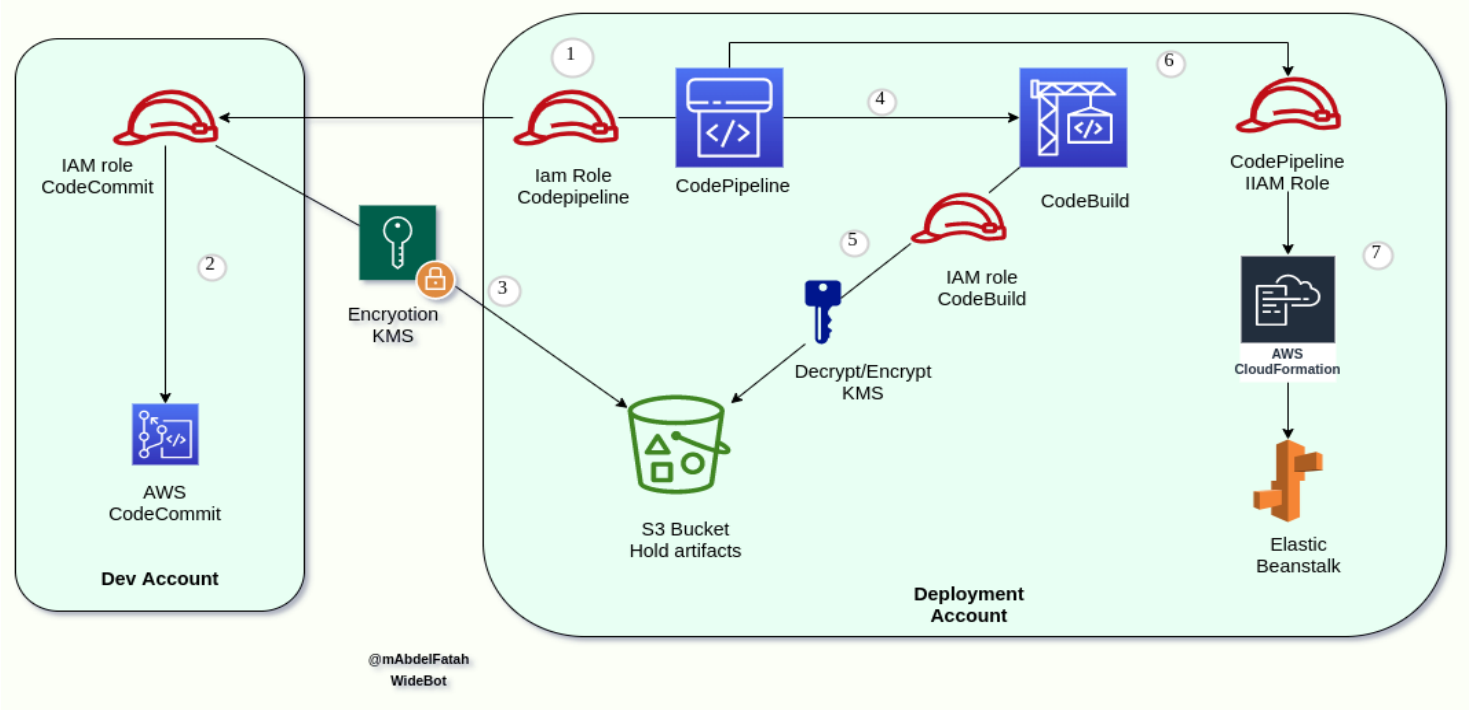
Cross Account CI/CD pipeline

Prerequisites setup:

In order to run scripts with no issues you should have:

- **Aws cliV2** installed: check out this link for cli update.
- (Optional) **Linux bash** - it's required if you want to run the **Single CLick bash script** to do all the stuff on your behalf.
- Clone **WideBot tools Repository** and checkout to Branch of **SAAS-DeploymenAutomation** in order to have access to all required scripts.



Solution:

The identities in our use case are set up as follows:

- **Dev Account:** Developers check the code into an **AWS CodeCommit** repository. It stores all the repositories as a single source of truth for WideBot projects code.
- **Deployment Account:** A central location for all the tools related to WideBot, including **continuous delivery/deployment** services such as **AWS CodePipeline** and **AWS CodeBuild.**

You have two options to apply the solution:

- Use a **SingleClick_BashScript** which applies all the required stuff automatically - requires bash linux shell to run on top.
- Use **Standalone Cloudformation scripts** and execute one by one.

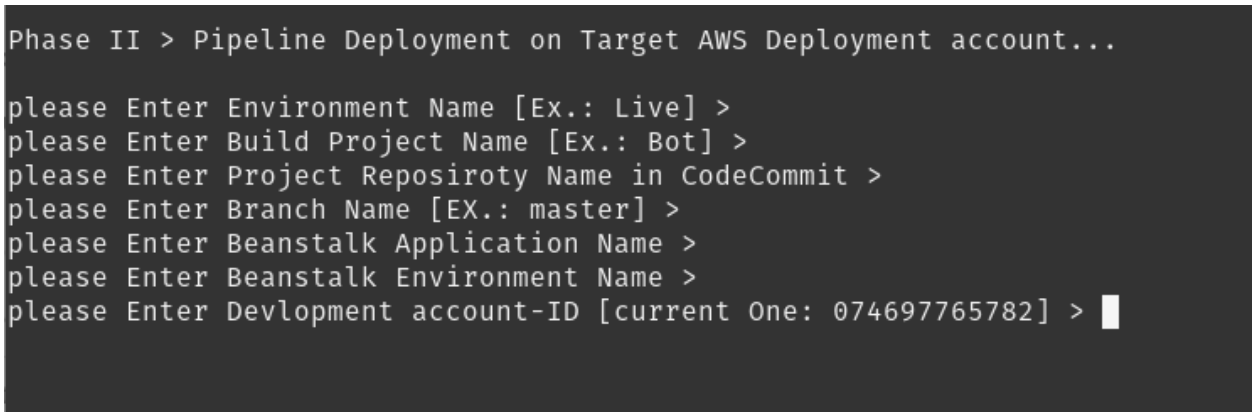
SingleClick Bash Script to Automate Cross-Account CD deployment:

Steps, In your bash shell:

- **Run script**:

  >$ `bash singleClick_CD_pipeline.sh`

- Pass in all the **required parameters** such as, Environment name, codecommit repository, Build Project name, etc..
  - NOte: preferable to pass each parameter as a single word to be more consistent and looks good for our standards of naming.



Standalone Cloudformation Scripts:

Note: if you are gonna proceed manually running the following steps, you should note that:

- The first & second steps already applied to our new and Dev accounts with IDs: **042264081003** & **074697765782**
- In the future when you will use a **new aws account**, you must start from the **first step** normally.
- Follow the steps in the order they're written. Otherwise, the resources might not be created correctly.

Deployment Steps:

1. In the **Deployment account**, deploy this CloudFormation template. It will create:
   a. **the customer master keys** (**CMK**) in AWS Key Management Service (AWS KMS)
      i. also, grant access to Dev & Deployment accounts to use **this key**.
   b. create an **Amazon S3** bucket to hold artifacts from AWS CodePipeline.

```
aws cloudformation deploy --stack-name pre-reqs \
--template-file DeployAcct/pre-reqs.yaml \
--parameter-overrides DevAccount=ENTER_DEV_ACCT \
--profile {awsAccountProfile} --region {awsRegion}
```

Parameters Definition:

stack-name: cloudformation stack name
template-file: path of the cloudformation template file
parameter-overrides: parameters passed to the template file code.
DevAccount: aws account ID which have Codecommit repositories resides in.
profile: if you want to connect to two aws accounts at the same machine, you should create profile for each account

- Navigate to `~/.aws/credentials` file
- append your credentials, here am using two profiles in addition to the default one:

```
[default]
aws_access_key_id = AKIARCZCVOOLIJD2IBXW
aws_secret_access_key = bQqj6n4Whax24MSX7DcOPNdtaJyLVuOip4ePfuV3
[04]
aws_access_key_id = AKIAQTVZCWJVXRMHKEUC
aws_secret_access_key = 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

[test-account]
aws_access_key_id = AKIAQTVZCWJVYOVGMZWL
aws_secret_access_key = 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
```

Why KMS key & S3 bucket:

- By default, **CodePipeline** uses **server-side encryption** with an AWS Key Management Service (AWS **KMS**) to encrypt the **release artifacts** or checked out code, and because other **stages** need to **decrypt** these **artifacts**, you need to create a customer managed CMK in the `Deployment` account in order all stages to have access to.
- **S3 bucket** is being used to either the checked out code from the repository or the artifacts that used by codebuild or other stages.
    - In the next step, we will Configure the bucket to use the customer managed CMK or **KMS key** created in the previous step. This makes sure the objects stored in this bucket are encrypted using that key,

2. In the **Dev account**, which hosts the AWS CodeCommit repository, deploy this CloudFormation template. This template will create:
    a. the **IAM roles**, which will later be assumed by the **pipeline role** running in the **Deployment account**. Enter the AWS account number for the Tools account and the CMK ARN.

```
aws cloudformation deploy --stack-name Deployacct-codepipeline-role \
--template-file DevAccount/toolsacct-codepipeline-codecommit.yaml \
--capabilities CAPABILITY_NAMED_IAM \
--parameter-overrides DdeployAccount=ENTER_Deploy_ACCT \
CMKARN=FROM_1st_Step DeploymentAccount=DEPLOY_ACCT-ID \
--profile {} --region {}
```

Parameters Definition:

Deploy Account: aws account id for the deployment account.

CMKARN: unique arn for the KMS key created in the first step, you can get it from the cloudformation console, in the Output section for the pre-req stack created.

Why Dev Account IAM role:

It will be assumed by codepipeline IAM role in the Deployment account because:

- The **IAM role** created in the **dev** account has permission to **upload** the checked-out code to the **S3 bucket** created in step 1.
- Also, it uses the **KMS key** created in step 1 for server-side encryption.

3. Edit the **params.json** file with all required parameters there, this params will be passed to the pipeline template in the next step.
4. in the **Deployment account**, which hosts AWS CodePipeline, execute this CloudFormation template.
    a. This creates a pipeline, but does not add permissions for the cross accounts (Dev), it will be added in the next step.

```
aws cloudformation deploy --stack-name sample-lambda-pipeline \
--template-file ToolsAcct/code-pipeline.yaml \
--parameter-overrides file://params.json
--profile {} --region {}
```

5. In the **Deployment account,** execute this CloudFormation template which,
    a. gives CodeBuild role created in step 4 the access to KMS key. This role will be assumed by AWS CodeBuild to decrypt artifacts in the S3 bucket.

    This is the same template that was used in step 1, but with different parameters.

```
aws cloudformation deploy --stack-name pre-reqs \
--template-file ToolsAcct/pre-reqs.yaml \
--parameter-overrides CodeBuildCondition=true \
ProjectName=$Build_ProjectName --profile {} --region {}
```

Parameters Definition:

Deploy Account:

CodeBuildCondition: this condition will add the aws codebuild role to the KMS key permissions policy created in the Prereq. Stack, in order codebuild to be able to use that KMS key.

ProjectName: build project name which has been created in the previous step and needs to be allowed in KMS permissions policy using the above condition.

6. In the **Deployment account**, execute this CloudFormation template, which will do the following:
    - Add the IAM role created in step 2. This role is used by AWS CodePipeline in the Tools account for checking out code from the AWS CodeCommit repository in the Dev account.A

```
aws cloudformation deploy --stack-name WB-Live-pipeline \
--template-file DeployAcct/code-pipeline.yaml \
--parameter-overrides CrossAccountCondition=true \
--capabilities CAPABILITY_NAMED_IAM \
--profile {} --region {}
```