Deploying a React App on AWS With Gitlab CI

Rodolfo Costa Follow
Jul 20, 2020 · 5 min read ★

Gitlab is a web tool that provides Git repository hosting service along with a lot of services related to DevOps such as continuos integration/continuous deployment pipeline features. This gives developers the possibility of automating the entire DevOps life cycle from build to deployment in the same environment where source code is hosted.

In this article we are focusing on the creation of a file called **".gitlab-ci.yml"**, which is responsible for implementing both Continuous Integration and Continuous Deployment functionalities for our project. It creates a pipeline that consists of one or more stages

that run sequentially and each stage can contain one or more jobs/scripts that can run in parallel.

In order to run ".gitlab-ci.yml", it must be placed at the **repository's root** and a **Gitlab Runner** must be installed to execute it. For Gitlab Runner installation, see http://docs.gitlab.com/ee/ci.

> *So, before start scripting the file, first it is important to plan what stages and jobs are needed to achieve desired results to make it as resource efficient as possible.*

Defining stages

Our source code is a front-end React.js based application. This means that:
1) lt generates static files for deployment.
2) It can be deployed to any hosting that supports static file hosting. In our case we, chose Amazon S3, because it is the easiest option for a front-end deployment under AWS ecosystem.

Besides, we want to have two environments being one for testing purposes and other for clients.

In the face of these conditions, we can divide our **".gitlab-ci.yml"** in two stages: **build** and **deploy**.

Build stage

In this stage, we generate our application static files. If there are no build commands defined inside **"package.json"** file under **"scripts"** property, simply use "npm run build", "yarn build" or a command according to your package manager of preference.

The build command will generate a **"/build"** folder containing all of our required static files. We need this folder to be available in the "deploy" stage since its files will be put in S3 bucket.

Here follows the code for the "build" stage:

```
.yarn_build:
  image: node:10
  script: |
```

```
    yarn # Install all dependencies
    yarn build:${APP_ENV} # Build command
  artifacts:
    paths:
      - ./build

yarn_dev:
  extends: .yarn_build
  stage: build
  before_script:
    - export APP_ENV="dev"
  only:
    refs:
      - develop

yarn_prod:
  extends: .yarn_build
  stage: build
  before_script:
    - export APP_ENV="prod"
  only:
    refs:
      - master
```

In the code above, we define 2 jobs “yarn_dev” and “yarn_prod” that represents our environments for the application. Both extend from “yarn_build” since they define the variable that sets the environment for the build command. We are capable of using **yarn** for our build command by using “node:10” image (from Docker Hub), which comes with yarn pre-installed. For more information about available node images: https://hub.docker.com/_/node/.

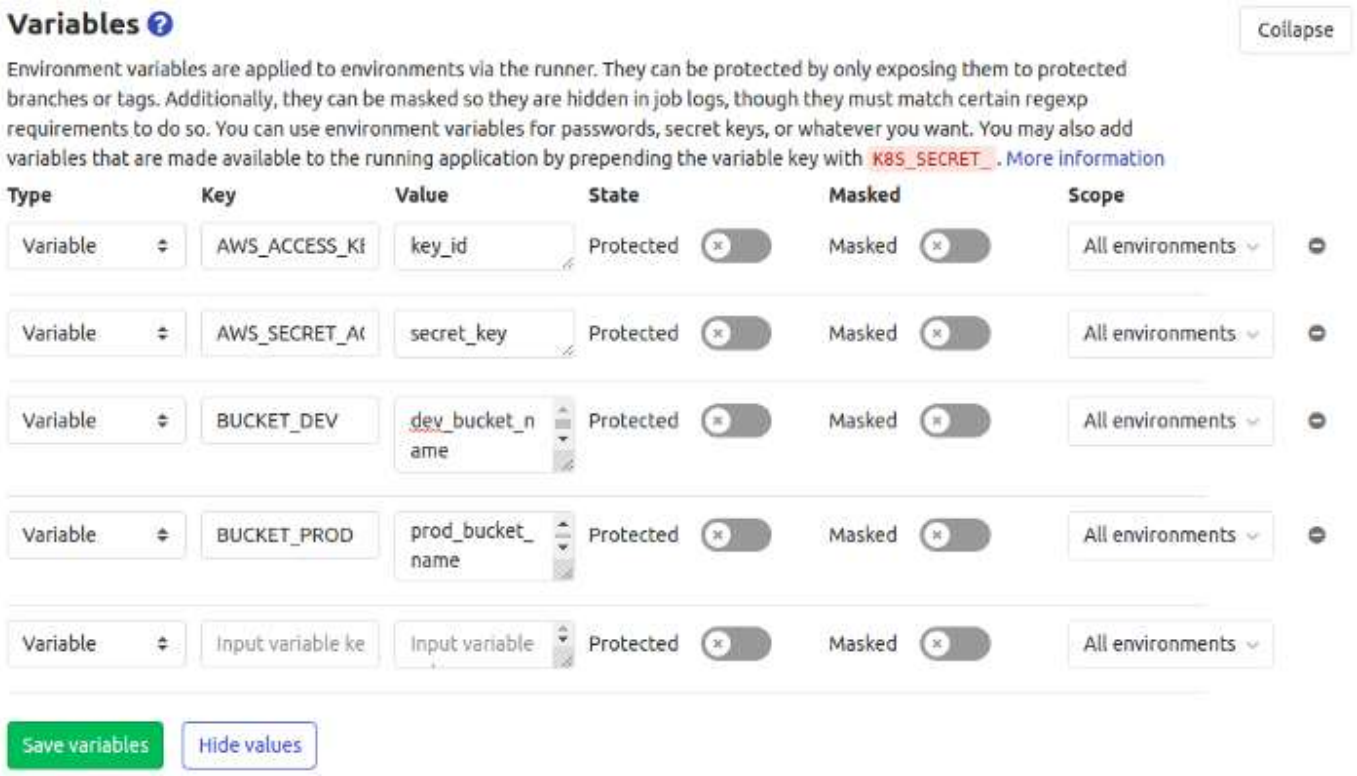
Each job will run depending on the branch the commit was made. A commit in “develop” will trigger “yarn_dev” job and a commit in “master” will trigger “yarn_prod” job. Other branches won’t trigger a pipeline since we want to save resources as much as possible. The “only-refs” configuration gives possibility of multiple branches triggering a pipeline for an environment in case it becomes necessary.

At the end, we set “/build” folder as an artifact of our build stage for it to be available and accessible in the next stage.

Deploy stage

In this stage, we send all built static files into our S3 bucket of choice. Given that we have two environments for our application, we created one bucket for each

environment. The bucket names are set as environment variables inside our repository settings. To define those variables, just go to “Settings” => CI/CD => Variables section.



Gitlab CI Environment Variables

By defining variables here, they are accessible from “.gitlab-ci.yml” without the need of updating the file itself in case of a required update of values.

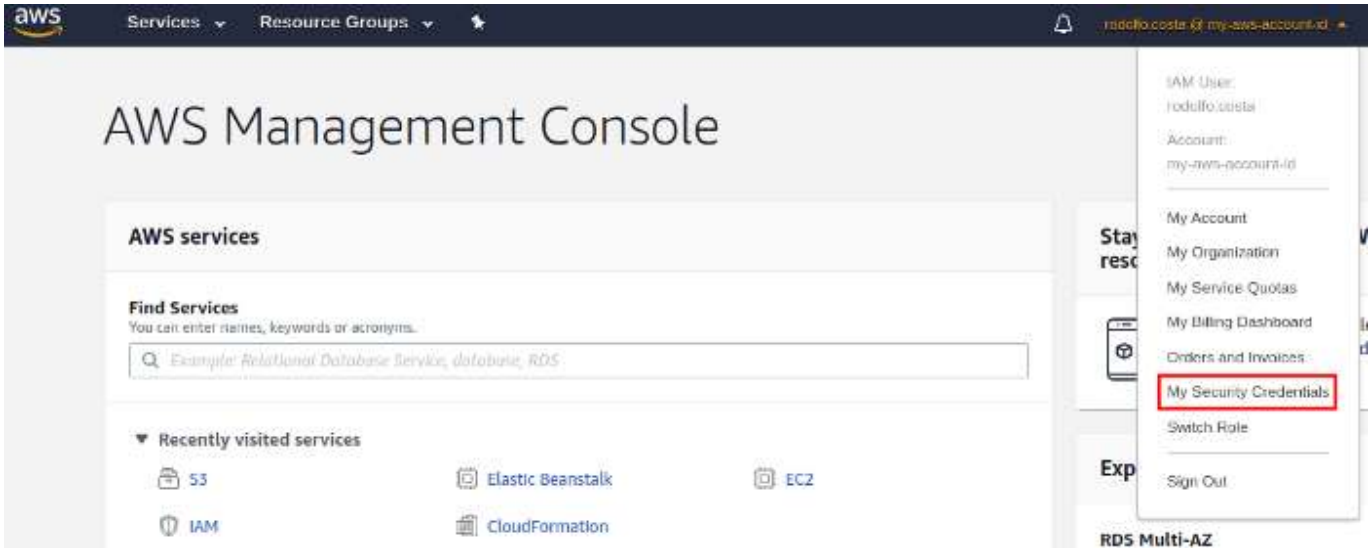
We defined both bucket names and required AWS credentials in order to run **“s3 sync”** as shown bellow:

```
.deploy_aws:
  image: python:latest
  when: manual
  script: |
    pip install awscli #Install awscli tools
    aws s3 sync ./build/ s3://${S3_BUCKET}

deploy_dev:
  extends: .deploy_aws
  stage: run
  dependencies:
    - yarn_dev
  before_script:
    - export S3_BUCKET=${S3_BUCKET_DEV}
```

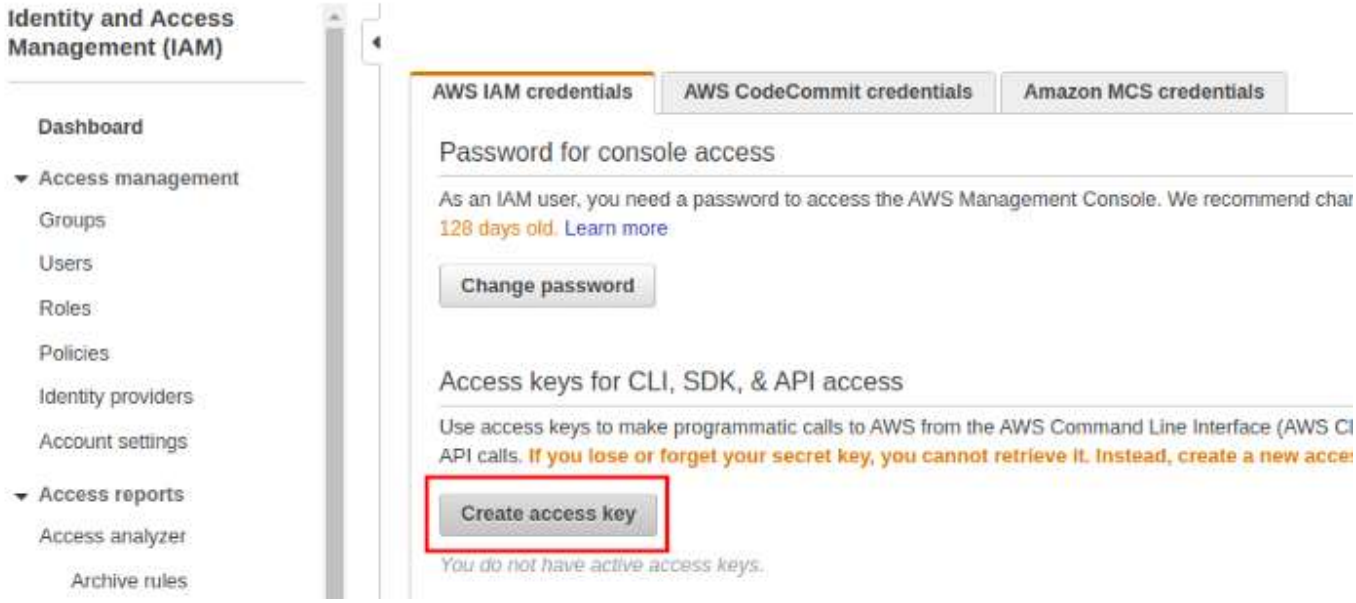
```
  only:
    refs:
      - develop

deploy_prod:
  extends: .deploy_aws
  stage: run
  dependencies:
    - yarn_prod
  before_script:
    - export S3_BUCKET=${S3_BUCKET_PROD}
  only:
    refs:
      - master
```

We use a python image since it has the most robust **awscli** implementation. The awscli is installed inside the image using python's "pip" package manager and, afterwards, a **"s3 sync"** command can be run to put our static files in the respective bucket. For this command to be successful, this bucket must be already created. For more details on Gitlab deployment, see https://about.gitlab.com/blog/2016/08/26/ci-deployment-and-environments/.

In addition, to run "s3 sync" command, we need both ***AWS_ACCESS_KEY*** and ***AWS_SECRET_ACCESS_KEY*** to be defined. They can be obtained when you're logged in your AWS account by going to: "Security Credentials" => "Create access key".



Security Credentials menu

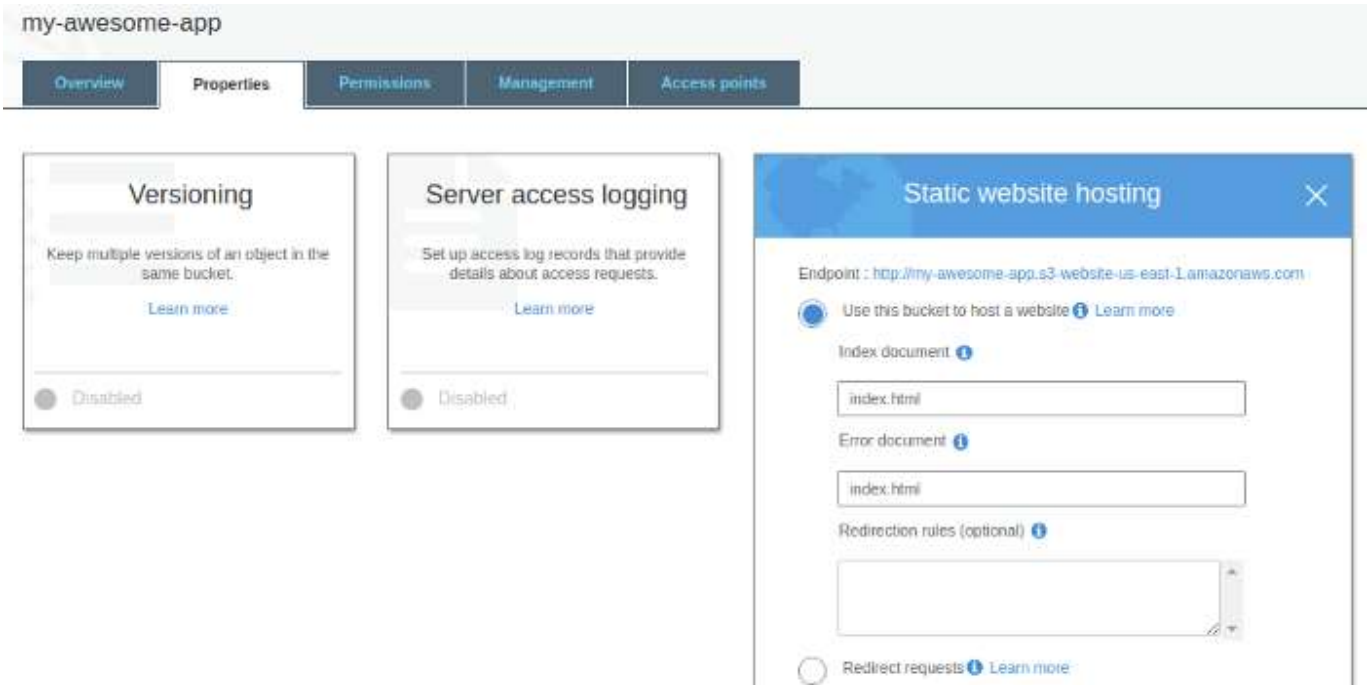


Create access key button

After clicking the button, a dialog with both values will appear. Once the dialog is closed, you won't have access to them again if you didn't choose to **download credentials** as a **.csv** file.

AWS Web Hosting

After running "**.gitlab-ci.yml**" successfully, all static files should be inside the bucket. But that doesn't mean they are already being served. To get that capability, first open your bucket. It opens with "Overview" tab selected, select "**Properties**" tab and some cards will appear. Choose "**Static website hosting**" as shown bellow.



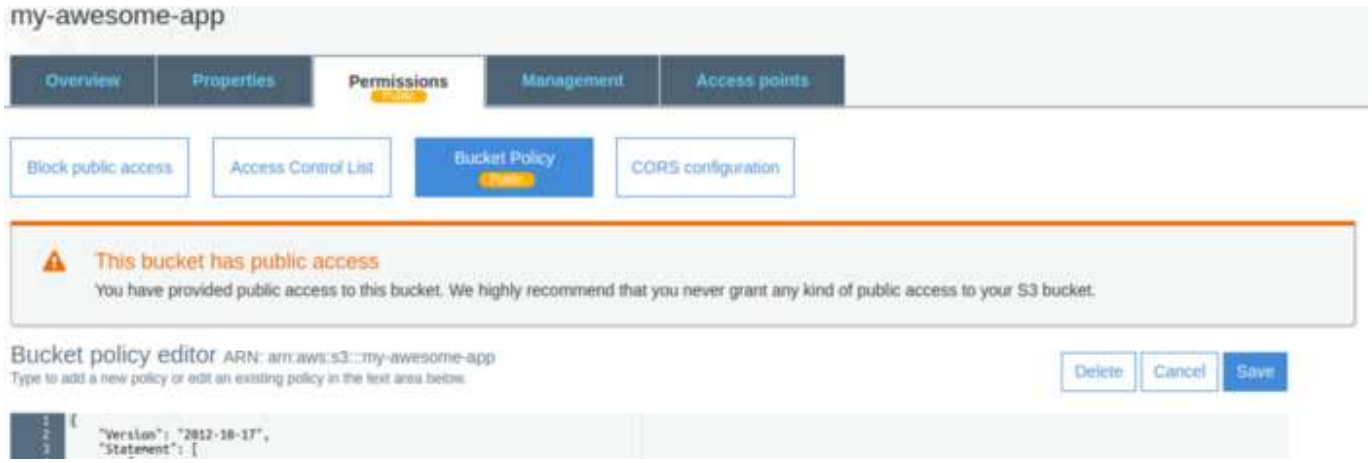
Static website hosting properties

With the card open, select “Use this bucket to host a website”, fill both “Index document” and “Error document” with **index.html** and save. After that, an endpoint will be created, but won’t be accessible yet.

To enable public access to your endpoint, go to “Permissions” tab and add the following bucket policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicReadGetObject",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::my-awesome-app/*"
            ]
        }
    ]
}
```

Inside **“Resource”** object, insert your bucket name and click on “Save” button. From now on, the “Permissions” tab will show a ***Public*** badge.



Bucket with policy added

That's awesome! Now we have our React app hosted and accessible from anywhere!

Conclusion

With all stages and variables defined, we can create a simple yet powerful script that automates the whole process of deployment of our application. From now on, we don't need to worry about how to proceed with deployment everytime we want to do it.

The script itself is easy to read and understand and is flexible enough to scale according to application demands and/or team strategies.

Here's the whole **".gitlab-ci.yml"** file:

Gitlab | Gitlab Ci | AWS | DevOps | React