How to pass token to postman scripts locally & from Jenkins

# Requirements:

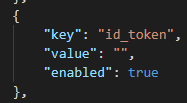
We are moving away from basic authentication (i.e., username + password) mentioned in “02 Multi\_Enviroment” to id\_token authentications (based on OpenID Connect). We need to make sure our postman scripts can still run locally and run in Jenkins CICD.

Sub-Tasks:

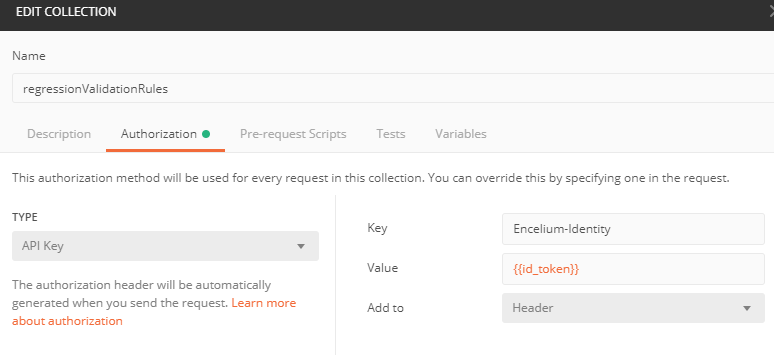
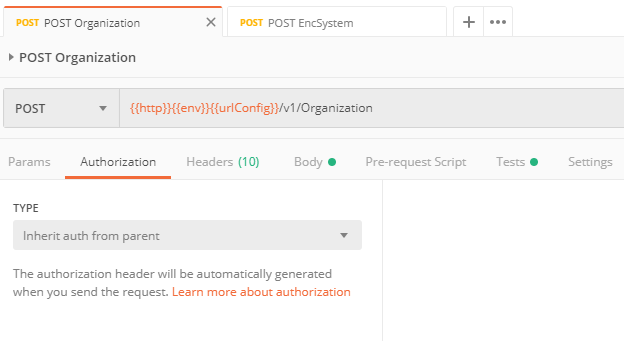
1. Modify postman script to switch from basic authentication to add token to each http call’s header.
2. Create a small app locally to return the id\_token, and pass it to npm/newman run.
3. Create credentials for different environments in Jenkins, use it to generate token, then pass it to npm/newman run.

# Modify postman script to switch from basic authentication to add token to each http call’s header.

Remove “username” and “password” from globalVariables.json (located at “03 Auth\_Token\bim\_resource\_api\qa\postman”), add “id\_token” there.



Load all postman collection json files, such as “regressionValidationRules.json” and “sanityResourceResources.json”, into Postman GUI. Edit the collections: switching from TYPE “Basic Auth”, to TYPE “API Key”. As per our developer, the key-value pair is “Encelium-Identity”, and the value that will be stored in global variable {{id\_token}}. Make sure all the requests under the collection are default to using “Inherit auth from parent” under Authorization – TYPE section. Save the changes and export your collections to json files.

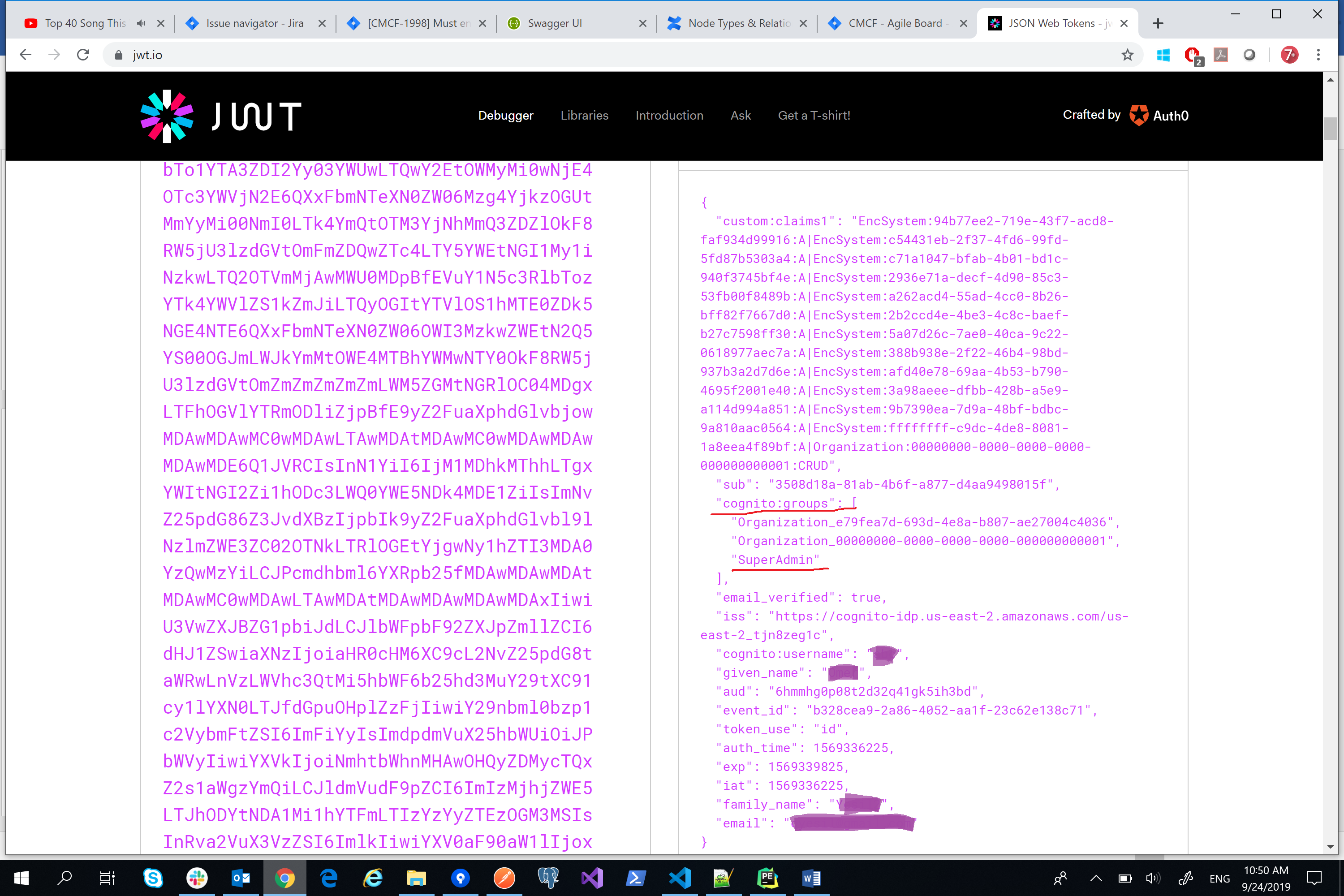
 

# Create a small app locally to return the id\_token and pass it to npm/newman run.

Now the question is how we should dynamically request the token (which expires in an hour) then pass it to the postman global variable. Our developer created a “cognitoauthenticator.dll” file (located in “bim\_resource\_api\qa\postman\tools\auth”) which if you input the correct AWS userPoolId, clientId, username, and password, then the dll will return you the id\_token.

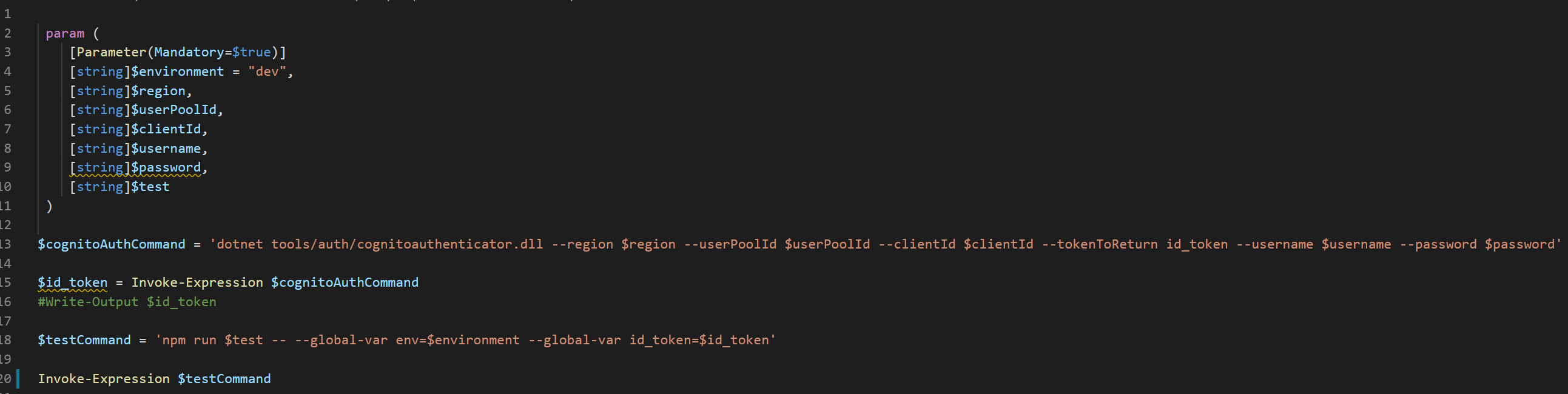
dotnet cognitoauthenticator.dll --region us-east-2 --userPoolId {ASK\_ME} --clientId {ASK\_ME} --tokenToReturn id\_token --username {ASK\_ME} --password {ASK\_ME}

The id\_token is a very long string (see left hand side below), which you can decrypt at jwt.io for the human readable information that the id\_token carries (see right hand side below). Note, in the above command, you need to use a user which belongs to the “SuperAdmin” group (see the red lines below), since by our architecture design, only SuperAdmin users can create organizations, and most of our postman scripts create organization at the beginning of the script, and delete everything under the organization and the organization itself at the end of the script.



We can get the id\_token now, but how would we pass the id\_token from the dotnet dll return to postman/newman/npm to overwrite the value of the newly created global variable {{id\_token}}? The answer is to use a batch file to pipe the return to npm run (this is how we run our postman collections, see “package.json” in “bim\_resource\_api\qa\postman”). For example, I want to run “sanityResourceResources.json”, then I can open power shell, and navigate to the folder that contains “package.json”, then use command “npm run sanity\_resources” to run it. Below is a copy of the “package.json”.



First, copy the “bim\_resource\_api\qa\postman\tools” folder over, which contains a bunch of dlls including the “cognitoauthenticator.dll” file. Second, create the “vsCodeRun.ps1” script as below: 

Line 13 saves a string to a variable $cognitoAuthCommand, then line 15 uses Invoke-Expression to accepts a string to be executed as code, which return is saved to $id\_token. Line 18 saves the “npm run blablabla” string to a variable $testCommand, which get executed in line 20. From the terminal of vsCode IDE, navigate to the folder that “vsCodeRun.ps1” is located, execute the following command.

./vsCodeRun.ps1 -environment **dev** -test **sanity\_resources** -region us-east-2 -userPoolId {ASK\_ME} -clientId {ASK\_ME} -username **{ASK\_ME}** -password **{ASK\_ME}**

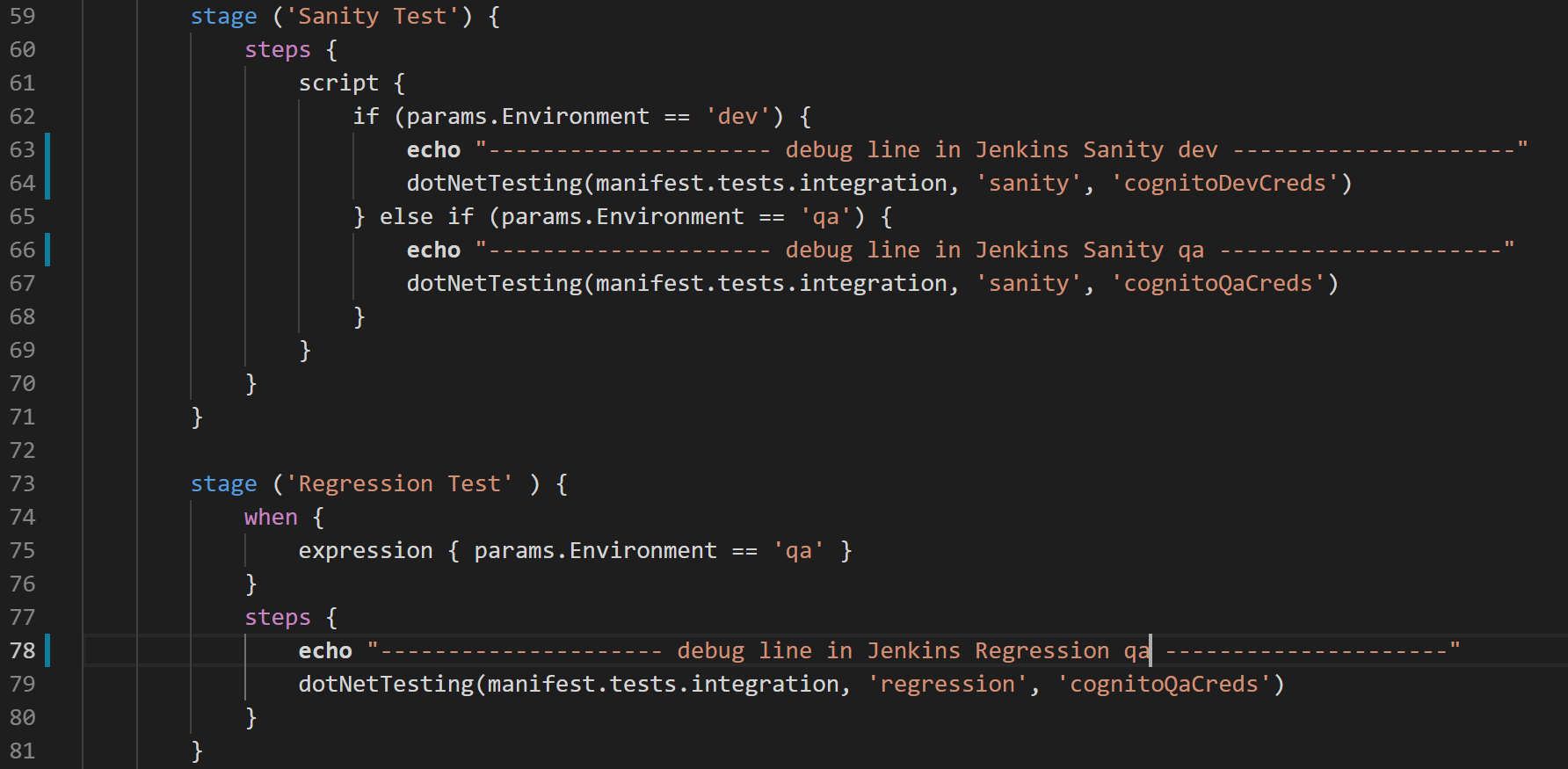
To test different postman collections, you can replace “**sanity\_resources**” to any script name that you defined in the “package.json” file, such as “regression\_validation”. To test different deployment environment, you can replace “**dev**” to any environment that you want to test, such as “qa”, “staging”, or “production”. Please also remember to replace the username and password with the SuperAdmin username and password within the environment.

# Create credentials for different environments in Jenkins, use it to generate token, then pass it to npm/newman run.

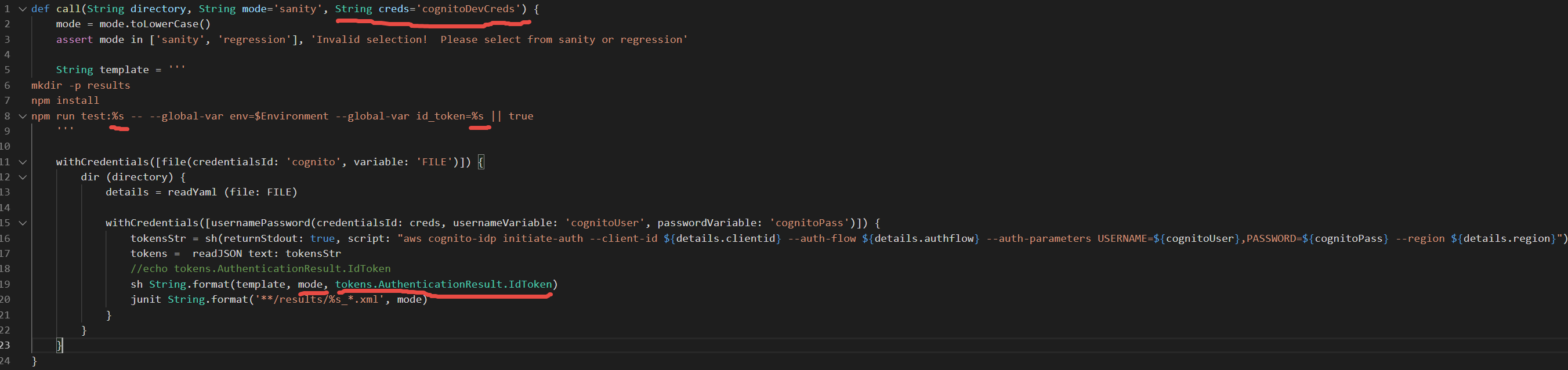
Now the question is how we should pass the token (which expires in an hour) from Jenkins to the postman global variable. First, create credentials in Jenkins GUI. Our devops created the “congnito” credential, which is a yaml file contains AWS cognito -region -userPoolId -clientId. I created “congnitoDevCreds” and “congnitoQaCreds” credentials which contains dev and qa environments’ SuperAdmin users’ username and password (Currently we only have dev and qa environments).



Second, modify the “Deploy.Jenkinsfile” located in “bim\_resource\_api” folder as below. Make sure the ‘cognitoDevCreds’ and ‘cognitoQAcreds’ are passed in at the right places (line 64, 66, and 79).



Line 64, 66, and 79 of the “Deploy.Jenkinsfile” calls “dotNetTesting” function. Third, modify the “dotNetTesting.groovy” file located at “jenkins\_shared\_libraries\vars” as below. Note line 1 indicates that the “dotNetTesting” function takes 3 parameters, this is why ‘cognitoDevCreds’ and ‘cognitoQAcreds’ are passed in line 64, 66, 79 above in the “Deploy.Jenkinsfile”.



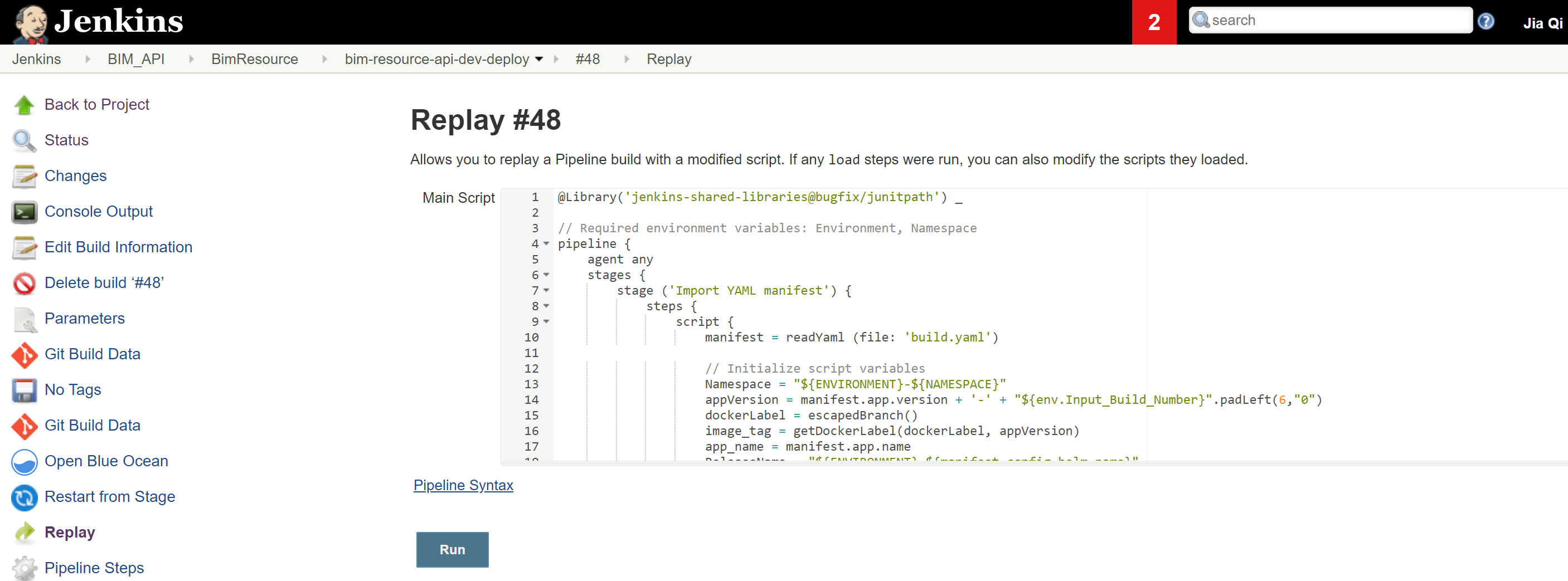
Line 11 of the above “dotNetTesting.groovy” file takes in the “congnito” credential from Jenkins GUI, which is a yaml file contains ${details.clientid}, ${details.authflow}, ${details.region} which are used in line 16. Line 15 of the “dotNetTesting.groovy” file takes in the other 2 credentials that I created in Jenkins GUI, which are either ‘congnitoDevCreds’ or ‘congnitoQaCreds’ depends on which one got passed in from the ‘Deploy.Jenkinsfile’. Line 15 parsed the ‘congnitoDevCreds’ or ‘congnitoQaCreds’ credentials into its username ‘congnitoUser’ and password ‘cognitoPass’, which are used in line 16. The format of “withCredentials” function, please refer to [Jenkins credential binding](https://jenkins.io/doc/pipeline/steps/credentials-binding/).

The return of line 16 contains 3 tokens, including the AccessToken (based on OAuth 2, dedicated to get access to certain AWS services), the IdToken(based on OpenID Connect, which is used by our application, so this is what I need), and the RefreshToken (use this to replace expired token). Line 17 and 18 parsed out the IdToken, which is “tokens.AuthenticationResult.IdToken”. The IdToken is used in line 19. “template” is used from line 5 to line 8, which line 8 takes in 2 %s, i.e., strings, which are “mode” and “tokens.AuthenticationResult.IdToken” (specified in line 19). $Enviroment in line 8 is defined inside different deployment projects inside Jenkins GUI (there are detailed explanation in “02 Multi\_Enviroment” folder).

Basically we get the token from line 16, and execute the npm run with line 8, such as npm run test:sanity -- --global-var env=dev –global-var id\_token=“a crazy long string”, this is very similar to line 13 and 18 of “vsCodeRun.ps1” file.

# Debug in Jenkins CICD:

The “dotNetTesting.groovy” file is located inside the “jenkins\_shared\_libraries” folder, which has its own repository in our Bitbuket. How can I debug the “dotNetTesting.groovy” file? The “bim\_resource\_api” has its own repo too, and it also has corresponding deployment project, such as the one showing below which is called “bim-resource-api-dev-deploy”.



Once your changes of “dotNetTesting.groovy” file are merged to the “jenkins\_shared\_libraries” repo and the changes of “Deploy.Jenkinsfile” are merged to the “bim\_resource\_api” repo, Jenkins will receive a trigger from Bitbuket to start the build project (the CI part, which creates the docker image). Once the build project succeed, Jenkins will auto start its downstream “bim-resource-api-dev-deploy” deployment project.

If the deployment project fails at “Sanity Test” stage (defined in the “Deploy.Jenkinsfile” file). Click on the deployment number, such as #48, then click “Replay” on the left-hand side to debug. Inside the “Main Script” body (see above picture), it is your “Deploy.Jenkinsfile”, you can modify it to the format that you think might fix the problem. The line 1 of “Main Script” defines which “jenkins\_shared\_libraries” branch is in use. Normally it is default to use dev branch (we use dev as the master branch), such as @Library('jenkins-shared-libraries@bugfix/junitpath') \_, you can change it to your branch (such as indicated in the picture above), which you need to check in the changes you think that might fix the problem.