# Deploying to Substrate using Codefresh



Document Level Classication

200



## Introduction

is a cloud-native continuous integration and delivery platform that enables teams to quickly and efficiently develop, deploy, and manage cloud-native applications. It fetches code from your Git repository and compiles it. Then it deploys the final artifact to a target environment. It is the preferred CI/CD platform for Substrate.

### Create a new pipeline

Codefresh is accessed through the page and using your Epic email. For any questions related to Codefresh access, you can reach out on Slack at .

## Placeholders

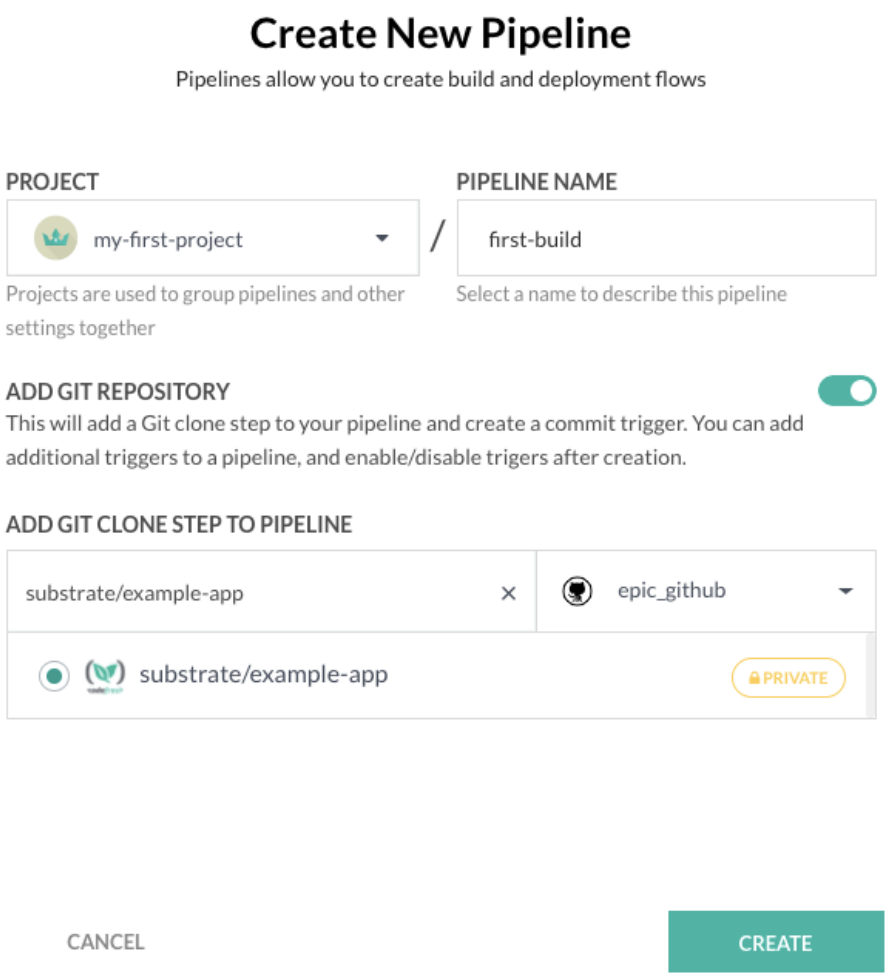
The instructions and code snippets in this tutorial use the following placeholders. Make sure to replace them appropriately based on your Substrate infrastructure.

| Placeholder | Description | Example |
| --- | --- | --- |
| `<PROJECT_NAME>` | The name of your pipeline project for Codefresh. | `my-first-project` |
| `<PIPELINE_NAME>` | The name of your pipeline for Codefresh. | `first-build` |
| `<SUBSTRATE_REPOSITORY>` | The name of your Github repository to connect the pipeline to. | `substrate/example-app` |
| `<HELM_CHART_DIRECTORY>` | The location of Helm chart in your directory. | `example-app/deploy/chart` |
| `<DOCKER_IMAGE_NAME>` | The name of your Docker image. | `substrate/example-app` |
| `<CHART_NAME>` | The name of your Helm chart | `deploy-helm-chart` |
| `<HELM_REPOSITORY_CONTEXT>` | The shared configuration name for the Helm repository with the *epic-artifactory* prefix e.g. if the project | `team-helm-dev` |

| | | |
|---|---|---|
| | key<br>is *btools* then *epic_artifactory_btools-helm-dev* would be a valid example for the *helm-dev* repository) | |
| `<KUBERNETES_CONTEXT>` | Entity defined inside kubeconfig in your cluster to alias cluster parameters with a human-readable name | `abcd-live-substrate` |

## Instructions

1. Log into Codefresh through the Okta login page > Corporate Single Sign-on using your Epic email.
2. Once logged in, you will be able to see a list of projects. Codefresh pipelines are grouped under projects which are similar to folders or directories for your pipelines.
3. Click on the ***New project*** button on the top right corner to get started. Enter a `<PROJECT_NAME>` for your project (e.g. `my-first-project`) and choose a sample icon that you like. You can also optionally add tags that will be used for access control. Once you have typed in a name, click the *Create* button. You now have a project in which you can start adding pipelines.
4. Ensure that you are inside the project you have just created. Click the ***New pipeline*** button in order to create a pipeline. Enter a `<PIPELINE_NAME>` name (e.g. `first-build`). Find the *substrate/example-app* repository from the list and select it. Make sure that the option ***Add Git commit trigger*** is selected. This ensures that your pipeline is automatically launched when a commit happens on this repository.




5. Click the ***Create*** button. Your pipeline is created, and you should now see the pipeline editor. Codefresh has already created a sample pipeline which we will not use for this tutorial. Remove the existing contents on the editor.

6. You will create a pipeline that checks out the source code and builds a docker image. For more details on the YAML, visit [Codefresh documentation]. Paste the following pipeline code on the editor:

```
version: 1.0
stages:                                  # to better visualize the pipel
  - clone
  - prebuild
  - build
```

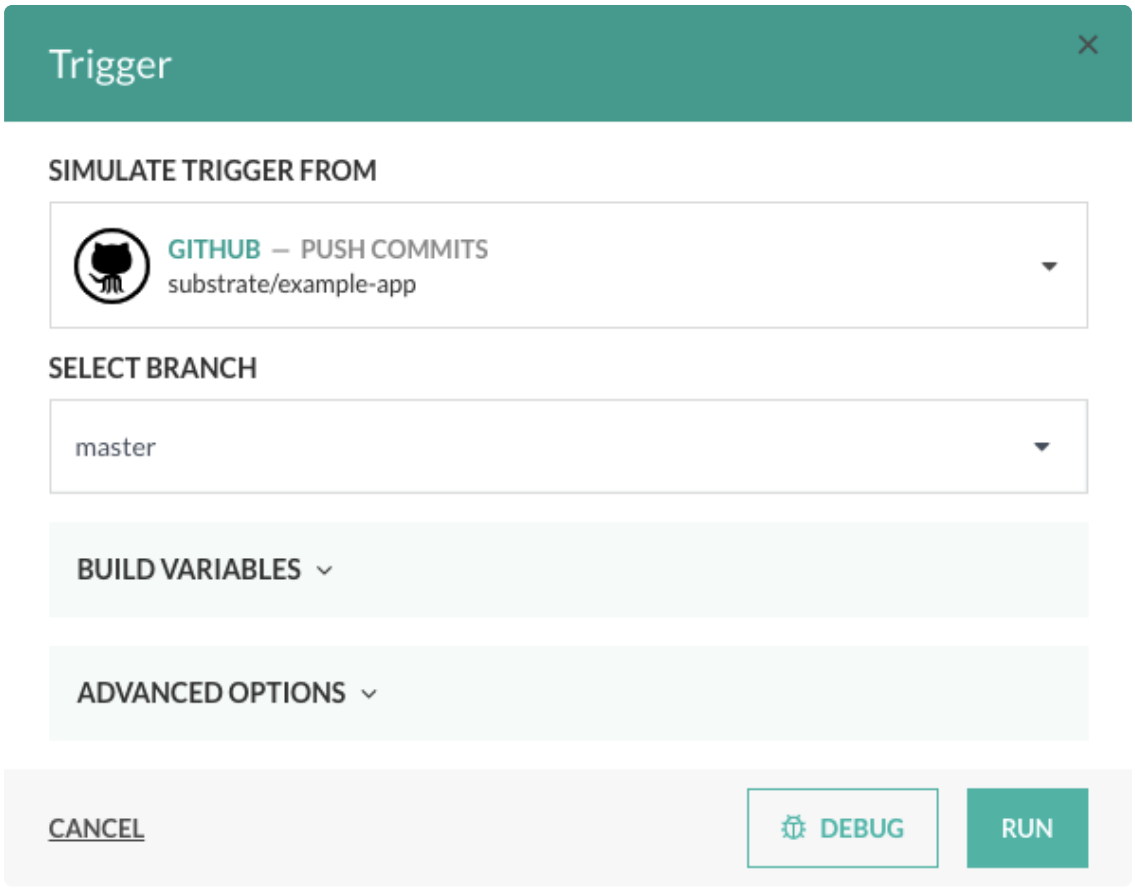
```
  - deploy
  - full_deploy

steps:                                # Codefresh pipelines are compo
  MainClone:                          # name of the step
    title: Clone repository           # title description for the ste
    type: git-clone                   # plugin used in a step for the
    repo: <SUBSTRATE_REPOSITORY>      # Substrate repository to be us
    revision: ${{CF_BRANCH}}          # branch used for the pipeline,
    stage: clone                      # this step will be grouped und
  PreBuild:
    title: Helm versioning
    stage: prebuild
    image: alpine:3.16                # well known image version that
    working_directory: ${{MainClone}}/<HELM_CHART_DIRECTORY>
    commands:
      - "sed -i '2 i \\  imageTag: ${{CF_BRANCH_TAG_NORMALIZED_LOWE
      - cat ./values.yaml
      - version=$(grep -m 1 version ./Chart.yaml)
      - sed -i -e "1,/$version/s/$version/$version-${{CF_BRANCH_TAG
      - cat ./Chart.yaml
  BuildArtifacts:
    type: parallel                    # parallel type allows us to st
    title: Build/push artifacts
    stage: build
    steps:                            # these steps below will be exe
      BuildDocker:
        title: Building Docker image
        type: build                   # build Docker images in your p
        image_name: <DOCKER_IMAGE_NAME>
        working_directory: ${{MainClone}}
        tag: ${{CF_BRANCH_TAG_NORMALIZED_LOWER_CASE}}-${{CF_SHORT_R
        registry: epic_artifactory  # pushes your Docker image to t
        dockerfile: Dockerfile
        stage: build
      PushHelmChart:                  # this step is started in paral
        stage: helm
```

```
      title: Push Helm chart
      type: online-builder-tools/cfstep-epic-helm     # This is a
      working_directory: ${{MainClone}}/<HELM_CHART_DIRECTORY>
      arguments:
        action: push
        chart_name: <CHART_NAME>
        release_name: <CHART_NAME>
        helm_version: 3.7.1
        helm_repository_context:  <HELM_REPOSITORY_CONTEXT>   # H
        credentials_in_arguments: true
  DeployToSubstrate:
    stage: deploy
    title: Deploy to Substrate
    type: online-builder-tools/cfstep-epic-helm           # Custom
    working_directory: ${{MainClone}}
    arguments:
      action: install
      helm_repository_context: <HELM_REPOSITORY_CONTEXT>
      credentials_in_arguments: true
      chart_version: 0.1.0-${{CF_BRANCH_TAG_NORMALIZED_LOWER_CASE}}
      chart_name: <CHART_NAME>
      release_name: <CHART_NAME>
      chart_subdir: tmp
      helm_version: 3.7.1
      kube_context: <KUBERNETES_CONTEXT>
      skip_cf_stable_helm_repo: true
      namespace: <NAMESPACE>
      custom_value_files:
        - deploy/chart/values.yaml
      cmd_ps: "--debug --atomic --wait --timeout 5m"
  FullDeployApproval:               # this step requests manual app
    fail_fast: false
    type: pending-approval
    stage: full_deploy
    title: Perform a full deployment?
  RollbackOnDeniedApproval:         # this step rolls back the depl
    title: Rollback on denied approval
```

```
    stage: canary_deploy
    type: online-builder-tools/cfstep-epic-helm
    arguments:
      action: auth
      kube_context: <KUBERNETES_CONTEXT>
      commands:
        - helm rollback -n <NAMESPACE> <CHART_NAME> 0
    when:
      steps:
        - name: FullDeployApproval
          on:
            - denied
  FullDeploy:
    stage: full_deploy
    title: Full deployment
    type: online-builder-tools/cfstep-epic-helm
    working_directory: ${{MainClone}}/<HELM_CHART_DIRECTORY>
    arguments:
      action: install
      chart_version: 0.1.0-${{CF_BRANCH_TAG_NORMALIZED_LOWER_CASE}}
      chart_name: <CHART_NAME>
      release_name: <CHART_NAME>
      helm_version: 3.7.1
      helm_repository_context: <HELM_REPOSITORY_CONTEXT>
      kube_context: <KUBERNETES_CONTEXT>
      skip_cf_stable_helm_repo: true
      namespace: <NAMESPACE>
      credentials_in_arguments: true
      custom_value_files:
        - ./values.yaml
      cmd_ps: --debug --atomic --wait --timeout 5m
    when:
      steps:
        - name: FullDeployApproval
          on:
            - approved
```

Click the ***Save*** button to apply your changes.

7. This pipeline contains 5 steps.
    1. A ***clone*** step for checking out the code.
    2. A ***prebuild*** step for versioning the Helm chart.
    3. A ***build*** step for building the docker image and pushing it to the connected Docker registry.
    4. A ***deploy*** step for readying deployment to Substrate.
    5. A ***full deploy*** step for requesting manual approval and rolling back if denied or deploying if approved.
8. Now you are ready to start your first build. Click the ***Run*** button to start your pipeline. On the dialog that appears, leave the default selections and click *Run*:




Once the build is started Codefresh will navigate you to the build progress of the sample application.

The build output is split into sections. Expand the section ***Building Docker Image*** and look at the logs. After a while, the build should finish with success.