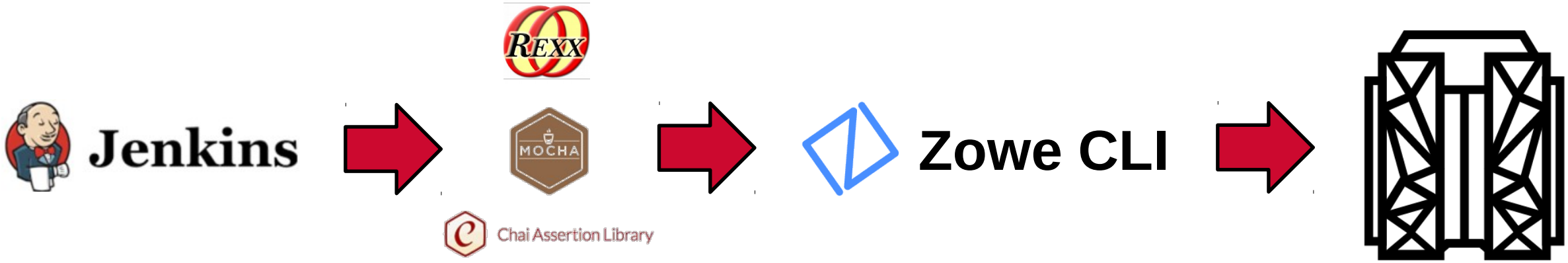

CA Brightside Infrastructure Workshop

Automate Maintenance Delivery with Jenkins and the Zowe Command Line Interface

Prerequisite Knowledge

- Required
  - A high-level understanding of what the Zowe Command Line Interface is and why it is useful
- Optional
  - Familiarity with IDEs
  - Familiarity with scripting
  - Familiarity with DevOps concepts
- Modern Tools/Languages used in this workshop
  - Zowe CLI
  - Jenkins
  - Eclipse Che
  - GitHub and git client
  - Mocha (testing framework)
  - ooRexx (task runner)
  - JavaScript

Agenda

- Project Setup (~15 minutes)
- Use the command line interface interactively to apply maintenance to a mainframe application to better understand the automation that we will write (~30 minutes)
- Develop automation to apply maintenance to a mainframe application using a task runner and the CLI. (~45 minutes)
- Develop automation to test that maintenance has been successfully applied using a modern testing framework and the CLI. (~30 minutes)
- Configure Jenkins and develop Jenkins pipeline to orchestrate the automation (~60 minutes)

Project Setup

GitHub Access

- Ensure you can access the GitHub project for this exercise at `https://github.com/chipset/Zowe-Maintenance-B-13.git`
- In this workshop, you will need to be added as a collaborator to a GitHub repository. Please share your assigned GitHub repository (Zowe-Maintenance-B-13) and public GitHub ID with the workshop administrator. If you do not have a public GitHub ID, please go to github.com and create one.

Jenkins Access

- Ensure you can access Jenkins at `http://mfwstwo.broadcom.com/jenkins/`
- Ensure you can login
  - Username: `workshop_013`
  - Password: `013_workshop`

Eclipse Che Access

- Ensure you can access Eclipse Che at `http://mfwstwo.broadcom.com/dashboard/`
- Ensure you can login
  - Username: `workshop_013`
  - Password: `013_workshop`
- Select a recent workspace under the RECENT WORKSPACES section in the left side navigation.

Git Clone and Project Setup

- After the workspace is initialized and operational, a "Welcome" page should appear. On the page, click "Git Clone…", enter in the repository URL (`https://github.com/chipset/Zowe-Maintenance-B-13.git`), and press enter. Clone your project into the projects folder under root. Tip: You will need to change the select box from theia to /.
  - You should then see the cloned project in your explorer. You can navigate to the explorer by clicking the explorer icon in the left side navigation of the IDE. 
- Open the terminal (Terminal -> Open terminal in specific container and select zowe/cli).
  - In the terminal, issue `cd Zowe-Maintenance-B-13`
  - This is normally performed, but this work **has already been performed** for you:
    - In the terminal, run `npm install gulp -g` to install the gulp CLI globally (optional for the REXX alternative). If you can not install tools globally in your Che environment, gulp commands can be preceded by npx. For example, `npx gulp download`
    - Run `npm install` to install the dependencies for this project.

CLI – Configuration

- Within your Eclipse Che development environment, you will need to create profiles to interact with the z/OS Management Facility (z/OSMF) and CA OPS/MVS services on z/OS.
- The creation of these profiles has been automated to save time. Please issue the following command on the terminal: `rexx rexxfile setupProfiles`
  and enter the following when prompted:
  Host name or IP address: `35.226.100.133`
  Username: `CUST013`
  Password: `CUST013`
- For awareness, the automation runs commands similar to:

```
zowe profiles create zosmf PE02 --host 35.226.100.133 --port
443 --user CUST013 --password CUST013 --reject-unauthorized
false
```

CLI – Tips

- You can press the up arrow to retrieve previous commands in the console.
- You can append -h to any command to learn more about command syntax and the available options.

CLI – Set Up z/OSMF Profile

- Verify connectivity to z/OSMF by issuing the following command:

```
zowe zosmf check status
```

- Verify you can list data sets by issuing the following command:

```
zowe files list ds "CUST013.MTE.*"
```


CLI – Validate CA OPS/MVS Profile

- Validate the CA OPS/MVS profile was successfully created to connect to the CA OPS/MVS instance by issuing the following command:

```
zowe ops show resource GS13001V
```


BROADCOM

Project Set Up Complete

BROADCOM

Using Zowe CLI Interactively to Apply Maintenance to SYSVIEW

Use CLI to download software maintenance

- Typically, you would download software maintenance from some trusted server. For simplicity in this workshop, we will use the Zowe CLI to download the software maintenance from a location on z/OS.
- The PTF to download is `SO07038` and it is located in USS under `/u/users/baumi07`. The PTF should be downloaded to "`bin/SO07038`".
- To download the PTF, in the integrated terminal, issue:
  ```
  zowe files download uf "/u/users/baumi07/SO07038" -f "bin/SO07038" -b
  ```
- Please confirm the file has been downloaded to a local folder named `bin`.

BROADCOM

Use CLI to upload software maintenance patch to USS

- The PTF that you downloaded is the software maintenance that needs to be applied to SYSVIEW. We will use the Zowe CLI to upload the file to our home directory in USS.
- In the integrated terminal, issue:

```
zowe files upload ftu "bin/SO07038"
"/u/users/cust013/SO07038" -b
```

- Please confirm the file has been uploaded by issuing the following command:

```
zowe files list uss "/u/users/cust013"
```


Review Workshop JCL

- All JCL for this workshop is in the CUST013.MTE.JCL PDS. You can easily download all members of this PDS via the CLI by issuing the following command:

```
zowe files download am "CUST013.MTE.JCL" -e jcl
```


BROADCOM

Run SMPE RECEIVE Job

- Submit the RECEIVE job by issuing the following command in the integrated terminal:

```
zowe jobs submit ds "CUST013.MTE.JCL(RECEIVE)" -d "job-archive/receive"
```

  The command will submit the job, await its completion, and download the spool output to the `job-archive/receive` directory.

  Review the output of the job for any ++HOLDDATA before proceeding (`./job-archive/receive/JOB*****/SMPEUCL/SMPRPT.txt`). Note that the `JOB*****` should be replaced by the `jobid` returned from the previous command.

BROADCOM

Run SMPE RECEIVE Job

- To receive a response that is easy to parse, you can append the `--rfj` flag to receive the response in JSON format. Try submitting the job again with the `--rfj` flag. Note that the job will complete with condition code 8 because we have already received the maintenance:

```
zowe jobs submit ds "CUST013.MTE.JCL(RECEIVE)" -d "job-archive/receive" --rfj
```


Run SMPE APPLY CHECK Job

- Similar to RECEIVE, submit the APPLYCHK job by issuing the following command in the integrated terminal:

```
zowe jobs submit ds "CUST013.MTE.JCL(APPLYCHK)" -d "job-archive/apply-check"
```

  Review the output of the job for any ++HOLDDATA before proceeding (`./job-archive/apply-check/JOB*****/SMPEUCL/SMPRPT.txt`). Note that the `JOB*****` should be replaced by the `jobid` returned from the previous command. Also see that we bypass Doc and Restart holds. We will automate resolving restart holds.

  Note that the module Maintained_Member in the CNM4BLOD SYSLIB is a module impacted by this PTF.

Run SMPE APPLY Job

- Using your knowledge from the previous exercises, try to run a command that will submit `CUST013.MTE.JCL(APPLY)` and download the output to `job-archive/apply`

- The solution is on the next slide

BROADCOM

Run SMPE APPLY Job

- Similar to RECEIVE and APPLYCHK, submit the APPLY job by issuing the following command in the integrated terminal:
  `zowe jobs submit ds "CUST013.MTE.JCL(APPLY)" -d "job-archive/apply"`

  Review the output of the job before proceeding (`./job-archive/apply/JOB*****/SMPEUCL/SMPRPT.txt`). Note that the `JOB*****` should be replaced by the `jobid` returned from the previous command.

Shutdown SYSVIEW using OPS/MVS SSM

- The STCs managed by SSM required to run the SYSVIEW instance are GS13001V and GS13001U. Automation has already been written start & stop these resources.
- Commands similar to the following are exercised in the scripts:

```
zowe ops stop resource
zowe ops start resource
zowe ops show resource
```

- Run the following command to stop the resources:

```
rexx rexxfile stop
```


Copy SMPE lib to the runtime environment

- To copy the SMPE lib to the runtime environment, we will submit an IEBCOPY job. A Rexx task has already been created for you that submits the following command, ensures successful completion, and logs output:

```
zowe jobs submit ds "CUST013.MTE.JCL(IEBCOPY)" -d "job-archive/copy"
```

- Run the following command to copy the SMPE lib to runtime environment:

```
rexx rexxfile copy
```


Use z/OSMF workflow to resolve holddata

- We had a restart hold on our PTF. We could resolve holddata via z/OSMF workflows. As an example, issue the following command to trigger a workflow:
  ```
  rexx rexxfile restartWorkflow
  ```

- The automation essentially runs the following Zowe CLI command:
  ```
  zowe zos-workflows start workflow-full –workflow-name
  restartSysview13 --wait
  ```

APF authorization

- After startup, the fixed module may require APF authorization. This process has also been scripted. Simply issue `rexx rexxfile apf` to complete.

- The script issues the following command:

```
zowe console issue command "SETPROG
APF,ADD,DSNAME=PRODUCT.SVRUN13.MSTRBRS.CNM4BLOD,SMS" --
console-name cust013
```

  and verifies the message ID `CSV410I` is present since the expected output is:

```
CSV410I SMS-MANAGED DATA SET
PRODUCT.SVRUN13.MSTRBRS.CNM4BLOD ADDED TO APF LIST
```

Restart SYSVIEW using OPS/MVS SSM

- Run the following command to start the resources

```
rexx rexxfile start
```


Verify maintenance has been successfully applied

- To verify the maintenance has been applied, we will view the fix level of the particular module of interest, Maintained_Member. In SYSVIEW, this can be accomplished by using the LISTDIR command:
  ```
  listdir
  PRODUCT.SVRUN13.MSTRBRS.CNM4BLOD(Maintained_Member),,,modid
  ```
- We have a job that will run this command in batch, `CUST013.MTE.JCL(CHECKVE)`. Try to submit this job and verify the FixLevel of Maintained_Member is SO07038.
- Detailed instructions are provided on the next slide.

Verify maintenance has been successfully applied

- To submit the batch test and download the output, issue:
  `zowe jobs submit ds "CUST013.MTE.JCL(CHECKVE)" -d "job-archive/version-check"`

- Manually verify that the fix level of `Maintained_Member` is SO07038 by reviewing `./job-archive/version-check/JOB*****/SYSVIEW/SYSPRINT.txt`. Note that the `JOB*****` should be replaced by the `jobid` returned from the previous command.

Maintenance successfully applied!

BROADCOM

Using Zowe CLI Programmatically to Automate the Application of Maintenance to SYSVIEW

Restore Maintenance Level to SYSVIEW

- Before automating the application of maintenance to SYSVIEW, we need to completely restore our previous maintenance level. This has been automated and the scripts are available for your review in `rexxfile.rex`
- To reset, issue `rexx rexxfile reset`

Review Steps to Interactively Apply Maintenance to SYSVIEW

- Some of these steps were already automated but this is the full list of manual commands needed.
- `zowe files download uf "/u/users/baumi07/SO07038" -f "bin/SO07038" -b`
- `zowe files upload ftu "bin/SO07038" "/u/users/cust013/SO07038" -b`
- `zowe jobs submit ds "CUST013.MTE.JCL(RECEIVE)" -d "job-archive/receive"`
- `zowe jobs submit ds "CUST013.MTE.JCL(APPLYCHK)" -d "job-archive/apply-check"`
- `zowe jobs submit ds "CUST013.MTE.JCL(APPLY)" -d "job-archive/apply"`
- `zowe ops stop resource GS13001U`
- `zowe ops show resource GS13001U`

BROADCOM

Review Steps to Interactively Apply Maintenance to SYSVIEW

- `zowe ops stop resource GS13001V`
- `zowe ops show resource GS13001V`
- `zowe fmp copy ds "PRODUCT.SYSV15.SMPE13.CNM4BLOD" "PRODUCT.SVRUN13.MSTRBRS.CNM4BLOD"`
- `zowe ops start resource GS13001V`
- `zowe ops show resource GS13001V`
- `zowe ops start resource GS13001U`
- `zowe ops show resource GS13001U`
- `zowe console issue command "SETPROG APF,ADD,DSNAME=PRODUCT.SVRUN13.MSTRBRS.CNM4BLOD,SMS" --cn cust013`

BROADCOM

How we can automate these tasks?

- Shell scripts, JavaScript, Python, etc.
- Modern task runners like Gulp.js
  - https://gulpjs.com/
  - Gulp is a toolkit for automating tasks
  - Gulp is an npm package familiar to JavaScript developers

- If your preferred scripting language is Rexx, you can make use of Open Object Rexx on your PC. Check out http://www.oorexx.org/ for more details.
  - ooRexx is 90% similar to z/OS standard REXX.
  - Fully compatible Windows & Linux scripting.

ooRexx: Getting Started

- Rexx tasks can be run form the command line in the following format:
  - rexx <rexx-script.rex> <args>
- Try issuing `rexx rexxfile help` in the integrated terminal.
- To help get you started, a few Rexx tasks have already been created for you.
  - `rexx rexxfile download` downloads the maintenance from a trusted location.
  - `rexx rexxfile receive` receives the maintenance into the SMPE environment
  - `Rexx rexxfile apply-check` checks the maintenance update
  - `rexx rexxfile start` starts the SYSVIEW SSM managed resources.
  - `rexx rexxfile stop` starts the SYSVIEW SSM managed resources.
- The scripts that power these commands are located in `rexxfile.rex`

BROADCOM

Rexxfile download

```
download:
   task = 'download' ; call display_init task
   command = 'zowe files download uf "' ,
             || serverFolder ||'/' || serverFile || '" -f "' ,
             || localFolder  ||'/' || localFile  || '" -b --rfj'
   call simpleCommand command, "command-archive/download"
   task = 'download' ; call display_end task
return
```

- At the beginning and at the end of each subroutine there are calls to display the timestamp of the start and on the finish of the task.
- rexxfile download exercises the `zowe files download uf` command. To make our `rexxfile.rex` easier to maintain and customize, parameters have been organized in a `config.json` file which is located at the project's root. Review `config.json` to view all the configurable variables for this project.
- The command is then passed into a `simpleCommand` function along with a directory to log the output to. Control will be returned when the upload task is complete. Let's investigate this function.

Rexxfile download - simpleCommand

```
simpleCommand:
   parse arg command,dir,expectedOutputs
   stem = rxqueue("Create")
   call rxqueue "Set",stem
   interpret "'"command" | rxqueue' "stem
   drop sal.; j = 0; sal = ''
   do queued()
      pull sal
      j=j+1; sal.j = sal
   end
   sal.0 = j
   call rxqueue "Delete", stem
   call writeToFile dir   /* log output */
   if expectedOutputs <> '' then call verifyOutput expectedOutputs
return
```

- `simpleCommand` provides logging and error handling for us. We send a command to run, a directory to log output to, and optionally a list of expected outputs from the command. `simpleCommand` then runs the command, logs the output, optionally verifies the output and makes the callback with appropriate error information. It should be used whenever you want to run a command, alert if the command has an error, and optionally alert if the output does not contain an expected output.

Rexxfile receive

```
receive:
   task = 'receive' ; call display_init task
   ds = remoteJclPds ||'('|| receiveMember ||')'
   call submitJobAndDownloadOutput ds, "job-archive/receive", 0
   call parseHolddata "job-archive/receive/" || jobid || "/SMPEUCL/SMPRPT.txt"
   task = 'receive' ; call display_end task
return
```

- Rexxfile receive calls the `submitJobAndDownloadOutput` function with three arguments
  - The data set that contains the JCL to submit
  - The location to save the job output to
  - The maximum return code of the job that is acceptable

- Let's investigate the `submitJobAndDownloadOutput` function

Rexxfile receive - submitJobAndDownloadOutput

- This function submits job, awaits completion, downloads output to the desired directory, logs the command output to a file, and verifies the condition code is less than some supplied value.
- It exercises the `zowe jobs submit data-set` command.
- The script submits the command with the `--rfj` flag to receive the output in JSON format for easy parsing. The script ensures the job completes with a return code value less than or equal to the supplied `maxRC`, else it fails the task.
- This function should be exercised when you want to submit a job, download the output, and confirm the job completes with a satisfactory condition code.

Rexxfile receive - parseHolddata

- In the case of the receive task, the callback will parse the hold data. This function accepts one parameter, the local filepath to read the holddata from. In this case, we point to the downloaded SMPRPT spool content.
- The function parses the holds and looks at SMP/E hold reason ids. It then creates a representative holddata/actions.json file. The contained JSON object has the following three Boolean properties
  1. remainingHolds -> True designates there are holds that cannot be handled by automation and requires manual intervention
  2. restart -> True designates automation needs to run to restart SYSVIEW
  3. reviewDoc -> True designates a doc hold was found

Rexxfile apply-check

```
applyCheck:
   desc = 'applyCheck , Apply Check Maintenance'
   call display_init
   ds = remoteJclPds ||'('|| applyCheckMember ||')'
   call submitJobAndDownloadOutput ds, "job-archive/applyCheck", 0
   call display_end
return
```

- rexx rexxfile applyCheck is similar to rexx rexxfile receive but a little simpler since the callback to the submitJobAndDownloadOutput is just the callback that signals the rexx task's completion. In this case, we are concerned only with the return code and we are not parsing holddata. However, we could parse the spool content for impacted modules. This would enable us to remove the maintainedMember and maintainedPds properties from config.json since we could dynamically determine them. While on the topic, the JCL (APPLY, APPLYCHK, etc.) could also be parameterized and included within the project itself to further simplify the config.

Rexxfile start & stop

```
start:
   task = 'start' ; call display_init task
   call start1; call start2
   task = 'start' ; call display_end task
return

stop:
   task = 'stop' ; call display_init task
   call stop2; call stop1
   task = 'stop' ; call display_end task
return
```

- Rexxfile start and stop tasks combine multiple tasks into a single action. Two resources must be brought up to start SYSVIEW.
- Rexxfile start first starts `GS13001V` then `GS13001U`
- Rexxfile stop first stops `GS13001U` then `GS13001V`
- Let's investigate the `start1` task

BROADCOM

Rexxfile start1

```
start1:
   task = 'start1' ; call display_init task
   call changeResourceState ssmResource1,"UP"
   task = 'start1' ; call display_end task
return
```

- rexx rexxfile start1 calls the `changeResourceState` function with two arguments
  - The SSM managed resource to change the state of
  - The desired state ("UP" or "DOWN")
- Let's investigate the `changeResourceState` function

Rexxfile start1 - changeResourceState

- The function accepts up to three parameters:
  - The SSM managed resource to change the state of
  - The desired state ("UP" or "DOWN")
  - Optional parameter where you can supply the name of a data set to be APF authorized after the SSM state is changed. It could be useful when changing the desired state of a resource to "UP".
- Depending on the desired state, it either exercises the `zowe ops start resource` command or the `zowe ops stop resource` command. The function calls another recursive function named awaitSSMState that polls the state of the resource using the `zowe ops show resource` command until the desired state or a timeout is reached. The output of each command issued is logged to a file. Finally, if the APF option is used, the function uses a `zowe console issue cmd` command to APF authorize a data set.
- This function should be used whenever you want to change the state of a SSM managed resource.

Rexxfile copy

```
copy:
   task = 'copy' ; call display_init task
   ds = remoteJclPds ||'('|| copyMember ||')'
   call submitJobAndDownloadOutput ds, "job-archive/copy", 0
   task = 'copy' ; call display_end task
return
```

- The `rexxfile copy` task handles copying the designated library from the SMP/E environment to the runtime environment.

Let’s Automate!

- The first task has already been automated:
  - `zowe files download uf "/u/users/baumi07/SO07038" -f "bin/SO07038" -b`

- Issue `rexx rexxfile download` to download the PTF.

Rexxfile upload

- The next task to automate is:
  - `zowe files upload ftu "bin/SO07038" "/u/users/cust013/SO07038" -b --rfj`
- Use the `rexx rexxfile download` task as a template.
- When creating additional rexx tasks, try to keep them in alphabetical order in `rexxfile.rex`.
- First, you will need to change the data set your are submitting. You will need to review `config.json` to find the property that represents the local and remote locations.
- Second, you should change where you are storing the output from `command-archive/download` to `command-archive/upload`
- A completed solution is available for reference on the next slide.
- Try it out! Be sure to save `rexxfile.rex` first. Issue `rexx rexxfile upload` to exercise the automation.

BROADCOM

Rexxfile upload

```
upload:
   task = 'upload' ; call display_init task
   command = 'zowe files upload ftu "' ,
                 || localFolder  ||'/' || localFile  ||'" "' ,
                 || remoteFolder ||'/' || remoteFile ||'" -b --rfj'
   call simpleCommand command, "command-archive/upload"
   task = 'upload' ; call display_end task
return
```

- rexx rexxfile upload exercises the `zowe files upload ftu` command.
- Similar to the `rexx rexxfile download` task, the command is then passed into the `simpleCommand` function along with a directory to log the output to.

Receive and applyCheck!

- The next couple tasks have already been automated:
  - `zowe jobs submit ds "CUST013.MTE.JCL(RECEIVE)" -d "job-archive/receive"`
  - `zowe jobs submit ds "CUST013.MTE.JCL(APPLYCHK)" -d "job-archive/applyCheck"`

- Issue `rexx rexxfile receive` to receive the maintenance into the SMPE environment.
- Issue `rexx rexxfile applyCheck` to check the maintenance update

Rexxfile apply

- The next task to automate is:
  - `zowe jobs submit ds "CUST013.MTE.JCL(APPLY)" -d "job-archive/apply"`
- Use the `rexx rexxfile receive` task as a template.
- First, you will need to change the data set your are submitting. You will need to review `config.json` to find the property that represents the `APPLY` member.
- Second, you should change where you are storing the output from `job-archive/receive` to `job-archive/apply`
- A completed solution is available for reference on the next slide.
- Try it out! Issue `rexx rexxfile apply` to exercise the automation.

Rexxfile apply

- Sample apply task:

```
apply:
   task = 'apply' ; call display_init task
   ds = remoteJclPds ||'('|| applyMember ||')'
   call submitJobAndDownloadOutput ds, "job-archive/apply", 0
   task = 'apply' ; call display_end task
return
```


Rexxfile stop

- These tasks have already been automated:
  - `zowe ops stop resource GS13001U`
  - `zowe ops show resource GS13001U`
  - `zowe ops stop resource GS13001V`
  - `zowe ops show resource GS13001V`

- Issue `rexx rexxfile stop` to stop the SYSVIEW resources.

Rexxfile copy

- This task has also already been automated:
  ```
  zowe jobs submit ds "CUST013.MTE.JCL(IEBCOPY)" -d "job-archive/copy"
  ```
- Run the following command to copy the SMPE lib to runtime environment:
  ```
  rexx rexxfile copy
  ```


Rexxfile apf

- The final task to automate in this section was:
  - `zowe console issue command "SETPROG APF,ADD,DSNAME=PRODUCT.SVRUN13.MSTRBRS.CNM4BLOD,SMS" --cn cust013`
- This task has been automated for you. Notice how the task checks to ensure the data set name and correct message id are supplied in the output. Recall that we expected output to be:

  `CSV410I SMS-MANAGED DATA SET PRODUCT.SVRUN13.MSTRBRS.CNM4BLOD ADDED TO APF LIST`

```
apf:
   task = 'apf' ; call display_init task
   ds = runtimeEnv ||'.'|| maintainedPds
   command = 'zowe console issue command "SETPROG APF,ADD,DSNAME=' || ds || ',SMS" --cn ' || consoleName
   output  = "CSV410I" ds
   call simpleCommand command,"command-archive/apf",output
   task = 'apf' ; call display_end task

return
```

- Issue `rexx rexxfile apf` to exercise the automation.

BROADCOM

Rexxfile start

- These tasks have already been automated:
  - `zowe ops start resource GS13001V`
  - `zowe ops show resource GS13001V`
  - `zowe ops start resource GS13001U`
  - `zowe ops show resource GS13001U`

- Issue `rexx rexxfile start` to start the SYSVIEW resources.

Mission Complete: Delivery of Maintenance is Now Automated

BROADCOM

Using Zowe CLI Programmatically to Verify the Successful Application of Maintenance to SYSVIEW

Review Steps to Verify Maintenance was Applied to SYSVIEW

- `zowe jobs submit ds "CUST013.MTE.JCL(CHECKVE)" -d "job-archive/version-check"`
- Manually verify that the fix level of `Maintained_Member` is SO07038 by reviewing `./job-archive/version-check/JOB*****/SYSVIEW/SYSPRINT.txt`. Note that the `JOB*****` should be replaced by the `jobid` returned from the previous command.

How we can automate this test?

- Shell scripts, JavaScript, Python, etc.
- Modern test frameworks like Mocha.js
  - https://mochajs.org/
  - Modern test frameworks have assertion libraries that make testing quick and easy.

- The tests are located in `test/test.js` from the project's root.

Mocha Tests

- Tests are located in `test/test.js`.
- Navigate to the describe `Maintenance` test suite. Under the project's test suite, there is a test plan titled `Module Check` where a single test resides: `it('should have maintenance applied'…`
  This test should verify the maintenance was applied successfully.
- This test calls `getModuleFixLevel` to retrieve the input module fix level. The function returns an `awaitFixLevelCallback` which consists of an `Error` object in case something undesirable occurs and a `string` containing the fixLevel if the module is found in the table or null if not found. The function submits a job using the `zowe jobs submit data-set` command, downloads the spool content, logs the command output, and verifies the job completed with condition code 0. Then the function parses the appropriate downloaded spool file, just like any other file, and returns the fixLevel if found, null if not found.

Assertion Libraries

- Assertion libraries are tools that are used to verify things are as we expect.
- Mocha can use the Chai Assertion Library
- To verify something is equal to a certain value, we can employ an assertion statement:
  `assert.equal(actual, expected, "message if not as expected");`
  For example,
  `assert.equal(banana.color, yellow, "Banana is no longer yellow");`
- Try adding the assertion in the test just before the `done()` is called.
- Test whether the actual `fixLevel` is equal to `config.expectedFixLevel` and if not, output the message `"Fix Level is not as expected for " + config.maintainedMember`
- A completed sample is provided on the next slide.

Complete Test

```
describe('Maintenance', function () {
  // Change timeout to 60s from the default of 2s
  this.timeout(60000);

  /**
   * Test Plan
   * Run MODID utility to verify module is appropriately updated
   */
  describe('Module Check', function () {
    it('should have maintenance applied', function (done) {
      // Get Fix Level for maintained member specified in config
      getModuleFixLevel(config.maintainedMember, function(err, fixLevel){
        if(err){
          throw err;
        }
        assert.equal(fixLevel, config.expectedFixLevel, "Fix Level is not as expected for " + config.maintainedMember);
        done();
      });
    });
  });
});
```


BROADCOM

Invoking the Test

- Npm scripts can be developed in `package.json` located at the project's root. Note the following snippet

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

- This means that the test script can be invoked with `npm test`
- Note that a reporter is also used to create graphical reports with no additional effort required by us.
- Give it a try! Run `npm test` in your terminal.

Viewing the Test Reports

- After the tests have run, a report should be generated. From the root directory, it is `mochawesome-report/mochawesome.html`. To view the report, right click on the html file and select `Open With>Preview`. This report is will also be archived in Jenkins in a later section.

Congratulations! You have now automated the testing of the application of maintenance to SYSVIEW!

BROADCOM

Orchestrating the Application of Maintenance to SYSVIEW via Jenkins

Restore Maintenance Level to SYSVIEW

- Before proceeding, issue `rexx rexxfile reset` to reset the maintenance level.

Jenkins Introduction

- Jenkins is an open source automation server commonly used for continuous integration testing.
- However, pipelines can be built to facilitate other automated tasks. It is useful for centralizing automation for multiple collaborators to develop, exercise, and manage.
- In this final section, we will construct a Jenkins pipeline that automate the delivery of maintenance to SYSVIEW using automation we have already developed. By developing a pipeline, we will have an audit trail and more transparency into the activities that are being performed.
- More information is available on Jenkins at https://jenkins.io/.

Jenkins Kickoff

- Navigate to our Jenkins instance at `http://mfwstwo.broadcom.com/jenkins/`
- Login
  - Username: `workshop_013`
  - Password: `013_workshop`
- Click on the workshop_013 project
- Click on the `master` branch
- Click on the `Build Now` button located in the top left navigation menu.
- Watch the pipeline progress in a modern view. Click the `Blue Ocean` button located in the top left navigation menu.
- The pipeline is just a template. You will need to fill in the details in this exercise.

Jenkins Kickoff

- The code that drives the Jenkins pipeline is contained within the Jenkinsfile at the project's root. Let's walk through this file to understand how it works.
- First, we have: `agent { label 'zowe-agent' }`
  - The agent directive dictates where the pipeline will run. In this case, the pipeline will run in a 'zowe-agent' which is the label to a docker container in our Jenkins configuration. This docker container contains the Zowe CLI as well as the necessary and CA OPS/MVS for Zowe CLI plugin.

Jenkins Kickoff

- Next, we have:

```
environment {
    // z/OS Host Information
    ZOWE_OPT_HOST=credentials('eosHost')
}
```

  - The environment directive allows for environment variables to be defined
  - Note that the directive supports a special helper method `credentials()` which is used to access Jenkins credentials by their unique identifier. In this case, a credential with ID of `eosHost` is accessed.
  - If you return to your workshop_013 project in the classic view (not Blue Ocean), you will see a `Credentials` button in the left side navigation. If you click that, you will be able to see two credentials scoped to your workshop_013 project. One is `eosHost`, and the other is `eosCreds`.

Command Line Precedence

- The CLI can receive options from three places. In order, they are
  - Directly on the command line (e.g. `--host myHost`)
  - Environment variables (`ZOWE_OPT_` followed by the option name, e.g. `ZOWE_OPT_HOST`)
  - Profiles (which you set up earlier in your Che environment)
- In this exercise, we will use `ZOWE_OPT_HOST`, `ZOWE_OPT_USER`, and `ZOWE_OPT_PASSWORD` to influence our commands.

Jenkins Kickoff

- Next, we have:

```
stage('local setup') {
    steps {
        sh 'node --version'
        sh 'npm --version'
        sh 'zowe --version'
        sh 'rexx -version'
        sh 'zowe plugins list'
        //sh 'npm install gulp-cli -g'
        sh 'npm install'
        //Create zosmf and fmp profiles, env vars will provide host, user, and password details
        sh 'zowe profiles create zosmf Jenkins --port 443 --ru false --host dummy --user dummy --password dummy'
        sh 'zowe profiles create fmp Jenkins --port 6001 --protocol https --ru false --host dummy --user dummy --password dummy'
    }
}
```

  - Stages are logical sets of steps
  - In the `local setup` stage, we report the versions of Node, NPM, and Zowe CLI as well as the list of available Zowe CLI plugins in our `zowe-agent` environment.
  - ooRexx should be installed on our base operating system.
  - Finally, we set up our zosmf and fmp profiles. `dummy` is used for host, user, and password since we will provide this information via environment variables. The ops profile will be set up later. The ops profile does not currently expose its profile options on each command so we will need access to our user and password environment variables to create the ops profile.

Jenkins Kickoff

- Next, we have:

```
stage('Download Maintenance') {
    steps {
        withCredentials([usernamePassword(credentialsId: 'eosCreds', usernameVariable: 'ZOWE_OPT_USER', pass
            sh 'echo download'
        }
    }
}
```

  - A stage for downloading the maintenance. Currently, the stages just echo their intention as output. We will be changing the content of these steps to make use of the automation we have already written.
  - Notice the script is run in a "withCredentials" block. Within these blocks, the code will have access the username and password variables that have been set up in Jenkins.

Jenkins Kickoff

- Next, we have:

```
stage('Upload Maintenance') {
    steps {
        withCredentials([usernamePassword(credentialsId: 'eosCreds', usernameVariable: 'ZOWE_OPT_USER', pass
            sh 'echo upload'
        }
    }
}
stage('Receive') {
    steps {
        withCredentials([usernamePassword(credentialsId: 'eosCreds', usernameVariable: 'ZOWE_OPT_USER', pass
            sh 'echo receive'
        }
    }
}
```

  - A stage for uploading maintenance to USS and a stage for receiving the maintenance into our SMP/E environment.

Jenkins Kickoff

- Next, we have:

```
stage('Apply Check') {
    steps {
        // script {
        //     def actions = readJSON file: 'holddata/actions.json'
        //     if (actions.remainingHolds) {
        //         input message: 'Unresolved holds detected. Please review the results of the receive job in the job-archive/receive artifacts. Proceed to applyCheck?'
        //     }
        // }
        withCredentials([usernamePassword(credentialsId: 'eosCreds', usernameVariable: 'ZOWE_OPT_USER', passwordVariable: 'ZOWE_OPT_PASSWORD')]) {
            sh 'echo applyCheck'
        }
    }
}
stage('Apply') {
    steps {
        withCredentials([usernamePassword(credentialsId: 'eosCreds', usernameVariable: 'ZOWE_OPT_USER', passwordVariable: 'ZOWE_OPT_PASSWORD')]) {
            sh 'echo apply'
        }
    }
}
```

  - A stage for Apply-Check and a stage for Apply.
  - Notice the commented out input directive in the Apply Check stage in the case manual intervention is needed (holds that cannot be resolved by our automation exist). This section can now be uncommented.

BROADCOM

Jenkins Kickoff

- Next, we have:

```
stage('Deploy') {
    steps {
        withCredentials([usernamePassword(credentialsId: 'eosCreds', usernameVariable: 'ZOWE_OPT_USER', passwordV
            //To deploy the maintenace, an OPS profile needs to be created since profile options are not exposed
            sh 'zowe profiles create ops Jenkins --host $ZOWE_OPT_HOST --port 6007 --protocol http --user $ZOWE_O
            echo 'deploy'

            // script {
            //     def actions = readJSON file: 'holddata/actions.json'
            //     if (actions.restart) {
            //         sh 'gulp restartWorkflow'
            //     }
            // }
        }
    }
}
stage('Test') {
    steps {
        withCredentials([usernamePassword(credentialsId: 'eosCreds', usernameVariable: 'ZOWE_OPT_USER', passwordV
            sh 'echo test'
        }
    }
}
```

  - A stage for Deploy and a stage for Test.
  - Notice in the Deploy stage, a zowe CLI profile is being created for OPS. This plugin does not currently expose all of its options on the command line so host, username, and password can not directly influence the commands. Therefore, we create an OPS profile with these values.
  - Also notice in the commented out section that we will check for restart holds and kickoff a workflow if needed. This section can now be uncommented.

BROADCOM

Jenkins Kickoff

- Finally, we have:

```
// post {
//     always {
//         archiveArtifacts artifacts: '*-archive/**/*.*, holddata/actions.json'
//         publishHTML([allowMissing: false,
//             alwaysLinkToLastBuild: true,
//             keepAll: true,
//             reportDir: 'mochawesome-report',
//             reportFiles: 'mochawesome.html',
//             reportName: 'Test Results',
//             reportTitles: 'Test Report'
//             ])
//     }
// }
```

  - A post stage to archive our artifacts for audit purposes. This stage can now be uncommented.

Jenkins: Download & Upload Maintenance Stages

- Replace the `echo download` command with `rexx rexxfile download` to make use of the existing automation to download the PTF

- Replace the `echo upload` command with `rexx rexxfile upload` to make use of the existing automation to upload the PTF to USS

Jenkins: Receive Stage

- Replace the `echo receive` command with `rexx rexxfile receive` to make use of the existing automation.
- Archive the artifacts of this stage. This can be done by adding: `archiveArtifacts artifacts: 'job-archive/**/*.*'` after the `withCredentials` block as shown below:

```
stage('Receive') {
    steps {
        withCredentials([usernamePassword(credentialsId: '
            sh 'rexx rexxfile receive'
        }
        archiveArtifacts artifacts: 'job-archive/**/*.*'
    }
}
```


BROADCOM

Jenkins: Apply-Check Stage

- Replace the `echo apply-check` command with `rexx rexxfile applyCheck` to make use of the existing automation.
- Archive the artifacts of this stage.

```
stage('Apply Check') {
    steps {
        script {
            def actions = readJSON file: 'holddata/actions.json'
            if (actions.remainingHolds) {
                input message: 'Unresolved holds detected. Please review the results of the receive job in the job-archive/receive artifacts. Proceed to applyCheck?'
            }
        }
        withCredentials([usernamePassword(credentialsId: 'eosCreds', usernameVariable: 'ZOWE_OPT_USER', passwordVariable: 'ZOWE_OPT_PASSWORD')]) {
            sh 'rexx rexxfile applyCheck'
        }
        archiveArtifacts artifacts: 'job-archive/**/*.*'
    }
}
```


BROADCOM

Jenkins: Apply Stage

- Try completing the Apply stage. A completed solution is available on the next slide.

Jenkins: Apply Stage

```
stage('Apply') {
    steps {
        withCredentials([usernamePassword(credentialsId: 'eosCreds', usernameVariable: 'ZOWE_OPT_USER', passwordVariable: 'ZOWE_OPT_PASSWORD')]) {
            sh 'rexx rexxfile apply'
        }
        archiveArtifacts artifacts: 'job-archive/**/*.*'
    }
}
```


BROADCOM

Jenkins: Deploy Stage

- Try completing the Deploy stage. A completed solution is available on the next slide. Hint: you will need to make use of the `stop`, `copy`, `start`, and `apf` Rexx tasks you previously created. The workflow is just a sample workflow for demonstration purposes. It does not actually restart Sysview as we will be using the OPS/MVS plug-in to accomplish this via our script. Please trigger the workflow after the copy but before the restart.

Jenkins: Deploy Stage

```
stage('Deploy') {
    steps {
        withCredentials([usernamePassword(credentialsId: 'eosCreds', usernameVariable: 'ZOWE_OPT_USER', passwordVariable: 'ZOWE_OPT_PASSWORD')]) {
            //To deploy the maintenace, an OPS profile needs to be created since profile options are not exposed on the command line
            sh 'zowe profiles create ops Jenkins --host $ZOWE_OPT_HOST --port 6007 --protocol http --user $ZOWE_OPT_USER --password $ZOWE_OPT_PASSWORD'
            sh 'rexx rexxfile stop'
            sh 'rexx rexxfile copy'
            script {
                def actions = readJSON file: 'holddata/actions.json'
                if (actions.restart) {
                    sh 'rexx rexxfile restartWorkflow'
                }
            }
            sh 'rexx rexxfile apf'
            sh 'rexx rexxfile start'
        }
    }
}
```


Jenkins: Test Stage

- Replace the `echo test` command with `npm test` to make use of the existing test automation

Push to GitHub

- Jenkins pulls the pipeline source from GitHub. Previously, you were only running the automation from within your Eclipse Che development environment. In order to update the source on GitHub, issue the following commands.
- To see what files have changed, issue: `git status`
- To commit your changes, issue: `git commit -a -m "Add Maintenance Deployment Automation"`
- To push your changes, issue: `git push`
  You will be prompted to enter your GitHub ID and password
  - Username: `zowe-013`
  - Password: `Zowe-Workshop-013`
- To confirm that you are in sync, issue: `git status`

Jenkins Pipeline Run

- Return to the `master` branch of your Jenkins project and click on the `Build Now` button located in the top left navigation menu.
- Watch the pipeline progress in a modern view. Click the `Blue Ocean` button located in the top left navigation menu.
- The pipeline should now proceed to deploy to maintenance to SYSVIEW asking for manual approvals along the way. All artifacts are stored in Jenkins and can be viewed by clicking the "Artifacts" tab in the upper right of the screen in the Blue Ocean view.
- Work with the facilitator should any issues arise.

Maintenance has been deployed via Jenkins!

Thank You