

ExtraCredit – RentTrack Integration

CI/CD Pipeline Configuration

Table of Contents

[1. High-level Infra Setup 3](#_Toc25765923)

[a. Non-Prod and Prod 3](#_Toc25765924)

[2. Continuous Integration and Deployment for Build Credit Module: Basic Overview 4](#_Toc25765925)

[a. Git Branching Strategy 4](#_Toc25765926)

[b. Project Structure 5](#_Toc25765927)

[c. CircleCI Configuration 5](#_Toc25765928)

[i. Basic Config 5](#_Toc25765929)

[ii. SSH Keys for Handling Additional Repositories 6](#_Toc25765930)

[iii. Environment Variables 7](#_Toc25765931)

[d. Heroku Configuration 8](#_Toc25765932)

[3. Web Module: Build and Deployment Configuration 8](#_Toc25765933)

[a. Basic Overview 8](#_Toc25765934)

[b. CircleCI Pipeline Config 8](#_Toc25765935)

[4. Messaging Module: Build and Deployment Configuration 10](#_Toc25765936)

[a. Basic Overview 10](#_Toc25765937)

[b. CircleCI Pipeline Config 10](#_Toc25765938)

[5. Jobs Module: Build and Deployment Configuration 12](#_Toc25765939)

[a. Basic Overview 12](#_Toc25765940)

[b. CircleCI Pipeline Config 12](#_Toc25765941)

[6. QA Automation Script Integration 14](#_Toc25765942)

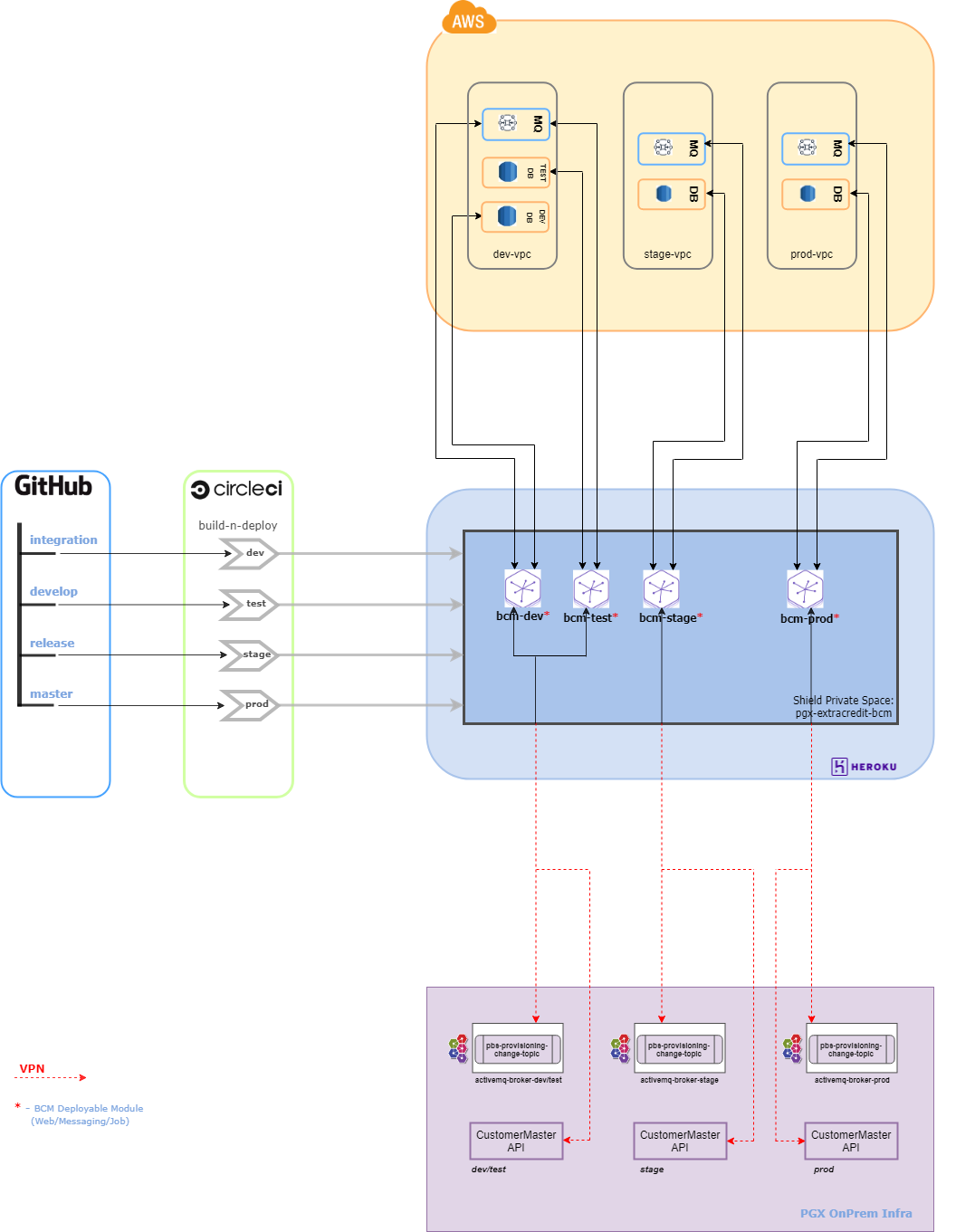
[a. Basic Overview 14](#_Toc25765943)

[b. CircleCI Pipeline Config 14](#_Toc25765944)

# High-level Infra Setup

## Non-Prod and Prod

The diagram given below gives a high-level view of the AWS and Heroku Infra setup in Non-Prod as well as Prod. Connectivity between AWS VPC and Heroku Shield Private Space is established through VPC Peering. Dev and Test Database instances are provisioned in the ‘dev-vpc’, and the existing MQ Broker in ‘dev-vpc’ is used for both Dev and Test region Queue. Stage and Prod database instances are provisioned in ‘stage-vpc’ and ‘prod-vpc’ respectively. The existing stage and prod MQ brokers are used for hosting BCM’s Stage and Prod regions Queues.



# Continuous Integration and Deployment for Build Credit Module: Basic Overview

## Git Branching Strategy

Build Credit Module’s GitHub Repo: <https://github.com/progrexion/extracredit-bcm-services.git>

For BCM, we would be using the following branching strategy to push builds to various environments:

**features/<feature-name>**: Individual developers create feature branches and deploy temporary versions of the app by themselves using this code base for testing in ‘DEV’ environment.

**integration**: Once the changes in a feature branch are tested, they can be merged into the ‘integration’ branch by raising a Pull Request. Code merge/commit happening in this branch would trigger the ‘DEV’ environment app to be built and deployed. Dev team would test this integrated version of the app in ‘DEV’ environment to verify whether the build can be promoted to the next environment.

**develop**: Once the ‘DEV’ environment app has been verified, the changes from ‘integration’ branch can be merged into ‘develop’ branch for triggering Build and Deploy processes for ‘TEST’ environment.

**release**: Once the app has been verified in ‘TEST’ environment, it can be promoted to ‘STAGE’ environment by merging the changes from ‘develop’ branch to ‘release’ branch. This will trigger Build and Deploy processes for ‘STAGE’ environment.

**master**: Once the app has been verified in ‘STAGE’ environment, it can be promoted to ‘PROD’ environment by merging the changes from ‘release’ branch to ‘master’ branch. This will trigger Build and Deploy processes for ‘PROD’ environment.

## Project Structure

The Build Credit Module Maven project follows the following multi-module structure:

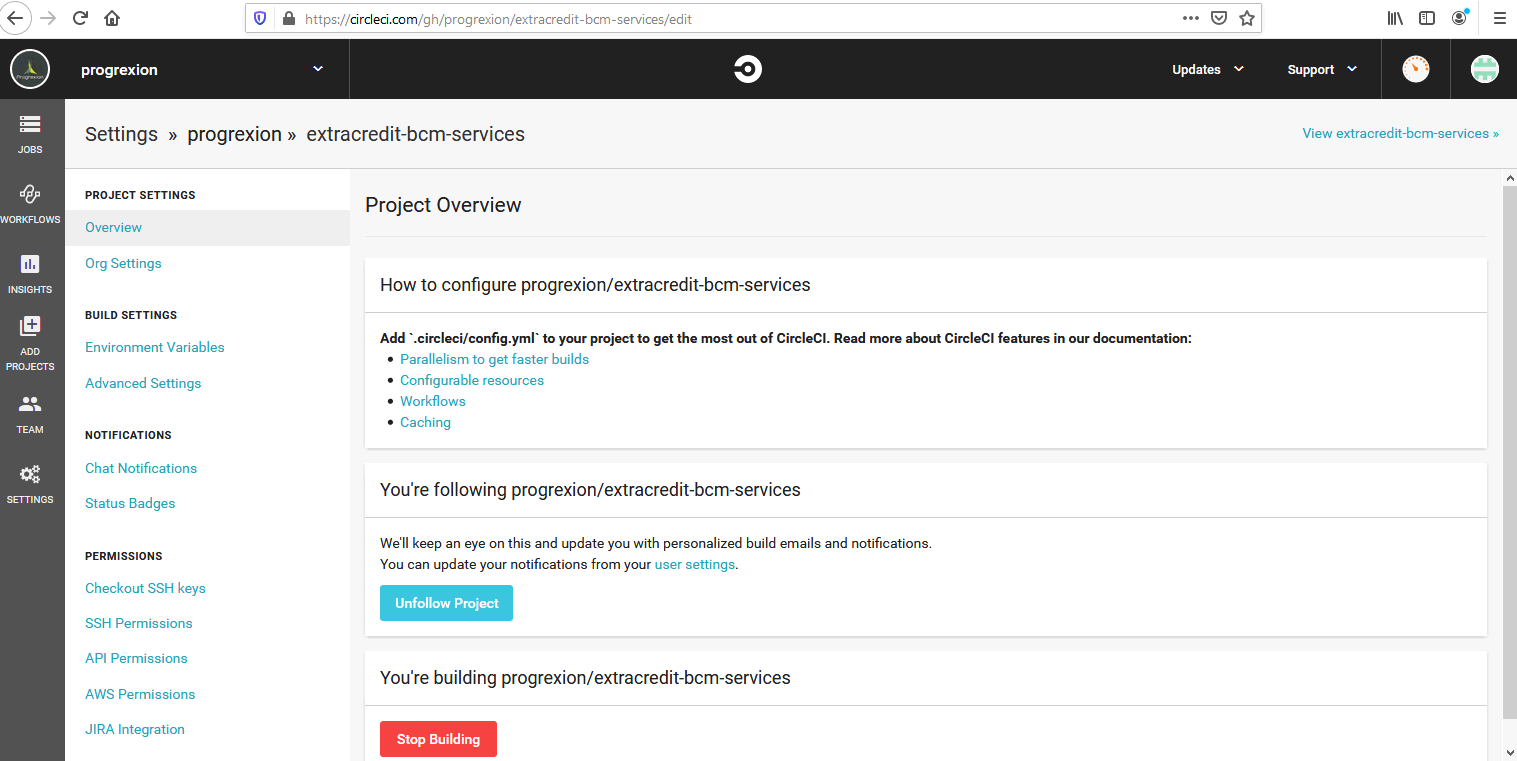
extracredit-bcm-services  
|\_ .circleci/config.yml  
|\_ *Dockerfile*  
|\_ *pom.xml*  
|\_ **bcm-common**  
|\_ **bcm-dto**  
|\_ **bcm-data**  
|\_ **renttrack-plugin**  
|\_ **bcm-service**  
|\_ **bcm-web**  
|\_ **bcm-messaging**  
 |\_*Dockerfile*|\_ **bcm-jobs**  
 |\_*Dockerfile*

GitHub URL: <https://github.com/progrexion/extracredit-bcm-services.git>

## CircleCI Configuration

* 1. Basic Config

Using ‘devops@progrexion’ GitHub user, Build Credit Module project has been added to the CircleCI workspace. The pipeline configuration is defined in the **<project-root>/.circleci/config.yml** file.



* 1. SSH Keys for Handling Additional Repositories

To support working with additional GitHub repositories, RSA Keys need to be generated and updated in GitHub and CircleCI accordingly. For example, for accessing the QA Automation Scripts codebase from within the BCM pipeline, do the following:

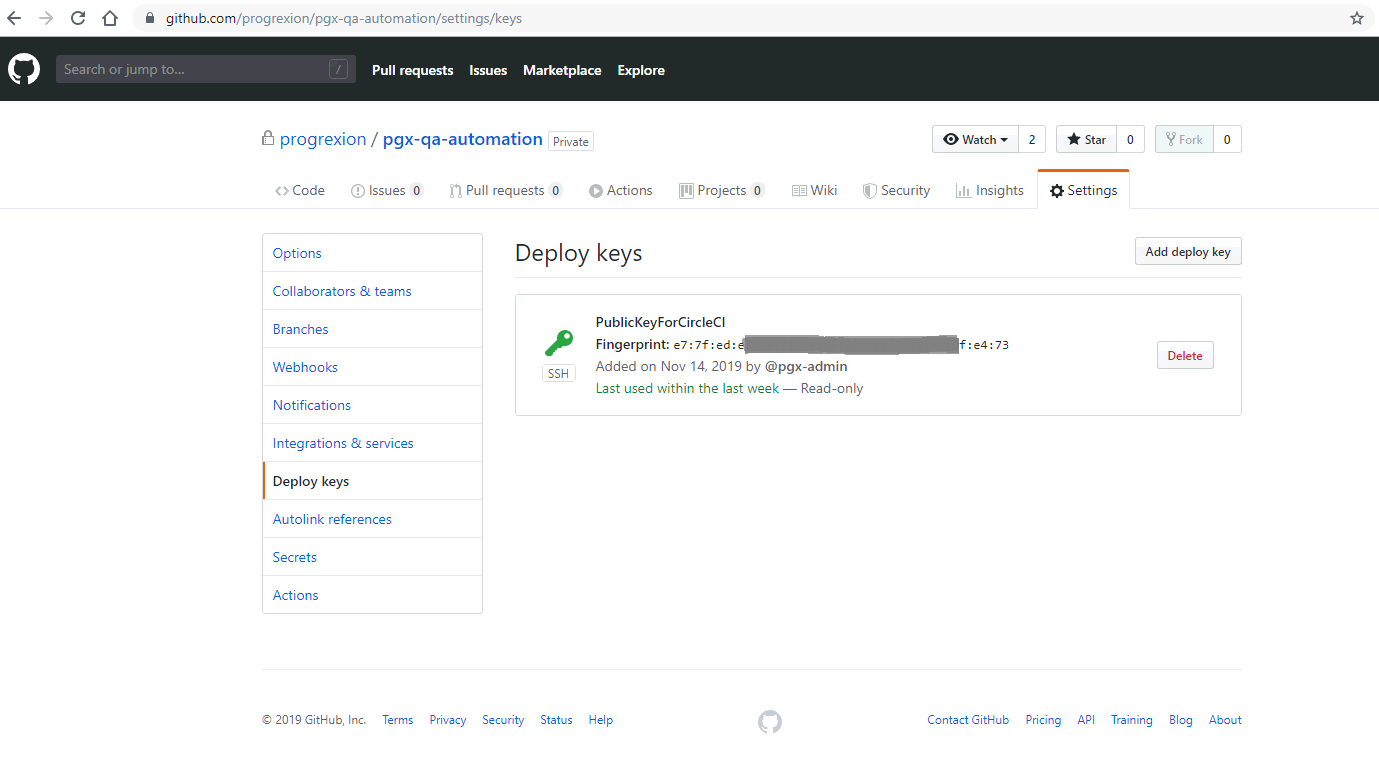
Generate RSA Private-Public key pair using:

ssh-keygen -m pem -t rsa -C “devops@progrexion.com”

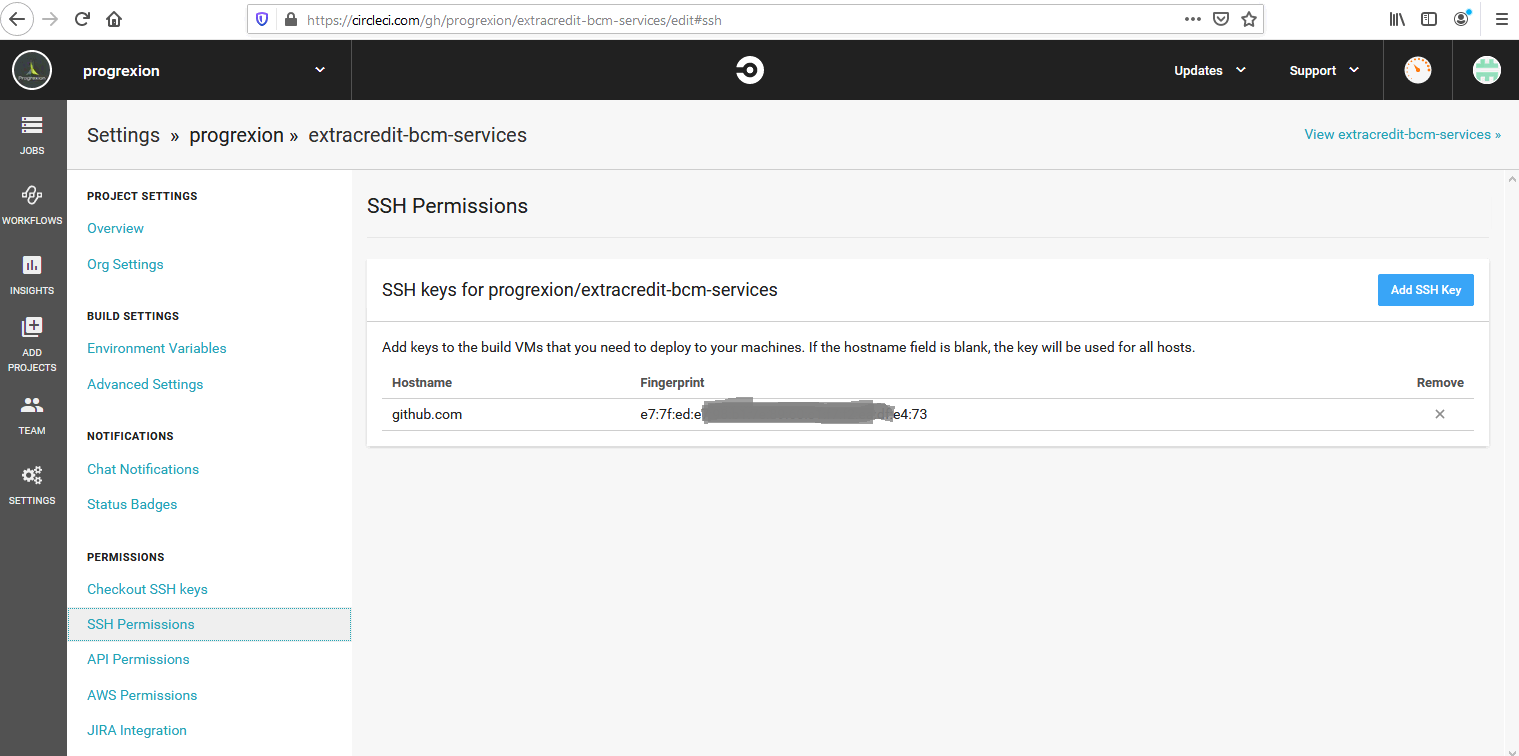
Be sure to use “-m pem” option to force the generation of Private keys with the PEM format that is required for CircleCI (it doesn’t support any other format as of now).

Also, to get the keys working with CircleCI, DO NOT provide a password when generating the keys.

Copy the contents for the public key file (normally this would be: ~/.ssh/id\_rsa.pub), and use it to create a ‘Deploy Key’ for the repository to be accessed. To add the Deploy Key, go to the GitHub repository > Settings > Deploy keys > Add deploy key. Give any ‘Title’ to identify it, and paste the id\_rsa.pub file’s content as the ‘Key’.



Copy the contents for the private key file (normally this would be: ~/.ssh/id\_rsa), and add it to CircleCI and create the ssh key fingerprint for the same. To do this, Go to CircleCI > Open ‘Settings’ page of the Project > SSH Permissions > Add SSH Key > Give ‘github.com’ as the ‘Hostname’, and the id\_rsa file contents as the ‘Private Key’. Once the Private key has been added, a fingerprint for the same will be listed in the SSH Permissions screen. It would be of the format: ‘a2:12:er:t4:23: …’.



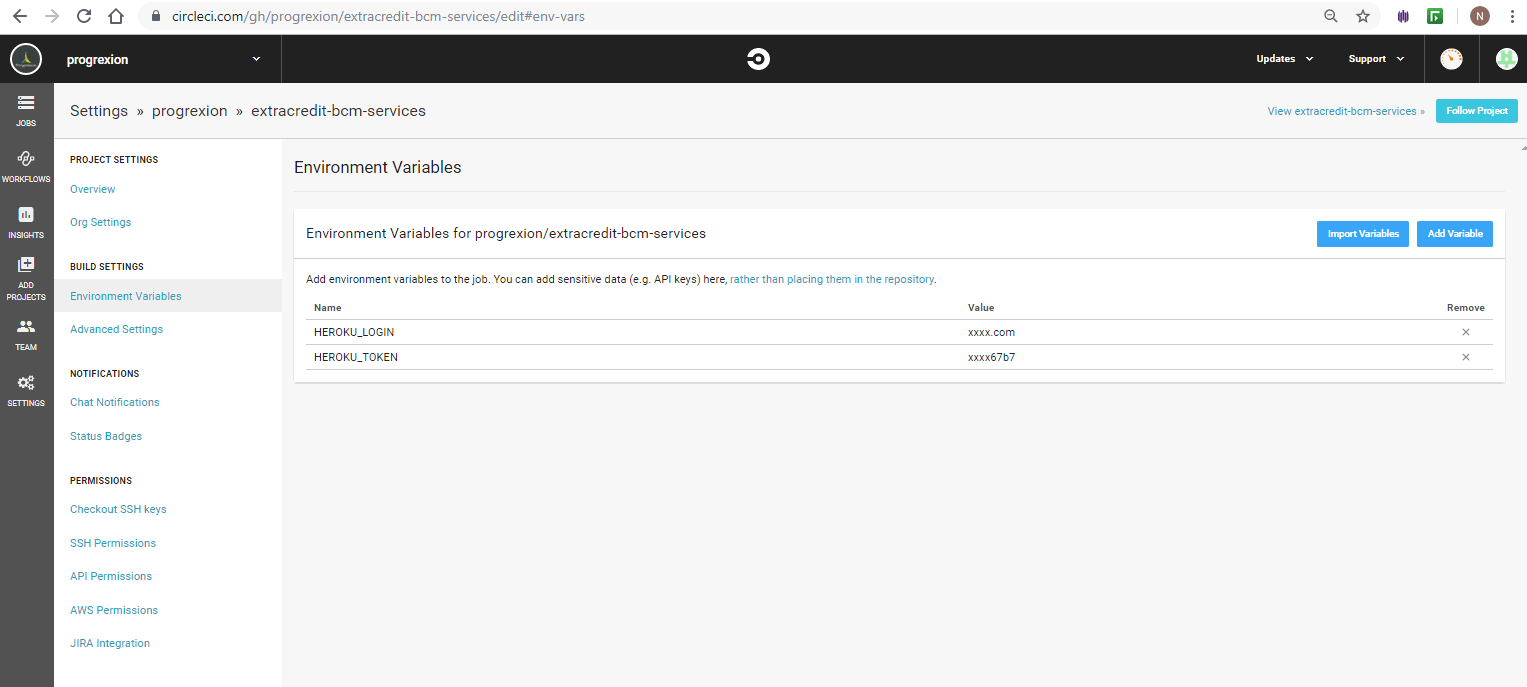
Use this fingerprint in the pipeline config (using **- add\_ssh\_keys** step in a job) to check-out the code from the additional repository.

* 1. Environment Variables

Listed below are the important Environment variables added to the project for accessing Heroku:

**HEROKU\_LOGIN**: This is currently set to ‘devops@progrexion.com’, which is the devops user configured in Heroku.

**HEROKU\_TOKEN**: This is the Heroku API Key for the ‘devops@progrexion.com’ user. To get this key in Heroku, you need to first login, and then navigate to Account Settings > API Key section. It also has a ‘Regenerate API Key…’ option to generate a new API Key. **Make sure to update this CircleCI Environment Variable if the Heroku API Key is regenerated.**



## Heroku Configuration

On the Heroku side, currently, the Apps need to be created once manually before the Pipeline to deploy that particular app is run.

Team Name: **progrexion**

Space Name: **pgx-extracredit-bcm** [Shield Private Space]

Heroku CLI command syntax and example are given below:

Syntax: heroku apps:create **<app-name>** -t **<team-name>** --space=**<space-name>**

Example: heroku apps:create **extracredit-bcm-dev** -t **progrexion** --space=**pgx-extracredit-bcm**

**NOTE:** *Names given to the applications in Heroku should not exceed 30 characters in length. This is a limit/restriction imposed by Heroku.*

Also, the Config-Vars are handled manually. Development team should provide the devops user who handles the deployment/release with the environment/config vars to be set for the application before the release. Once the devops user verifies that a deployment has been successful, he/she can:

* Login to Heroku,
* Navigate to the Application’s Settings page
* Click ‘Reveal Config Vars’ and set/update the config vars accordingly

# Web Module: Build and Deployment Configuration

## Basic Overview

Web Module packages the jar to be used as the main application that serves the various BCM RESTful APIs. Dockerfile present in the project root is used for carrying out the Maven build, copy the jar file produced by the Maven build process, and use that jar to create a Docker container that can then be deployed in Heroku.

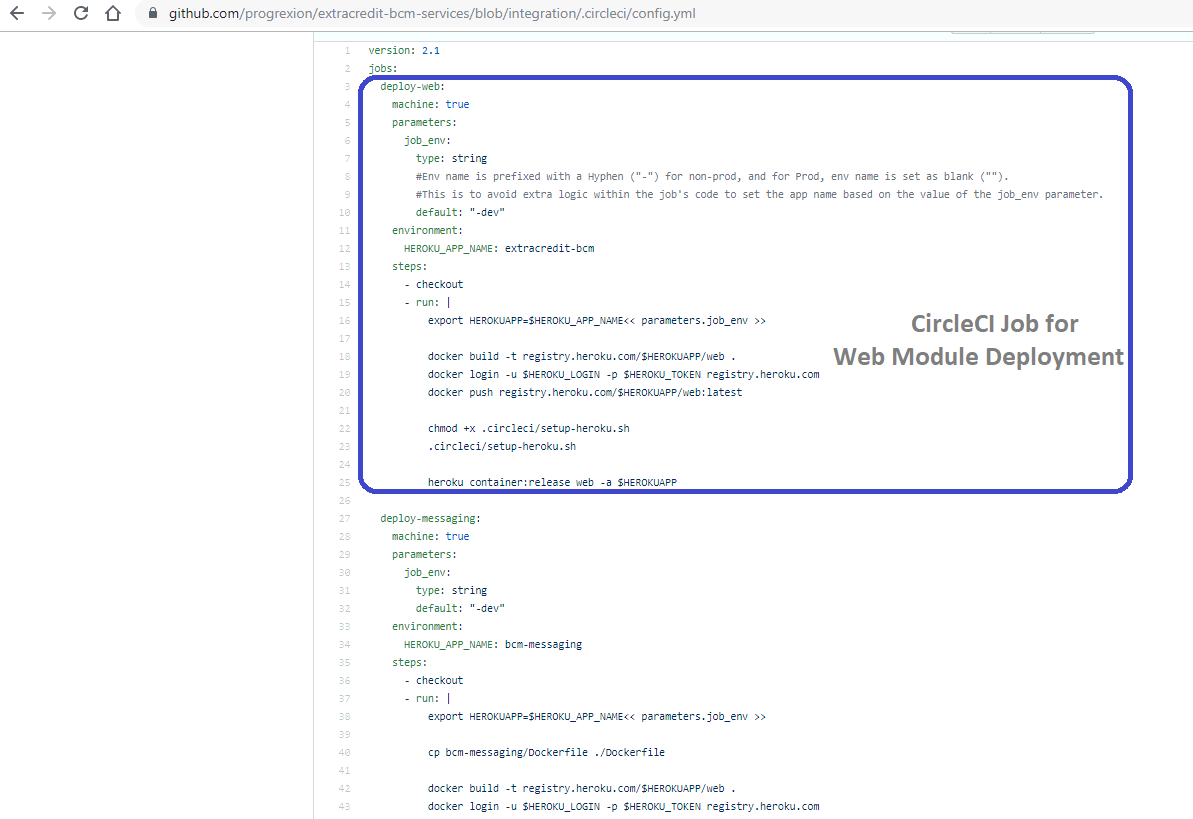
## CircleCI Pipeline Config

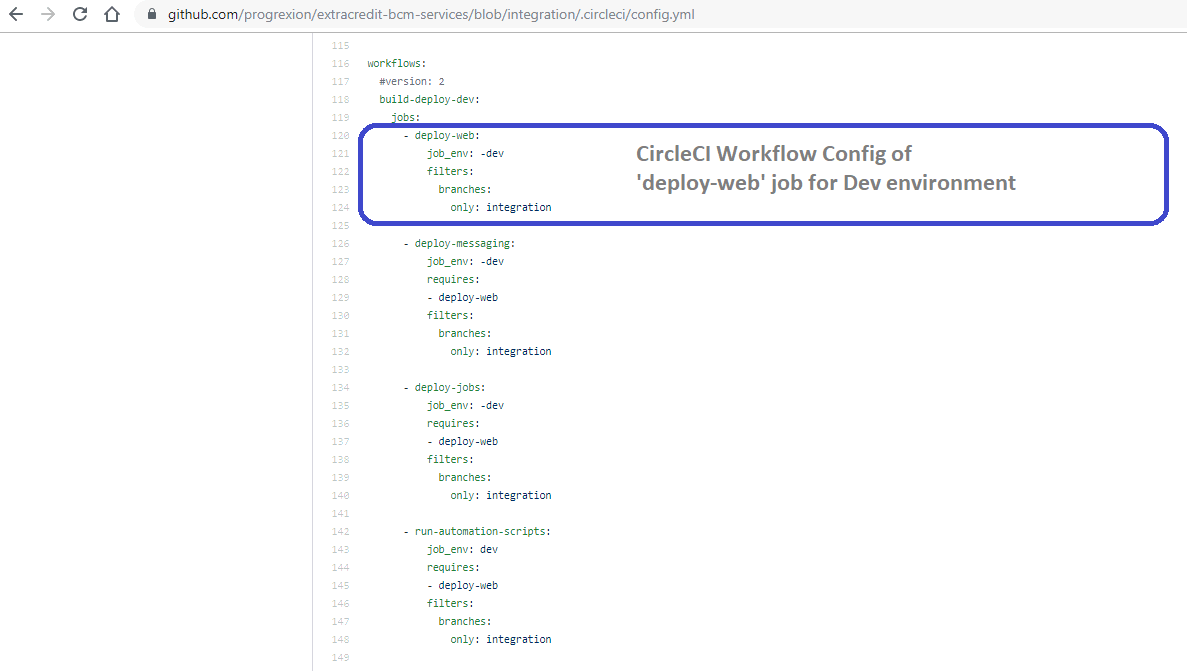
CircleCi Pipeline config is saved in the following file present under project root, which is where all pipeline related configuration is maintained for the entire project:

CircleCI Config File: **.circleci/config.yml**

Name of the Job that handles Web Module deployment: **deploy-web**

Snapshots of configuration:





# Messaging Module: Build and Deployment Configuration

## Basic Overview

Messaging Module packages the jar to be used as an application that serves to connect to Message Brokers related to Build Credit Module. At present, the only use case this module serves is the Listener component that reads messages published in the PBS.ProvisioningChange Topic hosted in an on-prem ActiveMQ Broker instance. This module is required to re-use the other modules present in the BCM parent project to integrate with RentTrack APIs. At present, the jars built from the BCM project are not hosted for distribution. This, coupled with the Dockerized Maven build process, requires the parent project to be built again (as of now) to create a Docker Container with the messaging jar. This is accomplished by the use of a separate Dockerfile kept within the **bcm-messaging** module, which is then used as ***the*** Dockerfile to use in the Docker build context.

## CircleCI Pipeline Config

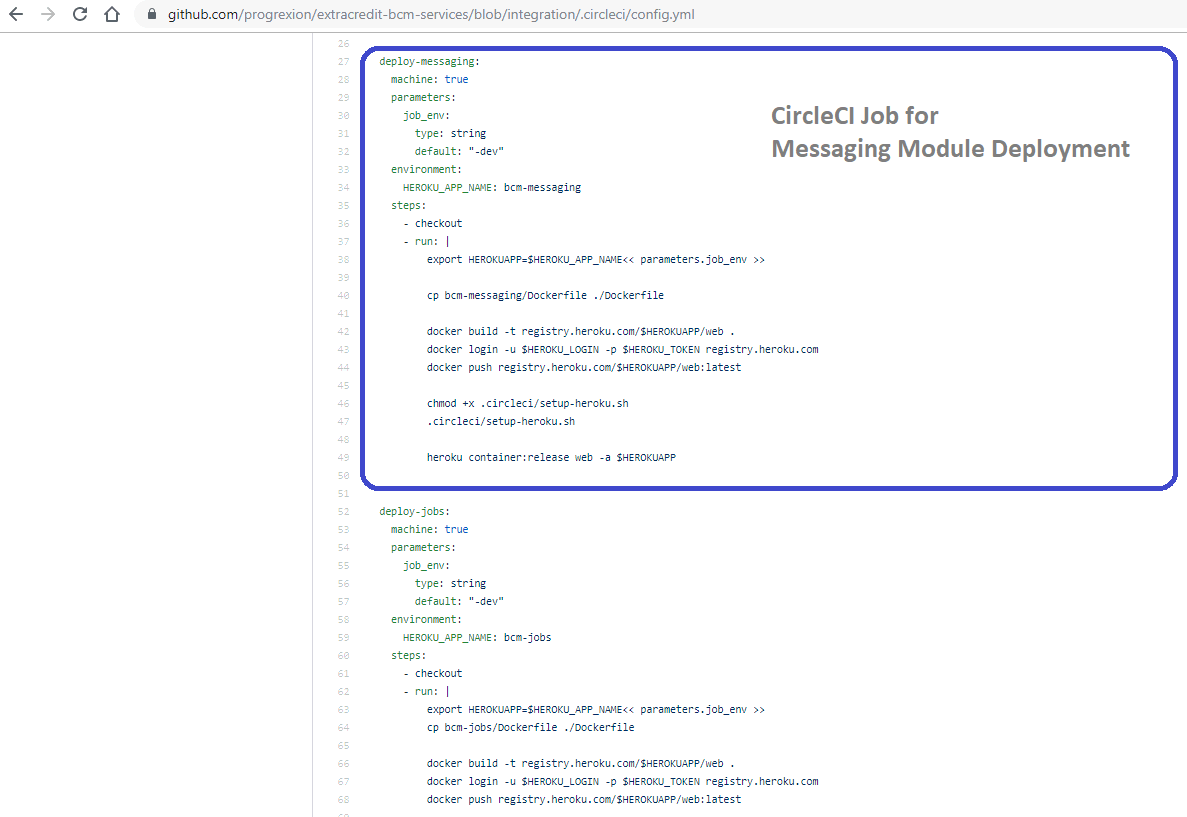
CircleCi Pipeline config is saved in the following file present under project root, which is where all pipeline related configuration is maintained for the entire project:

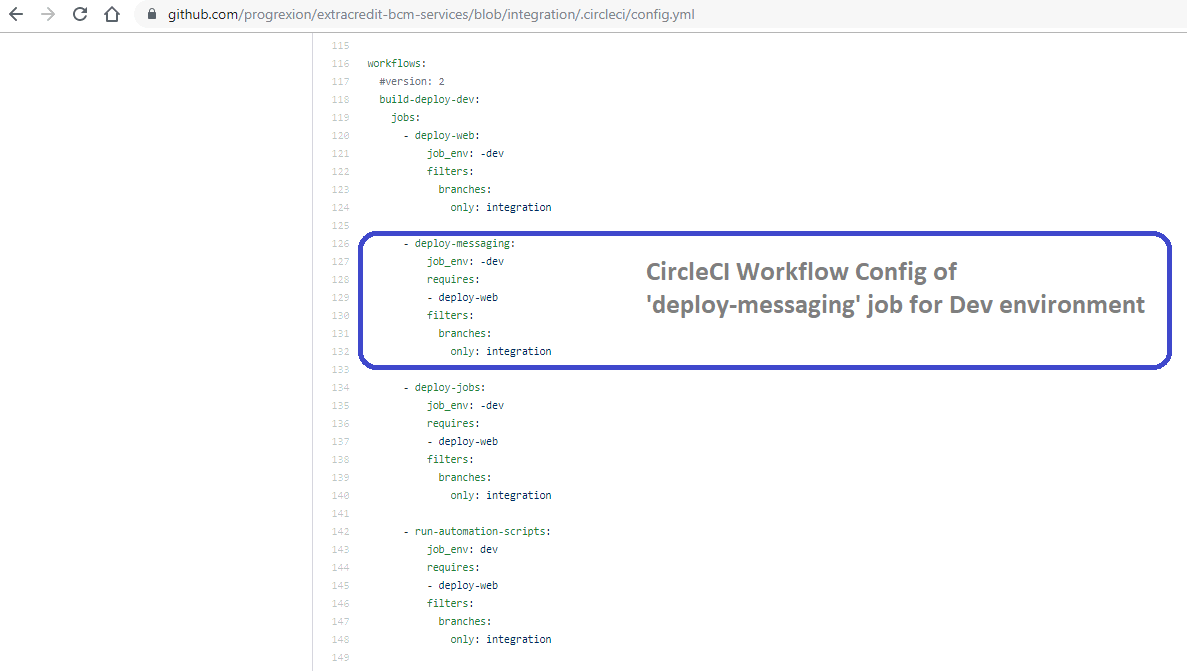
CircleCI Config File: **.circleci/config.yml**

Name of the Job that handles Messaging Module deployment: **deploy-messaging**

In the Workflow configuration, the job is invoked with the ‘deploy-web’ job as a dependency to ensure that the Web module deployment is completed first. This is not mandatory, though if we are planning to run the junit tests when running maven package/install step for the Parent project as part of the Web Module’s build and deploy pipeline phase, and have the Web Module deployment fail in case of any errors, we can avoid running the tests again to save some time when building the Parent project during Messaging Module’s build and deploy pipeline phase. This way, by making this job dependent on the ‘deploy-web’ job ensures that, the job is executed only when there are no build errors present in the project.

Snapshots of configuration:





# Jobs Module: Build and Deployment Configuration

## Basic Overview

Jobs Module packages the jar to be used as an application that serves RESTful endpoints, which would be invoked by scheduled Control-M jobs, to execute predefined tasks/processes related to Build Credit Module. Build and Deploy configuration follows the same approach used by the Messaging module, by making use of a separate Dockerfile kept within the **bcm-jobs** module, which is then used as ***the*** Dockerfile to use in the Docker build context.

## CircleCI Pipeline Config

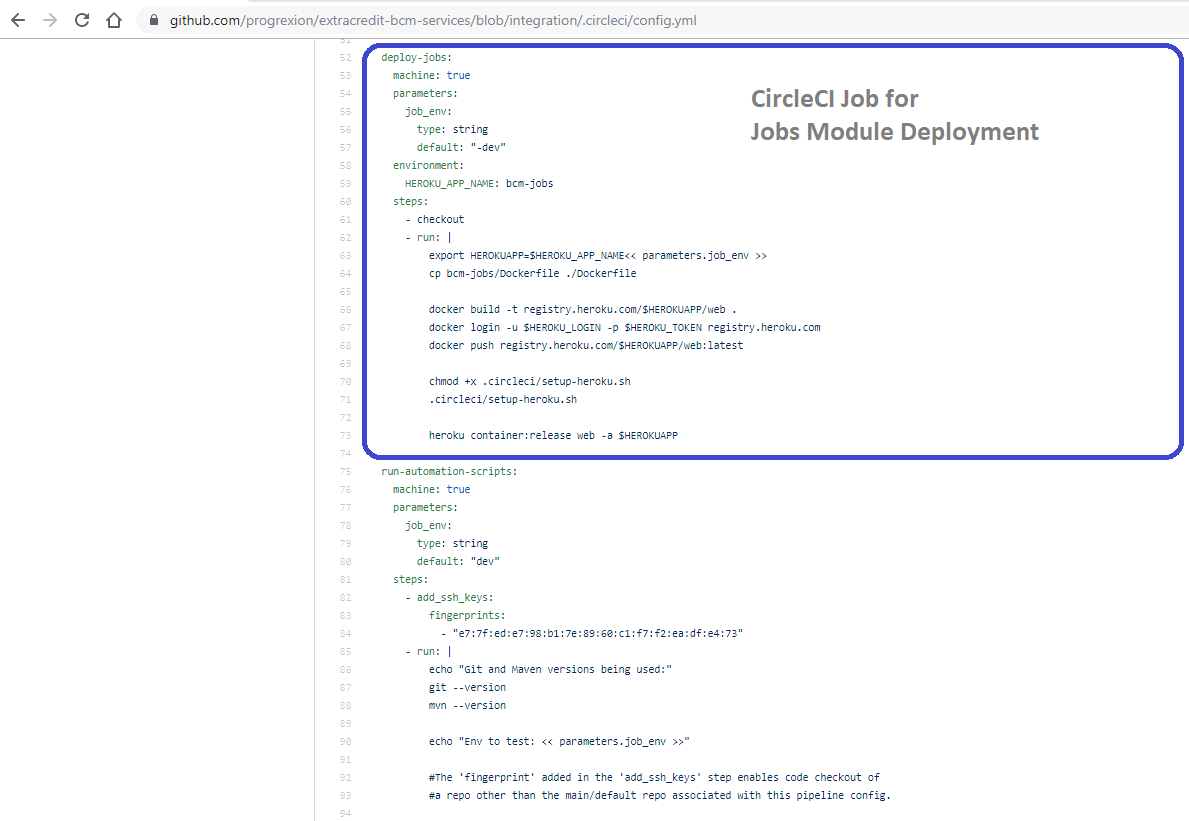
CircleCi Pipeline config is saved in the following file present under project root, which is where all pipeline related configuration is maintained for the entire project:

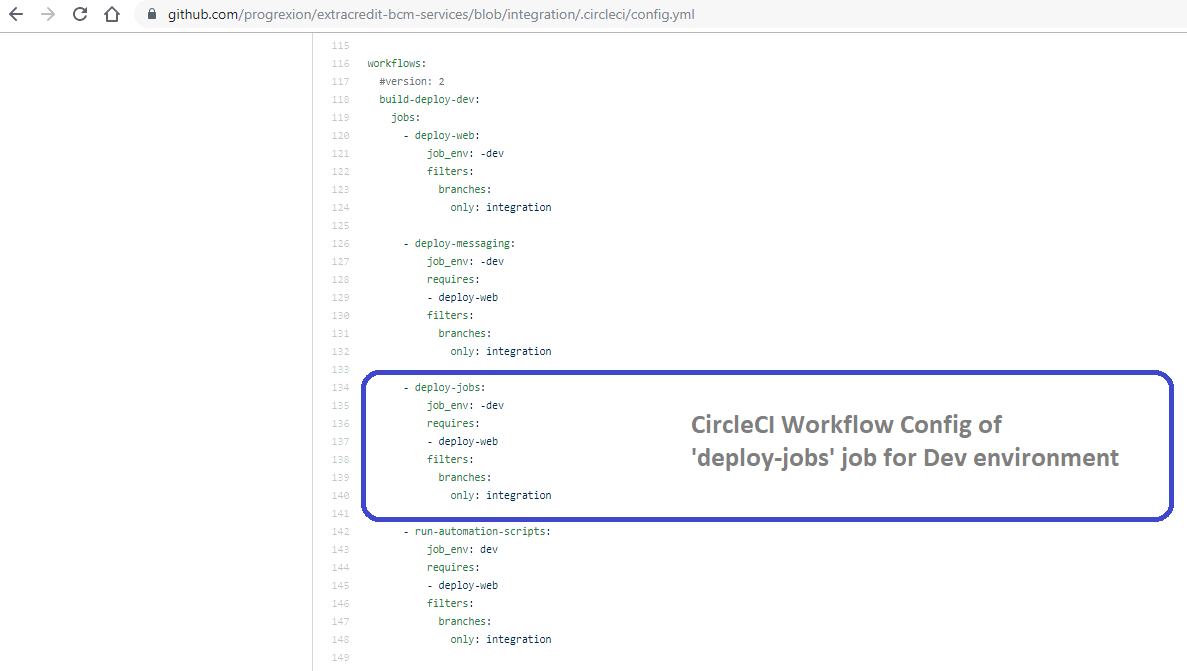
CircleCI Config File: **.circleci/config.yml**

Name of the Job that handles Messaging Module deployment: **deploy-jobs**

In the Workflow configuration, the job is invoked with the ‘deploy-web’ job as a dependency to ensure that the Web module deployment is completed first. This is not mandatory, though if we are planning to run the junit tests when running maven package/install step for the Parent project as part of the Web Module’s build and deploy pipeline phase, and have the Web Module deployment fail in case of any errors, we can avoid running the tests again to save some time when building the Parent project during Jobs Module’s build and deploy pipeline phase. This way, by making this job dependent on the ‘deploy-web’ job ensures that, the job is executed only when there are no build errors present in the project.

Snapshots of configuration:





# QA Automation Script Integration

## Basic Overview

QA Automation Scripts are hosted in a separate GitHub repository: <https://github.com/progrexion/pgx-qa-automation.git>

This secondary repo is checked-out/cloned in the corresponding CircleCI job after setting up the SSH keys, as mentioned in [2.c](#CIRCLECI_CONFIG), earlier in this document.

Scripts are run using Maven commands, and the test results are stored into the Job’s Artifacts section.

## CircleCI Pipeline Config

CircleCi Pipeline config is saved in the following file present under project root, which is where all pipeline related configuration is maintained for the entire project:

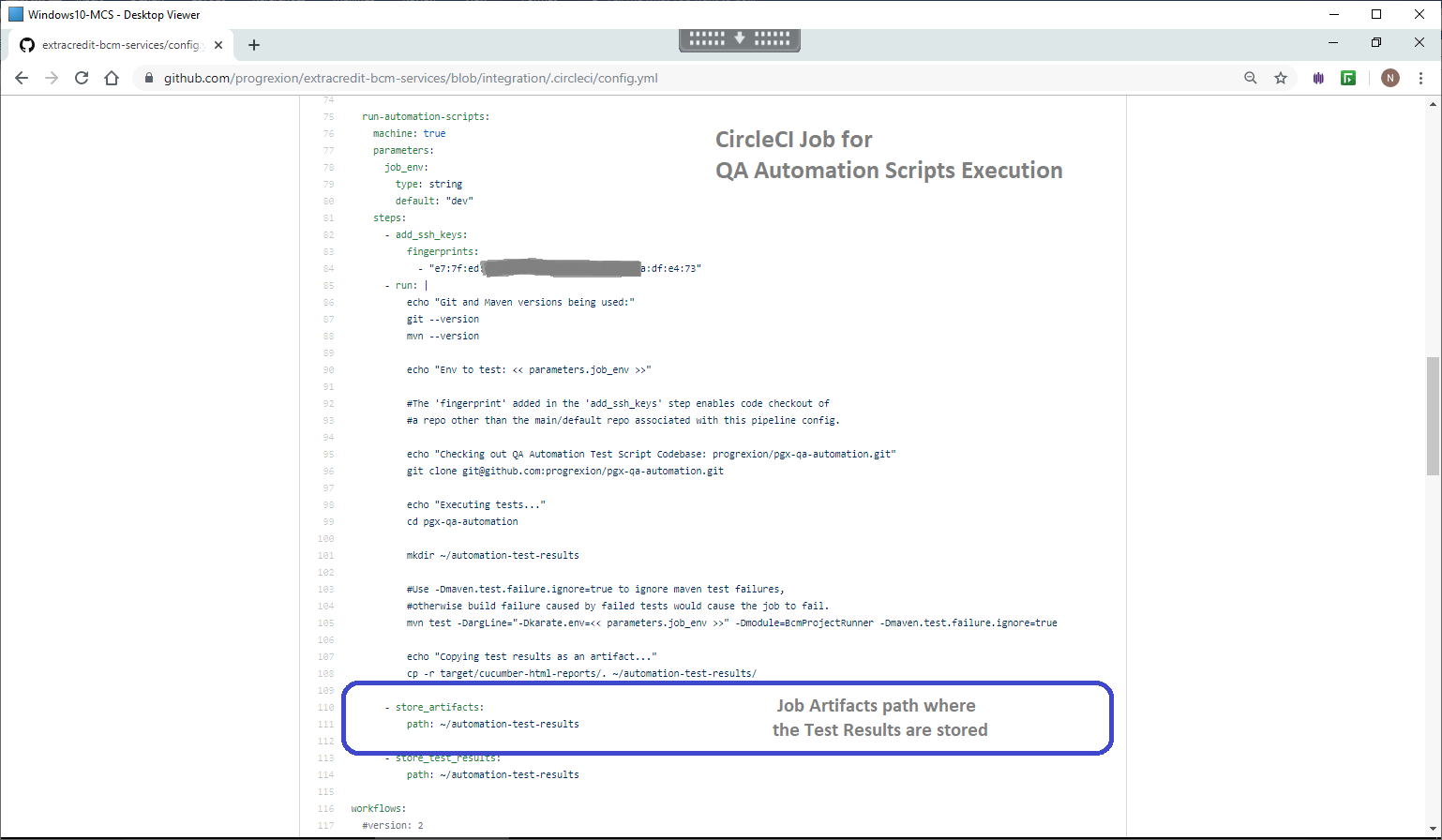
CircleCI Config File: **.circleci/config.yml**

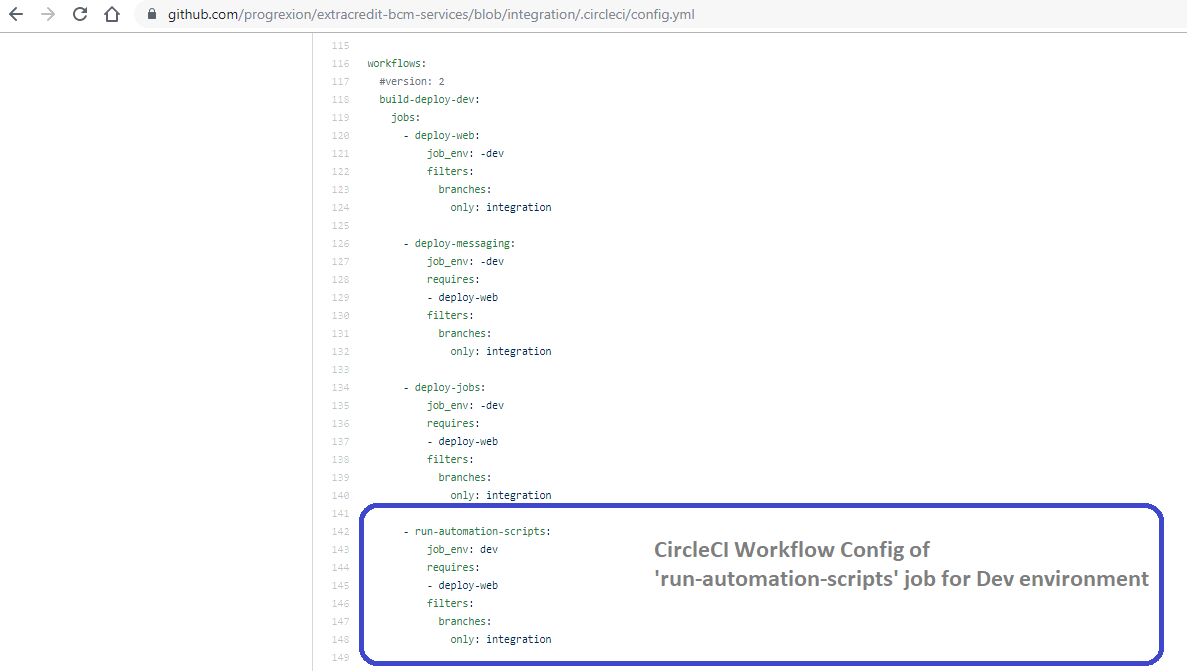
Name of the Job that handles QA Automation Script execution: **run-automation-scripts**

The job is written using an input parameter, which can be passed a value to control which BCM Web environment APIs need to be tested. CircleCI pipeline config version had to be updated to 2.1 to handle this.

In the Workflow configuration, the job is invoked with the ‘deploy-web’ job as a dependency to ensure that the tests are run only after the Web module deployment is completed.

Snapshots of configuration:





Test Results Stored as Job Artifacts (Sample taken from a Workflow/job run for Dev environment):

