ISSN: 2754-6705

Journal of Mathematical & Computer Applications

SCIENTIFIC Research and Community

Review Article

Open Access

# Automating The Deployment of MERN Stack on AWS App Runner Using AWS Code Pipeline

**Aauti**

USA

**ABSTRACT**

This paper explores the deployment of web applications utilizing AWS App Runner, a managed platform that simplifies the deployment process by allowing developers to select runtimes and deploy applications without the need for extensive configuration. It focuses on the advantages of using Docker runtime for API execution and web application deployment, highlighting the ease of dockerizing applications and the seamless integration with Amazon Elastic Container Registry (ECR). ECR provides a fully-managed Docker container registry, facilitating the efficient storage, management, and deployment of Docker container images. Additionally, the paper discusses the role of AWS CodeCommit as a secure, scalable, managed source control service for hosting private Git repositories, emphasizing its significance in the deployment pipeline. The combined use of these AWS services offers a streamlined, cost-effective solution for deploying scalable and secure web applications directly from source code or container images to the AWS Cloud, thereby enhancing the deployment process for developers.

***Corresponding author**

Aauti, USA.

**Received:** January 17, 2024; **Accepted:** January 23, 2024, **Published:** January 30, 2024

**Keywords:** MERN, AWS, Cloud Computing, Programming, Software Development

In this paper, we explore the adoption and implementation of AWS App Runner, a managed platform service by Amazon Web Services, tailored for streamlined application deployment. AWS App Runner stands out for its capability to simplify the deployment process by allowing developers to choose the desired runtime environment, such as NodeJS, or leverage Docker to containerize and deploy applications, particularly focusing on the MERN stack. This service eliminates the complexities typically associated with configuration and management, thereby facilitating the deployment of APIs and web applications directly from source code or container images. Additionally, it ensures scalability and security within the AWS Cloud ecosystem.

The utility of AWS App Runner extends to its seamless integration with Docker, where Docker images, including those from Amazon's Elastic Container Registry (ECR), can be effortlessly deployed. Our study delineates a comprehensive automation strategy encompassing the entire deployment lifecycle of a MERN Stack application using Docker within the AWS App Runner environment. This encompasses leveraging AWS CloudFormation for resource provisioning and adopting continuous integration and continuous deployment (CI/CD) methodologies to achieve efficient and automated application deployment. This paper aims to provide a detailed exploration of the end-to-end automation process, highlighting the practical and theoretical aspects of deploying a containerized MERN Stack on AWS App Runner, thereby demonstrating the efficiency and efficacy of managed platform services in modern cloud environments.

- Prerequisites
- Example Project
- Setup a MongoDB Atlas
- Build For Production
- Externalize Environment Variables
- Dockerize the project
- Running WebApp on Docker
- Creating ECR with CloudFormation
- Pushing Docker Image to ECR
- Deploy CF Template through CLI
- Setup CodeCommit Repos
- Create a ServiceRole for CodePipeline
- Setup CI/CD With AWS CodePipeline for ECR Deployment
- Setup CI/CD With AWS CodePipeline For AppRunner WebApp
- Testing the WebApp
- Summary
- Conclusion

## Prerequisites

For individuals embarking on their journey in web development, it is recommended to consult the following resource for a comprehensive guide on developing and constructing applications using the MERN stack. This material serves as an essential foundation for beginners in the field.

• How To Develop and Build MERN Stack (https://medium.com/bb-tutorials-and-thoughts/how-to-develop-and-build-mern-stack-9a7a1099624)

## Docker Essentials

You need to understand Docker concepts such as creating images, container management, etc. Below are some of the links that you can understand about Docker if you are new.

- Docker Docs (https://docs.docker.com/)
- Docker – A Beginners Guide (https://medium.com/bb-tutorials-and-thoughts/docker-a-beginners-guide-to-dockerfile-with-a-sample-project-6c1ac1f17490)
- Docker – Image Creation and Management (https://medium.com/bb-tutorials-and-thoughts/docker-image-creation-and-management-9d91e4c277b1)
- Docker – Container Management (https://medium.com/bb-tutorials-and-thoughts/docker-container-management-with-examples-c280906158a8)
- Understanding Docker Volumes (https://medium.com/bb-tutorials-and-thoughts/understanding-docker-volumes-with-an-example-d898cb5e40d7)

## AWS Prerequisites

Amazon Web Services (AWS), recognized as a pioneer in the cloud computing domain, offers an extensive portfolio of over 200 services. It is crucial for users to understand and select the appropriate AWS services that align with their specific requirements. If you are new to AWS or just getting started you can see the following article.

- How To Get Started with AWS (https://medium.com/bb-tutorials-and-thoughts/how-to-get-started-with-aws-9731a4f855a7)

## Example Project

Here is an example of a simple tasks application that creates, retrieves, edits, and deletes tasks. We actually run the API on the NodeJS server, and you can use MongoDB to save all these tasks.

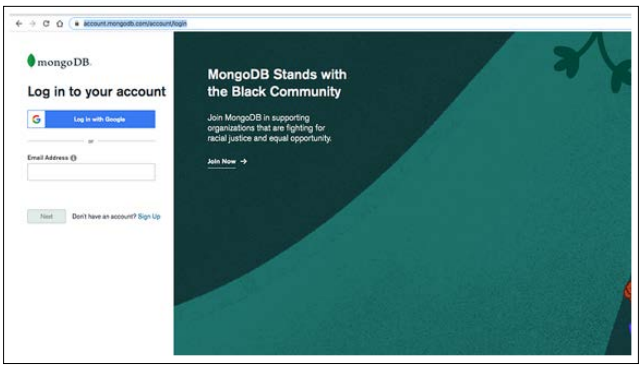

MERN Stack Example

Here is a GitHub link to this project. You can clone it and run it on your machine.

```
// clone the project
git clone https://github.com/bbachi/mern-stack-example

// React Code
cd ui
npm install
npm start

// API code
cd api
npm install
npm run dev
```

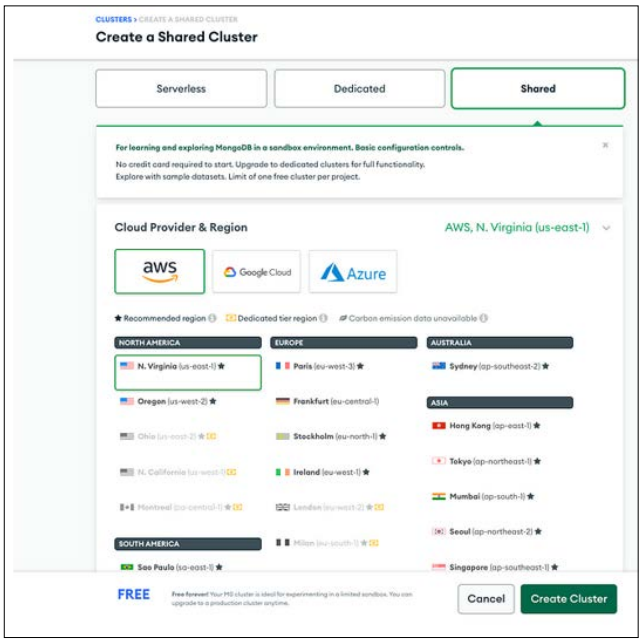
## Setup a MongoDB Atlas

The core of MongoDB Cloud is MongoDB Atlas, a fully managed cloud database for modern applications. Atlas is the best way to run MongoDB, the leading modern database. There are two ways to deploy MongoDB on AWS and you can check them here on this page. We are using a fully-managed MongoDB Cluster for this post. Let's create your MongoDB Account here. You can either log in with any of your Gmail accounts or you can provide any other email address to create the account.



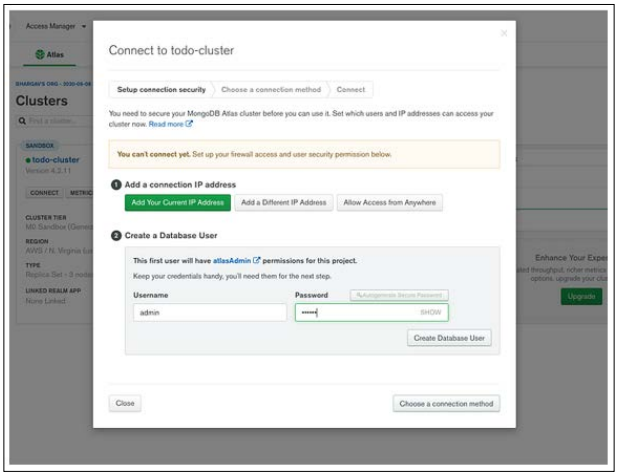

MongoDB Login

Once you log in with your account you will see the dashboard below where you can create clusters. Let's create a cluster called todo-cluster by clicking on build a cluster and selecting all the details below. Make sure you select AWS Cloud.



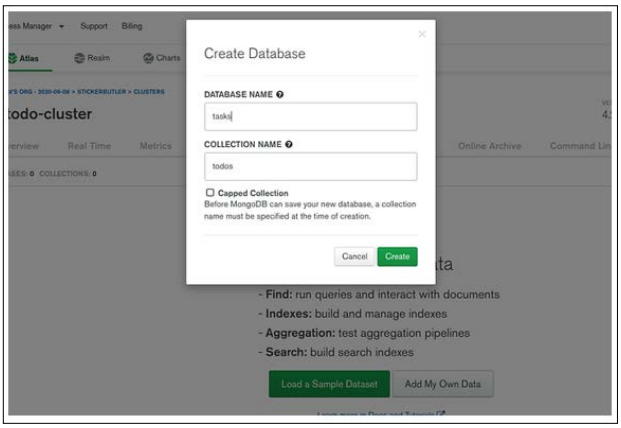

MongoDB Pricing

Make sure you select the Cloud Environment since we are deploying this on AWS Cloud. You can click on the connect button to see the details about connecting to the cluster. You need to create a user and Allow Access from anywhere for now.



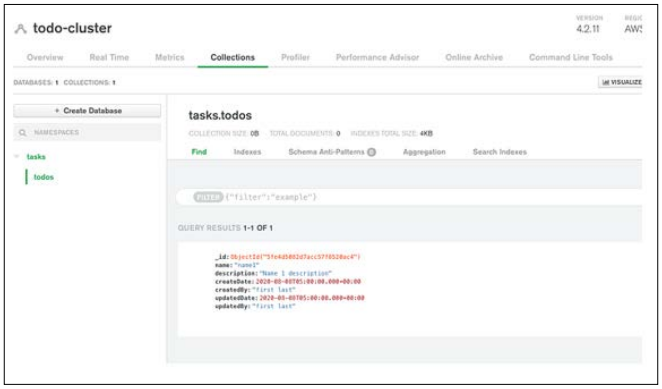

Connect to Cluster

You can see three ways of connecting to the cluster on the next screen. We have created a cluster and it's time to create a database. Click on the collections to create a new database as below. I have given a database name as tasks and the collection name is todos.



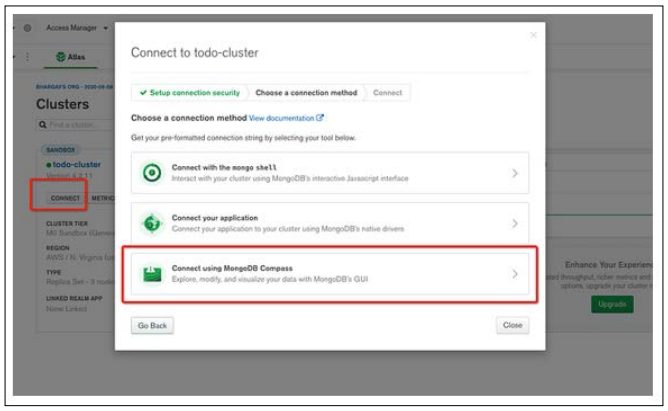

Creating a Database

Let's insert the first document into the collection by clicking the button insert document. We have seen three ways we can connect to this cluster and read the collections. Let's connect to the database with Mongo Compass.




Creating a Collection

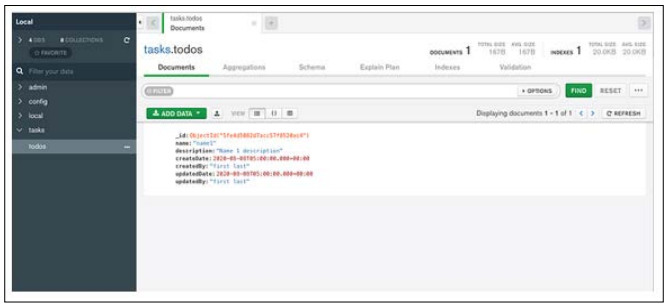
The first thing we need to do is to download and install Mongo Compass from this link. Let's get a connection string from the Atlas dashboard as below.




Connection Ways

You can see the same collection in the MongoDB Compass as well. Here is the connection string that you can connect to MongoDB.

mongodb+srv://admin123:admin123@todo-cluster.zpikr.mongodb.net/?retryWrites=true&w=majority




Mongo Compass UI

**Build for Production**

Numerous approaches exist for constructing a MERN Stack for production deployment, with the optimal strategy varying based on the specific use case or deployment environment. This paper delineates various methodologies for preparing the MERN Stack for production use.

• How to Build MERN Stack for Production (https://medium.com/bb-tutorials-and-thoughts/how-to-build-mern-stack-for-production-1462e70a35cb)

**Externalize Environment Variables**

Reading environment variables is one of the most common things that we do when we are building apps. It doesn't matter whether you are developing front end app or back-end API you have so many variables that should be outside of your application source code that makes your app or API more configurable. For example, if you want to hide logger statements in production or do something else based on the environment you can pass this as an environment variable. If you want to change later all you need to change is in one place.

• Reading Environment Variables in NodeJS api (https://medium.com/bb-tutorials-and-thoughts/reading-environment-variables-in-nodejs-rest-api-e75bb04b813d)

When it comes to this application, there are two environment variables, one is the Mongo Connection string, and another one is PORT.

```
PORT=80
MONGO_CONNECTION_STRING=mongodb+srv://admin123:admin123@todo-cluster.zpikr.mongodb.net/?retryWrites=true&w=majority
```

You must put these in the webpack.config.js file so that these values are used when we dockerize the app for production.

• Webpack.config.js file (https://gist.github.com/bbachi/aa25aec5b82320d28cc5ee137bb8b8cf#file-webpack-config-js)

**Dockerize the Webapp**

Amazon EKS is a managed service that makes it easy for you to run Kubernetes on AWS. The first thing you need to do is to dockerize your project.

We use multi-stage builds for efficient docker images. Building efficient Docker images are very important for faster downloads and lesser surface attacks. In this multi-stage build, building a React app and putting those static assets in the build folder is the first step. The second step involves building the API. Finally, the third step involves taking those static build files and API build and serving the React static files through the API server.

We need to update the server.js file in the NodeJS API to let Express know about the React static assets and send the index. html as a default route. Here is the updated server.js file. Notice the line numbers 41 and 20.

• Server.js file (https://gist.github.com/bbachi/e828985a85cfb9da08164afa88549211#file-server-js)

Let's build an image with the Dockerfile. Here are the things we need for building an image.

In constructing a production-ready application using the MERN Stack, the process begins with a base image of node:14-slim. Initially, both package.json files—one for the Node.js server and the other for the React UI—are copied into the Docker file system, with dependencies installed to enhance build speed for subsequent changes. This preemptive step prevents the redundancy of reinstalling dependencies with each source modification. Following this, all source files are copied, and dependencies installed, culminating in the execution of npm run build to generate the React application assets within a 'build' folder inside the 'ui' directory. The second stage also utilizes the node: 14-slim base image, focusing on the Node.js environment by copying its package.json into an './api' directory, installing necessary dependencies, and incorporating the server.js file into this directory. The final stage combines the elements, starting again with the node:14-slim image, to amalgamate the built UI and API files, concluding with the command node api.bundle.js to run the bundled server application, thereby streamlining the deployment of the MERN Stack for production.

Here is the Complete Dockerfile link where you can run on your machine.

• Docker file (https://gist.github.com/bbachi/06eecfc6c956d01c99180523c2677c15#file-dockerfile)

Let's build the image with the following command.

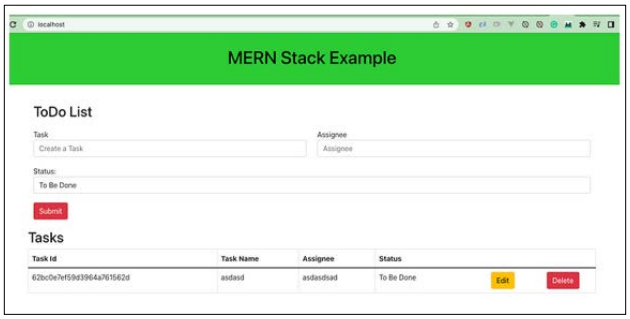
```
// build the image
docker build -t mern-image .

// check the images
docker images
```

**Running the Webapp on Docker**

Once the Docker image is built. You can run the image with the following command.

```
// run the container
docker run -d -p 80:80 --name mern-stack mern-image

// list the container
docker ps

// logs
docker logs mern-stack

// exec into running container
docker exec -it mern-stack /bin/sh
```

You can access the application on the web at this address http://localhost




Example Project

**Creating ECR with Cloud Formation**

First, you need to understand the anatomy of the CloudFormation template. We can't go through everything here you can look at the AWS Cloudformation docs here.

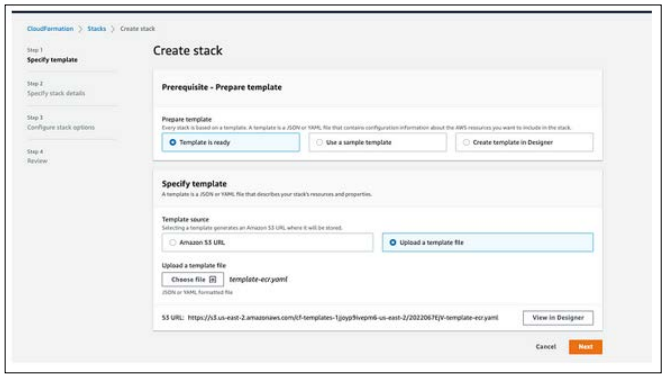
AWSTemplateFormatVersion: "version date"
Description: String
Metadata: Template Metadata
Parameters: Set of Parameters
Rules: Set of Rules
Mappings: Set of Mappings
Conditions: Set of Conditions
Transform: Set of Transforms
Resources: Set of Resources
Outputs: Set of Outputs

The only required one is the Resources of all these options in the template file. Below is the template YAML file with which we are creating the ECR repository through CloudFormation. The first one is the version and description. The version has only one value and in the description, you can put anything about

your repo or deployment. Since it's an ECR Repository, I have given the following description. The next main thing is the Resources section. Since we are creating only one resource which is AWS AppRunner, I have added one resource called ECRRepo. You can name it anything you want and the type is obviously AWS::ECR::Repository. The main thing here is adding a policy text where the users specified only can push the image into the repository. In production, you should create a role here.

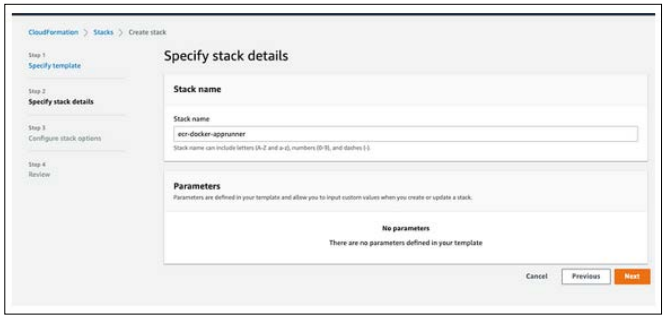
• ecr-template.yaml (https://gist.github.com/bbachi/f47addffb396297502fd8789b75d7cdc#file-template-ecr-yaml)

The output section contains the ARN of the ECR repository. Let's create this stack through AWS Console. You can do it either console or AWS CLI. You can click on the Create Stack button. On the next screen, you must upload the above YAML file by selecting the second option.



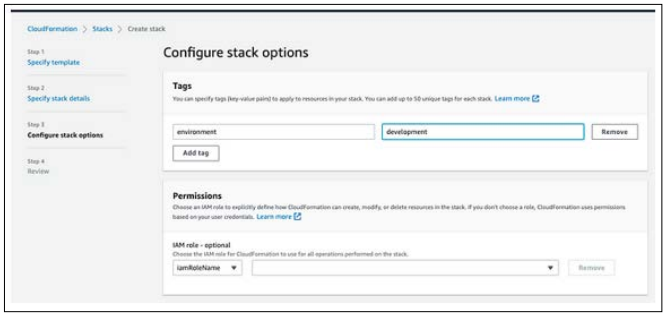

Creating a Stack

Let's give a stack name on the next screen.



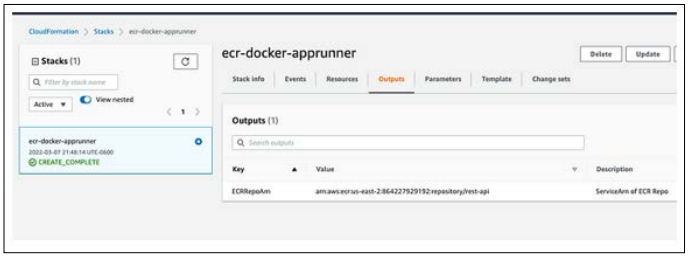

Stack Details

Configure the Stack options on the below screen.



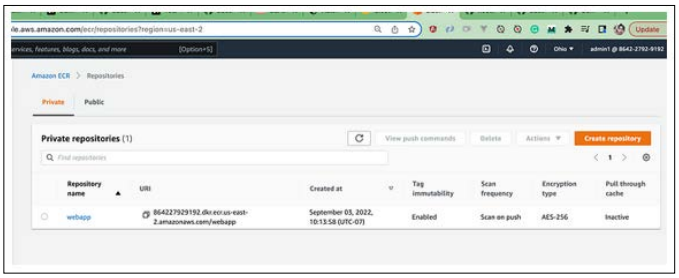

Configure the Stack

You can see the output ARN in the outputs section.




Output ARN

Let's go and check the ECR console to see if this repository is created or not.



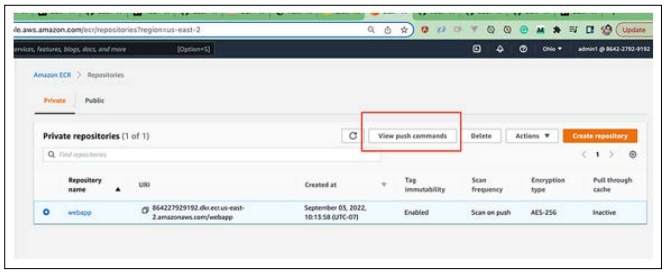

ECR Console

**Pushing Docker Image to ECR**

We have created an ECR repository in the above section. Let's create a docker image from the example project section above with the following command.

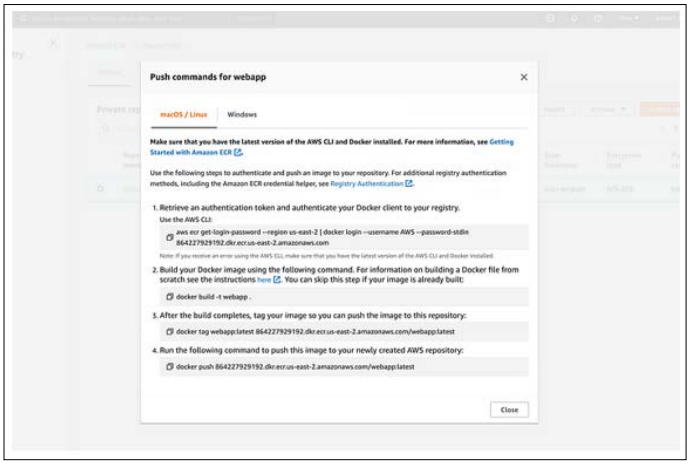
docker build -t webapp.

You can view further instructions after creating the Docker image in the top right corner.




Viewing Push Commands

You should authenticate first, then tag and finally push the docker image. Let's follow these commands.




Push Commands

You can tag and push the image with the following command.

```
docker tag webapp:latest 864227929192.dkr.ecr.us-east-2.amazonaws.com/webapp:v1

docker push 864227929192.dkr.ecr.us-east-2.amazonaws.com/webapp:v1
```

Once the image is pushed, you can view it on the ECR Console.

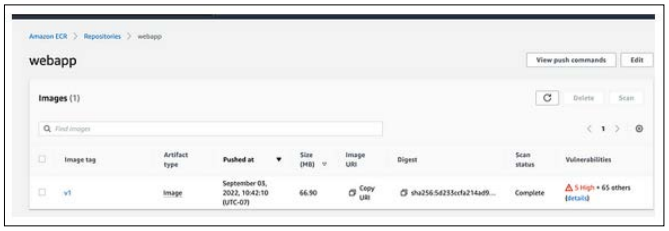



Image Pushed

**Creating CloudFormation Template**

We need to create multiple resources for the AppRunner Template, which we will go through one by one in this section. The initial ones are the version, description, and parameters. The version has only one value and in the description, you can put anything about your repo or deployment. Since it's a NodeJS REST API on App runner, I have given the following description. You can provide the parameters while deploying the template.

**Container Image:** This is the ECR Image URL. You can fetch it from the ECR Console.
**Environment:** The environment you want to deploy this stack such as dev, test, prod, etc.
**Welcome Message:** This is the environment variable you want to pass while deploying the template.
**Image Repository Type:** There are two types that the App Runner service accepts at this time of writing: ECR and ECR_PUBLIC

You can have conditions on your template so that we can execute something based on that. You can define that under the section called Conditions. Since the App Runner only accepts ECR and ECR_PUBLIC we are putting a condition for that. Here is a complete YAML file.

• ECR Template YAML (https://gist.github.com/bbachi/9cf4e6a88ca6414426f869b47d53ac34#file-template-yaml)

**AWS Cloud Formation Command's**

Here are some of the commands that you can run through AWS CLI to create and update the stack. You can explore more on AWS Docs.

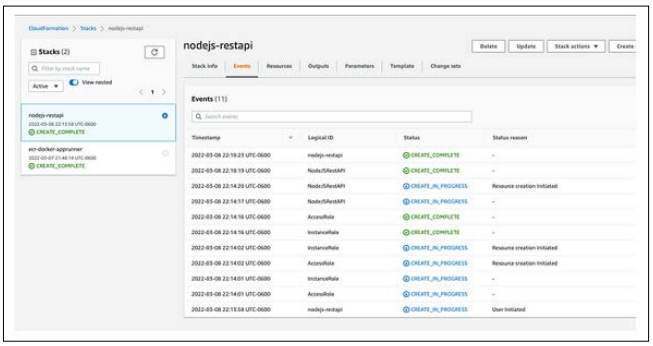
```
// create-stack
aws cloudformation create-stack
    --stack-name myteststack
    --template-body file:///home/testuser/mytemplate.json
     --parameters ParameterKey=Parm1,ParameterValue=test1 ParameterKey=Parm2,ParameterValue=test2

// listing stacks
aws cloudformation list-stacks

// describing stack
aws cloudformation describe-stack-events --stack-name <stack name>

// updating stack
aws cloudformation update-stack --stack-name mystack --template-url <>

// validating template
aws cloudformation validate-template
```

**Deploy CF Template Through CLI**

Let's create a resource through CLI with the following command. Make sure you update the command with your path of the file location.

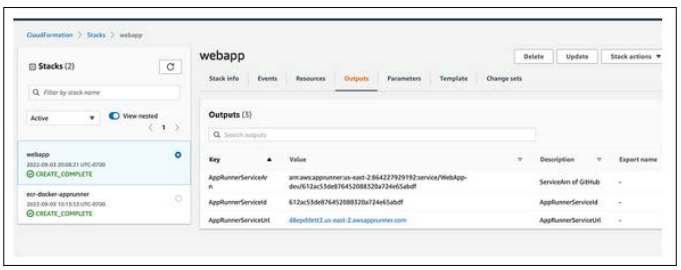
```
aws cloudformation create-stack
      --stack-name nodejs-restapi
      --template-body <file://file-location>
       --parameters ParameterKey=Environment,ParameterValue=dev      ParameterKey=WelcomeMessage,ParameterValue="Welcome from the CLI"
```

You can see the resources created on the respective screens below.

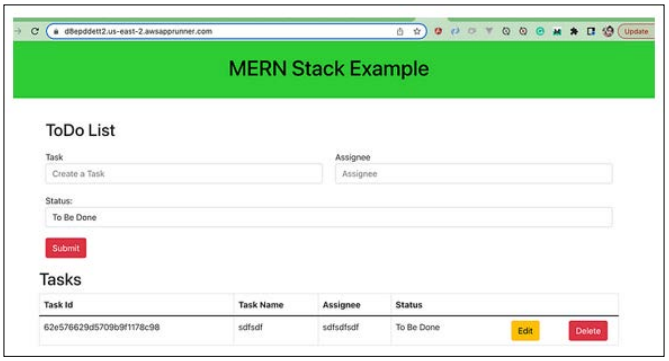



Cloud Formation Events

You can see the outputs listed on the AWS CloudFormation console.



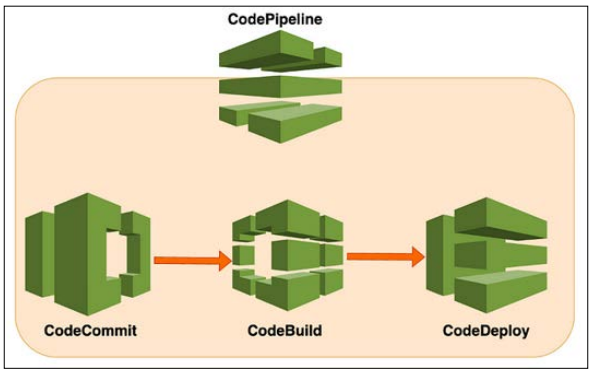

Template Created




Project Running Through AWS App Runner

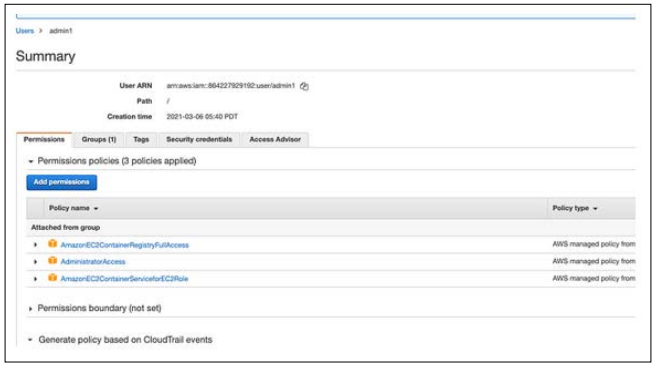
**Setup CodeCommit Repos**

The first step of CI/CD and automation flow is setting up the repositories on the AWS CodeCommit. AWS CodeCommit is a secure, highly scalable, managed source control service that hosts private Git repositories.



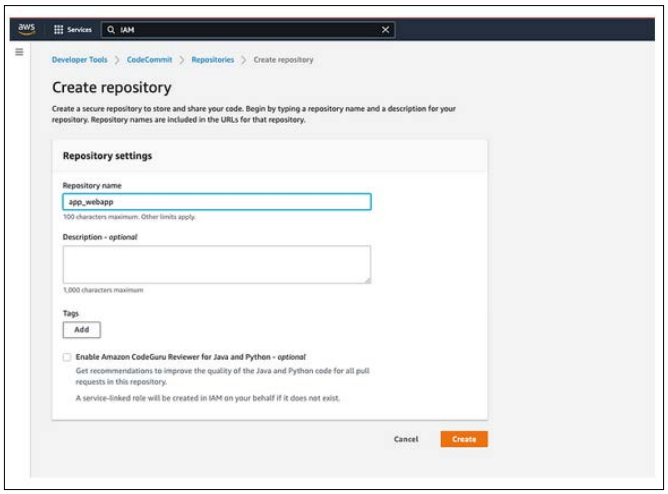

Code Commit Process

We are going to create two repositories: one for MERN Stack and another one for Cloudformation templates. Make sure you have Administrator access or access to the Codecommit to create the repos for the AWS user you created. I have AdministratorAccess that allows me to create repositories.



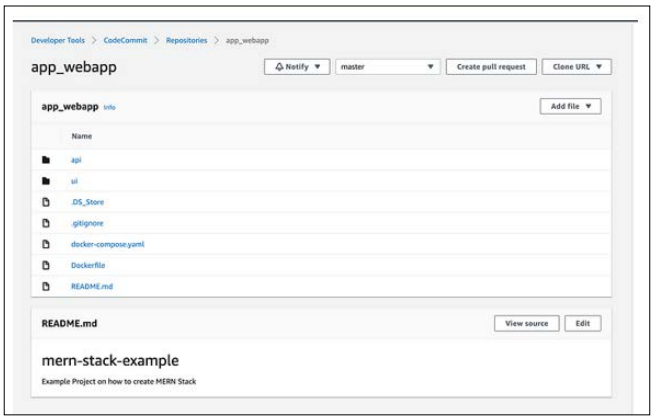

Permissions

You can go to the AWS CodeCommit and create a repository below.




Creating a Repository

Once created, you can clone it in different ways using HTTPS, SSH, and HTTPS (grc). You can use any of these methods to connect to this repository. If you want to use HTTPS, you must create Git credentials for your IAM user in the IAM section. Finally, I have created two repositories, and we can push the code later once we go through other important sections. You can copy all the source code from the example project in the above section to the AWS CodeCommit repository below.




Let's create another repository called app_cf_templates and push these two Yaml files into the repo. We can put these templates in this repo.

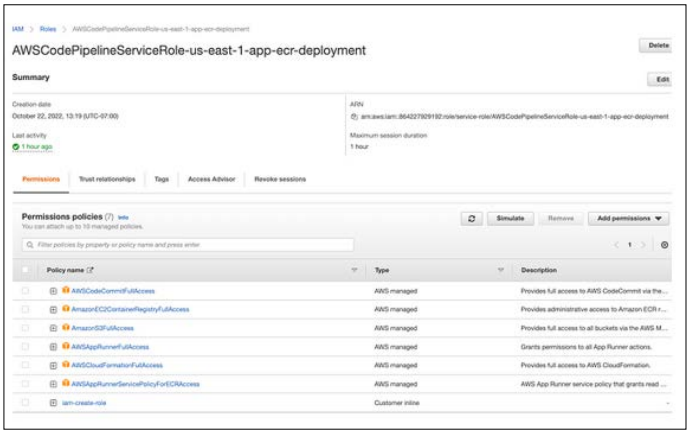
• Template-ecr.yaml (https://gist.github.com/bbachi/0d983547fd99f056ea21b880e73013f6#file-template-ecr-yaml)
• Teamplet-apprunner.yamlfile (https://gist.github.com/bbachi/3ebbee23c3c1fefb2fb7319f9f5f21a2#file-template-apprunner-yaml)

**Create a Service Role for Code Pipeline**

We have pushed the code to the AWS CodeCommit Repos in the above section. It's time to create a service role for the AWS Codepipeline to create the required resources when you run the pipeline.

Let's go to IAM dashboard Click on Roles and create role. Add these policies.

AmazonEC2ContainerRegistryFullAccess
AWSAppRunnerFullAccess
AWSCloudFormationFullAccess
AWSAppRunnerServicePolicyForECRAccess
AWSCodeCommitFullAccess
AmazonS3FullAccess

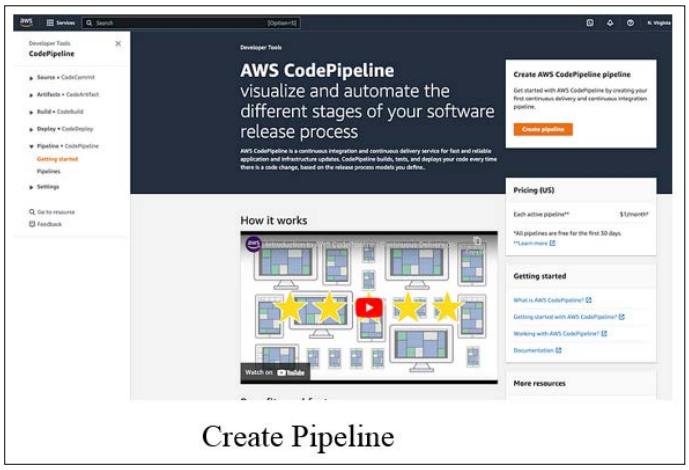



Service Role

You need to create one custom inline policy for AWS Codepipeline to create necessary roles while running the AWS CloudFormation. I have called this policy iam-create-role

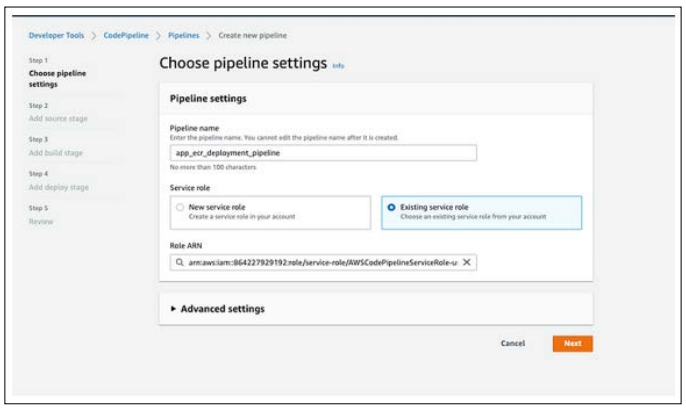
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:PassRole",
                "iam:DetachRolePolicy",
                "iam:DeleteRolePolicy",
                "iam:DeleteRole",
                "iam:CreateRole",
                "iam:AttachRolePolicy",
                "iam:PutRolePolicy"
            ],
            "Resource": "arn:aws:iam::*:role/*"
        }
    ]
}
```

**Setup CI/CD with AWS CodePipeline for ECR Deployment**
We have pushed the code to the AWS CodeCommit Repos in the above section. Access the CodePipeline dashboard below and click on the button Create pipeline.



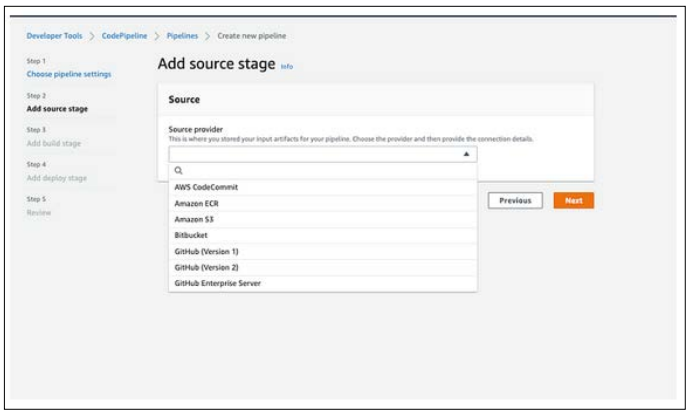

Create Pipeline

You can name the pipeline anything and you can select the servicerole created above or you can choose to create a new service role and the above policies to the role.



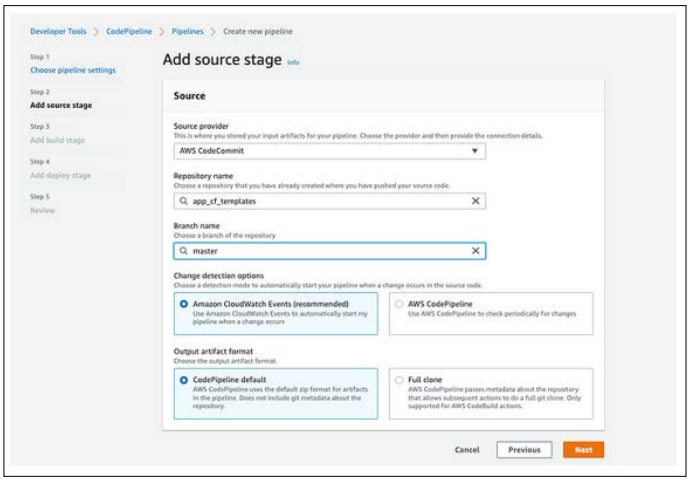

Pipeline Settings

You can click on the next button and choose the source where you read the code. I have selected AWS CodeCommit.



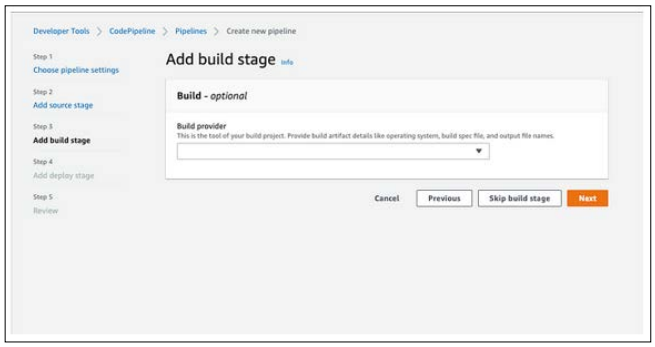

Selecting Source

Select the right repository and the branch you want to deploy.

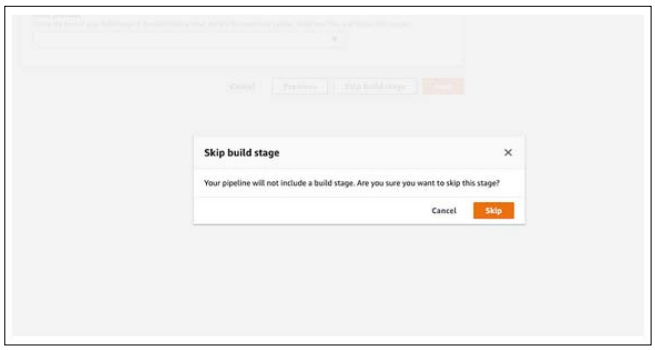



Source Stage

Since we are running the AWS Code Formation to create the ECR, we can skip the build stage.

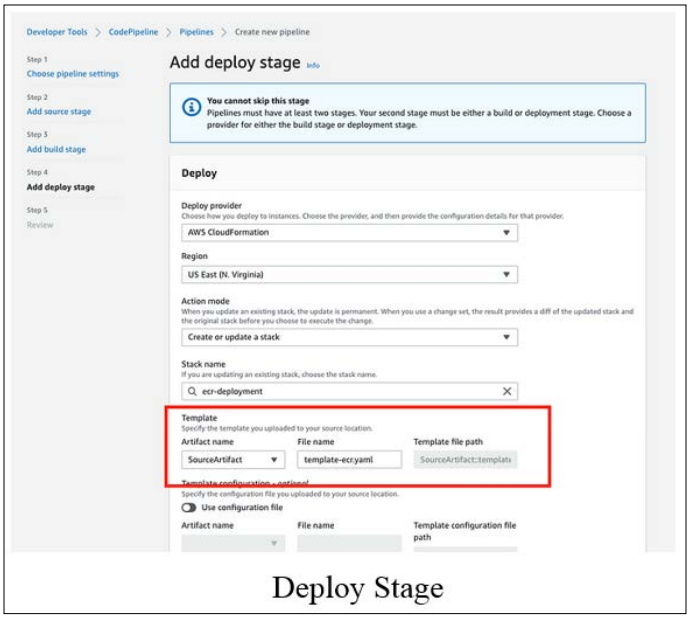



Skip Build Stage



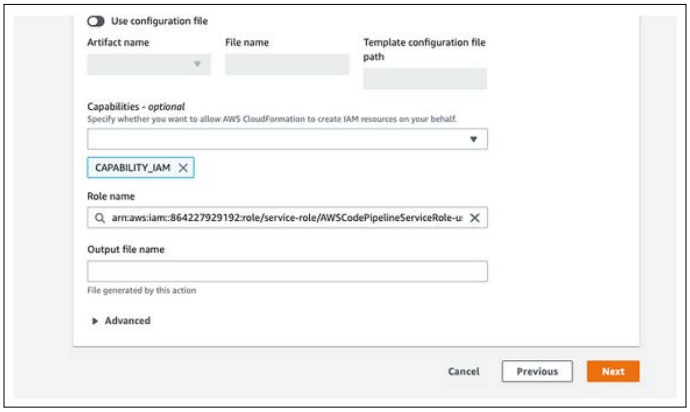

Skip Build Stage

You can't skip the deployment stage. I have chosen AWS CloudFormation as a Deployment provider. Make sure you select the right template from the repository.



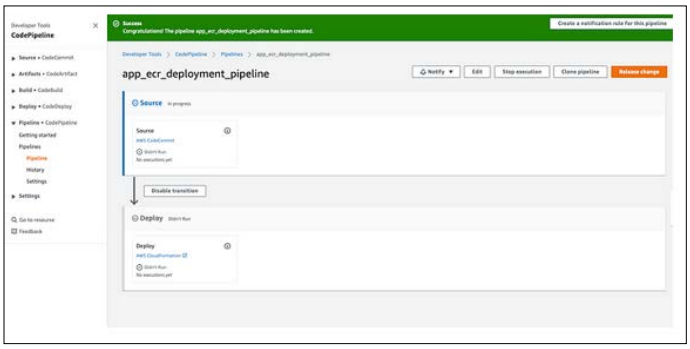

Deploy Stage

Choose the right role that we created above.



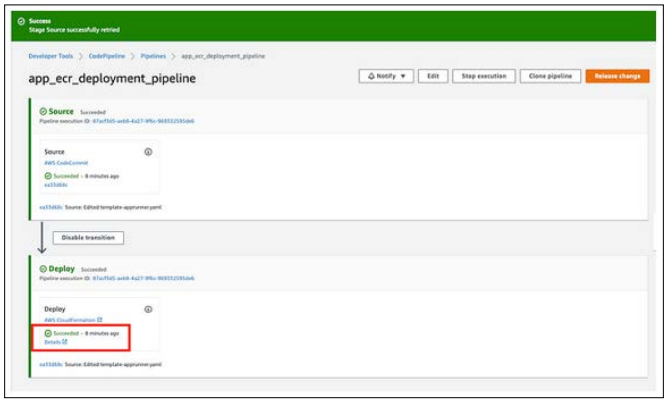

More Settings

Once you confirm everything and create a pipeline, you can see the pipeline created successfully and in progress status.



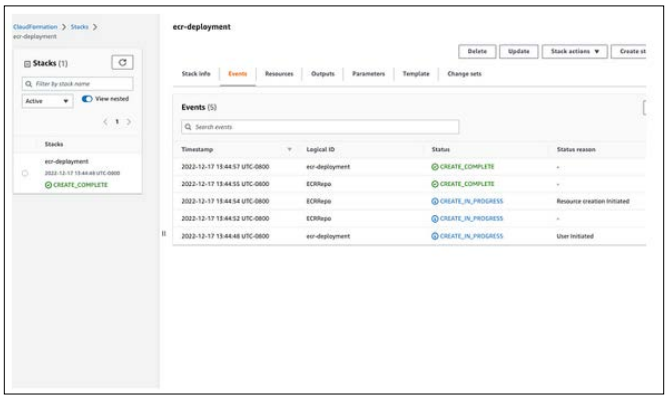

Pipeline in Progress

If everything is successful, you can see the pipeline successful below.



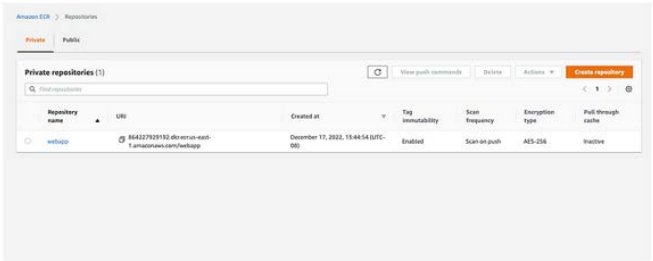

Pipeline Successful

You can click on the details link, and it takes you to the Cloud Formation page. You can see the events.



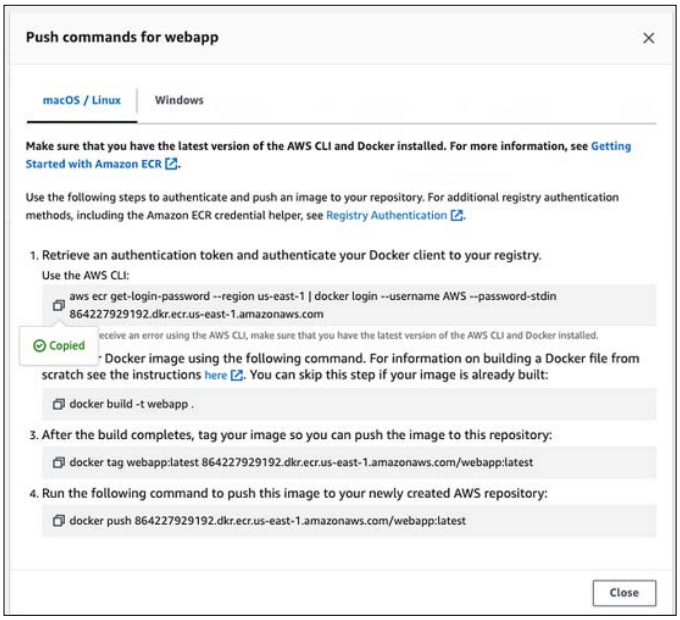

Cloud Formation

You can see the repository created below.




ECR Created

Let's push the Docker Image



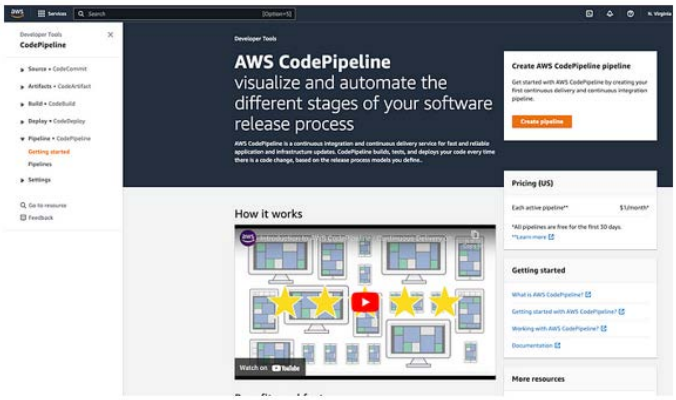

Let's run these commands to push the docker image to the ECR we just created. We will automate this step as well in future articles.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 864227929192.dkr.ecr.us-east-1.amazonaws.com

docker tag webapp:latest 864227929192.dkr.ecr.us-east-1.amazonaws.com/webapp:v1

docker push 864227929192.dkr.ecr.us-east-1.amazonaws.com/webapp:v1
```

You can see the image pushed to the ECR.


ECR Image

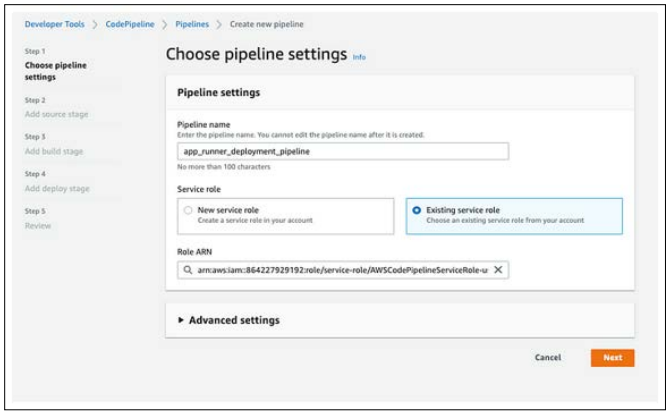
Setup CI/CD Pipeline with AWS Code Pipeline for AWS Apprunner Access the CodePipeline dashboard below and click on the button Create pipeline.




Create Pipeline

You can name the pipeline anything and you can select the service role created above or you can choose to create a new service role and the above policies to the role.

**Citation:** Aauti (2024) Automating The Deployment of MERN Stack on AWS App Runner Using AWS Code Pipeline. Journal of Mathematical & Computer Applications. SRC/JMCA-181. DOI: doi.org/10.47363/JMCA/2024(3)149

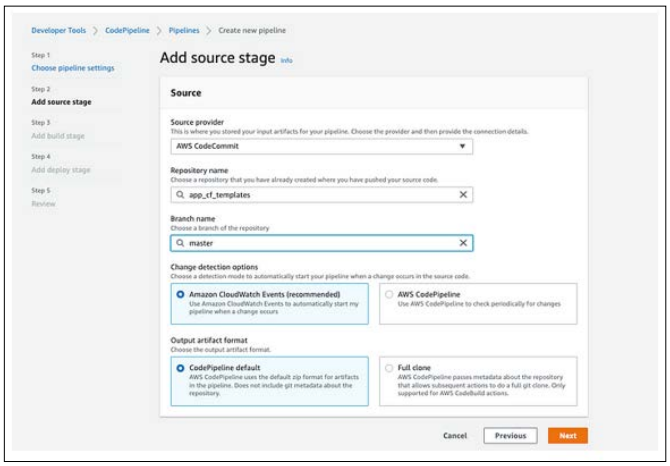



Code Pipeline

You can click on the next button and choose the source where you read the code. I have selected AWS CodeCommit.

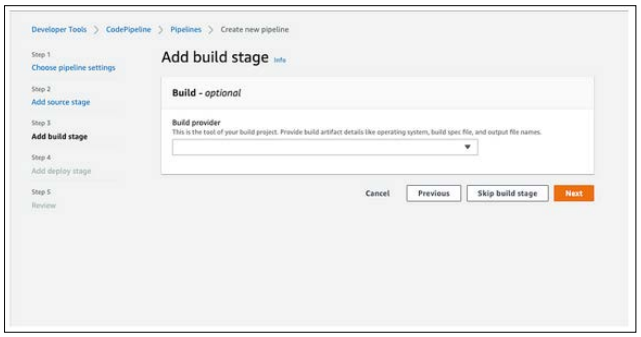

Selecting Source

Select the right repository and the branch you want to deploy.



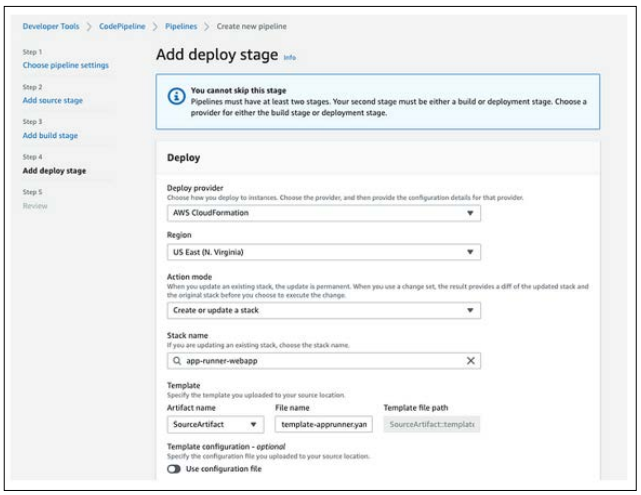

Source Stage

Since we are running the AWS Code Formation to create the AppRunner, we can skip the build stage. We can build the Docker image here and I will update that in future articles.



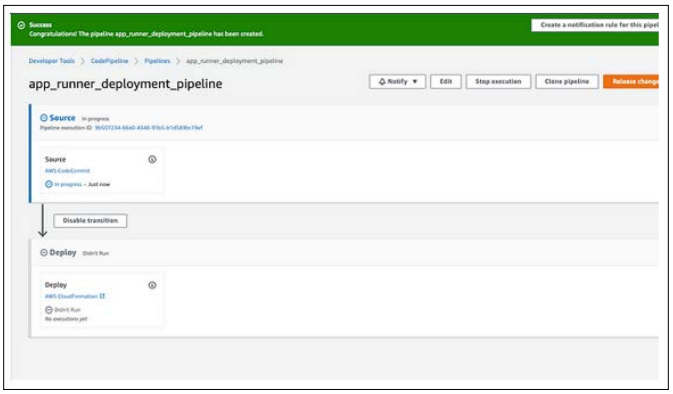

Skip Build Stage

You can't skip the deployment stage. I have chosen AWS CloudFormation as a Deployment provider. Make sure you select the right template from the repository.



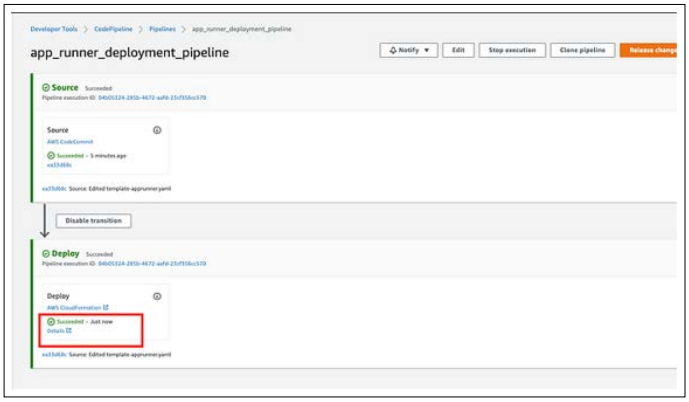

Deploy Stage

Once you confirm everything and create a pipeline, you can see the pipeline created successfully and in progress status.

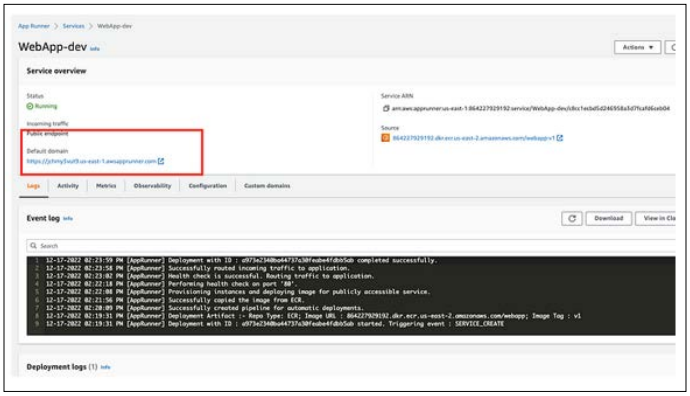



Pipeline in Progress

If everything is successful, you can see the pipeline successful below.

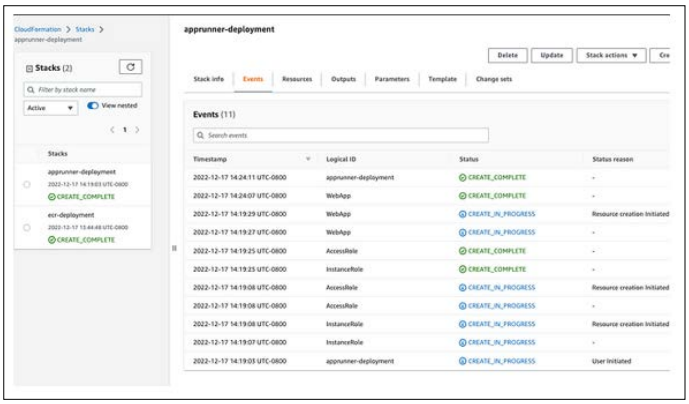



Pipeline Successful




App Runner Running Successfully

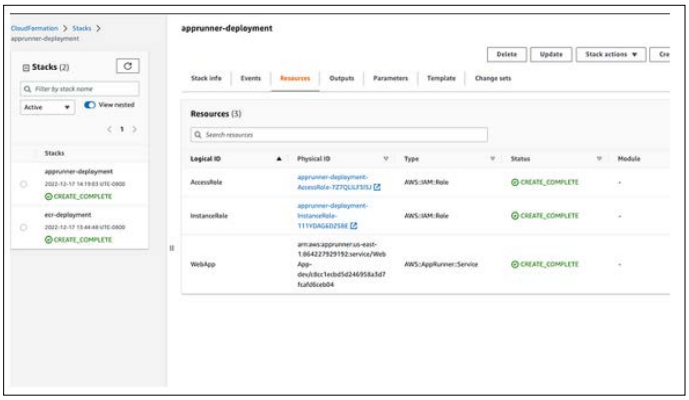
You can click on the details link and it takes you to the CloudFormation page. You can see the events.




Cloud Formation Events

## Testing the Webapp

We have created App Runner and ECR by running AWS CodePipeline in the above sections. You can click on the Resources section of CloudFormation [1-5].

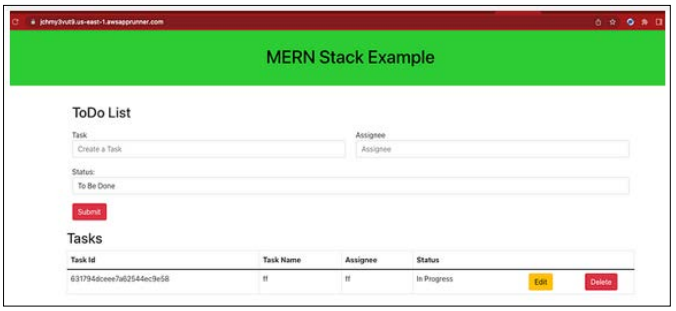



Resources Created

You can see the AppRunner Created and run successfully.


App Runner

You can test the webapp with the following URL.
https://jchmy3vut9.us-east-1.awsapprunner.com/




App Runner Running Successfully

## Summary

- If you want to deploy your application on the managed platform by selecting the runtime, AWS App Runner is the right choice.
- You can run the whole API with Docker runtime without any worry about the configuration from your side.
- You can dockerize the WebApp and deploy that in the Docker runtime. The Docker images can be pulled from ECR, etc.
- Amazon Elastic Container Registry (ECR) is a fully-managed Docker container registry that makes it easy for developers to store, manage, and deploy Docker container images.
- AWS App Runner is an AWS service that provides a fast, simple, and cost-effective way to deploy straight from source code or a container image directly to a scalable and secure web application in the AWS Cloud.
- AWS CodeCommit is a secure, highly scalable, managed source control service that hosts private Git repositories.

## Conclusion

In conclusion, AWS App Runner emerges as an exemplary choice for developers seeking to effortlessly deploy applications on a managed platform, emphasizing simplicity and minimal configuration requirements. By leveraging Docker runtime, it facilitates the smooth operation of APIs and the deployment of web applications directly from Docker images sourced from Amazon Elastic Container Registry (ECR). ECR enhances this ecosystem by offering a robust, managed Docker container registry, streamlining the storage, management, and deployment of container images. Furthermore, AWS App Runner's integration with AWS CodeCommit underscores its commitment to providing a secure, scalable, and efficient deployment pipeline. This synergy between AWS services simplifies the deployment process, from source code or container images to a fully scalable and secure web application, underscoring AWS's role in offering cost-effective,

rapid deployment solutions in the cloud.

## References

1. Official AWS Documentation https://docs.aws.amazon.com/.
2. AWS AppRunner Documentation https://docs.aws.amazon.com/apprunner/.
3. Bhargav Bachina (2022) How to develop and build MERN Stack https://medium.com/bb-tutorials-and-thoughts/how-to-develop-and-build-mern-stack-9a7a1099624.
4. Bhargav Bachina (2022) How to Build MERN Stack for Production https://medium.com/bb-tutorials-and-thoughts/how-to-build-mern-stack-for-production-1462e70a35cb.
5. Cloud Formation Documentation https://docs.aws.amazon.com/cloudformation/.