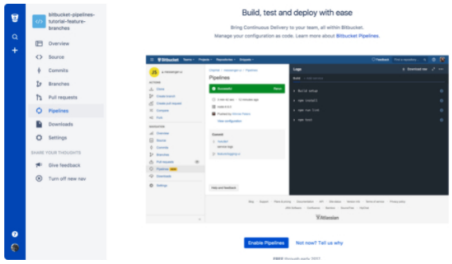
Running feature branches with Bitbucket Pipelines

Learn how to do continuous delivery with a Gitflow or feature branch workflow.

BY STEN PITTET

Whether you're using Gitflow or simply feature branches with a main branch, you can easily adopt continuous delivery (CD) with Bitbucket Pipelines. No need for you to configure a complex continuous integration (CI) server, you'll only need to enable Pipelines and define your workflows to be able to run tests and deployments on your branches.



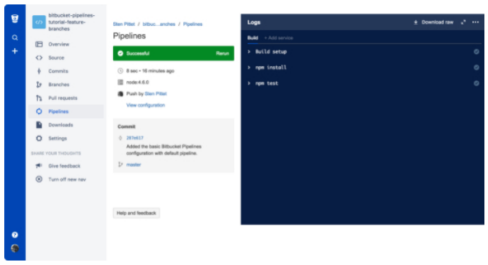
We will see in this tutorial how you can simply configure Bitbucket Pipelines to run different pipelines for different branches, as well as how you can protect your branches from bad merges.

Step 1: Start with a default pipelines to be run on feature branches

Using feature branches is a great way to prevent main from being often broken. Developers can work on a specific improvement on a separate branch and merge their changes when their build is green. However, this situation does not mean that it is less important to keep your feature branches stable. To enable great collaboration, it is just as important to keep feature branches in a green state. Using feature branches is a mean to make it easier to understand what changes have been made to solve a specific issue, it should not be taken as an opportunity to delay quality.

So the first thing we will want to do when enabling Bitbucket Pipelines is to create a default pipeline that will run tests for every branch. This is easily done by picking one of the default templates available.

All the language specific templates are using a default pipeline under the default keyword that will get executed for every new branch pushed. You can simply commit the bitbucket-pipelines.yml configuration file to your repository to get your first pipeline executed on the main branch.



RELATED TUTORIAL

Tips for scripting tasks with Bitbucket Pipelines

Try this tutorial →

SUBSCRIBE

Sign up for more articles

Email

email@example.com

Subscribe

You can just push a new branch with changes on it to verify that the same pipeline gets executed on a different branch.

Step 2: Add a new pipeline for the main branch

If you're practicing continuous delivery, then you will most likely want changes pushed to main to be deployed automatically to a staging environment. To achieve that we will add a new branch pipeline that deploys the code after running tests, and only gets executed for main.

```
image: node:4.6.0
  pipelines:
  default:
    - step:
        script:
          - npm install
          - npm test
  branches:
    main:
      - step:
          script:
            - npm install
            - npm test
            - ./deploy.sh
```

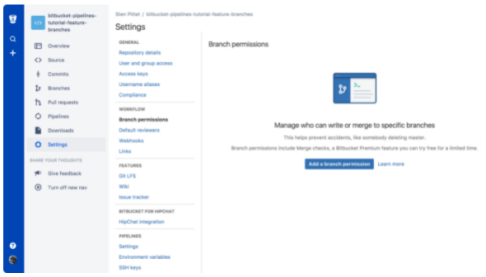
The *branches* section in the YML configuration above is where we define the pipeline that we want to execute when changes are pushed to main.

From now on, a push to main will trigger a deployment after having built and tested the application. Any other branch of the repository will simply build and test the new changes.

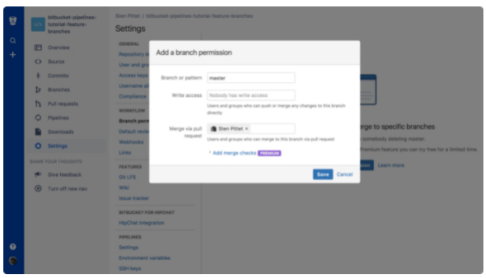
Step 3: Protect your release branches

After the completion of step 2, any developer can trigger a release to production by simply merging their changes to main. This is a risky situation to be in because someone could deploy changes that have not yet been reviewed by mistake. Thankfully you can easily prevent that with from happening by adding permissions to your branches in Bitbucket.

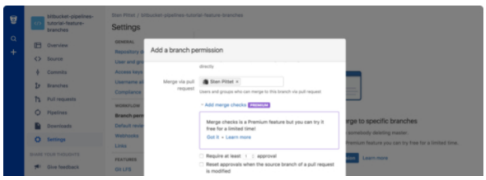
Go to Settings > Branch permissions in your repository.



Add a new branch permission for main where you leave the *Write* access blank to prevent developers from pushing straight to main. Then add yourself to the *Merge via pull request permission*.



Before saving the new branch permission, we will add a merge check to make sure that merges are not allowed unless there's at least one green build. Just expand the merge checks section to enable the corresponding feature.



After saving you can verify that the branch is properly protected. No user or group should have write access and merge via pull request should be allowed for your trusted team members.

Step 4: Use pull request to promote changes to production

Since you can't push straight to main anymore, you will need to use pull requests to deploy to production. Once the pull request is created, you simply need to merge the changes to main to trigger the deployment pipeline.

After the merge, you can go to the Pipelines section of your repository to see the deployment in action.