

# Tutorial for GOCD and Firebase

Dingli Mao [dingli@kth.se](mailto:dingli@kth.se)

Md. Rezaul Hasan [mrhasa@kth.se](mailto:mrhasa@kth.se)

Akshay Sinha [akshays@kth.se](mailto:akshays@kth.se)

## Introduction:

### GOCD

GOCD is an open source continuous integration and continuous delivery platform. Continuous Integration means the ability to integrate code into a shared repository frequently.[1] Continuous Delivery ensures that the software is in a deployable state at any point in time. [2] GOCD includes a server and one or more agents. The server controls everything. It provides the user interface for the users and the work for the agents. The agents are the ones that run commands, do deployments etc. They do all the work. Therefore, you need a server and at least one agent to work with GOCD.

In GOCD, there is only one server. The user can access the user interface of the server by using a specific URL. The server triggers the pipeline once there are any changes in the project repository. It assigns the jobs to different agents. It does not run any commands or do deployments. On the other hand, once the agent receives the job from the agent, the client runs a list of tasks of the job, such as running the command lines and deploying the project. After that, the client reports the status of the job to the server. Therefore, the server decides the stage of the job based on the information from the client.

### FireBase Hosting

FireBase Hosting is a production-ready platform which enables the developer to deploy their web applications. The application served is automatically scaled according to the number of users. It uses a CDN (Content Delivery Network) to serve assets faster to edge locations. One could classify this technology as serverless.

### Overview

In this tutorial we will be using GOCD platform to build and deploy a frontend application to firebase hosting. We will create two pipelines; the first pipeline (tutorial-pipeline) builds and tests the application. The second pipeline (second-pipeline) uses the firebase-tools package to deploy automatically to firebase hosting service. We use a forked version of a repository to trigger the first pipeline. The repository is continuously polled and any commits to the fork triggers the pipeline. The second pipeline is triggered automatically as tutorial-pipeline is given as material to it.

Repository we used to test in the following tutorial:

[https://github.com/YonatanKra/web-components-ui-elements/tree/before\\_ci\\_cd](https://github.com/YonatanKra/web-components-ui-elements/tree/before_ci_cd)

Before using this repository, please fork it to your own github account!

## Background:

While it is difficult to define the field of DevOps, it can be said that it is a set of practices that combines DEvelopment of software with the OPerational side of maintaining the software. It aims to provide high quality software while simultaneously shortening the systems development life cycle. [3]

Automated tests are tests performed using software which ensures that the expected outcome is equal to the actual outcome. It is used to improve software quality and ensure that the code can be integrated into the repository while minimizing bugs. Some testing tasks can be very laborious to do manually and therefore benefit from being automated. It is common to automate unit tests, integration tests and system tests. Automated testing provides a speedy solution compared to manual testing. [4]

Automated builds are the compiling or transformation of code to binary packages in an automated fashion. It can also be used to build files for deployment to a platform such as the web. Even though there is no compilation as such, it minifies and uses a transpiler to ensure that the code is rendered correctly by all browsers. It is an essential step for Continuous Integration/Continuous Delivery pipelines. [5]

GoCD is a popular tool for continuous integration and continuous delivery. By creating this tutorial, it is helpful to learn the basic steps of operating CI/CD on a project. Since CI and CD are the core concepts of DevOps, studying GoCD also enables people to know the importance and benefits of using DevOps while developing the application.

## Part 1: Installation:

**Link of downloading GOCD:** <https://www.gocd.org/download>

GOCD is available on Windows, OSX, Linux and other cloud providers.

In order to GOCD, one server and at least one client are needed to be installed.

The following steps of installing and configuring GOCD is based on Windows OS.

### Installing GOCD server:

After running the GOCD-server.exe package, it will be installed locally. You can access it using URL: localhost:8153 to check if it is successfully installed on your computer.

## Installing GOCD client

Next, we run GOCD-agent.exe executable. It is also installed locally. When you switch to the “Agents” tab at the server interface, you should see your installed agent listed as “IDLE”. After you start running the client, the client can connect to the server by typing the URL of the server (localhost:8153/go).

## Part 2: Creation and configuration of a new pipeline

### Step 1:

This step is to configure the material. Material is a project repository which is source controlled. GOCD keeps checking the changes of the materials and triggers the pipeline. Typically, a material is a source repository. We choose the github repository as the material in this tutorial. A pipeline can be a material as well.

### Part 1: Material

\* denotes a required field

Material Type\*

Git

Repository URL\*

<https://github.com/MDRezaulHasan/devops-tutorial-tes>

Test Connection

Connection OK

Advanced Settings

Material = Source Repository

A **material** triggers your pipeline to run. Typically this is a **source repository** or an **upStream pipeline**.

### Step 2:

This step is used to configure the pipeline. Pipeline is a workflow which contains a list of stages. Every stage is running in order. If one stage fails, the following stages will not be executed.

### Part 2: Pipeline Name

\* denotes a required field

Pipeline Name\*

No spaces. Only letters, numbers, hyphens, underscores, and periods. Max 255 chars.

▼ Advanced Settings


Pipeline Group\*

Use Template: ☐

Parameters ?

NAME	VALUE
<input type="text"/>	<input type="text"/>

+ Add



In GoCD, a **pipeline** is a representation of a **workflow**. Pipelines consist of one or more **stages**.

## Step 3:

In this step, one stage is configured. Every stage includes a series of jobs. Every job is independent. It means that GOCD can execute several jobs at the same time. If one job fails, this stage will fail but it won't influence the execution of other jobs.

### Part 3: Stage Details

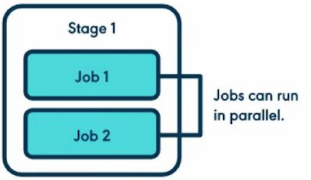
\* denotes a required field

Stage Name\*

No spaces. Only letters, numbers, hyphens, underscores, and periods. Max 255 chars.

▼ Advanced Settings

Automatically run this stage on upstream changes ? ☒



A **stage** is a group of jobs, and a **job** is a piece of work to execute.

## Step 4 :

A job is part of a stage. It is a sequence of tasks. A task is a command which runs in a job. If a task fails then all the following tasks will not be executed. The job is then considered as "failed". In this step, a job is configured to build and test our application.

Here is the command we used:

```
npm install
npm run build
npm run test:prod
```

## Part 4: Job and Tasks

\* denotes a required field

Job Name\*

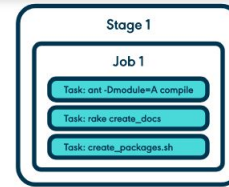
create-testing-file

No spaces. Only letters, numbers, hyphens, underscores, and periods. Max 255 chars.

Type your tasks below at the prompt\*

```
+ Caveats
+ Help
# Press <enter> to save, <shift-enter> for newline
$ npm install
$ npm run build
$ npm run test:prod
$
```

➤ Advanced Settings



A **job** is like a script, where each sequential step is called a **task**. Typically, a task is a single command.

»» tutorial-pipeline » Test-devops-tutorial-1 » create-testing-file

Job Settings Tasks Artifacts Environment Variables Custom Tabs

Tasks

Order	Task Type	Run If Conditions	Properties	On Cancel	Remove
▼	Custom Command	Passed	Command: npm Arguments: install	No	✖
▲	Custom Command	Passed	Command: npm Arguments: run build	No	✖
▲	Custom Command	Passed	Command: npm run test:prod	No	✖
+ Add new task ▼					

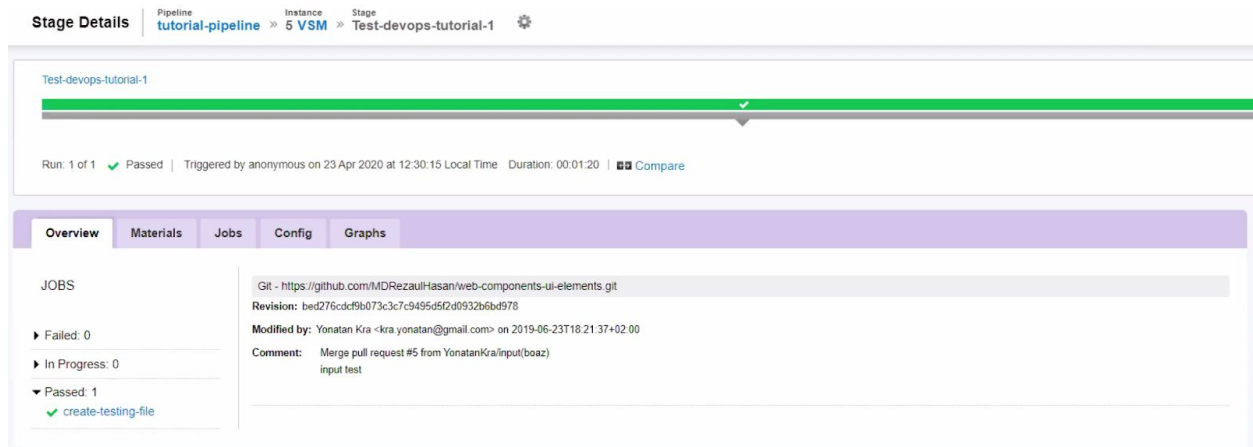
Here is the page after you successfully configure one pipeline.

The screenshot shows the GoCD dashboard with a green banner celebrating the creation of the first pipeline. The 'defaultGroup' section displays the 'tutorial-pipeline' with its history and instance details. The instance was triggered by an anonymous user on 23 Apr, 2020 at 11:58:33 Local Time.

Copyright © 2020 Thoughtworks, Inc. Licensed under Apache License, Version 2.0. GoCD includes third-party software.  
GoCD Version: 20.3.0 (11530-f54d0bae7a39fe7add5460f3f8676a7936abf698).

## Part 3: Execution of pipelines and Result

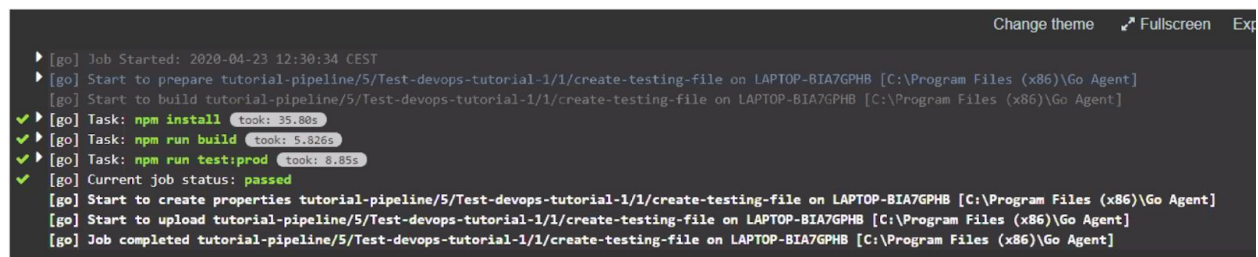
After all the tasks in a pipeline are executed, you can view the result by clicking the status button. The following screenshot shows that one job named “create-testing file” was successfully executed.



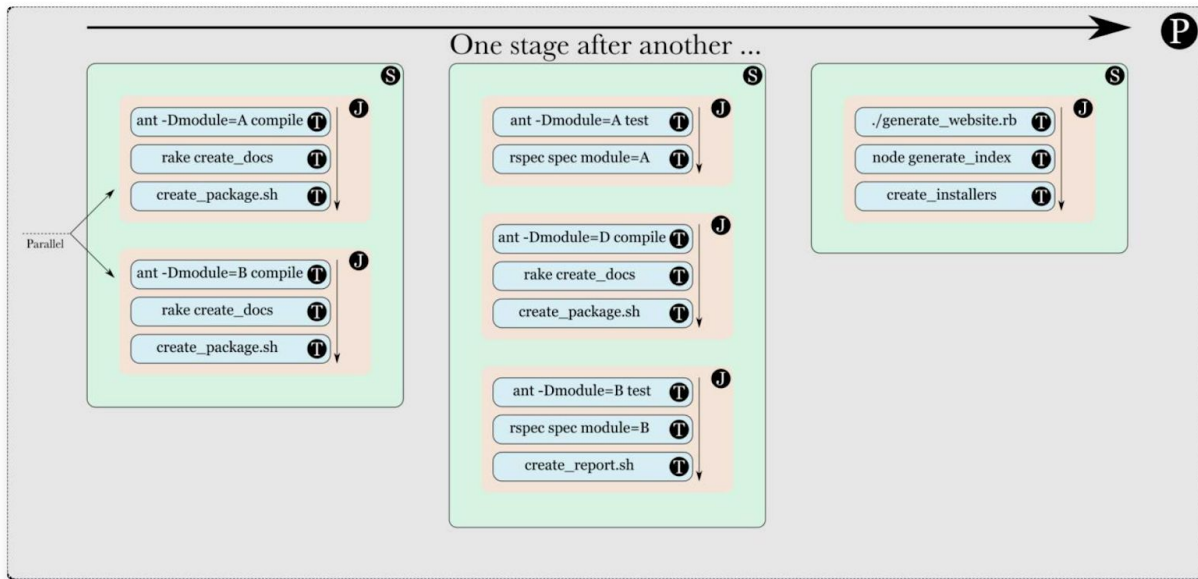
By clicking job, “create-testing-file”, the input and the output are revealed. In this case, the following command lines are successfully executed:

```
npm install
npm run build
npm run test:prod
```

The result is that all the tests are passed

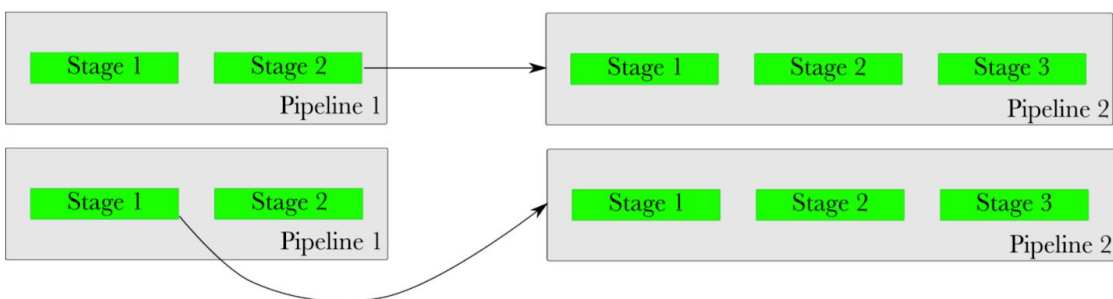


After completing the above tutorial, you may have a general understanding of pipeline, stage, job and task. The following graph also shows the relationship between pipeline(P), stage(S), job(J) and task(T):



## Part 4: Using a pipeline as a material

In the following guide, using a pipeline as a material will be revealed. It means that a downstream pipeline is chained with a stage of upstream pipeline. The downstream pipeline does not have to chain with the last stage of the upstream pipeline. Any other stages in the upstream can be connected with the downstream pipeline. The following picture demonstrates the relationship between two pipelines.



Pic 1: The relationship between the pipelines

## Artifact

Before configuring the second(downstream) pipeline, the artifact of the first(upstream) pipeline should be configured. Artifact is a file or a folder which represents the output of a pipeline. After configuring an artifact, GoCD makes sure that the artifact will be moved from agent to server so that the downstream pipeline can use it later.

## Add artifact in Upstream pipeline

The upstream pipeline has installed all the packages the demo project depends on. Therefore, we have to configure the artifact in the first pipeline in the following way so that all required files of the project are accessible to the downstream pipeline.

Pipelines » tutorial-pipeline **II**

Type ?	Source ?	Destination ?	
Build artifact	dist/		✖
Build artifact	deploy.js		✖
Build artifact	package.json		✖
Build artifact	firebase.json		✖
Build artifact	.firebaserc		✖

Select type: Build ▼ ➕ Add

The following step 1-3 is to create another pipeline(downstream pipeline).

### Step 1

Instead of choosing Git repository as a material, the pipeline is chosen as the material this time. In this case, the first pipeline(upstream pipeline) is chosen.



## Part 1: Material

\* denotes a required field

Material Type\*

Another Pipeline ▼

Upstream Pipeline\*

tutorial-pipeline

Upstream Stage\*

Test-devops-tutorial-1 ▼

➤ Advanced Settings

## Step 2 and Step 3

The name of the second pipeline and stage are configured.

## Part 2: Pipeline Name

\* denotes a required field

Pipeline Name\*

Secound-Pipeline

No spaces. Only letters, numbers, hyphens, underscores, and periods. Max 255 chars.

➤ Advanced Settings

## Part 3: Stage Details

\* denotes a required field

Stage Name\*

Pipeline-2-Stage-1

No spaces. Only letters, numbers, hyphens, underscores, and periods. Max 255 chars.

➤ Advanced Settings

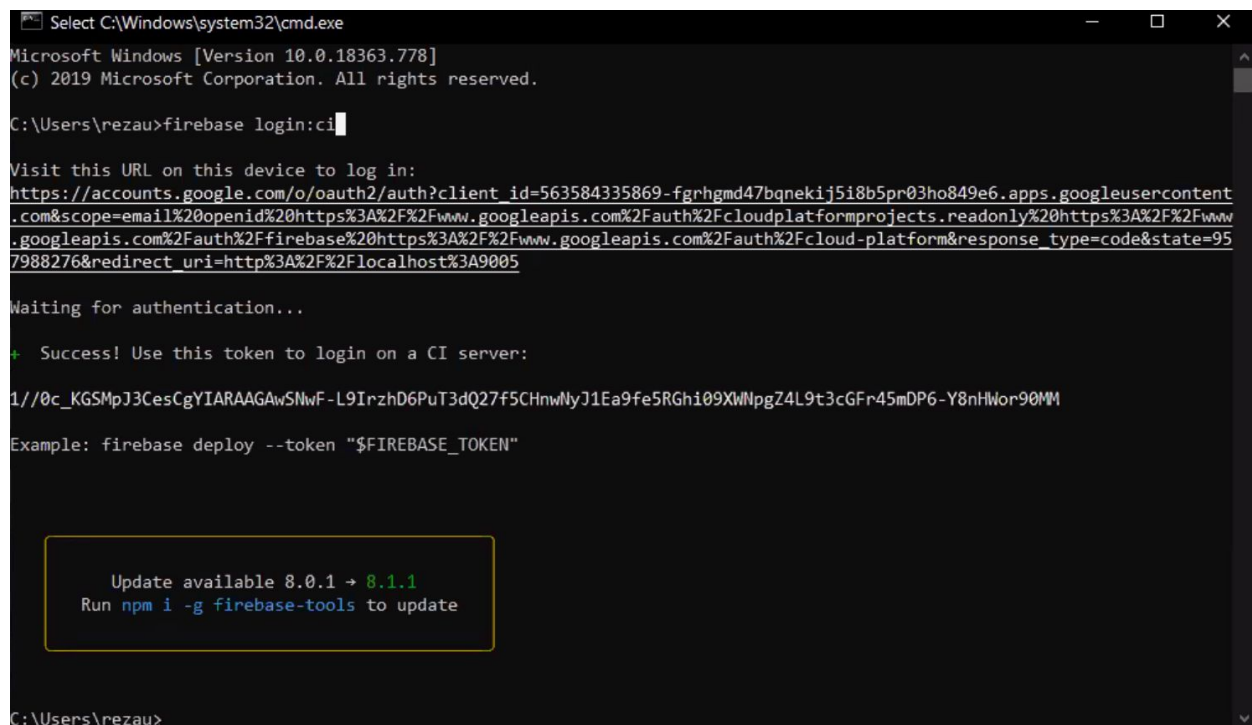
## Step 4 Generate Firebase token

After configuring the second pipeline, we generate the token of firebase. The firebase platform will be used later to deploy this project.

The following command lines are used to generate a firebase token.

```
npm install --global firebase-tools
firebase login:ci
```

The following picture shows the example of successfully generating a token of firebase.



```
Select C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.18363.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\rezau>firebase login:ci

Visit this URL on this device to log in:
https://accounts.google.com/o/oauth2/auth?client_id=563584335869-fgrhgm47bqnekij5i8b5pr03ho849e6.apps.googleusercontent.com&scope=email%20openid%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloudplatformprojects.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Ffirebase%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloud-platform&response_type=code&state=957988276&redirect_uri=http%3A%2F%2Flocalhost%3A9005

Waiting for authentication...

+ Success! Use this token to login on a CI server:

1//0c_KGSmpJ3CesCgYIARAAGAwSNwF-L9IrnzhD6PuT3dQ27f5CHnwNlyJ1Ea9fe5RGhi09XWNpgZ4L9t3cGFr45mDP6-Y8nHlWor90MM

Example: firebase deploy --token "$FIREBASE_TOKEN"

Update available 8.0.1 -> 8.1.1
Run npm i -g firebase-tools to update

C:\Users\rezau>
```

## Step 5 Add Deploy.js to Git repo.

The Firebase CLI can be used programmatically as a standard Node module. This file includes the token and project name created on firebase platform. Therefore, this test github repo can be deployed on firebase.

```
var client = require("firebase-tools");
client
  .deploy({
    project: "devopstutorial-d9959",
```

```

token:

"1//Oc_KGSMpJ3CesCgYIARAAGAwSNwF-L9IrzhD6PuT3dQ27f5CHnwNyJ1Ea9fe5RGhi09XWNpgZ4L9t3cGFr
45mDP6-Y8nHWor90MM",

  force: true,
})
.then(function () {
  console.log("Rules have been deployed!");
})
.catch(function (err) {
  // handle error
  console.log("Error! " + err);
});

```

Other than adding deploy.js file, “.firebase” file need to be changed to match the name of the project created on firebase.

```

{
  "projects": {
    "default": "devoptutorial-d9959"
  }
}

```

## Step 6 Fetch artifact in Downstream pipeline :

In order to use the output file after running the pipeline, the user has to create several new tasks of which type is fetch artifact. Those tasks can obtain all the files of the project and installed package by the first pipeline.

Pipelines » Secound-Pipeline II

Secound-Pipeline » Pipeline-2-Stage-1 » Job-2

Job Settings Tasks Artifacts Environment Variables Custom Tabs

Tasks

Order	Task Type	Run If Conditions	Properties	On Cancel	Remove
▼	Fetch Artifact	Passed	Pipeline Name: tutorial-pipeline Stage Name: Test-devops-tutorial-1 Job Name: create-testing-file Source File: deploy.js	No	✖
▲	Fetch Artifact	Passed	Pipeline Name: tutorial-pipeline Stage Name: Test-devops-tutorial-1 Job Name: create-testing-file Source File: package.json	No	✖
▲	Fetch Artifact	Passed	Pipeline Name: tutorial-pipeline Stage Name: Test-devops-tutorial-1 Job Name: create-testing-file Source Directory: dist	No	✖
▲	Fetch Artifact	Passed	Pipeline Name: tutorial-pipeline Stage Name: Test-devops-tutorial-1 Job Name: create-testing-file Source File: firebase.json	No	✖
▲	Fetch Artifact	Passed	Pipeline Name: tutorial-pipeline Stage Name: Test-devops-tutorial-1 Job Name: create-testing-file Source File: firebase.rc	No	✖
▲	Custom Command	Passed	Command: npm install	No	✖
▲	Custom Command	Passed	Command: node deploy.js	No	✖

➕ Add new task ▼

After configuring the artifacts, you are able to run the second pipeline.  
Here is the log file of successfully running second(downstream) pipeline:

The screenshot shows the 'Job Details' page for a pipeline named 'Second-Pipeline'. The job is 'Job-2' in stage 'Pipeline-2-Stage-1 / 1'. It was scheduled on 23 Apr 2020 at 14:46:07 and completed at 14:47:31. The agent is 'LAPTOP-BIA7GPHB (127.0.0.1)' and the build cause is 'triggered by tutorial-pipeline/6/Test-devops-tutorial-1/1'. The console log shows the following steps:

- [go] Job Started: 2020-04-23 14:46:21 CEST
- [go] Start to prepare Second-Pipeline/8/Pipeline-2-Stage-1/1/Job-2 on LAPTOP-BIA7GPHB [C:\Program Files (x86)\Go Agent]
- [go] Start to build Second-Pipeline/8/Pipeline-2-Stage-1/1/Job-2 on LAPTOP-BIA7GPHB [C:\Program Files (x86)\Go Agent]
- [go] Task: fetch artifact [package.json] => [] from [tutorial-pipeline/Test-devops-tutorial-1/create-testing-file] (took: 0.52s)
- [go] Task: fetch artifact [dist] => [dist] from [tutorial-pipeline/Test-devops-tutorial-1/create-testing-file] (took: 5.54s)
- [go] Task: fetch artifact [firebase.json] => [] from [tutorial-pipeline/Test-devops-tutorial-1/create-testing-file] (took: 0.47s)
- [go] Task: fetch artifact [firebase] => [] from [tutorial-pipeline/Test-devops-tutorial-1/create-testing-file] (took: 0.25s)
- [go] Task: npm install (took: 51.58s)
- [go] Task: node deploy.js (took: 33.284s)
- [go] Current job status: passed
- [go] Start to create properties Second-Pipeline/8/Pipeline-2-Stage-1/1/Job-2 on LAPTOP-BIA7GPHB [C:\Program Files (x86)\Go Agent]
- [go] Start to upload Second-Pipeline/8/Pipeline-2-Stage-1/1/Job-2 on LAPTOP-BIA7GPHB [C:\Program Files (x86)\Go Agent]
- [go] Job completed Second-Pipeline/8/Pipeline-2-Stage-1/1/Job-2 on LAPTOP-BIA7GPHB [C:\Program Files (x86)\Go Agent]

The project has been successfully deployed on Firebase, by clicking devops <https://devops-gocd.web.app/>, you can view the website. We decided not to show the website here. Please consider it as an easter egg!

The screenshot shows the 'Hosting' section of the 'DevOpsTutorial' dashboard. It displays a table of domains and a release history.

Domain	Status
devoptutorial-d9959.web.app	Default
devoptutorial-d9959.firebaseio.com	Default

devoptutorial-d9959 release history

Status	Time	Deploy	Files
★ Current	23 Apr 2020 14:47	rezauhasan0168@gmail.com a72966	7
📄 Deployed	23 Apr 2020 14:16	rezauhasan0168@gmail.com f8154c	6

## References:

1. <https://codeship.com/continuous-integration-essentials>
2. <https://continuousdelivery.com/>
3. <https://en.wikipedia.org/wiki/DevOps>
4. [https://en.wikipedia.org/wiki/Test\\_automation](https://en.wikipedia.org/wiki/Test_automation)
5. [https://en.wikipedia.org/wiki/Build\\_automation](https://en.wikipedia.org/wiki/Build_automation)
6. <https://stackshare.io/stackups/azure-devops-vs-go-cd>
7. <https://www.gocd.org/jenkins/>
8. <https://www.gocd.org/2017/04/25/gocd-over-jenkins.html>
9. <https://www.educba.com/gocd-vs-jenkins/>