Tutorial for GOCD

Dingli Mao dingli@kth.se
Md. Rezaul Hasan mrhasa@kth.se
Akshay Sinha akshays@kth.se

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. [1]

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. [2]

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. [3]

Continuous Integration means the ability to integrate code into a shared repository frequently. It can take place several times a day. This is accompanied by automated builds and automated tests. This allows us to detect and locate errors more quickly. [4]

Continuous Delivery ensures that the software is in a deployable state at any point in time. This means shipping out new features, bug fixes, configuration changes, etc. to the end user quickly and in a sustainable way. It is quite common to combine CI and CD into a CI/CD pipeline. This ensures any commit by a contributor is tested, built and ready to be deployed to production. Usually, deployment to production has to be performed manually. [5]

GOCD is an open source continuous integration and continuous delivery platform. It is split into GOCD server and agent. 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.

Comparing with other DevOps tools

Comparing with Azure DevOps tools:

GOCD is an open source continuous delivery tool allowing for advanced workflow modeling and dependencies management which is closely related to DevOps. Azure DevOps belongs to the "Project Management" category of the tech, while GOCD can be primarily classified under "Continuous Integration". "Complete and powerful" is the primary reason why developers consider Azure DevOps over the competitors, whereas "Open source" was statednas the key factor in picking GoCD.[6]

Compared with Jenkins DevOps tools:

If we need continuous delivery, we should be using GoCD, a solid Jenkins alternative. GoCD suits both beginners and those highly experienced in building deployment pipelines.

Topic	GoCD	Jenkins
Continuous Delivery	It was built on the principles of Continuous Delivery. This is visible in its abstractions as pipeline is a first class concept. GoCD also encourages that there be only one way to implement the fundamental CD patterns. When you search for help on how to implement the various deployment pipeline patterns, you will generally find a single, well-known, well-tested answer.[7]	With Jenkins 2.0, CD is implemented by the installation of a variety of plugins. Many common CD patterns(build an artifact only only once, full traceability up and down stream, and more) Are either impossible to implement or can only be cobbled together with fragile combinations of plugins.[7]
Continuous Integration	Although GoCD is built specifically with CD in mind, it has sophisticated features for continuous integration.[7]	Jenkins is built for CI. Anything beyond that, requires plugins.[7]

FireBase

FireBase is a platform which enables the developer to deploy their mobile and web application. In this tutorial, the demo github repository will be hosted on Firebase platform after GoCD automatically tests and deploys it.

Repository we used to test in the following tutorial:

https://github.com/YonatanKra/web-components-ui-elements/tree/before_ci_cd

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

Part 1: Installation and configuration of first pipeline:

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.

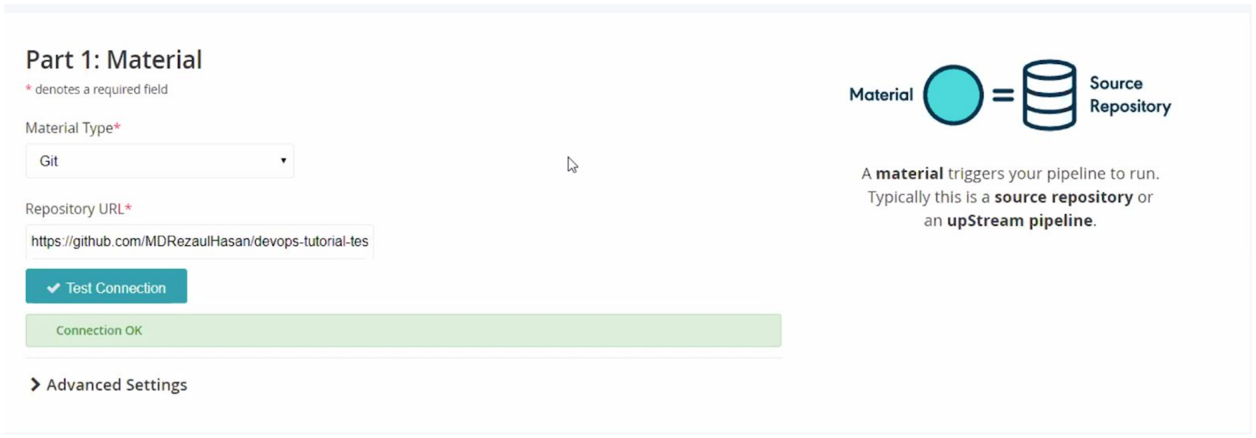
Installing GOCD client

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).

The following steps are used to create a new pipeline on the server.

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

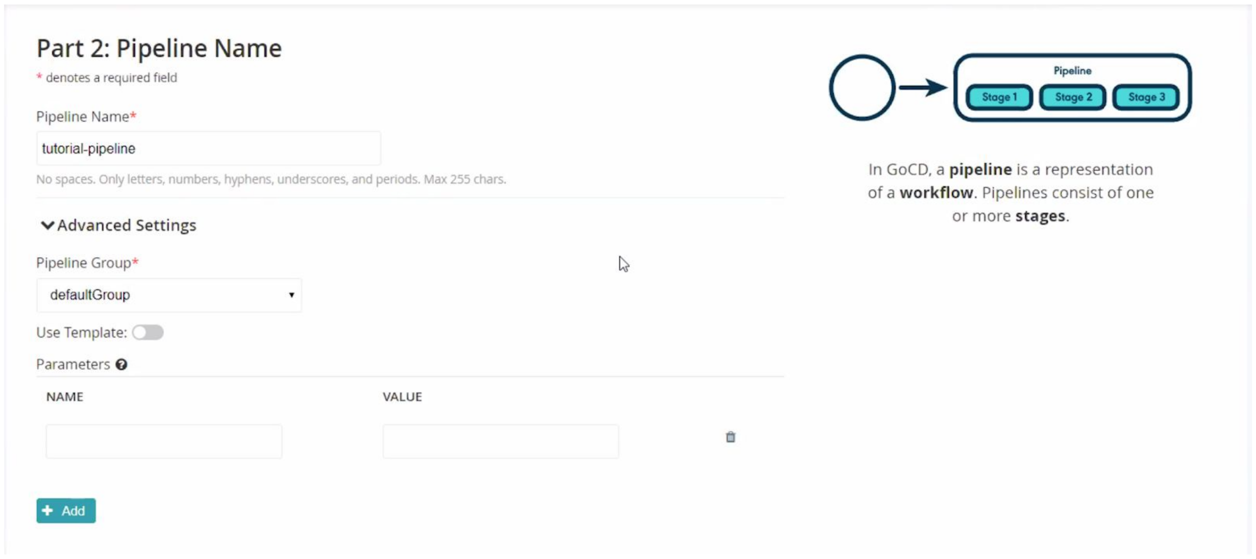
Repository URL*

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

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*

tutorial-pipeline

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

Advanced Settings

Pipeline Group*

defaultGroup

Use Template:

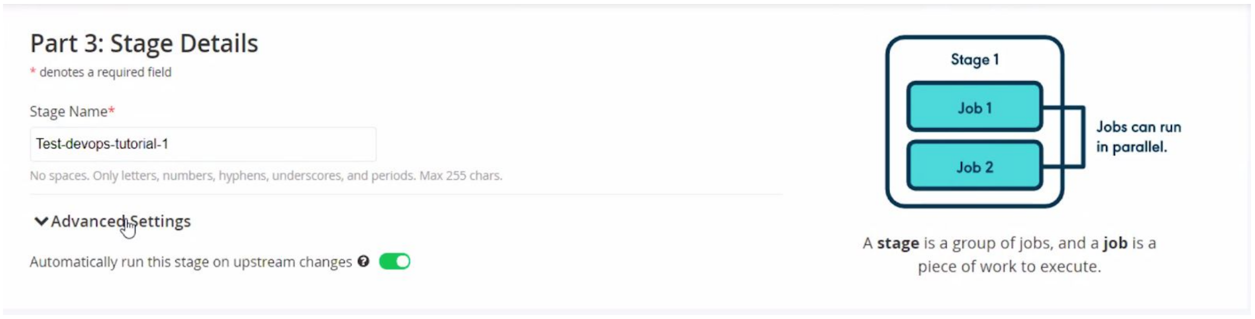
Parameters

NAME	VALUE

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 failed, this stage will fail but it won’t influence the execution of other jobs.


Part 3: Stage Details

* denotes a required field

Stage Name*

Test-devops-tutorial-1

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

Advanced Settings

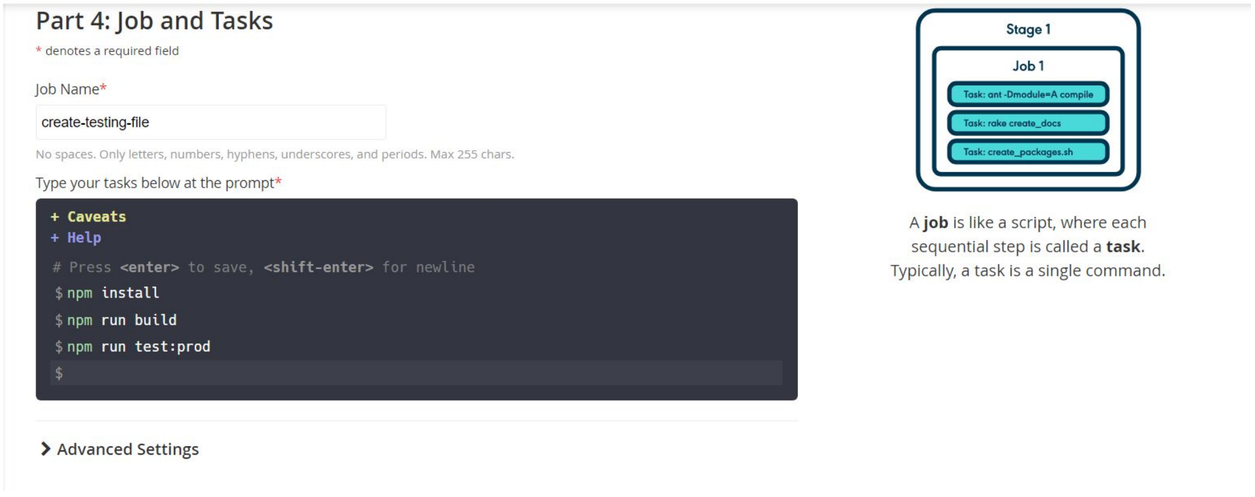
Automatically run this stage on upstream changes: on

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

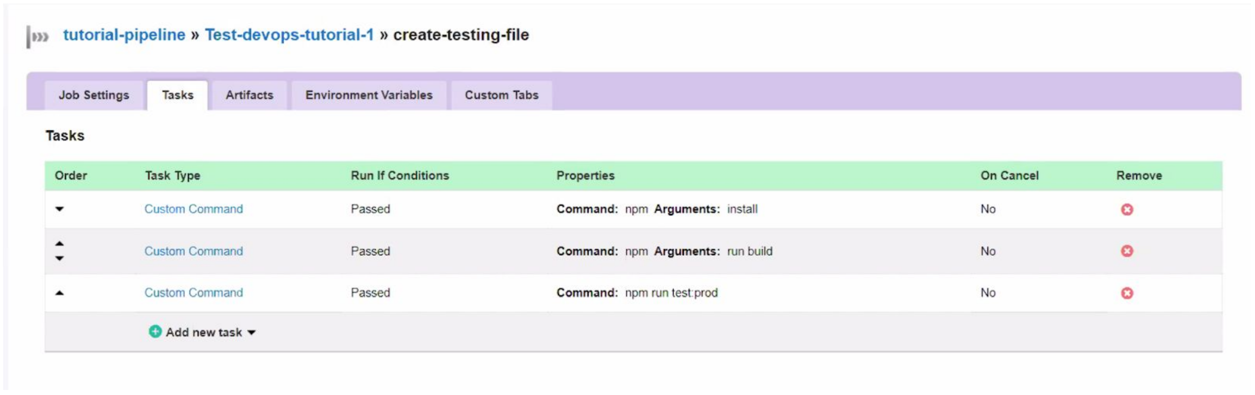
No spaces. Only letters, numbers, hyphens, underscores, and periods. Max 255 chars.

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

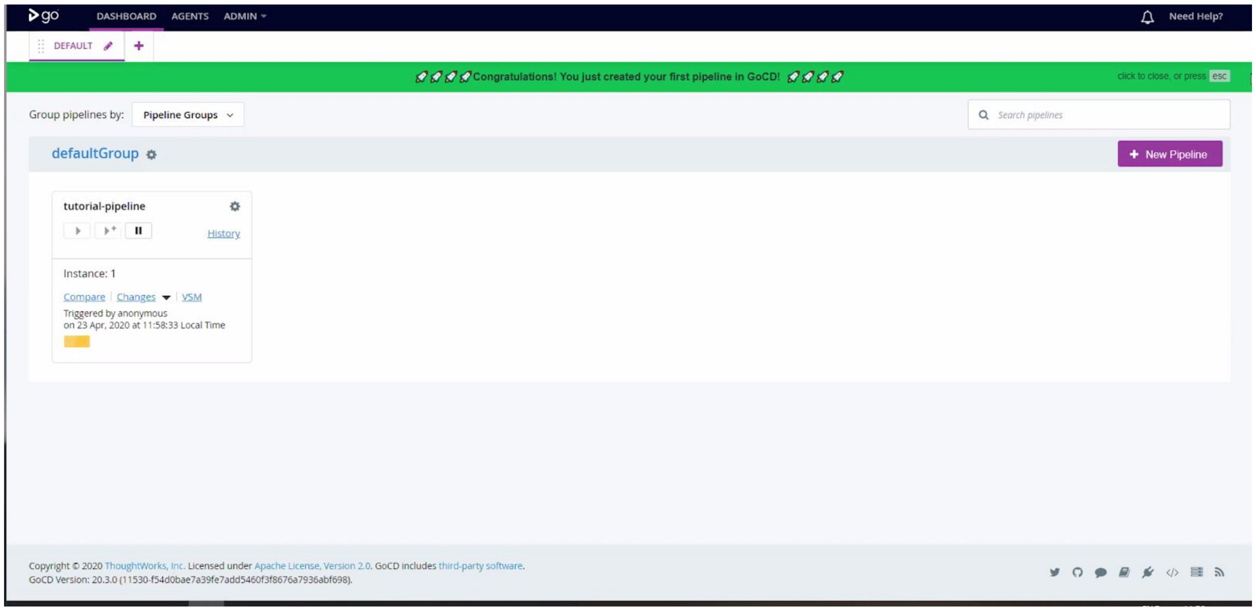
A **job** is like a script, where each sequential step is called a **task**. Typically, a task is a single command.


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

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	✖

Here is the page after you successfully configure one pipeline.



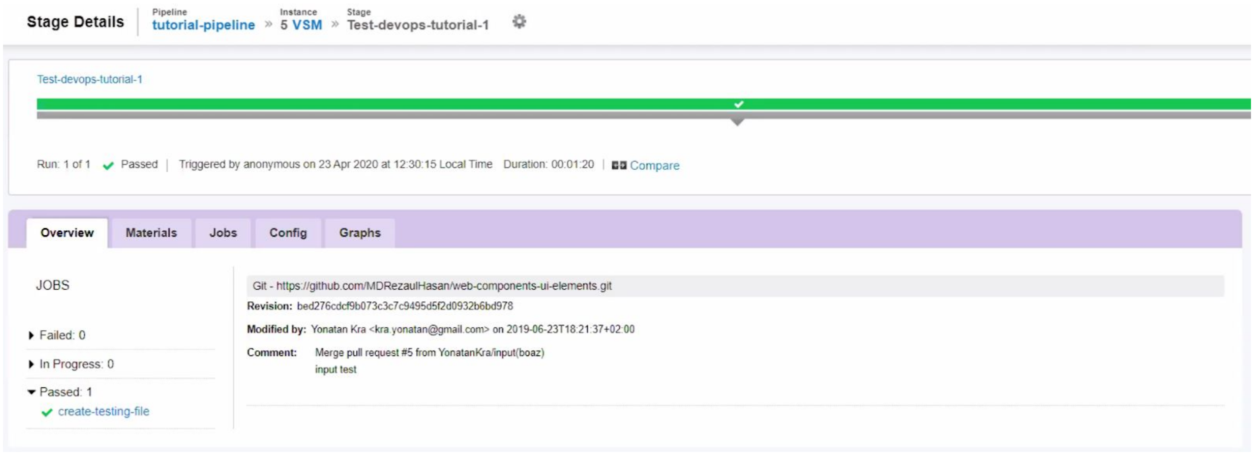
Congratulations! You just created your first pipeline in GoCD!

Instance: 1

Triggered by anonymous
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).

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.



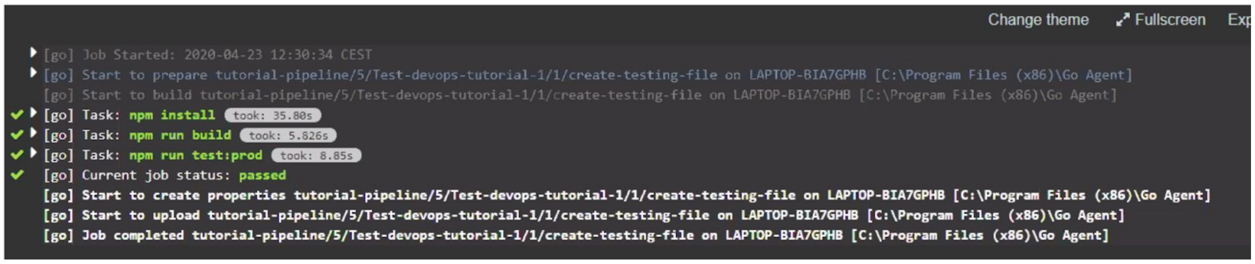
Run: 1 of 1 ✓ Passed | Triggered by anonymous on 23 Apr 2020 at 12:30:15 Local Time Duration: 00:01:20

Git - https://github.com/MDRezaulHasan/web-components-ui-elements.git
Revision: bed276cdcf9b073c3c7c9495d5f2d0932b6bd978
Modified by: Yonatan Kra <kra.yonatan@gmail.com> on 2019-06-23T18:21:37+02:00
Comment: Merge pull request #5 from YonatanKra/input(boaz)
input test

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

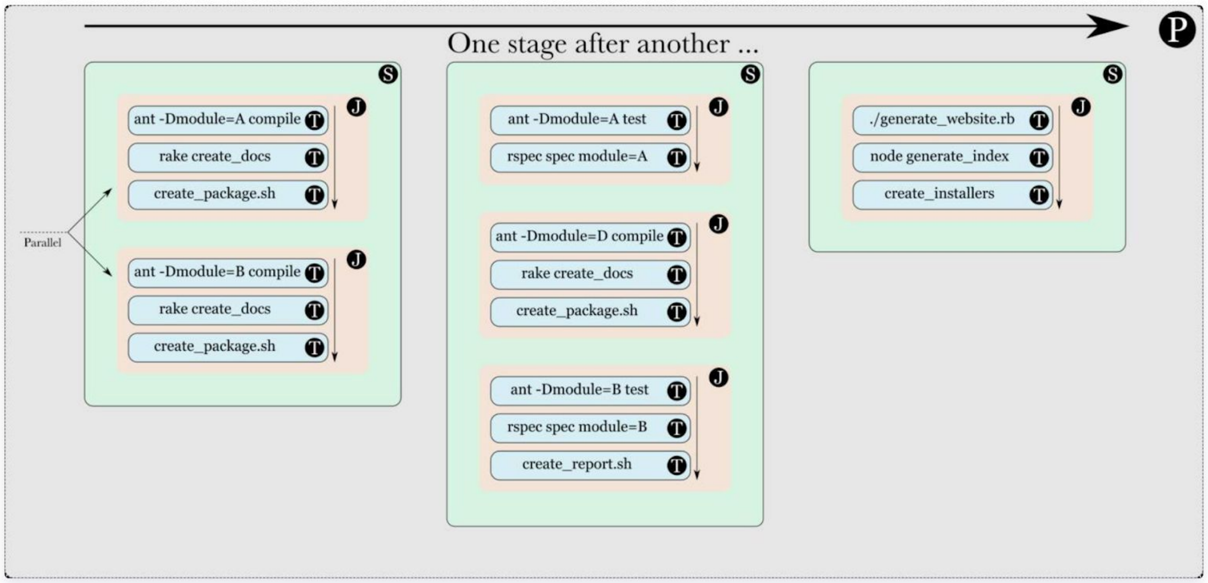
```
npm install
npm run build
npm run test:prod
```

The result is that all the tests are passed



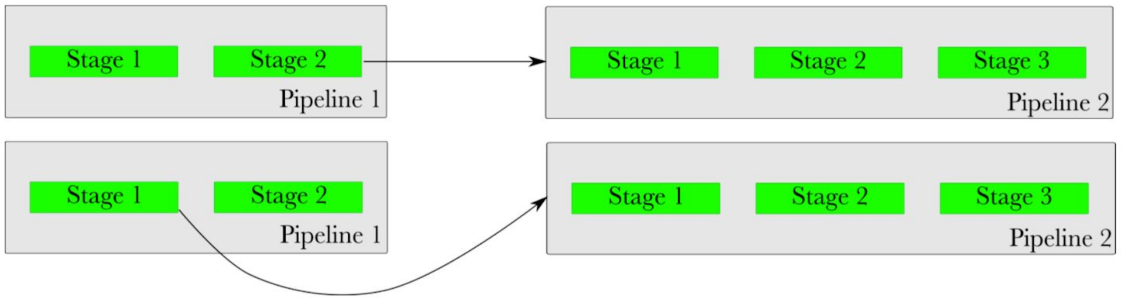
```
[go] Job Started: 2020-04-23 12:30:34 CEST
[go] Start to prepare tutorial-pipeline/5/Test-devops-tutorial-1/1/create-testing-file on LAPTOP-BIA7GPHB [C:\Program Files (x86)\Go Agent]
[go] Start to build tutorial-pipeline/5/Test-devops-tutorial-1/1/create-testing-file on LAPTOP-BIA7GPHB [C:\Program Files (x86)\Go Agent]
[go] Task: npm install  took: 35.80s
[go] Task: npm run build  took: 5.826s
[go] Task: npm run test:prod  took: 8.85s
[go] Current job status: passed
[go] Start to create properties tutorial-pipeline/5/Test-devops-tutorial-1/1/create-testing-file on LAPTOP-BIA7GPHB [C:\Program Files (x86)\Go Agent]
[go] Start to upload tutorial-pipeline/5/Test-devops-tutorial-1/1/create-testing-file on LAPTOP-BIA7GPHB [C:\Program Files (x86)\Go Agent]
[go] Job completed tutorial-pipeline/5/Test-devops-tutorial-1/1/create-testing-file on LAPTOP-BIA7GPHB [C:\Program Files (x86)\Go Agent]
```

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 2: 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.



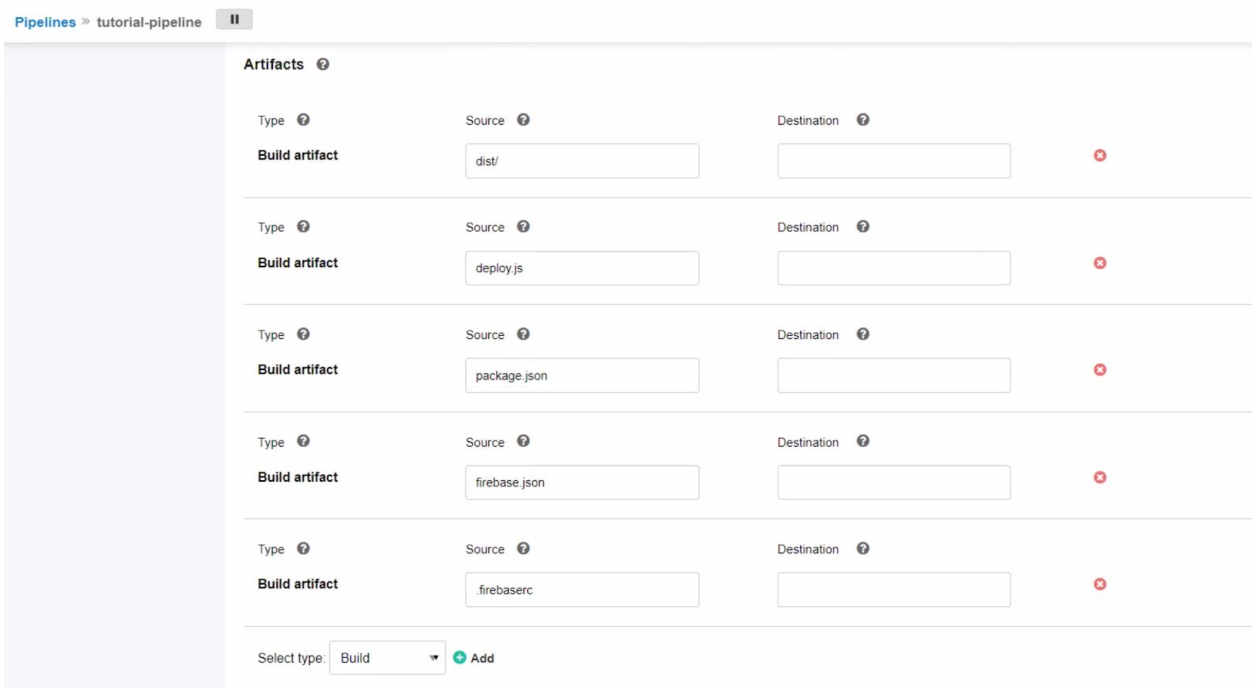
Pic 1: The relationship between the pipelines

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

Artifacts

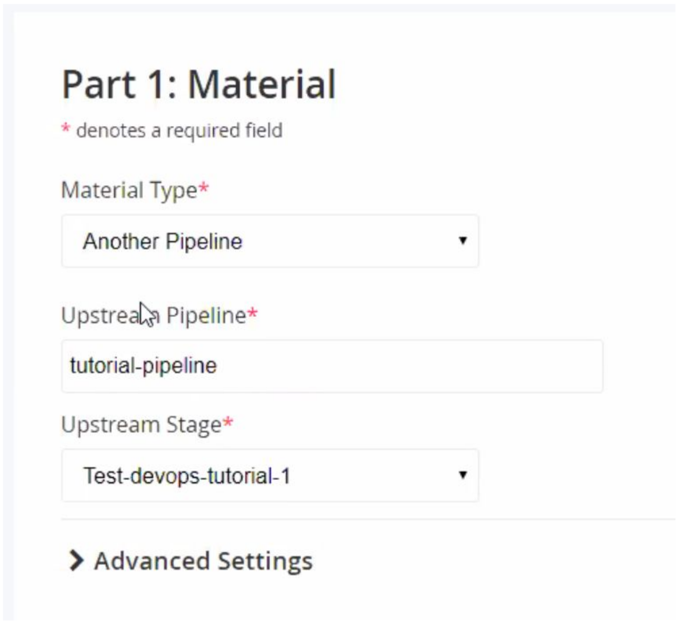
Type	Source	Destination
Build artifact	dist/	
Build artifact	deploy.js	
Build artifact	package.json	
Build artifact	firebase.json	
Build artifact	.firebaserc	

Select type: Build

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

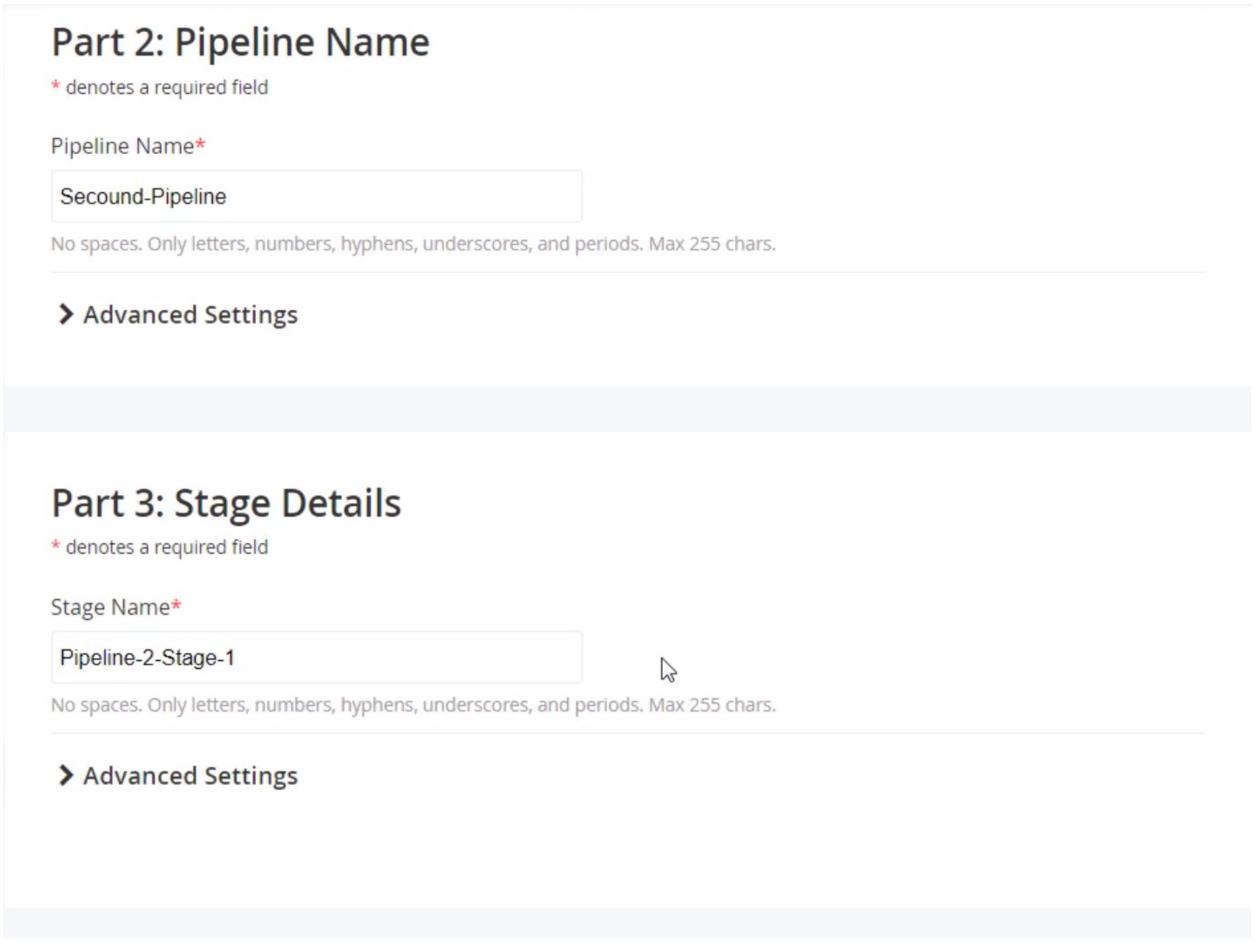
Upstream Stage*

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

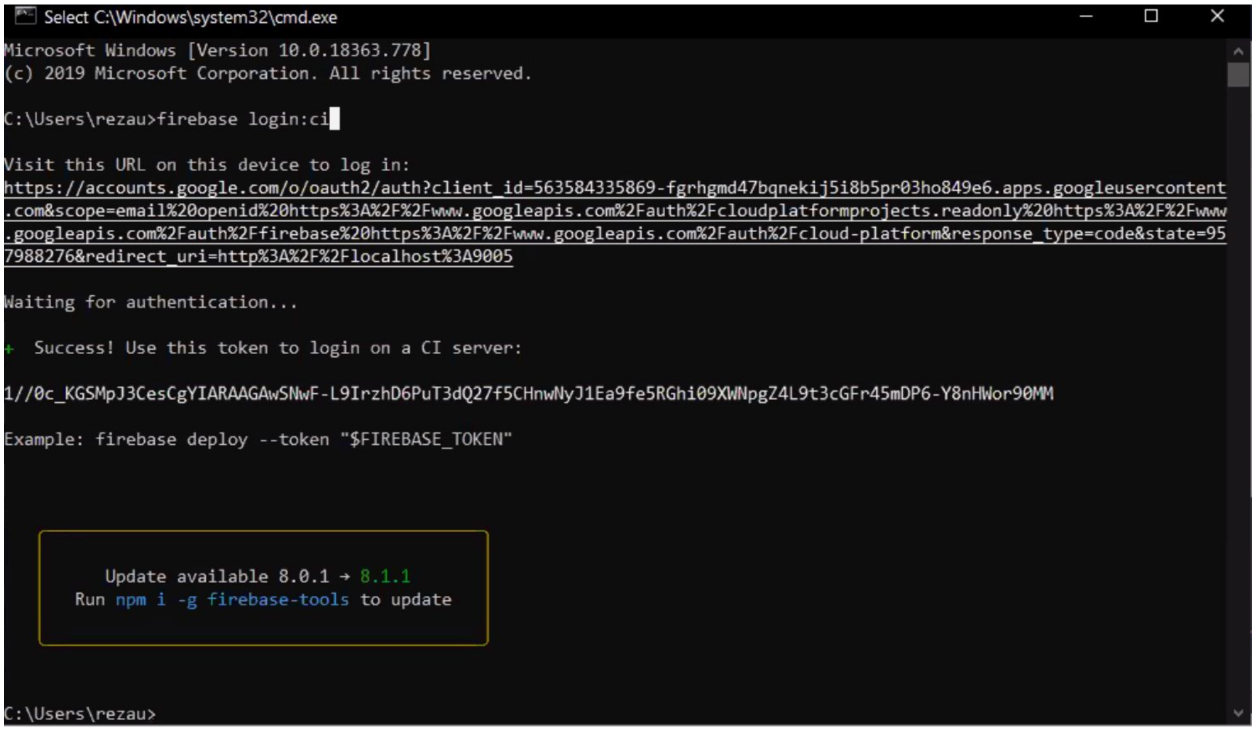
Step 4 Generate Firebase token

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-fgrhgmd47bqnekij5i8b5pr03ho849e6.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=95
7988276&redirect_uri=http%3A%2F%2Flocalhost%3A9005

Waiting for authentication...

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

1//0c_KGSMpJ3CesCgYIARAAGAwSNwF-L9IrzhD6PuT3dQ27f5CHnwNyJ1Ea9fe5RGhi09XWNpgZ4L9t3cGFr45mDP6-Y8nHWor90MM

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.

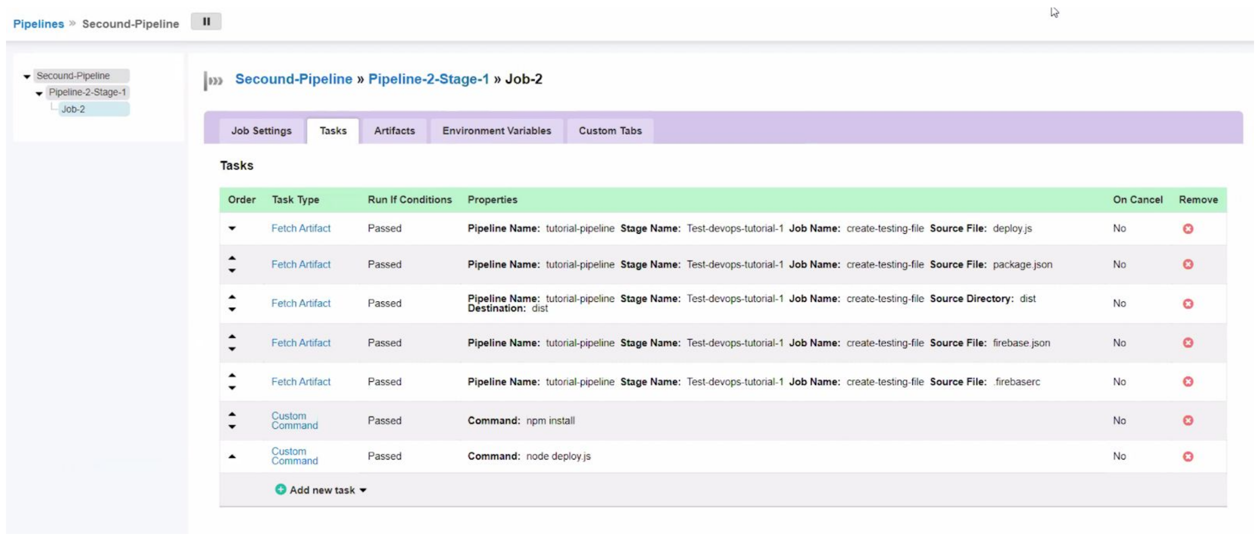
```
var client = require("firebase-tools");
client
 .deploy({
   project: "devopstutorial-d9959",
   token:

"1//0c_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, ".firebaserc" file need to be changed to match the name of the project created on firebase.

```
{
   "projects": {
     "default": "devopstutorial-d9959"
   }
 }
```

Step 6 Fetch artifact in Downstream pipeline :

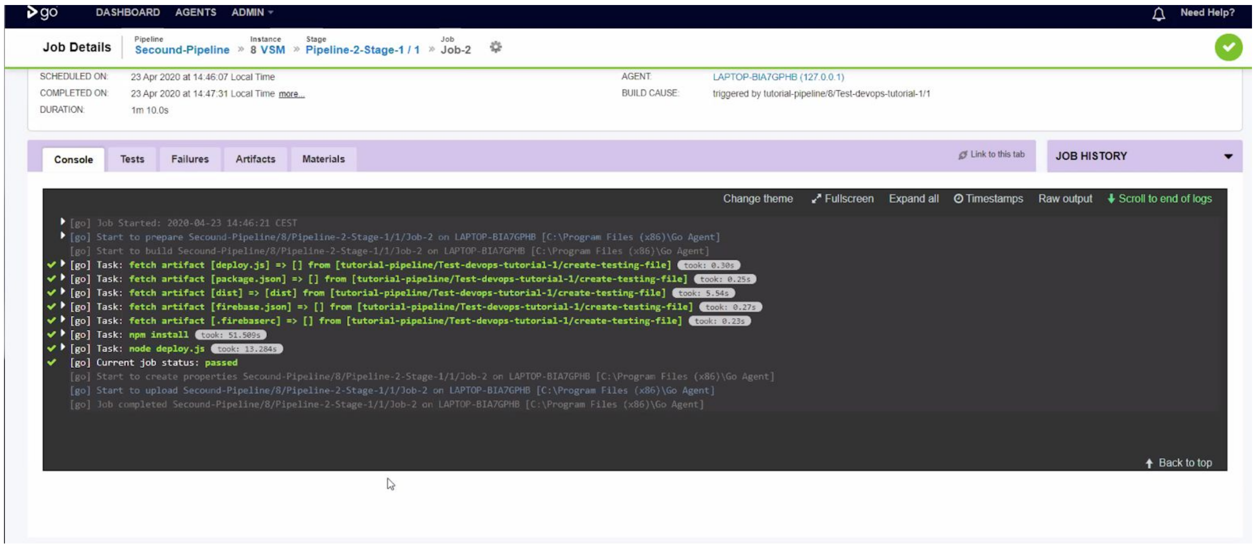
In order to use the output file after running the pipeline, the user has to create several new tasks of which type is fetch artifact. Those tasks can obtain all the files of the project and installed package by the first pipeline.



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 **Destination:** 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:** .firebaserc	No	✖
▲▼	Custom Command	Passed	**Command:** npm install	No	✖
▲	Custom Command	Passed	**Command:** node deploy.js	No	✖

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

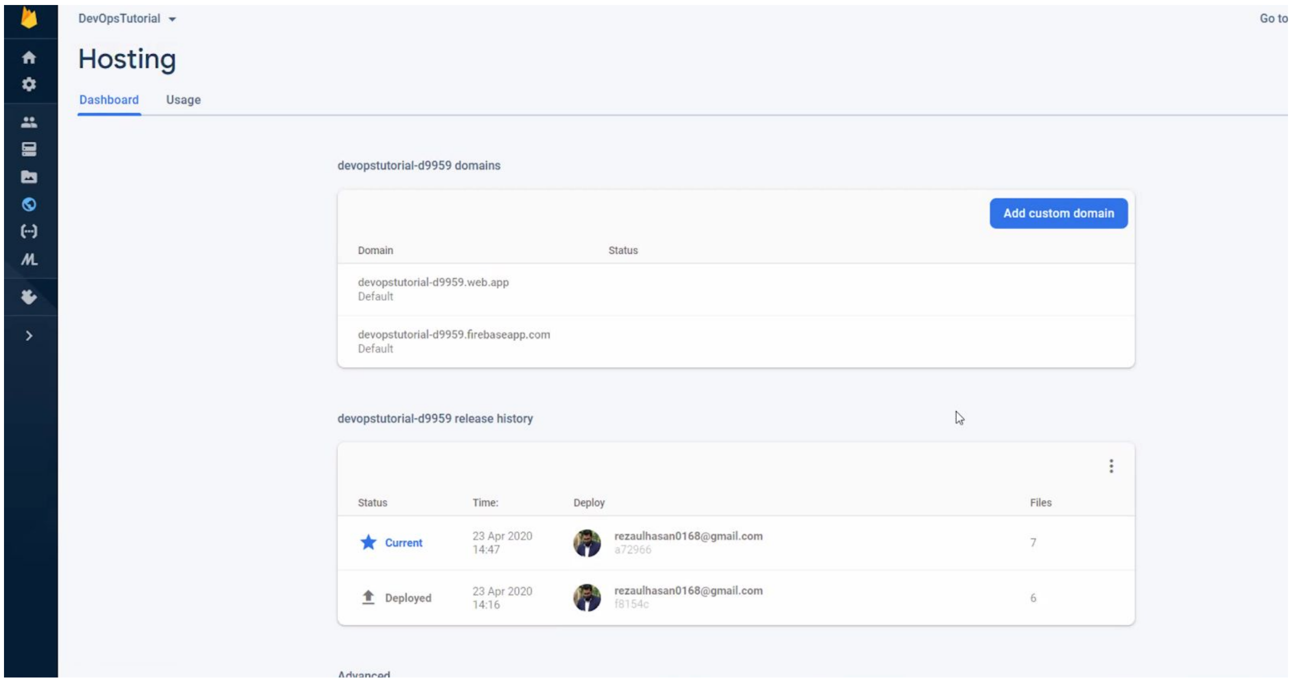


Pipeline: Secound-Pipeline » Instance: 8 VSM » Stage: Pipeline-2-Stage-1 / 1 » Job: Job-2

SCHEDULED ON:	23 Apr 2020 at 14:46:07 Local Time	AGENT:	LAPTOP-BIA7GPHB (127.0.0.1)
COMPLETED ON:	23 Apr 2020 at 14:47:31 Local Time more...	BUILD CAUSE:	triggered by tutorial-pipeline/8/Test-devops-tutorial-1/1
DURATION:	1m 10.0s		

```
[go] Job Started: 2020-04-23 14:46:21 CEST
[go] Start to prepare Secound-Pipeline/8/Pipeline-2-Stage-1/1/Job-2 on LAPTOP-BIA7GPHB [C:\Program Files (x86)\Go Agent]
[go] Start to build Secound-Pipeline/8/Pipeline-2-Stage-1/1/Job-2 on LAPTOP-BIA7GPHB [C:\Program Files (x86)\Go Agent]
✓ [go] Task: fetch artifact [deploy.js] => [] from [tutorial-pipeline/Test-devops-tutorial-1/create-testing-file] took: 0.30s
✓ [go] Task: fetch artifact [package.json] => [] from [tutorial-pipeline/Test-devops-tutorial-1/create-testing-file] took: 0.25s
✓ [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: 0.27s
✓ [go] Task: fetch artifact [.firebaserc] => [] from [tutorial-pipeline/Test-devops-tutorial-1/create-testing-file] took: 0.23s
✓ [go] Task: npm install took: 51.509s
✓ [go] Task: node deploy.js took: 13.284s
✓ [go] Current job status: passed
[go] Start to create properties Secound-Pipeline/8/Pipeline-2-Stage-1/1/Job-2 on LAPTOP-BIA7GPHB [C:\Program Files (x86)\Go Agent]
[go] Start to upload Secound-Pipeline/8/Pipeline-2-Stage-1/1/Job-2 on LAPTOP-BIA7GPHB [C:\Program Files (x86)\Go Agent]
[go] Job completed Secound-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!



devopstutorial-d9959 domains

Domain	Status
devopstutorial-d9959.web.app Default	
devopstutorial-d9959.firebaseapp.com Default	

devopstutorial-d9959 release history

Status	Time:	Deploy	Files
Current	23 Apr 2020 14:47	rezaulhasan0168@gmail.com a72966	7
Deployed	23 Apr 2020 14:16	rezaulhasan0168@gmail.com f8154c	6

References:

1. https://en.wikipedia.org/wiki/DevOps
2. https://en.wikipedia.org/wiki/Test_automation
3. https://en.wikipedia.org/wiki/Build_automation
4. https://codeship.com/continuous-integration-essentials
5. https://continuousdelivery.com/
6. https://stackshare.io/stackups/azure-devops-vs-go-cd
7. https://www.gocd.org/jenkins/