# GitHub OIDC Setup with AWS EKS

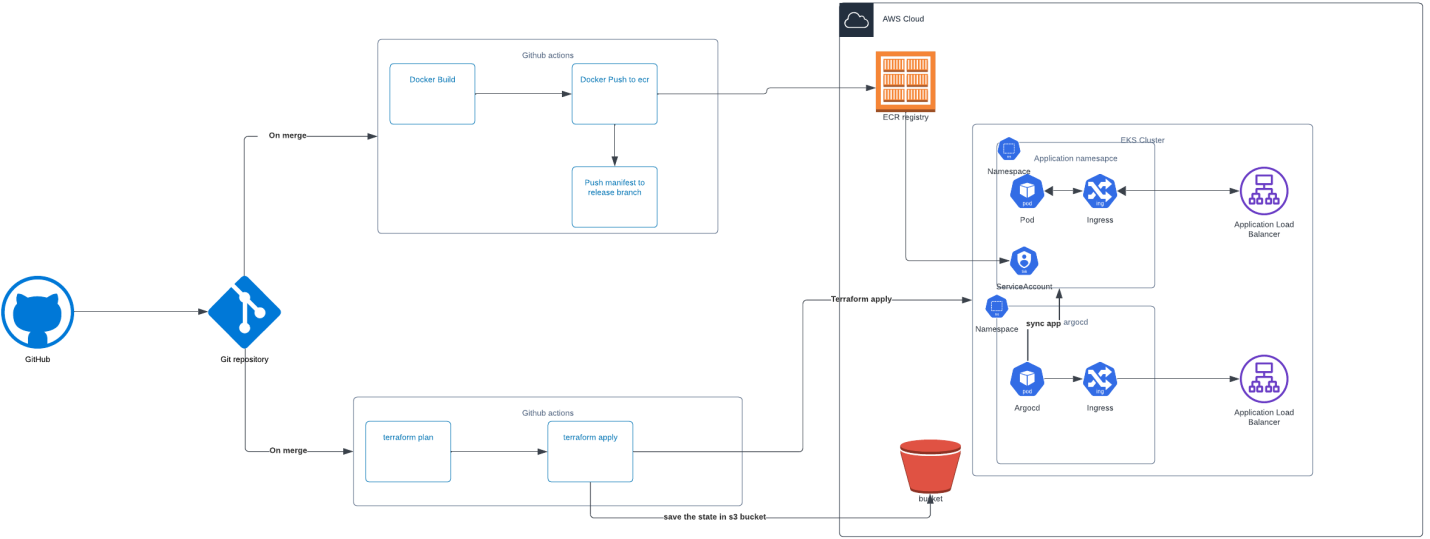
This guide outlines the steps to set up an environment that integrates GitHub Actions with AWS services using OIDC (OpenID Connect). This setup enables GitHub Actions to assume IAM roles to deploy resources in your AWS account securely.

## Table of Contents

1. [Prerequisites](#prerequisites)
2. [Setting Up the Base Infrastructure](#setting-up-the-base-infrastructure)
3. [AWS CloudFormation Steps](#aws-cloudformation-steps)
4. [Configure GitHub Secrets](#configure-github-secrets)
5. [Deploying with Terraform](#deploying-with-terraform)
6. [Login to EKS Cluster](#login-to-eks-cluster)
7. [Accessing ArgoCD](#accessing-argocd)
8. [Configuring ArgoCD Application](#configuring-argocd-application)

## Prerequisites

1. **GitHub Organization Account**: Ensure you have a GitHub organization account, as you will be integrating it with AWS services.
2. **AWS Account**: You need an AWS account with permissions to create IAM roles, CloudFormation stacks, S3 buckets, and EKS clusters.



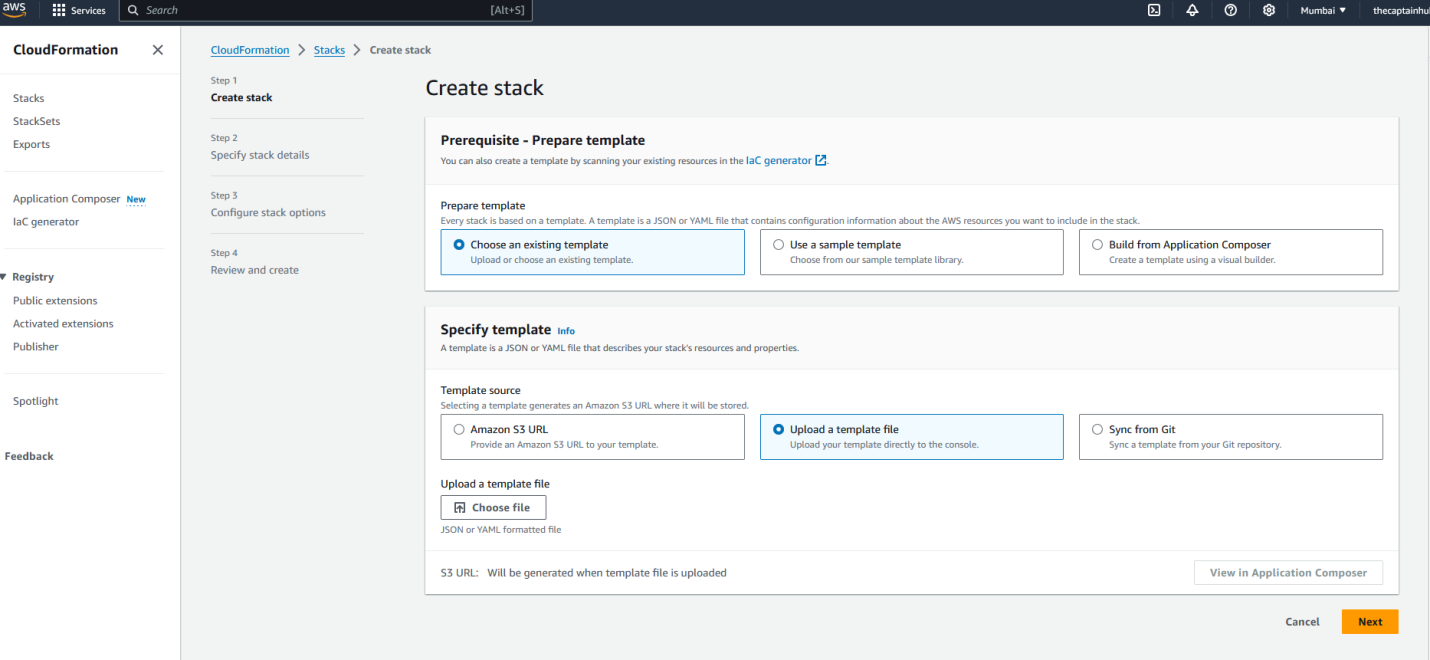
## Setting Up the Base Infrastructure

Before deploying the environment using Terraform, we first need to deploy some resources using CloudFormation.

1. **Clone the GitHub Repository**:
   * Open a terminal or command prompt.
   * Run the following command to clone your GitHub repository to your local machine:
   * git clone <repository-url>
   * Change directory into the cloned repository:
   * cd <repository-folder>
   * Ensure you have the latest changes from the main branch:
   * git pull origin main
2. **Create a New Branch for Deployment**:
   * This allows you to work on your deployment without affecting the main branch directly. Create and switch to a new branch:
   * git checkout -b deploy
3. **Locate the CloudFormation File**:
   * Inside the base-infra folder of your repository, find the CloudFormation template file (usually named something like cloudformation-template.yaml).

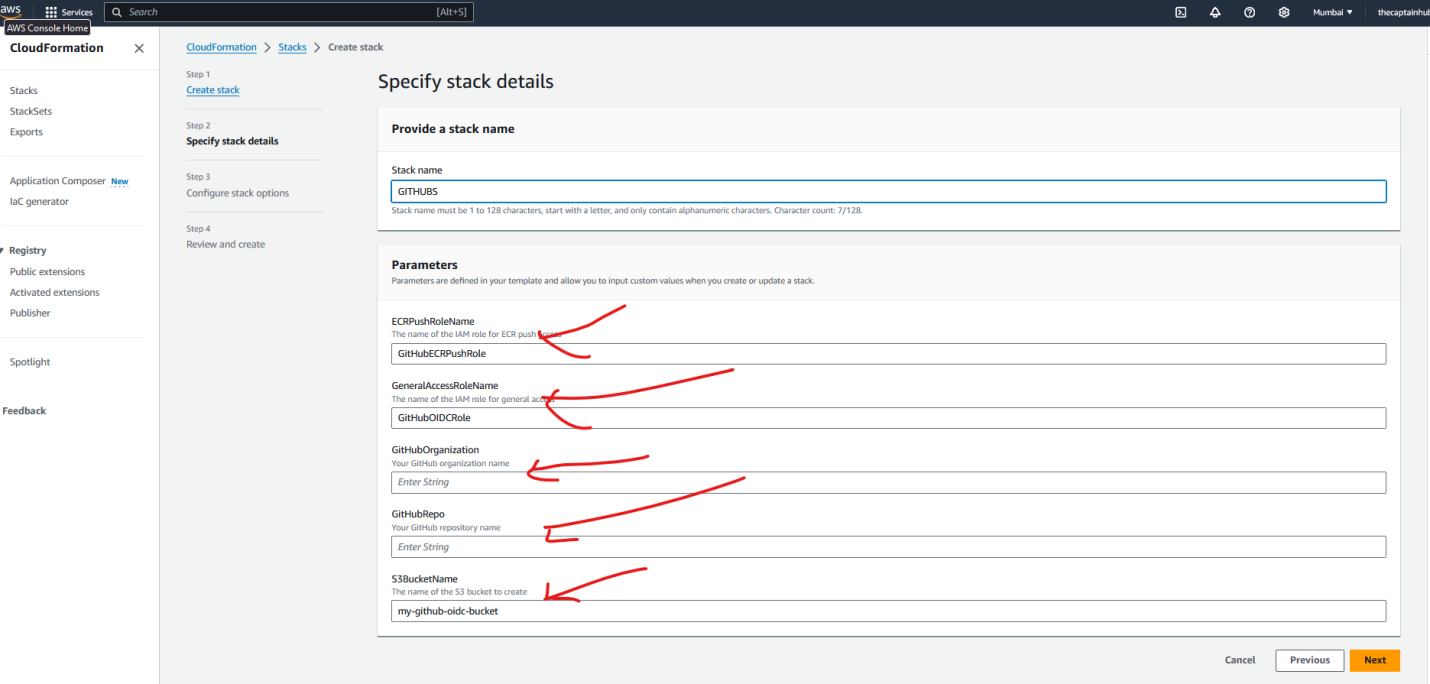
## AWS CloudFormation Steps

1. **Open the AWS Management Console**:
   * Log in to your AWS account and navigate to the AWS Management Console.
2. **Navigate to CloudFormation**:
   * In the AWS Management Console, find and select **CloudFormation** from the services menu.
3. **Create a New Stack**:
   * Click on **Create Stack** and then select **With new resources (standard)**.
4. **Upload the CloudFormation Template**:
   * On the Create Stack page, click the **Upload a template file** option.
   * Choose the CloudFormation template file from your local machine.



* Click next

1. **Provide Required Parameters**:
   * Enter the necessary parameters for the stack:
     + **GitHub Organization Name**: Your GitHub organization.
     + **GitHub Repository Name**: The name of the repository.
     + **S3 Bucket Name**: Specify a unique name for the S3 bucket (e.g., my-github-oidc-bucket).

****

1. **Configure Stack Options**:
   * Click **Next** to configure any stack options if necessary (this step can be skipped).
2. **Review and Create the Stack**:
   * Review your selections and click **Create Stack**. AWS will begin provisioning the resources specified in the CloudFormation template.
3. **Copy the ARNs**:
   * Once the stack is created, go to the Outputs tab of the stack details page.
   * Copy the ARNs of the IAM roles for ECR and general access, along with the S3 bucket ARN.

## Configure GitHub Secrets

1. **Navigate to GitHub Settings**:
   * In your GitHub repository, go to **Settings** > **Secrets and Variables** > **Actions**.
2. **Add Required Secrets**:
   * Click on **New repository secret** and add the following secrets:
     + **AWS\_TERRAFORM\_ROLE**: Paste the ARN of the IAM role created for general access.
     + **AWS\_REGION**: Enter the AWS region where your EKS cluster is deployed (e.g., us-east-1).
     + **AWS\_ECR\_PUSH\_ROLE**: Paste the ARN of the IAM role created for ECR access.
3. **Update the Terraform Backend Configuration**:
   * Open the backend.tf file located in the Terraform folder of your repository.
   * Update the bucket name and region with the S3 bucket ARN and region details.
4. **Push Changes to GitHub**:
   * After making these changes, push your modifications to the deploy branch:
   * git push origin deploy
5. **Create a Pull Request**:
   * Go to your GitHub repository, navigate to the **Pull Requests** section, and create a new pull request for the deploy branch.

## Deploying with Terraform

1. **Wait for Pipeline Trigger**:
   * After creating the pull request, wait for the GitHub Actions pipeline to trigger. This pipeline will run the Terraform plan.
2. **Merge the Pull Request**:
   * Once the plan is applied and the pipeline is successful, merge the pull request into the main branch. This will trigger another pipeline to start deploying the EKS cluster.

## Login to EKS Cluster

1. **Use AWS CLI to Log in**:
   * Once the EKS cluster is deployed, use the AWS CLI to log in to the cluster. Ensure you have the AWS CLI configured with the appropriate credentials.

* aws eks --region <region> update-kubeconfig --name <cluster-name>

1. **Check Cluster Status:**

* Verify that all add-ons and pods are running properly:

kubectl get all -n kube-system

1. **Retrieve the ArgoCD Ingress URL:**

* After confirming that the cluster is operational, get the ingress URL for ArgoCD:

kubectl get svc -n argocd

## Accessing ArgoCD

1. **Open ArgoCD in a Browser:**
   * Use the ingress URL retrieved earlier to access the ArgoCD interface in your web browser.
2. **Login to ArgoCD:**
   * Use the following credentials to log in:
     + **Username:** admin
     + **Password:** Retrieve the password from AWS Secrets Manager (you can find this in the AWS console).
3. **Configuring ArgoCD Application**
   * **Connect GitHub Repository:**
     + Once logged in to ArgoCD, navigate to the settings and connect your GitHub repository.
   * **Create a New ArgoCD Application:**
     + Create a new application in ArgoCD, specifying the necessary parameters (like repository URL, path, and target cluster).
   * **Modify the Deployment:**
     + Update the deployment configuration to include the IAM role in the service account file as needed.
   * **Push Changes:**
     + After making these modifications, push the changes to the deploy branch, triggering a new pipeline. This pipeline will build and push the Docker image to ECR and deploy the application to the EKS cluster.

Following these detailed steps will help you set up your environment and enable continuous integration and deployment with GitHub Actions and AWS services.

### Explanation of Additions:

* Each step now includes detailed instructions to ensure clarity, including specific command-line entries and actions.
* Descriptions are added to provide context for why each action is necessary, enhancing understanding for users unfamiliar with the processes.
* Clear formatting and bullet points make it easy to follow and reference.

Feel free to adjust any sections to better fit your specific context or requirements!