FLINT.Cloud Technical Guide

Sneha Mishra, Arnav Tiwari & contributors

Dec 23, 2022

CONTENTS

1 FLINT.Cloud Development Setup **3**
- 1.1 Git and Github guide . . . 3
- 1.2 GCBM Local Development Guide . . . 6

2 FLINT.Cloud Deployment **7**
- 2.1 Google Cloud Platform . . . 7
- 2.2 Azure . . . 7

The project aims to build a continuous deployment pipeline to offer FLINT as a SaaS over cloud. The FLINT-GCBM scripts are converted to a REST Api service built using Flask. This API Service is designed to be consumed by Gcloud.

This Documentation is meant for Developers wishing to contribute to FLINT.Cloud repository. If you would like to get in touch with the maintainers for other reasons, please drop a mail at info@moja-global.com.

CHAPTER

ONE

FLINT.CLOUD DEVELOPMENT SETUP

This section guides first-time contributors through installing FLINT.Cloud development environment.

The recommended method for installing the FLINT.Cloud development environment is using App Engine on Gcloud.

Before Setting Up FLINT.Cloud

- Please make sure that all the prerequisites required have been installed and configured correctly.
- The repository has been forked and cloned following the Git and Github guide here .
- It is highly recommended to first setup the FLINT.Cloud API locally to understand the endpoints.

Contents:

1.1 Git and Github guide

This guide is to help new contributors setup git, github and navigate their way through making contributions to moja global repositories. It covers the entire process of contributing right from installing git to opening pull requests.

1.1.1 Setup this project using Git

Before setting up this project using Git make sure you have installed and configured git by following the instructions here.

1.1.2 Fork and Clone this project

- In your browser, visit https://github.com/moja-global/FLINT.Cloud. In the upper left corner, there is a **Fork** button. Please click on it to create a fork/copy of the repository on your profile.
- In the terminal screen, clone this repo by running the command where `your-username` represents your Github username.

```
git clone https://github.com/<your-username>/FLINT.Cloud/
```

- Enter into the newly created project folder by running the command

```
cd FLINT.Cloud
```

- Configure upstream for the fork so that git can sync work from the upstream if it is updated by running the command

```
git remote add upstream https://github.com/moja-global/FLINT.Cloud/
```

- Check if upstream is configured by running the command and check if upstream is shown or not.

```
git remote -v
```

- Now, the project is setup using Git. Please carry on with instructions on how to set this up on Windows or Linux here. You can revisit this section when you are ready to make a contribution.

1.1.3 Claim an issue

This section will demonstrate how to claim an issue to work on using botmojaglobal.

To work on an issue, claim it by adding a comment with `@botmojaglobal claim` to the issue thread. botmojaglobal is a GitHub workflow bot forked from the zulipbot; it will assign you to the issue and label the issue as `in progress`. Some additional notes:

- You can only claim issues with the `Good for newcomers` or `Help Wanted` labels. botmojaglobal will give you an error if you try to claim an issue without one of those labels.
- Please feel free to ask questions on how to approach the issue or if the tests are failing. The maintainers/reviewers will try to get back to you as soon as possible. You can reach us on the moja-global slack, or through Github.
- If your pull request has some requested changes, after working on it don't forget to leave a comment asking for a review since the reviewers aren't notified when a pull request is updated.

1.1.4 Make a contribution

This section will show you step-by-step how to make a contribution to FLINT.Cloud using git.

- FLINT.Cloud stable branch is **master**. All pull requests should be against **master** branch only. Make sure you are in the project directory and checkout to master branch with this command.

```
git checkout master
```

- Choose an issue to work on. We have issues specifically labelled `Good for newcomers` and `Help Wanted` for new contributors to claim. Before starting to work on any issue, make sure you have claimed it.
- Create a new feature branch from master branch to work on. The feature branch should have a short name that is relevant to the issue that you will be working on. For example, if you are working on improving documentation in the readme for adding a badge, the branch can be name `add_badge_readme`.

```
git checkout -b <feature-branch-name>
```

- Work on the task. Add tests and documentation for your changes if required. When you are done with your changes, you can check all the files changes using the following command.

```
git status
```

- Add the relevant files and commit the changes. Please make sure that only those files required for this contribution are added. You can later modify your pull request to add other files as per your requirement.

```
git add <file> <file> ...
```

- While committing the changes, make sure your commit message follows our commit-message guidelines mentioned here.

```
git commit -m "relevant commit message"
```

- Make sure your fork is in sync with the latest changes of master. For this rebase your branch against the latest master by following the commands below.

```
git checkout master
git pull origin master
git checkout <your-branch-name>
git rebase master
```

- Incase there are any merge conflicts on running the rebase command, follow this guide to resolve them.
- You can now push your changes onto your feature branch using the command below.

```
git push origin <your-branch-name>
```

1.1.5 Create a pull request for your contribution

You can now create a pull request to get your changes merged into the upstream master branch. Follow this step-by-step guide to create a pull request on Github.

- Navigate to the pull requests tab under FLINT.Cloud. Click on the **New pull request** button. Compare your feature branch against the **master** branch to create the pull request. Fill the pull request template by linking the issue number solved.
- Incase your pull request is a work in progress, don't forget to add "WIP" in the title of your pull request to let the maintainers know that the pull request is not ready for review yet.
- Please be patient, someone from our team will review your pull request shortly and provide feedback. Incase there are changes requested, you can follow the section below on how to update/modify your pull request.
- Also make sure that your pull request is in sync with the latest master at all times.

NOTE: Don't forget to get credits for your contributions once it gets merged by following this guide here.

1.1.6 Modify your pull request

Incase your pull request needs further changes, you can update your pull request by following the steps below.

- Checkout on your feature branch of the pull request.
- Add the changes as required and commit using the amend flag. This will update the last commit thus keeping the commit history clean and within a single commit.

```
git add <file1> <file2>
git commit -amend
```

- Push this onto your feature branch but this time with force flag. This will update the pull request automatically. The reviewer won't be notified about this updation, so leave a comment in your pull request if you want a review.

```
git push origin <your-branch-name> --force
```

1.1.7 About Hadolint

Hadolint is a Dockerfile linter that helps you build best practice Docker images.

1.2 GCBM Local Development Guide

To start Development of GCBM, you can run the GCBM local-run container by pushing the following commands in the root directory.

```
docker compose up
```

This will take a few minutes to download and install all the required dependenceis once done, Navigate to http://localhost:8080/ in the browser. You can stop the running container by pushing the following command:

```
docker compose down
```

This will stop the docker container instance.

CHAPTER

TWO

FLINT.CLOUD DEPLOYMENT

This section guides you through the process of deploying FLINT.Cloud to your own server, through a public cloud service provider. We use Google Cloud Platform (GCP) and Azure to deploy FLINT.Cloud APIs that are used by FLINT-specific applications, like FLINT-UI. This guide assumes that you have some knowledge of the concepts of GCP and Azure, and that you have successfully deployed an application before.

2.1 Google Cloud Platform

To deploy FLINT.Cloud to GCP, we provide a production grade setup using a Layered Architecture setup on top of the Google Cloud. In this setup we use Terraform, an Infrastructure-as-a-Code tool, to deploy the infrastructure and the applications we want to use.

To deploy the FLINT.Cloud to GCP, follow these steps:

1. Create a GCP service account with project owner permissions in your project. This is used by Terraform to provision all the necessary resources.
2. Copy `main.tf` from the `layered` directory of this repository to your Cloud Console machine.
3. In `main.tf`, change the project variable to your `project ID`. Change any other variables, if necessary.
4. Download the key in JSON format for the service account created in `Step 1` to your project's Cloud Console machine. Rename it to `service_account.json`.
5. Run `terraform apply`. After this command finishes, it should output the URL to FLINT Cloud (ingress).

To tear down the infrastructure and delete the application, run `terraform destroy` in the same directory where `main.tf` is present. If this fails, run it again.

2.2 Azure

To deploy FLINT.Cloud to Azure, we recommend using the Azure App Service with a custom Microsoft Container Registry container built from our `local` directory. This setup uses Azure CLI to deploy the infrastructure and the applications to be used. You need to sign in to the Azure CLI by using the `az login` command. To finish the authentication process, follow the steps displayed in your terminal. To build images, we use Docker and then push them to Azure Container Registry.

2.2.1 Download the project

Clone the repository:

```
git clone https://github.com/moja-global/flint.cloud
```

Navigate to the `local` directory:

```
sh local
```

2.2.2 Build the images locally

To build the `rest_api_gcbm` image locally, run the following command:

```
pushd rest_api_gcbm
docker build --build-arg BUILD_TYPE=RELEASE --build-arg NUM_CPU=4 -t gcbm-api .
popd
```

To build the `rest_api_flint.example` image locally, run the following command:

```
pushd rest_api_flint.example
docker build -t flint-api .
popd
```

2.2.3 Create a resource group

To push images onto and deploy containers with the Azure App Service, you need to first prepare some resources. Start by creating a resource group that will collect all your resources.

```
az group create --name myResourceGroup --location centralindia
```

You can change the `--location` value to specify a region near you.

2.2.4 Create a Container Registry

You can now push the image to Azure Container Registry so that App Service can deploy it. Create an Azure Container Registry to push your images to:

```
az acr create --name <registry-name> --resource-group myResourceGroup --sku Basic --
↪admin-enabled true
```

Replace `<registry-name>` with a suitable name for your registry. The name must contain only letters and numbers, and must be unique across all of Azure.

Retrieve your credentials for the Container Registry:

```
az acr credential show --resource-group myResourceGroup --name <registry-name>
```

Use the `docker login` command to sign in to the container registry:

```
docker login <registry-name>.azurecr.io --username <registry-username>
```

Replace <registry-name> and <registry-username> with values from the previous steps. When prompted, type in one of the passwords from the previous step.

Tag the images with the registry name:

```
docker tag rest_api_gcbm <registry-name>.azurecr.io/rest_api_gcbm:latest
docker tag rest_api_flint.example <registry-name>.azurecr.io/rest_api_flint.
↪example:latest
```

Use the docker push command to push the image to the registry:

```
docker push <registry-name>.azurecr.io/rest_api_gcbm:latest
docker push <registry-name>.azurecr.io/rest_api_flint.example:latest
```

Use the `az acr repository list` command to verify that the push was successful:

```
az acr repository list -n <registry-name>
```

2.2.5 Deploy the image from registry

To deploy a container to Azure App Service, you first create a web app on App Service, then connect the web app to the container registry. When the web app starts, App Service automatically pulls the image from the registry.

Create an App Service plan using the `az appservice plan create` command:

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --is-
↪linux
```

Create the web app with the `az webpp create` command. Since we are deploying two images to two different web apps, you need to enter these commands twice. To deploy `rest_api_gcbm` to the first web app and `rest_api_flint.example` to the second web app, run the following commands:

```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-
↪name-1> --deployment-container-image-name <registry-name>.azurecr.io/rest_api_
↪gcbm:latest
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-
↪name-2> --deployment-container-image-name <registry-name>.azurecr.io/rest_api_flint.
↪example:latest
```

Use the `az webapp config appsettings set` to set the `WEBSITES_PORT` environment variable. In our case, the port to be exposed is `8080`.

```
az webapp config appsettings set --resource-group myResourceGroup --name <app-name-1> --
↪settings WEBSITES_PORT=8080
az webapp config appsettings set --resource-group myResourceGroup --name <app-name-2> --
↪settings WEBSITES_PORT=8080
```

Enable the system-assigned managed identity for the web app by using the `az webapp identity assign` command:

```
az webapp identity assign --resource-group myResourceGroup --name <app-name-1> --query␣
↪principalId --output tsv
az webapp identity assign --resource-group myResourceGroup --name <app-name-2> --query␣
↪principalId --output tsv
```

Replace <app-name> with the name you used in the previous step. The output of the command (filtered by the --query and --output arguments) is the service principal of the assigned identity.

Retrieve your subscription ID with the az account show command, which you need in the next step:

```
az account show --query id --output tsv
```

Grant the managed identity permission to access the container registry:

```
az role assignment create --assignee <principal-id> --scope /subscriptions/<subscription-
↪id>/resourceGroups/myResourceGroup/providers/Microsoft.ContainerRegistry/registries/
↪<registry-name> --role "AcrPull"
```

Replace the following values:

- <principal-id> with the service principal ID from the az webapp identity assign command.
- <registry-name> with the name of your container registry.
- <subscription-id> with the subscription ID retrieved from the az account show command.

Make sure the above steps are repeated for both of the apps that you are going to deploy. Configure your app to use the managed identity to pull from Azure Container Registry.

```
az resource update --ids /subscriptions/<subscription-id>/resourceGroups/myResourceGroup/
↪providers/Microsoft.Web/sites/<app-name-1>/config/web --set properties.
↪acrUseManagedIdentityCreds=True
az resource update --ids /subscriptions/<subscription-id>/resourceGroups/myResourceGroup/
↪providers/Microsoft.Web/sites/<app-name-2>/config/web --set properties.
↪acrUseManagedIdentityCreds=True
```

Replace the following values:

- <subscription-id> with the subscription ID retrieved from the az account show command.
- <app-name> with the name of your web app.

2.2.6 Deploy the image

Use the az webapp config container set command to specify the container registry and the image to deploy for the web app:

```
az webapp config container set --name <app-name-1> --resource-group myResourceGroup --
↪docker-custom-image-name <registry-name>.azurecr.io/rest_api_gcbm:latest --docker-
↪registry-server-url https://<registry-name>.azurecr.io
az webapp config container set --name <app-name-2> --resource-group myResourceGroup --
↪docker-custom-image-name <registry-name>.azurecr.io/rest_api_flint.example:latest --
↪docker-registry-server-url https://<registry-name>.azurecr.io
```

Replace <app-name-1> and <app-name-2> with the name of your web app, and replace all instances of <registry-name> with the name of your registry. When the az webapp config container set command completes, the web app is running in the container on App Service.

To test the app, browse to https://<app-name>.azurewebsites.net, replacing <app-name> with the name of your web app. To clean up the resources, you only need to delete the resource group that contains them:

```
az group delete --name myResourceGroup
```