Integration testing tutorial with Bitbucket Pipelines

Learn how to run integration tests with Bitbucket Pipelines by having multiple services running in separate Docker containers in a pipeline.

BY STEN PITTET

Testing is a critical part of continuous integration and continuous delivery. And if you're practicing continuous deployment it will be the last line of defense against bugs before changes get released to your customers. Unit tests validating individual methods and classes are a great start to prevent issues, but you will also need to run integration tests that make sure that the different modules used by your application (application server, database, cache) interact properly together. We will see in this tutorial how you can run integration tests with Bitbucket Pipelines by having multiple services running in separate Docker containers in a pipeline.

Requirements

I

Step 1: Running the sample application locally

We will use a basic Node.js application that displays a message on the homepage and logs visits to a database. To focus on the Bitbucket Pipelines configuration, you can simply clone the application from your terminal.

```
git clone git@bitbucket.org:spittet/bitbucket-pipelines-
```

Now go to your local repository and run npm install to install the dependencies required by the application.

```
cd bitbucket-pipelines-services-tutorial/ npm install
```

Before running the application, we will need to start a new MongoDB instance. Thanks to Docker this is something that you can easily do from your terminal.

```
docker run --name mongodb -d -p 27017:27017 mongo
```

Then start your application and go to http://localhost:3000 to see it in action.

```
npm start
```

```
Hello World!
```

You can go to http://localhost:3000/visits to make sure that a visit has been properly logged into the database.

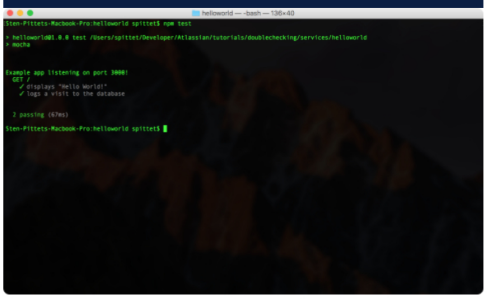
```
// 20170420151601
// http://localhost:3000/visits

[
  {
    "_id": "58f8c2558b2c5c1123d8a254",
    "visited_at": "2017-04-20T14:14:45.769Z",
    "__v": 0
  }
]
```

Finally, let's make sure that the tests are completing successfully locally via the command npm test. You should

see two tests completing after running that command.

```
npm test
```



This sample application is running two different tests:

- One test that verifies that the application displays Hello World in the homepage.
- One test that verifies that a new visit is logged in the database whenever someone accesses the homepage.

The first test will pass even if the database is down but the second test is an integration test that verifies that the web application interacts properly with the database server. It could also be understood as a functional test since it verifies some of the business requirements of the application. You can learn more about the different types of tests in our guide.

We will now see how you can use Bitbucket Pipelines to automate the testing of your application and configure it to be successful with a database.

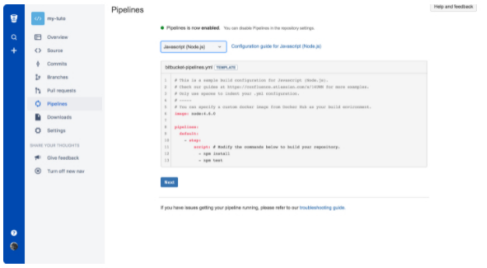
Step 2: Running tests automatically with Bitbucket Pipelines

Start by creating a new repository in your Bitbucket account and update the remote URL for origin to point to your Bitbucket repository.

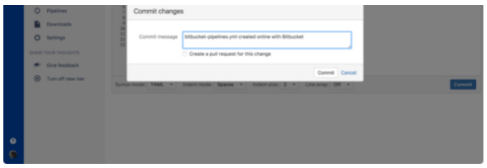
```
git remote set-url origin git@bitbucket.org:<my account>/
```

Go to the Pipelines section of your repository to enable Bitbucket Pipelines.

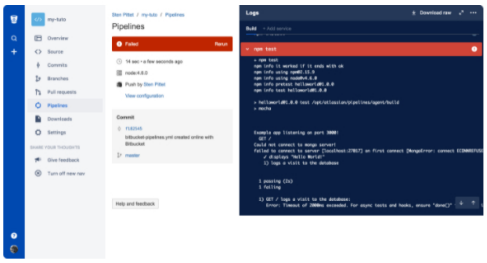
You can use the default Javascript template in the next screen. It will already have the npm install and npm test commands that you need to install dependencies and run the test suite.



Don't change anything in the YML configuration and commit your bitbucket-pipelines.yml file. The database service is missing, but we will add it later on.



Once you commit your file, you will be redirected to the Pipelines section of your repository where you can see your first pipeline in progress. Your pipeline will fail because the second test cannot run properly without a database connection. If you click through to your pipeline, you should see a screen similar to the one below where it says that 1 test passed and 1 test failed.



In the next section, we will fix that issue by adding a new service definition to your Pipelines configuration.

Step 3: Adding a service definition for the database

With Bitbucket Pipelines you can run up to 3 extra Docker containers on top of the main application running in a pipeline. You can use these containers to run services such as a datastore, analytic tool, or any 3rd party service that your application may need to complete the pipeline. In our case, we will use a separate service container to run MongoDB.

Go to the source section of your repository to see the list of files.

Click on the *bitbucket-pipelines.yml* configuration file to access it. You will find an Edit button in the top right corner that will let you edit the file and commit straight from your browser.

We need to add a service definition for our database at the bottom of the configuration file.

bitbucket-pipelines.yml

```
# This is a sample build configuration for Javascript (No
```

In the case of MongoDB, we don't need any extra settings in the image definition, but some Docker images for