|  |  |  |
| --- | --- | --- |
| **AWS SERVICES COVERED BY THIS PROJECT**  **(12 Services)** | 1. IAM 2. VPC 3. CODECOMMIT 4. CODEBUILD 5. CODEDEPLOY 6. CODE PIPELINE | 1. ECR 2. EC2 3. ECS 4. KINESIS 5. LAMBDA 6. S3 |
| **Services required for Data Engr. Associate Certification**  **(11 Focus Areas)**  **(42 Services)** | **Focus Area**   1. **Amazon Athena** 2. **Amazon Redshift** 3. **Amazon QuickSight** 4. **Amazon EMR (Amazon Elastic MapReduce)** 5. **AWS LakeFormation** 6. **AWS EventBridge** 7. **AWS Glue**    * **AWS Glue DataBrew**    * **All AWS Glue features** | **Focus Area:**   1. **Amazon Kinesis**    * **Amazon Kinesis Data Firehose**    * **Amazon Kinesis Data Streams** 2. **Amazon Managed Service for Apache Flink •** 3. **Amazon Managed Streaming for Apache Kafka (Amazon MSK)** 4. **Amazon OpenSearch Service** |
| **Compute:(4 Services)**   1. AWS Batch 2. Amazon EC2 3. AWS Lambda 4. AWS Serverless Application Model (AWS SAM)   **Analytics: (11 Services)**   1. Amazon Athena 2. Amazon EMR 3. AWS Glue 4. AWS Glue DataBrew 5. AWS Lake Formation 6. Amazon Kinesis Data Analytics 7. Amazon Kinesis Data Firehose 8. Amazon Kinesis Data Streams 9. Amazon Managed Streaming for Apache Kafka (Amazon MSK) 10. Amazon OpenSearch Service 11. Amazon QuickSight   **Containers**: **(3 Services)**   1. Amazon Elastic Container Registry (Amazon ECR) 2. Amazon Elastic Container Service (Amazon ECS) 3. Amazon Elastic Kubernetes Service (Amazon EKS)   **Developer Tools: (7 Services)**   1. AWS CLI 2. AWS Cloud9 3. AWS Cloud Development Kit (AWS CDK) 4. AWS CodeBuild 5. AWS CodeCommit 6. AWS CodeDeploy 7. AWS CodePipeline | **Application Integration: (6 Services)**   1. Amazon AppFlow 2. Amazon EventBridge 3. Amazon Managed Workflows for Apache Airflow (Amazon MWAA) 4. Amazon Simple Notification Service (Amazon SNS) 5. Amazon Simple Queue Service (Amazon SQS) 6. AWS Step Functions   **Cloud Financial Management: (2 Services)**   1. AWS Budgets 2. AWS Cost Explorer   **Database**: **(7 Services)**   1. Amazon DocumentDB (with MongoDB compatibility) 2. Amazon DynamoDB 3. Amazon Keyspaces (for Apache Cassandra) 4. Amazon MemoryDB for Redis 5. Amazon Neptune 6. Amazon RDS 7. Amazon Redshift   **Frontend Web and Mobile: (1 Services)**   1. Amazon API Gateway   **Machine Learning: (1 Services)**   1. Amazon SageMaker |
| **Services required for Devops Engr. Professional**  **Certification**  **(15 Focus Areas)**  **(14 Services)** | **Focus Area:**   1. AWS CloudFormation 2. AWS Lambda 3. Amazon EventBridge 4. Amazon CloudWatch Alarms 5. AWS CodePipeline 6. AWS CodeDeploy 7. AWS CodeBuild 8. AWS CodeCommit | **Focus Area:**   1. AWS Config 2. AWS Systems Manager 3. Amazon ECS 4. Amazon Elastic Beanstalk 5. AWS CloudTrail 6. AWS OpsWorks 7. AWS Trusted Advisor |
|  | **Developer Tools: (10 Services)**   1. AWS Cloud Development Kit (AWS CDK) 2. AWS CloudShell 3. AWS CodeArtifact 4. AWS CodeBuild 5. AWS CodeCommit 6. AWS CodeDeploy 7. Amazon CodeGuru 8. AWS CodePipeline 9. AWS CodeStar 10. AWS X-Ray | **SERVERLESS: (4 Services)**   1. AWS Step Functions 2. Amazon EventBridge (Amazon CloudWatch Events) 3. AWS Lambda 4. AWS Serverless Application Model (AWS SAM) |
| Reference: https://dev.to/aws-builders/passing-the-aws-certified-devops-engineer-professional-exam-16ca | **Domain 1 - SDLC Automation (22%)**  **Services in scope**   1. AWS Codepipeline - understand each stage 2. AWS CodeBuild 3. AWS CodeDeploy 4. AWS CodeCommit 5. AWS CodeArtifact 6. Amazon S3 7. Amazon Elastic Container Registry [Amazon ECR] 8. AWS Lambda 9. EC2 Image Builder 10. AWS Codestar 11. AWS Secrets Manager 12. AWS Systems Manager Parameter Store   **Domain 2 - Configuration Management and IaC (17%)**  **Services in scope**   1. AWS Serverless Application Model [AWS SAM] 2. AWS CloudFormation 3. AWS Cloud Development Kit [AWS CDK]) 4. AWS OpsWorks 5. AWS Systems Manager 6. AWS Config 7. AWS AppConfig 8. AWS Service Catalog 9. AWS IAM Identity Centre (formerly known as SSO)   **Domain 3 - Resilient Cloud Solutions (15%)**  **Implement highly available solutions to meet resilience and business requirements**  **Global Scalability**   1. Route 53 2. CloudFront 3. Secrets Manager 4. CloudTrail 5. Security Hub 6. Amazon ECR 7. AWS Transit Gateway 8. AWS IAM | **Domain 4 - Monitoring and Logging (15%)**  **Configure the collection, aggregation, and storage of logs and metrics**   1. CloudWatch 2. Namespaces 3. AWS Logs Log Driver 4. Cloudtrail 5. Cloudwatch logs subscriptions. 6. CloudWatch Events 7. CloudWatch Logs Insights 8. CloudWatch Log Group retention   **Domain 5 - Incident and Event Response (14%)**  **Manage event sources to process, notify, and take action in response to events.**   1. AWS Health 2. CloudTrail 3. EventBridge   **Implement configuration changes in response to events.**   1. AWS Systems manager, 2. AWS Auto Scaling 3. Systems Manager Fleet   **Domain 6 - Security and Compliance (17%)**  **Services in scope**   1. IAM 2. AWS IAM Identity Center 3. Organis 4. Security Hub 5. AWS WAF 6. VPC Flow Logs 7. Certificate Manager 8. AWS Config 9. Amazon Inspector 10. Guardduty 11. Macie |

|  |  |
| --- | --- |
| **PRE PROJECT INTEGRATION/DEPLOY**  **Covers files required for the app and aws requisite setup files** | |
| 1. **Prepare essential project files** | 1. Prepare the producer app 2. Prepare Dockerfile for building the container 3. Prepare the buildspec file for code build to build the docker image |
| 1. **Development Steps** | 1. Run project on local WSL/Linux installations 2. Run on local Docker Installation 3. Run on EC2 Instance 4. Run on ECS Cluster |
| 1. **Sett 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* |

|  |  |
| --- | --- |
| **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 AWS CodeCommit IAM User with HTTPs Git Credential for AWS CodeCommit** | 1. Do not provide user access to the AWS Management Console 2. Create as an IAM user; **D2B\_001\_USER\_codecommit** 3. Attach CodeCommit Access Policy to IAM user with: **AWSCodeCommitFullAccess** 4. 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): **d2b\_001\_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/** **d2b\_001\_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/](https://git-codecommit.us-east-1.amazonaws.com/v1/repos/%20d2b_001_codecommit_repo%20)****[d2b\_001\_codecommit\_repo](https://git-codecommit.us-east-1.amazonaws.com/v1/repos/%20d2b_001_codecommit_repo%20)** |
| 1. **FURTHER READING/DONT UNDERSTAND** | **Codecommit pull requests**  **Codecommit cdk** |
| 1. **Advanced learning** | 1. AWS CDK used in creating CICD pipeline 2. Interview Questions on each module done 3. Certification Questions on each module done 4. Further study/weak areas |

|  |  |
| --- | --- |
| **AWS CODEBUILD SETUP** | |
| **FUNCTION: *compile your source code, run tests 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: **d2b\_001\_ecr\_repo**   Leave all other settings as default |
| 1. **Setting up CodeBuild** | 1. **Under Project Configuration, Enter the following:**   Project name: **d2b\_001\_codebuild**   1. **Under Source/Source 1 - Primary, Enter the following: (when do you use multiple source?)**   Source provider: AWS CodeCommit  Repository: **d2b\_001\_codecommit\_repo**  Reference type: Branch  Branch: Master or main   1. **Under Environment, Enter the following:**   Provisioning model: On-demand  ?Environment image: Managed image  ?Compute: EC2  Operating system: Amazon Linux  Runtime(s): Standard  Image: Pick the latest one  Image version: Pick “Always use the latest image for this runtime version”  Service role: **d2b\_001\_CodeBuild\_ServiceRole**  Role name: Leave blank   1. **Under Additional configuration, Enter the following:**   Timeout: leave as default value  Queued timeout: leave as default value  Privileged: Check “Enable this flag if you want to build Docker images or want your builds to get elevated privileges” button  Report auto-discover: leave unchecked “Disable report auto-discover” button  Auto-discover directory: leave as default value  Certificate: leave as default value  VPC: select **precreated vpc**  Subnets: select private subnets using nat-gateway  Security groups: use created security groups  Click Validate VPC settings  Compute: 3 GB memory, 2 vCPUs   1. Add environmental Variables (these variables are used in buildspec.yaml file)  * # 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" (this is not required)** * # IMAGE\_REPO\_NAME = name of ECR repo ie d2b\_001\_ecr\_repo   **?Read about parameter and secrets manager**  our buildspec.yml will create a new ECR repository based on the image name.  **?Create Parameter button**  Under File systems leave all as default  **?Identifier: Leave as default**  **?Directory path: Leave as default**  **?Mount options: Leave as default**   1. **Under Buildspec, enter the following;**   Select “Use a buildspec file” button   1. **Under Batch configuration, enter the following:**   Leave unchecked the “Define batch configuration” button   1. **Under Artifacts, enter the following:**   Artifact 1 – Primary/Type: No Artifacts  Additional configuration/Encryption key: Leave Blank  **?Cache type**: Leave Blank   1. **Under Logs, enter the following:**   CloudWatch: Check “CloudWatch logs” button  Group name: leave blank  Stream name prefix: leave blank  S3/S3 logs: leave blank   1. Run the CodeBuild 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 CodeBuild created role allowing **CodeBuild** access to ECR   {required to void exit status error on codebuild prebuild stage~~}~~ |
| 1. **FURTHER READING/DONT UNDERSTAND** | Codebuild & lambda  Codebuild unit testing  CodeBuild cacheing  Artifacts? store the build details in the artifactory with the successful build  Using cloudwatch to automate code build  Codebuild cdk  The use of the buttons below  Create Trigger  Clone  Debug Build  Start build with overrides |
| 1. **Advanced learning** | 1. AWS CDK used in creating CICD pipeline 2. Interview Questions on each module done 3. Certification Questions on each module done 4. Further study/weak areas |

|  |  |
| --- | --- |
| **CODEPIPELINE** | |
|  | |
| 1. **Create IAM Role for Code Deploy** | Create an IAM role; **d2b\_001\_role\_codedeploy\_ECS**  Attach ECS Access Policy to IAM role with: **AWSCodeDeployRoleForECS** |
| 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  Under **Deploy*- optional***  CodeDeploy Provider: ECS  Region: US East(N. Virg.)  Input artifacts: **Build Artifact**  Cluster name: first\_session\_ECSCluster  Service name: first-session-service  Image definition file: imagedefinitions.json  DeploymentTimeout: **3**  Configure automatic rollback on stage failure:**Enabled** |
| 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 |  |