



# Zowe CICS Workshop

*Automate Mainframe Apps with Jenkins and Zowe*



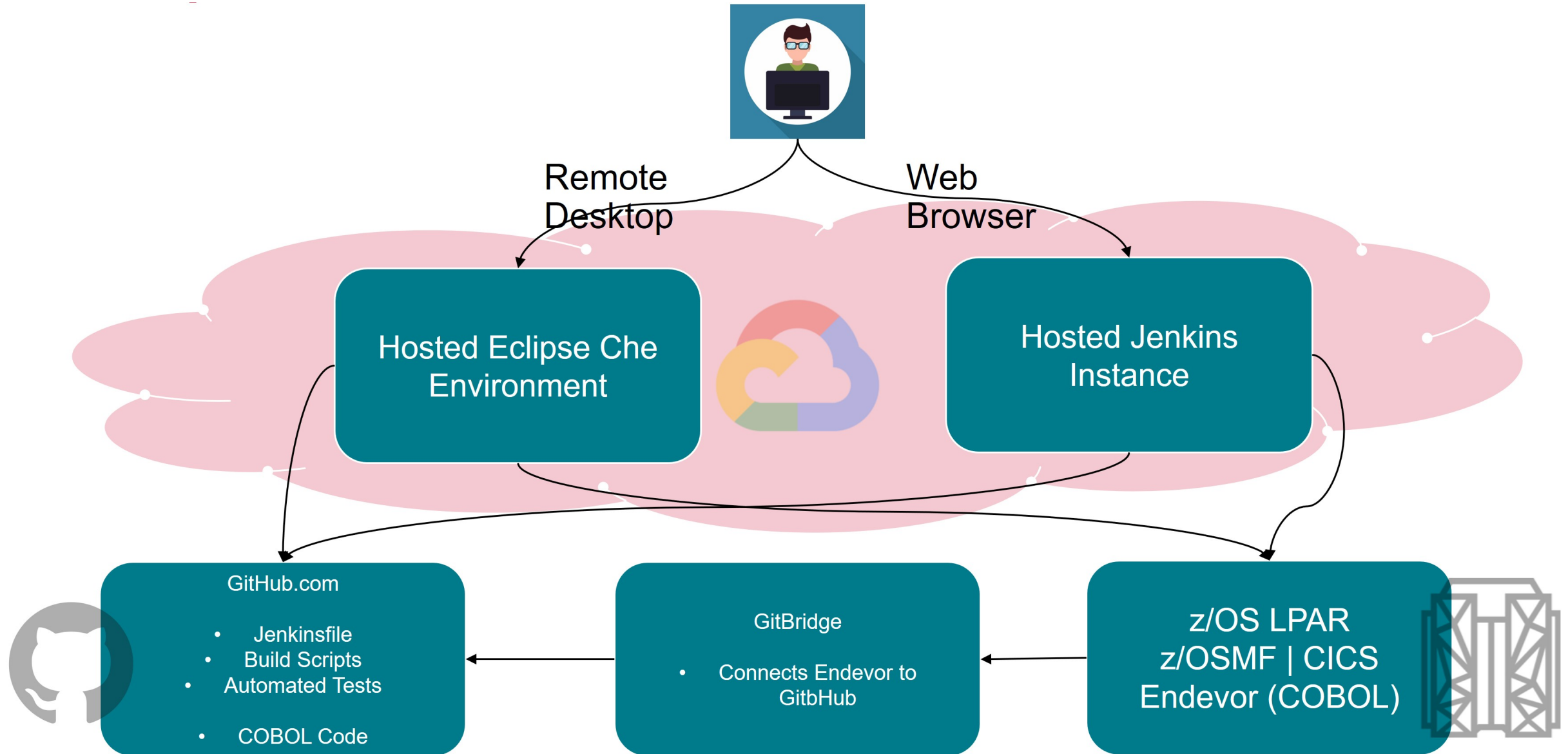
# Goals

- Write automation scripts for mainframe app to build and deploy
- Write automated tests for mainframe apps using open testing frameworks
- Build a Jenkins CI/CD pipeline for mainframe app



# Accessing your Workshop Environment

# Workshop Environment



# z/OS Services

- You have been assigned a single set of login credentials for accessing all of the Mainframe resources on a remote z/OS LPAR which is hosted by Broadcom, including TSO, z/OSMF,CICS, Db2 and CA Endevor SCM.
- Your userid is CUST001.
- Your password CUST001.

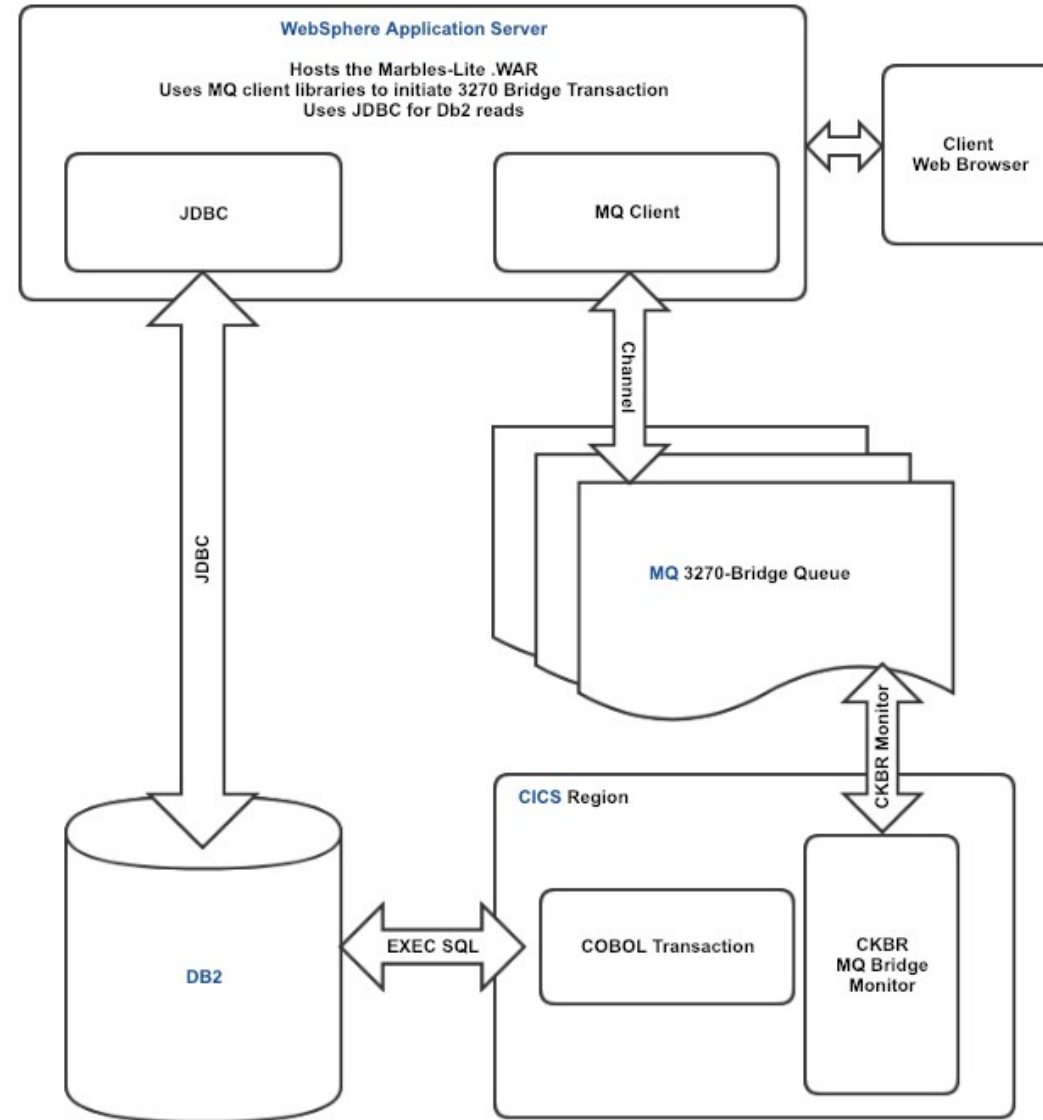
Service	Connection Information (Host:Port)
z/OSMF	104.208.142.8:1443
CICS	104.208.142.8:6000
CA Endevor	104.208.142.8:6032
DB2	104.208.142.8:6017



# Marbles



# Marbles App





Demo



# State of Marbles App - Before Workshop

## Marbles - before workshop exercise

Database - State of table

Color	Quantity	Cost
Silver	10	5
Gold	1	100

CICS Transaction

Name	Function
Create	Create a new color of marble
Delete	Delete an existing color of marble
Update	Update <b>only the Quantity</b> of an existing color of marble

GUI - State of GUI

Color	Quantity	Cost
Silver	10 (button to update - <b>succeeds</b> on click)	5 (button to update - <b>fails</b> on click)
Gold	1 (button to update - <b>succeeds</b> on click)	100 (button to update - <b>fails</b> on click)
(button to add color - succeeds on click)		


# Desired State of Marbles App - After Workshop

## Marbles - after workshop exercise

Database - State of Table

Color	Quantity	Cost
Silver	10	5
Gold	1	100
Color X	x	x

CICS Transaction

Name	Function
Create	Create a new color of marble 
Delete	Delete an existing color of marble
Update	Update the <u>Quantity and Cost</u> of an existing color of marble

GUI - State of GUI

Color	Quantity	Cost
Silver	10 (button to update - succeeds on click)	5 (button to update - <b>succeeds</b> on click)
Gold	1 (button to update - succeeds on click)	100 (button to update - <b>succeeds</b> on click)
Color X	X (button to update - succeeds on click)	X (button to update - <b>succeeds</b> on click)
(button to add color - succeeds on click)		



# Sample CI Pipeline

# Simple Pipeline Demo

- CI/CD orchestrator
- Process automation scripts to perform mainframe actions
- Scripts call Zowe CLI commands



# Section I:

## *Overview and Environment Setup*



# Steps for Section I

- Currently the CICS transaction is able to update the quantity of a marble. We want to enhance this transaction to be able to update the cost of a marble in addition to its existing functionality.
- Steps:
  - Download the COBOL transaction code to your remote desktop from Endeavor
  - Edit the code in Eclipse Che
  - Upload the code to Endeavor
  - Build (generate) the code on Endeavor
- Note: all the following steps should be performed from your assigned remote Eclipse Che environment.

# Developer Environment

- Open your remote desktop application
  - Che Desktop URL: `http://mfwsone.broadcom.com/dashboard/`
  - Login is `workshop_001`
  - Password is `001_workshop`
- Open command prompt
  - Issue `"zowe --h"`
  - Issue `"zowe plugins list"`
  - Issue `"npm -h"`
  - Issue `"git help"`

# Profiles

- Created to access different environments on the mainframe
  - Can connect to multiple LPARS with different usernames and passwords and ports
  - Multiple profiles can be created for an application
- Zowe V2 uses `zowe.config.json` to set up the profiles.
- Using the Zowe command line the first time, you will be prompted to enter credentials.
  - `cd /projects/Zowe-DevOps-A-01`
  - `zowe files list ds CUST001.*`
  - You will be prompted for the
    - IP Address: 104.208.142.8
    - Username: CUST001
    - Password: CUST001
  - If you make a mistake, you can run the command again, or issue `zowe config init`
  - Be sure you run a `zowe files list ds` command before accessing Gulp



## Section II:

*Modify Cobol Code*

# Download the element from Endeavor - Step 1

- Now that we've identified the element we know we need to change, let's download to our remote desktop using another command.
- **Note:** The command below uses options provided in the command which take precedence over your default profile from Step 1. This includes the Endeavor system, subsystem, etc.
- Position your terminal to the folder where you want the file downloaded to.
- Download element:
  - `zowe endeavor retrieve element MARBLE01 --type COBOL --to-file MARBLE01.cbl --override-signout`



# Edit the source code - Step 2

- We are now ready to make the code updates to implement the ability to change the cost of a marble using a CICS transaction.
- Open your source code using the explorer view (Upper Left Corner)
- Find the following code sequence and remove the highlighted text and Save the changes locally:

```
233.* =====
234.* Parse the transaction input
235.* =====
236. PARSE-CICS-INPUT.
237.         UNSTRING WS-CICS-INPUT DELIMITED BY SPACE
238.             INTO WS-INPUT-TRAN-ID,
239.                 WS-INPUT-VERB,
240.                 WS-INPUT-COLOR,
241.                 WS-INPUT-INV,
242.*<-- remove WS-INPUT-COST,
243.                 WS-INPUT-TRAILER
244.         END-UNSTRING.
```

# Upload the element to Endeavor - Step 3

- After making the code changes locally, we need to upload the element to Endeavor in order to perform a build.
- Ensure your terminal is positioned to the folder that contains the source file.
- Upload element:
  - `zowe endeavor update element MARBLE01 --type COBOL --os --ff MARBLE01.cbl`
  - This command provides help on how to upload an element to Endeavor:
    - `zowe endeavor update element -h`



**Generate the code**

# Generate the elements

- Now that our code changes have been uploaded to Endeavor, we can compile it using an Endeavor generate action to see if there are any errors.
- Generate element.
  - There are two `--type` of elements you want to generate, COBOL and LNK.
  - `zowe endeavor generate element MARBLE01 --type COBOL --os`
  - `zowe endeavor generate element MARBLE01 --type LNK --os`
  - **HINT:** Use your up arrow key and just change the type.
- Ensure that the generate actions are successful:
  - You should see text similar to `GENERATE of MARBLE01.COBOL finished with 0000` in the output.



## Section III:

*Deploy Marbles Application Manually*



# Deployment - Introduction

- Deployment is another step that is commonly automated. Once you've built your code and binary artifacts like load modules are ready, you may want to copy these artifacts to a system where you can run the program.
- Identify deployment steps
- Identify requirements for parametrization in deployment
  - Do the build artifacts come from a different location depending on whether it's a dev build or a team build?
  - Does the deployment system vary depending on the stage?
- Automate the deployment
- **Note:** Deployment scripts should be written in a parameterized fashion so that the same script can be used to deploy for devtest, QA, system-test or even production.

# Steps for Section III

- Deployment for marbles requires us to copy the load modules, and activate the changes in the target CICS environment.
  - Deploy manually using CLI commands
  - Create and implement a Deploy gulp task
  - Test the deployment

# Deploy manually

- When we generated the LNK element in Endevor in previous sections, Endevor created load modules. We can deploy these load modules to the proper dataset location that CICS is using and refresh CICS to pick up the changes.
- Confirm that the load modules exists in the dataset
  - We can now proceed to list the members in our LOADLIB and DBRMLIB to ensure our MARBLE entry exists.
    - `zowe files list all-members "PRODUCT.NDVR.MARBLES.MARBLES.D1.LOADLIB"`
    - `zowe files list am "PRODUCT.NDVR.MARBLES.MARBLES.D1.DBRMLIB"`

# Deploy manually

- Copy the LOADLIB and DBRMLIB modules to the desired location
  - There are multiple ways to copy load modules using Zowe. For this workshop, we are going to make use of a job to move the elements.
  - Now, let's try to copy our MARBLE element from the source libraries to the destination:
    - `zowe jobs submit data-set "CUST001.MARBLES.JCL(MARBCOPY)" --vasc`

# Deploy manually

- Submit JCL to perform the Bind & Grant
  - `zowe jobs submit data-set "CUST001.MARBLES.JCL(MARBIND) " --view-all-spool-content`
    - This function runs the command and returns all the job content
  - Alternative approach
    - `zowe jobs submit data-set "CUST001.MARBLES.JCL(MARBIND) "`
      - Example returned jobid: JOBXXXXXX
    - `zowe jobs view job-status-by-jobid JOBXXXXXX`
      - Confirm return code = CC 0004



# Deploy manually

- Activate the transaction changes on CICS
  - We will make use of the CICS plugin to refresh our CICS program.
  - Now, let's try to refresh our CICS program
    - The program that needs refreshed is named Marble.
    - `zowe cics refresh program MARBLE01`

# Test manually

- Run the command manually:
  - `zowe console issue command "F CICSTRN1,MB01 CRE RED 1 2" --console-name CUST001`
    - Did you get a **+SUCCESS** message?
  - Check the database
    - `zowe jobs submit ds "CUST001.MARBLES.JCL(MARBDB2)" --vasc`
      - Ensure database contains your marble with quantity and cost



**Review - What have we learned?**



## Section III:

*Automation Automate the Build*



# Automate the Code Build - Introduction

- Now that you've used the CLI to successfully make code changes and perform generates on Endeavor, it's time to automate these steps.
- CLI commands can be embedded in scripts that you can run repeatedly from your local machine.
- These same scripts can also be called from CI/CD tools like Jenkins
- Task Runners are a way to organize and interact with your automation scripts more easily.
- We will be using a javascript based Task Runner called Gulp for this section. Task Runners can also be called from CI/CD tools like Jenkins
- **Note:** Task Runners are an abstraction layer over scripts. They are helpful but are not a necessity.

# Steps for Section III

- Now that you've used the CLI to successfully make code changes and perform generates on Endeavor, we want to automate the following:
  - Generate operations for COBOL and LNK
  - Build the application
    - It should generate both the COBOL and LNK elements in a single task
- **Note:** Gulp is a JavaScript based task runner. Other task runners like Gradle use other scripting languages like Groovy.

# Starting the Automation

- We have performed some functions for you, like downloading the code and getting it ready.
  - Your code is located in /projects.
  - When you ran the gulp command earlier, you used that code.

# Create a Build Task in Gulp

- Generating the source element and the LNK element on Endeavor are steps that you'll need to perform every time after making code changes to create the load module. It's a great task to automate, so that you don't have to keep doing it manually. Let's start by reviewing our gulpfile and existing gulp `build-cobol` task. Then you will create a task in Gulp called `build-lnk`.
- Review gulpfile: A gulpfile is a file in your project directory titled `gulpfile.js` that automatically loads when you run the `gulp` command.
- At the top of the gulpfile, take note of three packages that we are using:
  - `gulp-help`: Adds a default help task to gulp and provides the ability to add custom help messages for gulp tasks. Try issuing `gulp help` in the terminal at your projects directory.
  - `gulp-sequence`: Allows a series of gulp tasks to be run in order
  - `node-cmd`: Simple terminal interface that allows cli commands to be run. These commands are run asynchronously.
  - `config`: requires a `config.json` file for all the options



# Reusable Code - config.json

- Using a configuration file allows the script to remain the same, but passing in variables for the differences.
- Here's a file call config.json containing the values
- Instead of hardcoding the values in the script, the script can read these values.
- To use these values, we can use `config.testElement` and it will read the color from this file and replace it in the code.

```
{
  "bindGrantJCL": "CUST001.MARBLES.JCL(MARBIND)",
  "cicsConsole" : "CUST001",
  "cicsProgram"  : "MARBLE01",
  "cicsRegion"   : "CICSTRN1",
  "cicsTran"     : "MB01",
  "db2QueryJCL"  : "CUST001.MARBLES.JCL(MARBDB2)",
  "devDBRMLIB"   : "PRODUCT.NDVR.MARBLES.MARBLES.D1.DBRMLIB",
  "devLOADLIB"   : "PRODUCT.NDVR.MARBLES.MARBLES.D1.LOADLIB",
  "marbleColor"  : "RED",
  "testDBRMLIB"  : "BRIGHT.MARBLES.DBRMLIB",
  "testElement"  : "MARBLE01",
  "testLOADLIB"  : "CICS.TRAIN.MARBLES.LOADLIB"
}
```

# Create a Build task in Gulp

```
gulp.task('build-cobol', 'Build COBOL element', function (callback) {  
  var command = `zowe endeavor generate element ${config.testElement} --type COBOL --override-signout --maxrc 0 --stage-number 1`;   
  
  simpleCommand(command, "command-archive/build-cobol", callback);  
});
```

- Review build-cobol task
  - Name of task: build-cobol
  - Description of task: Build COBOL element
  - function (callback): function that this gulp task runs. Because node-cmd runs terminal commands asynchronously, we supply a callback which is called upon task completion.
  - var command = ... : command to run from the command line, passing config.json values
  - simpleCommand runs the command with all the error checking in another section. This is reused a lot, so a function was created to make the code cleaner. The second option includes archiving the output to a directory.

# Create a Build task in Gulp

```
/**
 * Runs command and calls back without error if successful
 * @param {string}    command    command to run
 * @param {string}    dir        directory to log output to
 * @param {awaitJobCallback} callback function to call after completion
 * @param {Array}      [expectedOutputs] array of expected strings to be in the output
 */
function simpleCommand(command, dir, callback, expectedOutputs){
  cmd.get(command, function(err, data, stderr) {
    //log output
    var content = "Error:\n" + err + "\n" + "StdErr:\n" + stderr + "\n" + "Data:\n" + data;
    writeFile(dir, content);

    if(err){
      callback(err);
    } else if (stderr){
      callback(new Error("\nCommand:\n" + command + "\n" + stderr + "Stack Trace:"));
    } else if (typeof expectedOutputs !== 'undefined'){
      verifyOutput(data, expectedOutputs, callback);
    } else {
      callback();
    }
  });
}
```

- Review simpleCommand task
  - Name of task: simpleCommand
  - Description of task: Runs the zowe commmands
  - cmd.get ( : Runs the given command and checks to ensure it ran properly. If an error occurs, the code will immediately exit
  - It writes the output to the directory specified
  - If it is successful, the final callback will allow the code to continue executing.

# Create a Build task in Gulp

- Run `gulp build-cobol` and verify it completes successfully.
- Create a `build-lnk` gulp task using the `build-cobol` gulp task as a reference.
- Ensure the `build-lnk` task and description appear when you issue `gulp help`
- Ensure the `build-lnk` task completes without error when you issue:
  - `gulp build-lnk`

# Combine Build-Cobol and Build-LNK

- Let's take the `gulp build-cobol` and `gulp build-lnk` tasks and combine them into a single gulp task. The `gulp-sequence` package can help us achieve this.
- The following gulp task combines the existing build tasks into a single gulp build task:
  - `gulp.task('build', 'Build Program', gulpSequence('build-cobol', 'build-lnk'));`
- Ensure the build task and description appear when you issue:
  - `gulp help`
- Ensure the build task runs both the `build-cobol` and `build-lnk` tasks without error when you issue:
  - `gulp build`

# Completed Build Sequence Command

- Completed

```
gulp.task('build', 'Build Program', gulpSequence('build-cobol', 'build-lnk'));
```



## Section IV:

*Automate Deployment*



# Create and implement a Gulp Deploy task

- Similar to creating the gulp build tasks, we will now create gulp tasks to deploy our changes.
  - Review copy task
  - Review the `bind-n-grant` task
  - Review the `cics-refresh` task
  - Combine individual deploy tasks into one `deploy` task.

# Create and implement a Gulp Deploy task

```
gulp.task('copy', 'Copy LOADLIB & DBRMLIB to test environment', function (callback) {  
  var ds = config.copyJCL;  
  submitJobAndDownloadOutput(ds, "job-archive/copy", 4, callback);  
});
```

- Review copy task to copy the LOADLIB and DBRMLIB
  - This task uses a job to copy the lib elements to their destination

# Create and implement a Gulp Deploy task

```
gulp.task('bind-n-grant', 'Bind & Grant Job', function (callback) {  
  var ds = config.bindGrantJCL;  
  submitJobAndDownloadOutput(ds, "job-archive/bind-n-grant", 4, callback);  
});
```

- Review bind-n-grant task, which submits a job from a dataset.
  - The bind-n-grant task to submit MARBIND JCL and verify CC = 0004

# Create and implement a Gulp Deploy task

```
/**
 * Submits job, verifies successful completion, stores output
 * @param {string}      ds          data-set to submit
 * @param {string}      [dir="job-archive"] local directory to download spool to
 * @param {number}      [maxRC=0]    maximum allowable return code
 * @param {awaitJobCallback} callback function to call after completion
 */
function submitJobAndDownloadOutput(ds, dir="job-archive", maxRC=0, callback){
  var command = 'zowe jobs submit data-set "' + ds + '" -d ' + dir + " --rfj";
  cmd.get(command, function(err, data, stderr) {
    //log output
    var content = "Error:\n" + err + "\n" + "StdErr:\n" + stderr + "\n" + "Data:\n" + data;
    writeFile("command-archive/job-submission", content);

    if(err){
      callback(err);
    } else if (stderr){
      callback(new Error("\nCommand:\n" + command + "\n" + stderr + "Stack Trace:"));
    } else {
      data = JSON.parse(data).data;
      retcode = data.retcode;

      //retcode should be in the form CC nnnn where nnnn is the return code
      if (retcode.split(" ")[1] <= maxRC) {
        callback(null);
      } else {
        callback(new Error("Job did not complete successfully. Additional diagnostics:" + JSON.stringify(data,null,1)));
      }
    }
  });
}
```

- submitJobAndDownloadOutput submits the dataset
  - It checks for errors
  - It executes the job asynchronously
  - It runs a loop to ensure it completes
  - If there's an error, that is returned to the calling task
  - Captures output and writes it to disk

# Create and implement a Gulp Deploy task

- Review `cics-refresh` task to refresh the MARBLE PGM
- Combine the tasks into a single deploy task using `gulpSequence`.
  - Using the code you created to combine the `build-cobol` and `build-lnk` tasks into a single build task as a reference, combine the following tasks into a single deploy task that will deploy the program using:
    - `copy`
    - `bind-n-grant`
    - `cics-refresh`
  - Ensure your task appears when issuing `gulp help`
  - Run `gulp deploy`
  - Ensure your task completes without error

# Completed Deploy Sequence Command

- Completed

```
gulp.task('deploy', 'Deploy Program', gulpSequence('copy', 'bind-n-grant', 'cics-refresh'));
```



# Review

*What have we learned so far?*





# Section V:

*Automate Testing*

# What is automated testing?

- Applications behave in an expected manner. When making code changes, it's common for developers to test the expected behavior many times. This can be time consuming and boring. Developers started to write scripts that automate these repetitive tests so that they can stay focused on writing more code.
- Different types of automated testing
  - Unit Tests
  - Integration Tests
  - System Tests
  - Performance Tests

# Types of Automated Tests

- Unit Tests
  - Front-end code (JavaScript)
  - Web code (Java)
  - Back-end code (COBOL / CICS)
- Integration Tests
  - UI
  - Web Server
  - CICS
  - Db2
- System Tests
- Performance Tests

# Automating Tests

- Automating tests comes down to two choices:
  - Write scripts in your language of choice and manage them manually.
  - Choose a scripting framework that suits your needs and skills.
- Popular testing frameworks:
  - Mocha
  - Robot
  - Jest
  - Jasmine
  - JMeter



# Automate Testing of a CICS Transaction

# CICS Manual Test

- Manually test your deployed program to ensure the MABRLE was successfully created.
  - Recall that you previously issued the following command to ensure your program was successfully updated and deployed:
    - `zowe console issue command "F CICSTRN1,MB01 CRE RED 1 2" --console-name CUST001`
  - Now we will run a command to verify the database contains the correct information. Ensure your marble is in the database:
    - `zowe jobs submit ds "CUST001.MARBLES.JCL(MARBDB2)" --vasc`

# CICS Test Scenario

- A framework has been provided that tests that the quantity of marbles in inventory is manipulated appropriately. We will be updating the test plan to account for the cost being updated appropriately as well. The current test plan is as follows:

```
/**
 * Test Plan
 * Delete the marble to reset inventory to zero (Delete will be tested later)
 *
 * Create a marble
 * Verify that there is one marble in the inventory
 *
 * Create the marble entry "again"
 * Verify the appropriate error message is returned
 *
 * Update marble quantity to 2
 * Verify that there are two marbles in the inventory
 *
 * Delete the marble from the database
 * Verify there are no marbles in the inventory
 *
 * Delete the marble "again"
 * Verify appropriate error message is returned
 *
 * Update marble (which doesn't exist)
 * Verify appropriate error message is returned
 */
```



# MochaJS



simple, flexible, fun

Mocha is a feature-rich JavaScript test framework running on [Node.js](#) and in the browser, making asynchronous testing *simple* and *fun*. Mocha tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases. Hosted on [GitHub](#).

gitter join chat backers 90 sponsors 39

# Review Mocha Test File

- All tests for this workshop are located in test/test.js.
- In test.js find `describe('Inventory Manipulation, function () {`
  - This is our initial suite of inventory manipulation tests.
- The following snippet simply ensures the marble inventory for our color is reset to zero before we begin testing.

```
// Delete the marble to reset inventory to zero (Delete will be tested later)
before(function(done){
  deleteMarble(COLOR, function(){
    done();
  })
});
```

# Review Mocha Test File

- The following snippet shows the first test. Within the test, a marble is created and the contents of the database are verified, and the tests asserts that 1 marble of the specified color is in the inventory.

```
it.only('should create a single marble', function (done) {  
  // Create marble  
  createMarble(COLOR, 1, function(err, data, stderr){  
    if(err){  
      throw err;  
    } else if (stderr){  
      throw new Error("\nError: " + stderr);  
    } else {  
      // Strip unwanted whitespace/newline  
      data = data.trim();  
      assert.equal(data, "+SUCCESS", "Unsuccessful marble creation");  
  
      getMarbleQuantity(COLOR, function(err, quantity){  
        if(err){  
          throw err;  
        }  
        assert.equal(quantity, 1, "Inventory is not as expected");  
        done();  
      });  
    }  
  });  
});
```

# Review Mocha Test File

- The createMarble function issues a `zowe console` command to execute the CICS transaction to create a marble of a specified color with an initial specified inventory.
  - If the initial inventory is not specified, the function defaults it to 1.
  - Once the asynchronous command completes, it will call an optional callback.

```
/**
 * Creates a Marble with an initial quantity
 * @param {string}      color      color of Marble to create
 * @param {number}      [quantity=1] quantity of Marbles to initially create
 * @param {nodeCmdCallback} [callback] function to call after completion, callback(err, data, stderr)
 */
function createMarble(color, quantity=1, callback) {
  cmd.get(
    `zowe console issue command "F ${config.cicsRegion},${config.cicsTran} CRE ${color} ${quantity}" --cn ${config.cicsConsole}`,
    function (err, data, stderr) {
      //log output
      var content = "Error:\n" + err + "\n" + "StdErr:\n" + stderr + "\n" + "Data:\n" + data;
      writeFile("command-archive/create-marble", content);

      typeof callback === 'function' && callback(err, data, stderr);
    }
  );
}
```

# Review Mocha Test File

- The `getMarbleQuantity` function submits a job to query the Marbles table using SQL.
- It waits for the job to complete then retrieves the spool file, 104.
- It parses the spool output using regular expressions to find the row with the data.
  - Rows have a specific structure where the columns are separated with a "|"
- The data is assigned and the information sent through a callback to the calling function.

```
89  */
90  function getMarbleQuantity(color, callback) {
91    // Submit job, await completion
92    cmd.get(
93      'zowe jobs submit data-set "${config.db2QueryJCL}" --rff jobId --rft string',
94      function (err, data, stderr) {
95        if (err) {
96          throw err
97        } else {
98          // Strip unwanted whitespace/newline
99          var jobId = data.trim();
100
101          // Await the jobs completion
102          awaitJobCompletion(jobId, function (err) {
103            if (err) {
104              throw err
105            } else {
106              cmd.get(
107                'zowe jobs view sfbi ${jobId} 104',
108                function (err, data, stderr) {
109                  if (err) {
110                    callback(err);
111                  } else {
112                    var pattern = new RegExp(".*\\| " + color + " .*\\|.*\\|.*\\|", "g");
113                    var found = data.match(pattern);
114                    if (!found) {
115                      callback(err, null, null);
116                    } else { //found
117                      //found should look like nn_ COLOR | QUANTITY | COST |
118                      var row = found[0].split("|");
119                      quantity = Number(row[2])
120
121                      callback(err, quantity);
122                    }
123                  }
124                });
125            }
126          });
127        }
128      });
129    }
130  },
131 }
```

# Review Mocha Test File

- You could run the existing mocha tests and produce a report by issuing `mocha --reporter mochawesome`.
- However, if you look in your `package.json` at your project's root, you will see a `scripts` section.
- This demonstrates how to set up a test script. This test script can then be run with the `npm test` from your terminal at your project's root. Run:
  - `npm test`
- Run the tests and verify they all pass.

```
"scripts": {  
  "test": "mocha --reporter mochawesome"  
},
```

# Implement New Tests

- Now that we have ensured all the existing tests for marble inventory manipulation pass, let's adjust the test case for creating a single marble so that it creates a single marble with a cost of 1.
- First, locate the following test and update the title from 'should create a single marble' to 'should create a single marble with a cost of 1'
- Remove the `.only()` so it will run all the tests.
- Change the `createMarble` function call to `createMarble(COLOR, 1, 1, ...`
- Also, adjust the `createMarble` function call in the test case titled: should not create a marble of a color that already exists
- Next, we will update the `createMarble` function to accept a cost parameter.
- The function is displayed on the next slide.

# Implement New Tests

```
it('should create a single marble', function (done) {  
  // Create marble  
  createMarble(COLOR, 1, function(err, data, stderr){  
    if(err){  
      throw err;  
    } else if (stderr){  
      throw new Error("\nError: " + stderr);  
    } else {  
      // Strip unwanted whitespace/newline  
      data = data.trim();  
      assert.equal(data, "+SUCCESS", "Unsuccessful marble creation");  
  
      getMarbleQuantity(COLOR, function(err, quantity){  
        if(err){  
          throw err;  
        }  
        assert.equal(quantity, 1, "Inventory is not as expected");  
        done();  
      });  
    }  
  });  
});
```



# Change createMarble Function

```
/**
 * Creates a Marble with an initial quantity
 * @param {string}      color      color of Marble to create
 * @param {number}      [quantity=1] quantity of Marbles to initially create
 * @param {nodeCmdCallback} [callback] function to call after completion, callback(err, data, stderr)
 */
function createMarble(color, quantity=1, callback) {
  cmd.get(
    `zowe console issue command "F ${config.cicsRegion},${config.cicsTran} CRE ${color} ${quantity}" --cn ${config.cicsConsole}`,
    function (err, data, stderr) {
      //log output
      var content = "Error:\n" + err + "\n" + "StdErr:\n" + stderr + "\n" + "Data:\n" + data;
      writeFile("command-archive/create-marble", content);

      typeof callback === 'function' && callback(err, data, stderr);
    }
  );
}
```

- Locate the `createMarble()` function:
- Add another number parameter after quantity called `cost` with a default value of 1. (Use the quantity parameter as a reference).
- Adjust the `zowe` command being issued to include another space and `cost`. Next, we will update our function that retrieves the quantity of marbles of a specific color to also retrieve the cost.

# Change getMarbleQuantity function

```
function getMarbleQuantity(color, callback) {
  // Submit job, await completion
  cmd.get(
    'zowe jobs submit data-set "${config.db2QueryJCL}" --rff jobid --rft string',
    function (err, data, stderr) {
      if(err){
        throw err
      } else {
        // Strip unwanted whitespace/newline
        var jobId = data.trim();

        // Await the jobs completion
        awaitJobCompletion(jobId, function(err){
          if(err){
            throw err
          } else {
            cmd.get(
              'zowe jobs view sfbi ${jobId} 184',
              function (err, data, stderr) {
                if(err){
                  callback(err);
                } else {
                  var pattern = new RegExp(".*\\| " + color + " .*\\|.*\\|.*\\|", "g");
                  var found = data.match(pattern);
                  if(!found){
                    callback(err, null, null);
                  } else { //found
                    //found should look like nn_ COLOR | QUANTITY | COST |
                    var row = found[0].split("|");
                    quantity = Number(row[2]),
                    cost = Number(row[3]);

                    callback(err, quantity, cost);
                  }
                }
              }
            );
          }
        });
      }
    }
  );
}
```

- Locate the `getMarbleQuantity` function.
- In the row section, let's collect the cost and send it through the callback.
- Add `cost = Number(row[3])`; and adjust the callback.

# Update the test

- Relocate the test we are updating. Finally, add the cost parameter to your `getMarbleQuantity` callback function and assert that cost is equal to 1. A finished updated test is shown on the next slide.

```
it('should create a single marble with cost of 1', function (done) {  
  // Create marble  
  createMarble(COLOR, 1, 1, function(err, data, stderr){  
    if(err){  
      throw err;  
    } else if (stderr){  
      throw new Error("\nError: " + stderr);  
    } else {  
      // Strip unwanted whitespace/newline  
      data = data.trim();  
      assert.equal(data, "+SUCCESS", "Unsuccessful marble creation");  
  
      getMarbleQuantity(COLOR, function(err, quantity, cost){  
        if(err){  
          throw err;  
        }  
        assert.equal(quantity, 1, "Inventory is not as expected");  
        assert.equal(cost, 1, "Cost is not as expected");  
        done();  
      });  
    }  
  });  
});
```

# Run the test

- Run `npm test` verifies the tests all pass.
- Work with the facilitator should any issues arise.
- A report of your test has been generated. Open the following file from your project's root to view:
  - `mochawesome-report/mochawesome.html`



# Review

*What have we learned?*

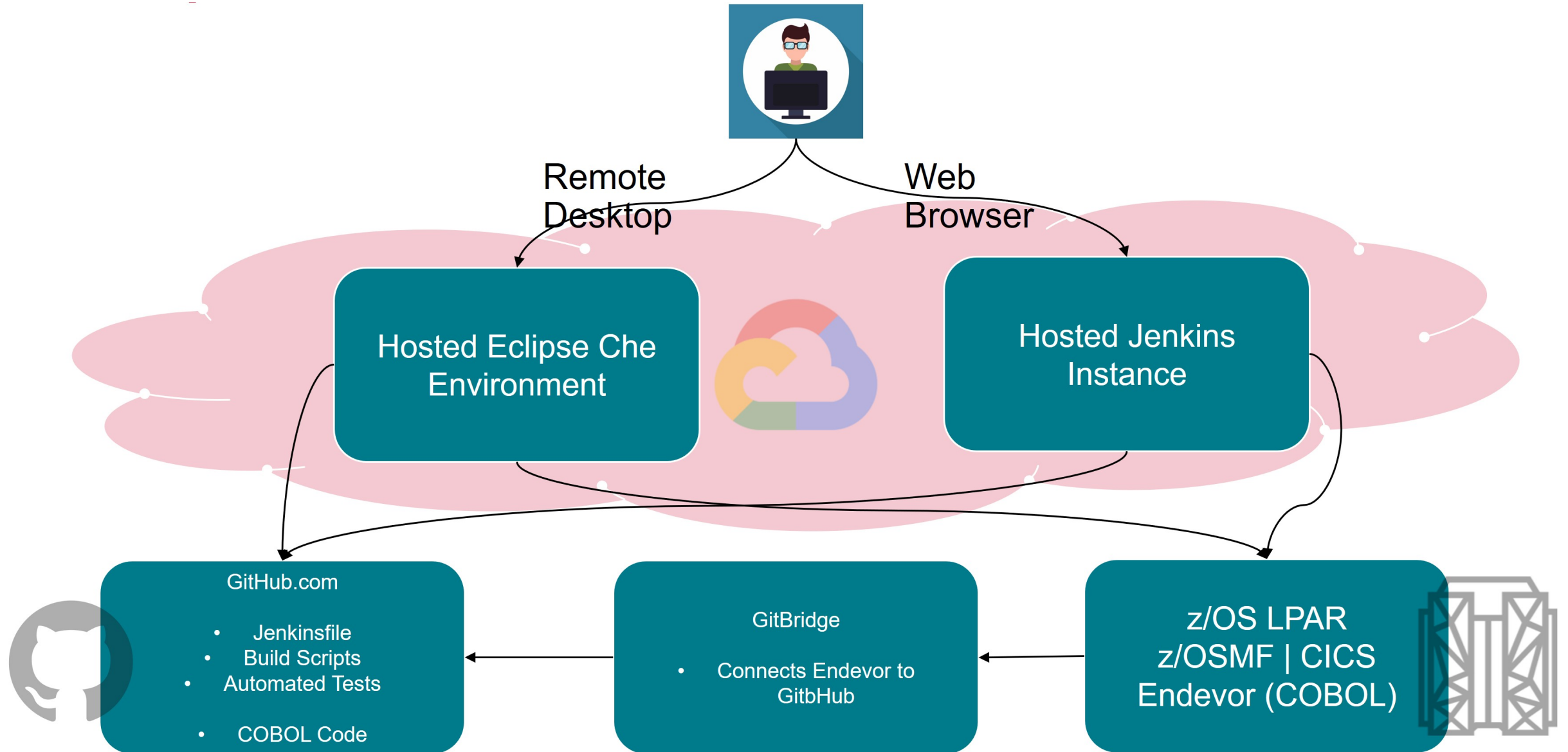


## Section VI:

*Add Code Build to Continuous Integration*



# Workshop Environment



# Continuous Integration - Introduction

- Task runners help individual developers run their automation scripts easily and avoid wasting time doing mundane tasks. We can take this one step further and allow many of these tasks to be performed by CI orchestrators when code changes occur in a shared team repository.
  - Identify tasks that need to be performed after code changes are made at a shared level.
  - Create stages in CI orchestrators for these tasks
  - Call automated scripts from these stages
- **Note:** In addition to automating these steps from CI, you will likely want to automate the trigger of the CI process from the shared code repository when code changes occur.



# Steps for Section VI

- In our case, the build step is a common one that could be performed from CI once a code change is made in Endevor. Let's add it to a Jenkins pipeline.
- Log in to Jenkins, view `Workshop_001` project, and verify pipeline runs.
- Review Jenkinsfile
- Enhance Jenkinsfile to build project
  - Enhance build stage in Jenkinsfile to run `gulp build, gulp deploy and npm test`
  - Add Environment variables to supply connection and project details.
- Manually kick off the pipeline to test

# Jenkinsfile Overview Terms

- Pipeline - defines the overall pipeline structure
- Agent - defines where the code will execute
- Environment - defines the environment variables for use in other commands
- Stages - defines the start of the stages
- Stage - defines each stage
- Post - performed after running the stages section

# Jenkinsfile

- To ease troubleshooting and ensure everything is set up, local setup does the following:
  - Verifies version of node, npm, zowe, zowe plugins
  - Installs gulp-cli
  - Installs npm dependencies
- This ensure all the components are available.
- It then creates dummy profiles, so the code is cleaner. These profiles are destroyed when the run finishes. This provides cleaner code, but still has security as usernames, passwords and hostnames are obfuscated.

```
stage('local setup') {
  steps {
    sh 'node --version'
    sh 'npm --version'
    sh 'zowe --version'
    sh 'zowe plugins list'
    sh 'npm install gulp-cli -g'
    sh 'npm install'

    //Create cics, db2, endeavor, fmp, and zosmf profiles, env vars will provide host, user, and password details
    sh 'zowe profiles create cics Jenkins --port 6000 --region-name CICSTRN1 --host dummy --user dummy --password dummy'
    sh 'zowe profiles create db2 Jenkins --port 6017 --database D10CPTIB --host dummy --user dummy --password dummy'
    sh 'zowe profiles create endeavor Jenkins --port 6002 --protocol http --ru false --host dummy --user dummy --password dummy'
    sh 'zowe profiles create endeavor-location Marbles --instance ENDEVOR --env DEV --sys MARBLES --sub MARBLES --ccid JENKXX --comment JENKXX'
    sh 'zowe profiles create fmp Jenkins --port 6001 --protocol https --ru false --host dummy --user dummy --password dummy'
    sh 'zowe profiles create zosmf Jenkins --port 443 --ru false --host dummy --user dummy --password dummy'
  }
}
```

# Log in to Jenkins

- Jenkins is a hosted application running on a web server. You can access it from most web browsers.
- Log in to Jenkins: `http://mfwsone.broadcom.com/jenkins/`
  - Username: `workshop_001`
  - Password: `001_workshop`
- Verify that the environment is in the right starting state
  - Click on the name of your project (`Workshop_001`)
  - Select the `master` branch
  - Click Build Now in the left side menu and verify the project builds successfully.
    - Build Now icon is shown below.



# Review Jenkinsfile

- A `Jenkinsfile` is a text file that contains the definition of a Jenkins Pipeline and is checked into source control. Using a `Jenkinsfile`, which is checked into source control, enables
  - Code review/iteration on the pipeline
  - Audit trail for the pipeline
  - Single source of truth for the pipeline
- More detailed information is available at <https://jenkins.io/doc/book/pipeline/jenkinsfile/>

# Review Jenkinsfile

- Reviewing the `Jenkinsfile` in our project's root directory, the agent directive instructs Jenkins to allocate an executor and workspace for the pipeline. In our workshop, we use the label `zowe-agent` to instruct Jenkins to execute the tasks in a docker container that contains Zowe Community Edition and is being pulled from Docker Hub.
- Environment variables can be declared within an environment directive or with a `withEnv` step.
  - An environment directive in the top-level pipeline block applies to all steps within the pipeline
  - An environment directive within a stage will only apply those variables to that stage.
- The stages directive contains various pipeline stages and the steps directive provides the tasks for each stage.

# Updating the Jenkinsfile

- Implement the Build stage by calling the Build gulp task.
  - Locate the build stage in your `Jenkinsfile`
  - Remove the following line of code which simply echoed out a statement:
    - `sh "echo build"`
- Uncomment the ensuing `withCredentials` code block.
  - Inside this block, we will have access to `eosCreds`. It is defined in Jenkins.
  - We define the user environment variable to be `ZOWE_OPT_USER` and the password to be `ZOWE_OPT_PASSWORD` because Zowe can be influenced by environment variables.
  - Let's take a moment to discuss command line precedence.

# Zowe Command Line Precedence

- You can specify any option on any command through the use of environment variables using the prefix `ZOWE_OPT_`.
- For example, you can specify the option `--host` by setting an environment variable named `ZOWE_OPT_HOST` to the desired value.
- For more information on defining environment variables, please reference:  
<https://docs.zowe.org/stable/user-guide/cli-configuringcli.html#defining-environment-variables>
- When a Zowe command is run, the order of precedence for determining the option values to use is:
  - Command line arguments
  - Environment variables
  - Profile settings
  - Default values



# Updating the Jenkinsfile

- Add the following command inside the `withCredentials` block to instruct Jenkins to run the gulp build task you created as part of build stage:
  - `sh 'gulp build'`

# Enhance Jenkinsfile for Deploy

- Deployment for marbles requires us to copy the load modules, and activate the changes in the target CICS environment.
  - Deploy manually using CLI commands
  - Create and implement a Deploy gulp task
  - Create and implement a Deploy Jenkins stage

# Create and implement a Deploy Jenkins stage - Step 4

- Implement Deploy stage
  - Uncomment withCredentials block
  - Call the `gulp deploy` task that you created. This will need placed inside the inner withCredentials block.
  - Note: Plugins also inherit the `ZOWE_OPT_` vars, but can be overridden on the command line.



## Section VII:

*Add Automated Testing to Continuous Integration*

# Testing Step in CI - Introduction

- Testing is another step that is commonly automated. Once you've built up a suite of automated tests, you may want to integrate them into your CI process. Running these tests on every CI run is crucial as passing tests increase your confidence in the code changes and the automation.
  - Identify set of tests that are worth running in every CI run. You may want to avoid time-consuming performance tests for a stage further down the pipeline such as before deploying to production. All of the tests invoked by `npm test` are system tests that we want to include.
  - Create a stage in Jenkins to invoke these tests after code is checked in, built and deployed.
- **Note:** Consider exploring code coverage tools to obtain metrics of code that is tested by your automated tests.

# Create Test stage

- We've already built an automated test using Mocha. Let's implement a test stage in Jenkins to call it.
  - Remove the call to the Test script and the TEST ENV var under the environment directive.
  - Uncomment withCredentials block in the 'test' stage
  - Call the `npm test` command that runs your test suite.

# Run the pipeline

- After adding the Test stage, test out the pipeline by pushing your changes and running it manually.
- We need to:
  - Commit and Push Code to GitHub
  - Log in to Jenkins and build your project
    - Debug any issues that may arise. Reach out to facilitator for guidance if needed.

# Commit and push changes to GitHub

- To Review the files you have changed before committing:
  - `git status`
- There may endeavor reports you wish to delete, commit or only keep locally.
  - If you wish to only keep them locally, add `endeavor-report*.txt` to your `.gitignore` file in your project's root directory. You can run `git status` again to verify you no longer see the endeavor-report files.
- Commit your changes when satisfied:
  - `git commit -a -m "Add gulp build tasks"`



# Commit and push changes to GitHub

- Push your changes to GitHub:
  - `git push`
- If you are prompted for your username and password:
  - Username: `zowe-001`
  - Password: `Zowe-Workshop-01`

# Log in to Jenkins

- Jenkins is a hosted application running on a web server. You can access it from most web browsers.
- Log in to Jenkins: `http://mfwsone.broadcom.com/jenkins/`
  - Username: `workshop_001`
  - Password: `001_workshop`
- Verify that the environment is in the right starting state
  - Click on the name of your project (`Workshop_001`)
  - Select the master branch
  - Click `Build Now` in the left side menu and verify the project builds successfully.



# Share the Test stage output with the Facilitator

- Now that you've created and run the pipeline in Jenkins with the Test stage, it's time to share this with the facilitator.
- Find the URL of your successful run with the Test stage.



**Review - What have we learned?**



# Section X (optional):

*Archive Artifacts in Jenkins*

# Archive Artifacts

- For audit purposes, you may want to archive build/test results for each pipeline run.
- Uncomment the post section of the `Jenkinsfile`
- Commit and Push Code to GitHub
  - `git commit -a -m "Updating"`
  - `git push`
- Log in to Jenkins and build your project
  - Debug any issues that may arise. Reach out to facilitator for guidance if needed.

# Summary

- Congratulations on completing the Zowe workshop. You learned about the following topics today:
  - Automating testing concepts.
  - Using Mocha to built an integration test for a CICS transaction.
  - Implementing a Test stage in Jenkins to call automated tests
  - Making code changes and enjoying your automated build, deploy and test process.





**Review - What have we learned?**





Thank You