CICD WORKSHOP

UPDATED: 5.10.2021

CONTENTS

“Whether you think you can or think
you can’t, you’re right.”
Henry Ford

SECTION 01

CICD OVERVIEW

CICD

Continuous Integration

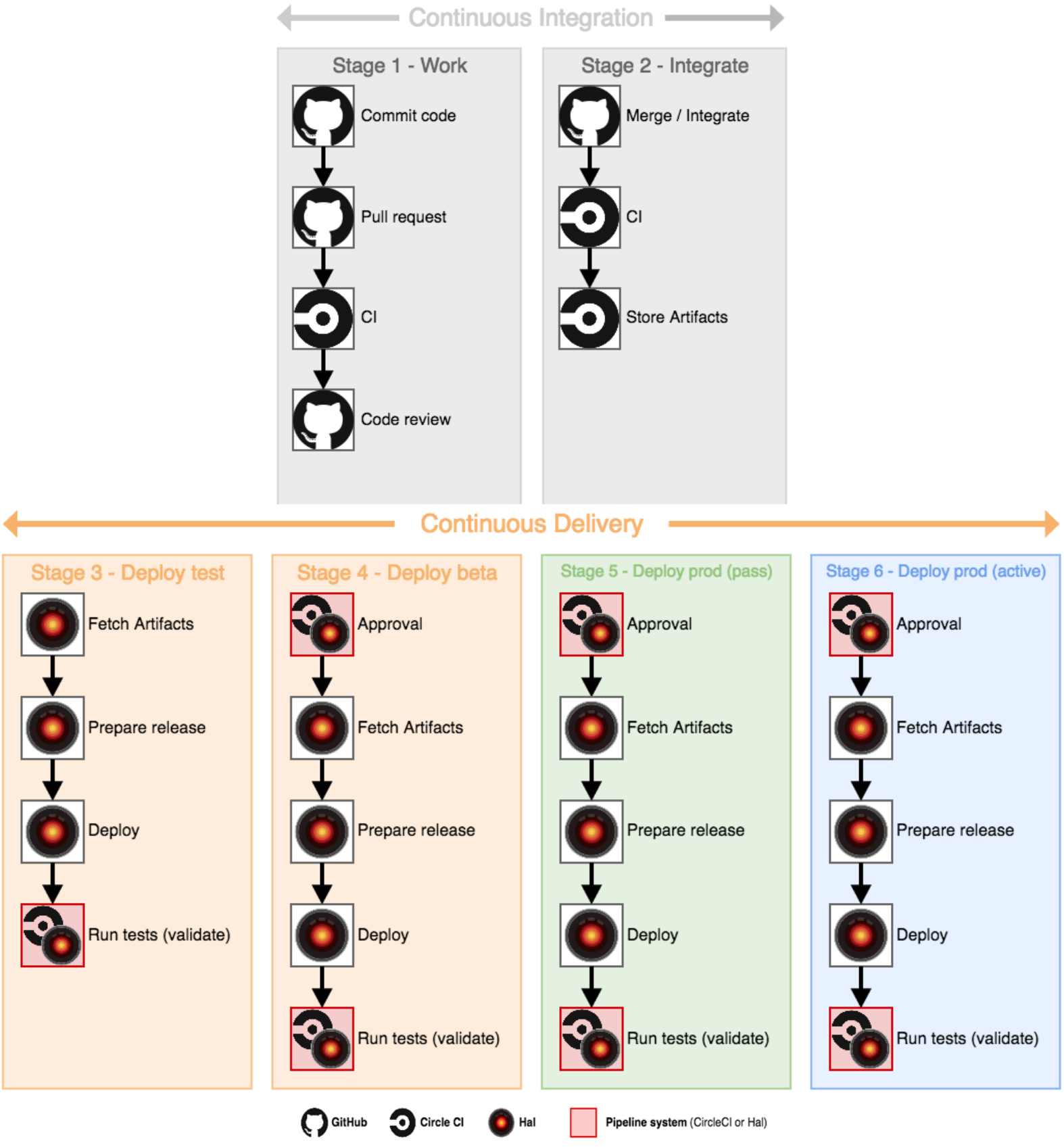
Continuous Integration is a development practice where developers continually run tests and integrate new code into their main branch.

If you merge code frequently, what might you avoid?

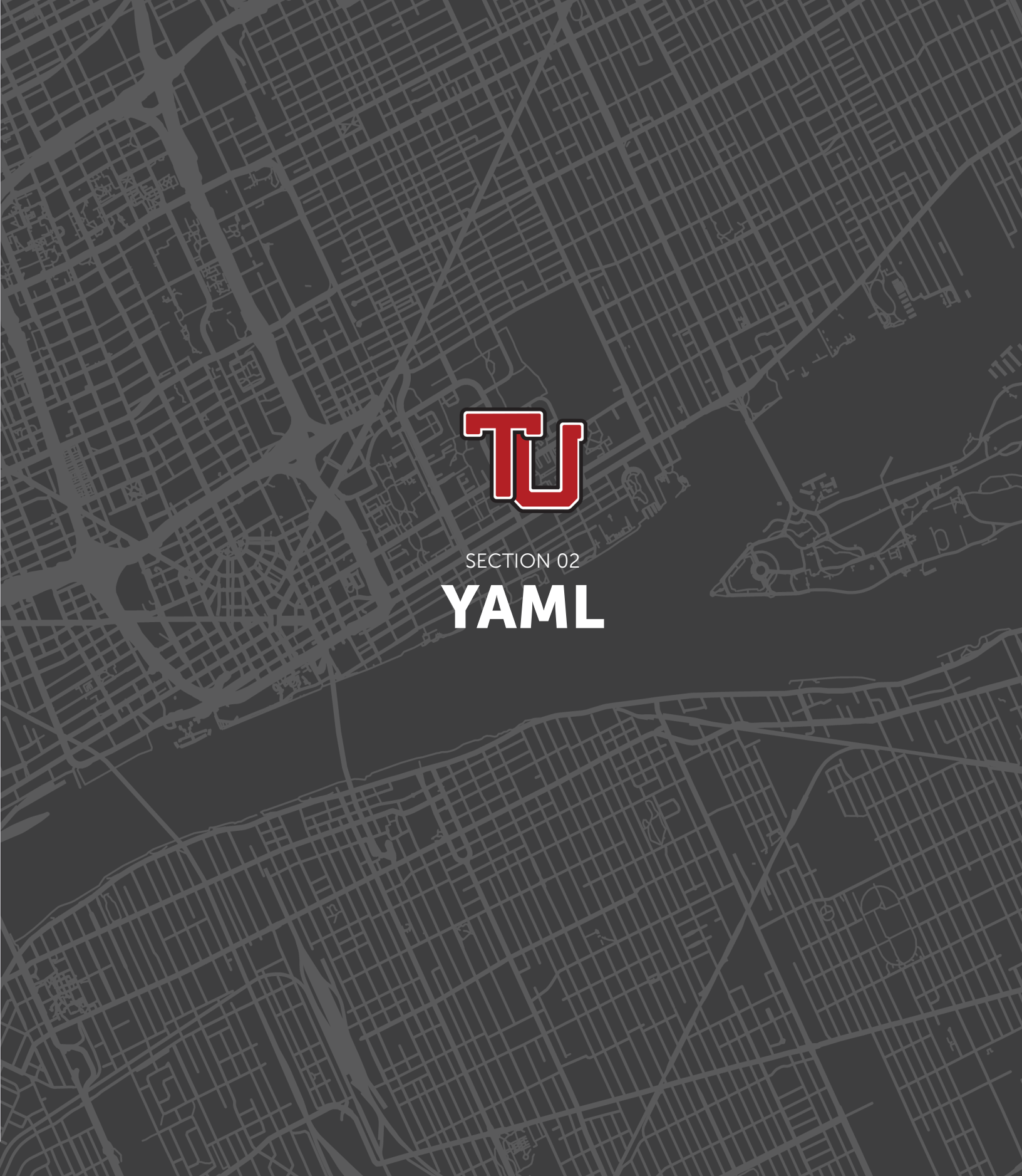
Continuous Delivery / Continuous Deployment

Continuous Delivery typically has manual steps (at least to prod), and deployment is fully automated.

Notes


Continuous Integration
Stage 1 - Work
Commit code
Pull request
CI
Code review
Stage 2 - Integrate
Merge / Integrate
CI
Store Artifacts
Continuous Delivery
Stage 3 - Deploy test
Fetch Artifacts
Prepare release
Deploy
Run tests (validate)
Stage 4 - Deploy beta
Approval
Fetch Artifacts
Prepare release
Deploy
Run tests (validate)
Stage 5 - Deploy prod (pass)
Approval
Fetch Artifacts
Prepare release
Deploy
Run tests (validate)
Stage 6 - Deploy prod (active)
Approval
Fetch Artifacts
Prepare release
Deploy
Run tests (validate)
GitHub
Circle CI
Hal
Pipeline system (CircleCI or Hal)

"Don't be afraid to give up the good to go for the great."
John D. Rockefeller



SECTION 02

YAML

YAML

“YAML Ain’t Markup Language”

YAML was created for data serialization, and is a superset of JSON.

Whitespace is important, and used for denoting structure. Remember to never use tabs with YAML. List members are marked by a leading hyphen, and only one member is allowed per line.

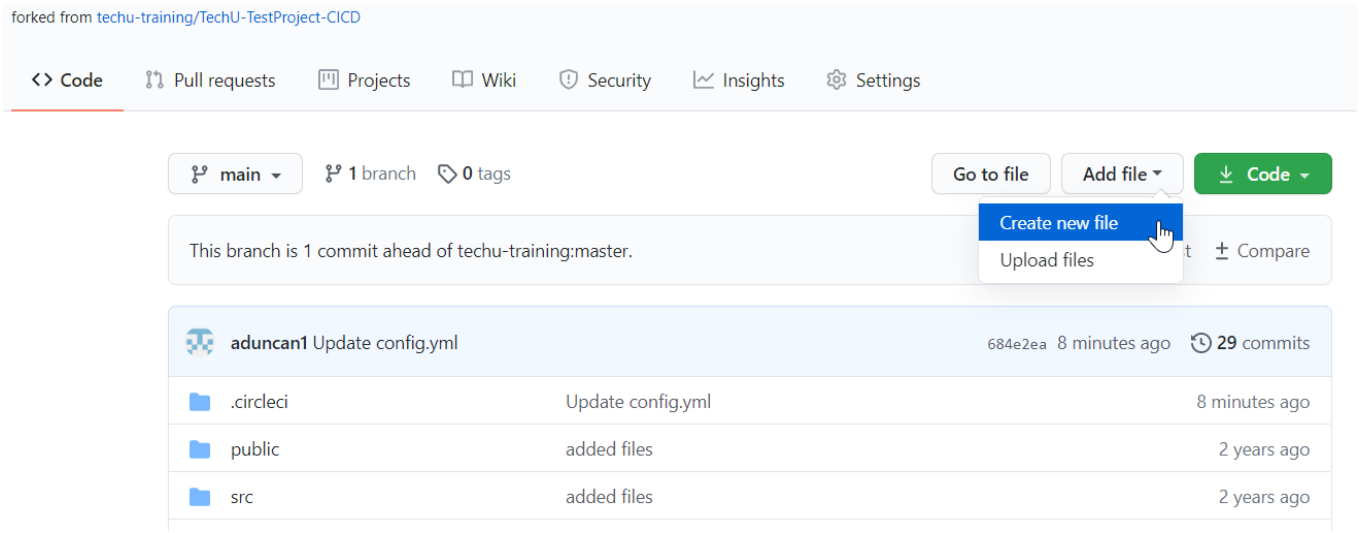
YAML has associative arrays that use key value pairs and colon space, ex: *key : value*.
An ampersand can be used as an anchor to create a node reference. An asterisk can then be used as an alias to refer back to the anchor.

NOTES

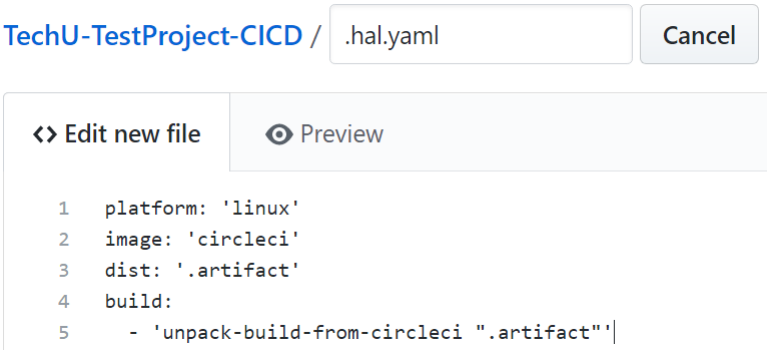
YAML CONFIG FILE

Create Config File

1) In GitHub, go to *your fork* of **TechU-TestProject-CICD**. If you need to fork the repo, you can find it here: https://git.rockfin.com/techu-training/TechU-TestProject-CICD
2) Click the button to **Create new file**



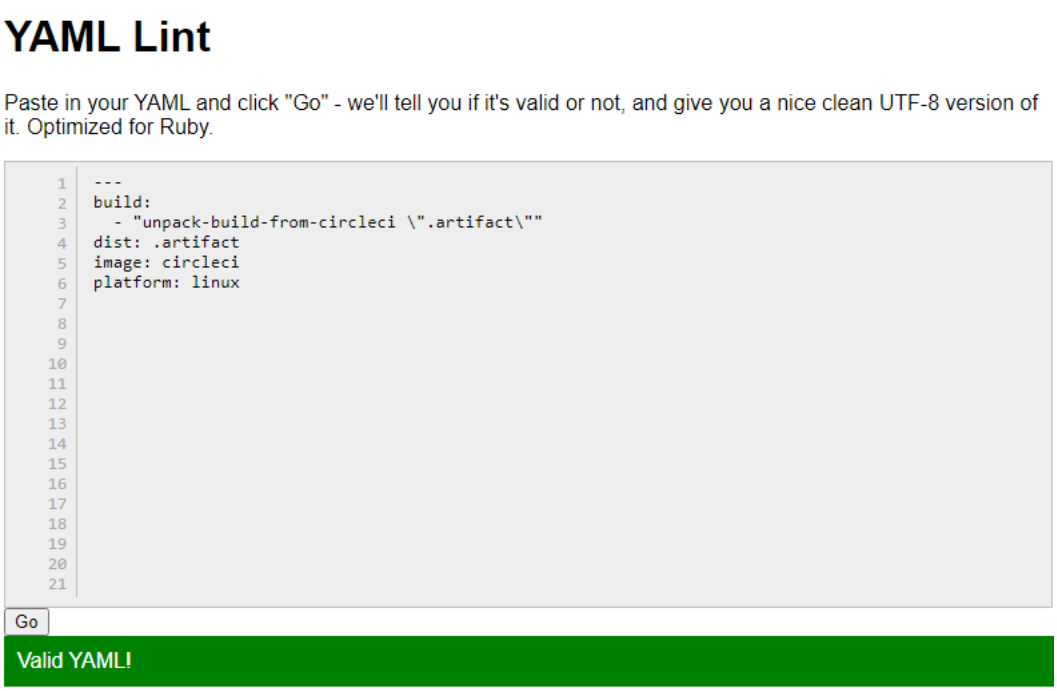
3) The new file will be called **.hal.yaml**
The five lines of the file will be as follows:



YAML Linter

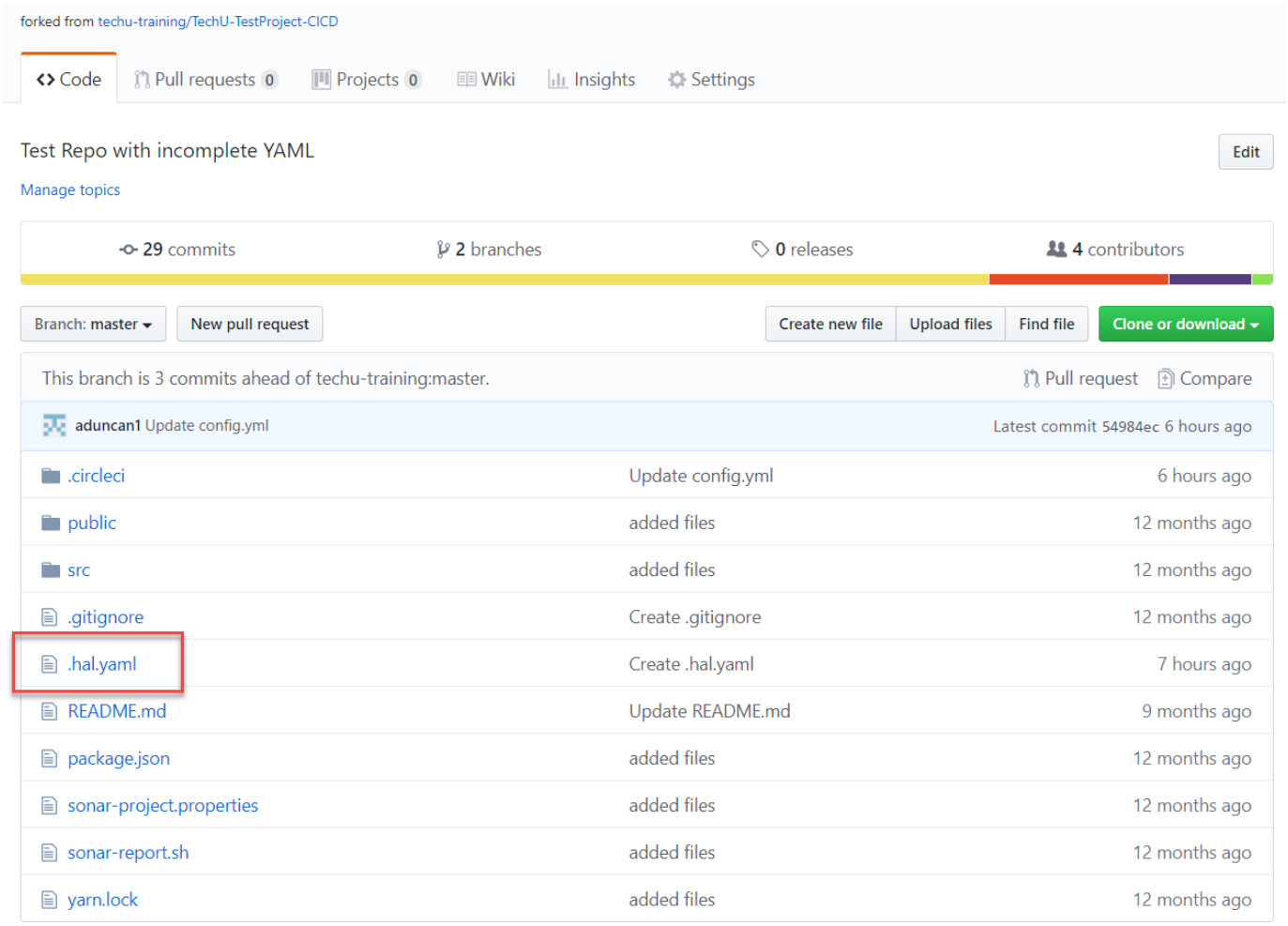
Before we commit our config file, we need to run it through a linter to verify it is formatted correctly.

1) Go to yamllint.com. This is the linter we will use today. ***Remember to never put secrets in an online linting tool***
2) Paste your YAML in and click Go. A green banner should pop up verifying you have valid YAML.



Commit Config File

1) In GitHub, scroll down and select the radio button to **Create a new branch for the commit and start a pull request**. You can keep the default name given to the branch.
2) Click **Propose new file**.
3) For the pull request, you can add a note "Adding Hal config file," then click **Create pull request**.
4) On the next page, click **Merge pull request**.
5) Finally, click **Attempt merge**. Your PR should now be successfully merged and closed, and the Hal config file should be in your repo.


forked from techu-training/TechU-TestProject-CICD
Code
Pull requests 0
Projects 0
Wiki
Insights
Settings
Test Repo with incomplete YAML
Edit
Manage topics
29 commits
2 branches
0 releases
4 contributors
Branch: master
New pull request
Create new file
Upload files
Find file
Clone or download
This branch is 3 commits ahead of techu-training:master.
Pull request
Compare
aduncan1 Update config.yml
Latest commit 54984ec 6 hours ago
.circleci Update config.yml 6 hours ago
public added files 12 months ago
src added files 12 months ago
.gitignore Create .gitignore 12 months ago
.hal.yaml Create .hal.yaml 7 hours ago
README.md Update README.md 9 months ago
package.json added files 12 months ago
sonar-project.properties added files 12 months ago
sonar-report.sh added files 12 months ago
yarn.lock added files 12 months ago

NOTES

“Creativity is intelligence having fun.”
Albert Einstein

SECTION 03

CIRCLECI CONFIG

Open up the **.circleci** folder to access the **config.yml** file. Here we can see how our build is prepared so it can be deployed by Hal. **Lines 1-37** are where many of our defaults are defined.

```
1  attach_workspace: &attach_workspace
2      attach_workspace:
3          at: '.'
4
5  persist_workspace: &persist_workspace
6      persist_to_workspace:
7          root: '.'
8          paths: [ '.' ]
9
10 persist_hal_build_id_to_workspace: &persist_hal_build_id_to_workspace
11     persist_to_workspace:
12         root: '.'
13         paths: [ '.hal_build_id' ]
14
15 persist_coverage_to_workspace: &persist_coverage_to_workspace
16     persist_to_workspace:
17         root: '.'
18         paths: [ 'test-report.xml', 'coverage/' ]
19
20 deploy_nonprod_job: &deploy_nonprod_job
21     working_directory: '~/project'
22     docker: [ {image: 'qldockerdtr.rockfin.com/circleci/hal-integration:hal-publisher'} ]
23     steps:
24         - *attach_workspace
25         - run: 'deploy-to-hal-passive-only'
26
27 deploy_prod_job: &deploy_prod_job
28     working_directory: '~/project'
29     docker: [ {image: 'qldockerdtr.rockfin.com/circleci/hal-integration:hal-publisher'} ]
30     steps:
31         - *attach_workspace
32         - run: 'deploy-to-hal'
33
34 deploy_defaults: &deploy_defaults
35     working_directory: '~/project'
36     docker:
37         - image: 'qldockerdtr.rockfin.com/circleci/hal-integration:hal-publisher'
```

NOTES

Line 39 indicates what version of the config schema we will use.
Line 40 is where we begin our jobs and essentially are telling CircleCI what to do/run.
Line 41-55 is the compile and build job, and is named based on naming connventions at shorty/gatenaming.
Line 57-64 is the unit tests job. This allows us to automate our unit tests so that they can be run any time new code is committed.

```
39  version: 2
40  jobs:
41    compile_and_build:
42      docker:
43        - image: node:12.6
44      steps:
45        - checkout
46        - run:
47            name: Greeting
48            command: echo Hello, world.
49        - run:
50            name: Yarn Install
51            command: yarn install
52        - run:
53            name: Yarn Build
54            command: yarn build
55        - *persist_workspace
56
57    unit_tests:
58      docker:
59        - image: node:12.6
60      steps:
61        - *attach_workspace
62        - run:
63            name: Yarn Test
64            command: yarn test
65
```

NOTES

Line 66-79 is the publish build to Hal job. This is how we get our build from CircleCI to Hal for deployment.
Line 81-89 is test deploy. This is what feeds the necessary information to Hal, so Hal can deploy our application to a designated test environment.
Line 91-97 measure code coverage.

```
66    publish_build_to_hal:
67      working_directory: '~/'
68      docker: [ {image: 'qldockerdtr.rockfin.com/circleci/hal-integration:hal-publisher'} ]
69      environment:
70          HAL_APP_ID: '6941'   #Fill in HAL application ID
71      steps:
72          - *attach_workspace
73          - run: mkdir -p "${HOME}/build/"
74          - run: cd build && zip -r "${HOME}/build/build.zip" . && cd ../
75          - run: ls "${HOME}/build/"
76          - store_artifacts:
77              path: "./build/build.zip"
78          - run: 'publish-build-to-hal'
79          - *persist_hal_build_id_to_workspace
80
81    test_deploy:
82      <<: *deploy_defaults
83      environment:
84        - HAL_APP_ID: '6941'    #Fill in HAL application ID
85        - HAL_TARGETS: "39319"  #Fill in HAL deployment target ID
86        - HAL_BUILD_FILE: '.hal_build_id'
87      steps:
88        - *attach_workspace
89        - run: 'deploy-to-hal'
90
91    cover:
92      docker:
93        - image: node:12.6
94      steps:
95        - *attach_workspace
96        - run: yarn cover
97        - *persist_coverage_to_workspace
```

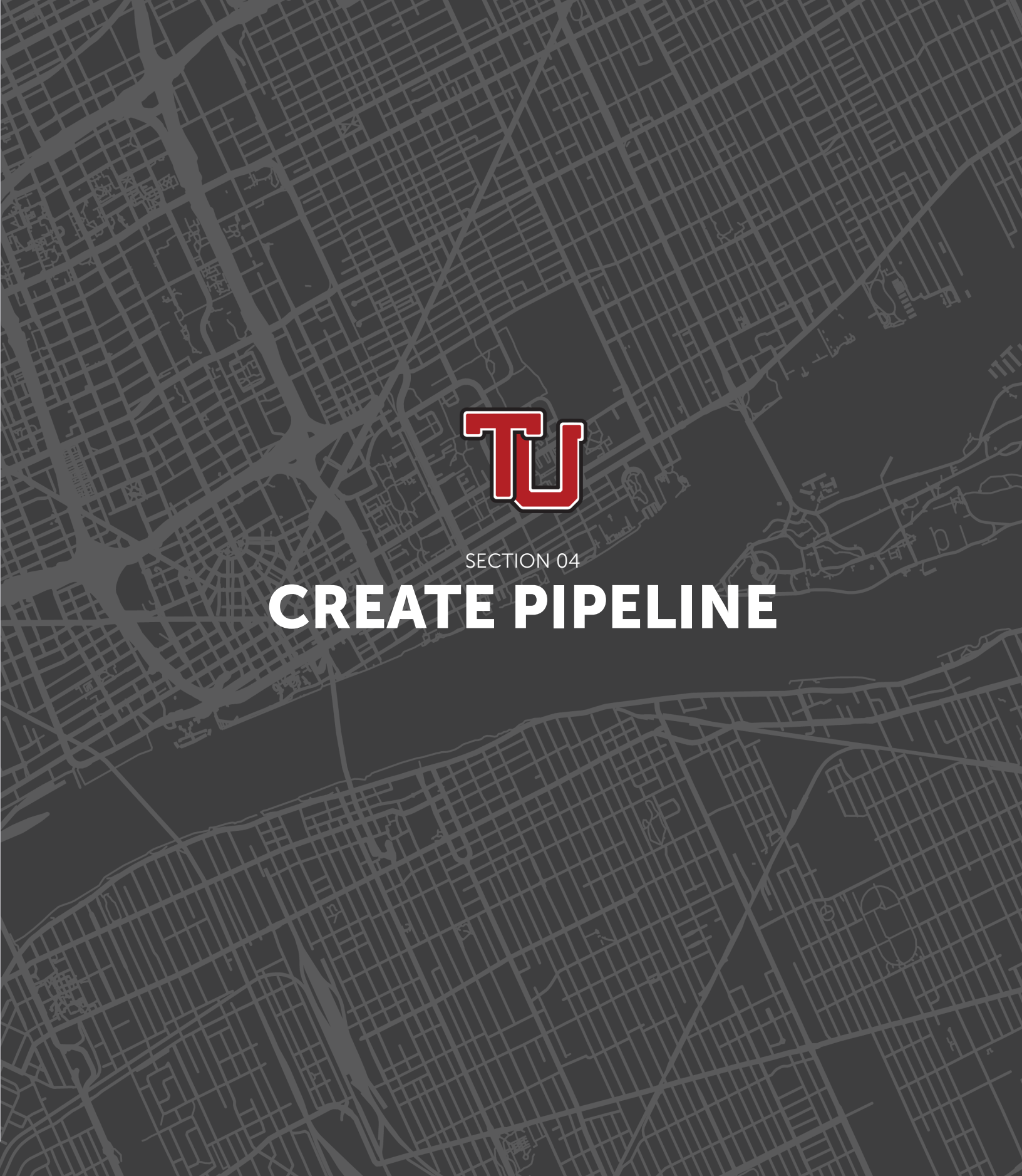
NOTES

Line 99-110 is what feeds testing data to SonarQube.
Line 112-129 is where we orchestrate how the jobs above are run using workflows.
Remember, the order designated in your workflows determines the order in which your jobs are run.

```
99    sonar:
100     working_directory: '~/project'
101     docker:
102       - image: qldockerdtr.rockfin.com/qapow/sonarscanner
103     steps:
104       - *attach_workspace
105       - run: ls coverage
106       - run:
107           name: Replace Circle Hash
108           command: sed -i -e "s~\${CIRCLE_SHA}~$CIRCLE_SHA1~;" sonar-project.properties
109       - run: cat sonar-project.properties
110       - run: ./sonar-report.sh
111
112 workflows:
113   version: 2
114   workflow:
115     jobs:
116     - compile_and_build
117     - unit_tests:
118         requires: [ compile_and_build ]
119     - cover:
120         requires: [ unit_tests ]
121     - sonar:
122         requires: [ cover ]
123     - publish_build_to_hal:
124         requires: [ unit_tests ]
125         filters:
126             branches:
127                 only: main
128     - test_deploy:
129         requires: [ publish_build_to_hal ]
```

NOTES

"The only limit to our realization of tomorrow will be our doubts of today."
Franklin D. Roosevelt

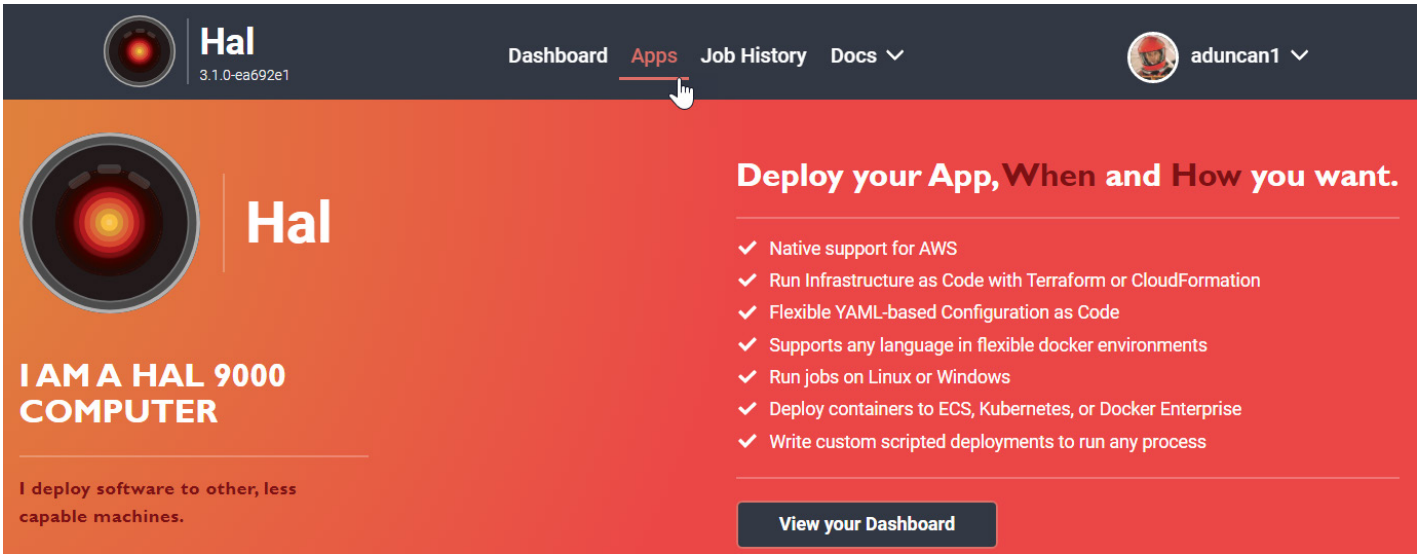


SECTION 04

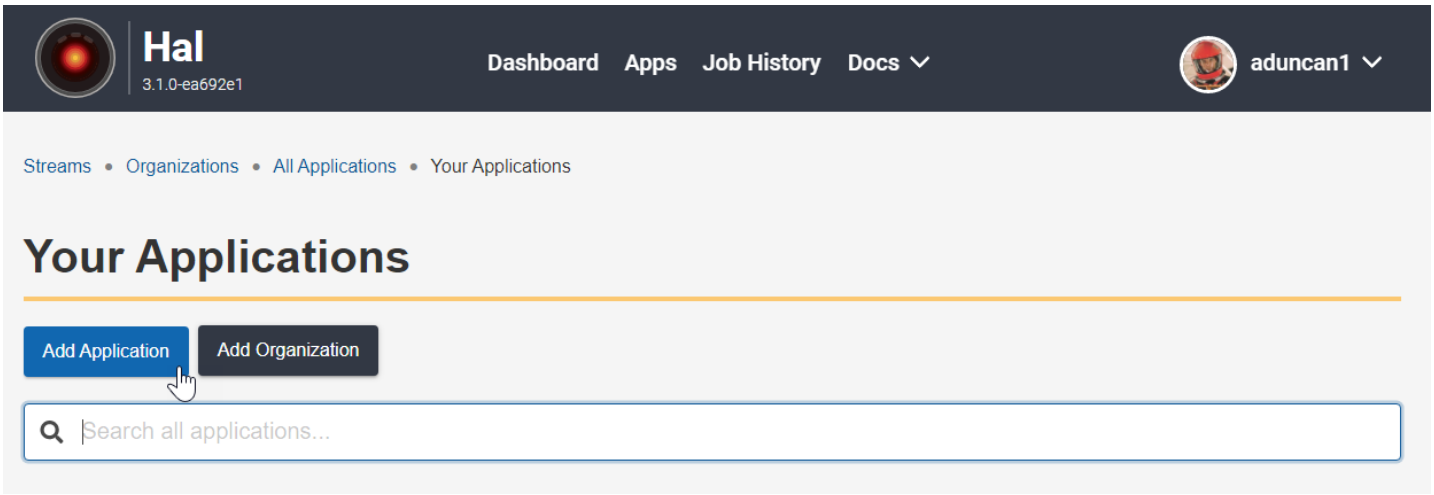
CREATE PIPELINE

ADD APPLICATION IN HAL

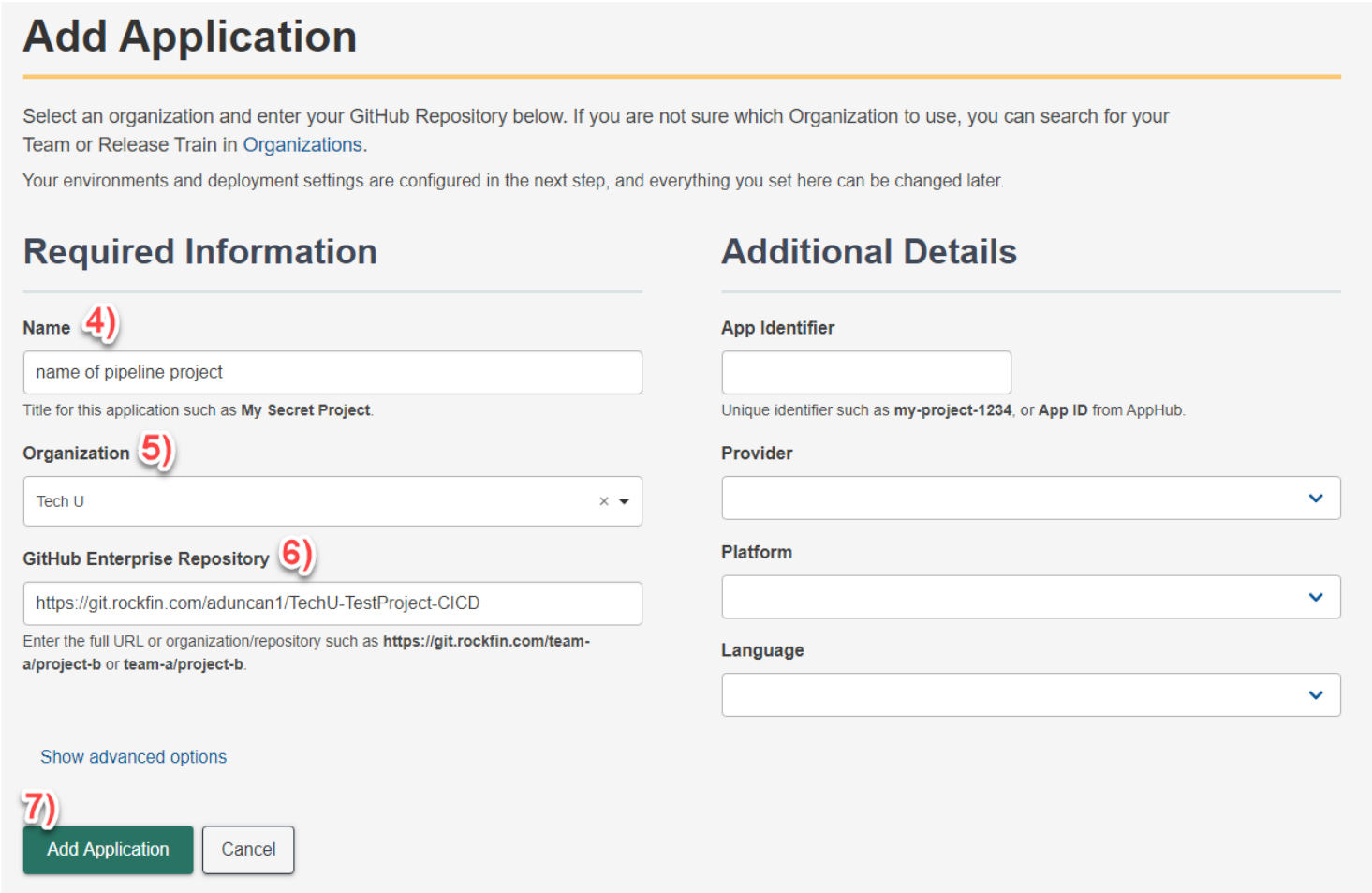
1) In Chrome, go to hal9000/ and sign into Hal. (Use a Dash account for more security)
2) Once signed-in, click on **Apps**.



3) Click on **Add Application**.

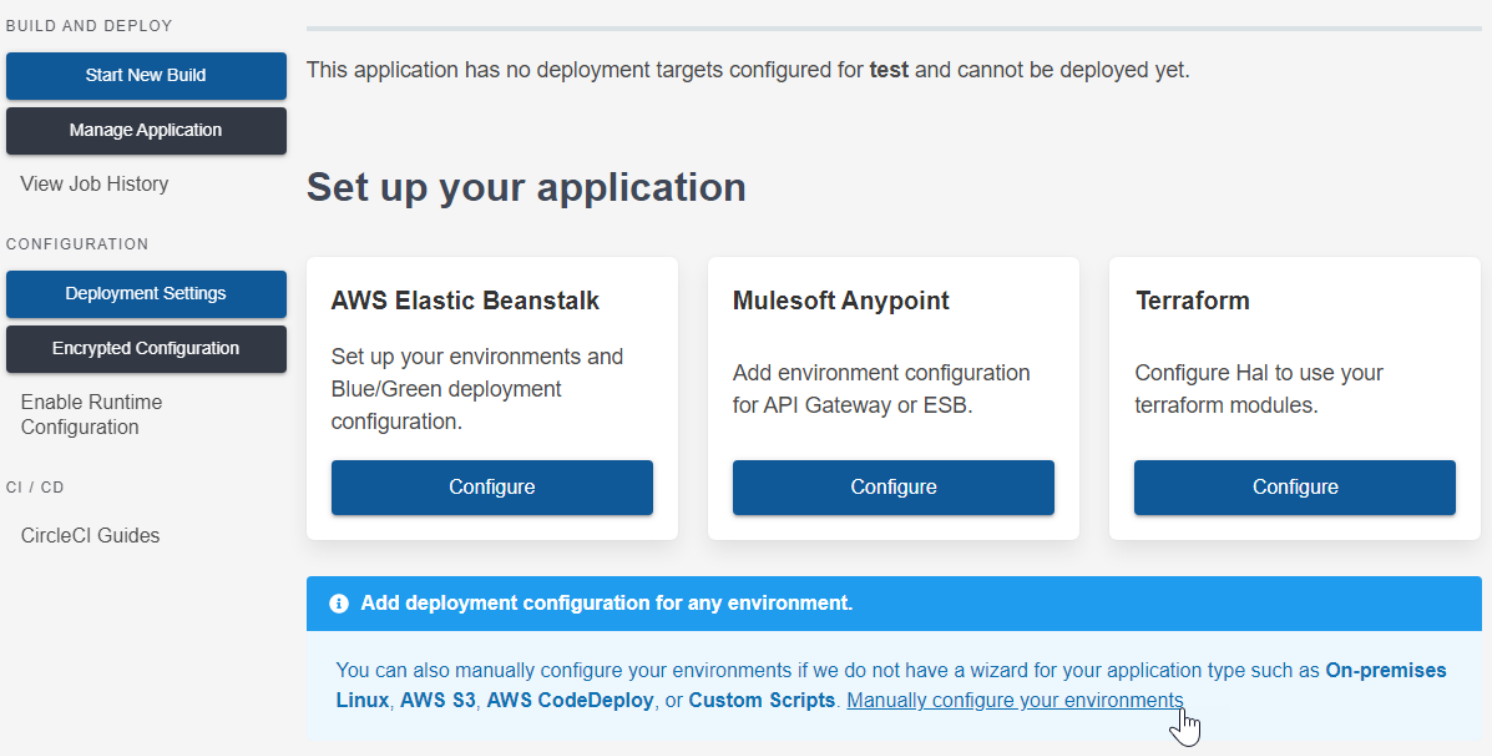


4) Give the new application for your pipeline a name, ex. Tech U CICD Workshop.
5) Set the organization to **Tech U**.
*Make sure you select Tech U with a space. If you select TechU without the space your pipeline will NOT work.
6) For **GitHub Enterprise Repository**, enter in the URL for *your* fork.
7) Click **Add Application**. (It is not necessary to fill out Additional Details or Advanced options for this workshop.)

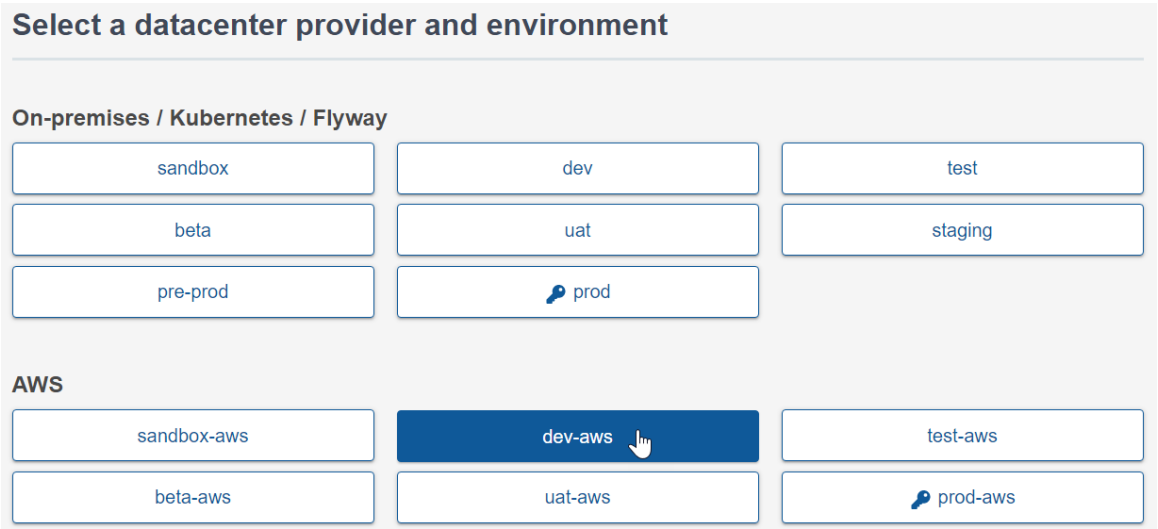


SETUP DEPLOYMENT TARGET

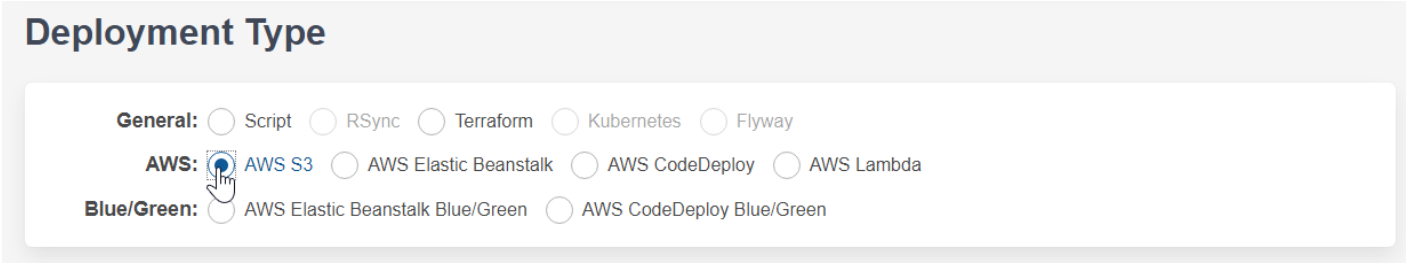
1) Scroll down under the blue header labeled **Add deployment configuration for any environment**, and click on **Manually configure your environments**.



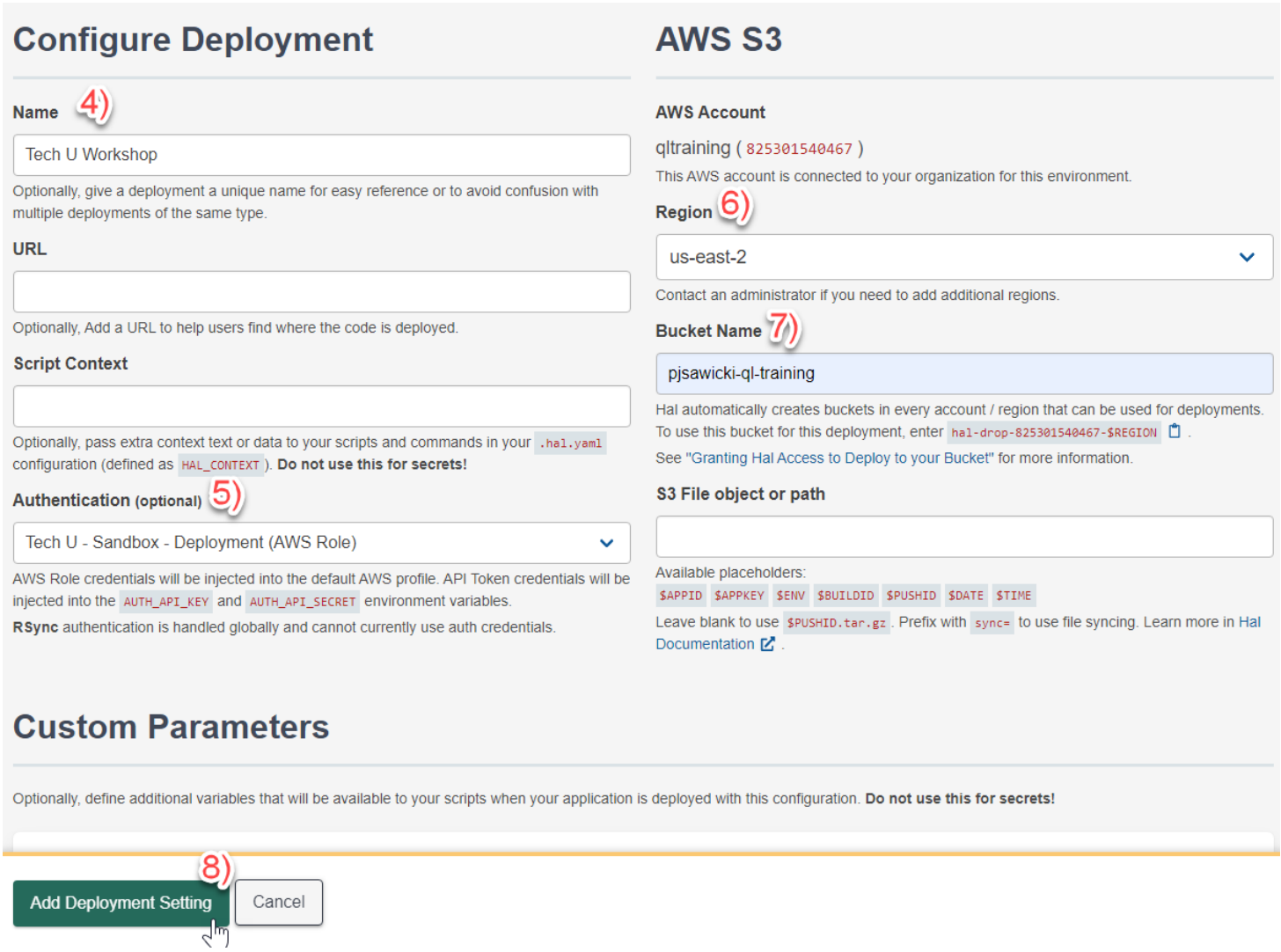
2) For the **datacenter provider and environment**, select **dev-aws**.



3) For the **Deployment Type**, select **AWS S3**.

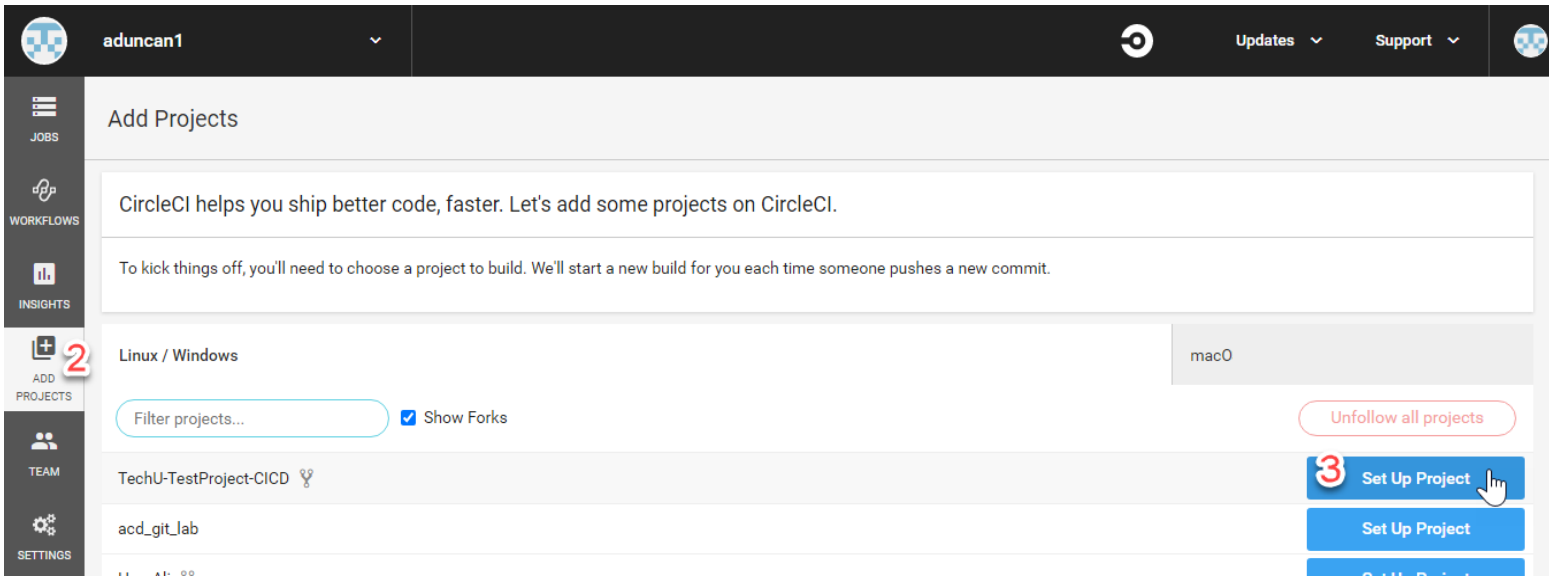


4) Under **Configure Deployment**, give your deployment a name, ex. Tech U Workshop.
5) For **Authentication**, select **Tech U - Sandbox - Deployment (AWS Role)**. *If this option is not available, ensure you selected **Tech U** as your organization when adding the app.
6) For **Region**, select **us-east-2**.
7) For **Bucket Name**, type in **pjsawicki-ql-training**.
8) Click **Add Deployment Setting**.

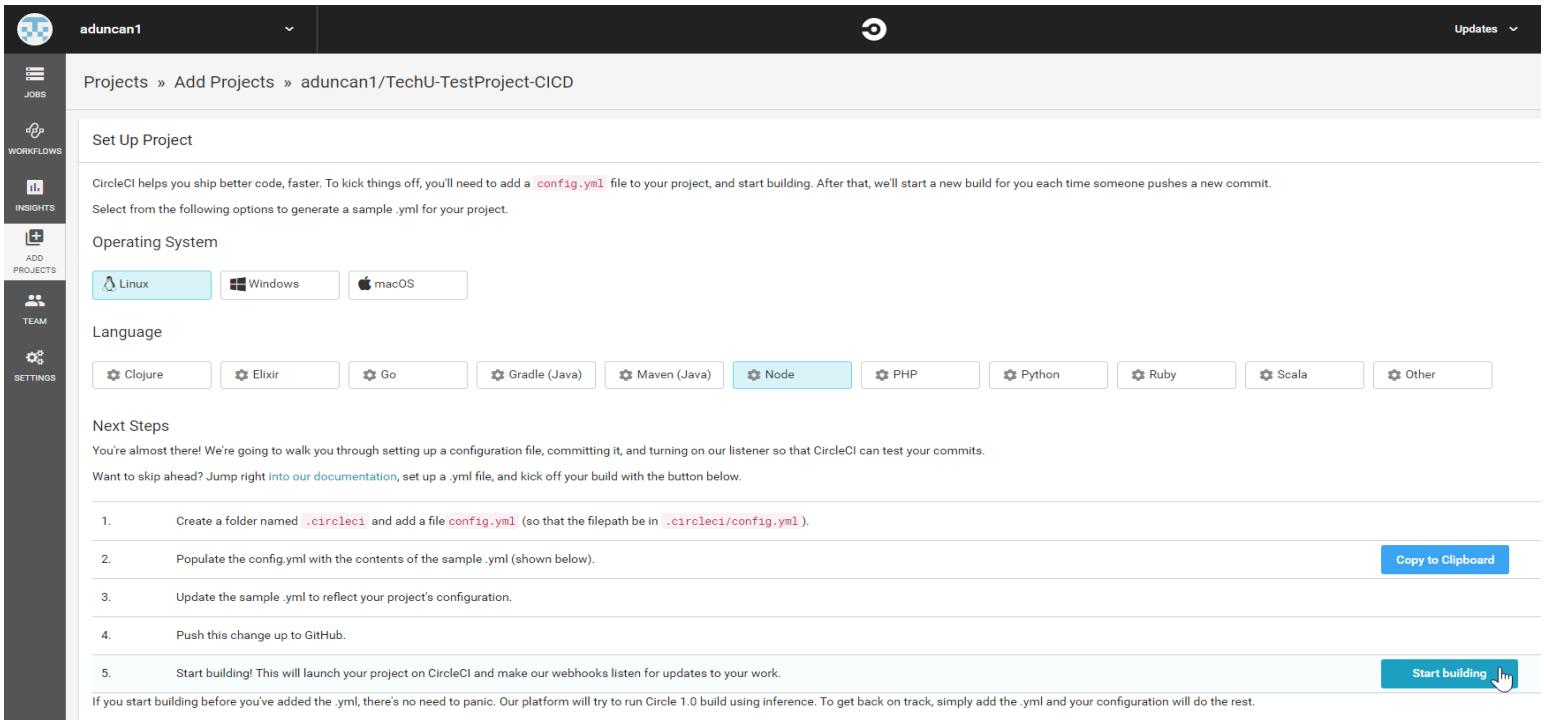


SETUP PROJECT IN CIRCLECI

1) Go to circleci.foc.zone and sign into CircleCI.
2) In CircleCI, you will click **Add Project** on the left to view your projects.
3) Click **Set Up Project** for your project named **TechU-TestProject-CICD**

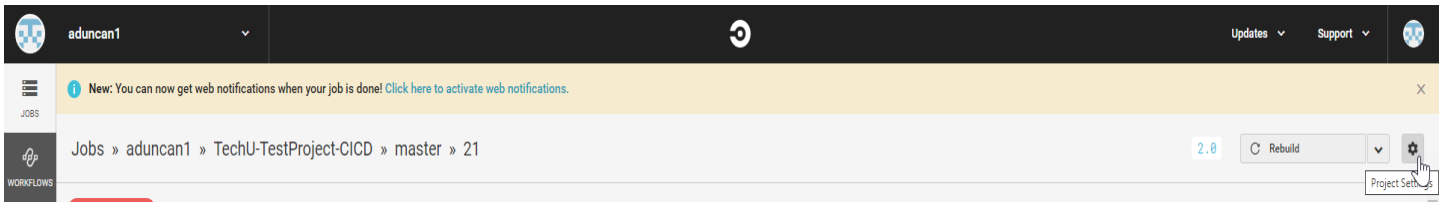


4) **Linux** and **Node** may already be selected for your **Operating System** and **Language**, respectively, but if not, go ahead and select those settings, then click **Start Build**.

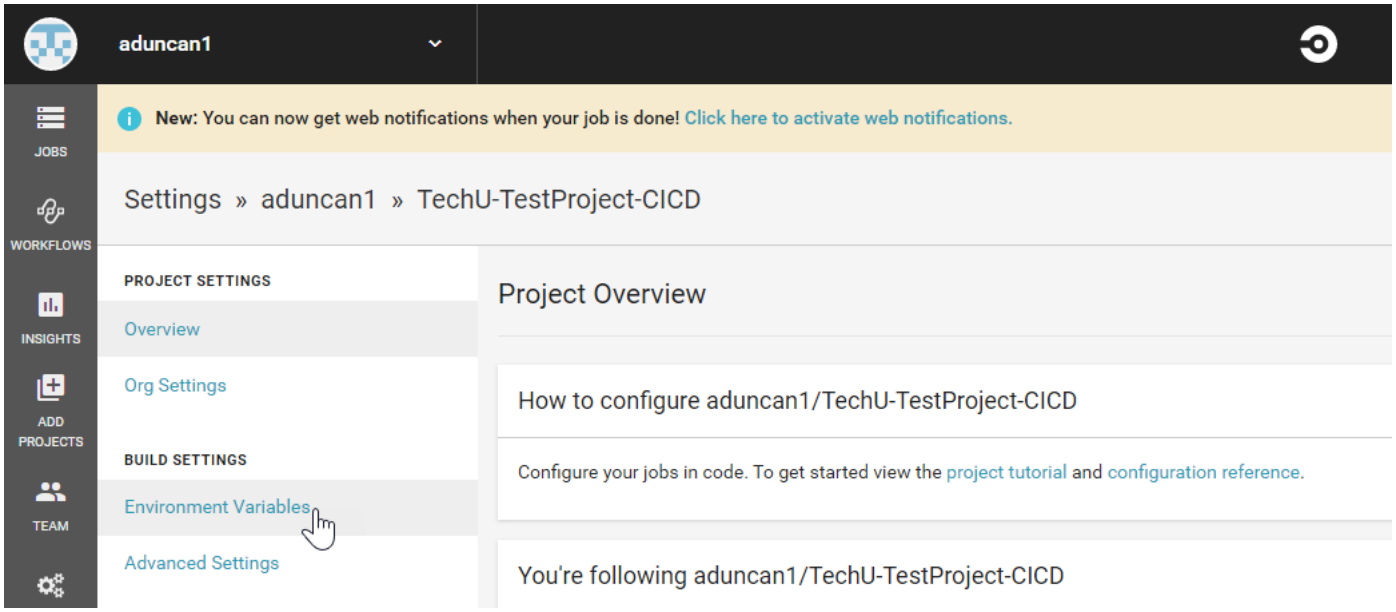


SET SONAR TOKEN IN CIRCLECI

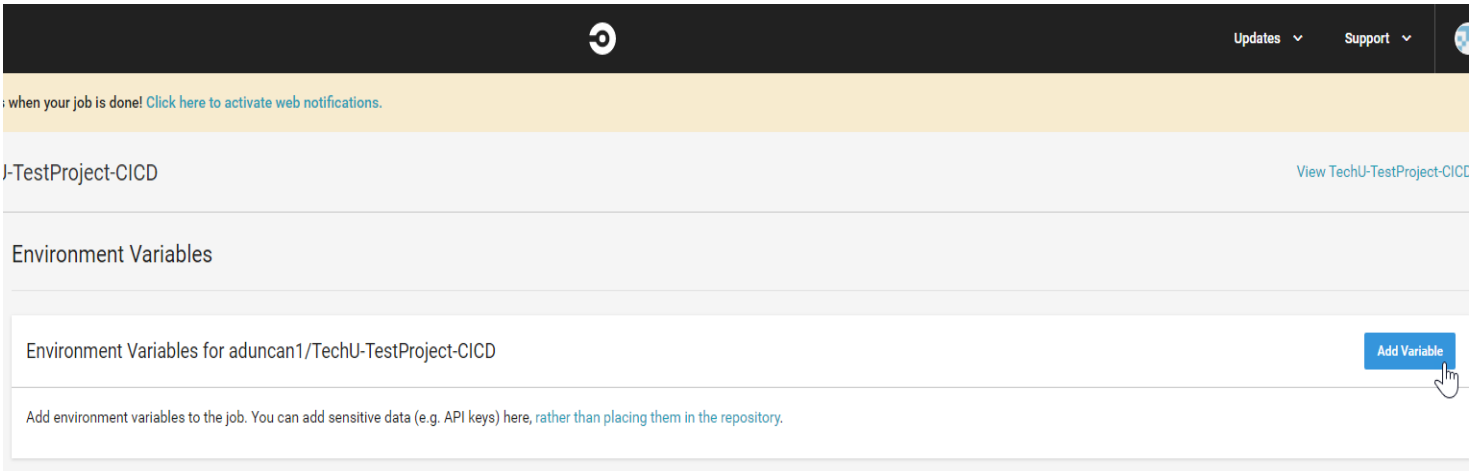
1) Once you are taken back to your project page in CircleCI, click the gear in the upper right-hand corner to get to your **Project Settings**.



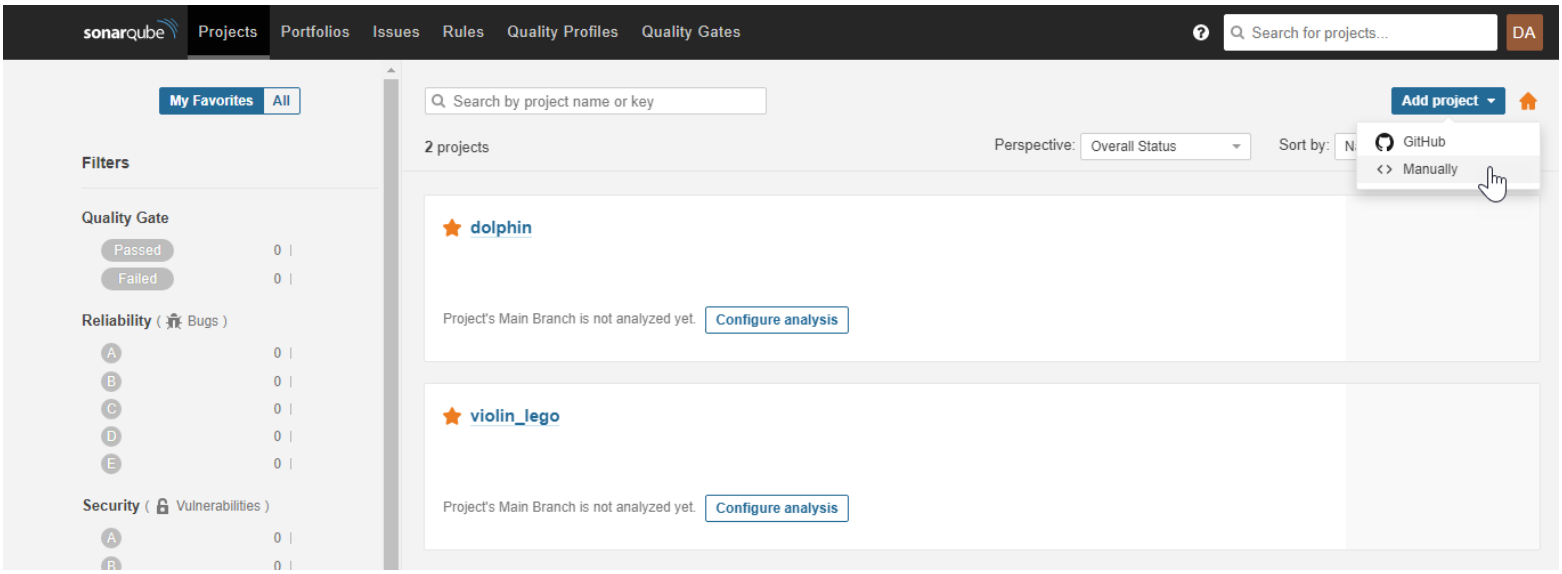
2) On the left, under **Build Settings**, click **Environment Variables**.



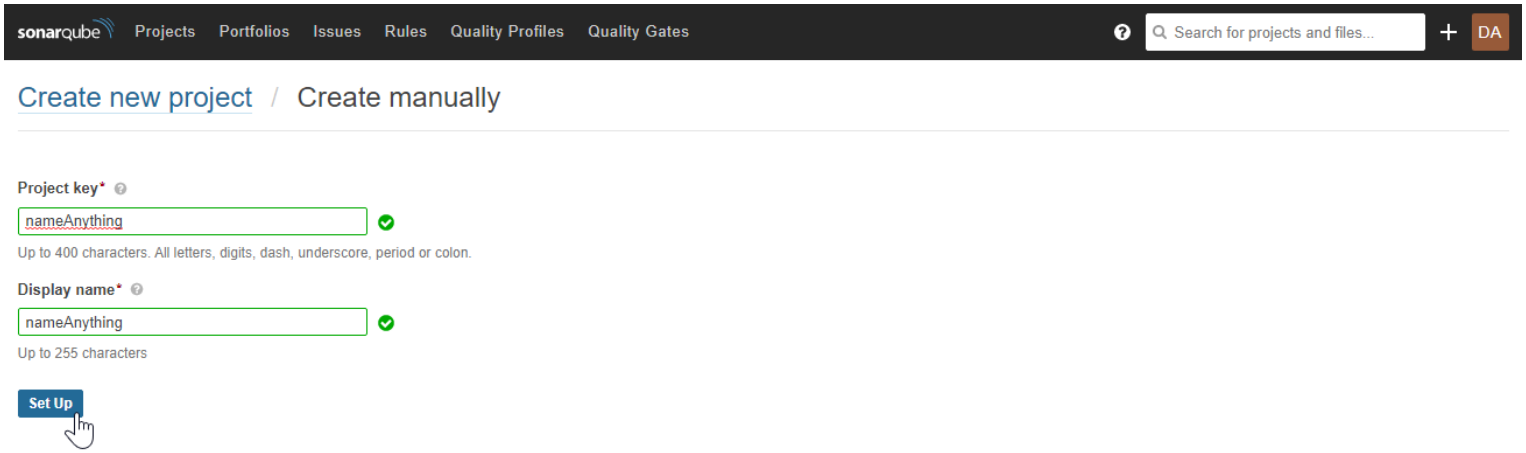
3) Inside **Environment Variables**, click **Add Variable**.



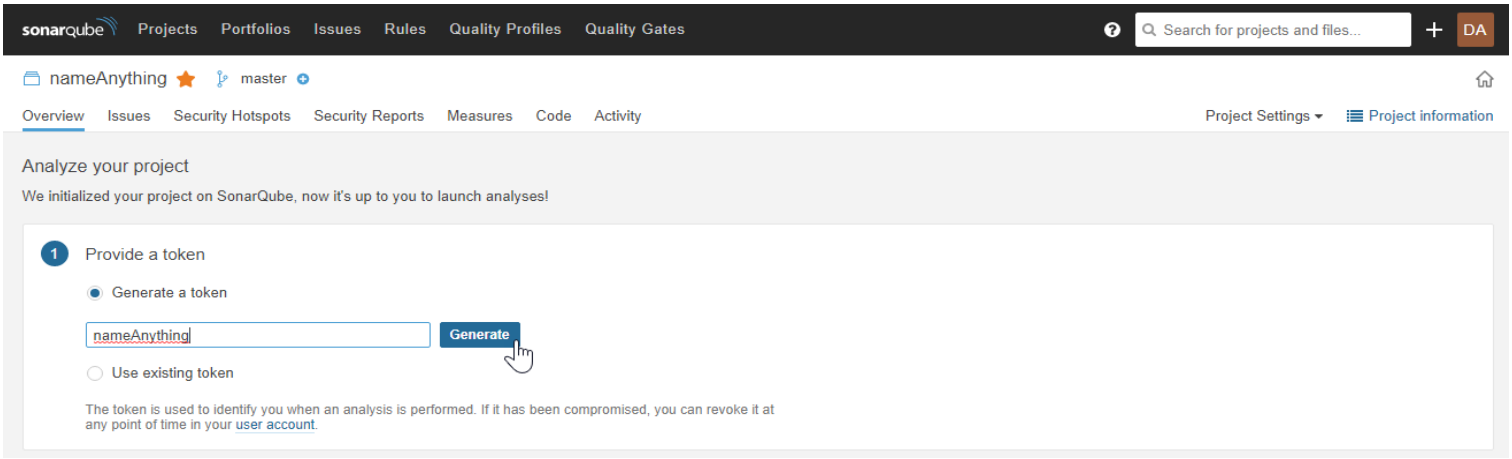
4) Go to **Sonar/** and log into **SonarQube** (you can login with GitHub).
5) Once logged in, click **Add project** and then select **Manually** from the drop-down menu.



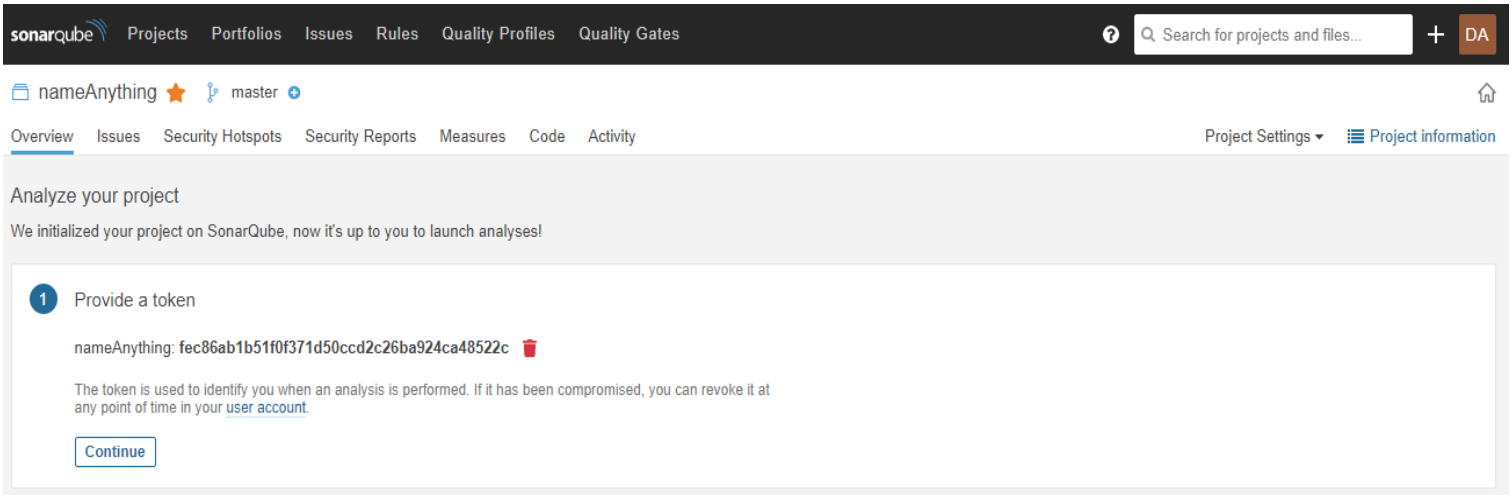
6) For your **Project key**, you can name it anything, the name is not important here. The **Display name** will be autogenerated.
7) Click **Set Up**.



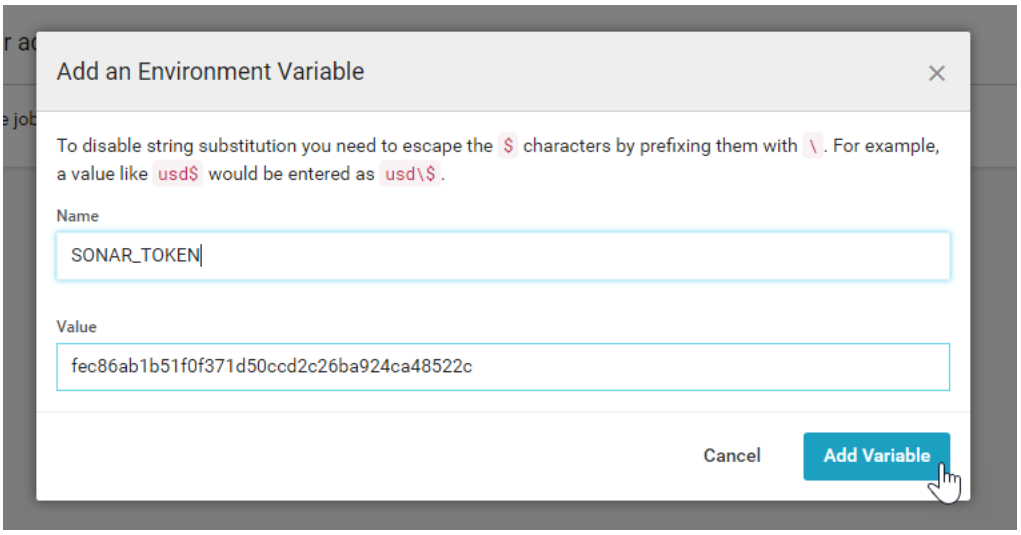
8) To generate a token, it can be named anything, the name is also not important at this step.
9) Click **Generate**.



10) Highlight the token value that is generated and copy it. You can exit SonarQube.

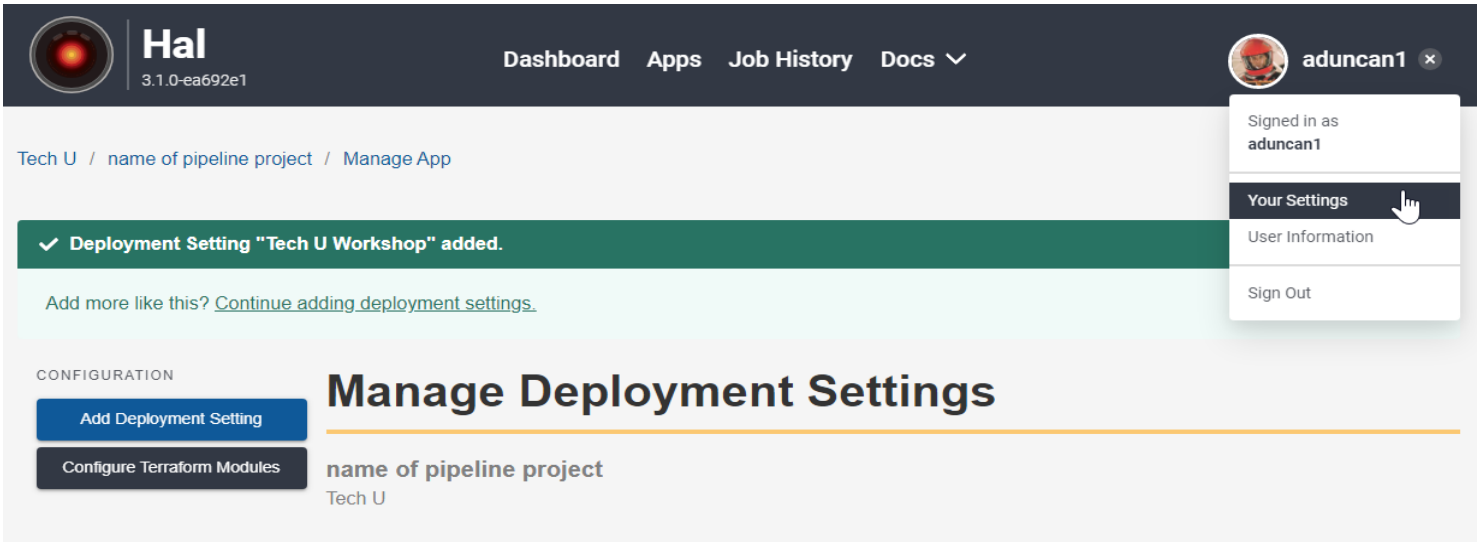


11) Back in **CircleCI**, in the **Add an Environment Variable** window, paste in the value from SonarQube.
12) For the **Name**, type **SONAR_TOKEN**. This name needs to be exact or CircleCI will not know what it is.
13) Click **Add Variable**.

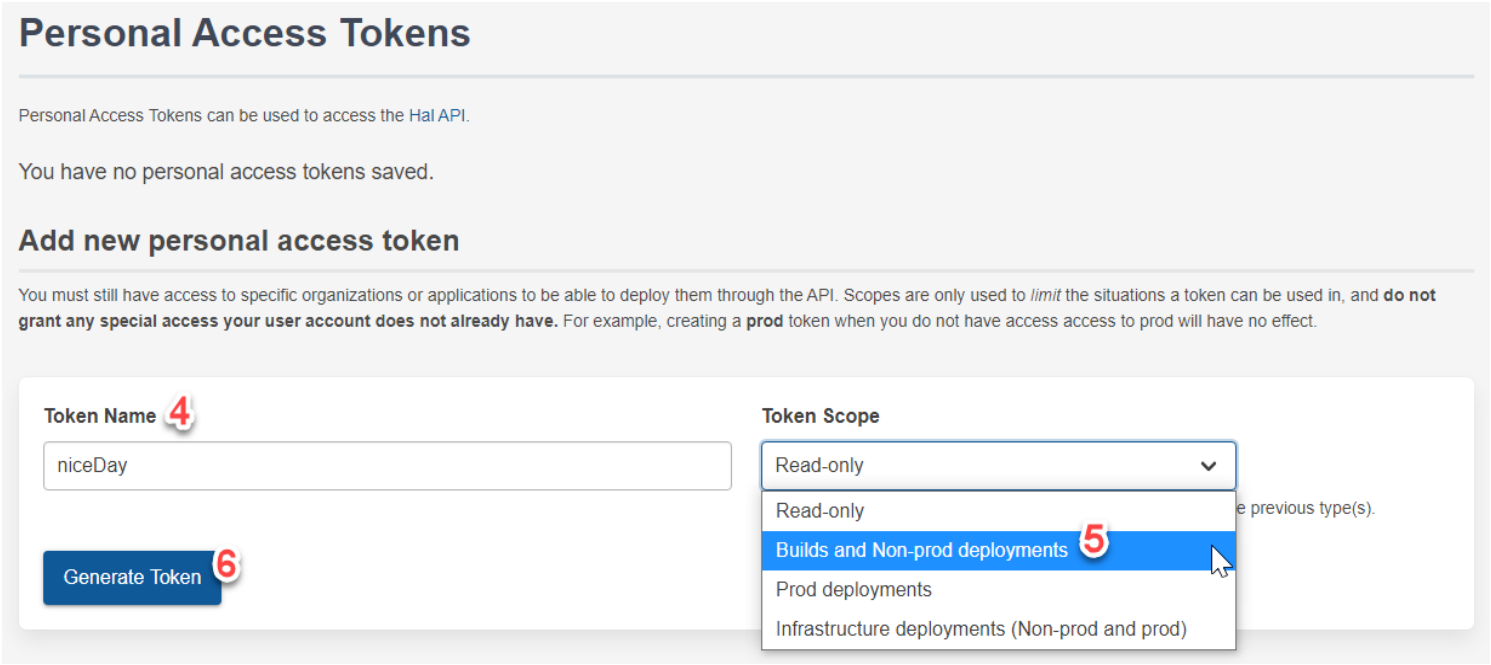


SET HAL API TOKEN IN CIRCLECI

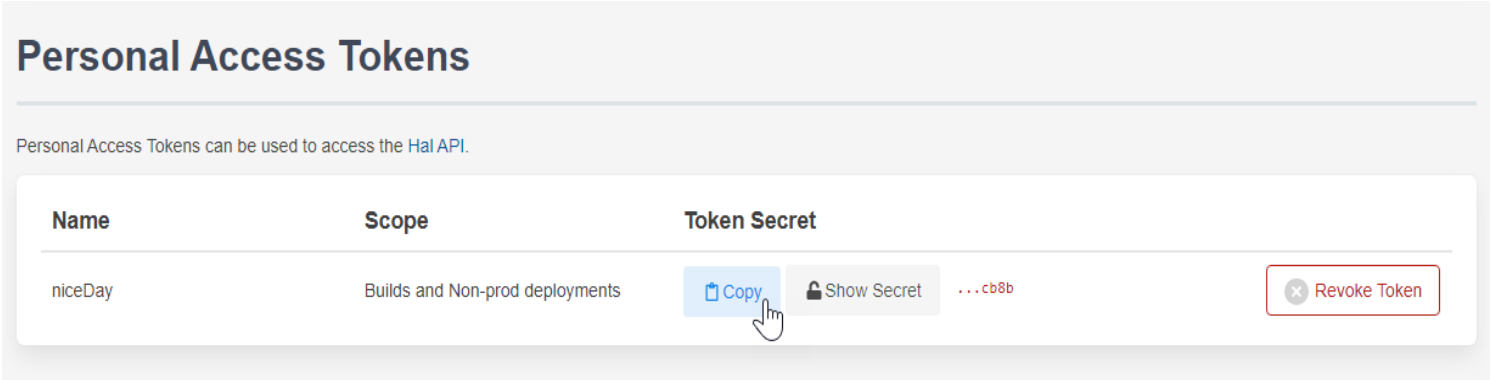
1) In **CircleCI**, inside **Environment Variables**, click **Add Variable**.
2) Go to **Hal**. Once there, click your user name in the top-right corner, then click **Your Settings** from the drop-down menu that appears.



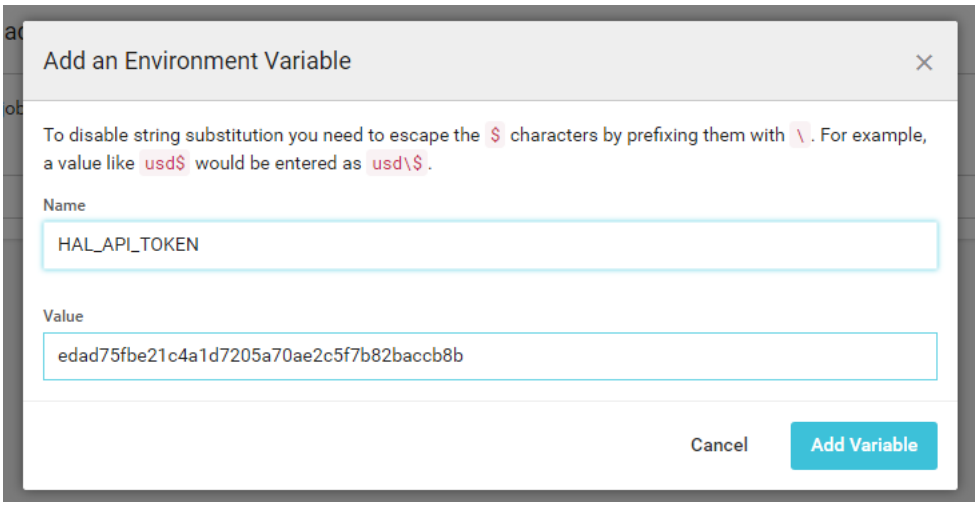
3) On the **Your Settings** page, scroll down to **Add new personal access token**.
4) For the **Token Name**, you can name it anything.
5) For the **Token Scope**, select **Builds and Non-prod deployments** from the drop-down.
6) Click **Generate Token**.



7) Now that your token has been generated, click **Copy** to copy the value of the token.

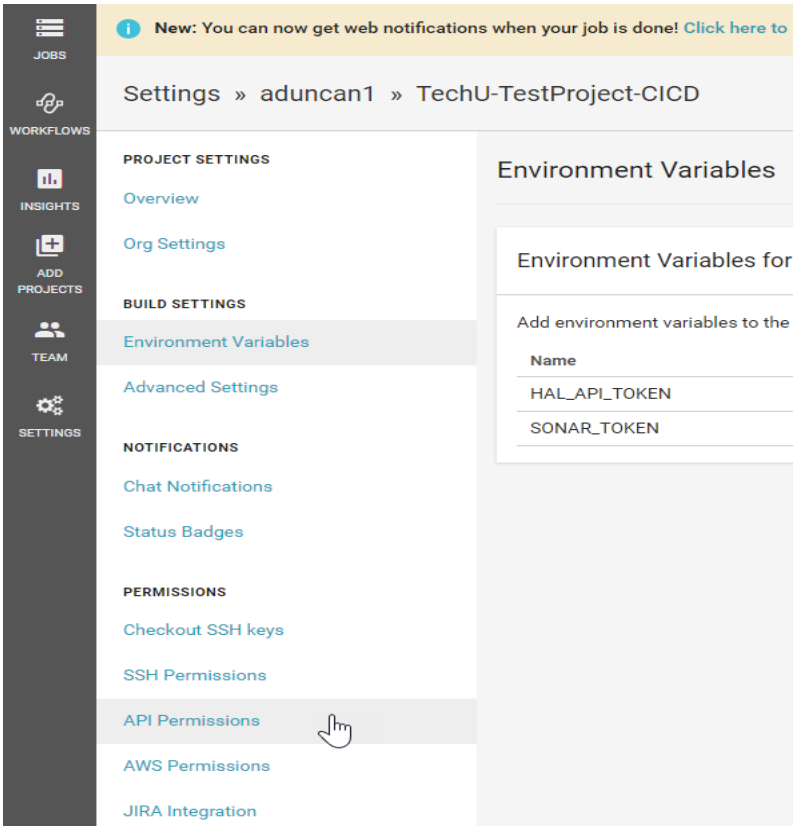


8)Back in **CircleCI**, in the **Add an Environment Variable** window, paste in the value from Hal.
9) For the **Name**, type **HAL_API_TOKEN**. This name needs to be exact or CircleCI will not know what it is.
10) Click **Add Variable**.

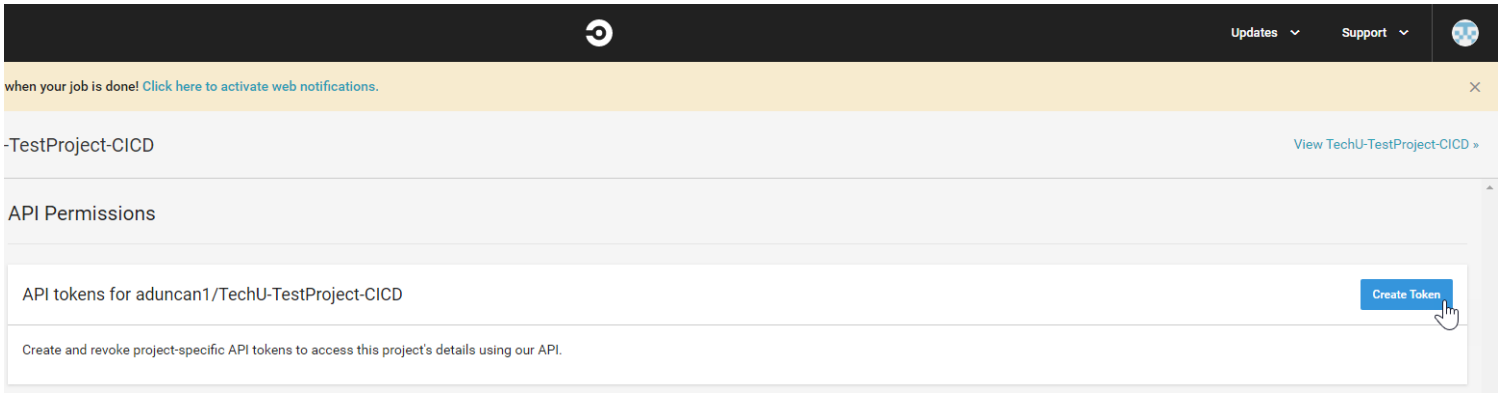


SET CIRCLECI TOKEN IN HAL

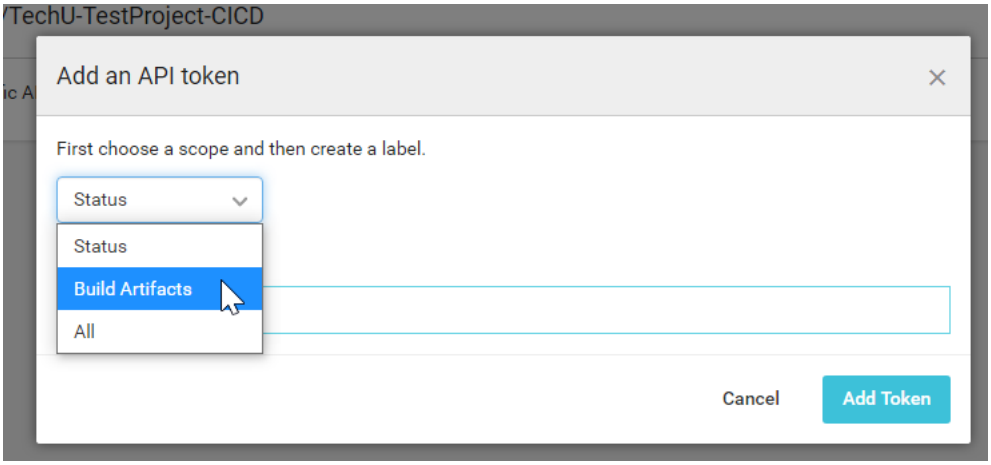
1) In the menu on the left side, scroll down under **PERMISSIONS** and click **API Permissions**.



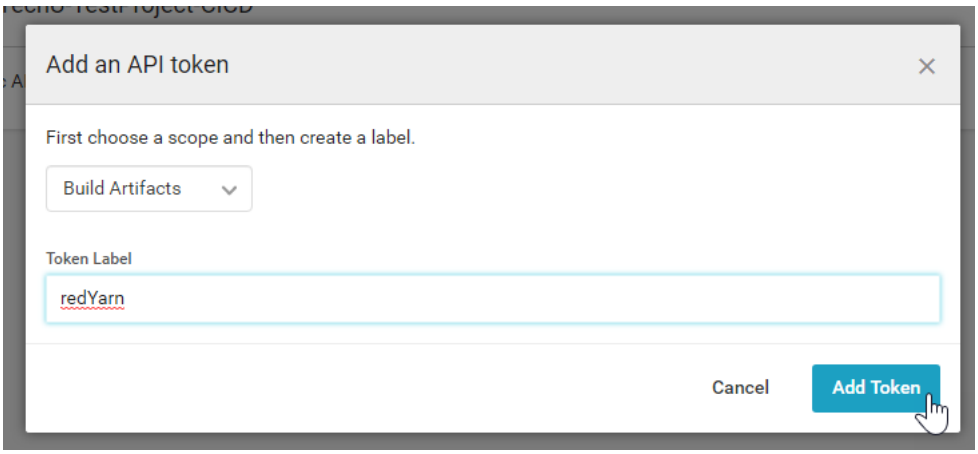
2) On the **API Permissions** page, click **Create Token**.



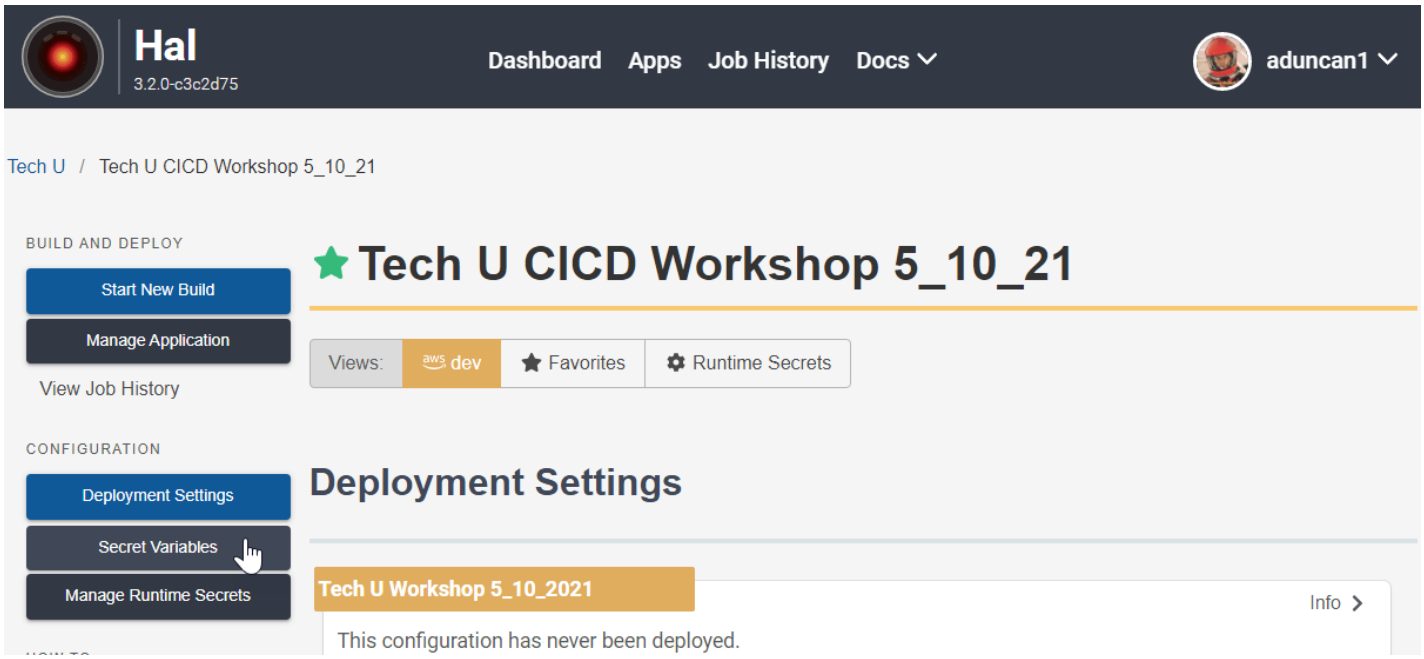
3) In the **Add an API token** window that pops-up, click the drop-down for **Status**, and select **Build Artifacts**.



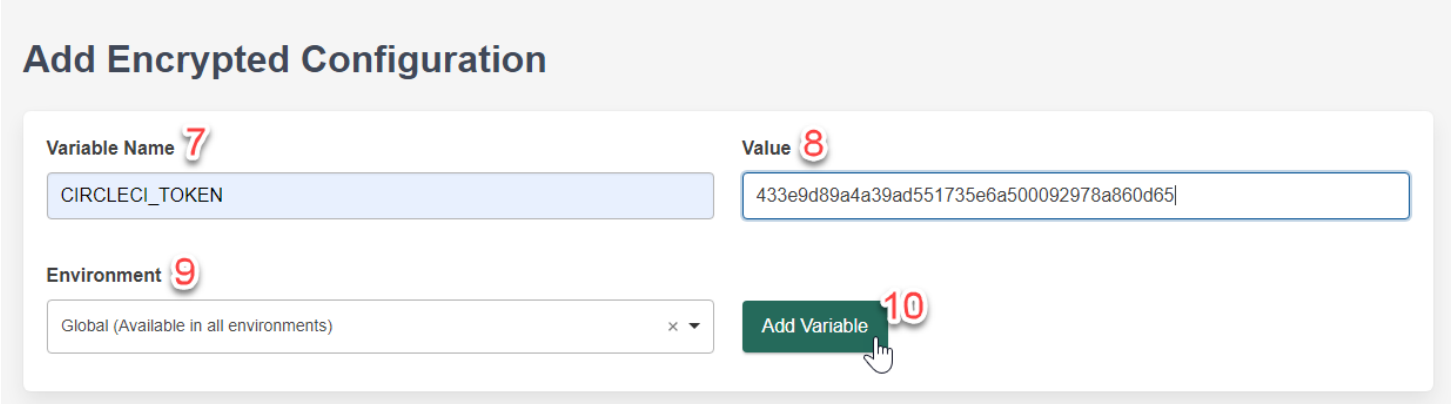
4) For the **Token Label**, you can call it anything you want.
5) Click **Add Token**.



6) In **Hal**, under **CONFIGURATION**, go to **Secret Variables**.

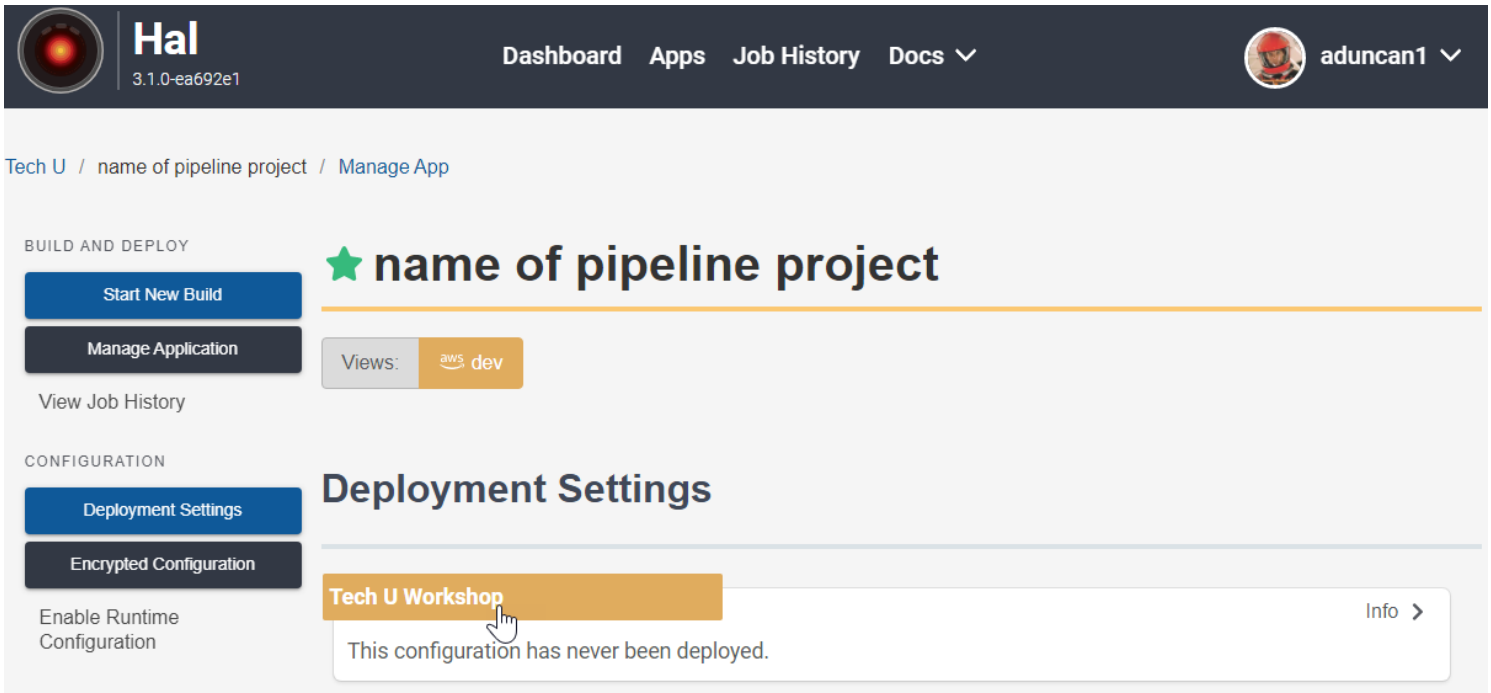


7) For the **Variable Name**, type **CIRCLECI_TOKEN**. This name needs to be exact or Hal will not recognize it.
8) For the **Value**, paste in the token value you copied from CircleCI.
9) For the **Environment**, select **Global (Available in all environments)** from the drop-down menu.
10) Click **Add Variable**.

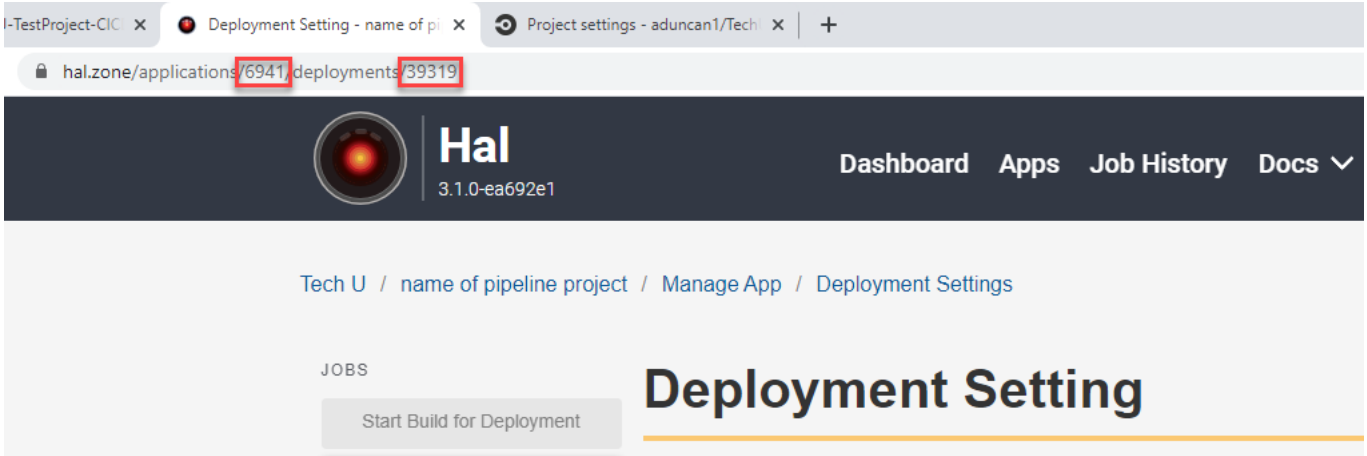


REPLACE HAL APP ID AND TARGETS

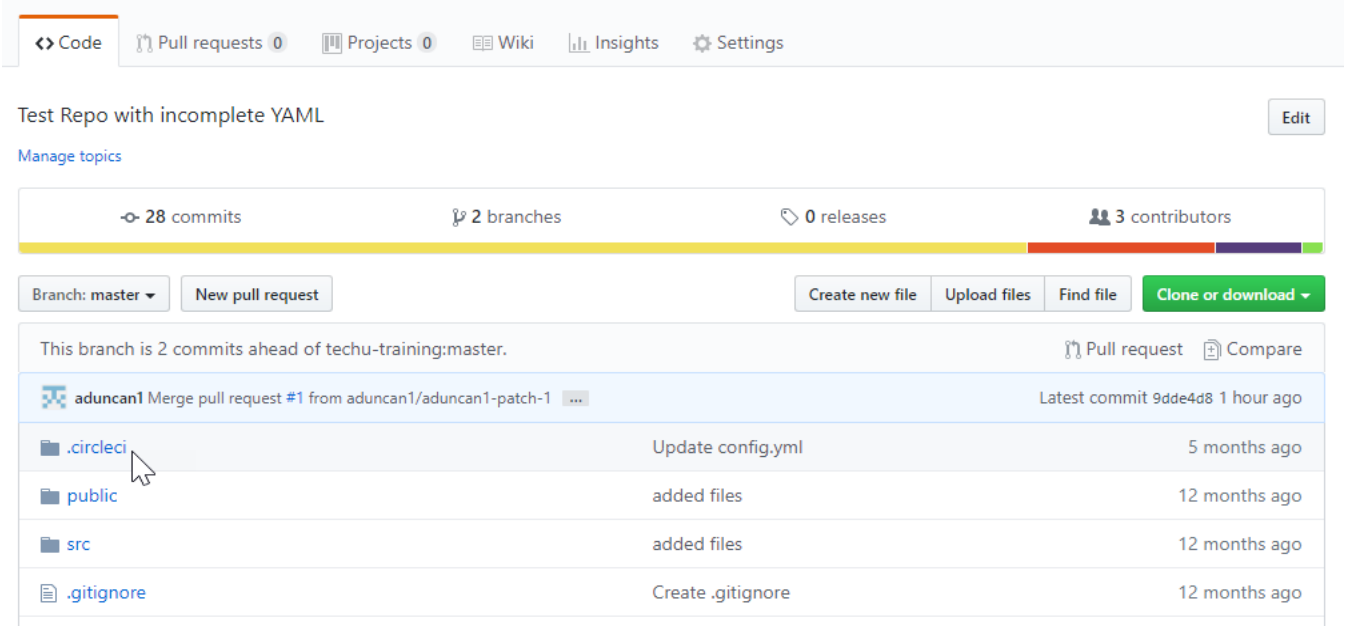
1) In **Hal**, on the main page for your app, scroll down to **Deployment Settings**, and click on the name of your project. Be sure to click the white text. Clicking the yellow box won't work.



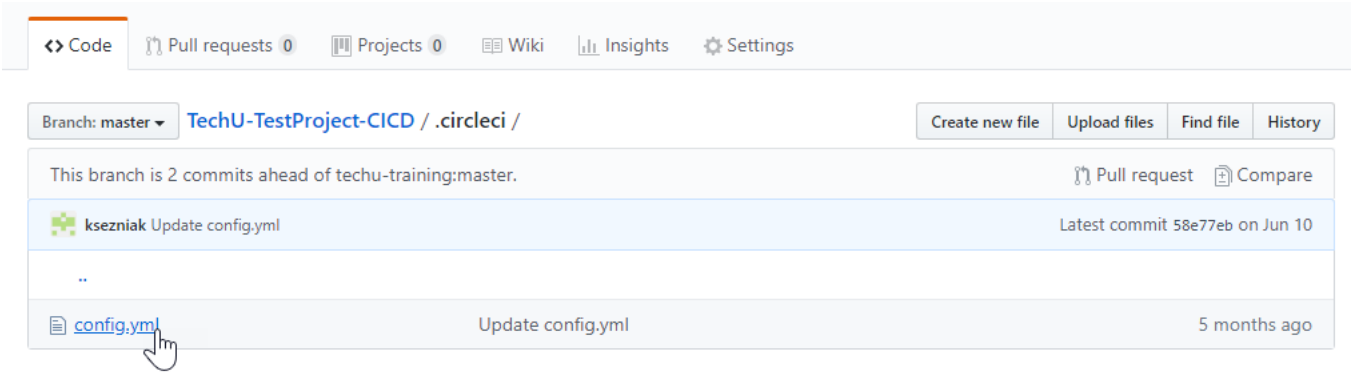
2) From this page, you can find your **Hal App ID** and **Hal deployment target ID** in the URL. The Hal App ID will be the 4-digit number, and the Hal deplyment target ID will be the 5-digit number. We will be copying and pasting these into our CircleCI YAML file in our Git repo.



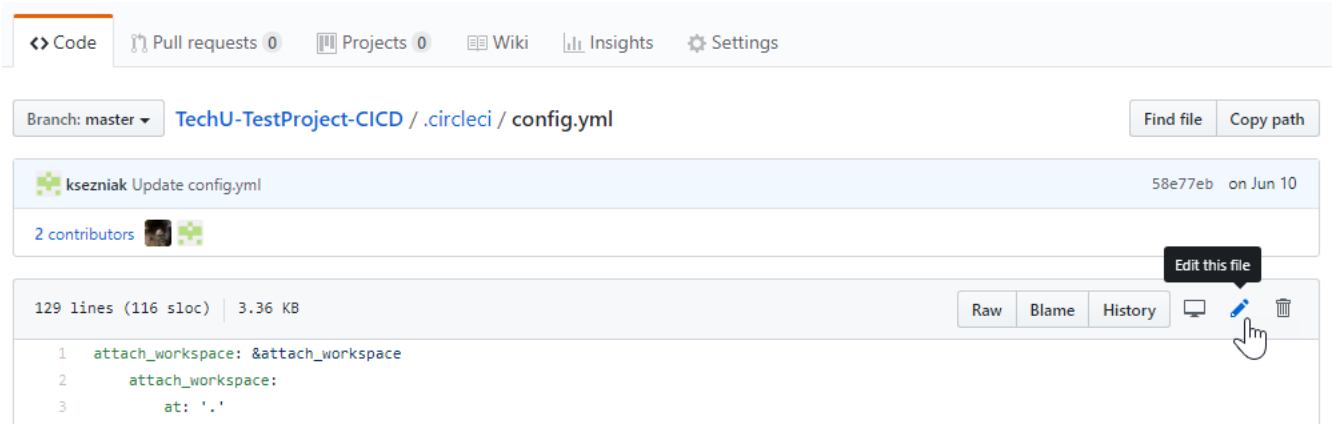
3) In **GitHub**, in your fork, open the folder called **.circleci**.



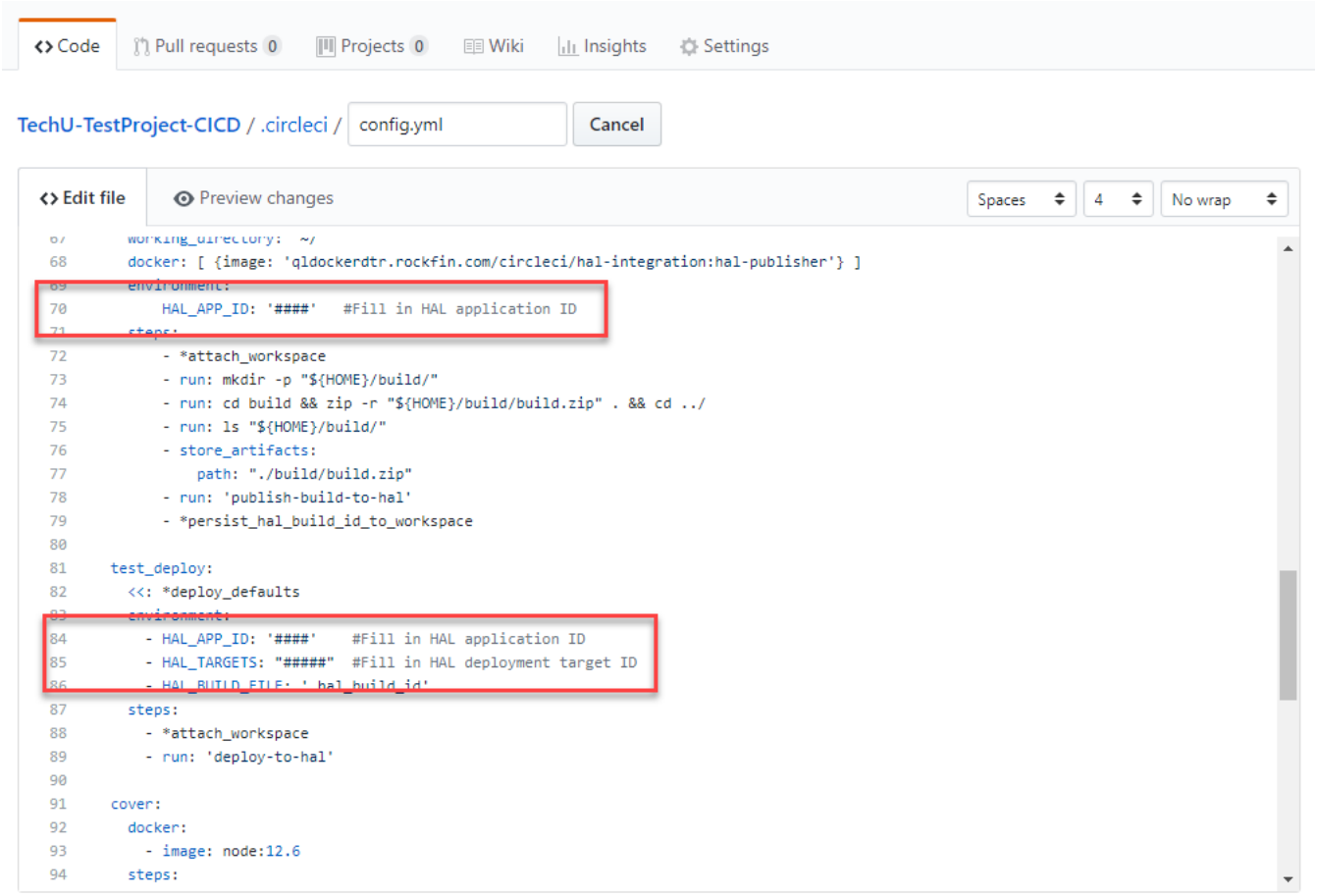
4) Open the file called **config.yml**.



5) Click the pencil icon on the right to **Edit this file**.

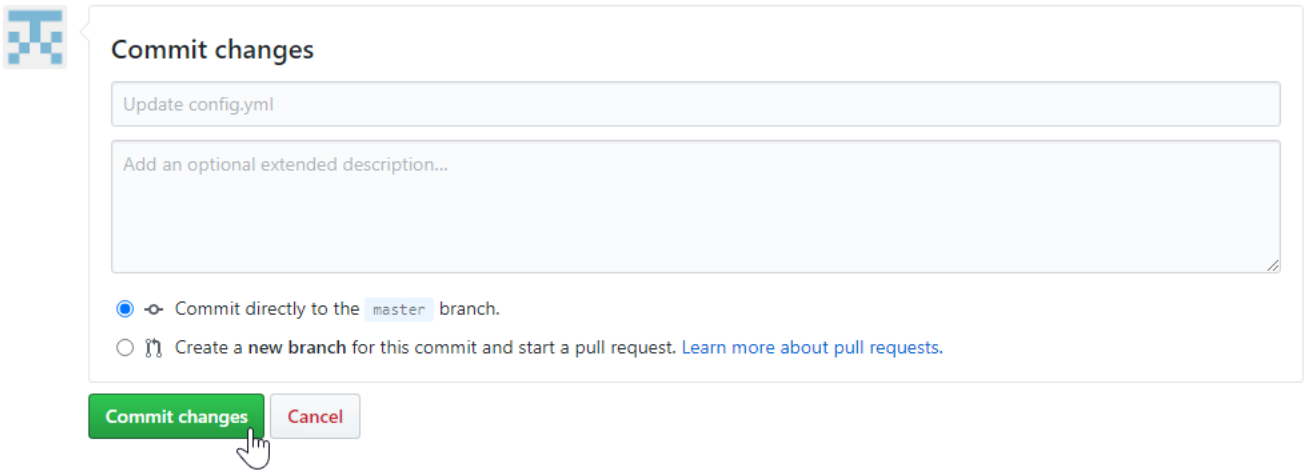


6) Scroll down to lines 70 though 85. You will be pasting in the **Hal App ID** on **line 70** and **line 84**. You will paste in the **Hal deployment target ID** on **line 85**.

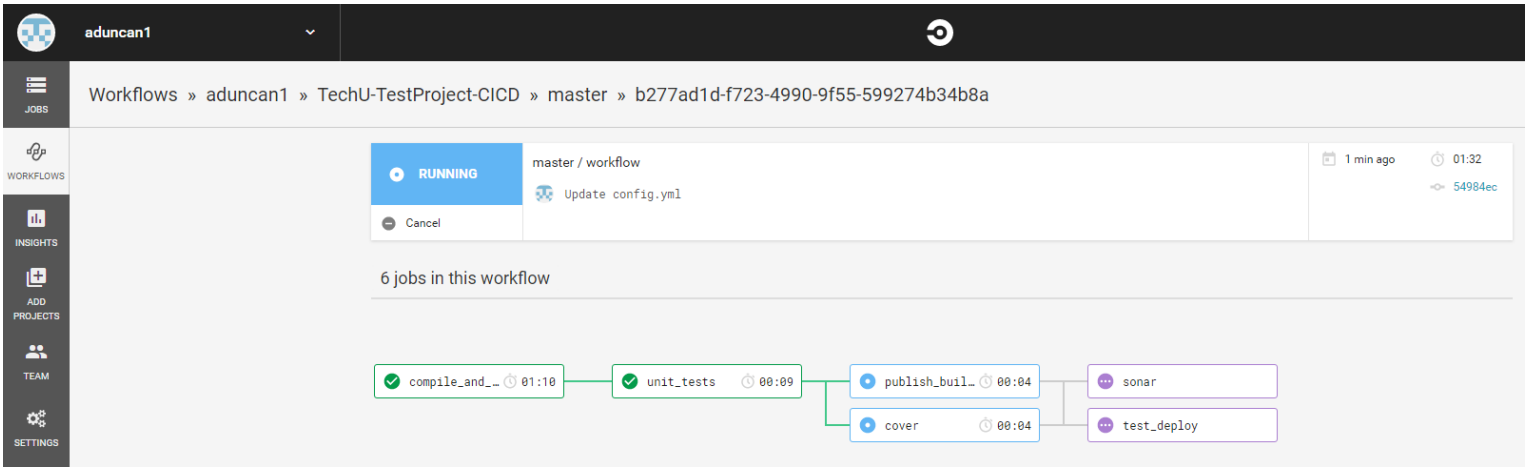


COMMIT CHANGES

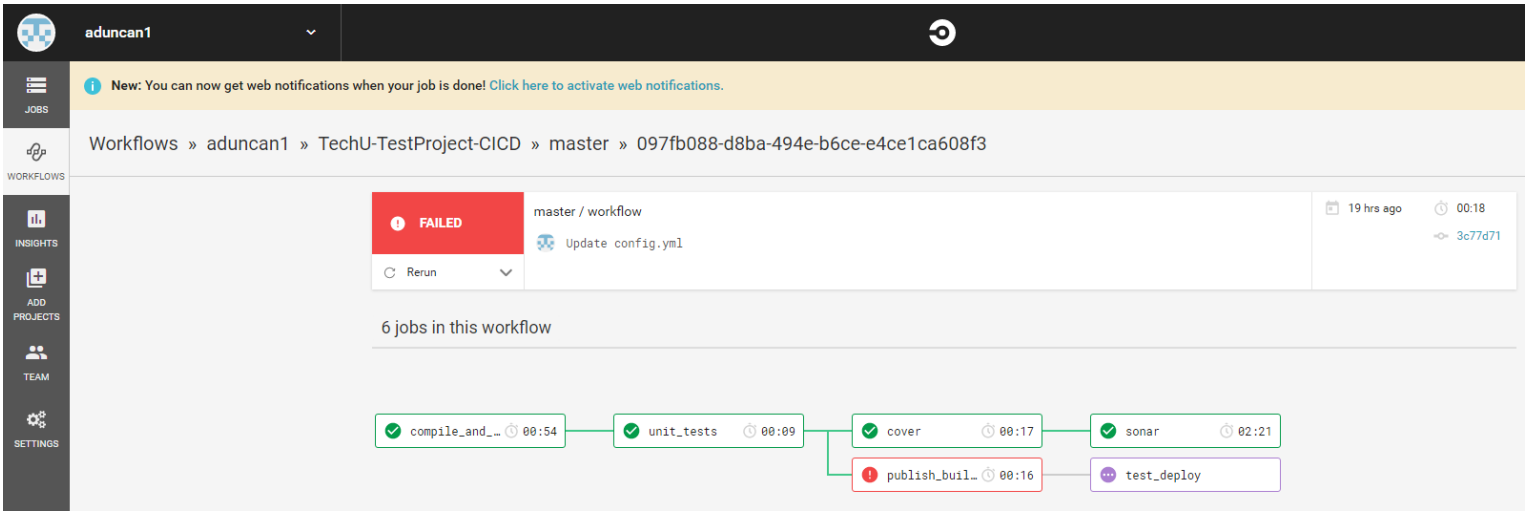
1) Click **Commit changes**.



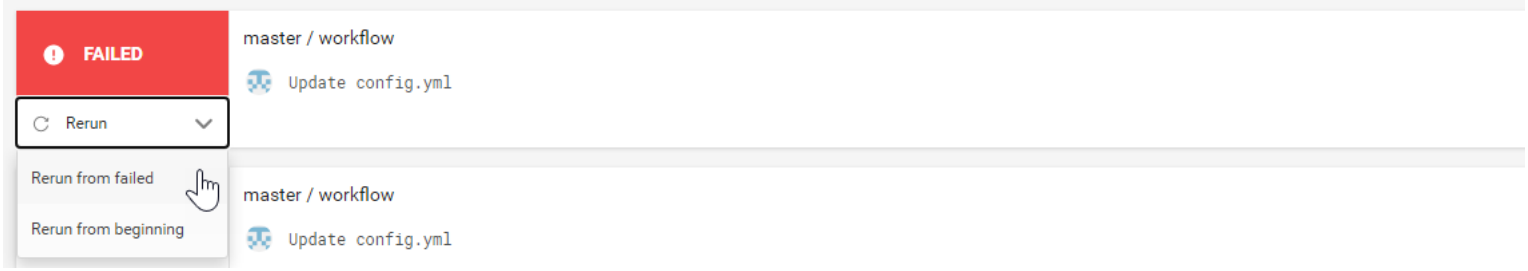
Once those changes are committed, that kicks off the pipeline to begin running again in CircleCI. You can watch it run in **CircleCI**, in **Workflows**. It can take a few minutes to run.



If the pipeline fails, you can see at what point it failed, and click on that job for more details.



Once the problem is resolved, instead of completely rerunning the build, you can **Rerun from failed**.



NOTES

CREATED BY: ALI DUNCAN | 313.545.4813