

1. Technical Architecture	2
1.1 Components	2
1.2 Continuous Integration/Continuous Deployment (CI/CD)	3
1.3 Cloud Formation	16
1.4 An example of deployment using ECS and Docker	18
2. Getting Access	26
3. Installation and Configuration Guide for Developers	27
3.1 Nodejs Installation	27
3.2 Back-End	28
3.3 Database	30
3.4 Front-End	31
3.5 API Requests	32
3.6 Setting up DNS	35

Technical Architecture

This documentation on the architecture of MeydIt, a web application built with ReactJS and Next.js on the front-end, and Node.js and AdonisJS on the back-end, the application uses PostgreSQL for its database. This is hosted on Elastic Beanstalk. To ensure an efficient development and deployment processes, AWS CodePipeline is used for continuous integration and delivery. Additionally, Amazon S3 is utilized for storing images, and Amazon SES serves as the email service.

This document provides a comprehensive guide for setting up the AWS environment using both the console and CloudFormation. The following sections will take you through a step-by-step process for each component of the architecture.

Components

Front-end: The user interface for the web application is built using Next.js, a framework built on top of ReactJS that provides features such as server-side rendering and automatic code splitting, to improve the performance.

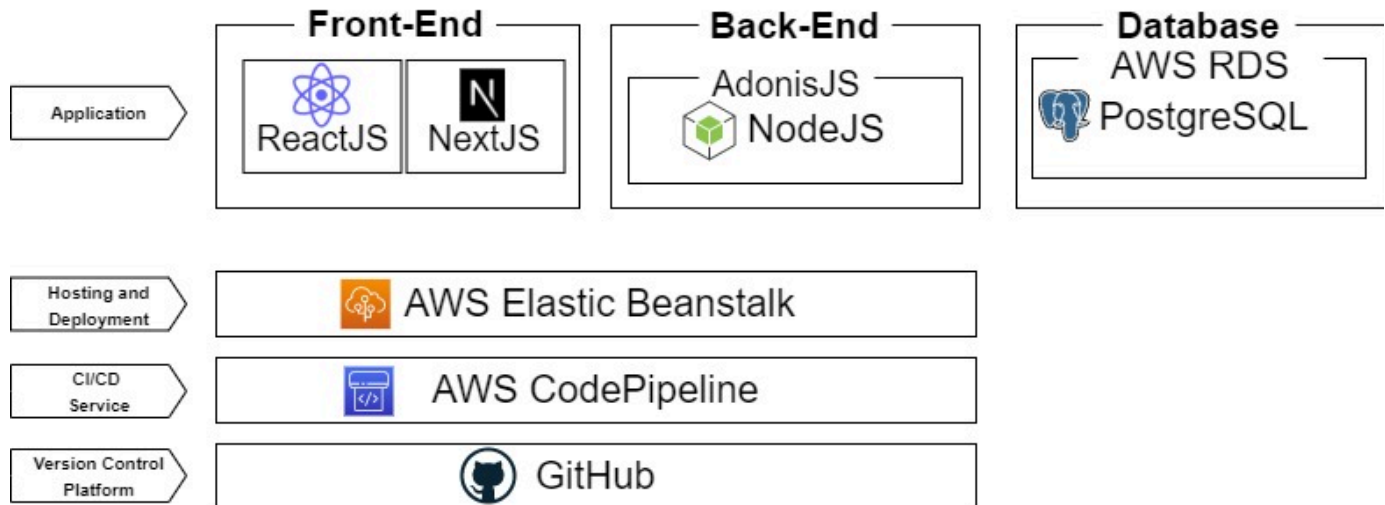
Back-end: Use NodeJS to create the server-side logic. NodeJS is a JavaScript runtime environment that is used to build server-side applications. The Node.js application is built with AdonisJS framework.

Database: Use Amazon RDS PostgreSQL to store and manage the data for the application. Amazon RDS is a managed relational database service that makes it easy to set up, operate, and scale a PostgreSQL database.

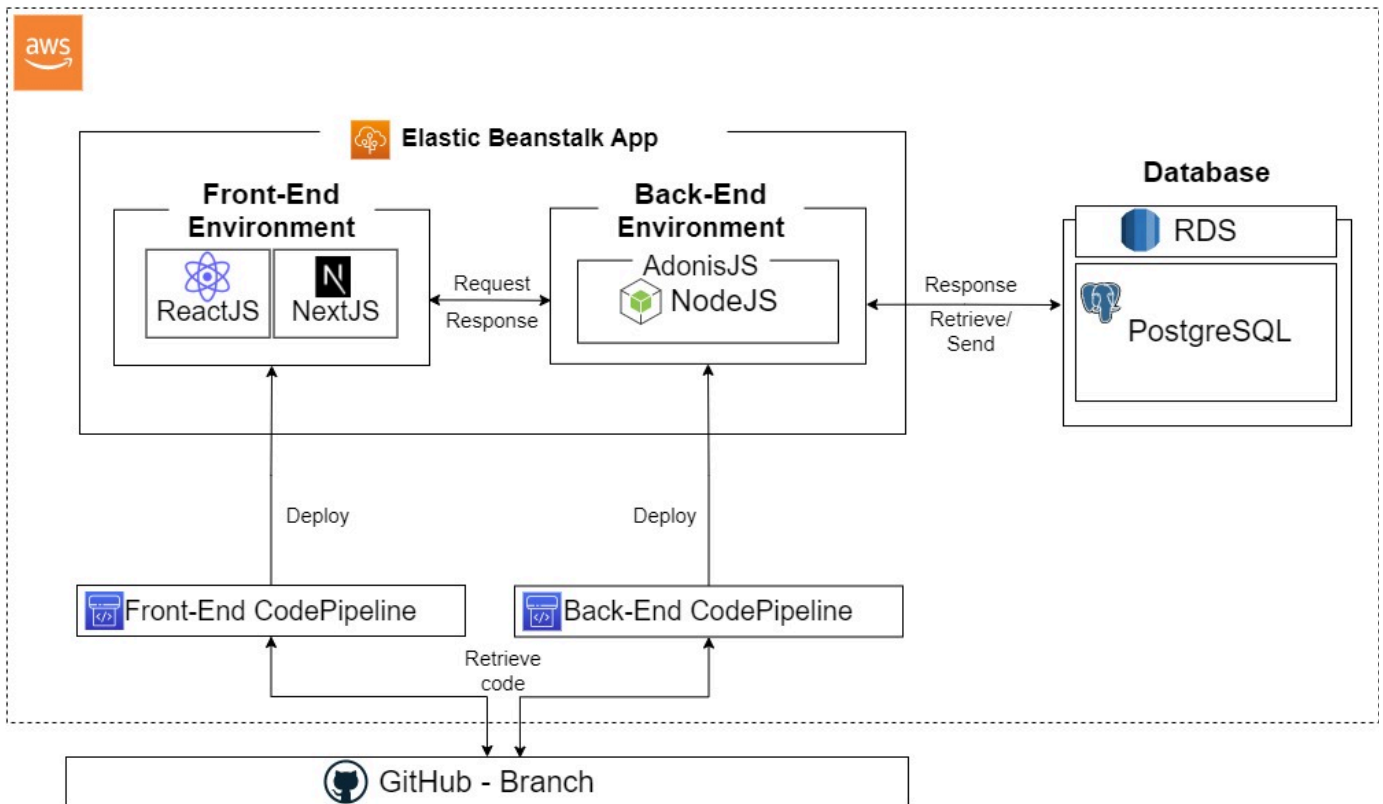
Deployment: Amazon Elastic Beanstalk to deploy and run the application. Elastic Beanstalk is a fully managed service that makes it easy to deploy, run, and scale web applications and services.

Continuous Integration and Deployment: Use AWS CodePipeline to automate the process of building, testing, and deploying the application. CodePipeline integrates with AWS Elastic Beanstalk and GitHub to allow for continuous integration and deployment.

Version control: Use GitHub to manage the source code for the application. GitHub is a web-based platform that provides version control and collaboration tools for software development.



Communication Flow of the Web Application



The communication between these components follows a request-response model, where the front-end (ReactJS and Next.js) makes a request to the back-end (Node.js and AdonisJS), which retrieves data from the database (RDS PostgreSQL) and returns it to the front-end. Next.js is responsible for server-side rendering and communicating with the server-side, while ReactJS handles client-side rendering and user interaction.

The back-end, built with Node.js and AdonisJS, follows an MVC (Model-View-Controller) architecture, where the model represents the data, the view handles the presentation, and the controller manages the flow between the model and view. AdonisJS provides a robust framework for building scalable and maintainable back-end applications, with built-in support for authentication, database integration, and more.

Elastic Beanstalk acts as the hosting platform for the application, providing load balancing and automatic scaling to ensure that the application can handle changes in demand.

CodePipeline integrates with these components to provide a streamlined and efficient release process, automating the deployment of code changes from GitHub to Elastic Beanstalk, and ensuring that the application is always up-to-date and ready for use.

Continuous Integration/Continuous Deployment (CI/CD)

In this document will be described step-by-step guide on how to set up the Continuous Integration/Continuous Deployment (CI/CD) pipeline using AWS CodePipeline and Elastic Beanstalk for MeydIt project, the repository is hosted on GitHub. The CI/CD pipeline automates the process of building, and deploying the application to a production environment.

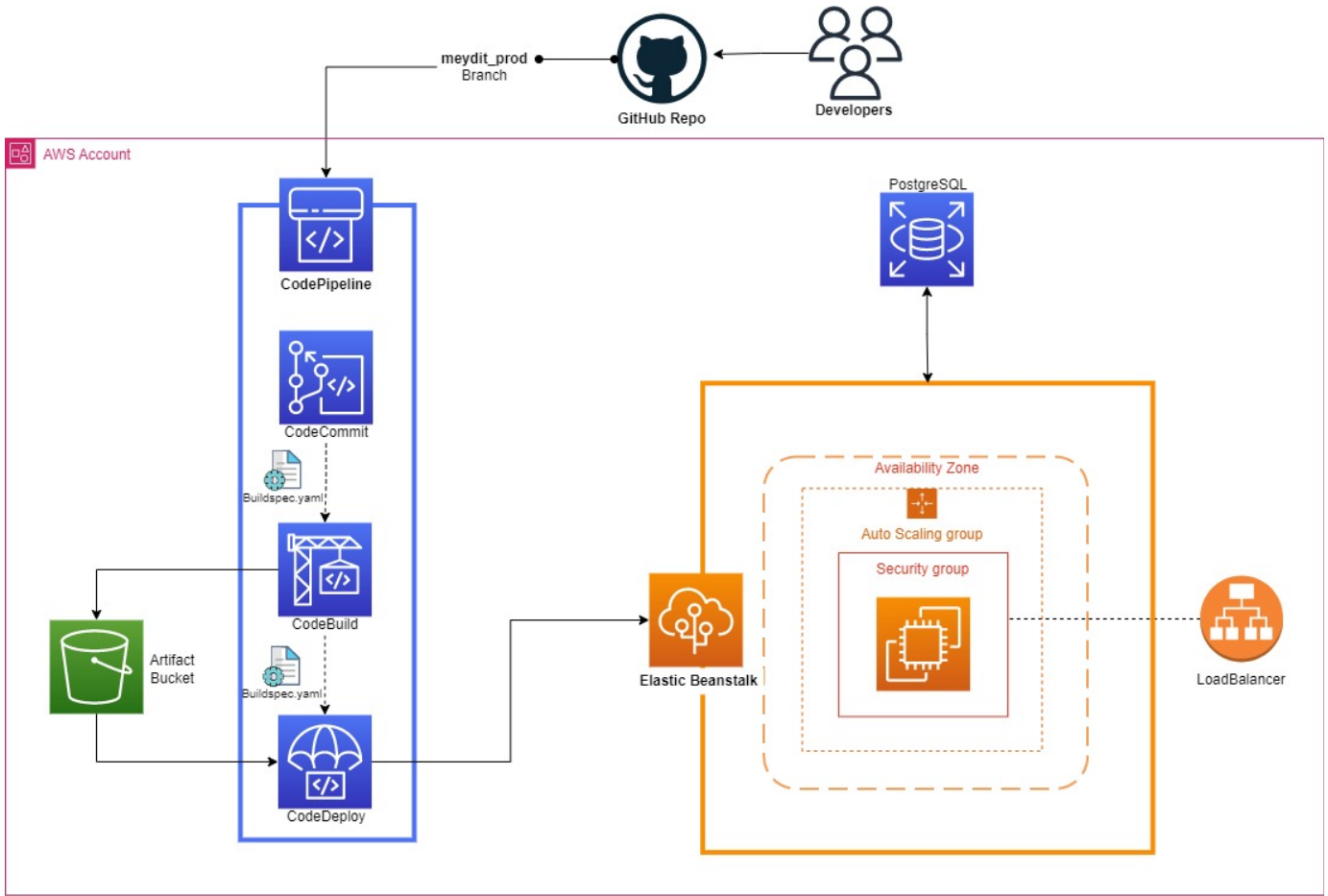
The following information can be found it in this document:

- [Overview](#)
- [Elastic Beanstalk](#)
- [IAM Roles for Elastic Beanstalk](#)
- [CodePipeline](#)
- [Buildspec.yml](#)

Overview

The CI/CD process of MeydIt project begins with the developer committing their code changes to the designated GitHub repository and branch (meydit_prod), which triggers the AWS CodePipeline. CodePipeline then pulls the code changes, builds the application, and deploys it to the Elastic Beanstalk environment.

The Elastic Beanstalk environment provides a scalable and reliable platform to host the application, which can be accessed by end-users. The entire process is automated, reducing the manual effort required for building and deploying the application, ensuring a faster and more efficient release process.



Elastic Beanstalk

To deploy the project is necessary to create an Elastic Beanstalk application and environment first. This provides a platform for hosting the application and allows easily manage and scale resources as needed.

Elastic Beanstalk uses EC2 instances to run the application code, load balancers to distribute incoming traffic, auto scaling groups to manage the number of EC2 instances based on demand, and availability zones to increase the availability and fault tolerance of the application.

• Elastic Beanstalk App

1. Open the Elastic Beanstalk service, and click in Application in the left panel.
2. Click in Create a New Application
3. Type the Application Name and then the button Create

Create new application

Application information

Application name

meydit-app

Maximum length of 100 characters, not including forward slash (/).

Description

• Elastic Beanstalk Environment

1. Navigate to Elastic Beanstalk service and click on `Create Environment` button.
2. Because the project is a web application, choose `Web server environment` and then `Select` button:


Environment tier [Info](#)

Amazon Elastic Beanstalk has two types of environment tiers to support different types of web applications.

☒ **Web server environment**

Run a website, web application, or web API that serves HTTP requests. [Learn more](#) 

☐ **Worker environment**

Run a worker application that processes long-running workloads on demand or performs tasks on a schedule. [Learn more](#) 

3. Type a `Environment Name` and `Domain`:

Environment information [Info](#)

Choose the name, subdomain and description for your environment. These cannot be changed later.

Environment name

Meydit-name

Must be from 4 to 40 characters in length. The name can contain only letters, numbers, and hyphens. It can't start or end with a hyphen. This name must be unique within a region in your account.

Domain

Leave blank for autogenerated value

.us-east-1.elasticbeanstalk.com

4. In platform info, select `Node.js` platform from the list, in branch `Node.js 16 running on 64bit Amazon Linux 2` (this is used for back-end and front-end:

Platform [Info](#)

Platform type

- ☒ **Managed platform**

Platforms published and maintained by Amazon Elastic Beanstalk. [Learn more](#)
- ☐ **Custom platform**

Platforms created and owned by you. This option is unavailable if you have no platforms.

Platform

Node.js

Platform branch


Node.js 16 running on 64bit Amazon Linux 2

Platform version

5.8.0 (Recommended)

5. In the "Application code" section, choose the option `Sample application` because the project will be deployed by the Codepipeline.
6. In Presets, currently the environment is set up with `Single instance (using spot instance)`. There are more options:
 - **Single instance (free tier eligible):** This preset creates a single EC2 instance that is eligible for the AWS free tier. This is a good option for low-traffic websites or applications that don't require high availability or scaling.
 - **Single instance (using spot instance):** This preset creates a single EC2 instance that is deployed using Spot instances. Spot instances are unused EC2 instances that are available at a lower cost. This is a cost-effective option for non-critical applications that can tolerate interruptions.
 - **High availability:** This preset creates multiple EC2 instances in multiple availability zones for high availability and fault tolerance. Elastic Load Balancing is used to distribute traffic across the instances. This is a good option for applications that require high availability and can handle the increased cost.
 - **High availability (using spot and on-demand instances):** This preset creates multiple EC2 instances using a combination of Spot and On-Demand instances to balance cost and availability. Elastic Load Balancing is used to distribute traffic across the instances. This is a good option for cost-sensitive applications that require high availability.
 - **Custom configuration:** This preset allows to configure the environment based on specific requirements. You can choose the number of instances, instance types, load balancers, and other resources to optimize the environment for the application.
7. In Service Access, the role to manage the environment are attached. Currently, the roles are already created, check the section [IAM Roles for Elastic Beanstalk](#):

Service access

IAM roles, assumed by Elastic Beanstalk as a service role, and EC2 instance profiles allow Elastic Beanstalk to create and manage your environment. Both the IAM role and instance profile must be attached to IAM managed policies that contain the required permissions. [Learn more](#) 

Service role

- ☐ Create and use new service role
- ☒ Use an existing service role

Existing service roles

Choose an existing IAM role for Elastic Beanstalk to assume as a service role. The existing IAM role must have the required IAM managed policies.

aws-elasticbeanstalk-service-role ▼



EC2 key pair

Select an EC2 key pair to securely log in to your EC2 instances. [Learn more](#) 

Choose a key pair ▼



EC2 instance profile

Choose an IAM instance profile with managed policies that allow your EC2 instances to perform required operations.

aws-elasticbeanstalk-ec2-role ▼



[View permission details](#)

-
8. This step is just for **back-end**. Configure the database instance. Select `Enable database`, `Engine postgres`, `Engine version 14.6`, `Instance class db.t3.medium`, `Storage 10 GB`, `Username and Password`:

Database [Info](#)

Integrate an RDS SQL database with your environment. [Learn more](#) 

☒ Enable database

Restore a snapshot - *optional*

Restore an existing snapshot from a previously used database.

Snapshot

None ▼

Database settings

Choose an engine and instance type for your environment's database.

Engine

postgres ▼

Engine version

14.6 ▼

Instance class

db.t3.medium ▼

Storage

Choose a number between 5 GB and 1024 GB.

10 GB

Username

meyditadmin

Password

.....|

Availability

Low (one AZ) ▼

Database deletion policy

This policy applies when you decouple a database or terminate the environment coupled to it.

☐ Create snapshot

Elastic Beanstalk saves a snapshot of the database and then deletes it. You can restore a database from a snapshot when you add a DB to an Elastic Beanstalk environment or when you create a standalone database. You might incur charges for storing database snapshots.

☐ Retain

The decoupled database will remain available and operational external to Elastic Beanstalk.

☒ Delete

Elastic Beanstalk terminates the database. The database will no longer be available.

9. In the Capacity section, configure Environment type Load Balanced, Instances Max 4, Fleet composition On-Demand instances, Architecture x86_64, Instance types t3.micro t3.small (for back-end), t3.medium t3.small (for front-end), and then click Next button:

▼ Capacity [Info](#)

Configure the compute capacity of your environment and auto scaling settings to optimize the number of instances used.

Auto scaling group

Environment type

Select a single-instance or load-balanced environment. You can develop and test an application in a single-instance environment to save costs and then upgrade to a load-balanced environment when the application is ready for production. [Learn more](#) [↗](#)

Load balanced ▼

Instances

1

Min

4

Max

Fleet composition

Spot instances are launched at the lowest available price. [Learn more](#) [↗](#)

- ☒ On-Demand instances
- ☐ Combine purchase options and instances

Maximum spot price

The maximum price per instance-hour, in USD, that you're willing to pay for a Spot Instance. Setting a custom price limits your chances to fulfill your target capacity using Spot instances.

- ☒ Default
- ☐ Set your maximum price

On-Demand base

The minimum number of On-Demand Instances that your Auto Scaling group provisions before considering Spot Instances as your environment scales out.

0

On-Demand above base

The percentage of On-Demand Instances as part of any additional capacity that your Auto Scaling group provisions beyond the On-Demand base instances.

0

%

Capacity rebalancing

Specifies whether to enable the capacity rebalancing feature for Spot Instances in your Auto Scaling Group. This option is only relevant when EnableSpot is true in the aws:ec2:instances namespace, and there is at least one Spot Instance in your Auto Scaling group.

- ☐ Turn on capacity rebalancing

Architecture

The processor architecture determines the instance types that are made available. You can't change this selection after you create the environment. [Learn more](#) [↗](#)

- ☒ x86_64
This architecture uses x86 processors and is compatible with most third-party tools and libraries.
- ☐ arm64 - new
This architecture uses AWS Graviton2 processors. You might have to recompile some third-party tools and libraries.

Instance types

Instance types

Add instance types for your fleet. Change the order that the instances are in to set the preferred launch order. This only affects On-Demand instances. We recommend you include at least two instance types. [Learn more](#)

Choose x86 instance types

t3.micro X

t3.small X

10. In the Rolling updates and deployments, choose as Deployment Policy `Rolling` and Batch Size `Percentage 30%`. This deployment policy helps to minimize downtime and reduce the risk of errors or failures during the deployment process by updating the application version gradually and monitoring the health of the updated instances before proceeding to the next batch.

Application deployments

Choose how Amazon Elastic Beanstalk propagates source code changes and software configuration updates. [Learn more](#)

Deployment policy

Rolling

Batch Size

☒ Percentage

☐ Fixed

30

% instances at a time

11. In Platform Software, subsection Environment properties Add the Environment Variables necessities to run the Application. For **back-end** those are some of the environment variables already set up:

ACCESS_KEY_ID

APP_KEY

APP_URL

BUCKET_LOCATION

CACHE_VIEWS

DB_CONNECTION

DB_DEBUG

DB_HEALTH_CHECK

DB_NAME

DOMAIN

GOOGLE_APPLICATION_CREDENTIALS_FILE

HOST

LOG_LEVEL

NODE_ENV

PG_DB_NAME

PG_HOST

PG_PASSWORD

PG_PORT

PG_USER

PORT

REGION

SECRET_ACCESS_KEY

For **front-end** just tree variables are needed:

GA_MEASUREMENT_ID

PORT

STRIPE_PUBLIC_KEY

12. Review your settings and click on `Submit` button to launch the environment.

13. Once the environment is created, Elastic Beanstalk will automatically provision the necessary resources and deploy the application to the environment. You can monitor the deployment process and check the status of the environment from the Elastic Beanstalk dashboard.

IAM Roles for Elastic Beanstalk

To be able to create an environment is a must to configure the service role and EC2 instance profile that Elastic Beanstalk uses to manage the environment. Currently the next roles are created in the environment:

- **aws-elasticbeanstalk-ec2-role**, the following policies are attached:

Policy name ↗

⊕  [AWSElasticBeanstalkWebTier](#)

⊕  [AWSElasticBeanstalkMulticontainerDocker](#)

⊕  [AWSElasticBeanstalkWorkerTier](#)

- **aws-elasticbeanstalk-service-role**, the following policies are attached:

Policy name ↗

⊕  [AWSElasticBeanstalkEnhancedHealth](#)

CodePipeline

Continuous Integration/Continuous Deployment (CI/CD) is a crucial part of modern software development that enables frequent, reliable, and automated deployments. In Meydit project, Amazon Web Services (AWS) CodePipeline are used to implement the CI/CD pipeline, which consists of three stages: source, build, and deploy. Find below a step by step guide to create the CodePipeline for back-end and front-end:

1. Log in to your AWS Management Console and navigate to the CodePipeline service.
2. Click on the `Create pipeline` button.
3. Give the pipeline a name and choose `New service role`. Then click on `Next` button:

Pipeline settings

Pipeline name
Enter the pipeline name. You cannot edit the pipeline name after it is created.

No more than 100 characters

Service role

☒ **New service role**
Create a service role in your account

☐ **Existing service role**
Choose an existing service role from your account

Role name

Type your service role name


☒ Allow AWS CodePipeline to create a service role so it can be used with this new pipeline

4. In the Source section, select `GitHub version2` as the source provider. To choose the connection is necessary to have permission as Owner in GitHub (to know who can provide that permission check this [Getting Access](#)). Select Repository name `Meyd-it/adonis-server`(for **back-end**) and `Meyd-it/next-react-app`(for **front-end**). Then provide the Branch name `meydit_prod`. Click `Next` button.

Source

Source provider
This is where you stored your input artifacts for your pipeline. Choose the provider and then provide the connection details.

GitHub (Version 2) ▼

 **New GitHub version 2 (app-based) action**
To add a GitHub version 2 action in CodePipeline, you create a connection, which uses GitHub Apps to access your repository. Use the options below to choose an existing connection or create a new one. [Learn more](#)

Connection
Choose an existing connection that you have already configured, or create a new one and then return to this task.

X or [Connect to GitHub](#)



Ready to connect

Your GitHub connection is ready for use.

Repository name

Choose a repository in your GitHub account.

Q Meyd-it/next-react-app



<account>/<repository-name>

Branch name

Choose a branch of the repository.

Q meydit_prod



Change detection options

☒ Start the pipeline on source code change

Automatically starts your pipeline when a change occurs in the source code. If turned off, your pipeline only runs if you start it manually or on a schedule.

Output artifact format

Choose the output artifact format.

☒ CodePipeline default

AWS CodePipeline uses the default zip format for artifacts in the pipeline. Does not include git metadata about the repository.

☐ Full clone

AWS CodePipeline passes metadata about the repository that allows subsequent actions to do a full git clone. Only supported for AWS CodeBuild actions.

5. In the Build stage, select AWS CodeBuild as the build provider and create a new CodeBuild project by clicking **Create Project** button. A new window to create the project is opened, give a name to the project.

Project configuration

Project name

meydit-project

A project name must be 2 to 255 characters. It can include the letters A-Z and a-z, the numbers 0-9, and the special characters - and _.

Description - *optional*

Enable concurrent build limit - *optional*

Limit the number of allowed concurrent builds for this project.

☐ Restrict number of concurrent builds this project can start

► Additional configuration

tags

In Environment section, choose Operating system `Ubuntu`, Runtime `Standard`, Image `aws/codebuild/standard:6.0` and Environment type `Linux`.

Environment

Environment image



Managed image

Use an image managed by AWS CodeBuild



Custom image

Specify a Docker image

Operating system

Ubuntu



The programming language runtimes are now included in the standard image of Ubuntu 18.04, which is recommended for new CodeBuild projects created in the console. See [Docker Images Provided by CodeBuild for details](#).

Runtime(s)

Standard

Image

aws/codebuild/standard:6.0

Image version

Always use the latest image for this runtime version

Environment type

Linux

In Buildspec section, give the name of the buildspec file that is in the root of the project. This file is different for back-end and front-end, check this section to know more [Buildspec.yml files](#). Click `Continue` to `CodePipeline` button.

Buildspec

Build specifications



Use a buildspec file

Store build commands in a YAML-formatted buildspec file



Insert build commands

Store build commands as build project configuration

Buildspec name - optional

By default, CodeBuild looks for a file named `buildspec.yml` in the source code root directory. If your buildspec file uses a different name or location, enter its path from the source root here (for example, `buildspec-two.yml` or `configuration/buildspec.yml`).

buildspec.yml

6. After create the project, in the build stage stage, click the `Next` button.

7. In the deploy stage section, choose the Deploy provider `Elastic Beanstalk`, Application Name and the Environment name already created in the Elastic Beanstalk. Click `Next` button.

Deploy - optional

Deploy provider

Choose how you deploy to instances. Choose the provider, and then provide the configuration details for that provider.

AWS Elastic Beanstalk

Region

US East (N. Virginia)

Application name

Choose an application that you have already created in the AWS Elastic Beanstalk console. Or create an application in the AWS Elastic Beanstalk console and then return to this task.

Q meydit-eb

Environment name

Choose an environment that you have already created in the AWS Elastic Beanstalk console. Or create an environment in the AWS Elastic Beanstalk console and then return to this task.

Q

Meyditeb-back-end-env

Meyditeb-frontend-env

kip deploy stage

Next

8. Review your pipeline settings and click on `Create pipeline`.
9. Once the pipeline is created, CodePipeline will automatically trigger a build and deploy process based on the settings that were setting up. You can monitor the progress of each stage in the pipeline and view the logs and artifacts generated by each action.

Buildspec.yml

- Back-End: Following is a description of the buildspec yaml file for the back-end:
 - The version: 0.2 indicates the version of the build specification syntax to use.
 - The phases section specifies the different phases of the build process. In this case, there are two phases: install and build. The *install* phase specifies that the Node.js runtime version 16 should be used, and that npm install should be run to install the dependencies for the project. The *build* phase specifies that the command npm run build should be run to build the project.
 - The artifacts section specifies the output artifacts of the build process. In this case, it specifies that all files should be included in the output artifact.

```
version: 0.2

phases:
  install:
    runtime-versions:
      nodejs: 16
    commands:
      - npm install

  build:
    commands:
      - npm run build

artifacts:
  files:
    - '**/*'
```

- Front-End: Following is a description of the builds spec yaml file for the front-end:
 - The version 0.2 at the top specifies the version of the build specification language.
 - The phases section defines the different stages of the build process. In this case, the *install* phase installs the Node.js runtime version 16 and runs the npm install command to install the project's dependencies. The *pre_build* phase runs a command to print a message saying it's installing NPM dependencies. The *build* phase runs the npm install command to install the @svgr/webpack@5.5.0 package and the npm run build command to build the Next.js application. The *post_build* phase prints a message saying the build is completed on the current date.
 - The artifacts section specifies the type of artifact and the files that should be included in it. In this case, the artifact is a zip file, and the files include the Next.js build output files in the .next directory, the package.json file, and the next.config.js file.
 - The cache section specifies the paths to be cached, which in this case is the node_modules directory to speed up future builds by reusing previously installed dependencies.

```
version: 0.2

phases:
  install:
    runtime-versions:
      nodejs: 16
    commands:
      - npm install
  pre_build:
    commands:
      - echo Installing source NPM dependencies...
  build:
    commands:
      - echo Build started on `date`
      - npm install @svgr/webpack@5.5.0
      - npm run build
  post_build:
    commands:
      - echo Build completed on `date`
artifacts:
  type: zip
  files:
    - '.next/**/*'
    - 'package.json'
    - 'next.config.js'
cache:
  paths:
    - node_modules
```

Cloud Formation

CloudFormation is used to create and manage a collection of related AWS resources, provision them in an orderly and predictable fashion, and update and delete them as a single unit. With CloudFormation, you can easily reproduce and manage the infrastructure in a consistent manner, while also maintaining version control of your infrastructure as code.

To use CloudFormation, a CloudFormation template in JSON or YAML format is required, it describes the desired resources and their configurations. There are two ways to create a template, first one is writing the CloudFormation template from scratch or use one of the pre-built templates provided by AWS or other third-party vendors. Once the CloudFormation template is created, it is possible to use the AWS Management Console, AWS CLI, or SDKs to create, update, or delete a stack that contains the resources specified in the template.

Using CloudFormation provides several benefits, including:

- Automation: CloudFormation automates the provisioning and deployment of AWS resources, so you don't have to perform each task manually.

- Consistency: You can ensure that your infrastructure is consistent across different environments and applications.
- Reusability: You can reuse templates across different applications and environments.
- Scalability: You can easily scale up or down your infrastructure based on the changing demands of your application.
- Cost-Effective: You only pay for the AWS resources that you use, and you can easily track and manage your expenses using AWS Cost Explorer.

Elastic Beanstalk

Application Using CDK

```
const application = new beanstalk.CfnApplication(this,
'UnoElasticBeanstalkCdk', {
  applicationName: 'unoelasticbeanstalkcdk'
});

// Create an IAM role for environment
const elasticbeanstalkRole = new iam.Role(this,
'ElasticBeanstalkRoleCdk', {
  assumedBy: new iam.ServicePrincipal('ec2.amazonaws.com'),
});

const instanceProfile = new iam.CfnInstanceProfile(this,
'ProfileEBRoleCdk', {
  instanceProfileName: 'profileebrolecdk',
  roles: [
    elasticbeanstalkRole.roleName
  ]
});
```

Environment Using CDK

```
const environment = new beanstalk.CfnEnvironment(this,
'UnoEnvironmentCdk', {
  environmentName: 'unoenvironmentcdk',
  applicationName: application.applicationName ||
'unoelasticbeanstalkcdk',
  solutionStackName: '64bit Amazon Linux 2 v5.7.0 running Node.js
16',
  optionSettings: [
    {
      namespace: 'aws:autoscaling:launchconfiguration',
      optionName: 'InstanceType',
      value: 't2.medium'
    },
    {
      namespace: 'aws:autoscaling:launchconfiguration',
      optionName: 'IamInstanceProfile',
      value: instanceProfile.instanceProfileName,
    }
  ]
});
```

```
    ],  
  } ) ;
```

CodePipeline

Cloud Formation Template

An example of deployment using ECS and Docker

The process to deploy the project using Elastic Container Service and CodePipeline works as following:

- Developers push the code changes to the source repository, which triggers the CodePipeline.
- The CodePipeline retrieves the source code from the source repository and starts the build process. The build process creates a Docker image that includes the application and its dependencies.
- After the build process completes, the Docker image is pushed to the container repository.
- The CodePipeline triggers an update to the ECS task definition, which specifies the container image to use and any other relevant configuration settings.
- The ECS service takes the new task definition and creates a new task with the updated container image. The new task is launched in the same cluster as the existing tasks.
- The ECS service automatically drains connections from the existing tasks and redirects them to the new task.
- Once all connections have been drained from the old tasks, they are stopped and removed.
- The deployment is complete, and the application runs on the updated container image.

The following information can be found it in this document:

- [Create container repository](#)
- [Create an ECS service](#)
- [Create a Task Definition](#)
- [Create a Cluster's Service](#)
- [Set up the CodePipeline](#)
- [Files in the project root](#)

Create container repository

The container repository is used to store the Docker image. Follow the next steps to create it:

1. Navigate to Elastic Container Registry (ECR)
2. Click in `Create Repository` button.

Provide a name, and click in `Create Repository` button.

General settings

Visibility settings | Info

Choose the visibility setting for the repository.

☒ Private

Access is managed by IAM and repository policy permissions.

☐ Public

Publicly visible and accessible for image pulls.

Repository name

Provide a concise name. A developer should be able to identify the repository contents by the name.

855390817614.dkr.ecr.us-east-1.amazonaws.com/ meydit-repository

Create an ECS service

1. Navigate to Elastic Container Service.
2. Click in `Create Cluster` button.
3. In `Cluster template`, choose `EC2 Linux + Networking`. Then click in `Next Step` button.

Select cluster template

The following cluster templates are available to simplify cluster creation. Additional configuration and integrations can be added later.

Networking only ⓘ

Resources to be created:

Cluster

VPC (optional)

Subnets (optional)

ⓘ For use with either AWS Fargate (Windows/Linux) or with External instance capacity.

EC2 Linux + Networking

Resources to be created:

Cluster

VPC

Subnets

Auto Scaling group with Linux AMI

EC2 Windows + Networking

Resources to be created:

Cluster

VPC

Subnets

Auto Scaling group with Windows AMI

4. Give a name to the cluster:

Configure cluster

Cluster name*

meydit-cluster

☐

Create an empty cluster

5. In instance configuration section, choose the instance type `t3.medium` and number of instances 1:

Instance configuration

Provisioning Model ☒ On-Demand Instance

With On-Demand Instances, you pay for compute capacity by the hour, with no long-term commitments or upfront payments.

☐ Spot

Amazon EC2 Spot Instances let you take advantage of unused EC2 capacity in the AWS cloud. Spot Instances are available at up to a 90% discount compared to On-Demand prices.

[Learn more](#)

EC2 instance type*

t3.medium



☐ Manually enter desired instance type

Number of instances*

1



EC2 AMI ID*

Amazon Linux 2 AMI [ami-0533...



Root EBS Volume Size (GiB)

30



Key pair

None - unable to SSH



You will not be able to SSH into your EC2 instances without a key pair. You can create a new key pair in the [EC2 console](#).

6. In networking section, choose VPC the one by default, Subnets select just one, Auto assign public IP enabled and security group the one by default. Then click **Create** button.

Networking

Configure the VPC for your container instances to use. A VPC is an isolated portion of the AWS cloud populated by AWS objects, such as Amazon EC2 instances. You can choose an existing VPC, or create a new one with this wizard.

VPC

vpc-7fbf2002 (172.31.0....



Check the structure for [vpc-7fbf2002](#) in the Amazon EC2 console.

Subnets

subnet-306b2211
(172.31.80.0/20) - us-east-1c



assign ipv6 on creation: Disabled

Select a subnet...

Auto assign public IP

Enabled

Security group

sg-d12e6dc8 (default)

Rules for [sg-d12e6dc8](#) in the EC2 Console.

Create a Task Definition

1. Navigate to Elastic Container Service.
2. Click in Task Definition in the left menu.
3. Click in Create new Task Definition button.
4. Click in EC2 as launch type compatibility. Then, click in Next button.

Select launch type compatibility

Select which launch type you want your task definition to be compatible with based on where you want to launch your task.

FARGATE



Price based on task size

Requires network mode awsvpc

AWS-managed infrastructure, no Amazon EC2 instances to manage

EC2



Price based on resource usage

Multiple network modes available

Self-managed infrastructure using Amazon EC2 instances

EXTERNAL



Price based on instance-hours and additional charges for other AWS services used

Self-managed on-premise infrastructure with ECS
Anywhere

5. Give a name to the task definition:

Configure task and container definitions

A task definition specifies which containers are included in your task and how they interact with each other. You can use volumes for your containers to use. [Learn more](#)

Task definition name*

meydit-task



Requires compatibilities* EC2

Task role

Select a role...



Optional IAM role that tasks can use to make API requests to authorized AWS services. Create an Amazon Elastic Container Service Task Role in the [IAM Console](#)

Network mode

<default>



If you choose <default>, ECS will start your container using Docker's default networking mode, which is Bridge on Linux and NAT on Windows. Windows tasks support the <default> and awsvpc network modes.

6. In task memory enter 128:

Task size



The task size allows you to specify a fixed size for your task. Task size is required for tasks using the Fargate launch type and is optional for the EC2 or External launch type. Container level memory settings are optional when task size is set. Task size is not supported for Windows containers.

Task memory (MiB)

128

The amount of memory (in MiB) used by the task. It can be expressed as an integer using MiB, for example 1024, or as a string using GB, for example '1GB' or '1 gb'.

Task CPU (unit)

The number of CPU units used by the task. It can be expressed as an integer using CPU units, for example 1024, or as a string using vCPUs, for example '1 vCPU' or '1 vcpu'.

Task memory maximum allocation for container memory reservation



0

128 shared of 128 MiB

7. Click in Add Container button. Give a name to the container, add an image URI and set the port mappings to 80:3000. Then, click in Add button.

Standard

Container name*

meydit-container



Image*

image-uri



Private repository authentication* ☐



Memory Limits (MiB)*

Hard limit ▼

128



[+ Add Soft limit](#)

Define hard and/or soft memory limits in MiB for your container. Hard and soft limits correspond to the `memory` and `memoryReservation` parameters, respectively, in task definitions. ECS recommends 300-500 MiB as a starting point for web applications.

Port mappings

Host port

Container port

Protocol



80

3000

tcp ▼



[+ Add port mapping](#)

8. Click in **Create** button.

Create a Cluster's Service

1. Navigate to Elastic Container Service.
2. Click in the name of the Cluster that was created.
3. In the tag **Services**, click in **Create** button.
4. Launch type choose **EC2**, select the task definition that was created, give a name for the service, number of tasks type 1 :

Configure service

A service lets you specify how many copies of your task definition to run and maintain in a cluster. You can optionally use an Elastic Load Balancing load balancer to distribute incoming traffic to containers in your service. Amazon ECS maintains that number of tasks and coordinates task scheduling with the load balancer. You can also optionally use Service Auto Scaling to adjust the number of tasks in your service.

Launch type ☐ FARGATE ☒ EC2 ☐ EXTERNAL



[Switch to capacity provider strategy](#)



Task Definition

Family

meydit-frontend ▼

[Enter a value](#)

Revision

13 ▼

Cluster

meydit-frontend-ecs01 ▼



Service name

meydit-service



Service type* ☒ REPLICAS ☐ DAEMON



Number of tasks ⓘ

Minimum healthy percent ⓘ

Maximum percent ⓘ

Deployment circuit breaker ⓘ

5. Click in `Create Service` button.

Set up the CodePipeline

Check the section [CodePipeline](#), the configuration is the same, except for the Deploy section. In Deploy stage instead Beanstalk, choose Deploy provider `Amazon ECS`, the Cluster Name and Service Name that were created:

Deploy - optional

Deploy provider

Choose how you deploy to instances. Choose the provider, and then provide the configuration details for that provider.

▼

Region

▼

Cluster name

Choose a cluster that you have already created in the Amazon ECS console. Or create a cluster in the Amazon ECS console and then return to this task.

🔍 ✕

Service name

Choose a service that you have already created in the Amazon ECS console for your cluster. Or create a new service in the Amazon ECS console and then return to this task.

🔍 ✕

The `Buildspec.yml` file is different, it will be explained in the next section.

Files in the project root

• Docker file

This code defines a Dockerfile that is used to build a Node.js application. The Dockerfile is divided into two stages: the build stage and the production stage.

- The build stage uses the official Node.js 16-alpine Docker image as a base image. The `ARG` directive is used to declare a build-time argument `ENV_VARIABLE`, which is used to set an environment variable `NEXT_PUBLIC_GA_MEASUREMENT_ID`, in this section more variable can be added. The `WORKDIR` directive sets the working directory for the subsequent instructions. The `COPY` directive copies the contents of the current directory (the root directory of the application) to the working directory of the Docker image. The `RUN` directive executes the `npm install` and `npm run build` commands to install dependencies and build the application. The last two `RUN` directives remove the development dependencies and install only the production dependencies.
- The production stage uses the official Node.js 16-alpine Docker image as a base image as well. It also uses the same `ARG` directive to set the same environment variable `NEXT_PUBLIC_GA_MEASUREMENT_ID`. The `ENV` directive sets the `NODE_ENV` environment variable to

production. The RUN directives create a new user and group, and create a working directory for the application. The COPY directive copies the necessary files from the build stage to the production stage, and sets the ownership of the files to the new user and group. The EXPOSE directive exposes port 8080 for the container, and the CMD directive specifies the command to start the application.

```
FROM node:16-alpine AS BUILD_IMAGE
ARG ENV_VARIABLE
ENV NEXT_PUBLIC_GA_MEASUREMENT_ID ${ENV_VARIABLE}
RUN mkdir -p /usr/app/
WORKDIR /usr/app

COPY ./ ./

RUN npm install
RUN npm run build
RUN rm -rf node_modules
RUN npm install --production

FROM node:16-alpine
ARG ENV_VARIABLE
ENV NEXT_PUBLIC_GA_MEASUREMENT_ID ${ENV_VARIABLE}
ENV NODE_ENV production
RUN addgroup -g 1001 -S user_group
RUN adduser -S application -u 1001
RUN mkdir -p /usr/app/
WORKDIR /usr/app
COPY --from=BUILD_IMAGE --chown=application:user_group /usr/app
/node_modules ./node_modules
COPY --from=BUILD_IMAGE --chown=application:user_group /usr/app/package.
json ./
COPY --from=BUILD_IMAGE --chown=application:user_group /usr/app/package-
lock.json ./
COPY --from=BUILD_IMAGE --chown=application:user_group /usr/app/public .
/public
COPY --from=BUILD_IMAGE --chown=application:user_group /usr/app/.next .
/.next

EXPOSE 8080
CMD ["npm", "start"]
```

- Buildspec.yml

This is the CodeBuild buildspec file for building a Docker image, tagging it, pushing it to a repository, and deploying the updated image to an ECS service.

- env section defines environment variables for the build process.
- pre_build phase performs actions before the build process starts, such as logging in to Amazon ECR and Docker Hub, and setting variables.
- build phase executes the main build process which includes building a Docker image, tagging it with a version tag derived from the commit hash or a default value, and pushing it to the ECR repository.
- post_build phase runs after the build and deployment is completed, which includes writing the image definition file, updating the ECS service with the latest image and forcing a new deployment.

- `artifacts` section specifies the output files to include in the build artifacts, which in this case is the `imagedefinitions.json` file that contains the image URI for the updated image.

```

version: 0.2

env:
  variables:
    DOCKERHUB_USERNAME: "your-username"
    DOCKERHUB_PASSWORD: "your-password"

phases:
  pre_build:
    commands:
      - echo Logging in to Amazon ECR...
      - aws --version
      - aws ecr get-login-password --region us-east-1 | docker login --
username AWS --password-stdin 855390817614.dkr.ecr.us-east-1.amazonaws.
com
      - REPOSITORY_URI=855390817614.dkr.ecr.us-east-1.amazonaws.com
/meydit-frontend
      - COMMIT_HASH=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c
1-7)
      - IMAGE_TAG=${COMMIT_HASH:=prod}
      - echo Logging in to Docker Hub...
      - echo "$DOCKERHUB_PASSWORD" | docker login --username
"$DOCKERHUB_USERNAME" --password-stdin
    build:
      commands:
        - echo Build started on `date`
        - echo Building the Docker image...
        - docker build --build-arg ENV_VARIABLE=357846786 -t
$REPOSITORY_URI:prod .
        - docker tag $REPOSITORY_URI:prod $REPOSITORY_URI:$IMAGE_TAG
    post_build:
      commands:
        - echo Build completed on `date`
        - echo Pushing the Docker images...
        - docker push $REPOSITORY_URI:prod
        - docker push $REPOSITORY_URI:$IMAGE_TAG
        - echo Writing image definitions file...
        - printf ' [{ "name": "meydit-frontend", "imageUri": "%s" } ] '
$REPOSITORY_URI:$IMAGE_TAG > imagedefinitions.json
        - aws ecs update-service --cluster meydit-frontend-ecs01 --
service meydit-frontend --force-new-deployment --region us-east-1 --
deployment-configuration "maximumPercent=200,minimumHealthyPercent=0"
    artifacts:
      files: imagedefinitions.json

```

Getting Access

Below are the platforms/tools that require credentials and the responsible for managing them:

Platform/Tool	Description	Responsible
Slack	This is the main communication channel for team messaging and collaboration	@ Susan Hansen
Confluence	We use Confluence to create and store all of the project documentation.	@ Susan Hansen
Jira/Trello	These tools help to manage development tasks, such as creating tickets for new features and tracking bugs.	@ Susan Hansen
GitHub	We use GitHub for version control and as a repository for our front-end and back-end code.	@ Mayjo Antony Parth @ Susan Hansen
GoDaddy	The project's DNS is managed by GoDaddy, which is the company responsible for its domain name registration and DNS configuration	@ Susan Hansen
AWS	The deployment and CD/CI processes are managed using AWS services.	@ Mayjo Antony @ Susan Hansen @ Evelyn Caviedes Arciniegas

Installation and Configuration Guide for Developers

This guide is intended to help developers set up the Meyd.It project to work with Front-End, Back-End, Database, and Test Back-End Routes. By following each step, you will be able to work with all the components of the project.

This is the Guide Content:

Nodejs Installation

The back-end is designed to work with Node.js version 16.19.0. For this reason, it is recommended to use a Node Version Manager (nvm). Follow the steps below to install it:

Install nvm

For Windows PowerShell, download and run the installation script using the following command:

```
iwr -useb https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | iex
```

For Git Bash, use the following command:

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash
```

If after do the installation the next message is displayed:

```
=> Profile not found. Tried ~/.bashrc, ~/.bash_profile, ~/.zshrc, and ~/.profile.
```

Then, run the next command:

```
export NVM_DIR="$HOME/.nvm"  
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh" # This loads nvm
```

Install Node.js

Install the needed version for back-end with the next command

```
nvm install 16.19.0
```

Install the version for front-end with the next command

```
nvm install 18.15.0
```

Switch between Node.js versions, run the next command, replace version for the version number that will be used

```
nvm use <version>
```

Back-End

Setting up

Use the following command to clone the `adonis-server` repository

```
git clone https://github.com/Meyd-it/adonis-server.git
```

Change your current directory to the cloned project directory:

```
cd adonis-server
```

Switch to the `minh_dev` branch:

```
git checkout minh_dev
```

Create a new branch based on the branch-naming conventions described below [Branch Naming Conventions](#) :

```
git checkout -b new_branch_name
```

Run the following command to install AdonisJS:

```
npm install -g @adonisjs/cli
```

Install the required dependencies for the project using the following command:

```
npm install
```

Create a new file called `.env` and copy the following values. The `APP_KEY` value will be generated in the next step.

```
#App
HOST=127.0.0.1
PORT=3333
NODE_ENV=development
APP_URL=http://${HOST}:${PORT}
APP_KEY=
LOG_LEVEL=debug

#DB
DB_CONNECTION=pg
DB_HEALTH_CHECK=true
DB_DEBUG=true

#PG
PG_HOST=127.0.0.1
PG_PORT=5432
PG_USER=postgres
PG_PASSWORD=yourpassword
PG_DB_NAME=yourdbname

#GOOGLE
STRIPE_SECRET_KEY=
GOOGLE_APPLICATION_CREDENTIALS_FILE=./google-credentials.json

SESSION_DRIVER=cookie
CACHE_VIEWS=false
```

Generate a new application key by running the following command:

```
adonis key:generate
```

Start the development server using the following command:

```
npm run dev
```

The server will be running at <http://127.0.0.1:3333>

Database

Install PostgreSQL

Download PostgreSQL from the official website at <https://www.postgresql.org/download/>

Install PostgreSQL using the downloaded installer.

During the installation process, set a password for the default "postgres" user.

After the installation is complete, open the env. file and replace "<your-password>" with the password you set during the installation process:

```
PG_PASSWORD=<your-password>
```

Note:

If the default port 5432 is already in use on your system, you can specify a different port for the installation, make sure to change the port number in the env. file, for example:

```
PG_PORT=5433
```

Migration

Navigate to the back-end directory `adonis-server`:

```
cd adonis-server
```

Install the `pg` package (PostgreSQL client for Node.js):

```
npm install pg
```

Start Adonis server in watch mode with this command:

```
node ace serve --watch
```

Run the next command to create the database:

```
node ace pg:setup
```

To create the tables, run the next command:

```
node ace migration:run
```

Front-End

Setting up

Clone the project from GitHub repository called `next-react-app`:

```
git clone https://github.com/Meyd-it/next-react-app.git
```

Change your current directory to the cloned project directory:

```
cd next-react-app
```

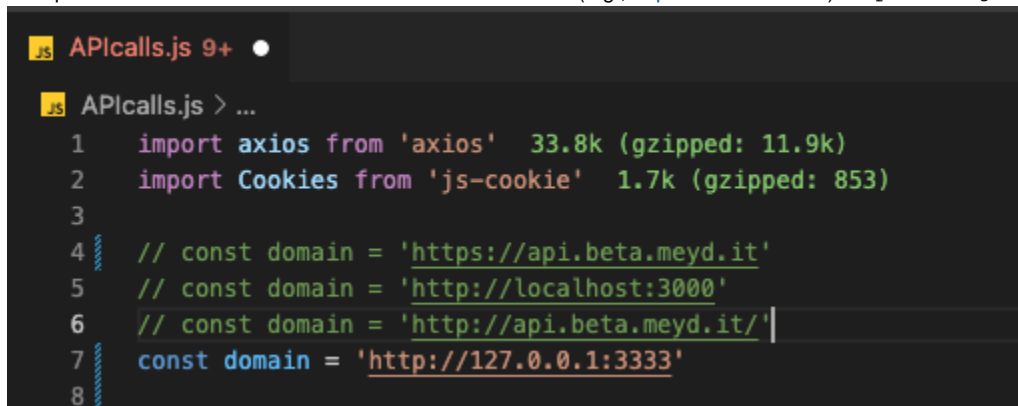
Navigate to the `minh_dev` development branch:

```
git checkout minh_dev
```

Create your own branch using the branch naming conventions specified below [Branch Naming Conventions](#):

```
git checkout -b new_branch_name
```

Set up the domain value obtained from the back-end server (e.g., <http://127.0.0.1:3333>) in `ApiCalls.js` file



```
1 import axios from 'axios' 33.8k (gzipped: 11.9k)
2 import Cookies from 'js-cookie' 1.7k (gzipped: 853)
3
4 // const domain = 'https://api.beta.meyd.it'
5 // const domain = 'http://localhost:3000'
6 // const domain = 'http://api.beta.meyd.it/'
7 const domain = 'http://127.0.0.1:3333'
8
```

Install the required dependencies by running the following command:

```
npm install --force
```

Start the development server by running the following command:

```
npm run dev
```

Error running the server

If you encounter the following error when running the server:

```
ERROR
  opensslErrorStack: [ 'error:03000086:digital envelope routines::
initialization error' ],
  library: 'digital envelope routines',
  reason: 'unsupported',
  code: 'ERR_OSSL_EVP_UNSUPPORTED'
```

Then, run the following command:

```
export NODE_OPTIONS=--openssl-legacy-provider
```

API Requests

To test the back-end routes, you will need to generate a token and install Postman to make API requests.

Postman Installation

- Go to the Postman website at <https://www.postman.com/downloads/>.
- Click the "Download" button for the last version of Postman.
- Once the download is complete, open the downloaded file to start the installation process.
- Once the installation is complete, open Postman and start using it to make API requests. To make request a token is required. In the next section it will be explain how to generate it.

Create token

Open the URL for the front-end.

Click on the `Create a New Account` button to create a new user:

E-mail


Password

☐ Remember me

FORGOT YOUR PASSWORD?

LOGIN →

Create an account

Save time and create account with: 

First name *

Name

Last name *

Surname

Email *

name@gmail.com

Confirm email *

name@gmail.com

Password *

Confirm password *

To see all the routes in the project run the following command:

```
adonis list:routes
```

This command display a list of all registered routes in the project:

Method	Route	Handler	Middleware	Name
HEAD, GET	/auth/login/:provider	Auth/SocialAuthController.redirect		auth.social.login
HEAD, GET	/auth/callback/:provider	Auth/SocialAuthController.callback		auth.social.callback
POST	/auth/session	Auth/SessionController.create		auth.session.login
DELETE	/auth/session	Auth/SessionController.destroy		auth.session.logout
HEAD, GET	/auth/session	Auth/SessionController.show		auth.session.check
POST	/auth/register	Auth/SessionController.register		auth.session.register
HEAD, GET	/ateliers	AteliersController.index	auth	ateliers.index
POST	/ateliers	AteliersController.store	auth	ateliers.store
HEAD, GET	/ateliers/:id	AteliersController.show	auth	ateliers.show
PUT, PATCH	/ateliers/:id	AteliersController.update	auth	ateliers.update
DELETE	/ateliers/:id	AteliersController.destroy	auth	ateliers.destroy
HEAD, GET	/contacts	ContactsController.index	auth	contacts.index

Make API request in Postman

To create API requests through Postman you need to set up the following information:

- Type of request: You can choose between GET, POST, PUT, and DELETE depend on the request you are doing.
- URL: It is the endpoint URL where you want to send the API request. To get the routes check the List Routes section [List-Routes](#).
- Authentication: It necessary to provide an API token. Use the Bearer Token type to authenticate your request. To know how to get the token check Create Token section [Create Token](#).

GET

http://127.0.0.1:3333/measurements

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

Type

Bearer Token

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collab variables. Learn more about [variables](#)

The authorization header will be automatically generated when you send the request. Learn more about [authorization](#)

Token

Token

- Body: If you are making a POST, PUT, or PATCH request, you will need to include a request body in the form of JSON, XML, or other data format. The body can contain information such as data to be inserted or updated. Following you can see an example of the data in JSON format:

Params

Authorization

Headers (10)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

3

...."avatar": "https://static.wixstatic.com/media/e89179_ebce2d3354f74bf6a12144ddad232000.00_0.01,enc_auto/e89179_ebce2d3354f74bf6a12144ddad232000.png",

4

...."rating": 5,

5

...."description": "Fashion design was my gateway to the creative industry which led to my eye for detail I approach every project with an innovative mind and aim to provide service including dress making, pattern making and grading, digital design and per

```

6 ... "gallery": "[\"https://static.wixstatic.com/media/e89179_1c956a84f84b4de89dfc195bd3b91f.jpg\", \"https://static.wixstatic.com/media/e89179_e731fd1208a9443bbad76ecc9dd581f7~mv2_e89179_e731fd1208a9443bbad76ecc9dd581f7~mv2_d_1706_2560_s_2.jpg\", \"https://static.wixstatic.com/media/e89179_5e6d8583b33541edab05ab2684c5fc97~mv2_d_3840_5760_s_4_2.jpg/v1/fill/w_446,h_446,e89179_5e6d8583b33541edab05ab2684c5fc97~mv2_d_3840_5760_s_4_2.jpg\", \"https://static.wixstatic.com/media/e89179_6228dd58962e47e3a755c2bc576ca0e7~mv2.jpg/v1/fill/w_238,h_238,q_90/e89179_6228dd58962e47e3a755c2bc576ca0e7~mv2.jpg\", \"https://static.wixstatic.com/media/e89179_f6ca7fcc488840fe9d4ced444ff6d268~mv2_d_1365_2048_s_2.jpg/v1/fill/w_269,h_267,q_90/e89179_f6ca7fcc488840fe9d4ced444ff6d268~mv2_d_1365_2048_s_2.jpg\", \"https://static.wixstatic.com/media/e89179_f74499b76d184d91908aa41c7727bb94~mv2.jpg/v1/fill/w_446,h_446,q_90/e89179_f74499b76d184d91908aa41c7727bb94~mv2.jpg\"]]"

```

Setting up DNS

Check the required domain name [Meyd.IT](#) is available on Go daddy website.

1. Log in to the GoDaddy account and navigate to the "My Products" page. Please note that [@ Susan Hansen](#) is the only one who has access to DNS Settings, please contact her for any DNS modification.
2. Click on the "DNS" button next to the domain name you want to set up.
3. On the DNS management page, you can add or modify your DNS settings by clicking on the "Add" or "Edit" buttons next to the appropriate fields. If the domain name is available purchase it from the go daddy website.
4. Set Up the Public DNS Service on GoDaddy website.
5. Setup A record in Public DNS to point to the public IP of the website host.

To set up an A record, which maps a domain name to an IP address, click on the "Add" button next to the "A (Host)" section. In the "Host" field, enter the domain name you want to set up "Meyd.it". In the "Points to" field, enter the IP address you want to map the domain name to. Click on the "Save" button to save your changes.

The "A" stands for "address" and this is the most fundamental type of DNS record: **it indicates the IP address of a given domain.**

A			
Type	Domain Name	TTL	Address
A	Meyd.it	1800	151.101.1.195 Check IP Blacklist Owner: Fastly Inc. WHOIS AS54113

6. It can now be verified on dns checker tool as shown in below screenshot.




A nameserver is a server in the DNS that **translates domain names into IP addresses**. Nameservers store and organize DNS records, each of which pairs a domain with one or more IP addresses. These servers act as the bridge between domain names, which we humans can remember, with IP addresses, which computers can process.

NS			
Type	Domain Name	TTL	Canonical Name
NS	Meyd.it	3600	ns81.domaincontrol.com. (97.74.101.32 Check IP Blacklist) Owner: GoDaddy.com LLC WHOIS AS44273
NS	Meyd.it	3600	ns82.domaincontrol.com. (173.201.69.32 Check IP Blacklist) Owner: GoDaddy.com LLC WHOIS AS44273

7. Setup MX Record for sending and receiving email for [meyd.it](#) domain.


An MX record, or mail exchange record, is a DNS record that **routes emails to specified mail servers**. MX records essentially point to the IP addresses of a mail server's domain.

MX				
Type	Domain Name	TTL	Preference	Address
MX	meyd.it	3600	30	mx4.email-hosting.net.au. (103.252.152.46 Check IP Blacklist) Owner: Synerqy Wholesale Pty Ltd WHOIS AS45638

MX	meyd.it	3600	10	mx1.email-hosting.net.au.(103.252.153.1 ↗ Check IP Blacklist) Owner: Synergy Wholesale Pty Ltd  WHOIS AS45638
MX	meyd.it	3600	10	mx2.email-hosting.net.au.(103.252.152.36 ↗ Check IP Blacklist) Owner: Synergy Wholesale Pty Ltd  WHOIS AS45638
MX	meyd.it	3600	20	mx3.email-hosting.net.au.(103.252.153.16 ↗ Check IP Blacklist) Owner: Synergy Wholesale Pty Ltd  WHOIS AS45638

8. Setup TXT Record as and when required for domain ownership verification.

By uploading a new TXT record with specific information included, or editing the current TXT record, an administrator can prove they control that domain. The tool or cloud provider can check the TXT record and see that it has been changed as requested.

TXT 			
Type	Domain Name	TTL	Record
TXT	meyd.it	3600	google-site-verification =xjdQ7LHOn9DSFu5Kss lRANm_6oRJjn6KWEc8- __lifU
TXT	meyd.it	3600	NETORGFT3959574.on microsoft.com
TXT	meyd.it	3600	v=spf1 +a +mx +includ e:spf.email-hosting.net. au ~all
TXT	meyd.it	3600	google-site-verification =ZuVYPKkHWSctcicMS 06RxzHLfLkmvowJAE71 BH9ffCI