

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

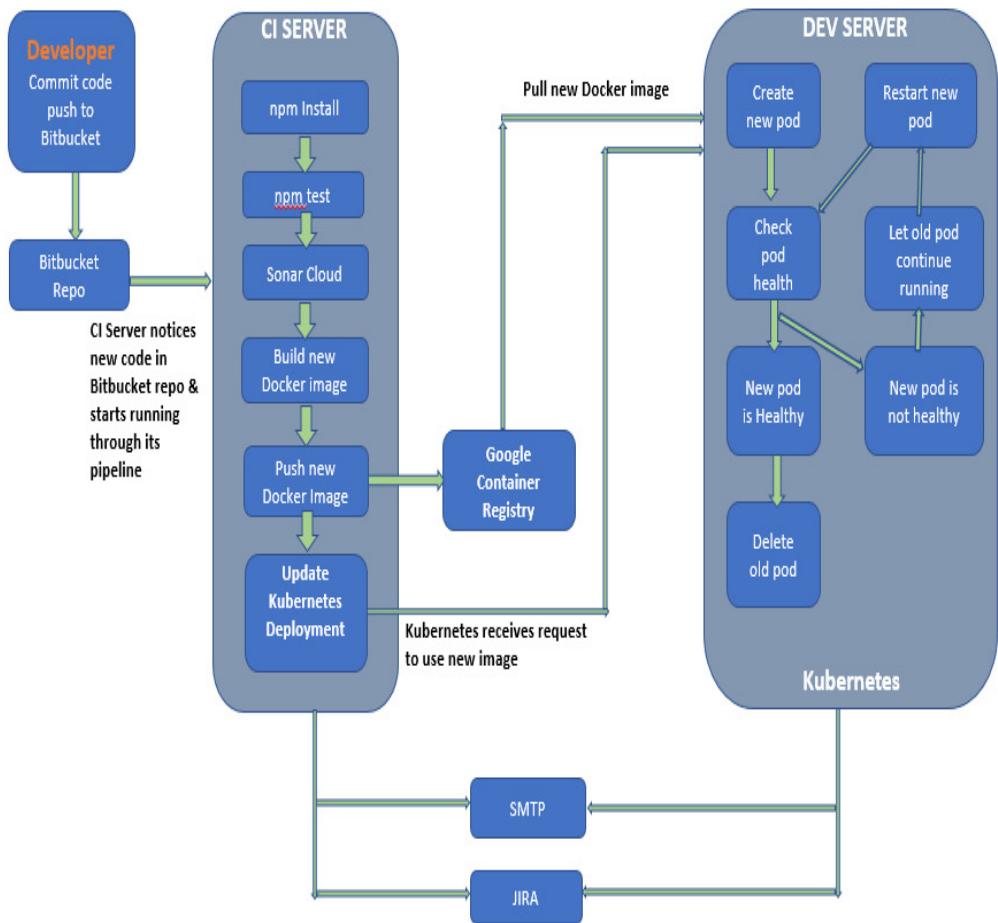
- GOOGLE CLOUD BUILD
- GOOGLE KUBERNETES ENGINE
- BITBUCKET
- NPM
- DOCKER
- GOOGLE CONTAINERS REPOSITORY
- GOOGLE CLOUD PERMISSION MANAGER (IAM & ADMIN)
- SONARCLOUD
- SONARQUBE-SCANNER
- GOOGLE CLOUD FUNCTIONS
- JIRA
- MAILGUN
- PUB/SUB

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:

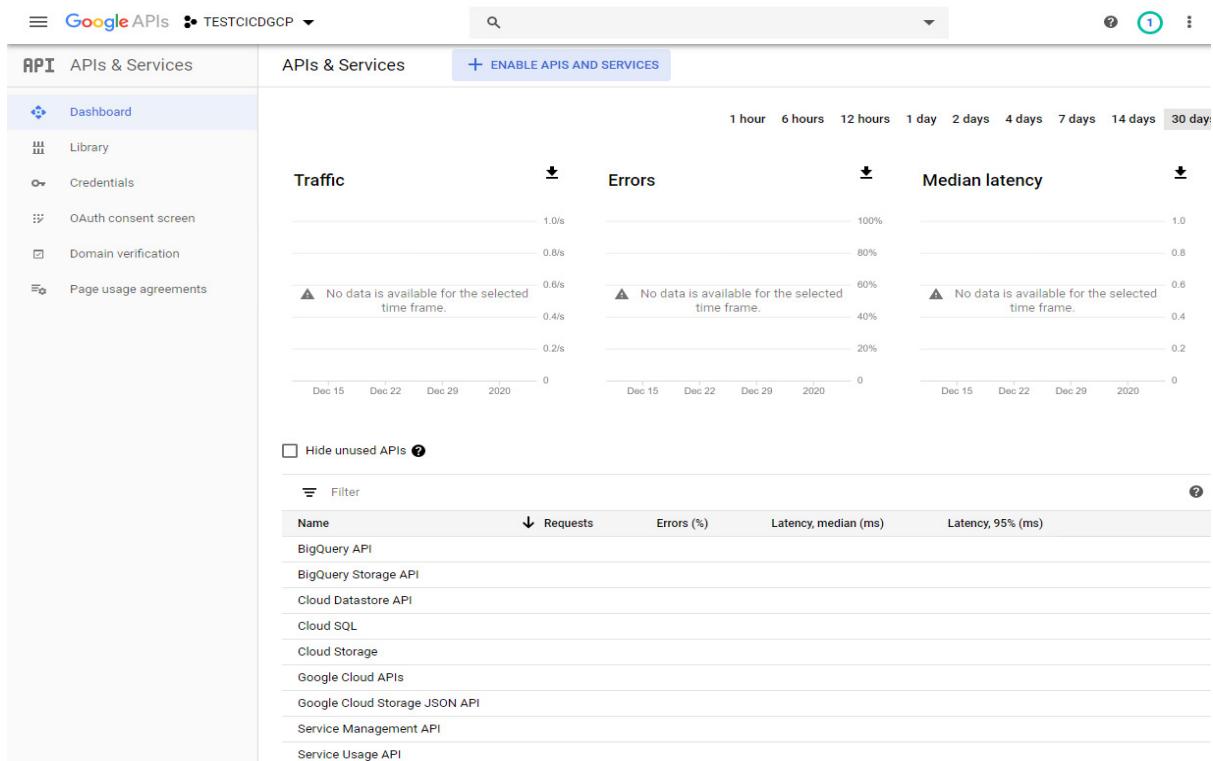
STEP1:

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

The screenshot shows the 'New Project' page in the Google Cloud Platform. At the top, there's a blue header bar with the 'Google Cloud Platform' logo, a search bar, and several icons. Below the header, the text 'New Project' is displayed. A warning message in a grey box says: '⚠ You have 19 projects remaining in your quota. Request an increase or delete projects. [Learn more](#)' with a 'MANAGE QUOTAS' link. The main form fields include: 'Project name *' with 'TESTCICDGCP' entered; 'Location *' with 'No organisation' selected and a 'BROWSE' button; and a 'Parent organisation or folder' dropdown which is empty. At the bottom, there are two buttons: a blue 'CREATE' button and a white 'CANCEL' button.

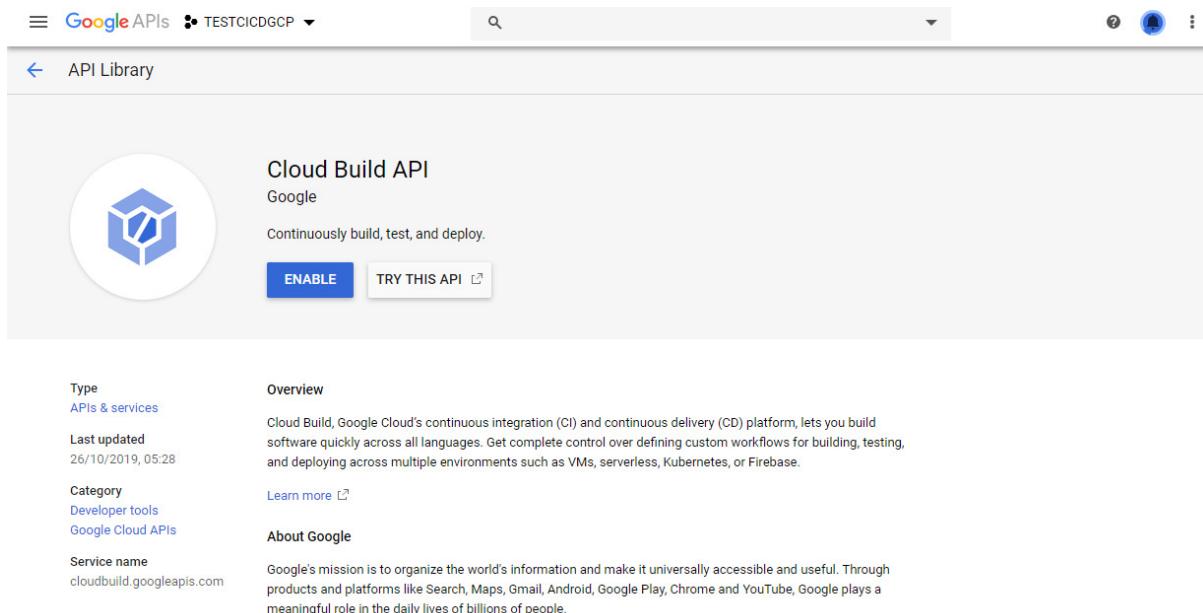
STEP2:

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.

The screenshot shows the Google Cloud Platform interface for the Cloud Build API. The top navigation bar includes the project name 'TESTCICDGC' and various icons. On the left, a sidebar lists 'Overview', 'Metrics', 'Quotas', and 'Credentials'. The main content area has a message: 'To use this API, you may need credentials. Click 'Create credentials' to get started.' Below this are two sections: 'Details' (Name: Cloud Build API, By: Google, Service name: cloudbuild.googleapis.com, Overview: Creates and manages builds on Google Cloud Platform, Activation status: Enabled) and 'Traffic by response code' (Request/sec (2 hr average) chart showing values from 0.2/s to 1.0/s). A 'CREATE CREDENTIALS' button is located in the top right corner of the message area.

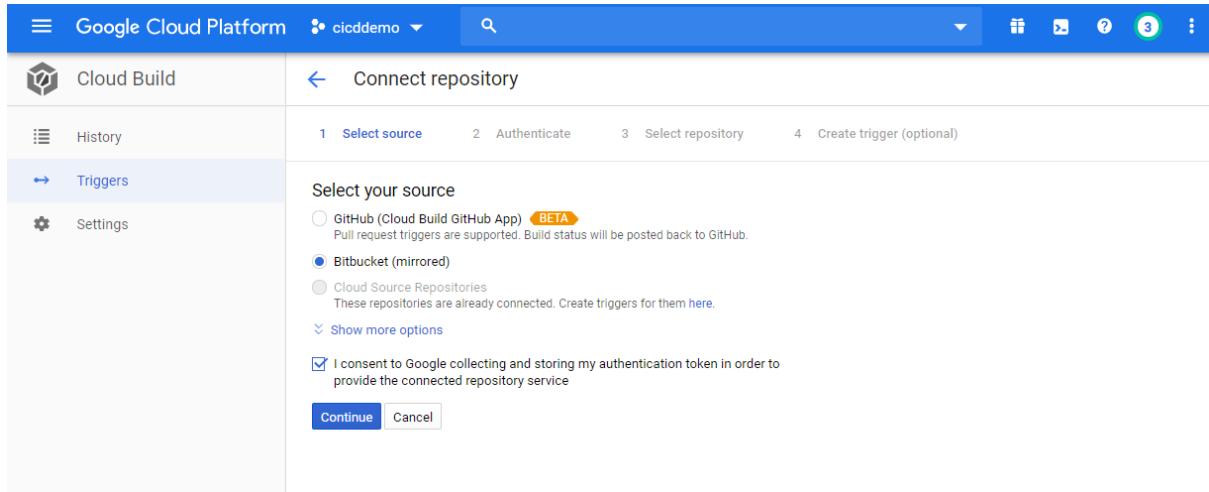
STEP5:

GO TO TRIGGERS IN CLOUD BUILD AND SELECT CONNECT REPOSITORY

The screenshot shows the Google Cloud Platform Cloud Build interface. The top navigation bar includes the project name 'TESTCICDGC' and various icons. On the left, a sidebar lists 'History', 'Triggers' (which is selected), and 'Settings'. The main content area displays a 'Cloud Build Triggers' section with the text: 'Connect your working project repositories to start building with Google Cloud Build. Once connected, you can create and configure specific triggers for your project repositories. Learn more'. A prominent blue 'Connect repository' button is centered below the text.

STEP6:

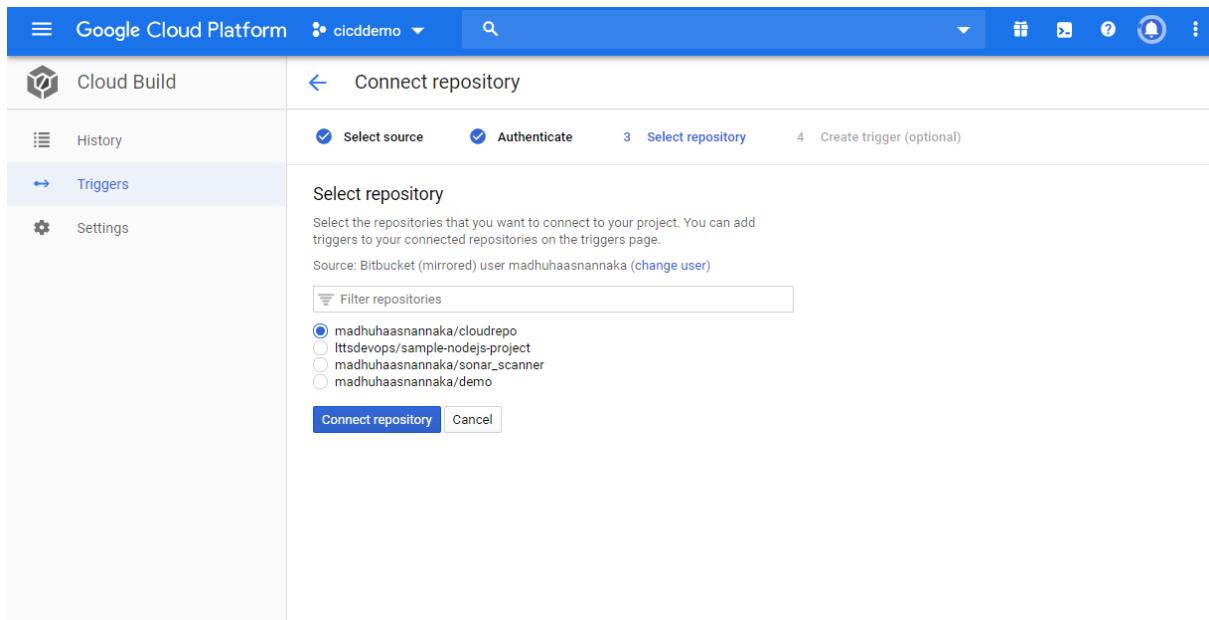
AND IN HERE SELECT AS A SOURCE AS BITBUCKET



The screenshot shows the Google Cloud Platform Cloud Build interface. On the left, there's a sidebar with 'Cloud Build' selected. The main area is titled 'Connect repository' with a sub-section 'Select your source'. It lists three options: 'GitHub (Cloud Build GitHub App) [BETA]', 'Bitbucket (mirrored)', and 'Cloud Source Repositories'. 'Bitbucket (mirrored)' is selected. Below it, there's a checkbox for 'I consent to Google collecting and storing my authentication token in order to provide the connected repository service', which is checked. At the bottom are 'Continue' and 'Cancel' buttons.

STEP7:

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



This screenshot shows the same 'Connect repository' screen after the Bitbucket source has been selected. The 'Select source' and 'Authenticate' steps are now marked as completed with blue checkmarks. The 'Select repository' step is highlighted in blue. The main area shows a list of repositories under 'Select repository': 'madhuhaasnannaka/cloudrepo' (selected), 'ltsdevops/sample-nodejs-project', 'madhuhaasnannaka/sonar_scanner', and 'madhuhaasnannaka/demo'. There's also a 'Filter repositories' input field. At the bottom are 'Connect repository' and 'Cancel' buttons.

STEP8:

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

The screenshot shows the Google Cloud Platform Cloud Build interface. On the left, there's a sidebar with 'Cloud Build' selected. The main area is titled 'Connect repository' with a back arrow. It has four checkboxes at the top: 'Select source' (checked), 'Authenticate' (checked), 'Select repository' (checked), and 'Create a push trigger (optional)' (unchecked). Below these are sections for 'Create a push trigger (optional)' and 'Create a push trigger for these repositories'. In the 'Create a push trigger (optional)' section, there's a table with columns 'Description', 'Event', 'Filter', and 'Build configuration'. One row is shown: 'Push to any branch' under 'Event', 'Push to branch' under 'Filter', and 'Auto-detected' under 'Build configuration'. In the 'Create a push trigger for these repositories' section, there's a checkbox 'Select all repositories' which is checked, and a list containing 'cloudrepo'. At the bottom are two buttons: 'Create push trigger' (blue) and 'Skip for now'.

STEP9:

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

The screenshot shows the Google Cloud Platform Cloud Build interface on the 'Triggers' page. The sidebar shows 'Cloud Build' selected. The main area has tabs 'Triggers BETA', 'CONNECT REPOSITORY', and '+ CREATE TRIGGER'. Below is a section for 'Repositories' with 'ACTIVE' and 'INACTIVE' tabs. Under 'ACTIVE', there are two entries: 'madhuhaasnannaka/cloudrepo' and 'madhuhaasnannaka/demo'. Both entries show a table with columns 'Description', 'Event', 'Filter', 'Build configuration', and 'Status'. For 'cloudrepo', the status is 'Enabled' and the build configuration is 'cloudbuild.yaml'. For 'demo', the status is 'Disabled' and the build configuration is 'cloudbuild.yaml'. Each entry also has a 'Run trigger' button.

STEP10:

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

Google Cloud Platform TESTCICDCGP

Cloud Build

Triggers

Create trigger

Repository: Select a repository

Name: Must be unique within the project
trigger-1

Description: Trigger 1

Trigger type: Branch (selected)

Branch (regex): Invert Regex

Included files filter (glob) (Optional): Changes affecting at least one included file will trigger builds
glob pattern example: src/**

Ignored files filter (glob) (Optional): Changes only affecting ignored files won't trigger builds
glob pattern example: .gitignore

Build configuration: Dockerfile (selected)

Dockerfile directory (Optional): The directory will also be used as the Docker build context
/

Dockerfile name (Optional): The filename is relative to the Dockerfile directory
Dockerfile

Image name (Optional): Supported variables: \$PROJECT_ID, \$REPO_NAME, \$BRANCH_NAME, \$TAG_NAME, \$COMMIT_SHA, \$SHORT_SHA

Timeout (Optional): The default timeout is 10 minutes
600 seconds

<| Create trigger Cancel

STEP11:

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

Google Cloud Platform TESTCICDCGP

Cloud Build

Triggers

Create trigger

Repository: madhuhaasnannaka/clourepo

Name: Must be unique within the project
TRIGGER1

Description: Push to any branch

Trigger type: Branch (selected)

Branch (regex): Invert Regex

Build configuration: Auto-detected (A cloudbuild.yaml or Dockerfile will be detected in the repository)

Cloud Build configuration file location: /cloudbuild.yaml

Substitution variables (Optional): Substitutions allow the re-use of a cloudbuild.yaml file with different variable values
Learn more

+ Add item

Create trigger Cancel

STEP12:

WE CAN SEE THE SAME AS BEFORE BUT TRIGGER IS GENERATED

The screenshot shows the Google Cloud Platform Cloud Build Triggers page. The left sidebar has options for History, Triggers (which is selected), and Settings. The main area is titled 'Triggers BETA'. It says 'Integrate your working repositories in order to start creating build triggers. Build triggers automatically build containers based on source code or tag changes in a repository.' Below this is a section for 'Repositories' with tabs for 'ACTIVE' (selected) and 'INACTIVE'. There is a 'Filter repositories' input field. Two triggers are listed:

Description	Event	Filter	Build configuration	Status	Action
Push to any branch	Push to branch	.*	cloudbuild.yaml	Enabled	Run trigger
Push to any branch (disabled)	Push to branch	.*	cloudbuild.yaml	Disabled	Run trigger

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/madhuaasnannaka/clourepo/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/$PROJECT_ID/nodejs:$COMMIT_SHA", "-t", "gcr.io/$PROJECT_ID/nodejs: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/$PROJECT_ID/nodejs:$COMMIT_SHA"]
- name: 'gcr.io/cloud-builders/docker'
  id: pushing_container_image_gke
  args: ["push", "gcr.io/$PROJECT_ID/nodejs: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/nodejs:latest
    - --location=us-east4-c
    - --cluster=standard-cluster-1
    - --app=nodejs
    - --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/clourepo/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



Create an organization

An organization is a space where a team or a whole company can collaborate across many projects.

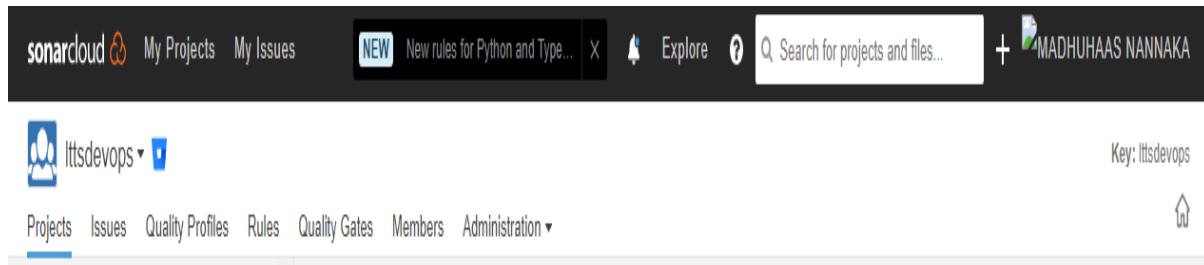


You will be asked to grant access to the SonarCloud application on your team or user account, which will allow you to choose which repositories you want to analyze.

Choose a team on Bitbucket

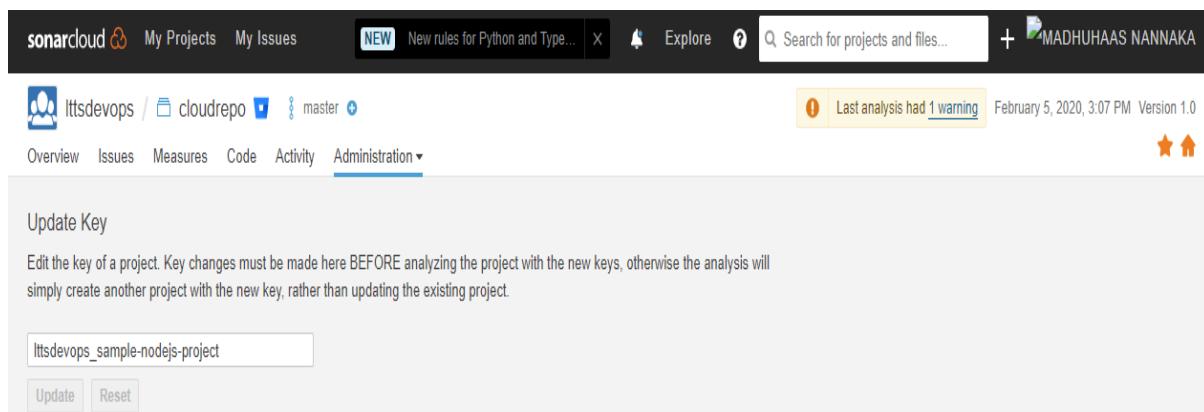
Just testing? You can [create manually](#).

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



The screenshot shows the SonarCloud dashboard for the organization 'ltsdevops'. At the top, there's a navigation bar with 'My Projects', 'My Issues', a 'New rules for Python and Type...' button, 'Explore', a search bar, and a user profile 'MADHUHAAS NANNAKA'. Below the navigation bar, the organization name 'ltsdevops' is displayed with a dropdown arrow. A horizontal menu bar follows with 'Projects', 'Issues', 'Quality Profiles', 'Rules', 'Quality Gates', 'Members', and 'Administration'. On the right side of the dashboard, there's a key icon labeled 'Key: ltsdevops'.

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



The screenshot shows the 'Administration' tab selected in the project navigation bar for the 'ltsdevops / clourepo' project. Under the 'Administration' section, there's a 'Update Key' form. It contains a text input field with the value 'ltsdevops_sample-nodejs-project', two buttons ('Update' and 'Reset'), and a status message indicating 'Last analysis had 1 warning' from February 5, 2020, at 3:07 PM, Version 1.0. There are also star and house icons on the right.

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': 'ltsdevops',
    'sonar.projectKey': 'ltsdevops_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

Kubernetes Engine Clusters

Kubernetes Engine API is being enabled. This may take a minute or more. [Learn more](#)

Kubernetes Engine
Kubernetes clusters

Containers package an application so that it can be easily deployed to run in its own isolated environment. Containers are managed in clusters that automate VM creation and maintenance. [Learn more](#)

Create cluster Deploy container Take the quickstart

STEP16:

HERE CLICK ON THE CREATE CLUSTER

Kubernetes Engine Clusters

Kubernetes Engine API is being enabled. This may take a minute or more. [Learn more](#)

Kubernetes Engine
Kubernetes clusters

Containers package an application so that it can be easily deployed to run in its own isolated environment. Containers are managed in clusters that automate VM creation and maintenance. [Learn more](#)

Create cluster Deploy container Take the quickstart

STEP17:

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

Cluster templates

Select a template with preconfigured setting, or customise a template to suit your needs

- Clone an existing cluster
- Standard cluster
- Your first cluster
- CPU intensive applications
- Memory intensive applications
- GPU Accelerated Computing
- Highly available

Standard cluster template

Continuous integration, web serving, backends. Best choice for further customisation or if you are not sure what to choose.

Some fields can't be changed after the cluster is created. Hover over the help icons to learn more.

Name: standard-cluster-1

Location type: Zonal

Zone: us-central1-a

Master version: 1.13.11-gke.14 (default)

Node pools

default-pool

Number of nodes: 3

Machine configuration

Machine family: General-purpose

Series: N1

Create Cancel Equivalent REST or command line

STEP18:

THE KUBERNETES CLUSTER IS CREATED

Kubernetes Engine	Kubernetes clusters																			
Clusters	CREATE CLUSTER																			
Workloads	A Kubernetes cluster is a managed group of VM instances for running containerised applications. Learn more																			
Services & Ingress	<input type="text"/> Filter by label or name																			
Applications	<table border="1"><thead><tr><th>Name</th><th>Location</th><th>Cluster size</th><th>Total cores</th><th>Total memory</th><th>Notifications</th><th>Labels</th></tr></thead><tbody><tr><td>standard-cluster-1</td><td>us-central1-a</td><td>3</td><td>3 vCPUs</td><td>11.25 GB</td><td></td><td></td></tr></tbody></table>						Name	Location	Cluster size	Total cores	Total memory	Notifications	Labels	standard-cluster-1	us-central1-a	3	3 vCPUs	11.25 GB		
Name	Location	Cluster size	Total cores	Total memory	Notifications	Labels														
standard-cluster-1	us-central1-a	3	3 vCPUs	11.25 GB																
Configuration	Connect																			
Storage																				

STEP19:

NOW GO TO IAM & ADMIN

Permissions for project TESTCICDGCP

Type	Member	Name	Role	Over-granted permissions	Inheritance
Compute Engine default service account	1008493335651-compute@developer.gserviceaccount.com	Compute Engine default service account	Editor		
Cloud Build Service Account	1008493335651@cloudbuild.gserviceaccount.com	Cloud Build Service Account	Editor		
Google APIs Service Agent	1008493335651@cloudservices.gserviceaccount.com	Google APIs Service Agent	Editor		
MADHUHAAS NANNAKA	madhuhaasnannaka@gmail.com	MADHUHAAS NANNAKA	Owner		
Compute Engine Service Agent	service-1008493335651@compute-system.iam.gserviceaccount.com	Compute Engine Service Agent	Editor		
Kubernetes Engine Service Agent	service-1008493335651@container-engine-robot.iam.gserviceaccount.com	Kubernetes Engine Service Agent	Editor		
Google Container Registry Service Agent	service-1008493335651@containerregistry.iam.gserviceaccount.com	Google Container Registry Service Agent	Editor		
Cloud Build Service Account	service-1008493335651@gcp-sa-cloudbuild.iam.gserviceaccount.com	Cloud Build Service Account	Cloud Build Service Account		

STEP20:

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

Edit permissions

Member: 1008493335651@cloudbuild.gserviceaccount.com Project: TESTCICDGCP

Role: **Cloud Build Service Account** Condition: [Add condition](#)

+ ADD ANOTHER ROLE

SAVE **CANCEL**

STEP21:

SEARCH FOR KUBERNETES ENGINE ADMIN AND CLICK ON SAVE

Edit permissions

Member 1008493335651@cloudbuild.gserviceaccount.com **Project** TESTCICDGCP

Role	Condition
Cloud Build Service Accou...	Add condition
Kubernetes Engine Admin	Add condition

Role Kubernetes Engine Admin **Condition** Add condition

Full management of Kubernetes Clusters and their Kubernetes API objects.

SAVE **CANCEL**

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

Type	Member	Name	Role	Over-granted permissions	Inheritance
Compute Engine default service account	1008493335651-compute@developer.gserviceaccount.com	Compute Engine default service account	Editor		
Cloud Build Service Account	1008493335651@cloudbuild.gserviceaccount.com	Cloud Build Service Account	Kubernetes Engine Admin		
Google APIs Service Agent	1008493335651@cloudservices.gserviceaccount.com	Google APIs Service Agent	Editor		
	madhuhaasnannaka@gmail.com	MADHUHAAS NANNAKA	Owner		
	service-1008493335651@compute-system.iam.gserviceaccount.com	Compute Engine Service Agent	Compute Engine Service Agent		

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

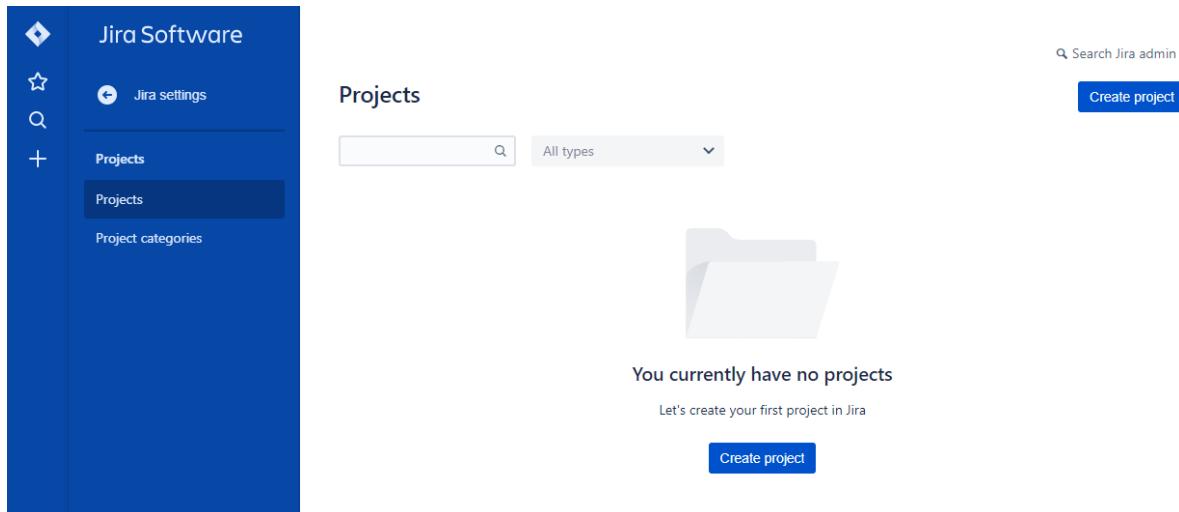
The screenshot shows a Bitbucket commit history page. On the left is a sidebar with navigation links: Source, Commits (which is selected), Branches, Pull requests, Pipelines, Deployments, Downloads, and Settings. The main content area shows a commit by MADHUHAAS NANNAKA with the commit ID fc78f65. The commit message is "Dockerfile edited online with Bitbucket". To the right are buttons for "View source" and "Approve". Below the commit details, there's a user profile for MADHUHAAS NANNAKA, a branch master, and a note about no tags. At the bottom are links for "View raw commit", "Stop watching", and "Run pipeline".

The screenshot shows the Google Cloud Platform Cloud Build interface. The top bar includes the Google Cloud logo, project name "cicddemo", a search bar, and a menu with a count of 7 notifications. The main area has tabs for "History" (selected) and "Triggers". The "History" tab displays a successful build log for commit "3428f80c" started on 5 Feb 2020 at 11:43:59. The log table has columns for Steps, Duration, and three tabs: BUILD LOG, EXECUTION DETAILS, and BUILD ARTEFACTS. The BUILD LOG tab shows a detailed list of build steps, each with a green checkmark indicating success. Step 0 installs npm, Step 1 builds with npm, Step 2 runs unit tests, Step 3 installs sonar-scanner, Step 4 installs typescript, Step 5 runs sonarpush, Step 6 builds an image using gcr.io/cicddemo..., Step 7 pushes the image to gcr.io/cicddemo..., Step 8 pushes a container to gcr.io/cicddemo..., and Step 9 deploys the container. The log output shows commands like "npm install", "node-sass@4.7.2 install", and "phantomjs-prebuilt@2.1.16 install". The EXECUTION DETAILS and BUILD ARTEFACTS tabs are also visible.

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

Create project

Name

Key
 i

Share settings with an existing project

Template

Scrum
Manage stories, tasks, and workflows for a scrum team - For teams that deliver work on a regular schedule

Change template

Create

STEP26:

NOW GO TO ADD ITEM AND SELECT REPOSITORY AND CONNECT IT

The screenshot shows the Jira interface with a sidebar on the left containing project navigation links like 'cicd Software project', 'CICD board Board', 'Backlog', 'Active sprints' (which is selected), and 'Reports'. The main area displays 'Active sprints' with a 'TO DO' section. A modal window titled 'Connect a repository' is open, prompting the user to link a repository. It contains a description: 'Linking a repository will show information about your branches, commits and pull requests in Jira issues.', a 'Repository link' input field containing 'https://bitbucket.org/madhuhaasnannaka/cludrepo', a 'Name' input field containing 'cludrepo', and two buttons at the bottom: 'Connect' (highlighted in blue) and 'Cancel'.

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

The screenshot shows the Jira interface with the same sidebar as before. The main area is now the 'Create issue' dialog. It includes fields for 'Project' (set to 'cicd (CICD)'), 'Issue Type' (set to 'Task'), 'Summary' (an empty text input), 'Components' (set to 'None'), 'Attachment' (a file upload area), and 'Description' (a rich text editor). At the bottom of the dialog are buttons for 'Create another' (disabled), 'Create' (highlighted in blue), and 'Cancel'.

STEP28:

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

The screenshot shows the Jira Software project interface. On the left, there's a sidebar with various project management options like CICD board, Backlog, Active sprints, Reports, Releases, Issues and filters (which is selected), Pages, Components, cloudrepo, Add item, and Project settings. The main area displays an issue titled "CICD-1" under the "Changes to be made" section. The issue has a status of "To Do". It includes fields for Assignee (Unassigned), Reporter (MADHUHAAS NANNAKA), Development (Create branch), Labels (None), and Priority (Medium). A note at the top says, "We're updating the issue view to help you get more done. Learn more or See the old view". There's also a "Show 6 more fields" link.

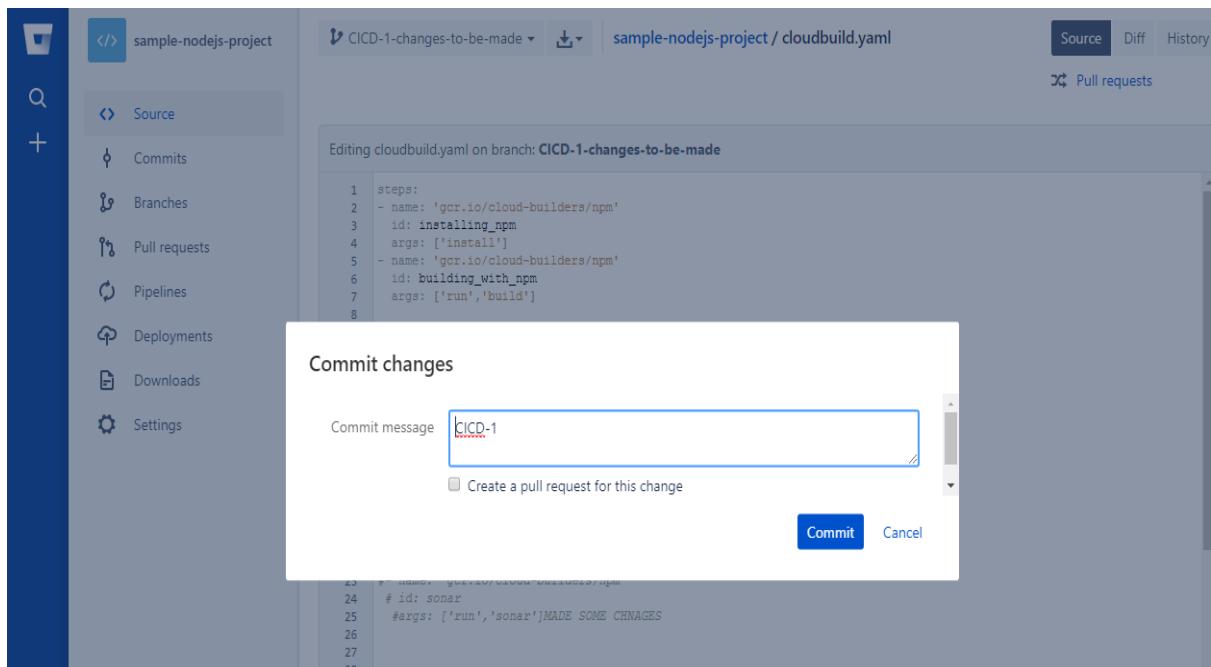
STEP29:

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

This screenshot shows the same Jira interface as above, but with changes. The "Assignee" field now lists "MADHUHAAS NANNAKA". The "Development" field shows "1 branch". The rest of the issue details remain the same: Reporter (MADHUHAAS NANNAKA), Labels (None), and Priority (Medium). The "Save" button is visible at the bottom of the issue view.

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

A screenshot of a Jira issue page for 'CICD-1'. At the top, there's a message about updating the issue view. Below that, the issue key 'CICD-1' is checked. To the right, there are buttons for '1' (status), a share icon, and three dots. The title is 'CHANGES TO BE MADE'. Below the title are buttons for 'Attach', 'Create subtask', 'Link issue', 'Link page', and three dots. The 'Description' section has a placeholder 'Add a description...'. The 'Activity' tab is selected, showing a comment input field with 'Add a comment...' and a note 'Pro tip: press M to comment'. To the right, under 'Assignee', it says 'MADHUHAAS NANNAKA'. Under 'Reporter', it also says 'MADHUHAAS NANNAKA'. Under 'Development', it shows '1 branch', '1 commit' (with a timestamp '56 seconds ago'), and a 'Labels' section with 'None'.

cicd Software project

We're updating the issue view to help you get more done. [Learn more](#) or [See the old view](#)

Development CICD-1

Branches [Commits](#) Pull requests Builds Deployments Feature flags

sample-nodejs-project (Bitbucket)

Author Commit Message Date Files

	456b054	CICD-1	02/05/2020	1 file
--	---------	--------	------------	--------

Show all files

NAKA
NAKA
2 minutes ago
tracking, Epic Link...
Configure

This screenshot shows a commit detail view in Jira. The commit is titled 'CICD-1' and was made by user '456b054' on 02/05/2020. The commit message is 'CICD-1'. There is one file associated with this commit. The interface includes tabs for Branches, Commits, Pull requests, Builds, Deployments, and Feature flags. A sidebar on the left lists various project management options like CICD Board, Backlog, Actions, Reports, Release, Issues, Pages, Configuration, Cloud, Add-ons, and Project.

STEP31:

NOW CLICK ON THE PULL REQUEST AND SELECT CREATE PULL REQUEST

cicd Software project

We're updating the issue view to help you get more done. [Learn more](#) or [See the old view](#)

Development CICD-1

Branches [Commits](#) [Pull requests](#) Builds Deployments Feature flags

Pull requests at a glance

Pull requests related to Jira issues will appear here for your team to track in one place.

Create pull request Learn more

This screenshot shows the 'Pull requests' tab in the Jira software project interface. It displays a placeholder message 'Pull requests at a glance' with a sub-instruction 'Pull requests related to Jira issues will appear here for your team to track in one place.' Below this are two buttons: 'Create pull request' and 'Learn more'. The interface is identical to the previous screenshot but with the 'Pull requests' tab selected.

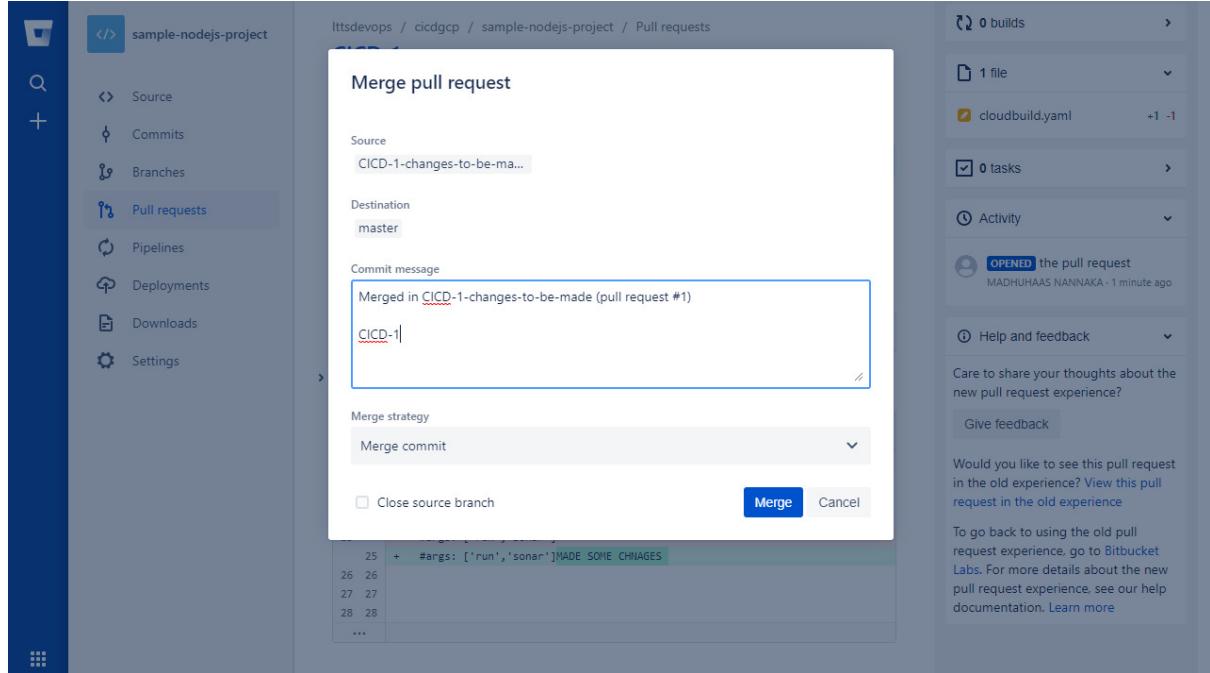
WHERE IT TAKES YOU TO THE BITBUCKET TO MAKE MERGE TO THE MASTER

STEP32:

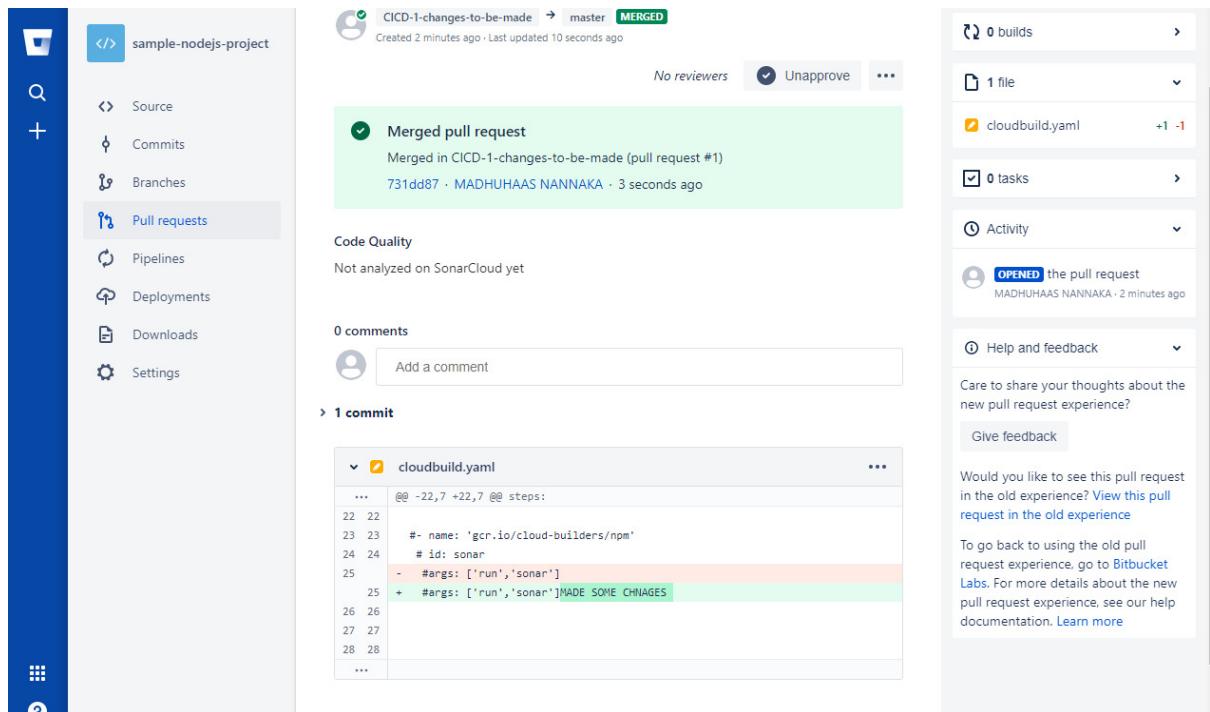
HERE CLICK ON THE CREATE PULL REQUEST AGAIN WHERE IT TAKES YOU TO A PAGE IN BITBUCKET TO APPROVE THE PULL REQUEST

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

1 const Mailgun = require('mailgun-js');
2 const humanizeDuration = require('humanize-duration');
3
4 const mailgun = Mailgun({
5   apiKey: '3035ae0db864f55a303e059986e53d20-9dfbeecd-bb8ba534',
6   domain: 'sandbox452769abe42d45559711656b80b03969.mailgun.org',
7 });
8
9 // subscribeMailgun is the main function called by Cloud Functions.
10 module.exports.subscribeMailgun = (pubSubEvent, context) => {
11   const build = eventToBuild(pubSubEvent.data);
12
13   // Skip if the current status is not in the status list.
14   const status = ['SUCCESS', 'FAILURE', 'INTERNAL_ERROR', 'TIMEOUT'];
15   if (status.indexOf(build.status) === -1) {
16     return;
17   }
18
19   // Send email.
20   const message = createEmail(build);
21   mailgun.messages().send(message, (error, body) => console.log(body.message));
22 }
23
24 // eventToBuild transforms pubsub event message to a build object.
25 const eventToBuild = (data) => {
26   return JSON.parse(Buffer.from(data, 'base64').toString());
27 }
28
29 // createEmail creates an email message from a build object.
30 const createEmail = (build) => {
31   const duration = humanizeDuration(new Date(build.finishTime) - new Date(build.startTime));
32   const msgText = `Build Id: ${build.id}<br>Build Status: ${build.status}<br>Build Duration: ${duration}.`;
33   let msgHtml = `<p>${msgText}</p><p>To see <a href="${build.logUrl}">Build logs</a></p>`;
34   if (build.images) {
35     const images = build.images.join(',');
36     msgHtml += `<p>Images: ${images}</p>`;
37   }
38   const message = {
39     from: 'bhaskaranesh22@gmail.com',
40     to: 'madhuhaasnannaka@gmail.com',
41     subject: 'Build Status',
42     text: msgText,
43     html: msgHtml
44   };
45   return message;

```

Function to execute [?](#)

subscribeMailgun

Environment variables, networking, timeouts and more

```

index.js  package.json

1 {
2   "name": "google-container-email",
3   "version": "0.0.1",
4   "description": "Email integration for Google Cloud Build, using Google Cloud Functions",
5   "main": "index.js",
6   "dependencies": {
7     "humanize-duration": "3.10.0",
8     "mailgun-js": "~0.11.2"
9   },
10  "devDependencies": {
11    "async": "^2.1.5",
12    "mocha": "3.2.0",
13    "should": "11.2.1"
14  }
15 }

```

function-1

Memory allocated

256 MB

Trigger

Cloud Pub/Sub

Topic

cloud-builds

Source code

- Inline editor
- ZIP upload
- ZIP from Cloud Storage
- Cloud Source repository

Runtime

Node.js 8

[index.js](#) [package.json](#)

```
1 const Mailgun = require('mailgun-js');
2 const humanizeDuration = require('humanize-duration');
3
4 const mailgun = Mailgun({
5   apiKey: '3035ae0db864ff55a303e059986e53d20-9dfbeecd-bb8ba534',
6   domain: 'sandbox452769abe42d45559711656b80b03969.mailgun.org'.
7 });
8
9 // subscribeMailgun is the main function called by Cloud Function
10 module.exports.subscribeMailgun = (pubSubEvent, context) => {
11   const build = eventToBuild(pubSubEvent.data);
12
13   // Skip if the current status is not in the status list.
14   const status = ['SUCCESS', 'FAILURE', 'INTERNAL_ERROR', 'TIMEOUT'];
15   if (status.indexOf(build.status) === -1) {
16     return;
17   }
18
19   // Send email.
20   const message = createEmail(build);
21   mailgun.messages().send(message, (error, body) => console.log(
22 );
```

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.

The screenshot shows the Google Cloud Platform Cloud Build interface. The build status is "Successful: ebd1562e". The build was triggered by "Push-to-any-branch" and the source is "madhuhasnannaka/clourepo". The branch is "master" and the commit ID is "5489280". The build log shows a summary of 7 steps, including Maven package, unit tests, SonarCloud code analysis, building the image, pushing it to GCR, pushing the container, and deploying it. The logs also show the download of Spring Boot dependencies from the Maven repository.

Steps	Duration
Build summary	00:06:57
0: maven_package	00:01:23
1: unit_tests	00:01:32
2: sonar_cloud_code...	00:02:33
3: building_image_us...	00:00:10
4: pushing_image_gcr	00:00:06
5: pushing_container...	00:00:01
6: deploying_contain...	00:01:03

BUILD LOG		EXECUTION DETAILS	BUILD ARTEFACTS		
<input checked="" type="checkbox"/> Wrap lines <input type="checkbox"/> Show newest entries first		T	VIEW		
1 starting build "ebd1562e-c320-4b70-9dce-bf496d6b424b" 2 3 FETCHSOURCE 4 Fetching storage object: gs://987293381995.cloudbuild-source.googleusercontent.com/548928040b13915421439cf86dc522477455059d-96c0fd1b-35bc-4b6e-9146-b7e4ff394454.tar.gz#1580896480746708 5 Copying gs://987293381995.cloudbuild-source.googleusercontent.com/548928040b13915421439cf86dc522477455059d-96c0fd1b-35bc-4b6e-9146-b7e4ff394454.tar.gz#1580896480746708... 6 / [0 files][0.0 B/397.9 KiB] 7 / [1 files][397.9 KiB/397.9 KiB] 8 Operation completed over 1 objects/397.9 KiB. 9 BUILD 10 Starting Step #0 - "maven_package" 11 Step #0 - "maven_package": Already have image (with digest): gcr.io/cloud-builders/mvn:3.5.0-jdk-8 12 Step #0 - "maven_package": [INFO] Scanning for projects... 13 Step #0 - "maven_package": Downloading: https://repo.spring.io/snapshot/org/springframework/boot/spring-boot-starter-parent/2.1.6.RELEASE/spring-boot-starter-parent-2.1.6.RELEASE. 14 Step #0 - "maven_package": Downloading: https://repo.spring.io/milestone/org/springframework/boot/spring-boot-starter-parent/2.1.6.RELEASE/spring-boot-starter-parent-2.1.6.RELEASE. 15 Step #0 - "maven_package": Progress (1): 0.6/9.5 kB 16 Progress (1): 1.9/9.5 kB 17 Progress (1): 3.3/9.5 kB 18 Progress (1): 4.7/9.5 kB 19 Progress (1): 6.0/9.5 kB 20 Progress (1): 7.4/9.5 kB					

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

The screenshot shows an email from bhaskarganesh222@gmail.com. The email subject is "via sandbox.mgsend.net to me". The body of the email provides build details: Build Id: ebd1562e-c320-4b70-9dce-bf496d6b424b, Build Status: SUCCESS, and Build Duration: 6 minutes, 57.87 seconds. It also includes a link to see the build logs.

Build Id: ebd1562e-c320-4b70-9dce-bf496d6b424b
Build Status: SUCCESS
Build Duration: 6 minutes, 57.87 seconds.
To see [Build logs](#)

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*****