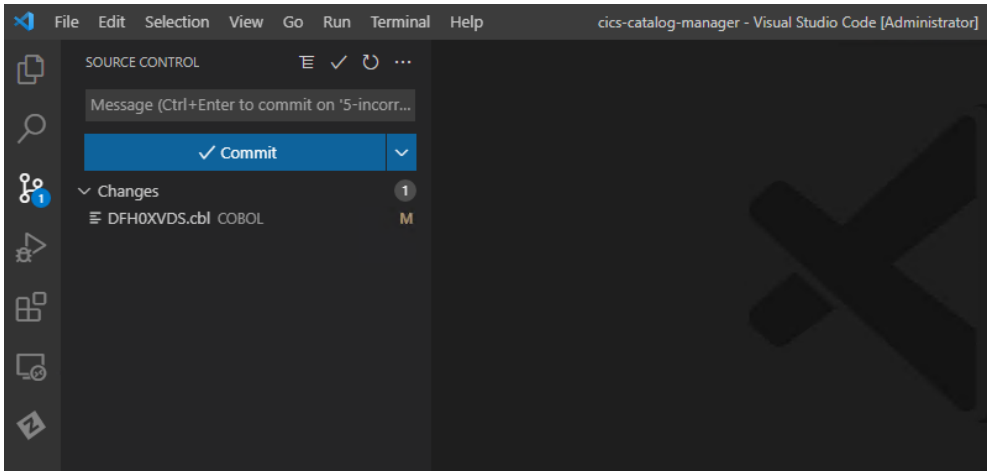
___**11.** This is what we call an **early integration test** so that you can shift left the testing before the deployment, removing dependencies to the external resources so that you can test earlier and more often. But obviously, this is not a replacement for a full integration test which is still required at a later stage.

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

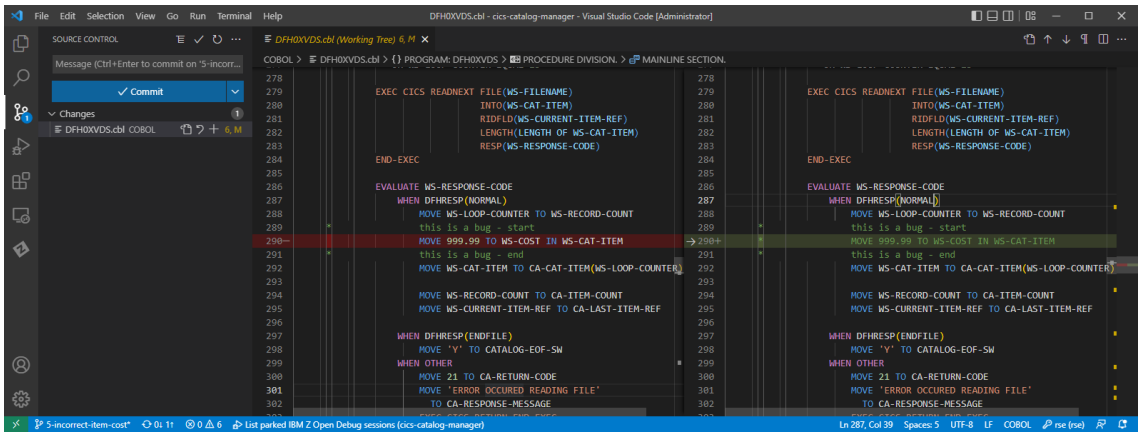
After testing, you are confident the code is ready so you will commit it to your local working branch of the Git repository and merge it to the remote GitLab repo to submit the work you've done. In this lab it will automatically trigger the pipeline for a full application build on the branch, and when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

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

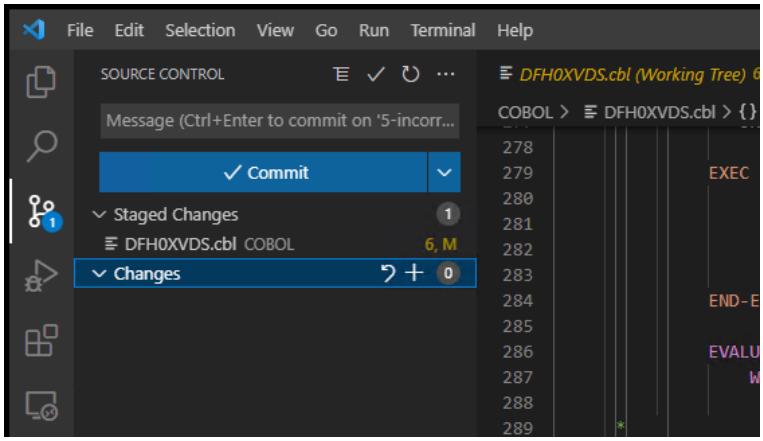
___**1.** Go to the **Source Control** panel using the button  on the left navbar in VS Code.



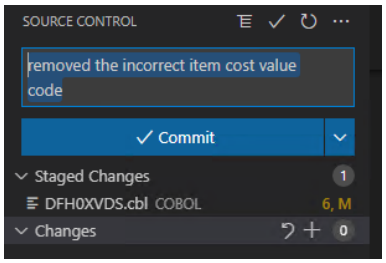
___**2.** Click on the program **DFH0XVDS.cbl** under the **changes** to see the diff between the local copy and the Git branch. You will see the line you have been commented out with the MOVE statement is highlighted.



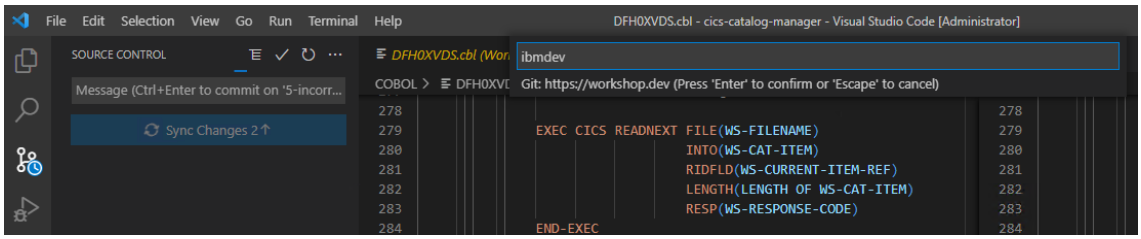
___**3.** Click the + (plus) sign next to the program to **stage** the changes for the commit.



___**4.** Put a commit message e.g. `Removed the incorrect item cost value code` and click **Commit** button to commit to local Git repository.



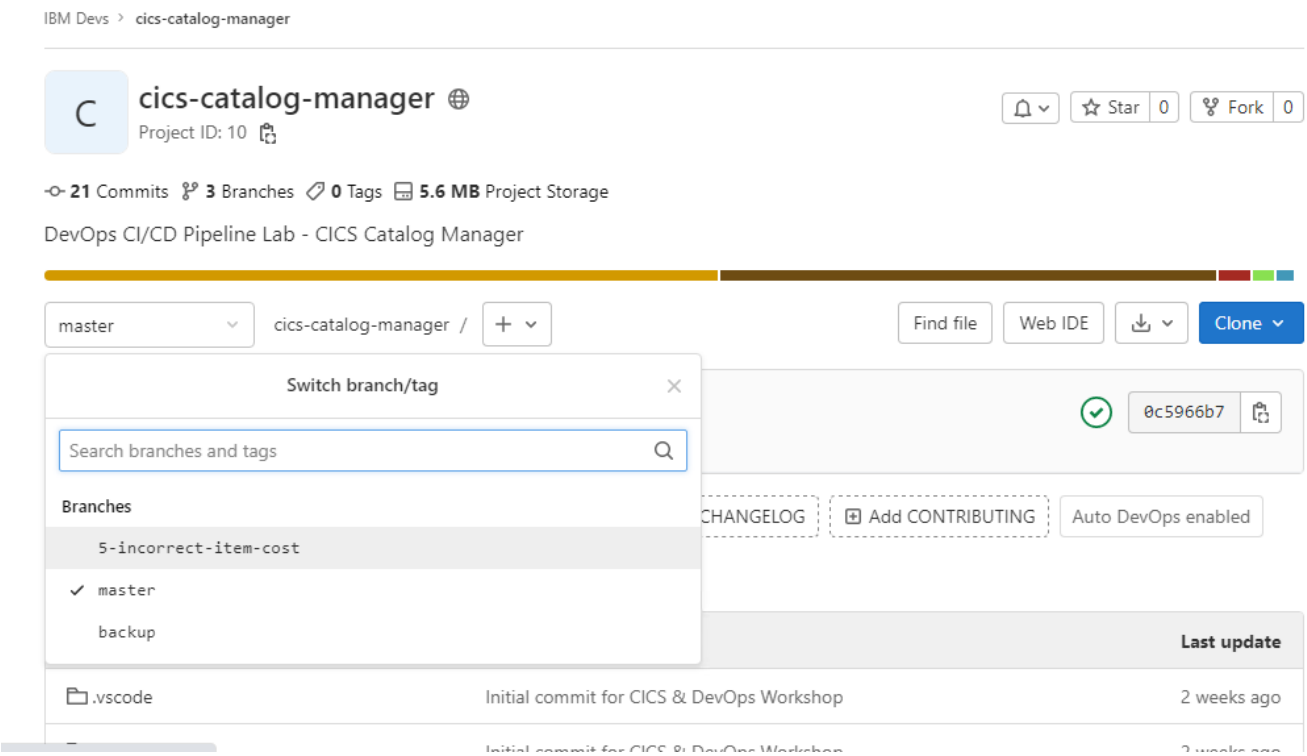
___**5.** Click **Sync Change** button to push the changes in local git repository to the GitLab remote repo, you will need to provide your GitLab credential `ibmdev` and `Passw0rd` in order to be able to push the changes.



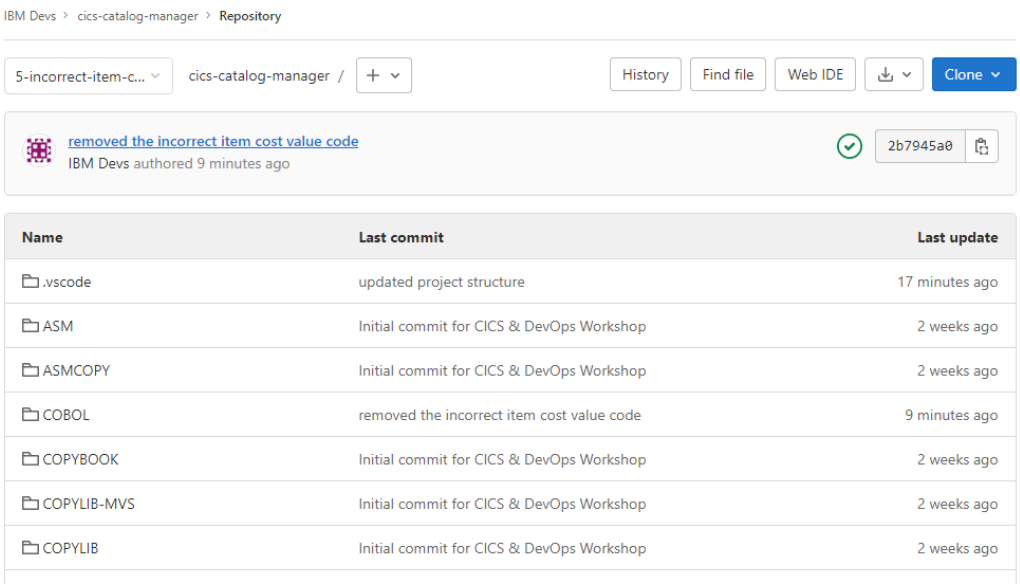
Review the changes on GitLab and pipeline for the full application build

___**6.** Go to GitLab browser window and navigate to the project homepage by clicking the project name on the top left C cics-catalog-manager

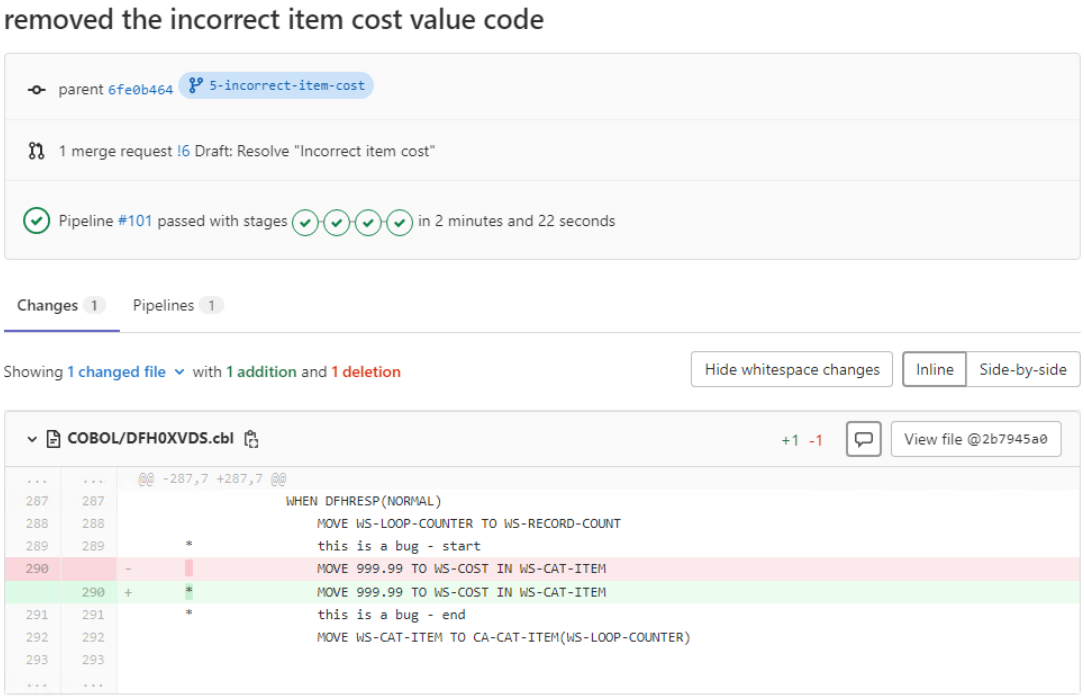
___**7.** Switch the branch to our working branch ***n*-incorrect-item-cost**



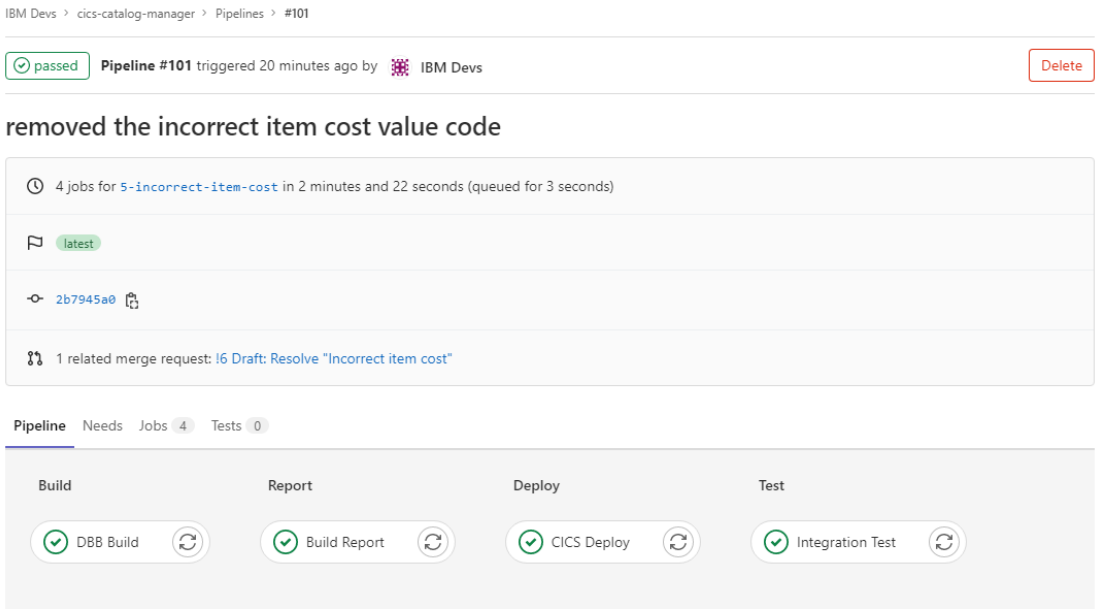
___**8.** Click on **commit message** of the latest commit we just made.



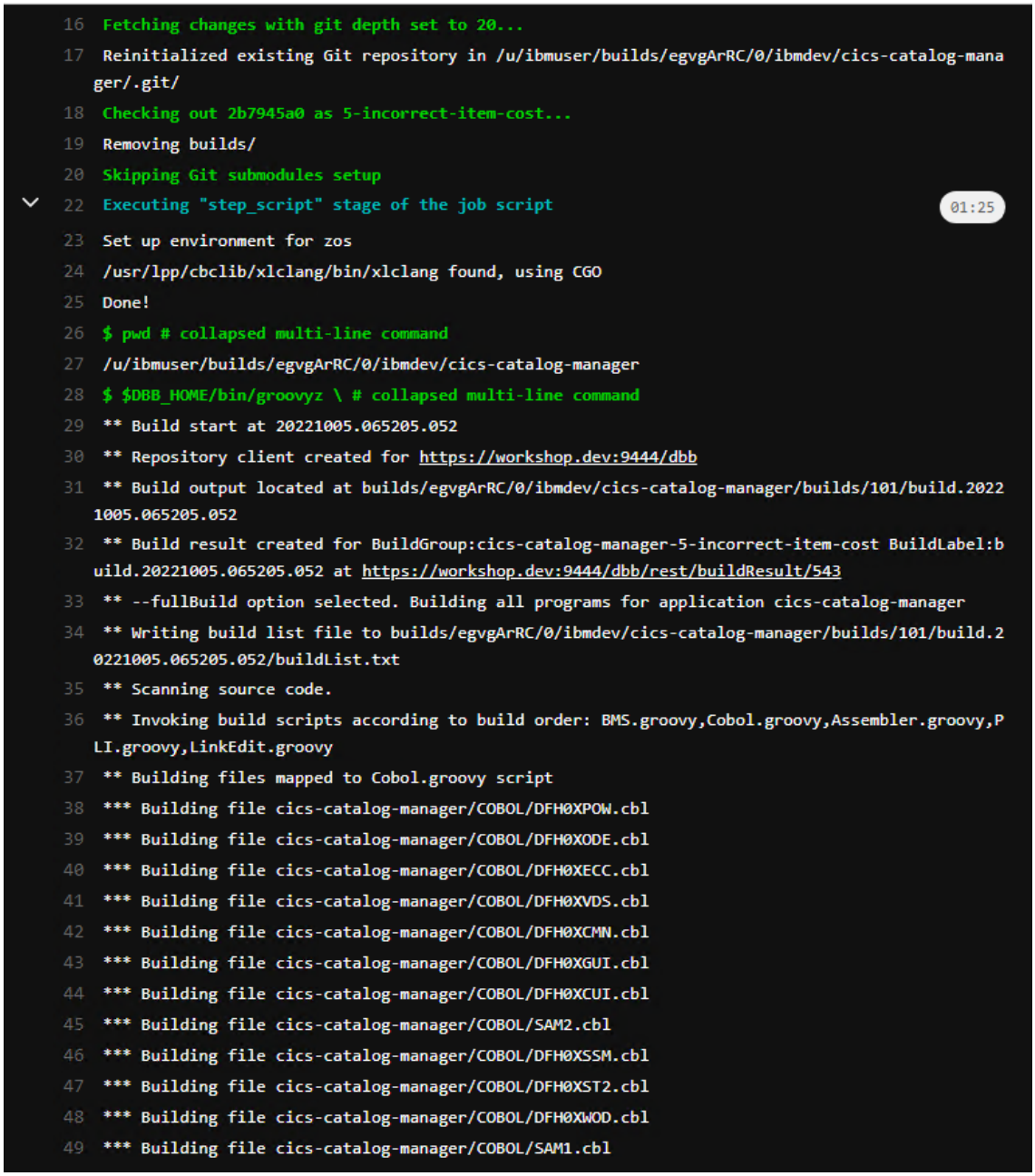
___**9.** This screen shows the commit details and the merge request we submitted for fixing this defect, as well as the pipeline that's triggered by this commit with all the individual jobs' status.



___**10.** Click on the Pipeline id #101 (your id will be different) to inspect the execution of the jobs in this pipeline.



___**11.** 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.

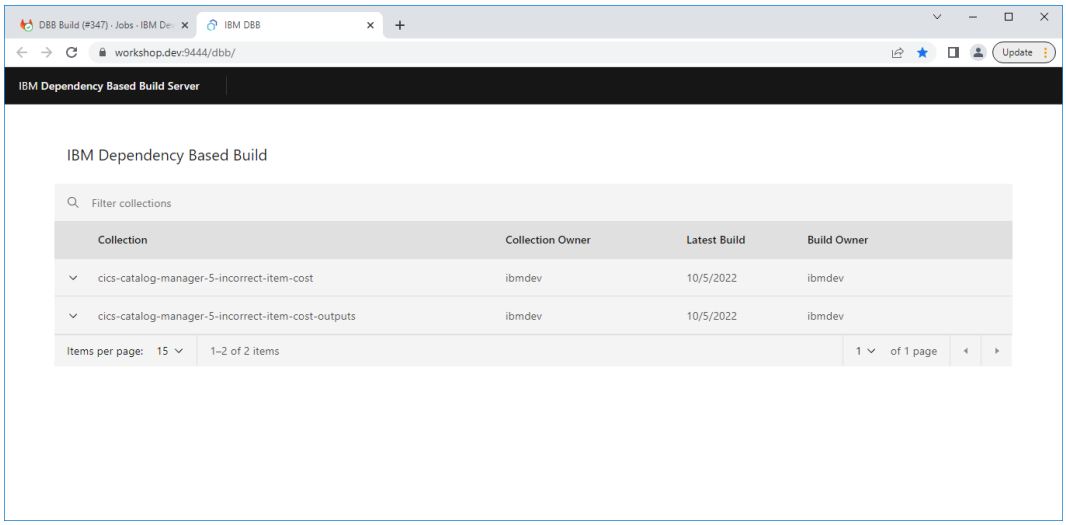


___**12.** You can also log on to the DBB web app to exam all the build results from all developers. **Right-Click** on this link in the DBB log and choose to **Open link in a new tab**.

`https://workshop.dev:9444/dbb`

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

___**13.** When prompted use `ibmdev` and `Passw0rd` to log in and now you can inspect the build collections and detail build result.



___**14.** 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	cics-catalog-manager/COBOL/DFH0XPOW.cbl Show Dependencies	IGYCRCTL	0	IBMUSER.EXMPCAT.COBOL(DFH0XPOW)			DFH0XPOW.cobol.log
		IEWBLINK	0		IBMUSER.EXMPCAT.LOAD(DFH0XPOW)	LOAD	
2	cics-catalog-manager/COBOL/DFH0XODE.cbl Show Dependencies	IGYCRCTL	0	IBMUSER.EXMPCAT.COBOL(DFH0XODE)			DFH0XODE.cobol.log
		IEWBLINK	0		IBMUSER.EXMPCAT.LOAD(DFH0XODE)	LOAD	
3	cics-catalog-manager/COBOL/DFH0XECC.cbl Show Dependencies	IGYCRCTL	0	IBMUSER.EXMPCAT.COBOL(DFH0XECC)			DFH0XECC.cobol.log
		IEWBLINK	0		IBMUSER.EXMPCAT.LOAD(DFH0XECC)	LOAD	
4	cics-catalog-manager/COBOL/DFH0XVDS.cbl Show Dependencies	IGYCRCTL	0	IBMUSER.EXMPCAT.COBOL(DFH0XVDS)			DFH0XVDS.cobol.log
		IEWBLINK	0		IBMUSER.EXMPCAT.LOAD(DFH0XVDS)	LOAD	
5	cics-catalog-manager/COBOL/DFH0XCMN.cbl Show Dependencies	IGYCRCTL	0	IBMUSER.EXMPCAT.COBOL(DFH0XCMN)			DFH0XCMN.cobol.log
		IEWBLINK	0		IBMUSER.EXMPCAT.LOAD(DFH0XCMN)	LOAD	
6	cics-catalog-manager/COBOL/DFH0XGUI.cbl Show Dependencies	IGYCRCTL	0	IBMUSER.EXMPCAT.COBOL(DFH0XGUI)			DFH0XGUI.cobol.log
		IEWBLINK	0		IBMUSER.EXMPCAT.LOAD(DFH0XGUI)	LOAD	

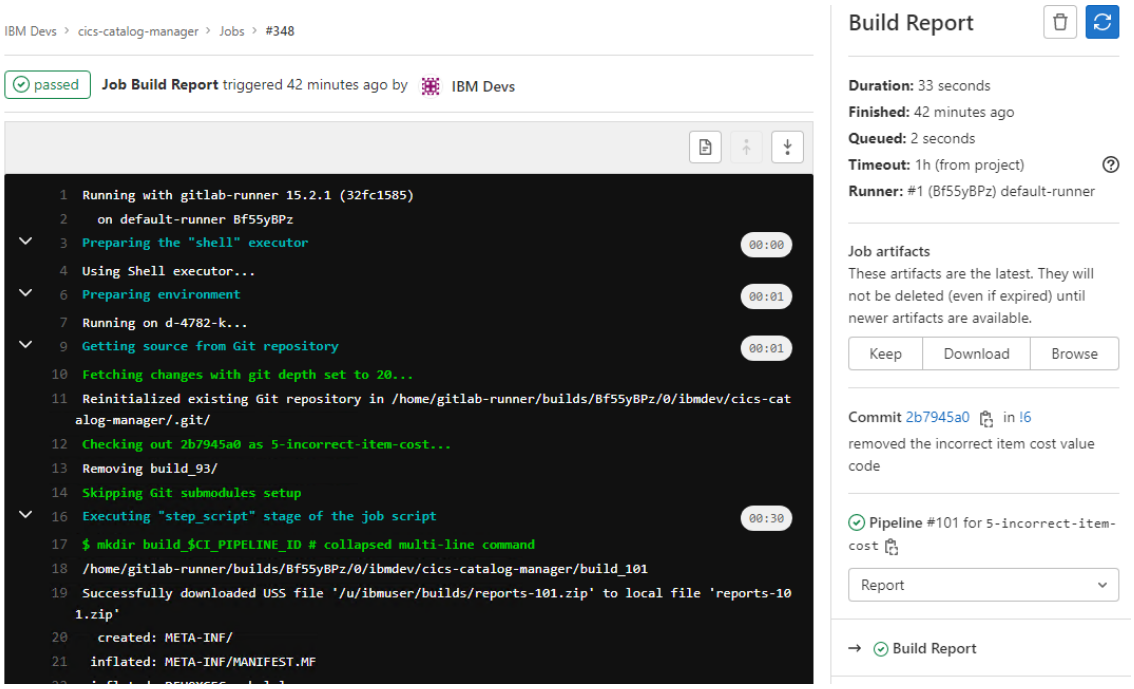
Note on DBB Collection

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.

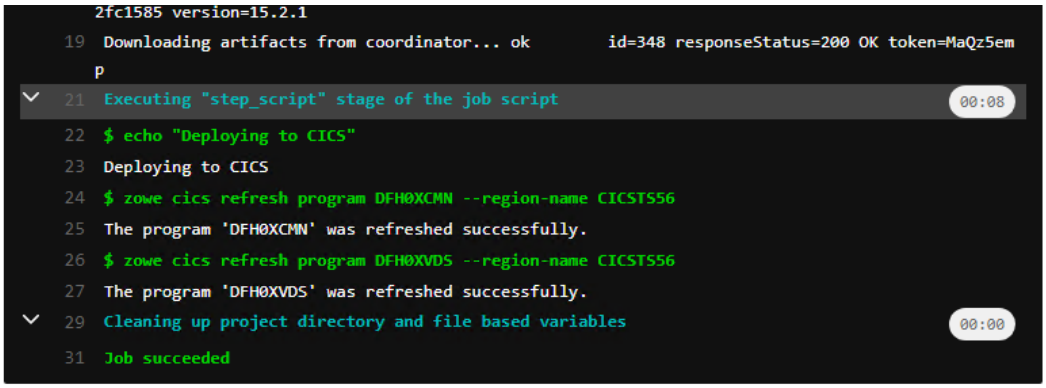
A collection is a repository container for logical files. The scope of a collection is determined by the user but generally a collection contains all the logical files of a Git branch. This way the logical files in a collection can use the scanned file's relative path from the sourceDir as a unique identifier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

___**15.** 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.



___**16.** 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.



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.

___**17.** 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.

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:

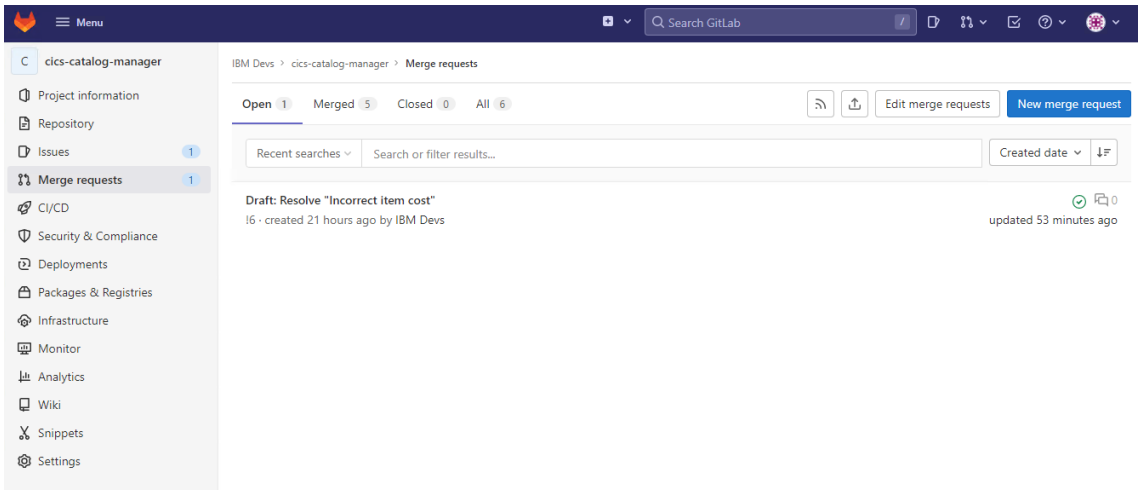
Single Program - Build a single program in the application.
List of Programs - Build a list of programs provided by a text file.
Full Build - Build all programs (or buildable files) of an application.
Impact Build - Build only programs impacted by source files that have changed since the last successful build.
Impact Build with baseline reference - Build only programs impacted by source files that have changed by diff'ing 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

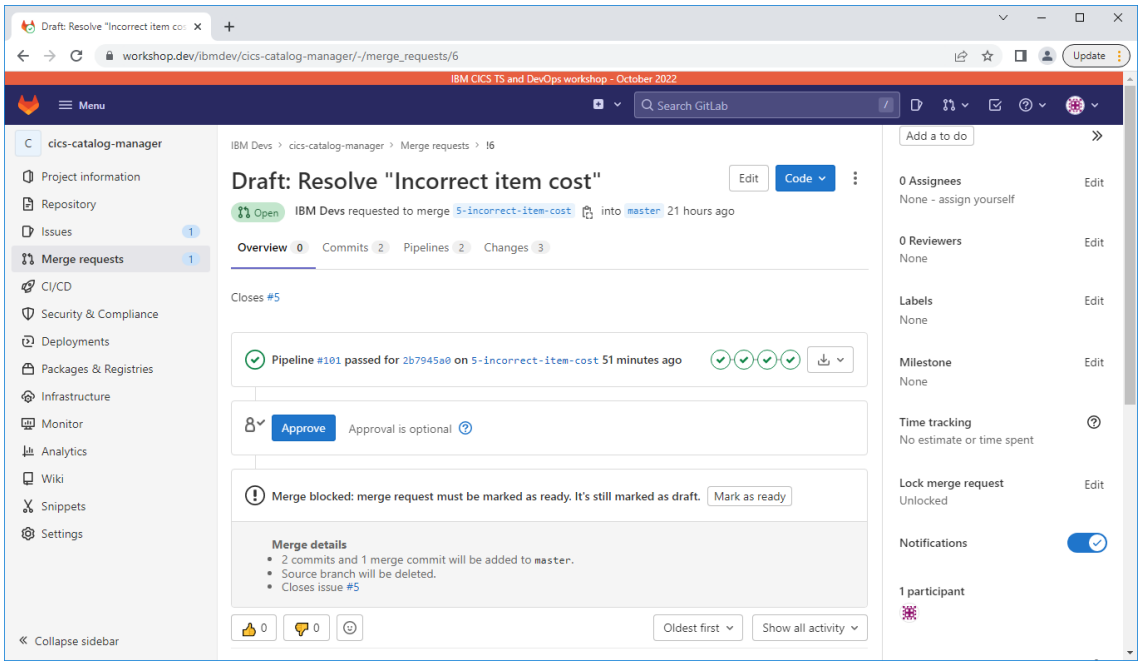
Complete the merge request to submit work to the master branch

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

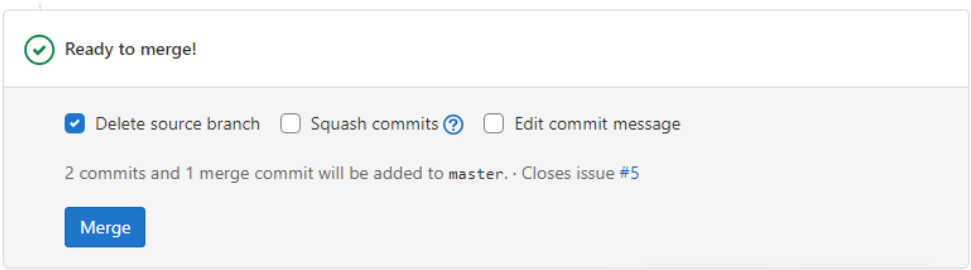
___**18.** Click on the **Merge Request** in the GitLab project on the left, and you can find the current draft merge request.



___**19.** 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.

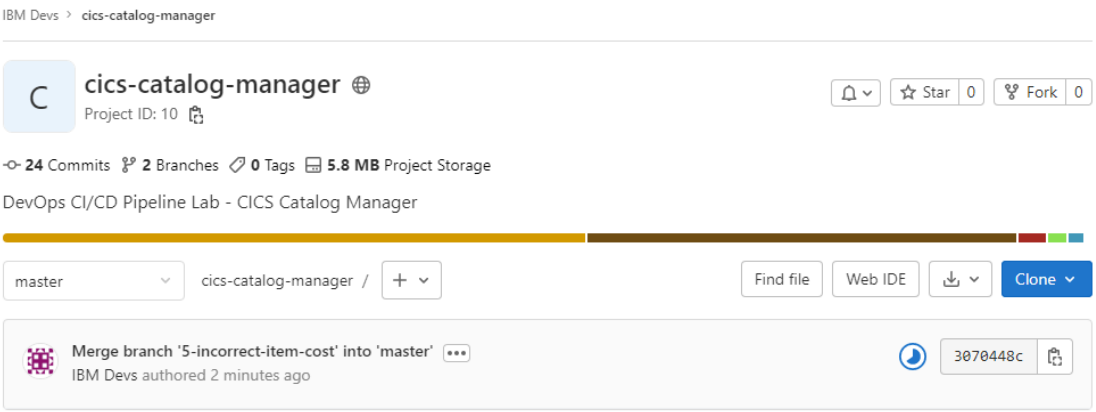


___**20.** Click **Merge** button to make it official.



___**21.** 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.

___**22.** 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. C cics-catalog-manager 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