|  |  |
| --- | --- |
| **PRE AWS**  **Covers files required for the app and aws requisite setup files** | |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **AWS CODECOMMIT SETUP** | |
| **FUNCTION: *AWS equivalent of GitHub to host your private Git repositories.***  **Reference; https://docs.aws.amazon.com/codecommit/latest/userguide/getting-started.html** | |
| 1. **Setting up GitHub Repo** | 1. Ensure you have a Dockerfile and buildspec.yml created for your app.   *Dockerfile and Buildspec.yml is required for code build* |
| 1. **Setting up AWS CodeCommit IAM User with HTTPs Git Credential for AWS CodeCommit** | 1. Do provide user access to the AWS Management Console (for my case only) 2. Create as an IAM user; **D2B003\_USER\_codecommit** 3. Attach CodeCommit Access Policy to IAM user with: **AWSCodeCommitFullAccess** 4. Login into the created user 5. Click on the created user to see “**Security Credential**” tab 6. Under “**Security Credential**” goto “**HTTPS Git credentials for AWS CodeCommit**” and select “**Generate Credentials**”, Store username/password |
| 1. **Create CodeCommit Repo**   **{not ecr repo!!!!!!}** | 1. Make sure you are in the proper region i.e. us-east-1 2. Switch to CodeCommit 3. Click on “Create Repository” 4. Provide Repository Name and Description(optional): D2B0003\_CODECOMMIT\_repo |
| 1. **Copy GitHub Repo Data to AWS CodeCommit** | 1. **Prerequisites:**     1. ensure git is installed on local computer    2. **You must have an AWS CodeCommit managed policy attached to your IAM user, belong to a CodeStar project team, or have the equivalent permissions see step 2.c** 2. **Clone CodeCommit Repo:** use the CodeCommit repo name from step 3.4 to mirror push “**git clone https://git-codecommit.us-east-1.amazonaws.com/v1/repos/D2B003\_CODECOMMIT\_REPO”** 3. Clone your Github repo to a local folder on your computer 4. Copy contents of Local git repo to the local copy of your AWS CodeCommit Repo **exclusive** of the .git folder 5. “**Git Add .**” followed by ‘**Git commit -m “comment**”’ commands 6. **Push to AWS CodeCommit:** use the CodeCommit repo name from step 3.4 to mirror push “**git push** [**https://git-codecommit.us-east-1.amazonaws.com/v1/repos/D2B\_002\_CODECOMMIT\_repo**](https://git-codecommit.us-east-1.amazonaws.com/v1/repos/D2B_002_CODECOMMIT_repo) |
| **~~Github Repo(Online)>>Github repo(local)>>~~codecommit repo(local)>>codecommit repo(online)** |
| 1. **FURTHER READING/DONT UNDERSTAND** |  |

|  |  |
| --- | --- |
| **AWS CODEBUILD SETUP** | |
| **FUNCTION: *compile your source code and produce artifacts*, docker images and push to ecr/dockerhub or s3** | |
| 1. **Prepare ECR for CodeBuild** | 1. Got to Elastic Container Registry 2. Create ecr repo 3. Settings/Entries  * Visibility Settings: Private/Public * Name: **d2b003\_ecr\_repo**   Leave all other settings as default  **Need to revise this to get implicit implication of all settings!!!!** |
| 1. **Setting up CodeBuild** | 1. Give your project a name and description 2. select the source code location as the newly created AWS CodeCommit repository as above 3. Under the "Environment" section, select the following settings:  * Managed image: Amazon Linux 2 * Privileged: Active * **Please check about the uses of a vpc in codebuild!!!**  1. Add environmental Variables  * # AWS\_DEFAULT\_REGION = your current aws default region ie "us-east-1" * # AWS\_ACCOUNT\_ID = 12 digit Account ID ie "182675479319" * # IMAGE\_TAG = value "latest" * # IMAGE\_REPO\_NAME = name of ECR repo ie d2b\_000\_ecr\_repo.stream   Please note the variables are used in the buildspec.yml to create a new ECR repository based on the image name.   1. Run the CodeBuild build |
| 1. **Additional Steps(Not required in our use-case)** | 1. ~~Unit tests?~~ 2. ~~Artifacts? store the build details in the artifactory with the successful build~~ 3. ~~Using cloudwatch to automate code build~~ |
| 1. **Setup IAM roles and permissions To allow CodeBuild to push Docker images to ECR**   **Note docker image can be pushed to dockerhub instead** | 1. ECR permissions to push Docker images to your repository; add the aws-managed **AmazonEC2ContainerRegistryPowerUser** policy to your AWS CodeCommit created role allowing **CodeBuild** access to ECR   {required to void exit status error on codebuild prebuild stage~~}~~ |
| 1. **FURTHER READING/DONT UNDERSTAND** |  |

**Advanced learning**

1. AWS CDK used in creating CICD pipeline
2. Interview Questions on each module done
3. Certification Questions on each module done

|  |  |
| --- | --- |
| **CODEPIPELINE** | |
|  | |
| 1. **Initialise CodePipeline** | Goto CodePipeline  Select Create Pipeline on the far right |
| 1. **Pipeline Settings** | Under Pipeline settings add the following;   * Pipeline name: second\_pipeline * Execution mode: Queued (Pipeline type V2 required) * Service role: New ServiceRole * Check box for the Allow AWS CodePipeline to create a service role so it can be used with this new pipeline * Leave items under **Variable** as default * Leave items under **Advanced settings** as default   Click Next |
| 1. **Add Source Stage** | Under Source Stage enter/select the following;   * Source Provider: AWS Codecommit * Repository Name: first\_session * Branch Name: master * Change detection options: Amazon CloudWatch Events (recommended) * Output artifact format: CodePipeline default   Click Next |
| 1. **Add Build Stage** | Under Build*– optional* enter/select the following;   * Build provider: CodeBuild * Region: us-east(N. Virginia) * Project name: first\_session\_codebuild\_build * We wont be adding any env. Variables here * Build type: Single Build   Click Next |
| 1. **Add Deploy Stage** | Click Skip deploy stage |
| Review | Click Create Pipeline  For the initial execution |
|  |  |

|  |  |
| --- | --- |
| **BUILDSPEC FILE MODIFICATION** | |
| ***Generate artifact to be able to use code pipeline with code deploy*** | |
| 1. Prepare **BuildSpec file for codedeploy** | Uncomment entries under the following comment text in buildspec.yaml located in local repo  **“# this lines are required for code deploy/codepipeline stage”** |
| 1. Push file to codecommit | Push file to codecommit |
| 1. Check pipeline runs successfully | Look at the pipeline named second\_pipeline and notice it runs automatically once changes are pushed to your codecommit  Proceed to next steps only if pipeline is completely successful |
| 1. Check contents of S3 BuildArtifact folder | Check s3 and look for bucket with prefix  “codepipeline-us-east-1”  Open folder and navigate to the folder named with the pipeline name  It will contain 2 folders   * BuildArtif/ * SourceArti/   Open the buildArtif folder and download the latest zip  Open the zip folder and check content of imagedefinitions.json file in it.  This file will be used when we deploy to ecs using code deploy |

|  |  |
| --- | --- |
| **CODEDEPLOY** | |
| ***Takes docker image created in codebuild stage and deploy to ecs*** | |
| 1. **Make Available ECS service infrastructure** | Use an existing ecs cluster/service/task-definition  Or create your own  Ref: |
| 1. **Create Deploy Stage** | Goto the second\_pipeline and click on edit  Click on add stage button at the very bottom after edit build  Under stage name enter “Deploy”  Click on the “add action group” button in the edit:Deploy box  under edit action enter the following;  Action Name: Deploy  Action provider: Amazon ECS  Region:Us East(N. Virginia)  Input artifacts:BuildArtifact  Cluster Name: second\_ECS\_EC2\_cluster  Service name: second\_ECS\_EC2\_cluster/second\_service  Image definitions file*– optional*: imagedefinitions.json  Click on Done  Save |
| 1. Run the Pipeline by making changes in the local repo and pushing to CodeCommit |  |
| 1. Delete all resources used |  |