CI/CD IN GOOGLE CLOUD PLATFORM:

Introduction:

To set a CI/CD pipeline for Node js project using Bitbucket and Google cloud platform.

The tools needed to implement this pipeline are

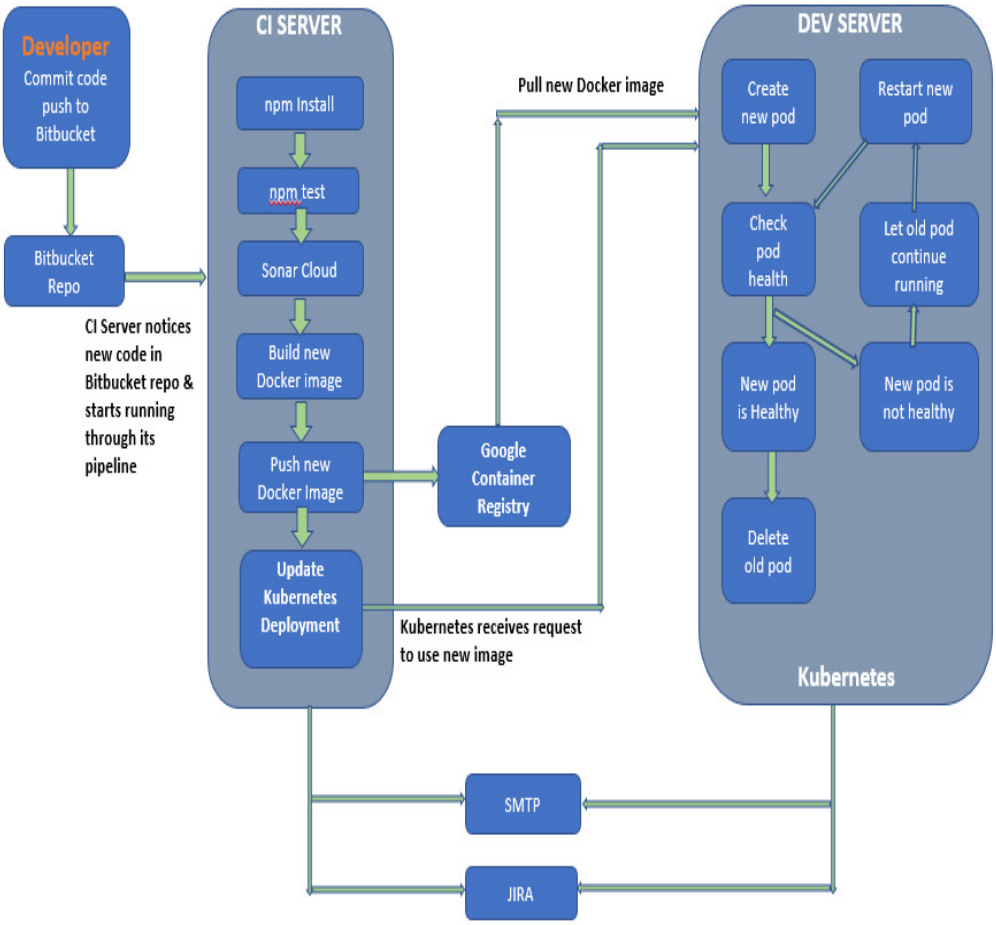
- GOOGLE CLOUD BUILD
- GOOGLE KUBERNETES ENGINE
- BITBUCKET
- NPM
- DOCKER
- GOOGLE CONTAINERS REPOSITORY
- GOOGLE CLOUD PERMISSION MANAGER (IAM & ADMIN)
- SONARCLOUD
- SONARQUBE-SCANNER
- GOOGLE CLOUD FUNCTIONS
- JIRA
- MAILGUN
- PUB/SUB

After making all the connections in the pipeline the small commit from developer into the bitbucket repository triggers the build in the google cloud build and in this the npm package is used to build and unit test the project, then after sonar cloud analysis is executed later docker builds the image and pushes the image to the google cloud repository from where the container image is deployed on to the google Kubernetes engine. In Kubernetes a load balancer and service are automatically created by Kubernetes engine.

How the pipeline works:

When a manager raises a ticket in Jira for code changes, a ticket is generated with the ticket id. Then create a branch for the new changes in the code for every commit in this branch a trigger occurs, and ci/cd is implemented which is deployed to the server after a successful build the branch can be merged with the master. The SMTP services implemented using PUB/SUB API, Mailgun and google cloud functions helps to notify the build success/failure through a mail. If the mail is success, then the branch can be merged with the master by manager.

ARCHITECTURE:



SOURCES:

Node Js project which was cloned to use for testing is

https://bitbucket.org/sonarsource/sample-nodejs-project

the project has a base project structure for implementing in ci/cd for this which we must add the files like sonar-project.js for sonar cloud integration test-report.xml for test report analysis.

For the project to be integrated in gcp we required CloudBuild.Yaml file and for a node Js project the package.json is most important as it has all the scripts to be called in cloud build

TOOLS USED TO BUILD THIS PIPELINE:

GOOGLE CLOUD BUILD

It is used to link the source code repository from GitHub/bitbucket/google cloud source repositories and helps to add triggers(webhooks). So, when a code is pushed into source code repository by a developer it automatically starts a build and uses cloud config file in the source code repository.

https://cloud.google.com/cloud-build

GOOGLE KUBERNETEES ENGINE

It is used to deploy the containers after the application is built using npm and docker. The built application is uploaded to the google cloud container repository from there the google cloud build deploys to the Kubernetes cluster.

https://cloud.google.com/kubernetes-engine

BITBUCKET

This is the tool used to store the source code repository and helps to trigger the builds when a code is pushed to the repository by a developer.

https://github.com

NPM

This is the tool used in the pipeline to build the project using npm components like install, build and test.

```
gcr.io/cloud-builders/npm is the tool representation in gcp
```

DOCKER

The docker in gcp helps us to build an image of the application and helps us to push it to the container repository in gcp.

https://github.com/GoogleCloudPlatform/cloud-builders/tree/master/docker

```
gcr.io/cloud-builders/docker is the tool representation in gcp.
```

GOOGLE CONAINERS REPOSITORY

This helps to store the container images of the project that were built by docker and uploaded to this repository.

https://cloud.google.com/container-registry

GOOGLE CLOUD PERMISSION MANAGER (IAM & ADMIN)

Cloud IAM lets administrators authorize who can act on specific resources, giving you full control and visibility to manage cloud resources centrally.

https://cloud.google.com/iam

SONARCLOUD

Sonar Cloud is the leading product for Continuous Code Quality online, totally free for open-source projects. It supports all major programming languages, including Java, C#, JavaScript, TypeScript, C/C++ and many more.

https://sonarcloud.io/

SONARQUBE-SCANNER

This the tool used to perform static code analysis and push the results to the sonar cloud.

GOOGLE CLOUD FUNCTIONS

Cloud Functions is Google Cloud's event-driven serverless compute platform. Run your code locally or in the cloud without having to provision servers. Go from code to deploy with continuous delivery and monitoring tools.

https://cloud.google.com/functions

JIRA

Jira is a proprietary issue tracking product developed by Atlassian that allows bug tracking and agile project management.

https://www.atlassian.com/software/jira

MAILGUN

Mailgun is an email automation service provided by Rack space. It offers a complete cloud-based email service for sending, receiving and tracking email sent through your websites and applications. Mail gun features are available through an intuitive RESTful API or using traditional email protocols like SMTP.

https://www.mailgun.com/

PUB/SUB API

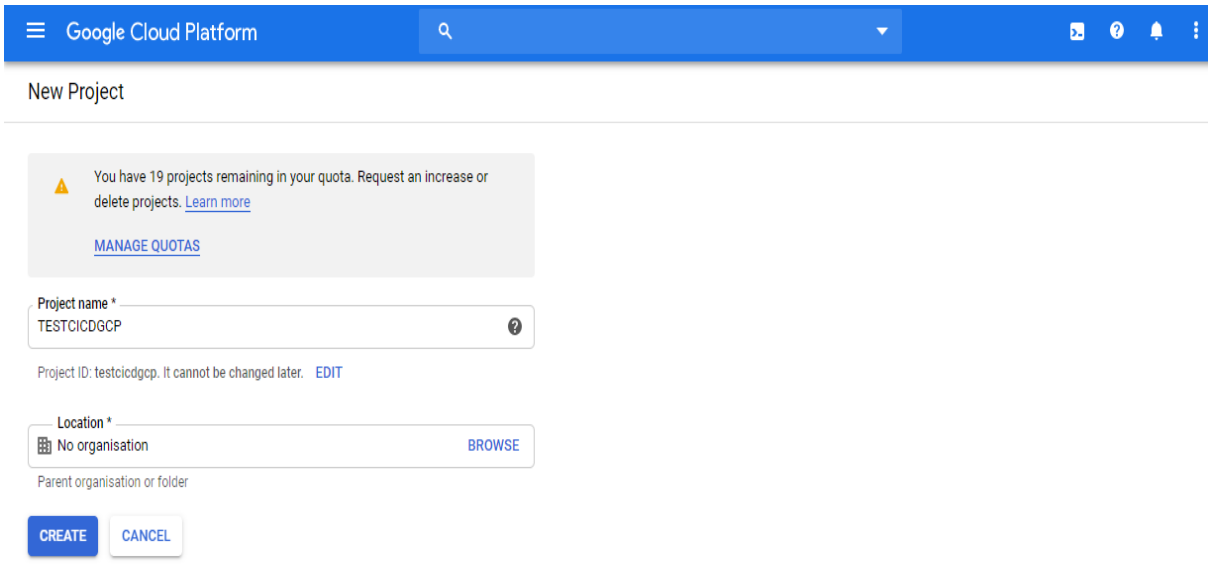
Cloud Pub/Sub is a fully managed real-time messaging service that allows you to send and receive messages between independent applications.

https://cloud.google.com/pubsub/

SETTING PIPELINE IN GCP:

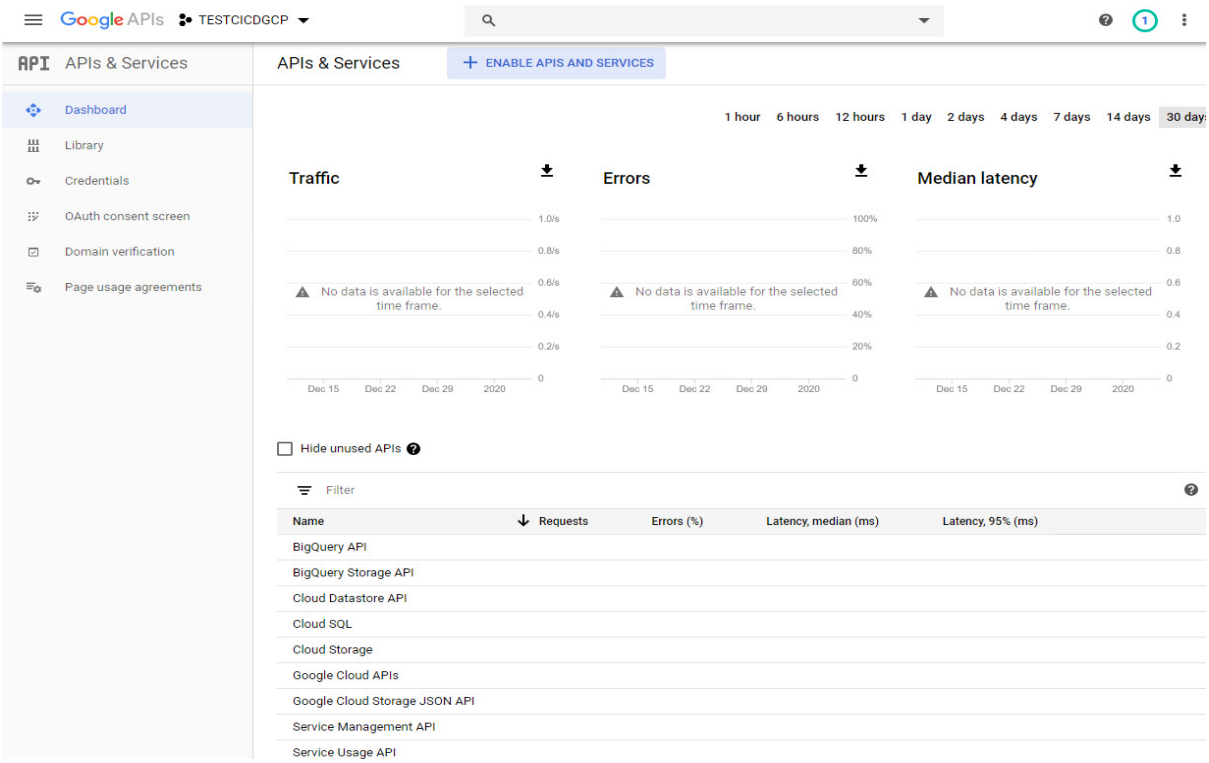
STEP1:

CREATE A NEW PROJECT IN THE GOOGLE CONSOLE (https://console.cloud.google.com).



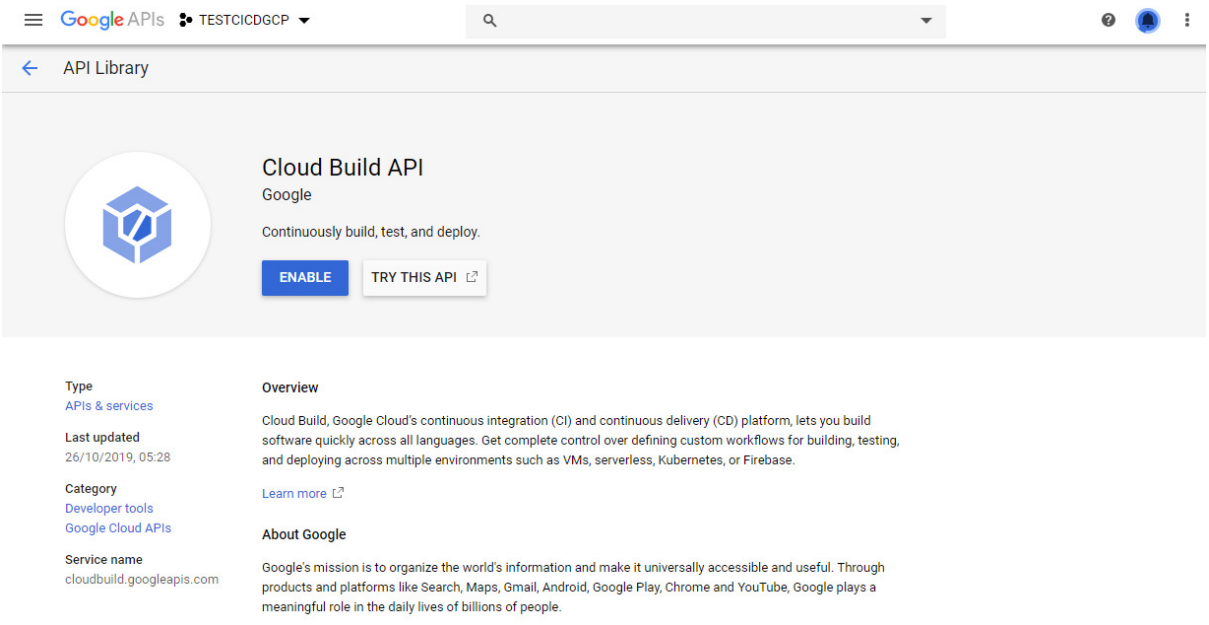
STEP2:

NOW GO TO THE APIS & SERVICES DASHBOARD IN THE SAME PROJECT AND CLICK ON ENABLE APIS AND SERVICES



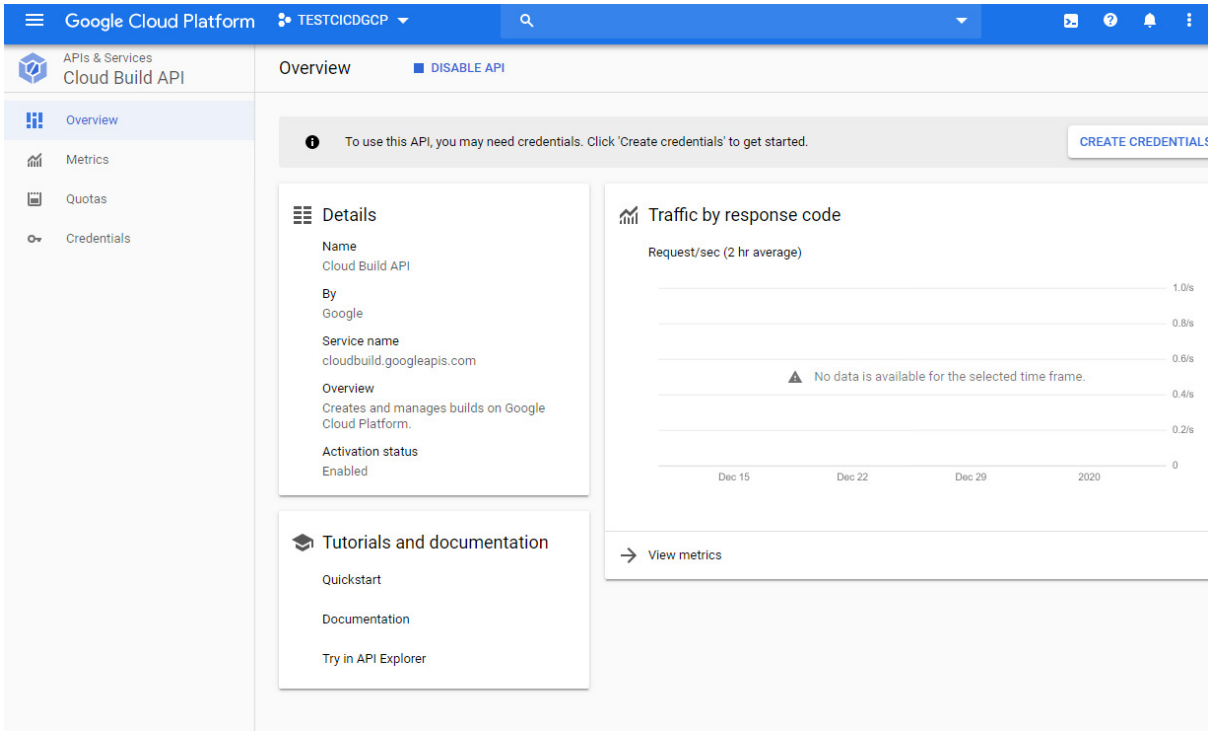
STEP3:

SEARCH FOR CLOUD BUILD API AND CLICK ON ENABLE



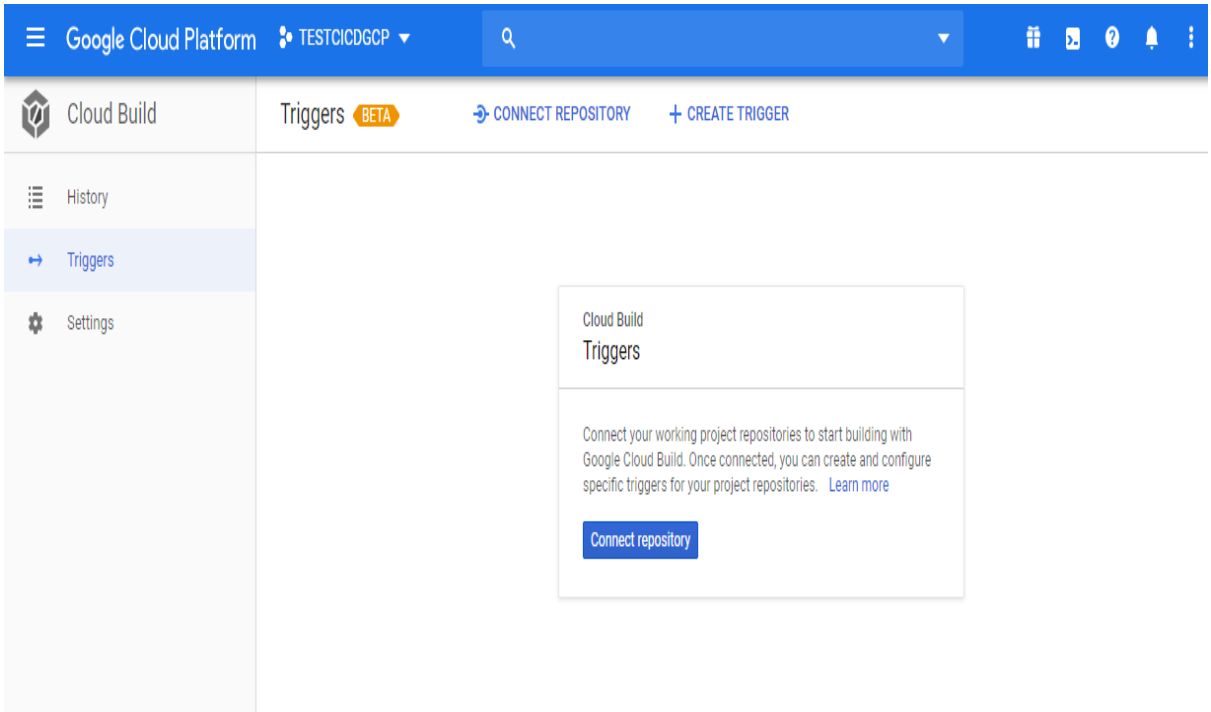
STEP4:

YOU CAN SEE THAT THE API IS ENABLED AND BILLING STARTED FOR THE USAGE.



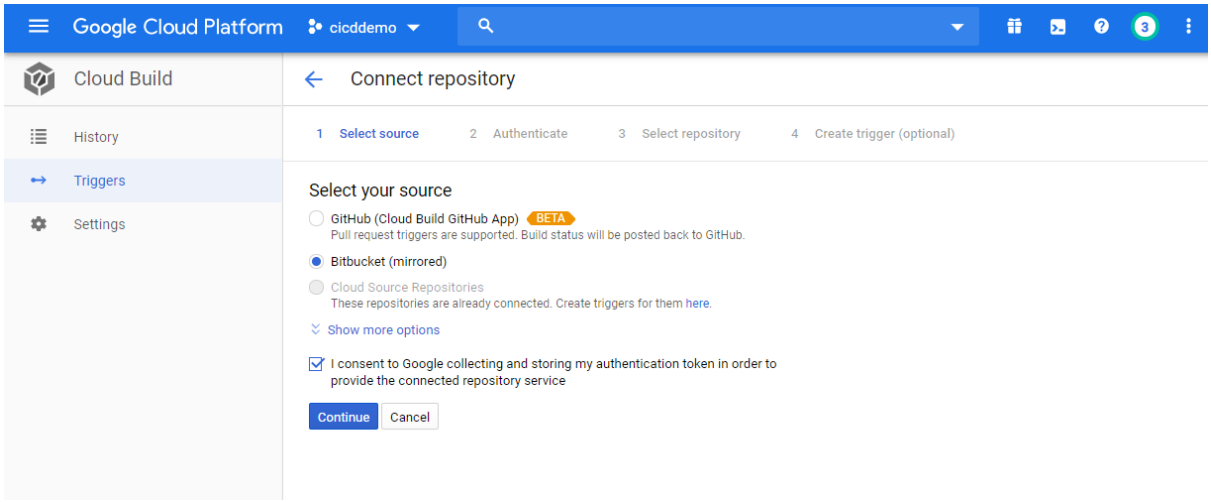
STEP5:

GO TO TRIGGERS IN CLOUD BUILD AND SELECT CONNECT REPOSITORY



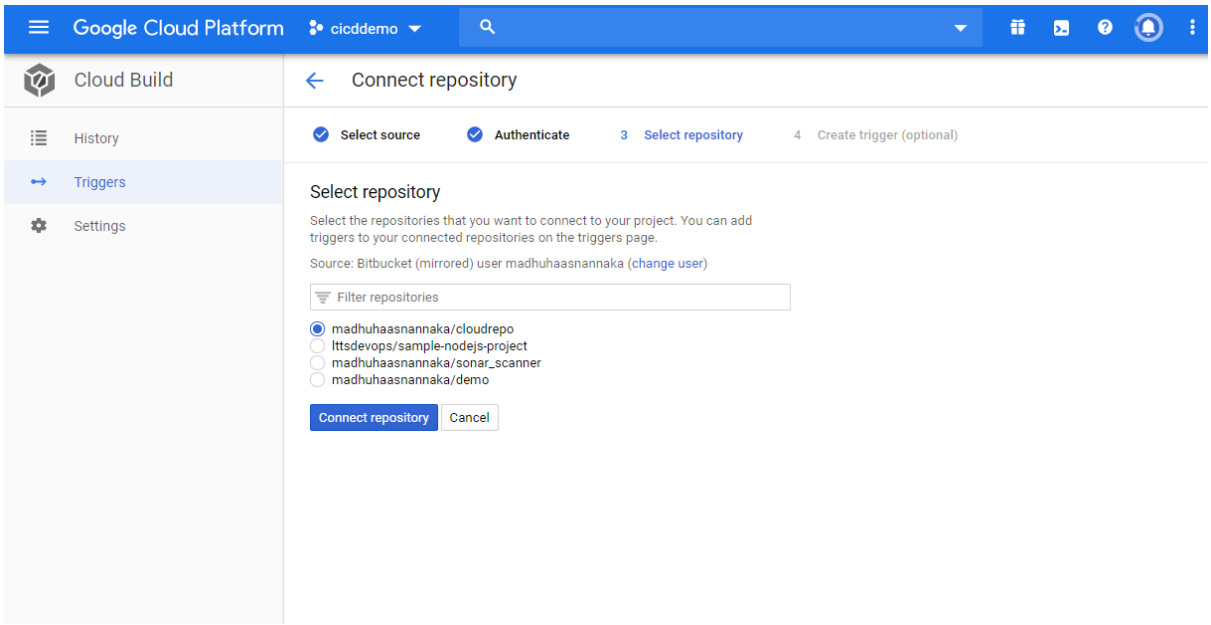
STEP6:

AND IN HERE SELECT AS A SOURCE AS BITBUCKET



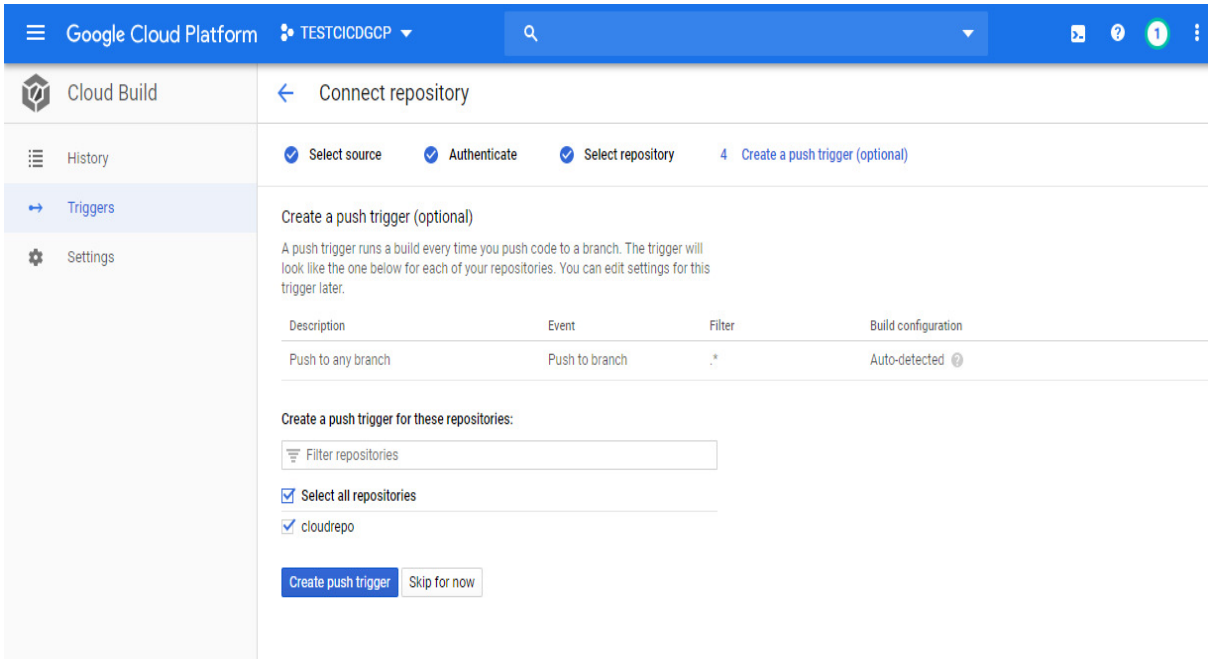
STEP7:

AS MY BITBUCKET IS ALREADY ATHUNTICATED IT DIRECTLY SHOWS THE REPOSITORIES IN MY ACCOUNT NOW WE HAVE TO SELECT THE REPOSITORY FOR SOURCE CODE



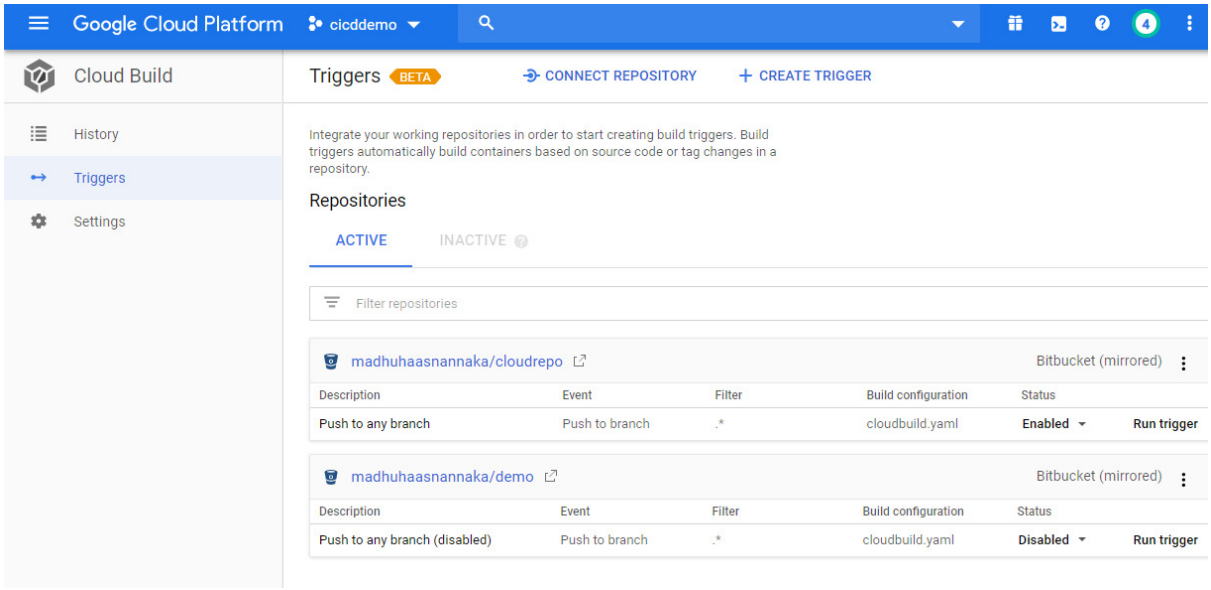
STEP8:

NOW CREATE A PUSH TRIGGER SO THAT WHENEVER A COMMIT OCCURRED IN SOURCE CODE THE BUILD STARTS AUTOMATICALLY



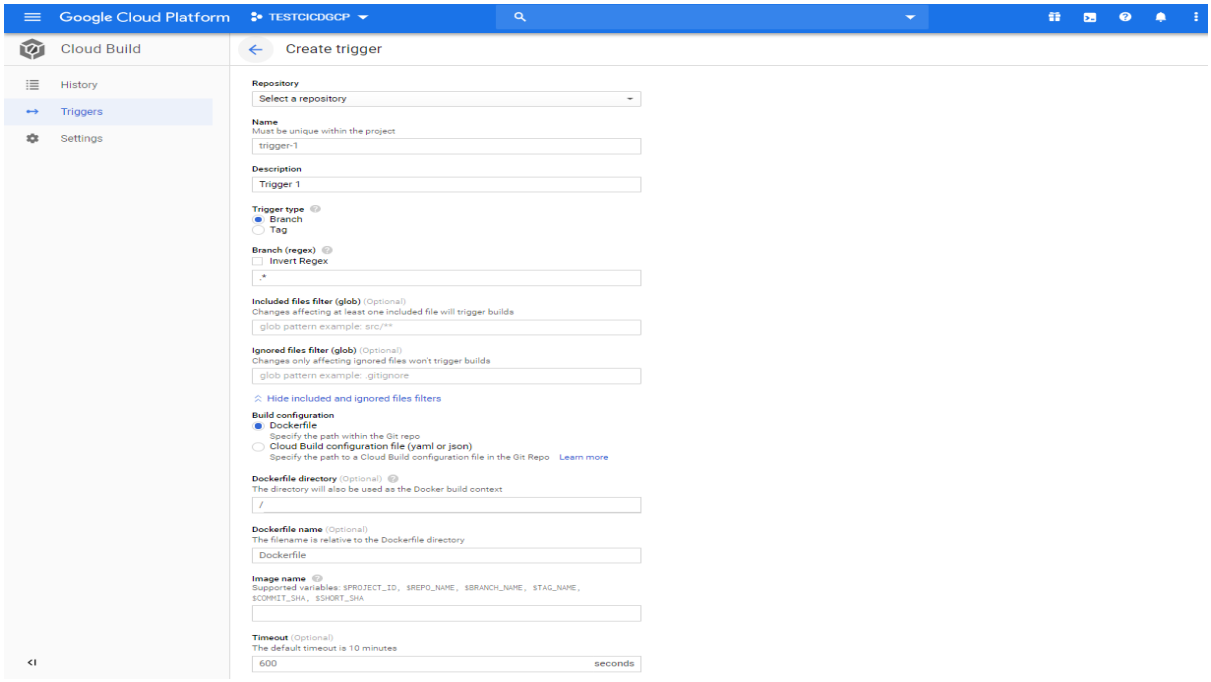
STEP9:

WE CAN SEE THAT THE TRIGGER IS ALSO ASSIGNED TO THE REPOSITORY



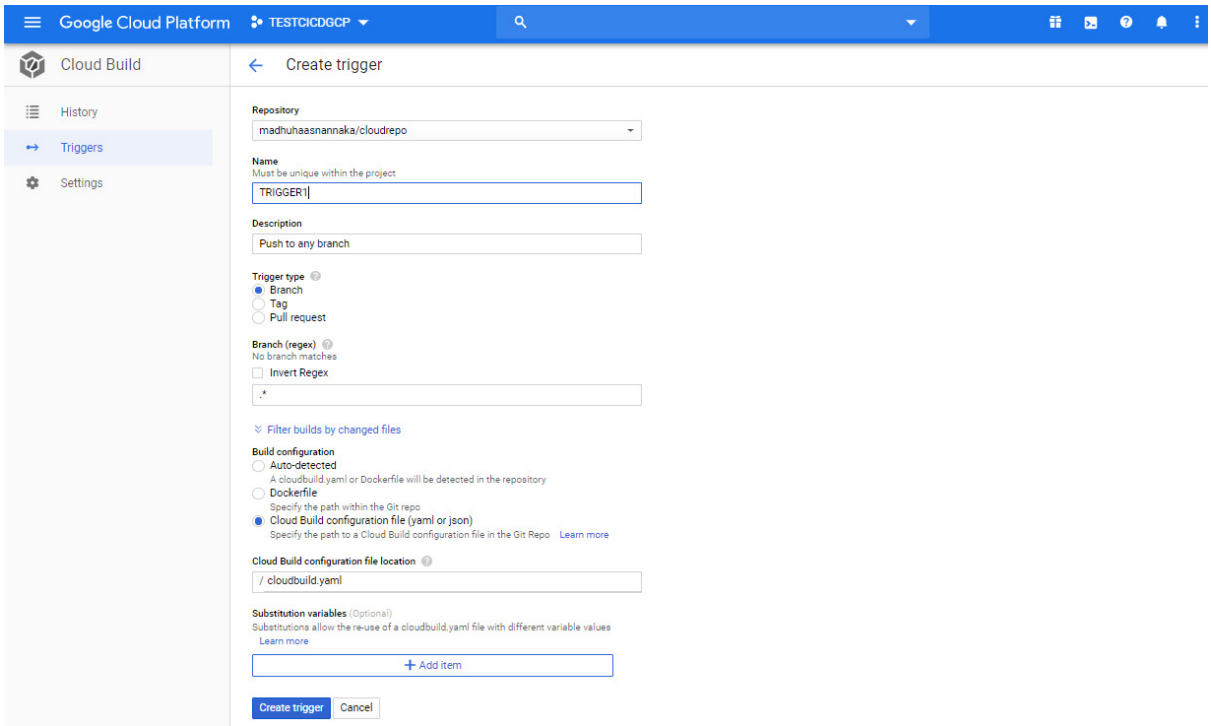
STEP10:

BUT IF YOU HAVE NOT SELECTD THE ENABLE TRIGGER IN THE CONNECT REPOSITORY FLOW, TO ENABLE THE TRIGGER CLICK ON CREATE TRIGGER



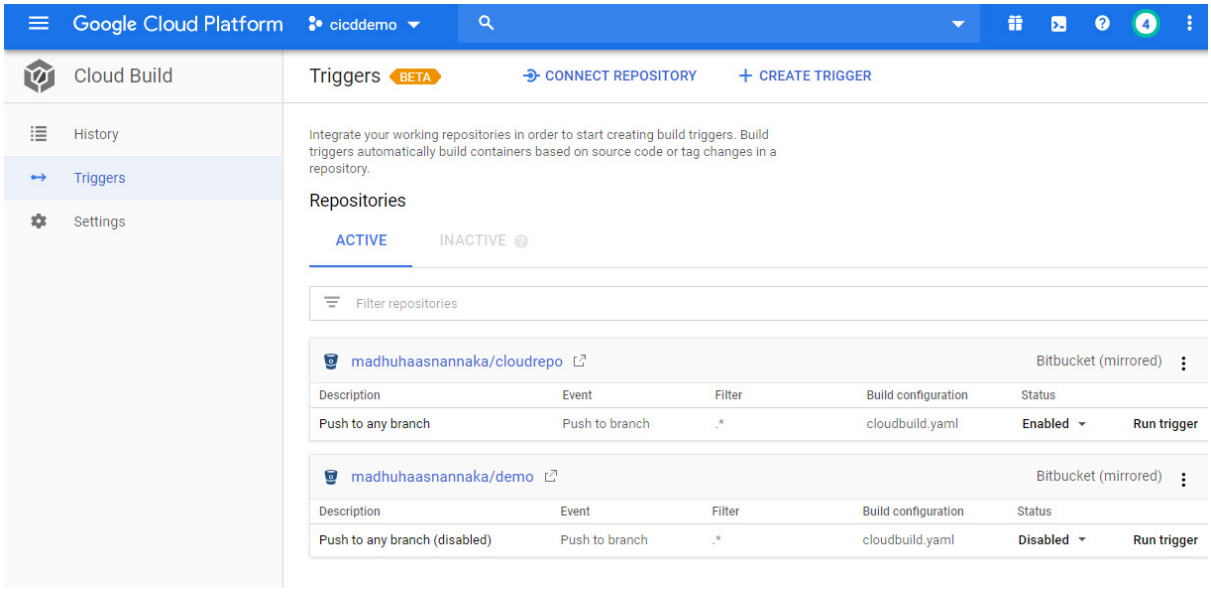
STEP11:

AND HERE SELECT THE REPOSITORY AND IN BUILD CONFIGURATION SELECT THE BUILD CONFIGURATION FILE WHICH IS (.YAML) FILE AND CLICK ON CREATE TRIGGER



STEP12:

WE CAN SEE THE SAME AS BEFORE BUT TRIGGER IS GENERATED



STEP13:

WRITE A BUILD CONFIGURATION FILE (cloudbuild. yaml) AND KEEP IT IN THE SOURCE CODE REPOSITORY

LET'S HAVE A LOOK AT THE FILE

https://bitbucket.org/madhuhaasnannaka/cloudrepo/src/master/cloudbuild.yaml

STEPS IN THE FILE INCLUDE:

I)USING NPM TO PACKAGE INSTALL, BUILD AND TEST

```
steps:

- name: 'gcr.io/cloud-builders/npm'
  id: installing_npm
  args: ['install']

- name: 'gcr.io/cloud-builders/npm'
  id: clean
  args: ['run','cache']

- name: 'gcr.io/cloud-builders/npm'
  id: building_with_npm
  args: ['run','build']


- name: 'gcr.io/cloud-builders/npm'
  id: unit_testing
  args: ['install','-g','karma-cli','run','test']
```

II)TO INSTALL SONARQUBE SCANNER AND RUNNING IT

```
- name: 'gcr.io/cloud-builders/npm'
  id: install sonar-scanner
  args: ['install','sonarqube-scanner']

- name: 'gcr.io/cloud-builders/npm'
  id: install typescript
  args: ['install','-D','typescript@latest']

- name: 'gcr.io/cloud-builders/npm'
  id: sonarpush
  args: ['run','sonar']
```

III)USING DOCKER TO BUILD AN IMAGE

```
- name: 'gcr.io/cloud-builders/docker'
  id: building_image_using_docker
  args: ["build", "-t", "gcr.io/$PROJECT_ID/nodej:$COMMIT_SHA", "-t", "gcr.io/$PROJECT_ID/nodej:latest", "."]
```

IV)NOW PUSHING BUILT IMAGE TO THE CLOUD IMAGE CONTAINER REPOSITORY

```
- name: 'gcr.io/cloud-builders/docker'
  id: pushing_image_gcr
  args: ["push", "gcr.io/$PROJECT_ID/nodej:$COMMIT_SHA"]
- name: 'gcr.io/cloud-builders/docker'
  id: pushing_container_image_gke
  args: ["push", "gcr.io/$PROJECT_ID/nodej:latest"]
```

IV)DEPLOYING IMAGE TO THE KUBERNETES

```
- name: 'gcr.io/cloud-builders/gke-deploy:stable'
  id: deploying_container_gke
  args:
  - run
  - --image=gcr.io/$PROJECT_ID/nodej:latest
  - --location=us-east4-c
  - --cluster=standard-cluster-1
  - --app=nodej
  - --expose=8080
  - --namespace=default
options:
    machineType: 'N1_HIGHCPU_8'
```

CODE FOR THE cloudbuild.yaml INCLUDES AS BELOW:

```
steps:
- name: 'gcr.io/cloud-builders/npm'
  id: clean
  args: ['run','cache']
- name: 'gcr.io/cloud-builders/npm'
  id: installing_npm
  args: ['install']
- name: 'gcr.io/cloud-builders/npm'
  id: building_with_npm
  args: ['run','build']
- name: 'gcr.io/cloud-builders/npm'
  id: unit_testing
  args: ['install','-g','karma-cli','run','test']
- name: 'gcr.io/cloud-builders/npm'
  id: install sonar-scanner
  args: ['install','sonarqube-scanner']
- name: 'gcr.io/cloud-builders/npm'
  id: install typescript
  args: ['install','-D','typescript@latest']
- name: 'gcr.io/cloud-builders/npm'
  id: sonarpush
  args: ['run','sonar']
- name: 'gcr.io/cloud-builders/docker'
  id: building_image_using_docker
  args: ["build", "-t", "gcr.io/$PROJECT_ID/nodej:$COMMIT_SHA", "-t",
"gcr.io/$PROJECT_ID/nodej:latest", "."]
- name: 'gcr.io/cloud-builders/docker'
  id: pushing_image_gcr
  args: ["push", "gcr.io/$PROJECT_ID/nodej:$COMMIT_SHA"]
- name: 'gcr.io/cloud-builders/docker'
  id: pushing_container_image_gke
  args: ["push", "gcr.io/$PROJECT_ID/nodej:latest"]
- name: 'gcr.io/cloud-builders/gke-deploy:stable'
  id: deploying_container_gke
  args:
  - run
  - --image=gcr.io/$PROJECT_ID/nodej:latest
  - --location=us-central1-a
  - --cluster=standard-cluster-4
  - --app=nodej
  - --expose=8080
  - --namespace=default
options:
    machineType: 'N1_HIGHCPU_8'
```

STEP14:

TO CONFIGURE THE SONARCLOUD FOR THIS ADD sonar-project.js WHICH IS

https://bitbucket.org/madhuhaasnannaka/cloudrepo/src/master/sonar-project.js

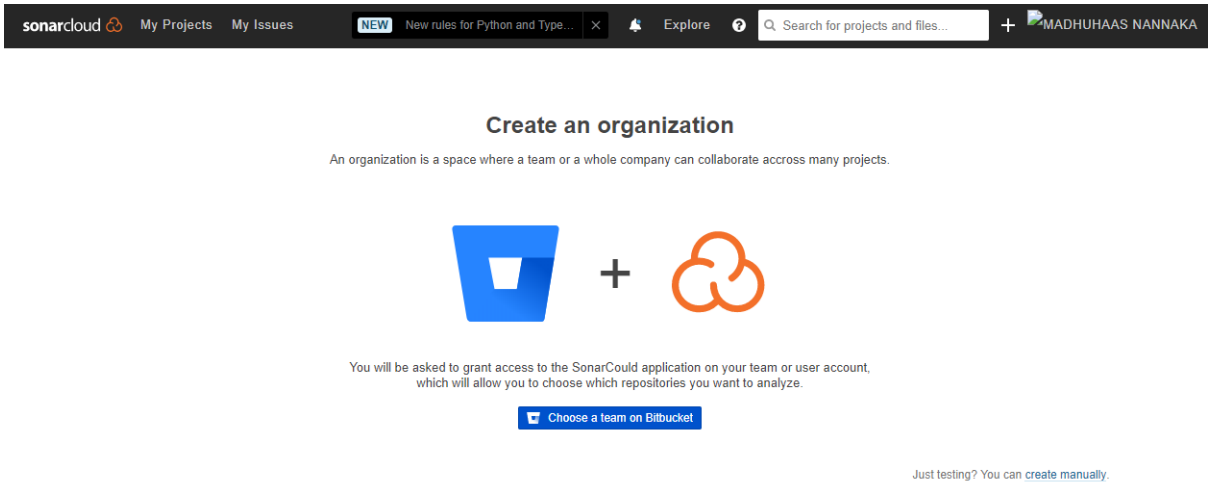
```
const sonarqubeScanner = require('sonarqube-scanner');
      sonarqubeScanner({
        serverUrl: 'https://sonarcloud.io/',
        token : '4ea57303c832f76242362c6cd9c9a73e63c27016',
        options : {
        'sonar.login': '4ea57303c832f76242362c6cd9c9a73e63c27016',
        'sonar.organization': 'lttsdevops',
        'sonar.projectKey': 'lttsdevops_sample-nodejs-project',
        'sonar.projectVersion': '1.0',
        'sonar.language': 'js',
        'sonar.sourceEncoding': 'UTF-8',
        //'sonar.sources': '.',
        'sonar.inclusions' : '**', // Entry point of your code
        'sonar.exclusions': '*.test.*',
        'sonar.tests': '.',
        'sonar.test.inclusions': '**/testing/**,**/*.spec.ts',
        'sonar.javascript.lcov.reportPaths': 'coverage/lcov.info',
        'sonar.testExecutionReportPaths': 'test-report.xml'


        }
      }, () => {});
```

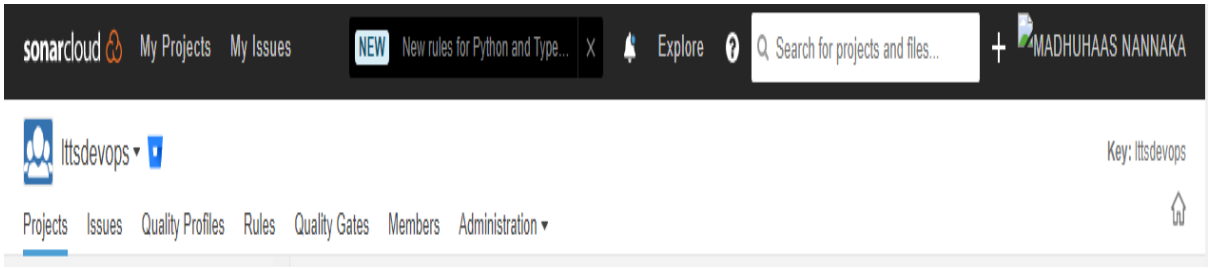
THE TOKEN, ORGANIZATION KEY AND PROJECT KEY MUST BE COPIED FROM SONAR CLOUD.

FOR THIS THE STEPS AS FOLLOWS:

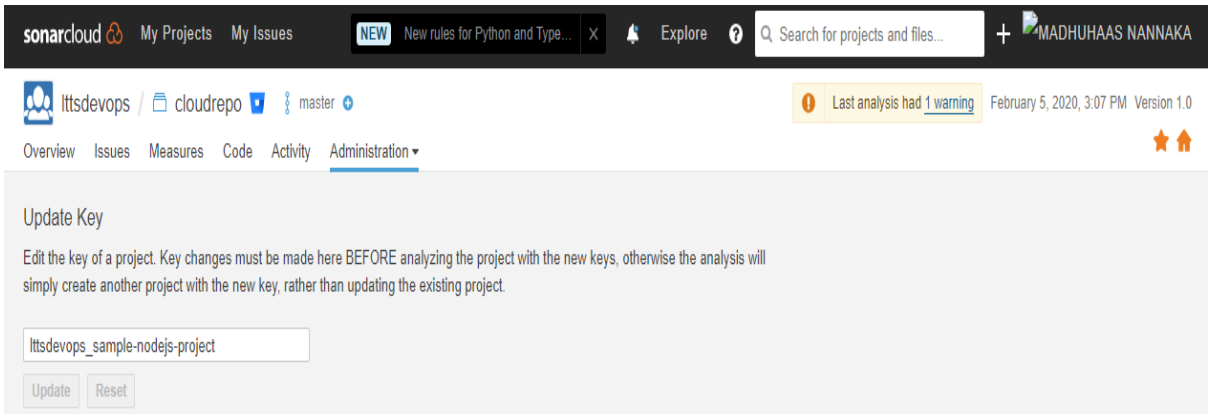
I)CREATE A NEW ORGANIZATION IN THE SONARCLOUD AND SELECT A TEAM FROM THE BITBUCKET



II)AFTER YOU LINK WITH THE BITBUCKET AND SELECTING THE PROJECT GET THE ORGANIZATION TOKEN HERE



III)YOU CAN GET THE PROJECT KEY IN PROJECT ADMINISTARTION UPDATE KEYS OPTION



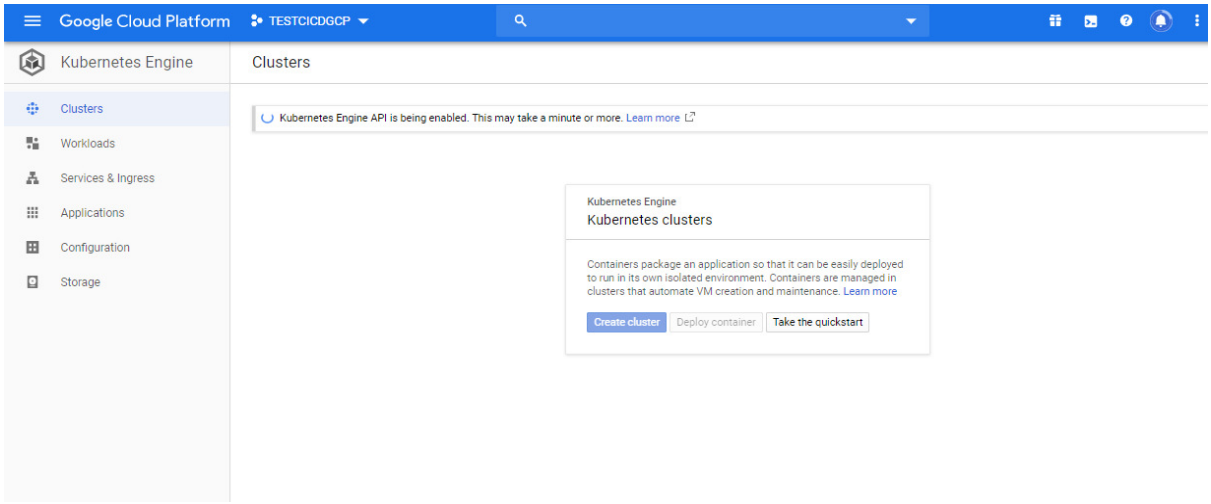
CODE FOR THE sonar-project.js INCLUDES AS BELOW:

```
const sonarqubeScanner = require('sonarqube-scanner');
     sonarqubeScanner({
       serverUrl: 'https://sonarcloud.io/',
       token : '4ea57303c832f76242362c6cd9c9a73e63c27016',
       options : {
       'sonar.login': '4ea57303c832f76242362c6cd9c9a73e63c27016',
       'sonar.organization': 'lttsdevops',
       'sonar.projectKey': 'lttsdevops_sample-nodejs-project',
       'sonar.projectVersion': '1.0',
       'sonar.language': 'js',
       'sonar.sourceEncoding': 'UTF-8',
       //'sonar.sources': '.',
       'sonar.inclusions' : '**', // Entry point of your code
       'sonar.exclusions': '*.test.*',
       'sonar.tests': '.',
       'sonar.test.inclusions': '**/testing/**,**/*.spec.ts',
       'sonar.javascript.lcov.reportPaths': 'coverage/lcov.info',
       'sonar.testExecutionReportPaths': 'test-report.xml'


       }
     }, () => {});
```

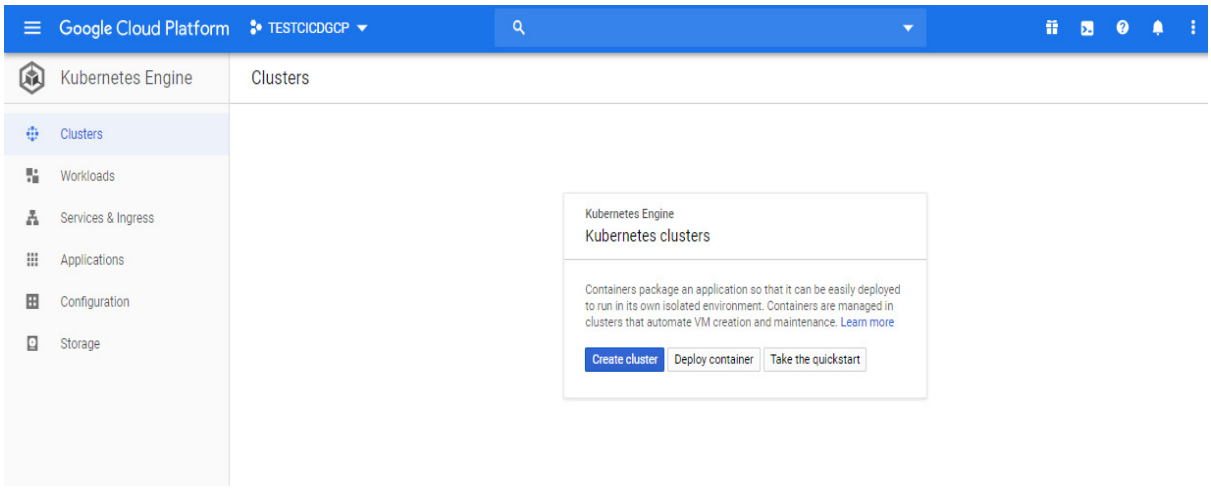
STEP15:

NOW OPEN GOOGLE CLOUD KUBERNETES ENGINE AND AS SOON AS IT IS OPENED THE API STARTS ENABLING



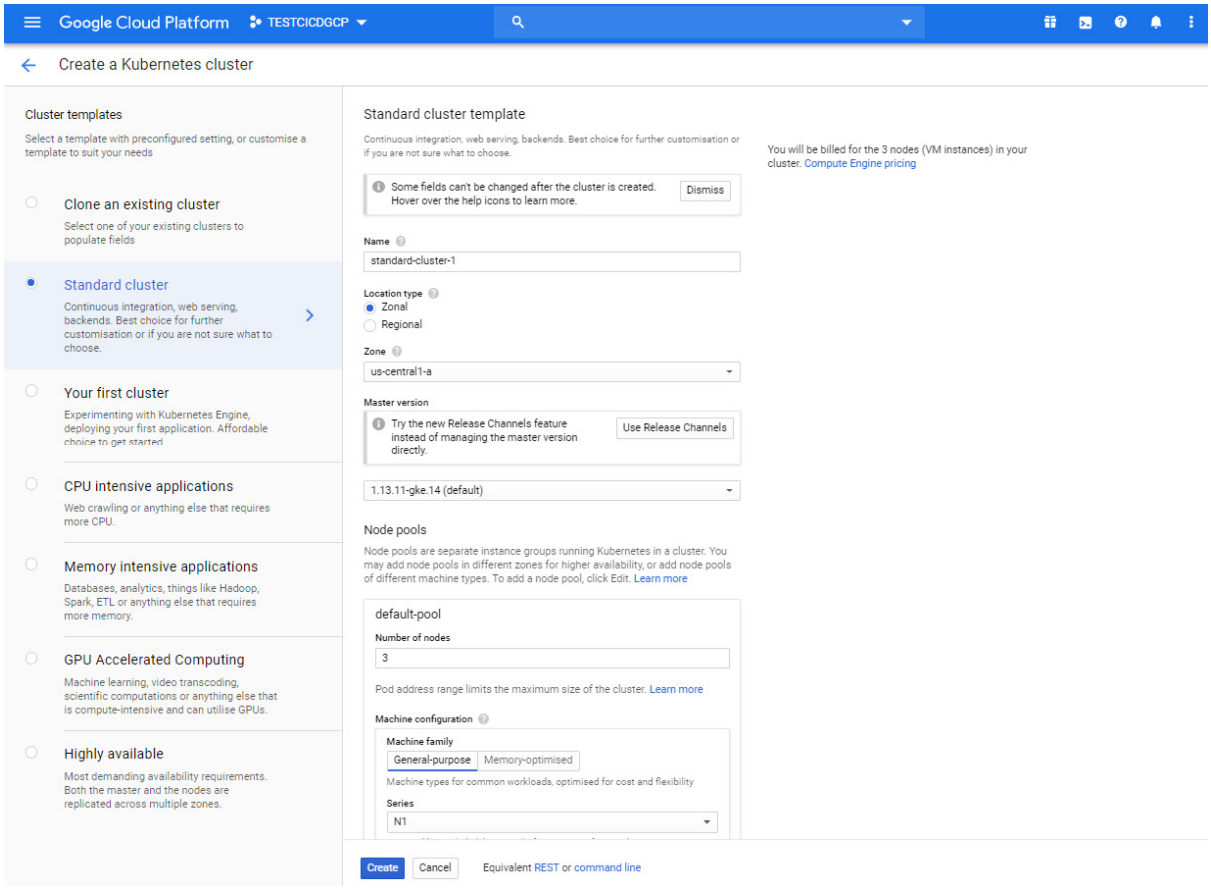
STEP16:

HERE CLICK ON THE CREATE CLUSTER



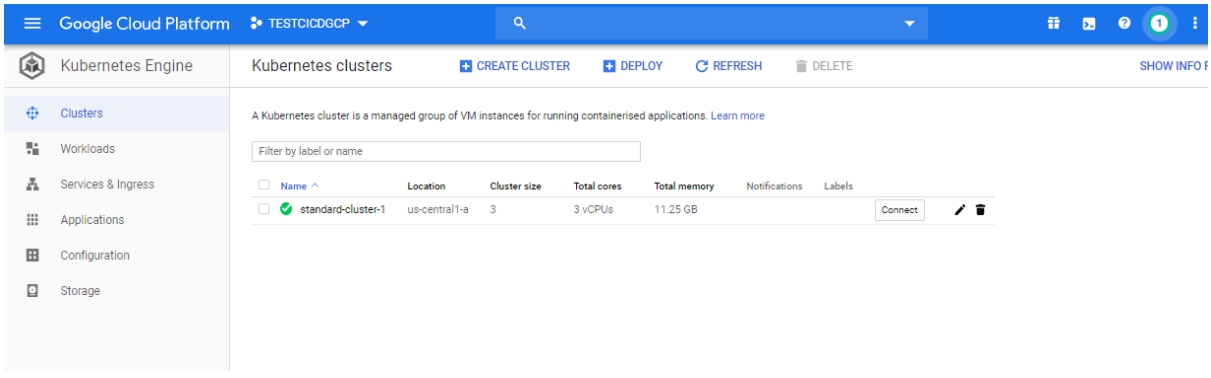
STEP17:

THEN CHECK FOR THE NAMES OF THE CLUSTERS AND LOCATION MATHCES AS IN THE CLOUD BUILD CONFIGURATION FILE AND THEN CLICK ON CREATE



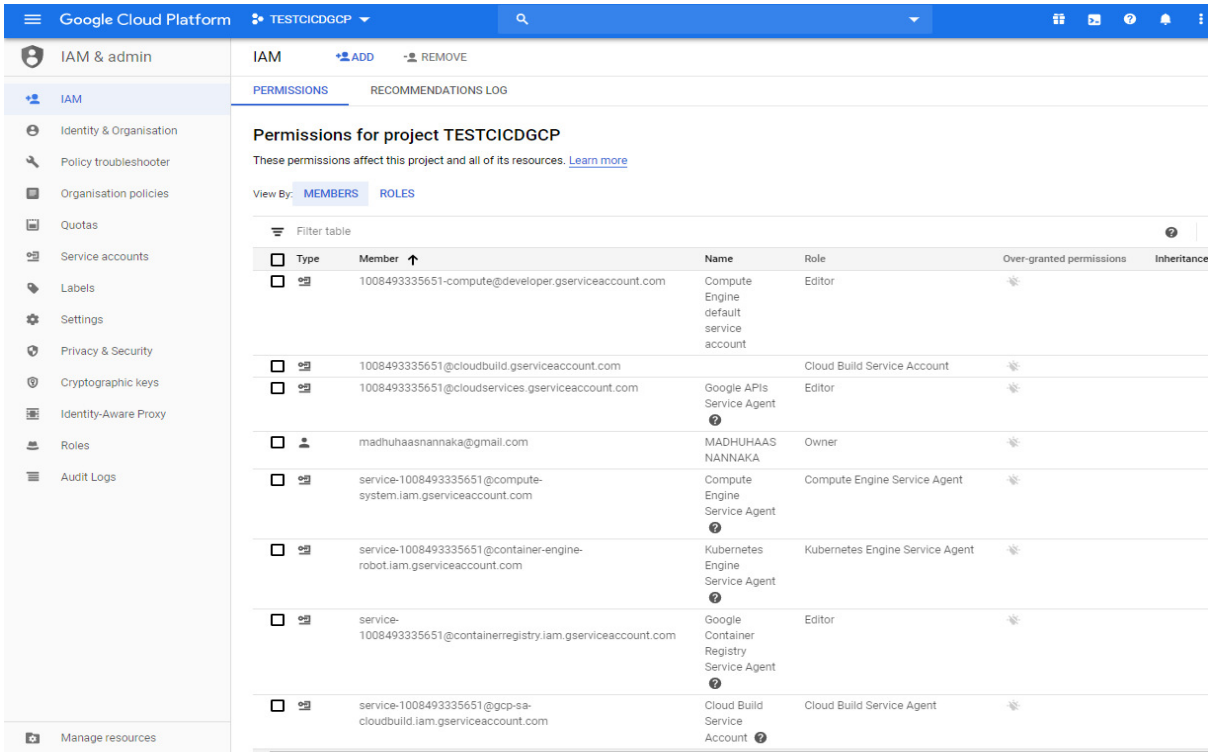
STEP18:

THE KUBERNETES CLUSTER IS CREATED



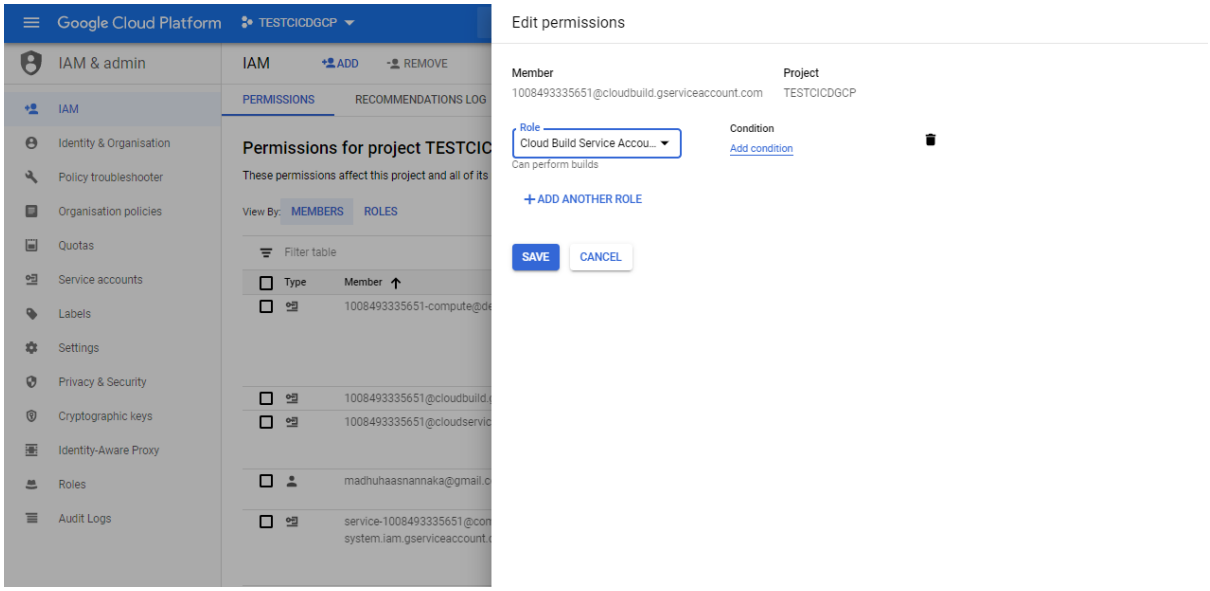
STEP19:

NOW GO TO IAM & ADMIN



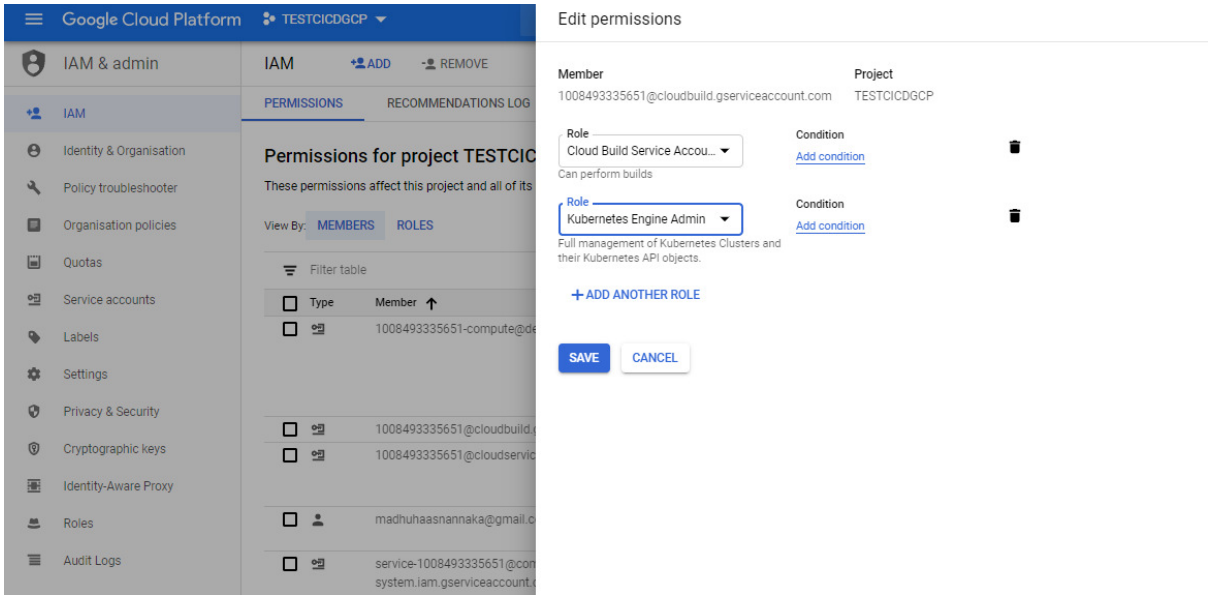
STEP20:

CHECK FOR THE cloudbuild.gserviceaccount.com AND CLICK ON EDIT OPTION AND SELECT ADD ANOTHER ROLE



STEP21:

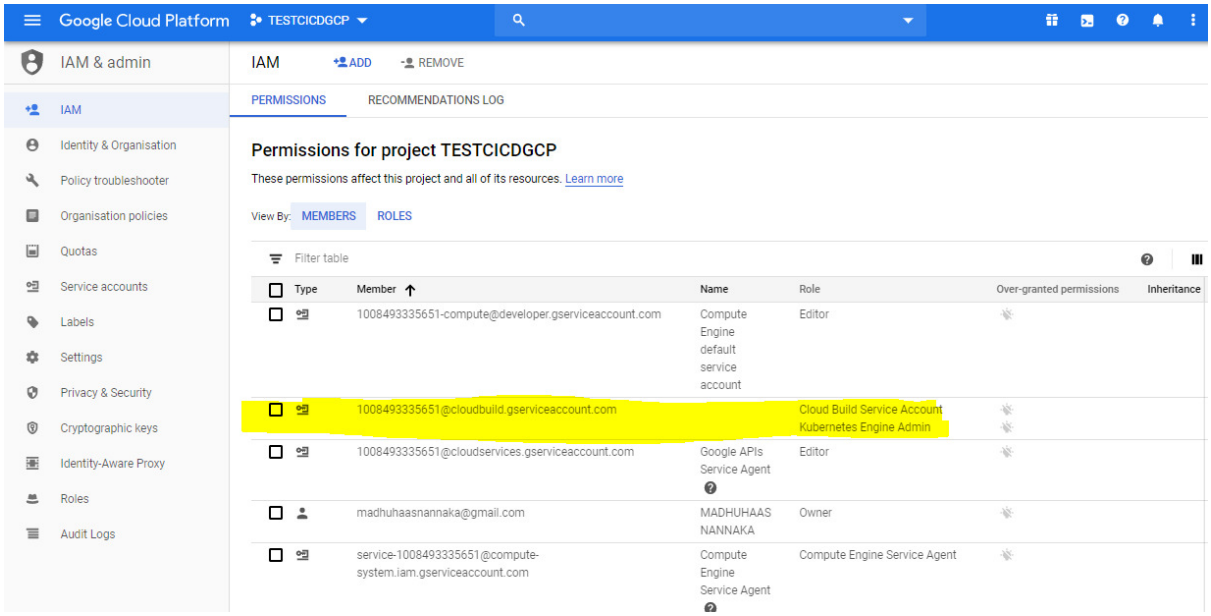
SEARCH FOR KUBERNETES ENGINE ADMIN AND CLICK ON SAVE



STEP22:

NOW THE KUBERNETES ENGINE PERMISSION IS ASSIGNED TO THE CLOUD BUILD SO THAT IT CAN DEPLOY THE APPLICATION IN THE CLUSTER

CHECK THE HIGHLIGHTED PART IN IMAGE

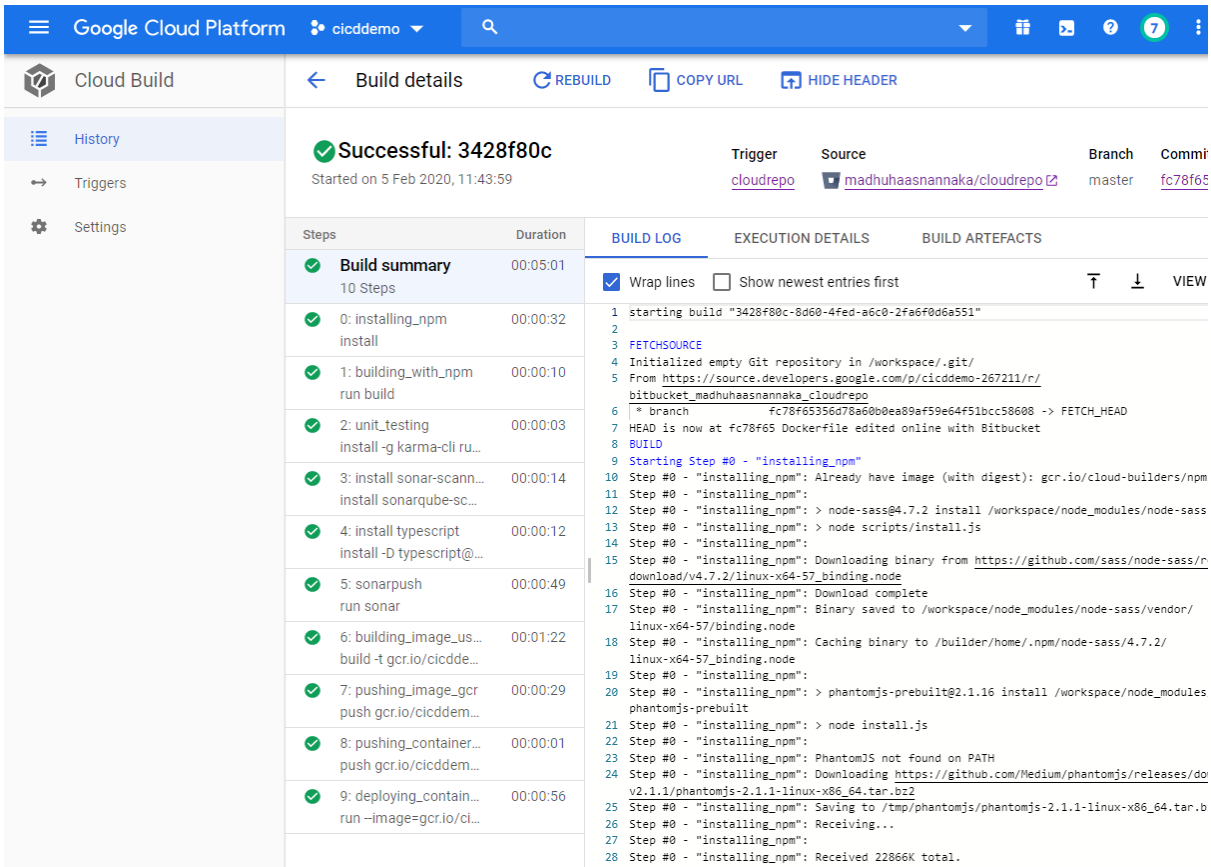


STEP23:

NOW CLICK ON RUN TRIGGER OPTION OR PUSH ANY CHANGES TO THE BITBUCKET SOURCE CODE REPOSITORY TO START THE BUILD

TO START THE BUILD, I PUSHED SOME CODE AND WE CAN SEE THE BUILD STARTED

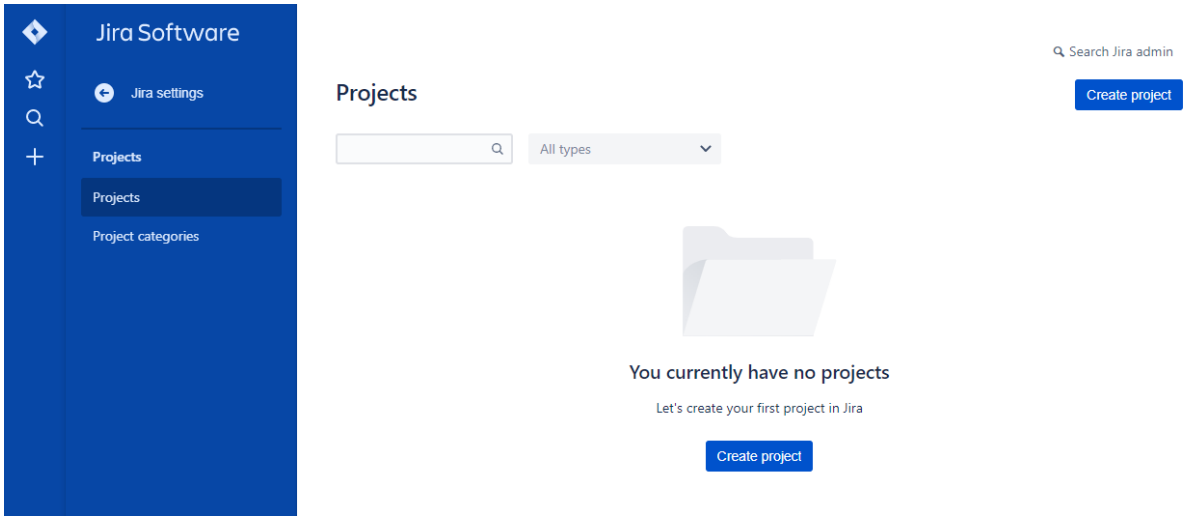
WE CAN SEE THAT IN GITHUB THE COMMIT ID FC78F65 MATCHES WITH CLOUD BUILD GIT COMMIT ID IN HISTORY



NOW AS THE CI/CD IS AUTOMATED LET'S SEE HOW TO CONNECT THE JIRA WITH THIS CI/CD USING BITBUCKET

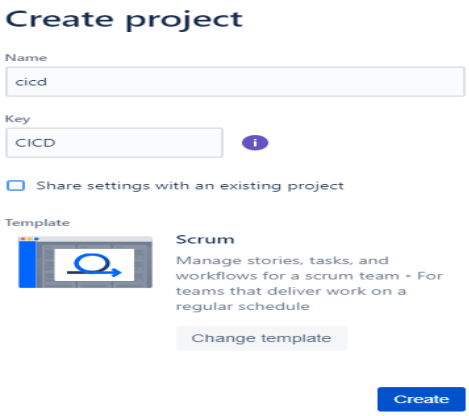
STEP24:

AFTER YOU LOGIN TO THE JIRA SOFTWARE SELECT 'CREATE PROJECT'.



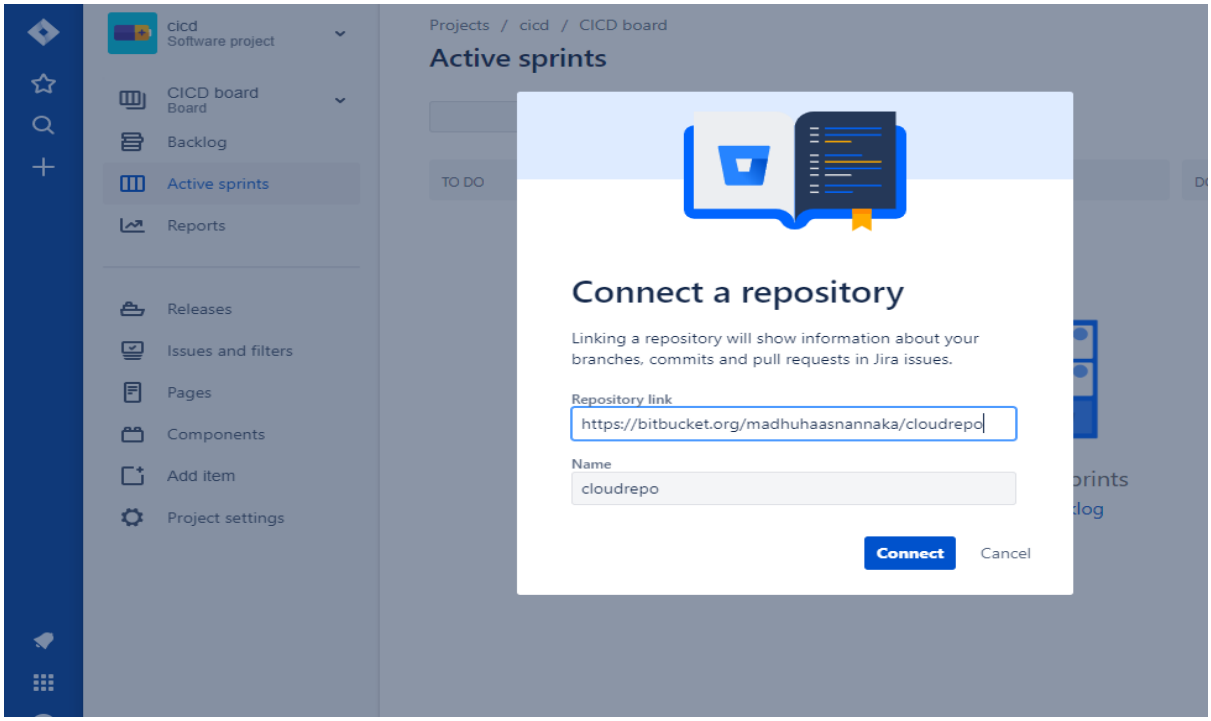
STEP25:

GIVE THE PROJECT NAME AND SELECT THE PROJECT TEMPLATE REQUIRED



STEP26:

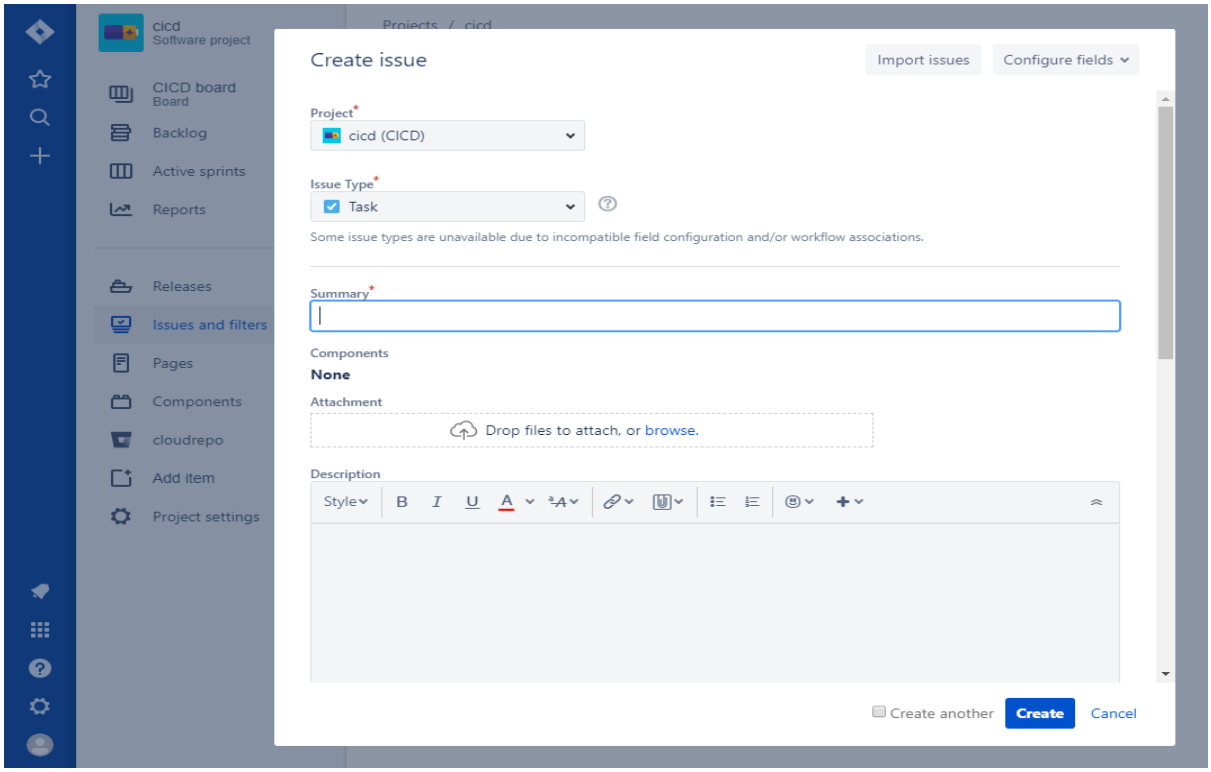
NOW GO TO ADD ITEM AND SELECT REPOSITORY AND CONNECT IT



THEN CLICK ON INTEGRATE WITH BITBUCKET AND GRANT ACCESS TO THE REPOSITORY

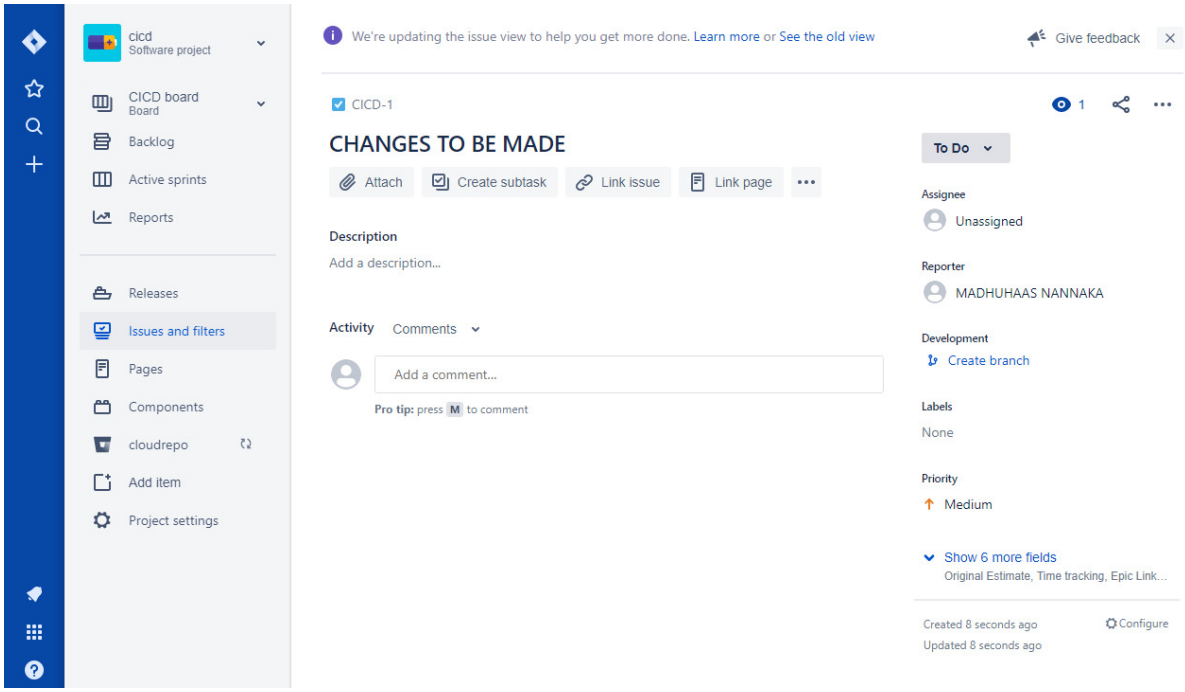
STEP27:

NOW AS THE ACCOUNT IS CONNECTED SELECT THE ISSUE BY CLICKING ON THE PLUS BUTTON AVAILABLE ON LEFT SIDE



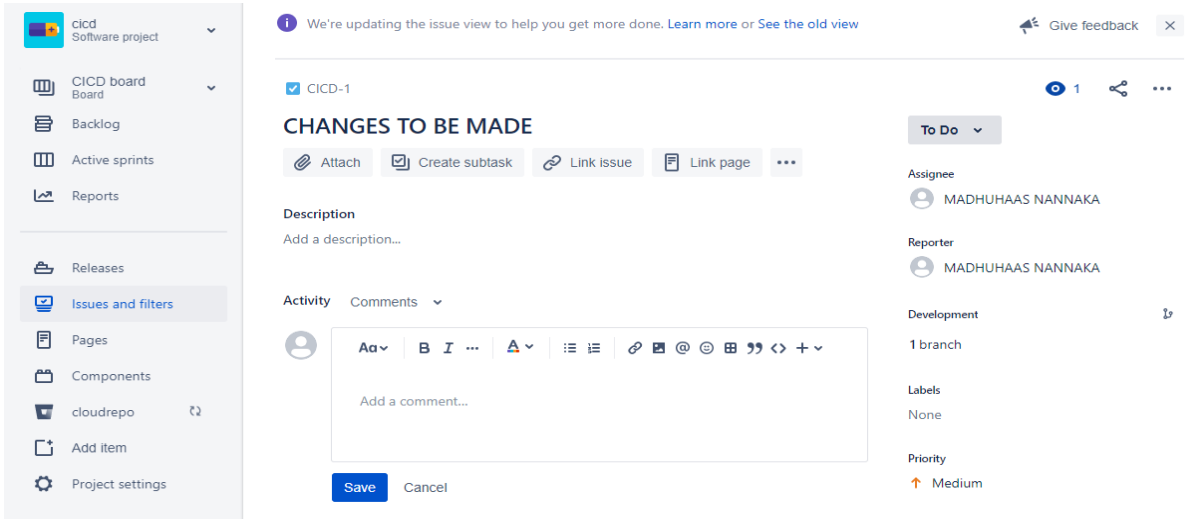
STEP28:

AS THE ISSUE IS GENERATED NOW ASSIGN THE TASK AND CREATE BRACH FOR THE CHANGES



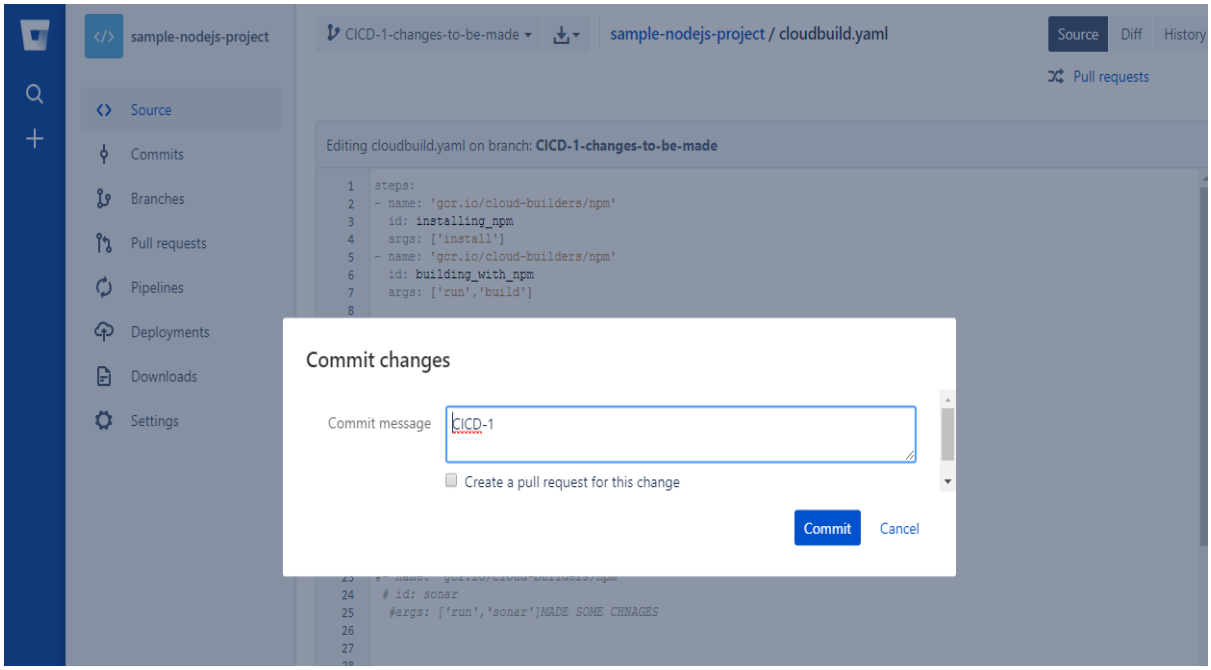
STEP29:

NOW THE ISSUE IS ASSIGNED AND BRANCH IS CREATED SHOWING 1 BRANCH

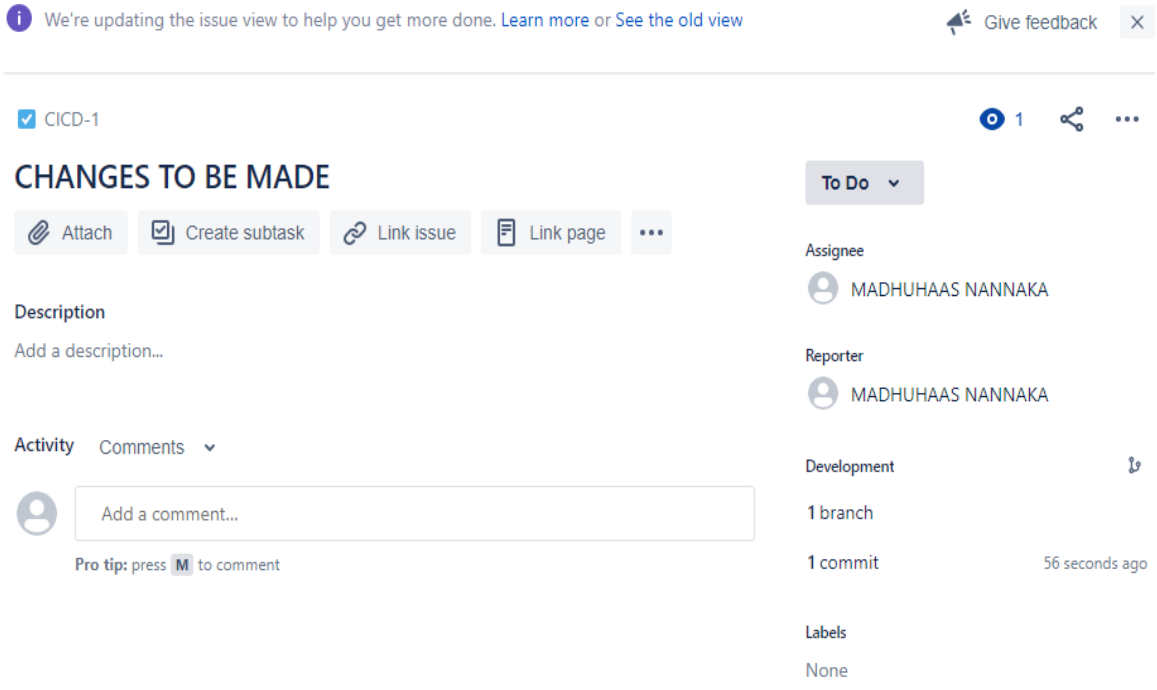


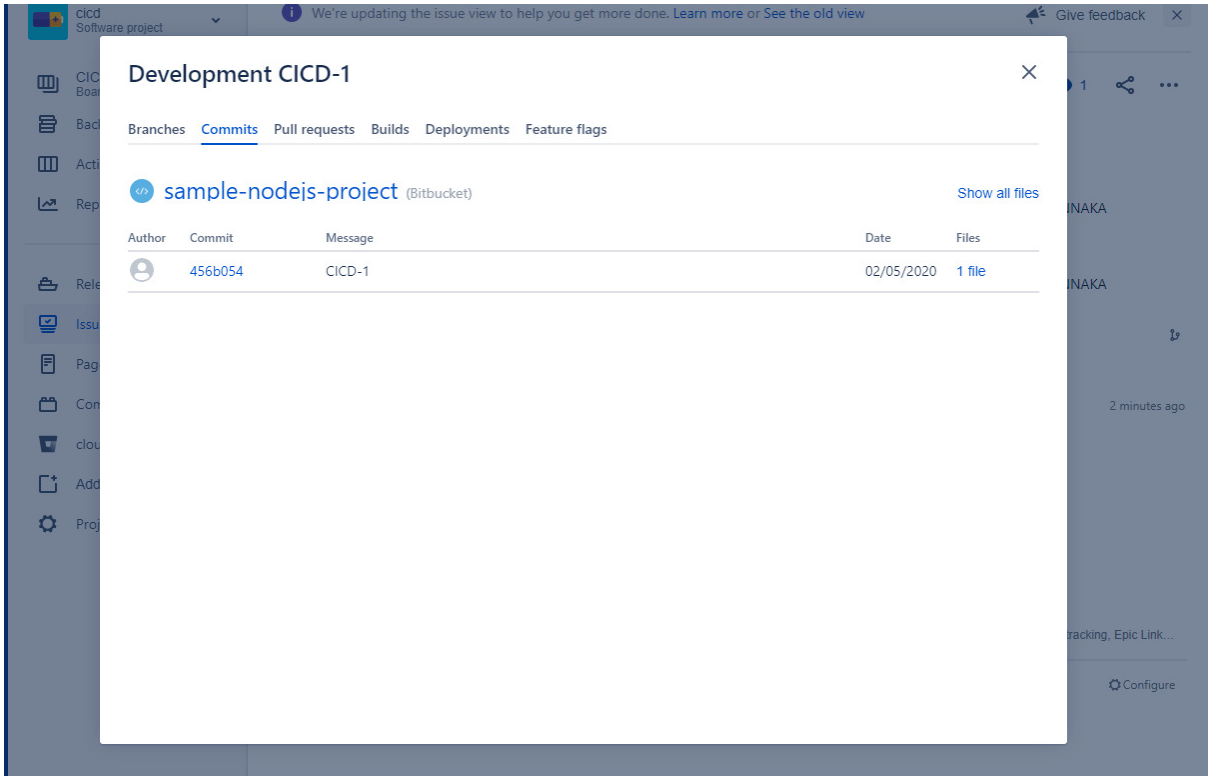
STEP30:

NOW GO TO THE BITBUCKET TO MAKE CHANGES AND COMMIT WITH COMMENTING WITH ISSUE ID



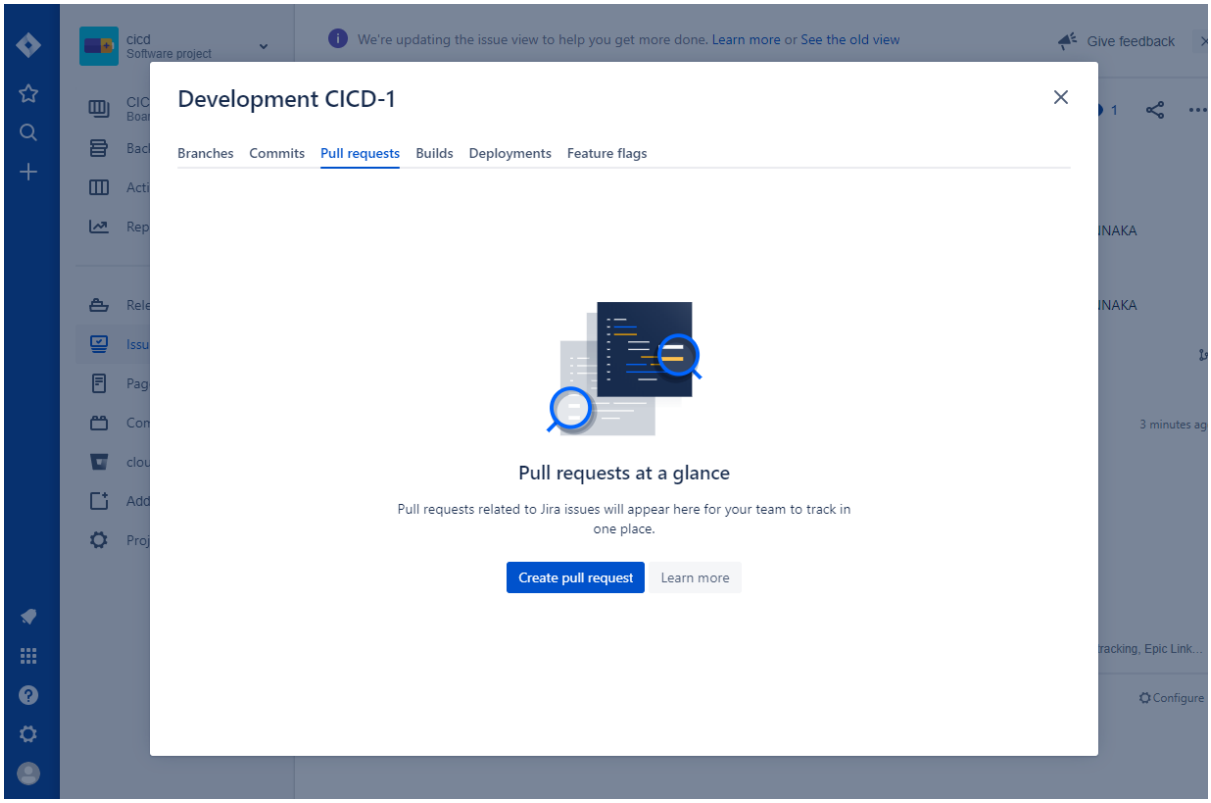
WE CAN SEE THE COMMIT IN THE JIRA TOO



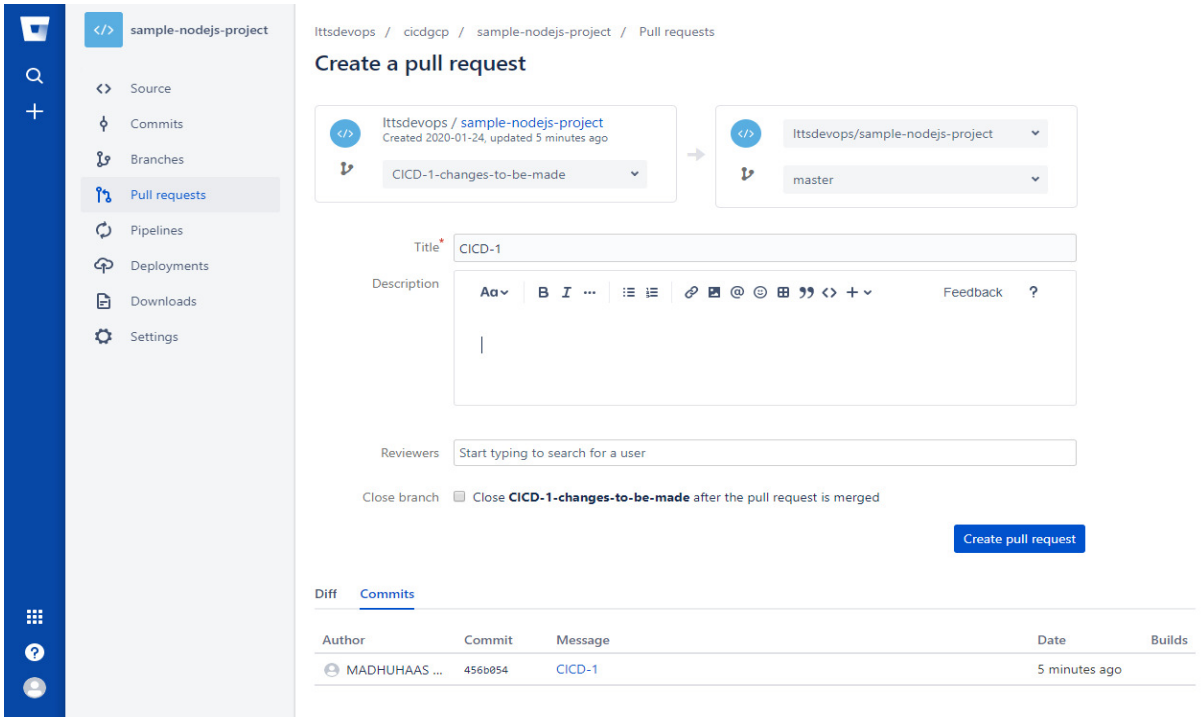


STEP31:

NOW CLICK ON THE PULL REQUEST AND SELECT CREATE PULL REQUEST

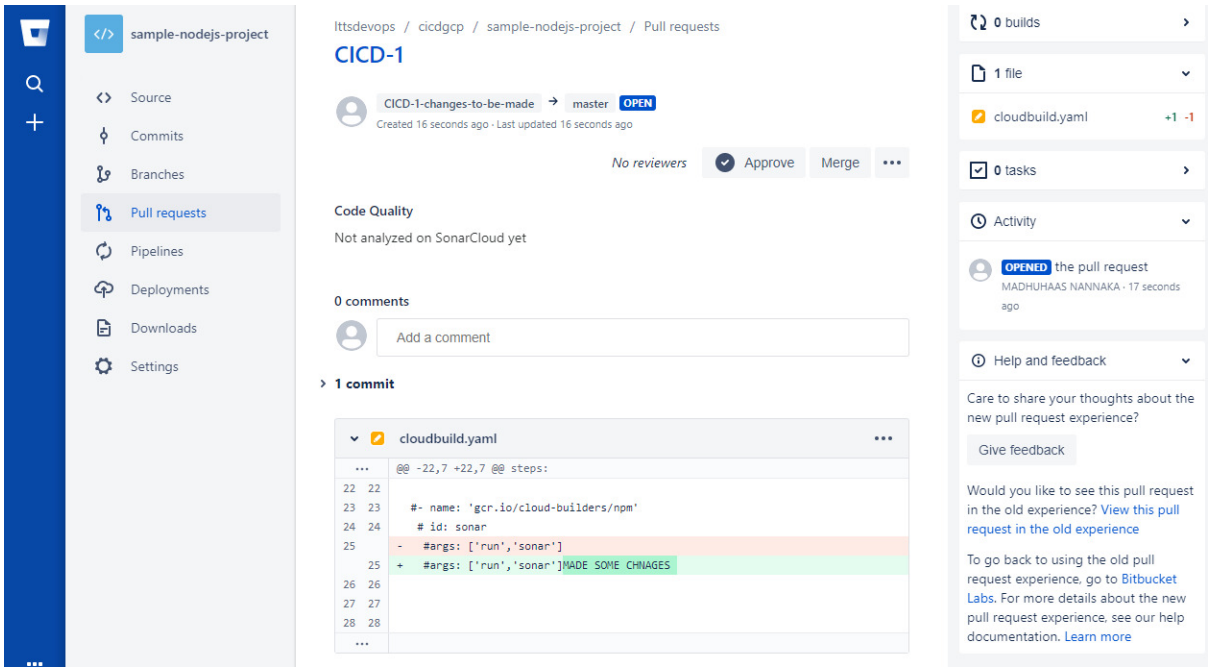


WHERE IT TAKES YOU TO THE BITBUCKET TO MAKE MERGE TO THE MASTER



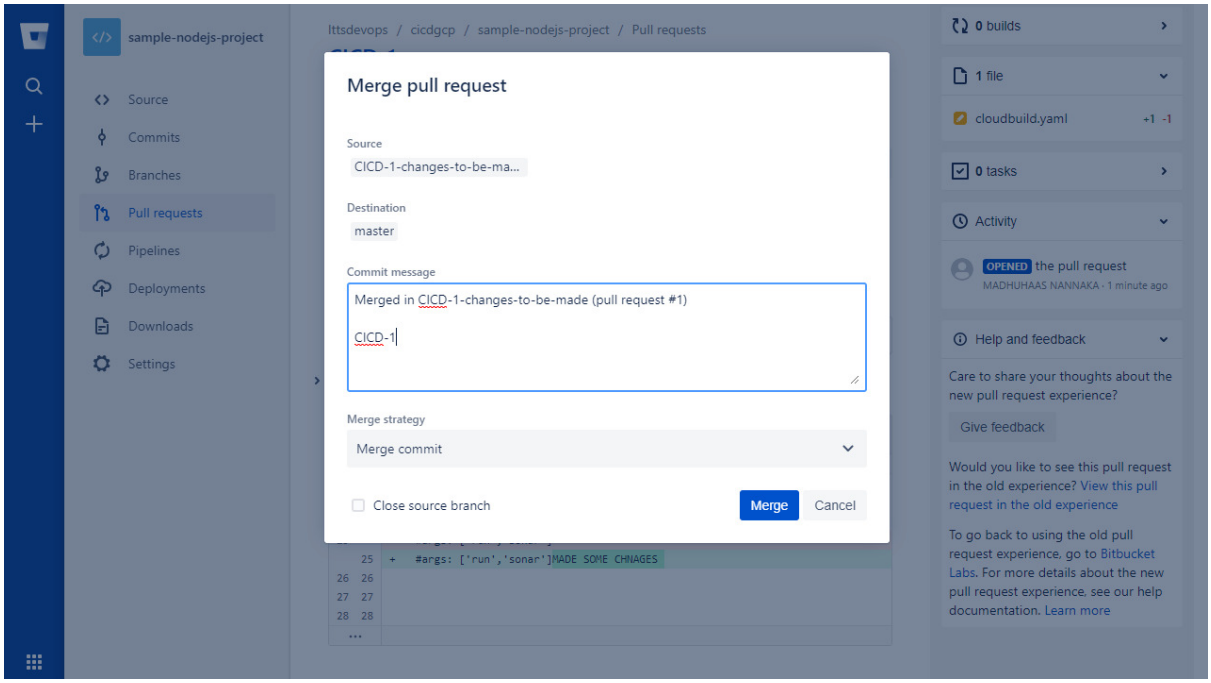
STEP32:

HERE CLICK ON THE CREATE PULL REQUEST AGAIN WHERE IT TAKES YOU TO A PAGE IN BITBUCKET TO APPROVE THE PULL REQUEST

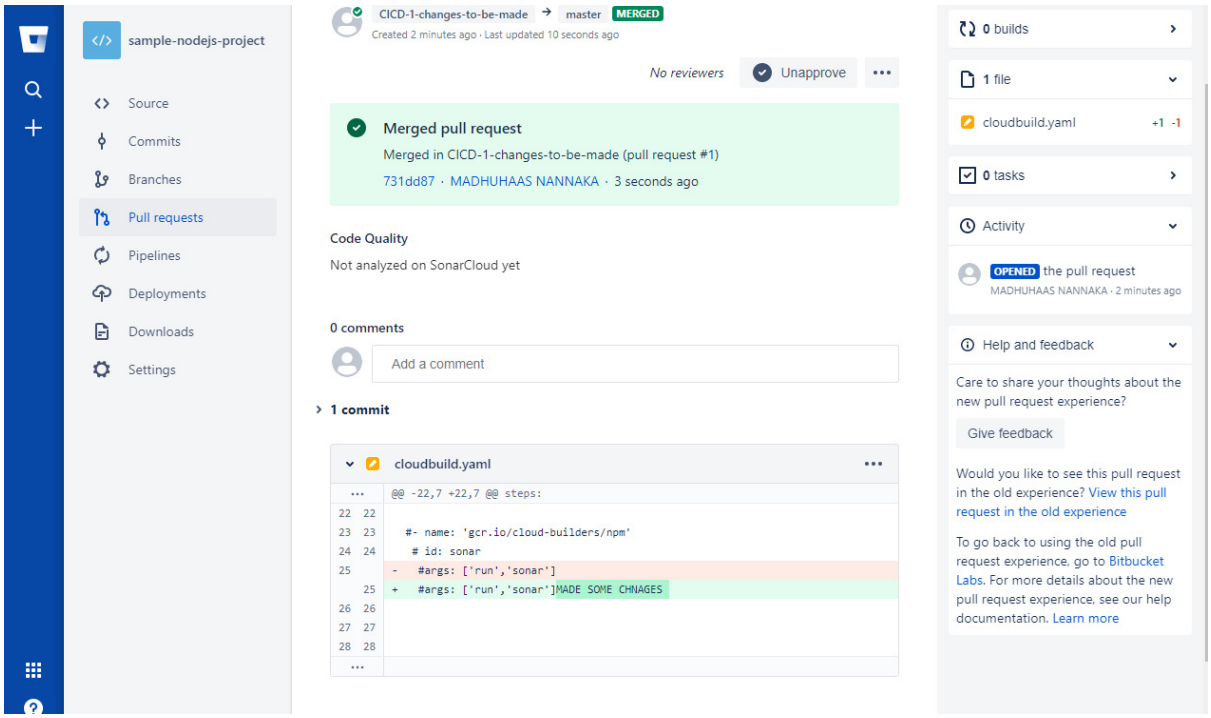


STEP33:

AFTER APPROVING IT CLICK ON MERGE IN A POP-UP WINDOW



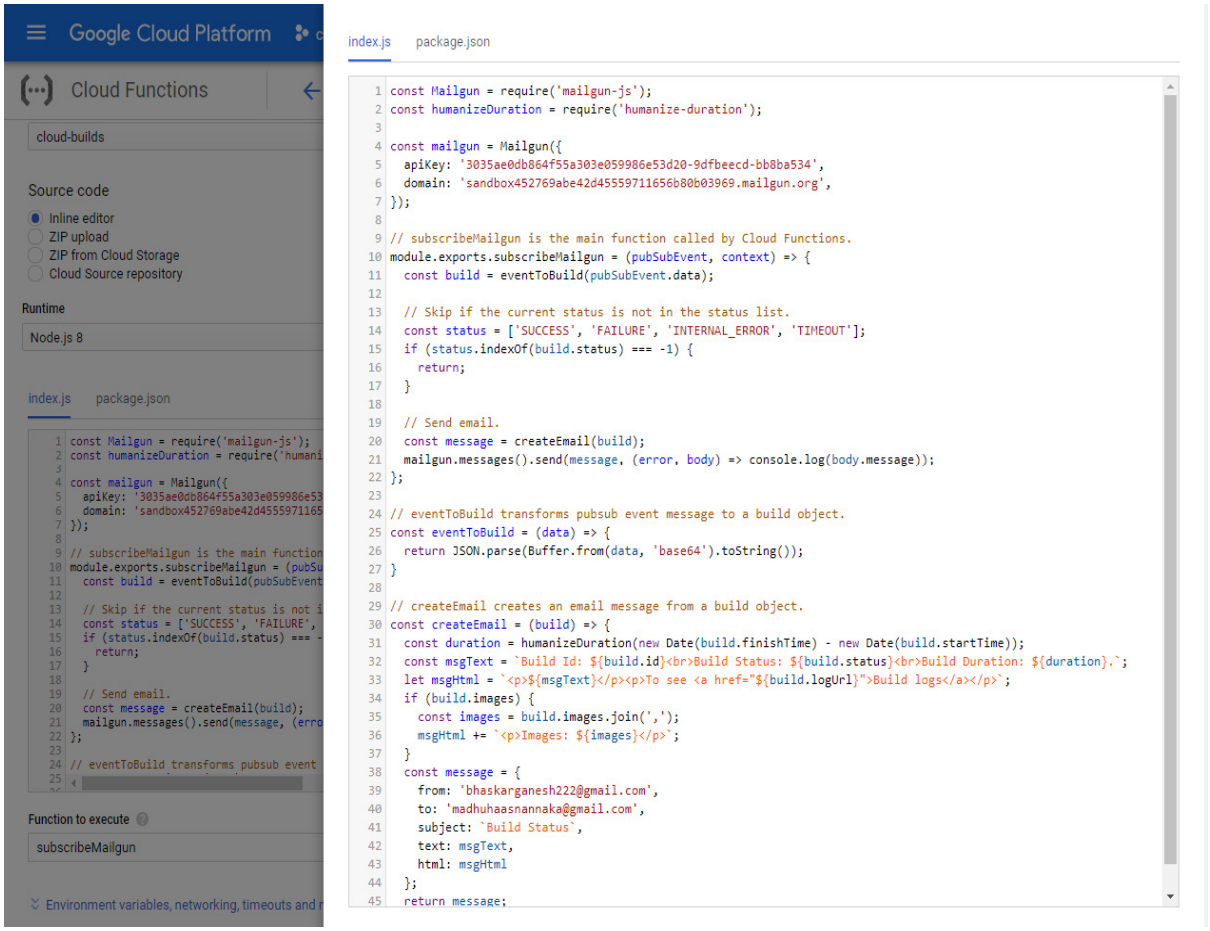
NOW WE CAN CLEARLY SEE THAT THE CODE IS SUCCESSFULLY MERGED WITH MASTER



NOW TO ENABLE SMTP SERVICES FOR THE CLOUD BUILD:

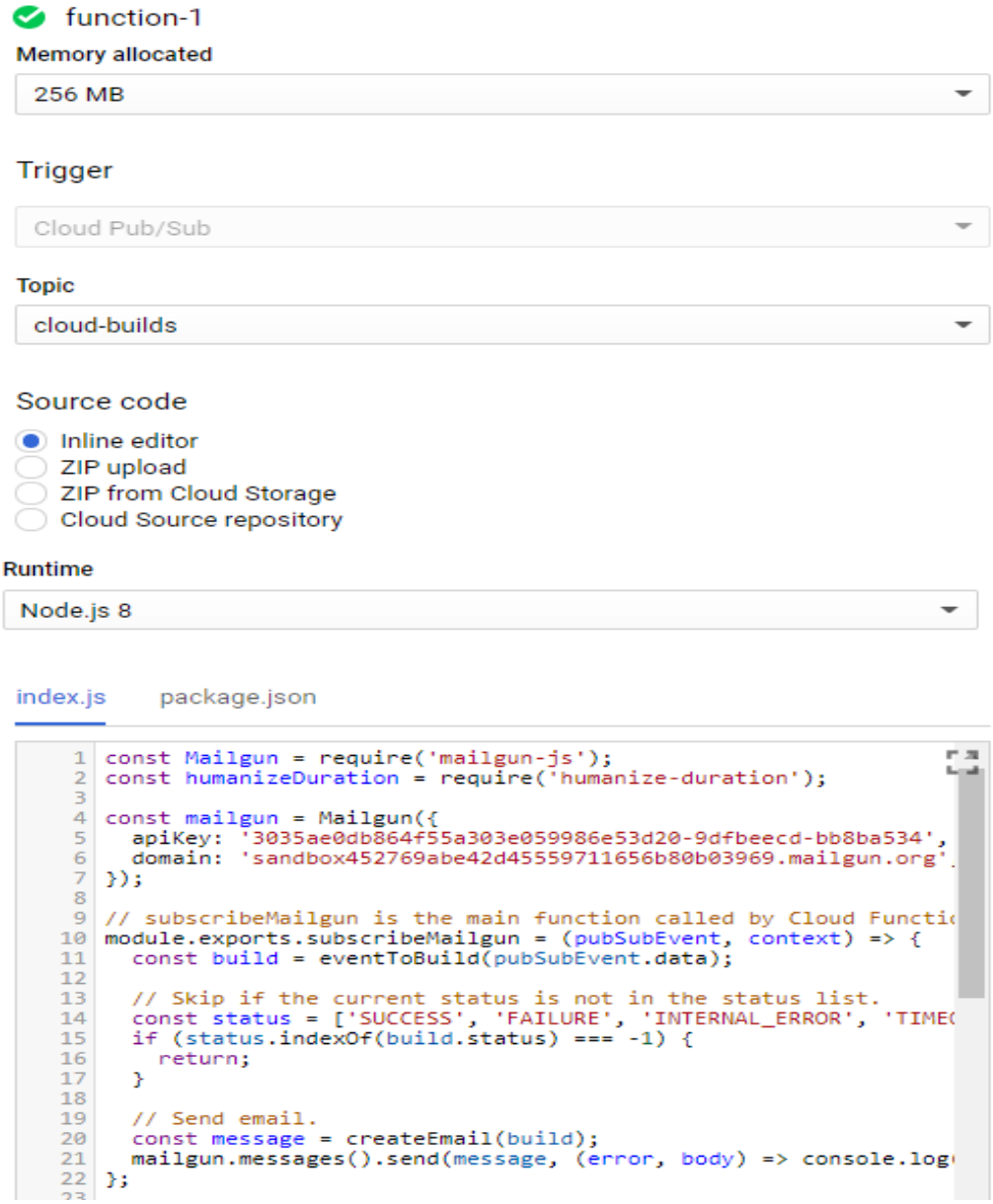
STEP34:

GO TO CLOUD FUNCTIONS API AND CLICK ON CREATE FUNCTION AND ADD THE FOLLOWING CODE



index.js package.json

```
{
  "name": "google-container-email",
  "version": "0.0.1",
  "description": "Email integration for Google Cloud Build, using Google Cloud Functions",
  "main": "index.js",
  "dependencies": {
    "humanize-duration": "3.10.0",
    "mailgun-js": "~0.11.2"
  },
  "devDependencies": {
    "async": "^2.1.5",
    "mocha": "3.2.0",
    "should": "11.2.1"
  }
}
```



SELECT TRIGGER AS CLOUD SUB/PUB AND TOPIC AS CLOUD-BUILDS AND SOURCE CODE AS INLINE EDITOR AND CLICK FUNCTION TO EXECUTIVE AS SUBSCRIBE MAILGUN AND CLICK ON DEPLOY. BUT BEFORE THAT CREATE

CODE FOR CLOUD FUNCTION index.js IS AS FOLLOWS:

```
const Mailgun = require('mailgun-js');
const humanizeDuration = require('humanize-duration');
const mailgun = Mailgun({
 apiKey: '3035ae0db864f55a303e059986e53d20-9dfbeecd-bb8ba534',
 domain: 'sandbox452769abe42d45559711656b80b03969.mailgun.org',
});
// subscribeMailgun is the main function called by Cloud Functions.
module.exports.subscribeMailgun = (pubSubEvent, context) => {
 const build = eventToBuild(pubSubEvent.data);
 // Skip if the current status is not in the status list.
 const status = ['SUCCESS', 'FAILURE', 'INTERNAL_ERROR', 'TIMEOUT'];
 if (status.indexOf(build.status) === -1) {
  return;
 }
 // Send email.
 const message = createEmail(build);
 mailgun.messages().send(message, (error, body) => console.log(body.message));
};
// eventToBuild transforms pubsub event message to a build object.
const eventToBuild = (data) => {
 return JSON.parse(Buffer.from(data, 'base64').toString());
}
// createEmail creates an email message from a build object.
const createEmail = (build) => {
 const duration = humanizeDuration(new Date(build.finishTime) - new Date(build.startTime));
 const msgText = `Build Id: ${build.id}<br>Build Status: ${build.status}<br>Build Duration: ${duration}.`;
 let msgHtml = `<p>${msgText}</p><p>To see <a href="${build.logUrl}">Build logs</a></p>`;
 if (build.images) {
  const images = build.images.join(',');
  msgHtml += `<p>Images: ${images}</p>`;
 }
 const message = {
  from: 'bhaskarganesh222@gmail.com',
  to: 'bhaskarganesh7777@gmail.com',
```

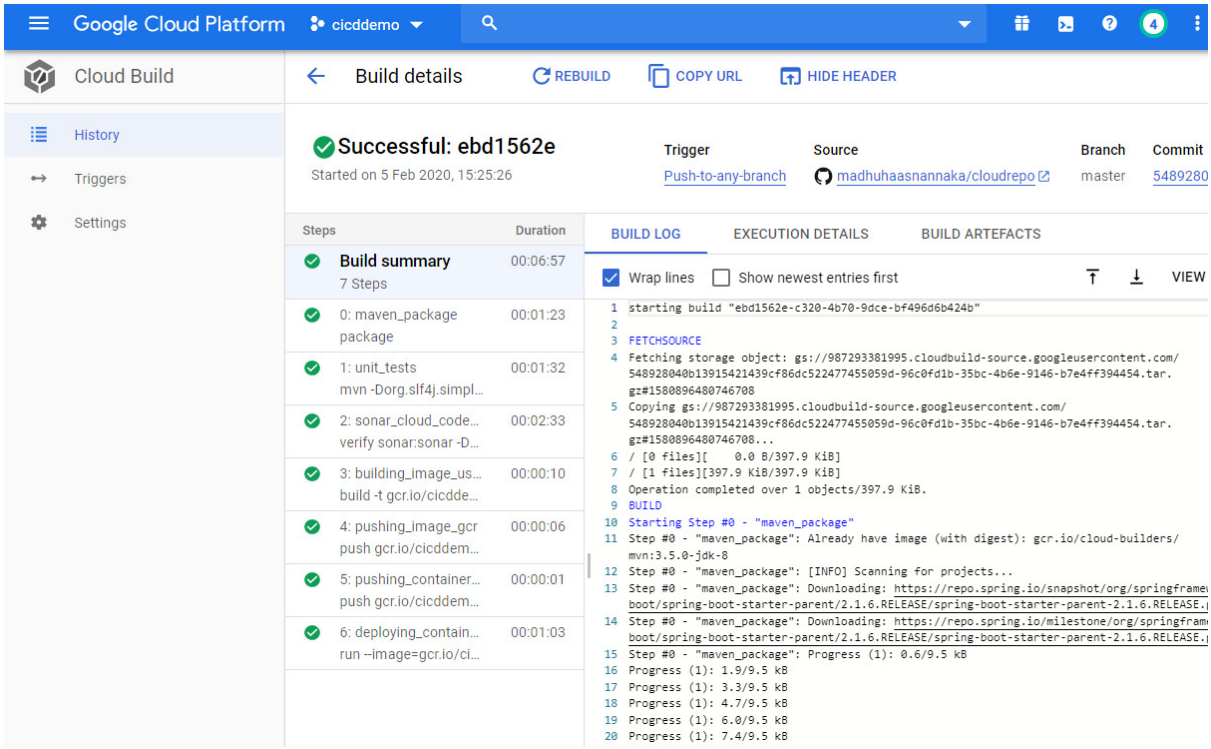
```
    subject: `Build Status`,
    text: msgText,
    html: msgHtml
  };
  return message;
}
```

CODE FOR CLOUD FUNCTION package.json IS AS FOLLOWS:

```
{
  "name": "google-container-email",
  "version": "0.0.1",
  "description": "Email integration for Google Cloud Build, using Google Cloud Functions",
  "main": "index.js",
  "dependencies": {
    "humanize-duration": "3.10.0",
    "mailgun-js": "~0.11.2"
  },
  "devDependencies": {
    "async": "^2.1.5",
    "mocha": "3.2.0",
    "should": "11.2.1"
  }
}
```

STEP35:

AN ACCOUNT IN MAIL GUN AND CREATE A DOMAIN KEY AND API KEY AS IN THE CODE. THEN WE CAN SEE FOR EVERY BUILD A MAIL IS GENERATED.



AS THE BUILD IS SUCCESSFUL, WE CAN SEE THE MAIL THAT NUILD IS SUCCESS WITH THE SAME BUILD ID AS EBD1562E...

Now the fully automated ci/cd for NodeJS project using bitbucket and GCP is ready with added features include SMTP services and JIRA integration.

**********THANK YOU**********