| **CI/CD Pipeline Workflow Documentation****Aim:** To establish a Continuous Integration/Continuous Deployment (CI/CD) pipeline that enables automated code integration, testing, and deployment, using a combination of CloudShell, GitHub, and AWS CodeCommit, involving multiple developers in a collaborative environment. **Objective:**  * To implement a CI/CD pipeline that automates the integration and deployment process, ensuring faster delivery of code changes. * To manage version control with multiple branches, handling commits, pushes, and merges in a team environment. * To utilize AWS CodeCommit for version control and ensure seamless integration with other AWS services in the CI/CD pipeline.  **Prerequisites:**  1. **Development Environment:**    * CloudShell access for all developers, which provides a pre-configured, browser-based command-line environment in AWS. 2. **Version Control Setup:**    * Git should be available by default in CloudShell for managing the repositories.    * GitHub and AWS CodeCommit accounts set up with repositories for code integration. 3. **CI/CD Tools:**    * Access to CI/CD tools and services like AWS CodeBuild, AWS CodeDeploy, or GitHub Actions for automated build and deployment processes.    * Basic understanding of CI/CD processes and the tools being used. 4. **Knowledge of Git:**    * Familiarity with Git commands for cloning, branching, committing, pushing, and merging code.  **Architecture Flow:**  1. **Code Development:**    * Developers use CloudShell to clone repositories from GitHub or AWS CodeCommit, create branches, and develop features.    * Code changes are committed to the feature branches and pushed to the remote repository. 2. **Integration:**    * Upon pushing to the remote repository, the CI/CD pipeline is triggered.    * CodeBuild automatically builds the application and runs tests to validate the changes.    * If tests pass, the code is merged back into the master branch. 3. **Deployment:**    * CodeDeploy is triggered upon successful integration to deploy the application to the designated environment.    * The deployment process is automated, ensuring consistent and error-free releases. 4. **Monitoring:**    * CloudWatch monitors the deployed application, collecting logs and metrics.    * Any issues are reported back to the developers via alerts or dashboards, enabling quick response and resolution.  **Steps:**  1. **Setting Up the CloudShell Environment:**    * Developers access CloudShell from the AWS Management Console.    * Since Git comes pre-installed, developers ensure that their Git configurations (like username and email) are correctly set. 2. **Cloning Repositories:**    * Developers clone the existing repository from GitHub or AWS CodeCommit using HTTPS, SSH, or other methods within the CloudShell environment.    * This allows developers to work on the project using the command-line interface provided by CloudShell. 3. **Branching for Development:**    * Developers create new branches (dev1, dev2) from the master branch to work on different features or tasks.    * Branches allow developers to work in isolation without affecting the main codebase. 4. **Making Code Changes:**    * Developers edit project files, such as index.html, and make necessary updates.    * Code changes are committed to the respective branches with appropriate commit messages (e.g., "1 file changed, 1 insertion, 1 deletion"). 5. **Pushing Code to Remote Repositories:**    * After committing changes locally, developers push their changes to the remote GitHub or AWS CodeCommit repositories from within CloudShell.    * This step integrates the new code into the version control system, making it accessible to other team members and the CI/CD pipeline. 6. **Merging and Integrating Changes:**    * The experienced developer merges the feature branches (dev1, dev2) back into the master branch.    * Merging can be done manually or through pull requests, depending on the version control strategy in use. 7. **Automated Testing and Build:**    * Once the code is merged into the master branch, the CI/CD pipeline triggers automated testing and builds.    * Tools like AWS CodeBuild or GitHub Actions compile the code, run tests, and prepare the application for deployment. 8. **Deployment to Production:**    * After a successful build, the pipeline triggers AWS CodeDeploy or other deployment tools to push the code to production.    * The CI/CD pipeline ensures that new changes are automatically deployed to the production environment, minimizing manual intervention. 9. **Monitoring and Feedback:**    * Post-deployment, the system monitors the application for any issues, providing feedback to the developers.    * Logs, metrics, and alerts help in identifying and resolving issues promptly, ensuring continuous delivery of high-quality software.   **Create a IAM User for Login:-**    **Create a User:-**      Set Permissions:-  Permissions Options**: “Select Attach Policies Directly”**.    Provide the Permission Policies as “**AWSCodeCommitFullAccess**”.    Review and Create:    Click on Create User and the user is created successfully:-    Click on the User Name:-    Click on Security Credentials and scroll down to “**HTTPS Git credentials for AWS CodeCommit (0)”**:-    Click on Generate Credentials:-    Credentials is generated:-    Search for **CodeCommit → Source → Repositories → Create Repository**.      CodeCommit Repository created successfully.    Open CloudShell:-    Run the commands:-  **mkdir CICD-Project**    **ls**    **cd CICD-Project**    When you find the error.      **sudo yum update -y**      **sudo yum install git -y**    **git --version**    Click on “**Clone HTTPS**”:-    **Git clone https://git-codecommit.ap-south-1.amazonaws.com/v1/repos/AmitCodeCommitRepo**    **ls**    cd AmitCodeCommitRepo    **Inside the AmitCodeCommitRepo find any files are stored.**    **vi index.html**    **Inside vi paste the code:-**  **<!DOCTYPE html>**  **<html lang="en">**  **<head>**  **<title>CICD Project with CloudShell</title>**  **<style>**  **body {**  **font-family: Arial, sans-serif;**  **background-color: #f4f4f4;**  **margin: 0;**  **padding: 20px;**  **}**  **.container {**  **max-width: 800px;**  **margin: 0 auto;**  **padding: 20px;**  **background-color: #fff;**  **border-radius: 8px;**  **box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);**  **}**  **h1 {**  **color: #333;**  **}**  **</style>**  **</head>**  **<body>**  **<div class="container">**  **<h1>Hello</h1>**  **</div>**  **</body>**  **</html>**    **ls**    **cat index.html**    **git add .**    **git commit -m "Commit to Master"**    **git config --global user.email "amitkumarnaidu2@gmail.com"**  **git config --global user.name "GitUser-Amit-at-891377318947"**    **git commit -m "Commit to Master"**    **git push origin master**    **After the push command you will find the index.html fille inside the CodeCommit Repository:-**      And I have master branch here.    Creating another branch.  **git branch dev1**    **git checkout dev1**    **git checkout master -- index.html**    ls    **cat index.html** to view whether the code is correctly pulled from Master or not.    vi index.html    Make some changes to the code:-    Check whether the changes are made or not using cat command.  cat index.html    **git add index.html**    **git commit -m "Commit done by dev1"**    **git push origin dev1**    After push command, you will find another branch in the CodeCommit Repository.    The changes made in the vi file are changed in the CodeCommit Repository also.    Create another branch.  git branch dev2    git checkout dev2    ls    cat index.html    git checkout dev1 -- index.html  cat index.html    git checkout master -- index.html    ls    cat index.html    Make changes in the vi file using vi command:-  vi index.html      cat index.html    git add index.html    git commit -m "Commit done by dev2"    git push origin dev2    Another branch is created in CodeCommit Repository.    Changes are done here.    Let’s compare the master, dev1 and dev2 branch codes.  master code    dev1 code    dev2 code    Create pull request from dev2.        Click on Create Pull request.    Pull request is successfully created.    We want to merge the final code to the master code.  Click on merge.    Select necessary details and click on Merge Pull request.    Merge is successfully done.    Let’s Check the master branch, if the code is merged or not.   **Observation:**  * The CI/CD pipeline automates the integration, testing, and deployment of code, significantly speeding up the software development lifecycle. * Branch management and automated testing reduce the likelihood of errors reaching production, improving software reliability. * Collaboration among developers is streamlined, with clear processes for merging and deploying code changes.  **Conclusion:** Implementing a CI/CD pipeline using Git, GitHub, AWS CodeCommit, and associated CI/CD tools via CloudShell facilitates faster, more reliable software development. The automation of code integration, testing, and deployment reduces manual errors, enhances collaboration, and ensures that new features and fixes are delivered to users more rapidly. This workflow is essential for maintaining agility and efficiency in modern software development environments. |
| --- |