

Zowe CICS ワークショップ

Jenkins と Zowe を使用してメインフレーム アプリを自動化する

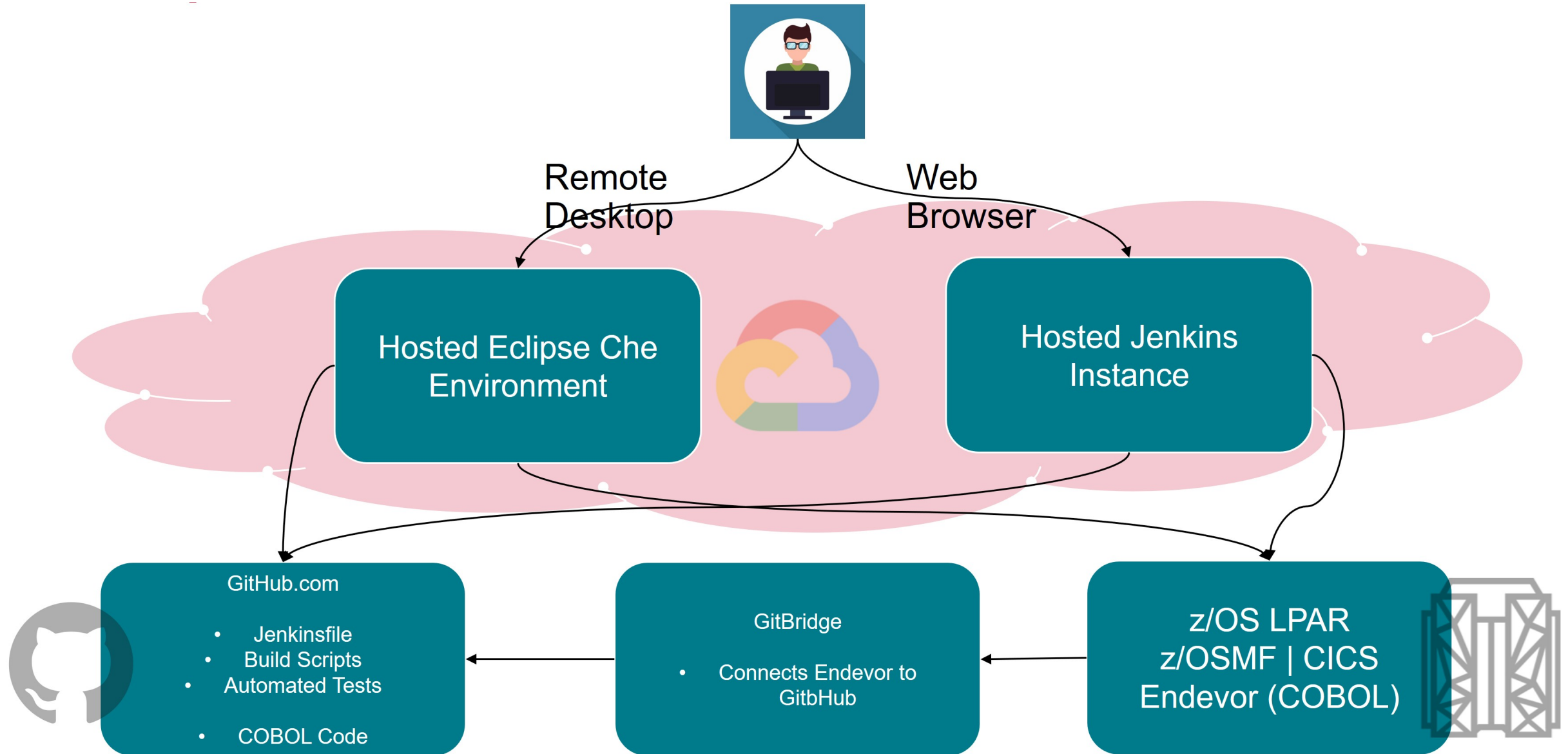


目標

- メインフレーム アプリの自動化スクリプトを作成してビルドおよびデプロイする
- オープン テスト フレームワークを使用してメインフレーム アプリの自動テストを作成する
- メインフレーム アプリ用の Jenkins CI/CD パイプラインを作成する

ワークショップ環境へのアクセス

Workshop ワークショップ環境



z/OS サービス

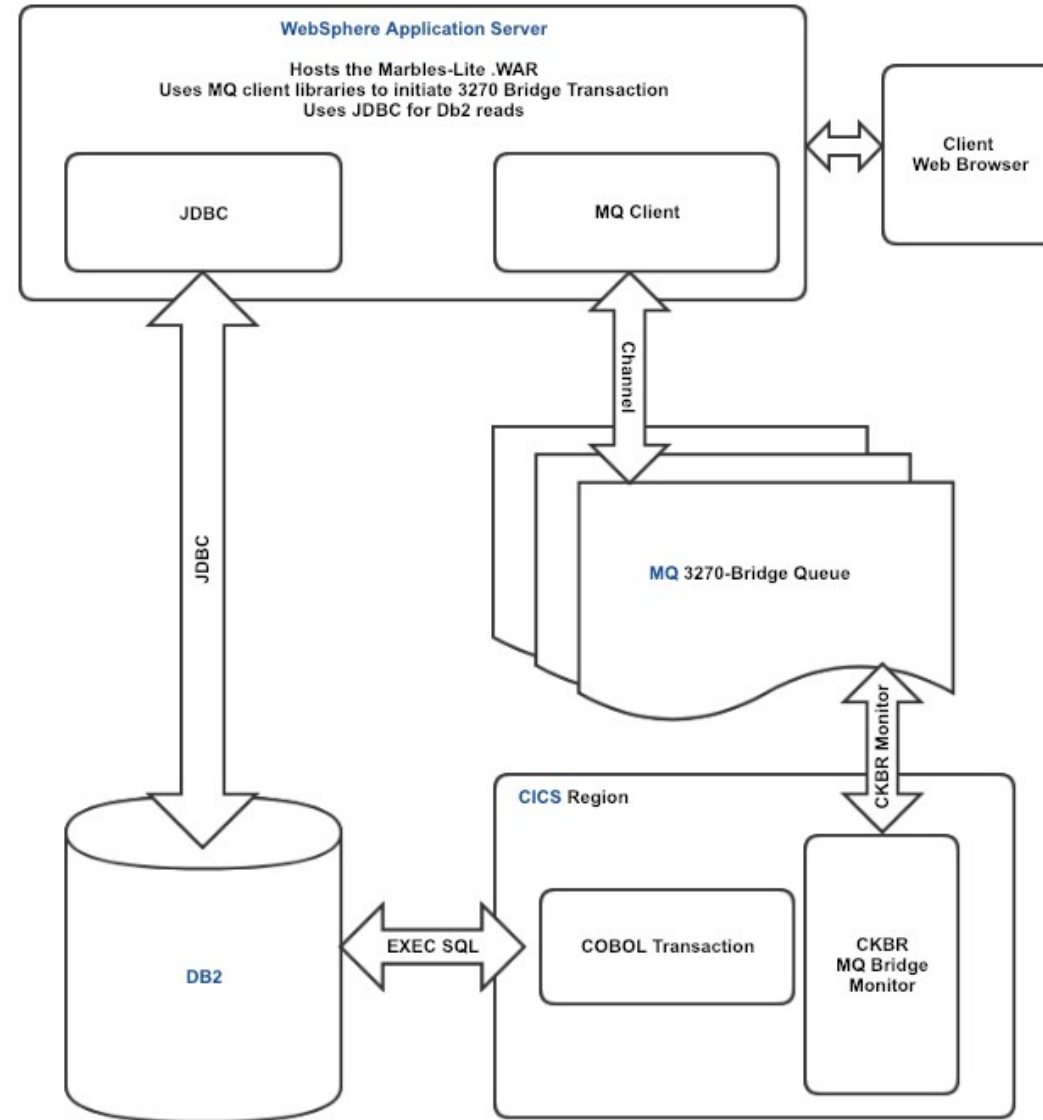
- TSO、z/OSMF、CICS、Db2、CA Endevor SCM など、Broadcom がホストするリモート z/OS LPAR 上のすべてのメインフレーム リソースにアクセスするための単一のログイン認証情報セットが割り当てられています。
- あなたのユーザー ID は CUST001.
- あなたのパスワード CUST001.

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

マール



マーブル アプリ





デモ

マール アプリの望ましい状態 - ワークショップ前

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 only the Quantity 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 - fails on click)
Gold	1 (button to update - succeeds on click)	100 (button to update - fails on click)
(button to add color - succeeds on click)		


マール アプリの望ましい状態 - ワークショップ後

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 - succeeds on click)
Gold	1 (button to update - succeeds on click)	100 (button to update - succeeds on click)
Color X	X (button to update - succeeds on click)	X (button to update - succeeds on click)
(button to add color - succeeds on click)		

サンプル CI パイプライン

シンプルなパイプラインのデモ

- CI/CD オーケストレーター
- 自動化スクリプトを処理してメインフレーム アクションを実行する
- スクリプトは Zowe CLI コマンドを呼び出します

セクション I:

概要と環境の概要と環境のセットアップ

セットアップ



セクション I のステップ

- 現在、CICS トランザクションは大理石の数量を更新することができます。このトランザクションを拡張して、既存の機能に加えて、大理石のコストを更新できるようにしたいと思います。
- ステップ
 - Endevor からリモート・デスクトップに COBOL トランザクション・コードをダウンロードします。
 - Eclipse Che でコードを編集します。
 - Endevor にコードをアップロードします。
 - Endevor でコードをビルド (生成) します。
 - 注: 以下のすべてのステップは、割り当てられたリモート Eclipse Che 環境から実行する必要があります。

開発者環境

- リモートデスクトップのアプリケーションを開く
 - Che デスクトップ: `http://mfwsone.broadcom.com/dashboard/`
 - ログインは `workshop_001`
 - パスワードは `001_workshop`
- コマンドプロンプトを開く
 - 発行 `"zowe --h"`
 - 発行 `"zowe plugins list"`
 - 発行 `"npm -h"`
 - 発行 `"git help"`

プロフィール

- メインフレーム上の異なる環境にアクセスするために作成されます。
 - 異なるユーザー名とパスワード、ポートで複数の LPARS に接続可能
 - 1つのアプリケーションに対して複数のプロファイルを作成することができます
- Zowe V2 では、プロファイルの設定に `zowe.config.json` を使用します。
- Zowe のコマンドラインを初めて使用する場合、認証情報を入力するプロンプトが表示されます
 - `cd /projects/Zowe-DevOps-A-01`
 - `zowe files list ds CUST001.*`
 - 以下のプロンプトが表示されます。
 - IP アドレス: 104.208.142.8
 - ユーザー名: CUST001
 - パスワード: CUST001
 - 間違えた場合は、再度コマンドを実行するか、または、以下のコマンドを発行してください `zowe config init`
 - Be sure you run a `zowe files list ds` command before accessing Gulp

セクション II:

Cobol コードの修正

Endevor からエレメントをダウンロードする - ステップ 1

- 変更する必要があるエレメントを特定したので、別のコマンドを使用してリモートデスクトップにダウンロードしましょう。
- **** 注意 ****: 注意 **: 以下のコマンドは、ステップ 1 のデフォルトプロファイルよりも優先されるコマンドで提供されるオプションを使用します。これには、Endevor システム、サブシステムなどが含まれます。
- ダウンロードするファイルのあるフォルダに端末を移動します。
- ダウンロードするエレメント:
 - `zowe endevor retrieve element MARBLE01 --type COBOL --to-file MARBLE01.cbl --override-signout`

ソースコードを編集する - ステップ 2

- CICS トランザクションを使用してマーブルのコストを変更する機能を実装するために、コードを更新する準備が整いました。
- エクスプローラー ビュー（左上隅）を使用してソース コードを開きます。
- 次のコード シーケンスを見つけて、強調表示されたテキストを削除し、変更をローカルに保存します：

```
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.
```

要素を Endeavor にアップロードする - ステップ 3

- ローカルでコードを変更した後、ビルドを実行するために要素を Endeavor にアップロードする必要があります。
- 端末がソース ファイルを含むフォルダーに配置されていることを確認します。
- 以下のエレメントをアップロード：
 - `zowe endeavor update element MARBLE01 --type COBOL --os --ff MARBLE01.cbl`
 - このコマンドは、要素を Endeavor にアップロードする方法に関するヘルプを提供します：
 - `zowe endeavor update element -h`



コードを生成する

エレメントを生成する

- コードの変更が Endeavor にアップロードされたので、Endeavor 生成アクションを使用してコンパイルし、エラーがあるかどうかを確認できます。
- 以下のエレメントを生成する。
 - 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`
 - **** ヒント ****: 上矢印キーを使用して、タイプを変更するだけです。
- 生成アクションが成功したことを確認してください：
 - 出力に `GENERATE of MARBLE01.COBOL finished with 0000` のようなテキストが表示されるはずです。

セクション III

Marbles アプリケーションを手動でデプロイする

デプロイメント - はじめに

- デプロイは、一般的に自動化されているもう 1 つのステップです。コードをビルドし、ロードモジュールのようなバイナリ成果物が準備できたら、これらの成果物を、プログラムを実行できるシステムにコピーしたいと思うかもしれません。
- 展開ステップの特定
- デプロイメントにおけるパラメトリケーションの要件を特定する。
 - ビルドの成果物は、開発ビルドかチームビルドかによって、異なる場所から来るのか？
 - デプロイメントシステムは、ステージによって異なりますか？
- デプロイの自動化
- **** 注 ****: デプロイメントスクリプトは、同じスクリプトを開発テスト、QA、システムテスト、あるいは本番環境でのデプロイに使用できるように、パラメータ化された方法で記述されるべきです。

セクション III のステップ

- マーブルのデプロイメントでは、ロードモジュールをコピーし、ターゲット CICS 環境での変更を有効にする必要があります。
 - CLI コマンドを使用して手動でデプロイします。
 - Deploy gulp タスクを作成し、実装します。
 - デプロイのテスト

Deploy manually

- 前のセクションで Endevor に LNK 要素を生成したとき、Endevor はロード・モジュールを作成しました。これらのロード・モジュールを CICS が使用している適切なデータセットの場所にデプロイし、CICS をリフレッシュして変更をピックアップすることができます。
- ロード・モジュールがデータセットに存在することを確認します。
 - MARBLE エントリーが存在することを確認するために、LOADLIB および DBRMLIB 内のメンバーのリストアップに進むことができます。
 - `zowe files list all-members "PRODUCT.NDVR.MARBLES.MARBLES.D1.LOADLIB"`
 - `zowe files list am "PRODUCT.NDVR.MARBLES.MARBLES.D1.DBRMLIB"`

手動でデプロイ

- LOADLIB モジュールと DBRMLIB モジュールを任意の場所にコピーします。
 - Zowe を使用してロードモジュールをコピーする方法は複数あります。今回のワークショップでは、ジョブを利用してエレメントを移動させます。
 - 今回のワークショップでは、ジョブを利用してエレメントを移動します：
 - `zowe jobs submit data-set "CUST001.MARBLES.JCL(MARBCOPY)" --vasc`

手動でデプロイ

- バインドとグラントを実行する JCL を送信します。
 - `zowe jobs submit data-set "CUST001.MARBLES.JCL(MARBIND)" --view-all-spool-content`
 - この関数は、コマンドを実行し、すべてのジョブ内容を返します。
 - 別のアプローチ
 - `zowe jobs submit data-set "CUST001.MARBLES.JCL(MARBIND)"`
 - E 返されたジョブ ID の例: JOBXXXXX
 - `zowe jobs view job-status-by-jobid JOBXXXXX`
 - リターンコード = CC 0004 を確認する

手動でデプロイ

- CICS 上でトランザクションの変更を有効にする
 - CICS プラグインを使用して CICS プログラムを更新します。
 - では、CICS プログラムを更新してみましょう。
 - リフレッシュが必要なプログラムは、Marble という名前です。
 - `zowe cics refresh program MARBLE01`

手動でのテスト

- 手動でコマンドを実行します:
 - `zowe console issue command "F CICSTRN1,MB01 CRE RED 1 2" --console-name CUST001`
 - SUCCESS** のメッセージは表示されましたか?
 - データベースのチェック
 - `zowe jobs submit ds "CUST001.MARBLES.JCL(MARBDB2)" --vasc`
 - データベースに数量とコストのマーブルが含まれていることを確認します



復習 - 何を学んだか？

セクション III:

オートメーション

ビルドを自動化する



コードビルドを自動化する - はじめに

- CLI を使用して Endeavor でコードの変更と生成を正常に実行したので、次はこれらの手順を自動化します。
- CLI コマンドは、ローカル マシンから繰り返し実行できるスクリプトに埋め込むことができます。
- これらの同じスクリプトは、Jenkins などの CI/CD ツールからも呼び出すことができます
- タスク ランナーは、自動化スクリプトをより簡単に整理して操作する方法です。
- このセクションでは、Gulp と呼ばれる JavaScript ベースの Task Runner を使用します。タスク ランナーは、Jenkins などの CI/CD ツールからも呼び出すことができます
- **** 注 ****: タスク ランナーは、スクリプト上の抽象レイヤーです。それらは役に立ちますが、必須ではありません。

セクション III の手順

- CLI を使用してコードを変更し、Endevor で生成を実行したので、次のことを自動化します：
 - 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.

自動化の開始

- コードのダウンロードや準備など、いくつかの機能を実行しました。
 - コードは /projects にあります。
 - 前に gulp コマンドを実行したときに、そのコードを使用しました。

Gulp でビルド タスクを作成する

- 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

再利用可能なコード - config.json

- 設定ファイルを使用することで、スクリプトはそのままに、差分として変数を渡すことができます。
- 以下は、設定値を含む config.json というファイルです。
- スクリプトに値をハードコードする代わりに、スクリプトはこれらの値を読み込むことができます。
- 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"  
}
```

Gulp で Build タスクを作成する

```
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.

Gulp で Build タスクを作成する

```
/**
 * 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
 - 指定されたディレクトリに出力を書き込みます。
 - 成功した場合、最後のコールバックでコードの実行を継続します。

Gulp で Build タスクを作成する

- 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`

Build-Cobol と Build-LNK を組み合わせる

- 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`

完成したビルド シーケンス コマンド

- 完了

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




Section IV:

デプロイを自動化する

Gulp Deploy タスクの作成と実装

- gulp build タスクの作成と同様に、今度は変更をデプロイするための gulp タスクを作成します。
 - コピータスクを確認する
 - Review the `bind-n-grant` task
 - Review the `cics-refresh` task
 - Combine individual deploy tasks into one `deploy` task.

Gulp Deploy タスクの作成と実装

```
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

Gulp Deploy タスクの作成と実装

```
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

Gulp Deploy タスクの作成と実装

```
/**
 * 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

Gulp Deploy タスクの作成と実装

- 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

デプロイシーケンスコマンドの完了

- 完了

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




レビュー

ここまでで学んだことは？

セクション V:

テストの自動化

自動化されたテストとは？

- アプリケーションは期待されたとおりの動作をします。コードを変更するとき、開発者は期待される振る舞いを何度もテストするのが普通です。これには時間がかかり、退屈することもあります。開発者は、より多くのコードを書くことに集中できるように、これらの繰り返しテストを自動化するスクリプトを書くようになりました。
- 自動テストの種類
 - ユニットテスト
 - 統合テスト
 - システムテスト
 - パフォーマンステスト

自動化されたテストの種類

- ユニットテスト
 - フロントエンドのコード (JavaScript)
 - ウェブコード (Java)
 - バックエンドコード (COBOL/CICS)
- 統合テスト
 - UI
 - ウェブサーバー
 - CICS
 - Db2
- システムテスト
- パフォーマンステスト

テストの自動化

- テストの自動化は、2つの選択肢に集約されます。
 - 選択した言語でスクリプトを書き、手動で管理する。
 - あなたのニーズとスキルに合ったスクリプトフレームワークを選択します。
- 人気のあるテストフレームワークです。
- 人気のあるテストフレームワーク：
 - Mocha
 - Robot
 - Jest
 - Jasmine
 - JMeter

CICS トランザクションの自動テスト

CICS 手動テスト

- MABRLE が正常に作成されたことを確認するために、デプロイされたプログラムを手動でテストします。
 - 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

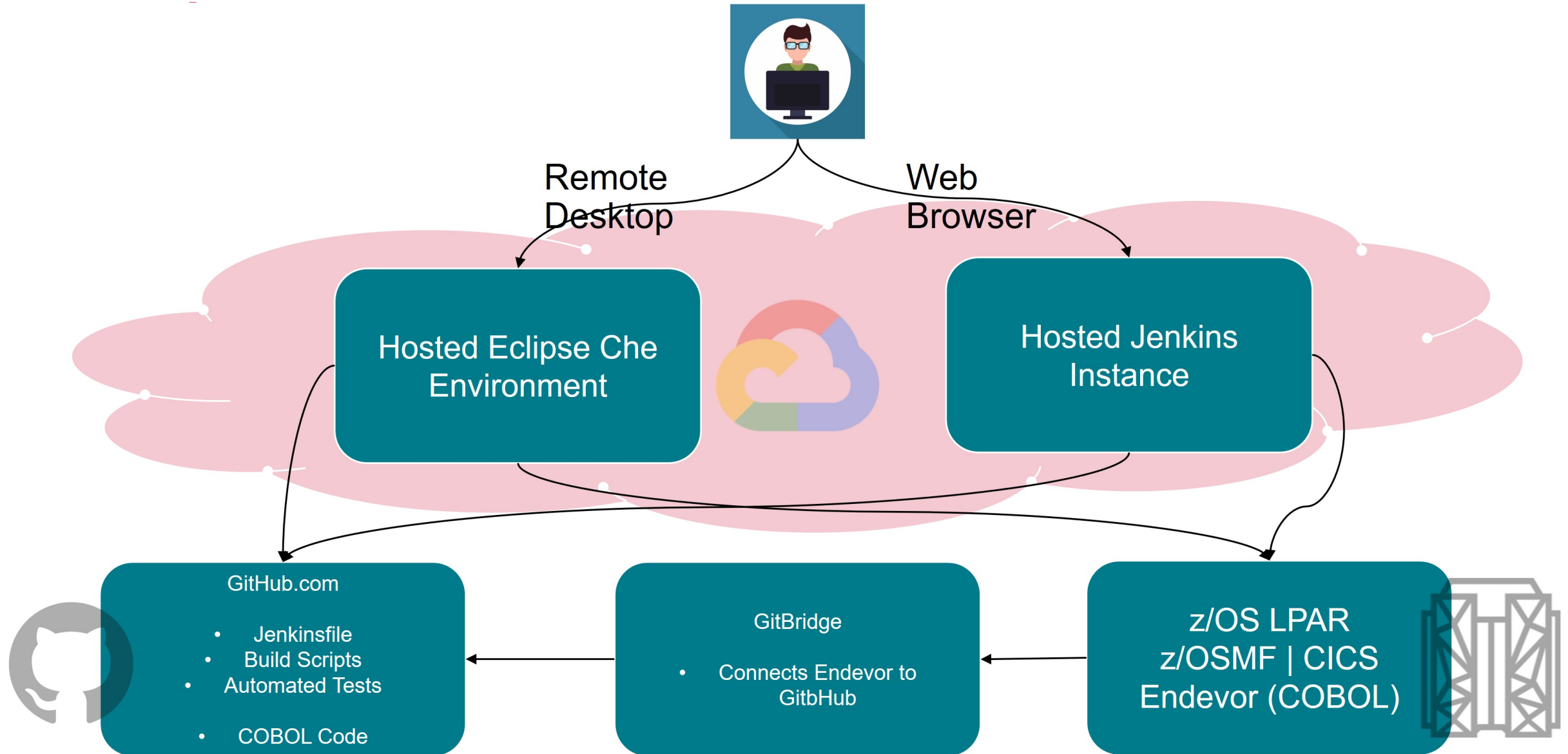
What have we learned?



Section VI:

Add Code Build to Continuous Integration

Workshop Environment



継続的インテグレーション - はじめに

- タスクランナーは、個々の開発者が自動化スクリプトを簡単に実行し、平凡なタスクで時間を浪費するのを防ぐのに役立ちます。これをさらに一歩進めて、チームの共有リポジトリでコード変更が発生したときに、CI オーケストレーターがこれらのタスクの多くを実行できるようにすることができます。
 - コード変更後に実行する必要があるタスクを共有レベルで特定する。
 - これらのタスクのために CI オーケストレーターにステージを作成する。
 - 自動化されたスクリプトをこれらのステージから呼び出す
- ****注**:** 注 **: CI からの自動化に加えて、コード変更時の共有コードリポジトリからの CI プロセスのトリガーも自動化したい場合が多いでしょう。

第 VI 節のステップ

- 我々の場合、ビルドステップは、Endevor でコード変更が行われた時点で CI から実行できる一般的なものである。Jenkins のパイプラインに追加してみましょう。
- 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 の概要用語

- パイプライン - 全体的なパイプライン構造を定義します
- エージェント - コードが実行される場所を定義します
- 環境 - 他のコマンドで使用する環境変数を定義します
- ステージ - ステージの開始を定義します
- ステージ - 各ステージを定義します
- ポスト - ステージ セクションの実行後に実行

ジェンキンスファイル

- トラブルシューティングを容易にし、すべてがセットアップされていることを確認するために、ローカル セットアップでは次のことを行います。
 - node、npm、zowe、zowe プラグインのバージョンを確認します
 - gulp-cli をインストールします
 - npm の依存関係をインストールします
- これにより、すべてのコンポーネントが利用可能になります。
- 次に、ダミーのプロファイルを作成するため、コードがよりクリーンになります。これらのプロファイルは、実行が終了すると破棄されます。これにより、よりクリーンなコードが提供されますが、ユーザー名、パスワード、およびホスト名が難読化されるため、セキュリティは維持されます。

```
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>
- Zowe コマンドを実行する際、使用するオプション値を決定する優先順位は以下の通りです。
 - コマンドライン引数
 - 環境変数
 - プロファイルの設定
 - デフォルトバリュー

Jenkinsfile の更新

- `withCredentials` ブロック内に次のコマンドを追加して、ビルド ステージの一部として作成した gulp ビルド タスクを実行するよう Jenkins に指示します：
 - `sh 'gulp build'`

デプロイ用に Jenkinsfile を強化する

- マーブルの展開では、ロード モジュールをコピーし、ターゲットの CICS 環境で変更を有効にする必要があります。
 - CLI コマンドを使用して手動でデプロイする
 - Deploy gulp タスクを作成して実装する
 - Deploy Jenkins ステージを作成して実装する

Deploy Jenkins ステージを作成して実装する - ステップ 4

- Deploy ステージの実装
 - withCredentials ブロックのコメントを外します
 - 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.

セクション VII:

自動テストを継続的インテグレーションに追加する

CI のテスト手順 - はじめに

- テストは、一般的に自動化されるもう 1 つのステップです。自動化されたテストのスイートを作成したら、それらを CI プロセスに統合することができます。CI を実行するたびにこれらのテストを実行することは、テストに合格することでコードの変更と自動化に対する信頼が高まるため、非常に重要です。
 - 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.
 - コードがチェックイン、ビルド、デプロイされた後、これらのテストを呼び出すために Jenkins でステージを作成します。
- **** 注 ****: 自動テストでテストされるコードのメトリクスを取得するために、コードカバレッジツールを検討することを検討してください。

テストステージの作成

- Mocha を使って自動テストをすでに構築しています。それを呼び出すために、 Jenkins でテストステージを実装してみましょう。
 - Test スクリプトの呼び出しと、 environment ディレクティブの下にある TEST ENV var を削除します。
 - Uncomment withCredentials block in the 'test' stage
 - Call the `npm test` command that runs your test suite.

パイプラインの実行

- テストステージを追加した後、変更をプッシュしてパイプラインを手動で実行し、テストしてください。
- 必要なのは
 - コミットを行い、GitHub にコードをプッシュします。
 - Jenkins にログインし、プロジェクトをビルドします。
 - 問題が発生した場合はデバッグを行います。必要であれば、ファシリテーターに指導を仰いでください。

変更をコミットして GitHub にプッシュする。

- コミットする前に、変更したファイルを確認すること：
 - `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.



Test ステージの出力をファシリテーターと共有する

- Jenkins でパイプラインを作成し、Test ステージで実行した後は、これをファシリテーターと共有することになります。
- Test ステージでの実行が成功した URL を検索してください。



Review - What have we learned?

セクション X (オプション):

Jenkins におけるアーカイブアーティファクト

アーカイブアーティファクト

- 監査のために、パイプラインの実行ごとにビルド / テストの結果をアーカイブしたい場合があります。
- Uncomment the post section of the `Jenkinsfile`
- Commit and Push Code to GitHub
 - `git commit -a -m "Updating"`
 - `git push`
- Jenkins にログインし、プロジェクトをビルドします。
 - 問題が発生した場合はデバッグしてください。必要であれば、ファシリテーターに指導を仰ぐ。

まとめ

- Zowe ワークショップを修了された方、おめでとうございます。今日は以下のトピックについて学びました。
 - テストの自動化の概念。
 - CICS トランザクションの統合テストを構築するための Mocha の使用。
 - Jenkins で Test ステージを実装し、自動テストを呼び出す。
 - コードの変更を行い、自動化されたビルド、デプロイ、およびテストプロセスを楽しむ。



復習 - 何を学んだか？

ありがとうございました